usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

ARTIFACT EVALUATED
usenix ASSOCIATION
FUNCTIONAL

ARTIFACT EVALUATED
usenix ASSOCIATION
REPRODUCED

BBQ: A Block-based Bounded Queue for Exchanging Data and Profiling

Jiawei Wang, **Diogo Behrens**, Ming Fu, Lilith Oberhauser,
Jonas Oberhauser, Jitang Lei, Geng Chen, Hermann Härtig, Haibo Chen

HUAWEI

TECHNISCHE UNIVERSITÄT DRESDEN

SHANGHAI JIAO TONG UNIVERSITY

Bounded queues (aka ring buffers) are everywhere...

Netty
LMAX™ EXCHANGE
GO
openGauss
MariaDB
PostgreSQL
R
Linux™
SPDK
DPDK
DATA PLANE DEVELOPMENT KIT
I/O URING
OpenEuler

Why are they important to us?

Crucial for the
performance and correctness
of systems and applications!

Why are they important to us?

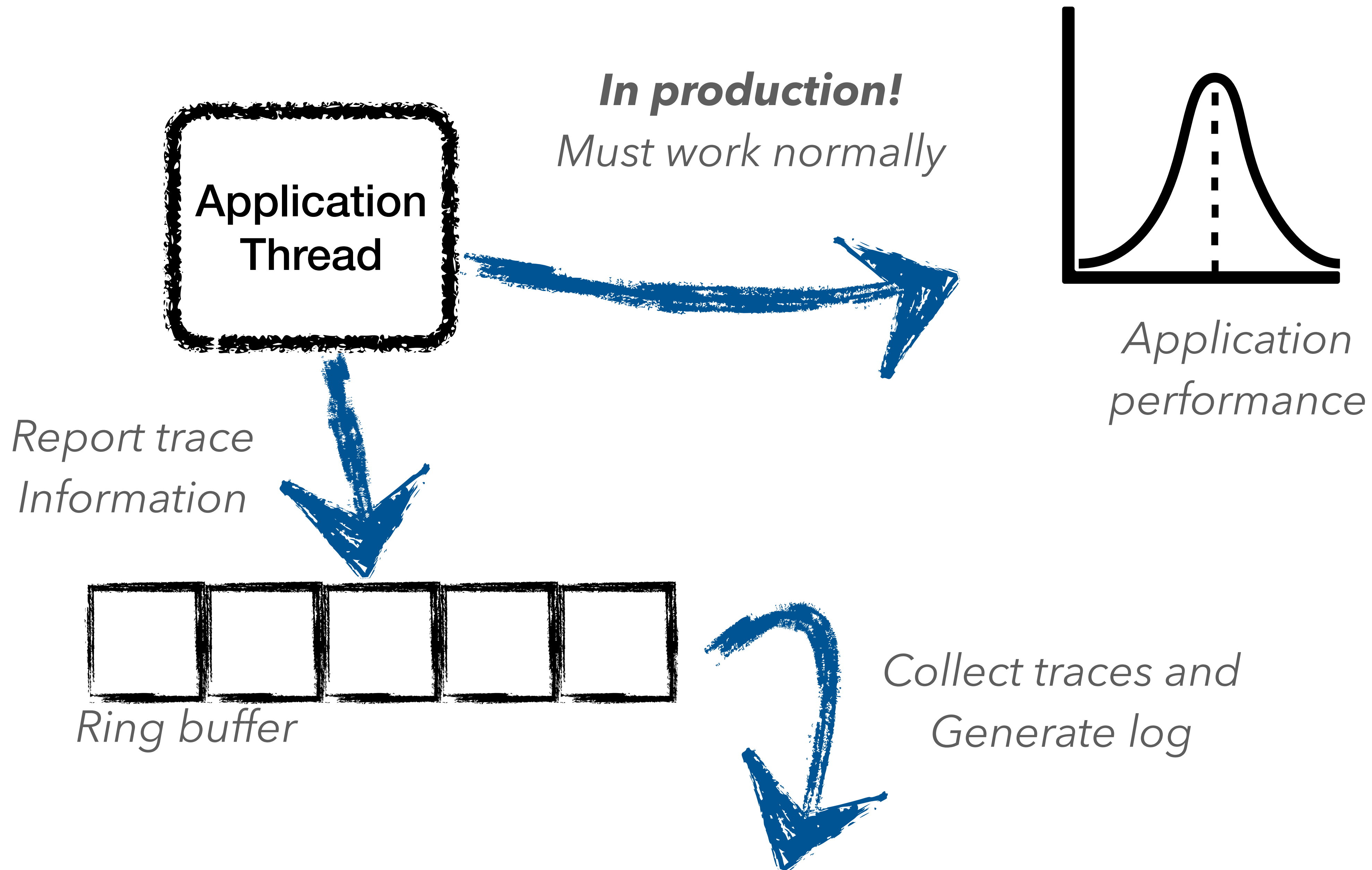
Crucial for the
performance and correctness
of systems and applications!

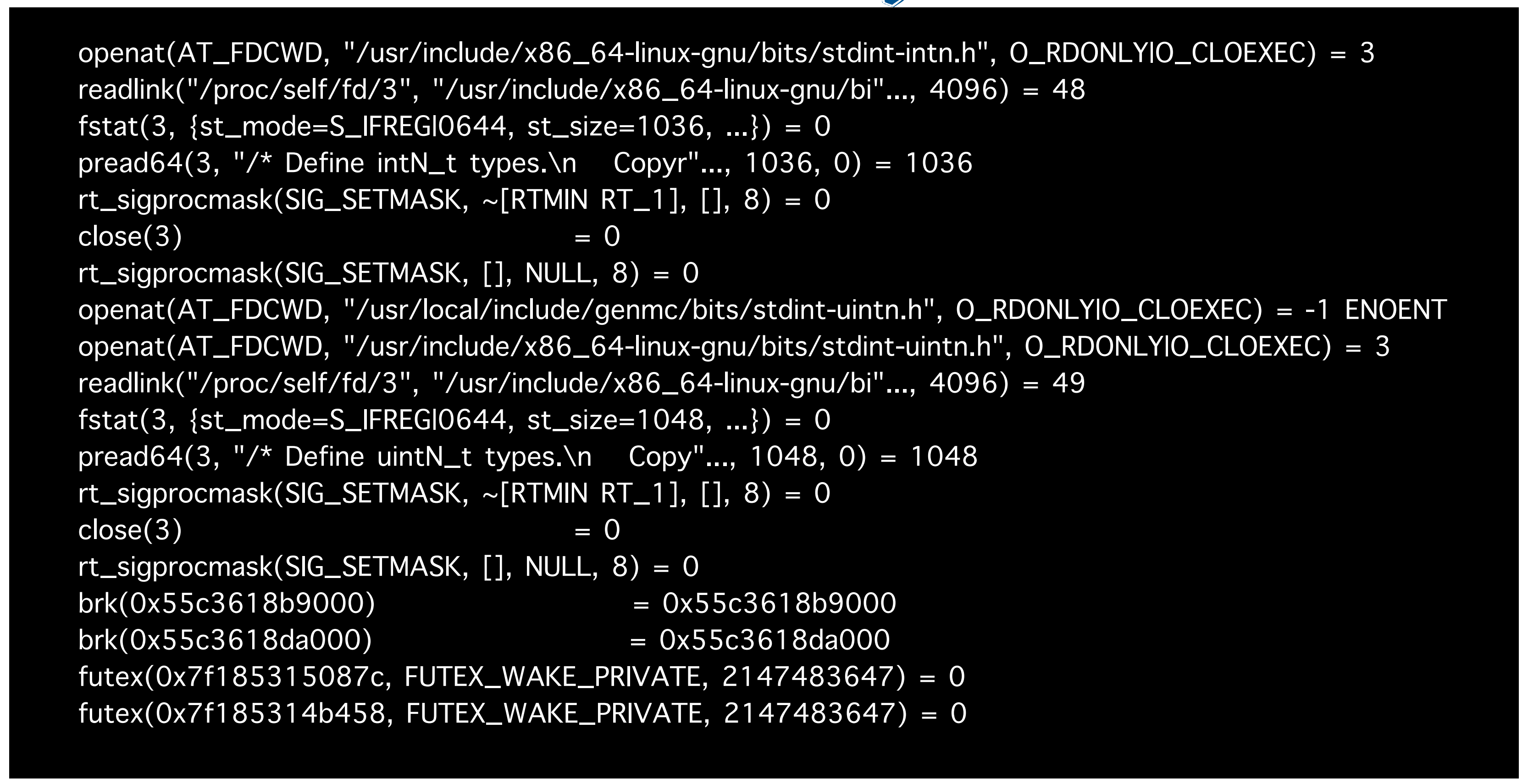
Next, **3 ring buffer stories**
from Huawei software development

Story 1: Tracing overhead and operation interference

In-house OS with a **new tracing tool**

- Ring buffer used to collect traces
- Used to generate application profile
- Reporting **must be fast!**





```
openat(AT_FDCWD, "/usr/include/x86_64-linux-gnu/bits/stdint-intn.h", O_RDONLY|O_CLOEXEC) = 3
readlink("/proc/self/fd/3", "/usr/include/x86_64-linux-gnu/bi"..., 4096) = 48
fstat(3, {st_mode=S_IFREG|0644, st_size=1036, ...}) = 0
pread64(3, "/* Define intN_t types.\n   Copyr"..., 1036, 0) = 1036
rt_sigprocmask(SIG_SETMASK, ~[RTMIN RT_1], [], 8) = 0
close(3)                                = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
openat(AT_FDCWD, "/usr/local/include/genmc/bits/stdint-uintn.h", O_RDONLY|O_CLOEXEC) = -1 ENOENT
openat(AT_FDCWD, "/usr/include/x86_64-linux-gnu/bits/stdint-uintn.h", O_RDONLY|O_CLOEXEC) = 3
readlink("/proc/self/fd/3", "/usr/include/x86_64-linux-gnu/bi"..., 4096) = 49
fstat(3, {st_mode=S_IFREG|0644, st_size=1048, ...}) = 0
pread64(3, "/* Define uintN_t types.\n   Copy"..., 1048, 0) = 1048
rt_sigprocmask(SIG_SETMASK, ~[RTMIN RT_1], [], 8) = 0
close(3)                                = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
brk(0x55c3618b9000)                     = 0x55c3618b9000
brk(0x55c3618da000)                     = 0x55c3618da000
futex(0x7f185315087c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
futex(0x7f185314b458, FUTEX_WAKE_PRIVATE, 2147483647) = 0
```

Trace output

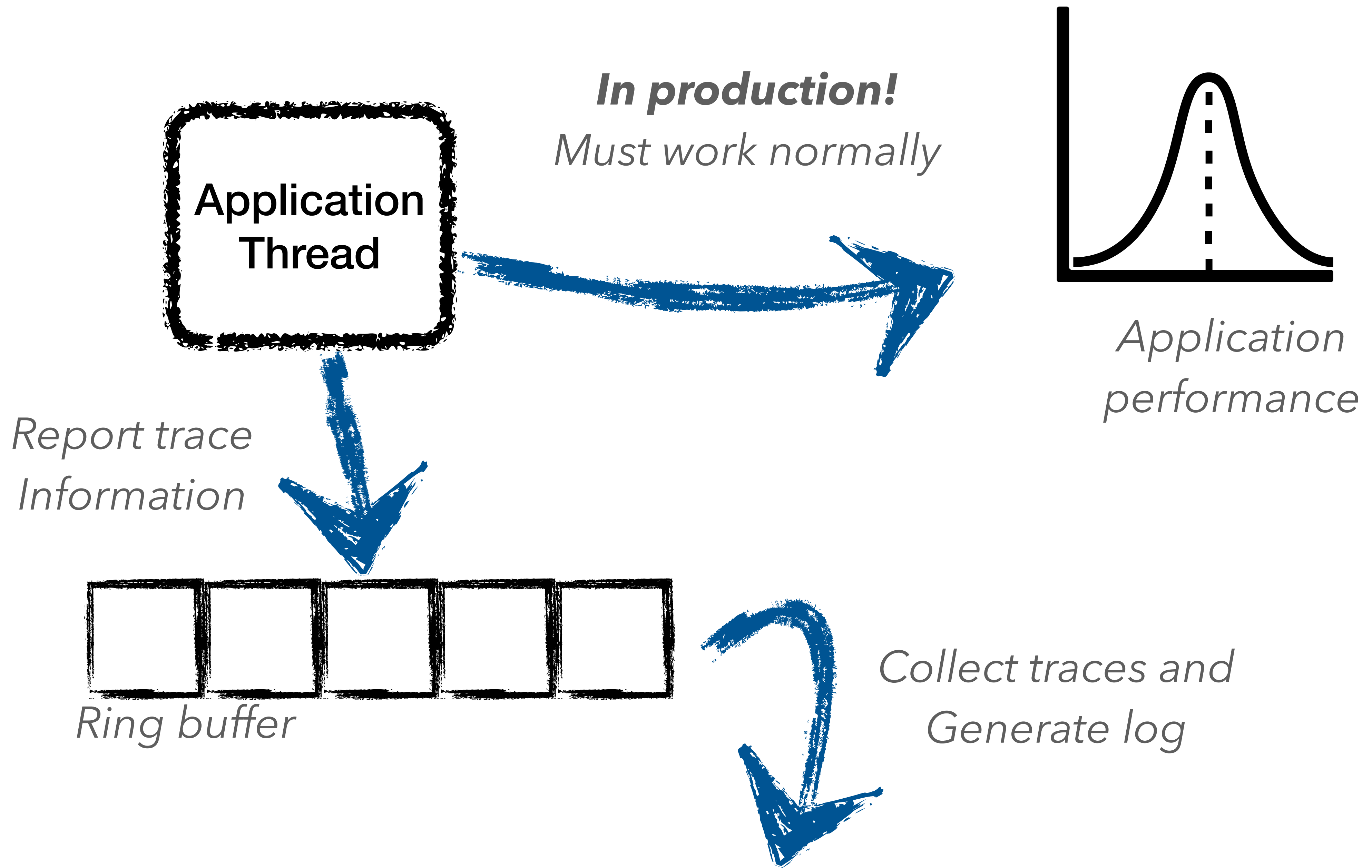
Story 1: Tracing overhead and operation interference

