ID1020: Priority queues

ch 2.4

Priority Queues

- Many applications builds on that we can insert elements with priorities into an ADT and retrieve the element with the highest priority
- Eg:
  - Process/thread scheduling in an OS
  - Ordering of events by timestamp in simulations
- High priority:
  - Often low numerical value: Priority of processes, timestamps (next event)...

Priority queues: basic operations

- Enqueue – insert a new element
- Dequeue – remove and return the element with the highest priority

- Stability: Often we want to preserve (causal) order between elements with equal priority
  - Preserve FIFO-order among elements with equal priority

Priority queues: Lazyness

- Priority queues need only be partially sorted
  - Retrieve the highest priority element

Simple priority queue implementations

- Ordered list
  - Keep elements ordered in descending priority order
  - Enqueue – sort the new element into correct position
  - Dequeue – remove the first (highest) priority element

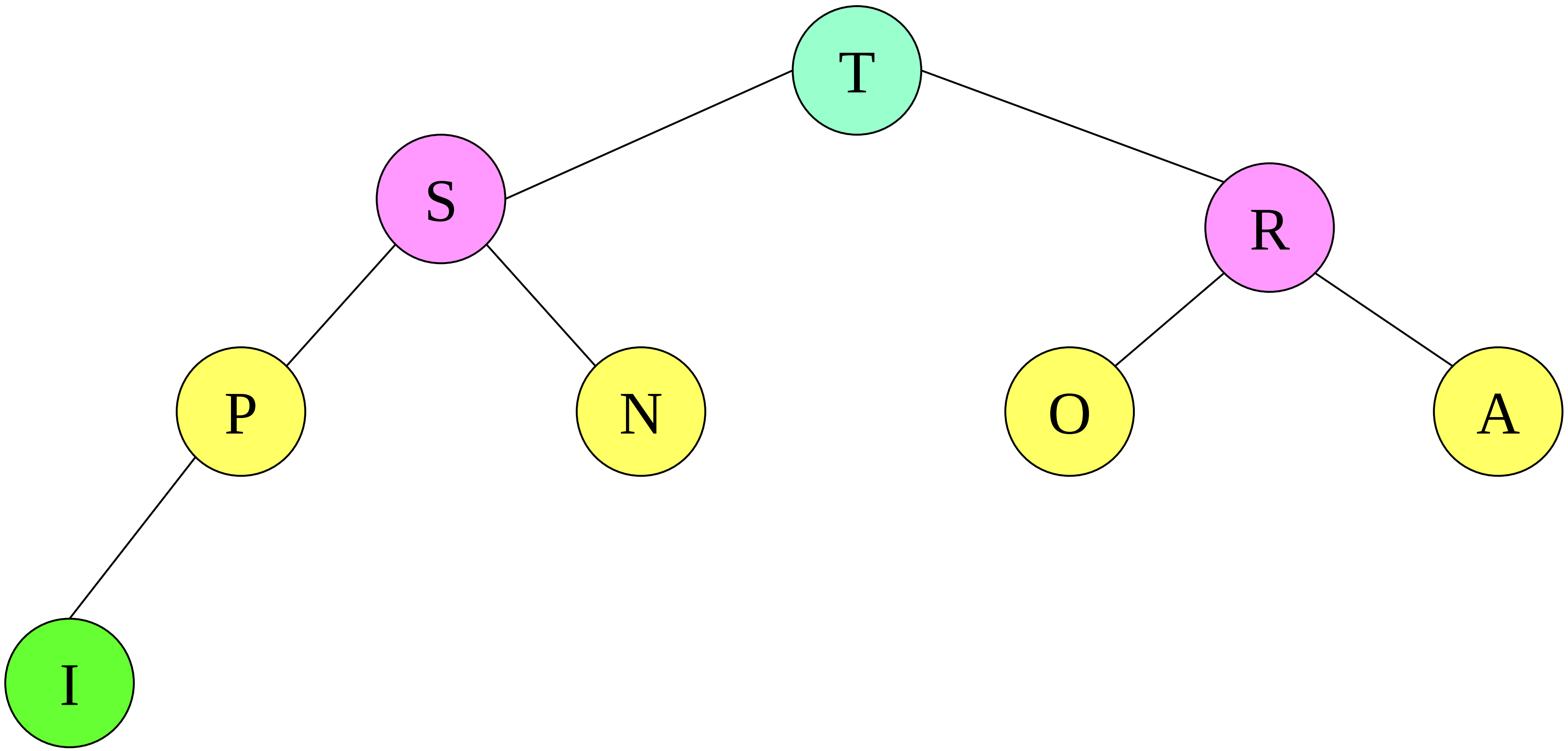
- Time complexity
  - Enqueue: O(N)
  - Dequeue: O(1)

