FIBONACCI HEAP

Module 3

Introduction

- Introduce Fibonacci Heap
  - another example of mergeable heap
  - excellent amortized cost to perform each operation

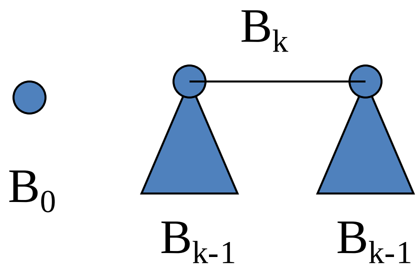
Binomial Trees

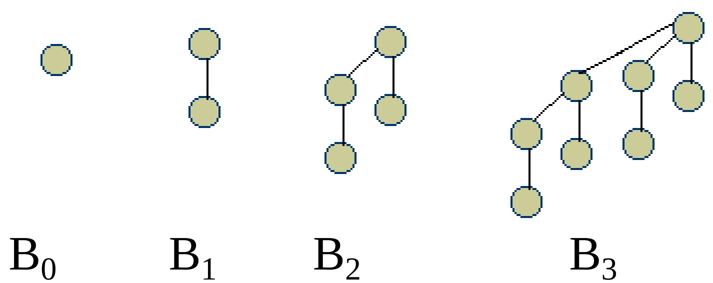
- A Binomial tree can be defined recursively where:

  A binomial tree B_k consists of two B_{k-1} binomial trees that are linked together. The root of one is the leftmost child of the root of the other.

- Properties of a binomial tree are given by the following lemma:

  1. there are 2^k nodes
  2. the height of the tree is k
  3. there are exactly $^{K}C_{3}$ nodes at depth I for I = 0, 1, …, k and
  4. the root has degree k, which is greater than that of any other node





What is a Fibonacci Heap?….. (cont.)

- Collection of unordered Binomial Trees.

- Support Mergeable heap operations such as Insert, Minimum, Extract Min, and Union in constant time O(1)

- Desirable when the number of Extract Min and Delete operations are small relative to the number of other operations.

- Most asymptotically fastest algorithms for computing minimum spanning trees and finding single source shortest paths, make use of the Fibonacci heaps.

Fibonacci Heap

- Like binomial heap, Fibonacci heap consists of a set of min-heap ordered component trees
- However, unlike binomial heap, it has
  - no limit on nodes of a trees , and
  - no limit on height of a tree (up to O(n))

Fibonacci Heap

- Consequently,
  - Find-Min, Extract-Min, Union,
  - Decrease-Key, Delete
- all have worst-case $O(n)$ running time

- However, in the amortized sense, each operation performs very quickly

Amortized analysis ……. a quick definition

The complexity analysis of Fibonacci Heaps is heavily dependent upon the "potential method" of amortized analysis

- The time required to perform a sequence of data structure operations is *averaged* over all the operations performed.

- Can be used to show that *average cost* of an operation is small, if you average over a sequence of operations, even though a single operation might be expensive

Comparison of Three Heaps

	Binary (worst-case)	Binomial (worst-case)	Fibonacci (amortized)
Make-Heap	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Find-Min	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$
Extract-Min	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
Insert	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
Delete	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
Decrease-Key	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
Union	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$

DEFINITION OF FIBONACCI HEAP

- Fibonacci heap is a heap data structure consisting of collection of trees

- Collection of **min-heap tree-trees** in a Fibonacci heap are not constrained to be binomial trees.

- Unlike binomial heap,fibonacci heap can have many trees of same degree and a **tree does not have exactly 2^i nodes.**

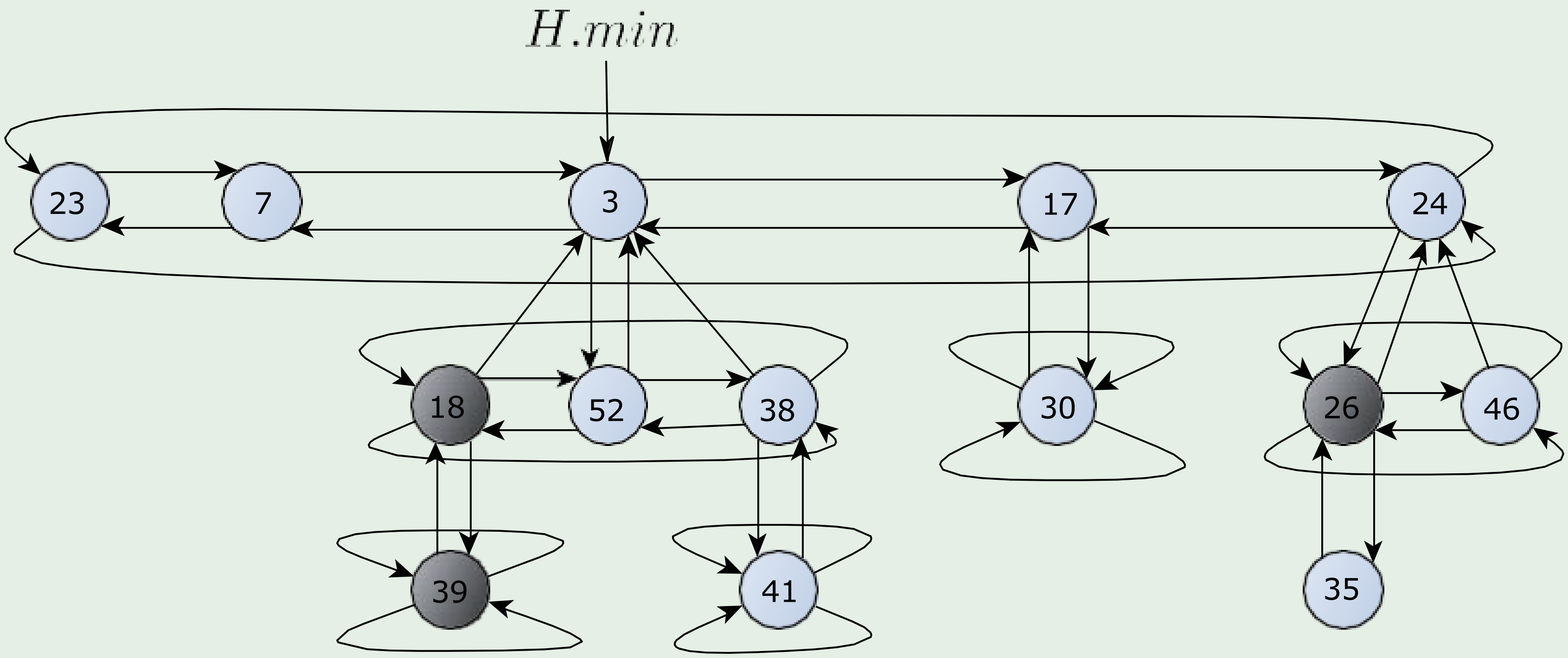
- Unlike trees with binomial heap which are ordered, tree in a fibonacci heap are rooted but unordered

DEFINITION OF FIBONACCI HEAP

- Each nodes will store a degree ie number of children
- **Min(H)** is a pointer pointing to **minimum node in root list.**
- **Each node also has mark(x), a boolean field indicating whether x has lost a child since x was made child to other node.**
- **Newly created nodes are unmarked.**
- **The number of children in child list of node x is stored in the degree(x)**

Fibonacci Structure

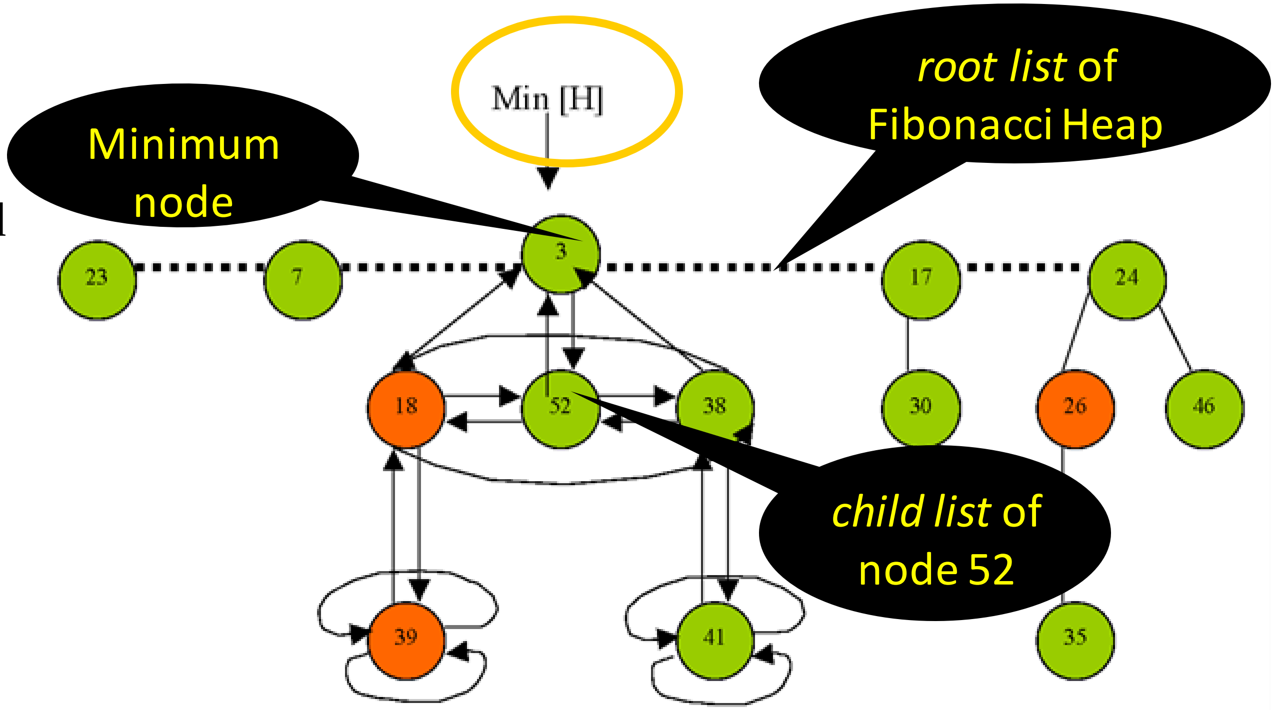
Example



What is a Fibonacci Heap?

data structure… advantages of removal & concatenation

- Heap ordered Trees
- Rooted, but unordered
- Children of a node are linked together in a Circular, doubly linked List, E.g node 52

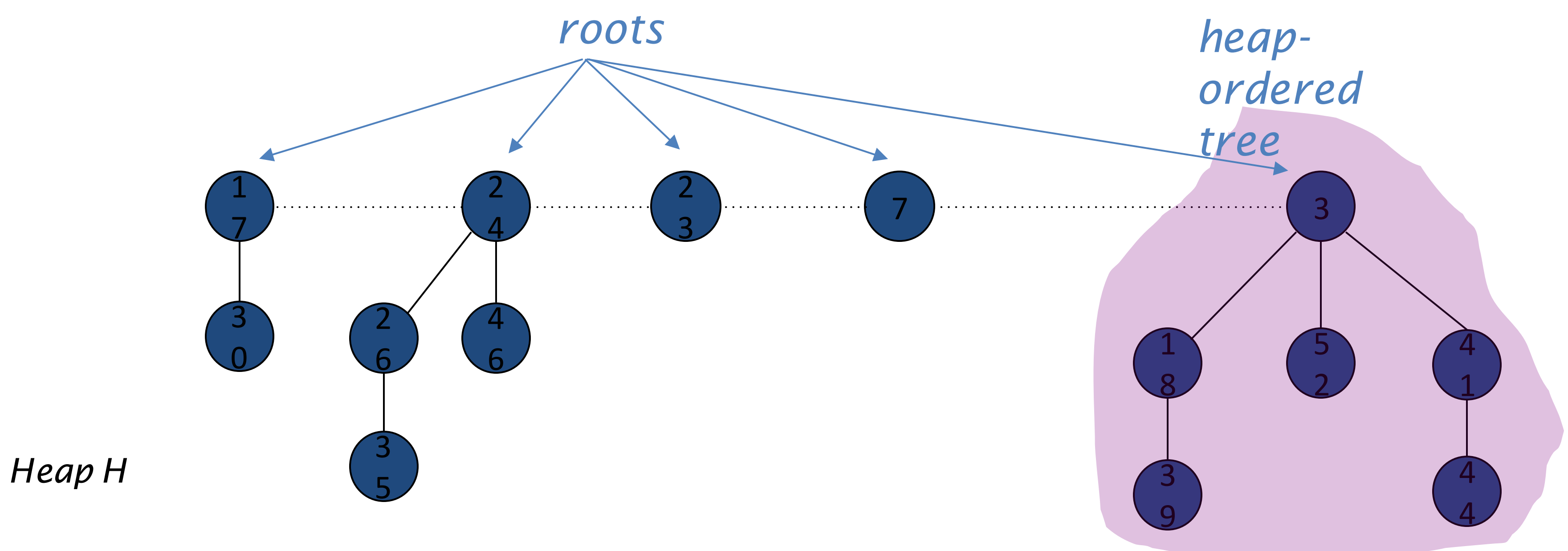


Node Pointers
- left [x]
- right [x]
- degree [x] - number of children in the child list of x
- mark [x]

