Fibonacci Heaps

Chiranjeev Kumar

IIT(ISM), Dhanbad

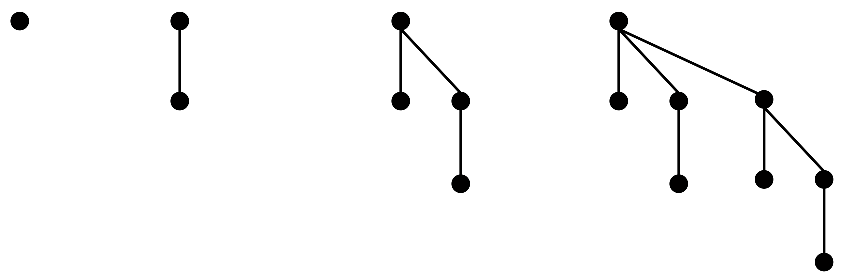
History. [Fredman and Tarjan, 1986]

- 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.

- Similar to binomial heaps, but less rigid structure.
- Binomial heap: eagerly consolidate trees after each `insert`.

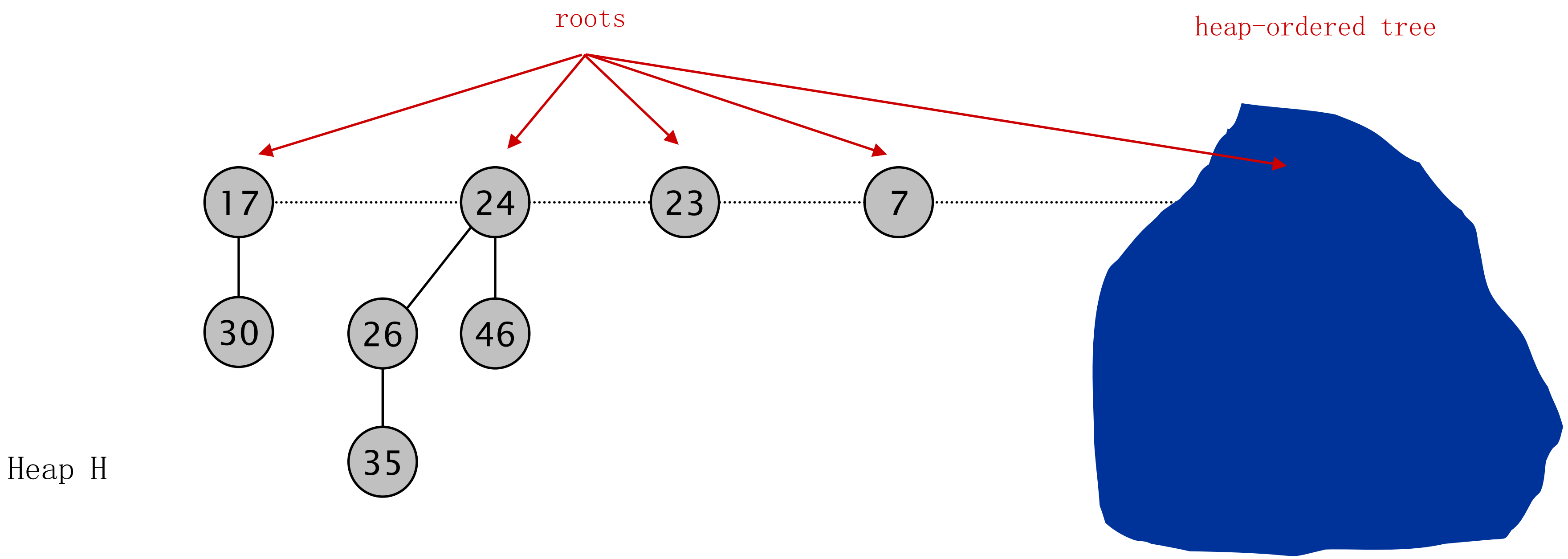


- Fibonacci heap: lazily defer consolidation until next `delete-min`.

Fibonacci Heaps: Structure

each parent smaller than its children

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

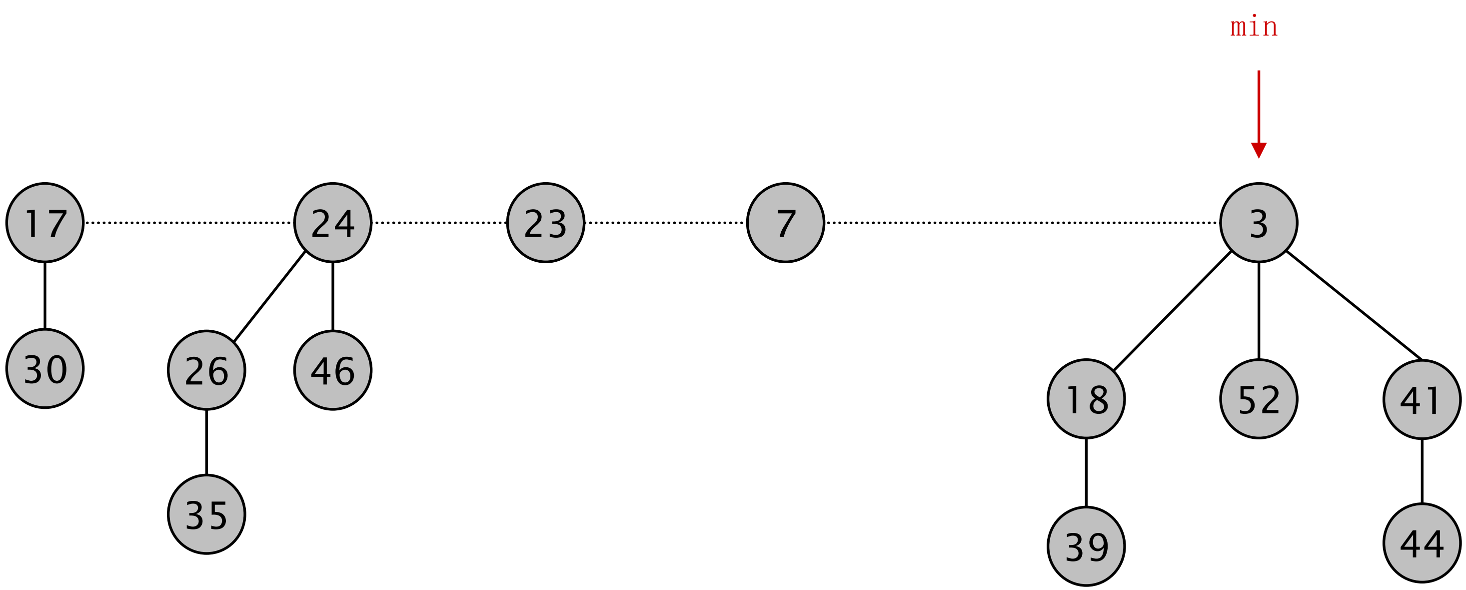


Fibonacci Heaps: Structure

Fibonacci heap.

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

find-min takes O(1) time



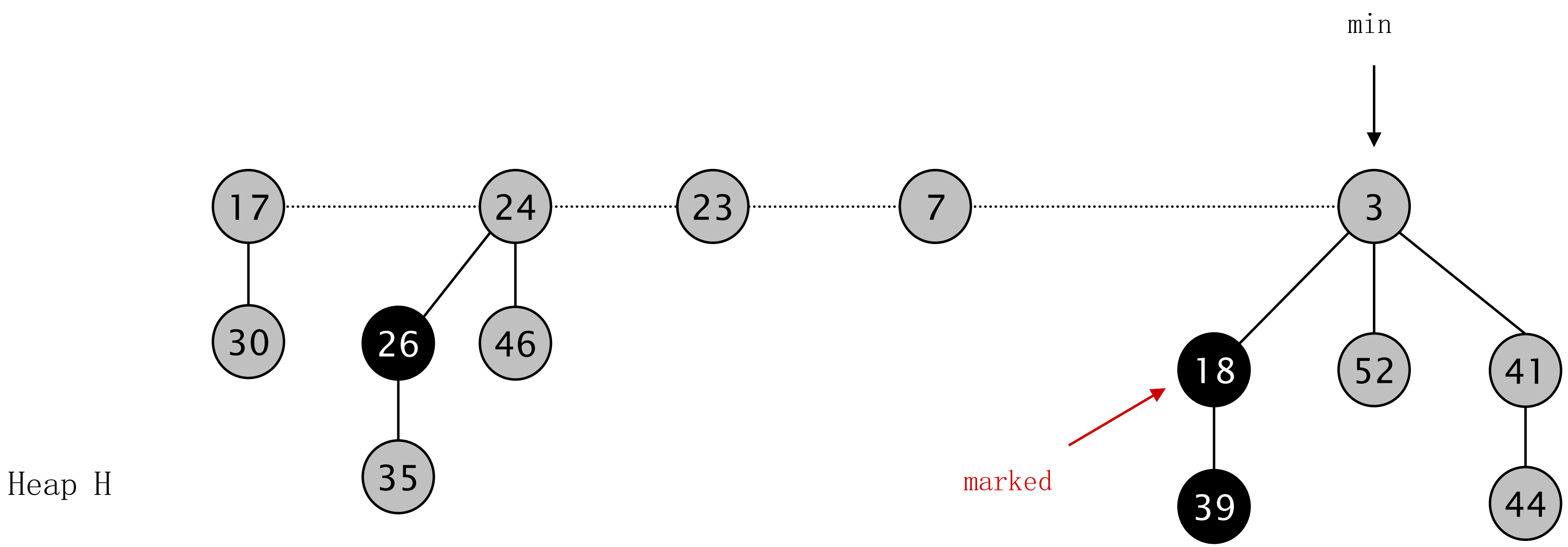
Heap H

Fibonacci Heaps: Structure

Fibonacci heap.

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

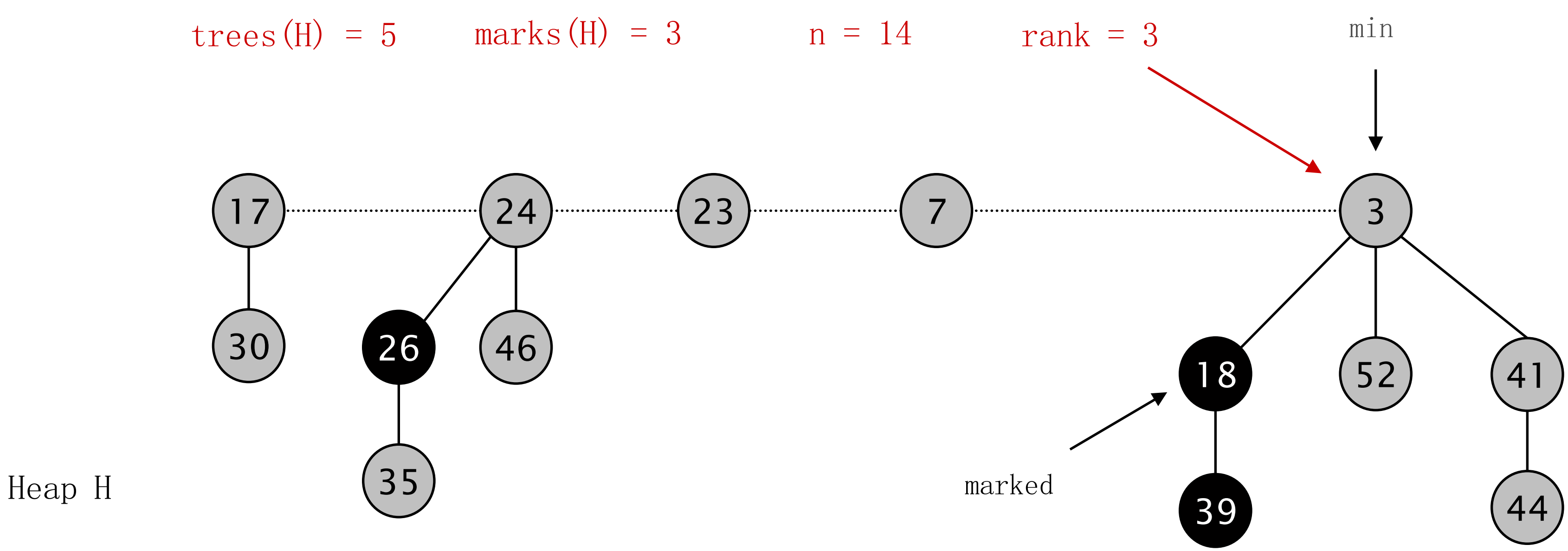
use to keep heaps flat (stay tuned)



Fibonacci Heaps: Notation

Notation.

- n = number of nodes in heap.
- degree(x) or 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.

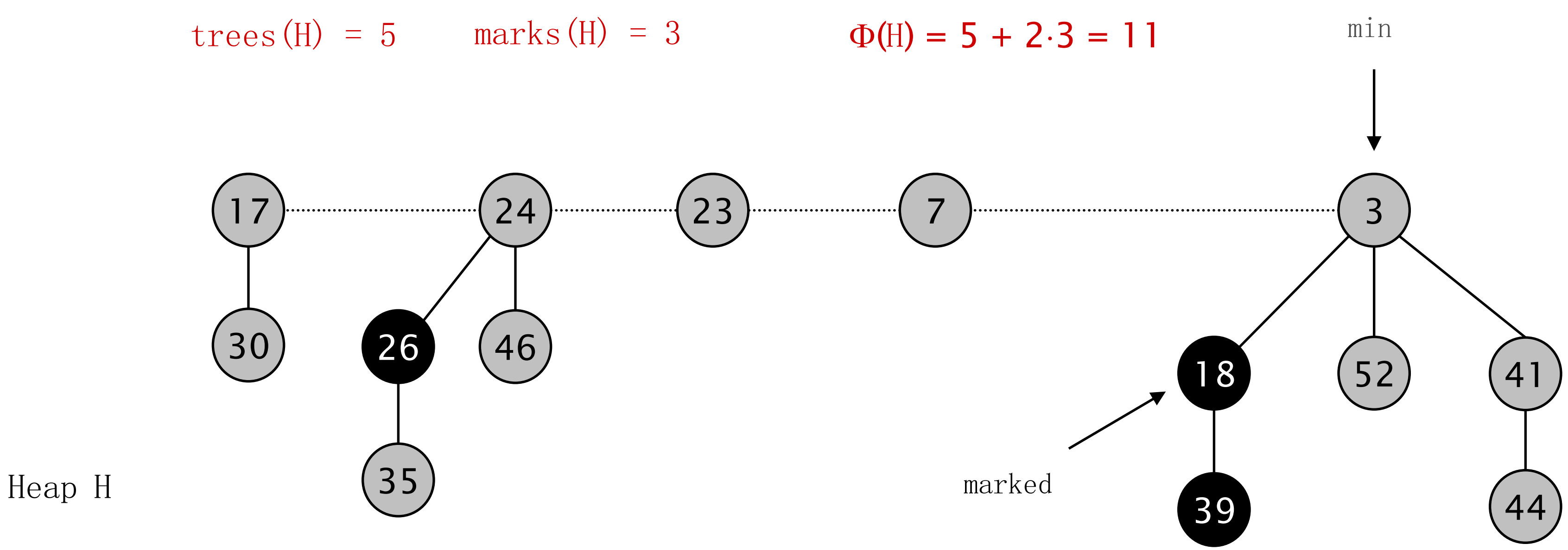


Fibonacci Heaps: Potential Function

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

potential of heap H

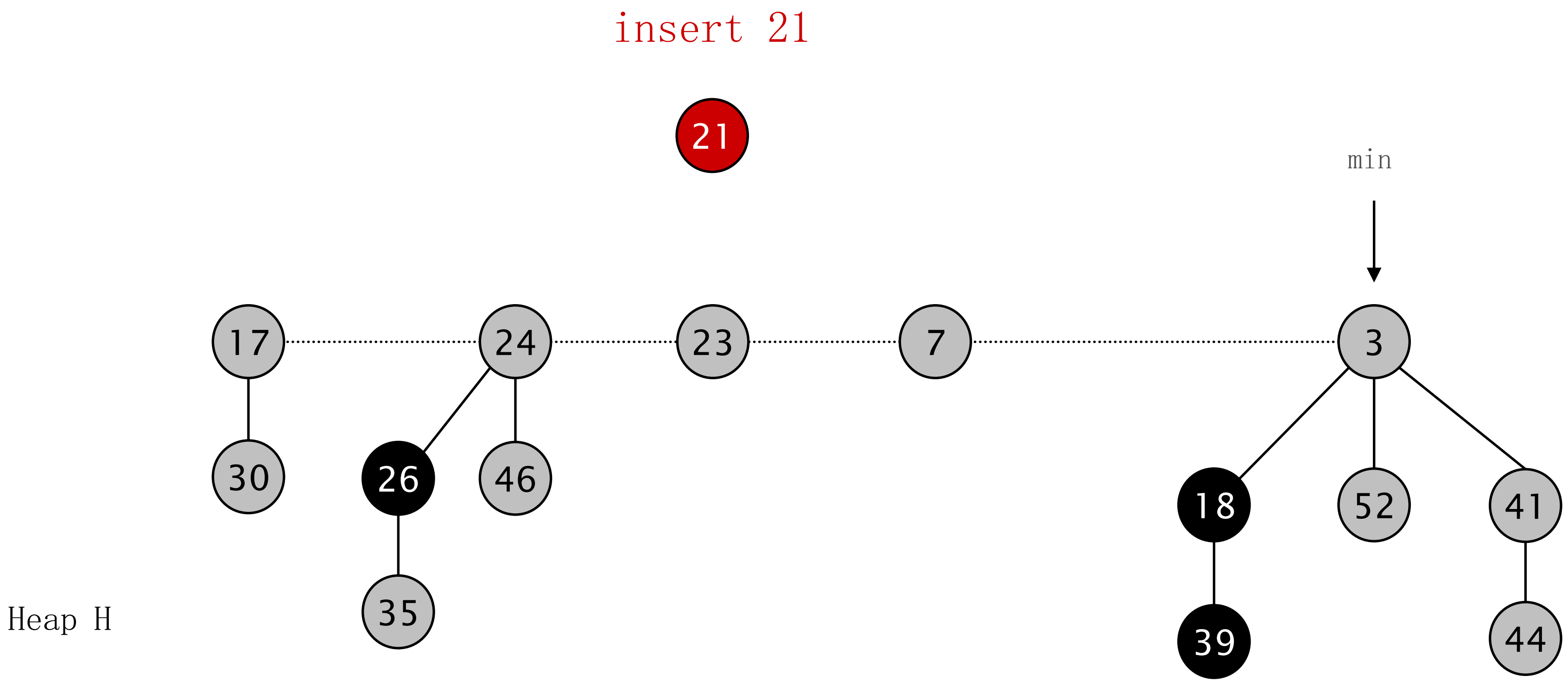
trees(H) = 5   marks(H) = 3 $\Phi(H) = 5 + 2 \cdot 3 = 11$



Insert

Insert.

