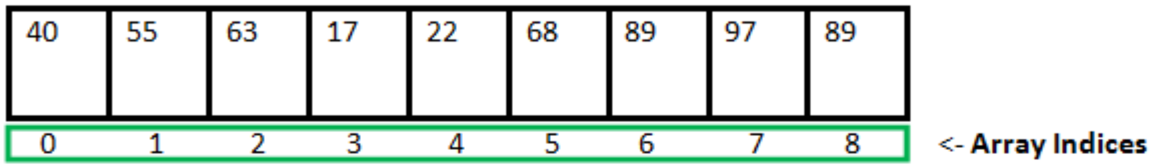
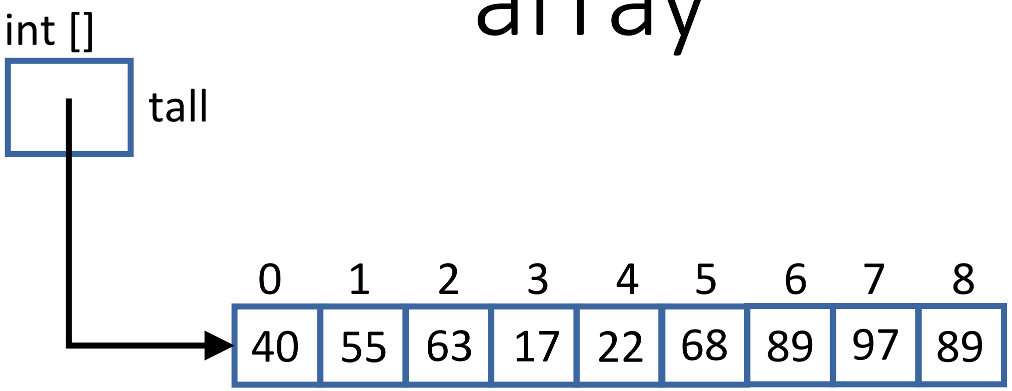
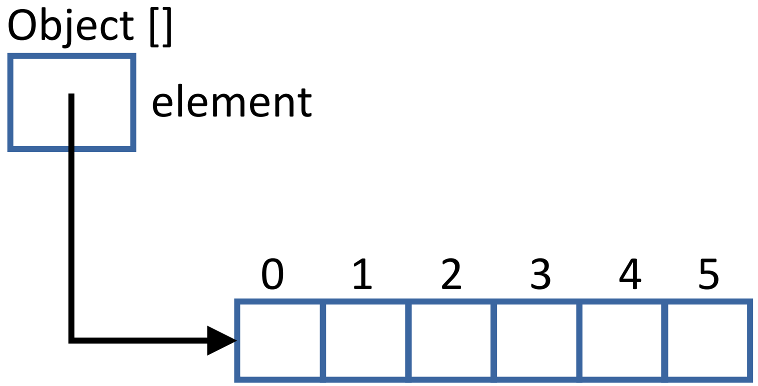
Datastrukturer

