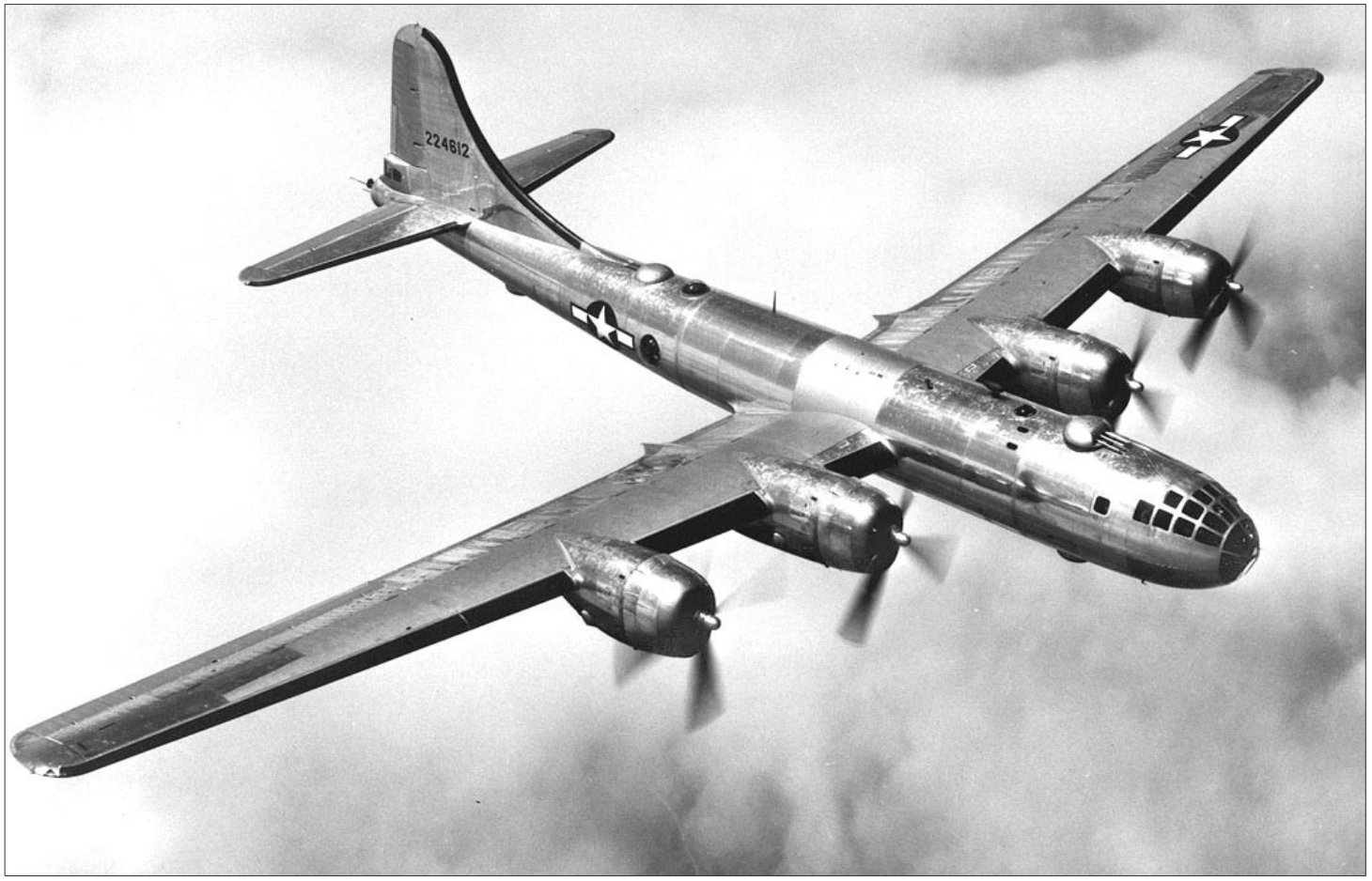
Flerveis søketrær og B-trær



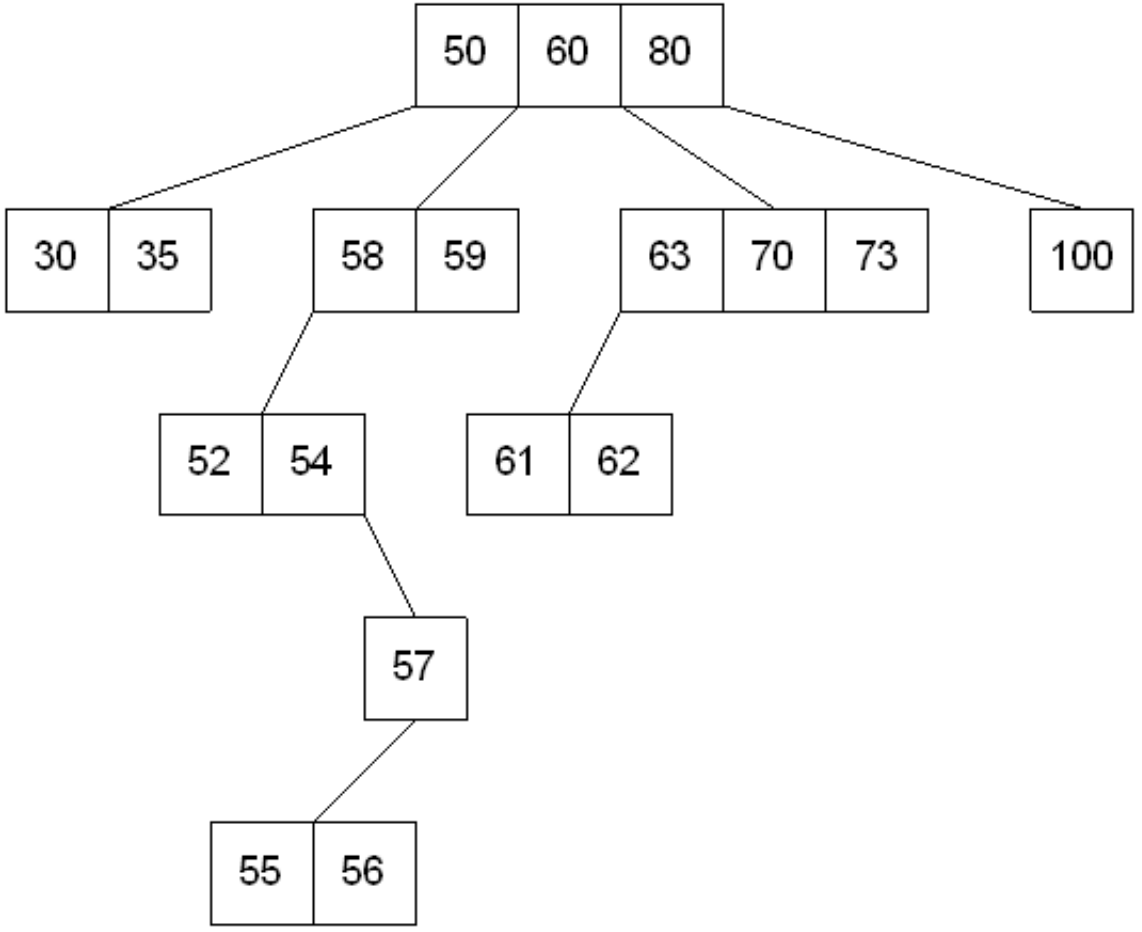
Flerveis søketre *

- Generalisering av binært søketre
- Binært søketre:
  - Hver node har maksimalt 2 subtrær/barn og 1 verdi
  - Barna ligger sortert på verdi i forhold til den ene verdien
- Flerveis søketre av orden** *m*:
  - Hver node har maksimalt *m* barn og *m* – 1 sorterte verdier
  - Barna ligger sortert «innimellom» verdiene i noden