Fibonacci Heaps: Structure

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

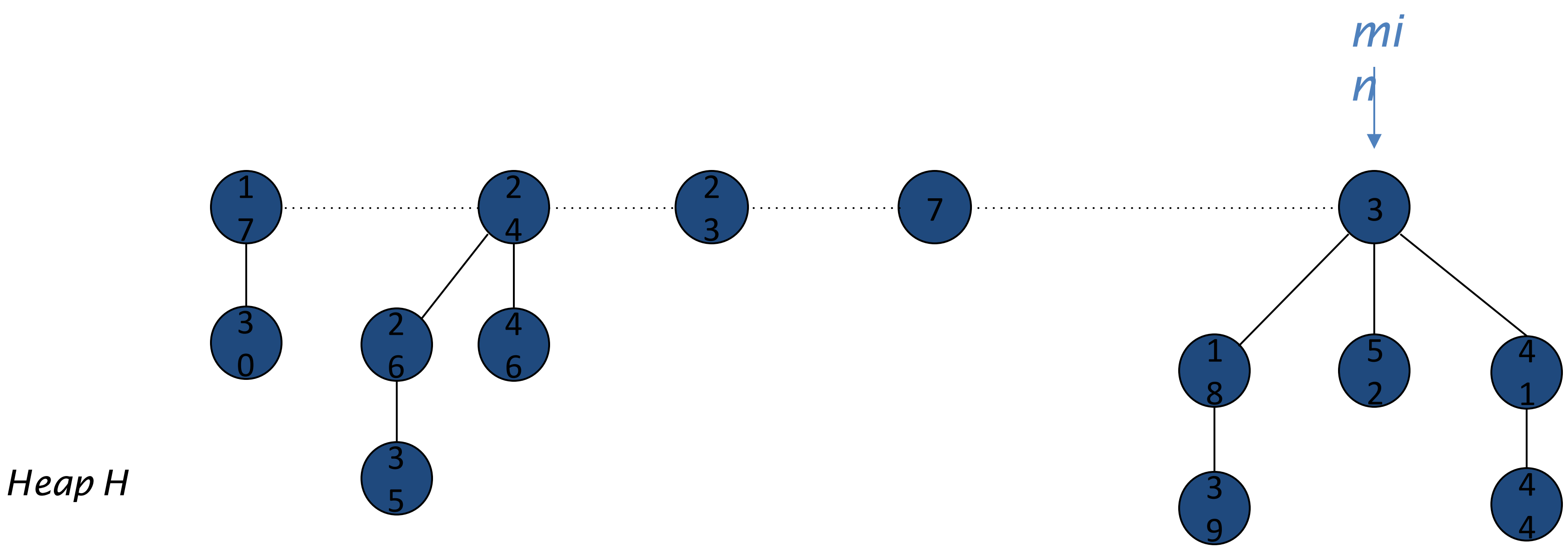
each parent larger than its children



Fibonacci Heaps: Structure

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

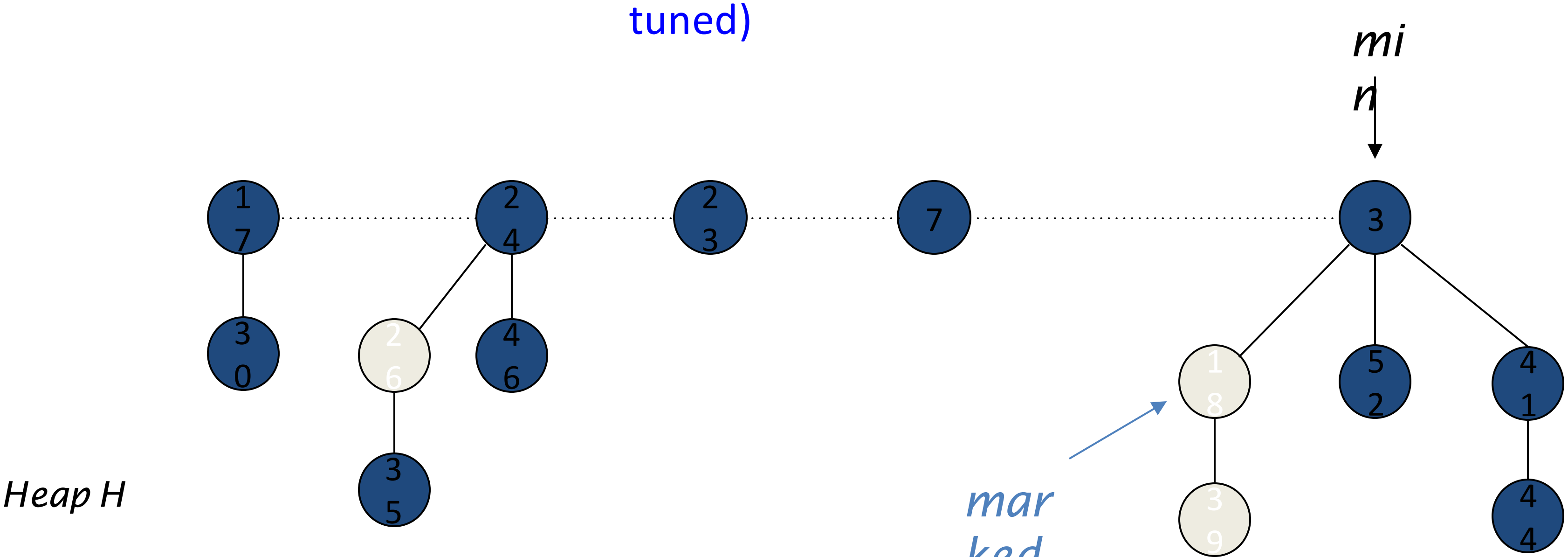
find-min takes O(1) time



Fibonacci Heaps: Structure

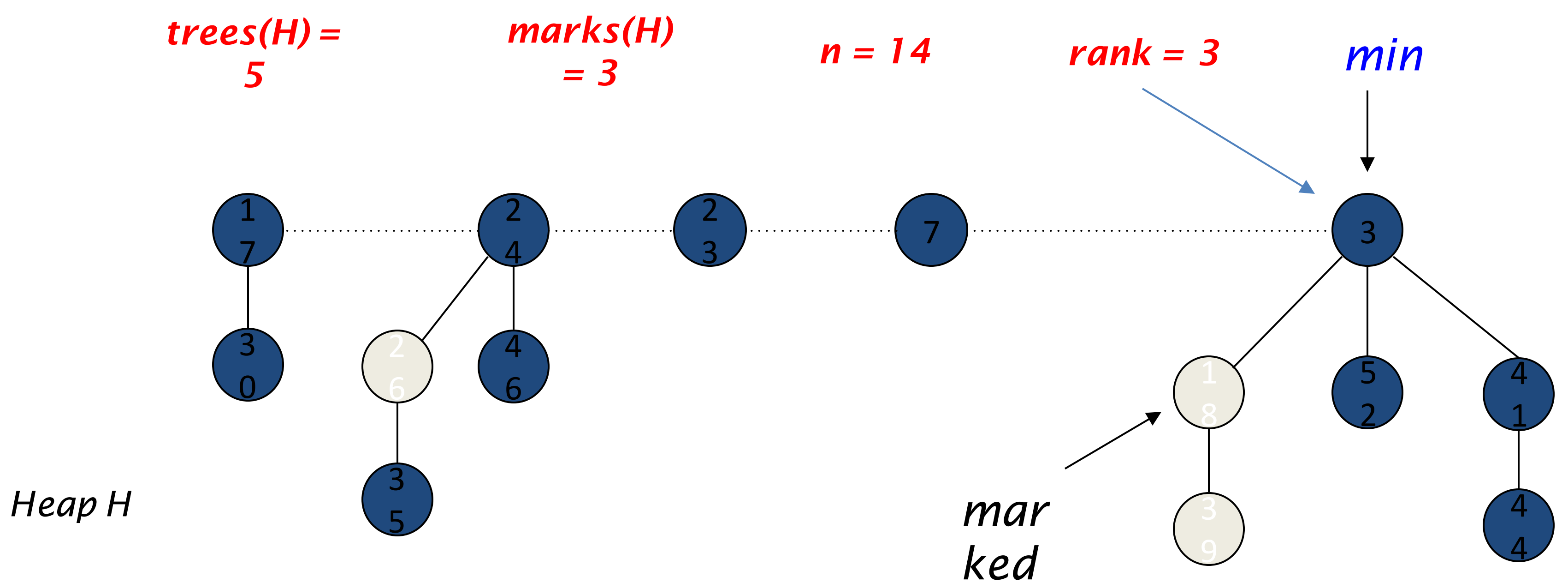
- Fibonacci heap.
  - Set of heap-ordered trees.
  - Maintain pointer to minimum element.
  - Set of marked nodes.

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.



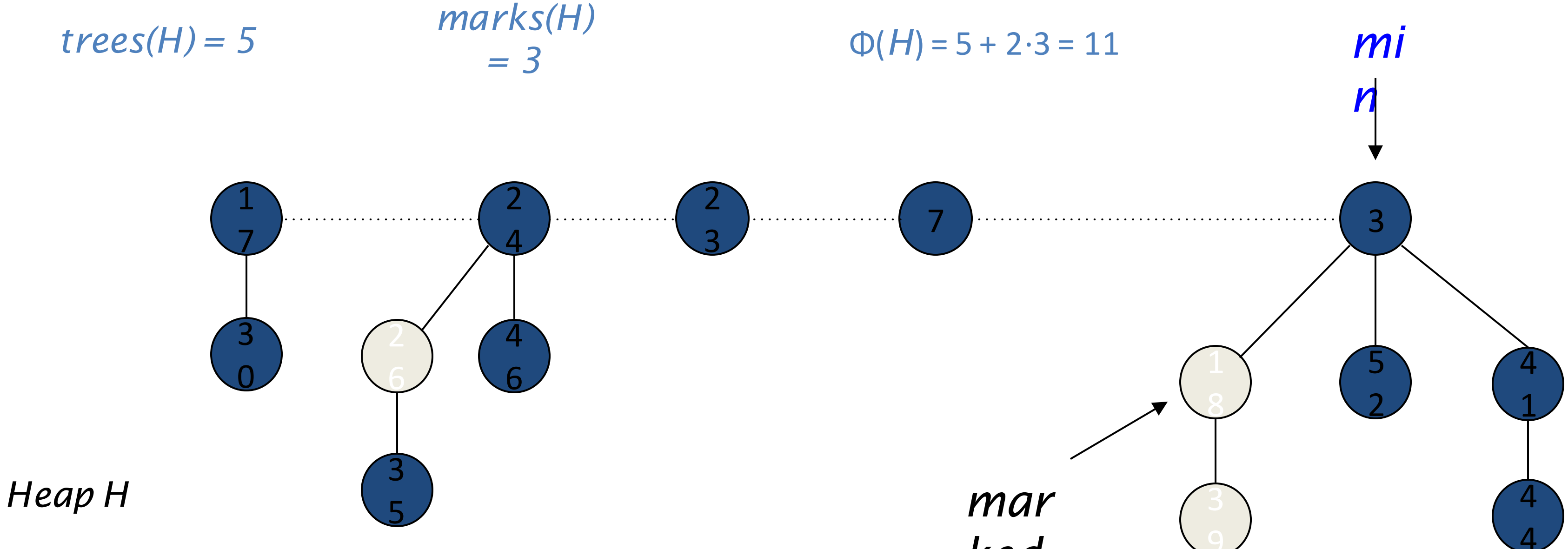
Fibonacci Heaps: Potential Function

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

potential of heap H

Where tree(H) = no.of root nodes and mark(H)=no. of marked nodes

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



Creating a new Fibonacci Heap

The **MAKE FIB HEAP()** creates a new empty fibonacci heap

- It allocates and return the Fib Heap object H where
- n(H)=0 and min(H)=NIL
- indicates that there are no trees in H
- The Amortized cost of MAKE FIB HEAP() is thus equals to O(1) ie Actual Cost

Amortized cost=Actual Cost +Potential Function

- **The potential function of empty fibonacci heap is**

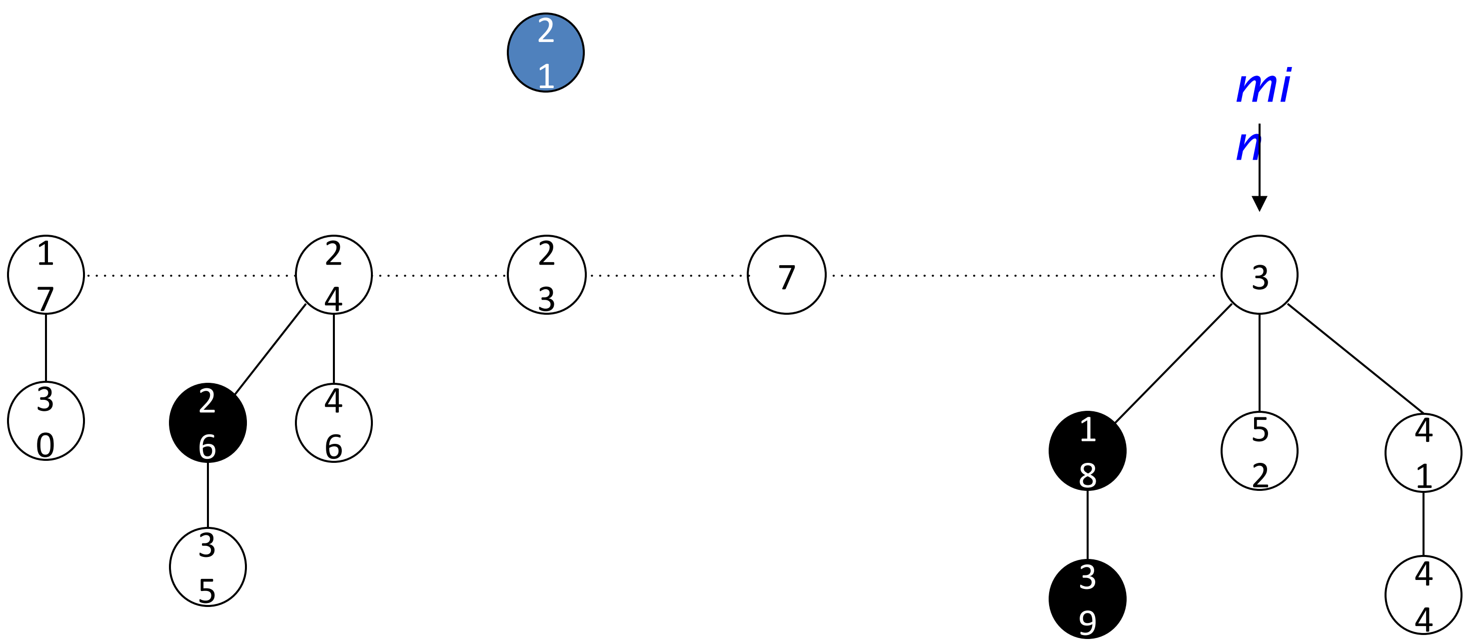
$$\Phi(H) = \mathit{trees}(H) + 2 \cdot \mathit{marks}(H)$$

Where tree(H) = no.of root nodes and mark(H)=no. of marked nodes

Fibonacci Heaps: Insert

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

insert 21

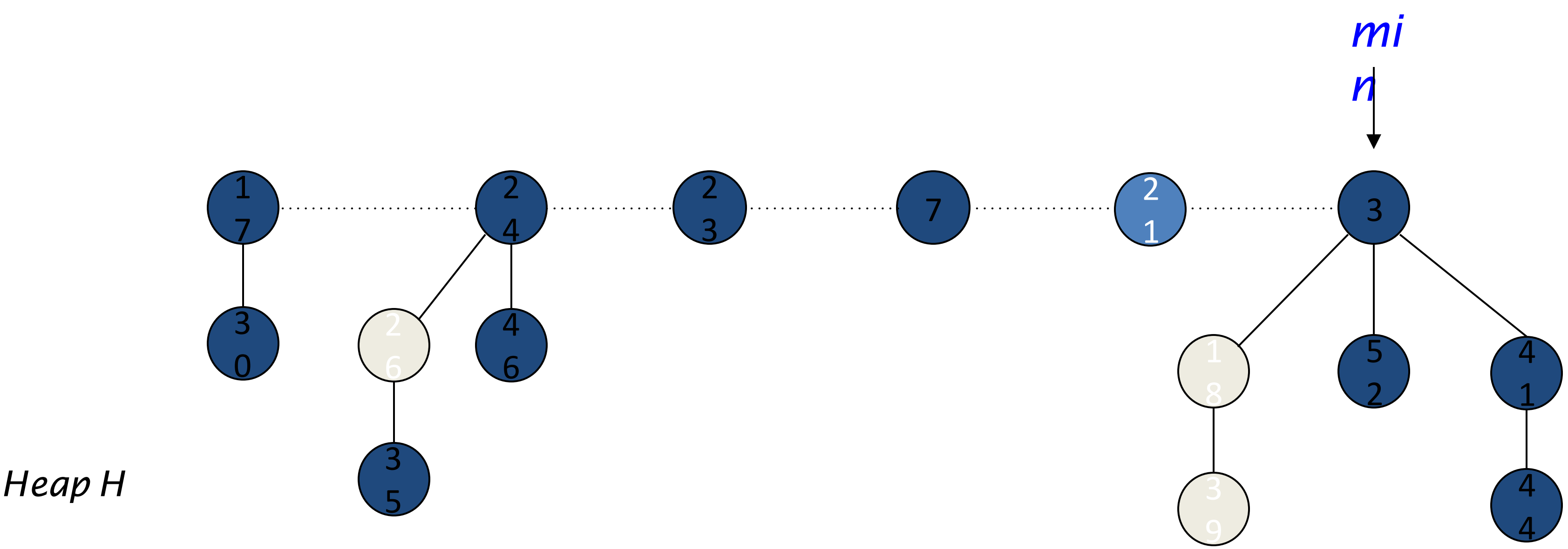


Heap H

Fibonacci Heaps: Insert

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

insert 21

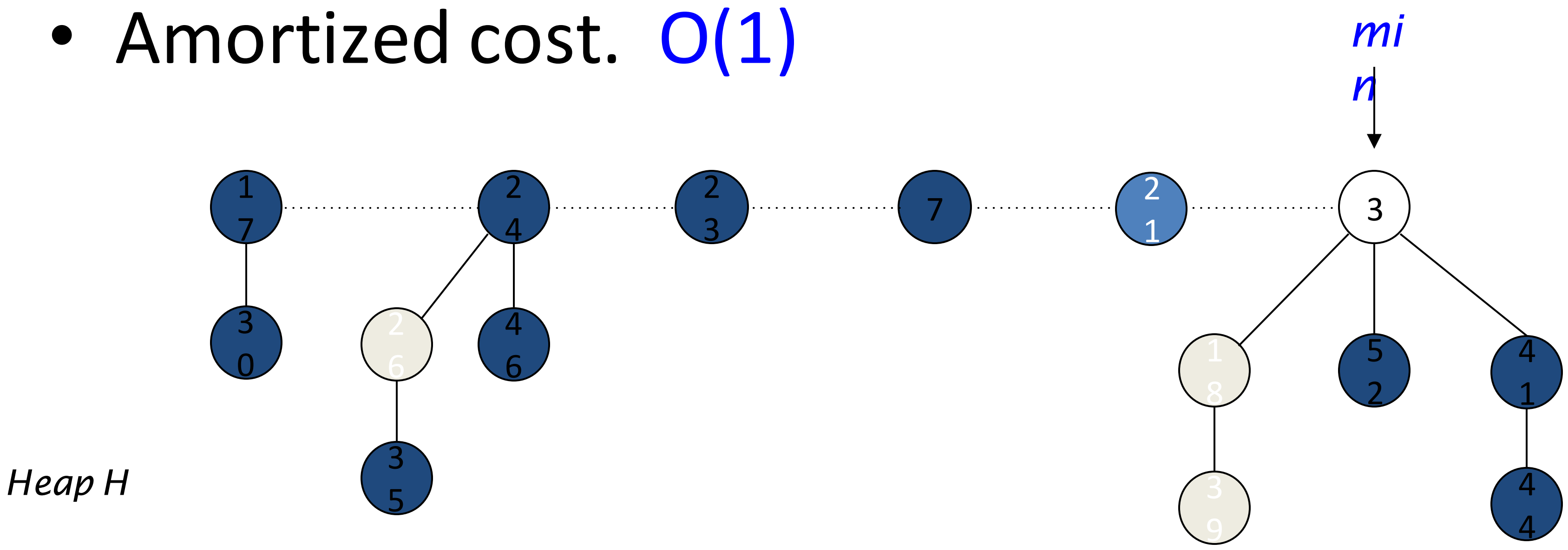


Fibonacci Heaps: Insert Analysis

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

potential of heap H

- Actual cost. O(1)
- Change in potential. +1
- Amortized cost. O(1)



Analysis -insert

- Let H = input Fibonacci heap and H'= resulting Fibonacci heap
- tree(H') $= trees(H)$ +1 and mark(H')= $marks(H)$

Insertion

Code for Inserting a node

Fib-Heap-Insert(H , x)

```
Degree(x) = 0
p(x) = NIL
child(x) = NIL
Left(x)=x
Right(x) =x x.mark =
FALSE  if H.min =
NIL
        Create a root list for H containing just x  H.min
        = x
else insert x into H's root list
        if x.key < H.min.key
                H.min = x
H.n = H.n + 1
```

