UNIVERSITETET
I OSLO

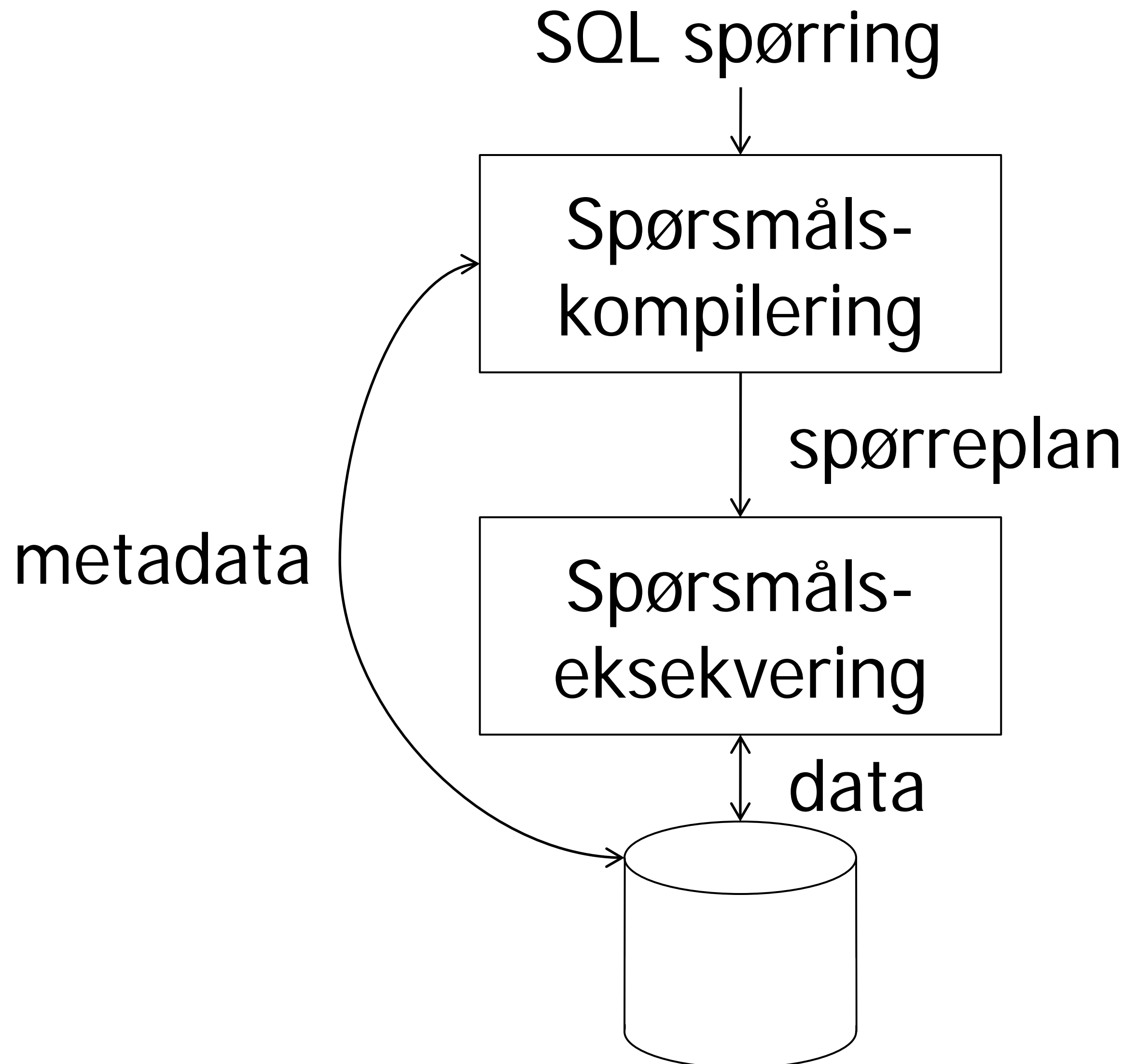
Effektiv eksekvering av spørsmål

Basert på foiler av Hector Garcia-Molina,
Ragnar Normann

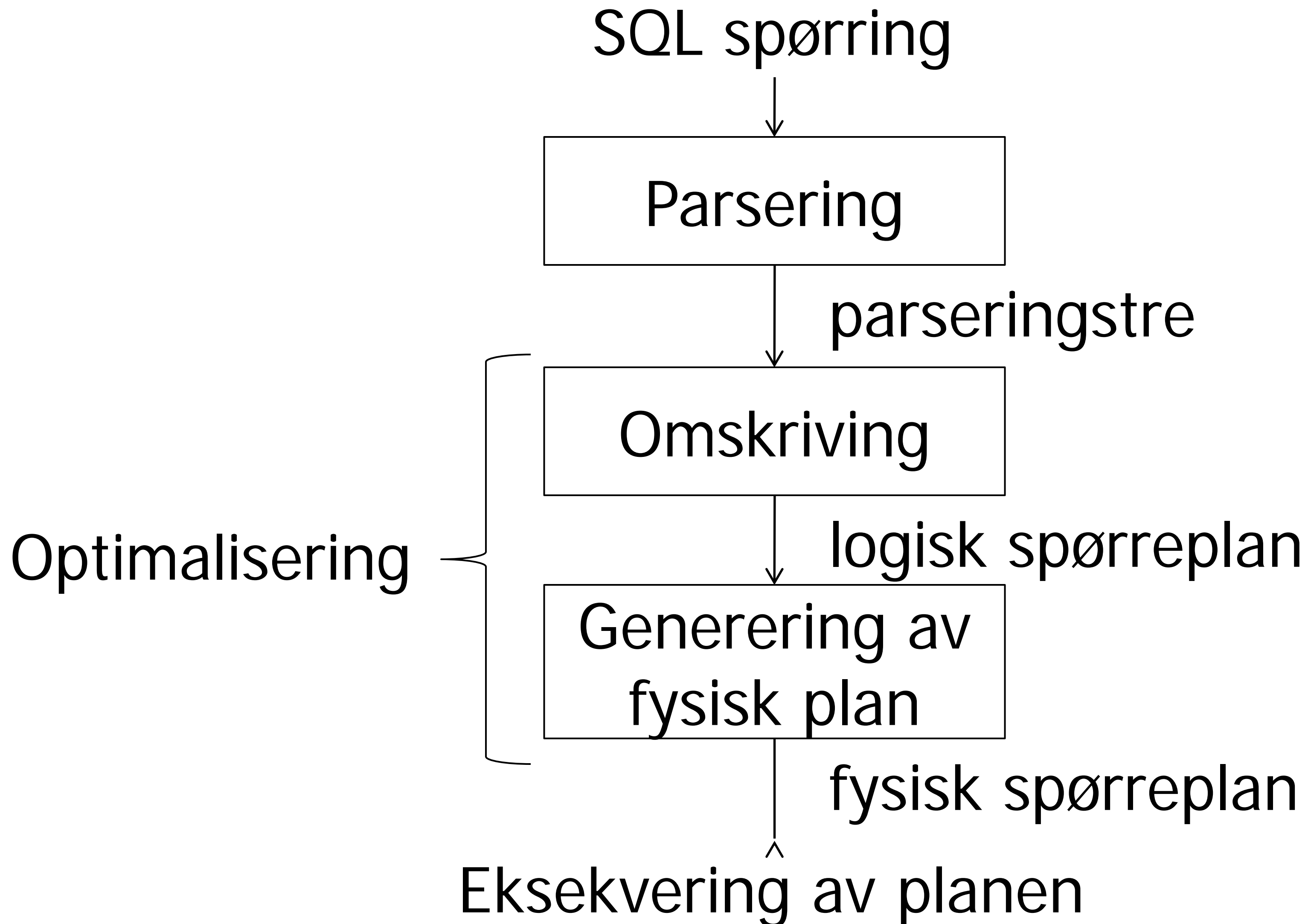
Oversikt

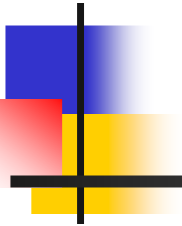
- ✓ Spørsmålshåndtering
- ✓ Modell for kostnadsberegning
- ✓ Kostnad for basis-operasjoner
- ✓ Implementasjons-algoritmer

Spørsmålshåndtering



Spørsmålskompilering





Modell for kostnadsberegning

Operatorer

- En spørring består av flere operasjoner fra relasjonsalgebraen.
  - En fysisk spørreplan implementeres av et sett operatorer tilsvarende operatorene fra relasjonsalgebraen.
  - I tillegg trenger vi basis-operatorer for å lese (skanne) en relasjon, sortere en relasjon osv.
- For å kunne velge en god plan må vi kunne estimere kostnaden til hver operator.
  - Vi vil bruke antall disk-I/O og antar generelt at
    - Operandene må initielt hentes fra disk.
    - Resultatet brukes direkte fra minnet.
    - Andre kostnader kan ignoreres.

Kostnadsparametre

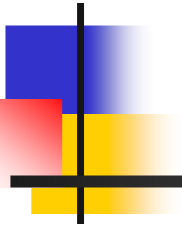
- Hvilken mekanisme som har lavest kostnad avhenger av flere faktorer, inkludert
  - Antall tilgjengelige blokker i minnet, M
  - Om det finnes indekser, og hva slags.
  - Layout på disken og spesielle disk-egenskaper.
