Mergeable Heaps

A mergeable heap is any data structure that supports the following five operations, in which each element has a key:

MAKE-HEAP (): Creates and returns a new heap containing no elements.

INSERT(H, x): Inserts element x, whose key has already been filled in, into heap H.

MINIMUM(H): Returns a pointer to the element in heap H whose key is minimum.

EXTRACT-MIN(H):Deletes the element from heap H whose key is minimum, returning a pointer to the element.

UNION(H1, H2): Creates and returns a new heap that contains all the elements of heaps H1 and H2. Heaps H1 and H2 are “destroyed” by this operation.

Mergeable Heaps

In addition to the mergeable-heap operations above, Fibonacci heaps also support the following two operations:

DECREASE-KEY(H.x, k) : Assigns to element x within heap H the new key value k, which we assume to be no greater than its current key value.

DELETE(H, x) : Deletes element x from heap H.

Mergeable Heaps Implementation Cost

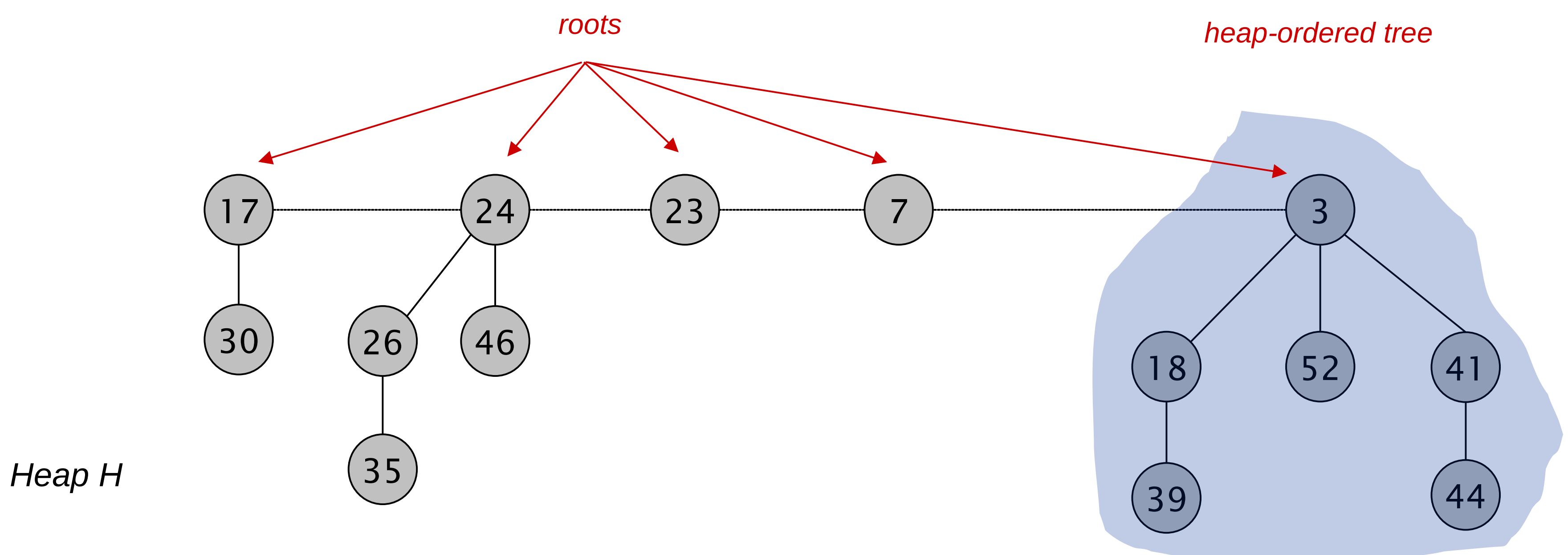
Operations	Binary heap (Worst)	Fibonacci heap(amortized)
Make-Heap	$\theta(1)$	$\theta(1)$
Insert	$\theta(\log n)$	$\theta(1)$
Minimum	$\theta(1)$	$\theta(1)$
Extract-Min	$\theta(\log n)$	$\theta(\log n)$
Union	$\theta(n)$	$\theta(1)$
Decrease-Key	$\theta(\log n)$	$\theta(1)$
Delete	$\theta(\log n)$	$\theta(\log n)$

Fibonacci Heaps: Structure

each parent larger than its children

Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

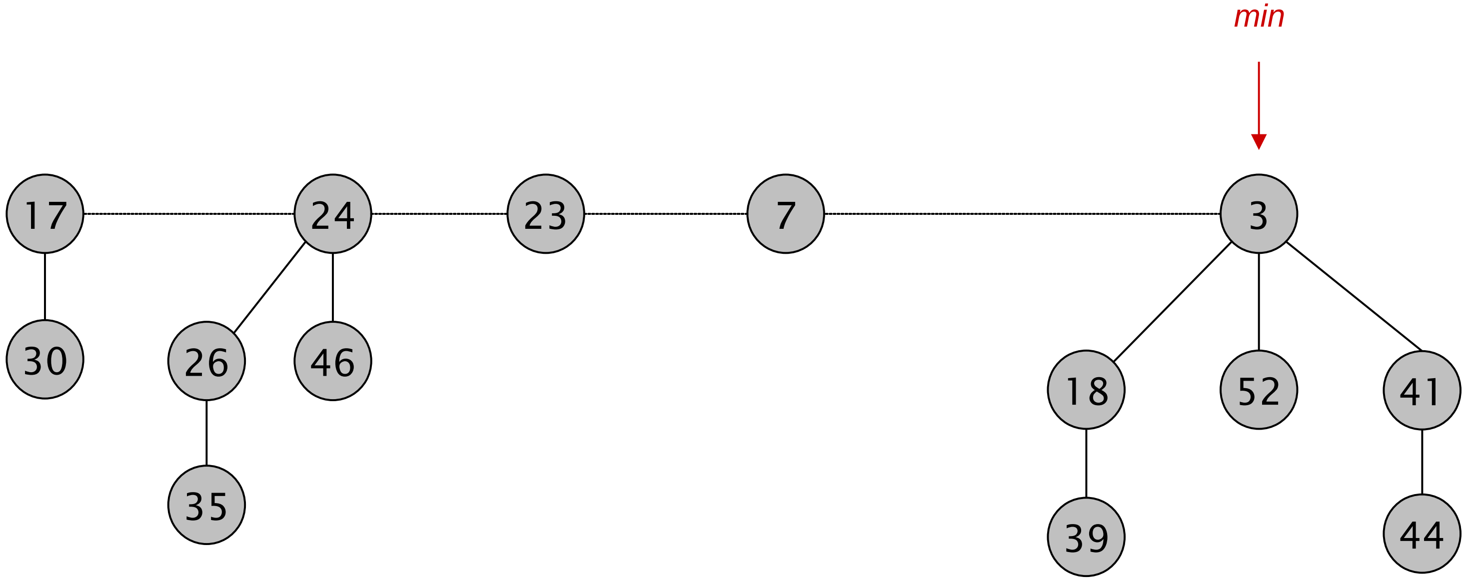


Fibonacci Heaps: Structure

Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

find-min takes O(1) time



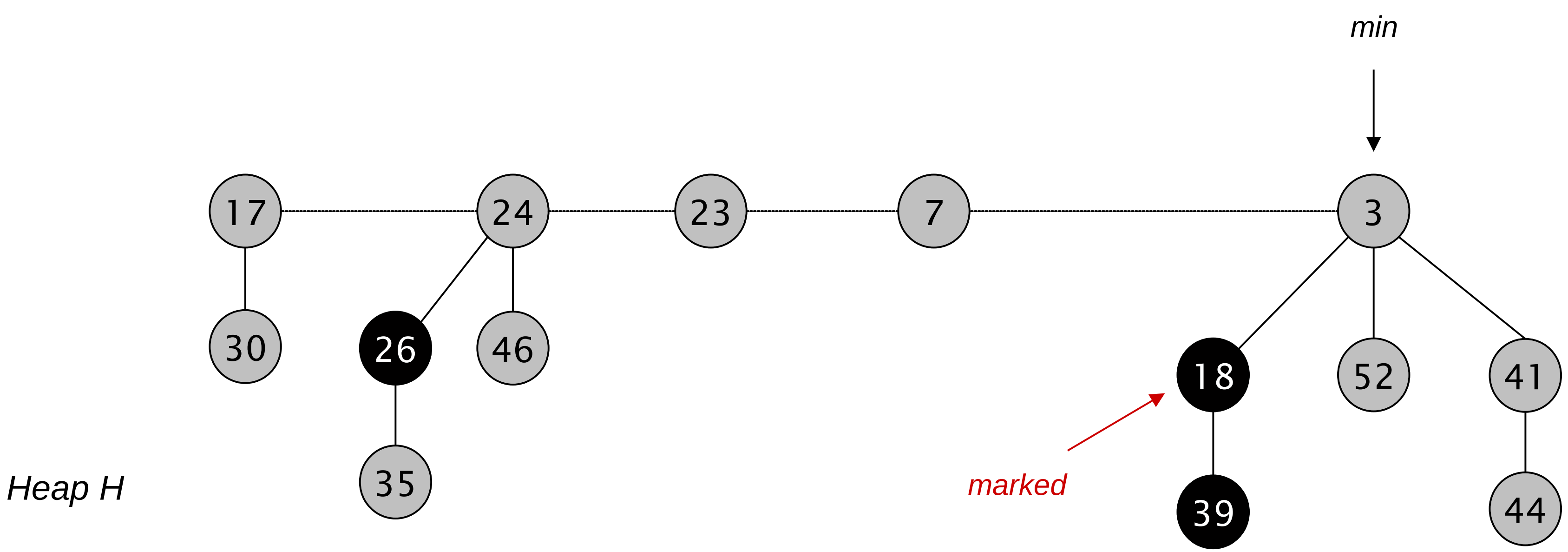
Heap H

Fibonacci Heaps: Structure

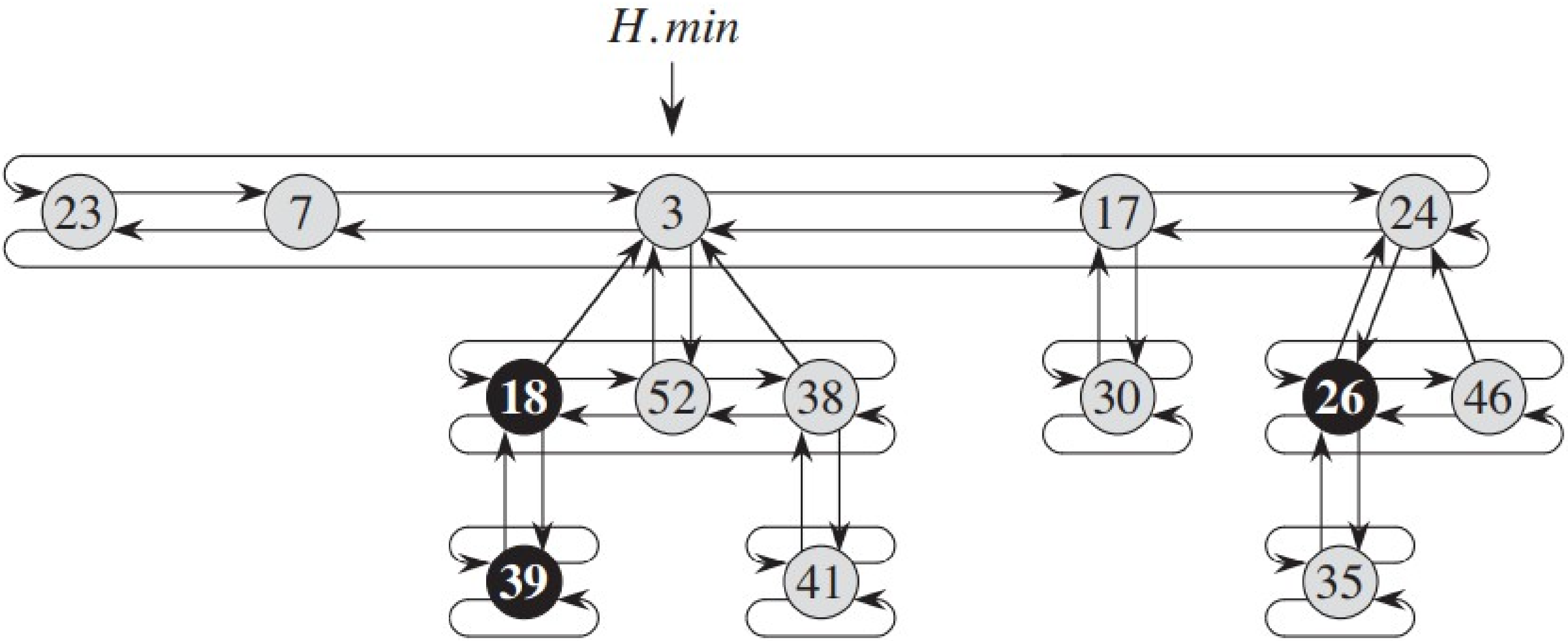
Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

use to keep heaps flat

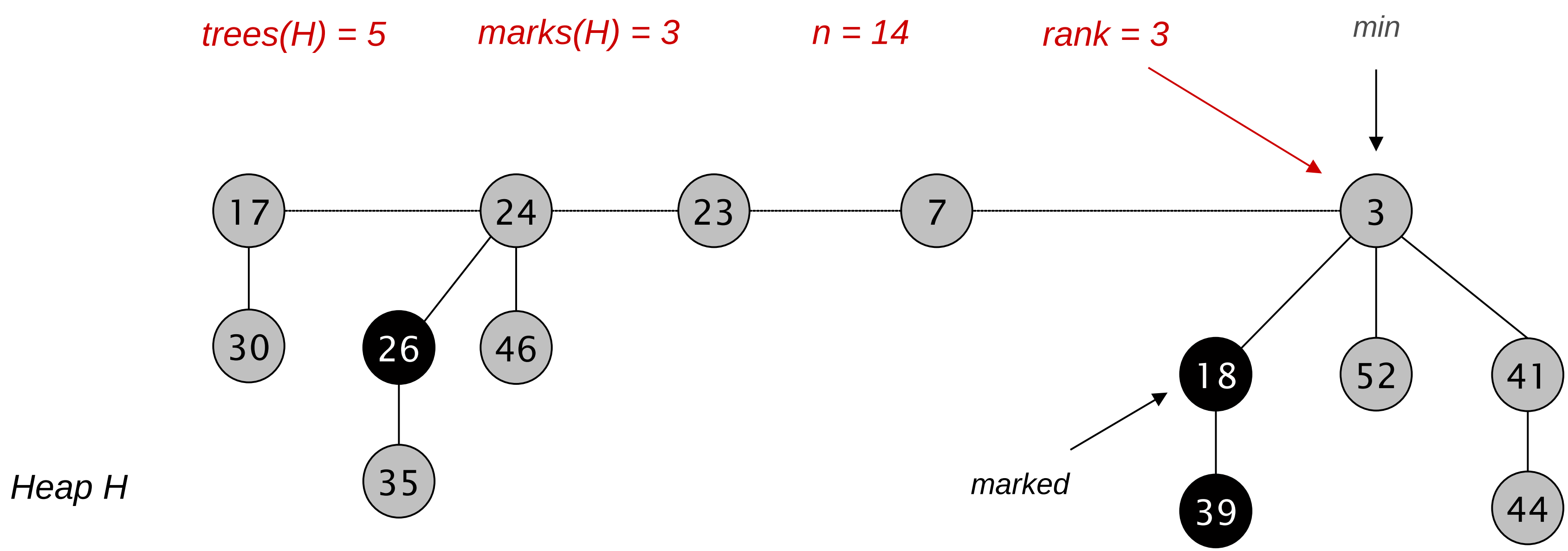


Fibonacci Heaps: Structure Implementation



Notation.