1. *Degree(x)* = 0
2. *p(x) = NIL*
3. *child(x) = NIL*
4. *Left(x)=x*
5. *Right(x) =x*
6. *mark (x)= FALSE*
7. *Concatenate the root list of x with root list of h*
8. **if** *H.min* = *NIL*
9. Create a root list for *H* containing just *x* *H.min* = *x*
10. **else** insert x into *H's* root list
    11. **if** *x.key* < *H.min.key*
    12. *H.min* = *x*
13. *H.n* = *H.n* + 1

Union

- Concatenate te root list of H1 and H2 into new root list H
- Set the minimum node of H
- Set n(H) be the total number of nodes
- Analysis
- Change in potential
- $\Phi(H) = \Phi(H) - (\Phi(H) + \Phi(H))$

$\Phi(H) = trees(H) + 2 \cdot marks(H)$-[$trees(H) + 2 \cdot marks(H)$+$trees(H) + 2 \cdot marks(H)$]=0

Amortized Cost=Actual Cost+change in Potential

= O(1)+O(0)

=O(1)

Algorithm Union

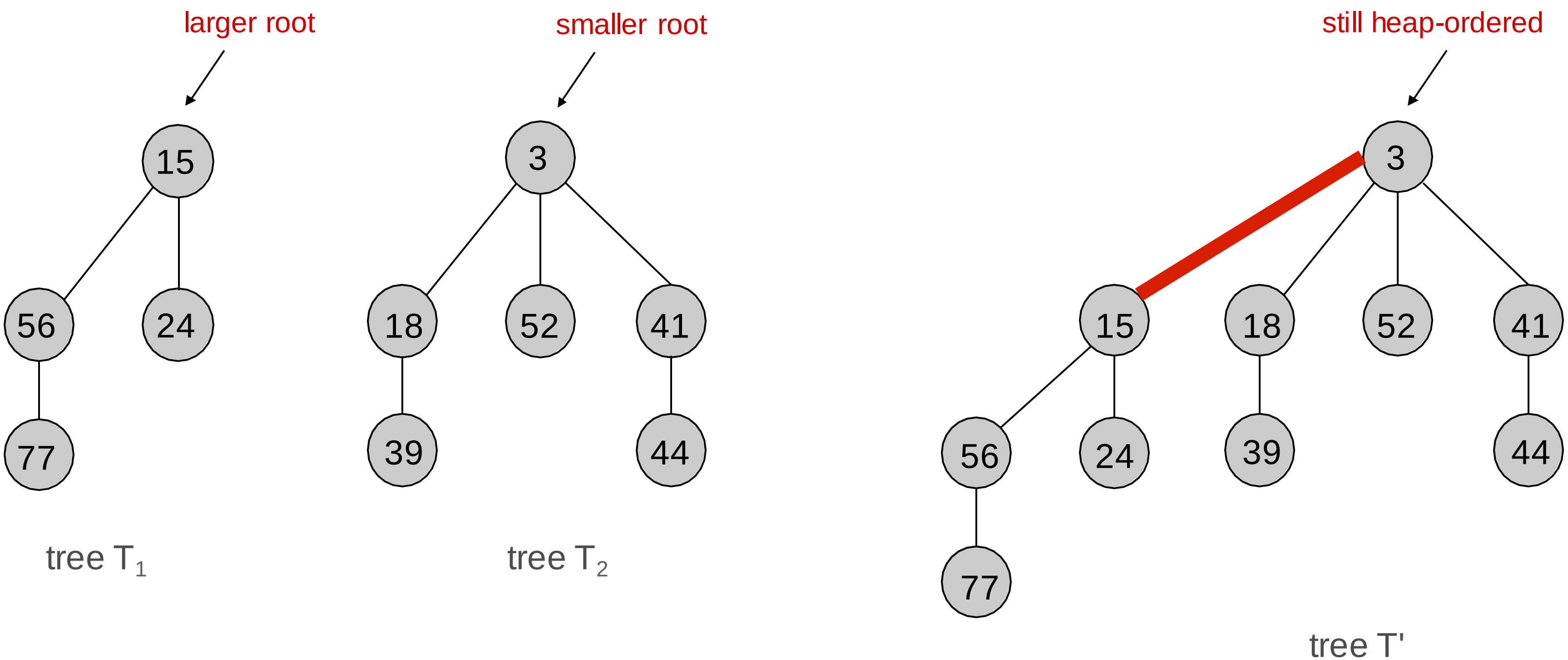
FIB-HEAP-UNION(H1,H2)

1. H← MAKE_FIB_HEAP()
2. Min(H)← min(H1)
3. Concatenate the root list of H2 with the root list of H
4. If (min(H1)==NIL or min(H2)!= NILand min(H2)<min(H1))
5. Then min(H)← min(H2)
6. N(H)← n(H1)+n(H2)
7. Free the object of H1 and H2
8. Return H

Extract-Min

Linking Operation

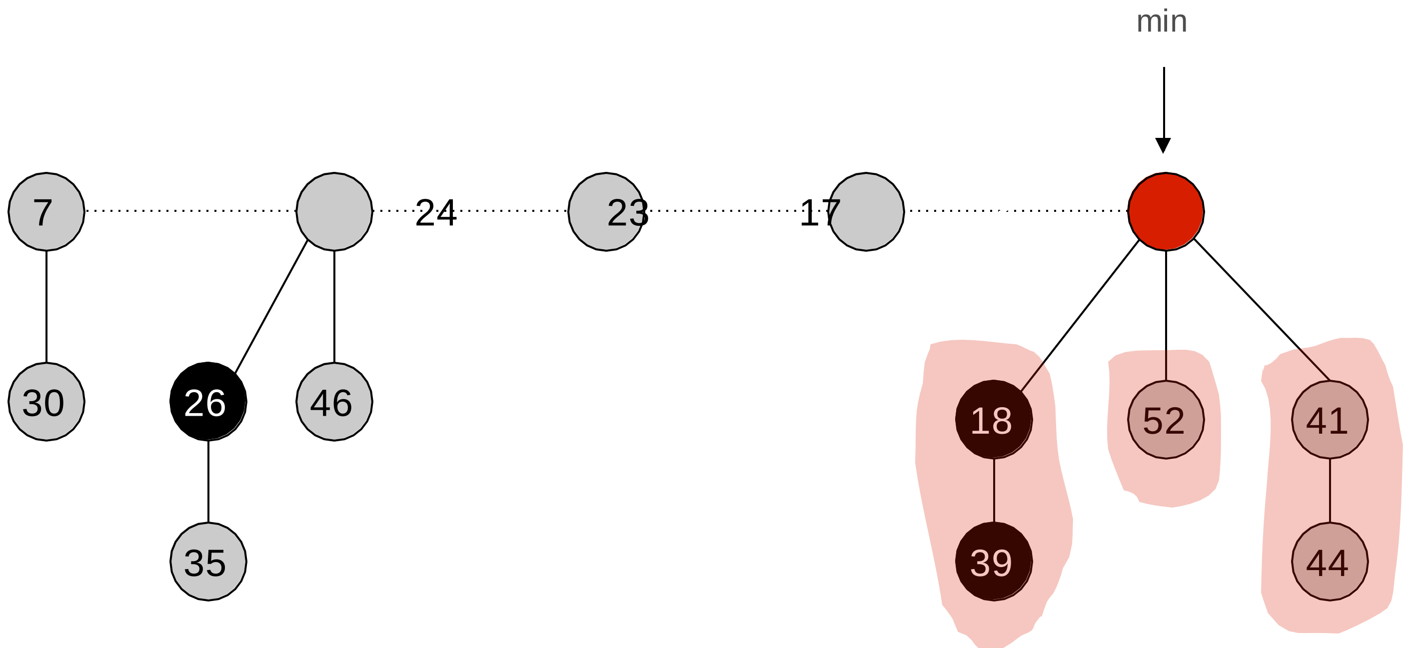
Linking operation. Make larger root be a child of smaller root.



Fibonacci Heaps: Extract-Min

Extract-min.

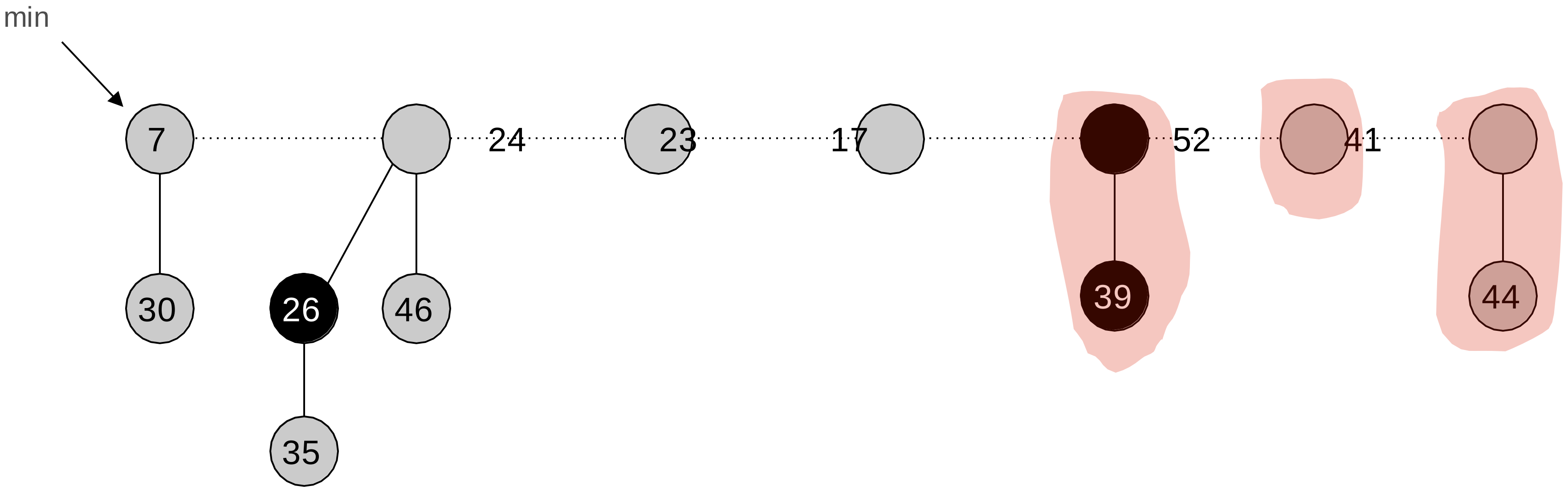
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

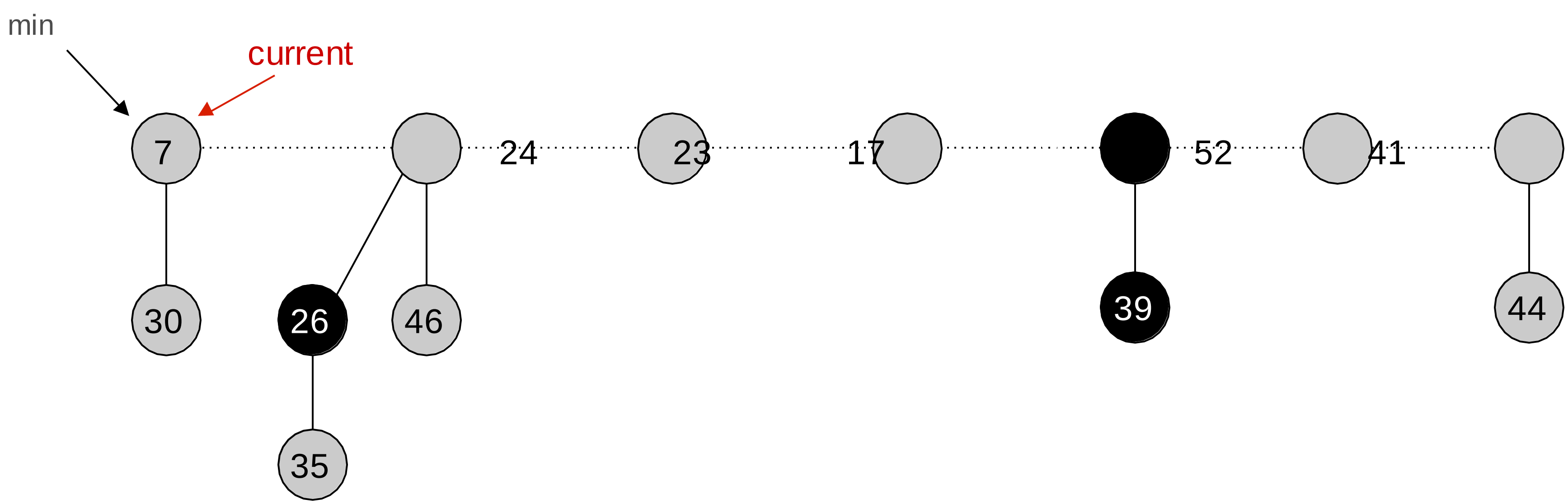
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

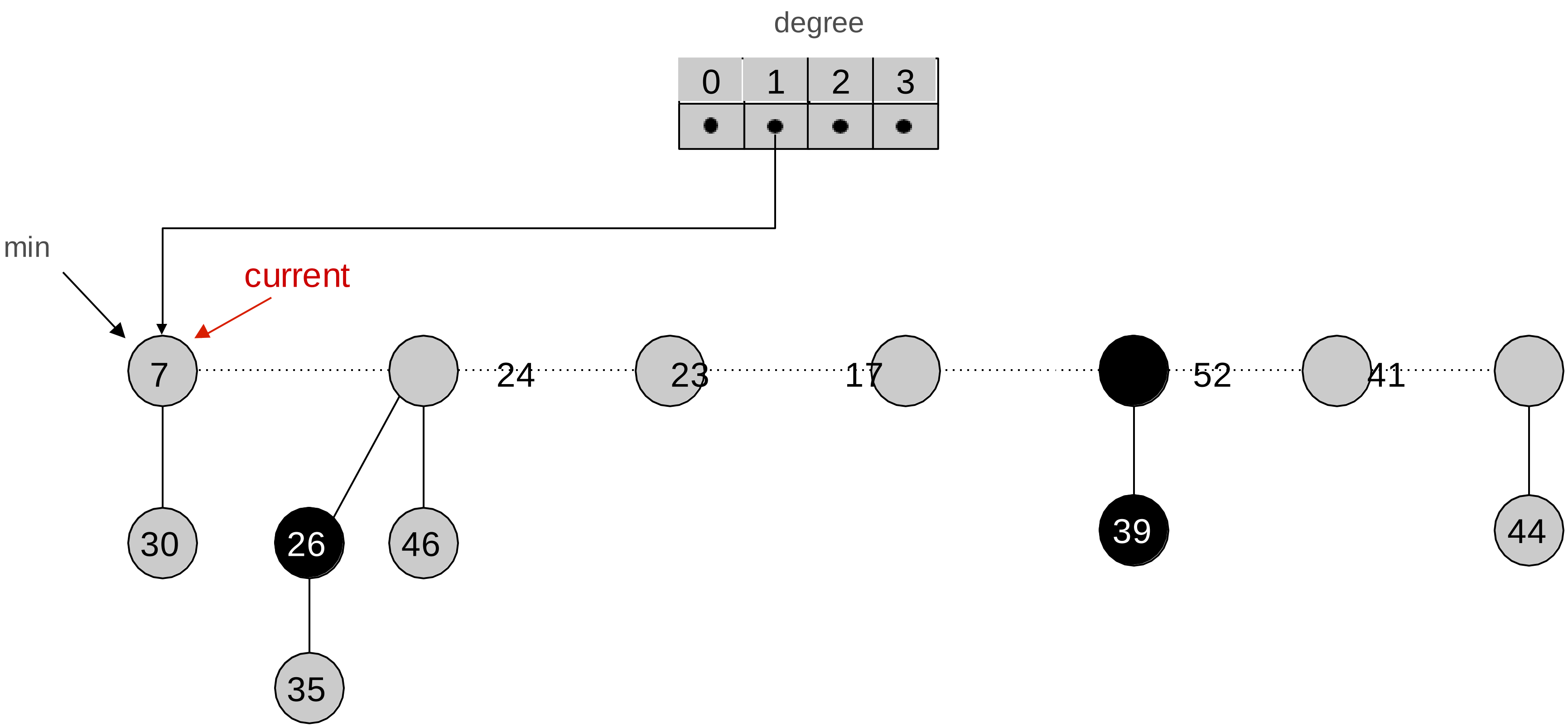
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

