Fibonacci Heaps

Chapter 19

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

Theorem. Starting from empty Fibonacci heap, any sequence of a_1 insert, a_2 delete-min, and a_3 decrease-key operations takes $O(a_1 + a_2 \log n + a_3)$ time.

Fibonacci Heaps

History. [Fredman and Tarjan, 1986]

- Ingenious data structure and analysis.
- Original motivation: improve Dijkstra's shortest path algorithm from O(E log V) to O(E + V log V).

V insert, V delete-min, E decrease-key

Basic idea.

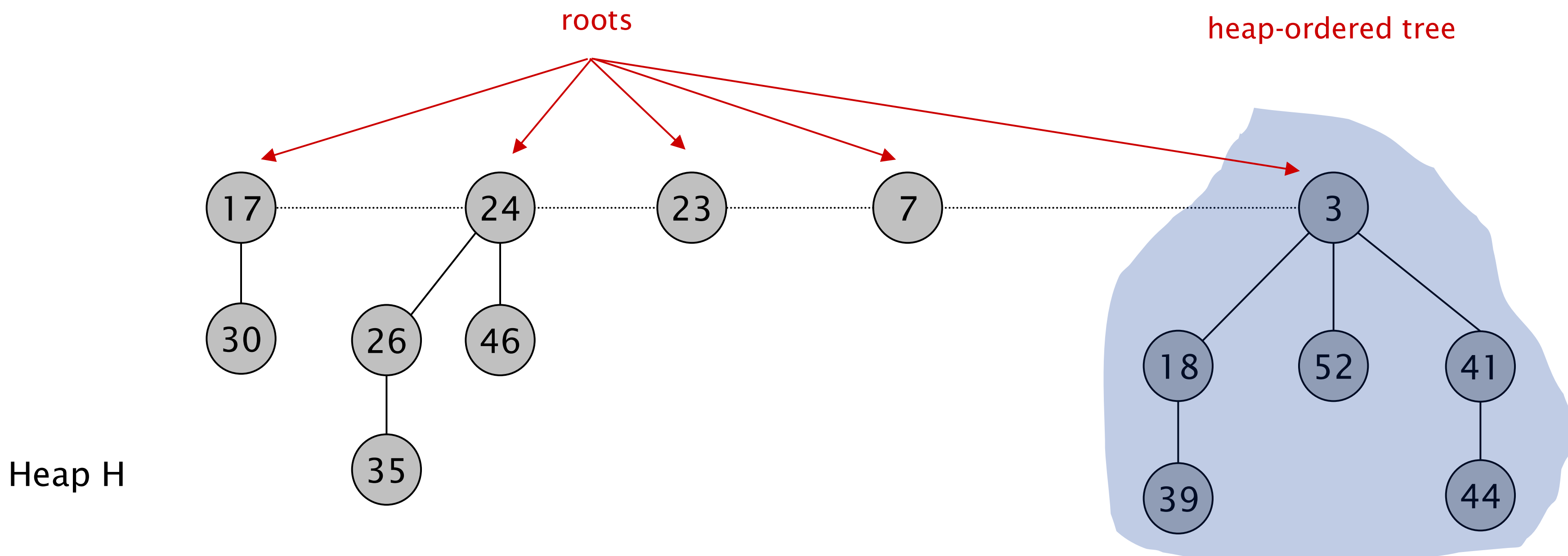
- It is a collection of rooted trees that are min-ordered
- Each node points to its parent and its children
- The children are linked together in a circular, doubled linked-list
- Fibonacci heap: lazily defer consolidation until next delete-min.

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.

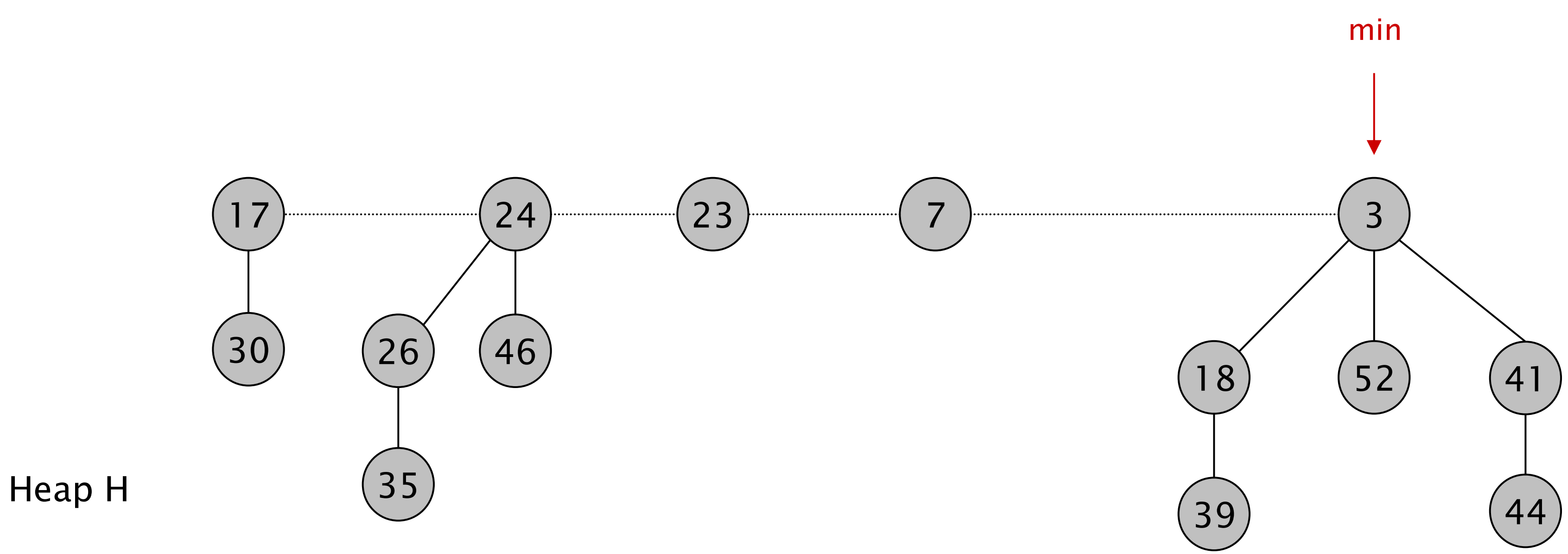


Fibonacci Heaps: Structure

Fibonacci heap.

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

find-min takes O(1) time

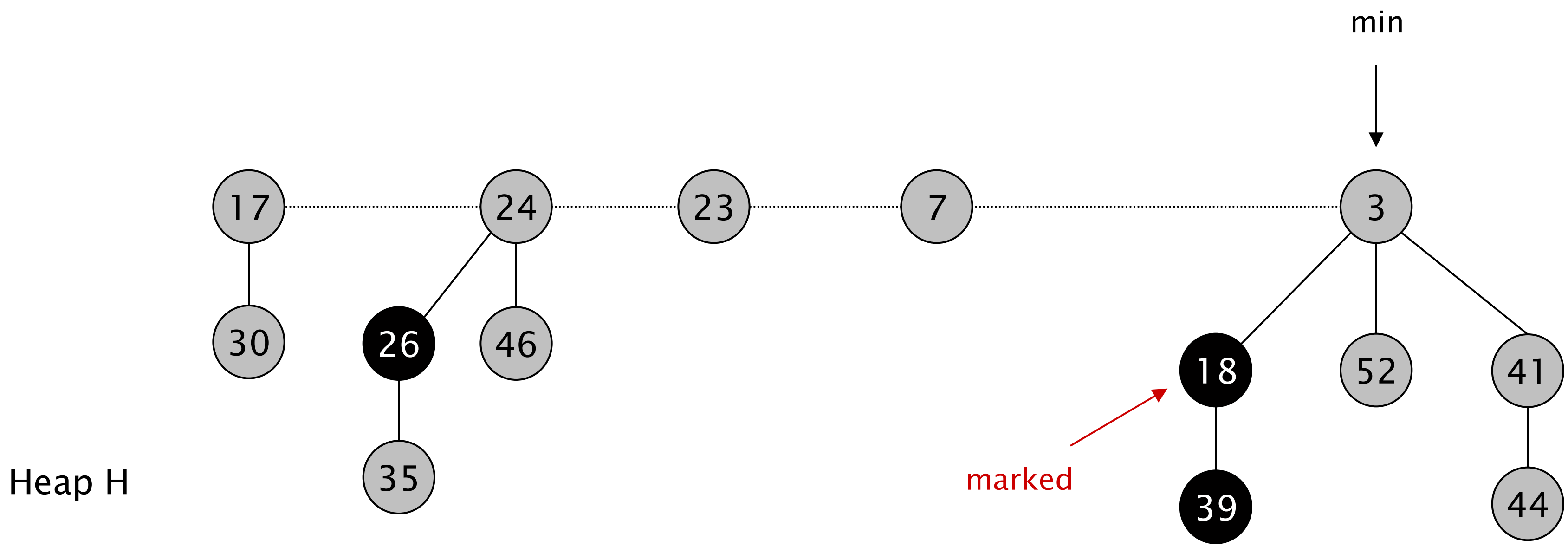


Fibonacci Heaps: Structure

Fibonacci heap.

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

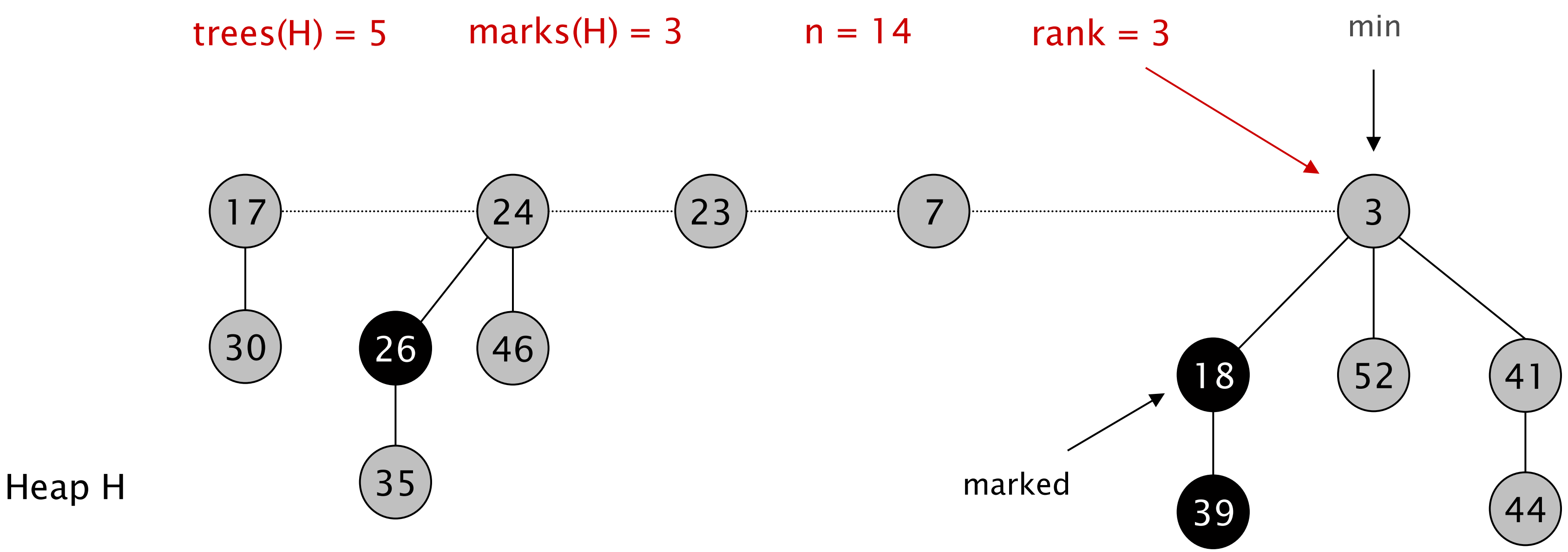
use to keep heaps flat (stay tuned)



Fibonacci Heaps: Notation

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.

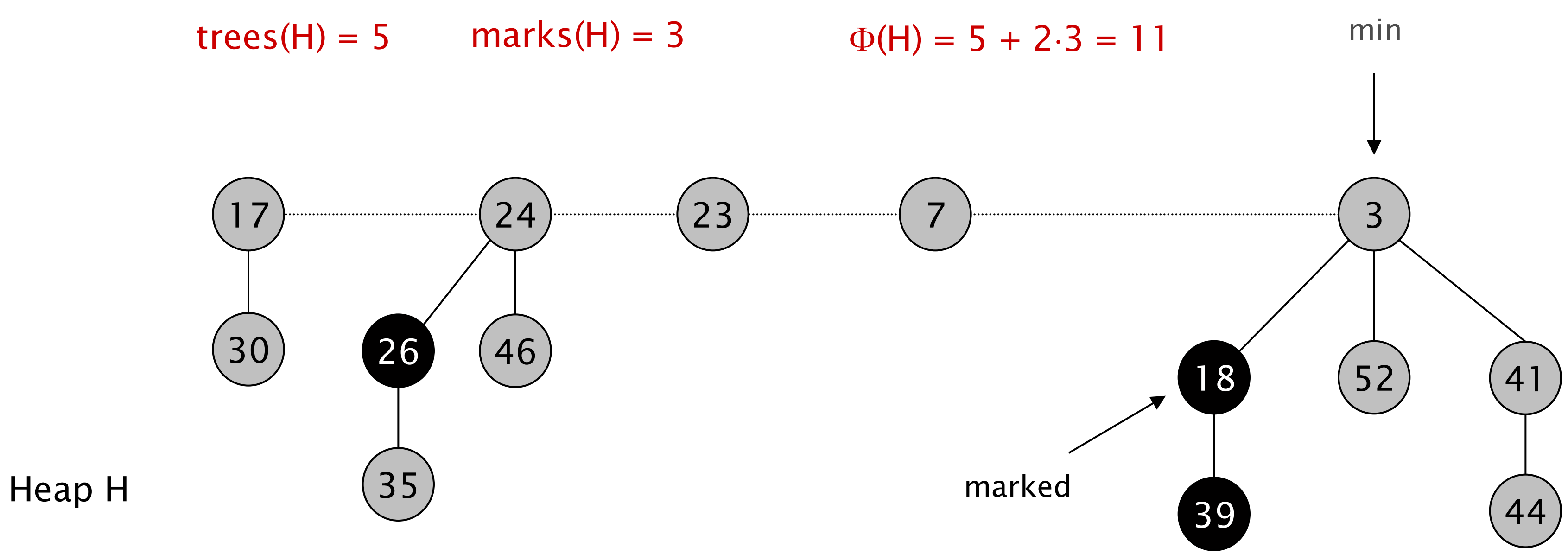


Fibonacci Heaps: Potential Function

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

potential of heap H

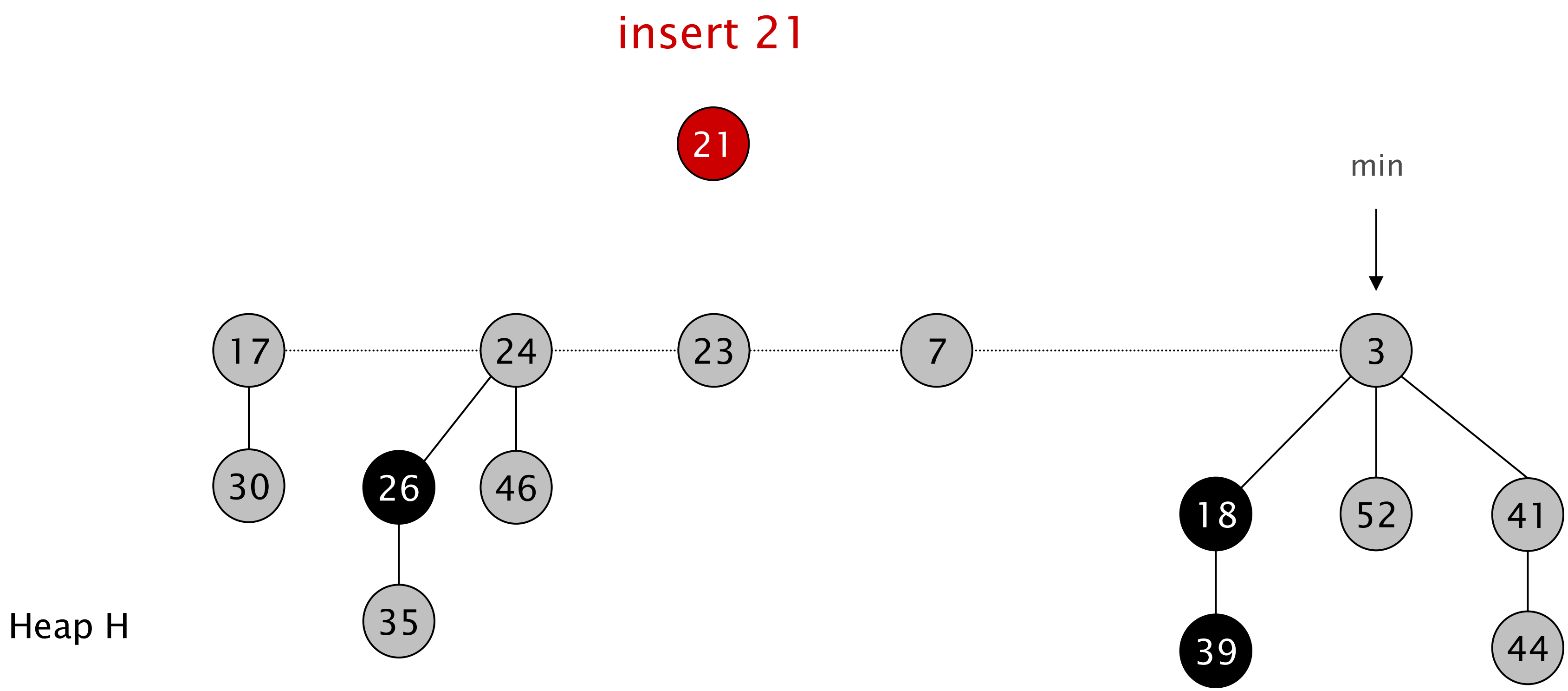
trees(H) = 5 marks(H) = 3 $\Phi(H) = 5 + 2 \cdot 3 = 11$



