Fibonacci Heaps

Lecture slides adapted from:

- Chapter 20 of *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein.
- Chapter 9 of *The Design and Analysis of Algorithms* by Dexter Kozen.

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.

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

Hopeless challenge. O(1) *insert*, *delete-min* and *decrease-key*. Why?

Fibonacci Heaps

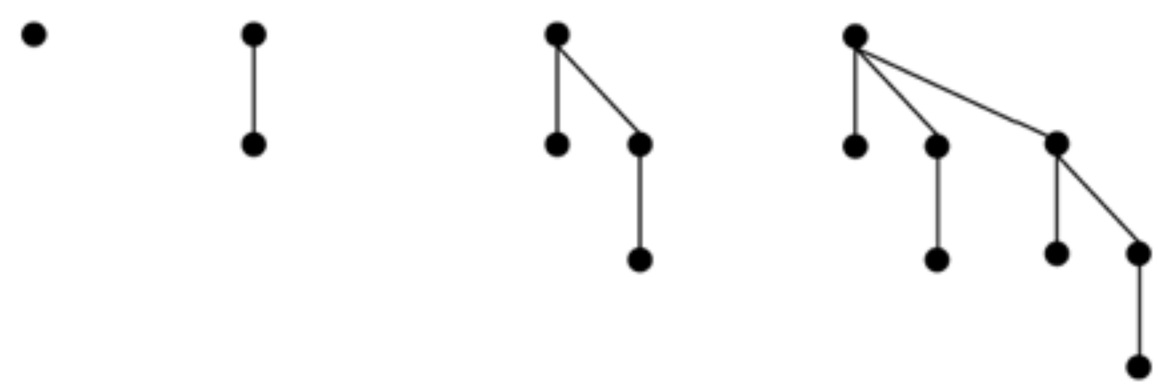
History. [Fredman and Tarjan, 1986]

- 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.

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



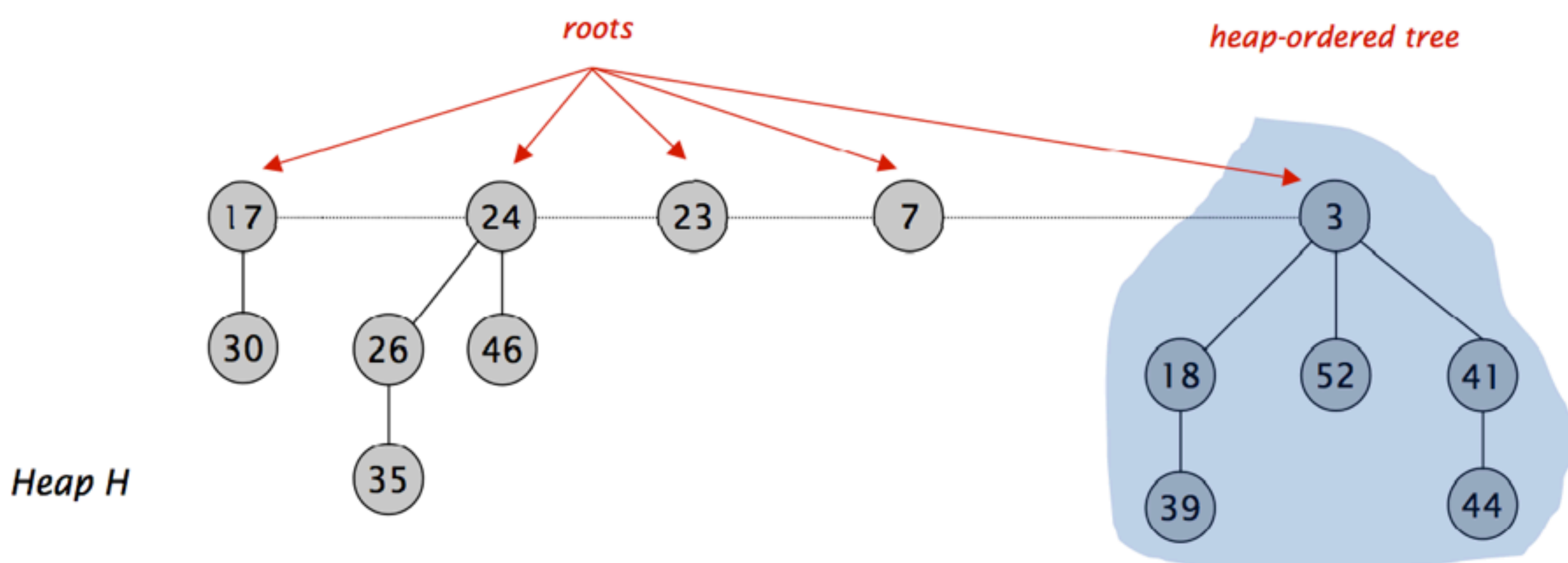
- Fibonacci heap: lazily defer consolidation until next *delete-min*.

Fibonacci Heaps: Structure

each parent larger than its children

Fibonacci heap.

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

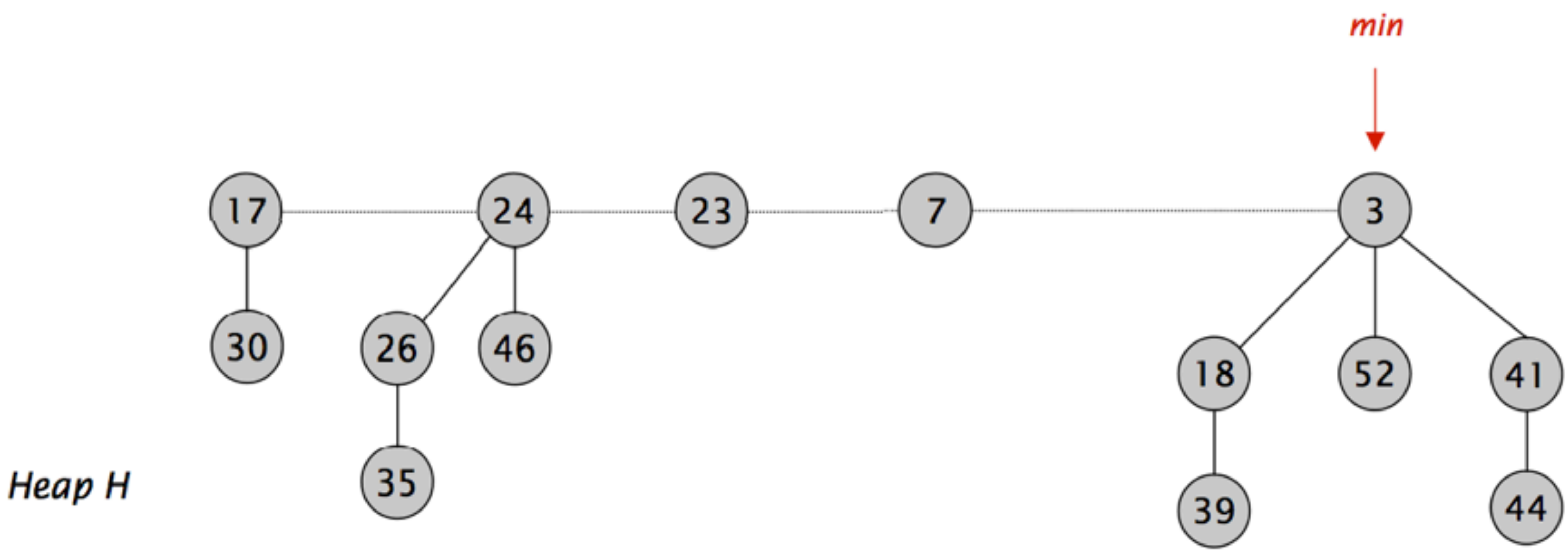


Fibonacci Heaps: Structure

Fibonacci heap.

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

find-min takes O(1) time

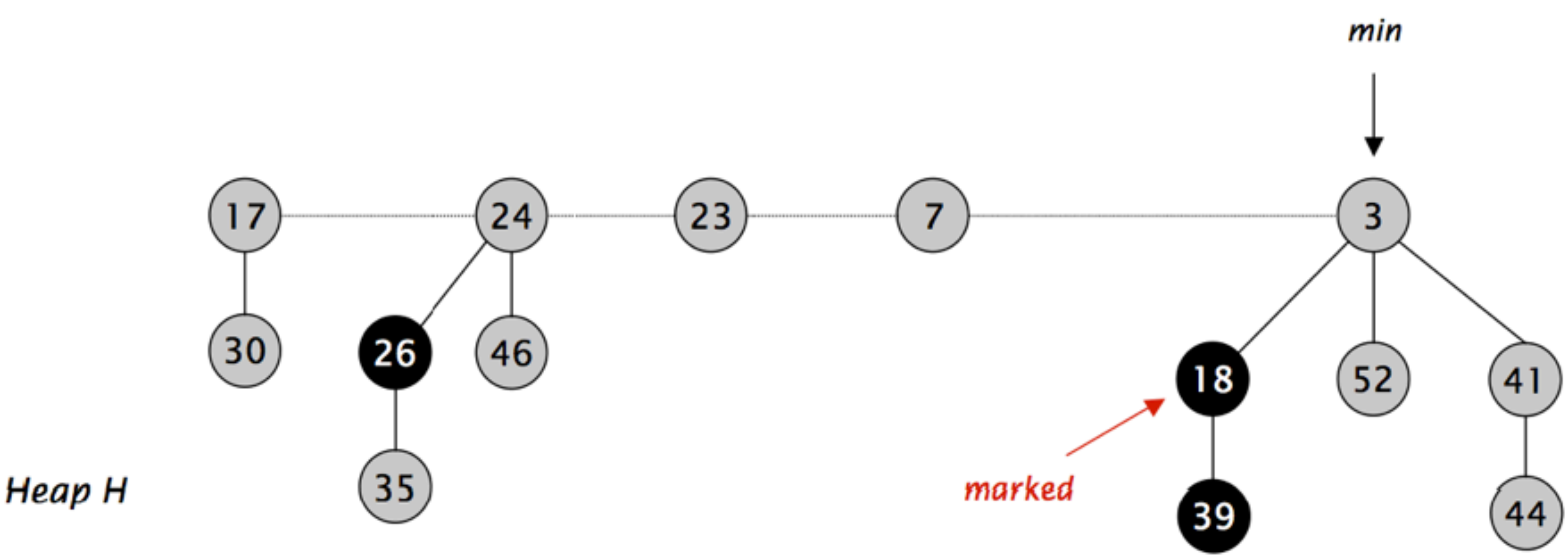


Fibonacci Heaps: Structure

Fibonacci heap.

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

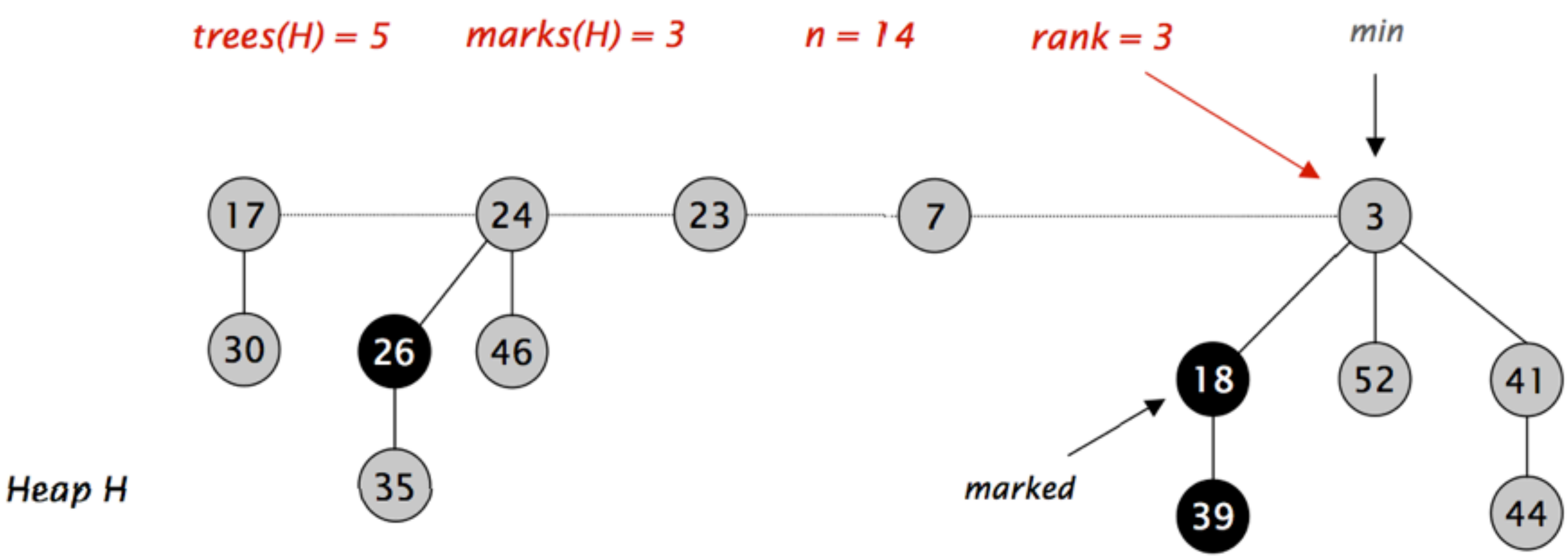
use to keep heaps flat (stay tuned)



Fibonacci Heaps: Notation

Notation.

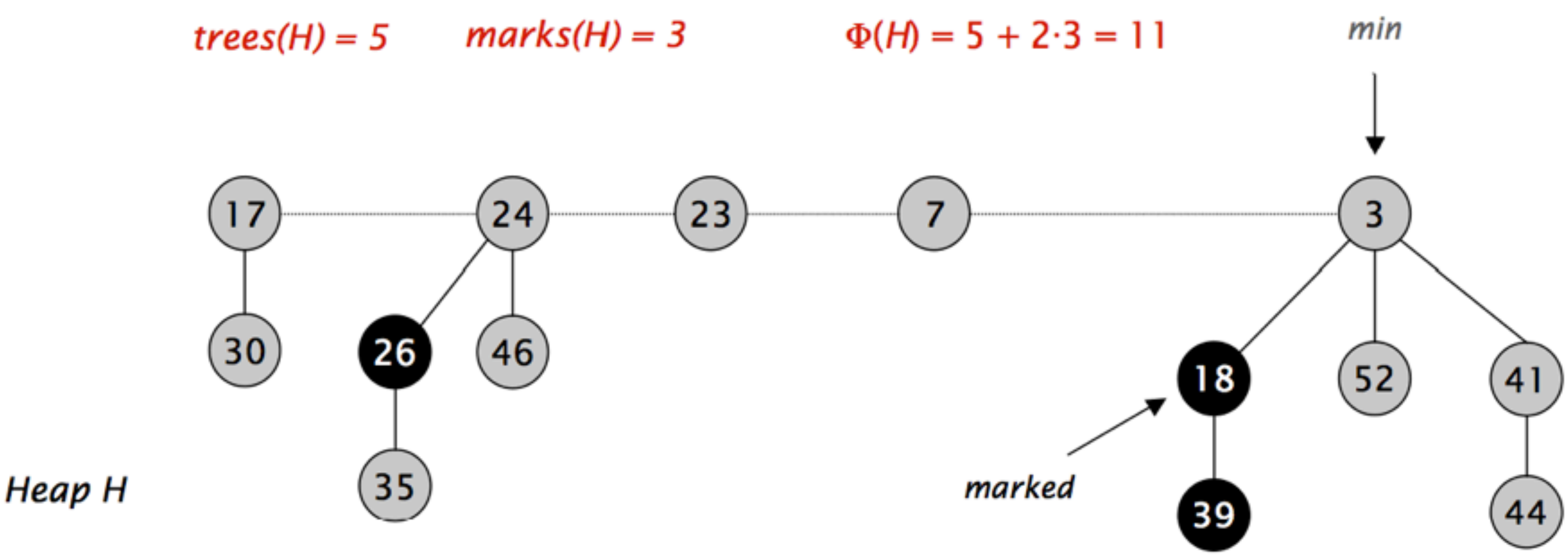
- *n* = number of nodes in heap.
- *rank(x)* = number of children of node *x*.
- *rank(H)* = max rank of any node in heap *H*.
- *trees(H)* = number of trees in heap *H*.
- *marks(H)* = number of marked nodes in heap *H*.



Fibonacci Heaps: Potential Function

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

potential of heap H

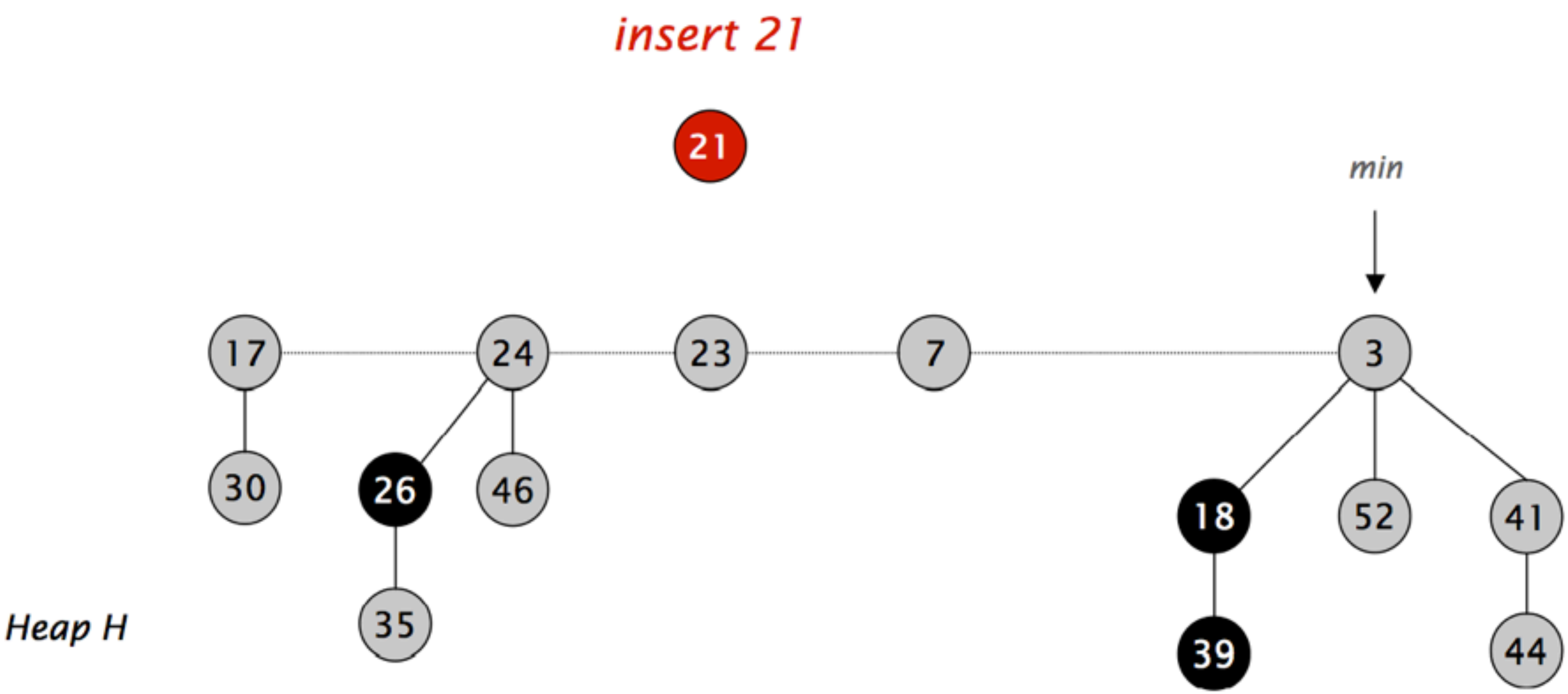


Insert

Fibonacci Heaps: Insert

Insert.

