# Theory and Practice of Chunked Sequences

**Umut Acar**   **Arthur Charguéraud**   **Mike Rainey**

Inria & Carnegie Mellon University

ESA   2014/09/08

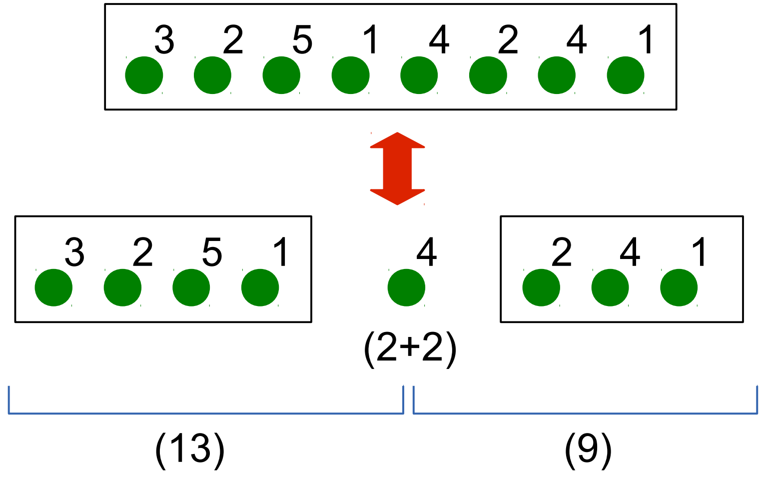
# Goal

**Ephemeral deques (double-ended queues) with:**

→ push and pop at the two ends, in O(1) amortized

→ concat, and split at arbitrary indices, in O(log n)

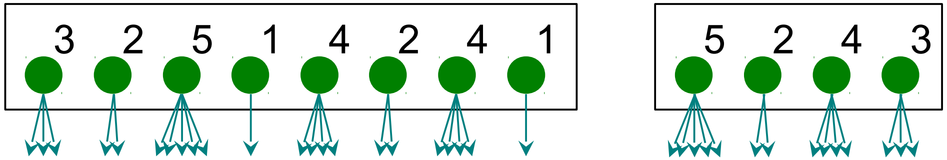
**Weighted split operation:** (e.g. split at 13)



# Motivation

**Application to parallel BFS and parallel DFS**

→ each processor push/pop vertices from its frontier
→ split is used for dynamically distributing the load
→ concatenation is used to merge the new frontiers




**Requirements:**

→ push and pop must be very efficient
→ split and concatenation is sublinear time

# Contribution

**Amortized O(1) push/pop and O(log N) concat/split**

**Prior work:**

→ Kaplan and Tarjan (1996)
→ Hinze and Parterson (2006)

Purely functional data structures (confluently persistent).
Yet, very large constant factors (even if made ephemeral).

**This work:** ephemeral catenable/splittable deques with small constant factors (e.g., not far from C++ STL deques).

# Overview

**(1) Chunked sequences**

→ assume a potentially-slow deque data structure

→ construct a fast deque data structure, using chunks

- amortize allocations in worst-case push-pop scenarios
- ensure space efficiency in worst-case concat scenarios

**(2) Bootstrapped chunked sequences**

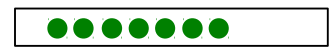
→ build a stand-alone, fast, catenable/splittable deque

→ use structural decomposition and recursive slowdown

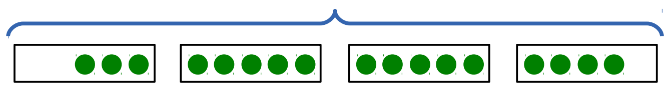
(Dietz 1982; Buchsbaum & Tarjan 1995; Kaplan & Tarjan 1996)

# Challenges with chunks

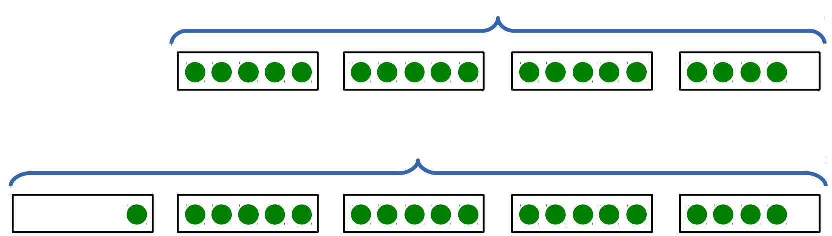
**A chunk = fixed-capacity ring buffer** (repr. as an array)