  - …
- For en relasjon R trenger vi i tillegg å vite
  - Antall blokker som kreves for å lagre alle tuplene, B(R)
  - Antall tupler i R, T(R)
  - Antall distinkte verdier for et attributt a, V(R, a)
(gjennomsnittlig antall tupler som er like på a er da T(R)/V(R,a))

Faktorer som øker kostnaden

- ✓ Bruk av indeks som ikke ligger i minnet.
- ✓ Et sett med tupler som passer inn i b blokker kan likevel bruke b+1 blokker ved at de ikke starter på begynnelsen av den første blokken.
- ✓ Blokkene kan inneholde ledig plass for innsetting av nye tupler.
- ✓ Data kan være sortert og gruppert, lagret på hver sin blokk – fragmentering.
- ✓ R lagres sammen med andre relasjoner – clustered file organization

- ✓ Vi vil *ikke* inkludere disse faktorene i våre estimater.

Kostnad for basis-operasjoner



Kostnad for basisoperasjoner

- Kostnaden for å lese en diskblokk er 1 disk I/O
- Kostnaden for å skrive en diskblokk er 1 disk I/O
- Oppdatering koster 2 disk I/O

- En av de fundamentale operasjonene er å lese en relasjon R → kostnaden avhenger av lagringen
  - Samlet (clustered) relasjon, alle poster lagret samlet – B(R) disk I/O
  - Spredt (scattered) relasjon, poster på ulike blokker – max T(R) disk I/O

  - Vi vil generelt anta samlede relasjoner.