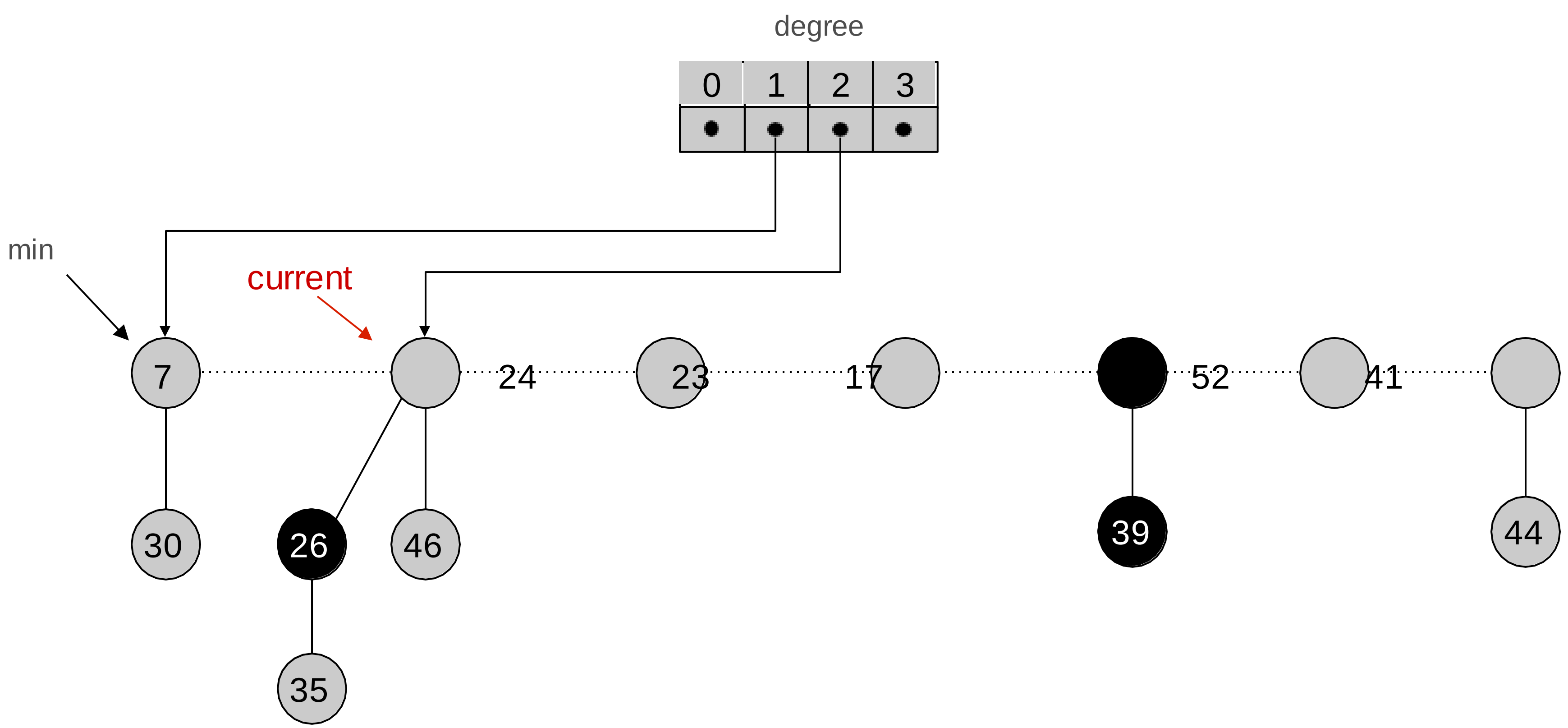
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

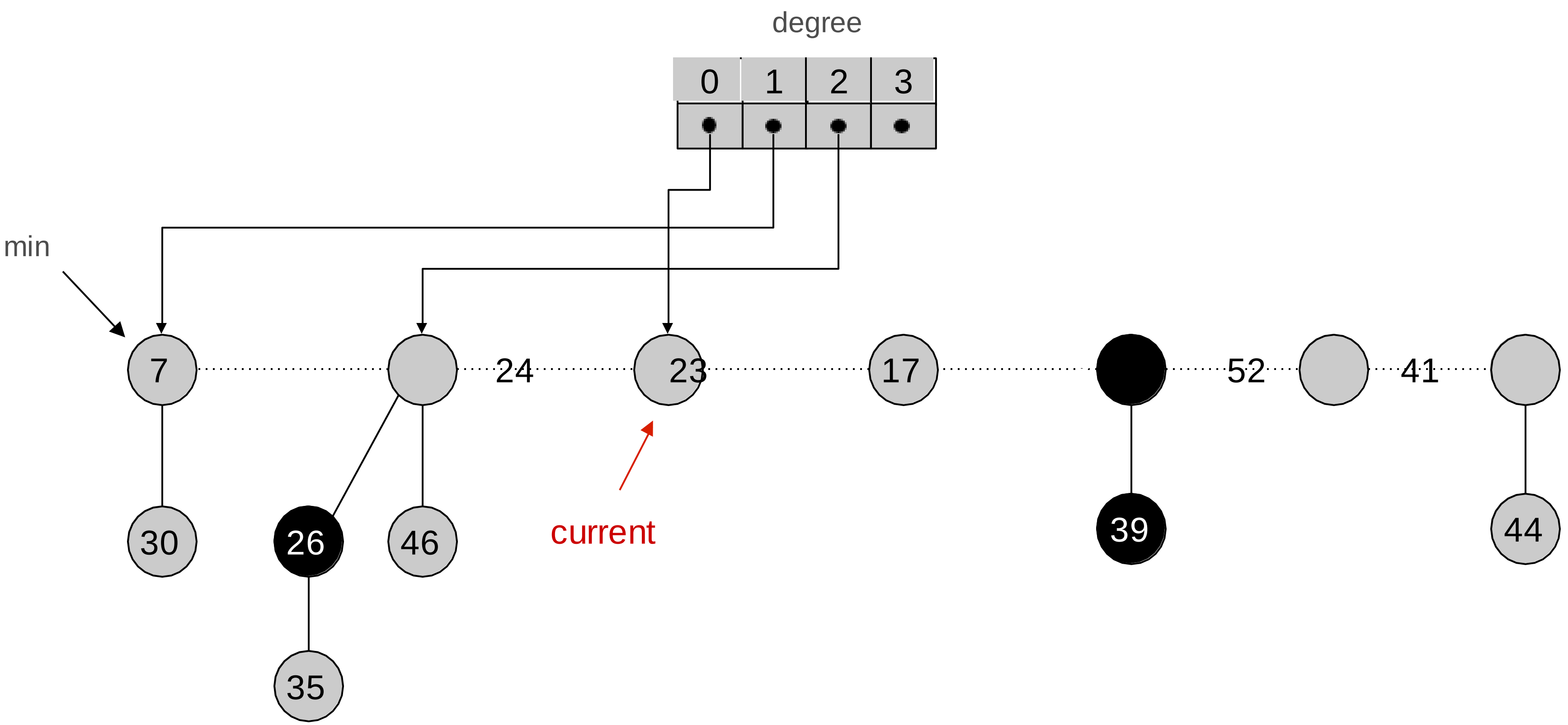
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

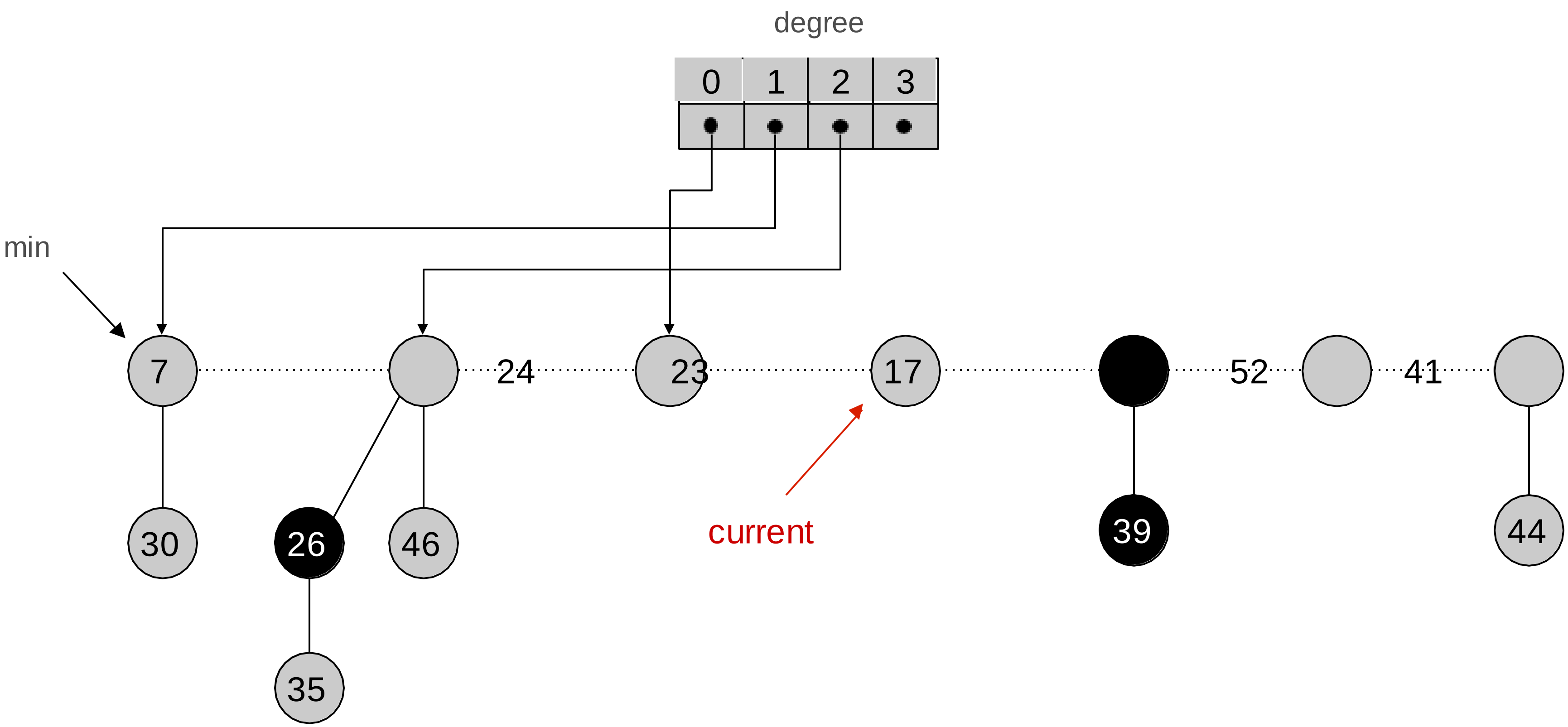
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



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.

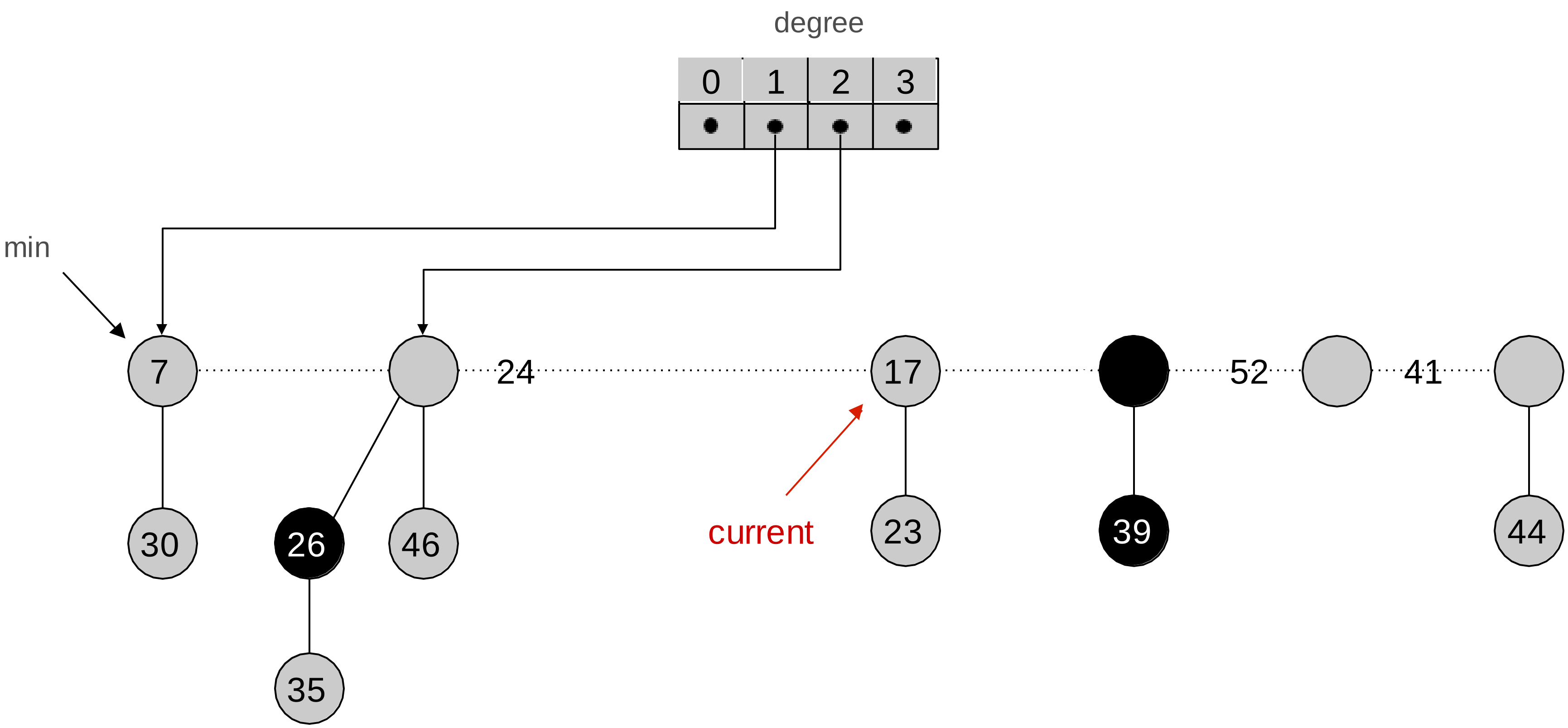


link 23 into 17

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.

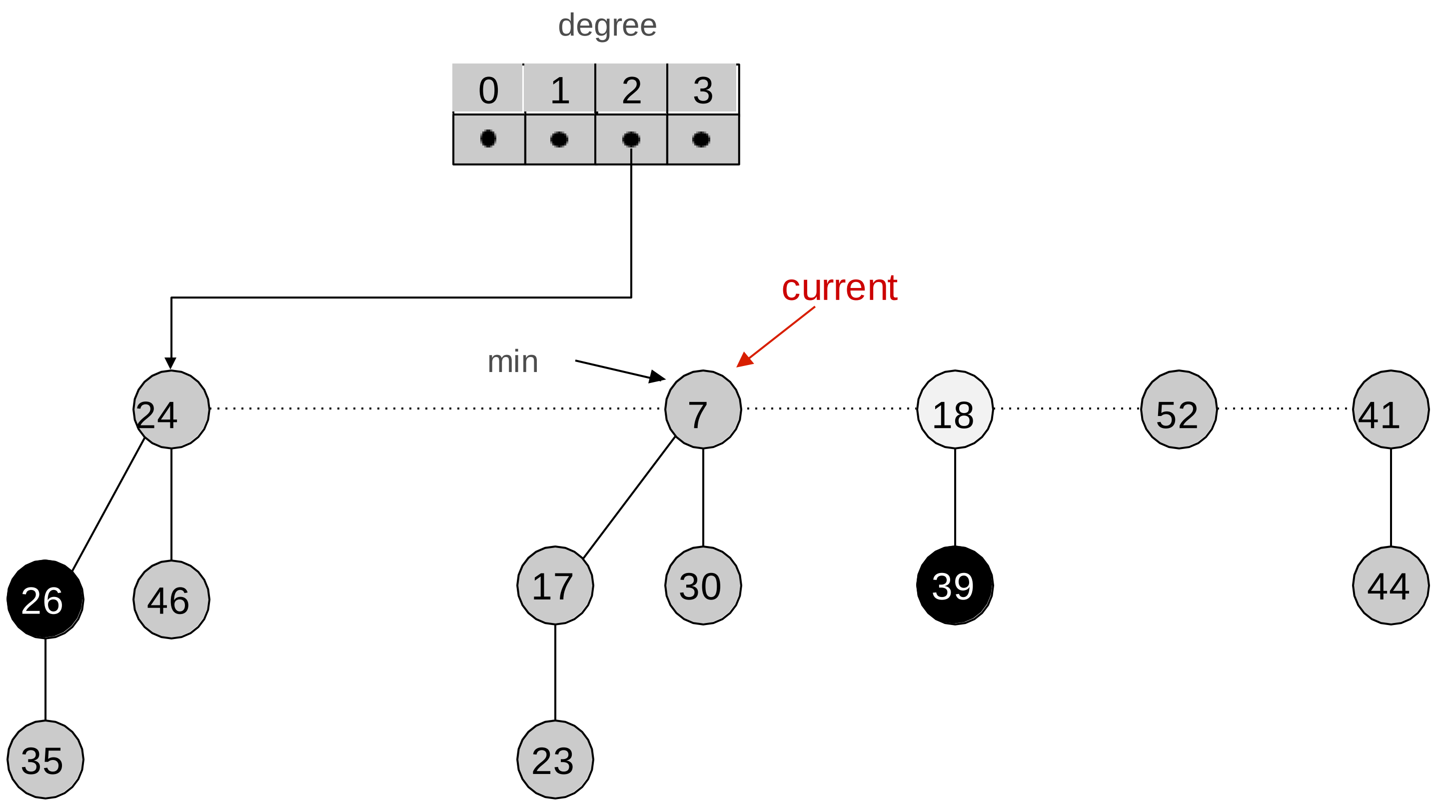


link 17 into 7

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.