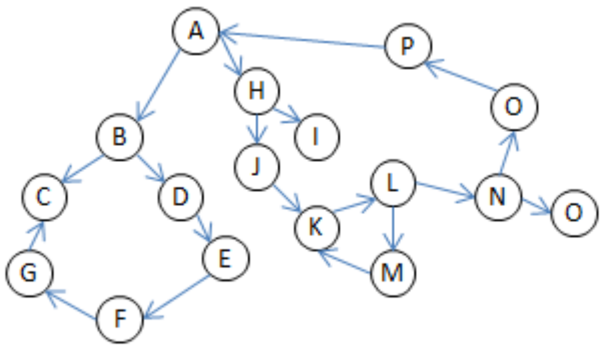


Array Length = 9
First Index = 0
Last Index = 8

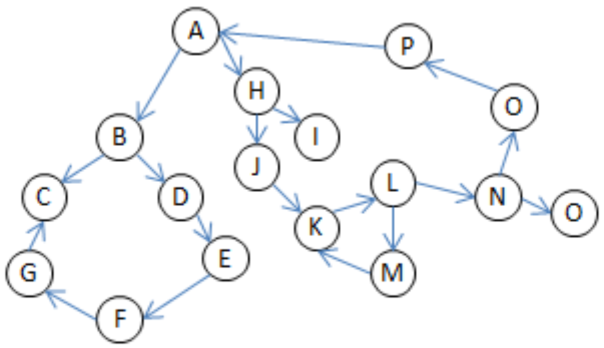
array





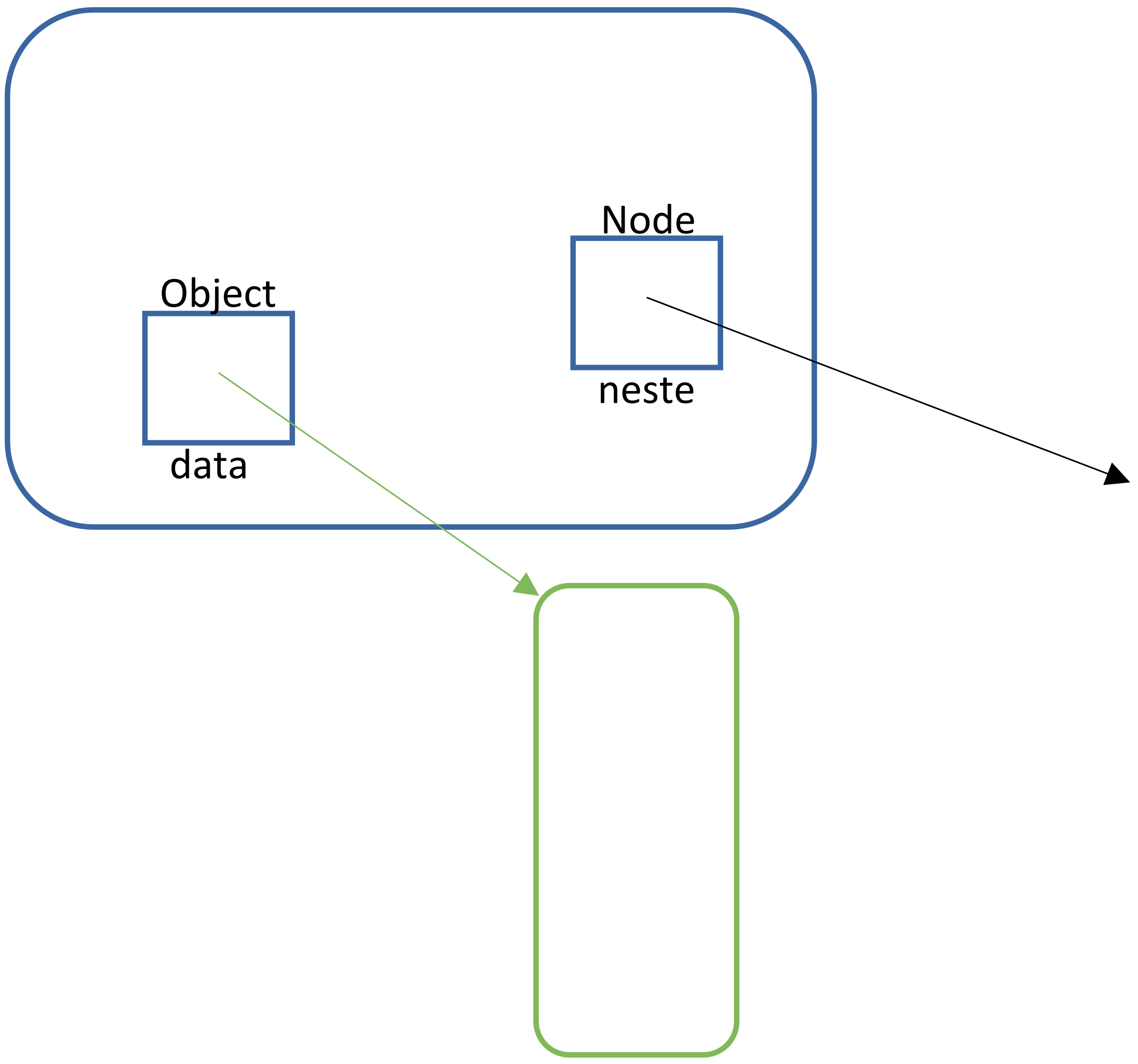


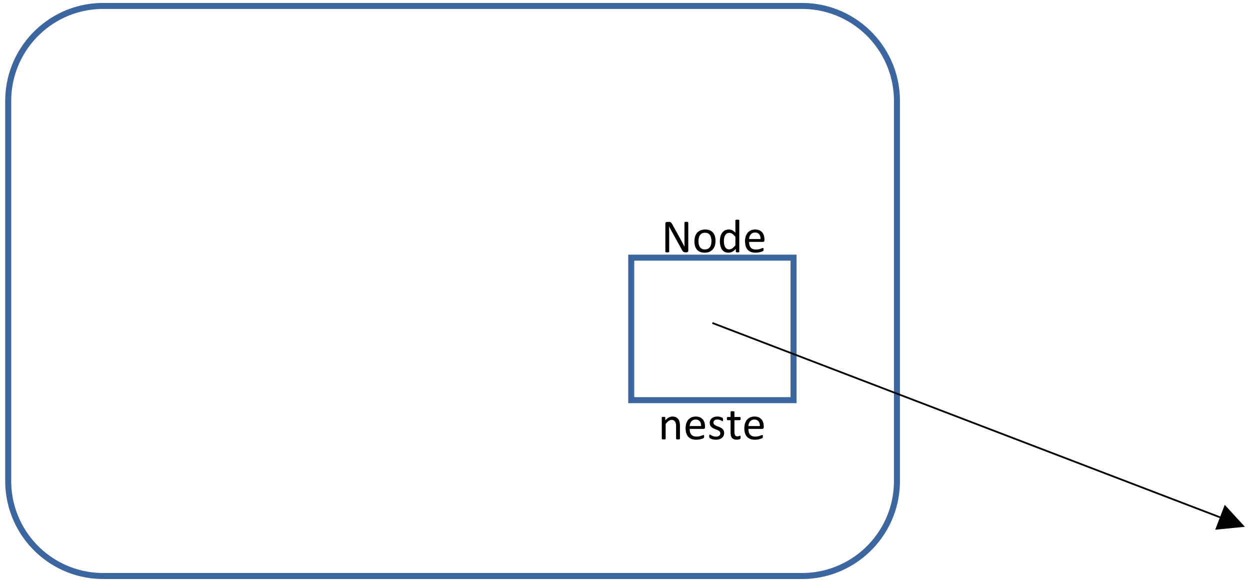
graf

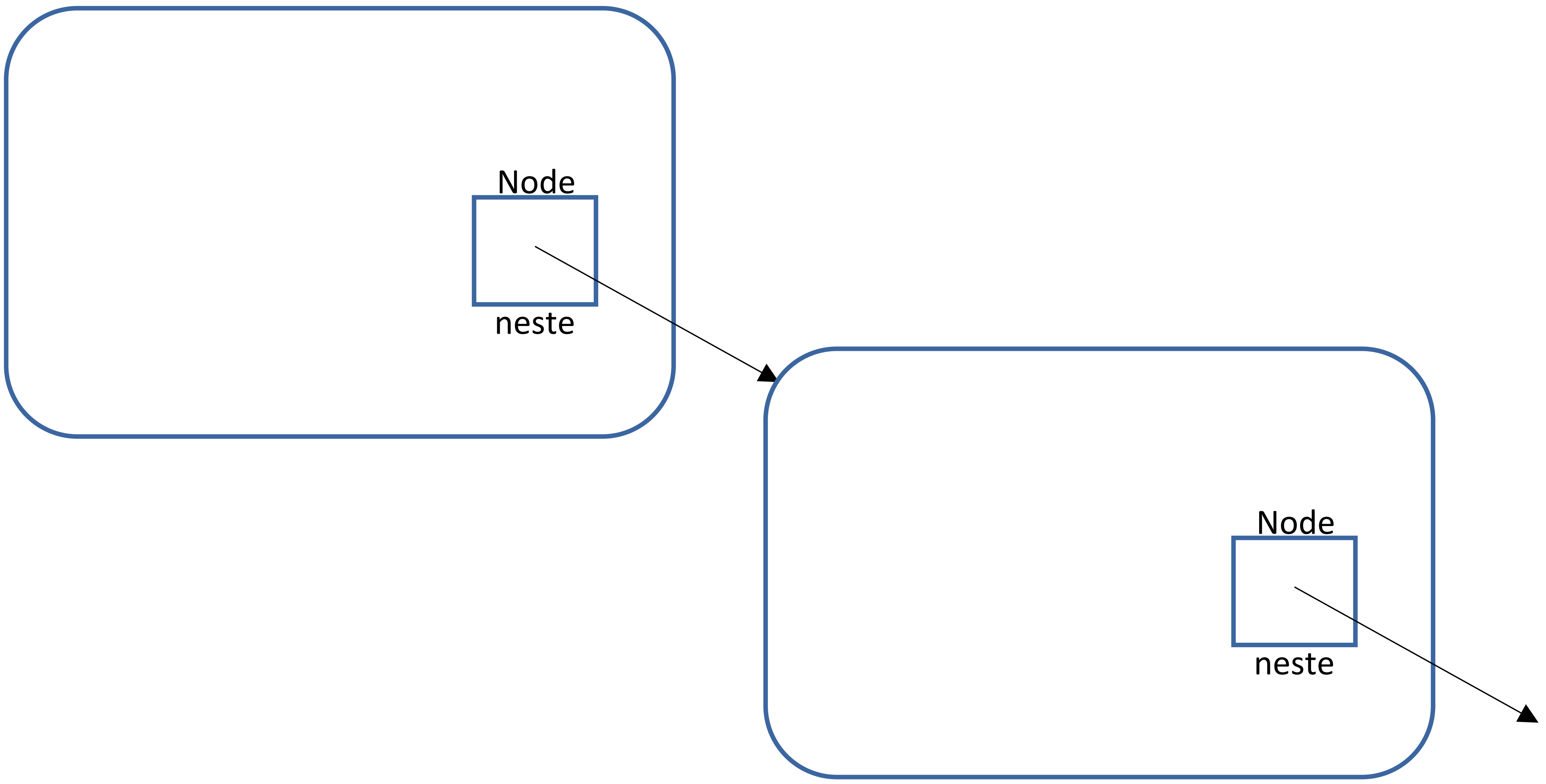


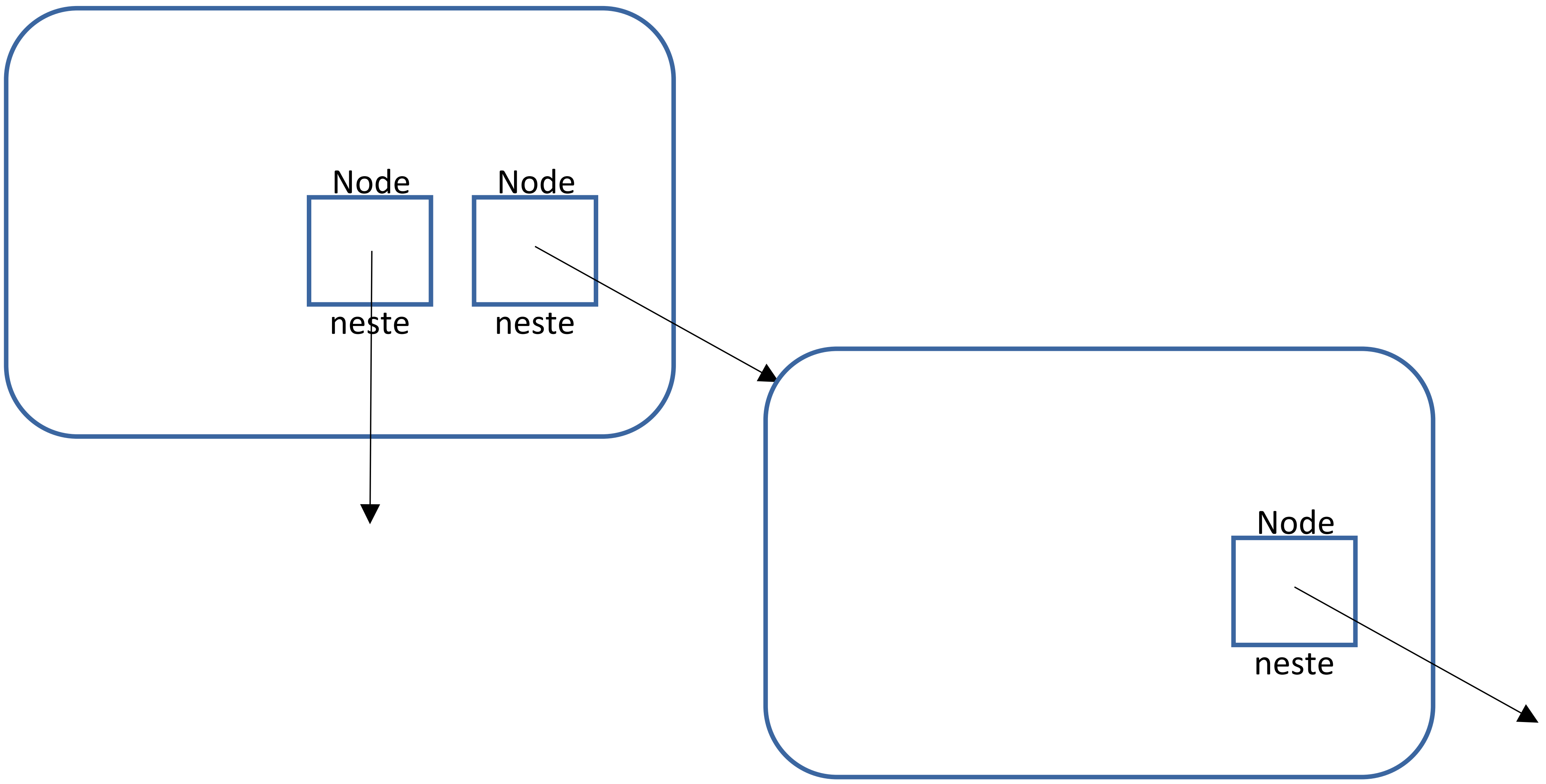
graf

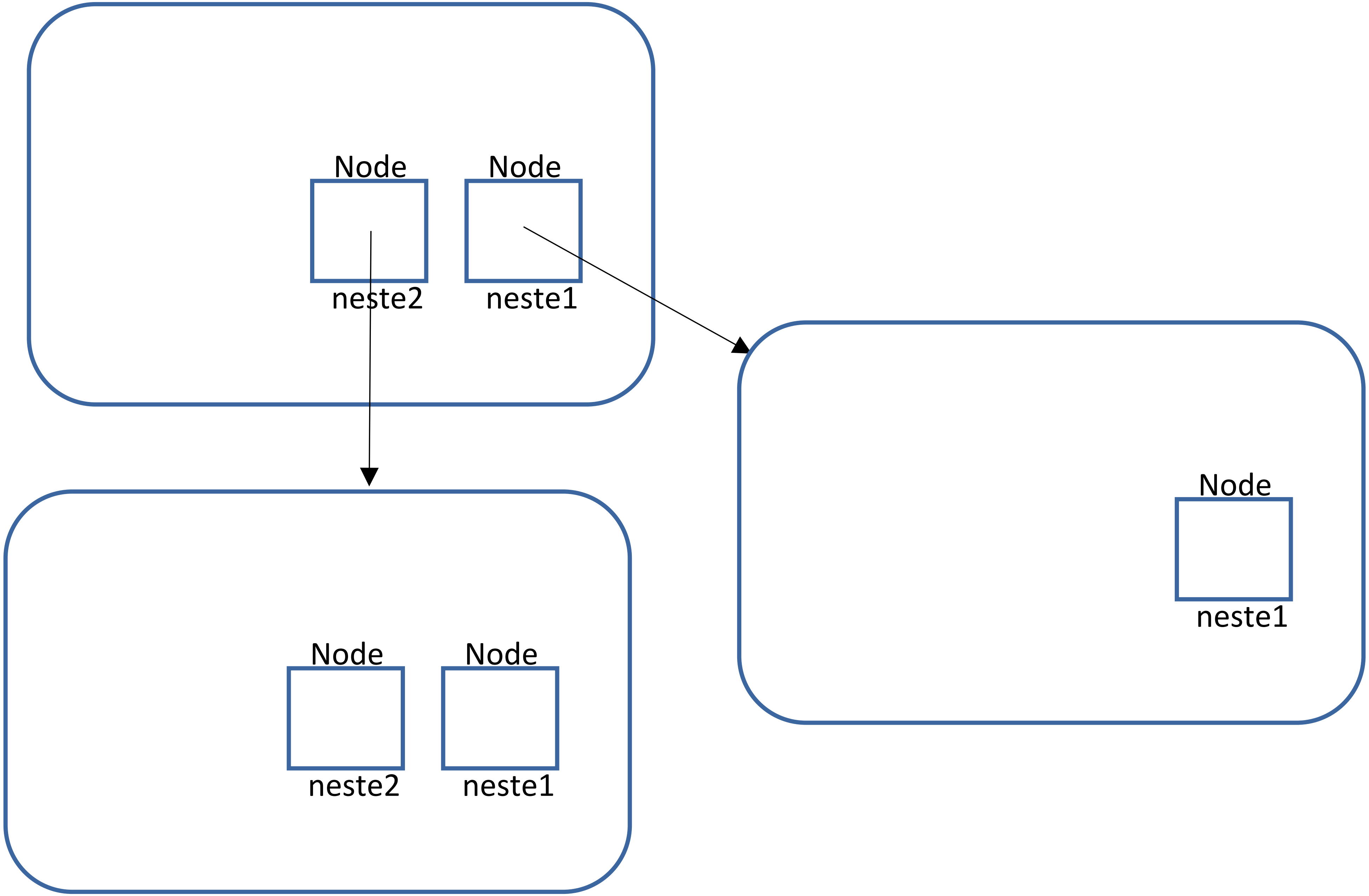
lenkeliste

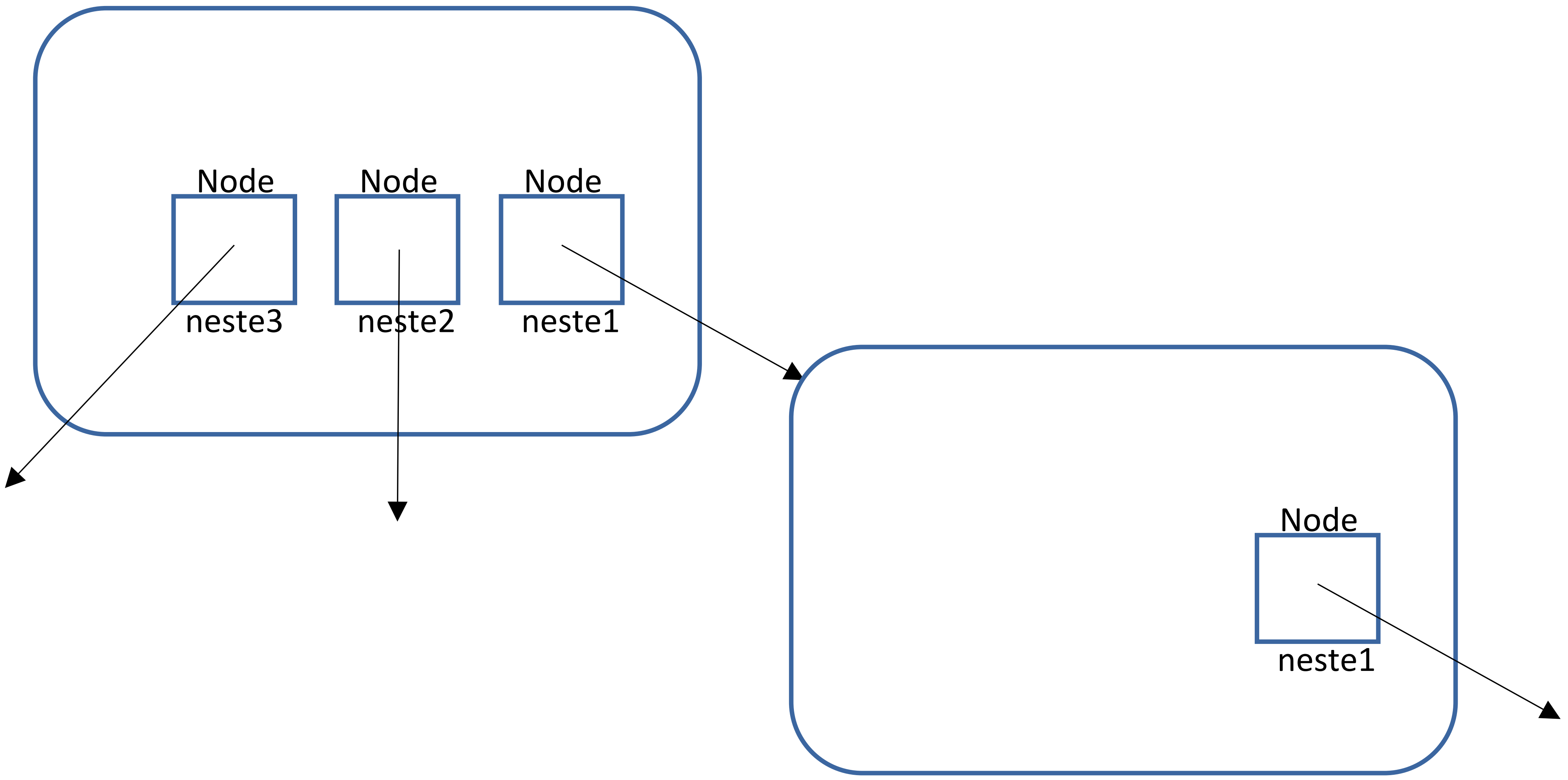

Node
Object
neste
data

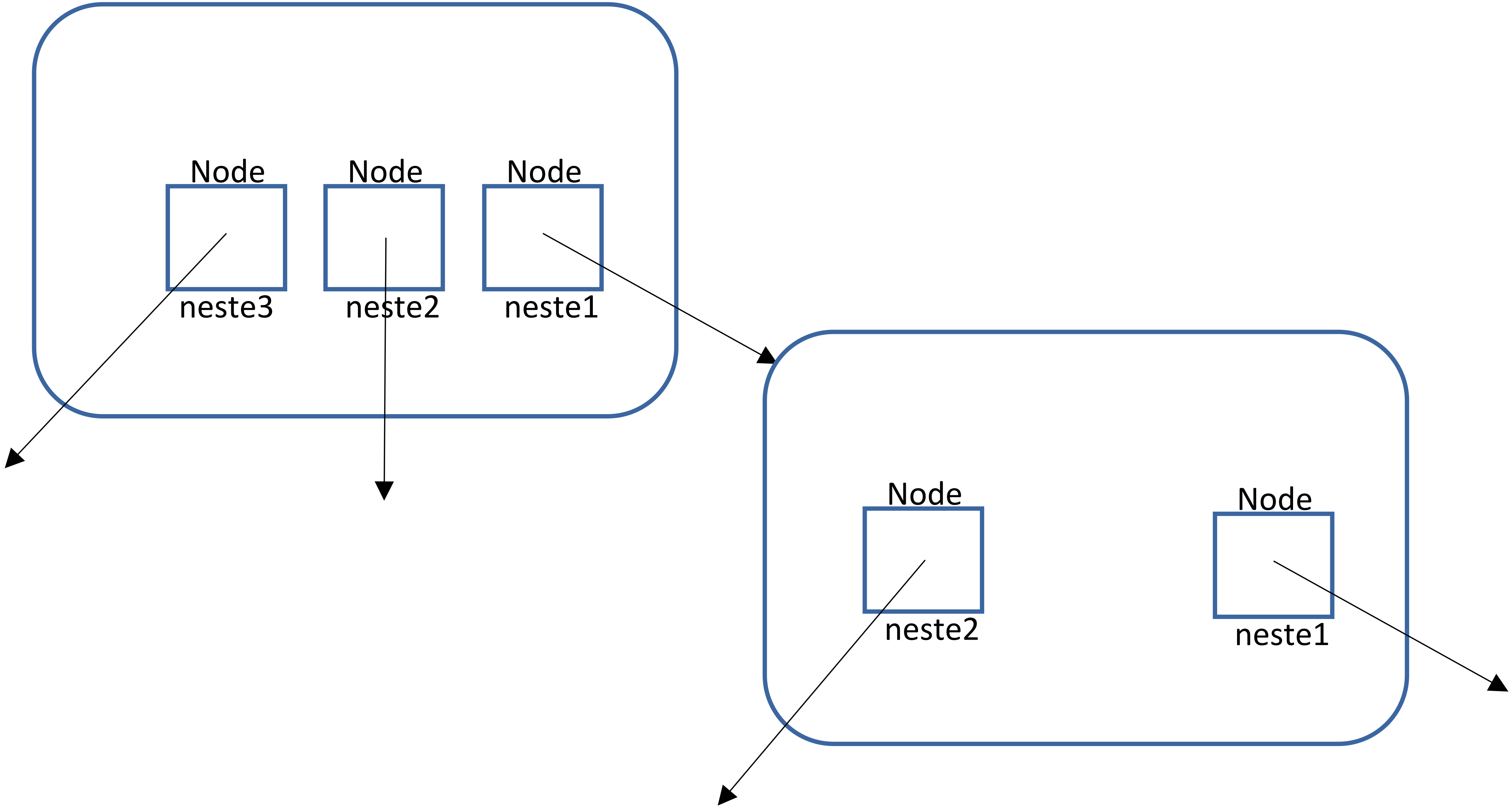

Node
neste

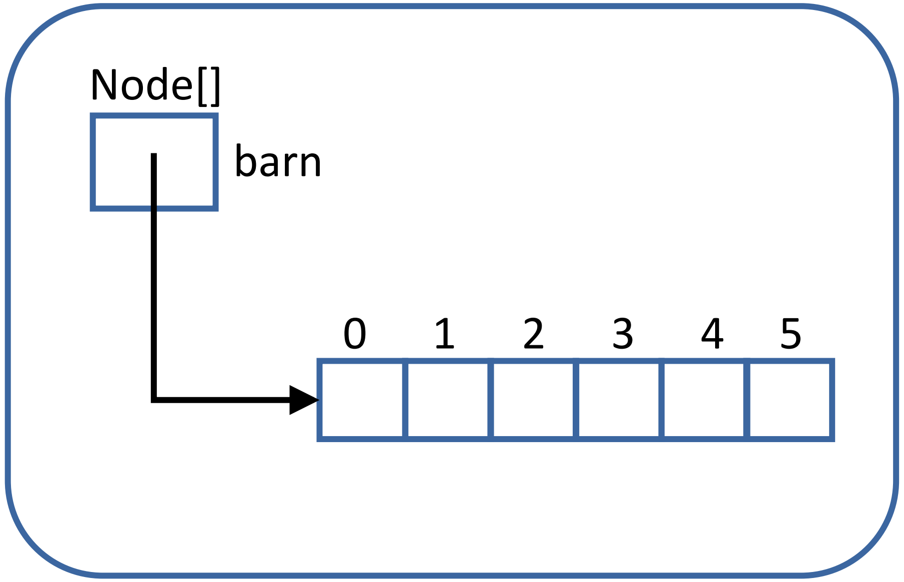

Node
neste
Node
neste

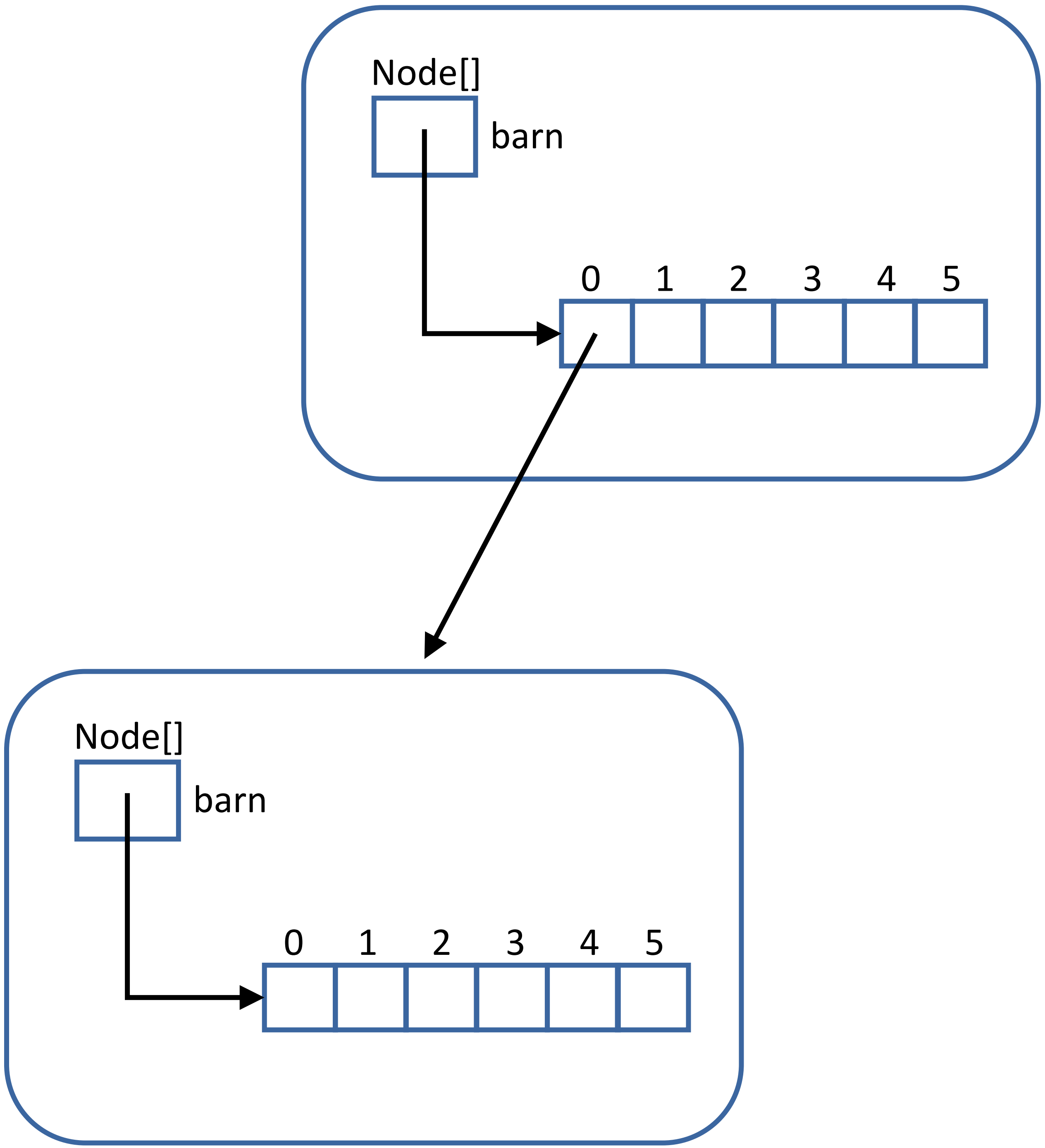

Node
Node
neste
neste
Node
neste

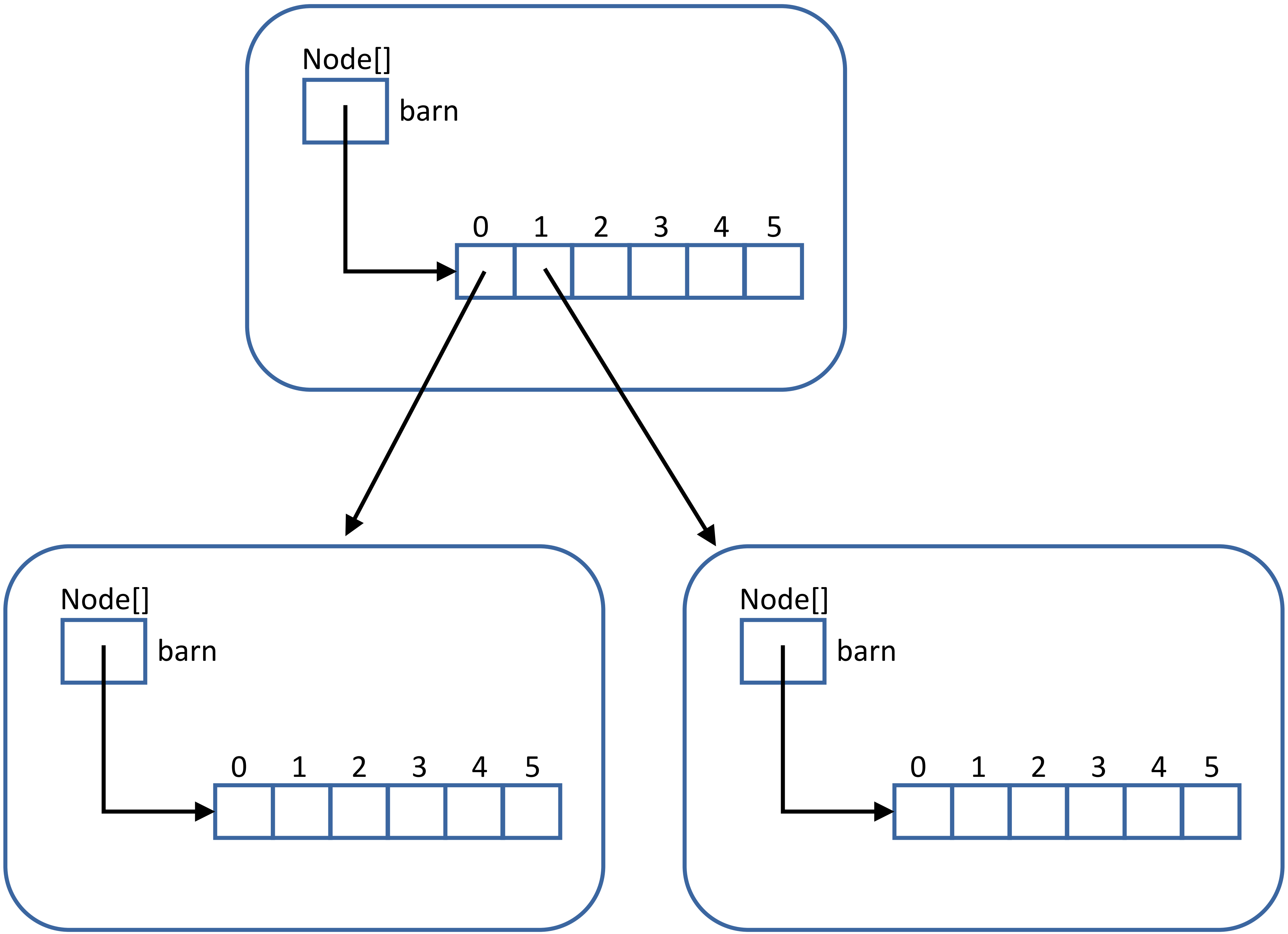

Node
Node
neste2
neste1
Node
neste1
Node
Node
neste2
neste1


Node
Node
Node
neste3
neste2
neste1
Node
neste1

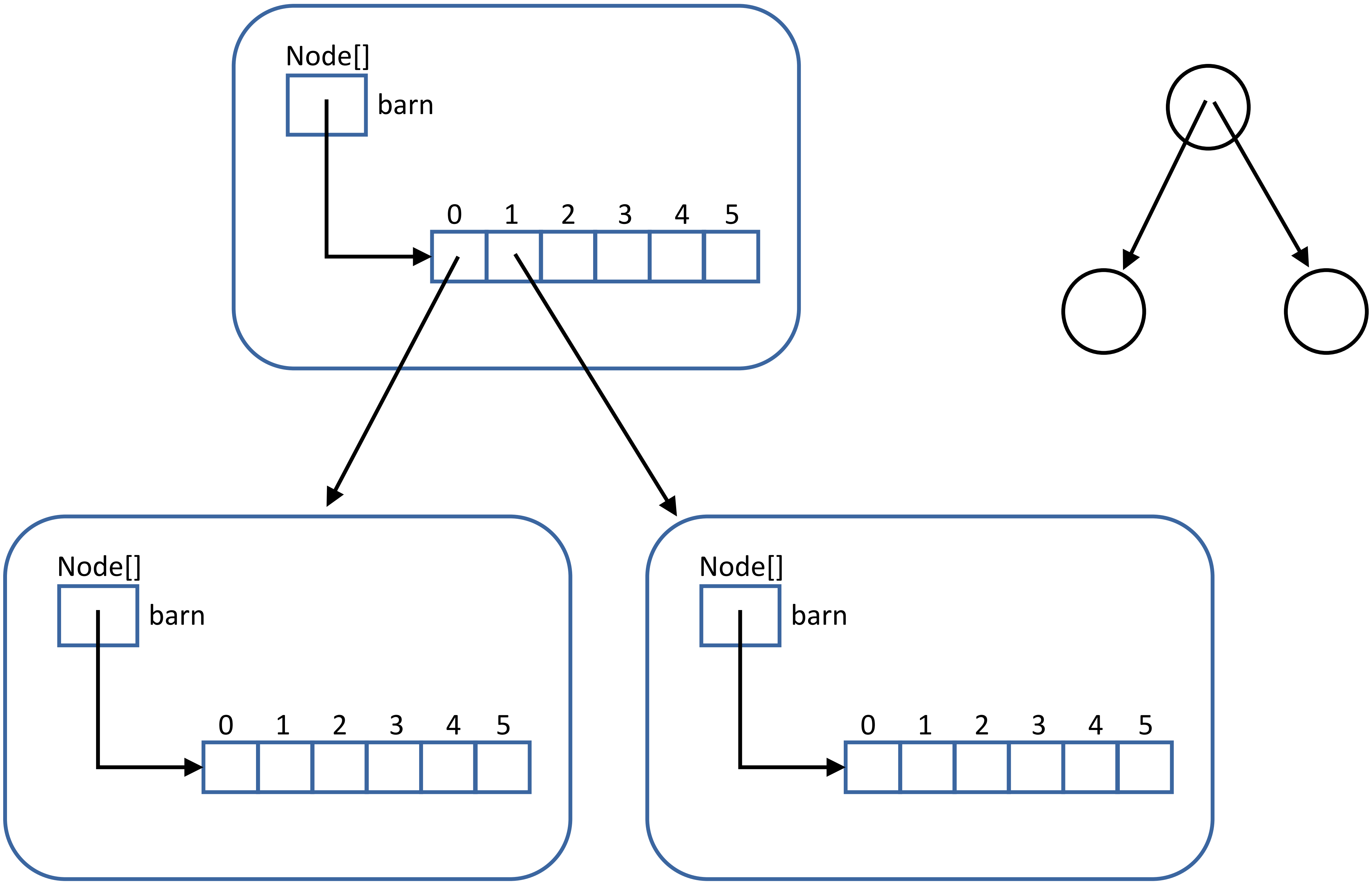

Node
Node
Node
neste3
neste2
neste1
Node
Node
neste2
neste1


Node[]
barn
0
1
2
3
4
5

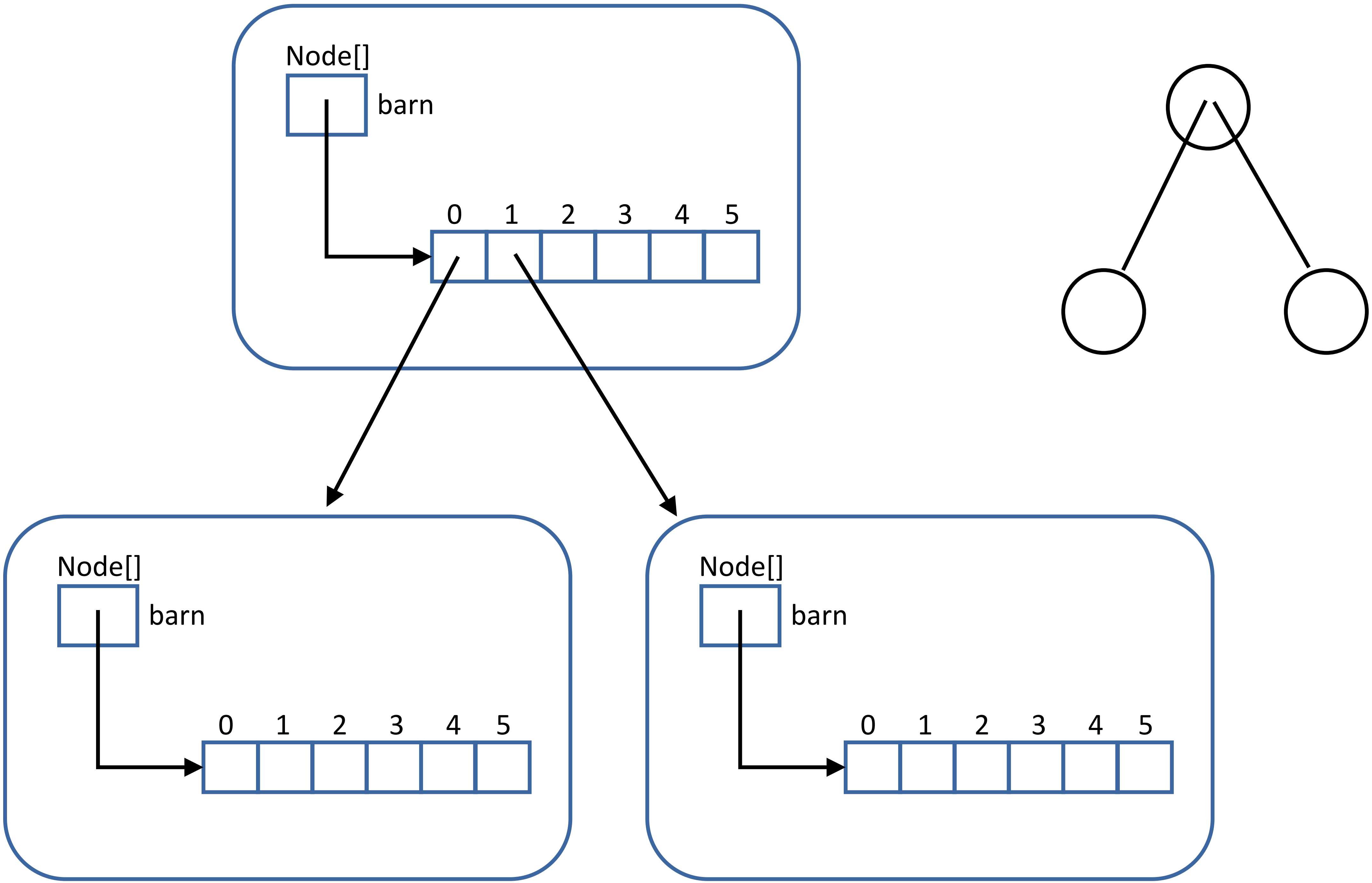

Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5

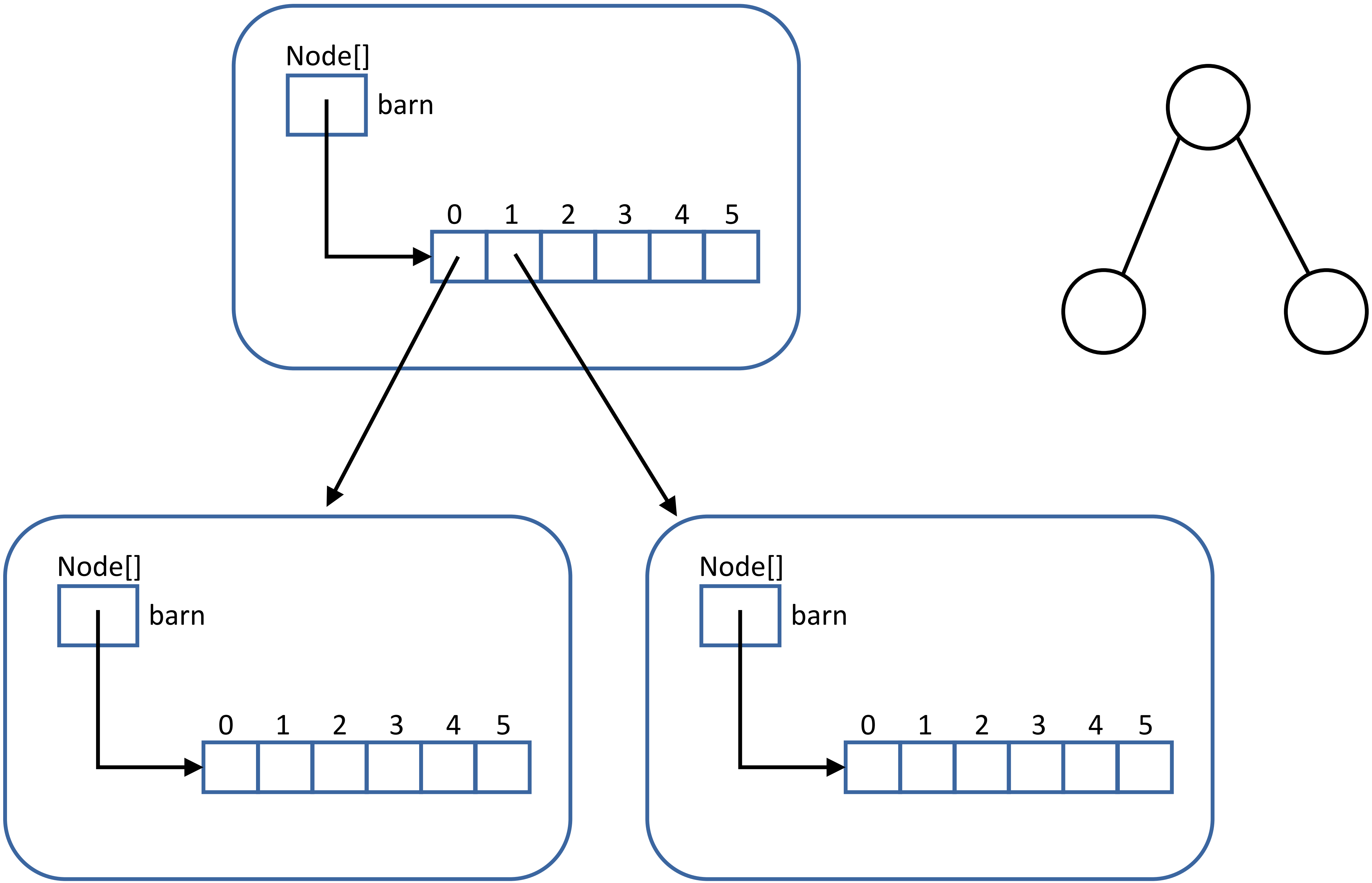


Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5

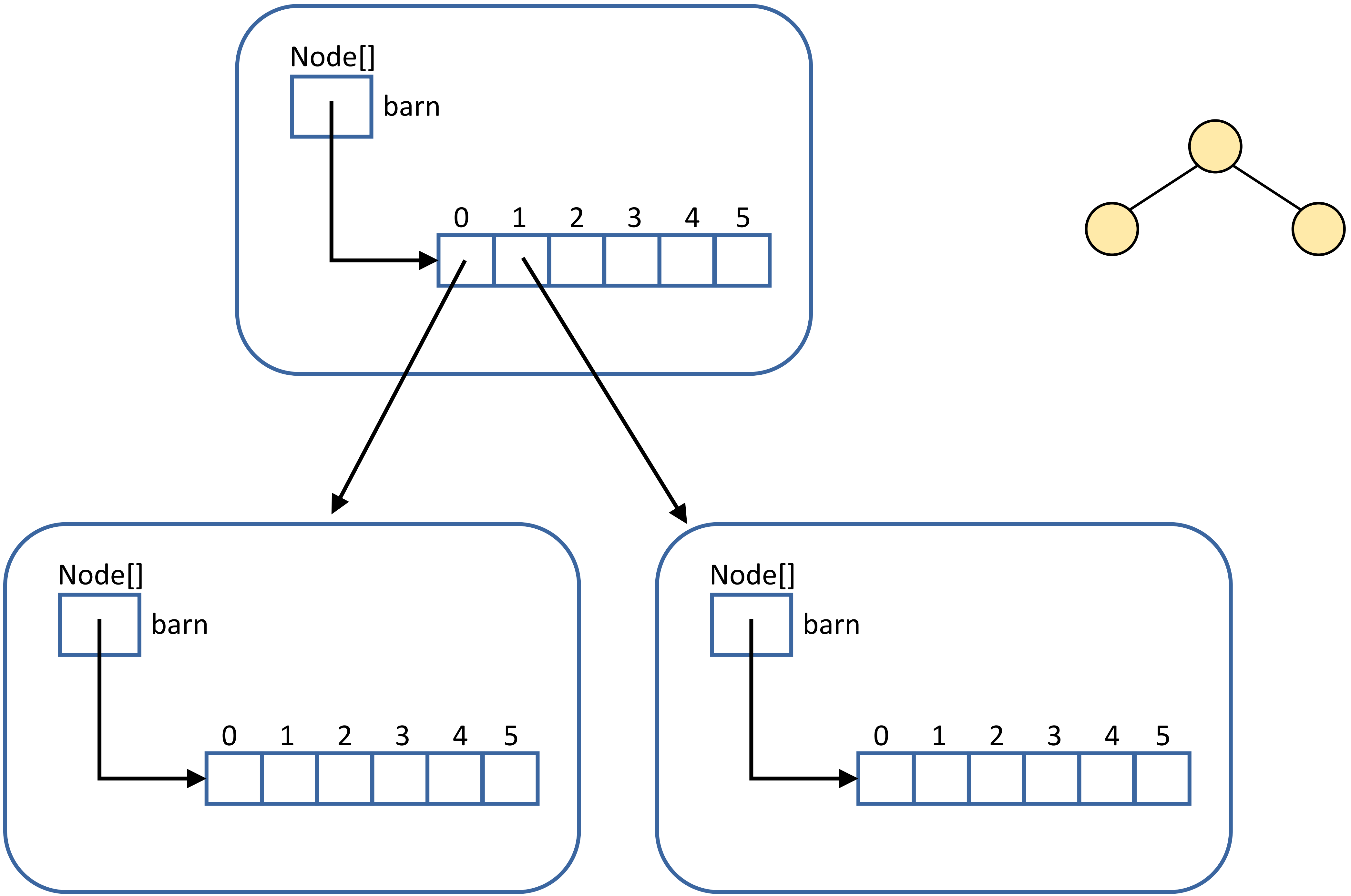


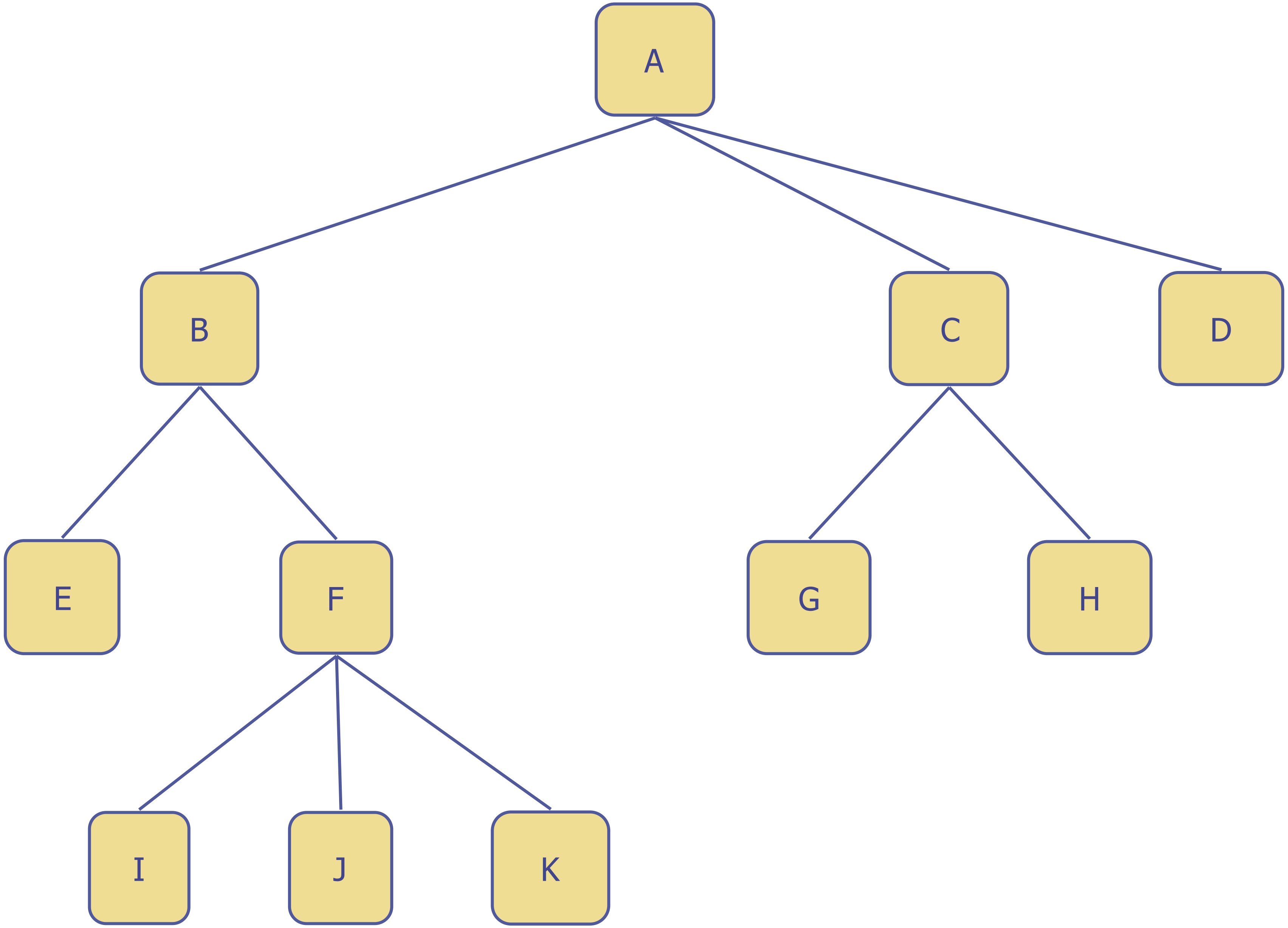
Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5

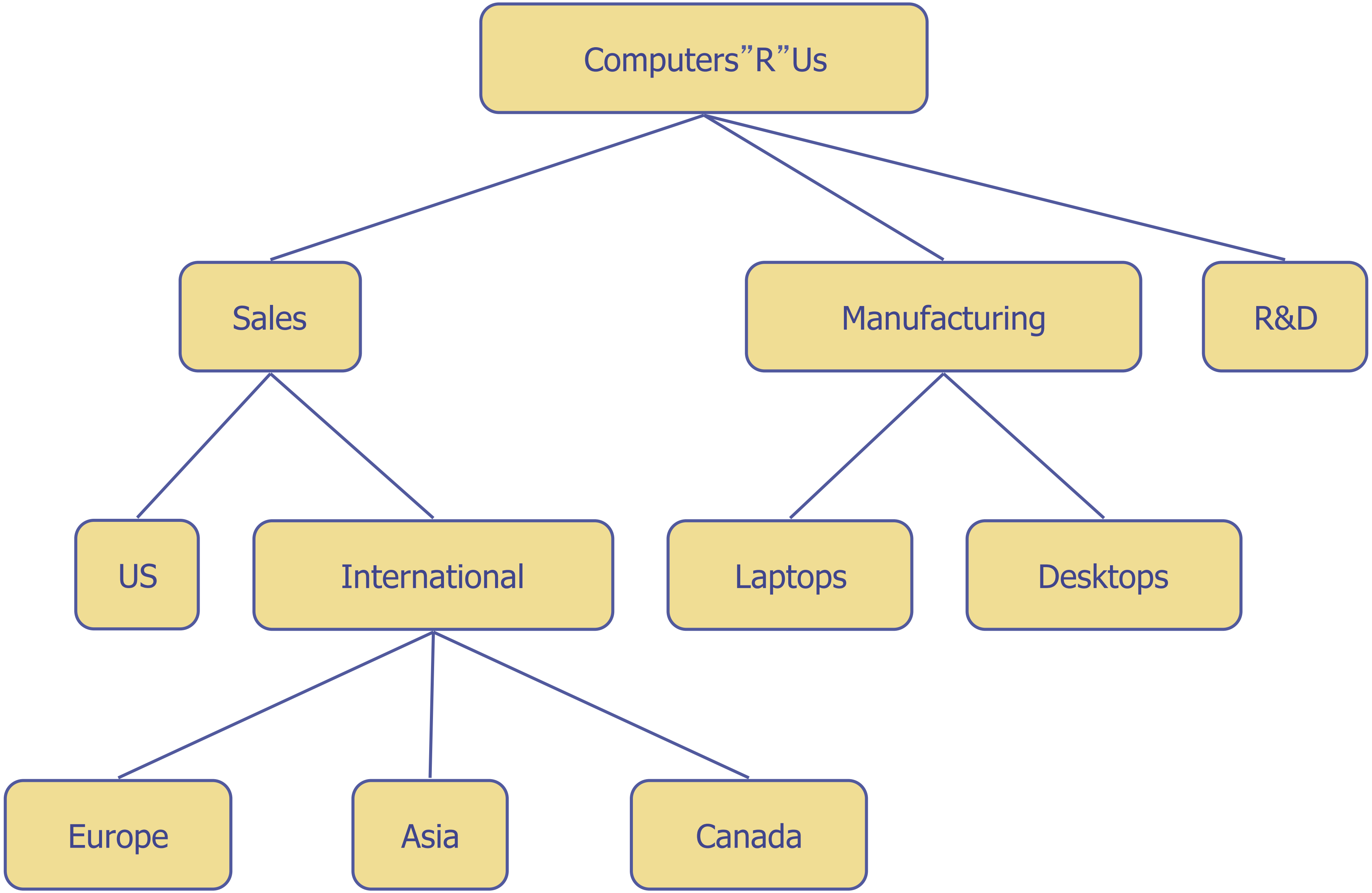

Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5



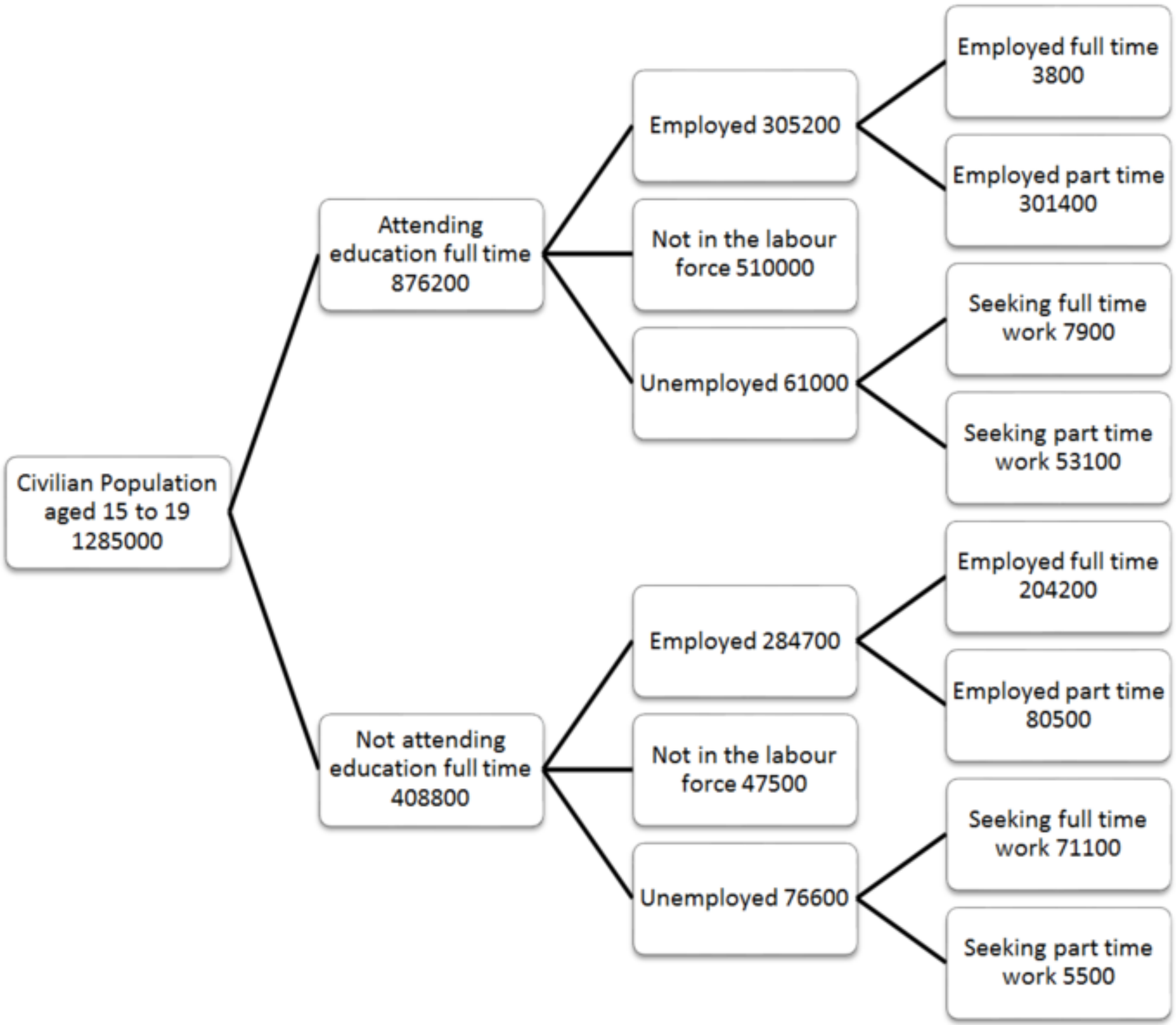
Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5