Sortering

- Sort-scan leser en relasjon R og returnerer R i sortert rekkefølge.
  - Kan bruke en indeks med liste av sorterte pekere, f.eks. B-trær eller en sekvensiell indeksfil.
  - Hvis hele relasjonen får plass i minnet, bruk en effektiv sorteringsalgoritme – kostnad B(R) disk I/O
  - Hvis relasjonen er for stor til å få plass i minnet, må vi bruke en sorteringsalgoritme som håndterer data i flere pass
    → ofte brukt: two-phase multiway merge sort (TPMMS)

Two-Phase, Multiway-Merge Sort (TPMMS)

- Fase 1: Sorter deler av relasjonen (så mye som det er plass til i minnet) om gangen.
  - Fyll alle tilgjengelige minneblokker med blokker som inneholder relasjonen.
  - Sorter alle blokkene i minnet.
  - Skriv de sorterte listene tilbake til disk.
  - Gjenta til alle blokkene er lest og alle postene er sortert i sub-lister.
  - ⇨ Kostnad 2B(R), dvs alle blokkene er både lest og skrevet.

Two-Phase, Multiway-Merge Sort (TPMMS)

- Fase 2: flett (merge) alle de sorterte sub-listene til en sortert liste
  - Les den første blokken til hver sub-liste inn i minnet og sammenlign det første elementet i hver blokk.
  - Plasser det minste elementet i den nye listen.
  - ⇨ Kostnad B(R)

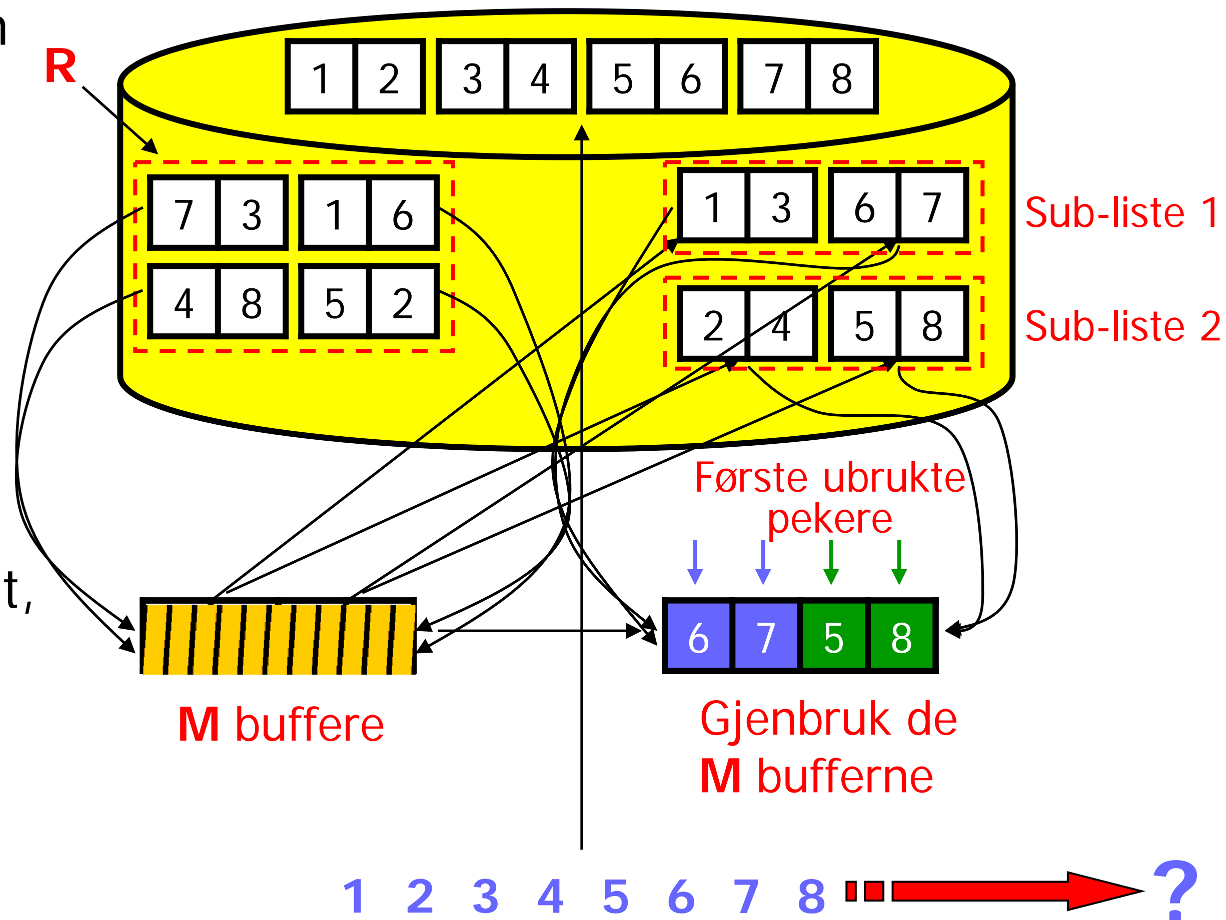
⇨ Total kostnad 3B(R)

TPMMS: Eksempel

- M=2, B(R)=4, T(R)=8
  - Fyll minnet
  - Sorter
  - Skriv tilbake sub-listen
  - Gjenta
  - Les første blokk i hver sub-liste
  - Sammenlign første *ubrukte* element
  - Output minste element, hent ny blokk hvis nødvendig
  - Gjenta de to siste stegene