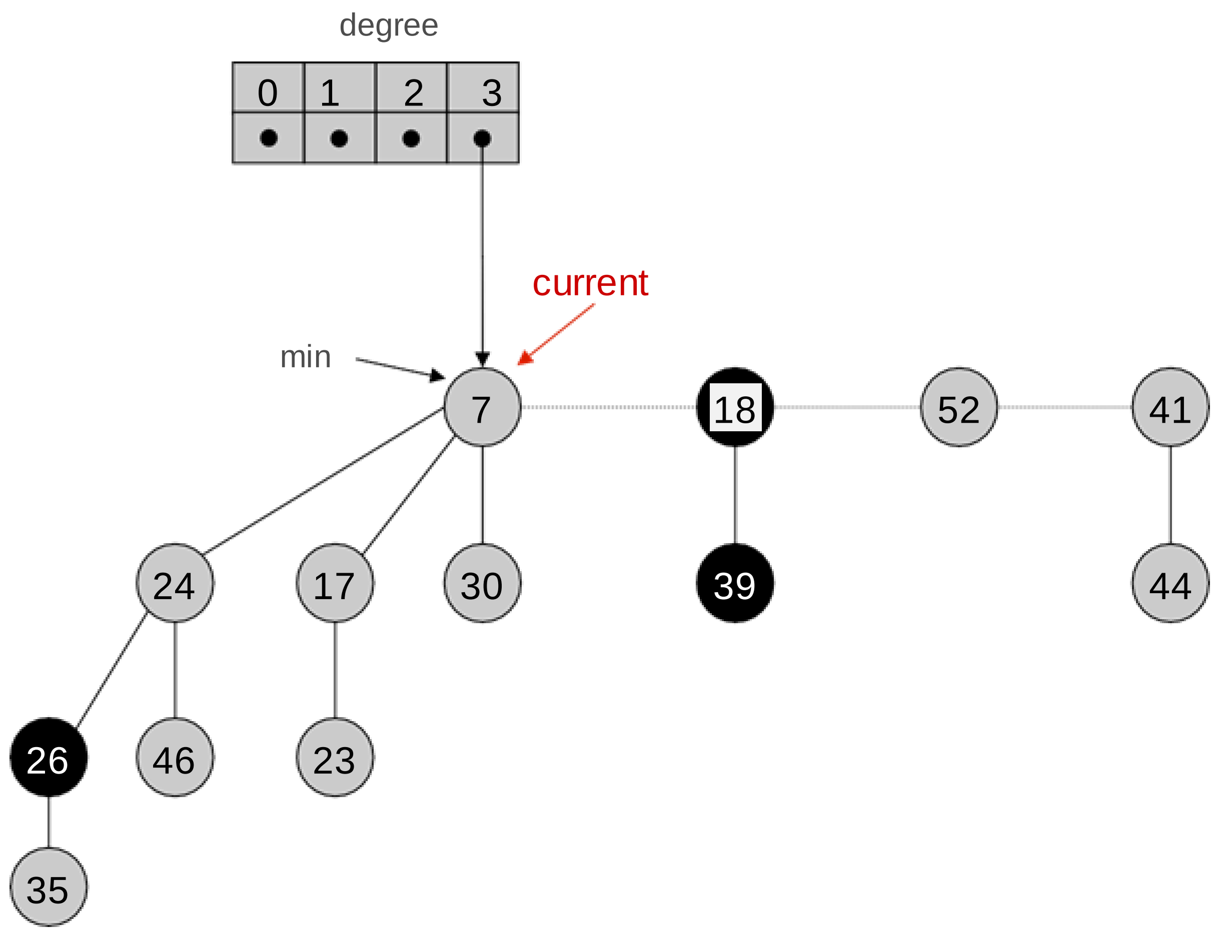


link 24 into 7

Fibonacci Heaps: Extract-Min

Extract-Min.

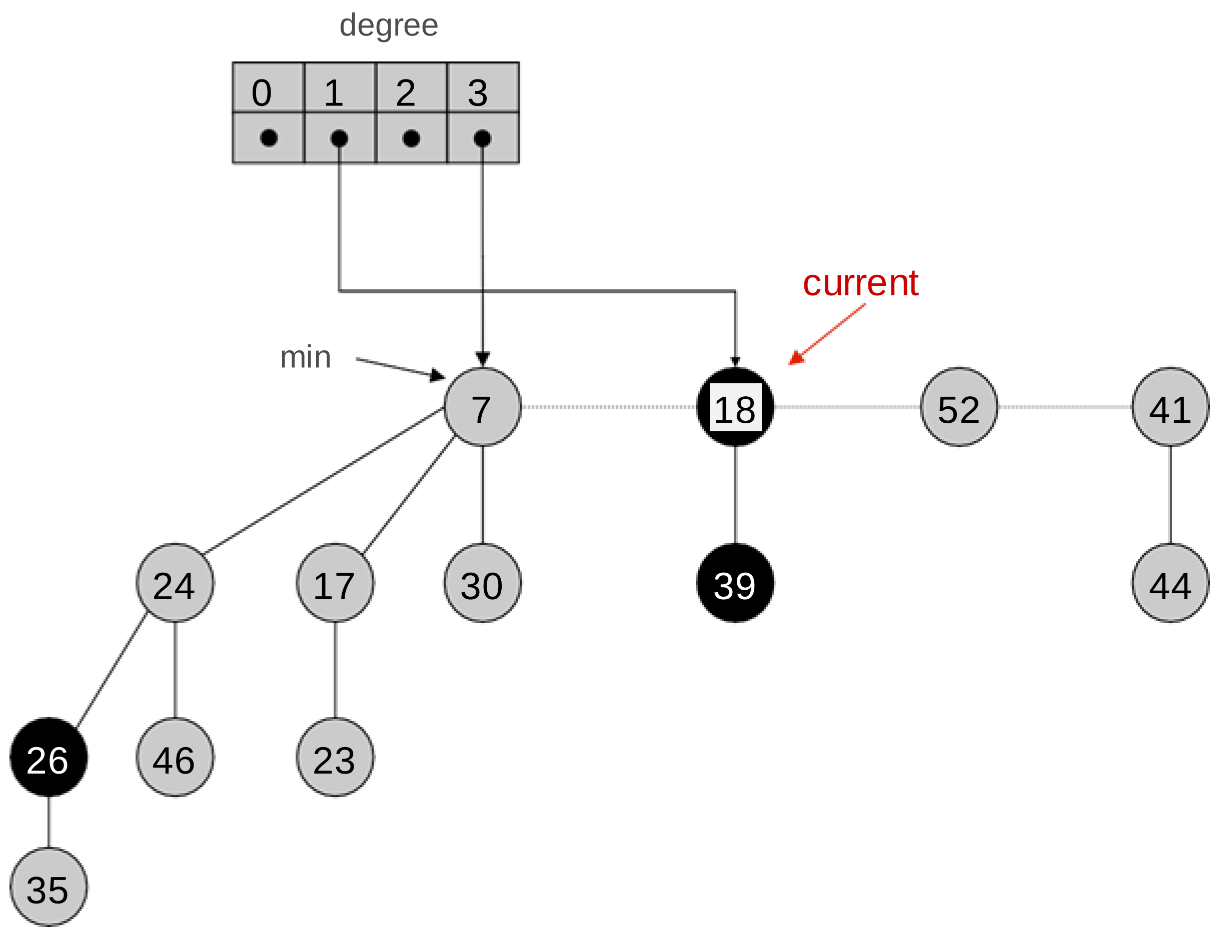
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

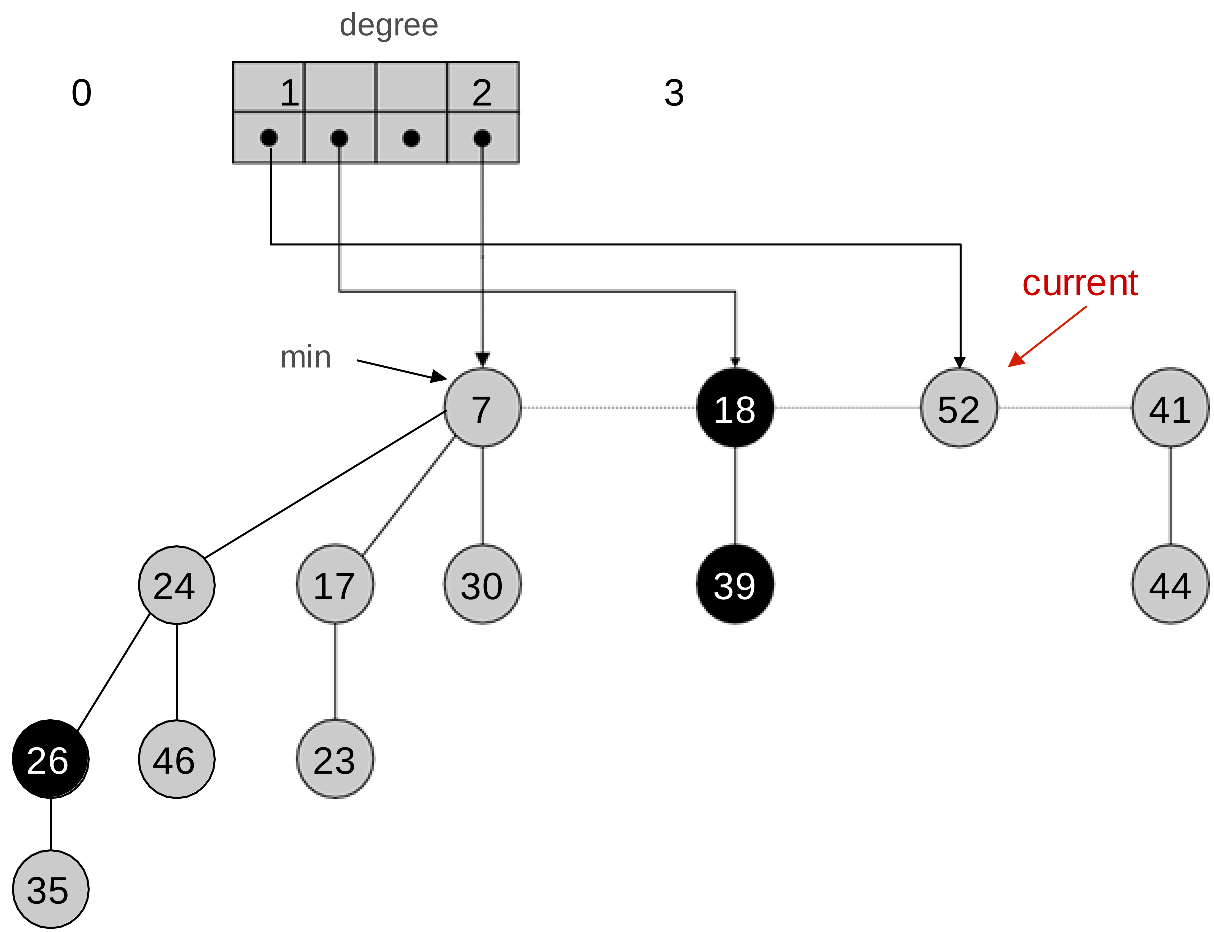
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

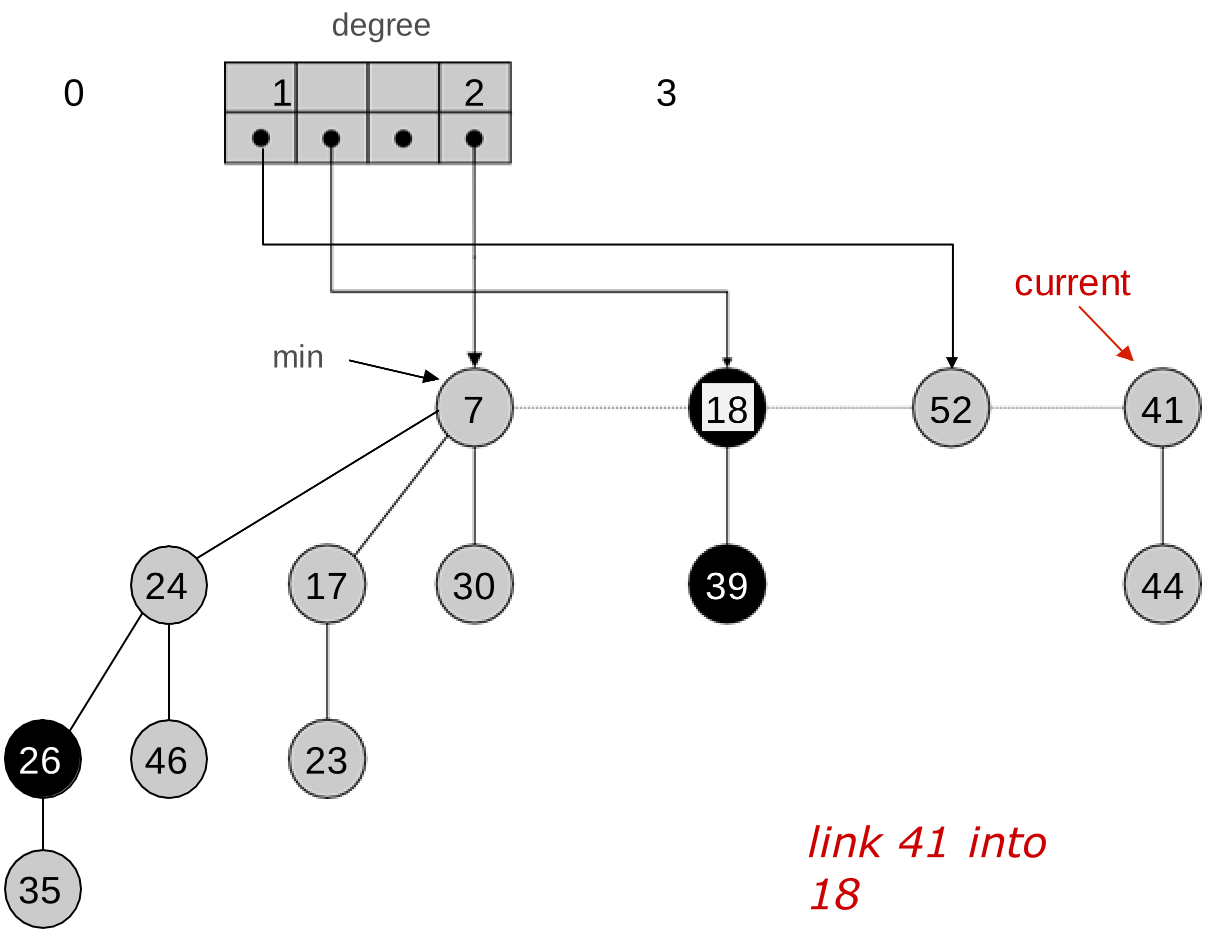
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



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.

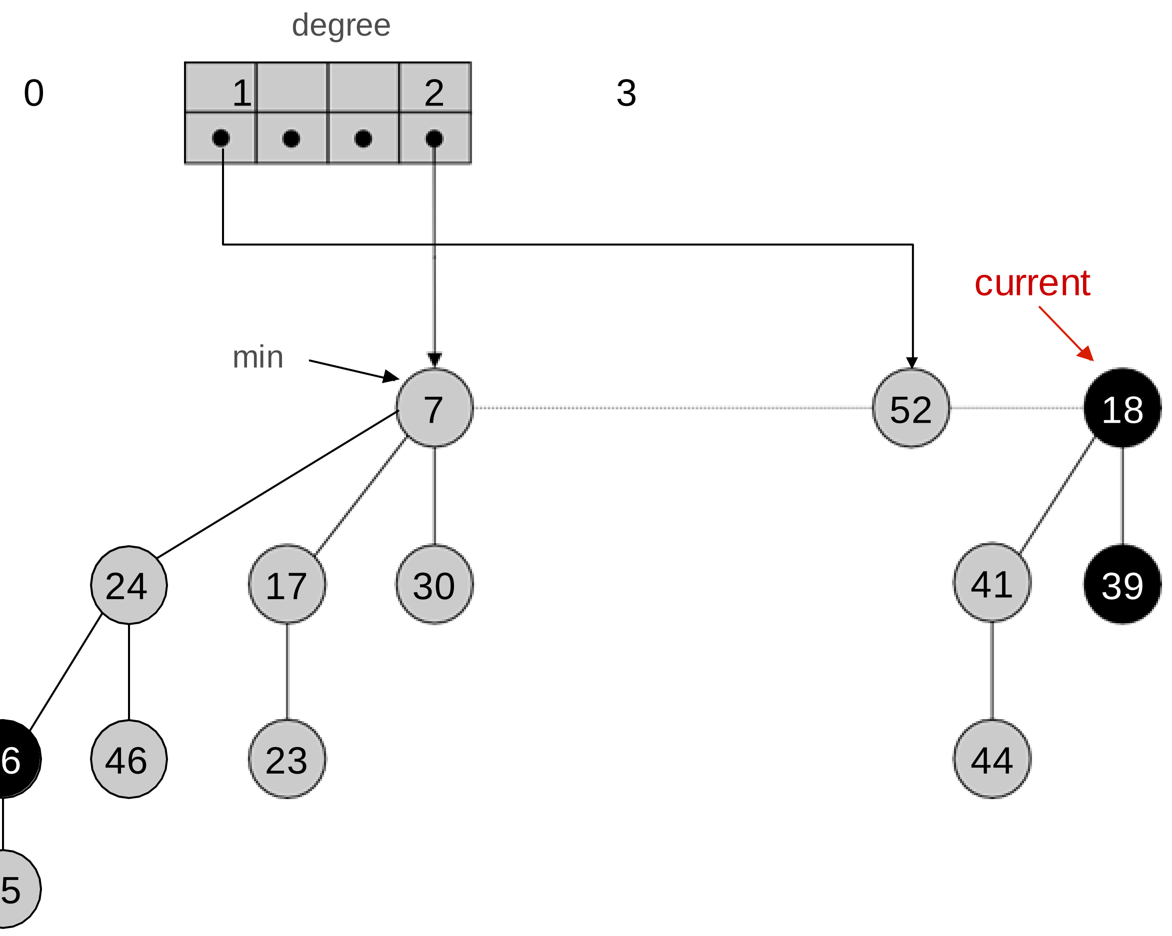


link 41 into 18

Fibonacci Heaps: Extract-Min

Extract-Min.

