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 Koze

COS 423 Theory of Algorithms · Kevin Wayne · Spring 2007

Adapted by Cheng Li and Virgil Pavlu

Fibonacci Heaps

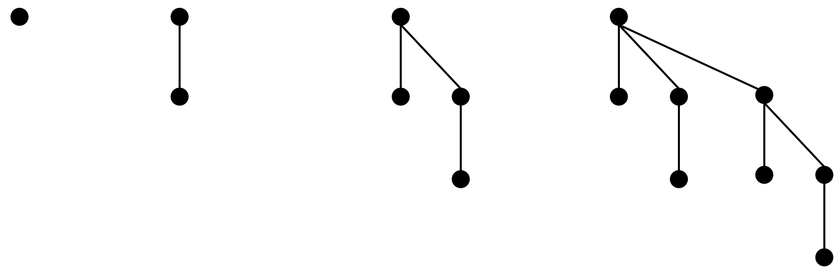
History. [Fredman and Tarjan, 1986]

- Ingenious data structure and analysis.
- Original motivation: improve Dijkstra's shortest path algorithm (module 12) from $O(E \log V)$ to $O(E + V \log V)$

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

Basic idea.

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



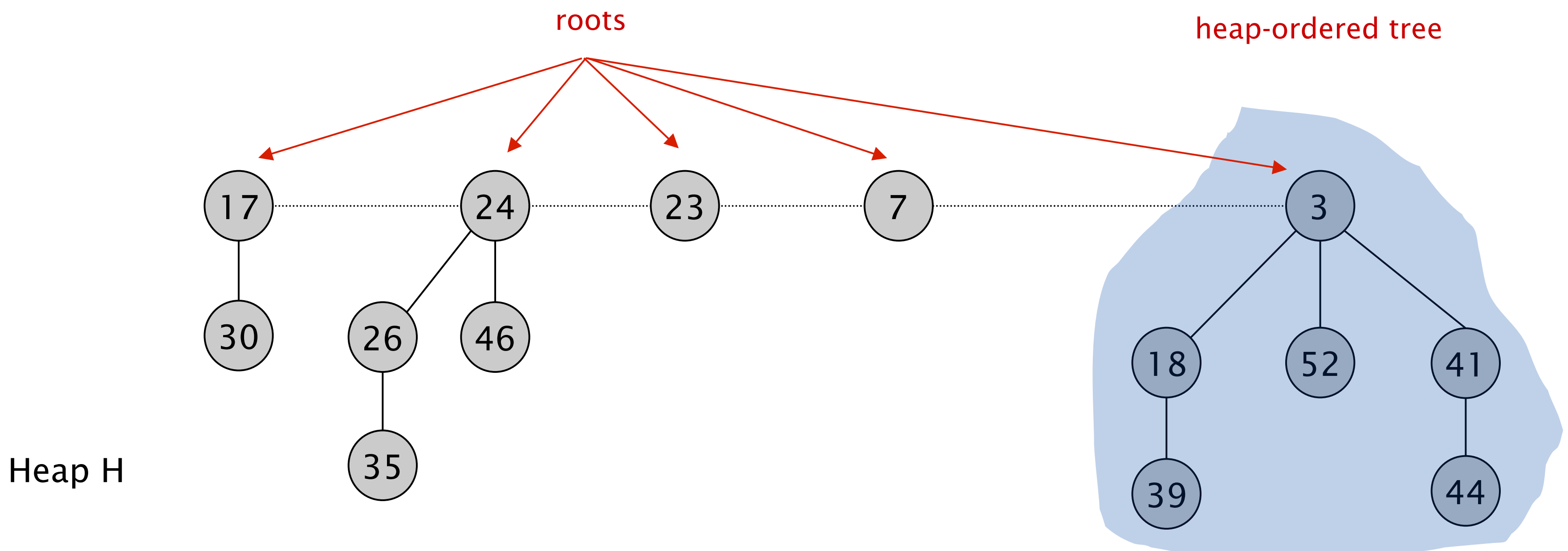
- Fibonacci heap: lazily defer consolidation until next extract-min.

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.

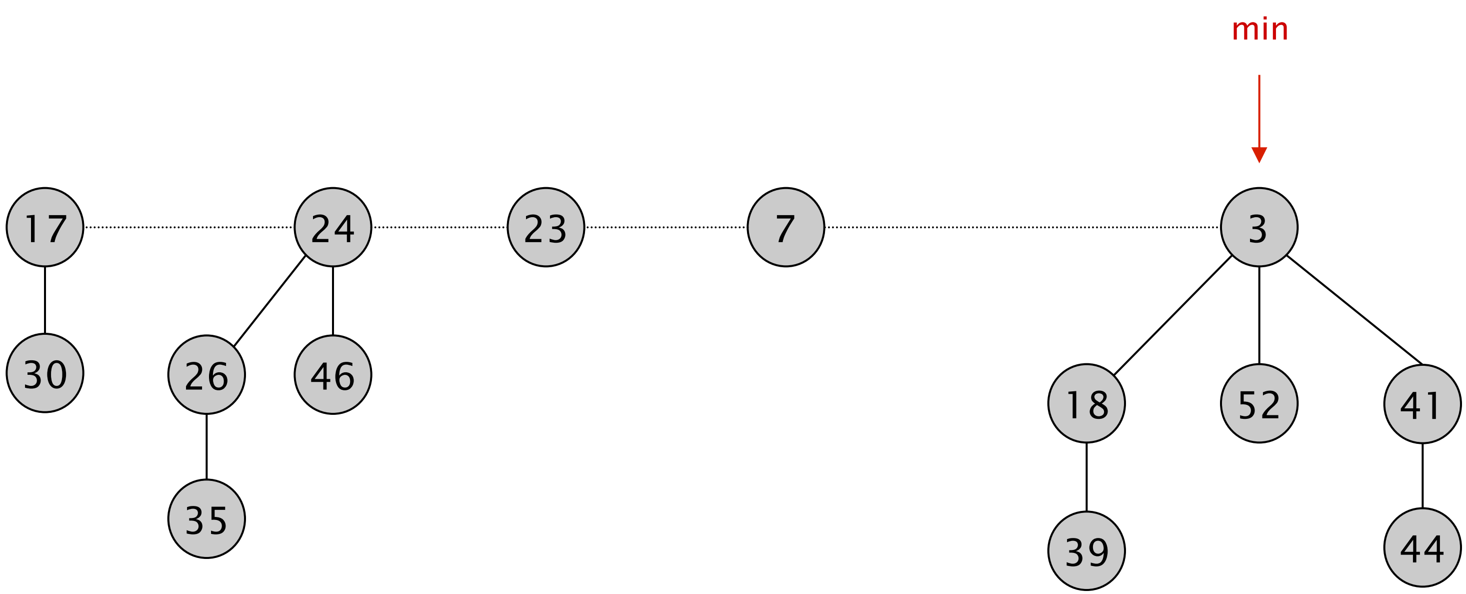


Fibonacci Heaps: Structure

Fibonacci heap.

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

find-min takes O(1) time



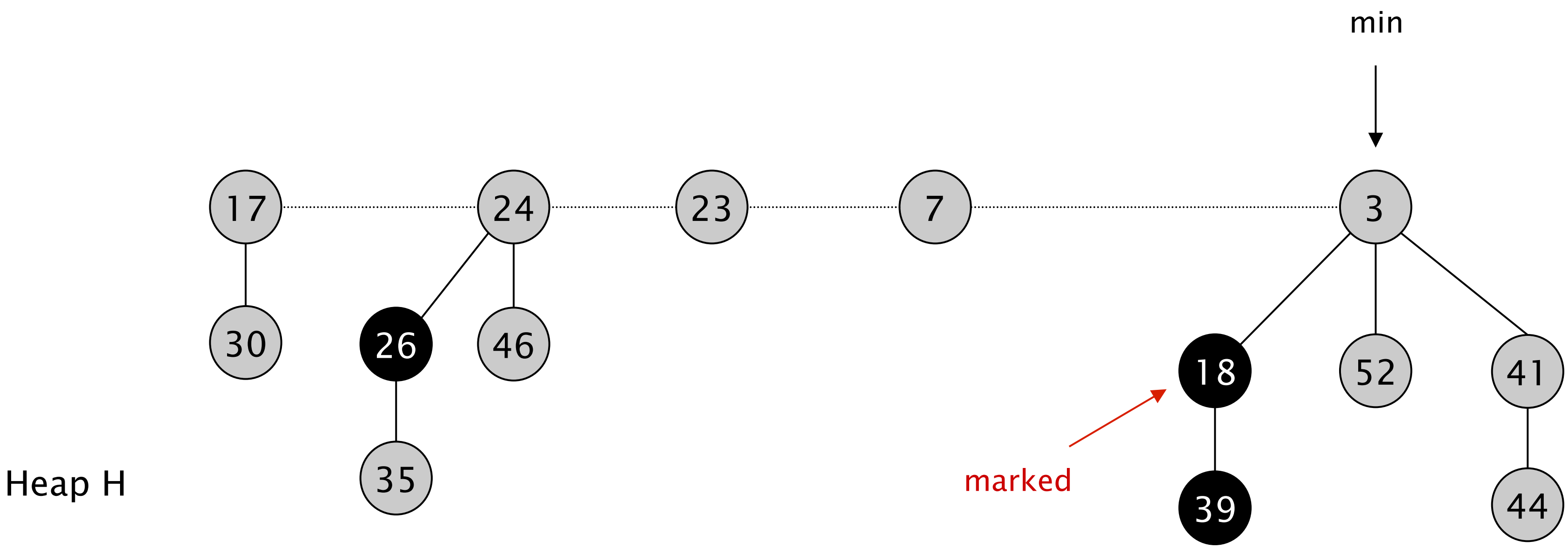
Heap H

Fibonacci Heaps: Structure

Fibonacci heap.

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

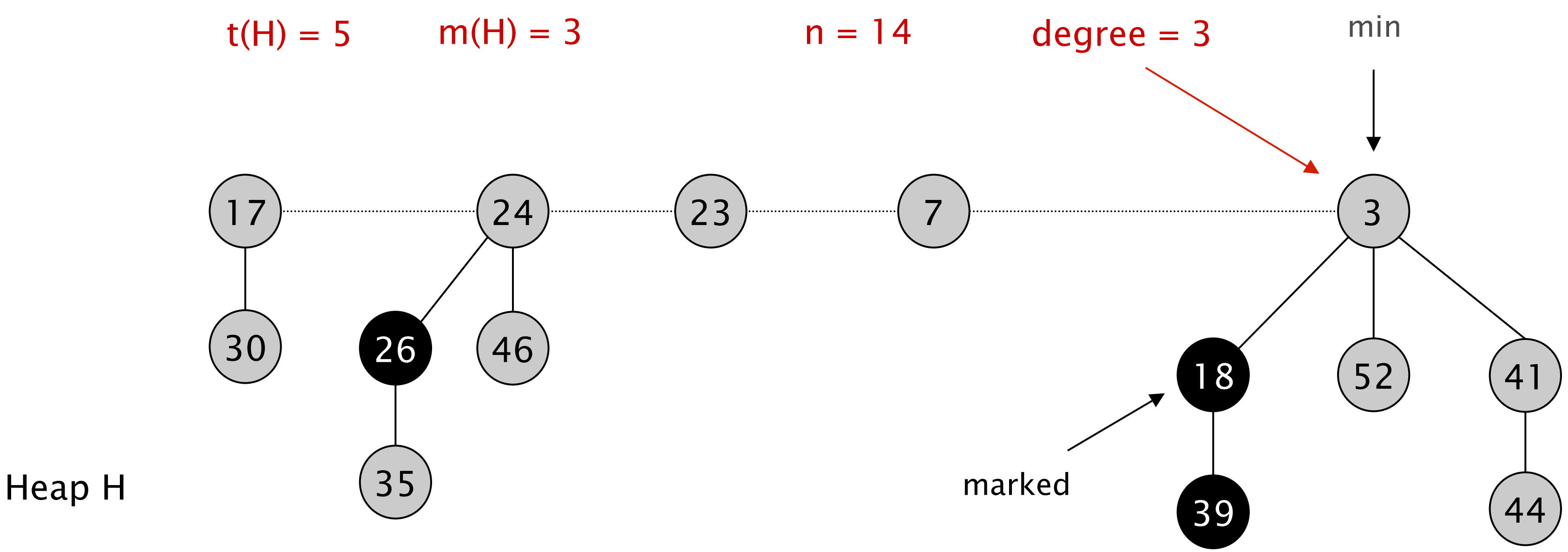
use to keep heaps flat (stay tuned)



Fibonacci Heaps: Notation

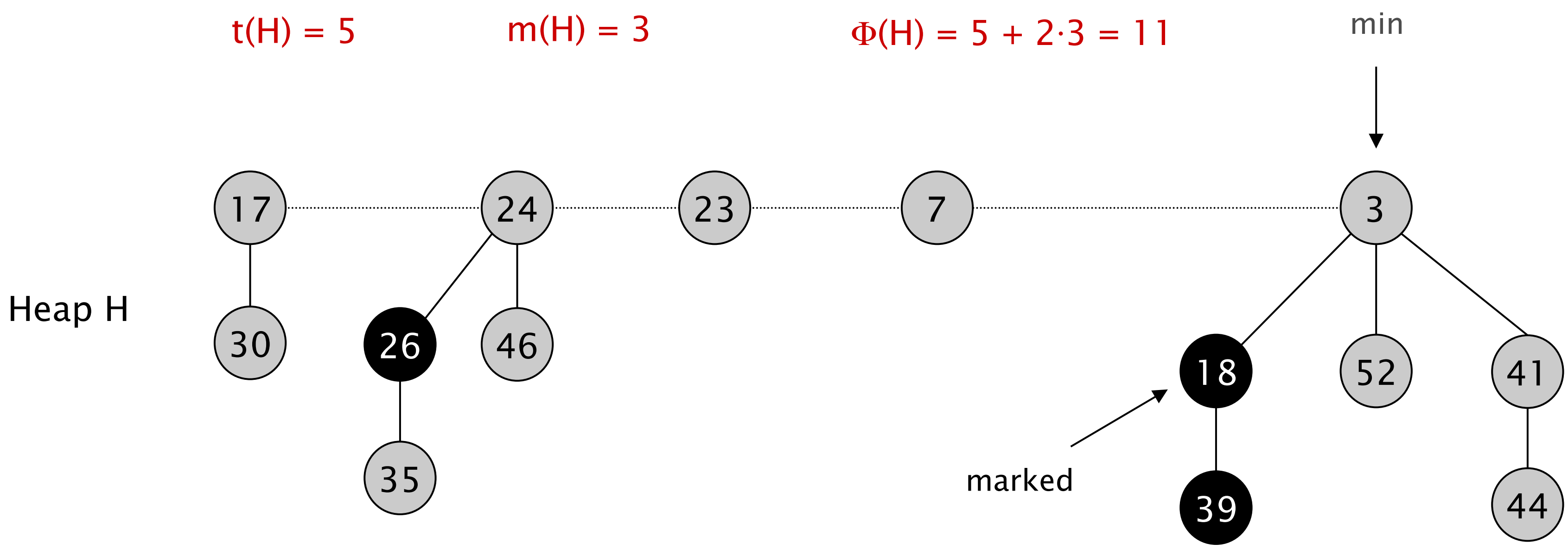
Notation.

- n = number of nodes in heap.
- degree(x) = number of children of node x.
- $D(n)$ = upper bound on the maximum degree of any node. In fact, $D(n) = O(\log n)$. The proof (omitted) uses Fibonacci numbers.
- $t(H)$ = number of trees in heap H.
- $m(H)$ = number of marked nodes in heap H.