Merk 1:
ofte skrives resultatet tilbake til disk, men vi antar at resultatet gis videre til en annen operator eller returneres som sluttresultatet – kostnad 3B(R)

Merk 2:
hvis R ikke er samlet, kostnad T(R)+2B(R)



Hash-partisjonering

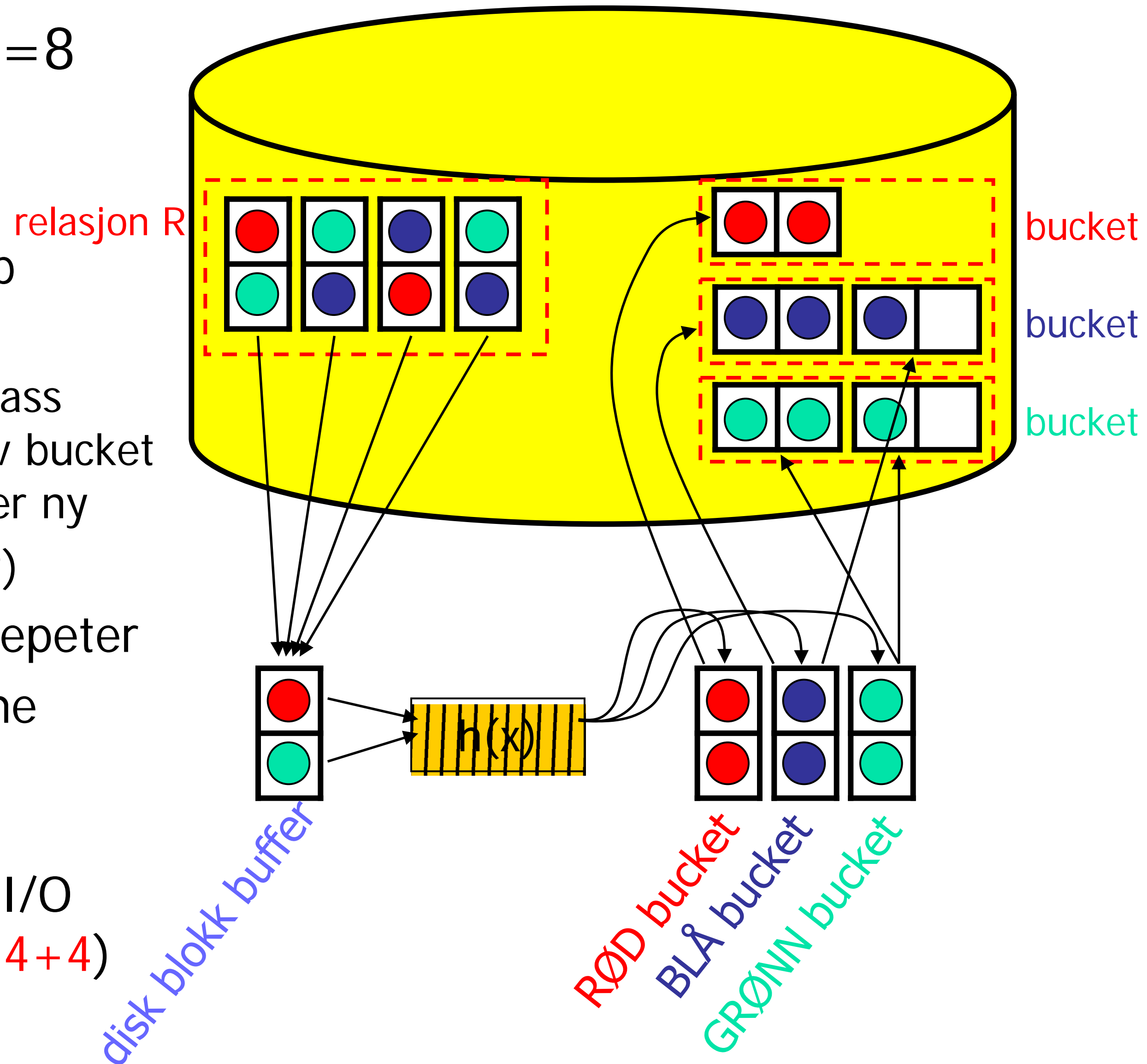
- ✓ Hash-funksjonen samler tupler som skal sees på i sammenheng.
- ✓ Med M tilgjengelige buffere:
  bruk M-1 buffere for buckets, 1 for å lese disk-blokker
- ✓ Algoritme:

```
FOR hver blokk b i relasjon R {
        les b inn i buffer M
        FOR hvert tuppel t i b {
              IF NOT plass i bucket h(t) {
                    kopier bucket h(t) til disk
                    initialiser ny blokk for bucket h(t) }
              kopier t til bucket h(t) }
        }
FOR hver ikke-tom bucket { skriv bucket til disk }
```

⇨ Kostnad 2B(R) – les alle data og skriv dem partisjonert tilbake. (NB: Dette inkluderer altså skriving tilbake!)

