Panel 1

Purely Functional Data Structures

Brian Howard
DePauw University

1

Panel 2

Why?

- Functional...
- Immutable ("values"):
  - Copy by sharing
  - Thread-safe
  - Persistent?

2

Panel 3

Persistent (adj.): not ephemeral.

Fully persistent: can access and modify all versions

Partially persistent: can access all versions and modify current version

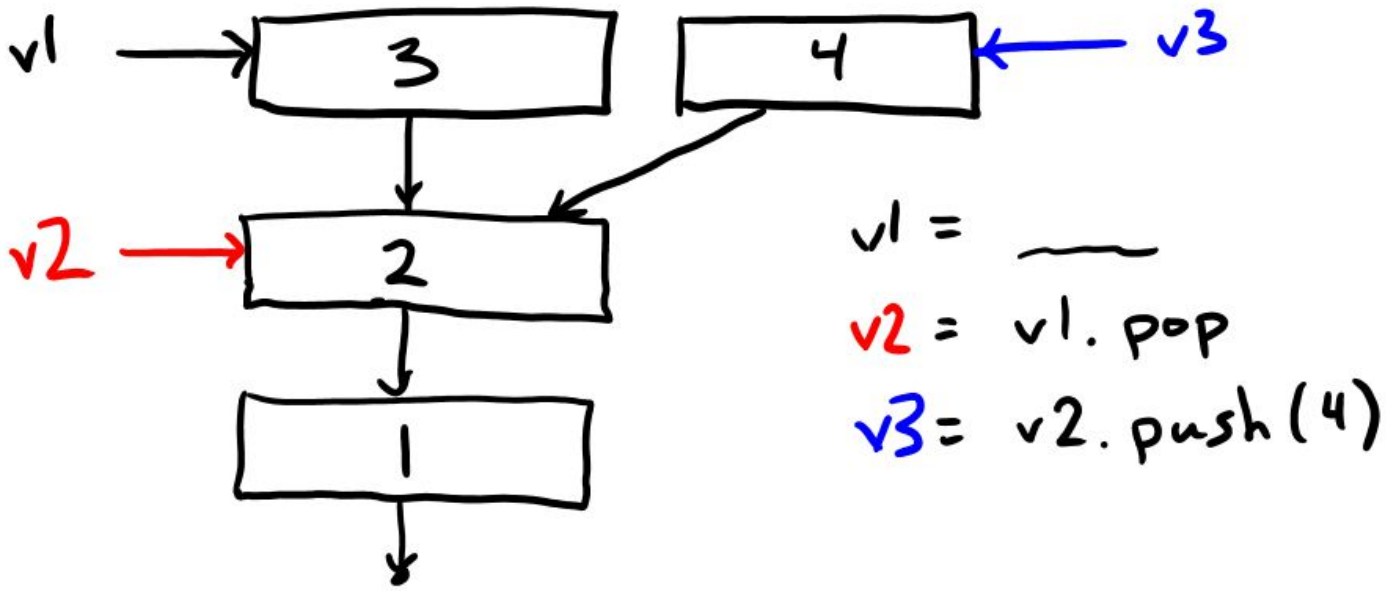
Confluently persistent: can merge previous versions into new version

