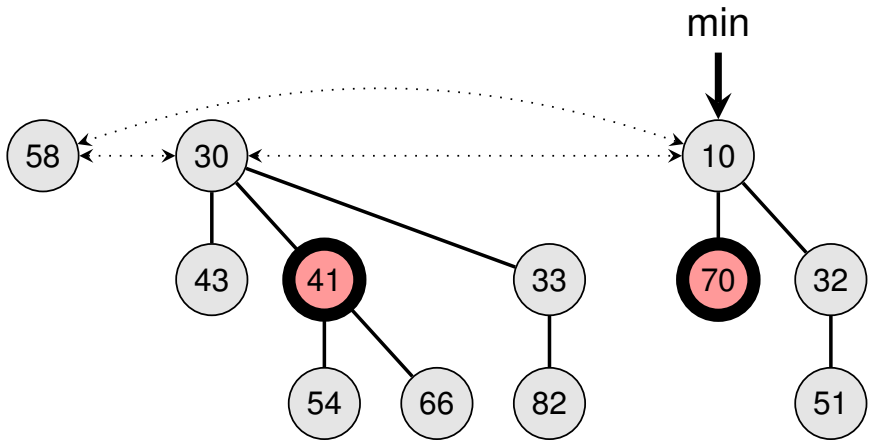


5.2 Fibonacci Heaps

Frank Stajano

Thomas Sauerwald

Lent 2016

UNIVERSITY OF CAMBRIDGE

Priority Queues Overview

Operation	Linked list	Binary heap	Binomial heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
EXTRACT-MIN	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Priority Queues Overview

Operation	Linked list	Binary heap	Binomial heap	Fibon. heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap	Fibonacci heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap **actual cost**	Fibonacci heap **amortized cost**
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap **actual cost**	Fibonacci heap **amortized cost**
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

n is the number of items in the heap when the operation is performed.

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap **actual cost**	Fibonacci heap **amortized cost**
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

n is the number of items in the heap when the operation is performed.

Binomial Heap: $k/2$ DECREASE-KEY
+ $k/2$ INSERT

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap **actual cost**	Fibonacci heap **amortized cost**
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

n is the number of items in the heap when the operation is performed.

Binomial Heap: $k/2$ DECREASE-KEY + $k/2$ INSERT

- $c_1 = c_2 = \cdots = c_k = \mathcal{O}(\log n)$

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap **actual cost**	Fibonacci heap **amortized cost**
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

n is the number of items in the heap when the operation is performed.

Binomial Heap: $k/2$ DECREASE-KEY
+ $k/2$ INSERT

- $c_1 = c_2 = \cdots = c_k = \mathcal{O}(\log n)$

$\Rightarrow \sum_{i=1}^{k} c_i = \mathcal{O}(k \log n)$

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap **actual cost**	Fibonacci heap **amortized cost**
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

n is the number of items in the heap when the operation is performed.

Binomial Heap: $k/2$ DECREASE-KEY + $k/2$ INSERT

Fibonacci Heap: $k/2$ DECREASE-KEY + $k/2$ INSERT

- $c_1 = c_2 = \cdots = c_k = \mathcal{O}(\log n)$

$\Rightarrow \sum_{i=1}^{k} c_i = \mathcal{O}(k \log n)$

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap **actual cost**	Fibonacci heap **amortized cost**
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

n is the number of items in the heap when the operation is performed.

Binomial Heap: $k/2$ DECREASE-KEY + $k/2$ INSERT

- $c_1 = c_2 = \cdots = c_k = \mathcal{O}(\log n)$

$\Rightarrow \sum_{i=1}^{k} c_i = \mathcal{O}(k \log n)$

Fibonacci Heap: $k/2$ DECREASE-KEY + $k/2$ INSERT

- $\widehat{c}_1 = \widehat{c}_2 = \cdots = \widehat{c}_k = \mathcal{O}(1)$

Binomial Heap vs. Fibonacci Heap: Costs

Operation	Binomial heap **actual cost**	Fibonacci heap **amortized cost**
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
MERGE	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

n is the number of items in the heap when the operation is performed.

Binomial Heap: $k/2$ DECREASE-KEY + $k/2$ INSERT

- $c_1 = c_2 = \cdots = c_k = \mathcal{O}(\log n)$

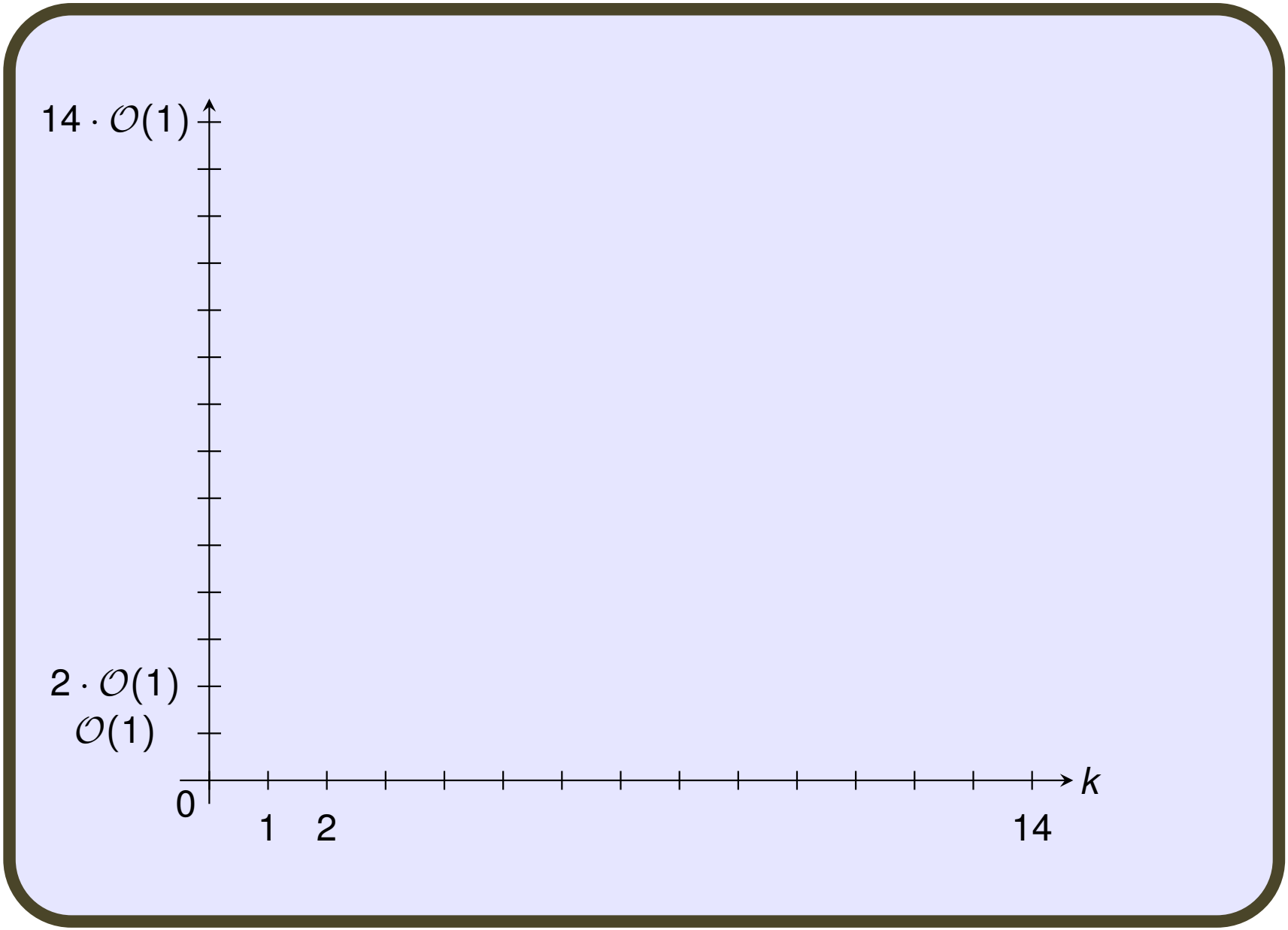
$\Rightarrow \sum_{i=1}^{k} c_i = \mathcal{O}(k \log n)$

Fibonacci Heap: $k/2$ DECREASE-KEY + $k/2$ INSERT

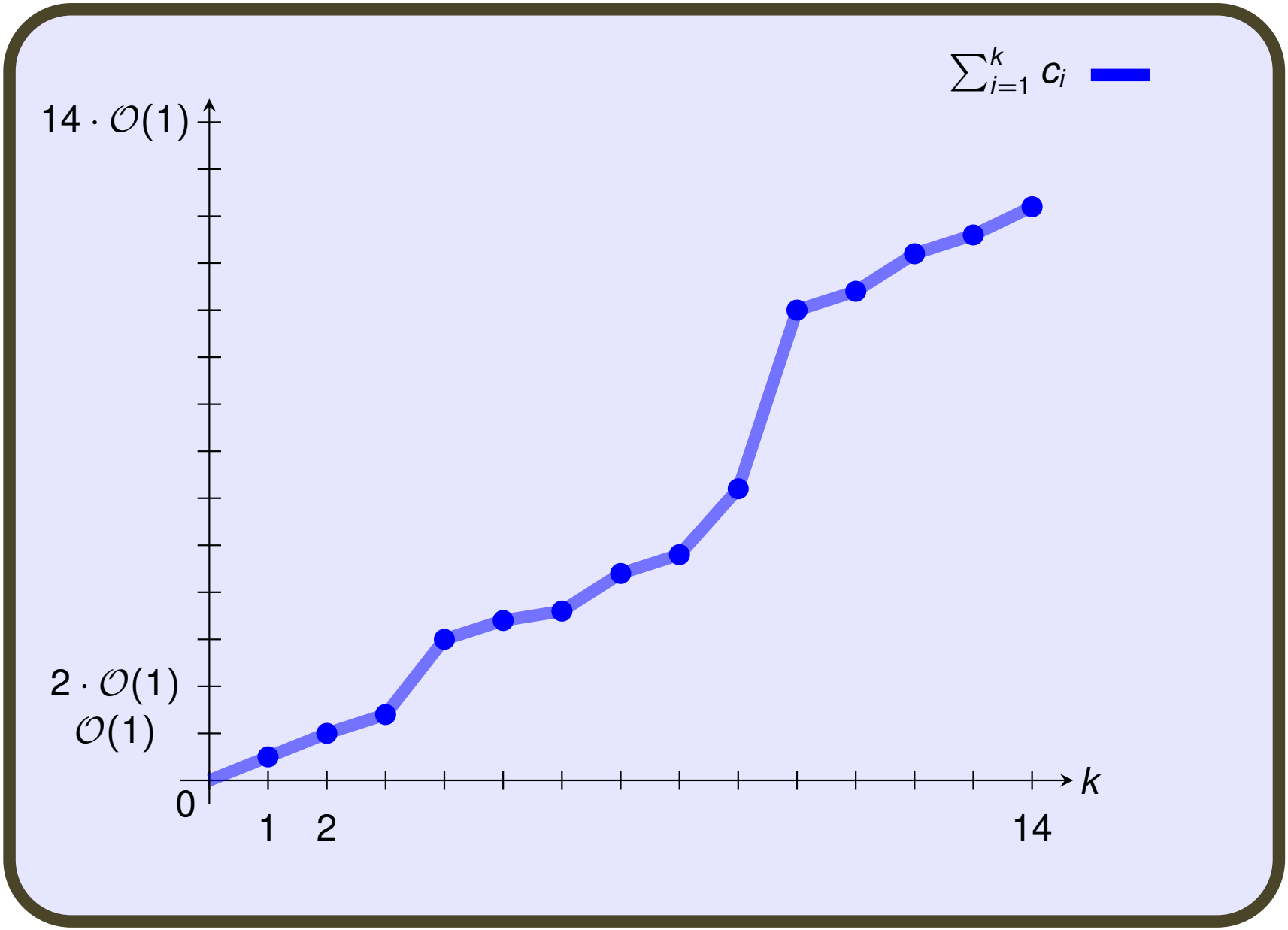
- $\widehat{c}_1 = \widehat{c}_2 = \cdots = \widehat{c}_k = \mathcal{O}(1)$

$\Rightarrow \sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} \widehat{c}_i = \mathcal{O}(k)$

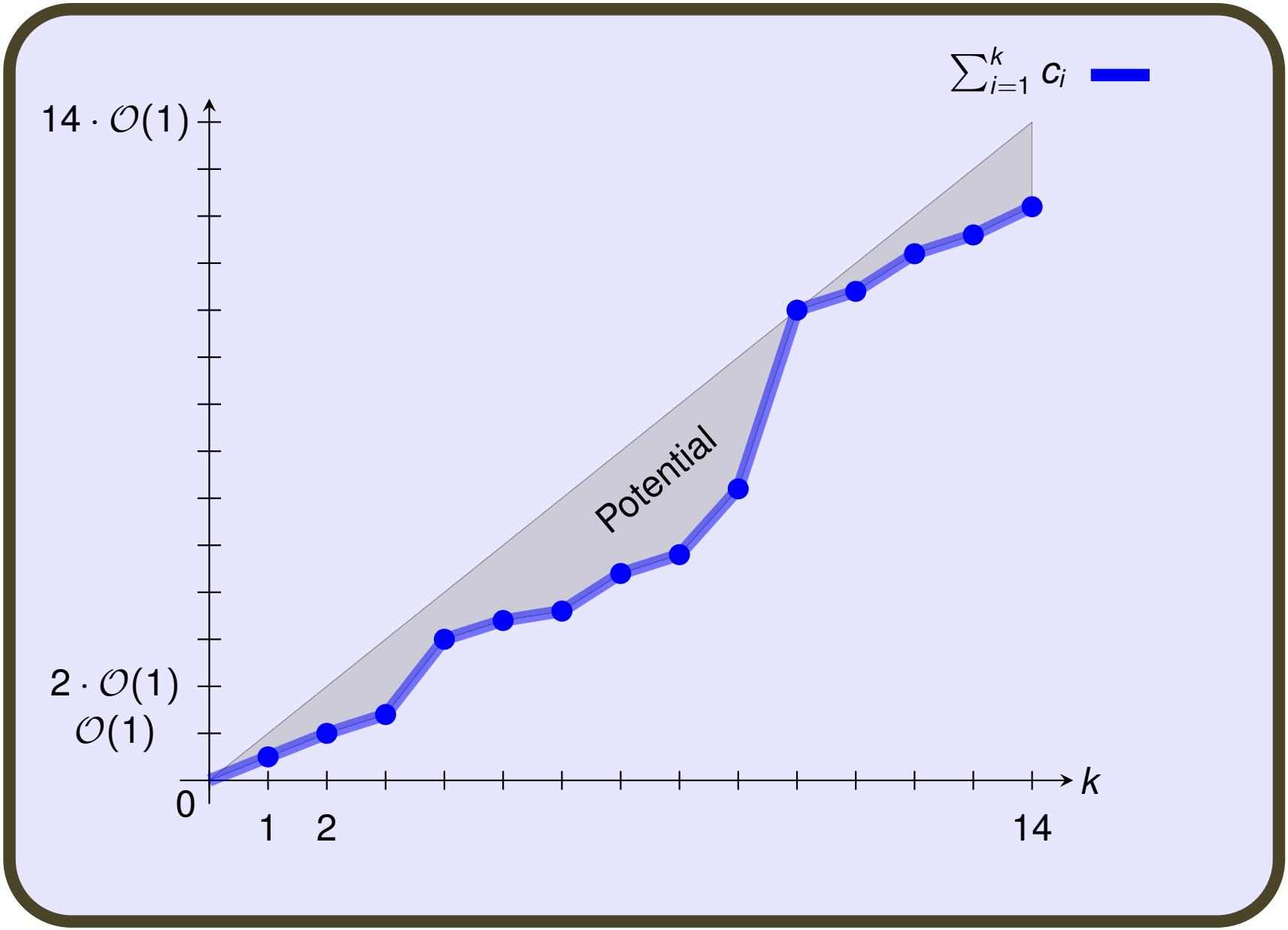
Actual vs. Amortized Cost



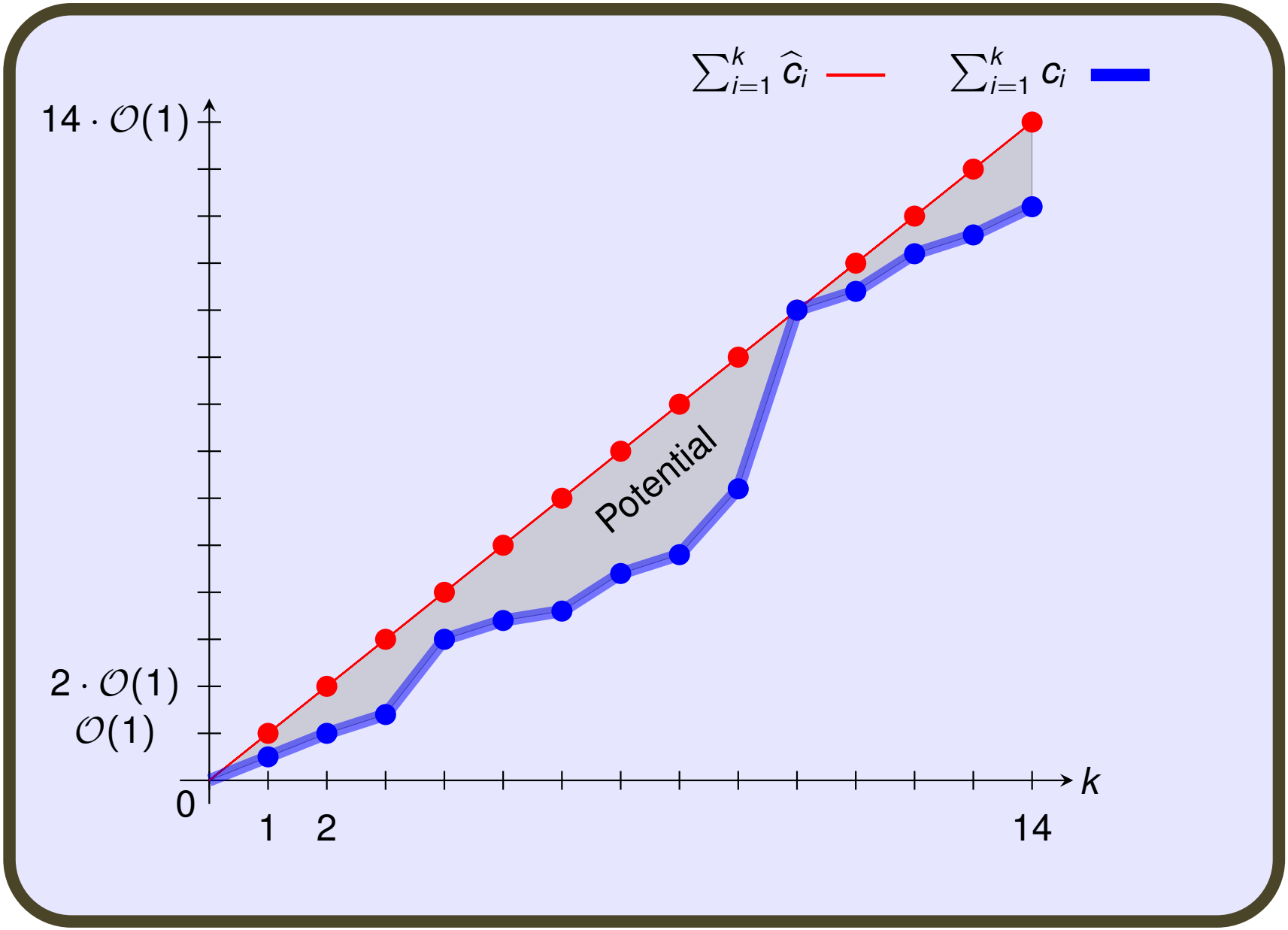
Actual vs. Amortized Cost



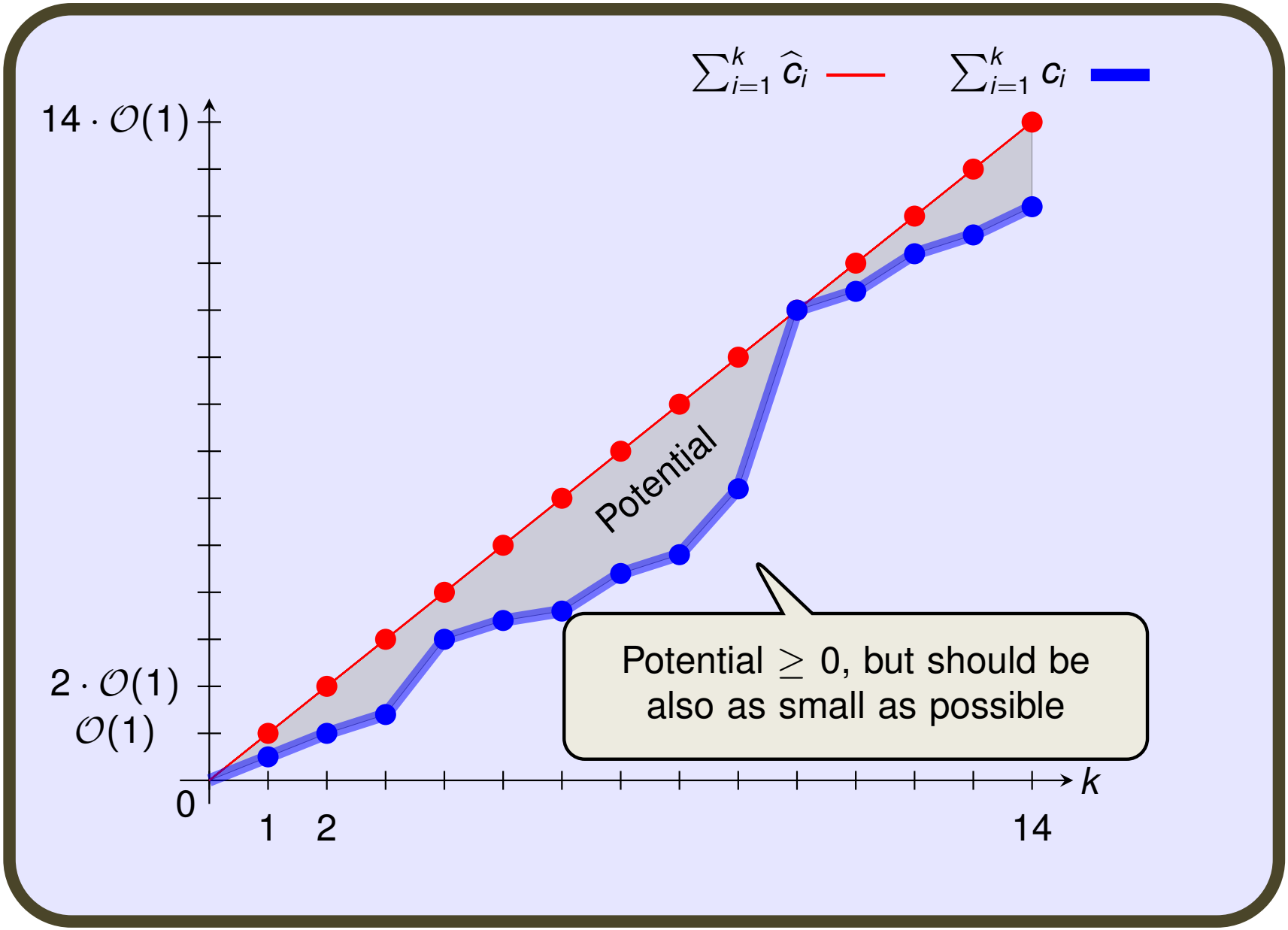
Actual vs. Amortized Cost



Actual vs. Amortized Cost



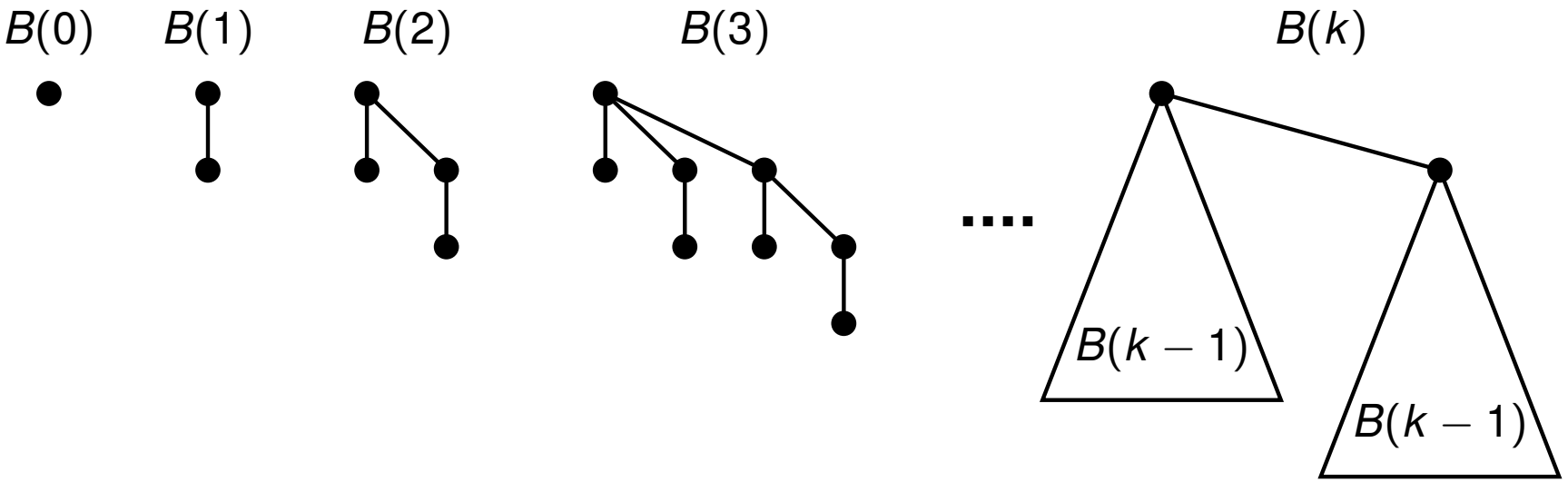
Actual vs. Amortized Cost



Outline

Reminder: Binomial Heaps

Binomial Trees

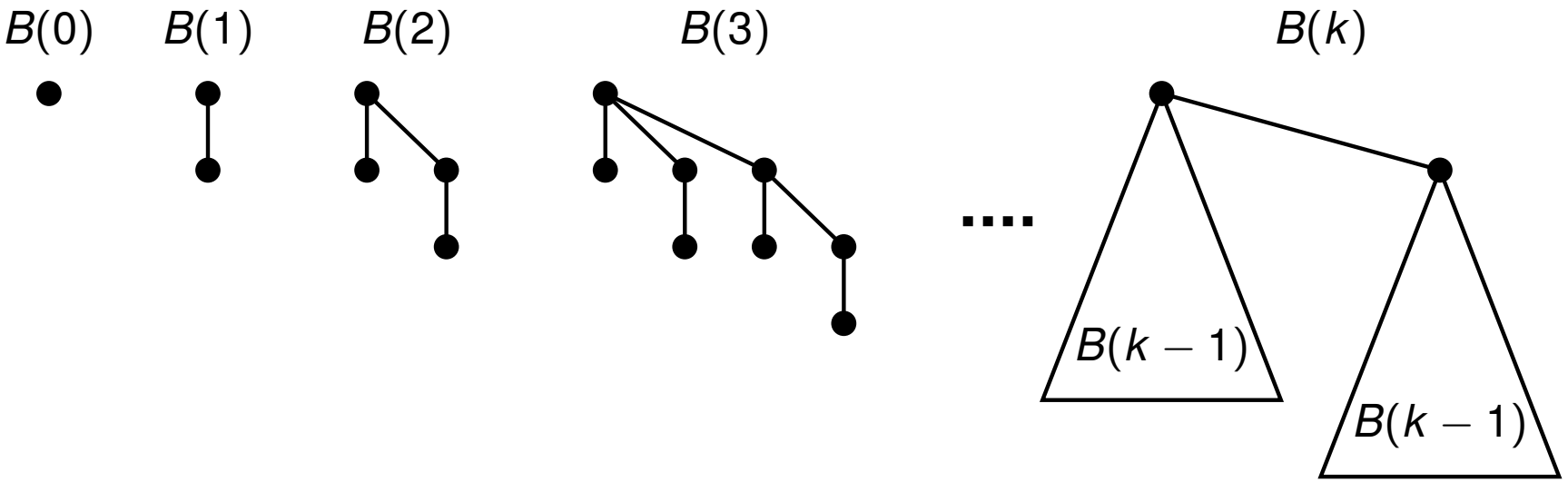


Binomial Heaps

- Binomial Heap is a collection of binomial trees of different orders, each of which obeys the heap property

Reminder: Binomial Heaps

Binomial Trees

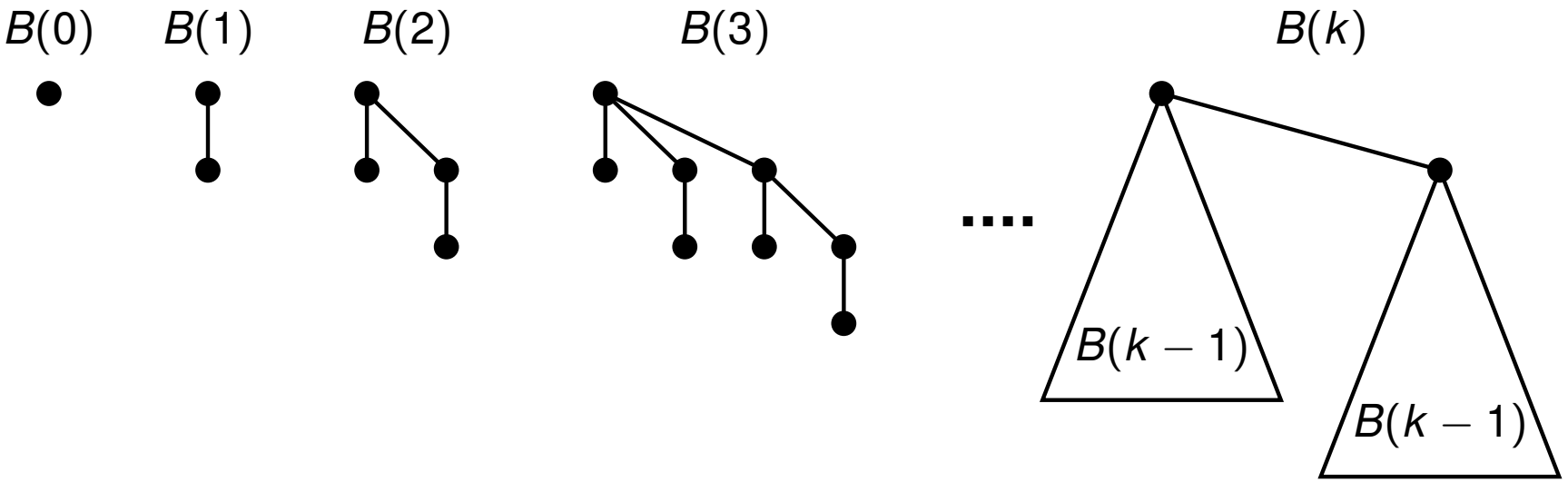


Binomial Heaps

- Binomial Heap is a collection of binomial trees of different orders, each of which obeys the heap property
- Operations:

Reminder: Binomial Heaps

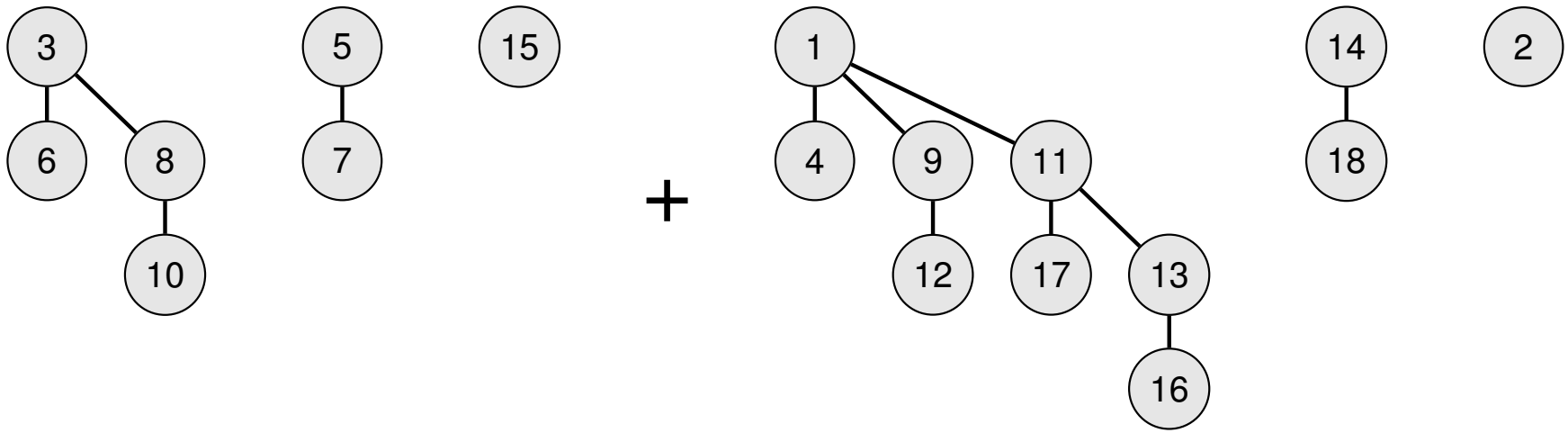
Binomial Trees



Binomial Heaps

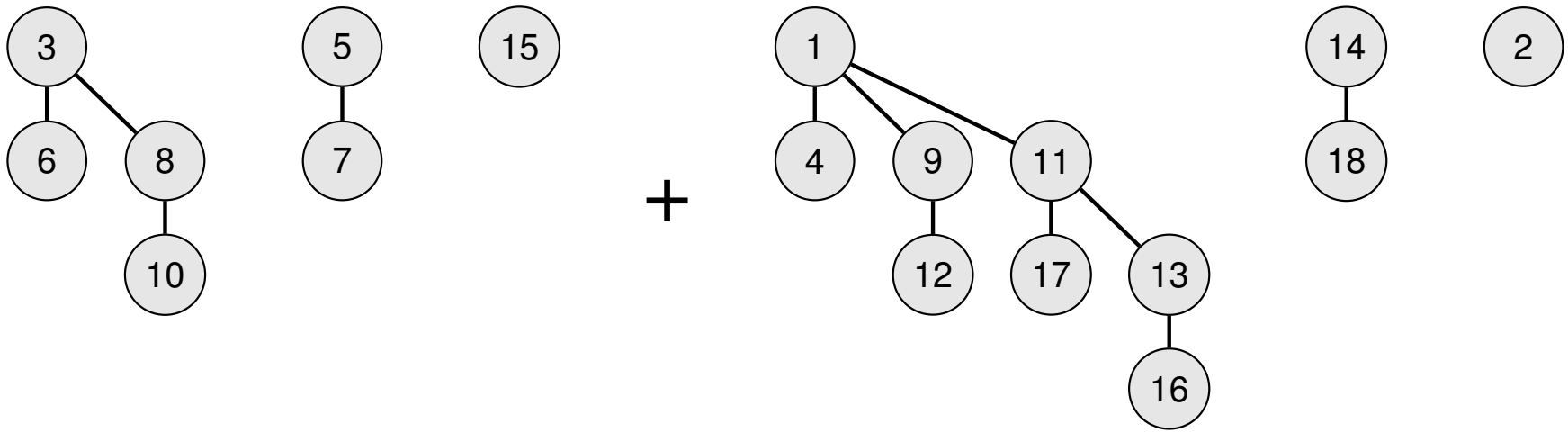
- Binomial Heap is a collection of binomial trees of different orders, each of which obeys the heap property
- Operations:
  - MERGE: Merge two binomial heaps using Binary Addition Procedure
  - INSERT: Add $B(0)$ and perform a MERGE
  - EXTRACT-MIN: Find tree with minimum key, cut it and perform a MERGE
  - DECREASE-KEY: The same as in a binary heap

Merging two Binomial Heaps



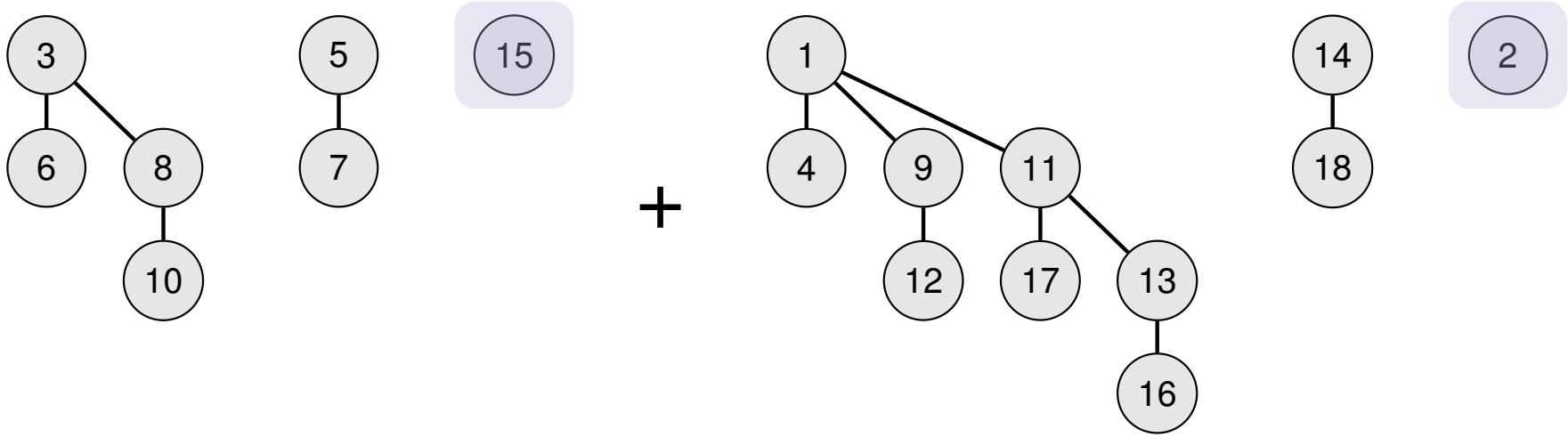
$$
\begin{array}{cccccc}
0 & 0 & 1 & 1 & 1 & = 7 \\
0 & 1 & 0 & 1 & 1 & = 11 \\
 & 1 & 1 & 1 & 1 & \\
\hline
1 & 0 & 0 & 1 & 0 & = 18
\end{array}
$$

Merging two Binomial Heaps



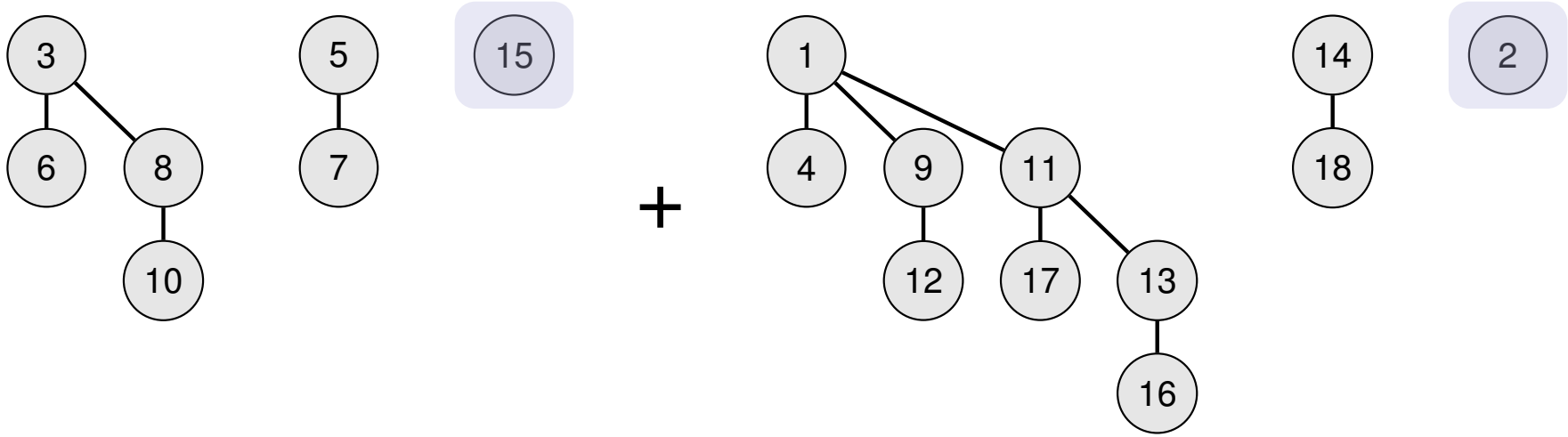
0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