Insert

Fibonacci Heaps: Insert

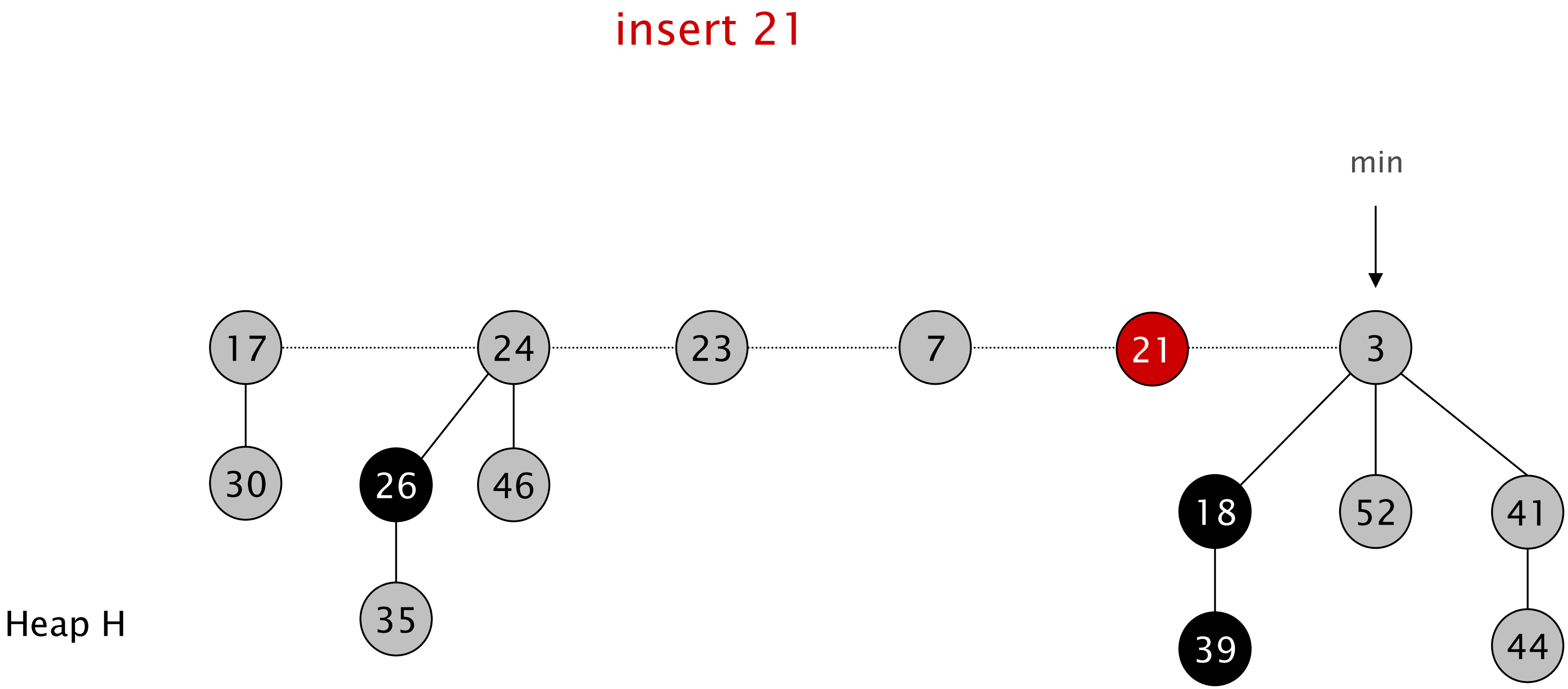
Insert.
- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).



Fibonacci Heaps: Insert

Insert.

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



Fibonacci Heaps: Insert Analysis

Actual cost. O(1)

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

potential of heap H

Change in potential. +1

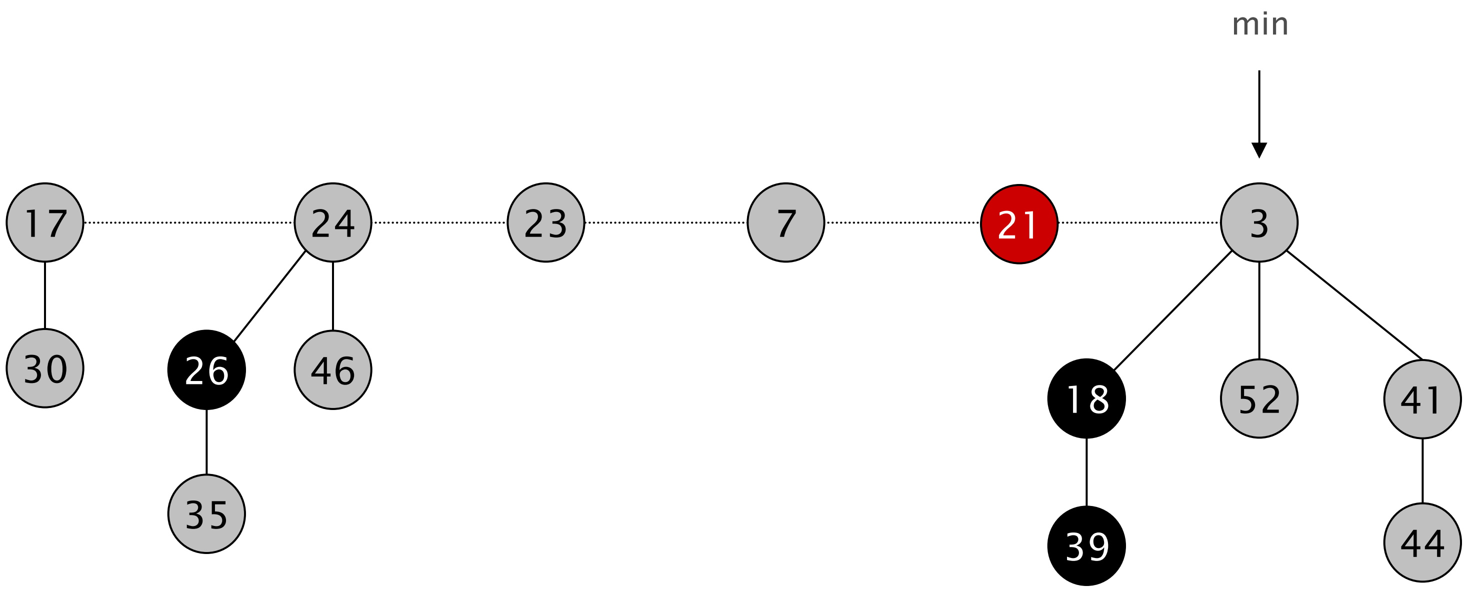
Actual cost. O(1)

$\Phi(H) = 5 + 2 \cdot 3 = 11$

$\Phi(H') = 6 + 2 \cdot 3 = 12$

Change in potential = +1

Amortized cost. O(1)+ 1 = O(1)

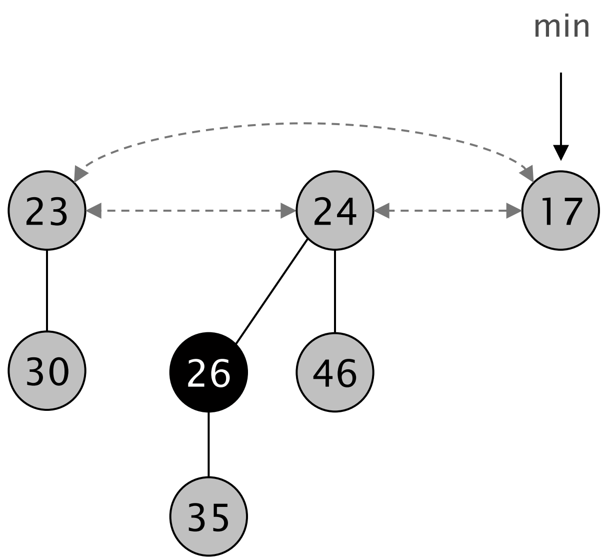


Heap H

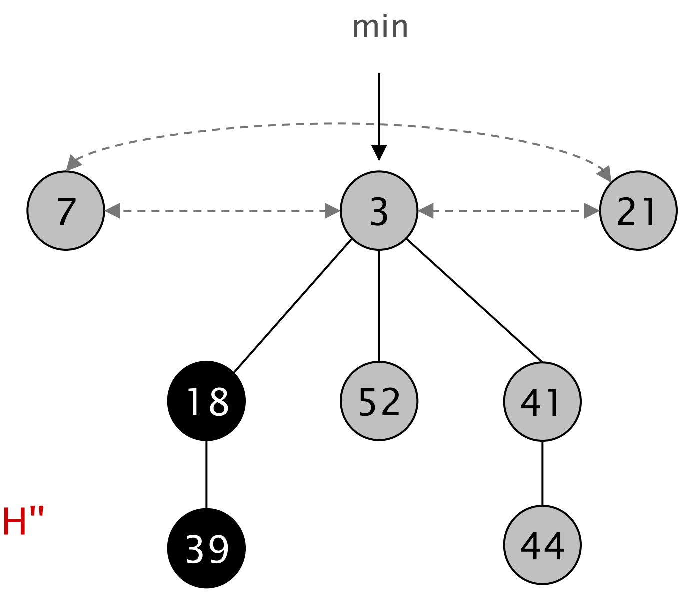
Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Heap H'

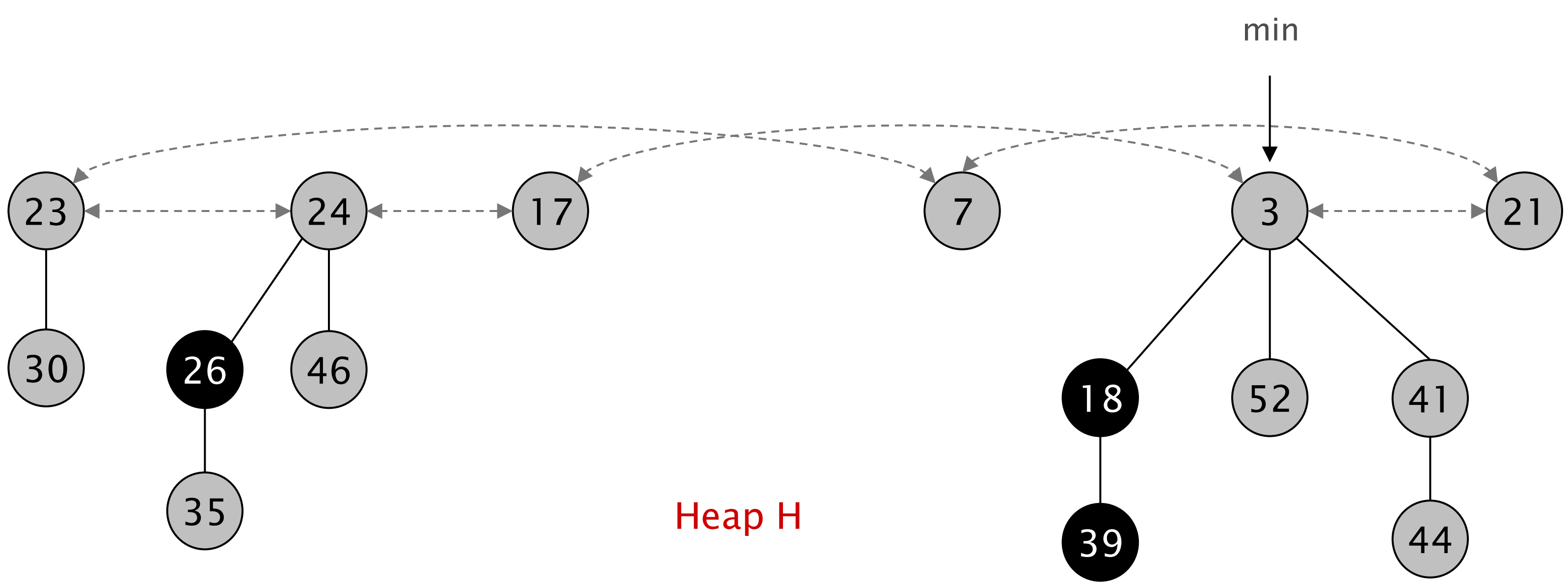


Heap H''

Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

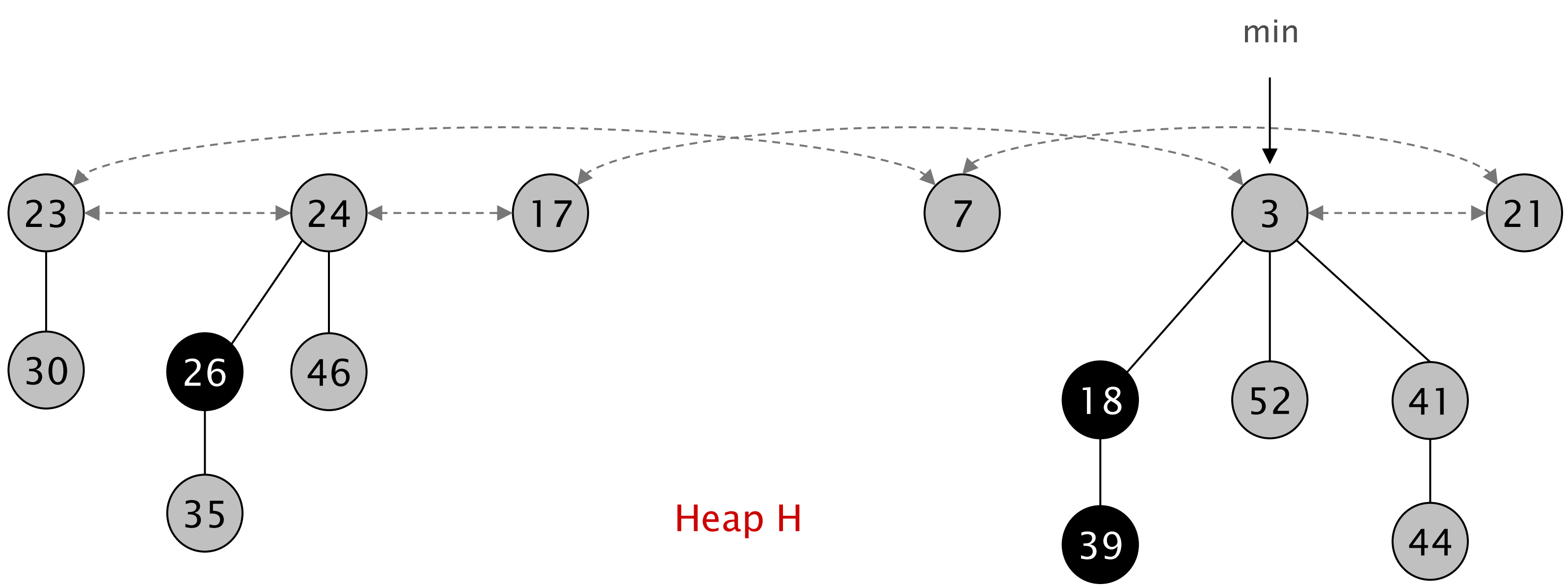
Actual cost. $O(1)$

Change in potential. 0

Amortized cost. $O(1)$

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

potential function

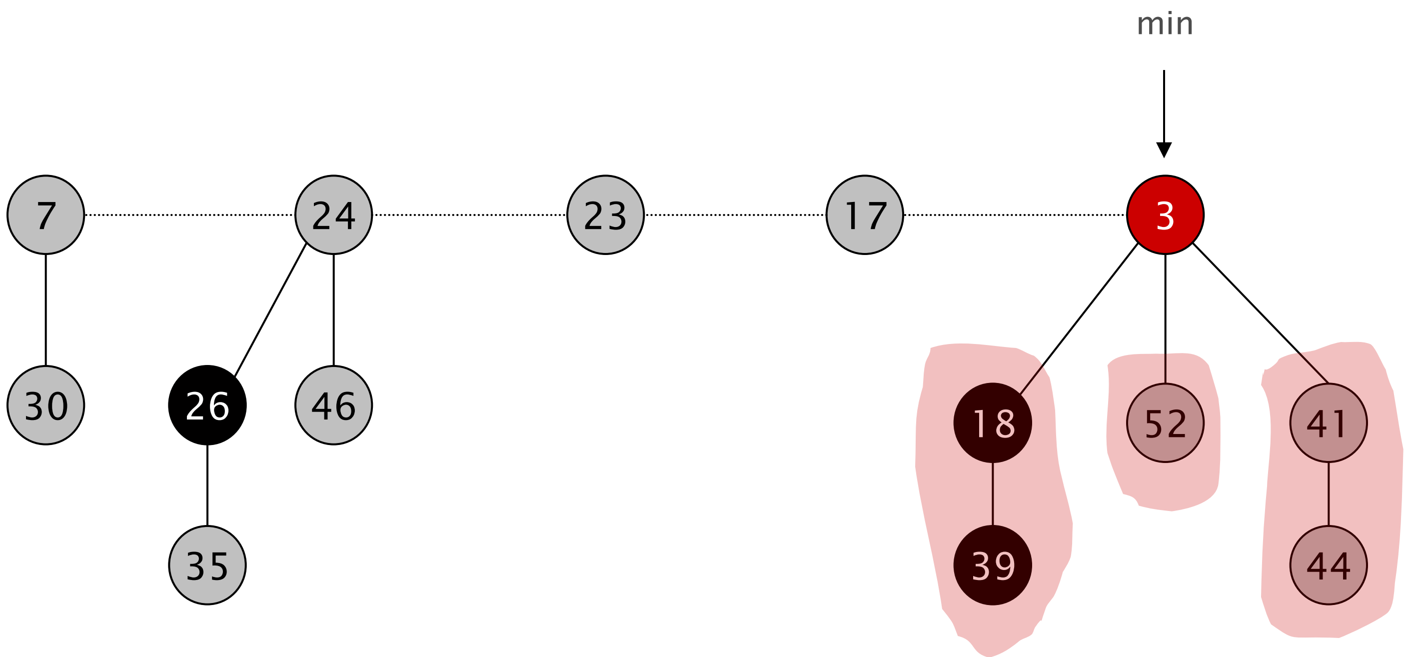


ExtractMin

Fibonacci Heaps: Extract Min

Extract min.