*: Aka. «multi-way search tree», «n-ært søketre»

**: Begrepet orden for et søketre er ikke entydig definert i litteraturen

Et flerveis søketre av orden 4

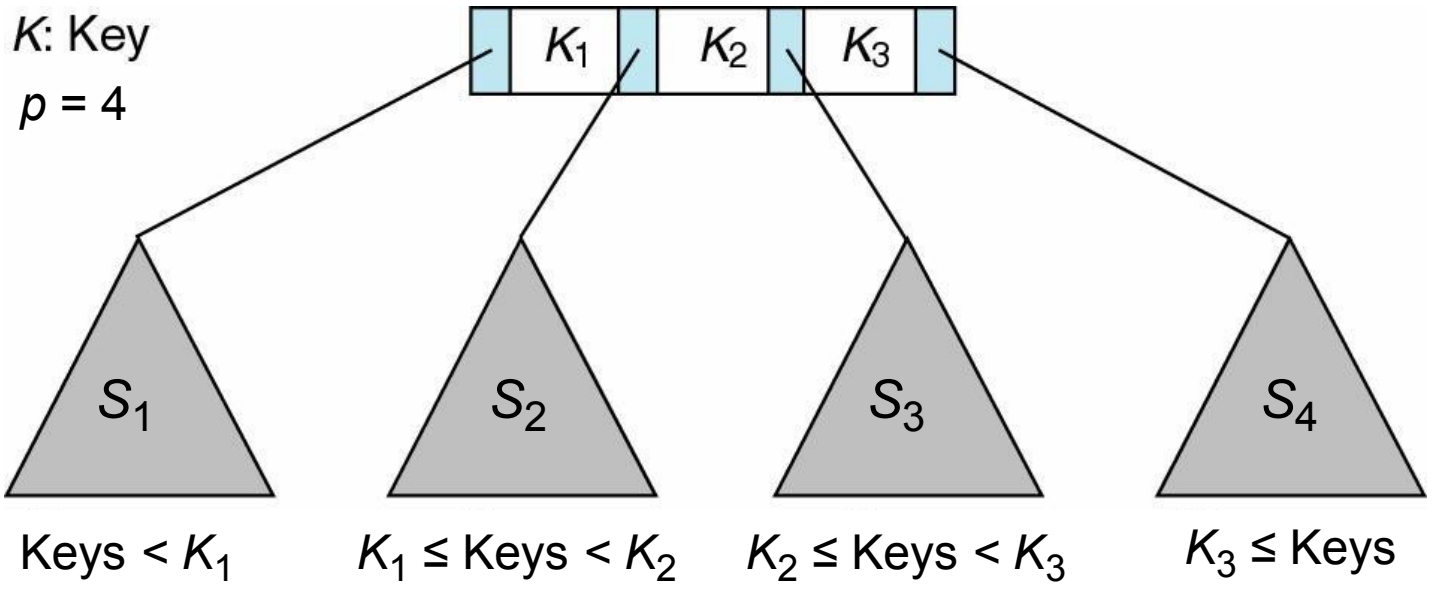


Ordning av et flerveis søketre

- En node med p barn/subtrær har $p - 1$ verdier
- Verdier: $K_1, K_2, \ldots, K_{p-1}$
- Subtrær: $S_1, S_2, \ldots, S_p$
- Verdiene internt i en node er sortert stigende:

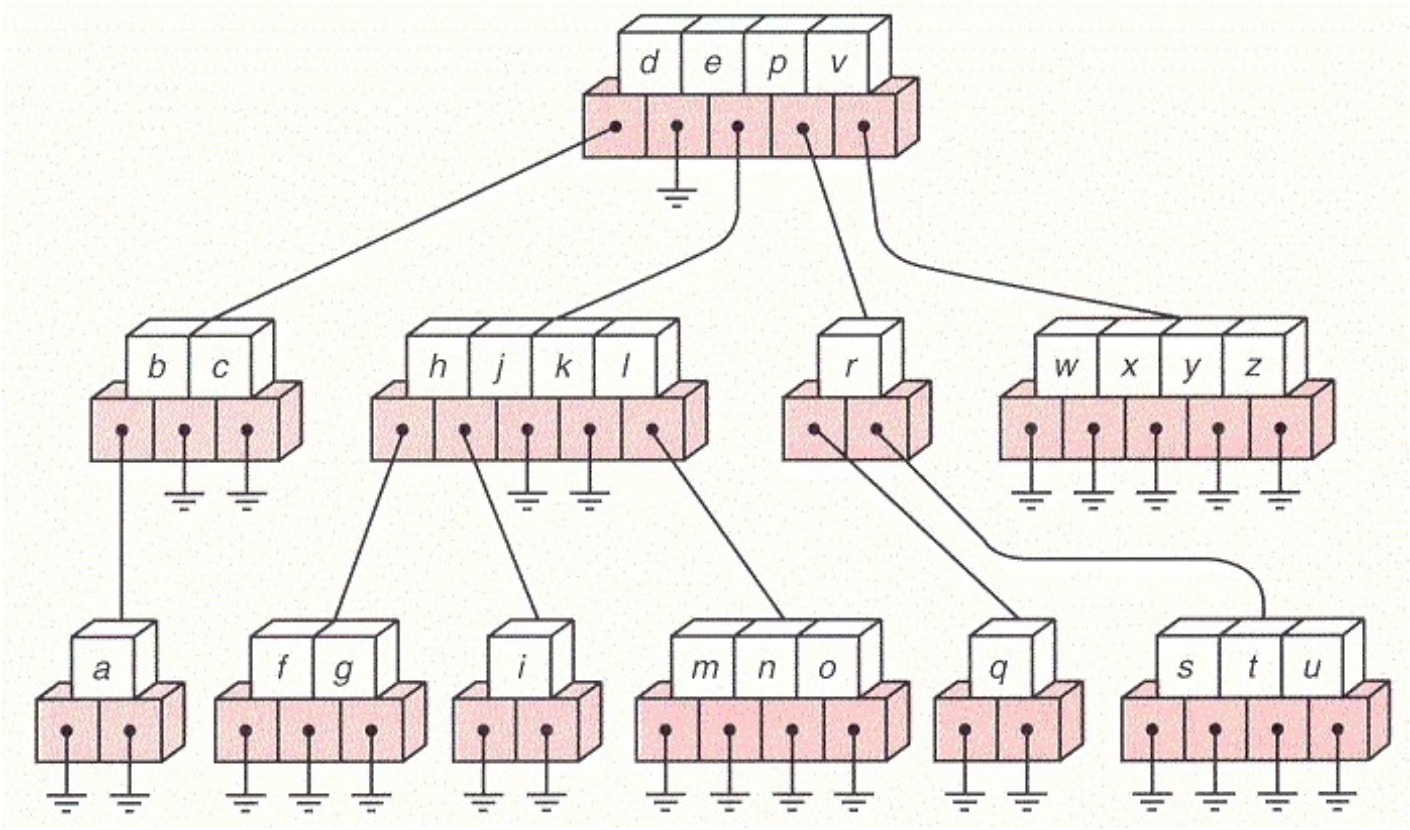
  $$K_1 \leq K_2 \leq \ldots \leq K_{p-2} \leq K_{p-1}$$