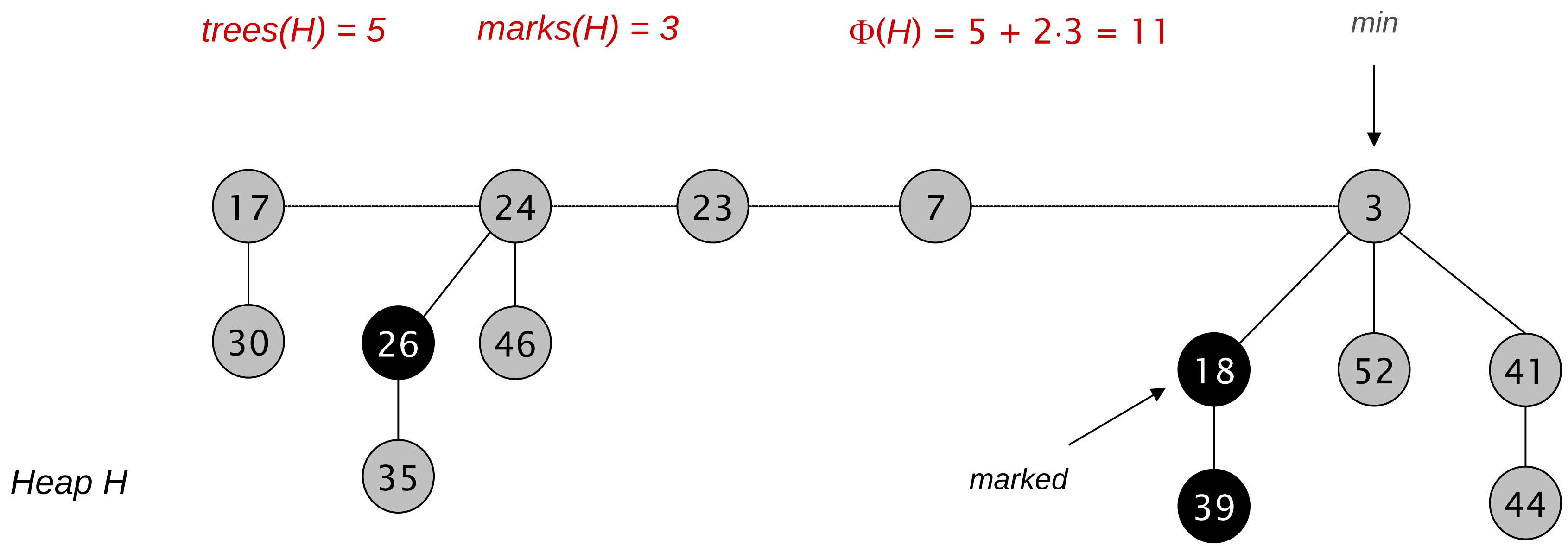
- *n* = number of nodes in heap.
- *rank(x)* = number of children of node *x*.
- *rank(H)* = max rank of any node in heap *H*.
- *trees(H)* = number of trees in heap *H*.
- *marks(H)* = number of marked nodes in heap *H*.



Fibonacci Heaps: Potential Function

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

potential of heap H



Make-Heap (Fibonacci heap)

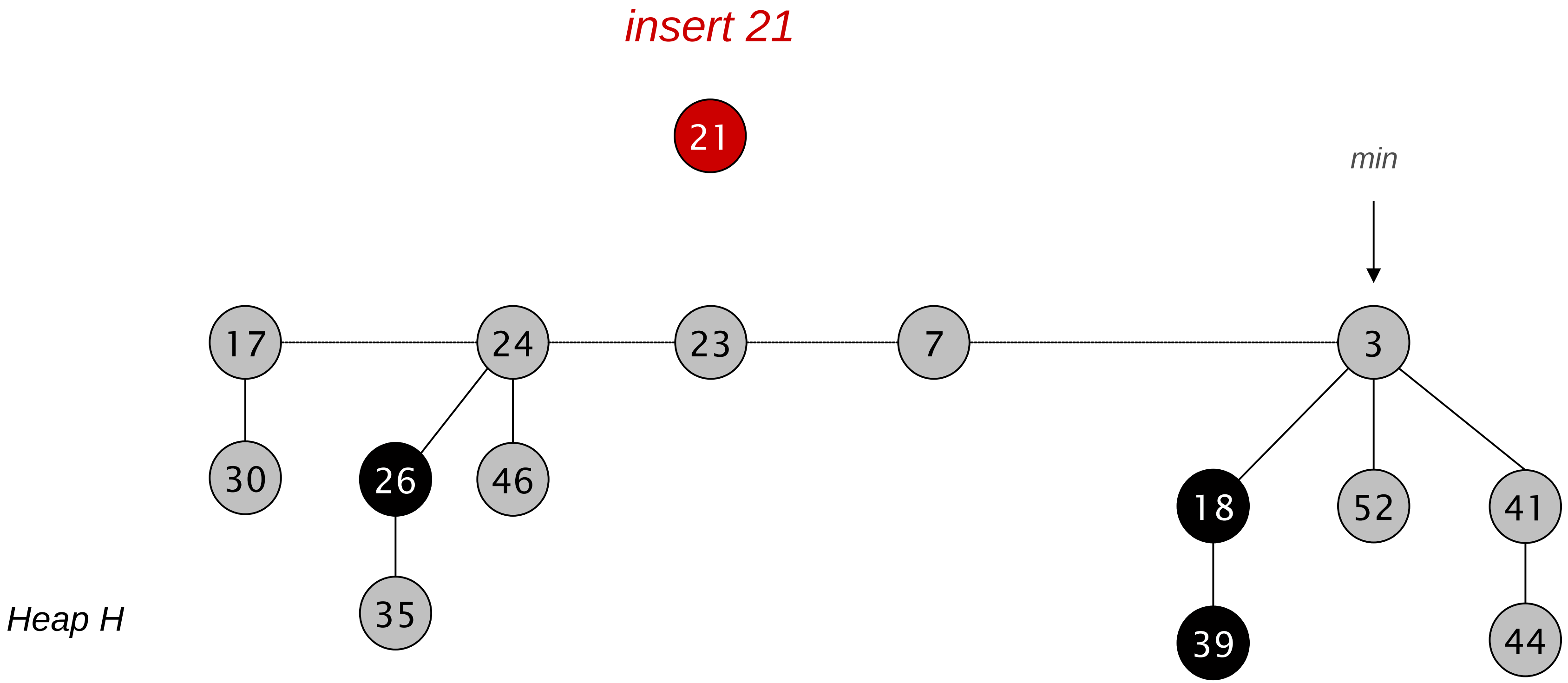
To make an empty Fibonacci heap, the MAKE-FIB-HEAP procedure allocates and returns the Fibonacci heap object H, where H.n=0 and H.min=NIL; there are no trees in H. Because t(H)=0 and m(H)=0, the potential of the empty Fibonacci heap is $\Phi(H)=0$. The amortized cost of MAKE-FIB-HEAP is thus equal to its O(1) actual cost.

Insert

Fibonacci Heaps: Insert

Insert.

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

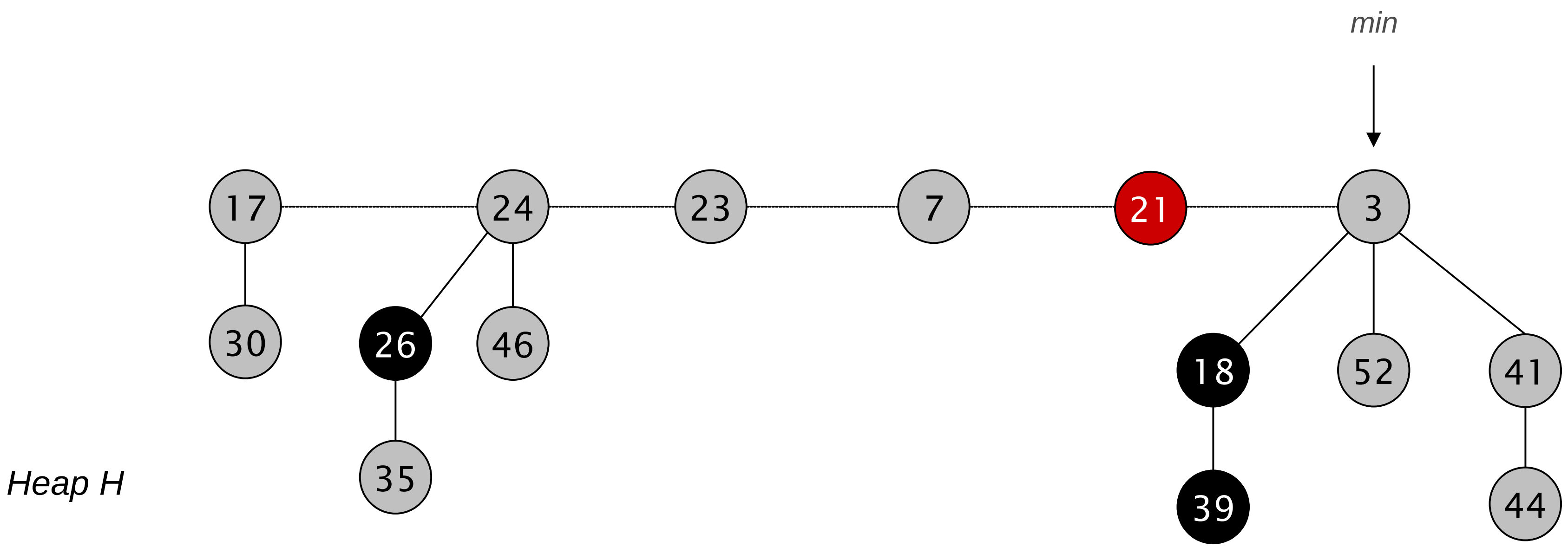


Fibonacci Heaps: Insert

Insert.

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

insert 21



Fibonacci Heaps: Insert Analysis

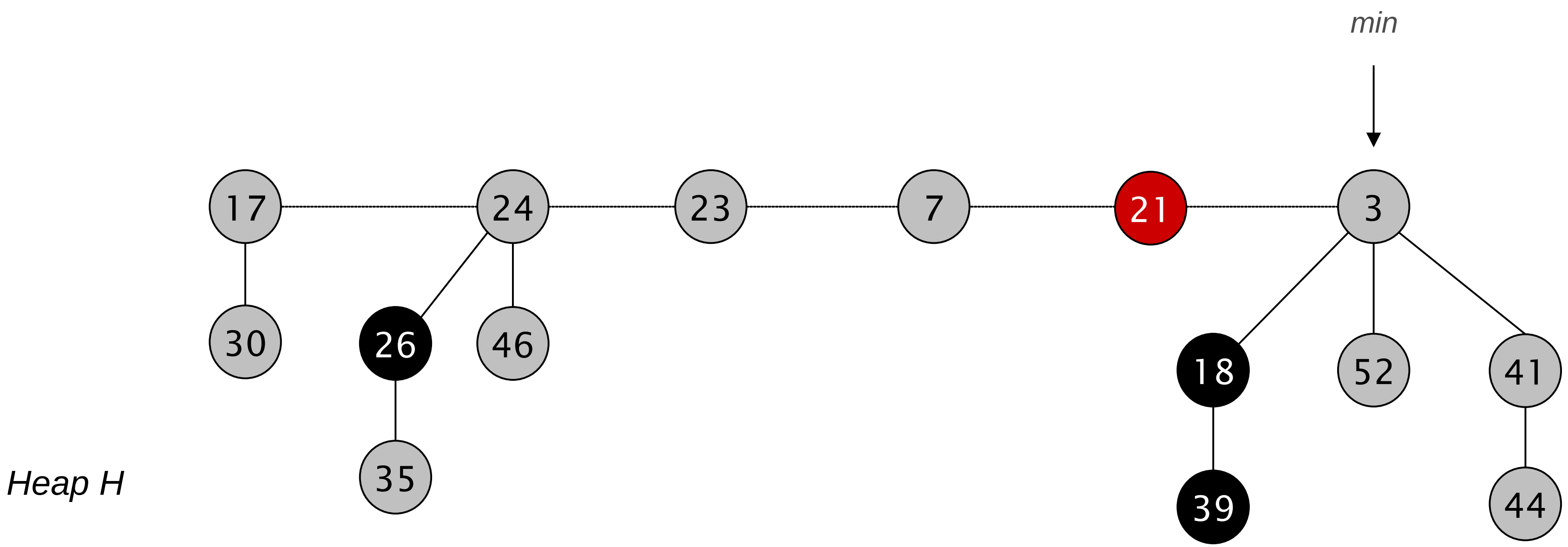
Actual cost. O(1)

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

potential of heap H

Change in potential. +1

Amortized cost. O(1)

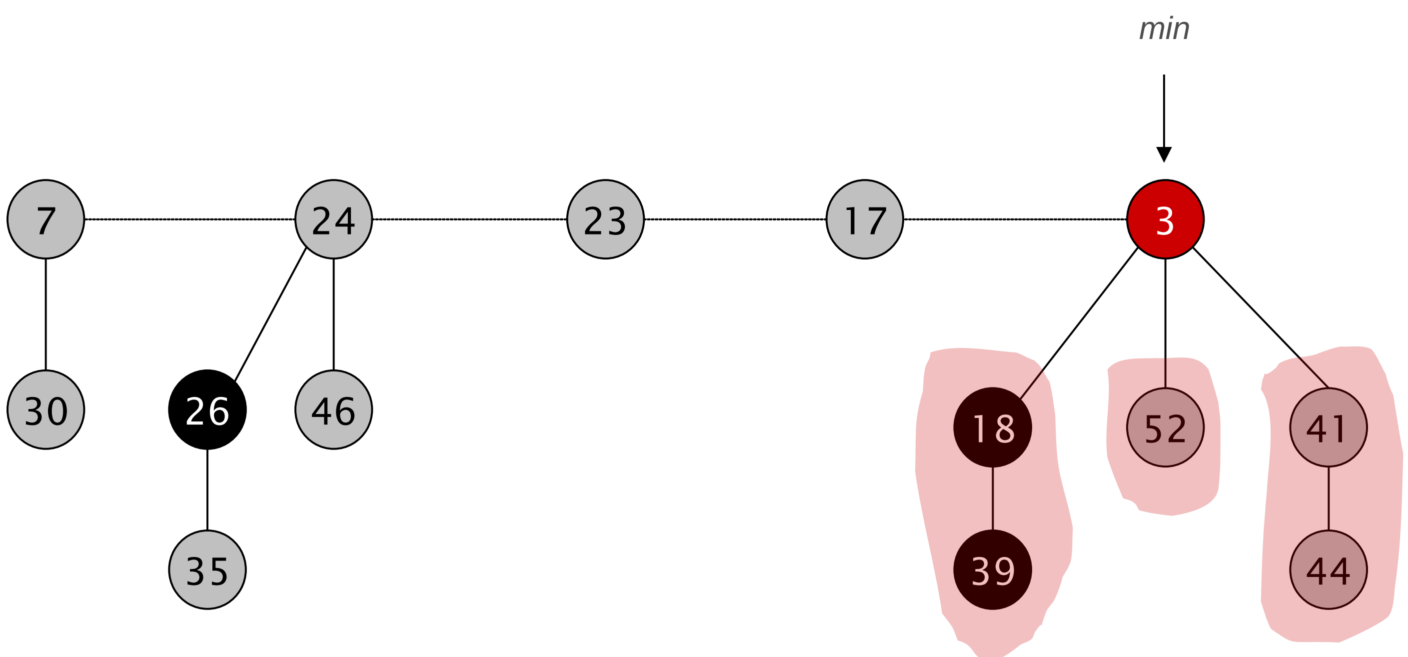


Delete Min

Fibonacci Heaps: Delete Min

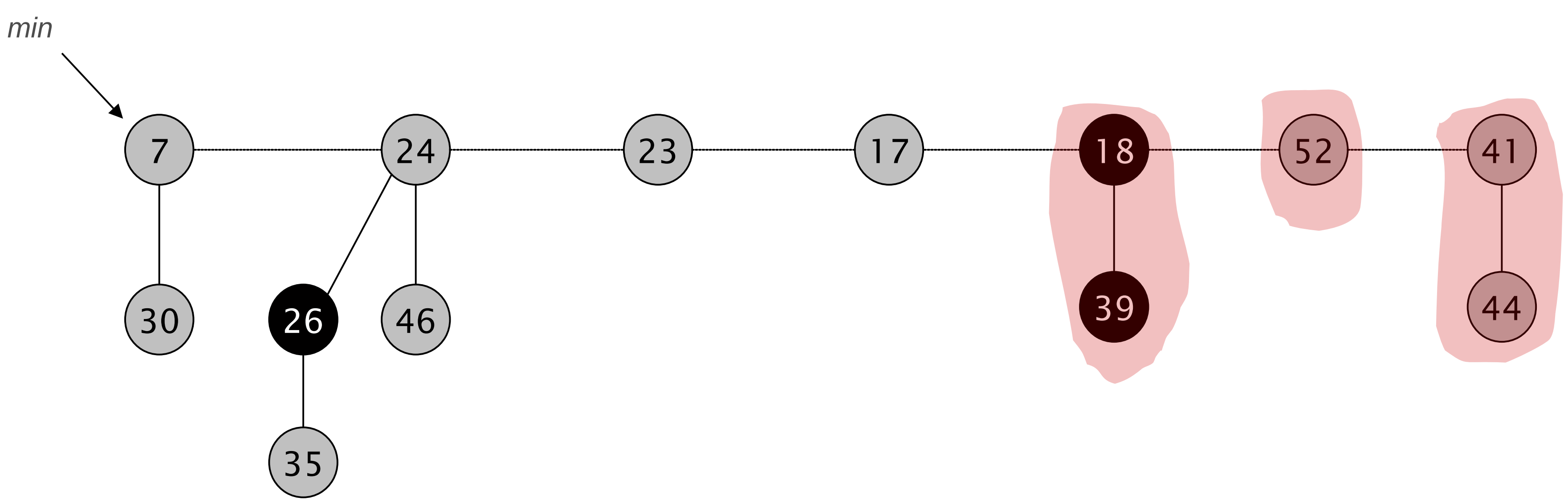
Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

