Forelesning 2

For å unngå grunnleggende kjøretidsfeller er det viktig å kunne organisere og strukturere data fornuftig. Her skal vi se på hvordan enkle strukturer kan implementeres i praksis, og hva vi vinner på å bruke dem i algoritmene våre.

Pensum

- ☐ Kap. 10. Elementary data structures: Innledning og 10.1–10.3
- ☐ Kap. 11. Hash tables: s. 253–264
- ☐ Kap. 17. Amortized analysis: Innledning og s. 463–465 (tom. «at most 3»)

Læringsmål

- [B1] Forstå *stakker* og *køer*
- [B2] Forstå *lenkede lister*
- [B3] Forstå implementasjon av *pekere* og *objekter*
- [B4] Forstå *direkte adressering* og *hashtabeller*
- [B5] Forstå *konfliktløsing ved kjeding*
- [B6] Kjenne enkle *hashfunksjoner*
- [B7] Vite at man for *statiske datasett* kan ha *worst-case* O(1) for søk
- [B8] Kunne definere *amortisert analyse*
- [B9] Forstå *dynamiske tabeller*

Forelesningen filmes

Forelesning 2

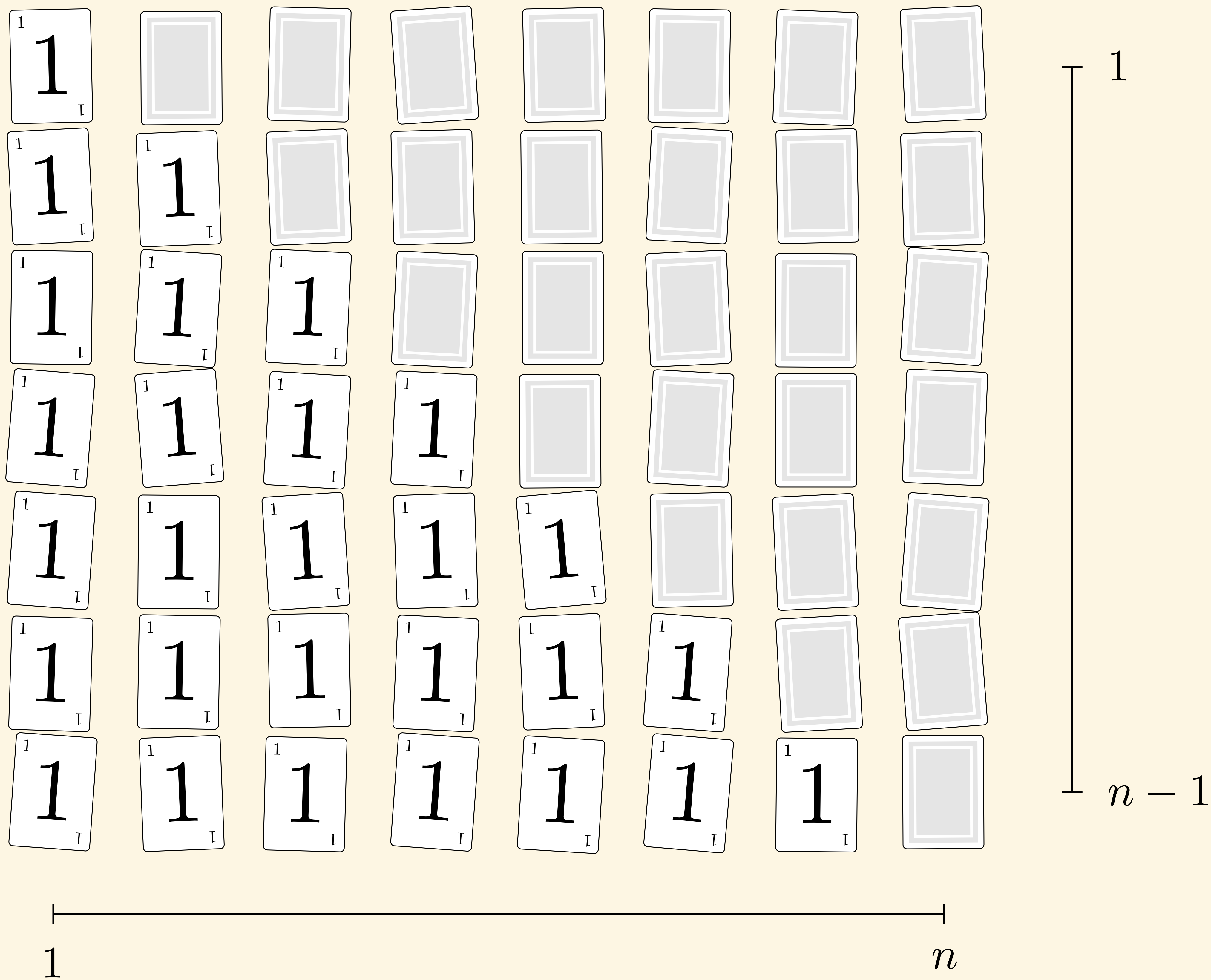
Datastrukturer

0:4

To kritiske summer

$$1 + 2 + 3 + 4 + 5$$

1
$+2$
$+3$
$+4$
$\vdots$
$+n-2$
$+n-1$



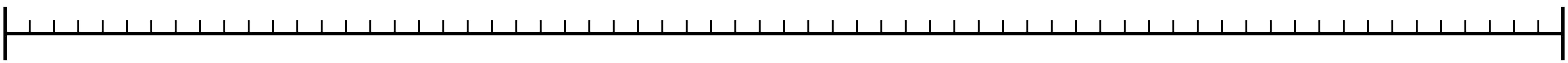
$$\sum_{i=0}^{n-1} i = \frac{n \cdot (n-1)}{2}$$

$$\sum_{i=0}^{n-1} i = \frac{n \cdot (n-1)}{2}$$

antall × gjennomsnitt av første og siste

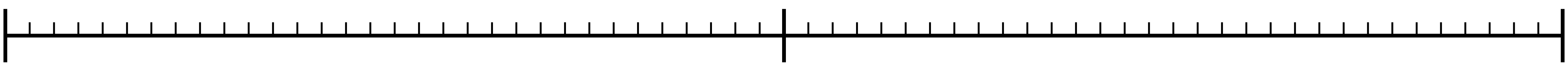
$$1 + 2 + 4 + 8 + 1$$

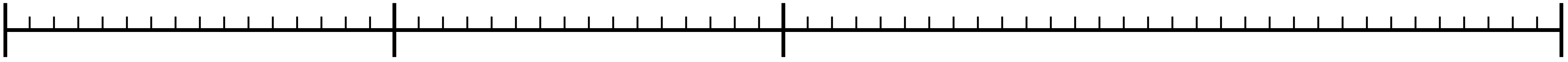
Ett av Xeno sine berømte paradokser: Du kan aldri komme deg fra A til B, fordi først må du komme halvveis, og før det, så må du komme halvveis til halveien, etc.

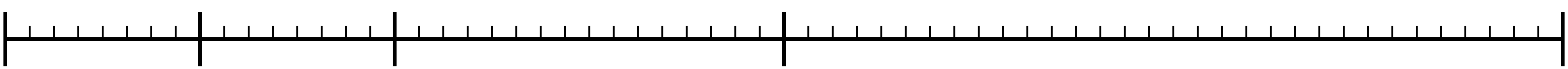


Hvis vi bruker reelle tall og ser på grenseverdien av n/2 + n/4 + n/8 + … så blir jo svaret bare n. Men hva om vi har begrenset oppløsning, og bare kan bruke heltall?

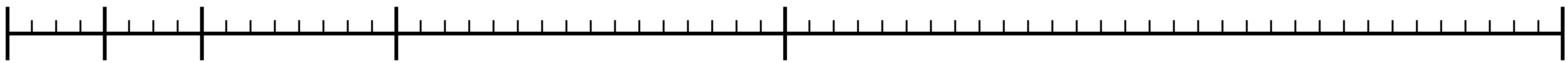
Vi kan anta at n = 2^h for et eller annet ikke-negativt heltall h.

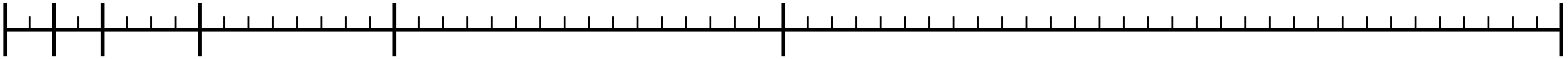


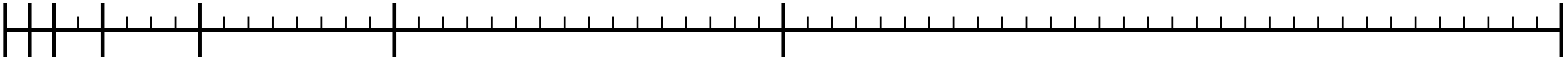


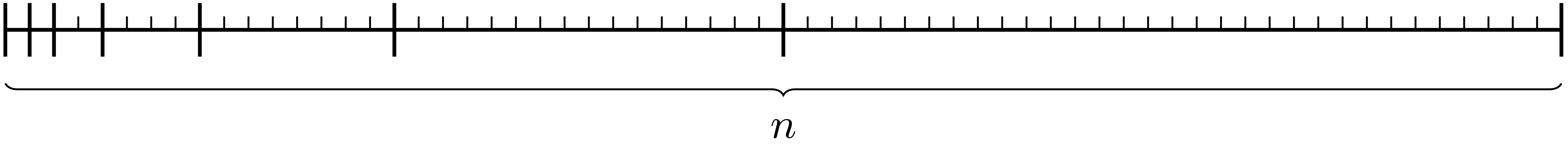


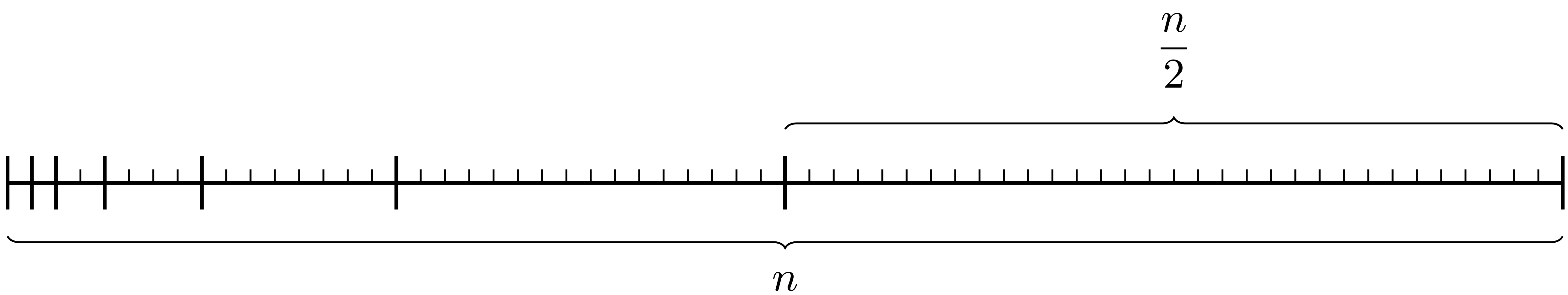
mlh@ntnu.no

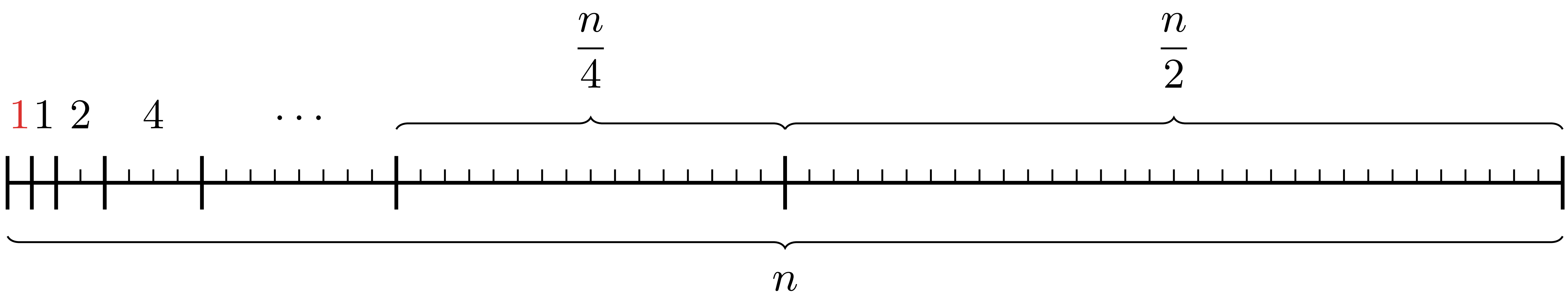




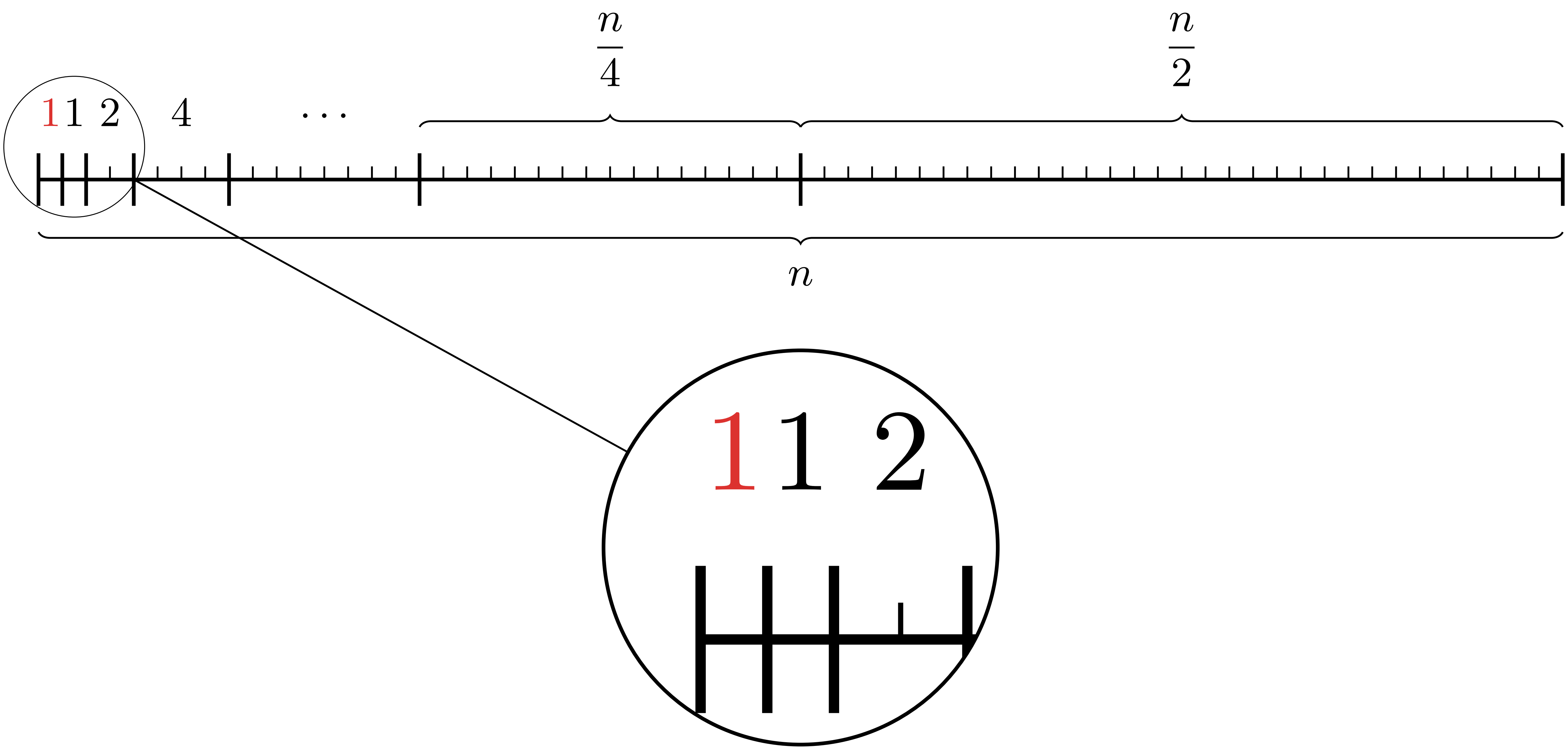



n


$\frac{n}{2}$
n



1 1 2 4 ...
n/4
n/2
n



1 1 2
4
$\cdots$
$\frac{n}{4}$
$\frac{n}{2}$
n
1 1 2

$$\sum_{i=0}^{h-1} 2^i = 2^h - 1$$

Alternativ huskeregel 1: Utslagsturnering med n deltakere.

I runde 1 trenger du n/2 matcher. I runde 2 trenger du n/4, etc., helt til du til slutt har én match mellom de to siste. Antall matcher er 1 + 2 + 4 + … + n/2.

Og hvor mange matcher er det? I hver match går én deltaker ut av turneringen, helt til du sitter igjen med bare vinneren, som aldri blir slått ut. Altså n – 1 matcher.

Med andre ord: 1 + 2 + 4 + … + n/2 = n–1.

$$\sum_{i=0}^{h-1} 2^i = 2^h - 1$$

Alternativ huskeregel 2: Tenk på et tall i totallssystemet som bare består av ettall. Dvs., 11111111…111. Hvilken verdi har dette tallet?

Vel, hvis det har h siffer, er det summen av de h første toerpotensene, 1 + 2 + 4 + … + 2^(h-1).

Men hva skjer om du legger til 1? Da får du plutselig 100000000…000 – ett siffer ekstra, og tallet er lik neste toerpotens. Så 11111111…111 er én mindre enn neste toerpotens.

Med andre ord: 1 + 2 + 5 + … + 2^(h-1) = 2^h – 1.

$$1 + 2 + 4 + \cdots + \frac{n}{4} + \frac{n}{2} = n - 1$$

1. **Stakker og køer**
2. **Lenkede lister**
3. **Hashtabeller**
4. **Dynamiske tabeller**

10.3: Selvstudium

1:4

Stakker

Kun adgang øverst

Stakk vs stack. Stakk som i «høystakk» – mer passende navn hadde kanskje vært «stabel».

STACK-EMPTY(S)

Er stakken tom?

Stack-Empty(S)
1 **if** S.top == 0

top er øverste indeks, dvs. antall elementer

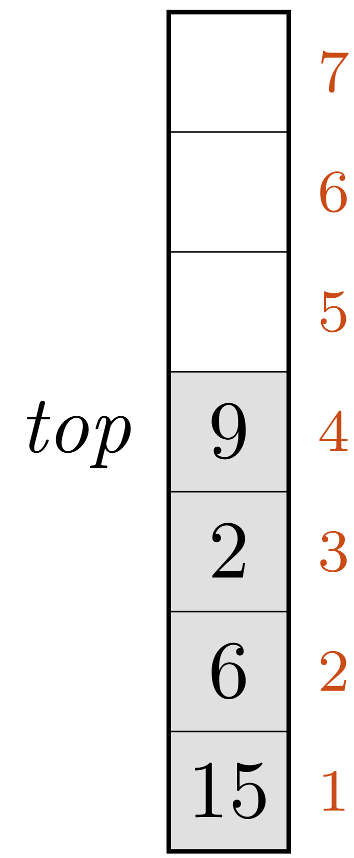
```
Stack-Empty(S)
    if S.top == 0
        return TRUE
```

Ingen elementer, så stakken er tom

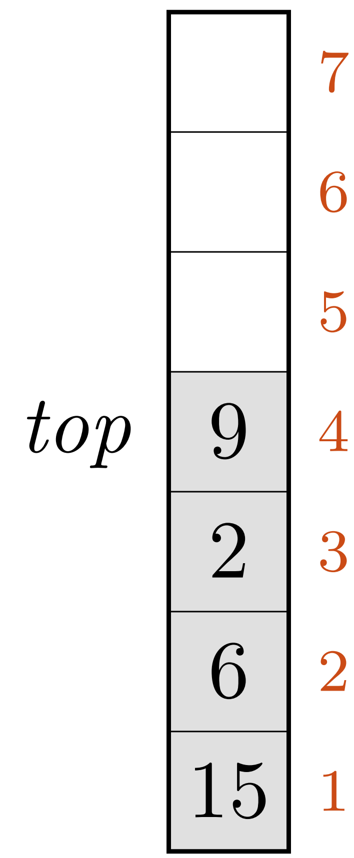
STACK-EMPTY(S)
if S.*top* == 0
    return TRUE
else return FALSE

Vi har elementer, så stakken er ikke tom

```
STACK-EMPTY(S)
if S.top == 0
        return TRUE
else return FALSE
```



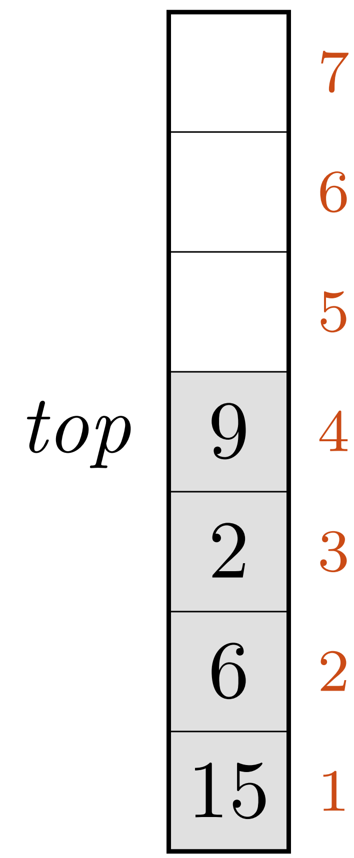
```
STACK-EMPTY(S)
if S.top == 0
        return TRUE
else return FALSE
```



STACK-EMPTY(S)

```
if S.top == 0
        return TRUE
else return FALSE
```

→ FALSE



$\textsc{Push}(\mathrm{S}, x)$

Legg et element på toppen

$\textsc{Push}(\mathrm{S}, x)$

$\mathrm{S}.top = \mathrm{S}.top + 1$

Øk antallet med én

$\textsc{Push}(\mathrm{S}, x)$

1 $\mathrm{S}.top = \mathrm{S}.top + 1$

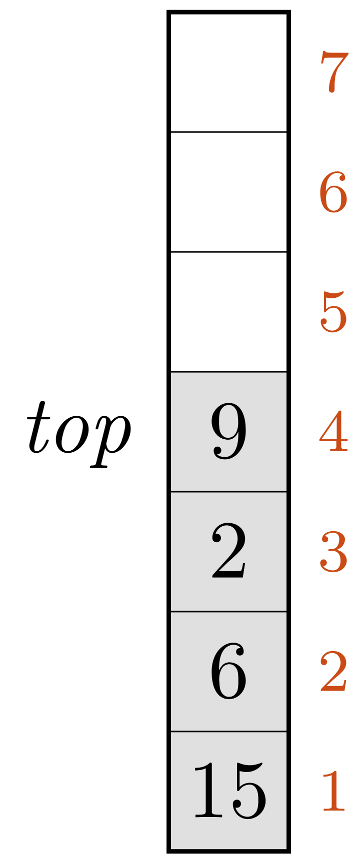
2 $\mathrm{S}[\mathrm{S}.top] = x$

Øverste indeks refererer nå til x

PUSH(S, x)

$S.top = S.top + 1$

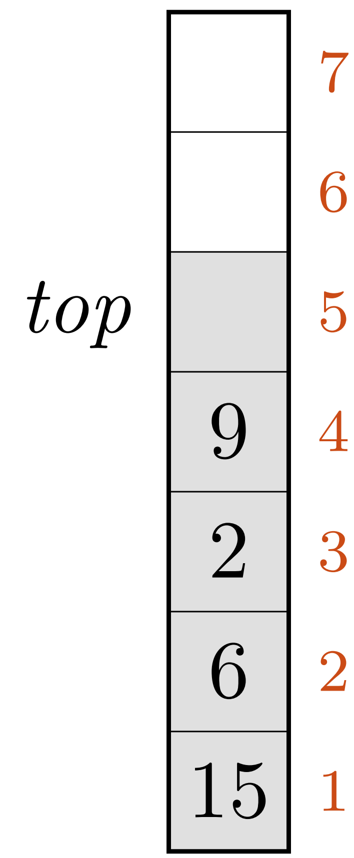
$S[S.top] = x$



$x = 17$

```
PUSH(S, x)
S.top = S.top + 1
S[S.top] = x
```

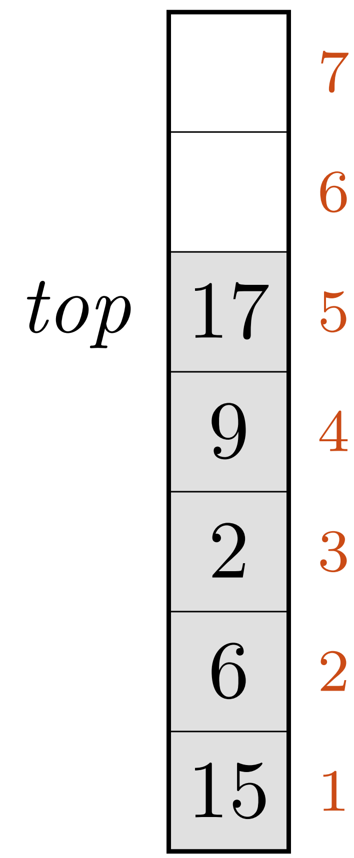
$x = 17$



PUSH(S, x)

$\mathrm{S}.top = \mathrm{S}.top + 1$

$\mathrm{S}[\mathrm{S}.top] = x$

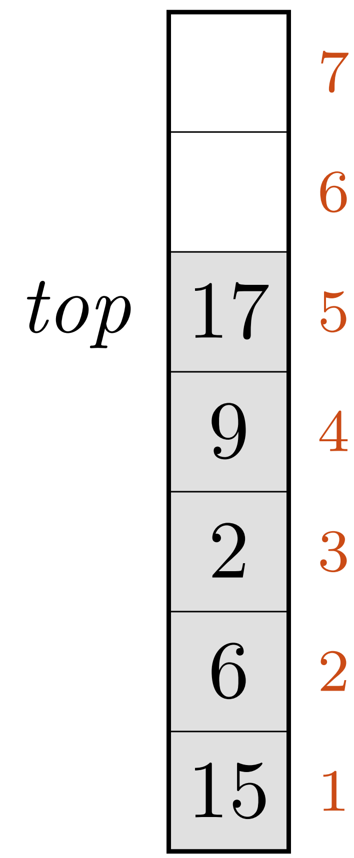


$x \;=\; 17$

$\textsc{Push}(S, x)$

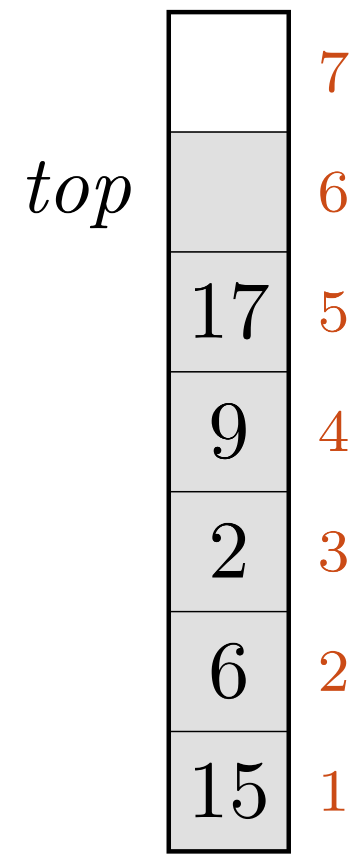
$S.top = S.top + 1$

$S[S.top] = x$



$x = 3$

Push(S, x)
$S.top = S.top + 1$
$S[S.top] = x$

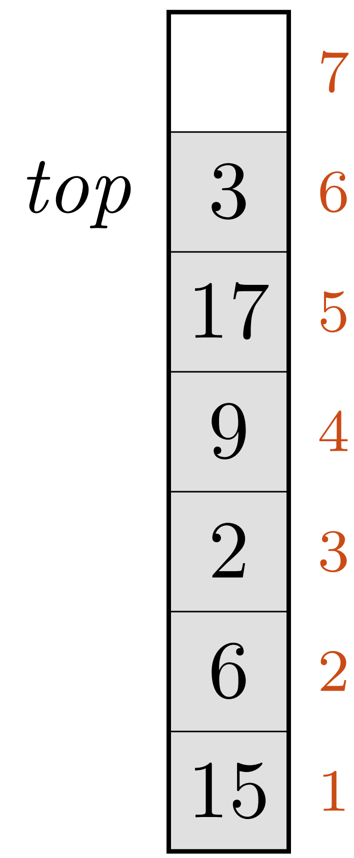


$x \;=\; 3$

Push(S, x)

$\text{S}.top = \text{S}.top + 1$

$\text{S}[\text{S}.top] = x$



$x = 3$

Pop(S)

Fjern ett element fra toppen

```
Pop(S)
if Stack-Empty(S)
```

```
POP(S)
if STACK-EMPTY(S)
    error "underflow"
```

Kan ikke fjerne noe fra en tom stakk

```
Pop(S)
if Stack-Empty(S)
    error "underflow"
else S.top = S.top − 1
```

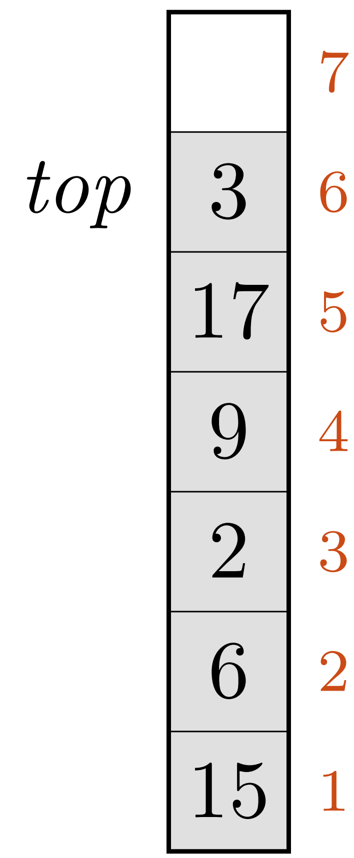
Redusér antallet med én

mlh@ntnu.no

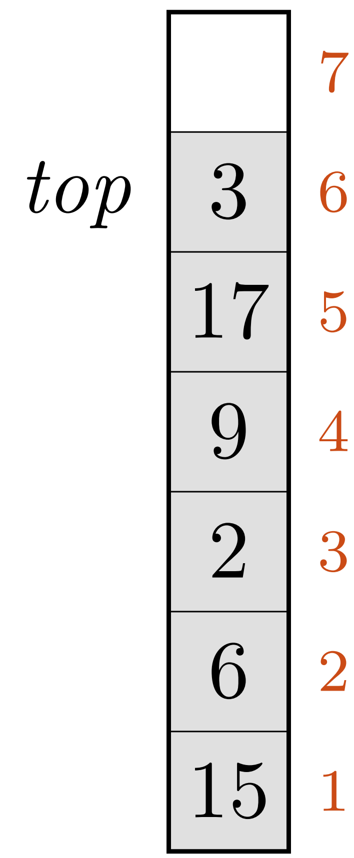
```
POP(S)
if STACK-EMPTY(S)
        error "underflow"
else S.top = S.top − 1
        return S[S.top + 1]
```

Returnér den gamle toppen (nå utenfor stakken)

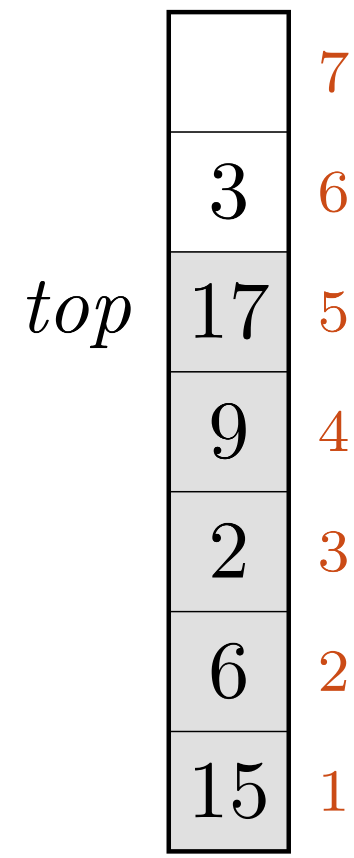
```
Pop(S)
if Stack-Empty(S)
        error "underflow"
else S.top = S.top − 1
        return S[S.top + 1]
```



```
Pop(S)
if Stack-Empty(S)
        error "underflow"
else S.top = S.top − 1
        return S[S.top + 1]
```

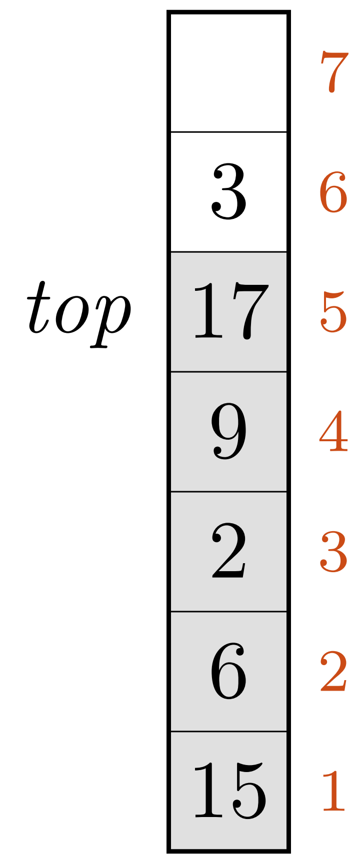


```
Pop(S)
if Stack-Empty(S)
        error "underflow"
else S.top = S.top − 1
        return S[S.top + 1]
```



Pop(S)
1 **if** Stack-Empty(S)
2 **error** "underflow"
3 **else** S.top = S.top − 1
4 **return** S[S.top + 1]

→ 3



Køer

Først inn, først ut

Kan implementeres på flere måter (inkl. lenkede lister, f.eks.); her bruker vi et såkalt «ringbuffer».

$\textsc{Enqueue}(Q, x)$

Legg til et element bakerst i køen

$\textsc{Enqueue}(\mathrm{Q}, x)$
1 $\mathrm{Q}[\mathrm{Q}.tail] = x$

Q[1] er først i køen og Q[*tail*] er den ledige plassen bak

$\textsc{Enqueue}(Q, x)$

$Q[Q.tail] = x$

if $Q.tail == Q.length$

Vi må forskyve *tail*, men har kommet til slutten!

