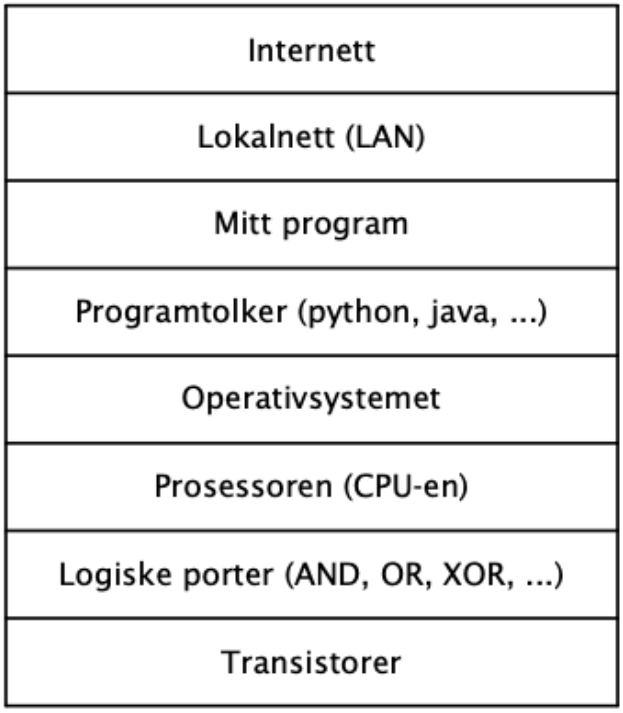
NOTATER – IN1020 – EKSAMEN HØST 2018

HENRIK HEIBERG

«Henrik skildrer datamaskinen historie på vakkert vis…», VG.

«..aldri før har ordene konfidensialitet og informasjonsintegritet blitt brukt på en slik sensuell måte..», Dagbladet.



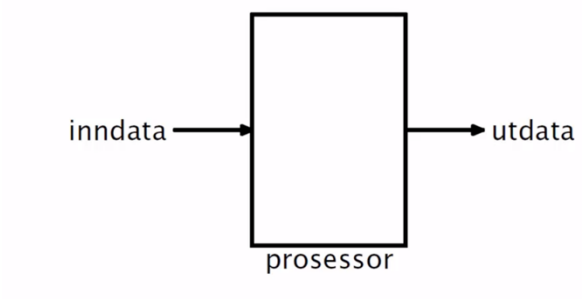
Datamaskinens historie / anatomi.

Hva var den første datamaskinen? Det kommer an på hvordan man definerer en datamaskin

Antikythera-maskinen (100-200 f. Kr)

Kunne sannsynligvis beregne planeters bane, solformørkelse osv.

Vi kan se for oss en datamaskin som dette:



I så fall er Antikythera-maskinen en datamaskin.

Analoge og digitale datamaskiner:

Analog: verdier representeres av fysiske størrelser. Lengde, rotasjon eller spenning

Man lagde etterhvert ganske avanserte analoge maskiner

Eks: bombesikte: beregne ett treffpunkt basert på vind, høyde, hastighet osv.

I 1930-årene hadde UiO den mest avanserte analoge datamaskinen.

Regnestaven var en analog datamaskin. Var i bruk lenge i skolesystemet. Kalkulatoren tok over etter hvert.

Digital: verdier representeres symbolsk. Eks. Bit.

På 60-tallet overtok de digitale datamaskinene.

Mye lettere å sette opp

Mer nøyaktige

Kunne løse flere typer problemer.

Hva gjorde man før man fikk datamaskiner?

Tabeller var utbredt, kunne sette inn kjente tall, «input» og slå opp i tabellen for «output». Tabeller ble laget for å utregne renter, tidevann, astronomi, sinus, logaritmer etc. Disse tabellene ble laget i store fabrikker. De var dyre å lage, kunne ta lang tid å lage hvis nye behov oppstod og de kunne inneholde bevisste og ubevisste feil. Man la f.eks inn feil for å kjenne igjen sitt eget verk dersom konkurrenter kopierte tabellene.

Charles Babbage prøvde å lage «difference engine» på 1800-tallet. Den ble ikke laget ferdig han selv, men ferdigstilt av andre i 1991. maskinen tok input og man måtte manuelt dreie en vinsj rundt mens maskinen kalkulerte og til slutt skrev ut svaret på papir. Senere forsøkte han å lage «analytical engine» en større og forbedre versjon.

Hva er en digital datamaskin?

Dag Langmyhr har tre krav:

1. Generell (kan løse alle typer problemer)
2. Programmerbar (med programmet i minnet)
3. ferdig og tatt i bruk

i tillegg bør den være:

4. binær.
5. elektronisk.

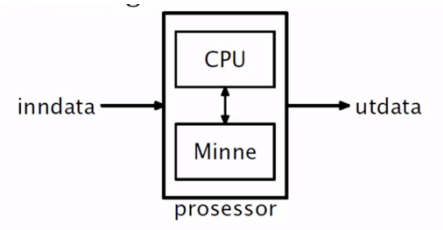
Hullkortmaskiner:

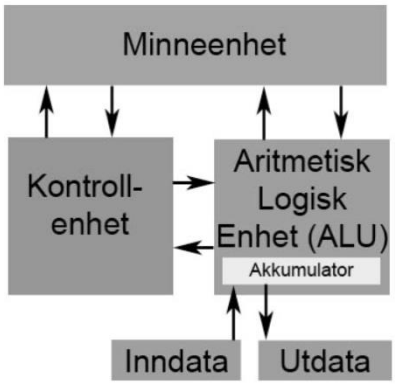
Skulle hjelpe til med folketelling i USA.

Var i bruk.

To person betød spesielt mye: John von Neumann og Alan Turing. De definerte det vi i dag kaller Von Neumann-arkitektur:

> Von-Neumann arkitektur er en måte å designe datamaskiner hvor både instruksjoner og data ligger i samme minneenhet. Prosessoren er separat fra minneenheten. Arkitekturen skulle bestå av følgende deler:
> Et arbeidalager som skulle inneholde data og instruksjoner på binær form.
> En ALU (aritmetisk logisk enhet) som kunne utføre enkle matematiske og logiske operasjoner
> En kontrollenhet som tolket instruksjonene i minnet og sørget for at de ble utført.
> Inn/ut enheter som sørger for kommunikasjon mellom bruker og kontrollenhet.





Alle datamaskiner bruker denne arkitekturen i dag.

Eniac:
- Laget i USA for å beregne ballistiske tabeller.
- Generell, i bruk, elektronisk.

ABC:
- Laget av John Atanasoff rundt 1940.
- Binær, elektronisk.

Harvard mark 1:
- Første maskinen i salg
- I bruk
- Funnet møl i maskinen som førte til feil, derav «bug»

Colossus:
- Bygget under krigen for å knekke tyske koder
- I bruk, binær, elektronisk

Zuse z4:
- Generell, i bruk, binær

Manchester SSEM («baby»)
- Kanskje første datamaskin
- Generell, programmerbar i Ram, binær, elektronisk, i bruk?

EDSAC:
- Dag mener dette er den første datamaskinen. Bygget 6. mai 1949.
- Fyller alle fem kriteriene.
- Brukte forsinkelsesminne:
  - Tok et langt rør med kvikksølv. Lyd brukte litt tid på å sendes igjennom røret, og man Kunne derfor lagre bits.

Det er vanlig å dele datamaskinene inn i generasjoner:

	År	**Teknologi**	**Størrelse**	**Instr/sek**	**Pris** (2017-kr)
1	1945-60	Radiorør	$10m^3$	2000	60 mill
2	1960-68	Transistorer	$650dm^3$	500 000	50 mill
3	1968-78	IC	$80dm^3$	300 000	200 000
4	1978-??	LSI og VLSI	$0{,}1\text{-}20dm^3$	10^9	1-10 000

Vi er fremdeles på nivå 4. Hva skjer videre?

Kvantedatamaskiner? (bitene kan ha ulike verdier samtidig)

Biologiske datamaskiner?

Digital representasjon

Alt som lagres og gjøres i en datamaskin er lagret eller gjøres som bit. Et bit er et begrep for noe som har to tilstander, 0 eller 1, false eller true, strøm på eller strøm av.

Tekst:

Sett opp en tabell over tegn vi trenger og gi hvert tegn et nummer. Problemet er at alle kanskje ikke bruker samme tabell.

ASCII – første vellykkede forsøk på standardisering i 1963. Den har 128 tegn.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

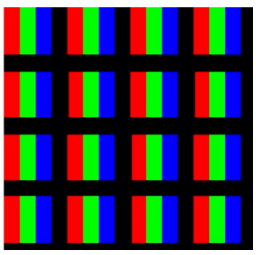
ASCII-tabellen hadde mangler. Den hadde f.eks ikke æ, ø og å, og andre bokstaver og tegn.
Latin-1 var en bedre løsning. Den hadde 256 tegn. De 128 første tegnene var like ASCII.
Likevel endte man opp med at ulike deler av verden trengte sine egne tabeller for å representere de tegn som forekom unikt der.
Løsningen var derfor å lage en tegnkoding/tabell som omfattet alle skriftspråk i verden.

- Den heter Unicode og er nå stort sett ferdig.
- Den har plass til ca. 1 000 000 tegn.
- Litt over 137 000 tegn er foreløpig definer.
- Det meste av nyere programvare støtter Unicode.
- Problemet med Unicode er at det tar stor plass å skulle kunne lagre så mange tegn. Det finnes derfor komprimerte versjoner av Unicode.

UTF-8 er en komprimert form av Unicode. Den bruker fra 1 til 4 byte.

Bilder:

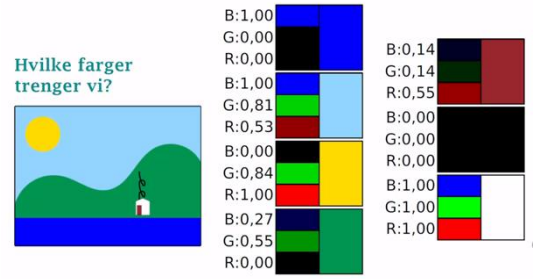
Bilder lagres også som bit i en datamaskin.
På en bildeskjerm består hvert bildepunkt (pixel, se tegning til høyre) av tre farger: rød, grønn og blå. De kan lyse sterkt eller svakt.



Man lager et rutenett på bildet og noterer mengden RGB i hver rute. Hvis
Vi bruker i byte til hver fargemengde får vi:

135 206 255 (lyseblåfargen øverst til venstre)
...

0 0 255 (mørkeblå fargen nederst til høyre)



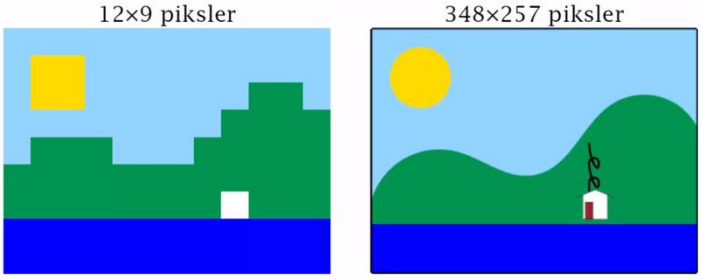
Antall piksler har mye å si ved lagring av bilder. En piksel kan kun representere én farge. Ved få piksler vil derfor bildet være uklart og kornete, men tar liten lagringsplass.

Kan bruke en fargetabell for å lagre bildet på mindre plass. Vi bruker 3 byte = 24 bit til hvert piksel, men vi har bare 7 ulike farger på eksempelbildet vårt. Da kan vi sette opp en tabell:

0	0 0 255	mørkeblå
1	135 206 255	lyseblå
2	255 214 0	gul
3	0 140 69	skoggrønn
4	140 36 36	brun
5	0 0 0	svart
6	1 1 1	hvit

Da trenger vi bare 3 bit per piksel. Det er fordi tallene fra 0-6 kan representeres med 3 bit:

000
001
010
011
100
101
110



Hvis vi bruker en fargetabell til å lagre bildet til høyre på 363 682 byte, kan det komprimeres til 45 483 byte.

Hvis det er mange like piksler på rad, kan vi lagre fargen og hvor mange piksler på rad som kommer med samme fargen. Png og Gif bruker slike teknikker. Størrelsen på bildet til høyre kan da komprimeres ytterligere fra 45 483 byte til 2 917 byte.

Trenger vi å spare så mye plass? Ja, bilder sendes da mye raskere over internett og tar mindre lagringsplass.

Fotografier er litt annerledes. På fotografier er det få brå overganger. Vi mennesker kan bare skjelne et begrenset antall nyanser. Vi søker automatisk etter mønstre.

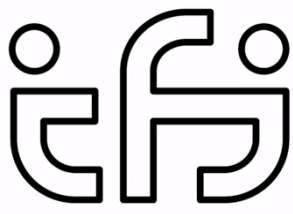
JPEG-formatet lager en komprimert versjon av bildet, og senker egentlig kvaliteten. Dette er komprimering med tap. Bildet kan ikke gjenopprettes til original. Til sammenligning er PNG og GIF sin komprimeringsteknikker ikke komprimering med tap.

Vektorgrafikk. Det er mulig å lagre bilder som linjer og kurver. Man lagrer egentlig bildet som en geometrisk beskrivelse. Forstørring og forminsking av bildet er lett.

```
   :
newpath
38.4094 2.83464 moveto
38.4094 15.44878 lineto
28.3464 15.44878 lineto
21.03242 15.44878 14.93857
21.1286 14.93857 28.3464 curveto
14.93857 35.77315 lineto
36.56685 35.77315 lineto
36.56685 48.38728 lineto
2.32443 48.38728 lineto
2.32443 28.3464 lineto
2.32443 14.16295 14.0667
2.83464 28.3464 2.83464 curveto
closepath stroke
   :
```

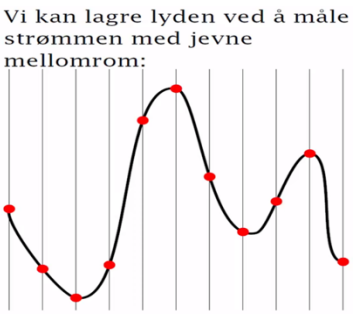
Anbefaling:

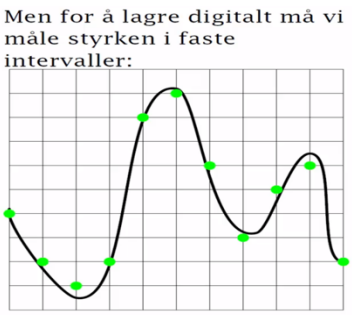
bruk vektorgrafikk om mulig: SVG, EPS, PDF.
For fotografier bruk JPEG i så god kvalitet som mulig.
Et nytt format HEIF/HEIC vil kanskje ta over for JPEG.
Dette er et bedre format.
For annen rastergrafikk bruk PNG med så mange piksler som mulig.



Lyd:

Lyd er bølger i luft. De kan overføres som strøm. Vi kan lagre lyden ved å måle strømmen med jevne mellomrom.

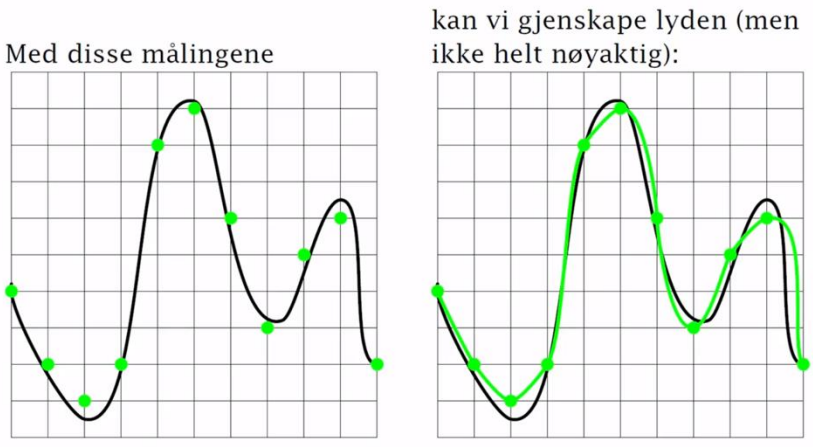




Vi får ikke heeelt nøyaktig gjengivelse av lydbølgene.

Kvaliteten av lyden blir da avhengig av hvor ofte vi måler og hvor mange trinn vi benytter til målingen.

CD: en vanlig CD har 74 minutter spilletid med god lyd. 44 100 målinger per sekund. 65 536 intervaller (2 byte). 2 kanaler (høyre og venstre). En CD må derfor ha plass til 783 MB.



Mulig å spare plass? Høyre og venstre kanal er stort sett like. Det er derfor lurt å lagre bare venstre side, samt forskjellene. Mennesker kan heller ikke høre lyder under 20hz eller over 20 000 hz. Disse lydene kan derfor utelukkes. Om vi hører en kraftig lyd med en frekvens, hører vi ikke en litt svakere lyder med noe høyere frekvens.

Ekte CD	1410 kb/s	
MP2	ca 256 kb/s	
MP3	ca 160 kb/s	
AAC	ca 128 kb/s	(Youtube, iTunes..)

Kvaliteten går ned om det komprimeres mer enn dette.

Binære tall

Å telle med binære tall er som å late som at man bare har to fingre. Vi gjør som i 10-tallssystemet. Vi øker alltid siste tall med 1, hvis vi ikke har flere siffer, så setter vi tallet til 0 og gjentar for sifferet til venstre.

Binært	0	1	10	11	100	101	110	111	1000	1001	1010	1011
Desimalt	0	1	2	3	4	5	6	7	8	9	10	11

Notasjon

Hvis det mulighet for tvil, skriver vi binære tall som 1001_2 og desimale tall som 1001_{10}.

Et matematisk blikk

I det desimale tallsystemet har posisjonene vekt $1, 10, 100, 1000, \ldots = 10^0, 10^1, 10^2, 10^3, \ldots$

(Notasjonen a^n («a i n-te») betyr $\underbrace{a \times a \times \cdots \times a}_{n \text{ ganger}}$.)

I det binære tallsystemet har posisjonene vekt $1, 2, 4, 8, \ldots = 2^0, 2^1, 2^2, 2^3, \ldots$

	8	4	2	1	
	2^3	2^2	2^1	2^0	
	↓	↓	↓	↓	
1011_2 =	1	0	1	1	$= 11_{10}$

Desimal til binært:

Vi deler på 2 og ser på «resten»

Verdi	Rest
25	1
12	0
6	0
3	1
1	1

Svaret blir altså 11001

Binært til desimal:

La oss regne om 11001_2 til desimalt.

1. Skriv binærtallet *oppover* (med siste siffer øverst).
2. Skriv 2-erpotensene nedover til høyre for streken.
3. Stryk alle verdiene der binærtallet har siffer 0.
4. Legg sammen de verdiene som er igjen.

1	1
0	~~2~~
0	~~4~~
1	8
1	16
	25

De fleste kalkulatorer (i hvert fall på datamaskiner) kan konvertere fra binært til desimal og vice versa.

Heksadesimal notasjon:

Binære tall blir fort ekstremt lange/store. Vi erstatter derfor fire og fire binære sifre med heksadesimale sifte 0x0-0xF:

0000	0001	0010	0011	0100	0101	0110	0111
0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
1000	1001	1010	1011	1100	1101	1110	1111
0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF

$1000000_{10} = 1111\,0100\,0010\,0100\,0000_2$ (4 2 4 0)

= 0xF4240

Oktal notasjon:

En grunn til at hex-notasjonen har overtatt er at en byte er 8 bit, noe som passer fint til hex-notasjonen sammenlignet med oktal-notasjon.

Tidligere brukte man ofte **oktal** notasjon der man slår sammen tre og tre bit:

$$\underbrace{111}_{07}\ \underbrace{110}_{06}\ \underbrace{101}_{05}\ \underbrace{100}_{04}\ \underbrace{011}_{03}\ \underbrace{010}_{02}\ \underbrace{001}_{01}\ \underbrace{000}_{00}$$

(I Java skriver man oktale tall som ***0nnn***; i Python 2 skriver man dem også som ***0nnn***, men i Python 3 bruker man ***0onnn***.)

I dag har hex-notasjonen overtatt.

En bite er 8 bit.

Vi kan ikke hente ut 1 og 1 bit. I stedet lagrer vi dem som byte (altså grupper av 8 bit)

0	0	0	0	0	0	0	0	$= 0_{10}$
0	0	0	0	0	0	0	1	$= 1_{10}$
0	0	0	0	0	0	1	0	$= 2_{10}$
⋮								
0	1	1	1	1	1	1	0	$= 126_{10}$
0	1	1	1	1	1	1	1	$= 127_{10} = 2^7 - 1$

I en byte kan vi telle fra 0-127. altså 128 unike tall.
Grunnen til at første bit «ikke brukes», er fordi den representerer om tallet er negativt eller positivt. Vi kaller dette en fortegnsbit og denne har verdien -128.

1	1	1	1	1	1	1	1	$= -1_{10} = -128 + 127$
1	1	1	1	1	1	1	0	$= -2_{10}$
1	1	1	1	1	1	0	1	$= -3_{10}$
⋮								
1	0	0	0	0	0	0	1	$= -127_{10}$
1	0	0	0	0	0	0	0	$= -128_{10} = -128 + 0$

En byte kan derfor dekke alle tall fra -128 til +127.

Java tilbyr disse tallene:

byte	8 bit	−128 - +127
short	16 bit	−32 768 - +32 767
int	32 bit	−2 147 483 648 - +2 147 483 647
long	64 bit	−9 223 372 036 854 775 808 - +9 223 372 036 854 775 807

I Java og de fleste andre programmeringsspråk har vi disse flyttallene:

			Minste	Største
float	32 bit	7 sifre	$1{,}2 \cdot 10^{-38}$	$3{,}4 \cdot 10^{38}$
double	64 bit	16 sifre	$2{,}2 \cdot 10^{-308}$	$1{,}8 \cdot 10^{308}$

Flyttall lagrer tallet i to sekvenser, tallet og potensen. Da kan vi lagre enormt store tall.

Boolsk algebra og logiske porter

1 bit kan representere 2 tall.
2 bit kan representere 4 tall.
3 bit kan representere 8 tall.
4 bit kan representere 16 tall.

I boolsk algebra regner vi med 1 og 0. true eller false.

Definerte basis operasjoner:

AND	" • "	" ∧ "	
OR	" + "	" ∨ "	
NOT	" ´ "	" ¬ "	" ¯ "

Sannhetstabell

AND

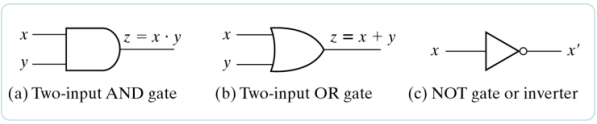
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

OR

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

NOT

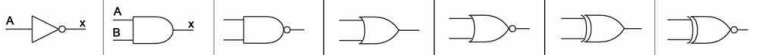
X	Y
0	1
1	0



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

Operasjonene over er de tre basisreglene eller basisoperasjonene/basisportene. Med disse tre portene kan man i teorien implementere en hvilken som helst datamaskin.

Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A X: 0 1; 1 0	B A X: 0 0 0; 0 1 0; 1 0 0; 1 1 1	B A X: 0 0 1; 0 1 1; 1 0 1; 1 1 0	B A X: 0 0 0; 0 1 1; 1 0 1; 1 1 1	B A X: 0 0 1; 0 1 0; 1 0 0; 1 1 0	B A X: 0 0 0; 0 1 1; 1 0 1; 1 1 0	B A X: 0 0 1; 0 1 0; 1 0 0; 1 1 1

NAND = a' + b'
a eller b må være 0
NOR = a'b'
Både a og b må være 0
XOR = a'b + ab'
A må være 0 og b være 0, eller a er 1 og b 0.
XNOR = a'b' + ab
A og b er begge 0 eller a og b er begge 1

Huntington postulater

(P0)	Mengden {0,1} er lukket under "+" og " • "		lukket
(P1)	x + x = x	x • x = x	
(P2)	x + 0 = x	x • 1 = x	ident.el.
(P2b)	x + 1 = 1	x • 0 = 0	
(P5)	x + x´ = 1	x • x´= 0	komplem.
(P6)	0 ≠ 1		minst 2 el.
(P3)	x + y = y + x	x • y = y • x	kommutativ
(P4)	x•(y + z) = x•y + x•z	x+(y•z) = (x+y) •(x+z)	distributiv
(P5)	(x´)´ = x		

Boolske funksjoner kan representeres enten som en sannhetsverditabell eller som et funksjonsuttrykk. Et funksjonsuttrykk viser kun når funksjonen er sann (se NAND, NOR osv over).

Funksjonsutrykk kan forenkles ved å følge f.eks. reglene til høyre:

Dualitet for postulatene:
Kan bytte "•" med + hvis man bytter "0" med "1"

Presedens:
Først utføres "0", så "´", så "•" og til slutt "+"

Eksempel:	Eksempel:	Eksempel:
F = x(x'+y)	F = x+x'y	F = (x+y)(x+y')
F = xx'+xy	F = (x+x')(x+y)	F = x + (yy')
F = 0+xy	F = 1(x+y)	F = x + 0
F = xy	F = x+y	F = x

Minterm:

I en funksjon kan x være enten x eller x', altså sann eller ikke sann. En funksjon kan være gitt på «sum av produkt» form. Eksempel: F = xy + xy' + x.

Hvert produktledd som inneholder alle variablene kalles en minterm. For to variabler finnes det 4 forskjellige mintermer:

xy + xy' + x'y + x'y'

for 3 variable finnes det 2^3 forskjellige mintermer.

Maksterm:

En funksjon kan være gitt på «produkt av sum» form. Eksempel: F = (x+y)(x+y')y

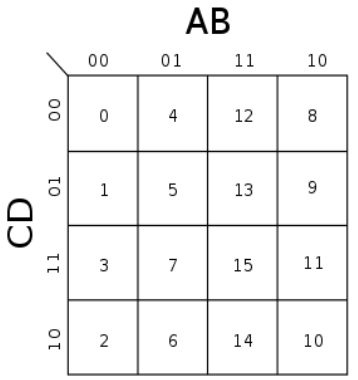
Hvert summeledd som inneholder alle variablene kalles en maksterm. For to variable finnes det 4 forskjellige Makstermer:

(x + y) (x + y') (z' + y) (x' + y')

for 3 variable finnes det 2^3 forskjellige makstermer.

Karnaughdiagram:

Et karnaughdiagram er en metode for forenkling av boolske utrykk. Uttrykket må være representert ved sum av mintermer. Metoden egner seg for funksjoner med 2-4 variable. Til høyre er et diagram med 4 variable.



ABCD	ABCD
0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - 10
0011 - 3	1011 - 11
0100 - 4	1100 - 12
0101 - 5	1101 - 13
0110 - 6	1110 - 14
0111 - 7	1111 - 15

OR funksjonen kan skrives som:

F = ab + a'b + ab'

Dette tilsvarer minterm3, minterm1 og minterm 2.

Minterm 0 = 00 = a'b'

Minterm 1 = 01 = a'b

Minterm 2 = 10 = ab'

Minterm 3 = 11 = ab

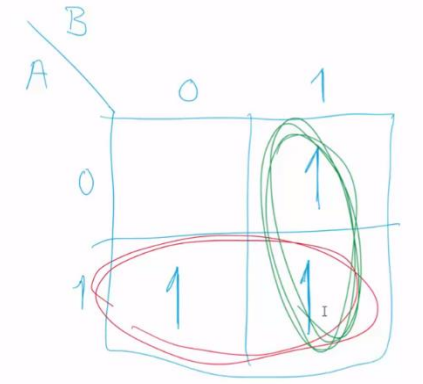
Vi kan nå slå sammen horisontale eller vertikale naboer som går opp i toerpotens, altså 1,2,4,8,16 naboer.

Som vi ser på figuren til høyre kan vi slå sammen minterm 2 og minterm 3 (2 naboer). Disse representerer ab' og ab, noe som kan forkortes til a.

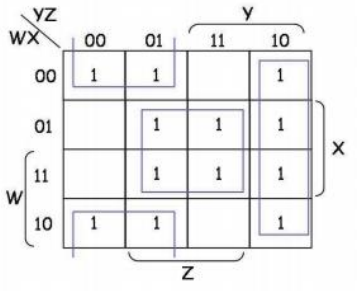
Vi kan også slå sammen minterm 1 og minterm 3 (2 naboer). De representerer a'b og ab, som kan forkortes til b.

Hele uttrykket kan derfor forkortes til a + b.

F = a + b

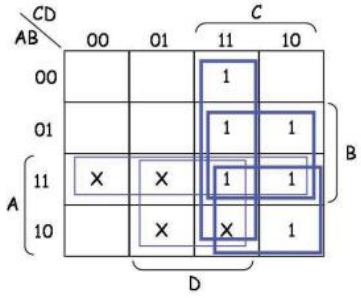


c) F(W,X,Y,Z) = Sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 14, 15) => $F = \bar{X}\bar{Y} + XZ + Y\bar{Z}$



12) Forenkle følgende funksjoner F med tilhørende "don´t care"

a) F(A,B,C,D) = Sum m(3, 6, 7, 10, 14, 15), don't-care: d(A,B,C,D) = Sum m(9, 11, 12, 13)



$F = CD + BC + AC$

Addisjonskretser- halvadder og fulladder

Binære addere er en av de mest brukte digitale kretsene. De finnes i blant annet i ALU-en i mikroprosessorer (CPU) og er basis for addisjon, subtraksjon, multiplikasjon, divisjon og mange andre matematiske operasjoner. I utgangspunktet er binær addisjon identisk med desimal addisjon.



Til høyre legger vi sammen de binære tallene 00101 og 01101 eller 5 og 13 i titallssystemet. Det sentrale er egentlig hvordan vi håndterer mente.

Eksempel: Legger sammen A=5 og B=13:

MENTE		1	1	0	1	
	Tallverdi	Bit 4 16	Bit 3 8	Bit 2 4	Bit 1 2	Bit 0 1
A	5	0	0	1	0	1
B	13	0	1	1	0	1
SUM	18	1	0	0	1	0
		Fulladder	Fulladder	Fulladder	Fulladder	Halvadder

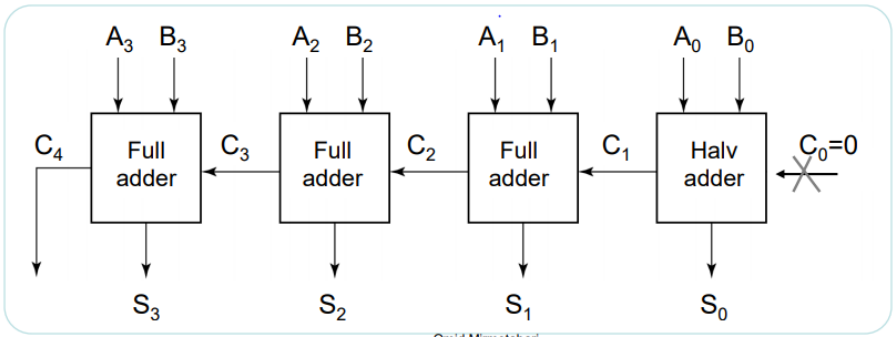
For å lage en addisjonskrets som kan plusse sammen to binære tall, trenger vi to elementer: halvadder og fulladder.

Halvadder:

Oppgaven til halvadderen er kun å addere sammen de to minst signifikante bitene. A0 og B0. Den har altså 2 inputs: A0 og B0. Den har to outputs: S0 som er den totale summens første bit og C som er carry, eller mente.

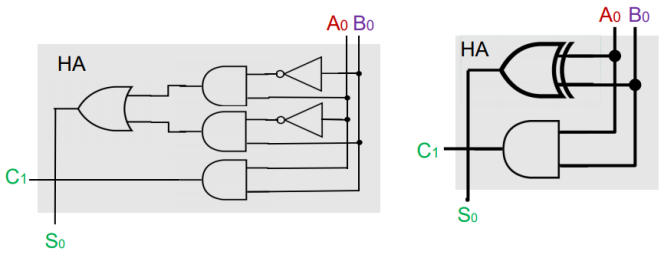
Halvadder: Tar ikke mente inn

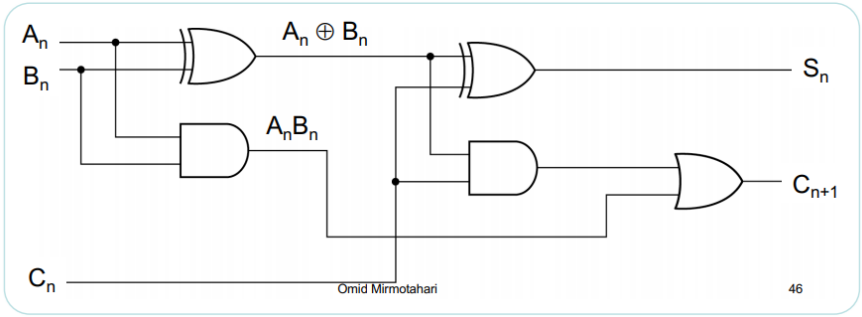
Fulladder: Tar mente inn



Fulladder:

Fulladderens oppgave er å addere sammen An med Bn, n står for det bitet vi er på (som ikke kan være 0 for akkurat de bitene er halvadderens oppgave å legge sammen), og C. C er carry eller mente. Den har altså tre inputs. Den har to outputs, Bn og Cn.





