Heap Runtime

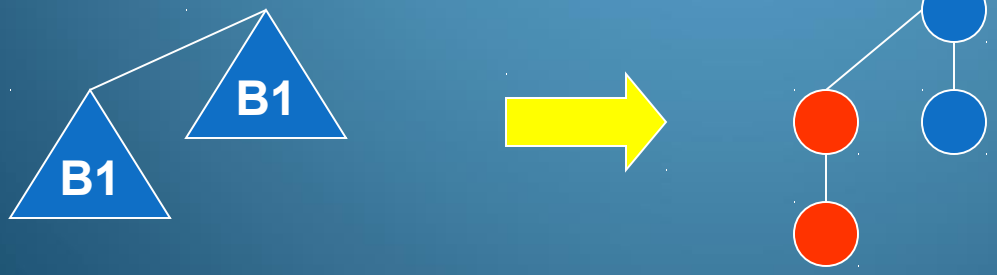
Operation	Binary Heap (worst-case)	Binomial Heap (worst-case)	Fibonacci Heap (amortized)
Insert	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
Minimum	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
Extract-min	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$
Union	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
Decrease-key	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
Delete	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$

Binomial Heaps

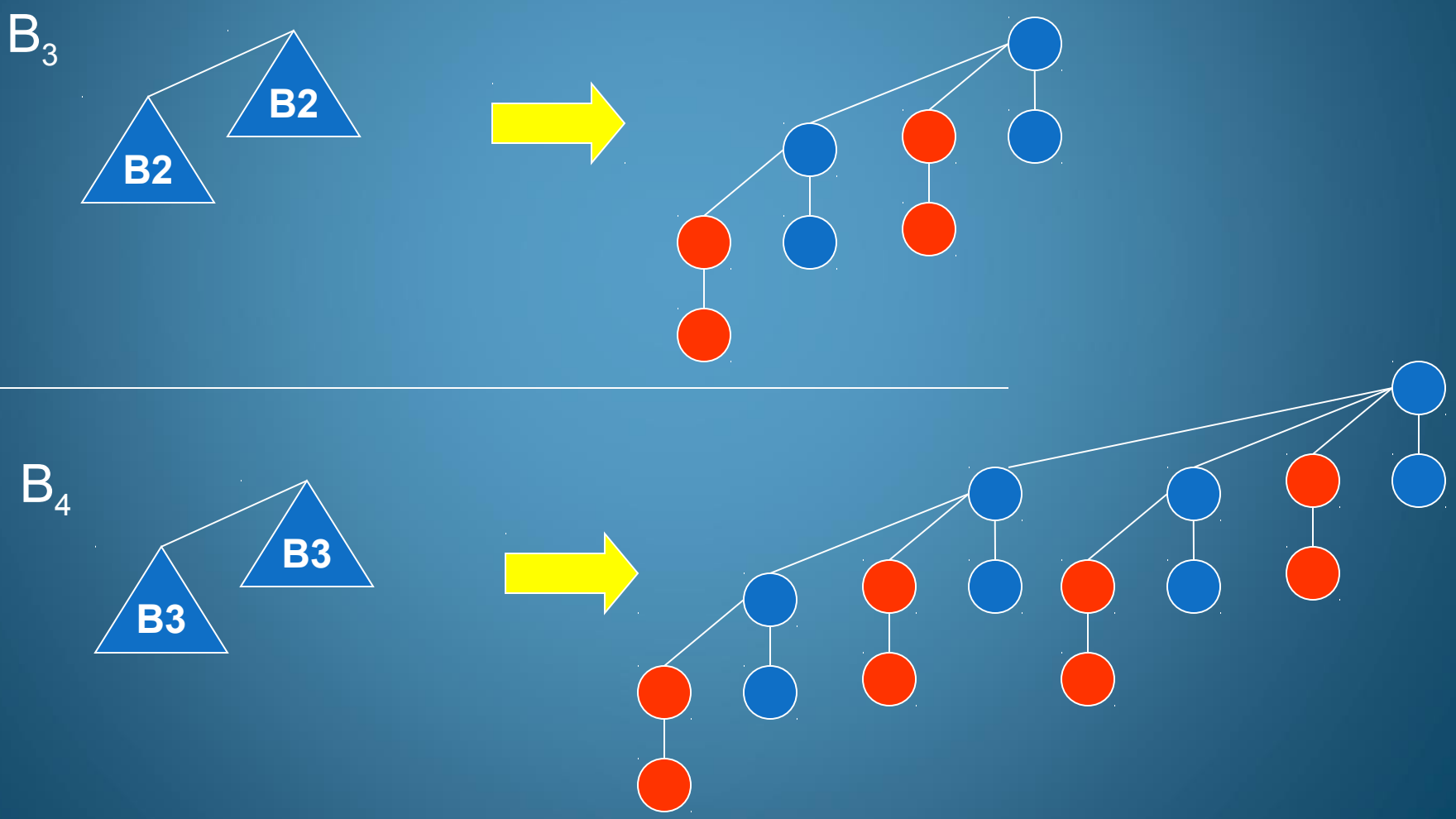
The binomial tree B_k is an ordered tree defined recursively.