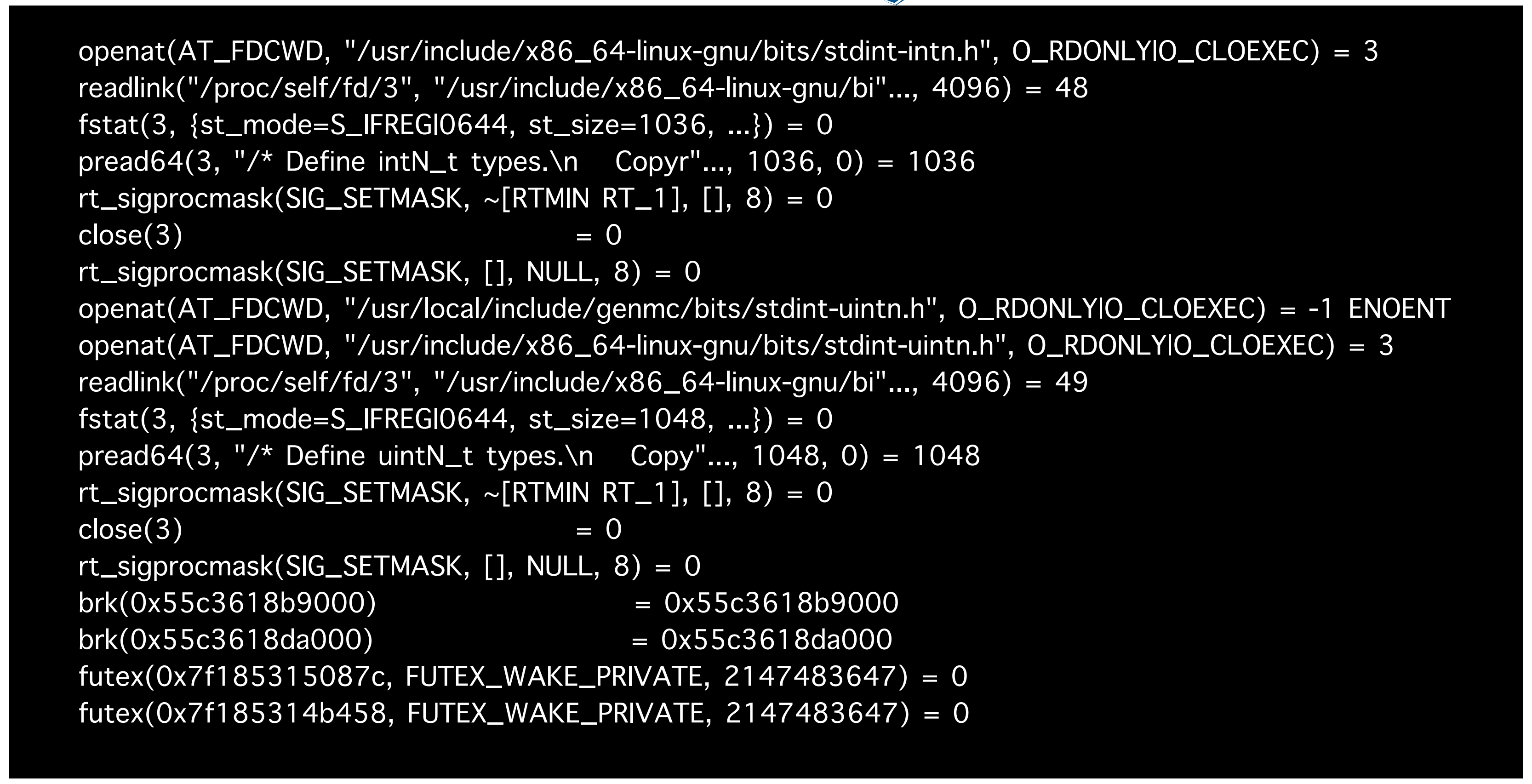
In-house OS with a **new tracing tool**

- Ring buffer used to collect traces
- Used to generate application profile
- Reporting **must be fast!**

Problem

- Consumer **slowdowns** producer!


Application Thread
In production!
Must work normally
Application performance
Report trace Information
Ring buffer
Collect traces and Generate log

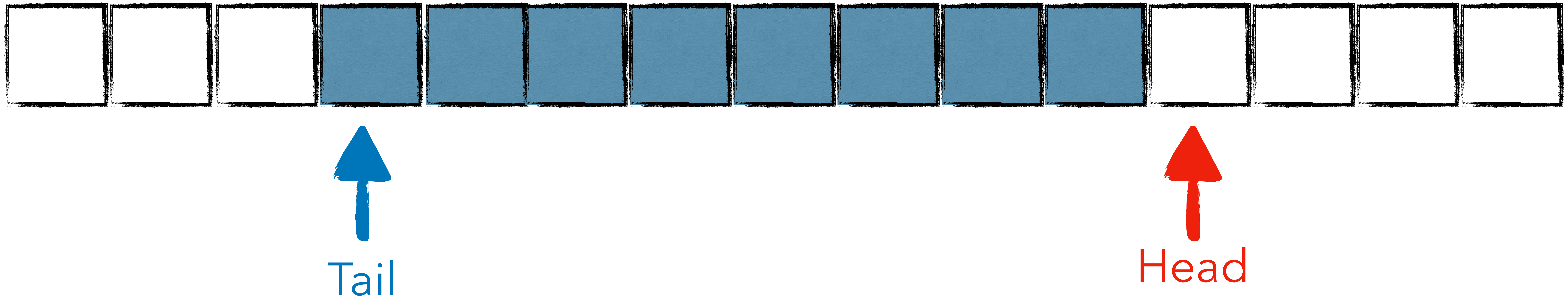


```
openat(AT_FDCWD, "/usr/include/x86_64-linux-gnu/bits/stdint-intn.h", O_RDONLY|O_CLOEXEC) = 3
readlink("/proc/self/fd/3", "/usr/include/x86_64-linux-gnu/bi"..., 4096) = 48
fstat(3, {st_mode=S_IFREG|0644, st_size=1036, ...}) = 0
pread64(3, "/* Define intN_t types.\n   Copyr"..., 1036, 0) = 1036
rt_sigprocmask(SIG_SETMASK, ~[RTMIN RT_1], [], 8) = 0
close(3)                                = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
openat(AT_FDCWD, "/usr/local/include/genmc/bits/stdint-uintn.h", O_RDONLY|O_CLOEXEC) = -1 ENOENT
openat(AT_FDCWD, "/usr/include/x86_64-linux-gnu/bits/stdint-uintn.h", O_RDONLY|O_CLOEXEC) = 3
readlink("/proc/self/fd/3", "/usr/include/x86_64-linux-gnu/bi"..., 4096) = 49
fstat(3, {st_mode=S_IFREG|0644, st_size=1048, ...}) = 0
pread64(3, "/* Define uintN_t types.\n   Copy"..., 1048, 0) = 1048
rt_sigprocmask(SIG_SETMASK, ~[RTMIN RT_1], [], 8) = 0
close(3)                                = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
brk(0x55c3618b9000)                     = 0x55c3618b9000
brk(0x55c3618da000)                     = 0x55c3618da000
futex(0x7f185315087c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
futex(0x7f185314b458, FUTEX_WAKE_PRIVATE, 2147483647) = 0
```