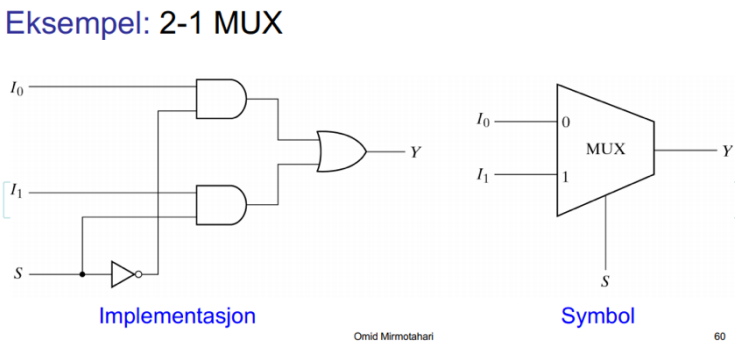
Kombinatoriske og sekvensielle kretser

Kombinatorisk logikk: utgangsverdiene er entydig bestemt/gitt av nåværende kombinasjon av inngangsverdier.

Multiplekser:

En multiplekser velger hvilke innganger som slippes ut. Det er styringssignalet S som bestemmer dette. Hvis vi har mange innganger på styringssignalet være på flere bit.

Vi bruker multipleksere for å kunne ta valg inne i en datamaskin. Det er en god måte å styre ting på.



Demultiplekser:

Motsatte av en multiplekser. En inngang og flere utganger.

Komparator:

Krets som sammenligner binære tall, om de er like eller ulike (større eller mindre enn). 2 innganger, to tall, og 3 utganger, like store, større eller mindre.

Dekoder:

Dekomponerer signal. Tar inn et binært ord og gir ut alle mintermer.
3 bit inn 8 bit ut.
4 bit inn 16 bit ut.

Hvis dekoderen har tre innganger, som eksempelet til høyre, z, y og z. Så har den åtte utganger. Minterm0, minterm1 osv.

Eksempel: 3bit inn

Innganger			Utganger							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Enkoder:

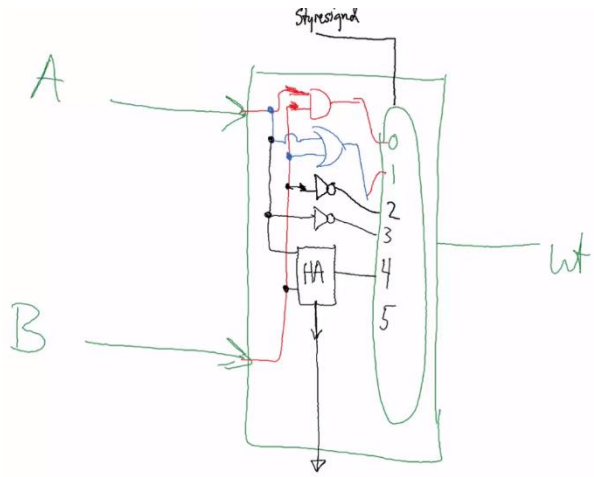
Motsatte av dekoder.

ALU:

Aritmetisk logisk enhet. Gjør aritmetiske operasjoner (simpel matte) og logiske operasjoner (not, and, or osv.) Viktigste del av CPU. Stort sett adderer den hele tiden.

En ALU trenger blant annet fulladder, multiplekser, and, or og not. Moderne ALU-er kan utføre hundrevis av operasjoner.
Til høyre er et eksempel på en 1-bits ALU.

Den har input A og B, et styringssignal til multiplekseren og et utsignal for carry, og en output.



Sekvensiell logikk: inneholder hukommelse (låsekretser). Utgangsverdiene er gitt av nåværende kombinasjon av inngangsverdier, samt sekvensen (tidligere inngangs-/utgangsverdier). Utgangen er ikke avhengig kun av inngangene, men OGSÅ av minneelement, eller tidligere innganger.

Minneelement:

Måte å lagre en bit

Klokke:

CPU har en klokkefrekvens, GHz. Klokkefrekvensen styrer CPU som et trafikklys.
Ved 1 GHz går en klokkeperiode på 1 nanosekund.
En klokkeperiode består av høy og lav, 1 og 0.

Minneelement og klokkefrekvens sørger for at vi kan synkronisere. Når klokken er 1 kan man lese til minnet, når den er 0 kan man hente fra minnet.

Tilstandsmaskin:

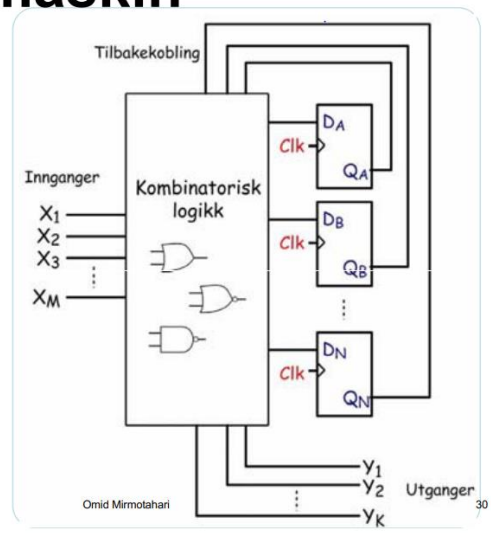
En tilstandsmaskin er et sekvensielt system som gjennomløper et sett med tilstander styrt av verdiene på inngangssignalene.

Tilstandsmaskin

Generell tilstands-maskin basert på D flip-flops

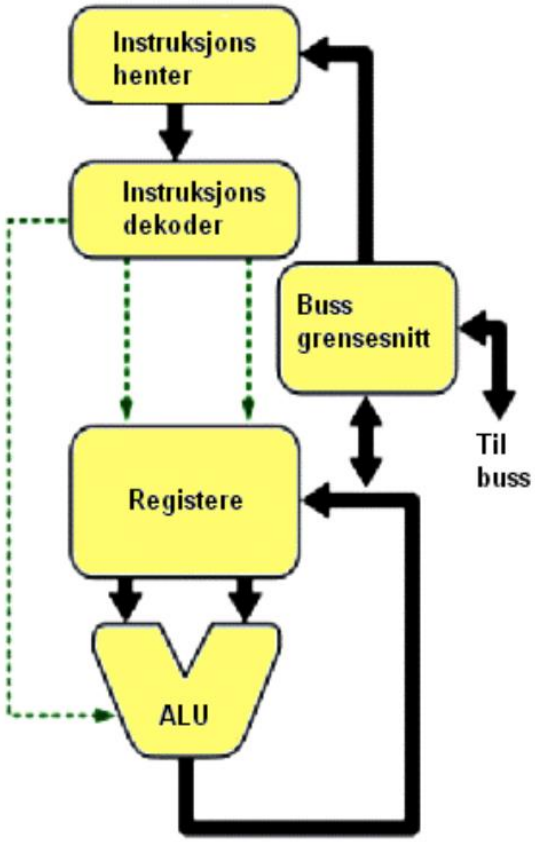
N-stk flip-flops gir 2^N forskjellige tilstander

Utgangssignalene er en funksjon av nåværende tilstand pluss evt. inngangsverdier



Prosessoren – Oversikt

Prosessoren er datamaskinens hjerne. I en PC består prosessoren av en enkelt integrert krets og kalles en mikroprosessor. Verdens første mikroprosessor ble lansert av Intel i 1971, og Intel har siden da forsynt markedet med stadig nye generasjoner mikroprosessorer. Fra 1971 til 2008 har antallet transistorer i mikroprosessoren økt fra ca. 2300 til 2 milliarder, det vil si at antallet er fordoblet omlag 19,5 ganger. I følge Moores lov skal en slik økning i kompleksitet ta ca. 19,5 * 2 = 39 år, noe som ikke er langt unna tidsrommet fra 1971 til 2008.
Prosessorens hovedkomponenter:



ALU:
Arithmetic Logical Unit. Utfører all elementær Aritmetikk, eller avgjør om en logisk betingelse er sann Eller usann. Aritmetikk defineres som regel som de elementære operasjonene vi har på tall: addisjon, subtraksjon, multiplikasjon og divisjon.

Registre:
Registrene er små, lynraske minneceller, som en moderne prosessor har noen hundre av. Verdiene som ALU opererer på ligger i registre, dit legges resultatene som ALU produserer, og ALU bruker dem til mellomlagringer. Instruksjonsregistret inneholder den instruksjonen som ALU for øyeblikket arbeider med, og instruksjonspekeren er et register som forteller hvor i internminnet neste instruksjon skal hentes.

Bussgrensesnitt:
Bussgrensesnittet håndterer prosessorens kommunikasjon med omverdenen.

Instruksjonshenter:
Instruksjonshenteren sørger for å, via bussgrensesnittet, hente neste instruksjon som skal utføres, og overlate den til instruksjonsdekoderen.

Instruksjonsdekoder:
Instruksjonsdekoderen dekoder instruksjonen som ALU skal utføre, og plasserer resultatet i registre.

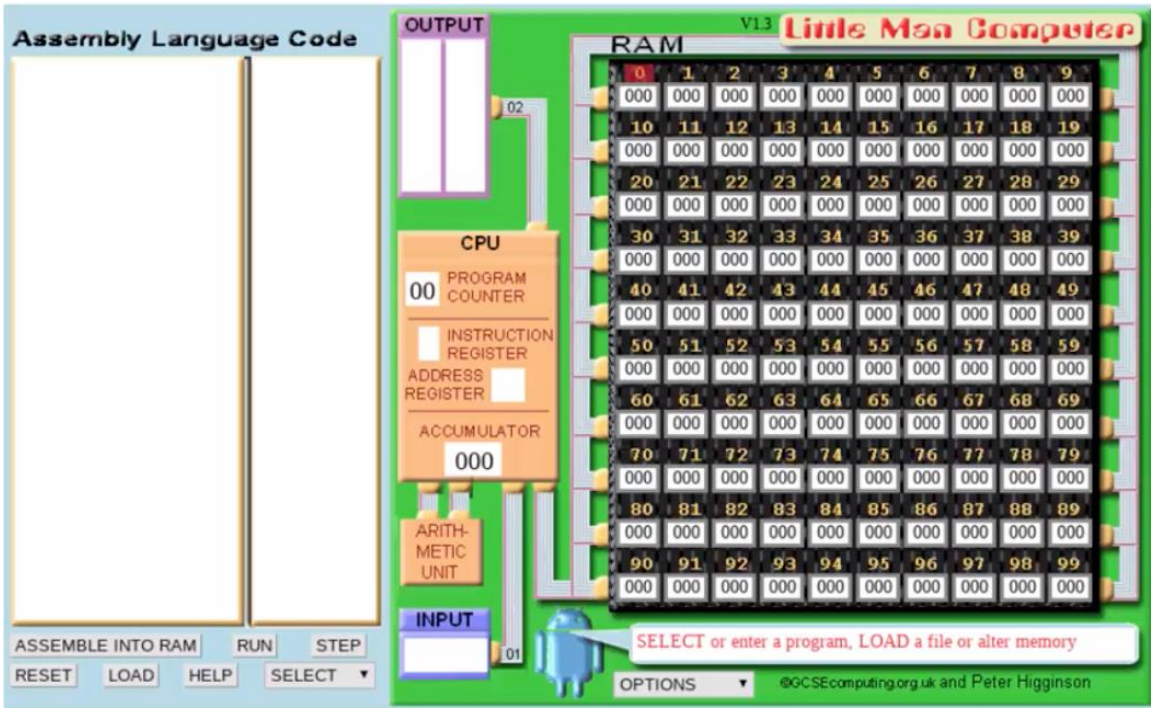
I prinsippet gjennomføres en prosessors arbeidssyklus slik:

1. Instruksjonshenteren sjekker instruksjonspekeren (programtelleren) for å finne ut hvilken adresse i internminnet neste instruksjon skal hentes fra, og kontakter bussgrensesnittet med denne informasjonen
2. Instruksjonen kommer inn via bussgrensesnittet.
3. Instruksjonspekeren settes til å peke på neste instruksjon.
4. Instruksjonen dekodes av instruksjonsdekoderen og resultatet lagres i registre.
5. ALU utfører instruksjonen, og resulterende verdier plasseres i registre. Dersom instruksjonen er en hopp-instruksjon, oppdateres registret som inneholder instruksjonspekeren. (En hopp-instruksjon er en instruksjon som forteller at programutførelsen skal fortsette en ny plass)

I praksis er det hele mer komplekst, moderne prosessorer inneholder blant annet komponenter som hele tiden prøver å hente instruksjoner og data før det faktisk blir behov for dem. Prosessorens arbeidssyklus styres av ei klokke. Selv om det kan variere hvor mange klokketikk en prosessor trenger for å utføre en oppgave, gir klokkehastigheten allikevel en pekepinn på prosessorens arbeidskapasitet. Klokka i en moderne mikroprosessor arbeider med en frekvens på i overkant av 3 GHz, det vil si at den tikker mer enn 3 milliarder ganger pr. sekund. Moderne prosessorer kan ha flere kjerner, noe som gjør det mulig å behandle flere instruksjoner i parallell.

En enkel, men teoretisk korrekt datamaskin, er LMC. Little Man Computer.

Det finnes en simulator av LMC på nettet. Eksempel til høyre:



LMC kan lese tall som brukeren skriver inn. Dette er inputboksen. LMC kan også skrive svar i form av tall, i output.

På moderne datamaskiner kan data inn komme fra mange forskjellige inputs. Tastatur, mus osv. Samme med outputs, som sendes til skjerm osv.

Minnet i LMC er en posthylle med hundre hyller nummerert fra 0 til 99. Hver hylle kan lagre ett tall.

CPU er den sentrale delen. Central Processing Unit. Den inneholder:

ALU

Se over.

Registre

Små lagerenheter/minneelementer som er veldig raske. Hurtig å få tak i. LMC inneholder de fire registrene vi kan se over:

Program counter – angir hvilken instruksjon i minnet som skal utføres neste gang.
instruction register – inneholder koden til instruksjonen som utføres
address register – inneholder adresseinformasjonen til den instruksjonen som utføres
accumulator – til beregninger. Svaret kommer hit.

Moderne prosessorer har fra 20 til noen hundre registre.

Arbeidsbeskrivelsen lagres på numerisk form i minnet. Hver instruksjon er et tall 0-999 der første siffer angir instruksjonen og 2 og 3 siffer angir en adresse i minnet.

599 betyr feks «hent verdien i adresse 99 i minnet og legg den i akkumulatorregisteret.» Kode 5 er altså en instruksjon om at prosessoren skal hente noe i minnet.

Kontrollogikken i CPU leser verdien i programtelleren og henter tallet i minnet med samme tall som programtelleren. Samtidig øker den programtelleren med en. Så splittes den innhentede instruksjonen og legger de forskjellige delene i instruksjonsregisteret og adresseregisteret. Utfører så det instruksjoneregisteret angir. Så gjentas prosessen.

Instruksjoner:

000 – HLT – («halt») stopper kjøringen
1xx – ADD – legger tallet i minnet med adresse xx til tallet i akkumulatoren
2xx – SUB – trekker tallet i minnet med adresse xx fra tallet i akkumulatoren
3xx – STA – lagrer tallet i akkumulator i minnet med adresse xx
5xx – LDA – henter verdi i minnet xx og setter det i akkumulatoren
6xx – BRA – setter programtelleren til xx adressen. Xx vil altså utføres neste gang.
7xx – BRZ – hvis akkumulatoren er 000, så settes programtelleren til xx
8xx – BRP – hvis akkumulatoren er 000 eller positivt, så settes programtelleren til xx.
901 – INP – hent input fra brukeren, stopper programmet til brukeren kommer med input. Input legges i akk.
902 – OUT – kopierer verdien i akk til output.

```
INP       // Les et tall
STA   99  // og legg det i celle 99.
INP       // Les et tall til
ADD   99  // og legg til tallet i celle 99.
OUT       // Skriv ut svaret.
HLT       // Avslutt.
```

Slik tekstlig kode kalles **assemblerkode**.

Kode	Navn	Beskrivelse
0xx	HLT	Stopper
1xx	ADD	Adderer angitt minneverdi med akkumulatoren
2xx	SUB	Subtraherer minneverdi fra akkumulatoren
3xx	STO	Lagrer akkumulatoren i minnet
4xx	—	(Ikke i bruk)
5xx	LDA	Henter verdi fra minnet til akkumulatoren
6xx	BRA	Hopper til angitt adresse
7xx	BRZ	Hopper hvis akkumulatoren er 0
8xx	BRP	Hopper hvis akkumulatoren er ≥ 0
901	INP	Leser verdi; svar i akkumulatoren
902	OUT	Skriver ut verdien i akkumulatoren
922	OTC	Skriver ut ASCII-tegn (ikke i boka)

Som LMC viser må hele tiden CPU frem og tilbake fra CPU og minnet. Dette kalles for Von Neumann-flaskehalsen. Vi har egentlig ingen bedre løsninger i dag.

Hvis vi ønsker å skrive «hei» i LMC er koden:

507 - henter verdi i minne 07 (i minne 07 ligger verdien 072)
922 - skriver ut ACII tegnet som er representert av verdien som ligger i akkumulatoren, altså 72. Dette er «h».
508 – samme
922 - som
509 - over
922 - …
000 – stopper programmet
072 – de tre neste kodene er egentlig bare for å lagre verdier i minnet som representerer ASCII-tegn.
101 -
105 -

Komprimert ASCII-tabell.

10	LF	39	'	47	/	62	>	93	]	124	|
32		40	(	48	0	63	?	94	^	125	}
33	!	41	)	:		64	@	95	_	126	~
34	"	42	*	57	9	65	A	96	‘		
35	#	43	+	58	:	:		97	a		
36	$	44	,	59	;	90	Z	:			
37	%	45	-	60	<	91	[	122	z		
38	&	46	.	61	=	92	\	123	{		

Numerisk kode (som over, altså 507, 922 osv.) er tungvint og vanskelig, derfor har vi assemblerkode. Assemblerkode oversettes til numerisk kode før programmet kjøres. I tillegg kan vi sette navn på celler i minnet. Vi kan også skrive kommentarer.

```
START  LDA   H
       OTC
       LDA   e
       OTC
       LDA   i
END    HLT

H      DAT   72    // = «h»
e      DAT   101   // = «e»
i      DAT   105   // = «i»
```

Ved å bruke «DAT» kan vi lagre en verdi i minnet med et navn. Vi lagrer for eksempel verdien 72 med navnet «H». Da kan vi skrive LDA H for å hente verdien til akkumulatoren, og så skrive OTC for å skrive it verdien i akkumulatoren (altså 72) som ASCII-tegn.

Instruksjonen BRA:

Branch. Med kode 6xx hopper programkjøringen til et annet sted. Dette gjør at vi kan lage løkker.

```
        LDA   v10    //henter verdien v10, som er 10, og legger den i akkumulator.
TELL    OUT          //skriver ut verdi i akkumulator, altså 10.
        SUB   v1     //Trekker verdien som ligger i v1 fra verdien i akkumulator. Altså 10 - 1. Akk er nå 9.
        BRA   TELL   //hopper til TELL og vi har derfor en løkke. Programtelleren settes til TELL, som er 1.
        HLT          //hit kommer vi aldri, siden programmet over er en evig løkke.

v1      DAT   1      //vi setter en konstant v1 og en konstant v10
v10     DAT   10
```

Instruksjonen BRP eller BRZ:

Hvis vi ønsker å hoppe noen ganger, men ikke alltid. BRP hopper til angitt adresse hvis akkumulator er større eller lik 0. BRZ hopper til angitt adresse hvis akkumulator er lik 0. Sjekk tabell over for kode.

```
        LDA   v10    //henter verdien v10, som er 10, og legger den i akkumulator.
TELL    OUT          //skriver ut verdi i akkumulator, altså 10.
        SUB   v1     //Trekker verdien som ligger i v1 fra verdien i akkumulator. Altså 10 - 1. Akk er nå 9.
        BRP   TELL   //hopper til TELL hvis akkumulator er større eller lik 0
        HLT          //Denne koden vil derfor utføres til slutt, i motsetning til koden over.

v1      DAT   1      //vi setter en konstant v1 = 1 og en konstant v10 = 10.
v10     DAT   10
```

Multiplikasjon:

Har ikke noen kode som kan multiplisere to tall i LMC. Men vi vet at:

A x B er det samme som B + B + B (A ganger)

3 x 4 er det samme som 4 + 4 + 4 (3 ganger)

Vi kan derfor lage en løkke med addisjoner for å finne svaret på multiplikasjonsoppgaver.

Kommentarene i eksempelet til høyre er skrevet i Pythonkode. Vi har tre variabler, a og b og Res. Og to konstanter v0 og v1, som er 0 og 1.

Vi starter med å lese inn input. Inputen legges i akkumulatoren og så lagrer vi akkumulatoren i variabelen a. Vi gjør det samme med variabel b. Så henter vi konstanten v0 til akkumulatoren og lagrer verdien i akkumulatoren i Res. Nå har vi altså de to verdiene som skal ganges sammen og et midlertidig resultat, som er 0.

Neste steg er løkken. Vi skal legge b til hverandre a ganger. B + B + B …

Vi starter med å hente verdien i variabelen a til akk. Verdien er bestemt av bruker. Hvis akkumulatoren er 0 så hopper vi til EXIT. Hvis den ikke er 0 så trekker vi konstanten v1 fra verdien i akk (som er a) også lagrer vi verdien i akk i a. (Nå er altså a redusert med 1). Så henter vi verdien i variabelen Res og legger den i akk. Så legges verdien i variabel b til verdien i akk, og så lagrer vi verdien i akk til Res. Så hopper vi til LOOP og sjekker på nytt om a er 0…

```
START INP       //     int(input(""))
      STA a     // a =
      INP       //     int(input(""))
      STA b     // b =
      LDA v0    //       0
      STA Res   // Res =

LOOP  LDA a     // while a != 0:
      BRZ EXIT  //
      SUB v1    //       a - 1
      STA a     //   a =

      LDA Res   //         Res
      ADD b     //             + b
      STA Res   //   Res =
      BRA LOOP  //

EXIT  LDA Res   //       Res)
      OUT       // print(
      HLT       //

a     DAT
b     DAT
Res   DAT

v0    DAT 0
v1    DAT 1
```

Overflyt:

Cellene i minnet er av en fast størrelse og kan bare lagre et begrenset antall verdier. Hvis vi forsøker å lagre tall som er større enn cellen kan «takle» støter vi på et problem.
Overflyt er når vi har for store verdier til å sette inn i tilgjengelig lagerplass. For å håndtere overflyt har vi to teknikker:

Alt 1: Strutseteknikken.

Overser problemet. Om det blir overflyt, så er det programmererens feil. Dette er raskt, men kan da åpenbart føre til galt svar. Brukes i Java, C, C++ osv. (Nettopp på grunn av at dette er raskt)

Alt 2: Test hver operasjon

Sjekk ved hver operasjon om svaret er så stort at det må lagres på et annet format.
Alltid riktig svar. Koster mye eksekveringstid. Brukes blant annet i Python.

LMC kan lagre tall fra -999 til 999.
De fleste datamaskiner i dag har minneceller på 8bit som kan lagre -128 til 127, men de tilbyr også støtte for å slå sammen byte for å kunne lagre større tall.

byte	8 bit	−128 - +127
short	16 bit	−32 768 - +32 767
int	32 bit	−2 147 483 648 - +2 147 483 647
long	64 bit	−9 223 372 036 854 775 808 - +9 223 372 036 854 775 807

Instruksjoner og data:

Hvordan vet datamaskinen at verdien i celleminnet inneholder en instruksjon eller data? Det vet den egentlig ikke. Et sett med instruksjoner bør/må derfor avsluttes med en 000/HLT instruksjon, ellers vil den forsøke å utføre neste celle i rekken, som bare er en verdi, som en instruksjon.

Når instruksjoner kodes som tall, kan vi endre programmet vårt mens det kjøres. Programmet til høyre vil kjøre i en evig løkke.

```
Skriv   LDA   txt        v1    DAT    1    // Konstant 1
        OTC
        LDA   Skriv      txt   DAT   86    // 'V'
        ADD   v1               DAT  101    // 'e'
        STA   Skriv            DAT  108    // 'l'
        BRA   Skriv            DAT  107    // 'k'
                               DAT  111    // 'o'
Ferdig  HLT                    DAT  109    // 'm'
                               DAT  109    // 'm'
                               DAT  101    // 'e'
                               DAT  110    // 'n'
                               DAT   33    // '!'
```

Lister:

En Python-liste er et sammenhengende område av celler i minnet. Vi bruker et heltall (en indeks) til å hente riktig element.

```
m_leng = [ 0, 31, 28, 31, 30, 31, 30,
           31, 31, 30, 31, 30, 31 ]
m = int(input(""))
print(m_leng[m])
```

kan vi skrive for LMC ved å bruke selvmodifiserende kode:

```
        INP           // Les m.            m_leng  DAT    0
        ADD    LDA_i  // Lag LDA                   DAT   31  // Jan
        STA    hent   // i koden.                  DAT   28  // Feb
hent    DAT    0      // Plass til LDA.            DAT   31  // Mar
        OUT           // Skriv svaret.             DAT   30  // Apr
        HLT           // Ferdig.                   DAT   31  // Mai
                                                   DAT   30  // Jun
LDA_i   LDA    m_leng                             DAT   31  // Jul
                                                   DAT   31  // Aug
                                                   DAT   30  // Sep
                                                   DAT   31  // Okt
                                                   DAT   30  // Nov
                                                   DAT   31  // Des
```

OBLIG 2:

```
LOKKE   INP         //input fra bruker lagres i akkumulator

        BRZ AVSL    //hvis input er 0 går programmet på avslutningen

        STA naa      //akkumulator-tall lagres i celle naa

        LDA teller  //oppdaterer teller med +1
        ADD v1
        STA teller

        LDA naa
        SUB maks    //trekker verdien i variabel maks fra verdien i akkum.
        BRP SETT    //Hvis akkum nå er større eller lik null er det nye tallet størst
        BRA LOKKE

SETT    LDA naa
        STA maks
        BRA LOKKE

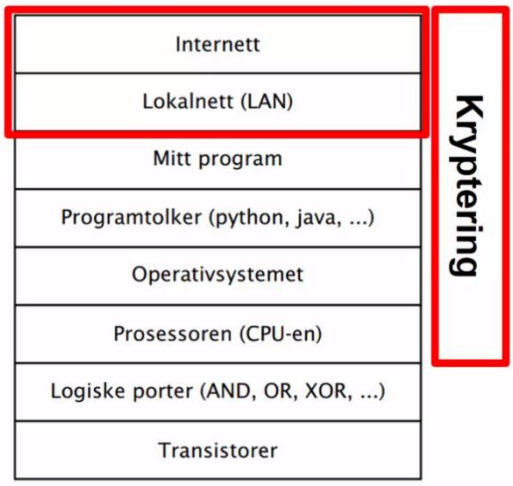
AVSL    LDA n
        OTC
        LDA =
        OTC
        LDA teller
        OUT
        LDA nl
        OTC

        LDA m
        OTC
        LDA a
        OTC
        LDA x
        OTC
        LDA =
        OTC
        LDA maks
        OUT
        HLT

n DAT 110
= DAT 061
m DAT 109
a DAT 097
x DAT 120
nl DAT 010

v1 DAT 1
naa DAT 000
teller DAT 000
maks DAT 000
```

Kryptering / kryptografi



Kryptografi er kunsten å skrive hemmelig. Scramble informasjonen til noe helt ugjenkjennelig og samtidig muliggjøre at rett person med en hemmelig metode/nøkkel kan lese dette.



Plaintext: klar tekst
Ciphertekst: kryptert tekst

Algoritmer: oppskrift for å gjøre beskjeden hemmelig

Disse bør være offentlige. Da kan man luke ut svakheter. Hjelper uansett ikke å dekode informasjon med algoritme uten nøkkel. Så lenge algoritmen er designet godt hjelper det ikke å lese hvordan den funker, man må uansett ha nøkkelen. Er algoritmen offentlig får man «gratis» hjelp av kryptoanalytikere. Kommersielle algoritmer er ofte offentlige, mens militære er hemmelige.

Nøkler: noe som brukes til å gjøre om tekst fra plain til ciphertext eller omvendt.

Kryptograf: finner opp smarte algoritmer
Kryptoanalytiker: knekker smarte algoritmer.

Hvordan knekke kryptering?

Kompleksitet vs. Brute-force:

Brute force er å prøve alle kombinasjoner for å knekke en kode.
Man kan foreta avskjæringer ved frekvensanalyse. Se hvor ofte enkelte bokstaver forekommer i dagligtalen og bruke dem først ved forsøket på gjette passordet. Eller prøve ord i ulike ordbøker.

Hvis man kjenner plaintekst og ciphertekst kan man sammenligne og teste.
Kjenner man kun deler av plain eller cipherteksten kan man teste forventet verdi til man finner riktig.

Sikkerhet i trådløse nettverk

Hvis man kobler seg opp til et åpent og ukryptert nettverk kan alle lese all kommunikasjon.
Moderne GPUer kan teste over 1 million nettverkspassord i sekundet.

8 siffer = 28 timer.
6 bokstaver = 4 dager.

Forskjellige typer kryptering

Symmetrisk kryptering:

Vanlig kryptering. Man deler en nøkkel. Har plaintekst. Kjører den igjennom krypteringsalgoritmen med den hemmelige nøkkelen og får ciphertekst. Mottakeren må også bruke den hemmelige nøkkelen på cipherteksten for å få tak i plainteksten.



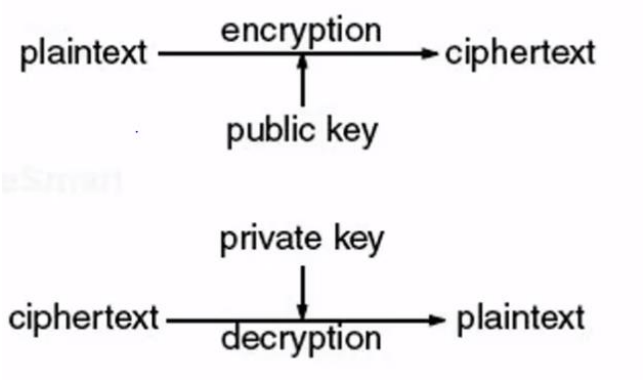
To typer symmetrisk kryptering:

Block cipher – krypter blokk for blokk, altså større deler man krypterer. (AES)
Brukes i datanett.

Stream cipher – krypterer fortløpende hver bit. (RC4)
Brukes i mobilnett.

Block cipher er mest brukt og da gjerne i blokker på 128-bits.
Så lenge algoritmen er sikker og nøkkelen er hemmelig kan man snakke på en usikker kanal.
Problemet med symmetrisk kryptering er at man må få levert den hemmelige nøkkelen til mottaker på en sikker måte. Det er ikke alltid mulig å møtes privat.

Asymmetrisk kryptering.



Også kjent som offentlig nøkkelkryptering. En offentlig nøkkel som gjerne ligger åpent på nettet for alle. Denne bruker alle som snakker med deg. En privat nøkkel som er hemmelig. Denne låser opp alt som kommer til deg og alt som går ut. Man har altså to nøkkelpar. Trenger ikke utveksle nøkler for å snakke. Asymmetriske krypteringsalgoritmer er mange ganger tregere enn symmetriske algoritmer. Man bruker derfor ofte asymmetrisk kryptering for å dele en hemmelig nøkkel, så krypterer man resten av samtalen symmetrisk.
Kan også brukes som digital signatur. Man bruker da mottagers offentlige nøkkel til å kryptere teksten til ciphertekst. Så sendes denne til mottaker. Mottaker bruker da sin egen private nøkkel til å dekryptere cipherteksten til plaintekst.

Hash-algoritmer / enveisfunksjoner / sjekksummer

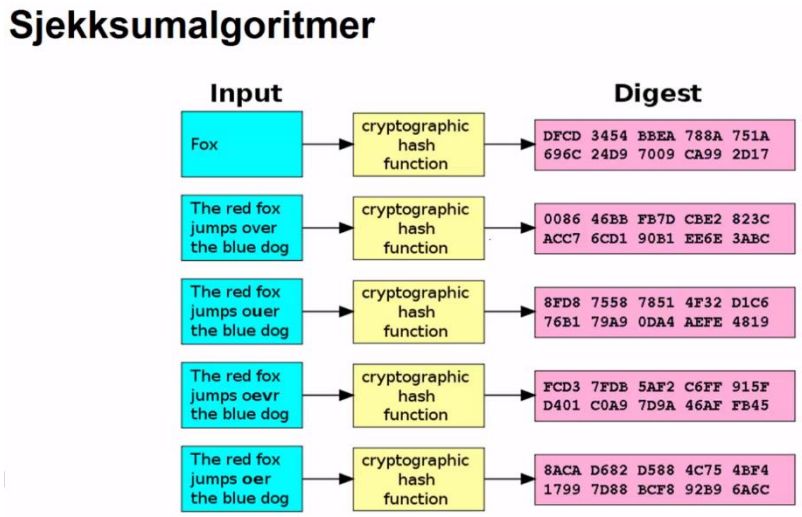
(Hashing er å gjøre store data om til enkle små data)

Kryptografiske enveisfunksjoner. Skal ikke være reverserbar. En sjekksum er en kort kode som brukes til å sjekke integriteten av data. Brukes til signering, ikke kryptering.

