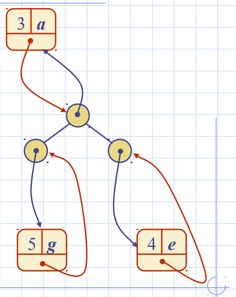
Adaptable Priority Queues



Items and Priority Queues

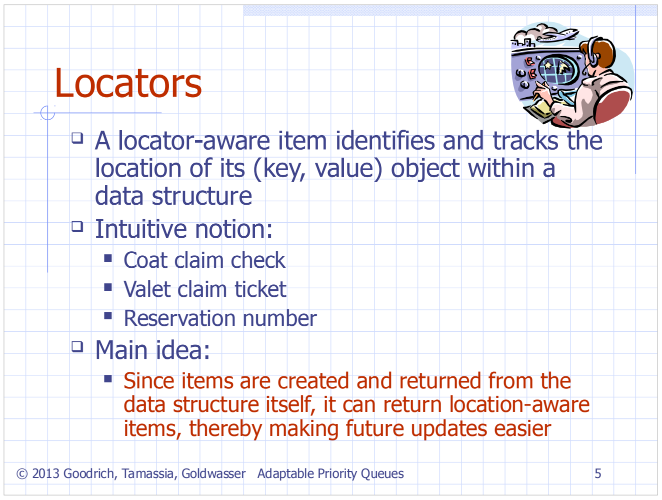
- An item stores a (key, value) pair
- Item fields:
  - _key: the key associated with this item
  - _value: the value paired with the key associated with this item
- Priority Queue ADT:
  - add(k, x) inserts an item with key k and value x
  - remove_min() removes and returns the item with smallest key
  - min() returns, but does not remove, an item with smallest key
  - len(P), is_empty()

Example

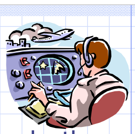
- Online trading system where orders to purchase and sell a stock are stored in two priority queues (one for sell orders and one for buy orders) as (p,s) entries:
  - The key, p, of an order is the price
  - The value, s, for an entry is the number of shares
  - A buy order (p,s) is executed when a sell order (p',s') with price p'≤p is added (the execution is complete if s'≥s)
  - A sell order (p,s) is executed when a buy order (p',s') with price p'≥p is added (the execution is complete if s'≥s)
- What if someone wishes to cancel their order before it executes?
- What if someone wishes to update the price or number of shares for their order?

Methods of the Adaptable Priority Queue ADT

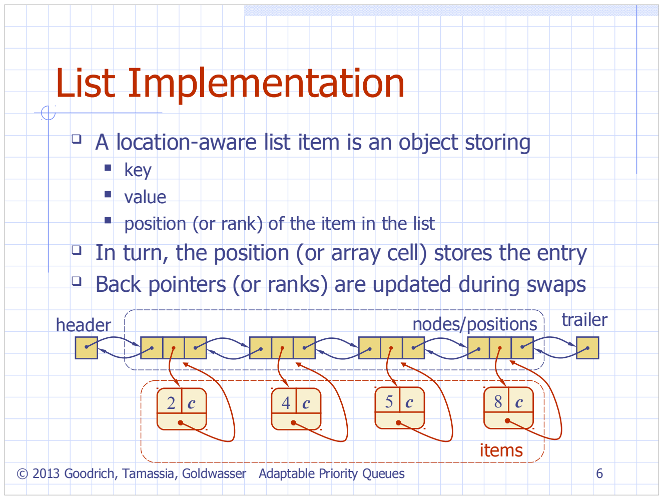
- remove(loc): Remove from P and return item e for locator loc.
- update(loc,k,v): Replace the key-value pair for locator, loc, with (k,v).