B_0 

B_1 

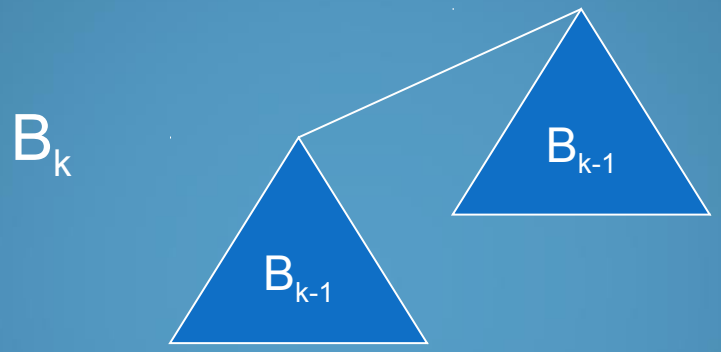
B_2 

Binomial Trees



Binomial Trees

In general:



Properties for tree B_k:

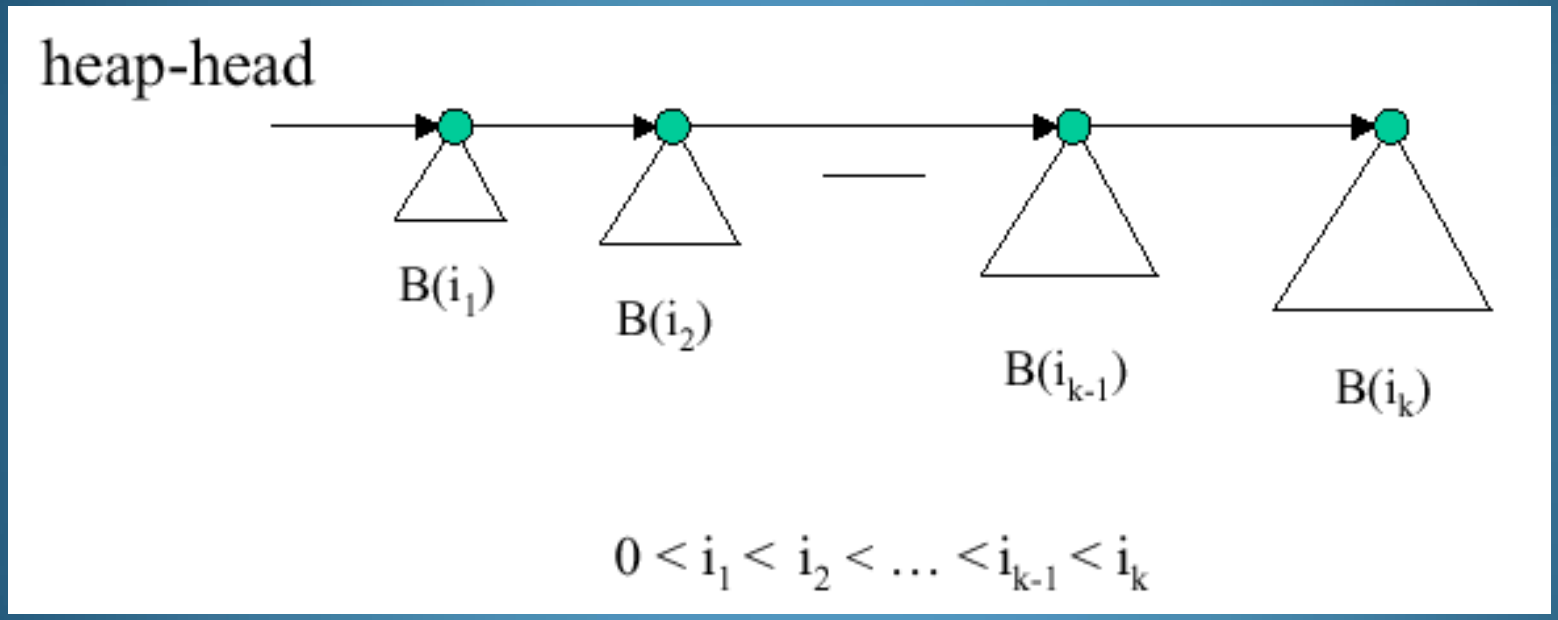
- There are 2^k nodes
- The height of the tree is k
- The number of nodes at depth i for i = 0…k is $\binom{k}{i} = \frac{k!}{i!(k-i)!}$
- The root has degree k which is greater than any other node