3

Panel 4

Linked List, used as a Stack

Fully Persistent, O(1) push/pop/top

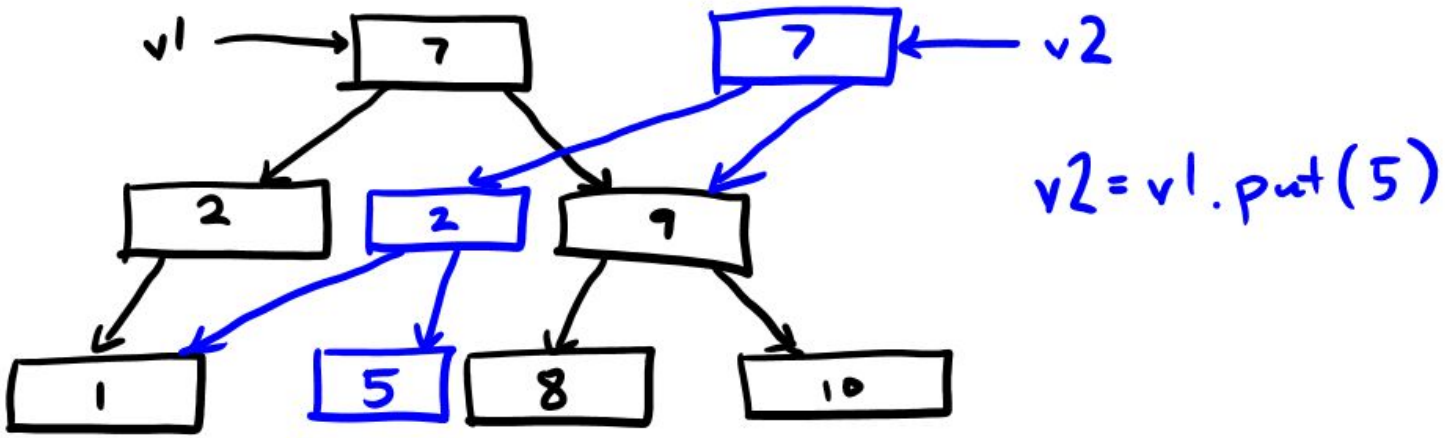


4

Panel 5

Binary Search Tree (Set or Map)

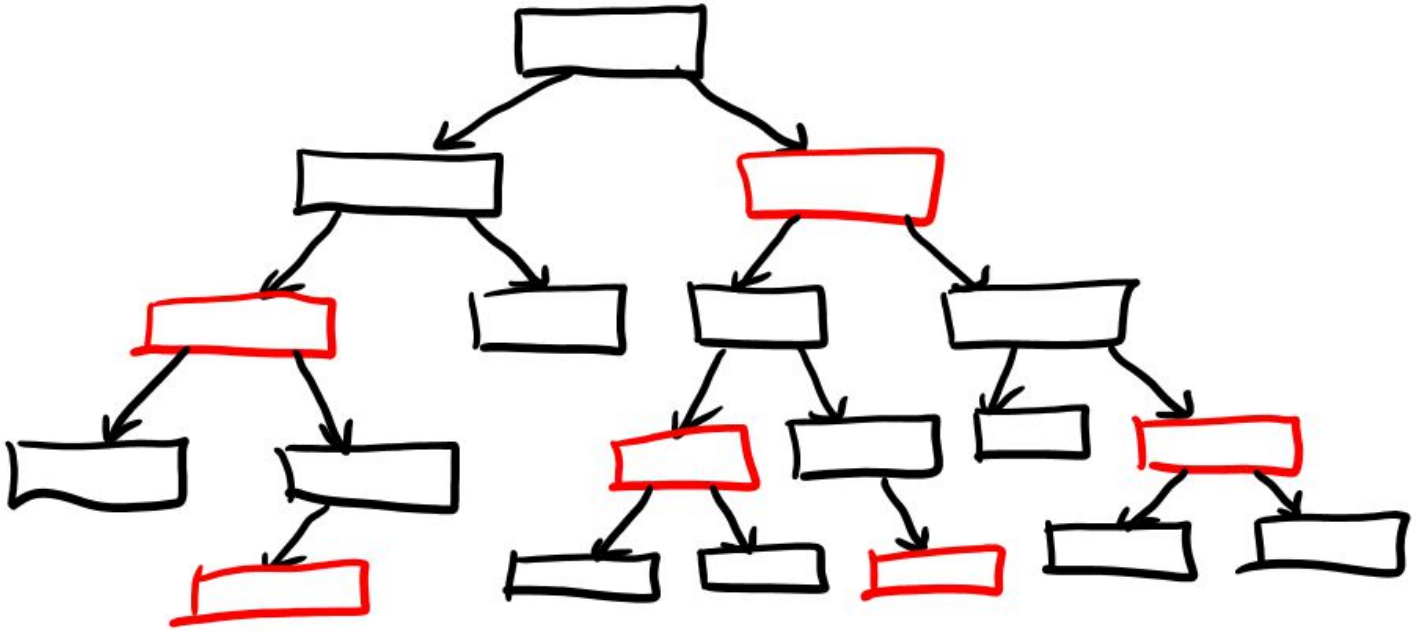
Fully Persistent, Average O(log N) get/put