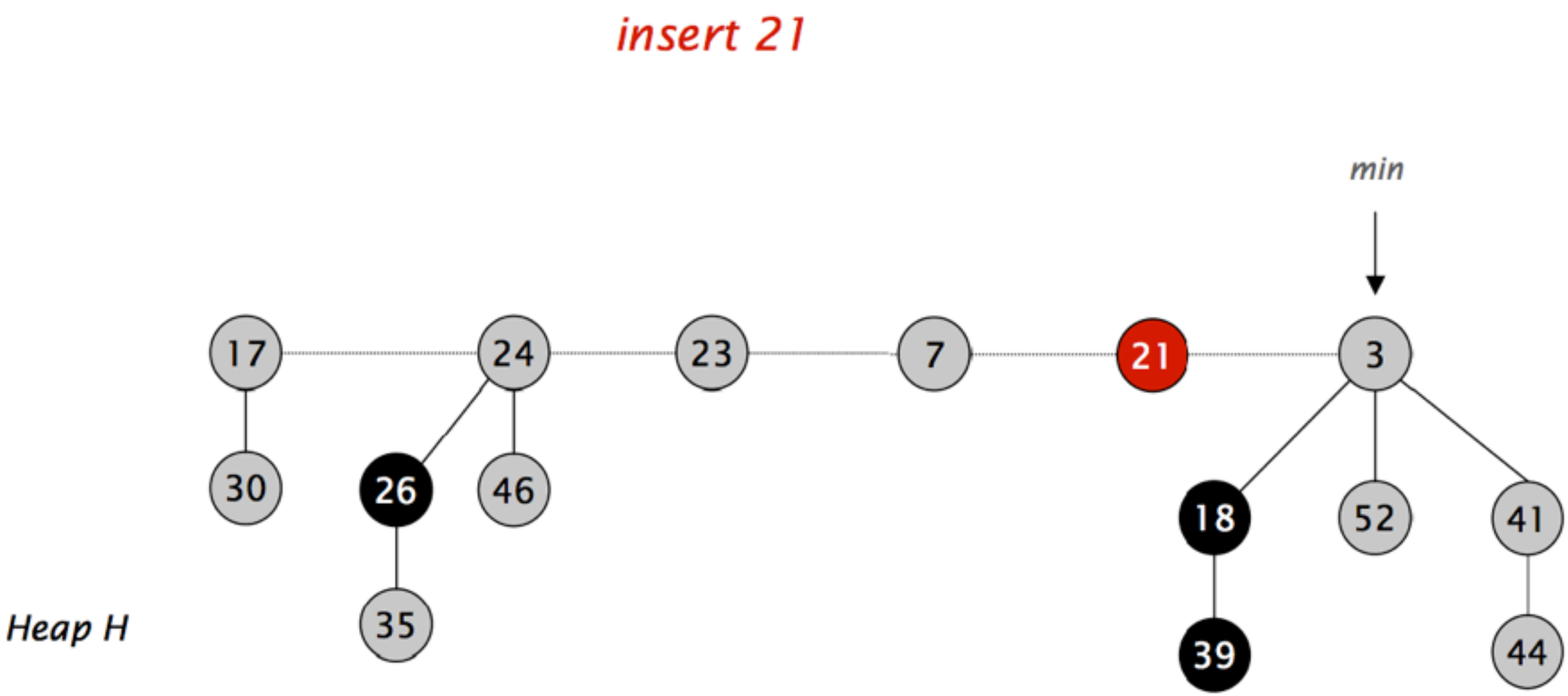
- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).



Fibonacci Heaps: Insert

Insert.

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



Fibonacci Heaps: Insert Analysis

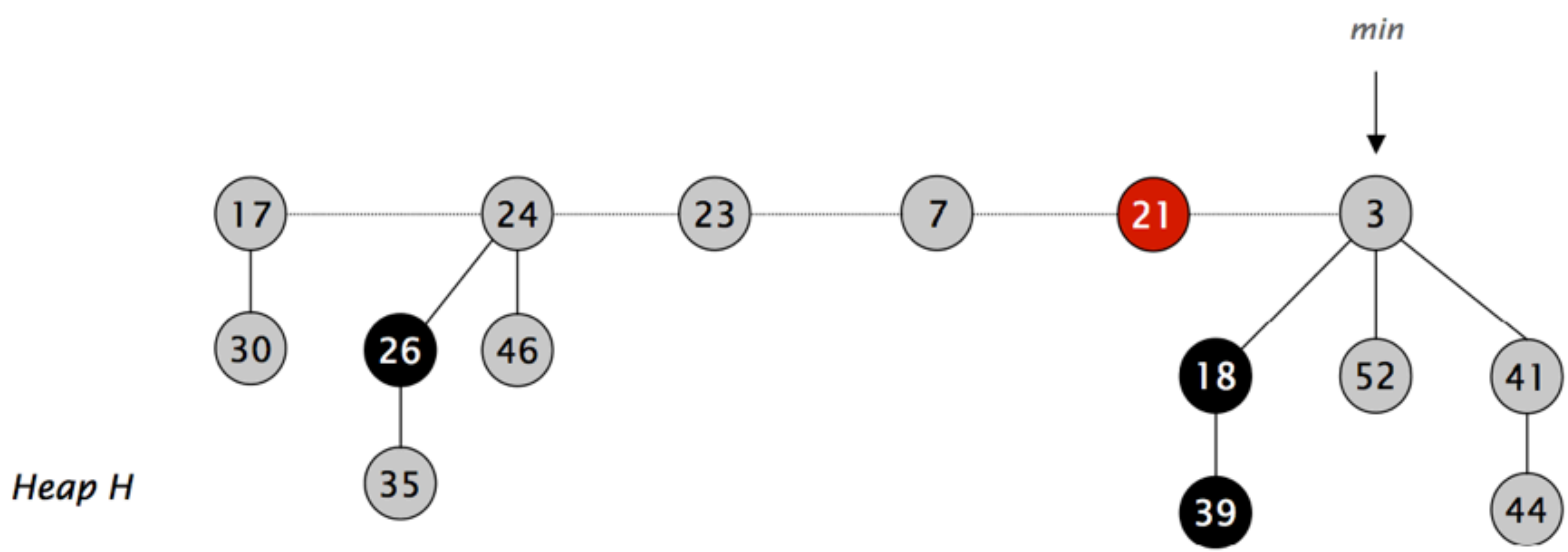
Actual cost. $O(1)$

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

potential of heap H

Change in potential. $+1$

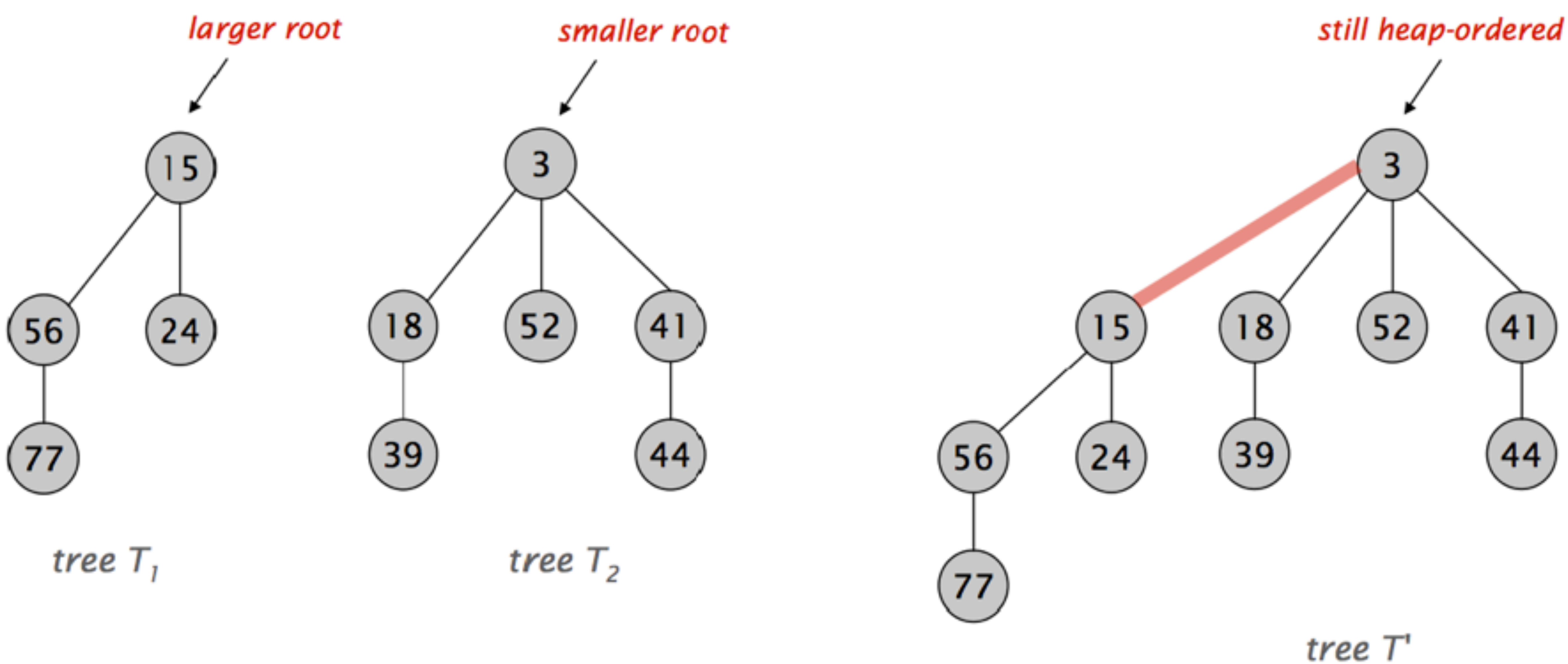
Amortized cost. $O(1)$



Delete Min

Linking Operation

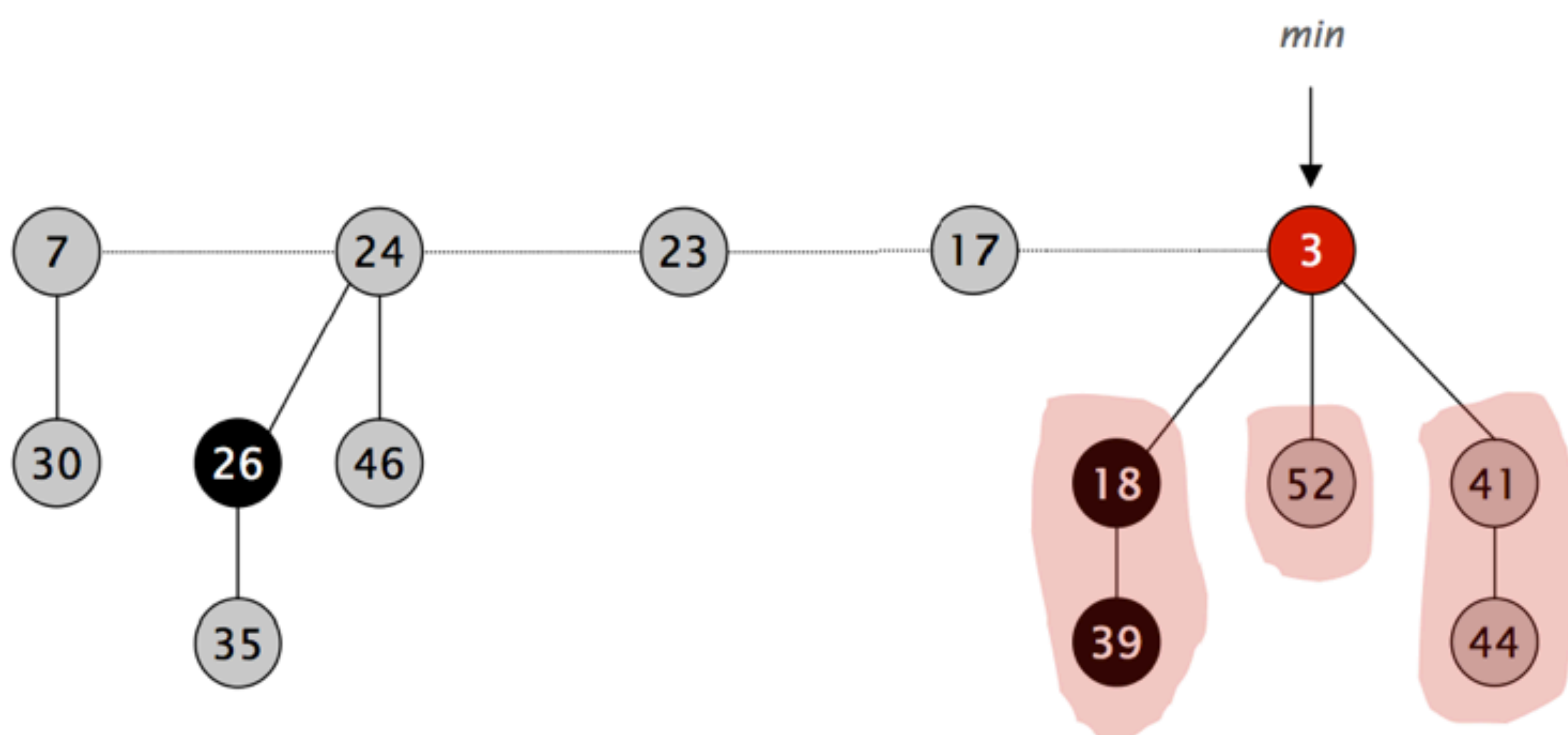
Linking operation. Make larger root be a child of smaller root.



Fibonacci Heaps: Delete Min

Delete min.

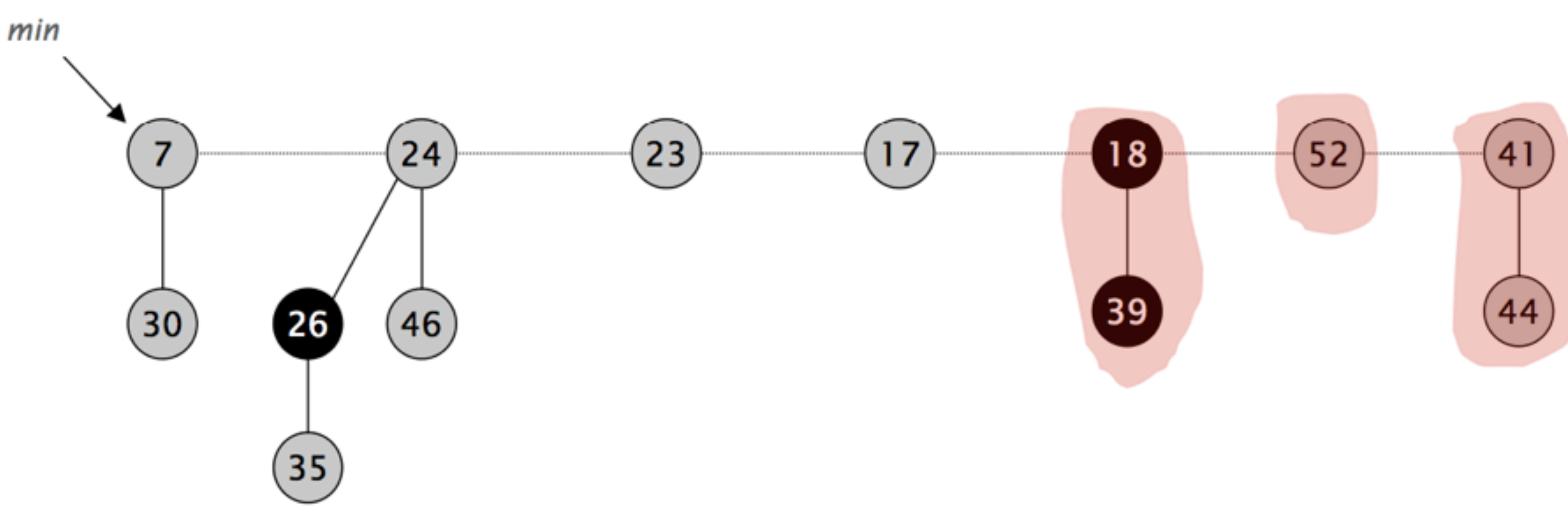
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

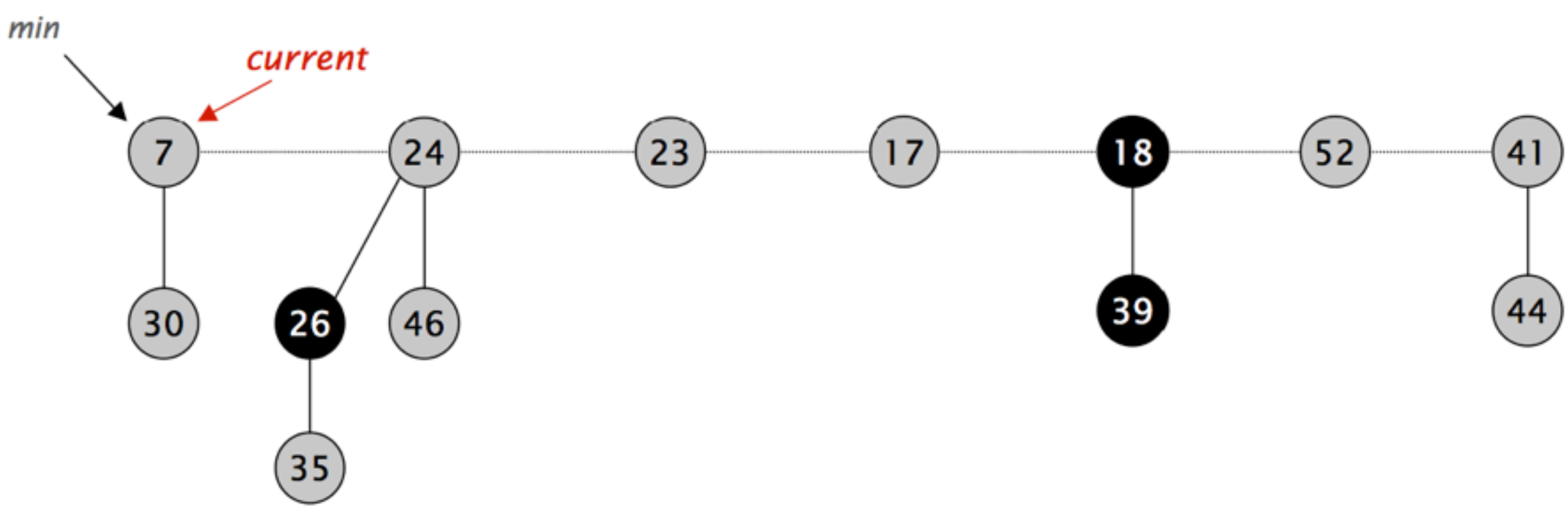
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