Fibonacci Heaps: Potential Function

potential of heap H : $\Phi(H) = t(H) + 2m(H)$



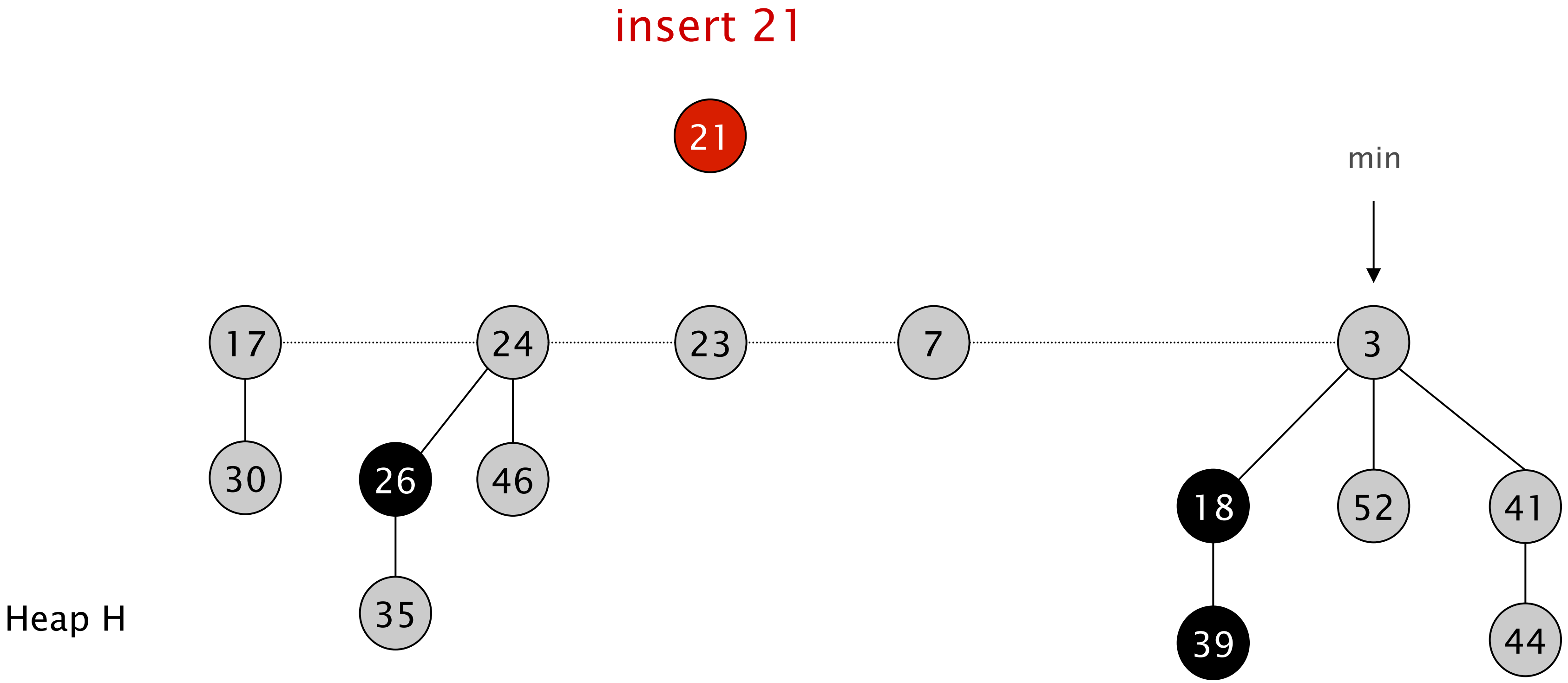
This lecture is not a complete treatment of Fibonacci heaps; in order to implement (code) and use them, more details are necessary (see book). Our main purpose here is to understand how the potential function works.

Next: analyze change in potential and amortized costs for heaps operations:
- Insert (easy, required)
- Extract min (medium, required)
- Decrease Key (difficult, optional)
- Union (easy, required)
- Delete (medium, required)

Insert

Fibonacci Heaps: Insert

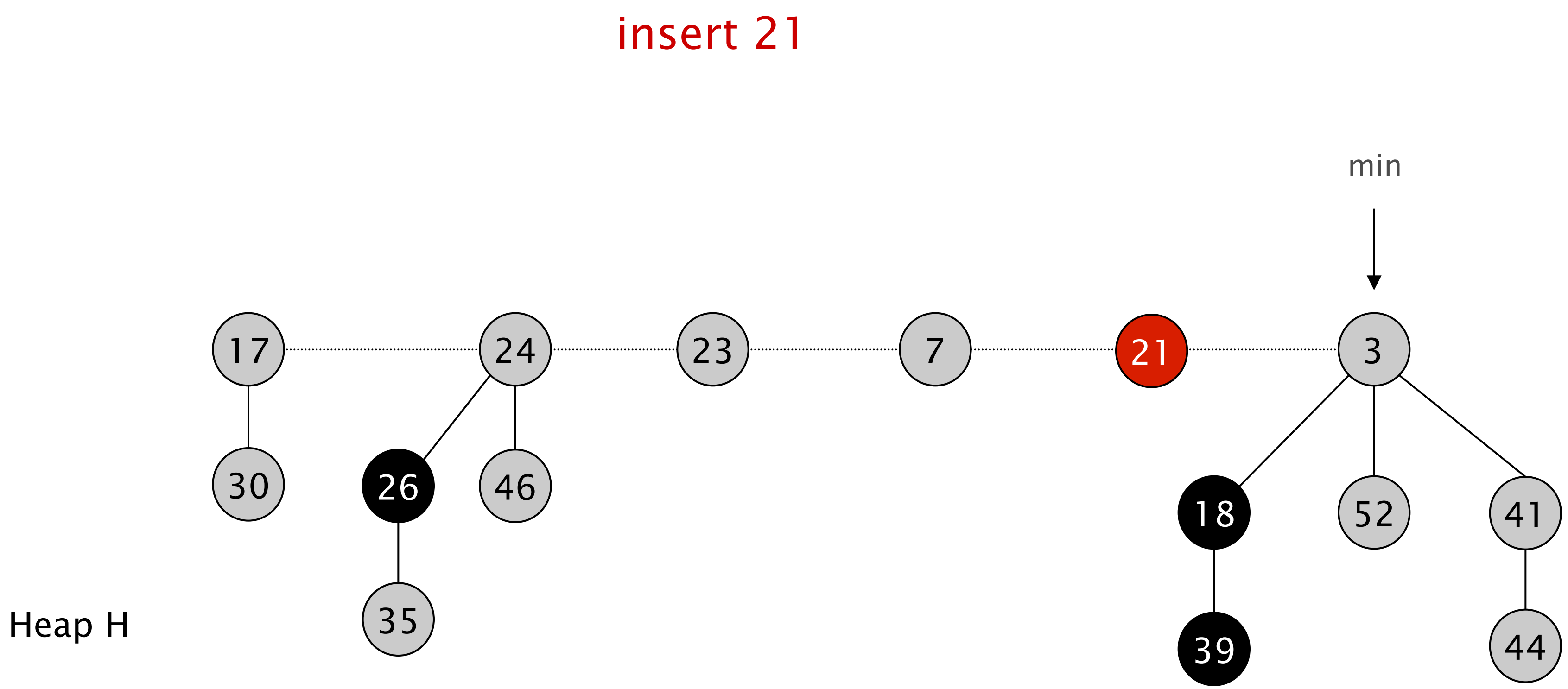
Insert.
- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).



Fibonacci Heaps: Insert

Insert.

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



Fibonacci Heaps: Insert Analysis

Actual cost. $O(1)$

- H' = the heap after insert

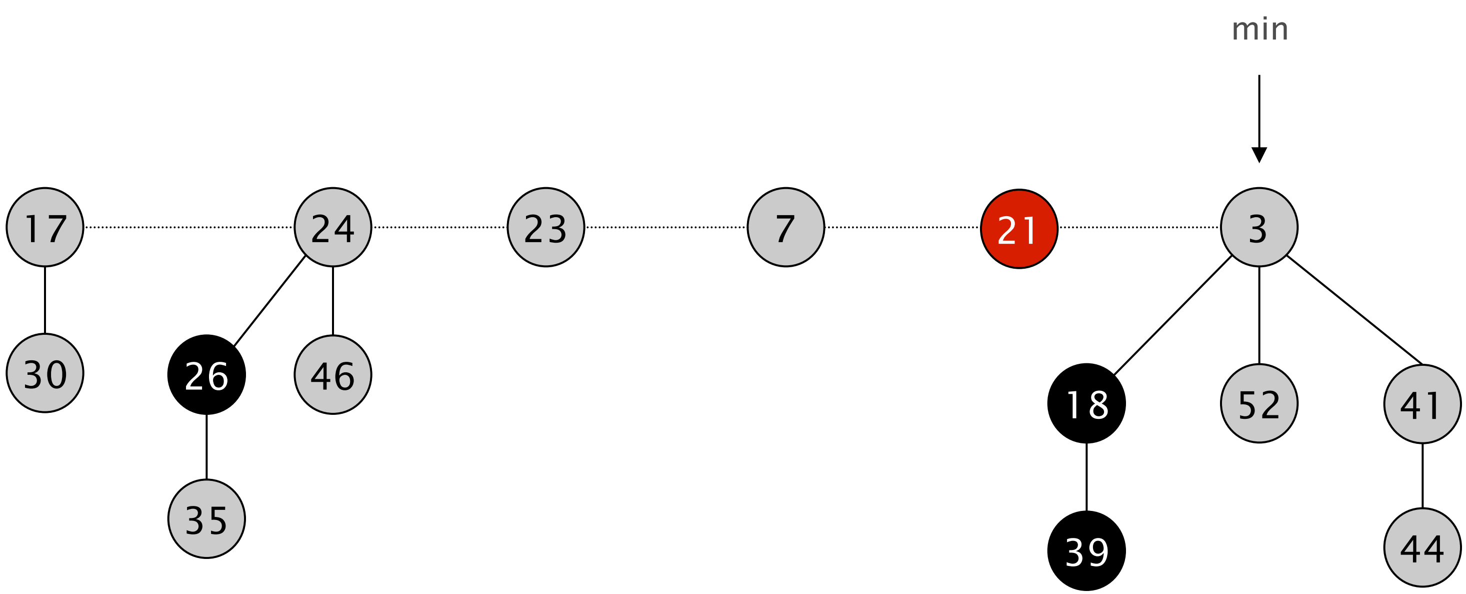
$$\Phi(H) = t(H) + 2m(H)$$

potential of heap H

Change in potential.

$$\Delta\Phi = (t(H') + 2m(H')) - (t(H) + 2m(H)) = $$
$$(t(H) + 1 + 2m(H)) - (t(H) + 2m(H)) = 1$$

Amortized cost. $O(1)$

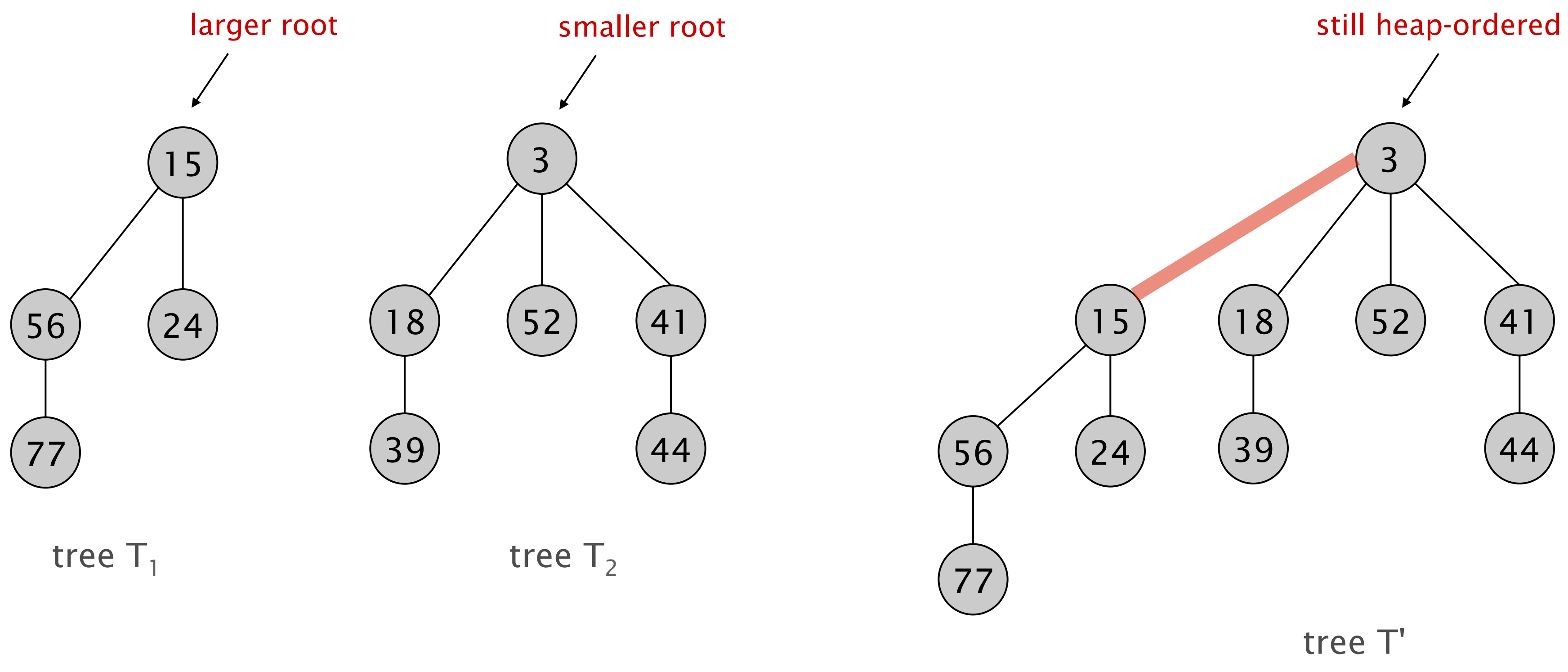


Heap H

Extract-Min

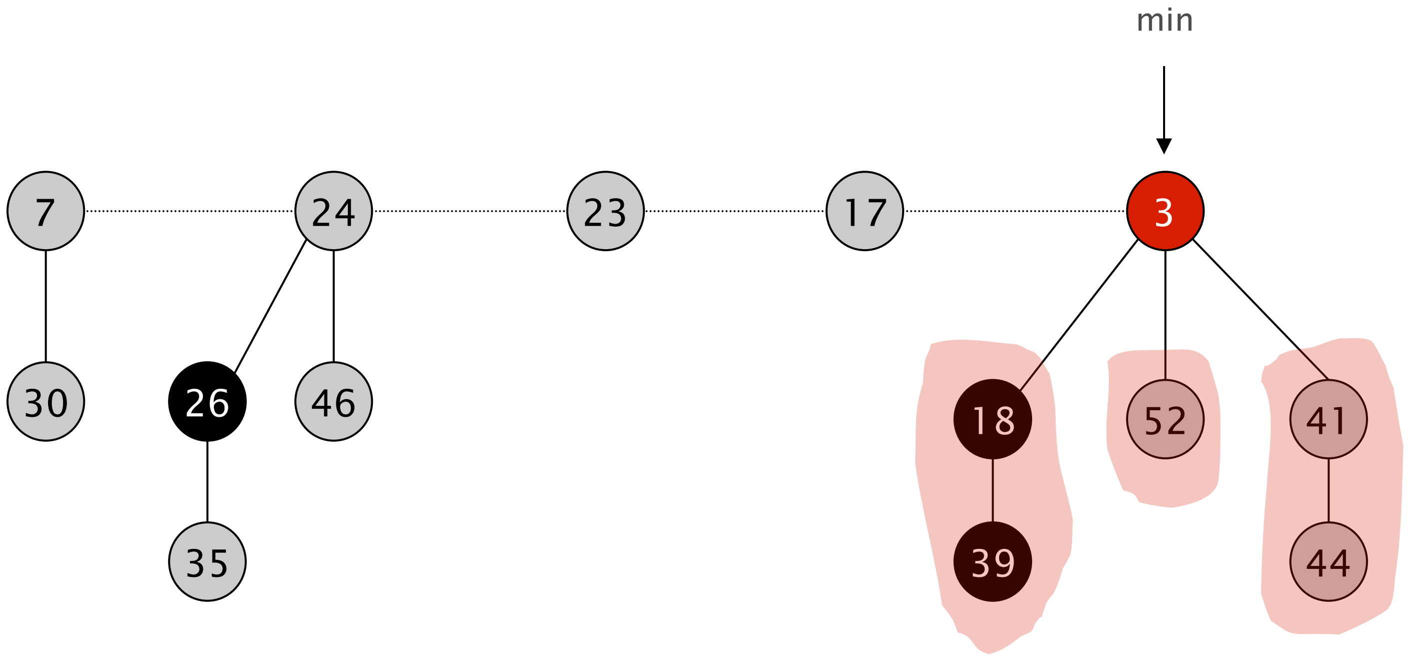
Linking Operation

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



Extract-min.