5

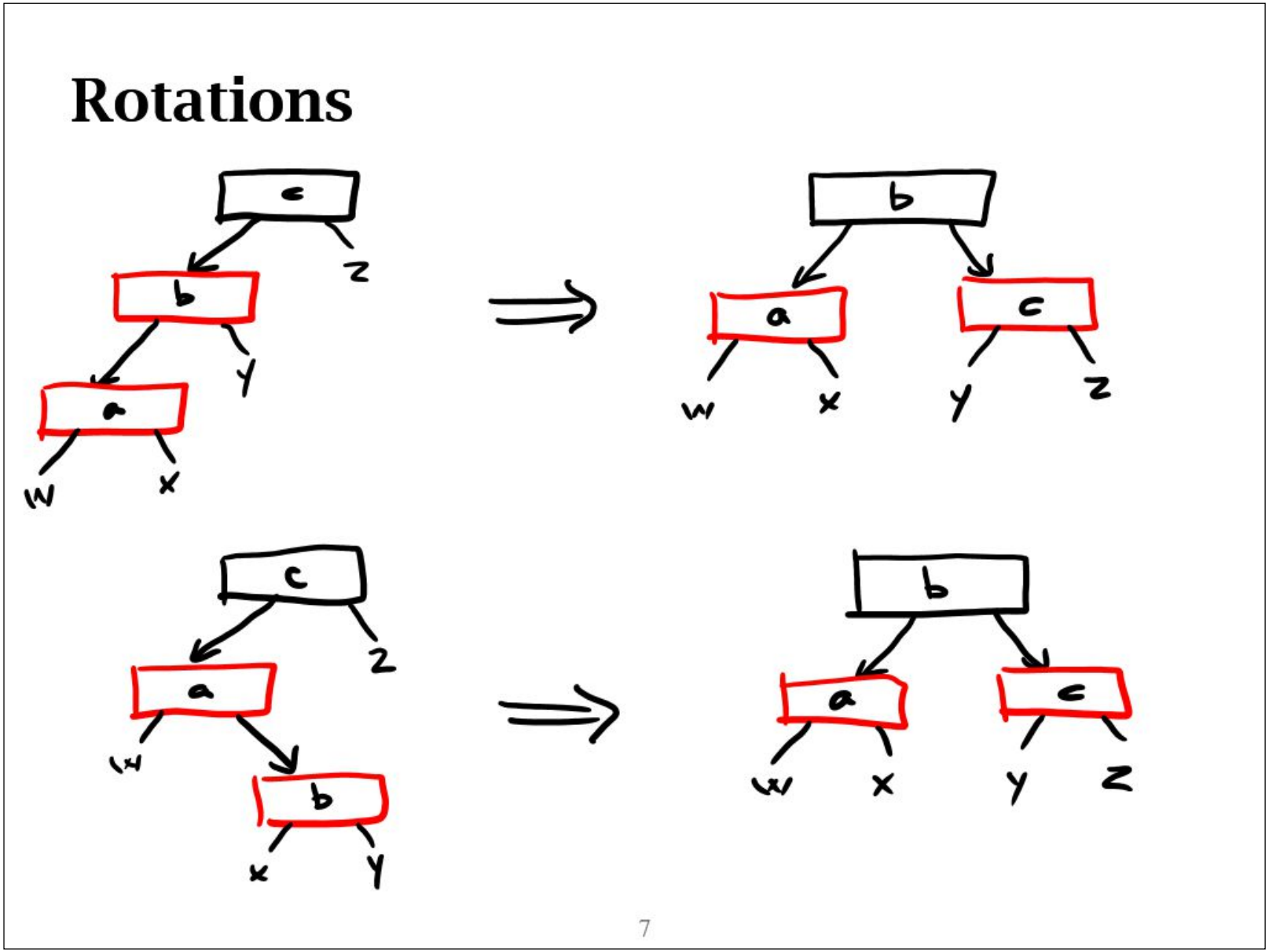
Panel 6

Red-Black Tree: add balance condition (same # black nodes per path to leaf; no adjacent red nodes on any path) Worst-case O(log N)