- Linked list is often more efficient than ordered array as less movement of data at enqueue

Binary heap

- A balanced binary tree
- The heap property
  - All nodes have higher or equal priority than any of its two children
  - The highest priority element is found in the root
- Time complexity: Enqueue and dequeue O(Log(N))
- Each level in the tree is populated from left to right when elements are enqueued

Binary heap example



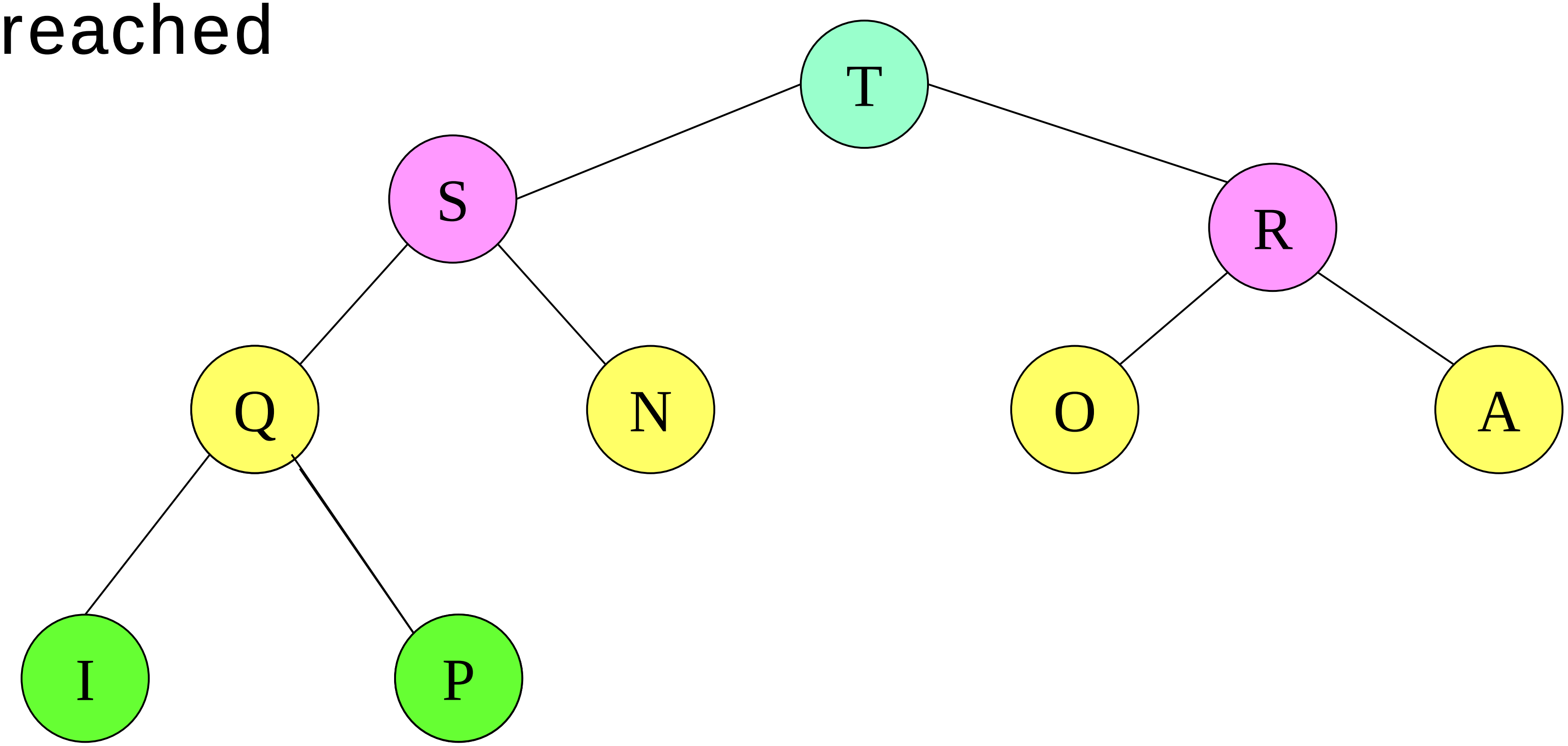
Array representation

T	S	R	P	N	O	A	I							

0 1 2 3 4 5 6 7 8 9

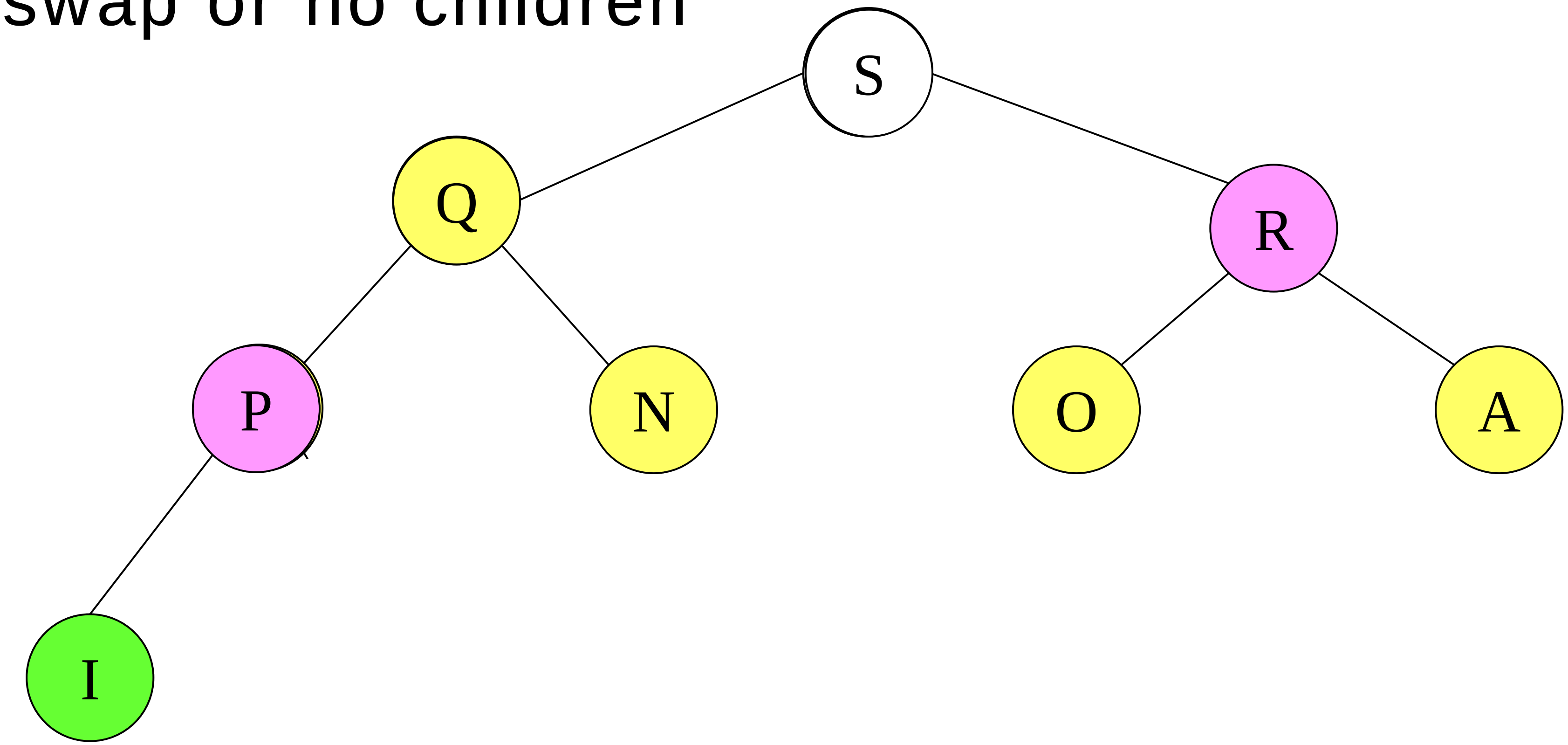
Bottom-up reheapify (swim)