Kriterier for en god sjekksumalgoritme:

- Enkelt å regne ut sjekksum for en gitt beskjed
- Ikke mulig å finne en beskjed for en gitt sjekksum. Altså gå fra sjekksum til beskjed
- Ikke mulig å nedre en beskjed uten at sjekksummen blir endret
- Ikke mulig å finne to forskjellige beskjeder med samme sjekksum

For å lage god kryptografi er det viktig med en god tilfeldig tall generator.



Lifetimes of popular cryptographic hashes (the rainbow chart)

Function	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
Snefru																												
MD2 (128-bit)[1]																												
MD4																												
MD5															[2]													
RIPEMD															[2]													
HAVAL-128[1]															[2]													
SHA-0																												
SHA-1																												[3]
RIPEMD-160																												
SHA-2 family																		[4]										
SHA-3 (Keccak)																												

Key: Didn't exist/not public | Under peer review | Considered strong | Minor weakness | Weakened | Broken | Collision found

Datamaskinarkitektur

Egen intro:

Hastigheten til en prosessor, CPU, bestemmes av klokkefrekvensen, eller klokkesyklus. En klokkefrekvens er antall svingninger, sykluser eller pulser per sekund prosessoren operer med. Alle datamaskiner inneholder klokker som sender en taktfast strøm av elektriske pulser for å koordinere samspillet mellom prosessoren, minnet og andre enheter. En 4 GHz prosessor utfører 4 000 000 000 klokkesykluser per sekund. Prosessorer utfører en eller flere instruksjoner per klokkesyklus.

I forelesningene går Omid igjennom pipelining, som er å utforme kretsen slik at CPUen kan utføre flere instruksjoner om gangen. Det motsatte av pipelining kan sies å være en en-syklus-implementasjon. Da jobber CPUen kun med en instruksjon av gangen.

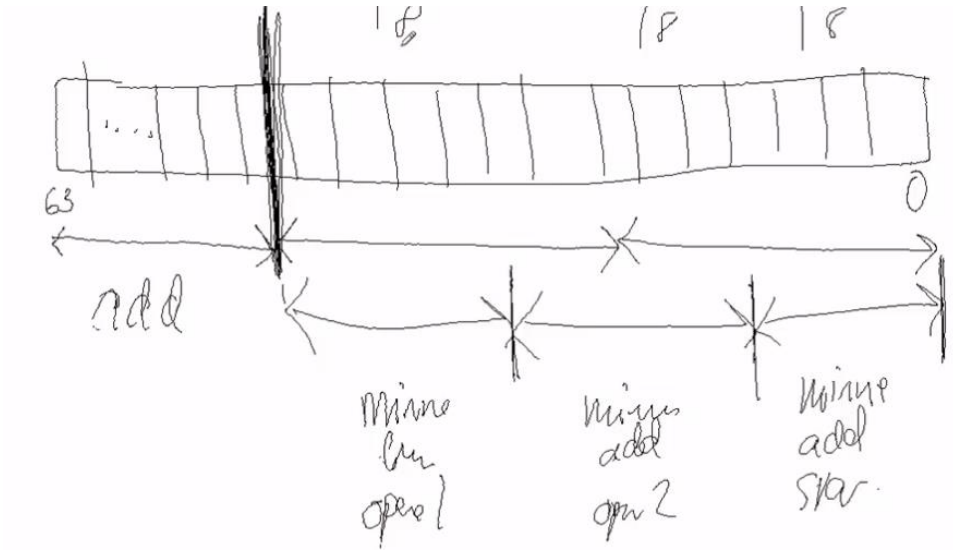
Vi ønsker at ting skal gå fort, bruke minst mulig strøm og være billig.

123 i assemblerkode er koden for å addere verdien i minnet 23 til akkumulatoren.
Koden 123 kan deles opp i to deler. Første delen 1 er instruksjonen (som er å addere) mens andre delen 23 er adressen i minnet. For moderne datamaskiner er en maskinkode på typisk 32 eller 64-bit. Er forskjellig fra prosessor til prosessor hvor mange instruksjoner som finnes.

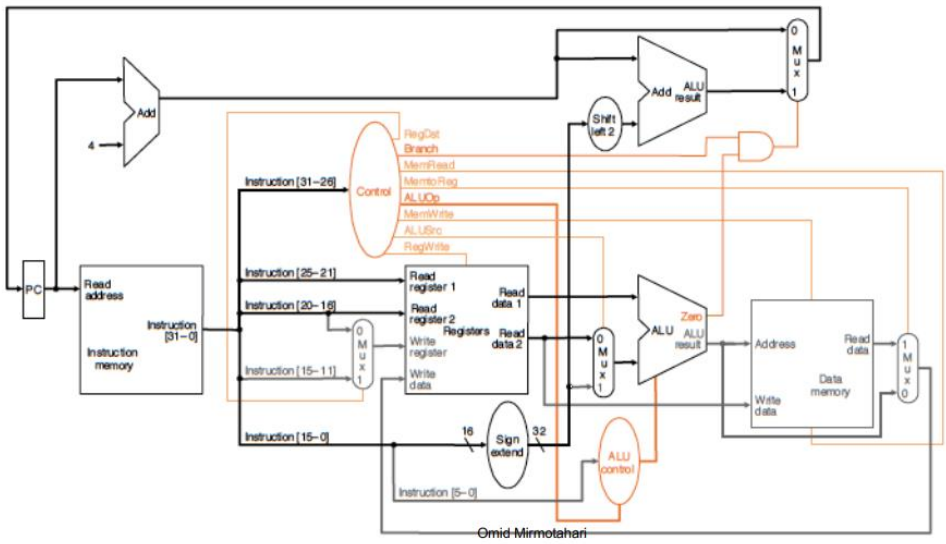
64 bits arkitektur:

En maskinkode er på 64 bit. Vi setter en del av de mest signifikante bitene til å henvise til instruksjoner. Hvis vi da skal lage en maskinkode som adderer to tall i to forskjellige minneceller og lagrer svaret i en tredje minnecelle, så må vi splitte opp den 64 bit store maskinkoden i fire deler. 1 del til instruksjonen (som sier at vi skal addere) 2 deler til tallene som skal adderes og 1 del til svaret.

I tegningen til høyre er 10 bit satt av til instruksjonen (10 største bitene). Vi har satt av 18 bit til de tre andre delene som er tallene som skal adderes og svaret. Det høyeste tallet 18 bit kan representere er 2^{18} - 1. Det er 262 000 bit. Hvis hver maskinkode øker fra 64 til 128 bit kan vi aksessere mye større tall.

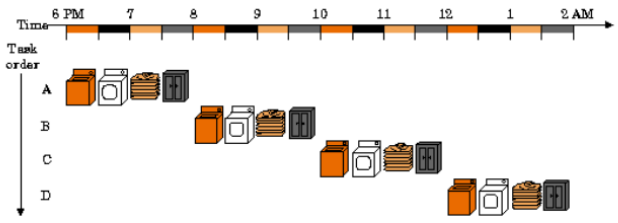


Alt går igjennom ALU. Jo flere operasjoner ALU kan gjøre, jo flere bit trengs. I tegningen over er det den oransje kontrollenheten som mottar instruksjonsdelen av maskinkoden. En del av denne kan vi se at går til ALU for å bestemme hva slags operasjon som skal utføres.

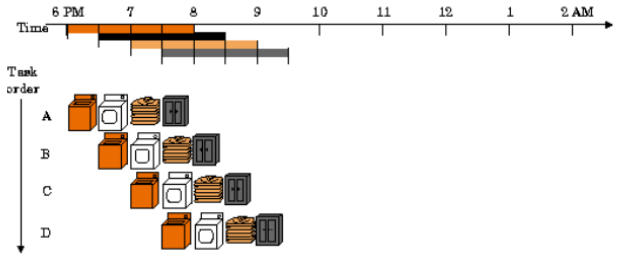


Pipelining:

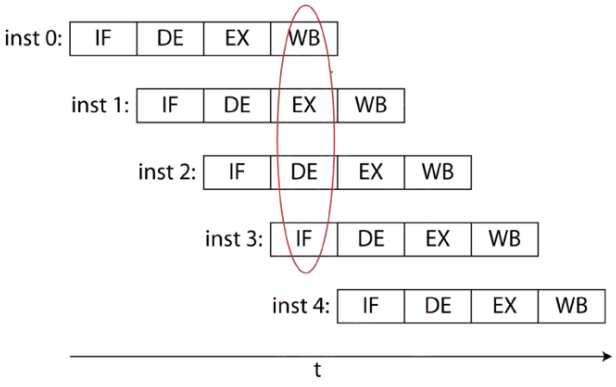
Et samlebåndsprinsipp for eksekvering av instruksjoner. Hver instruksjon splittes opp i uavhengige deler som skal utføres etter hverandre. Hver subinstruksjon kan utføres uavhengig av de andre subinstruksjonene. Neste instruksjon settes i gang før forrige instruksjon er helt ferdig. Hver instruksjon tar da like lang tid å utføre som til vanlig, MEN prosessoren utfører flere instruksjoner i et gitt tidsrom.



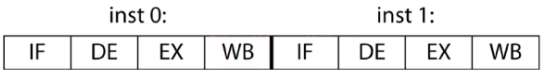
- Eksempelvis kan vi ha følgende subinstruksjoner i en pipeline
  - IF: Instruction fetch (get the instruction)
  - DE: decode and load (from a register)
  - EX: Execute
  - WB: Write back (write the result to a register)



- **PS! Read og write fra register/MEM kan gjøres i hver sin halvdel av en sykel (1/2 klokkeperiode)**



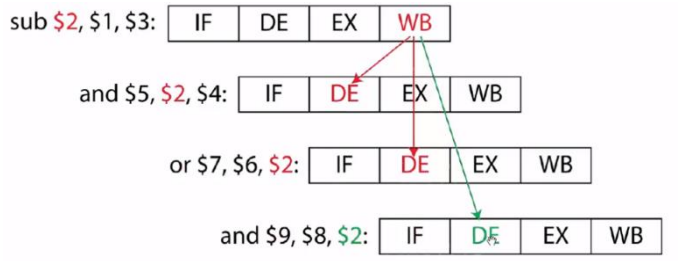
without pipelining



I eksempelet til venstre har vi fire steg. Likevel går det ikke fire ganger så raskt. Det går bort tid til administrering av instruksjoner.

Resource hazard: to subinstruksjoner i pipelinen ønsker å aksessere samme ressurs, for eksempel minne.

Data hazard: to forskjellige instruksjoner trenger å aksessere samme data samtidig.



Bare den siste instruksjonen vil lese riktig data.

Control hazard: Hvis vi får en jump instruksjon må vvi tømme pipelinen for andre instruksjoner.

Andre prosessorer enn LMC

Flere generasjoner datamaskiner, noen av dem norske.
Dagens moderne prosessorer er ofte x86-64.

Det er vanlig å dele datamaskinene inn i generasjoner:

	År	Teknologi	Størrelse	Instr/sek	Pris (2017-kr)
1	1945-60	Radiorør	10m³	2000	60 mill
2	1960-68	Transistorer	650dm³	500 000	50 mill
3	1968-78	IC	80dm³	300 000	200 000
4	1978-??	LSI og VLSI	0,1-20dm³	10^9	1-10 000

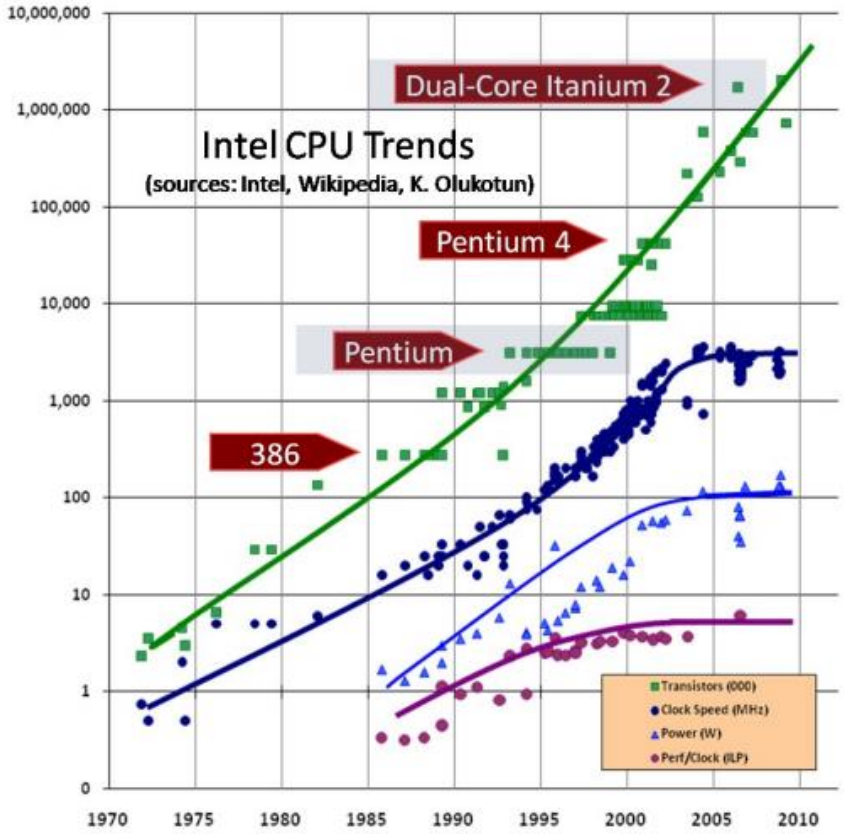
Første generasjon. Radiorør:
Var gigantiske og radiorørene måtte byttes ofte. Eniac mest kjente maskin.
Andre generasjon. Transistorer:
Maskinene ble mindre, raskere og mye mer pålitelige.
Tredje generasjon. Integrerte kretser.
Enda mindre og billigere.
Norsk data. Datamaskinprodusent fra 1967. laget flere minimaskiner, Nord-1, Nord-10 etc. 1987 hadde de 4500 ansatte, Norges nest største firma. Selskapet kollapset i 1992. De holdt seg til et middelmådig egenlaget operativsystem.
Fjerde generasjon: mikroprosessorer.
Hele CPUen fikk plass på en brikke. Før (tredje gen) hadde CPUen plass på et datakort. Nå kunne datamaskinen bli allemannseie.
Mycron, norsk datamaskin fra 1974. Lagde noen av de første mikromaskinene i verden. Sluttet å lage datamaskiner i 1990.
Tiki Data, norsk datamaskinprodusent fra 1983. Satset på rimelige mikromaskiner, spesielt for norske skoler. Firmaet opphørte i 1996.

Prosessoren x86-64:
Intels x86 har gått fra å være en av mange prosessorer til å bli den dominerende. De var tidligere ute med nyvinninger og bakoverkompatible.

X86-64 fra intel og AMD finnes i de aller fleste vanlige datamaskiner i dag.

Både LMC og x86-64 har klassisk von Neumann-struktur, men x86-64 har i tillegg cache for raskere minneaksess.

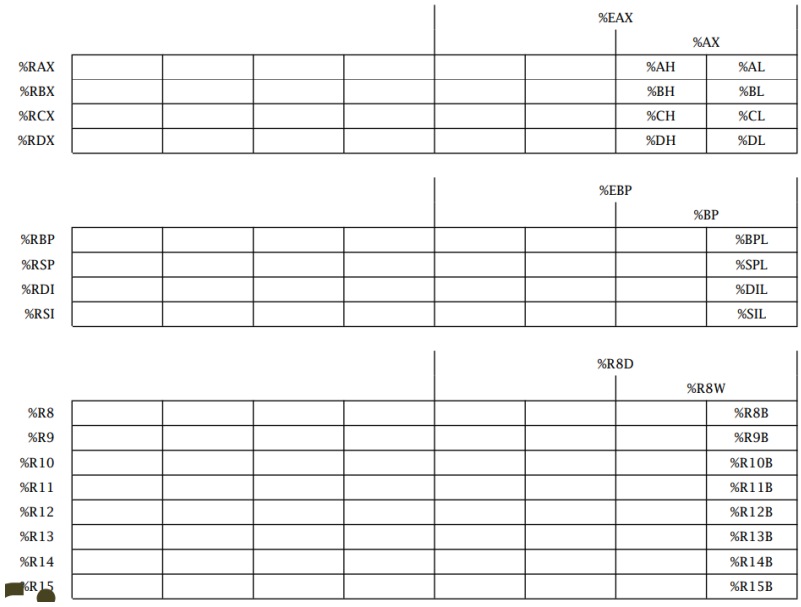
Begge har også samme eksekveringsløkke:
1. Hent instruksjonen i minnet, programtelleren angir hvor.
2. Dekod instruksjonen i operasjon og adresse, de



legges i instruksjonsregisteret og adresseregisteret.

3. Utfør instruksjonen
4. Om aktuelt, skriv svar i minnet.

LMC har ett register for programmereren. Det kan lagre verdier -999 til 999.
Z86-64 har 16 registre. De kan brukes som 1-, 2- ,4- og 8-bytes registre.



LMC har 100 celler i minnet, og hver celle kan lagre verdier -999 til 999.
X86-64 kan ha inntil 2^{64} celler i minnet, dvs, 17 000 000 TB (terabyte). Ingen maskiner har i dag så mye installert. Hver celle inneholder en byte. I tillegg har x86-64 støtte for virtuelt minne for å kunne dele minnet mellom flere programmer og brukere samtidig.

LMC er en RISC-maskin mens x86-64 er en CISC-maskin.

CISC: Complex instruction set computer. har noen enkle og noen ganske kompliserte instruksjoner, som:
REP MOVSB: kan flytte et vilkårlig antall byte fra ett sted i minnet til et annet.

RISC: Reduced instruction set computer. Har et begrenset antall enkle instruksjoner, men kan utføre disse ekstremt raskt siden alle er enkle.

Femte generasjons datamaskiner vil (sannsynligvis) ha samme ytelse for hver enkeltprosessor som i dag. Høy grad av parallellisering.

Et eksempel

Anta at vi skal lage kode tilsvarende Python-setningen

```
a[i] += 1
```

x86-64

```
leaq    a,%r8
movq    i,%r9
incq    (%r8,%r9,8)
```

LMC

```
        LDA     LDA_A_i
        ADD     i
        STA     INCR_i
        LDA     STA_A_i
        ADD     i
        STA     INCR_i2
INCR_i  DAT     0
        ADD     v1
INCR_i2 DAT     0
          :
LDA_A_i LDA     A
STA_A_i STA     A
v1      DAT     1
```

Operativsystemer

Datamaskiner bruker et OS for å kunne kommunisere med brukeren, administrere kjøringen (starte og stoppe programmer) og administrere lagring av filer. Dette skal skje på en sikker (at jeg ikke ødelegger for noen) og trygg (at noen ikke ødelegger for meg) måte. Operativsystemet er et program som alle andre i datamaskinen. Det er skrevet i et programmeringsspråk. Forskjellen er at det er det første som lastes inn i minnet når maskinen startes, og det styrer kjøringen av alle programmer som lastes inn senere.

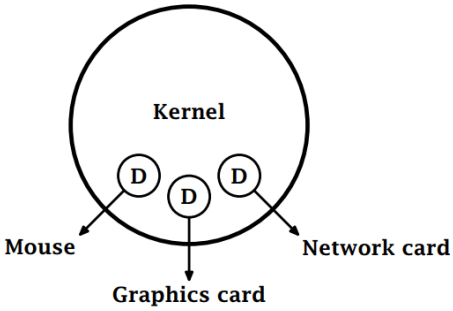
Lærebokens definisjon er:

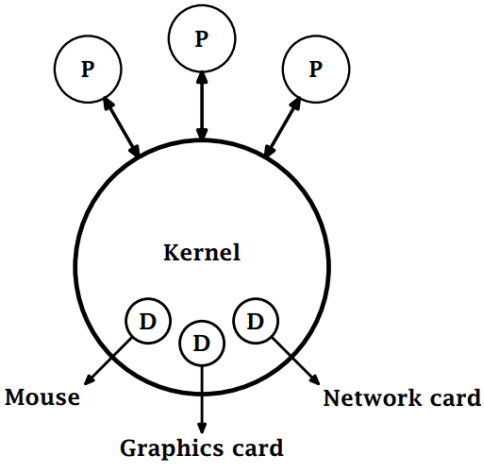
> *An operating system is a collection of computer programs that integrate the hardware resources of the computer and make those resources available to a user and the user's programs, in a way that allows the user access to the computer in a productive, timely, and efficient manner.*

Oppbygging av et operativsystem:

Et hovedproblem er at det skjer mye samtidig. Den vanligste løsningen er å ha en kjerne som styrer alt. Den kjører i «supervisor mode».

Kjernen kommuniserer med periferienheter med spesialskrevne kodebiter kalt drivere (D). F.eks. hvis man putter en minnepinne i datamaskinen vil denne installere sin driver.





Brukerprogrammene er prosesser. De kjører i protected mode og må kommunisere med kjernen for kommunikasjon med periferenhetene og kommunikasjon med andre prosesser.

Hvis datamaskinen har flere kjerner, kan prosessene kjøre på ulike kjerner og dermed gå fortere.

Noen Oser tilbyr også tråder. Flere tråder i en prosess kan kjøres parallelt, men de deler minne.

Når det er flere prosesser som skal kjøres på samme kjerne, vil Oset bytte på å kjøre dem. (tidsdeling eller timeslicing).

Når man slår på en datamaskin starter den en bootstrap loader som ligger i BIOS (Basic Input Output System). BIOS ligger i ROM (Read Only Memory) Dette er en type hardkodet minne. BIOS er spesiallaget for den enkelte datamaskinmodell. Det er en samling rutiner som kan lese fra disk eller nettverk. Bootloaderen leser inn kjernen i selve OS-et og starter det.

Alle system- og brukerprogrammer kjører som prosesser. Alle prosesser har et unikt nummer, PID. Kommandoen ps gir oversikten til høyre. Kan skrive ps -ef for å vise alle prosessene på maskinen, det vil være flere hundre.

```
$ ps
  PID TTY          TIME CMD
19967 pts/1    00:00:00 bash
24715 pts/1    00:00:18 emacs
26332 pts/1    00:00:00 preview
26333 pts/1    00:00:00 sh
26336 pts/1    00:00:04 xpdf
27842 pts/1    00:00:00 ps
```

Starte nye prosesser:

Et program må kalle på en funksjon i kjernen ved å benytte såkalte systemkall. Dette kan gjøres fra de fleste programmeringsspråk, i Python benyttes biblioteket subprocess.

Følgende program starter en klokke som går inntil vi stopper den.

```
import subprocess

print("Starter 'xclock'...")
p = subprocess.Popen(["xclock", "-update", "1"])
print("Da er den i gang.");
```

Drepe prosesser:

Funksjonen kill vil drepe en angitt prosess.

```
import subprocess
import time

print("Starter 'xclock'...")
p = subprocess.Popen(["xclock", "-update", "1"])
print("Da er klokken i gang.");

for sek in range(5, 0, -1):
    print(sek)
    time.sleep(1)

p.kill()
print("Nå er klokken borte.")
```

Kommandotolker:

Kommandotolkeren (Shell-et) er en av prosessene; den vanligste i Linux heter bash (Bourne again shell). Men alle kan enkelt lage sin egen.

Kommandotolkeren leser kommandonavn som brukeren skriver, og oppretter en prosess som utfører den.

Normalt vil bash vente til prosessen er ferdig før den ber om ny kommando. Hvis kommandolinjen slutter med «&», venter den ikke. Derfor vi skriver «Firefox &» i Linux.

Ulike commands:

Date, cal, ping, xeyes, ps,

Ctrl + c stopper den prosessen som nå kjører i kommandovinduet.

Det er enkelt å lage sin egen kommandotolker, se kode til høyre:

Som nevnt er det veldig enkelt å lage sin egen kommandotolker:

```
import subprocess

while True:
    linje = input("Kommando: ")
    kommando = linje.split()
    if kommando[0] == "exit":
        break

    p = subprocess.Popen(kommando)
    p.wait()
```

Fjerninnlogging:

Det er anledning til å åpne et kommandovindu på en annen maskin.

Alle moderne OSer har et GUI (Graphical User Interface) for å ha mange vinduer på skjermen, ha grafikk i vinduene og for å få input fra mus.

Filer:

En fil er en samling byte lagret i eller ved datamaskinen. På disk, SSD, HDD eller lignende. Operativsystemet holder orden på filens navn, hvilken enhet den er lagret på og hvor i mappestrukturen den ligger.
Fillagring skjer i et tresystem. Kommandoen ls (list files) forteller oss om filene i den mappen vi er i. ls – l gir informasjon om filene i mappen vi er i.

```
-rw-r--r--. 1 dag ifi-a   71193 Oct 10 14:04 hullband.jpg
```

-rw-r--r--.	er beskyttelsen
1	er antall linker
dag	er filens eier
ifi-a	er filens gruppe
71193	er filens størrelse i byte
Oct 10 14:04	er dato og klokkeslett da filen ble laget
hullband.jpg	er navnet

Beskyttelse av filer: unix operer med tre kategorier brukere.

user	(forkortet «**u**») er filens eier, dvs den som opprettet den.
group	(forkortet «**g**») er gruppen tilordnet filen. (Mer om dette siden.)
other	(forkortet «**o**») er alle andre brukere.

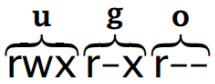
Det finnes tre privilegier for filer:

read (forkortet «**r**») er retten til å lese filen.

write (forkortet «**w**») er retten til å skrive på filen, dvs endre den.

execute (forkortet «**x**») er retten til å utføre den, dvs bruke den som et program.

Privilegiene angis i bolker på 3×3 bokstaver:



En bokstav angir at man har privilegiet, en «-» at man ikke har det.

Egne programmer kan kjøres fra den mappe de ligger i. /filnavn
De kan også kjøres fra andre mapper om man angir hele filnavnet. /Kurs/in1020/filnavn

Ved å sette **x**-bit-et kan man enkelt lage nye kommandoer:

1. Lag en tekstfil med de kommandoene vi ønsker utført.
2. Sett **x**-bit-et:
   $ chmod ugo+x *filnavn*
3. *filnavn* er nå et program og kan utføres med
   $ *./filnavn*

Dette gjør det svært enkelt å tilpasse systemet med egne kommandoer.

Dette programmet **nett-ok** sjekker om nettet er OK ved å spørre Ifi:

```
$ more nett-ok
ping -c4 www.ifi.uio.no

$ chmod uog+rx nett-ok

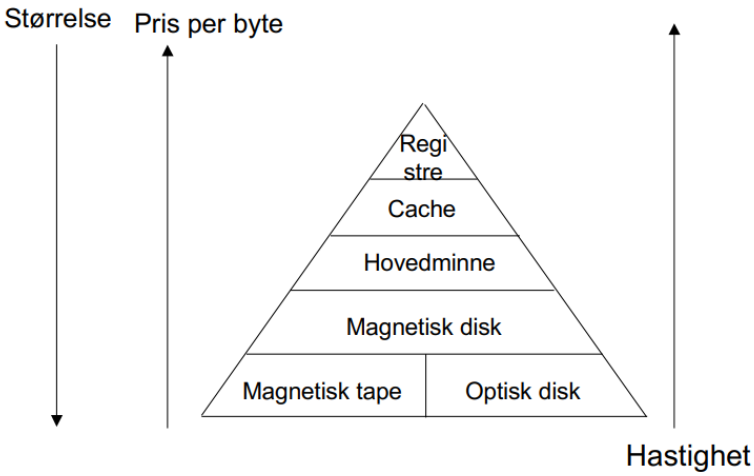
$ ./nett-ok
PING webserver.ifi.uio.no (129.240.65.48) 56(84) bytes of data.
64 bytes from webserver.ifi.uio.no (129.240.65.48): icmp_seq=1 ttl=63 time=0.411 ms
64 bytes from webserver.ifi.uio.no (129.240.65.48): icmp_seq=2 ttl=63 time=0.534 ms
64 bytes from webserver.ifi.uio.no (129.240.65.48): icmp_seq=3 ttl=63 time=0.539 ms
64 bytes from webserver.ifi.uio.no (129.240.65.48): icmp_seq=4 ttl=63 time=1.56 ms

--- webserver.ifi.uio.no ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.411/0.761/1.563/0.466 ms
```

Minnehierarki

Register:

Intern kladdebok for CPUen med rask aksess til innholdet. Integrert på CPUen. Om lag 32-128 registre.



Cache:

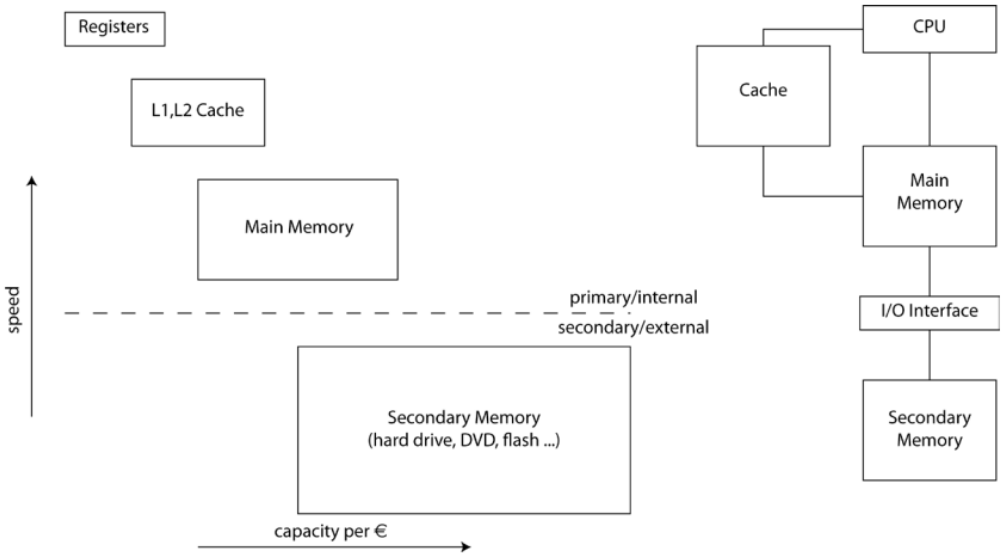
Hurtig mellomlager for både instrukser og data for å jevne ut hastighetsforskjellen mellom CPUen og hurtigminnet. Mellomlager internt (L1) eller i nærheten (L2, L3) av CPU. Typiske kapasiteter er fra 10 KB (L1) til 1 MB (L2) og flere MB (L3).

RAM:

Buffer mellom eksternt lagringsmedium og CPU med rask både lese- og skriveaksess. Internt på hovedkortet i nærheten av CPUen, størrelser på flere GB.
På Registre, Cache og RAM mister man all data når strømmen skrus av.

SSD/Flash/Harddisk:

Høykapasitetsmedium for program/data. Kapasitet på opptil flere TB.

