SliceDeque

A double-ended queue with contiguous elements

About this talk

- My name is Gonzalo (**@gnzlbg** on Github)
- **Disclaimer:** I work **@nvidia**, these thoughts are my own, and do not necessarily reflect the views of my employer.
- I wrote a somewhat popular crate called “slice_deque”
  - ~200k all-time downloads
  - Network I/O applications, e.g., Firefox’s QUIC (neqo)
  - Audio applications, e.g., minimp3
- The idea behind it is not new, but it is an idea worth sharing: https://fgiesen.wordpress.com/2012/07/21/the-magic-ring-buffer/ (and others)

std::VecDeque recap

```rust
let mut deq = VecDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```

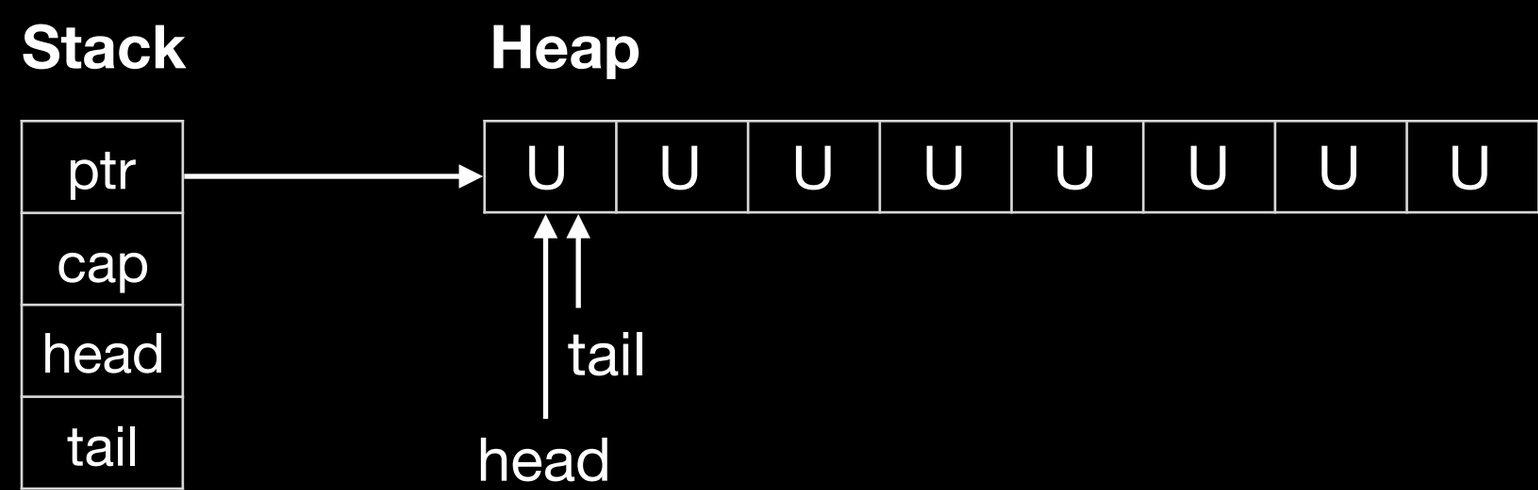
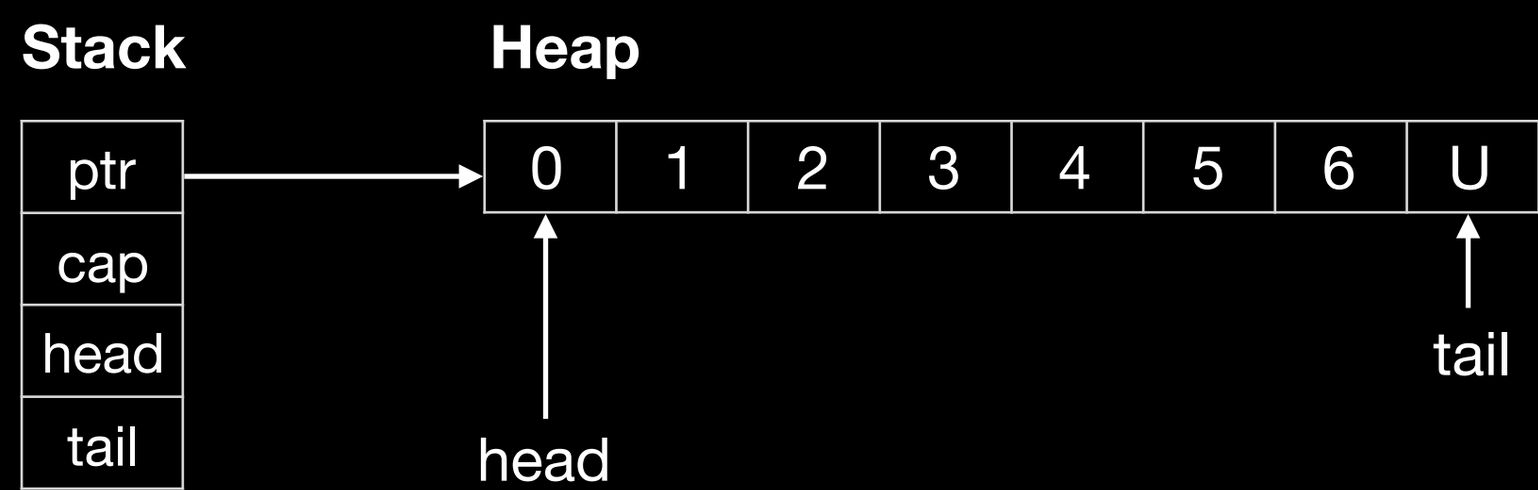
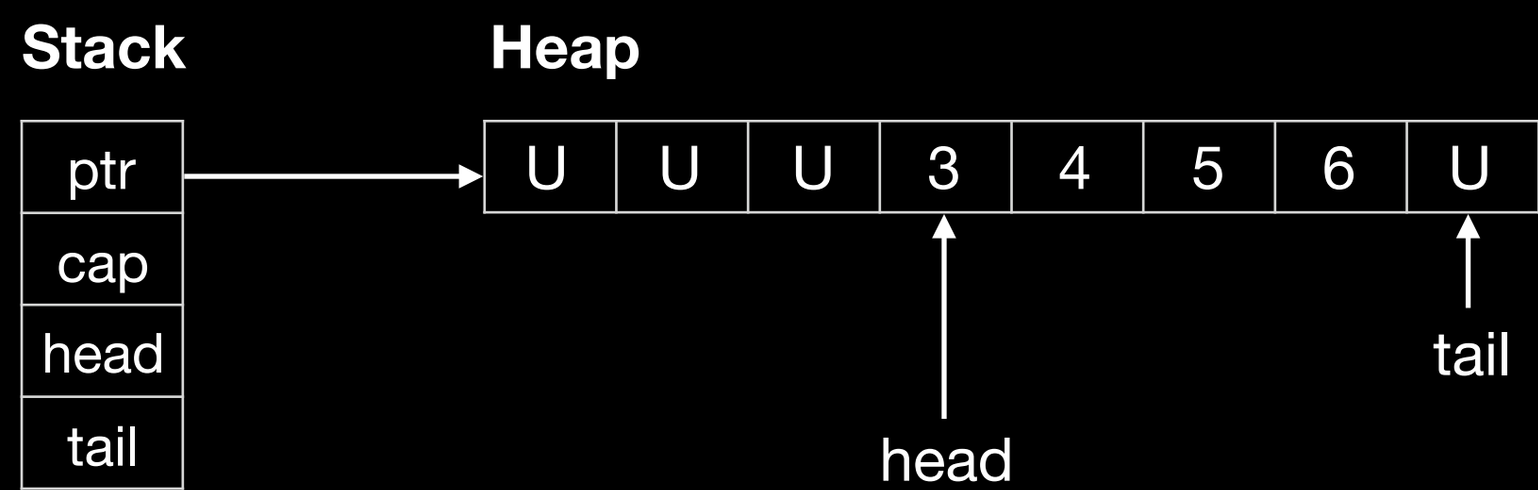
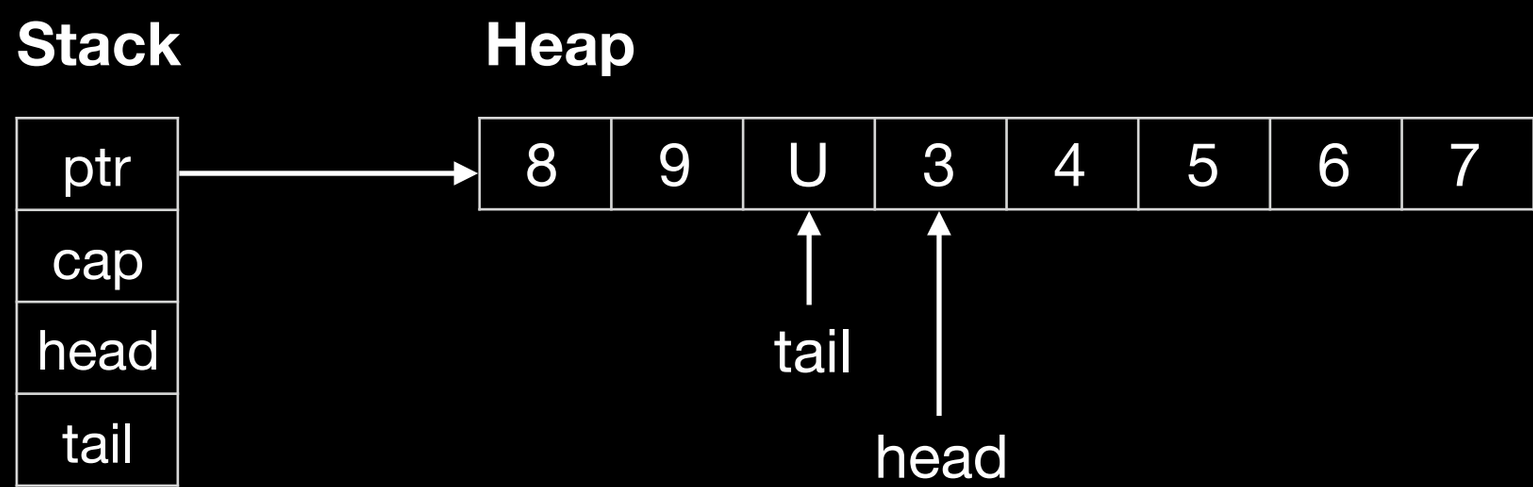
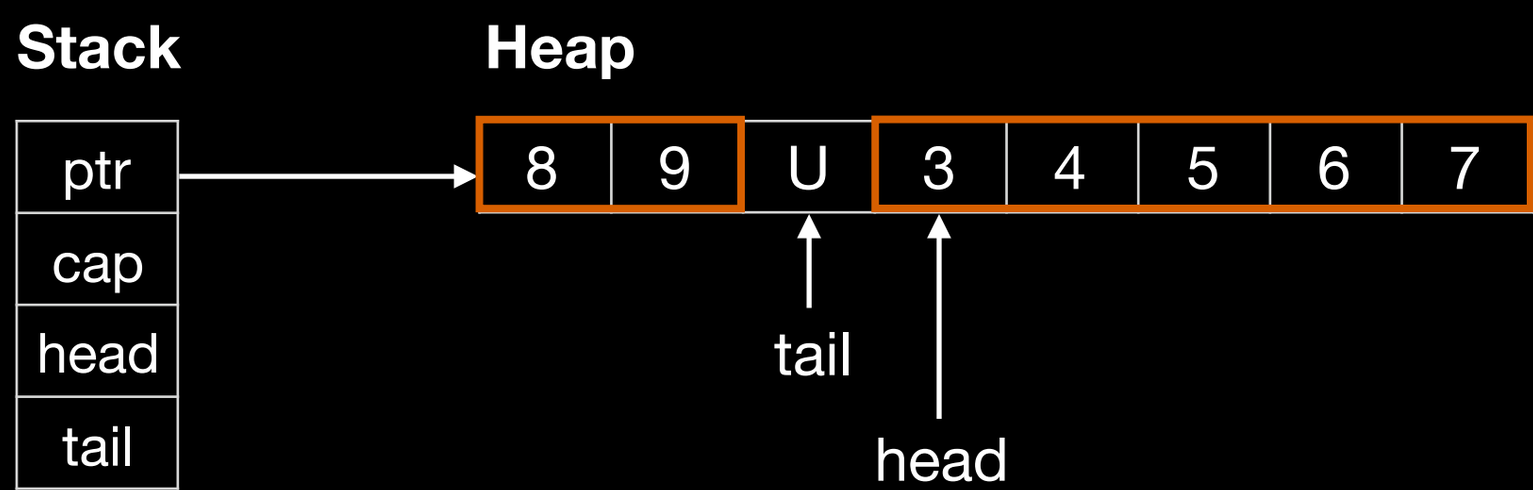
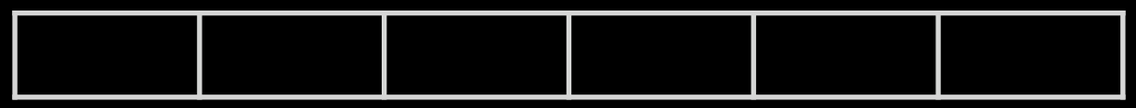
O(1) push_front/back
O(1) pop_front/back

std::VecDeque recap