$\textsc{Enqueue}(\mathrm{Q}, x)$
$\mathrm{Q}[\mathrm{Q}.tail] = x$
if $\mathrm{Q}.tail == \mathrm{Q}.length$
    $\mathrm{Q}.tail = 1$

Tabellen «går i ring», så neste posisjon er 1

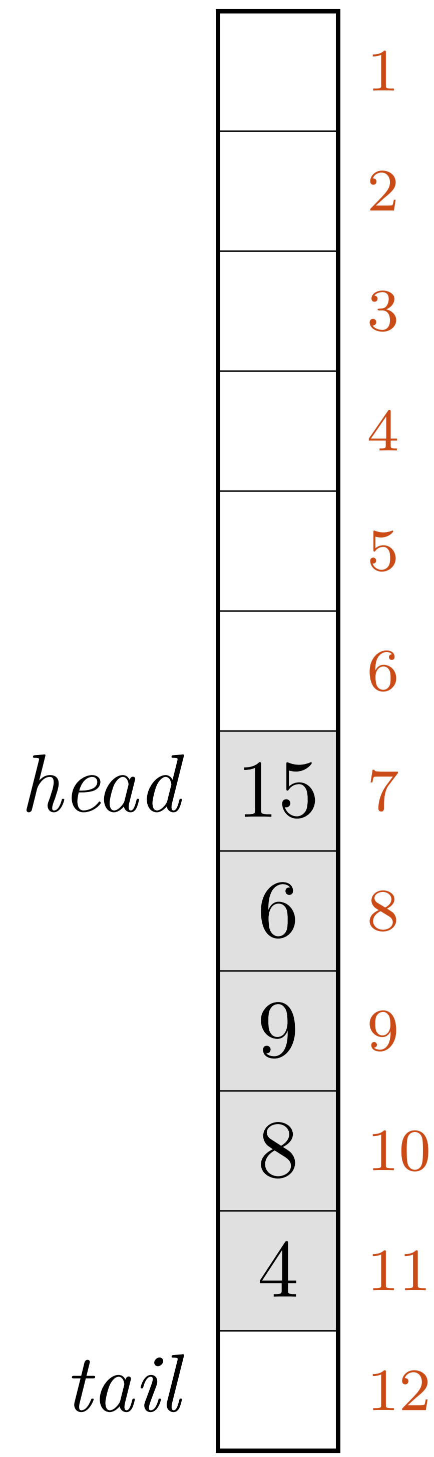
```
ENQUEUE(Q, x)
Q[Q.tail] = x
if Q.tail == Q.length
    Q.tail = 1
else Q.tail = Q.tail + 1
```

Har vi ledig plass, er det jo ikke noe problem

mlh@ntnu.no

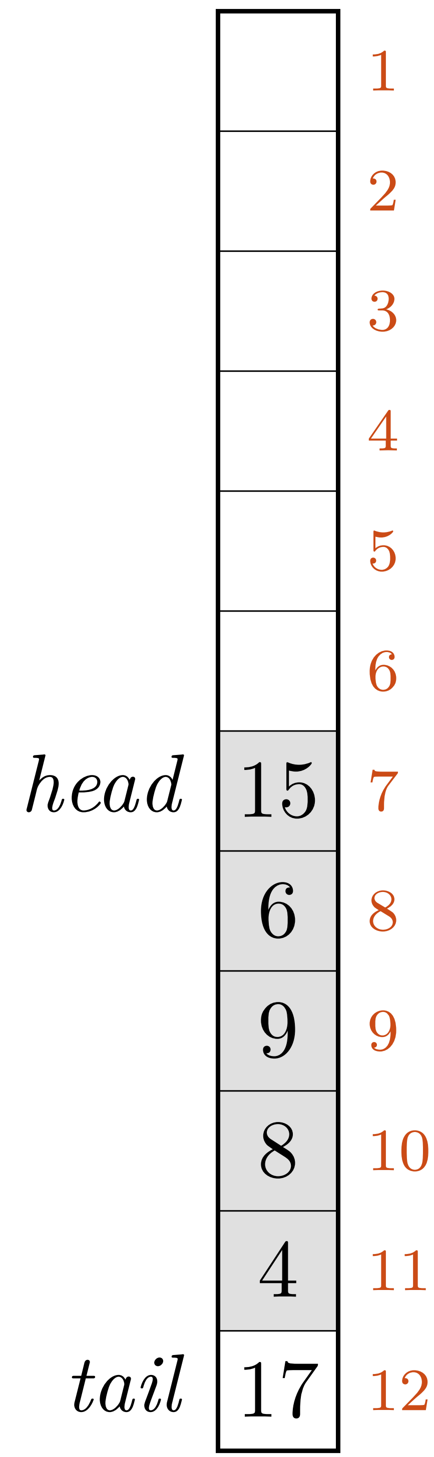
```
ENQUEUE(Q, x)
    Q[Q.tail] = x
    if Q.tail == Q.length
        Q.tail = 1
    else Q.tail = Q.tail + 1
```

$x = 17$

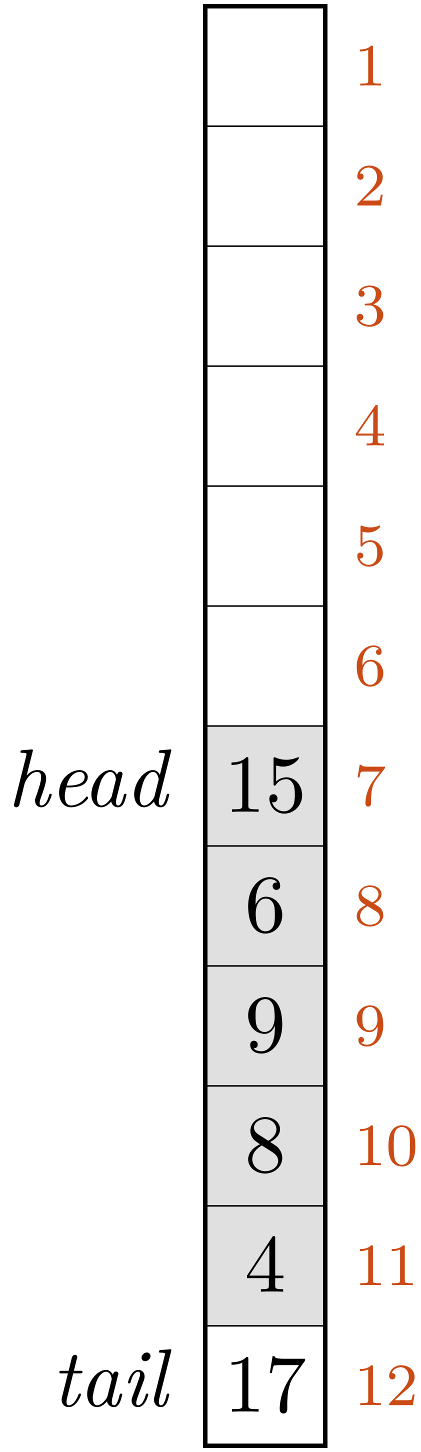


```
ENQUEUE(Q, x)
  Q[Q.tail] = x
  if Q.tail == Q.length
          Q.tail = 1
  else Q.tail = Q.tail + 1
```

$x = 17$



```
ENQUEUE(Q, x)
Q[Q.tail] = x
if Q.tail == Q.length
        Q.tail = 1
else Q.tail = Q.tail + 1
```

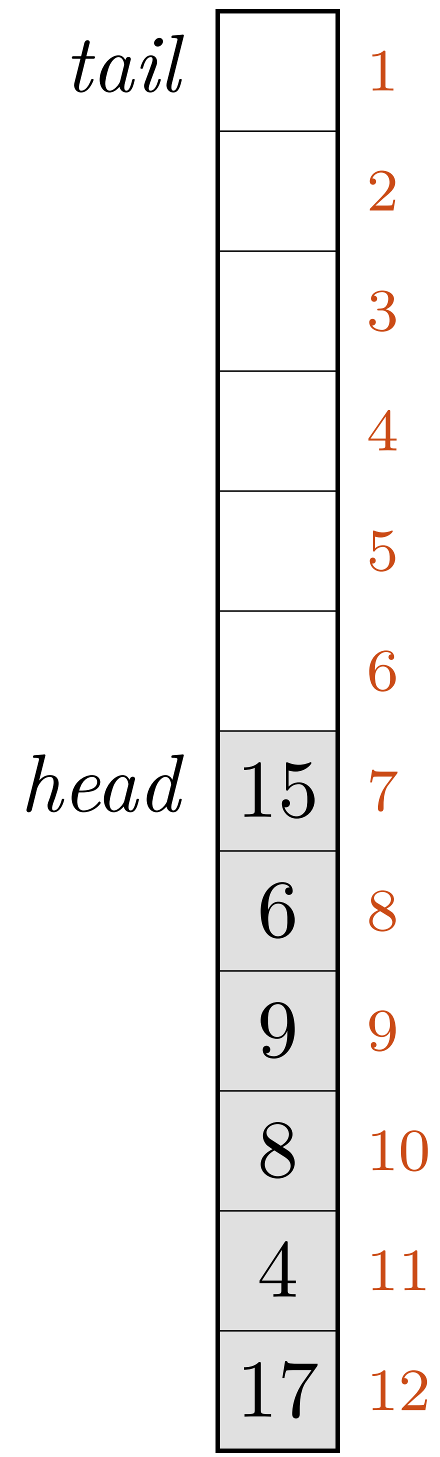


$x = 17$

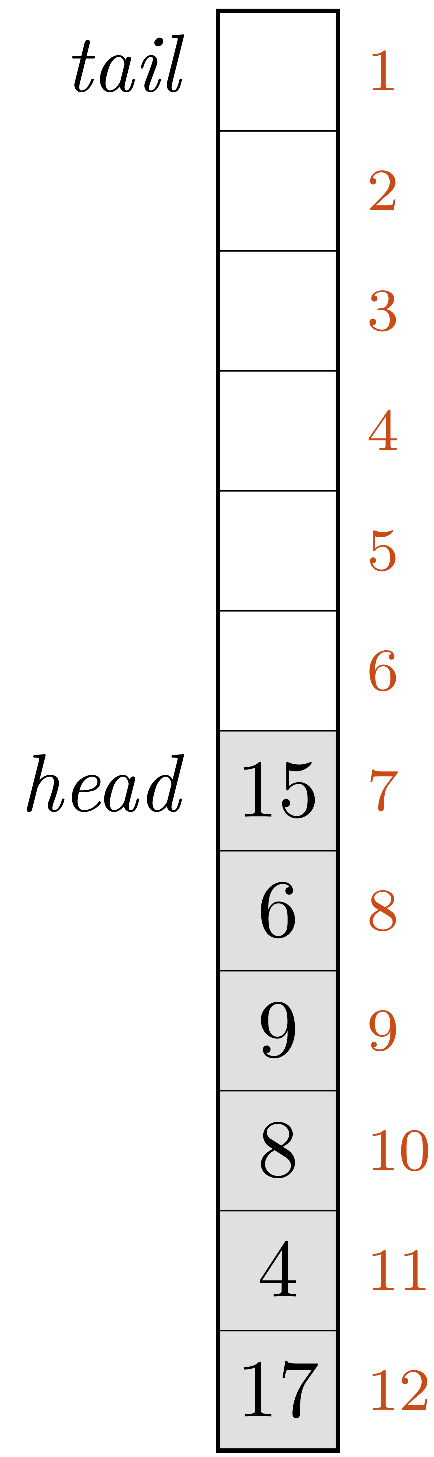
Enqueue(Q, x)

$Q[Q.tail] = x$
if $Q.tail == Q.length$
    $Q.tail = 1$
else $Q.tail = Q.tail + 1$

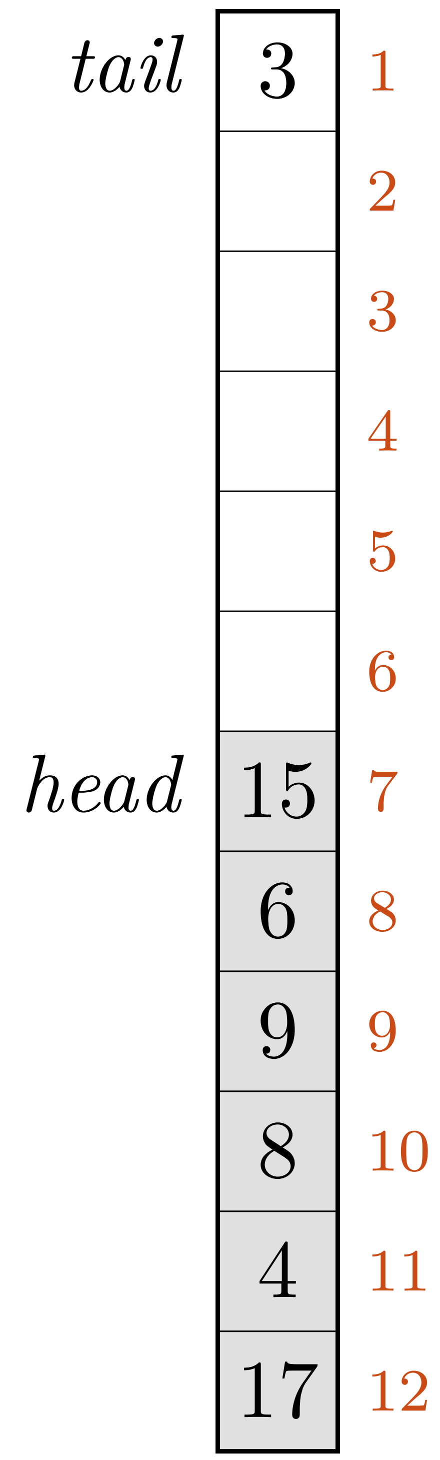
$x = 17$



```
ENQUEUE(Q, x)
Q[Q.tail] = x
if Q.tail == Q.length
        Q.tail = 1
else Q.tail = Q.tail + 1
```



$x = 3$

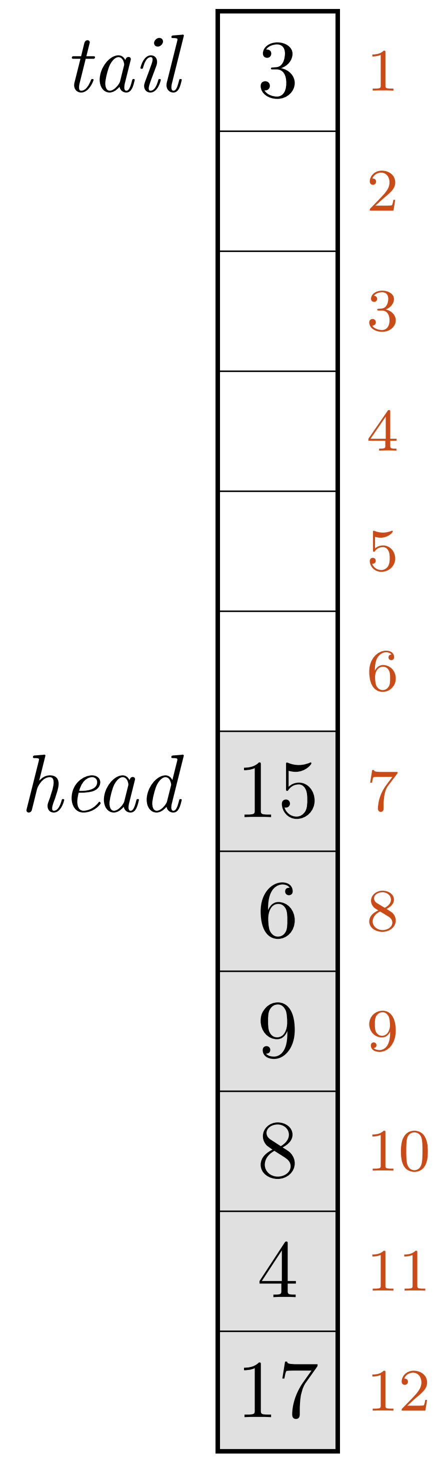



```
ENQUEUE(Q, x)
Q[Q.tail] = x
if Q.tail == Q.length
        Q.tail = 1
else Q.tail = Q.tail + 1
```

$x = 3$

ENQUEUE(Q, x)

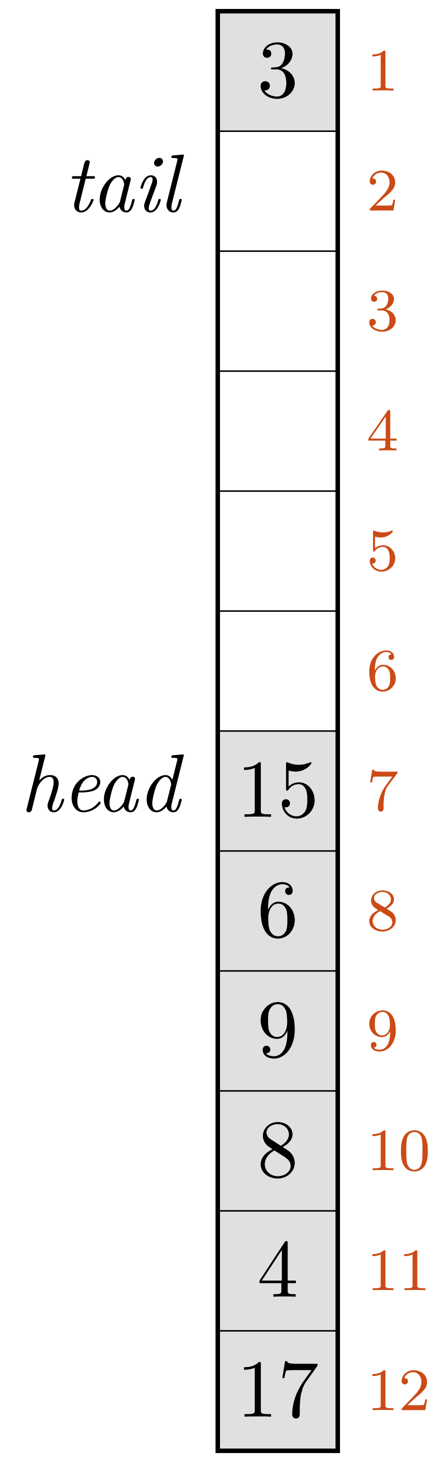
```
Q[Q.tail] = x
if Q.tail == Q.length
      Q.tail = 1
else Q.tail = Q.tail + 1
```



$x = 3$

ENQUEUE(Q, x)

```
Q[Q.tail] = x
if Q.tail == Q.length
        Q.tail = 1
else Q.tail = Q.tail + 1
```

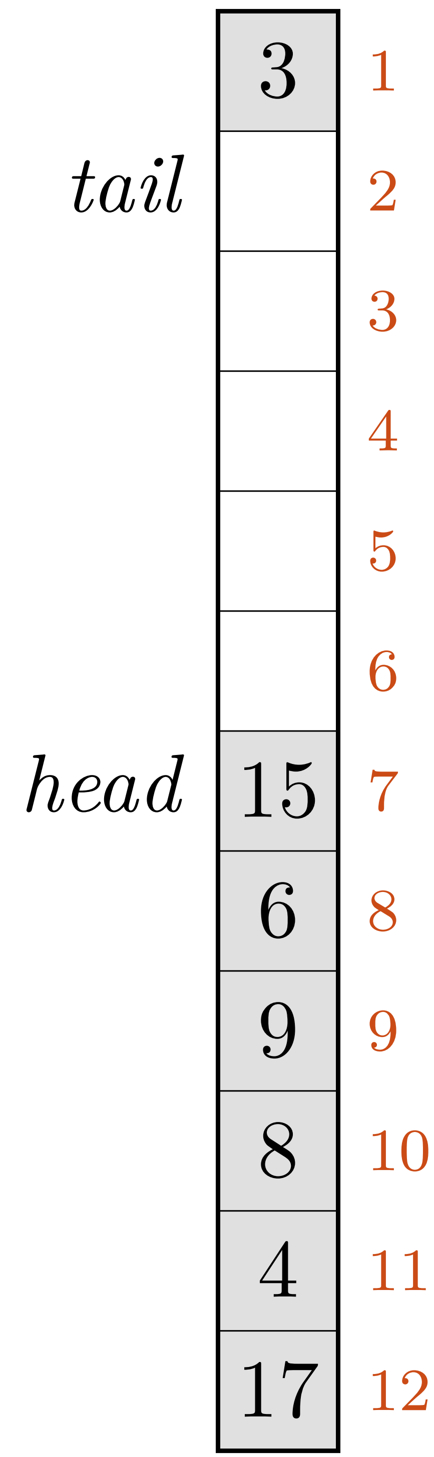


$x \;=\; 3$

ENQUEUE(Q, x)

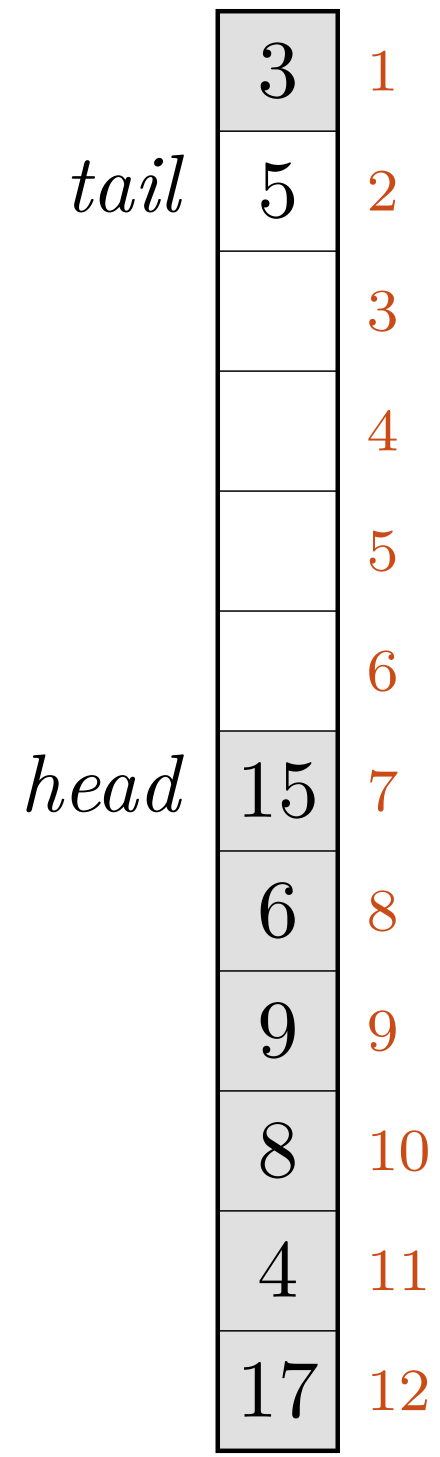
```
Q[Q.tail] = x
if Q.tail == Q.length
        Q.tail = 1
else Q.tail = Q.tail + 1
```

$x = 5$



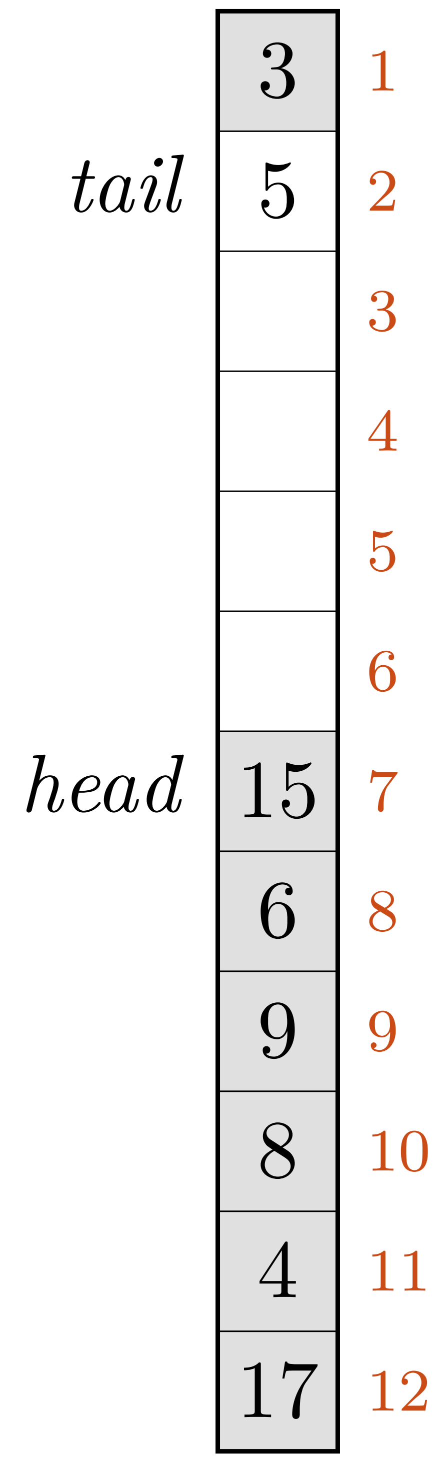
```
Enqueue(Q, x)
Q[Q.tail] = x
if Q.tail == Q.length
        Q.tail = 1
else Q.tail = Q.tail + 1
```

$x = 5$



```
Enqueue(Q, x)
Q[Q.tail] = x
if Q.tail == Q.length
        Q.tail = 1
else Q.tail = Q.tail + 1
```

$x = 5$



ENQUEUE(Q, x)

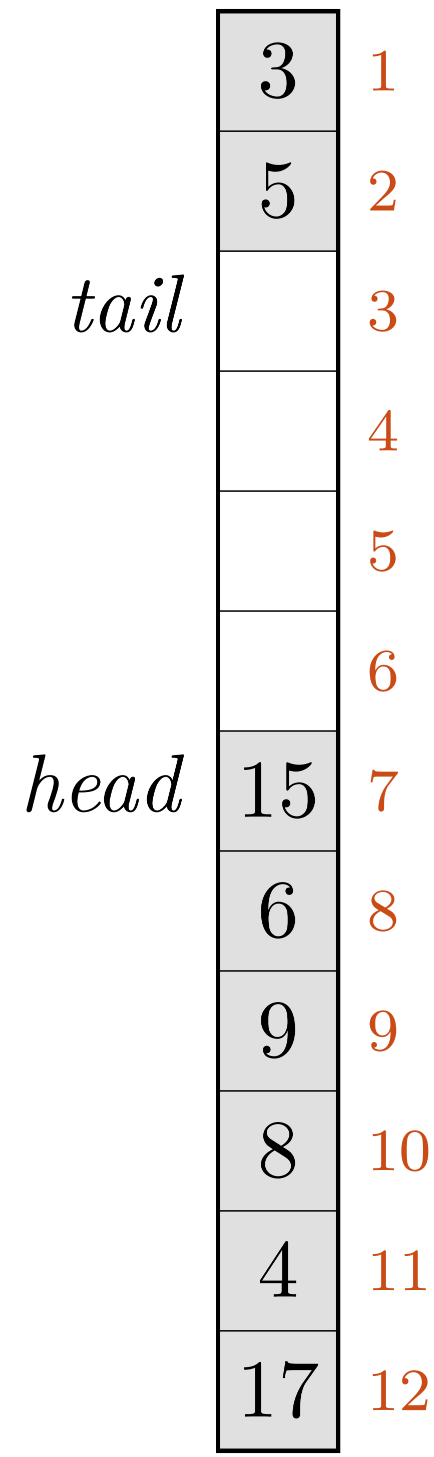
$Q[Q.tail] = x$

if $Q.tail == Q.length$

  $Q.tail = 1$

else $Q.tail = Q.tail + 1$

$x = 5$



Dequeue(Q)

Den første i køen går ut

$\textsc{Dequeue}(\mathrm{Q})$
$1 \quad x = \mathrm{Q}[\mathrm{Q}.\mathit{head}]$

x er den som står først i køen

$\text{D}\textsc{equeue}(Q)$
$x = Q[Q.\mathit{head}]$
if $Q.\mathit{head} == Q.\mathit{length}$

Vi må forskyve *head*, men har kommet til slutten!

```
DEQUEUE(Q)
x = Q[Q.head]
if Q.head == Q.length
    Q.head = 1
```

Tabellen «går i ring», så neste posisjon er 1

```
DEQUEUE(Q)
x = Q[Q.head]
if Q.head == Q.length
        Q.head = 1
else Q.head = Q.head + 1
```

Har vi ledig plass, er det jo ikke noe problem

$\textsc{Dequeue}(Q)$

$x = Q[Q.\mathit{head}]$

if $Q.\mathit{head} == Q.\mathit{length}$

  $Q.\mathit{head} = 1$

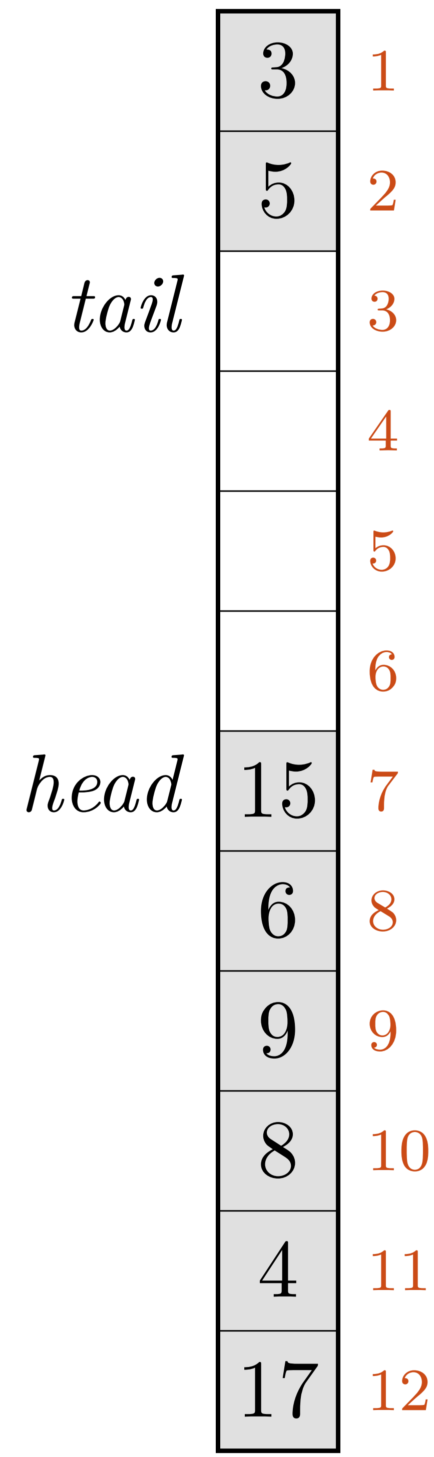
else $Q.\mathit{head} = Q.\mathit{head} + 1$

return x

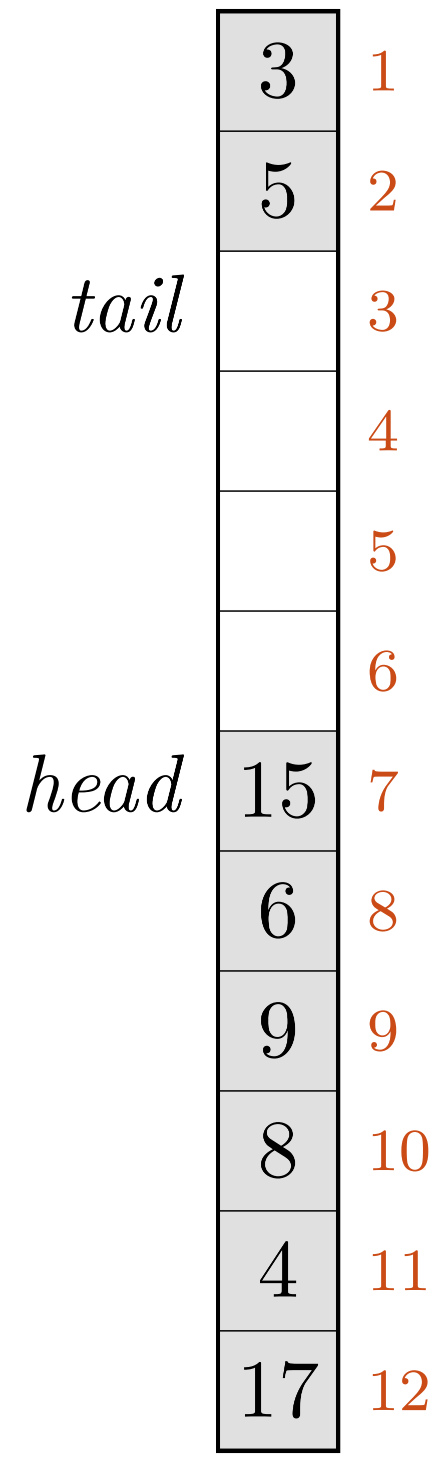
DEQUEUE(Q)

```
x = Q[Q.head]
if Q.head == Q.length
        Q.head = 1
else Q.head = Q.head + 1
return x
```

x = -



```
DEQUEUE(Q)
x = Q[Q.head]
if Q.head == Q.length
        Q.head = 1
else Q.head = Q.head + 1
return x
```



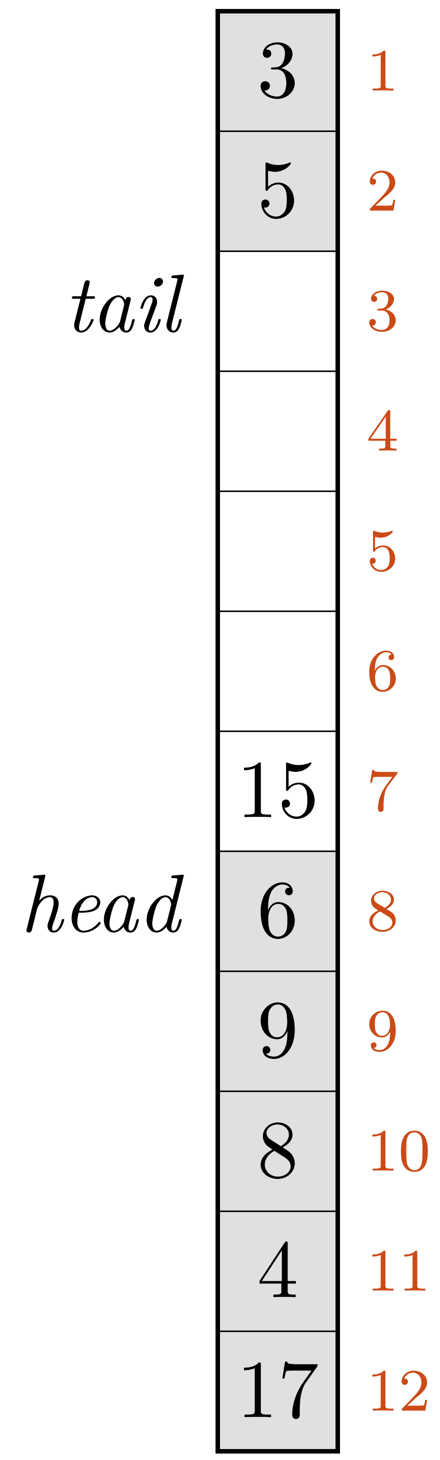
$x = 15$

```
DEQUEUE(Q)
x = Q[Q.head]
if Q.head == Q.length
      Q.head = 1
else Q.head = Q.head + 1
return x
```

	3	1
	5	2
tail		3
		4
		5
		6
head	15	7
	6	8
	9	9
	8	10
	4	11
	17	12

$x \; = \; 15$

```
DEQUEUE(Q)
x = Q[Q.head]
if Q.head == Q.length
        Q.head = 1
else Q.head = Q.head + 1
return x
```

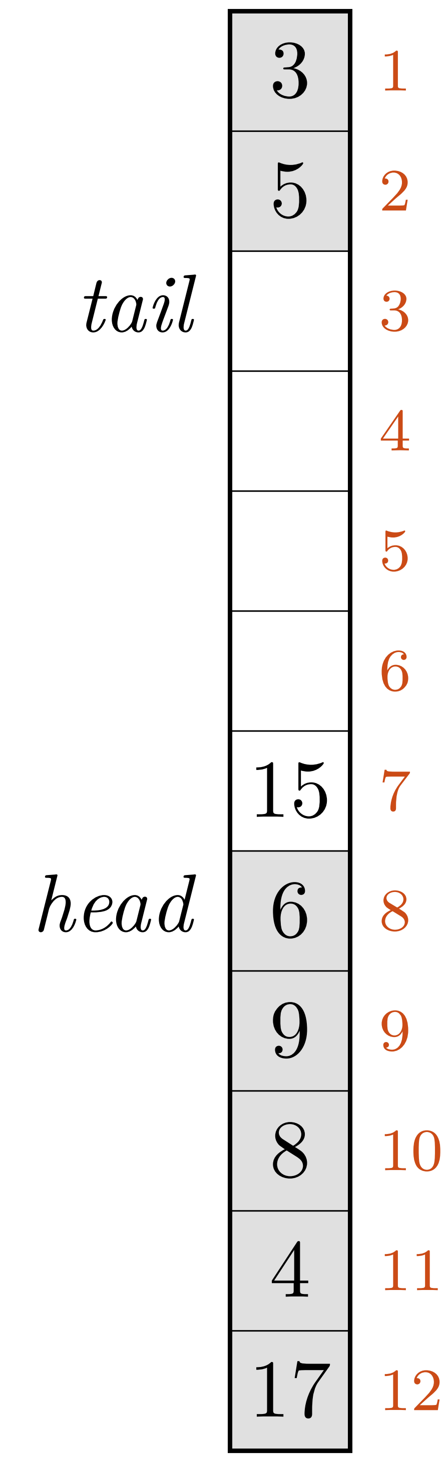


$x \;=\; 15$

```
DEQUEUE(Q)
x = Q[Q.head]
if Q.head == Q.length
        Q.head = 1
else Q.head = Q.head + 1
return x
```

$\rightarrow$ 15

$x \; = \; 15$



2:4

Lenkede lister

2:4

Lenkede lister

- **Består av «noder», som peker på neste (og kanskje forrige)**
- **Tar lineær tid å slå opp på en gitt posisjon**
- **Tar konstant tid å sette inn/slette elementer**

List-Search(L, k)

L liste
k nøkkelverdi

Finn en node med nøkkel k

$\textsc{List-Search}(\mathrm{L}, k)$
1 $x = \mathrm{L}.head$

L liste
k nøkkelverdi
x nodepeker

Vi begynner med den første noden

LIST-SEARCH(L, k)

$x = \mathrm{L}.head$

while $x \neq$ NIL and $x.key \neq k$

L liste
k nøkkelverdi
x nodepeker

Har vi nådd slutten eller funnet k?