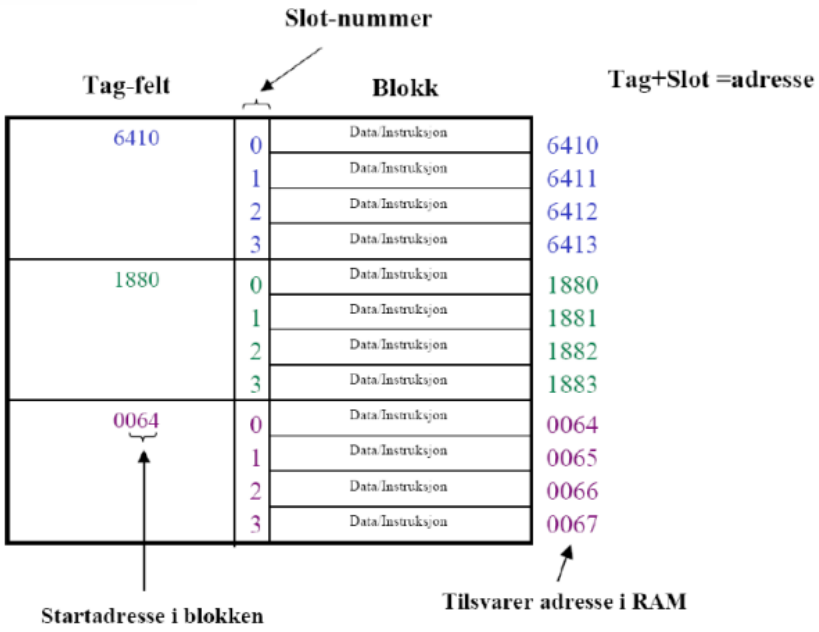


Aksesshastighet:

Registers	< 1ns	≈ 100 Byte
L1 (på CPU) cache	≈ 1ns	≈ 10 KB
L2,L3 (utenfor CPU) cache	2-10ns	≈ 1 MB
Hovedminne (RAM)	20-100ns	≈ 1 GB
SSD/Flash	100ns-1us	≈ 1 TB
Harddisk	1ms	≈ 1 TB

Cache:

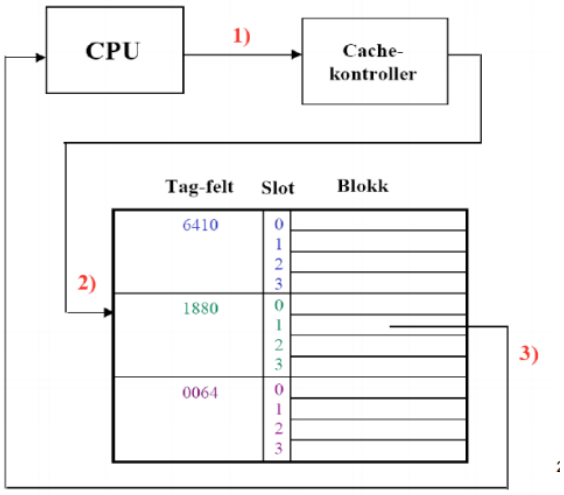
Cache forbedrer von Neumann arkitekturens flaskehals med hensyn på minneaksessering. Cache inneholder en kopi av et subsett av hurtigminne (RAM). CPUen henter data/instruksjoner fra cache istedenfor RAM. Siden cache er mindre enn RAM, må det bestemmes hvilke data og instruksjoner som skal ligge der. Hvordan man allokerer plassen som kan opptas i cache har mye å si for datamaskinens hastighet. Hvis data/instruksjonen CPUen trenger ligger i cache, får man en cache hit. Hvis den ikke ligger i cache, får man en cache miss. En essensiell del for utnyttelse av cache er å kunne raskt sjekke for hit eller miss.



Cache hit

- Når en prosessor skal hente en instruksjon eller lese/skrive data, vet den ikke om den skal hente fra RAM eller cache. Hvis det prosessoren ber om ligger i cache, kalles det en cache hit

Eks: hent innholdet i loaksjon 1881.



Cache miss:

Ved cache miss må data/instruksjon først kopieres fra hurtigminnet (RAM) over i cache før CPUen kan bruke det. Cache fylles fra hurtigminnet første gangen en bestemt lokasjon blir referert.

I eksempelet til høyre er et sentralt spørsmål, hvilke instruksjoner/data skal vi kaste ut/overskrive? Hvordan skal vi fylle cache? Det er tre hovedmetoder:

Direct-mapped cache:
En bestemt blokk fra RAM kan bare plasseres i en bestemt blokk i cache. Flere RAM-lokasjoner må «konkurrere» om samme blokk i cache. Det blir da lett å sjekke om riktig blokk finnes i cache, siden man bare trenger å sjekke ett tag-felt og se om denne inneholder blokken man leter etter. Ulempen er at vi får høy miss rate, selv om det er ledig andre plasser.

Cache miss

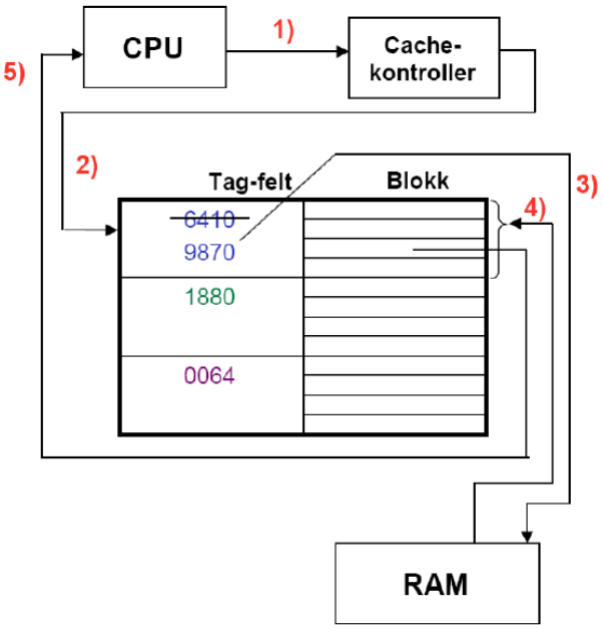
- Hvis data ikke ligger i cache, får man en cache miss

(1) CPU ber om innholdet i lokasjon 9872

(2) Cachekontrolleren finner ut at denne ikke finnes i cachen

(3) Blokken som starter på 9870 leses inn fra RAM

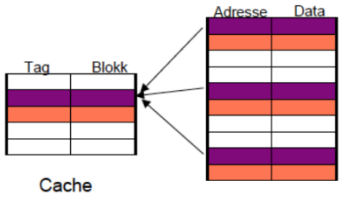
(4) Blokken som starter på 6410 overskrives for å gi plass til 9870



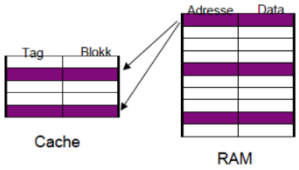
Set-associative cache:
En bestemt blokk fra RAM kan plasseres i et begrenset antall blokk-lokasjoner i cache. Fordelen med dette er at det er lett å sjekke om riktig blokk finnes i cache, trenger bare søke gjennom et begrenser antall tag-felt. Ulempen er at søketiden er lenger enn ved direct-mapped cache.

Full associative cache:
En bestemt blokk fra RAM kan plasseres hvor som helst i cache. Cachen utnyttes da meget godt, har minst miss chance av alle metodene. Ulempen er at søket for å finne en blokk kan ta tid- Man kan lage mekanismer for å forenkle søk, for eksempel hashing, men dette kompliserer cache-kontrolleren og krever ekstra hardware.

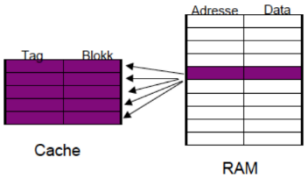
Direct-mapped Cache



Set-assosiative



Full-assosiative



Write strategier:

Hvis noe data i cache er endret og vi skal overskrive denne dataen står vi overfor en problemstilling. Vi kan ikke bare overskrive dataen.

Write through:

Etter hver gang det skrives til en blokk, skrives innholdet i blokken også tilbake til RAM. Dette er enkelt å implementere. Men

Write back:

Det drittet her forklarte Omid så jævlig rævva.

Overføring – hastighetsregning.

1 megabyte/sek = 8 Megabit/sek
1 Megabit/sek = 0,125 Megabytes/sek

For å finne ut hvor lang tid det tar å laste ned en gitt mengde data, må man dele datamengden på hastigheten

Datamengde (i megabit) / hastighet (i Mbps) = tid.

Vi skal laste ned en fil på 50 megabyte og nedlastingshastighet på internettforbindelsen er 20 megabit per sekund.

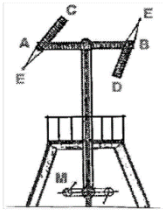
50 MB = 400 Mb.
400Mb / 20 Mbps = 20 sek

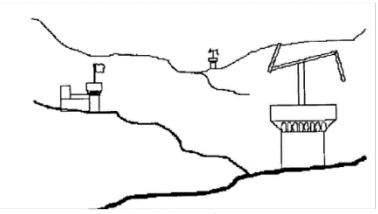
Altså det tar 20 sek å laste ned filen.

```
Mbps : Megabit per second (Mbit/s or Mb/s)
MB/s : Megabyte per second
1 byte = 8 bits
1 bit  = (1/8) bytes
1 bit  = 0.125 bytes
1 megabyte = 1000^2 bytes
1 megabit  = 1000^2 bits
1 megabit  = 0.125 megabytes
1 megabit/second = 0.125 megabytes/second
1 Mbps = 0.125 MB/s
```

Datanettverk – Historie

Allerede på 1700-tallet lagde man analoge signalnettverk. Semafor-telegraf var et nettverk i Frankrike som kunne brukes til å sende enkle beskjeder over lange avstander.







Morse-Telegraf:

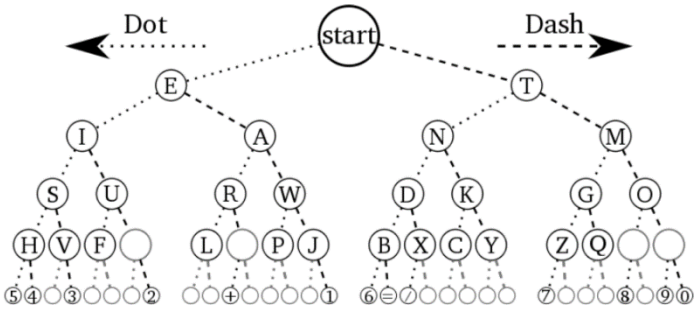
Kopperkabel som er koblet mellom to apparater. Kan trykke på en knapp og sende et kort eller et langt signal. Dette skrives ut på en lapp. Den amerikanske marinen brukte morse helt til 1999.

Kan vi kalle dette et kommunikasjonsnettverk?

Ja, store nettverk i forskjellige land. Flere noder som kobles sammen. Men hver node/telegrafsentral må sende signalet manuelt videre.

Grovt sett hadde morsenettverket en del til felles med dagens internett.

Kun en kunne sende av gangen.



Telegrafi benyttet seg av det vi kan kalle «message switching», sammenlignet med «pakke switching» i dagens internett. Beskjeden som sendes blir videresendt fra sender til mottaker. Det er ingen dedikert linje mellom sender og mottaker.

Telegrafi er en tilkoblingsløs tjeneste. Det vil si at når beskjeden sendes, så vet ikke avsender eller mottaker hvilke vei beskjeden vil ta. Er en telegrafstasjon nede kan beskjeden sendes en annen vei. Dette minner om dagens internett med tanke på «pakke switching».

Letter	Relative frequency in the English language
e	12.702%
t	9.056%
a	8.167%
o	7.507%
i	6.966%
n	6.749%
s	6.327%
h	6.094%
r	5.987%
d	4.253%
l	4.025%
c	2.782%
u	2.758%
m	2.406%
w	2.361%
f	2.228%
g	2.015%
y	1.974%
p	1.929%
b	1.492%
v	0.978%
k	0.772%
j	0.153%
x	0.150%
q	0.095%

Baudot-telegraf:

Tidsmultipleksing: en måte dele opp kommunikasjonen. For eksempel at hver person i samtalen fikk 10 sek, før neste person fikk 10 sek.

Problemet var at morsekoden er så variabel lengde. Man løste dette på at hver sending fikk en fast lengde på 5-bit. 5 bits grensen var på grunn av hardware-begrensninger.

Telefon:

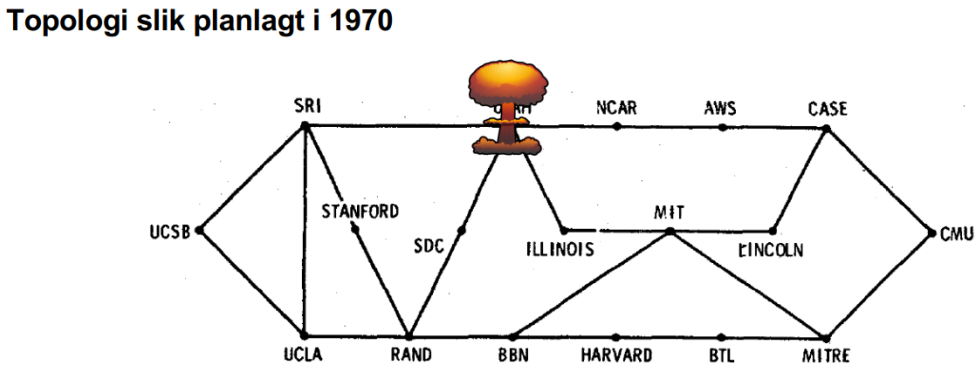
Første telefoner ble solgt parvis på 1870-tallet. Man måtte ha en dedikert linje i starten. Altså en fysisk kabel mellom telefon 1 og telefon 2. Hvis alle skulle kunne ringe alle måtte hver kunde ha forferdelig mange telefoner. Dette problemet ble løst med sentralbord (switch). Telefonen hadde nå en dedikert linje til sentralbordet. Sentralbordene hadde linjer til andre sentralbord. Sentralbordene var manuelt operert og de hadde fysiske kabler som måtte kobles sammen for at to telefoner skulle kunne kommunisere.

Etter hvert ble telefonsentralene automatisert.

Tilkoblingsorientert tjeneste. Kommunikasjonen følger alltid samme vei. Konseptet med å ha en dedikert linje mellom sender og mottaker er ikke lenger i bruk. Det tar opp mye kapasitet og er lite effektivt.

Internett:

ARPANET. Forgjengeren til internett. Kom på slutten av 60-tallet. Målet med ARPANET var i utgangspunktet lastbalansering. Man skulle kunne sende både programmer og data for prosessering på en annen maskin. Det var altså en tjeneste for utveksling av data.



Det var fysiske telefonlinjer mellom hver node i nettverket. Det skulle være minst to stier mellom forskjellige noder i nettverket, designet for å motstå atomangrep. ARPANET baserte seg derfor på pakke switching.

Norge var første tilkobling utenfor USA. Vi var koblet til med et satelittsignal.
Standardisering ble sentralt. Det var forskjellige aktører som ønsket å bruke nettet. Man måtte derfor samkjøre utviklingen av internett. Løsningen ble Request for Comments (RFC). De er fortsatt i bruk og i dag finnes det mange tusen RFCer.

Internetworking. Konseptet ble foreslått i 1973. målet var å koble sammen flere forskjellige nettverk. Man lagde grunnlegger som fremdeles brukes i dag. Best effort kommunikasjon. Bla bla. Internett skulle ikke bli kontrollert av ett land.

ARPANET klarte ikke tilfredsstille kravene om internetworking. TCP/IP presentert i 1974, bytestrømmer av data. ARPANET byttet til TCP/IP i 1983.

Siden 1980 har internett eksplodert. Web-tjenester, streaming, sosiale nettverk osv.

Andre typer nettverk:

Trådløse nettverk (WLAN). Standardiseringen av 802.11 startet i 1997.
Utfordring å finne ledige og åpne frekvensbånd i hele verden. Linklaget bygget for å være kompatibelt med Ethernet. Sikkerhet har alltid vært en stor utfordring.
Mobilnettverk har også hatt stor utvikling i senere tid.

Interconnect-nettverk. Høyhastighetsnettverk for å koble sammen maskiner. Har ikke klart å bli enige om en standard slik man har for internett. Ekstremt høy båndbredde og lav latens.

Ekstreme nettverk: NASA deep space network.

Vi må kunne snakke med satellitter og rovere sendt til andre planeter.
Har tre lokasjoner rundt i verden. USA, Spania og Australia. Da har man konstant kontakt med satellittene selv om jorden roterer.

Lagdeling i internettarkitekturen

Hva er en protokoll og hvorfor trenger vi det?

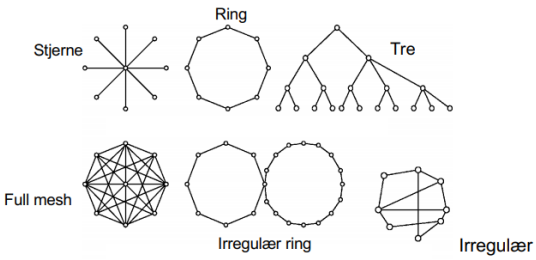
En protokoll definerer strukturen på beskjeder sendt over et nettverk. Generelt er en protokoll et sett med regler som beskriver noe. En slags generell oppskrift. Hvordan skal maskinvaren oppføre seg? Hvordan skal beskjeden finne frem? Er det noen garantier for levering? Hvordan håndtere kø, tap og andre problemer? Dette er nødvendig for å for eksempel å få en Iphone og en Android til å kunne kommunisere.

Endesystemer er for eksempel min telefon, laptop osv. Intermediate systemer eller mellomsystemer er for eksempel rutere.

Man kan bygge datanettverk på forskjellige måter. Vi har to grunnleggende **nettverkssystemer**:

Punkt til punkt:

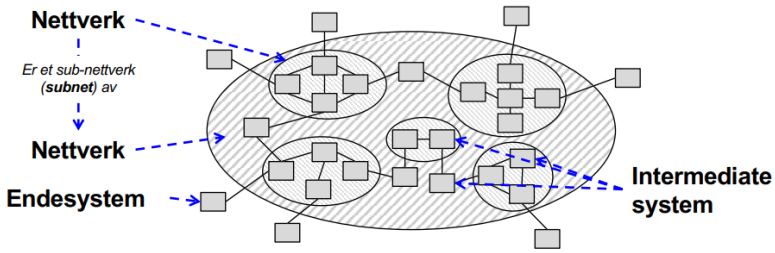
En til en overføring:



Broadcast-systemer:

En sender, mange lytter. WiFi.

Nettverkskomponenter



Endesystemer:

- Endesystemer finnes helt ytterst i nettverket
- For eksempler: datamaskiner, mobiltelefoner, sensorer, skrivere

Intermediate system:

- For eksempel:
  - ruter, switch
  - gateway
  - repeater, bridge

Nettverkstyper

Avstand mellom punktene	Lokasjon	Eksempel
0,1 m og lavere	Kretskort	Multi-core prosessorer
1 m	Systemer	NFC, BAN, PAN
10 m	Rom	LAN, SAN
100 m	Bygninger	
1 km	Campus	
10 km	Byer	MAN
100 km	Land	WAN
1.000 km	Kontinenter	
10.000 km +	Planeter	

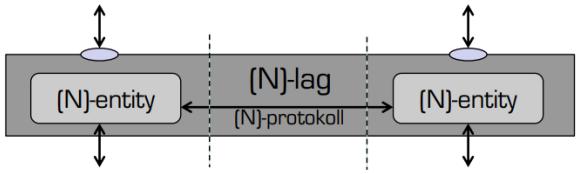
- *NFC*: near field communication, *BAN*: body area network, *PAN*: personal area network
- *LAN*: Local Area Network: IEEE 802.3 (Ethernet), IEEE 802.11 ("WiFi", "WLAN"), ...
- *SAN*: storage area network (iSCSI, NVMe-oF)
- *MAN*: Metropolitan Area Network: DSL, EPON, ...
- *WAN*: Wide Area Network: Frame Relay, SDH, ATM, optiske nettverk (WDM)
- Interplanetært Internett: http://www.ipnsig.org/

Protokoller og lag:

Utfordringen er at det potensielt kan være veldig komplisert med kommunikasjon til fremmede maskiner på nettet, interaksjon mellom forskjellige typer system og eller nettverk.

Vi introduserer derfor standardiserte abstraksjonsnivåer, se TCP/IP-modellen under.

Protokoll: Kommunikasjon mellom samme lag



Hva definerer en protokoll?

- *Definerer:*
  - Formatet på beskjeden og header (konvolutt).
  - Rekkefølgen på beskjedene.
  - Utvekslingen av beskjeder mellom to eller flere kommunikasjonssystemer.
  - Hva skal skje ved mottak eller sending av en beskjed.
- *Definerer ikke:*
  - Tjenestene tilbudt til laget over (N+1)
  - Tjenestene brukt i laget under (N-1)-SAP

TCP/IP modellen:

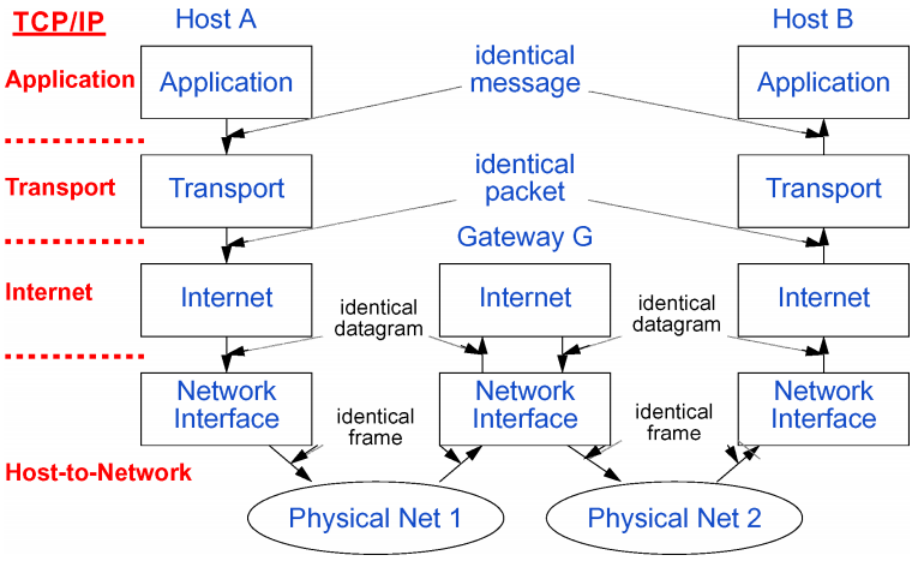
TCP/IP modellen er navnet på en gruppe med flere kommunikasjonsprotokoller som benyttes for å koble sammen datamaskiner i nettverk. Modellen består av flere protokoller, hvorav TCP og IP er de viktigste (eller meste kjente). TCP/IP har blitt standarden for sende data over nettverk. Fem lags referansemodellen. Noen slår sammen link-laget og det fysiske-laget.

Lag		Funksjon
5	**Applikasjon**	Applikasjonsrelaterte tjenester, og eventuell funksjonalitet fra OSI lag 5 og 6.
4	**Transport**	Kobler sammen systemene ende-til-ende (TCP/UDP)
3	**Nettverk**	Sende data fra ende-til-ende systemer (IP)
2	**Link**	Pålitelig overføring mellom to noder
1	**Fysisk**	Sender bit ut på mediet (kablet eller trådløst)

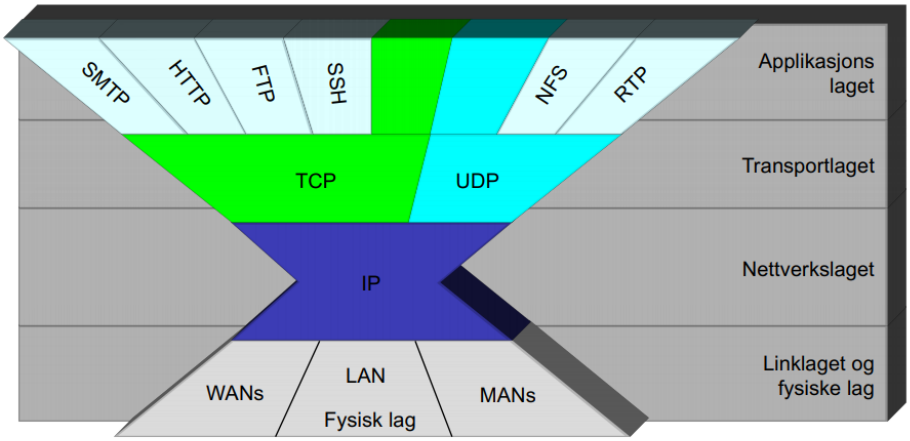
Dataenheter i TCP/IP-modellen

- "Beskjeder" fra applikasjonslaget defineres som «data units».
- Data units har forskjellige navn, avhengig av hvilket lag vi befinner oss i:
  - Segments: Brukes på transportlaget (kan inneholde fragmenter)
  - Packets: Brukes på nettverkslaget
  - Frame: Brukes på linklaget
  - Bits: Brukes på det fysiske laget

Kommunikasjon mellom en bruker A og en bruker B.

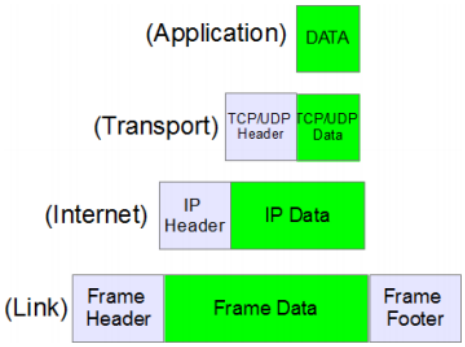


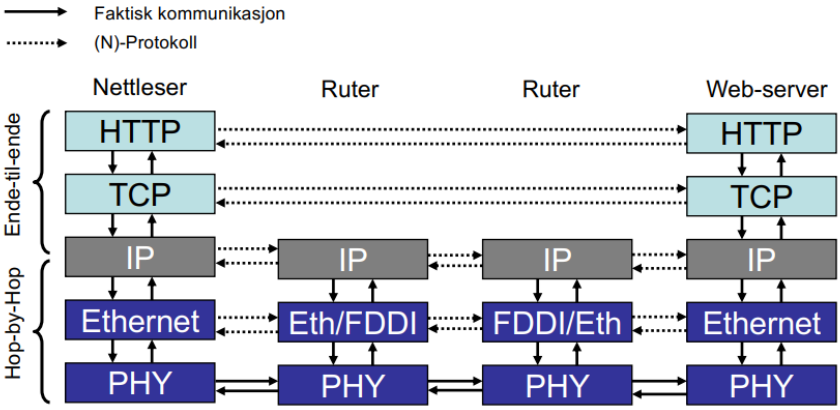
Det finnes forskjellige protokoller man kan bruke på hvert lag i TCP/IP modellen, se bilde til høyre.



Kallenavn: "Timeglass-modellen"

Nettverkslaget bruker alltid samme protokoll. Dette er fordi alle noder, både endesystemer og Intermediate systemer skal kunne kommunisere.





Vi kan tenke oss at dataen fra applikasjonslaget (som kan være data fra en nettleser, skype, netflix osv) sendes til transportlaget. Her legges dataen inn i en konvolutt med en TCP (eller UDP) header. Så sendes denne konvolutten til nettverkslaget der konvolutten blir lagt i en ny konvolutt med en ny header (IP-header). Så sendes denne konvolutten til linklaget/det fysiske laget der konvolutten blir lagt i en tredje konvolutt med en header og en tail/footer. Så sendes pakken fysisk til neste node, som enten sender pakken videre etter å ha pakket opp første del av konvolutten for å sjekke «IP-pakken», eller pakken åpner pakken i sin helhet hvis dette er endestasjonen.

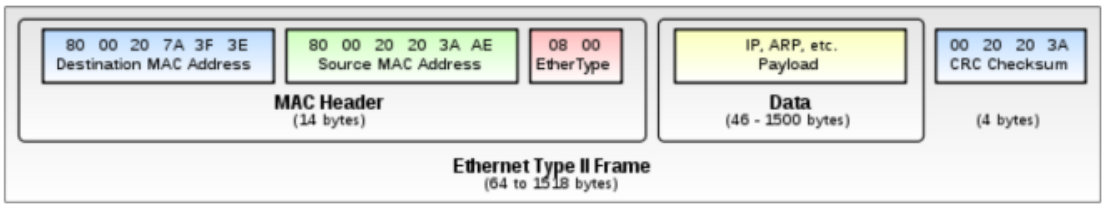
- Forespørselen går nedover lagene på "*nettleser*"-siden.
- Det fysiske laget håndterer selve sendingen til neste (nabo) node.
- Nettverksprotokollen (IP) tar hånd om rutingen til endelig destinasjon.
  - På hver ruter så går pakken opp til nettverkslaget for prosessering.
- Transport og applikasjonslaget kommuniserer ende-til-ende.

Lag 1 – Det fysiske laget.

Signalrepresentasjonen av bits. Sørger for at 1 bit blir mottatt som 1 bit. Sørger egentlig bare for at en strøm av bits blir sendt gjennom et medium og mottatt av mottaker som de samme bitsene.
Mekanikk: koblingstype, kabler..

Lag 2 – Linklaget

Pålitelig overføring mellom to enheter. Linklaget mottar bitstrømmen fra det fysiske laget og stykker det opp i frames.
Linklaget gjør noe feilretting. Har en enkel flytkontroll.
Bitsene deles opp i rammer, frames. Som skal sendes.
Ethernet og WiFi. Bruker en 6-byte adresse (48-bit) som ofte er lagret i nettverkskortet. En sjekksum kan sendes med for å bekrefte at dataen er riktig.



Lag 3 – Nettverkslaget

Kobler sammen ende-til-ende systemer i nettet.
Vet altså noe om hvilken vei en pakke må gå for å komme til endestasjonen.
Ruting/Routing. Den meste brukte versjonen av IP er IPv4. Den bruker en 32-bit adresse. Den nye versjonen IPv6 har 128-bit adresser.

Lag 4 – transportlaget

TCP. Oppsett av forbindelse (3 way handshake). Garanterer at pakkene leveres i riktig rekkefølge. Pålitelig, pakker sendes på nytt hvis kvitteringen (ACK) ikke kommer frem. Filoverføring, e-post osv.
UDP. Tilkoblingsløs forbindelse, sender bare ut data. Ingen garantier. Strømming av video og lyd.

- Bruker «port» som en unik identifikator.
  - Representeres med et 16-bit heltall

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

UDP Datagram Header Format

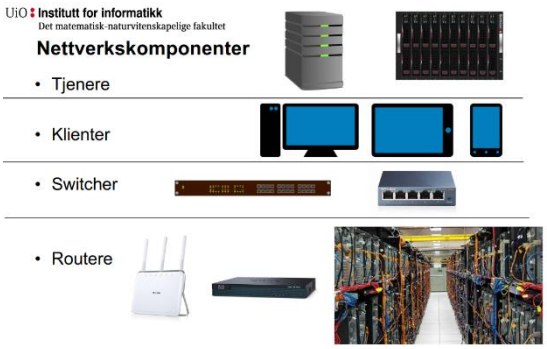
Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

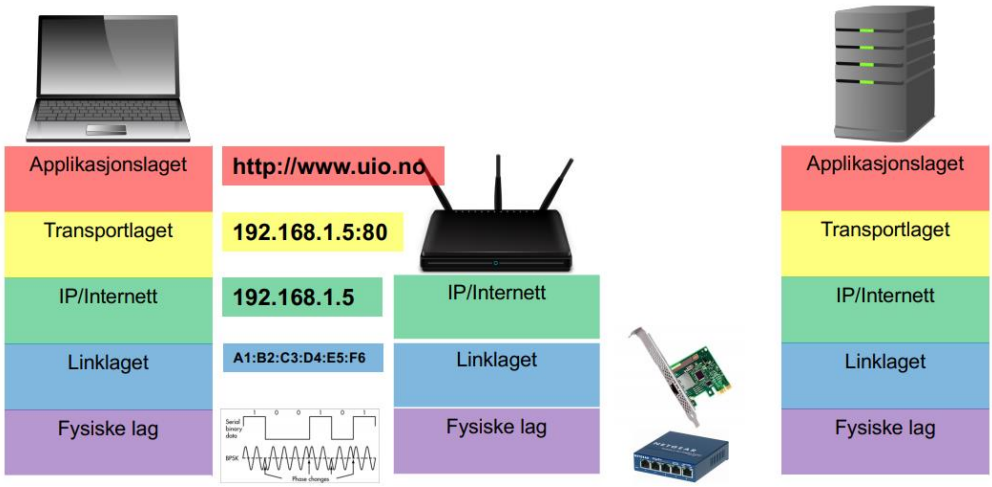
Lag 5 – applikasjonslaget

Lag med tjenester for applikasjoner.
Nettlesere, e-post, filoverføring osv.

Internettlagene spiller sammen

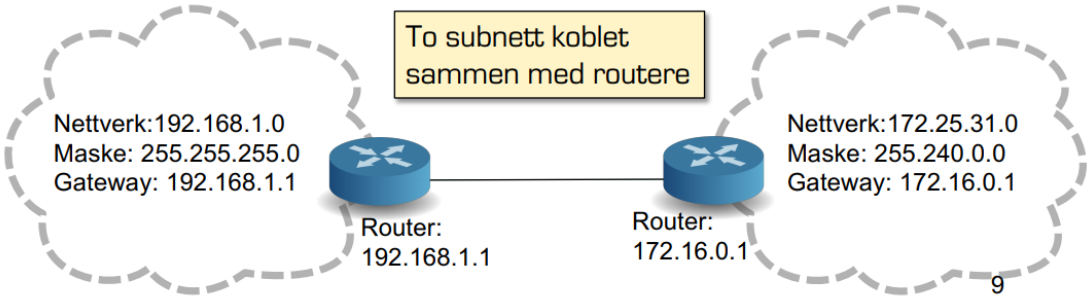
Internett kan sies å være en samling av små lokale nettverk som kobles sammen av rutere.