- ```
  let mut deq = VecDeque::new();
  deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
  for _ in 0..3 { deq.pop_front(); }
  deq.extend(&[7, 8, 9]);
  ```

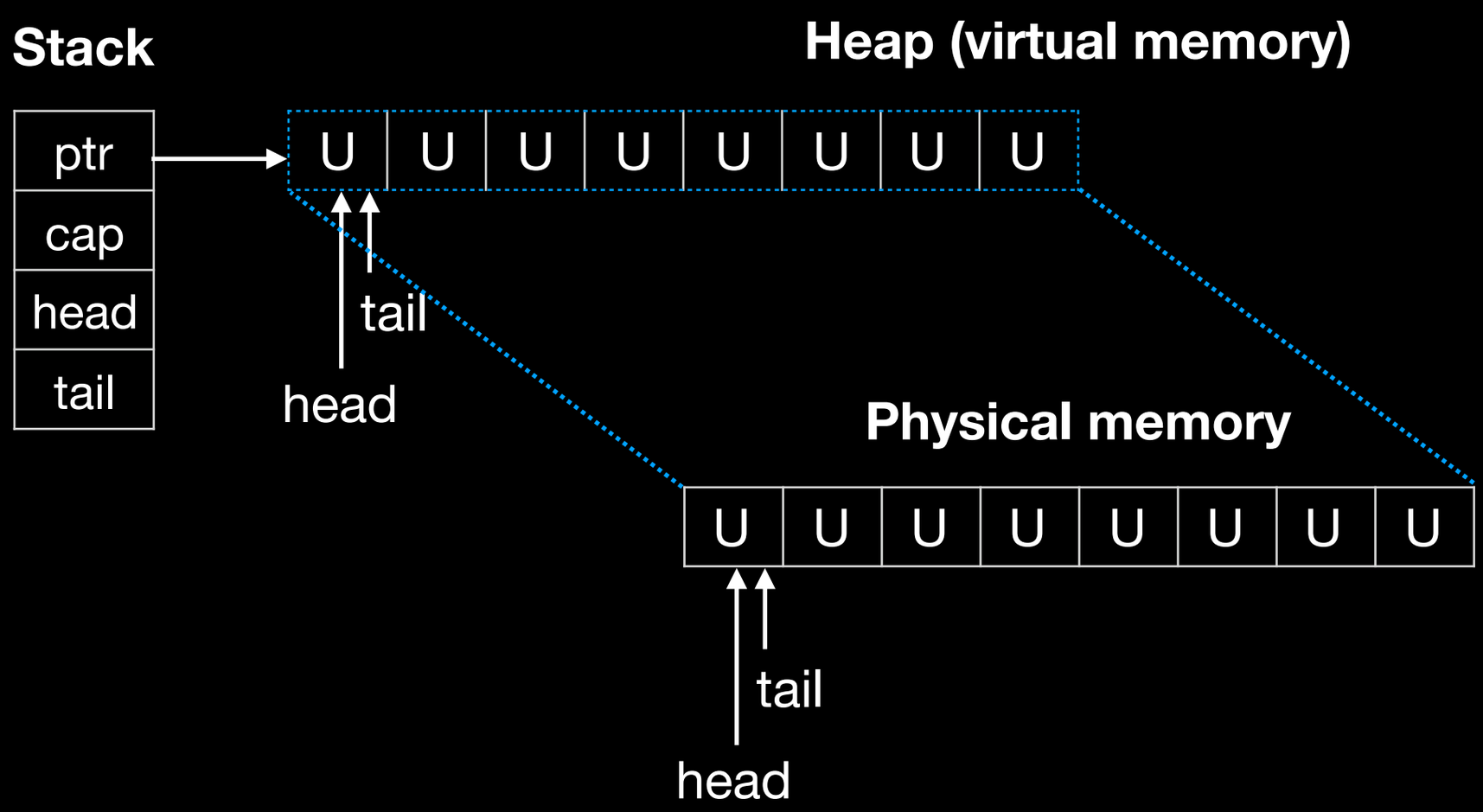
  O(1) push_front/back
  O(1) pop_front/back

Stack

ptr
cap
head
tail
  ```