- Subtrærne er sortert slik at:
  - Alle verdier i S_1 er mindre enn K_1
  - Alle verdier i S_i er større eller lik K_{i-1} og mindre enn K_i, i = 2, 3, 4, ..., $p - 1$
  - Alle verdier i S_p er større eller lik K_{p-1}


K: Key
p = 4
K1
K2
K3
S1
S2
S3
S4
Keys < K1
K1 ≤ Keys < K2
K2 ≤ Keys < K3
K3 ≤ Keys

Flerveis søketre av orden 5

Merk at nodene kan ha flere tomme subtrær



Søking i flerveis søketre

- Samme prinsipp som for binært søketre
- Følger en vei fra roten og nedover i treet
- I hver node sjekker vi om verdien vi søker er lik, mindre eller større enn de sorterte nøkkelverdiene i noden
- Går videre fra en node til det subtreet der verdien skal ligge, hvis den finnes i treet

Effektivitet av flerveis søketre

- Søking, innsetting og fjerning har arbeids-mengde proporsjonal med *høyden* av treet
- Høyden av et flerveis søketre av orden *m* avhenger av:
  - Om treet er balansert
  - Hvor *fulle* nodene er, dvs. hvor mange verdier som er lagret i hver node (maksimalt *m* - 1)
- Hvis treet er balansert og nodene er godt fylt opp, er søking og innsetting meget raskt

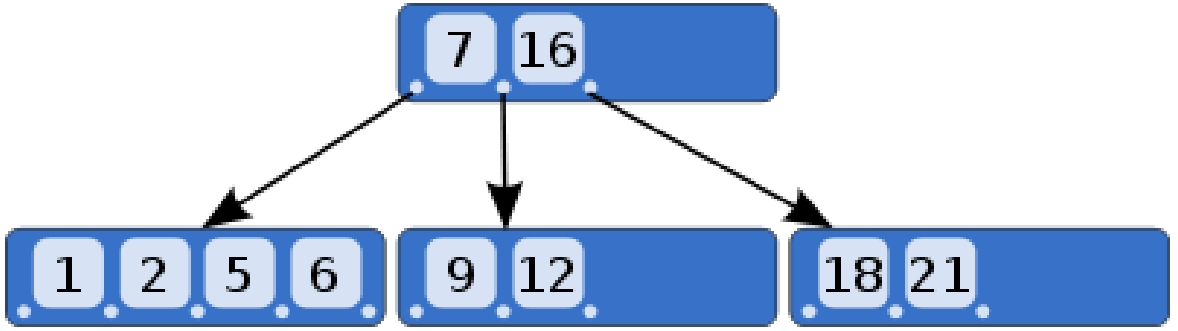
B-tre*

- Et flerveis, selv-balanserende søketre
- Bayer / McCreight: "Organization and Maintenance of Large Ordered Indexes", *Acta Informatica* (1972)
- Et B-tre er alltid *perfekt* balansert
- Nodene i treet er minst *halvveis* fylt opp med verdier
- B-trær av svært høy orden brukes ofte i database-systemer og filsystemer
- Garanterer $O(\log n)$ oppførsel for søking, innsetting og fjerning

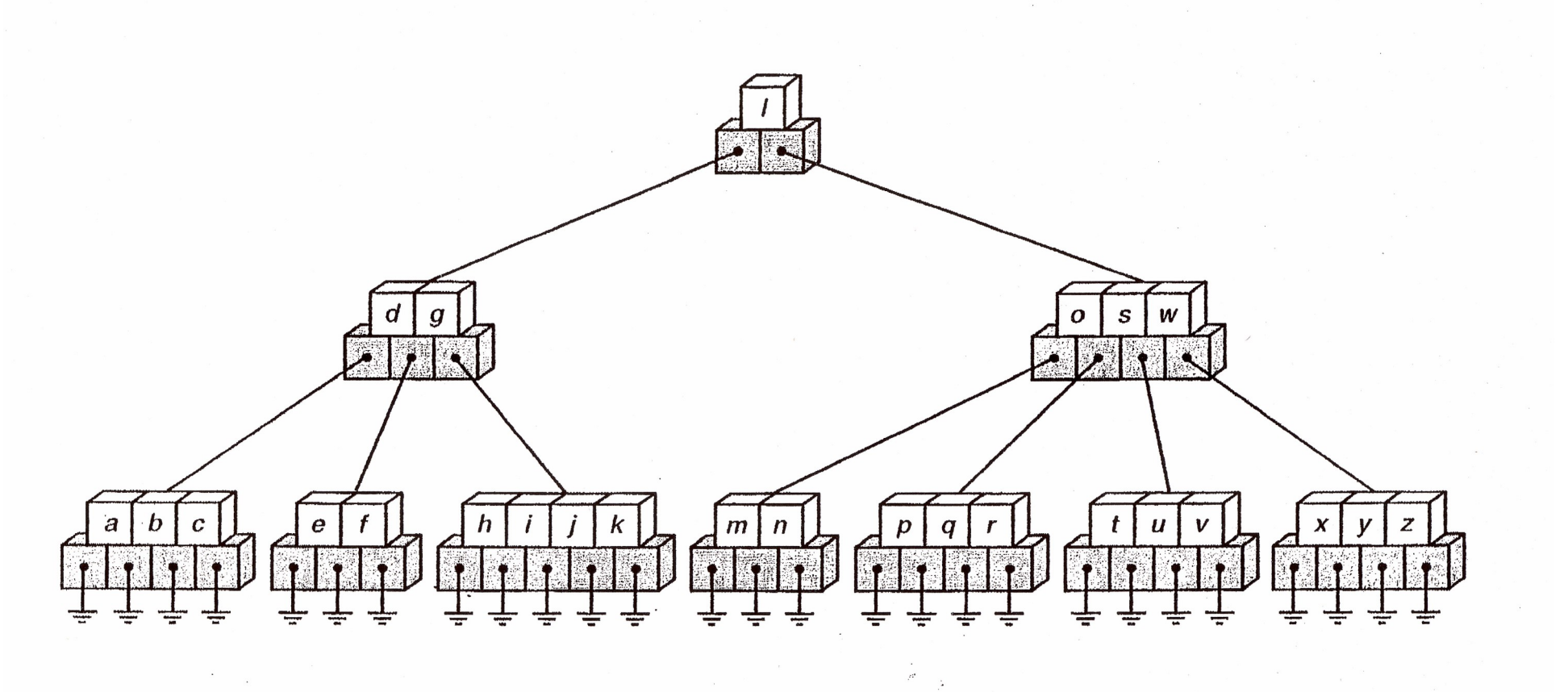
*: B?