**A deque of chunks** (e.g. C++ STL deques)

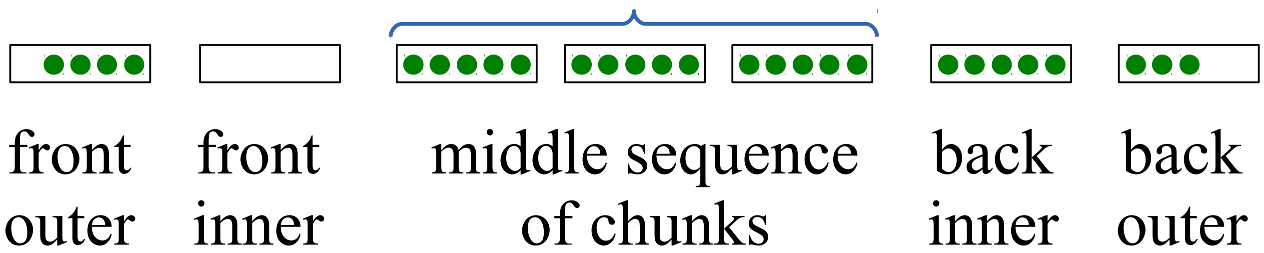


**Challenge:** (iterated push/pop operations)



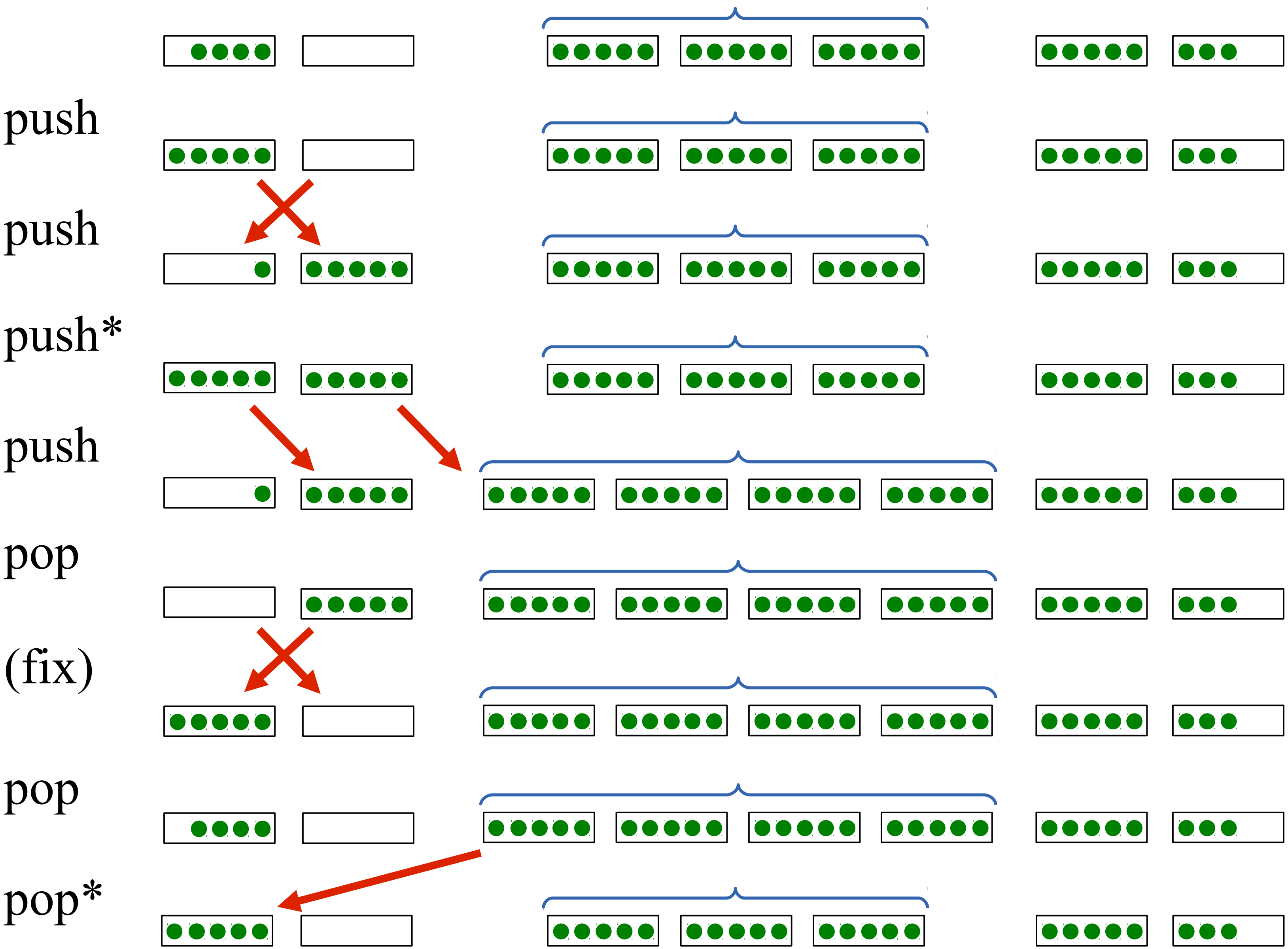
# Our chunked sequence

**Approach:** place two special chunks on each side




**Invariant:** the inner chunks must be either empty or full.

# Implementation of push and pop



# Amortization with chunked sequences

**Theorem:** the amortized