Locators

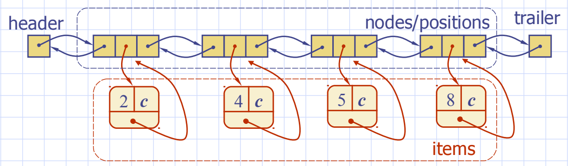


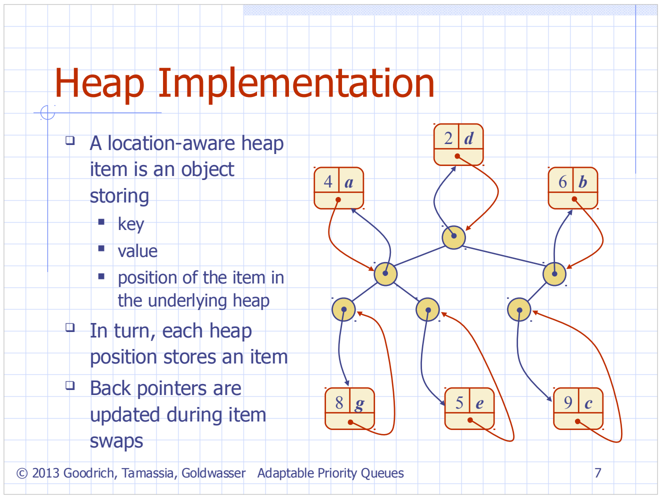
- A locator-aware item identifies and tracks the location of its (key, value) object within a data structure
- Intuitive notion:
  - Coat claim check
  - Valet claim ticket
  - Reservation number
- Main idea:
  - Since items are created and returned from the data structure itself, it can return location-aware items, thereby making future updates easier



List Implementation

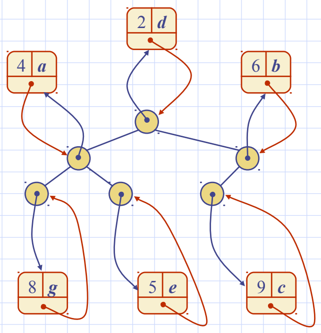
- A location-aware list item is an object storing
  - key
  - value
  - position (or rank) of the item in the list
- In turn, the position (or array cell) stores the entry
- Back pointers (or ranks) are updated during swaps

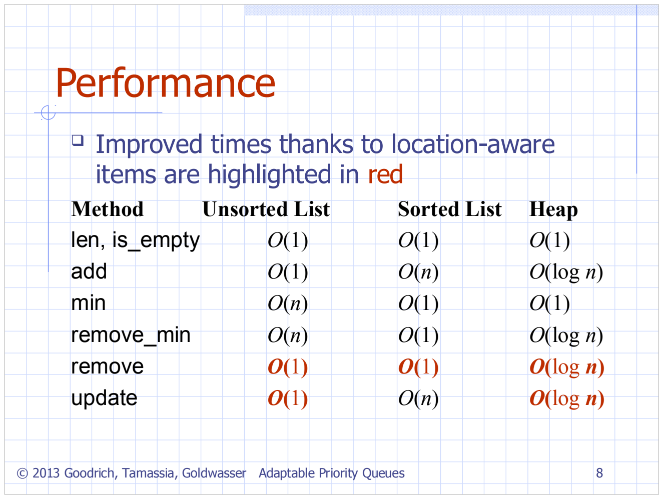




Heap Implementation

- A location-aware heap item is an object storing
  - key
  - value
  - position of the item in the underlying heap
- In turn, each heap position stores an item
- Back pointers are updated during item swaps





Performance

- Improved times thanks to location-aware items are highlighted in red

Method	Unsorted List	Sorted List	Heap
len, is_empty	$O(1)$	$O(1)$	$O(1)$
add	$O(1)$	$O(n)$	$O(\log n)$
min	$O(n)$	$O(1)$	$O(1)$
remove_min	$O(n)$	$O(1)$	$O(\log n)$
remove	$\boldsymbol{O(1)}$	$\boldsymbol{O(1)}$	$\boldsymbol{O(\log n)}$
update	$\boldsymbol{O(1)}$	$O(n)$	$\boldsymbol{O(\log n)}$