Trace output

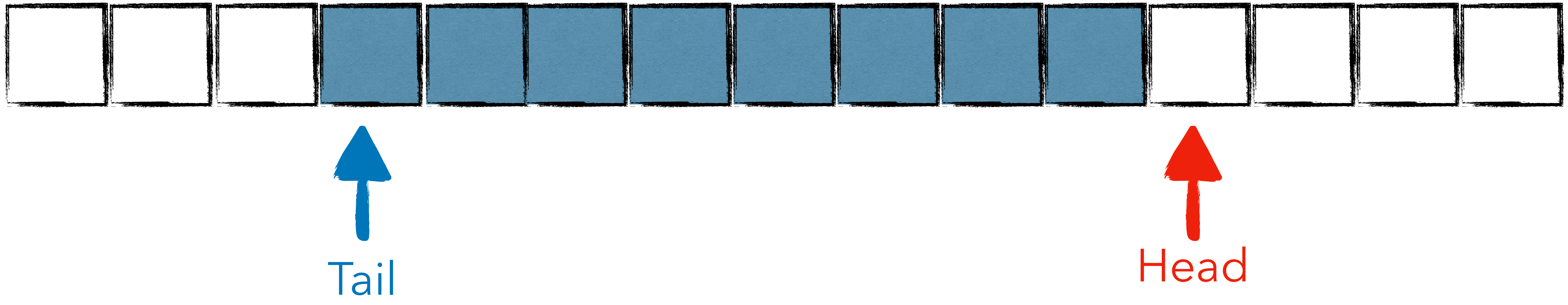
Interference sources

Ring buffers are arrays with indices



Interference sources

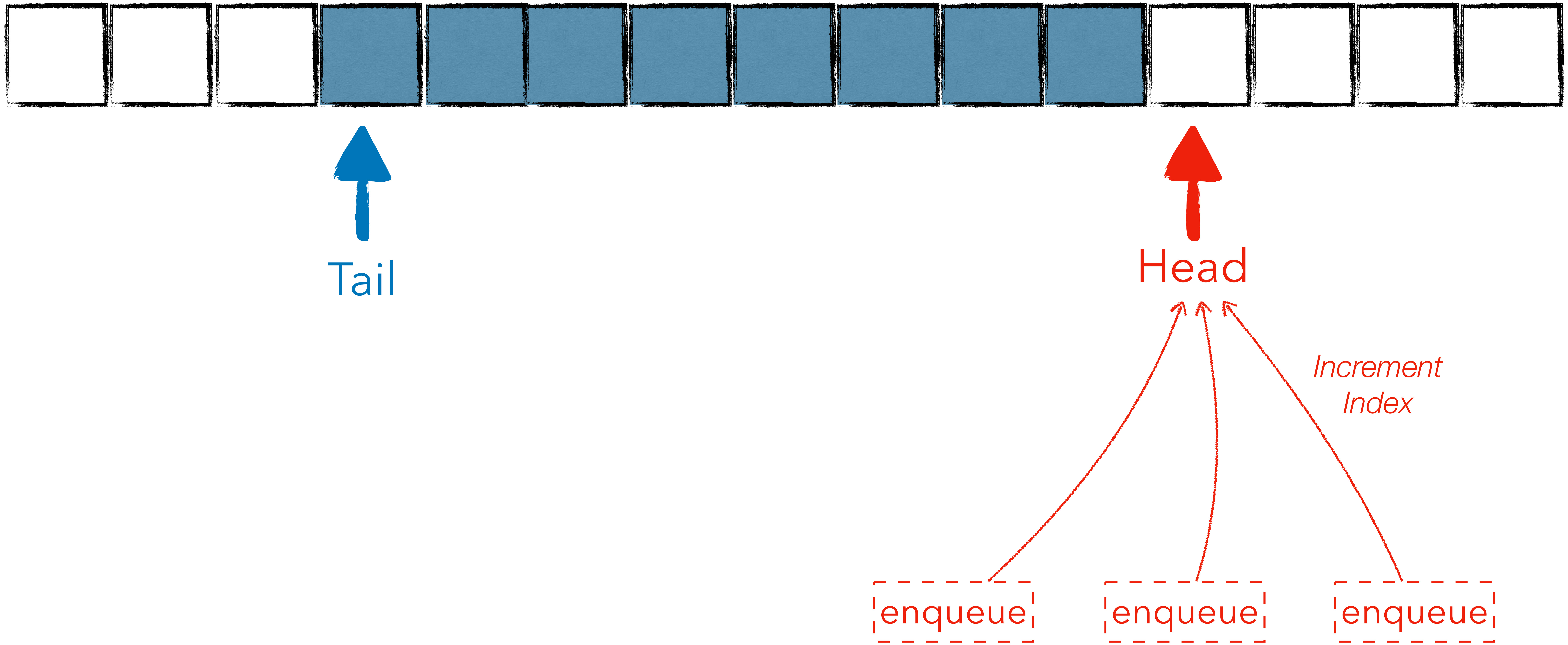
Ring buffers are arrays with indices



enqueue enqueue enqueue

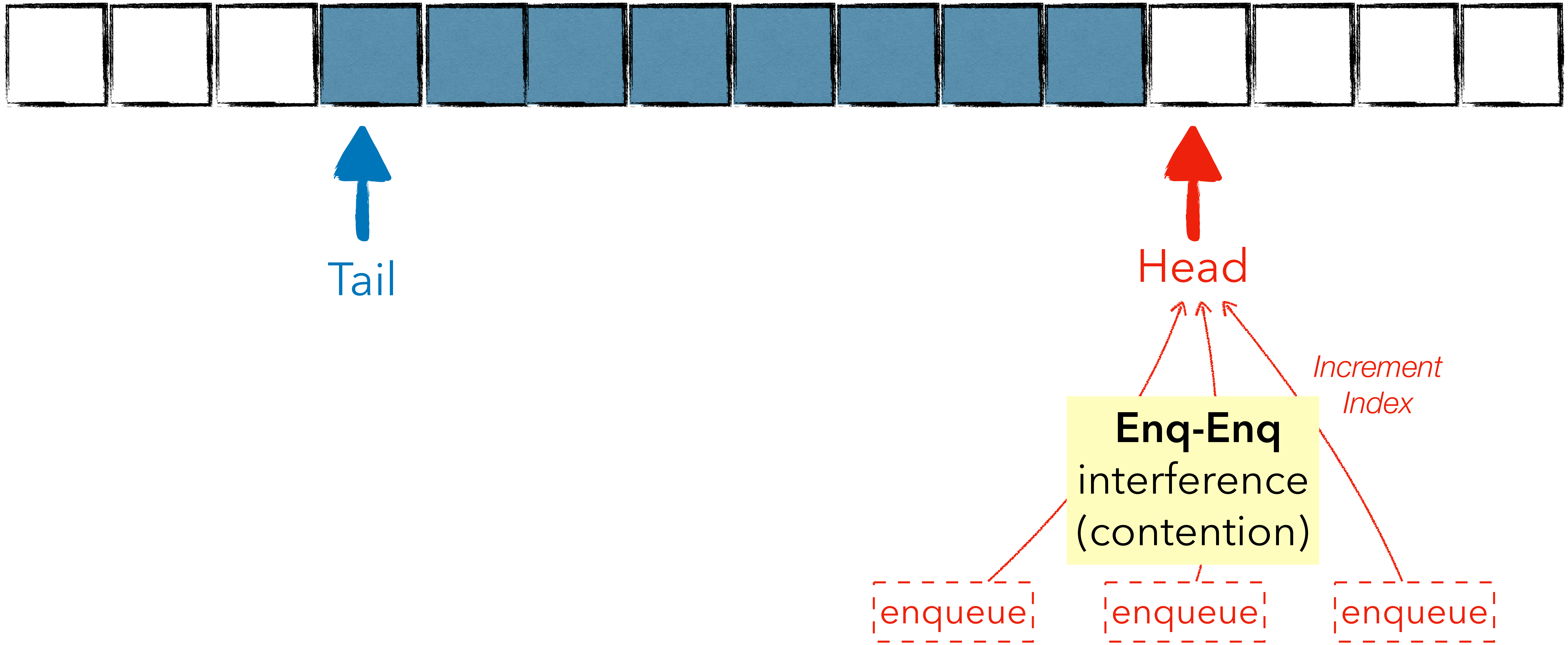
Interference sources

Ring buffers are arrays with indices



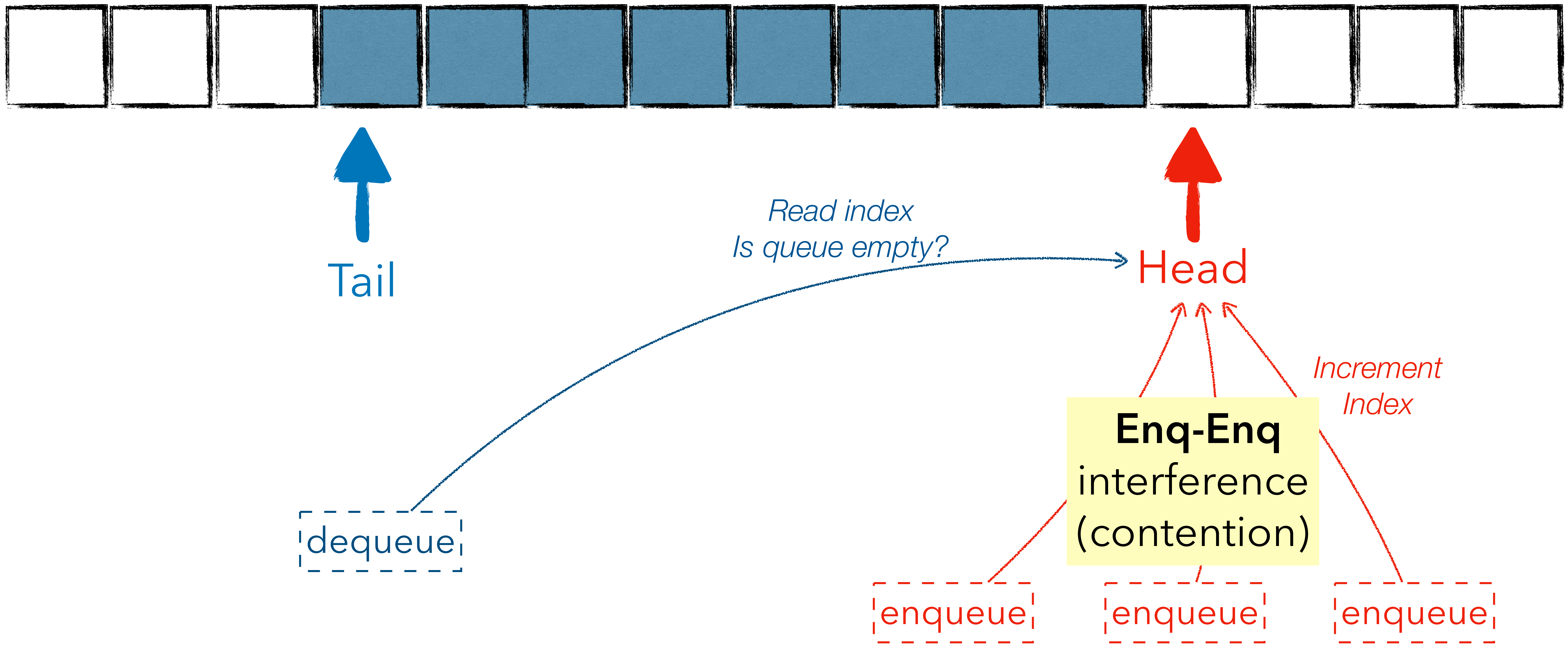
Interference sources

Ring buffers are arrays with indices



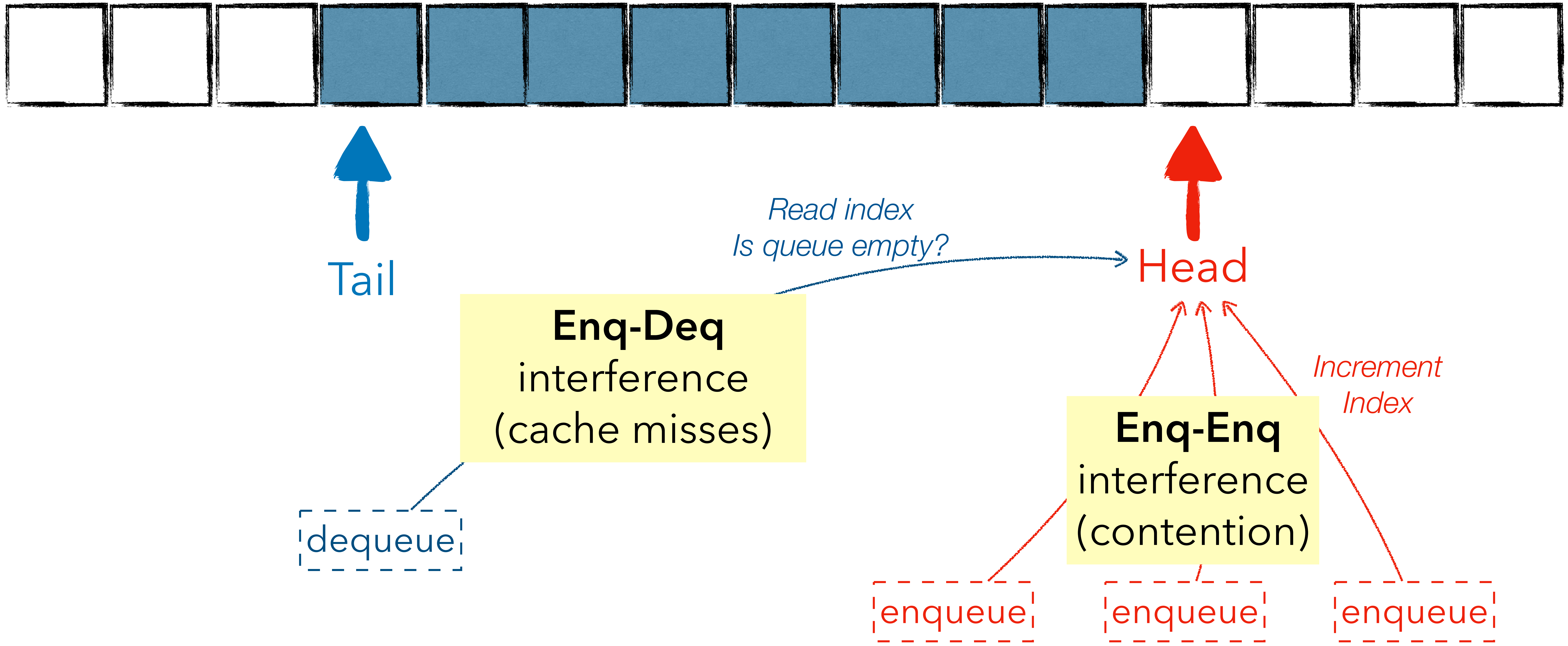
Interference sources

Ring buffers are arrays with indices



Interference sources

Ring buffers are arrays with indices



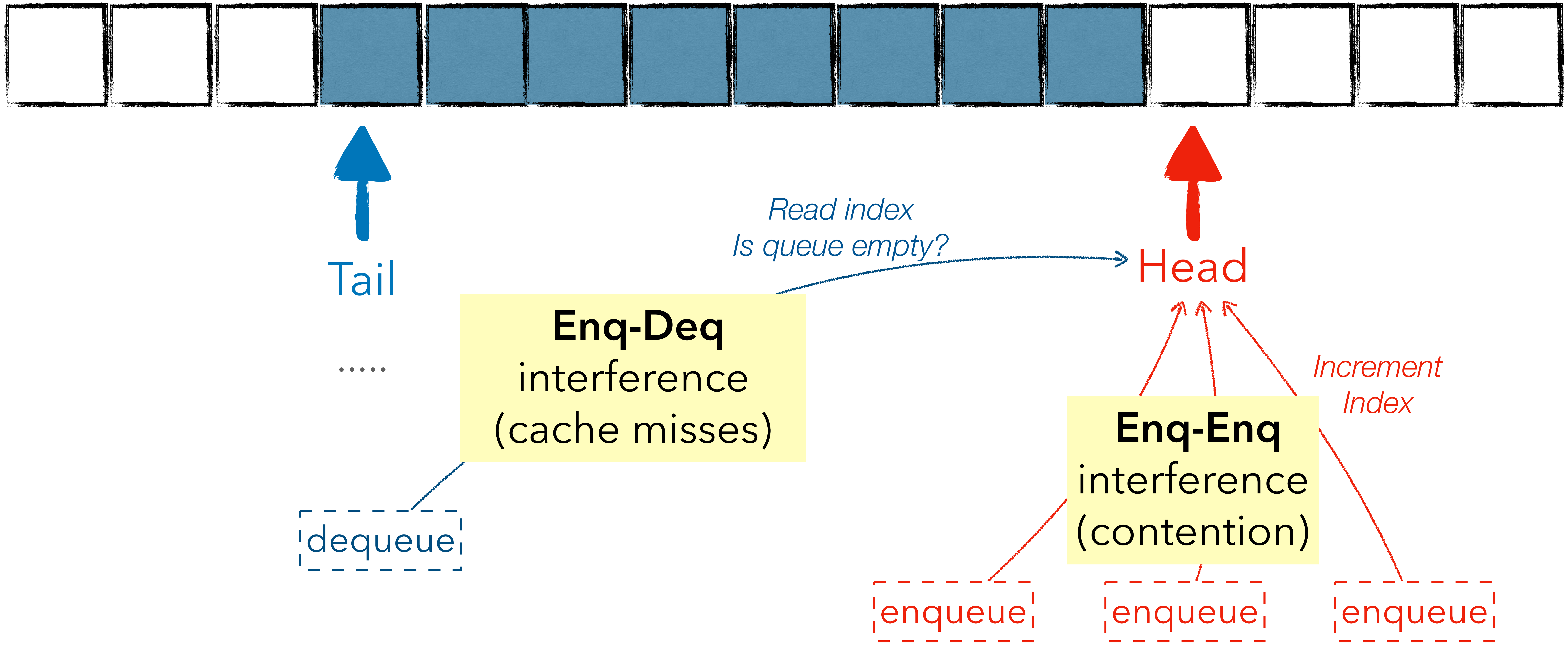
Read index
Is queue empty?

