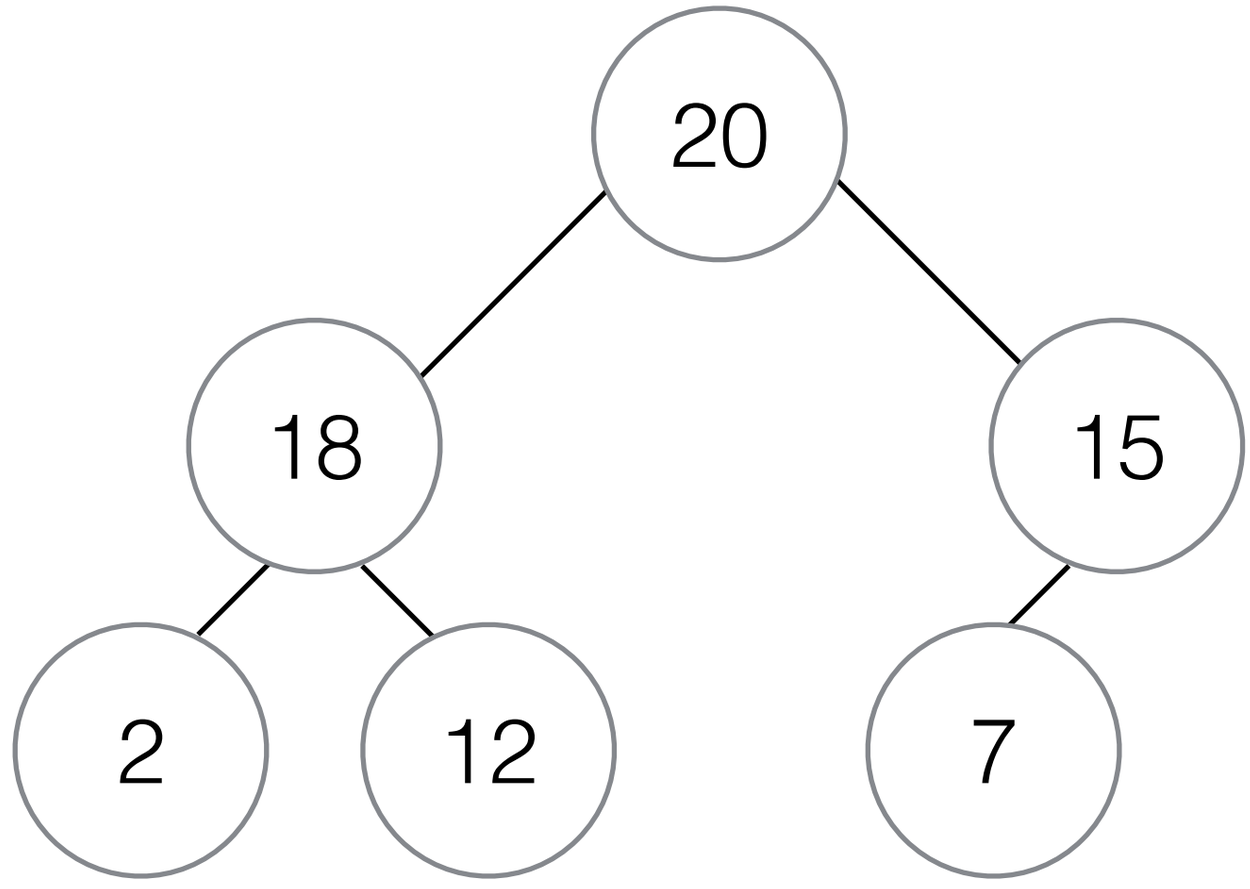
Heaps

Anton Gerdelan
<gerdela@scss.tcd.ie>

Heaps

(not to be confused with *the* heap memory)