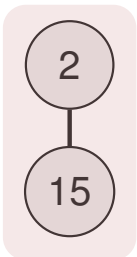
Merging two Binomial Heaps



0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

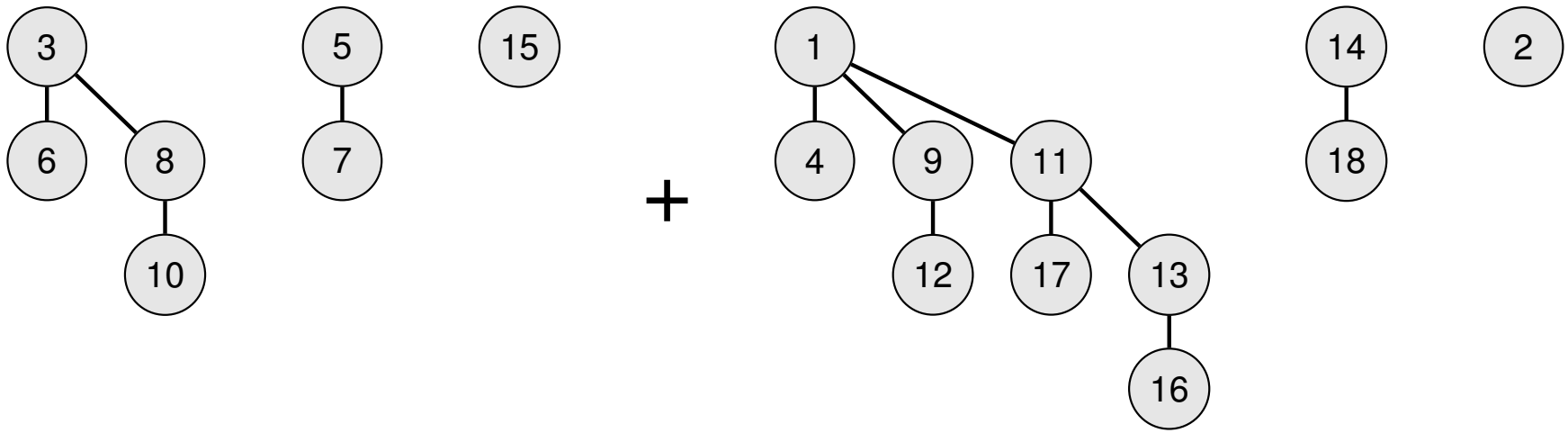
Merging two Binomial Heaps

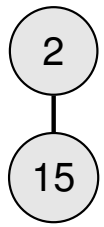




0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

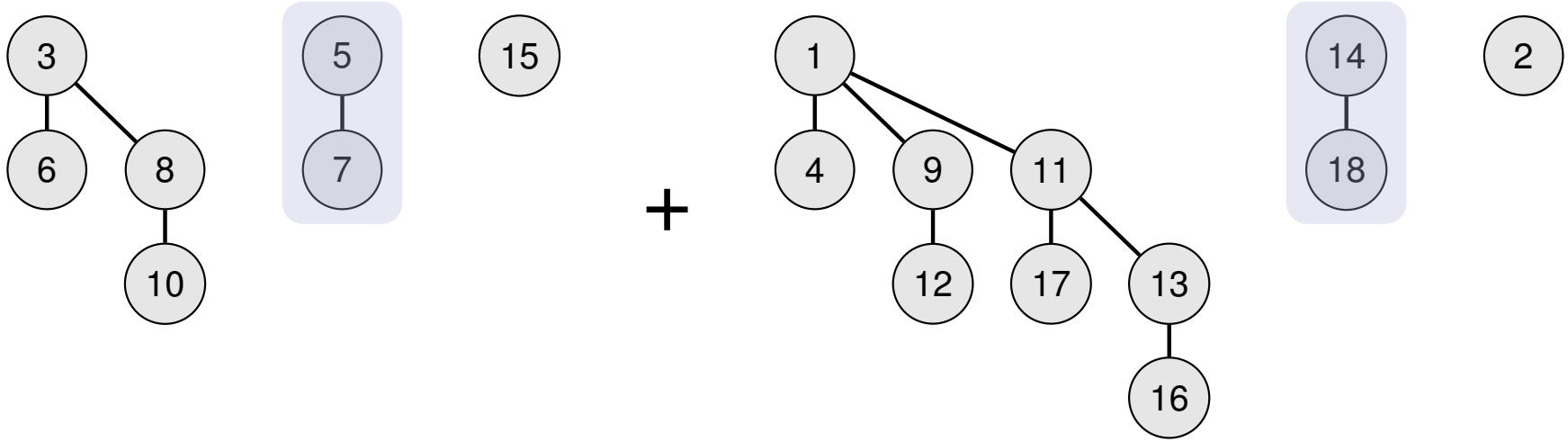
Merging two Binomial Heaps

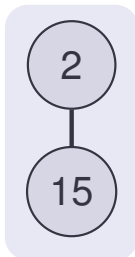




0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

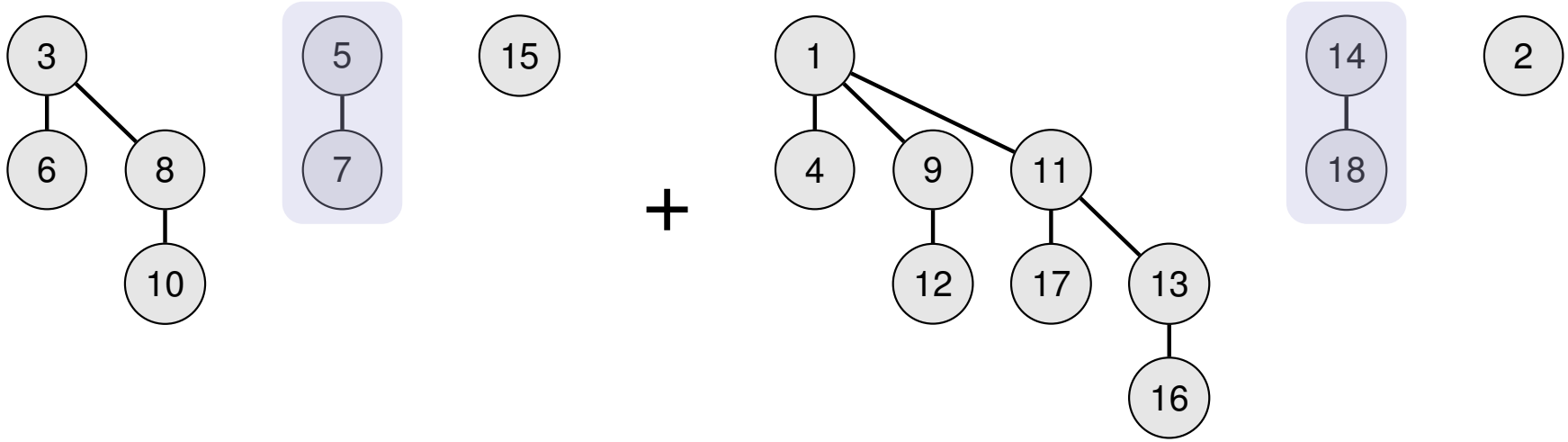
Merging two Binomial Heaps

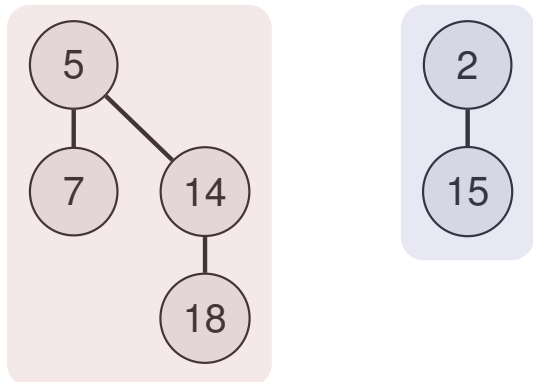




0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

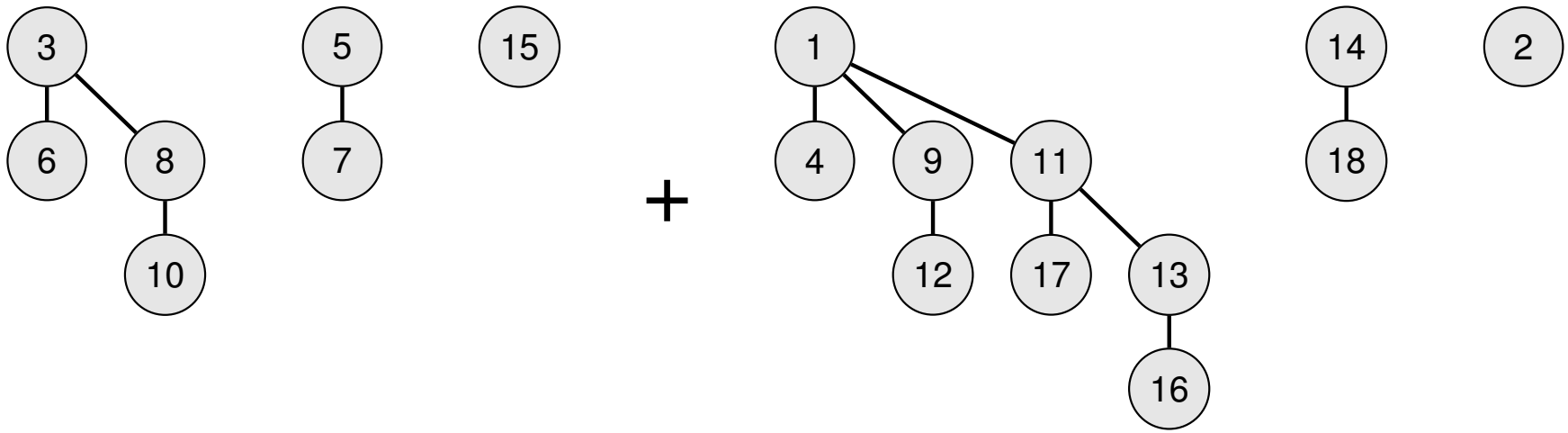
Merging two Binomial Heaps

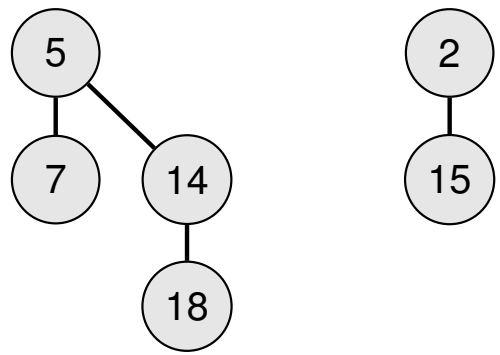




0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

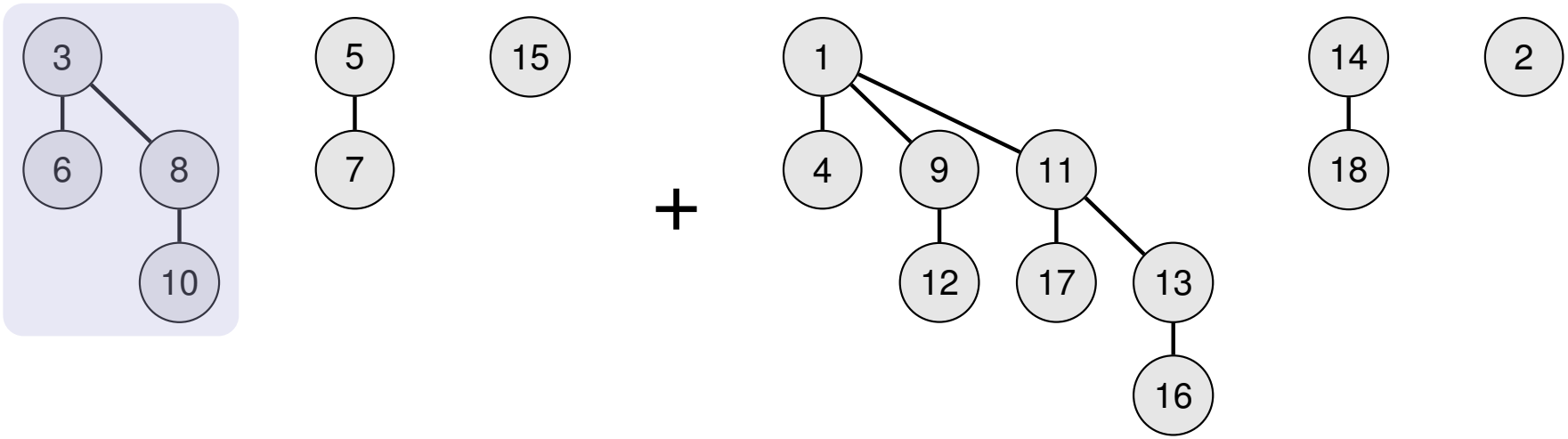
Merging two Binomial Heaps

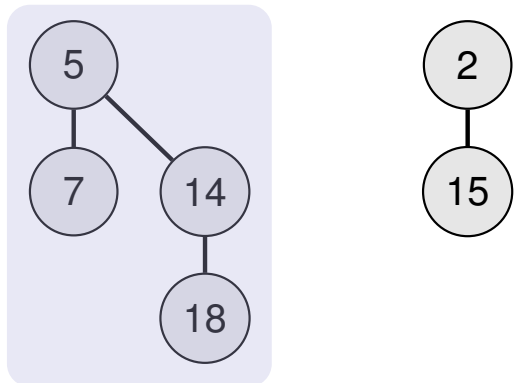




0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

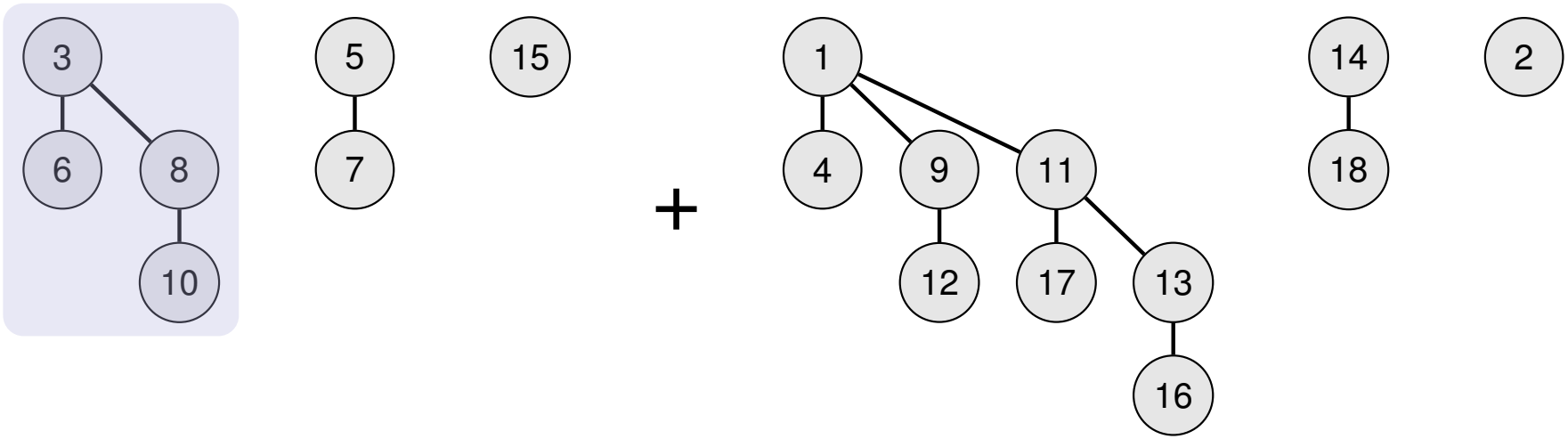
Merging two Binomial Heaps

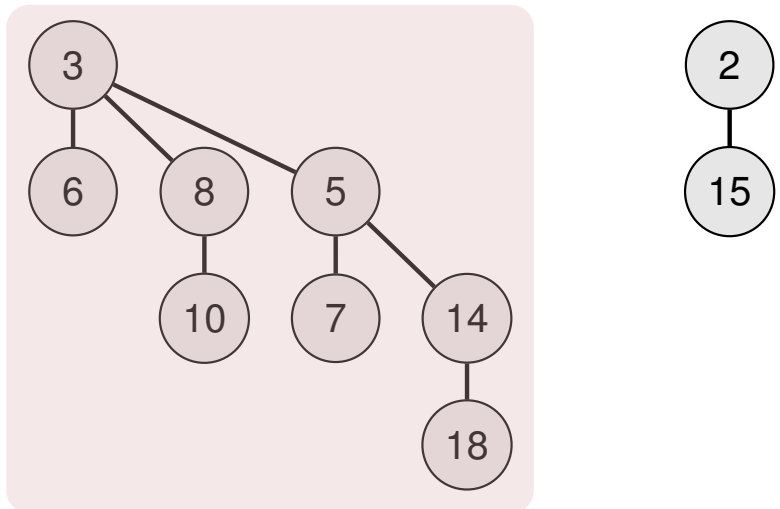




0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

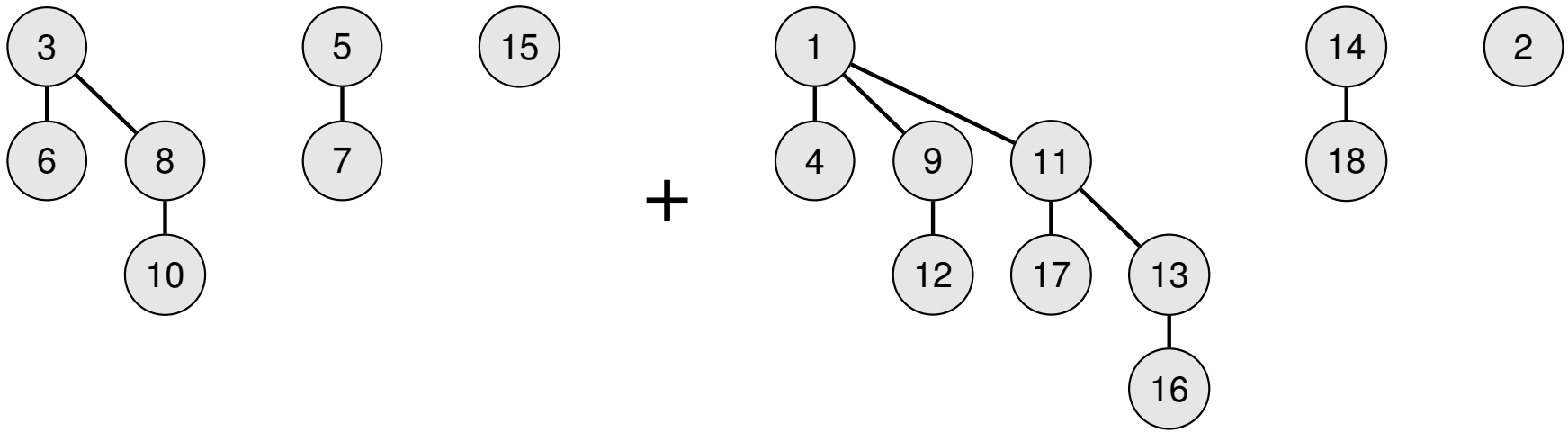
Merging two Binomial Heaps

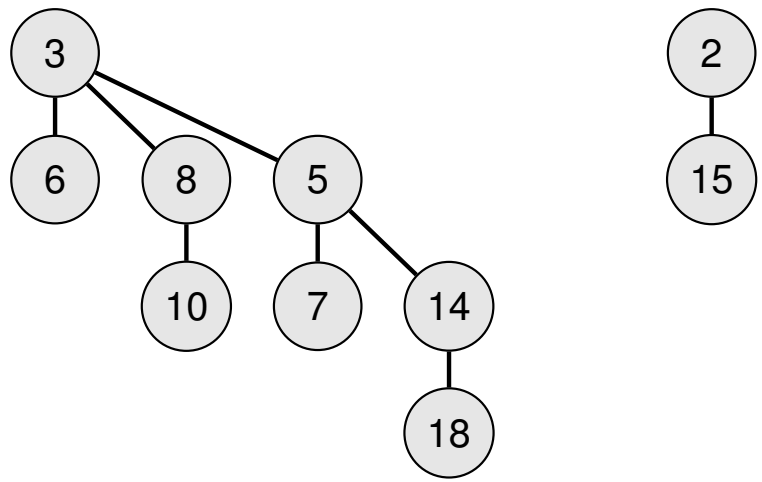




0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

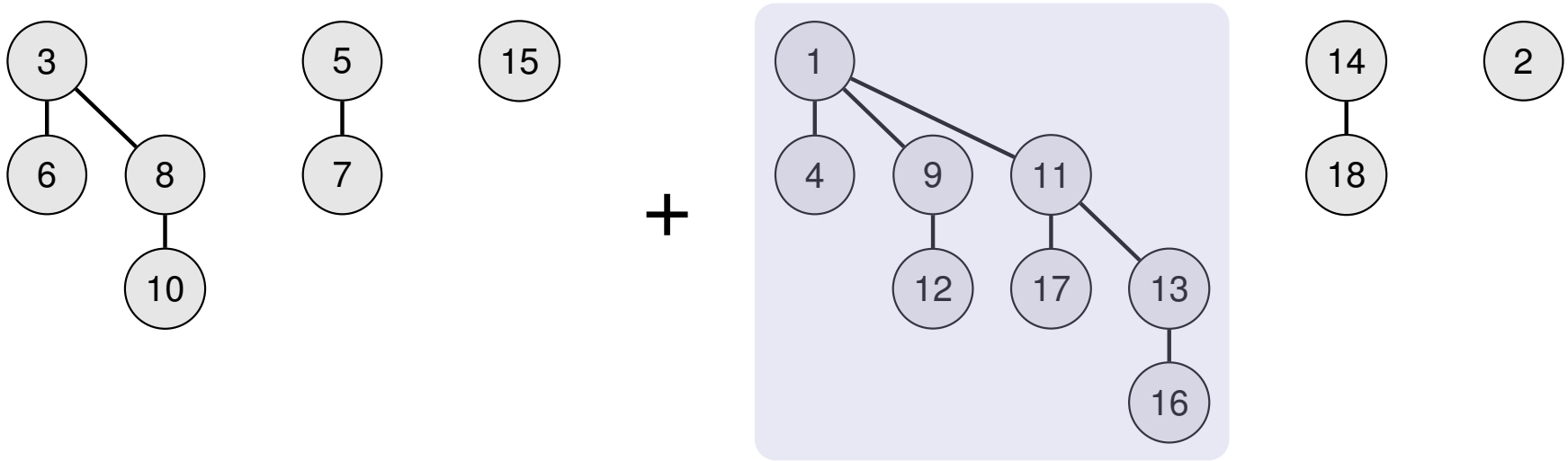
Merging two Binomial Heaps

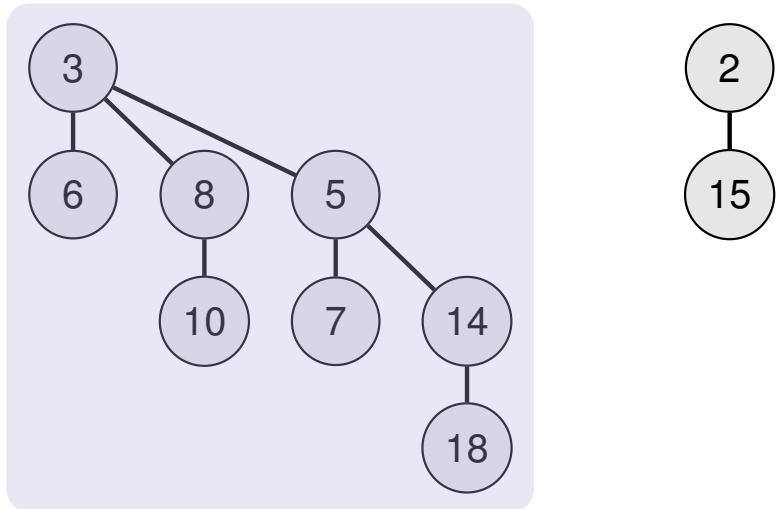




0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

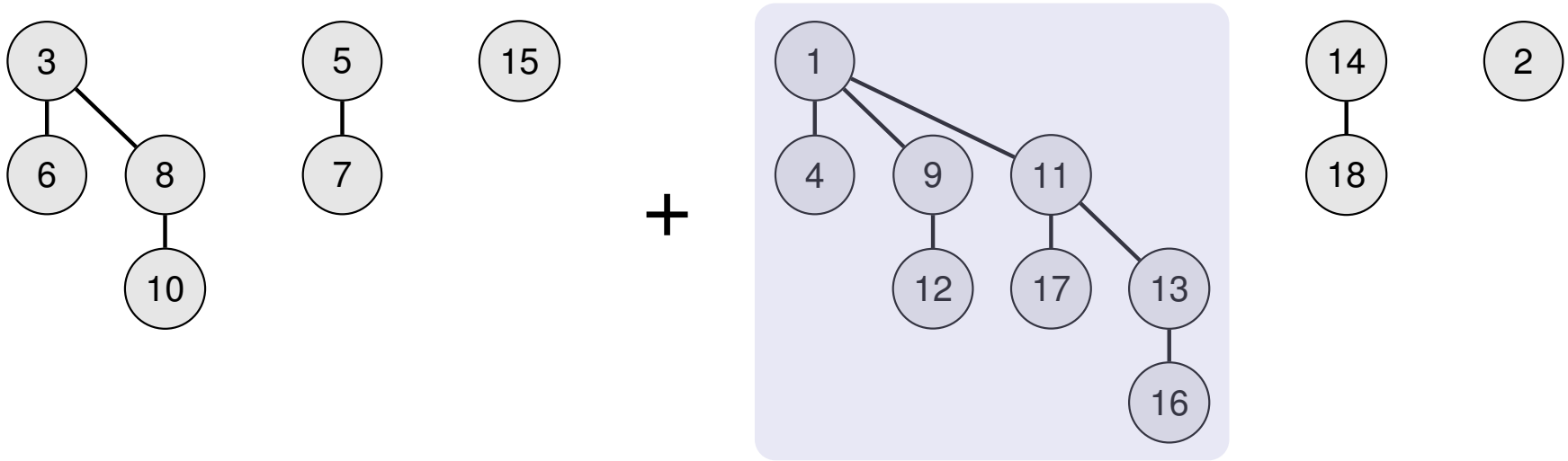
Merging two Binomial Heaps



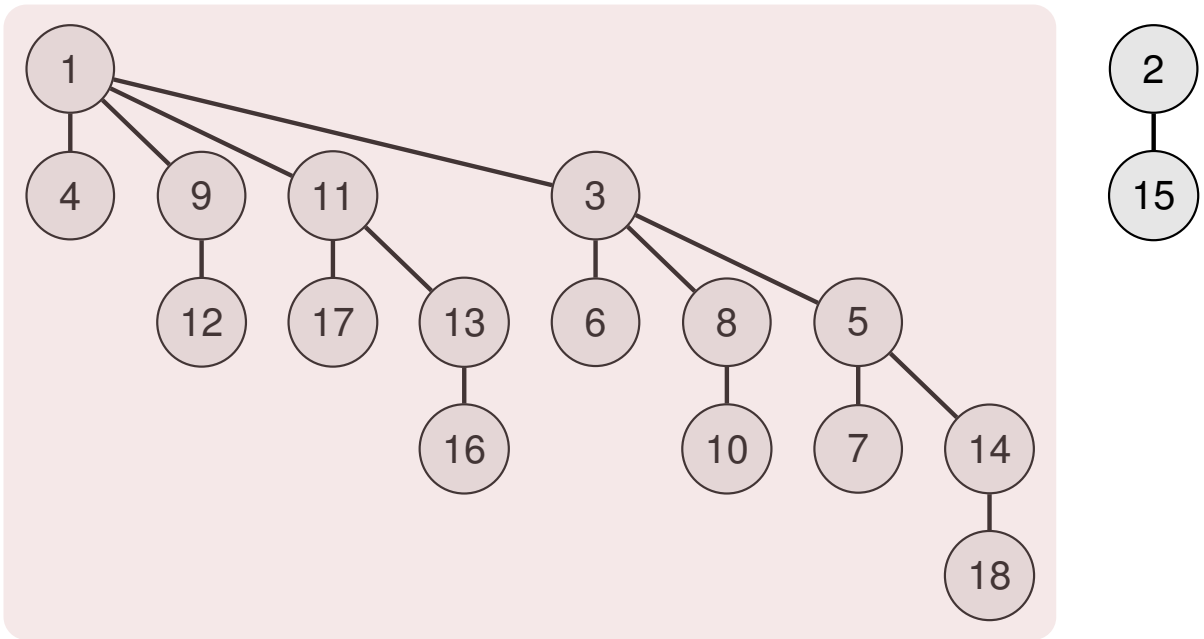


0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

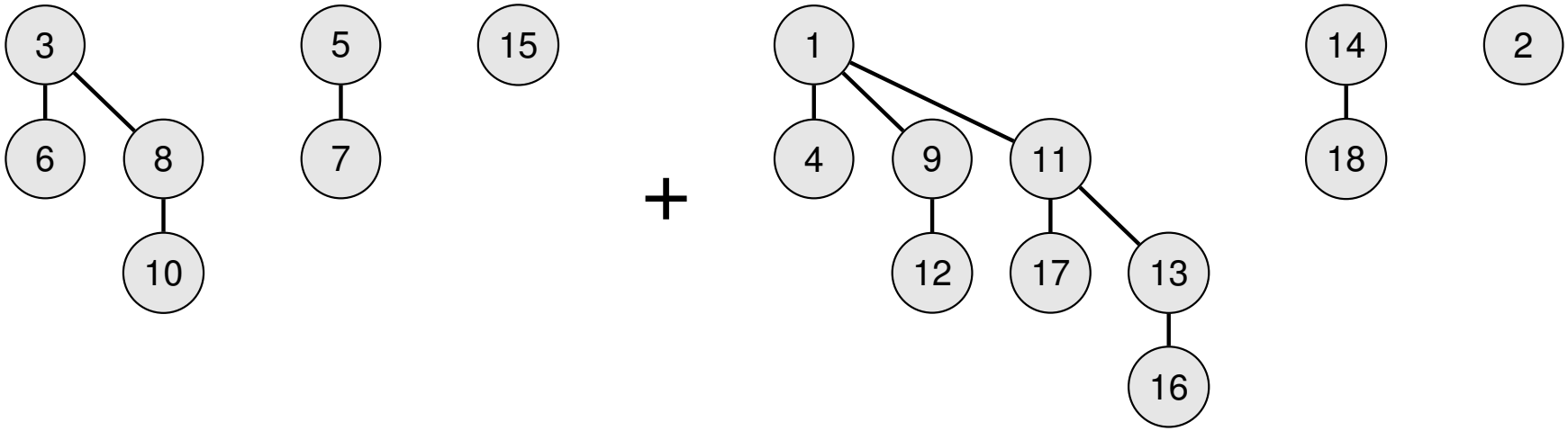
Merging two Binomial Heaps

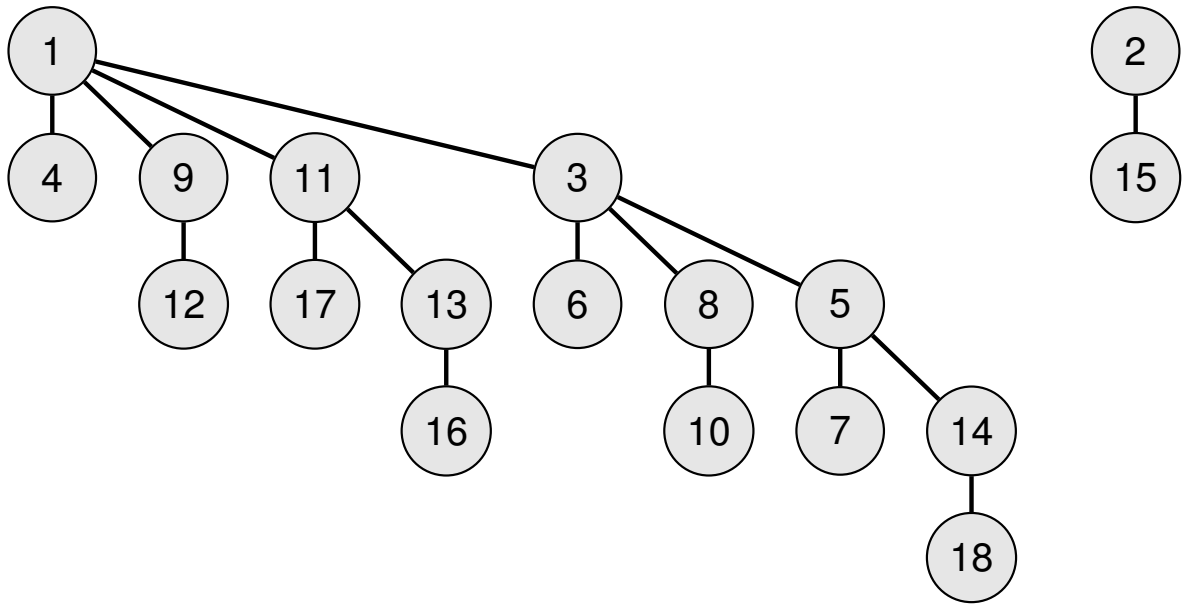


0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18



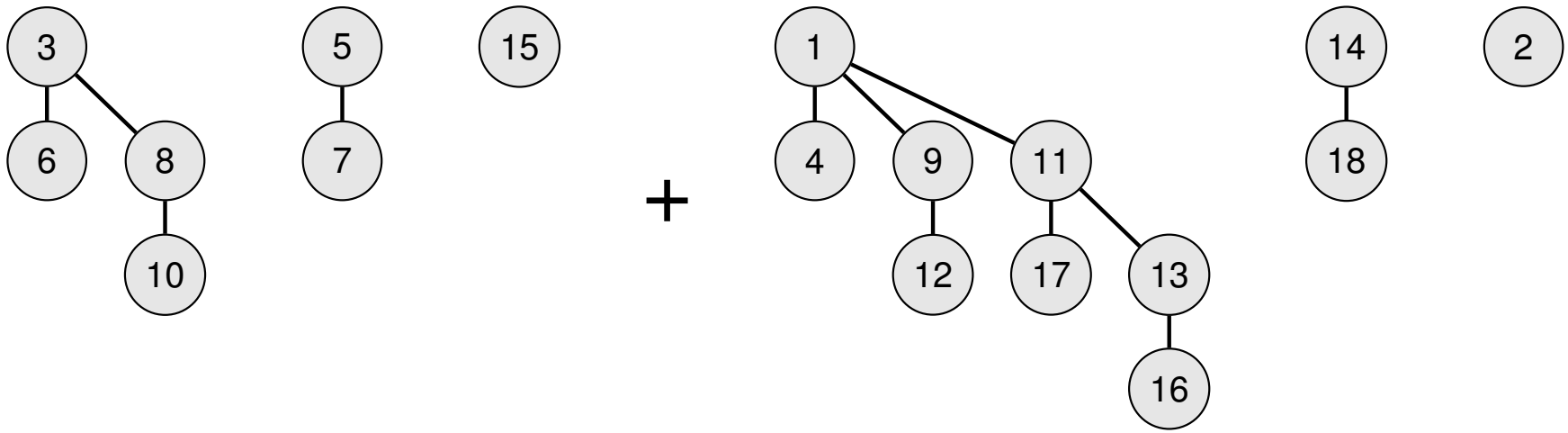
Merging two Binomial Heaps



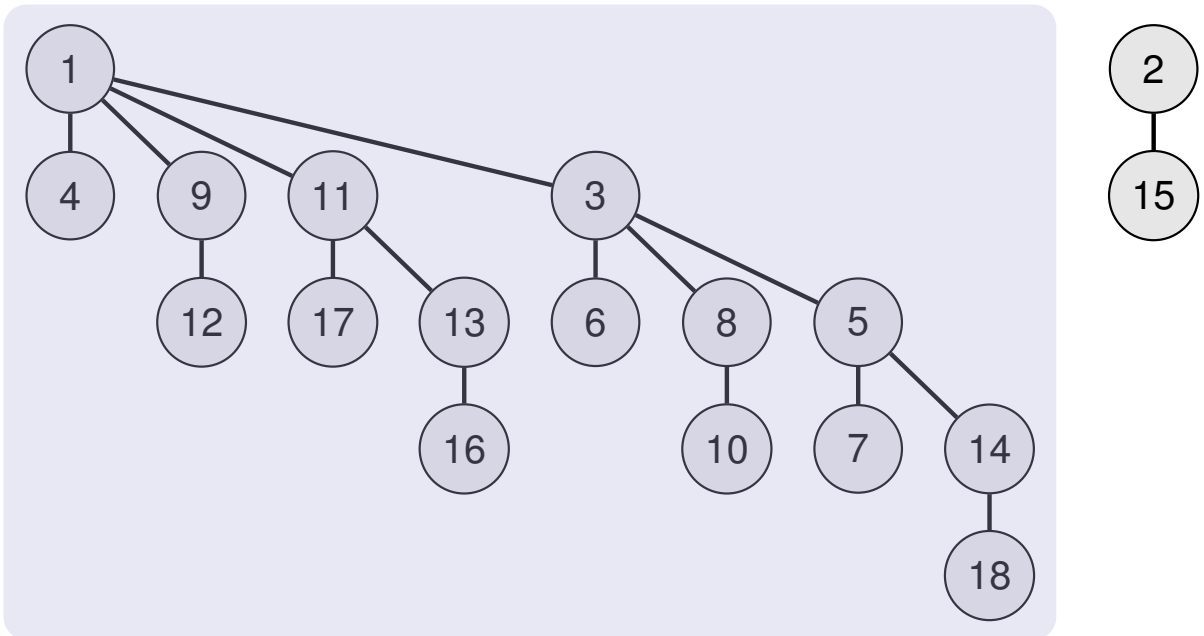


0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

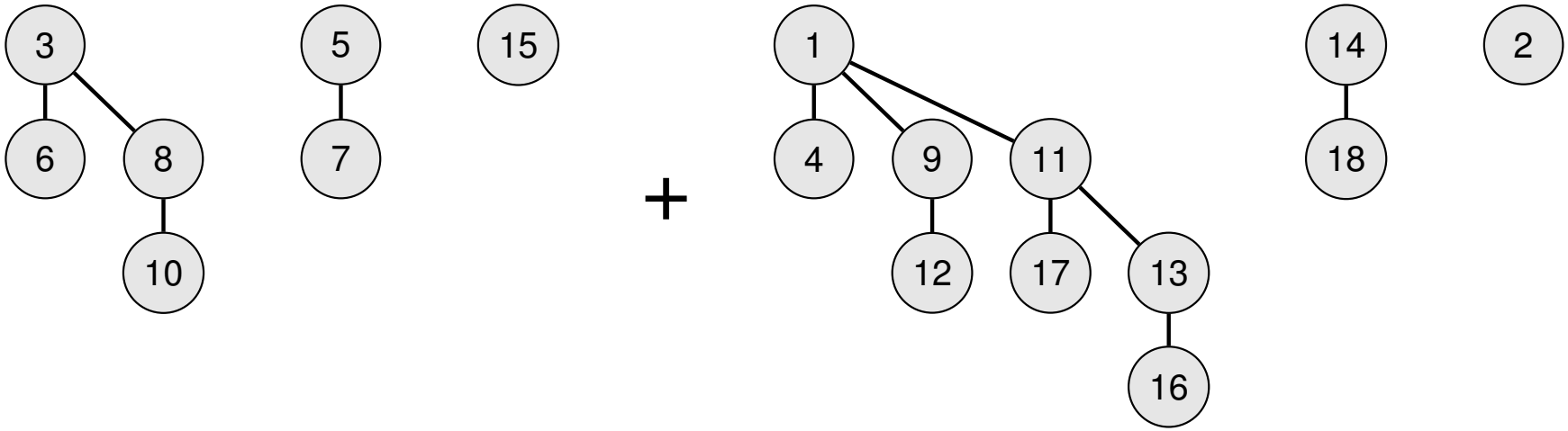
Merging two Binomial Heaps

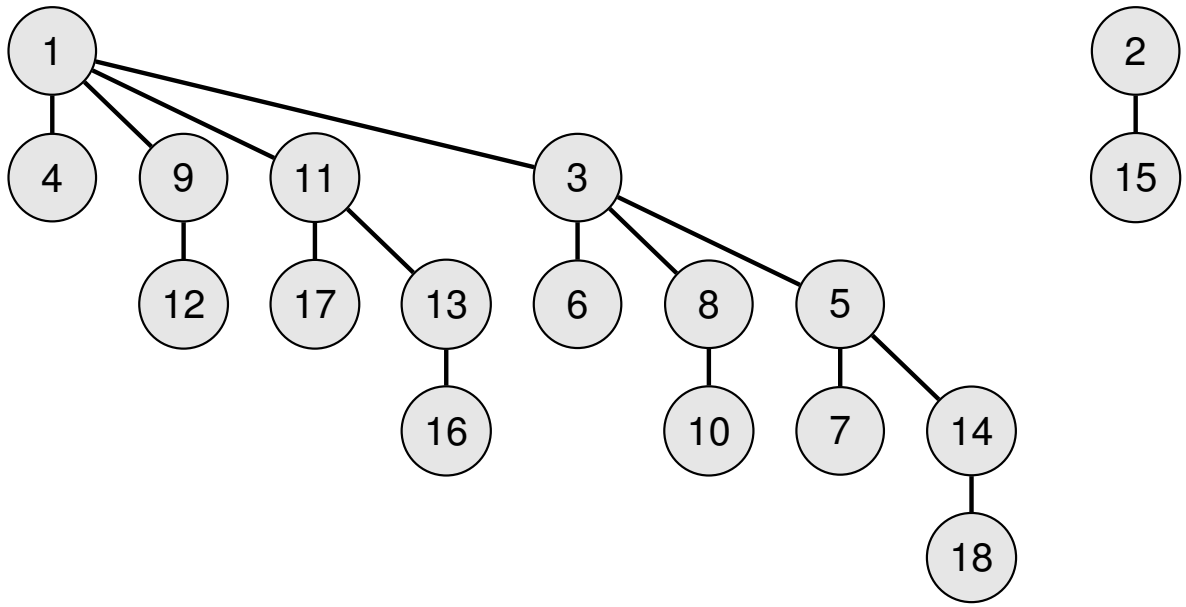


0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18



Merging two Binomial Heaps



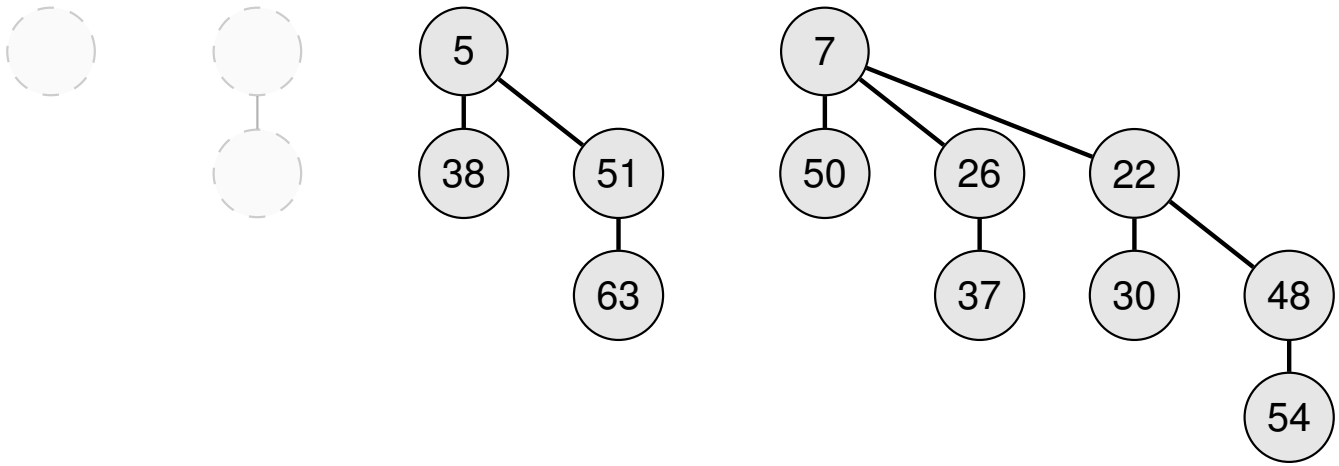


0	0	1	1	1	= 7
0	1	0	1	1	= 11
1	1	1	1		
1	0	0	1	0	= 18

Binomial Heap vs. Fibonacci Heap: Structure

Binomial Heap:

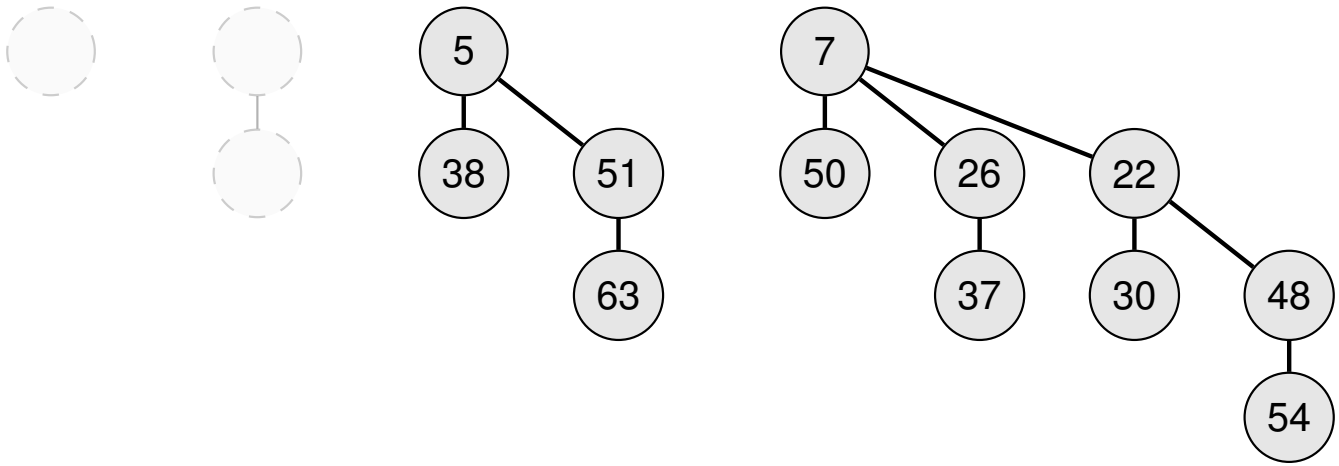
- consists of binomial trees, and every order appears at most once
- immediately tidy up after INSERT or MERGE



Binomial Heap vs. Fibonacci Heap: Structure

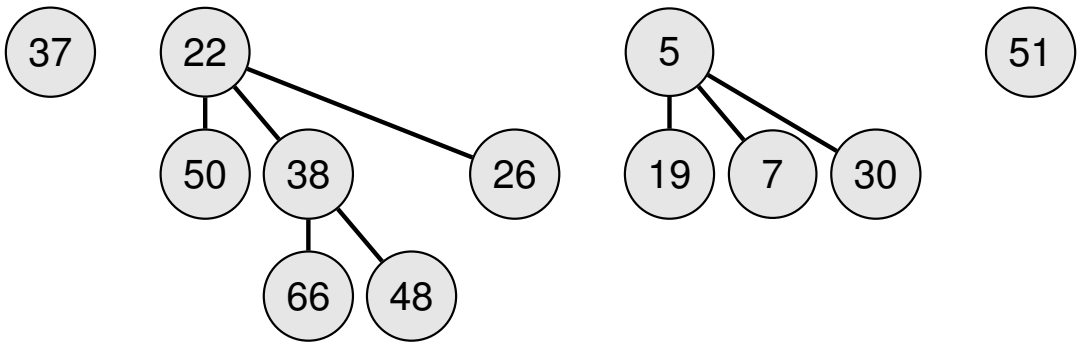
Binomial Heap:

- consists of binomial trees, and every order appears at most once
- immediately tidy up after INSERT or MERGE



Fibonacci Heap:

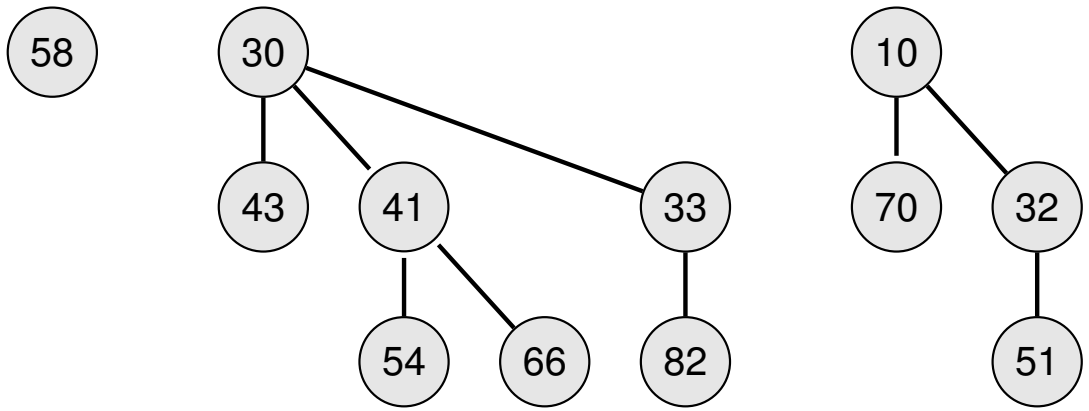
- forest of MIN-HEAPs
- lazily defer tidying up; do it on-the-fly when search for the MIN



Structure of Fibonacci Heaps

Fibonacci Heap

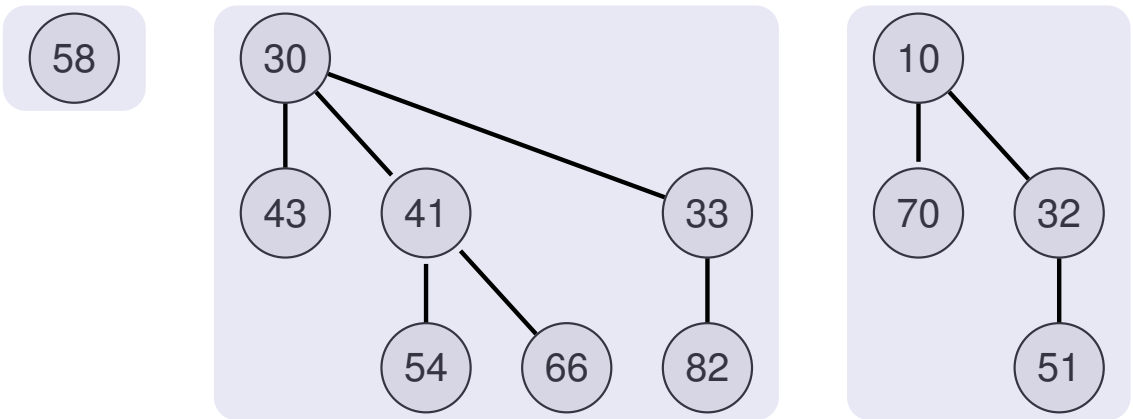
- Forest of MIN-HEAPs



Structure of Fibonacci Heaps

Fibonacci Heap

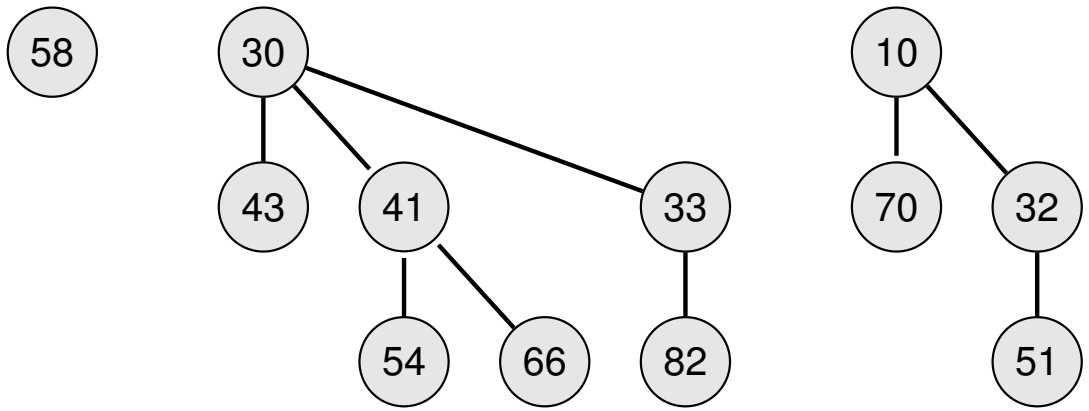
- Forest of MIN-HEAPs



Structure of Fibonacci Heaps

Fibonacci Heap

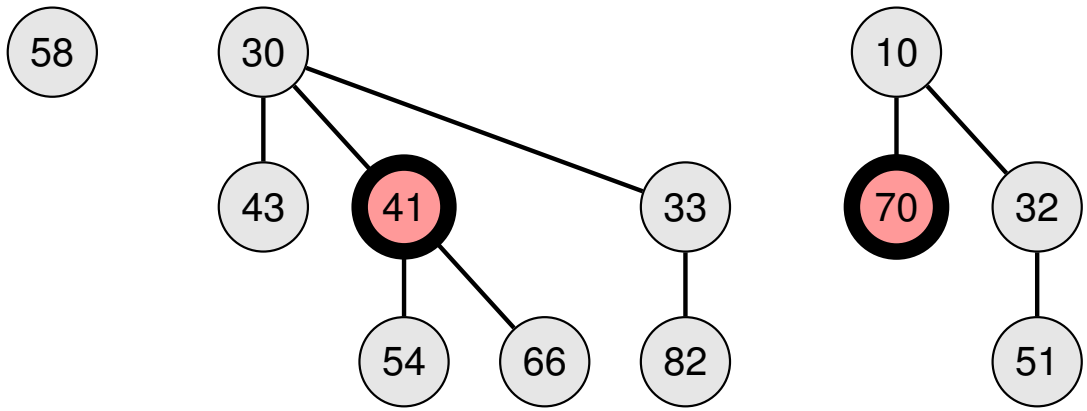
- Forest of MIN-HEAPs
- Nodes can be marked (roots are always unmarked)



Structure of Fibonacci Heaps

Fibonacci Heap

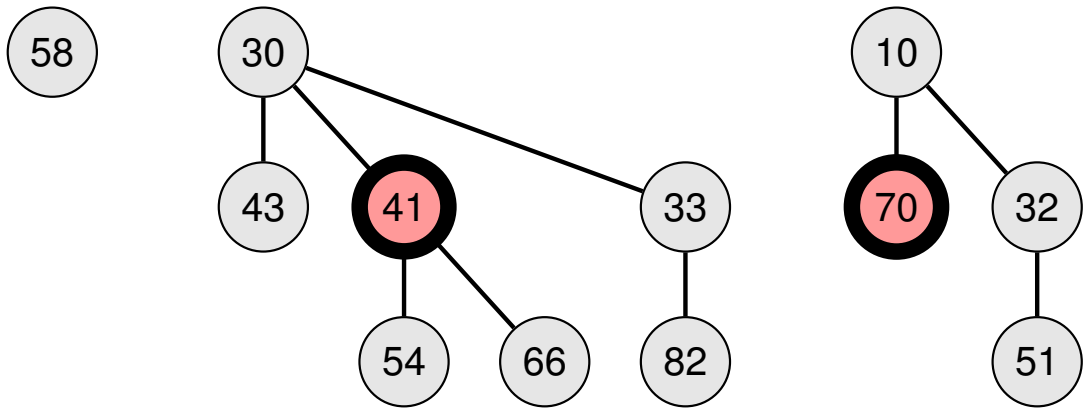
- Forest of MIN-HEAPs
- Nodes can be marked (roots are always unmarked)



Structure of Fibonacci Heaps

Fibonacci Heap

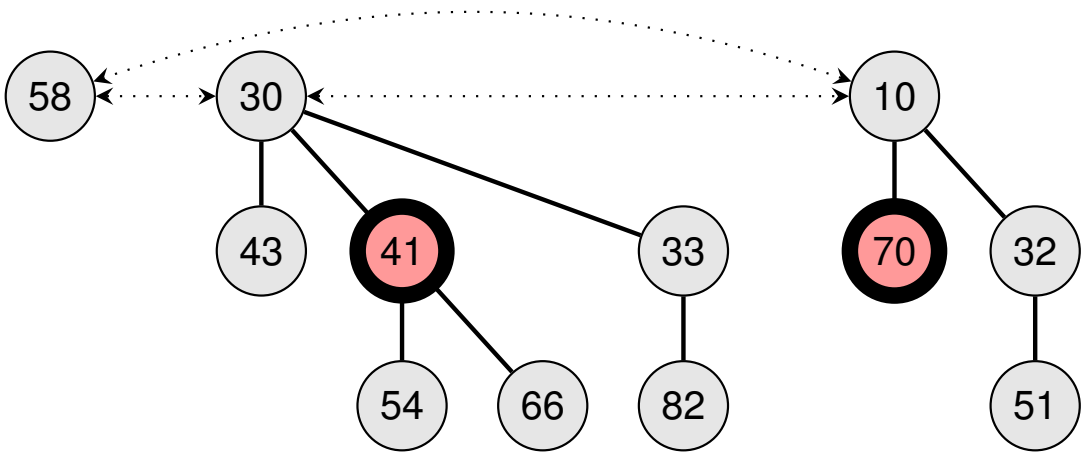
- Forest of MIN-HEAPs
- Nodes can be marked (roots are always unmarked)
- Tree roots are stored in a circular, doubly-linked list



Structure of Fibonacci Heaps

Fibonacci Heap

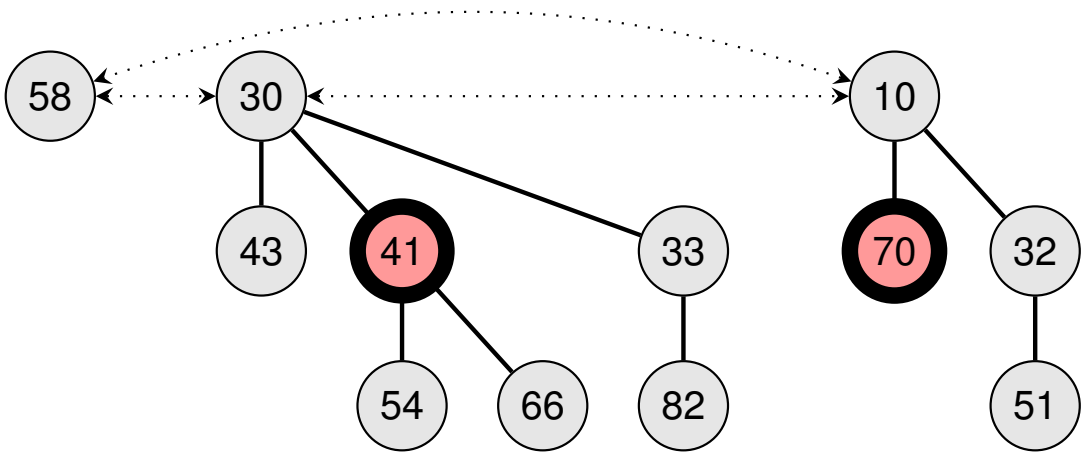
- Forest of MIN-HEAPs
- Nodes can be marked (roots are always unmarked)
- Tree roots are stored in a circular, doubly-linked list



Structure of Fibonacci Heaps

Fibonacci Heap

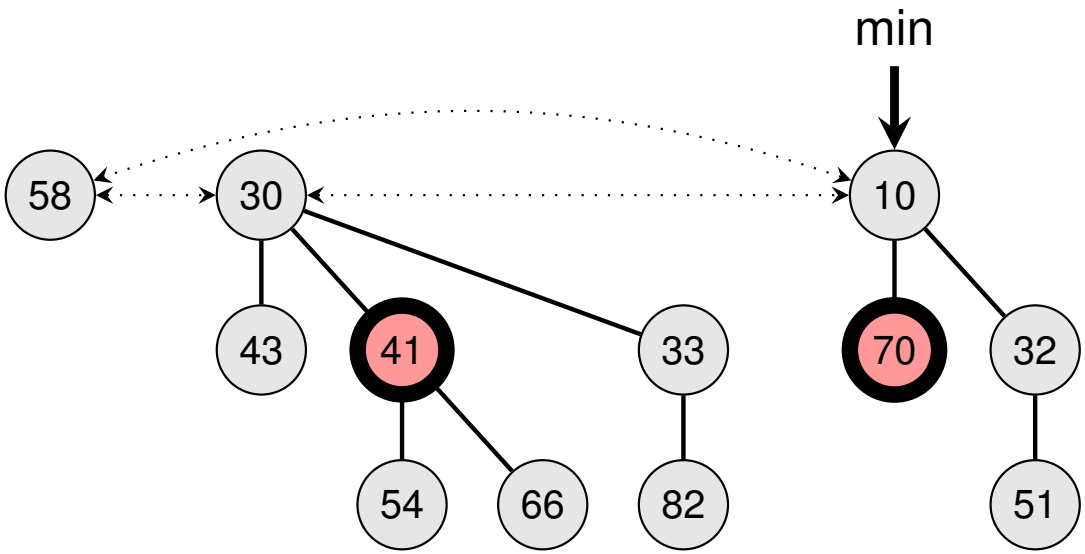
- Forest of MIN-HEAPs
- Nodes can be marked (roots are always unmarked)
- Tree roots are stored in a circular, doubly-linked list
- Min-Pointer pointing to the smallest element



Structure of Fibonacci Heaps

Fibonacci Heap

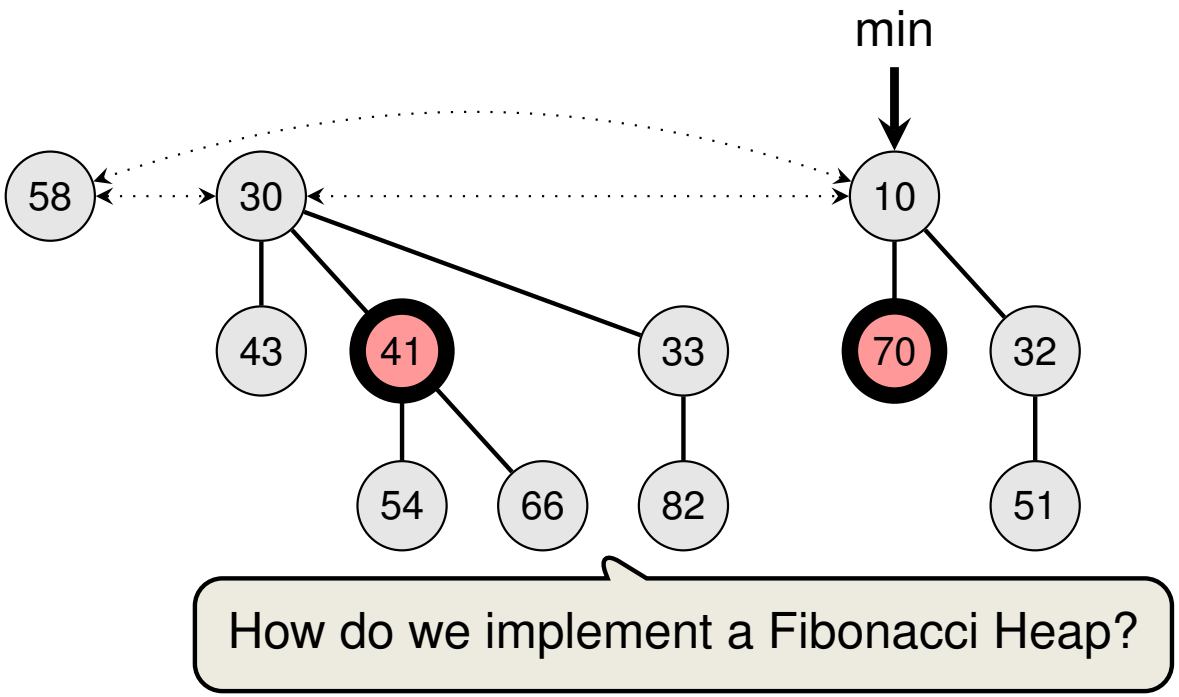
- Forest of MIN-HEAPs
- Nodes can be marked (roots are always unmarked)
- Tree roots are stored in a circular, doubly-linked list
- Min-Pointer pointing to the smallest element



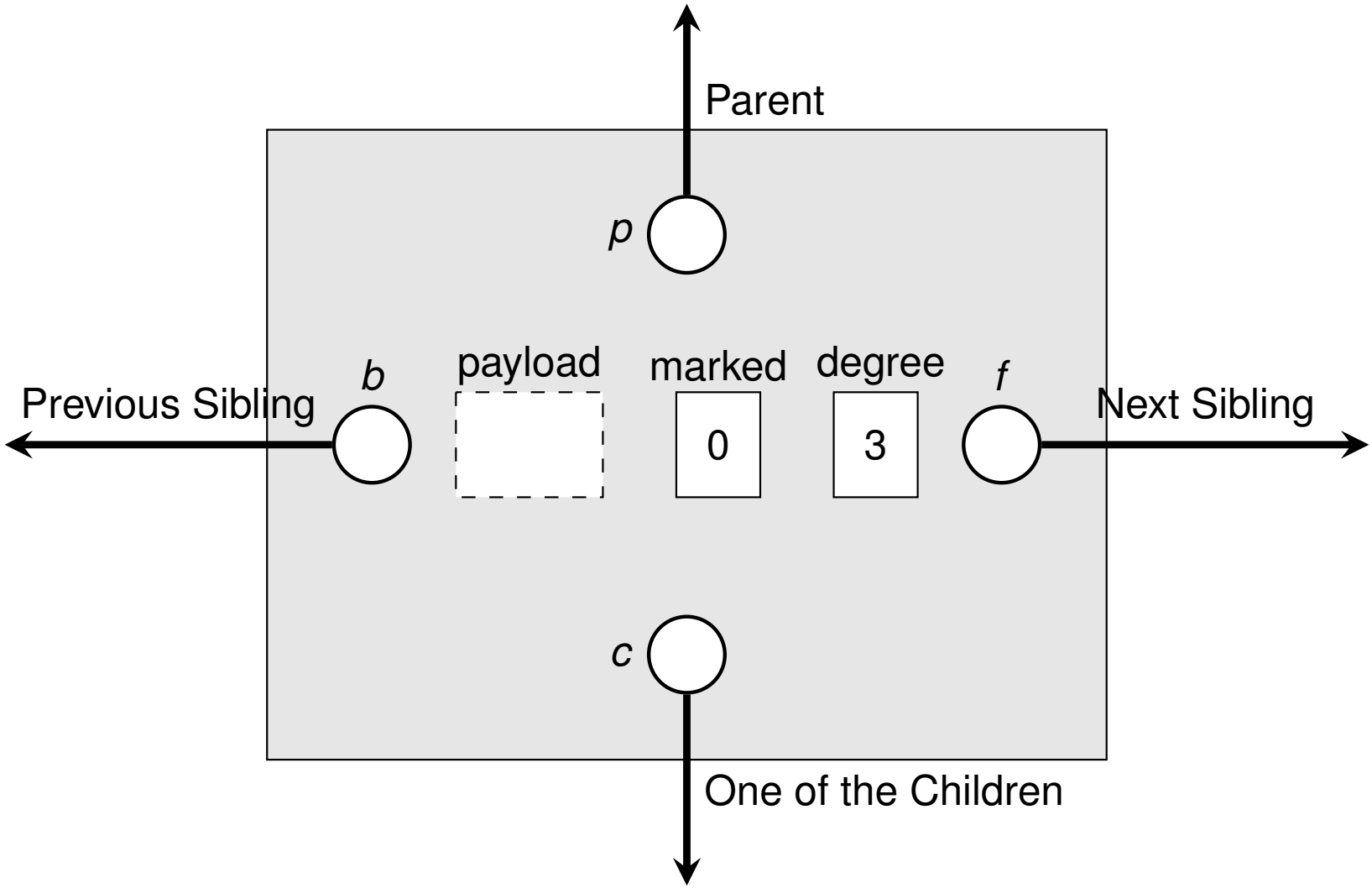
Structure of Fibonacci Heaps

Fibonacci Heap

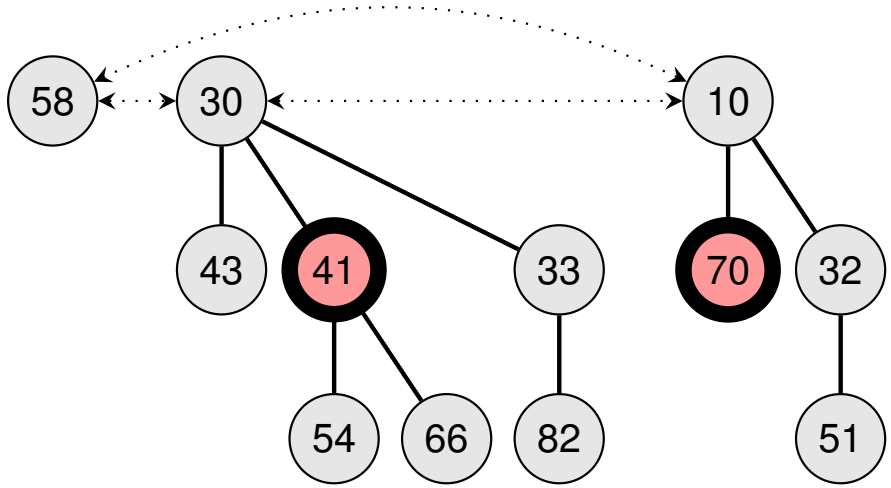
- Forest of MIN-HEAPs
- Nodes can be marked (roots are always unmarked)
- Tree roots are stored in a circular, doubly-linked list
- Min-Pointer pointing to the smallest element



