Forelesning 2

Datastrukturer

Pensum

- Kap. 10. Elementary data structures
- Kap. 11. Hash tables: s. 253–264
- Kap. 17. Amortized analysis: Innledning og s. 463–465

Komitéproblem

Dette er ekvivalent med det NP-komplette CLIQUE-problemet, der man prøver å finne minst k noder i en graf som alle har kanter til hverandre.

Dere lærer mer pom NP-kompletthet senere.

Finn minst k personer som kjenner hverandre

Finn minst k personer som hver kjenner minst k–1 av de andre

Dekomponering

Dekomponering

Løst

Uløst

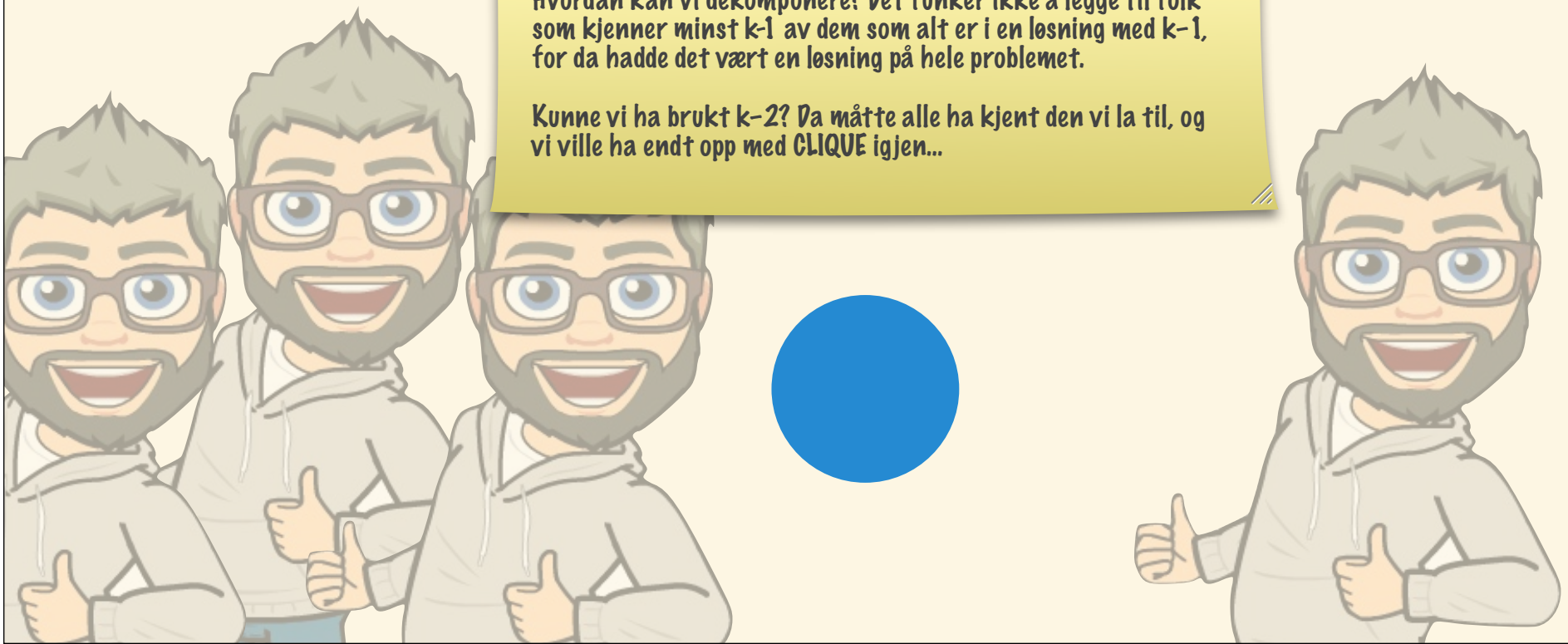
Dekomponering

Løst

Uløst

Hvordan kan vi dekomponere? Det funker ikke å legge til folk som kjenner minst k-1 av dem som alt er i en løsning med k-1, for da hadde det vært en løsning på hele problemet.

Kunne vi ha brukt k-2? Da måtte alle ha kjent den vi la til, og vi ville ha endt opp med CLIQUE igjen...



Dekomponering

Løst

Uløst

Dekomponering

Uløst

Løst

Kjenner du ikke minst k–1 av de andre?

Dekomponering

Uløst

Løst

Hvorfor blir det rett?

Forslag til invariant?

F.eks. en ganske direkte beskrivelse av algoritmen:
Alle vi har undersøkt så langt har blitt fjernet hvis og bare hvis de ikke kan være med i løsningen.

Til slutt vil dette altså gjelde alle – og vi kan da se om vi har minst k personer igjen.

Stakker

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

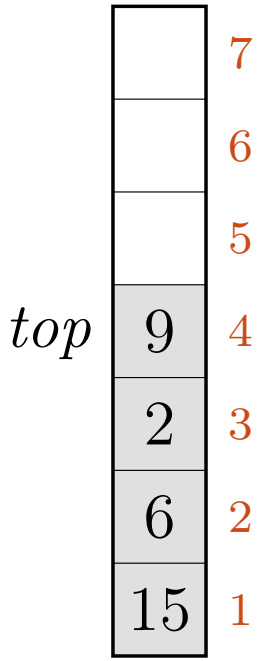
STACK-EMPTY(S)

```
STACK-EMPTY(S)
1  if S.top == 0
```

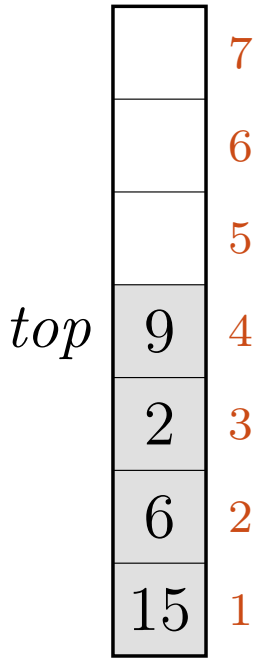
```
STACK-EMPTY(S)
1   if S.top == 0
2           return TRUE
```

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

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

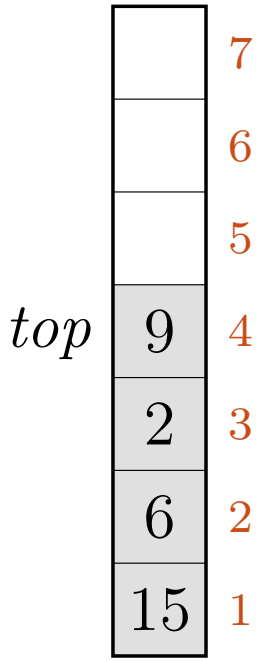


```
STACK-EMPTY(S)
1   if S.top == 0
2         return TRUE
3   else return FALSE
```



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

→ FALSE



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

$\textsc{Push}(S, x)$
1 $\quad S.top = S.top + 1$

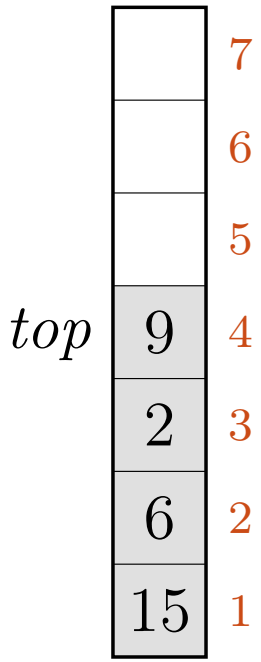
$\textsc{Push}(S, x)$
1 $S.top = S.top + 1$
2 $S[S.top] = x$

PUSH(S, x)

1 $S.top = S.top + 1$

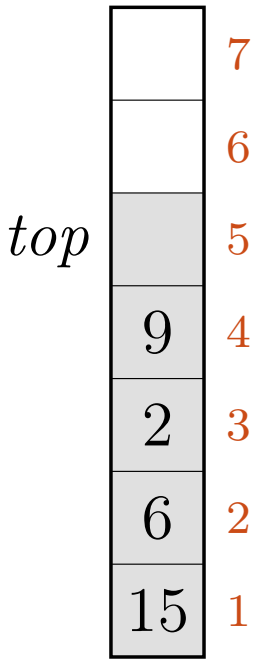
2 $S[S.top] = x$

$x = 17$



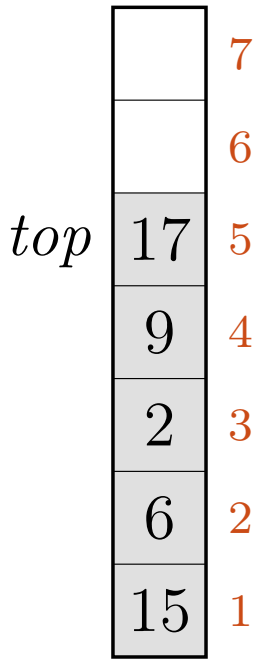
```
PUSH(S, x)
1   S.top = S.top + 1
2   S[S.top] = x
```

$x = 17$



PUSH(S, x)
1 $S.top = S.top + 1$
2 $S[S.top] = x$

$x = 17$

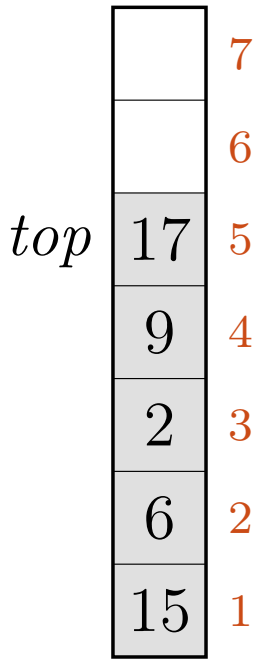


PUSH(S, x)

1 $S.top = S.top + 1$

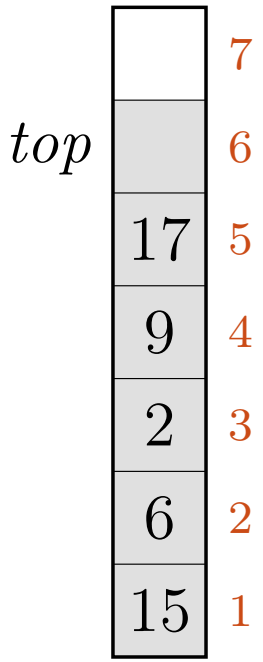
2 $S[S.top] = x$

$x = 3$



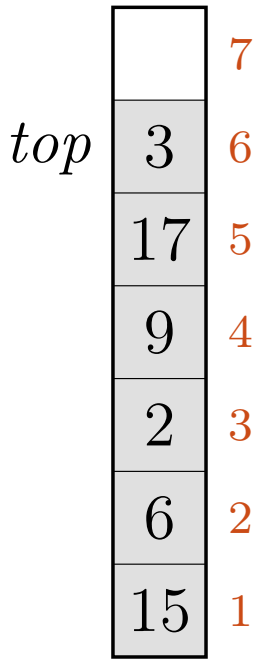
PUSH(S, x)
1 $S.top = S.top + 1$
2 $S[S.top] = x$

$x = 3$



PUSH(S, x)
1 $S.top = S.top + 1$
2 $S[S.top] = x$

$x = 3$



$\textsc{Pop}(S)$

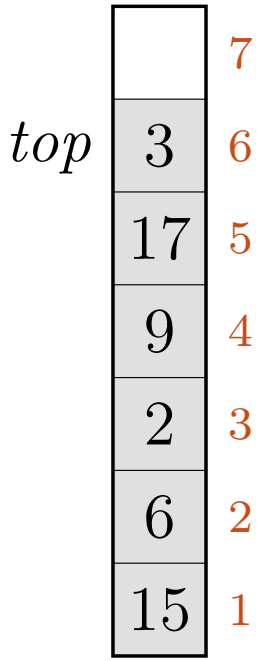
```
POP(S)
1  if STACK-EMPTY(S)
```

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

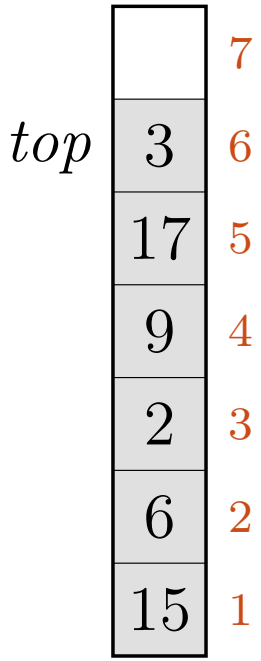
```
POP(S)
1   if STACK-EMPTY(S)
2         error "underflow"
3   else S.top = S.top − 1
```

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

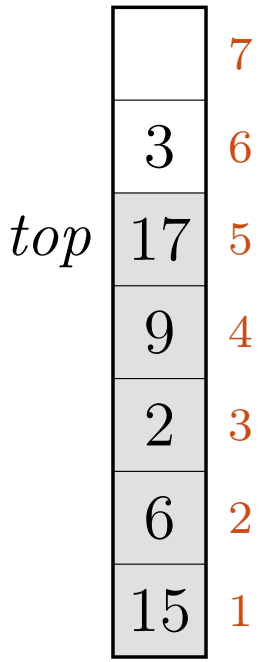
```
POP(S)
1   if STACK-EMPTY(S)
2         error "underflow"
3   else S.top = S.top − 1
4         return S[S.top + 1]
```



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

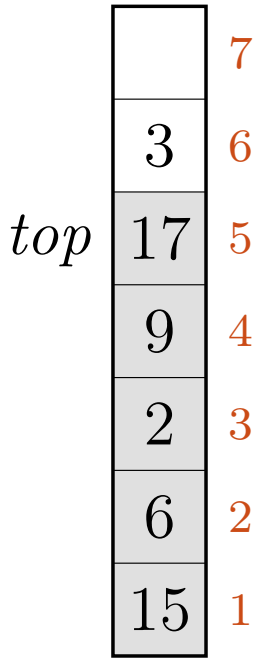


```
POP(S)
1  if STACK-EMPTY(S)
2        error "underflow"
3  else S.top = S.top − 1
4        return S[S.top + 1]
```



```
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

ENQUEUE(Q, x)

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

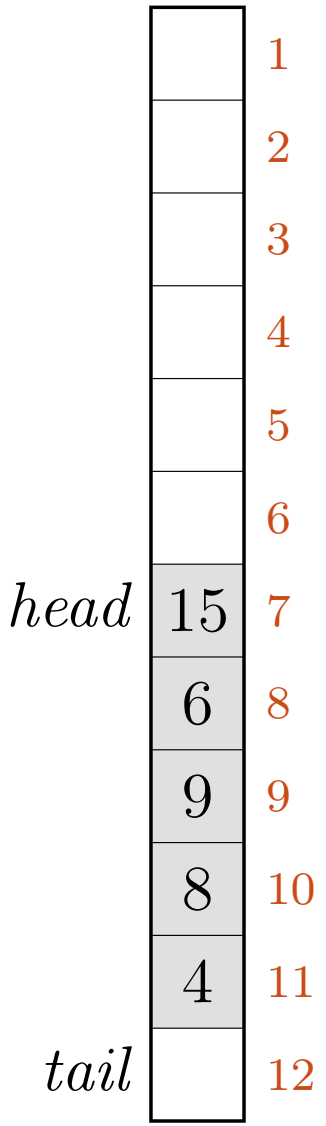
$\textsc{Enqueue}(Q, x)$
1 $\quad Q[Q.tail] = x$
2 $\quad$ **if** $Q.tail == Q.length$

```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3           Q.tail = 1
```

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

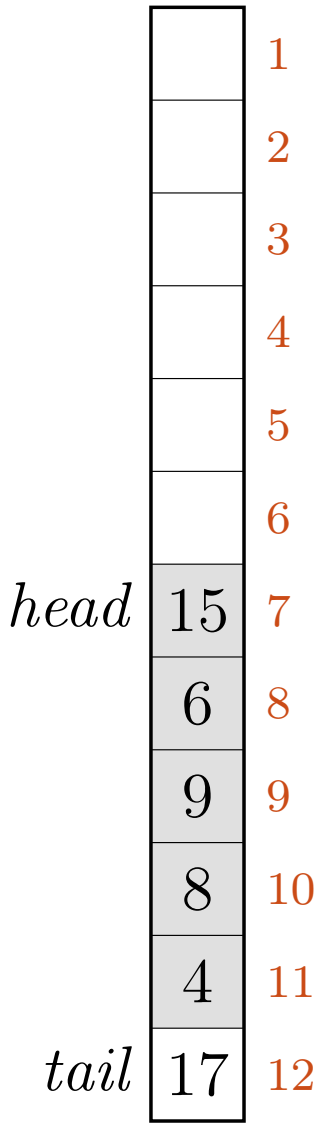
```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3         Q.tail = 1
4   else Q.tail = Q.tail + 1
```

$x = 17$



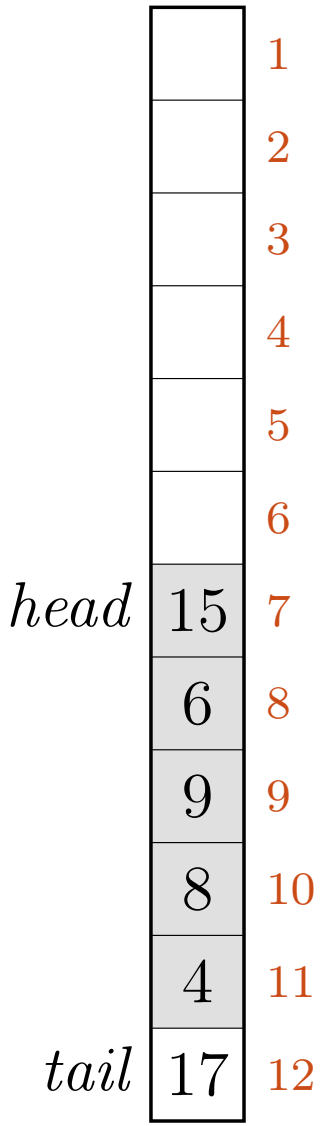
```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3         Q.tail = 1
4   else Q.tail = Q.tail + 1
```

$x = 17$



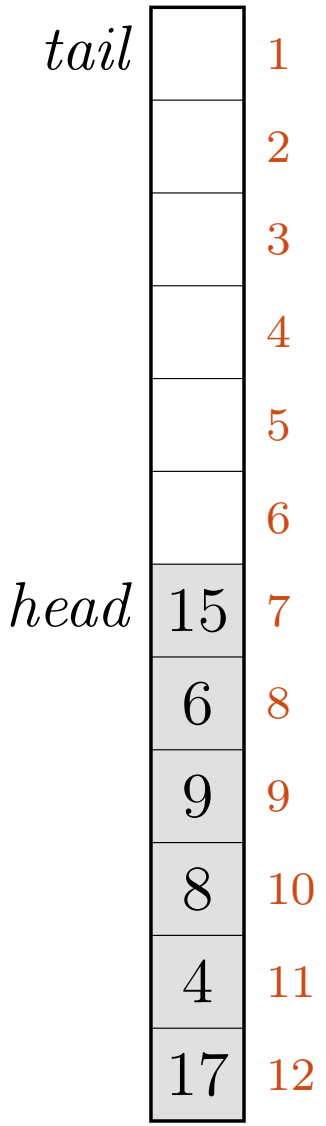
```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3         Q.tail = 1
4   else Q.tail = Q.tail + 1
```

$x = 17$



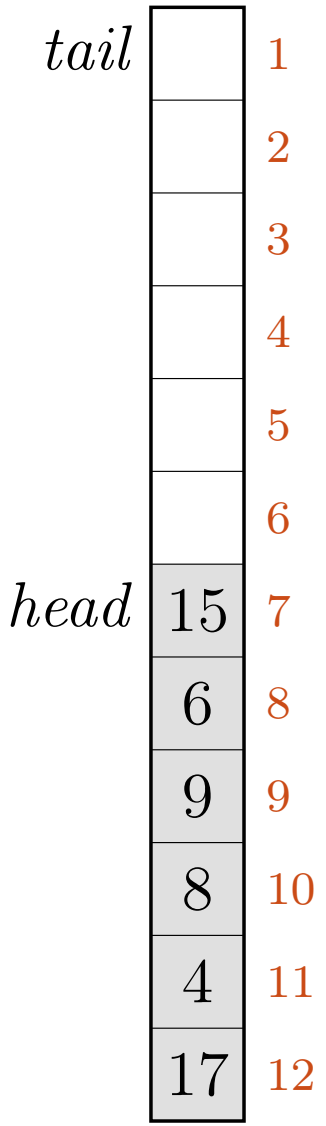
```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

$x = 17$



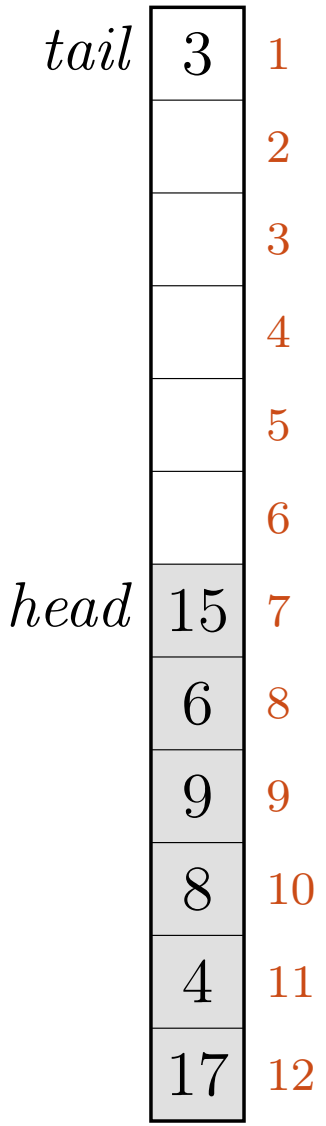
```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3         Q.tail = 1
4   else Q.tail = Q.tail + 1
```

$x = 3$



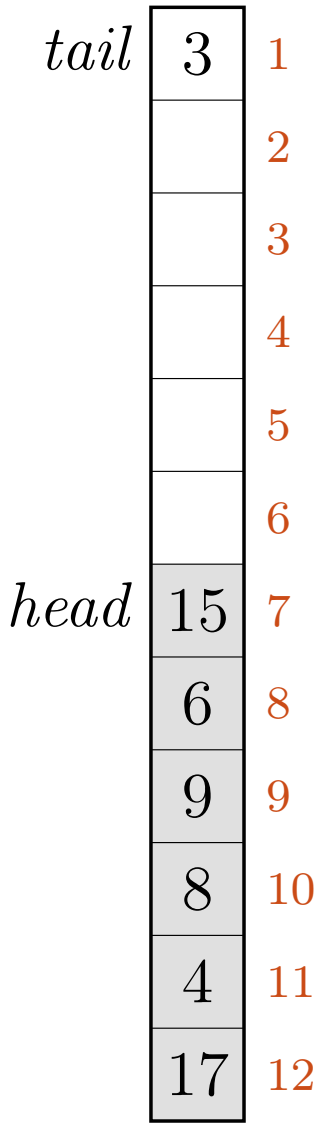
```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3         Q.tail = 1
4   else Q.tail = Q.tail + 1
```

$x = 3$



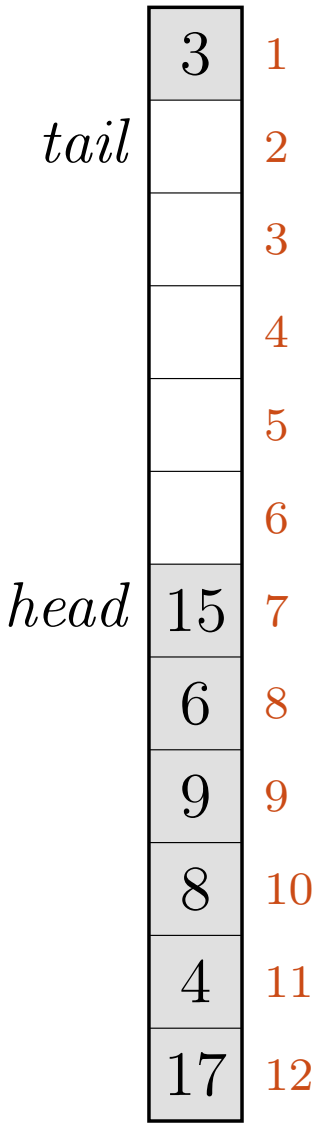
```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3         Q.tail = 1
4   else Q.tail = Q.tail + 1
```

$x = 3$



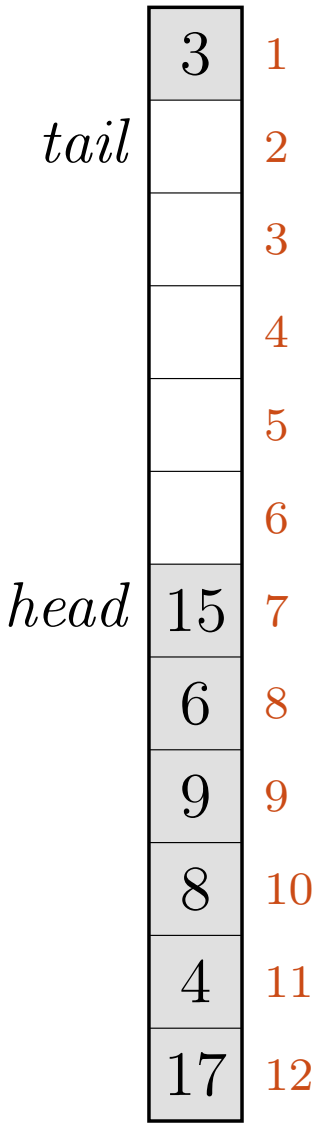
```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

$x = 3$



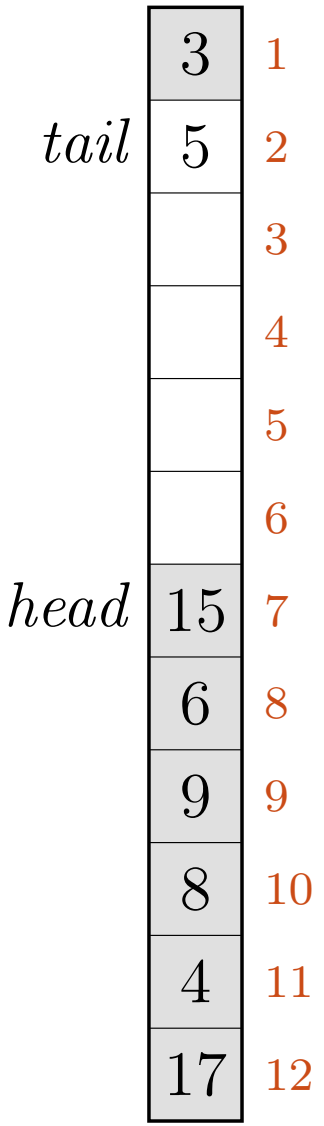
```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