cost of push (including pop) is

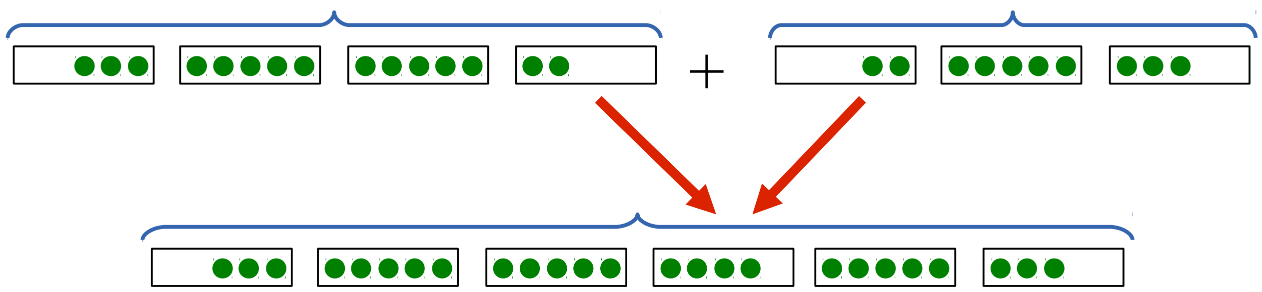
$$C + \frac{A+M}{K} + \mathrm{O}(1)$$

**where:**

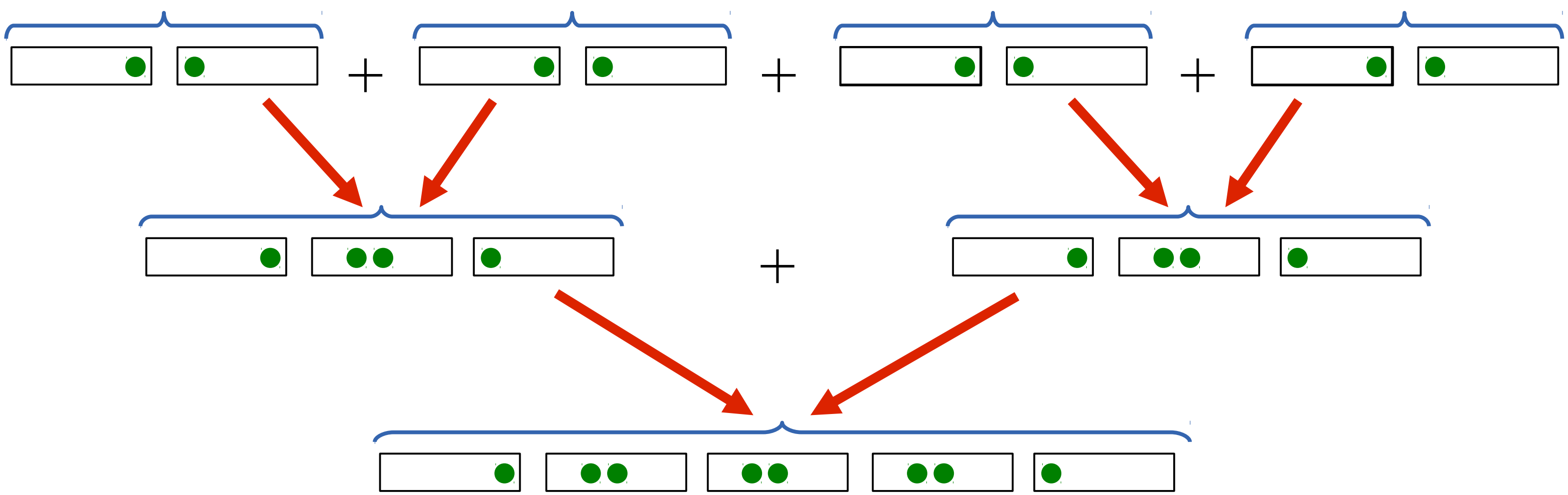
- $\boldsymbol{C}$ cost of push (including associated pop) in a chunk
- $\boldsymbol{A}$ cost of allocation (including deallocation) of a chunk
- $\boldsymbol{M}$ cost of push (including pop) in the middle sequence
- $\boldsymbol{K}$ capacity of a chunk