$\textsc{List-Search}(\mathrm{L}, k)$
1 $x = \mathrm{L}.\mathit{head}$
2 **while** $x \neq \textsc{nil}$ and $x.\mathit{key} \neq k$
3 $\quad x = x.\mathit{next}$

L	liste
k	nøkkelverdi
x	nodepeker

Hvis ikke, så går vi til neste node

LIST-SEARCH(L, k)
1 $x = \mathrm{L}.head$
2 **while** $x \neq$ NIL and $x.key \neq k$
3 $\quad x = x.next$
4 **return** x

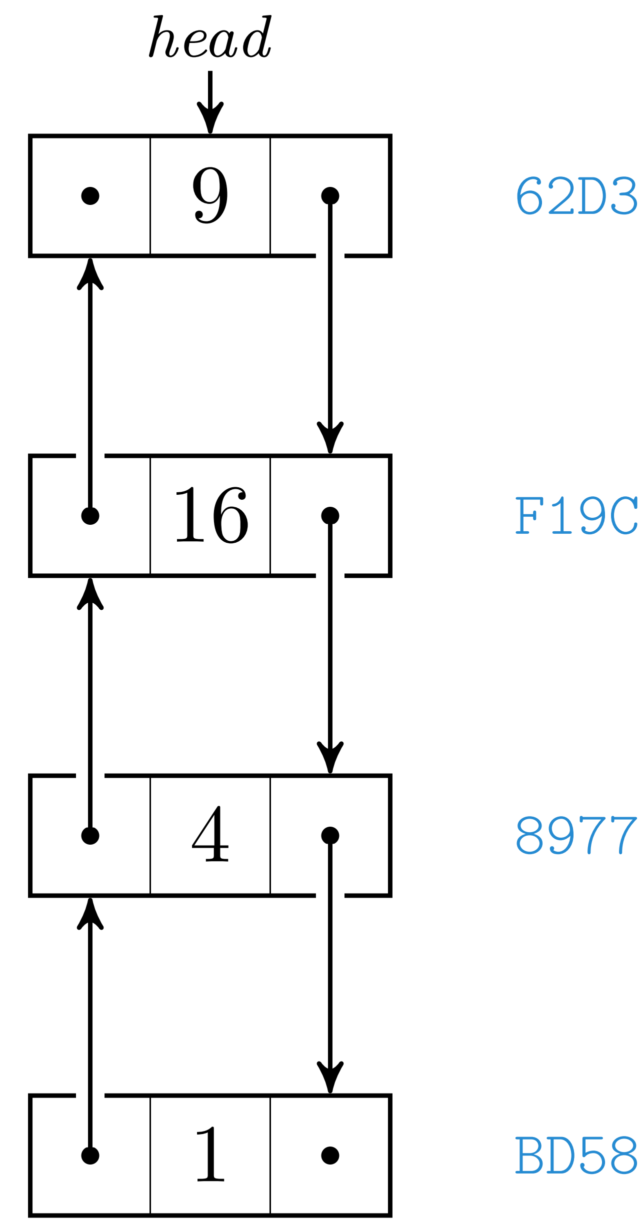
L	liste
k	nøkkelverdi
x	nodepeker

Her er enten $x ==$ NIL eller $x.key == k$

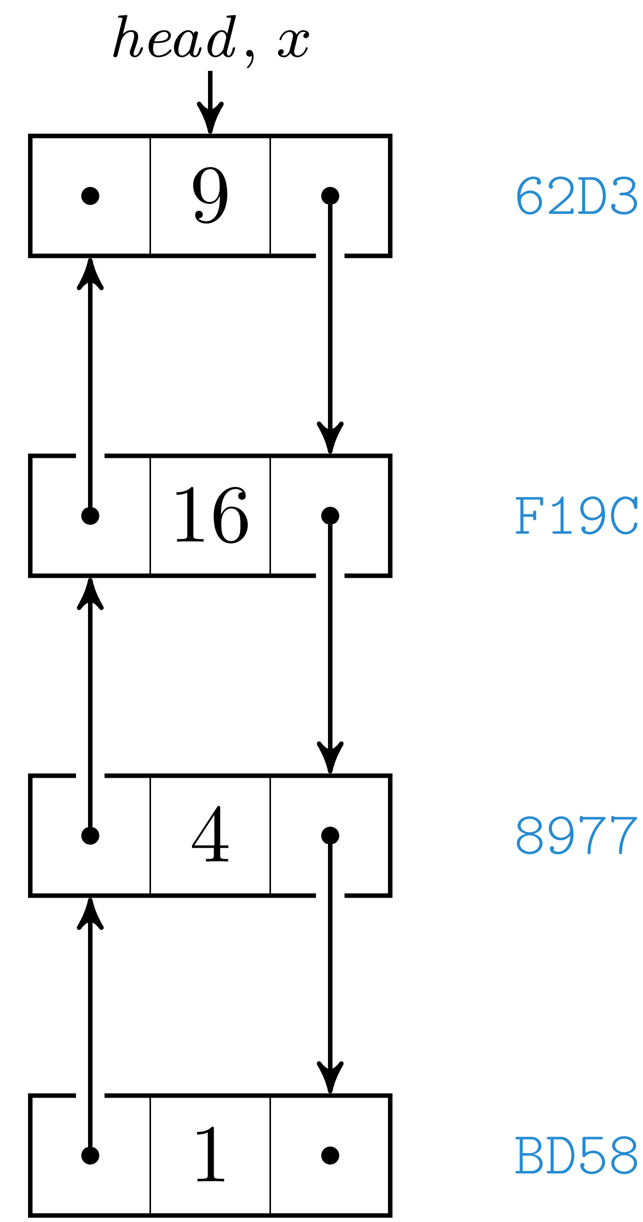
```
LIST-SEARCH(L, k)
x = L.head
while x ≠ NIL and x.key ≠ k
    x = x.next
return x
```

$k, x \;=\; 4, -$

De blå tallene ved siden av nodene er ment å være minne-adresser. En peker er egentlig bare en tallvariabel som inneholder en slik adresse.



```
List-Search(L, k)
x = L.head
while x ≠ NIL and x.key ≠ k
        x = x.next
return x
```



$k, x \;=\; 4,$ 62D3

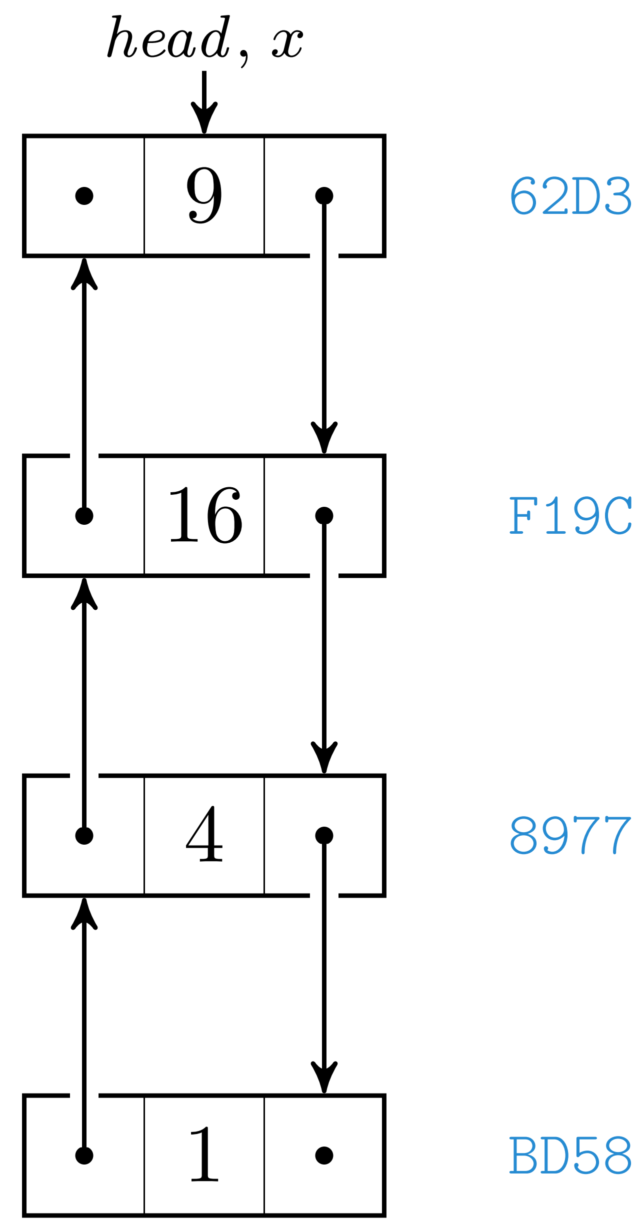
List-Search(L, k)

$x = \text{L}.head$

while $x \neq$ NIL and $x.key \neq k$

  $x = x.next$

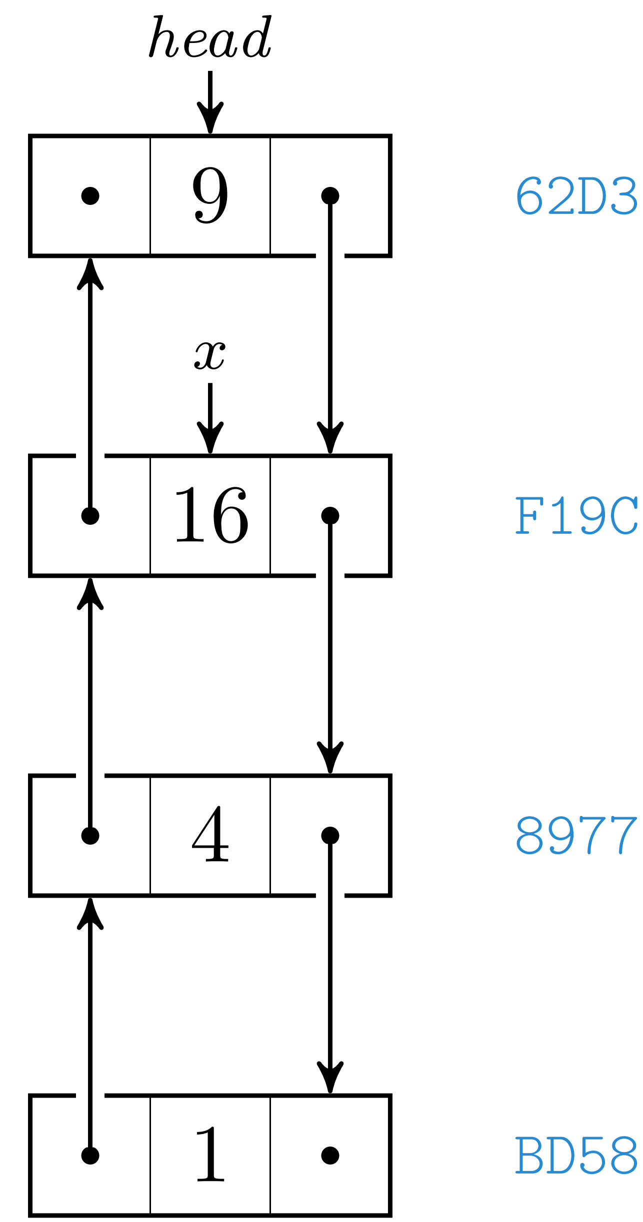
return x



$k, x \;=\; 4,$ 62D3

```
LIST-SEARCH(L, k)
x = L.head
while x ≠ NIL and x.key ≠ k
        x = x.next
return x
```

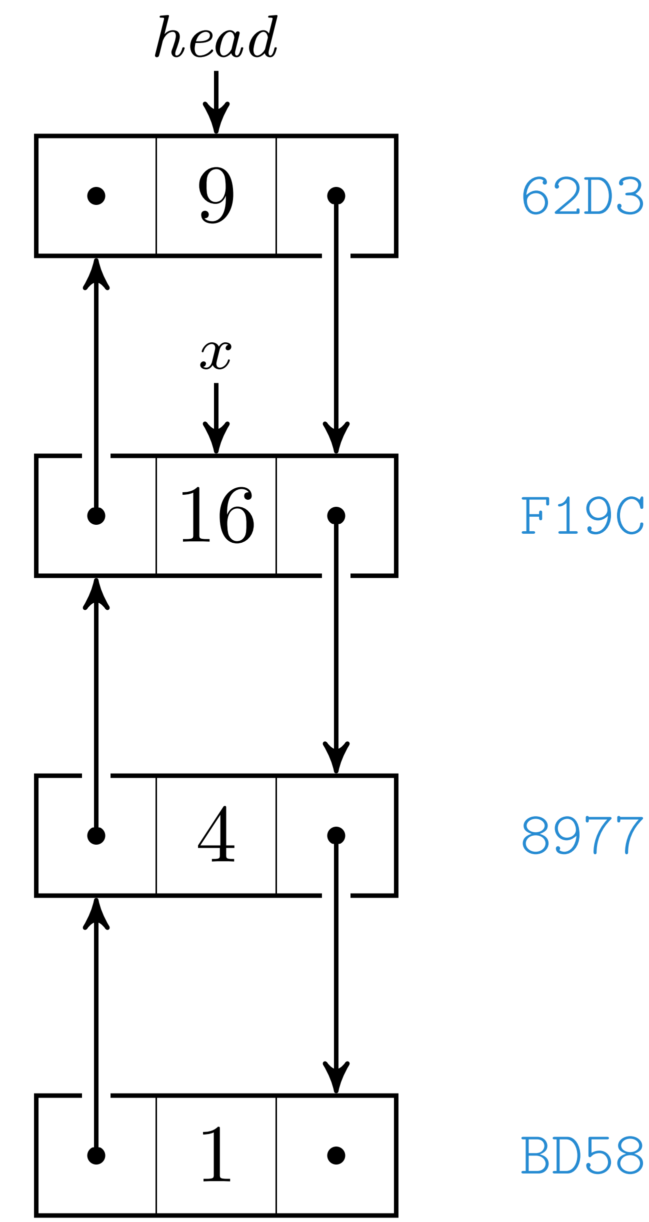
$k, x \;=\; 4,$ F19C



List-Search(L, k)

```
x = L.head
while x ≠ NIL and x.key ≠ k
    x = x.next
return x
```

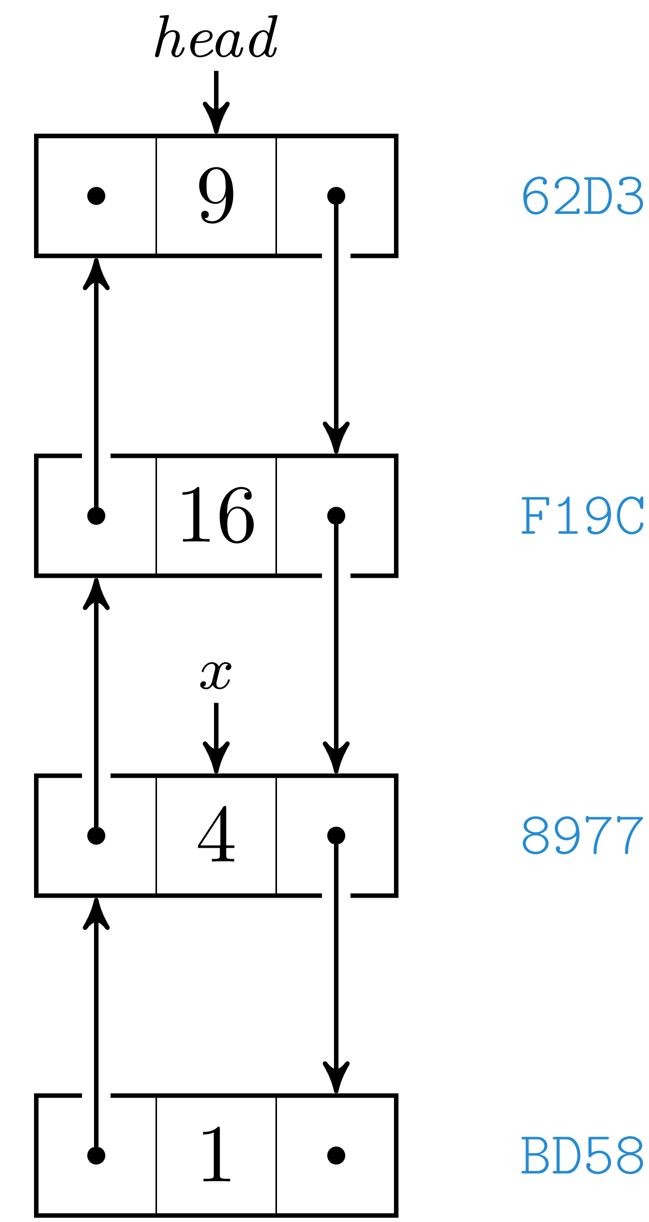
$k, x \;=\; 4,$ F19C



LIST-SEARCH(L, k)

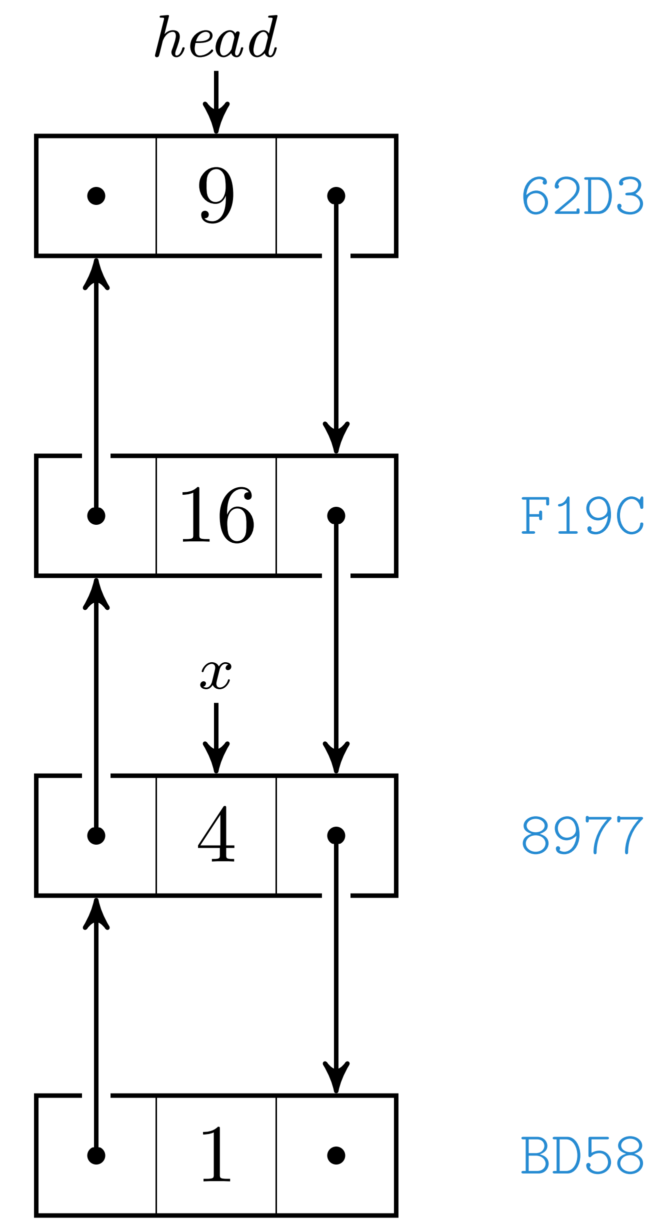
```
x = L.head
while x ≠ NIL and x.key ≠ k
        x = x.next
return x
```

$k, x \;=\; 4, 8977$



List-Search(L, k)

$x = \mathrm{L}.head$
while $x \neq$ NIL and $x.key \neq k$
    $x = x.next$
return x

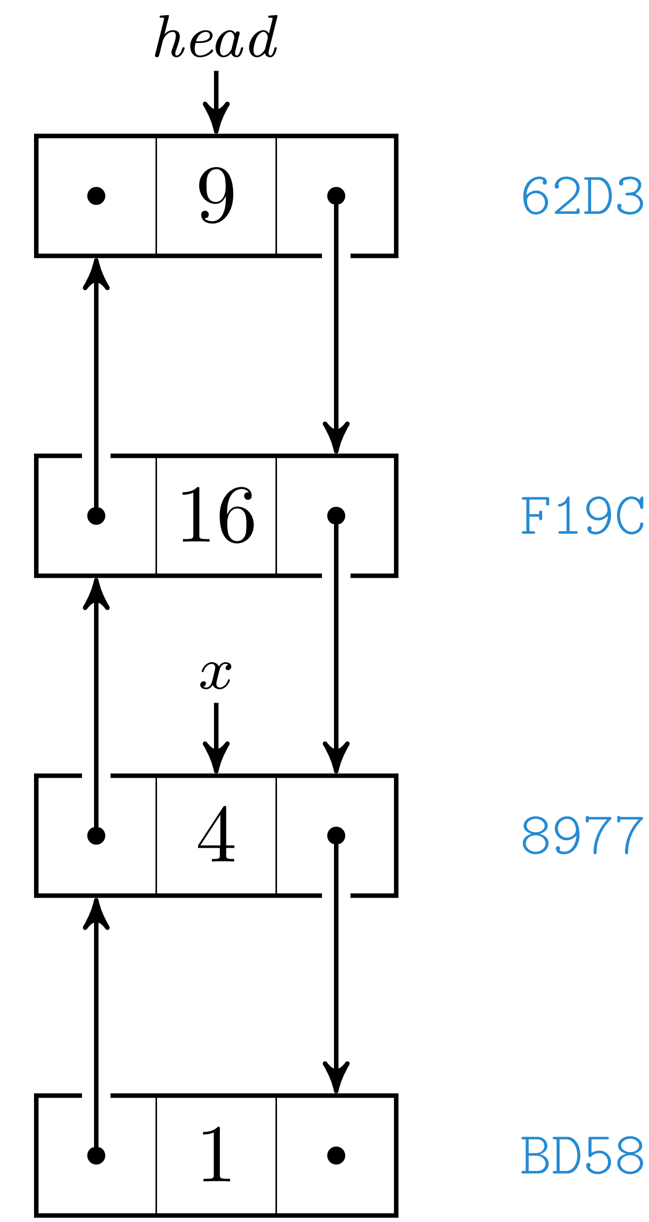


$k, x \;=\; 4,$ 8977

LIST-SEARCH(L, k)

```
x = L.head
while x ≠ NIL and x.key ≠ k
        x = x.next
return x
```

$\longrightarrow$ 8977

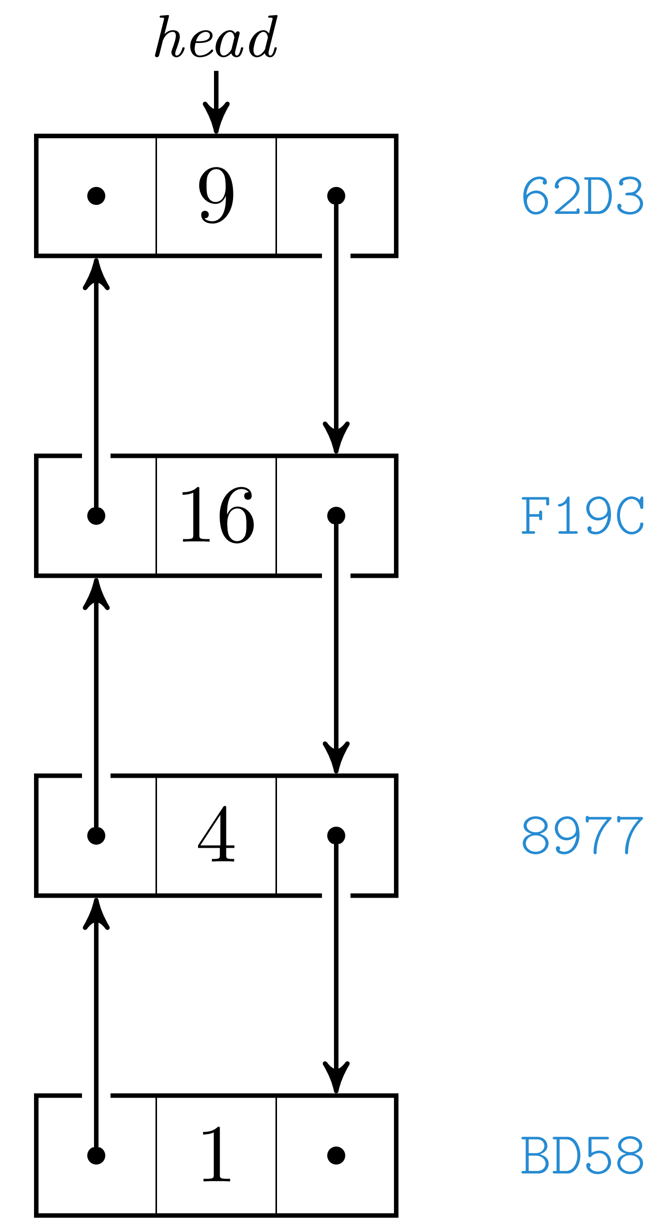


$k, x \; = \; 4,$ 8977

LIST-SEARCH(L, k)

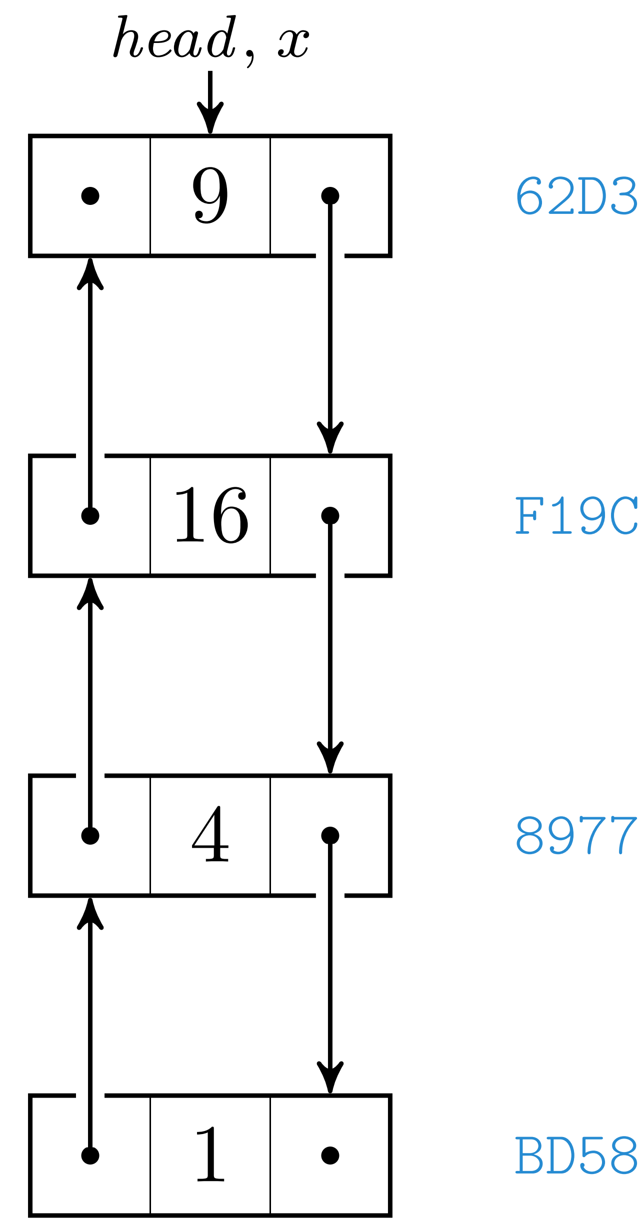
```
x = L.head
while x ≠ NIL and x.key ≠ k
    x = x.next
return x
```

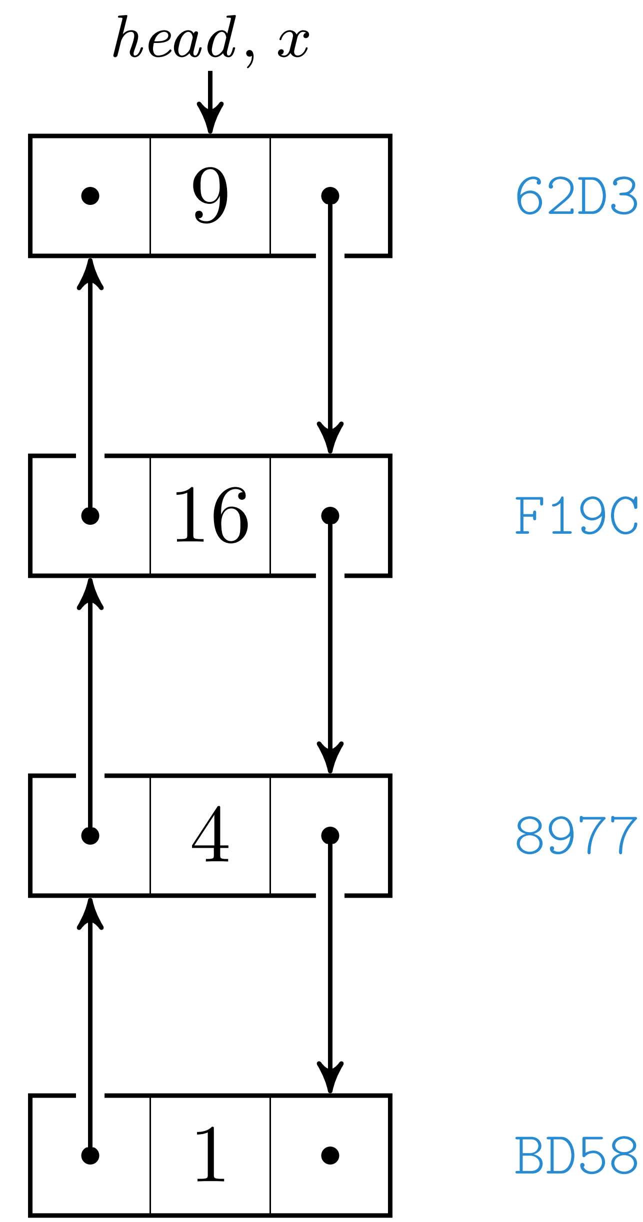
$k, x \; = \; 7, \text{-}$



```
LIST-SEARCH(L, k)
x = L.head
while x ≠ NIL and x.key ≠ k
      x = x.next
return x
```

$k, x \;=\; 7,$ 62D3





List-Search(L, k)

$x = L.head$

while $x \neq$ NIL and $x.key \neq k$

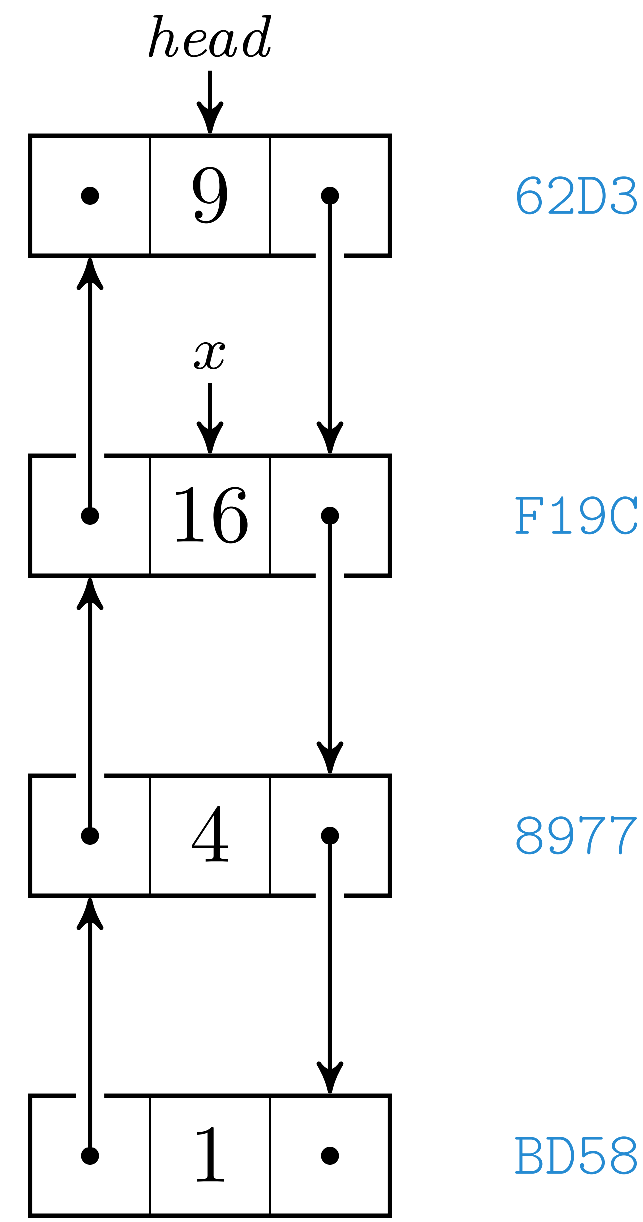
  $x = x.next$

return x

$k, x \;=\; 7,$ 62D3

LIST-SEARCH(L, k)