$x = 5$



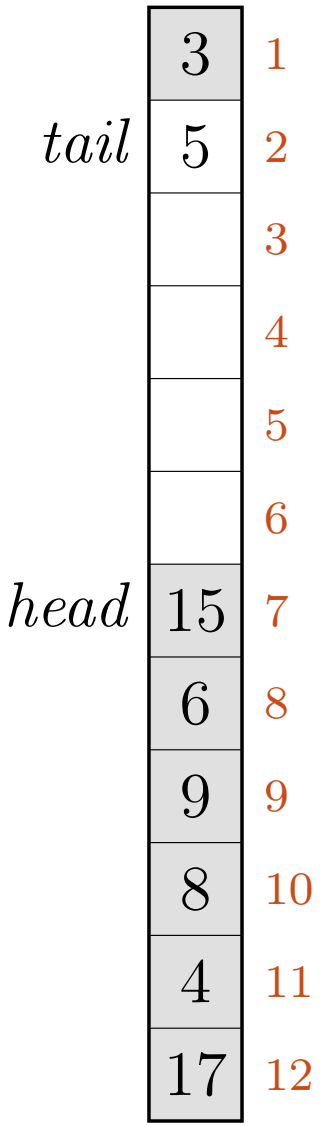
```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3         Q.tail = 1
4   else Q.tail = Q.tail + 1
```

$x = 5$



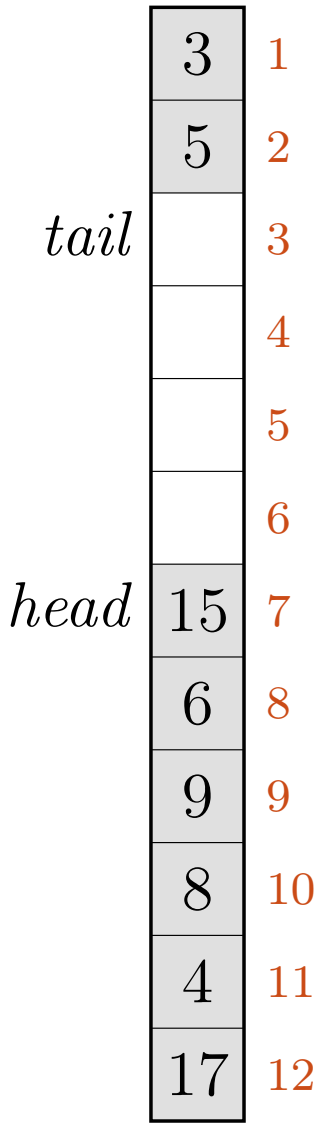
```
ENQUEUE(Q, x)
1   Q[Q.tail] = x
2   if Q.tail == Q.length
3         Q.tail = 1
4   else Q.tail = Q.tail + 1
```

$x = 5$



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

$x = 5$



DEQUEUE(Q)

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

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

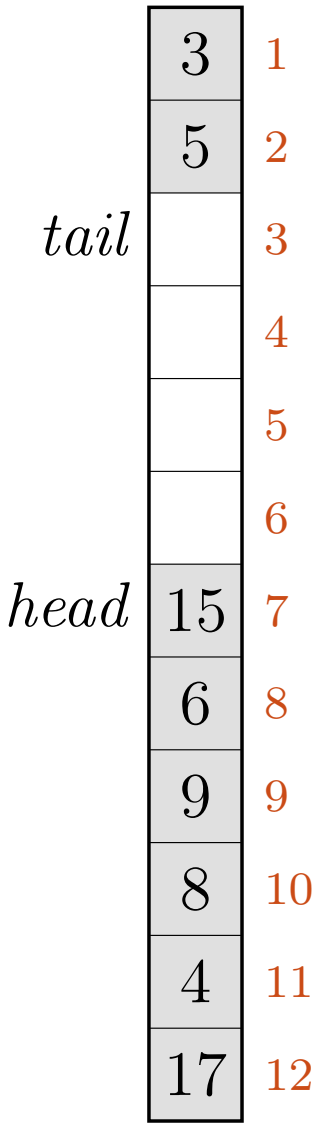
```
DEQUEUE(Q)
1   x = Q[Q.head]
2   if Q.head == Q.length
3           Q.head = 1
```

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

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

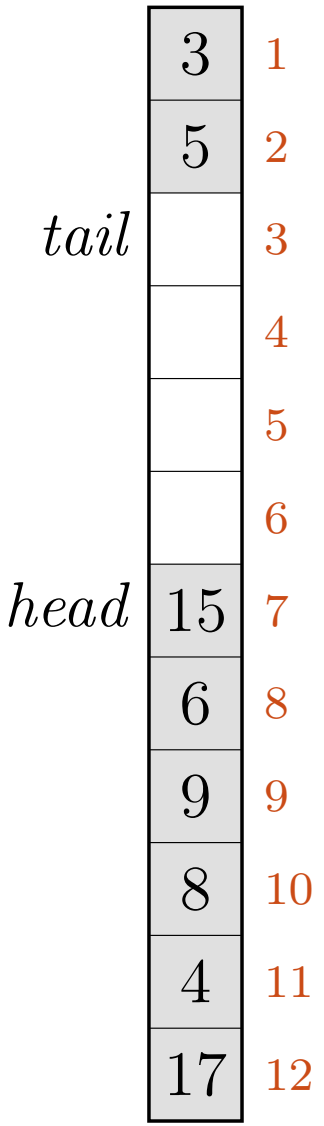
```
DEQUEUE(Q)
1   x = Q[Q.head]
2   if Q.head == Q.length
3         Q.head = 1
4   else Q.head = Q.head + 1
5   return x
```

$x = -$



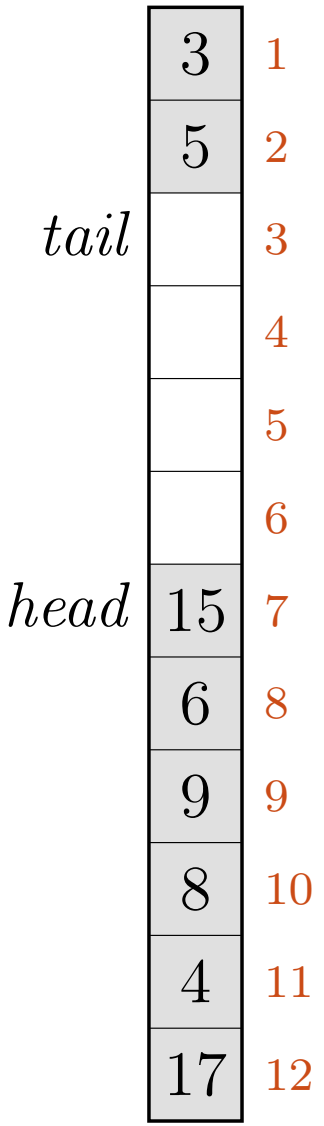
```
DEQUEUE(Q)
1   x = Q[Q.head]
2   if Q.head == Q.length
3         Q.head = 1
4   else Q.head = Q.head + 1
5   return x
```

$x = 15$



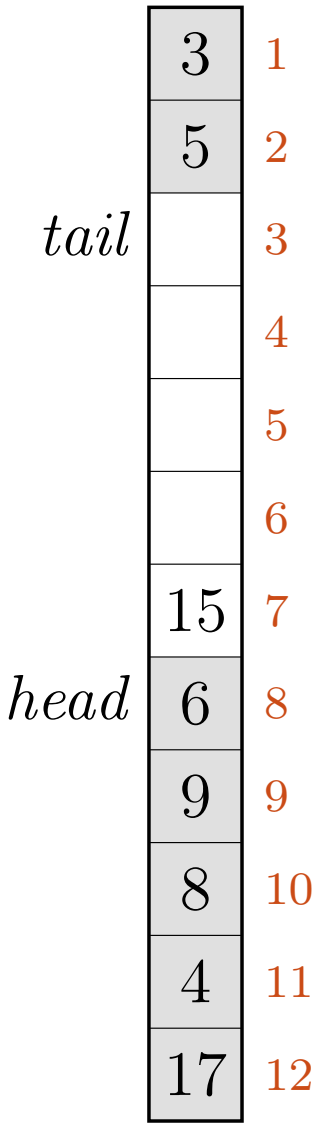
```
DEQUEUE(Q)
1   x = Q[Q.head]
2   if Q.head == Q.length
3         Q.head = 1
4   else Q.head = Q.head + 1
5   return x
```

$x = 15$



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

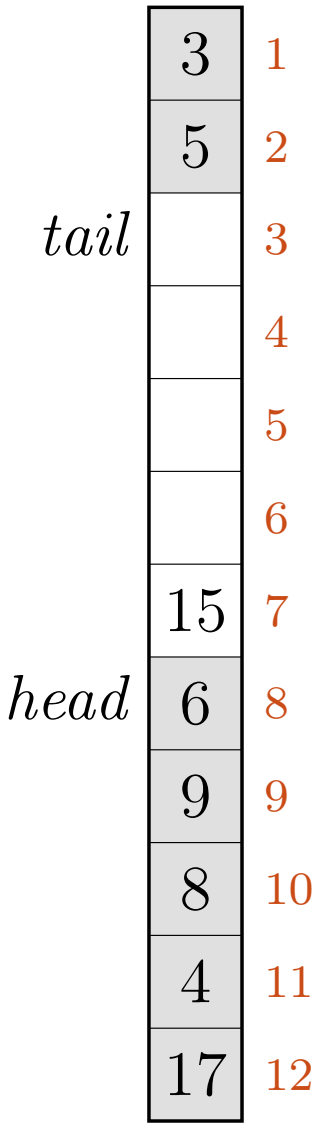
$x = 15$



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

→ 15

$x = 15$



Snitt-kjøretid 1 & 2

Avg-case og amortisering

Avg-case

Snitt over instanser

$$I_n = \text{Instanser med størrelse } n$$

$I_n = \text{Instanser med størrelse } n$

$T_x = \text{Tid for input } x$

$$I_n = \text{Instanser med størrelse } n$$

$$T_x = \text{Tid for input } x$$

$$T_{\text{avg}}(n) = \frac{\sum_{x \in I_n} T_x}{|I_n|}$$

$$I_n = \text{Instanser med størrelse } n$$

$$T_x = \text{Tid for input } x$$

$$T_{\text{avg}}(n) = \frac{\sum_{x \in I_n} T_x}{|I_n|}$$

Uniform sannsynlighet

$$I_n = \text{Instanser med størrelse } n$$

$I_n = \text{Instanser med størrelse } n$

$T_x = \text{Tid for input } x$

$$I_n = \text{Instanser med størrelse } n$$

$$T_x = \text{Tid for input } x$$

$$p(x; n) = \text{Sannsynlighet for } x$$

$$I_n = \text{Instanser med størrelse } n$$

$$T_x = \text{Tid for input } x$$

$$p(x; n) = \text{Sannsynlighet for } x$$

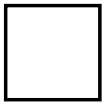
$$T_{\text{avg}}(n) = \sum_{i \in I_n} \big(p(x; n) \cdot T_x \big)$$

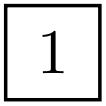
Amortisering

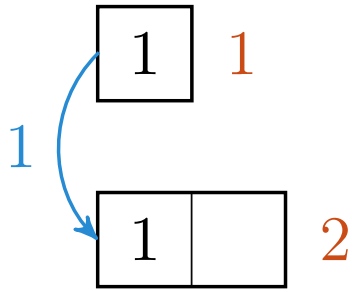
Snitt over operasjoner!

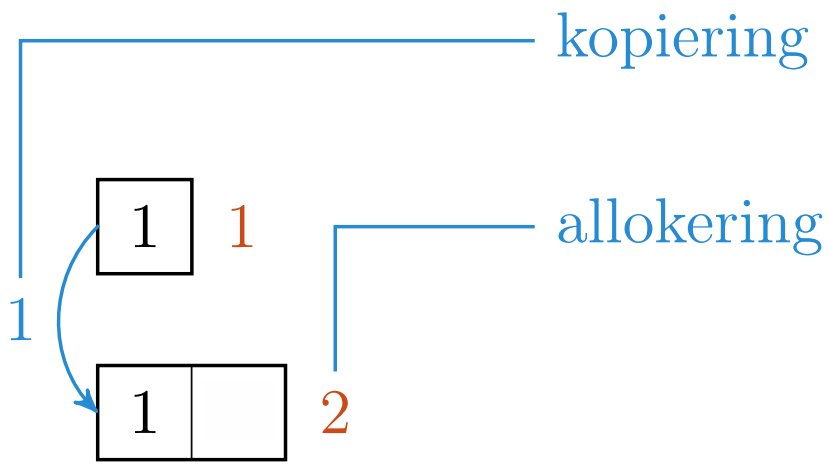
Eksempel: Dynamiske tabeller

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

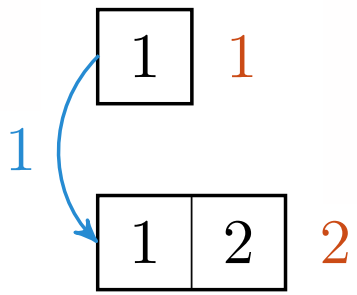


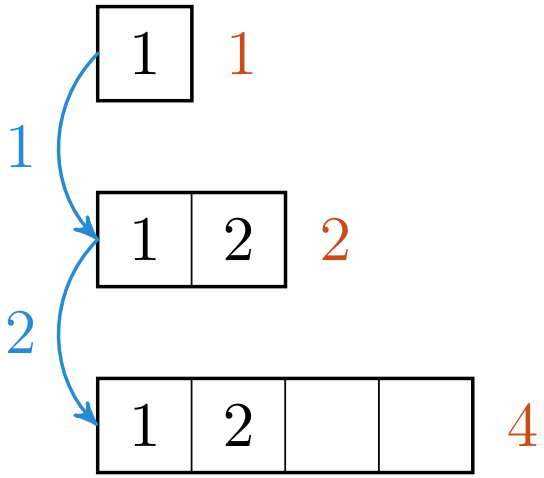

1

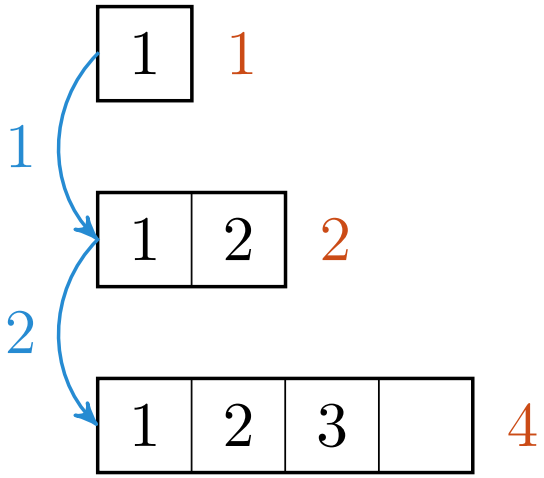

1
1
1
1
2

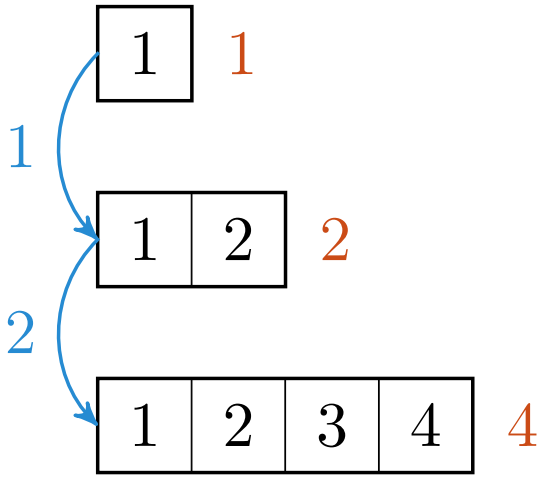


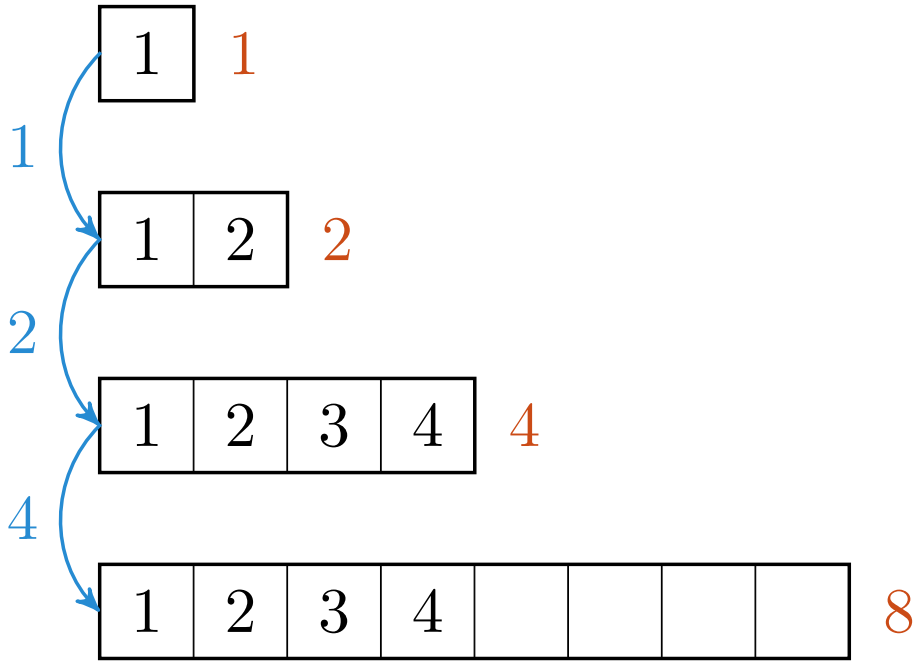
Vi bør jo ta med deallokering av den forrige tabellen òg – men det endrer ikke det asymptotiske resultatet.