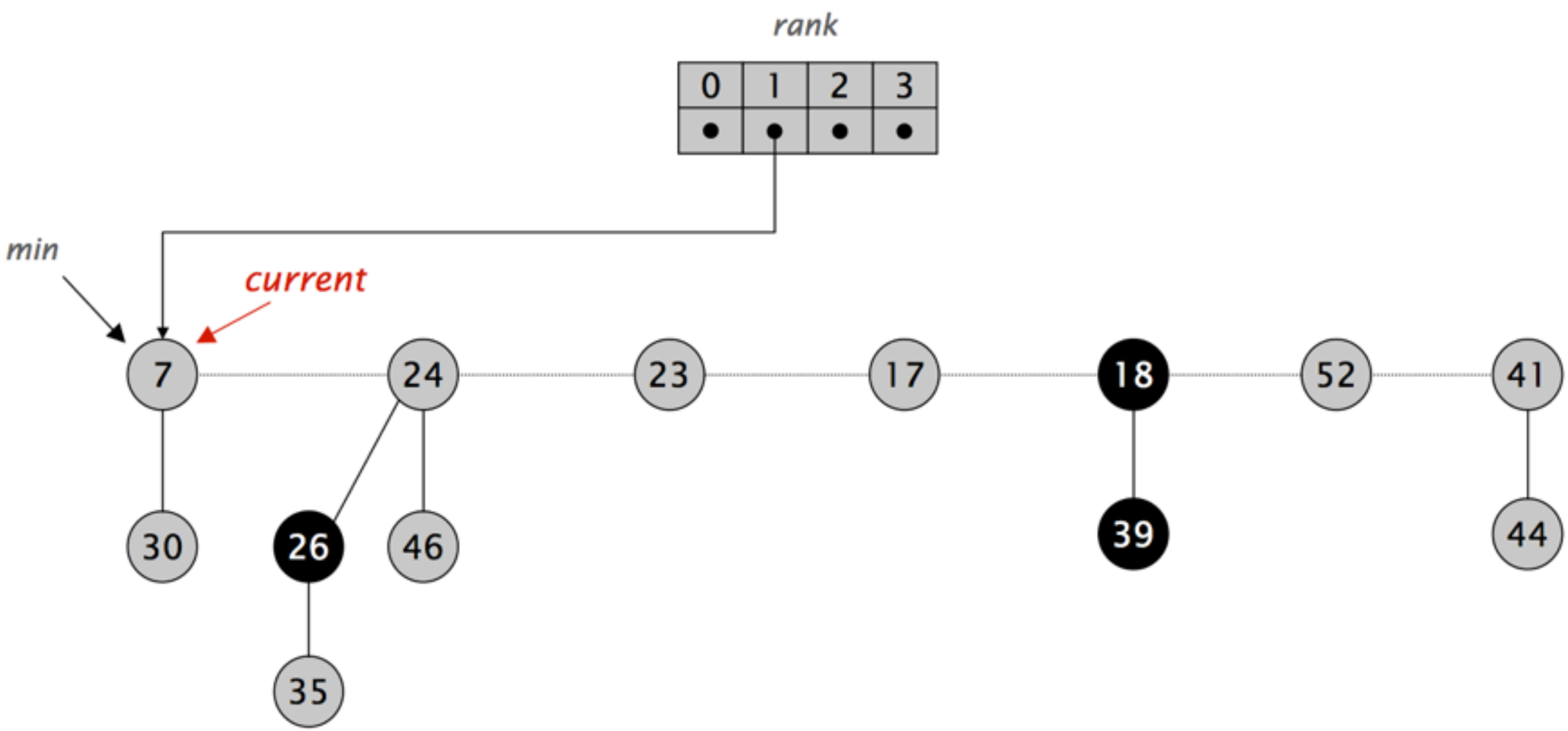
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

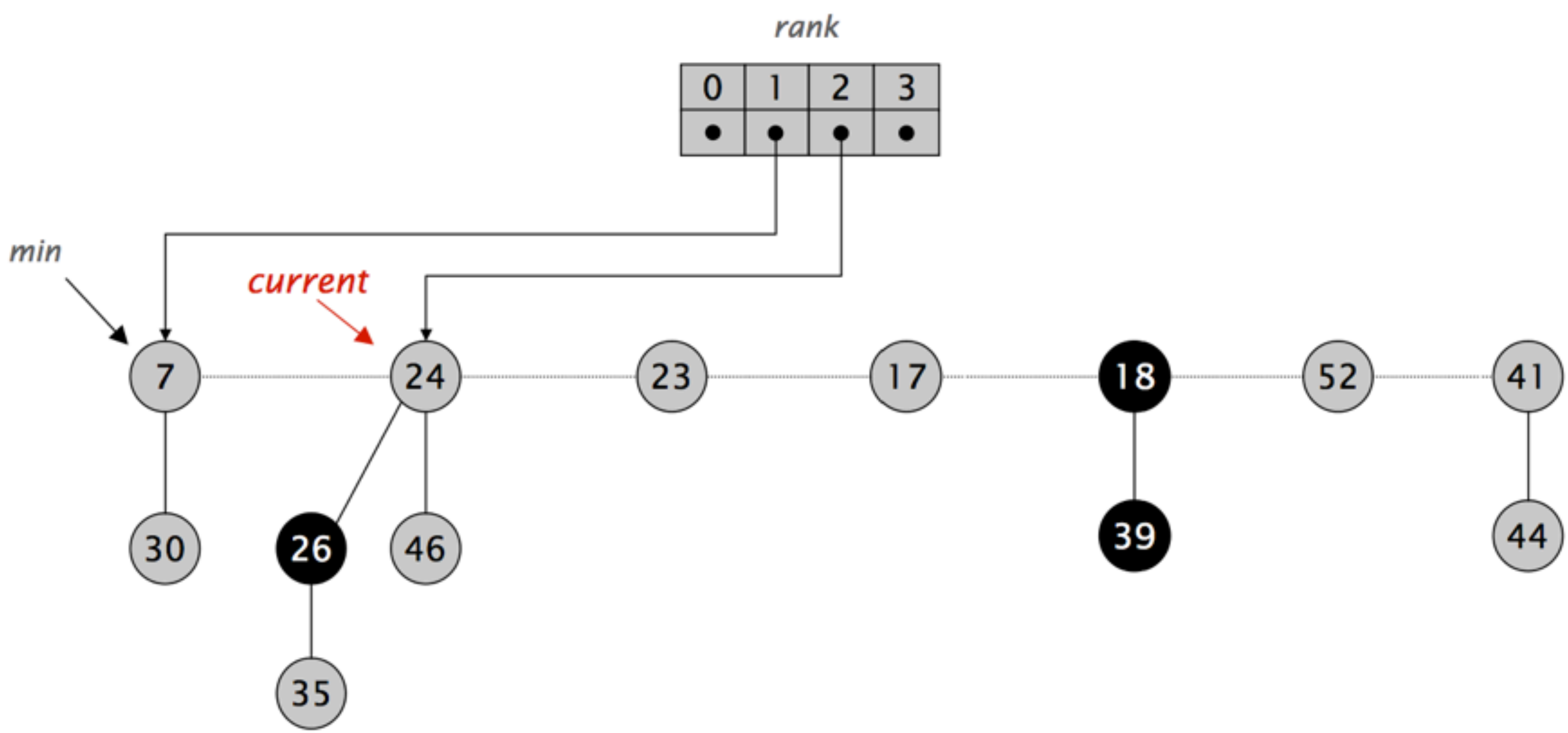
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

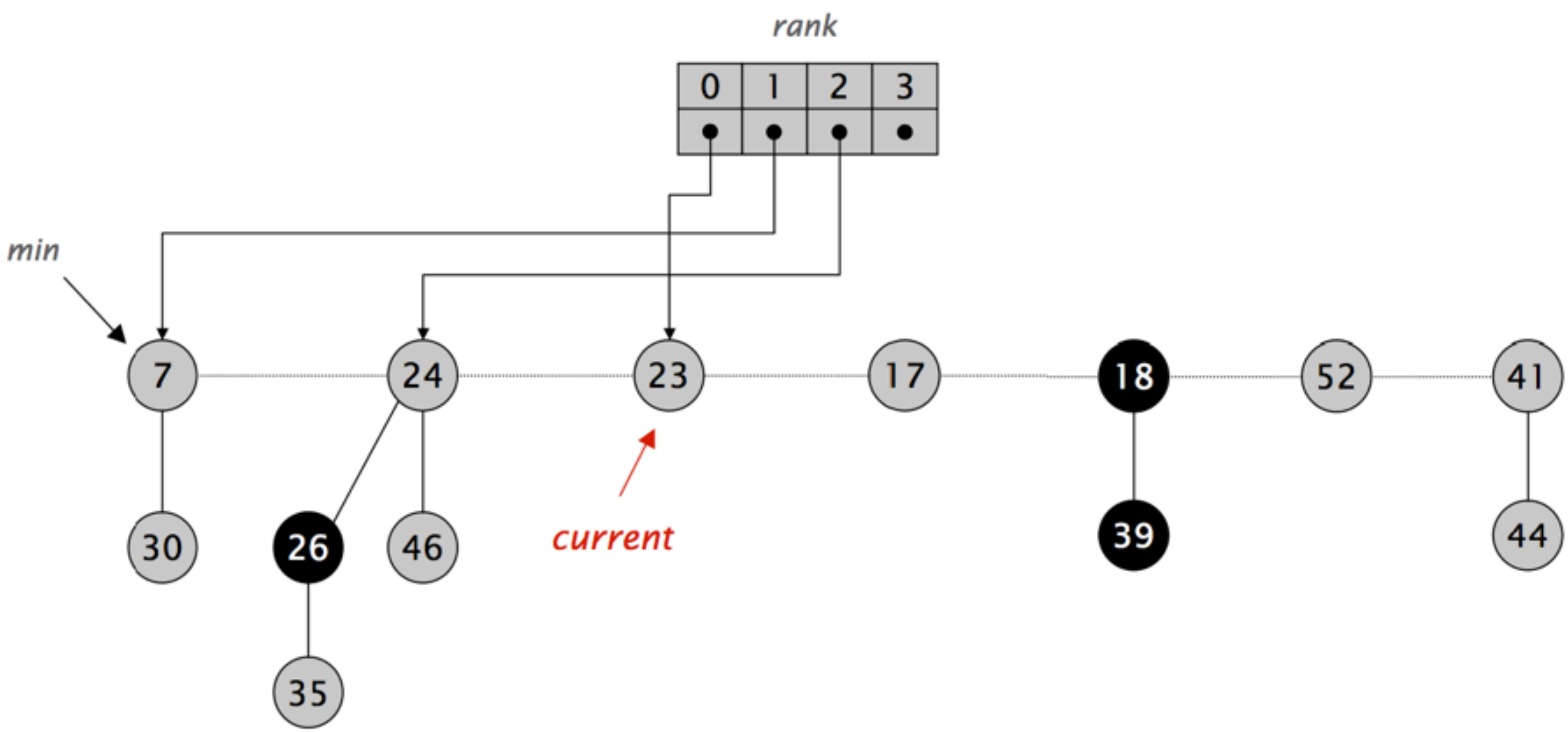
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

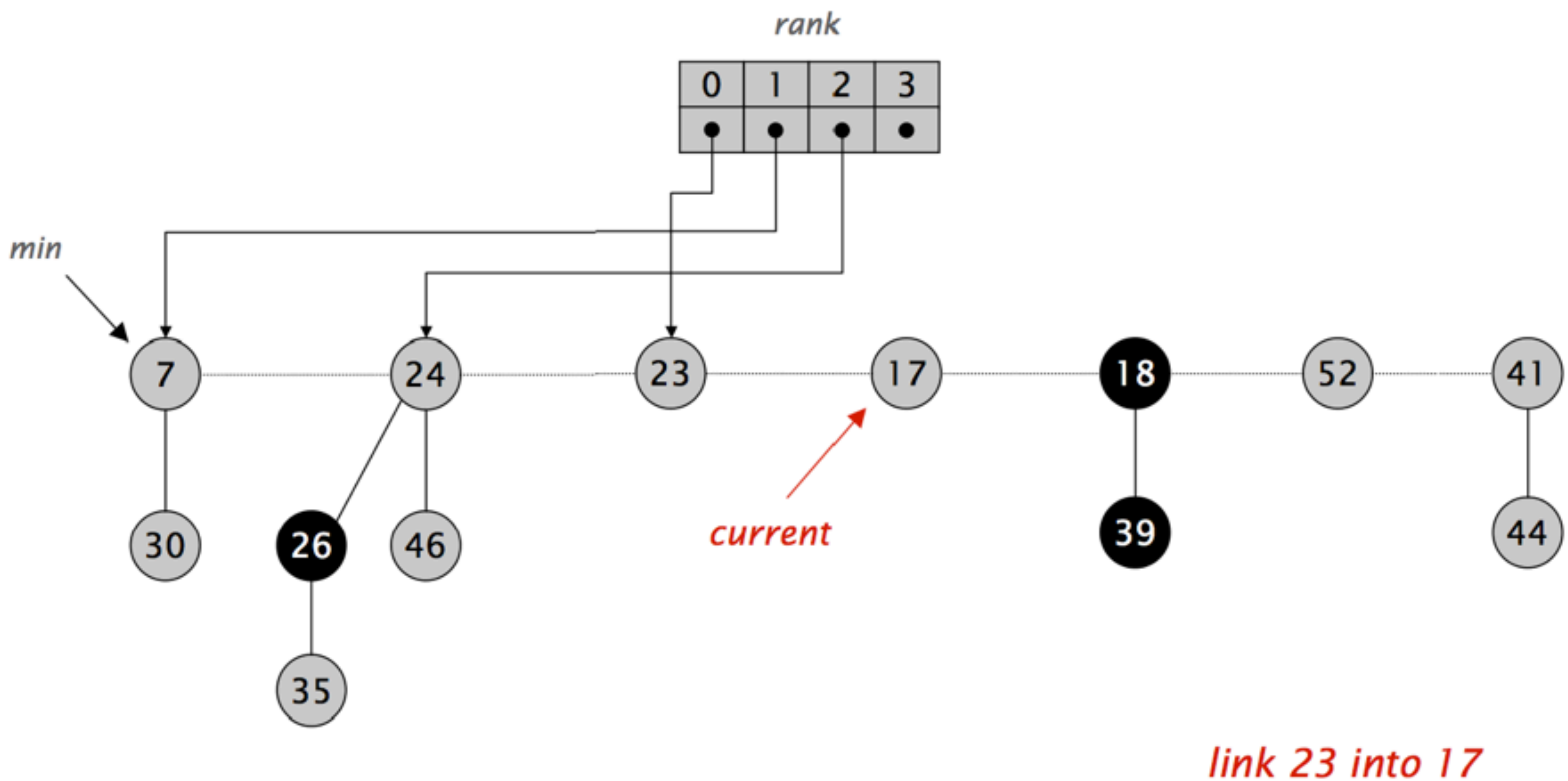
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

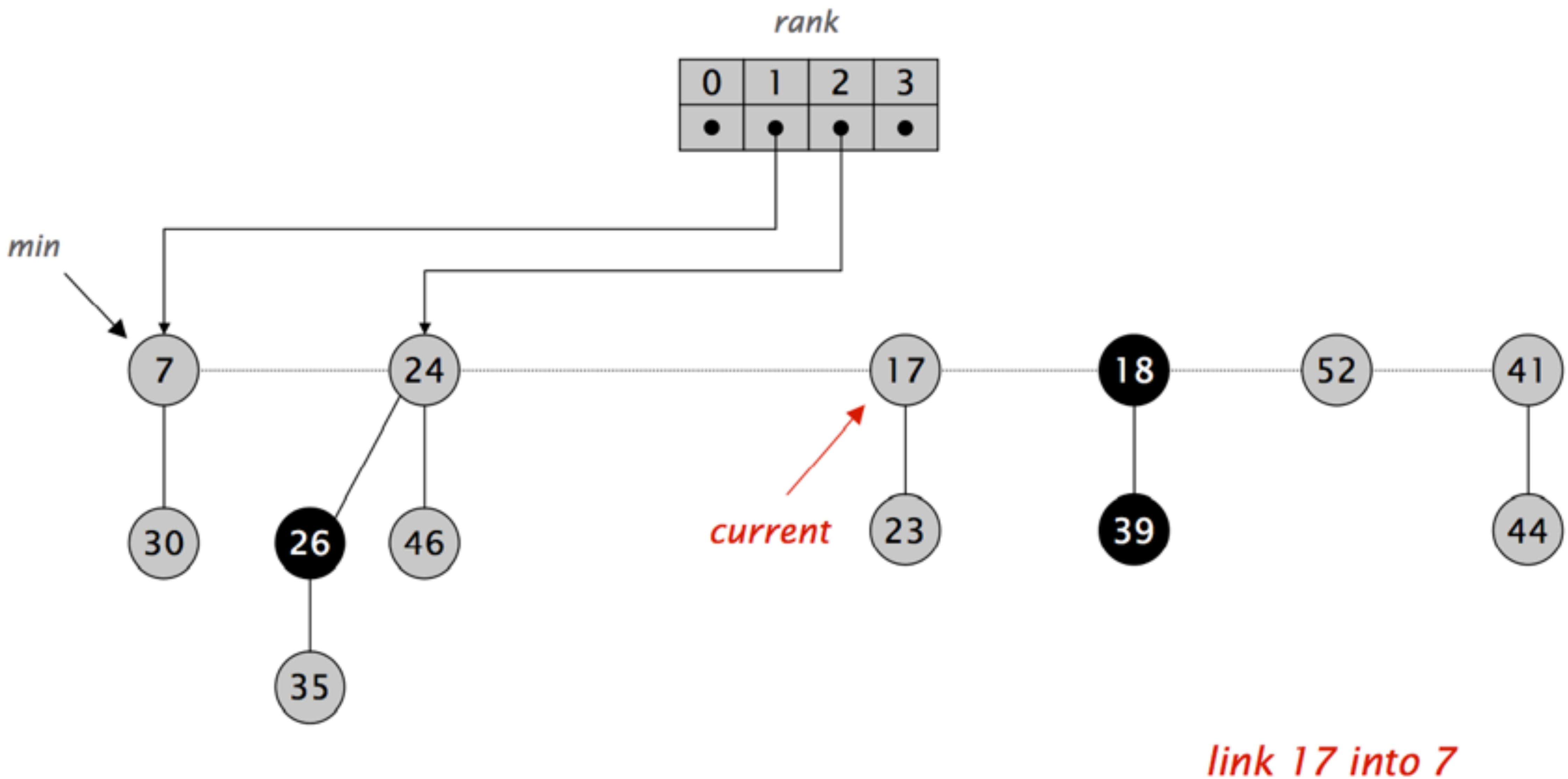
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

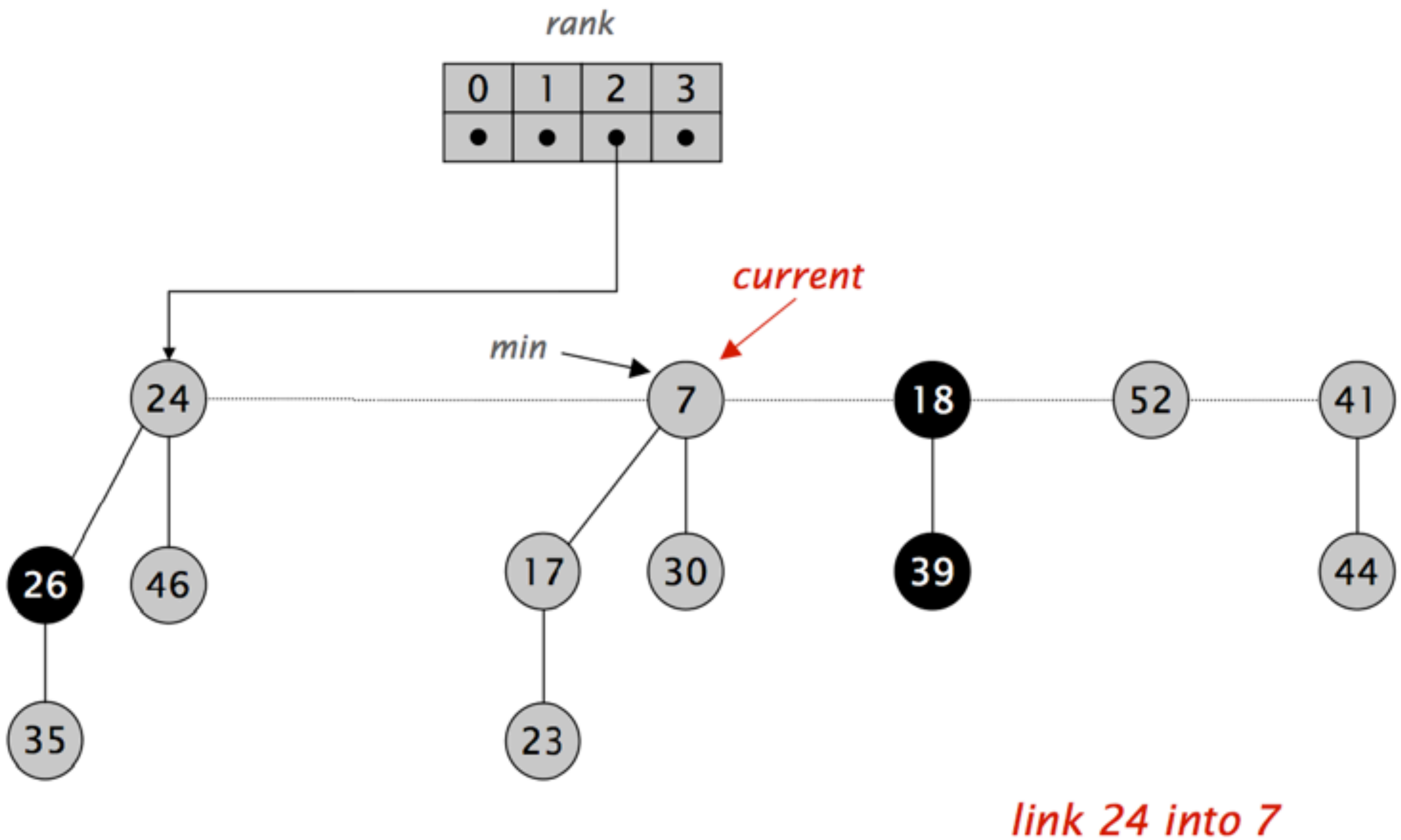
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