std::VecDeque recap

```
let mut deq = VecDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```

O(1) push_front/back
O(1) pop_front/back



std::VecDeque recap

```
let mut deq = VecDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```

O(1) push_front/back
O(1) pop_front/back



std::VecDeque recap

```
let mut deq = VecDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```

O(1) push_front/back
O(1) pop_front/back



std::VecDeque recap

```
let mut deq = VecDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```

O(1) push_front/back
O(1) pop_front/back



std::VecDeque recap

```
let mut deq = VecDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```

O(1) push_front/back
O(1) pop_front/back



- Empty if *head* == *tail* ⇒ $sz_{max} = cap - 1$
- Efficiency: accessing the *i*-th element is non-trivial (powers-of-two cap), hierarchical iterators
- API, VecDeque::as_slices(&self) -> (&[T], &[T]), requires some “duck tape” to interface with
  - This usually means that if you are processing bytes from the network via the Bytes trait, you need to batch the processing around the end of the deque
- **Can we do better?**

Virtual memory 101

Virtual memory 101

- OS gives each process its own private “virtual” address space, isolating it from all other processes (protection), and abstracting how “virtual” addresses are backed by “physical” memory.
- **Security**: CPU traps if process tries to access memory w/o appropriate rights
- **Efficiency:** OS can change how virtual memory is backed (RAM, NVM, swap)
- MMUs, MPUs, etc. provide hardware support for virtual memory & protection

Virtual memory 101

- OS gives each process its own private “virtual” address space, isolating it from all other processes (protection), and abstracting how “virtual” addresses are backed by “physical” memory.
- **Security**: CPU traps if process tries to access memory w/o appropriate rights
- **Efficiency:** OS can change how virtual memory is backed (RAM, NVM, swap)
- MMUs, MPUs, etc. provide hardware support for virtual memory & protection

Physical memory



Virtual memory 101

- OS gives each process its own private “virtual” address space, isolating it from all other processes (protection), and abstracting how “virtual” addresses are backed by “physical” memory.
- **Security**: CPU traps if process tries to access memory w/o appropriate rights
- **Efficiency:** OS can change how virtual memory is backed (RAM, NVM, swap)
- MMUs, MPUs, etc. provide hardware support for virtual memory & protection

Physical memory

Process A

allocate 4 linked list nodes

VM

Virtual memory 101

- OS gives each process its own private “virtual” address space, isolating it from all other processes (protection), and abstracting how “virtual” addresses are backed by “physical” memory.
- **Security**: CPU traps if process tries to access memory w/o appropriate rights
- **Efficiency:** OS can change how virtual memory is backed (RAM, NVM, swap)
- MMUs, MPUs, etc. provide hardware support for virtual memory & protection

Physical memory

Process A

allocate 4 linked list nodes
delete one node

VM

Virtual memory 101

- OS gives each process its own private “virtual” address space, isolating it from all other processes (protection), and abstracting how “virtual” addresses are backed by “physical” memory.
- **Security**: CPU traps if process tries to access memory w/o appropriate rights
- **Efficiency:** OS can change how virtual memory is backed (RAM, NVM, swap)
- MMUs, MPUs, etc. provide hardware support for virtual memory & protection

Physical memory

Process A

allocate 4 linked list nodes
delete one node

VM

Process B

allocate a contiguous 3 node array

VM

SliceDeque