- Usage: Enqueue a new element to the first empty place in the lowest level, restore heap order by bottom-up reheapify
- Swim: If the child has higher prio than parent – swap them. Continue until no swap or root is reached



Top-down reheapify (sink)

- Usage: Dequeue – remove the top element and replace it with the last (rightmost) element in the bottom level – do a top-down reheapify (sink)
- Sink: If the node has lower priority then any child – swap it with highest priority child. Continue until no swap or no children



Heap performance

- Simple to implement
- Theoretical O(log(N)) for both enqueue and dequeue
- Optimal!?

Heap sort

- Enqueue elements to be sorted
- Dequeue the sorted sequence

Heap practical problems

- Poor cache performance

- Solution – store the elements (levels) in the array in different order (see You're doing it wrong)

Heap

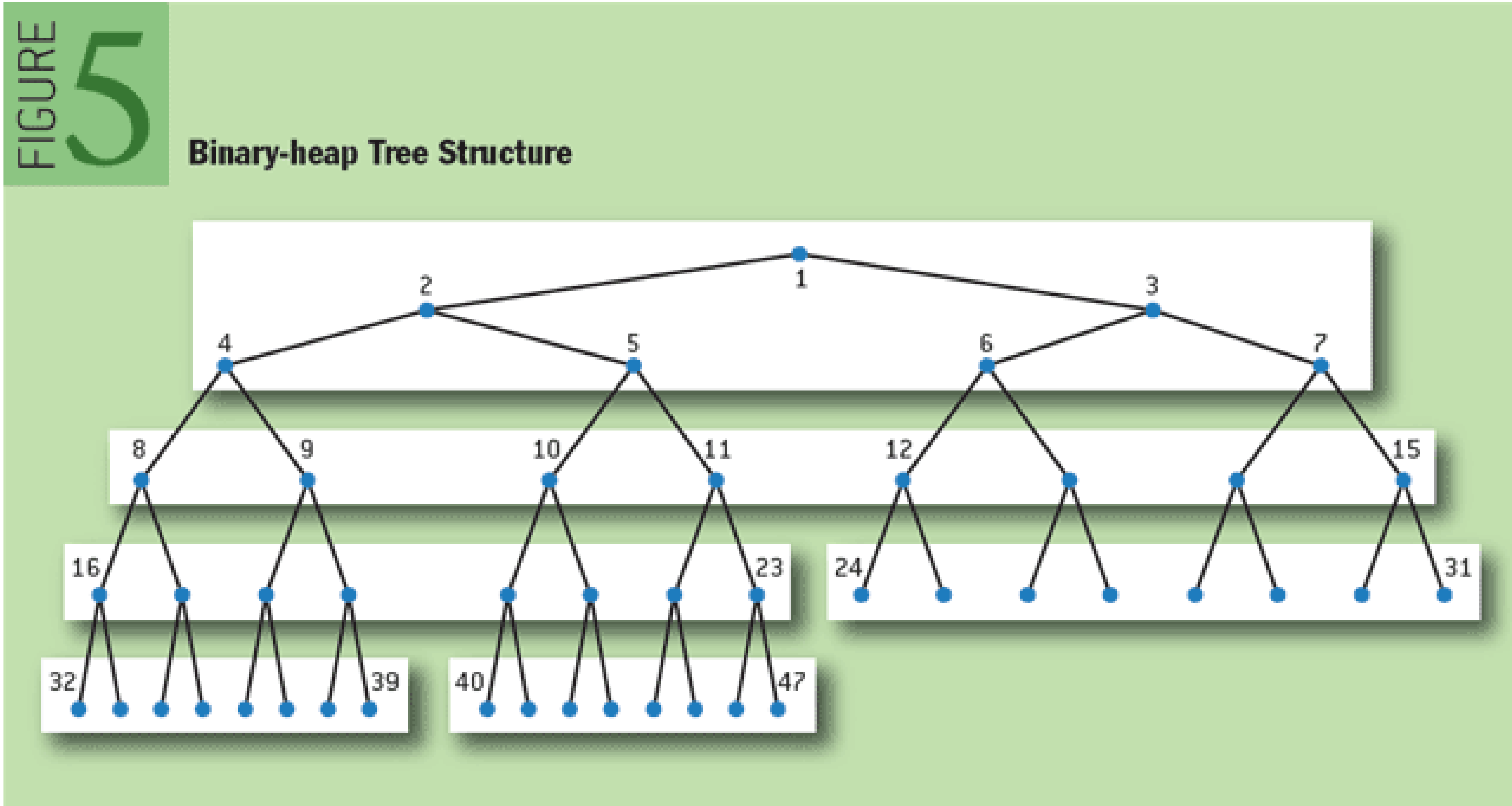
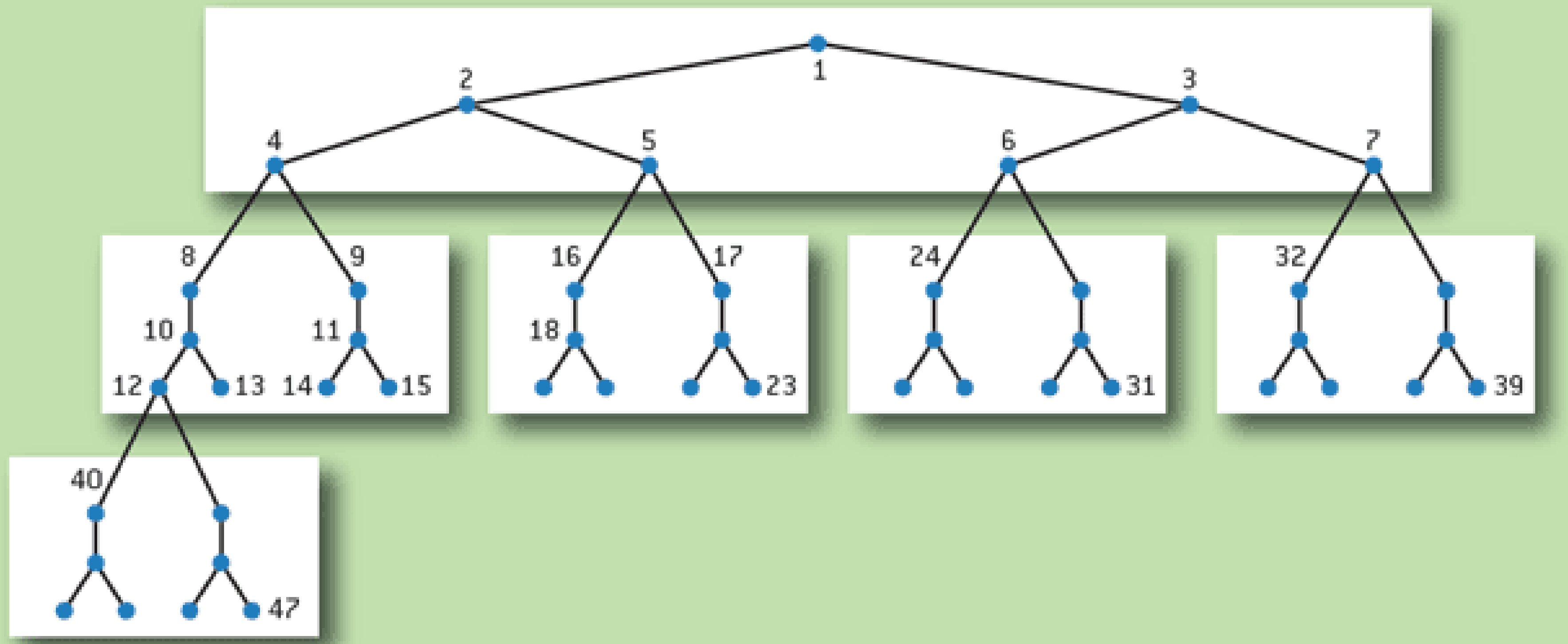