Binomial Heaps

A binomial heap H is a set of binomials trees that satisfies the following binomial-heap properties:

1. Each binomial tree in H obeys the min-heap property.
2. For any nonnegative integer k, there is at most one binomial tree in H whose root has degree k.
3. Binomial trees will be joined by a linked list of the roots

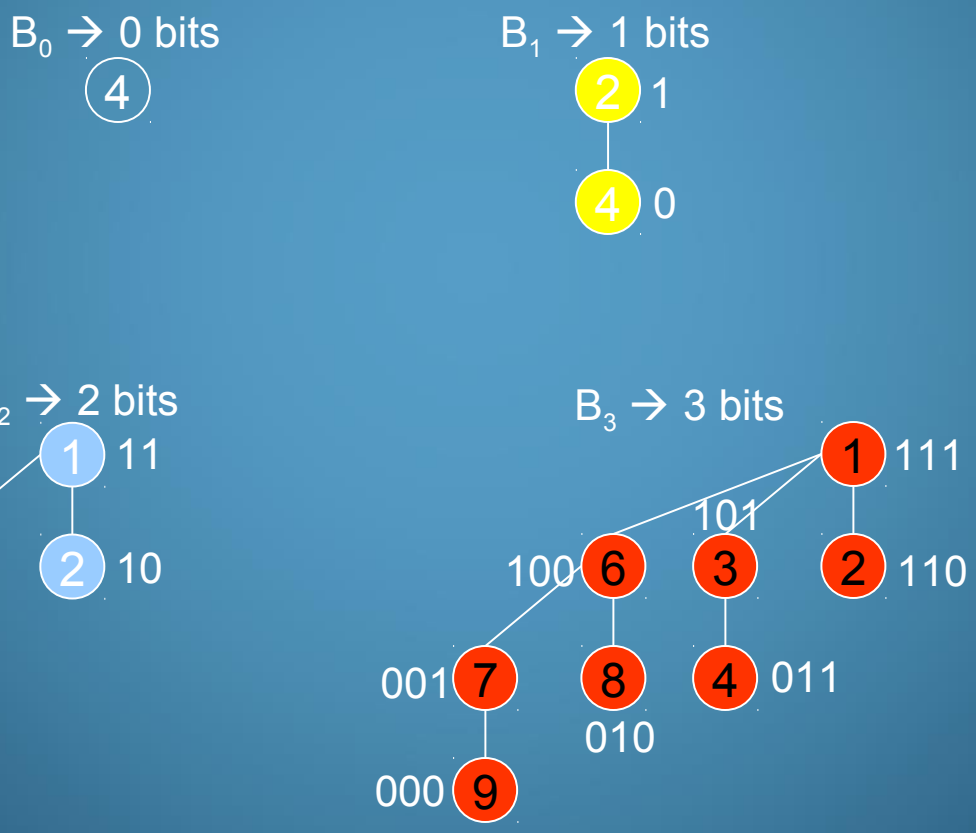
Binomial Heap Example



An n node binomial heap consists of at most Floor(lg n) + 1 binomial trees.