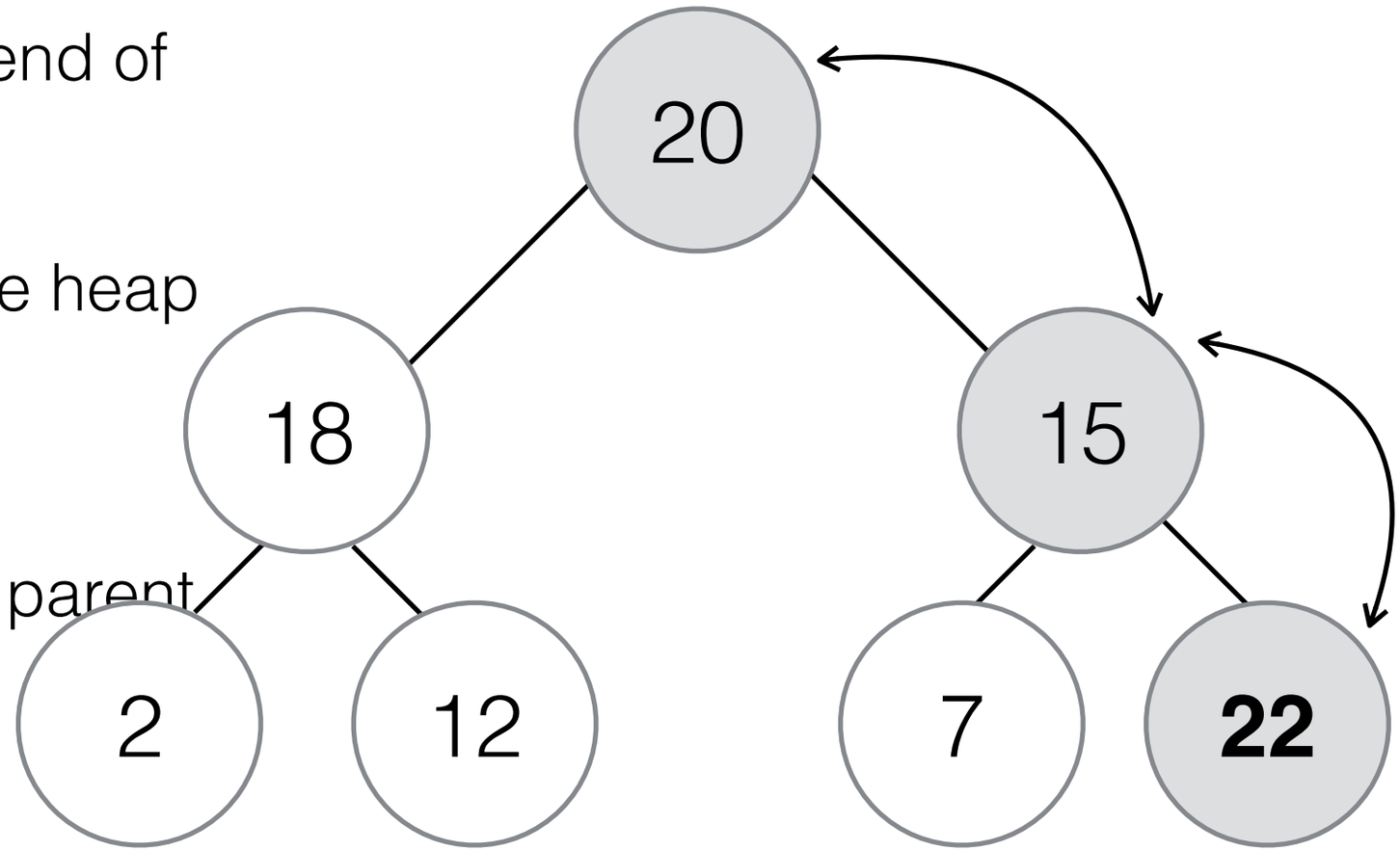
- **Binary tree**
  (not necessarily a BST)
- value at node **>=** value of children
- tree is perfectly **balanced**
- leaves are all 'as **left as possible**'
- heap is easily stored in an array
  - work from top to bottom, left to right



20	18	15	2	12	7

Heaps

- To add to heap - add to end of array
- but tree may no longer be heap
- to-rebalance tree
  - compare new value to parent
  - swap if bigger
  - repeat until no swap or at root
  - swap array values