Definisjon av B-tre

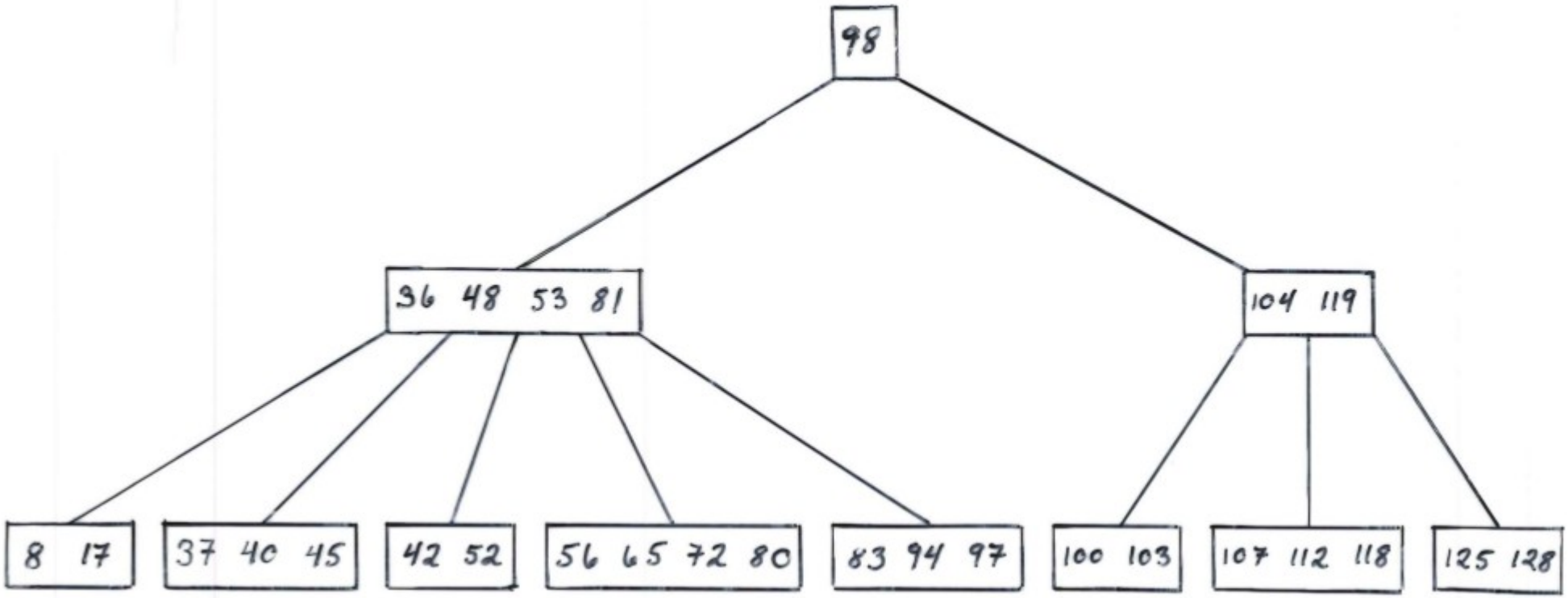
- Et B-tre av orden *m* er et flerveis søketre der:
  - Roten er enten et blad eller har minst to subtrær
  - Alle noder (unntatt roten) inneholder minst *m*/2 verdier og maksimalt *m* - 1 verdier
  - En node med *k* verdier har enten 0 barn (en bladnode) eller *k* + 1 barn (indre node)
  - Alle bladnoder ligger på samme nivå i treet