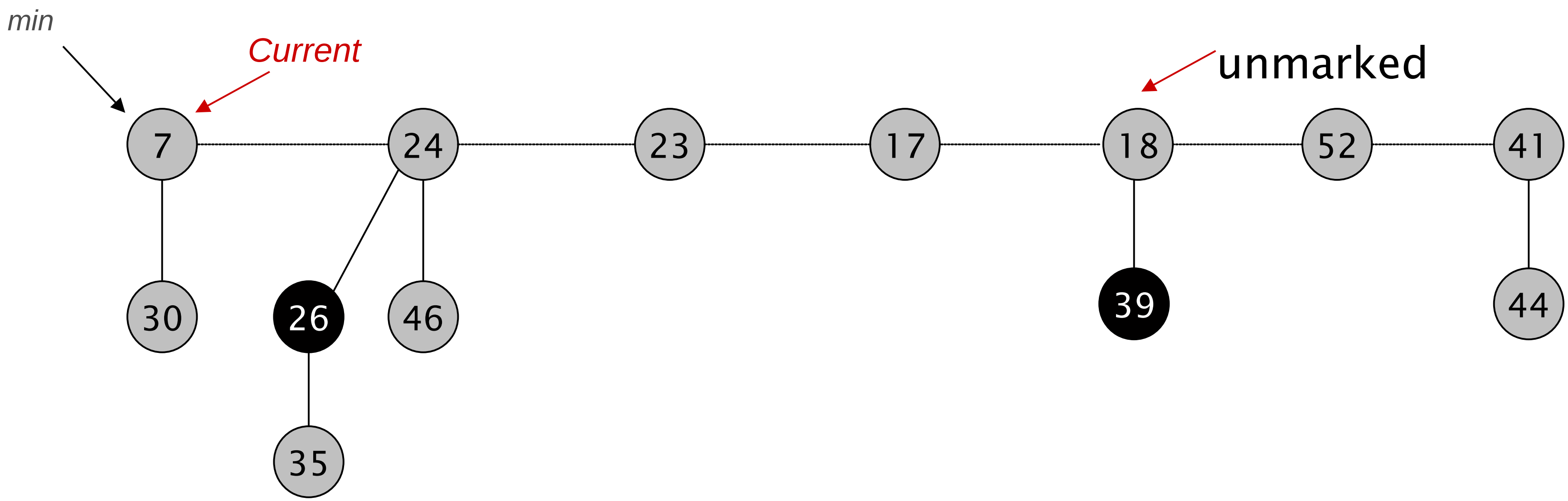
Delete min.
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

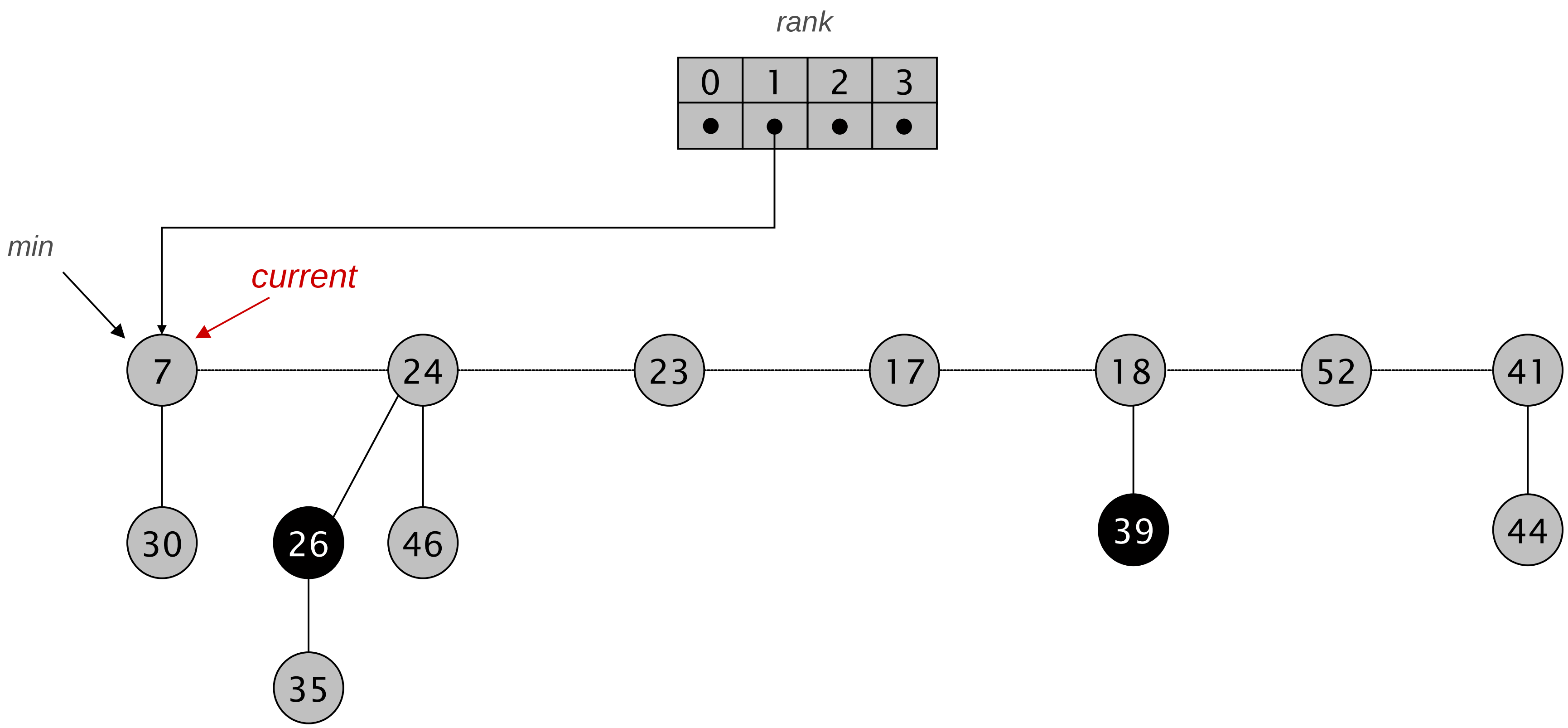
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

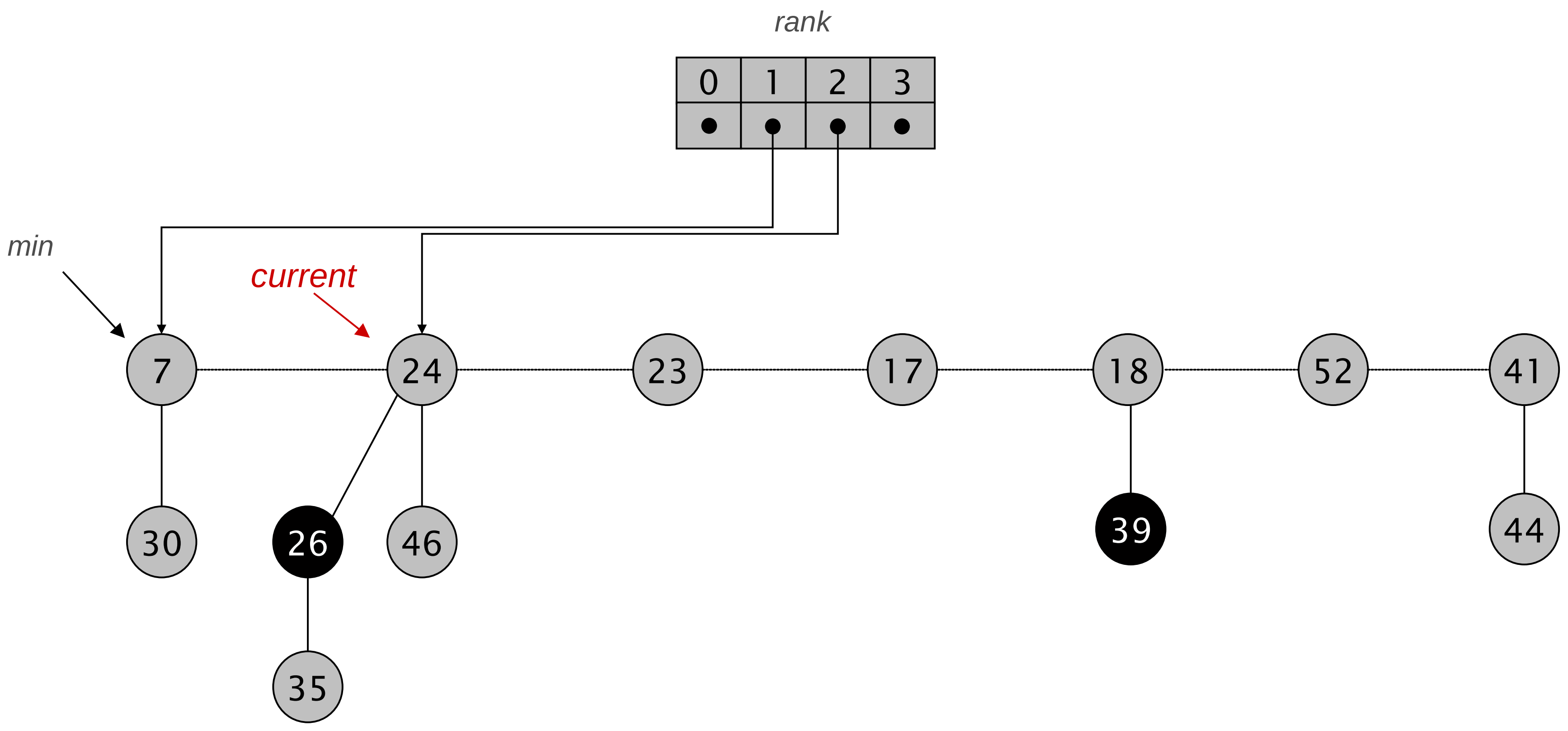
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

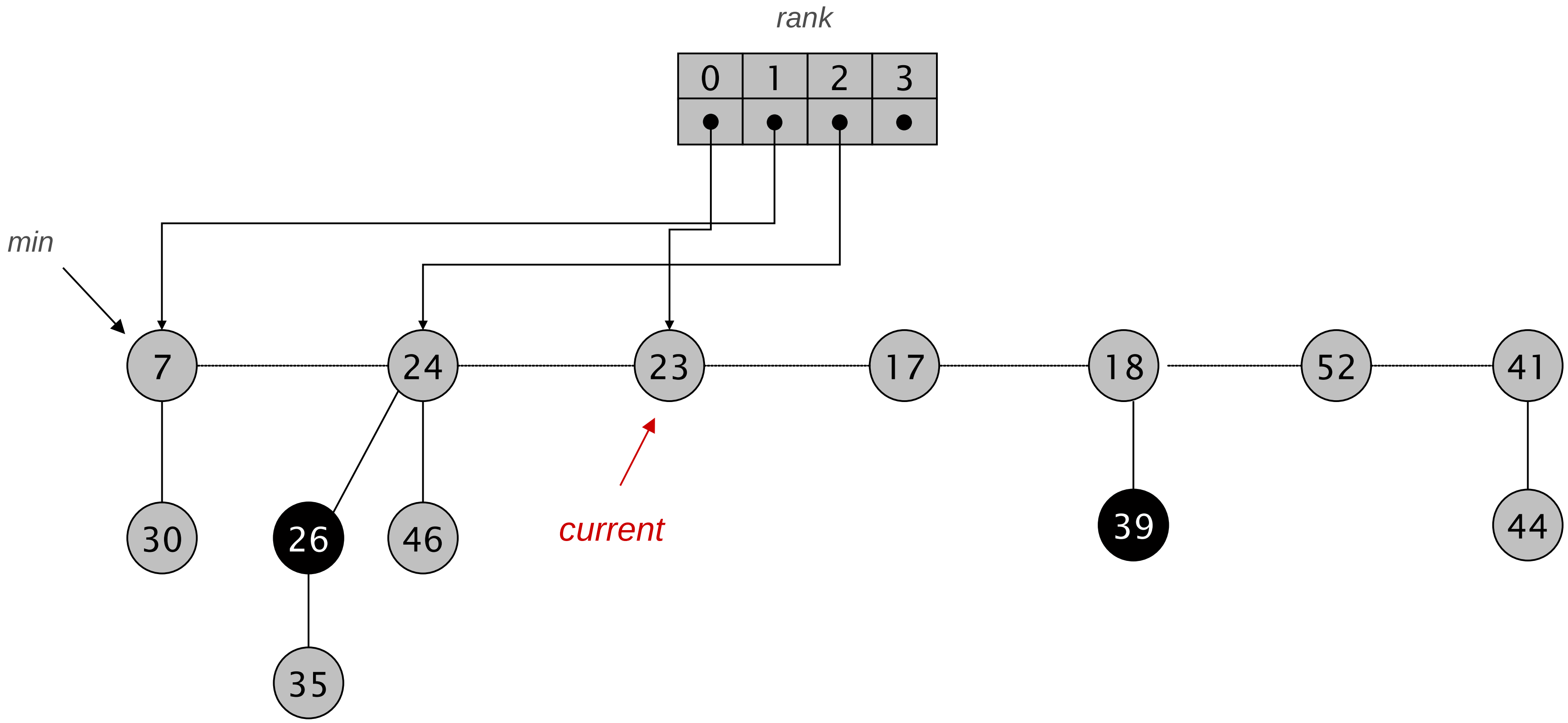
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

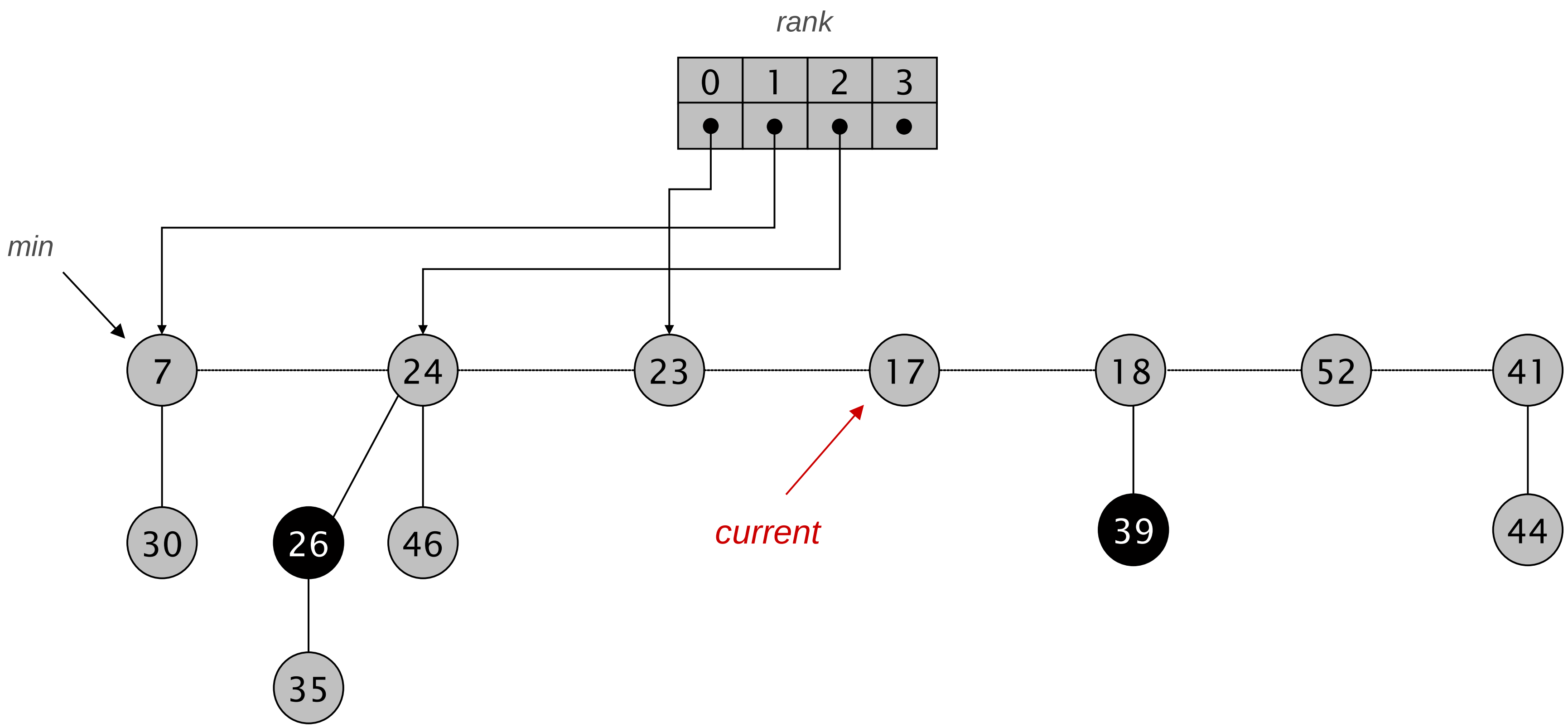
Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Delete min.