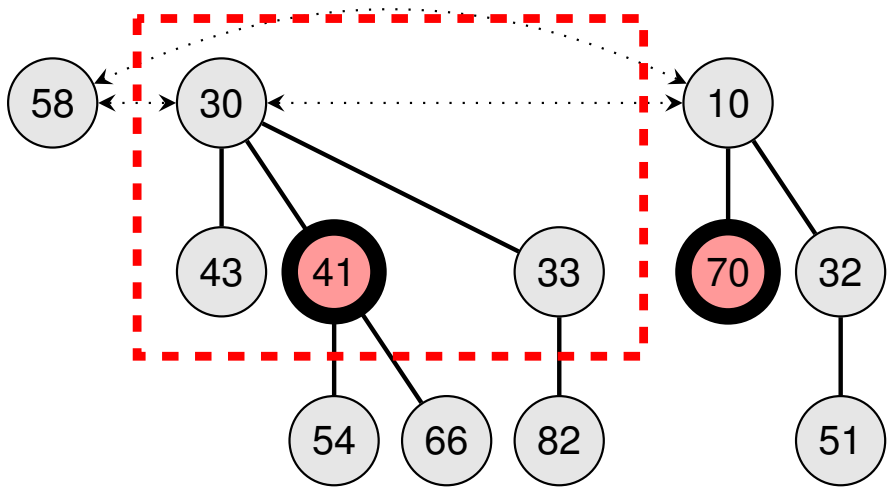
A single Node



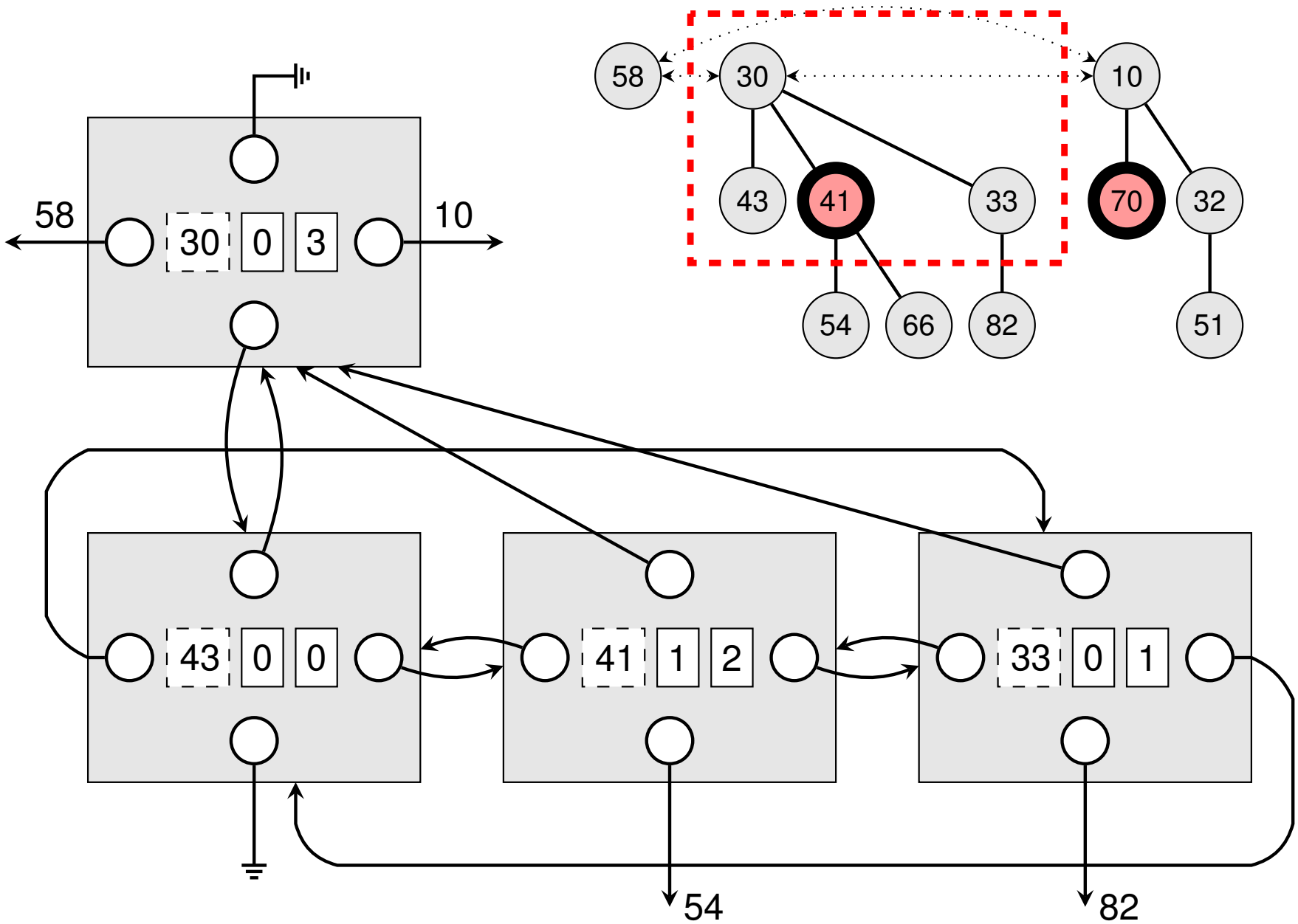
Magnifying a Four-Node Portion



Magnifying a Four-Node Portion



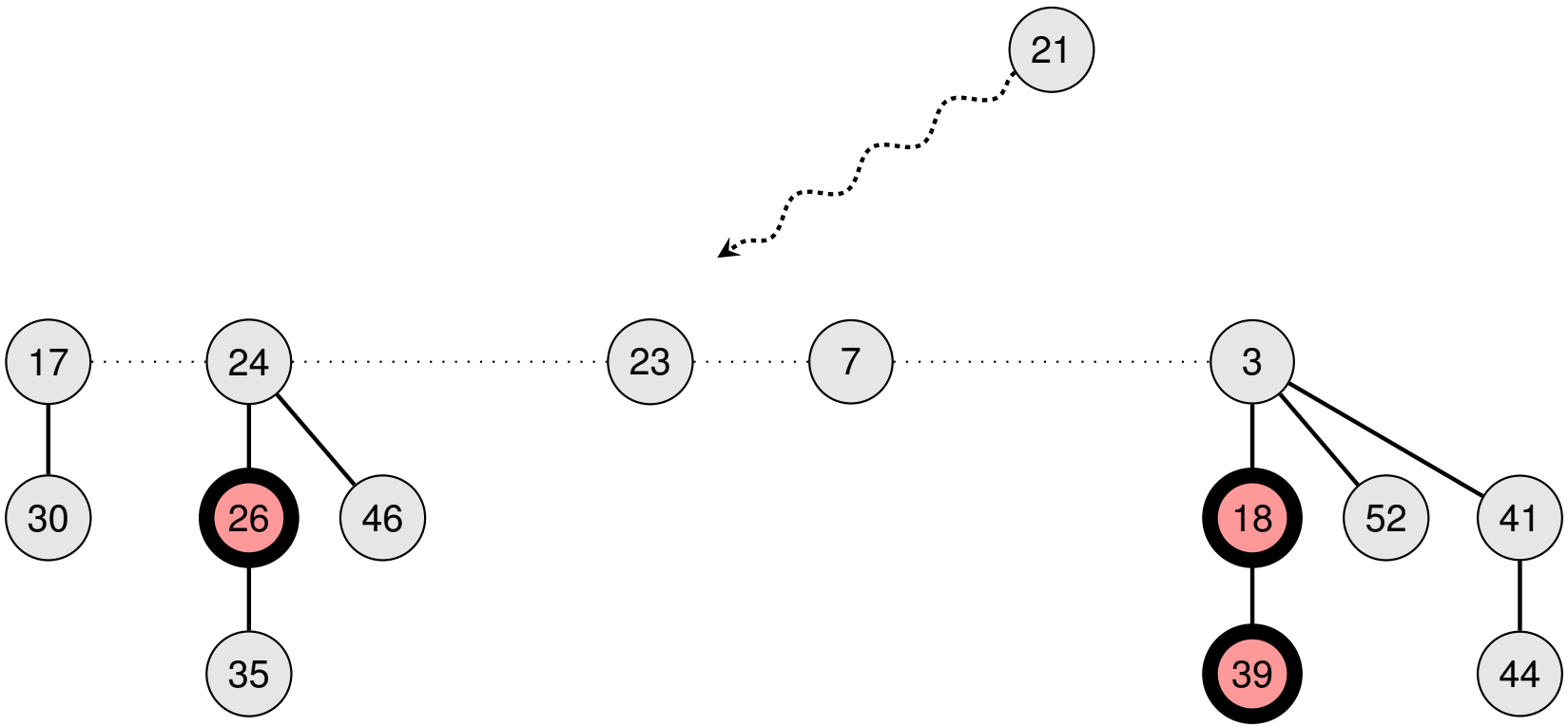
Magnifying a Four-Node Portion



Outline

Fibonacci Heap: INSERT

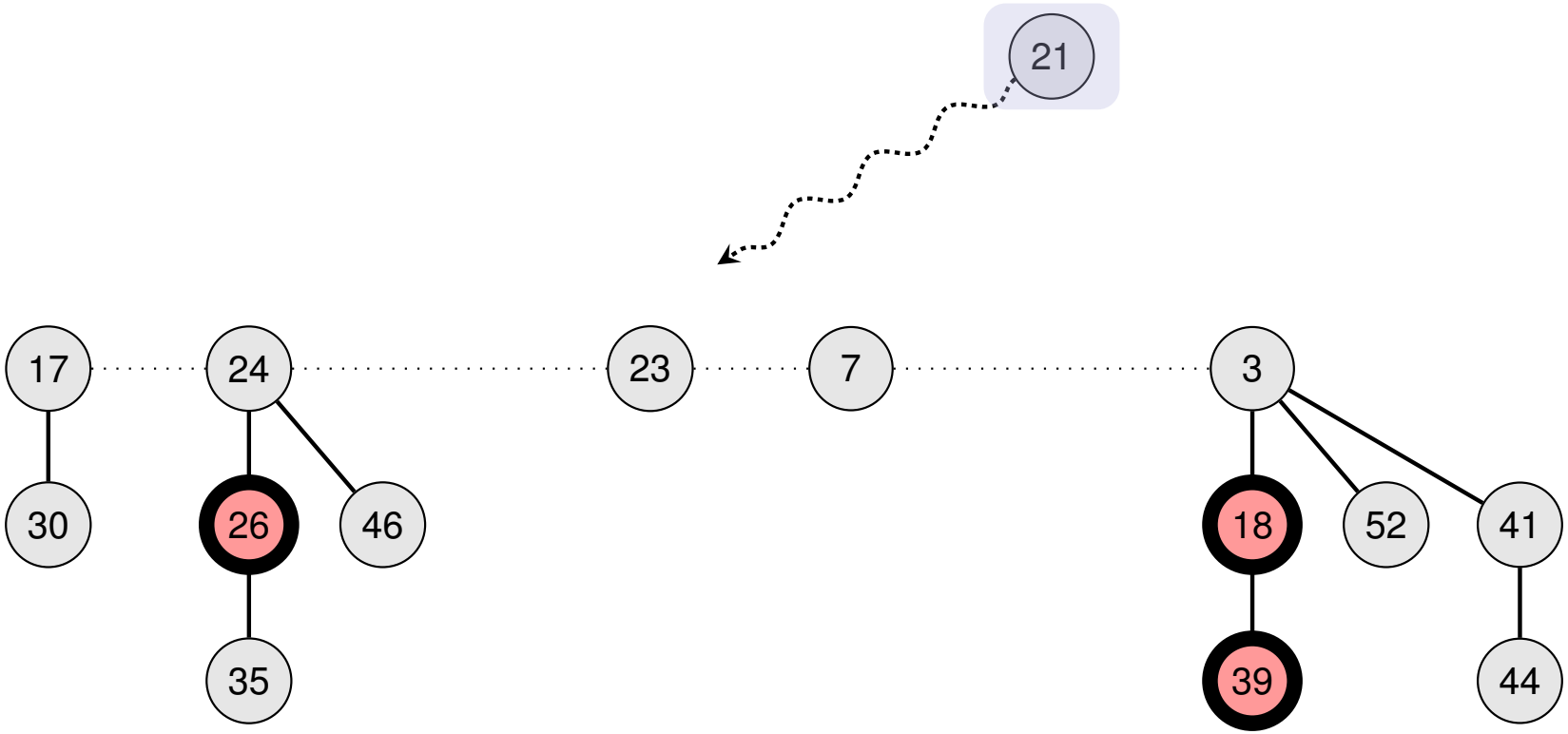
INSERT



Fibonacci Heap: INSERT

INSERT

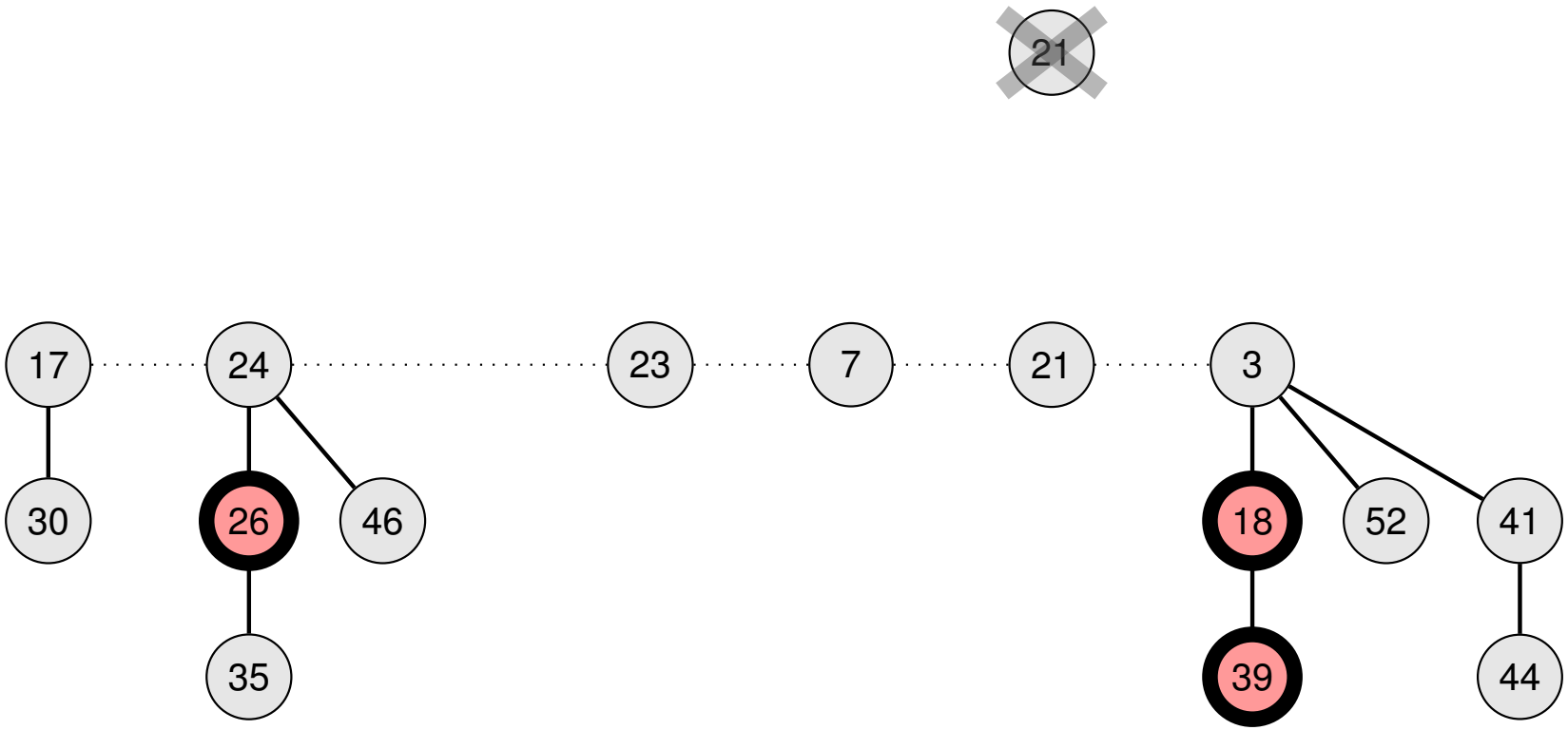
- Create a singleton tree



Fibonacci Heap: INSERT

INSERT

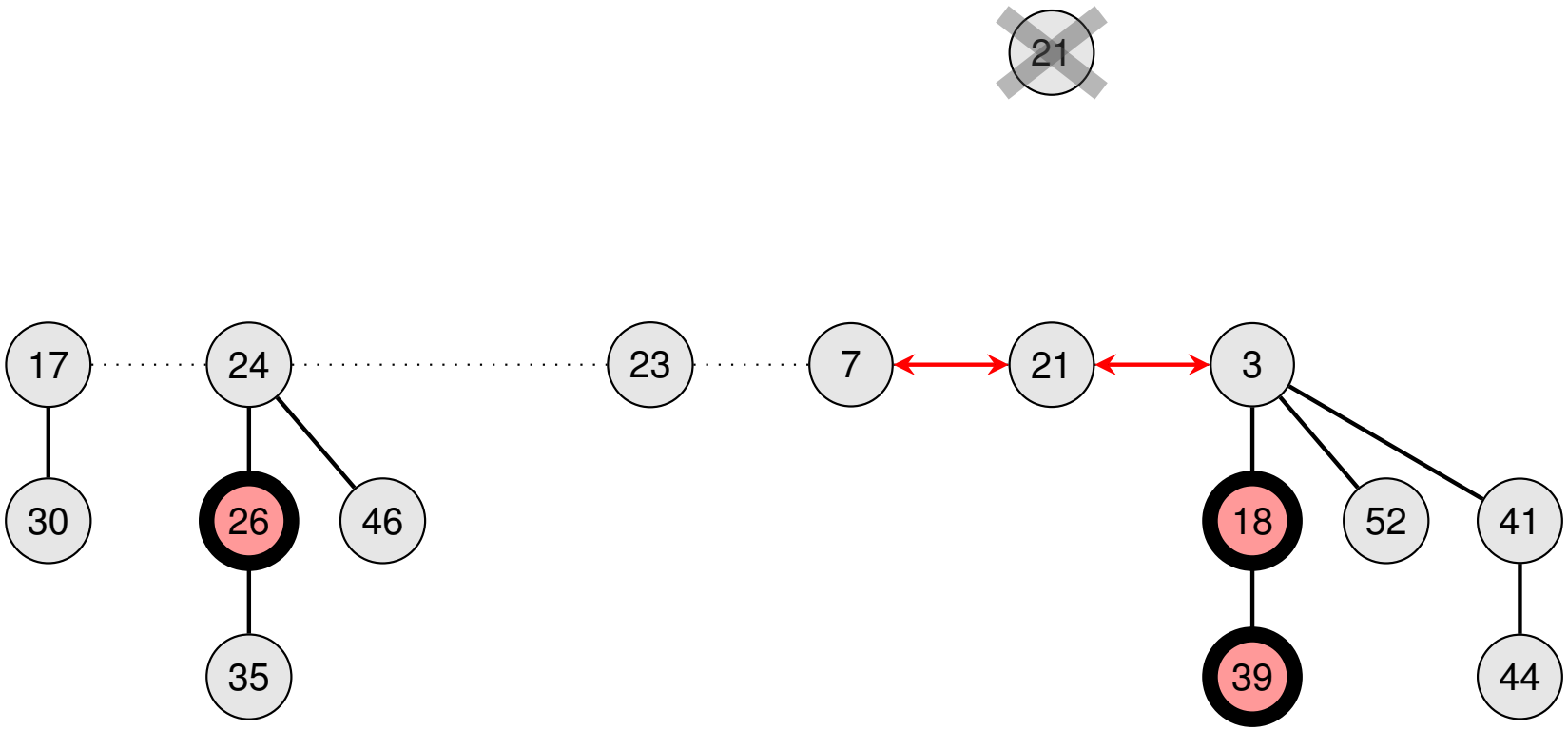
- Create a singleton tree
- Add to root list



Fibonacci Heap: INSERT

INSERT

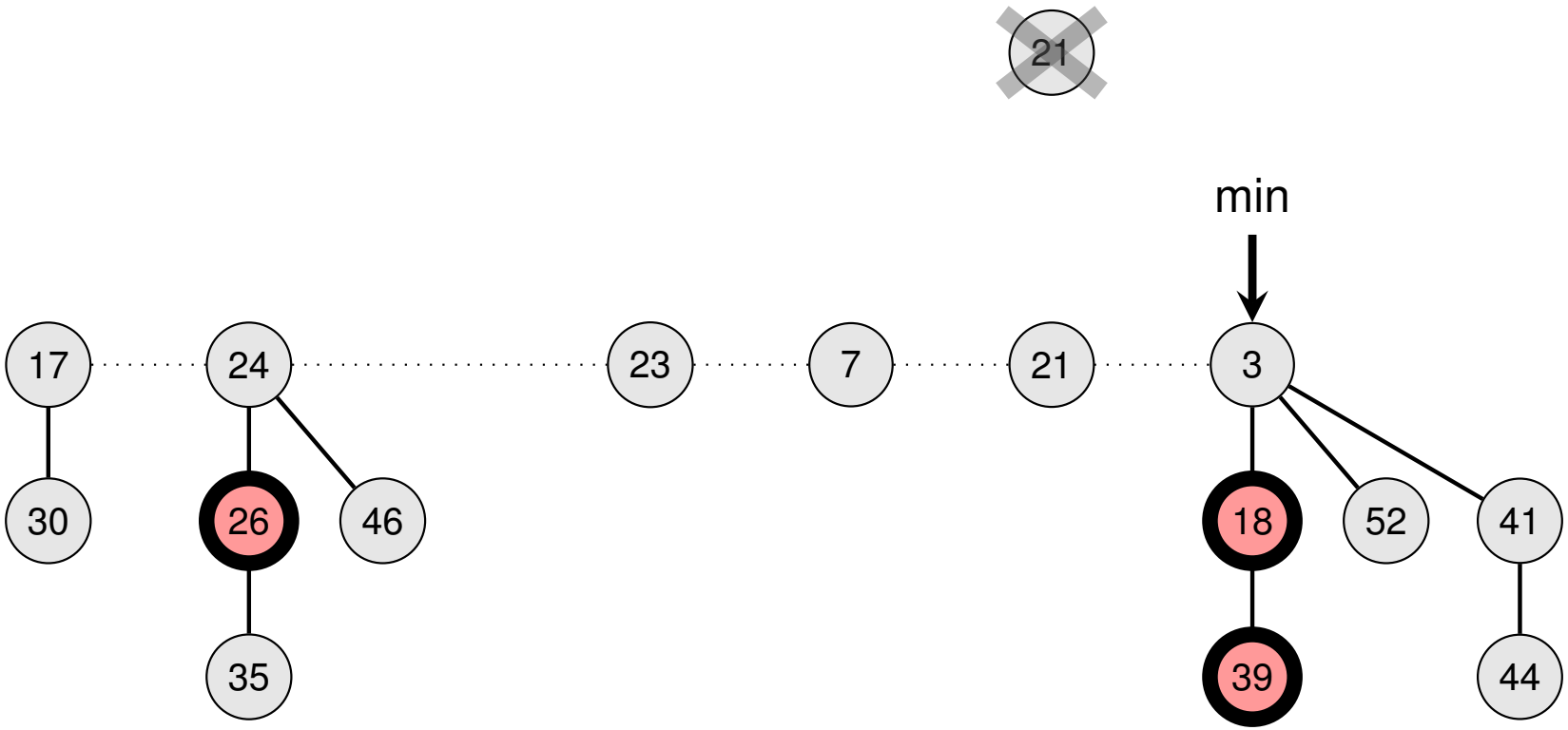
- Create a singleton tree
- Add to root list



Fibonacci Heap: INSERT

INSERT

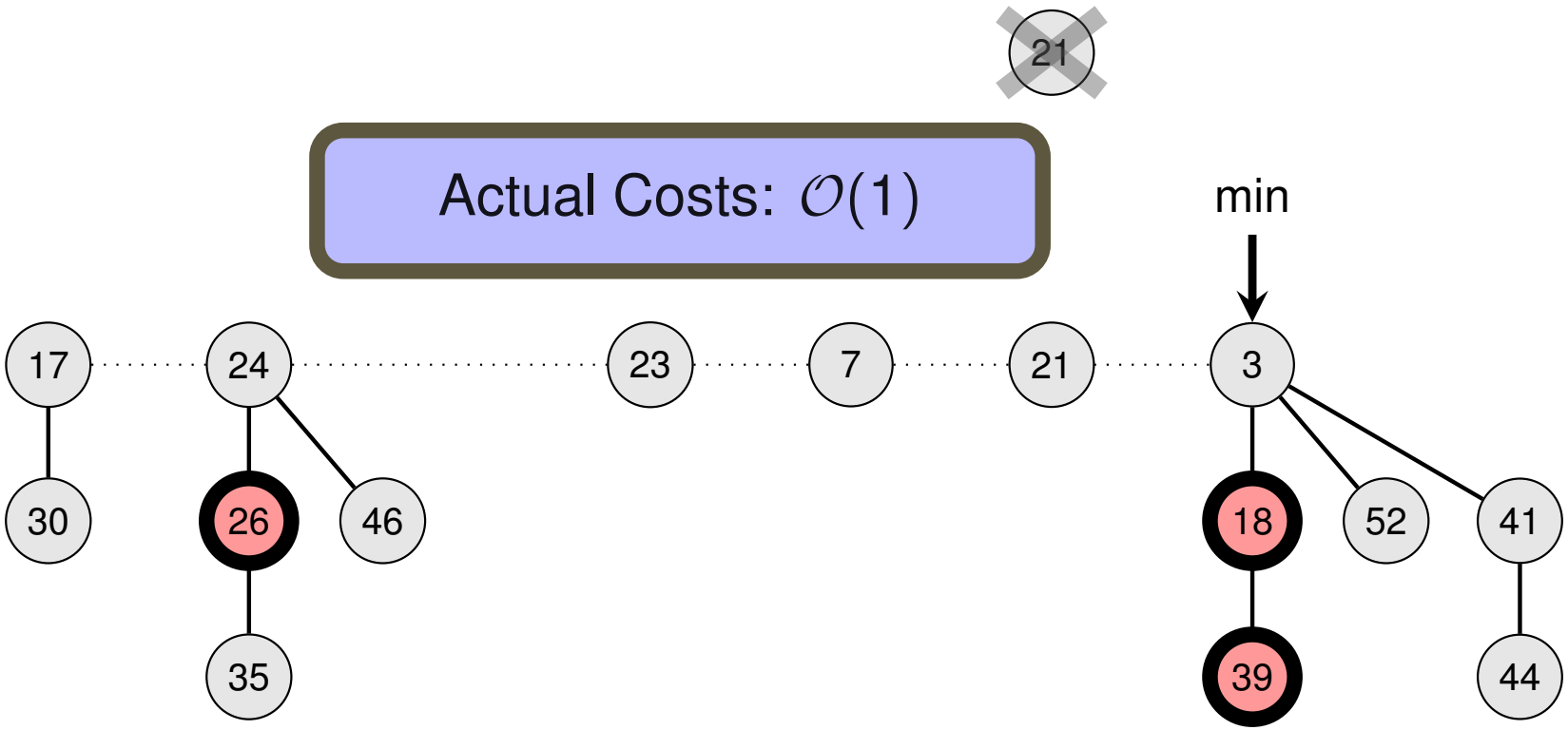
- Create a singleton tree
- Add to root list and update min-pointer (if necessary)



Fibonacci Heap: INSERT

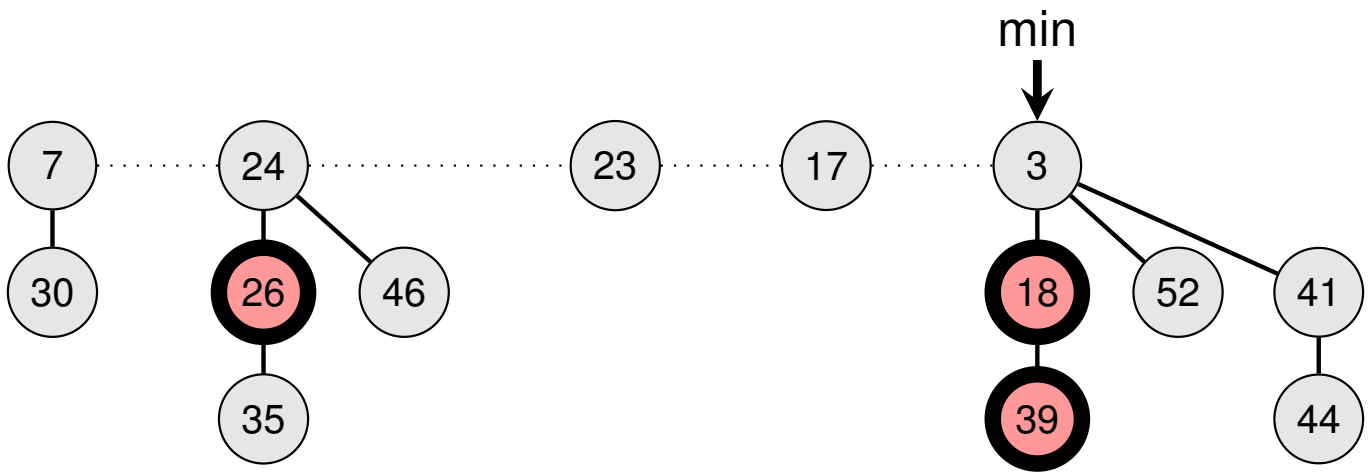
INSERT

- Create a singleton tree
- Add to root list and update min-pointer (if necessary)



Fibonacci Heap: EXTRACT-MIN

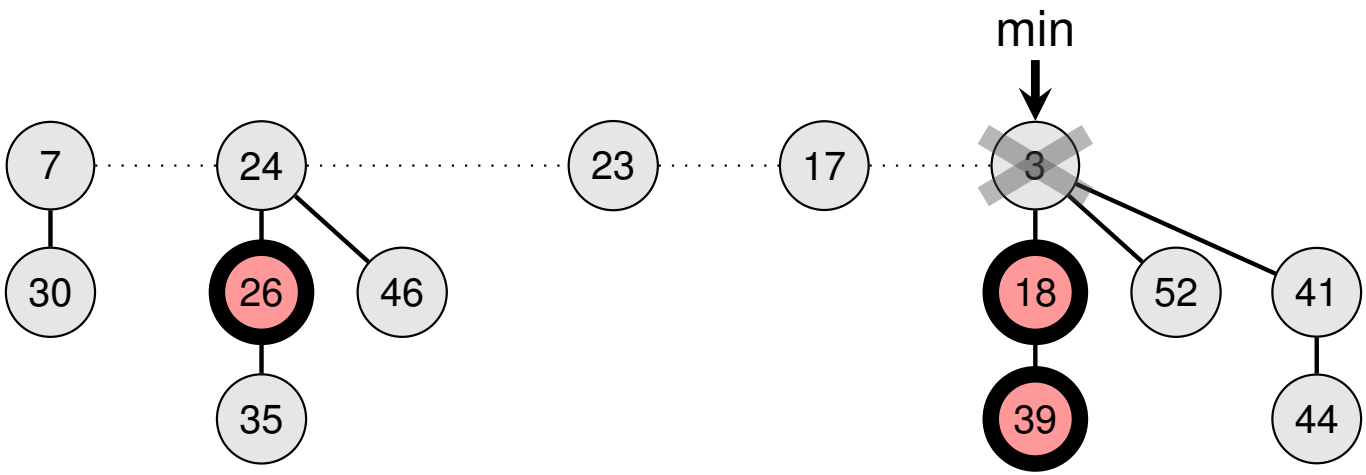
EXTRACT-MIN



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

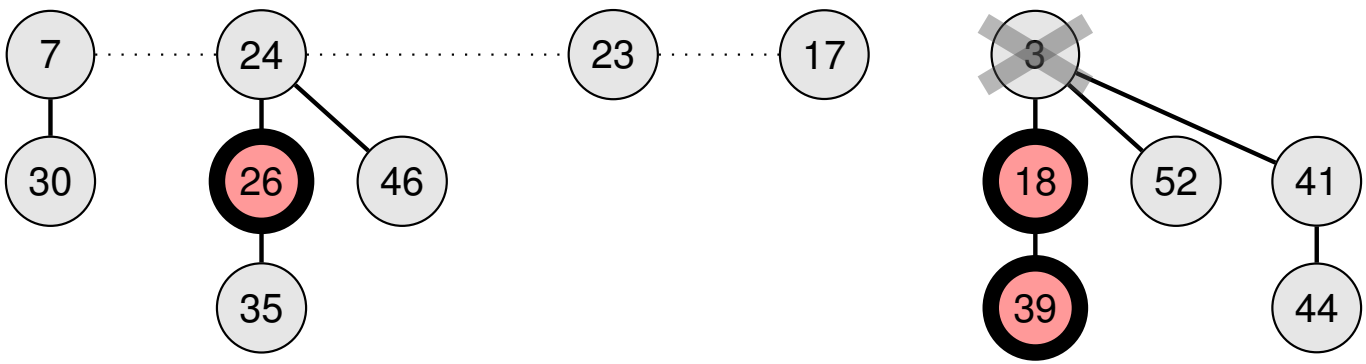
- Delete min



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

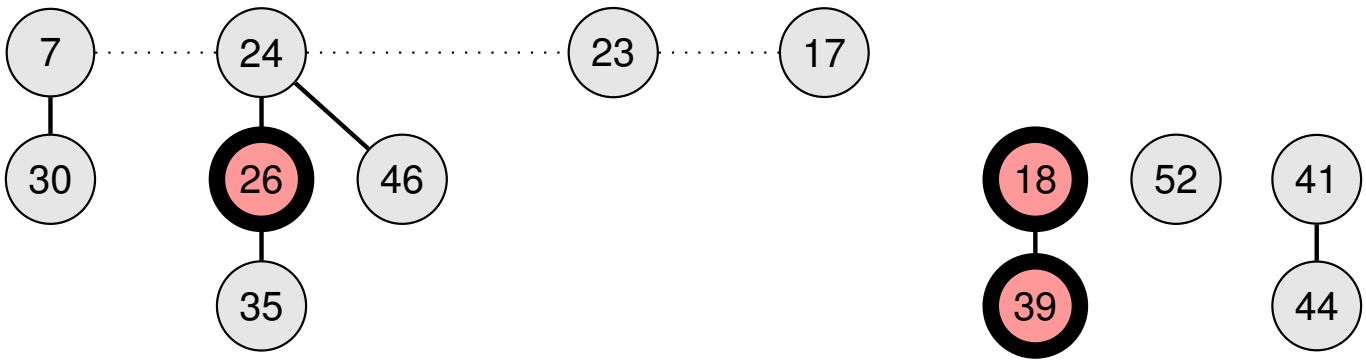
- Delete min ✓



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

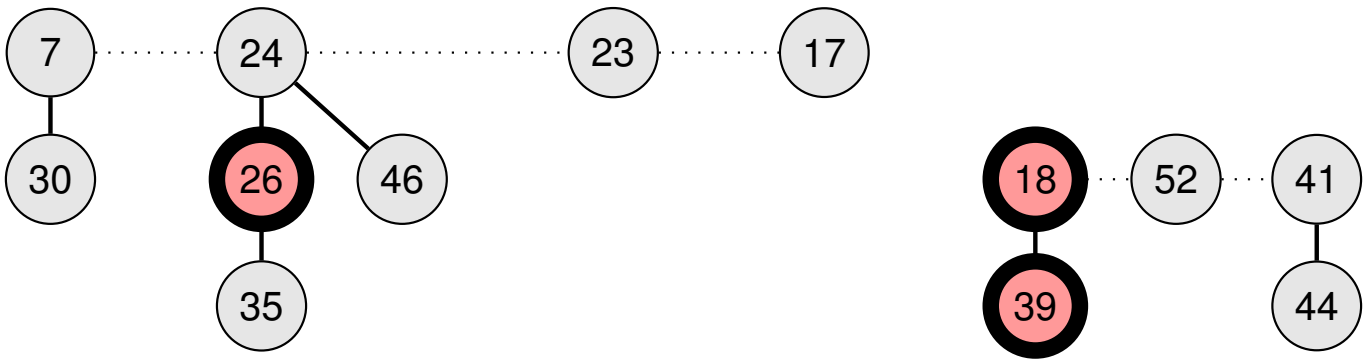
- Delete min ✓
- Meld childen into root list and unmark them



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

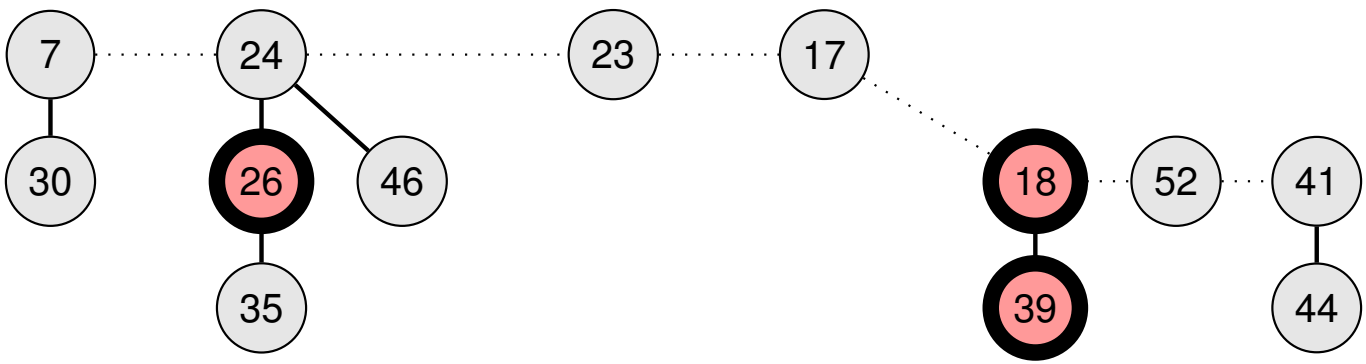
- Delete min ✓
- Meld childen into root list and unmark them



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

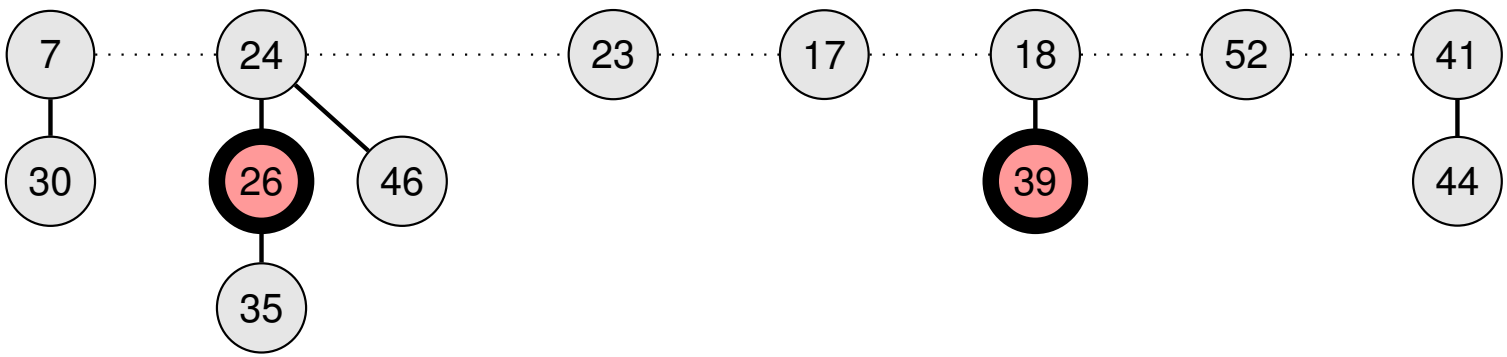
- Delete min ✓
- Meld childen into root list and unmark them



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

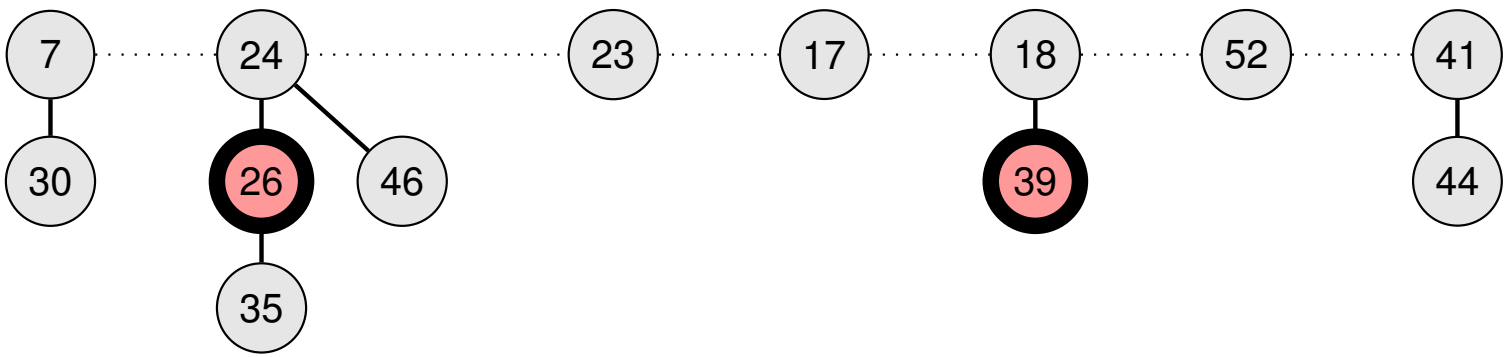
- Delete min ✓
- Meld childen into root list and unmark them ✓



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

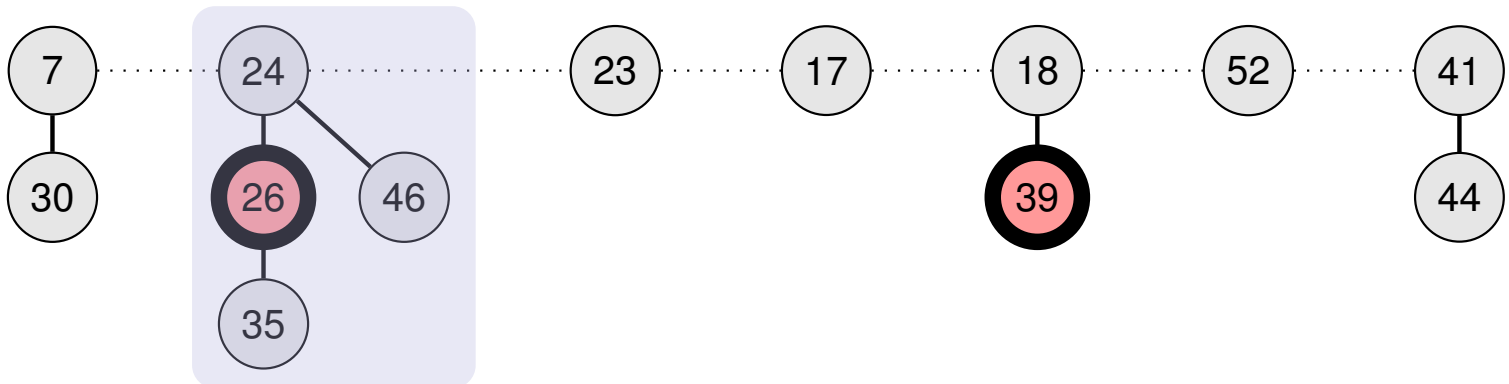
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

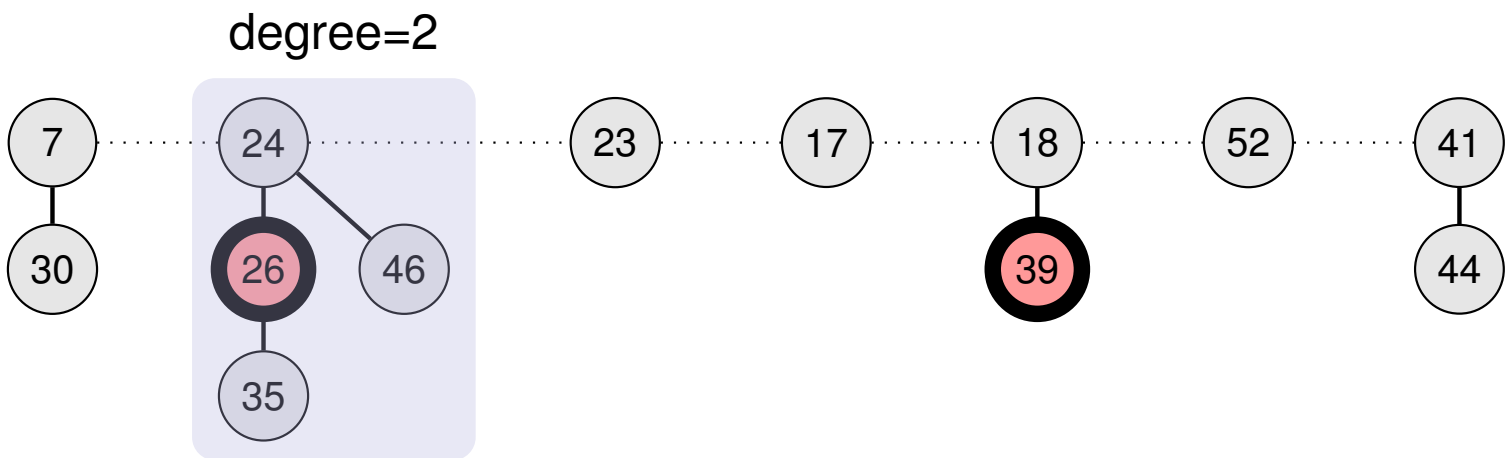
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

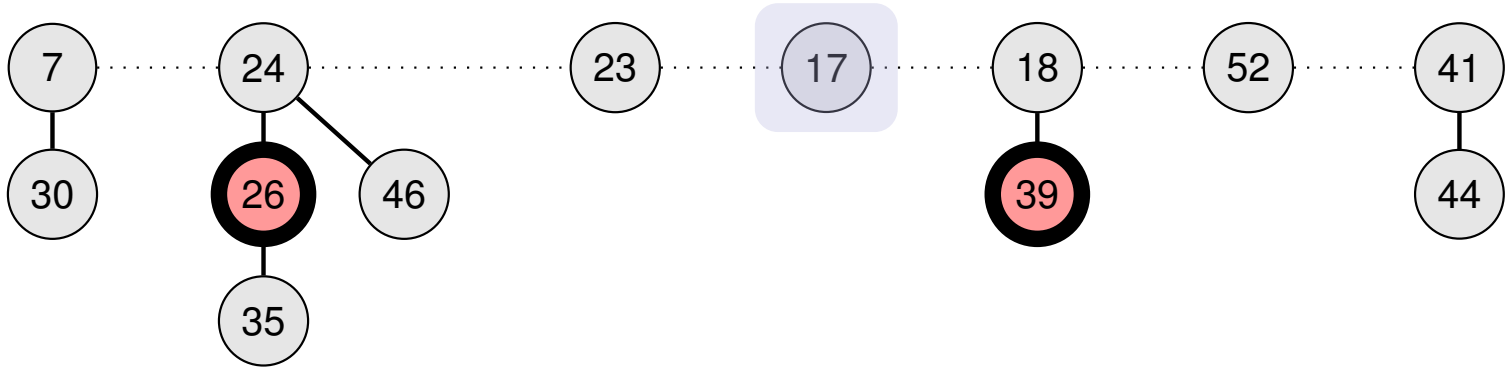
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

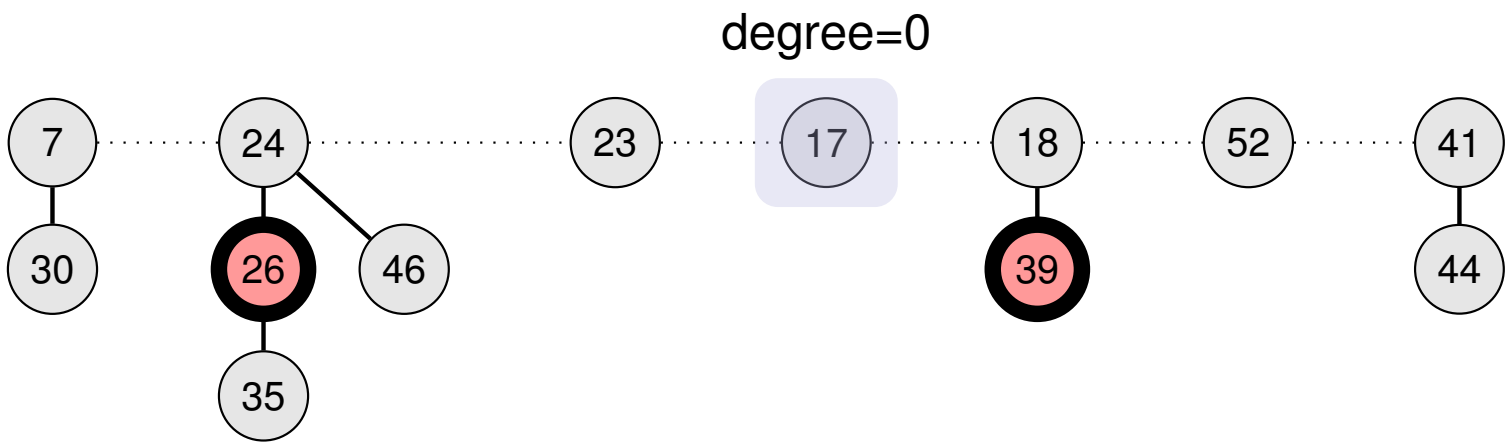
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

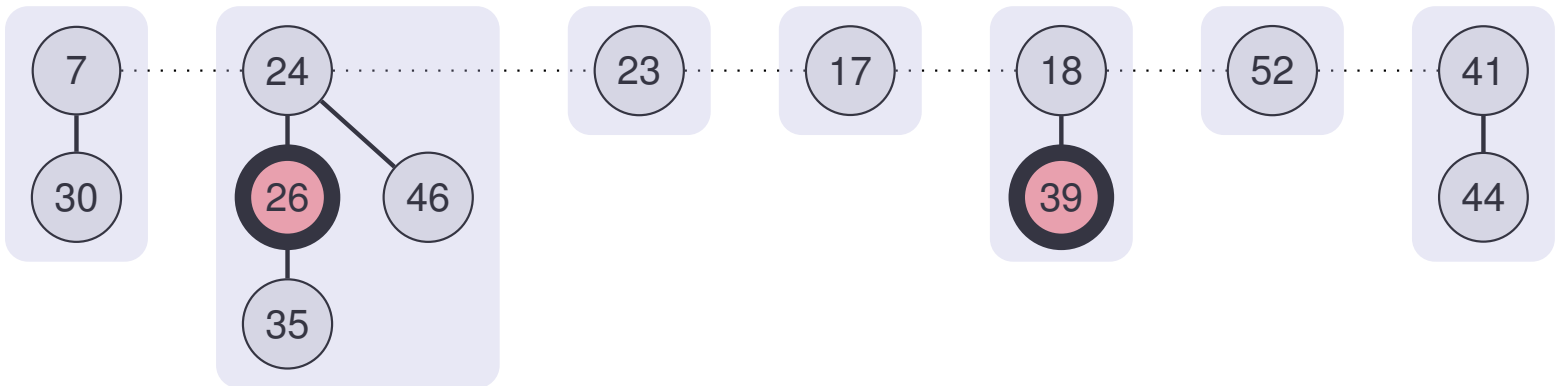
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

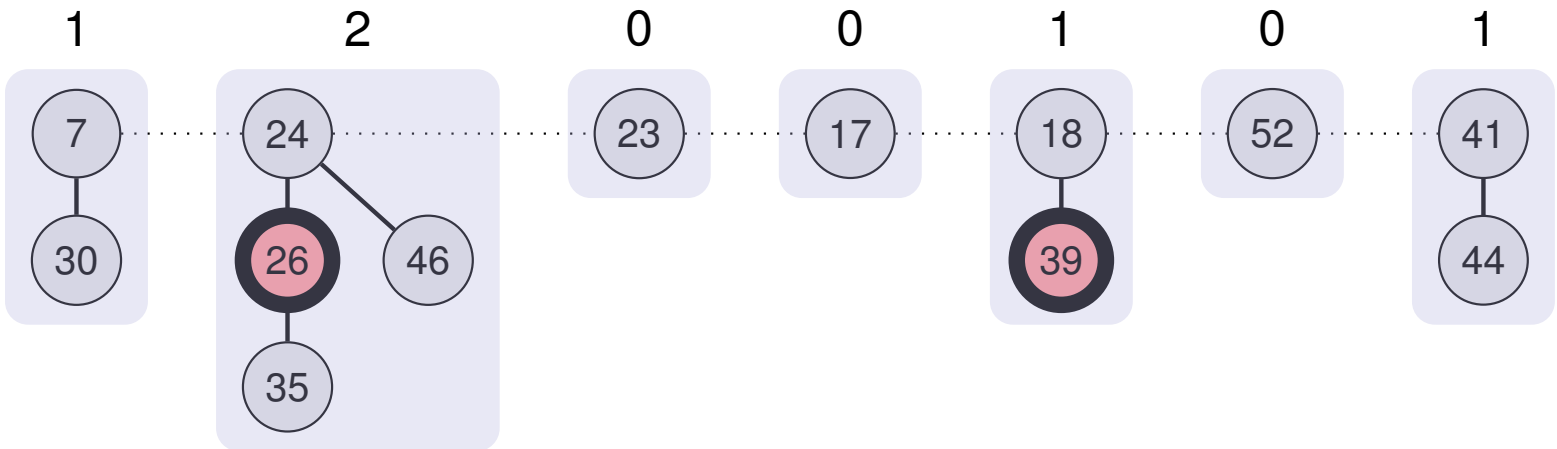
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

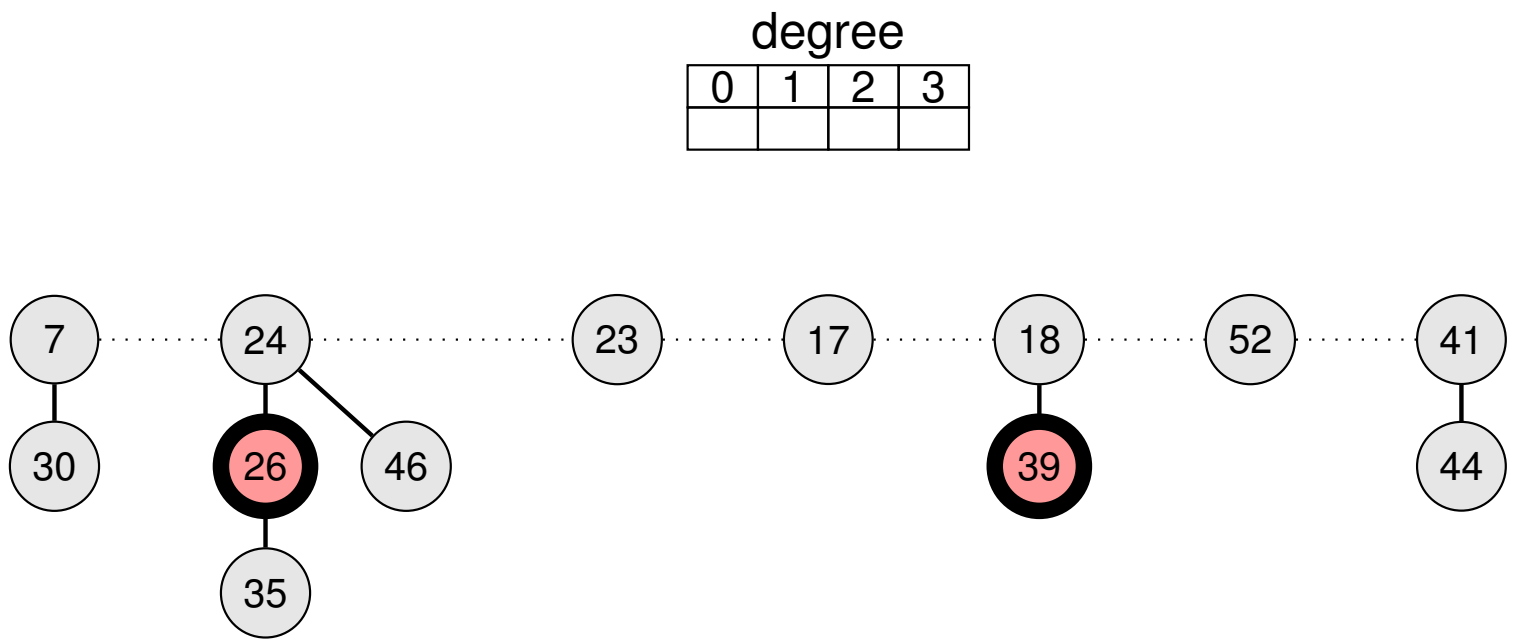
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

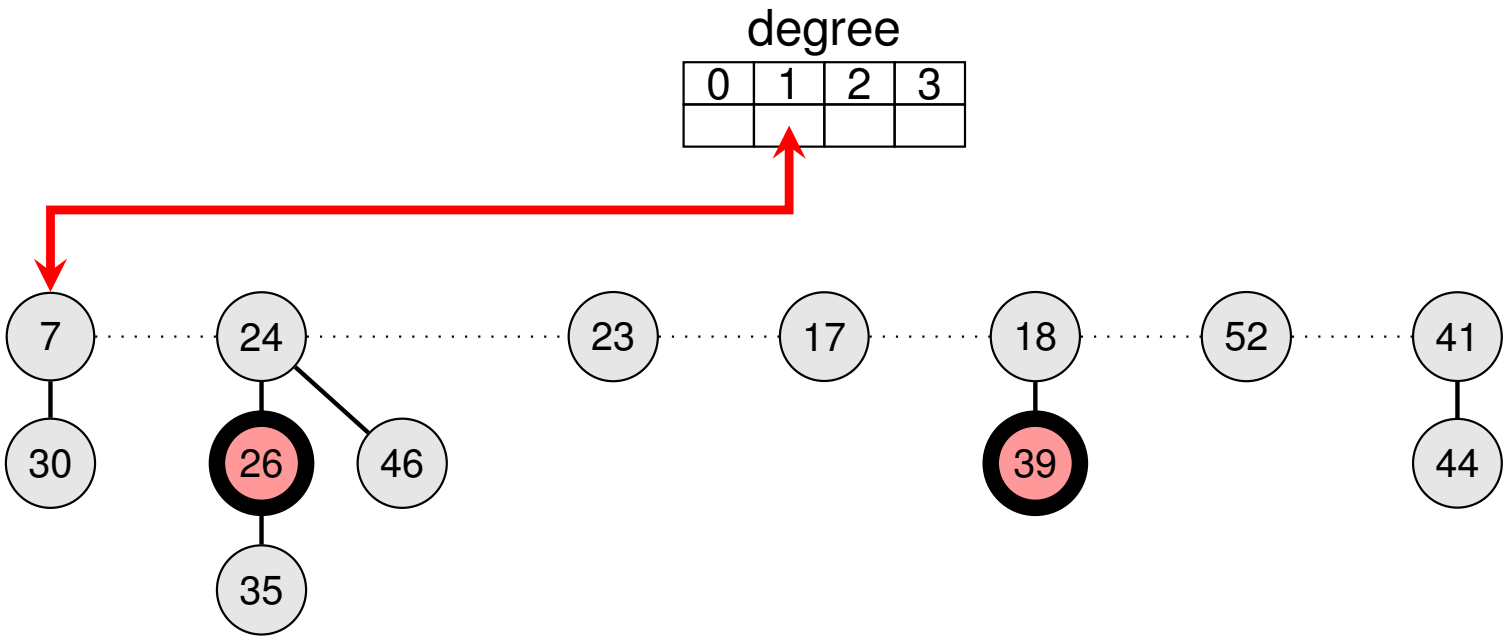
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

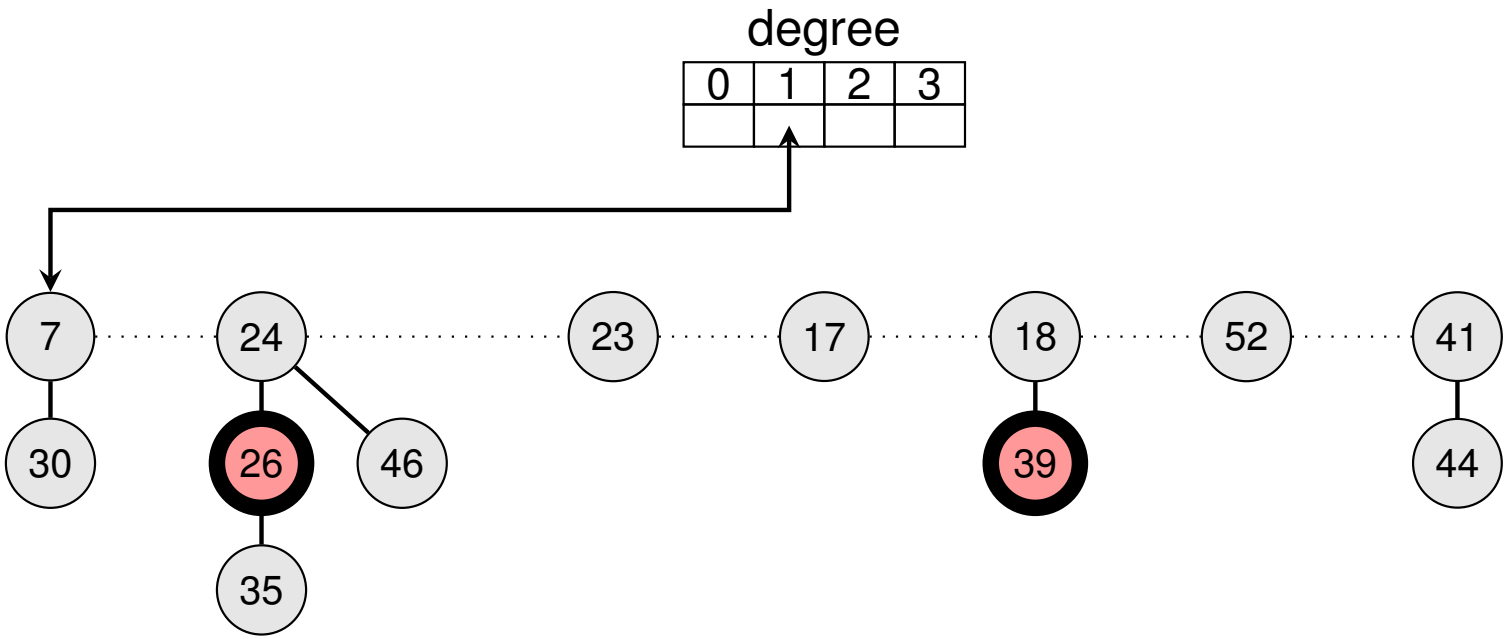
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

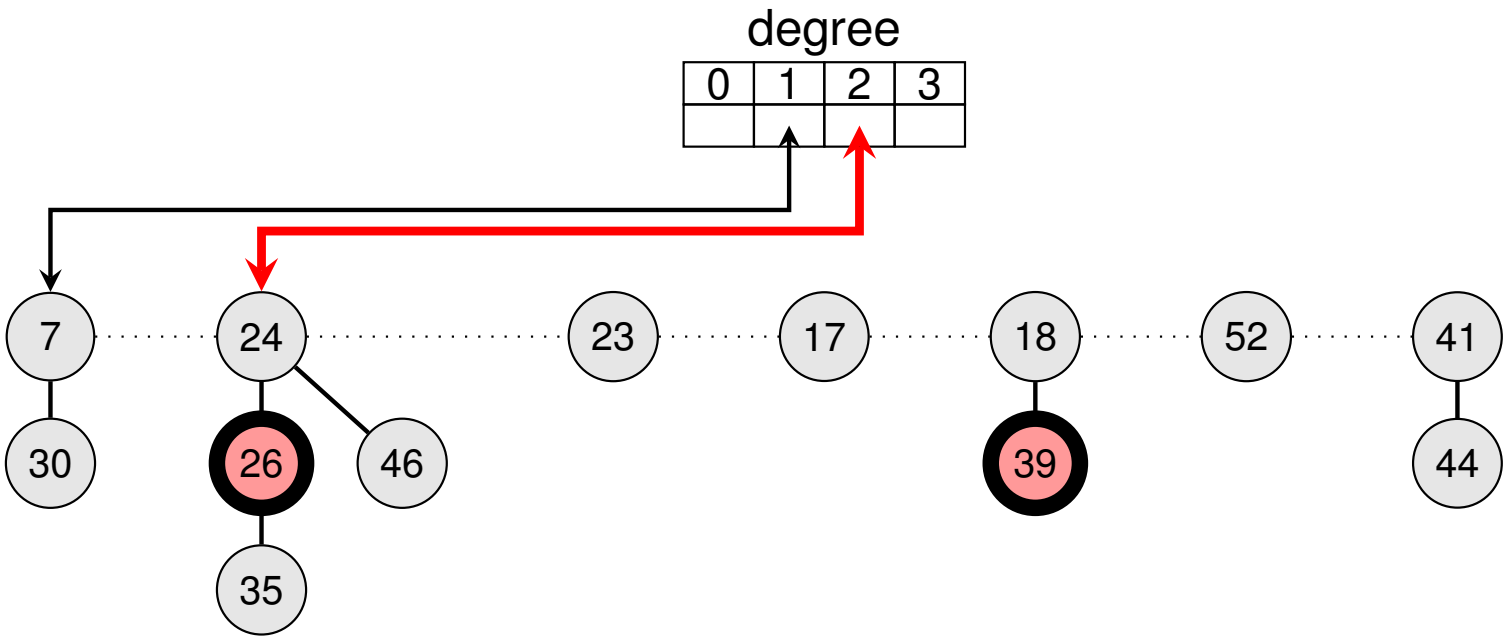
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