Hash-partisjonering: eksempel

- M=4, B(R)=4, T(R)=8
  - initialiser bufferne
  - les blokk b
  - for hvert tuppel t i b
    - beregn h(t)
    - hvis det ikke er plass i bucket h(t), skriv bucket til disk og initialiser ny
    - legg t i bucket h(t)
  - les neste blokk og repeter
  - skriv alle ikke-tomme buckets til disk
  - kostnad 2B(R) disk I/O (egentlig 4+5, ikke 4+4)



Eksekvering av spørsmål

- Tre hovedklasser med algoritmer:
  - sortings-baserte
  - hash-baserte
  - indeks-baserte

- I tillegg kan kostnad og kompleksitet deles inn i ulike nivåer
  - ett-pass algoritmer – data passer i minnet, leses bare en gang fra disk.
  - to-pass algoritmer – data for stort for minnet,
    les data, prosesser, skriv tilbake, les på nytt
  - n-pass algoritmer – rekursive generaliseringer av to-pass algoritmer for metoder som trenger flere pass over hele datasettet

Ulike grupper operatorer

- tuppel-om-gangen, unære operasjoner:
  - seleksjon (σ)
  - projeksjon (π)
- full-relasjon, unære operasjoner:
  - gruppering (γ)
  - duplikat-eliminasjon (δ)
- full-relasjon, binære operasjoner:
  - sett og bag union ($\cup$)
  - sett og bag snitt ($\cap$)
  - sett og bag differanse ($-$)
  - join ($\bowtie$)
  - produkt ($\times$)

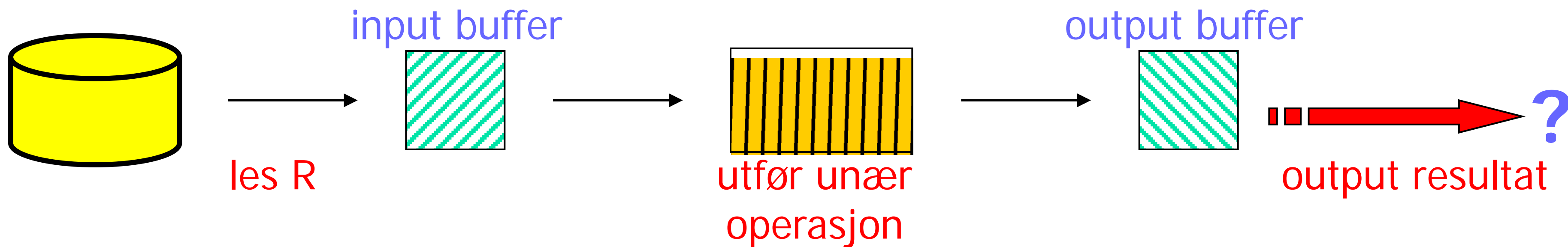
Vi skal nå se på flere måter å implementere disse operatorene på ved hjelp av ulike algoritmer og ulikt antall pass.

Unære, Tuppel-om-gangen Operasjoner

Merk at heretter foreleses bare utvalgte deler og oppsummeringer!

Tuppel-om-gangen operatorer

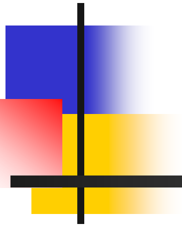
- Både seleksjon (σ) og projeksjon (π) har opplagte algoritmer – uavhengig av om relasjonen passer i minnet eller ikke:



- Eneste krav til minnet er $M \geq 1$ for input-bufferet.
- Kostnaden avhenger av hvordan R ligger
  - i minnet – 0
  - på disk, typisk
    - B(R) disk I/O hvis R ligger samlet
    - T(R) disk I/O hvis R ikke ligger samlet (max)

Tuppel-om-gangen: Worst-case

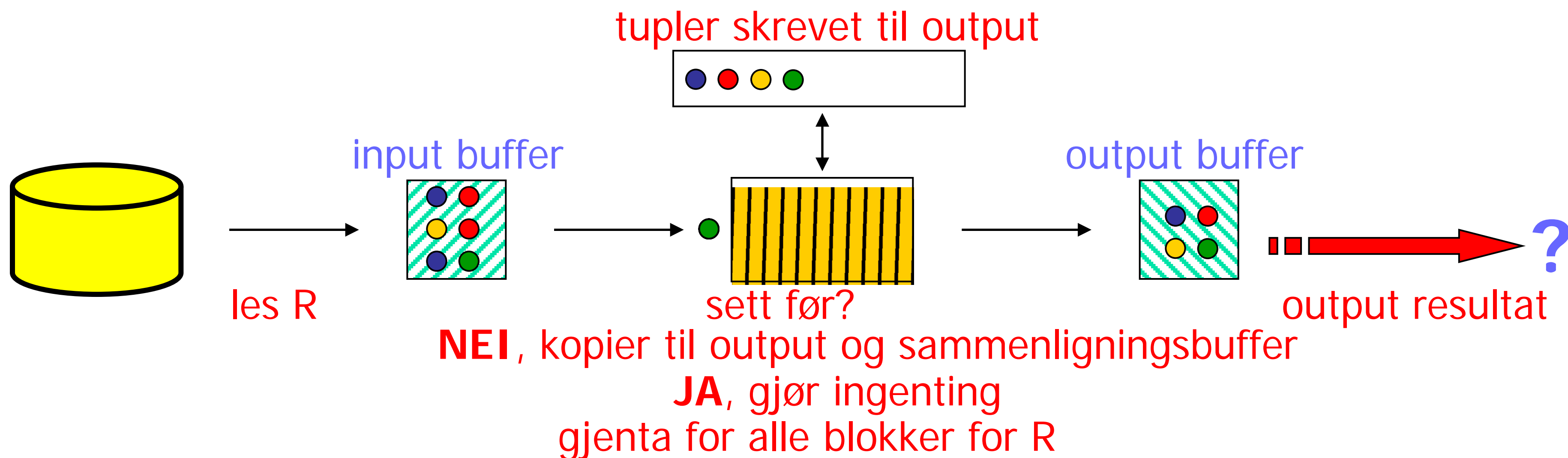
- **Worst-case** eksempel: T(R) = 20.000, B(R) = 1000, $\sigma_{a = v}(R)$
  - ingen indeks
    - R samlet – hent alle blokkene → 1000 disk I/O
    - R ikke samlet – alle tuplene er på forskjellige blokker → 20.000 disk I/O
  - index, R ikke samlet – hent T(R) / V(R, a)
    - V(R, a) = 100 → 20.000 / 100 = 200 disk I/O
    - V(R, a) = 10 → 20.000 / 10 = 2000 disk I/O
  - clustering index (R samlet) – hent B(R) / V(R, a)
    - V(R, a) = 100 → 1000 / 100 = 10 disk I/O
    - V(R, a) = 10 → 1000 / 10 = 100 disk I/O
  - V(R, a) = 20.000, dvs a er en nøkkel → 1 disk I/O