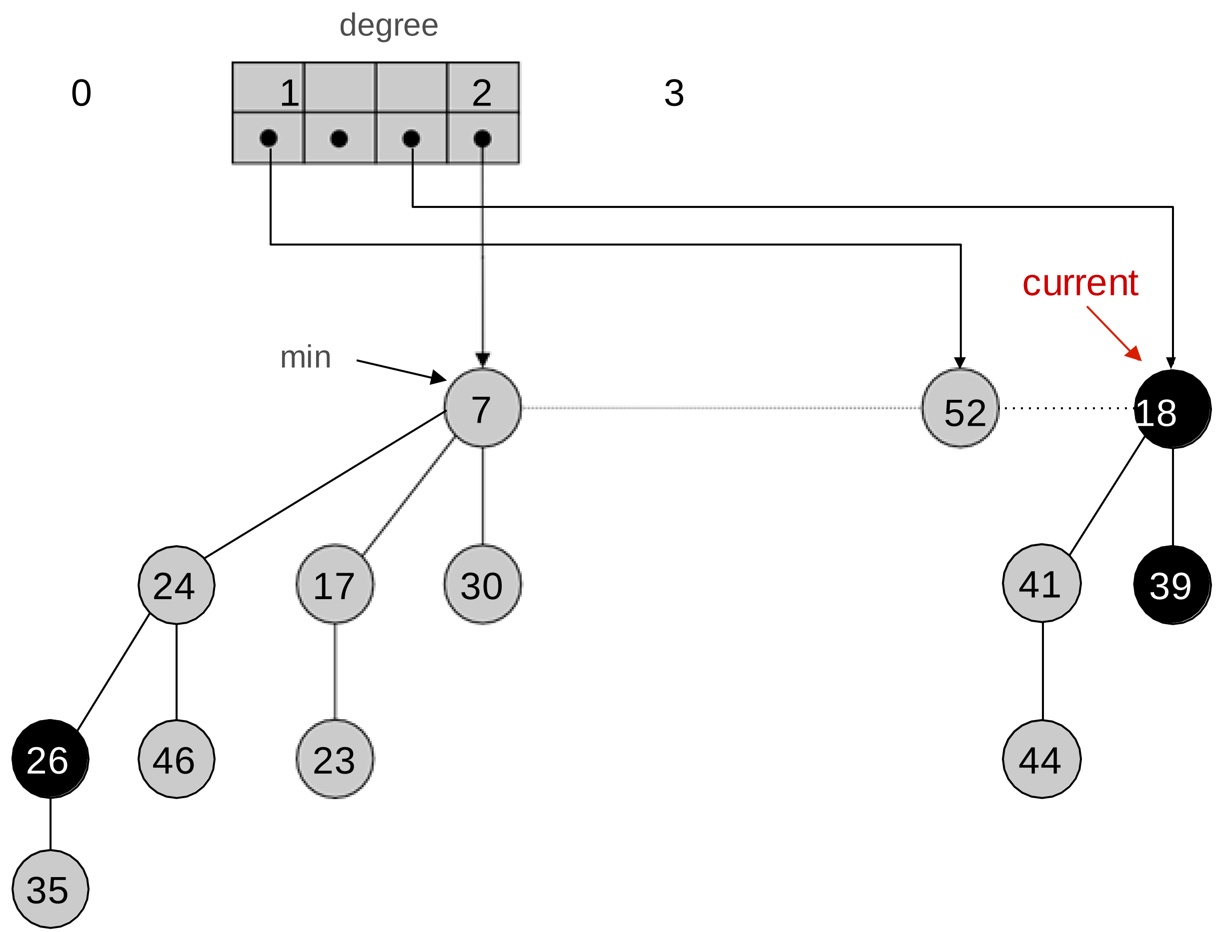
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract-Min

Extract-Min.

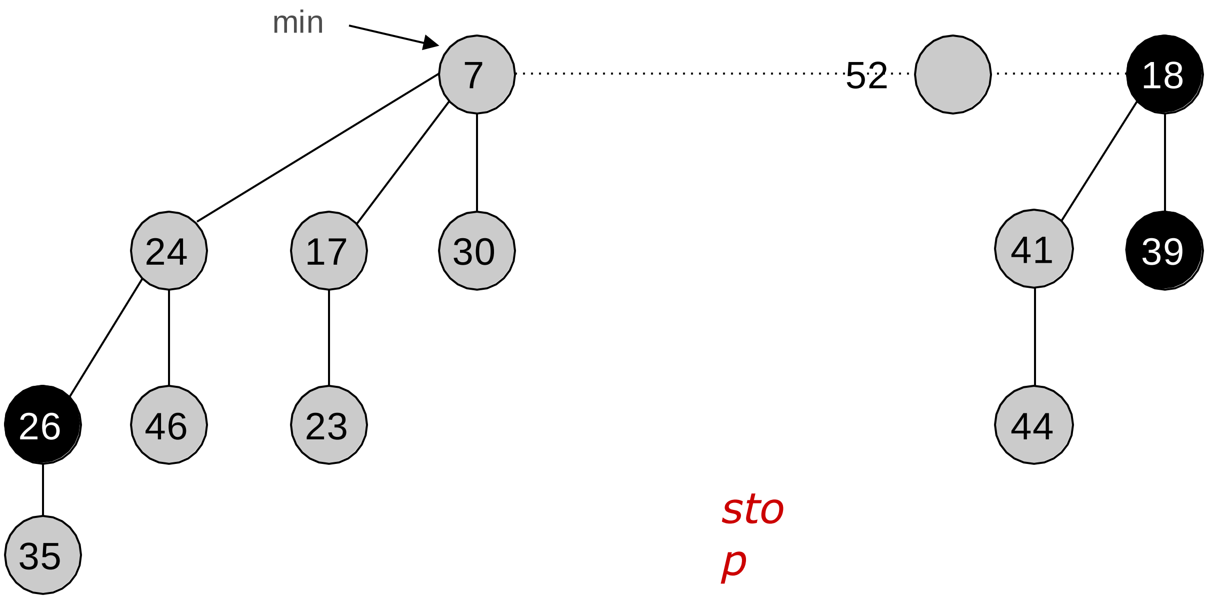
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same degree.



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 Analysis

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 rc $D(n)+1$ distinct degrees remain and no nodes become marked during the operation)

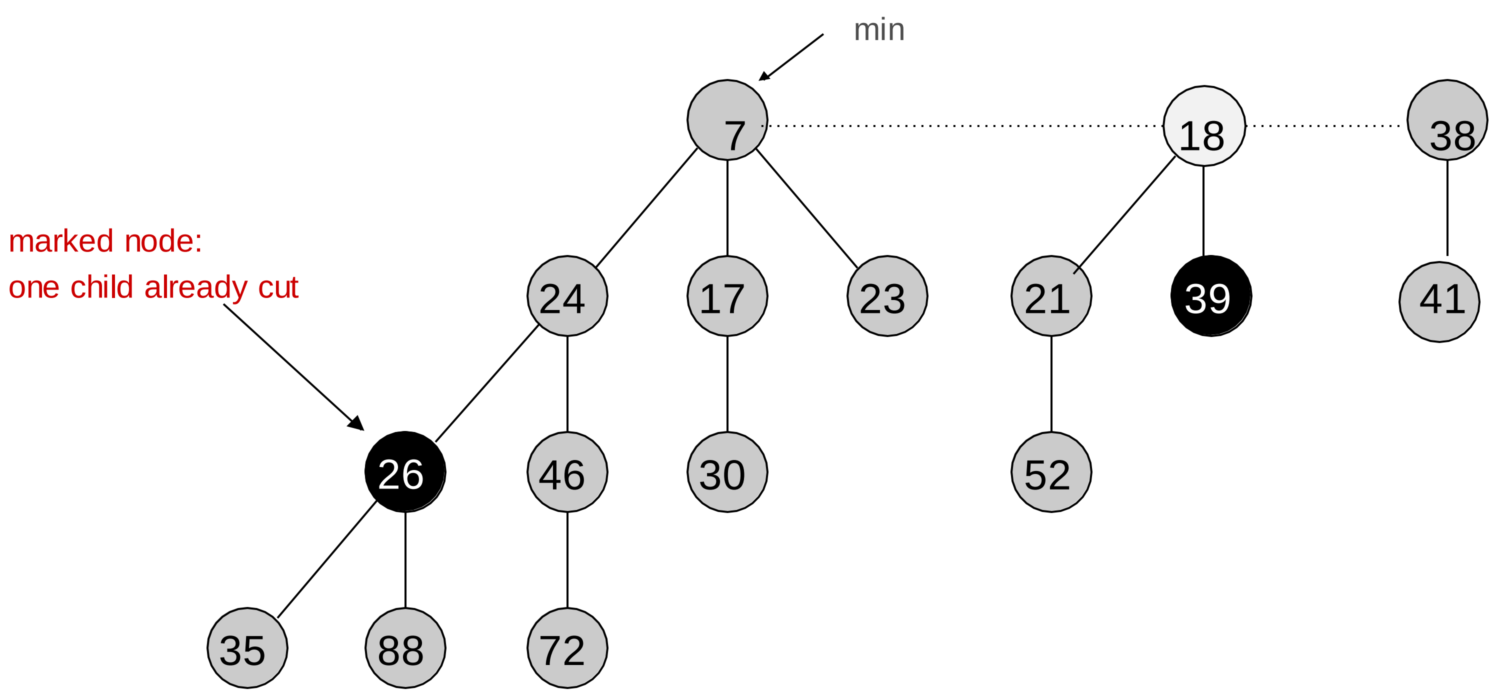
Amortized cost: $O((D(n))$

Decrease Key

Fibonacci Heaps: Decrease Key

Intuition for deceasing the key of node x.

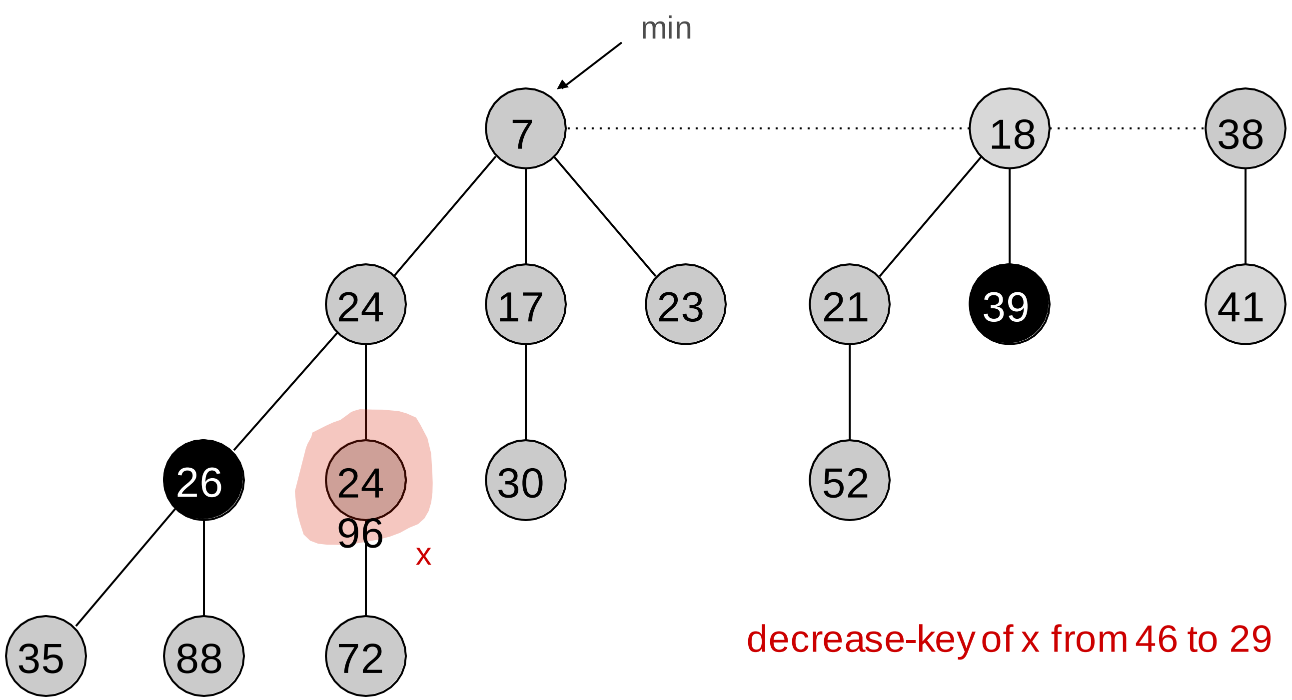
- If heap-order is not violated, just decrease the key of x.
- Otherwise, cut tree rooted at x and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

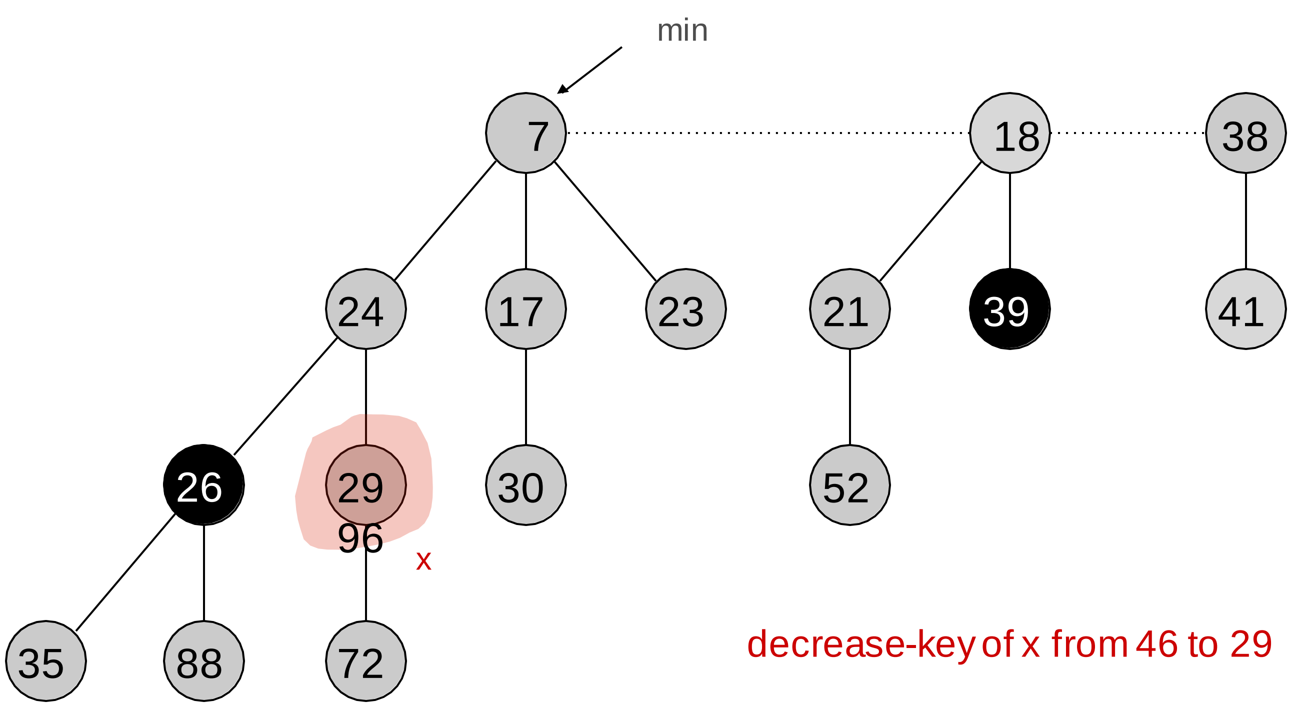
- Decrease key of x.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