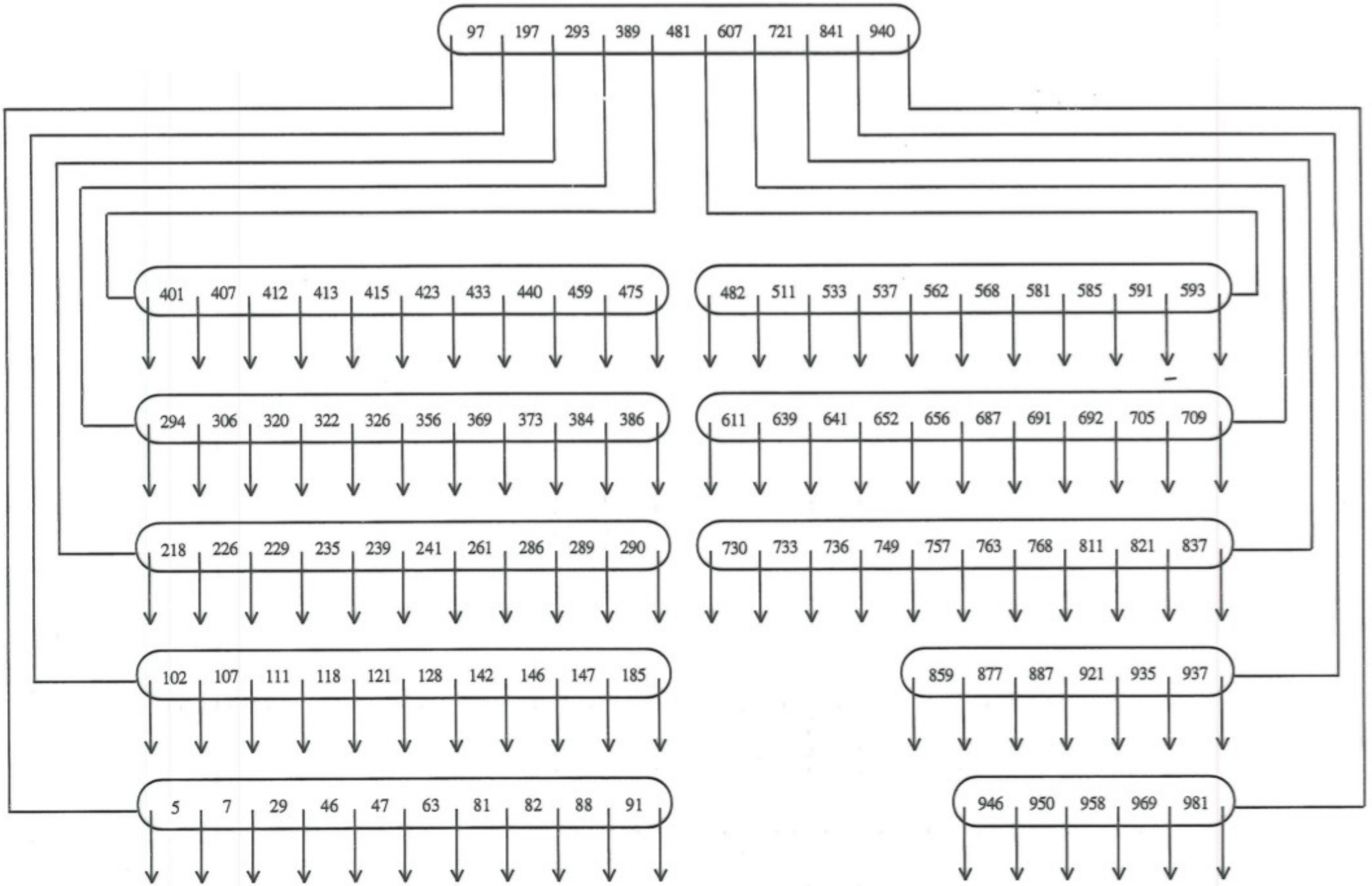
B-tre av orden 5



B-tre av orden 5



De to øverste nivåene i et B-tre av orden 11



Effektivitet, B-tre av orden 201

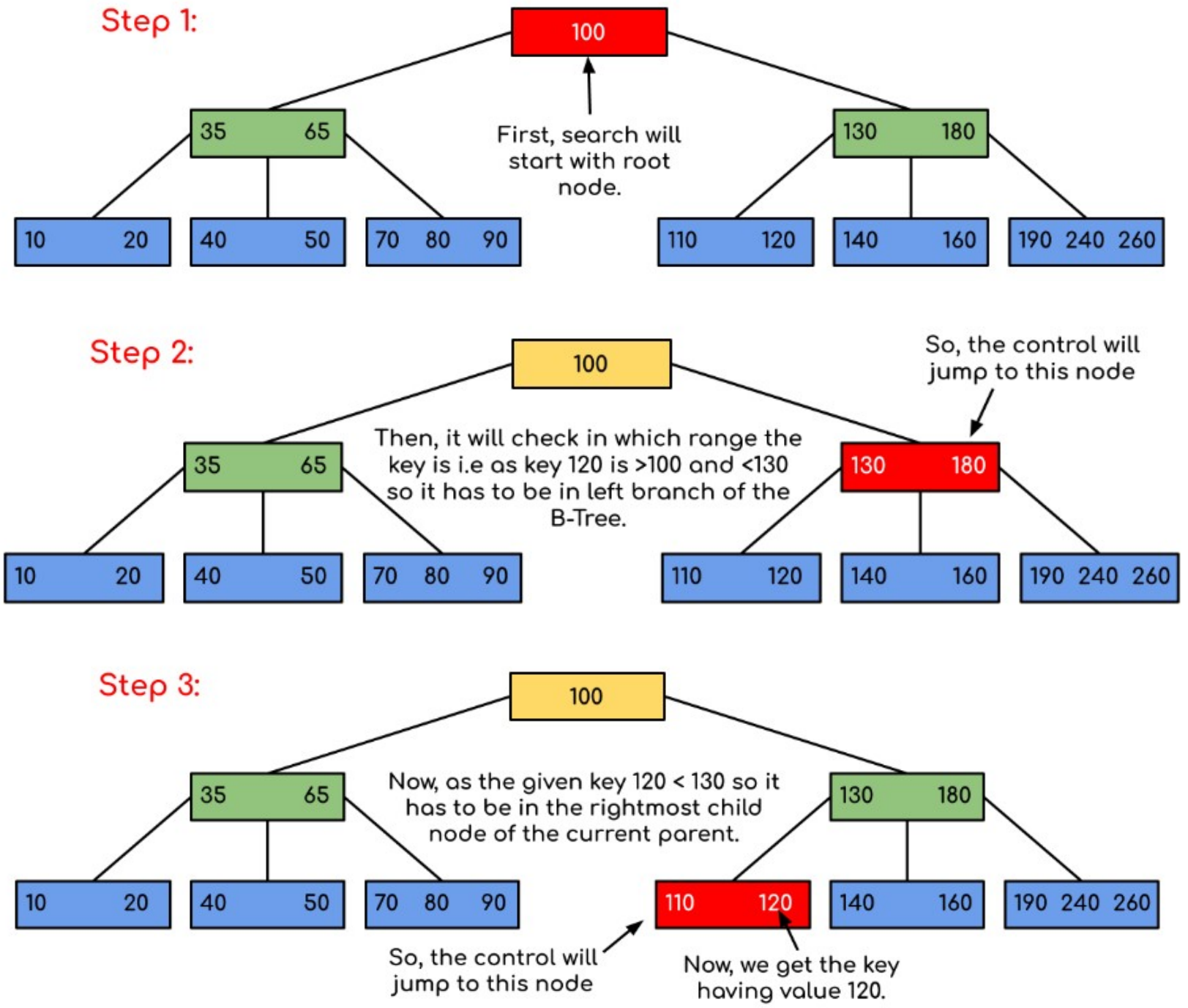
Antall noder og verdier i hvert nivå i treet:

Nivå	Min. antall noder	Min. antall verdier	Maks. antall noder	Maks. antall verdier
1	1	1	1	200
2	2	200	201	40 200
3	202	20 200	40 401	8 080 200
4	20.402	2.040.200	8.120.601	1.624.000.000

Søking i B-trær

- «An algorithm for finding a key in B-tree is simple. Start at the root and determine which pointer to follow based on a comparison between the search value and key fields in the root node. Follow the appropriate pointer to a child node. Examine the key fields in the child node and continue to follow the appropriate pointers until the search value is found or a leaf node is reached that doesn't contain the desired search value»

Søk etter verdien 120



Vekst av B-trær

- Kravet om at alle blader skal være på samme nivå gir en karakteristisk oppførsel for B-trær
- B-trær vokser ikke «nedover» ved at de får nye blader, som ved innsetting i binære søketrær
- Istedet vokser de «oppover» ved at de får ny rot hver gang antall nivåer i treet øker med 1