Unære, Full-relasjon Operasjoner

Duplikat-eliminasjon (δ): Ett pass

- Duplikat-eliminasjon (δ) kan utføres ved å lese en blokk om gangen og for hvert tuppel
  - kopiere til output-buffer hvis det er første forekomst
  - ignorere hvis vi har sett en duplikat
- Krever en kopi av hvert tuppel i minnet for sammenligning.



Duplikat-eliminasjon (δ): To pass

- Sorterings-basert algoritme:
  tilsvarende Two-Phase, Multiway-Merge Sort (TPMMS)
  - Les M blokker inn i minnet.
  - Sortert disse M blokkene og skriv sub-lister til disk.
  - I stedet for å flette sub-listene, kopier første tuppel og eliminer duplikater i front av sub-listene.

- Hash-basert algoritme:
  - Partisjoner relasjonen.
  - Duplikate tupler vil ha samme hash-verdi.
  - Les hver bucket inn i minnet og utfør ett-pass algoritmen som fjerner duplikater.

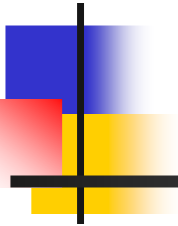
Oppsummering: Kostnad og minnekrav for $\delta(R)$

Algoritme	Minnekrav	Disk I/O
Ett-Pass	$M \geq 1 + B(\delta(R))$	B_R
To-Pass Sortering	$M \geq \sqrt{B_R}$	$3B_R$
To-Pass Hashing	$M \geq \sqrt{B_R}$	$3B_R$

Oppsummering: Kostnad og minnekrav for $\gamma(R)$

Algoritme	**Minnekrav**	**Disk I/O**
Ett-Pass	$M \geq 1 + B(\gamma(R))$	B_R
To-Pass Sortering	$M \geq \sqrt{B_R}$	$3B_R$
To-Pass Hashing	$M \geq \sqrt{B_R}$	$3B_R$

Binære, Full-relasjon Operasjoner



Binære, full-relasjon operasjoner

- En binær operasjon tar to relasjoner som argumenter:
  - union: $R \cup S$
  - snitt: $R \cap S$
  - differanse: $R - S$
  - join $R \bowtie S$
  - produkt: $R \times S$

  Merk at det er forskjell på set- og bag-versjonene av disse operatorene. (union, snitt, differanse)

  Vi skal se på *naturlig join*, de andre operatorene kan implementeres på samme måte. (join, produkt)

- Alle operasjoner som krever sammenligning må bruke en søkestruktur (f.eks. binærtrær eller hashing) som også krever ressurser. Vi tar imidlertid ikke med disse i våre estimater.

Union (∪) : Ett-Pass

- **Bag** union (R∪S) kan beregnes med en veldig enkel ett-pass algoritme:
  - Les og kopier hvert tuppel i R til output bufferet.
  - Les og kopier hvert tuppel i S til output bufferet.
- Total kostnad B(R) + B(S) disk I/O
- Minnekrav 1 (les blokk direkte til output buffer)

- **Sett** union må fjerne duplikater
  - Les den *minste* relasjonen (anta S) inn i M-1 buffere og kopier hvert tuppel til output.
  - Les blokkene i R inn i det siste bufferet, og sjekk for hvert tuppel om det eksisterer i S → hvis ikke, kopier til output
- Minnekrav er nå 1 + (M-1) = M, $B(S) < M$

Union (∪) : To-Pass Sortering

- **Sett** union må fjerne duplikater
  - Gjør fase 1 av TPMMS for både R og S (lag sorterte sub-lister).
  - Bruk ett buffer for hver av sub-listene til R og S.
  - Gjenta: Finn første gjenværende tuppel i hver sub-liste
    - Output tuppelet.
    - Fjern duplikater fra fronten av alle listene

- Total kostnad er $3B(R) + 3B(S)$ disk I/O
- Minnekrav
  - M buffere kan lage M blokk lange sub-lister (totalt).
  - $B(R) + B(S) \leq M^2 \rightarrow \sqrt{(B(R)+B(S))} \leq M$
  - hvis $B(R) + B(S) > M^2$ → mer enn M sub-lister og algoritmen vil ikke fungere.