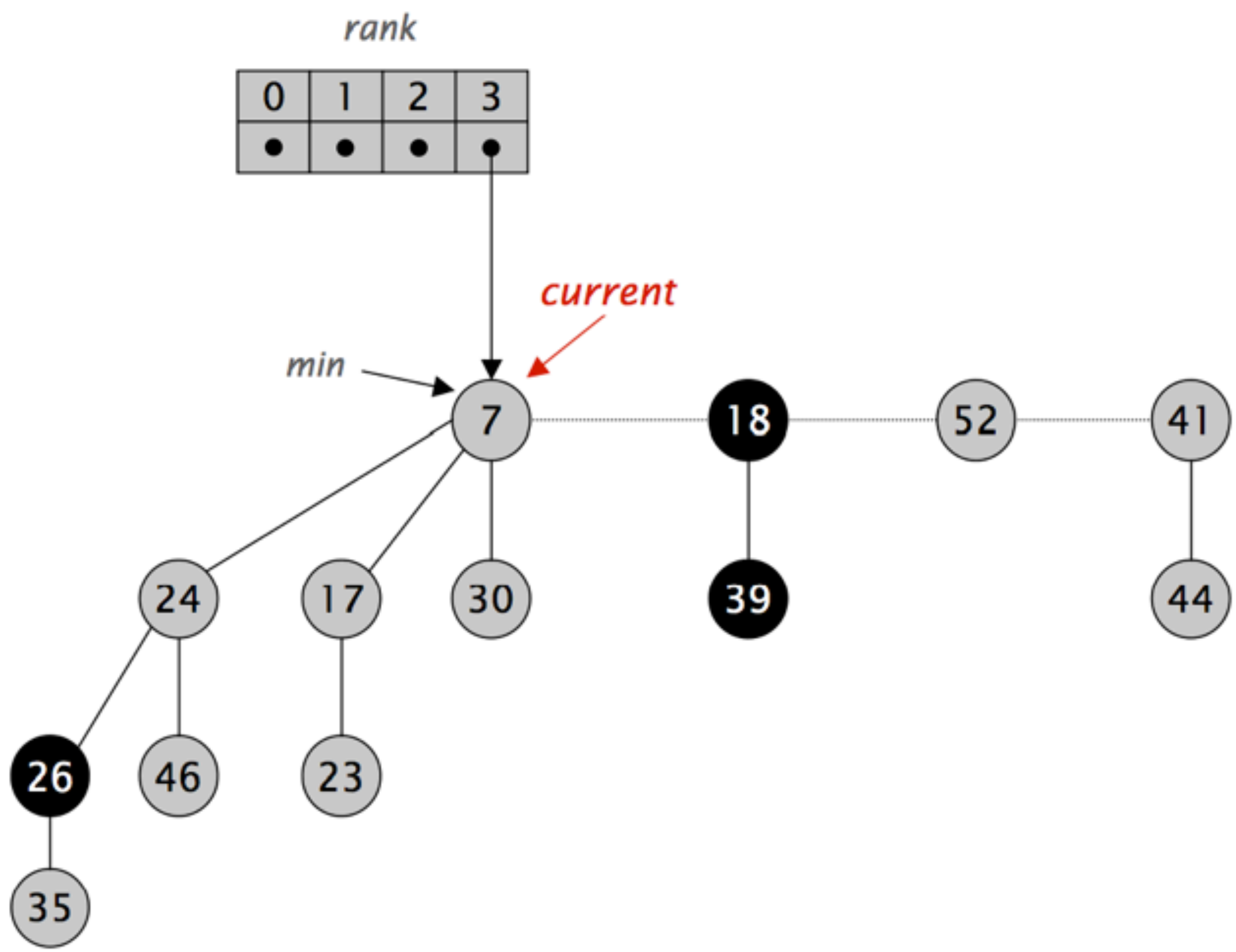
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

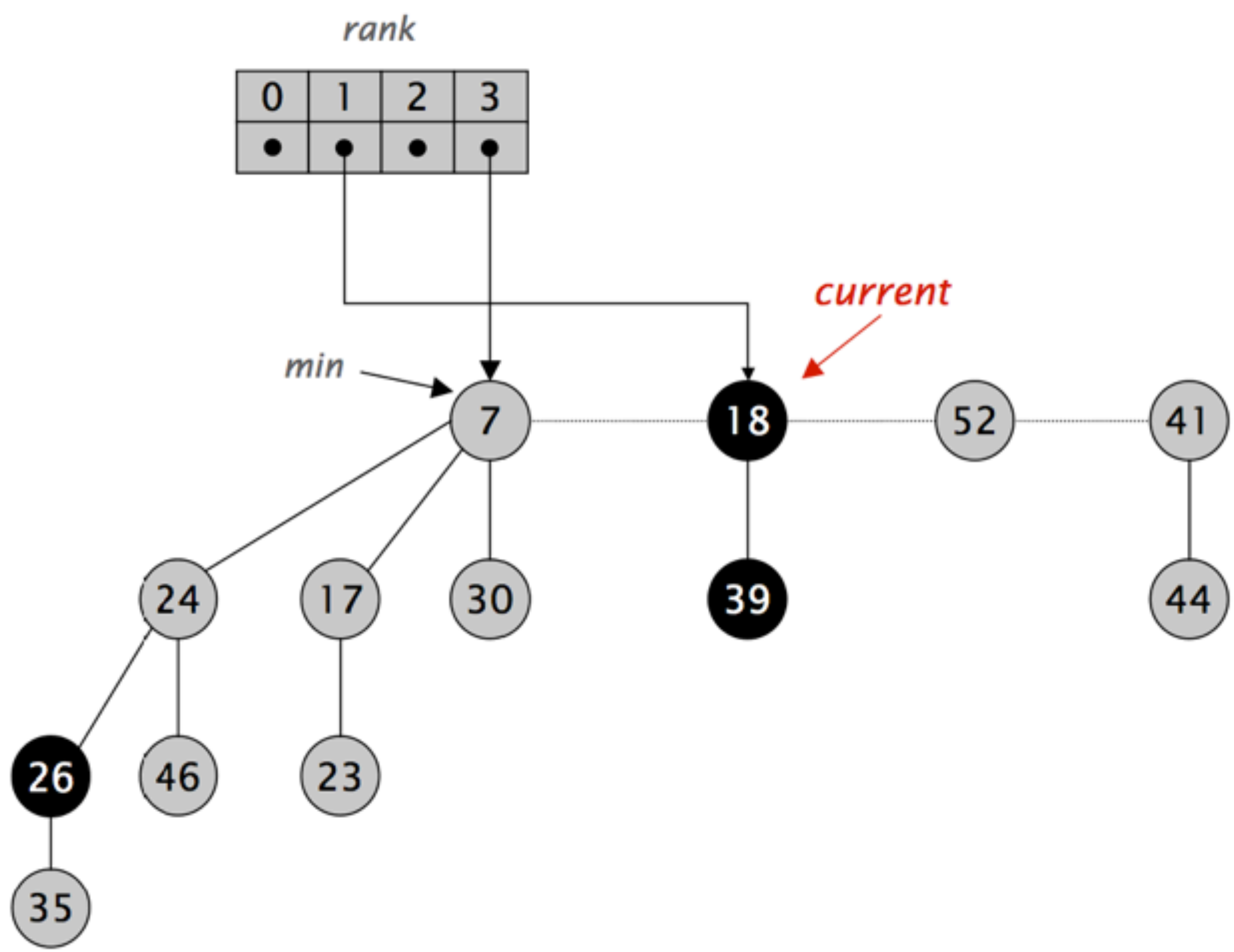
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

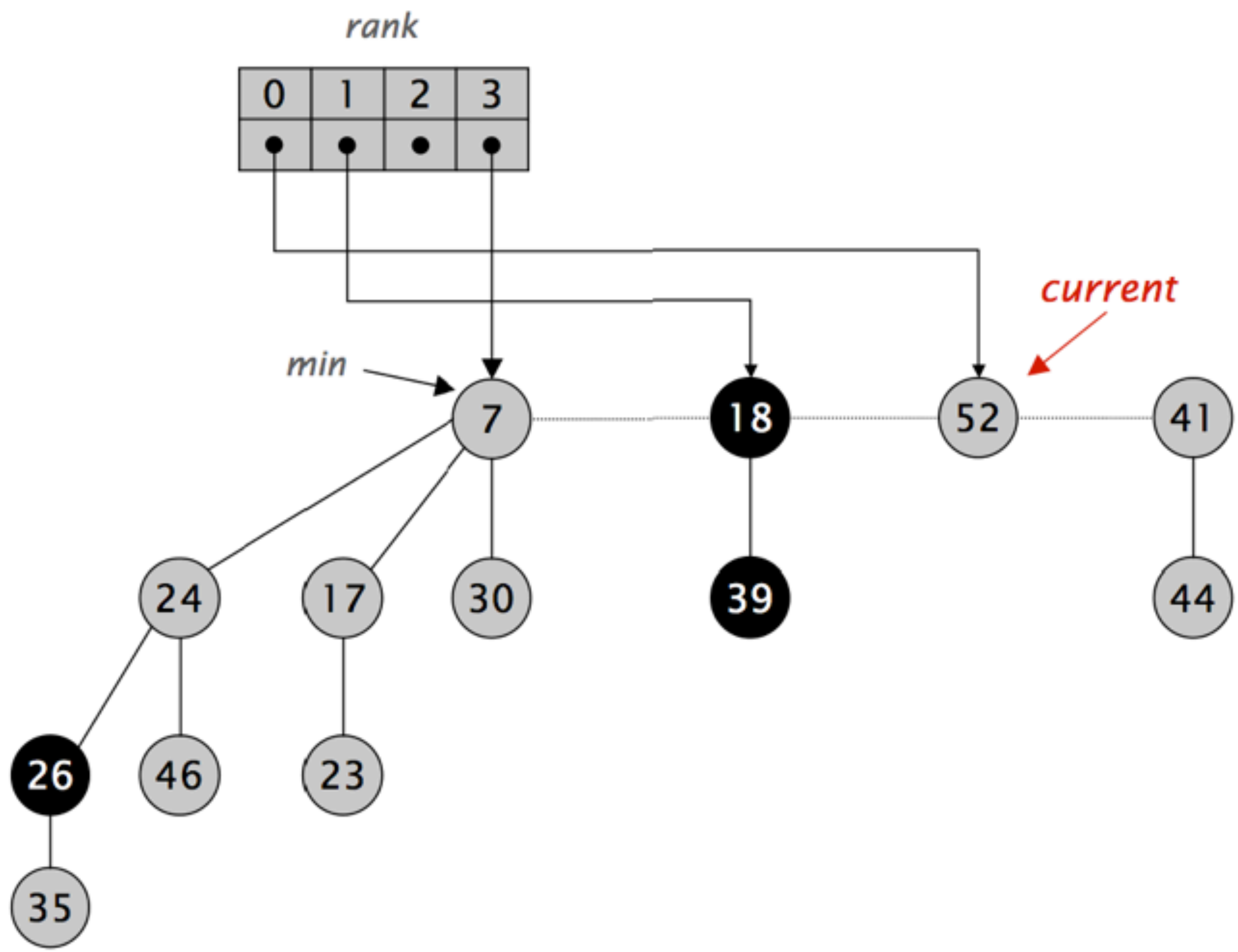
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

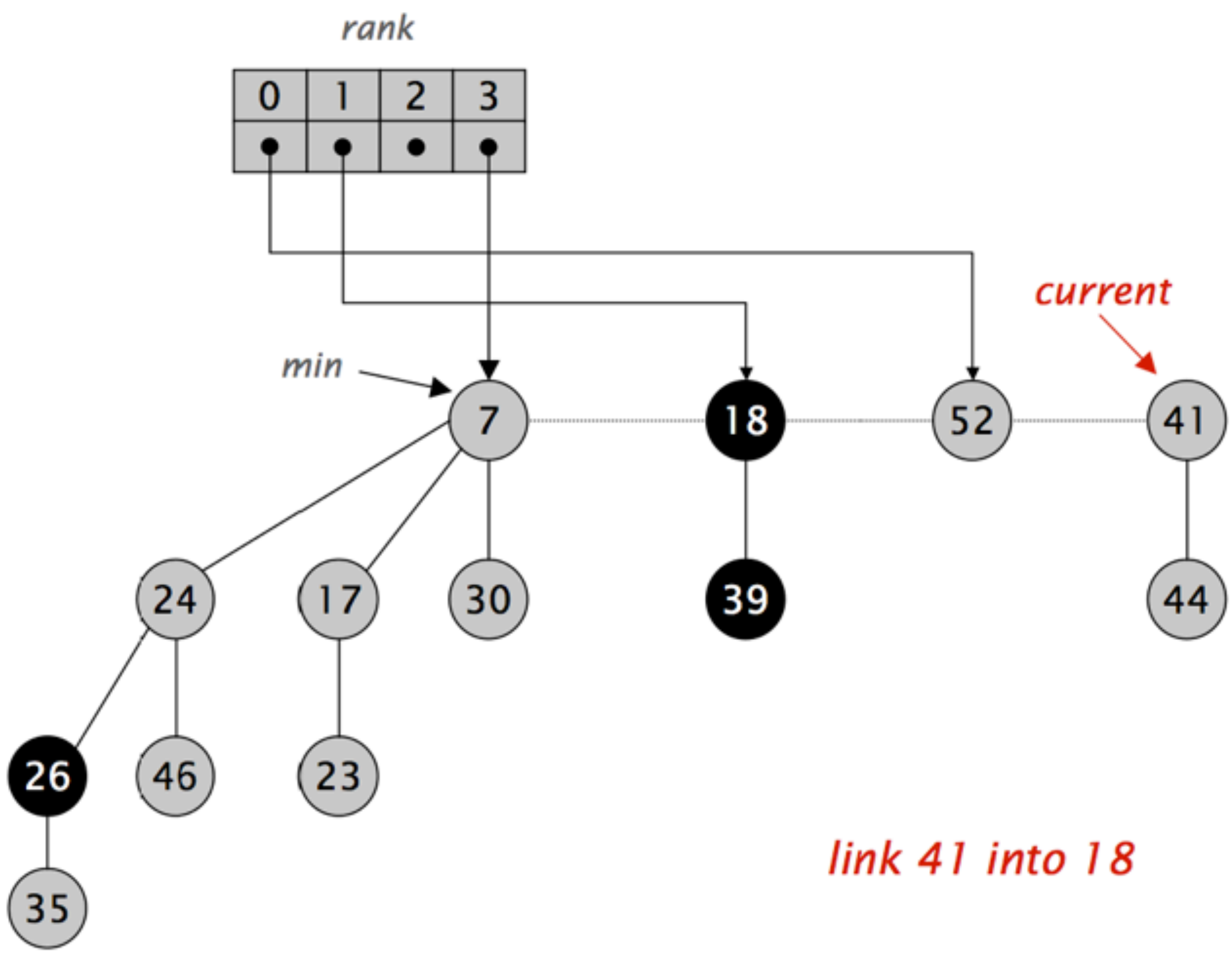
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

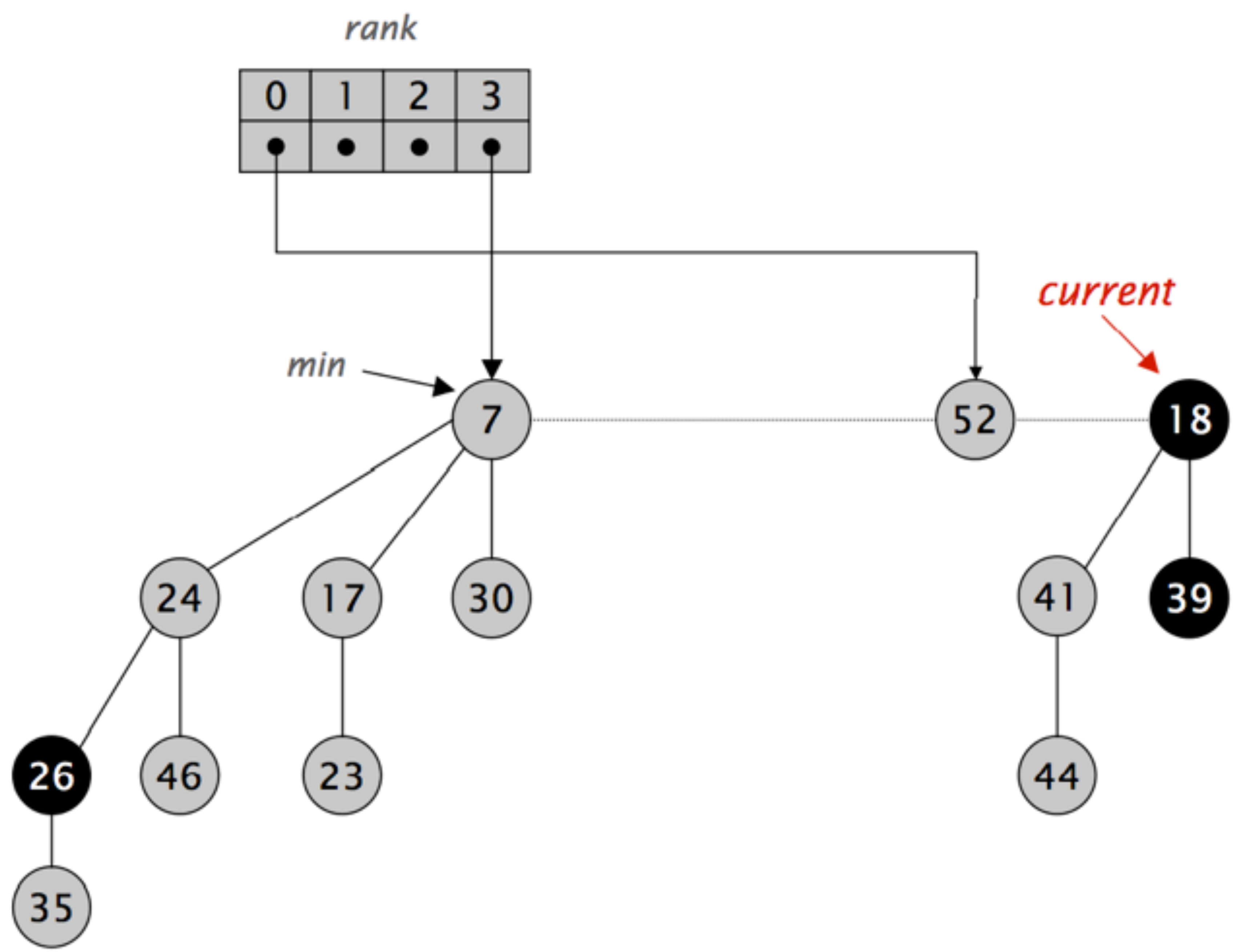
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