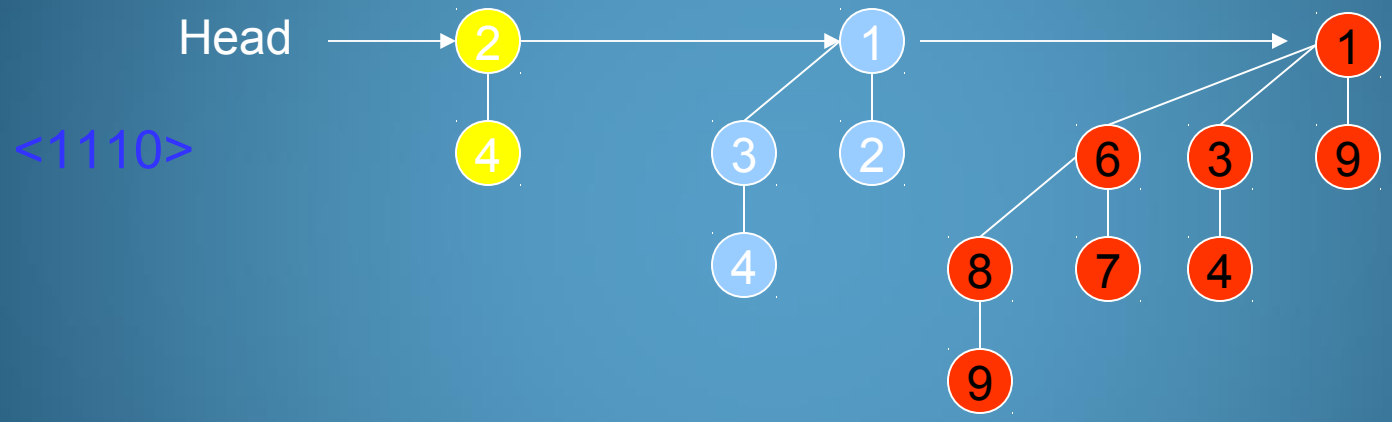
Binomial Heaps

How many binary bits are needed to count the nodes in any given Binomial Tree? Answer: k for B_k, where k is the degree of the root.



Binomial Heaps

Representing a Binomial Heap with 14 nodes

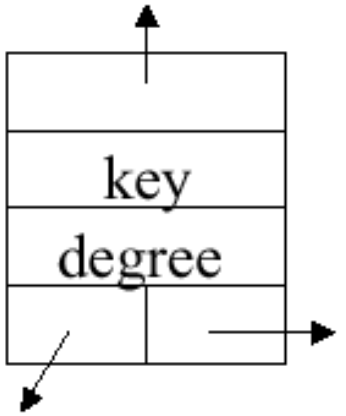


There are 14 nodes, which is 1110 in binary. This also can be written as <1110>, which means there is no B_0, one B_1, one B_2 and one B_3. There is a corresponding set of trees for a heap of any size!