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

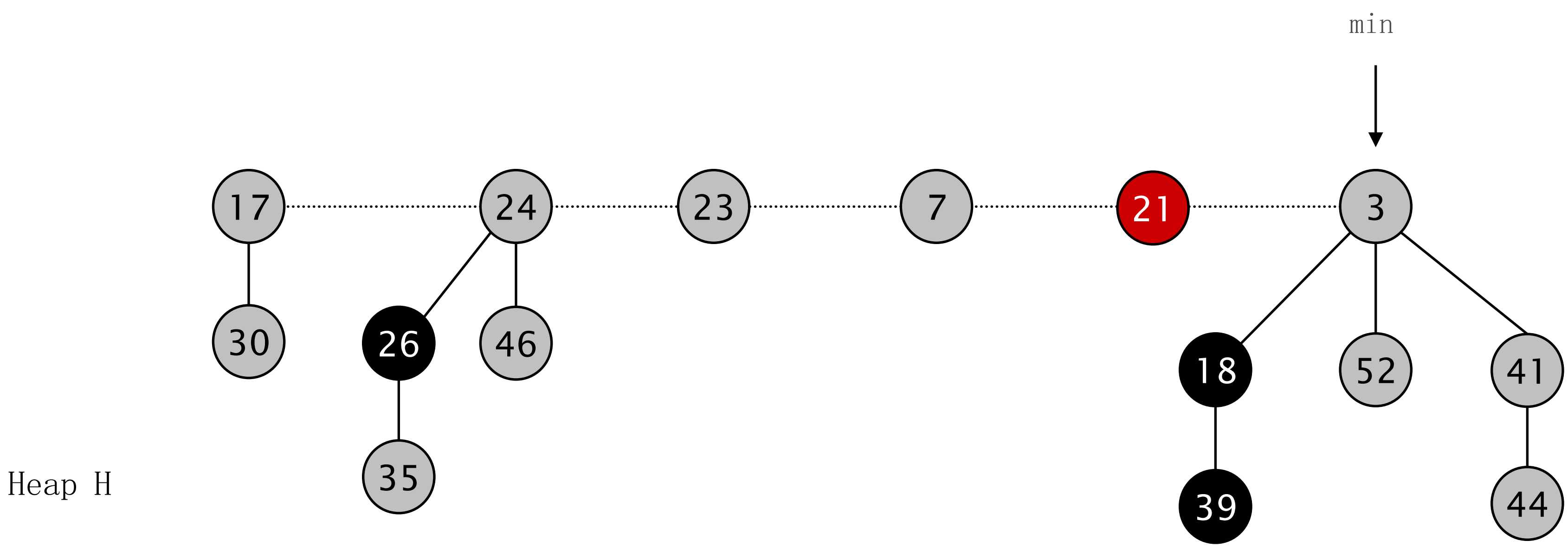


Fibonacci Heaps: Insert

Insert.

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

insert 21



Fibonacci Heaps: Insert Analysis

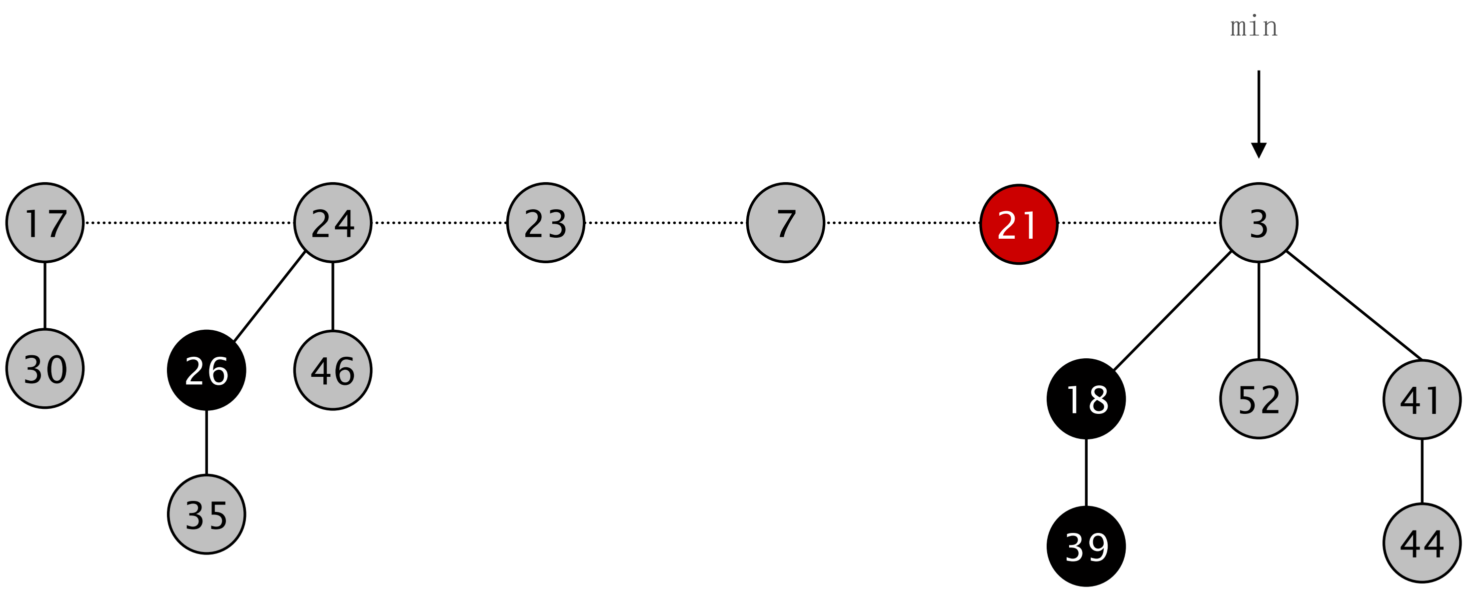
Actual cost. $O(1)$

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

potential of heap H

Change in potential. +1

Amortized cost. $O(1)$

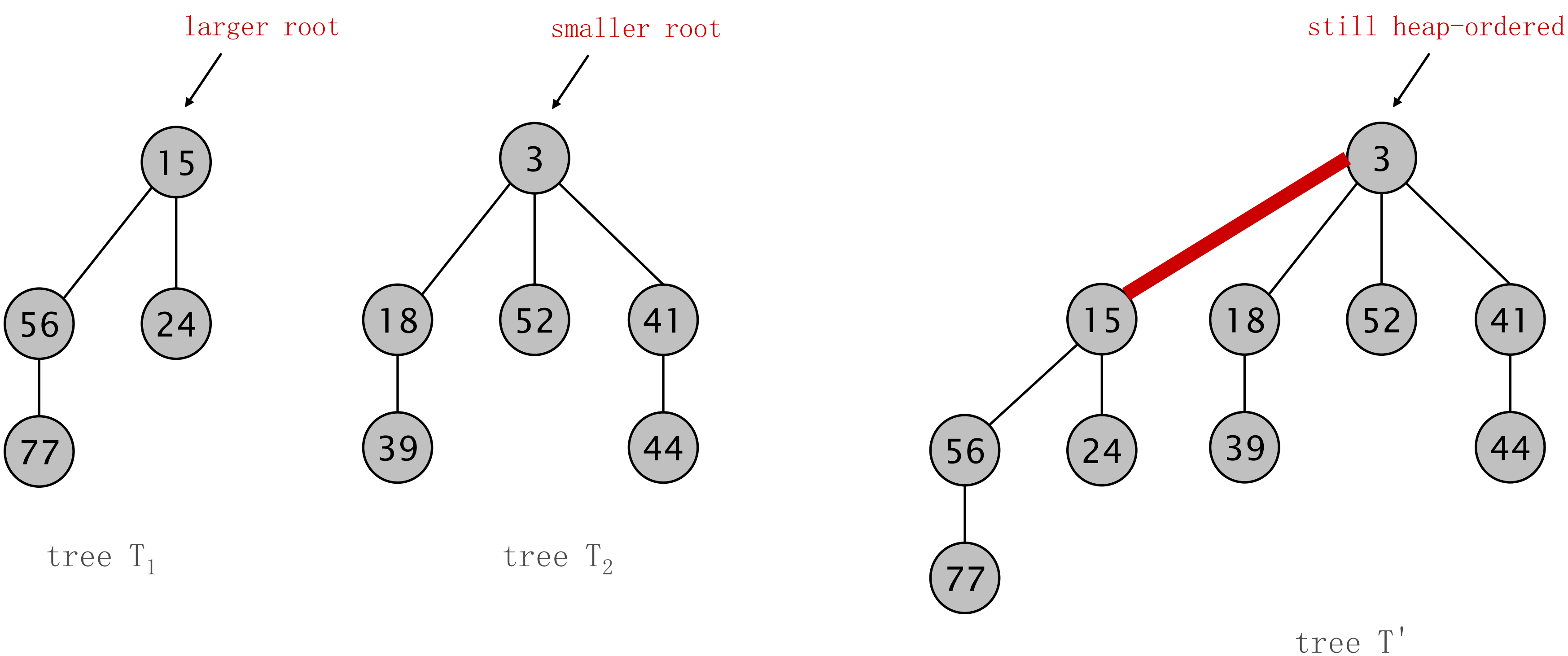


Heap H

Delete Min

Linking Operation

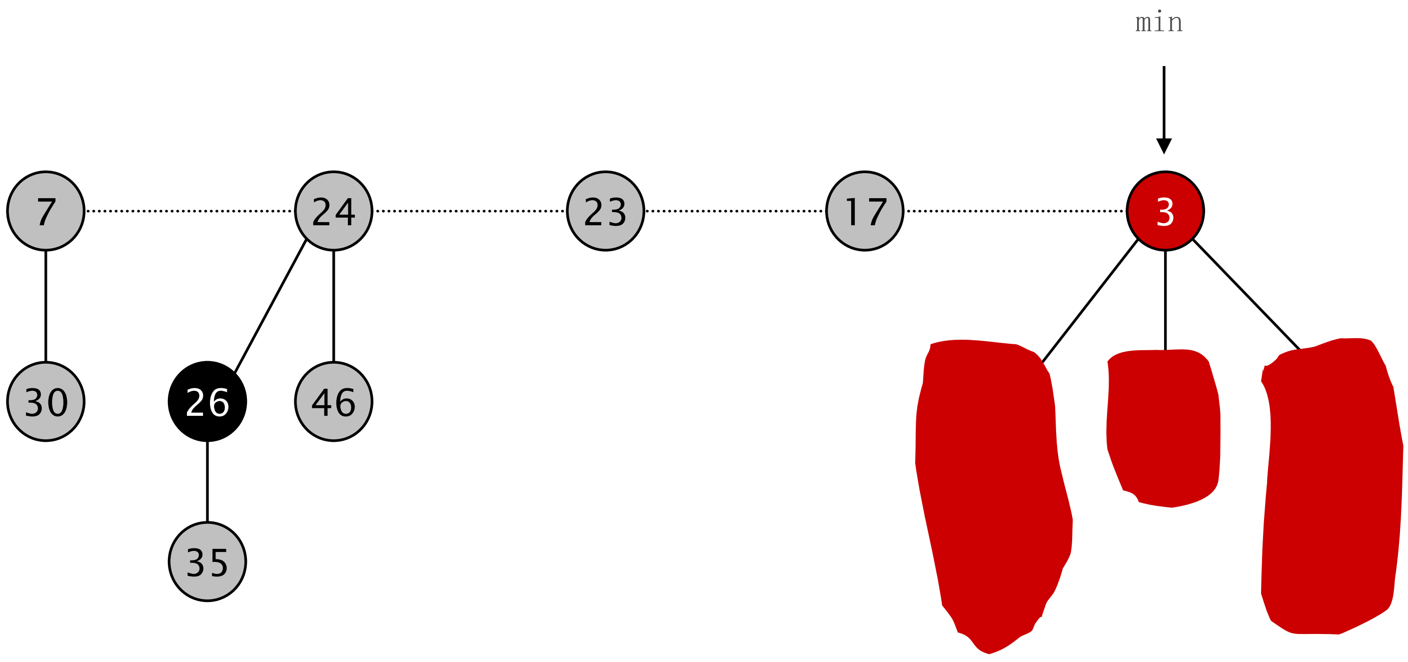
Linking operation. Make larger root be a child of smaller root.



Fibonacci Heaps: Delete Min

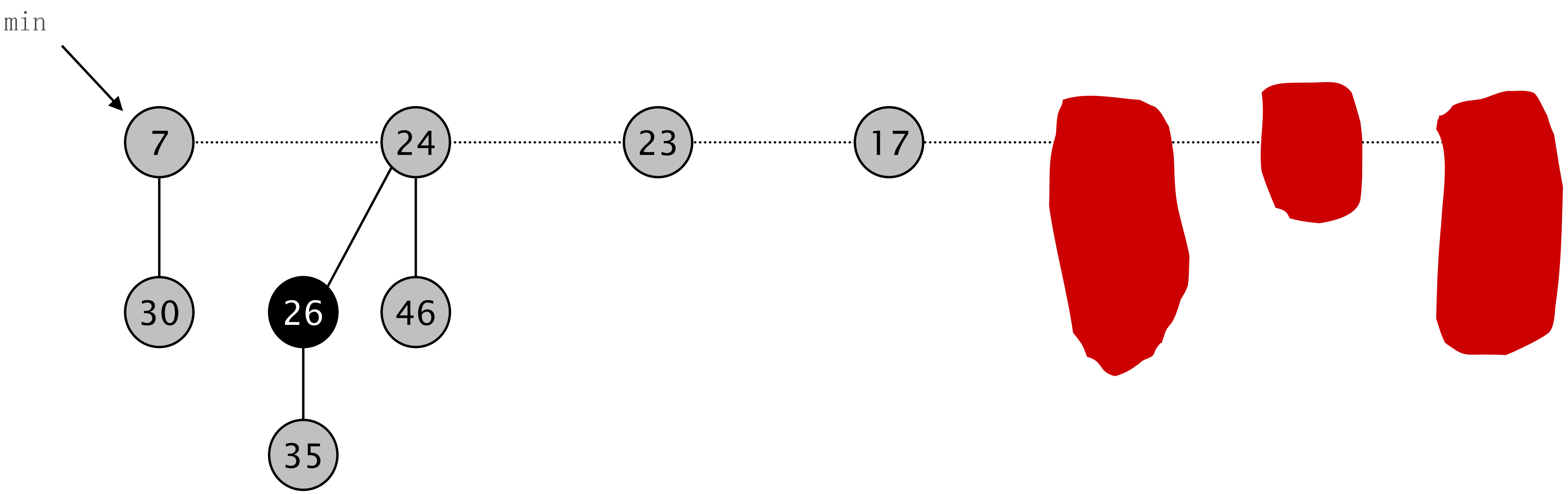
Delete min.