Innsetting av ny verdi i B-tre

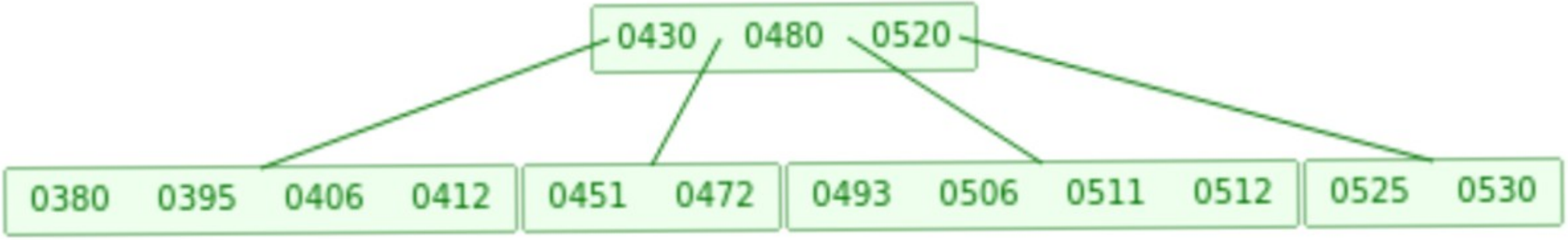
- En ny verdi settes inn i eksisterende bladnode
- Når en bladnode er full *deles* den i to, og en verdi sendes «oppover» i treet
- Demo av innsetting i B-tre av orden 3, med maksimalt 2 verdier og 3 barn per node
- Innsetting er alltid $O(\log n)$

Algoritme for innsetting i B-tre

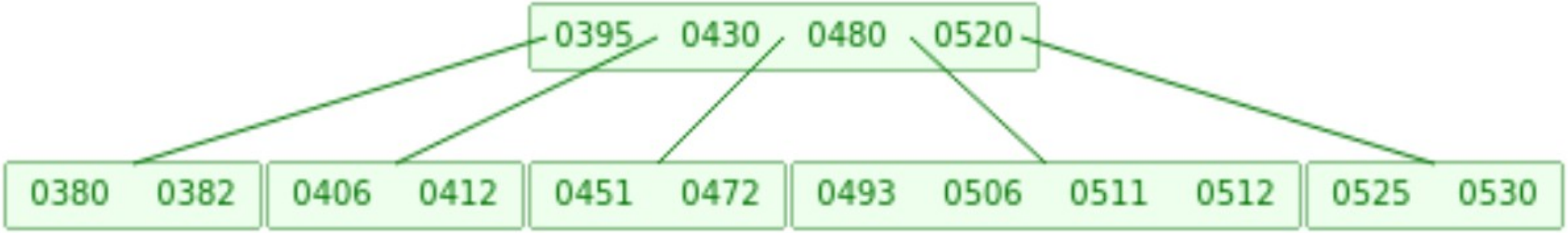
- Finn bladnoden der ny verdi skal settes inn
- Hvis bladnoden ikke er full:
  - Sett inn ny verdi sortert i bladnoden
- Ellers, hvis bladnoden er full av verdier:
  - Del bladnoden i to noder, med halvparten av verdiene i hver node
  - Send midterste verdi et nivå oppover for innsetting i foreldernoden
  - Hvis foreldernoden også blir full, fortsettes sending av verdier oppover i treet på samme måte

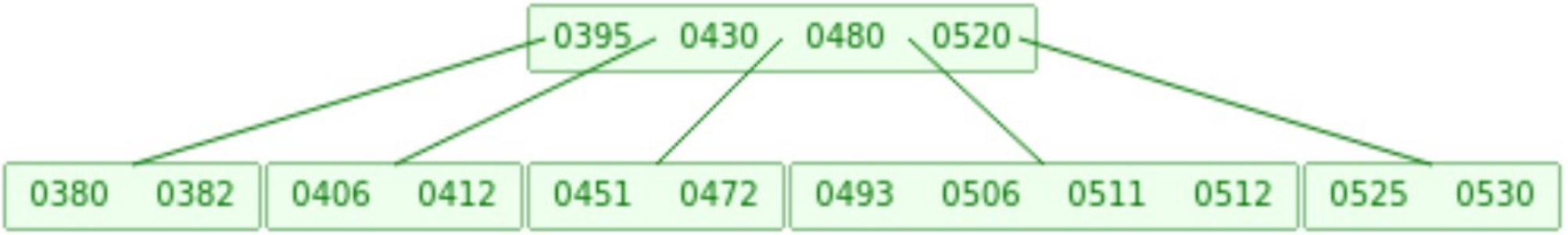
Eksempel:
Innsetting i B-tre av orden 5

Etter innsetting av 395

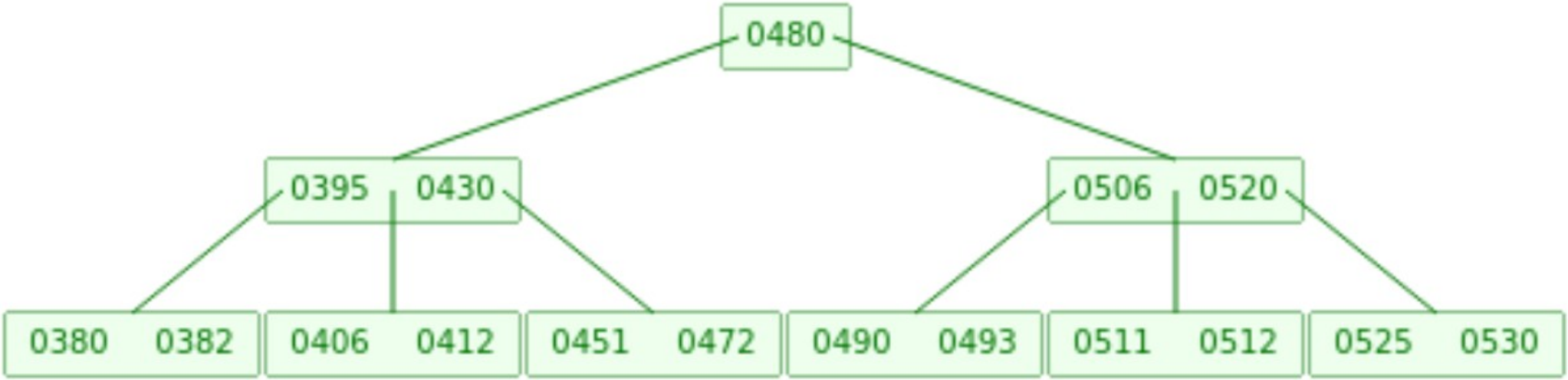


Etter innsetting av 382





Etter innsetting av 490



- Animasjon av innsetting, søking og fjerning i B-trær

Fjerning av verdi i B-tre

- Kan resultere i noder med for *få* verdier (< $m/2$)
- Vil lage et «hull» i strukturen hvis verdien som fjernes er i en indre node
- Algoritmen for fjerning «reparerer» treet med:
  - Flytting av verdier mellom noder (aka rotasjoner)
  - Sammenslåinger (merging) av to noder til én ny node
- Fjerning av verdier er vist i læreboka for B-trær med orden $m = 3$ (se avsnitt 14.2)
- Fjerning i B-trær er relativt «fiklete», med flere spesialtilfeller, men er alltid $O(\log n)$