```
let mut deq = SliceDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```

SliceDeque

- ```
  let mut deq = SliceDeque::new();
  deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
  for _ in 0..3 { deq.pop_front(); }
  deq.extend(&[7, 8, 9]);
  ```

  Stack

ptr
cap
head
tail
  ```

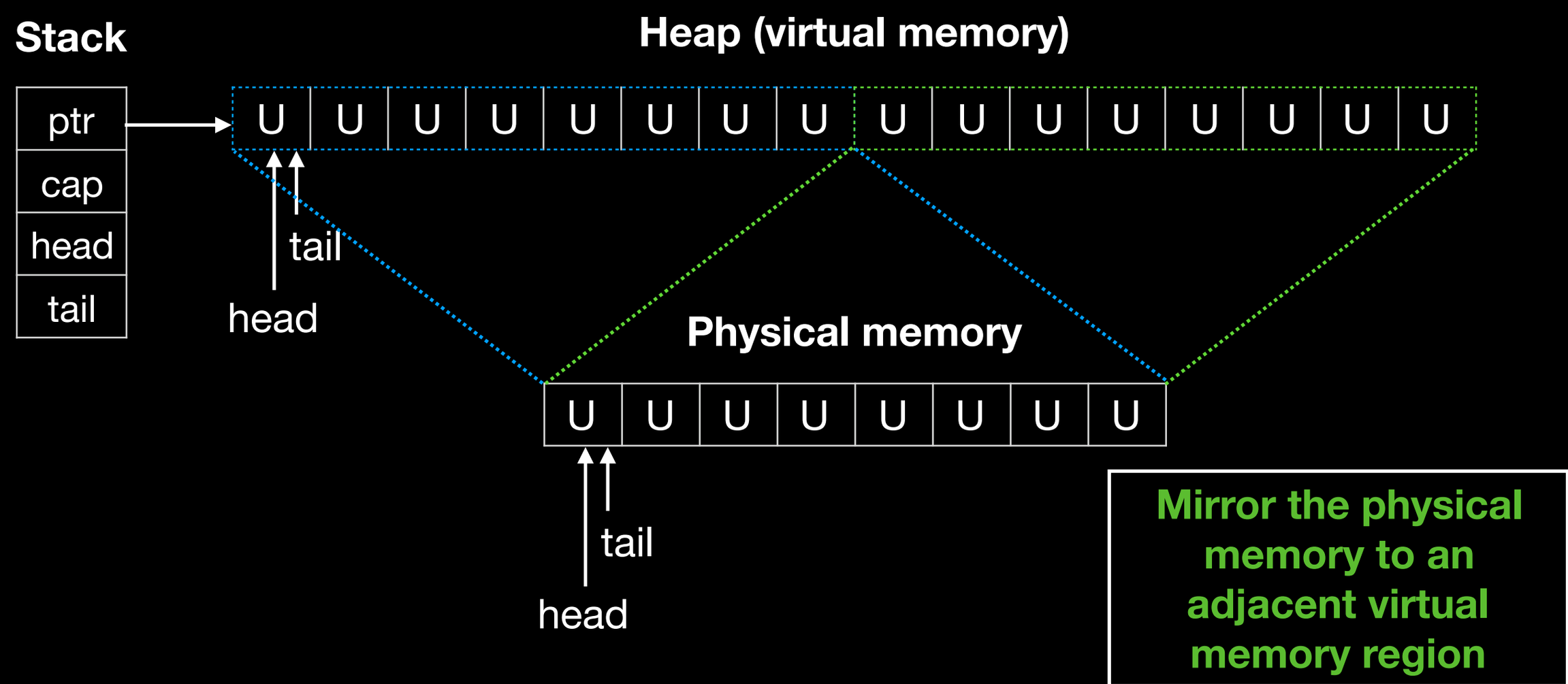
SliceDeque

```
let mut deq = SliceDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```



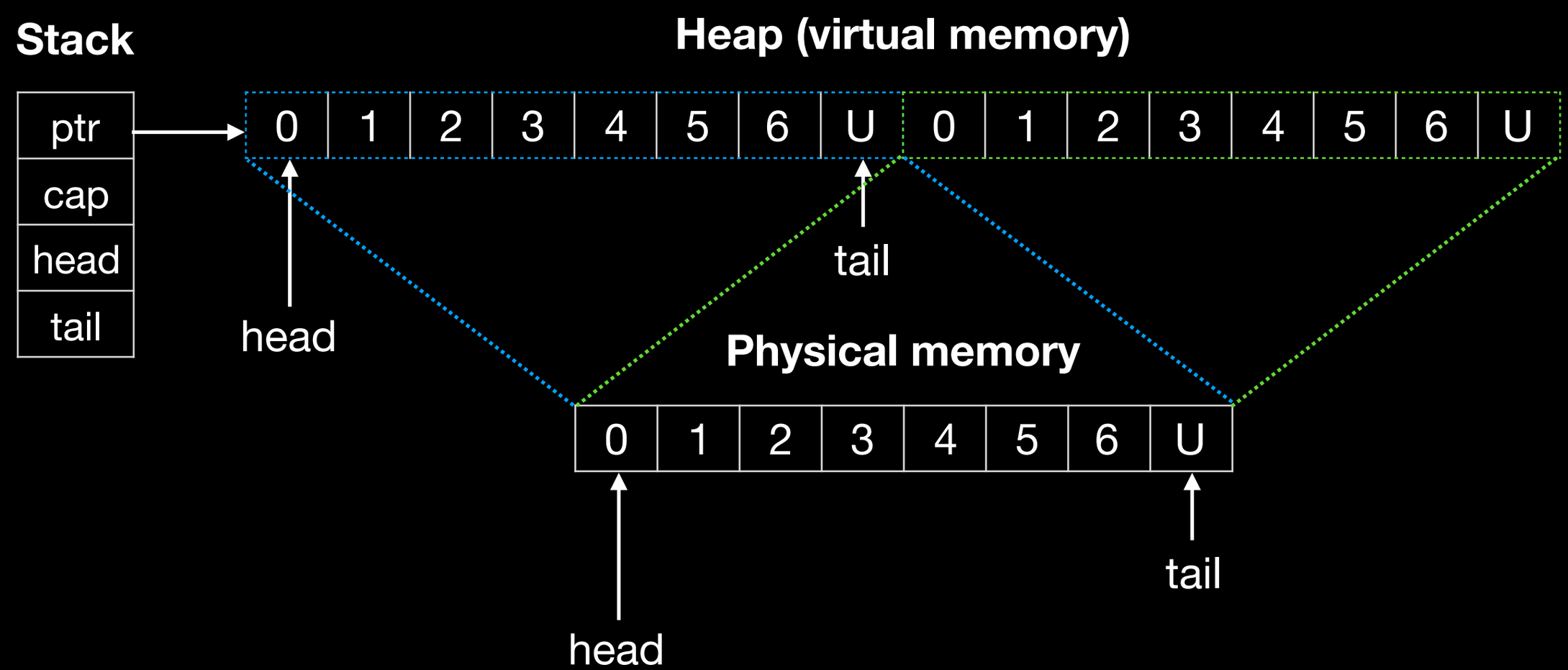
SliceDeque

```
let mut deq = SliceDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```



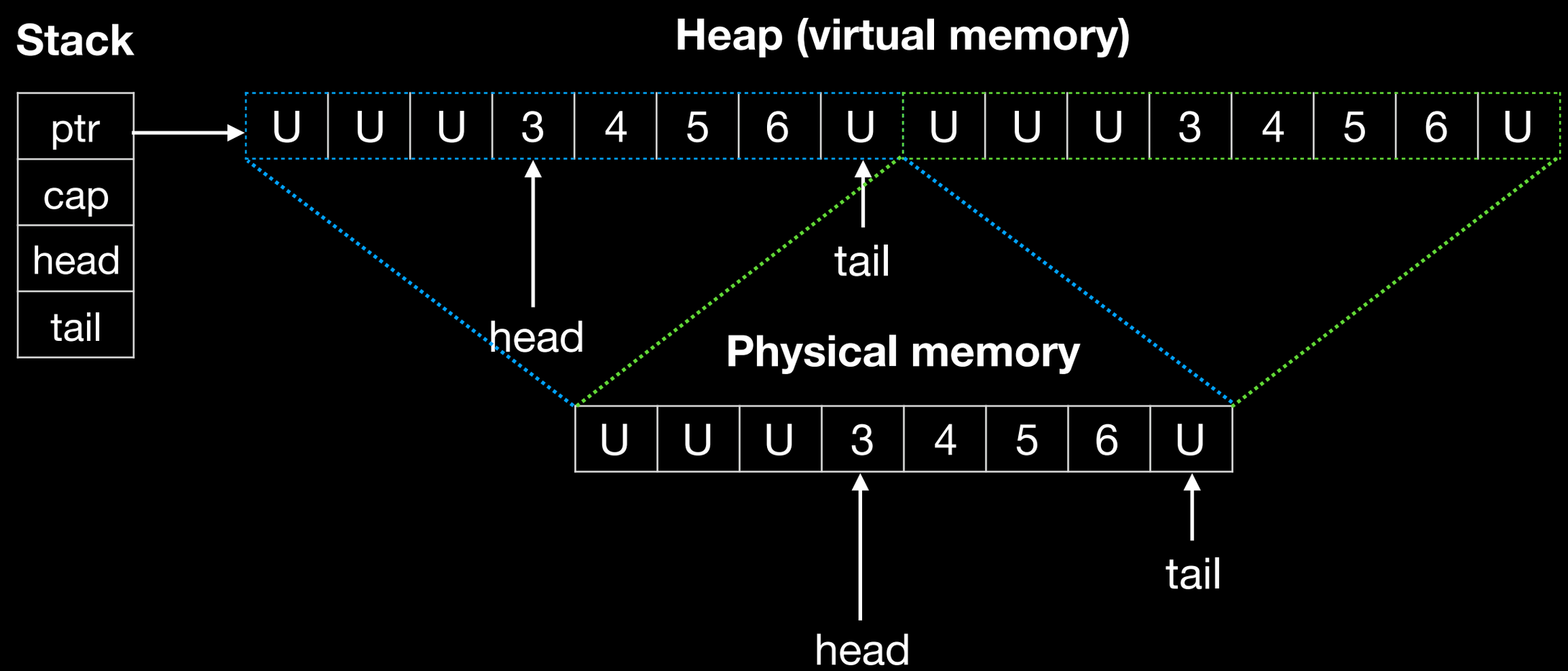
SliceDeque