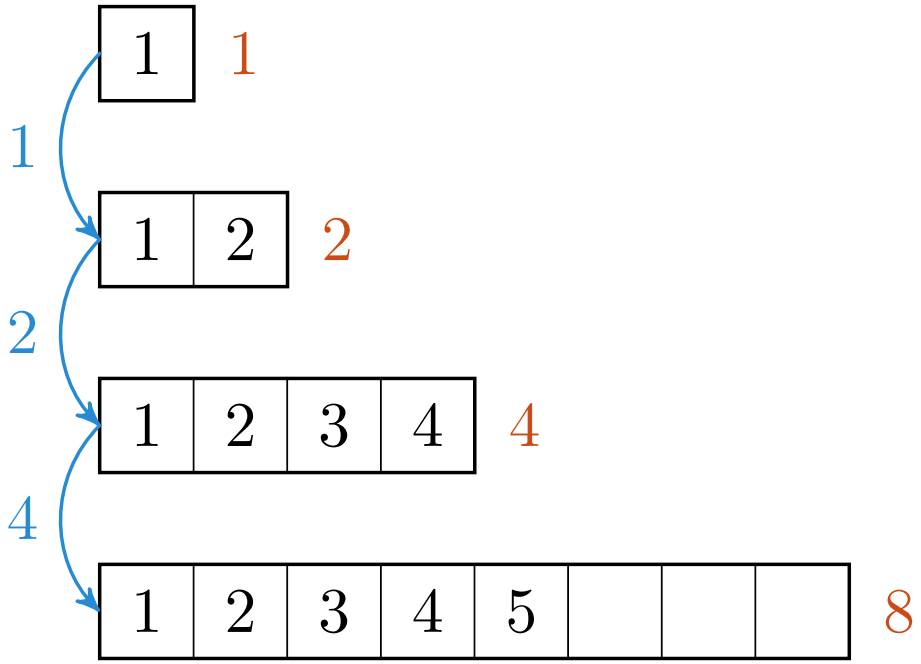

1
1
1
1
2
2

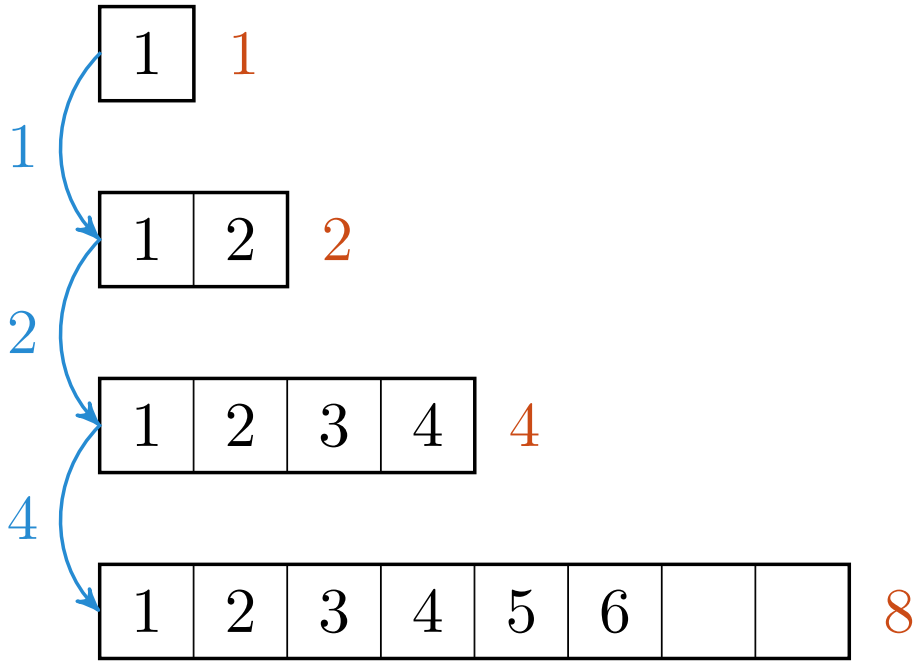

1
1
1
1
2
2
2
1
2
4

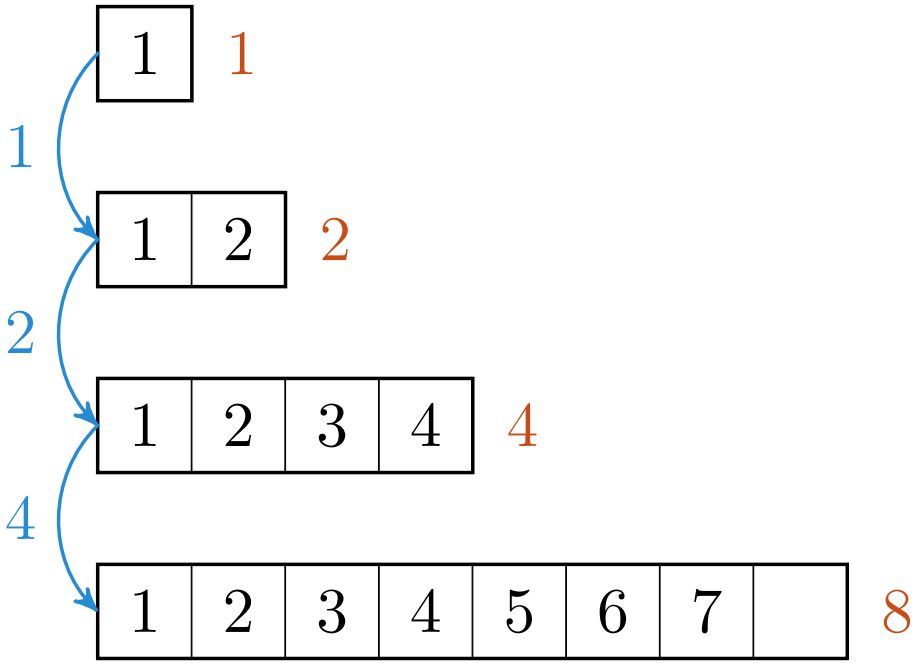

1
1
1
1
2
2
2
1
2
3
4

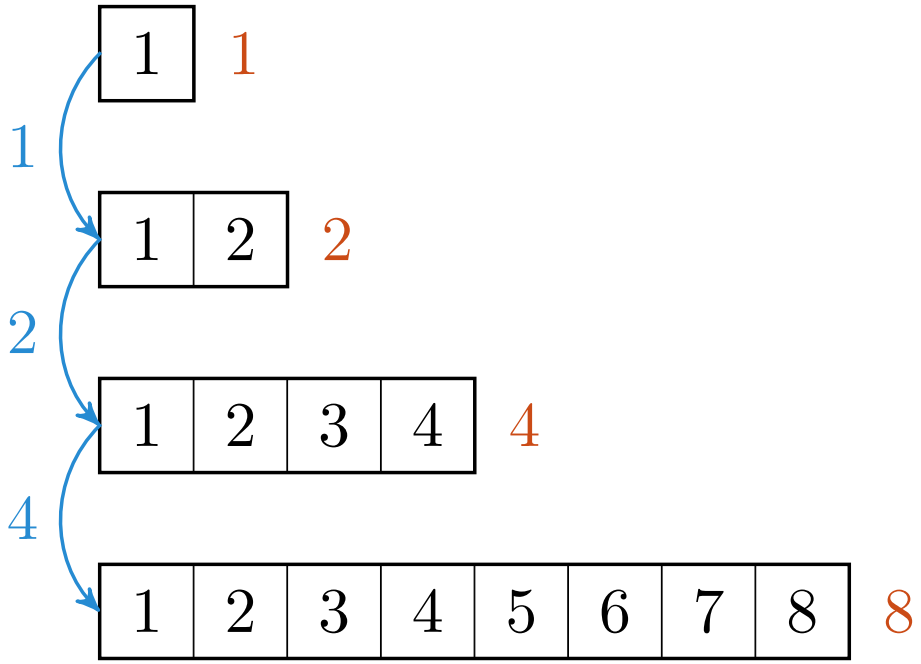

1
1
1
1
2
2
2
1
2
3
4
4

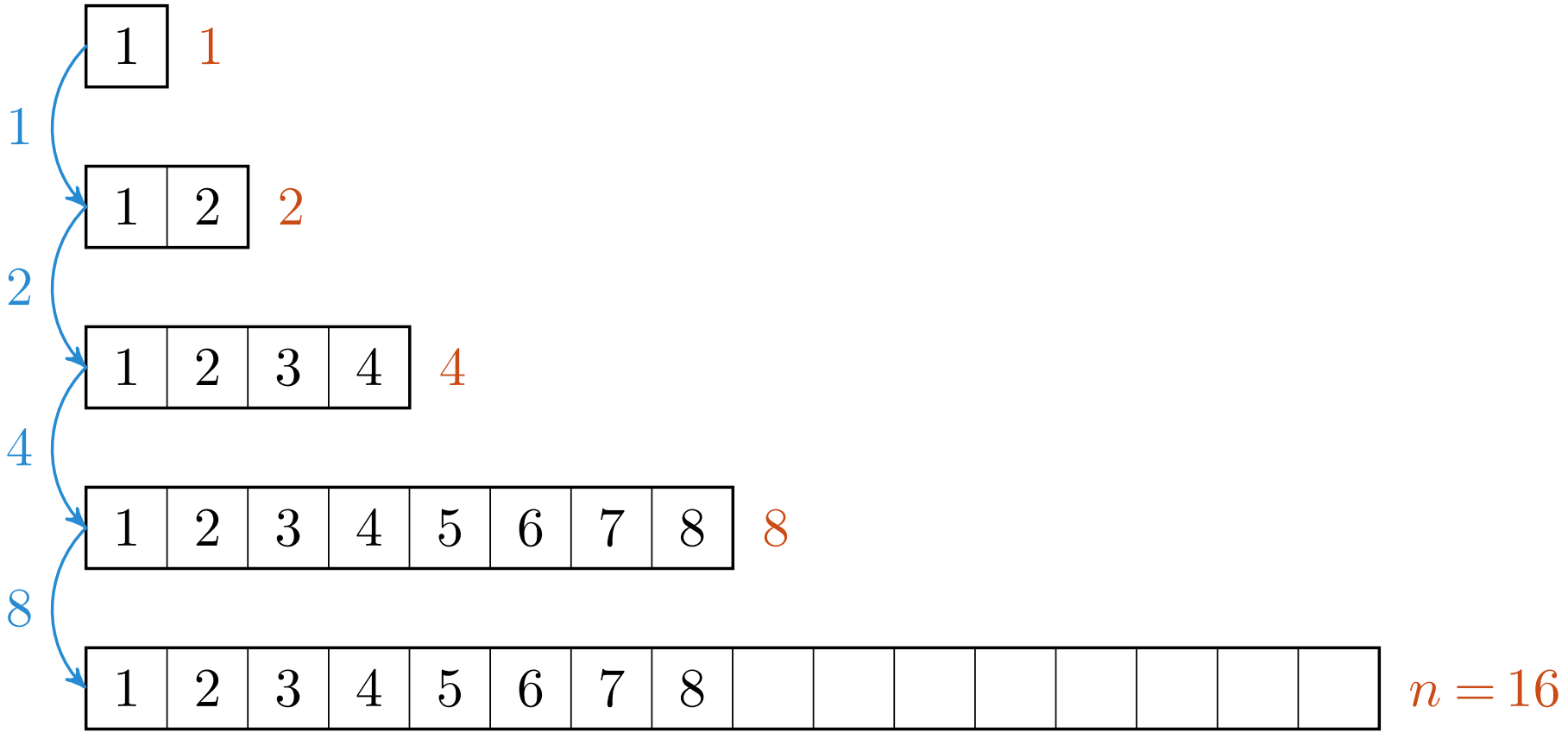

1
1
1
1
2
2
1
2
3
4
4
4
1
2
3
4
8

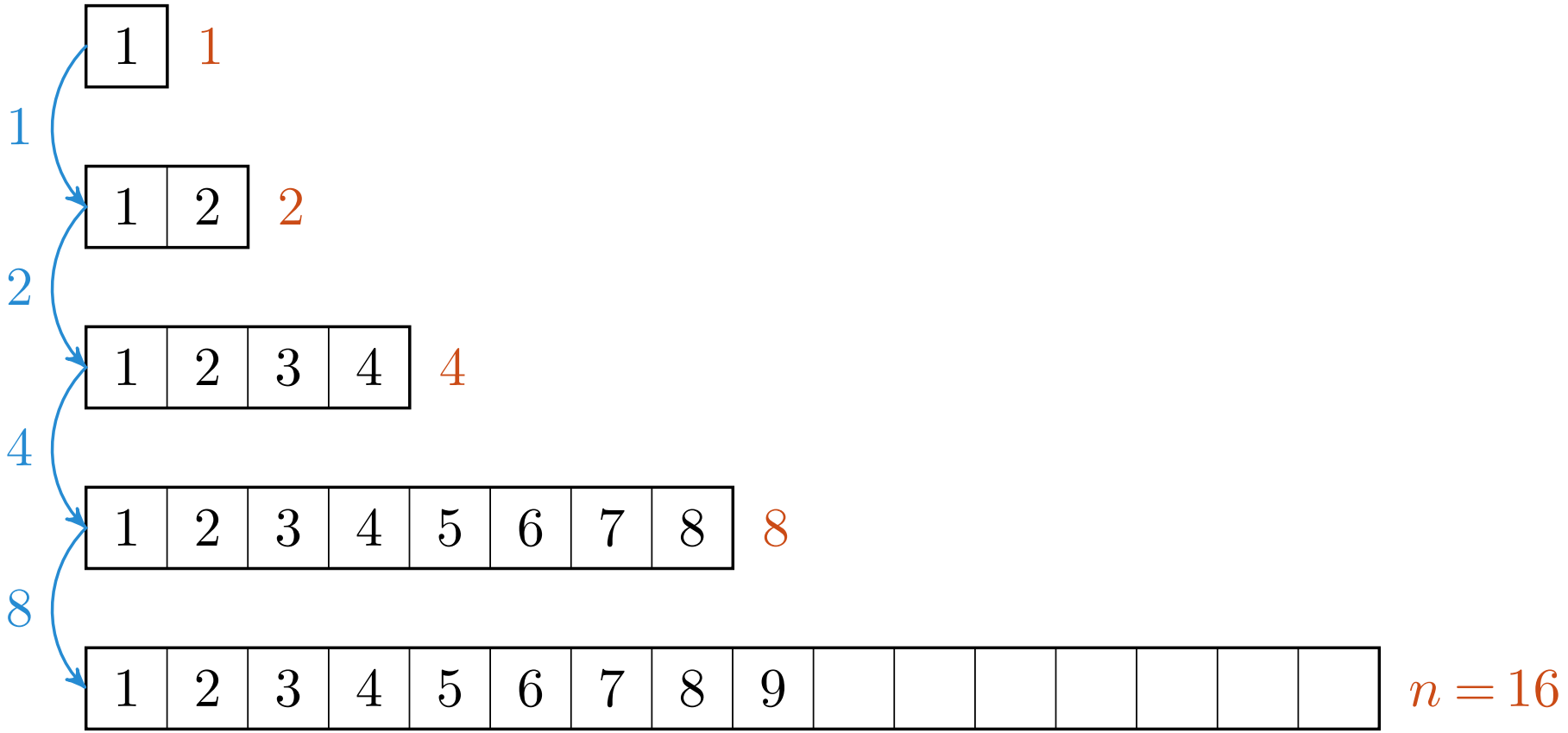

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5
8

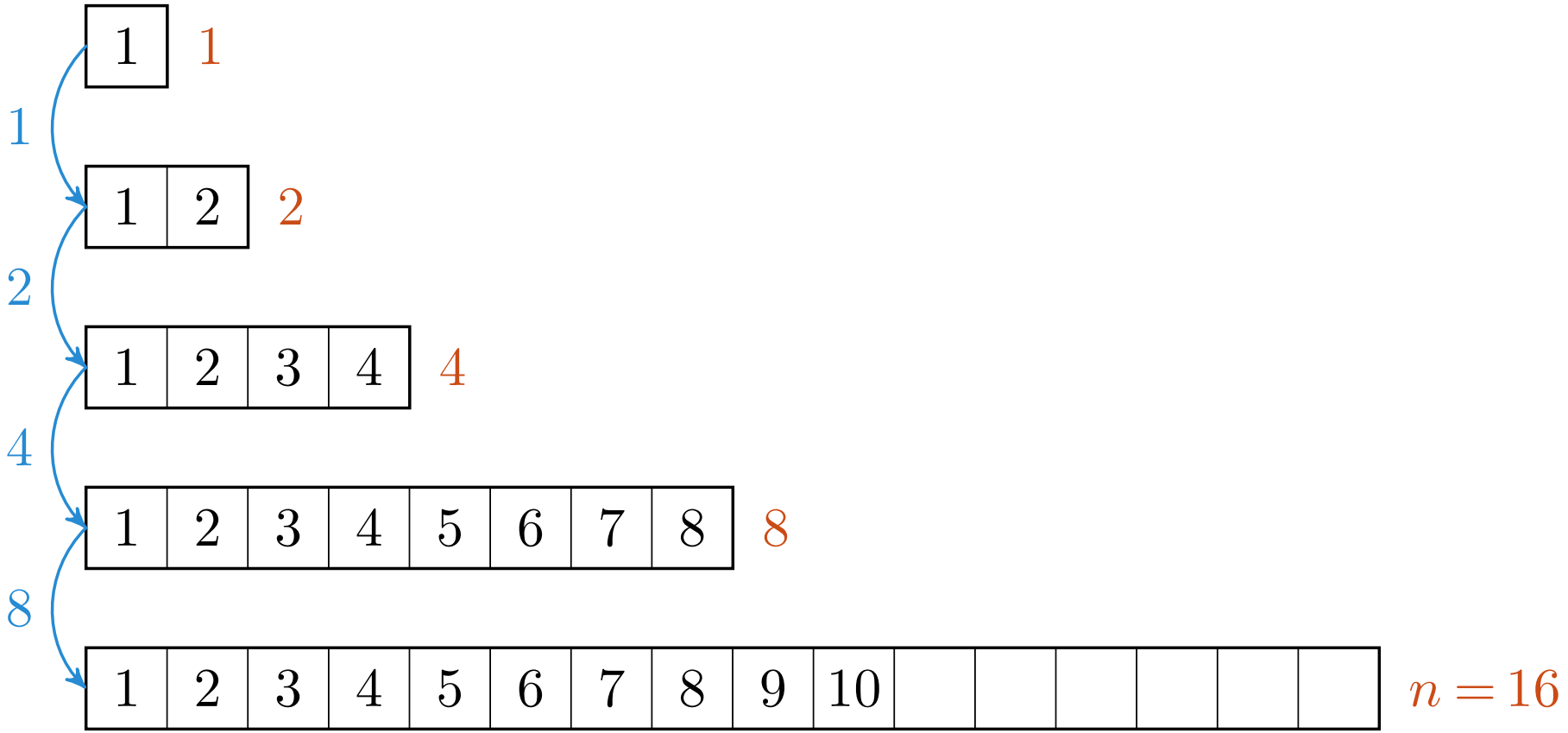

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6
8

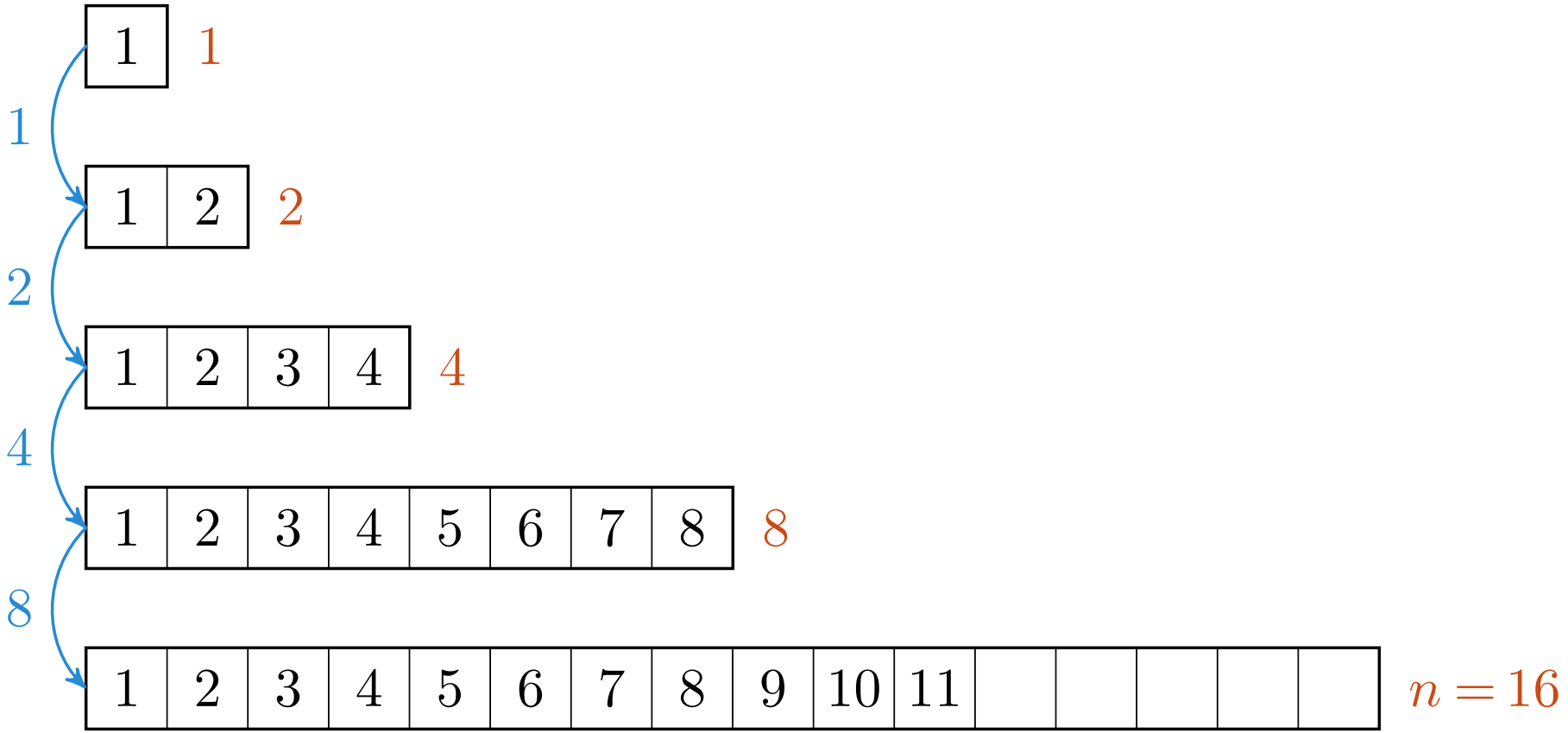

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7
8

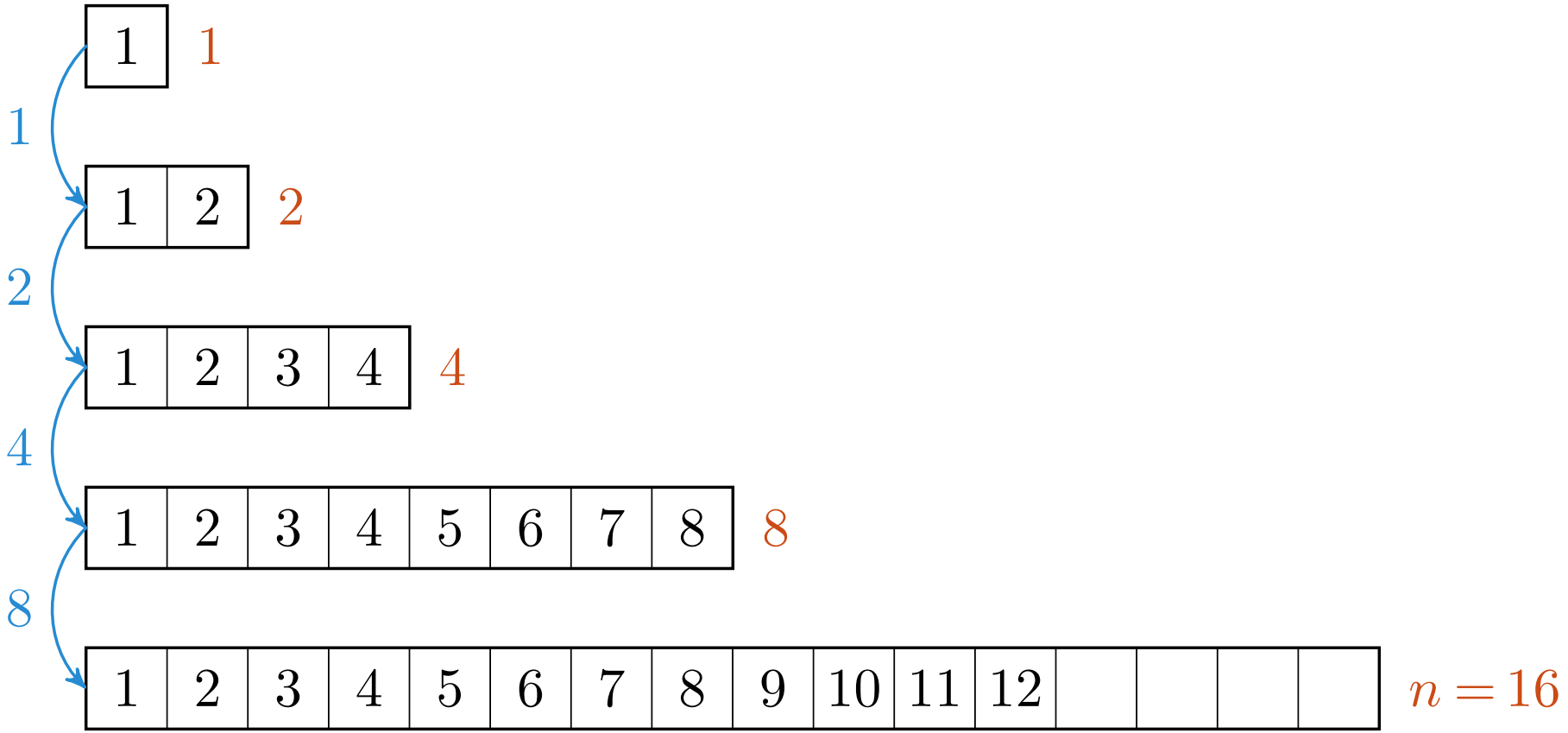

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8

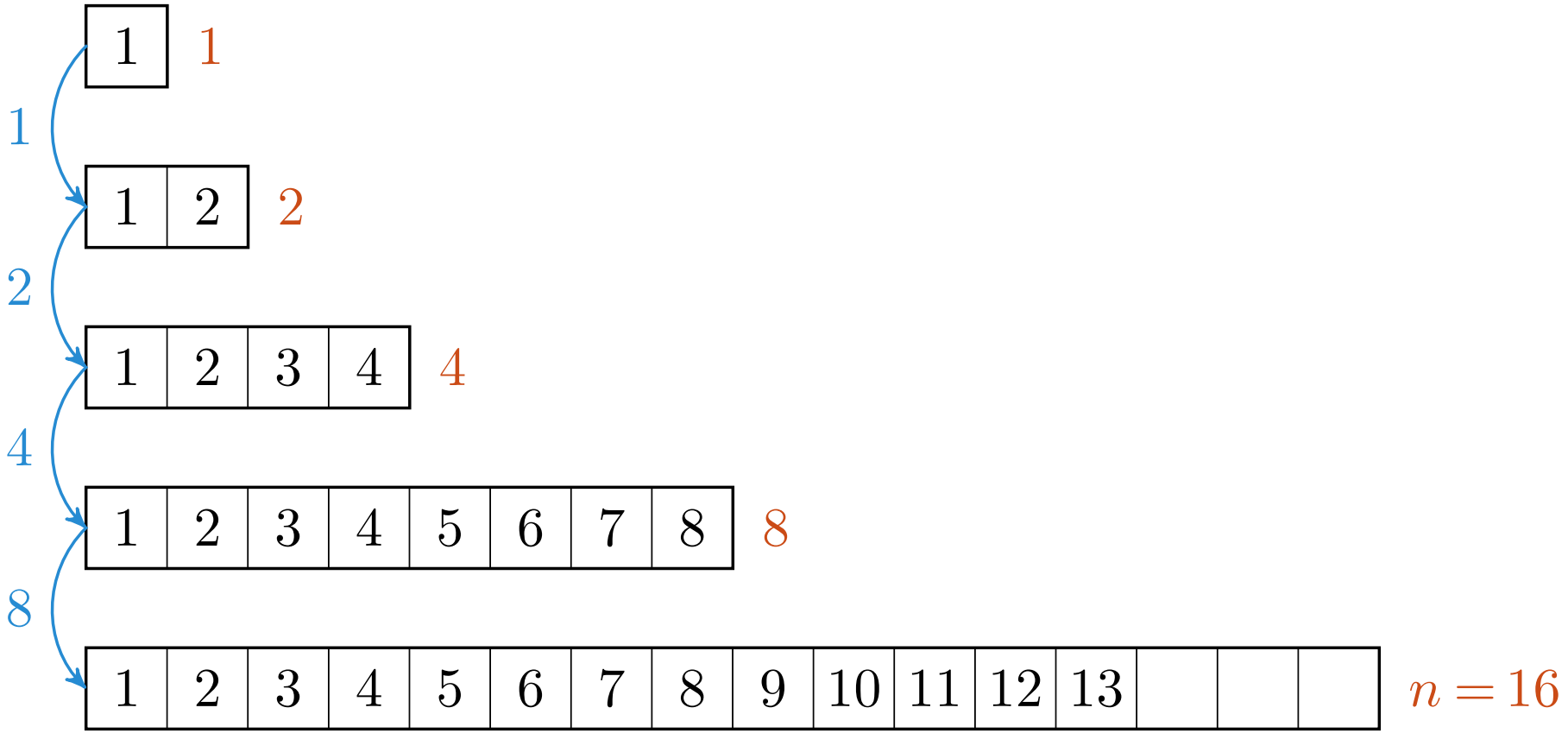

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8
n = 16

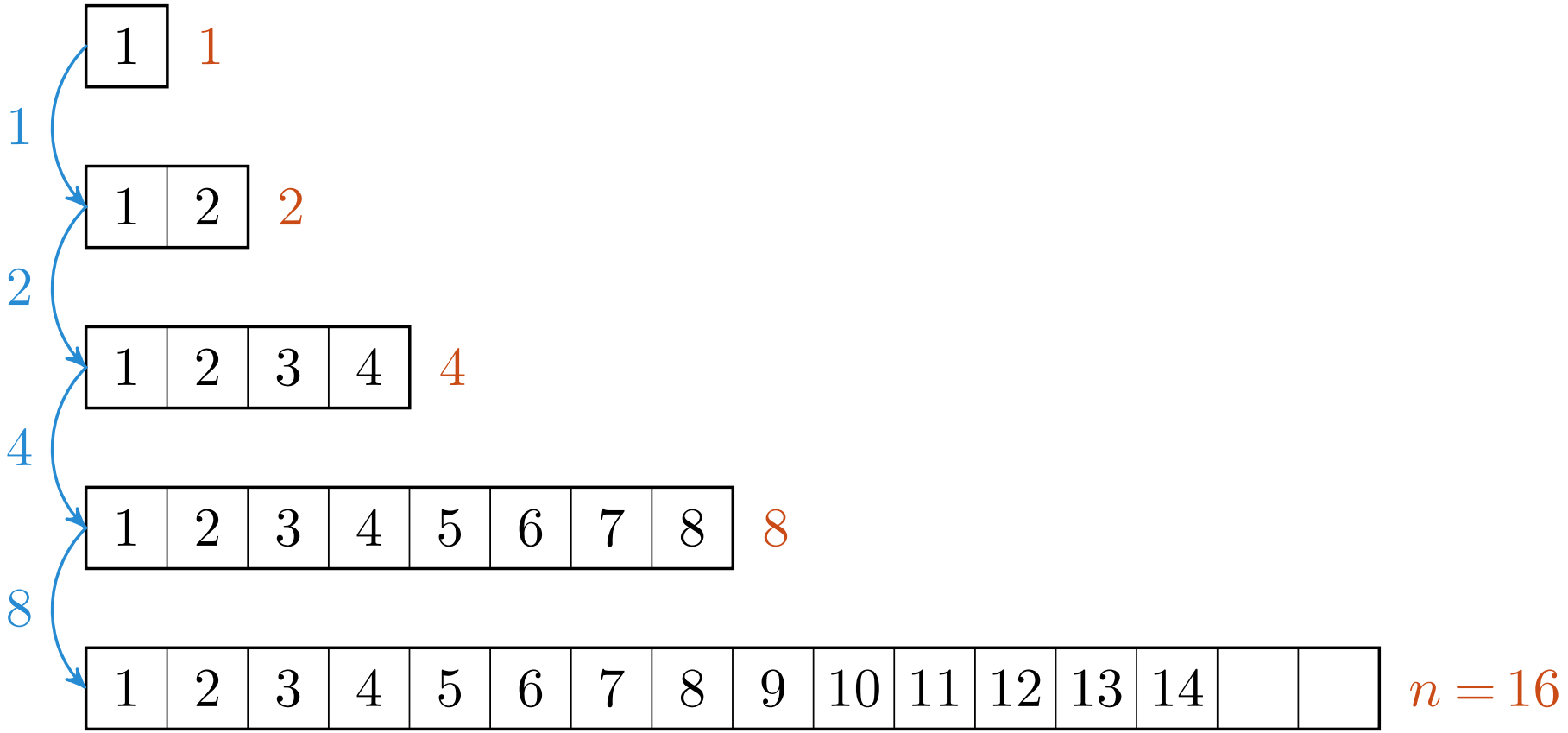

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9
n = 16

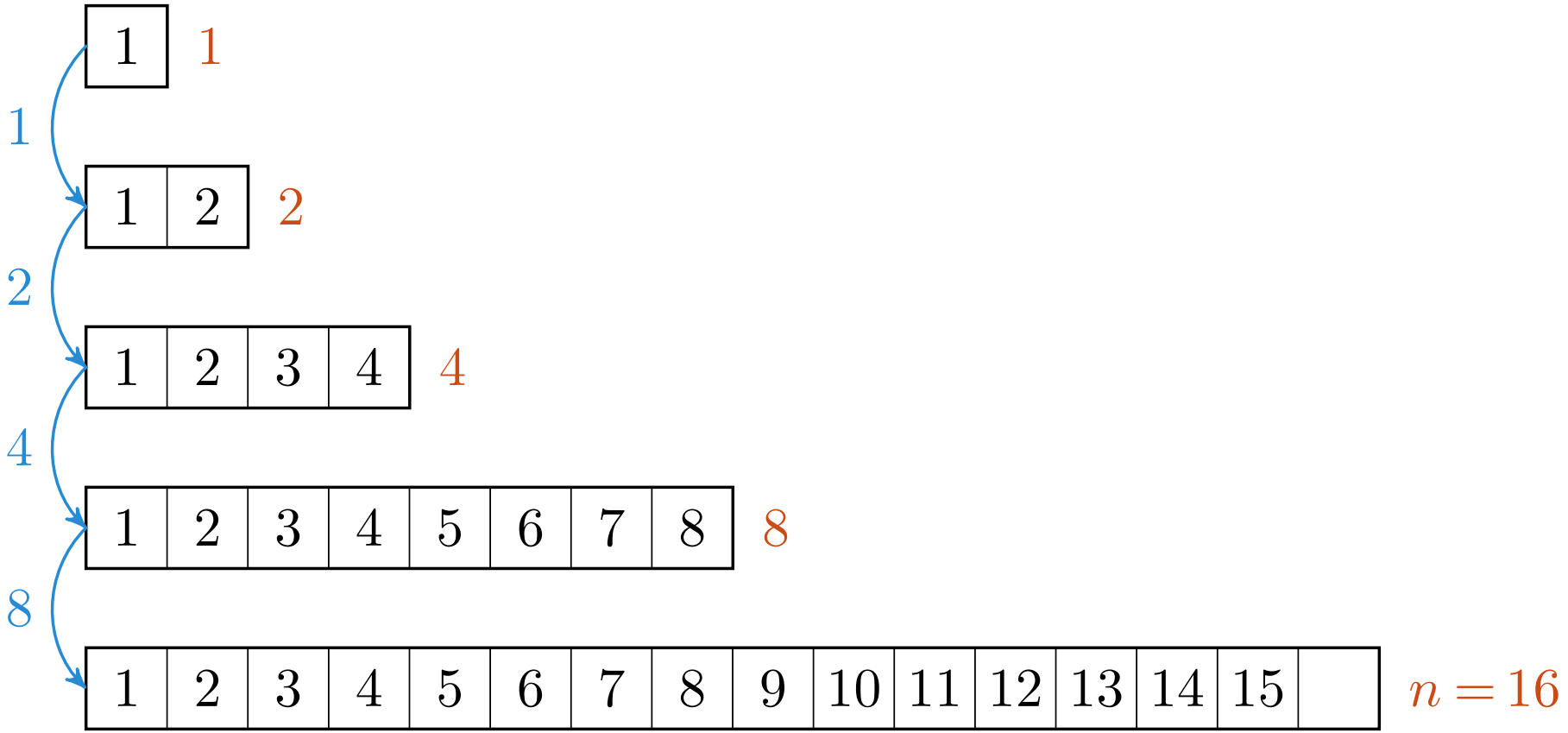

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10
n = 16

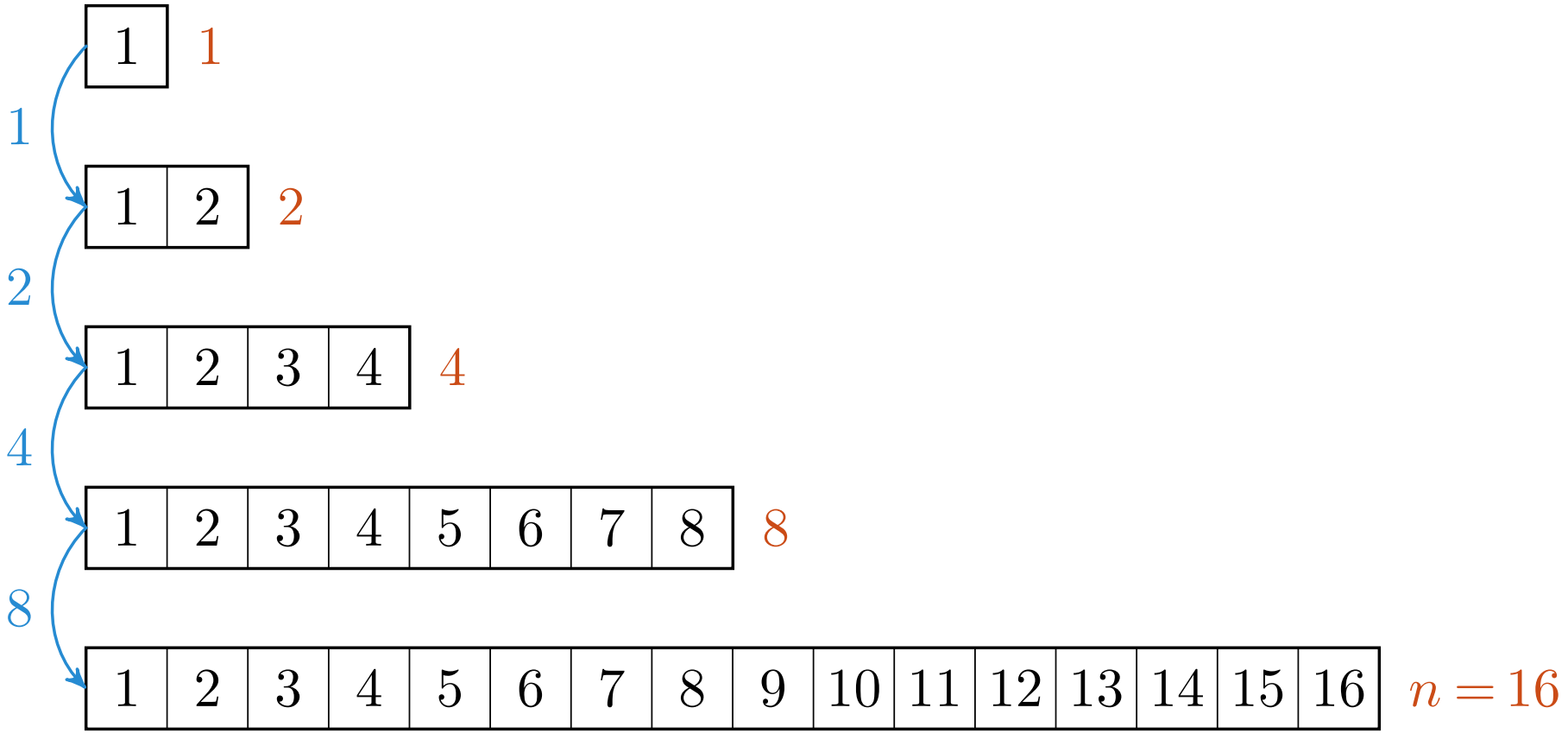

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11
n = 16

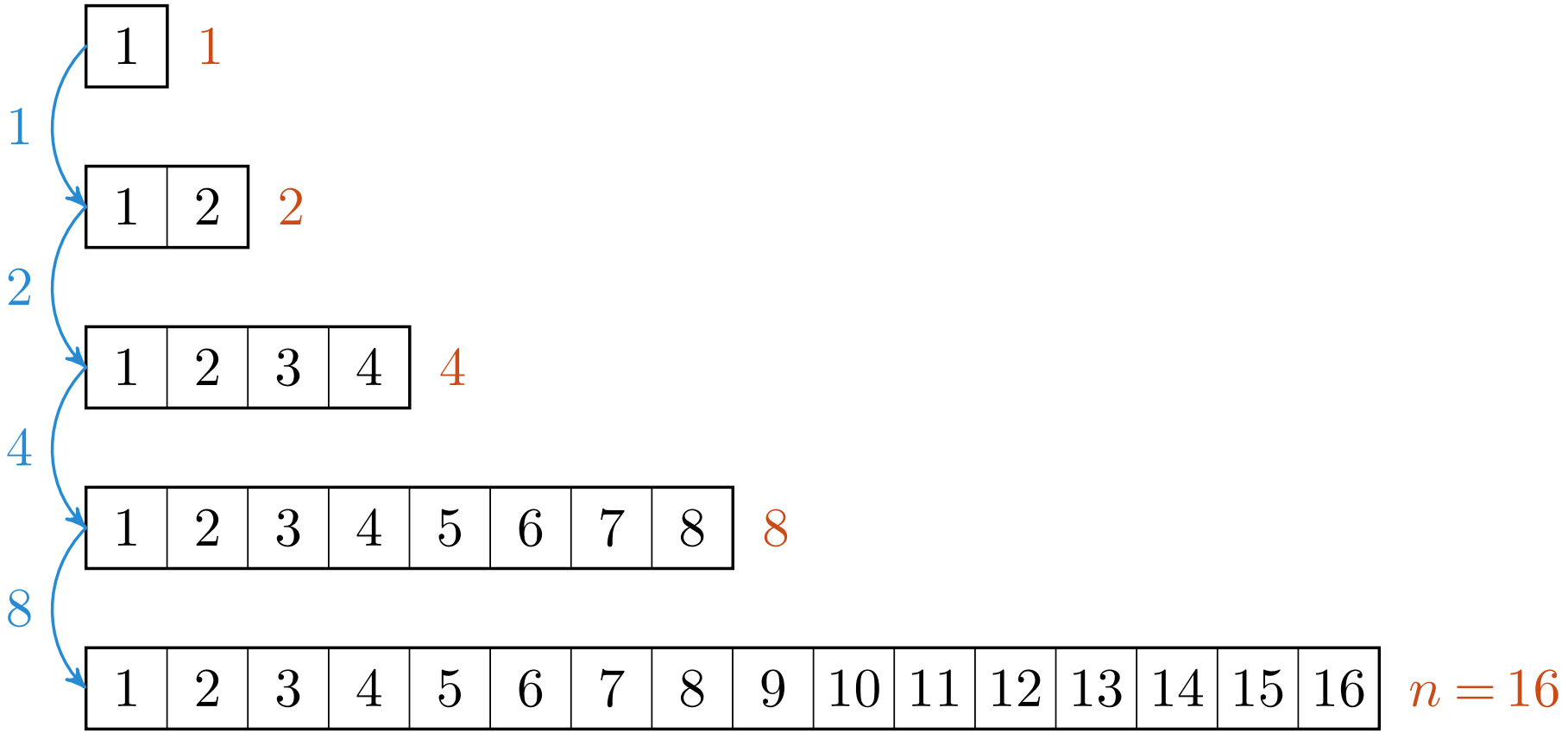

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11 12
n = 16


1
1
1