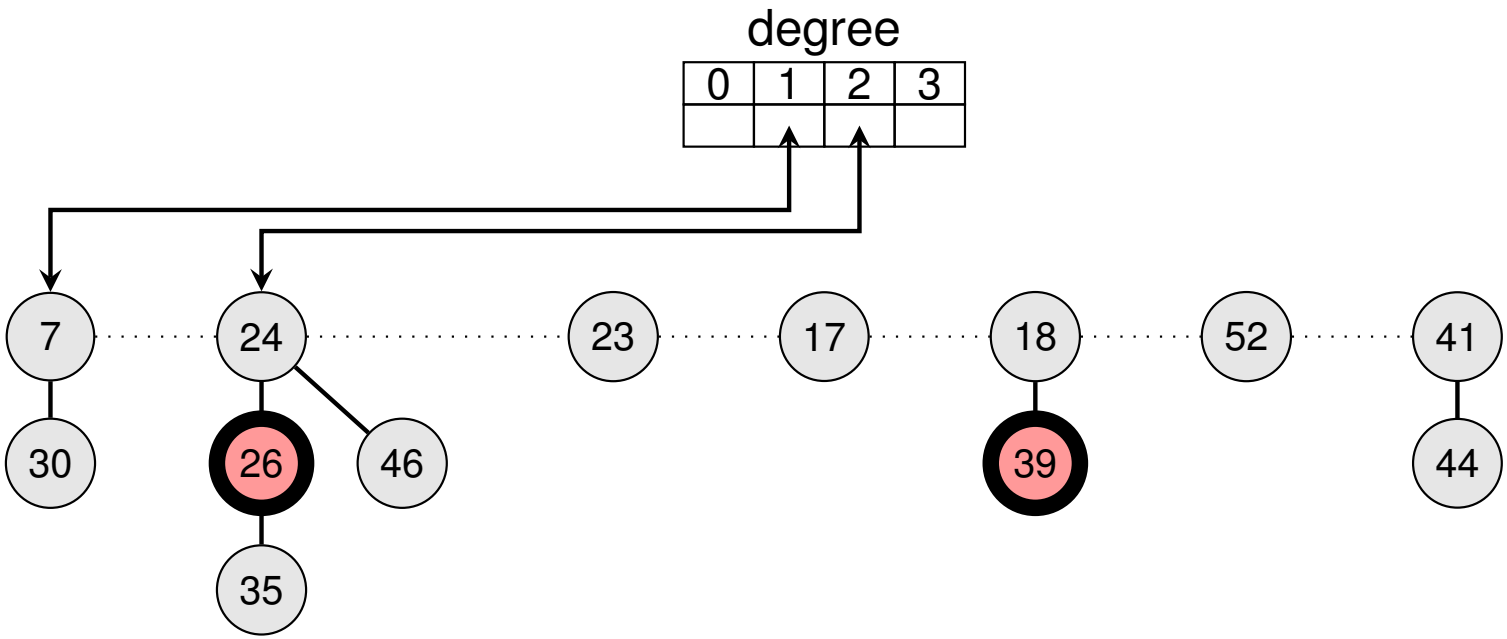
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

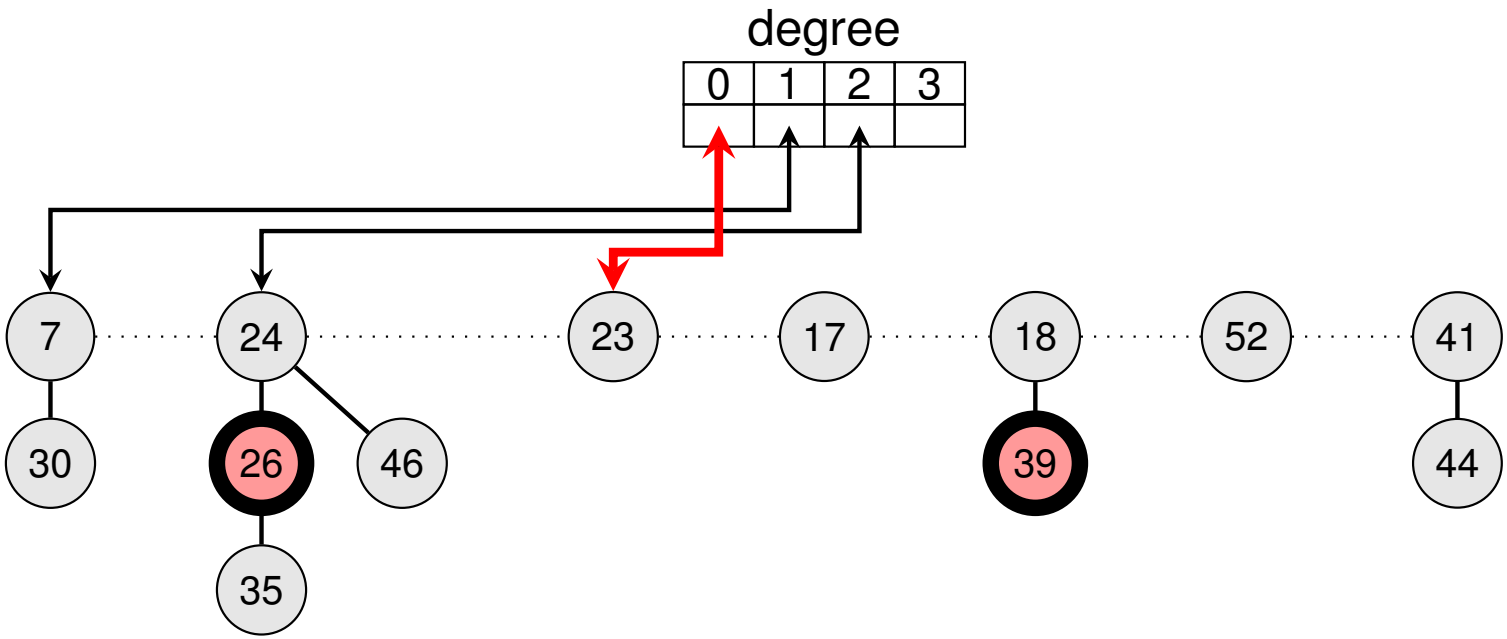
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

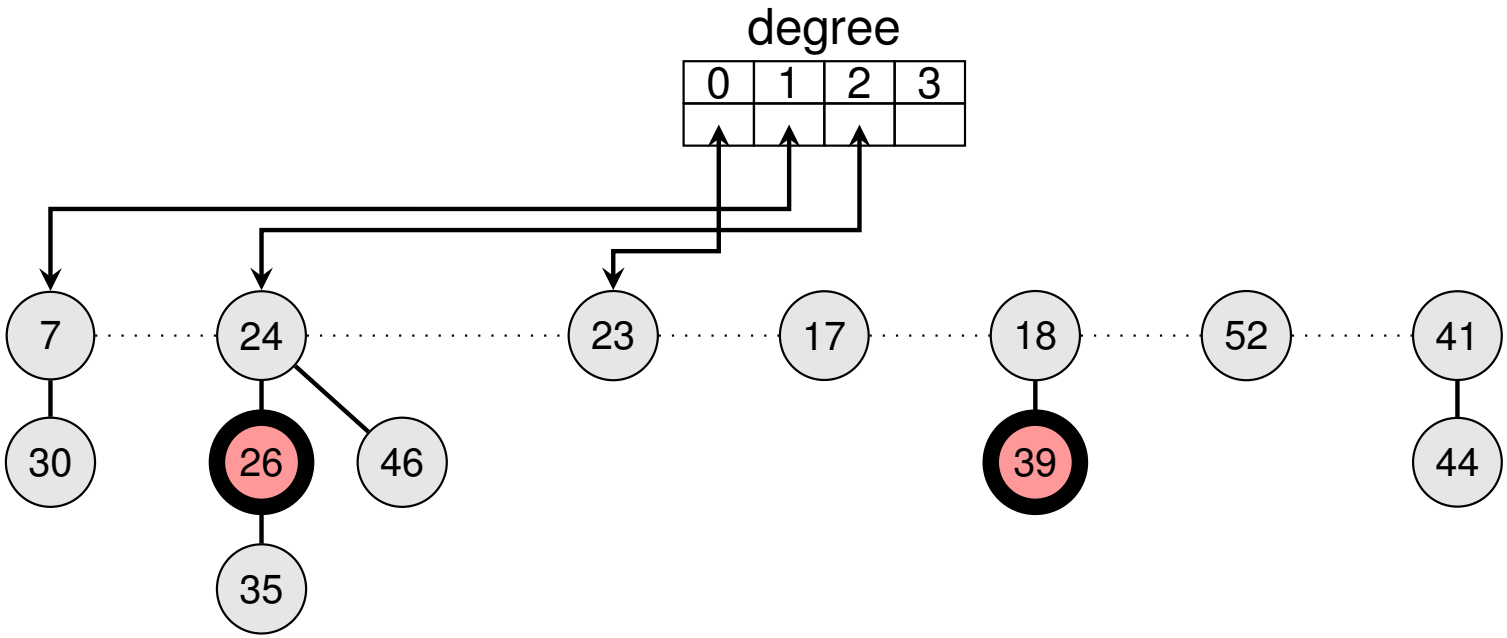
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

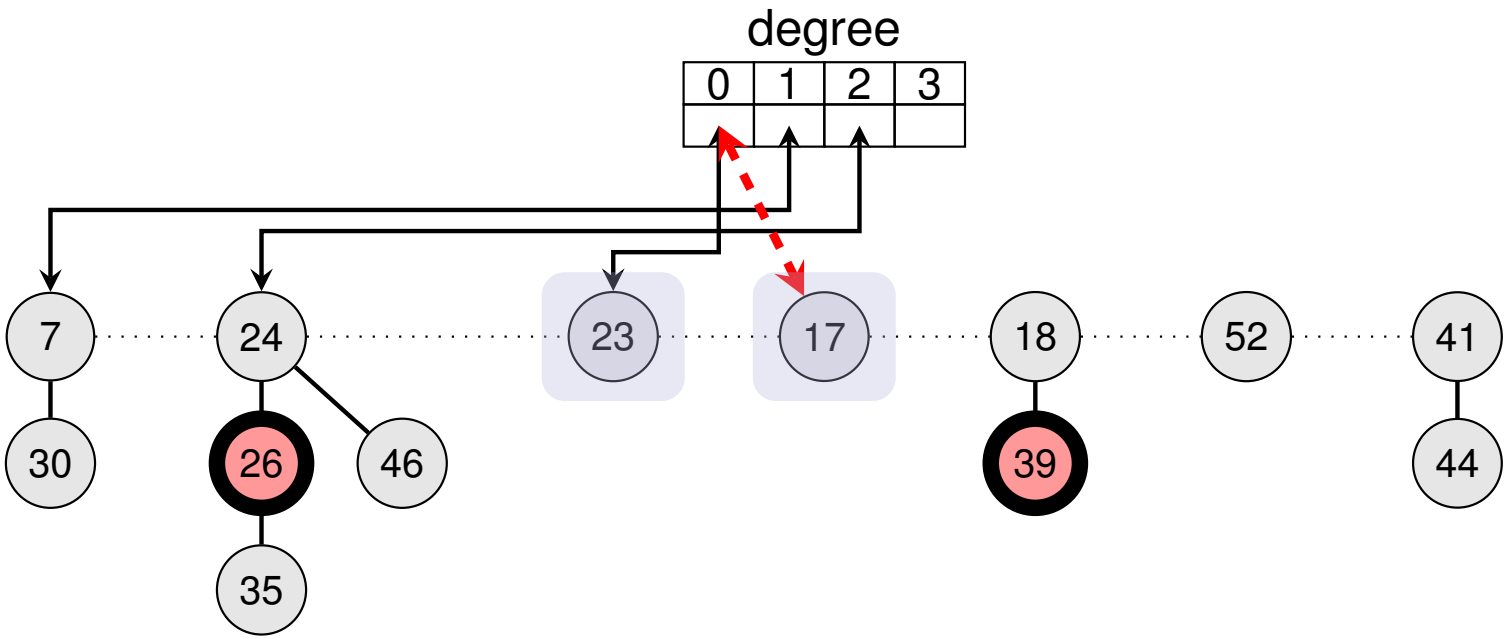
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

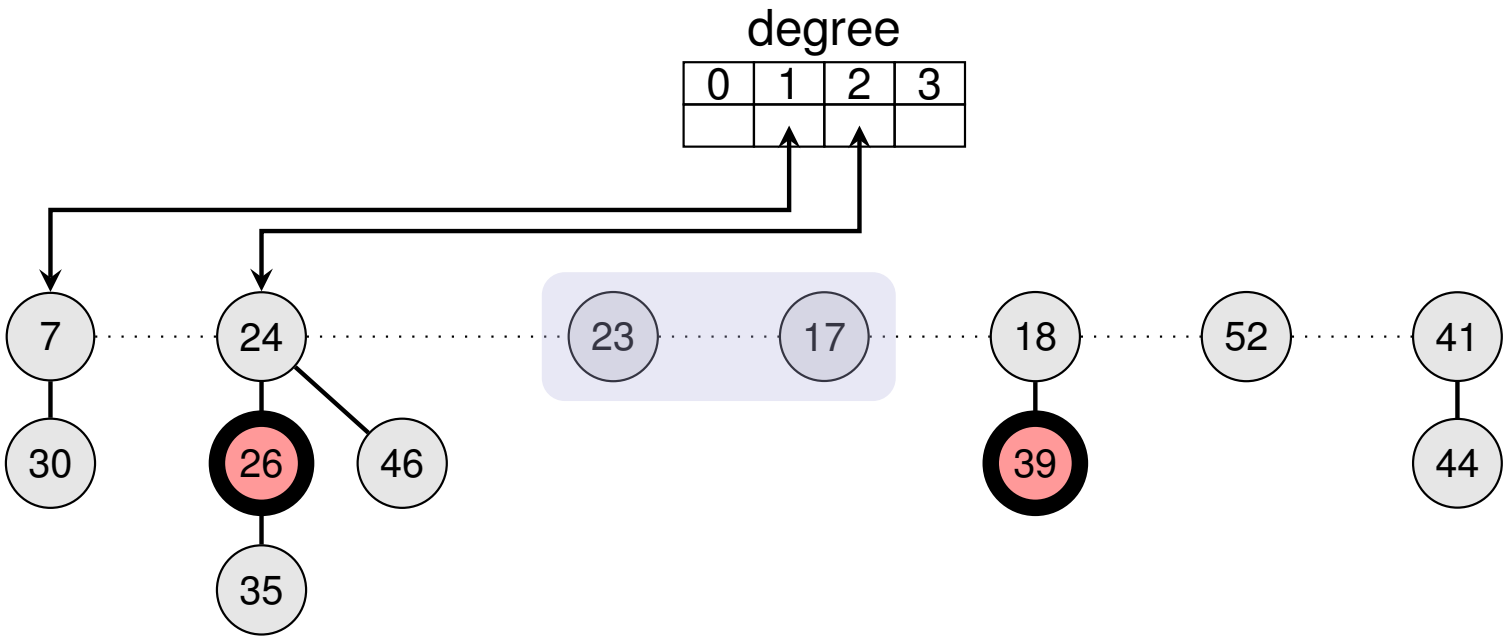
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

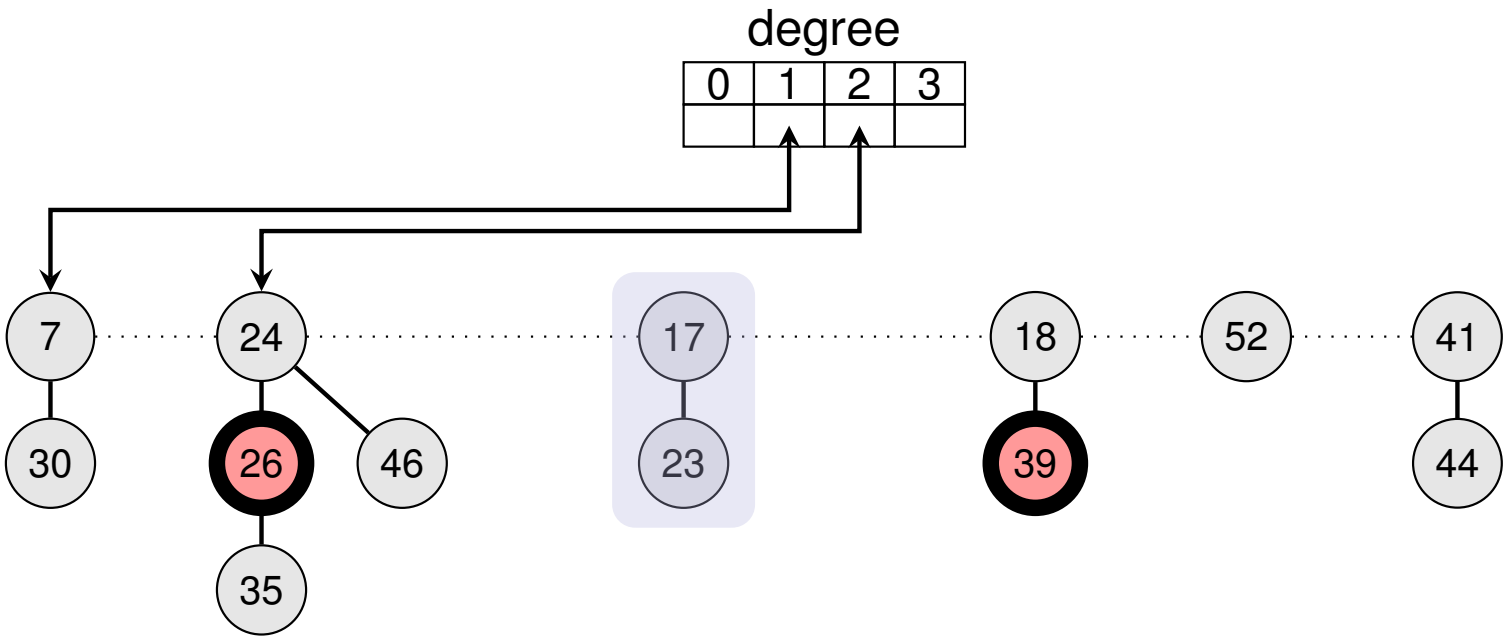
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

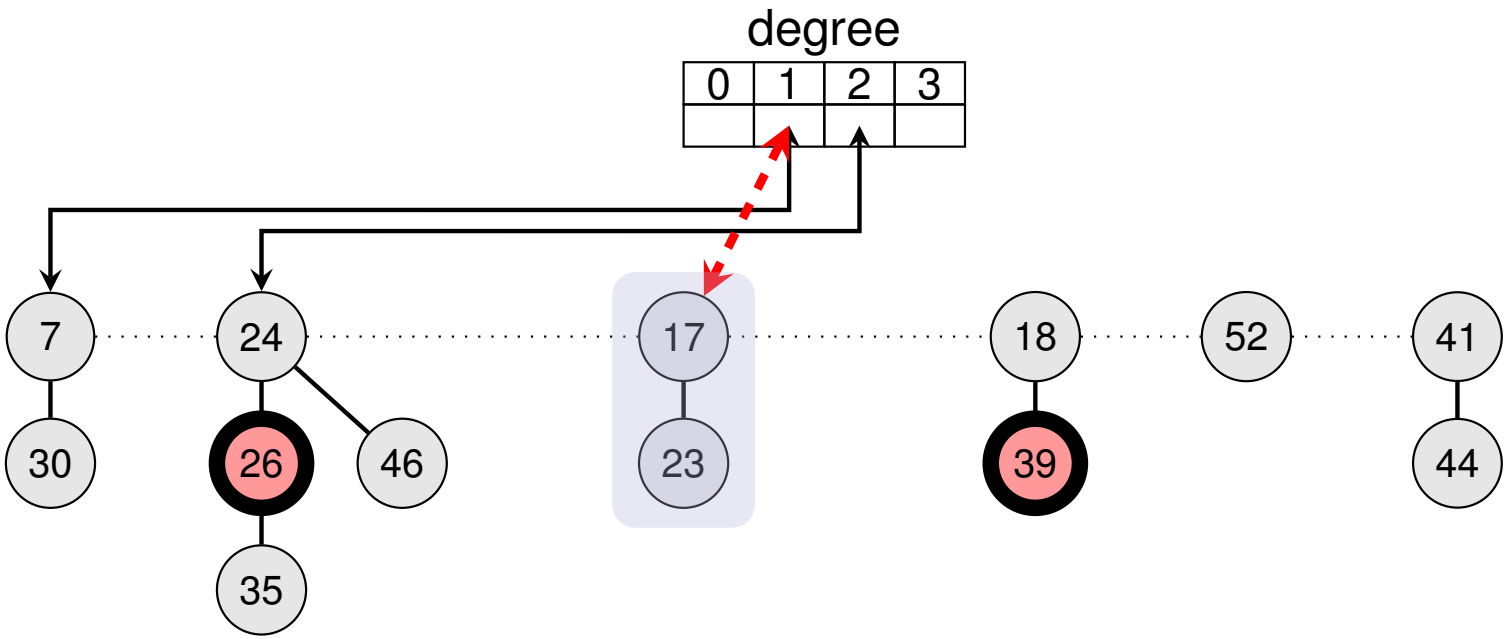
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

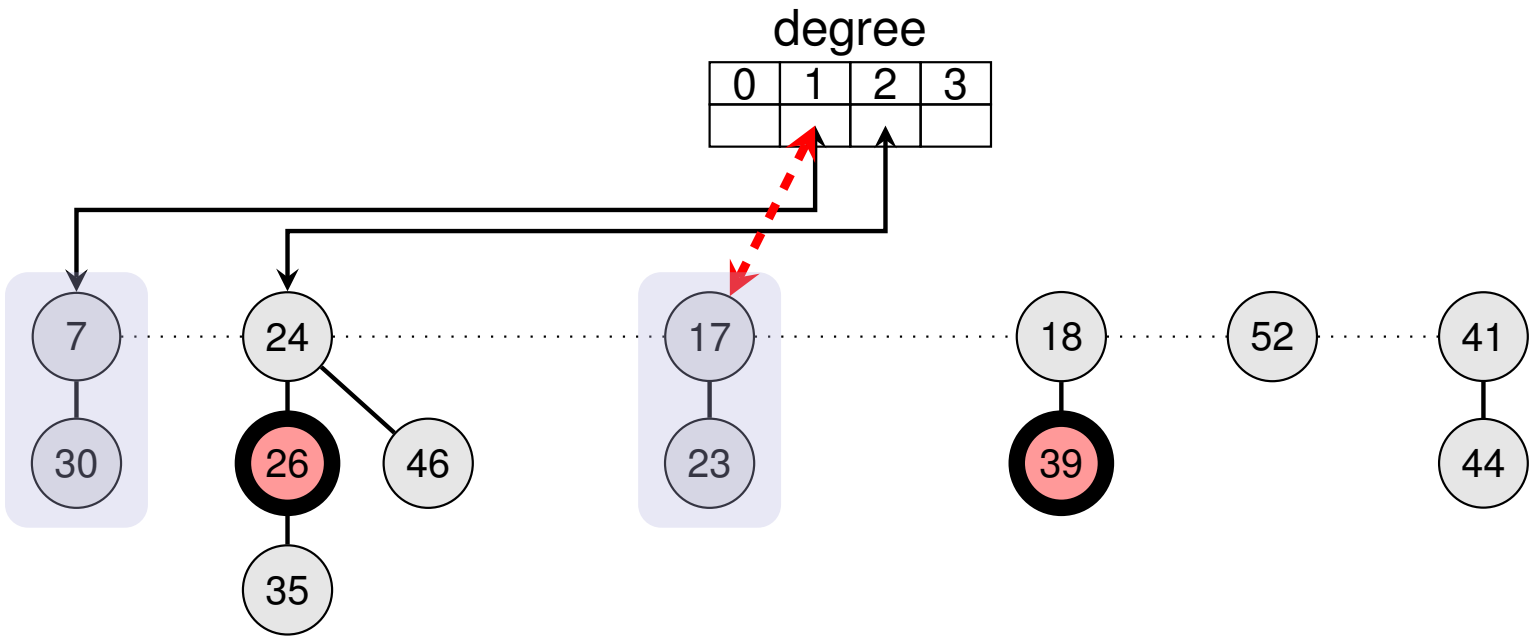
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

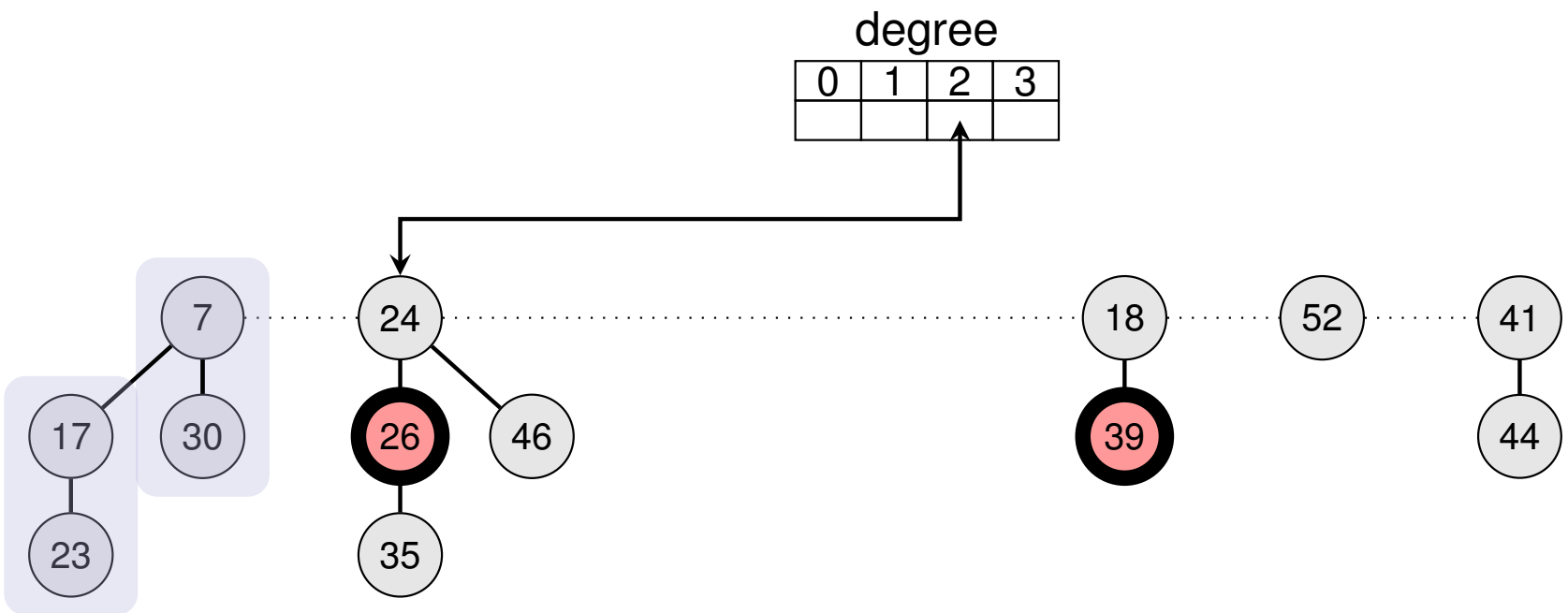
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

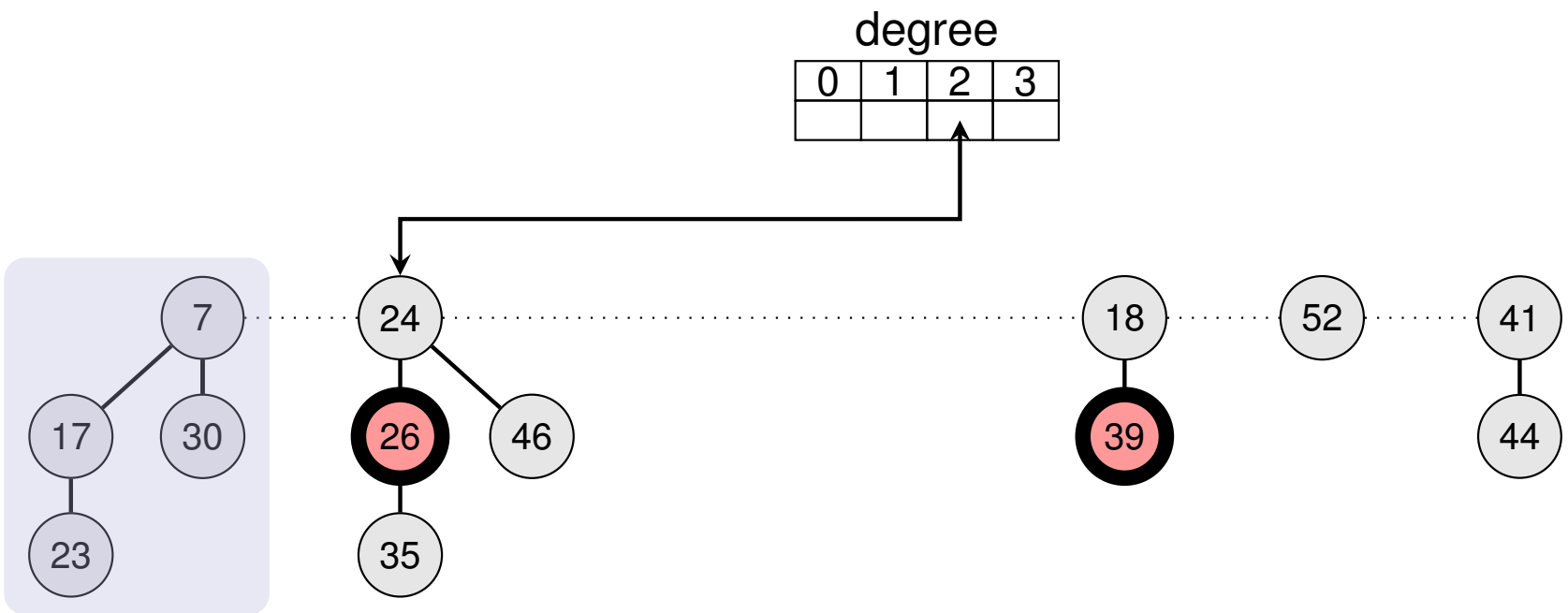
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

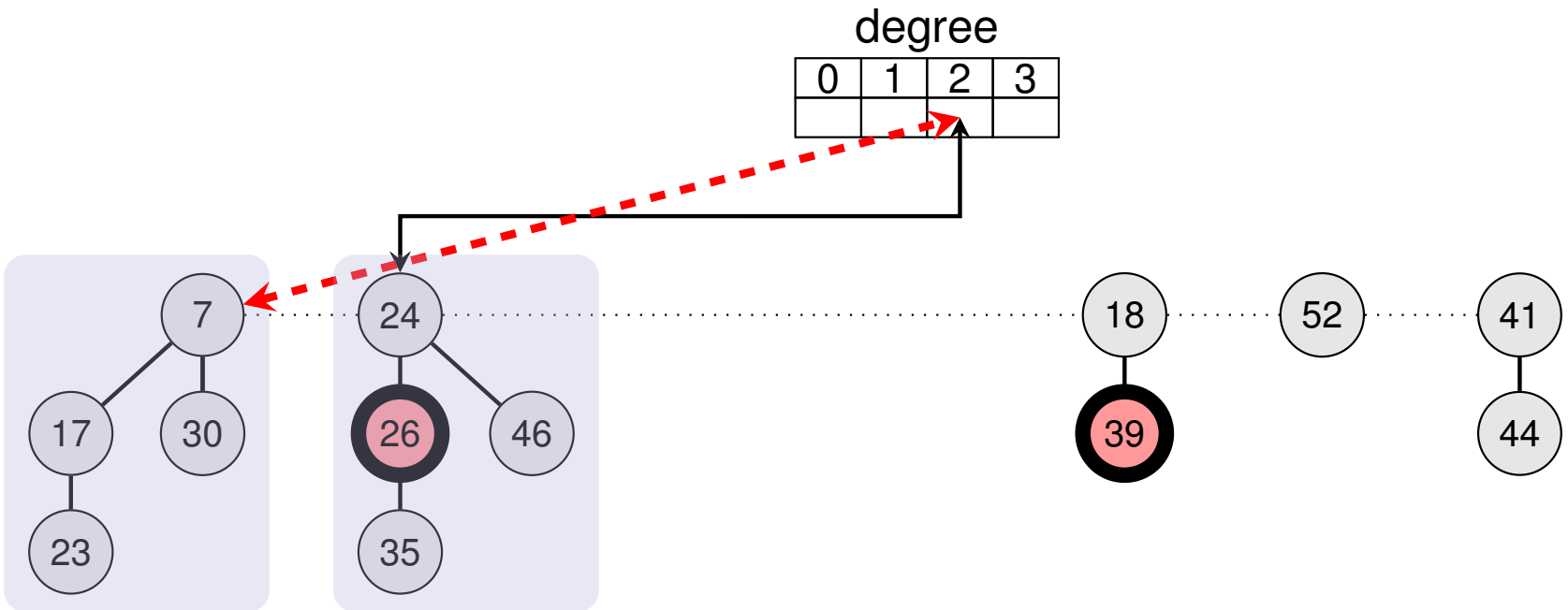
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



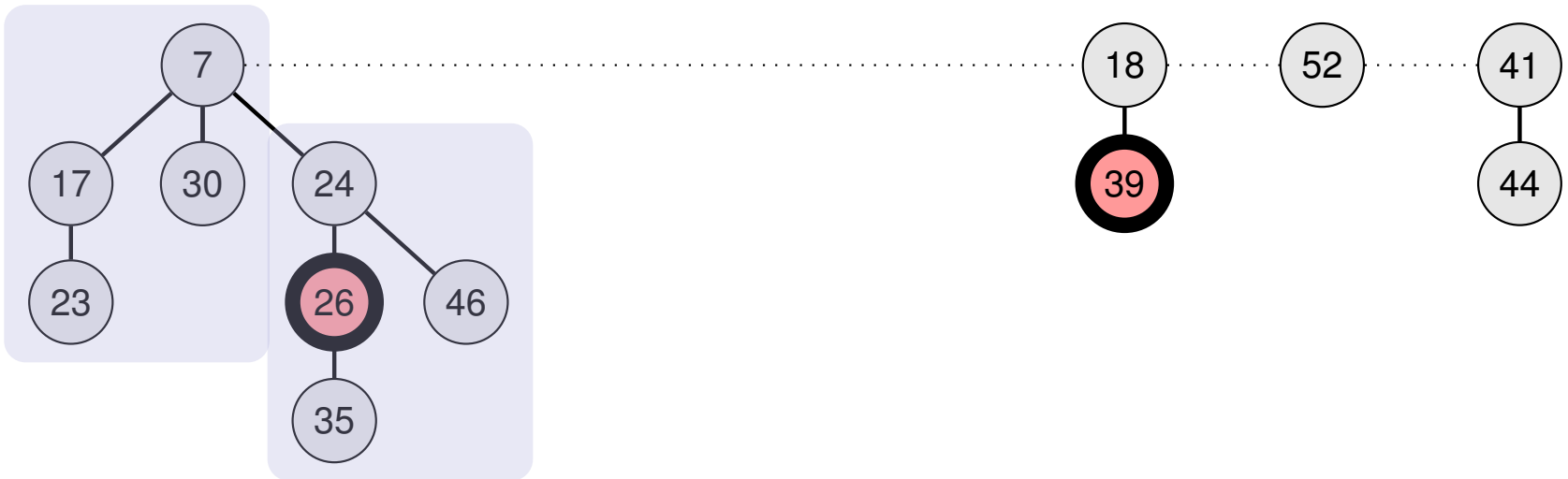
Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

degree

0	1	2	3



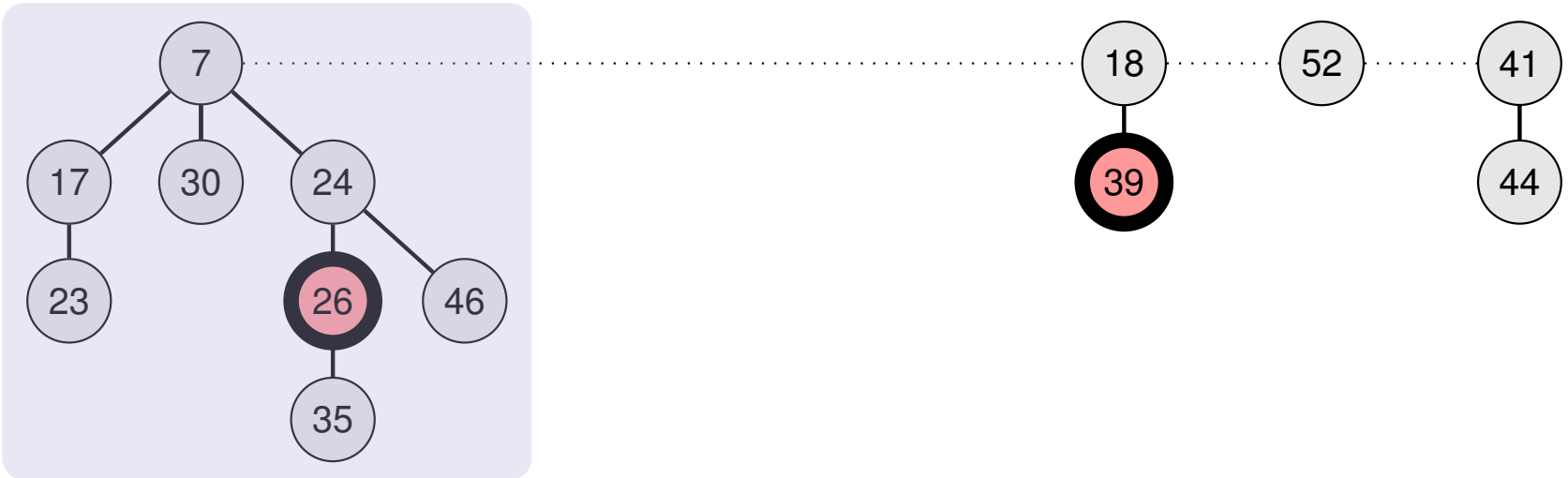
Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

degree

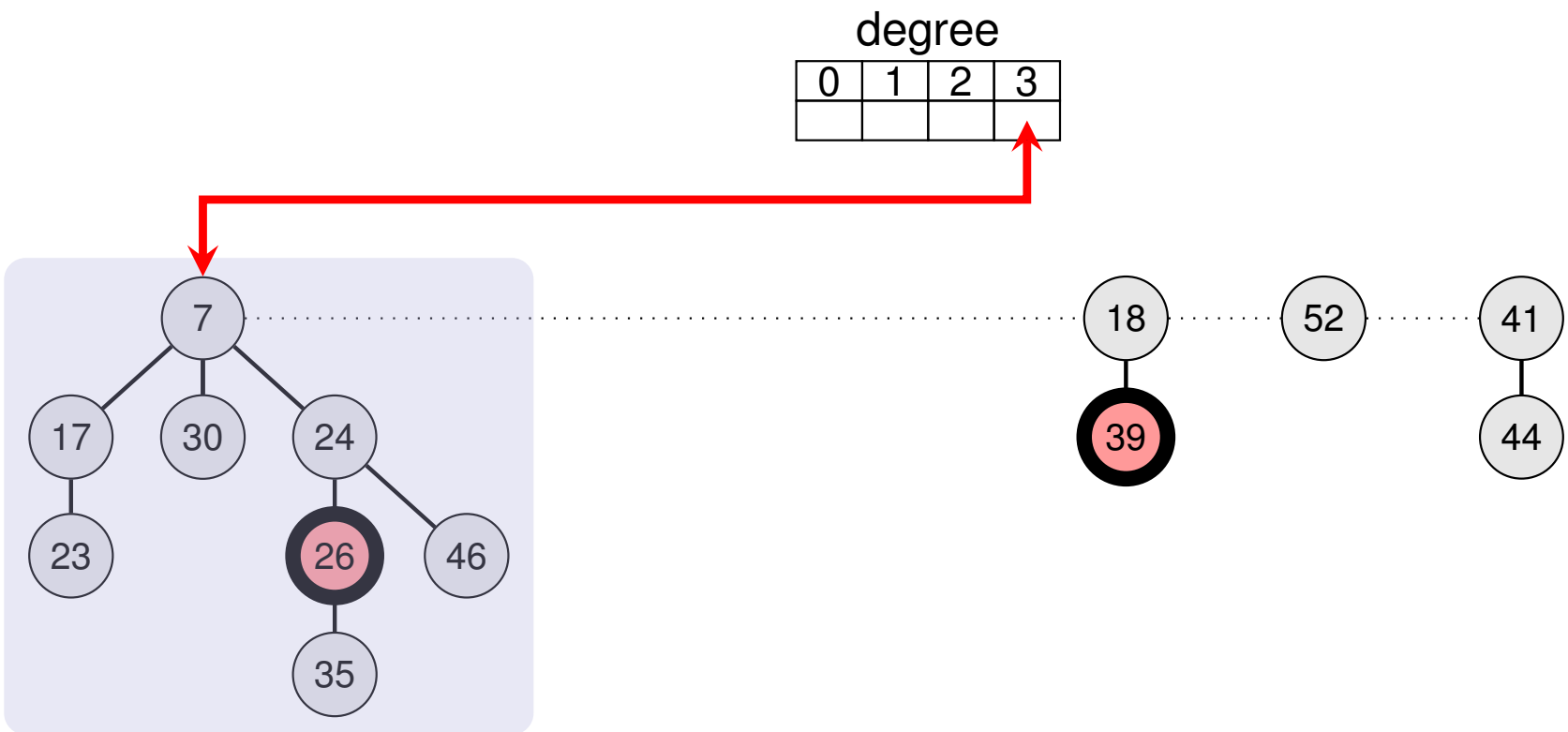
0	1	2	3



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

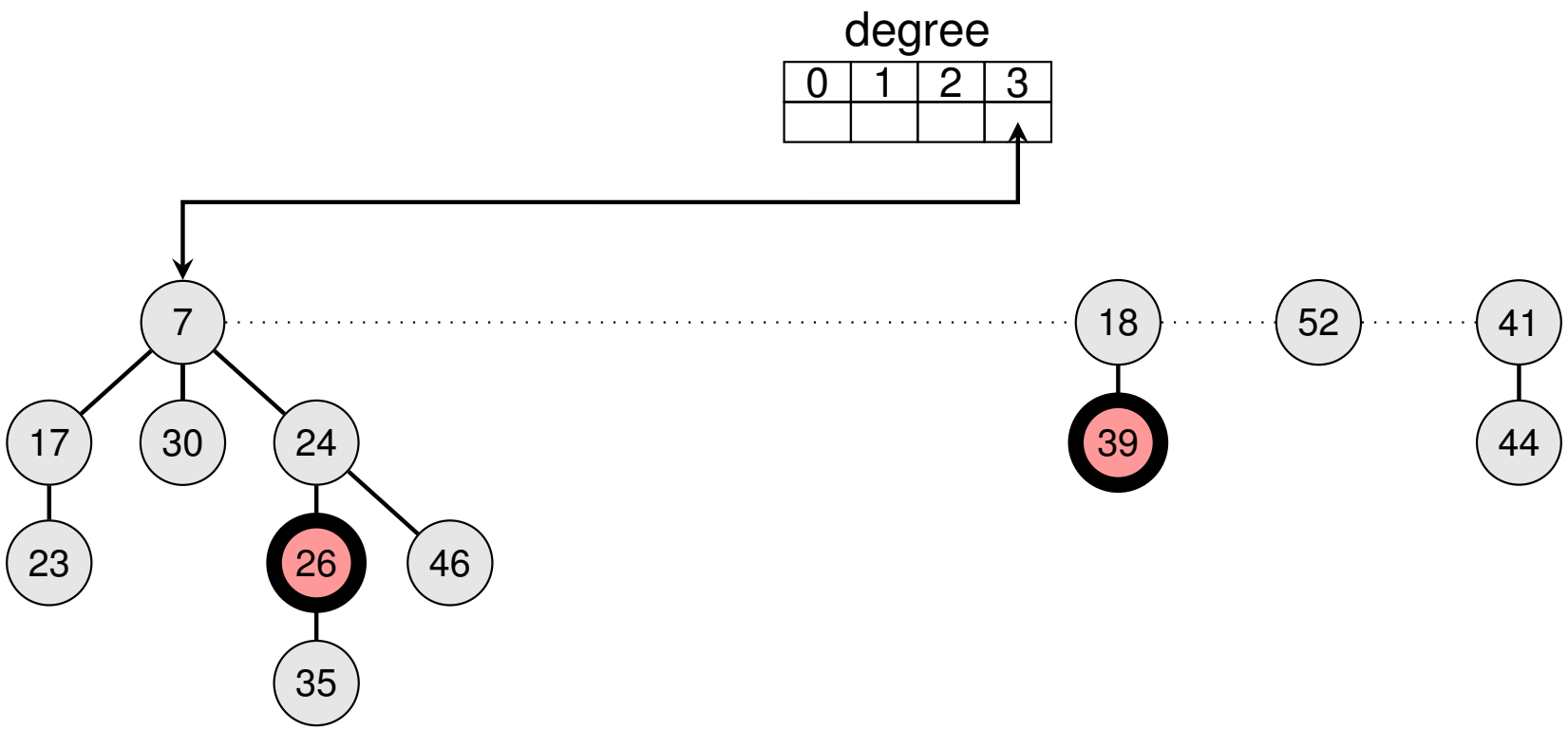
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

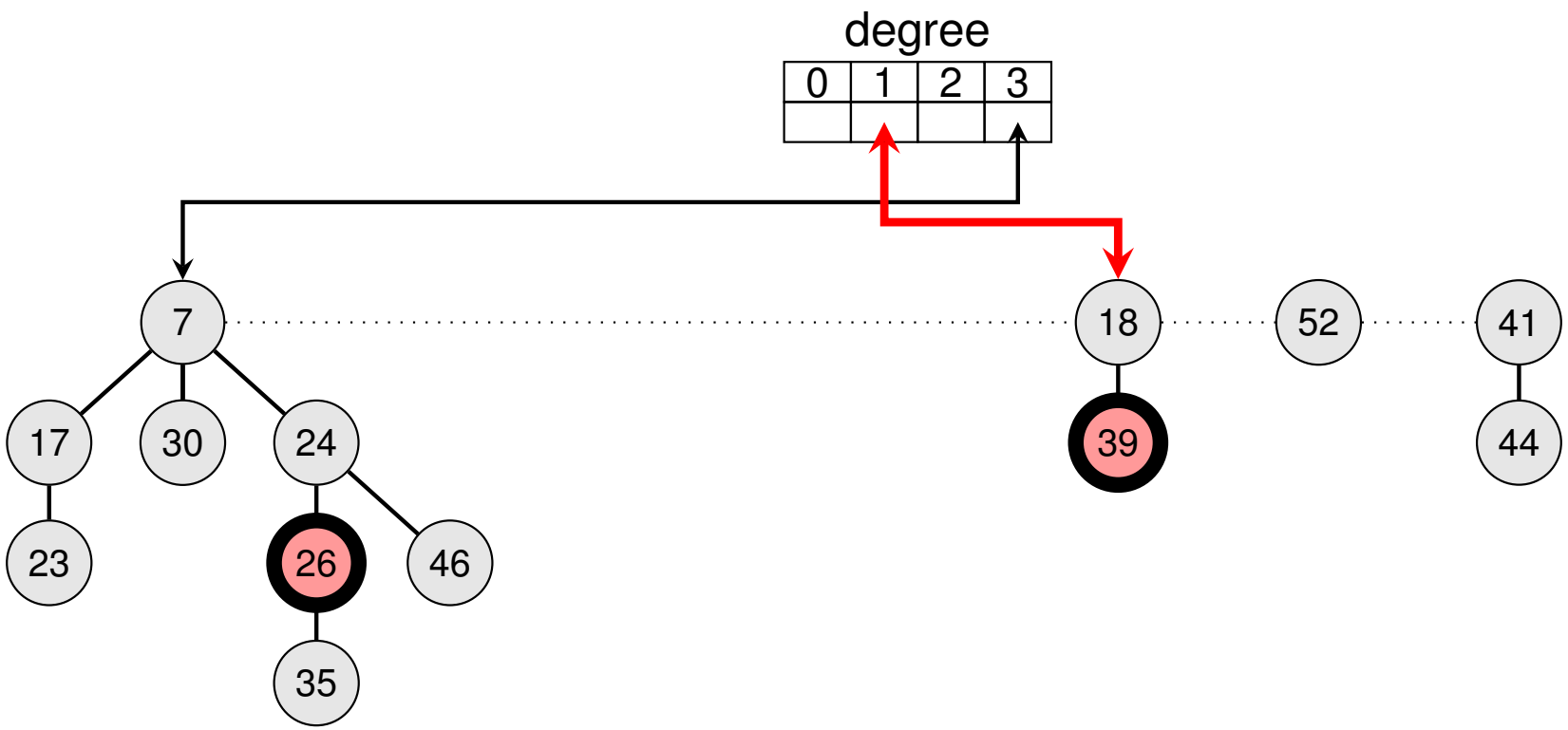
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

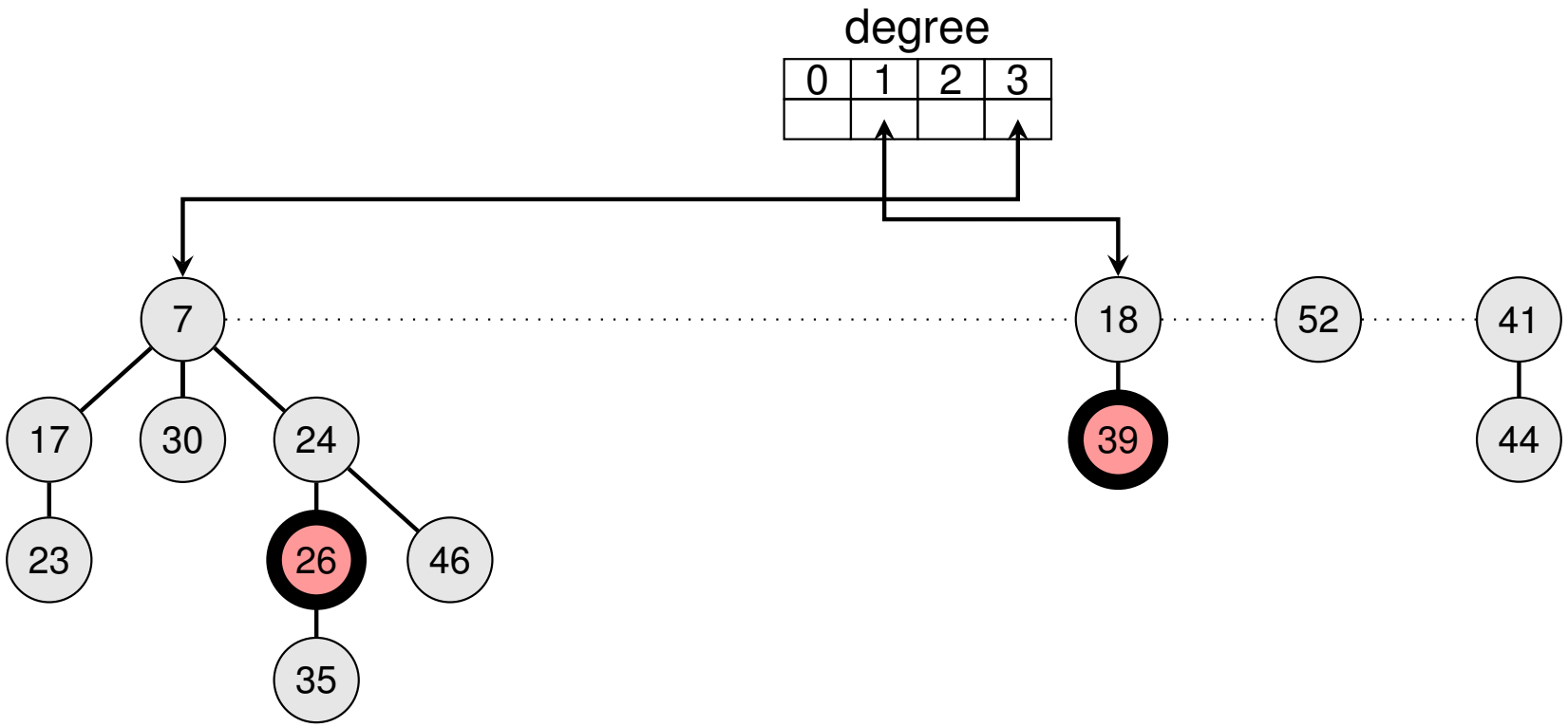
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

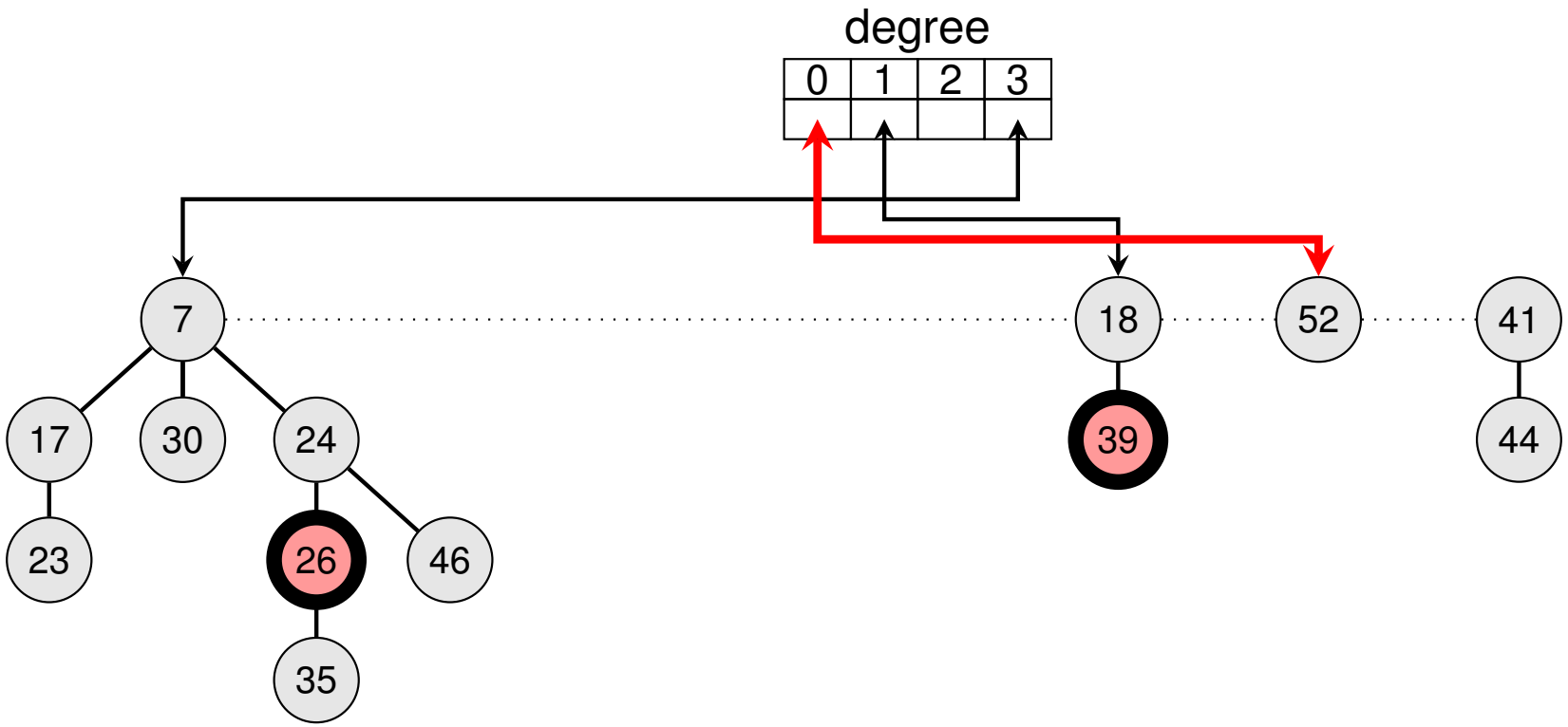
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

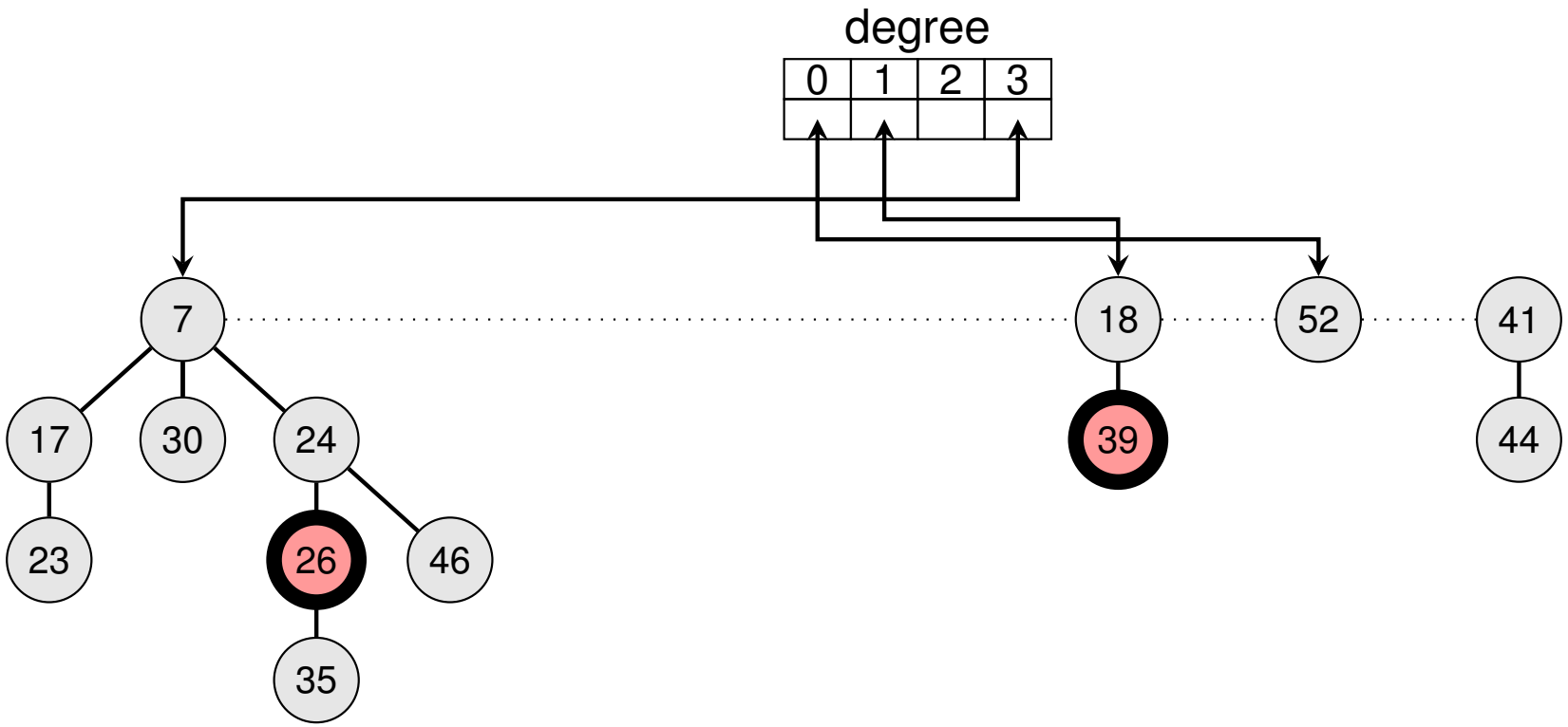
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