Enq-Deq
interference
(cache misses)

Enq-Enq
interference
(contention)

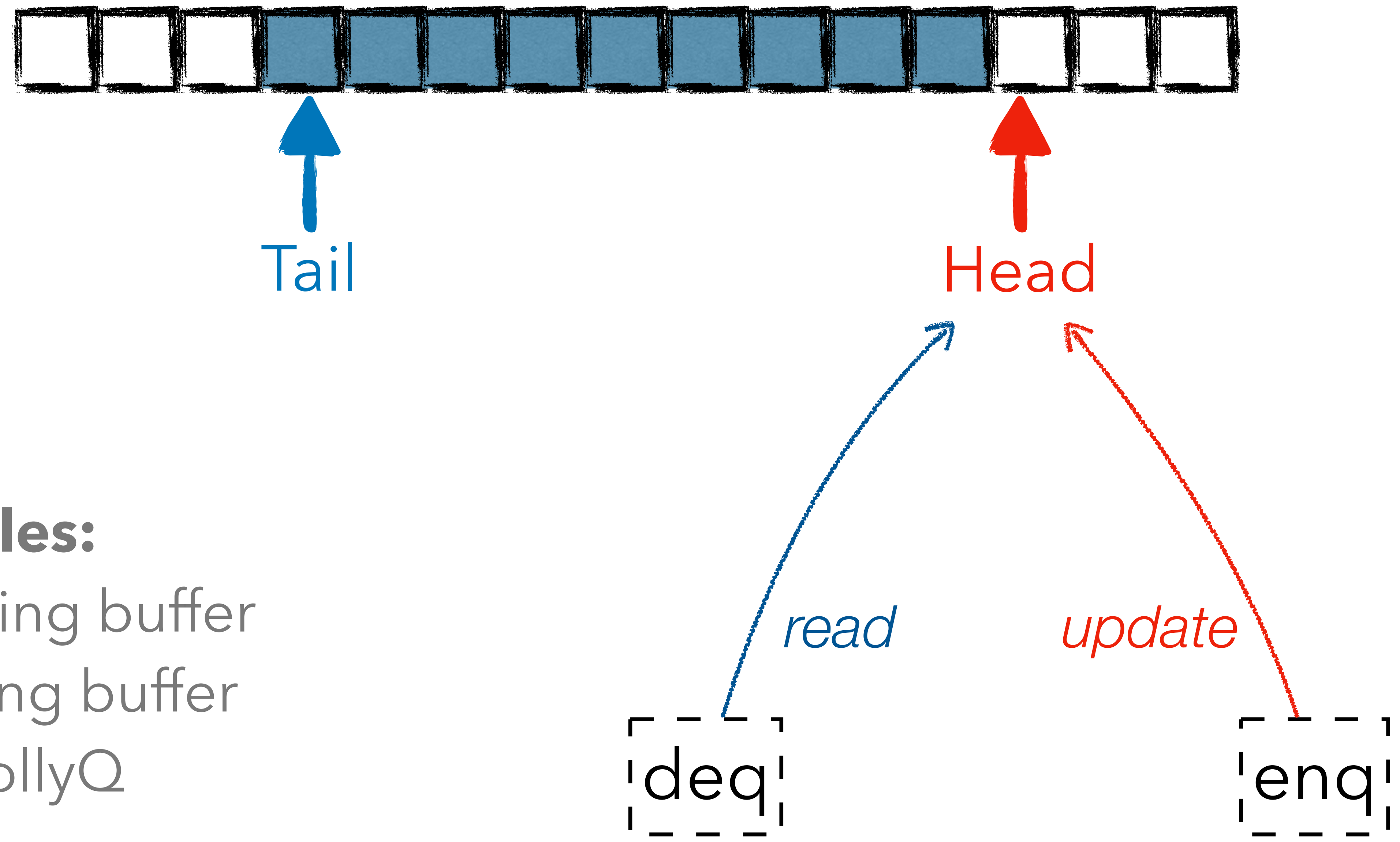
Interference sources

Ring buffers are arrays with indices



Interference sources – Existing work

Enq-Deq interference

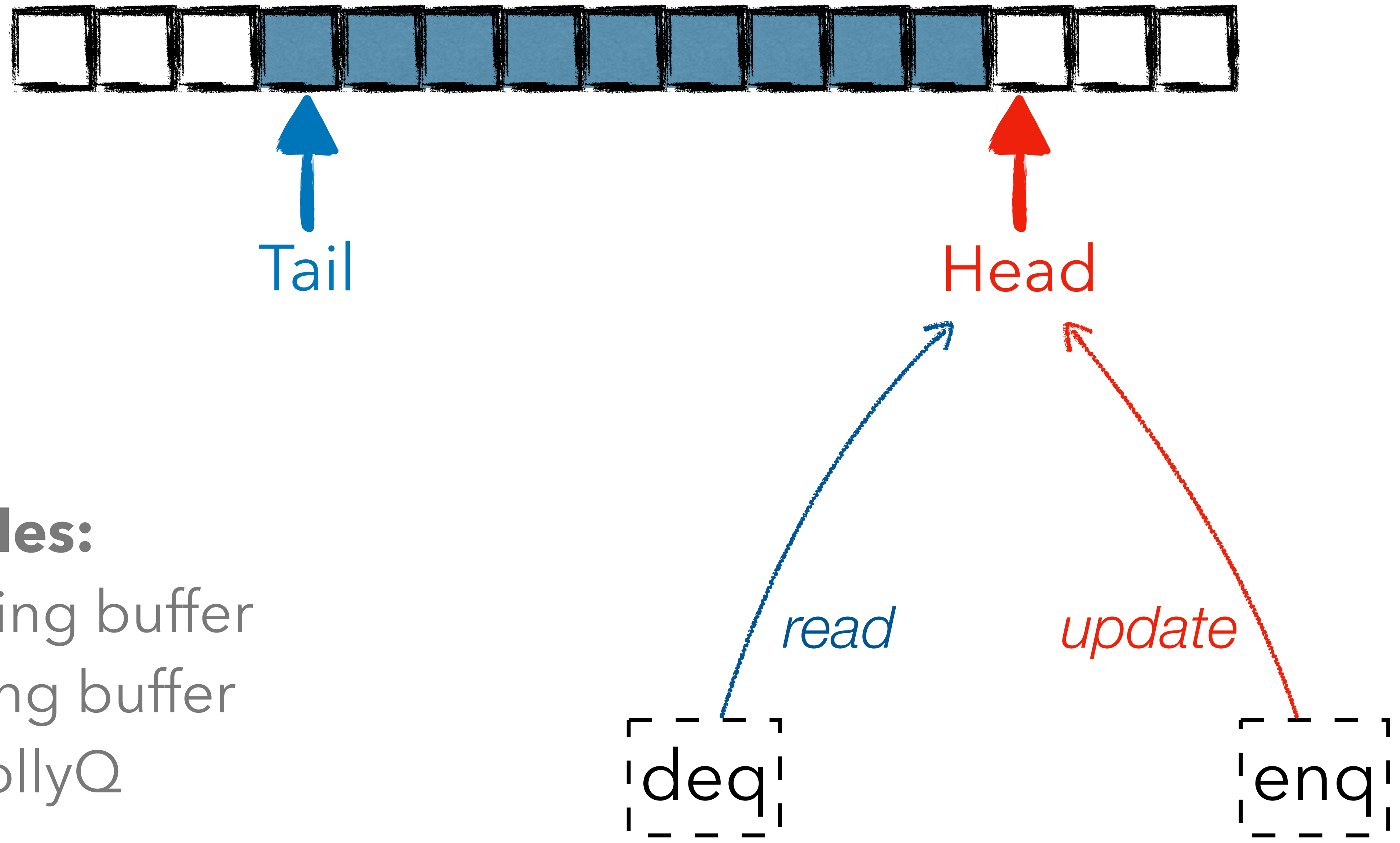


Examples:
DPDK ring buffer
Linux ring buffer
Meta FollyQ
SCQ

Mostly neglected

Interference sources – Existing work

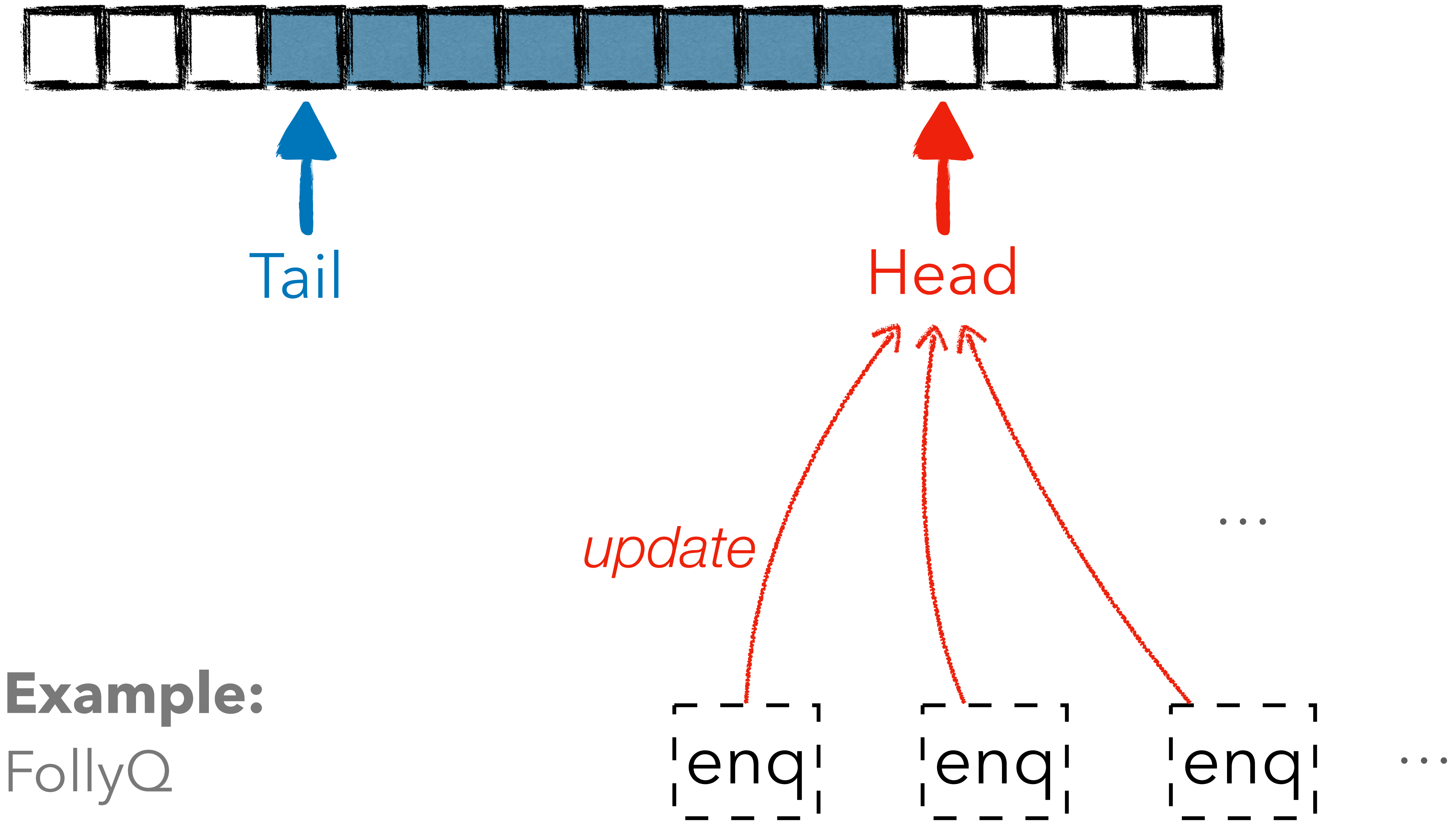
Enq-Deq interference



Examples:
DPDK ring buffer
Linux ring buffer
Meta FollyQ