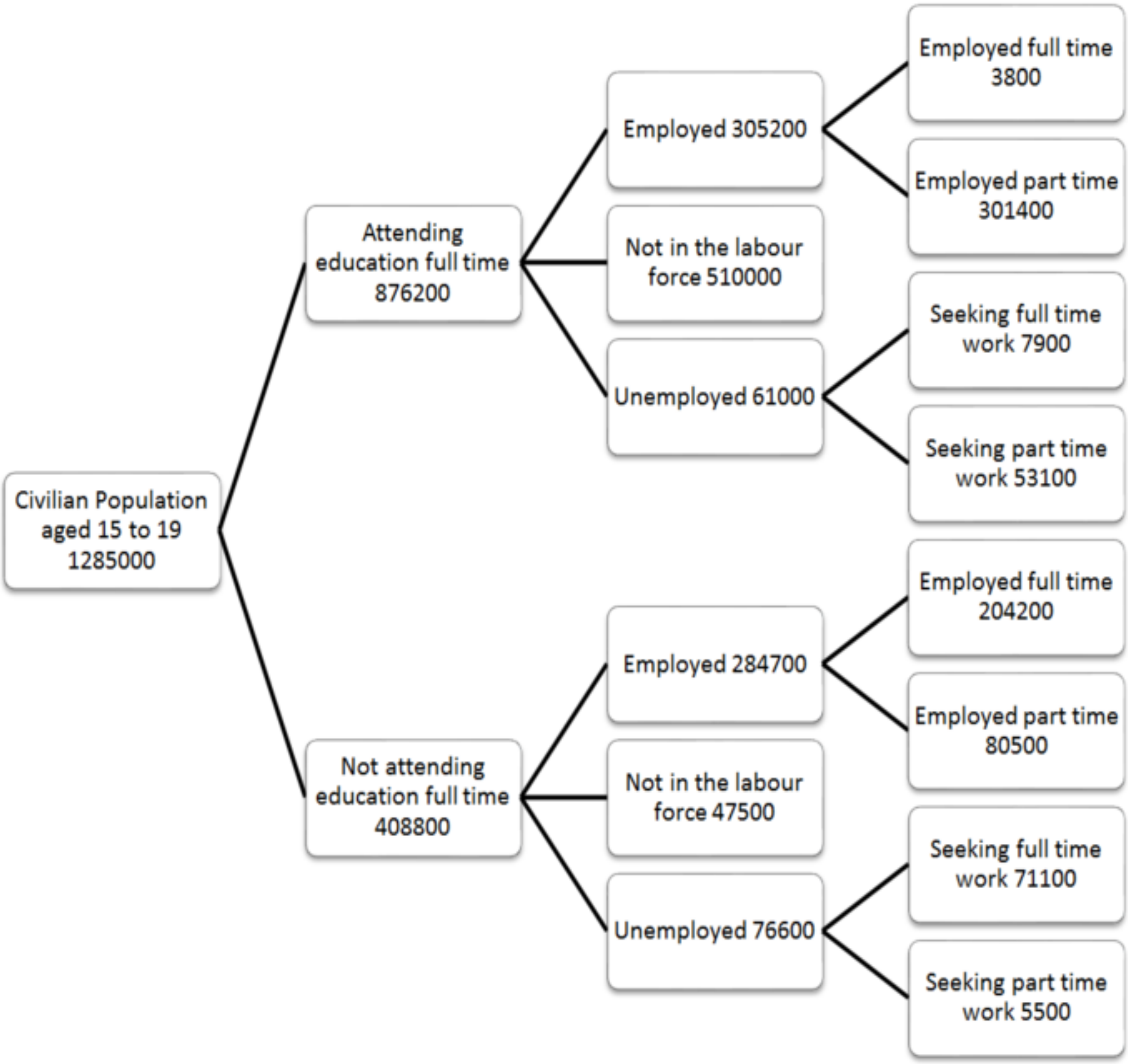

Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5
Node[]
barn
0 1 2 3 4 5

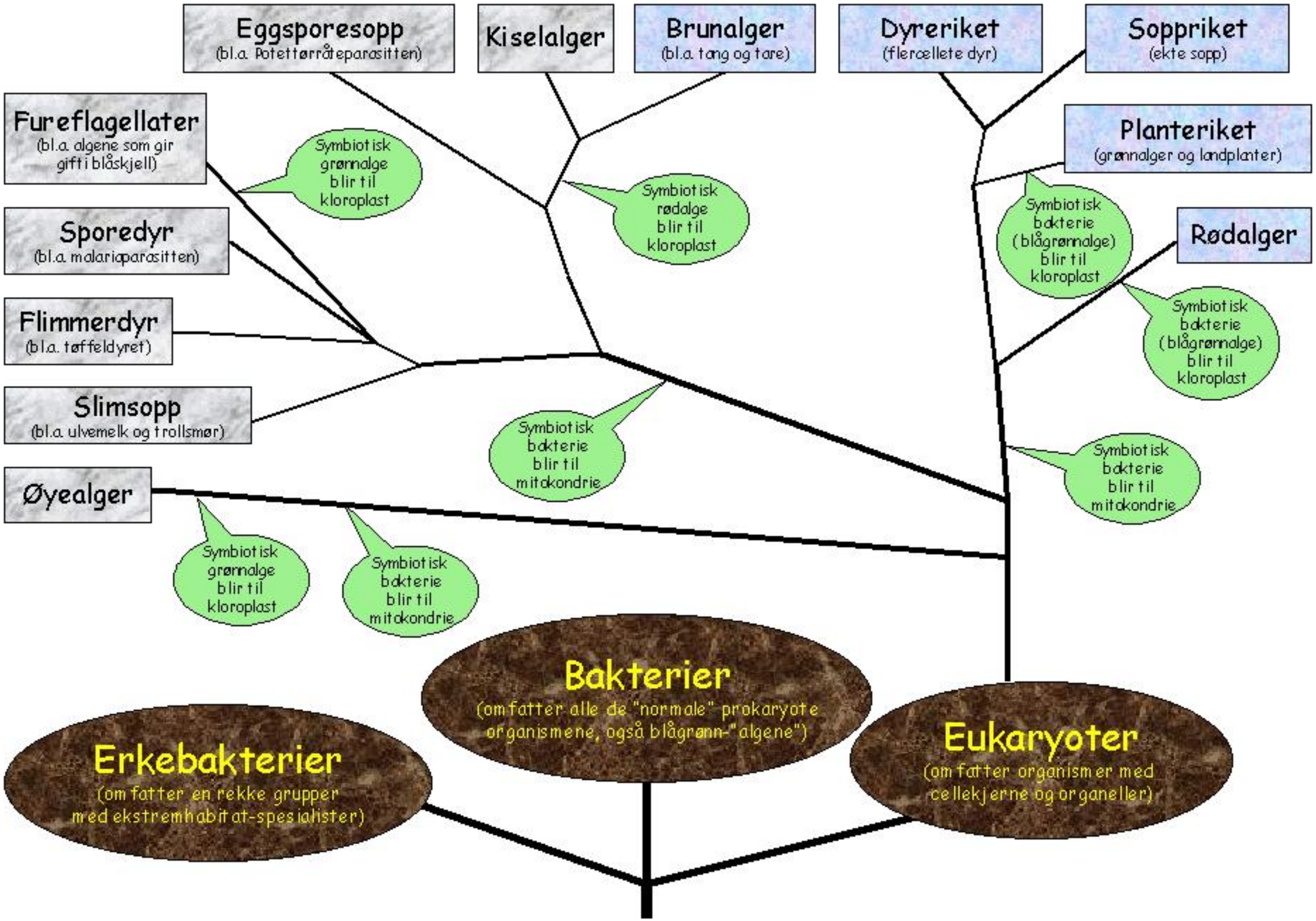

A
B
C
D
E
F
G
H
I
J
K



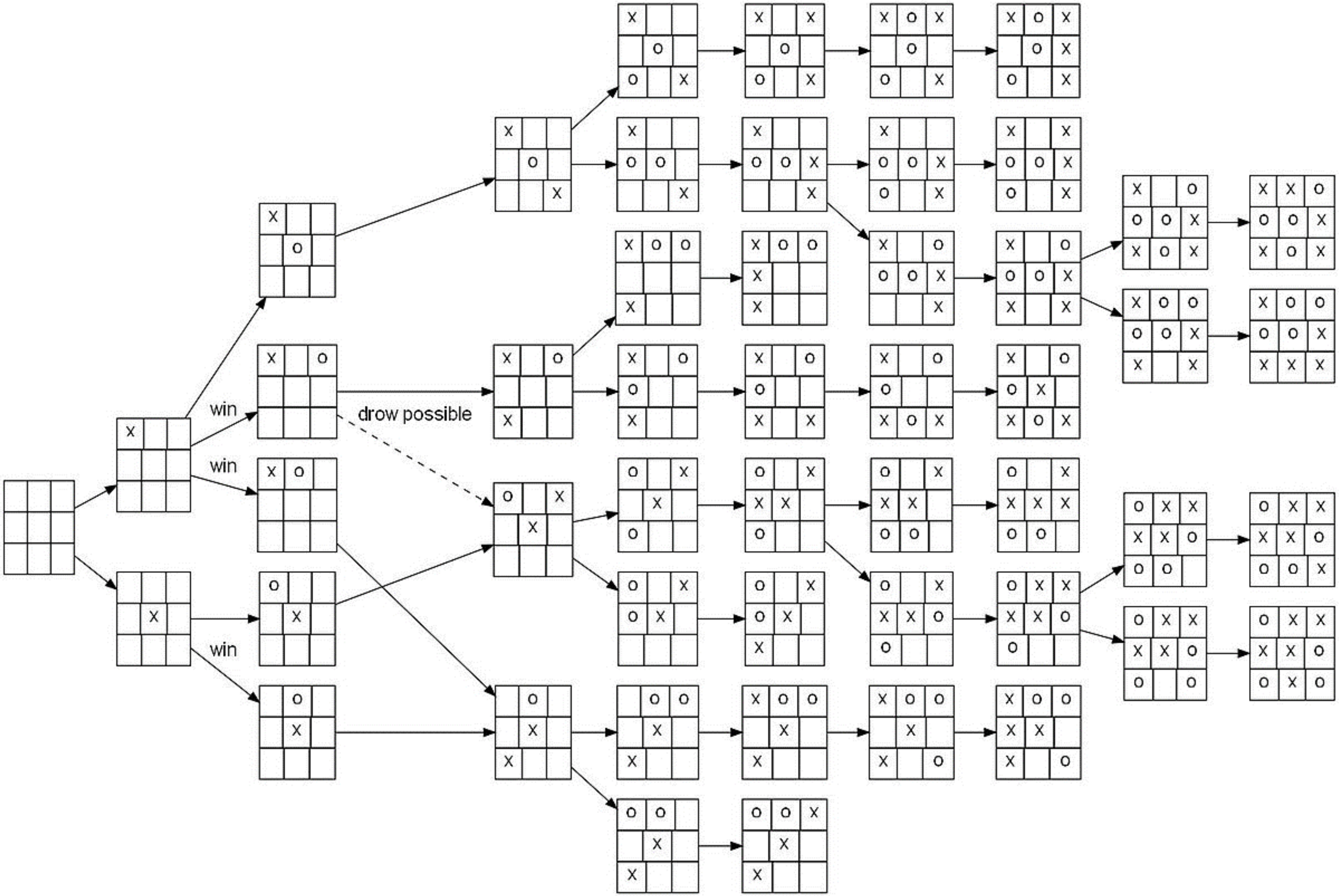
Computers"R"Us
Sales
Manufacturing
R&D
US
International
Laptops
Desktops
Europe
Asia
Canada


Civilian Population aged 15 to 19 1285000
Attending education full time 876200
Employed 305200
Employed full time 3800
Employed part time 301400
Not in the labour force 510000
Unemployed 61000
Seeking full time work 7900
Seeking part time work 53100
Not attending education full time 408800
Employed 284700
Employed full time 204200
Employed part time 80500
Not in the labour force 47500
Unemployed 76600
Seeking full time work 71100
Seeking part time work 5500

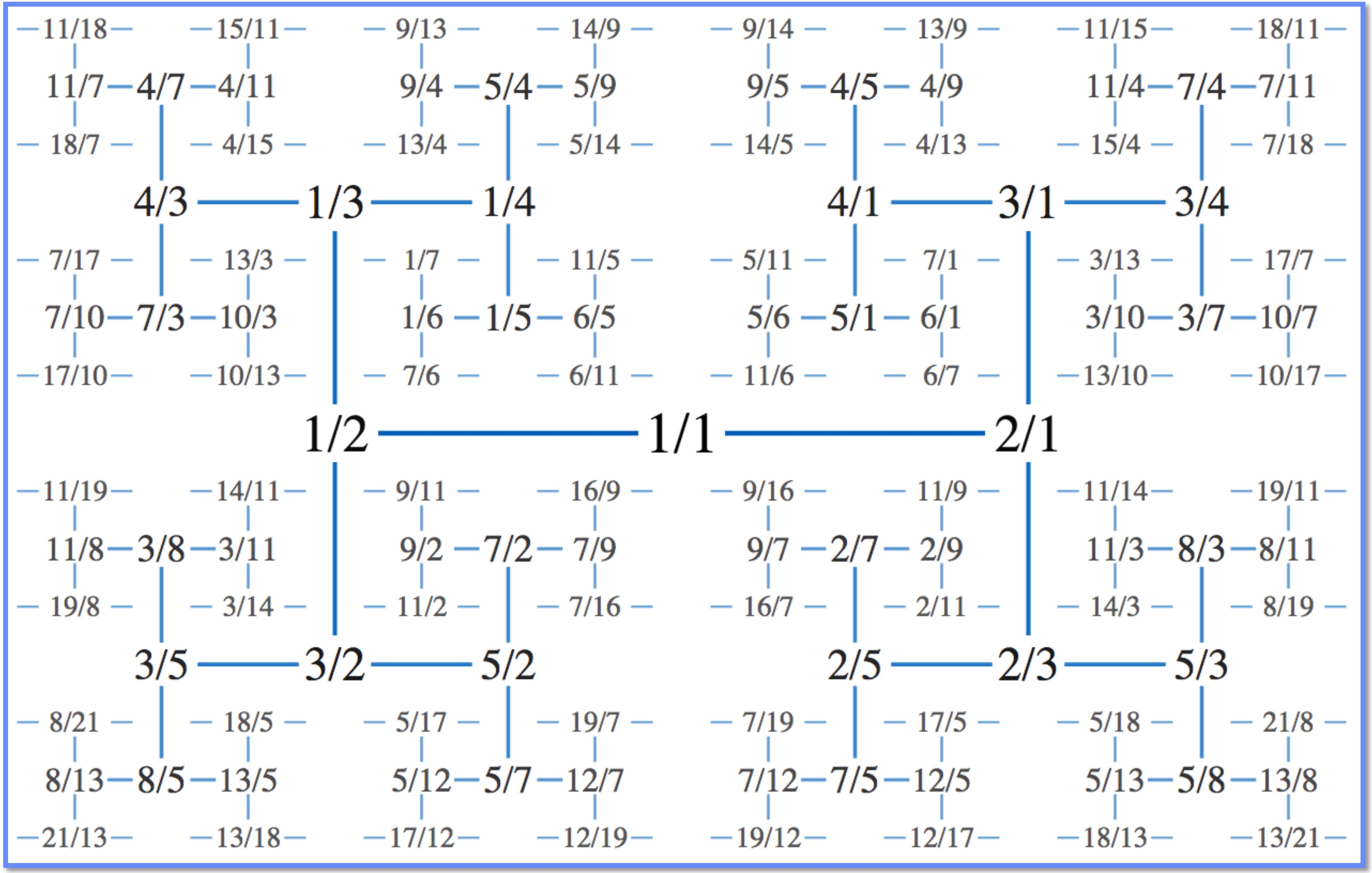

Civilian Population aged 15 to 19 1285000
Attending education full time 876200
Employed 305200
Employed full time 3800
Employed part time 301400
Not in the labour force 510000
Unemployed 61000
Seeking full time work 7900
Seeking part time work 53100
Not attending education full time 408800
Employed 284700
Employed full time 204200
Employed part time 80500
Not in the labour force 47500
Unemployed 76600
Seeking full time work 71100
Seeking part time work 5500

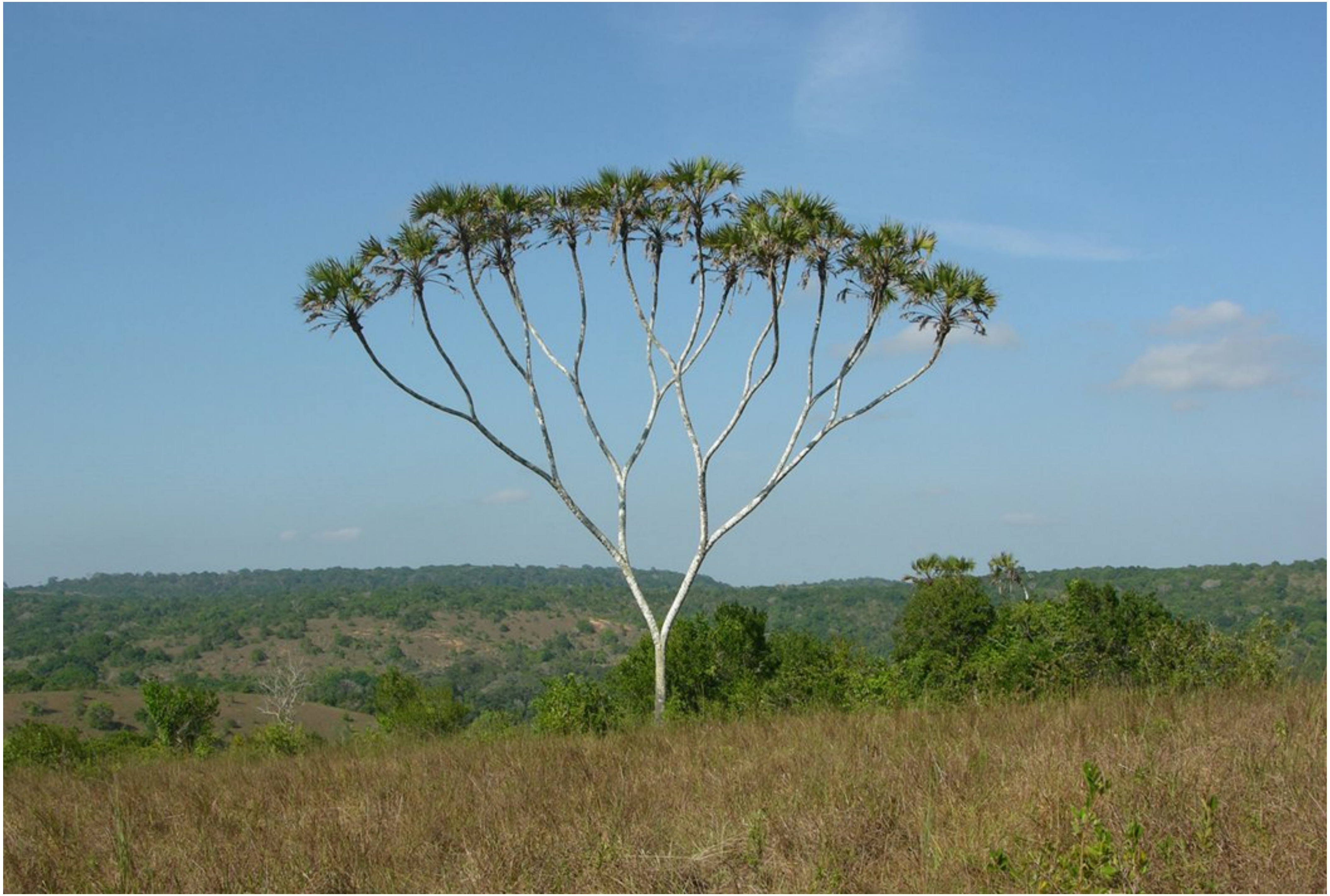


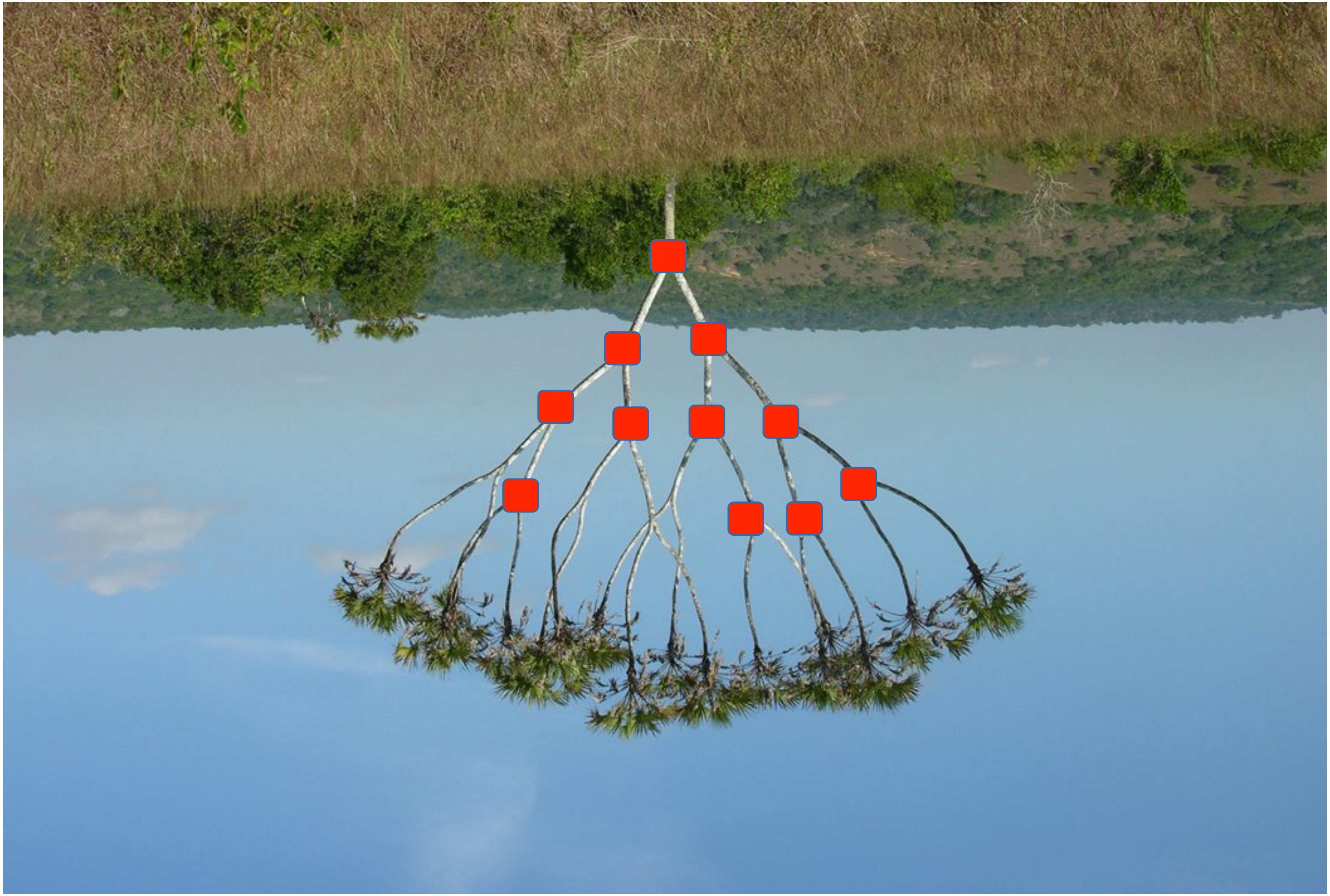
Eggsporesopp
(bl.a. Potettørråteparasitten)
Kiselalger
Brunalger
(bl.a. tang og tare)
Dyreriket
(flercellete dyr)
Soppriket
(ekte sopp)
Fureflagellater
(bl.a. algene som gir gift i blåskjell)
Planteriket
(grønnalger og landplanter)
Symbiotisk grønnalge blir til kloroplast
Symbiotisk rødalge blir til kloroplast
Sporedyr
(bl.a. malariaparasitten)
Symbiotisk bakterie (blågrønnalge) blir til kloroplast
Rødalger
Symbiotisk bakterie (blågrønnalge) blir til kloroplast
Flimmerdyr
(bl.a. tøffeldyret)
Slimsopp
(bl.a. ulvemelk og trollsmør)
Symbiotisk bakterie blir til mitokondrie
Symbiotisk bakterie blir til mitokondrie
Øyealger
Symbiotisk grønnalge blir til kloroplast
Symbiotisk bakterie blir til mitokondrie
Bakterier
(omfatter alle de "normale" prokaryote organismene, også blågrønn-"algene")
Erkebakterier
(omfatter en rekke grupper med ekstremhabitat-spesialister)
Eukaryoter
(omfatter organismer med cellekjerne og organeller)

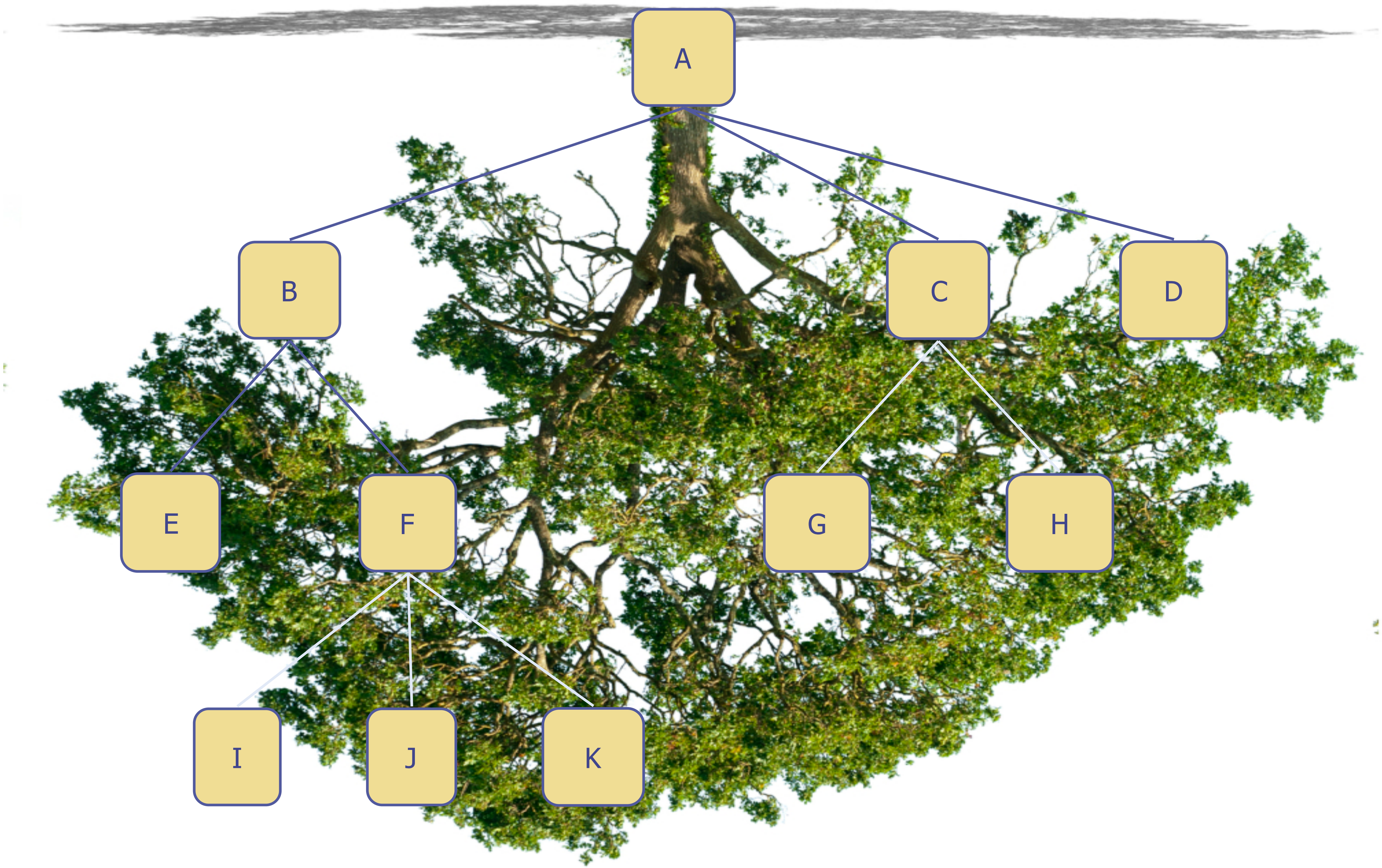


win
win
drow possible
win

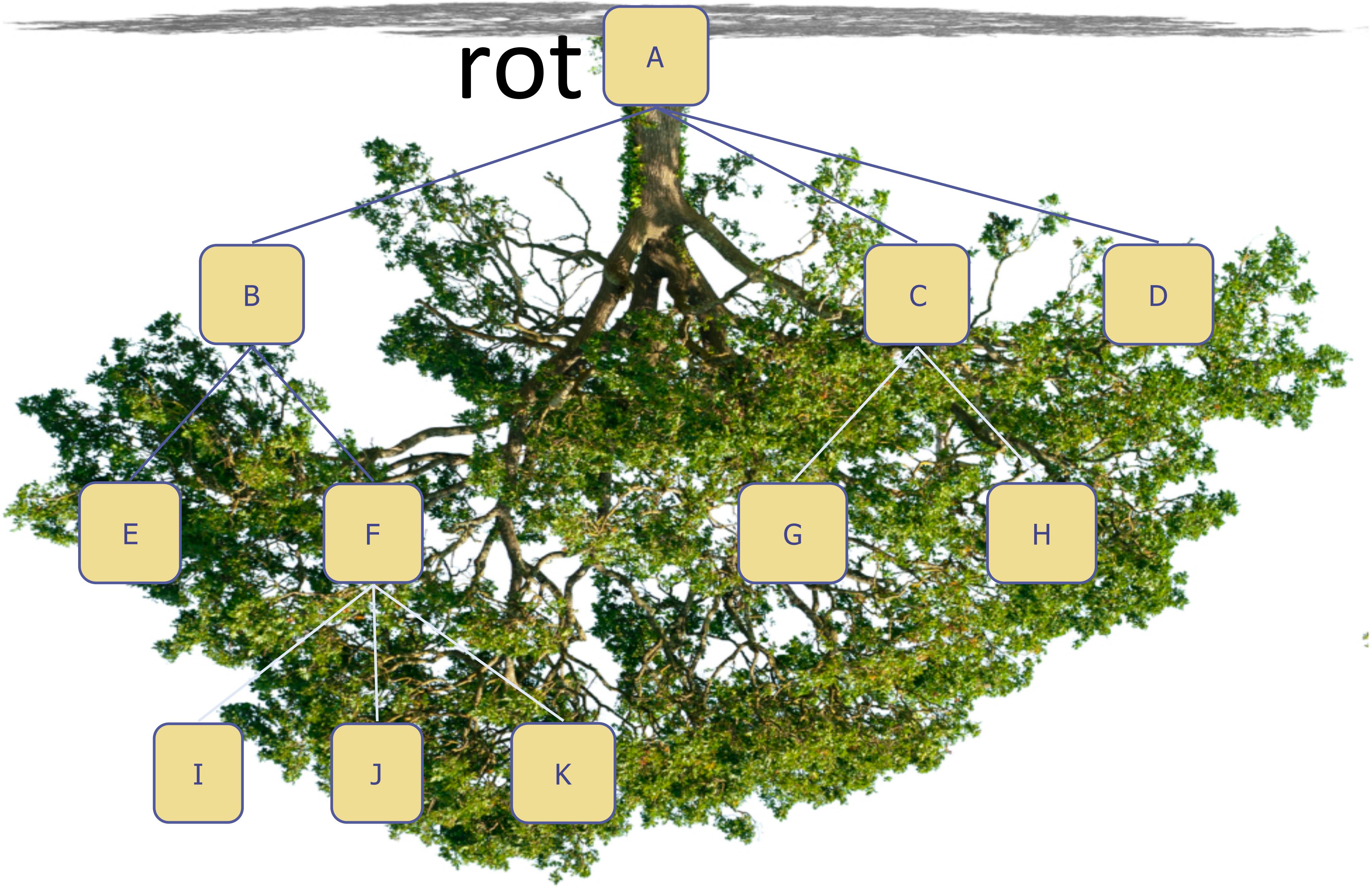

11/18 15/11 9/13 14/9 9/14 13/9 11/15 18/11
11/7 4/7 4/11 9/4 5/4 5/9 9/5 4/5 4/9 11/4 7/4 7/11
18/7 4/15 13/4 5/14 14/5 4/13 15/4 7/18
4/3 1/3 1/4 4/1 3/1 3/4
7/17 13/3 1/7 11/5 5/11 7/1 3/13 17/7
7/10 7/3 10/3 1/6 1/5 6/5 5/6 5/1 6/1 3/10 3/7 10/7
17/10 10/13 7/6 6/11 11/6 6/7 13/10 10/17
1/2 1/1 2/1
11/19 14/11 9/11 16/9 9/16 11/9 11/14 19/11
11/8 3/8 3/11 9/2 7/2 7/9 9/7 2/7 2/9 11/3 8/3 8/11
19/8 3/14 11/2 7/16 16/7 2/11 14/3 8/19
3/5 3/2 5/2 2/5 2/3 5/3
8/21 18/5 5/17 19/7 7/19 17/5 5/18 21/8
8/13 8/5 13/5 5/12 5/7 12/7 7/12 7/5 12/5 5/13 5/8 13/8
21/13 13/18 17/12 12/19 19/12 12/17 18/13 13/21