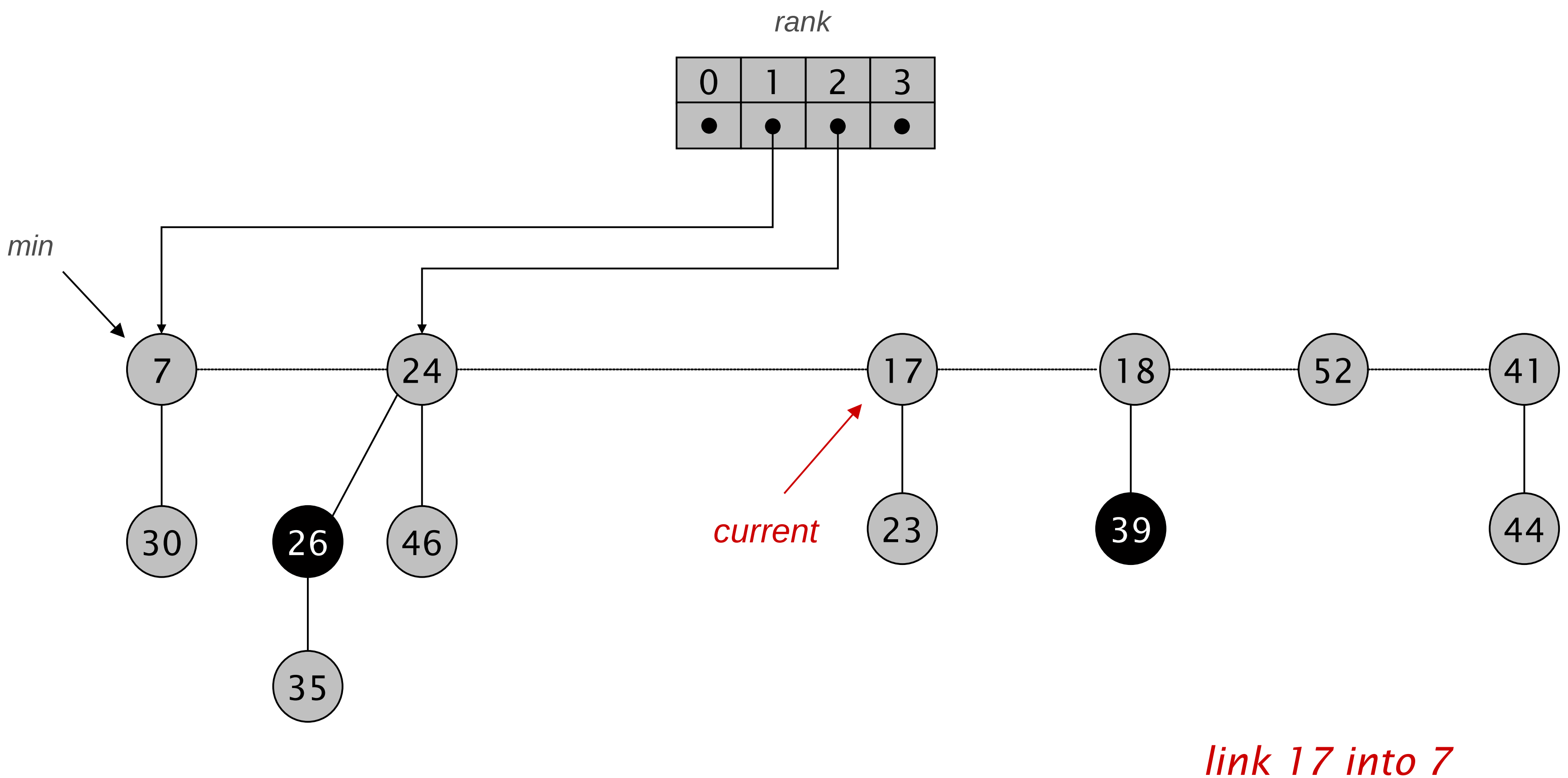
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



link 23 into 17

Fibonacci Heaps: Delete Min

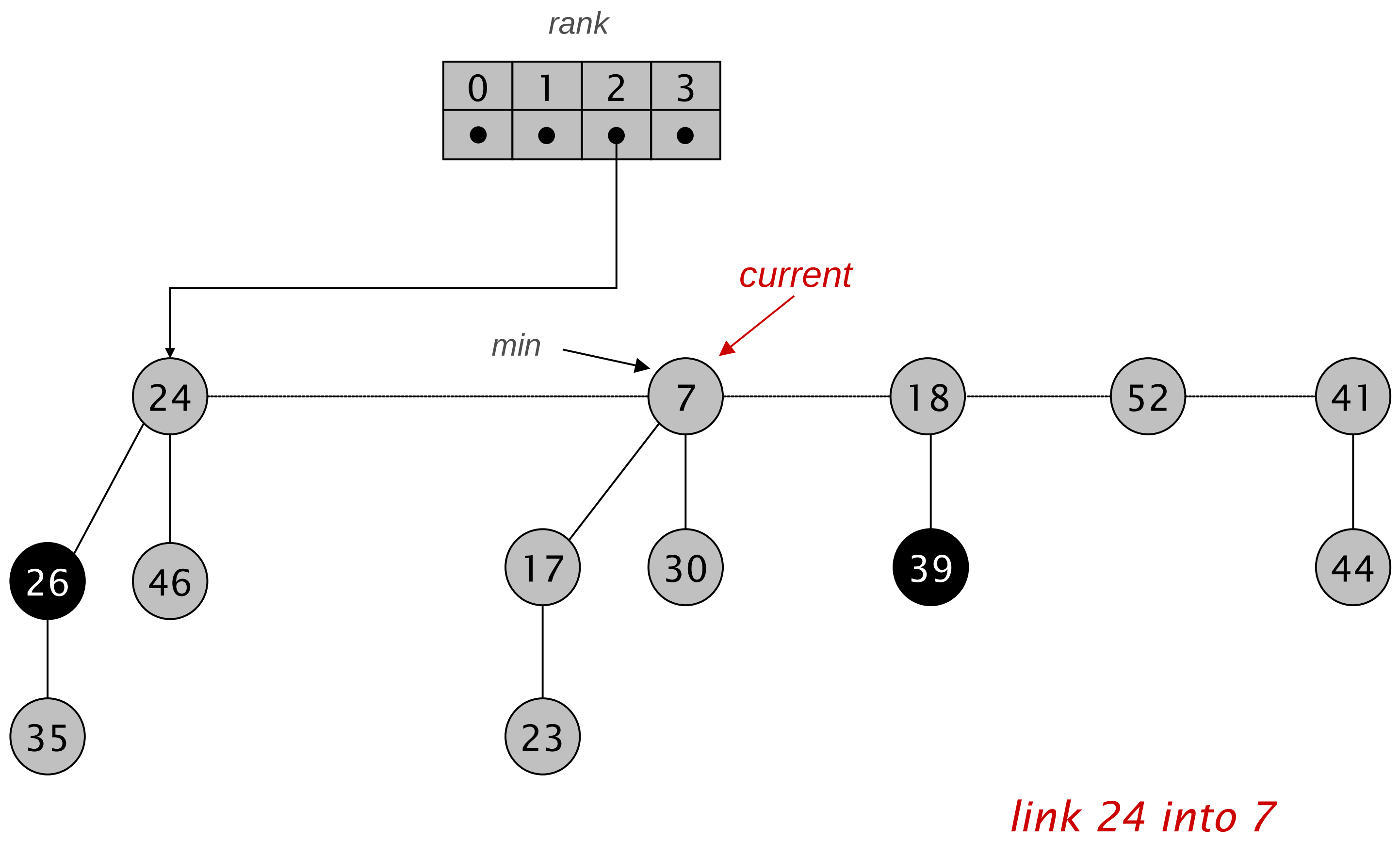
Delete min.
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

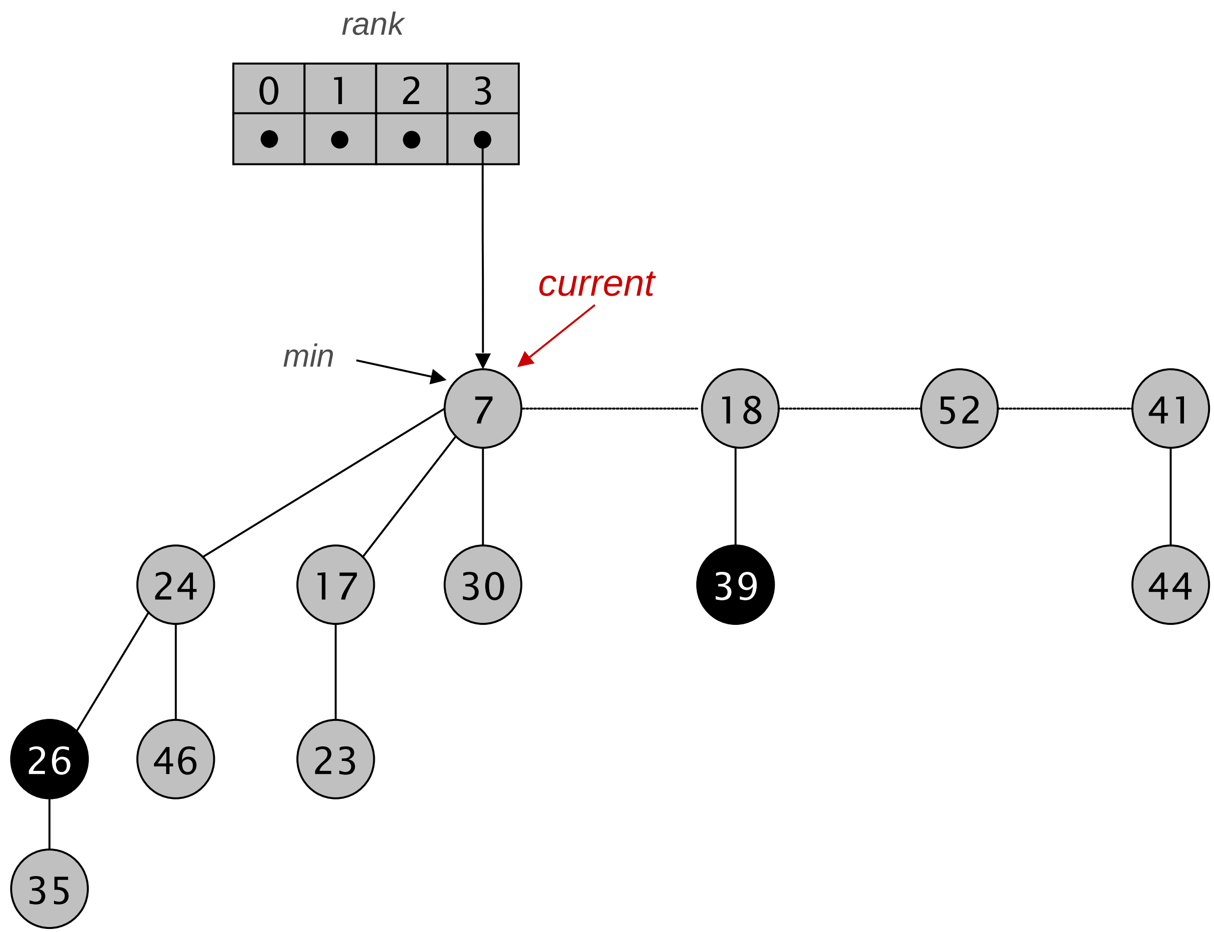
Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