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

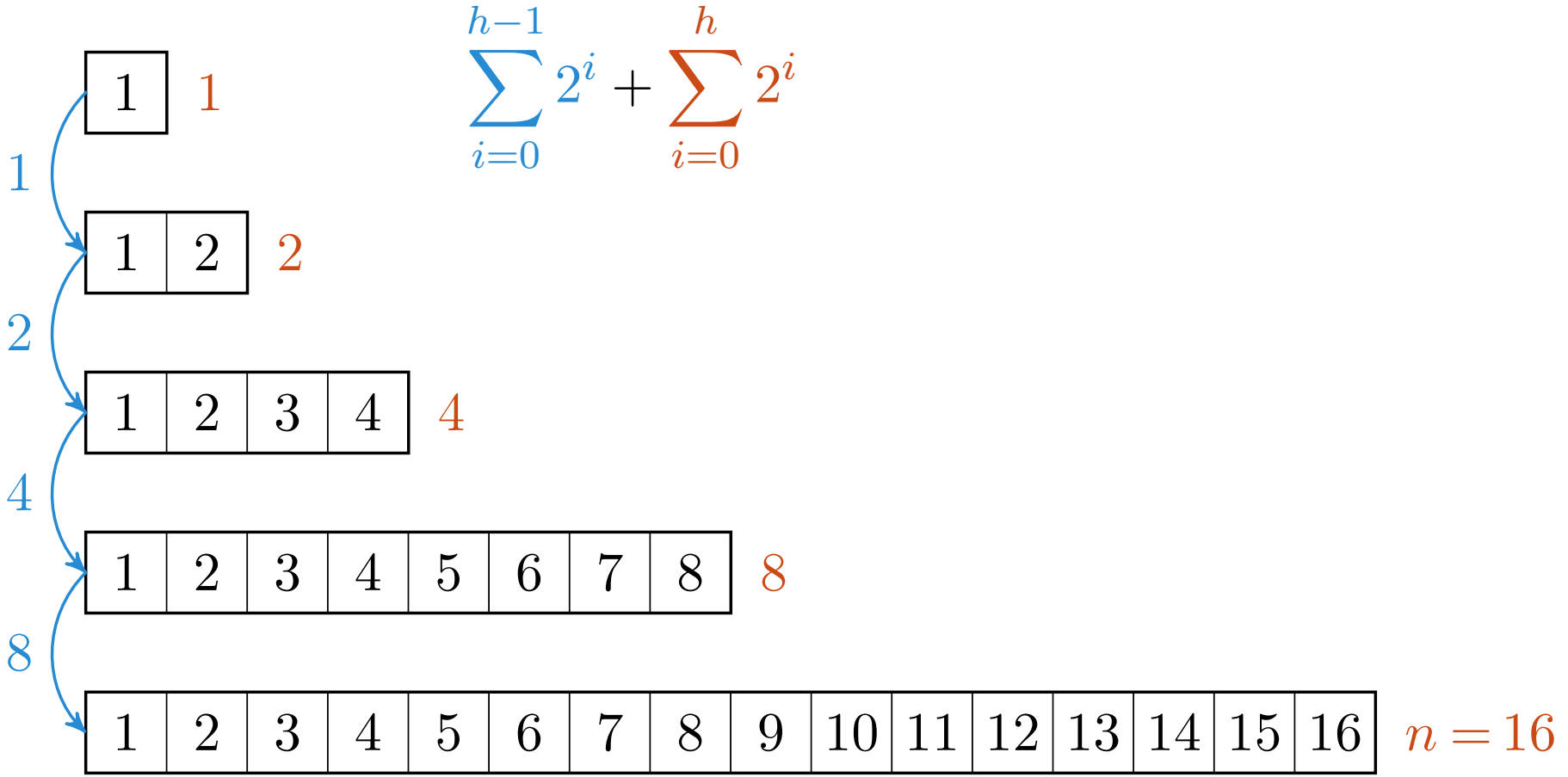

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11 12 13 14
n = 16


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

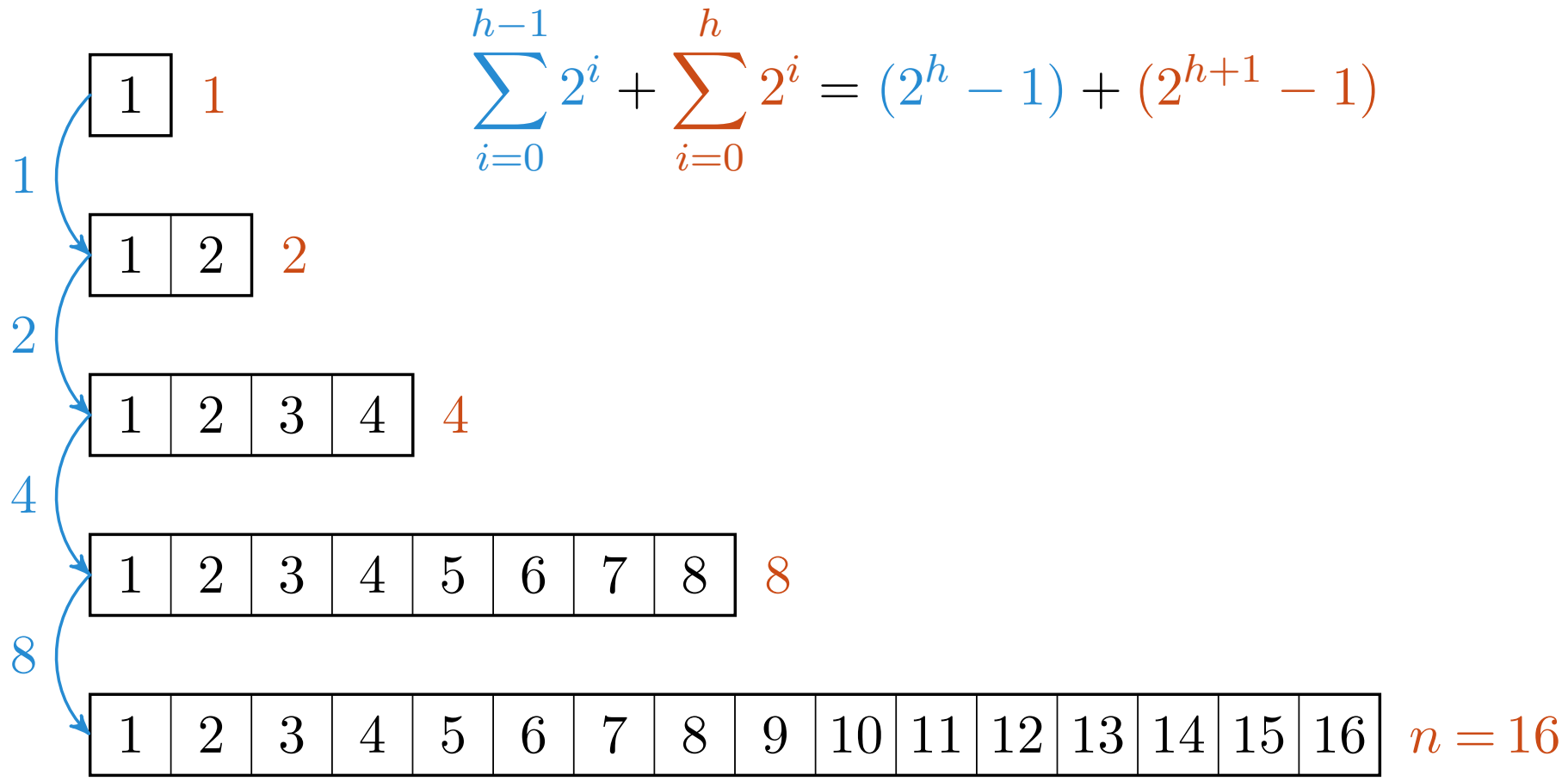

1
1
1
1 2
2
2
1 2 3 4
4
4
1 2 3 4 5 6 7 8
8
8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
n = 16



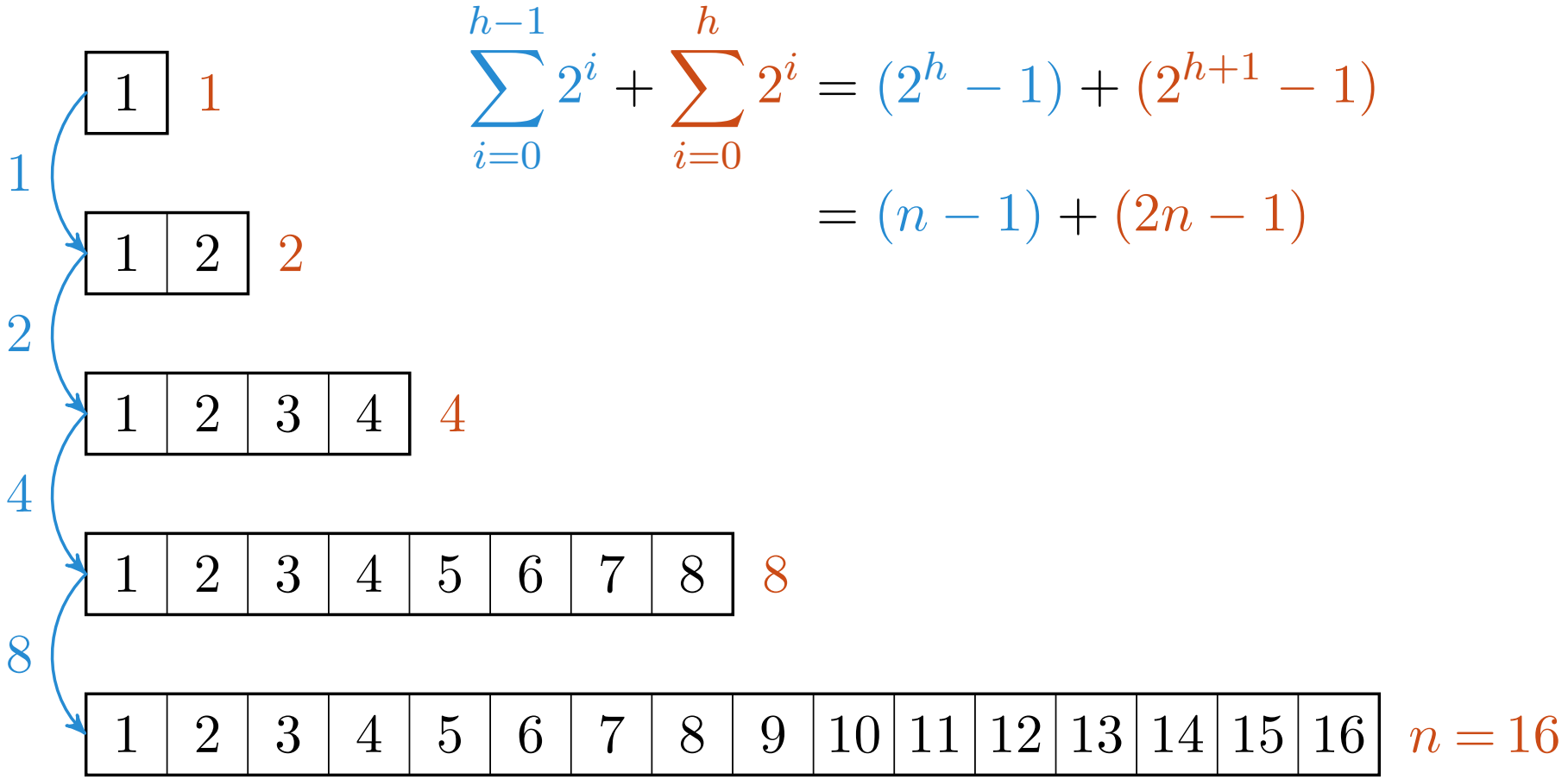
$$h = \log_2 n = 4$$


$\sum_{i=0}^{h-1} 2^i + \sum_{i=0}^{h} 2^i$
1
1 2
2
1 2 3 4
4
1 2 3 4 5 6 7 8
8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
$n = 16$

$$h = \log_2 n = 4$$


$\sum_{i=0}^{h-1} 2^i + \sum_{i=0}^{h} 2^i = (2^h - 1) + (2^{h+1} - 1)$
1
1 2
2
1 2 3 4
4
1 2 3 4 5 6 7 8
8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
$n = 16$

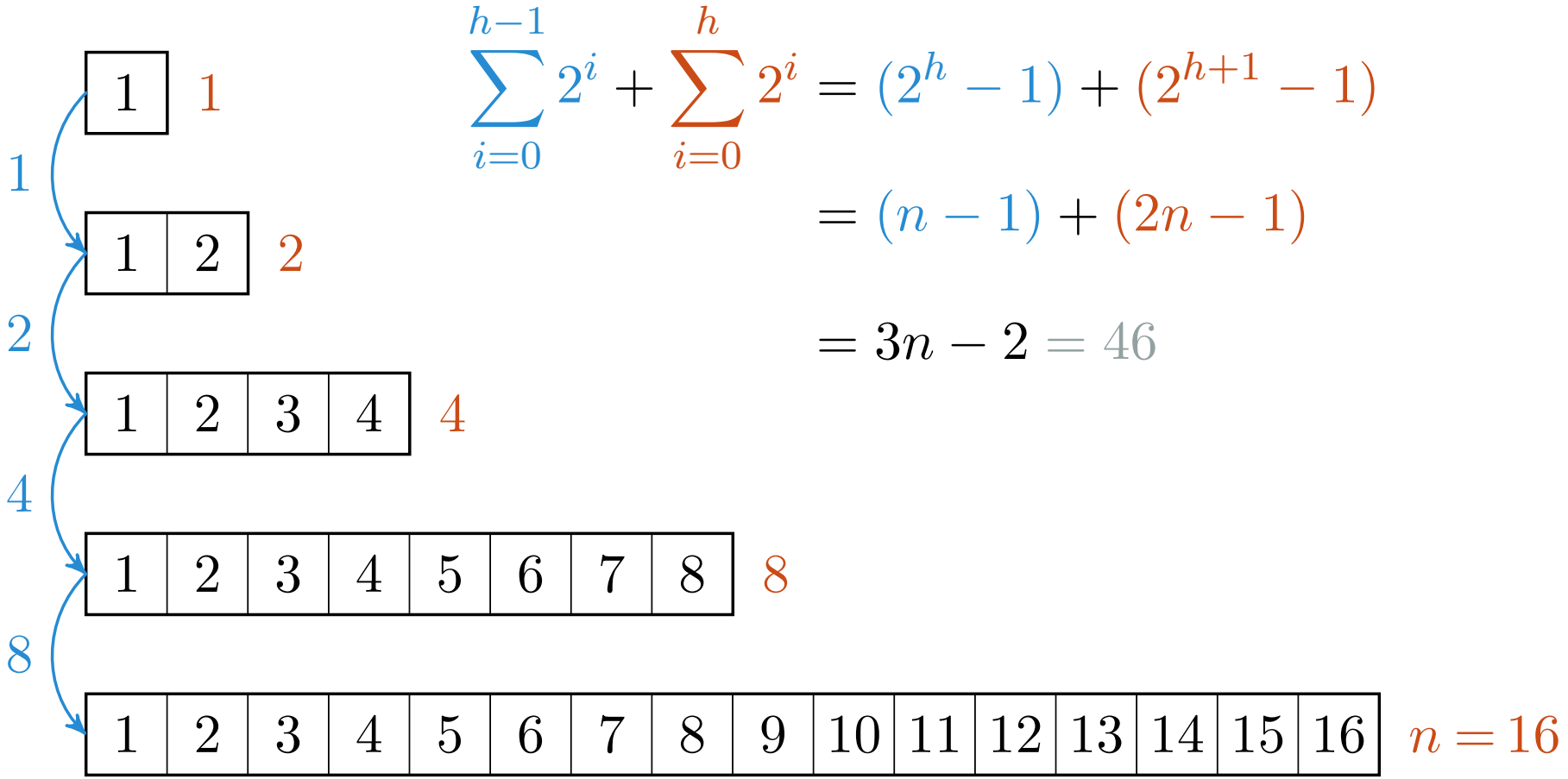
$$h = \log_2 n = 4$$



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

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

$$h = \log_2 n = 4$$



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

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

$$= 3n - 2 = 46$$

$$h = \log_2 n = 4$$

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

Med andre ord: Om vi setter inn n elementer på slutten av en tabell, så vil det i snitt ta konstant tid per element.