FIGURE 5
Binary-heap Tree Structure
1
2
3
4
5
6
7
8
9
10
11
12
15
16
23
24
31
32
39
40
47

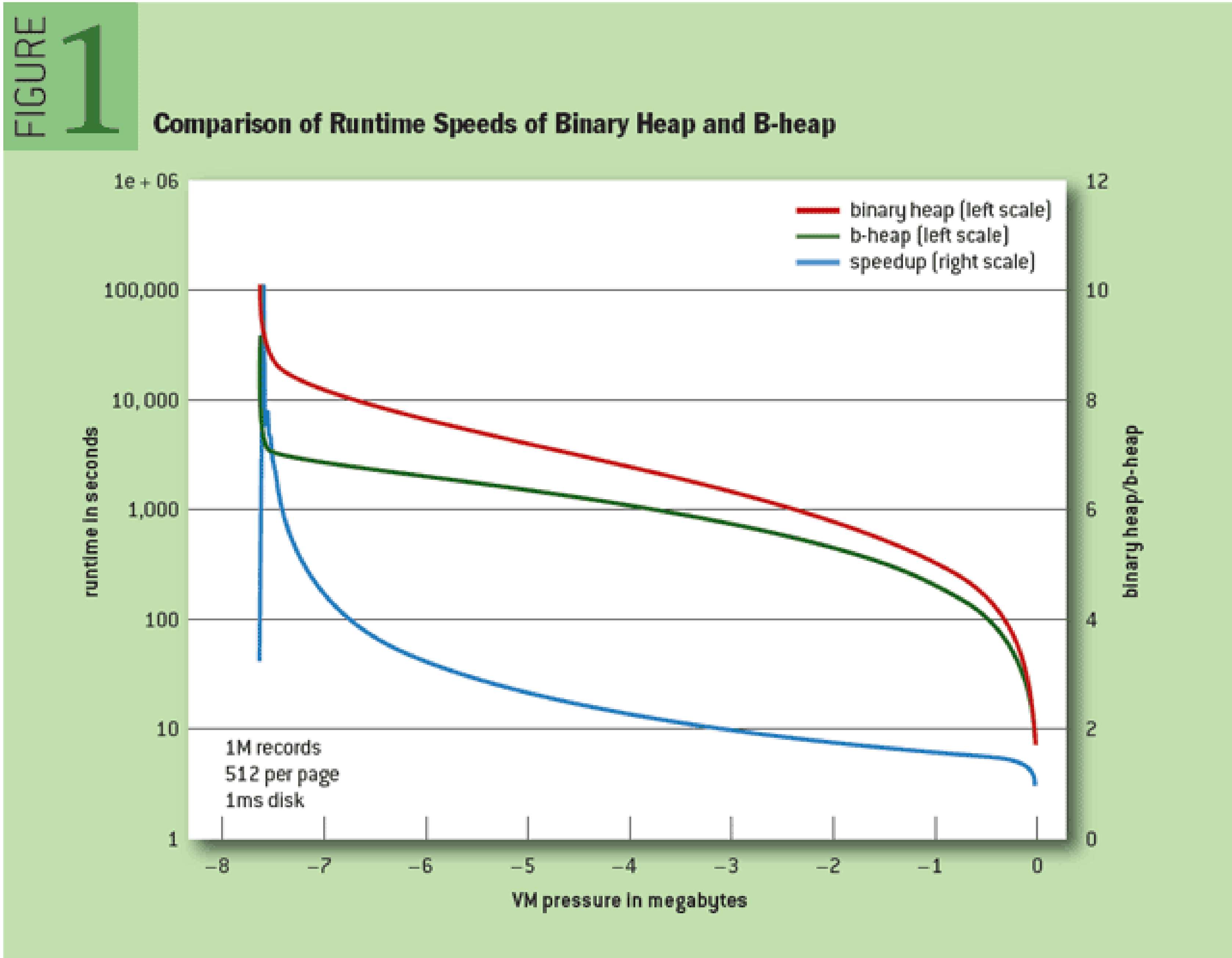
B-heap

FIGURE 6

B-heap Tree Structure

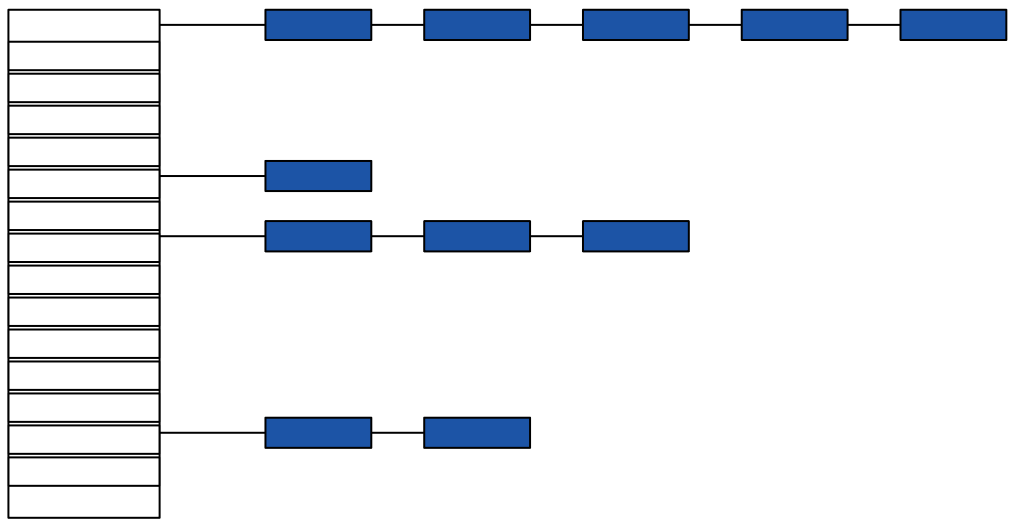


Heap vs B-Heap



Special purpose priority queues

- Process/thread scheduling in OS
  - Fixed number of priorities, eg. 0-40
  - Array of linked lists, one list per priority
  - Enqueue and dequeue is O(1)