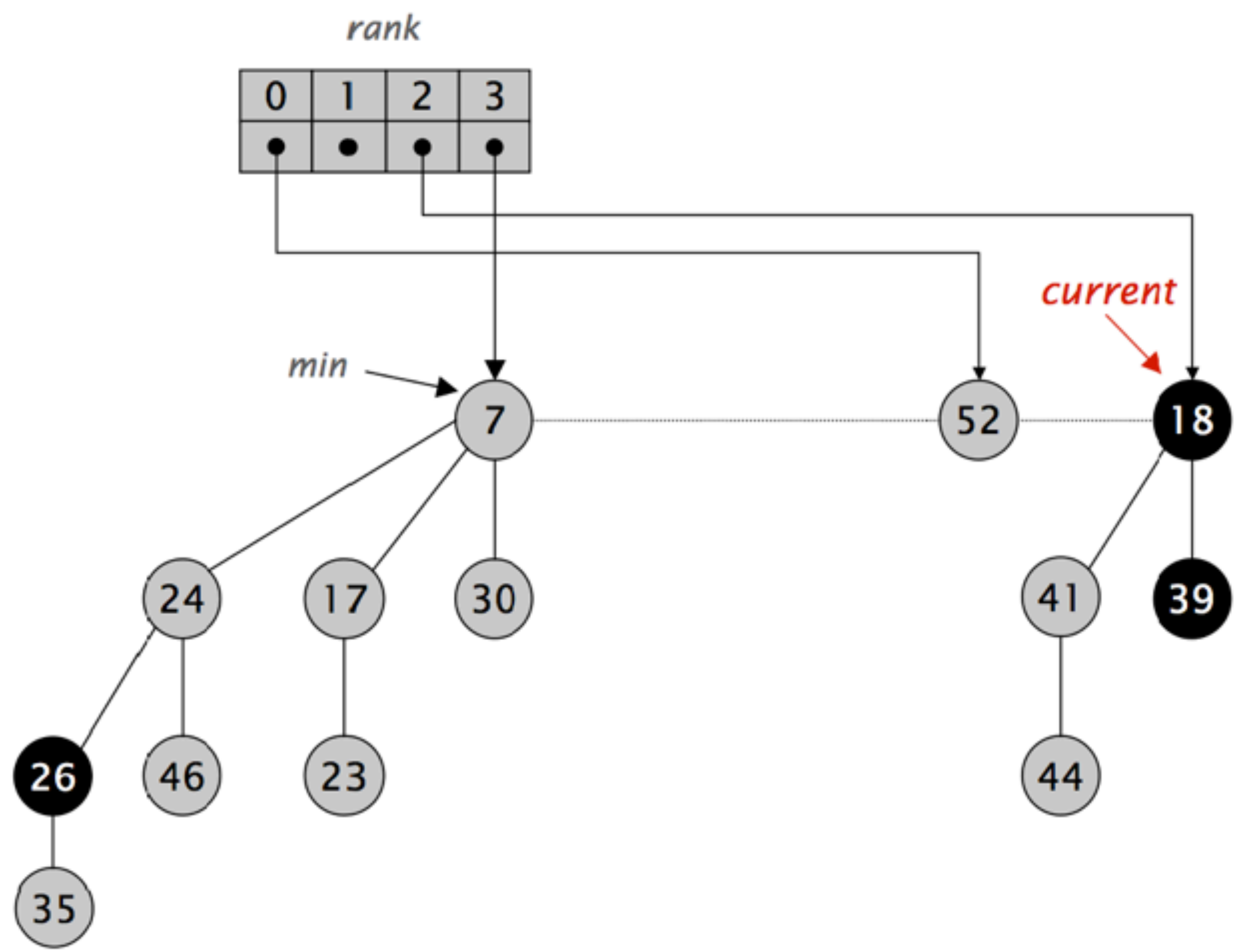
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

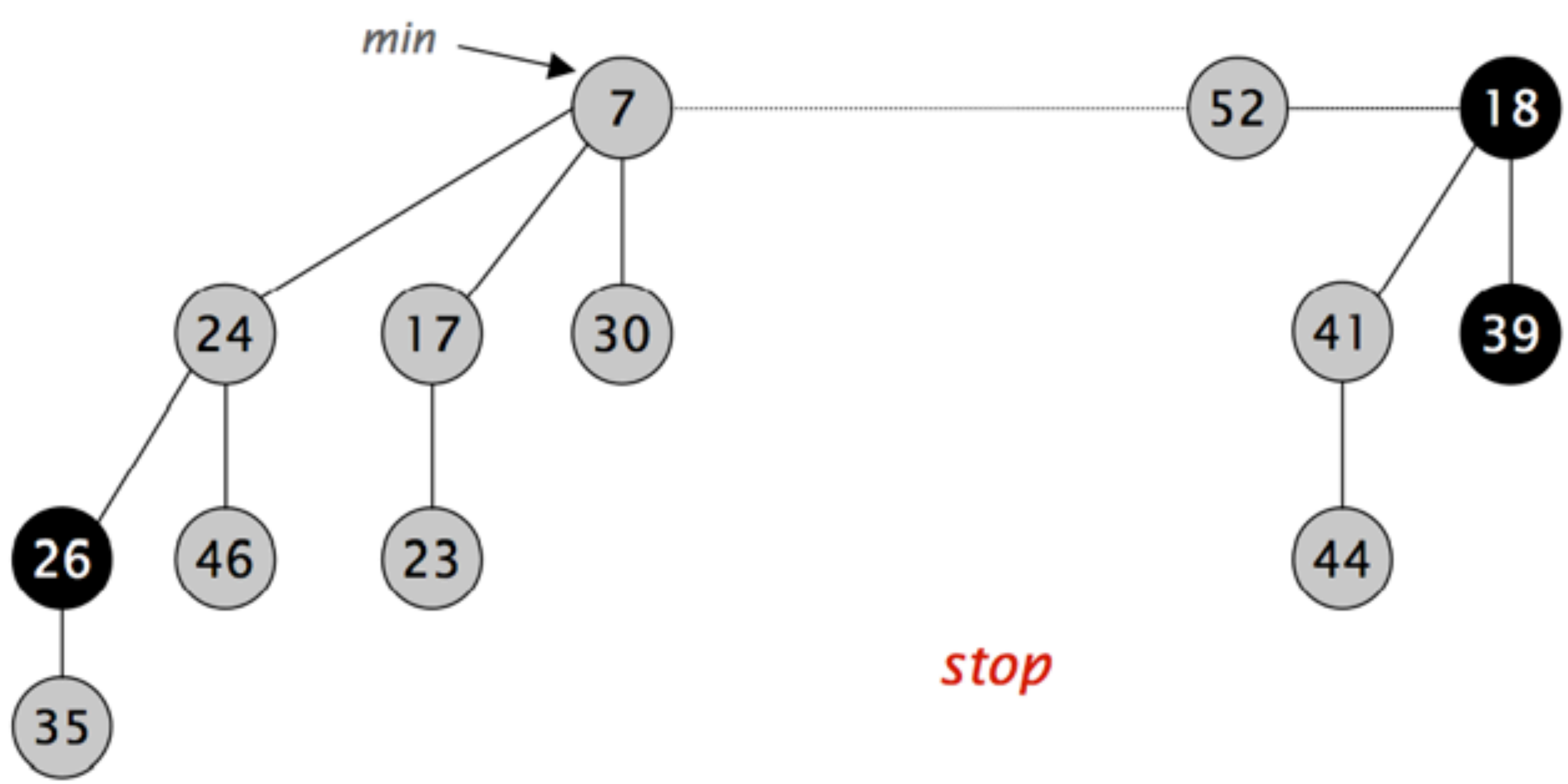
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



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 Analysis

Delete min.

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

potential function

Actual cost. $O(rank(H)) + O(trees(H))$

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

Change in potential. $O(rank(H)) - trees(H)$

- $trees(H') \leq rank(H) + 1$ since no two trees have same rank.
- $\Delta\Phi(H) \leq rank(H) + 1 - trees(H)$.

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

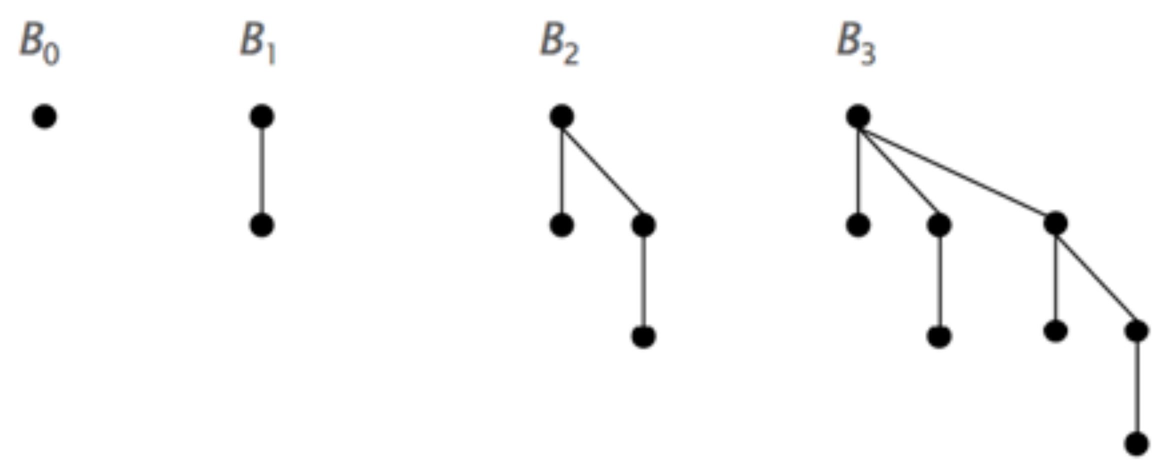
Fibonacci Heaps: Delete Min Analysis

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 $rank(H) \le \lg n$.

we only link trees of equal rank



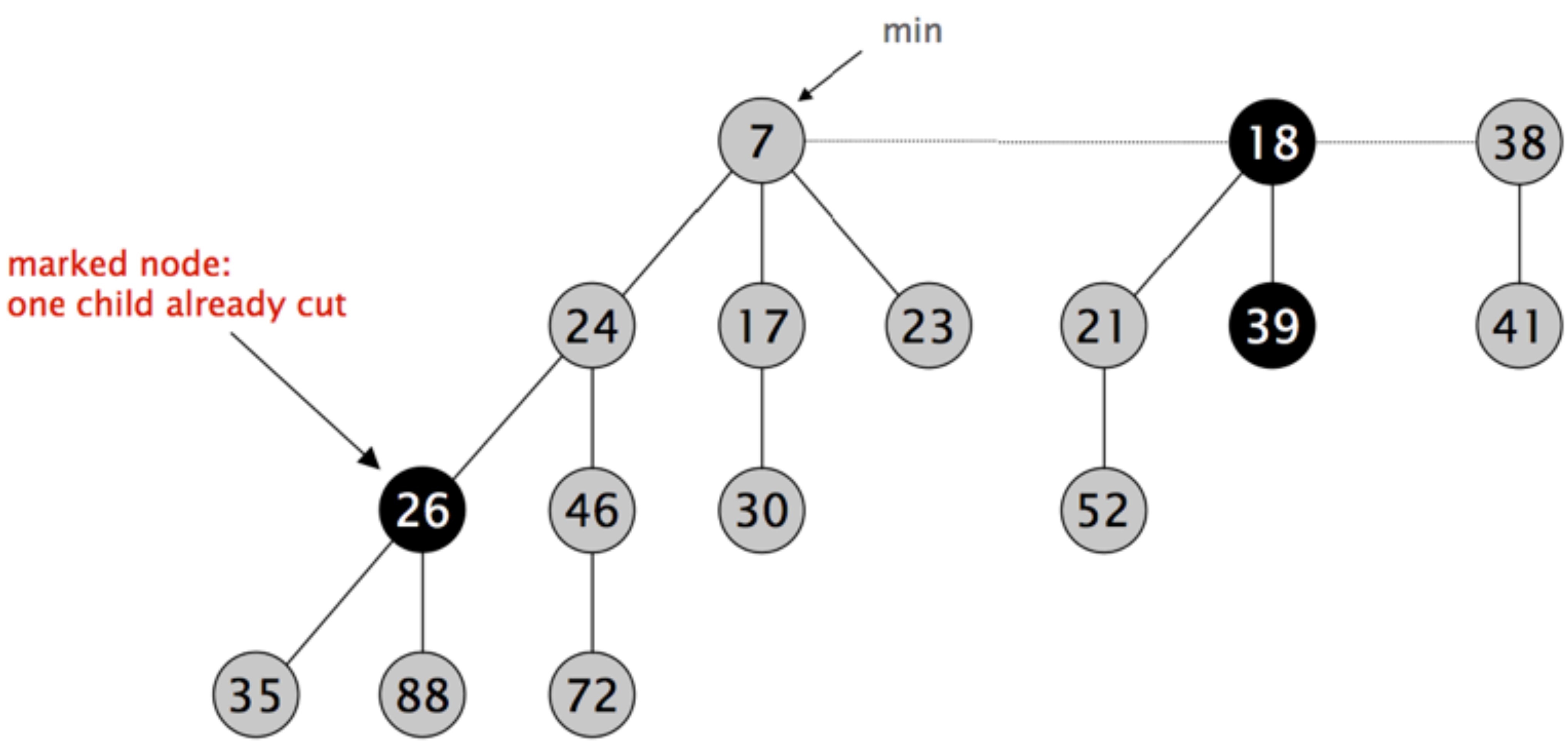
A. Yes, we'll implement *decrease-key* so that $rank(H) = O(\log n)$.

Decrease Key

Fibonacci Heaps: Decrease Key

Intuition for deceasing the key of node *x*.

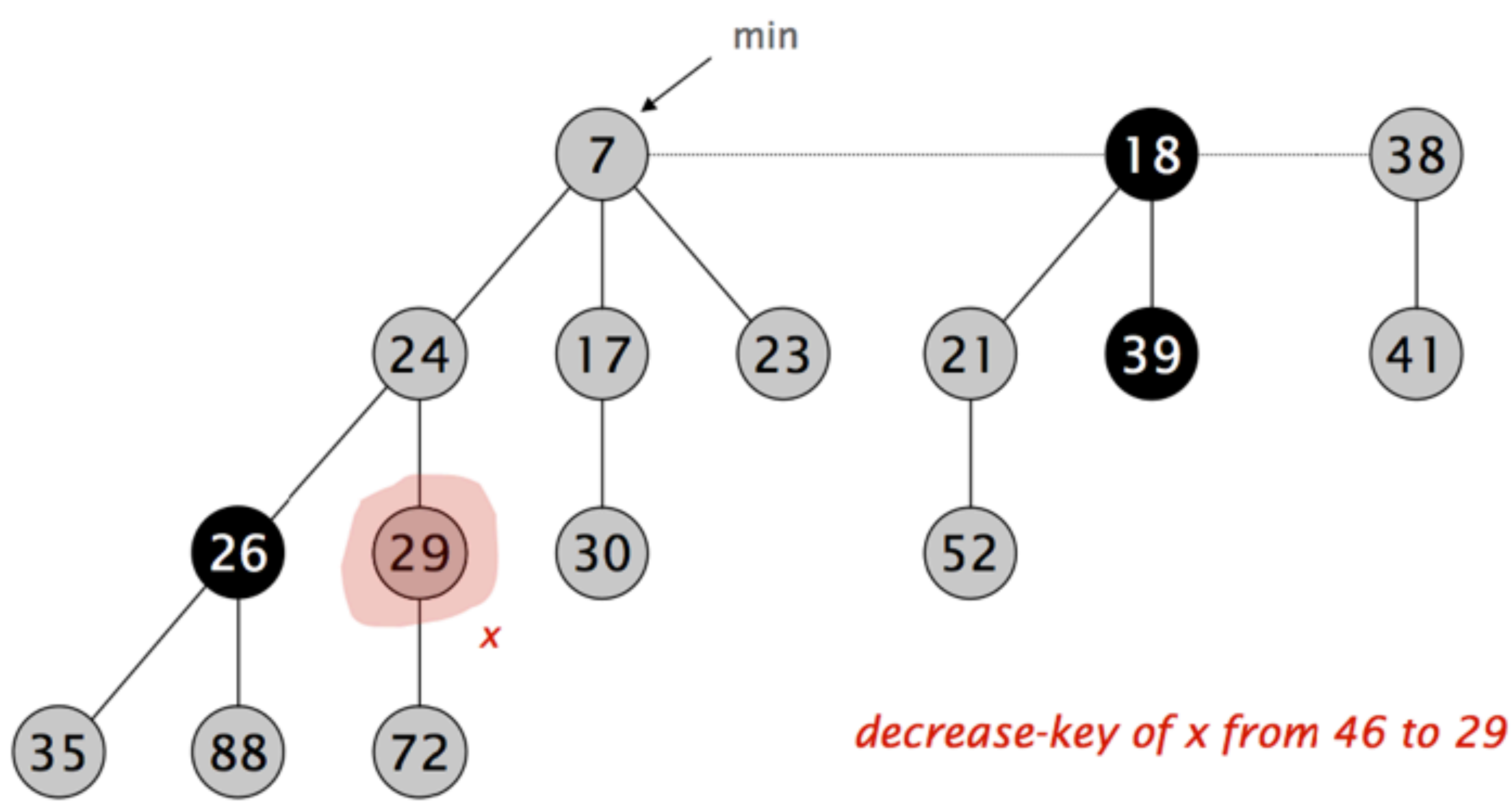
- If heap-order is not violated, just decrease the key of *x*.
- Otherwise, cut tree rooted at *x* and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

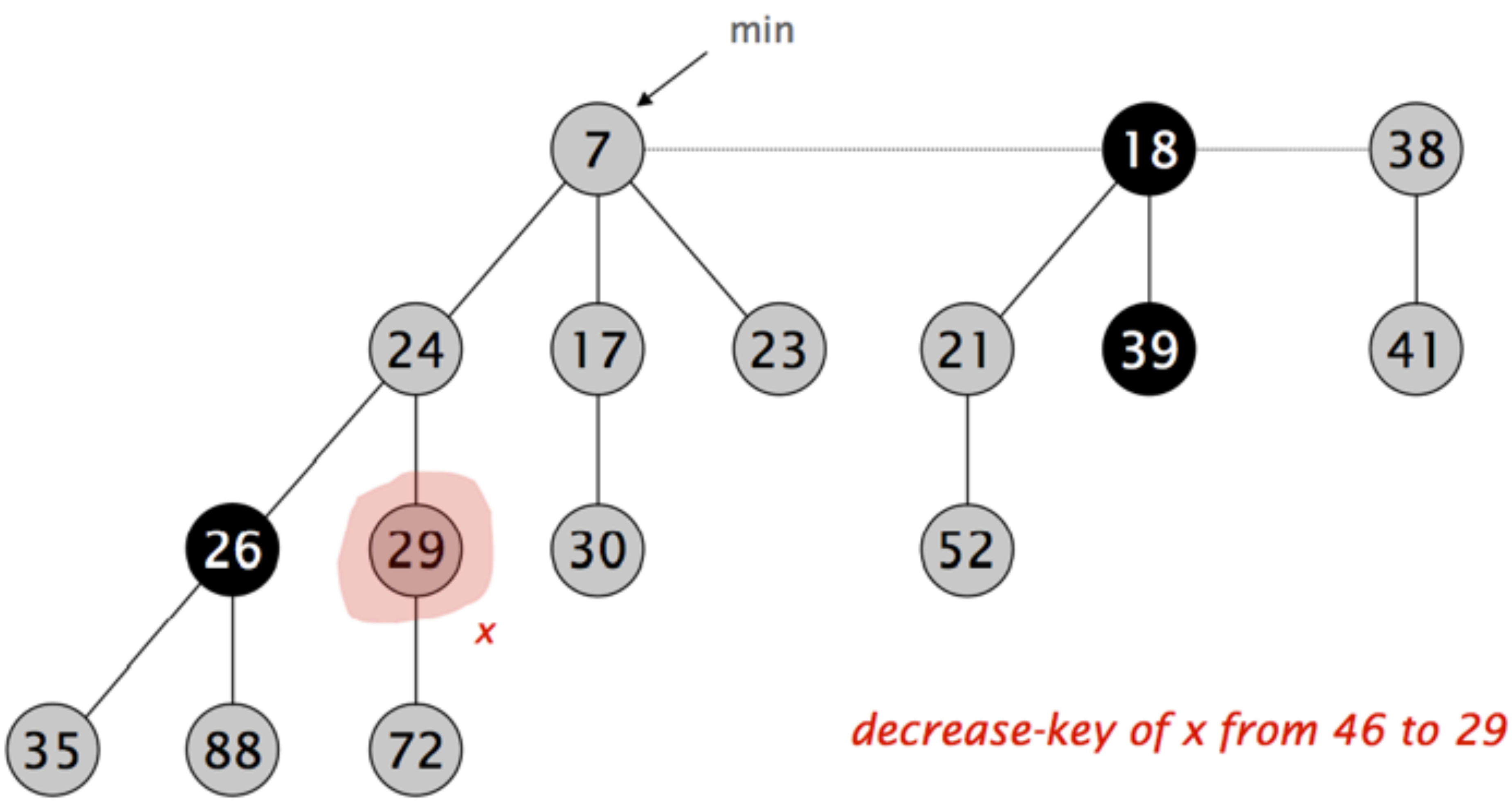
- Decrease key of *x*.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