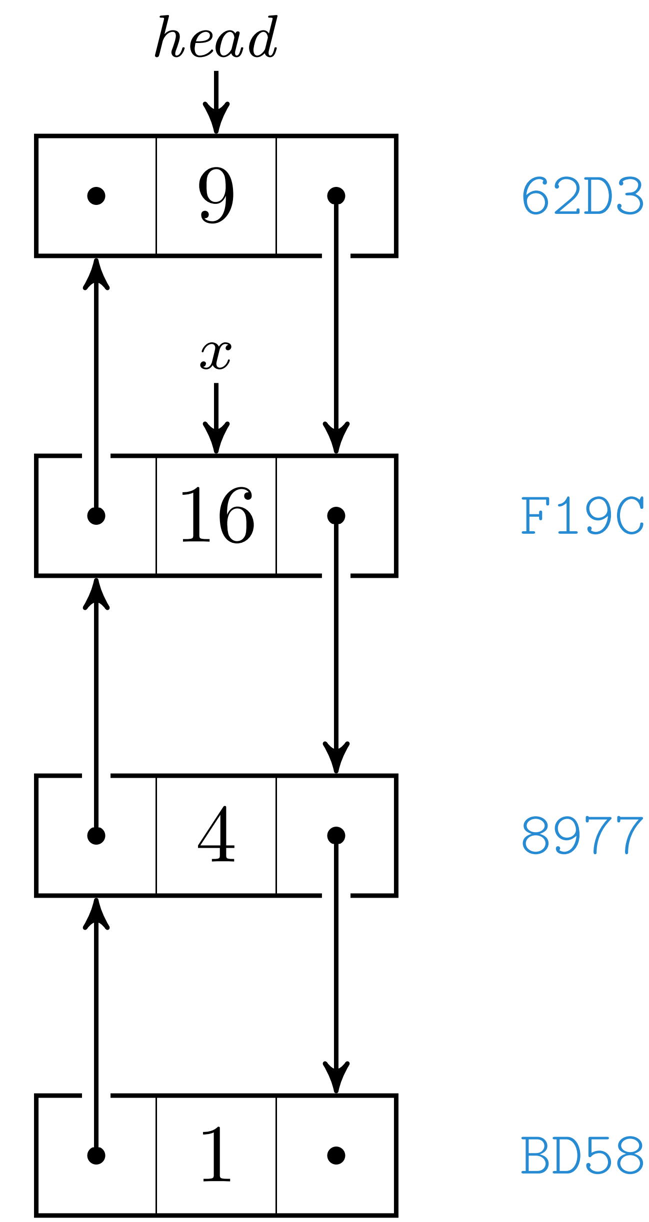
```
x = L.head
while x ≠ NIL and x.key ≠ k
        x = x.next
return x
```



$k, x \;=\; 7,$ F19C

LIST-SEARCH(L, k)

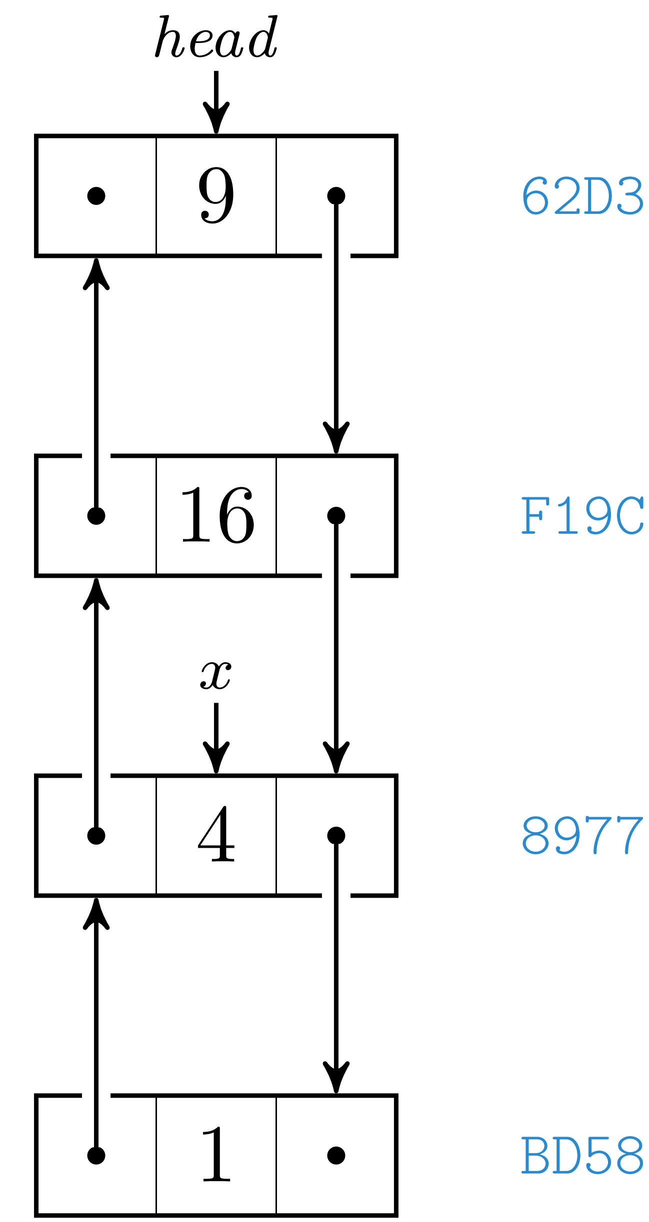
```
x = L.head
while x ≠ NIL and x.key ≠ k
        x = x.next
return x
```



$k, x \;=\; 7,$ F19C

List-Search(L, k)

```
x = L.head
while x ≠ NIL and x.key ≠ k
    x = x.next
return x
```



$k, x \;=\; 7,$ 8977

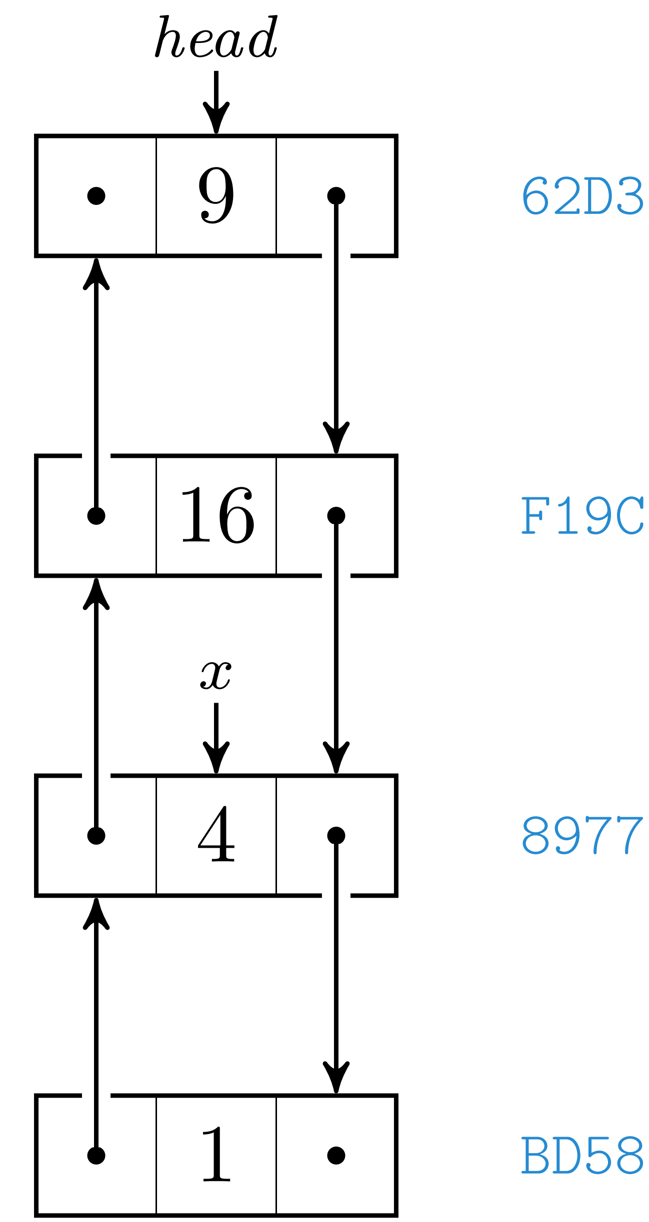
List-Search(L, k)

$x = \mathrm{L}.head$

while $x \neq$ NIL and $x.key \neq k$

   $x = x.next$

return x

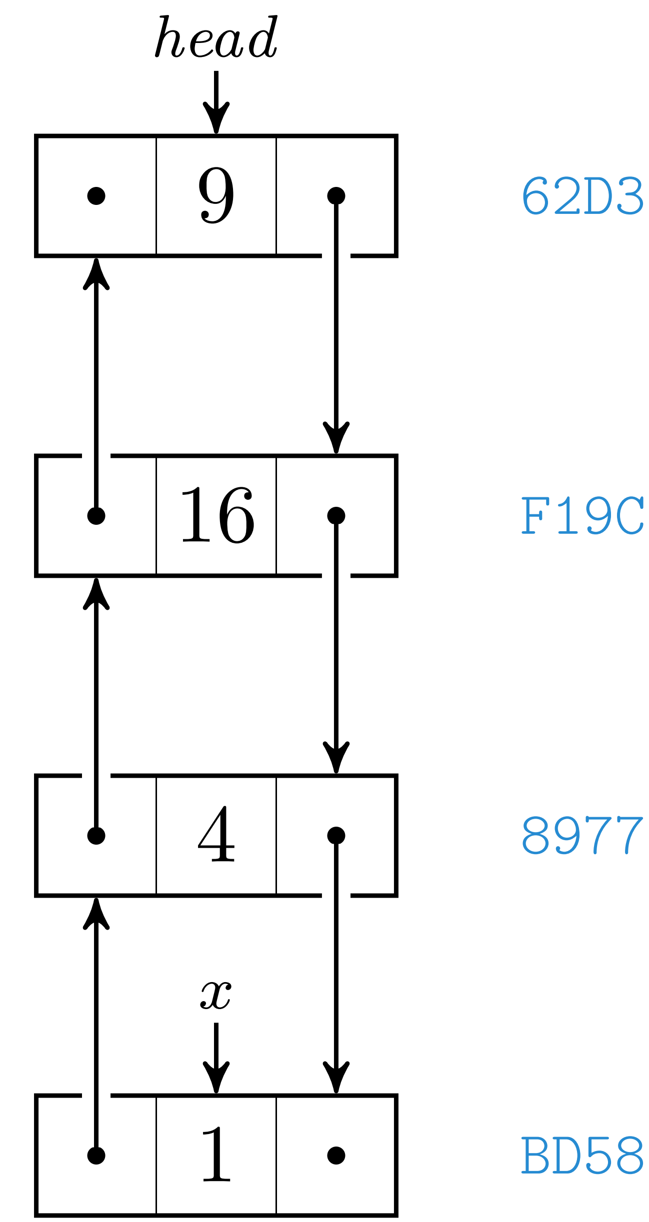


$k, x \;=\; 7,$ 8977

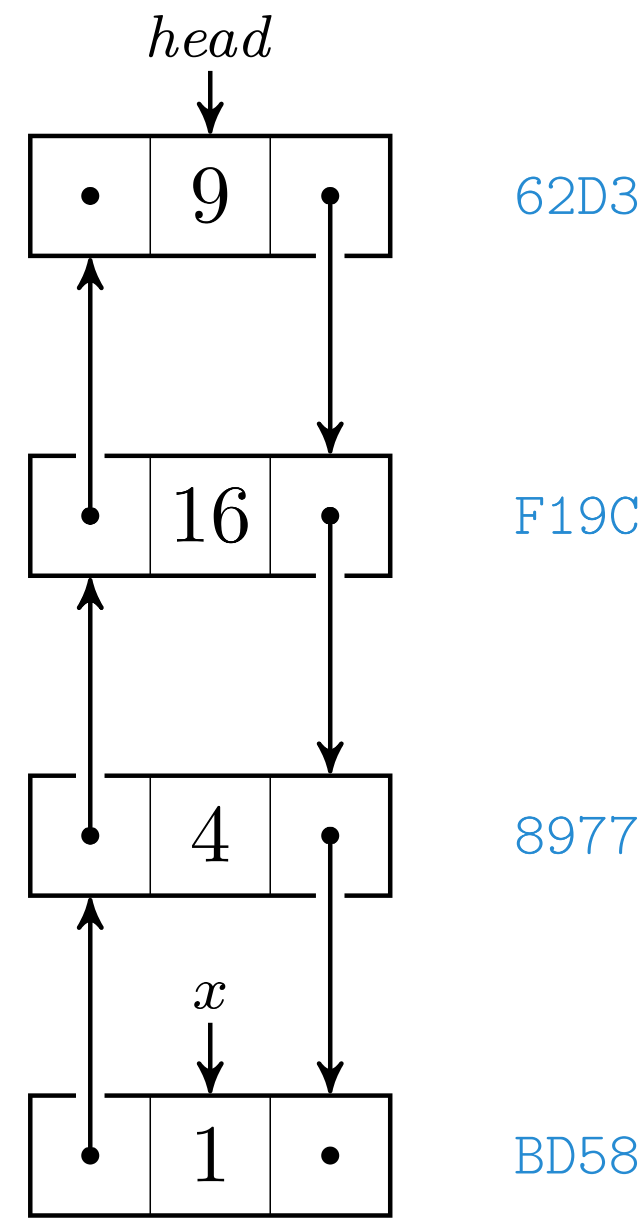
$\text{List-Search}(L, k)$

$x = L.head$
while $x \neq \text{NIL}$ and $x.key \neq k$
    $x = x.next$
return x

$k, x \;=\; 7,$ BD58

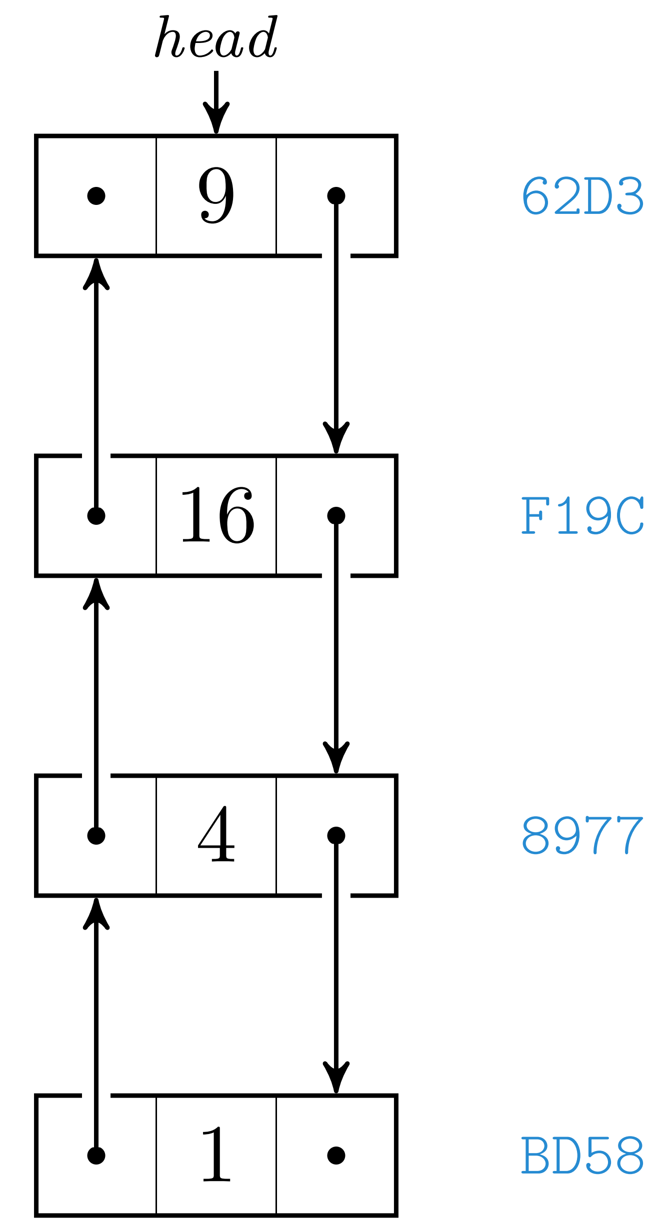


```
LIST-SEARCH(L, k)
x = L.head
while x ≠ NIL and x.key ≠ k
    x = x.next
return x
```



$k, x \;=\; 7,$ BD58

```
LIST-SEARCH(L, k)
x = L.head
while x ≠ NIL and x.key ≠ k
    x = x.next
return x
```

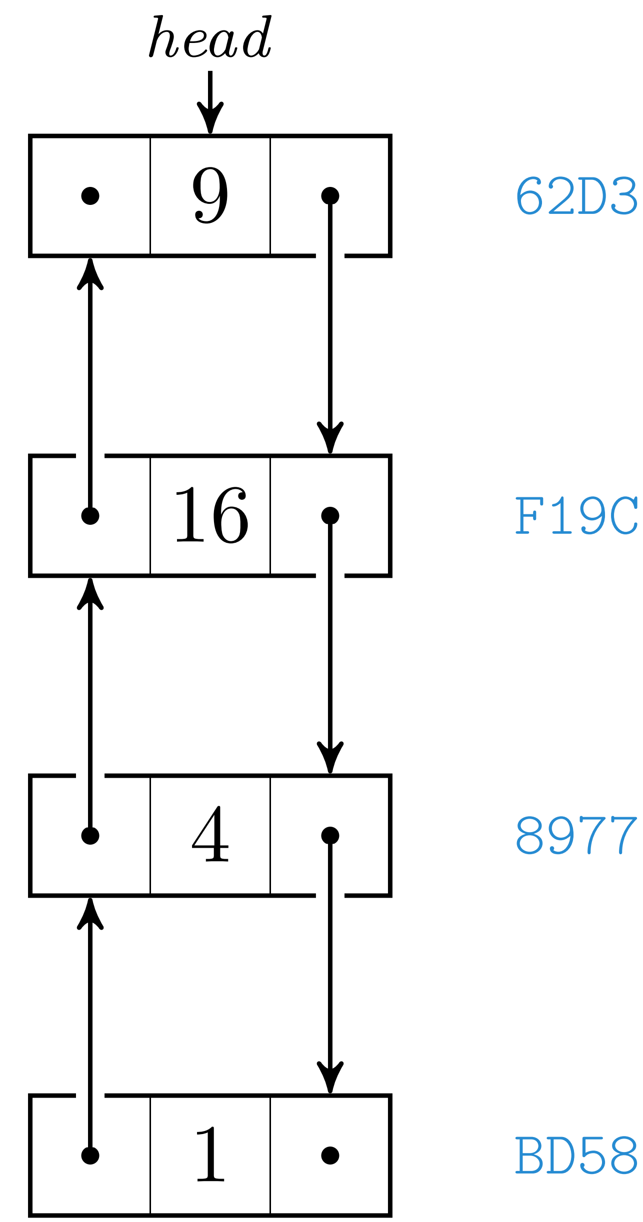


$k, x \;=\; 7, \text{NIL}$

List-Search(L, k)

$x = \text{L}.head$
while $x \neq \text{NIL}$ and $x.key \neq k$
  $x = x.next$
return x

$k, x \;=\; 7, \text{NIL}$

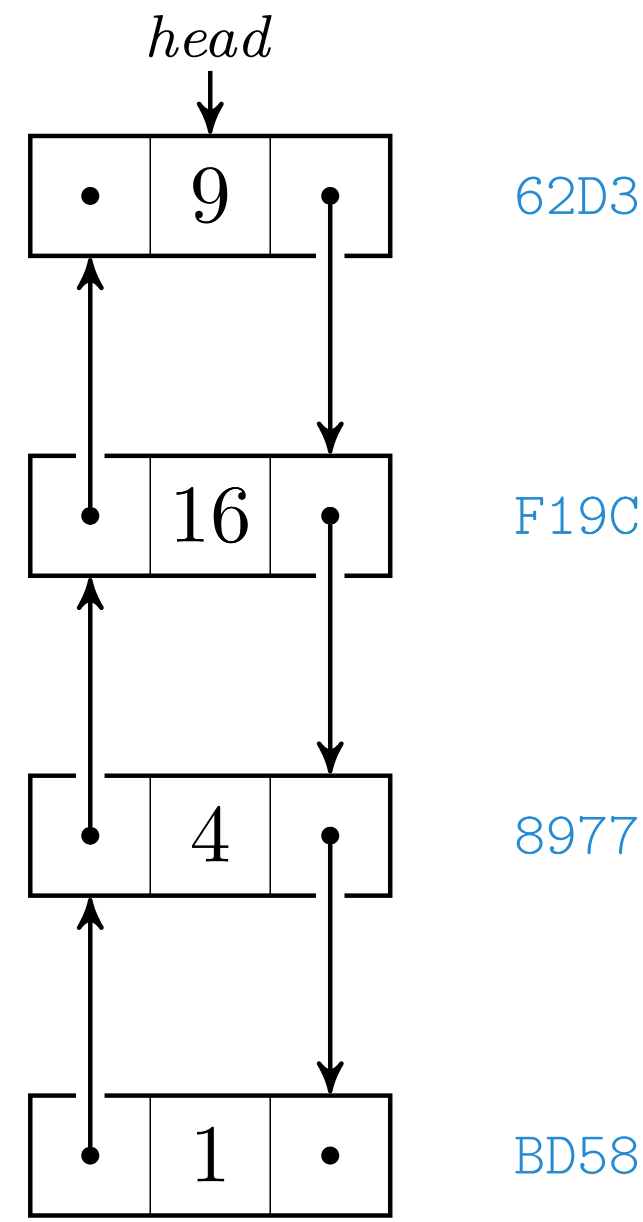


List-Search(L, k)

```
x = L.head
while x ≠ NIL and x.key ≠ k
        x = x.next
return x
```

$\longrightarrow$ NIL

$k, x \;=\; 7,$ NIL



List-Insert(L, x)

L liste
x peker, ny node

Sett inn noden x som nytt hode i lista L

$\textsc{List-Insert}(\mathrm{L}, x)$
1 $x.next = \mathrm{L}.head$

L liste
x peker, ny node

Forrige hode kommer nå etter x

```
List-Insert(L, x)
    x.next = L.head
    if L.head ≠ NIL
```

L liste
x peker, ny node

(Med mindre lista var tom . . .)

```
LIST-INSERT(L, x)
x.next = L.head
if L.head ≠ NIL
    L.head.prev = x
```

L liste
x peker, ny node

Det forrige hodet har nå x som forgjenger

mlh@ntnu.no

List-Insert(L, x)

```
x.next = L.head
if L.head ≠ NIL
    L.head.prev = x
L.head = x
```

L liste

x peker, ny node

«Kryssreferanser» er på plass: Sett inn noden!

List-Insert(L, x)

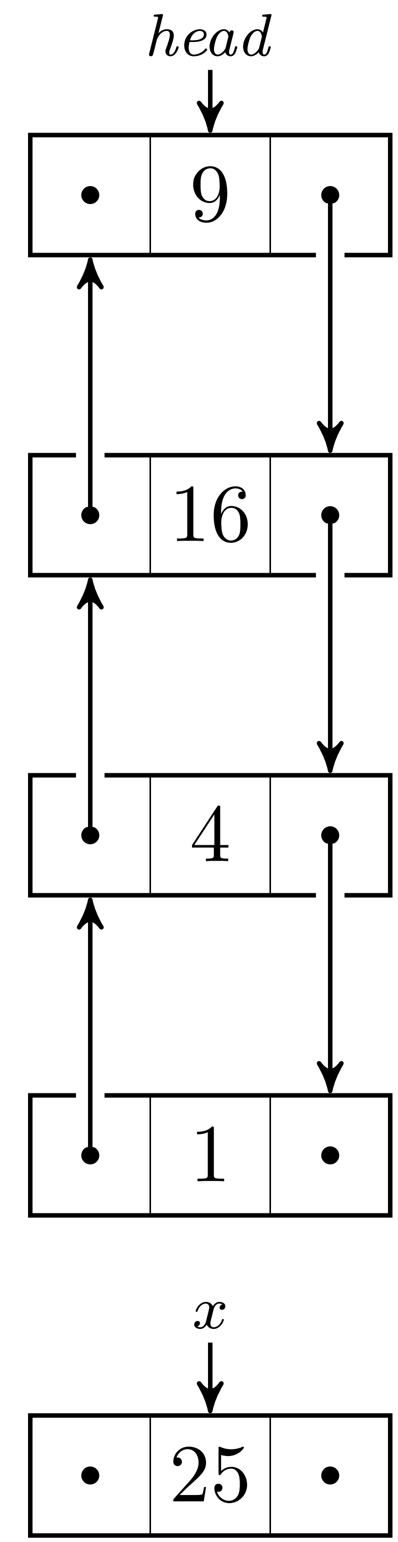
$x.next = \mathrm{L}.head$
if $\mathrm{L}.head \neq$ NIL
    $\mathrm{L}.head.prev = x$
$\mathrm{L}.head = x$
$x.prev =$ NIL

L liste
x peker, ny node

(…og husk at x ikke har noen forgjenger)

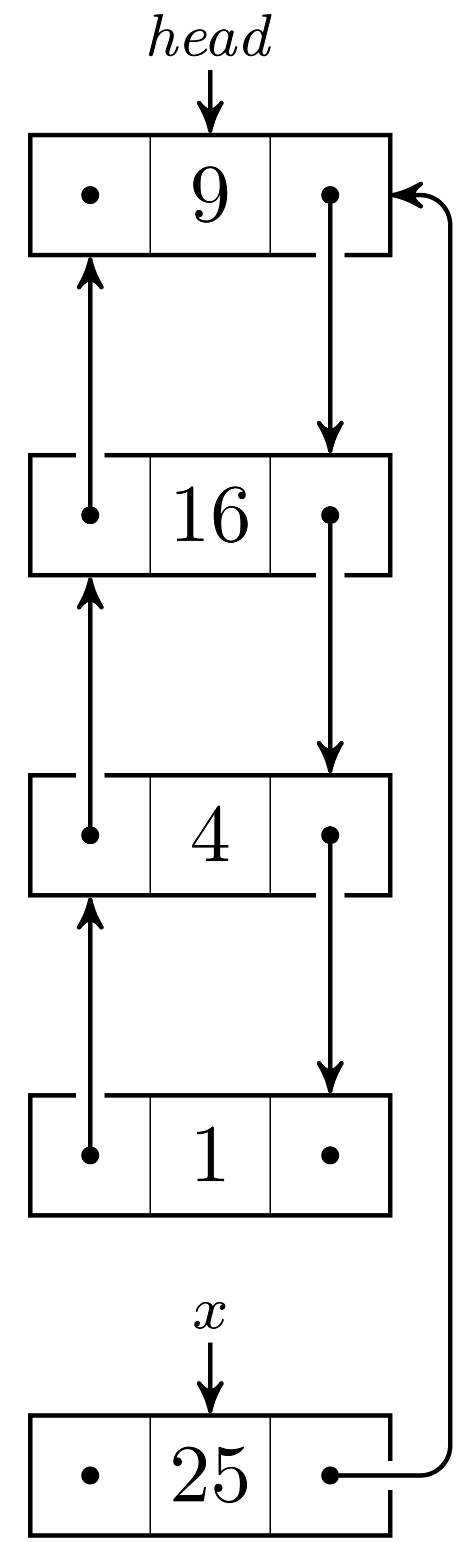
List-Insert(L, x)

```
x.next = L.head
if L.head ≠ NIL
        L.head.prev = x
L.head = x
x.prev = NIL
```



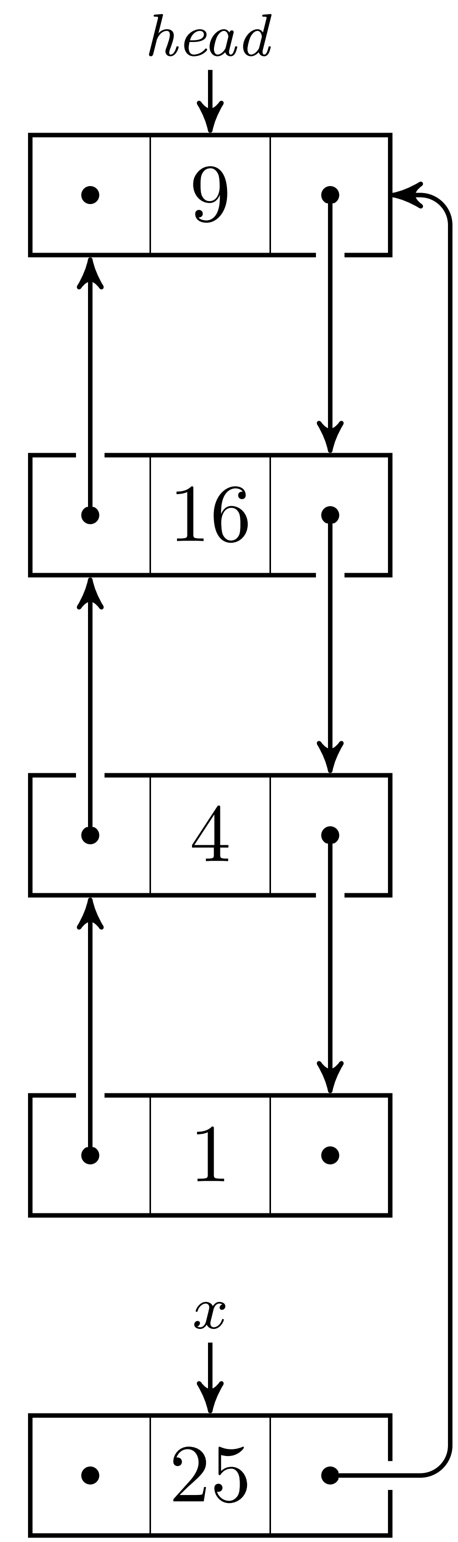
List-Insert(L, x)

$x.next = \text{L}.head$
if L.$head \neq$ NIL
    L.$head.prev = x$
L.$head = x$
$x.prev =$ NIL



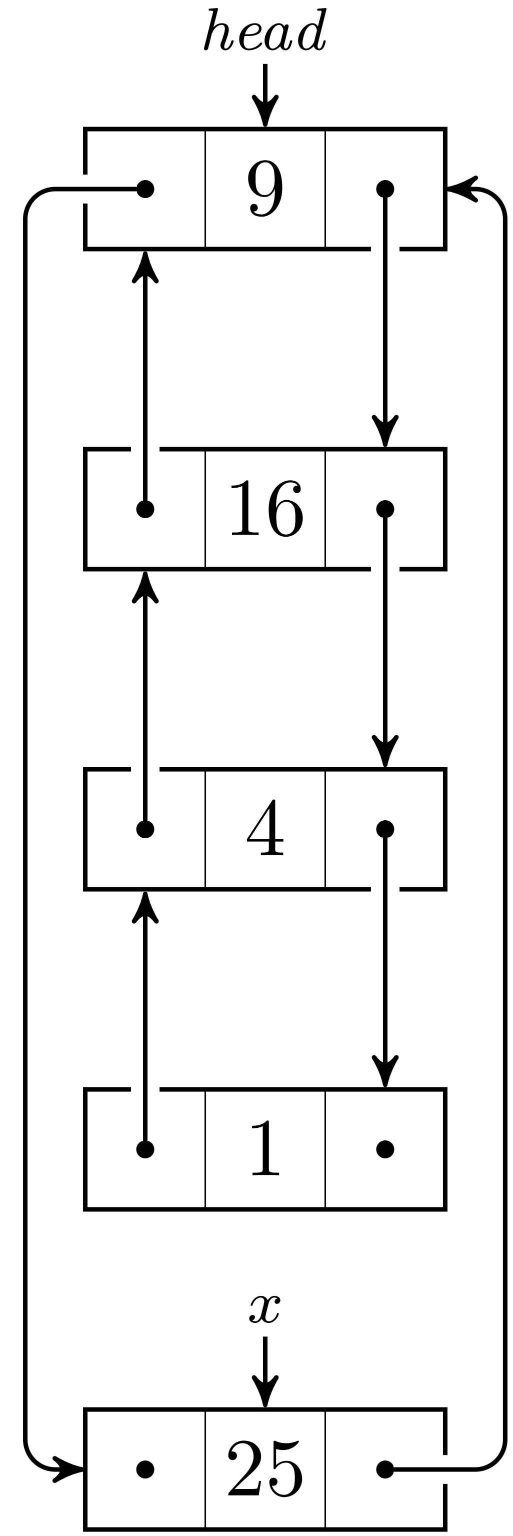
List-Insert(L, x)

$x.next = \mathrm{L}.head$
if $\mathrm{L}.head \neq$ nil
$\quad \mathrm{L}.head.prev = x$
$\mathrm{L}.head = x$
$x.prev =$ nil



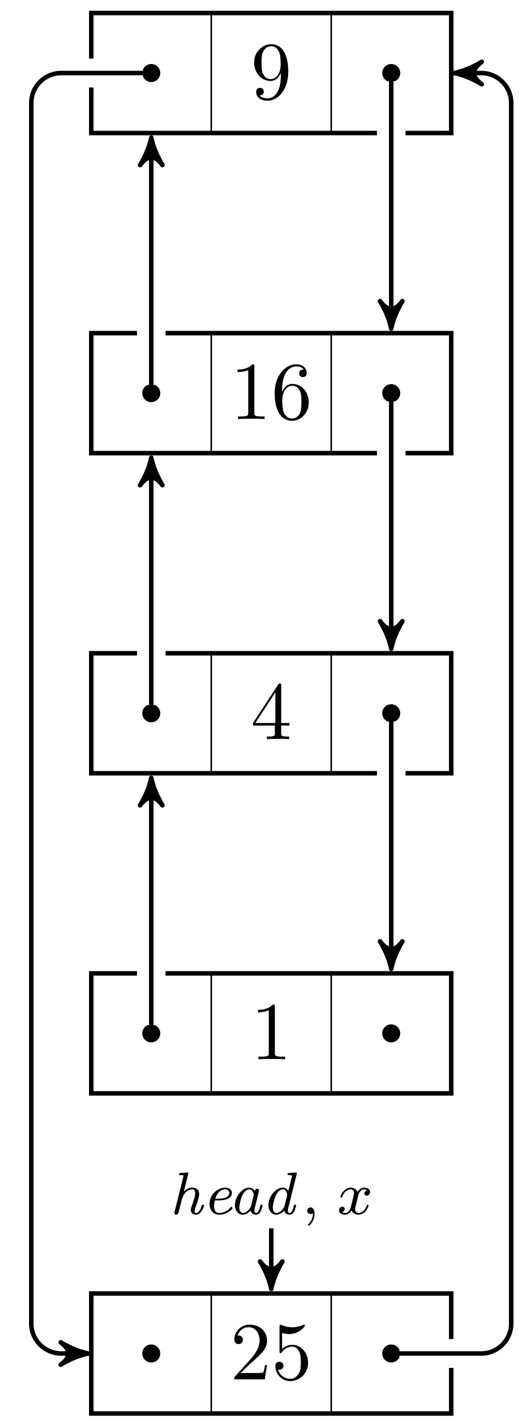
LIST-INSERT(L, x)

$x.next = \mathrm{L}.head$
if $\mathrm{L}.head \neq$ NIL
        $\mathrm{L}.head.prev = x$
$\mathrm{L}.head = x$
$x.prev =$ NIL



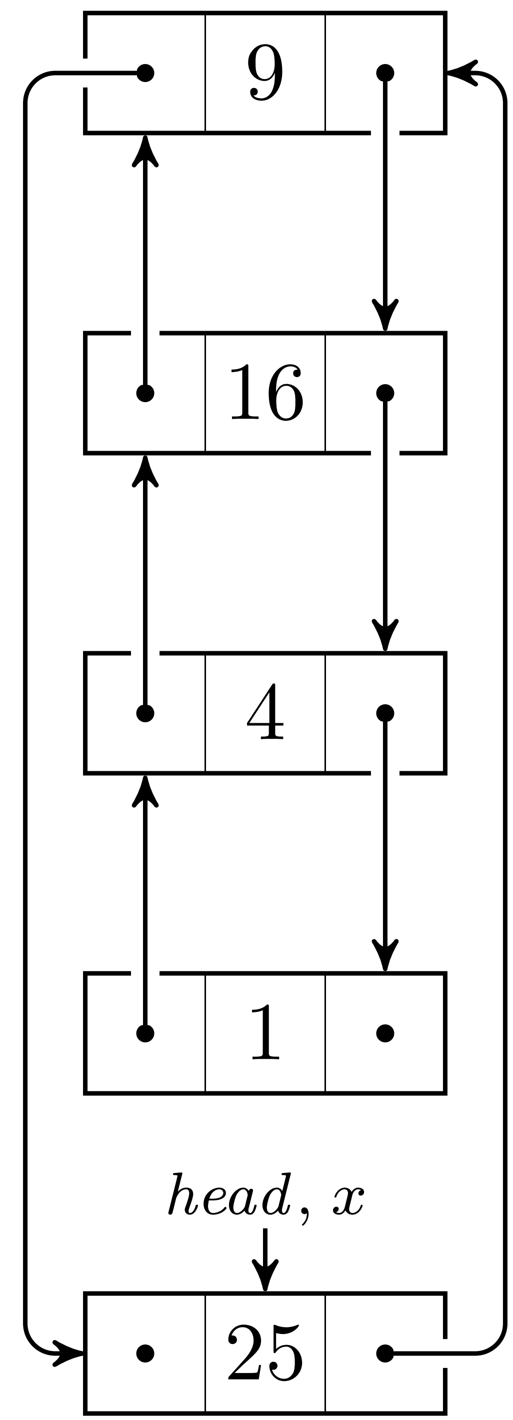
LIST-INSERT(L, x)

$x.next = \mathrm{L}.head$
if $\mathrm{L}.head \neq$ NIL
    $\mathrm{L}.head.prev = x$
$\mathrm{L}.head = x$
$x.prev =$ NIL



List-Insert(L, x)

$x.\mathit{next} = \mathrm{L}.\mathit{head}$
if $\mathrm{L}.\mathit{head} \neq$ NIL
    $\mathrm{L}.\mathit{head}.\mathit{prev} = x$
$\mathrm{L}.\mathit{head} = x$
$x.\mathit{prev} =$ NIL



$\textsc{List-Delete}(\mathrm{L}, x)$

L liste
x skal fjernes

Få nodene før og etter til å referere til hverandre, og «hoppe over» x, som fortsatt peker på disse nodene, men som ikke *blir pekt på* av noen noder i lista lenger.