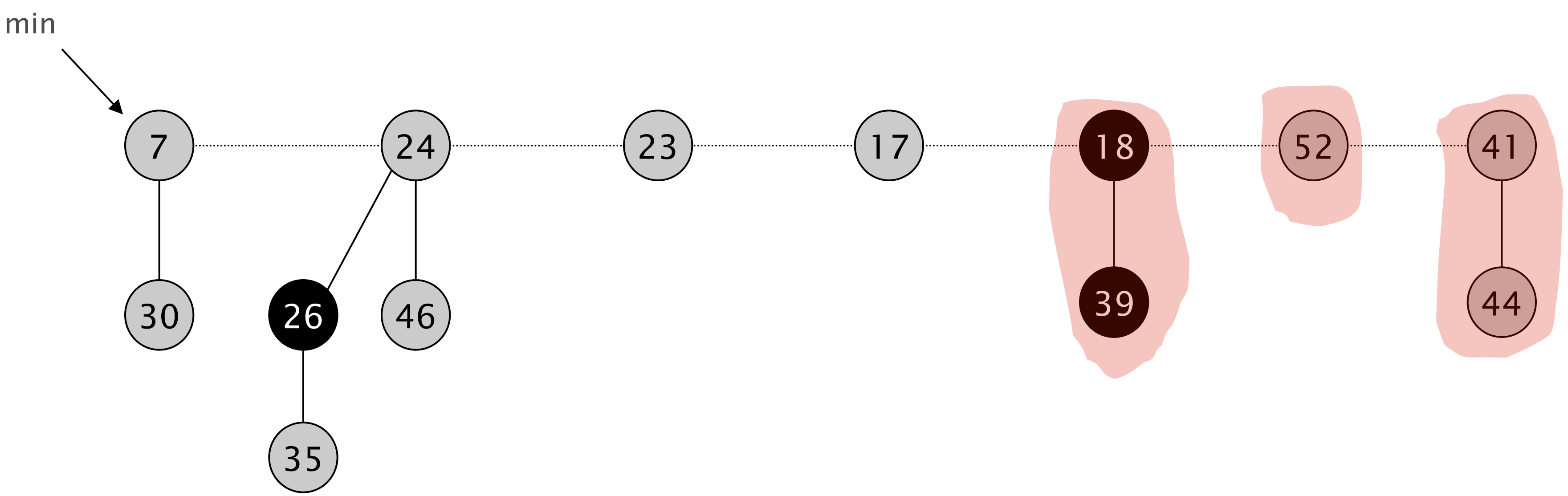
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

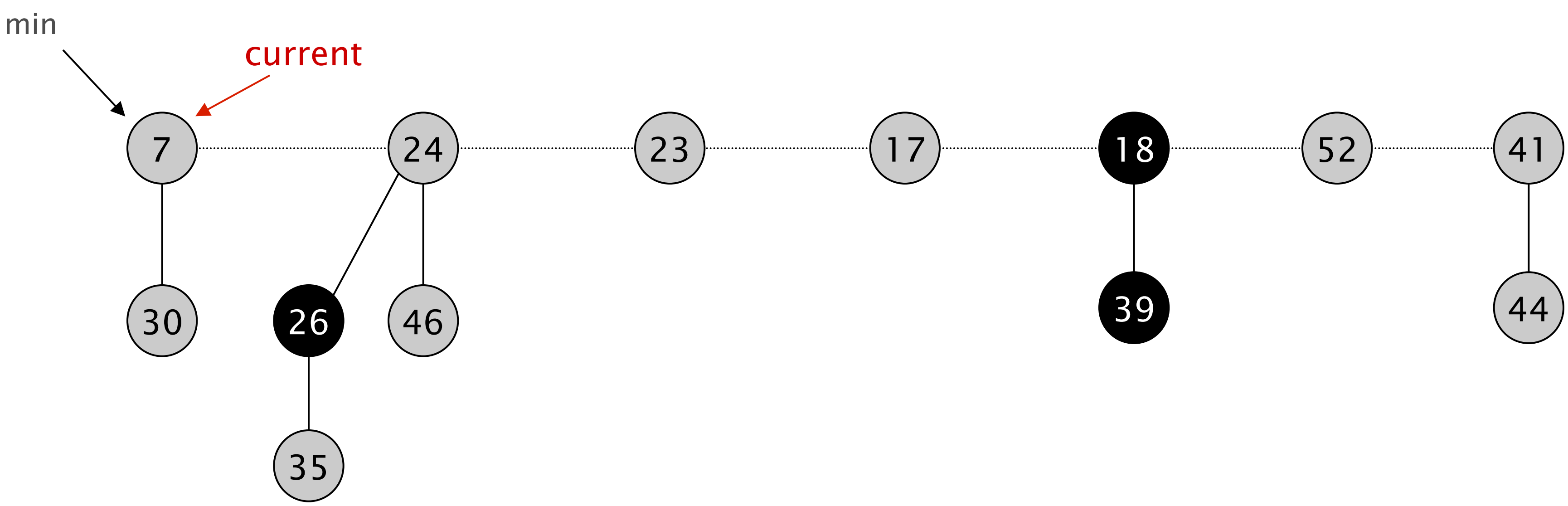
Extract-Min.

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



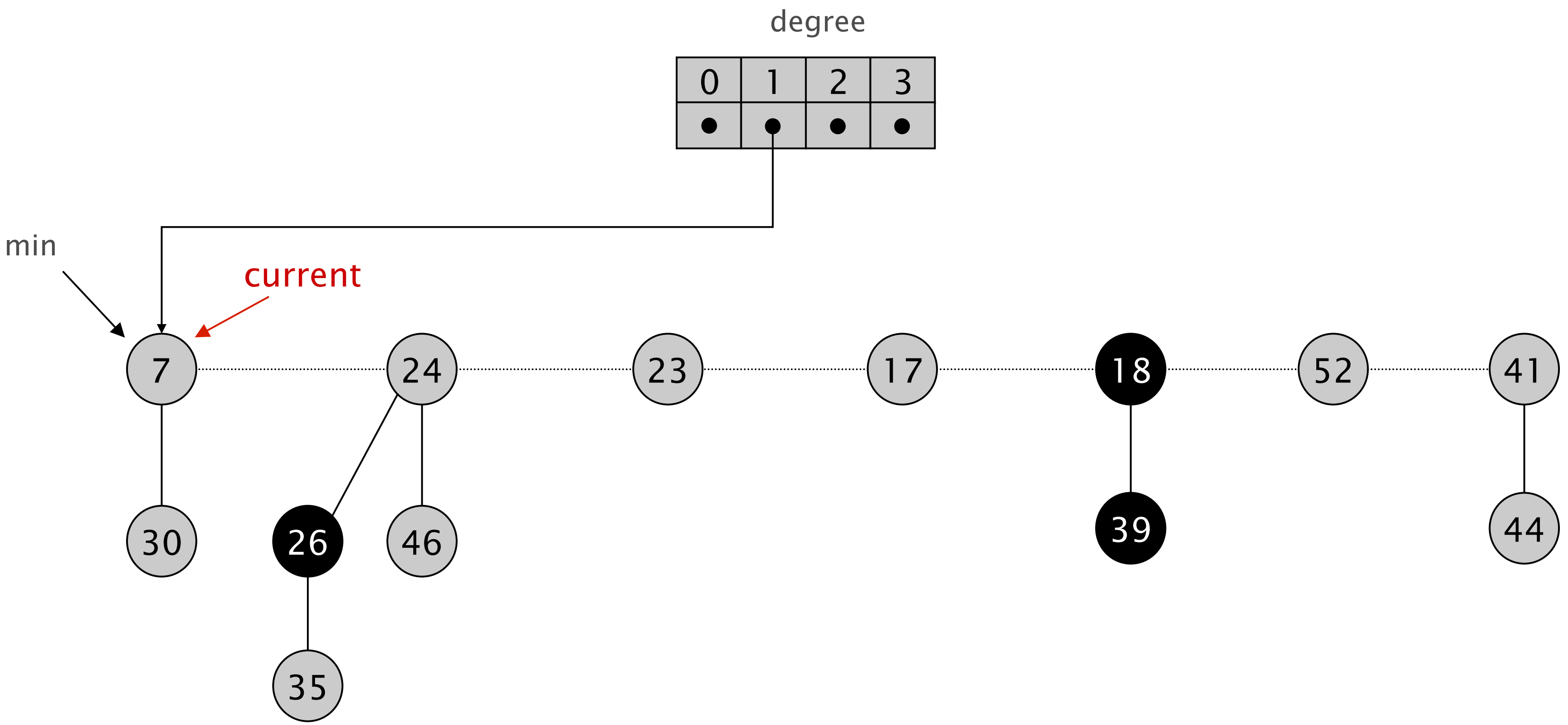
Extract-Min.

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



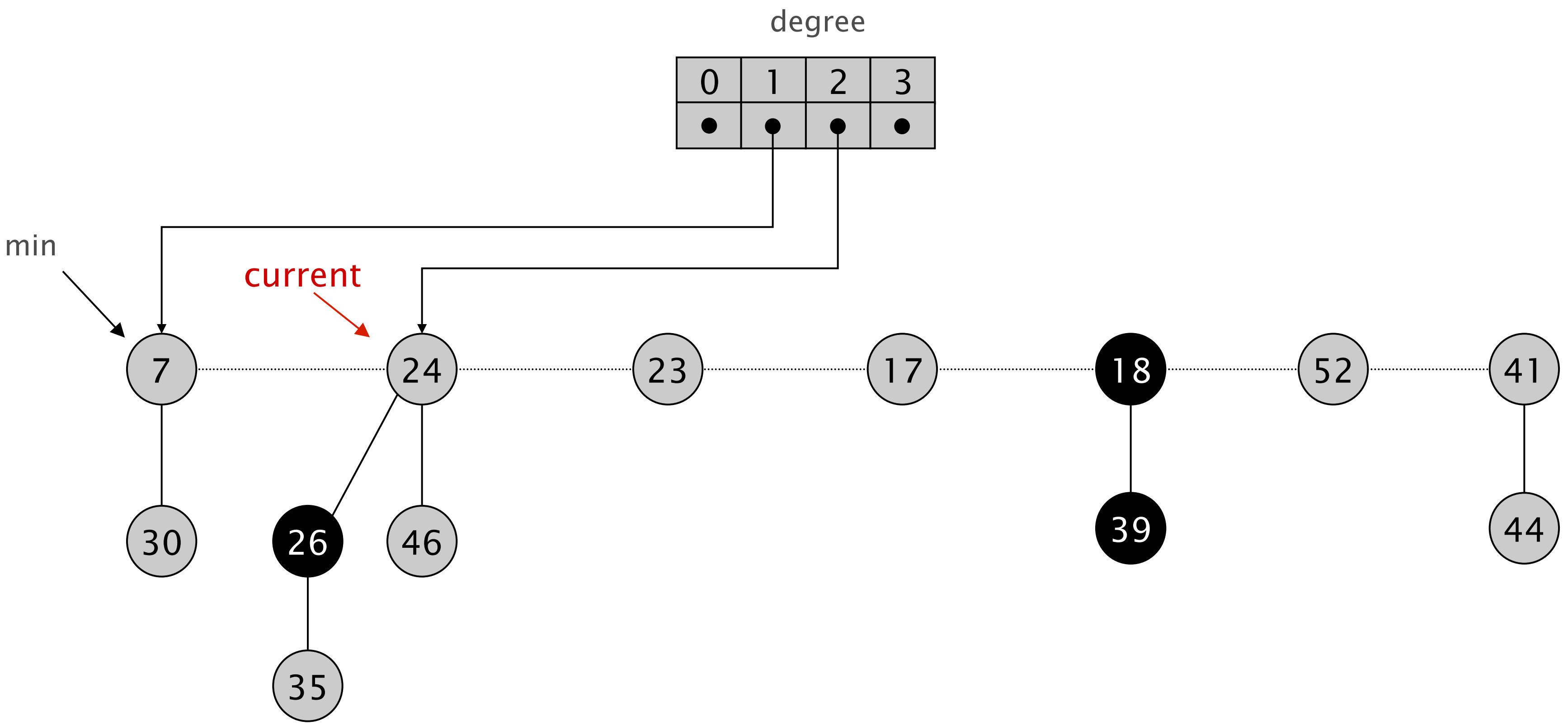
Extract-Min.

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



Extract-Min.

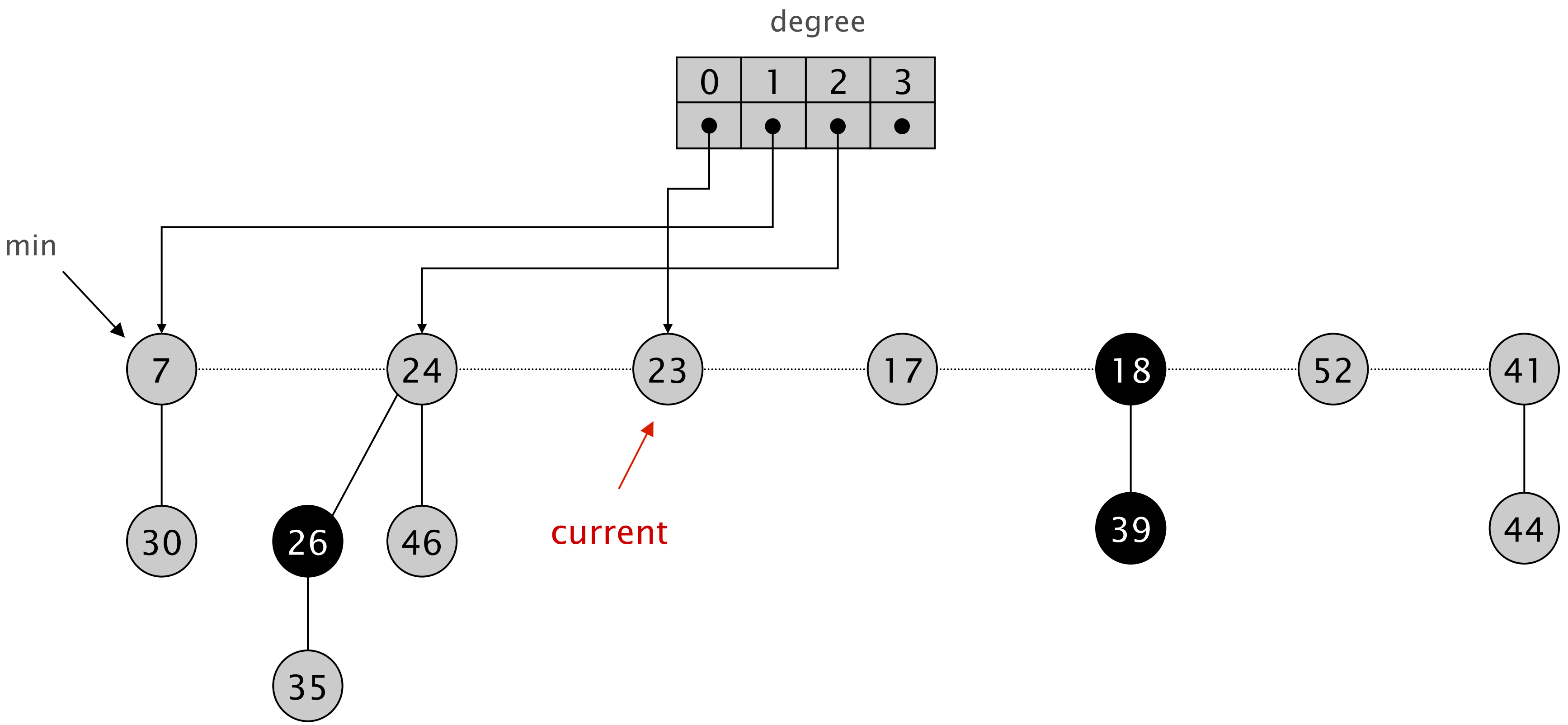
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

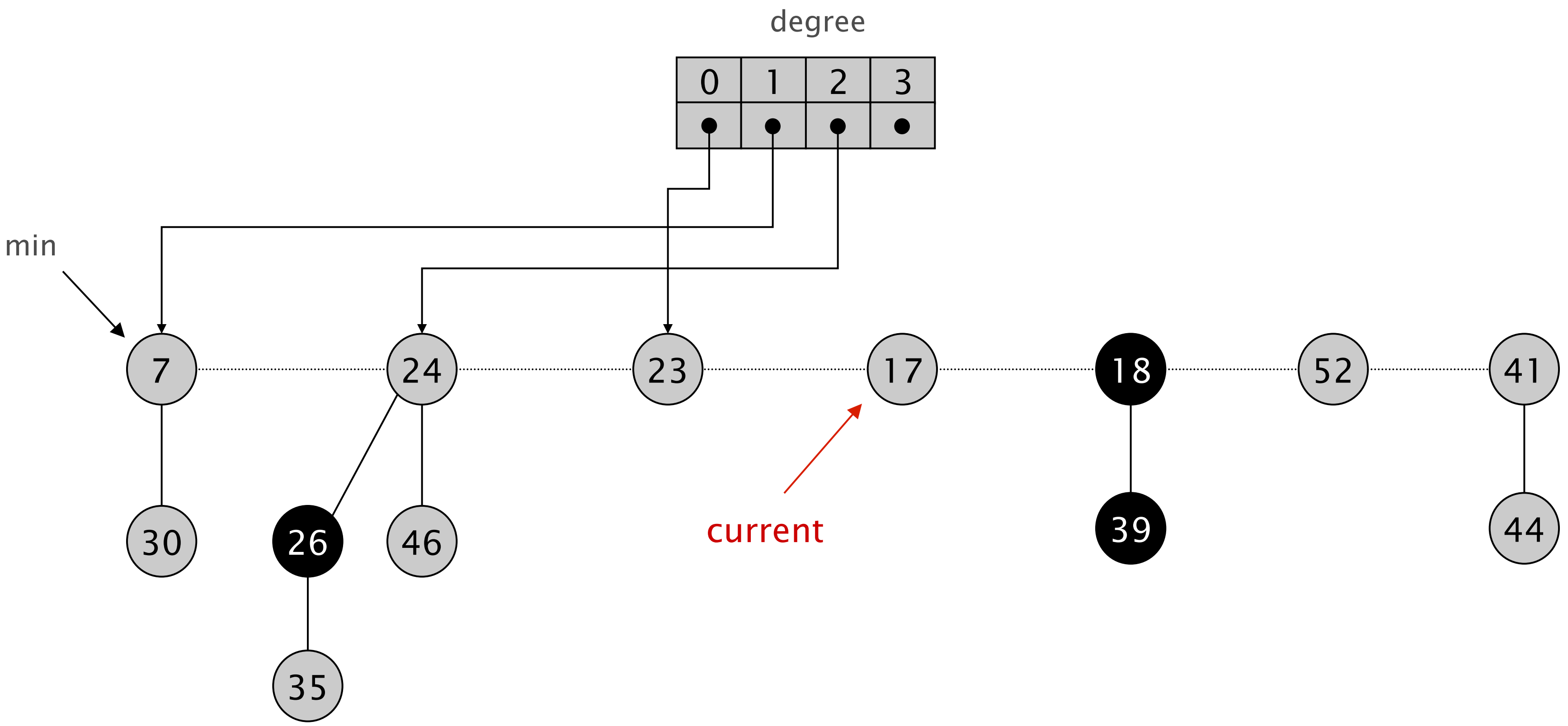
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



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 degree.