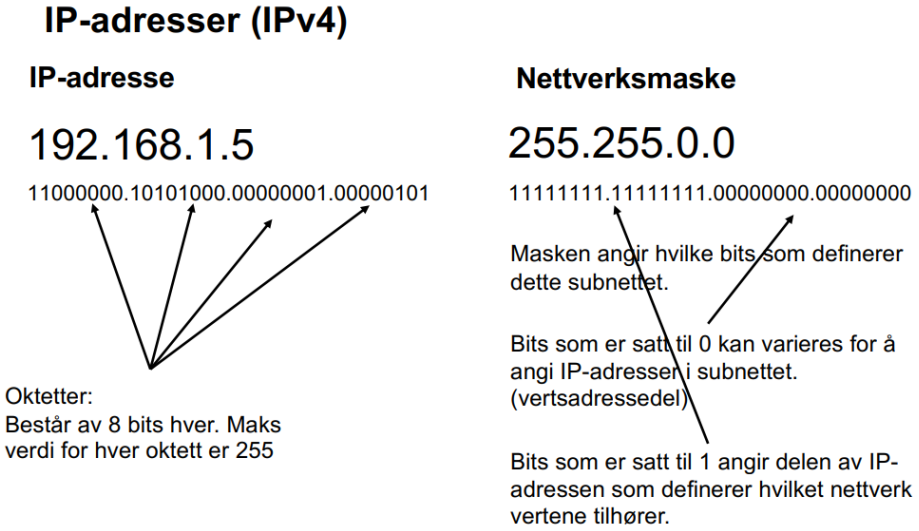


Hva er en IP-adresse?

En måte å sette en etikett på en datapakke som gjør at du kan nå en unik maskin av milliarder av maskiner.
En IP adresse kan være lokal eller offentlig, private eller public.
For å sende en pakke til en maskin utenfor det lokale nettverket må pakken sendes til en ruter som vet hvor den skal videresendes.



Nettverksmasken består alltid av en sammenhengende serie 1 bit og deretter en serie 0 bit.
Eks: 255.255.255.0.
11111111.11111111.11111111.00000000.
Nettverksmasken deler opp IP-adressen i nettverksdelen og vertsdelen.



Det er to vanlige måter å notere omfanget av et subnett.
Punktnotasjon:

192.168.1.0
Da må vi også oppgi nettverksmaske. 255.255.255.0

CIDR notasjon:

192.168.1.0/24
Vanlig punktnotasjon først, så et tall som angir hvor mange bits nettverksmasken består av

Kringkasting (send til alle):

En melding som sendes ut på en spesiell adresse.
Leveres til alle enheter som er koblet på samme LAN.
Nettverkslaget(MAC):
FF:FF:FF:FF:FF:FF

Regne ut subnettet fra en IP + nettverksmaske

En maskin i nettet har IP 192.168.1.5 = `11000000.10101000.00000001.00000101`

Nettverksmasken er 255.255.255.0 =`.11111111.11111111.11111111.00000000`

For å finne subnettadressen til maskinen må du gjøre en bitvis AND operasjon mellom IP-adressen og nettverksmasken.

11000000.10101000.00000001.00000000 = **192.168.1.0**

Dette er den første IP-adressen i subnettet og brukes til å identifisere subnettet.

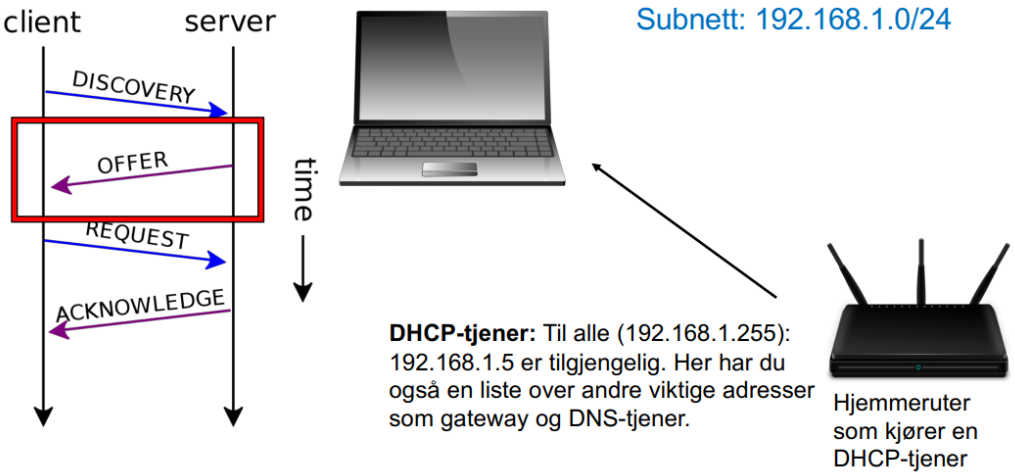
For en maskin på et subnett, finner du kringkastingsadressen ved å gjøre en bitvis OR-operasjon mellom maskinens IP-adresse og bit komplement (bitvis invers) av nettverksmasken.

- Eks: IP-adresse192.168.1.5 nettverksmaske: 255.255.255.0
- (192.168.1.5) OR (0.0.0.255) = 192.168.1.255

Et nettverk har altså en nettverksadresse som er den første adressen i nettverket.
Et nettverk har en kringkastingsadresse som er den siste adressen i nettverket.

Hvordan får maskiner en IP-adresse i første omgang?
DHCP – automatisk utdeling av IP-adresser.

Rutere støtter som regel DHCP. Når en datamaskin finner et lokalt nettverk, spør den broadcasting-adressem om det finnes en enhet som kan dele ut IP-adresser. DHCP tjeneren vil da tilby en ledig IP-adresse og komme med annen informasjon som gateway og DNS-tjenester. Datamaskinen godtar da tilbudet, hvoretter DHCP-tjeneren tildeler maskinen den aktuelle IP-adressen for et gitt tidsrom (ofte 24 timer).



ARP – koblingen mellom nettverk og IP. ARP er en protokoll.

Nettverkskortene har en 6 byte lang MAC-adresse som brukes til å identifisere maskinen innenfor et kringkastingsdomene. For at IP skal fungere må avsender vite hvilken MAC-adresse pakken skal sendes til. ARP kobler IP (nettverkslaget) og MAC (linklaget). En tabell, ARP cache, holder oversikt over korrelasjon mellom IP-adresser og MAC-adresser. Den enkelte datamaskin som er koblet på et lokalt nettverk har en ARP cache. Dersom kringkastingsdomenet inneholder for mange enheter, kan det by på problemer: ARP går til alle maskiner i kringkastingsdomenet. Om domenet er for stort kan dette stjele kapasitet som burde ha vært brukt til å overføre data.

En IP-adresse – mange porter.

Transportprotokollene implementerer porter som muliggjør totalt 65535 samtidige forbindelser på en IP-adresse.



Transportprotokollene (UDP, TCP) implementerer "porter" som muliggjør totalt 65535 samtidige forbindelser på én IP-adresse

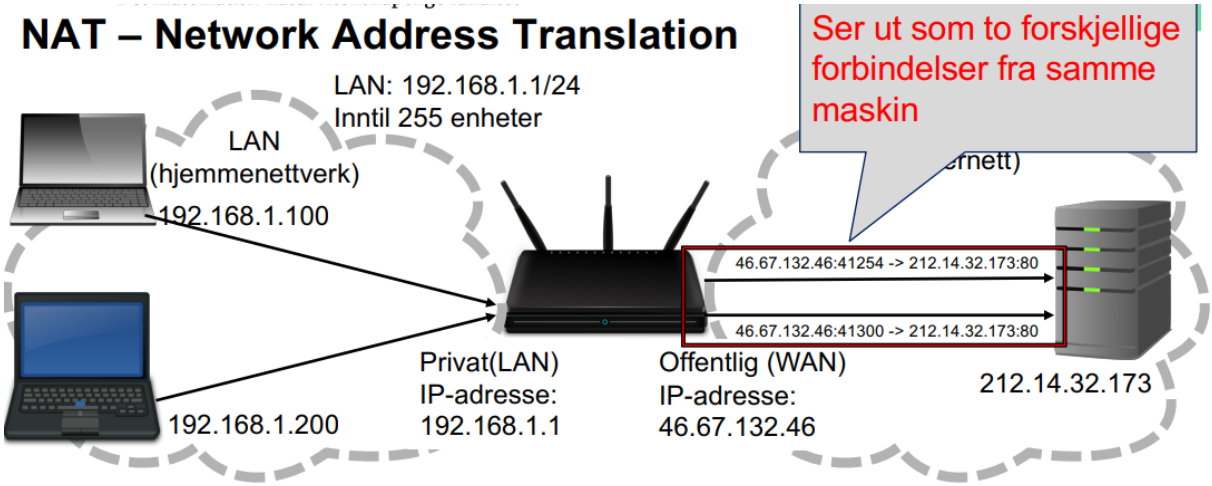
En vanlig måte å notere IP og port på er IP:port (eks.: 192.168.1.5:22)

Men du ønsker å kjøre mange tjenester på samme maskin.
Hvordan skiller man tjenestene fra hverandre?

IP: 192.168.1.5

Port	Tjeneste
0	Reservert
1	tcpmux
…	
22	SSH
…	
25	SMTP
…	
1024-49151	Brukerporter
49152-65535	Dynamisk / privat

Maks antall mulige IP-adresser på 32 bit (IPv4) er 2^{32}, eller nesten 4,3 milliarder. Anslag sier likevel at det er over 20 milliarder enheter tilkoblet internett i dag. Dette muliggjøres av NAT.



Kilde IP	Mottaker	Oversatt adresse
192.168.1.100	212.14.32.173:80	46.67.132.46:41254
192.168.1.200	212.14.32.173:80	46.67.132.46:41300

Private IP-adresser

Private IP-adresser er adresser som er reservert for bruk i lukkede nettverk og nettverk med NAT mot internett. Disse IP-adressene skal ikke være direkte koblet mot internett. En hjemmeruter er vanligvis satt opp til å gi deg LAN med et subnett fra en av disse segmentene:

RFC1918 name	IP address range	number of addresses	largest CIDR block (subnet mask)	host id size	mask bits
24-bit block	10.0.0.0 – 10.255.255.255	16,777,216	10.0.0.0/8 (255.0.0.0)	24 bits	8 bits
20-bit block	172.16.0.0 – 172.31.255.255	1,048,576	172.16.0.0/12 (255.240.0.0)	20 bits	12 bits
16-bit block	192.168.0.0 – 192.168.255.255	65,536	192.168.0.0/16 (255.255.0.0)	16 bits	16 bits

Ulemper med NAT. Fører til ekstra kompleksitet i nettverket. Ruteren må bevare tilstanden til forbindelsene. Nye forbindelser må initieres fra innsiden av NAT-nettverket. Gjør det vanskelig å koble til utenifra.

Selv med NAT i bruk på svært mange nettverk, er antall tilgjengelige IPv4 adresser i ferd med å bli kritisk lavt. Det er en lang prosess med å få IPv6 i drift ettersom protokollen må støttes i alle noder fra ende-til-ende.

Kommandoen «traceroute» (tracert i windows) bruker en protokoll som heter ICMP (Internet Control Message Protocol» til å spore hvor datapakkene er innom på vei til målet.

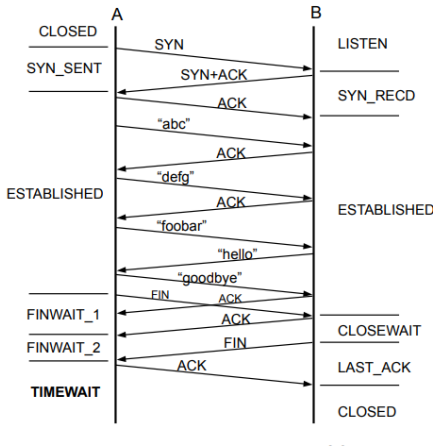
Ruting i Internett (svært kort)

- Målet for ruting: å videresende en datapakke slik at den til slutt når måladressen sin.
- En Internet Service Provider (ISP) driver et nettverk og leverer datatransport til andre ISPer og sluttbrukere.
- ISP Peering er når ISPer inngår avtaler om å videresende hverandres trafikk. Økonomiske prinsipper er da med på å bestemme hvor trafikken flyter.
- Border Gateway Protocol (BGP)– Rutingprotokoll som brukes mellom ISPer. Svært store selskaper kan også bruke BGP.
- OSPF – eksempel på rutingalgoritme for mindre nettverk. Bygger et kart over mulige ruter til måladressene. Skalerer ikke for store nettverk.

Transportlaget- TCP:

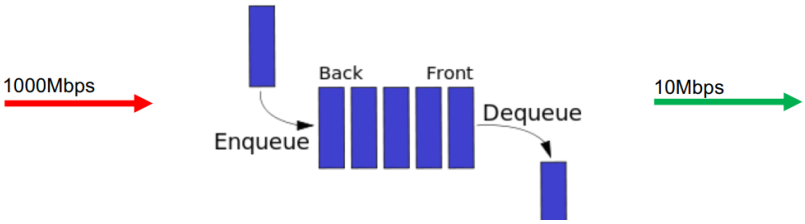
Transmission Control Protocol (TCP)

- Forbindelsesorientert
  - Settes opp ved et 3-veis-håndtrykk
    - SYN-SYN+ACK-ACK (se figur)
- Flytkontroll
  - Ikke sende fortere enn mottageren kan ta imot
- Metningskontroll
- Byte-strøm og levering i rekkefølge
- Pålitelighet
  - Implementert ved at bekreftelser på hver pakke sendes tilbake fra mottakeren
- Feilsjekking av nyttelasten (sjekksum)



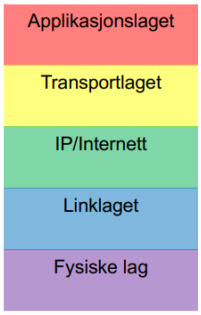
Metningskontroll:

- En ruter er i utgangspunktet bare en FIFO (first-in-first-out) kø.
- Om det blir for mye trafikk over en gitt kø, vil det føre til stopp i trafikken. Dette kalles ”congestion”. Om det er så mye traffikk at det blir full stopp for alle, kalles det “congestion collapse”.
- For å unngå dette bygde man inn mekanismer i TCP for å tilpasse senderaten til nettverksforholdene.



Dersom flere enheter ønsker å sende 10 Mbps med data vil serveren stykke dette opp. Ressursene deles da teoretisk likt, men ved å åpne flere forbindelser i parallell får man mer av kapasiteten.

Kryptering / sikkerhet i lagene



Secure Sockets Layer – kryptering for ende-til-ende
Applikasjoner – f.eks nettbank eller butikker.

F.eks. tcpcrypt– har som mål at alle TCP-forbindelser som settes opp skal være kryptert

VPN (IPSEC etc.) – kobler to subnett sammen så det fungerer som ett LAN selv om de er fysisk adskilt

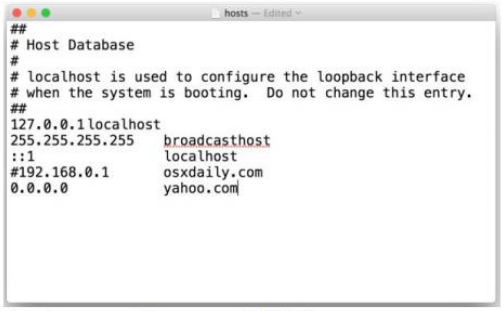
Kryptering på flere lag gjør det vanskeligere for uvedkomne å lytte til kommunikasjonen.
Adressen (avsender / mottakeren) er vanskelig å kryptere, da routere må vite hvor pakken skal leveres.

Hvordan koble til en annen maskin?

Kan bruke IP-adresse.

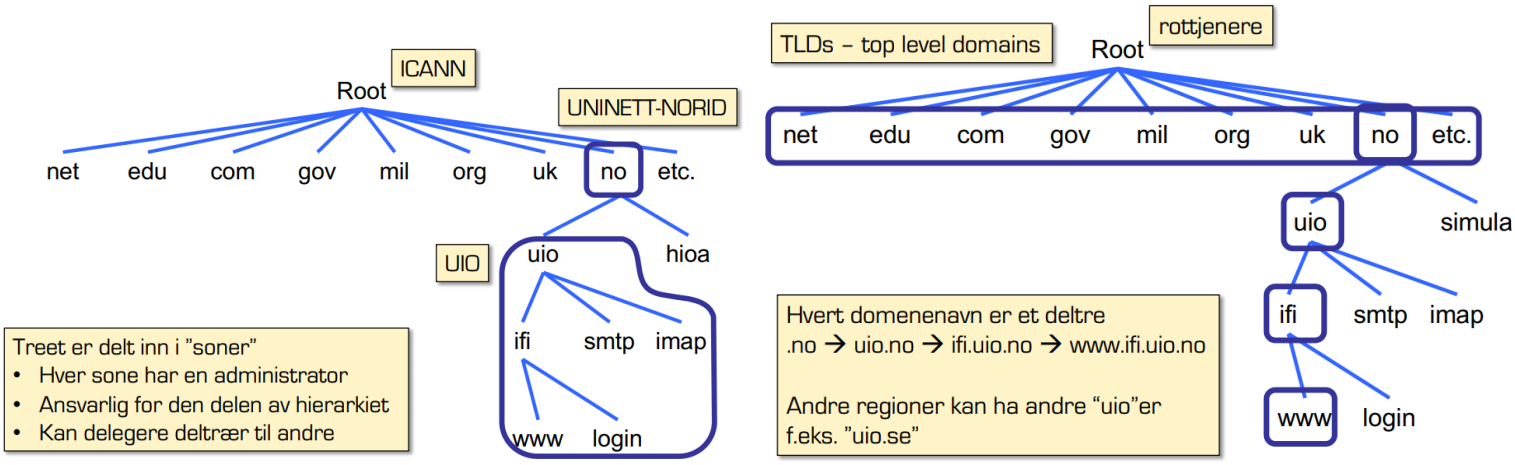
Med IP-adresse?

- `telnet 127.0.0.1 23`
  snakker med min egen maskin
  - Brukes ofte da det er en veldefinert adresse
- `wget http://173.194.39.31:80/`
  snakker med en av Google sine maskiner
  mulig å huske, men ikke praktisk.
- `ssh 9.228.93.3`
  forsøke å kontakte en maskin du visste hadde denne adressen i 2001
  umulig å huske om du ikke bruker den daglig



- det er mulig å lage alias for en IP-adresse i filen /etc/hosts (Unix variants)
- Opprinnelig administrert av Stanford Research Institute. Endringer ble distribuert via epost ☺

Dette er åpenbart tungvint. Derfor har vi Domain Name System.
Vi skriver vg.no i stedet for 195.88.55.16



DNS er en sentral del av internettet og leverer en måte å matche navn til IP-adresser. Hver DNS-tjener har visse funksjoner:

Autoritet over en del av hierarkiet

Lagre alle oppføringene for maskiner/domener i sin sone

Dette må replikeres for å sikre oppetid (minst 2 tjenere)

Må kjenne adressene til rottjenerne.

Rottjenerne kjenner til alle TLD (Top level domains)

Tjenester i internett

Nå er vi på applikasjonslaget.

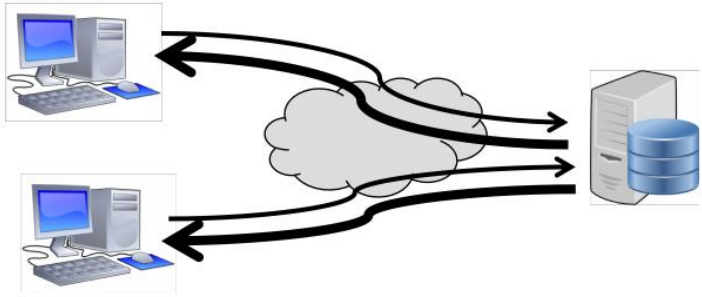
Forbindelsesstrategier:

Pull: en maskin ber en tjener om en tjeneste, for eksempel et dokument. Tradisjonelt den vanligste metoden.
Push: Tjeneren dytter en tjeneste til klienten. Krever at det er en forbindelse fra før, eller at klienten lytter.
Publish-subscribe: variant av Push der tjeneren dytter ut beskjeder til en gruppe av abonnenter.

Aksemodeller:

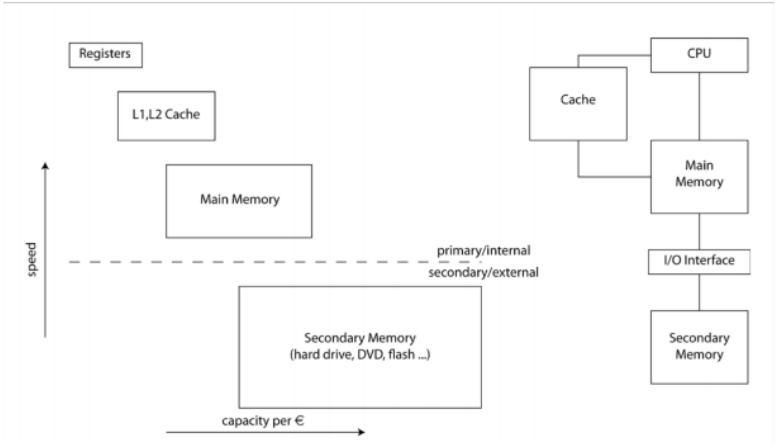
Klient-tjener: tradisjonell kommunikasjonsmodell. Klienter ber om en tjeneste (oppretter en forbindelse). Tjenere leverer tjenesten (svarer på forespørselen). Potensielt store kostnader med å få infrastrukturen skalerbar. Tjeneren er et «point of failure».

- Eksempler: Webklient (nettleser)/Webtjener, Mailklient/tjener, FTP (filoverføring)



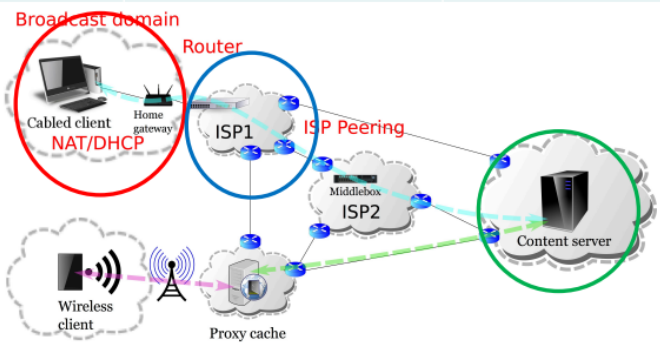
Peer-to-Peer (P2P): Det første velkjente programmet var napster. Brukt til fildeling. Ble erklært ulovlig. Etterfulgt av andre P2P tjenester som BitTorrent. Ideen er å unngå kontroll og sensur. Alle som bruker tjenesten samarbeider også om å levere tjenesten. Alle nodene på nettet vet om de andre nodene på nettverket som man må samarbeide med. Ikke noe point of failure. Problem er at maskinene er forskjellige.

- Dere husker cachehierarkiet fra Omids forelesning?



- Denne figuren kan utvides så den gjelder også utenfor datamaskinen

Cachenivå	Fysisk beliggenhet	Est. RTT
Lokal Proxy	Organisasjon / LAN	< 10ms
Content Delivery Network	ISP	< 50ms
Original datakilde	Internett	~10ms - ~250ms

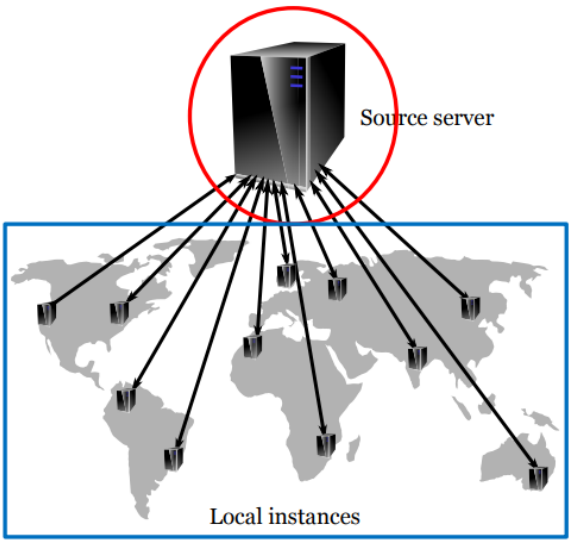


UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Content Delivery Network (CDN)

- Innhold som skal leveres til klienter over hele verden…
- …kan replikeres til maskiner som ligger fysisk nær brukerne.

- Koster ekstra maskinvare og lagringsplass
+ Sparer kapasitet i backbone-nettet
+ Gir lavere forsinkelse på forespørsler
+ Hindrer overbelastning av tjeneren



Figur fra: "B. Briscoe *et al.*, "Reducing Internet Latency: A Survey of Techniques and Their Merits," in *IEEE Communications Surveys & Tutorials*

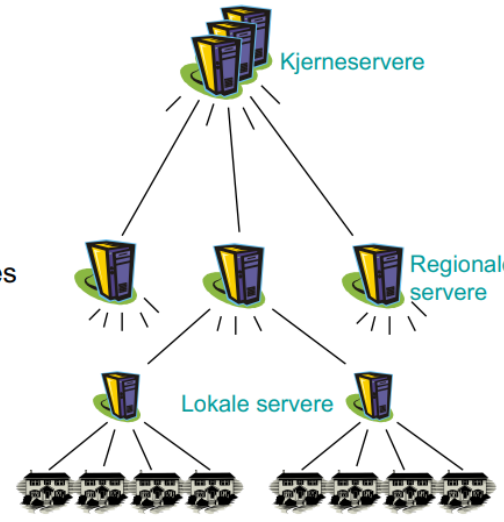
UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Hierarkisk innholdsdistribusjon med CDN

I stedet for å replikere hele innholdet i serverne til maskiner som ligger nærmere, kan man heller replikere deler av innholdet på hovedserveren. Man tar da typisk det mest populære innholdet. Vanlig hvis man har liten kapasitet med mange brukere. Typisk brukt på fly, der mange for eksempel besøker vg.no.

- **Hierarkiske systemer for å distribuere innhold**
  - Vanlig å "cache" populært innhold.

- Populært innhold blir lagret nær sluttbrukerne
- Innhold som ikke er så populært lagres enten på eller nær kjerneserveren
- Spesielt viktig hvis båndbredden er begrenset.
  - For eksempel: WiFi på fly



UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Proxy-cache

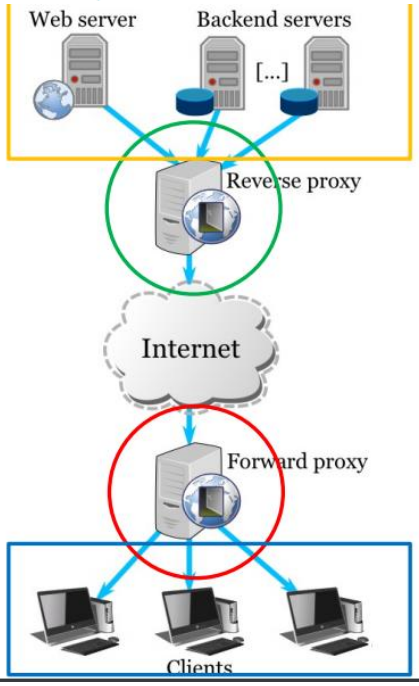
litt samme konsept. Forward Proxy er basically samme ting som CDN over.

Reverse Proxy er at man har en enhet som mellomlagrer data nær tjeneren. Dette er lurt hvis klienten henter data fra forskjellige andre klienter.

De fleste tjenester som leveres har en kombinasjon av forward-proxy og reverse-proxy.

- En "Forward Proxy" står nær klienten
  - og mellomlagrer data for klientene, slik at de ikke behøver å gå helt til kilden.
- En "Reverse Proxy" står nær tjeneren
  - og mellomlagrer data fra én eller flere tjenere, slik at klienten slipper å gå helt til kilden(e).

+ Lastbalansering
+ Sparer nettverkskapasitet
+ Lavere forsinkelse



Figur fra: "B. Briscoe *et al.*, "Reducing Internet Latency: A Survey of Techniques and Their Merits," in *IEEE Communications Surveys & Tutorials*

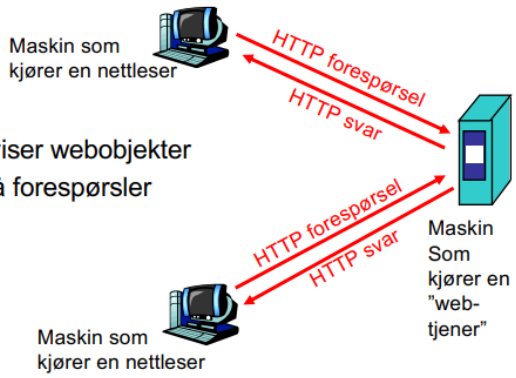
WWW og http-protokollen:

Hypertext transfer protocol er en protokoll på applikasjonslaget for surfing på web.
Klient/tjenermodell:

http bruker TCP som transport. Klienten oppretter en TCP-forbindelse (socket) til tjeneren, port 80. Tjeneren godtar TCP-forbindelsen fra klienten. http-meldinger (protokollmeldinger på applikasjonslaget) utveksles mellom nettleseren (http-klient) og webtjeneren (http-tjener). Til slutt lukkes TCP-forbindelsen.

HTTP: hypertext transfer protocol

- Applikasjonslagsprotokollen for Web
- Klient-/tjenermodell
  - *klient:* nettleser som spør etter, får og viser webobjekter
  - *tjener:* sender webobjekter som svar på forespørsler



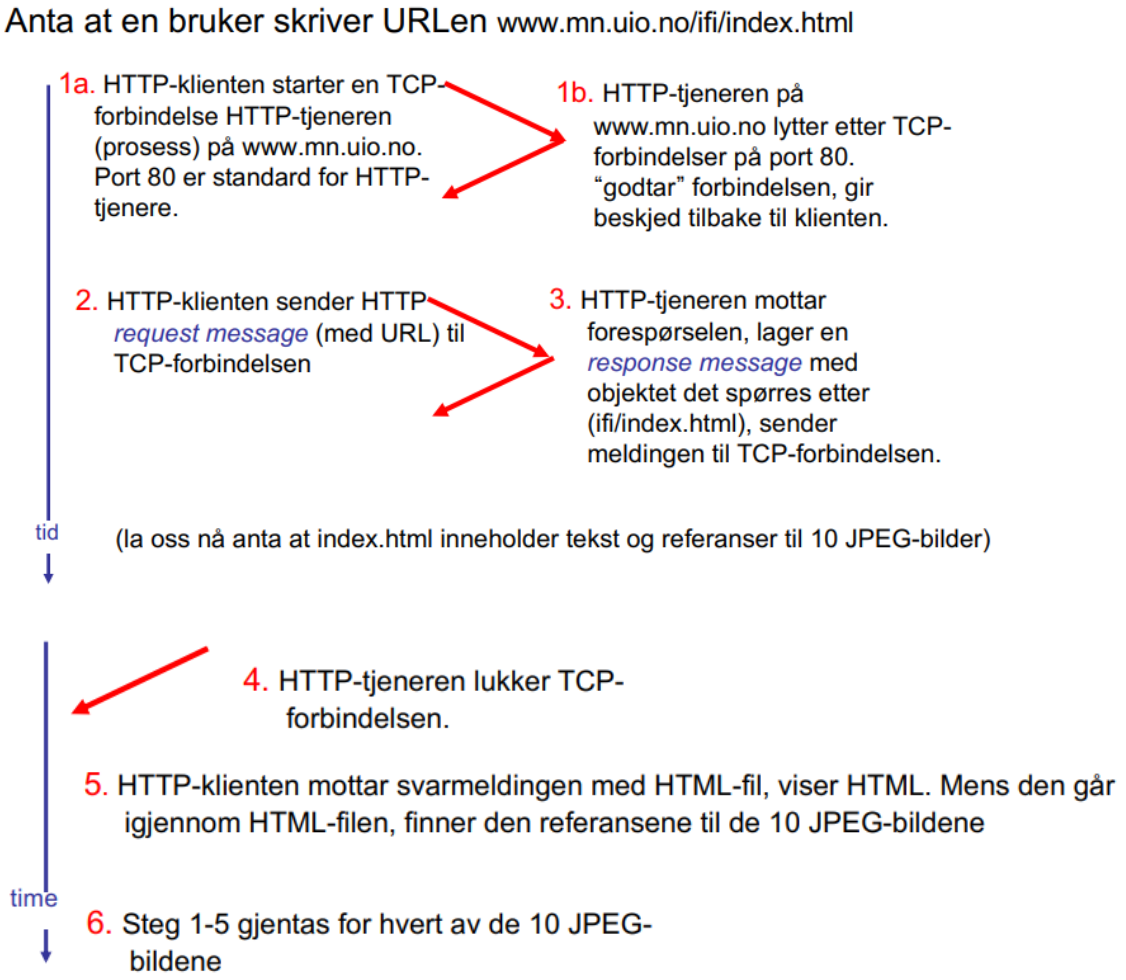
- Tre hovedversjoner:
- HTTP/1.0 (1990)
- HTTP/1.1 (1999)
- HTTP/2 (2015)

http er «stateless». Tjeneres sparer ikke på tilstandsinformasjon om tidligere forespørsler.