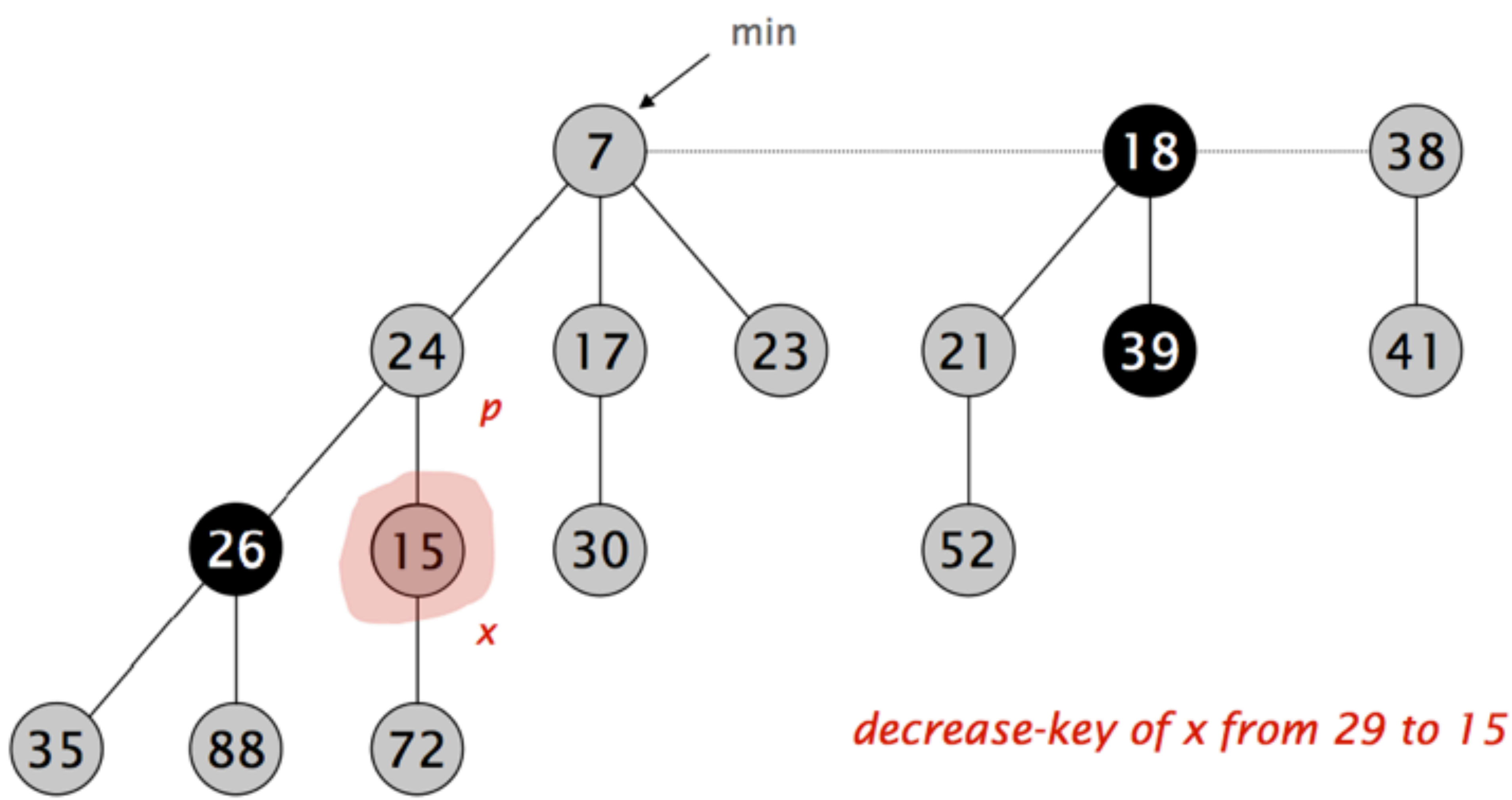
- Decrease key of *x*.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

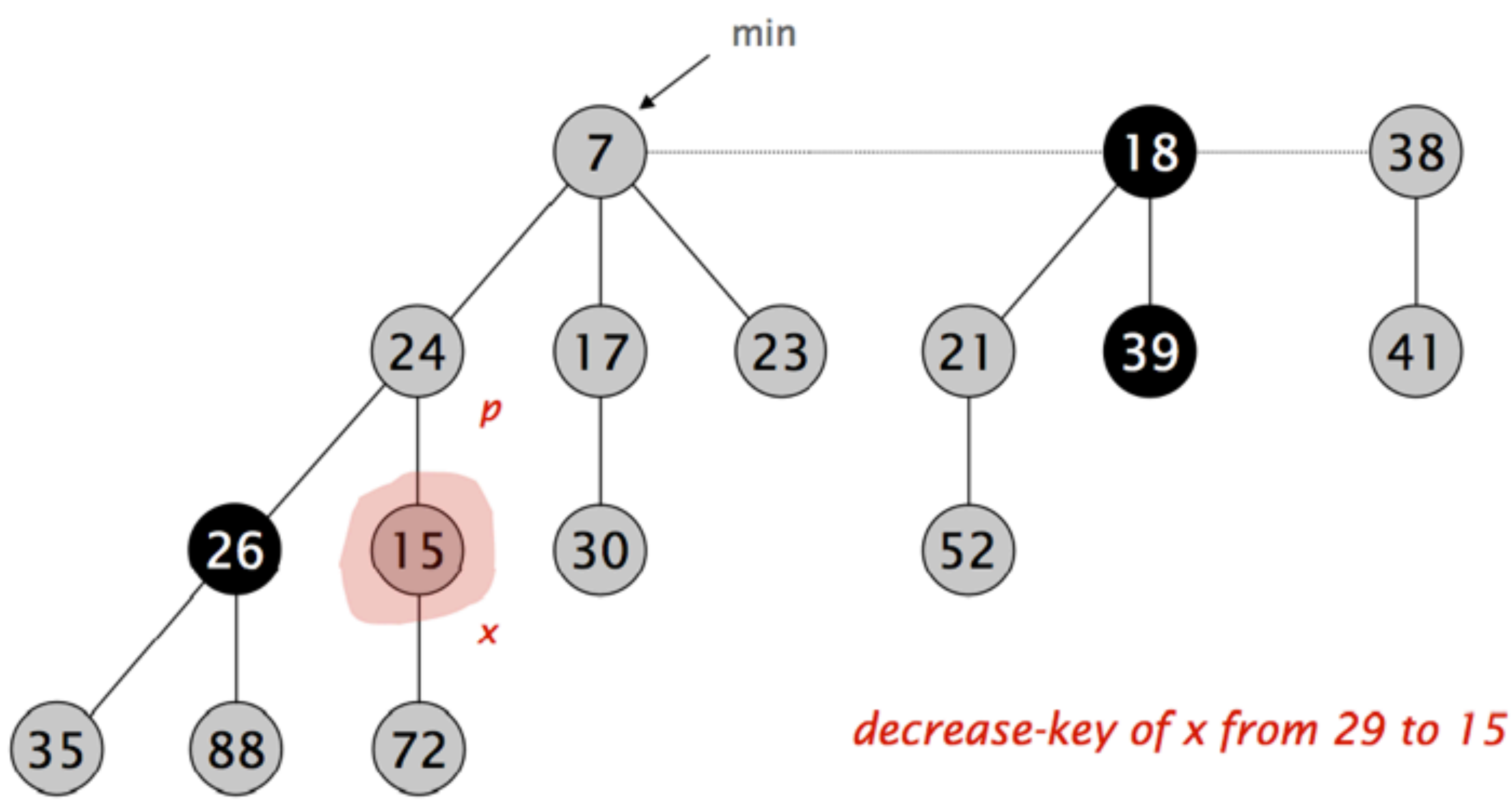
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

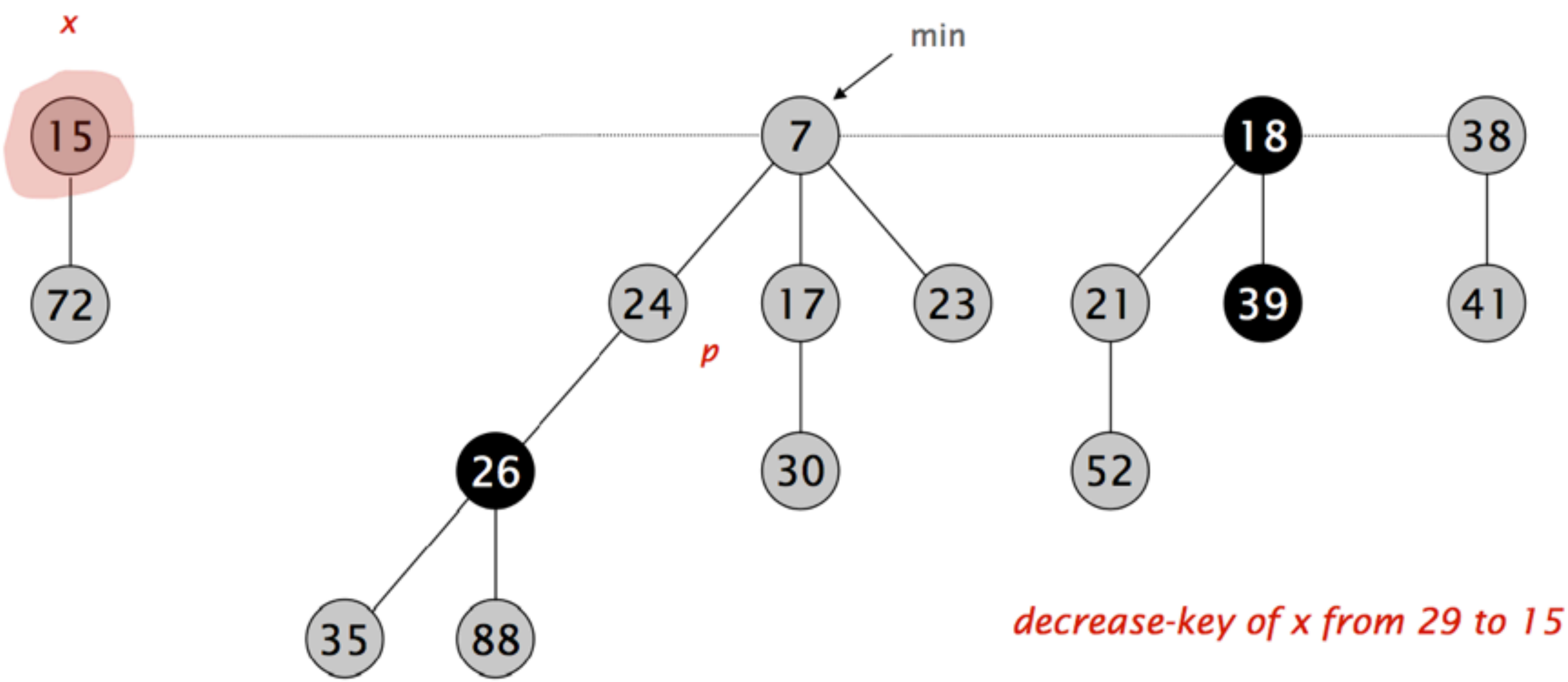
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it; Otherwise, cut *p*, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

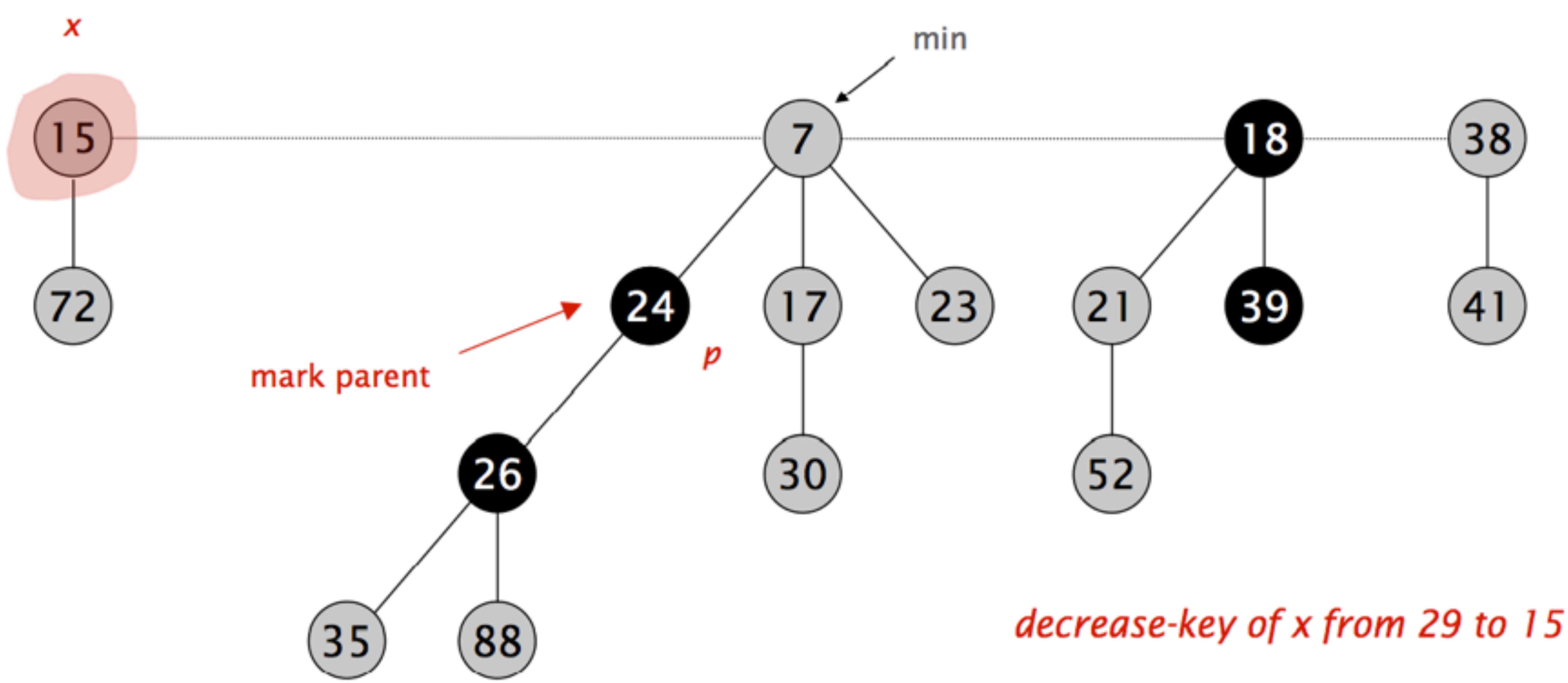
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it; Otherwise, cut *p*, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

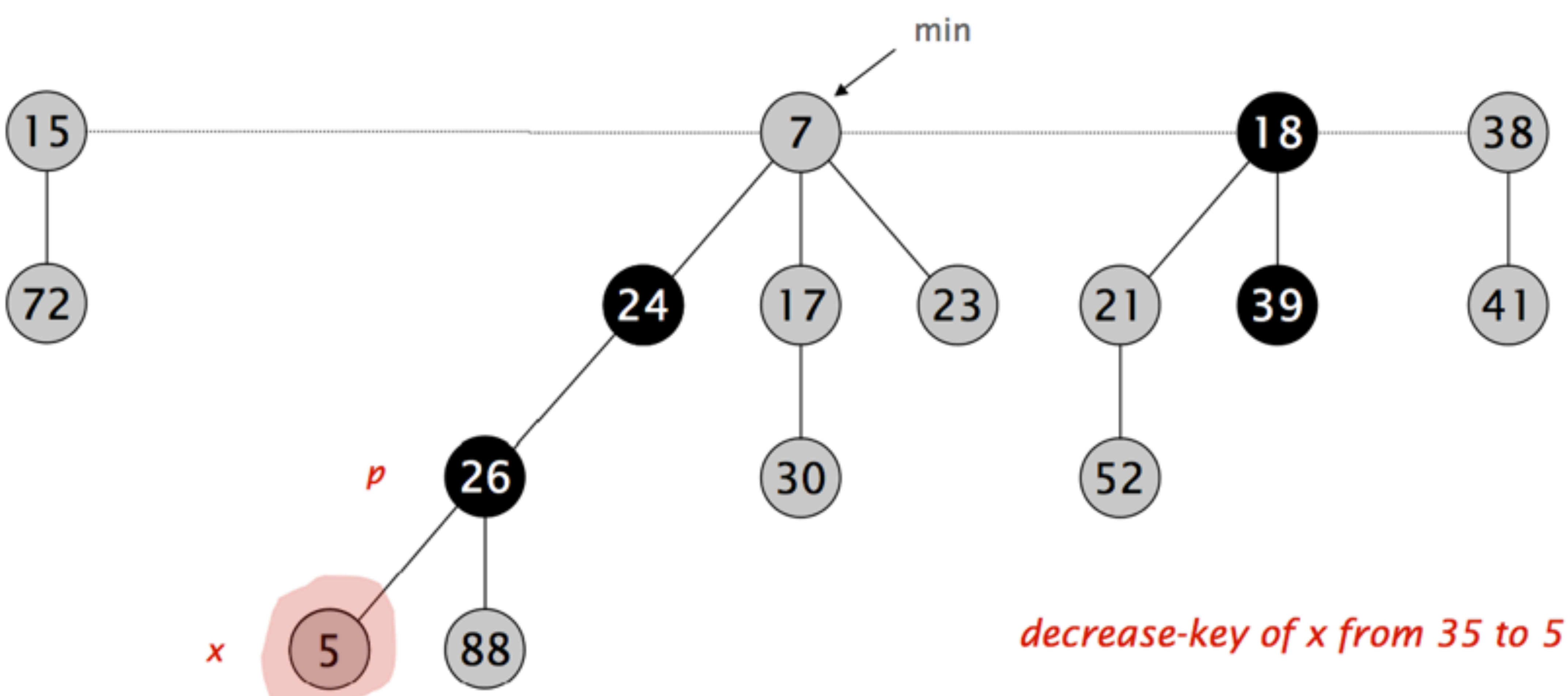
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