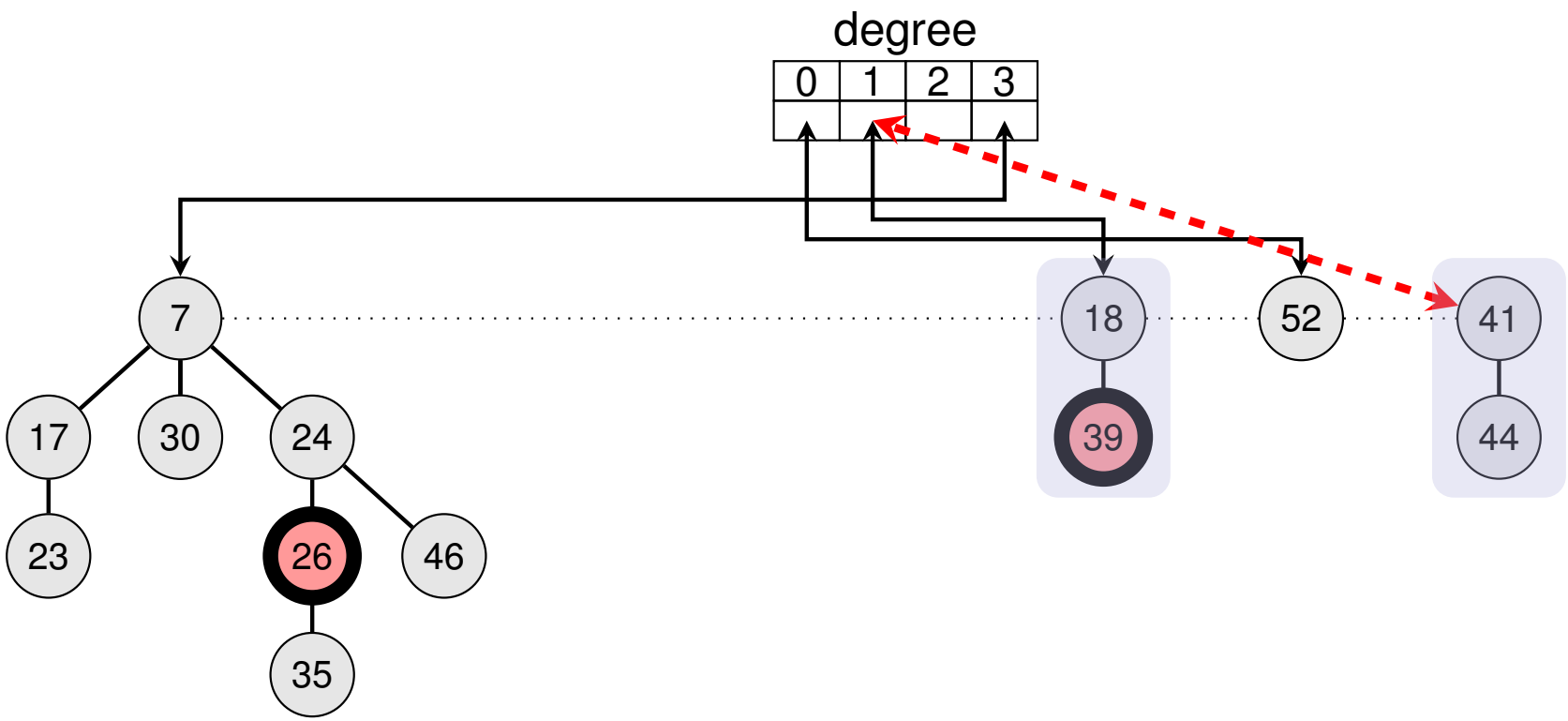
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

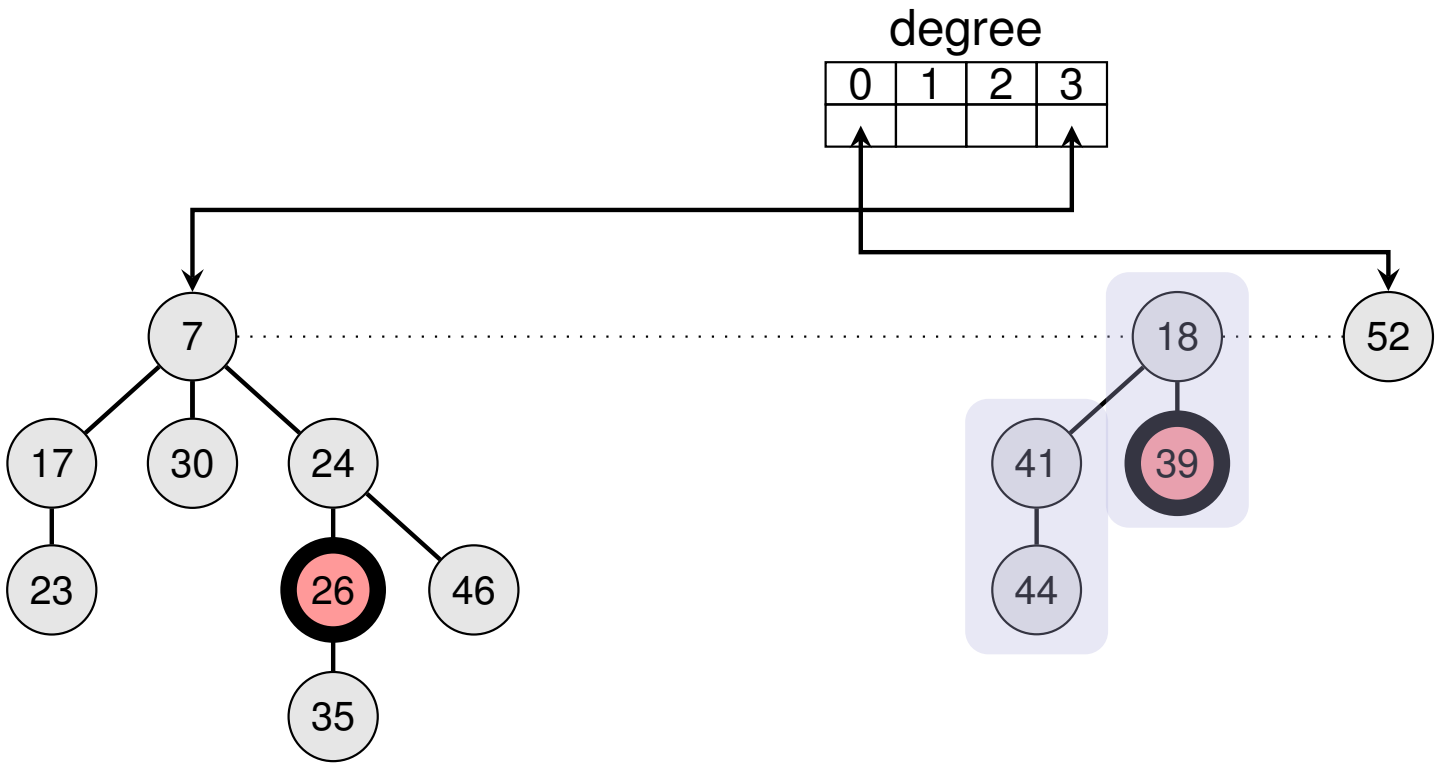
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

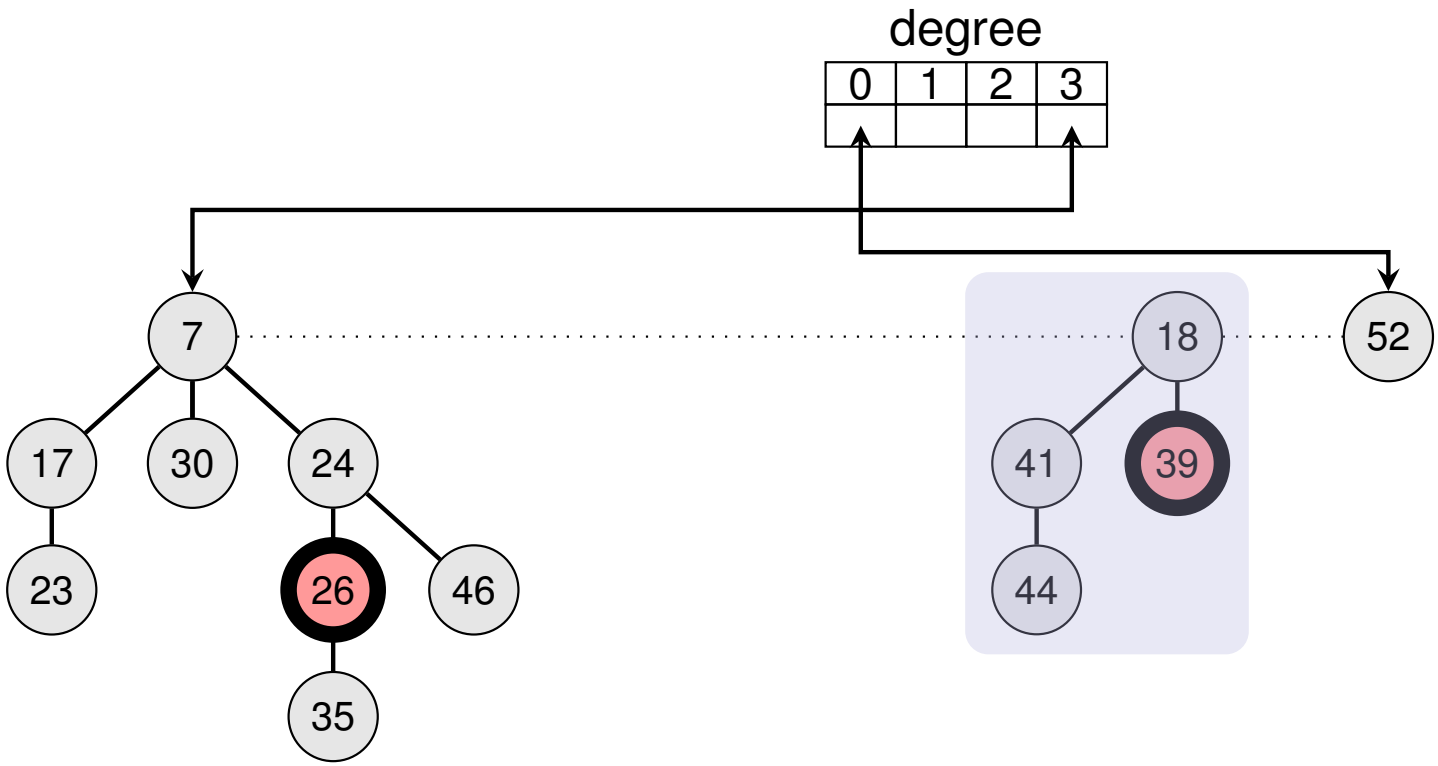
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

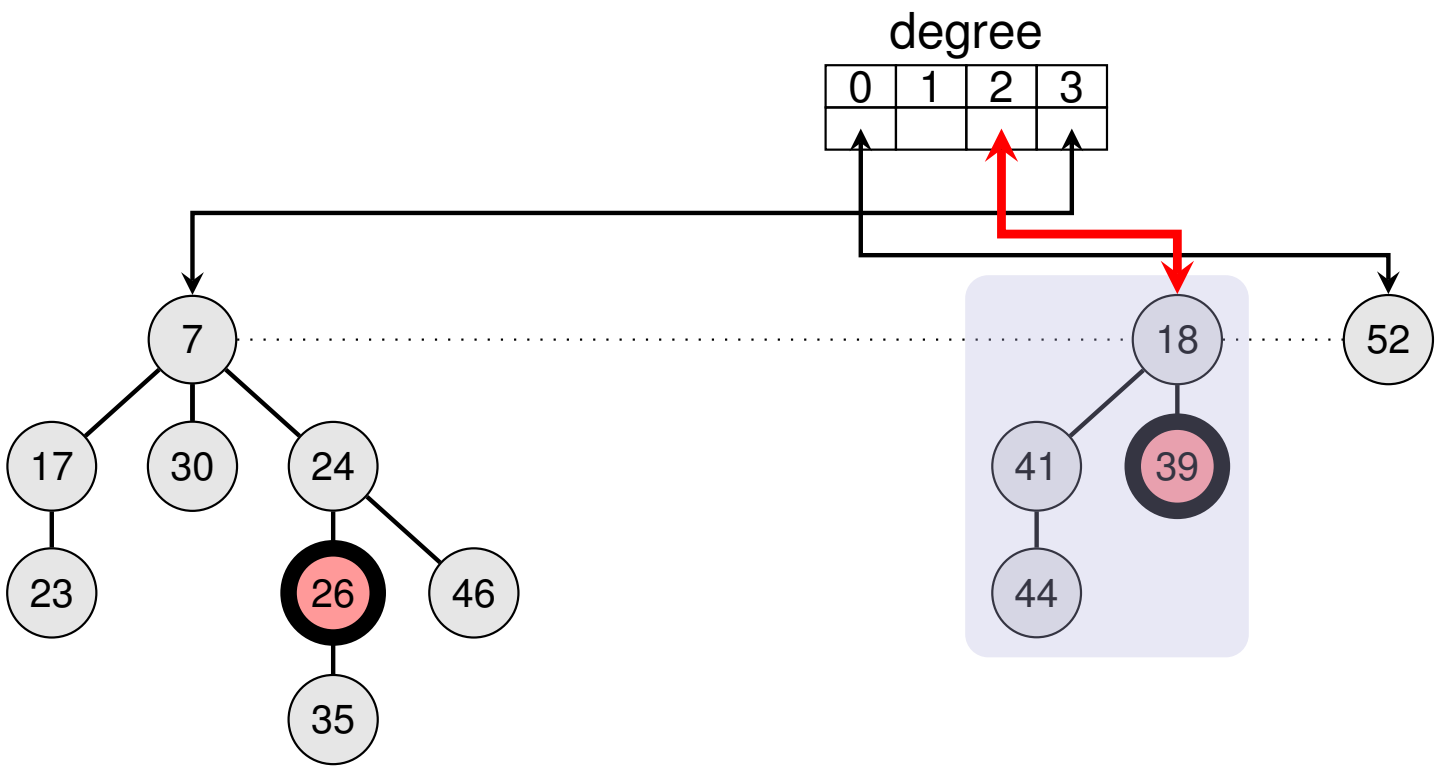
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

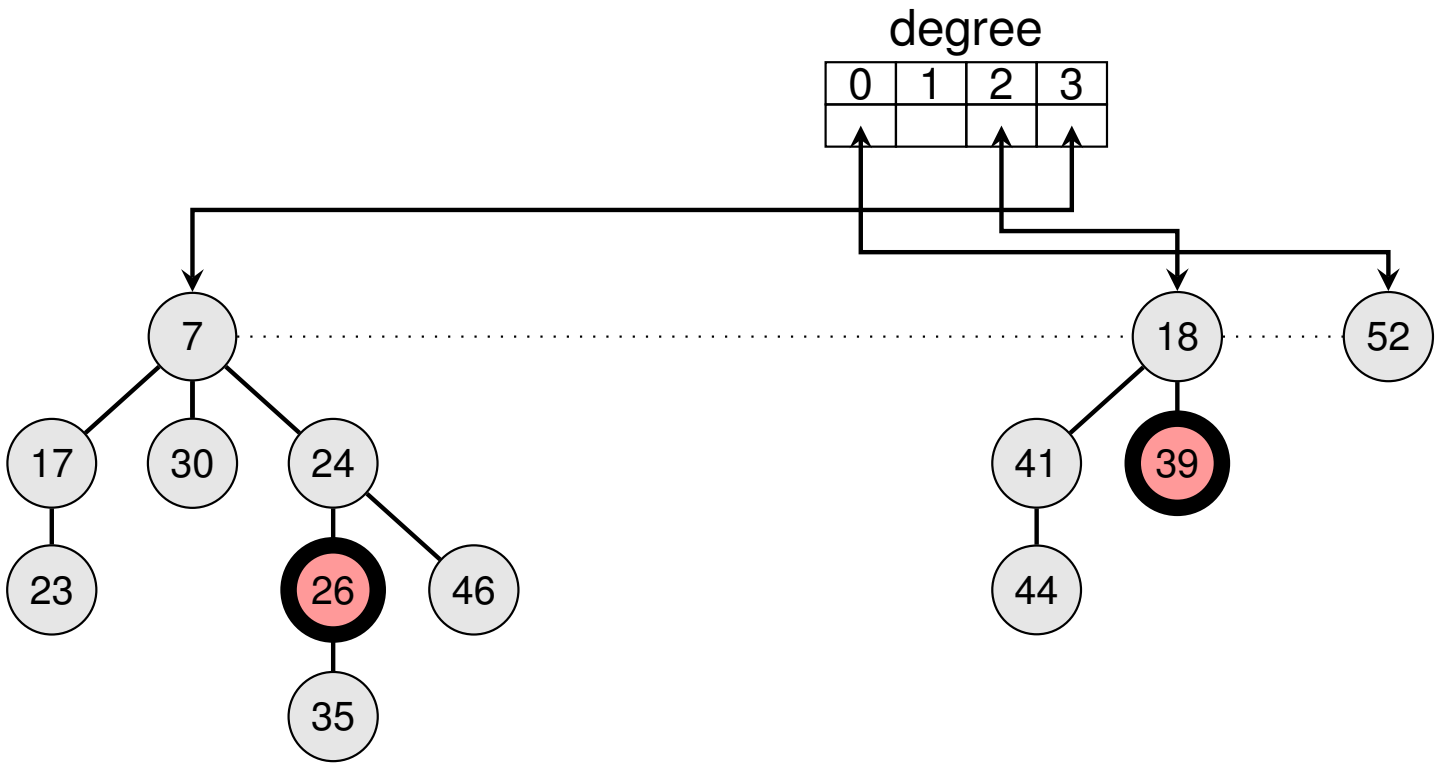
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

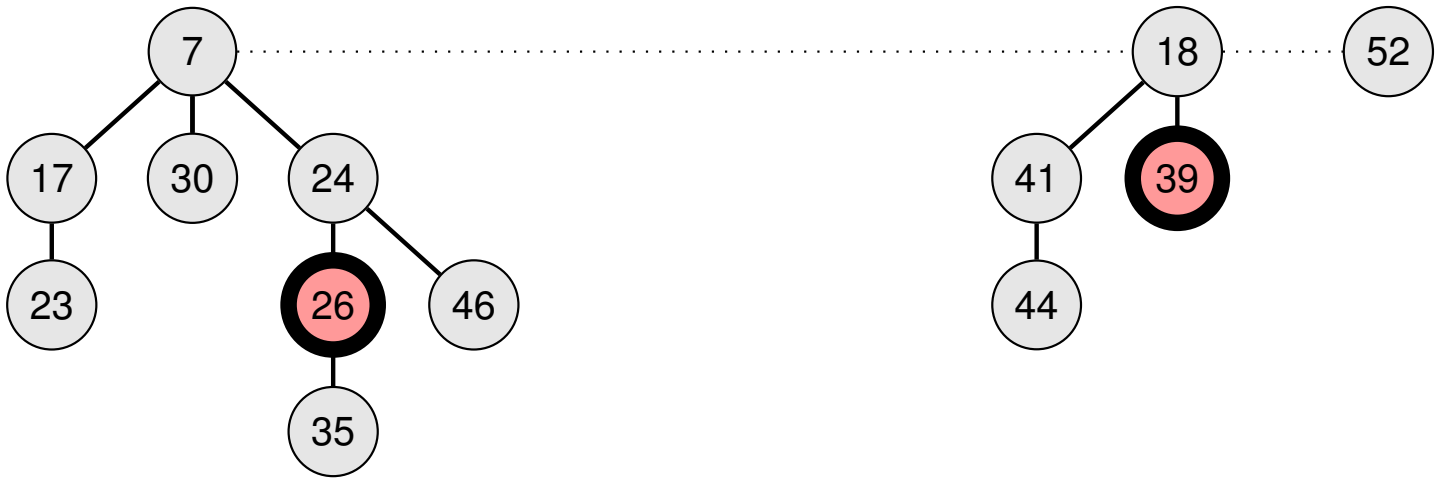
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

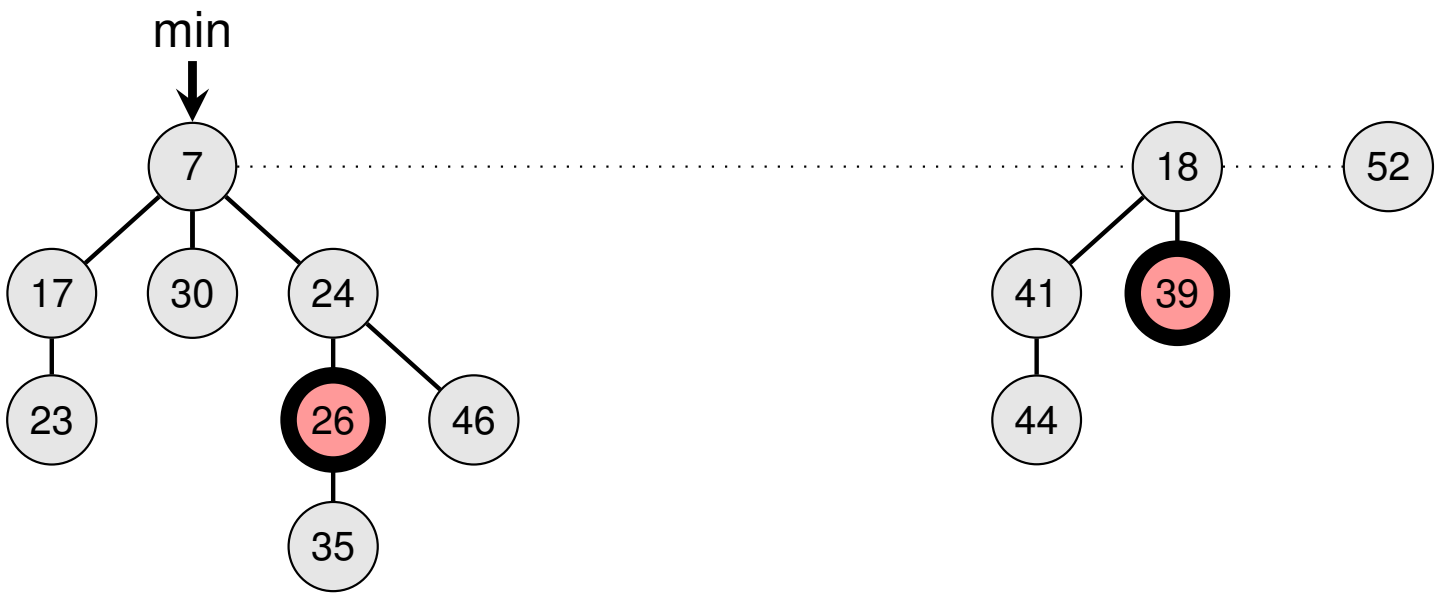
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

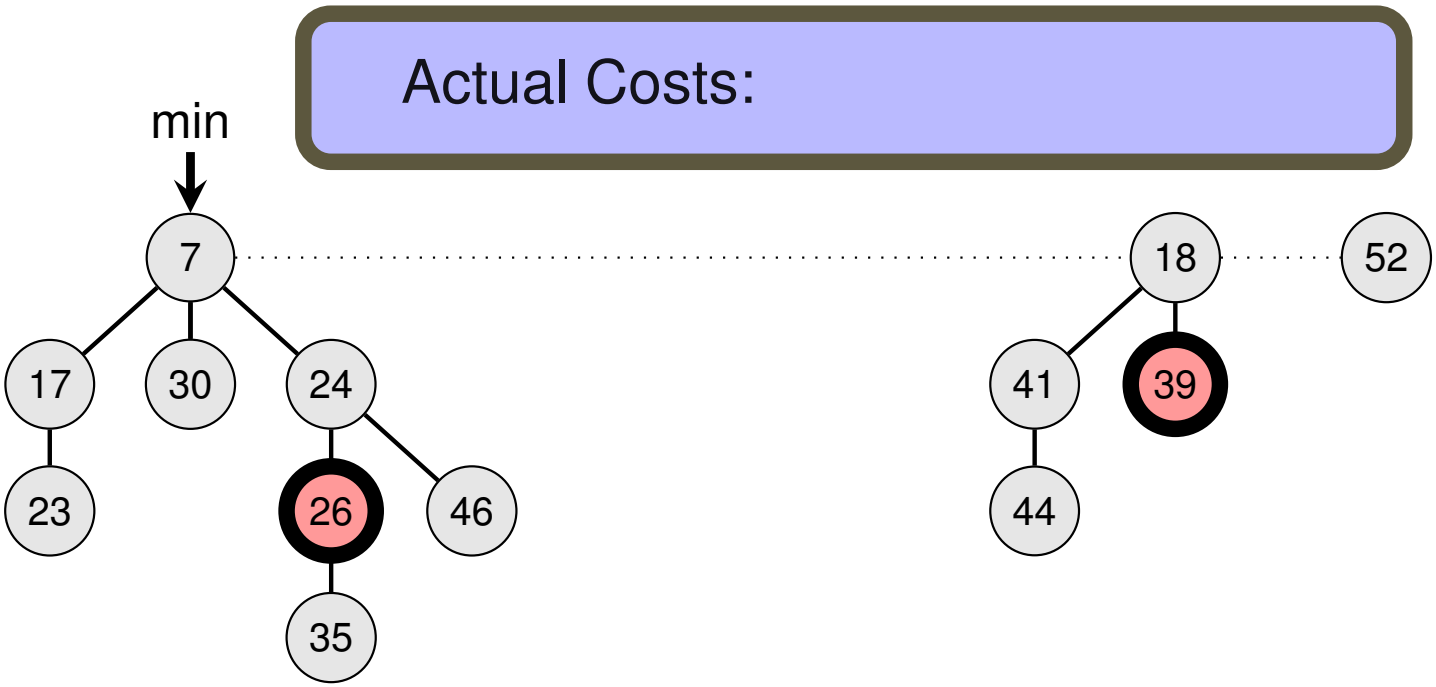
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓



Fibonacci Heap: EXTRACT-MIN

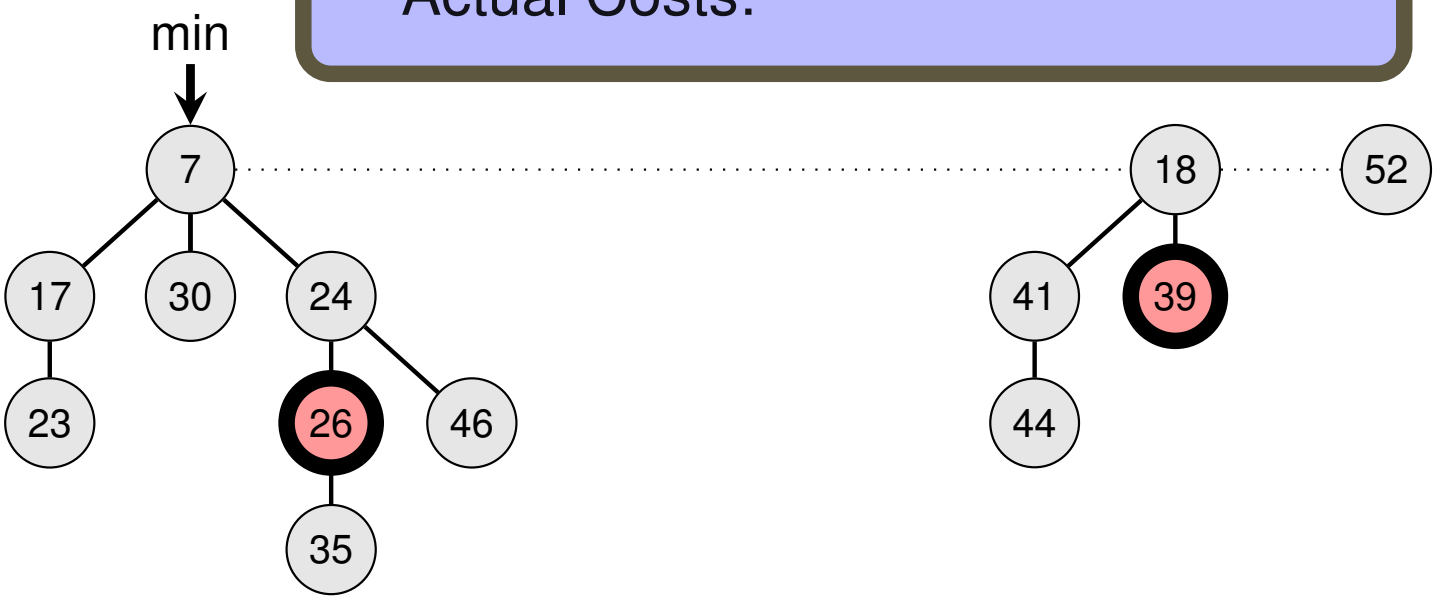
EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size n

Actual Costs:



Fibonacci Heap: EXTRACT-MIN

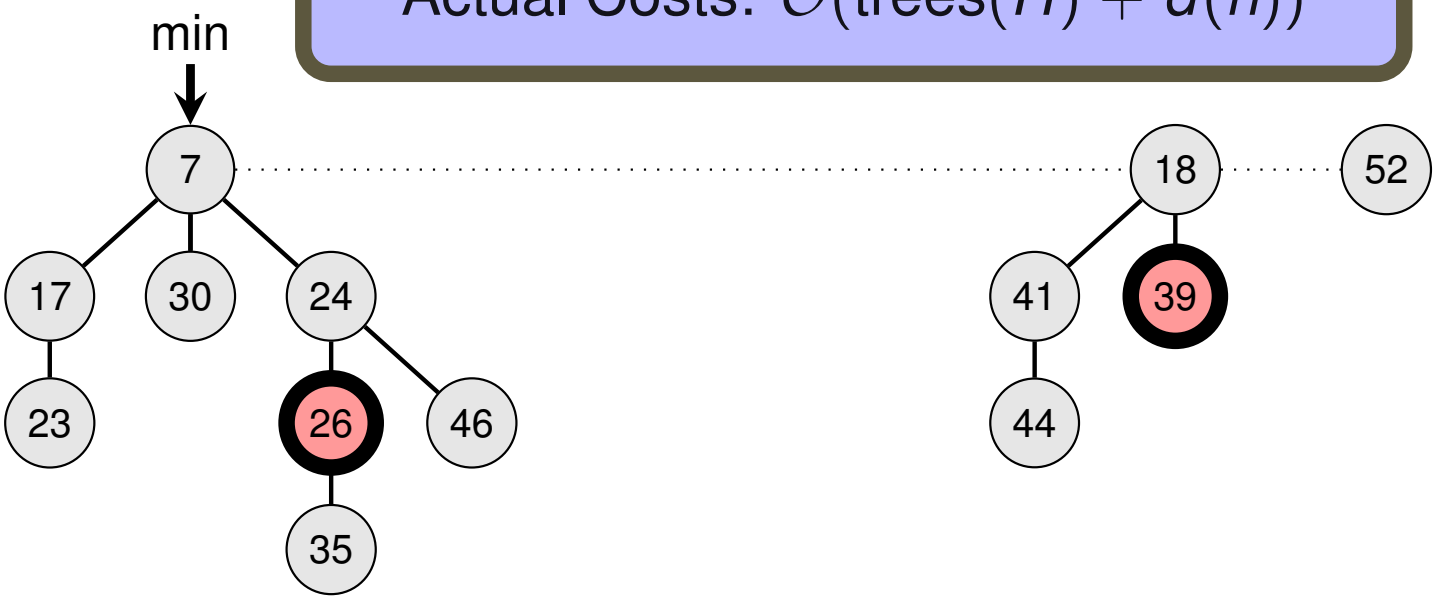
EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size n

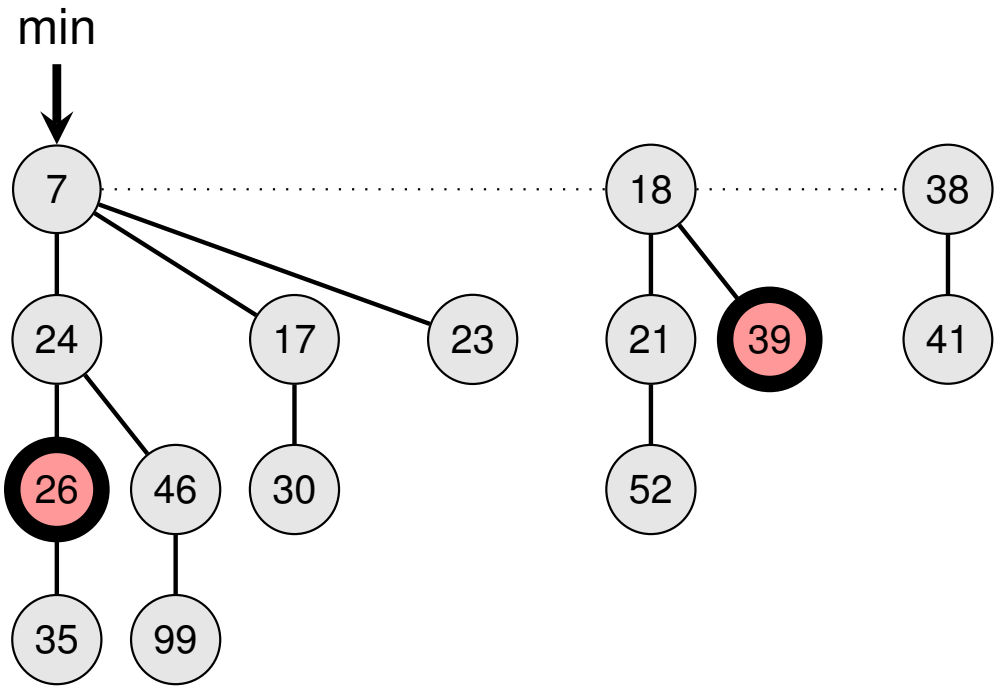
Actual Costs: $\mathcal{O}(\text{trees}(H) + d(n))$



Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

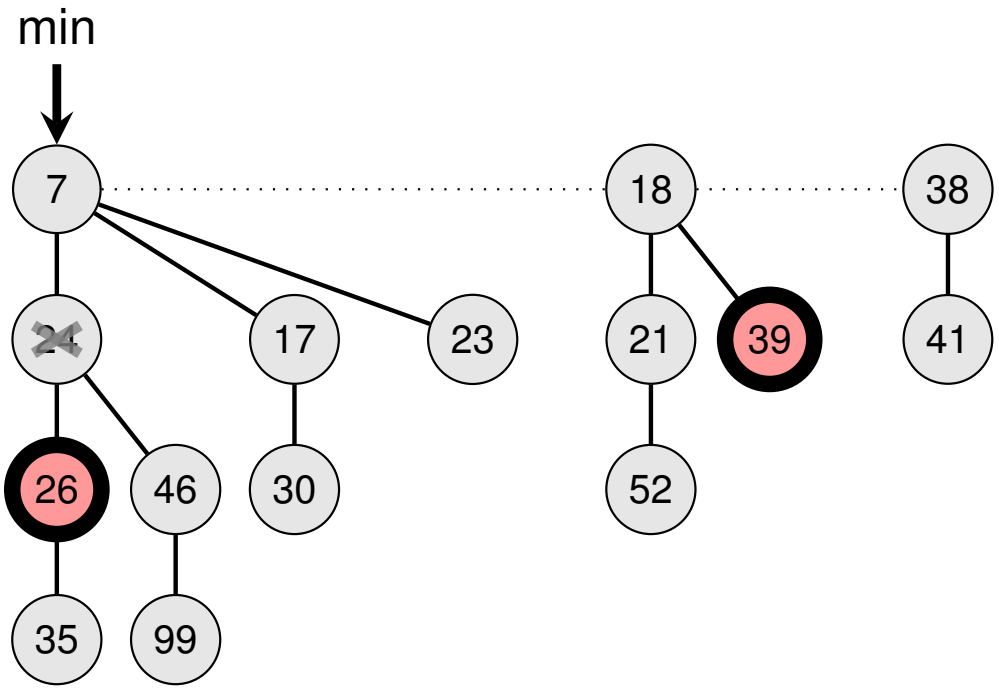
- Decrease the key of x (given by a pointer)



Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)

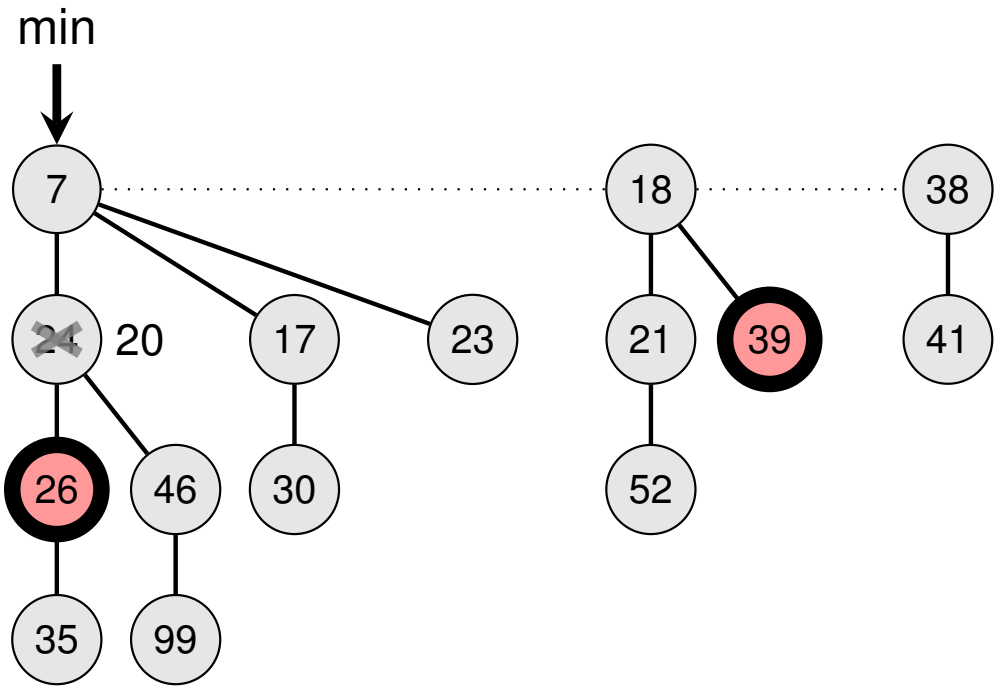


1. DECREASE-KEY 24 $\rightsquigarrow$ 20

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)

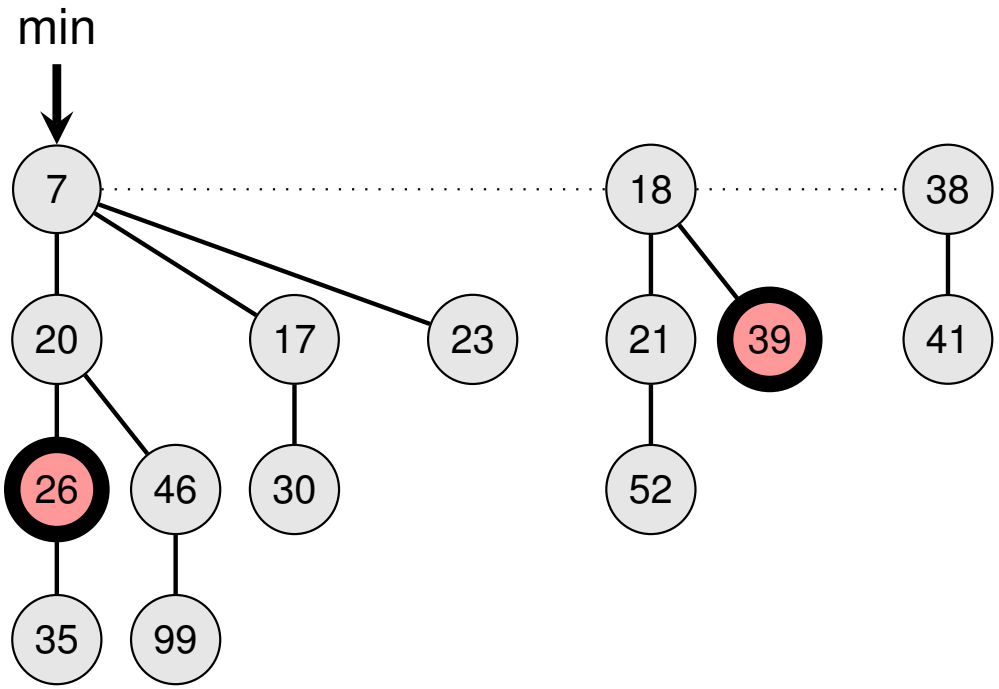


1. DECREASE-KEY 24 $\rightsquigarrow$ 20

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated

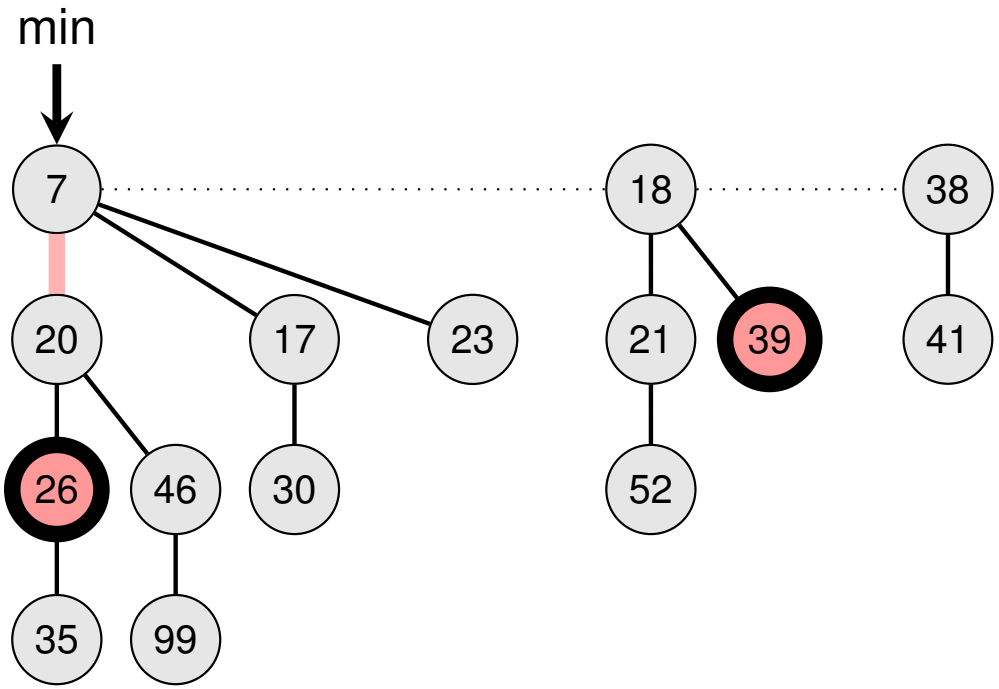


1. DECREASE-KEY 24 $\rightsquigarrow$ 20

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated

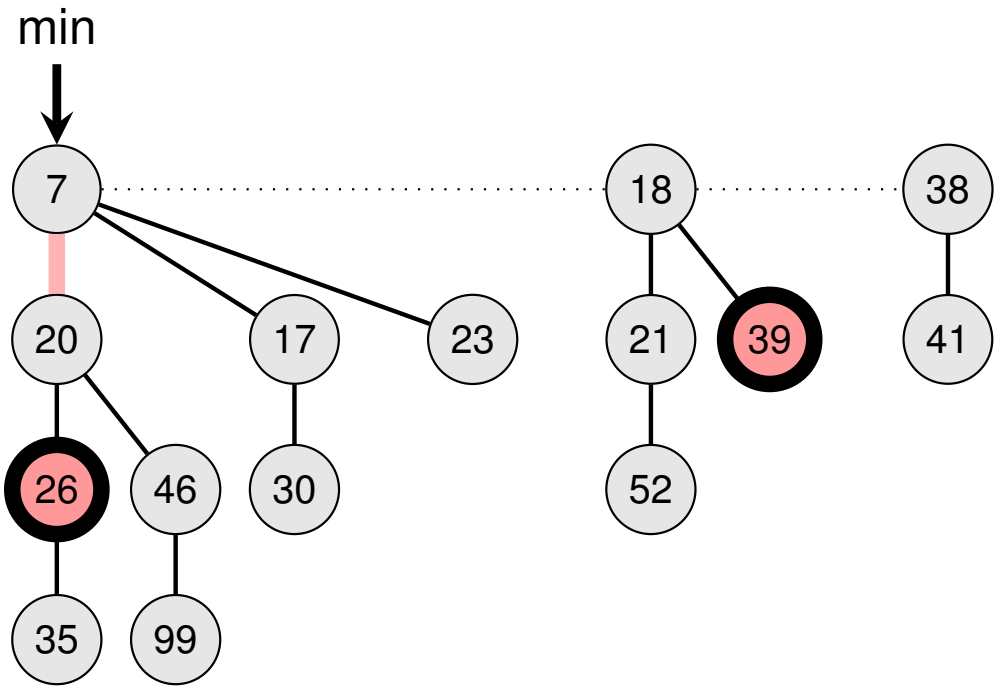


1. DECREASE-KEY 24 $\rightsquigarrow$ 20

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not

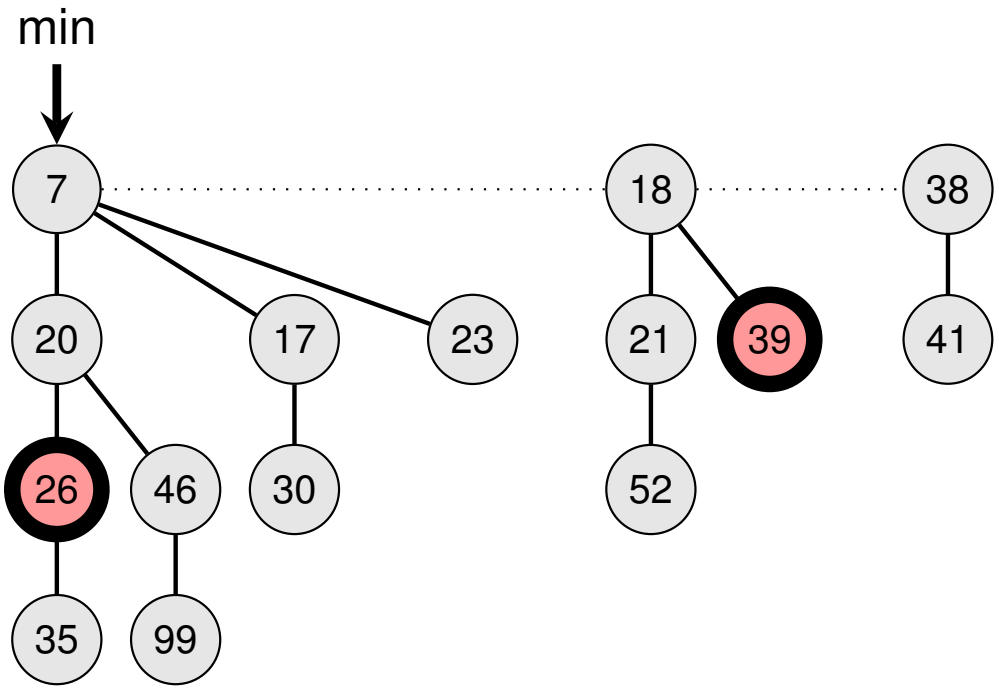


1. DECREASE-KEY 24 $\rightsquigarrow$ 20

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.