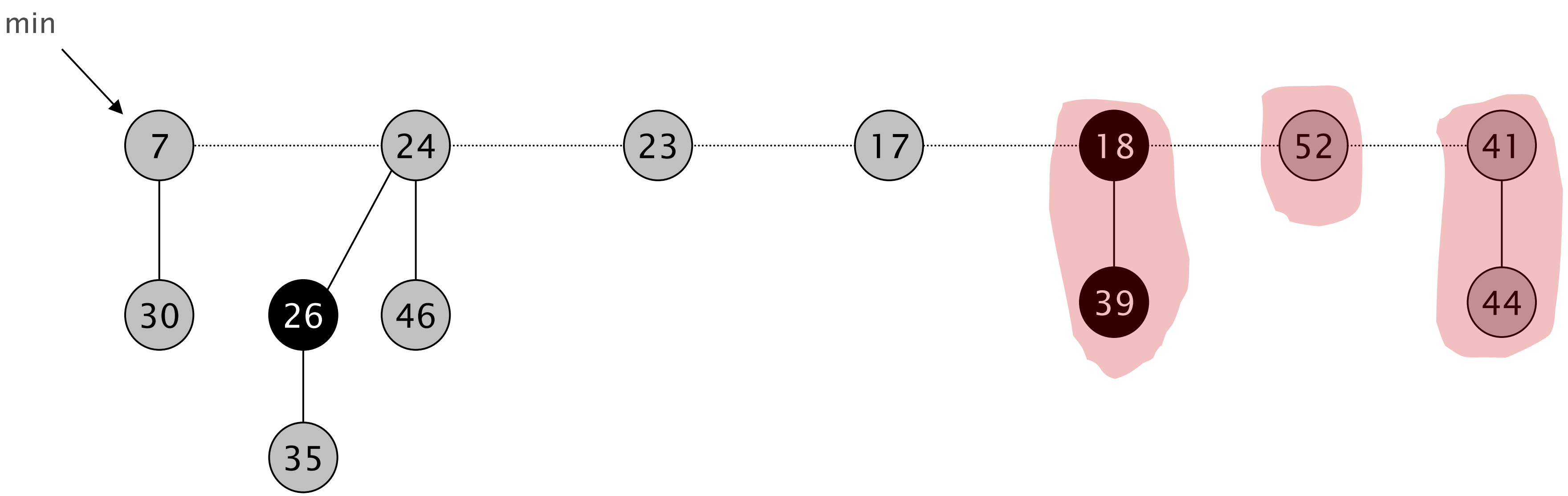
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

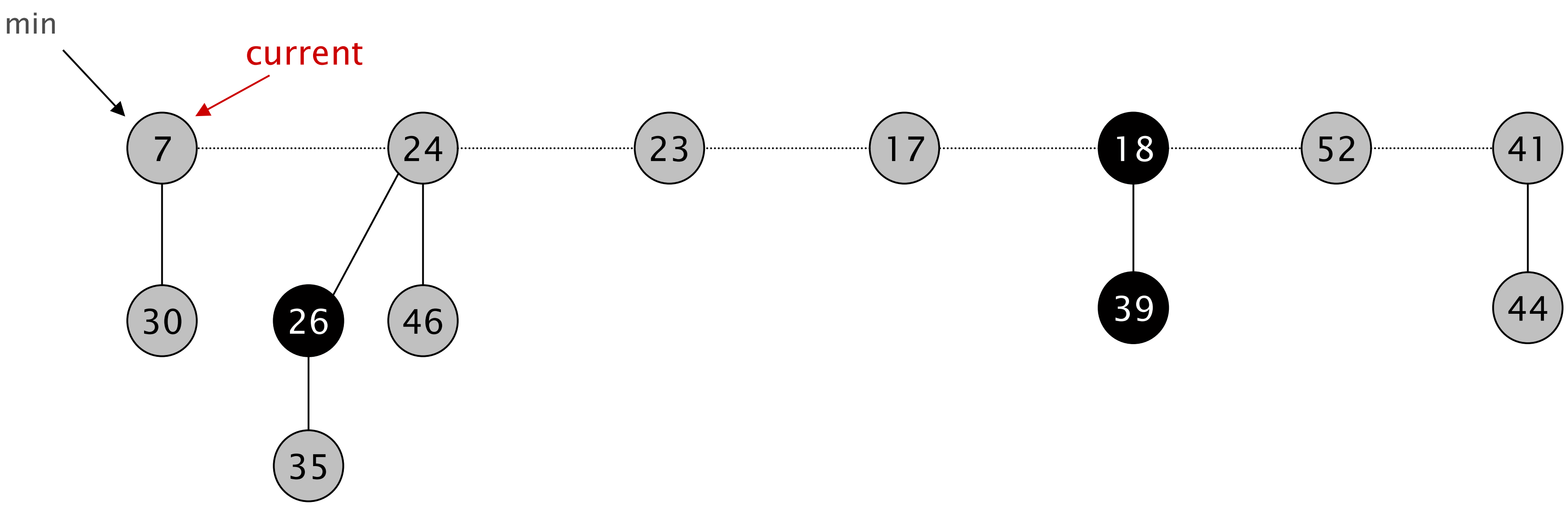
Extract min.

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



Extract min.

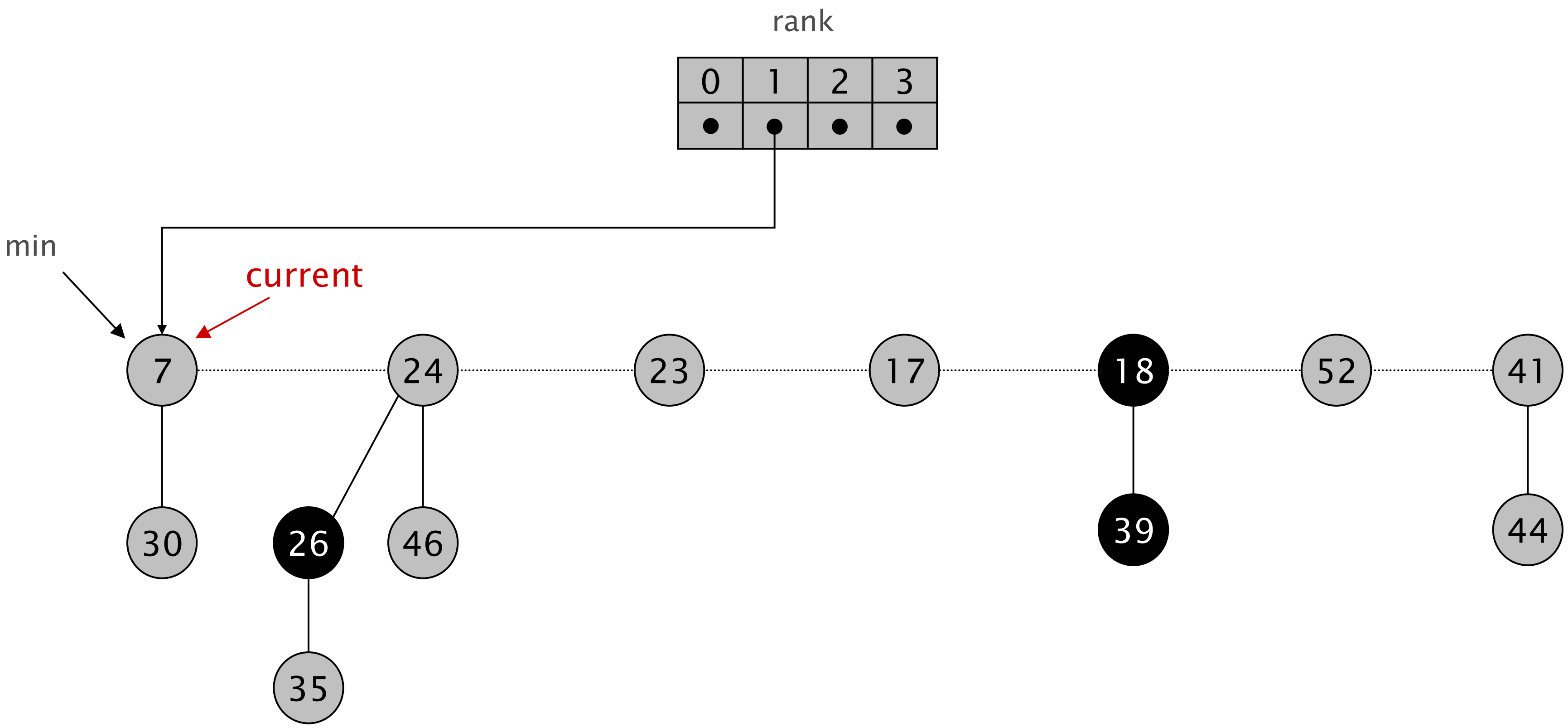
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Extract Min

Extract min.

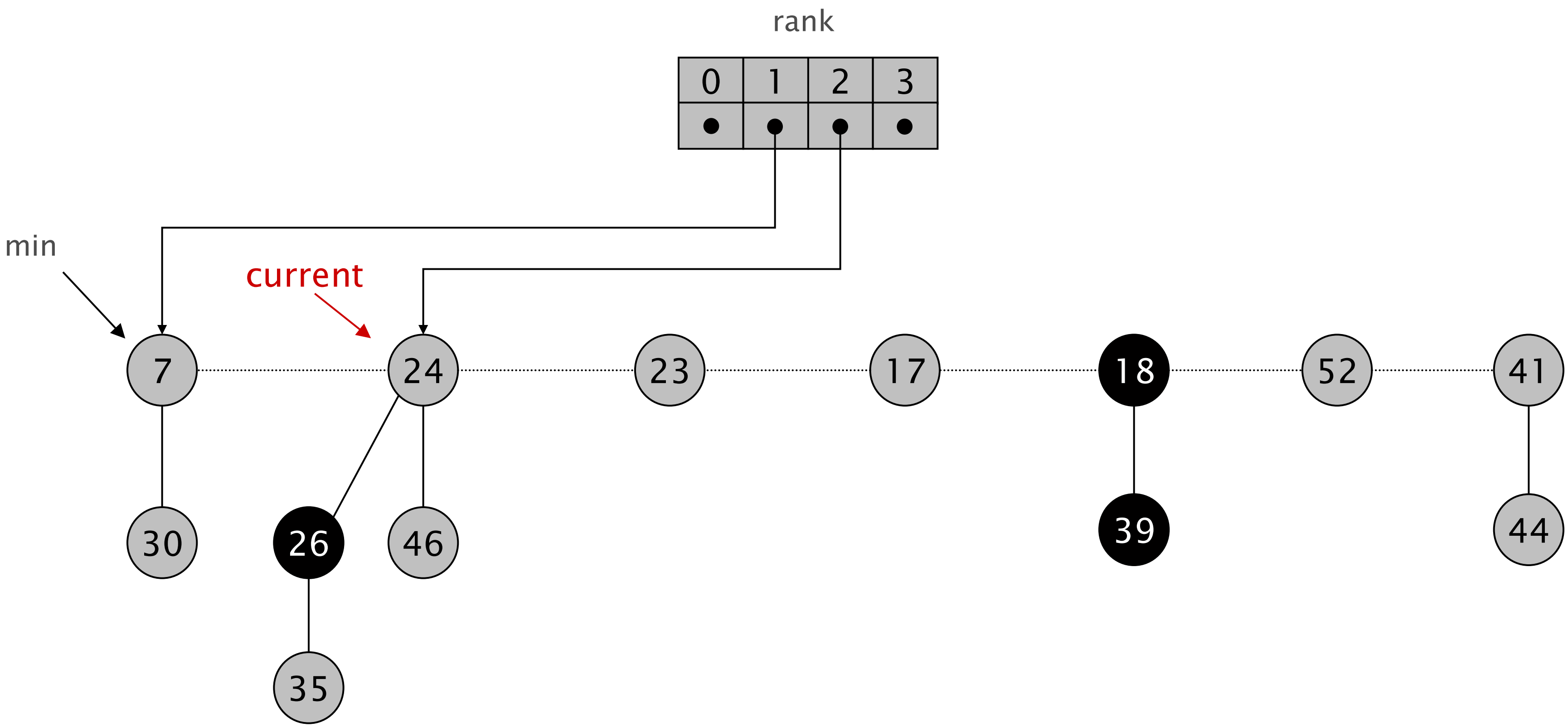
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Extract Min

Extract min.

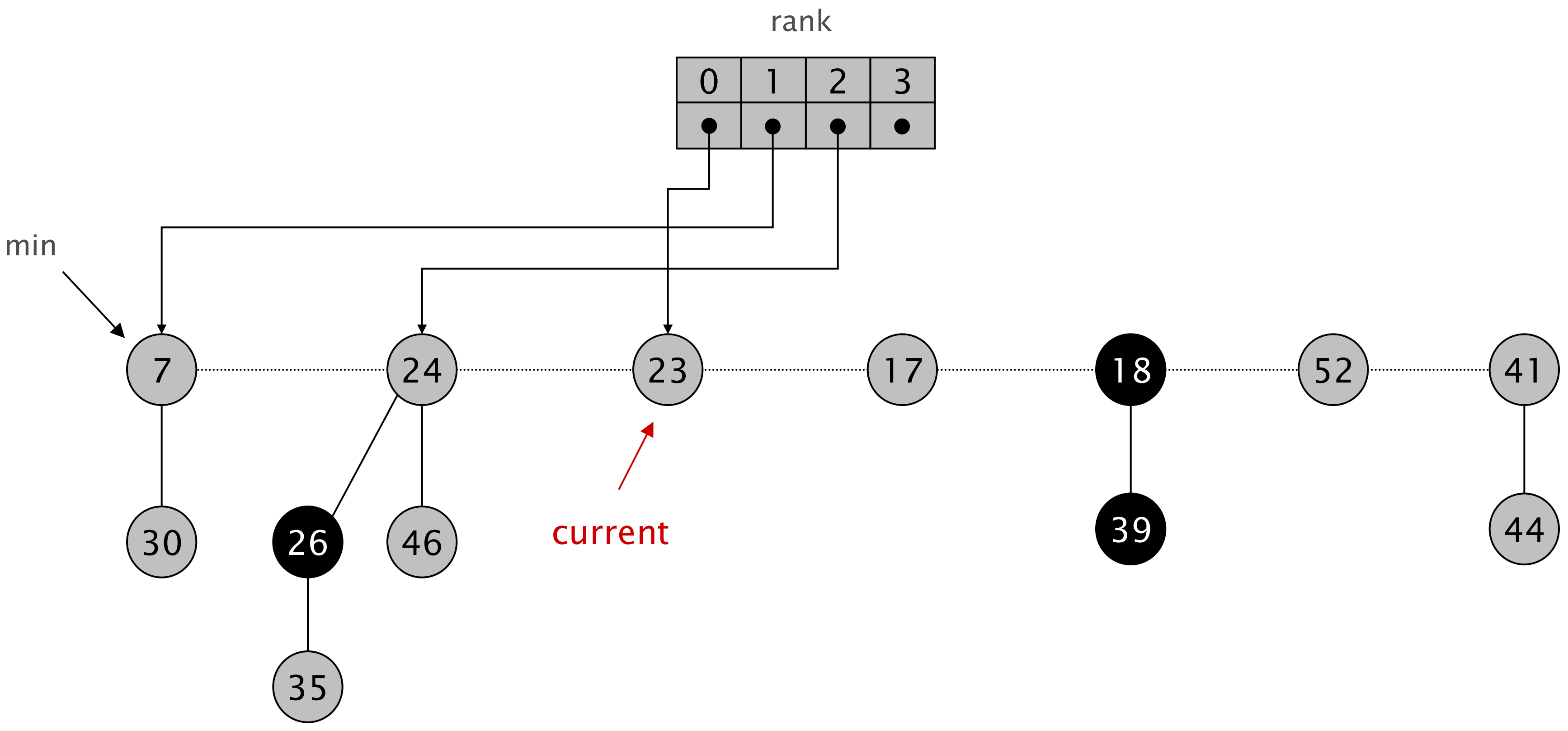
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Extract Min

Extract min.