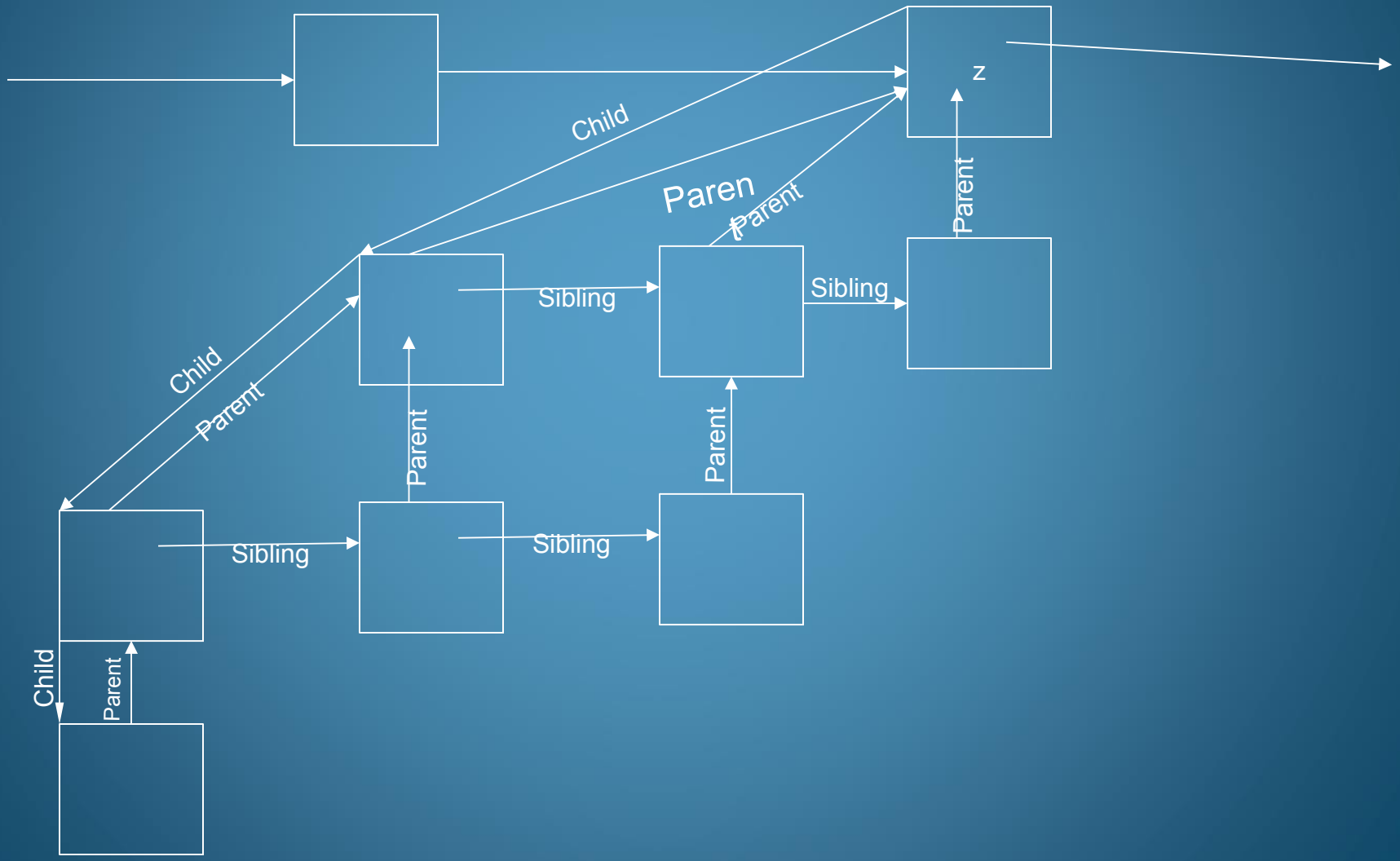
Node Representation

Each node consists of

- pointers to
  - its parent (if any),
  - its leftmost child (if any) and
  - the sibling directly to its right (if any)
- the degree of the node
- the key associated with the node