```
let mut deq = SliceDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```



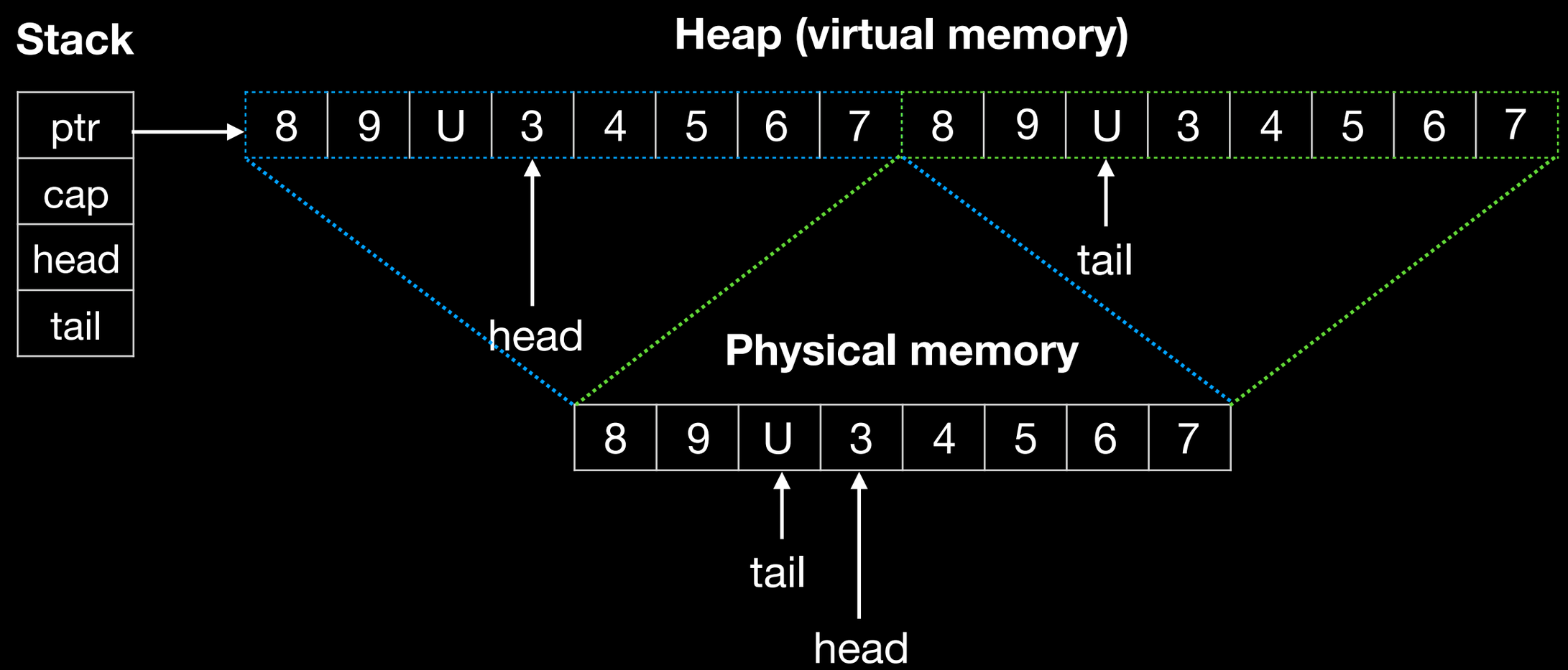
SliceDeque

```
let mut deq = SliceDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```



SliceDeque

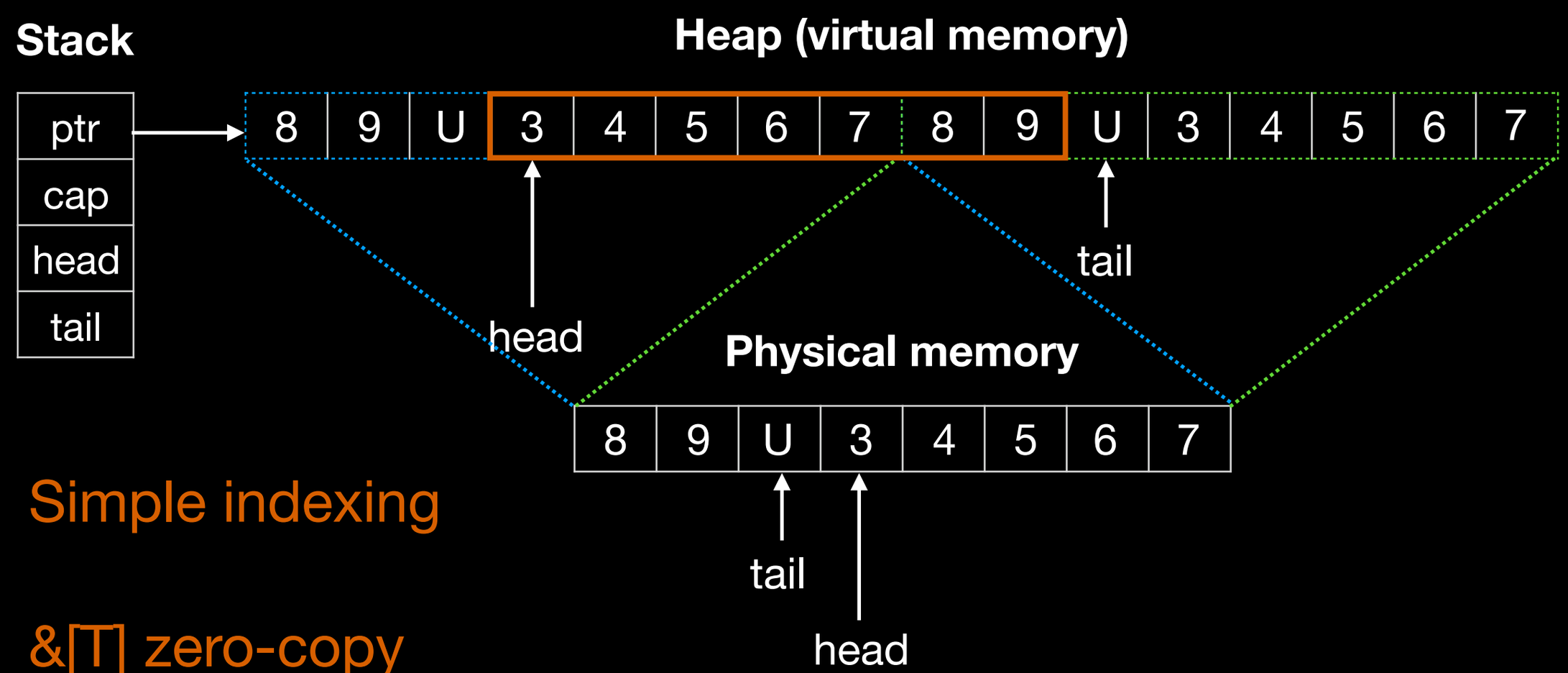
```
let mut deq = SliceDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```