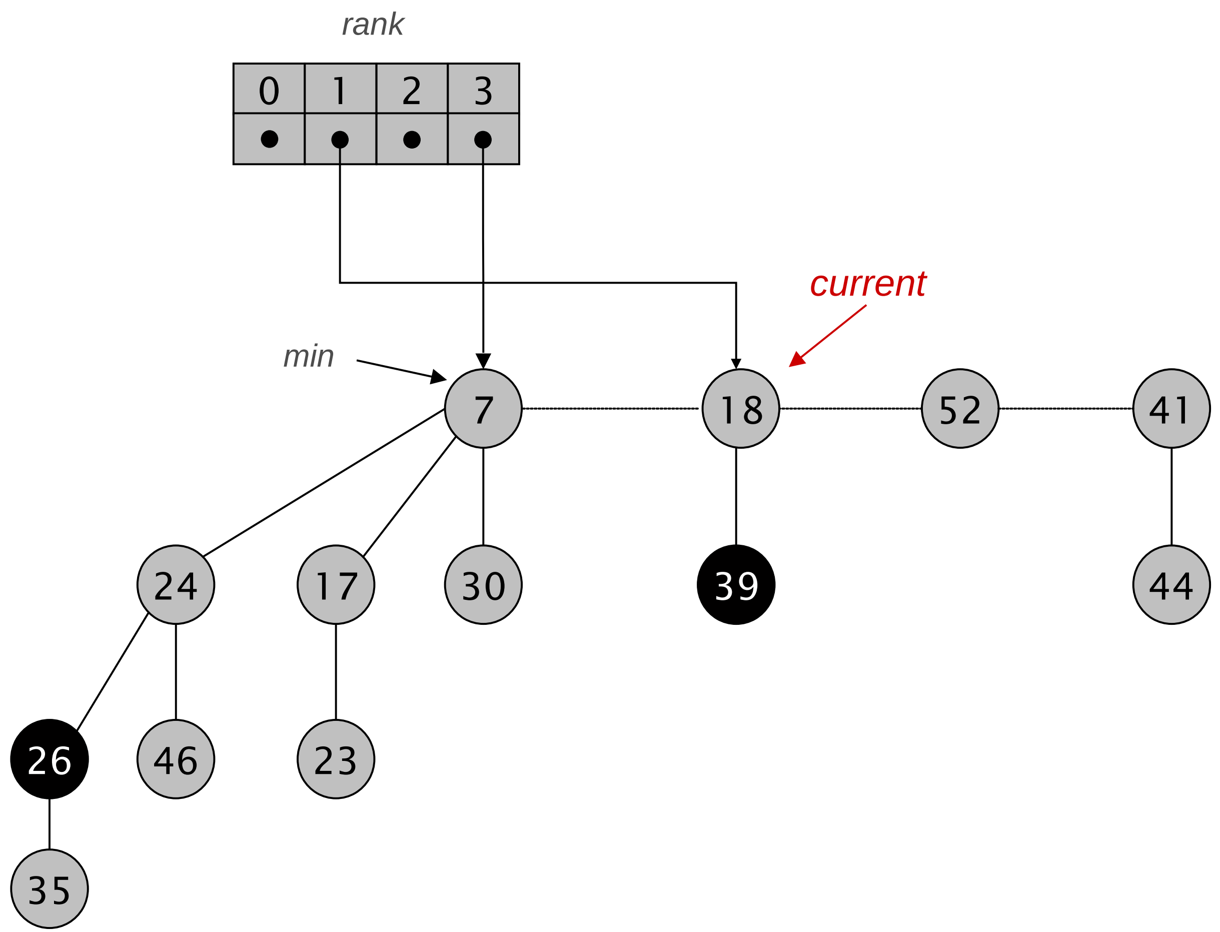


Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

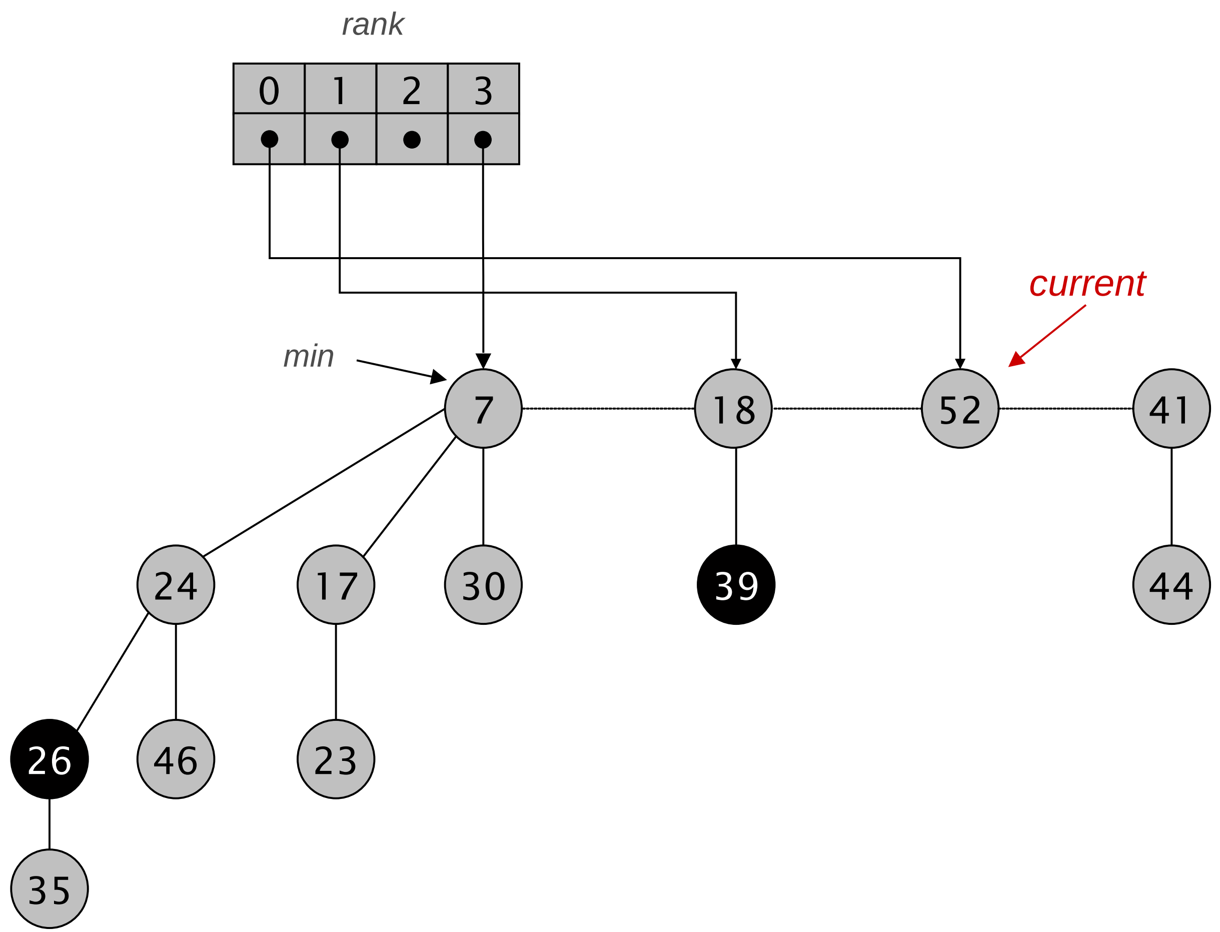


Delete min.
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



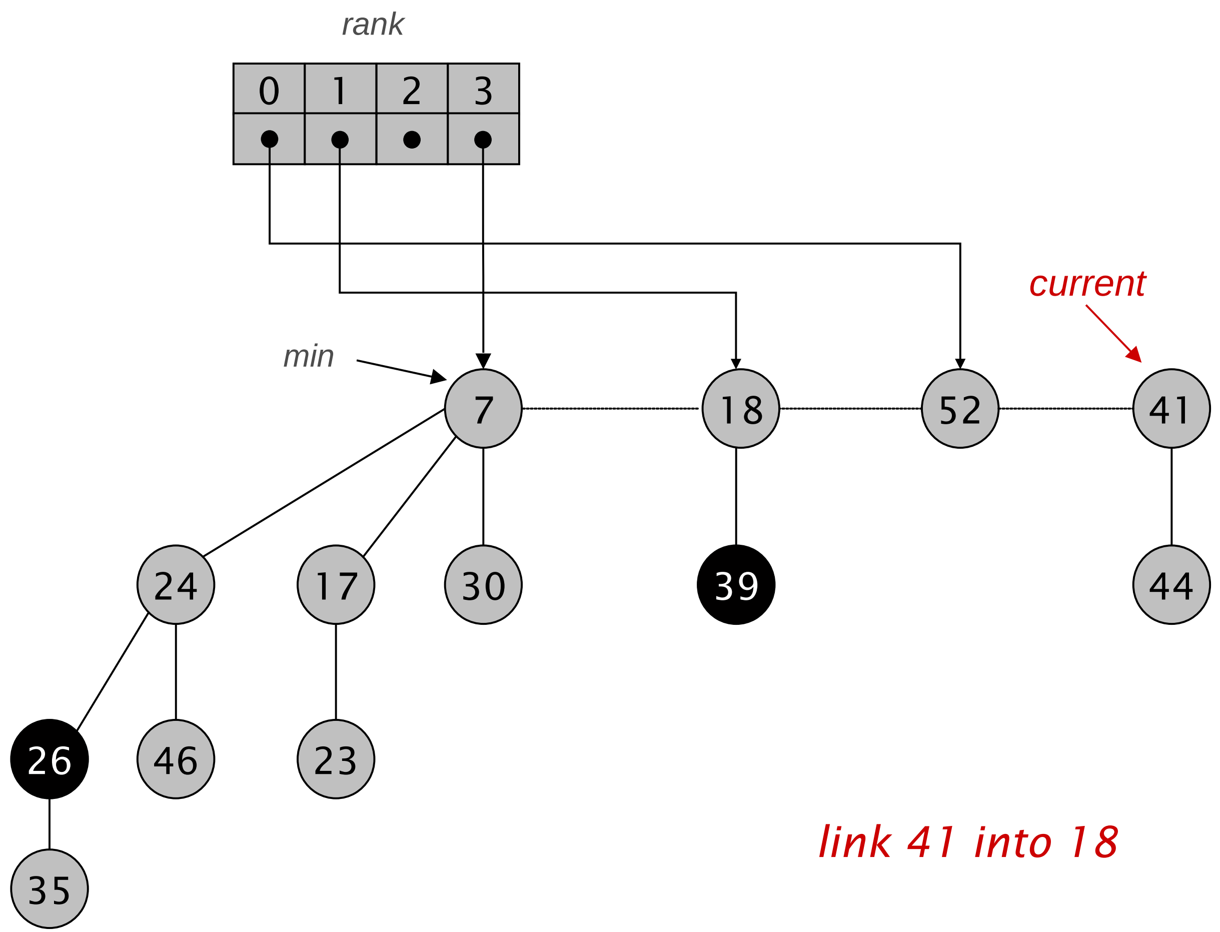
Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Delete min.

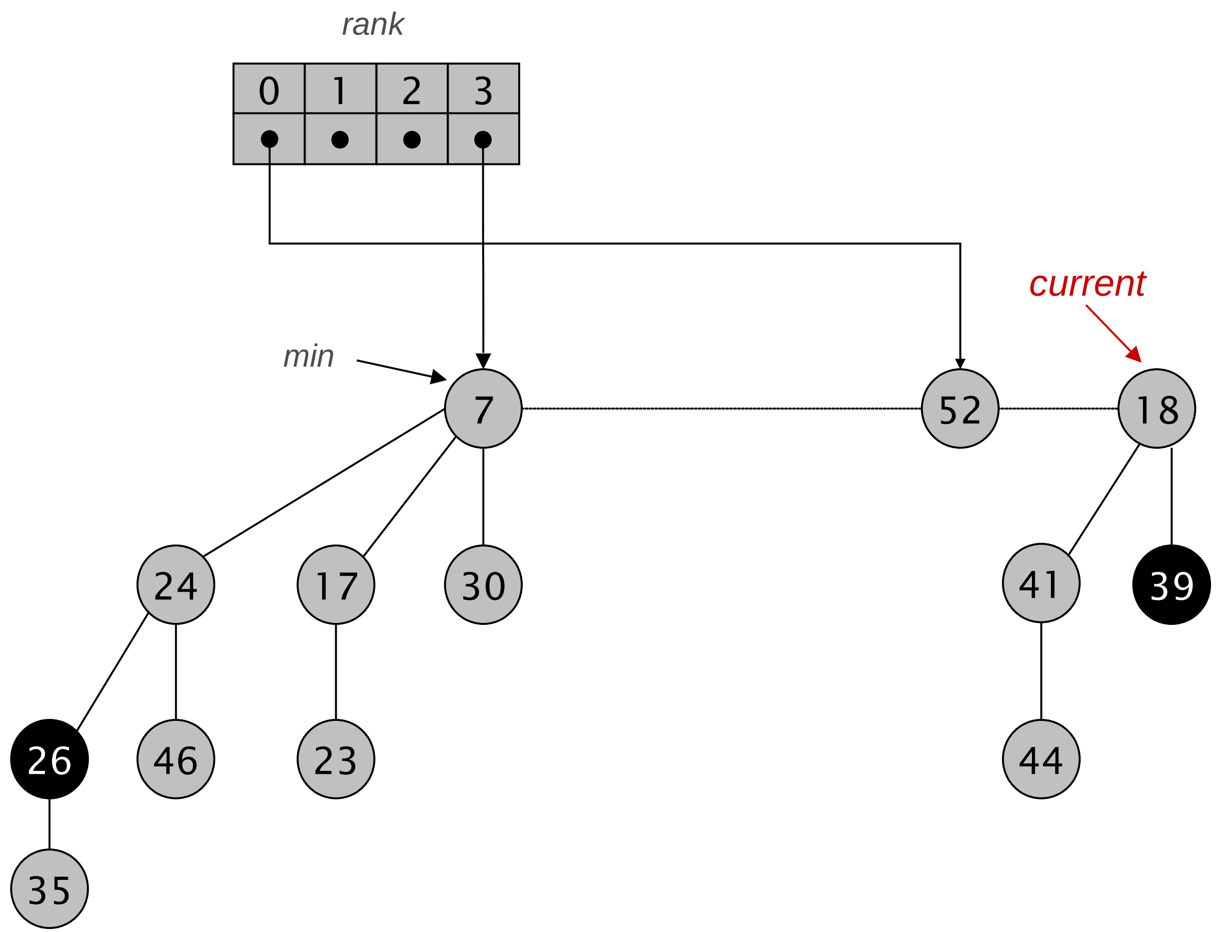
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

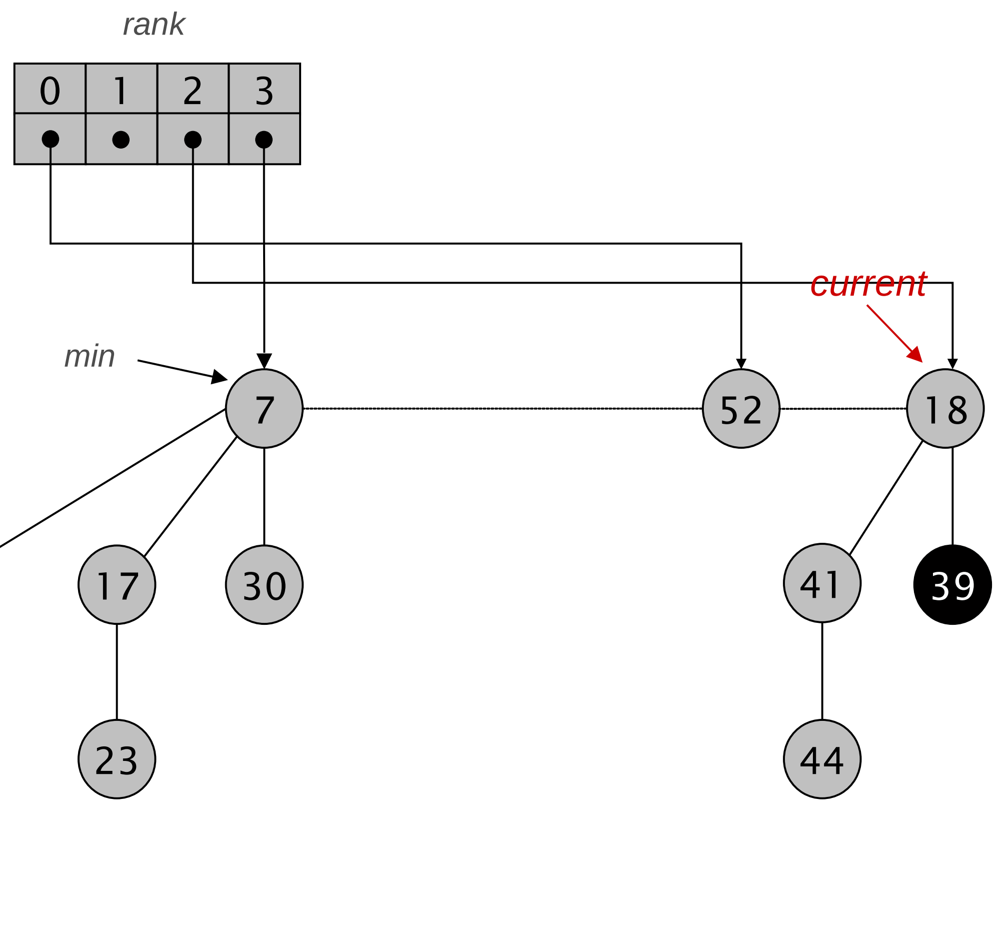
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

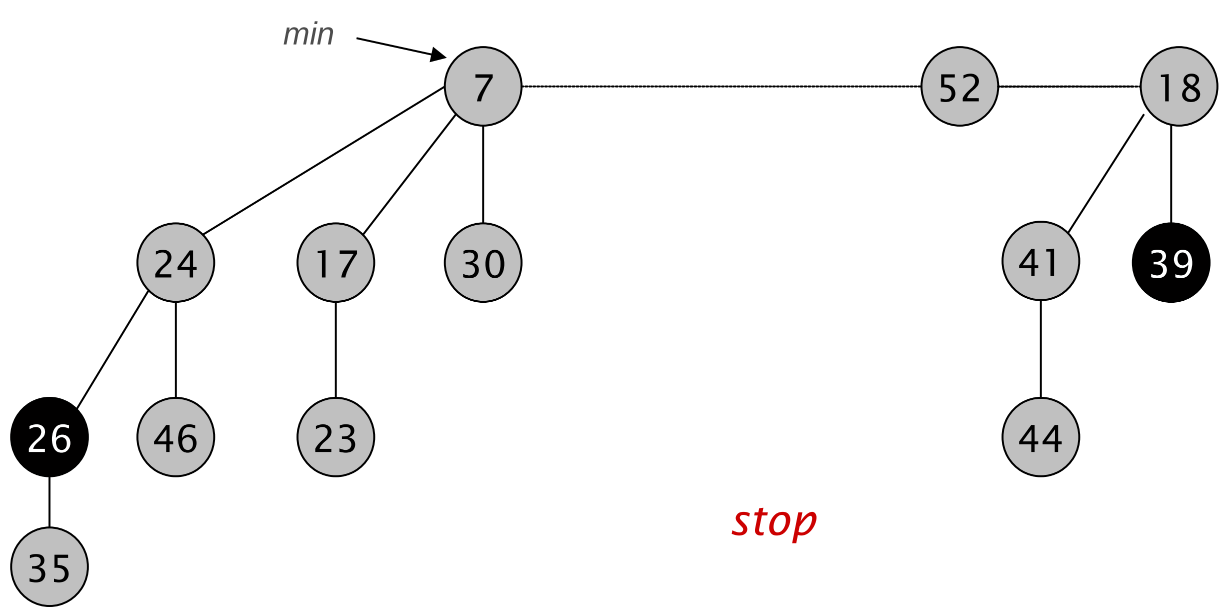
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

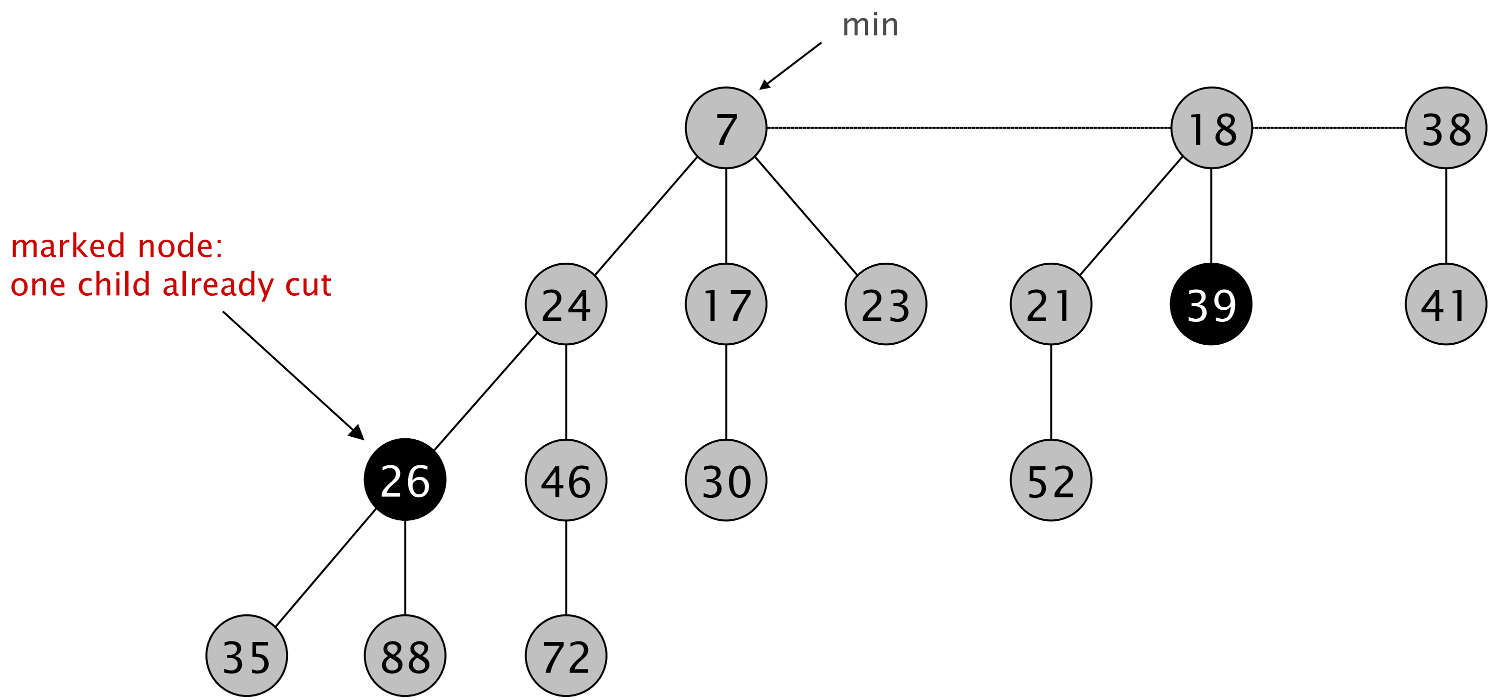


Decrease Key

Fibonacci Heaps: Decrease Key

Intuition for decreasing the key of node *x*.