# Challenges with concatenation

## Concatenation of deques of chunks (with merge)

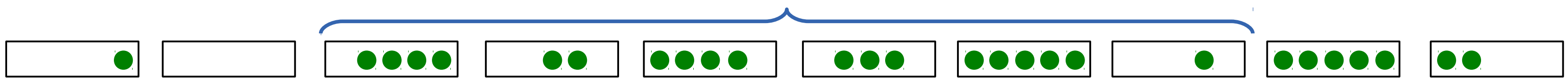


## Worst-case scenario:

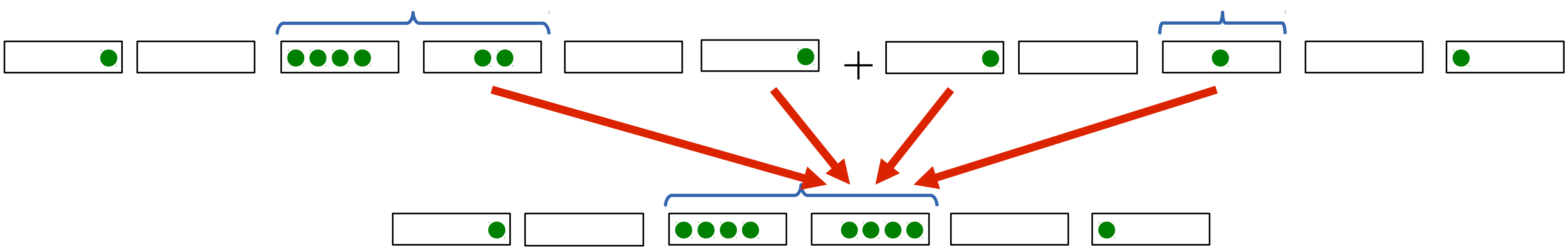


# Implementation of concatenation

**Invariant:** any two consecutive chunks in the middle sequence must store a total number of more than K items.

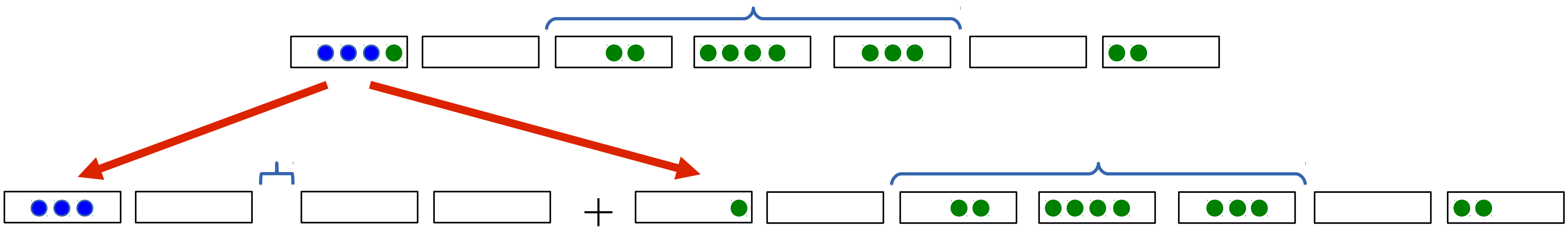


**Concatenation:** (up to 4 chunks need to be merged)

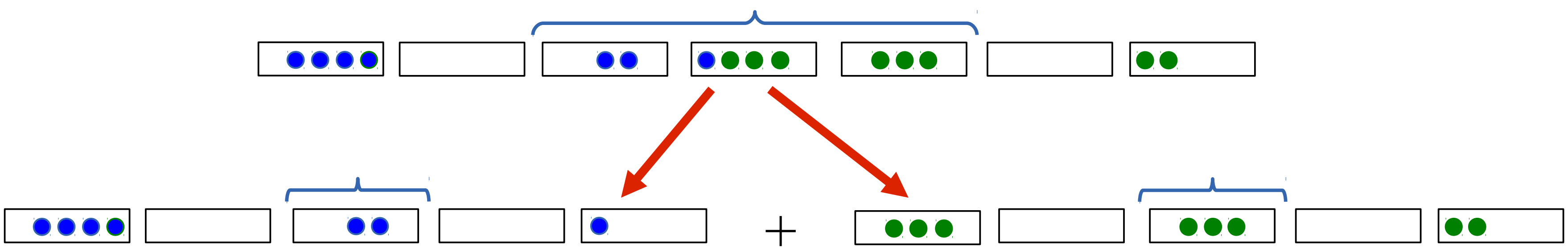


# Implementation of split

## Case 1:

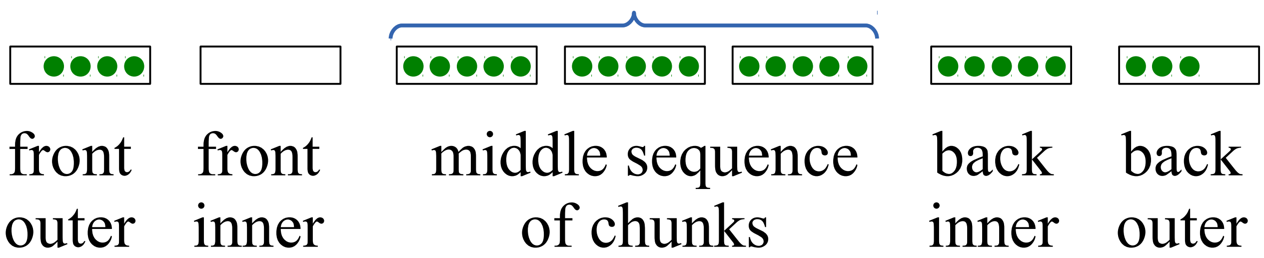


## Case 2:



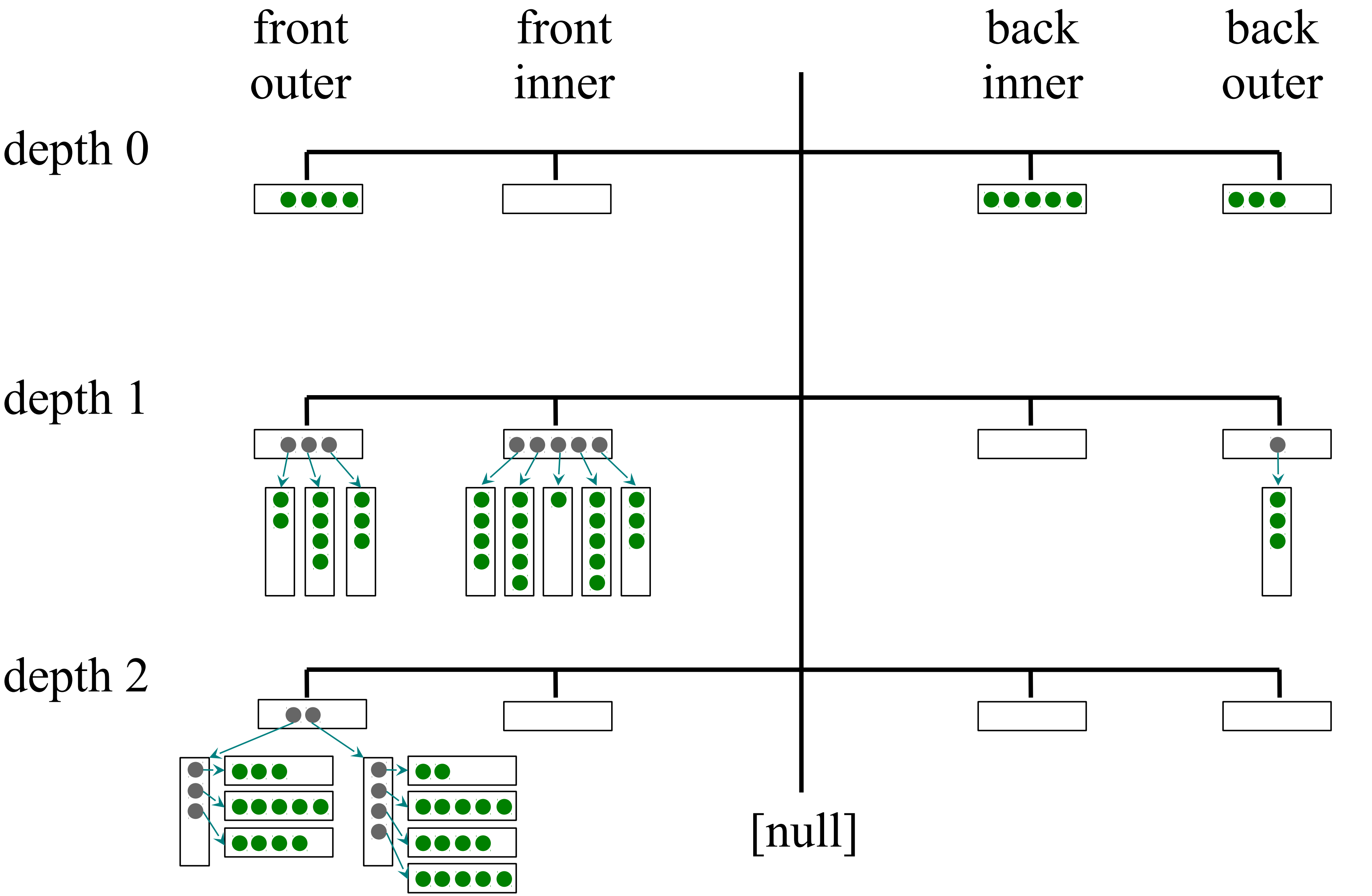
# Towards bootstrapping

**Summary:** given a potentially-slow catenable/splittable deque, we built a catenable/splittable deque structure with small constant factors, even in worst-case scenarios.