Representasjon av B-tre med pekere

- En node i et B-tre av orden *m* kan representeres med et klasseobjekt med to arrayer:
  - En array av lengde *m* med pekere til barna
  - En sortert array av lengde *m* – 1 med verdier
- I tillegg må hver node holde rede på antall verdier som er lagret i noden
- Tilgang til hele treet gjennom en peker til roten

Representasjon av B-tre med array

- B-trær brukes ofte til å håndtere data som er lagret i sekundærminnet (typisk på disk)
- Fungerer i praksis dårlig med «pekere» (diskadresser) i noder som ligger spredt rundt i sekundærminnet
- Det er derfor vanlig i stedet å representere hele B-treet som én lang array:
  - Arrayen er delt opp et stort antall like lange segmenter
  - Hvert segment lagrer data for én node
  - «Pekerne» til barna er bare indekser i arrayen

B-trær og sekundærminne

- Hvis datamengdene er for store til å få plass i RAM, må de flyttes frem og tilbake mellom RAM og disk (paging)
- Lesing og skriving til/fra disk/sekundærminne er flere tusen ganger tregere enn å lese/skrive RAM
- Disker leses og skrives derfor i store «datablokker» for å øke effektiviteten, typisk blokkstørrelse er 4 KBytes
- B-trær lages slik at antall diskaksesser minimeres:
  - Størrelsen på en node tilpasses blokkstørrelsen i filsystemet, slik at alle dataene i noden kan håndteres med bare én read/write-operasjon

To vanlige varianter av B-trær

- B^* - trær:
  - Utnytter plassen på hvert nivå i treet bedre
- B^+ - trær:
  - Gjør traversering av treet mer effektivt

B* - trær

- B*-treet har alle egenskapene til et B-tre, men:
  - I et B-tre av orden *m* har alle noder minst *m*/2 verdier
  - I et B*-tre av orden *m* har alle noder minst 2·*m*/3 verdier
  - Flere verdier i hver node betyr færre nivåer i treet
- Algoritmene for innsetting og fjerning er forskjellige:
  - I stedet for å dele en node i to med en gang den er full, flyttes verdier over til søskennoder
  - Når to søskennoder er fulle av verdier, deles de i tre nye noder

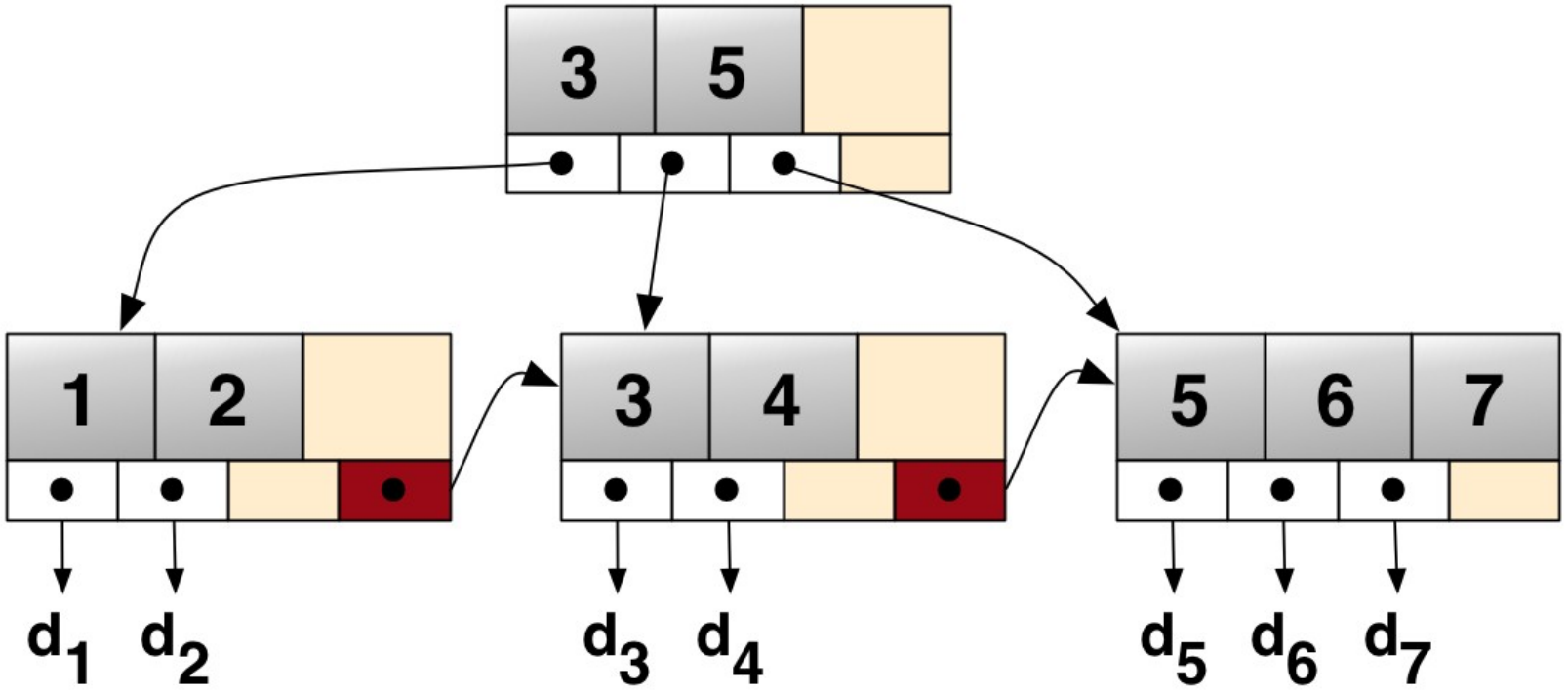
Traversering av B-trær

- En sortert gjennomgang av alle dataene lagret i et B-tre kan gjøres med inorder traversering
- Hver node må oppsøkes opptil *m* + 1 ganger i løpet av en traversering av B-treet
- Hvis deler av B-treet ligger utenfor RAM, blir dette svært lite effektivt, med mye lesing av data fra disk
- B^+ - trær løser dette problemet