Fjern noden x fra lista L

List-Delete(L, x)

1 **if** $x.prev \neq$ NIL

L liste

x skal fjernes

(Med mindre x står først . . .)

List-Delete(L, x)

if $x.prev \neq$ NIL

    $x.prev.next = x.next$

L liste

x skal fjernes

Forgjengeren arver x sin etterkommer

```
LIST-DELETE(L, x)
if x.prev ≠ NIL
    x.prev.next = x.next
else L.head = x.next
```

L liste
x skal fjernes

Hvis x sto først, står etterkommeren nå først

List-Delete(L, x)
```
1  if x.prev ≠ NIL
2        x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
```

L liste
x skal fjernes

(Med mindre x står sist . . .)

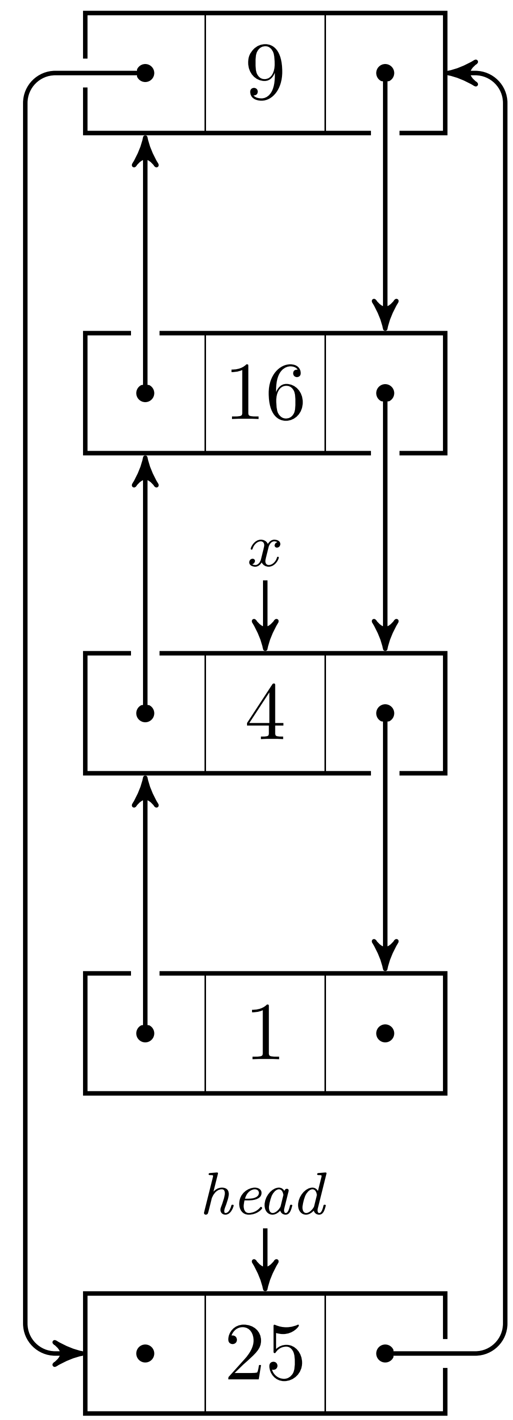
List-Delete(L, x)

```
if x.prev ≠ NIL
    x.prev.next = x.next
else L.head = x.next
if x.next ≠ NIL
    x.next.prev = x.prev
```

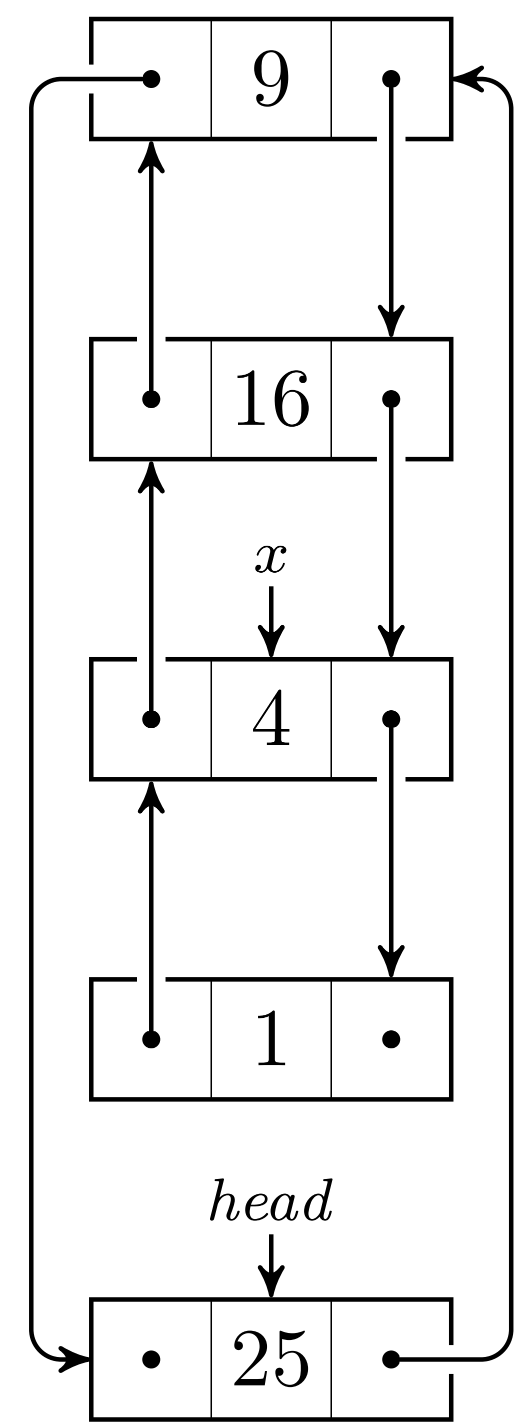
L liste
x skal fjernes

Etterkommeren arver x sin forgjenger

```
List-Delete(L, x)
if x.prev ≠ nil
    x.prev.next = x.next
else L.head = x.next
if x.next ≠ nil
    x.next.prev = x.prev
```

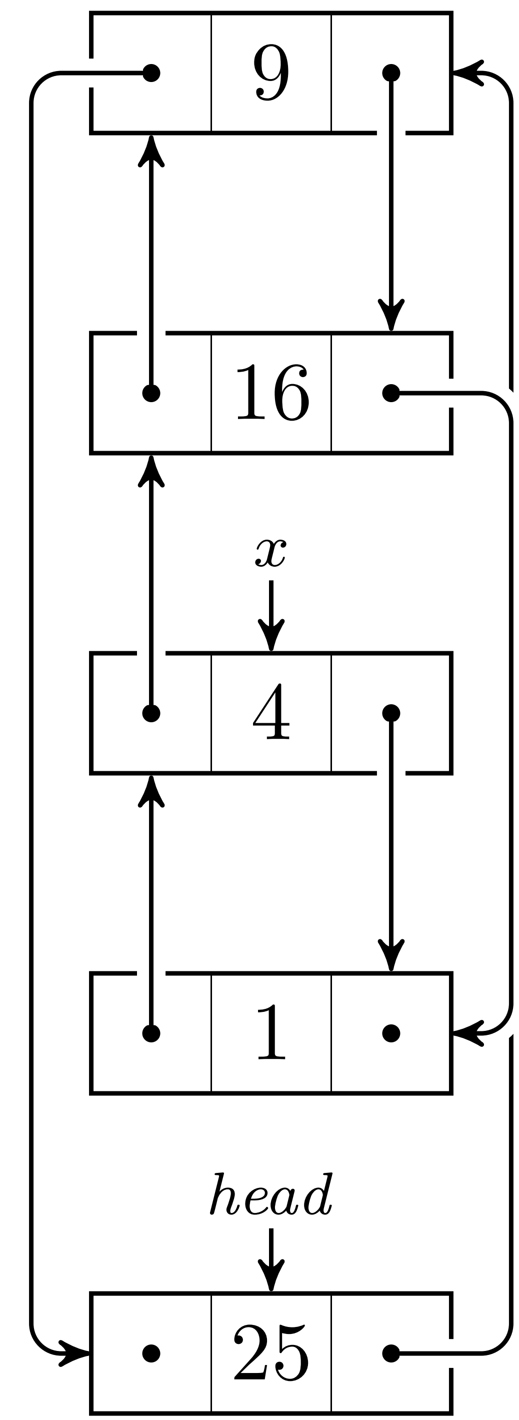


```
List-Delete(L, x)
if x.prev ≠ NIL
      x.prev.next = x.next
else L.head = x.next
if x.next ≠ NIL
      x.next.prev = x.prev
```



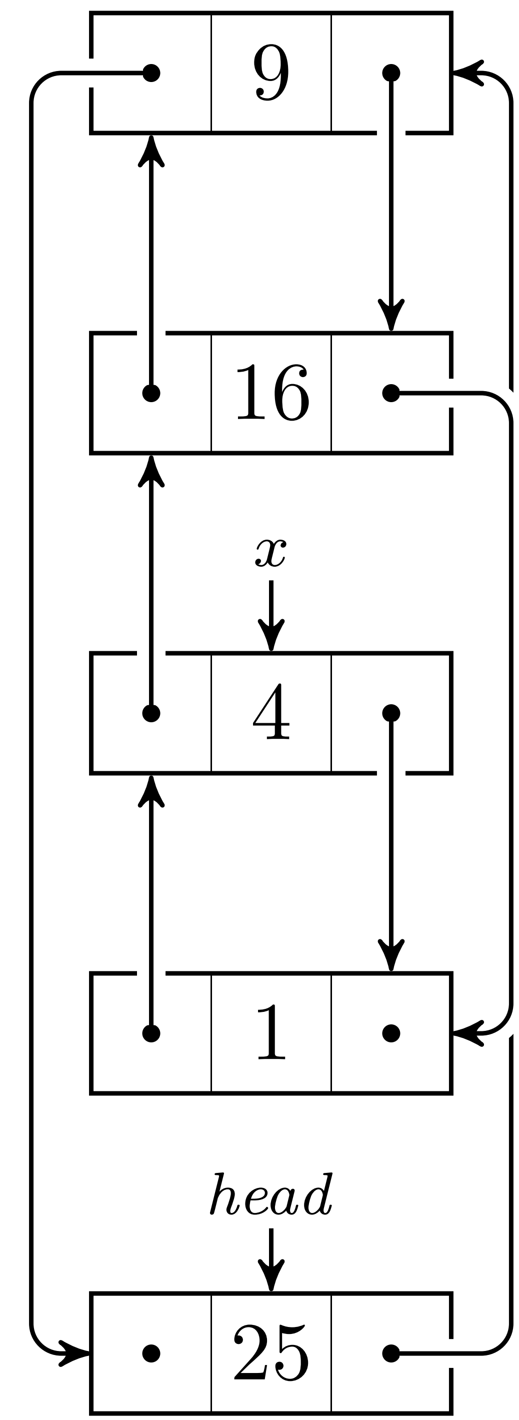
List-Delete(L, x)

```
if x.prev ≠ NIL
    x.prev.next = x.next
else L.head = x.next
if x.next ≠ NIL
    x.next.prev = x.prev
```



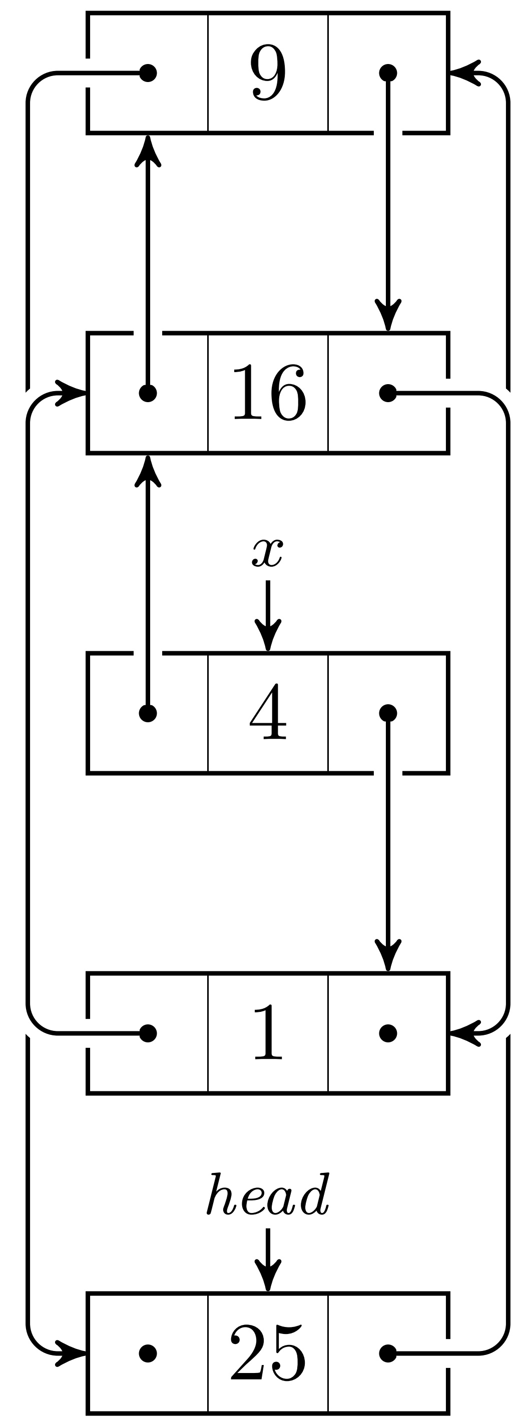
List-Delete(L, x)

```
if x.prev ≠ NIL
    x.prev.next = x.next
else L.head = x.next
if x.next ≠ NIL
    x.next.prev = x.prev
```



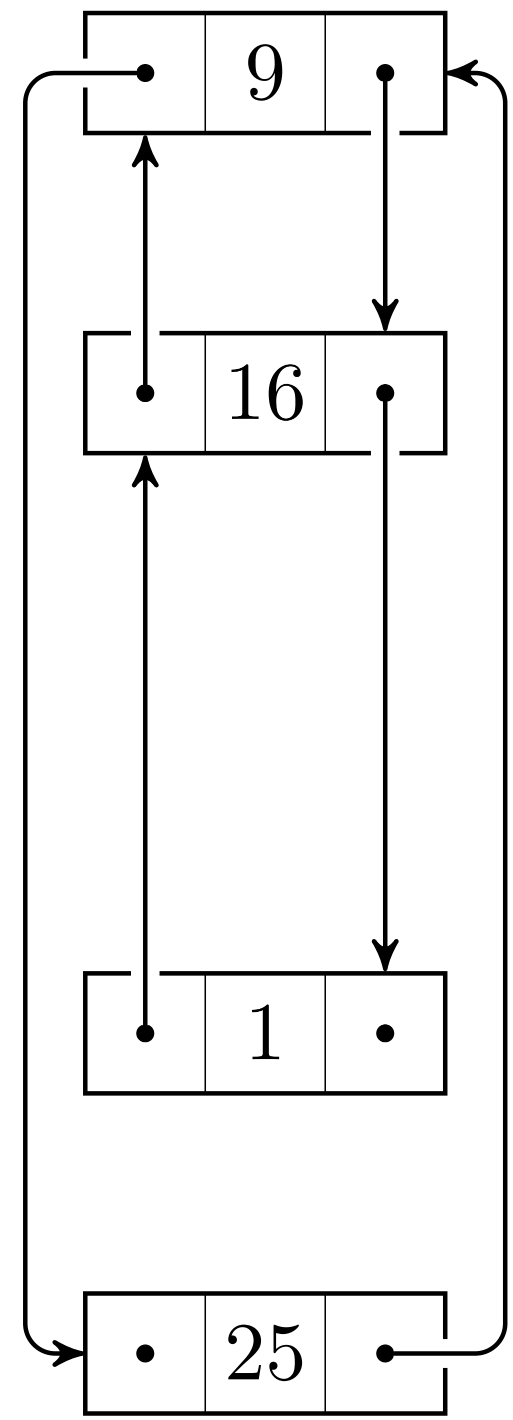
List-Delete(L, x)

if $x.prev \neq$ NIL
  $x.prev.next = x.next$
else $\mathrm{L}.head = x.next$
if $x.next \neq$ NIL
  $x.next.prev = x.prev$



List-Delete(L, x)

if $x.prev \neq$ NIL
    $x.prev.next = x.next$
else $L.head = x.next$
if $x.next \neq$ NIL
    $x.next.prev = x.prev$



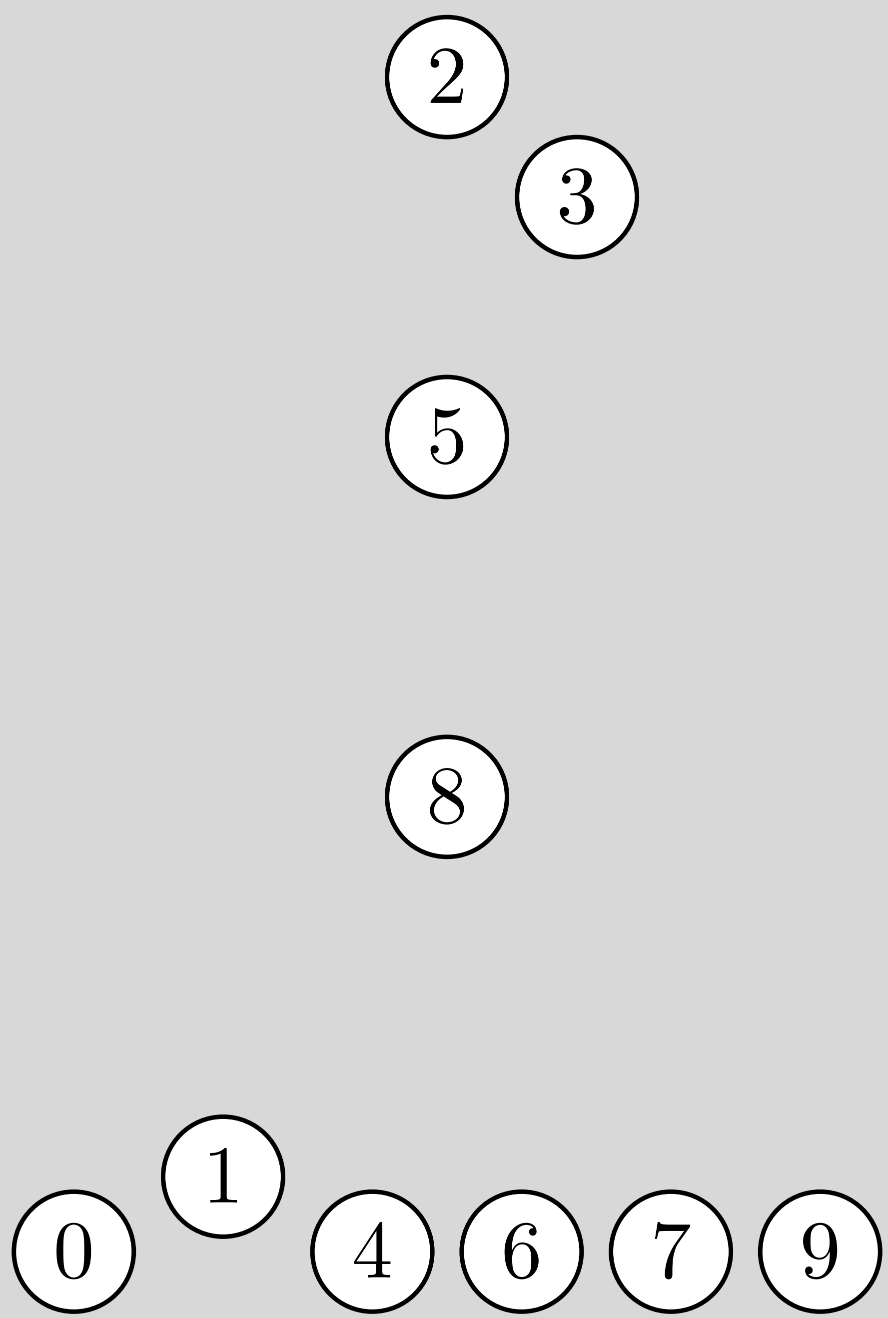
3:4

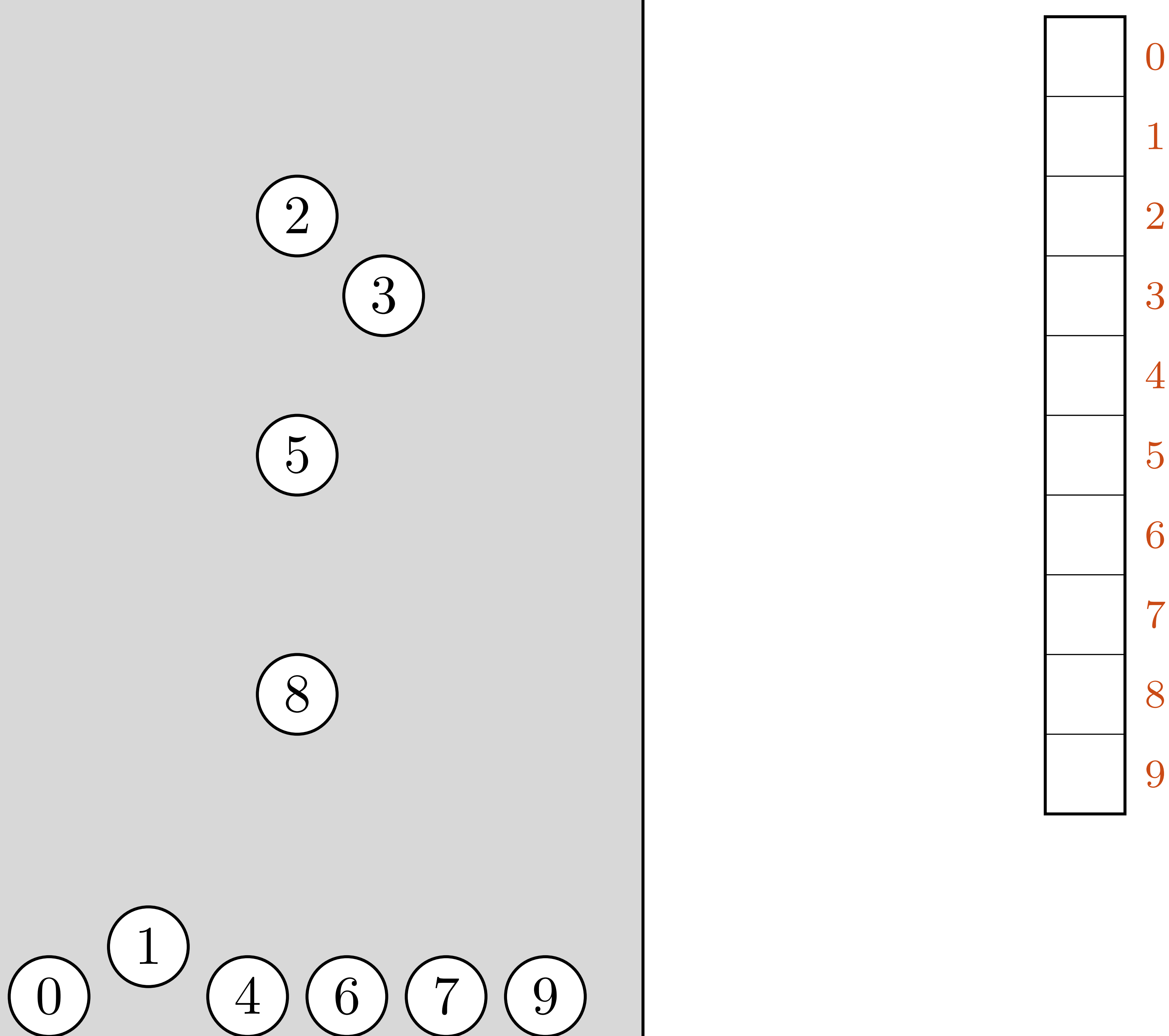
Hashtabeller

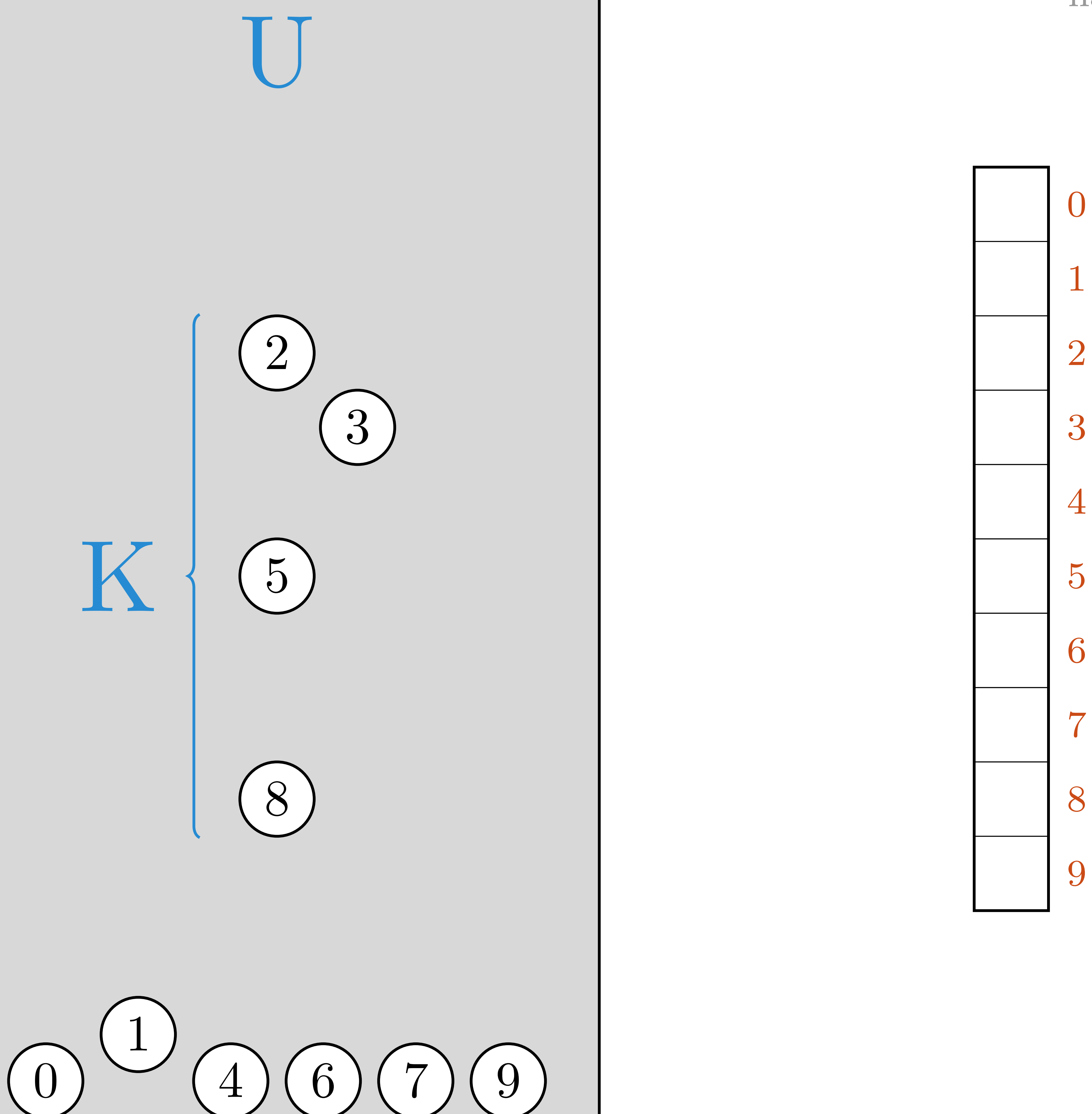
Direkte adressering

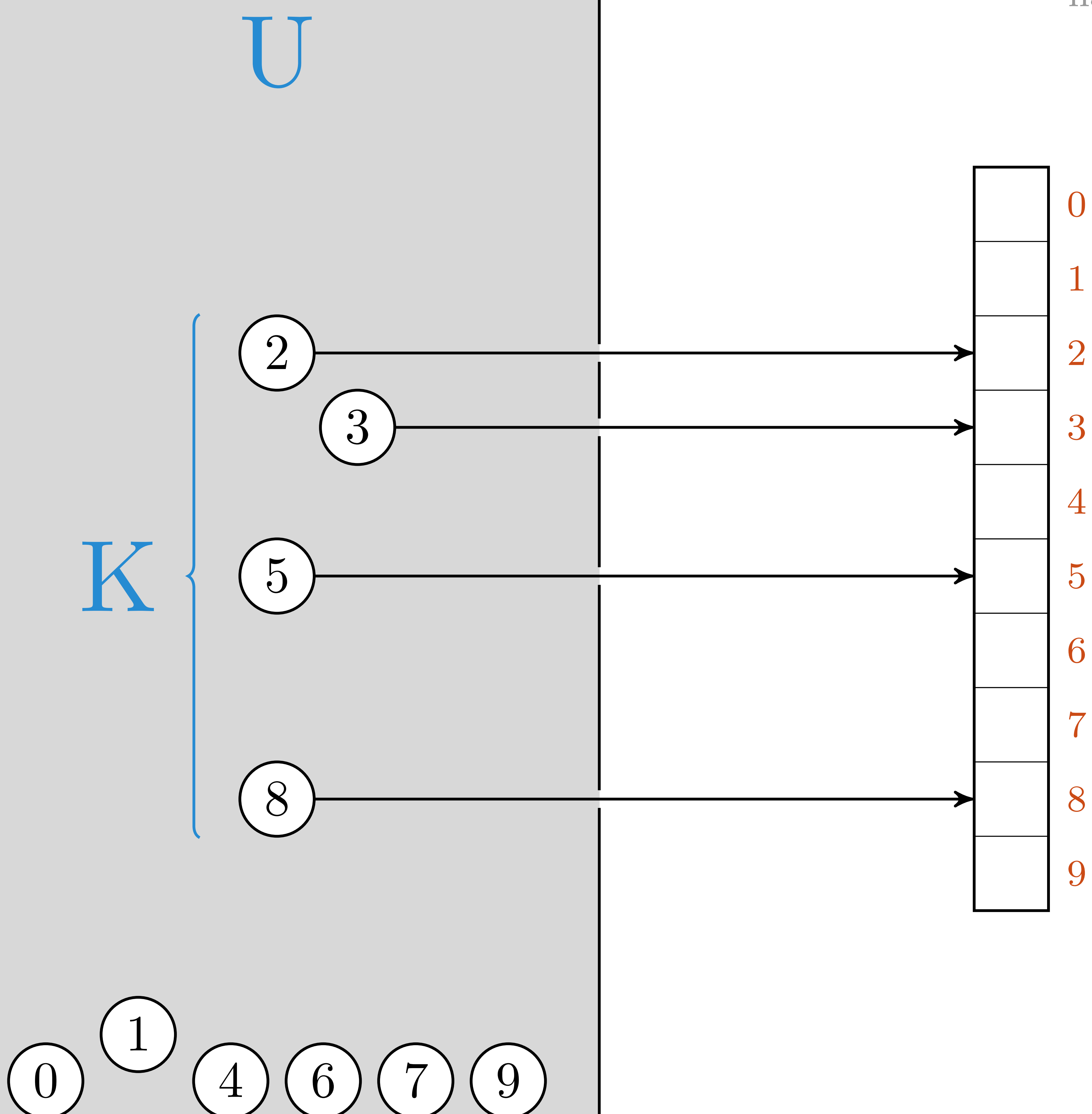
Nøkkel = indeks

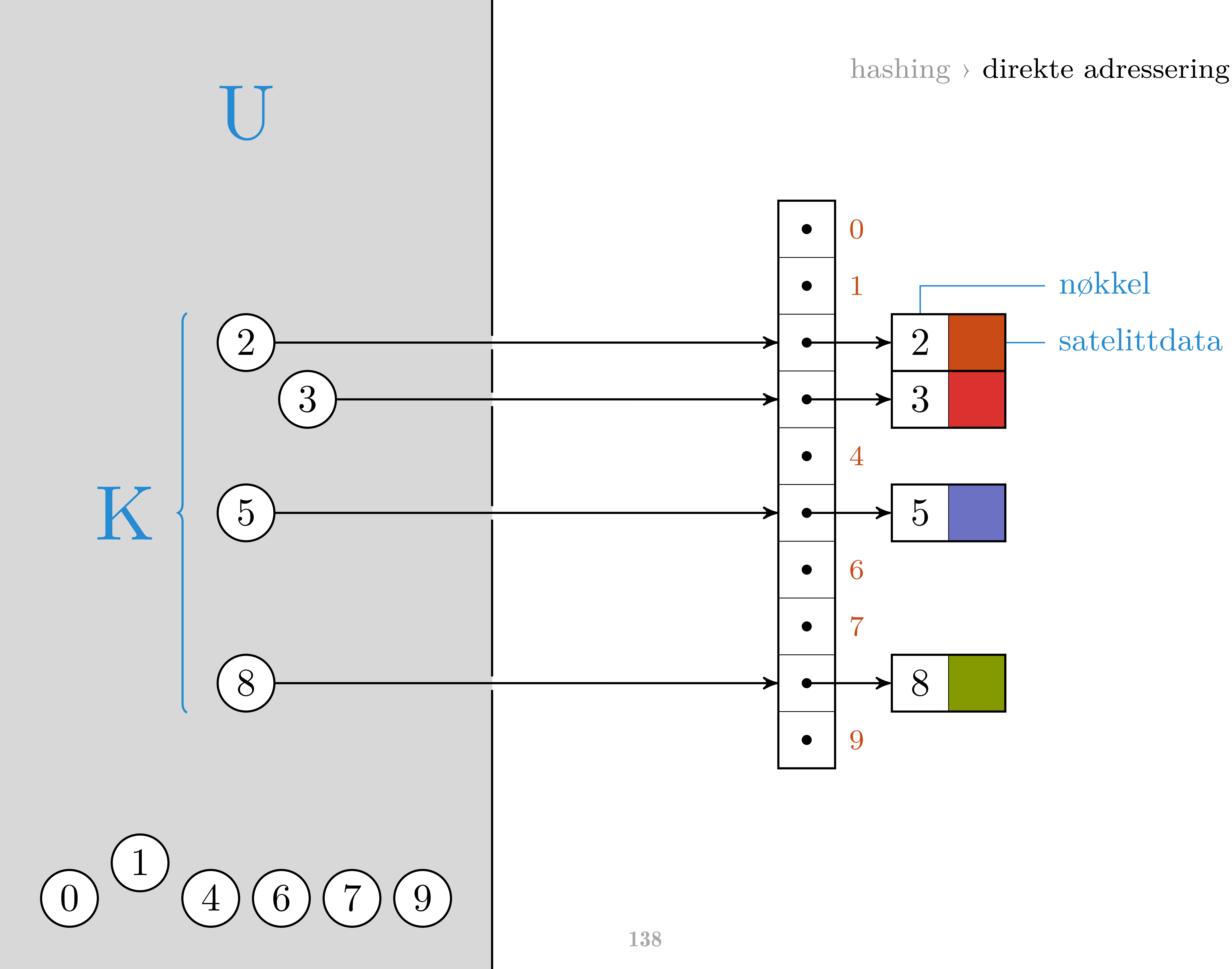
Egentlig bare en «myk start» på hashing. Vi har en verdi k som vi bruker som nøkkel i en oppslagstabell. Direkte adressering er å bare bruke k som indeks, direkte.