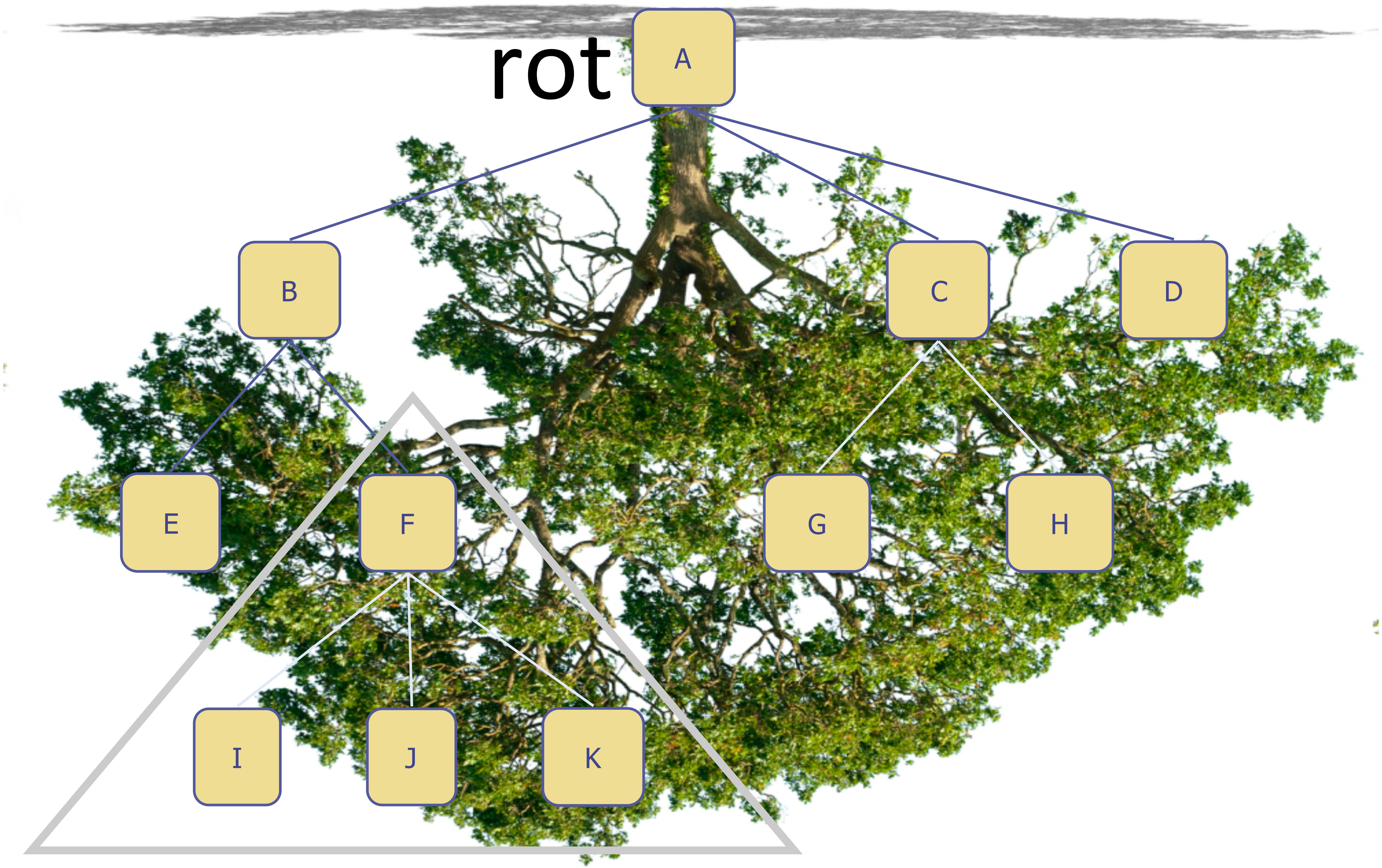


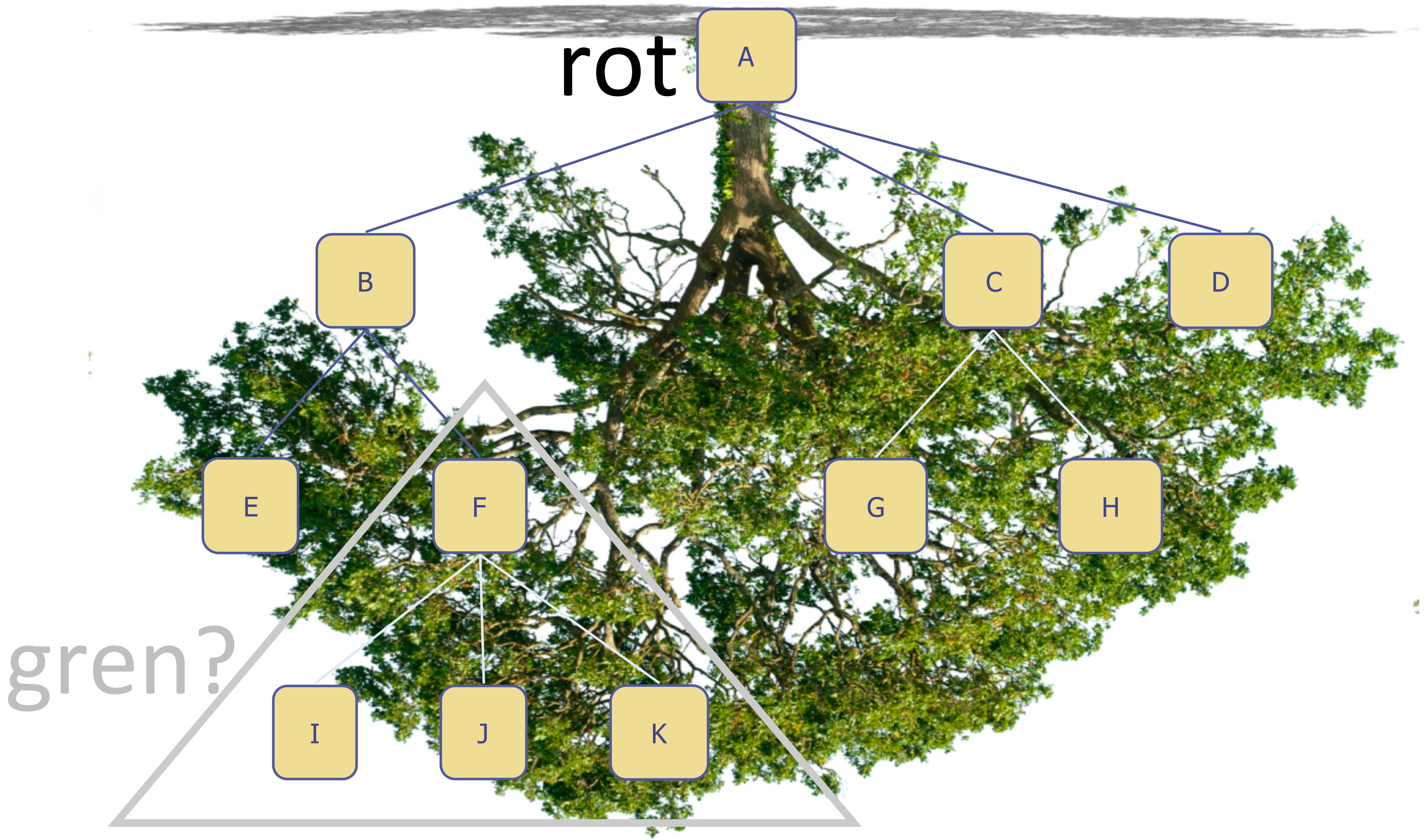


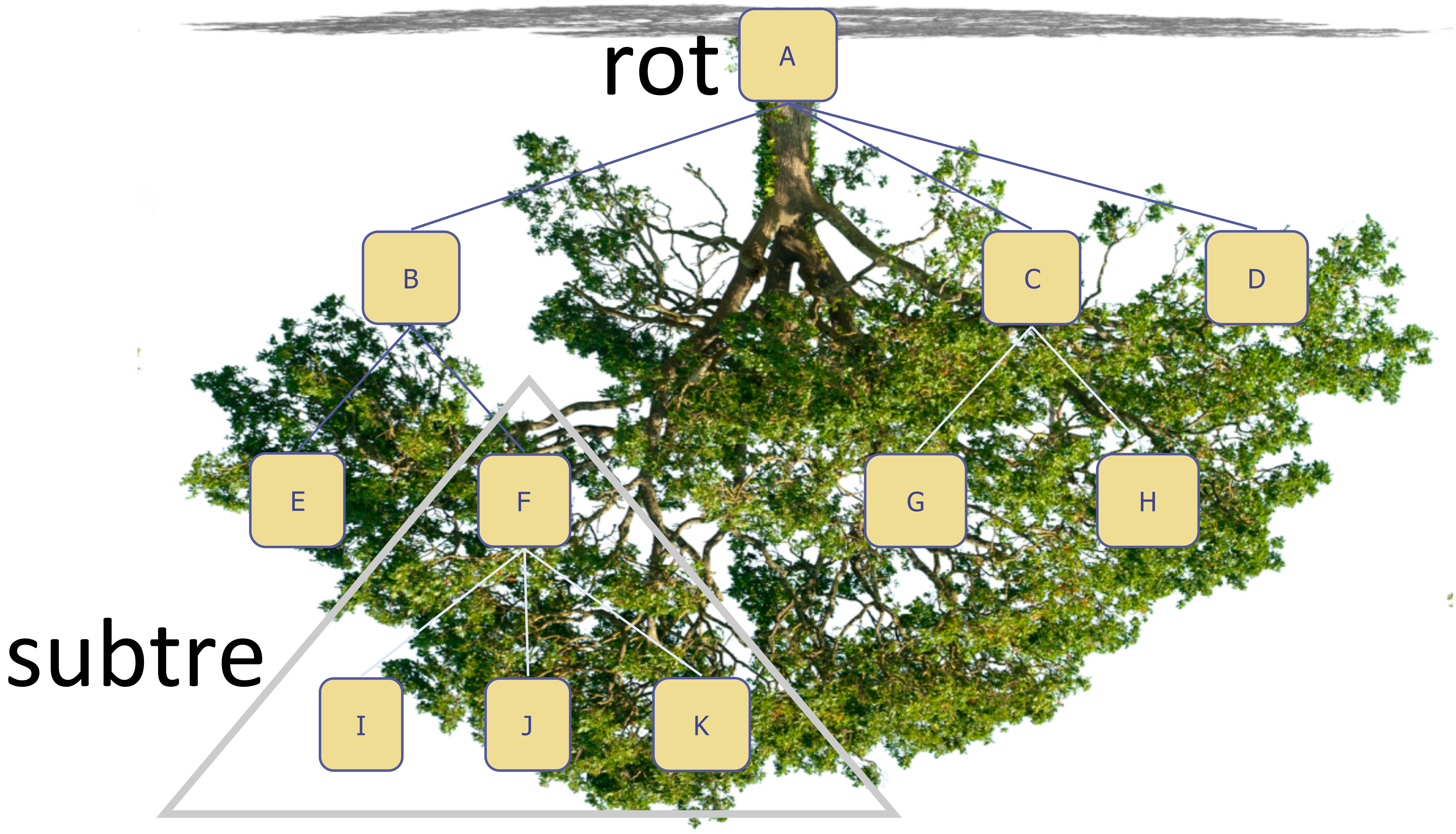

A
B
C
D
E
F
G
H
I
J
K

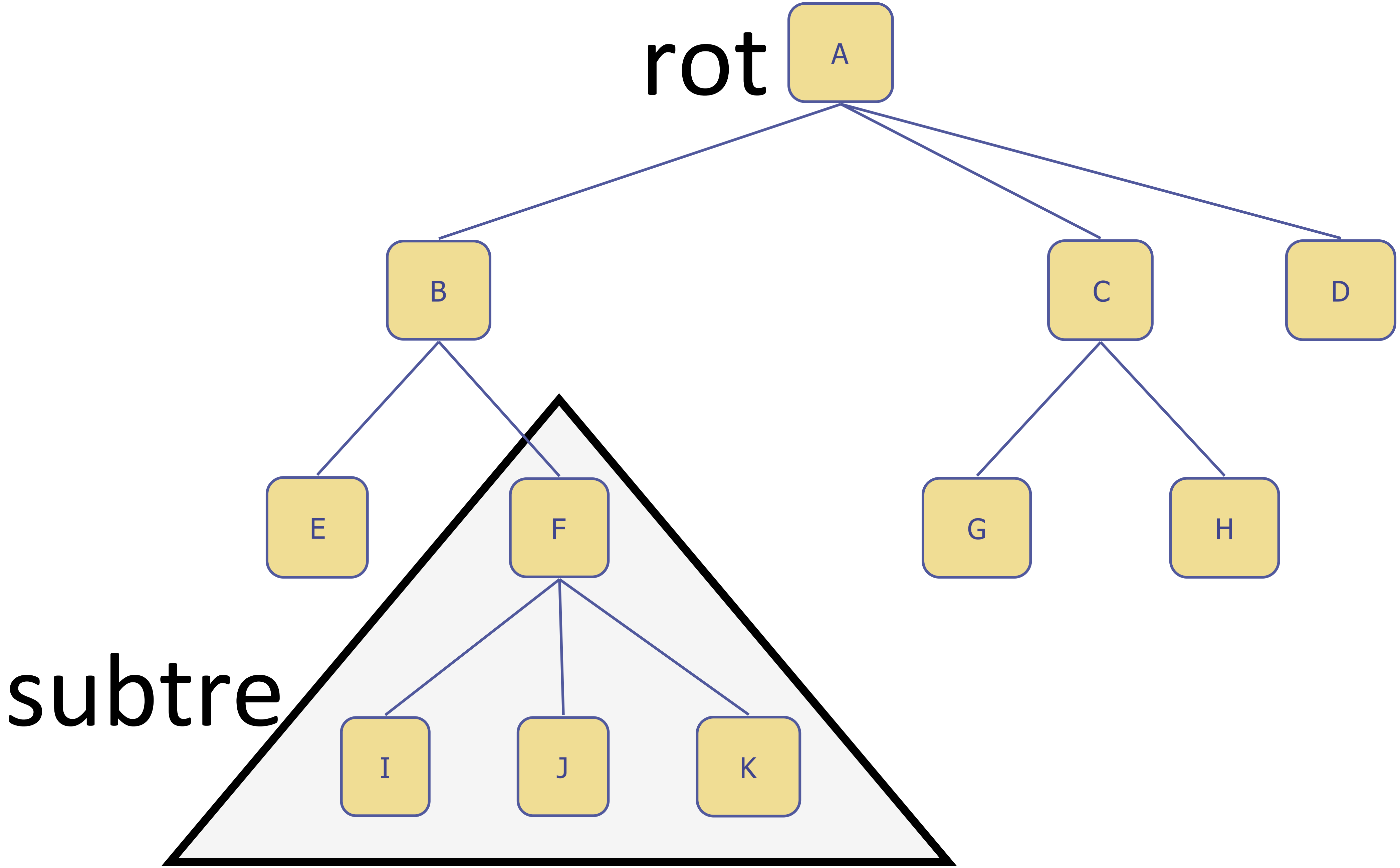
rot

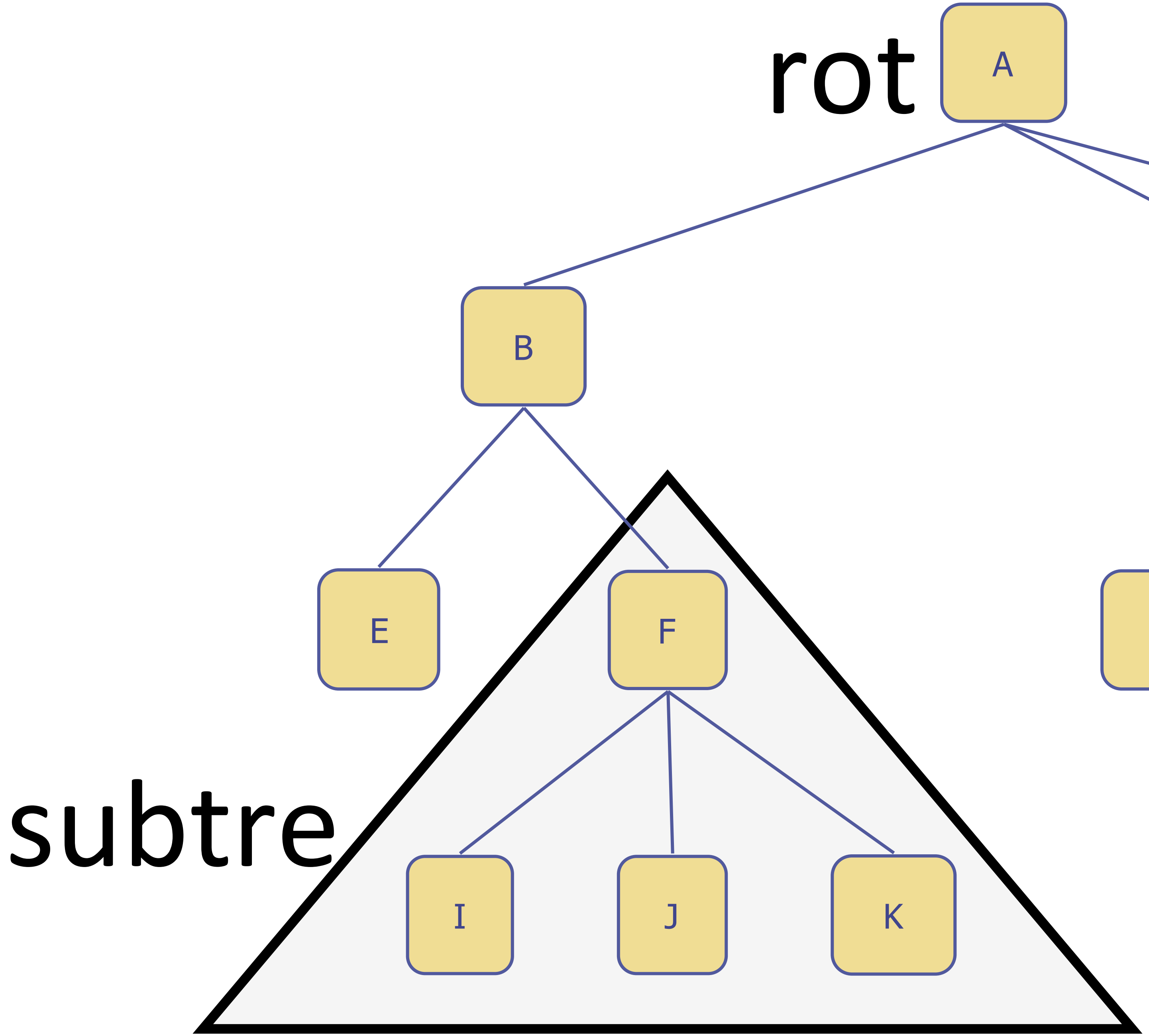



rot
A
B
C
D
E
F
G
H
I
J
K


rot
A
B
C
D
E
F
G
H
gren?
I
J
K

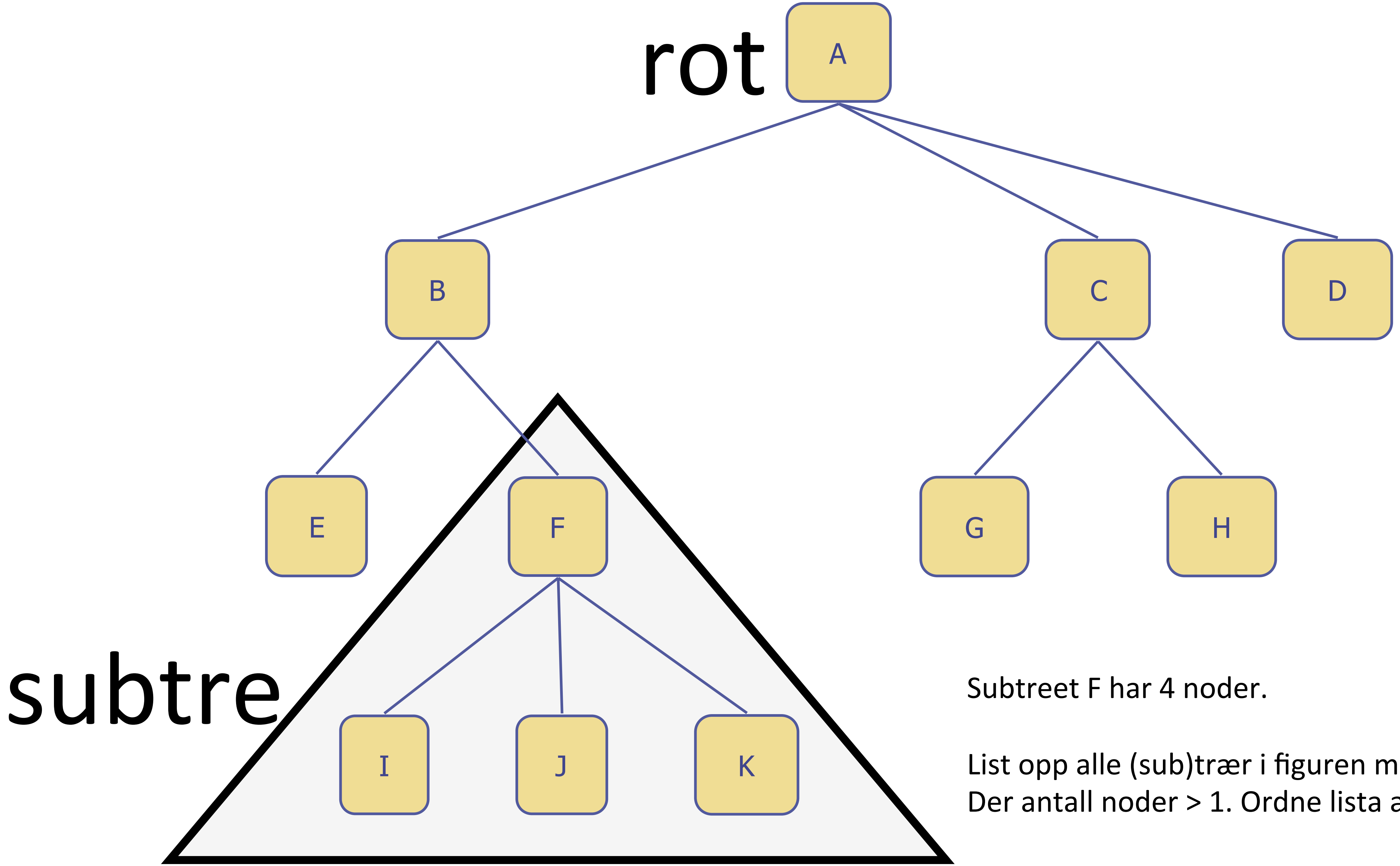

rot
A
B
C
D
E
F
G
H
I
J
K
subtre


rot
A
B
C
D
E
F
G
H
subtre
I
J
K



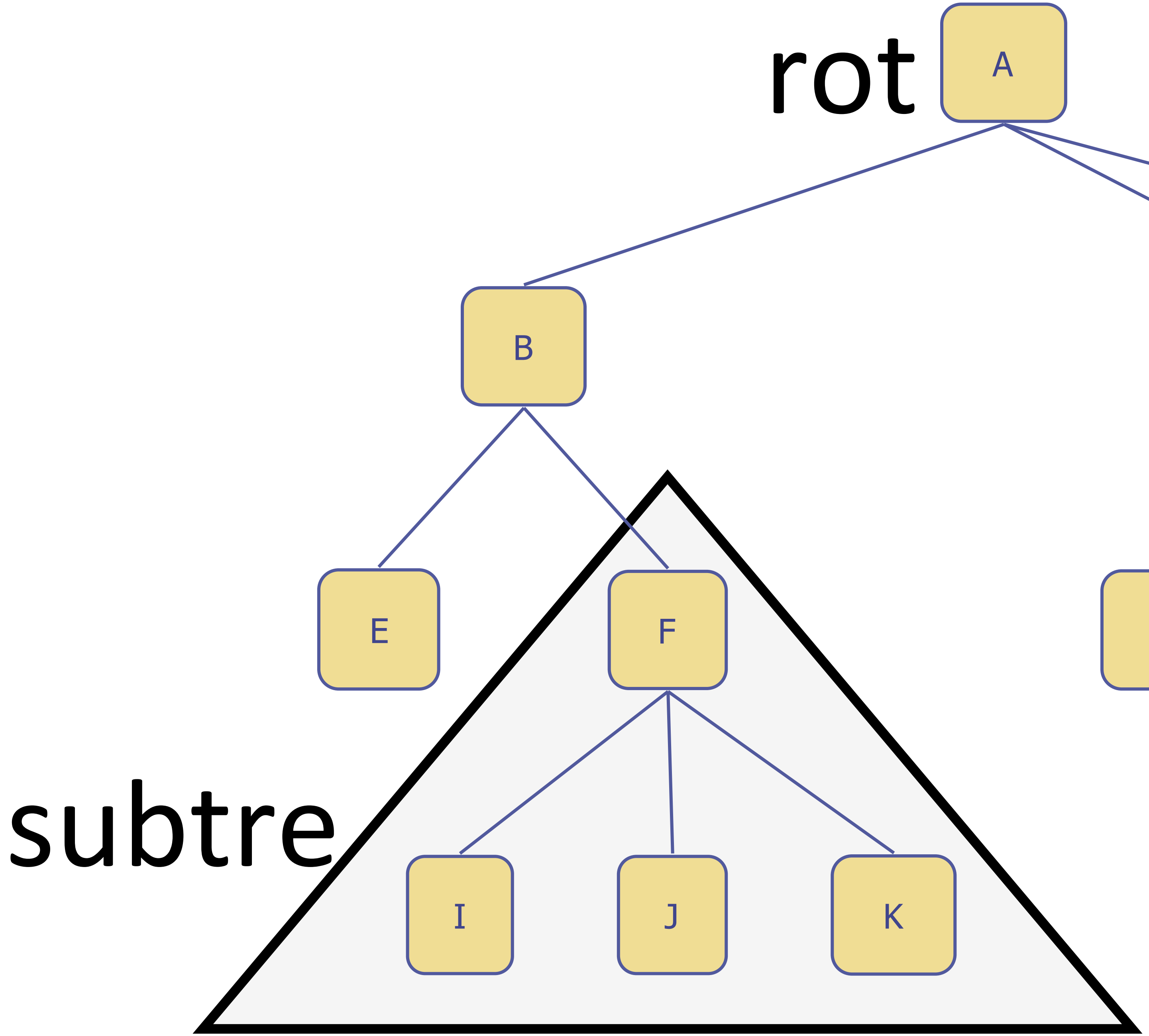
Oppgave:

Hvilken bokstav er noden som er rot i subtreet merket med?



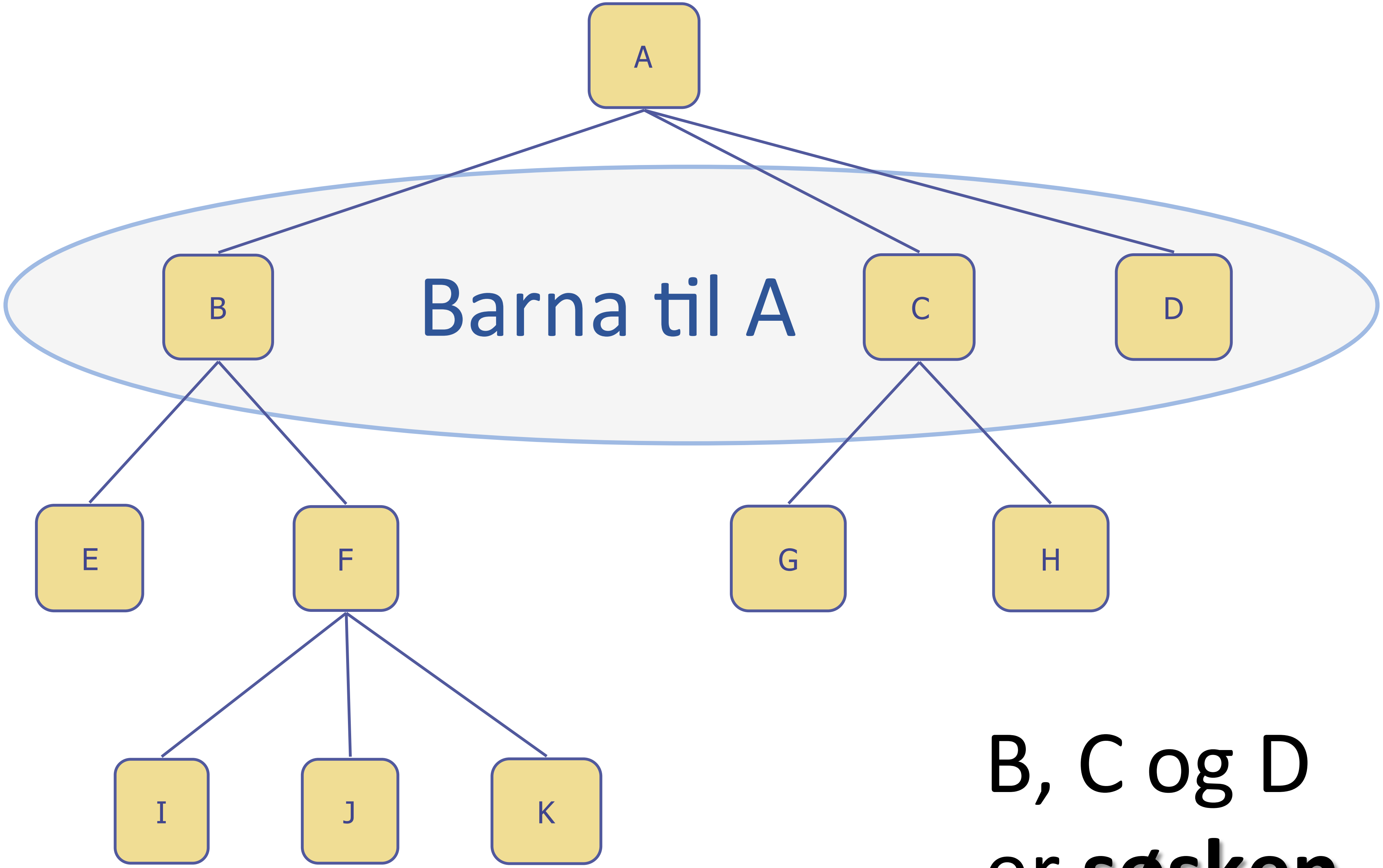
Subtreet F har 4 noder.

List opp alle (sub)trær i figuren med antall noder,
Der antall noder > 1. Ordne lista alfabetisk.