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



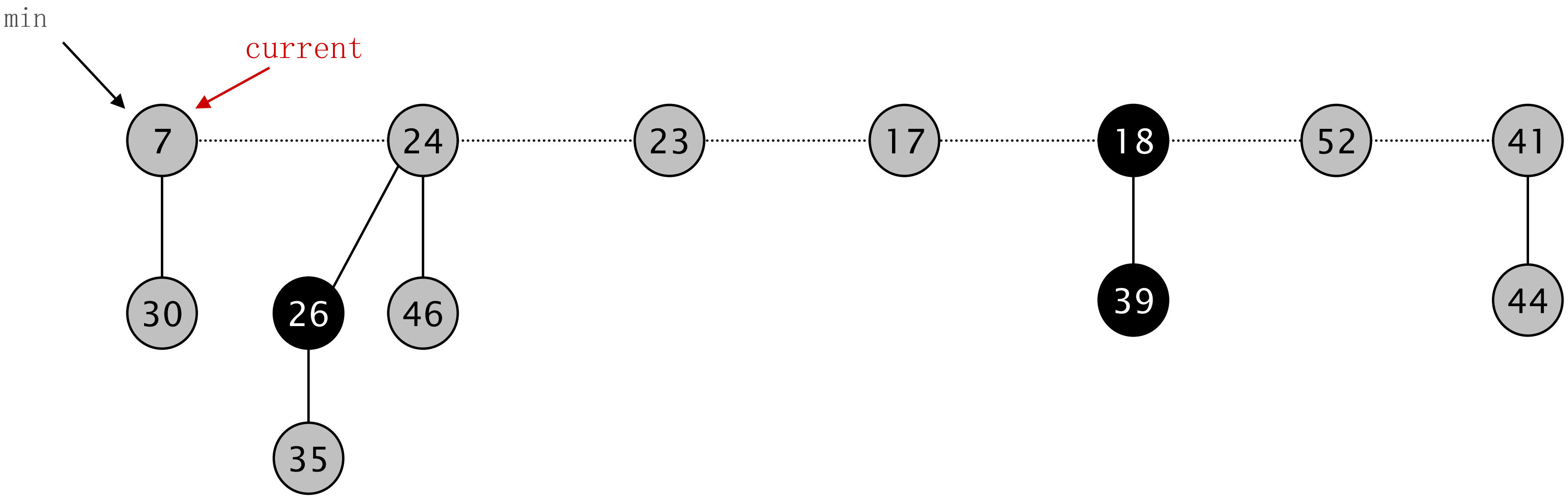
Delete min.

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



Delete min.

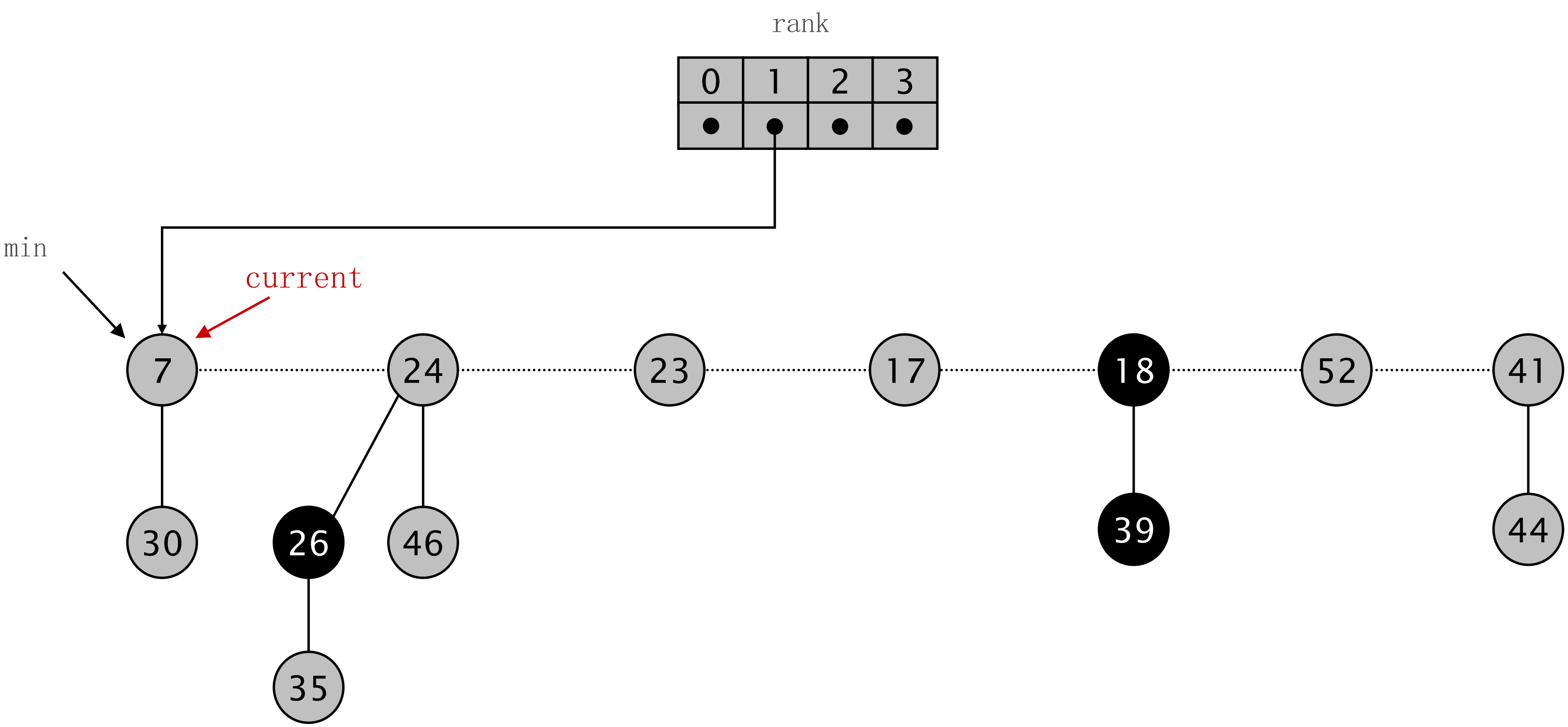
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

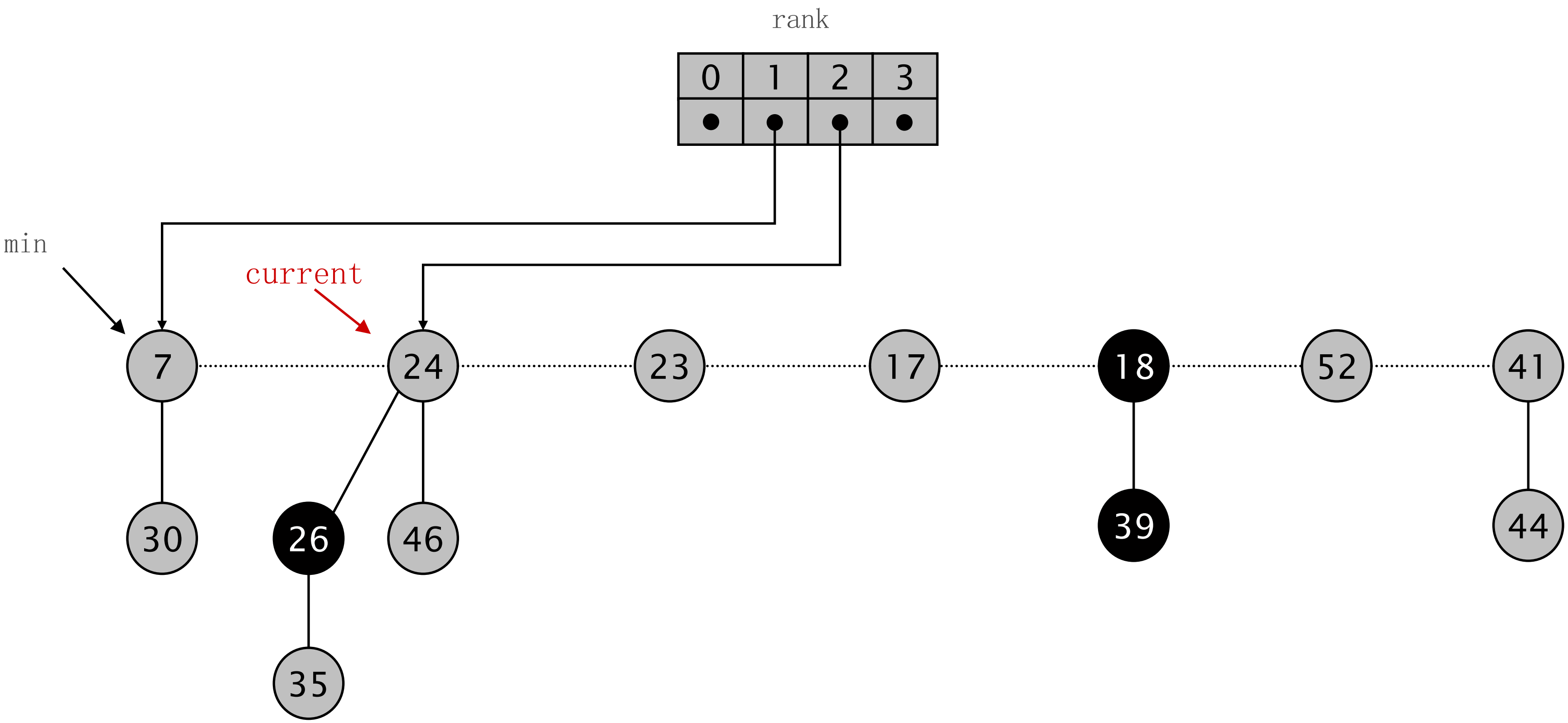
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

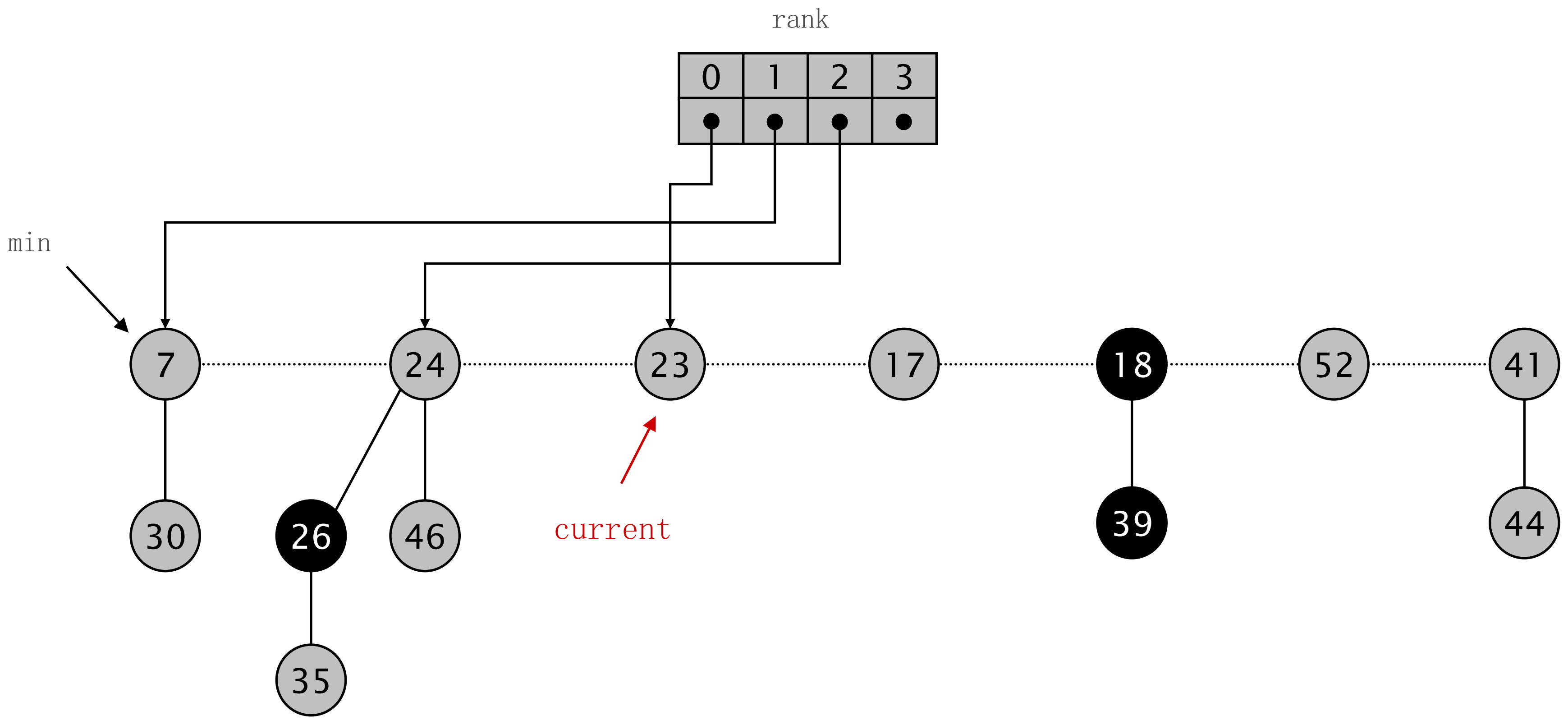
Delete min.

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



Delete min.

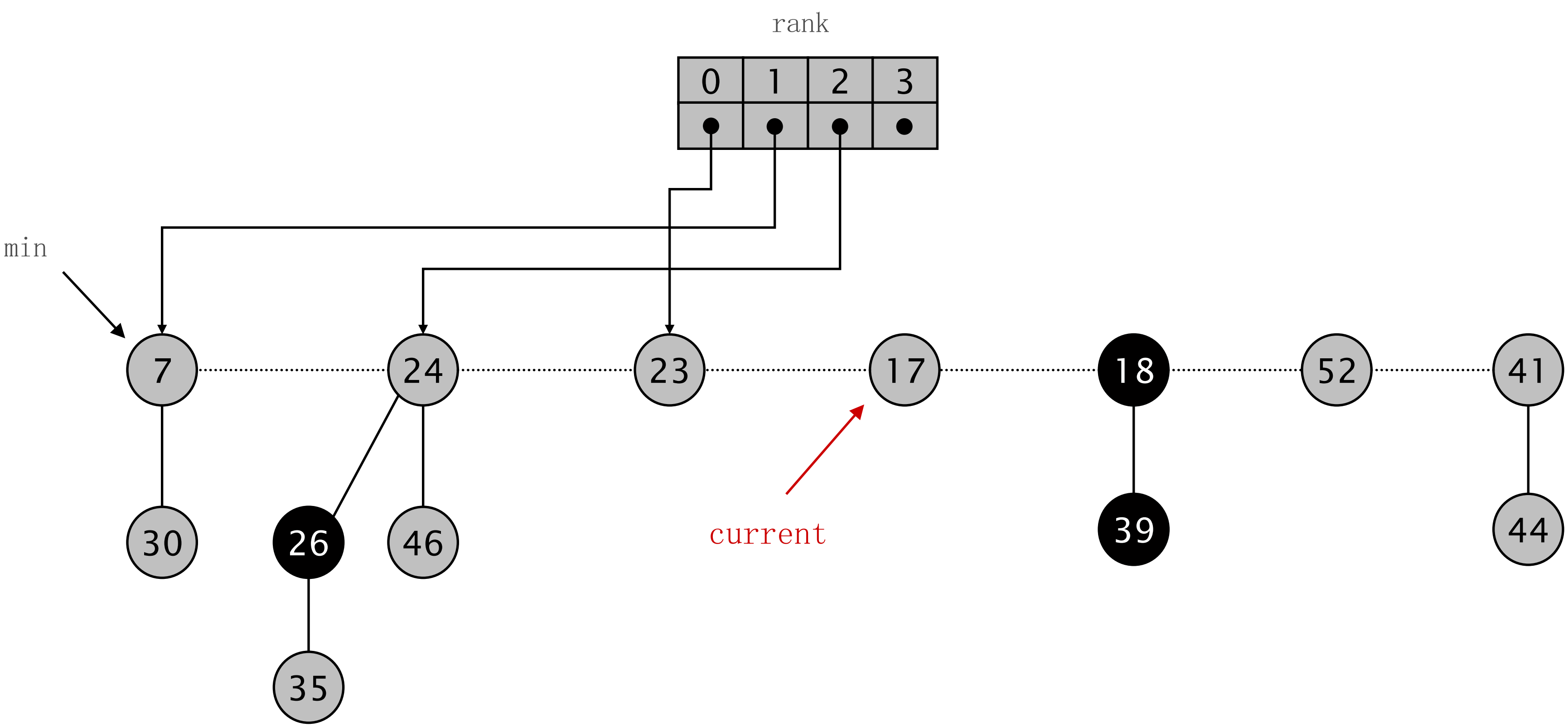
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



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.