Lenkede lister

LIST-SEARCH(L, k)

```
LIST-SEARCH(L, k)
1   x = L.head
```

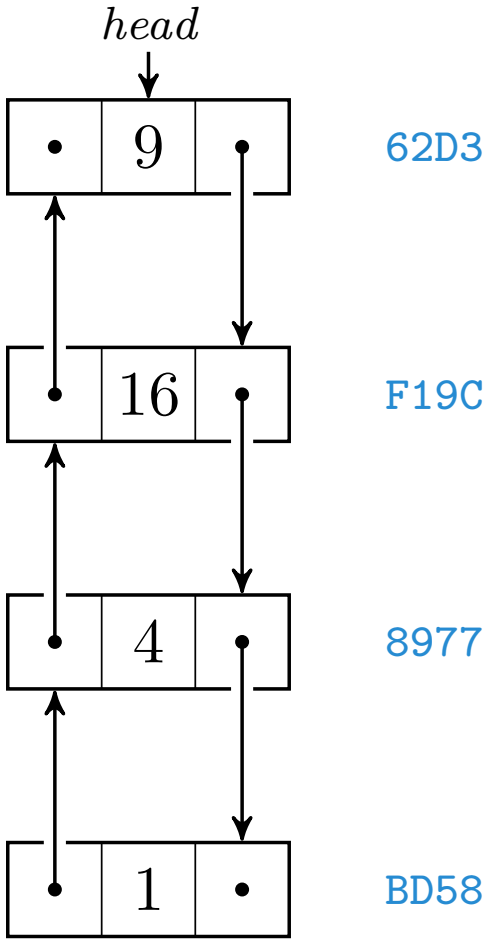
```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
```

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

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

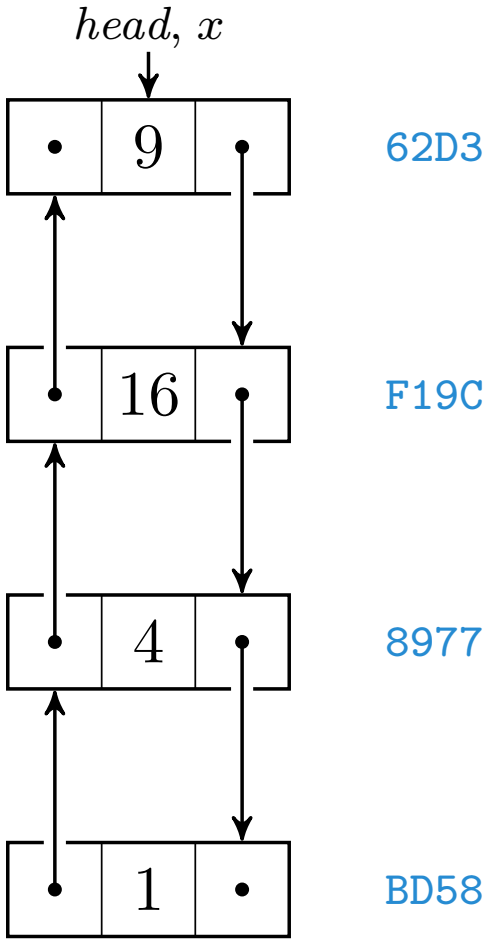
```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3        x = x.next
4  return x
```

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



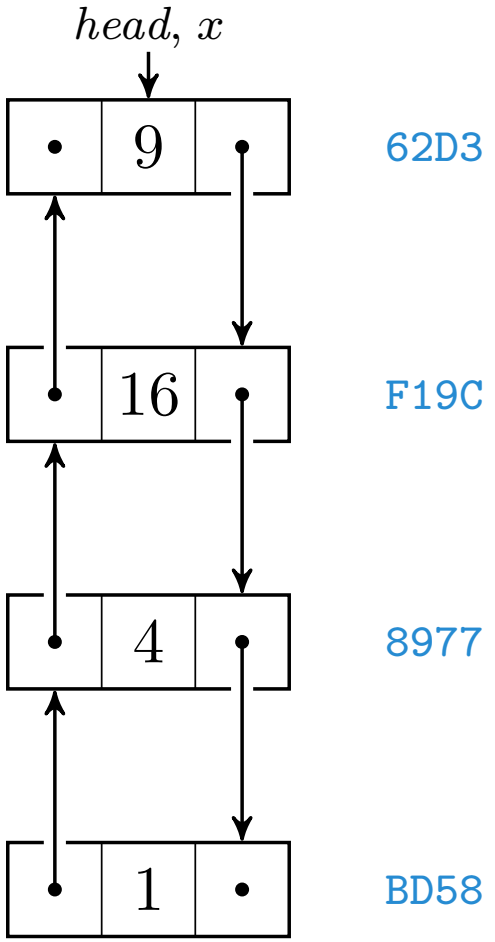
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 4, \texttt{62D3}$



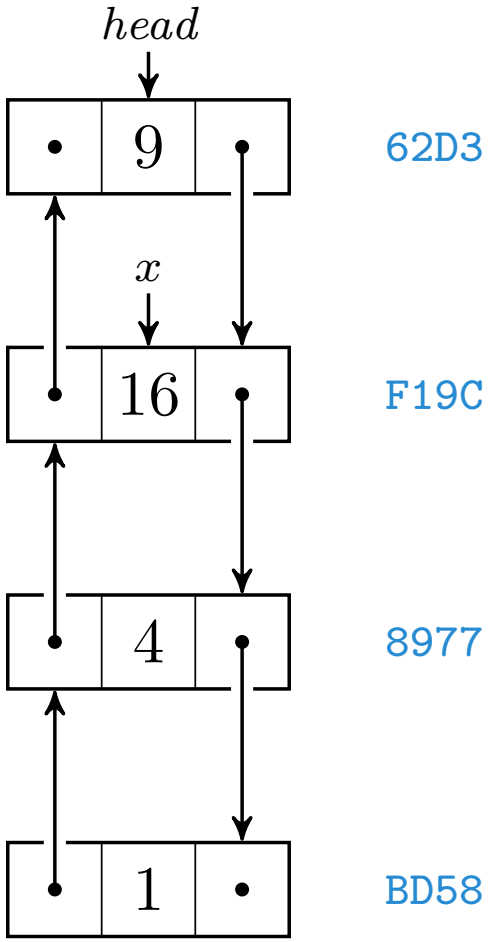
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 4, \texttt{62D3}$



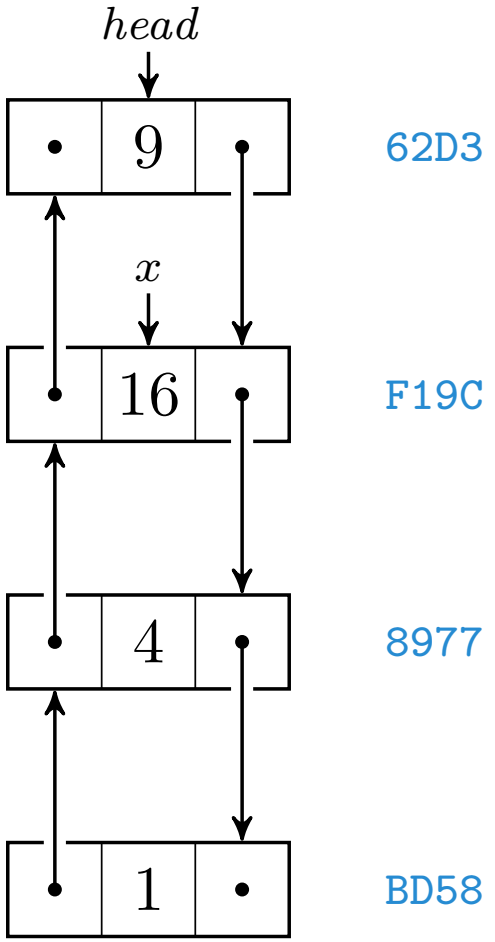
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 4, \texttt{F19C}$



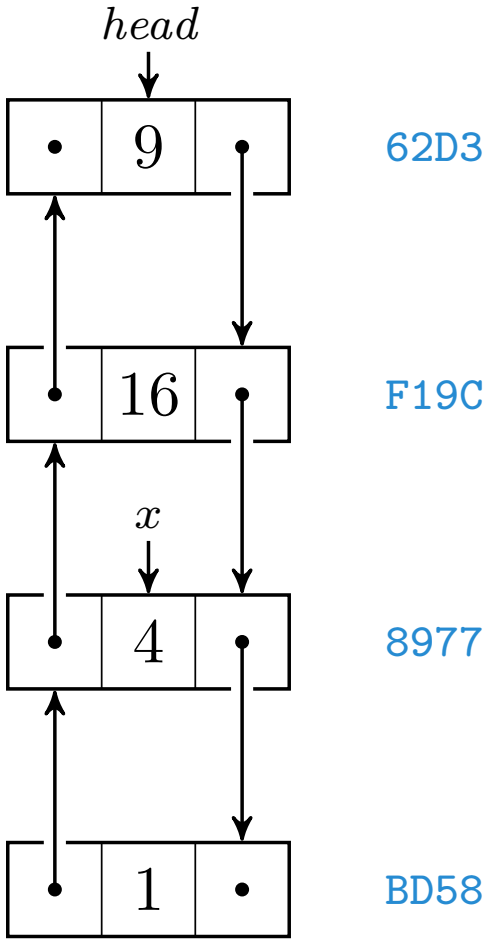
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 4, \texttt{F19C}$



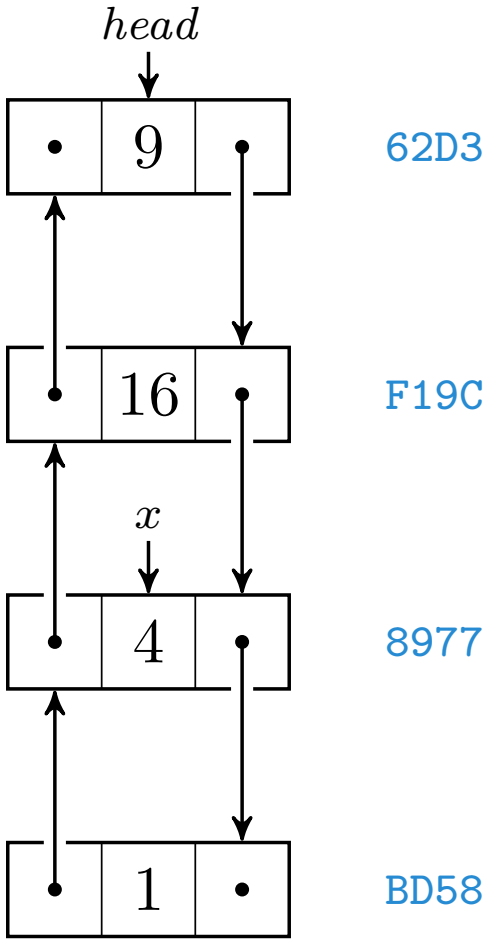
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 4, 8977$



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

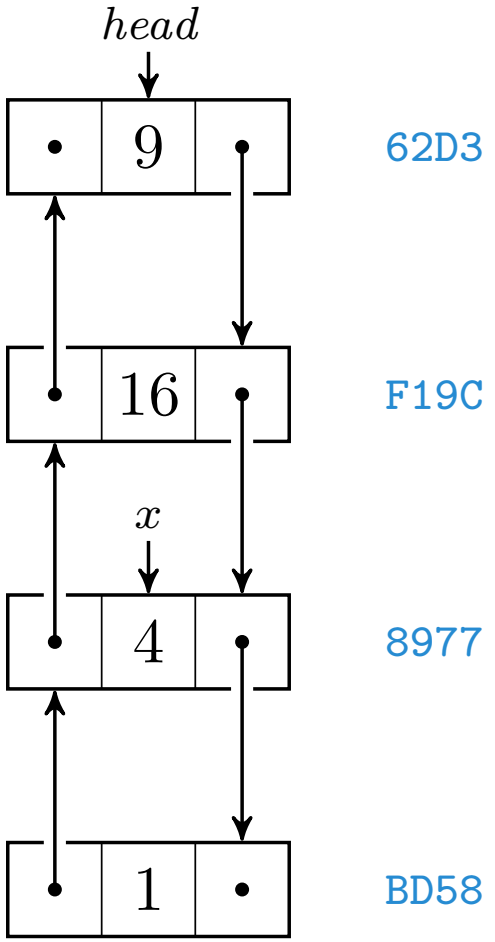
$k, x = 4, 8977$



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

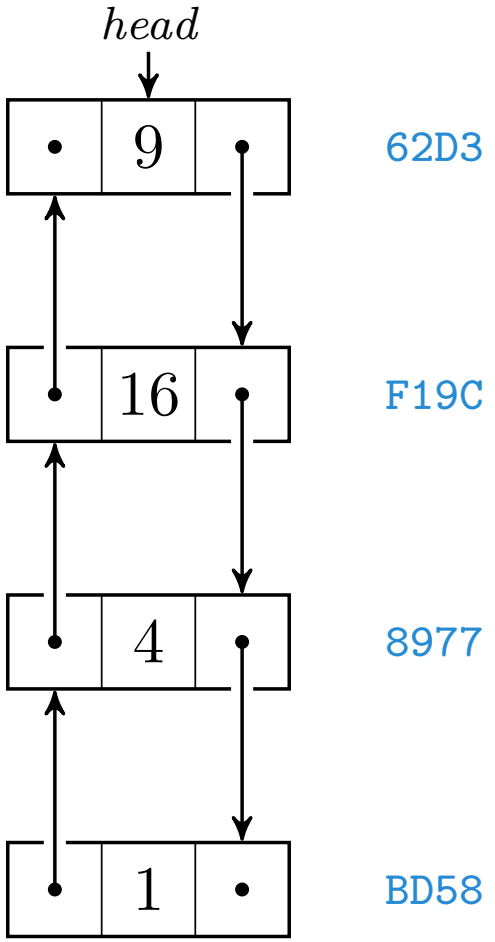
→ 8977

$k, x = 4, 8977$



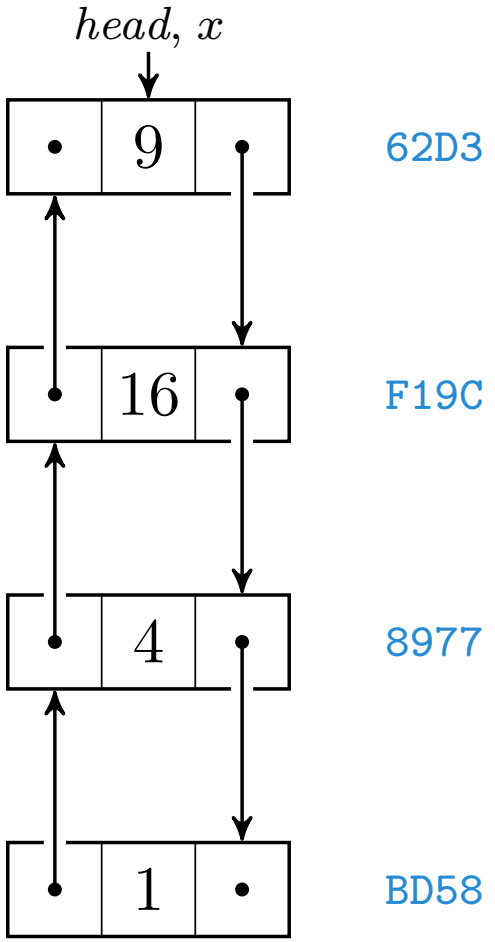
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

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



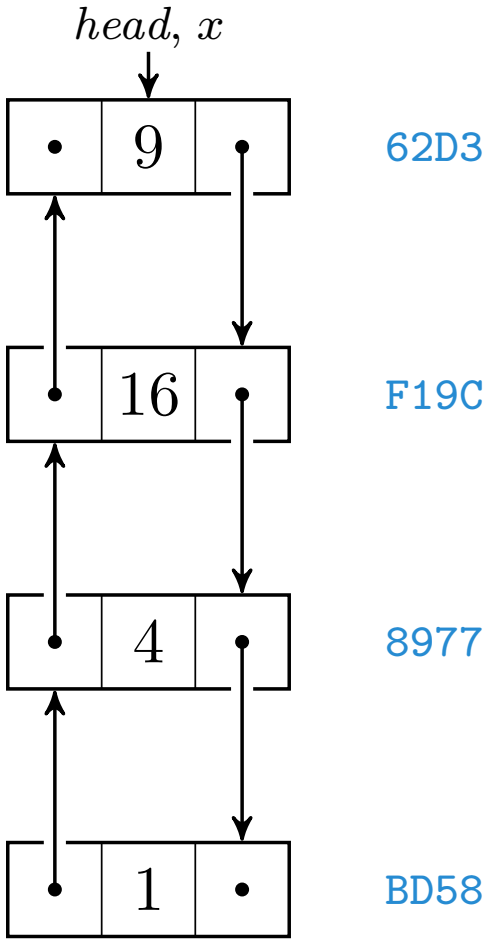
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 7, \texttt{62D3}$



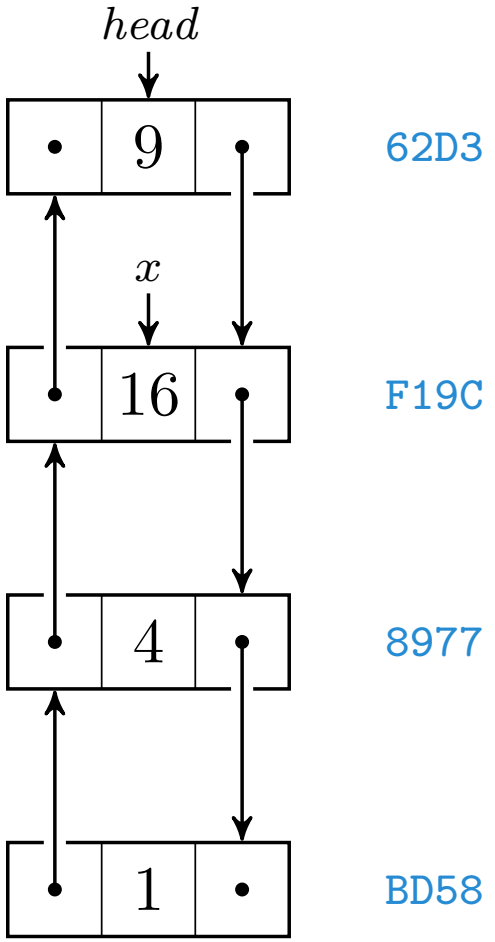
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 7, \texttt{62D3}$



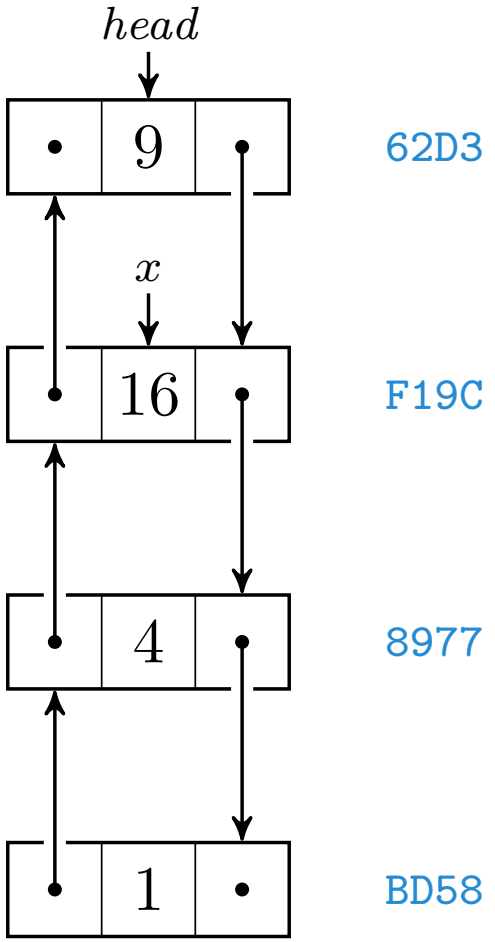
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 7, \texttt{F19C}$



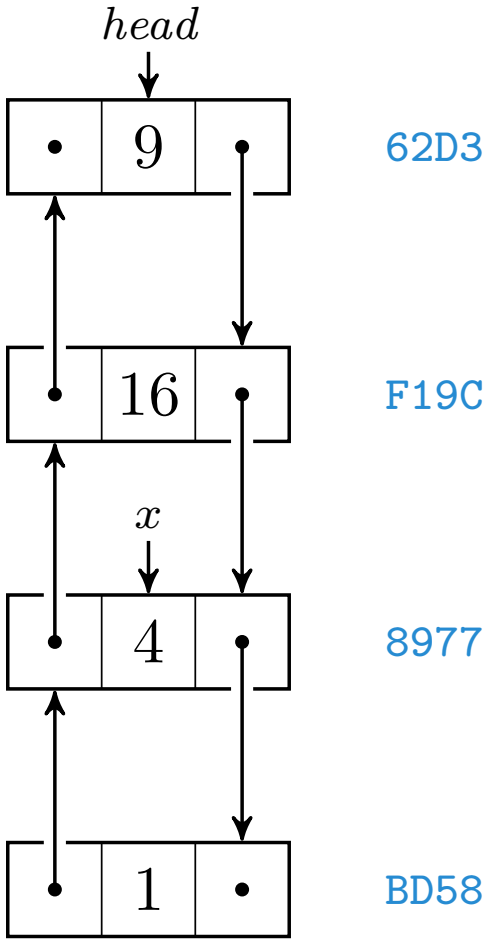
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 7, \texttt{F19C}$



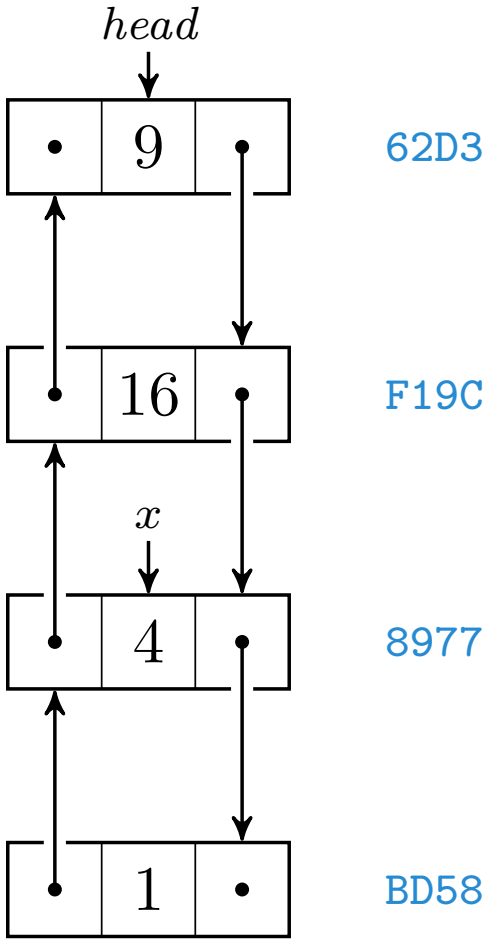
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 7, 8977$

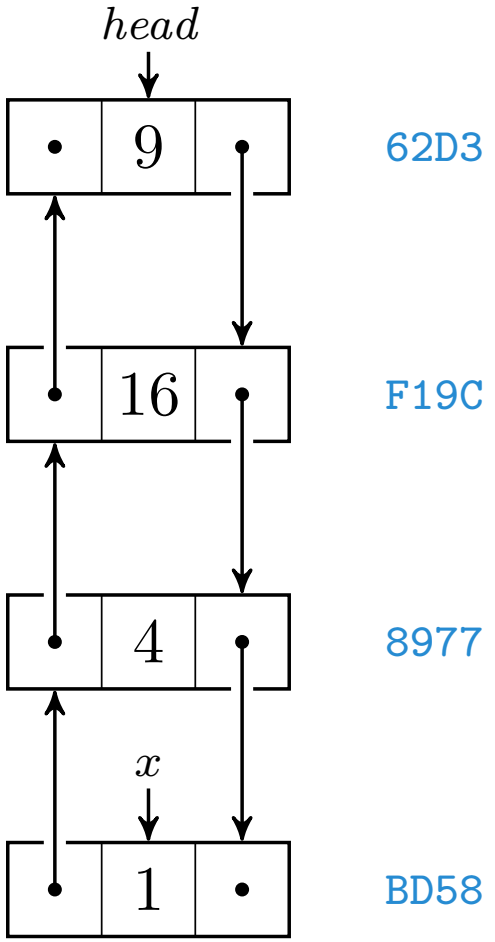


```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 7, 8977$



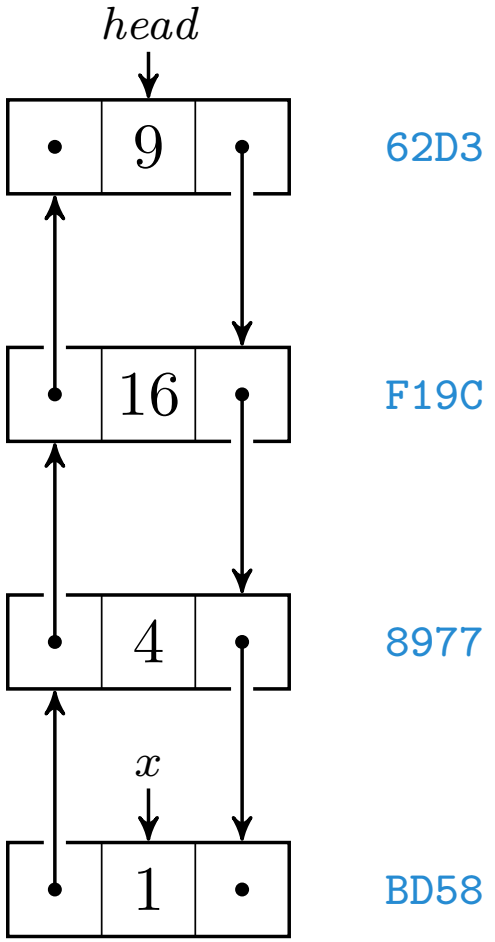
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```



$k, x = 7, \texttt{BD58}$

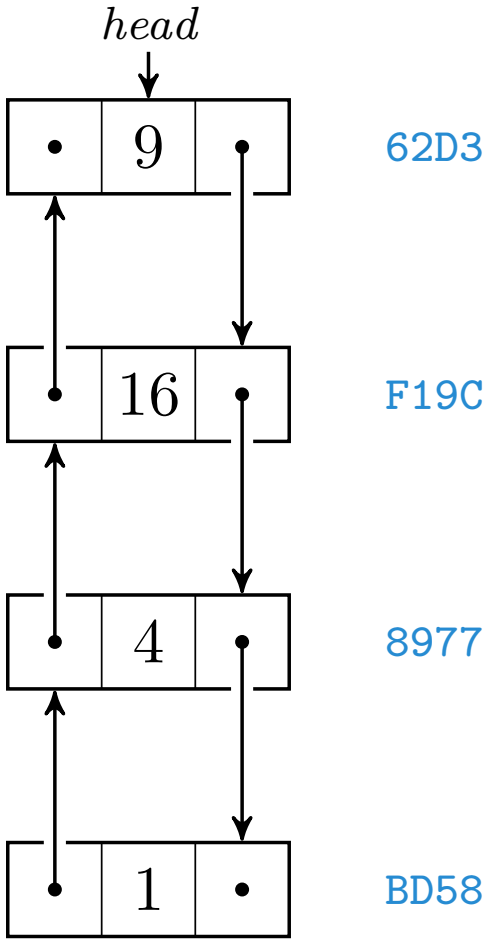
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

$k, x = 7, \texttt{BD58}$



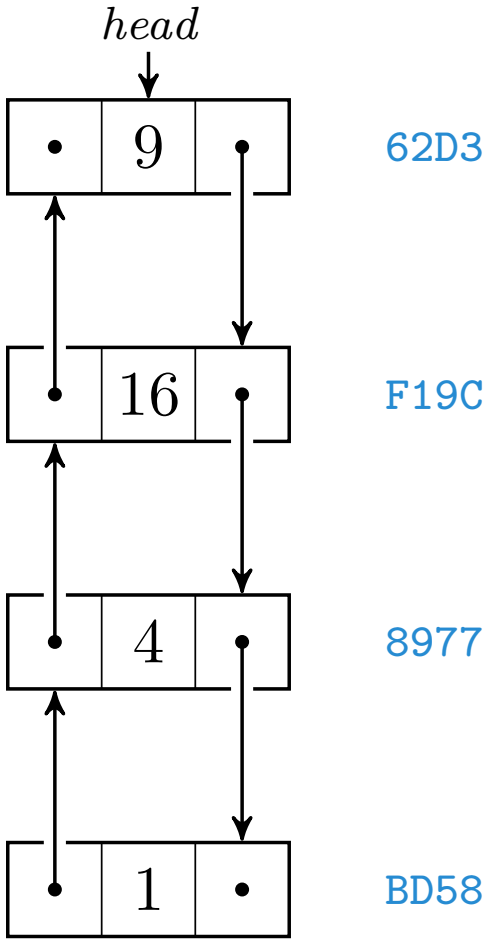
```
LIST-SEARCH(L, k)
1   x = L.head
2   while x ≠ NIL and x.key ≠ k
3         x = x.next
4   return x
```

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



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

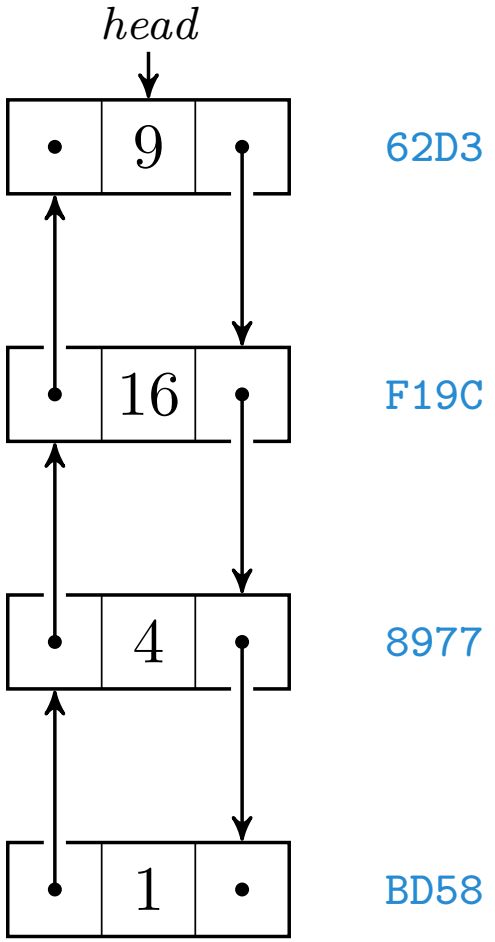
$k, x = 7, \text{NIL}$



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

→ NIL

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



LIST-INSERT(L, x)

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

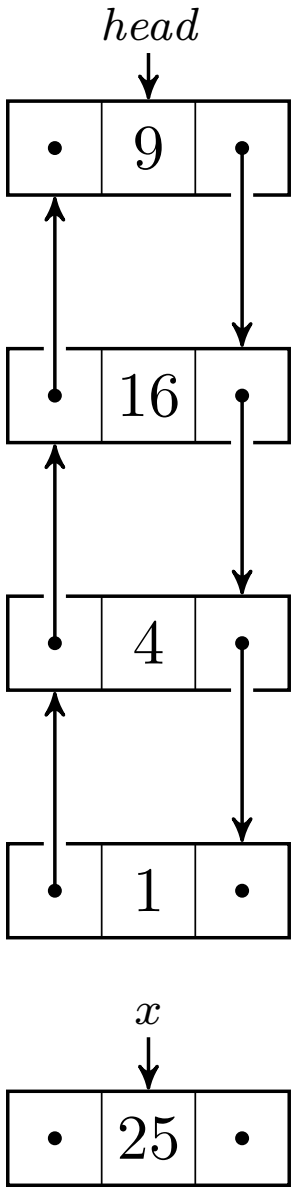
```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
```

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

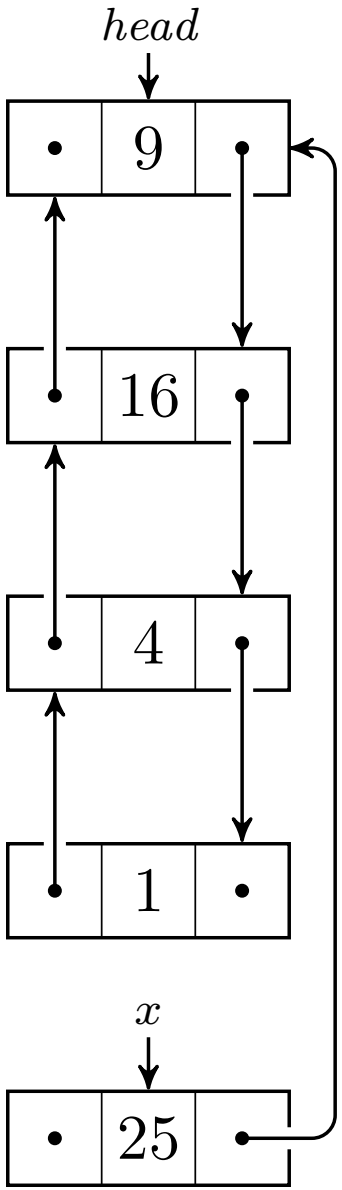
```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
3           L.head.prev = x
4   L.head = x
```