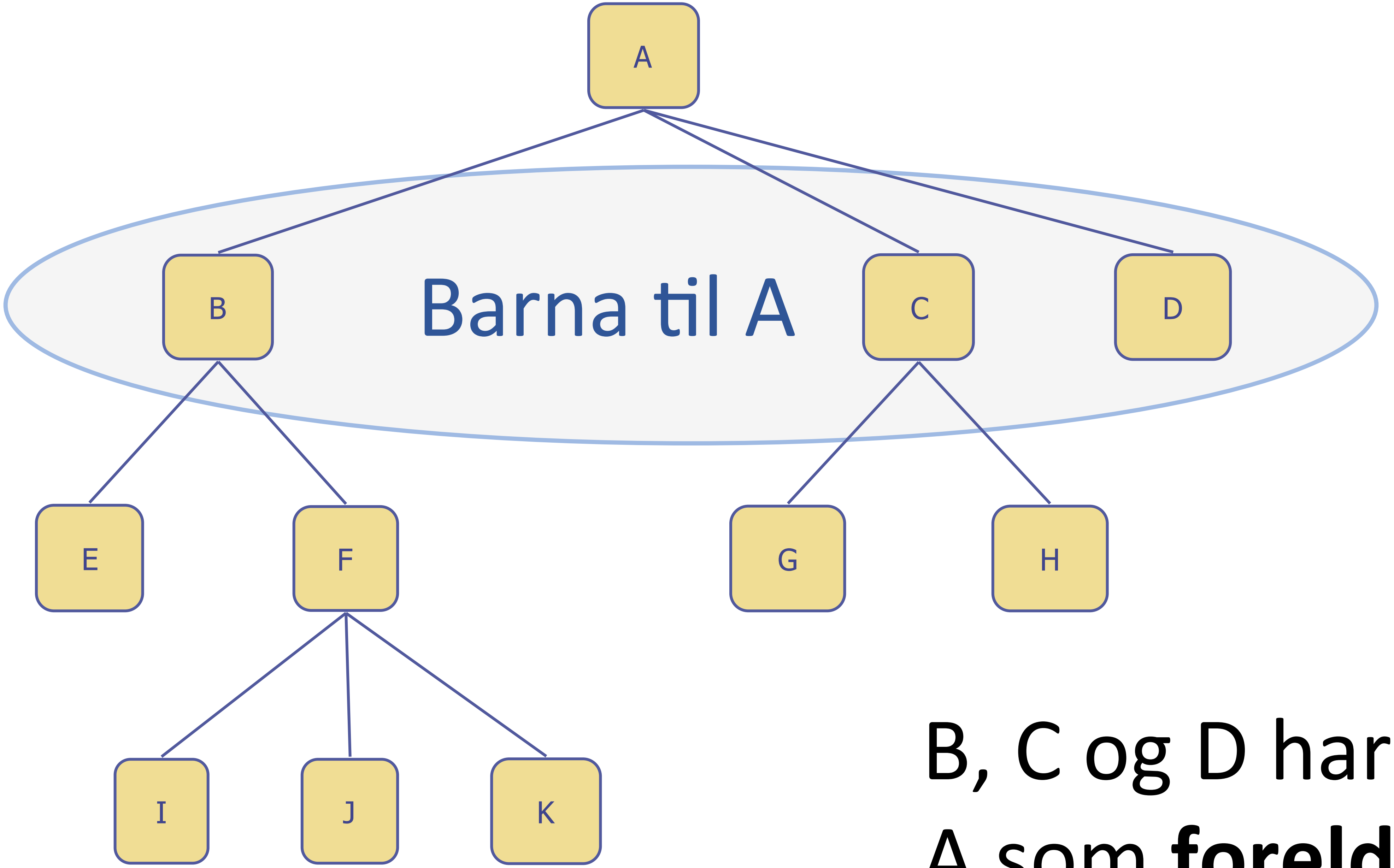


løsning:

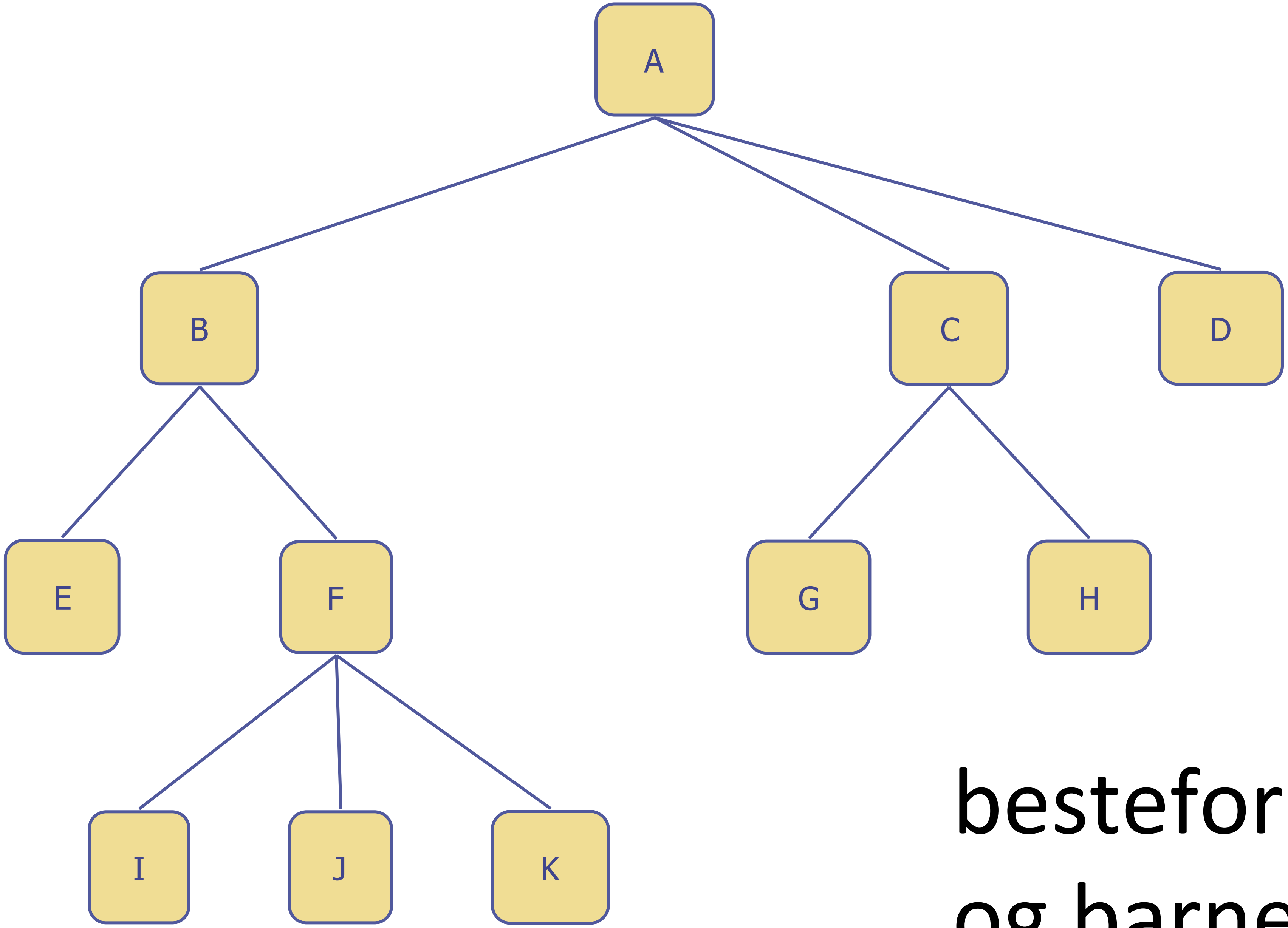
A 11
B 6
C 3
F 4



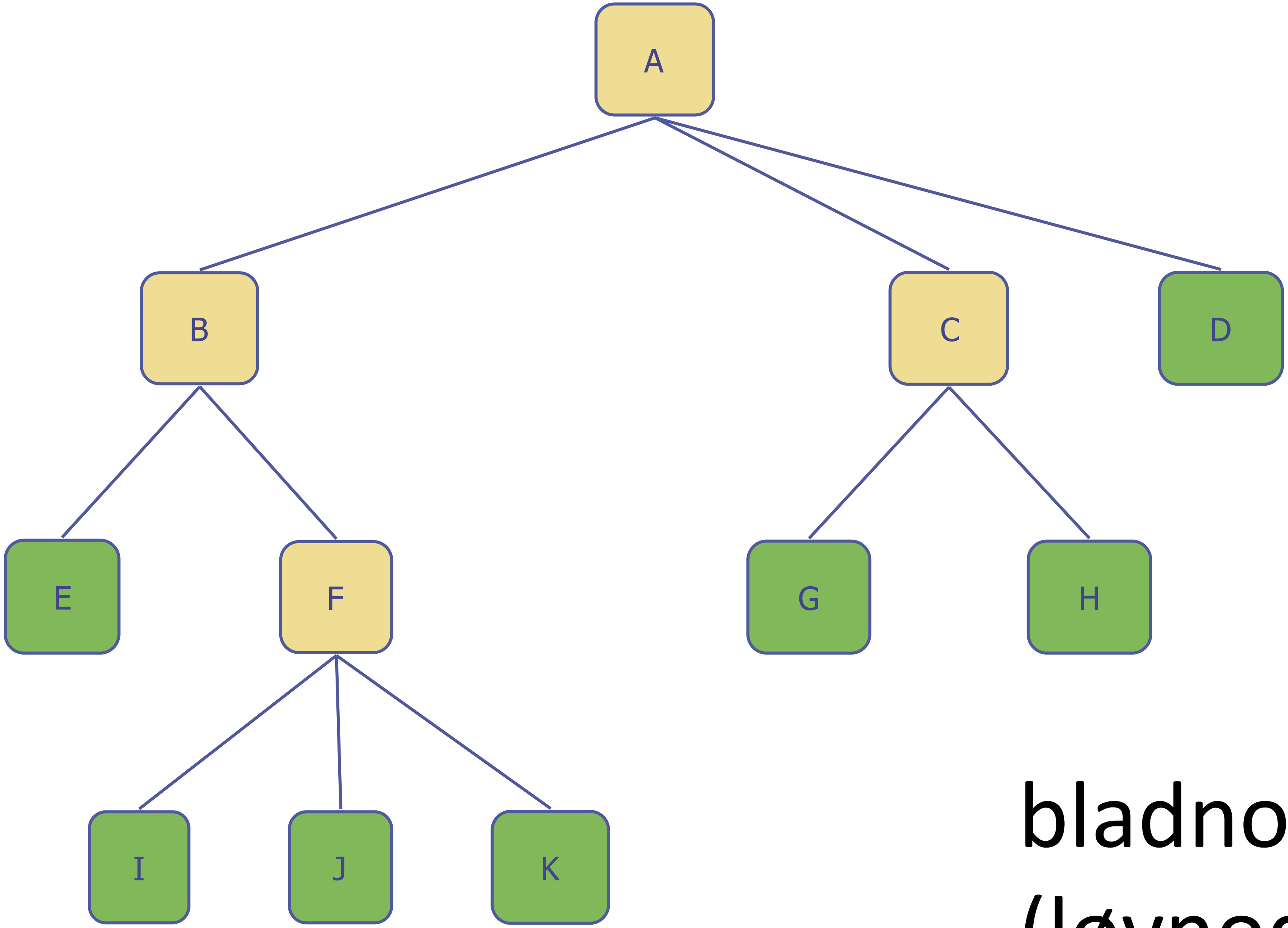
B, C og D
er **søsken**



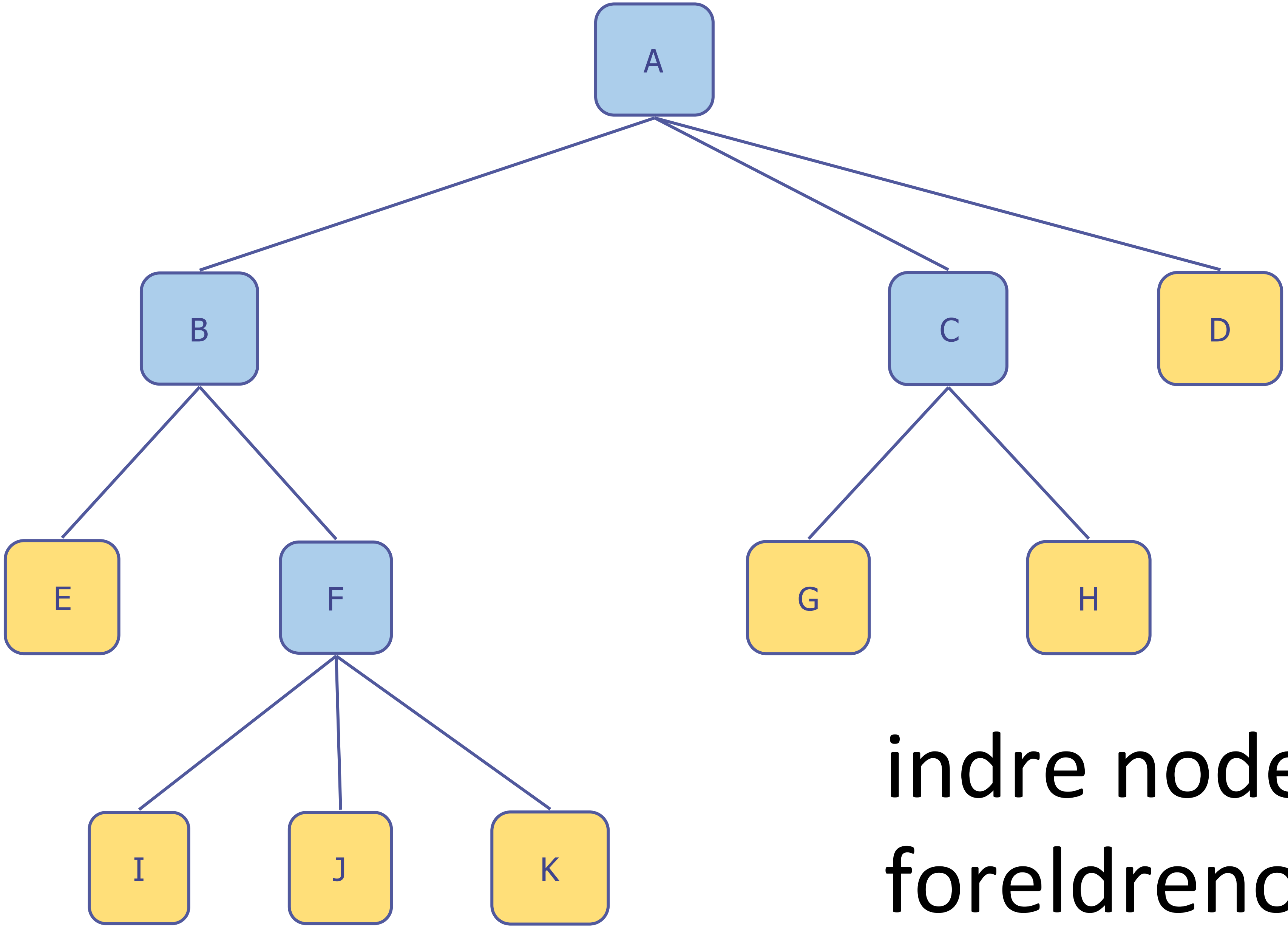
B, C og D har
A som **forelder**



besteforeldre
og barnebarn?



bladnoder
(løvnoder)



indre noder

foreldrenoder

ikke bladnoder

Treterminologi

rot

indre node

bladnode

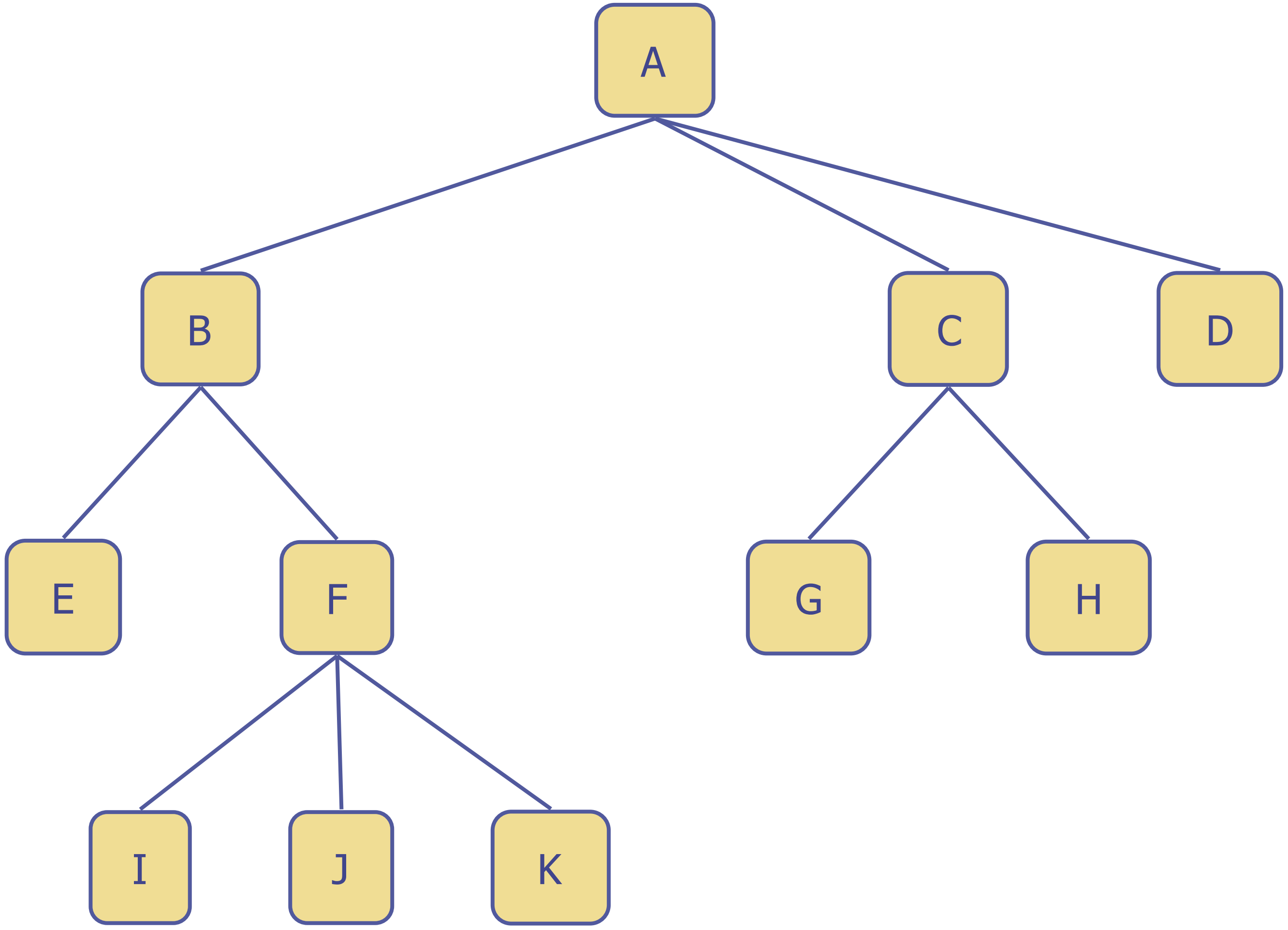
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Treterminologi

rot

indre node

bladnode

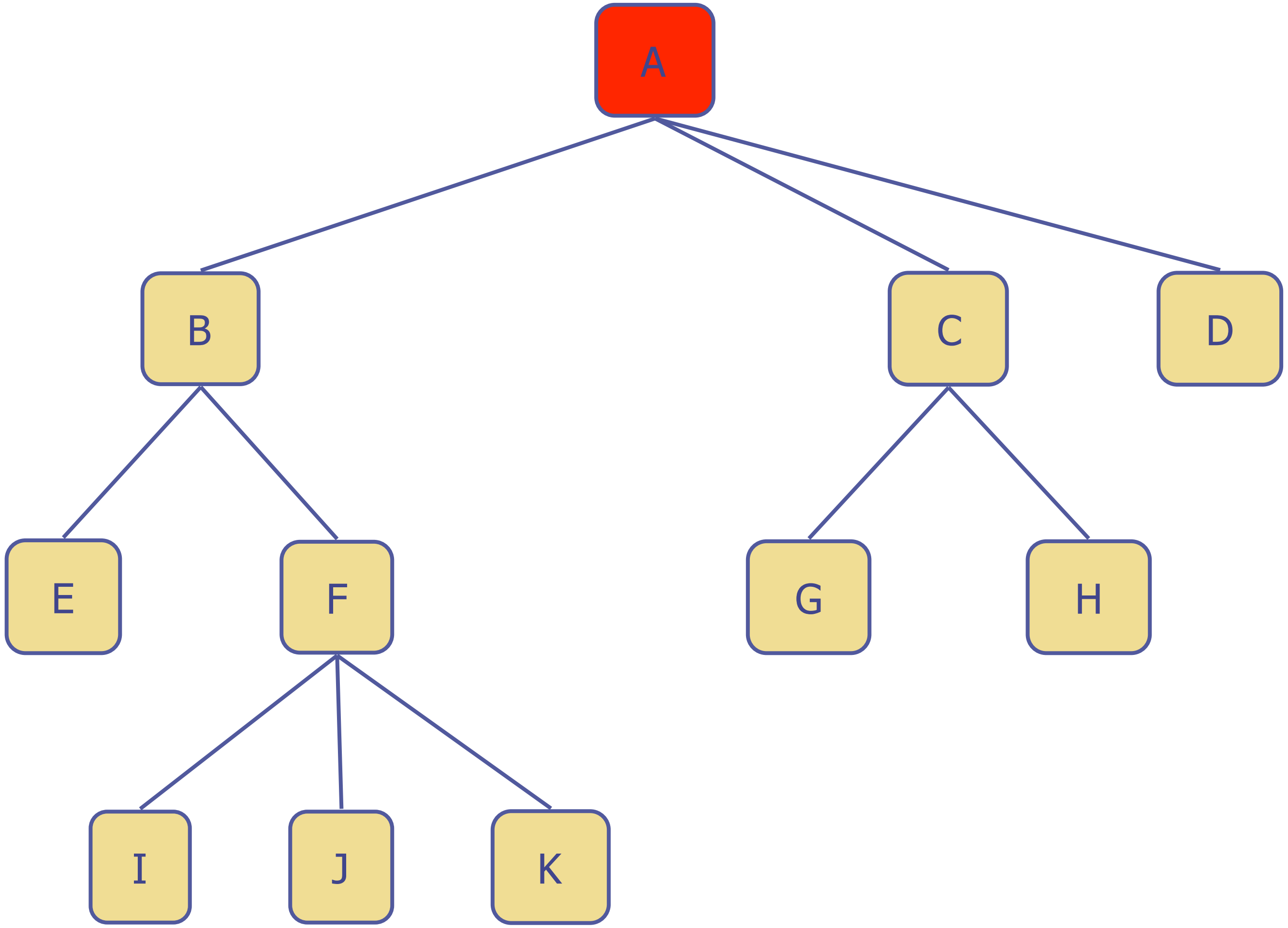
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Treterminologi

rot

indre noder

bladnode

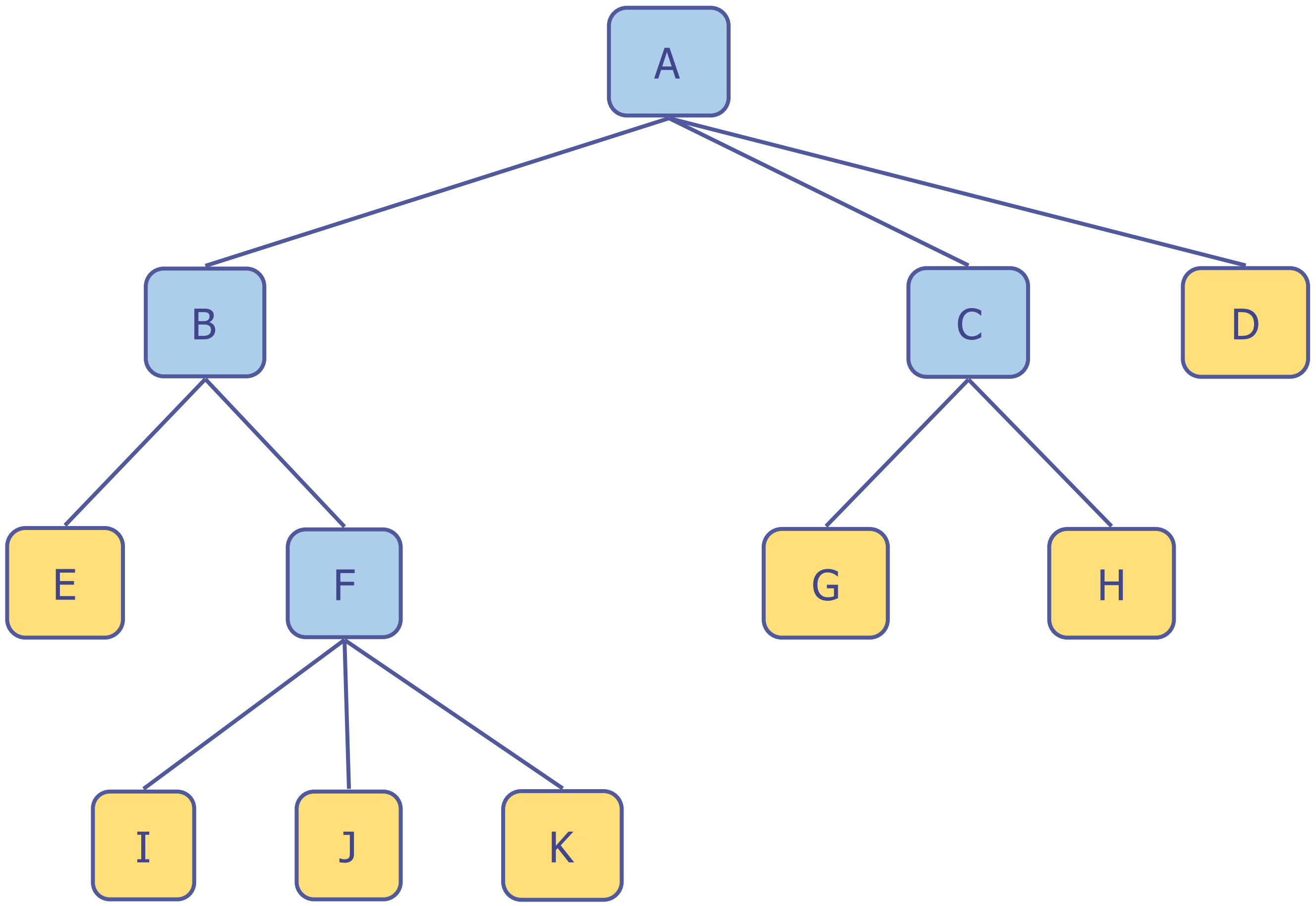
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Treterminologi

rot

indre node

bladnoder

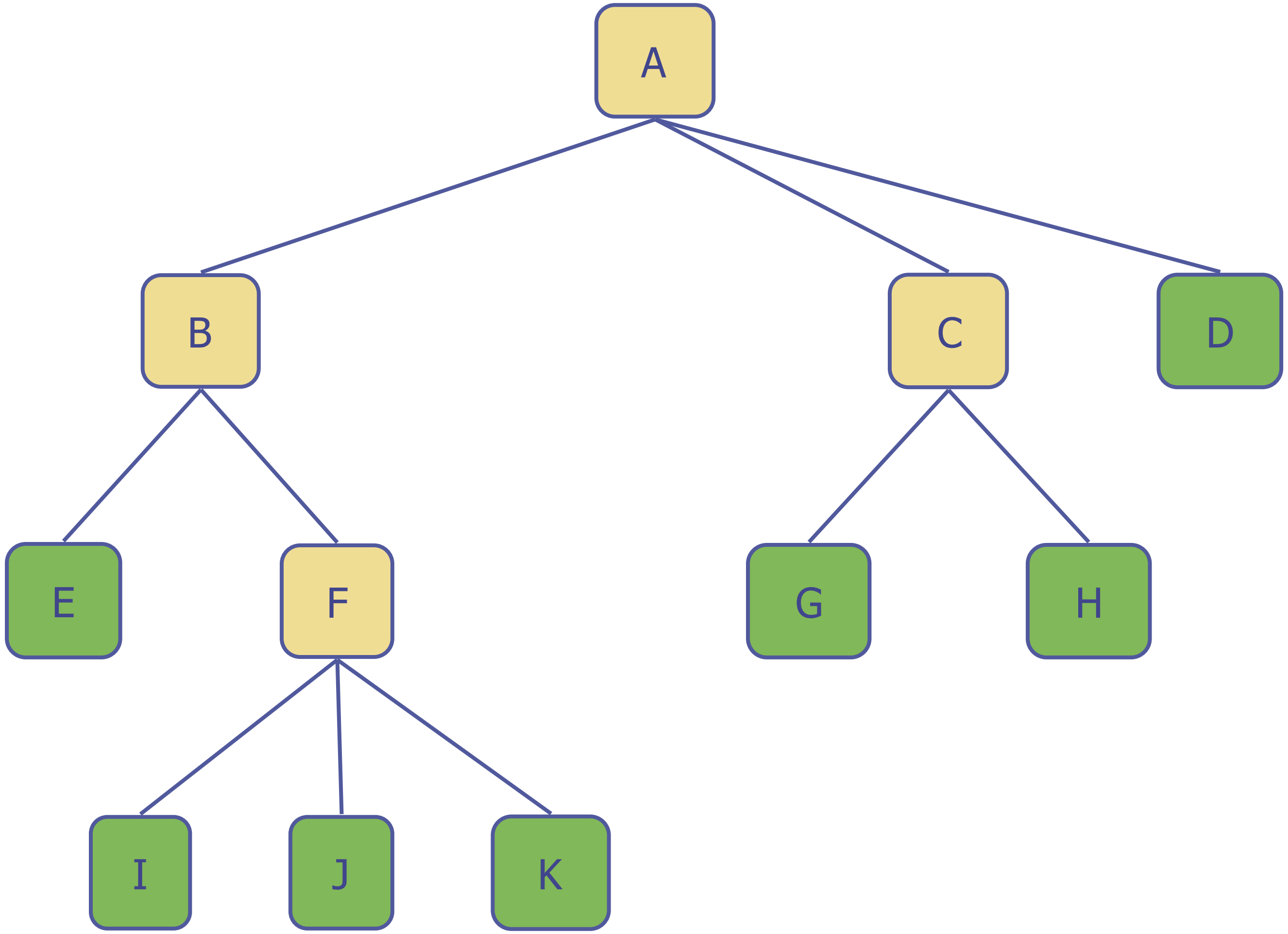
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Treterminologi

rot

indre node

bladnode

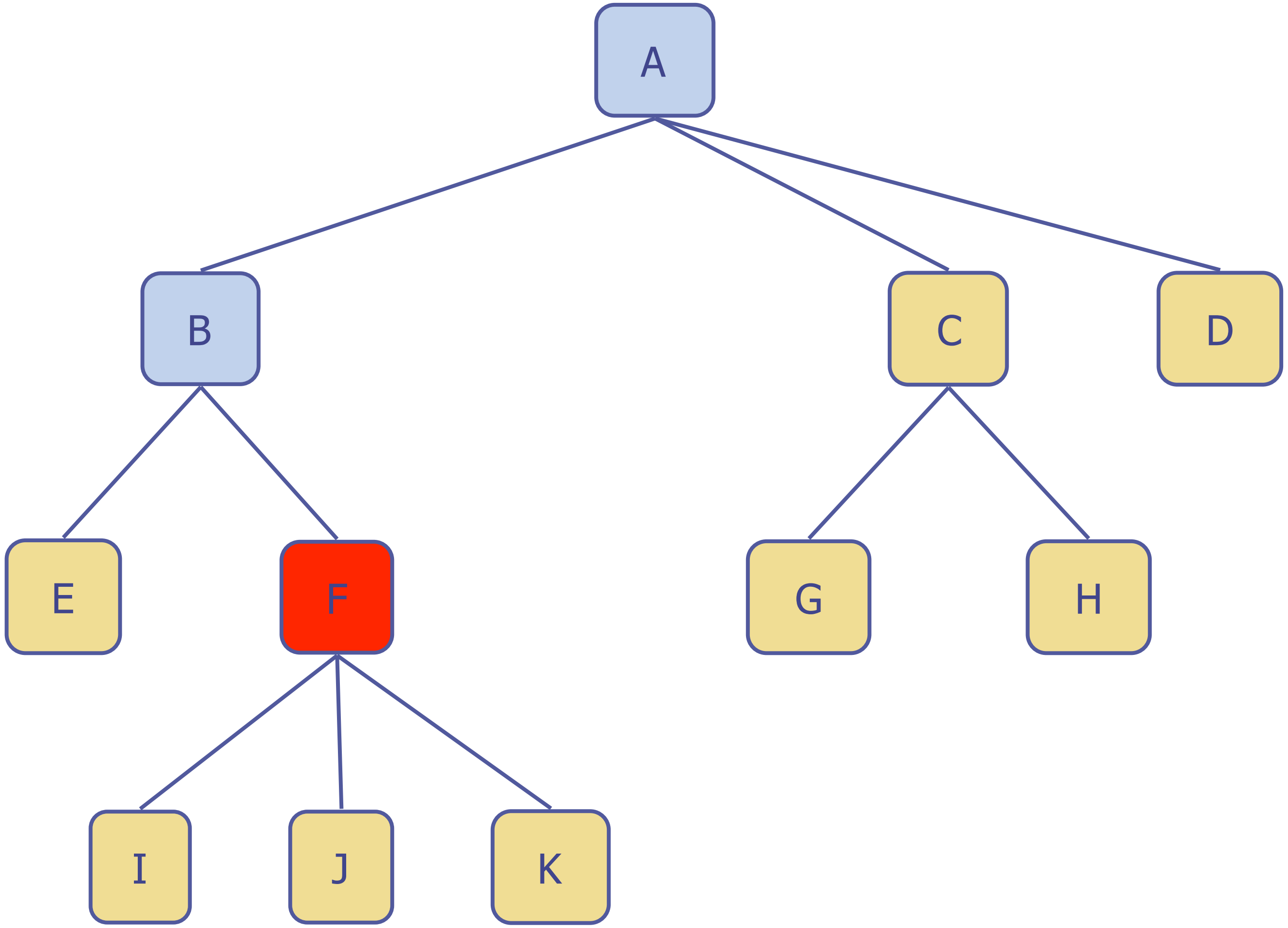
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Treterminologi

rot

indre node

bladnode

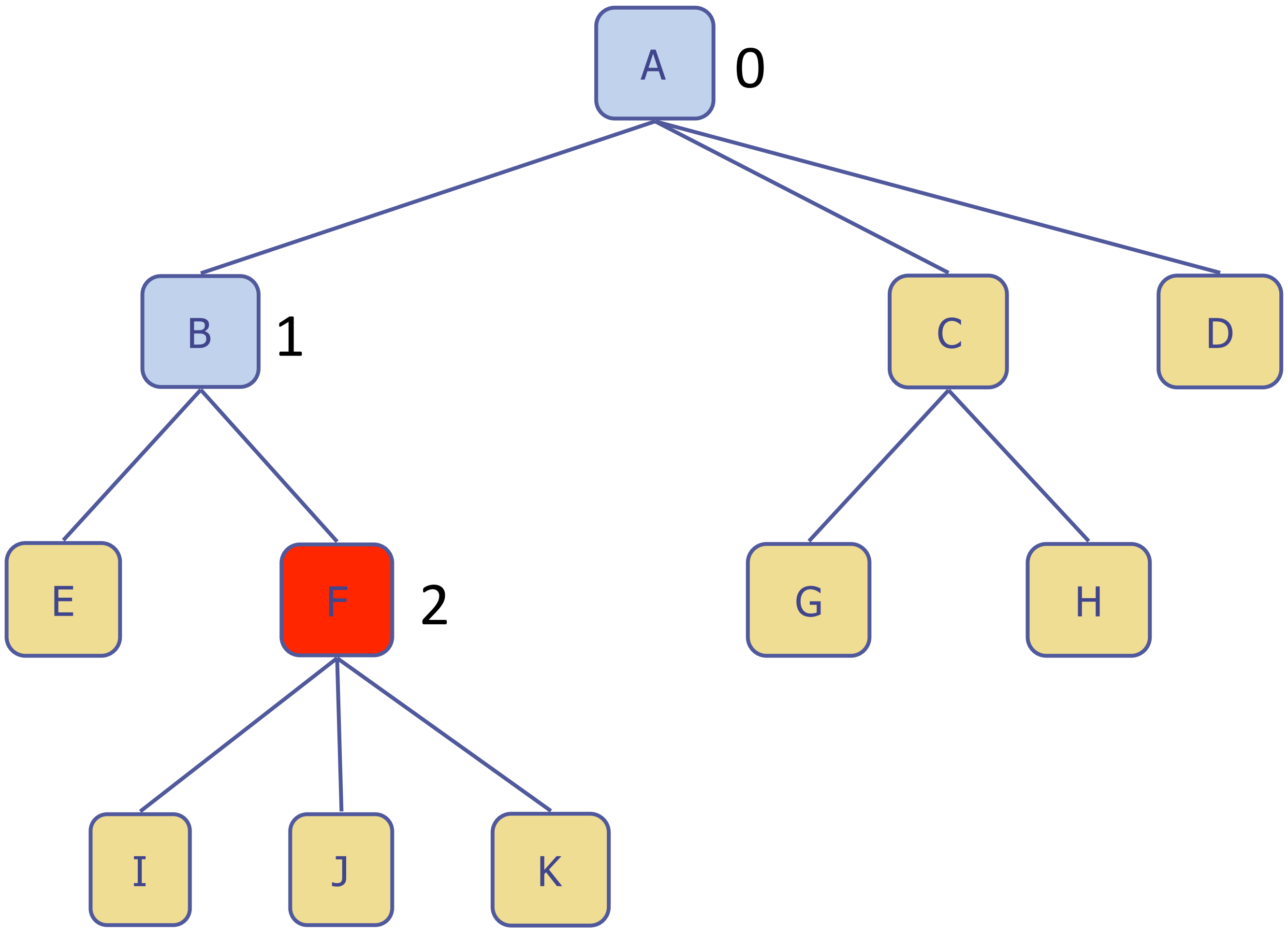
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Rekursiv definisjon av dybden til en node *v*

- Hvis *v* er rot, er dybden til *v* lik 0
- Ellers er dybden til *v* lik 1 + dybden til forelderen til *v*.