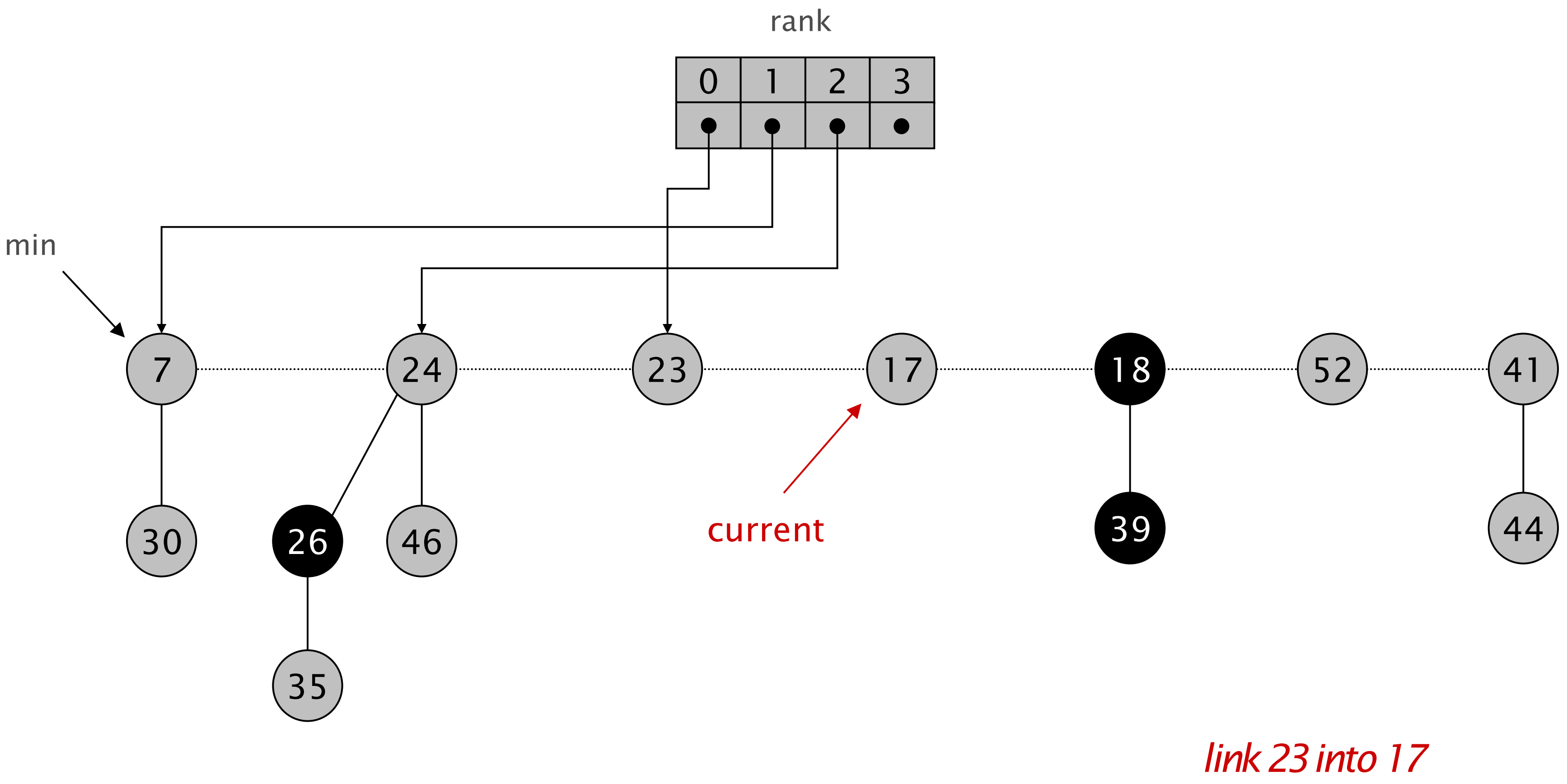
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Extract Min

Extract min.

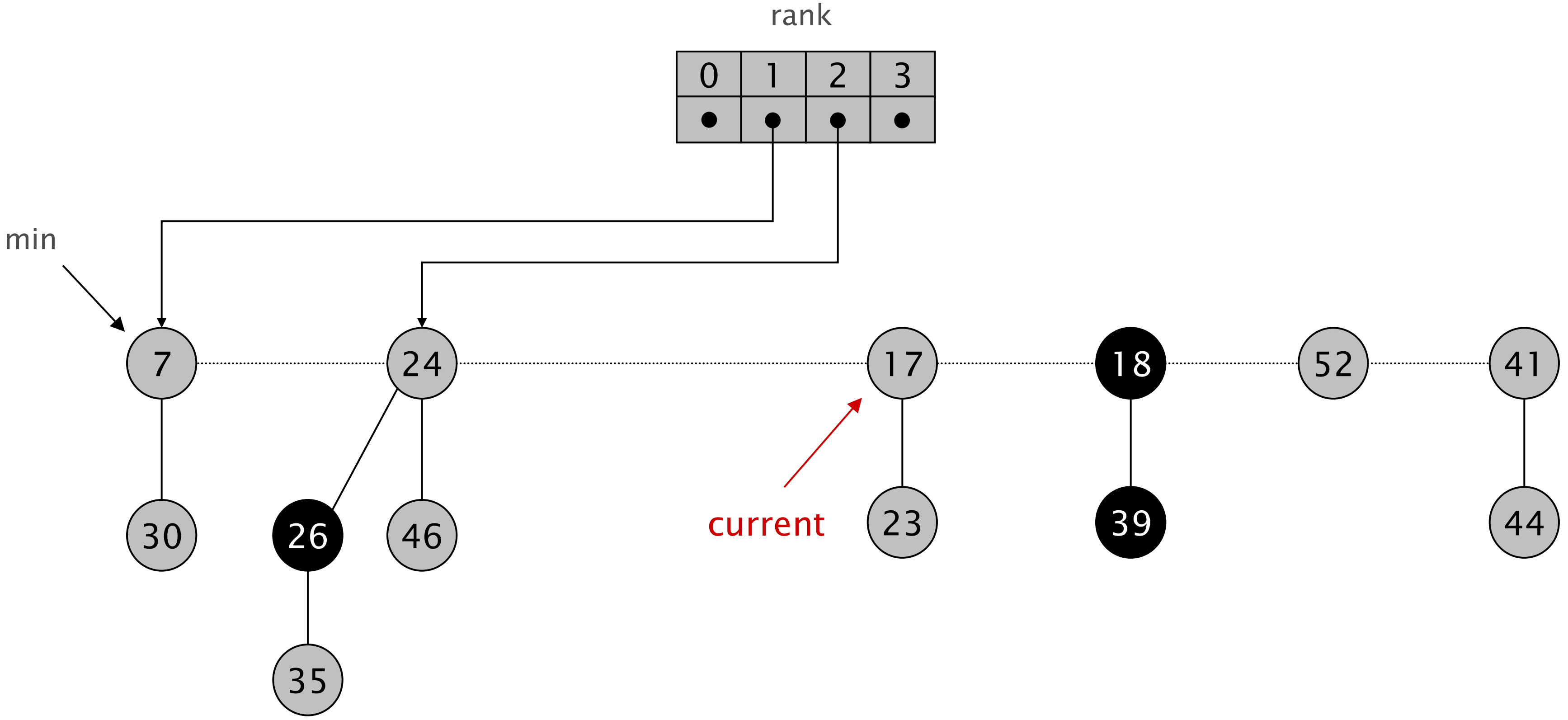
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Extract Min

Extract min.

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

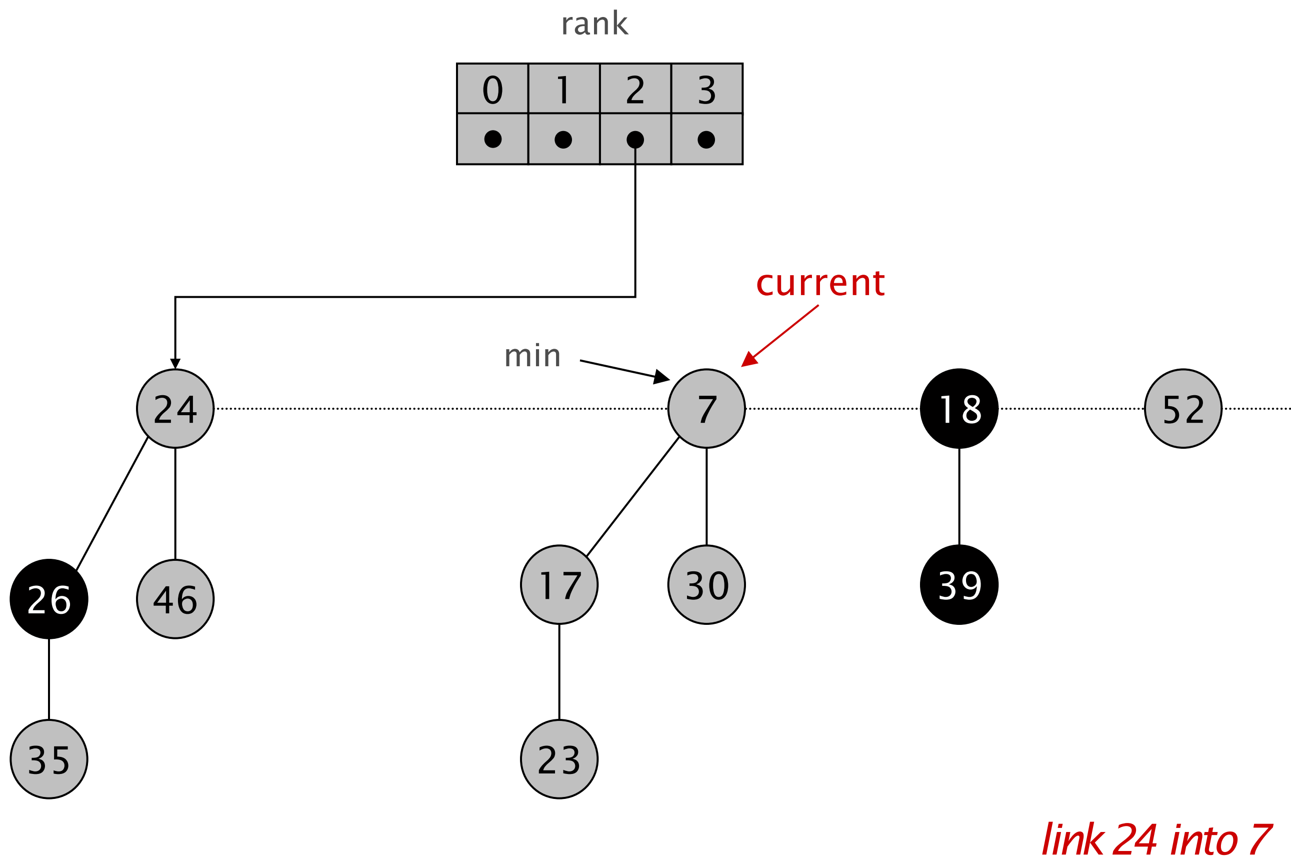


link 17 into 7

Fibonacci Heaps: Extract Min

Extract min.

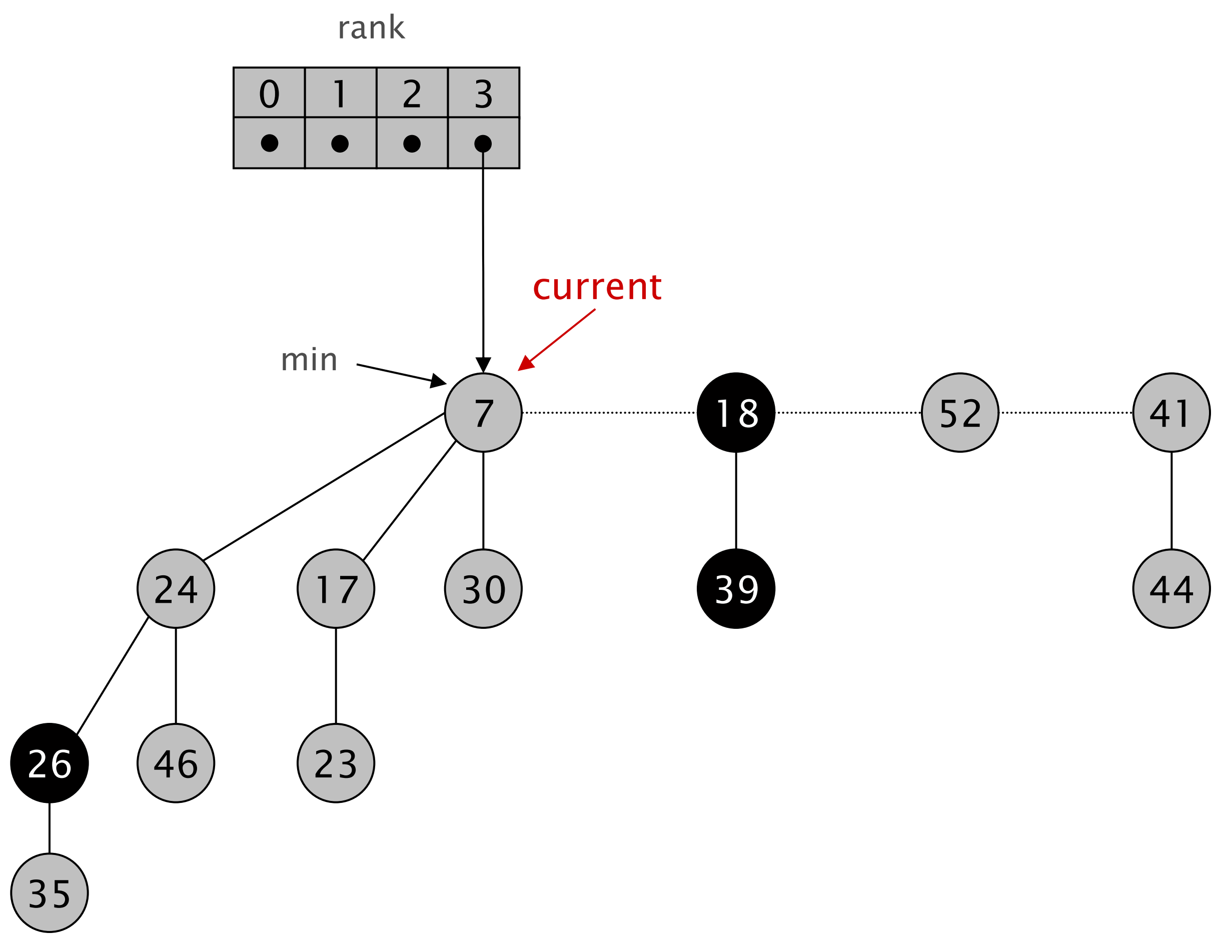
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Extract Min

Extract min.