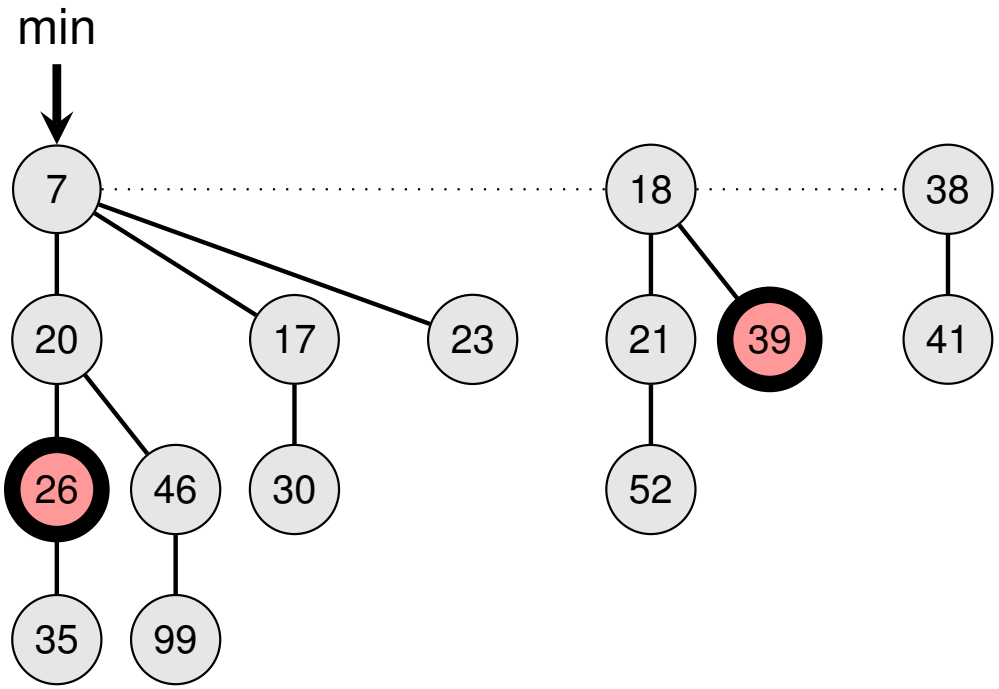


1. DECREASE-KEY 24 $\rightsquigarrow$ 20

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise,

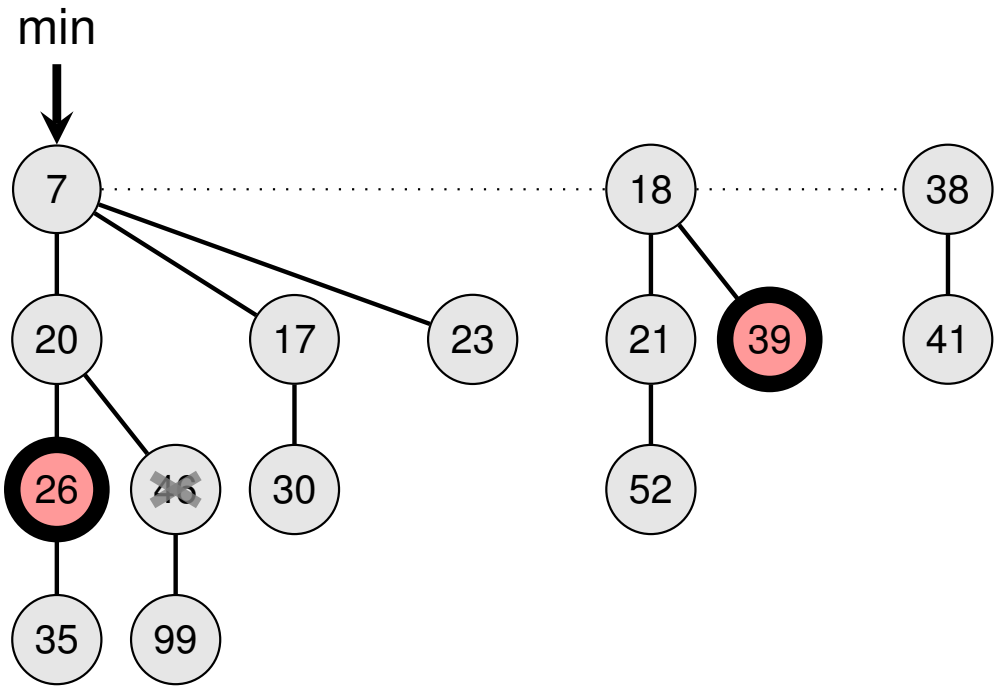


1. DECREASE-KEY 24 $\rightsquigarrow$ 20

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise,

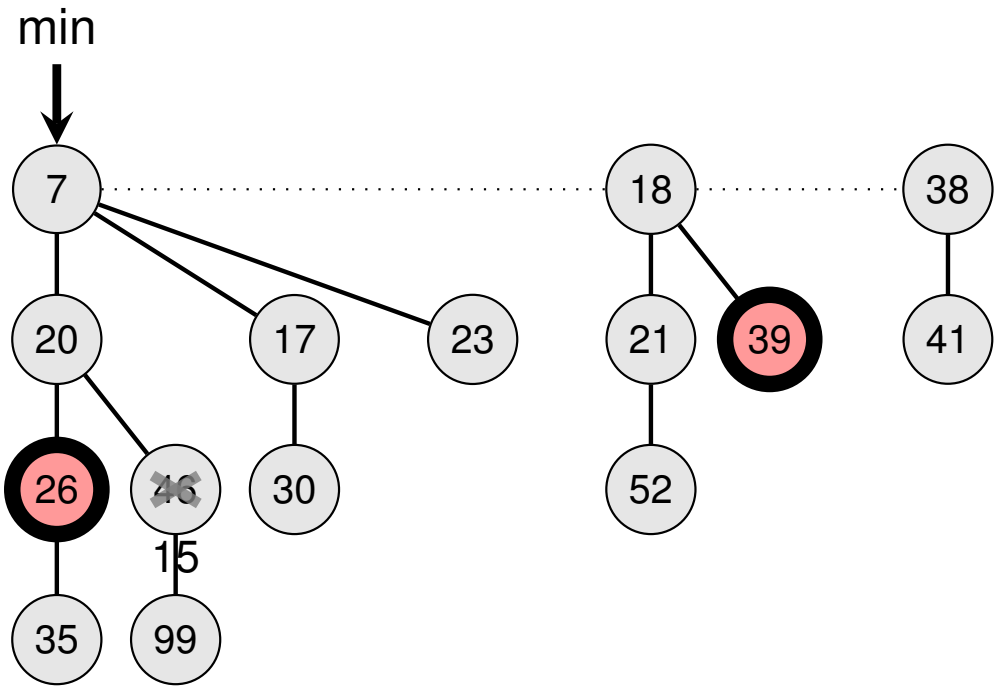


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise,

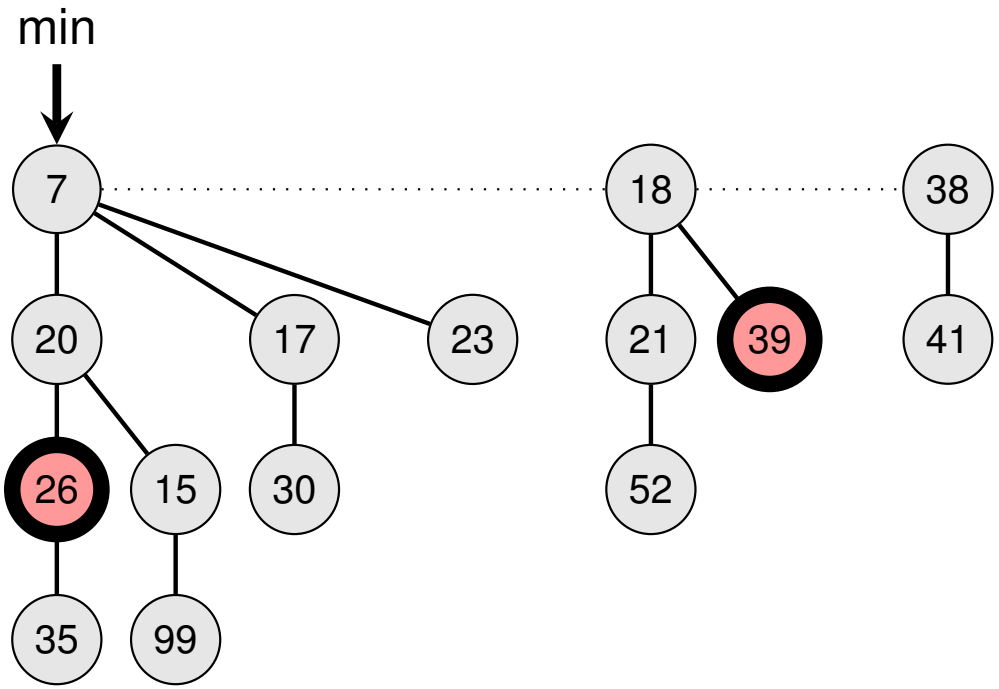


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise,

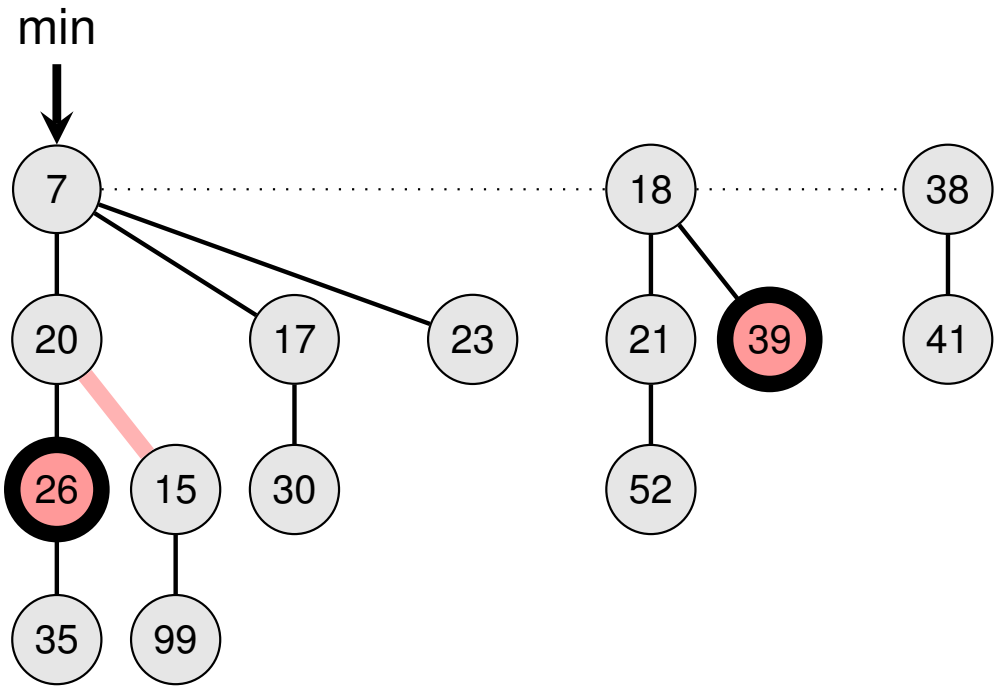


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise,

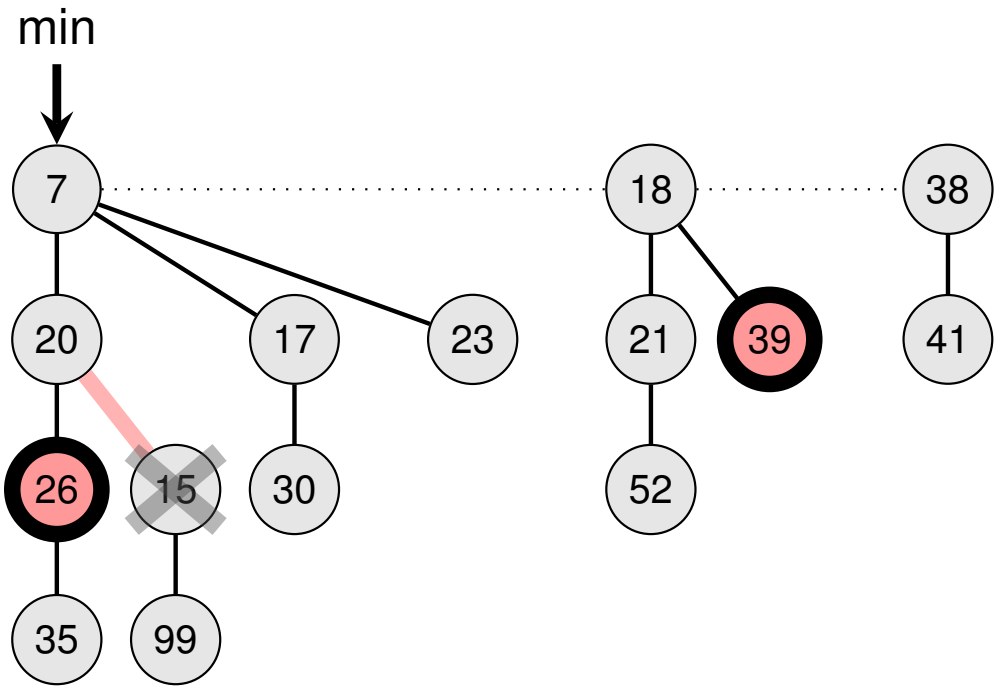


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

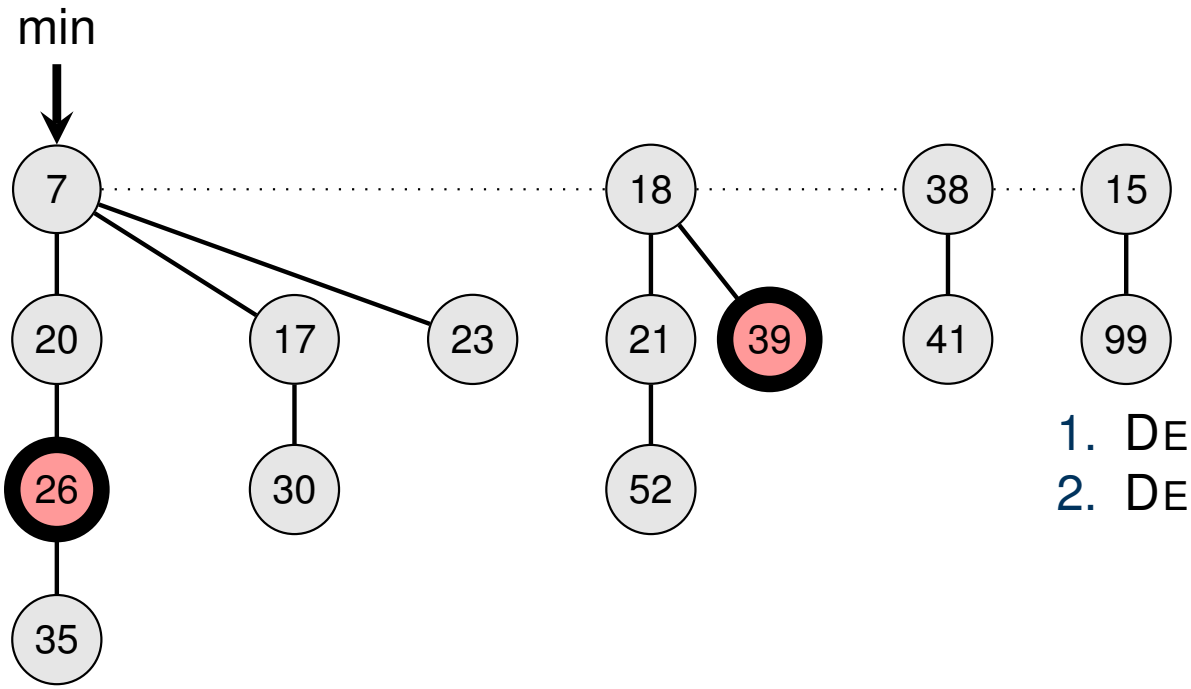


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

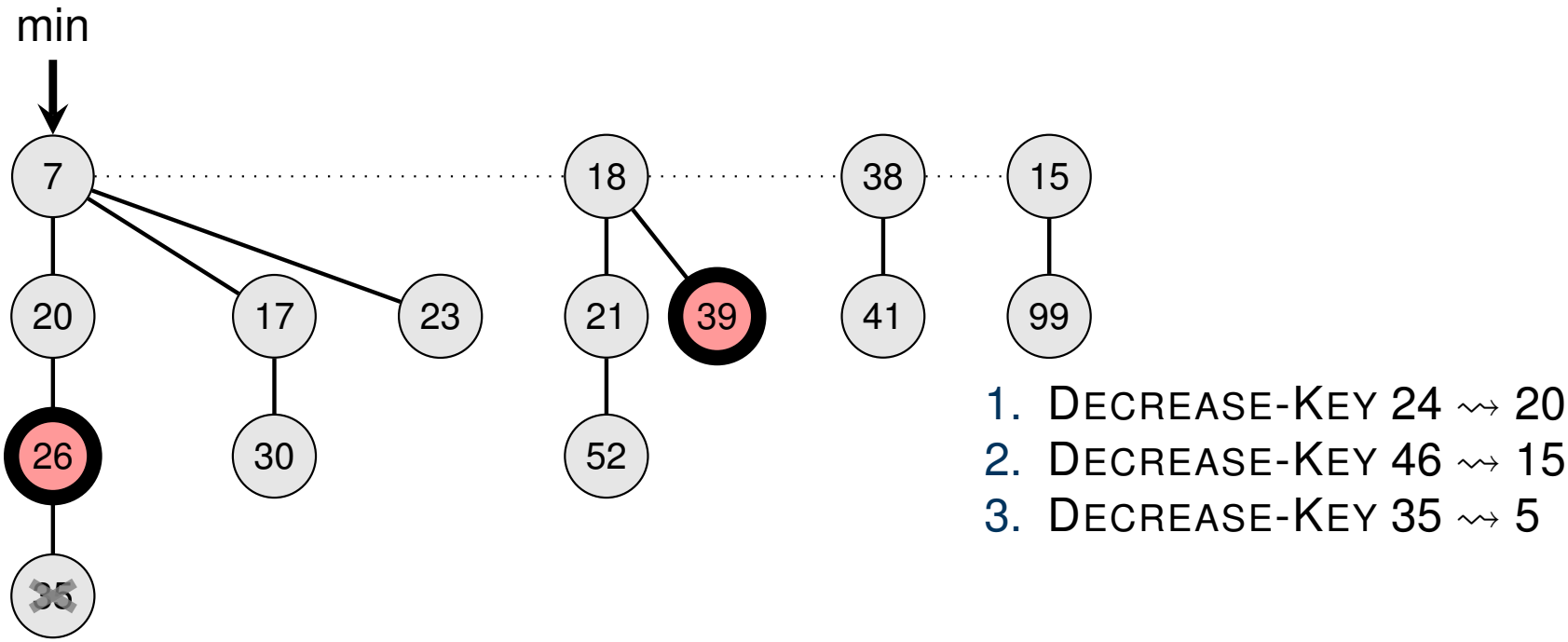


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

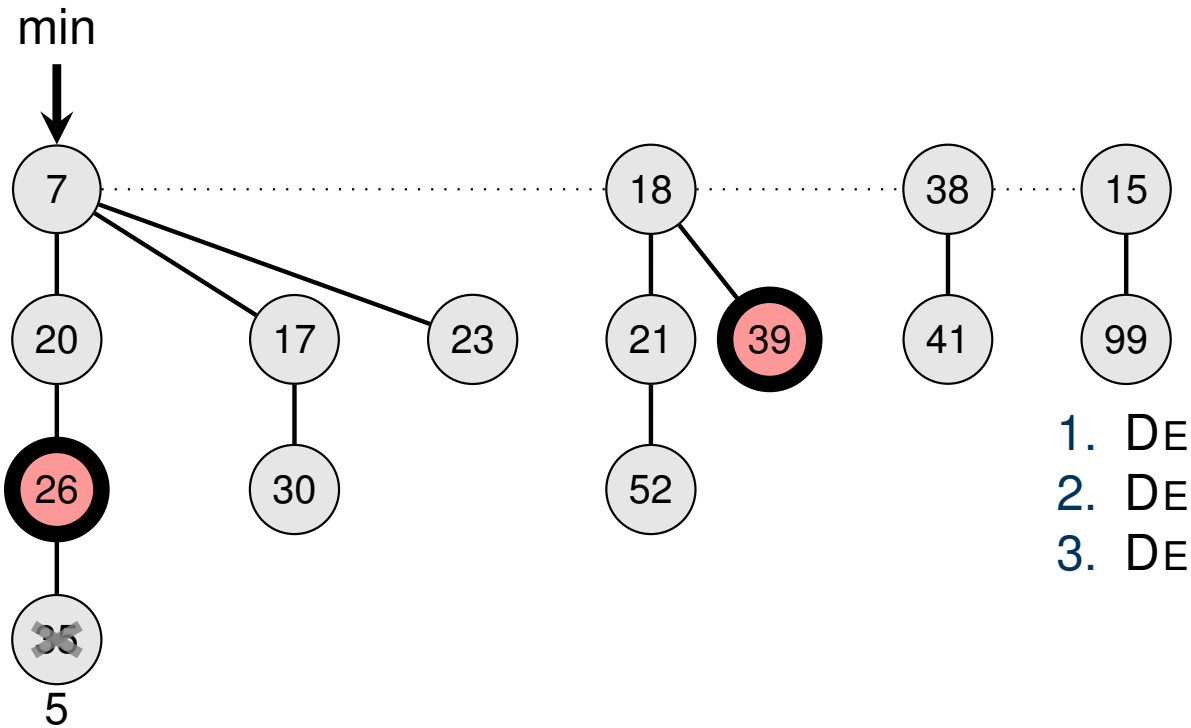
- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).



Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

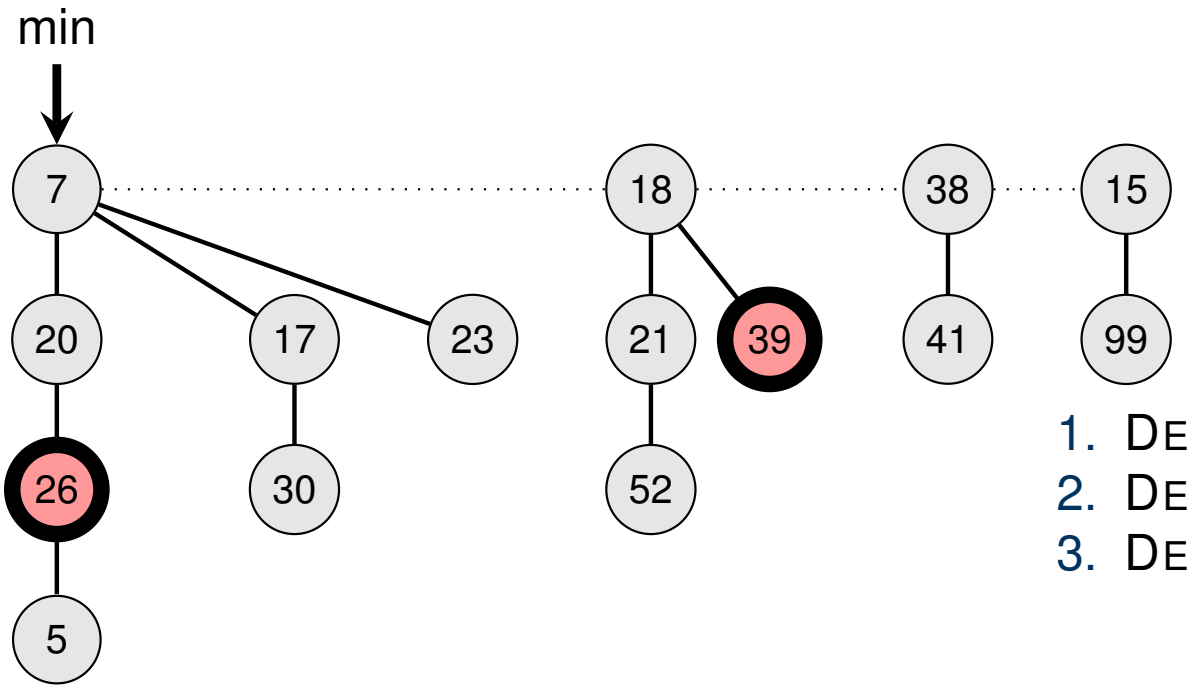


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

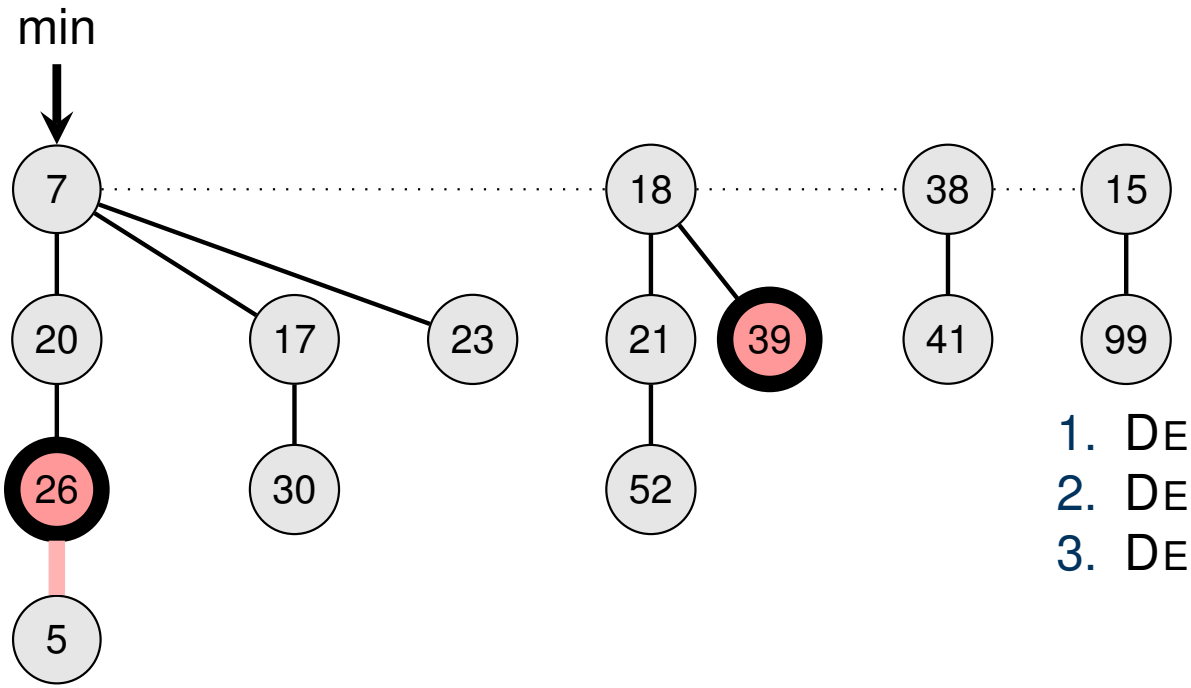


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

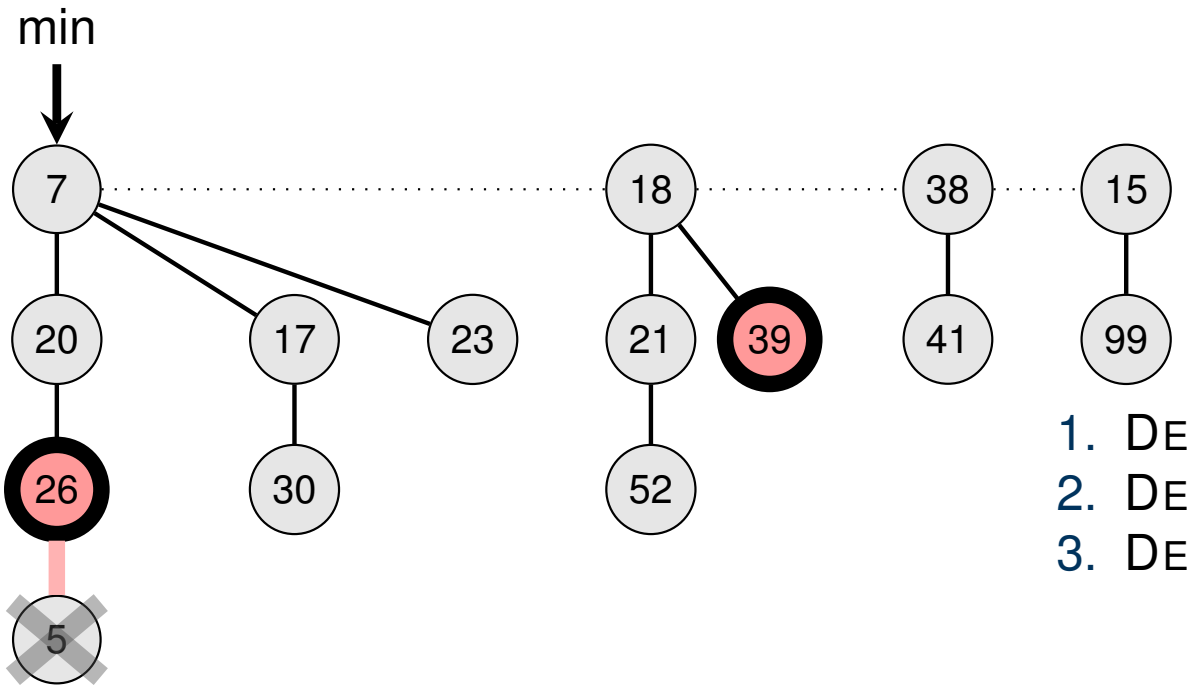


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

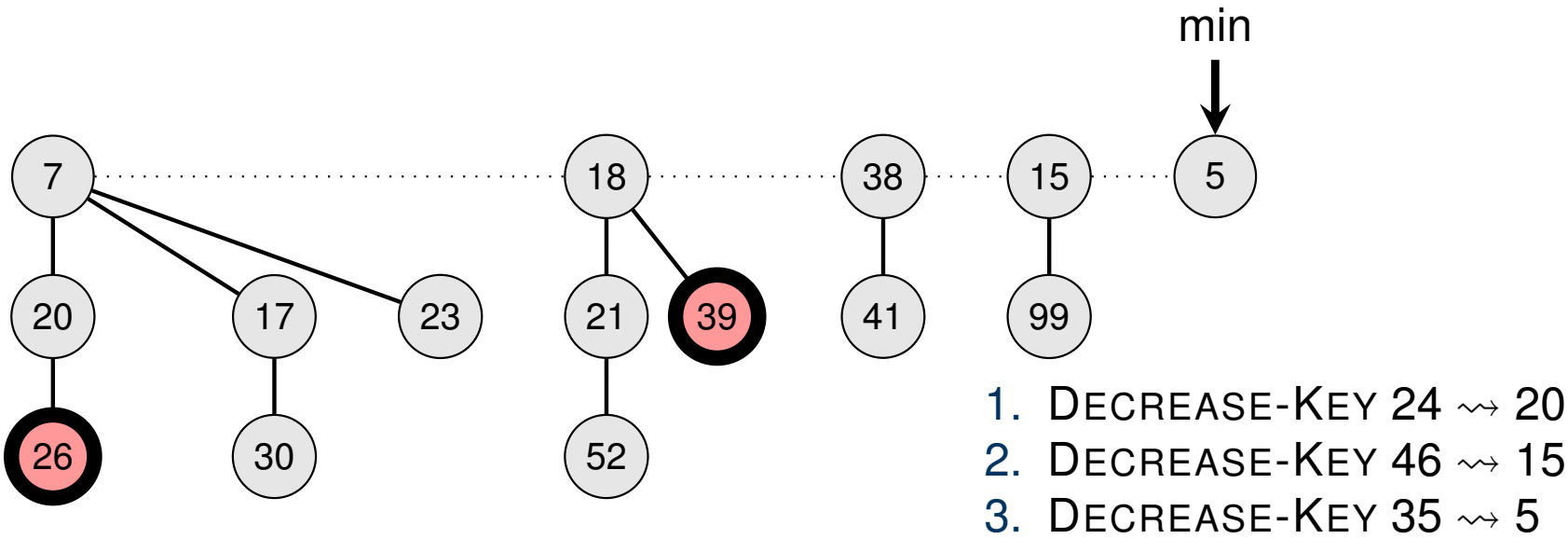


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

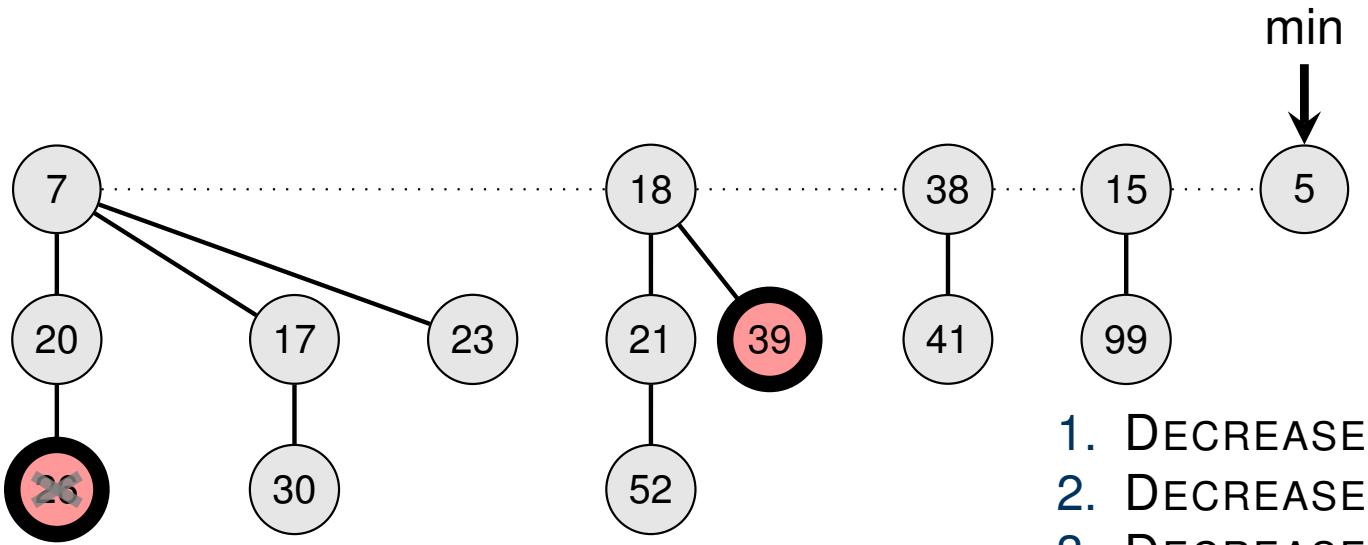


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

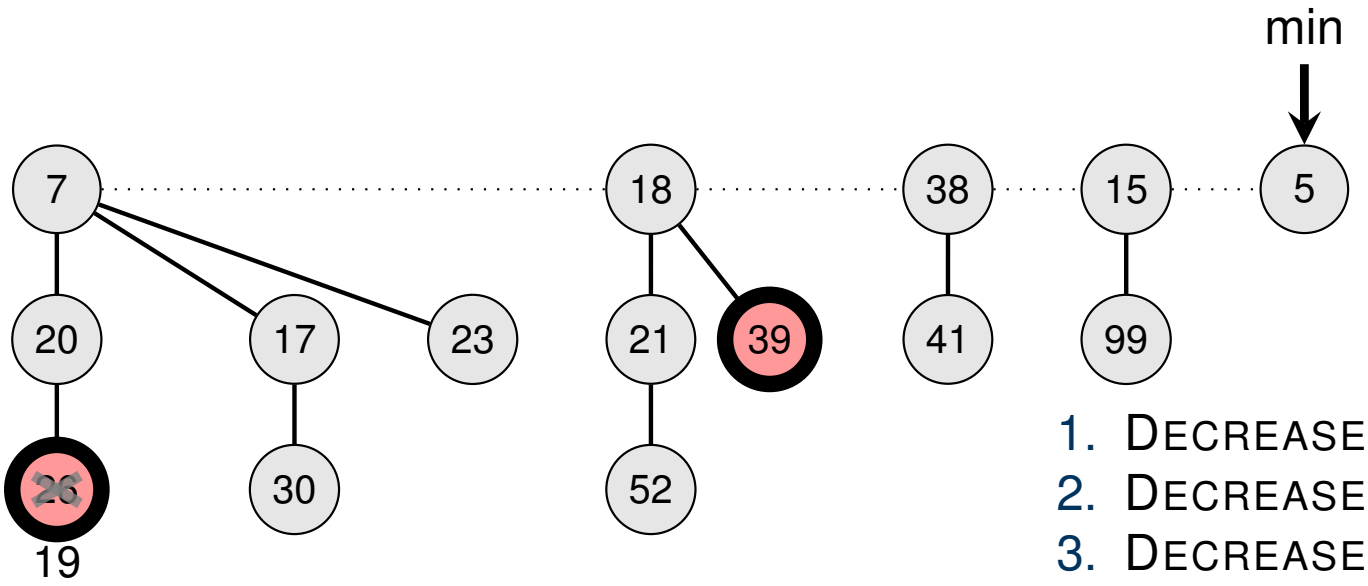


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

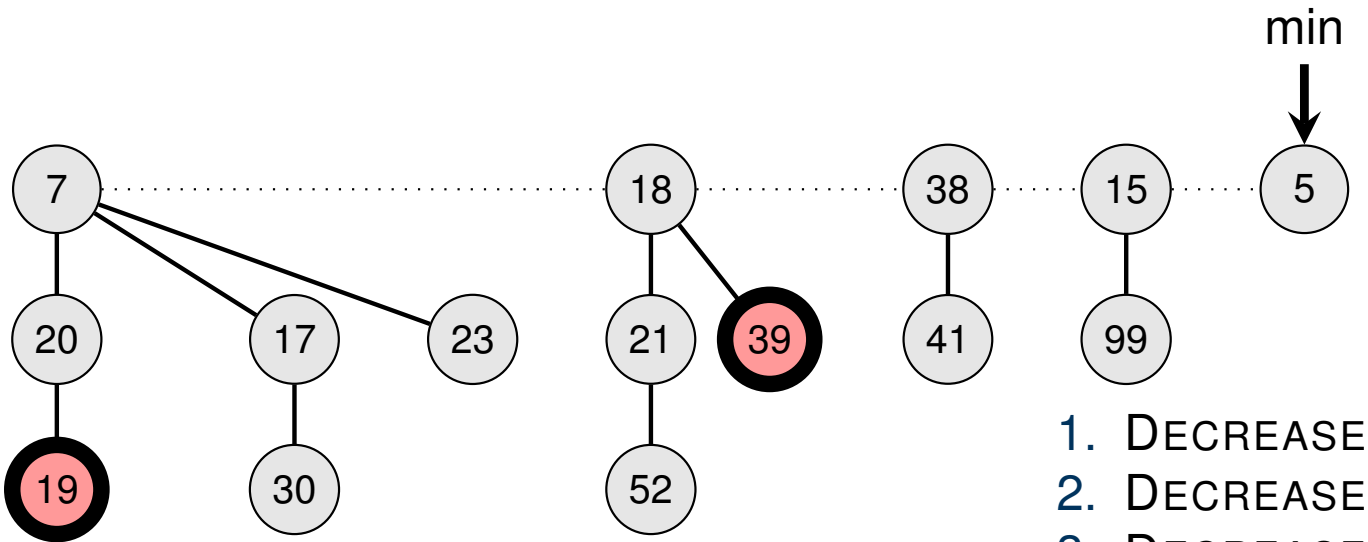


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

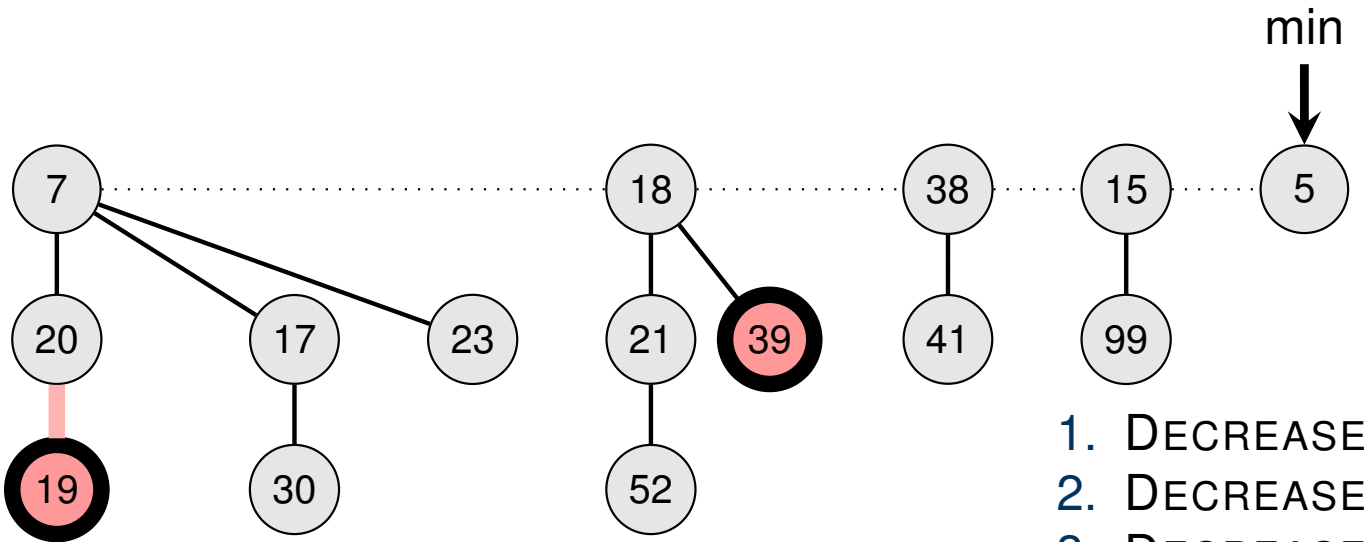


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

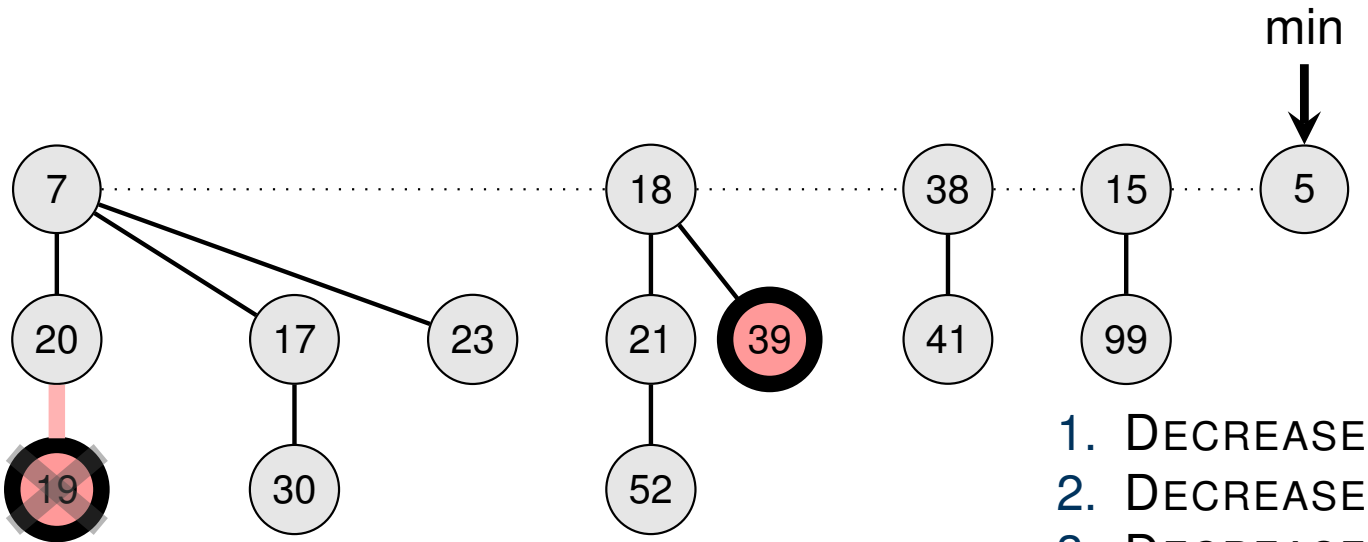


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

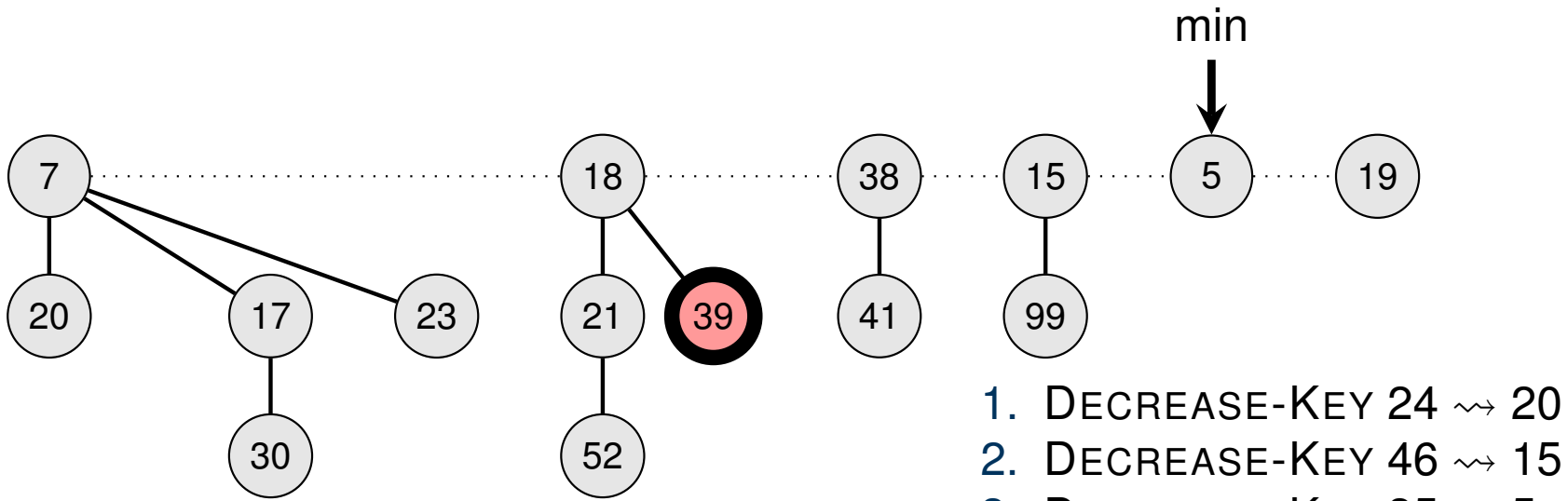


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

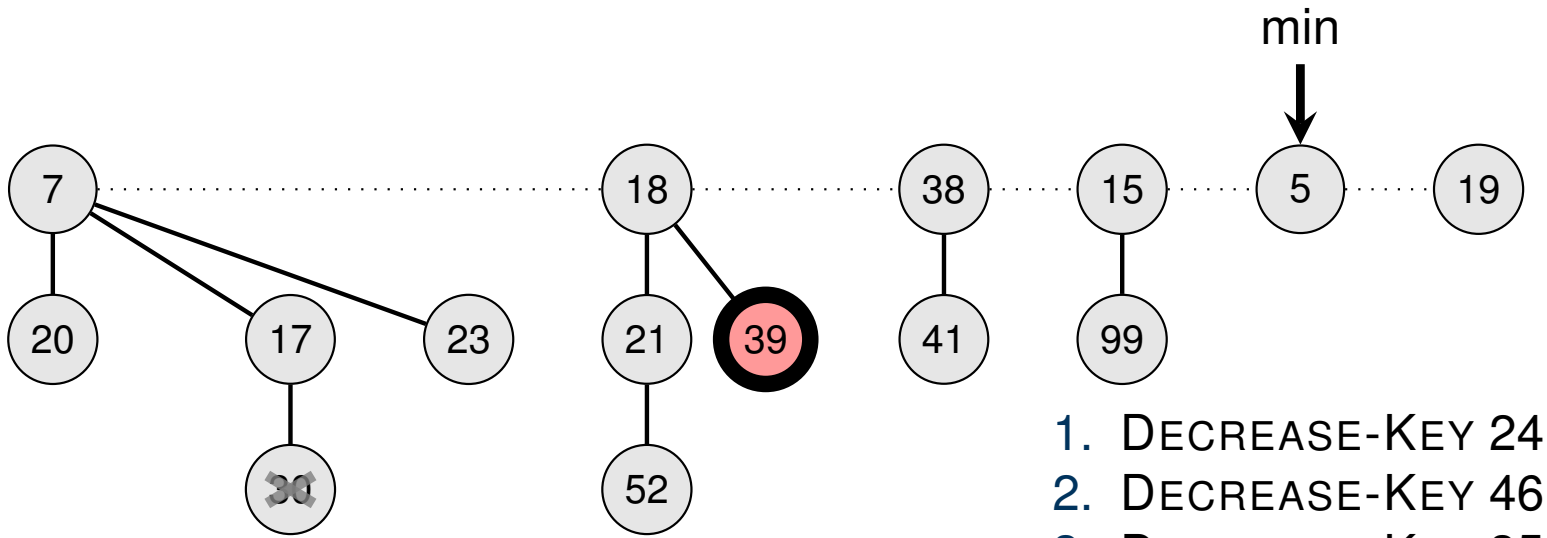


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

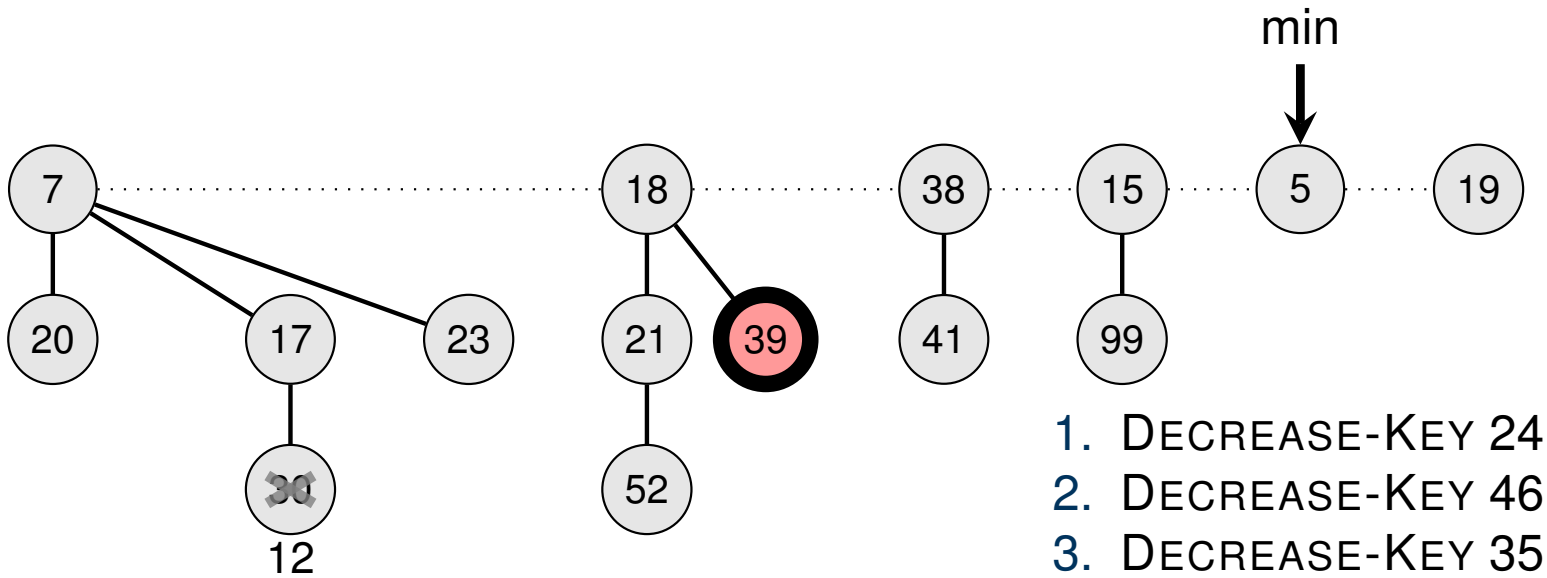


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19
5. DECREASE-KEY 30 $\rightsquigarrow$ 12

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

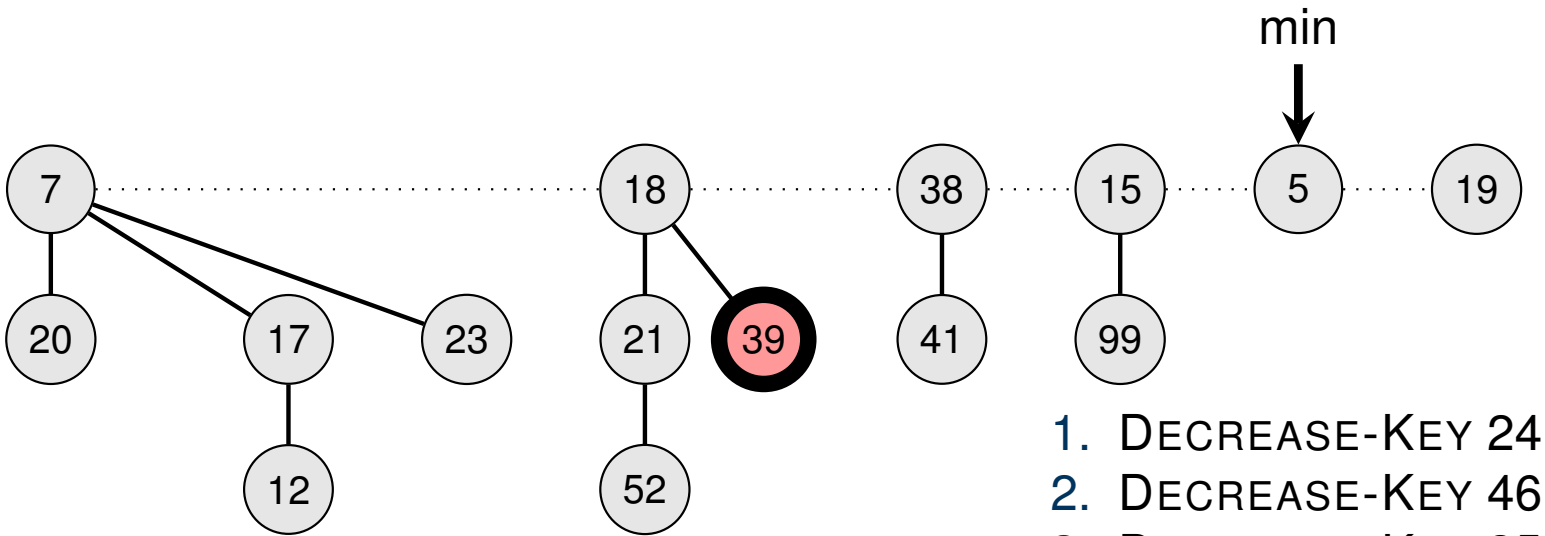


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19
5. DECREASE-KEY 30 $\rightsquigarrow$ 12

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

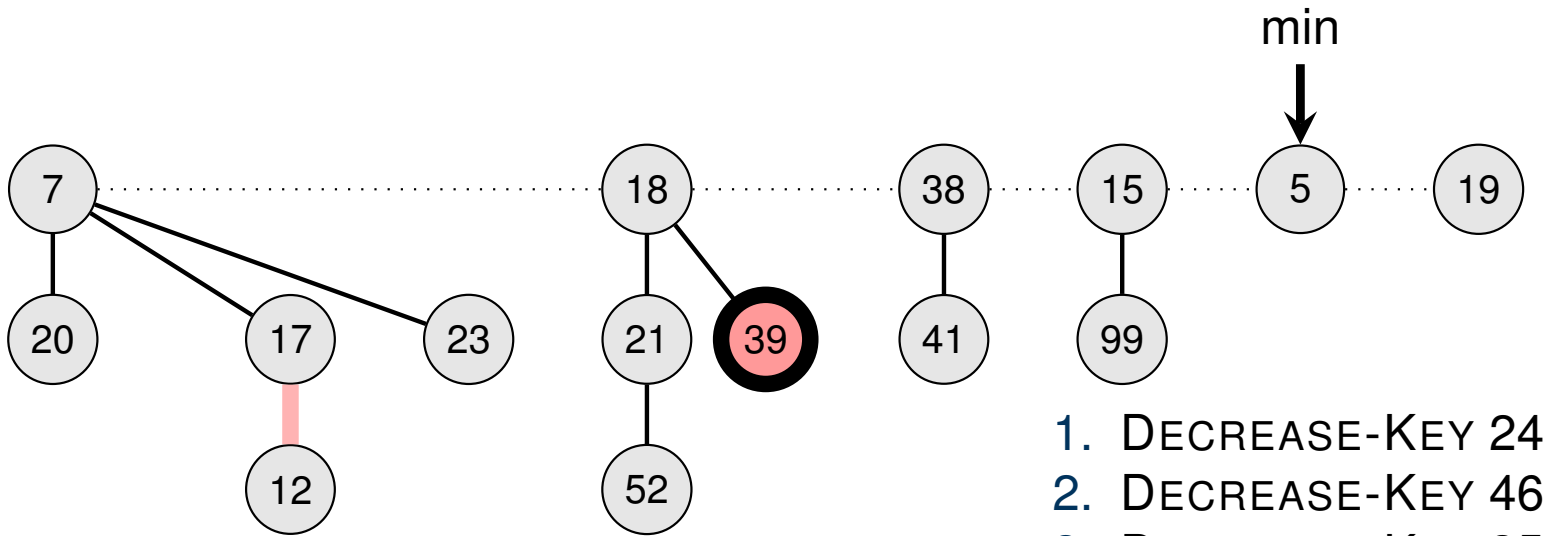


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19
5. DECREASE-KEY 30 $\rightsquigarrow$ 12

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

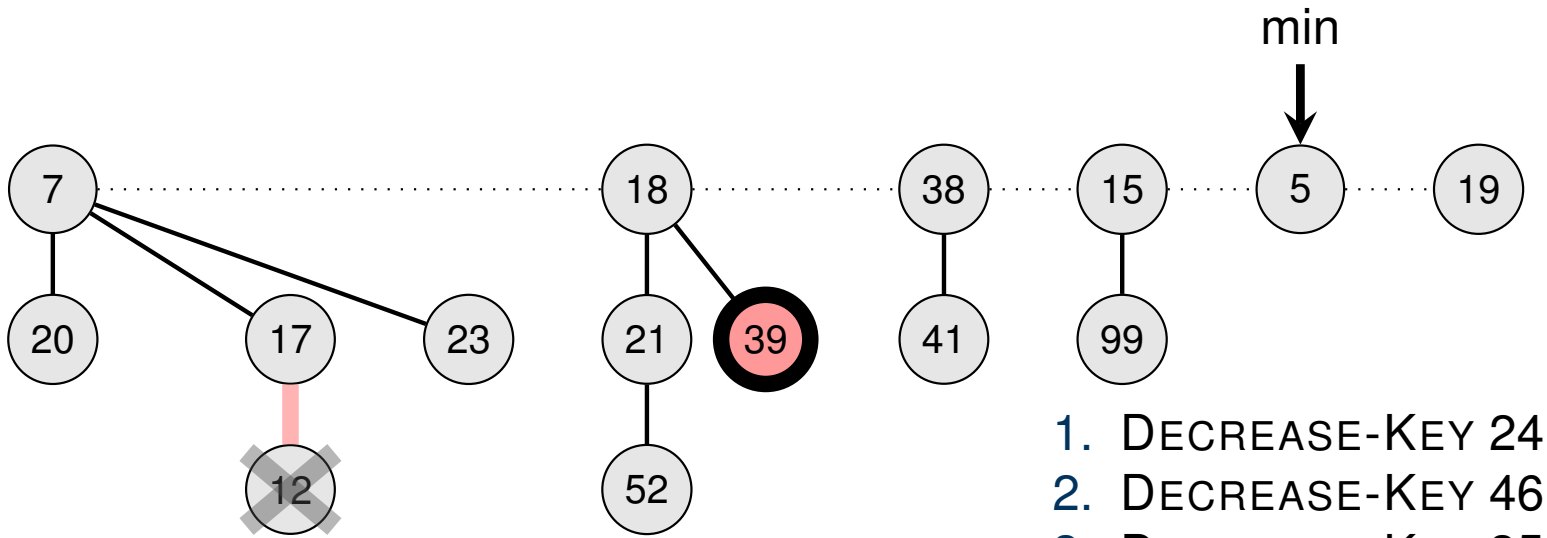


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19
5. DECREASE-KEY 30 $\rightsquigarrow$ 12

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

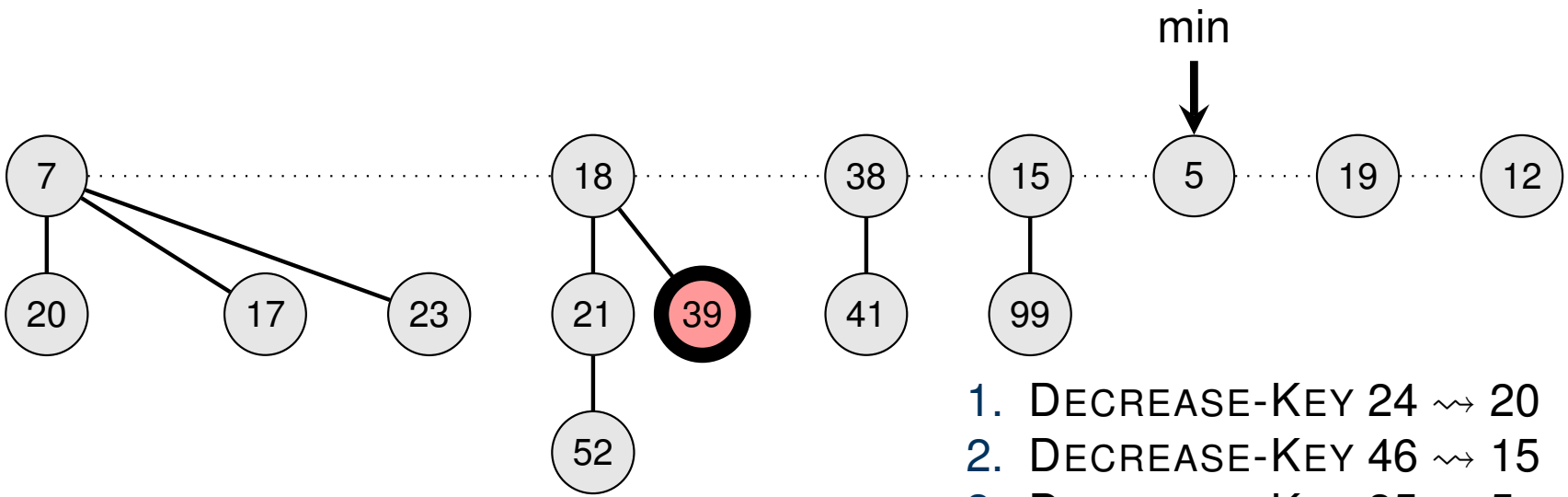


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19
5. DECREASE-KEY 30 $\rightsquigarrow$ 12

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

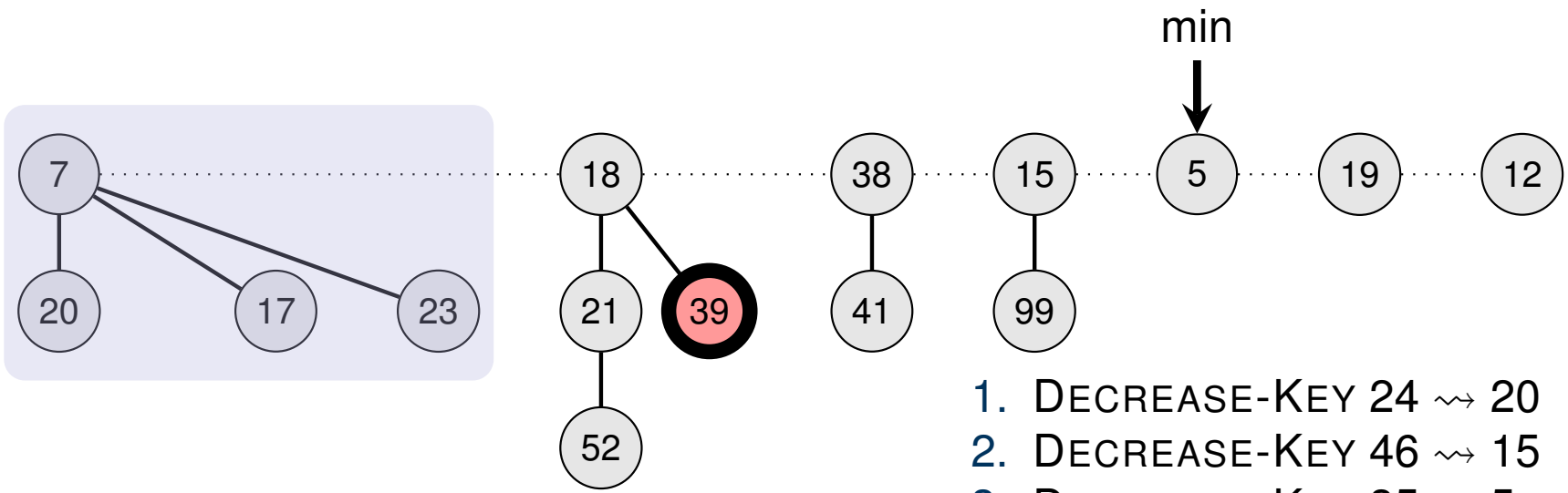


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19
5. DECREASE-KEY 30 $\rightsquigarrow$ 12

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

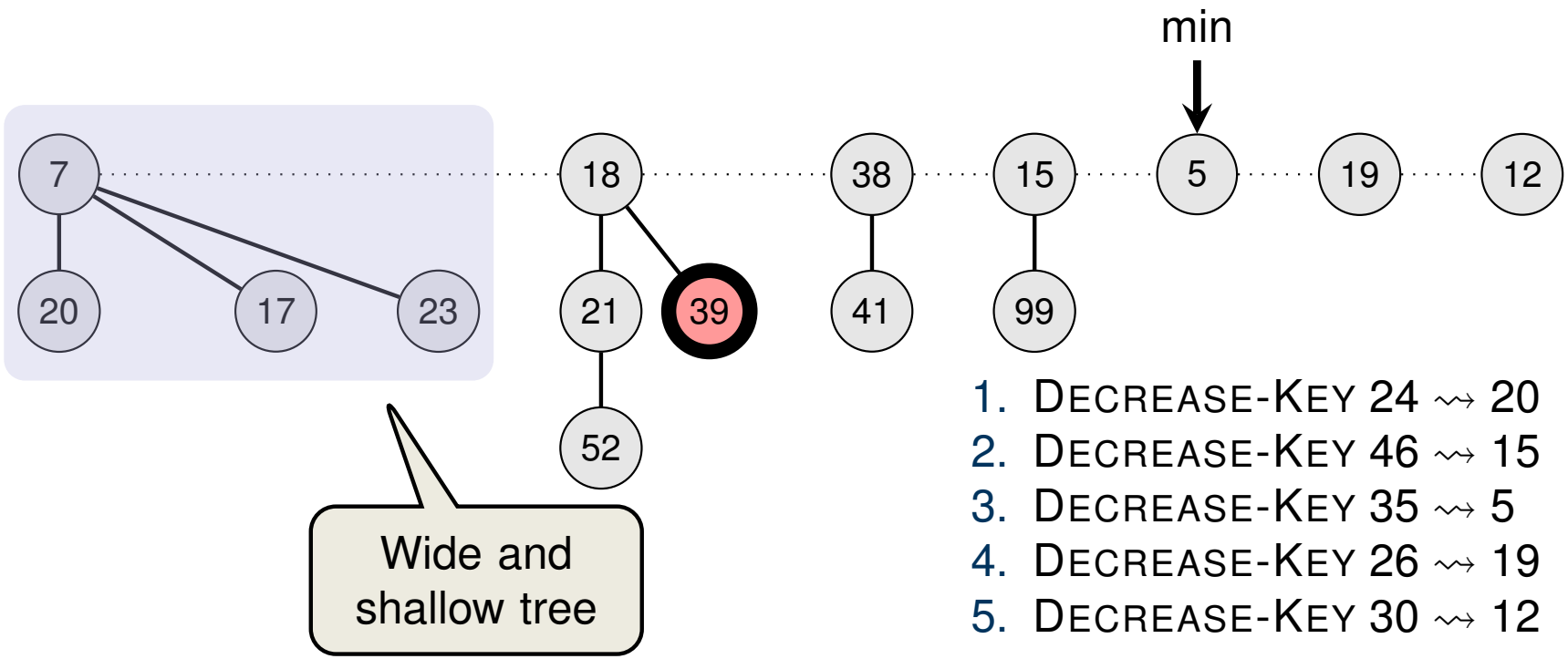


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19
5. DECREASE-KEY 30 $\rightsquigarrow$ 12

Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

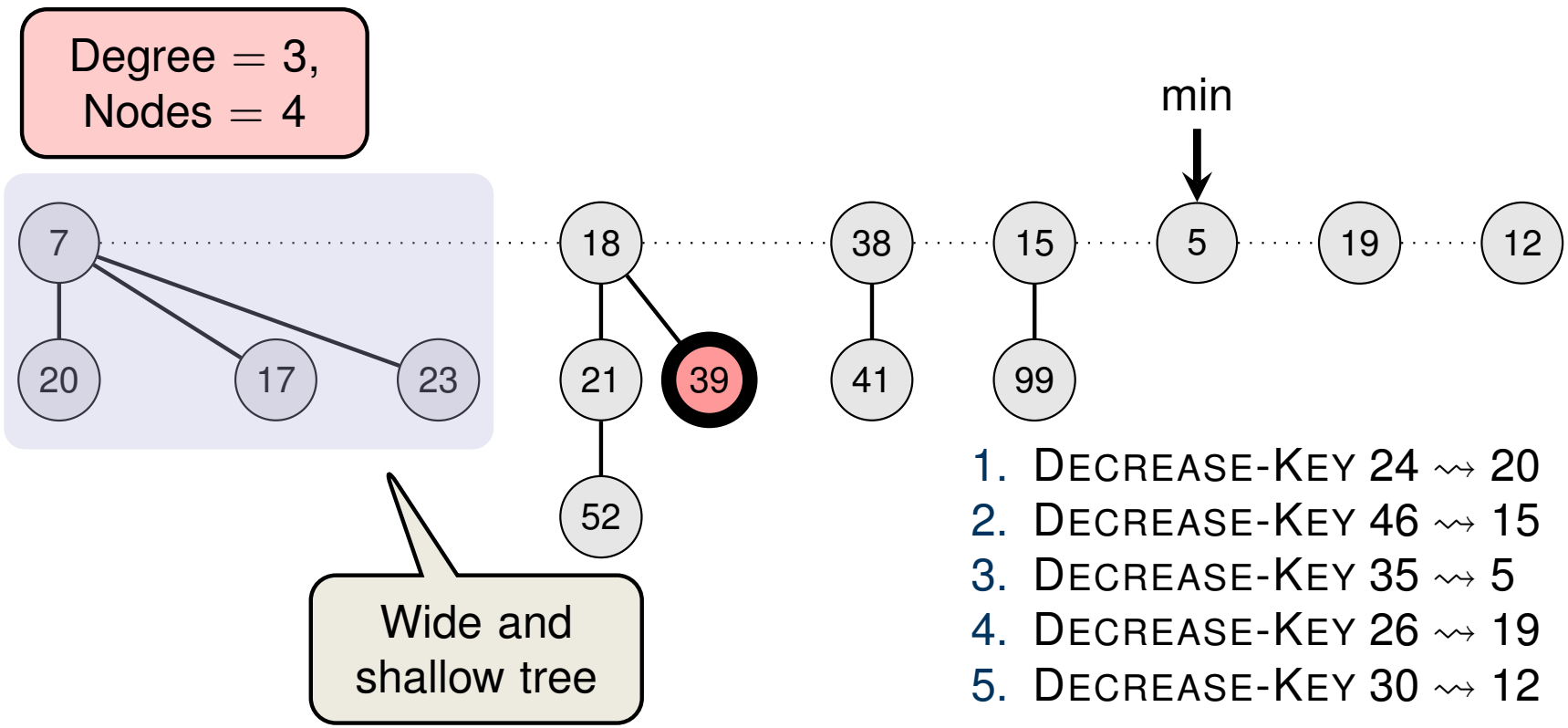
- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).



Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

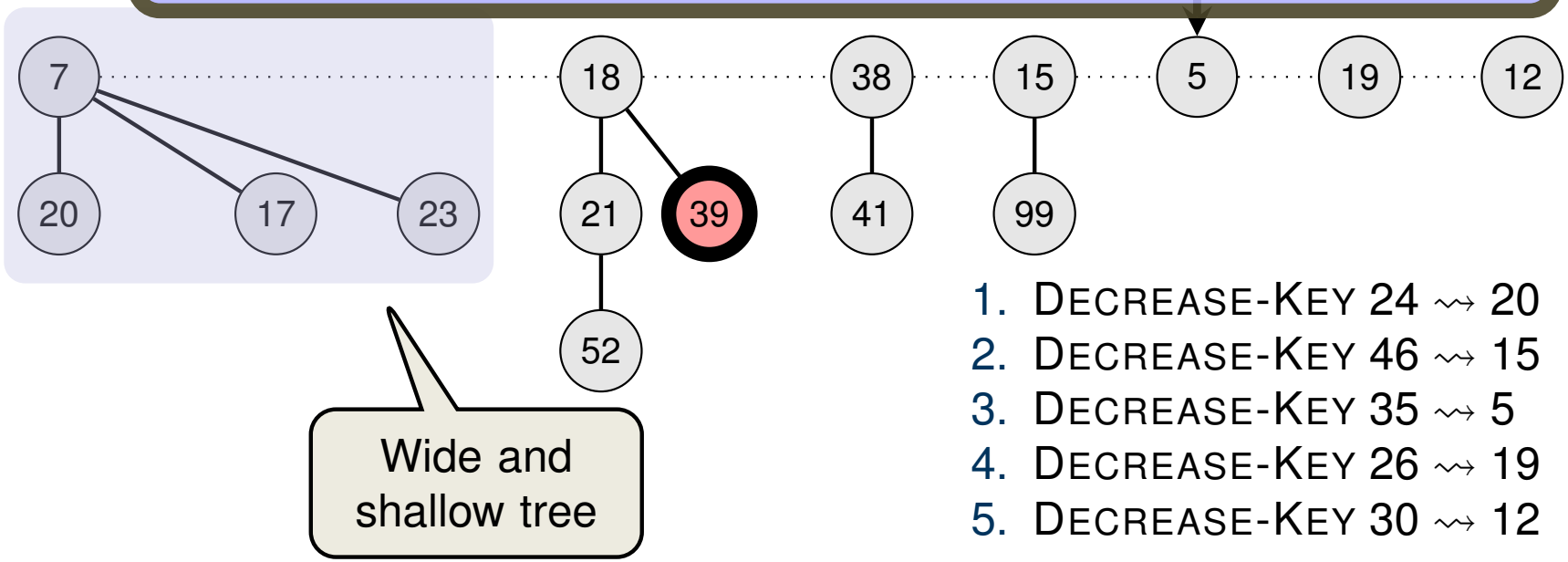


Fibonacci Heap: DECREASE-KEY (First Try)

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- Check if heap-order is violated
  - If not, then done.
  - Otherwise, cut tree rooted at x and meld into root list (update min).

Peculiar Constraint: Make sure that each non-root node loses at most one child before becoming root

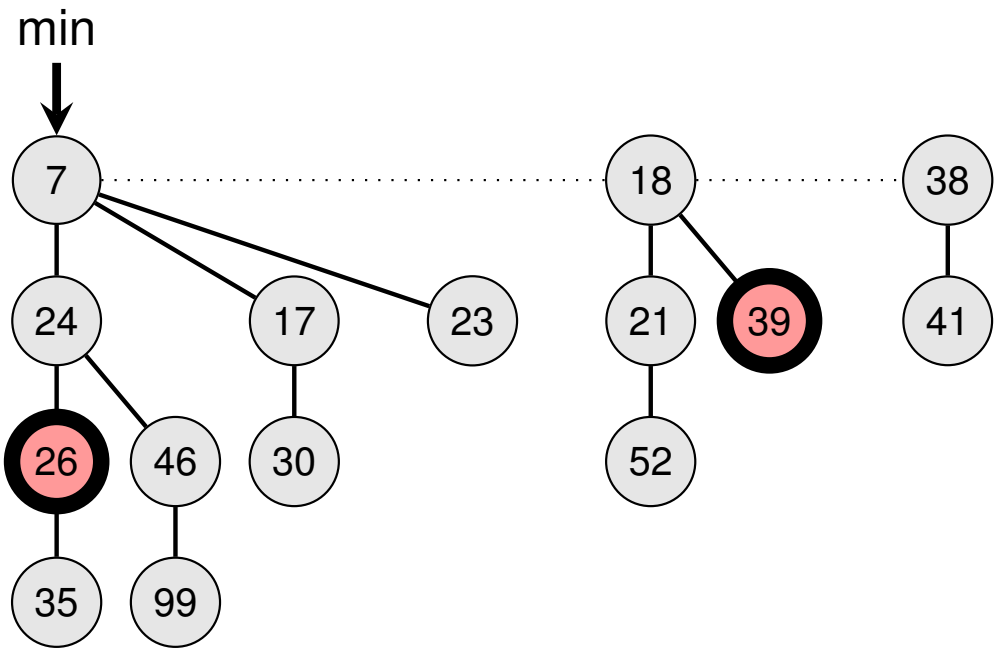


1. DECREASE-KEY 24 $\rightsquigarrow$ 20
2. DECREASE-KEY 46 $\rightsquigarrow$ 15
3. DECREASE-KEY 35 $\rightsquigarrow$ 5
4. DECREASE-KEY 26 $\rightsquigarrow$ 19
5. DECREASE-KEY 30 $\rightsquigarrow$ 12

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

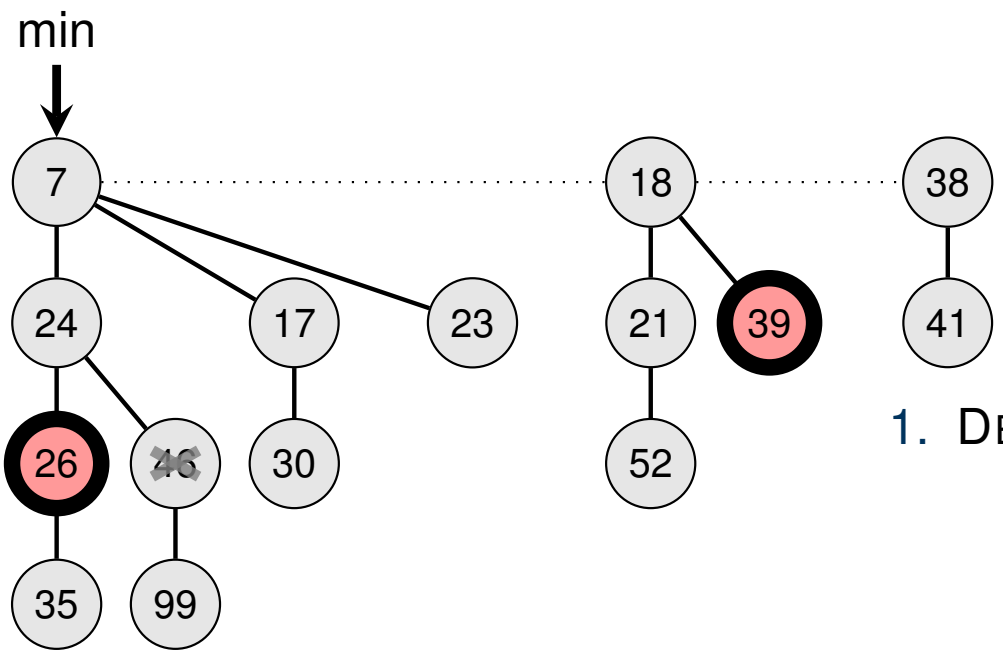
- Decrease the key of x (given by a pointer)



Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)



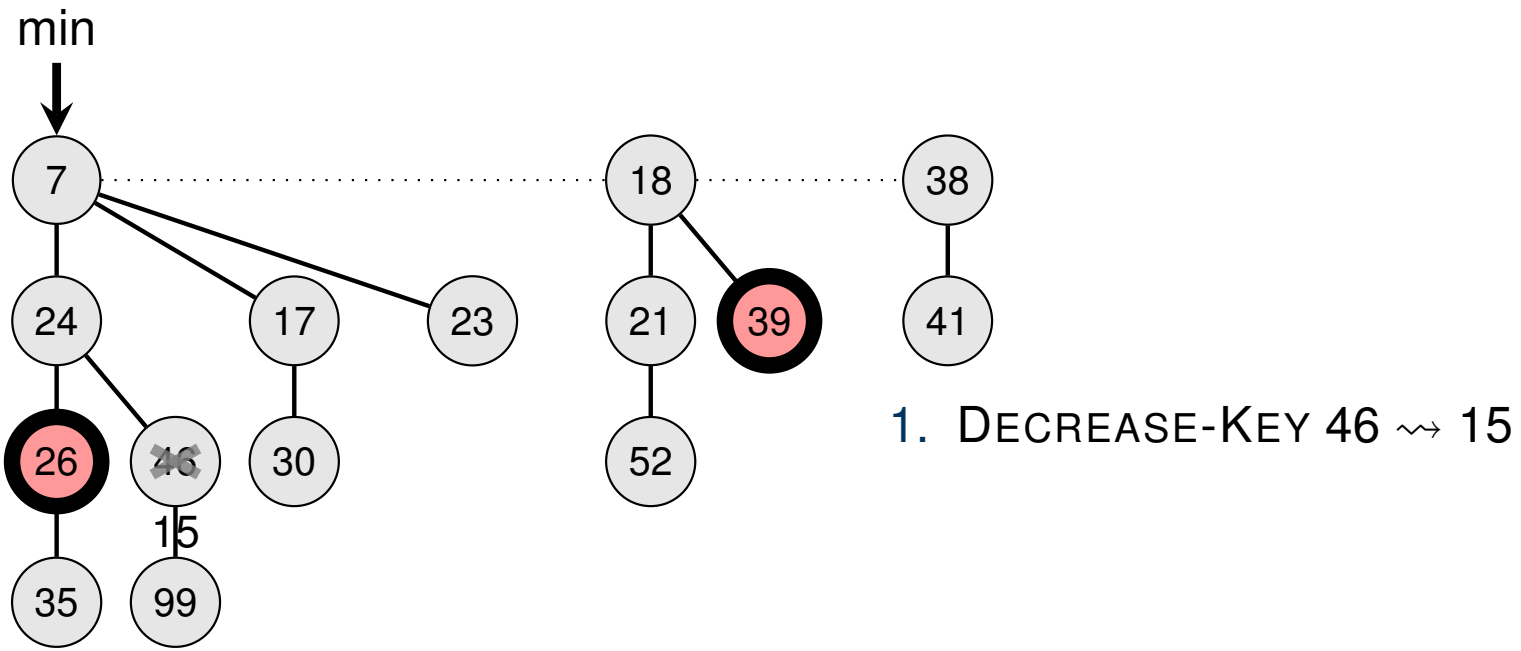
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list



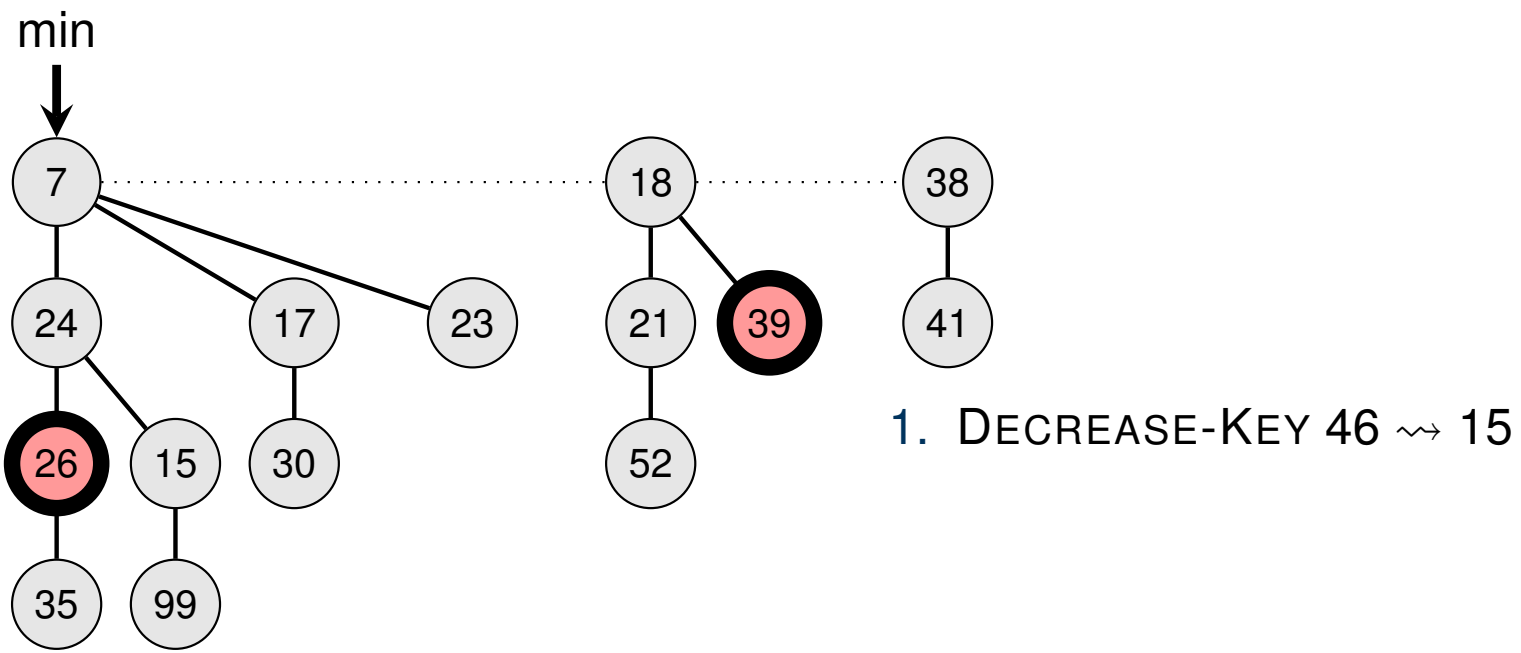
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list



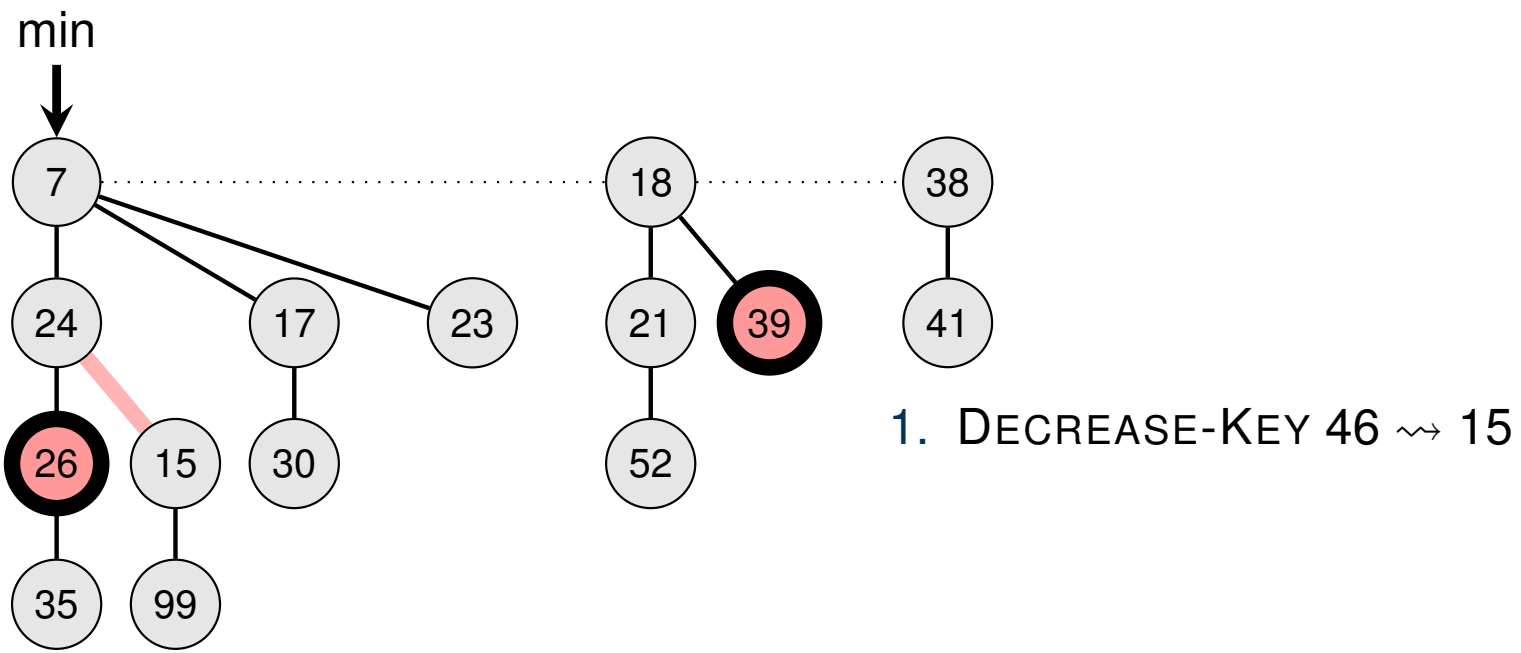
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list



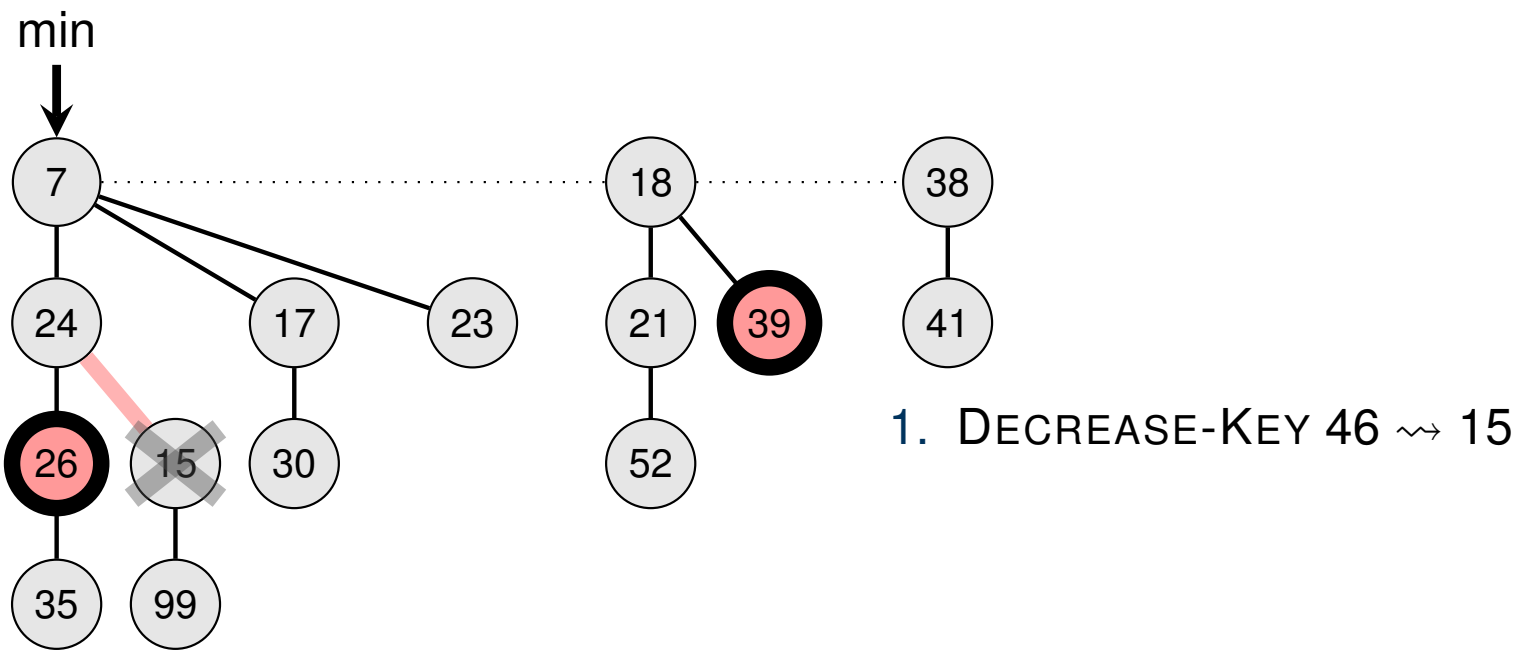
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list



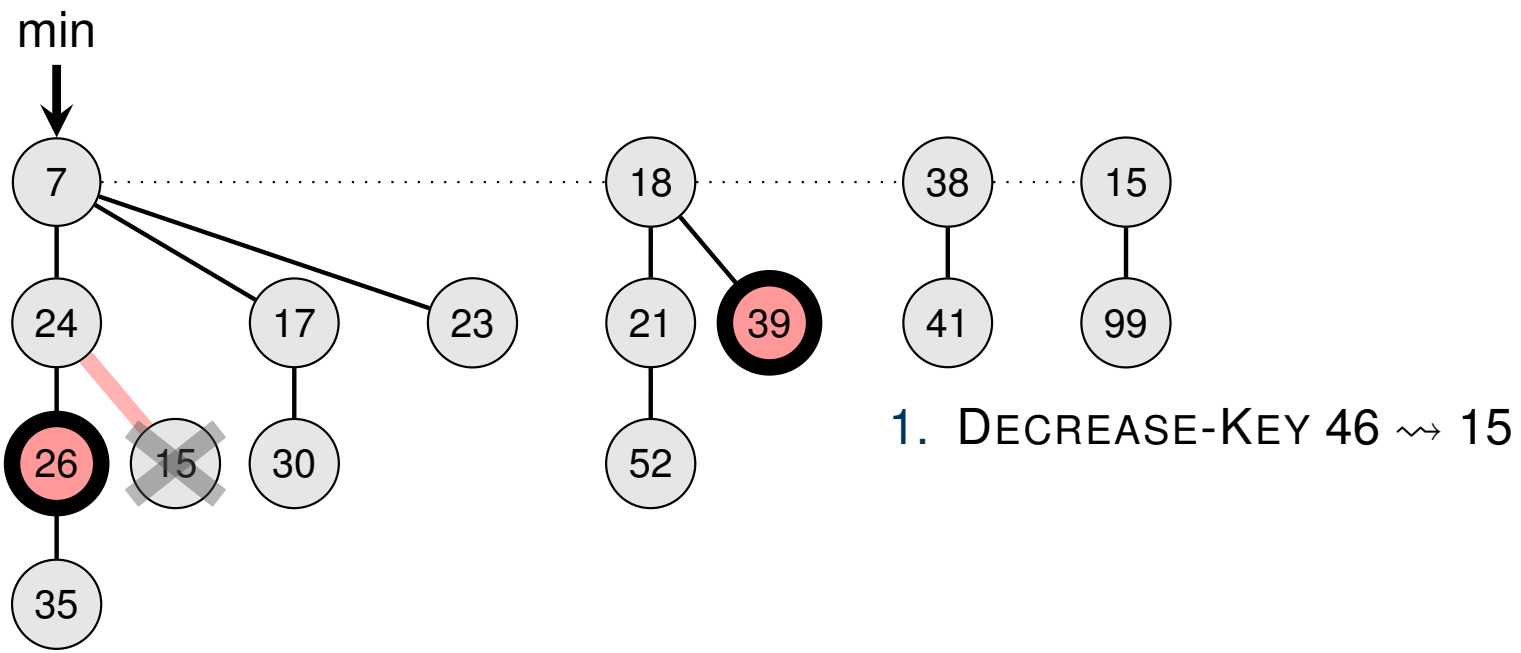
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:



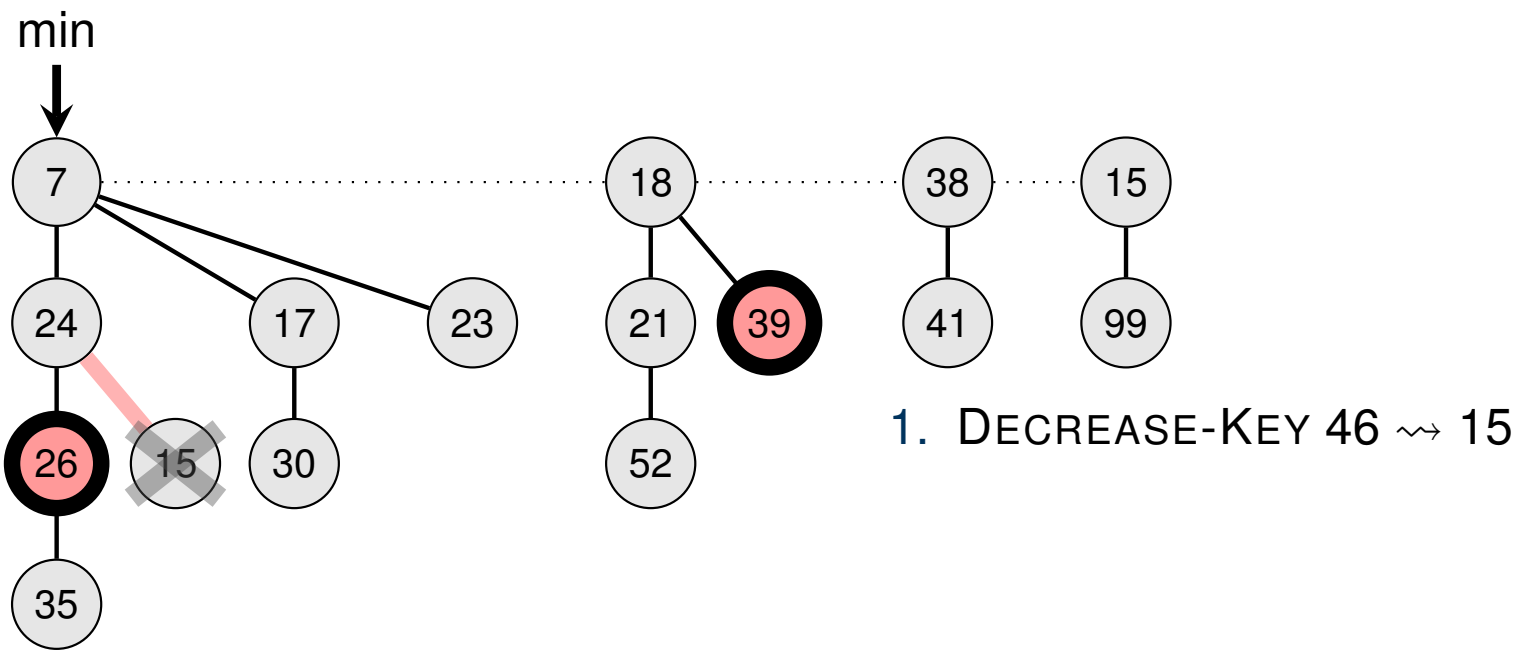
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked



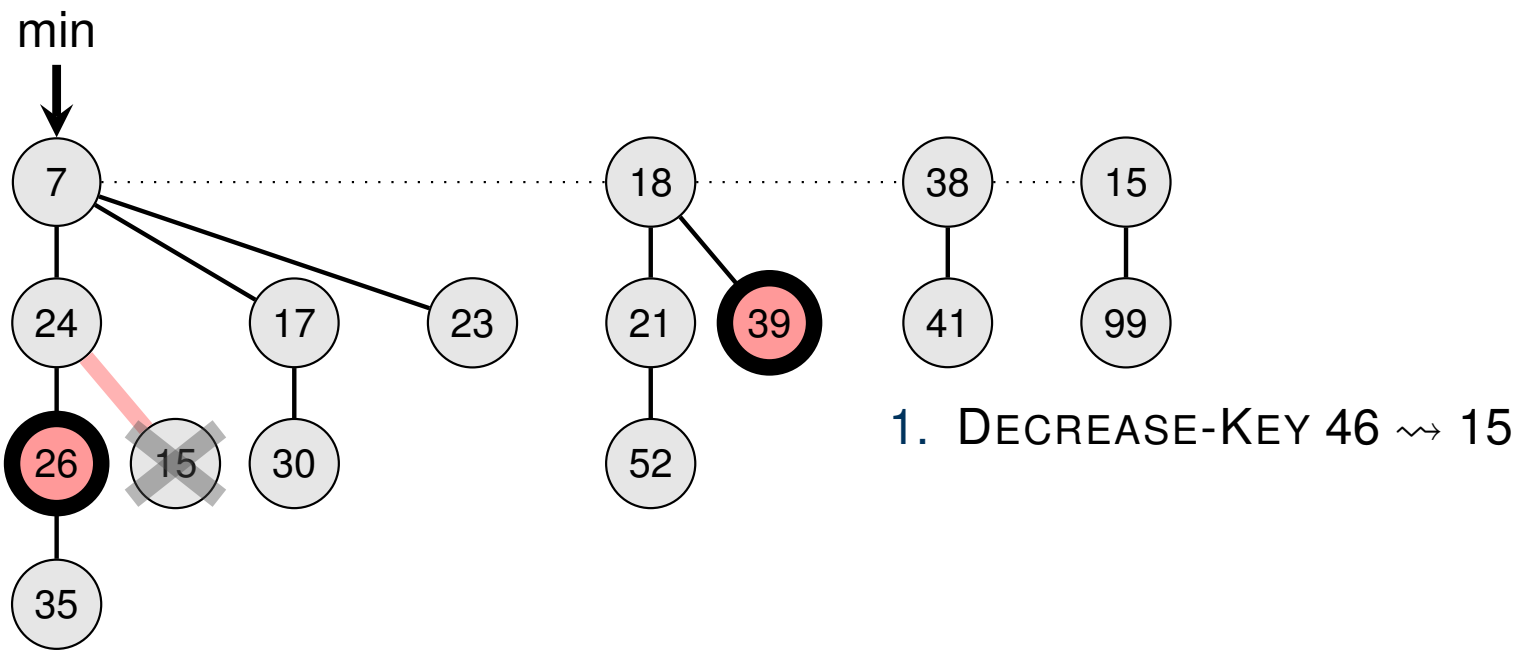
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



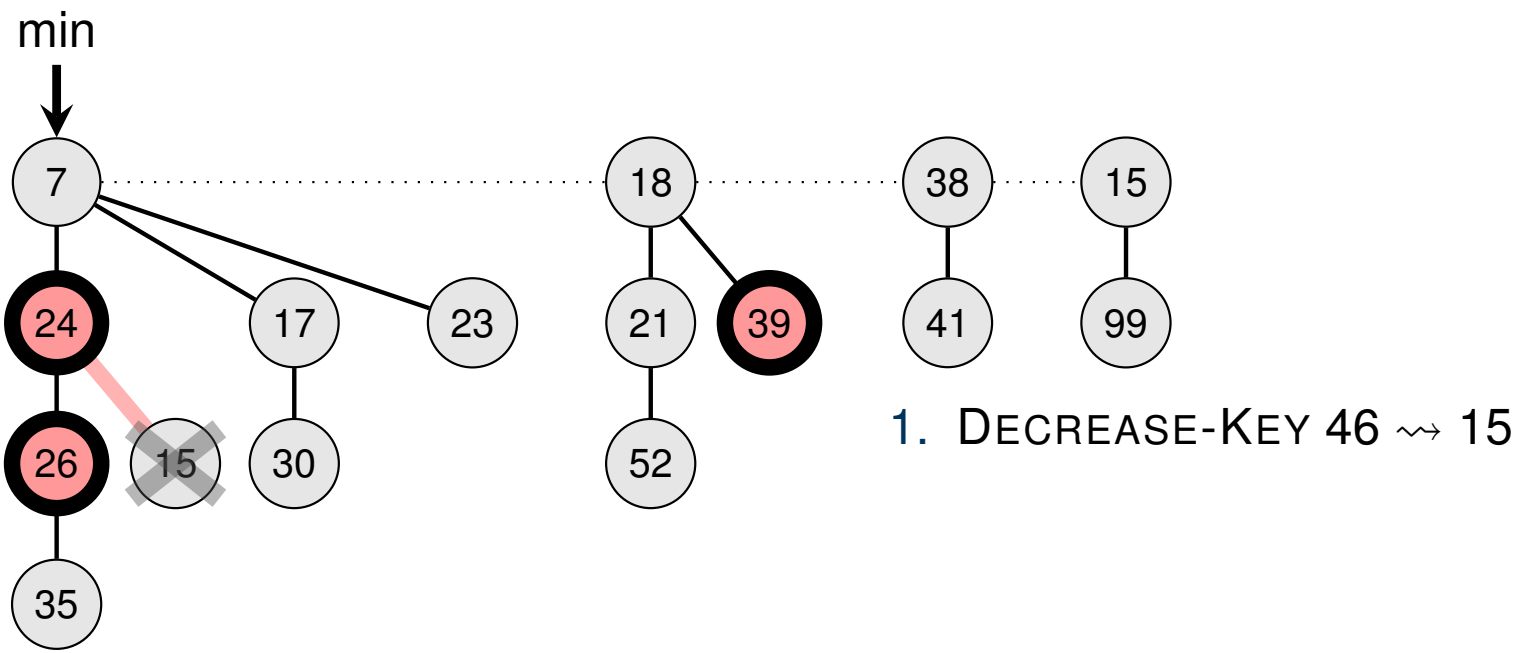
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



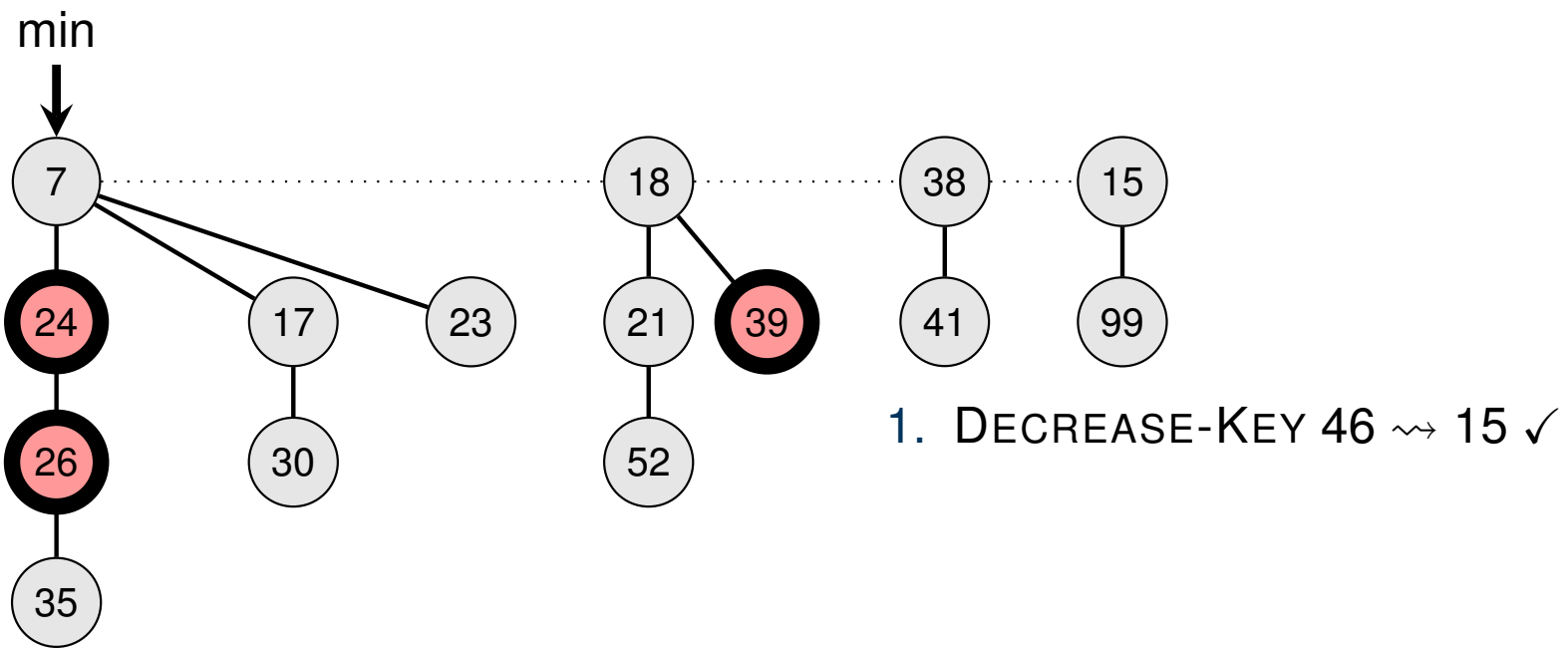
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓

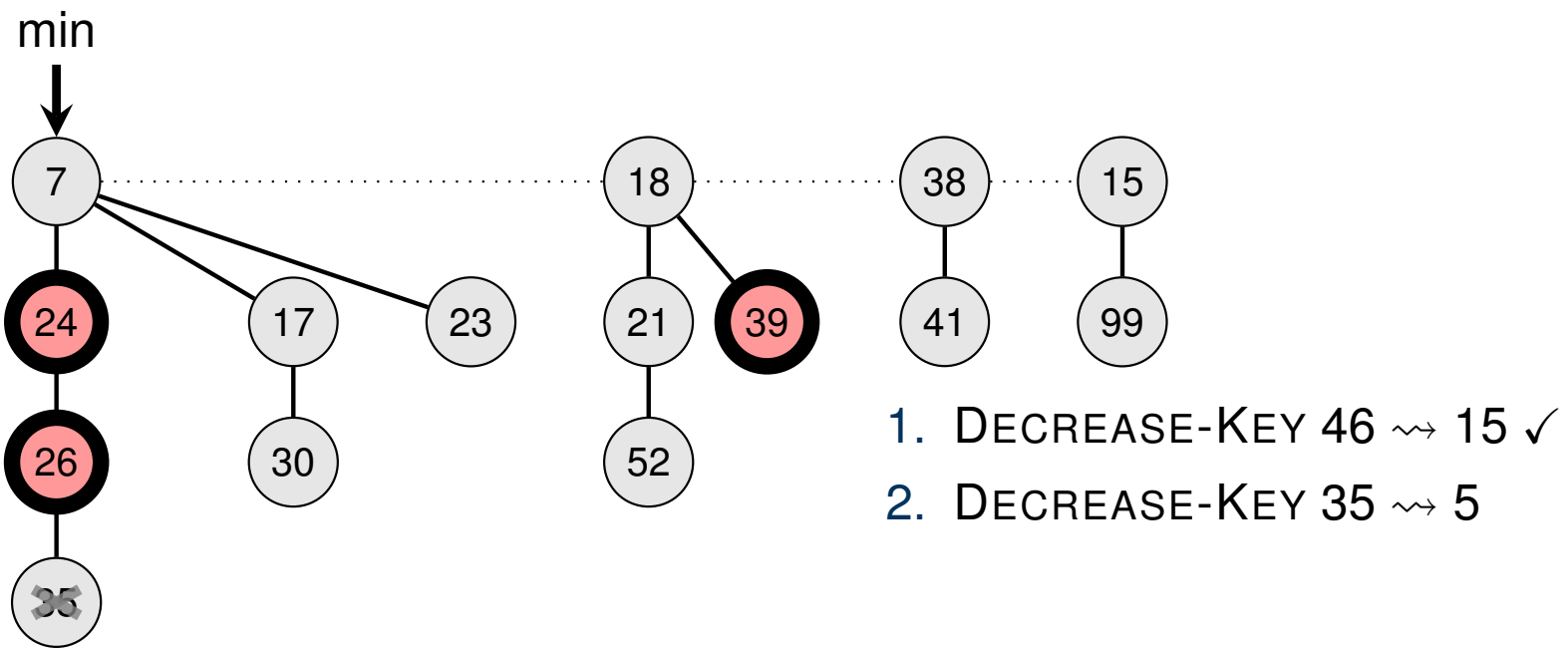
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

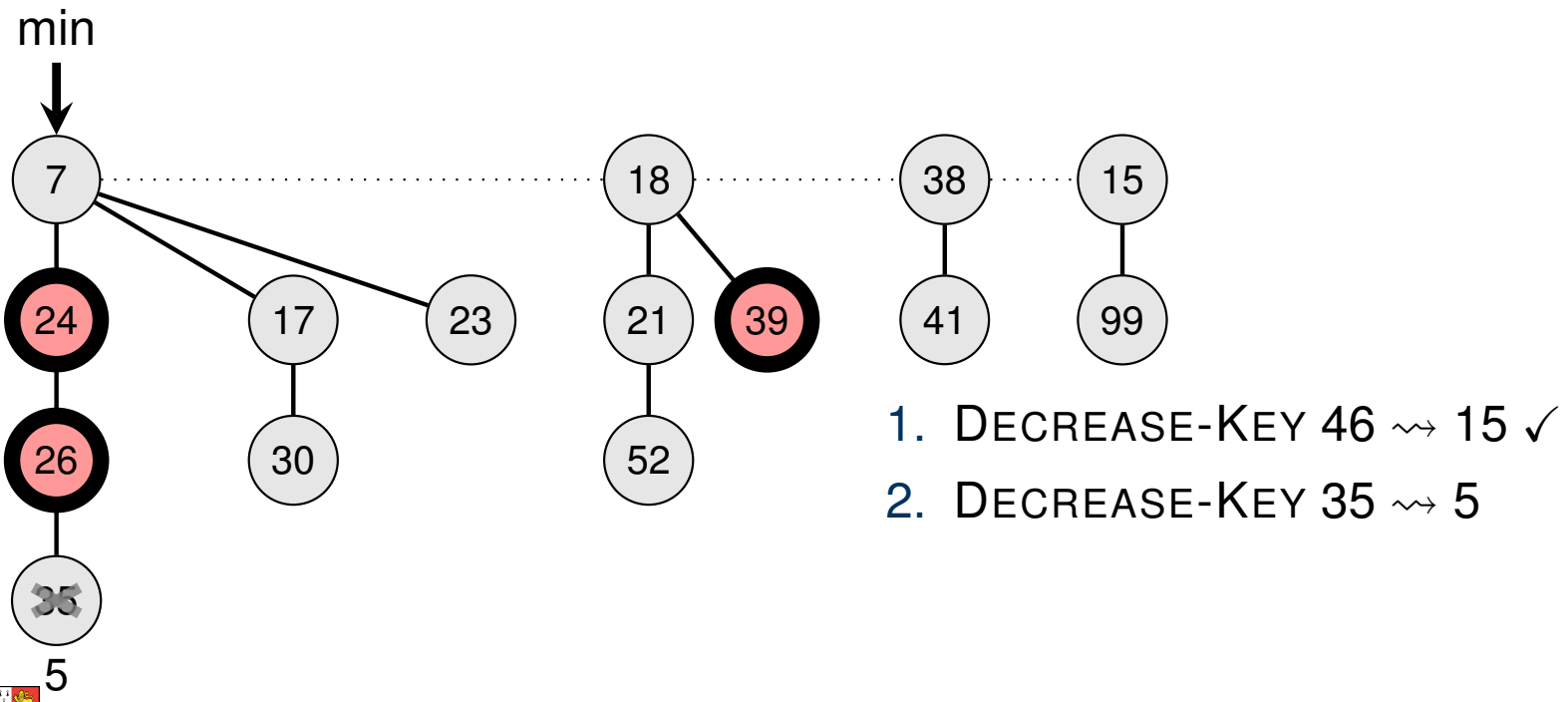
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

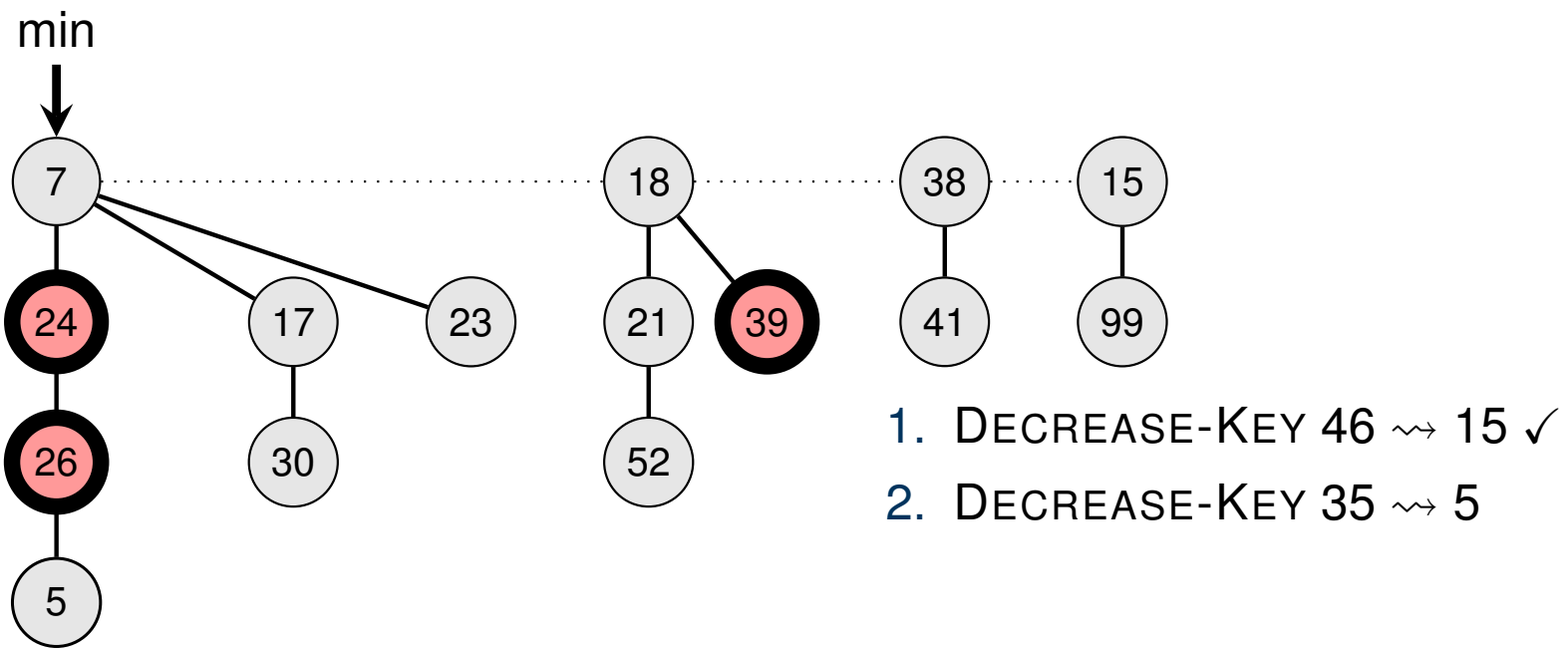
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

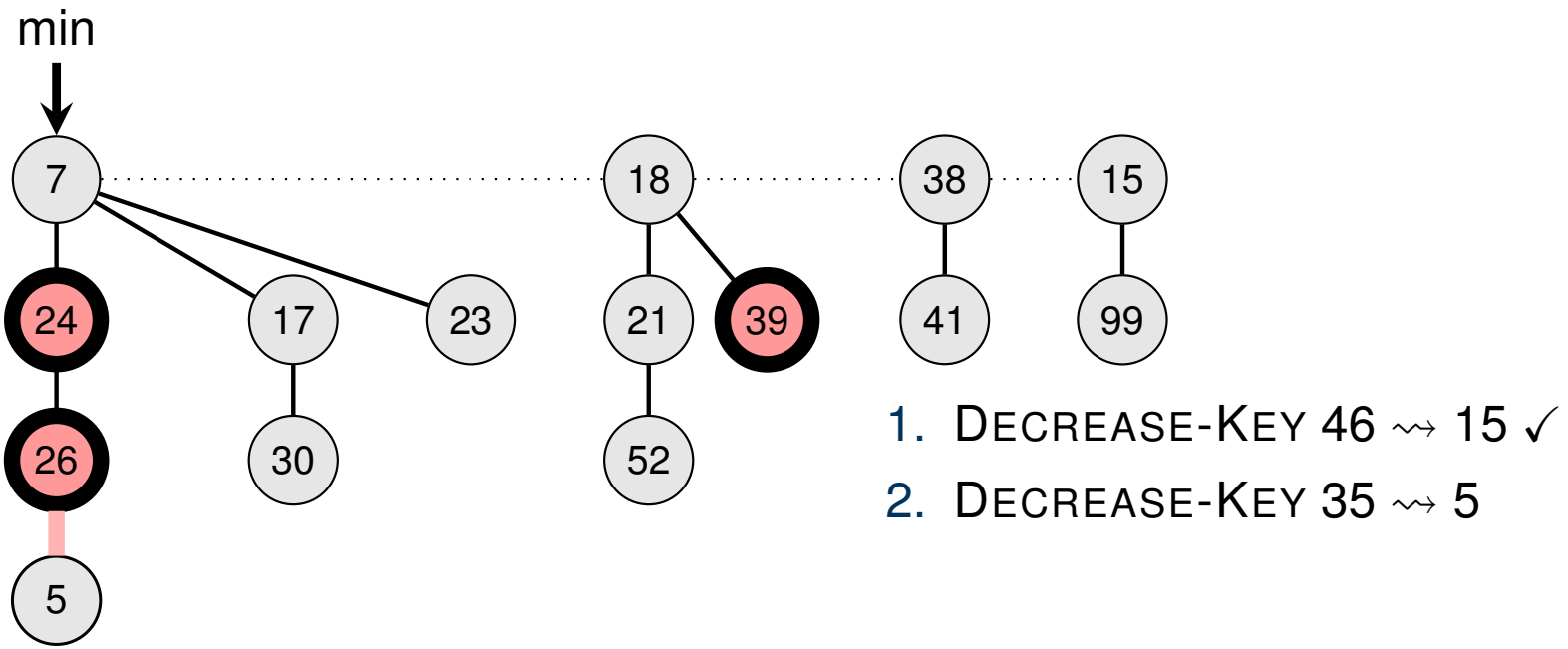
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

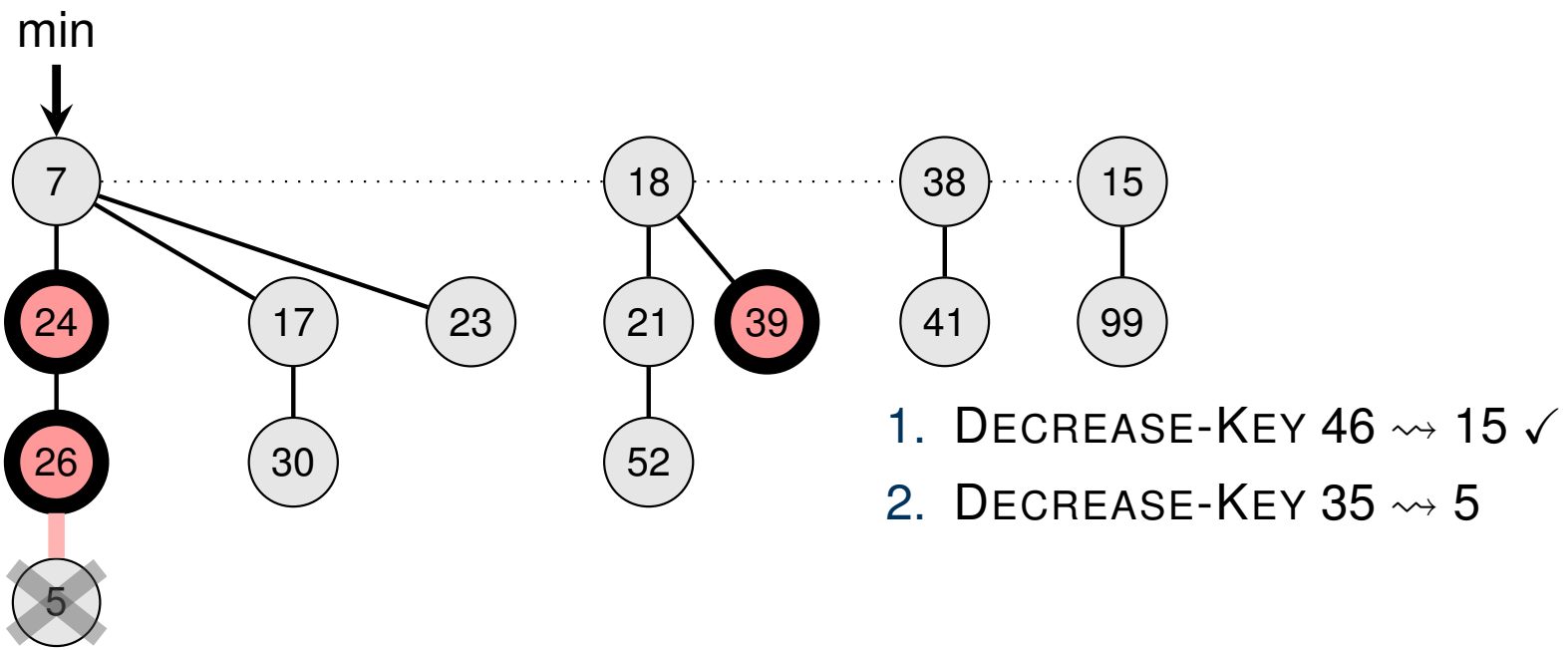
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