SCQ

Mostly neglected

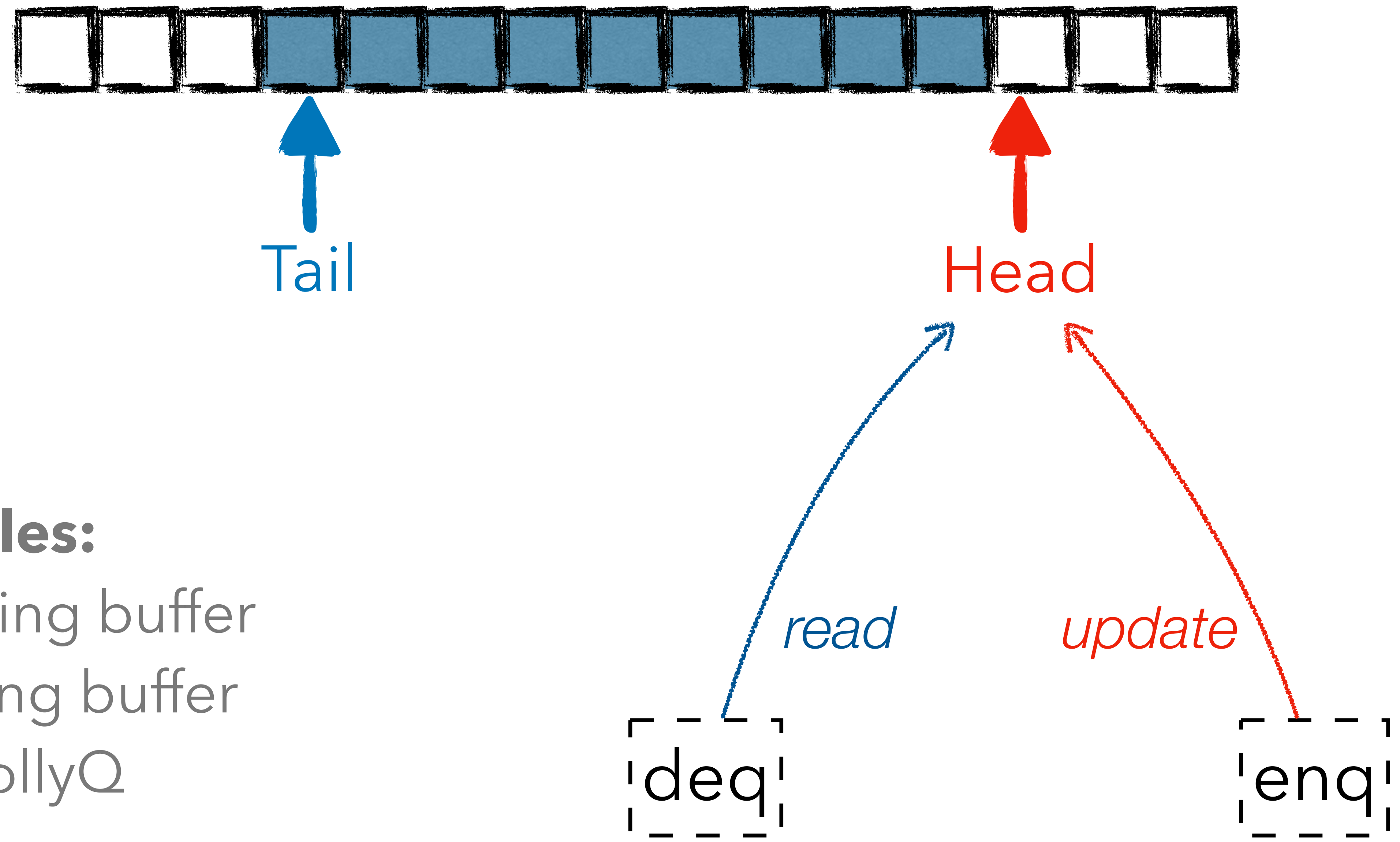
Enq-Enq interference



Example:
FollyQ

Interference sources – Existing work

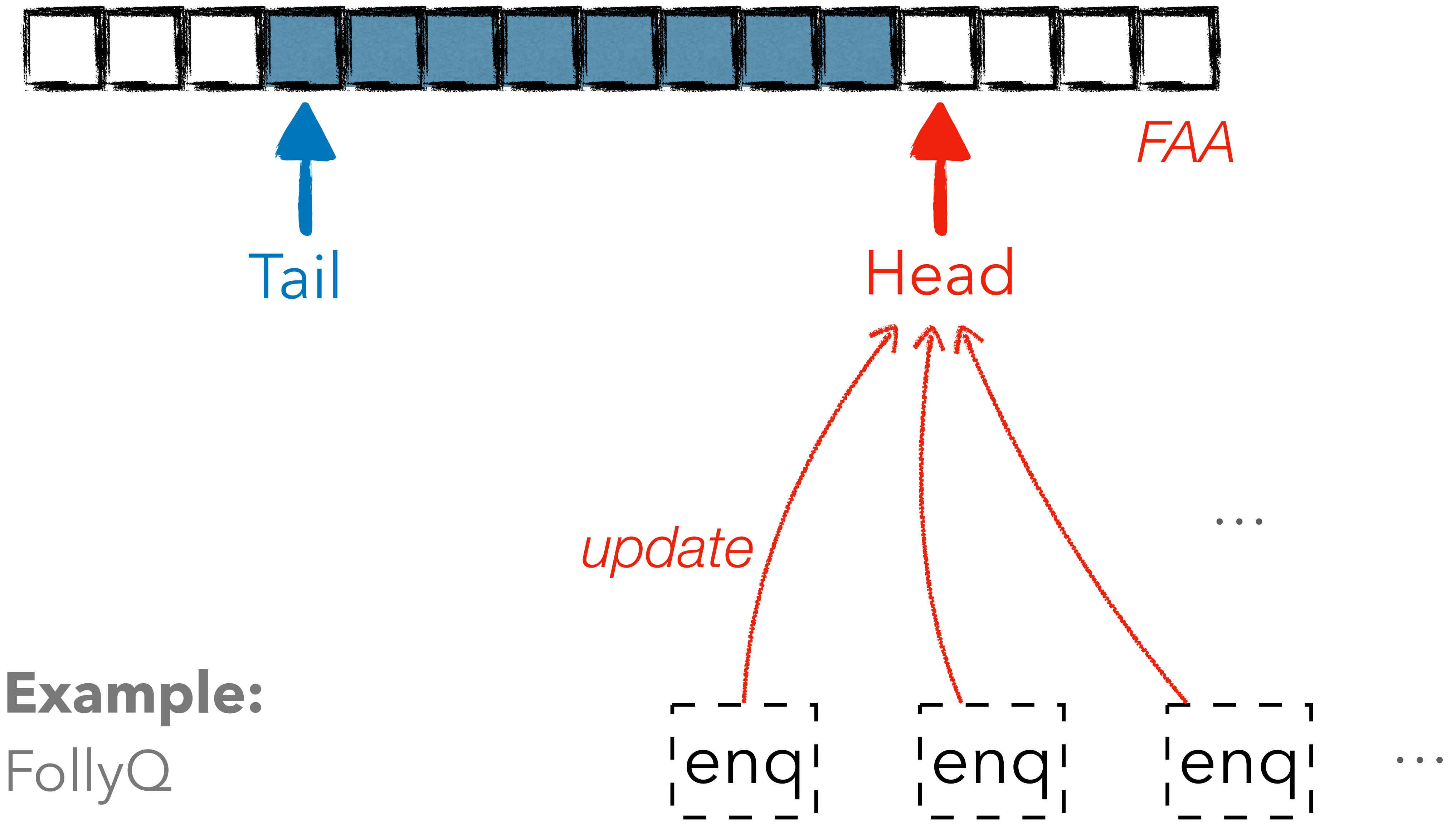
Enq-Deq interference



Examples:
DPDK ring buffer
Linux ring buffer
Meta FollyQ
SCQ

Mostly neglected

Enq-Enq interference

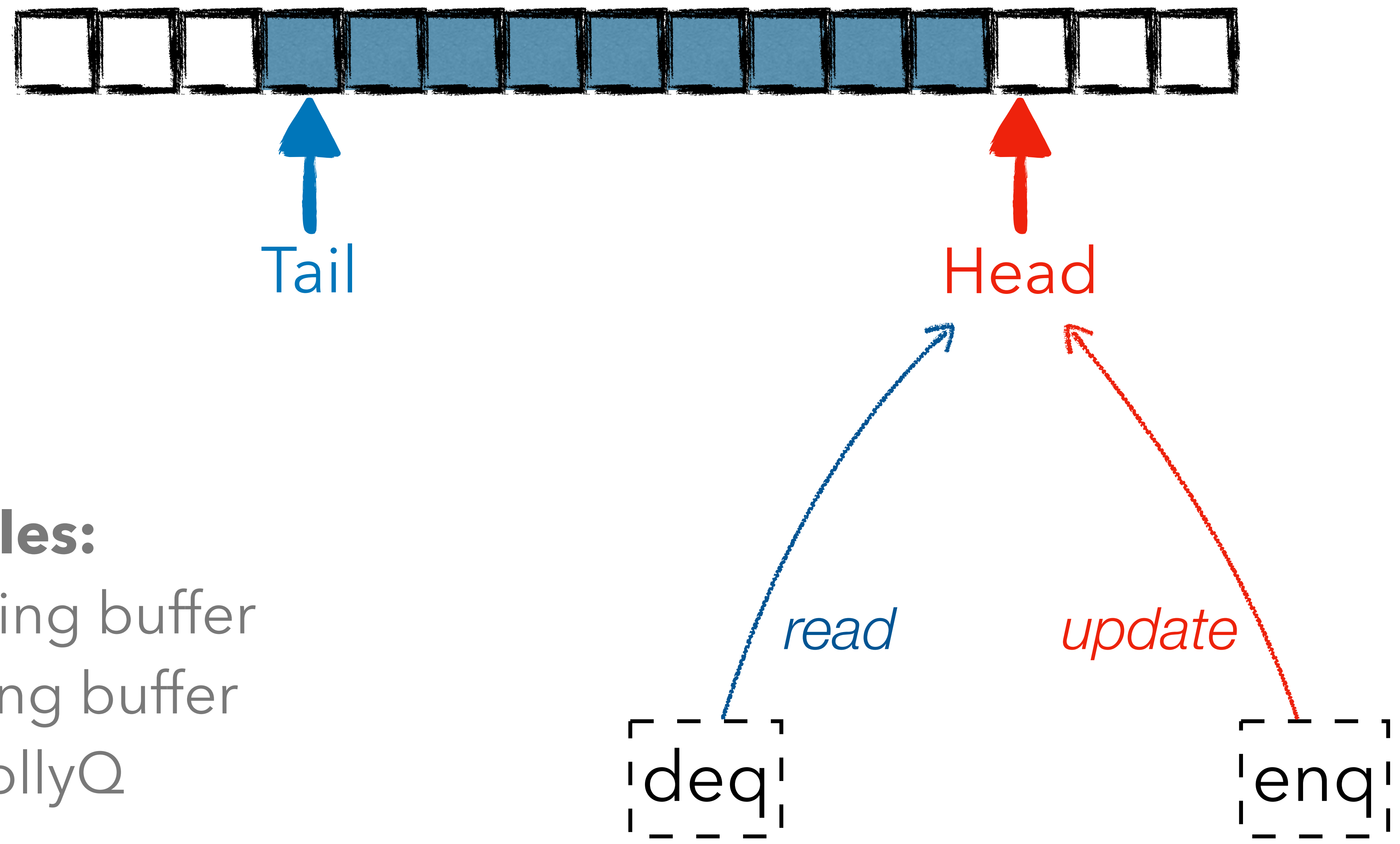


Example:
FollyQ

FAA typically faster than CAS

Interference sources – Existing work

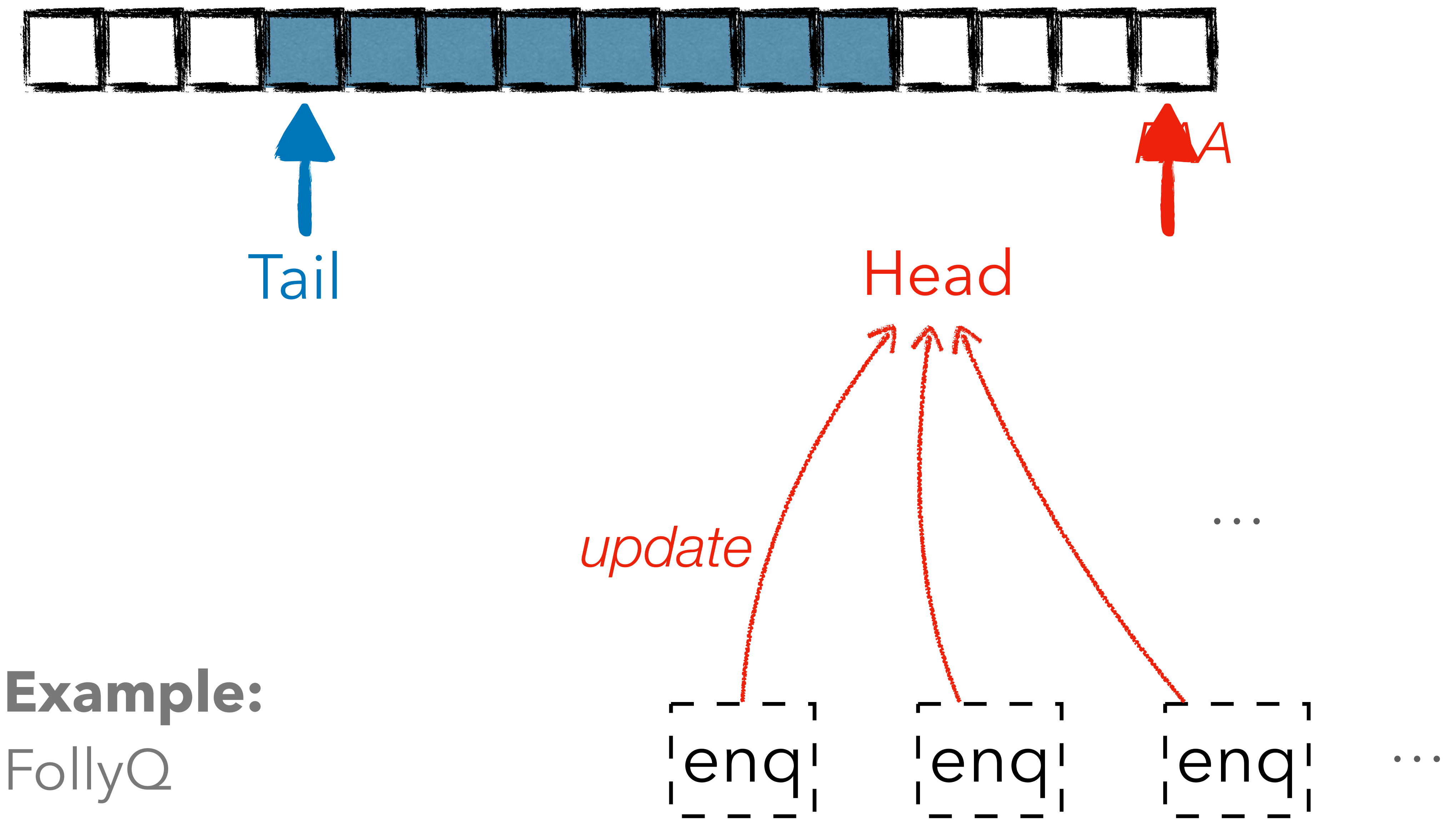
Enq-Deq interference



Examples:
DPDK ring buffer
Linux ring buffer
Meta FollyQ
SCQ

Mostly neglected

Enq-Enq interference

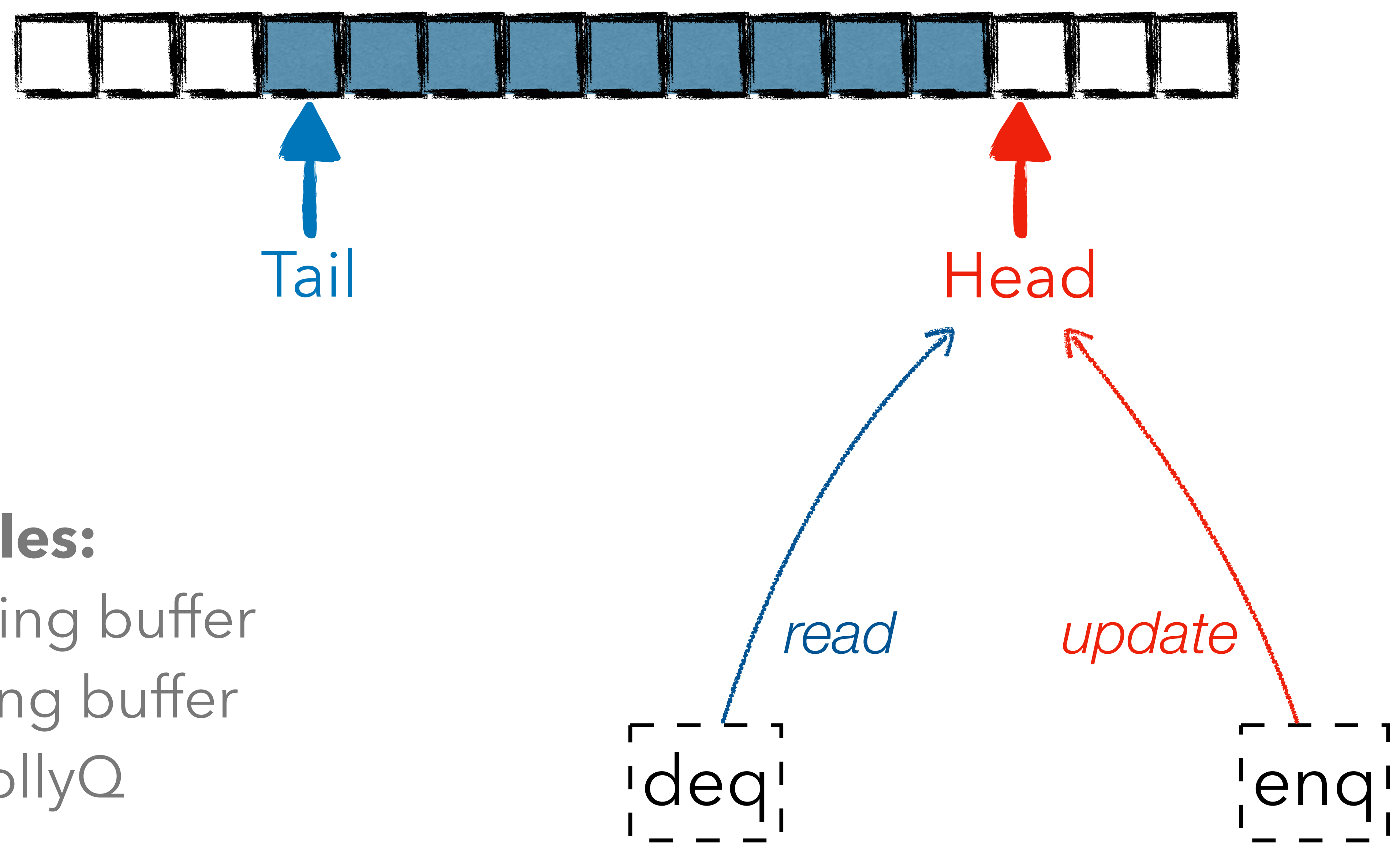


Example:
FollyQ

FAA typically faster than CAS

Interference sources – Existing work

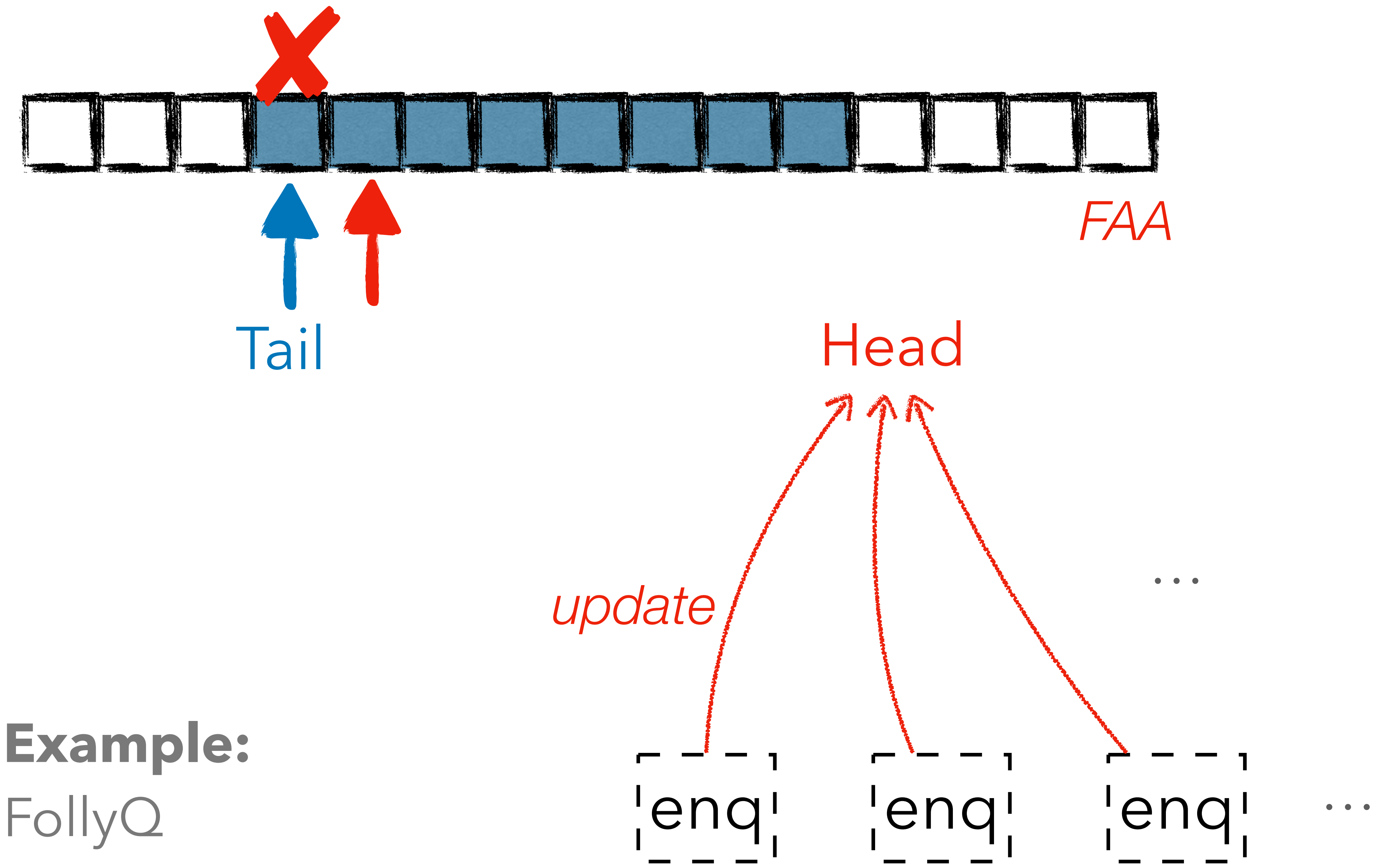
Enq-Deq interference



Examples:
DPDK ring buffer
Linux ring buffer
Meta FollyQ
SCQ

Mostly neglected

Enq-Enq interference



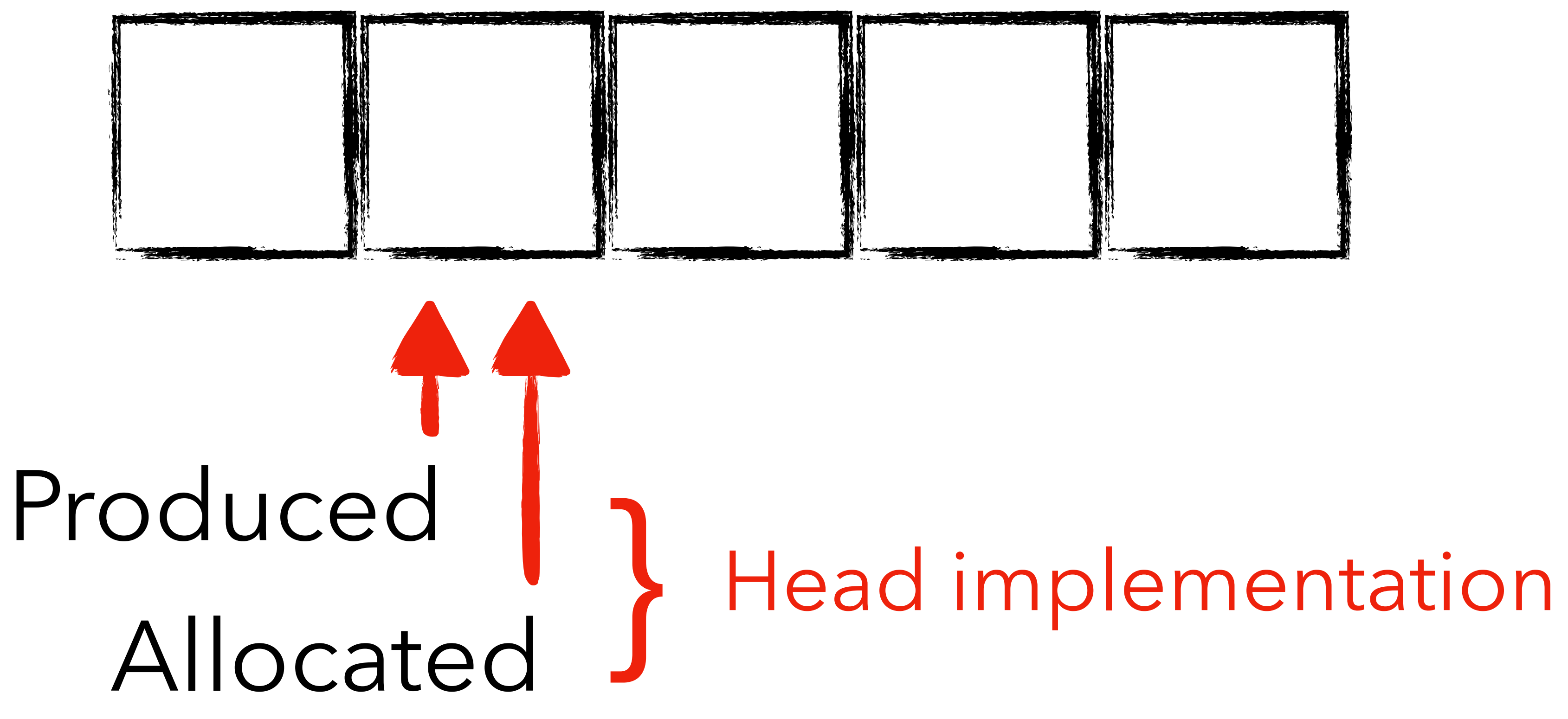
Example:
FollyQ

FAA typically faster than CAS

Story 2: Oversubscription and out-of-order operations