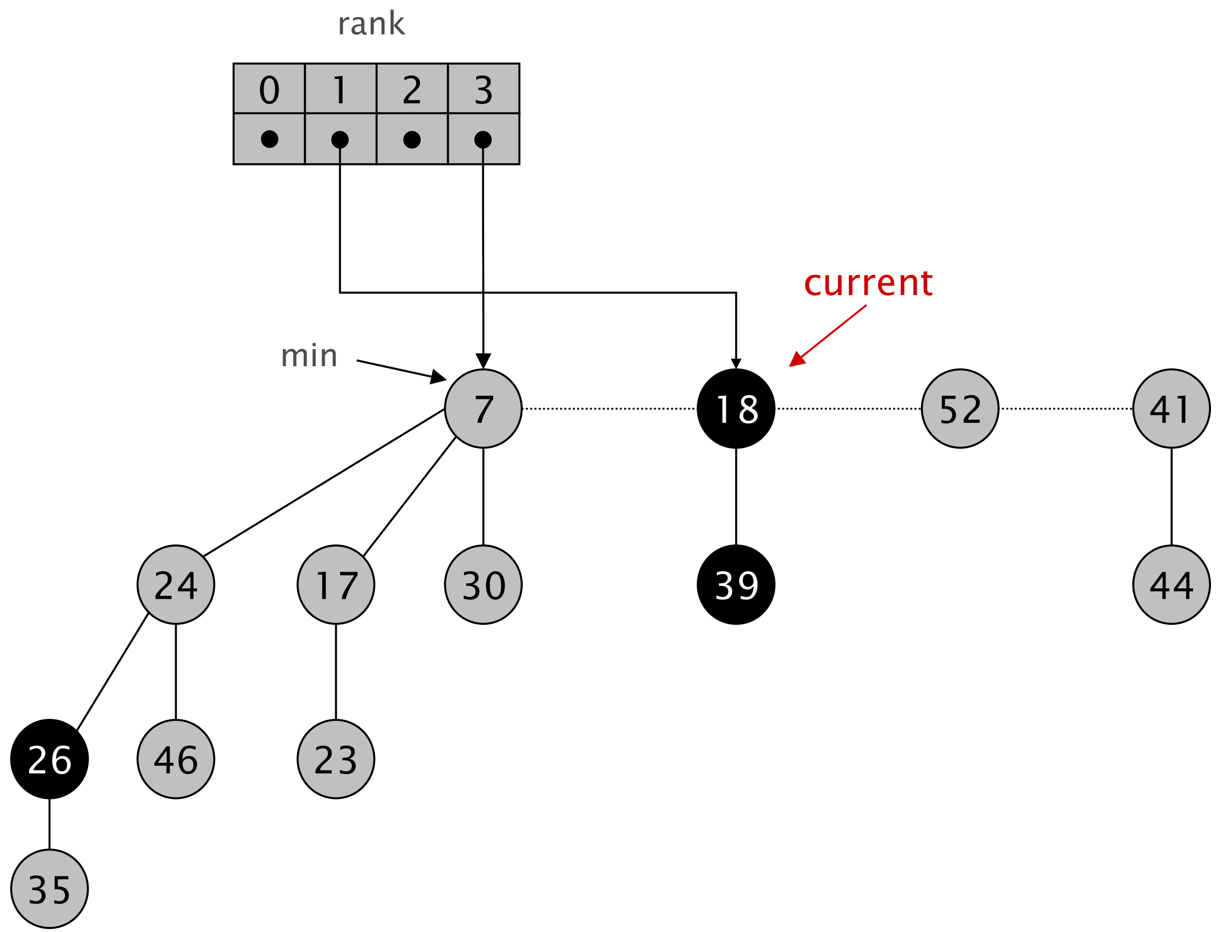
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Extract Min

Extract min.

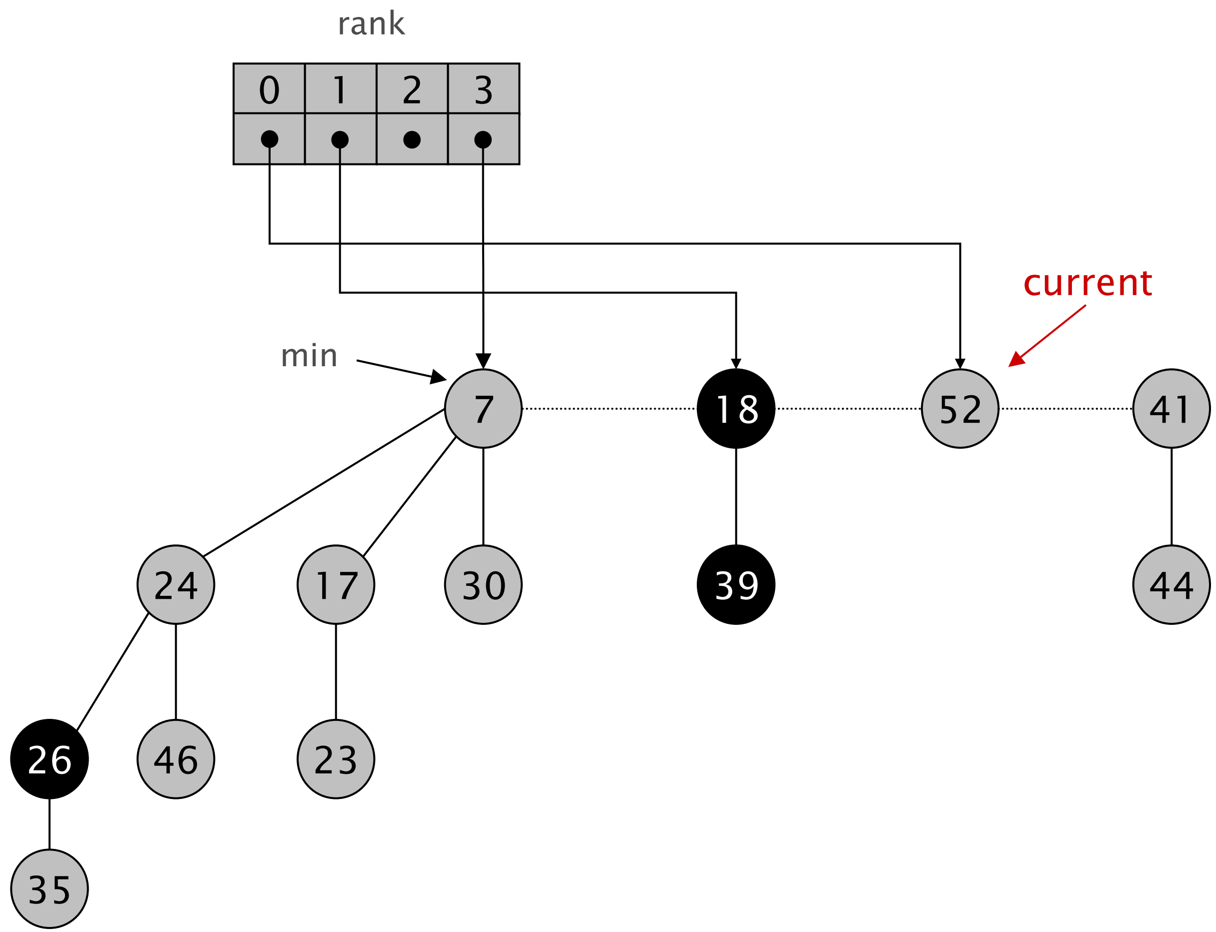
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

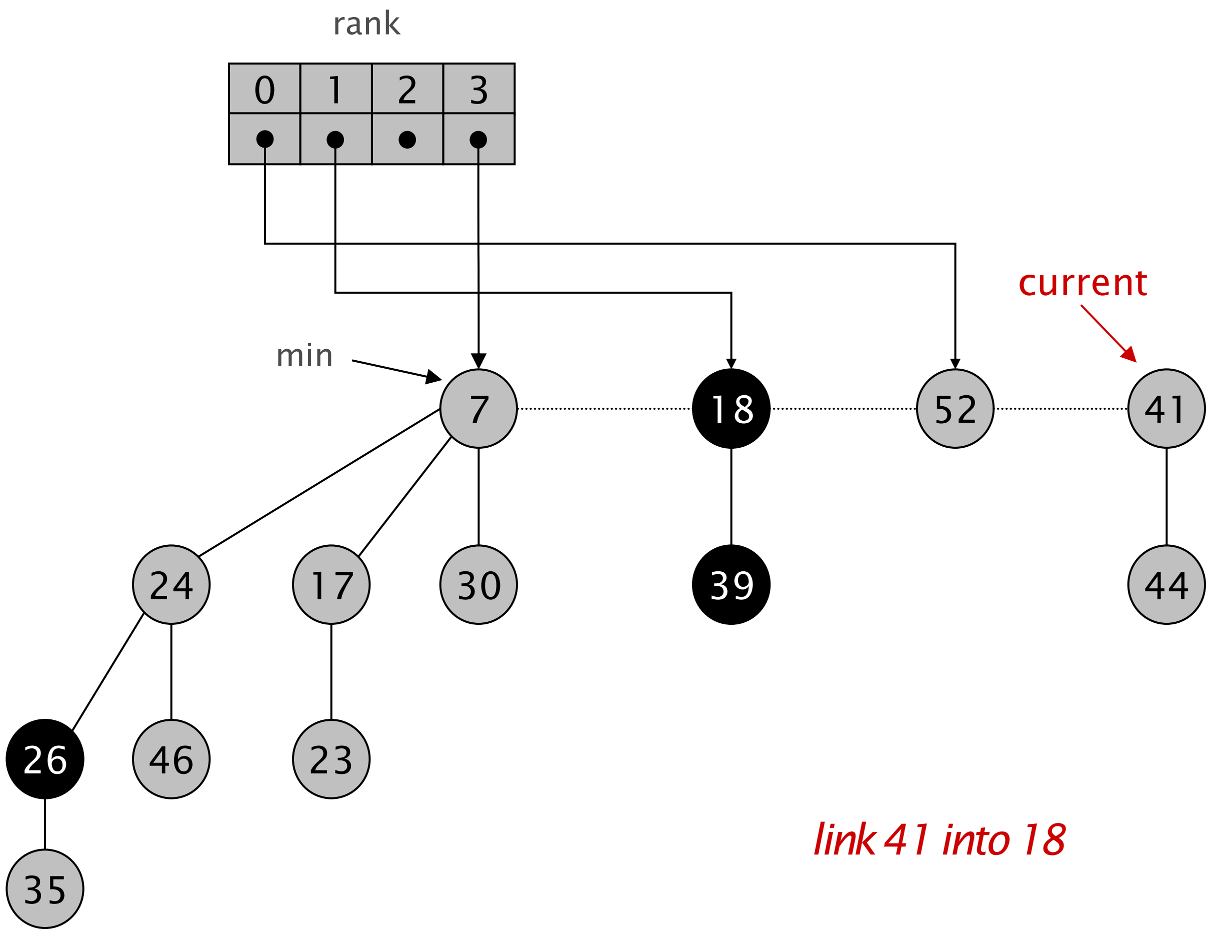
Extract min.

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



Extract min.

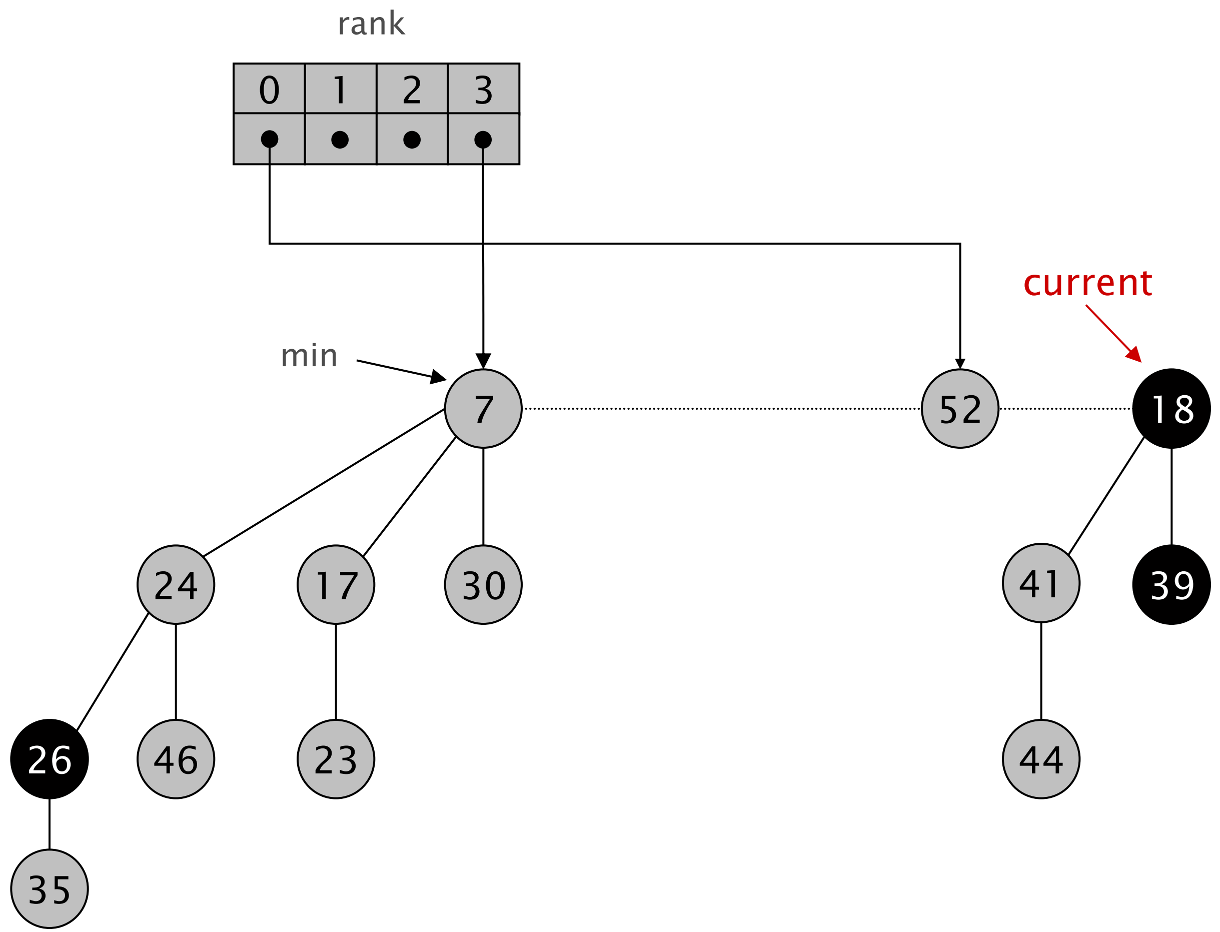
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

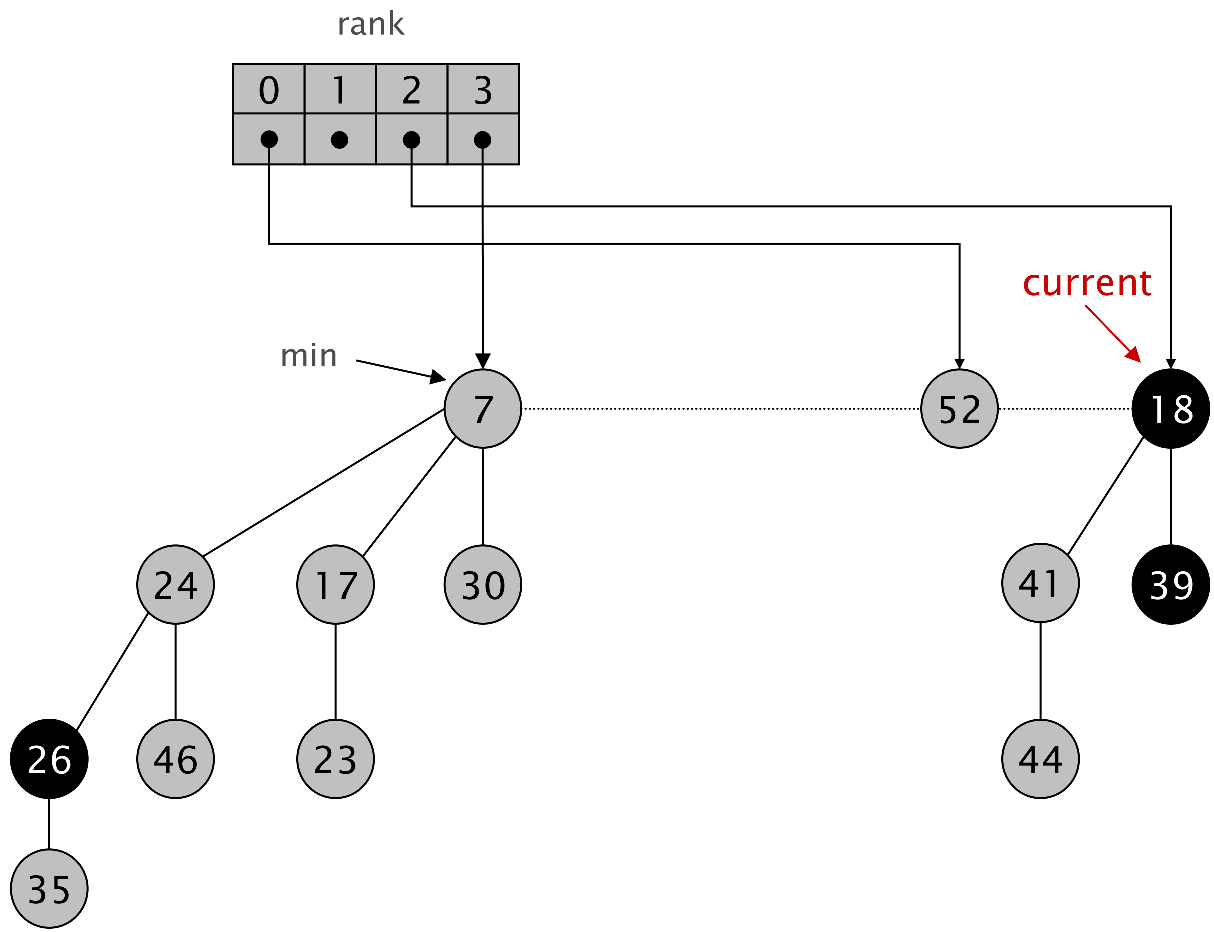
Extract min.

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



Extract min.