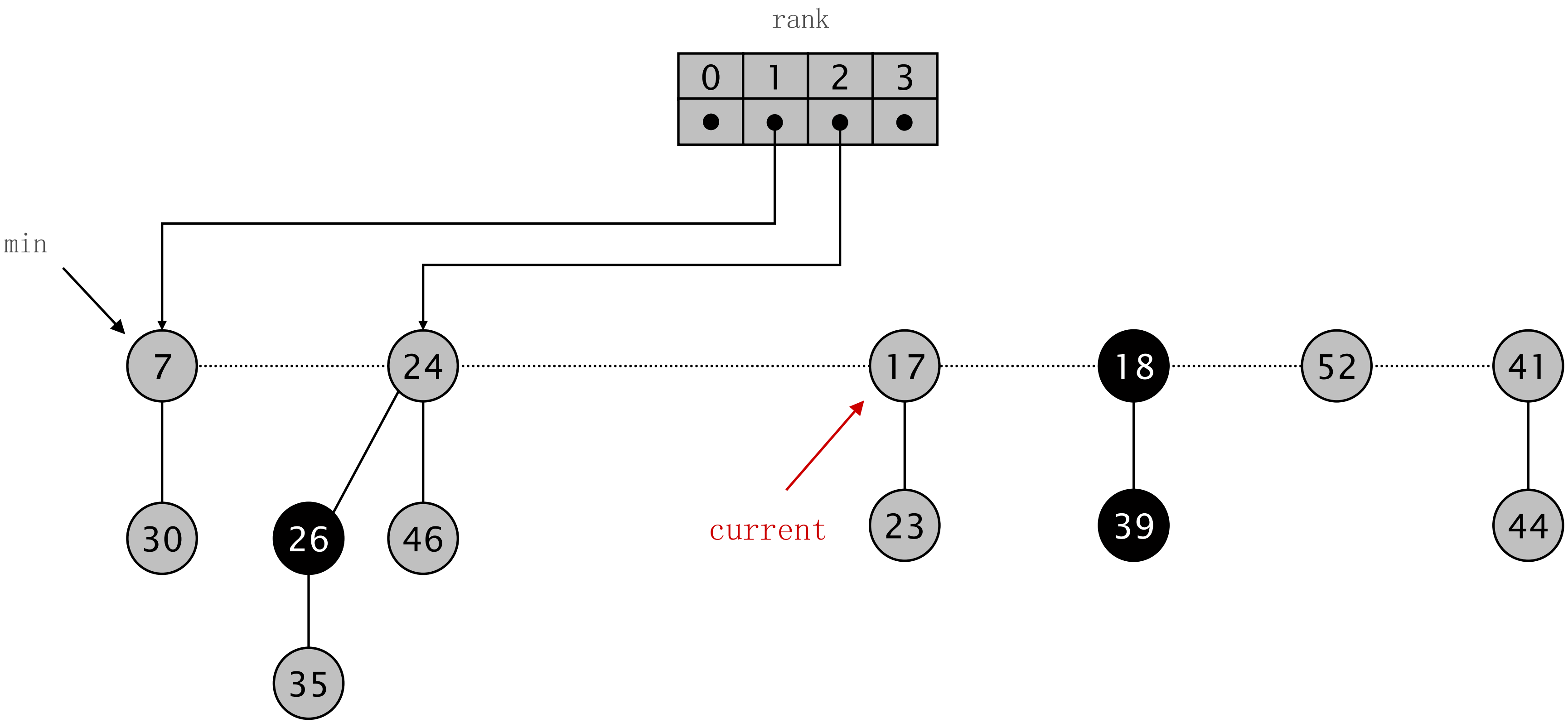


link 23 into 17

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.

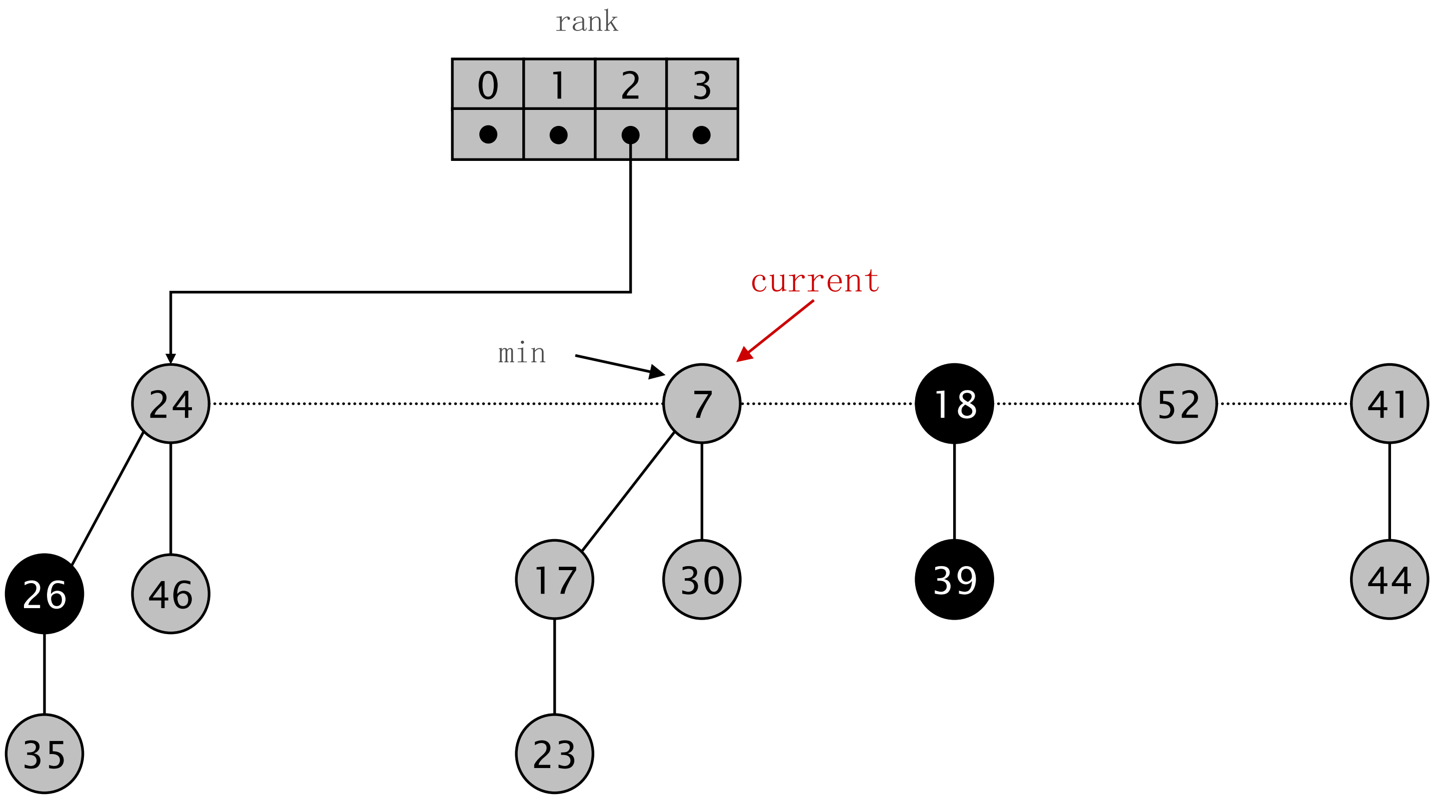


link 17 into 7

Fibonacci Heaps: Delete Min

Delete min.

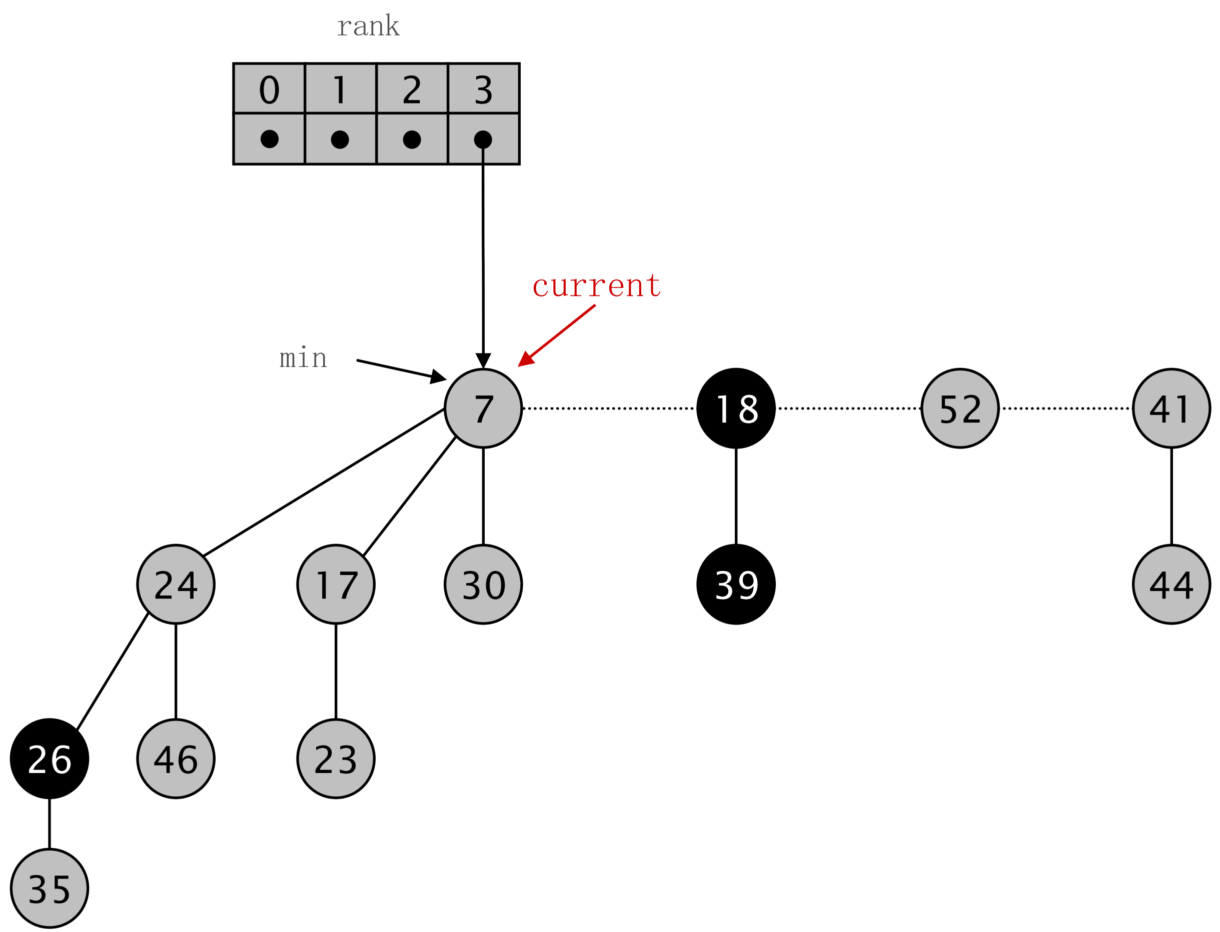
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



link 24 into 7

Delete min.

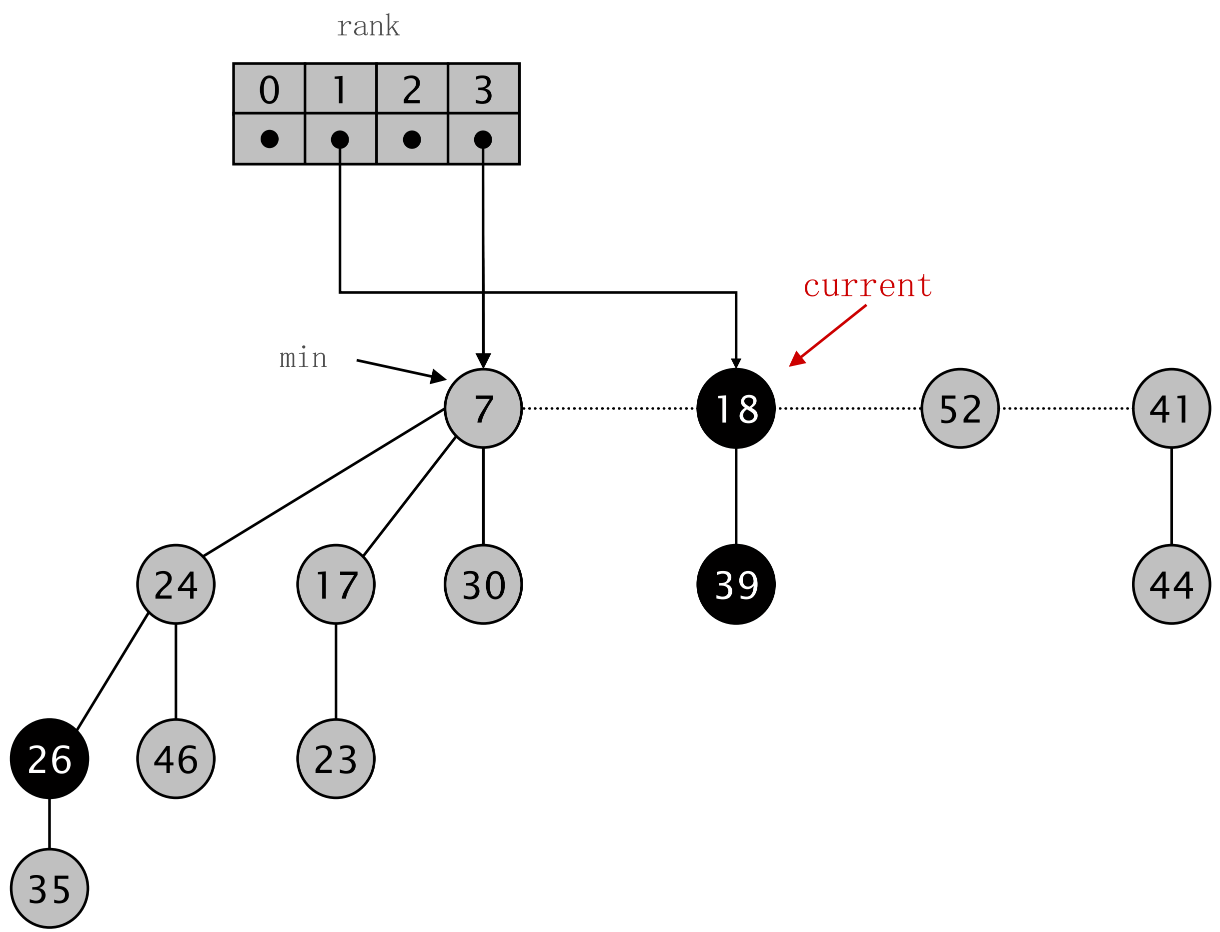
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