20	18	15	2	12	7	**22**

22	18	**20**	2	12	7	**15**

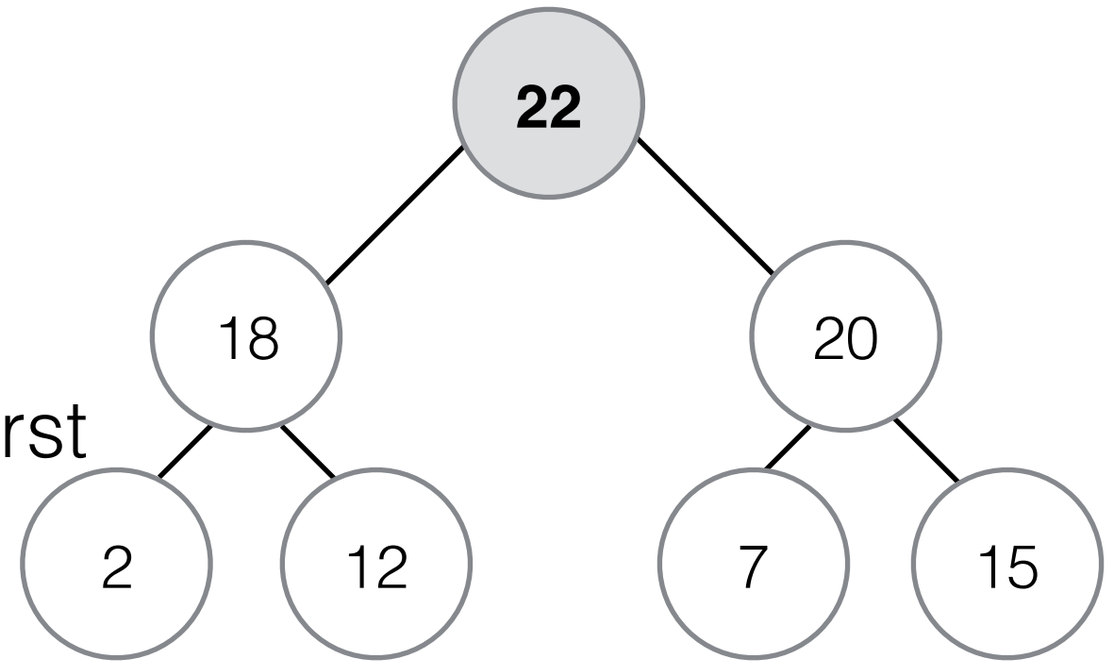
Heap Array Rules

- for any i < n/2
  - heap[i] >= heap[2 * i + 1]
  - heap[i] >= heap[2 * i + 2]

```
heap[heap_len] = new_value;
heap_len++;
child = heap_len - 1;
parent = (child - 1) / 2;
while( child != 0 ) {
  if( heap[parent] >=
      heap[child] ){
    break;
  }
  swap( heap[parent],
        heap[child]);
  child = parent;
  parent = ( child-1 ) / 2;
}
```

Heap as a Priority Queue

- item is added to queue
- value is the priority
- highest priority items taken first
- i.e. root downwards
- to remove root from queue -
  - copy last element into [0]
  - decrement queue length counter



22	18	20	2	12	7	15

Removing Root

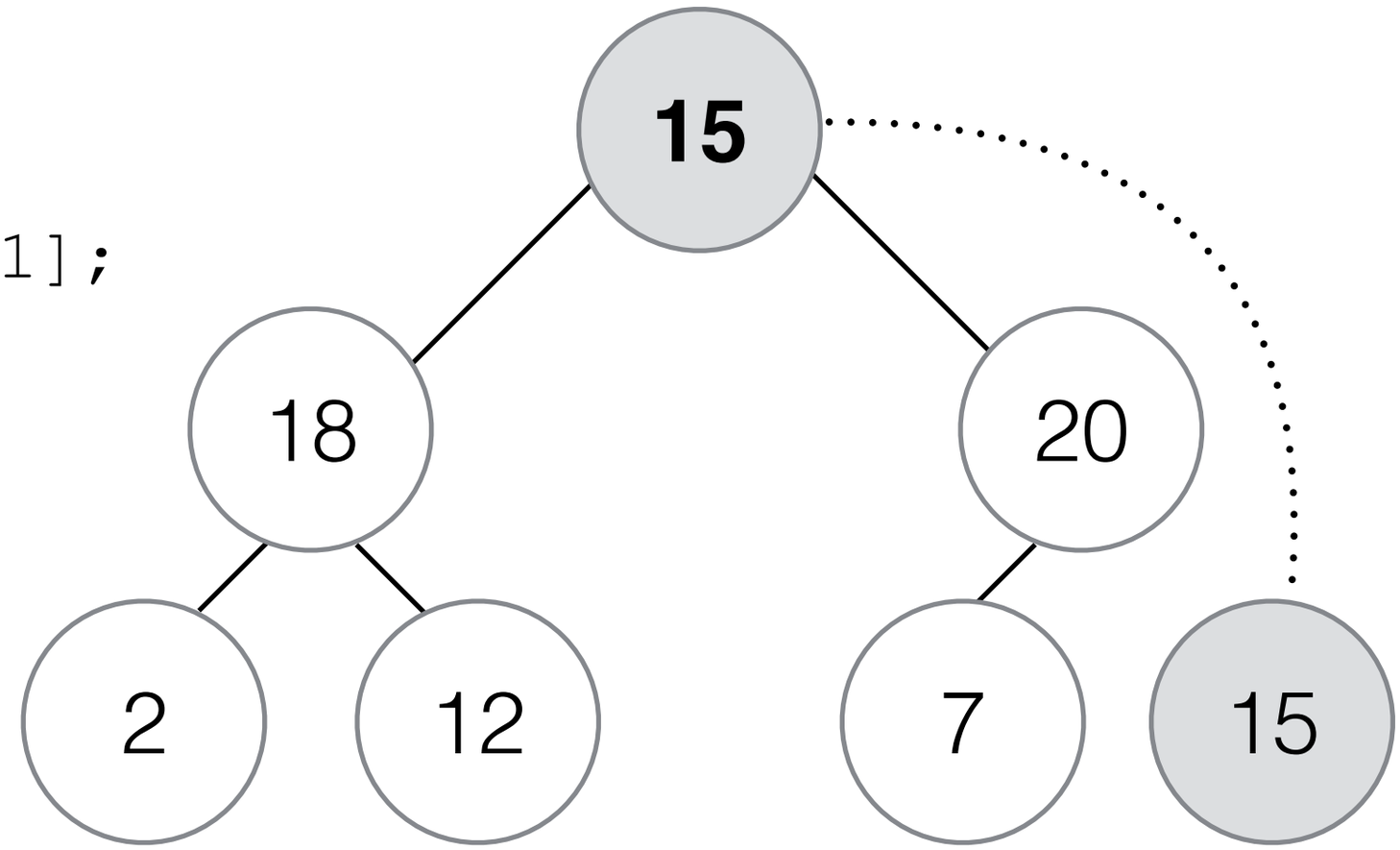
- `result = heap[0];`
  `heap[0] =`
  `  heap[heap_len - 1];`
  `heap_len--;`