Binomial Heaps



Create New Binomial Heap

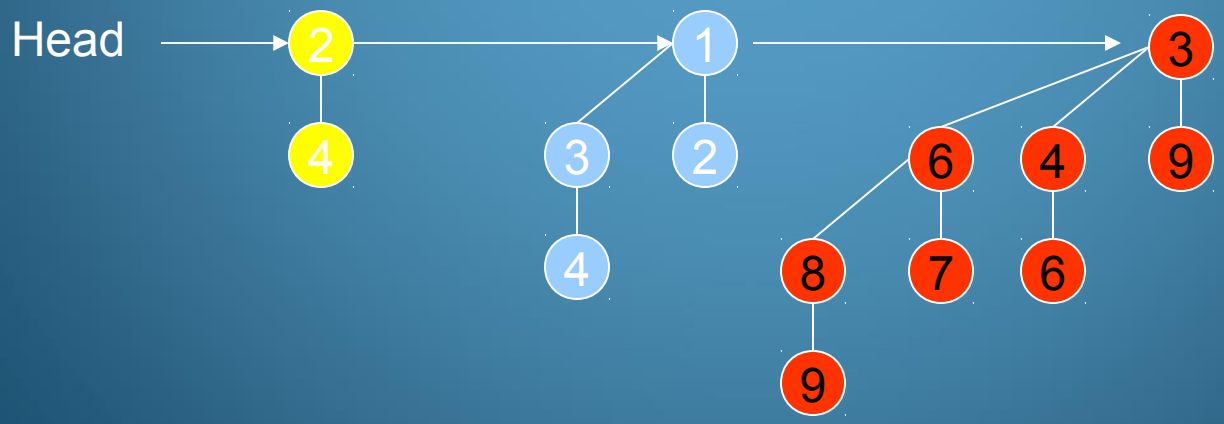
- Just allocate an object H, where

  head[H] = NIL

- $\Theta(1)$ runtime

Binomial Min-Heap

- Walk across roots, find minimum
- O(lg n) since at most lg n + 1 trees

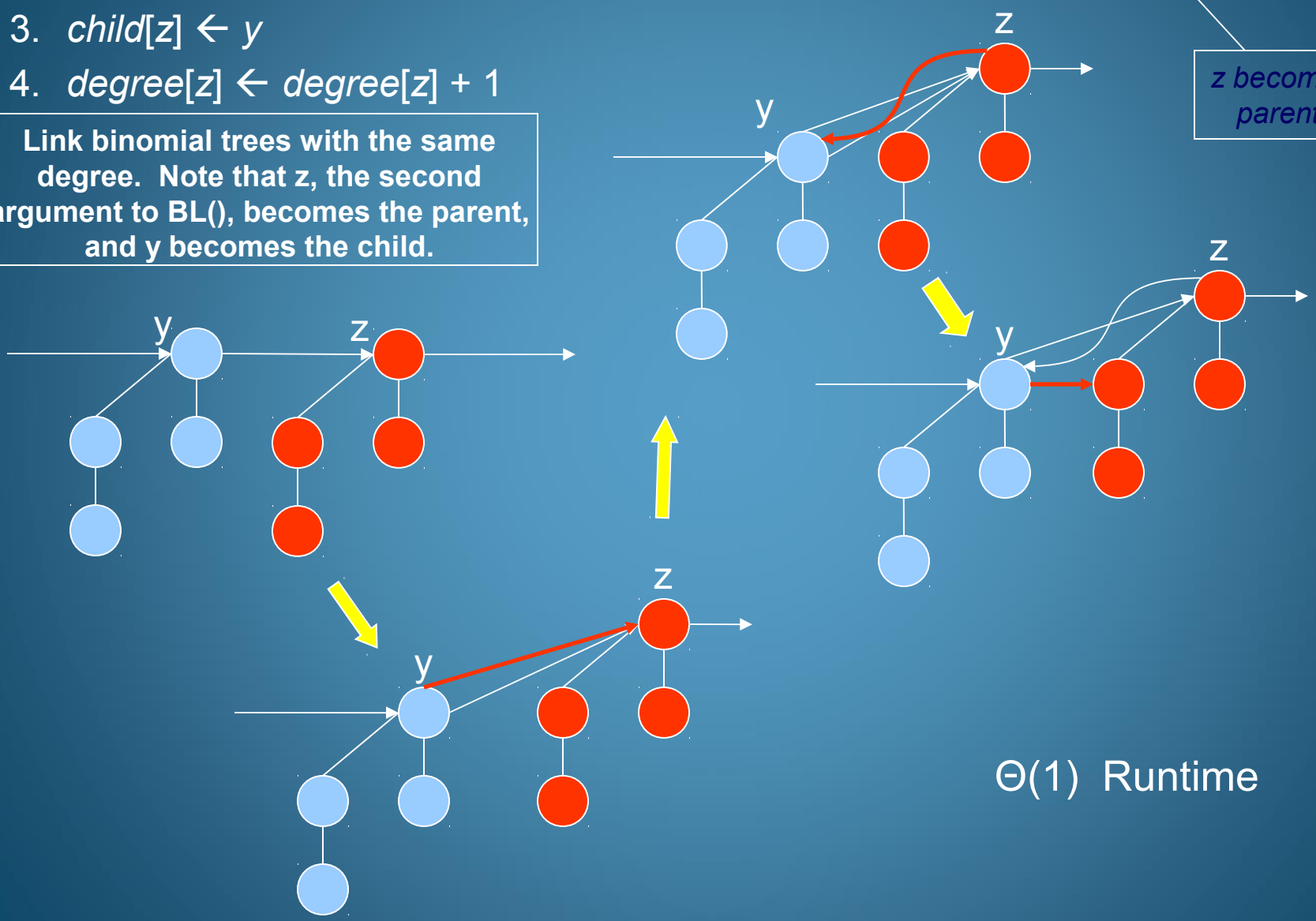


Binomial-Link(y,z)

1. $p[y] \leftarrow z$
2. $sibling[y] \leftarrow child[z]$
3. $child[z] \leftarrow y$
4. $degree[z] \leftarrow degree[z] + 1$

Link binomial trees with the same degree. Note that z, the second argument to BL(), becomes the parent, and y becomes the child.

z becomes the parent of y



Θ(1) Runtime

Binomial-Heap-Union(H_1, H_2)