2
3
5
8
1
0
4
6
7
9


2
3
5
8
0
1
4
6
7
9
0
1
2
3
4
5
6
7
8
9


U
K
2
3
5
8
0
1
4
6
7
9
0
1
2
3
4
5
6
7
8
9


U
K
2
3
5
8
0
1
4
6
7
9
0
1
2
3
4
5
6
7
8
9


U
K
2
3
5
8
0
1
4
6
7
9
nøkkel
satelittdata

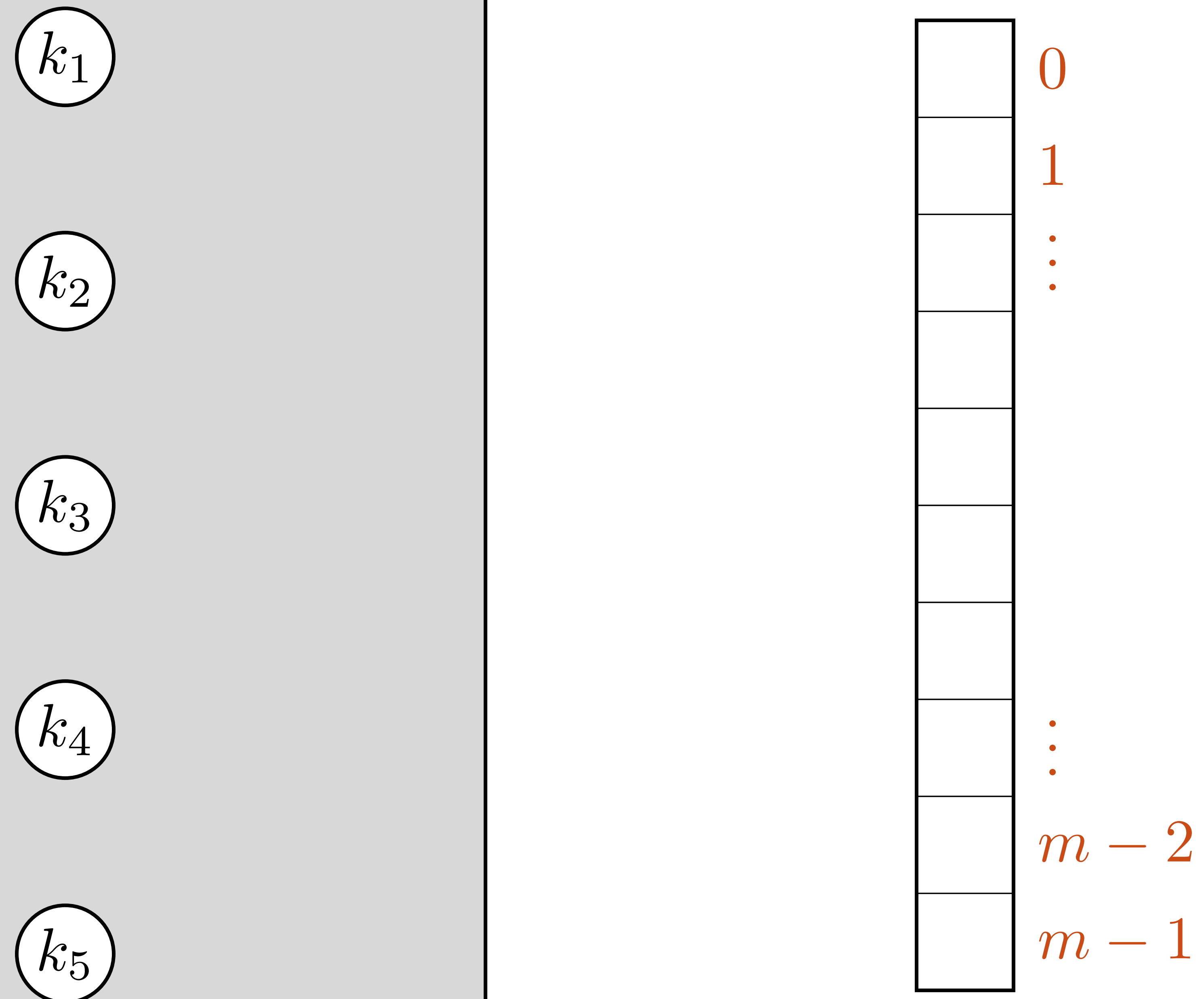
Vi ønsker å tillate store nøkler uten å ha store tabeller. Vi kan da «knøvle» nøkkelen til å bli en akseptabel, tilsynelatende tilfeldig, indeks.

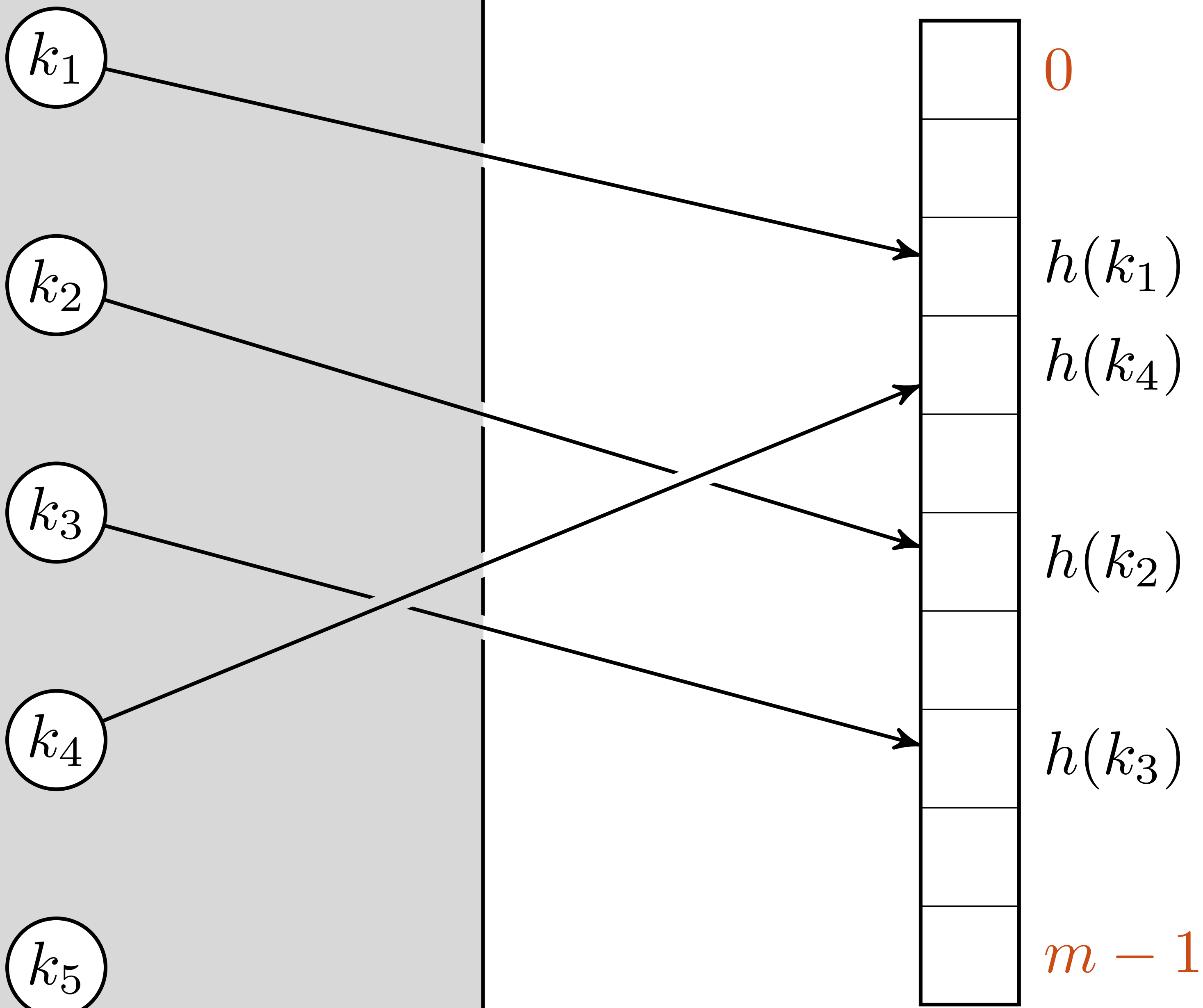
Hashtabeller

Modifisert nøkkel er indeks

Obs: Input til en hashfunksjon kan være en vilkårlig bitstreng, tolket som et tall. Vi kan godt hashe andre ting som strenger og mer kompliserte objekter.

Vi stapper objektene inn i hashfunksjonen og får en gyldig indeks ut.


k_1
k_2
k_3
k_4
k_5
0
1
⋮
⋮
$m-2$
$m-1$



Hashing: Regn ut en indeks fra nøkkelverdien.

$$h(k) = \lfloor km \rfloor \quad 0 \leq k < 1$$

$$h(k) = \lfloor km \rfloor \quad 0 \le k < 1$$

$$h(k) = k \bmod m$$

$$h(k) = \lfloor km \rfloor \quad 0 \le k < 1$$

$$h(k) = k \bmod m$$

$$h(k) = \lfloor m(k\mathrm{A} \bmod 1) \rfloor$$

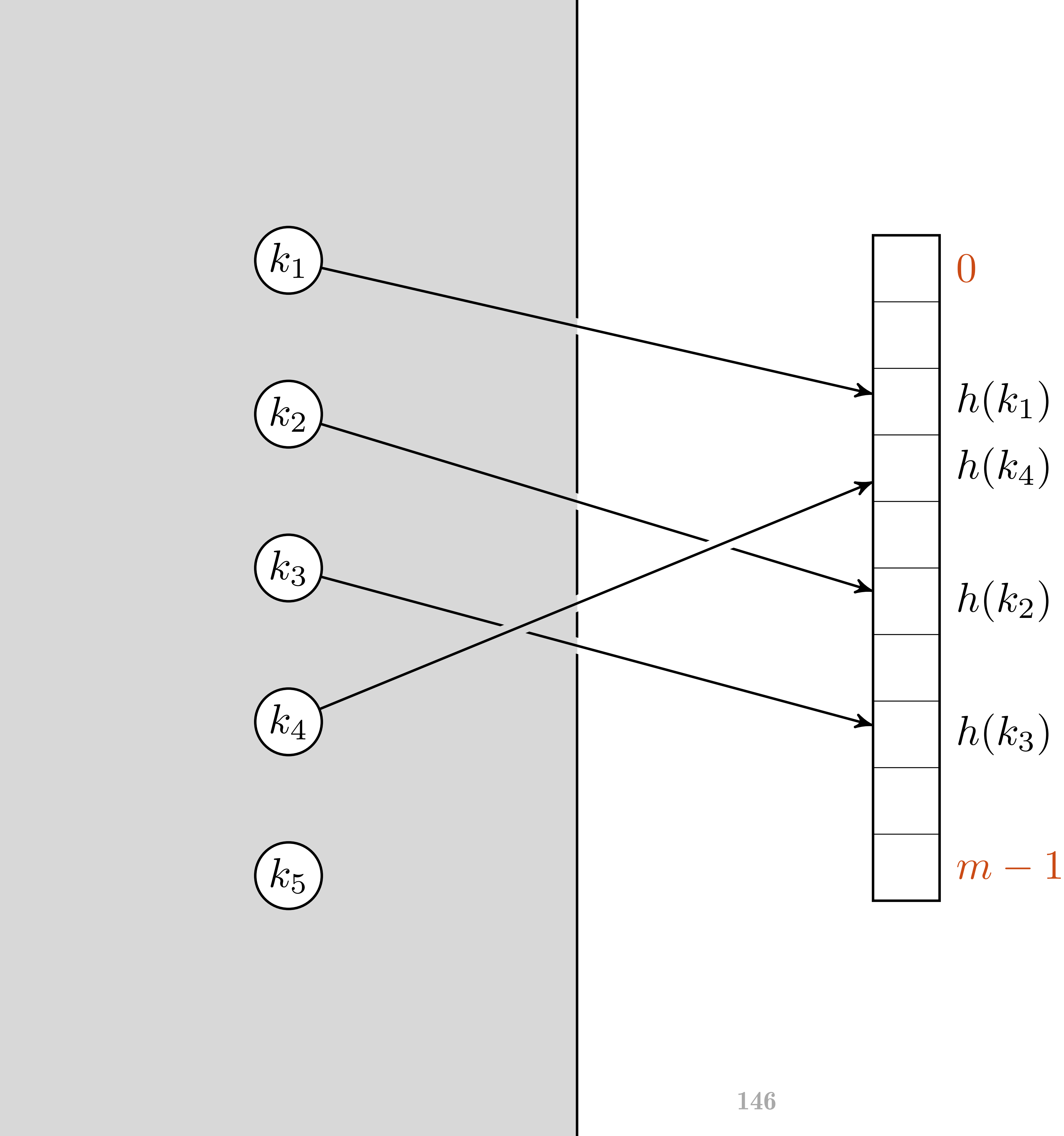
$$0 < \mathrm{A} < 1$$

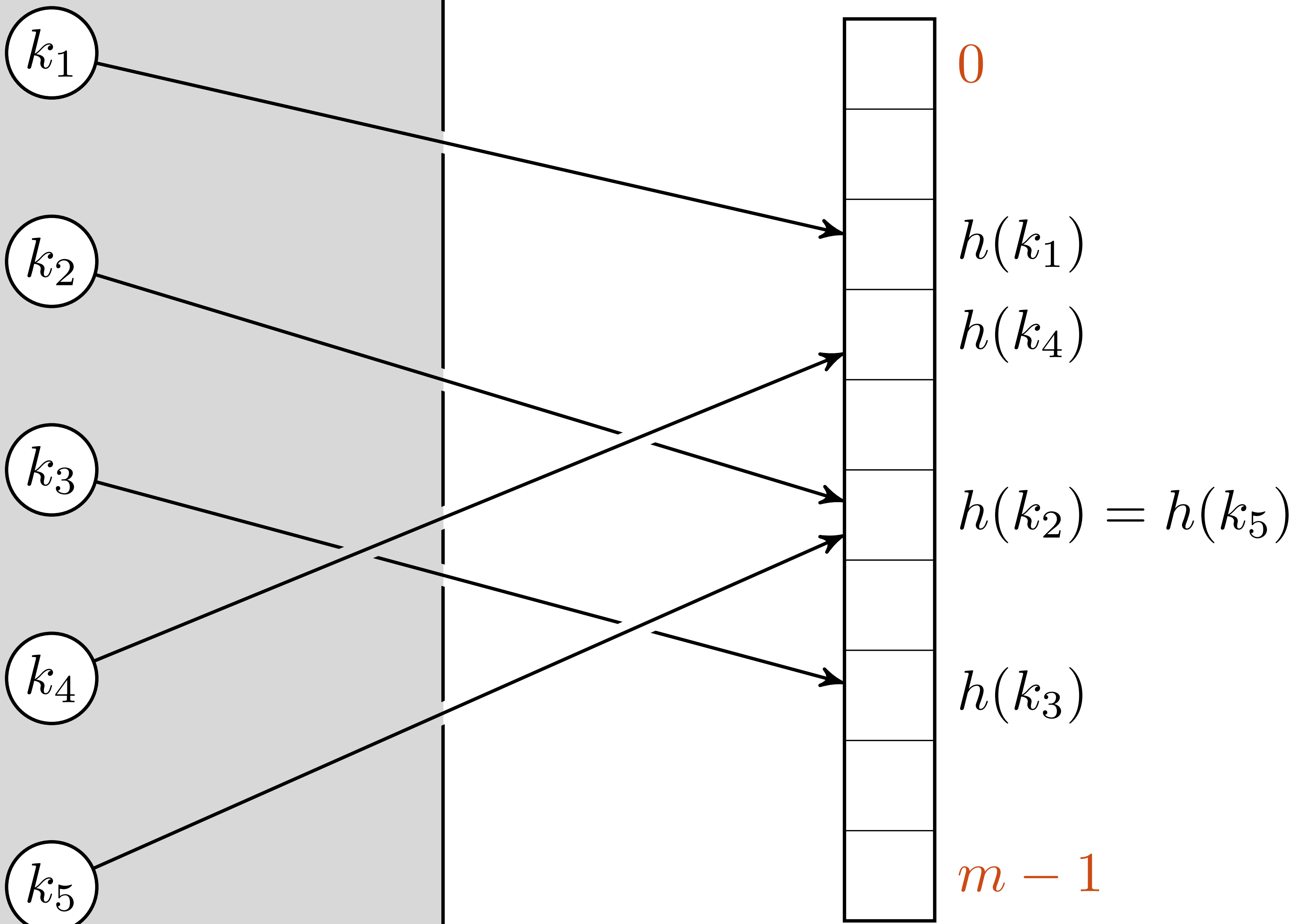
f.eks.

Strenger e.l.?

Regn ut et heltall

F.eks. behandle tegn som siffer

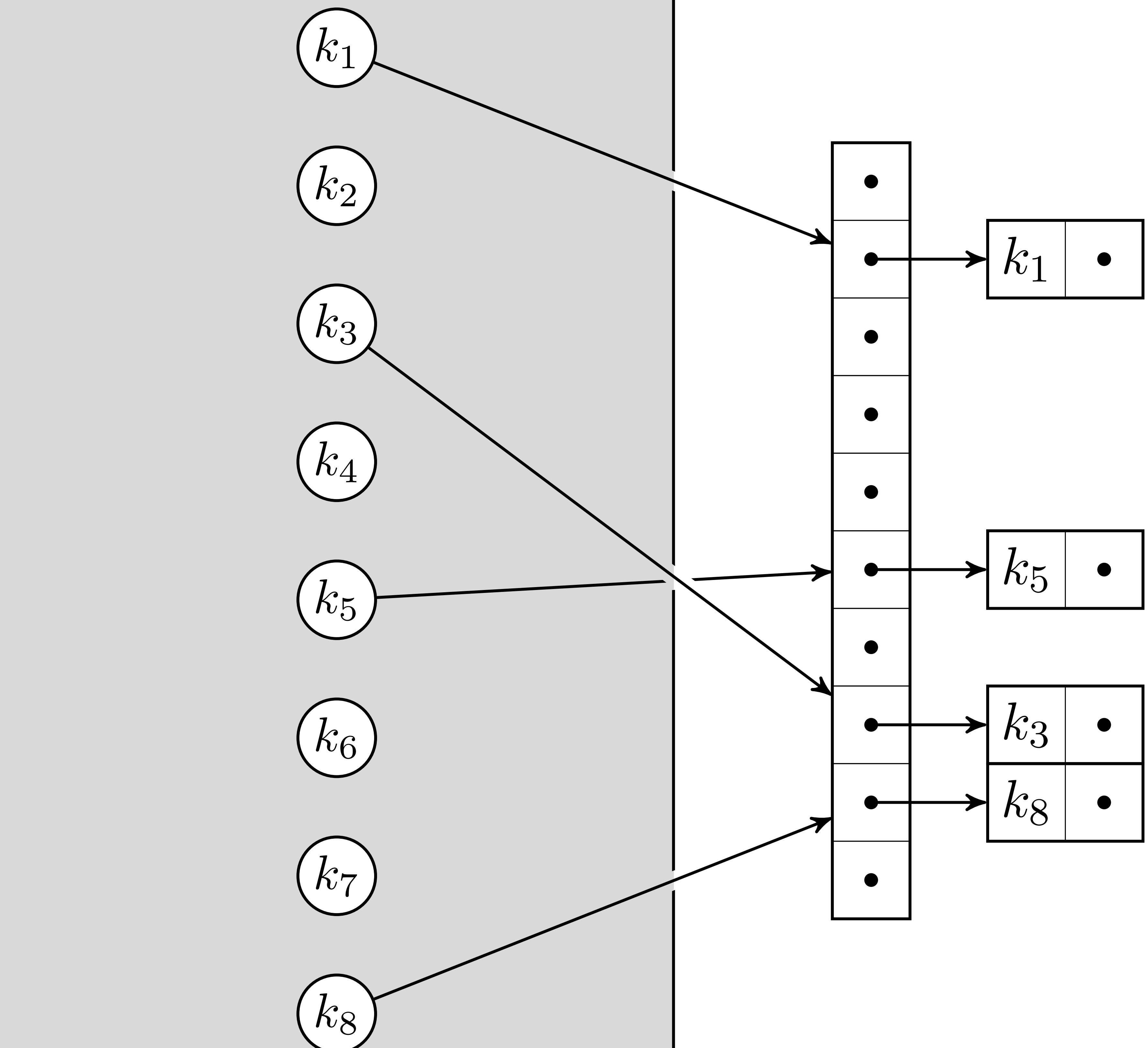

k_1
k_2
k_3
k_4
k_5
0
$h(k_1)$
$h(k_4)$
$h(k_2)$
$h(k_3)$
$m-1$


k_1
k_2
k_3
k_4
k_5
0
$h(k_1)$
$h(k_4)$
$h(k_2) = h(k_5)$
$h(k_3)$
$m - 1$

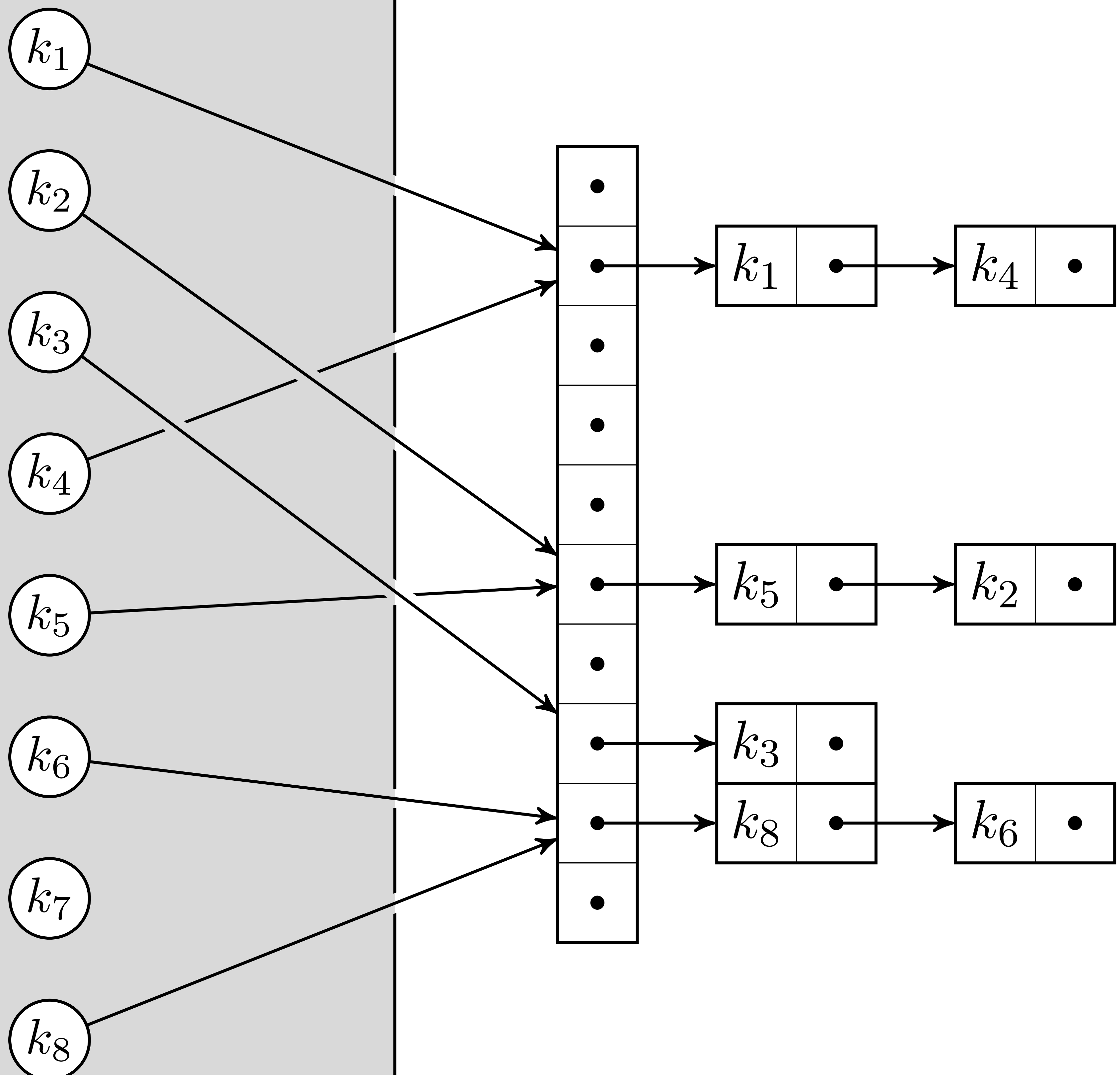
Kjeding (chaining)

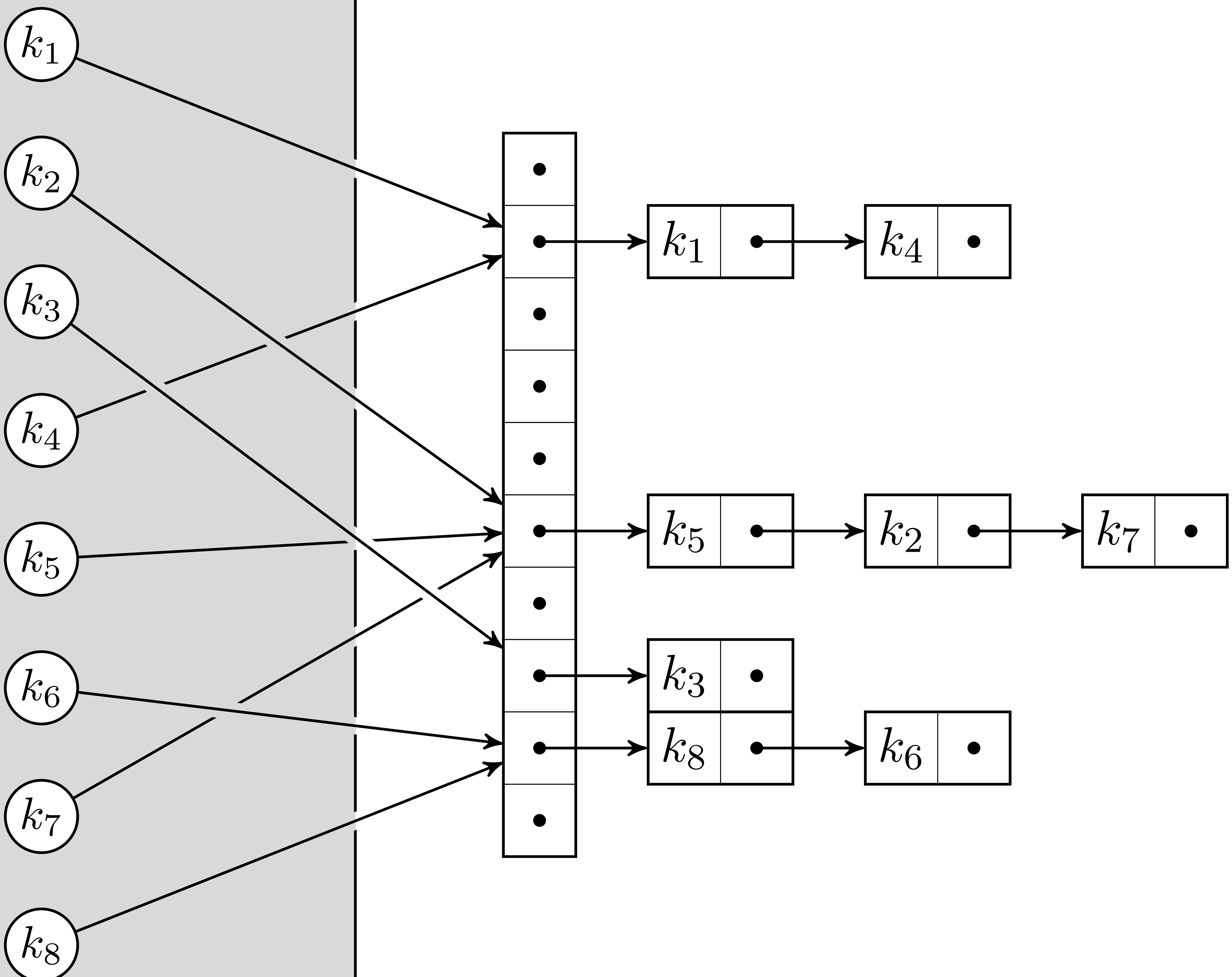
Hver posisjon har en liste

Om to verdier hasher til samme indeks, så har vi en kollisjon; for å ta vare på begge verdiene, kan vi ha en lenket liste (f.eks.) i hver celle i tabellen.



k_1
k_2
k_3
k_4
k_5
k_6
k_7
k_8
k_1
k_5
k_3
k_8


k_1
k_2
k_3
k_4
k_5
k_6
k_7
k_8
k_1
k_4
k_5
k_2
k_3
k_8
k_6


k_1
k_2
k_3
k_4
k_5
k_6
k_7
k_8
k_1
k_4
k_5
k_2
k_7
k_3
k_8
k_6

- **Mange kollisjoner: Lineært lange lister**
- **Søk vil ta lineær tid**
- **Anta lineært stor tabell**
- **Anta jevn, «tilfeldig» fordeling**
- **Konstant forventet kjøretid!**

Statisk datasett?

Lag custom hashfunksjon!