- Queue is no longer balanced

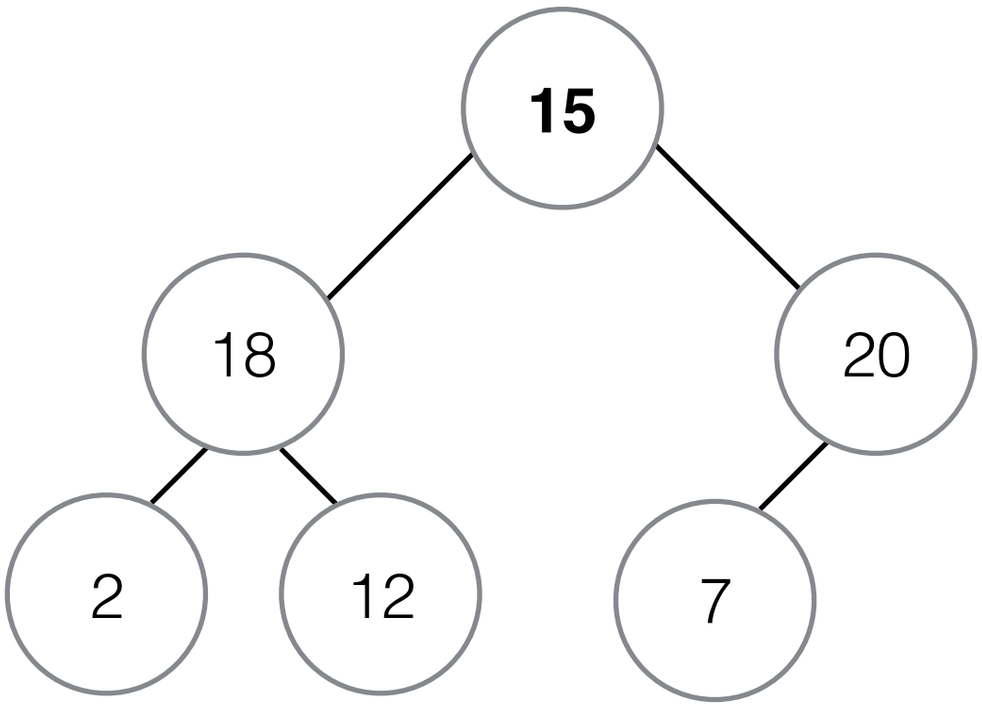
- work down from root

  - if any child is greater than root

  - swap



15	18	20	2	15	7	~~15~~



15	18	20	2	15	7

step-through

```
void movedown( int first ) {
  int parent = first;
  int max_child = 2 * parent + 1;

  while( max_child < heap_len ) {
    //i has 2 children
    if( max_child < heap_len - 1) {
      //right child is bigger
      if( heap[max_child] <
          heap[max_child + 1] ) {
        max_child++;
      }
    }
    if( heap[parent] >=
        heap[max_child]) {
      break;
    }
    swap( heap[parent],
          heap[max_child] );
    parent = max_child;
    max_child = 2 * parent - 1;
  }
}
```

Heapsort

- The heap was created for Heapsort by JWJ Williams (1964).
  - Build a heap
  - Algorithm removes biggest value from heap
    - add to end of new list/array
    - update heap to maintain balance
    - when heap is empty -> sorted array

Binary Heaps

- **Space** O(n). O(1) aux. space used in Heapsort.
- **Search** O(n)
- **Insert** O(1) average, O(log n) worst
- **Delete** O(log n) average, O(log n) worst
- **Heapsort** O(n) best, O(n log n) average, O(n log n) worst
- Worse cache performance than merge-sort - **why**?
- Not a **stable** sort
- Hard to parallelise
- Better worst-case time complexity than quicksort