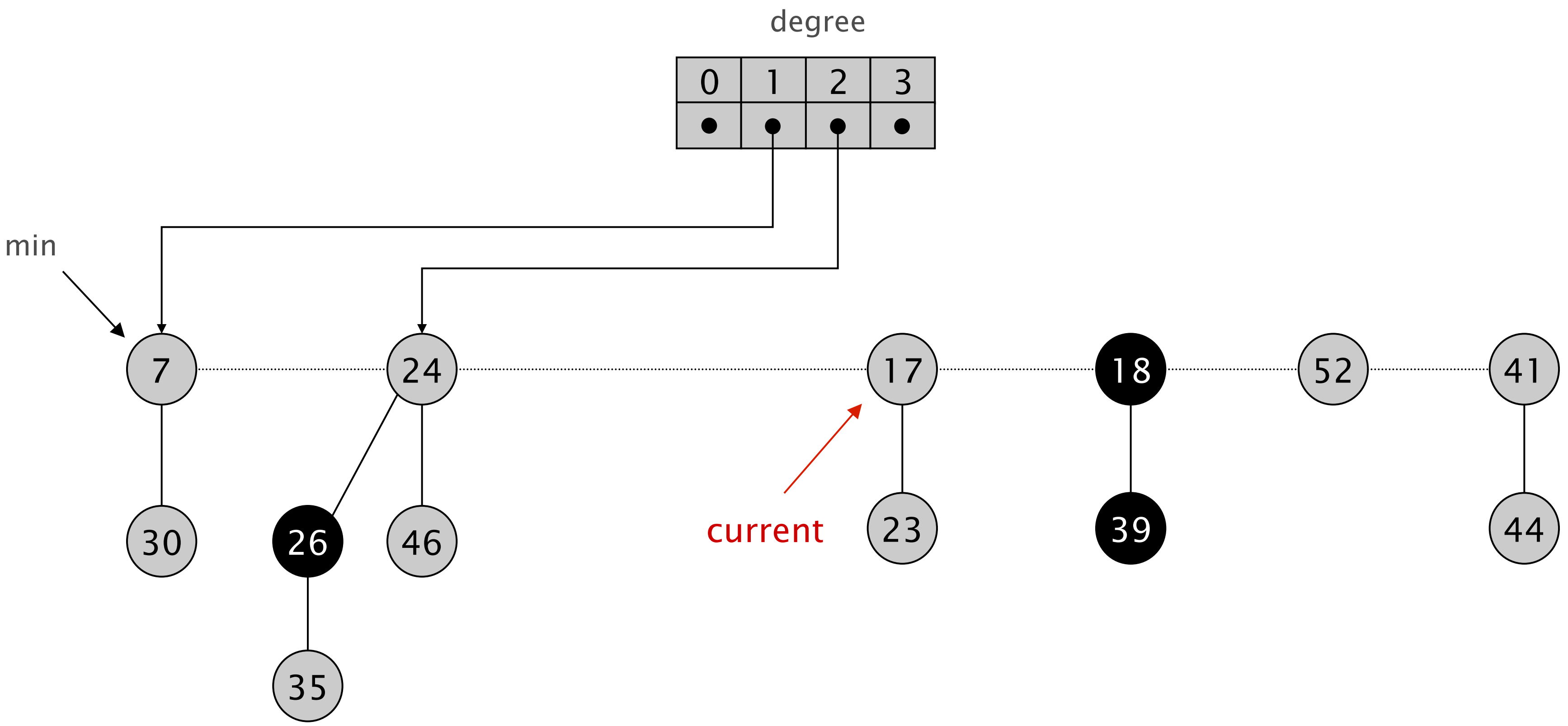


link 23 into 17

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 degree.

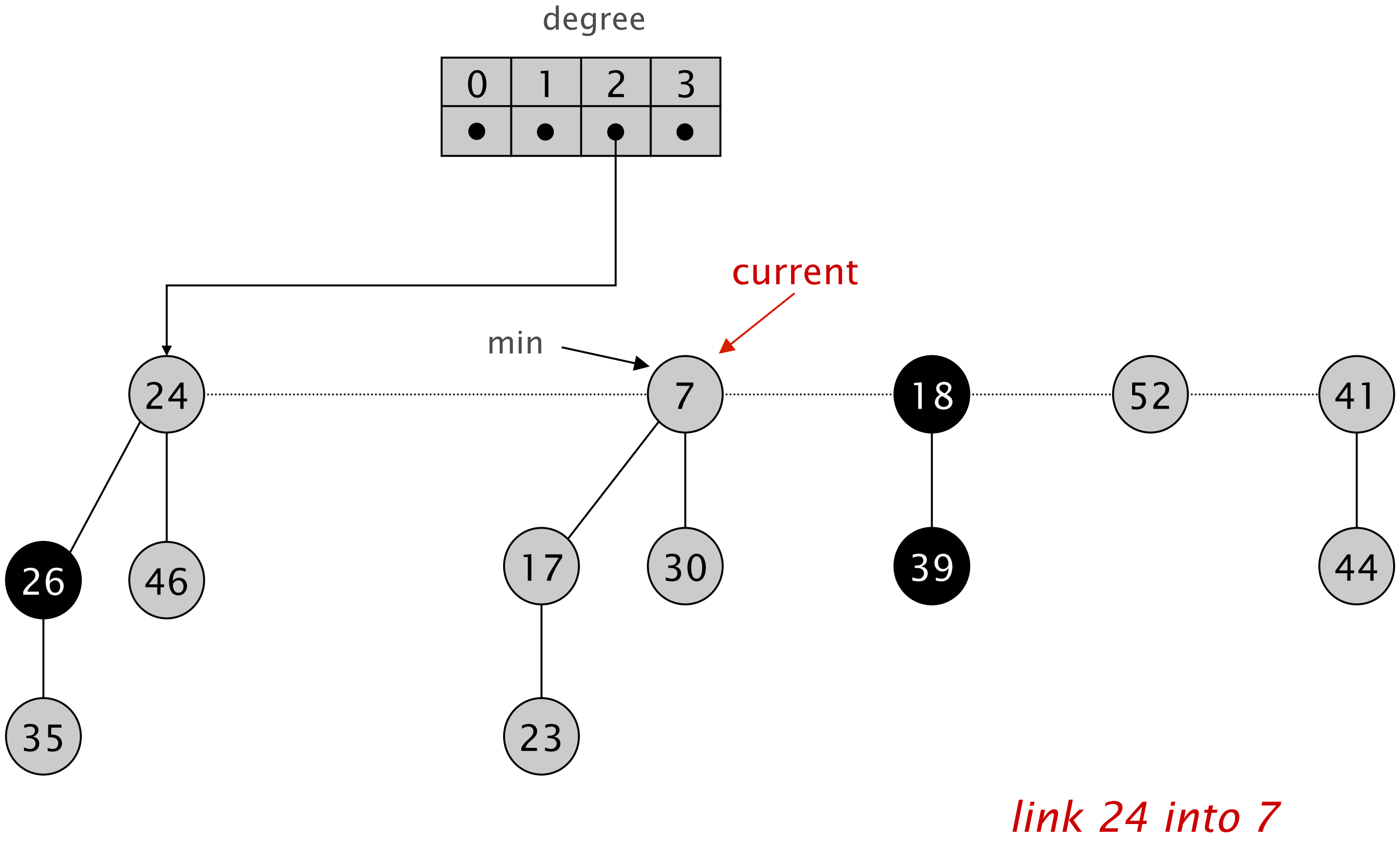


link 17 into 7

Fibonacci Heaps: Extract-Min

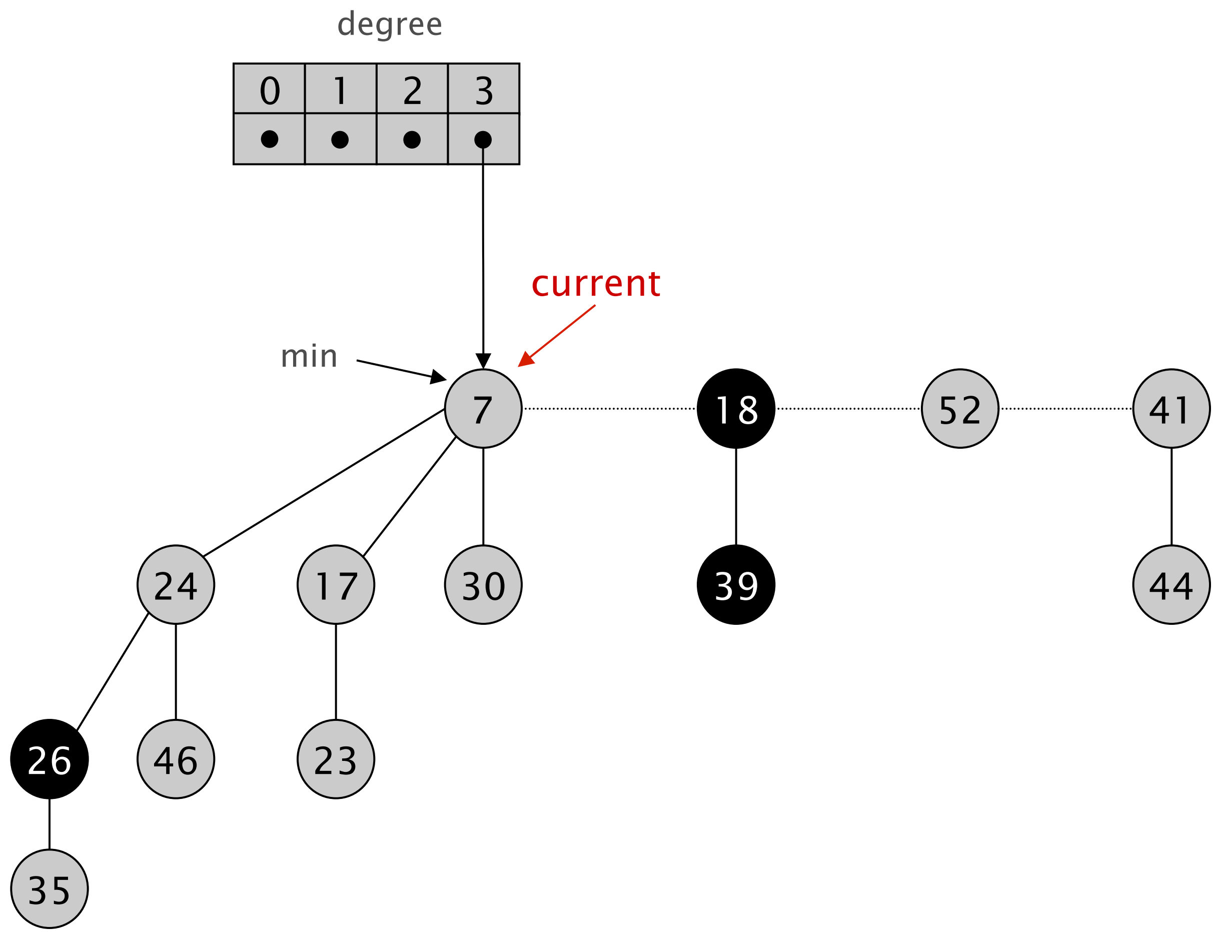
Extract-Min.

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



Extract-Min.

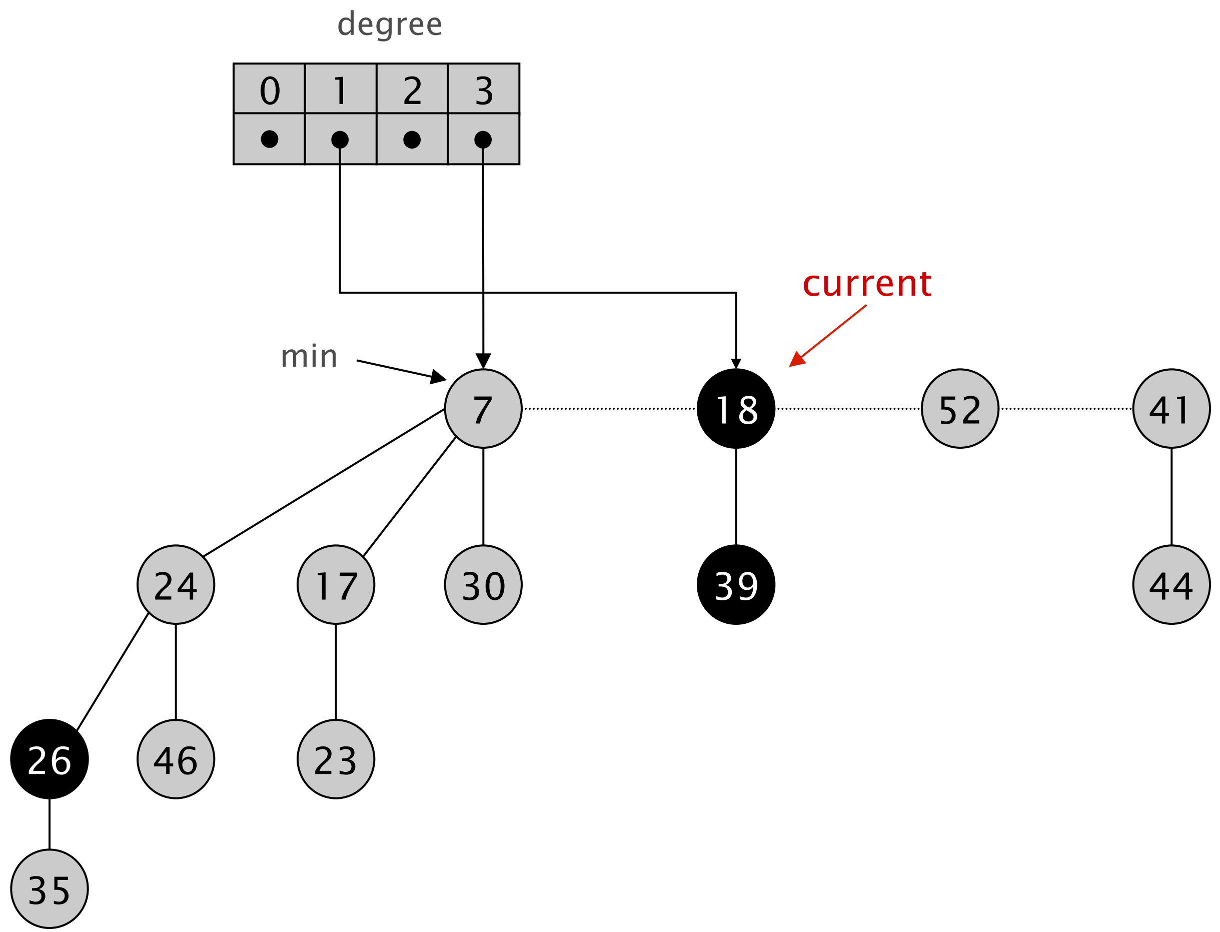
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

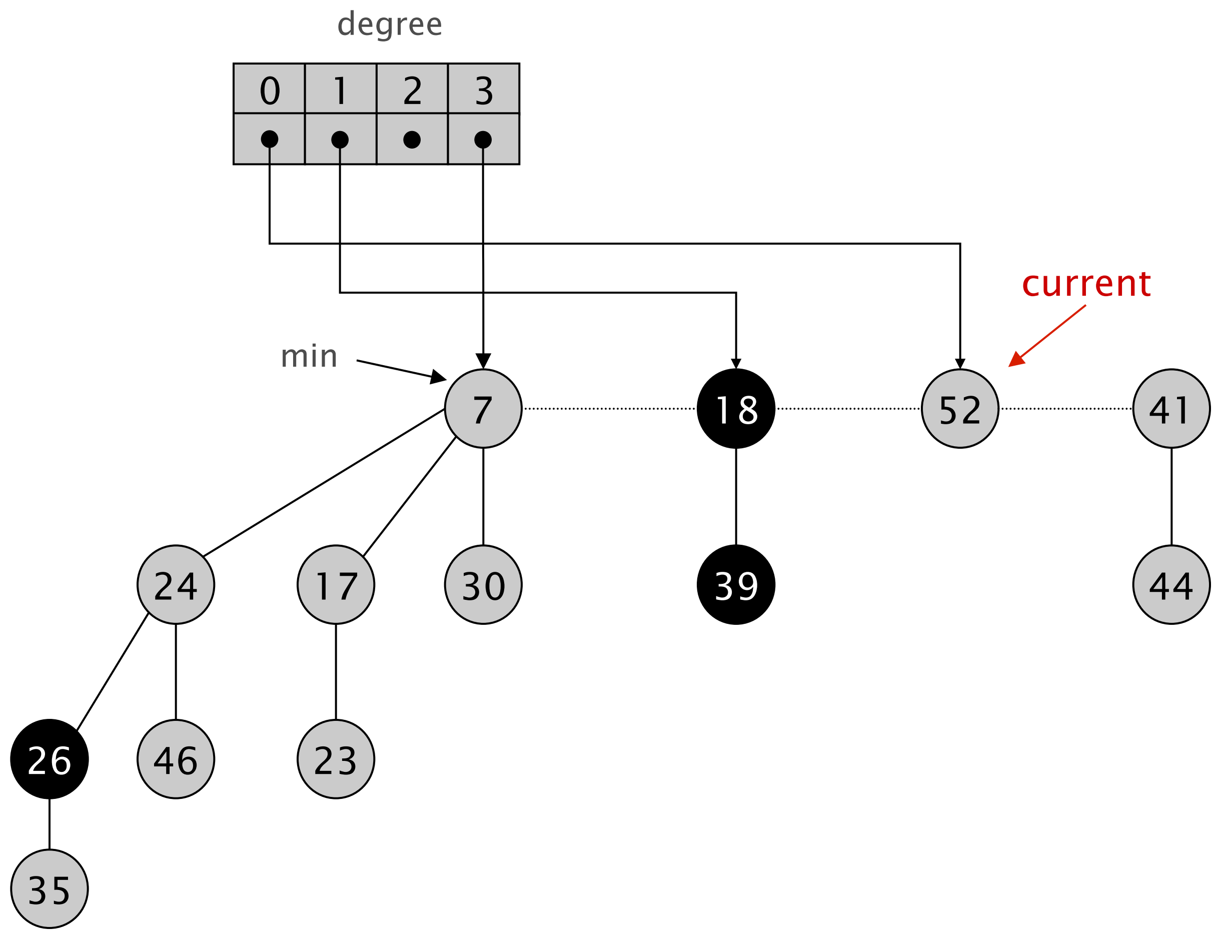
Extract-Min.