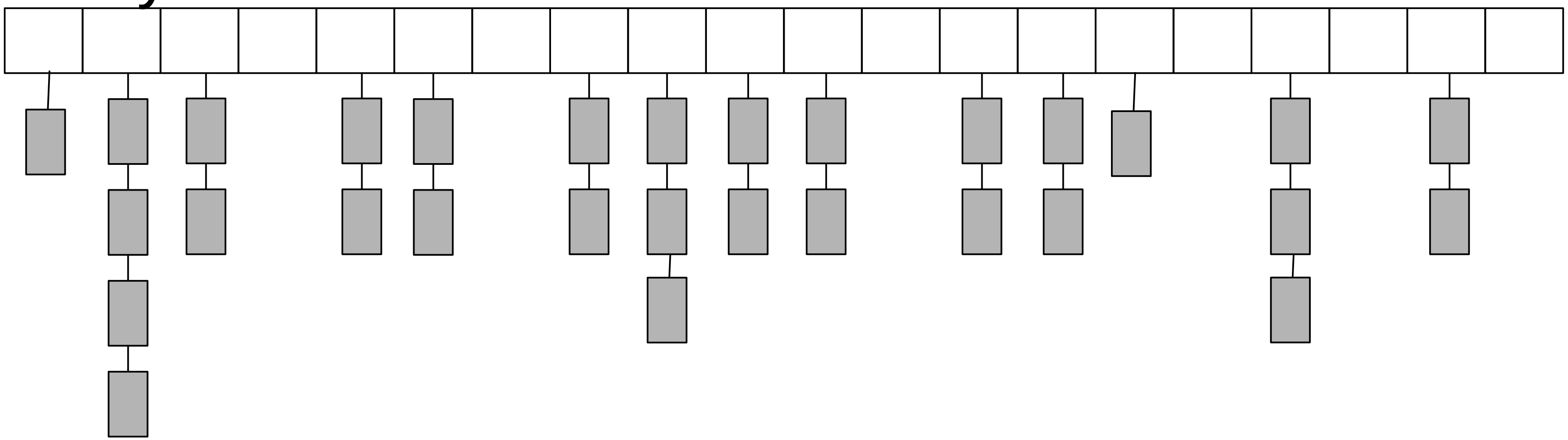


Special purpose priority queues

- Discrete event simulation – priority is timestamp of when the event is to take place in simulated time
- Idea: combine ideas from OS scheduling, resizeing arays with an ordinary (paper) calendar
- Calendar queue: Near O(1) enqeue and dequeue on average, breaks for certain distributions

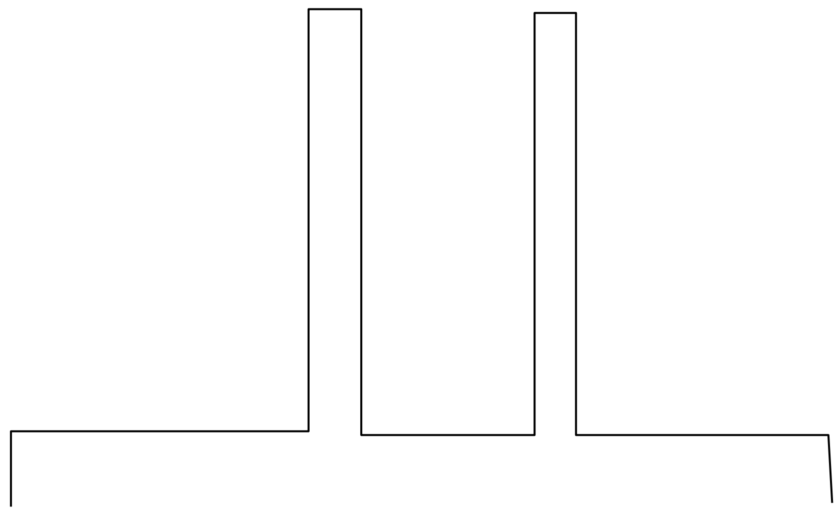
Calendar queue

- Resize when size doubles or halves
- When resizing:
  - Calculate new number of days = N/2
  - Calculate day length so that the average number of elements per day will be 2
- Sort the day lists
- When dequeue – scan the days to find next item on this year



Calendar queue

- Works well as long as the priorities are relatively evenly distributed
- Breaks for some special probability distributions



Fast general purpose priority queue

- **Splay tree**, Tarjan & Sleator

Priority queues: other applications

- They can be used as a basis for other algorithms
  - Sorting
  - Graph-searching
  - Data compression