Union (∪) : To-Pass Hashing

- **Sett** union to-pass hash algoritme:
  - Partisjoner både R og S i M-1 buckets ved samme hash-funksjon.
  - Gjør union på hvert bucket-par – $R_i \cup S_i$ – separat (ett-pass sett union)

- Total kostnad: $3B(R) + 3B(S)$ disk I/O
  - 2 for partisjoneringen
  - 1 for union av de ulike buckets

- Minnekrav: M buffere
  - M buffere kan lage M-1 buckets for hver relasjon
  - for hvert bucket-par, R_i and S_i, enten $B(R_i) \leq M-1$ eller $B(S_i) \leq M-1$
  - tilnærmet $\min(B(R), B(S)) \leq M^2 \rightarrow \sqrt{\min(B(R), B(S))} \leq M$
  - hvis den minste av R_i og S_i ikke får plass i M-1 buffere vil algoritmen ikke fungere.

Oppsummering: kostnad og minnekrav for $R \cup S$:

Hvis $B_S \leq B_R$:

BAG-versjon :

Algoritme	Minnekrav	Disk I/O
Ett-Pass	$M \geq 1$	$B_R + B_S$

SETT-versjon :

Algoritme	Minnekrav	Disk I/O
Ett-Pass	$B_S \leq M - 1$	$B_R + B_S$
To-Pass Sortering	$M \geq \sqrt{}B_R + B_S$	$3B_R + 3B_S$
To-Pass Hashing	$M \geq \sqrt{}B_S$	$3B_R + 3B_S$

Oppsummering: Kostnad og minnekrav for $R \cap S$:

Hvis $B_S \leq B_R$:

Algoritme	Minnekrav[1]	Disk I/O
Ett-Pass	$B_S \leq M - 1$	$B_R + B_S$
To-Pass Sortering	$M \geq \sqrt{B_R + B_S}$	$3B_R + 3B_S$
To-Pass Hashing	$M \geq \sqrt{B_S}$	$3B_R + 3B_S$

[1]**BAG**-versjonen trenger i tillegg plass til tuppel-tellere.

Oppsummering: Kostnad og minnekrav for R–S:

Hvis $B_S \leq B_R$:

Algoritme	Minnekrav[1]	Disk I/O
Ett-Pass	$B_S \leq M - 1$	$B_R + B_S$
To-Pass Sortering	$M \geq \sqrt{B_R + B_S}$	$3B_R + 3B_S$
To-Pass Hashing	$M \geq \sqrt{B_S}$	$3B_R + 3B_S$

[1]**BAG**-versjonen trenger i tillegg plass til tuppel-tellere.

Naturlig join (⋈) : Ett-Pass

- ✓ Naturlig join (⋈) konkatenerer tupler fra R(X,Y) og S(Y,Z) som er slik at R.Y = S.Y
- ✓ Ett-pass algoritme:
  - Les den *minste* relasjonen (anta S) inn i M-1 buffere.
  - Les relasjon R blokk for blokk. For hvert tuppel t
    - Join t med matchende tupler i S
    - Flytt resultatet til output
- ✓ Total kostnad B(R) + B(S) disk I/O
- ✓ Minnekrav 1 + (M-1) = M, B(S) < M

Naturlig Join (⋈) : Nestet-Loop Join – tuppel-basert algoritme

- Nestet-loop join kan brukes for relasjoner av vilkårlig størrelse
- *Tuppel-basert* algoritme:

```
FOR hvert tuppel s i relasjon S
    FOR hvert tuppel r i R
        IF  r og s matcher, join til output
```

- Worst case kostnad T(R)T(S) disk I/O
- Minnekrav 2 (en R-blokk og en S-blokk)

Naturlig Join (⋈) : Nestet-Loop Join – blokk-basert algoritme

- Hold så mye som mulig av den minste relasjonen (anta S) i M-1 blokker
- Algoritme:

```
FOR hver partisjon p av S med størrelse M-1 {
        les p inn i minnet
        FOR hver blokk b for R {
                les b inn i minnet
                FOR hvert tuppel t i b {
                        finn tupler i p som matcher t
                        join hver av disse med t til output }}}
```

Bare ett pass gjennom tuplene til R!

- Total kostnad B(S) + [B(R)*B(S)/(M-1)] disk I/O
  (Les S en gang, les R en gang for hver S-partisjon)
- Minnekrav 2 (en R-blokk og en S-blokk)

Naturlig join (⋈) : To-Pass Sortering – enkel algoritme

- Sorter R og S hver for seg ved hjelp av TPMMS på join-attributtene.
- Join de sorterte R og S ved å
  - Hvis et buffer for R eller S er tomt, hent ny blokk fra disk
  - Finn tupler som har den minste v-verdien for join-attributtene (NB: Kan også finnes i påfølgende blokker!)
  - Hvis det finnes v-verdi tupler i både R og S, join disse og skriv til output
  - Ellers, fjern alle v-verdi tupler.
- Total kostnad: $5B(R) + 5B(S)$ disk I/O
  - 4 for TPMMS
  - 1 for å flette de sorterte R og S
- Minnekrav: M buffere
  - TPMMS på begge relasjonene krever $B \leq M^2$, dvs, $B(R) \leq M^2$ og $B(S) \leq M^2$
  - Hvis det finnes en samling v-verdi tupler som ikke passer i M blokker, virker ikke algoritmen.

Naturlig Join (⋈) : To-Pass Hashing

- Algoritme:
  - Partisjoner både R og S i M-1 buckets med samme hash-funksjon.
  - Gjør naturlig join på hver bucket separat – $R_i \bowtie S_i$ – ett-pass join.