Kan da garantere konstant kjøretid

4:4

Dynamiske tabeller

- **Hva om en hashtabell blir for full?**
- **Eller hva med en stakk eller kø?**
- **Vi kan allokere nytt minne og kopiere**
- **Men det tar jo lineær tid ...**
- **... så vi vil gjøre det sjelden!**
- **Vi tar i, og allokerer mye minne**

Amortisert arbeid

- **Kjøretid for én enkelt operasjon: Ikke altid informativt**

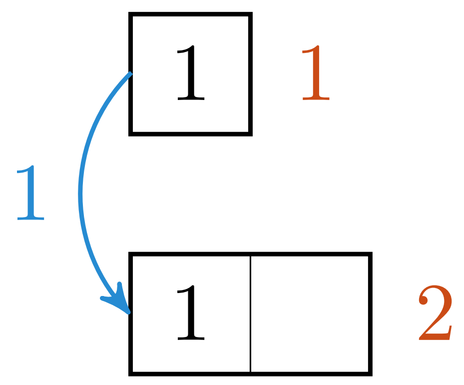
- **Se på gjennomsnitt per operasjon etter at mange har blitt utført!**

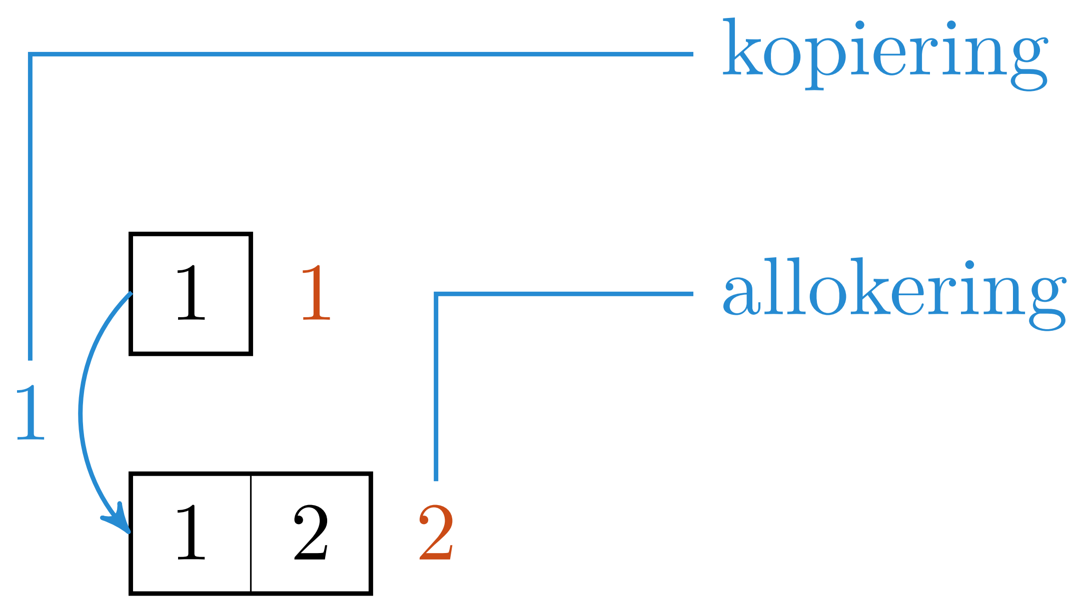
Husk denne!

$$\sum_{i=0}^{h-1} 2^i = 2^h - 1$$

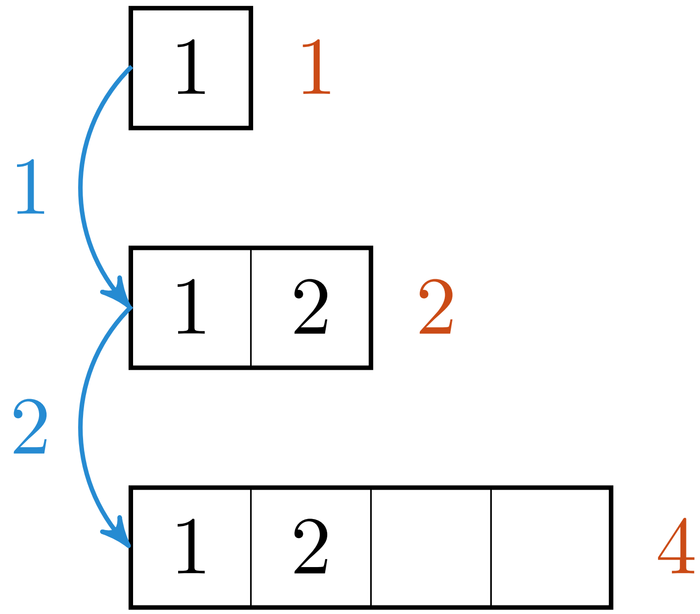
$\square$

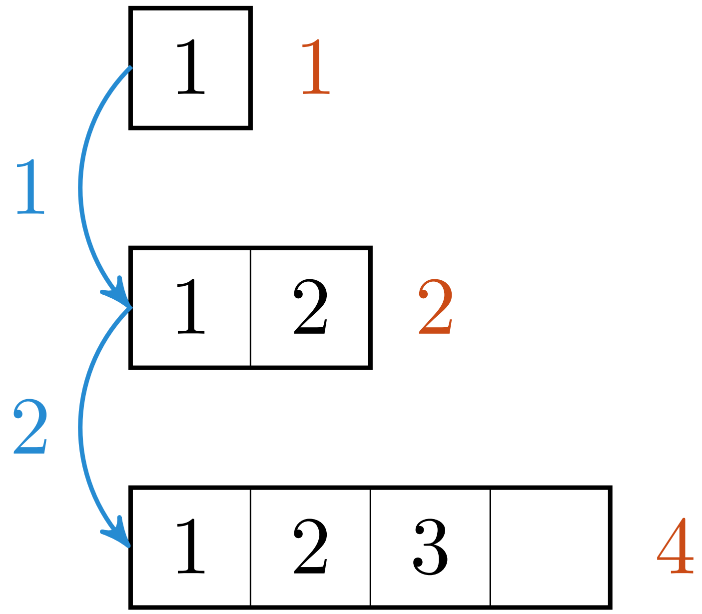
$\boxed{1}$

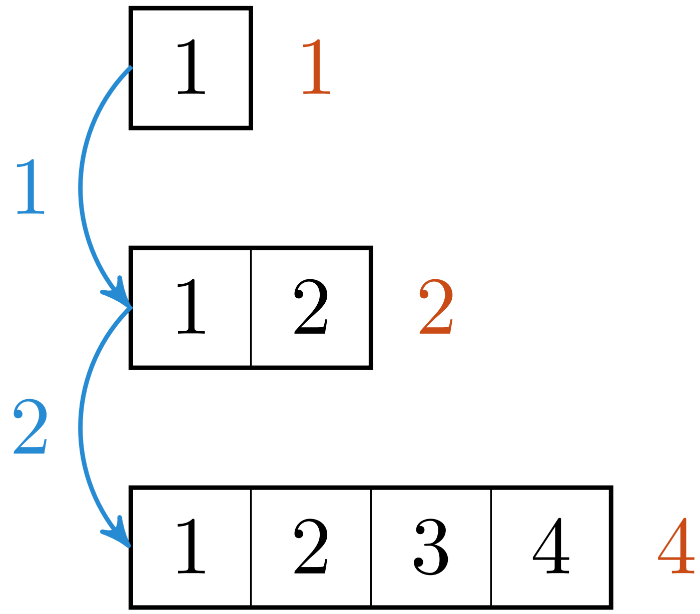

1
1
1
1
2

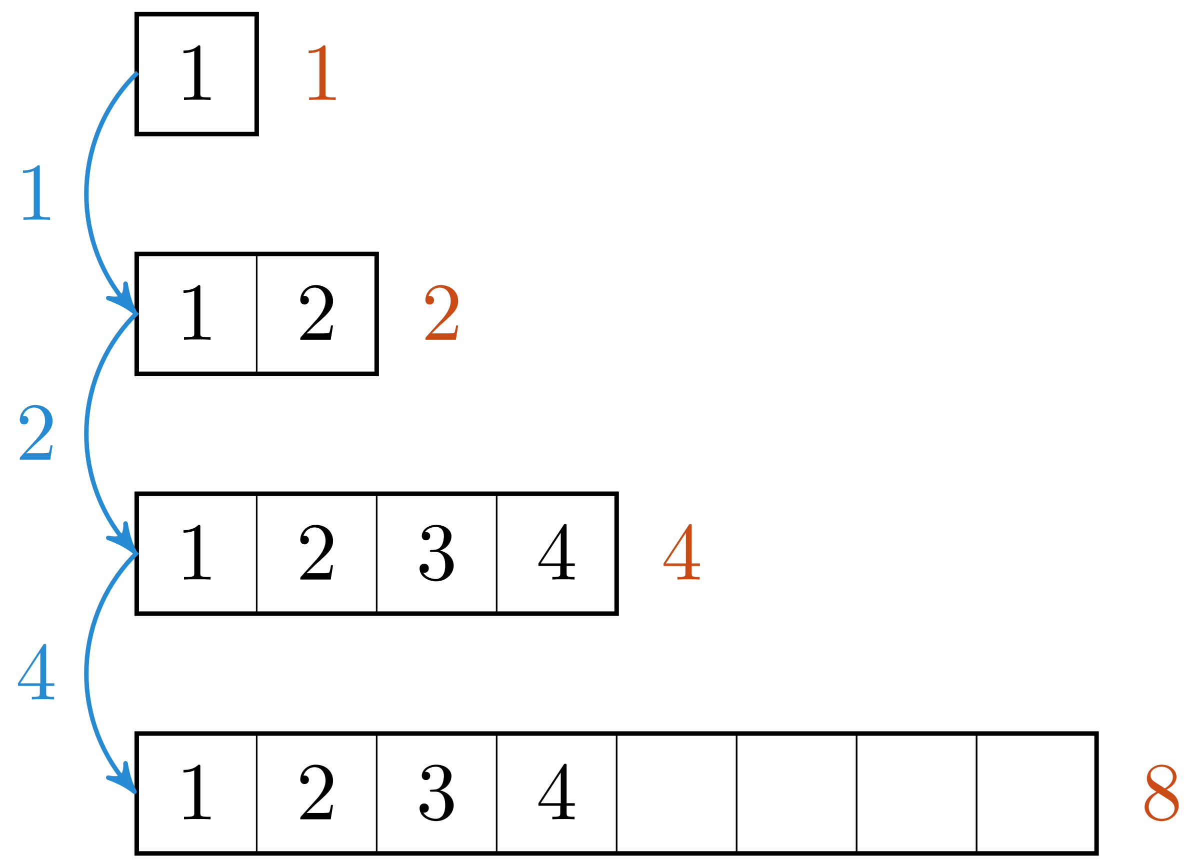


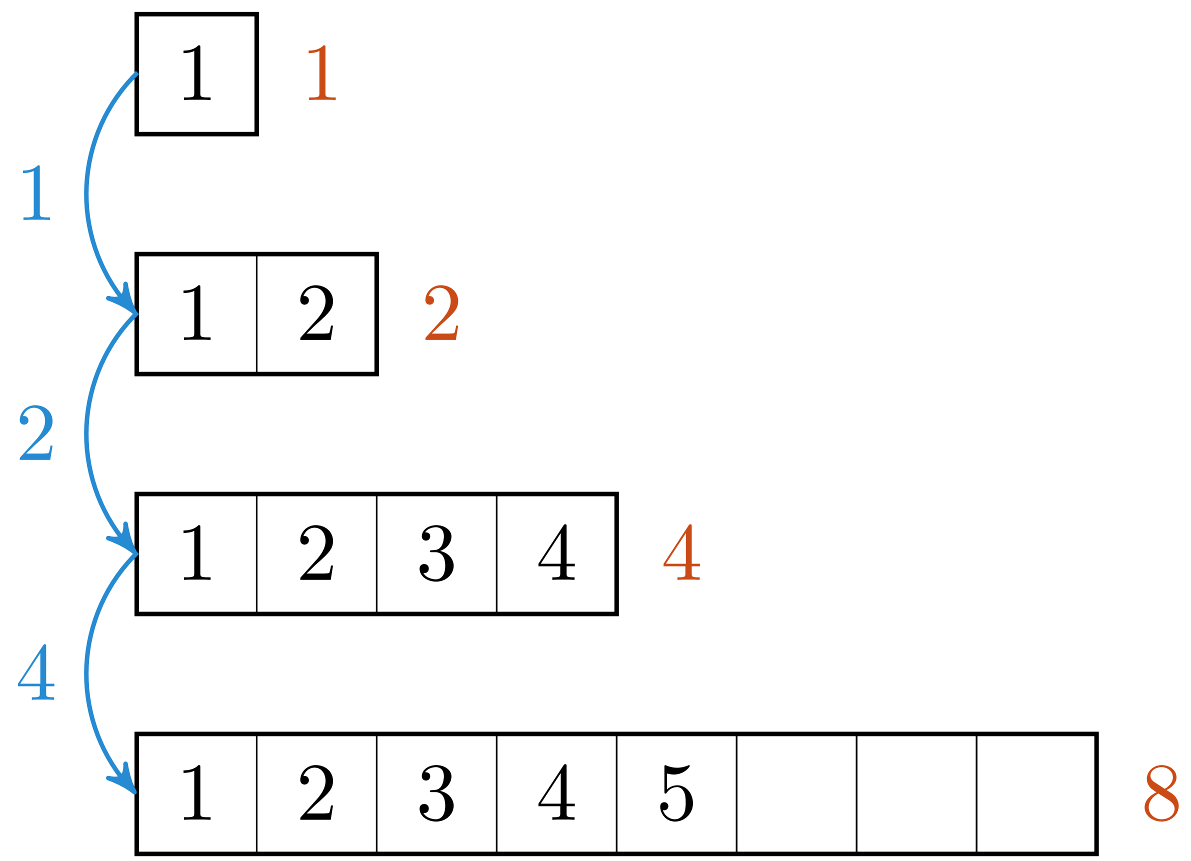
Vi bør jo ta med deallokering av den forrige tabellen òg – men det endrer ikke det asymptotiske resultatet.