Treterminologi

rot

indre node

bladnode

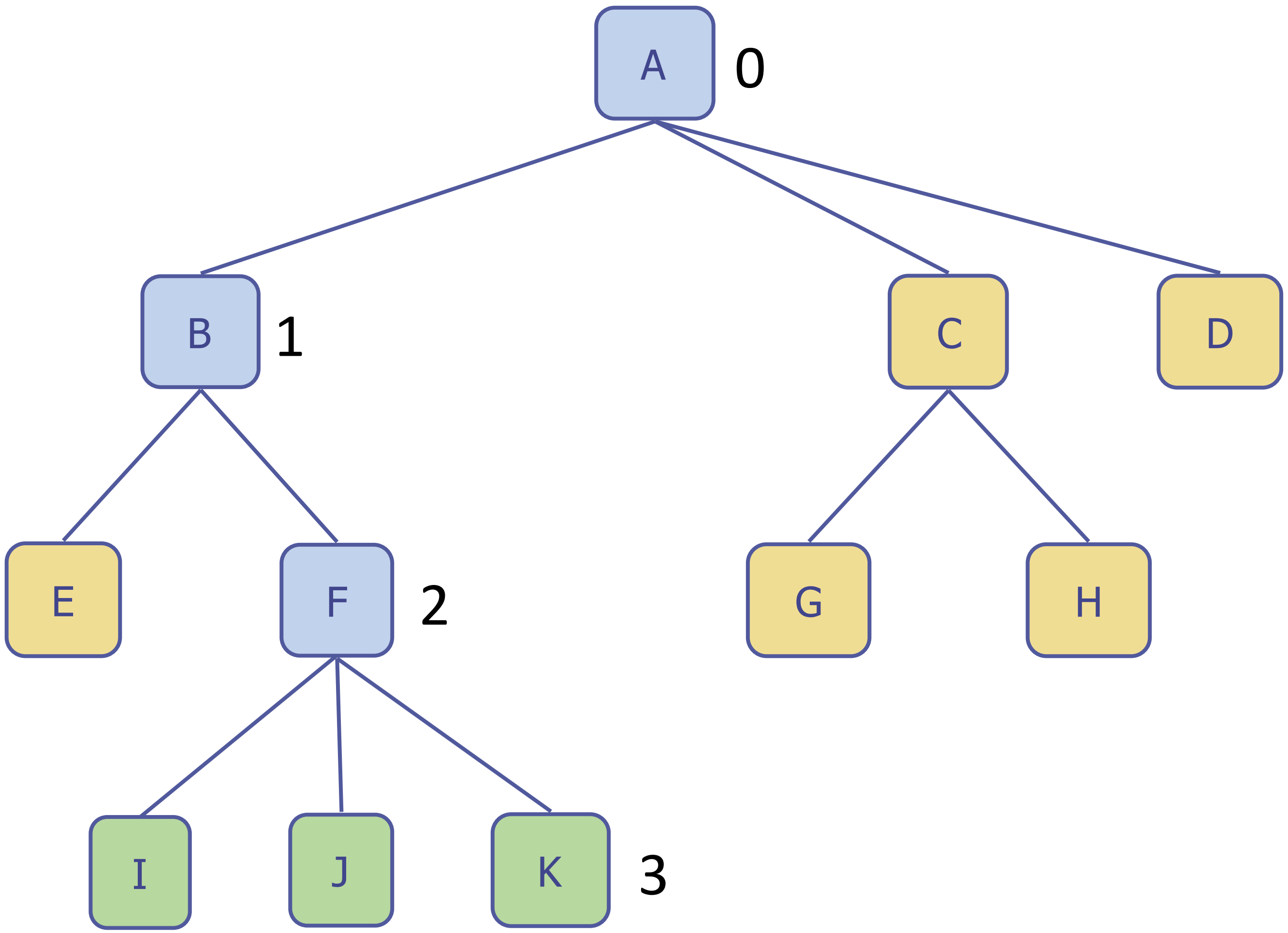
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Treterminologi

rot

indre node

bladnode

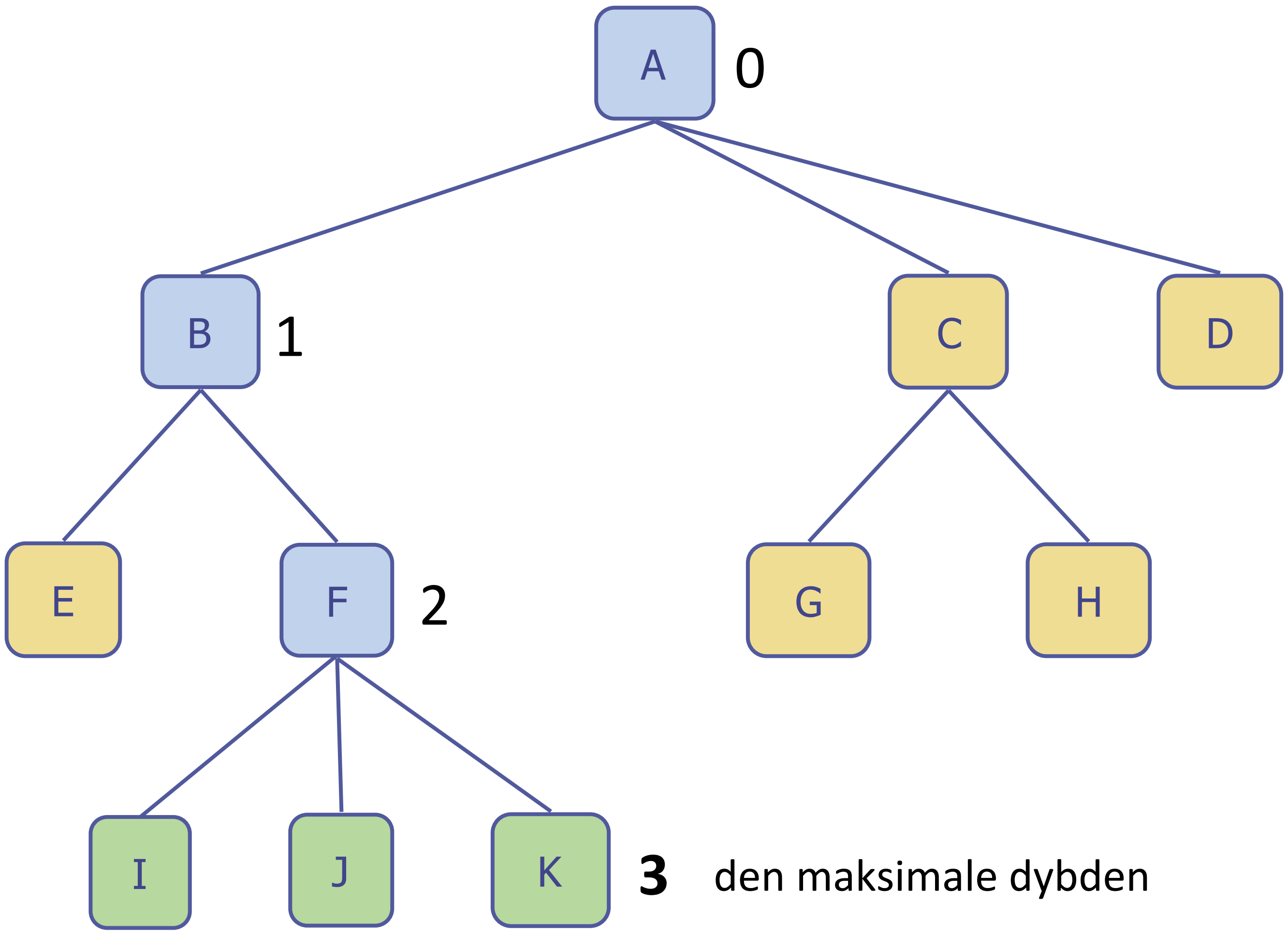
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Treterminologi

rot

indre node

bladnode

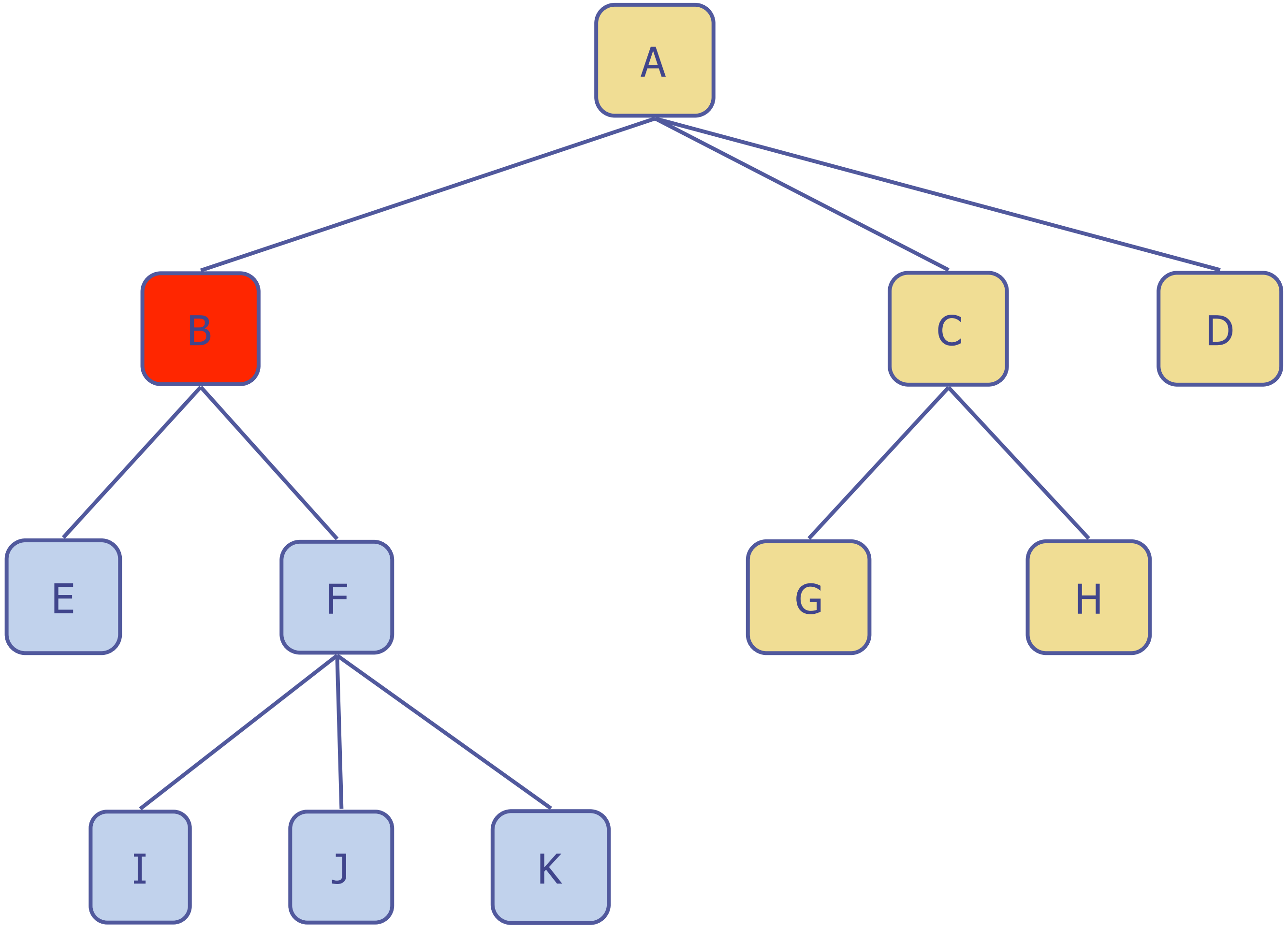
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



Treterminologi

rot

indre node

bladnode

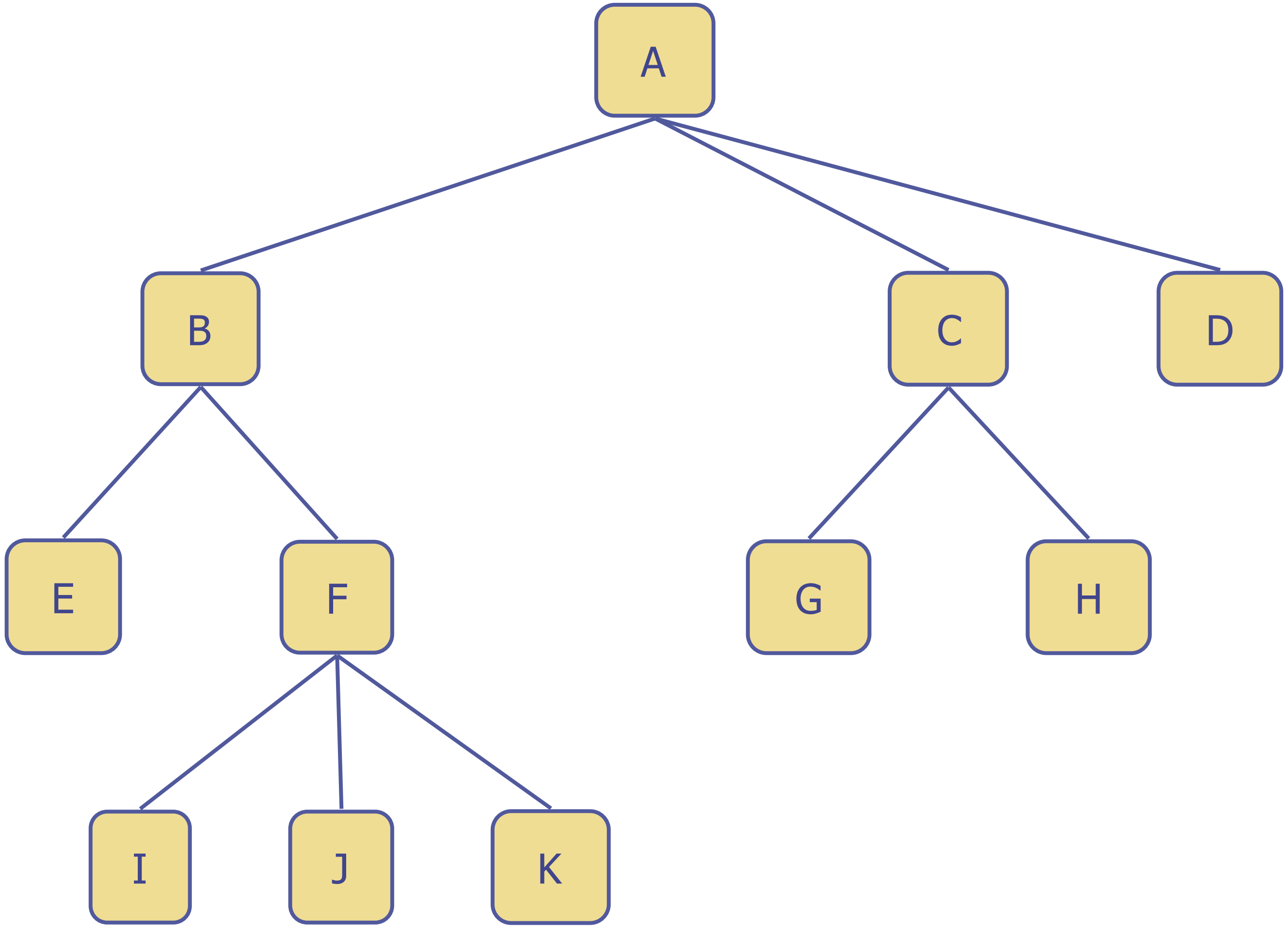
forfedre til en node

dybden til en node

høyden til et tre

etterkommere til en node

subtre



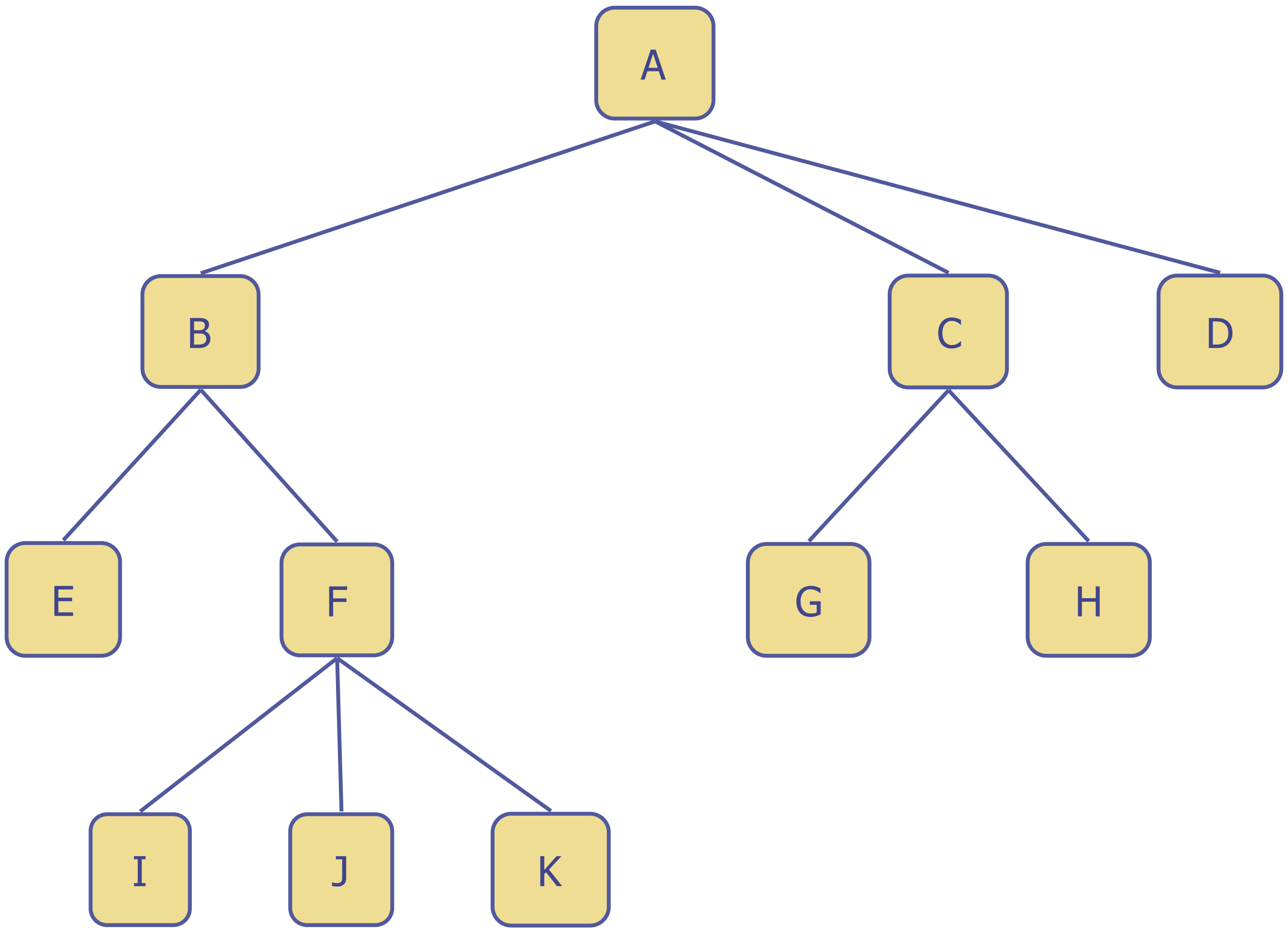
Rekursiv definisjon av et tre

Et **tre** er en samling noder.

Et ikke-tomt tre består av en **rot-node** og null eller flere ikke-tomme **subtrær**.

Fra roten går det en **rettet kant** til roten i hvert subtre.

En rettet kant er en kant med retning, den går fra en node og til en annen

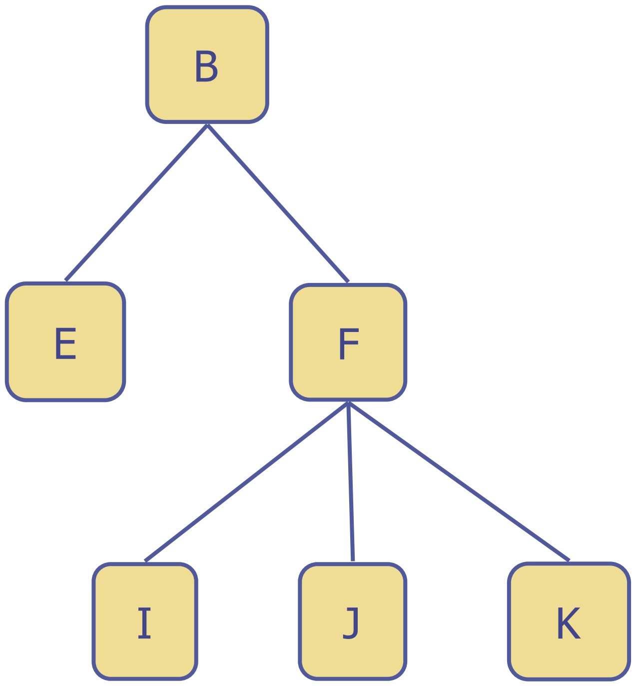


Rekursiv definisjon av et tre

Et **tre** er en samling noder.

Et ikke-tomt tre består av en **rot-node** og null eller flere ikke-tomme **subtrær**.

Fra roten går det en **rettet kant** til roten i hvert subtre.

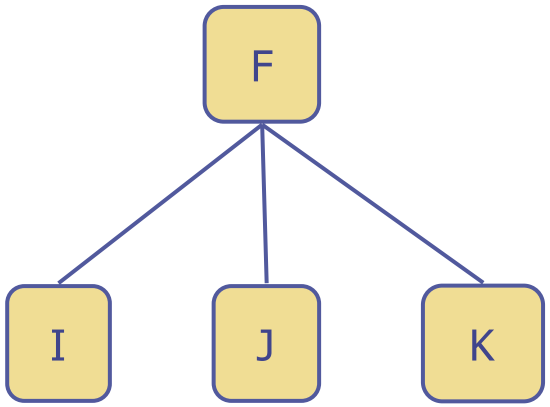


Rekursiv definisjon av et tre

Et **tre** er en samling noder.

Et ikke-tomt tre består av en **rot-node** og null eller flere ikke-tomme **subtrær**.

Fra roten går det en **rettet kant** til roten i hvert subtre.

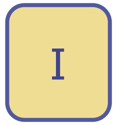


Rekursiv definisjon av et tre

Et **tre** er en samling noder.

Et ikke-tomt tre består av en **rot-node** og null eller flere ikke-tomme **subtrær**.

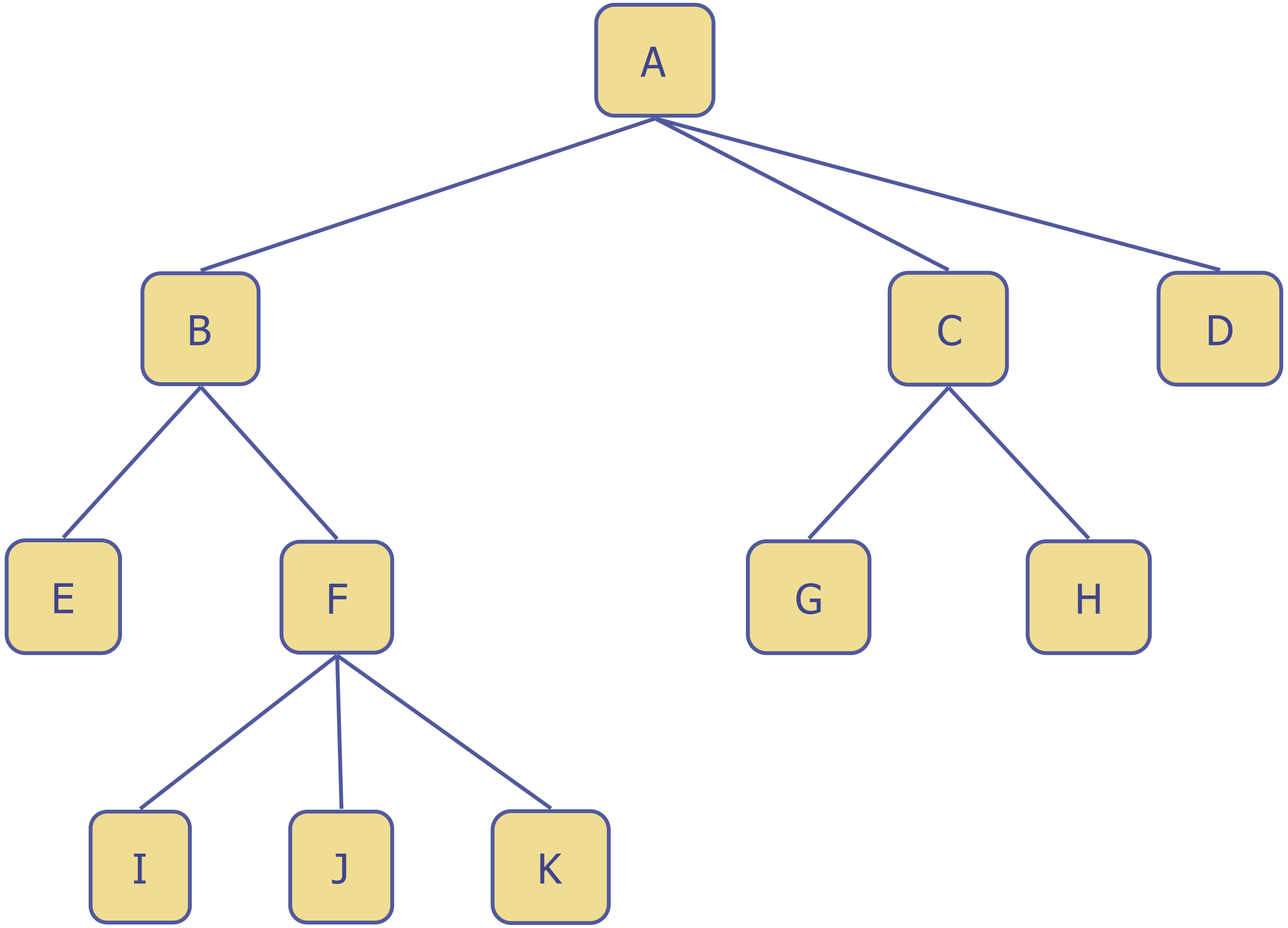
Fra roten går det en **rettet kant** til roten i hvert subtre.



Treterminologi

En **vei** (sti) fra en node n_1 til en node n_k er definert som en sekvens av noder n_1, n_2, …. n_k slik at n_i er forelder til n_{i+1} for $1 \leq i \leq k$.

Lengden av denne veien er antall kanter på veien, det vil si k-1.