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



Extract-Min.

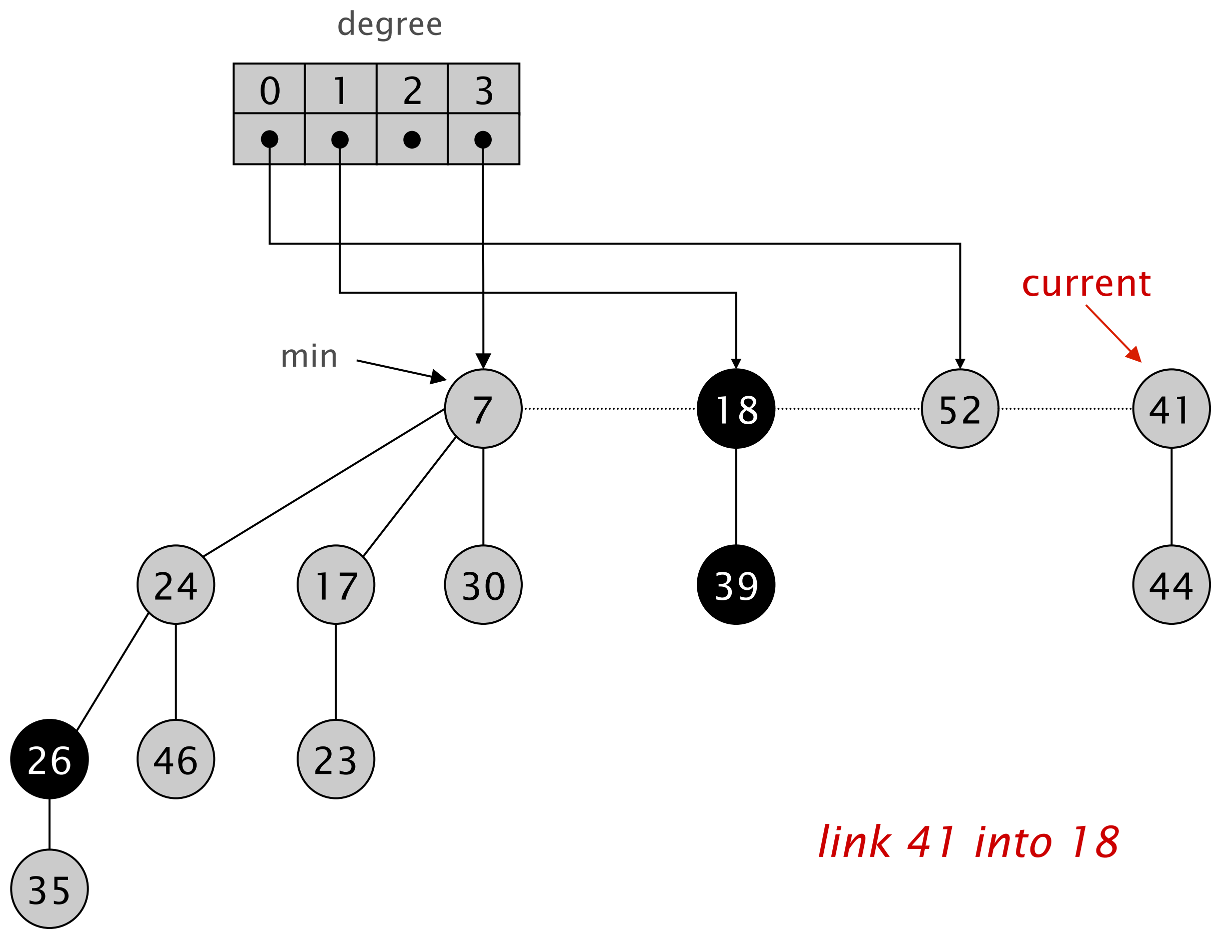
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

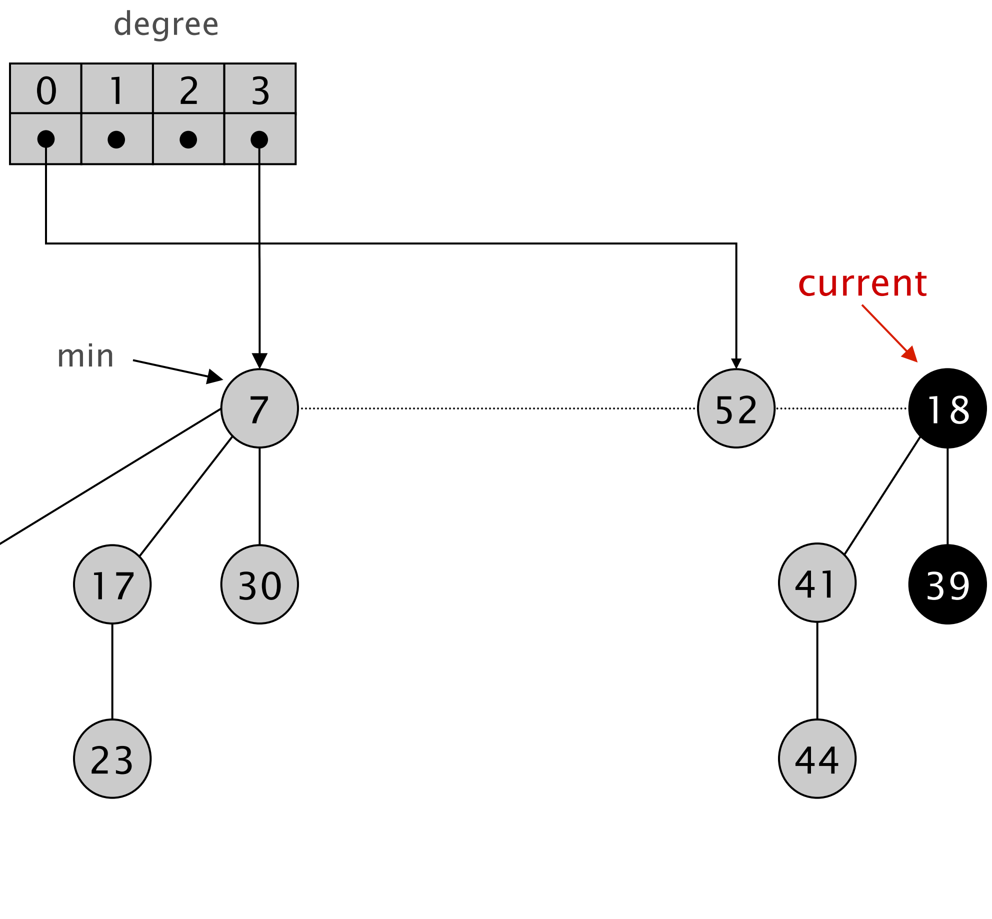
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

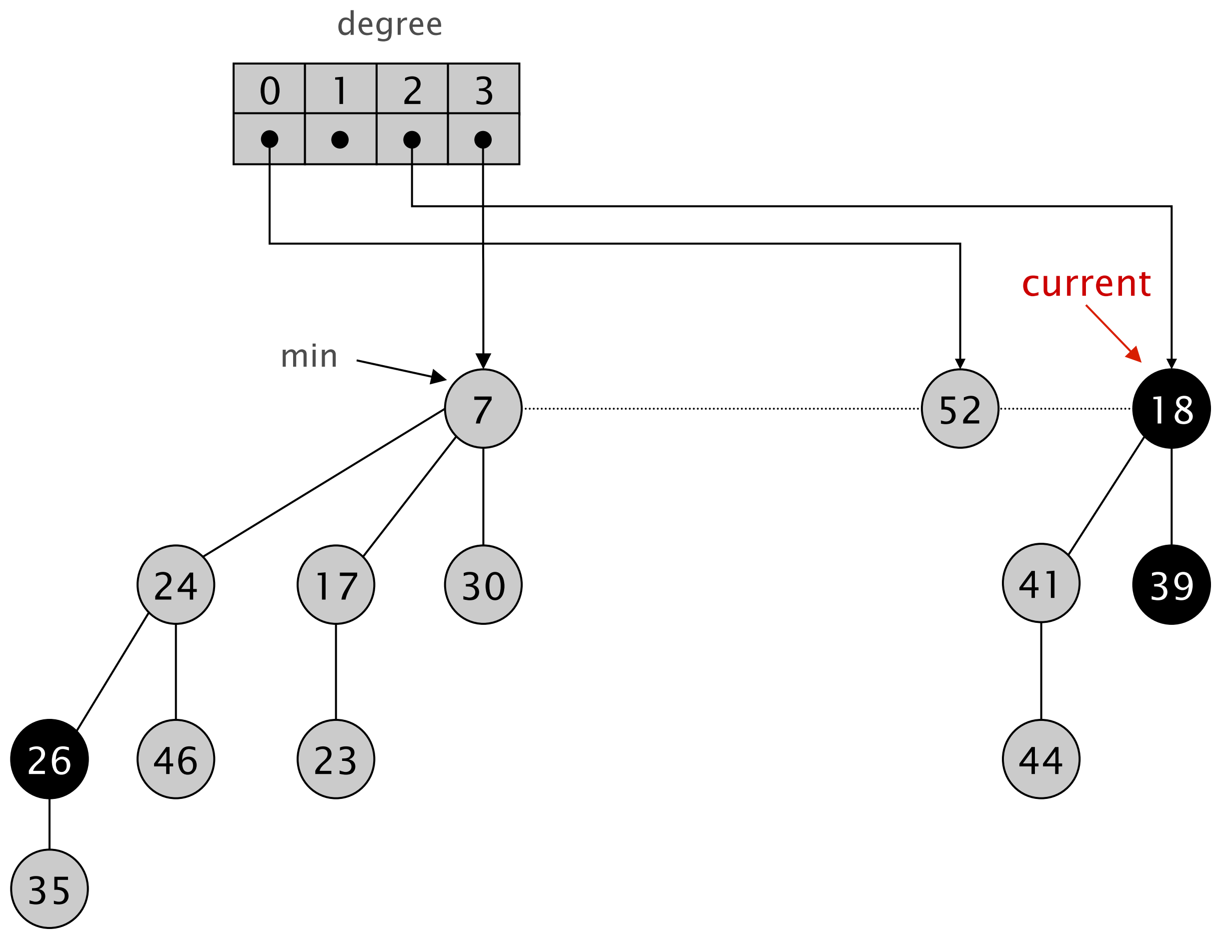
Extract-Min.

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



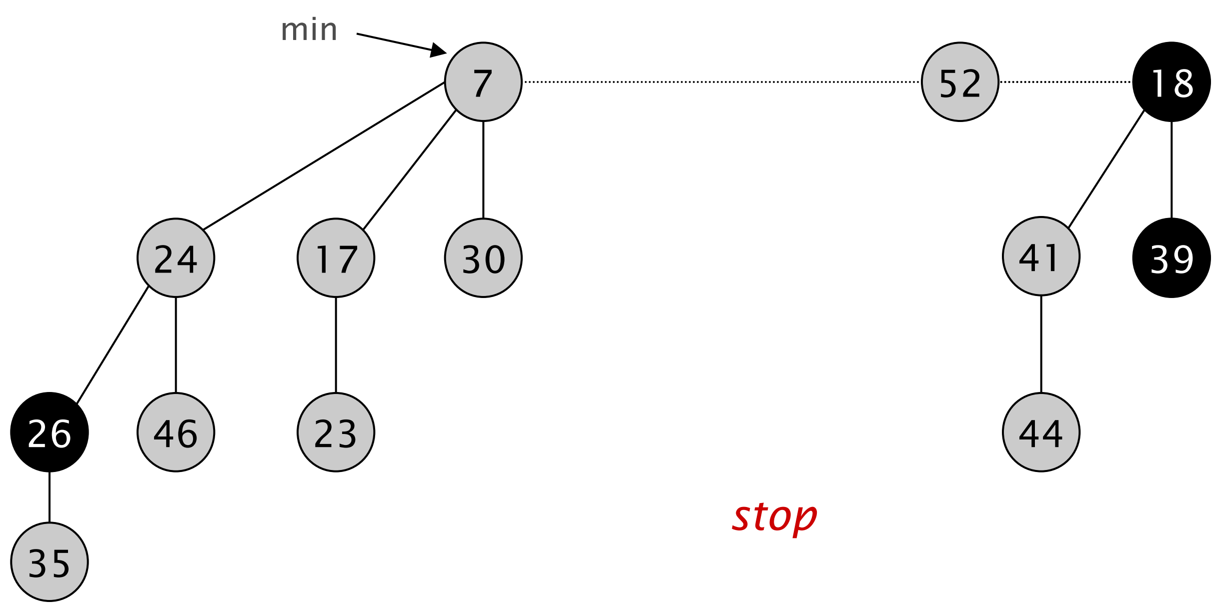
Extract-Min.

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



Extract-Min.

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



Extract-Min.

$$\Phi(H) = t(H) + 2m(H)$$

potential function

Actual cost. $O(D(n)) + O(t(H))$

- $O((D(n))$ to meld min's children into root list. (at most $D(n)$ children of min)
- $O(D(n)) + O(t(H))$ to update min. (the size of the root list is at most $D(n) + t(H) - 1$)
- $O(D(n)) + O(t(H))$ to consolidate trees. (one of the roots is linked to another in each merging, and thus the total number of iterations is at most the number of roots in the root list.)

Change in potential: $O(D(n)) - t(H)$

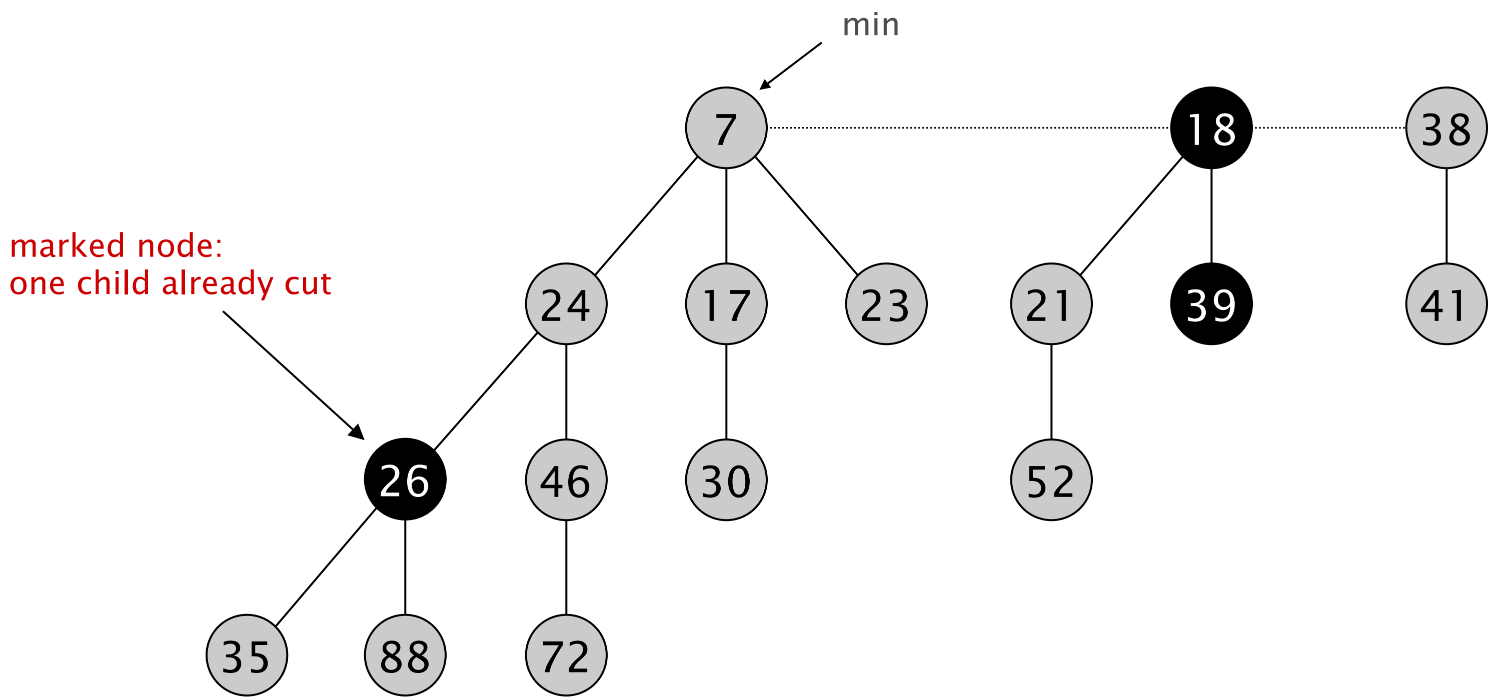
- $\Phi(H') = D(n) + 1 + 2m(H)$ (at most $D(n) + 1$ roots with distinct degrees remain and no nodes become marked during the operation)

Amortized cost: $O((D(n))$

Decrease Key

Intuition for deceasing the key of node x.