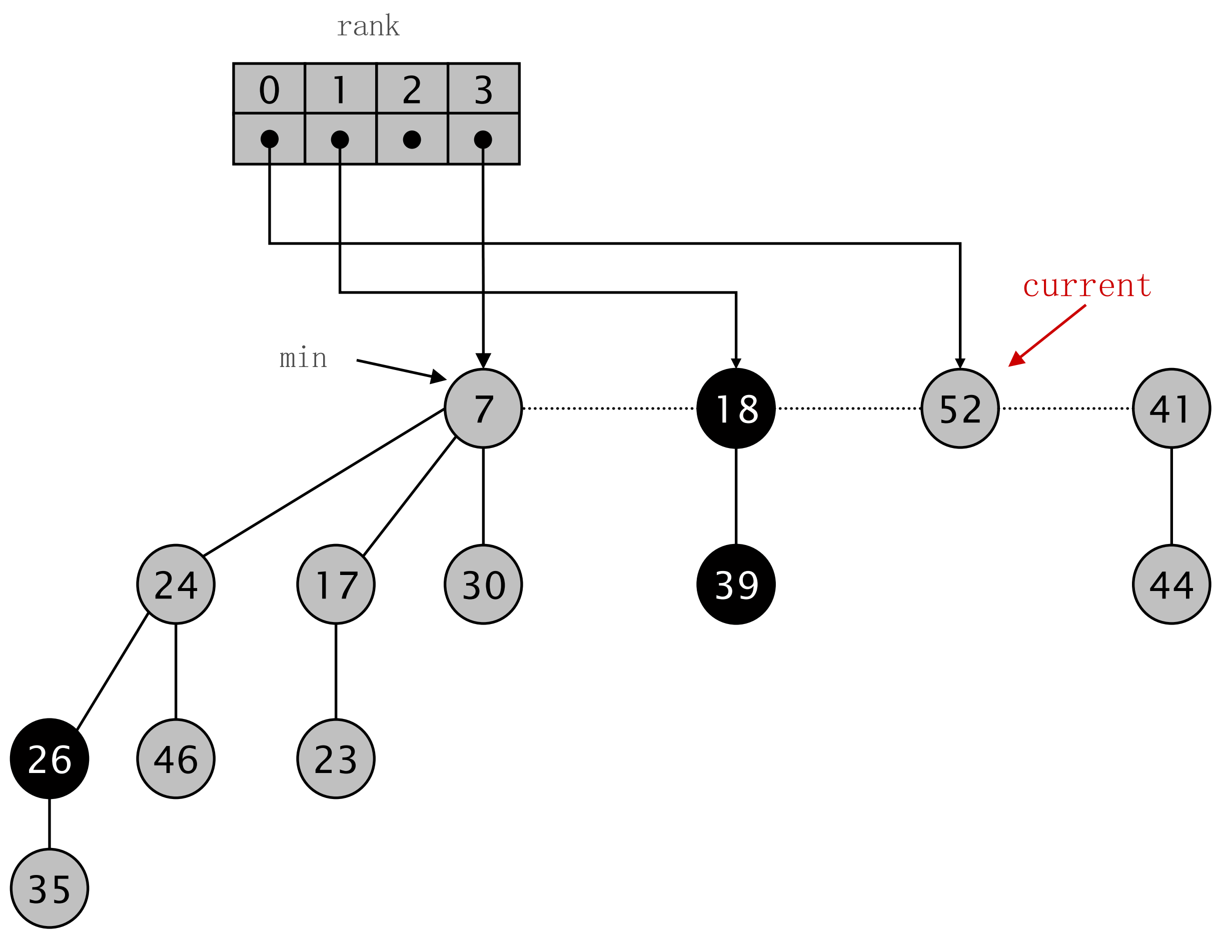
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

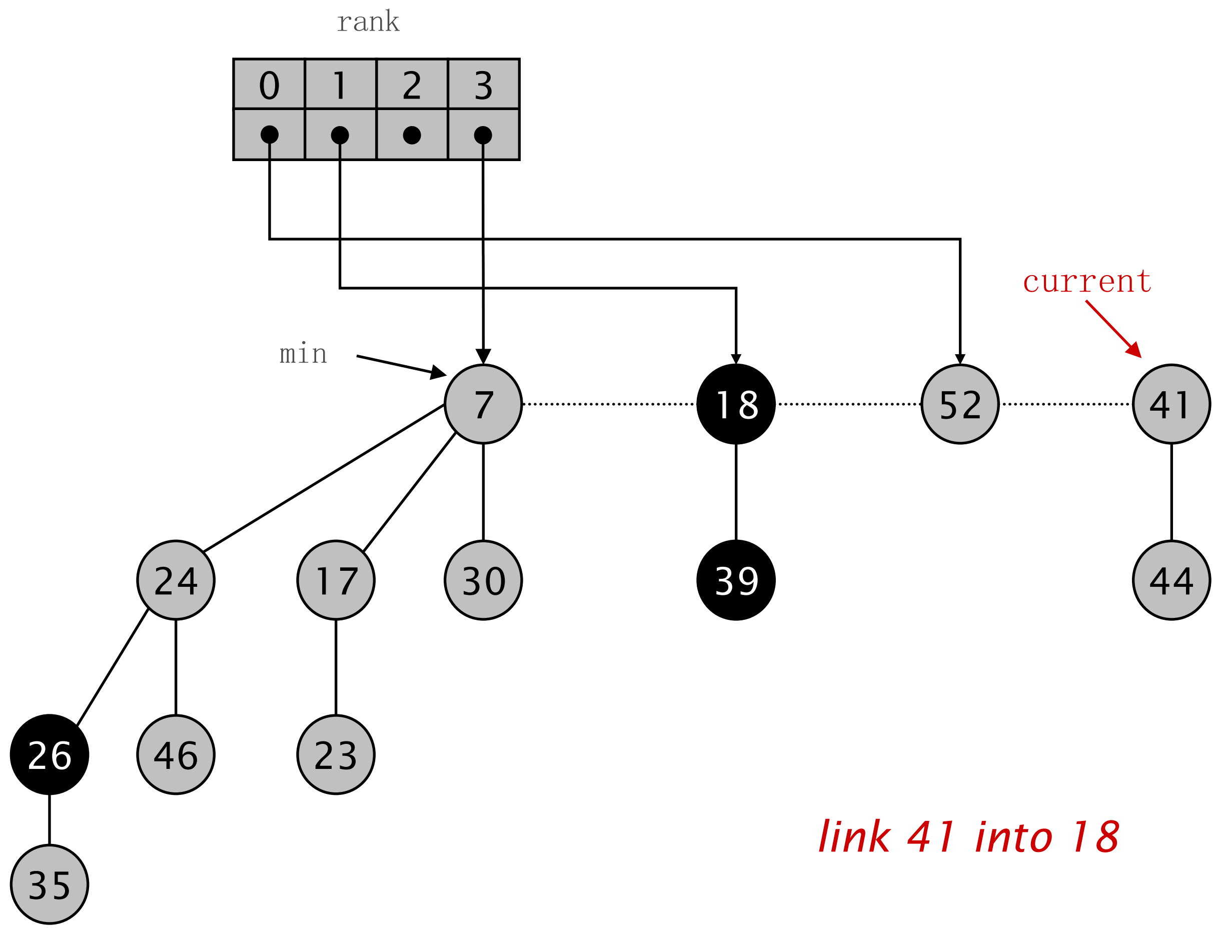
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

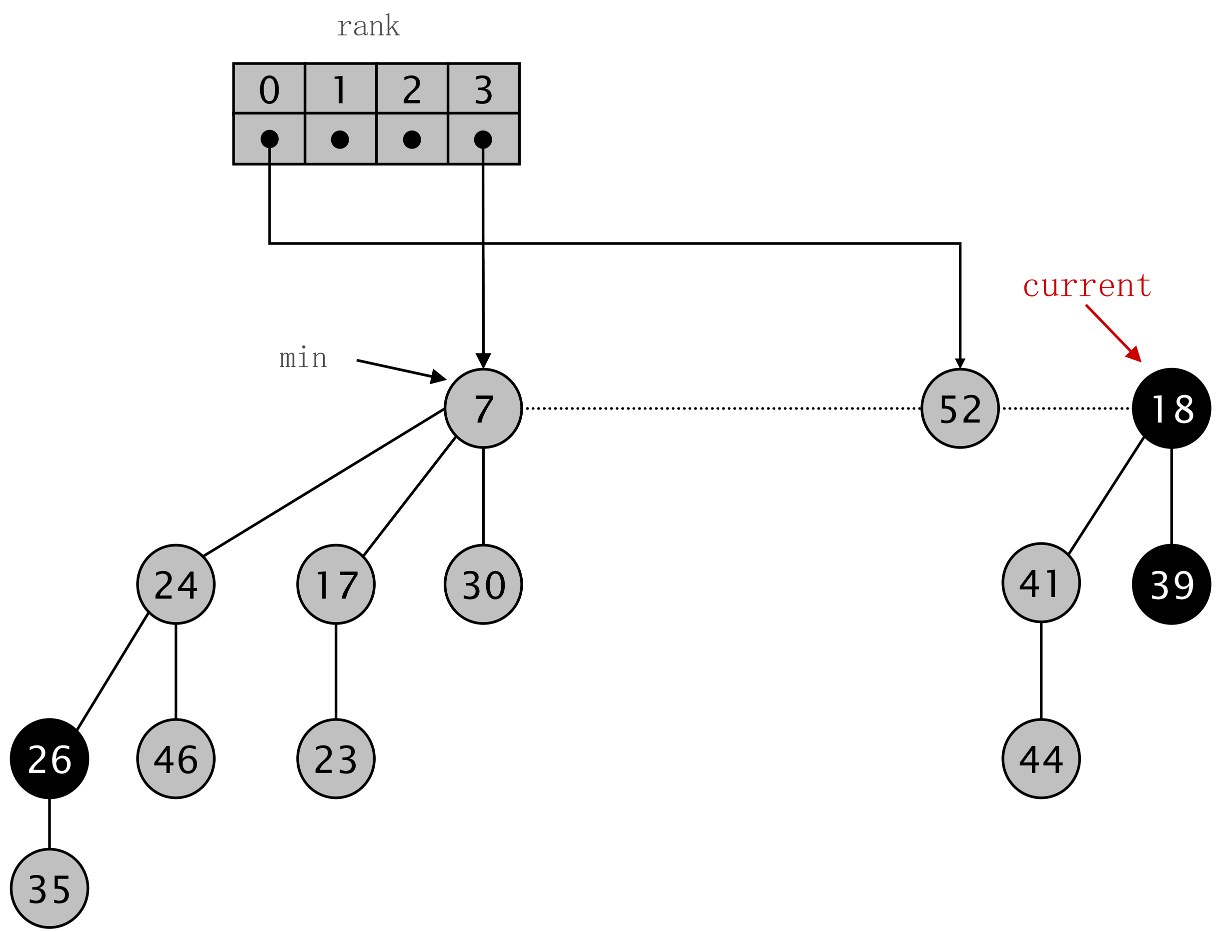
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

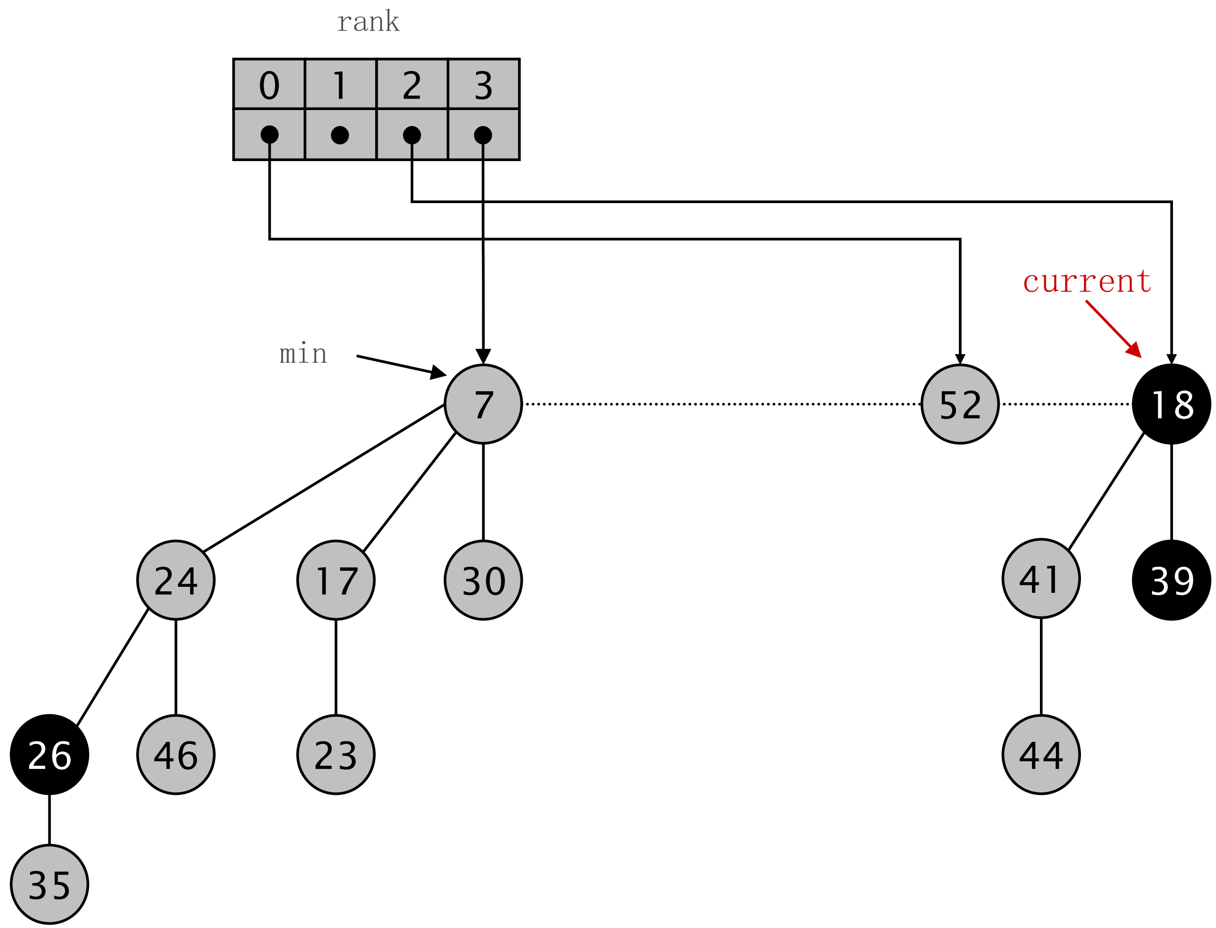
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

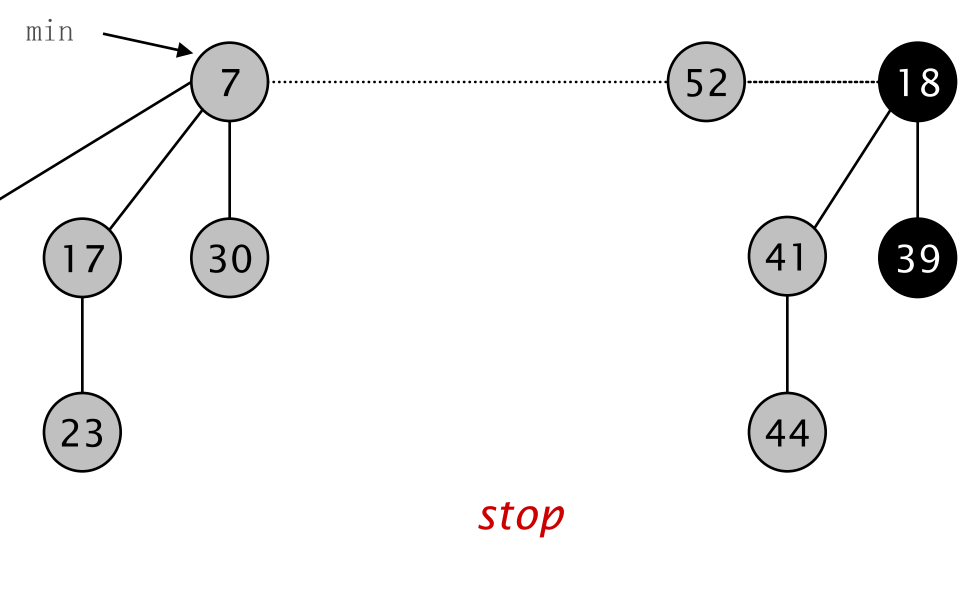
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



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.