SliceDeque

```
let mut deq = SliceDeque::new();
deq.extend(&[0, 1, 2, 3, 4, 5, 6]);
for _ in 0..3 { deq.pop_front(); }
deq.extend(&[7, 8, 9]);
```

In virtual memory, the deque elements are always contiguous!



- Simple indexing
- &[T] zero-copy

Pros / Cons

- **Widely supported:** SliceDeque supports Linux, MacOSX, Windows, iOS, Android, FreeBSD, OpenBSD, Solaris, NetBSD, … It is trivial to port it to any POSIX compliant OS.

- **Simple API:** interface directly with all APIs using **&[T]** (sorting, binary search, etc.).

- **Efficient:** simple indexing and iterators, vectorization, bulk processing, etc.

- **Requires platform support:** the “OS” (and hardware) must support virtual memory. QoI is platform dependent: great on MacOSX, ok-ish on Linux >= 3.17, less ok-ish everywhere else: resources, race-conditions; improvements in **#10**.

- **Bypasses the global memory allocator:** requiring multiple syscalls on growth and rellocation, etc. There is a PR open that improves it.

- **Constrained minimum capacity:** the memory usage of SliceDeque is often constrained to multiples of the allocation granularity of the OS 4/8/64 kB, since the virtual to physical memory map works on memory pages.

When does it make sense?

If the problem requires:

- a fixed-number of deques,
- the deques are long-lived,
- each deque is fixed-size,
- their size is large enough (at least 1 memory page),
- the target supports it,
- the resources required by the target are acceptable,

then using a SliceDeque ***might*** make sense.

Examples: I/O buffers for: audio-channels, executors, etc.

That’s all Folks!

- **Idea:** explicit virtual memory use enables “impossible” data-structures
  others: non-reallocating vectors, concurrent lock-free deques, etc.

- The details are at: https://github.com/gnzlbg/slice_deque

- Great Rust-specific resources:

  - **Writing an OS in Rust** (x64) by Philipp Opperman: *Introduction to Paging* (https://os.phil-opp.com/paging-introduction/) and *Paging implementation* (https://os.phil-opp.com/paging-implementation/)

  - **RISC-V OS using Rust** by Stephen Marz: *Page-grained memory allocation* (http://osblog.stephenmarz.com/ch3.html) and *Memory-management unit* (http://osblog.stephenmarz.com/ch3.2.html)

- Thank you for your attention and I wish you a happy hacking!