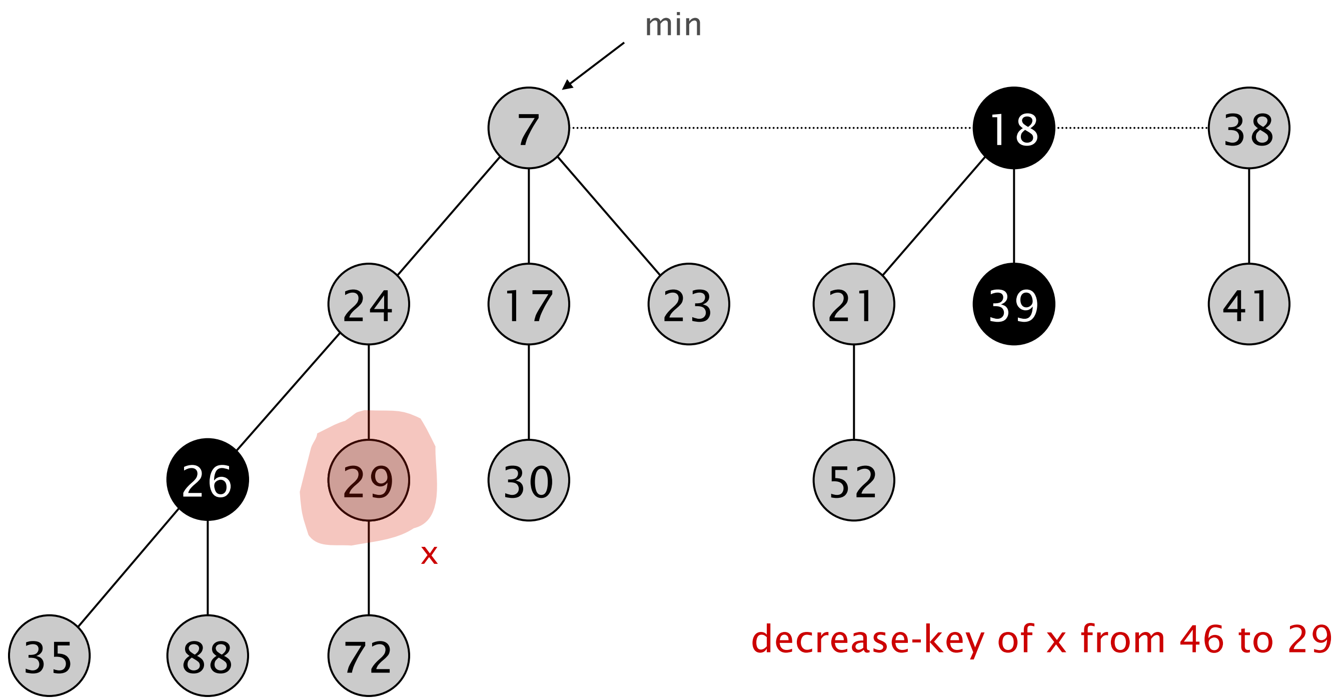
- If heap-order is not violated, just decrease the key of x.
- Otherwise, cut tree rooted at x and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

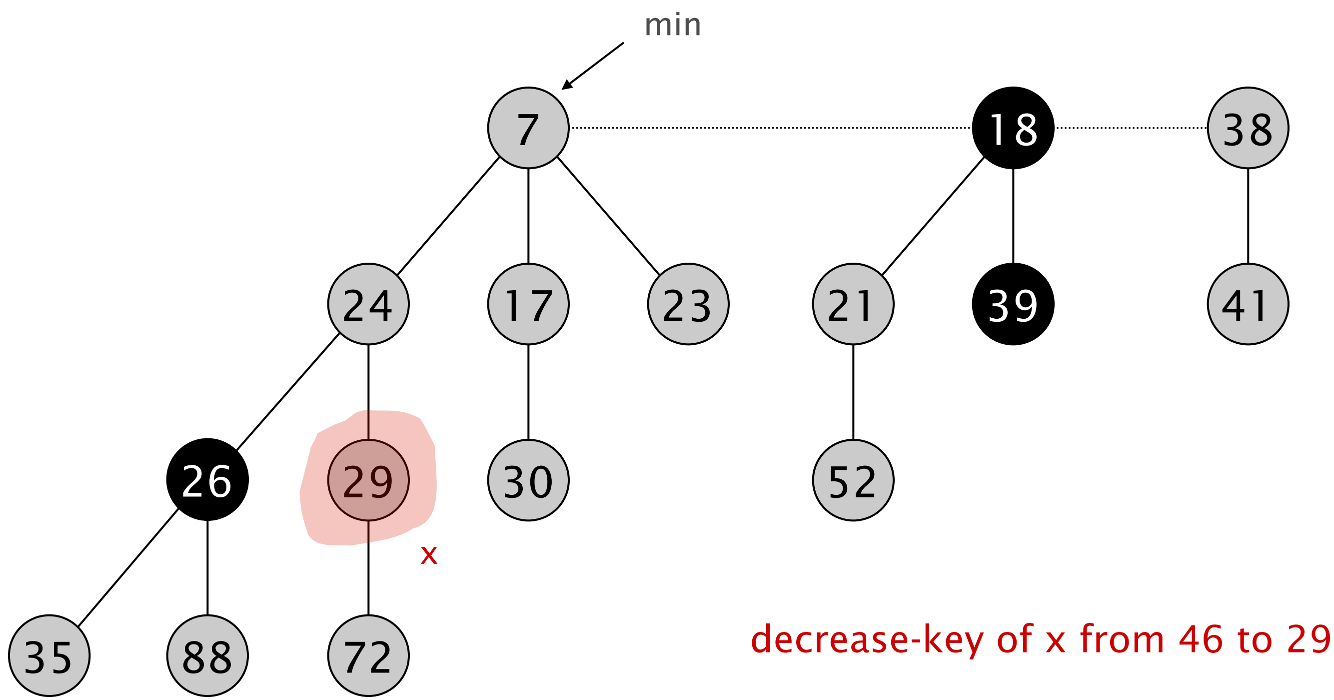
- Decrease key of x.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

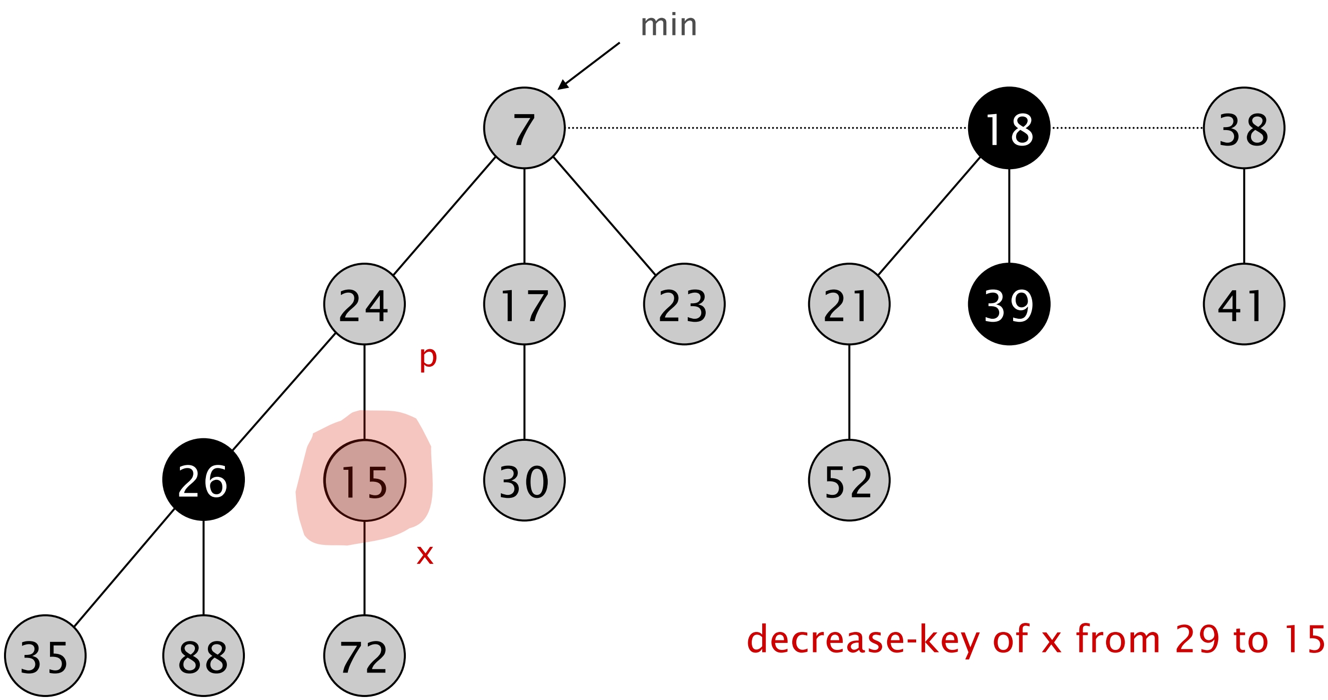
- Decrease key of x.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

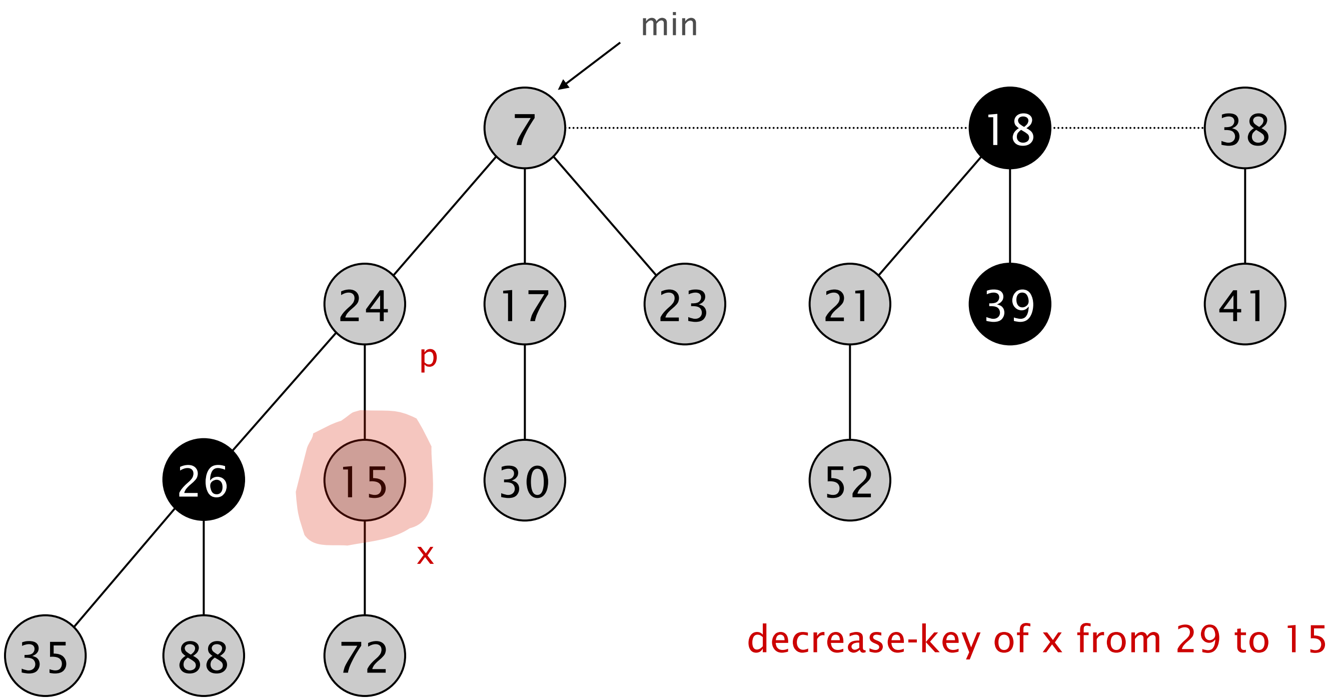
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

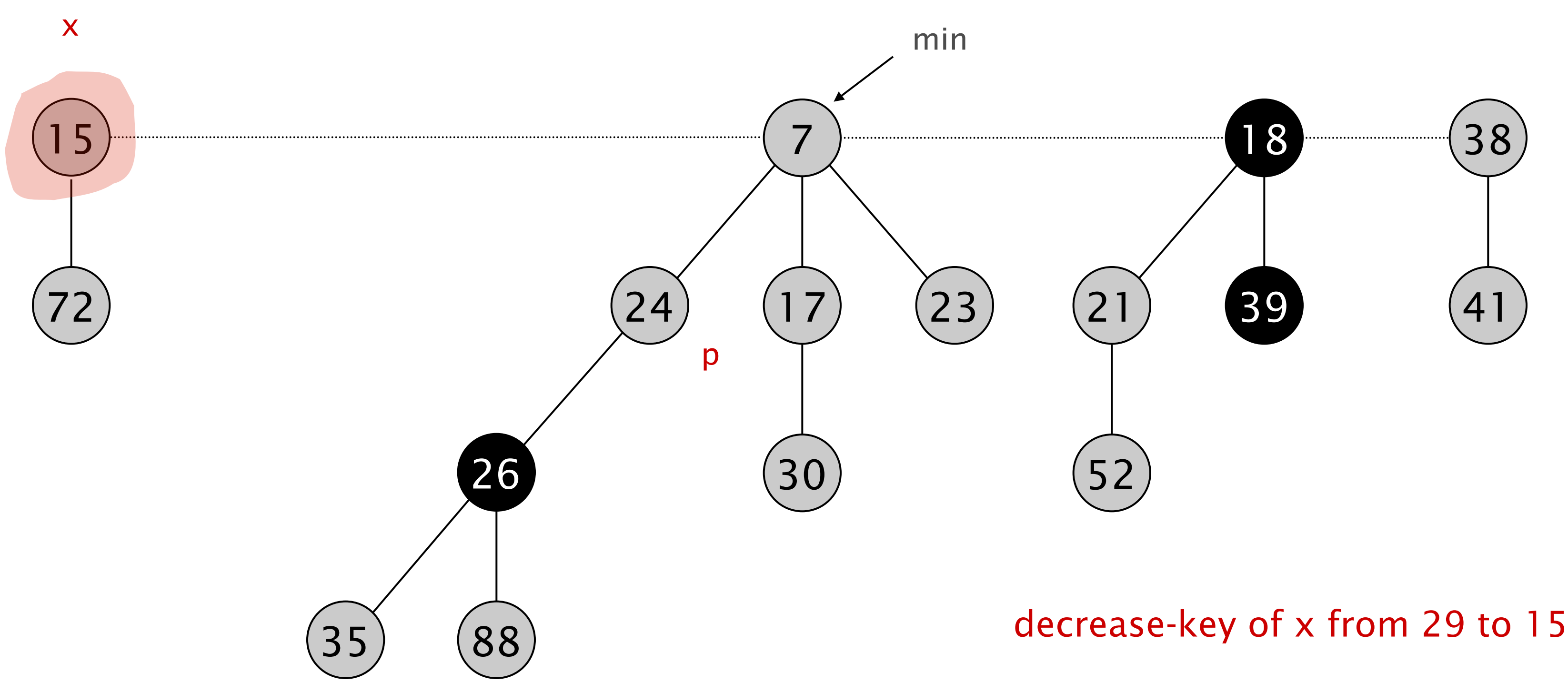
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

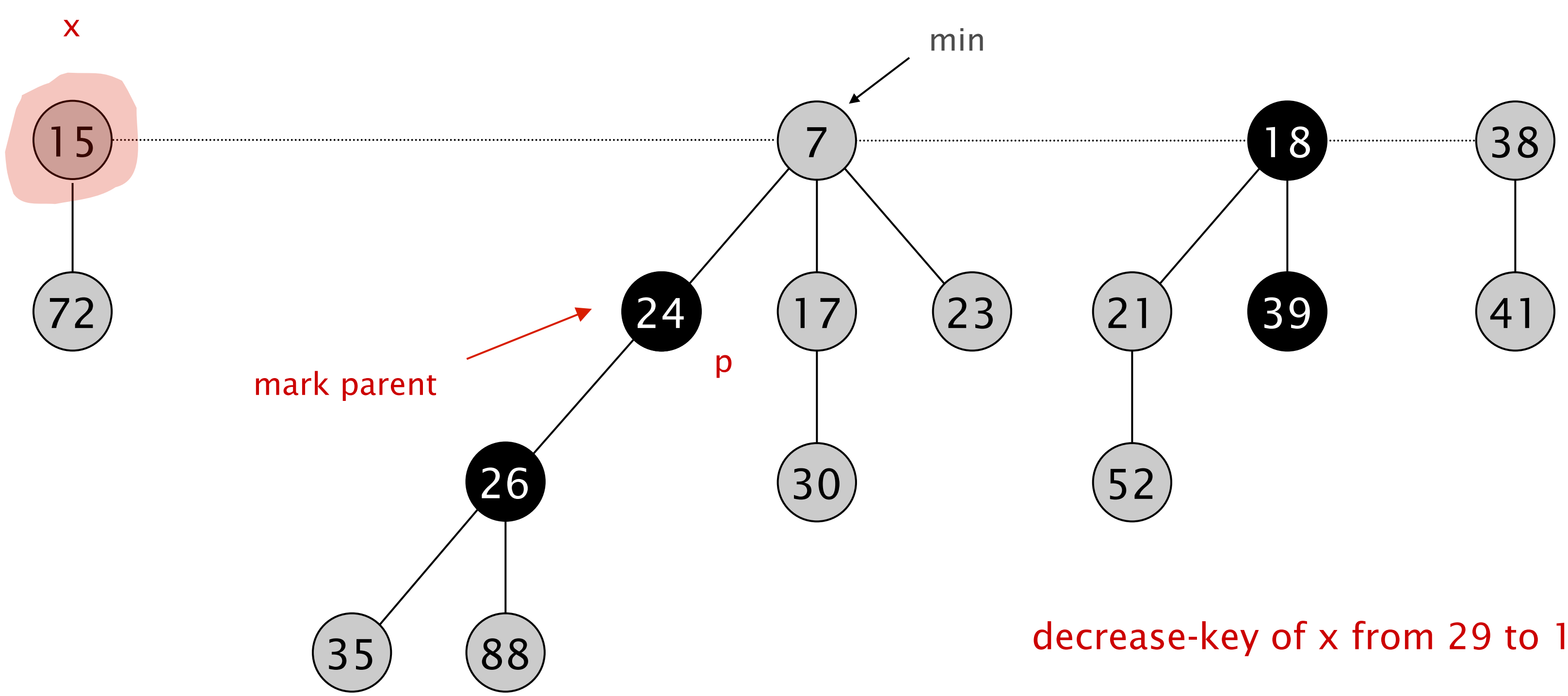
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



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).