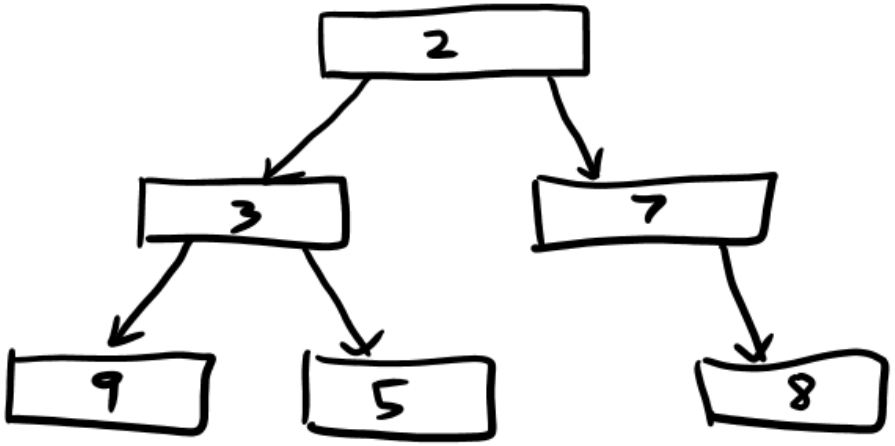
6

Panel 7



Panel 8

Priority Queue: tree with "heap order"

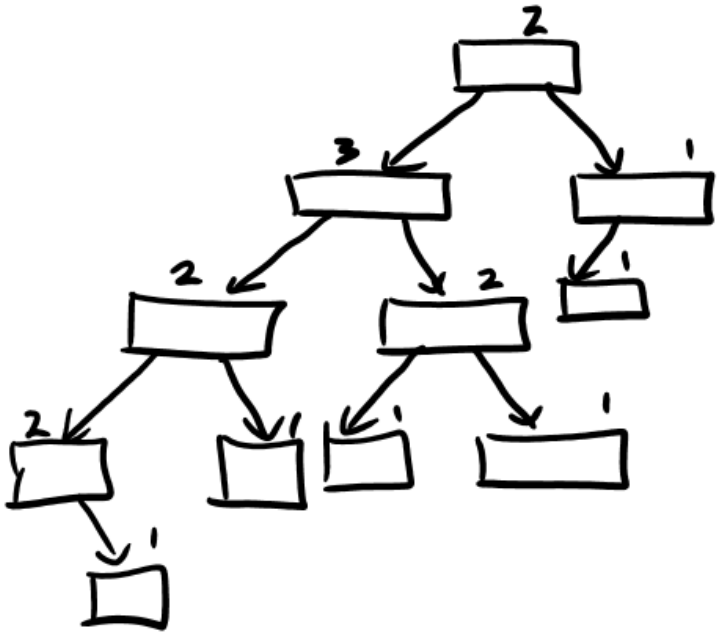


head: O(1)
enqueue/dequeue: average O(log N)

8

Panel 9

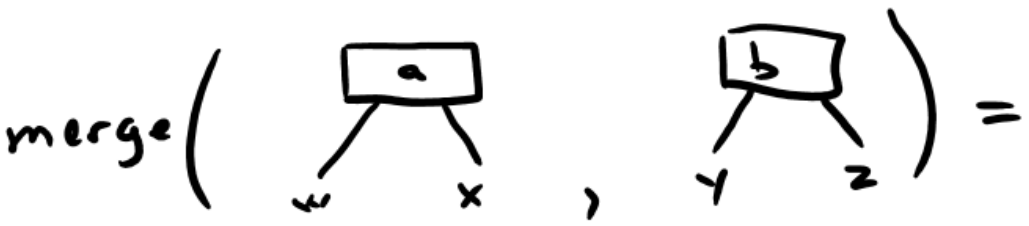
Leftist Heap: add balance condition (left children have >= "rank" -- length of right "spine")