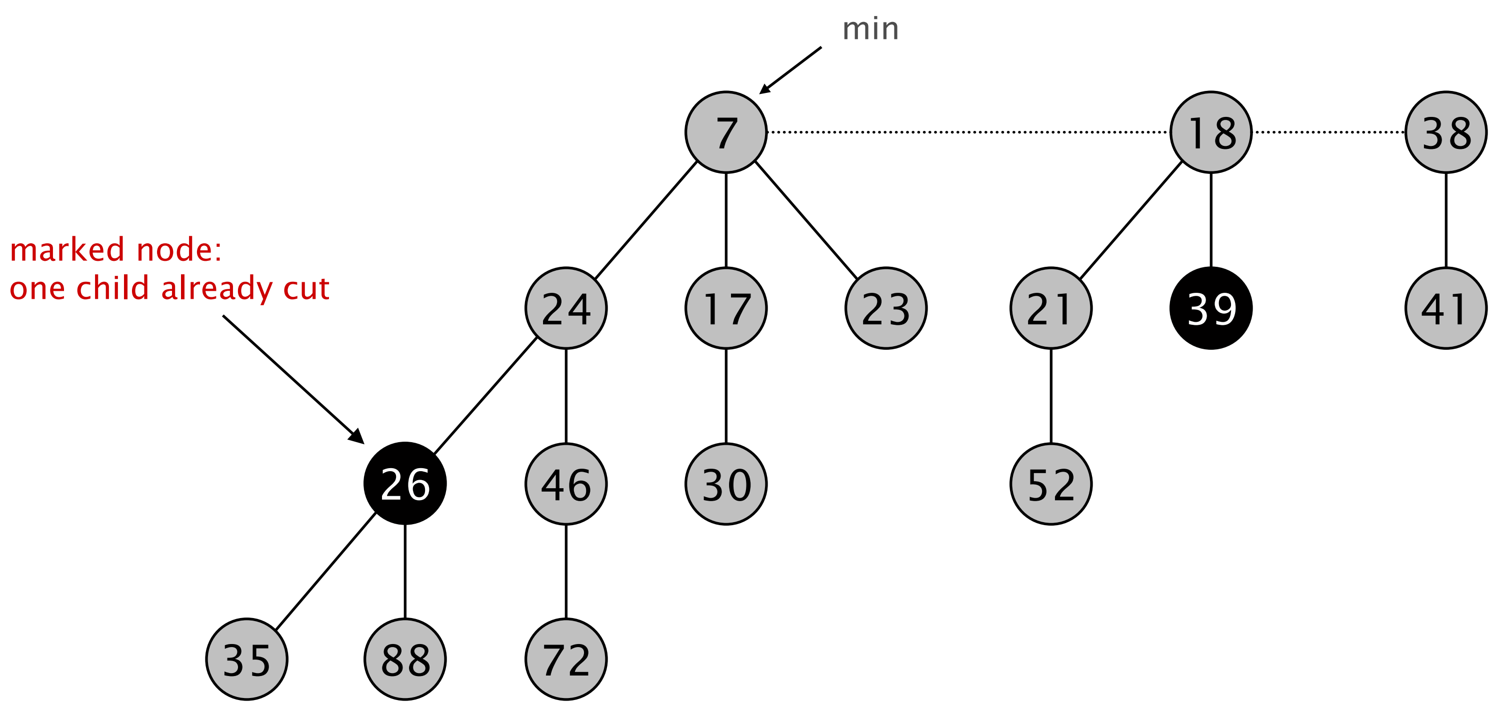


Decrease Key

Fibonacci Heaps: Decrease Key

Intuition for deceasing the key of node x.

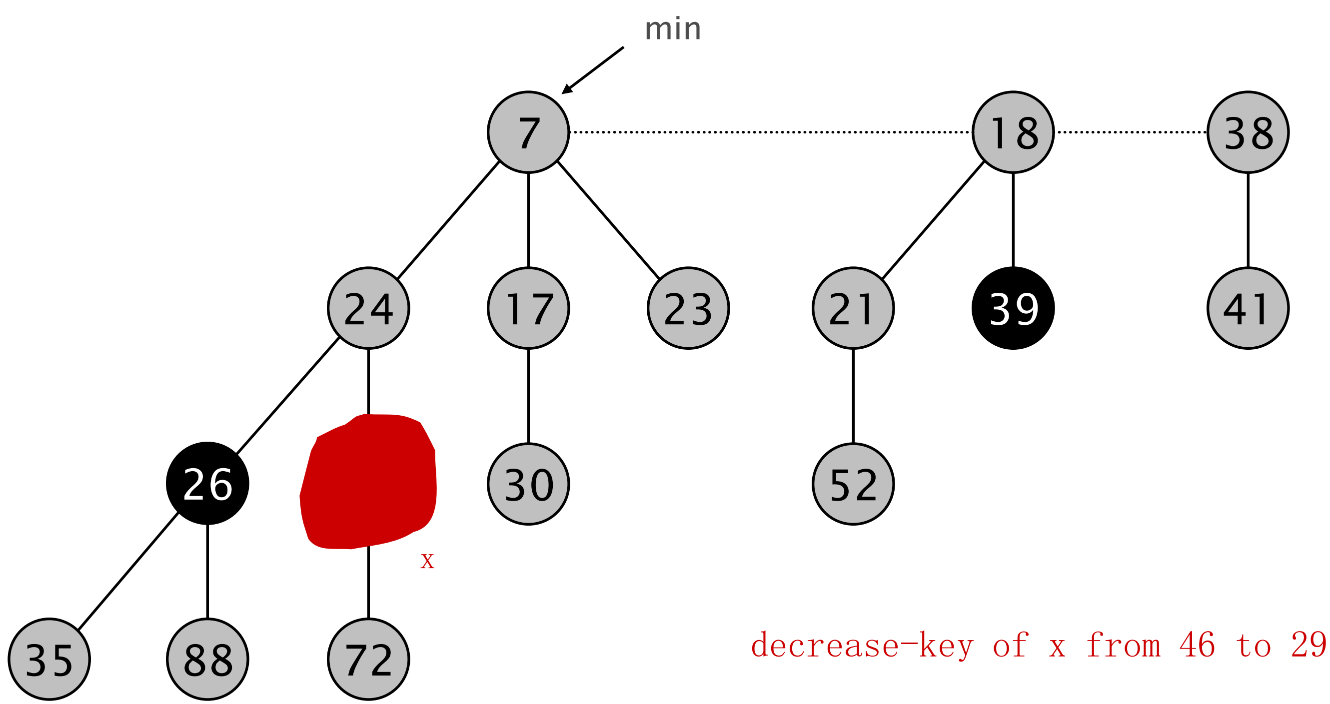
- If heap-order is not violated, just decrease the key of x.
- Otherwise, cut tree rooted at x and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

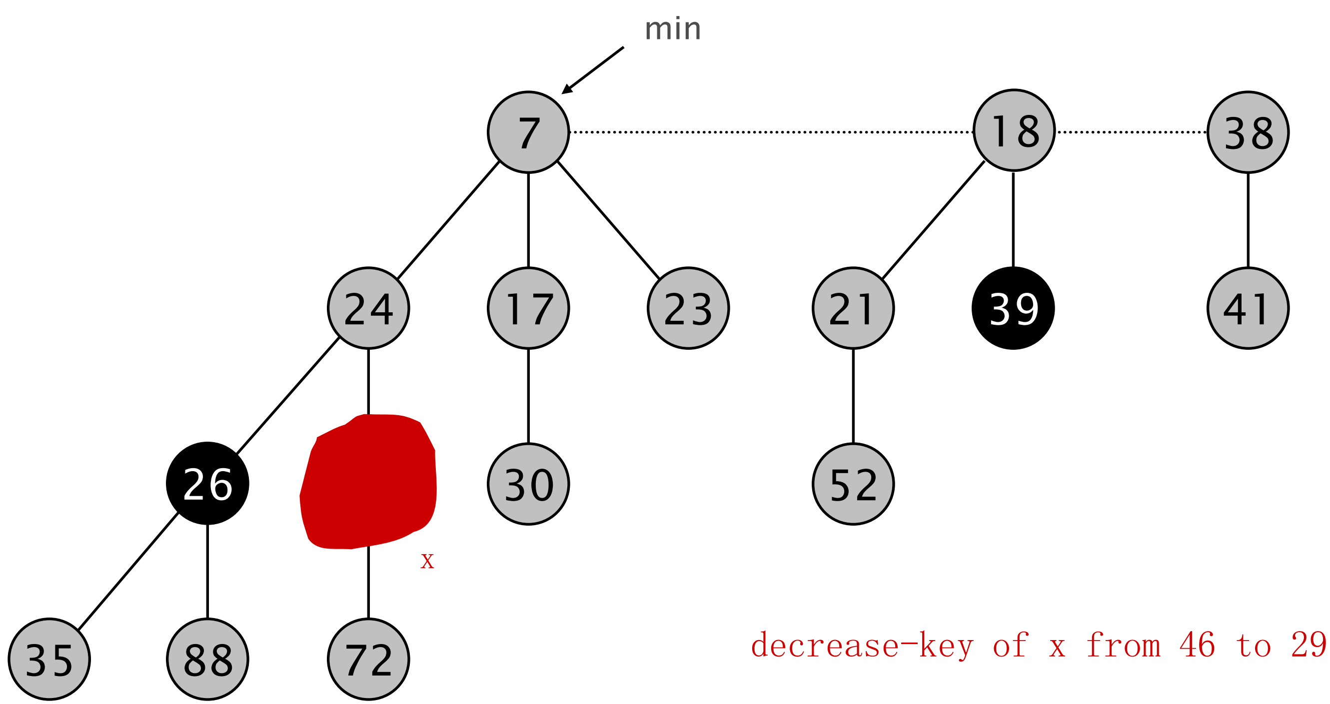
- Decrease key of x.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

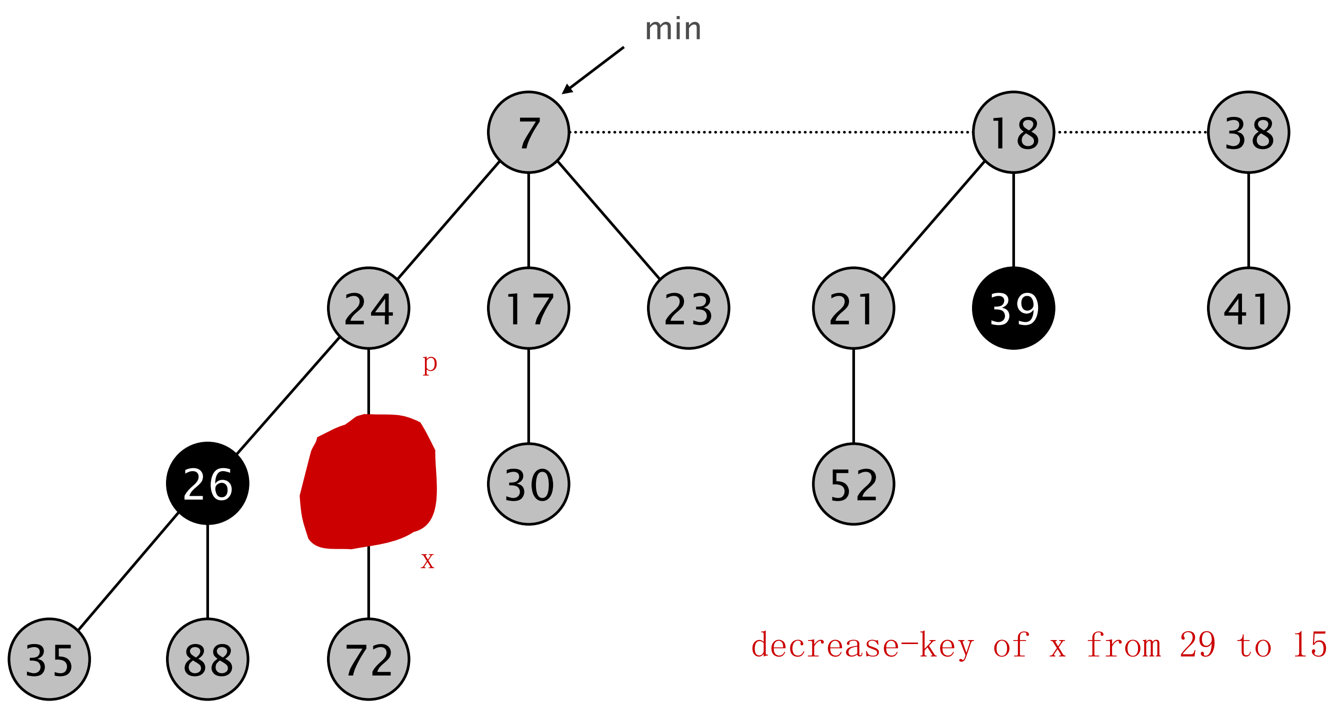
- Decrease key of x.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

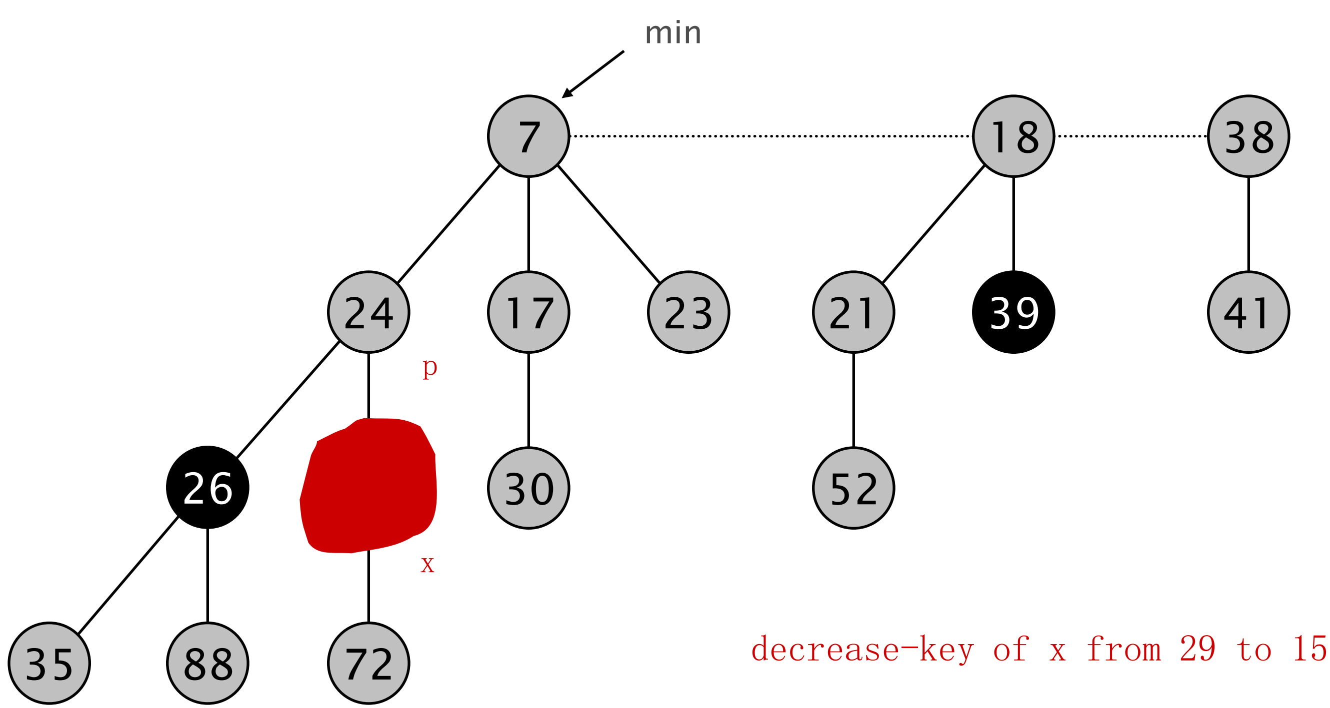
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

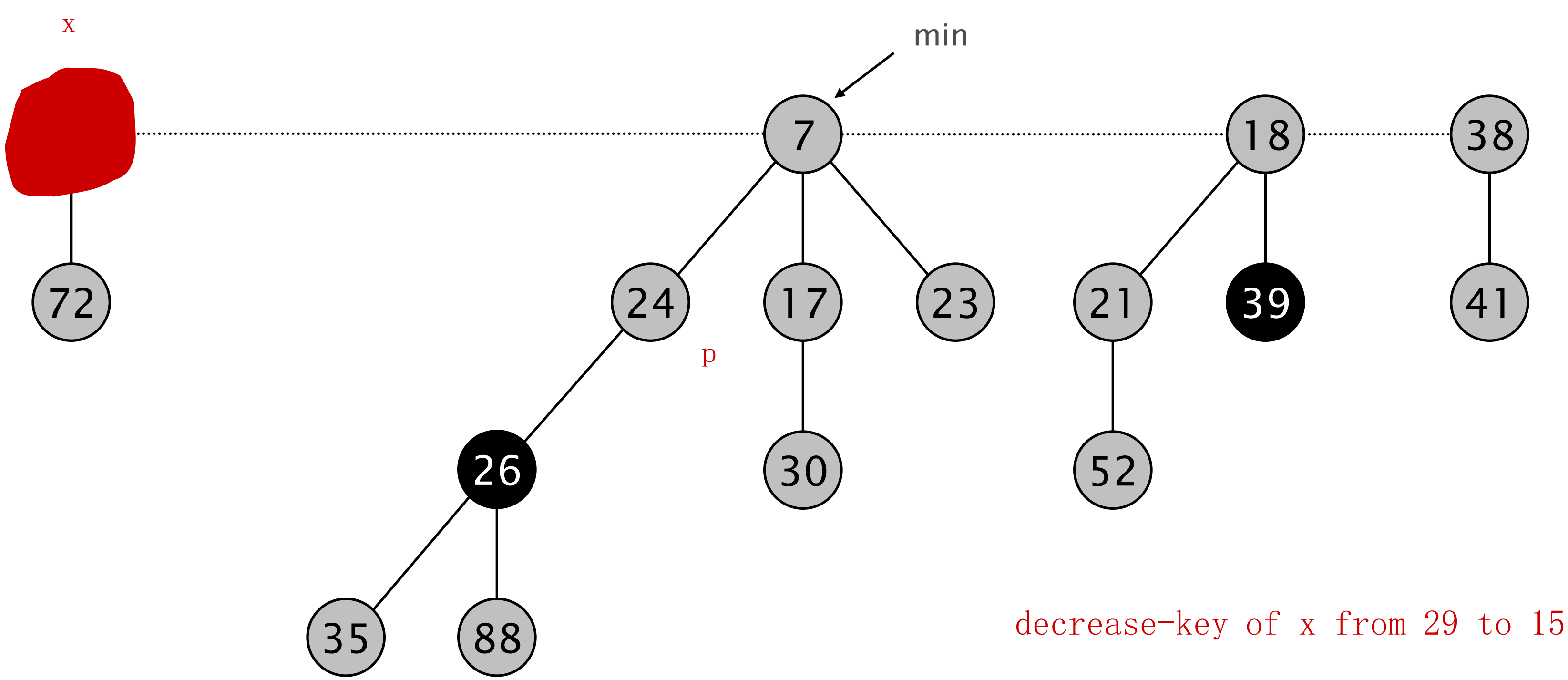
Case 2a. [heap order violated]