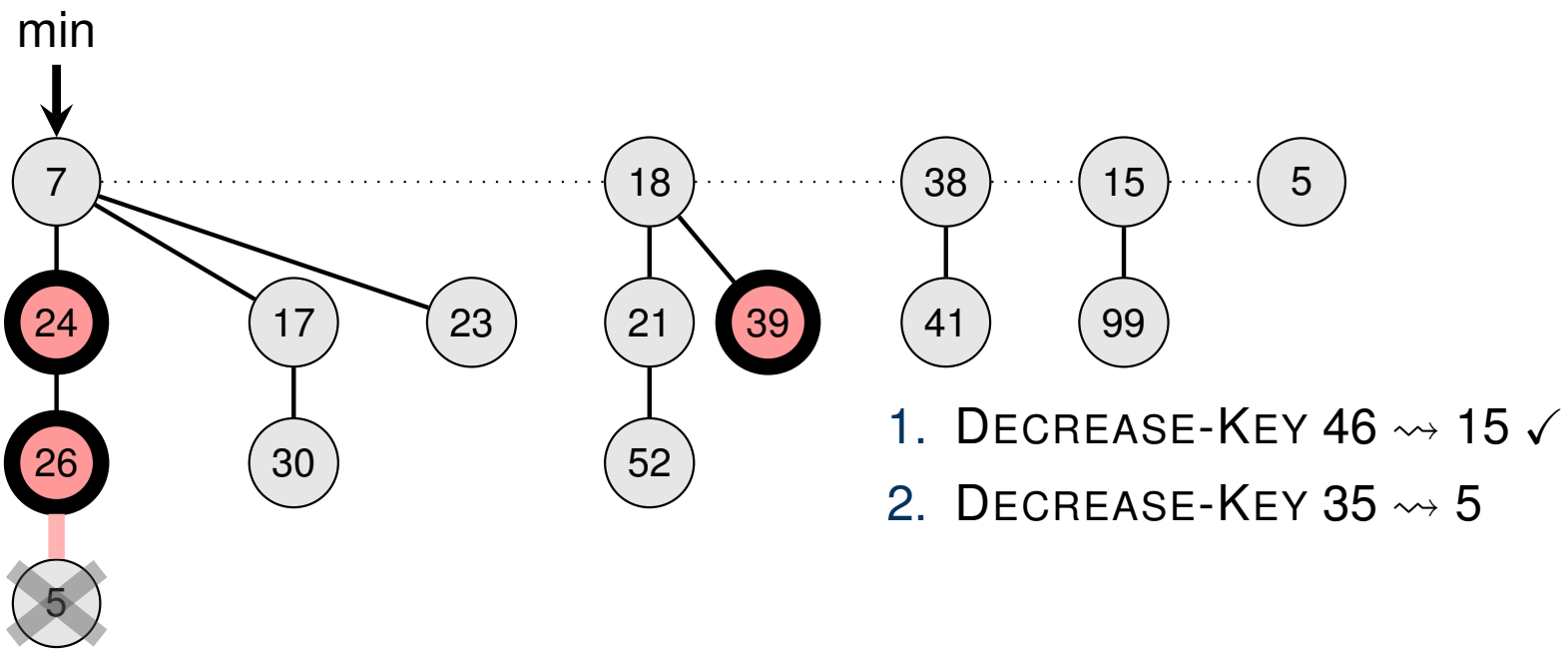
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



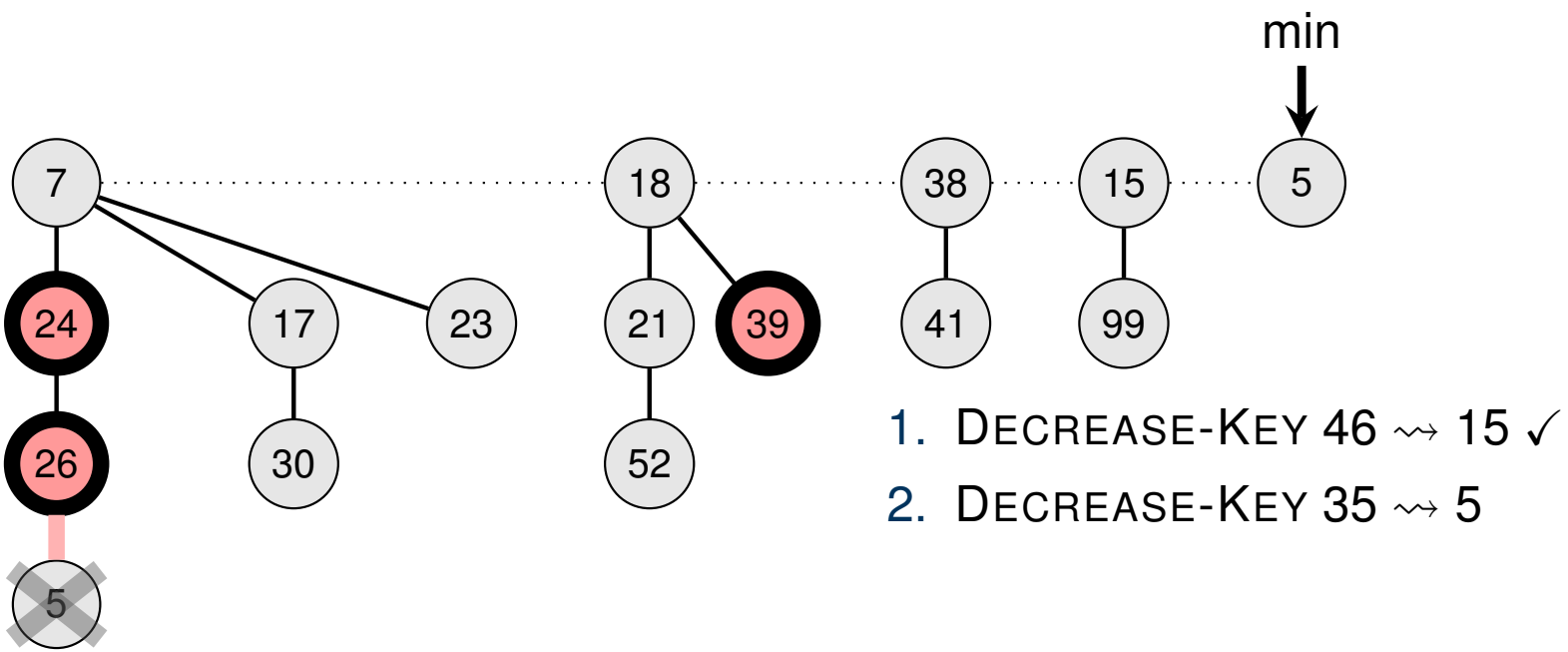
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked,



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

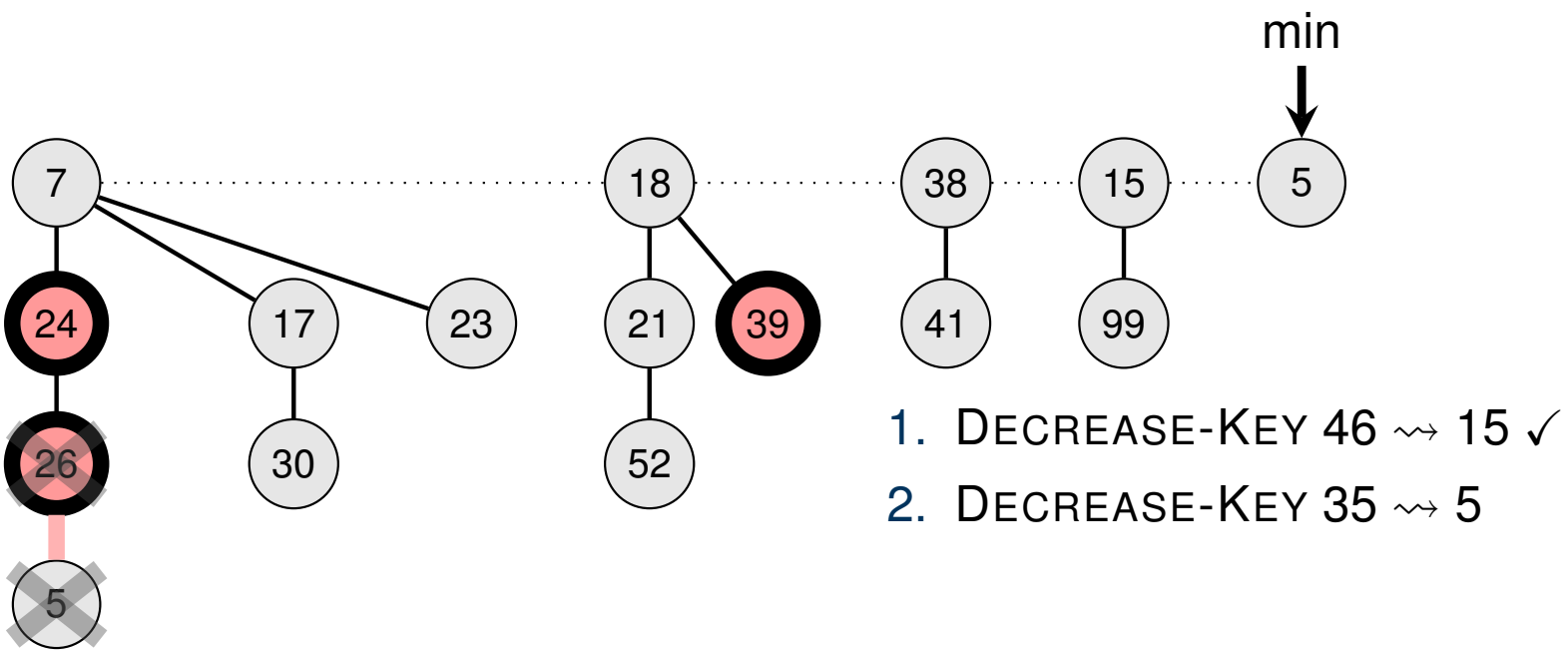
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



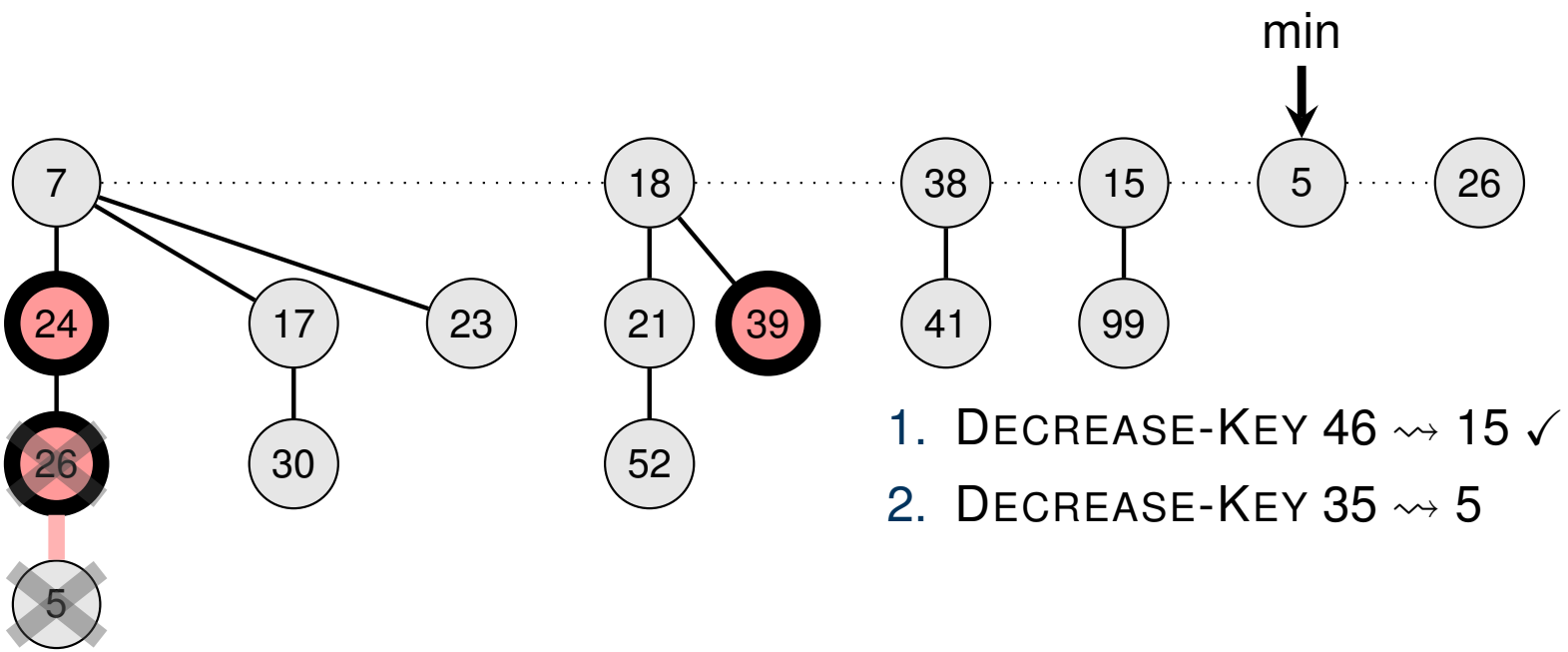
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



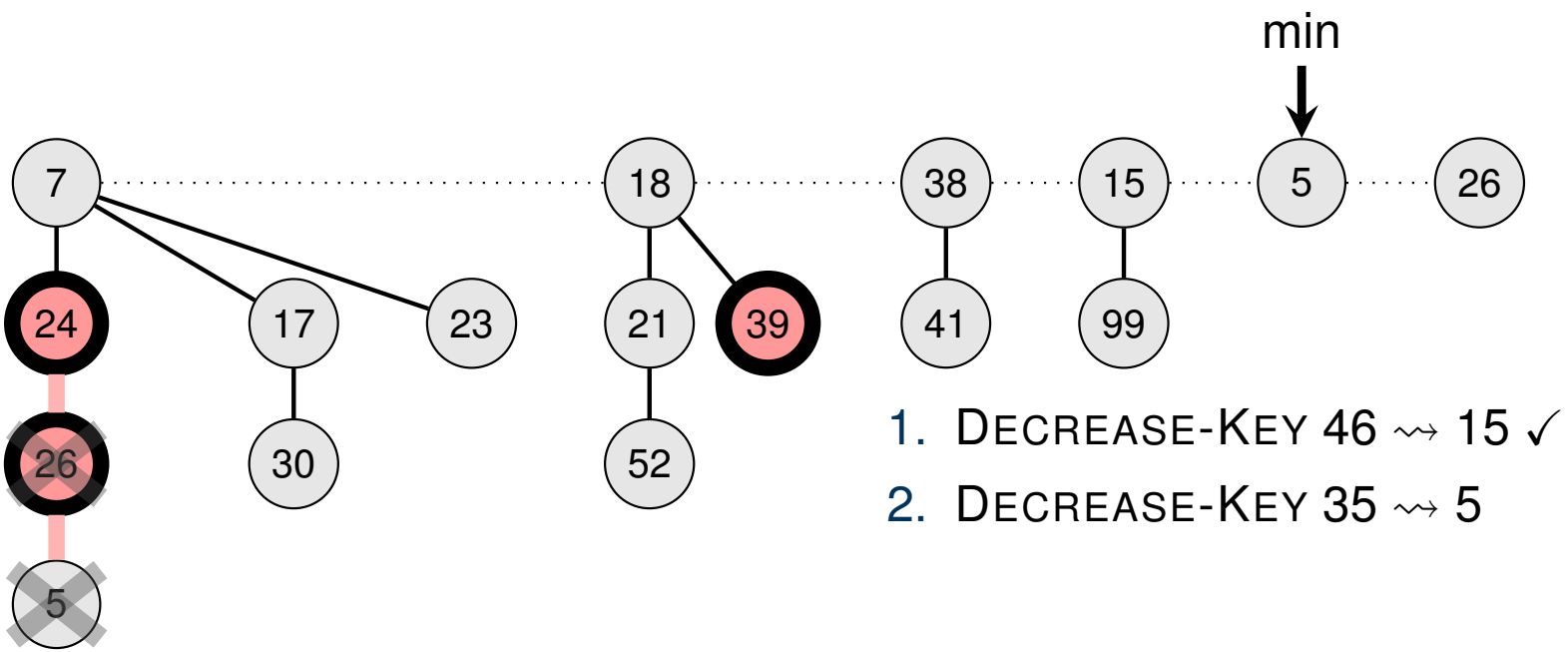
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



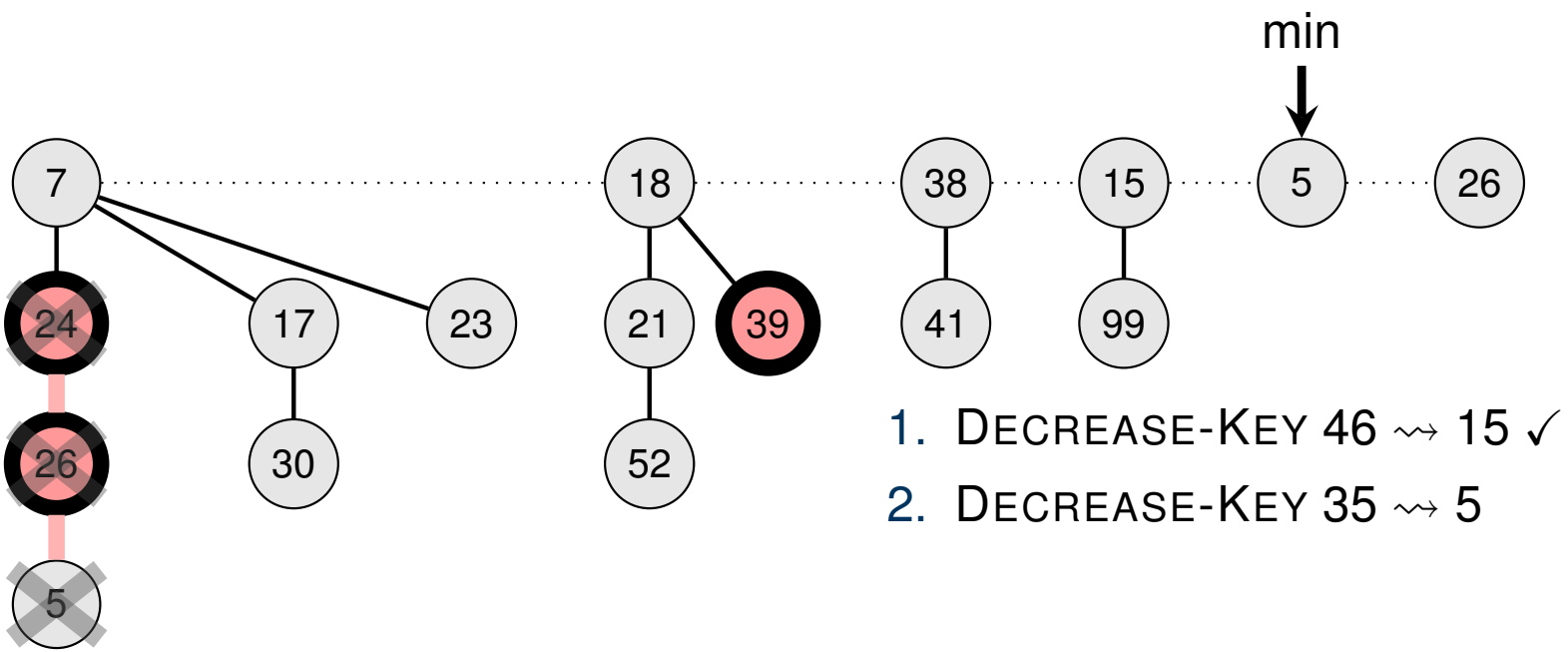
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



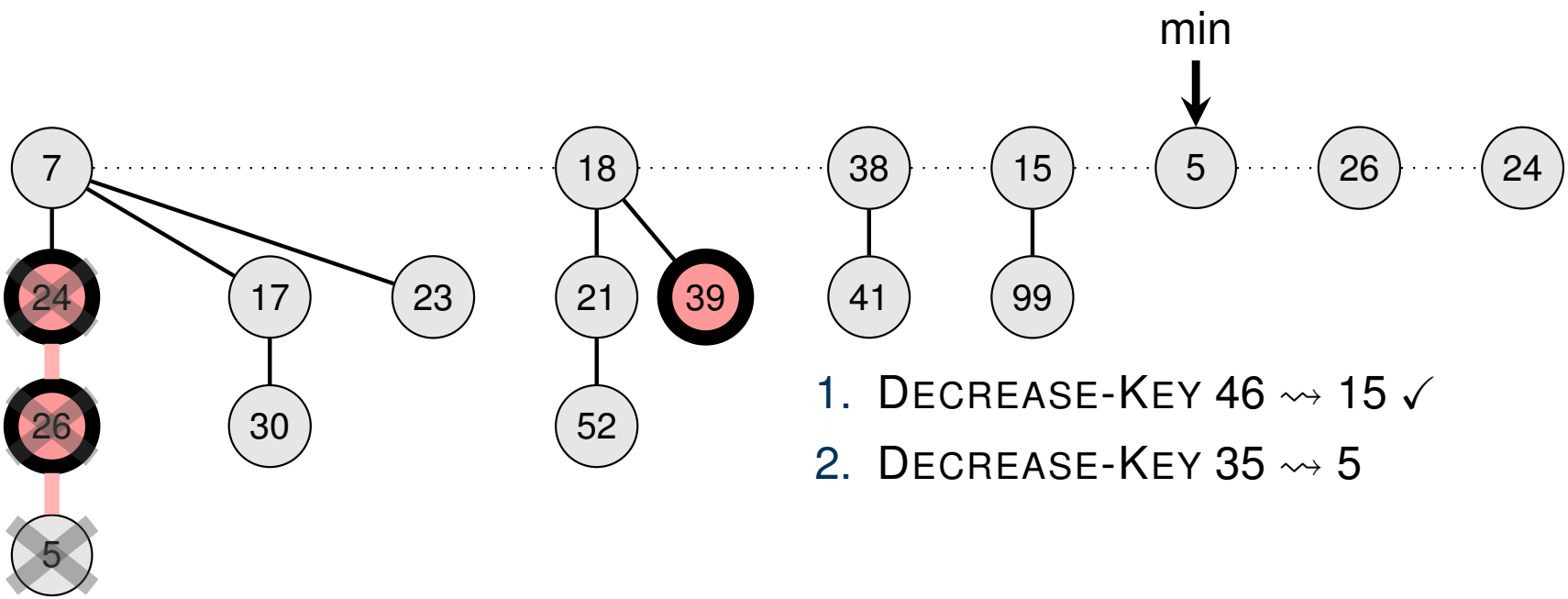
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



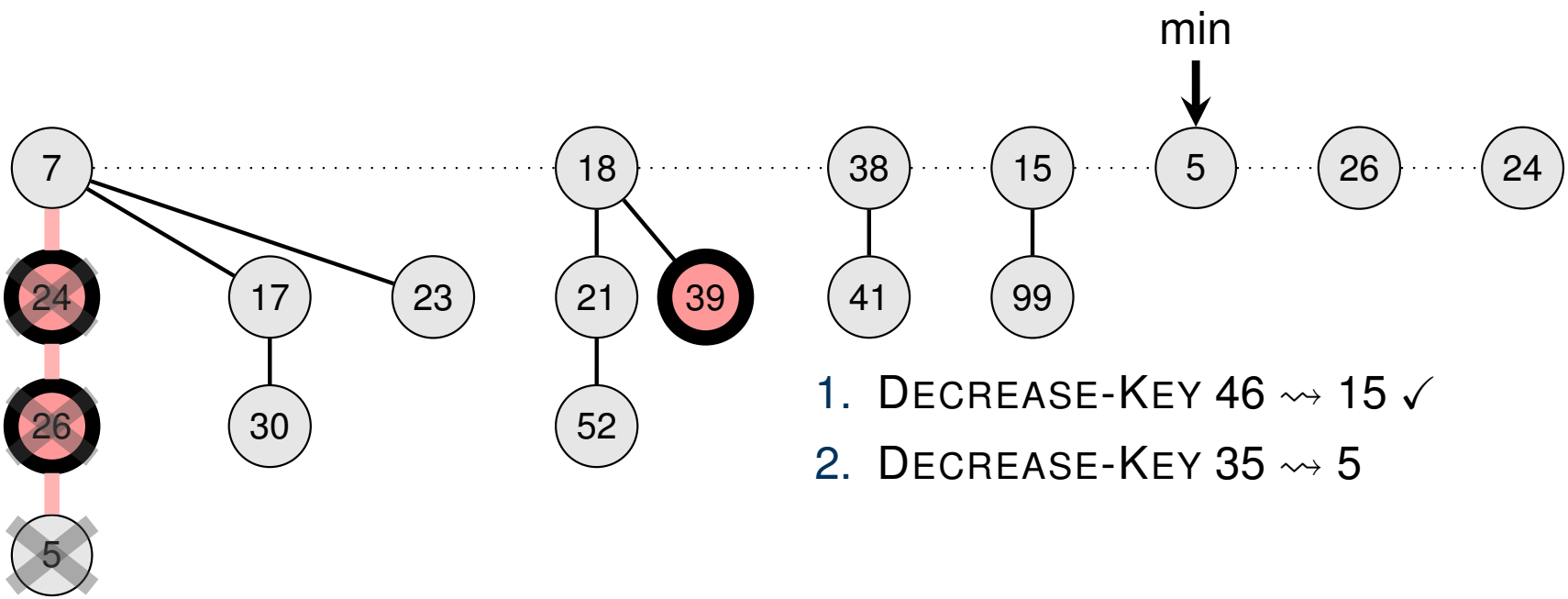
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



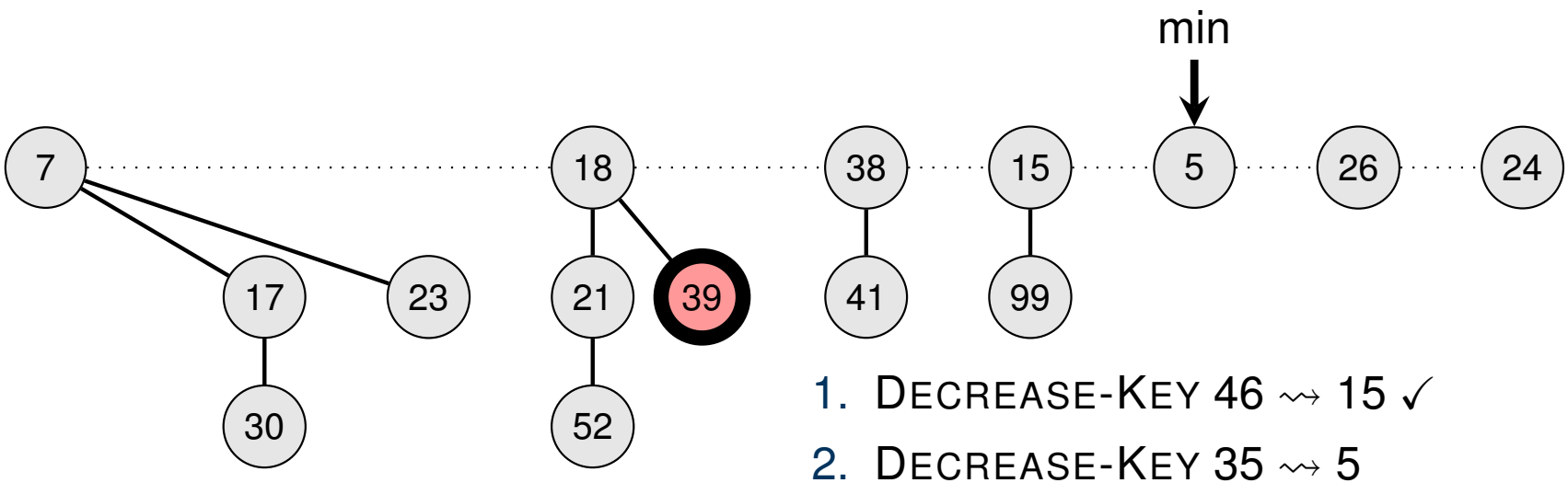
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

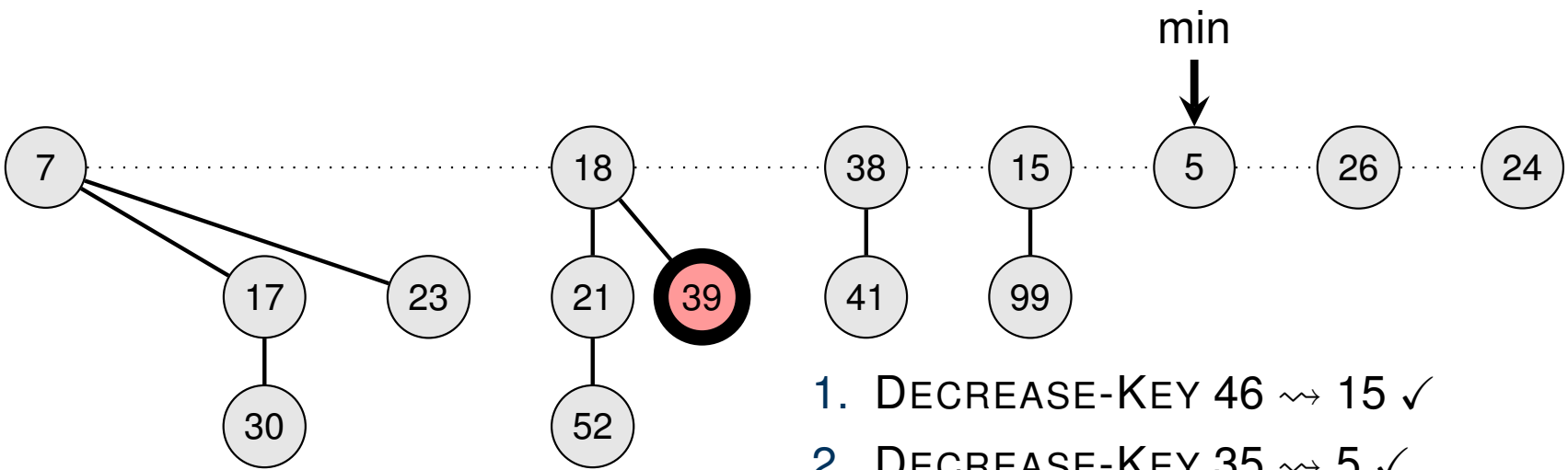
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5 ✓

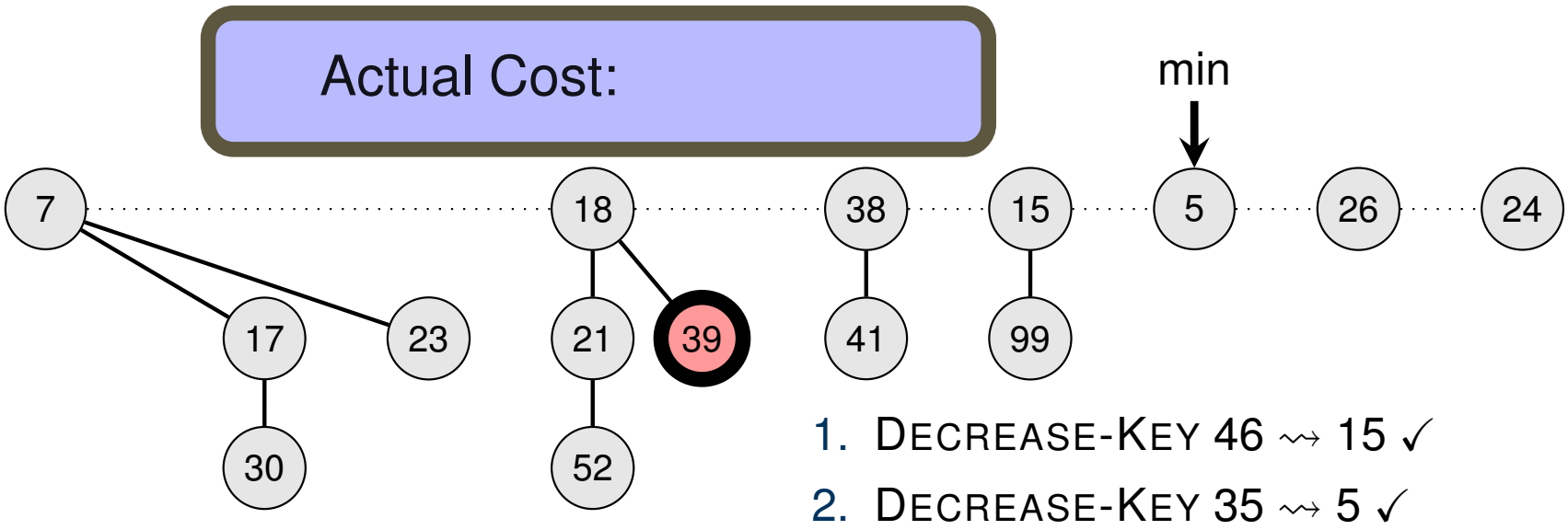
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



Fibonacci Heap: DECREASE-KEY

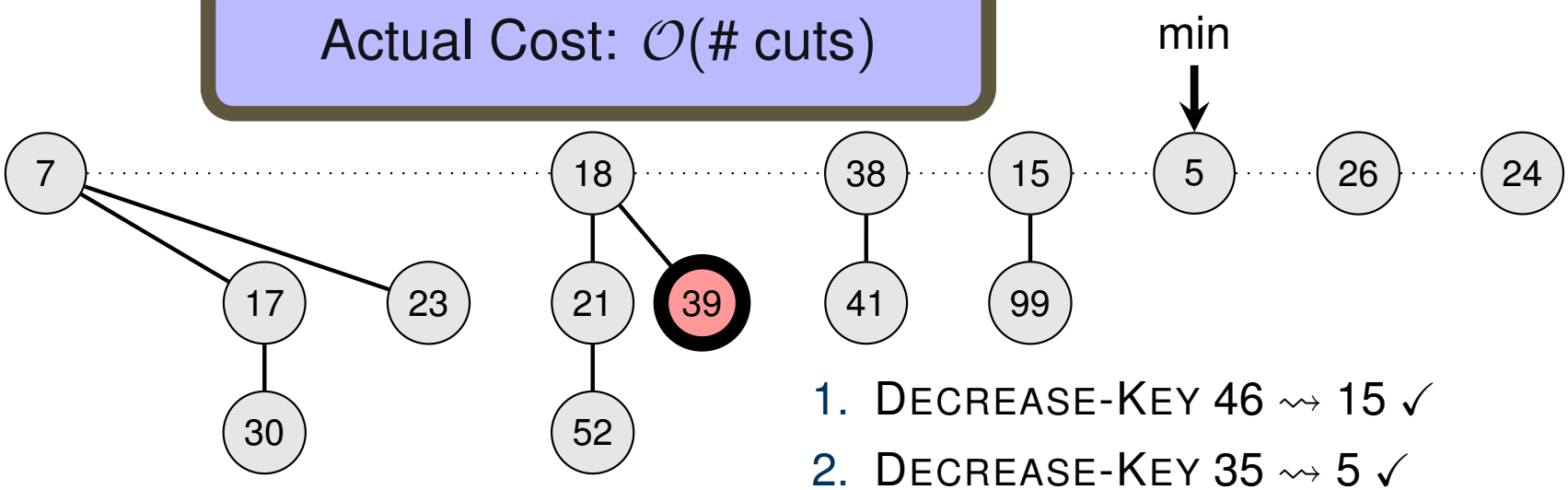
DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

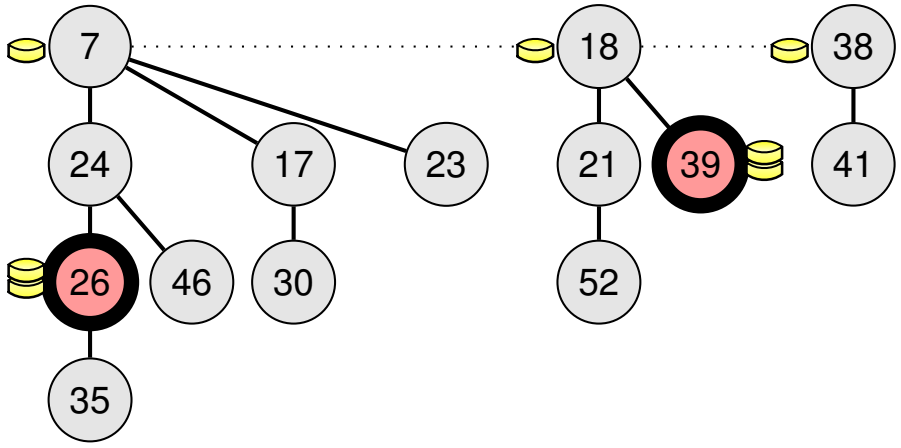
$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)

Actual Cost: $\mathcal{O}$(# cuts)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5 ✓



5.2 Fibonacci Heaps (Analysis)

Frank Stajano

Thomas Sauerwald

Lent 2016

UNIVERSITY OF CAMBRIDGE

Outline

Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

Amortized Analysis via Potential Method

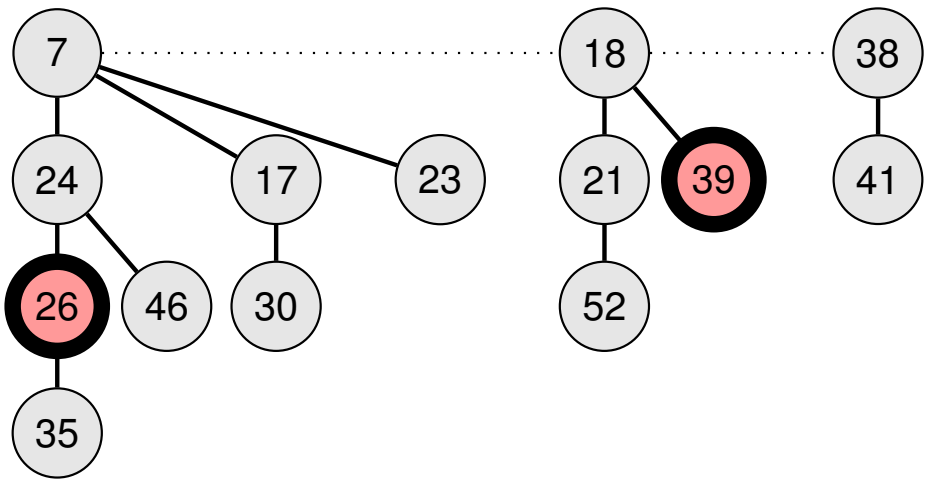
- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

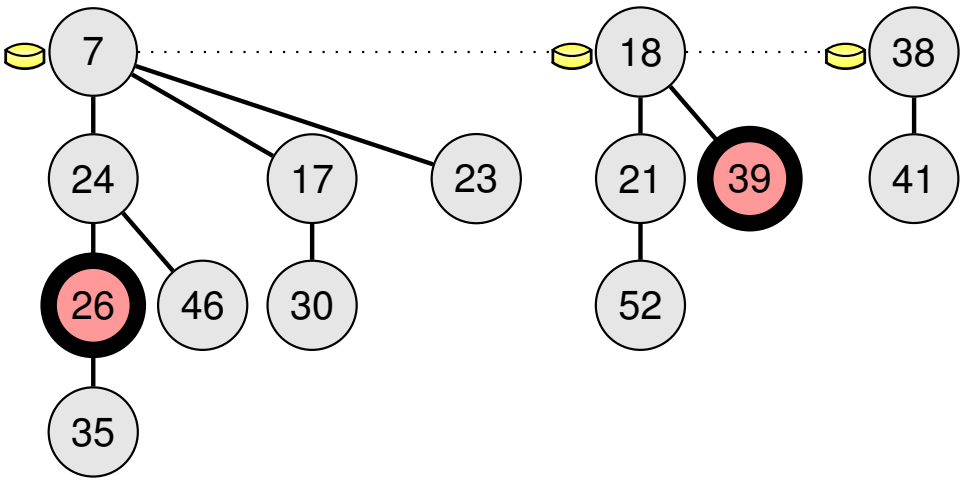
$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$



Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

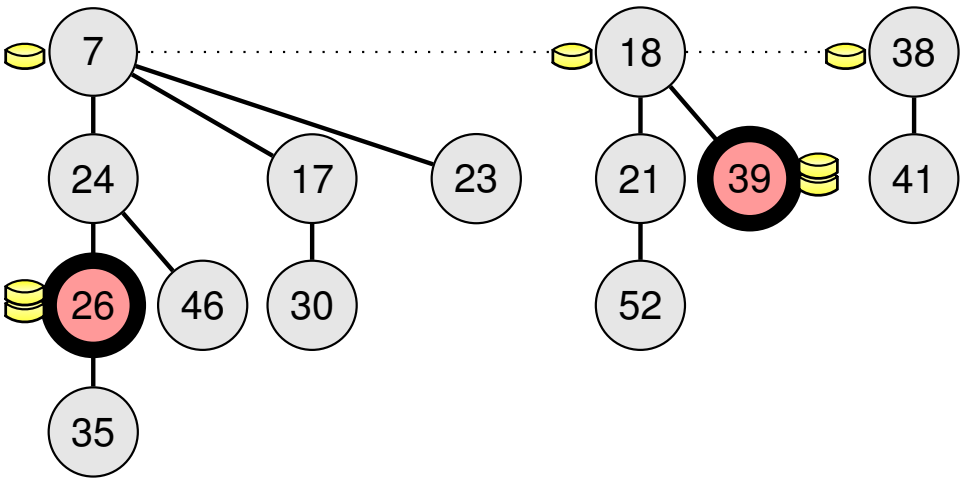
$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$



Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

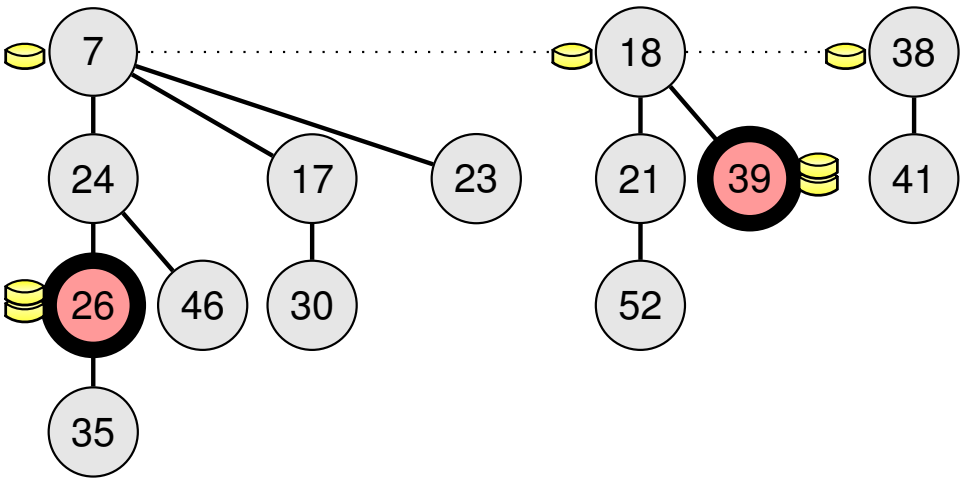
$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$



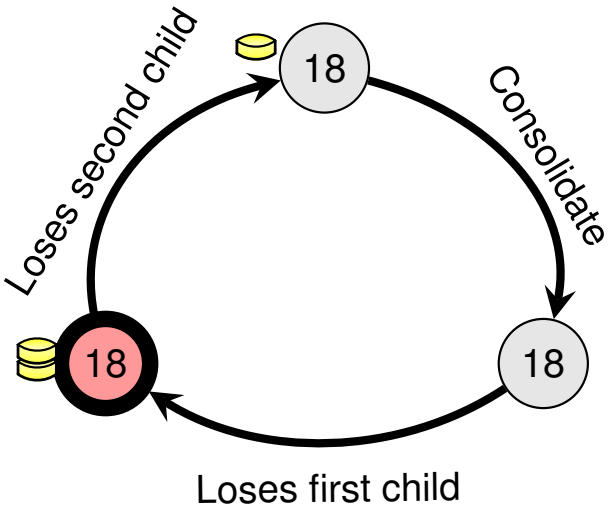
Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$ amortized $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$ amortized $\mathcal{O}(d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$ amortized $\mathcal{O}(1)$

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$



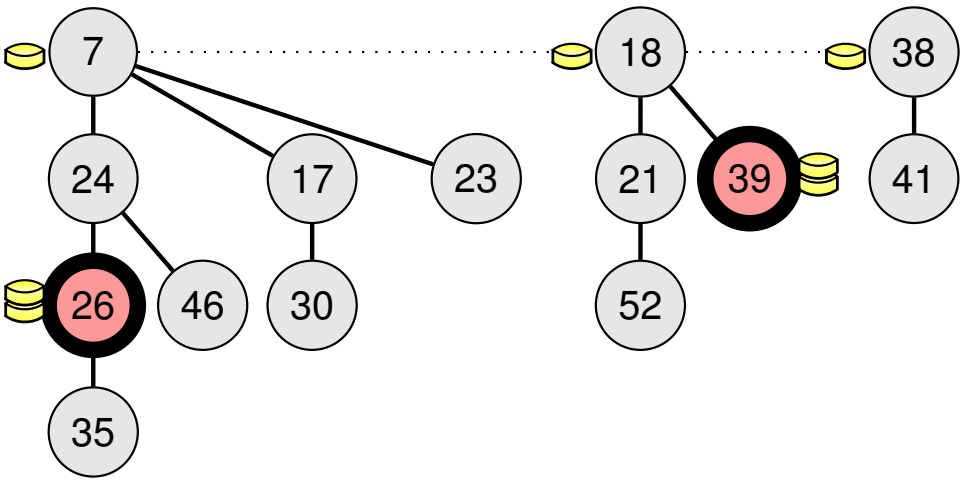
Lifecycle of a node



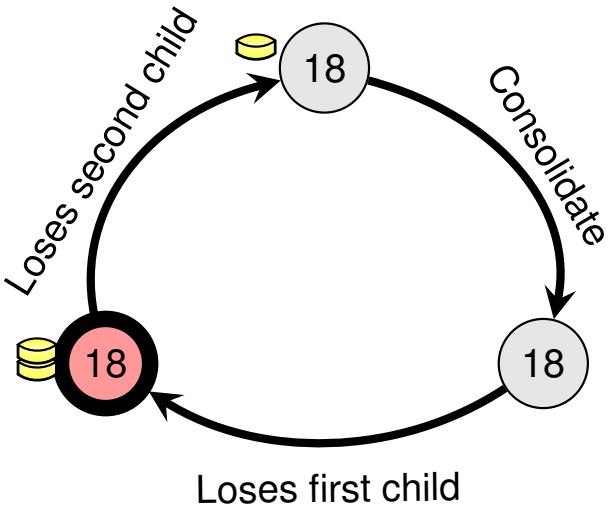
Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$ amortized $\mathcal{O}(1)$ ✓
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$ amortized $\mathcal{O}(d(n))$?
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$ amortized $\mathcal{O}(1)$?

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$



Lifecycle of a node



Outline

Amortized Analysis of DECREASE-KEY

Actual Cost

- DECREASE-KEY: $\mathcal{O}(x + 1)$, where x is the number of cuts.

Amortized Analysis of DECREASE-KEY

Actual Cost

- DECREASE-KEY: $\mathcal{O}(x + 1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

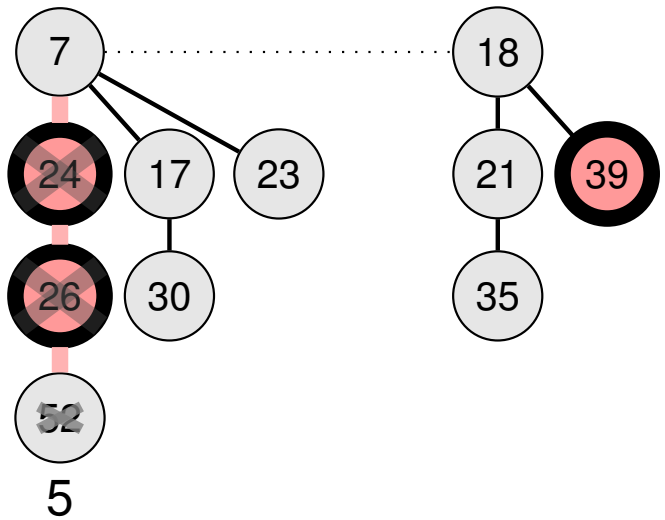
Amortized Analysis of DECREASE-KEY

Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential



Amortized Analysis of DECREASE-KEY

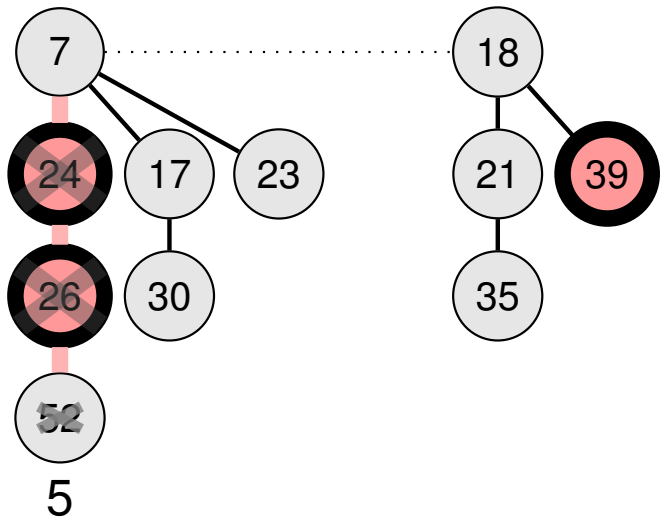
Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential

- $\text{trees}(H') =$



Amortized Analysis of DECREASE-KEY

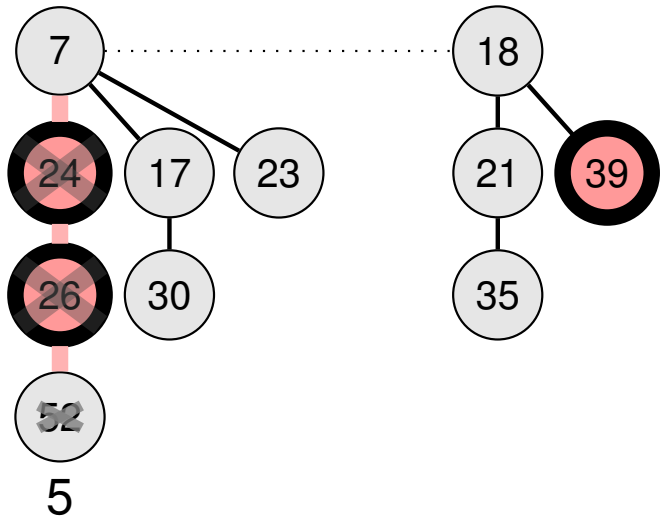
Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$



Amortized Analysis of DECREASE-KEY

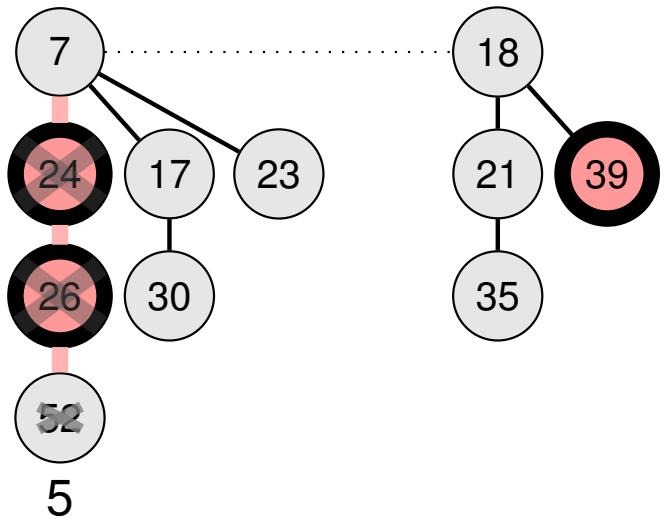
Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq$



Amortized Analysis of DECREASE-KEY

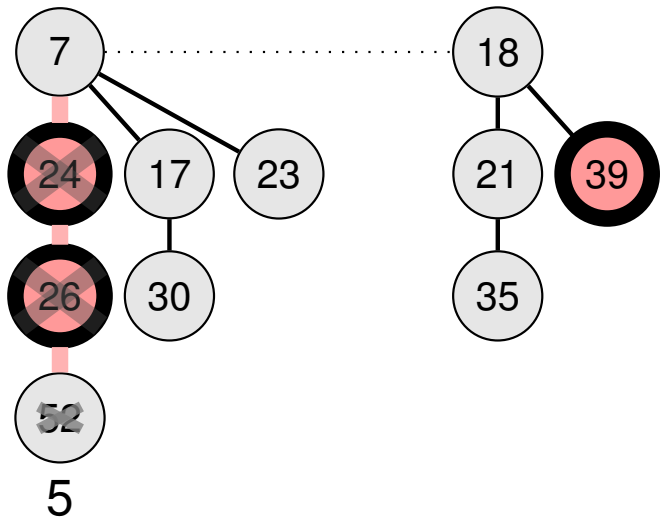
Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$



Amortized Analysis of DECREASE-KEY

Actual Cost

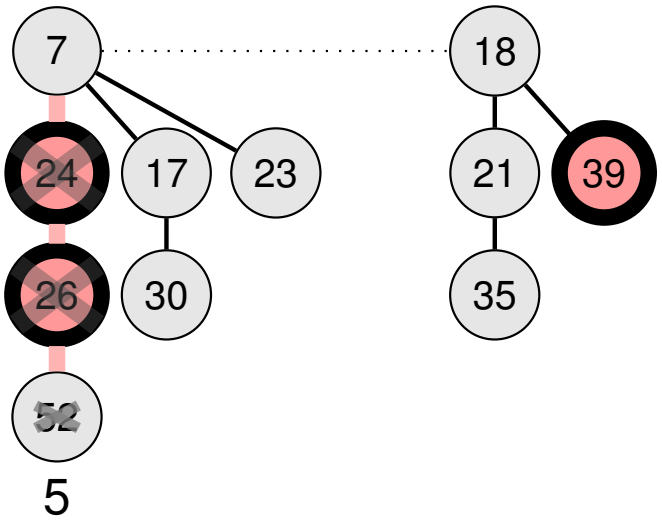
- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \quad \Delta\Phi \leq x + 2 \cdot (-x+2) = 4 - x.$



Amortized Analysis of DECREASE-KEY

Actual Cost

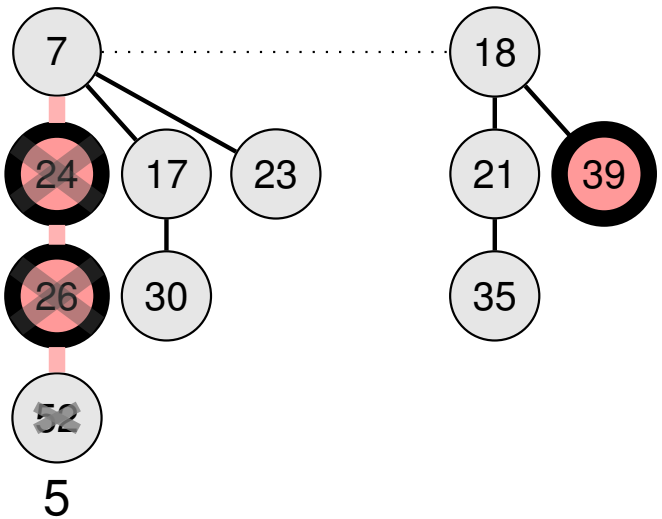
- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \quad \Delta\Phi \leq x + 2 \cdot (-x+2) = 4 - x.$



Amortized Cost

$$\widehat{c}_i = c_i + \Delta\Phi$$

Amortized Analysis of DECREASE-KEY

Actual Cost

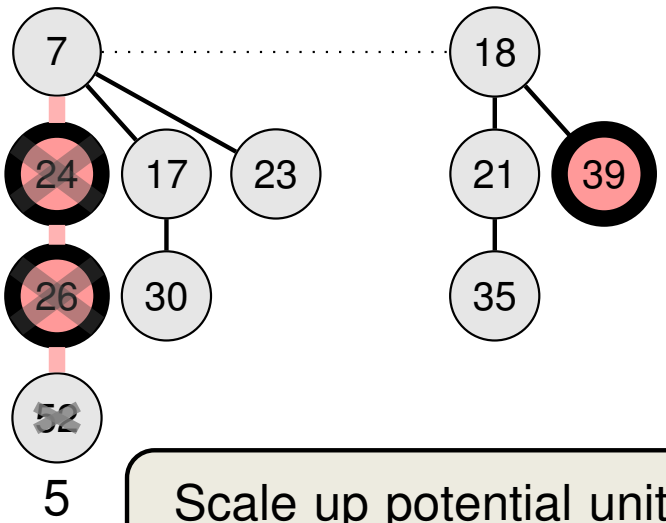
- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \quad \Delta\Phi \leq x + 2 \cdot (-x+2) = 4 - x.$



Scale up potential units

Amortized Cost

$$\widehat{c}_i = c_i + \Delta\Phi \leq \mathcal{O}(x+1) + 4 - x$$

Amortized Analysis of DECREASE-KEY

Actual Cost

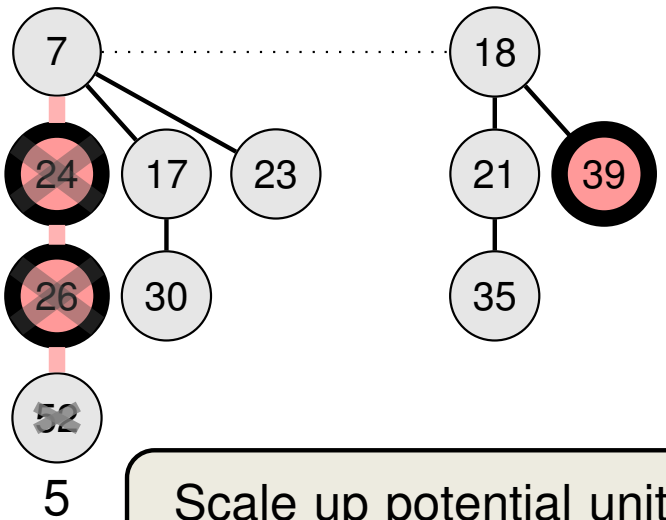
- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \quad \Delta\Phi \leq x + 2 \cdot (-x+2) = 4 - x.$



Scale up potential units

Amortized Cost

$$\widehat{c}_i = c_i + \Delta\Phi \leq \mathcal{O}(x+1) + 4 - x = \mathcal{O}(1)$$

Amortized Analysis of DECREASE-KEY

Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

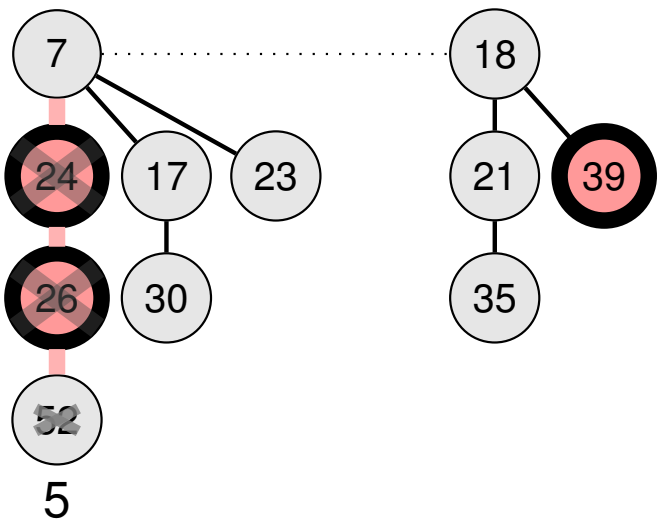
$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

First Coin $\rightsquigarrow$ pays cut
Second Coin $\rightsquigarrow$ increase of trees(H)

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \quad \Delta\Phi \leq x + 2 \cdot (-x+2) = 4 - x.$



Amortized Cost

$$\widehat{c}_i = c_i + \Delta\Phi \leq \mathcal{O}(x+1) + 4 - x = \mathcal{O}(1)$$