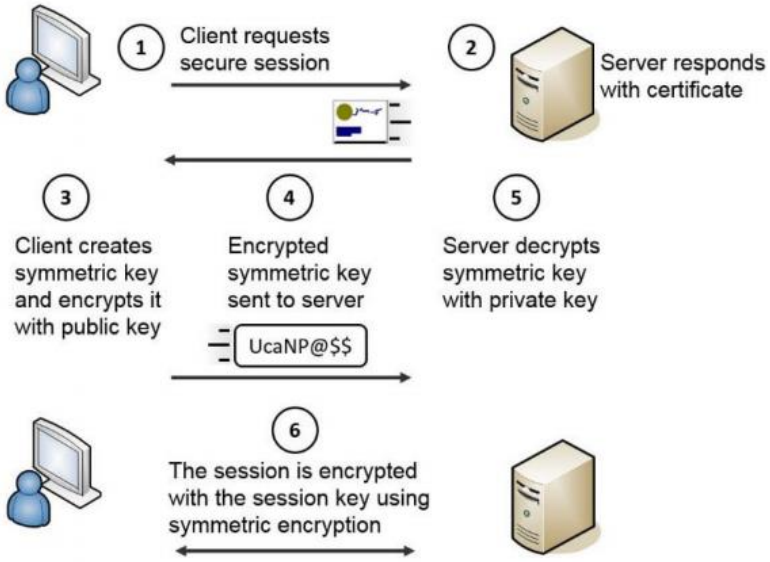
Protokoller som sparer på "tilstand" er komplekse!

- Tilstanden må vedlikeholdes
- Om en tjener eller klient "kræsjer", kan tilstanden bli ulik mellom dem. Da må den gjennoprettes.

http/1.0



http med SSL.
Klienten oppretter en asymmetrisk forbindelse for å dele en nøkkel som blir brukt til å opprette en symmetrisk kryptert forbindelse.



UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Persistente og ikke-persistente forbindelser

Ikke-persistent
- HTTP/1.0: tjeneren leser forespørselen, svarer, lukker TCP-forbindelsen
- 2 RTTer for å hente objektet
  - TCP-forbindelse
  - forespørsel og overføring
- Hver overføring lider av TCP sin gradvise økning i senderate (slow start)
- Mange nettlesere åpner flere parallelle forbindelser

Persistent
- Standard for HTTP/1.1
- over samme TCP-forbindelse: tjeneren leser forespørselen, svarer, leser ny forespørsel…
- Klienten sender forespørsler for alle de objektene den trenger så fort den mottar hoveddokumentet (HTML)
- Færre RTTer, mindre slow start

Persistent med "pipelining"
- Spør etter mange objekter på én gang (enda færre RTTer)
- Svaret kommer i serie etter hverandre i rekkefølgen forespørslene ankom tjeneren.

request

- To typer HTTP-meldinger: *request*, *response*
- HTTP request:
  - ASCII (lesbart av mennesker)

Request (kommandoer: GET, POST, HEAD) → **GET /ifi/index.html HTTP/1.0**

header:
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:no

Blanke linjer (return), indikerer slutten på Meldingen. → (ekstra return, ny linje)

response

status (protokoll statuskode statusfrase) → **HTTP/1.0 200 OK**

header:
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ……
Content-Length: 6821
Content-Type: text/html

data, f.eks. HTML-fil → **data data data data data ...**

HTTP/1.x response statuskoder (eksempler)

200 OK
- Forespørselen var vellykket, objektet kommer senere i meldingen

301 Moved Permanently
- Objektet har byttet plassering. Referanse til ny plassering kommer senere i meldingen.

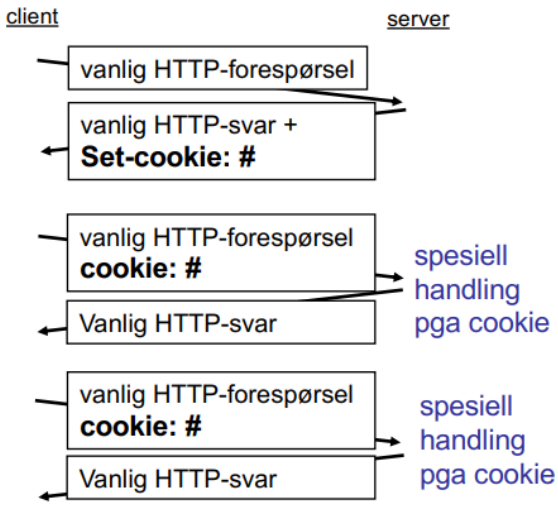
400 Bad Request
- Forespørselen var uforståelig for tjeneren

404 Not Found
- Dokumentet ble ikke funnet på tjeneren

505 HTTP Version Not Supported

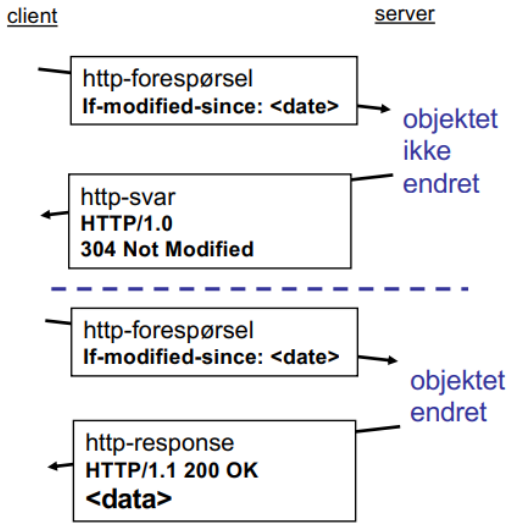
Cookies: Tjeneren oppretter et cookie-nummer når den svarer på en http-request. Dette nummeret blir da sendt med hver request fremover og tjeneren vet hvilken bruker det er snakk om og kan da levere brukertilpasset data.

- Tjeneren lager et cookie # , tjeneren husker #, senere brukt til:
  - autentisering
  - Huske brukerpreferanser, tidligere valg
    - produkter brukeren har sett på o.l.
- tjeneren sender cookien til klienten i svarmeldingen
  Set-cookie: 1678453
- Klienten legger ved cookien med etterfølgende forespørsler
  cookie: 1678453



Hvis man besøker vg.no flere ganger på få tid og innholdet ikke er oppdatert på siden, kan man hente informasjonen fra lokal cache i stedet for å hente data fra siden og øke nettrafikken.

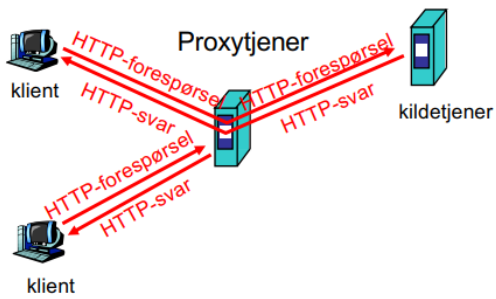
- Mål: ikke sende objektet om brukeren har en oppdatert versjon i lokal cache
- klient: oppgi tid/dato for cachet kopi i HTTP-forespørselen
  If-modified-since: <date>
- tjener: svaret inneholder ikke objektet dersom den cachede kopien er oppdatert:
  HTTP/1.0 304 Not Modified



En http-proxy tjener leverer svar til klienten uten å gå helt til kilden.

Mål: levere svar til klienten uten å gå helt til kilden

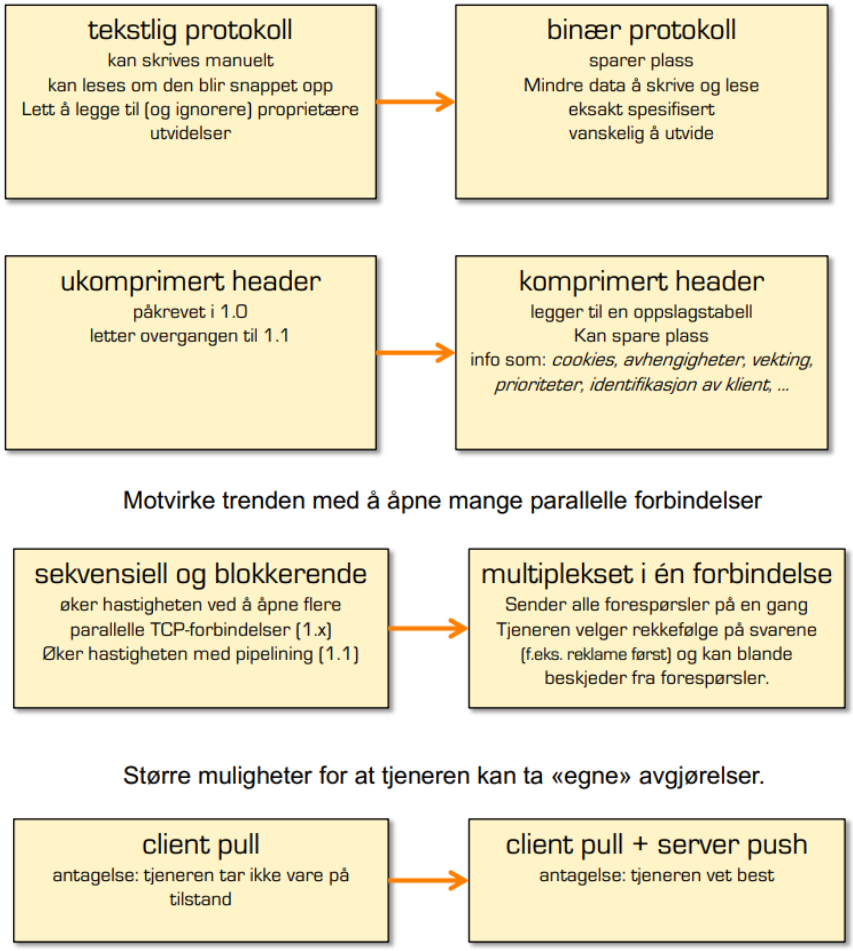
- Brukeren konfigurerer nettleseren: Web skal gå via proxytjener
- Klienten sender alle HTTP-forespørslene til proxytjeneren
  - Om objektet er i web cache: Proxytjeneren leverer objektet
  - Om objektet ikke er i web cache, sender Proxytjeneren en forespørsel til kilden og lagrer svaret før den sender svaret til klienten.



Antagelse: cachen er nærmere klienten (f.eks i samme nettverk) => raskere svar, mindre langdistansetrafikk

Endringer i http/2. http/1 http/2

Dette vil være den mest brukte protokollen i årene fremover.

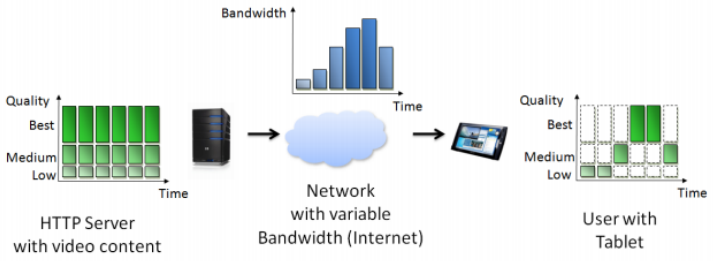


DNS prefetching for web. Sjekker igjennom alle linker og gjør DNS oppslag før du trykker på linken. Da sparer man tid.

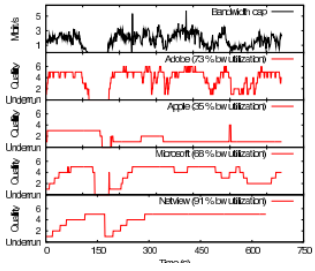
Videostreaming:

Videostreaming står nå for 58% av all nedlastingstrafikk på internett.
Netflix benytter seg av html. Altså TCP som transportprotokoll og ikke UDP som tidligere var det mest vanlige.

Dynamisk, adaptiv streaming over HTTP (DASH)



- Dele videoen i segmenter: fullstendig uavhengige små filmer
- Velge varighet på segmentene: 2-10 sekunder vanlig
- Velge antall kvalitetslag
- Velge tilpasningsstrategi
  - Klienten velger, ikke tjeneren
  - Strategien hos avspilleren er det som utgjøre forskjellen



Epost: denne delen var ikke med på forelesningen, så her er det bare slidesene fra forelesning.

- Hovedkomponenter
  - "mailklienter"
    Message User Agent (MUA)
  - "mailtjenere"
    Mottak av meldinger / Videresending av meldinger
    - Mail submission agent (MSA)
    - Mail transfer agent (MTA)
    - Mail delivery agent (MDA)
    - Mail retrieval agent (MRA)
    - Ofte realisert som én komponent kalt Message Handling Service (MHS)
- MUA
  - eller "epostleser"
  - Skrive, redigere, organisere og lese eposter
  - utgående, innkommende lagres på eposttjener

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

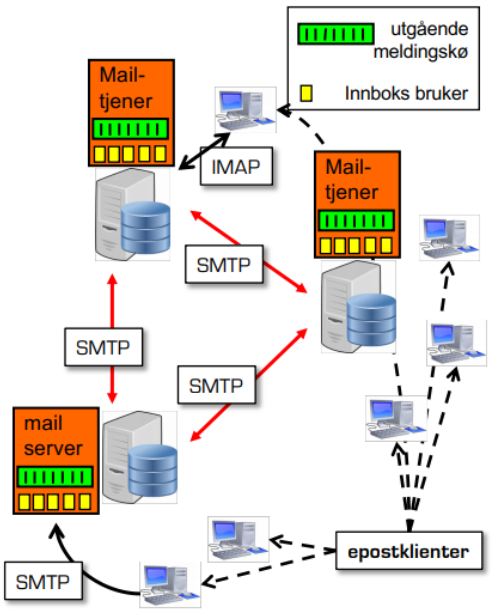
Epost: tjenere

Mailtjenere

- *mailbox* inneholder innkommende meldinger (hittil uleste) til brukeren
- *meldingskø* av utgående epostmeldinger (for sending)

Simple Mail Transfer Protocol (SMTP)

- Mellom eposttjenere for å sende epostmeldinger
- klient: sender av en epost
- tjener: den som mottar eposten



UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Epost: SMTP

- Bruker TCP til å, pålitelig, overføre epost fra klient til tjener.
- Standardisert port: 25
- Direkte overførsel: fra senderen til tjeneren som tar imot.
- Tre faser i overføringen
  - håndtrykk (greeting)
  - Overføring av beskjeder
  - avslutning
- Kommandoer/interaksjon
  - kommandoer: ASCII-tekst
  - svar: statuskode og frase
- Meldinger må være i 7-bit ASCII



Eksempel på interaksjon

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C:   How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```


SMTP: final words

SMTP bruker persistente forbindelser

SMTP krever (header & body) i 7-bit ASCII

Visse tegnkombinasjoner er ikke tillatt I meldingen (f.eks, `CRLF.CRLF`). Meldingen må derfor kodes (vanligvis i base-64 eller “quoted printable”)

SMTP-tjeneren bruker `CRLF.CRLF` for å avgjøre når meldingen slutter (ingen angivelse av lengde i header)

Sammenligning med HTTP/1.x:

- HTTP: pull
- SMTP: push
  - Frem til den siste tjeneren!
- Begge har ASCII kommando/svar interaksjon, statuskoder
- HTTP
  - Hvert objekt kapslet inn i sin egen svarbeskjed
- SMTP
  - Originalt som HTTP
  - nå: mange objekter sendt i meldinger med mange deler.

Datasikkerhet.

TSD – Tjenester for sensitive data. Hele forelesningen om TSD var vel egentlig en lang reklame om TSD, som er et system der man sikkert skal kunne lage sensitive persondata og gjøre ulike forskningsfokuserte ting med dataene.

Lovverk

- Lov av 14. april 2000 nr. 31 om behandling av personopplysninger (popplyl.)
- Forskrift av 15. desember 2000 nr. 1265 (personopplysningsforskriften)
- Lov av 18. mai 2001 nr. 24 om helseregistre og behandling av helseopplysninger (hlsregl.)

Tidligere lovverk. Nå er det sentrale GDPR:

GDPR: General Data Protection Regulation. Styrking av personvernet. Samtykke er det sentrale. Man er selv eier av egne personopplysninger. Eller, man kontrollerer sine egne personopplysninger.

TSD systemkrav:

Personvernforordningen artikkel 4:

Personopplysninger er «enhver opplysning om en identifisert eller identifiserbar fysisk person («*den registrerte*»); en identifiserbar fysisk person er en person som direkte eller indirekte kan identifiseres, særlig ved hjelp av en identifikator, f.eks. et navn, et identifikasjonsnummer, lokaliseringsopplysninger, en nettidentifikator eller ett eller flere elementer som er spesifikke for nevnte fysiske persons fysiske, fysiologiske, genetiske, psykiske, økonomiske, kulturelle eller sosiale identitet»

Systemkrav

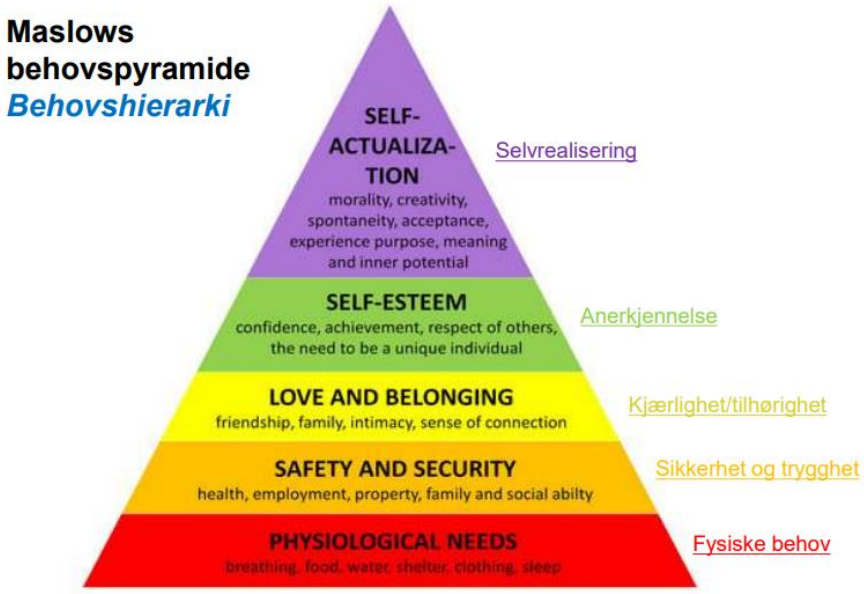
- IT-sikkerhet, isolasjon og tilgangskontroll ihht lov
- Ubegrenset lagring
- Flerprosjekt og flerbrukerløsning
- Tungregnekapasitet
- Høy båndbredde
- Enkelt vedlikehold og drift
- Enkelt å bruke (inkl video og lyd)
- Frihet i "user-space"
- BYOD tilgjengelig fra hvorsomhelst og hvasomhelst
- Tilgjengeliggjøre mest brukte software og referansedata
- Fleksibilitet for ny ønsker og behov
- Windows og Linux støtte på server-siden
- Muligheter for datainnsamling
- Muligheter for datadeling

Sensitive personopplysninger (kalt særlige kategorier i loven)

I loven er det definert en rekke kategorier av opplysninger som det skal mer til å kunne behandle enn andre opplysninger (artikkel 9 og 10):

1. opplysninger om rasemessig eller etnisk opprinnelse
2. opplysninger om politisk oppfatning
3. opplysninger om religion
4. opplysninger om filosofisk overbevisning
5. opplysninger om fagforeningsmedlemskap
6. genetiske opplysninger
7. biometriske opplysninger med det formål å entydig identifisere noen
8. helseopplysninger
9. opplysninger om seksuelle forhold
10. opplysninger om seksuell legning
11. opplysninger om straffedommer
12. opplysninger om lovovertredelser

Informasjonssikkerhet og sikkerhetsmål



Hva er sikkerhet?

«Sikkerhet kan defineres som en tilstand; fravær av uønskede hendelser eller frihet fra fare og frykt. Denne tilstanden er imidlertid ikke statisk, men påvirkes av endringer i faktorer som trussel og farer, sårbarhet og verdi.

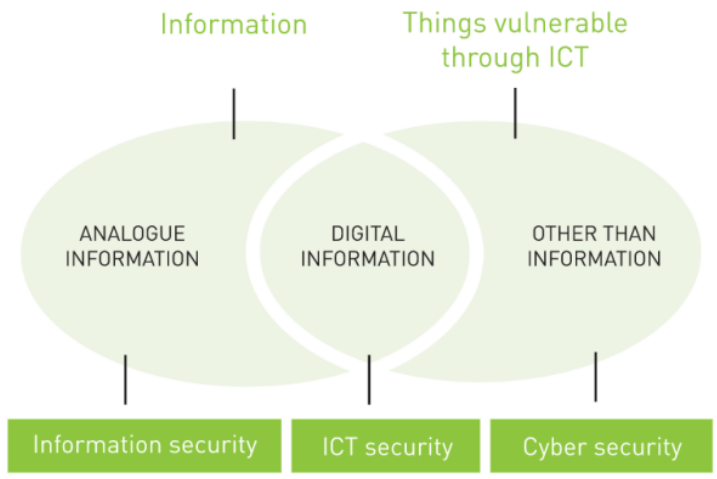
Ordet brukes også om tiltakene som benyttes for å oppnå denne tilstanden, samt for virksomheter eller enheter som utfører denne typen tjenester internt i egen virksomhet eller for eksterne kunder. Tiltakene kan deles opp i tre kategorier; teknologiske, organisatoriske og menneskelige. Tiltakene kan videre deles opp i forebyggende- og skadereduserende tiltak.»

Store norske leksikon, snl.no

Verdier. Hva som er en verdi kan variere fra person til person. Sikkerhet handler om å beskytte verdier mot skade og ødeleggelse. Verdi har i denne settingen en svært vid betydning. Hva som utgjør en trussel kan defineres ulikt i ulike situasjoner.

Cybersikkerhet. Beskytte ting som er sårbare via IKT.
Informasjonssikkerhet. Beskytte informasjon og/eller informasjonsressurser (IT-utstyr, infrastruktur, datafiler, programvare)

Informasjonssikkerhet vs cybersikkerhet



Overlappet mellom cyber- og informasjonssikkerheten kommer i den digitale informasjonen.

Hvorfor er vi opptatte av sikkerhet?

Hva hvis alle kan lese eposten din, eller dokumentene dine?
Eller noen kan endre karakteren din fra B til F.
Noen manipulerte signalsystem for togtrafikk.

Digitalisering er viktig, men like viktig er det at sikkerheten holder tritt med utviklingen. Det vi gjør på nett har konsekvenser i våre virkelige liv. For at ny teknologi skal bli tatt i bruk må brukerne ha tillit til systemet.

Digitale trusler øker. Datakriminalitet, sabotasje, industrispionasje, etterretning.
Digital sårbarhet:

Rett og slett en feil i et IKT-system. Svakhet, feildesign eller feilimplementering.
To kategorier:
Kjente og aksepterte.
Ukjente eller feilvurderte, og dermed heller ikke håndterte.

Trusselaktør:

Den som ønsker å utføre noe potensielt skadelig
Innsidebrukere utgjør ofte den største trusselen, altså ansatte, admins, tjenesteleverandører, kunder, familiemedlemmer osv.
Hvilke trusselaktører vi bryr oss om vil variere.



Drivkraft bak angrep kan være økonomisk vinning, tilgang til hemmeligheter, svekke konkurrenter eller stater, politiske interesser, ideologi, anerkjennelse. Datakriminaliteten øker i dag.

Er de fleste angrep tilfeldige eller målrettet? Stort sett er de fleste angrep tilfeldige. Tilfeldige personer er målet.

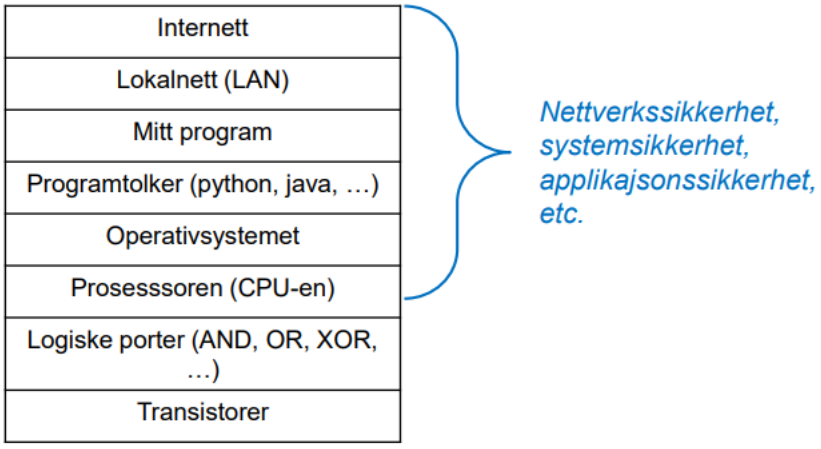
Nye tjenester fører til nye trussler, eksempelvis AMS som er smarte strømmålere:

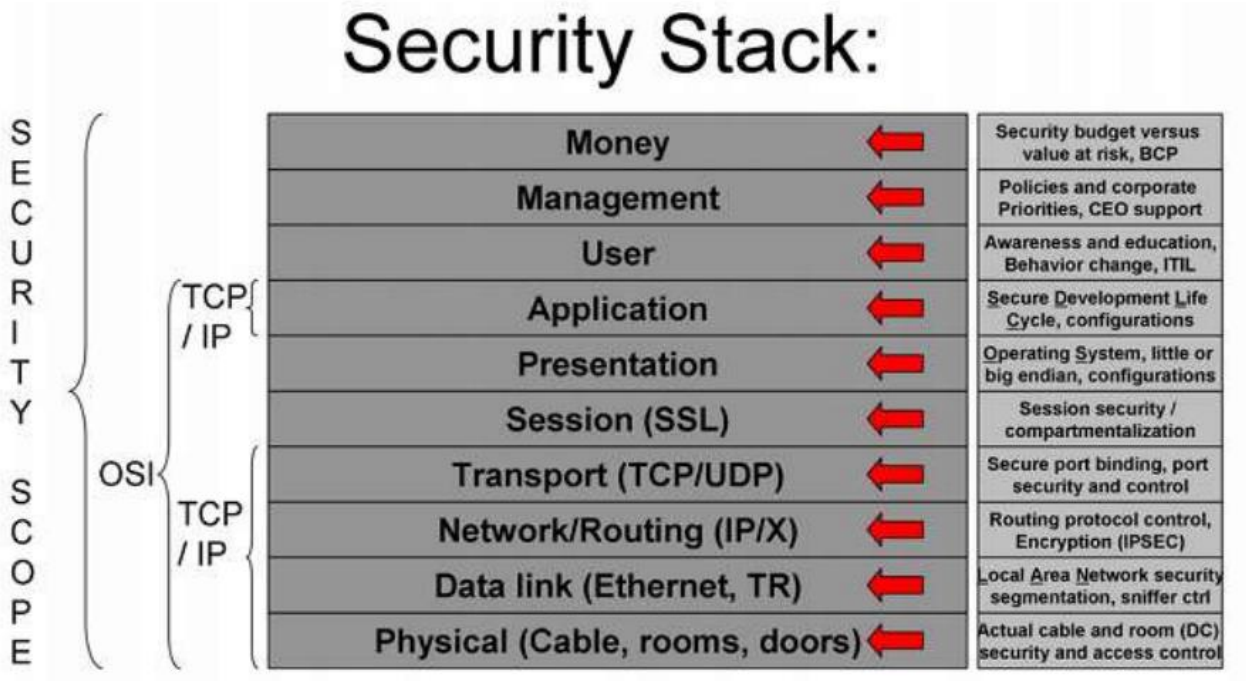
Formålet er bedre informasjon til sluttbruker om strømbruk, mulighet for styring av strømforbruk osv. De opererer over nett.
Samler data som potensielt kan si mye om folks vaner.
Kan måledata manipuleres?
Kan uvedkommende tilegne seg adgang til styringsenhetene?

Økende behov for informasjonssikkerhet

- Ville det ikke være mulig å løse alle sikkerhetsproblemer en gang for alle? **Nei,** fordi:
  - Hurtig innovasjon i nye IT-produkter og tjenester skaper også nye sårbarheter og insentiver for angrep
  - Forbrukere rangerer ny teknologi framfor sikker teknologi
  - Flere og flere tjenester er eksponert online
  - Informasjonssikkerhet ignoreres ofte når utviklere har tidspress
  - Kriminalitet følger nye tjenester og forretningsmodeller
  - Mer effektive angrepsverktøy utvikles
  - Økende antall trusler
- Konklusjon: Informasjonssikkerhet er en kontinuerlig prosess for å oppdage og hinder trusler og å fjerne sårbarheter

Hvor plasserer vi sikkerhet?





Sikkerhetsmål og sikkerhetstiltak

- *Sikkerhetsmål*
  - En egenskap vi ønsker å oppnå
  - Er uavhengig av spesifikk implementering.
  - Kan ivaretas med ulike sikkerhetstiltak.

- *Sikkerhetstiltak*
  - Tiltak for å oppnå/ivareta sikkerhetsmål
  - Er basert på spesifikk implementering
  - Benyttes for å forebygge, å oppdage, eller gjenopprette



Sikkerhetstiltak: i ulike tilstander

Informasjon kan befinne seg i ulike tilstander

Under lagring, under overføring, når den er i bruk.

Informasjon må beskyttes i alle tilstander.

Sikkerhetsmål for informasjonssikkerhet:

- **Konfidensialitet**
- **Integritet**
- **Tilgjengelighet**

Opprinnelige definisjoner (ISO 27000)

- **Autentisering**
- **Uavviselighet** *(non-repudiation)*
- **Sporbarhet, etterrettelighet** *(accountability)*

- **Personvern?** GDPR…

Konfidensialitet

Å sikre at kun de med rettmessige behov har tilgang til en ressurs.

Sikkerhetstiltak: Kryptering, tilgangskontroll, skallsikring.

Integritet

Å sikre at en ressurs ikke endres eller slettes utilsiktet.

Sikkerhetstiltak: Tilgangskontroll, endringskontroll, kryptografiske sjekksumalgoritmer (hash), skallsikring.

Tilgjengelighet

Å sikre at en ressurs er tilgjengelig for rett person til rett tid.

Sikkerhetstiltak: sikkerhetskopier, redundante systemer (backup-system som kan settes inn for å ivareta oppetid, altså identiske system), gode rutiner for hendelseshåndtering og gjenoppretting.

Autentisering

Legitimere at noe eller noen er det eller de det utgir seg for å være. Trenger ikke være en person, men kan også være autentisering av et dokument eller en nettside.

Brukerautentisering, autentisering av dataopprinnelse.

Uavviselighet

Hindre mulighet for fornektelse av sending og mottak av meldinger. Bevis for at data er sendt, bevis for at data er mottatt.

Kan realiseres ved bruk av digitale signaturer. Asymmetrisk kryptering og offentlig nøkkel.

Sporbarhet

Å kunne knytte en gitt identitet til en gitt hendelse.

Ivaretas gjennom logging av alle hendelser, indentifisere alle identiteter, skallforsvar (ikke gi uvedkommende tilgang)

Ved innbrudd vil en angriper mest sannsynlig forsøke å fjerne spor.

Personvern?

Sosial manipulasjon.

Phishing (nettfisking)
Lure offeret med falske nettsider i håp om å tilegne seg informasjon (passord, bankinfo osv.)

Spoofing
Forfalske avsender. SMS, epost.

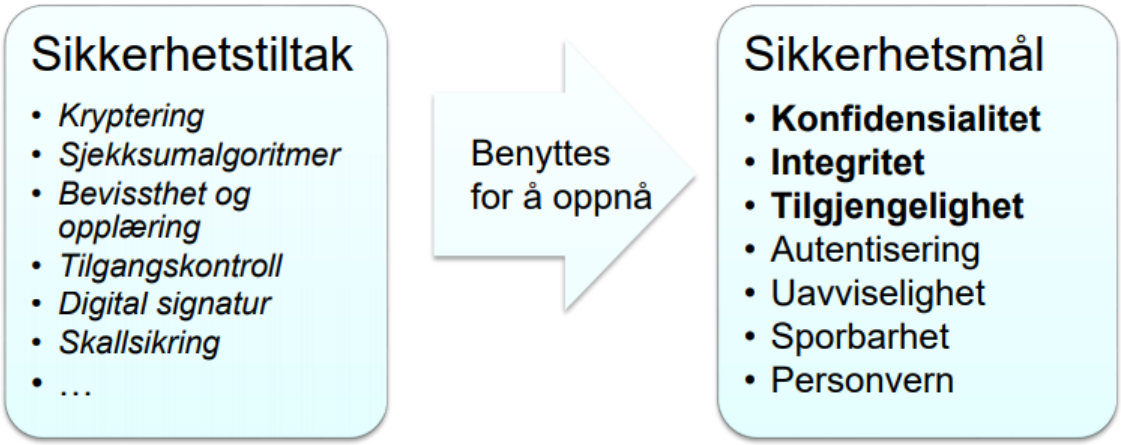
Sosial manipulering (Social engineering)

- Utnytter menneskelig svakhet. Tillitt, tilgang, svindel.
- Mest brukte, enkleste, og kanskje det eldste trikset…
- Berører alt fra privatpersoner til store virksomheter.
- Vanlige innfallsvinkler:
  - E-post som inneholder falske lenker eller skadevare:
    - Fra noen du kjenner
    - Fra kjent virksomhet (Banken din, Posten, arbeidsgiver, etc)
    - Spiller på følelser; Glede eller frykt.
  - Du kontaktes av noen som ber om informasjon (passord, kontoinformasjon) eller å gi dem tilgang til en ressurs (fysisk, et system eller datamaskinen din).