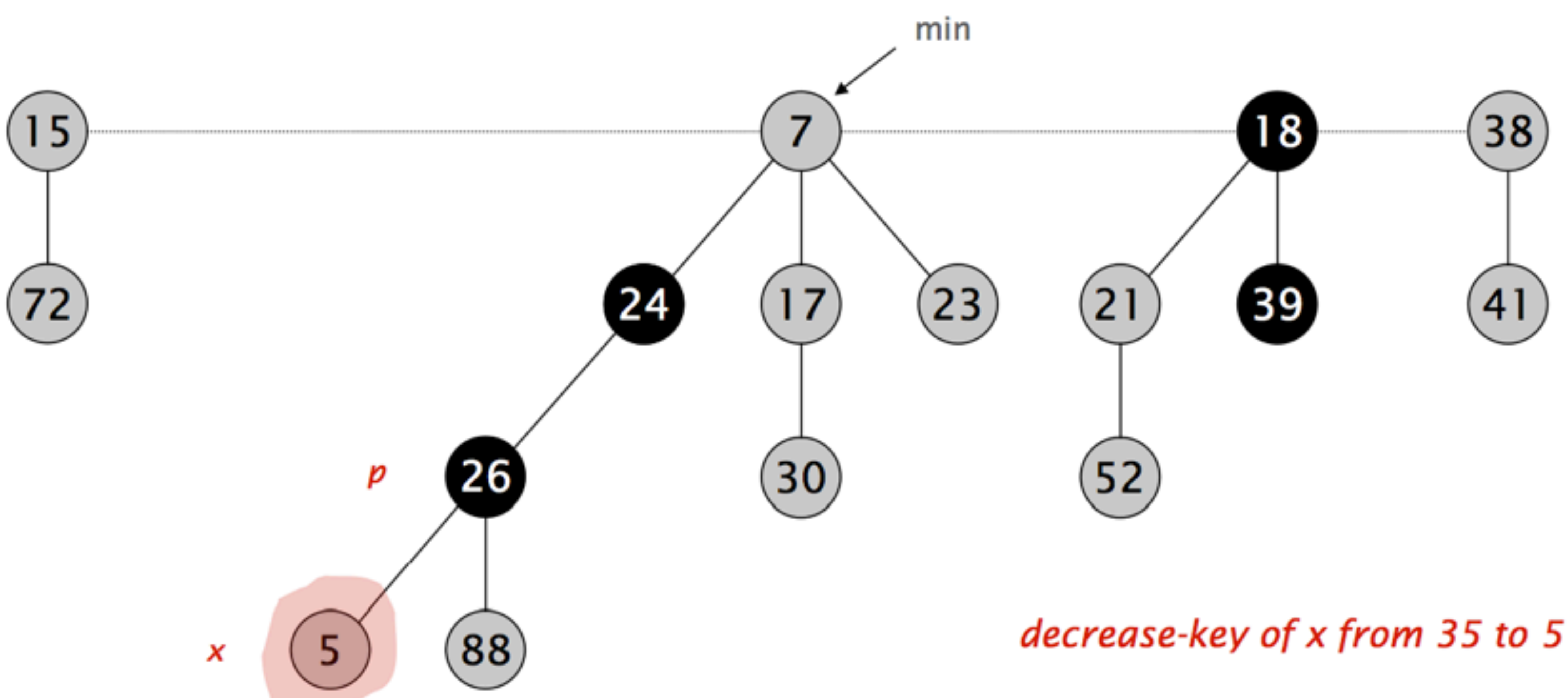
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it; Otherwise, cut *p*, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

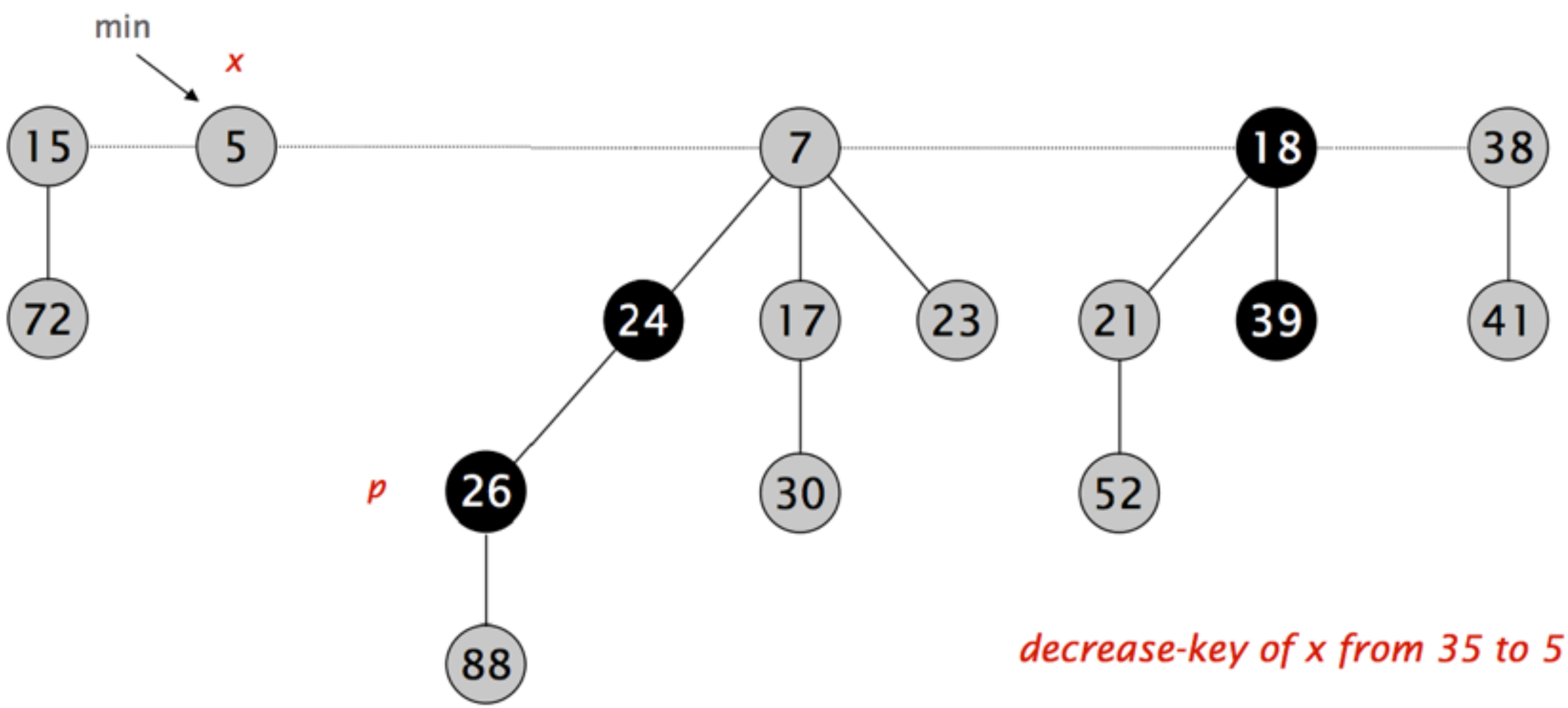
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



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).



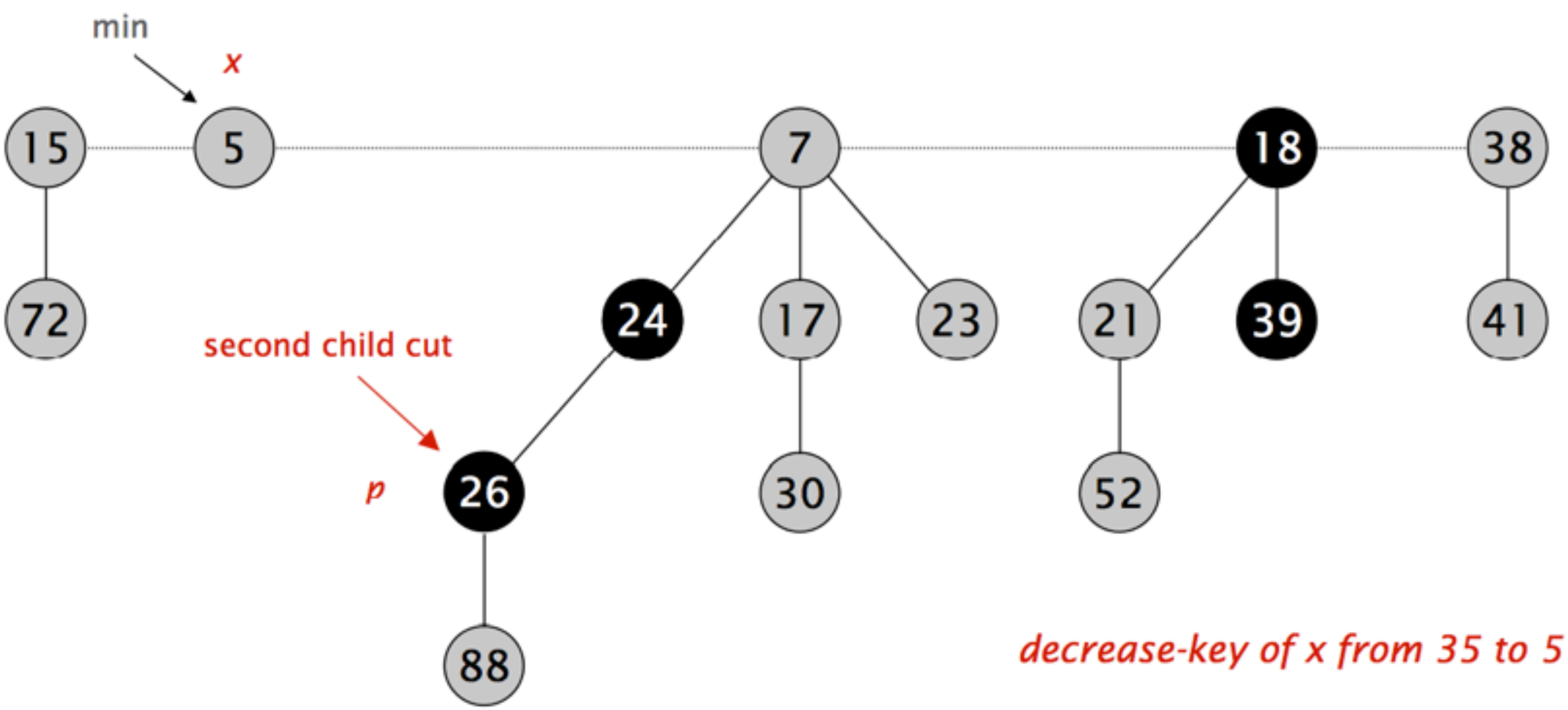
Fibonacci Heaps: Decrease Key

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

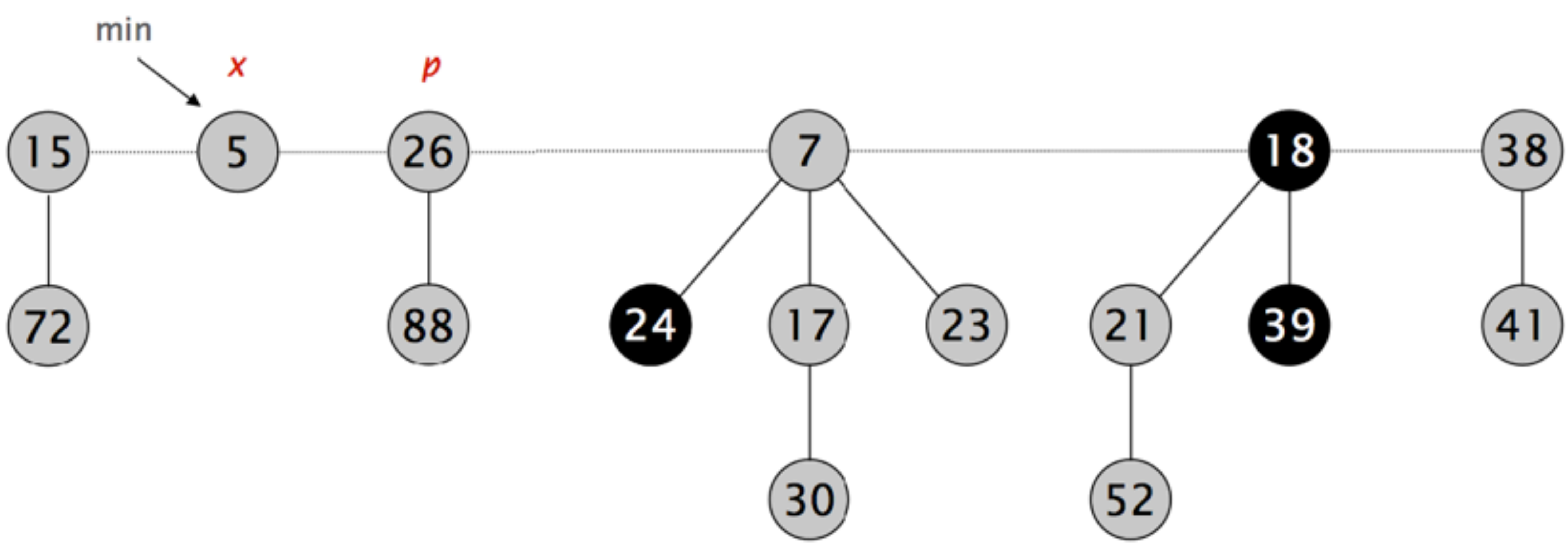
  (and do so recursively for all ancestors that lose a second child).



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).

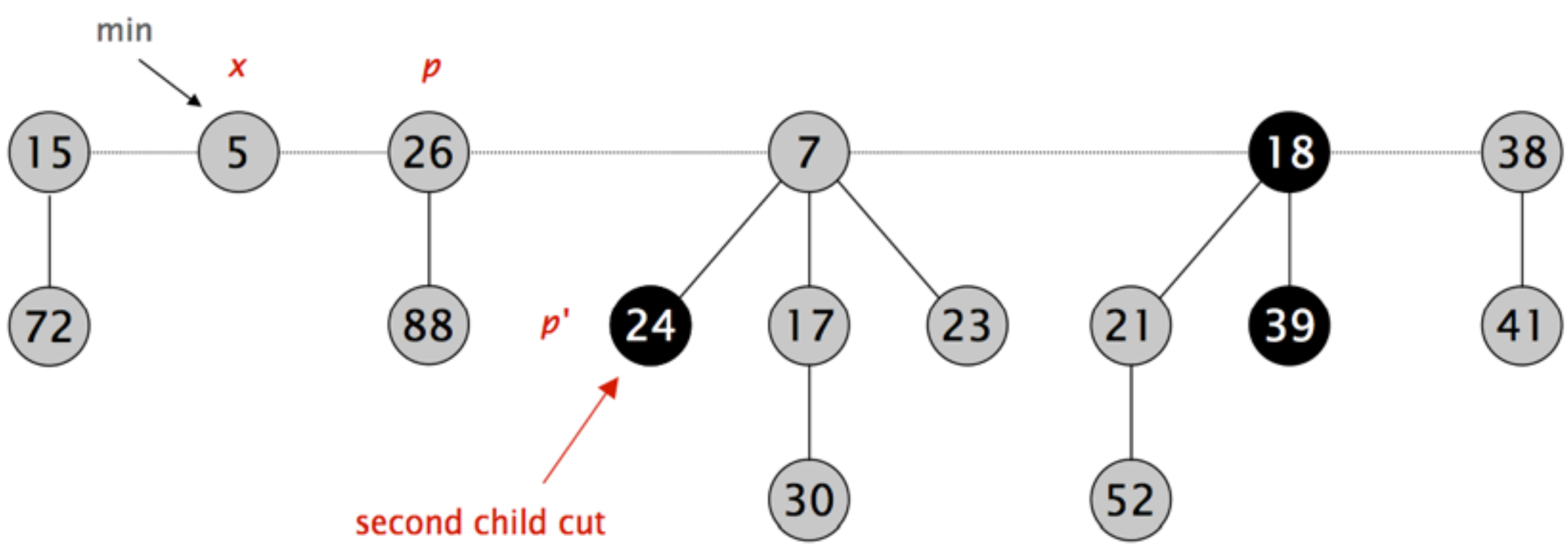


decrease-key of x from 35 to 5

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).

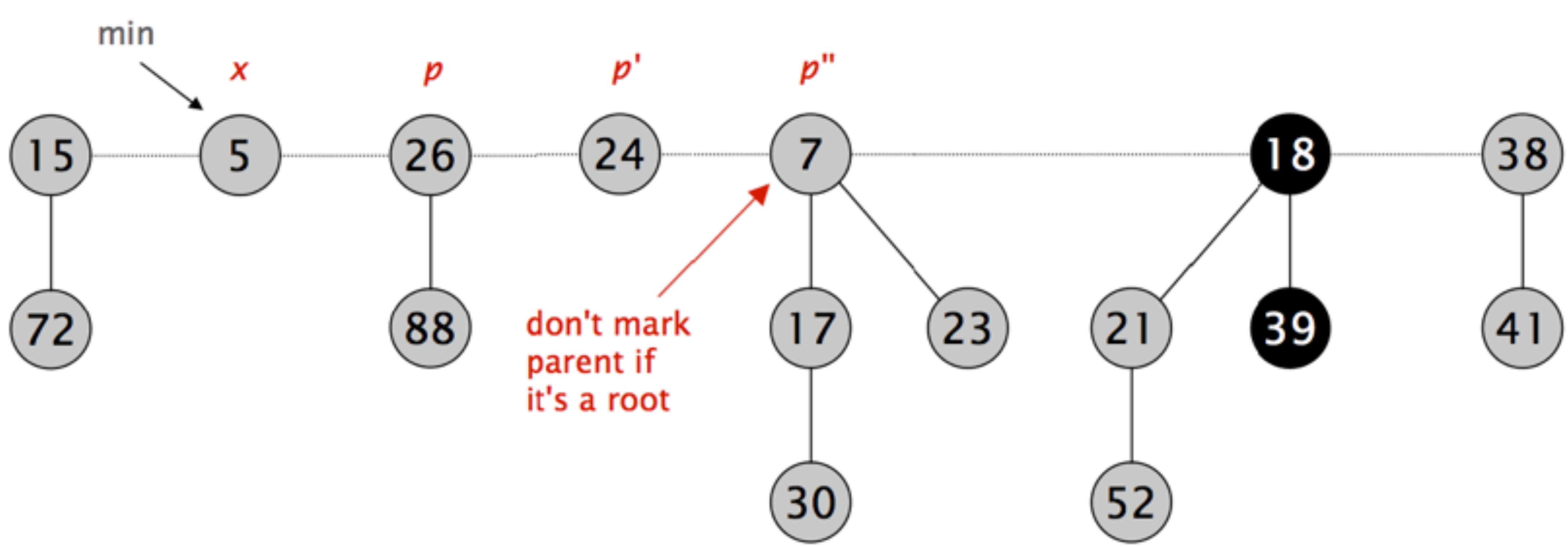


decrease-key of x from 35 to 5

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

Fibonacci Heaps: Decrease Key Analysis

Decrease-key.

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

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. $O(1) - c$

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

Amortized cost. $O(1)$

Analysis

Analysis Summary

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

† amortized

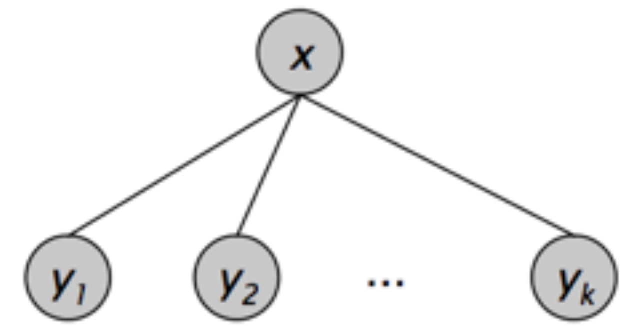
Key lemma. $rank(H) = O(\log n)$.

↖ 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.