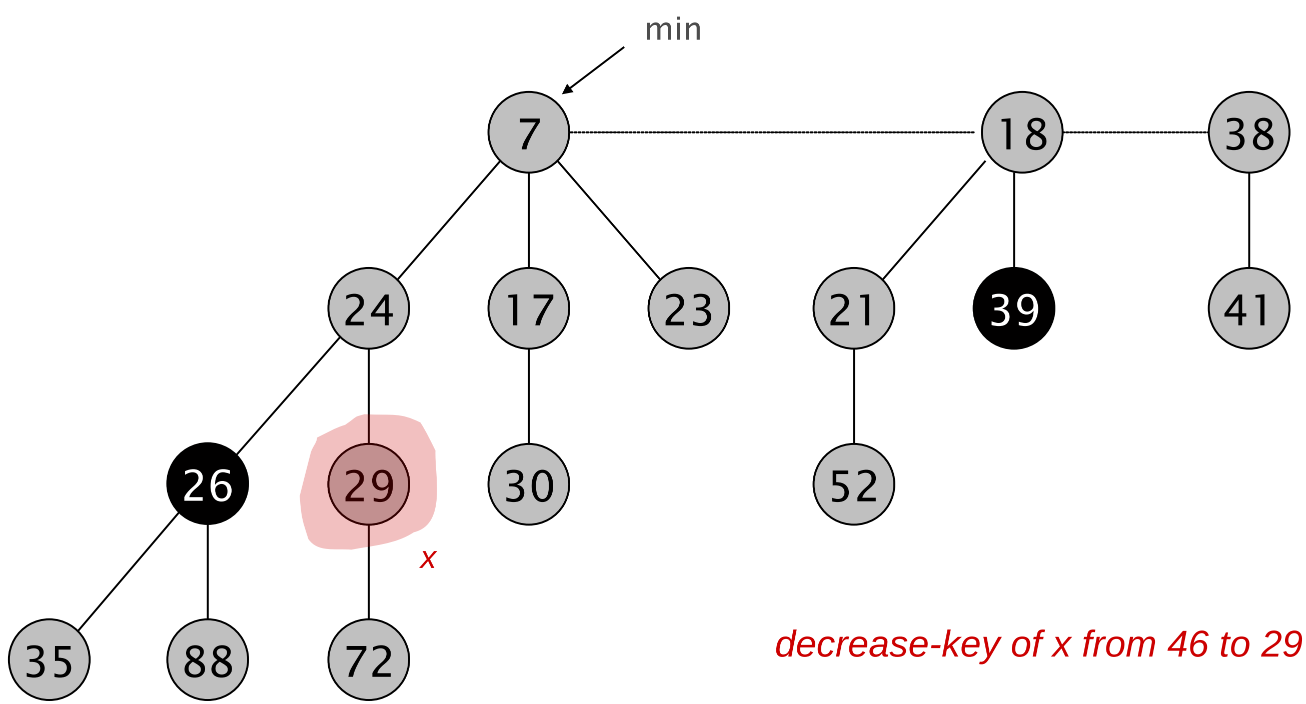
- If heap-order is not violated, just decrease the key of *x*.
- Otherwise, cut tree rooted at *x* and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

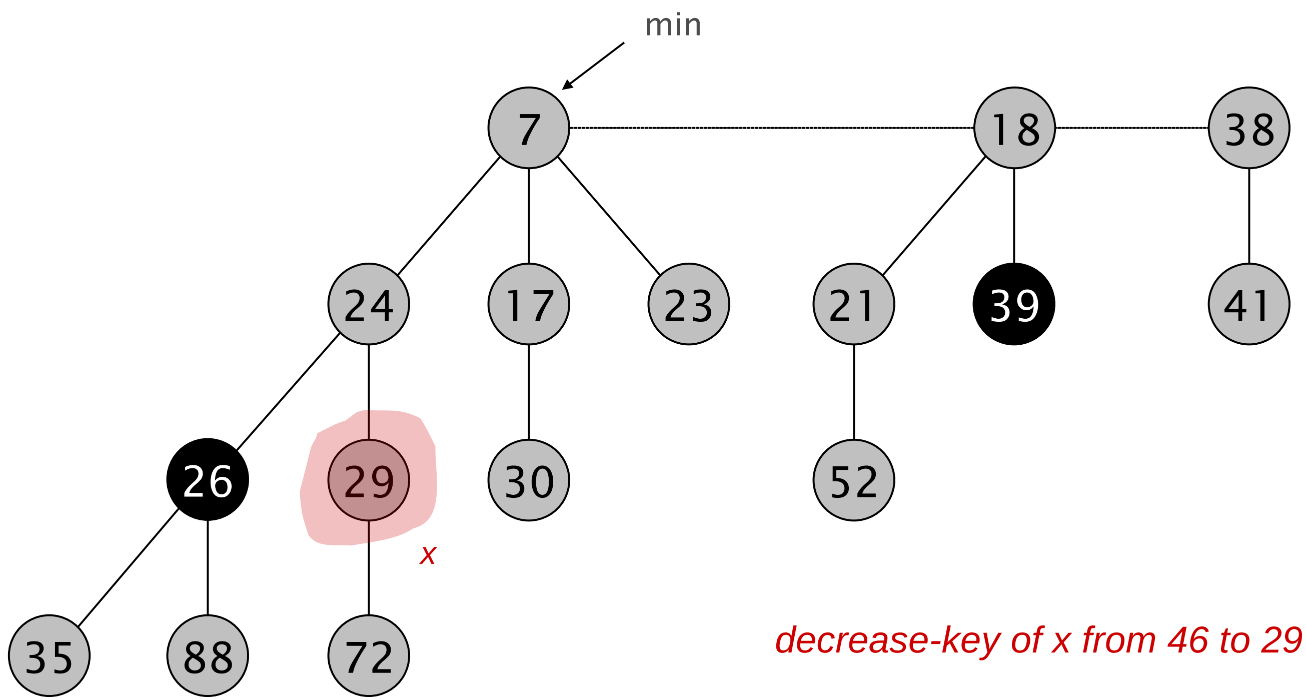
- Decrease key of *x*.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

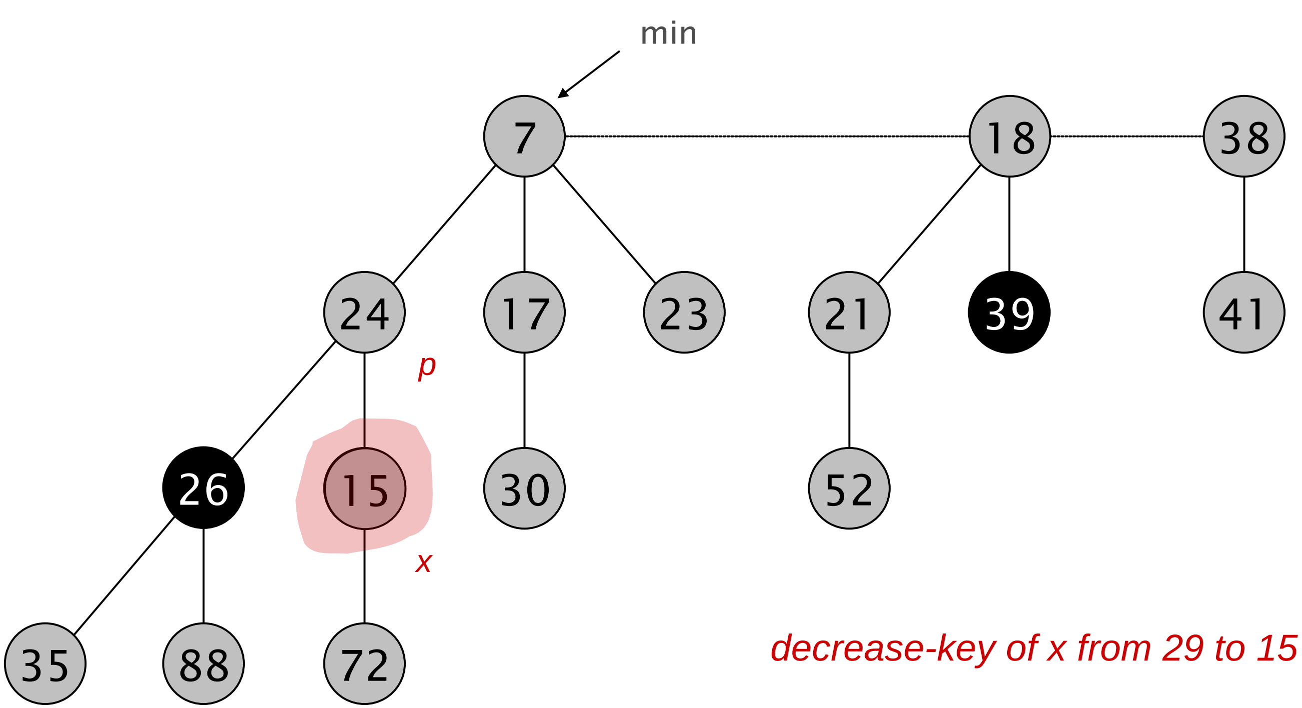
- Decrease key of *x*.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

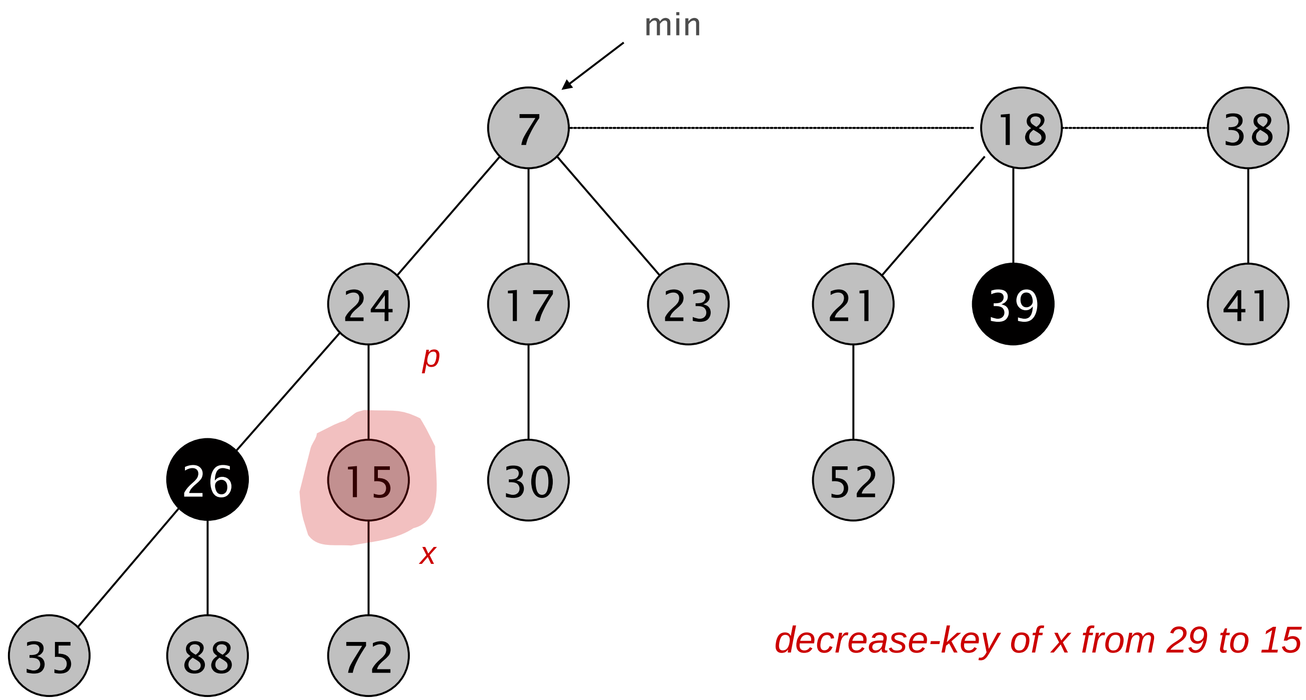


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

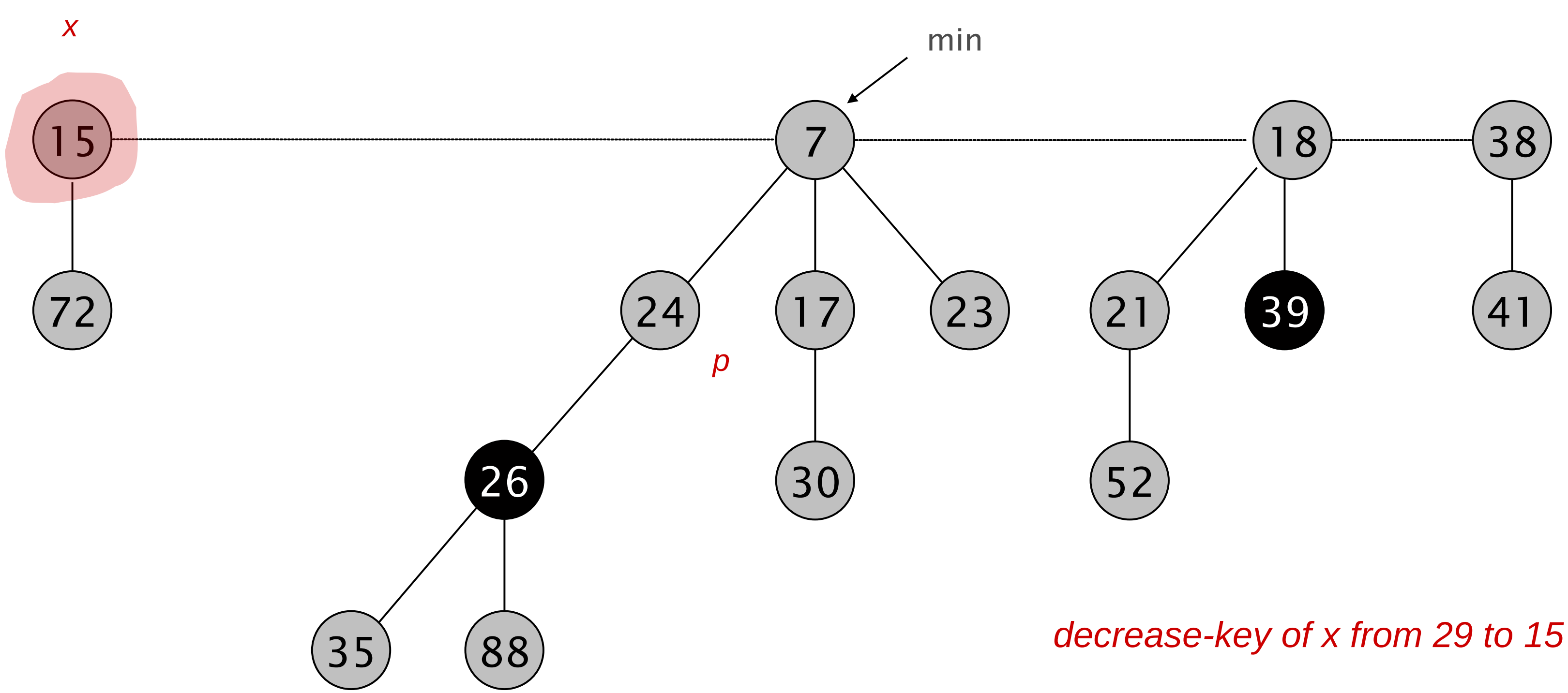
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