Bevvistgjøring, opplæring av mennesker er et viktig sikkerhetstiltak. Sjekk avsender på epost. Vær varsom med å følge lenker i e-post.

Repetisjon: Informasjonssikkerhet

- Beskytte *digital* informasjon og verdier.
- Beskytte *informasjonsressurser* mot skadelige hendelser, både tilsiktet og utilsiktet.



Autentisering

Autentisering av dataopprinnelse: Å verifisere at en melding/data kommer fra en hevdet sender, og at innholdet ikke er blitt endret underveis. Ivaretas gjennom bruk av kryptografiske protokoller, for eksempel digital signatur eller Message authentication code.

Organisasjonsautentisering: å verifisere at en virksomhet er den den utgir seg for å være. Ivaretas gjennom kryptografiske protokoller som TLS og PKI.

Systemautentisering: å verifisere at et system er det det utgir seg for å være. Ivaretas gjennom kryptografiske protokoller som PKI, IPSec.

Autentisering

- Identitet: Den eller det noen eller noe er.
- Identifisering: Å gi seg til kjenne. Å etablere eller indikere hvem eller hva noen eller noe er. Entydig?
  - Vanligvis ikke hemmelig
  - Eks: Navn, brukernavn, IP-adresse, IMEI, MAC-adresse, osv.
- Autentisering: Å bekrefte en hevdet identitet.
  - Personautentisering (user authentication)
  - Organisasjonsautentisering (organization authentication)
  - Systemautentisering (system authentication)
  - Autentisering av dataopprinnelse (data-origin authentication/ message authentication)

Personautentisering: en prosess som verifiserer en hevdet identitet for en (lovlig) bruker som ber om tilgang til et system eller et program. Eks: logge inn på UiOs epost-tjeneste, brukernavn (identitet) og passord (autentifisering).

Autentisering av attributter: det kan være behov for å autentisere en attributt, i stedet for en identitet, for eksempel om personen er over 21 år, er student osv.

Autentiseringsfaktorer

- Noe du vet: Passord, kode
- Noe du har: Kodebrikke, SIM-kort, nøkkel
- Noe du er: Fingeravtrykk, iris-mønster, ansikt

"Always remember the three main authentication factors: what can be easily guessed, what can be left in a cab, and what can be chopped off"

@m8urnett

Noe du vet:
Passord/-fraser og koder er den vanligste autentiseringsfaktoren. Passord er lett å glemme, lett å gjette, blir kanskje lagret på disk eller i minnet på en datamaskin. Lengde, sammensetning av tegn, bruk av ord og setninger er viktig. Unike passord for ulike sider.

Noe du har:
Nøkkel til lås. Engangspassord: kodekort, minnepenn. Studentkort. Kryptografiske protokoller. Utfordringen er at dette er noe som lett kan mistet og derfor lett kan misbrukes.

Noe du er:
Ansiktsgjenkjenning. Fingeravtrykk, iris. Utfordring: er det unikt nok? Sikkert nok? Behov for ekstra utstyr. Hvor ressurskrevende er sammenligningen?

Passordhash:

Passord lagres (forhåpentligvis) kryptert som et passord-hash.
Bruker «salting» for å unngå at to brukere får samme hash hvis de har samme passord. Man legger da altså på noe til passordet før det kvernes i hashalgoritmen og man får en hash som lagres.

Passordknekking.

Kan foreta intelligente søk basert på kunnskap om deg.
Kan brute force.
Utnytte svakheter i hashalgoritmen.

To-faktor autentisering.

Ofte en kombinasjon av noe du vet, og noe du har (engangskode),
Øker sikkerheten betraktelig.

Autentiseringsløsninger:

Aktører tilbyr elektronisk identitet og autentisering som en tjeneste. Eksempler: Weblogin, Feide, Google, Facebook, BankID, MinID.
Ulempe: knekkes passordet har angriper tilgang til flere nøkler.

Kryptering og PKI

Hovedmål: Sikre konfidensialitet og/eller integritet
Brukes også til autentisering av dataopprinnelse og for å oppnå uavviselighet
Krypteringsformer:

Hash-algoritmer (enveis-kryptering).

Symmetrisk (en delt nøkkel)

Sikrer ikke uavviselighet. Avsender kan si at det er mottaker som har kryptert meldingen og sendt meldingen.

Asymmetrisk (nøkkelpar: offentlig + privat nøkkel)

Sikrer uavviselighet, så lenge man vet at den offentlige nøkkelen er fra personen man skal ha en beskjed fra. Vi bruker derfor sertifikater, se PKI.

Hvis jeg vil sende en kryptert melding til en person, så «låser» eller krypterer jeg meldingen ved hjelp av vedkommendes offentlige nøkkel. Det er da bare vedkommende med sin private nøkkel som kan åpne meldingen.

Asymmetrisk kryptering og sertifikater

Public key infrastructure (PKI). PKI er et rammeverk for kryptering, autentisering og signering av dokumenter eller programvare.
Mål: tilby mekanismer for å sikre pålitelige forbindelser.
PKI er basert på asymmetrisk kryptering, med et unikt nøkkelpar per identitet.
Utfordring: hvordan stole på at en offentlig nøkkel er ekte? Benytter seg av sertifikater.

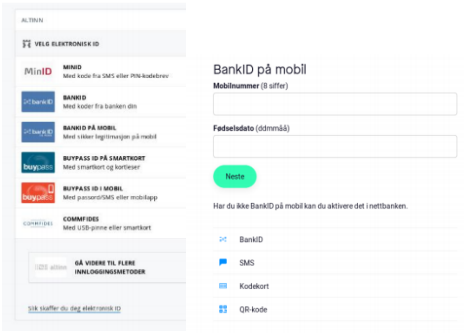
- Utfordring: Hvordan dele nøklene, og hvordan være sikre på at nøklene ikke er falske?
- Løsning: *Et sertifikat* som følger nøkkelen knytter sammen en offentlig nøkkel og en identitet.
- Neste utfordring: Hvordan stole på et sertifikat?
- Løsning: *Sertifikatet* utstedes av en betrodd virksomhet (Certificate Authority), som går god for nøklenes ektehet. Trusted third party (TTP).

- Benytter sertifikater
  - Offentlig nøkkelen bakes inn i et *elektronisk sertifikat* som knytter nøkkel og identitet sammen.
  - Et *sertifikat* utstedes av en *tiltrodd sertifikatutsteder* (Certificate Authority), som går god for den offentlige nøkkelens ekthet.
  - Den private nøkkelen kan oppbevares på en datamaskin, på SIM-kortet i mobilen, på et smartkort, i banken (Bank ID).
  - Den offentlige nøkkelen må være tilgjengelig for mottagere som skal kryptere og dekryptere meldinger. Eks. *public key servers*.
- I praksis noe mer komplekst enn dette, da man opererer med *hierarki av sertifikater*, *tillitskjeder*, mm. Utenfor IN1020-pensum ☺

PKI i bruk

- Sikker kommunikasjon i helsevesenet:
  - eResept, elektronisk sykemelding, etc
  - Knytter melding til individ (signatur med personlig sertifikat for lege) og sikre konfidensialitet (kryptering av forsendelse).
- Publikumsrettede PKIer i Norge:
  - MinID (offentlig), BankID, Buypass, Commfides (private)
  - Difi tilbyr ID-porten, en felles infrastruktur for innlogging til offentlige tjenester, basert på ovenfornevnte.
- Web PKI eller Browser PKI
  - Nettlesere kjenner den offentlige nøkkelen til alle betrodde sertifikatutstedere (CA), og bruker denne til å forsikre seg om at sertifikatet er gyldig.

Eksempel: BankID-samarbeidet



- Tilbyr «Elektronisk legitimasjon»
- Obligatorisk to-faktor-autentisering
- Baserer seg på PKI:
  - BankID: Banklagret nøkkel
  - BankID på mobil: SIM-lagret nøkkel
- Bruksområder:
  - PKI-Autentisering
  - Digital Signering
- Brukes av offentlige tjenester og banker

Sikkerhetstiltak – sikring av ressurser

Repitisjon: informasjon kan befinne seg i ulike tilstander, de ulike tilstandene krever ulike sikkerhetstiltak: Under lagring, under overføring, når den er i bruk.

Man må ha lag på lag med sikkerhet.

- **Forebyggende (preventive) tiltak:**
  - Forhindre og avskrekke angrepsforsøk.
- **Detekterende (avdekkende) tiltak:**
  - Varsle angrep som forsøkes eller som allerede er skjedd.
- **Korrigerende tiltak:**
  - Gjenopprette skade på dataressurser etter angrep.

Det er alltid nødvendig å benytte en kombinasjon av tiltak fra alle tre faser for å opprettholde dekkende beskyttelse.

- Hvis en inntrenger kommer seg gjennom ett *sikkerhetslag*, vil du ønske å kunne stoppe dem i det neste. Sørg for:
  - ✓ Fysisk sikring
  - ✓ Sikker lagring
  - ✓ Datamaskiner og IT-systemer uten sårbarheter
  - ✓ Nettverkssikkerhet
  - ✓ Sikker kommunikasjon
  - ✓ Bevisste brukere

Det er summen av alle sikkerhetstiltakene i de ulike lagene som gir god sikkerhet.

Terminologi

- Angrep (attack):
  - Noen forårsaker med hensikt en uønsket hendelse
- Angrepsvektor (attack vector):
  - Metoden en angriper benytter for å utføre et angrep
- Angrepsflate (attack surface):
  - Det stedet en trusselaktør kan komme seg inn i systemet
- Zero-day sårbarhet:
  - En sårbarhet som er ukjent for leverandør
- Utnyttelse (exploit):
  - Noe (en oppskrift, et program, en implementasjon) som kan utnytte en sårbarhet.
- Botnett:
  - Nettverk av «kaprede» maskiner som (fjern)styres av persone med uærlige hensikter

Hvordan opererer en trusselaktør?

Vanlige angrepsvinkler:

- Å tilegne seg urettmessig tilgang til systemer. Utnytte sårbarhet, benytte skadevare, få tilgang til, forfalske eller slette informasjon.
- Å avlytte eller forfalsker nettverkstrafikk for å få tilgang til eller manipulere informasjon.
- Å skape og utnytte tillitt (sosial manipulering) for å tilegne seg urettmessig tilgang eller gevinst.
- Å utilgjengeliggjøre systemer eller tjenester (DDoS eller DoS).

Skadelig programvare, *skadevare*

(Engelsk: Malicious software eller *malware)*

- Samlebetegnelse på programvare som med hensikt er inkludert eller plassert i et datasystem for å gjøre skade.
- Skadevaren kan like gjerne utnytte sårbarheter i kjernen som i applikasjoner eller andre programmer.
- Skadevaren kan være avhengig av å være en del av et *vertsprogram*, eller er et enkeltstående program som kan kjøres som en egen prosess av operativsystemet.
- Noe skadevare formerer seg og infiserer andre systemer, som regel avhengig av en trigger.

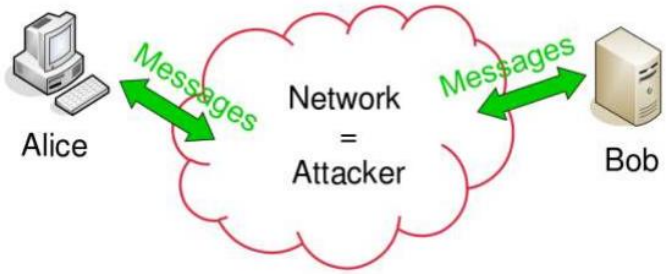
Ulike typer skadevare

Gruppene er til dels overlappende, og ikke alltid presist definert:

Virus	Kjennetegnes ved at det sprer seg hver gang en fil med virus blir åpnet av bruker eller systemet. Eks. løsepengevirus.
Ormer	Skadevare som på egenhånd er i stand til å spre seg og finne nye ofre.
Trojaner	Skadevare som utgir seg for å være et nyttig program.
Bakdør	En ubeskyttet og ukjent åpning inn i et datasystem, laget av en angriper.
Spionvare	Programvare for å spionere på en bruker i håp om å få tak i f.eks. informasjon, passord, kontoinformasjon.
Logisk bombe	Skadevare som kjøres på et forhåndsgitt tidspunkt eller ved en bestemt handling eller hendelse.
Tastelogger (Keylogger)	Enhet eller programvare som er koblet til tastatur eller maskin og som lagrer alle tastetrykk.
Rootkit	Skadevare som endrer funksjoner i OS eller applikasjoner, med mål å ta kontroll over en maskin eller et system.

Nettverkssikkerhet

Tradisjonell sikkerhetsmodell: stoler på endesystemene, nettverket anses upålitelig. Endesystemer sender og mottar meldinger til og fra nettverket, og nettverket kan levere, slette, endre eller forfalske meldingene.
Nettverket er som en åpen oppslagstavle. Det er derfor vi krypterer.



Sikkerhetstrusler.
Tradisjonelle trusler:

- Avlytting. Lytter til nettverkstrafikk. Passiv type angrep.
- Forfalsking.
- Man in the middle-angrep. Sitter en i midten og lytter og griper inn i kommunikasjon.

Nettverkssikkerhet kan sies å bestå av to hovedområder:

- Kommunikasjonssikkerhet:
  - Beskytte data i transporten mellom virksomheter (og endenoder)
  - Beskytte kommunikasjon med andre nettverk
- Skallforsvar:
  - Beskytte en virksomhets nettverk mot uautorisert aksess fra omverdenen.
  - Beskytte eget lokalt nettverk

Det er et behov for sikre nettverksprotokoller. To av de mest benyttede er:

- TLS: sikkerhet i transportlaget. Benyttes i https. Basert på PKI.
- IPSec: Sikkerhet i nettverkslaget. Brukes for å tilny VPN. Baserer seg på kryptering, autentisering og nøkkelhåndtering.

Sårbarheter finnes likevel, protokollene er ikke sikrere enn algoritmene de baserer seg på

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Eksempel: Heartbleed

- OpenSSL er en svært vanlig brukt *implementasjon* av *TLS-protokollen*
- Heartbleed var en sårbarhet introdusert i Open SSL i 2012, men ble ikke offentlig kjent før i 2014.
- Sårbarheten gjorde det mulig for hvem som helst (på internett) å lese minnet til systemer som var beskyttet av OpenSSL
  - Mulig for angripere å få tak i private nøkler
  - Mulig for uvedkommende å avlytte alle data som ble transportert mellom tjenester og brukere

Lærdom: Selv kritisk programvare basert på åpen kildekode kan inneholde sårbarheter

Skallforsvar

Brannmur – førstelinjeforsvar

- Avgjør hva som skal slippes inn og/eller ut fra et lokalt nettverk.
- Fungerer som en tilgangskontroll i nettverket.
- Implementeres i programvare eller maskinvare.

Innbruddsdeteksjon (IDS)

- Overvåking. Nesten alle bedrifter gjør dette i dag.
- Kan detektere misbruk, både forsøk og suksessfulle innbrudd. Skadevare. DDoS-angrep.

DoS og DDoS. (Destributed denial of service)
Kan true tjenesters tilgjengelighet. Angripere kontrollerer botnet og kjører koordinerte angrep (pakkeoversvømmelse) mot en tjeneste.

DNS-sikkerhet.
Navneoppslag, knytter maskinnavn til IP-adresse.
Sårbart:

DNS-forfalsking
Uvedkommende introduserer uriktige opplysninger i navnetjener.

DNS-modifisering
Man in the middle-angrep, forfalsker meldinger.

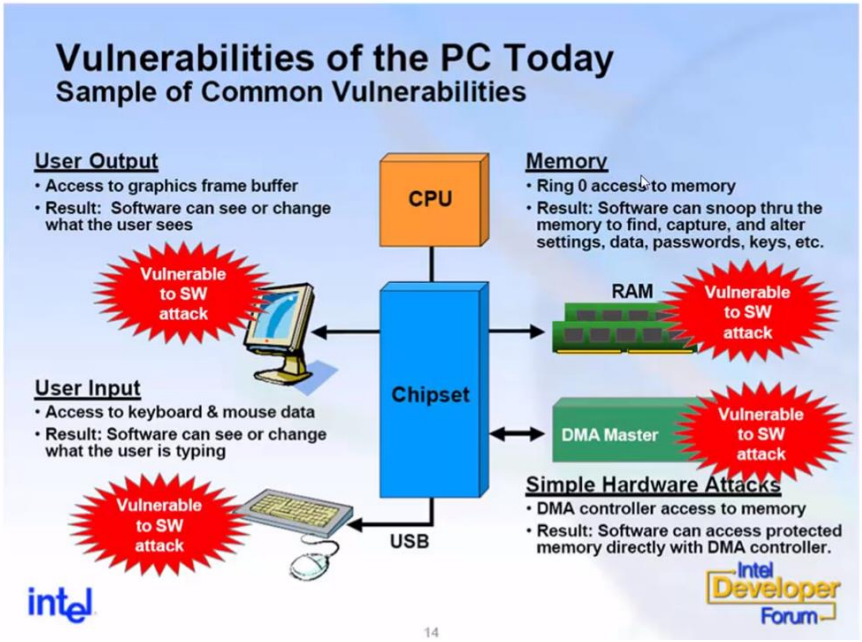
DNS-kapring.
Ende hvilke navnetjener offerets maskin benytter.

Man bør derfor aldri stole ubegrenset på et domenenavn. Det er vanskelig å oppdage feil.

Trådfast vs trådløs forbindelse

- Kommunikasjon i trådløse nett går via radiosignaler, og skaper derfor andre utfordringer.
  - Radiosignaler følger ikke fysiske grenser, hvilket betyr sårbarhet for avlytting også utenfor f.eks. en bygning.
  - Åpner for uautorisert nettverkstilgang (internett, intranett) uten fysisk kobling.
- Vanlige sårbarheter
  - Falske aksesspunkter. En trådløs enhet vil koble seg til AP-et med sterkest signal.
  - Signalforstyrrelser (tilsiktet eller utilsiktet) kan føre til utilgjengelighet (DoS og DDoS)
  - Aksesspunkter kan kompromitteres, spesielt åpne nett/aksesspunkter.

Trussel mot datamaskiner i dag.



Sikkerhet i operativsystemet.

Kjernen i OS styrer alt. Kommuniserer med periferenhetene i maskinen gjennom drivere. Brukerprosesser må kommunisere via kjernen for å kunne benytte periferenhetene. Minnet i en datamaskin benyttes bla til mellomlagring.

Kjernen og drivere er dataprogrammer som kan inneholde sårbarheter. Kjernen kjører i supervisor mode, og har tilgang til alt. En sårbarhet her er relativt kritisk. Viktig å holde OS (kjerne) og drivere oppdatert for å unngå at uvedkommende utnytter eventuelle sårbarheter.

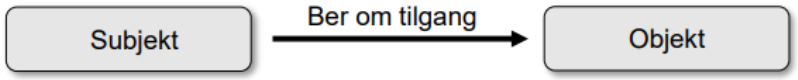
Autorisering. Å autorisere er å sjekke at et subjekt har tilgang til å utføre den ønskede handlingen på et objekt etter forhåndsdefinerte regler. Autorisering er ikke teknologi, men policy. En ansatt i lønnsavdelingen skal ha tilgang til lønnssystemet.

Tilgangskontroll.

Ethvert system må ha en tjeneste hvis oppgave er å kontrollere tilgang fra subjekt til objekt. Tilgang innvilges eller avvises. Følger policyen som er satt. Logger hendelser.

Tilgangskontroll kan være på flere nivåer:
Brukere og passord kan kontrolleres i applikasjonslaget.
Tilgang til ulike nettverksporter.
Tilgang basert på IP-adresser.

- Et subjekt vil utføre handling på objekt



  - Anne vil lese en fil
  - Per vil oppdatere kontobalansen på en bankkonto
  - En prosess vil åpne en nettverksforbindelse
- Tilgangskontroll = autentisering + autorisasjon

Tilgangskontroll i UNIX

- Brukere (brukernavn, UID) og grupper (gruppenavn, GID) er grunnleggende elementer, begge er permanente identiteter.
- Brukere har en brukerkonto.
- Brukere tilhører en eller flere grupper, en av gruppene er *primærgruppe*.
- Subjekter i UNIX:
  - En prosess
  - En prosess har en *ekte* UID og en effektiv UID. Tilsvarende for GID.
  - *Ekte* UID/GID: Arves fra foreldreprosessen, typisk UID/GID til innlogget bruker
  - *Effektiv* UID/GID: Arves også fra foreldreprosessen, eller fra programmet som kjøres.
- Objekter i UNIX:
  - Filer (programmer), mapper og *devicer* (maskinvareenheter, representert som filer).
- Se forelesning om operativsystemer for detaljer om tilgang til og beskyttelse av filer og mapper.

Lagring av data.

Data kategoriseres gjerne etter hvilket beskyttelsesbehov det har:
Åpen informasjon
Informasjon med begrenset beskyttelsesbehov
Informasjon med sterkt beskyttelsesbehov
Hvilke sikkerhetstiltak som iverksettes avhenger av beskyttelsesbehovet, for eksempel: kryptering, tilgangskontroll, perimeterforsvar, sikkerhetskopier.

Kryptering. Kryptering av data bidrar til å sikre konfidensialitet. Kan kryptere filer eller hele harddisker. Man bør ta godt vare på krypteringsnøkkelen, mister man denne er man dum dum.

Løsepengevirus. WannaCry ransomware attack. Utnyttet kjent sårbarhet i en fildelingsprotokoll. Sårbarheten var patchet 2 mnd. før, men brukere av Windows som ikke hadde oppdatert fikk viruset.

Sikkerhetskopiering.
Strategi for å ta vare på informasjon over tid.
Mål: kunne gjenopprette systemer og data ved uønskede hendelser som endrer eller sletter informasjon. Tar vare på ulike versjoner av informasjonen.
Sikkerhetskopien må også sikres, lagres trygt adskilt fra opprinnelige disker/data.

Sikker avhending av datautstyr. Destruering:
Å slette en fil markerer plassen på disk som ledig, men rådataene finnes fortsatt.
Å overskrive en fil er heller ingen garanti.
Filsystemer organiserer ofte data på uortodokse måter.
Overskrive all ledig diskplass med tilfeldig data?
Magnetisk behandling, destruering?
Fysisk destruering.

Applikasjonssikkerhet.

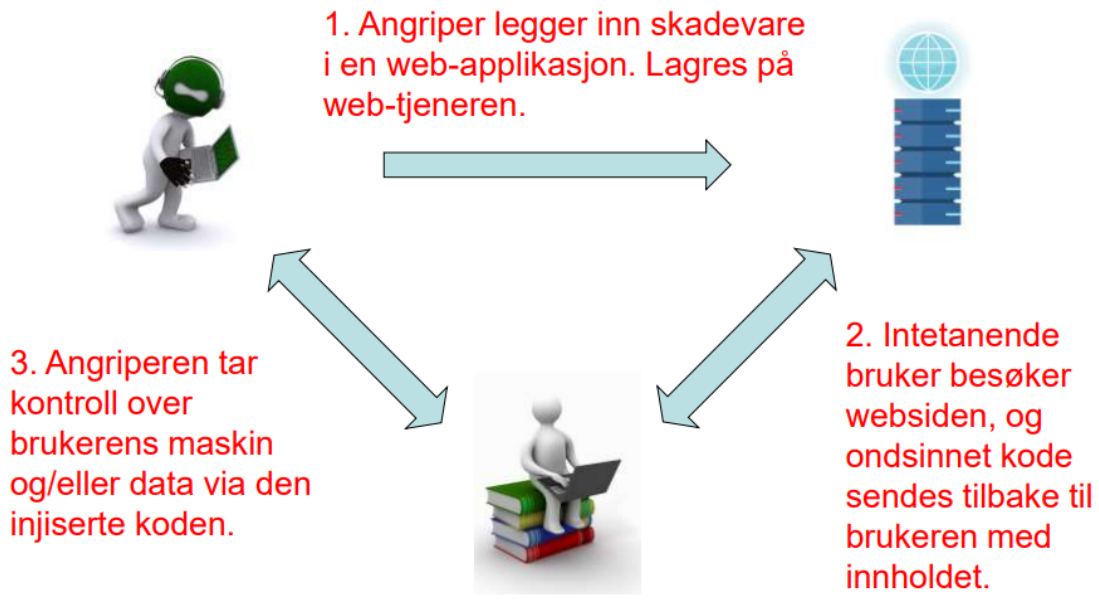
Applikasjoner må designes og utvikles med tanke på å kunne brukes trygt.
I designfasen: trusselmodellering:
- Hvilke trusler ser vi?
- Hvilke tiltak kan vi iverksette for å motvirke truslene?

I programmeringsfasen:
- Unngå å skape/tilrettelegge sårbarheter

Bufferoverskrivelse, vanlig sårbarhet.
- Et buffer er en sammenhengende seksjon i minnet i datamaskinen, avsatt til å inneholde data.
- En feil i et program, som gjør at programmet når det kjøres kan overskrive bufferets grenser i minnet, kalles bufferoverskridelse (buffer overflow)
- En bufferoverskridelse kan krasje et program, føre til korrupte data, eller i verste fall utnyttes av angripere.
- Kreative programmerer kan utnytte en slik sårbarhet:
  - Plasserer skadevare på steder i minnet hvor det ligger kjørbar kode.
  - Endre en funksjons returadresse til å isteden peke til adressen i minnet der skadevare er plassert.
- Hvordan sikre seg mot slike sårbarheter?
  - Sjekke all input fra brukere

SQL injisering, vanlig sårbarhet.
- SQL er et spørrespråk mot relasjonsdatabaser
- Relasjonsdatabaser brukes svært ofte i bakkant av web-apps for lagring av data/informasjon.
- SQL-injisering:
  - Å lure inn spørre-setninger sammen med informasjon som er gitt som input i et grensesnitt
  - Muliggjøres på grunn av «slapp» programmering.

Cross Site Scripting (XSS), vanlig sårbarhet.



Tiltak mot SQL injisering og XSS:
- Filtrer alltid all input fra brukere!

Årlig rapport. De vanligste sårbarhetene.

OWASP Top 10

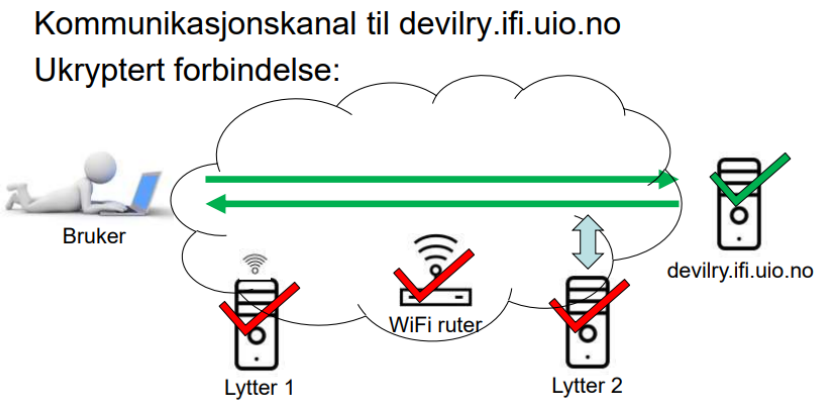
The Open Web Application Security Project

OWASP Top-10 2017:

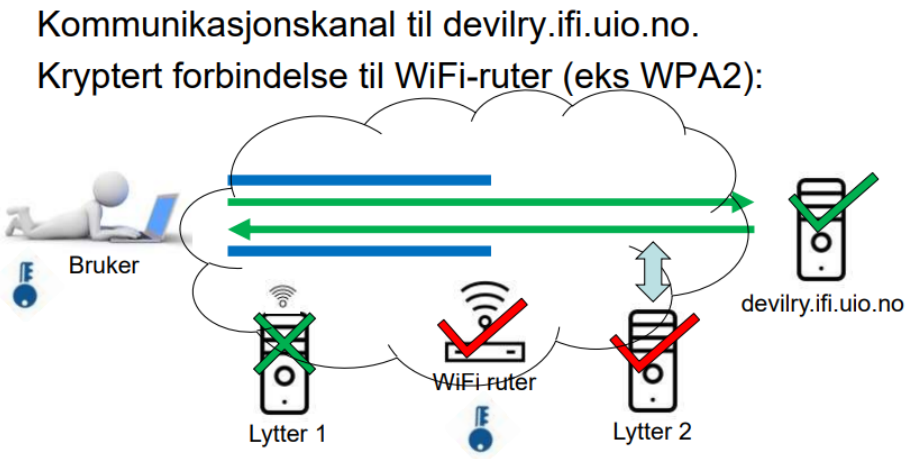
1. Injection
2. Broken Authentication and Session Management
3. Cross Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

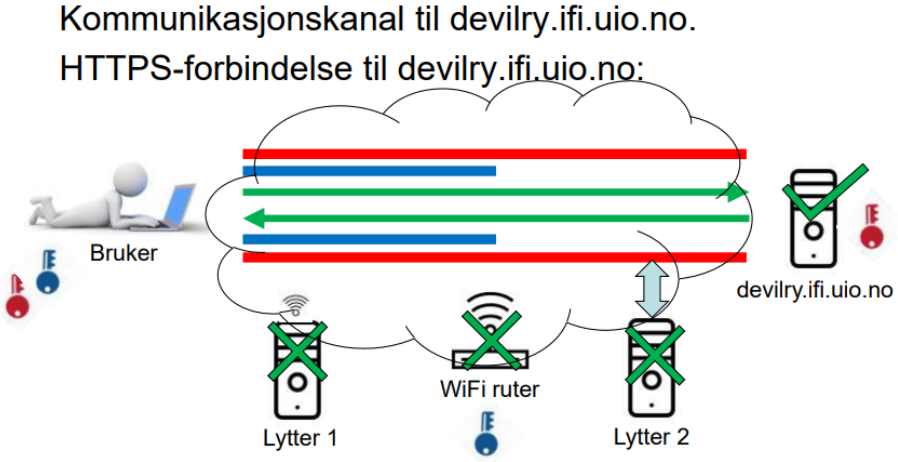


Hvis har en helt ukryptert kanal, så er det det samme som å «snakke høyt på bussen».



Kryptert forbindelse mellom bruker og ruter. Lytter 1 vil da ikke fange opp relevant data, men det vil lytter 2 kunne gjøre. Et offentlig nettverk kan være som dette. Man vet ikke om eierne av ruteren krypterer datakommunikasjonen som går ut fra nettverket.





En personopplysning

- Personopplysninger er alle opplysninger og vurderinger som kan knyttes til deg som enkeltperson
  - Navn, adresse/lokasjon, epost-adresse, fødselsnummer, bilde, adferdsmønster, fingeravtrykk, identifikasjonsnummer, telefonnummer, registreringsnummer på bil, …
- Særlige kategorier personopplysninger
  - Rasemessig eller etnisk opprinnelse, politisk oppfatning, religion, fagforeningsmedlemskap, genetikk, biometri med formål entydig identifikasjon, helseopplysninger, seksuell legning, straffedommer og lovovertredelser, …

EUs Personvernforordning (20.07.18)

- Virkeområde: EU samt virksomheter som tilbyr varer og tjenester til EU-borgere
- Regulerer
  - *Retten* til personvern
  - *Hjemmel* for behandling
  - *Samtykke* fra den registrerte
  - *Åpenhet* om bruk av opplysninger:
    - Hvem, hvorfor, hvordan, hvor og hvor lenge
  - Krav til *innebygd personvern* og personvern som standard
- I tillegg: Grunnloven § 102:

  «Enhver har rett til respekt for sitt privatliv og familieliv, sitt hjem og sin kommunikasjon. Husransakelse må ikke finne sted, unntatt i kriminelle tilfeller.»

Innebygd personvern.
Sørge for at informasjonssystemene vi bruker oppfyller personvernprinsippene.
Ta hensyn til personvern i alle utviklingsfaser av et system eller en løsning.
Det minst personverninngripende alternativet skal være standarden i alle systemer og løsninger.

- Mengden personopplysninger
- Omfang av behandling
- Lagringstid
- Tilgjengelighet

Innebygd sikkerhet.
Ivareta informasjonssikkerhet fra start til slutt
Er et overordnet prinsipp i arbeidet med informasjonssikkerhet
Innebygd informasjonssikkerhet oppnås når informasjonssikkerhet er:

- Innarbeidet i virksomhetsstyringen og understøtter virksomhetens mål
- Innarbeidet i virksomhetens prosesser og prosjekter fra starten av
- Tatt hensyn til i hele livssyklusen til IKT-løsninger
- Et tema alle ansatte i virksomheten kjenner til og vet hva innebærer for sine arbeidsoppgaver.