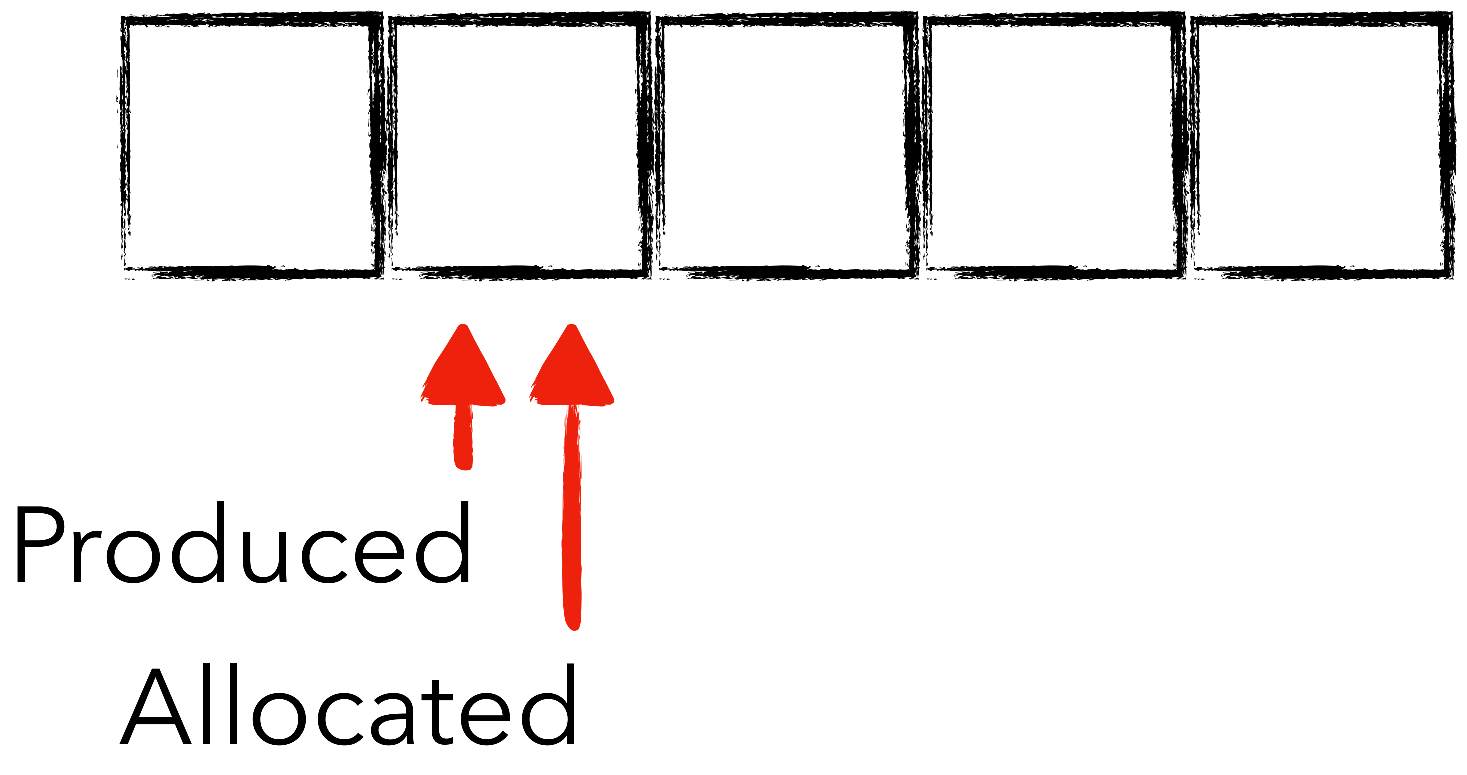
- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

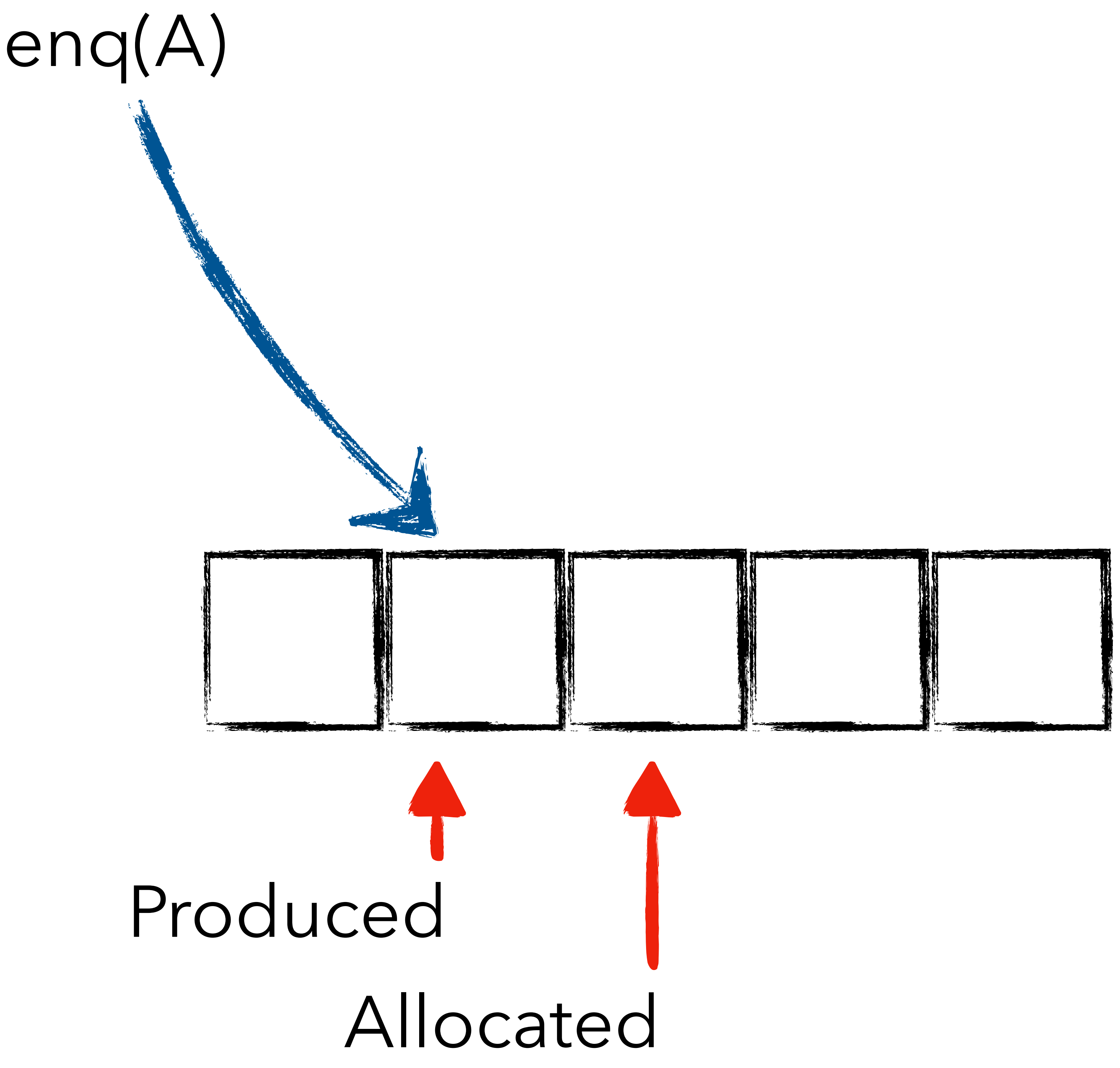
- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers
- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

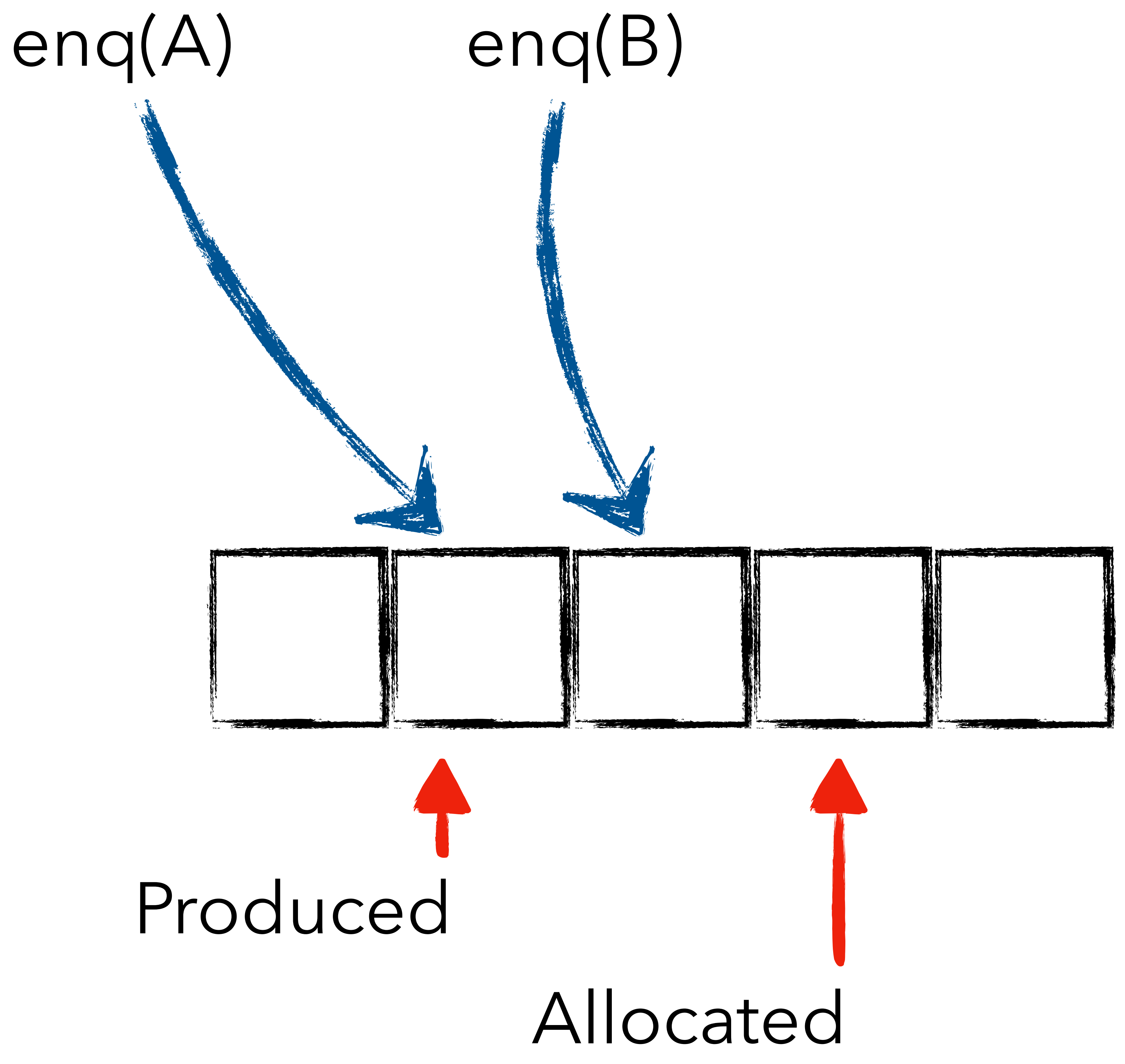
- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

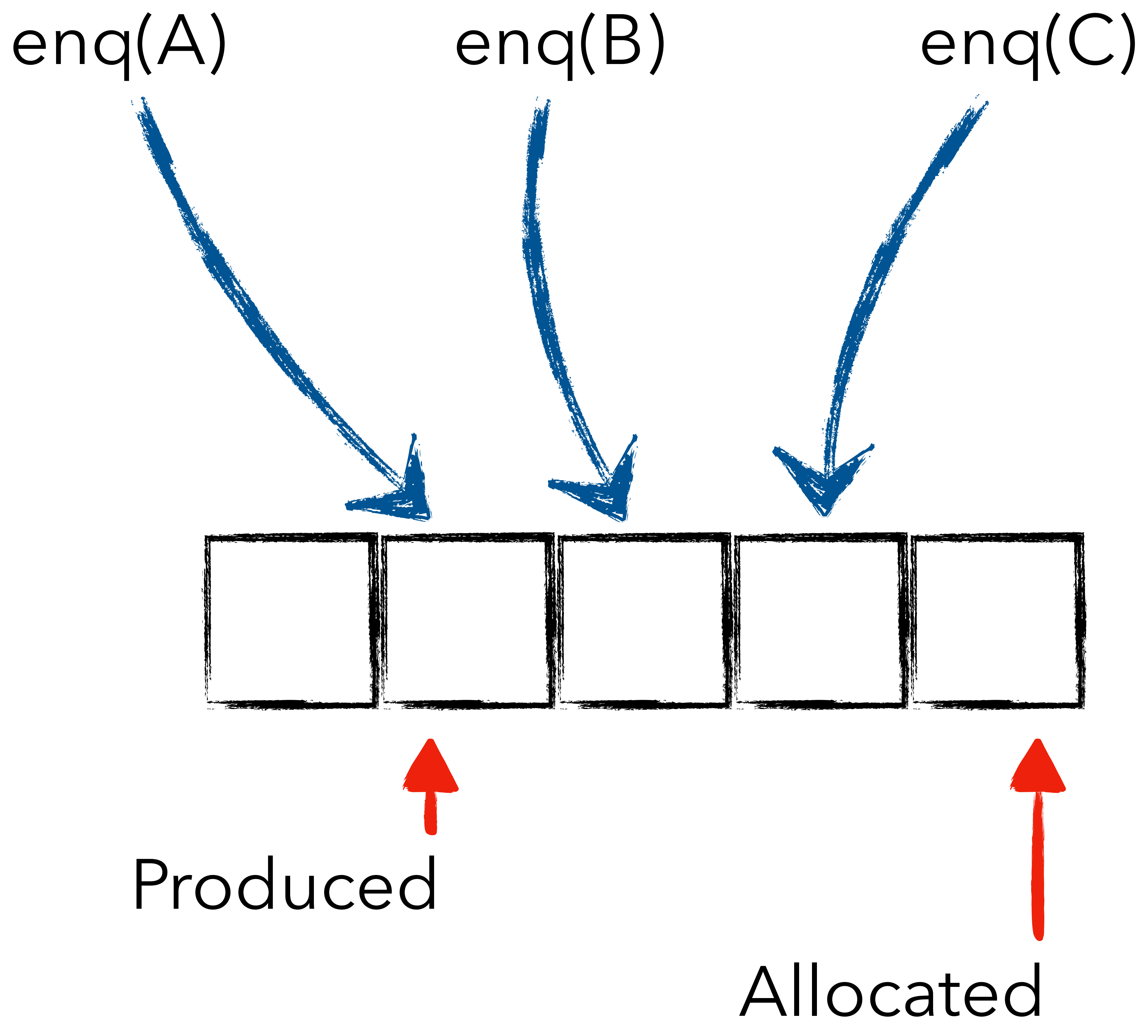
- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers
- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