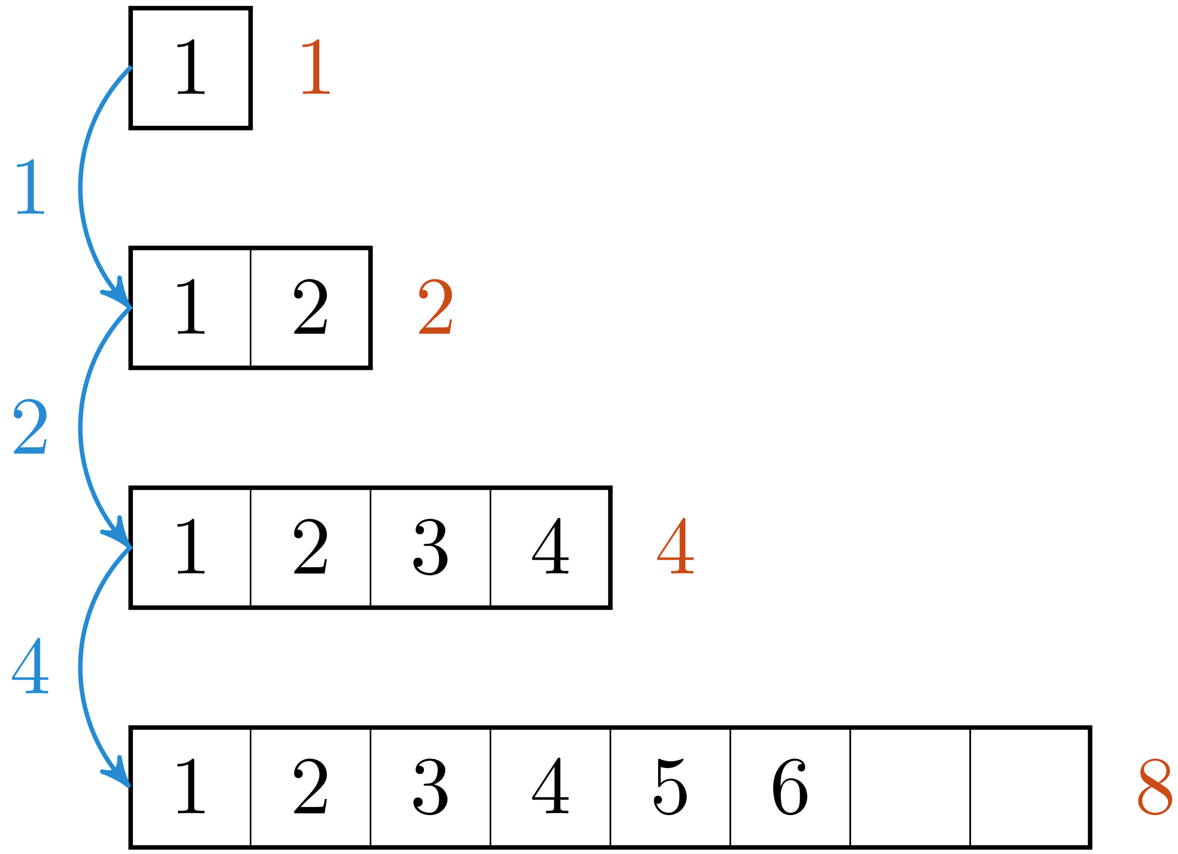

1
1
1
1 2
2
2
1 2
4

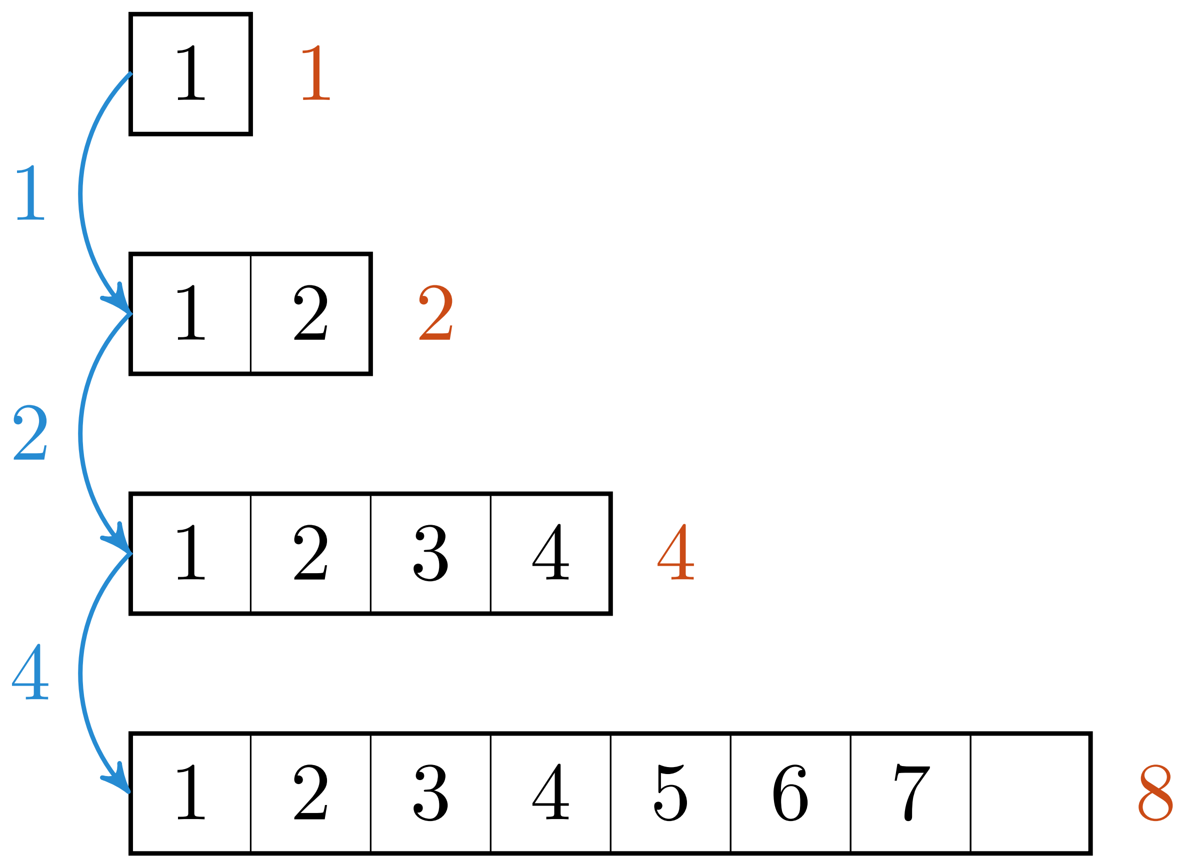

1
1
1
1 2
2
2
1 2 3
4

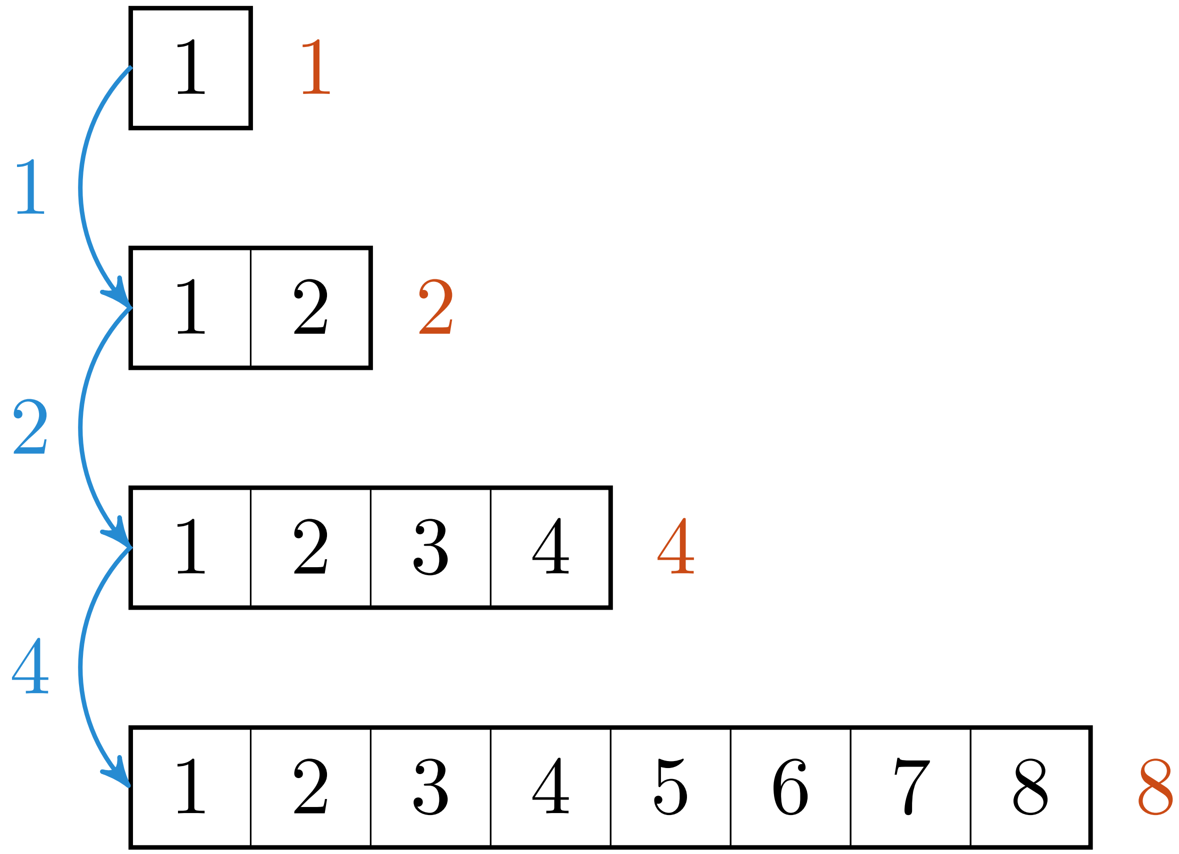

1
1
1
1 2
2
2
1 2 3 4
4

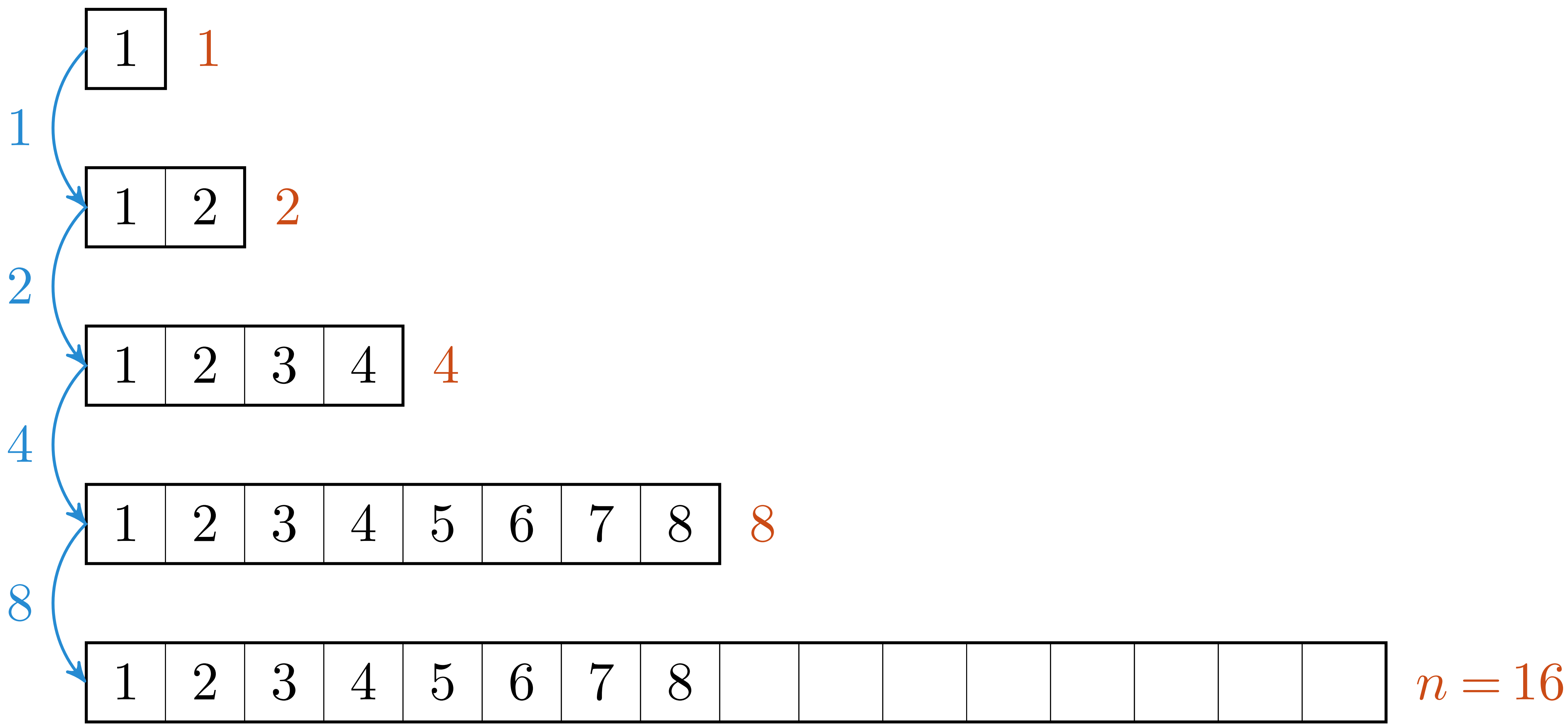

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4
8

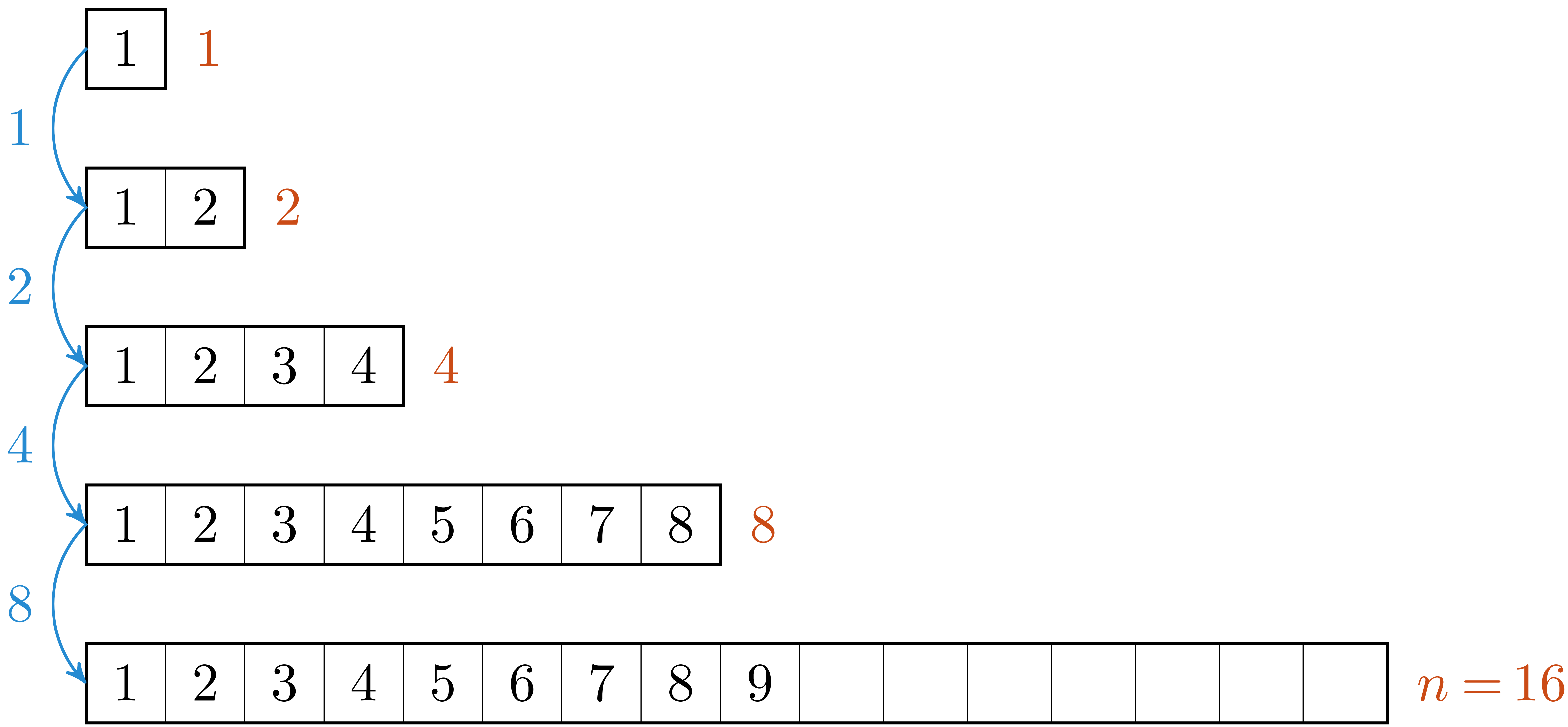

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5
8

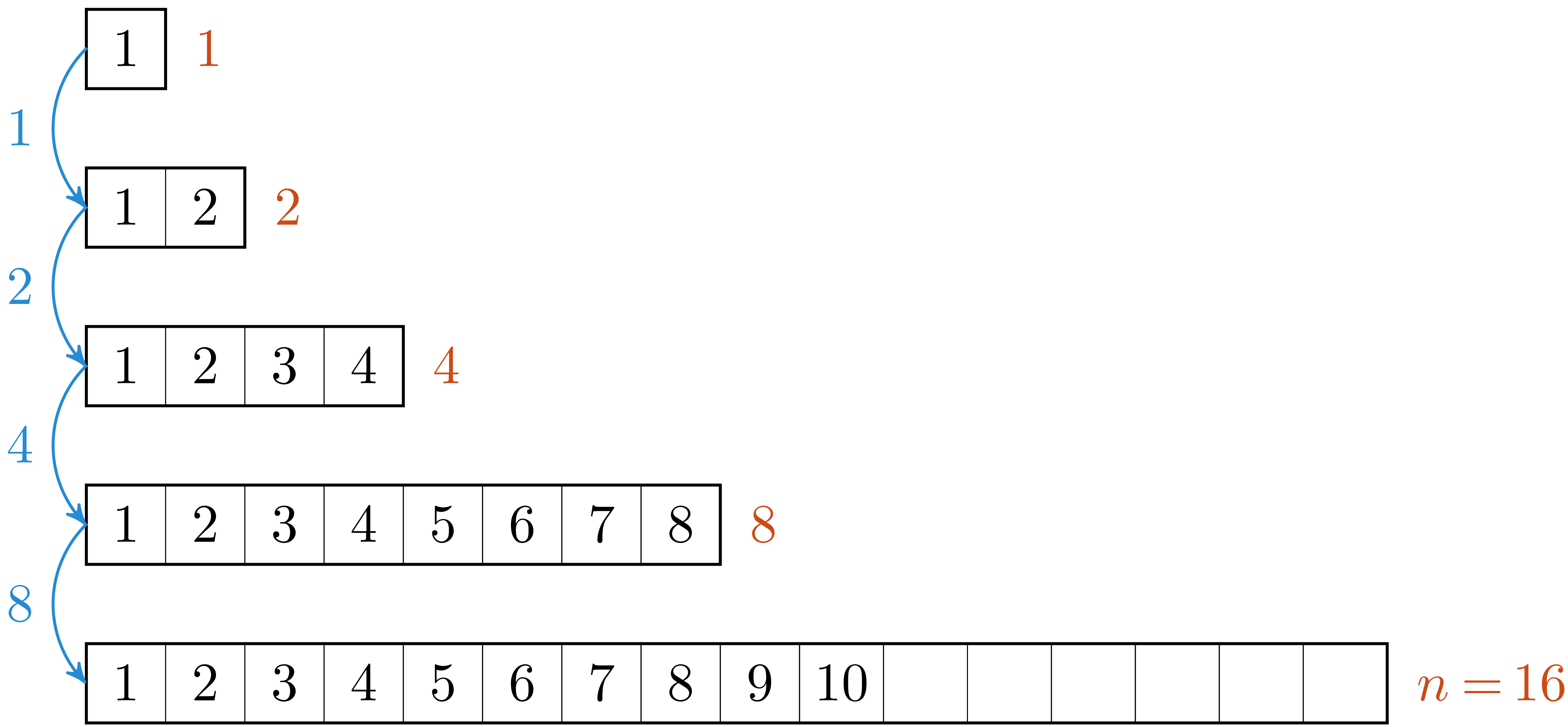

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6
8

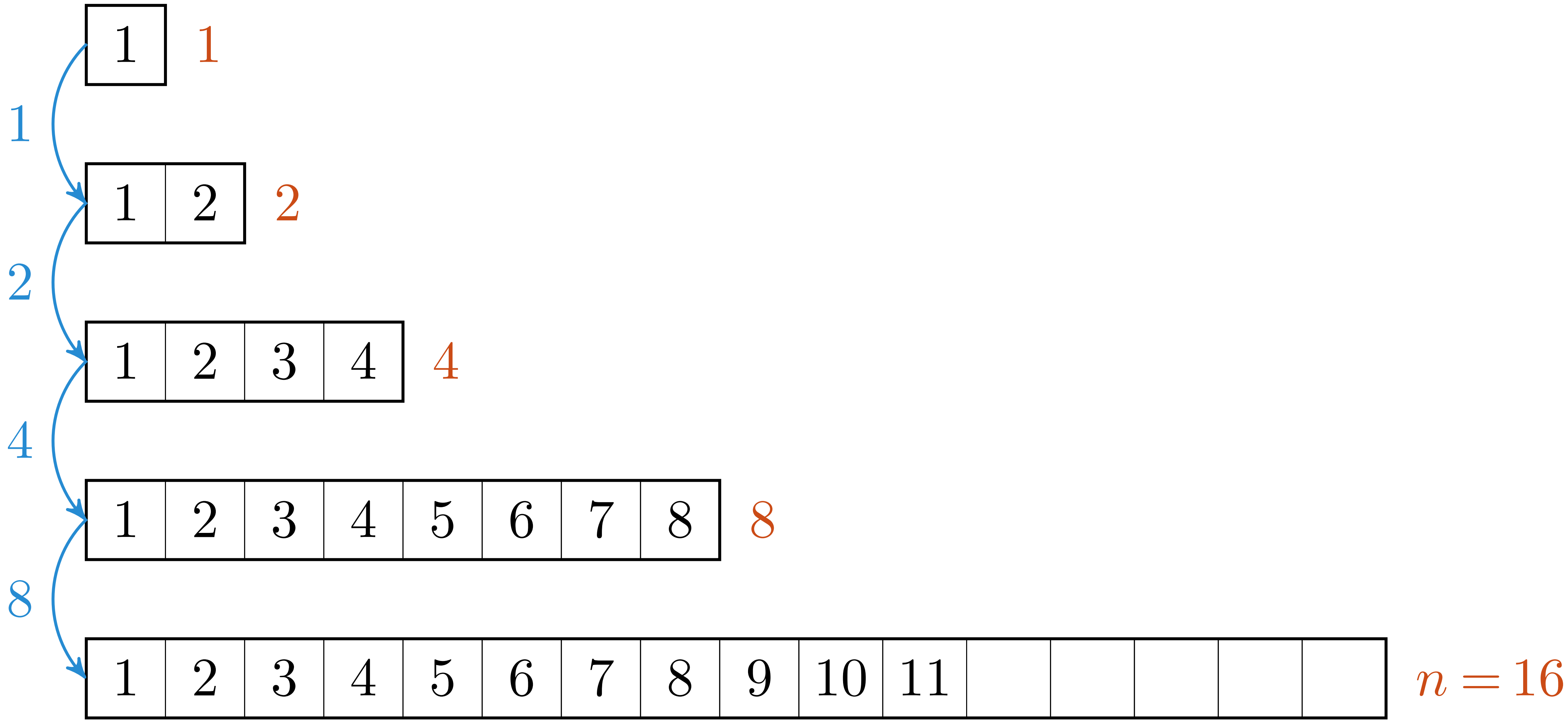

1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7
8

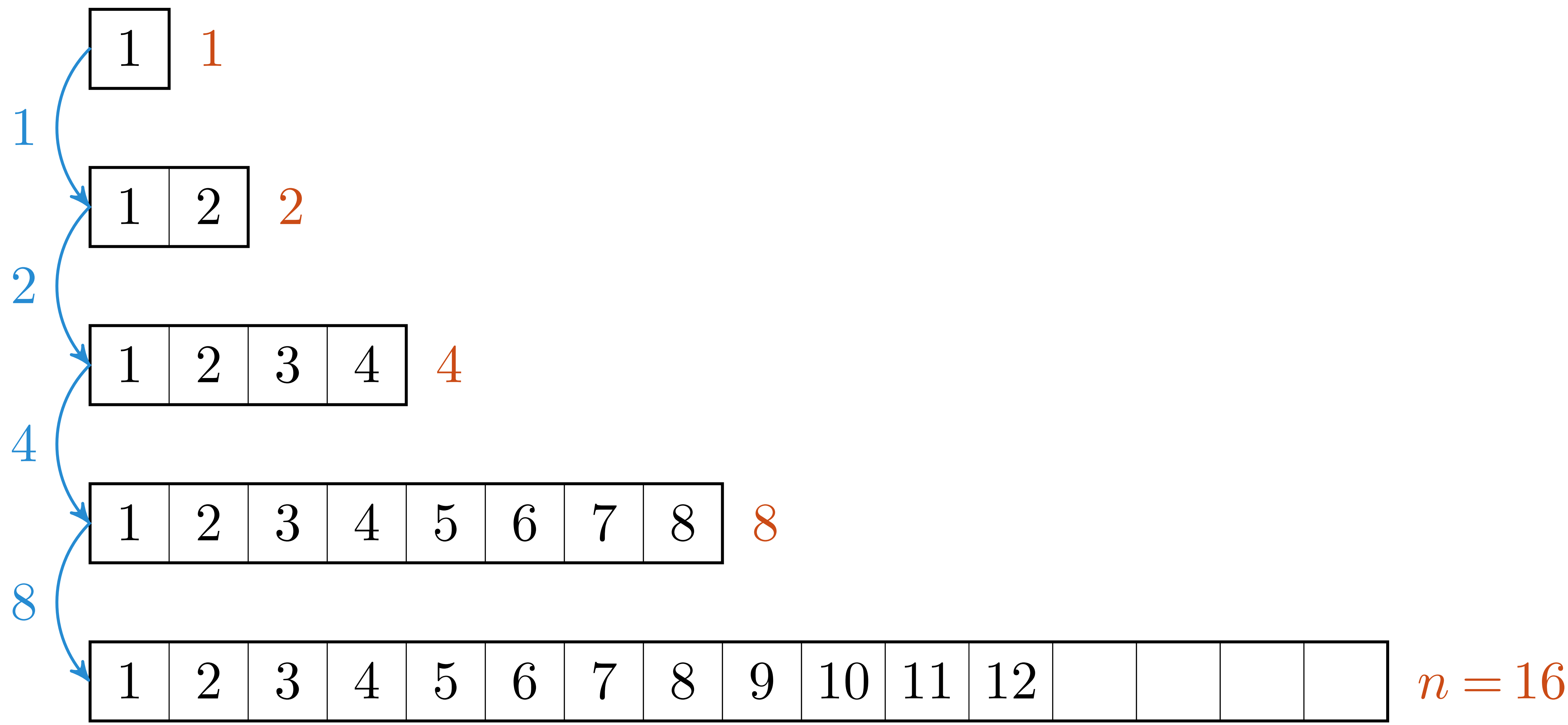

1
1
1
2
1
2
2
4
1
2
3
4
4
8
1
2
3
4
5
6
7
8

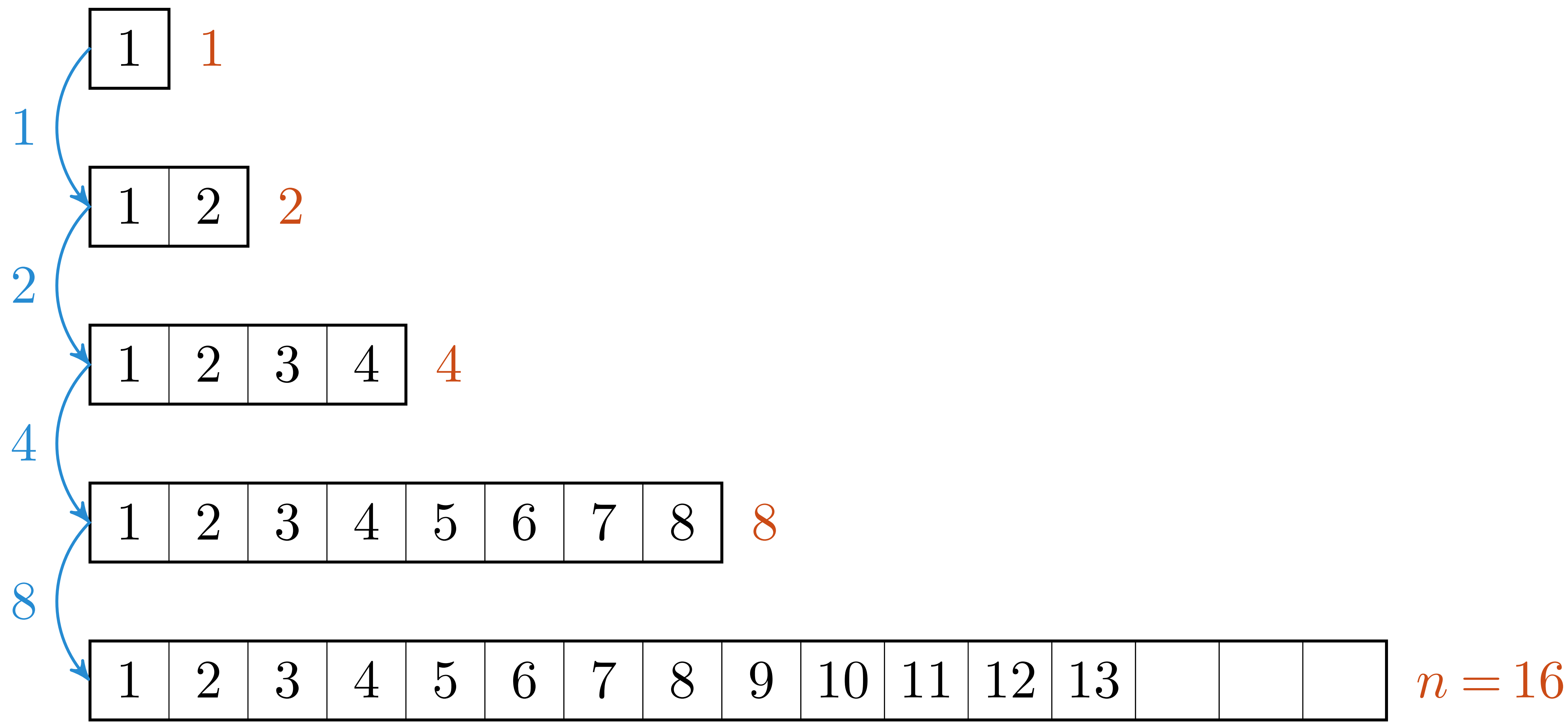

1
1
1
2
1 2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8
$n = 16$

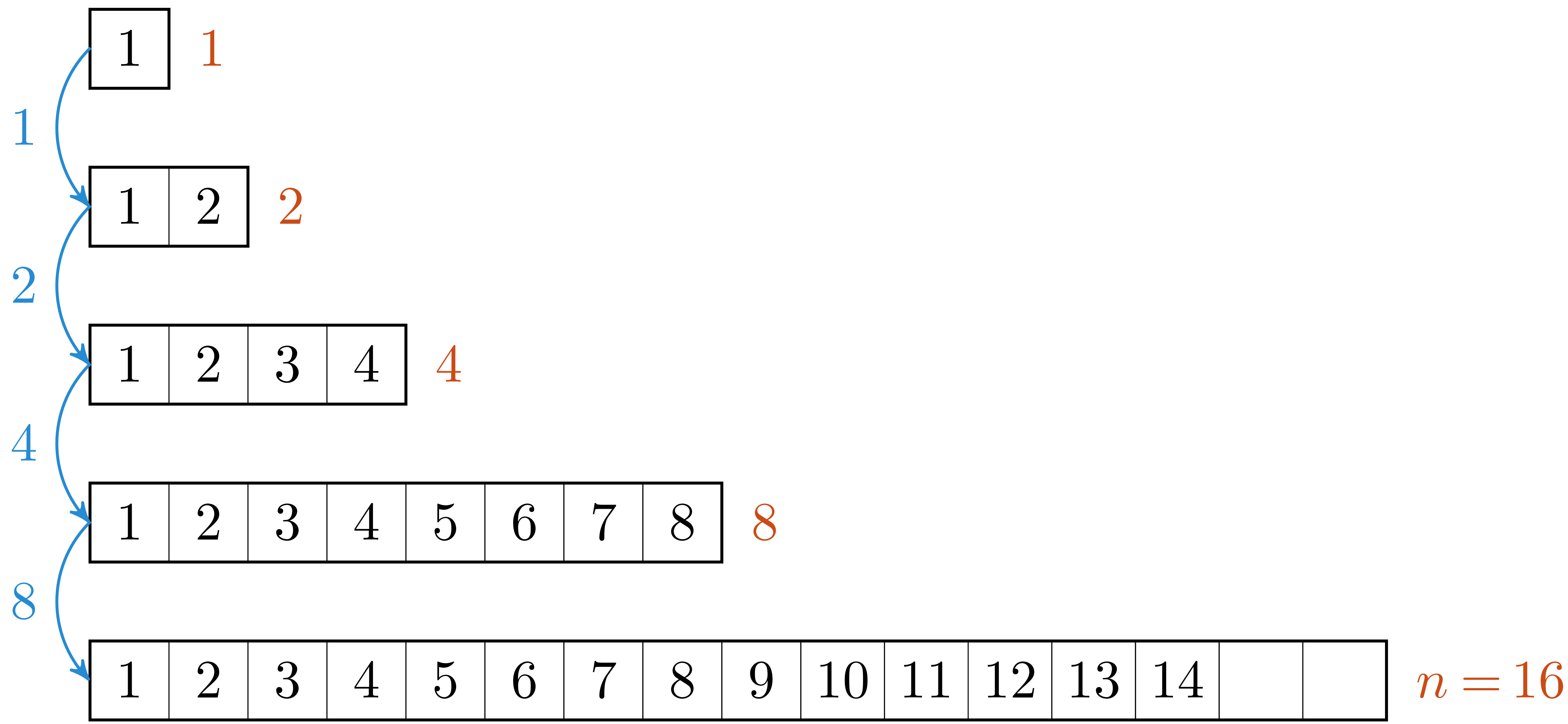

1
1
1
2
1
2
2
1
2
3
4
4
4
1
2
3
4
5
6
7
8
8
8
1
2
3
4
5
6
7
8
9
$n = 16$

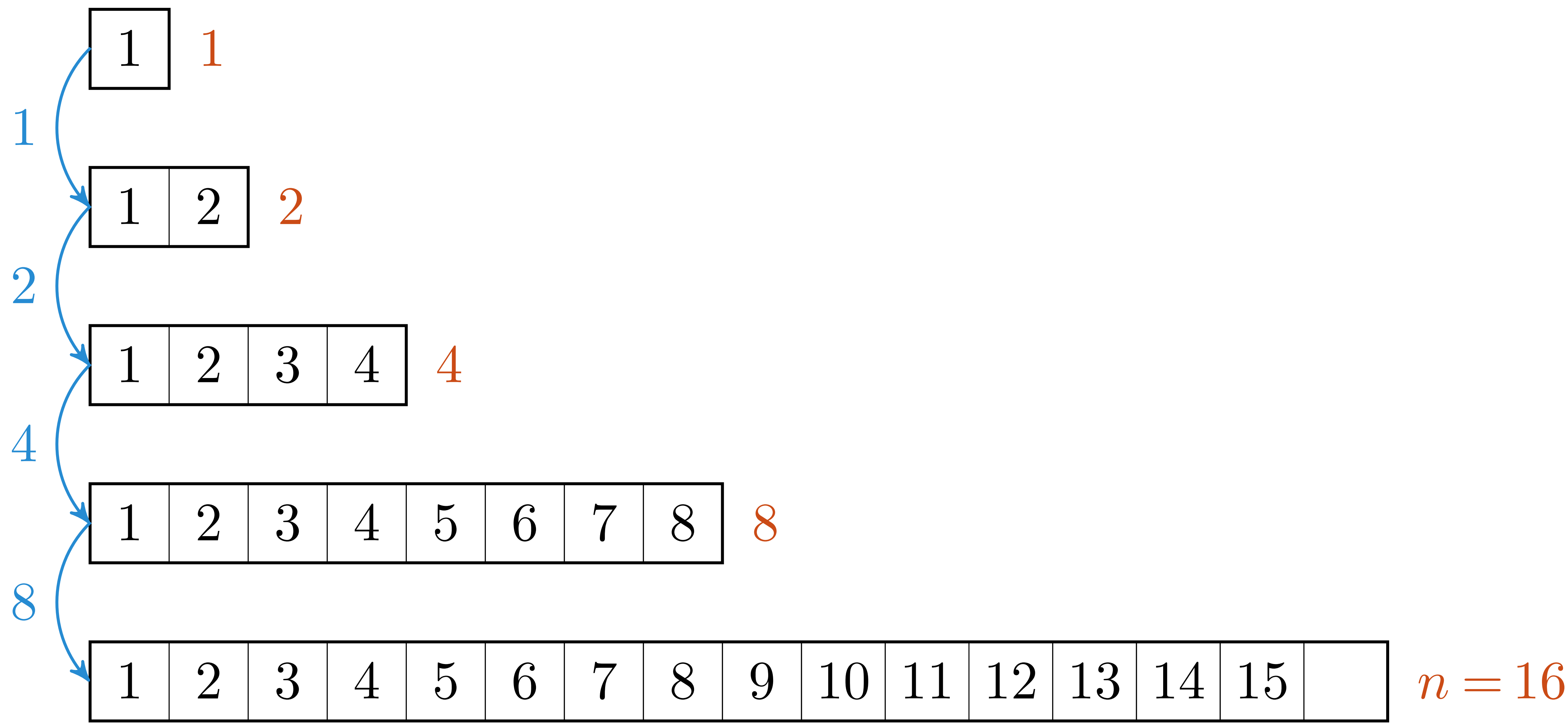

1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10
n = 16

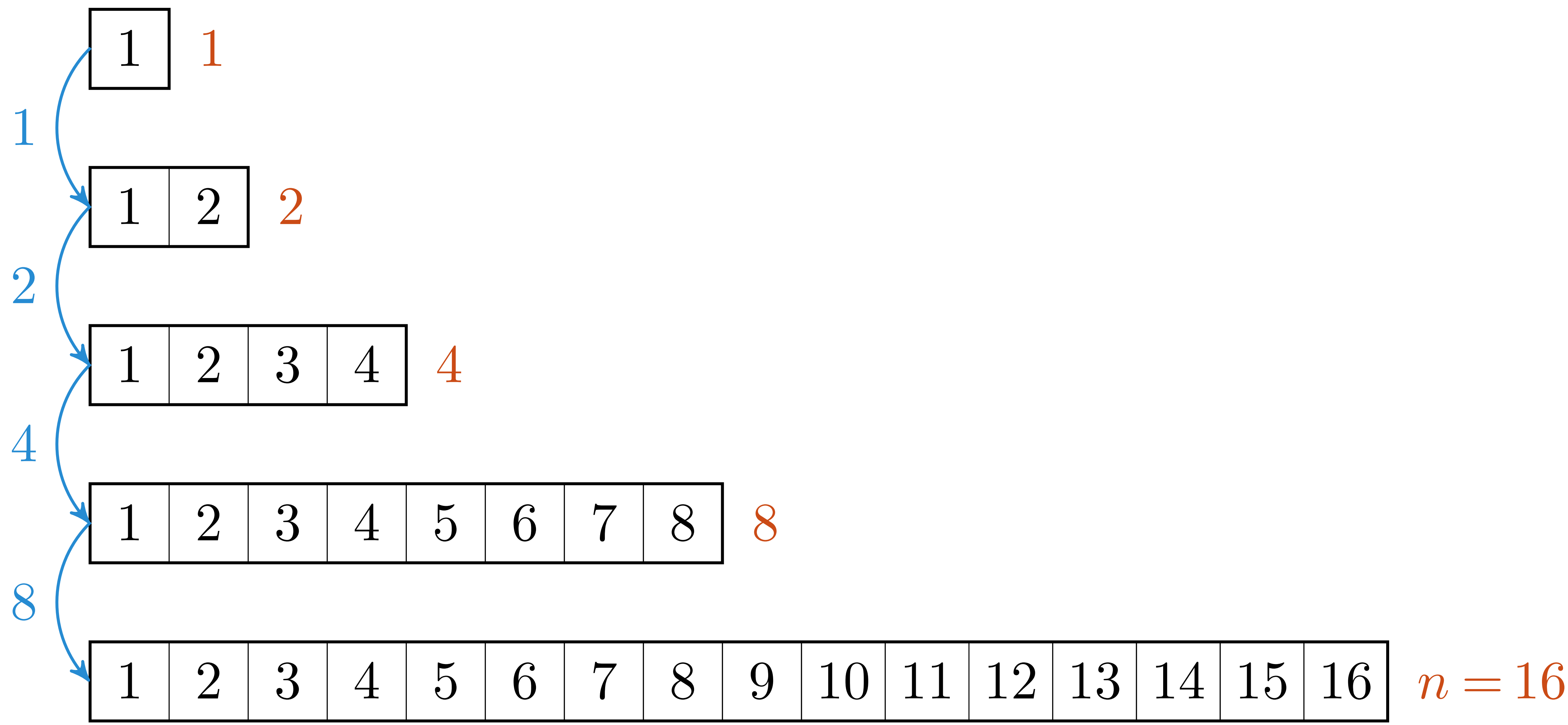

1
1
1
2
1 2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11
$n = 16$

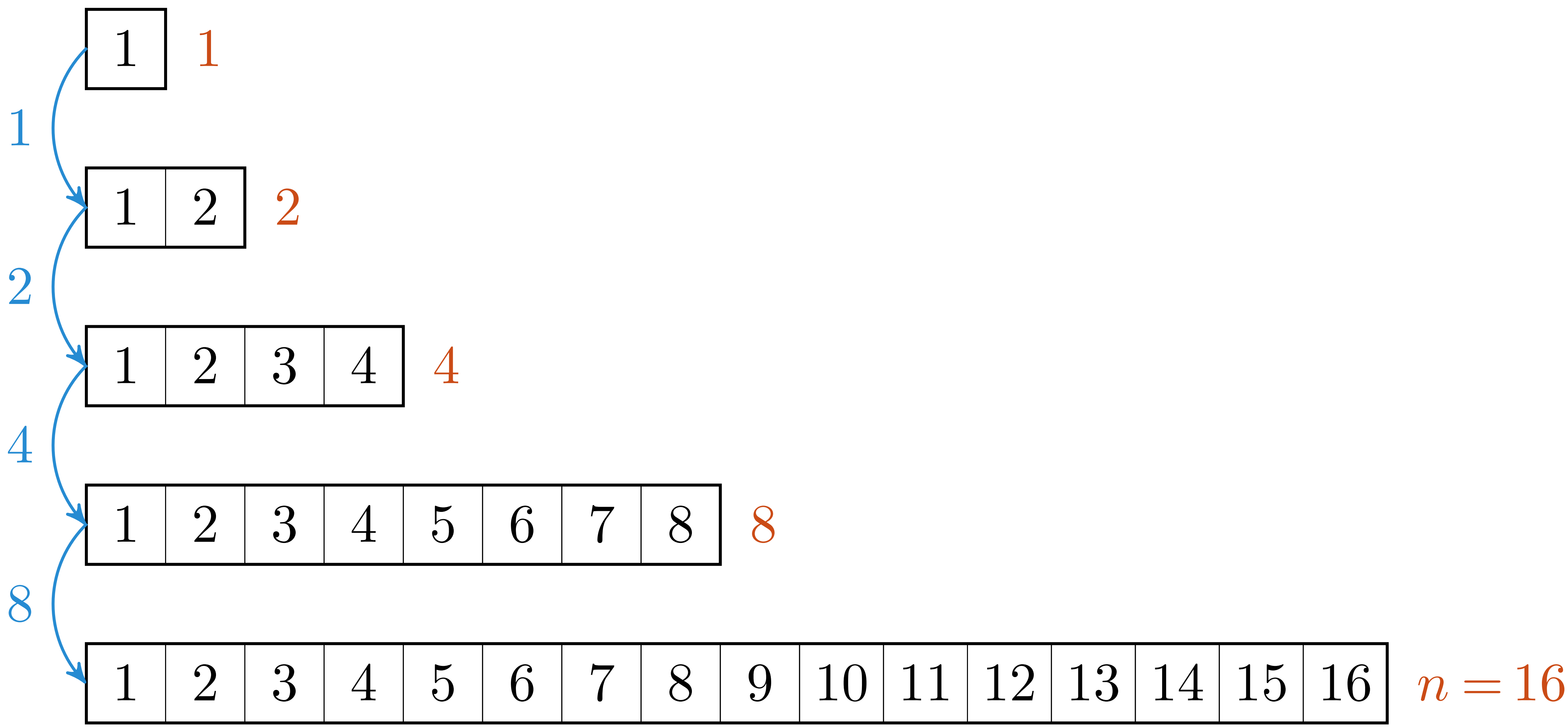

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11 12
n = 16


1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11 12 13
n = 16


1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11 12 13 14
$n = 16$

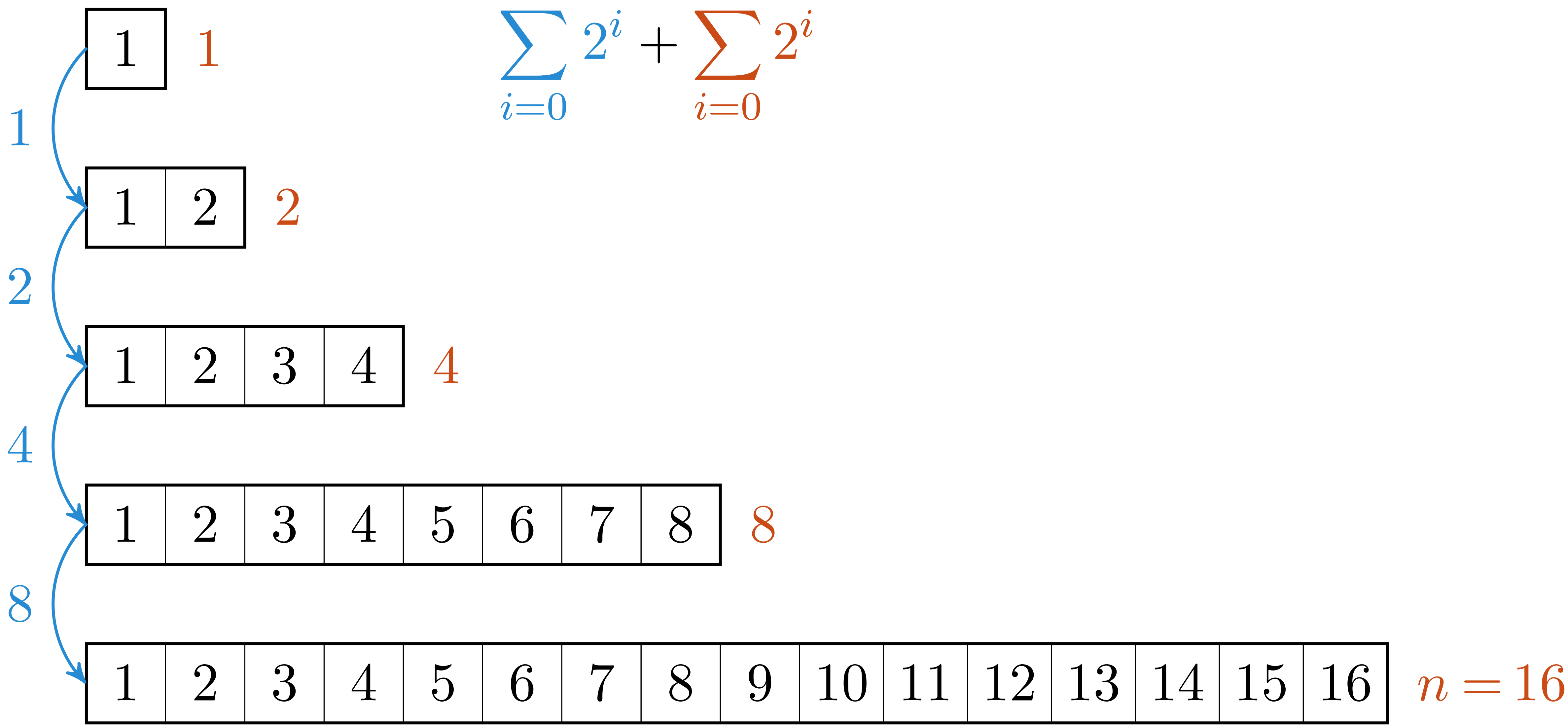

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
n = 16


1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
$n = 16$



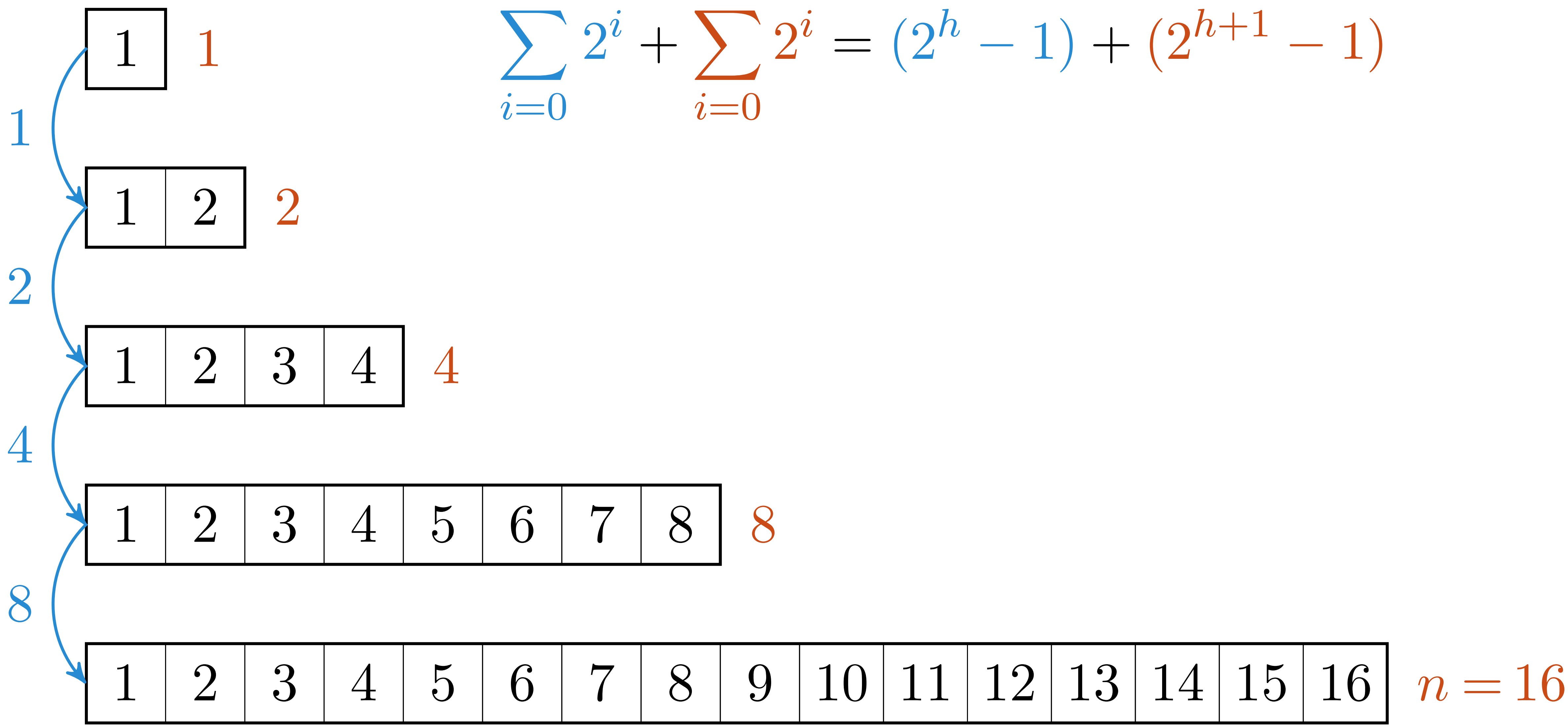
$$h = \log_2 n = 4$$

$$\sum_{i=0}^{h-1} 2^i + \sum_{i=0}^{h} 2^i$$

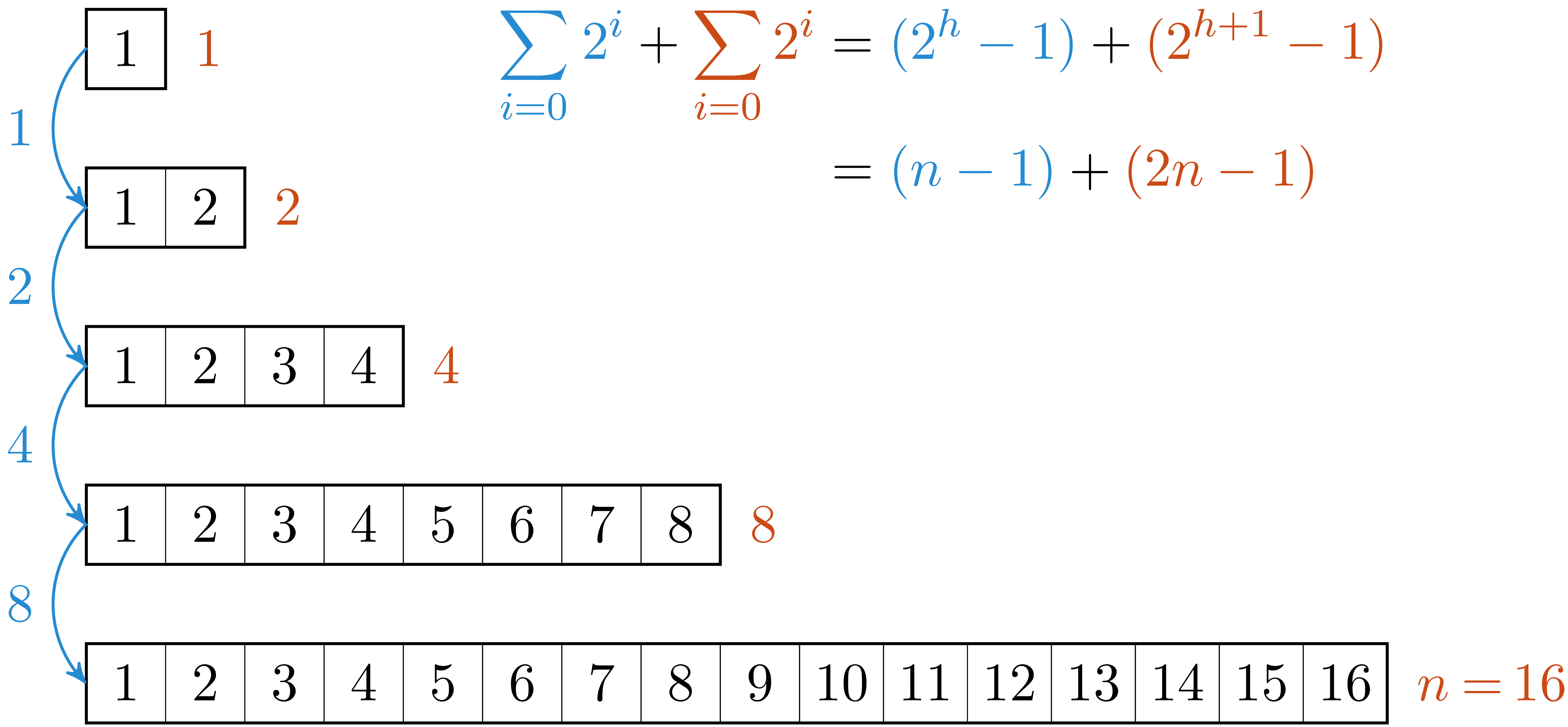


$$h = \log_2 n = 4$$

$$\sum_{i=0}^{h-1} 2^i + \sum_{i=0}^{h} 2^i = (2^h - 1) + (2^{h+1} - 1)$$

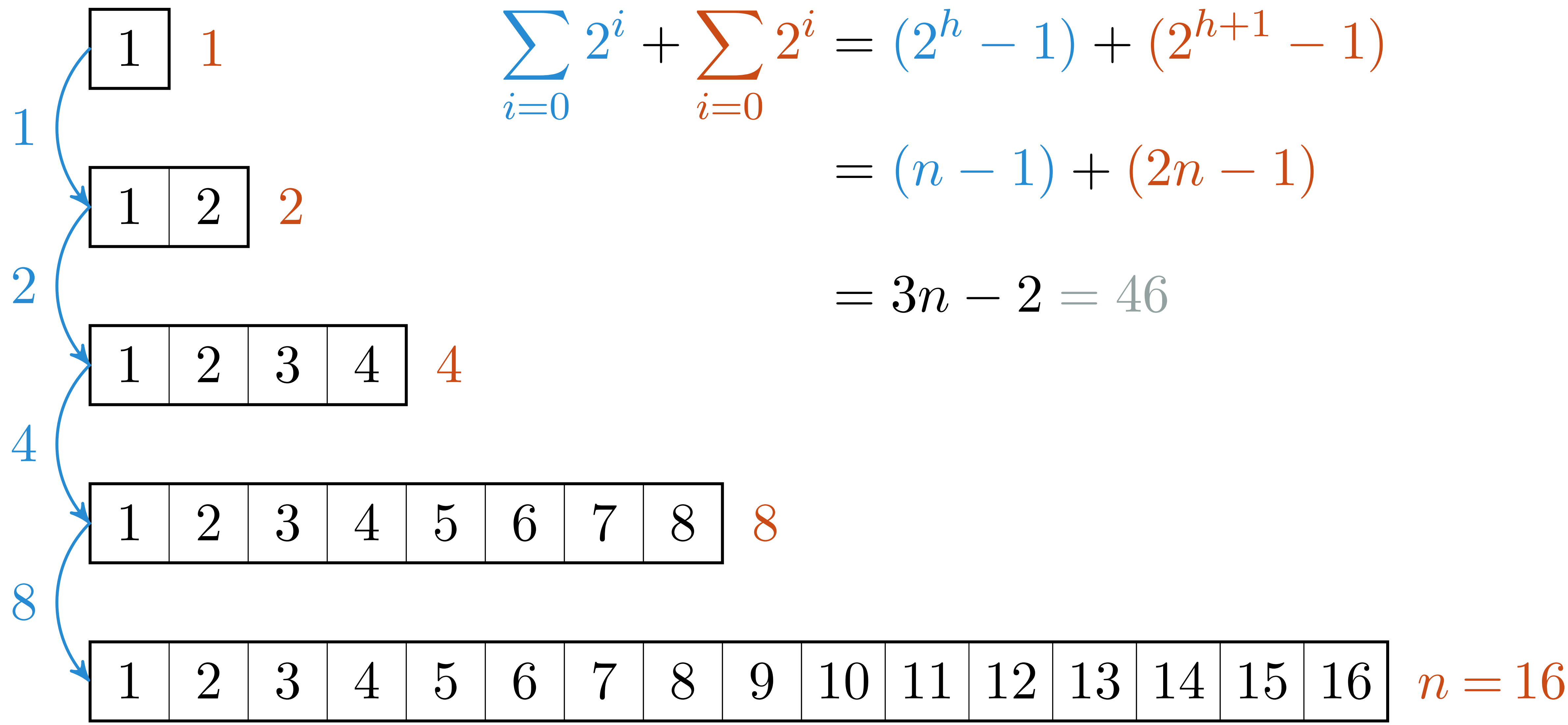


$$h = \log_2 n = 4$$



$$\sum_{i=0}^{h-1} 2^i + \sum_{i=0}^{h} 2^i = (2^h - 1) + (2^{h+1} - 1)$$
$$= (n - 1) + (2n - 1)$$

$$h = \log_2 n = 4$$



$$\sum_{i=0}^{h-1} 2^i + \sum_{i=0}^{h} 2^i = (2^h - 1) + (2^{h+1} - 1)$$

$$= (n - 1) + (2n - 1)$$

$$= 3n - 2 = 46$$

$$h = \log_2 n = 4$$

Ekstra arbeid per insetting: $\dfrac{3n-2}{n} = \Theta(1)$

Snitt-kjøretid 1 & 2

Avg-case og amortisering

Average-case: Forventet kjøretid.

Avg-case

Snitt over instanser

Amortisering

Snitt over operasjoner

1. **Stakker og køer**
2. **Lenkede lister**
3. **Hashtabeller**
4. **Dynamiske tabeller**

Neste gang: Splitt og hersk
(divide & conquer)!

Husk summene!