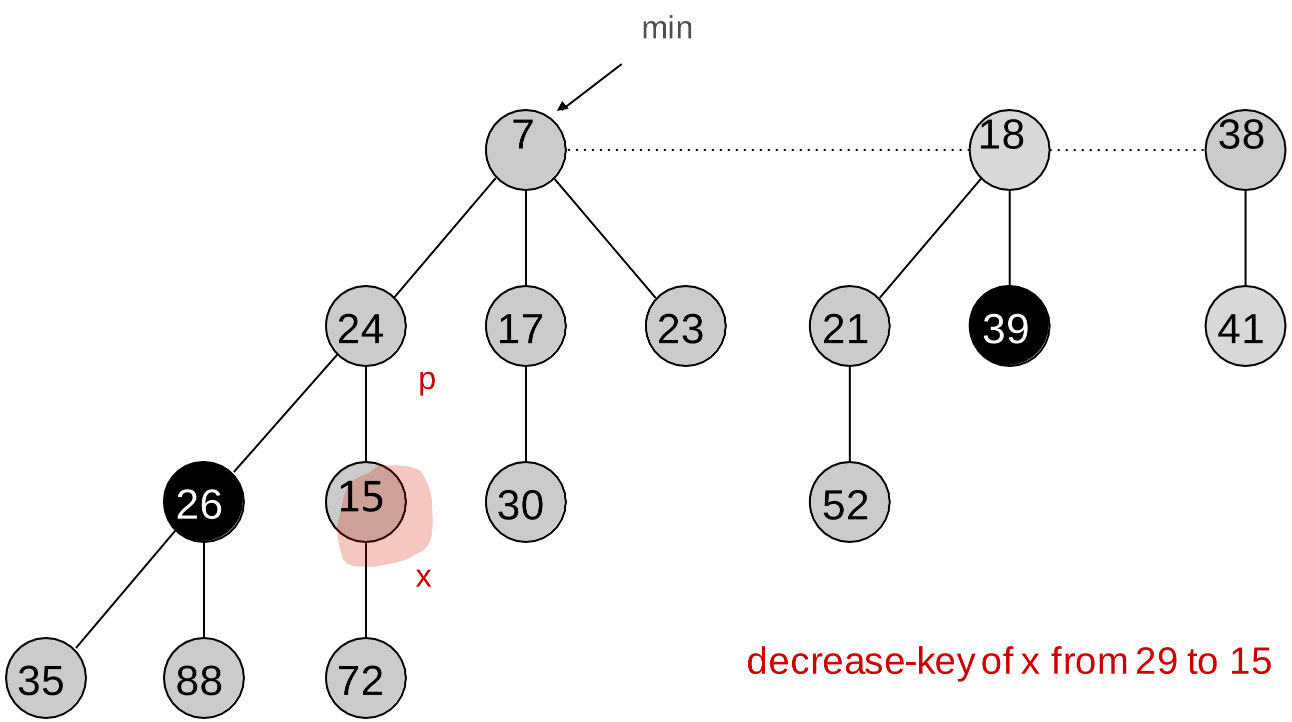
- Decrease key of x.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

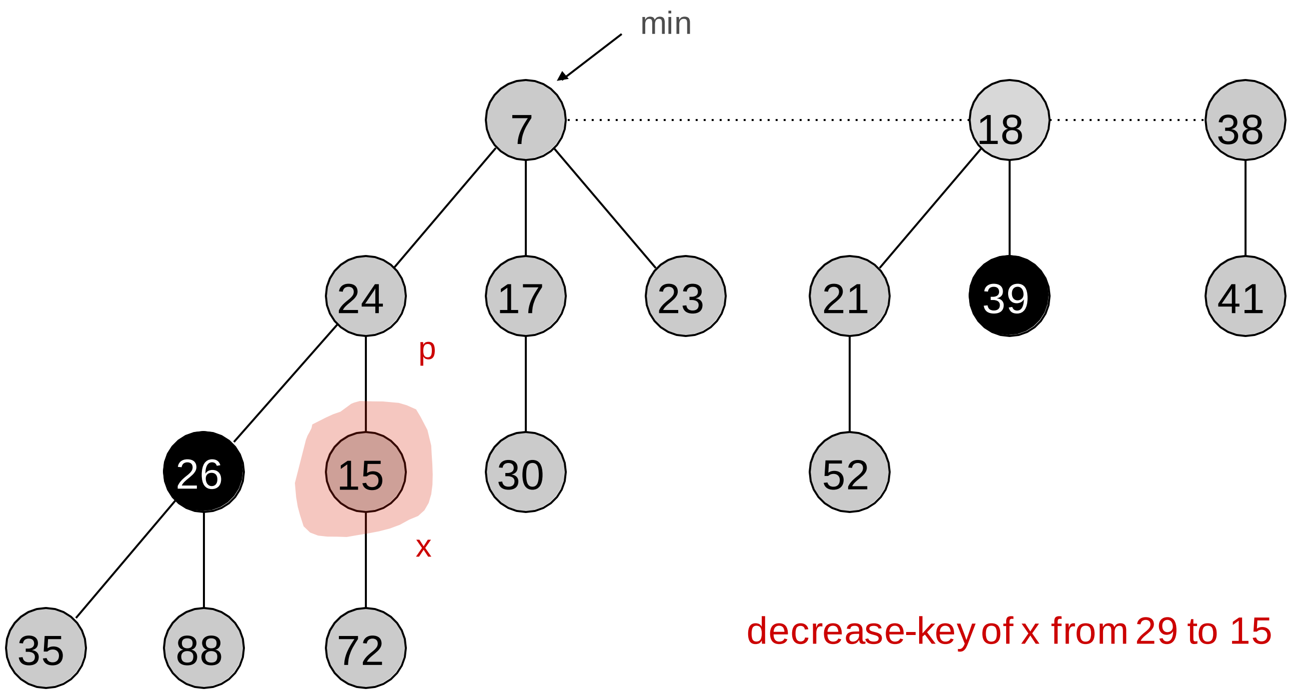
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



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).

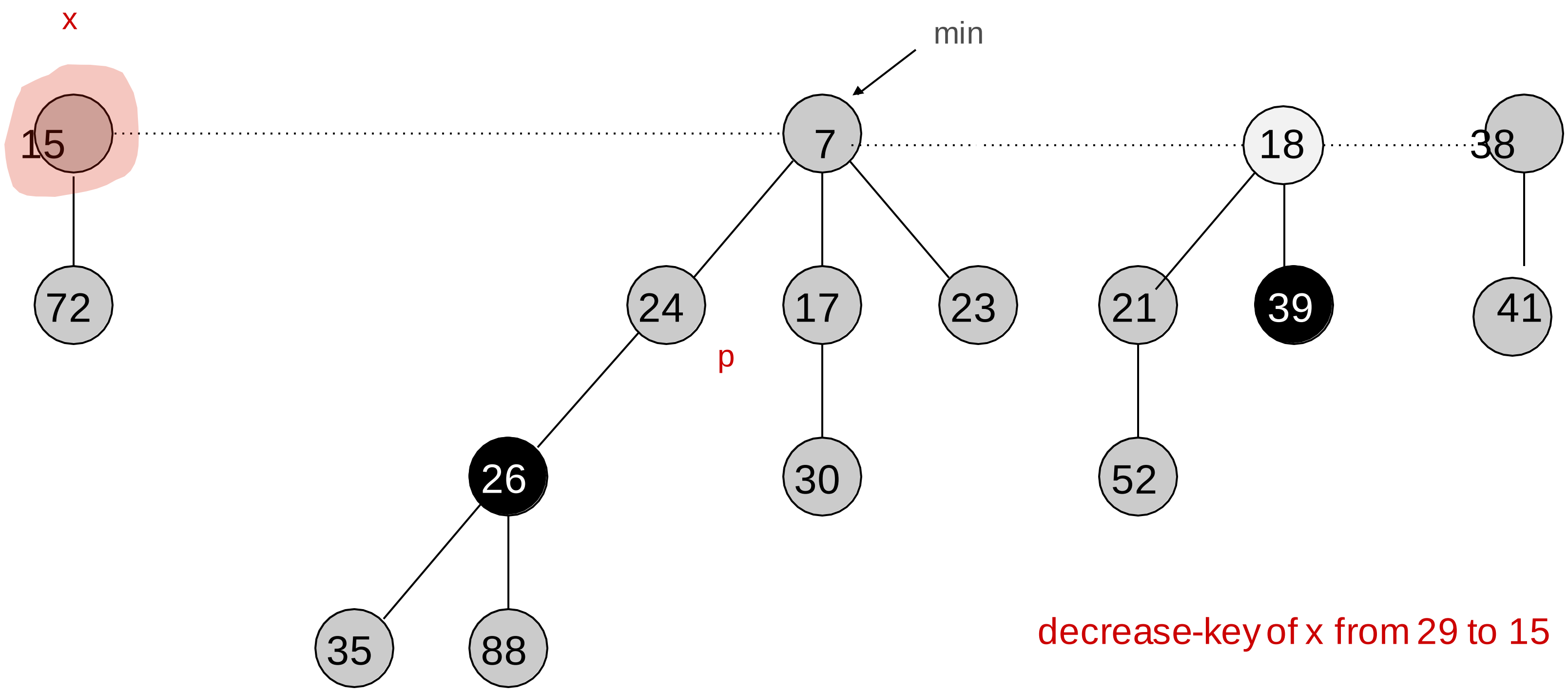


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

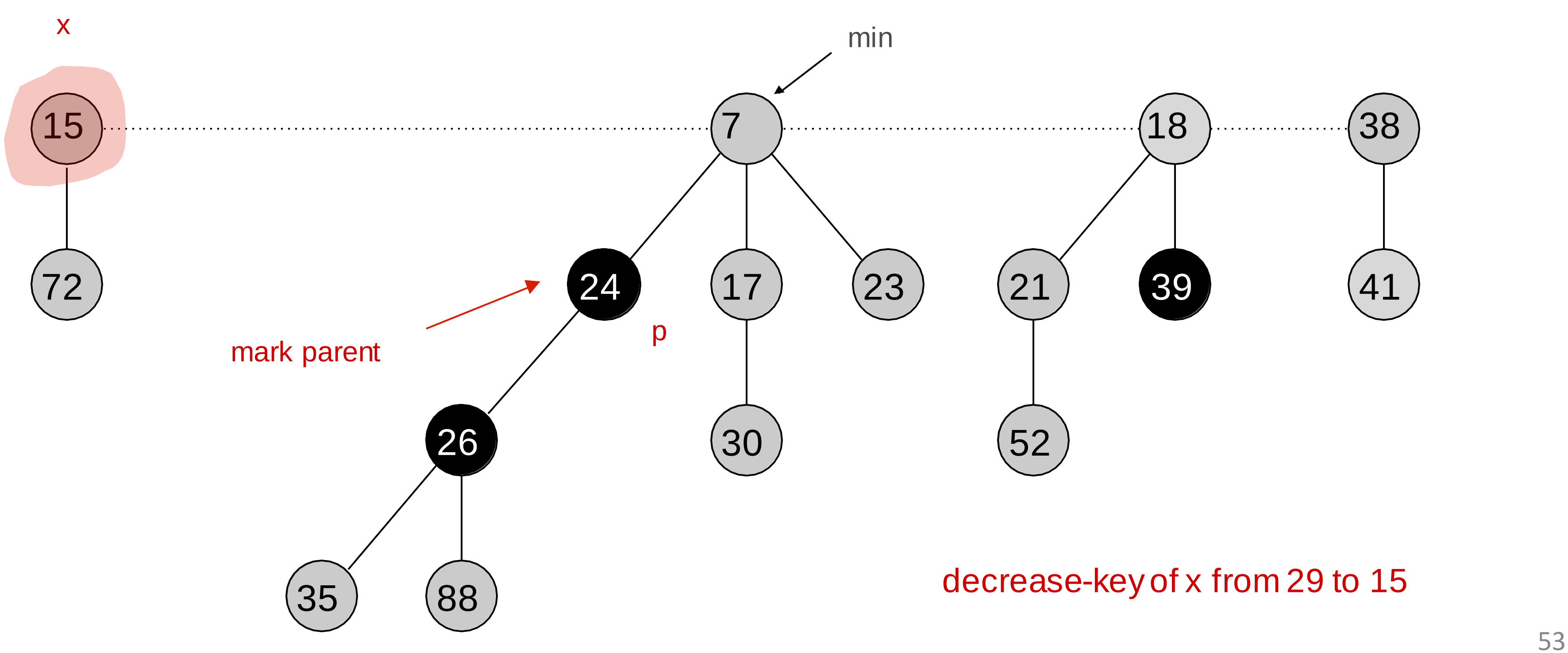
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

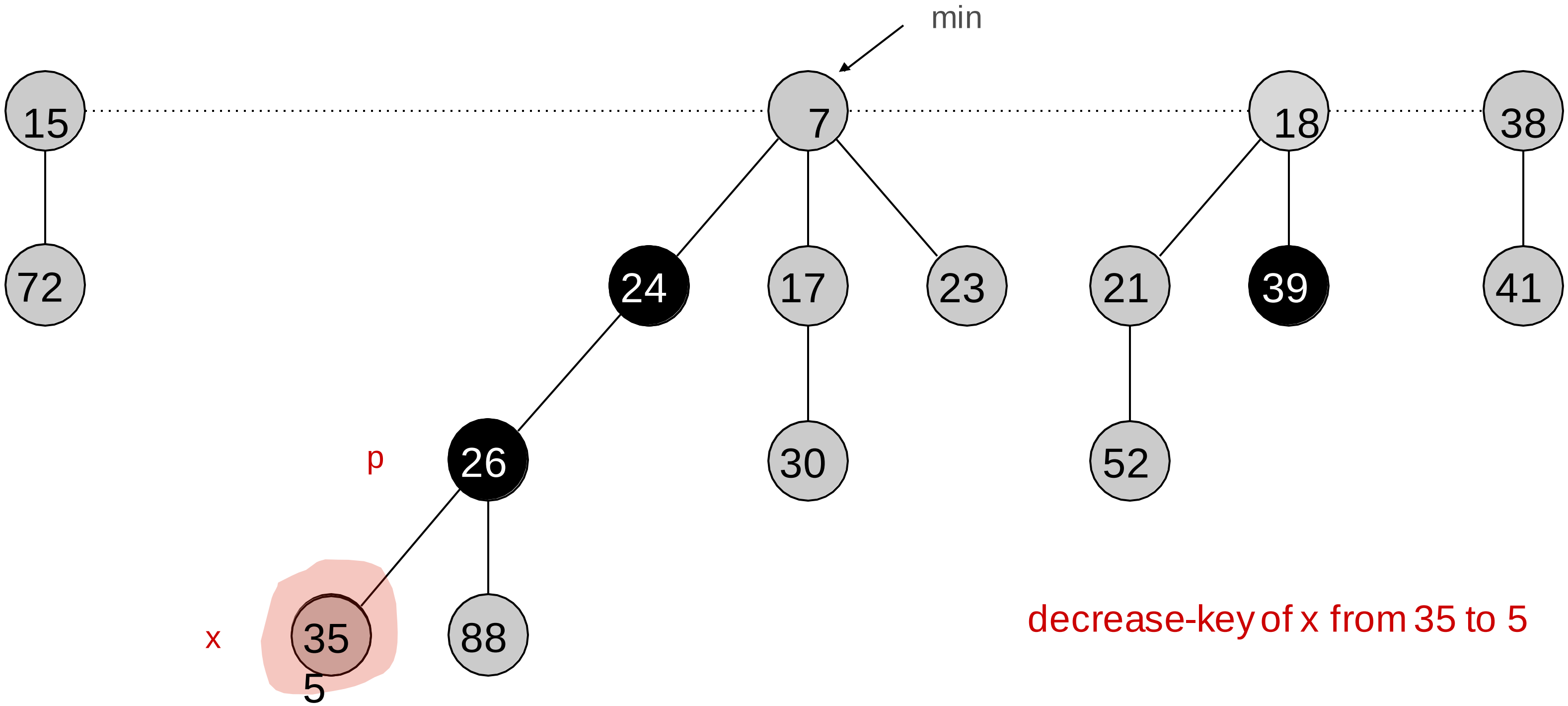
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

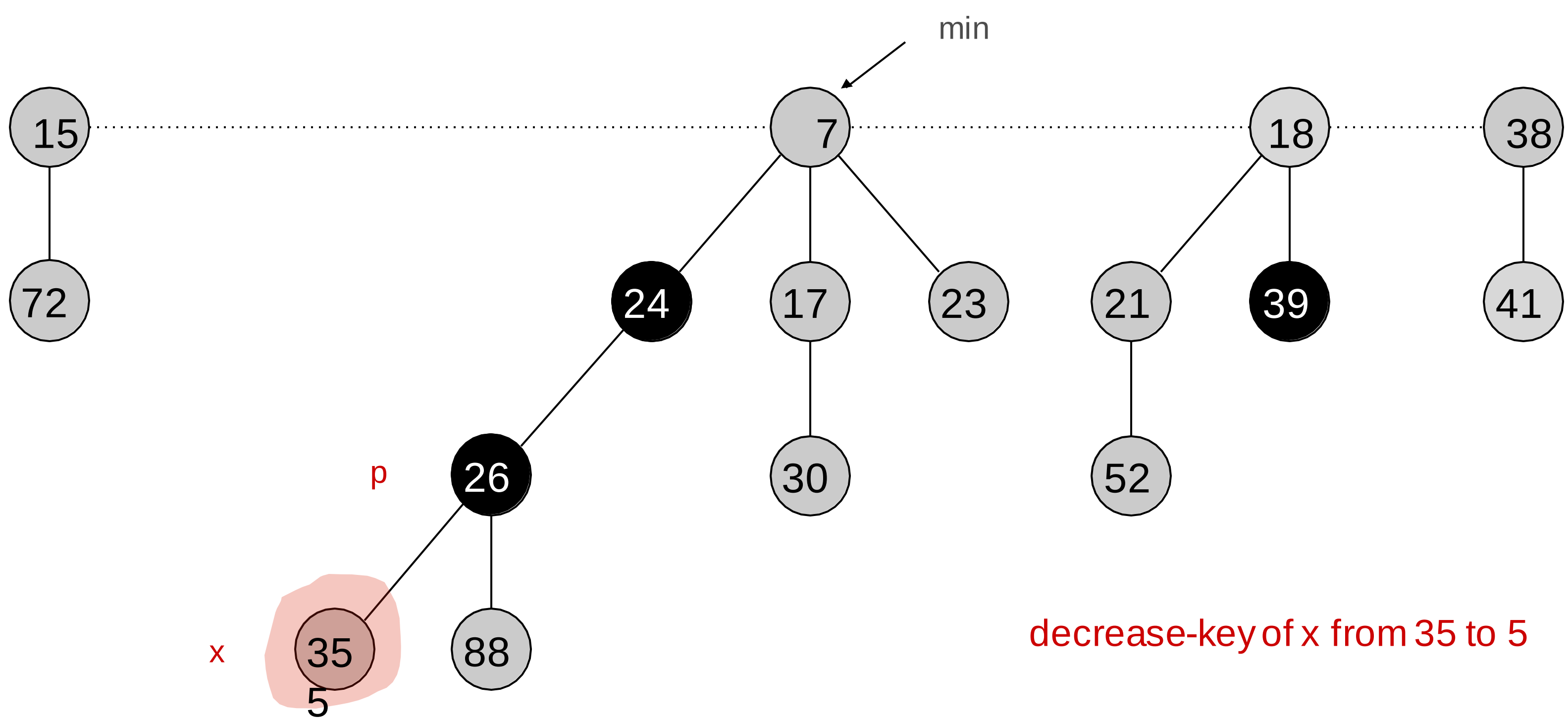
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