1. H ← Binomial-Heap-Merge(H_1, H_2)

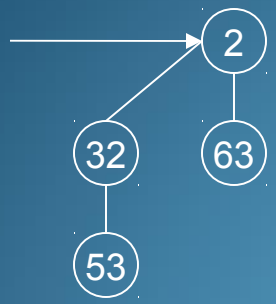
   This merges the root lists of H_1 and H_2 in increasing order of root degree

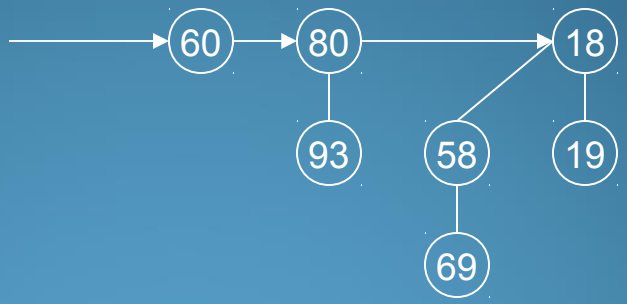
1. Walk across the merged root list, merging binomial trees of equal degree. If there are three such trees in a row only merge the last two together (to maintain property of increasing order of root degree as we walk the roots)

   Concept illustrated on next slide; skips some implementation details of cases to track which pointers to change

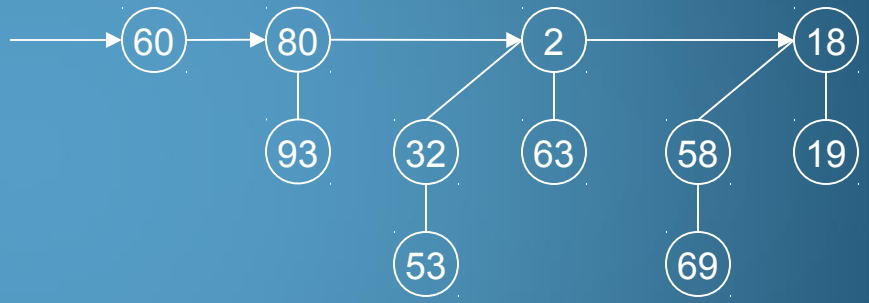
   Runtime: Merge time plus Walk Time: O(lg n)

Starting with the following two binomial heaps:

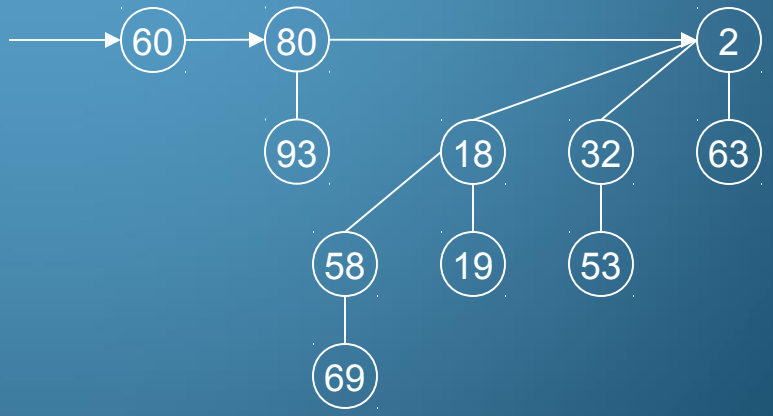




Merge root lists, but now we have two trees of same degree



Combine trees of same degree using binomial link, make smaller key the root of the combined tree



Binomial-Heap-Insert(H)

To insert a new node, simple create a new Binomial Heap with one node (the one to insert) and then Union it with the heap