decrease-key of x from 29 to 15

Case 2a. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

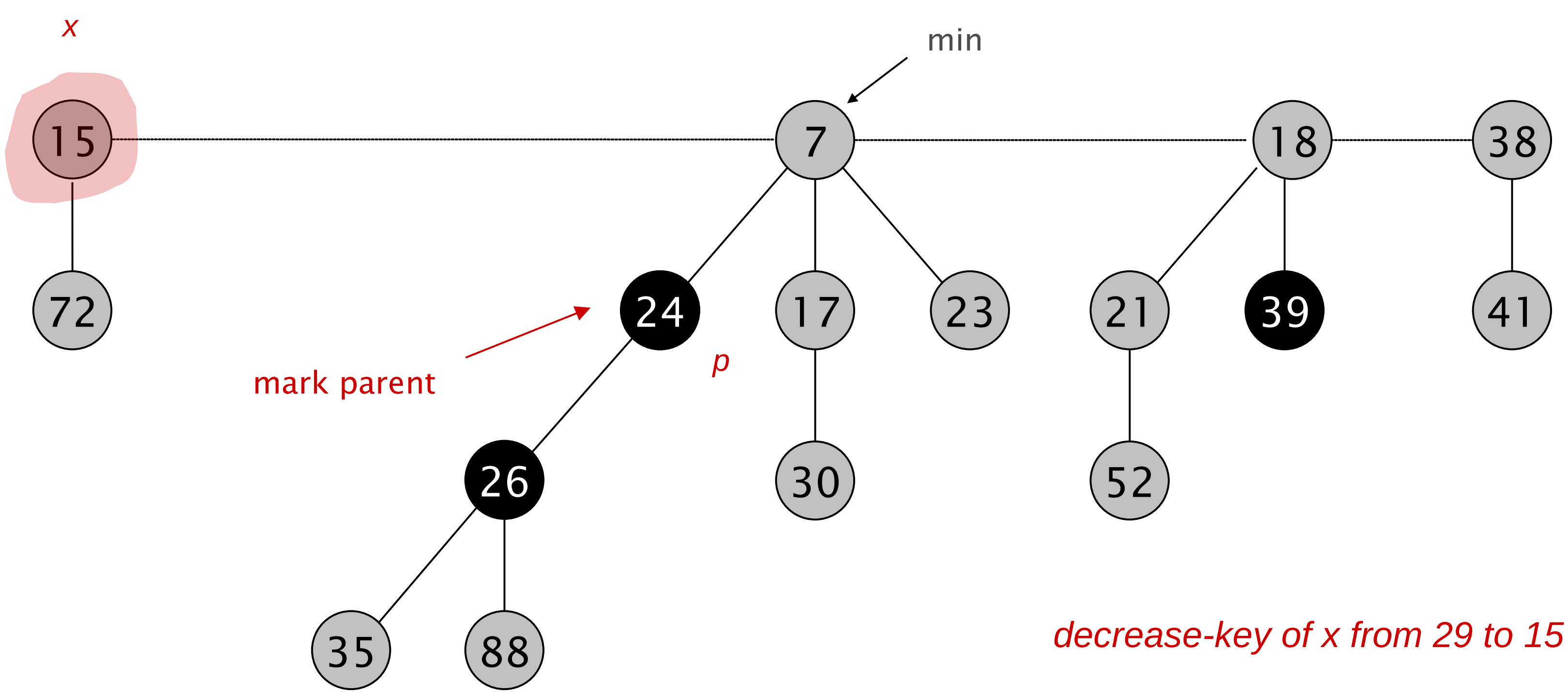


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

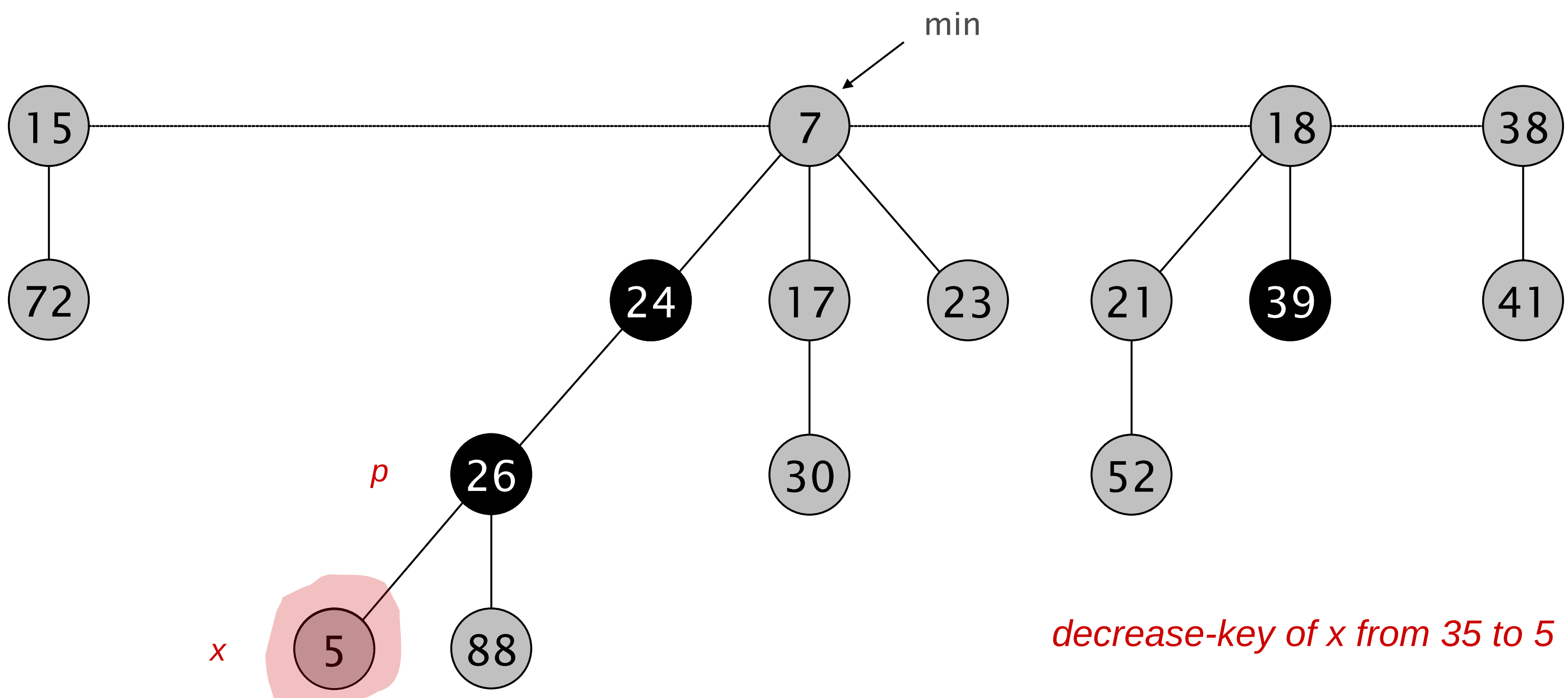


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

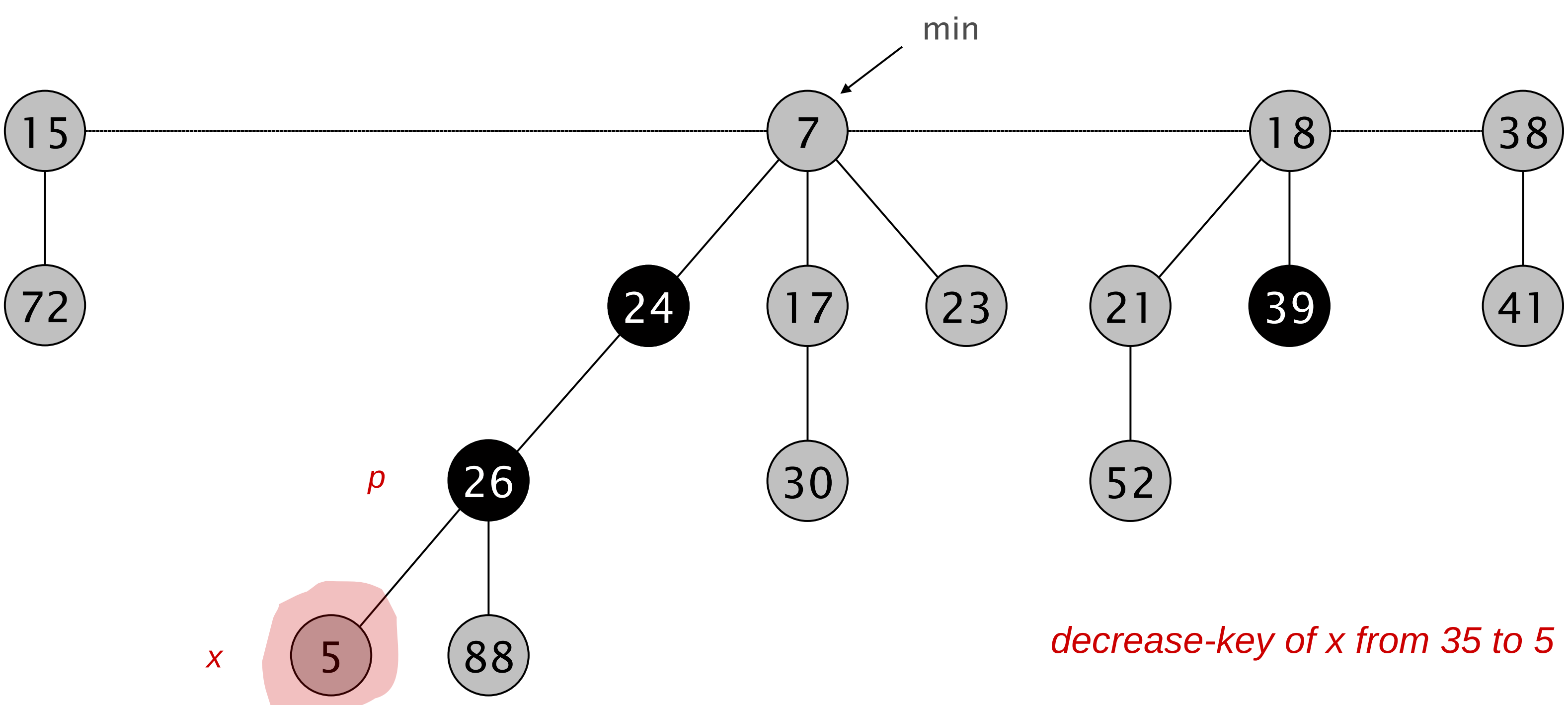
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

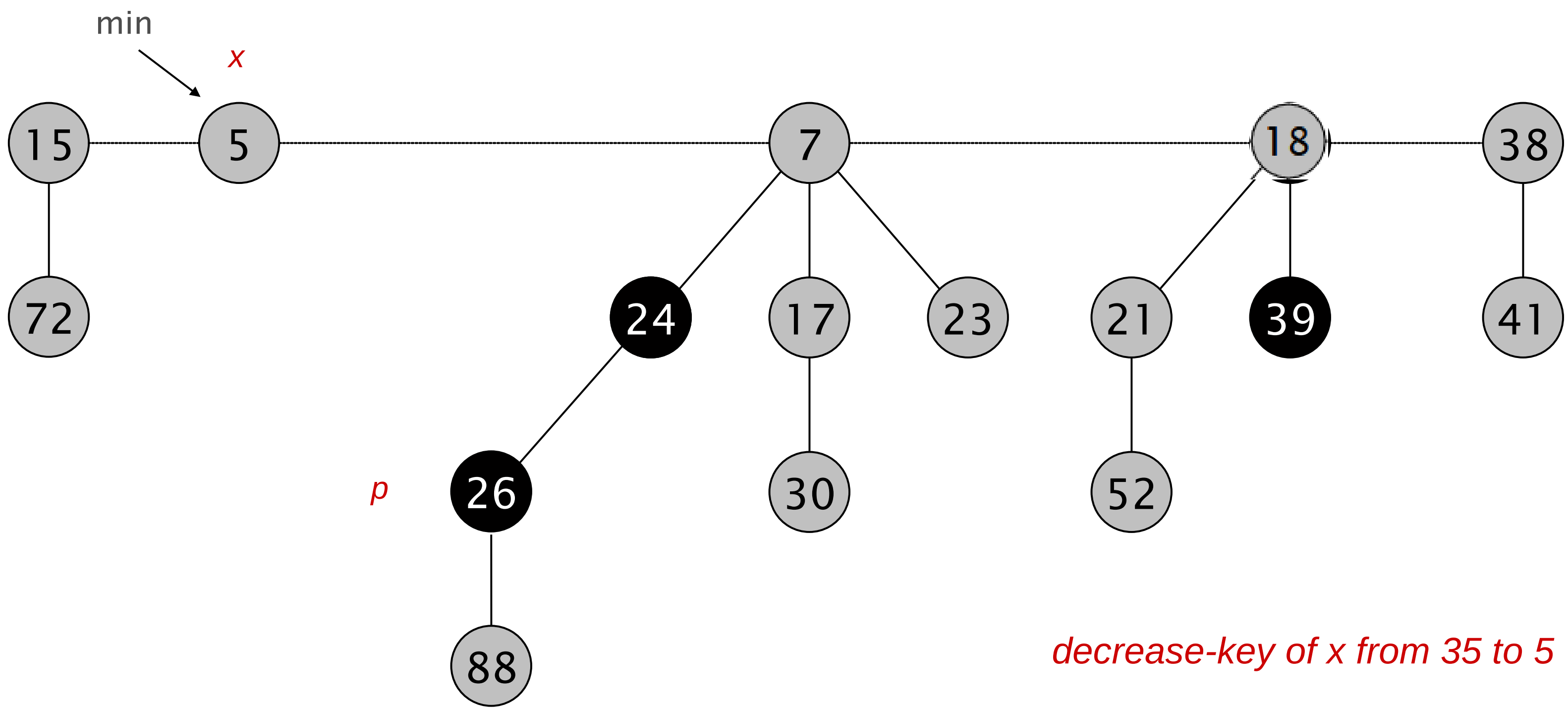
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

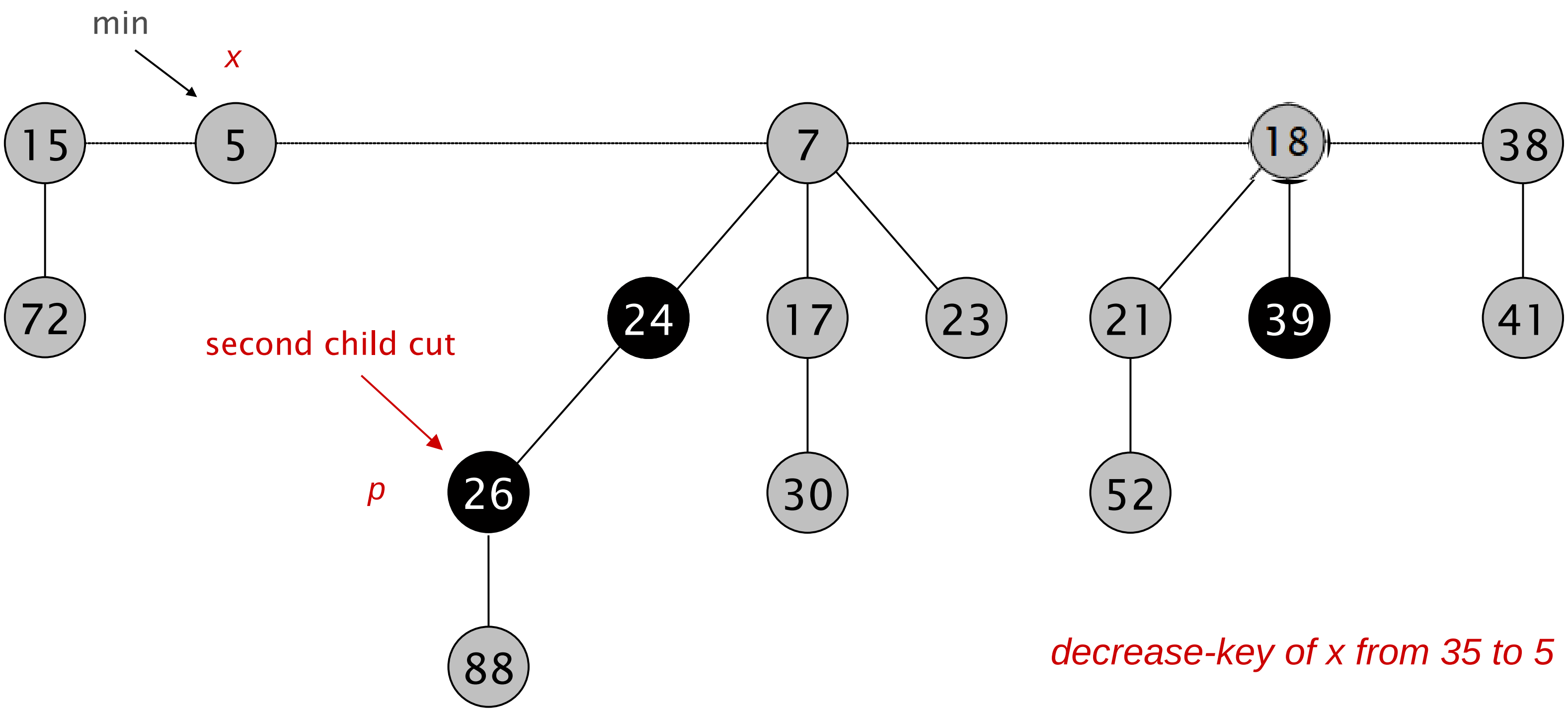
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