**Next step:** implement the middle sequence of our chunked sequence using... our chunked sequence. Do so recursively.

# Bootstraped chunked sequence



# Efficiency analysis

**Theorem:** the depth is at most

$$\left\lfloor \log_{(K+1)/2} n \right\rfloor + 1$$

**Remark:** depth is bounded by 7 for all practical purposes.

**Theorem:** push/pop has cost O(1), with a small constant

**Theorem:** concat and split have cost

$$O\left( K \ * \ \log_{(K+1)/2}\left(\min\left(n_1, n_2\right)\right) \right)$$

where $n_1$ and $n_2$ denote the size of the two parts involved.

→ compare with: $O\left( \log_2\left(\min\left(n_1, n_2\right)\right) \right)$

# Space-usage analysis

**Theorem:** asymptotic space usage is

$$\left(2 + \frac{\mathrm{O}(1)}{K}\right) * n$$

**Alternative:** with concat twice slower, density is 3/4, thus

$$\left(1.33 + \frac{\mathrm{O}(1)}{K}\right) * n$$

**Alternative:** for bag semantics (unordered items), usage is

$$\left(1 + \frac{\mathrm{O}(1)}{K}\right) * n$$

# Implementation and benchmarks

**Two implementations:**

→ OCaml (mechanized proof using CFML, in Coq)

→ C++ (carefully optimized code)

**Performance of C++ code:**

→ first layer with unweighted chunks of capacity 512

→ deeper layers with weighted chunks of capacity 32

| Experiment | STL deque | Bootstr. chunked |
|---|---|---|
| LIFO ($10^6 * 10^3$) | 5.46 | +28% |
| LIFO ($10^3 * 10^6$) | 9.15 | +20% |
| LIFO ($10^0 * 10^9$) | 12.07 | +12% |
| FIFO ($10^6 * 10^3$) | 5.51 | +16% |
| FIFO ($10^3 * 10^6$) | 9.16 | +15% |
| FIFO ($10^0 * 10^9$) | 12.32 | +8% |

Thanks!