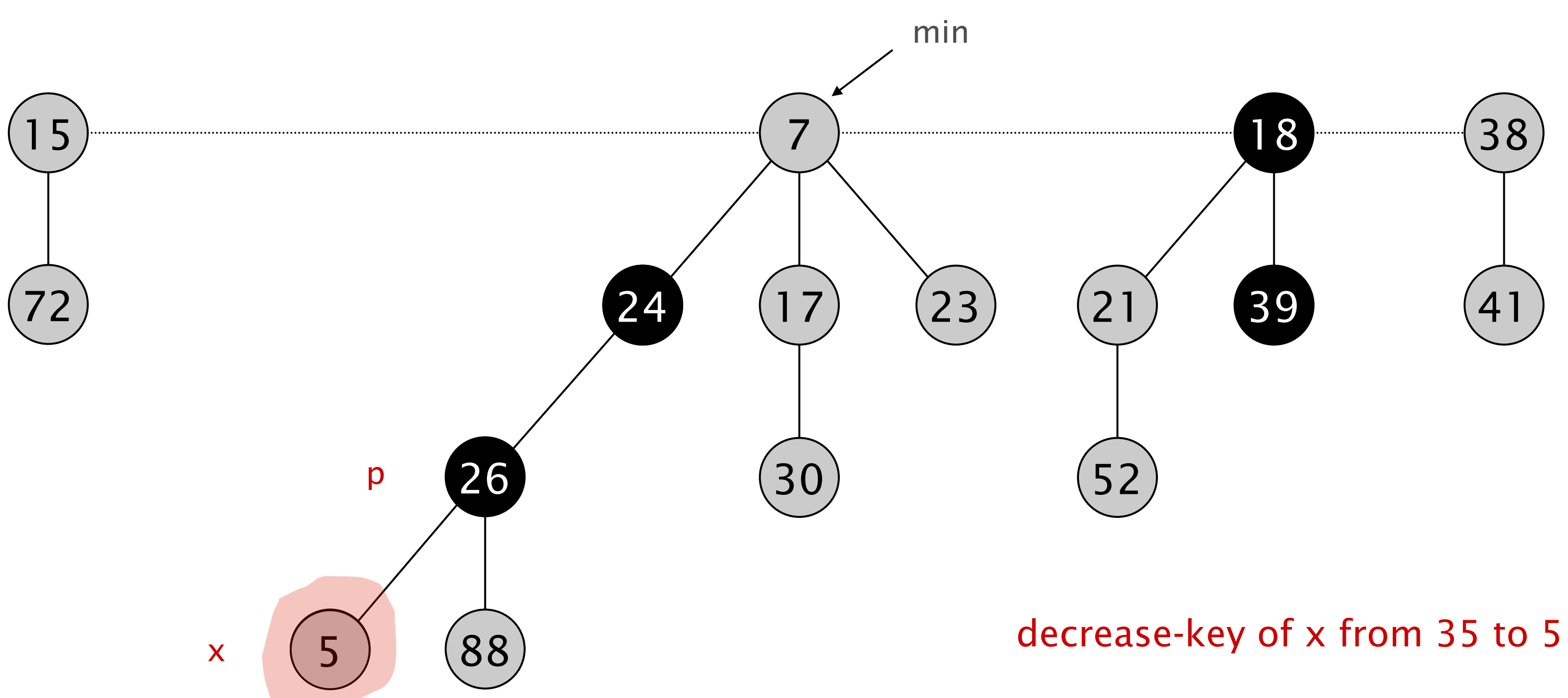


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

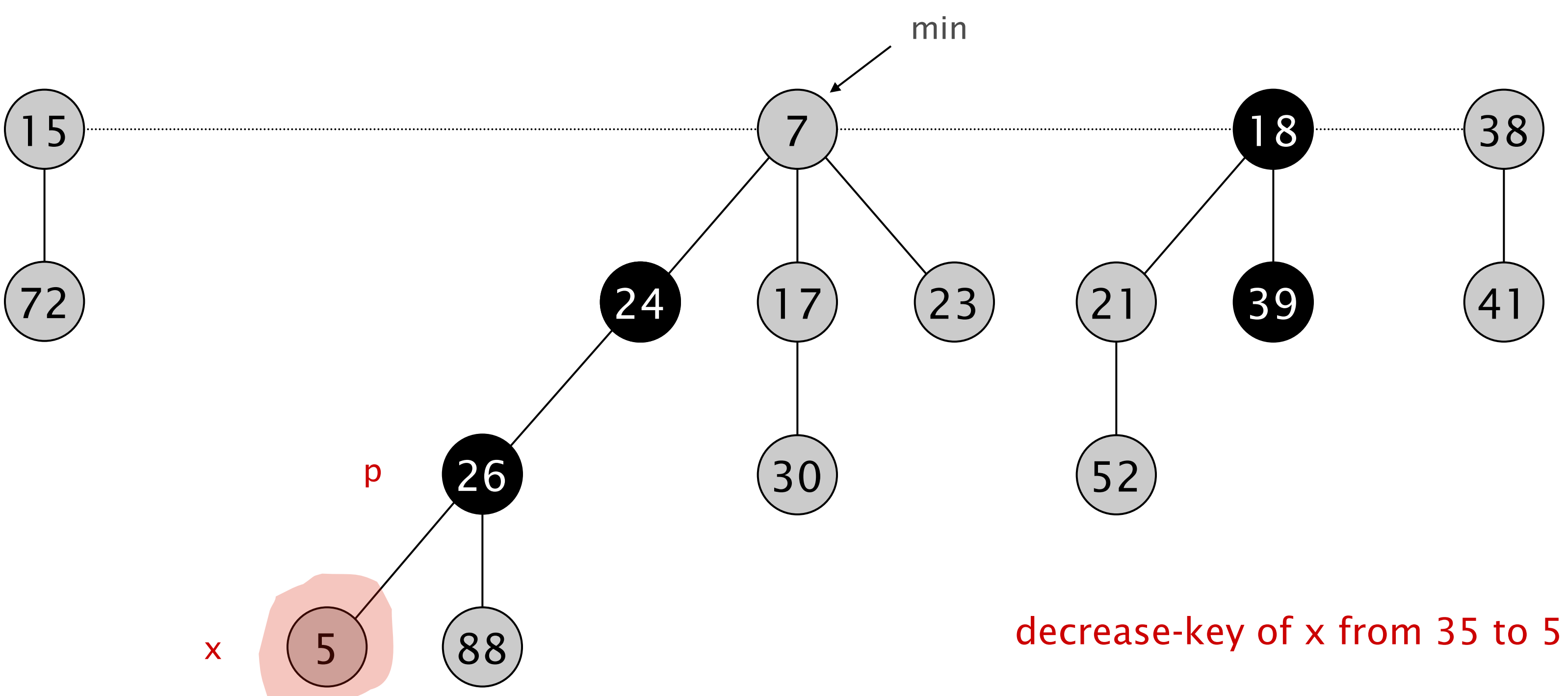
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

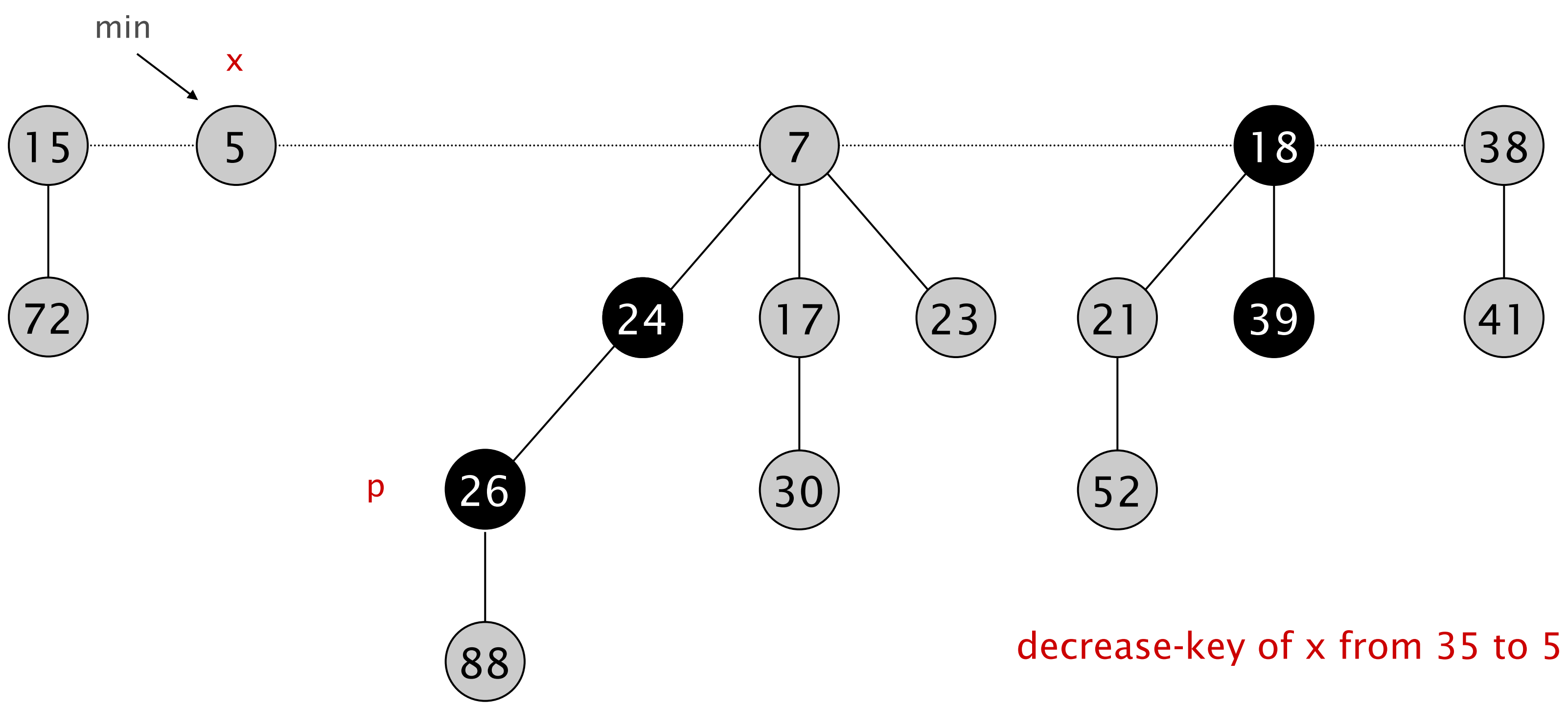
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

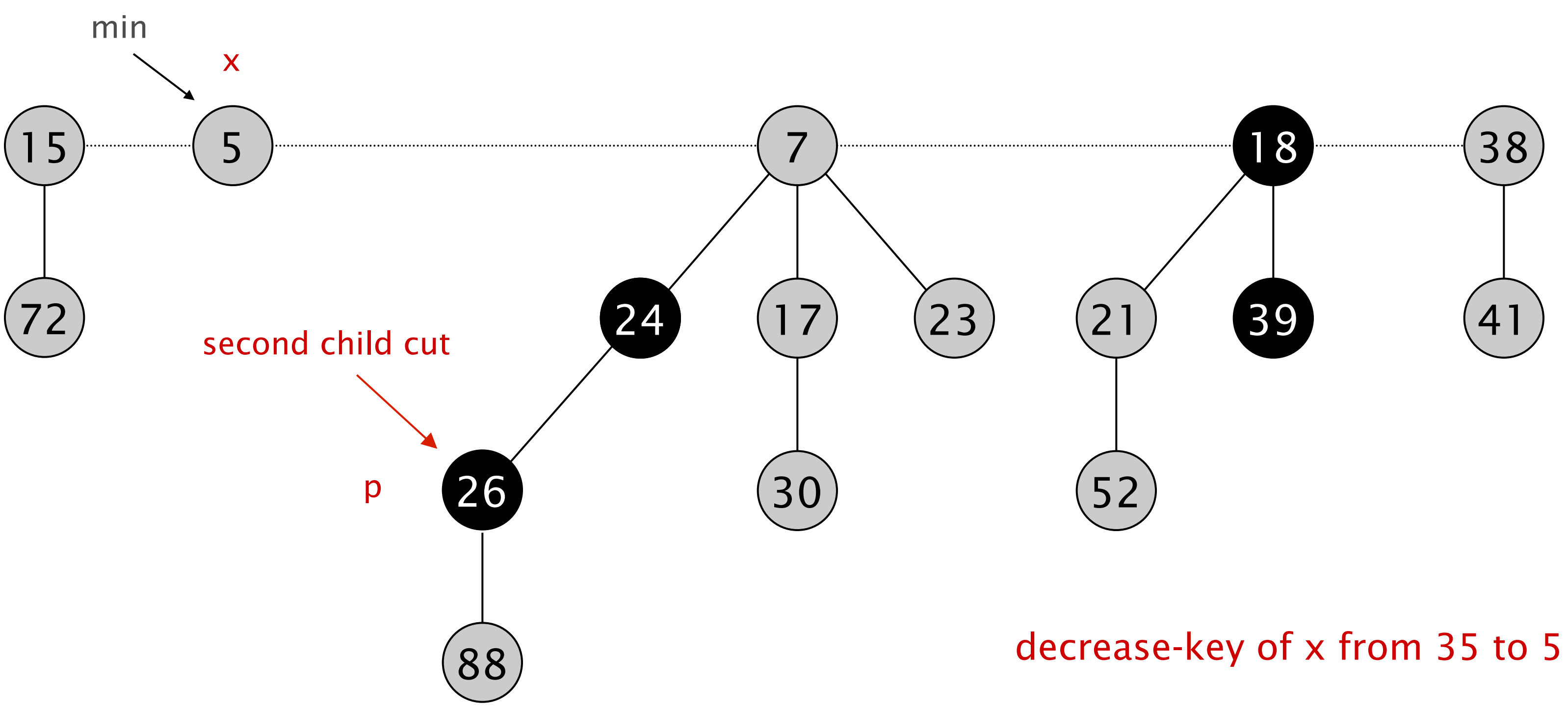
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

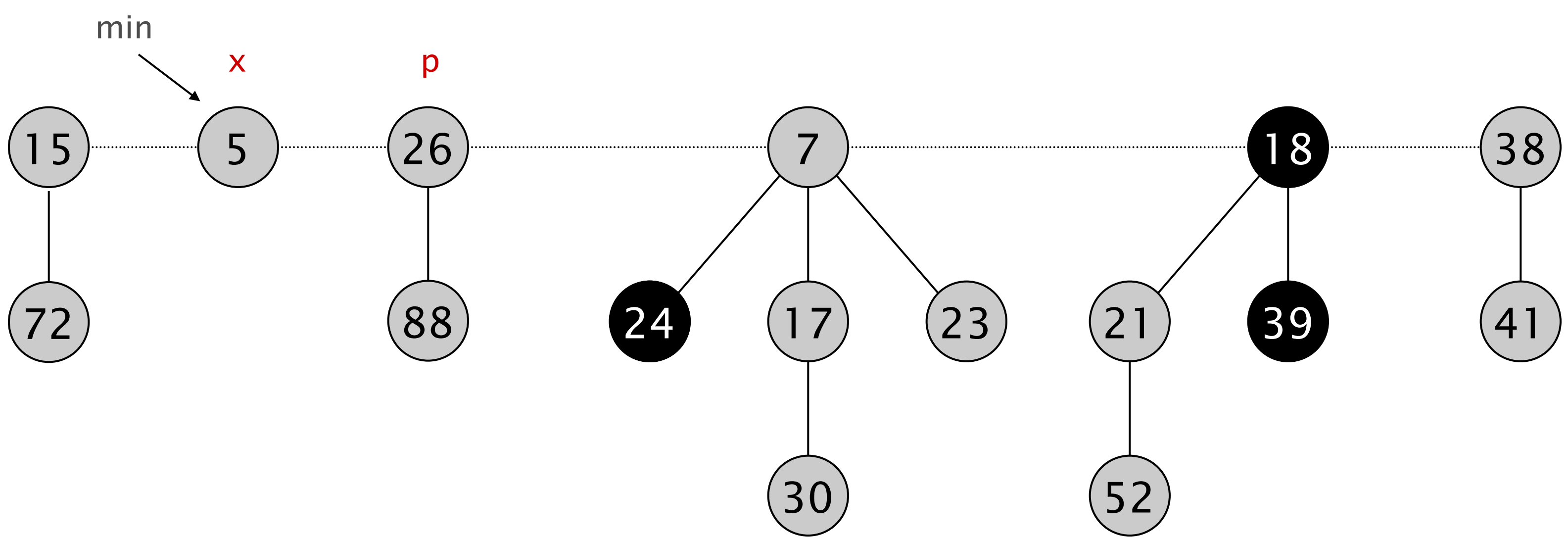
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



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).