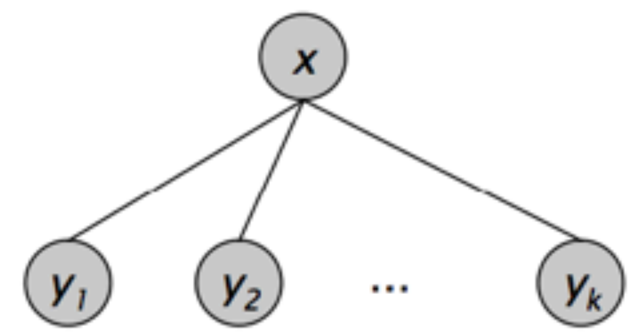
- When y_i was linked into x, x had at least i-1 children $y_1, \ldots, y_{i-1}$.
- Since only trees of equal rank are linked, at that time $rank(y_i) = rank(x_i) \geq i - 1$.
- Since then, y_i has lost at most one child.
- Thus, right now $rank(y_i) \geq i - 2$. ∎

or y_i would have been cut

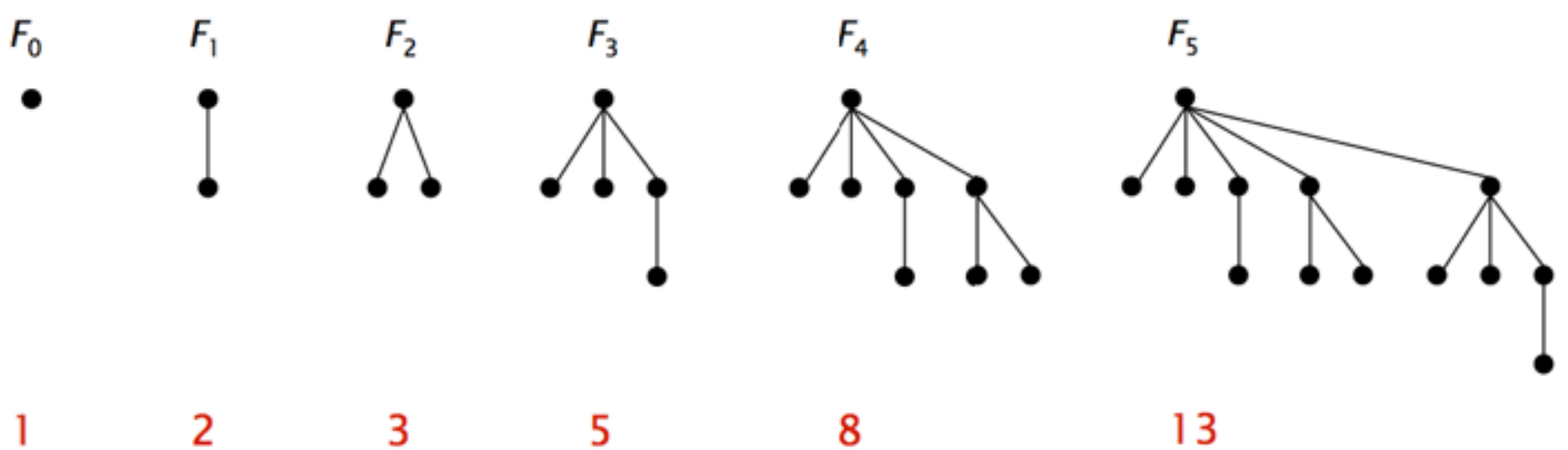
Fibonacci Heaps: Bounding the Rank

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:

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



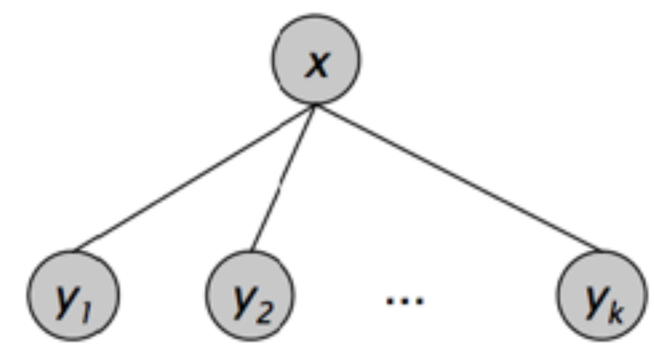
Def. Let F_k be smallest possible tree of rank k satisfying property.



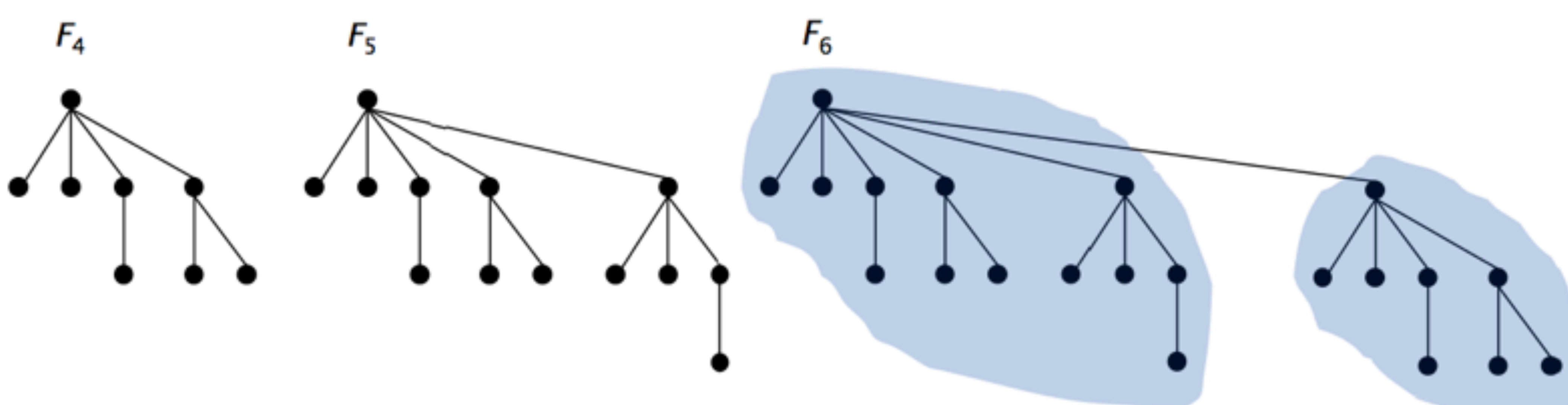
Fibonacci Heaps: Bounding the Rank

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.



8

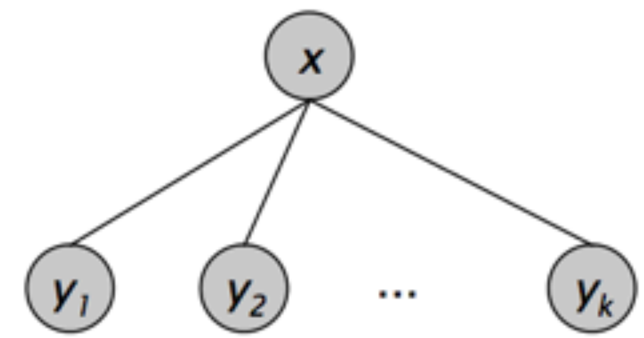
13

8 + 13 = 21

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.

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

golden ratio

Corollary. $rank(H) \leq \log_\phi n$.

Fibonacci Numbers

Fibonacci Numbers: Exponential Growth

Def. The Fibonacci sequence is: 1, 2, 3, 5, 8, 13, 21, ...

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

slightly non-standard definition

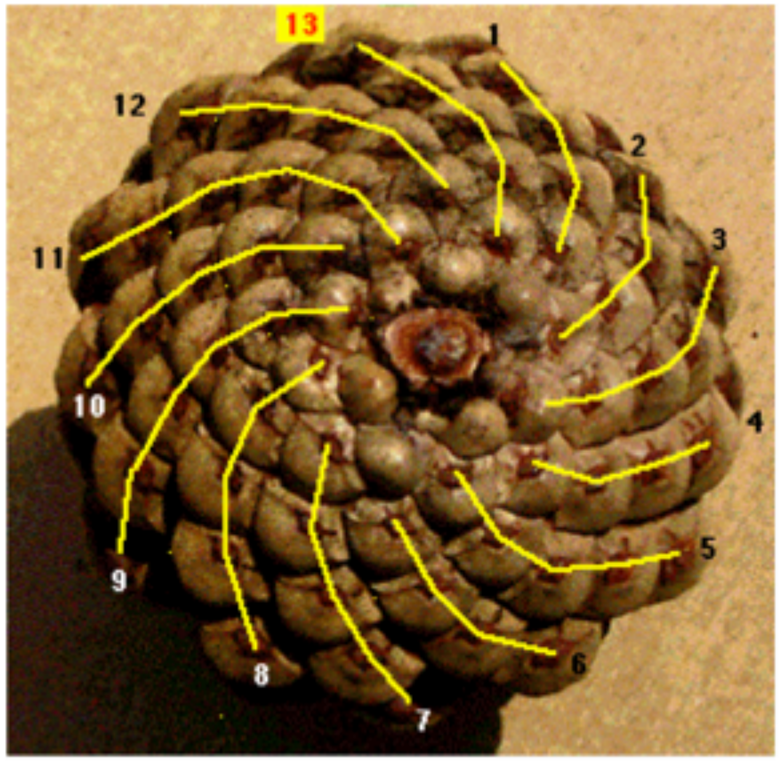
Lemma. $F_k \geq \phi^k$, where $\phi = (1 + \sqrt{5}) / 2 \approx 1.618$.

Pf. [by induction on k]

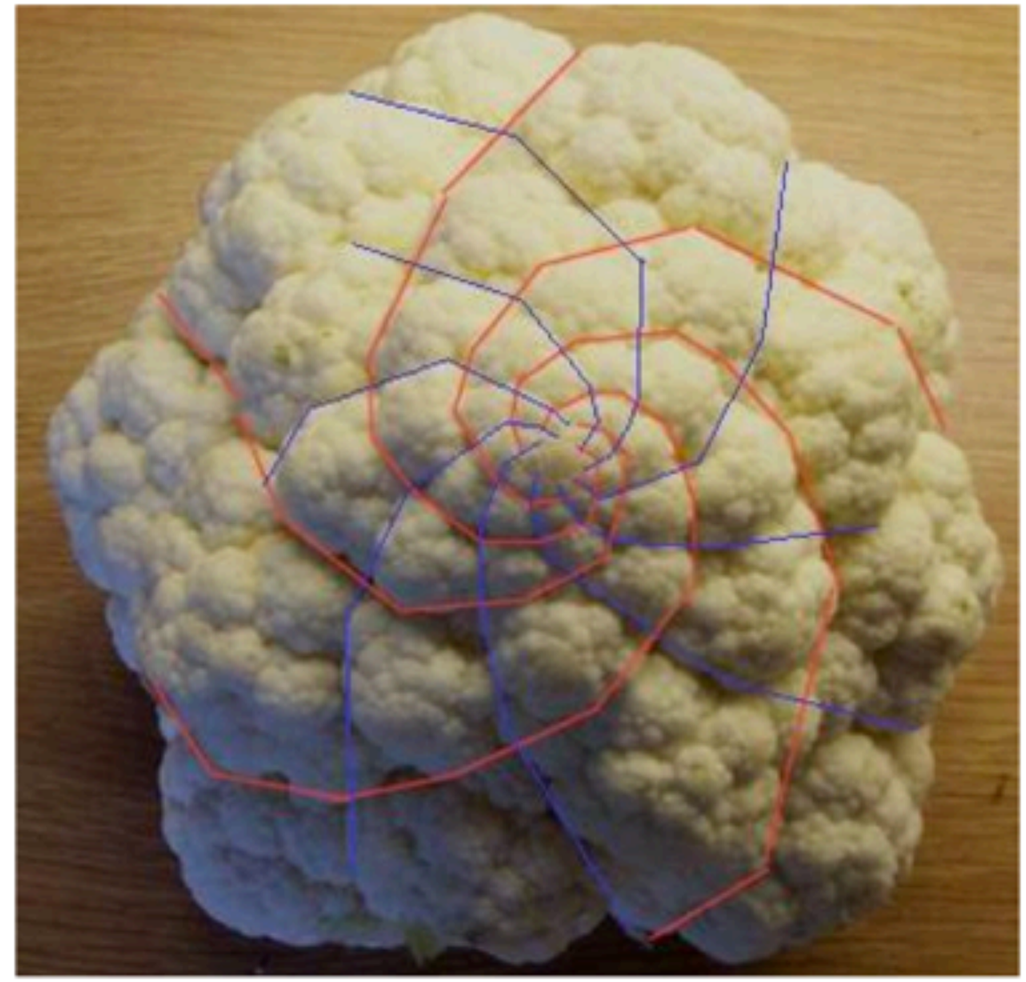
- Base cases: $F_0 = 1 \geq 1$, $F_1 = 2 \geq \phi$.
- Inductive hypotheses: $F_k \geq \phi^k$ and $F_{k+1} \geq \phi^{k+1}$

$F_{k+2} = F_k + F_{k+1}$		(definition)
$\geq \phi^k + \phi^{k+1}$		(inductive hypothesis)
$= \phi^k (1 + \phi)$		(algebra)
$= \phi^k (\phi^2)$		($\phi^2 = \phi + 1$)
$= \phi^{k+2}$		(algebra)

Fibonacci Numbers and Nature



pinecone



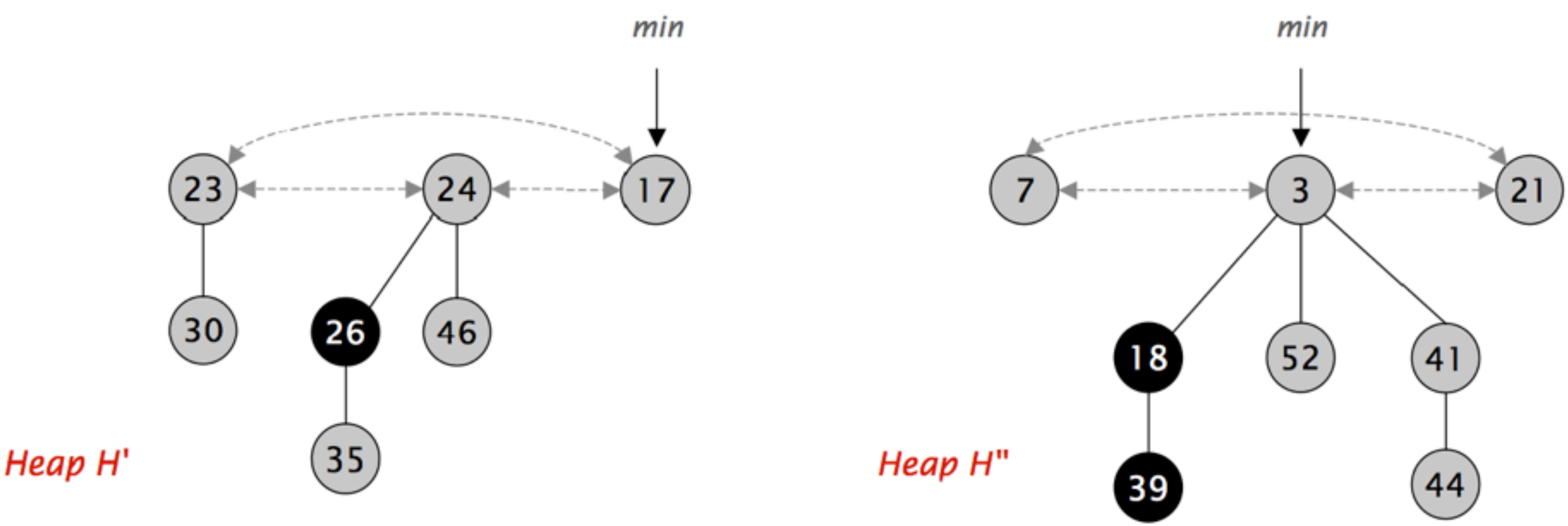
cauliflower

Union

Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

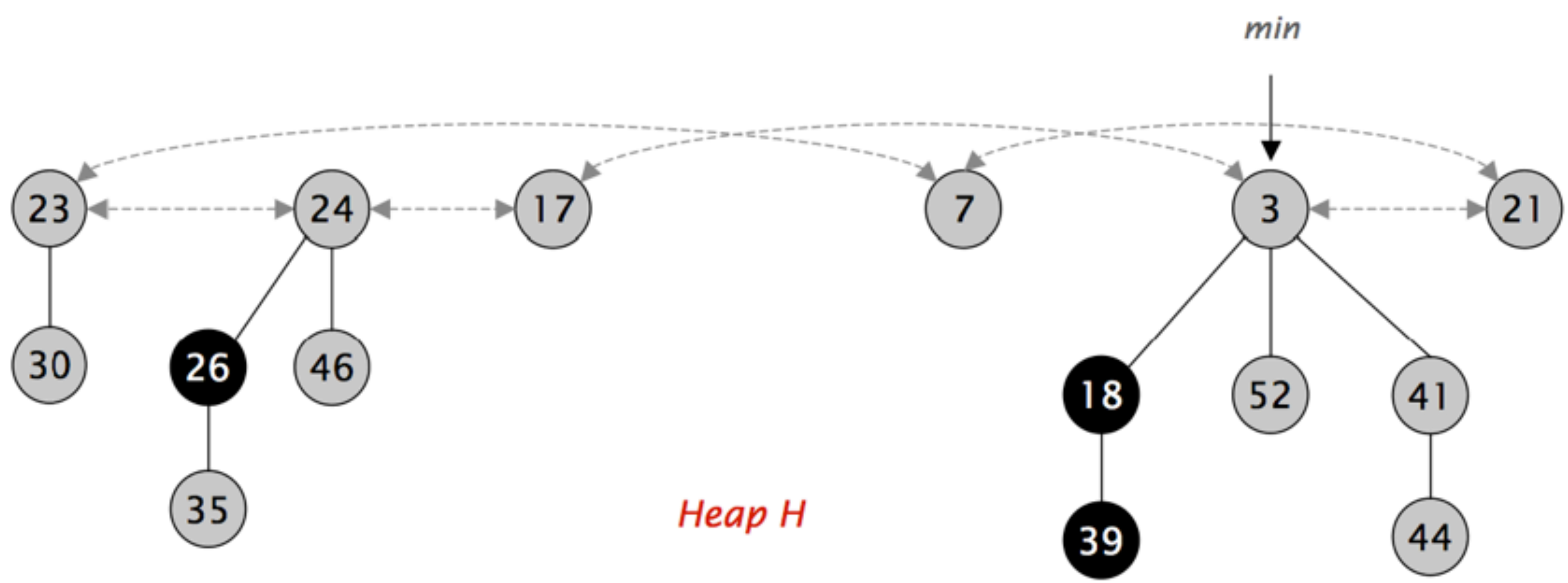
Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

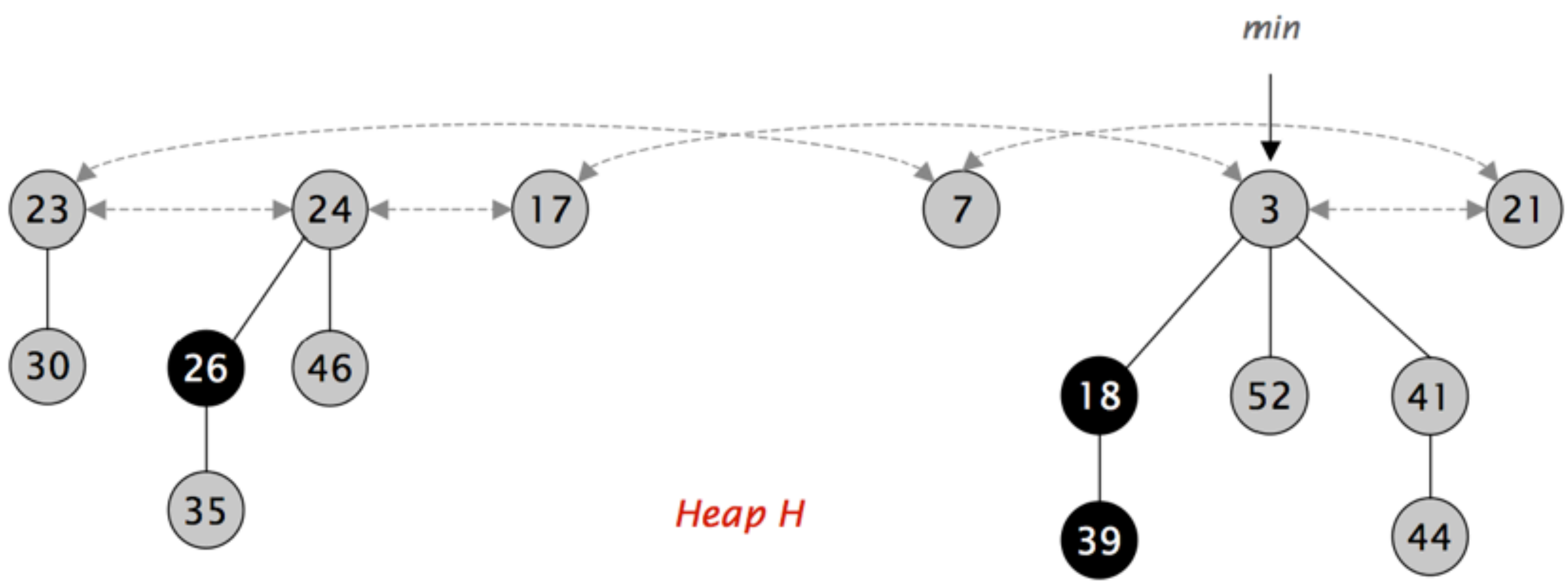
Actual cost. $O(1)$

Change in potential. 0

Amortized cost. $O(1)$

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

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*.

On Complicated Algorithms

"Once you succeed in writing the programs for [these] complicated algorithms, they usually run extremely fast. The computer doesn't need to understand the algorithm, its task is only to run the programs."

R. E. Tarjan