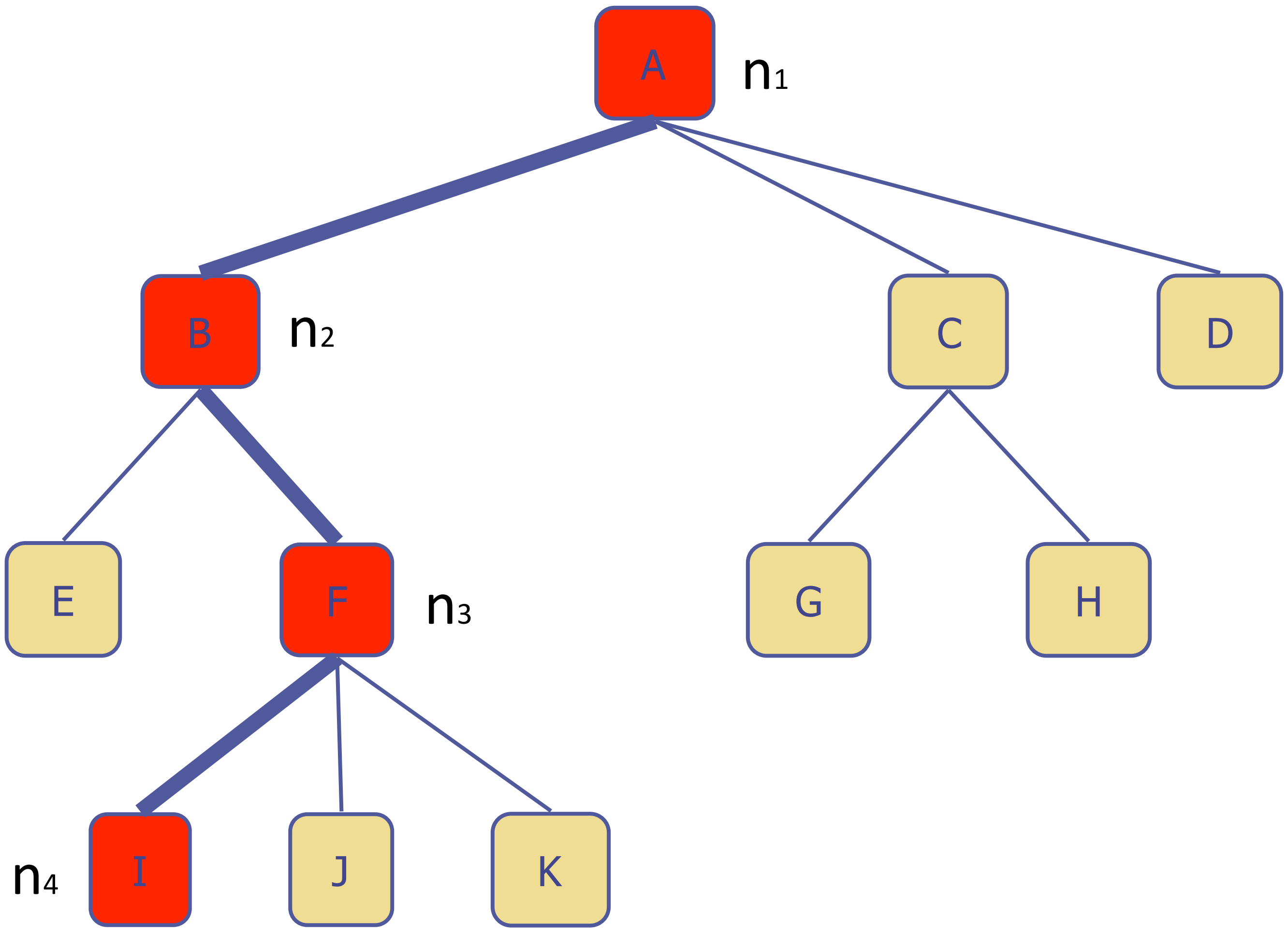


Treterminologi

En **vei** (sti) fra en node n_1 til en node n_k er definert som en sekvens av noder n_1, n_2, …. n_k slik at n_i er forelder til n_{i+1} for $1 \leq i \leq k$.

Lengden av denne veien er antall kanter på veien, det vil si k-1.

n_k



```java
class Node {

        private Node[] barn;
        private Node forelder;

        public boolean erRot() {
                return forelder == null;
        }

        public boolean erBladnode() {
                return barn[0] == null;
        }

        public boolean erIndreNode() {
                return barn[0] != null;

        }
}
```

```
class Node {

	private Node[] barn;
	private Node forelder;

	public boolean erRot() {
		return forelder == null;
	}

	public boolean erBladnode() {
		return barn[0] == null;
	}

	public boolean erIndreNode() {
		return barn[0] != null;

	}
}
```

Hvilken ***invariant*** gjelder for arrayen *barn*?

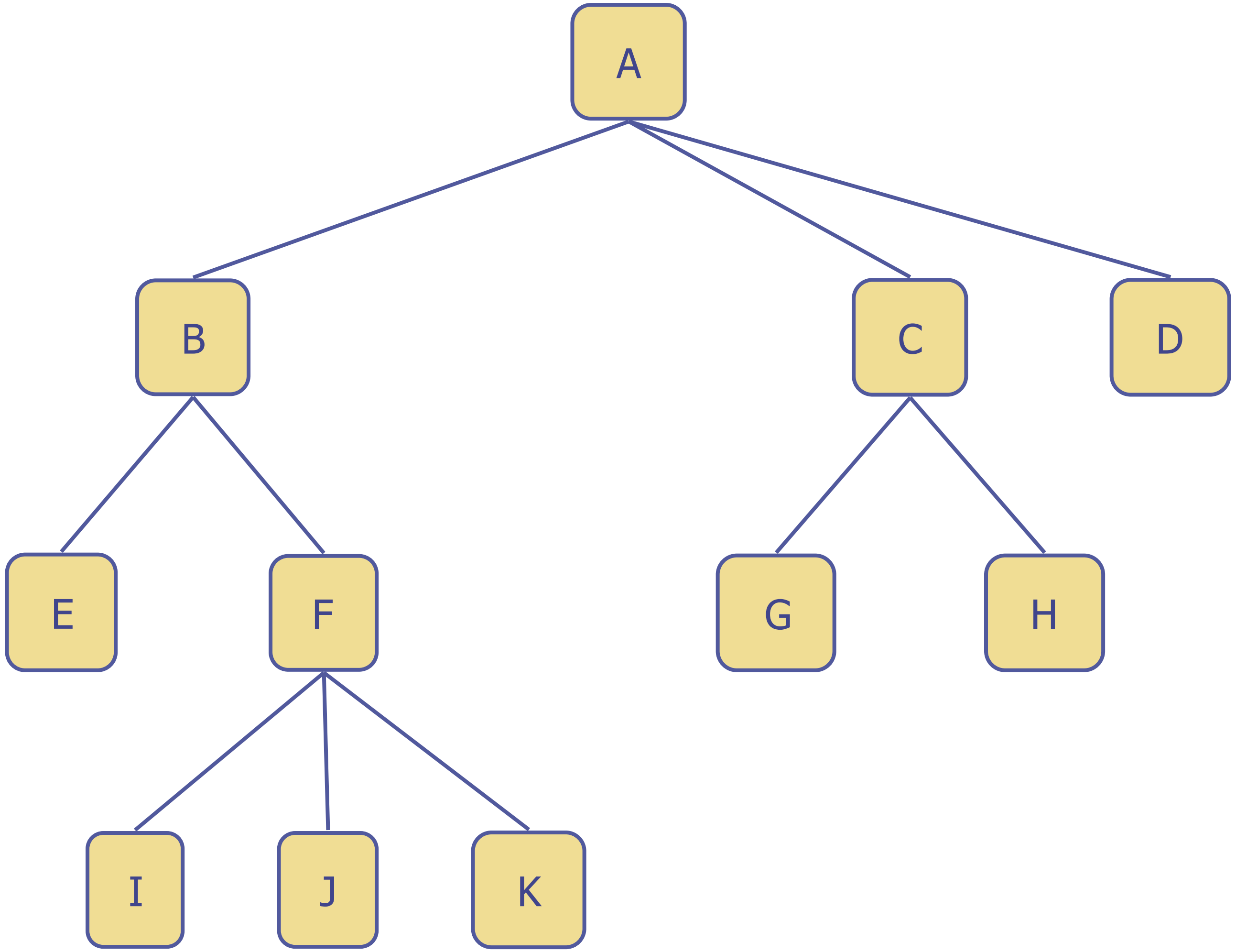
Traversering

- Gjøre noe i hver node, f.eks. skrive ut innholdet i (hele) treet
- Finne maksverdi o.l.
- Summere
- Søke noe (som ikke er i treet)

Traversering

De to vanligste måtene:

- Prefiks (preorder): behandle noden før vi går videre til barna.
- Postfiks (postorder): behandle noden etter at vi har besøkt alle barna til noden.



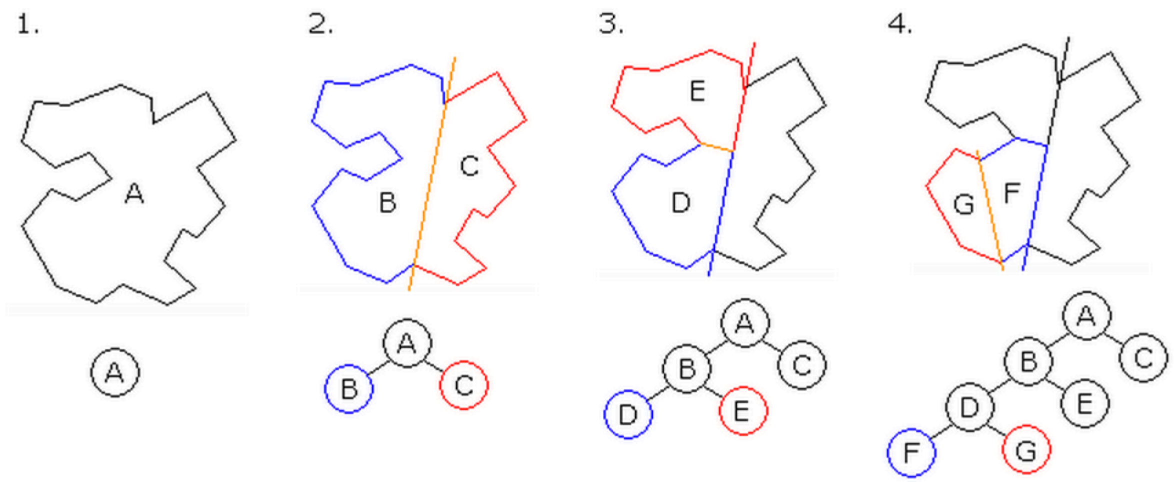
- Prefiks (preorder): A, B, E, F, I, J, K, C, G, H, D
- Postfiks (postorder): E, I, J, K, F, B, G, H, C, D, A

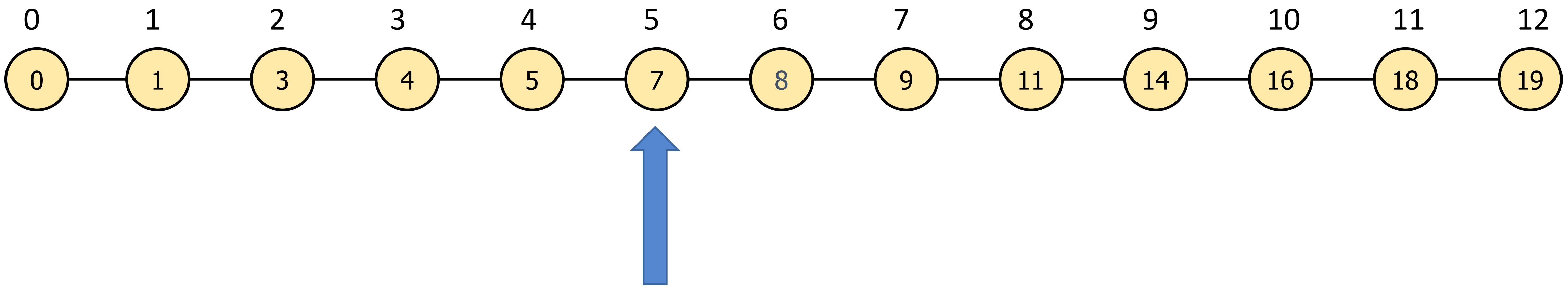
trær

binære trær

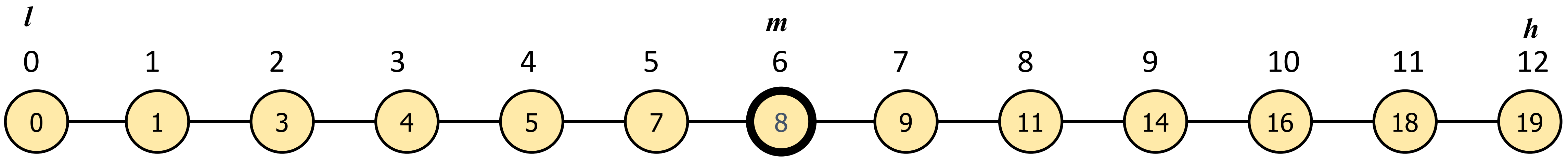
binære søketrær

Binærsøk



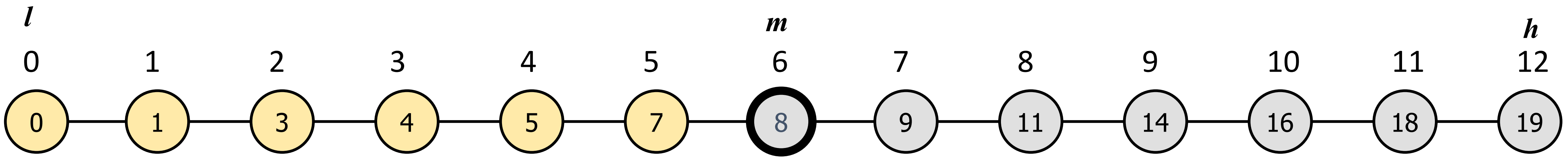


0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 3 4 5 7 8 9 11 14 16 18 19



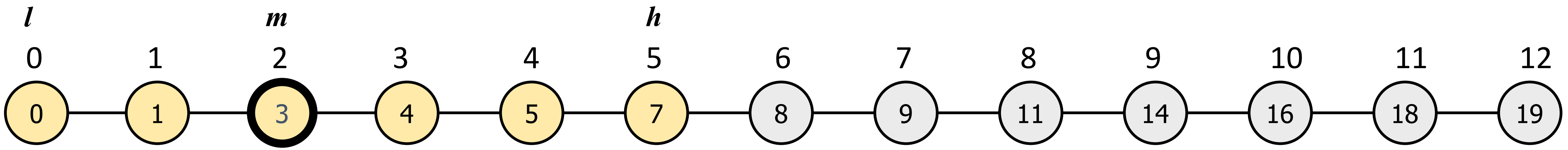
m = (l + h) / 2 ;

m = (0 + 12) / 2 = 12/2 = 6



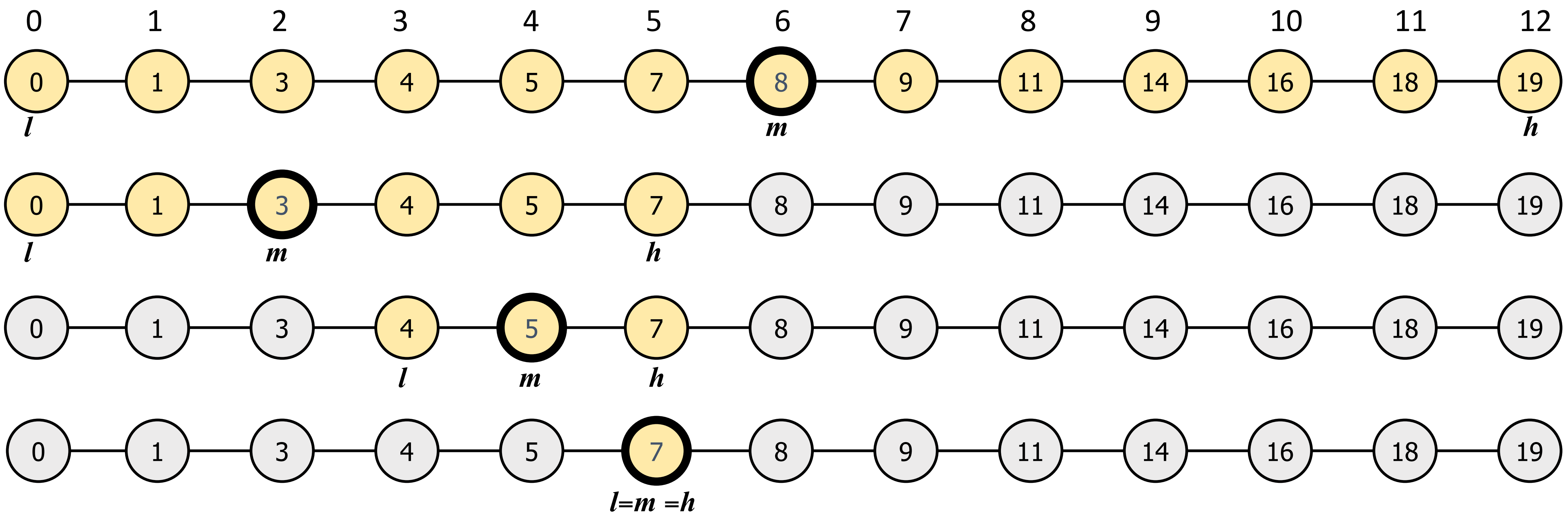
m = (l + h) / 2 ;

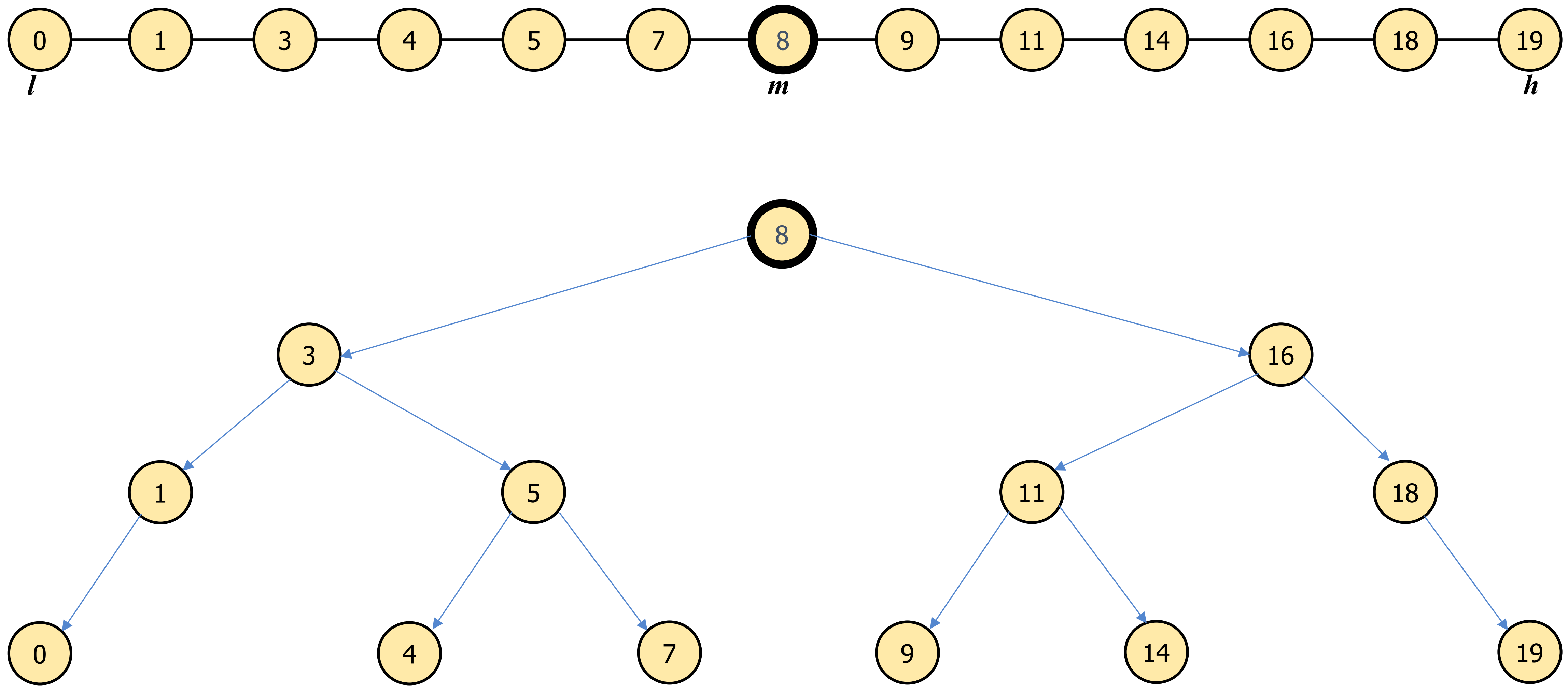
m = (0 + 12) / 2 = 12/2 = 6

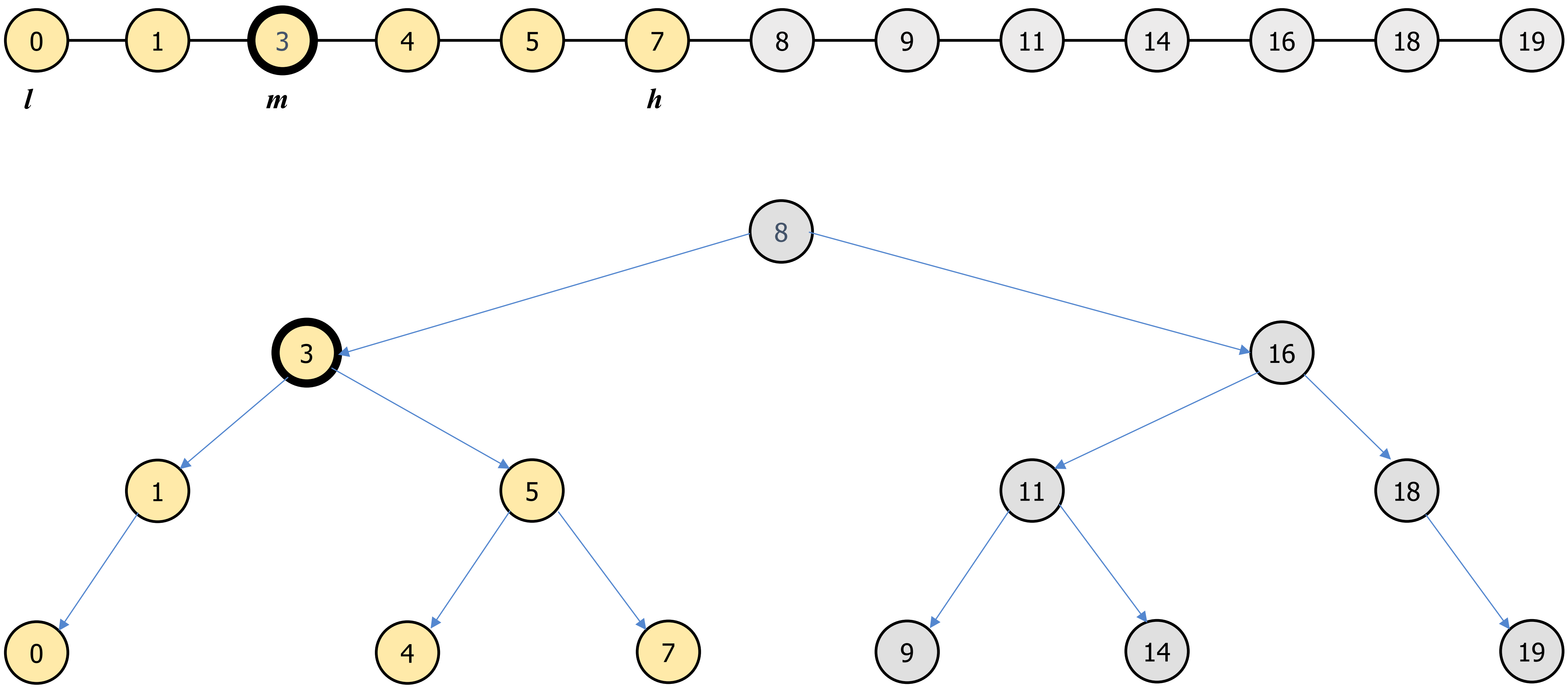


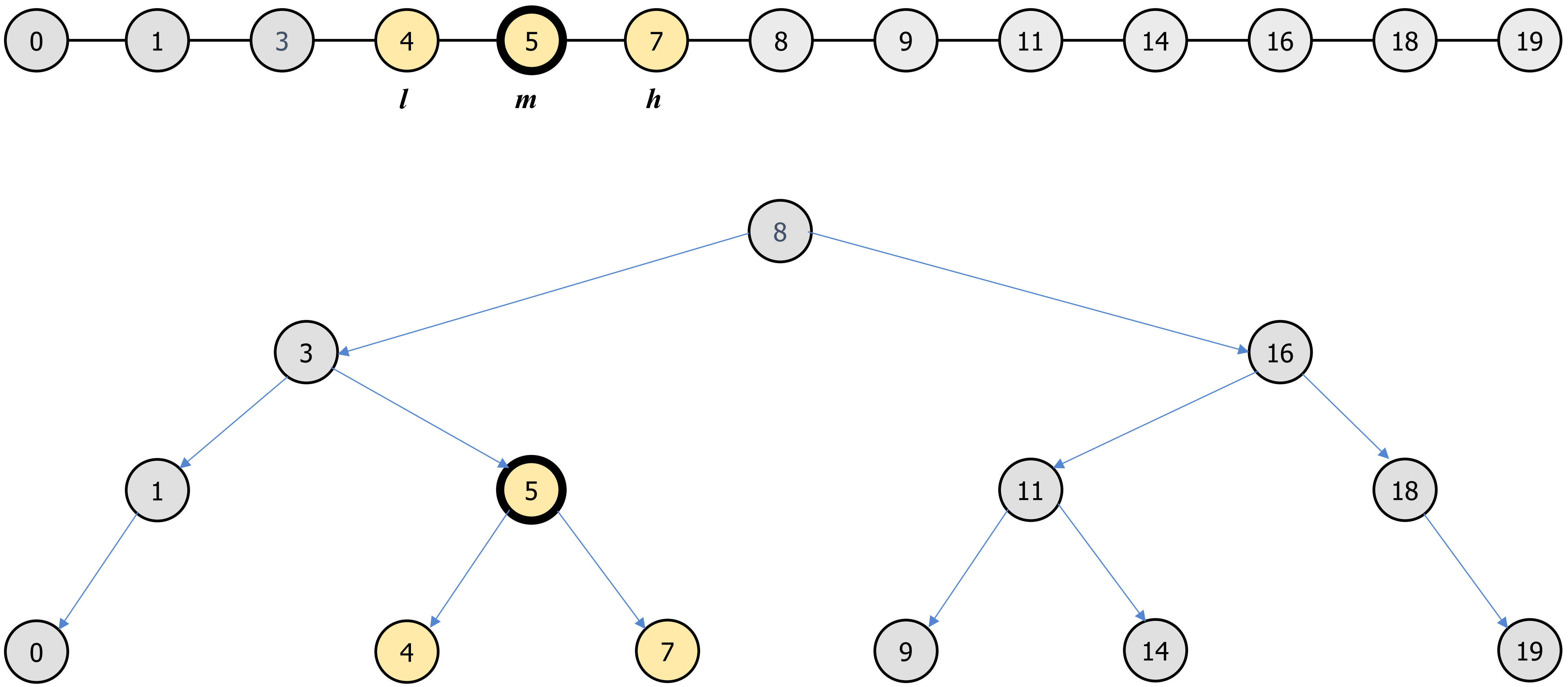
m = (l + h) / 2 ;

m = (0 + 5) / 2 = 5/2 = 2

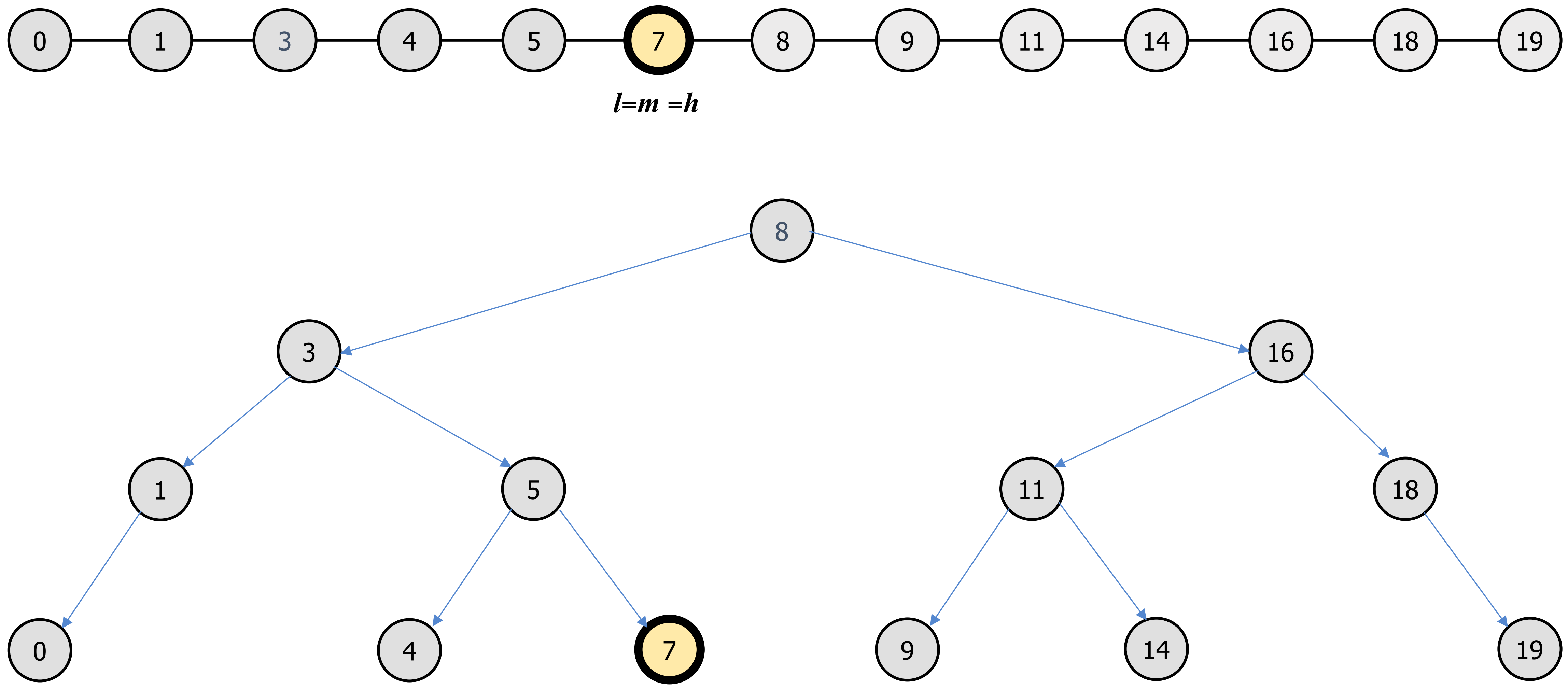

0
1
2
3
4
5
6
7
8
9
10
11
12
0
1
3
4
5
7
8
9
11
14
16
18
19
l
m
h
l
m
h
l
m
h
l=m =h

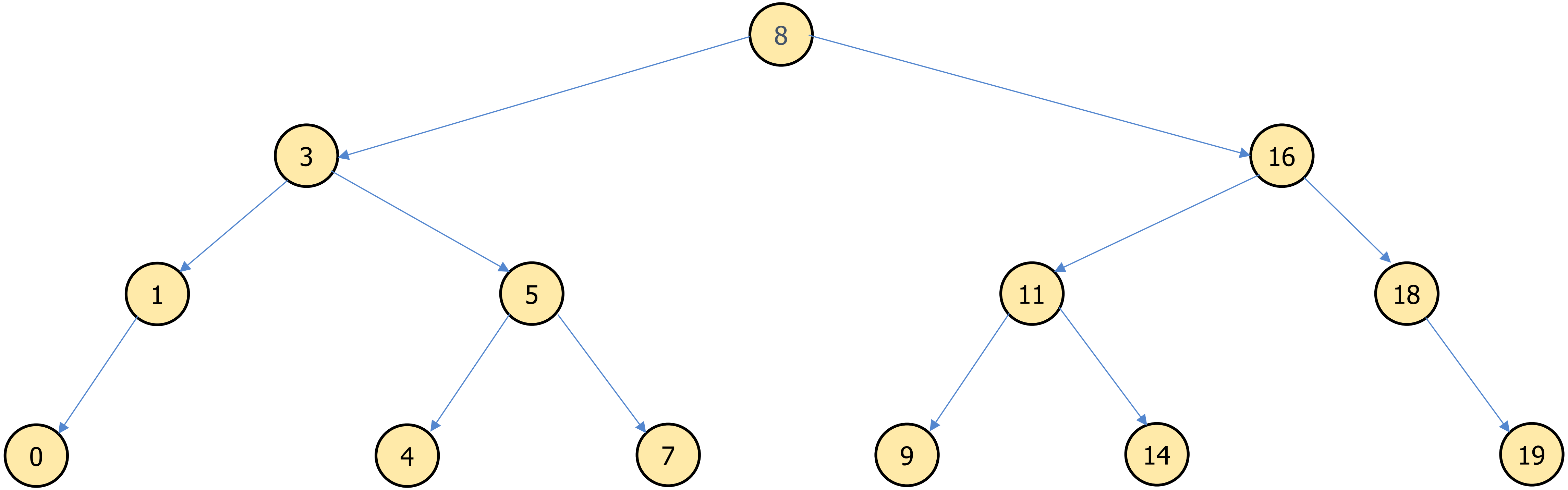

0 1 3 4 5 7 8 9 11 14 16 18 19
l
m
h
8
3
16
1
5
11
18
0
4
7
9
14
19


0
1
3
4
5
7
8
9
11
14
16
18
19
l
m
h
8
3
16
1
5
11
18
0
4
7
9
14
19



0
1
3
4
5
7
8
9
11
14
16
18
19
l
m
h
8
3
16
1
5
11
18
0
4
7
9
14
19


0
1
3
4
5
7
8
9
11
14
16
18
19
l=m =h
8
3
16
1
5
11
18
0
4
7
9
14
19



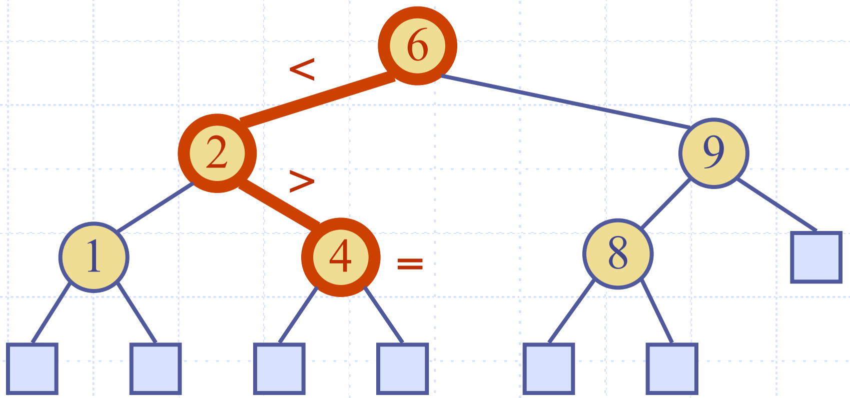
Binære søketrær er binærtrær hvor følgende gjelder for hver node i treet:

- Alle verdiene i ***venstre*** subtre er ***mindre*** enn verdien i noden selv.
- Alle verdiene i ***høyre*** subtre er ***større*** enn verdien i noden selv.