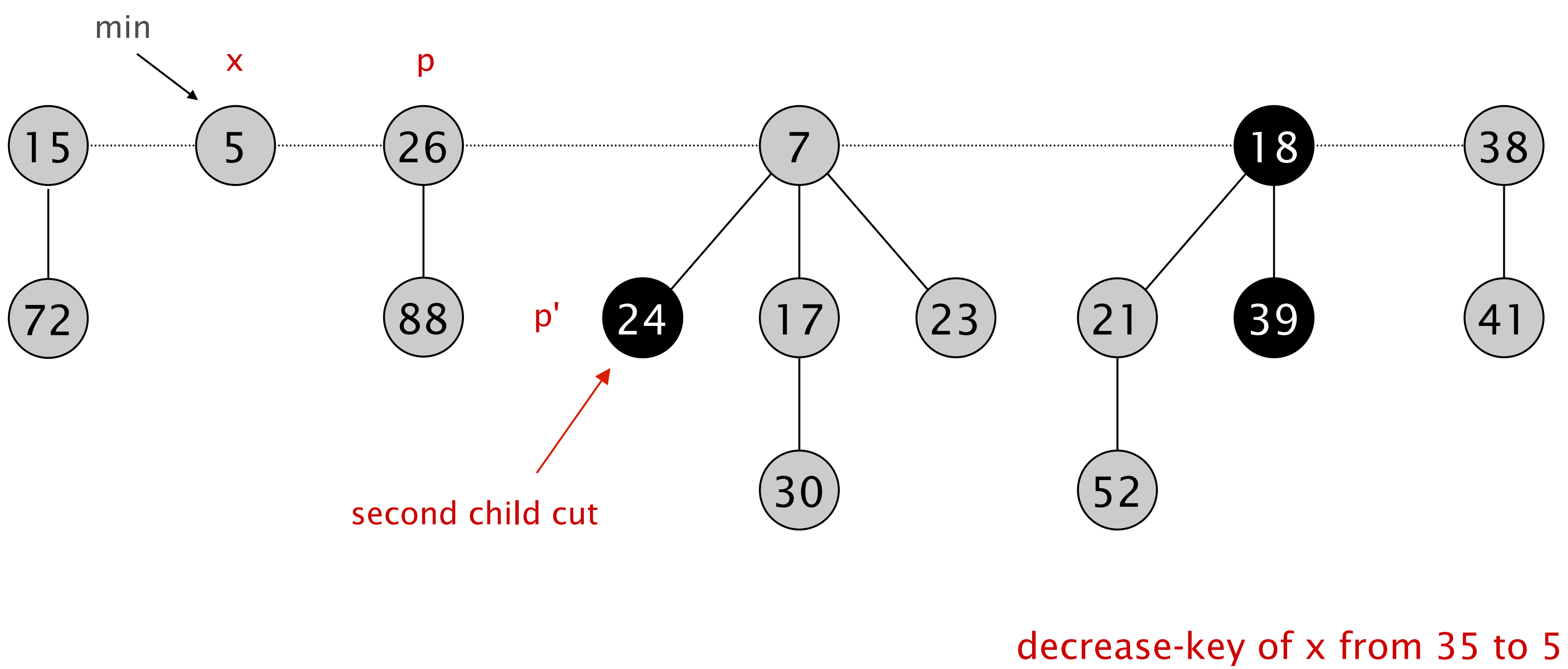


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

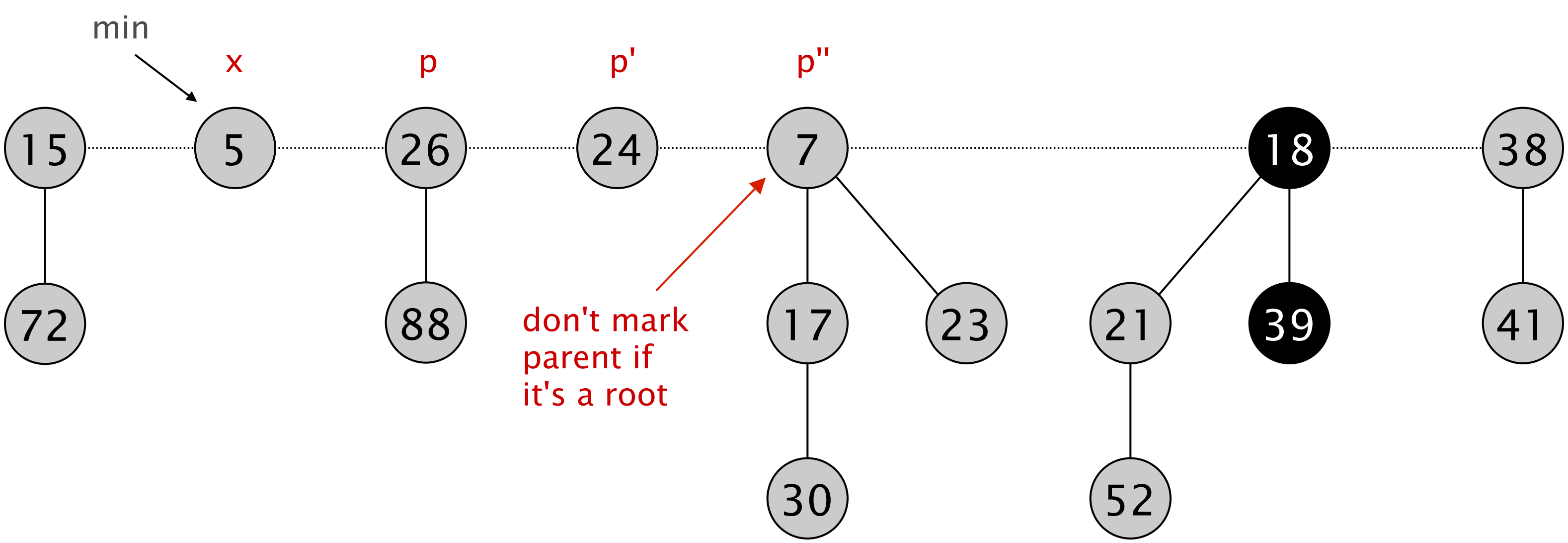
Case 2b. [heap order violated]

- 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).



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.

$$\Phi(H) = t(H) + 2m(H)$$

potential function

Actual cost $O(c)$, where c is the number of cascading cuts

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

Change in potential. $O(1) - c$

- $t(H') = t(H) + c$ (the original $t(H)$ees and c trees produced by cascading cuts)
- $m(H') \leq m(H) - c + 2$ (c-1 nodes were unmarked by the first c-1 cascading cuts and the last cut may have marked a node)
- Difference in potential $\Delta\Phi \leq c + 2(-c + 2) = 4 - c$

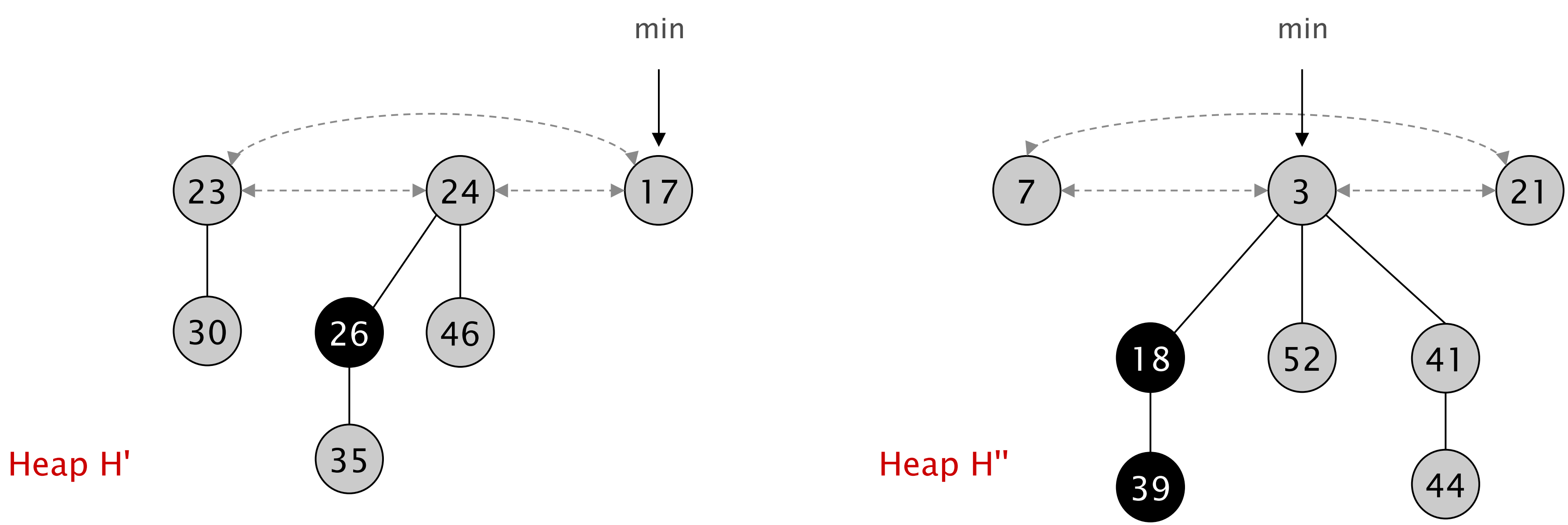
Amortized cost. $O(1)$

Union

Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

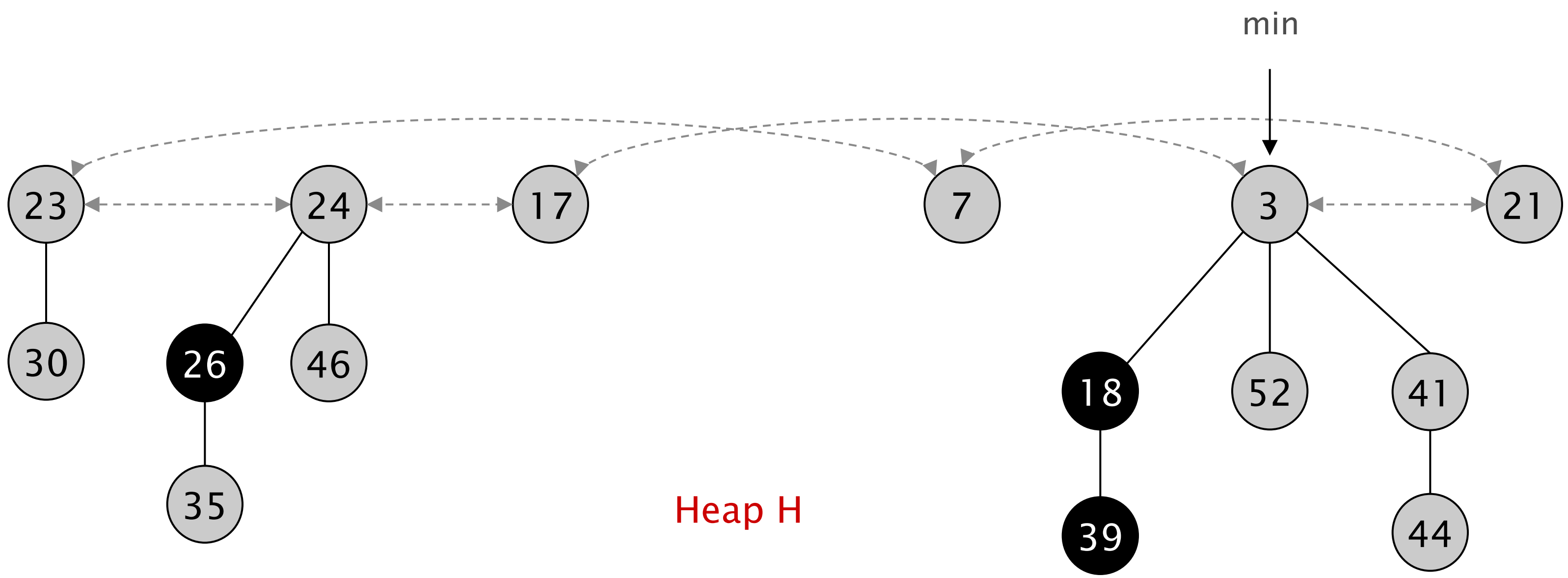
Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

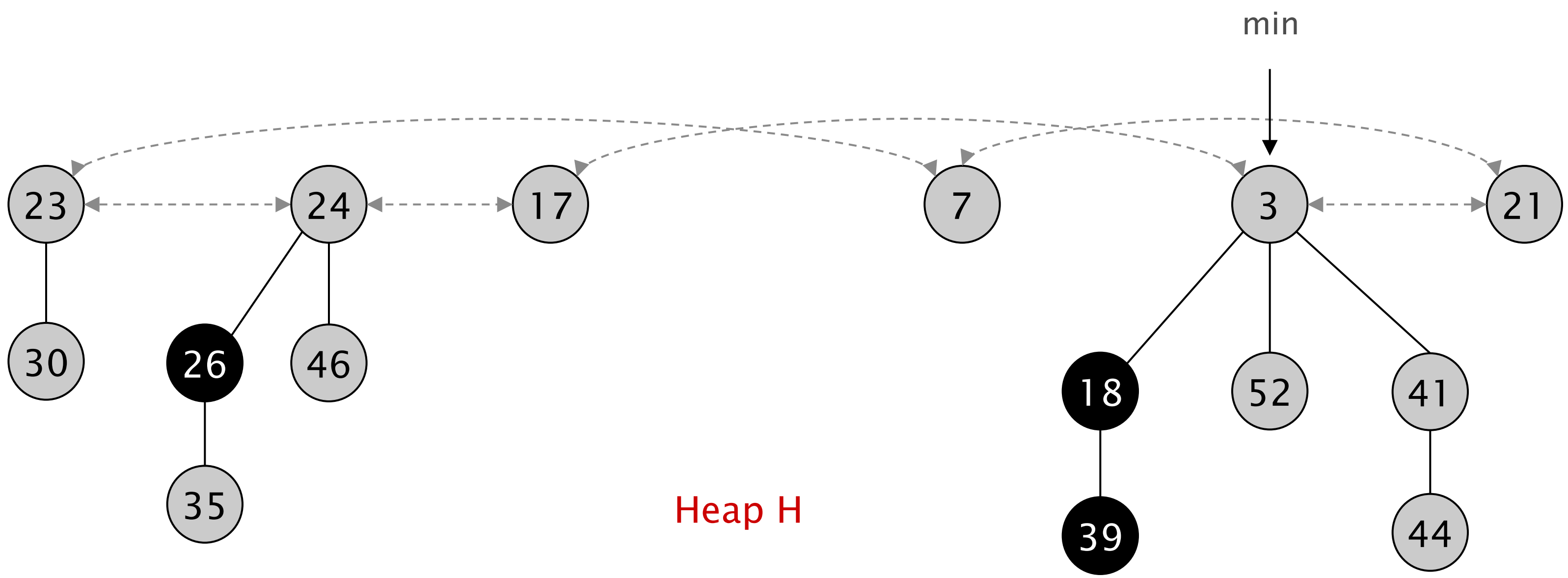
Actual cost: $O(1)$

$$\Phi(H) = t(H) + 2m(H)$$

potential function

Change in potential: 0 ($t(H)$ and $m(H)$ remain the same)

Amortized cost: $O(1)$



Delete

Fibonacci Heaps: Delete

$$\Phi(H) = t(H) + 2m(H)$$

potential function

Delete node x.

- decrease-key of x to -∞.
- extract-min element in heap.

Amortized cost $O(D(n))$

- $O(1)$ amortized for decrease-key.
- $O(D(n))$ amortized for extract-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
extract-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