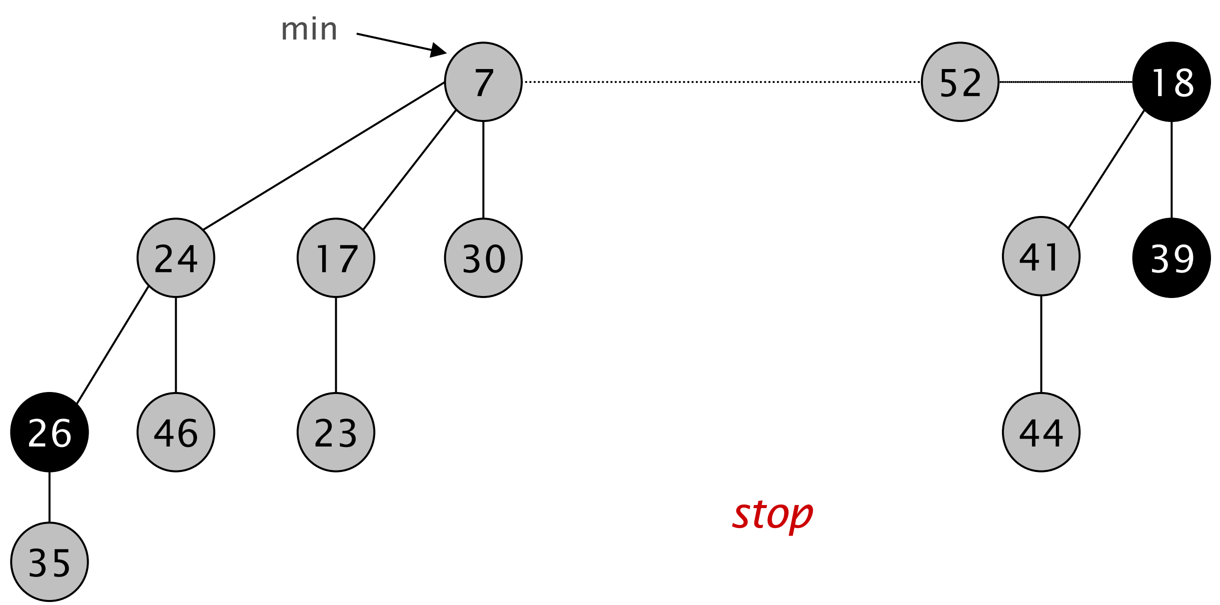
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Extract Min

Extract min.

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



Fibonacci Heaps: ExtractMin Analysis

Extract min.

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

potential function

Actual cost. $O(\text{rank}(H)) + O(\text{trees}(H))$

- $O(\text{rank}(H))$ to meld min's children into root list.
- $O(\text{rank}(H)) + O(\text{trees}(H))$ to update min.
- $O(\text{rank}(H)) + O(\text{trees}(H))$ to consolidate trees.

Change in potential. $O(\text{rank}(H)) - \text{trees}(H)$

- $\text{trees}(H') \le \text{rank}(H) + 1$ since no two trees have same rank.
- $\Delta\Phi(H) \le \text{rank}(H) + 1 - \text{trees}(H)$.

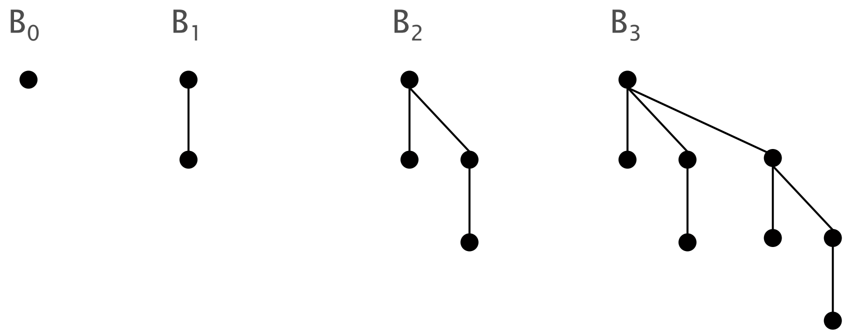
Amortized cost. $O(\text{rank}(H))$

Fibonacci Heaps: Delete Min Analysis

Q. Is amortized cost of O(rank(H)) good?

A. Yes, if only insert and delete-min operations.
- In this case, all trees are binomial trees.
- This implies rank(H) $\leq$ lg n.

we only link trees of equal rank

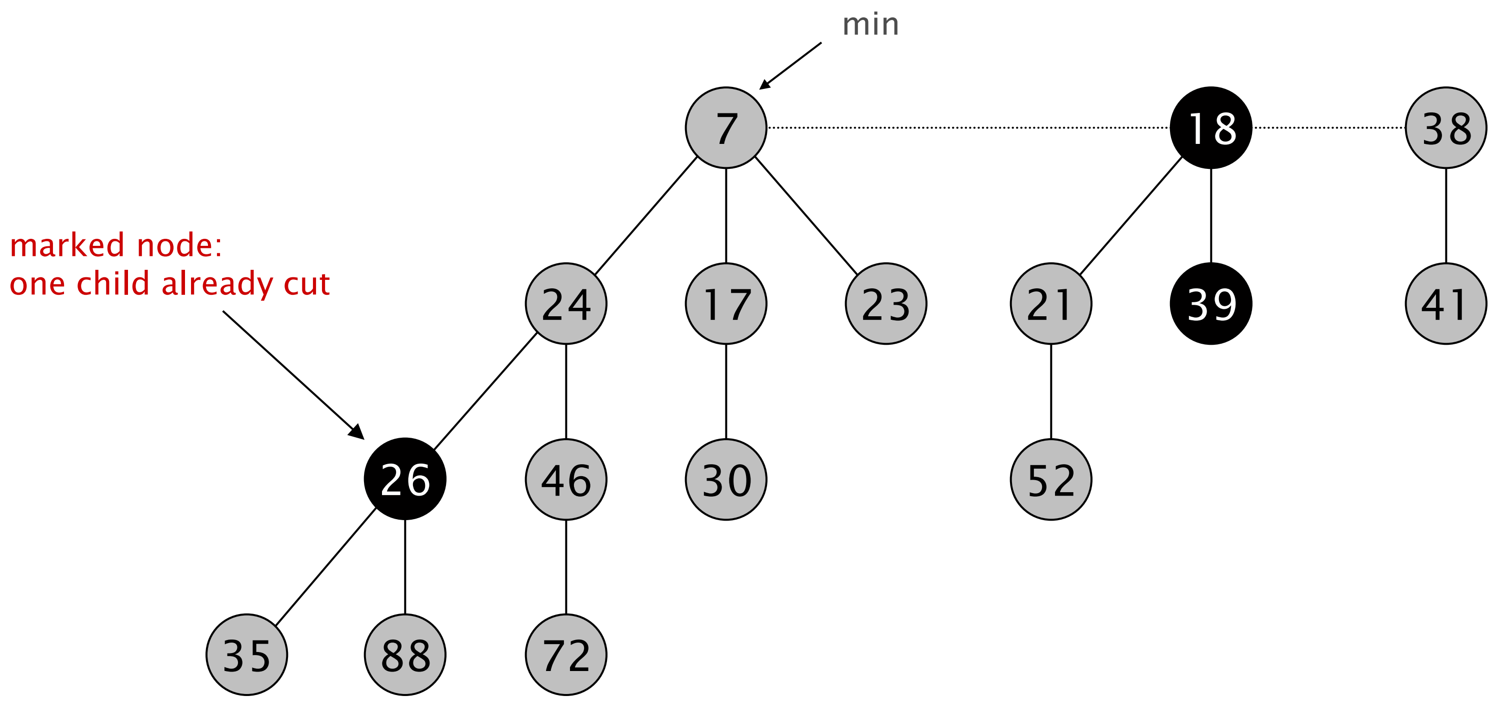


A. Yes, we'll implement decrease-key so that rank(H) = O(log n).

Decrease Key

Intuition for deceasing the key of node x.

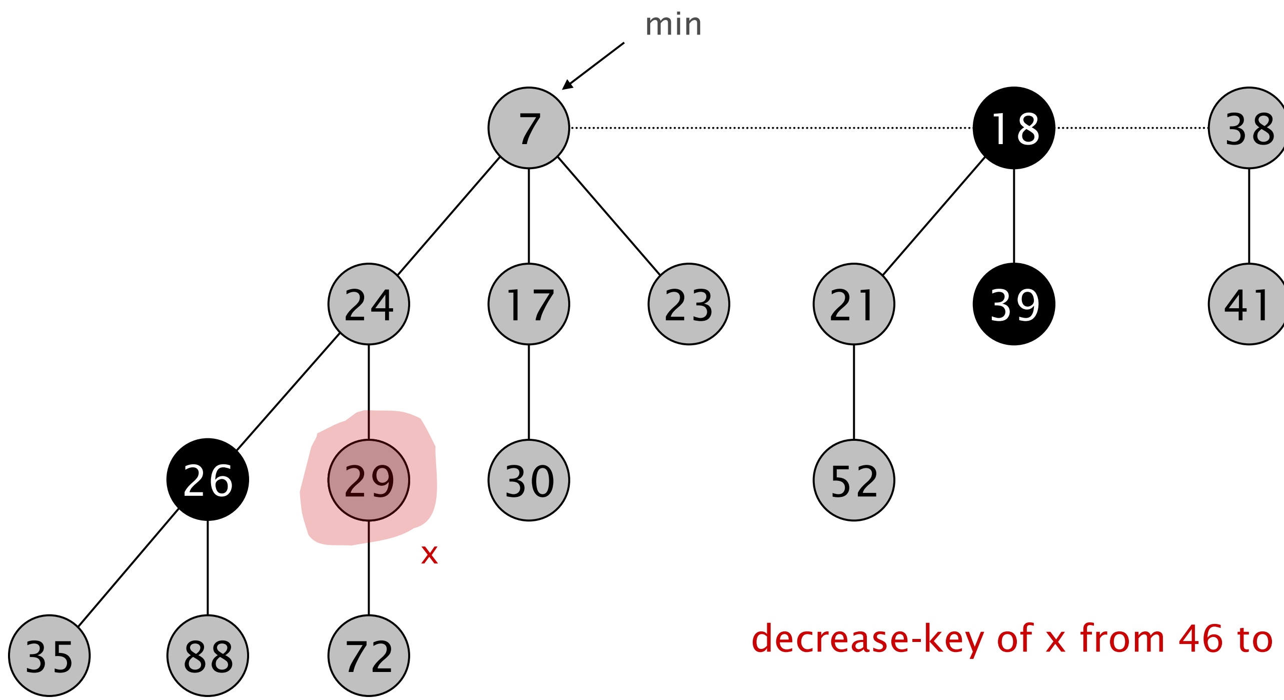
- If heap-order is not violated, just decrease the key of x.
- Otherwise, cut tree rooted at x and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

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

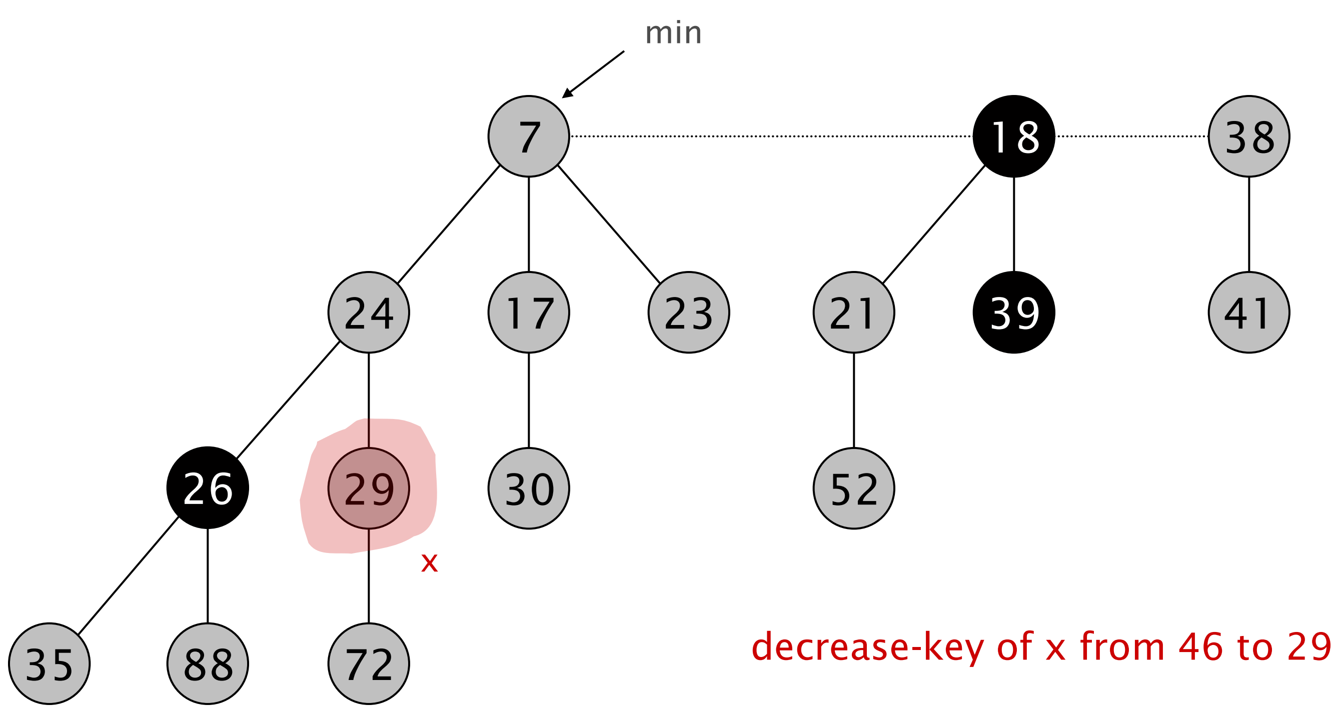


decrease-key of x from 46 to 29

Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

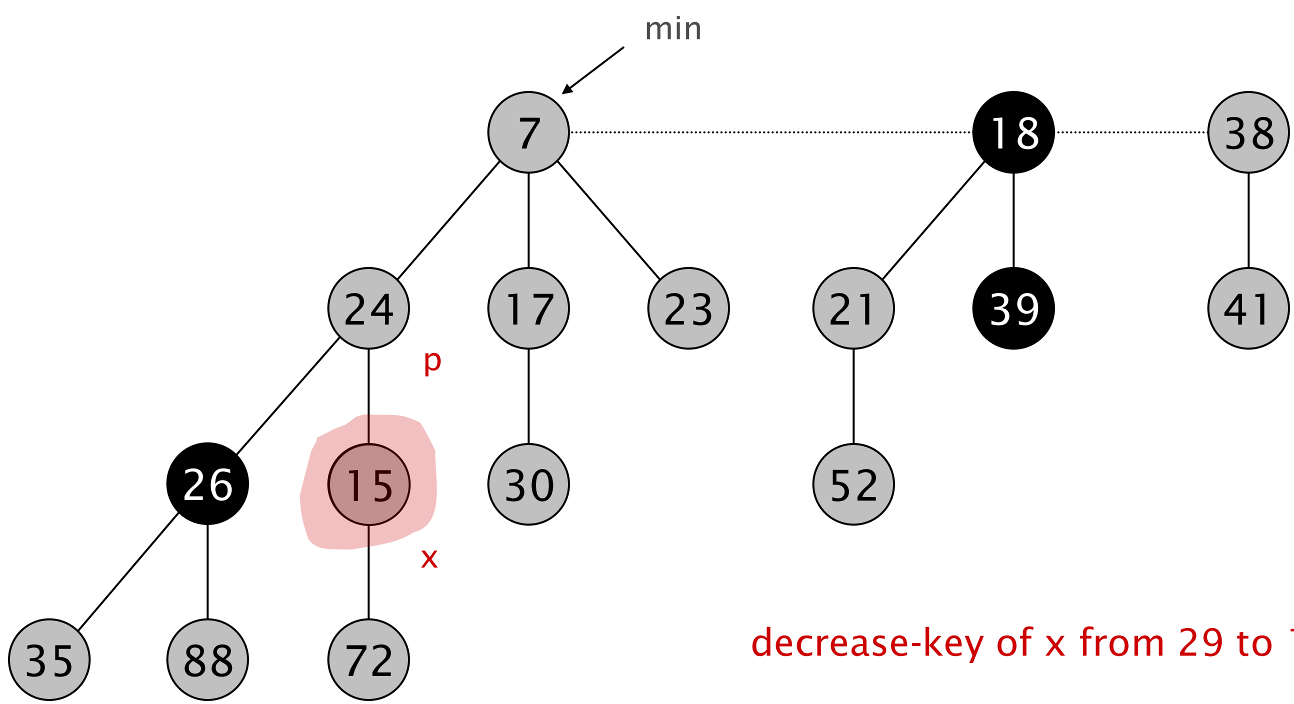
- Decrease key of x.
- Change heap min pointer (if necessary).



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).