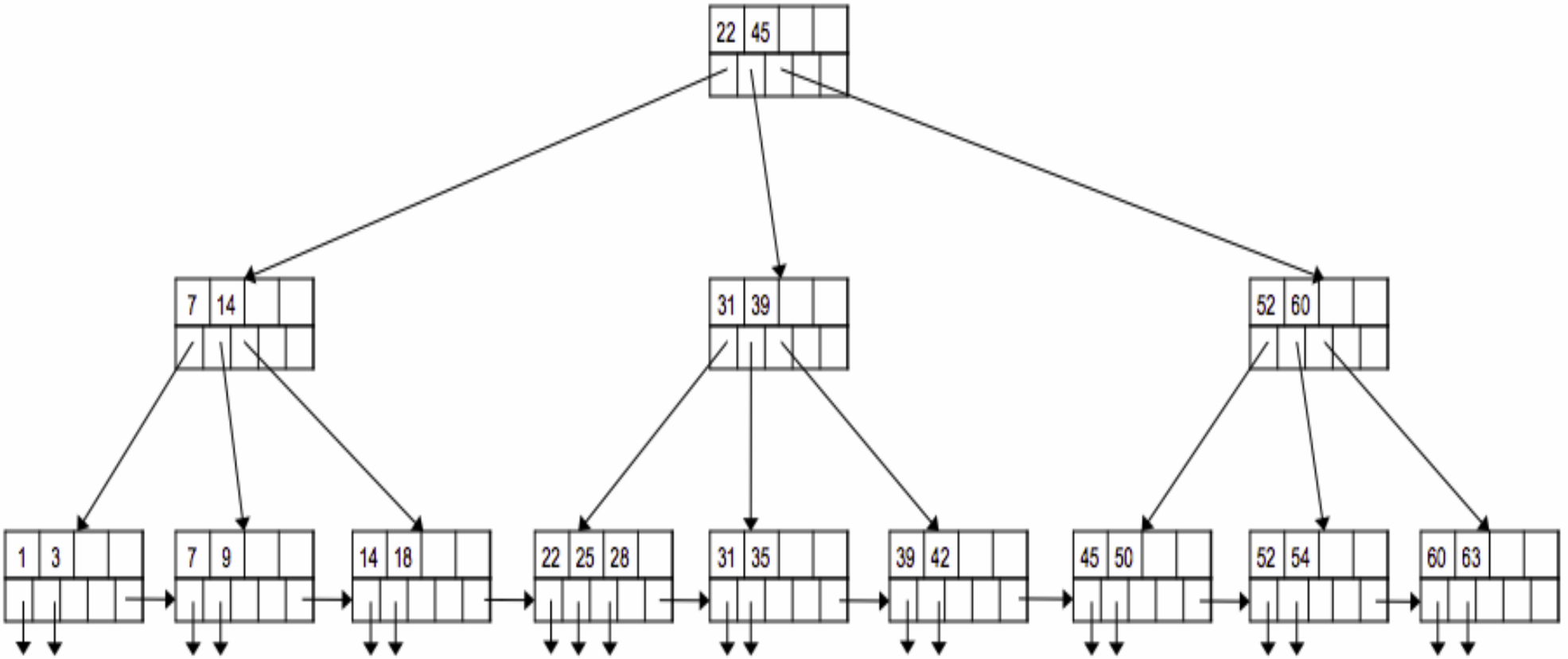
B^+ - trær

- Alle nøkkelverdier i indre noder lagres *en gang til* som sin *egen inorder etterfølger* i en bladnode
- Alle bladnodene lenkes sammen med «pekere» for å kunne gå gjennom alle dataene sortert med minimalt antall diskaksesser
- Lagrer *bare* nøkkelverdier inne i treet
- Bruker et *ekstra lag* med noder på nederste nivå som inneholder pekere til selve dataobjektene
- Animasjon av innsetting, fjerning og søk i B+-trær

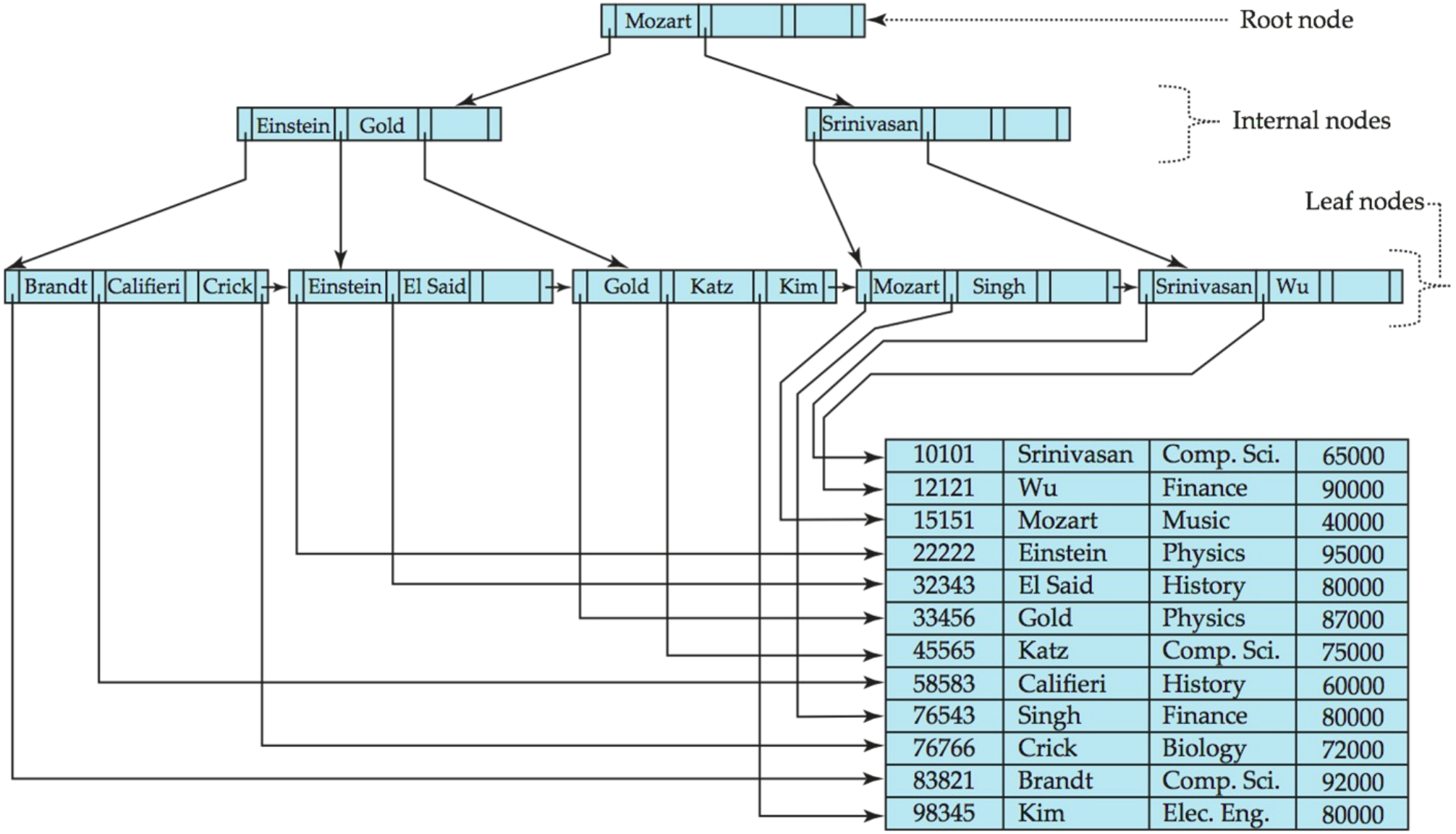
Eksempel: B^+ - tre



Eksempel: B^+- tre



Eksempel: B^+- tre



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000