- Total kostnad: 3B(R) + 3B(S) disk I/O
  - 2 for å partisjonere relasjonene
  - 1 for å gjøre join

- Minnekrav: M buffere
  - M buffere kan lage M-1 buckets for hver relasjon
  - for hvert par R_i og S_i er enten $B(R_i) \leq M-1$ eller $B(S_i) \leq M-1$
  - omtrent $\min(B(R), B(S)) \leq M^2 \rightarrow \sqrt{\min(B(R), B(S))} \leq M$
  - hvis den minste av R_i og S_i ikke får plass i M-1 buffere, virker ikke algoritmen

Naturlig Join (⋈): Indeks-basert

- Algoritme R(X,Y) ⋈ S(Y,Z) :
  - Anta at det finnes en indeks for S på attributt(er) Y.
  - Les hver blokk for R. For hvert tuppel
    - finn tupler i S med samme join-attributter ved hjelp av indeksen
    - les de tilsvarende blokkene, gjør join på de aktuelle tuplene og output resultatet
- Total kostnad: ? disk I/O
  - Hvis R er samlet, trenger vi B(R) disk I/O, ellers opp til T(R) for å lese alle R-tupler.
  - I tillegg, for *hvert tuppel i R,* må vi lese de tilsvarende S-tuplene:
    - For clustered indeks sortert på Y: B(S) / V(S,Y)
    - Hvis S ikke er sortert på Y: T(S) / V(S,Y)
    - Vil i snitt bruke T(S) / V(S,Y)
  - ⇨ Dvs at lesing av S-tupler dominerer: T(R)T(S) / V(S,Y)

Oppsummering: Kostnad og minnekrav for R ⋈ S

Hvis $B_S \leq B_R$:

Algoritme	Minnekrav	Disk I/O
Ett-Pass	$B_S \leq M - 1$	$B_R + B_S$
Tuppel-Basert Nestet-Loop	$2 \leq M$	worst case $T_R T_S$
Blokk-Basert Nestet-Loop	$2 \leq M$	$B_S + [(B_S / M\text{-}1) \times B_R]$
Enkel To-Pass Sortering	$\sqrt{B_R} \leq M$	$5\ B_R + 5\ B_S$
Sort-Join	$\sqrt{B_R + B_S} \leq M$	$3\ B_R + 3\ B_S$
Hash-Join	$\sqrt{B_S} \leq M$	$3\ B_R + 3\ B_S$
Hybrid Hash-Join	$\sqrt{B_S} \leq M$	$(3 - 2M/B_S)(B_R + B_S)$
Indeks-Join	$2 \leq M$	$B_R + (T_R B_S / V_{S,Y})$
Zig-Zag Indeks-Join	$B(T_R/V_{R,a}) + B(T_S/V_{S,a}) \leq M$	$B_R + B_S$

Naturlig join: eksempel

- Eksempel:
  - T(R) = 10.000, T(S) = 5.000
  - V(R,a) = 100, V(S,a) = 10
  - Både R og S er samlede
  - 4 KB blokker (ingen blokk-header)
  - Både R og S poster er på 512 B (inkludert header)
  - Har clustering indeks på attributt a for både R og S

  ⇨ B(S) = 5.000 / 8 = 625
  B(R) = 10.000 / 8 = 1250

Naturlig join: eksempel (forts.)

- T(R) = 10.000, T(S) = 5.000, B(R) = 1250, B(S) = 625, M = 101

Algoritme	Minimum minne	Disk I/O
Ett-Pass	626	1875
Tuppel-Basert Nestet-Loop	2	781875
Blokk-Basert Nestet-Loop	2	9375
Enkel To-Pass Sortering	36	9375
Sort-Join	44	5625
Hash-Join	25	5625
Hybrid Hash-Join	25	5019
Indeks-Join	2	626250 (S-indeks) 63125 (R-indeks)
Zig-Zag Indeks-Join	76	1875

Valg av algoritme

- ✓ Ett-Pass algoritmer er bra hvis ett av argumentene (relasjonene) får plass i minnet.
- ✓ To-Pass algoritmer må brukes hvis vi har store relasjoner.
  - Hash-baserte algoritmer
    - Krever mindre minne sammenlignet med sorterings-tilnærminger – er bare avhengig av den minste relasjonen – brukes ofte.
    - Antar omtrent lik bucket-størrelse (god hash-funksjon) – i virkeligheten vil det være noe variasjon, må anta mindre bucket-størrelser.
  - Sorterings-baserte algoritmer
    - Gir sortert resultat, som kan utnyttes av påfølgende operatorer.
  - Indeks-baserte algoritmer
    - Utmerket for seleksjon og for join hvis begge har clustered indekser.

N–Pass Algoritmer

- For virkelig store relasjoner er ikke to pass tilstrekkelig.
- Eksempel: B(R) = 1.000.000
  - TPMMS krever at $B(R) < M^2$ → M > 1000
  - Hvis ikke 1000 blokker er tilgjengelige, virker ikke TPMMS.
  - ⇨ Trenger flere pass gjennom datasettet.
- Sorterings-baserte algoritmer:
  - Hvis R får plass i minnet, sorter.
  - Hvis ikke, partisjoner R i M grupper og sorter hver R_i rekursivt.
  - Flett sub-listene.
  - Total kostnad: (2k– 1)B(R), k er antall pass som er nødvendig
  - Vi trenger $\sqrt[k]{B(R)}$ buffere, dvs $B(R) \leq M^k$
- Tilsvarende kan gjøres for hash-baserte algoritmer.