Trusselmodellering.
En prosess som gjennomgår sikkerheten til et system, identifiserer problemområder, og evaluerer risikoen assosiert med hvert område. Identifisere potensielle trusler. Bør gjøre dette tidlig i en utviklingsprosess.
Ønsker:

- Tidlig håndtering av sikkerhetsproblemer
- Bedre sikkerhetsvurderinger
- Mer effektiv sikkerhetstesting

Ulike innfallsvinkler:

- Fokus på trusselaktør:
  - Identifisere aktørens mål, hvordan de vil oppnå dem, hva er angrepsvektor og angrepsflate.
- Fokus på system:
  - Plukke fra hverandre systemet. Identifisere ulike deler og hvordan og hvor de kan angripes.
- Fokus på verdi:
  - Identifisere verdiene i systemet, avdekke hvordan brudd på informasjonssikkerhet kan intreffe.
- Felles for dem alle, lære av feil: hva har gått galt tidligere?

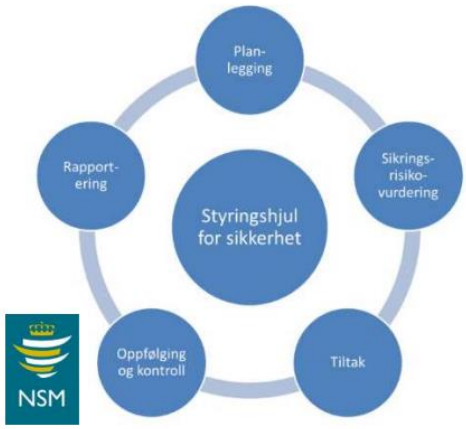
Risikoanalyse.
Risiko involverer konsekvens, ofte i kroner og øre.
«Risiko er et produkt av sannsynlighet for hendelse og kostnad/tap».
Skal man fokusere på usannsynlige hendelser som medfører store tap, eller sannsynlige hendelser med mindre tap?
    Ikke noe klart svar på dette.

Administrative sikkerhetstiltak.
    Internkontroll / LSIS – ledelsessystem for informasjonssikkerhet
    Policyer / standarder
    Prosedyrer og praksis
    Bevissthetstrening
    Sikkerhetsøvelser
    Hendelseshåndtering

Styringssystem for sikkerhet

*«**Sikkerhetsstyring** er systematiske aktiviteter for å oppnå og opprettholde et sikkerhetsnivå i overensstemmelse med de mål og krav en organisasjon har satt seg»*

- Informasjonssikkerhet må inkluderes i virksomhetsstyringen
- Eksempel: NSMs styringshjul for informasjonssikkerhet



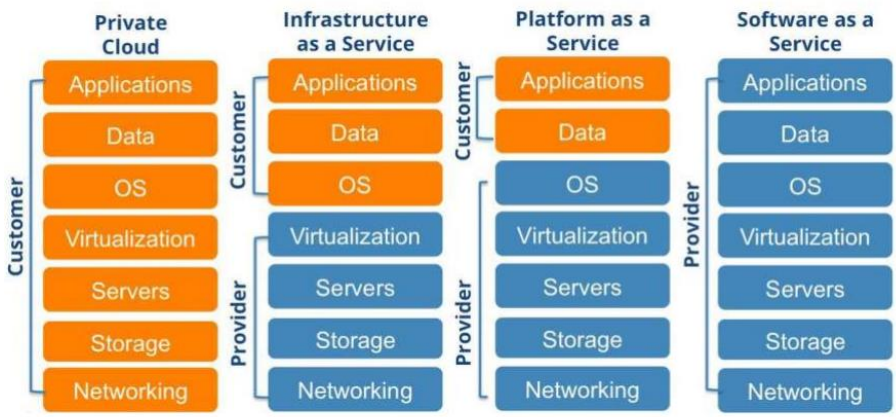
God informasjonssikkerhet koster:
    Kompetanse og bevissthet
    Styring og kontroll
    Drift og vedlikehold
    Tjenesteutsetting
    Sikkerhetstiltak
Styret i en bedrift må derfor være bevisst sitt ansvar om å implementere god informasjonssikkerhet.

På sikt er sikkerhet lønnsomt. Å beskytte ressurser = skape verdi. Tillit, rykte, merkevare.
Konkurransefortrinn. Forebygge og redusere tap.
Skape robusthet og sikre kontinuitet. Øker selskapets verdi.

Tjenesteutsetting og skytjenester.

Tjenesteutsetting innebærer å sette ut en eller flere funksjoner/tjeneste til andre leverandører.
    På eget datautstyr, hos leverandør, kombinasjon
    Tradisjonell tjenesteutsetting: produkter leies til fast pris
Skytjenester som modell innebærer å få tilgang til et sett med dataressurser
    Lett tilgjengelige overalt
    Data blir lagret i store datasentre
    Blir levert og priset etter behov, rask anskaffelse og tilgjengelighet
    Betaler kun for ressursene

Ulike typer skytjenester



Tjenesteutsetting

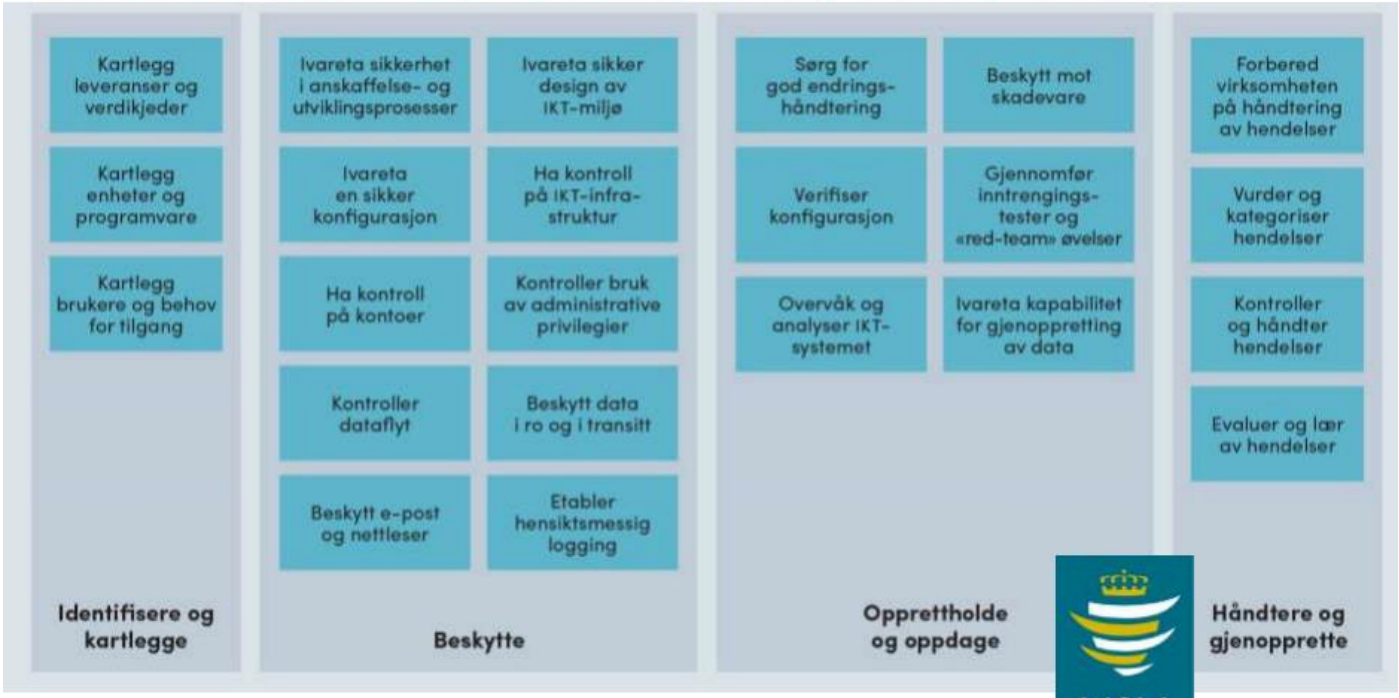


- Fordeler:
  - Reduserte driftskostnader: kompetanse, infrastruktur
  - Besparelser i form av betaling for faktisk bruk
  - Lokasjonsuavhengig tilgang
  - Sannsynligvis god informasjonssikring
- Ulemper:
  - *Dersom en virksomhet ønsker å benytte skytjenester til behandling av personopplysninger, er virksomheten juridisk ansvarlig for at personopplysningene prosesseres og lagres i samsvar med personvernregelverket.*

    ➡ Databehandleravtale

Grunnprinsipper for IKT-sikkerhet

https://www.nsm.stat.no/publikasjoner/rad-og-anbefalinger/grunnprinsipper-for-ikt-sikkerhet/



Oppgaver og løsningsforslag:

Oppgave 1
Beskriv kort forskjellene mellom linjesvitsjing (circuit switched) og beskjedsvitsjing (message switched) nett. Hva ligner mest på Internett i dag?

Svar:
I Linjesvitsjing settes det opp en dedikert linje som brukes under hele kommunikasjonen, denne er statisk, og kan ikke endre seg. Den er tilkoblingsorientert, hvilket vil si at kommunikasjonen må først settes opp igjennom hele nettet før kommunikasjonen kan begynne. Når kommunikasjonen er ferdig kobles linjen ned.
Beskjedsvitsjing er tilkoblingsløs, beskjeden trenger ikke nødvendigvis følge samme vei hver gang, og det er ingen dedikert linje mellom sender og mottaker.
Internett i dag bruker pakkesvitsjing, som ligner veldig mye på beskjedsvitsjing.

Oppgave 2
Hvorfor er nettverket organisert i lag?

Svar:
Hvert lag leverer et unikt sett med tjenester til lagene over, noe som muliggjør utskifting av mekanismer/teknologier på alle lag. TCP/IP-modellen er litt spesiell, da IP er en fellesnevner for (så å si) all transport, med et utall av teknologier og protokoller i lagene over og under. Lagdelingen gjør det f.eks. mulig for nettverkskomponenter (som swithcher og routere) å implementere bare et delsett av lag, for å støtte bare det som trengs for å sende en pakke videre.

Oppgave 3
A)
Hvilke lag finner vi vanligvis i Internett?

Svar:
Fysiske lag
Linklaget
Nettverkslaget (IP)
Transportlaget (TCP/UDP)
Applikasjonslaget

B)
Hva er oppgaven til hvert av de forskjellige lagene?

Svar:
Fysiske lag - sikre at signaler som kan tolkes som bits (0 eller 1) kan leveres til neste ledd i kommunikasjonen. F.eks trådløst, eller over en ledning.
Linklaget - sikre at en gitt mengde bits kan deles opp i håndterbare enheter kalt “frames”, og at disse kan sendes over linken til neste mottaker uten å gå tapt eller miste informasjon.
Nettverkslaget (IP) - Levere en datapakke til en annen vert på Internett, ofte på tvers av flere lokale nettverk (LAN).
Transportlaget (TCP/UDP) - ekstra tjenester i tillegg til adressering. TCP gir f.eks. pålitelighet, forbindelsesorientering, metningskontroll, bytestrøm, levering i samme rekkefølge som data ble sendt, feilsjekking og flytkontroll. UDP leverer et minimum av tjenester, og overlater de mer avanserte tjenestene til å bli implementert av de som skriver applikasjonene. Transportlaget gjør det også mulig å skille forskjellige applikasjoner fra hverandre som mottakere innenfor én IP-adresse (vertsmaskin) ved hjelp av konseptet “porter”.
Applikasjonslaget - Alle programmer som bruker lagene under. F.eks. epost (SMTP) og web (HTTP).

Oppgave 4
A)
Hvorfor trenger protokoller en header?

Svar:
For å håndtere og overlevere metadata som trengs for å levere tjenestene protokollen tilbyr.

B)
Hvordan blir header lagt til av de forskjellige lagene i en stack, og hva skjer på mottakersiden?

Svar:
Header blir lagt til lag for lag, etter hvert som pakken beveger seg gjennom lagene i sendeprosessen.
Eks fra forelesning: data fra applikasjon får en TCP-header, så en IP-header, så en Ethernet-header.
En router (som skal sende pakker ut av et LAN), vil ta av og bytte ut headere for linklaget. Når pakken kommer frem til målet, blir headerene tatt av lag for lag i motsatt rekkefølge av da pakken ble sendt etter hvert som pakken beveger seg oppover i protokollstakken mot applikasjonslaget.

Oppgave 1:
a) Hva er tjenere, klienter, switcher og routere, og hvilken rolle spiller hver av disse komponentene i nettverkskommunikasjonen?

Svarforslag:
Tjenere: enheter/verter som leverer tjenester over nettverket.
Klienter: enheter/verter som benytter seg av tjenester levert over nettverket.
Switcher: Kobler enheter sammen i et subnett.
Routere: Kobler forskjellige subnett sammen. Nødvendig for å sende pakker ut av et LAN.

b) TCP og UDP er de mest vanlige protokollene i transportlaget.
Hvilke tjenester tilbyr UDP?

Svar: UDP tilbyr multiplexing over en IP-adresse (porter) og kontroll av nyttelasten (checksum)

Hvilke tjenester tilbyr TCP?

Svar: samme som UDP, men i tillegg pålitelighet (garantert levering), levering i rekkefølge, flytkontroll (for ikke å sende fortere enn. mottakeren kan ta i mot), metningskontroll (for å dele nettverksressursene med andre datastrømmer.

Hvorfor ønsker man noen ganger å bruke UDP til tross for at TCP tilbyr flere tjenester?

Svar: Det krever mindre ekstra data for overføring (mindre overhead). Det kan hende man ønsker å bygge sin egen versjon av flytkontroll, mentningskontroll e.l. i applikasjonen. Det at sendetidspunktet ikke styres av metningskontroll, gjør at applikasjonen som sender pakken kan bestemme sendetidspunktet selv (nyttig i f.eks telekonferanse og videostreaming).

c) Hva er en MAC-adresse, og på hvilket lag brukes den?

En adresse som identifiserer ett nettverksinterface (NIC) som bruker overføringsteknologier som f.eks. Ethernet eller WiFi. Logisk sett opererer en MAC-adresse på linklaget, selv om den ofte er bygget inn i maskinvaren som gjør overføringen. En MAC-adresse gjør at man kan addressere andre maskiner på samme lokale nettverk (LAN).

Hva er ARP, og hvorfor trenger vi denne protokollen?

ARP gjør at man kan finne ut hvilken IP-adresse (Nettverkslaget) som korresonderer til MAC-adressene i et LAN. Dette gjør at maskinene kan adressere hverandre innefor et LAN ved å bruke IP-adresser.

Oppgave 2: IP-adresser
IP'en til en maskin er gitt i CIDR-notasjon ved: 192.168.0.165/28

Hva er IP-adressen med hver oktett notert binært?

Svar: 11000000.10101000.00000000.10100101

Hva blir nettverksmasken med hver oktett notert binært?

Svar: 11111111.11111111.11111111.11110000

For å finne subnettadressen til maskinen må du gjøre en bitvis AND operasjon mellom IP-adressen og nettverksmasken.

Hva blir subnettet til maskinen over i CIDR-notasjon?

Svar: 192.168.0.160/28

Hvor mange adresser er tilgjengelig til verter i dette subnettet?

Svar: 14

For å finne kringkastingsadressen til et subnett, må du gjøre en bitvis OR-operasjon mellom maskinens IP-adresse og bit komplement (bitvis invers) av nettverksmasken.

Hva er kringkastingsadressen til subnettet i oppgave 2 a) i CIDR-notasjon?

Svar: 192.168.0.175

Hva er NAT, og hvordan fungerer det?

Svar: NAT står for «Network Address Translation», og er en metode for å skrive om en IP-adresse til en annen når den passerer en enhet i nettverket. Den benyttes for det meste for å gjøre at man kan gjenbruke IP-adresser i forskjellige lokale nettverk ved å skrive dem om, slik at alle forbindelser med opprinnelse innenfor et NAT-et nettverk ser ut som det kommer fra den samme offentlige IP-adressen sett fra utsiden. Dette gjøres ved at hver forbindelse kobles opp mot et portnummer tilhørende den eksterne IP-adressen. NAT-enheten holder oversikt over hvilke enheter/forbindelse på innsiden av LANet som korresponderer med de forskjellige portnumrene som benyttes utad.

Hva er hovedgrunnen til at man forøker å gå over til IPv6?

Svar: Man er i ferd med å slippe opp for ledige IP-adresser i Internett med IPv4, selv med den utstrakte bruken av NAT. Dette vil gjøre det mer å mer vanskelig å få muligheten til å benytte offentlig synlige IP-adresser. IPv6 har veldig mange flere adresser (128 bits adresse mot 32 bit i IPv4).

Hvorfor er det vanskelig å bytte IP-protokollen?

Svar: Alle enhetene som trenger å behandle IP-protokollen på veien mellom avsender og mottaker må ha støtte for IPv6 om ikke leveringen skal mislykkes. Det vil si at for å sette det i global drift er det svært mange enheter som må støtte dette om det ikke skal oppstå mange soner som ikke får levert data. Dette tar lang tid og er krevende.

Oppgave 3: DHCP og DNS

Hva er DHCP, og hvorfor bruker man denne protokollen?

Svar: DHCP står for «Dynamic Host Configuration Protocol» og benyttes for å automatisk tildele IP-adresser til enheter som kobler seg på et lokalt nettverk (LAN). Den brukes fordi det er en omstendelig prosess å skulle manuelt tildele (og formidle) IP-adresser til alle som vil koble seg opp mot nettverket. Automatisering av denne prosessen gjør at de færreste trenger å tenke over dette når de kobler seg opp.

Hvorfor er det vanlig å «frigi» en IP-adresse utdelt med DHCP etter en viss tid?

Svar: Nettverk med mange enheter som kobler seg av og på over tid vil da kunne gjenbruke IP-adressene til enheter som ikke har vært aktive på en stund. Dette gjør at man ikke trenger et veldig stort subnett med mange IP-adresser som holdes opptatt av enheter som ikke svarer eller som har forlatt nettverket.

Hva er et DNS-navn, og hvorfor benytter man dette i tillegg til IP-adresser?

Svar: Et DNS-navn er 2 eller flere ord med punktum i mellom som man bruker for å koble opp mot maskiner på internett i stedet for å huske på IP-adressen til maskinen. Vi benytter dette da IP-adresser ikke er særlig menneskevennlig (vanskelig å huske). Det er også flere tjenester som bedre lar seg løse ved å koble navn til IP-adressen, som f.eks aliasing (fler DNS-navn til én IP) og round-robin DNS (samme DNS-navn går på rundgang mellom flere IP-adresser).

Hvilken tjeneste utfører en DNS rottjener?

Svar: De holder oversikten over hvilke maskiner som har ansvaret for hvert TLD (Top Level Domain). TLDer er det høyeste nivået i DNS-hierarkiet, ordet etter det siste punktumet (f.eks .org, .net, .com, .no).

Forklar hva DNS aliasing og round-robin DNS er, og hvorfor man benytter det?

Svar: DNS aliasing er når man kobler flere forskjellige DNS-navn til samme IP-adresse. Dette benyttes når man vil at flere forskjellige tjenester skal leveres av den samme fysiske enheten, som f.eks én webserver som leverer hjemmesider til mange bedrifter med forskjellige domenenavn.
Round-robin DNS er når det samme DNS-navnet på rundgang peker på forskjellige IP-adresser på en liste. Dette kan benyttes for å spre trafikken på populære tjenester over mange fysiske maskiner, så ingen av maskinene knekker sammen av belastning pga for mye trafikk.

Oppgave 2:

Hva er forskjellene mellom klient/tjener og peer-to-peer topologiene? Nevn eksempler på begge modeller.

Svar:
Klient-tjener: Typisk pull-tjeneste. Tjeneren har en spesiell rolle der den leverer en tjeneste til mange potensielle brukere. Eks. HTTP eller IMAP mail.
P2P: alle noder i nettverket er likeverdige og samarbeider om å levere en tjeneste. Eksempler: Bittorrent, The Onion Router (TOR), Bitcoin

Oppgave 3:

Hva er et "Content Delivery Network (CDN)"?

Svar: Innhold som trenger å leveres til svært mange brukere over hele verden kopieres til mange maskiner som ligger fysisk nær sluttbrukerne. Dette gjør at kildemaskinen ikke blir overbelastet og sparer kapasitet på nettverket.

Oppgave 4:

Nevn 3 grunner til at man velger å bruke et CDN for distribusjon av data:

Svar: 1) Å ha innholdet fysisk nærmere brukeren reduserer round trip time (RTT) og gjør det derfor raskere å aksessere, 2) Ved å cache innholdet nær brukeren sparer man trafikk over backbone-nettet, 3) Ved å distribuere innholdet, avlaster man tjeneren som leverer dette innholdet, noe som gir en mer skalerbar tjeneste.

Oppgave 5:

Hva er Dynamisk, Adaptiv Streaming over HTTP (DASH)?

Svar: En måte å strømme video ved hjelp av HTTP-tjenere. Videoen “klippes” opp i små uavhengige segmenter som kan kodes i flere kvalitetsnivåer. Dette gjør at en klient kan spørre etter høy kvalitet i perioder når den opplever gode nettverksforhold og redusere kvaliteten når forholdene er dårligere.

Oppgave 6:

Forklar kort kommunikasjonen mellom en nettleser og en HTTP-tjener når du spør etter en hjemmeside over nettverket.

Svar: En TCP-forbindelse opprettes med et 3-veis håndtrykk, om SSL skal brukes, forhandles dette frem ved å først bruke asymmetrisk kryptering til å utveksle en felles nøkkel for videre symmetrisk kryptering av forbindelsen. Klienten sender så en "GET"-forespørsel på det aktuelle dokumentet til HTTP-tjeneren. Tjeneren leverer et "200 OK" svar tilbake sammen med det aktuelle dokumentet. Om hjemmesiden inneholder dokumenter fra flere kilder opprettes tilsvarende forbindelser for å hente resten av det aktuelle innholdet.

Oppgave 7:

Hvilken transportprotokoll bruker HTTP, og hvilken port er standard for tilkobling for denne tjenesten?

HTTP bruker TCP og har port 80 som standard.

Bonusoppgave: Hvilken port er standard for kryptert (HTTPS) oppkobling?

HTTPS har port 443 som standard.

Oppgave 8:

Hva er forskjellen på persistente og ikke-persistente TCP-forbindelser?

Svar: En ikke-persistent forbindelse blir lukket så snart et svar er levert. Påfølgende forespørsler må åpne en ny forbindelse. En persistent forbindelse vil gjenbruke den samme forbindelsen til flere forespørsler.

Oppgave 9:

Man kan bruke programmet telnet til å opprette en TCP-forbindelse til en annen maskin og sende tekstlige kommandoer.
Kjør kommandoen "telnet www.ifi.uio.no 80"
Skriv "GET / HTTP/1.1"
Skriv "Host: www.ifi.uio.no"
Trykk "Enter" to ganger.
Hva får du tilbake?

Svar: Du skal få tilbake et HTML-dokument som oppgir at du bør bruke https til å aksessere nettsiden.

Oppgave 10:
a) Forklar kort hva som menes med "Surface web", "Deep web" og "Dark web".

Surface web: Alt som kan indekseres av søkemotorer på nettet, og som kan åpnes og leses uten bruk av passord eller krav om å være på et spesifikt lukket nettverk for å få tilgang.
Deep web: Alt som er tilgjengelig via en lukket tjeneste som krever innlogging, eller som på annen måte er beskyttet mot lesing av uautoriserte.
Dark web: Tjenester som, i tillegg til autentisering, krever anonymisering f.eks gjennom The Onion Router (TOR) og har strenge krav til diskresjon. Ofte kriminelle nettverk eller forum for grupper som risikerer å bli forfulgt om de blir avslørt.

b) om lag hvor stor andel av informasjonen som finnes på Internett er offentlig tilgjengelig via "Surface web"?

Svar: Estimert til under 10% (ca 4% i henhold til referansen på foilene).

Oppgave 11:

Hva er SMTP og MIME, og hvilken Internett-applikasjon benytter disse?

Simple Mail Transfer Protocol: transporterer e-post fra avsender til mottaker.

Multipurpose Internet Mail Extension:Protokoll på topp av SMTP som tillater overføring av multimedia (vedlegg) gjennom e-post. Brukes av e-postprogrammer som Outlook, Mac Mail og Thunderbird

4. Sikkerhetsmål.
Sikkerhetsmål er altså egenskaper ved informasjonssikkerhet, egenskaper vi ønsker å oppnå eller tilby, og som på sett og vis beskriver informasjonssikkerhet.

Definer og diskuter de syv egenskapene ved informasjonssikkerhet nevnt på forelesning 9/11:

a) Konfidensialitet
Konfidensialitet betyr at informasjon ikke tilgjengeliggjøres for annet enn de entiteter, systemer og/eller prosesser som er autorisert for tilgang til informasjonen. Det kan gjelde forretningshemmeligheter, personopplysninger, eller å f.eks. sikre anonymitet der dette er nødvendig. Konfidensialitet omfatter alle typer informasjon/data, alt fra forretningshemmeligheter, personopplysninger til konfigurering av systemer og da data som er med på å styre datamaskiner.

Som sikkerhetstiltak benyttes kryptering (lagring og kommunikasjon), tilgangskontroll (god styring på hvem som har tilgang til hva) og det som kalles skallforsvar (hindre uvedkommende tilgang innenfor et "skall", jmfr et godt gjerde rundt en bygning. Helseopplysninger er eksempler på konfidensielle opplysninger som skal beskyttes særlig.

b) Integritet
Integritet handler om å kunne stole på at informasjon eller systemer ikke er endret eller slettet av entiteter, systemer eller prosesser som ikke er autorisert for handlingen. Brudd på integritet kan skyldes både angrep utenfra og feil i systemer som fører til korrupte data. I likhet med konfidensialitet omfatter integritet også alle typer informasjon/data.

Som tiltak er autentisering og logging av andringer viktig, dessuten kryptering, gjerne i form av hash-algoritmer ved deling av data, samt digitalt signert programvare. Når du f.eks. skal laste ned programvare fra nett får du ofte programvaren sammen med en sjekksum, slik at du selv kan sjekke om programvaren stemmer med sjekksummen etter nedlasting (sikrer både mot korrupt programvare og nedlastingsfeil).

c) Tilgjengelighet
Tilgjengelighet dreier seg om at informasjon til enhver tid skal være tilgjengelig for den som skal ha tilgang til informasjonen. Mobilnett, nødnett, pasientsystemer, banksystemer, betalingssystemer er viktige systemer vi i dag er forholdsvis avhengige av i hverdagen, og som dermed bør ha sterkt fokus på tilgjengelighet. Samtidig vil det for en nettbutikk føre til svikt i omsetningen hvis deres nettbutikk gjøres utilgjengelig for en periode, selv om det kanskje ikke berører oss forbrukere på samme måte (vi velger en annen butikk...). DDOS-angrep eller andre angrep som tar ned et system/en tjeneste er en trussel, men det samme er f.eks. systemfeil, naturkatastrofer eller strømbrudd.

Som sikkerhetstiltak benyttes gjerne backup (av både informasjon og programvare, slik at gjenoppretting blir enkelt, samt å ha redundante systemer tilgjengelig.

d) Autentisering
Autentisering er å sikre at alle parter er de som de utgir seg for å være. F.eks. at nettsiden du besøker er den nettsiden du mente å besøke, og ikke en identisk kopi.

Hovedtrusselen er at noe eller noen klarer å utgi seg for å være noen/noe de ikke er. Passord er f.eks. enkelt å stjele/gjette/knekke, og falske nettsider ganske enkelt å lage. Som sikkerhetstiltak benyttes først og fremst ulike autentiseringsfaktorerog kryptografiske protokoller. Dette kommer vi nærmere inn på i forelesning om autentisering.

e) Uavviselighet (tidligere ikke-fornektelse).

Det skal ikke være mulig å påstå at man ikke har gjort noe man har gjort, og at man har gjort noe man ikke har gjort. Dette gjelder f.eks om man har undertegnet en kontrakt (f.eks for boliglån), eller glemmer å overføre penger i nettbanken.

Uavviselighet er altså å bekrefte at en handling eller et informasjonselement er uendret (integritet) og at det kan knyttes til en bestemt identitet. Uavviselighet er i mange sammenhenger også omtalt om ikke-benekting. En løsning for uavviselighet er en løsning som gjør det mulig for også en tredjepart å innhente tilstrekkelig dokumentasjon for at en annen part ikke kan nekte for å ha gjennomført en handlig eller ha valgt å bekrefte/vedgå seg et informasjonselement.

En løsning for uavviselighet er bygget opp på samme måte som løsninger for autentisering. Forskjellen er at brukeren skal være i stand til ikke bare å bekrefte hvem man er over nettet, men også å legge igjen dokumentasjon som entydig knytter personen til en handling eller et uendret informasjonselement.

f) Sporbarhet

Sporbarhet, også omtalt som etterprøvbarhet, er egenskapen som sikrer at alle handlinger kan spores tilbake til en entydig entitet. Sporbarhet garanterer at alle operasjoner som utføres av individer (brukere, personer, systemer eller prosesser som kjøres kan identifiseres, og at sporene bevares (logges) for senere bruk.

Ved et datainnbrudd øker dette muligheten for å spore en kriminell handling tilbake til den eller de som utfører handlingen. Men fortsatt er det dessverre slik at det på tvers av landegrenser, uten felles lovverk, kan være svært vanskelig å stille gjerningsmenn til ansvar for kriminelle handlinger.

Å sørge for sporbarhet i et system vil i mange tilfeller også være med på å rette opp feil som skjer. Et eksempel er, som nevnt på forelesning, hvis du plutselig oppdager at karakteren din plutselig er endret fra en ståkarakter til stryk, uten grunn. I et system som sikrer sporbarhet vil denne endringen kunne spores, og handlingen knyttes til f.eks. en identifiserbar bruker. Personen vil kunne stilles til ansvar for handlingen, og man kan finne ut om det er en feil, sabotasje, el.

For å kunne sikre sporbarhet brukes de preventive tiltakene autentisering og logging, mens etterforskning er et tiltak som benyttes i etterkant, som et korrigerende tiltak når en hendelse er avdekket og man må nøste i hva som har skjedd og til slutt "rydde opp".

g) Personvern (privacy)

Ved innføring av EUs Personvernforordning har man fått på plass et felles lovverk for behanding av personopplysninger (opplysninger som kan knyttes til "ekte"/fysiske personer) innen EU/EØS, som også vil gjelde alle som leverer tjenester til borgere i EU/EØS.

Personvern forordningen stiller krav til hjemmel (lovligheten) av behandling (og oppbevaring) av denne type informasjon, at det gis samtykke, osv.

5. Personopplysninger
Ta utgangspunkt i datatilsynets artikkel "Hva er en personopplysning?" (https://www.datatilsynet.no/rettigheter-og-plikter/personopplysninger/). Diskuter personopplysninger og særlige kategorier personopplysninger. Hvilke opplysninger faller under hvilke kategori, og kan du tenke deg hvorfor? Se gjerne på GDPR artikkel 4, 9 og 10.

SVAR: Ingen fasit her, diskuter med utgangpunkt i artikkelen. Det viktige er å forstå hva som er personopplysninger. Tenk også spesielt over hvorfor opplysningene som er kategorisert som særlige kategorier data er nettopp det.

6. Diskusjonoppgave:

Jim jobber i justisdepartementet. Han er veldig fornøyd med å få jobbe med deres nye satsningsområde, automatisert saksbehandling, de neste to årene. Han har fått beskjed om å bytte kontorplass fordi alle som jobber i prosjektet skal sitte samlet i et åpent landskap i øverste etasje. Samtidig som de flytter inn i landskapet får de utdelt nye PC-er. På disse er det et klistremerke med budskapet «Lås PC-en når du går fra den!». Jim husker at informasjonssikkerhetsansvarlig snakket om dette på nyansattkurset han deltok på for lenge siden, men har egentlig aldri forstått hvorfor han skal låse PC-en sin. Alle i landskapet jobber på det samme prosjektet. Hvorfor skal de egentlig låse PC-en sin?

SVAR:
Alle sider av informasjonssikkerhet må vurderes, men spesielt konfidensialitet, integritet, autentisitet og sporbarhet (etterprøvbarhet). Skal alle alltid ha tilgang til all informasjon selv om man jobber på samme prosjekt? Hvis alle kan bruke en ulåst pc vet man ikke hvem som har endret/lagt til/slettet informasjon. Hvilke konsekvenser vil dette få, og for hvem? (Arbeidsplassen, deg selv, andre brukere) Kan du få problemer

dersom en kollega bruker din pc til å utføre handlinger som ikke er tillatt? F.eks. spre informasjon som ikke skal spres, laste ned filer ulovlig eller utført andre handlinger i strid med intern sikkerhetsinstruks.