- 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 2a. [heap order violated]

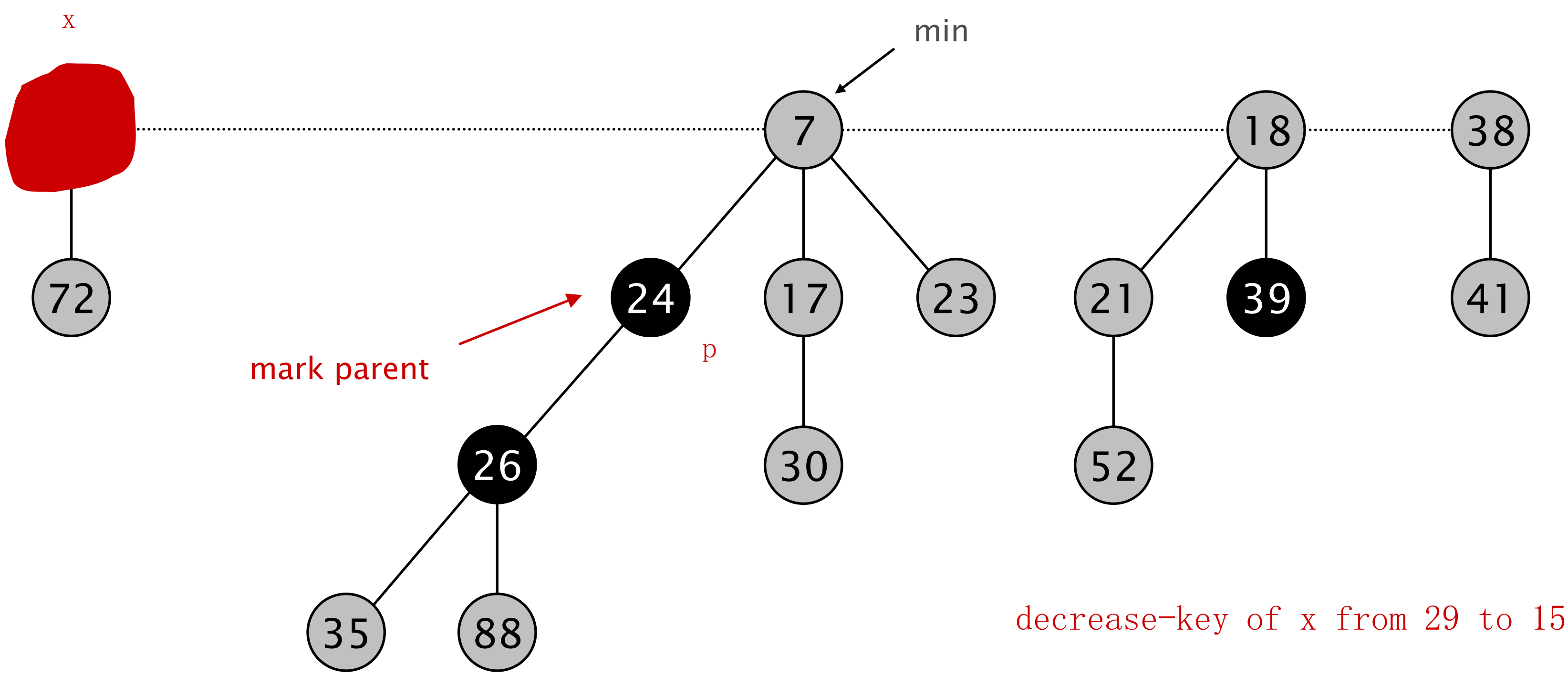
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

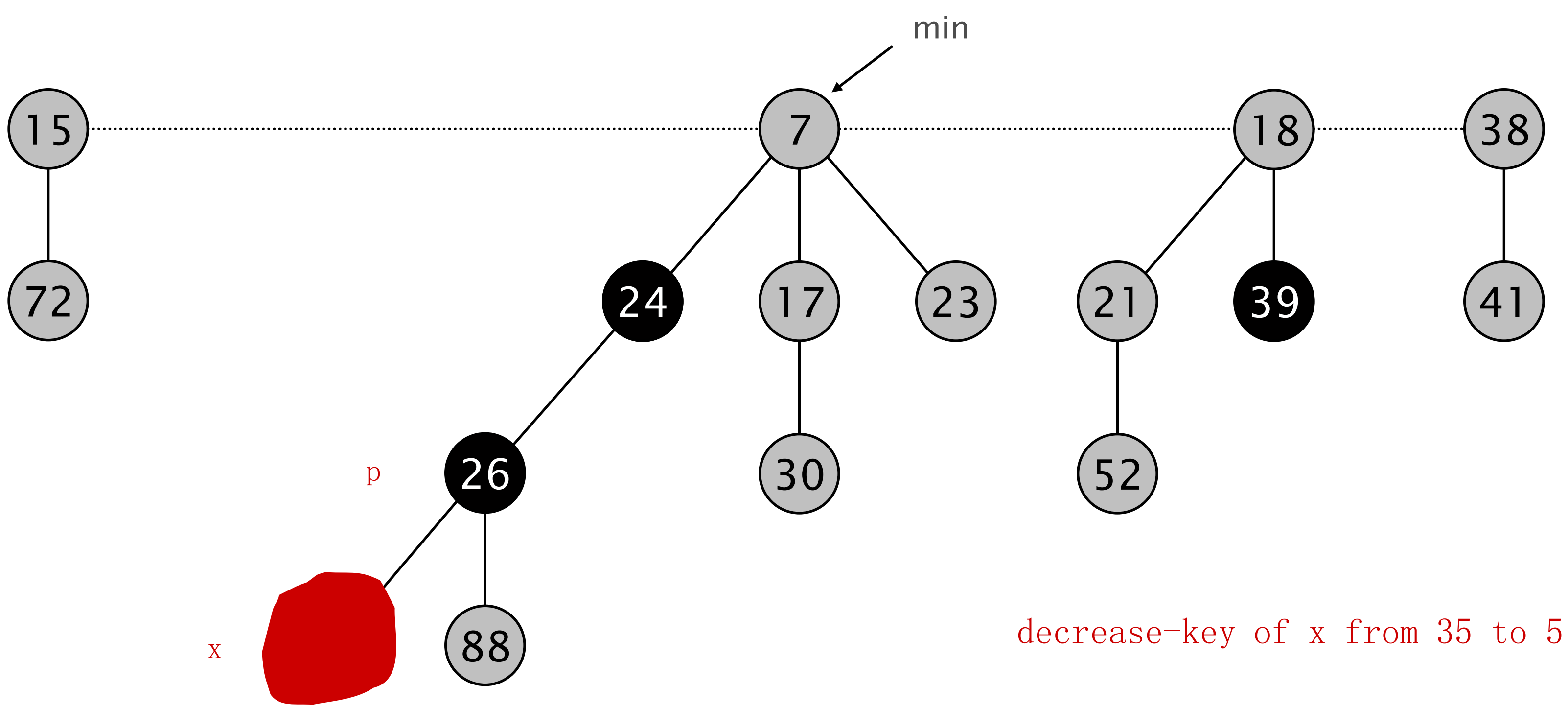
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

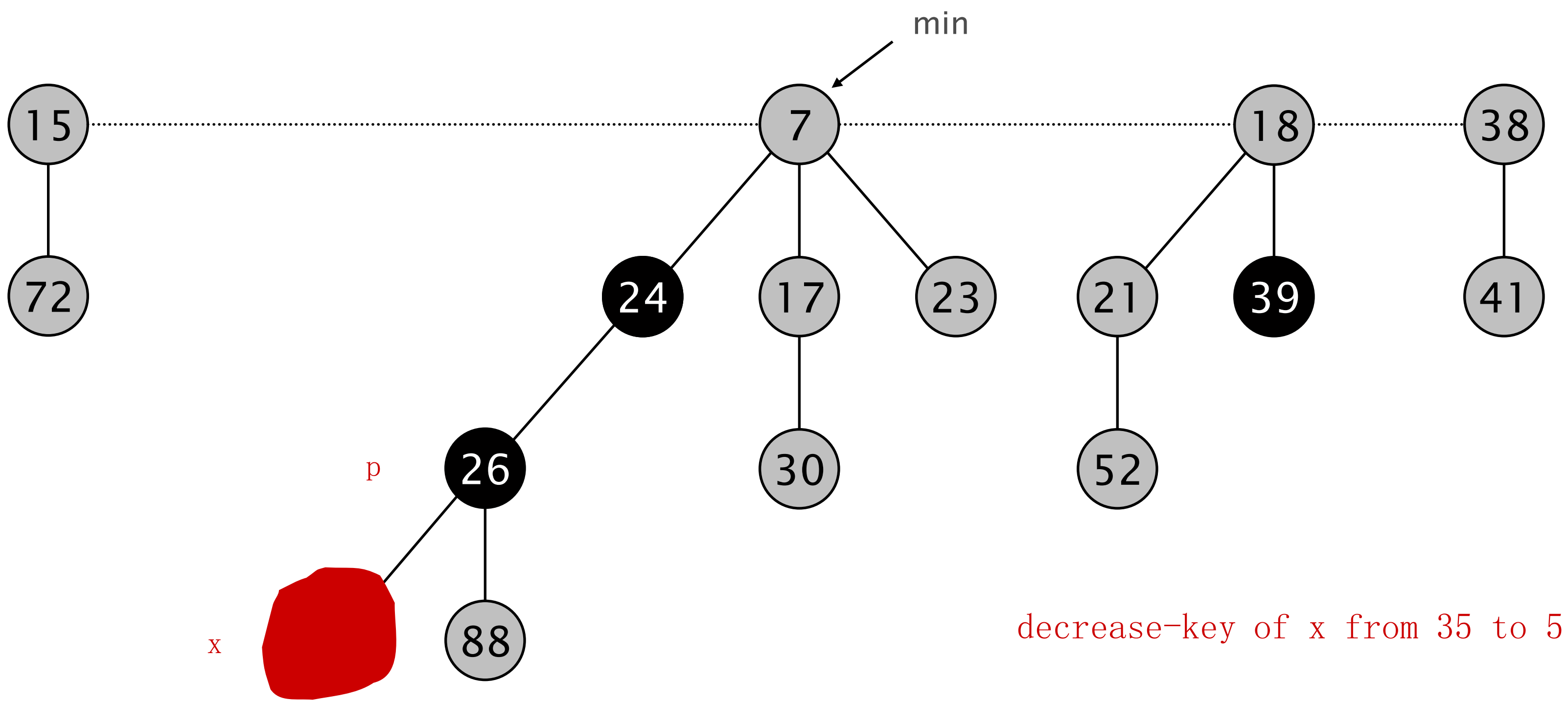
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

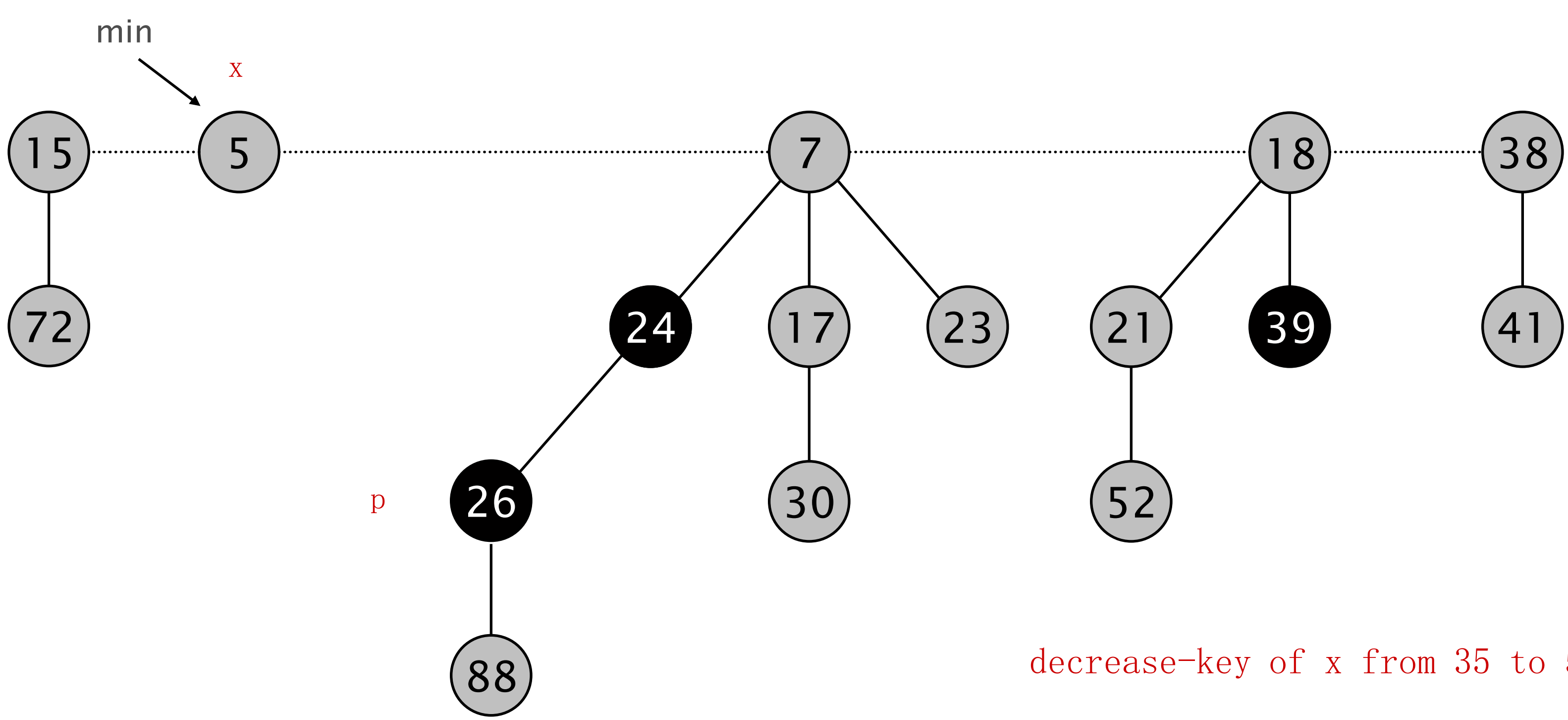
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



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).