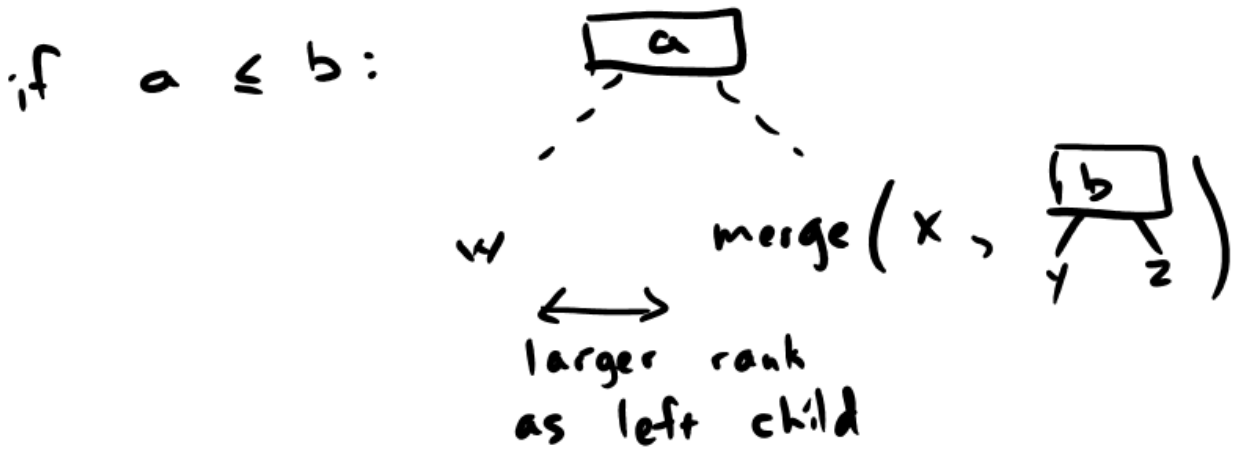


9

Panel 10

Merge operation preserves balance:

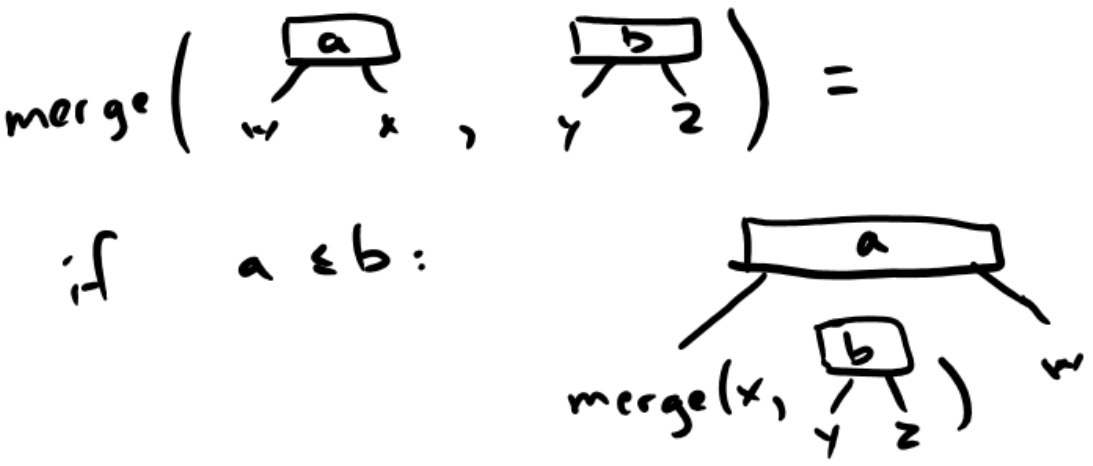




10

Panel 11

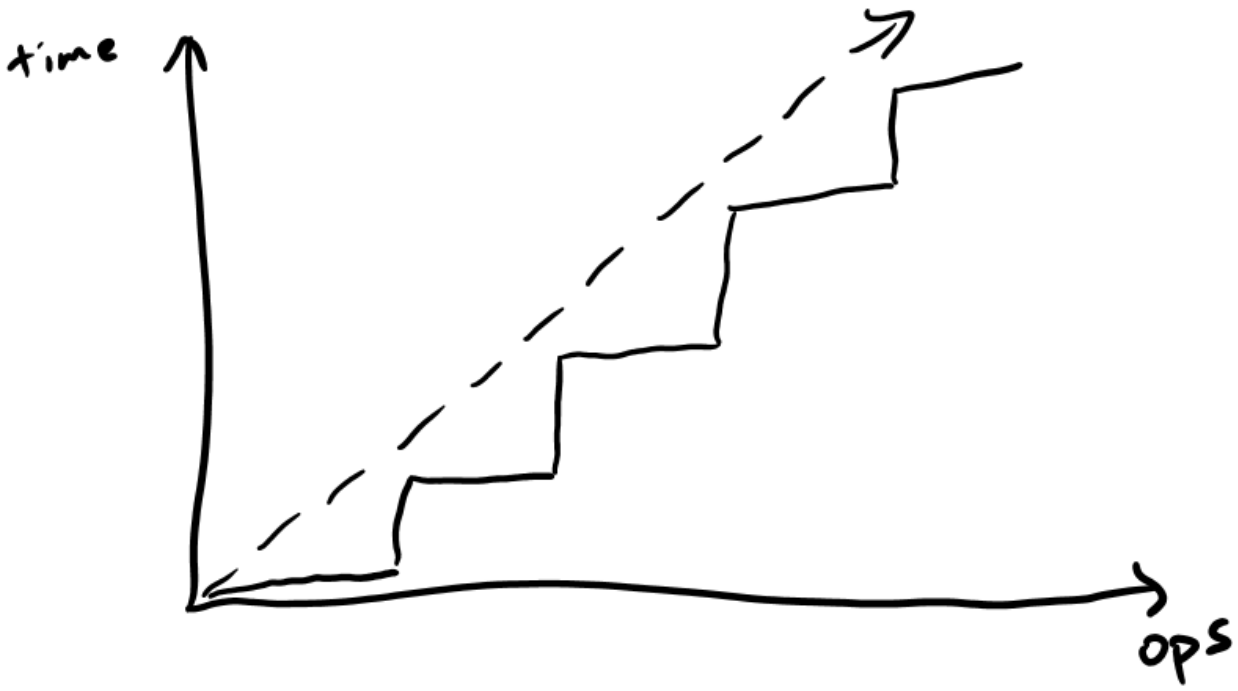
Skew Heap: *always* swap children when merging
Simpler, but only "amortized" O(log N)