Search

- To search for a key k, we trace a downward path starting at the root
- The next node visited depends on the comparison of k with the key of the current node
- If we reach a leaf, the key is not found
- Example: get(4):
  - Call TreeSearch(4,root)
- The algorithms for nearest neighbor queries are similar

Algorithm *TreeSearch*(k, v)
    if *T.isExternal* (v)
        return v
    if $k < key(v)$
        return *TreeSearch*(k, *leftChild*(v))
    else if $k = key(v)$
        return v
    else { $k > key(v)$ }
        return *TreeSearch*(k, *rightChild*(v))

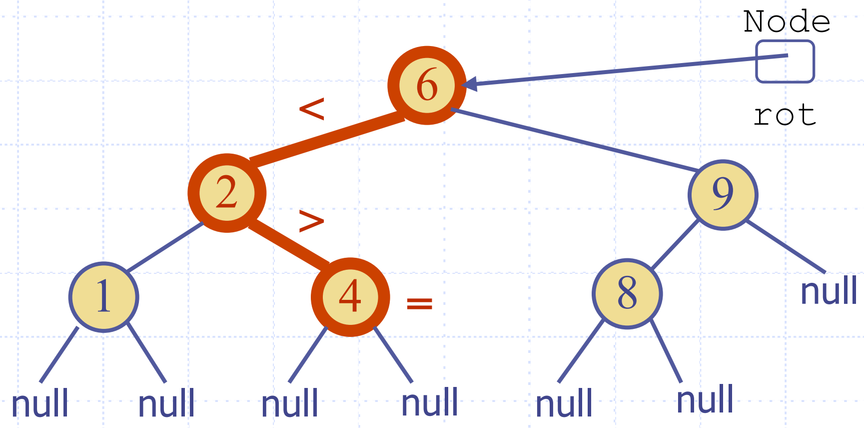


```java
public class BinTre {
  Node rot = null;

  class Node {
    int verdi;
    Node venstre, hoyre;
  }

  public Node finnVerdiIBinTre (int verdi, Node tre) {
    Node retur = null;
    if ( tre == null )
      retur = null;
    else if ( verdi < tre.verdi )
      retur = finnVerdiIBinTre( verdi, tre.venstre );
    else if ( verdi == tre.verdi )
      retur = tre;
    else if ( verdi > tre.verdi )
      retur = finnVerdiIBinTre( verdi, tre.hoyre );
    return retur;
  }
}
```

Søking

- For å søke etter en node med en bestemt Verdi starter vi i rotnoden og søker nedover i treet
- Neste subtre det skal søkes I avhenger av sammenligningen mellom verdien vi leter etter og verdien i noden
- Hvis subtreet vi skal fortsette å søke i er tomt, finnes ikke verdien i treet.
- Eksempel:
  - `finnVerdiIBintre(4,rot);`

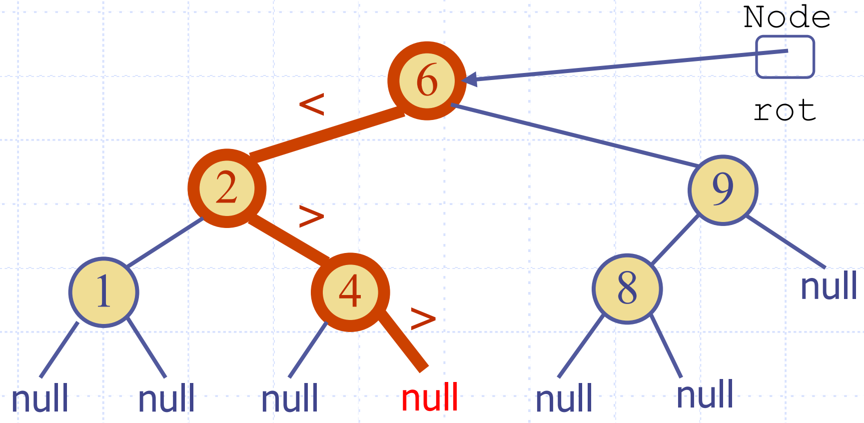
```
public Node finnVerdiIBinTre (int verdi, Node tre) {
   Node retur = null;
   if ( tre == null )
      retur = null;
   else if ( verdi < tre.verdi )
      retur = finnVerdiIBinTre( verdi, tre.venstre );
   else if ( verdi == tre.verdi )
      retur = tre;
   else if ( verdi > tre.verdi )
      retur = finnVerdiIBinTre( verdi, tre.hoyre );
   return retur;
}
```



Søking

- For å søke etter en node med en bestemt Verdi starter vi i rotnoden og søker nedover i treet
- Neste subtre det skal søkes I avhenger av sammenligningen mellom verdien vi leter etter og verdien i noden
- Hvis subtreet vi skal fortsette å søke i er tomt, finnes ikke verdien i treet.
- Eksempel:
  - `finnVerdiIBintre(5,rot);`

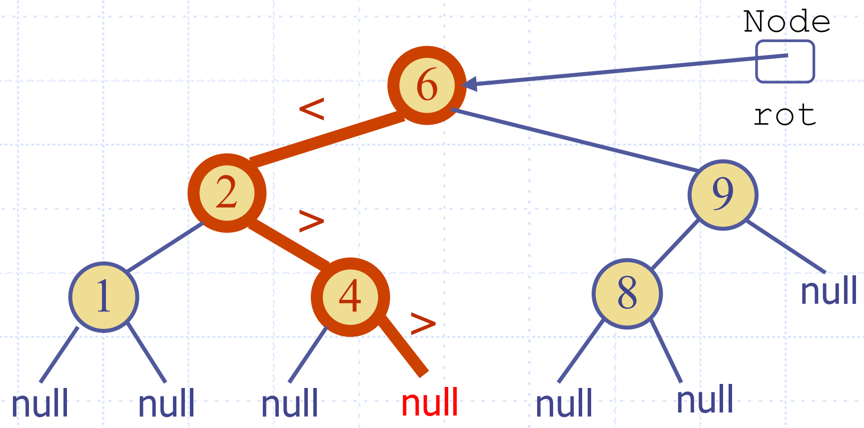
```
public Node finnVerdiIBinTre (int verdi, Node tre) {
    Node retur = null;
    if ( tre == null )
        retur = null;
    else if ( verdi < tre.verdi )
        retur = finnVerdiIBinTre( verdi, tre.venstre );
    else if ( verdi == tre.verdi )
        retur = tre;
    else if ( verdi > tre.verdi )
        retur = finnVerdiIBinTre( verdi, tre.hoyre );
    return retur;
}
```



Søking

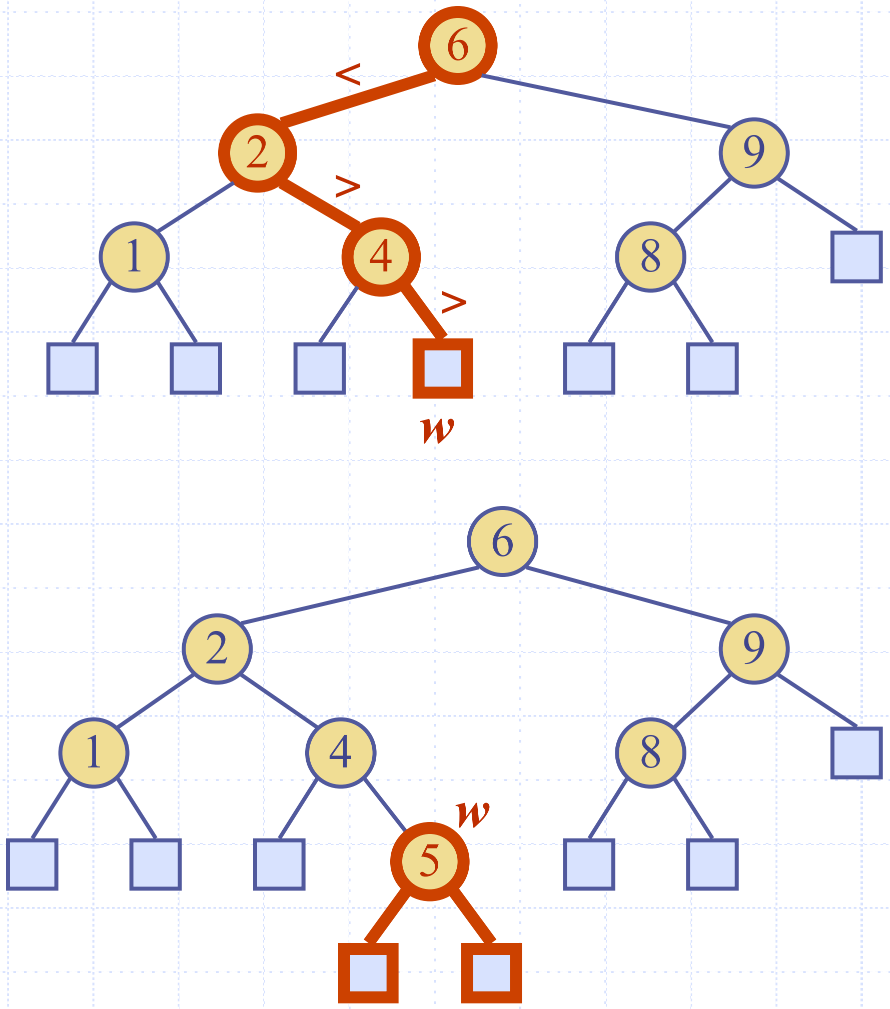
- For å søke etter en node med en bestemt Verdi starter vi i rotnoden og søker nedover i treet
- Neste subtre det skal søkes I avhenger av sammenligningen mellom verdien vi leter etter og verdien i noden
- Hvis subtreet vi skal fortsette å søke i er tomt, finnes ikke verdien i treet.
- Eksempel:
  - `finnVerdiIBintre(5,rot);`

```java
public Node finnVerdiIBinTre (int verdi, Node tre) {
        if ( tre == null )
                return null;
        else if ( verdi < tre.verdi )
                return finnVerdiIBinTre(verdi,tre.venstre);
        else if ( verdi == tre.verdi )
                return tre;
        else   // verdi > tre.verdi
                return finnVerdiIBinTre( verdi, tre.hoyre );
}
```



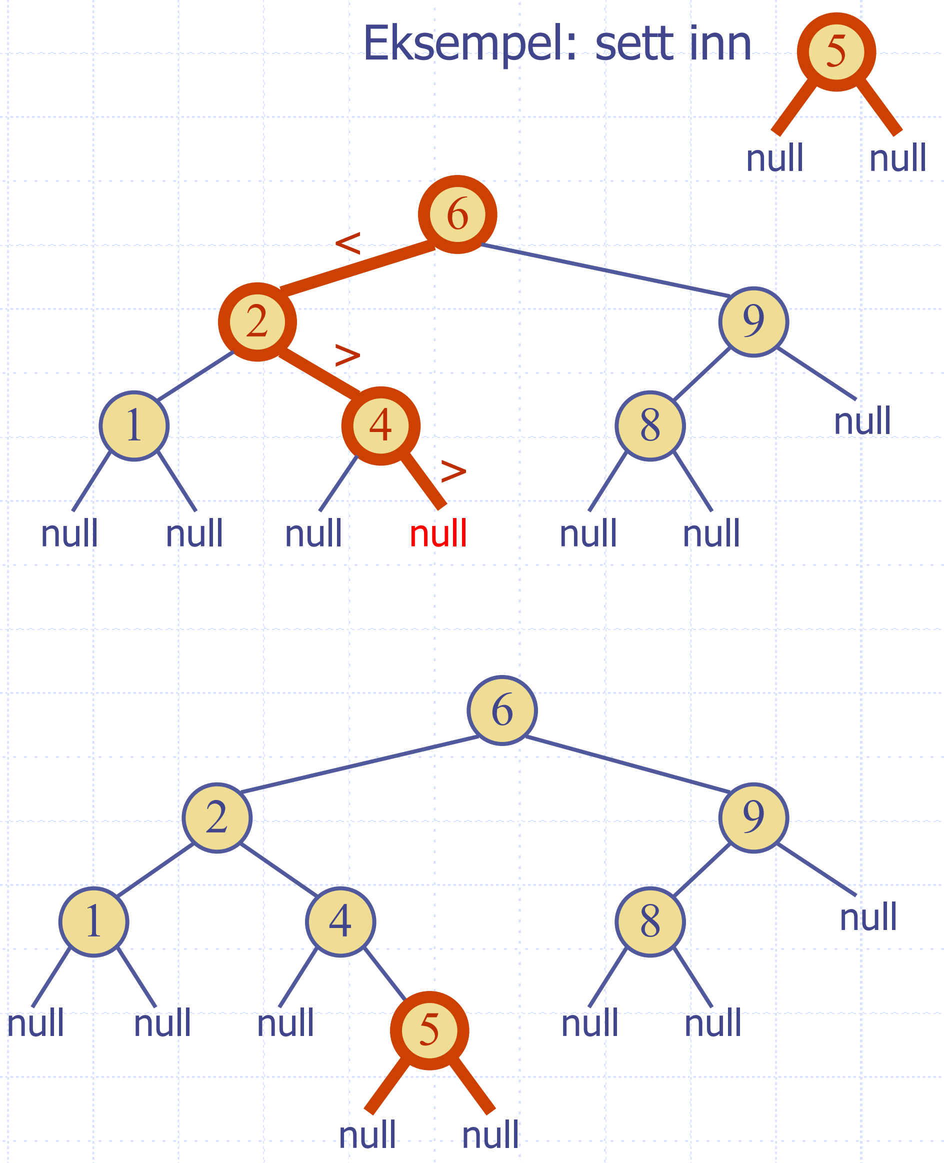
Insertion

- To perform operation put(k, o). we search for key k (using TreeSearch)
- Assume k is not already in the tree, and let w be the leaf reached by the search
- We insert k at node w and expand w into an internal node
- Example: insert 5



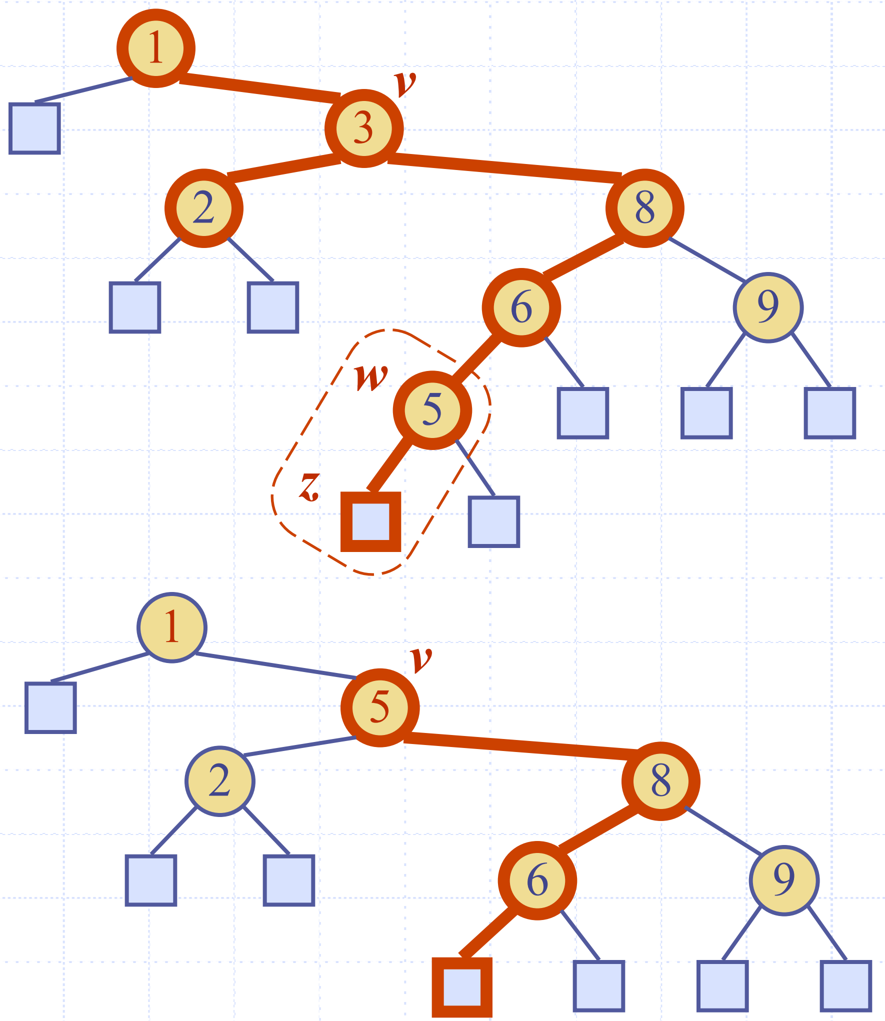
Innsetting

- Vi søker først på vanlig måte etter en node med verdien k vi skal sette inn
- Anta at k ikke finnes i treet fra før. Vi vil da ende opp med å finne et tomt subtree (nullpeker) der noden med verdi k skulle vært.
- Vi setter inn den nye noden istedet for det tomme treet vi fant.
- Eksempel: sett inn 5



Deletion (cont.)

- We consider the case where the key *k* to be removed is stored at a node *v* whose children are both internal
  - we find the internal node *w* that follows *v* in an inorder traversal
  - we copy *key*(*w*) into node *v*
  - we remove node *w* and its left child *z* (which must be a leaf) by means of operation removeExternal(*z*)
- Example: remove 3

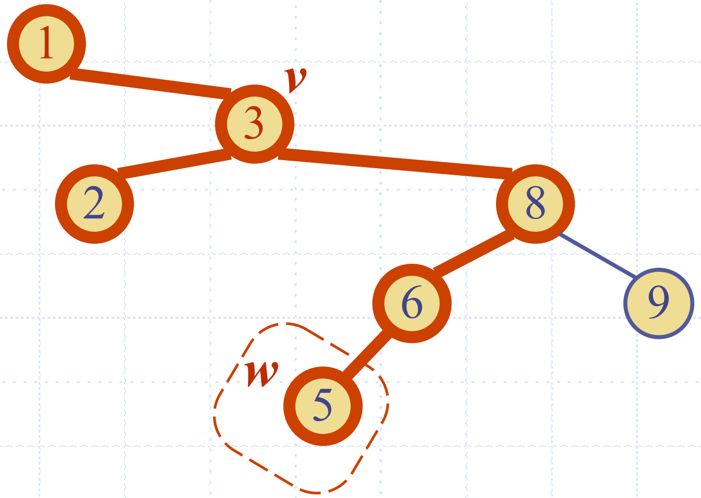


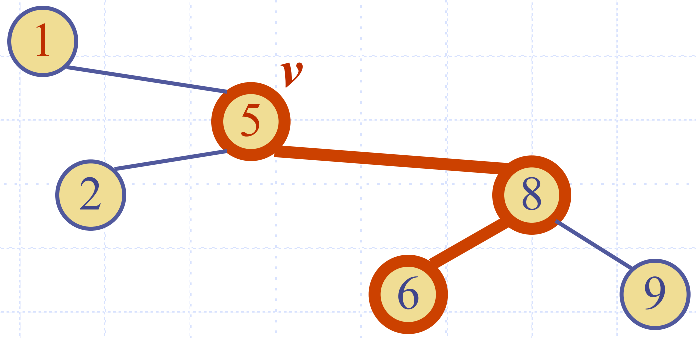
Fjerning av indre node

Vi ser på et eksempel der noden v som skal fjernes ikke er en bladnode:

- Vi finner den minste noden w i det høyre subtreet til v
- Vi erstatter v med w
- Vi fjerner noden w

Eksempel: Fjern noden med verdi 3

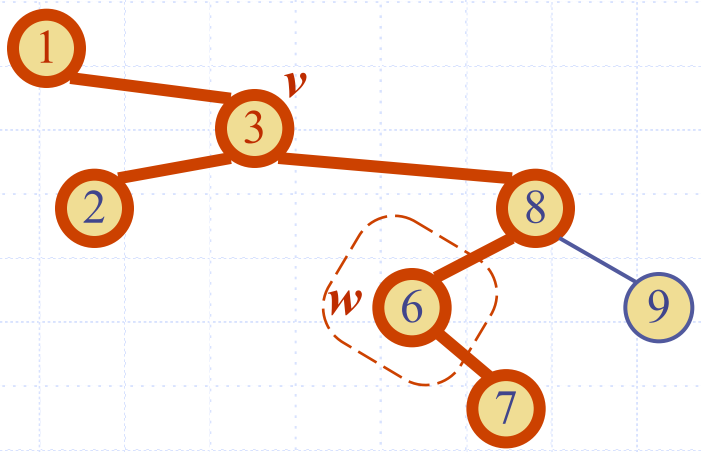




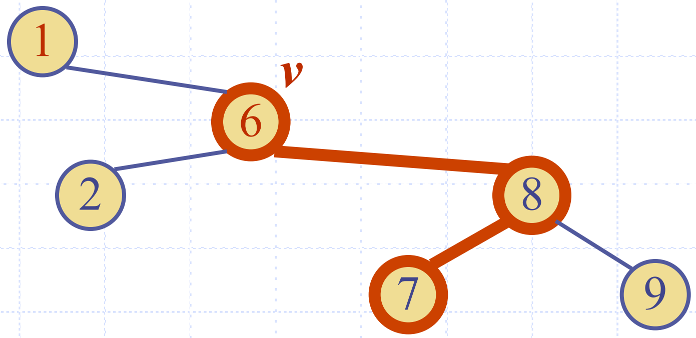
Fjerning av indre node

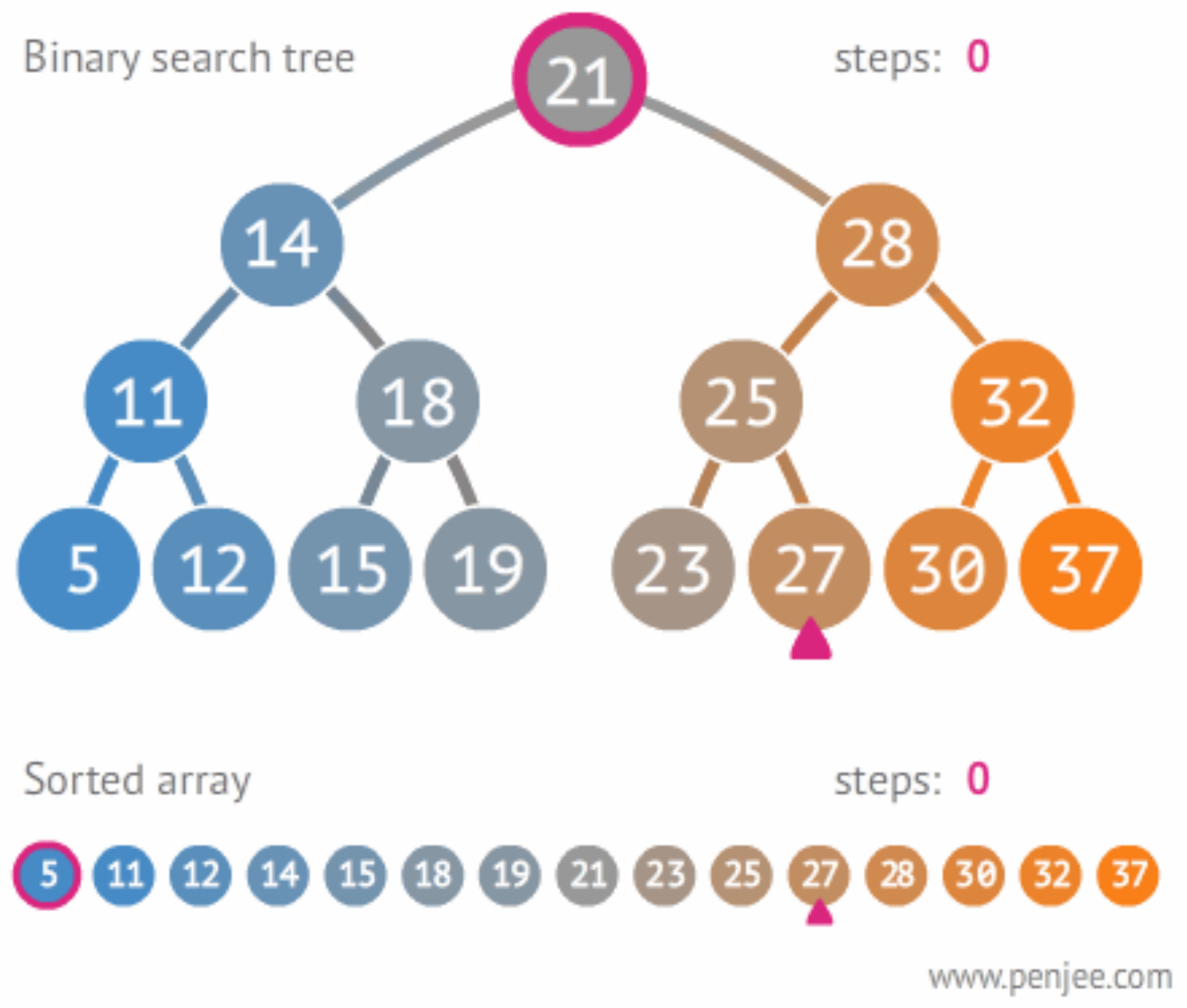
Vi ser på et eksempel der noden v som skal fjernes ikke er en bladnode:

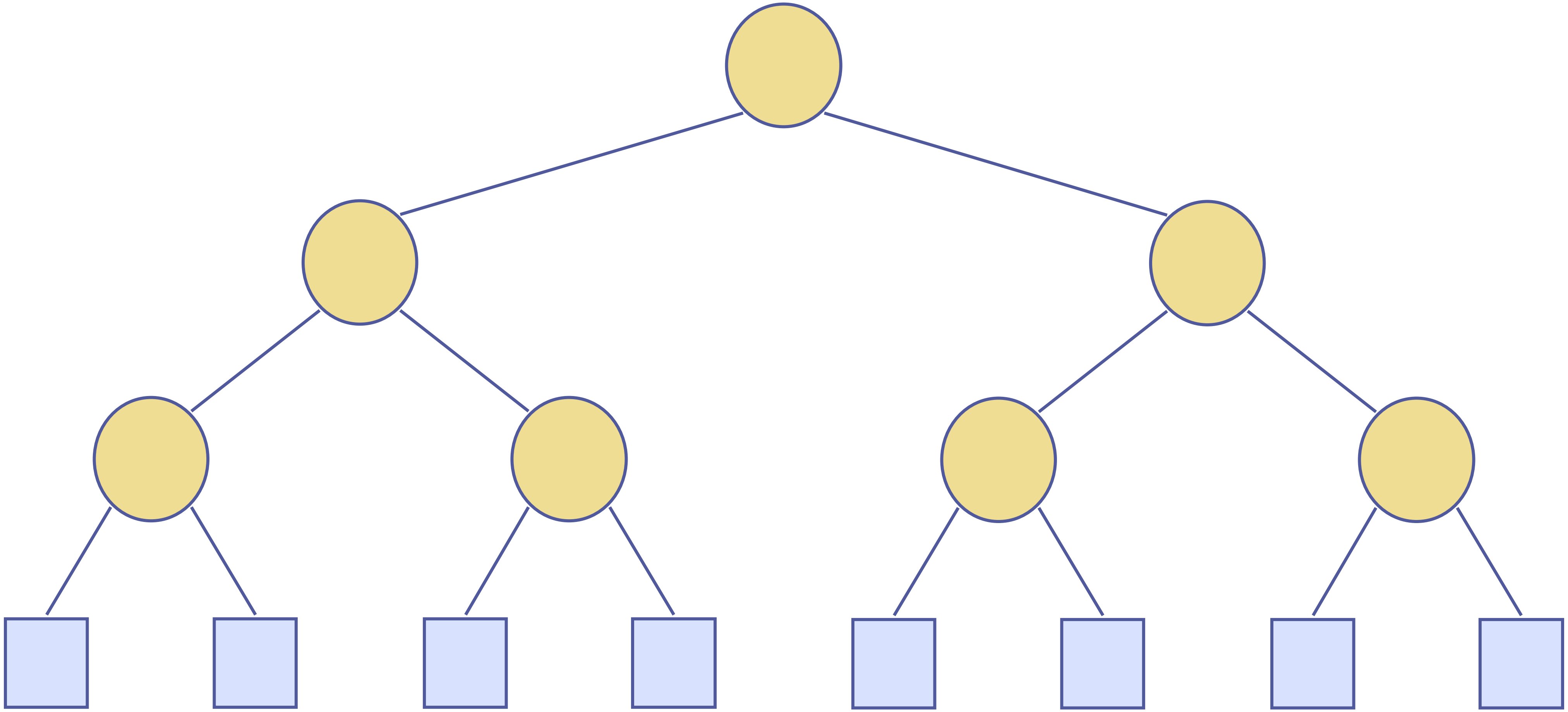
- Vi finner den minste noden w i det høyre subtreet til v
- Vi erstatter v med w
- Vi fjerner noden w



Eksempel: Fjern noden med verdi 3




Binary search tree
steps: 0
21
14
28
11
18
25
32
5
12
15
19
23
27
30
37
Sorted array
steps: 0
5 11 12 14 15 18 19 21 23 25 27 28 30 32 37
www.penjee.com



1

3

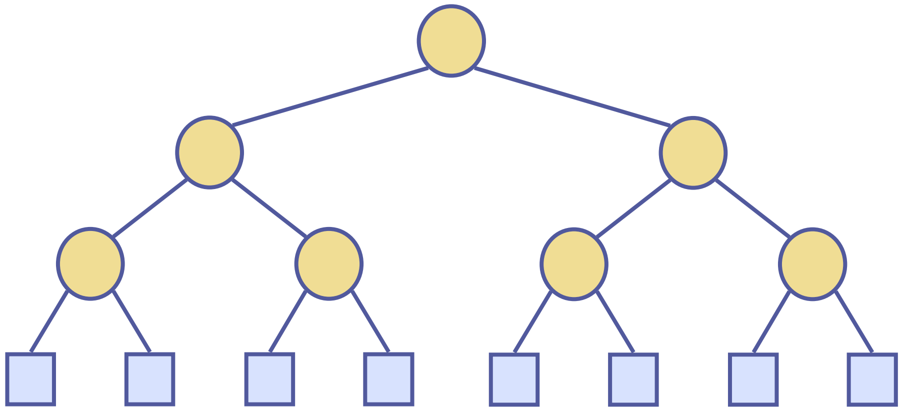
7

15

31

63

127



n	2^n	$2^n - 1$
0	1	0
1	2	1
2	4	3
3	8	7
4	16	15
5	32	31
6	64	63

$$2^n-1$$

$2^1 = 2$
$2^2 = 4$
$2^4 = 16$
$2^8 = 256$
$2^{16} = 65.536$
$2^{32} = 4.294.967.296$
$2^{64} = 18.446.744.073.709.551.616$ (20 siffer)
$2^{128} = 340.282.366.920.938.463.463.374.607.431.768.211.456$ (39 siffer)

$2^{256} =$
115.792.089.237.316.195.423.570.985.008.687.907.853.269.984.665.640.564.039.457.584.007.913.129.
639.936 (78 siffer)

$2^{512} =$
13.407.807.929.942.597.099.574.024.998.205.846.127.479.365.820.592.393.377.723.561.443.721.764.0
30.073.546.976.801.874.298.166.903.427.690.031.858.186.486.050.853.753.882.811.946.569.946.433.6
49.006.084.096 (155 siffer)