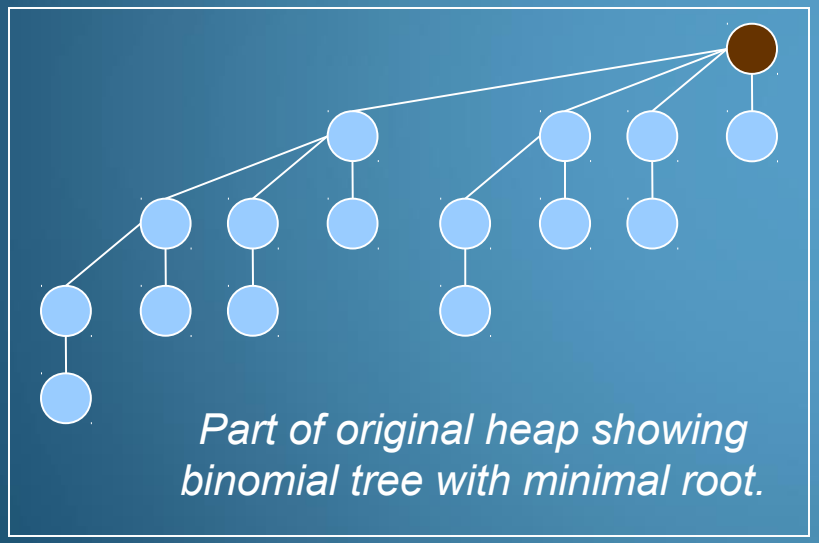
1. H' ← Make-Binomial-Heap()
2. p[x] ← NIL
3. child[x] ← NIL
4. sibling[x] ← NIL
5. degree[x] ← 0
6. head[H'] ← x
7. H ← Binomial-Heap-Union(H, H')

Runtime: O(lg n)

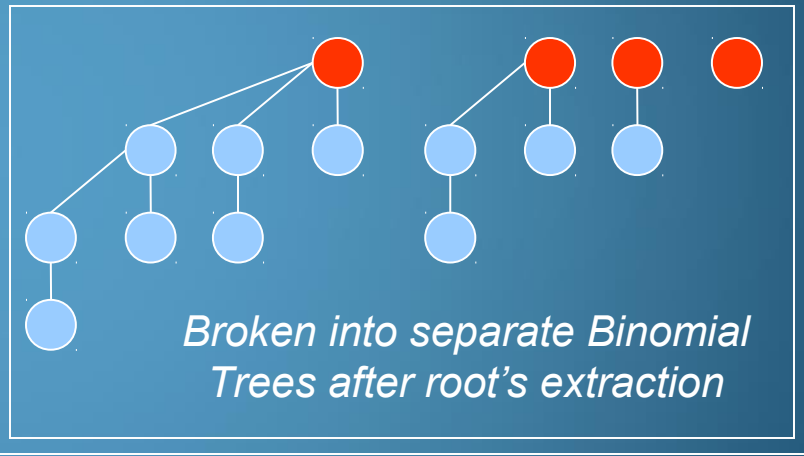
Binomial-Heap-Extract-Min(H)

With a min-heap, the root has the least value in the heap.

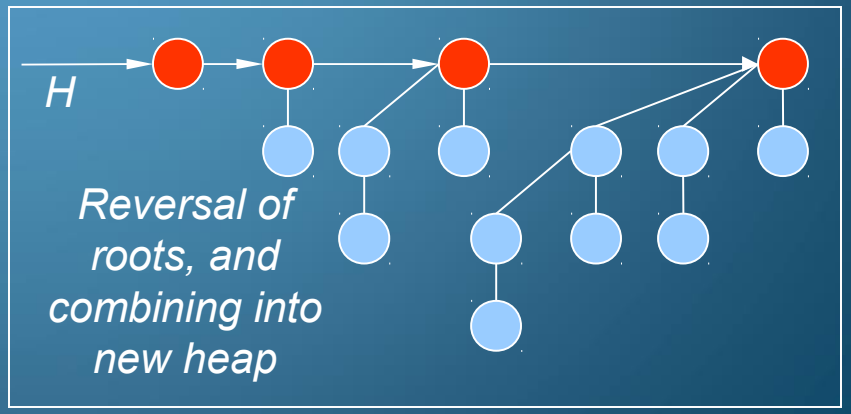
Notice that if we remove the root from the figure below, we are left with four heaps, and they are in decreasing order of degree. So to extract the min we create a root list of the children of the node being extracted, but do so in reverse order. Then call Binomial-Heap-Union(..)



Part of original heap showing binomial tree with minimal root.



Broken into separate Binomial Trees after root's extraction



Reversal of roots, and combining into new heap

Runtime: $\Theta(\lg n)$

Heap Decrease Key

- Same as decrease-key for a binary heap
  - Move the element upward, swapping values, until we reach a position where the value is ≤ key of its parent

Runtime: $\Theta(\lg n)$

Heap Delete Key

- Set key of node to delete to –infinity and extract it:

```
Decrease-key(H, p, -∞)
Extract-min(H)
```

Runtime: $\Theta(\lg n)$