```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
3           L.head.prev = x
4   L.head = x
5   x.prev = NIL
```

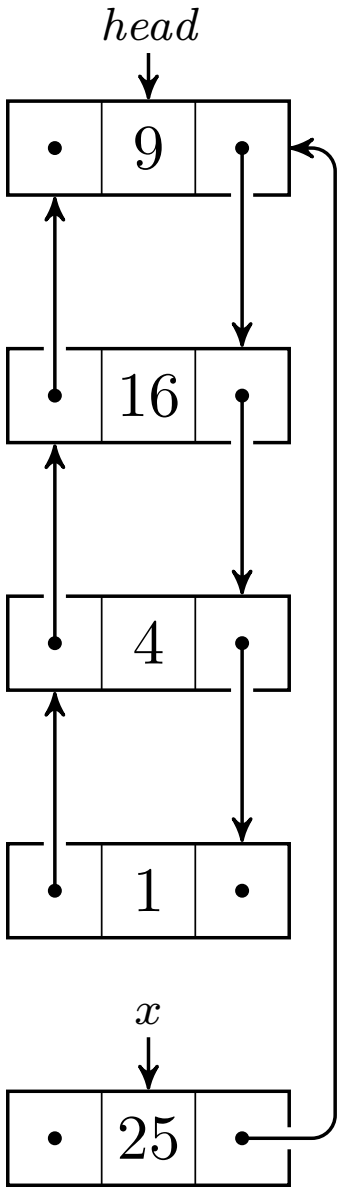
```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
3         L.head.prev = x
4   L.head = x
5   x.prev = NIL
```



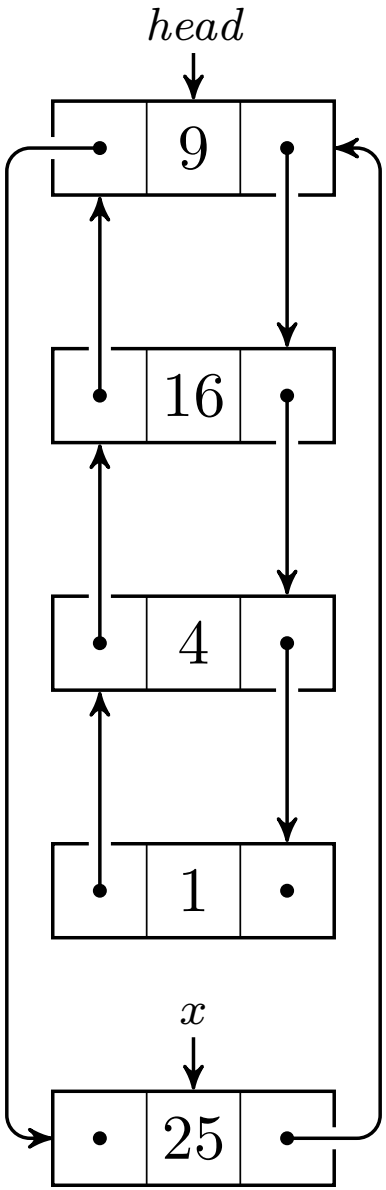
```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
3       L.head.prev = x
4   L.head = x
5   x.prev = NIL
```



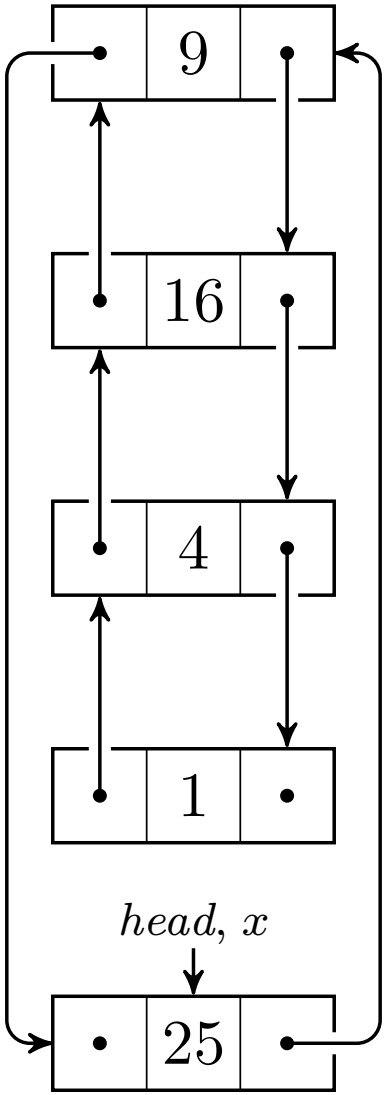
```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
3           L.head.prev = x
4   L.head = x
5   x.prev = NIL
```



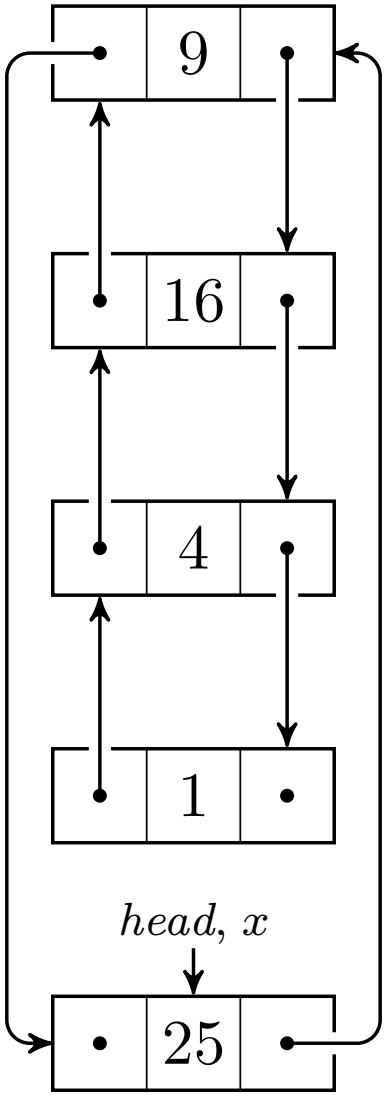
```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
3         L.head.prev = x
4   L.head = x
5   x.prev = NIL
```



```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
3         L.head.prev = x
4   L.head = x
5   x.prev = NIL
```



```
LIST-INSERT(L, x)
1   x.next = L.head
2   if L.head ≠ NIL
3         L.head.prev = x
4   L.head = x
5   x.prev = NIL
```



LIST-DELETE(L, x)

```
LIST-DELETE(L, x)
1  if x.prev ≠ NIL
```

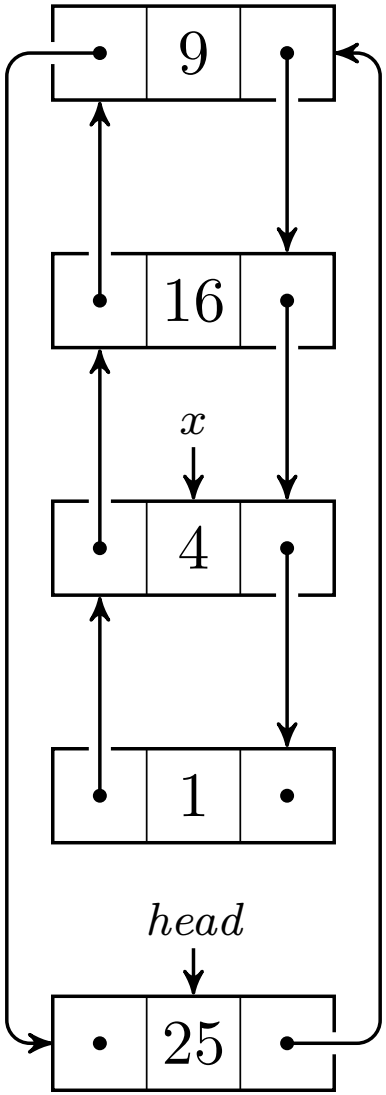
```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2           x.prev.next = x.next
```

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

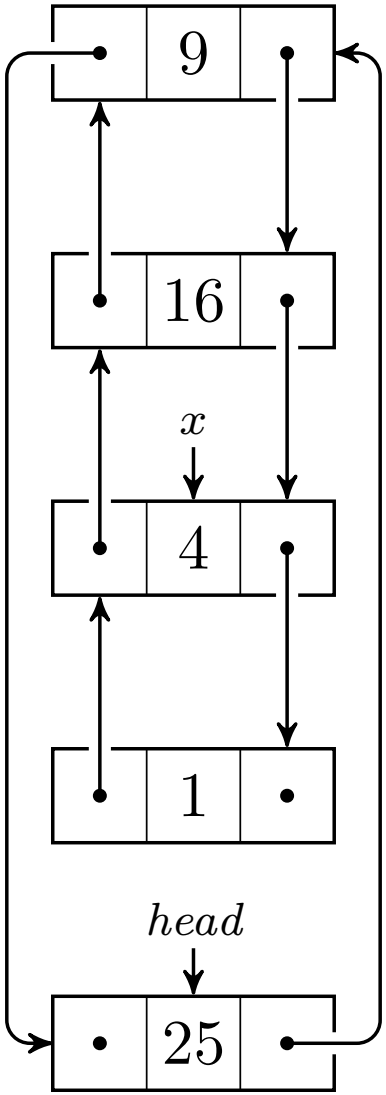
```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2         x.prev.next = x.next
3   else L.head = x.next
4   if x.next ≠ NIL
```