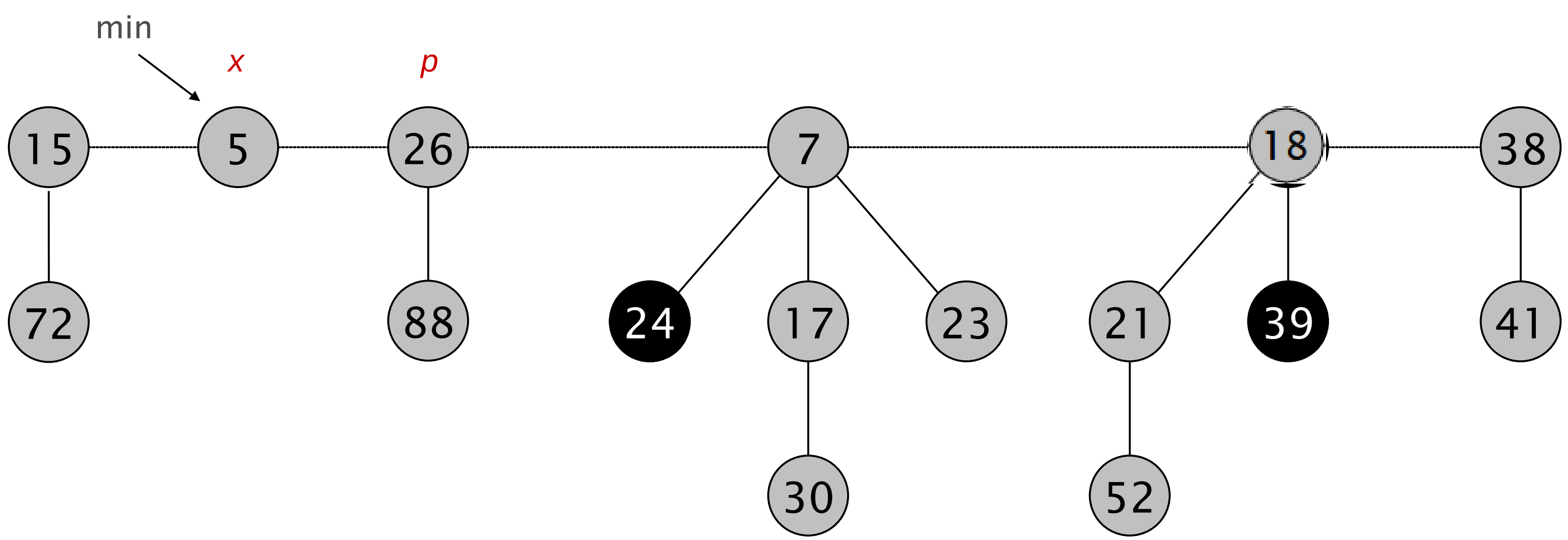
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

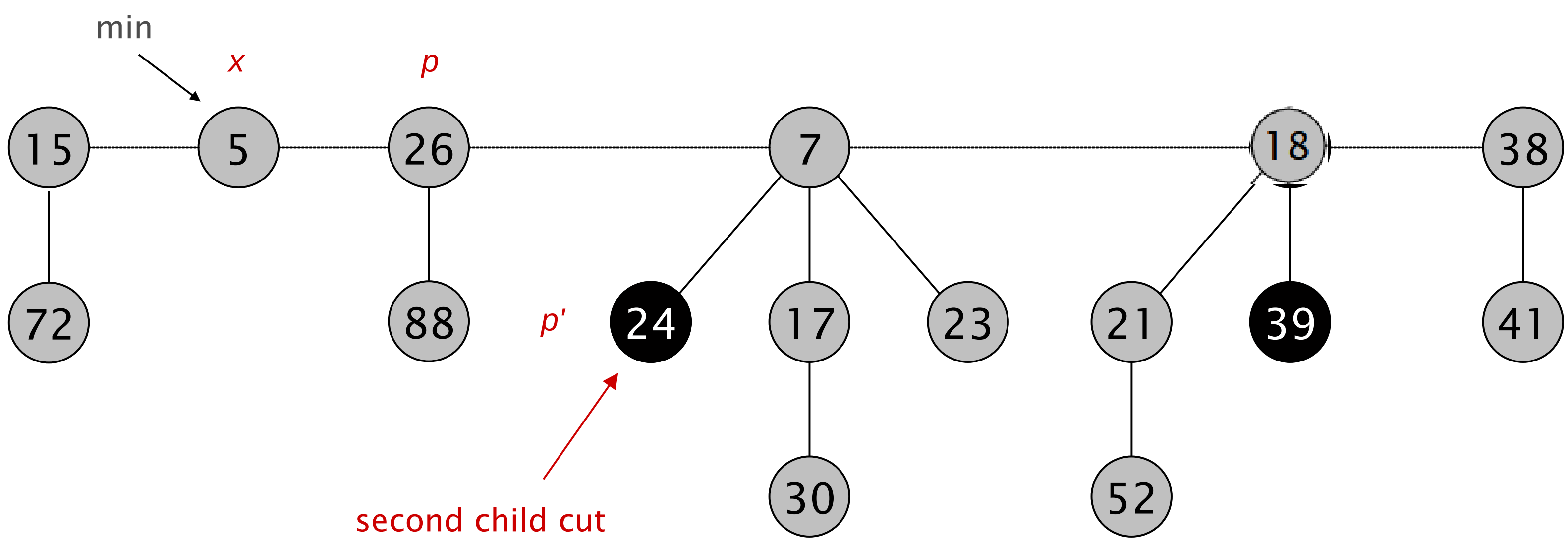


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it; Otherwise, cut *p*, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).

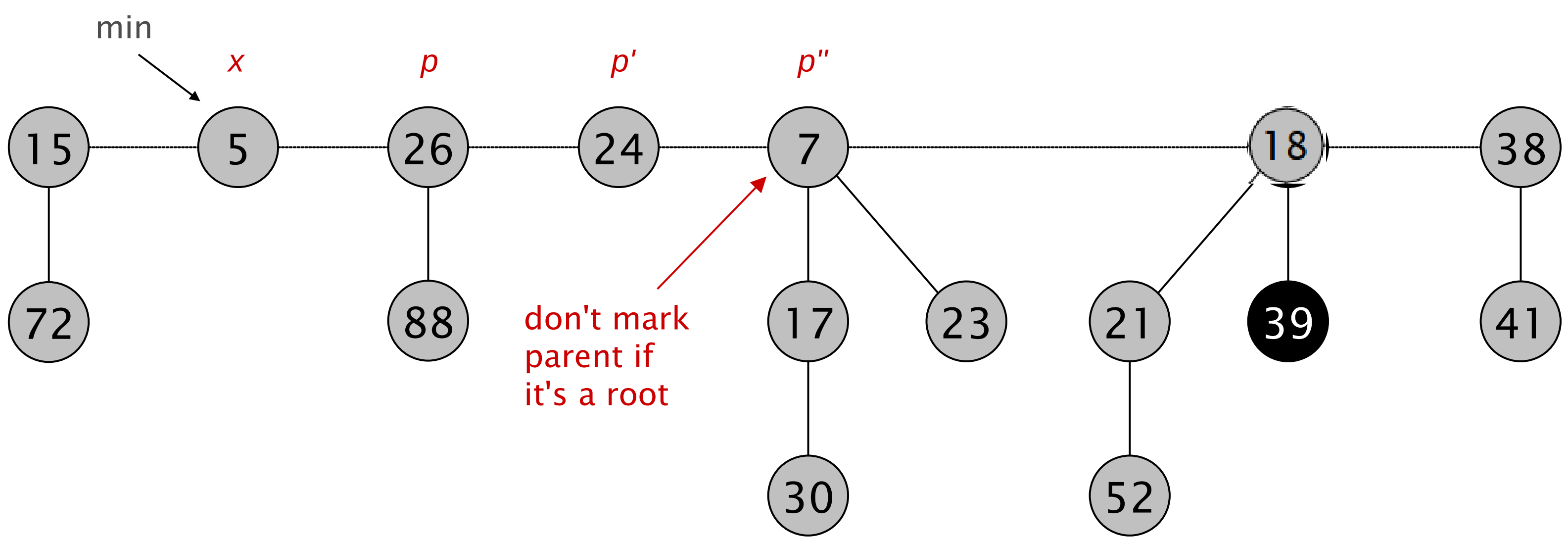


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



decrease-key of x from 35 to 5

Analysis

Analysis Summary

Insert.	O(1)
Delete-min.	O(*rank*(*H*)) †
Decrease-key.	O(1) †

† amortized

Key lemma. $rank(H) = O(\log n)$.

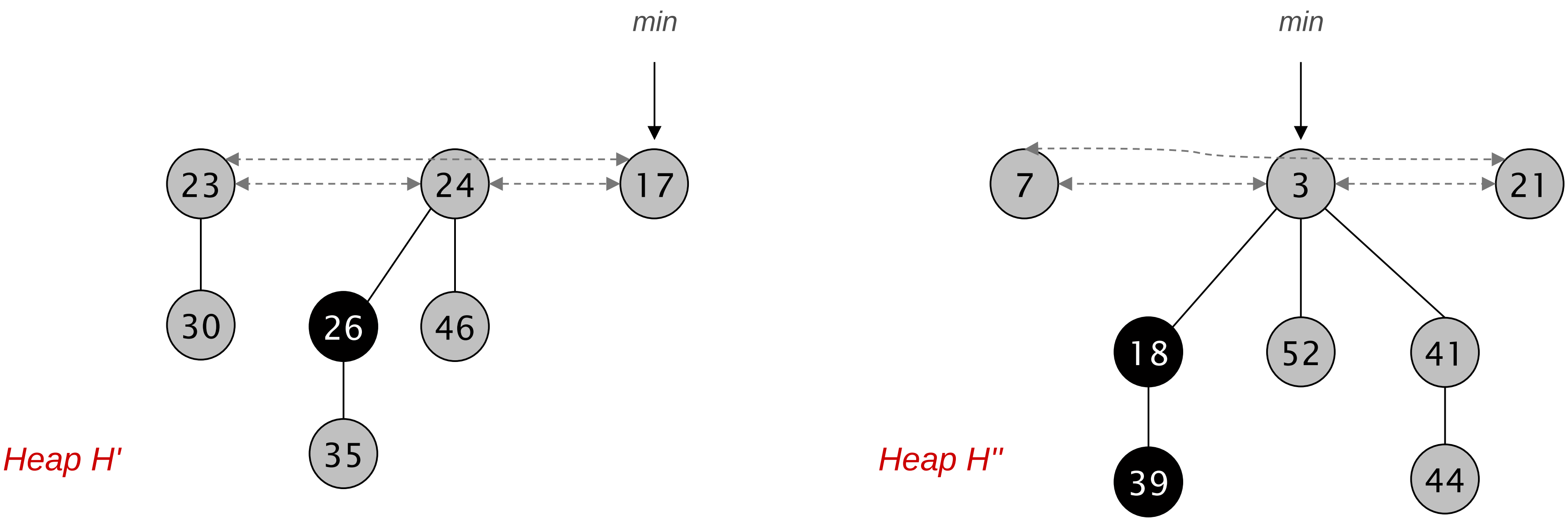
number of nodes is exponential in rank

Union

Fibonacci Heaps: Union

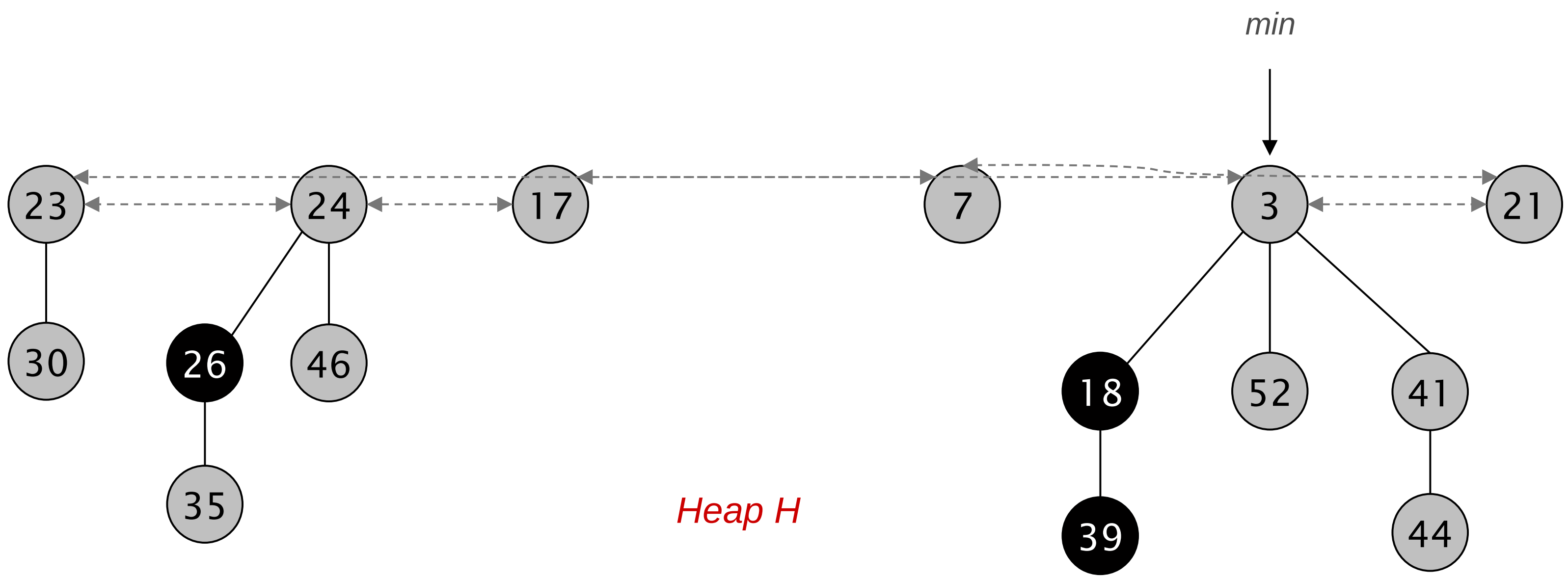
Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

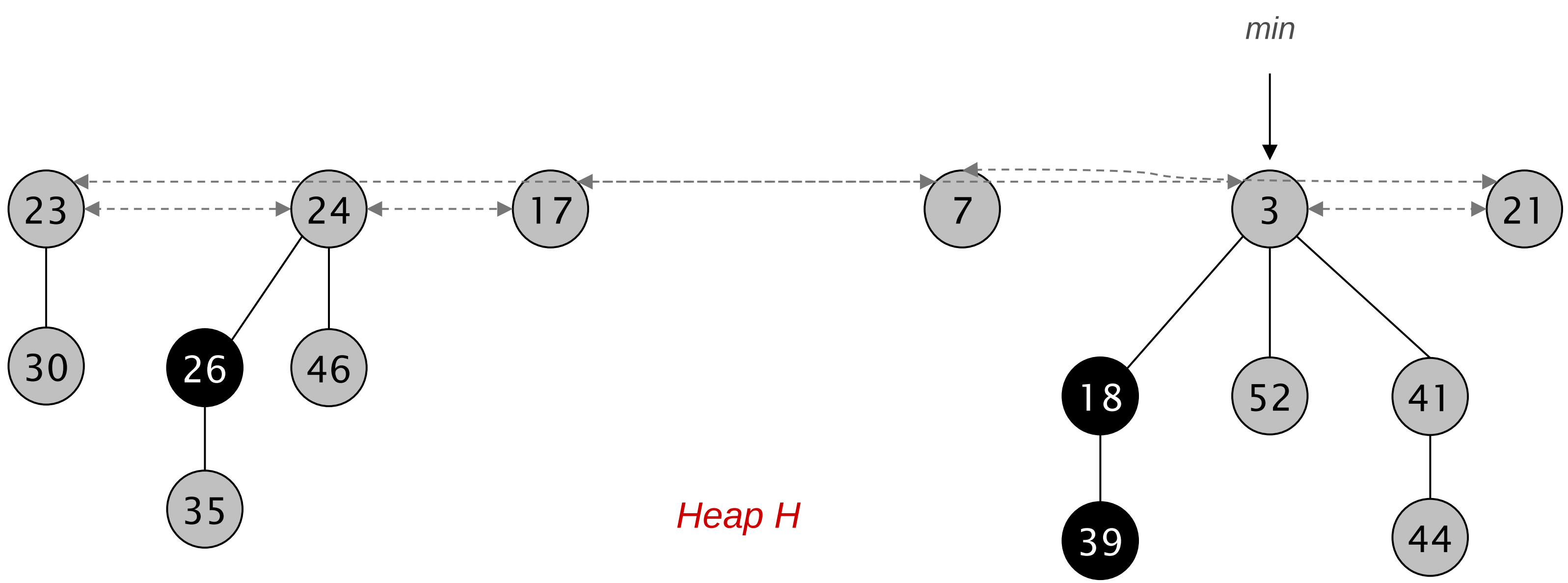
Actual cost. O(1)

Change in potential. 0

Amortized cost. O(1)

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

potential function



Delete

Fibonacci Heaps: Delete

Delete node x.

- *decrease-key* of x to $-\infty$.
- *delete-min* element in heap.

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

potential function

Amortized cost. $O(rank(H))$

- $O(1)$ amortized for *decrease-key*.
- $O(rank(H))$ amortized for *delete-min*.

Priority Queues Performance Cost Summary

Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap †	Relaxed Heap
make-heap	1	1	1	1	1
is-empty	1	1	1	1	1
insert	1	log *n*	log *n*	1	1
delete-min	*n*	log *n*	log *n*	log *n*	log *n*
decrease-key	*n*	log *n*	log *n*	1	1
delete	*n*	log *n*	log *n*	log *n*	log *n*
union	1	*n*	log *n*	1	1
find-min	*n*	1	log *n*	1	1

n = number of elements in priority queue

† amortized