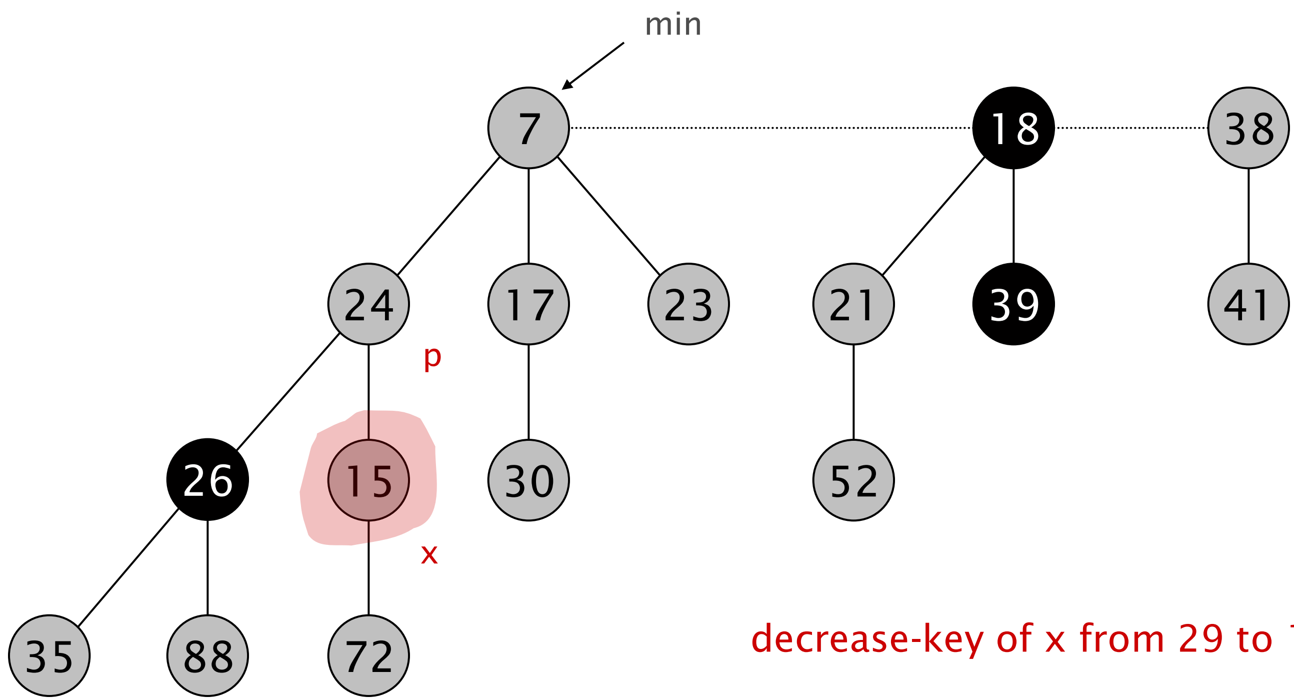


decrease-key of x from 29 to 15

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).

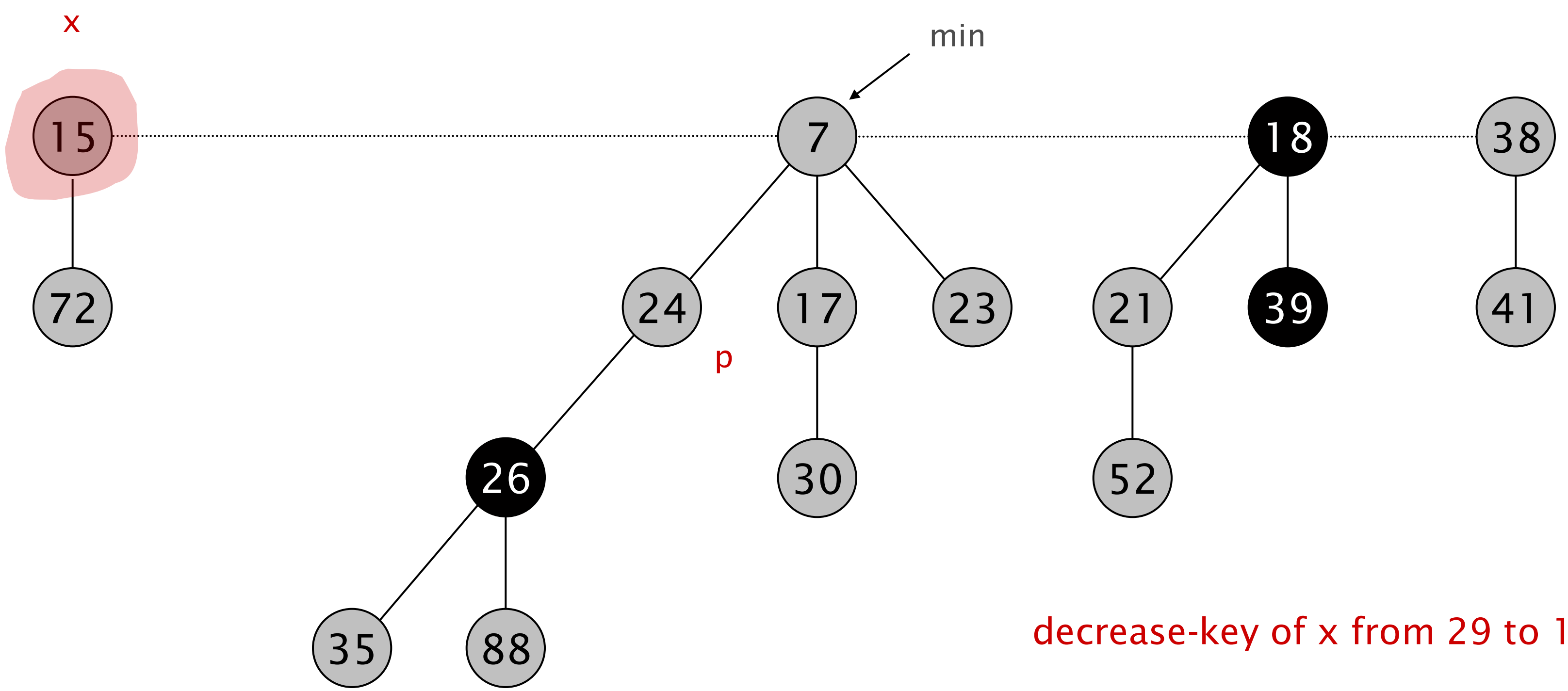


decrease-key of x from 29 to 15

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).

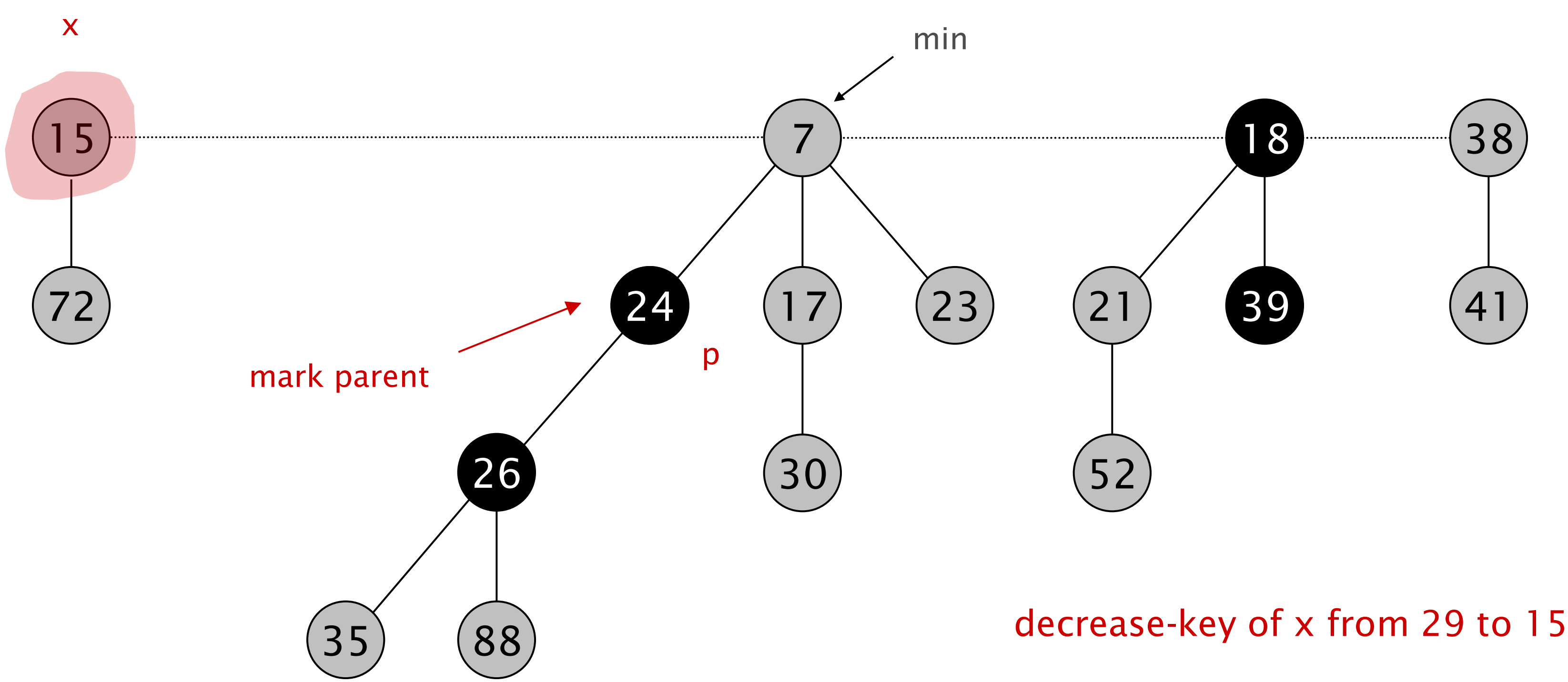


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

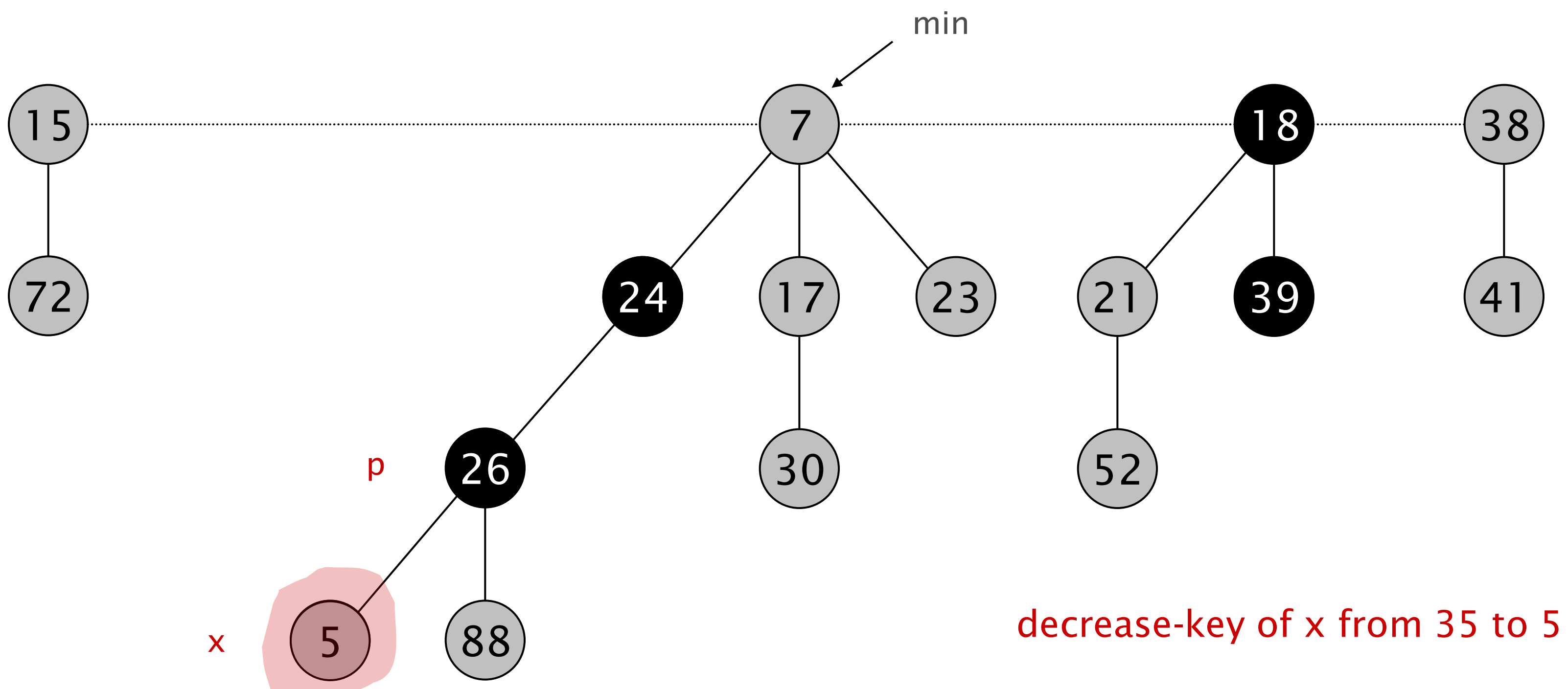
Case 2a. [heap order violated]

- 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).



Case 2b. [heap order violated]

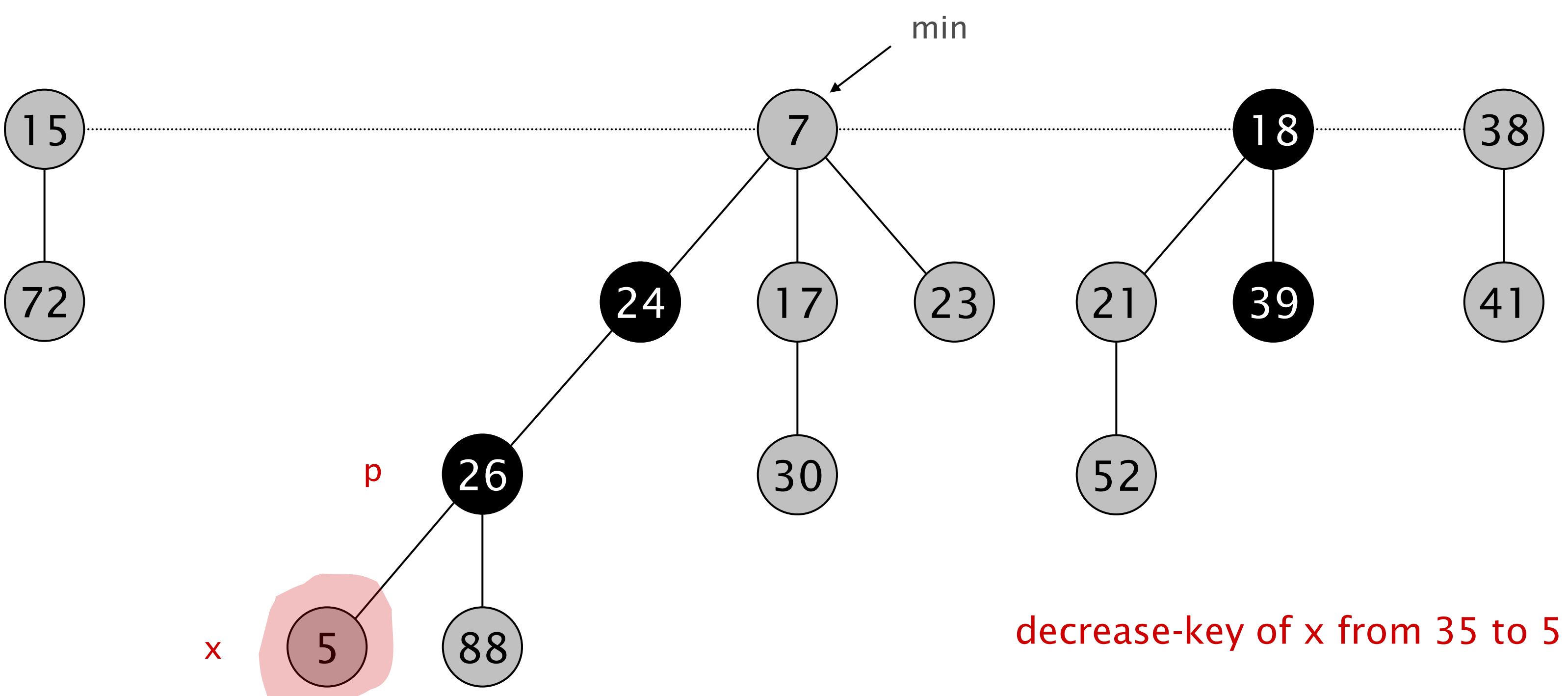
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