```
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
5         x.next.prev = x.prev
```

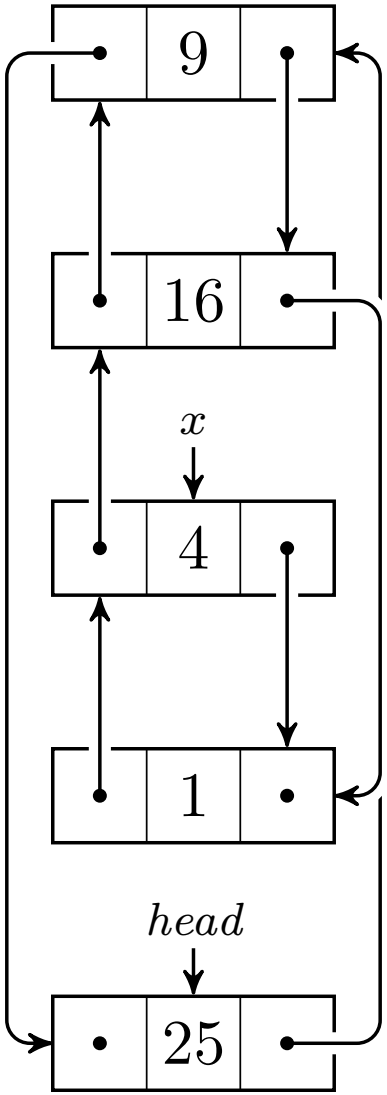
```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2         x.prev.next = x.next
3   else L.head = x.next
4   if x.next ≠ NIL
5         x.next.prev = x.prev
```



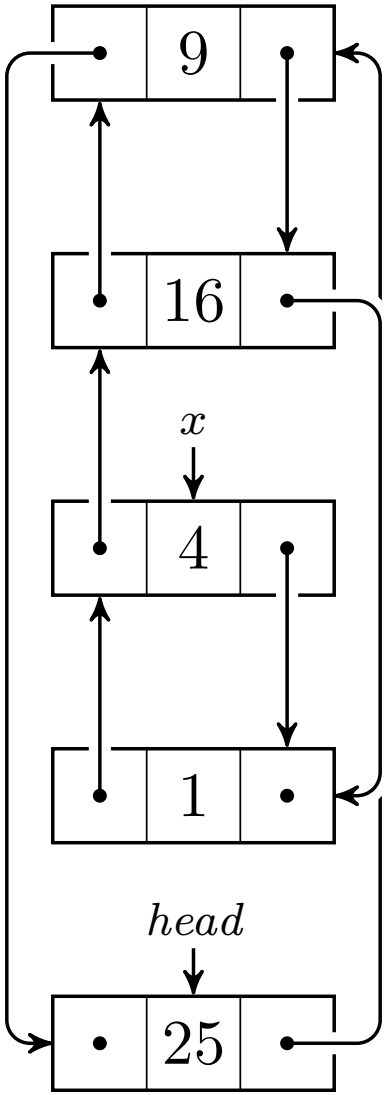
```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2         x.prev.next = x.next
3   else L.head = x.next
4   if x.next ≠ NIL
5         x.next.prev = x.prev
```



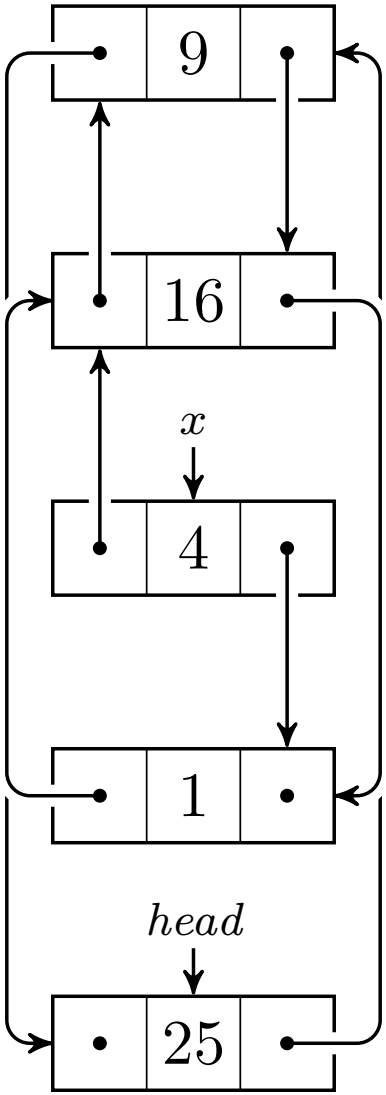
```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2         x.prev.next = x.next
3   else L.head = x.next
4   if x.next ≠ NIL
5         x.next.prev = x.prev
```



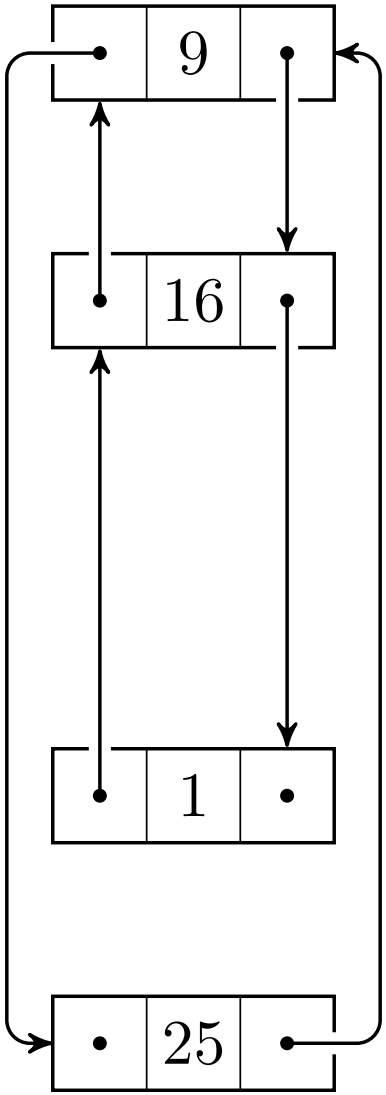
```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2         x.prev.next = x.next
3   else L.head = x.next
4   if x.next ≠ NIL
5         x.next.prev = x.prev
```



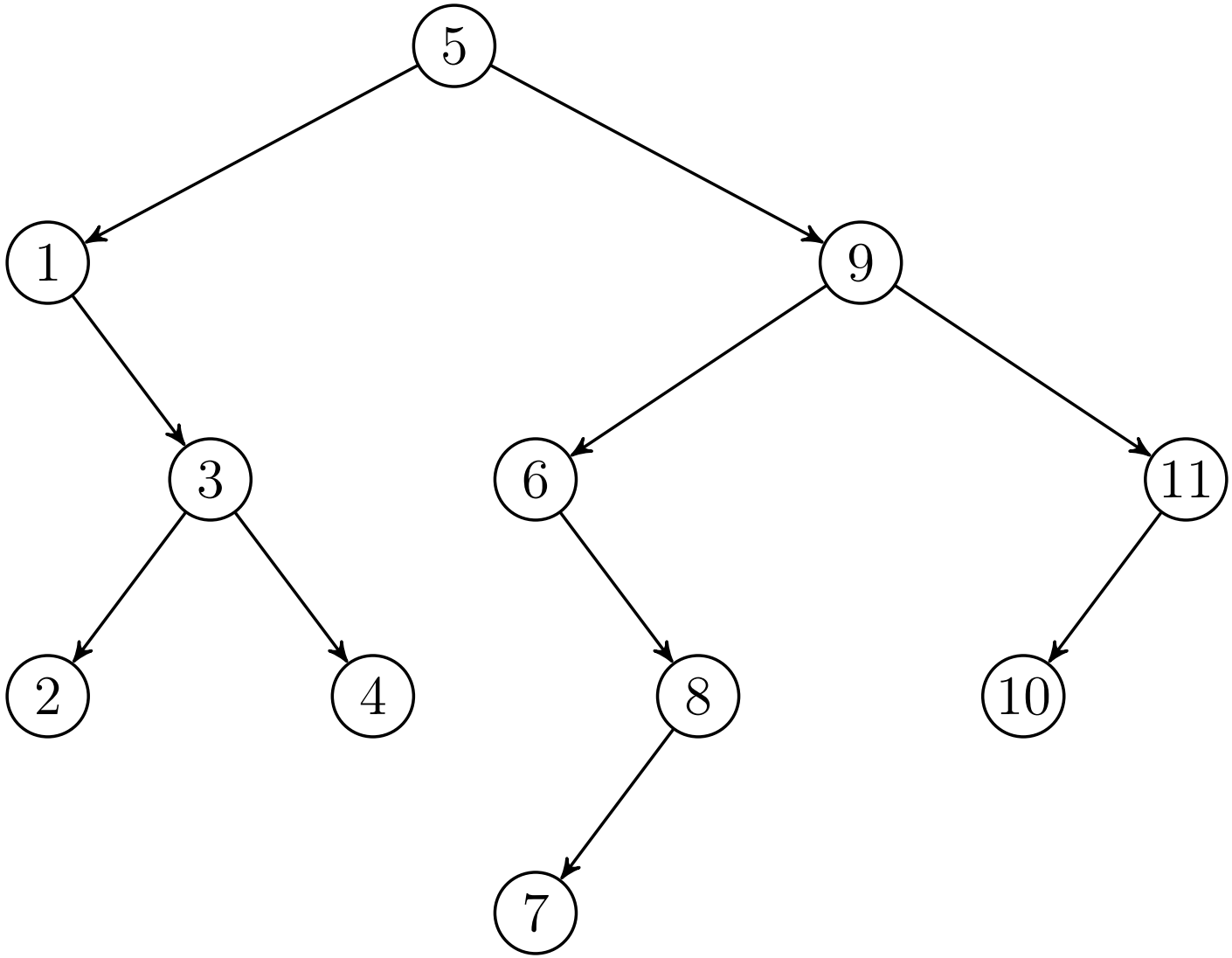
```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2         x.prev.next = x.next
3   else L.head = x.next
4   if x.next ≠ NIL
5         x.next.prev = x.prev
```

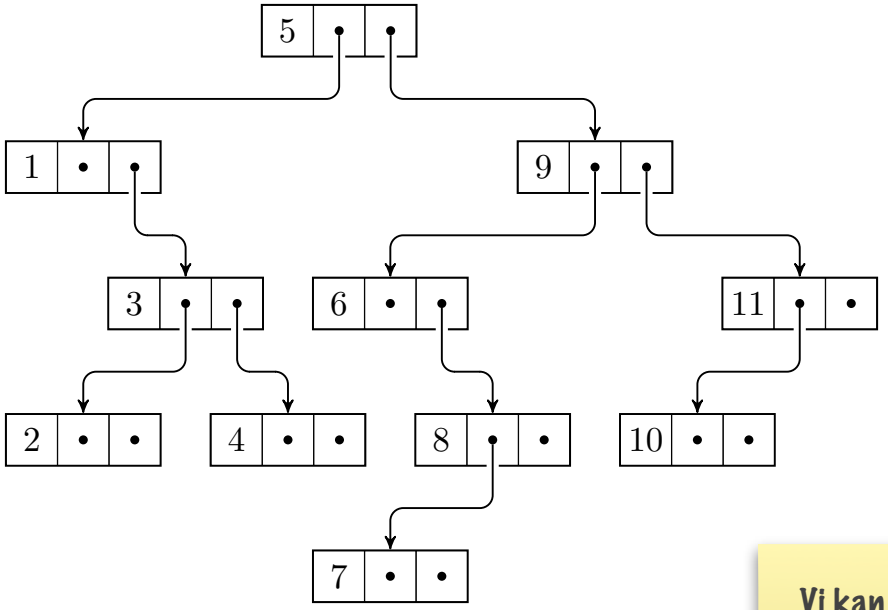


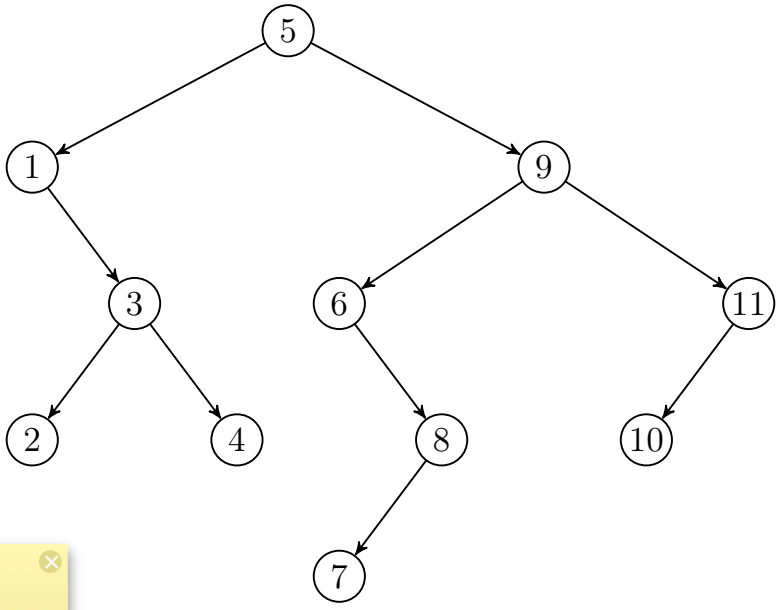
```
LIST-DELETE(L, x)
1   if x.prev ≠ NIL
2         x.prev.next = x.next
3   else L.head = x.next
4   if x.next ≠ NIL
5         x.next.prev = x.prev
```



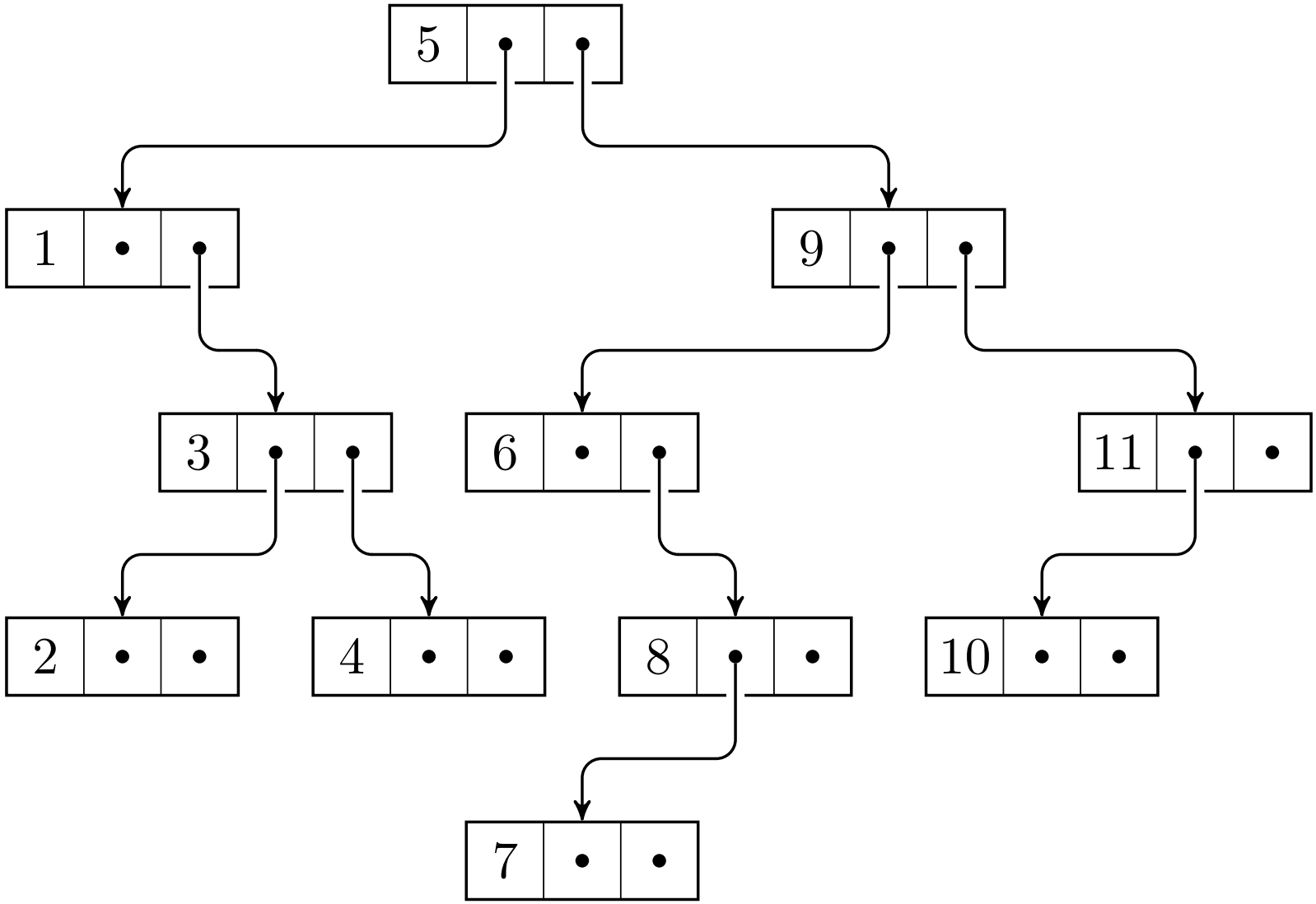
Binære rotfaste trær


5
1
9
3
6
11
2
4
8
10
7

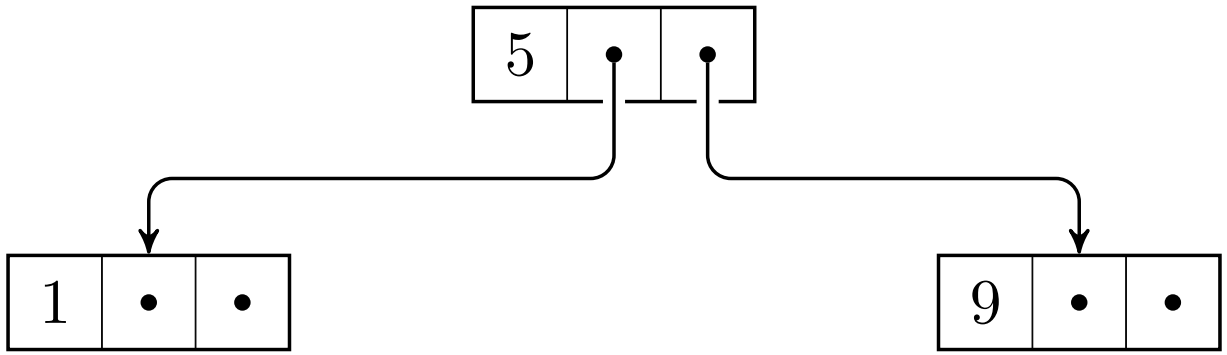


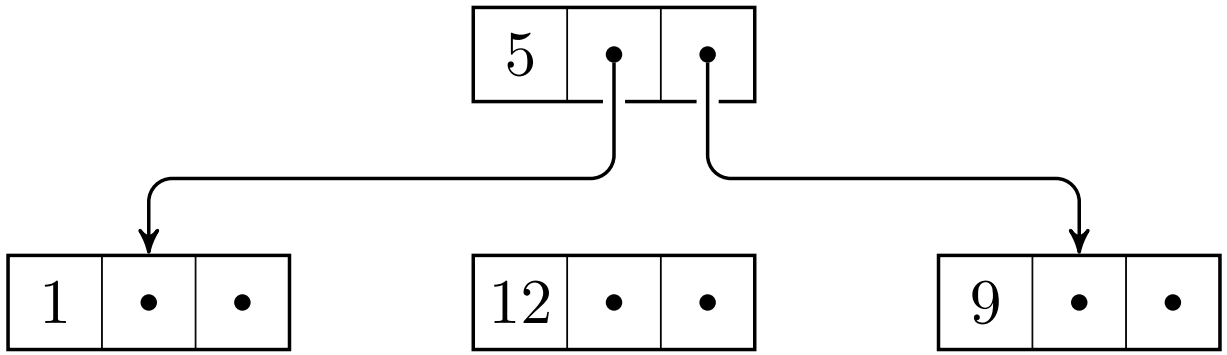


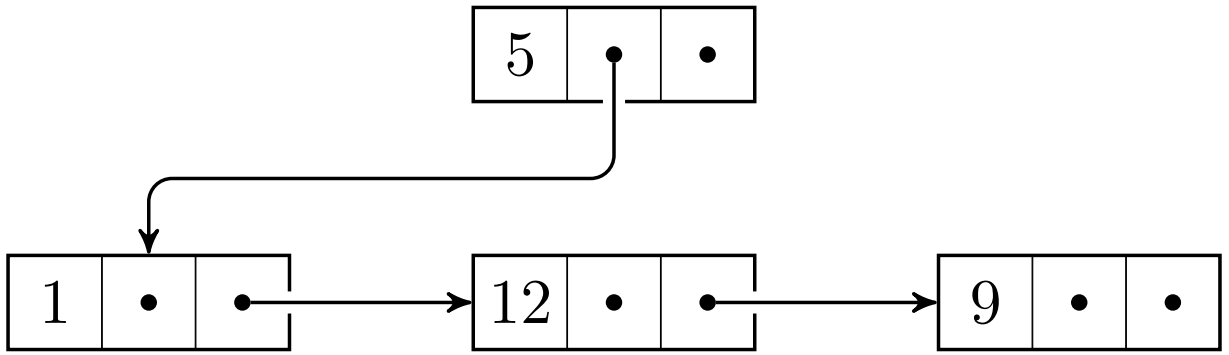
Vi kan implementere binære trær ved å la hver node ha pekere til venstre og høyre barnenode.


5
1
9
3
6
11
2
4
8
10
7

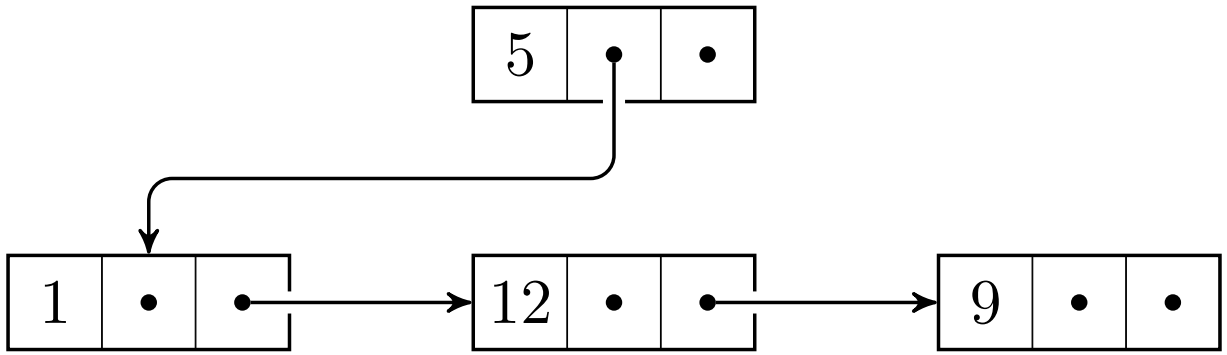
Generelle rotfaste trær


5
1
9


5
1
12
9


5
1
12
9

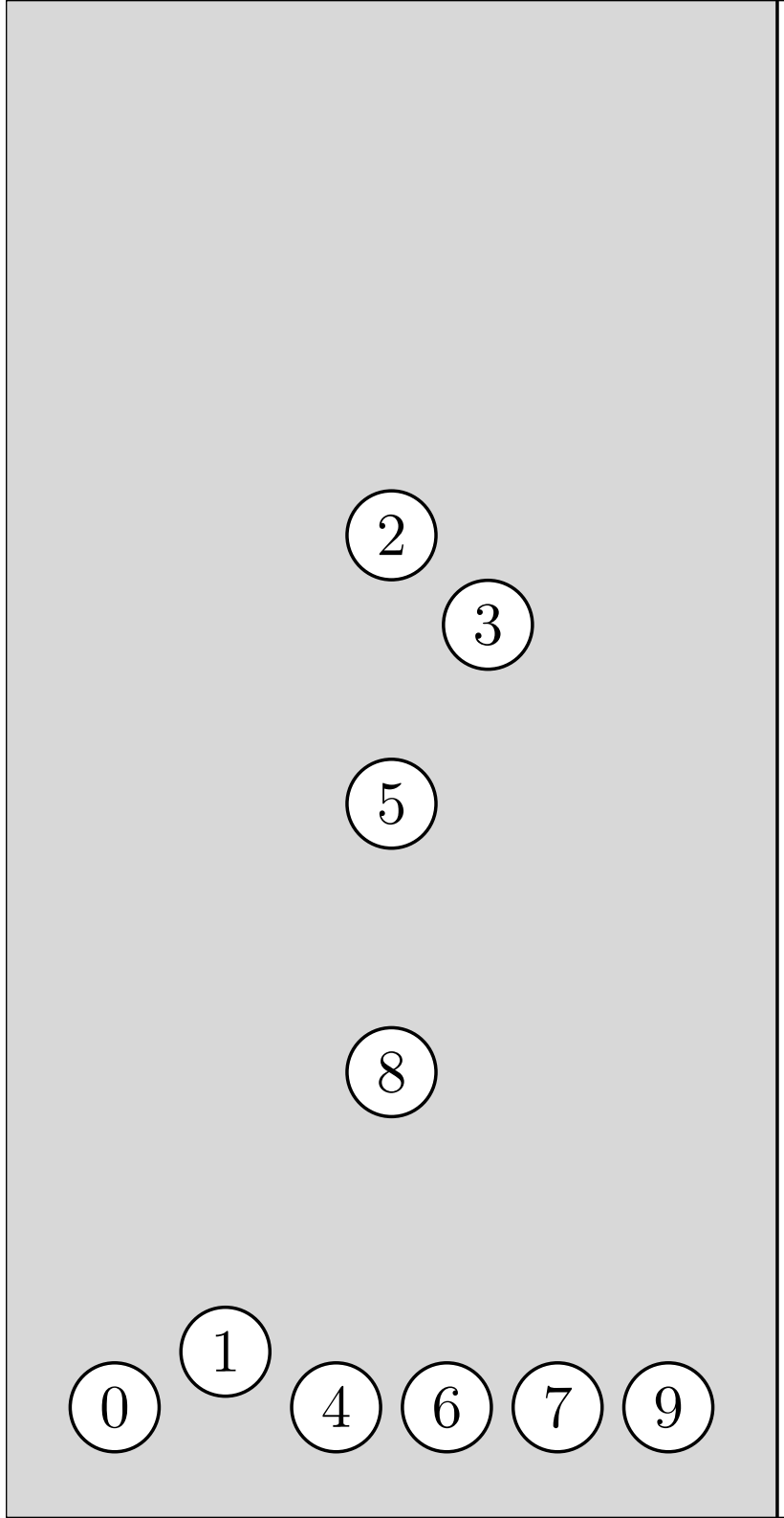
Vi kan implementere mer generelle trær slik:

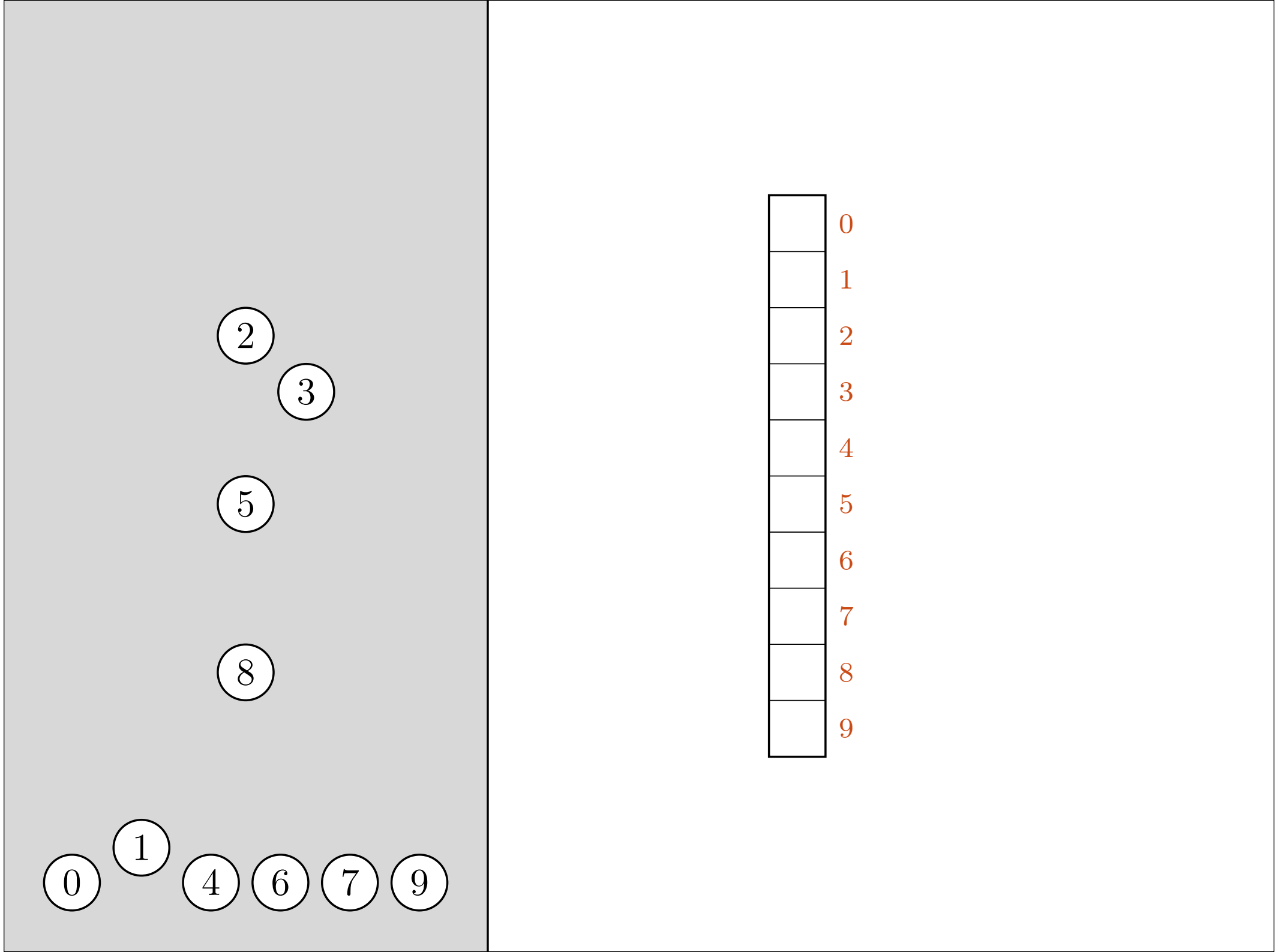


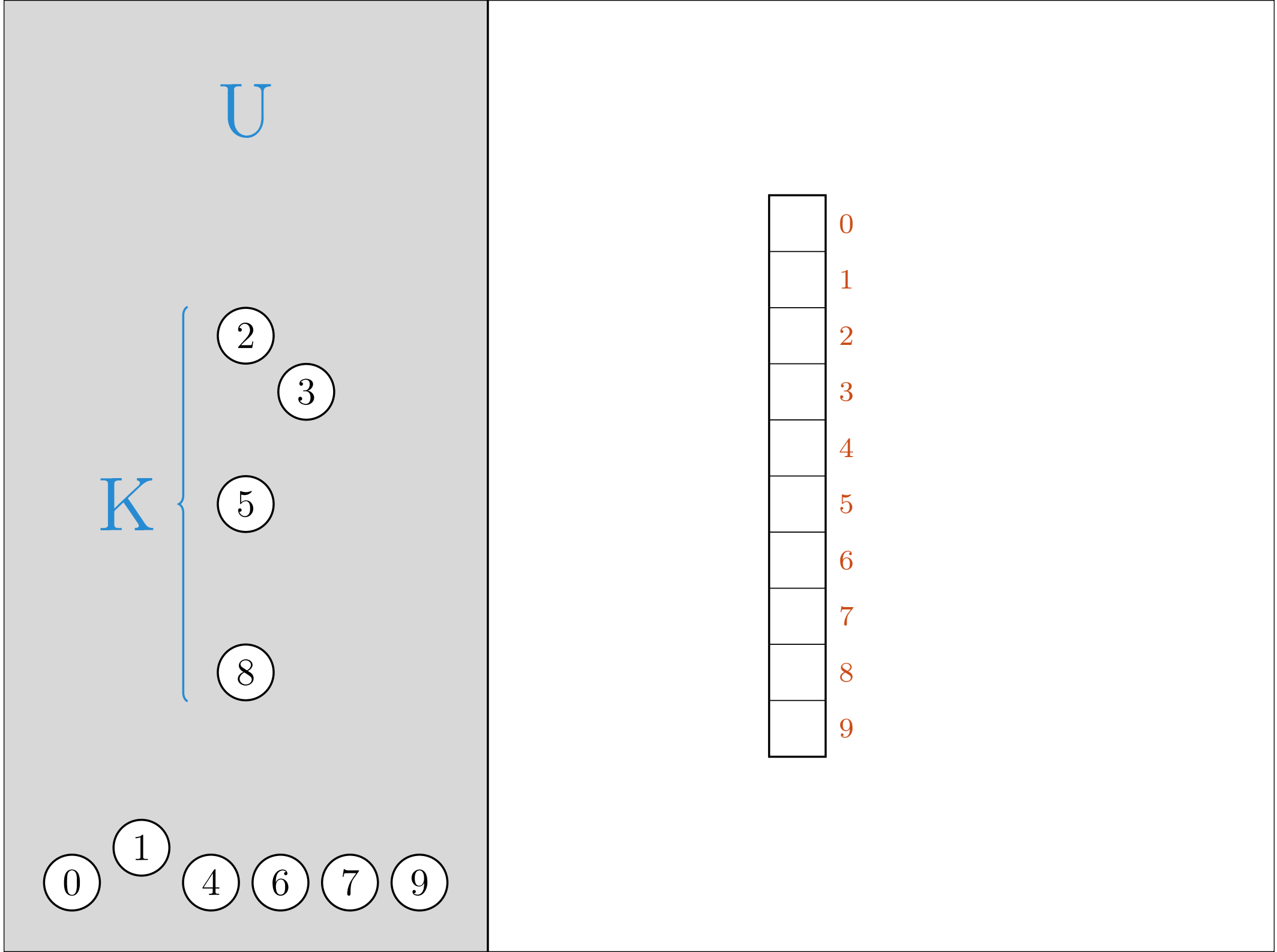
Left child, right sibling

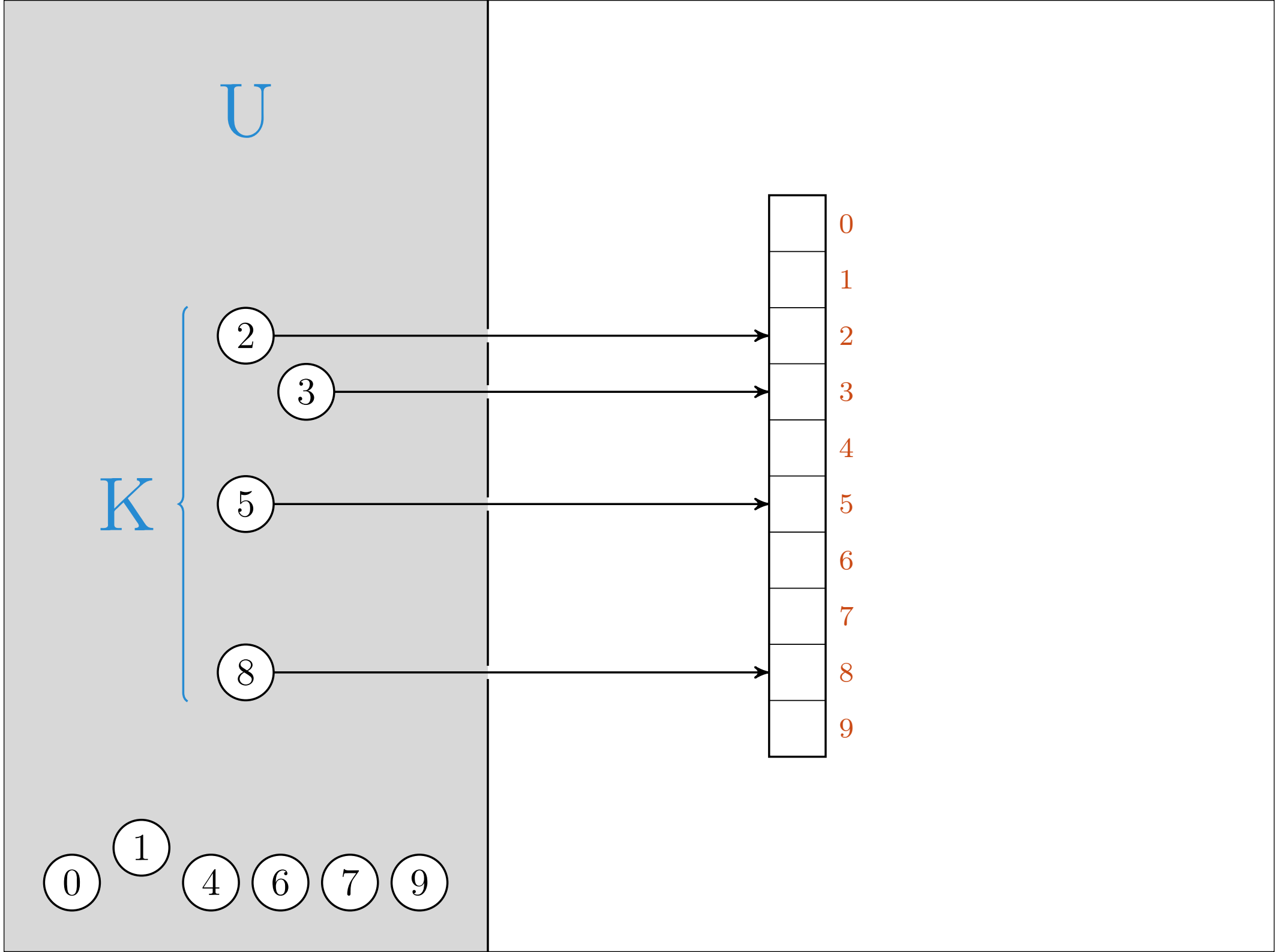