11

Panel 12

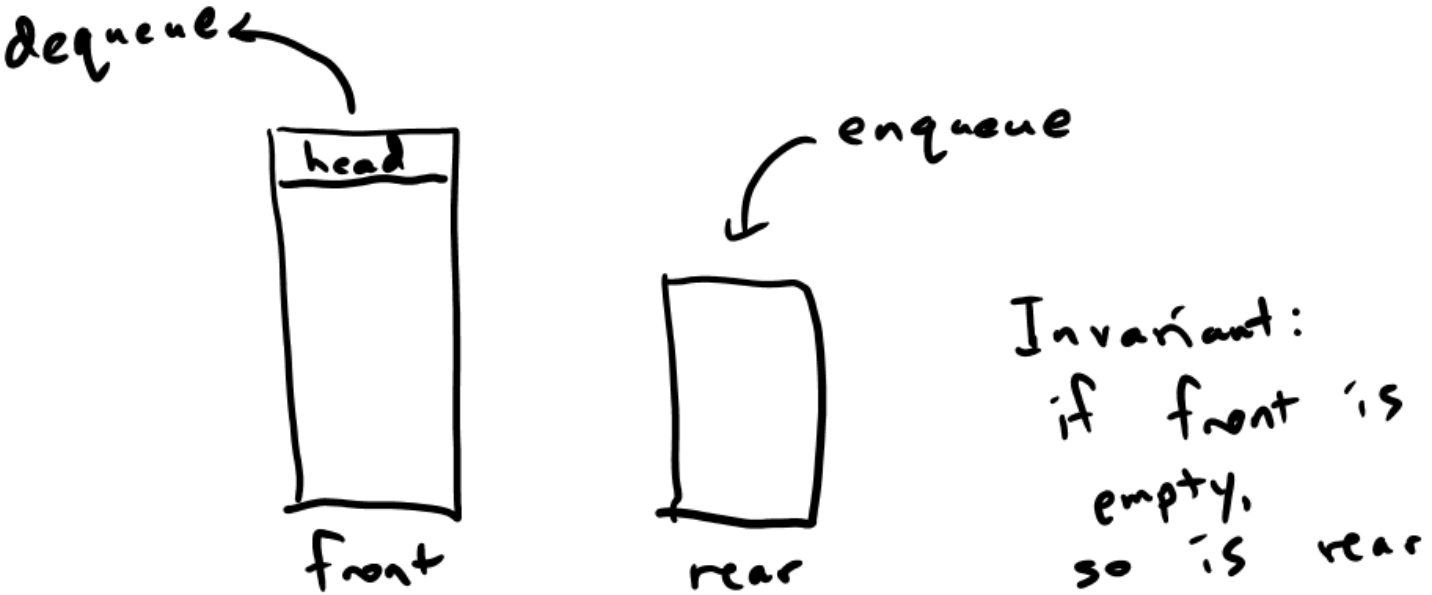
Amortization: expensive operations have been "pre-paid", so the overall average is still cheap



12

Panel 13

Batched Queue: pair of stacks worst-case O(1) head/enqueue amortized O(1) dequeue (*not persistent!*)



13

Panel 14

Real-Time Queue (Okasaki): use "laziness" to "schedule" an incremental reversal of rear to front

Fully-persistent, worst-case O(1) (but not technically immutable!)