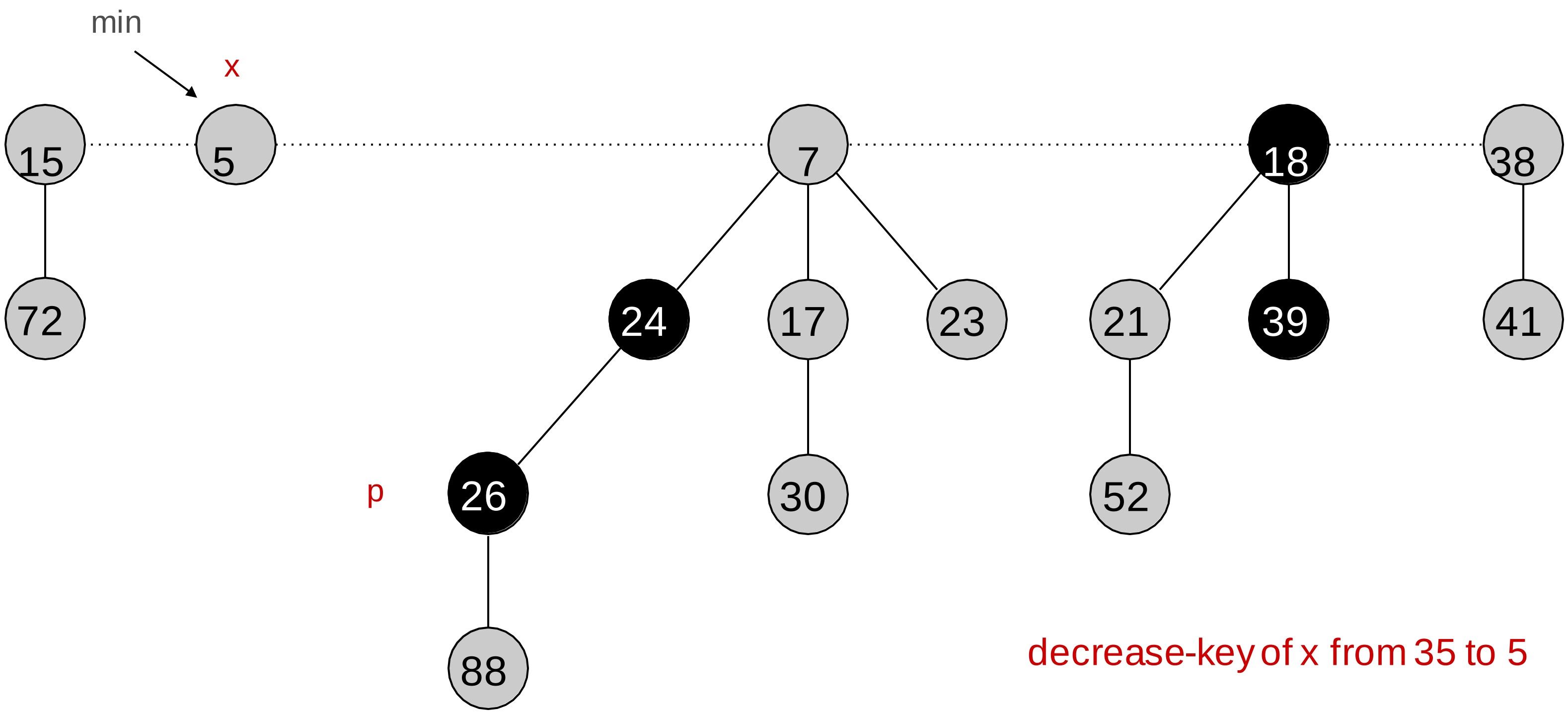
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



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).



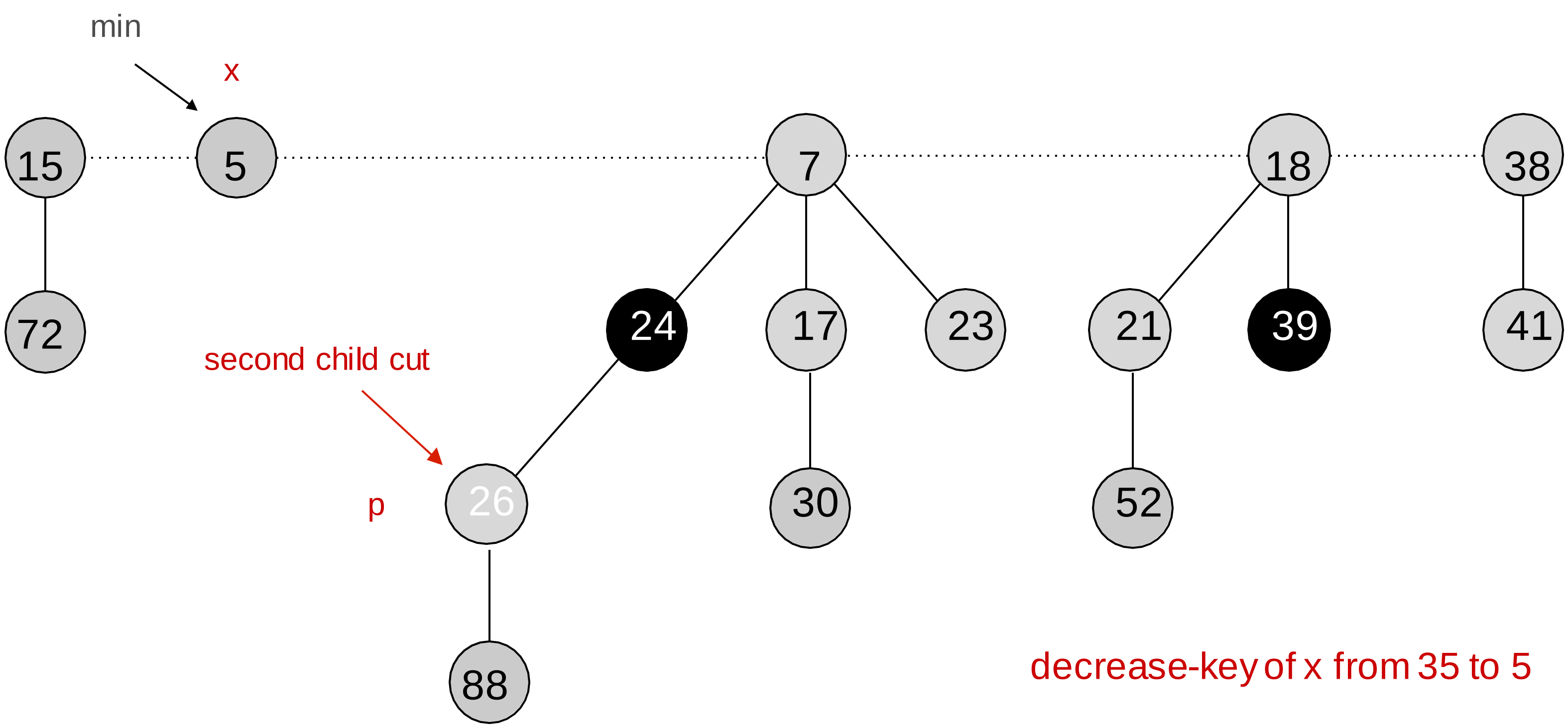
Fibonacci Heaps: Decrease Key

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).



decrease-key of x from 35 to 5

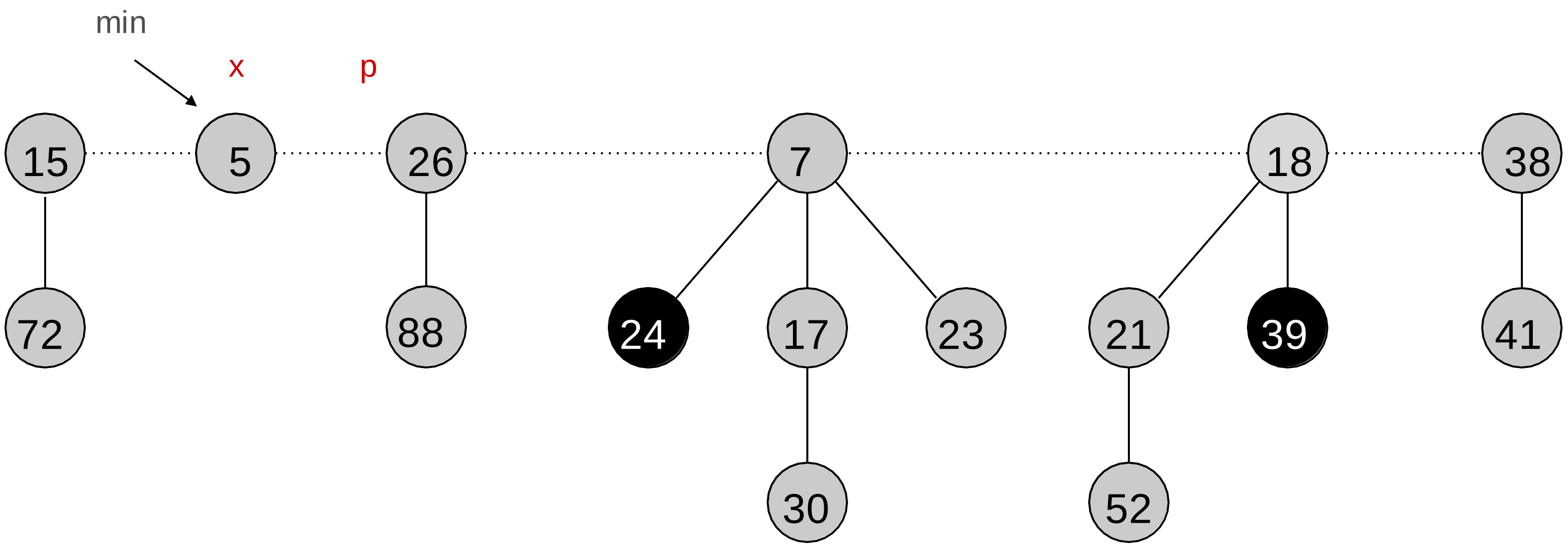
Fibonacci Heaps: Decrease Key

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).

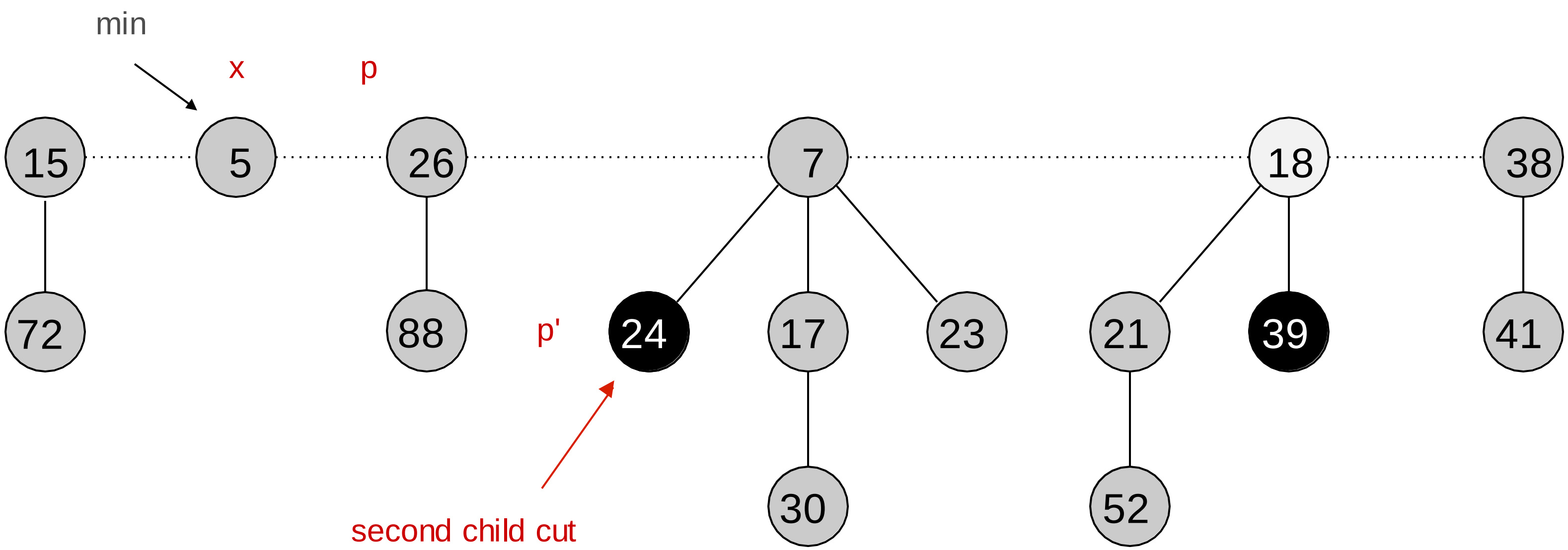


decrease-key of x from 35 to 5

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).

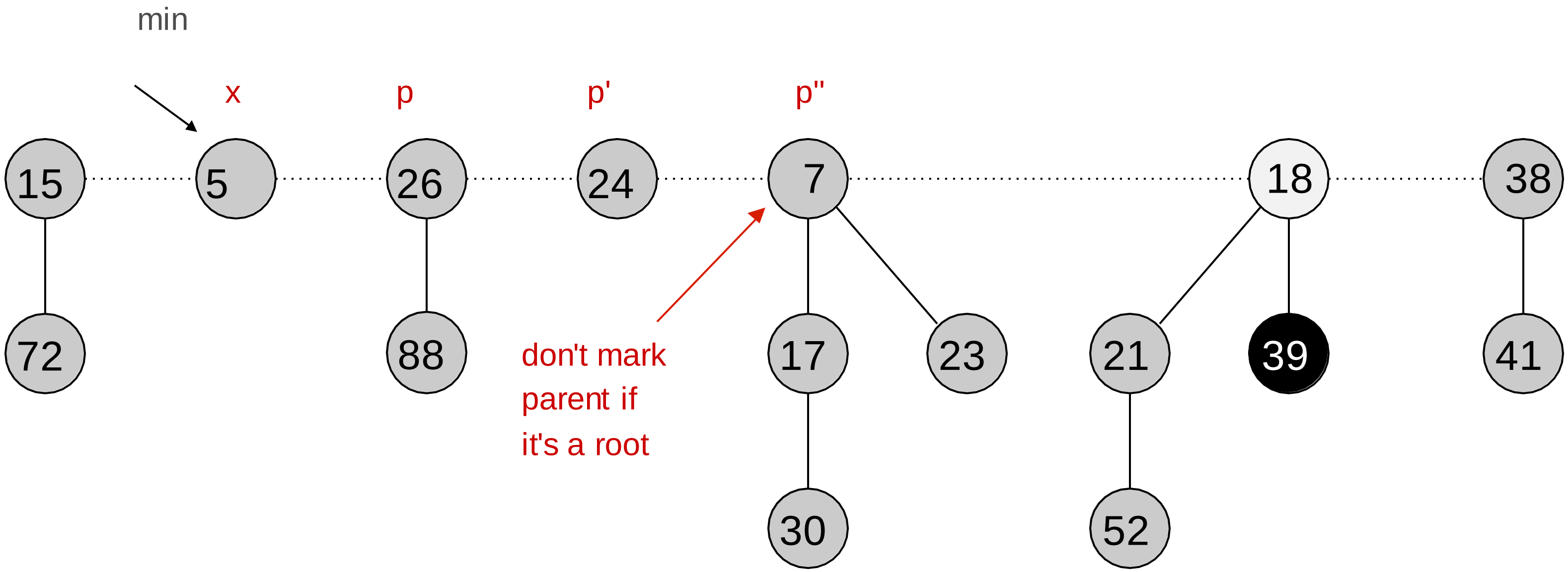


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

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

$O(1)$

Amortized cost.

Delete

- Fib-heap-decrease(H,x,-∞)
- Fib-heap-Extract-Min(H)