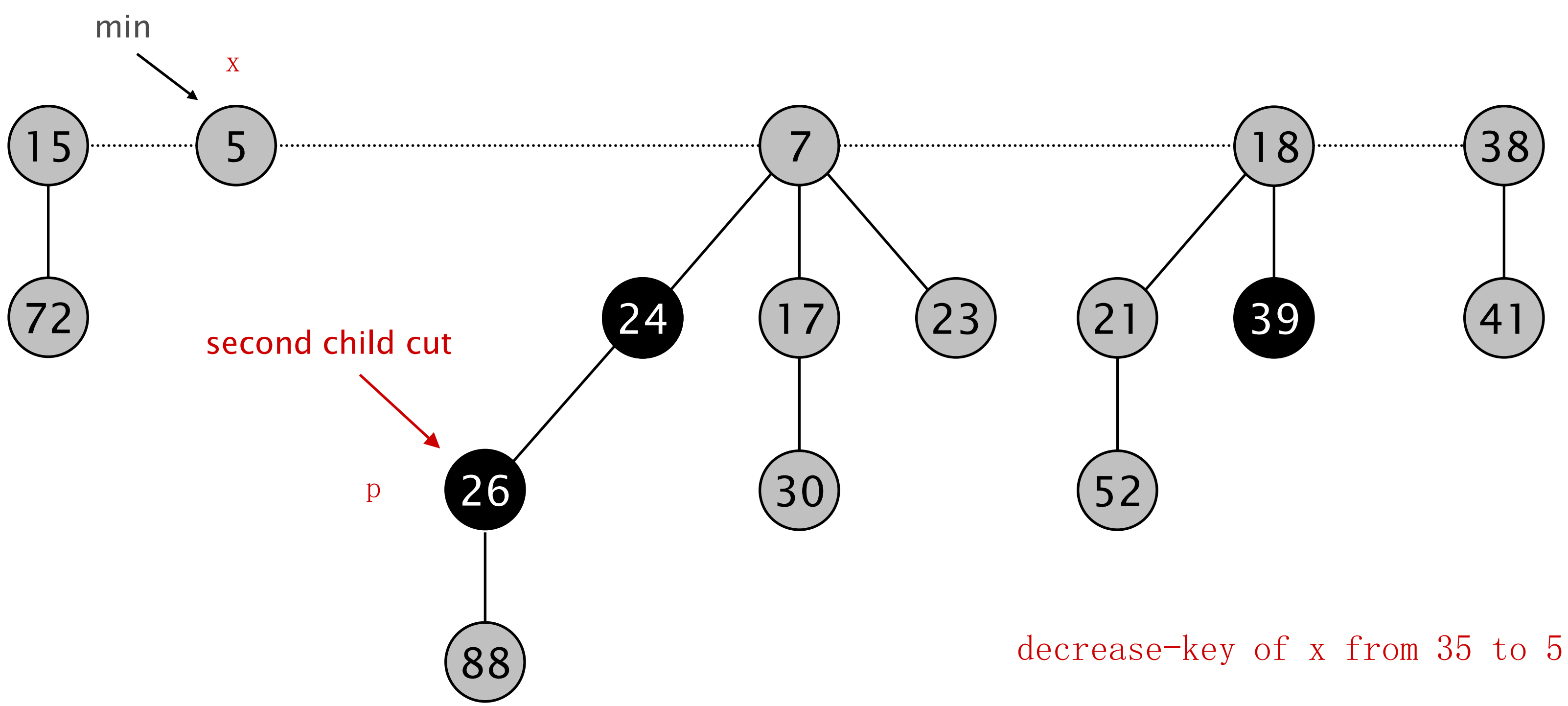


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

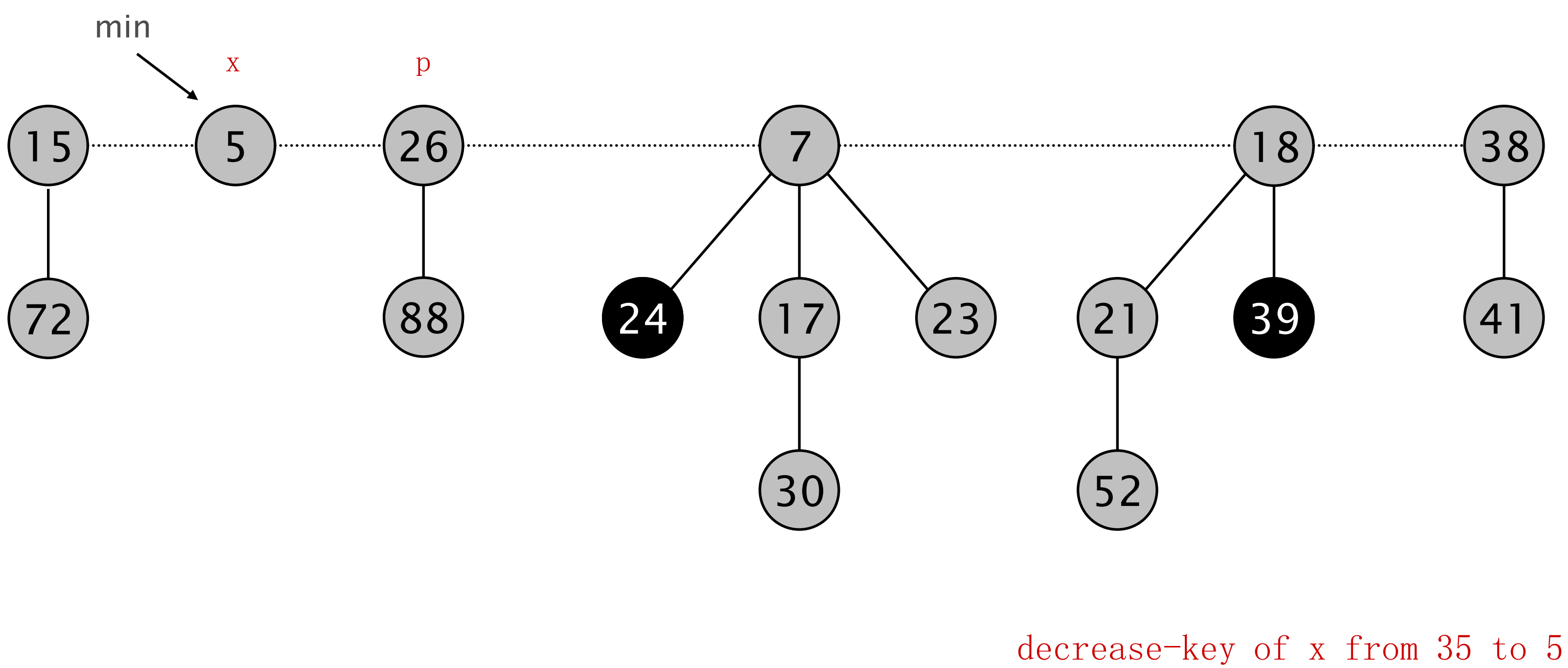
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

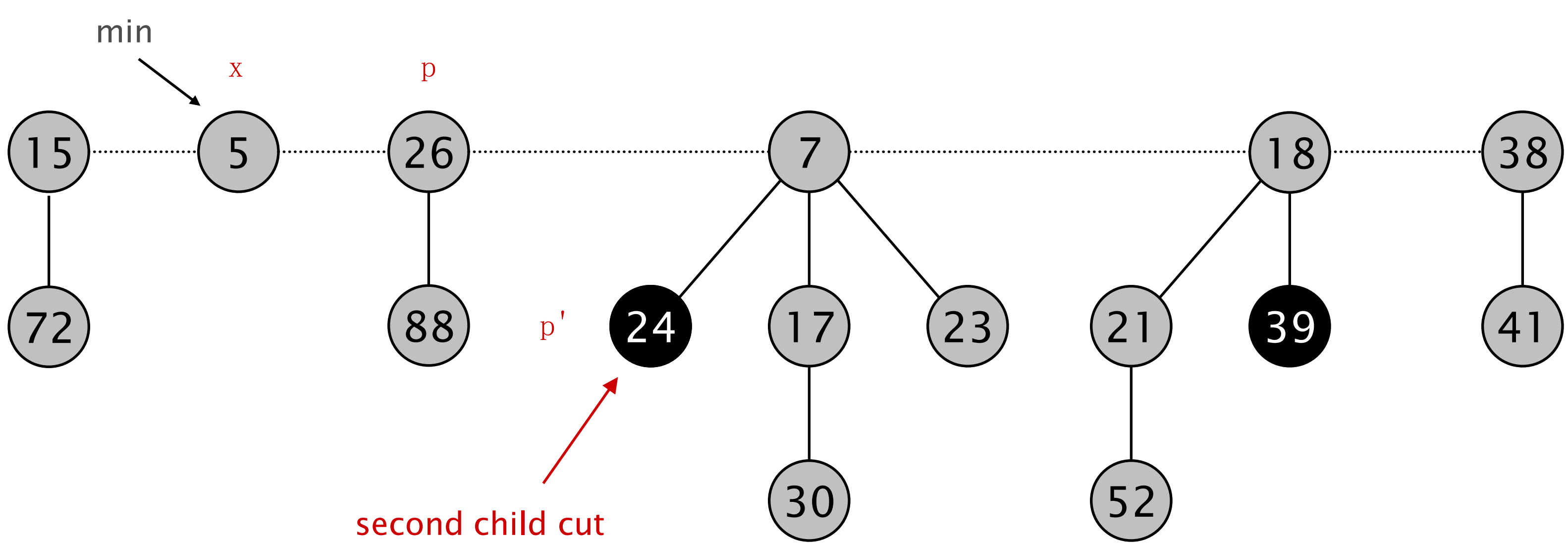
Case 2b. [heap order violated]

- 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).



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).

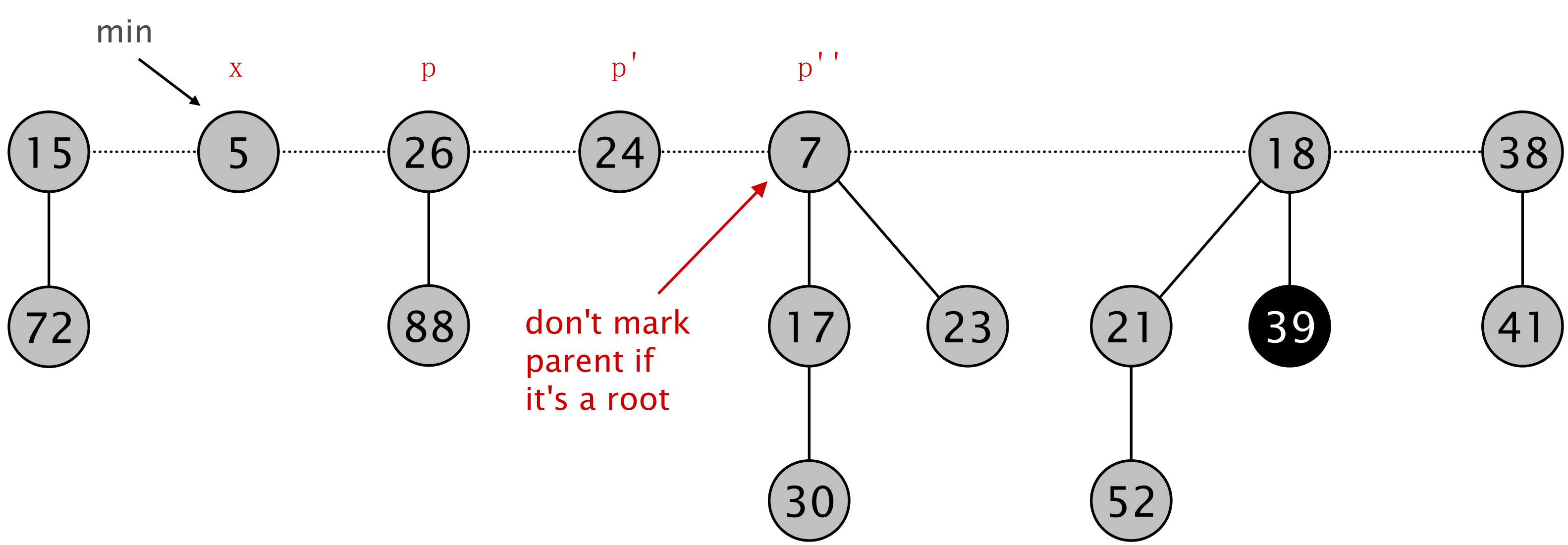


decrease-key of x from 35 to 5

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).

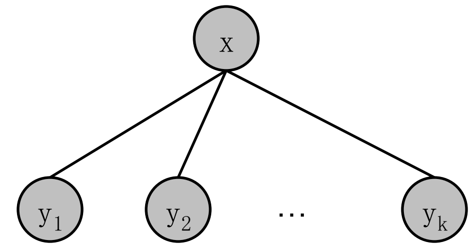


decrease-key of x from 35 to 5

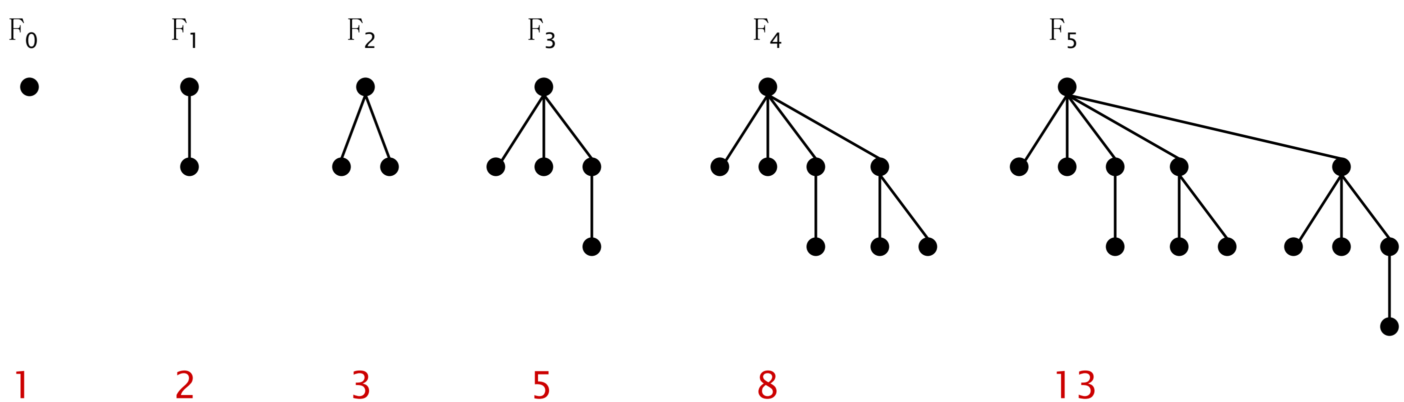
Fibonacci Heaps: Bounding the Rank

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:

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

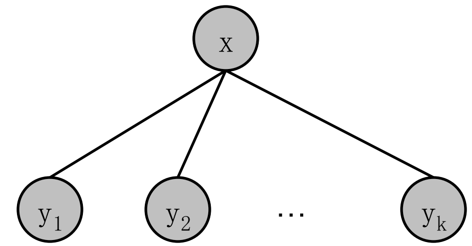


Def. Let F_k be smallest possible tree of rank k satisfying property.

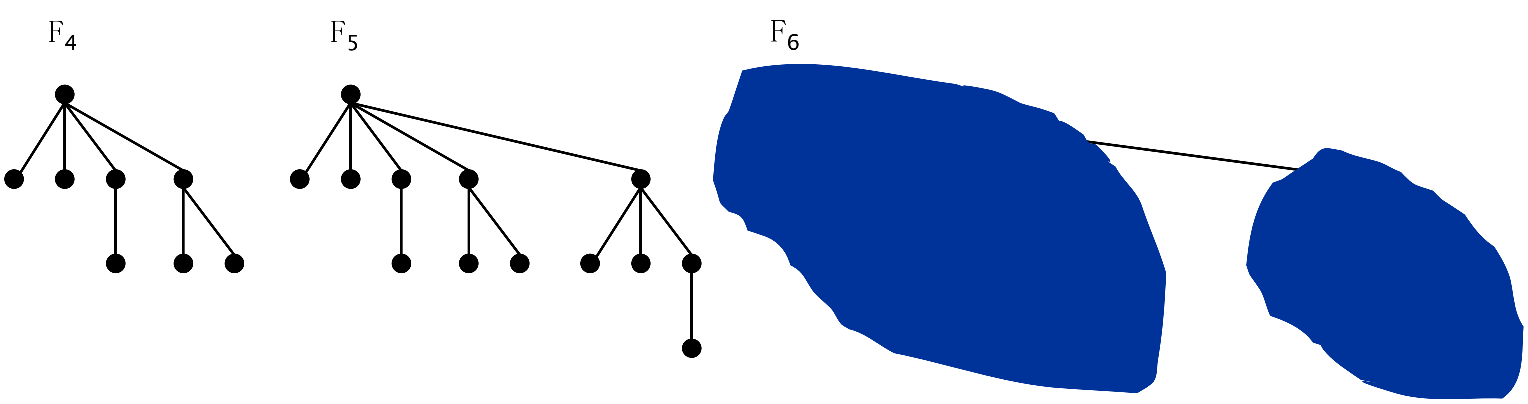


Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

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



Def. Let F_k be smallest possible tree of rank k satisfying property.



8 13 8 + 13 = 21