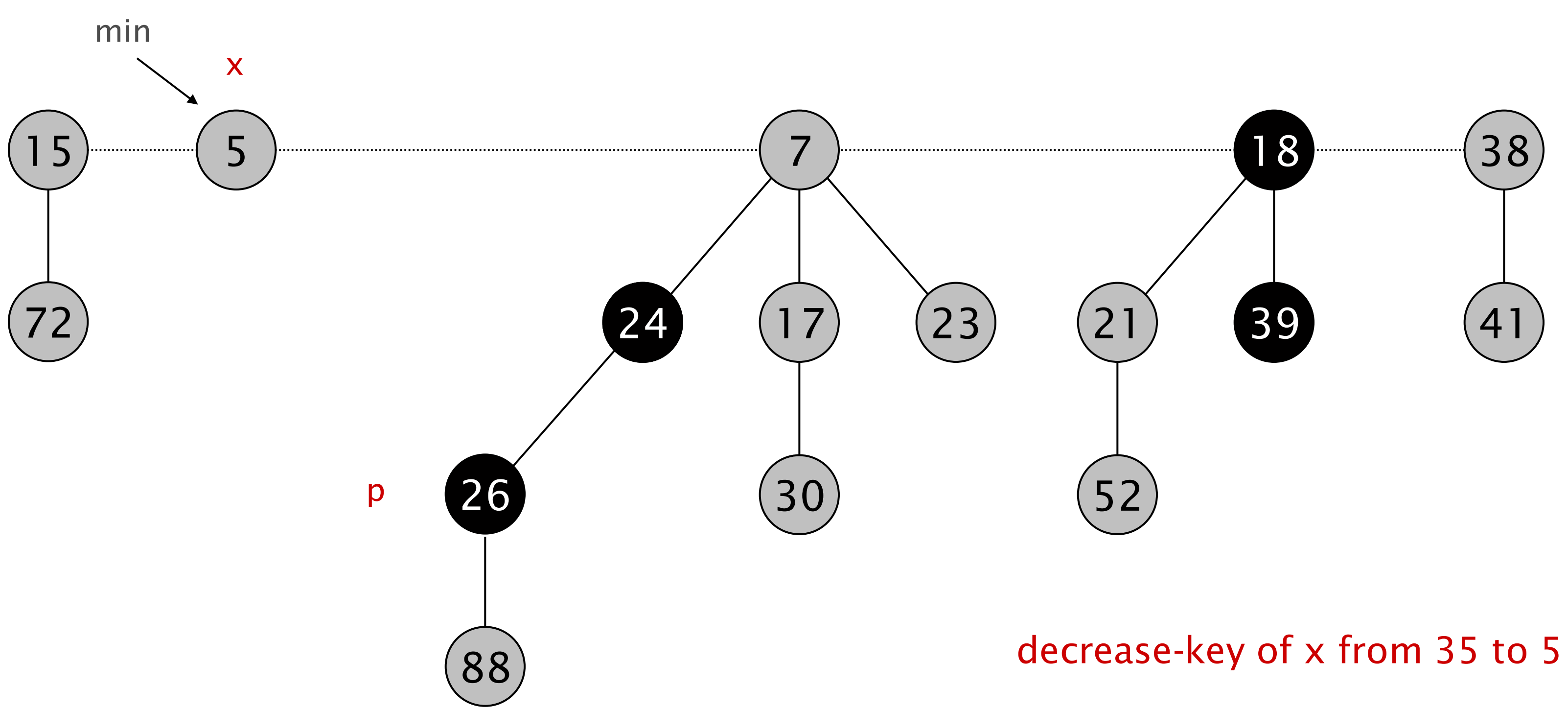
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

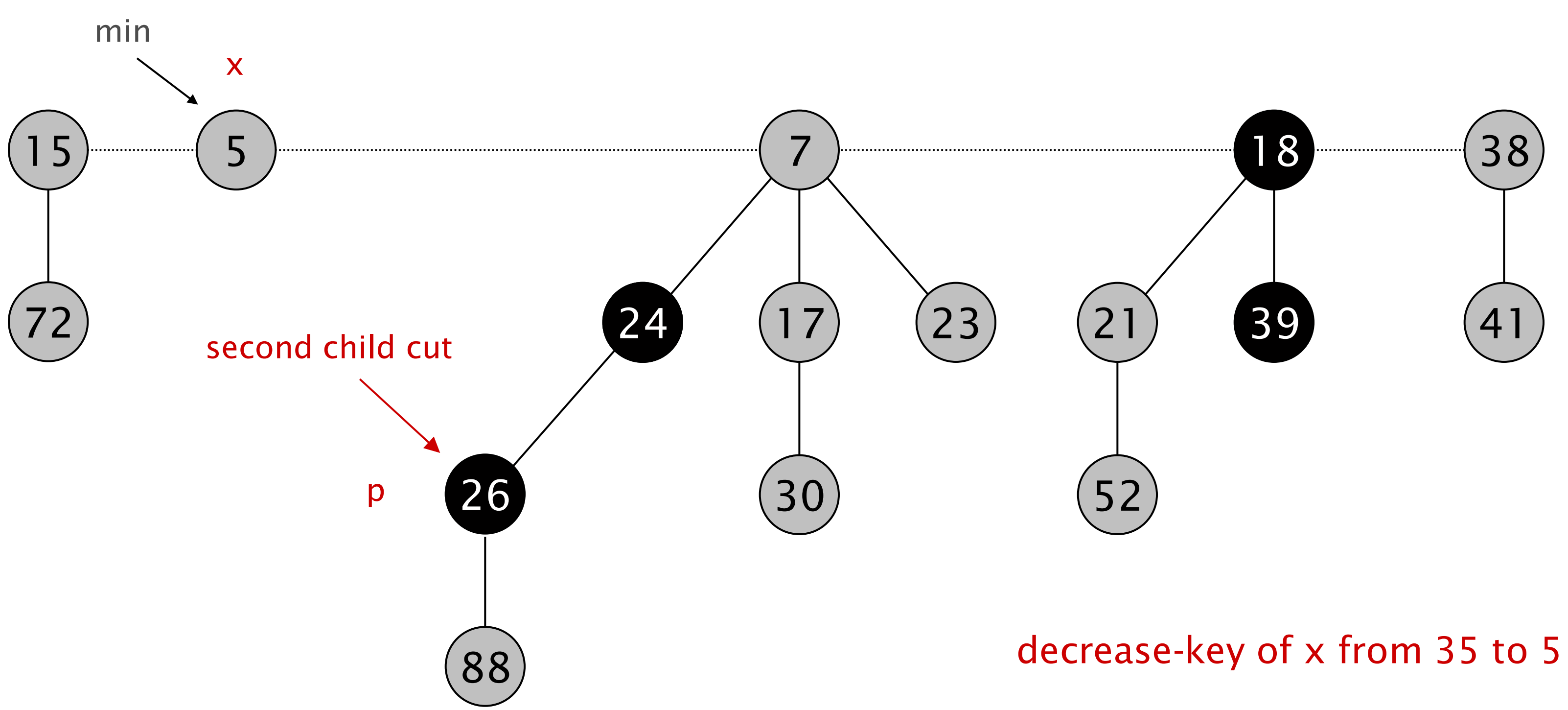
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

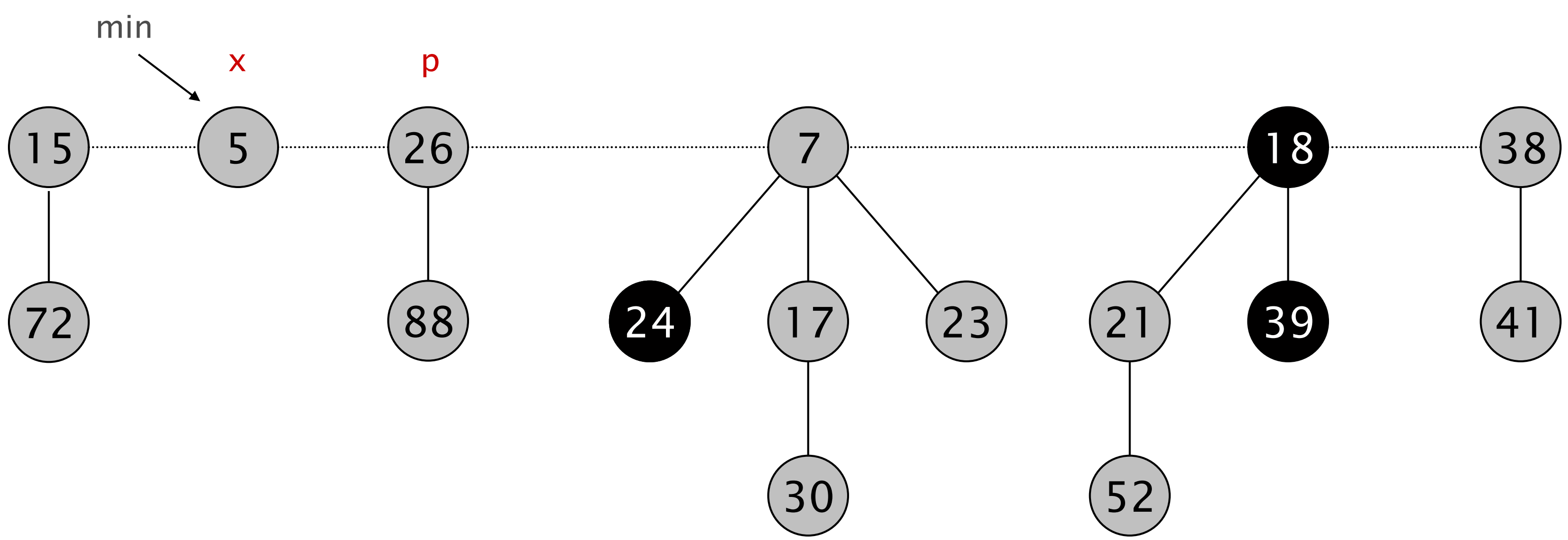
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



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).

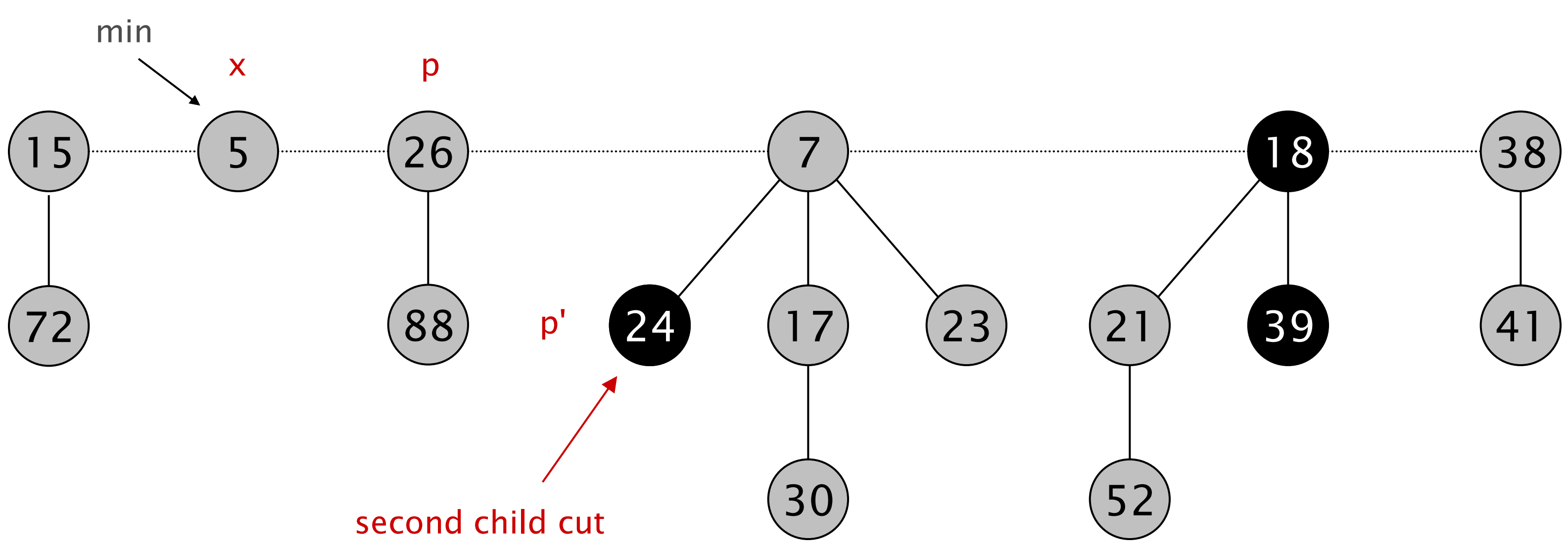


decrease-key of x from 35 to 5

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).

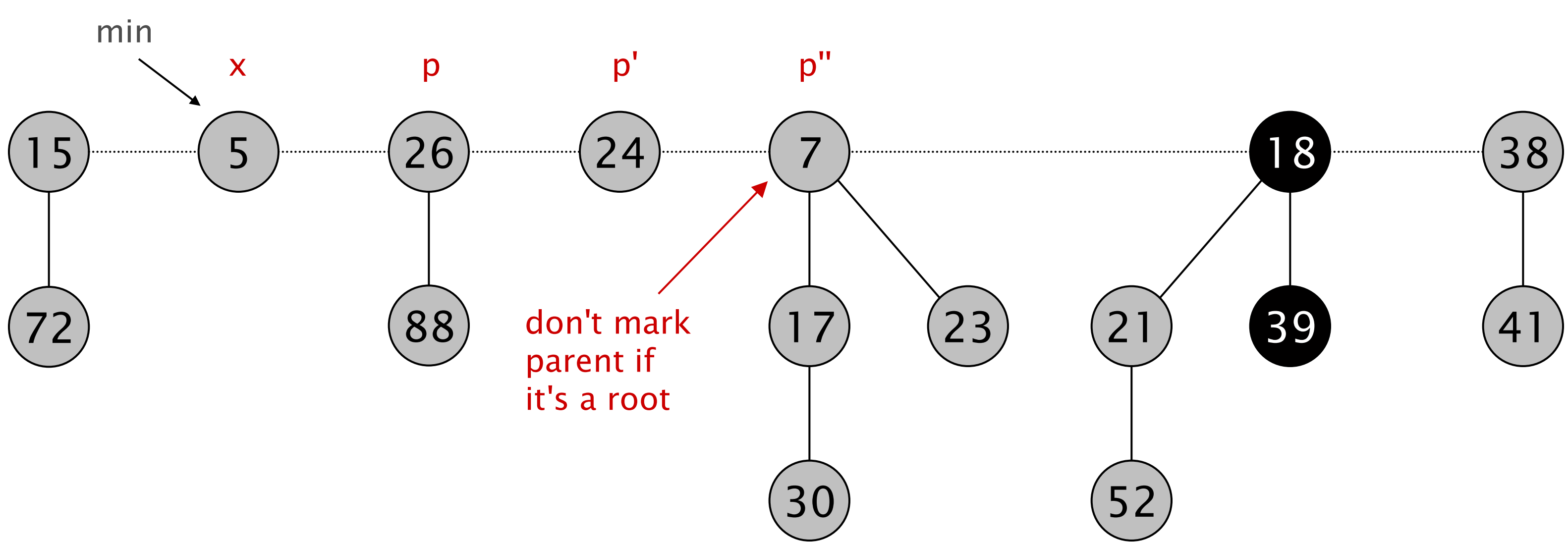


decrease-key of x from 35 to 5

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

Fibonacci Heaps: Decrease Key Analysis

Decrease-key.

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

potential function

Actual cost. O(c)

- O(1) time for changing the key.
- O(1) time for each of c cuts, plus melding into root list.

Change in potential. O(1) - c

- trees(H') = trees(H) + c.
- marks(H') $\leq$ marks(H) - c + 2.
- $\Delta\Phi \leq c + 2 \cdot (-c + 2) = 4 - c$.

Amortized cost. O(1)

Analysis

Analysis Summary

Insert. $O(1)$
Delete-min. $O(\text{rank}(H))$ †
Decrease-key. $O(1)$ †

† amortized

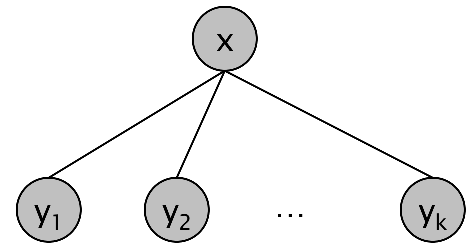
Key lemma. $\text{rank}(H) = O(\log n)$.

number of nodes is exponential in rank

Fibonacci Heaps: Bounding the Rank

Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

$$rank(y_i) \geq \begin{cases} 0 & \text{if } i = 1 \\ i - 2 & \text{if } i \geq 1 \end{cases}$$



Pf.

- When y_i was linked into x (during Consolidate), x had at least i -1 children $y_1, \ldots, y_{i-1}$.
- Since only trees of equal rank are linked, at that time $rank(y_i) = rank(x_i) \geq i - 1$.
- Since then, y_i has lost at most one child (with Cut).
- Thus, right now $rank(y_i) \geq i - 2$.

or y_i would have been cut

■

Why are they called Fibonacci heaps?

$$F_k = \begin{cases} 0 & \text{if } k = 0 \\ 1 & \text{if } k = 1 \\ F_{K-1} + F_{k-2} & \text{if } k \geq 2 \end{cases}$$

Lemma: For all k ≥ 0 $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

Lemma: For all k ≥ 0, the (k+2)nd Fibonacci number satisfies

$F_{k+2} \geq \phi^k$, where $\phi = (1 + \sqrt{5}) / 2 \approx 1.618$.

Lemma: let x be any node in a Fibonacci heap, and k = rank(x), and size(x) the number of nodes in the tree including x.

Then size(x) ≥ $F_{k+2} \geq \phi^k$

rank(n) ≥ size(x) ≥ ϕ^k

K <= $\log_\phi n$

Fibonacci Heaps: Bounding the Rank

Def. Let F_k be smallest possible tree of rank k