Kalles også «First child, next sibling»

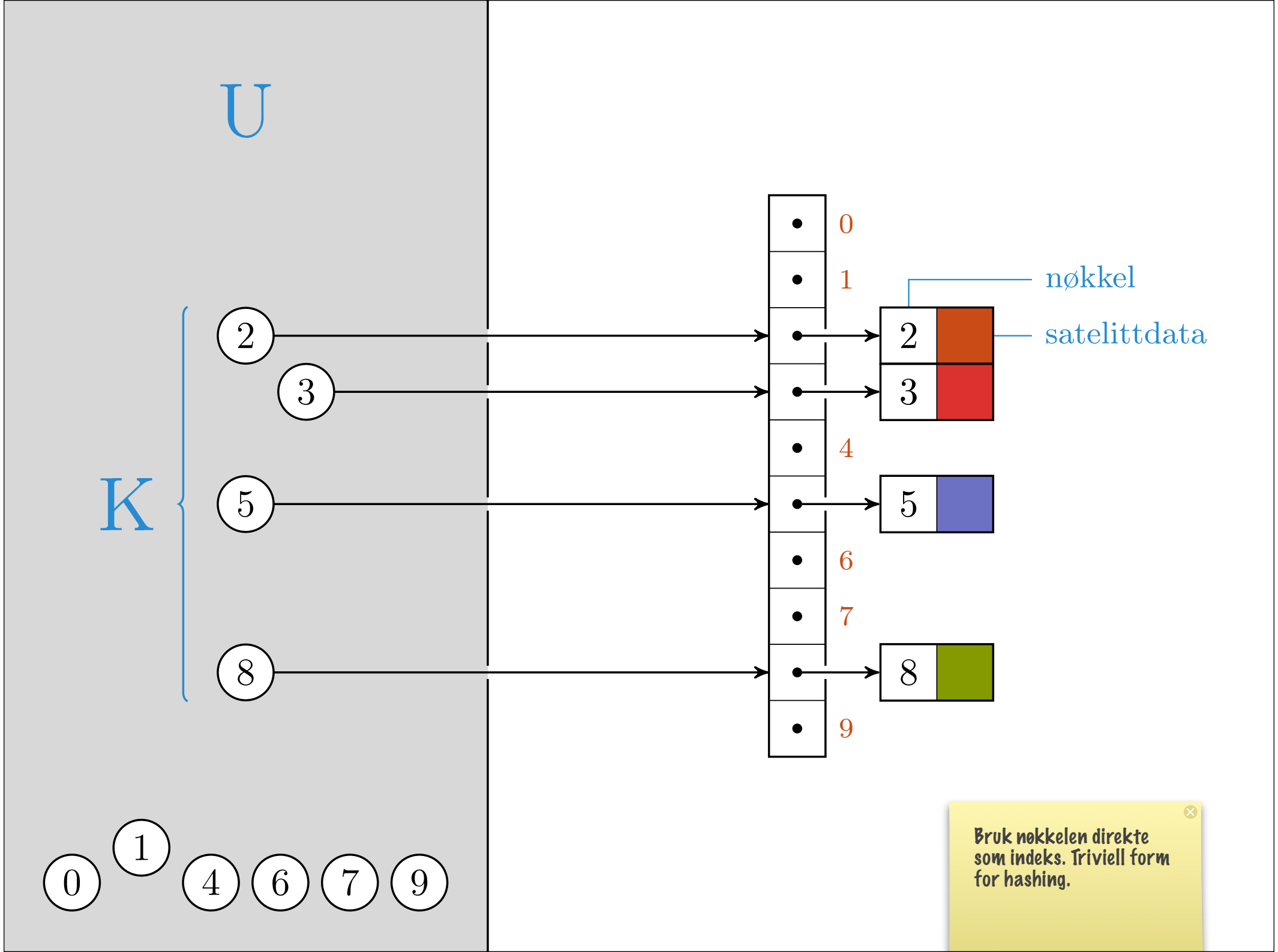
Direkte adressering


2
3
5
8
1
0
4
6
7
9

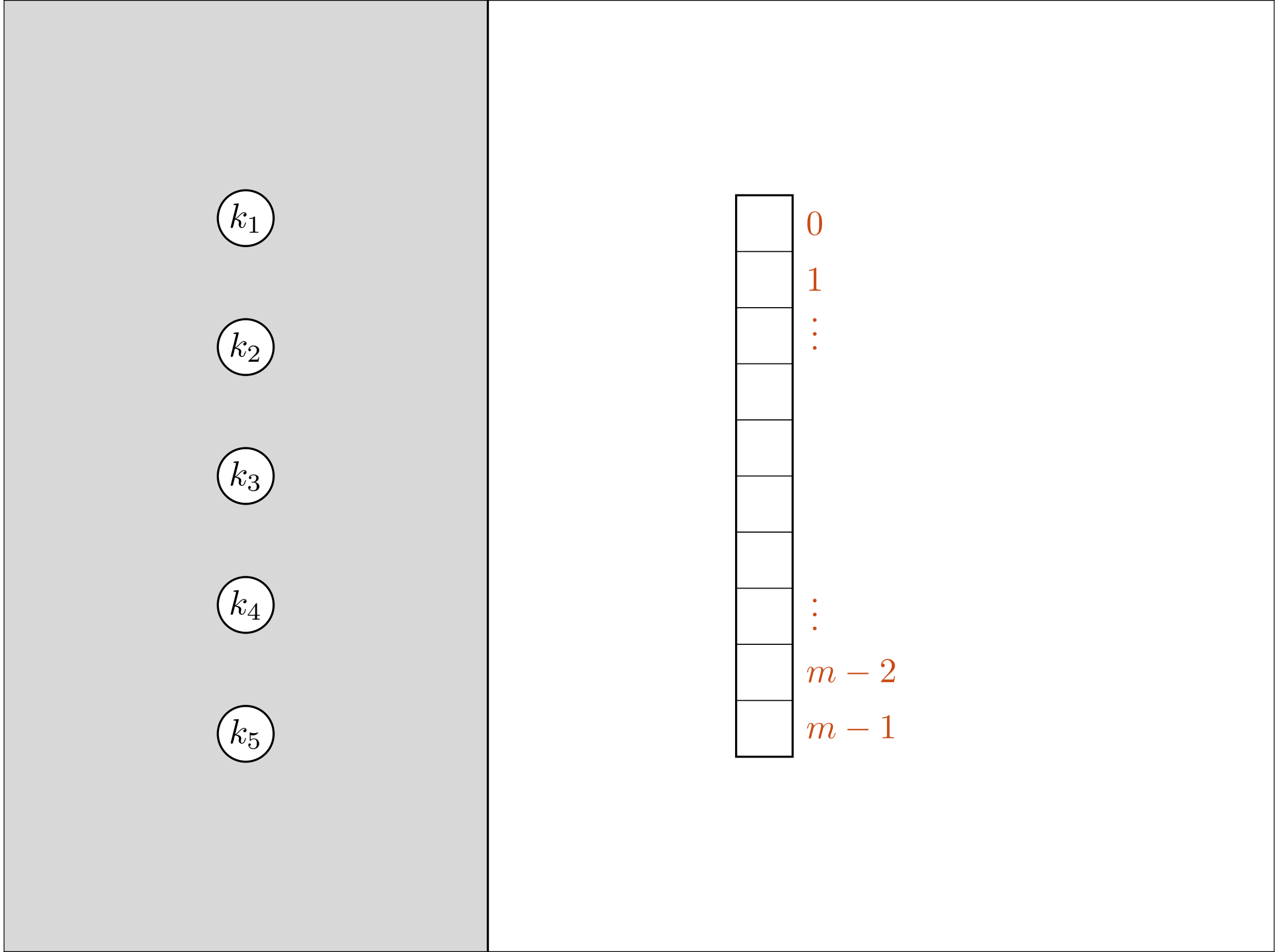

2
3
5
8
0
1
4
6
7
9
0
1
2
3
4
5
6
7
8
9

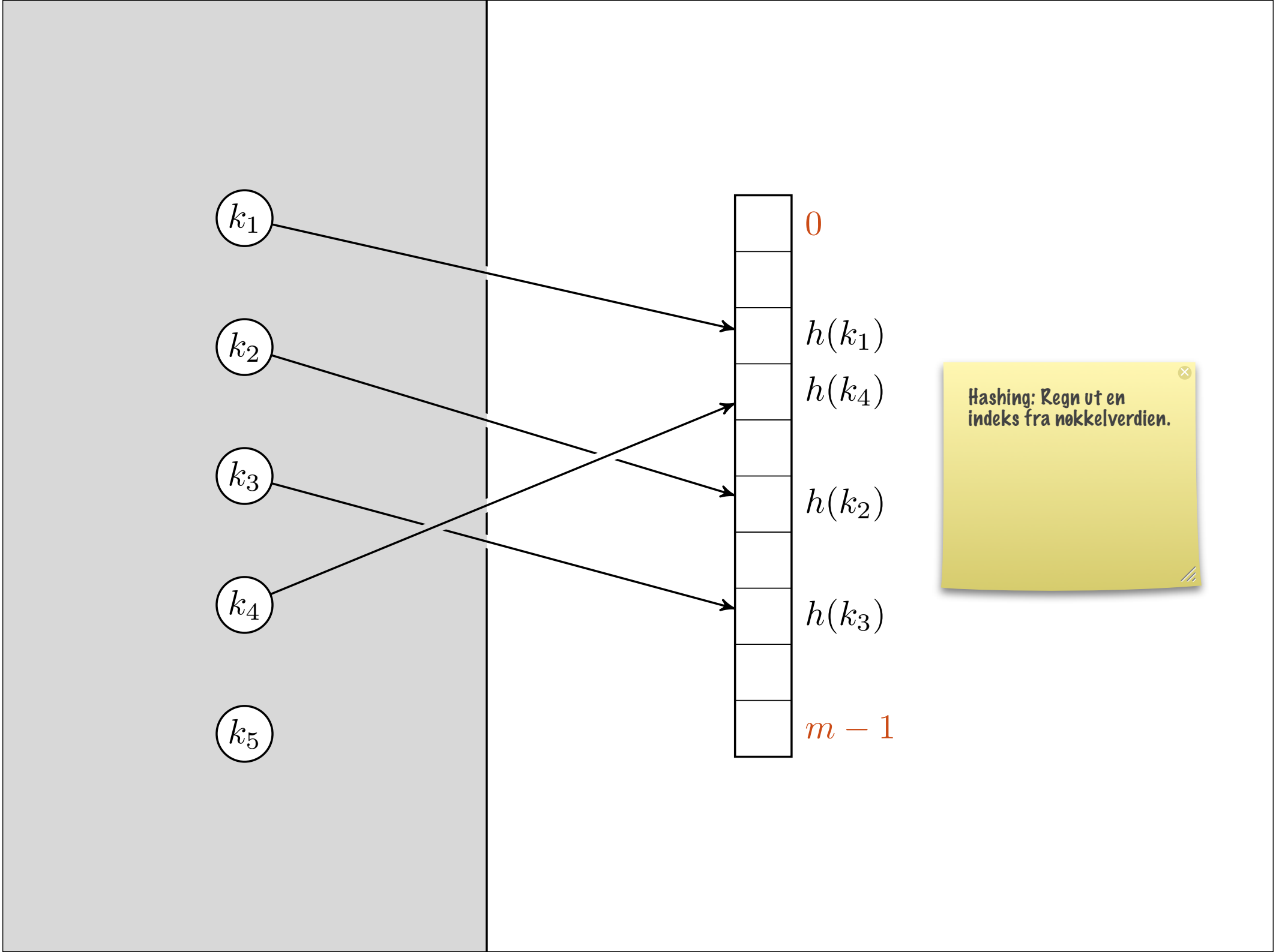

U
K
2
3
5
8
0
1
4
6
7
9
0
1
2
3
4
5
6
7
8
9


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
2
3
5
8
Bruk nøkkelen direkte som indeks. Triviell form for hashing.

Hashtabeller


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


k_1
k_2
k_3
k_4
k_5
0
$h(k_1)$
$h(k_4)$
$h(k_2)$
$h(k_3)$
$m-1$
Hashing: Regn ut en indeks fra nøkkelverdien.

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

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

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

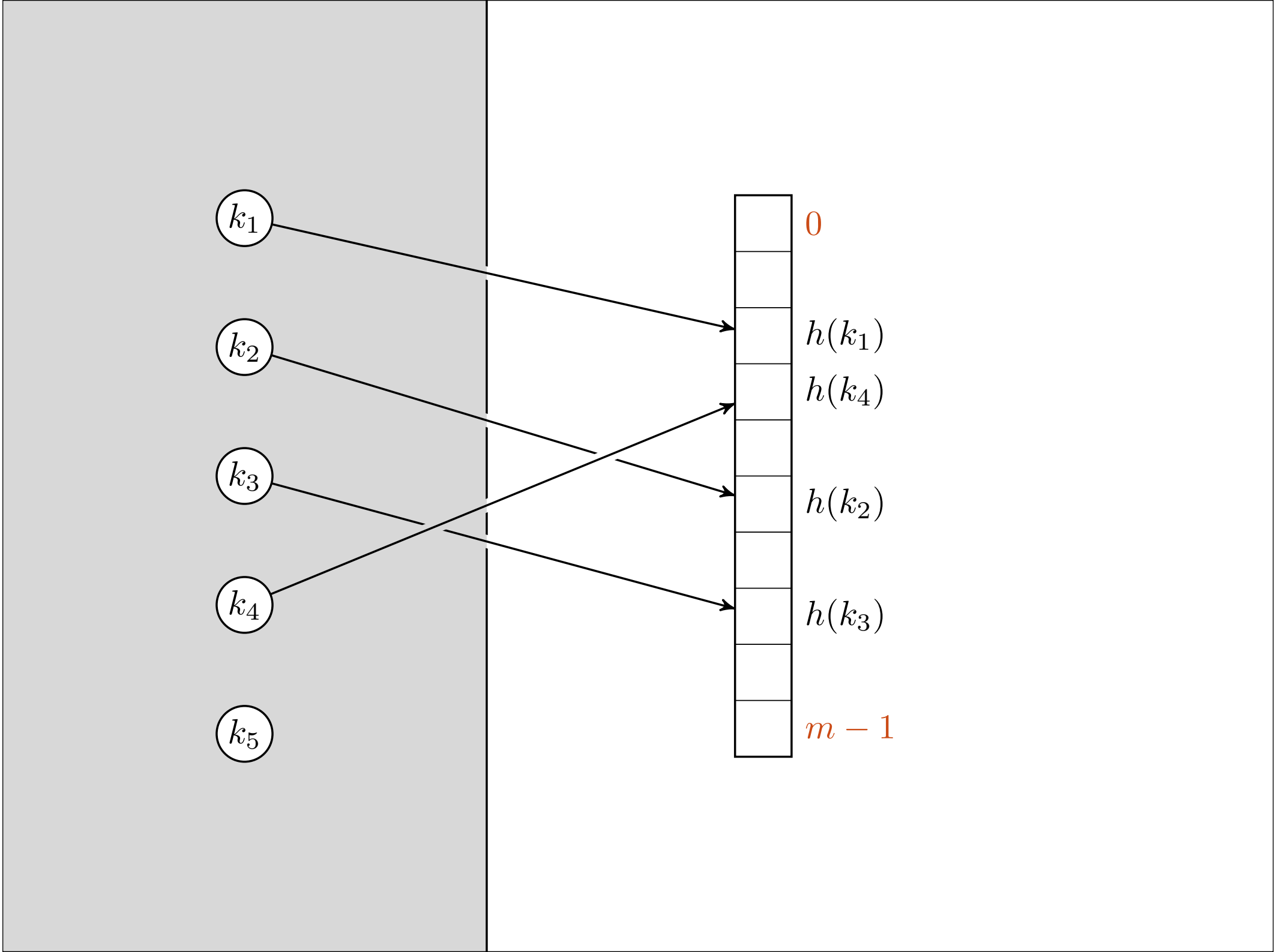
$$h(k) = \lfloor km \rfloor \quad 0 \leq k < 1$$

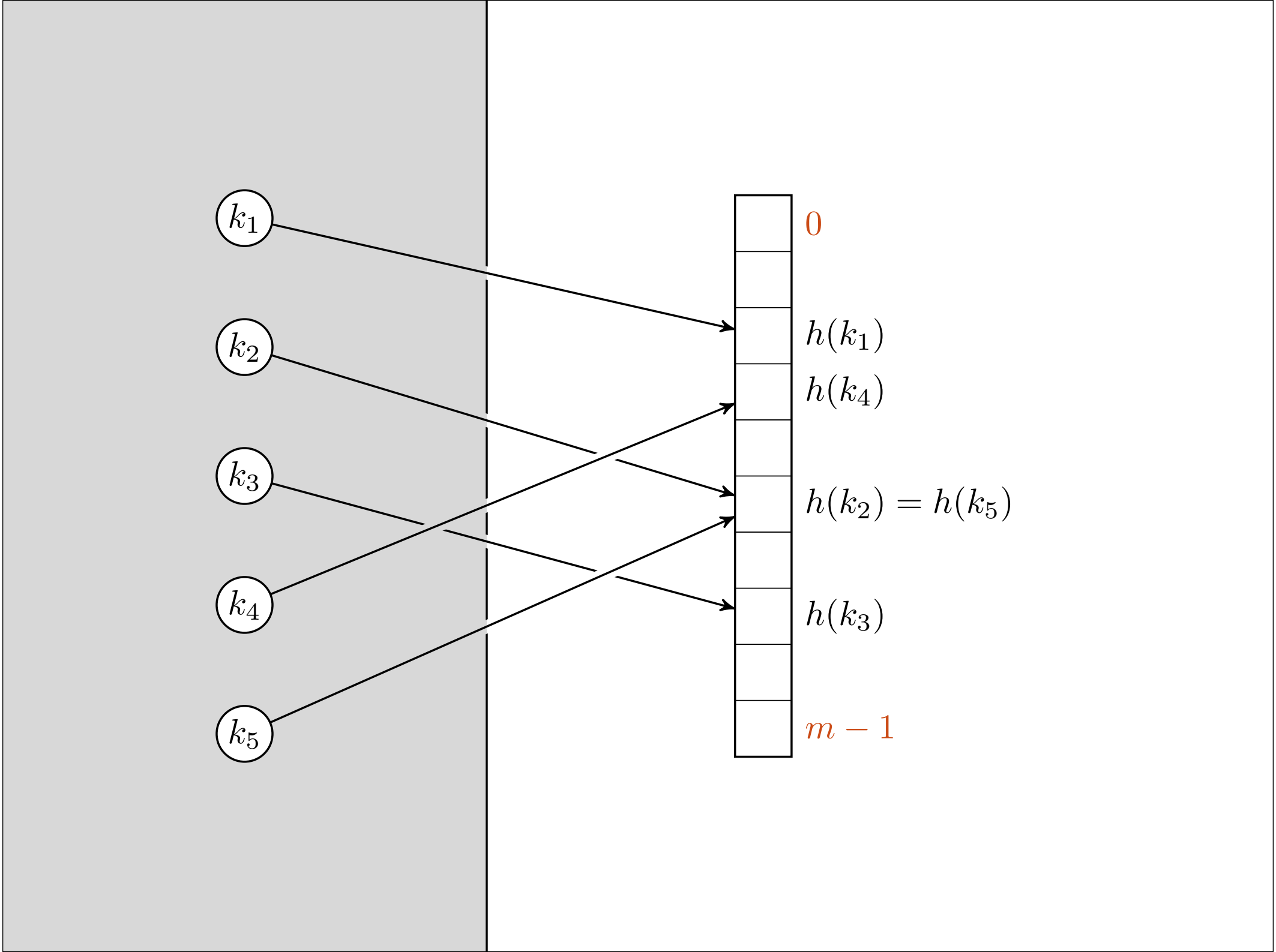
$$h(k) = k \bmod m$$

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

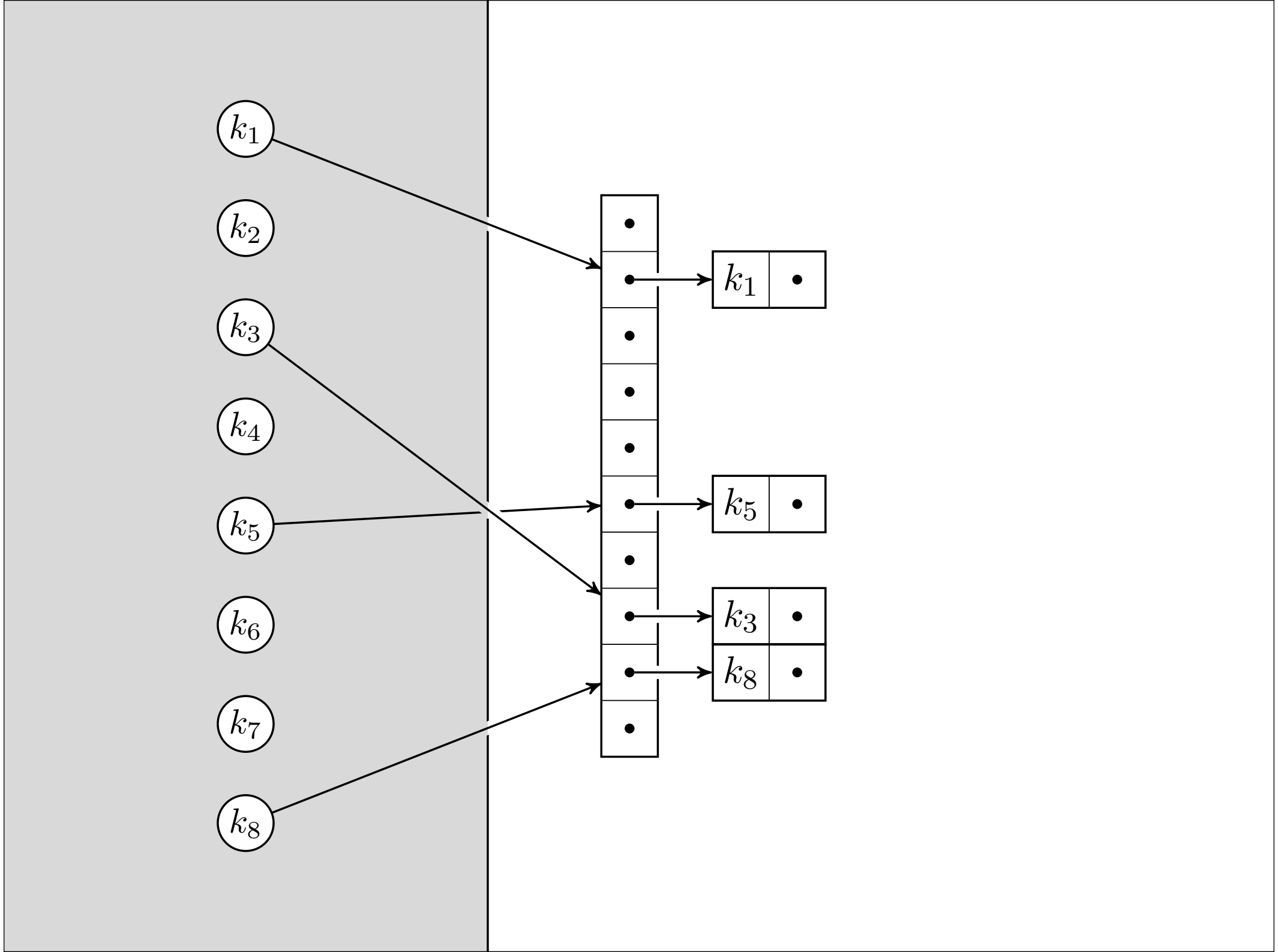
$$0 < A < 1$$

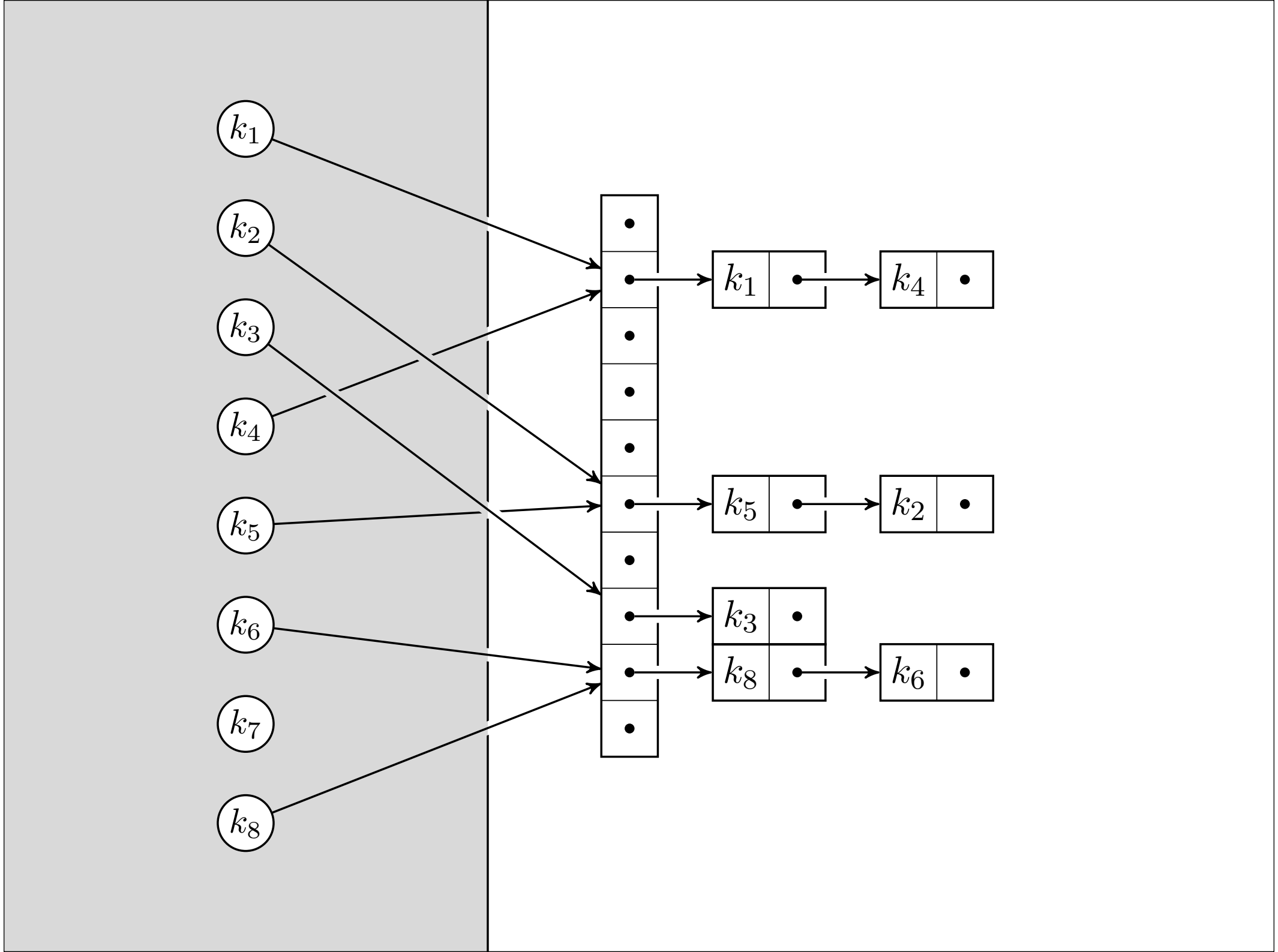
f.eks.

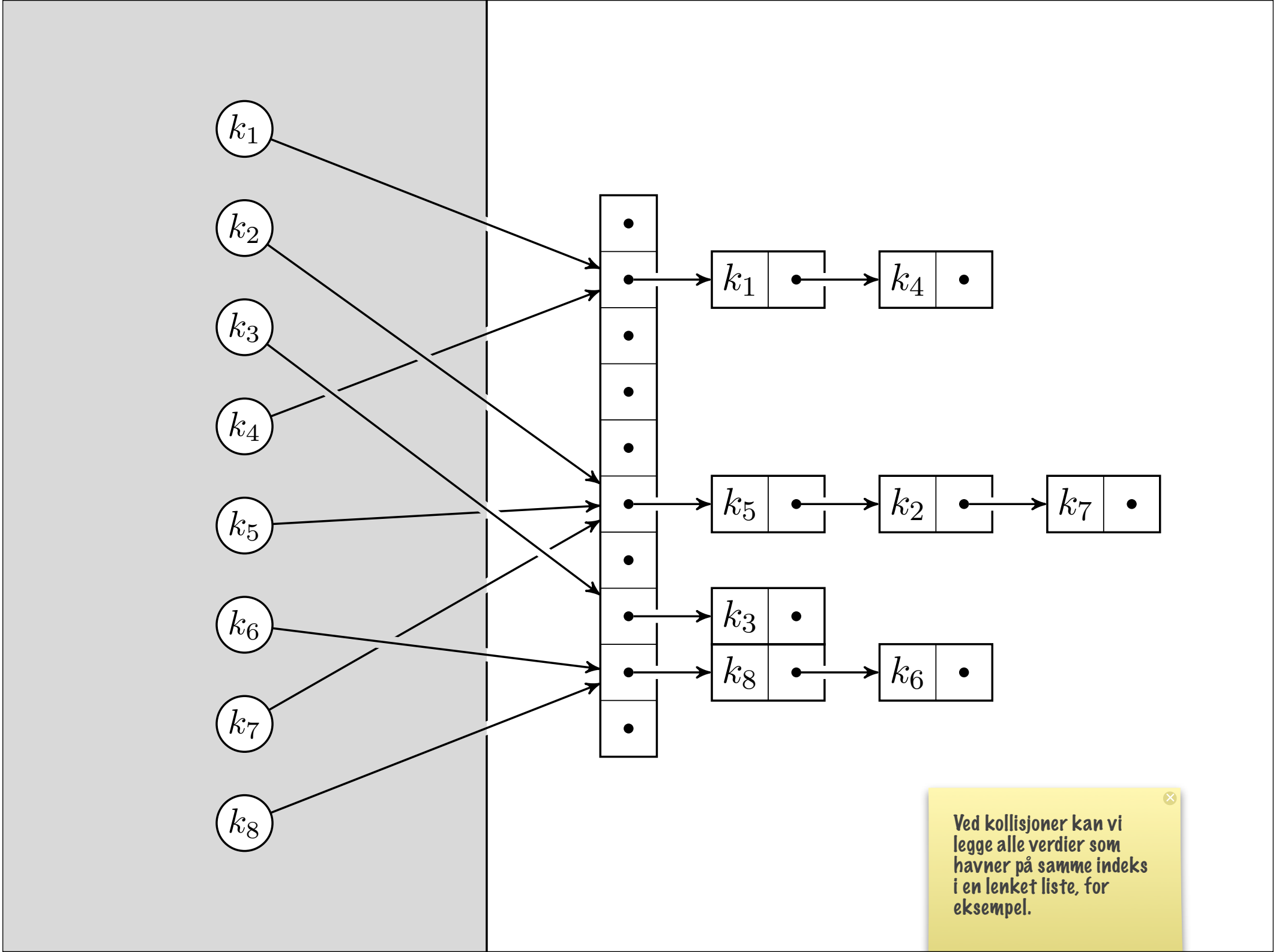

k_1
k_2
k_3
k_4
k_5
0
$h(k_1)$
$h(k_4)$
$h(k_2)$
$h(k_3)$
$m-1$


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)


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
Ved kollisjoner kan vi legge alle verdier som havner på samme indeks i en lenket liste, for eksempel.

Statisk datasett?

Lag custom hashfunksjon!

Kan da garantere konstant kjøretid

Pekere og objekter

Alt. 1: Flere tabeller

- Peker til objekt = adresse
- La pekere være indekser
- Ha én tabell per attributt
- objekt.attributt blir da attributt[objekt]

Alt. 2: Én tabell

- Pekere fortsatt indekser
- Objekt = sammenhengende intervall
- Attributter = avstand fra peker
- objekt.attributt blir
  minne[objekt + attributt]

Alt. 3: Objekt = hashtabell

- Ikke diskutert i boka
- Mindre effektivt, men mer fleksibelt
- Attributter angis ved hashnøkler
- Brukes f.eks. i Python
- objekt.attributt blir objekt[attributt]