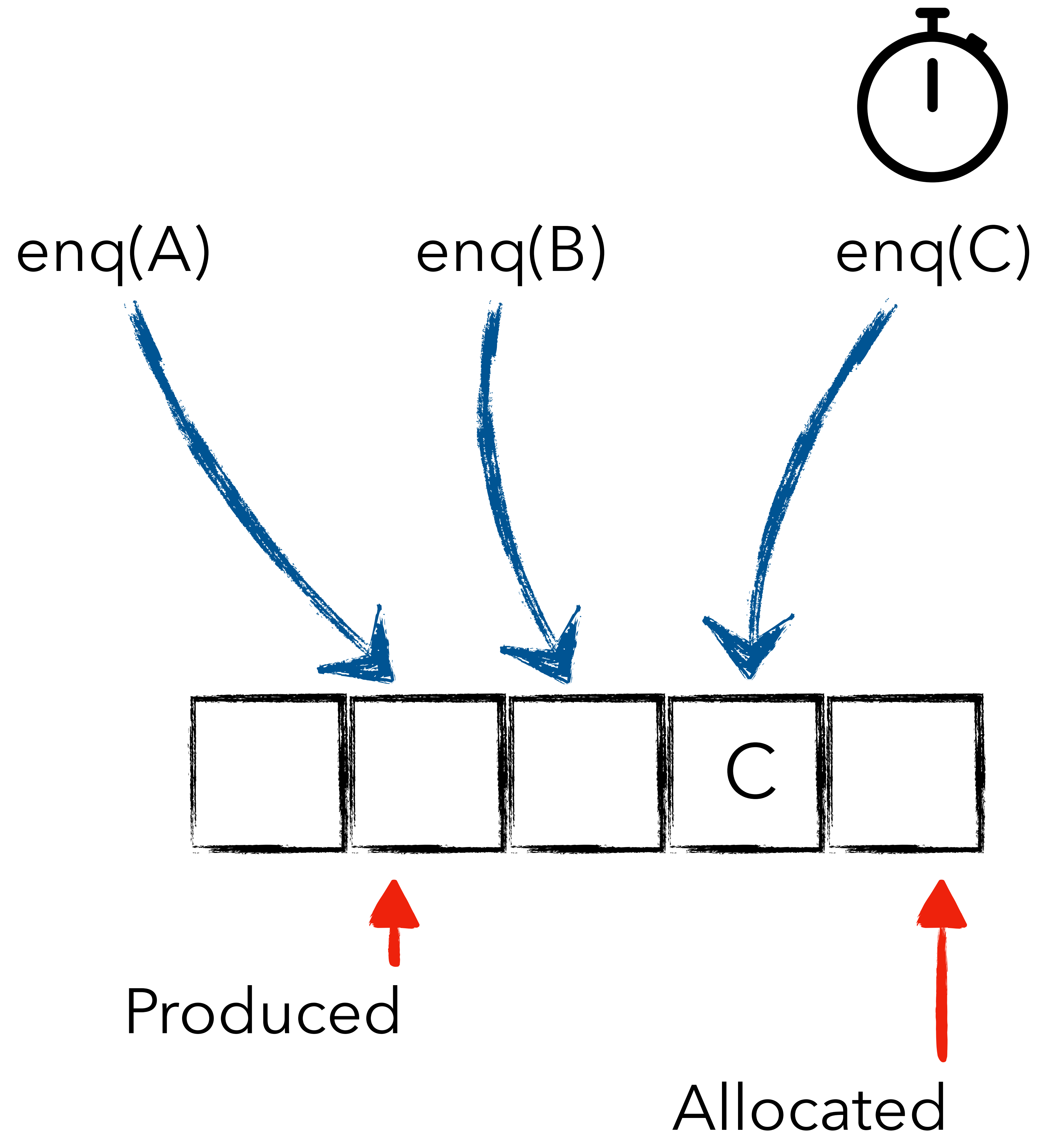
- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

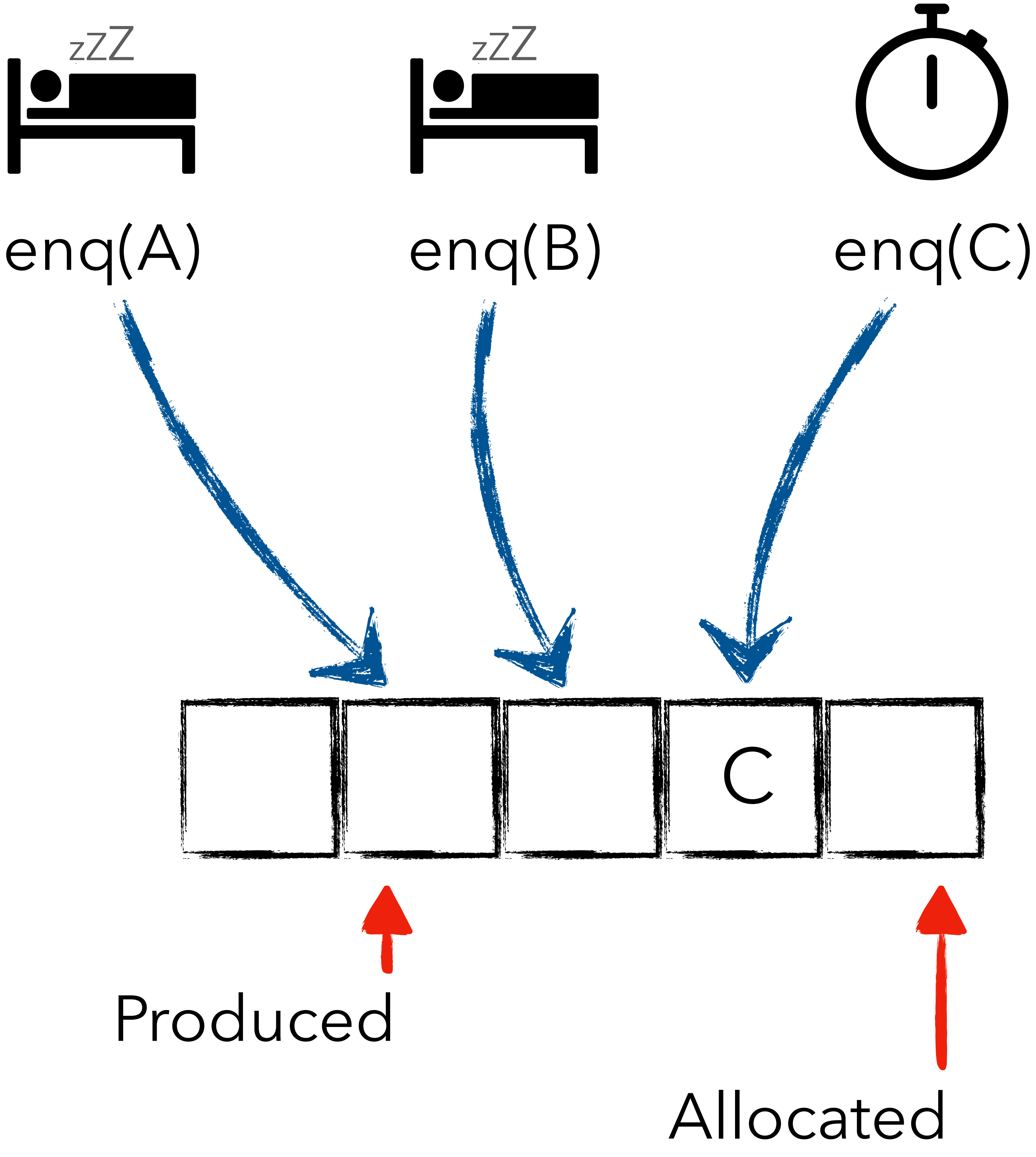
- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

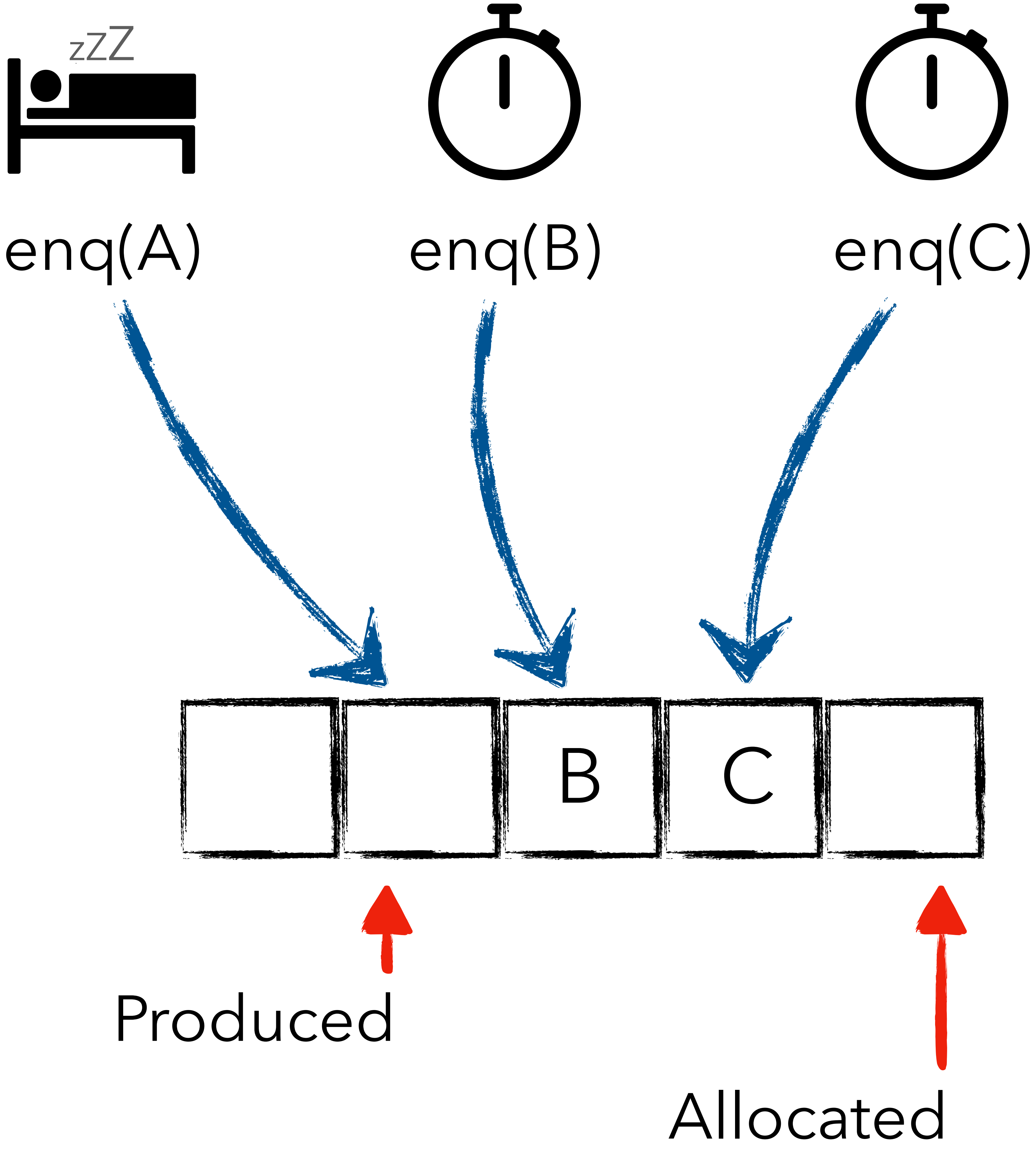
- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

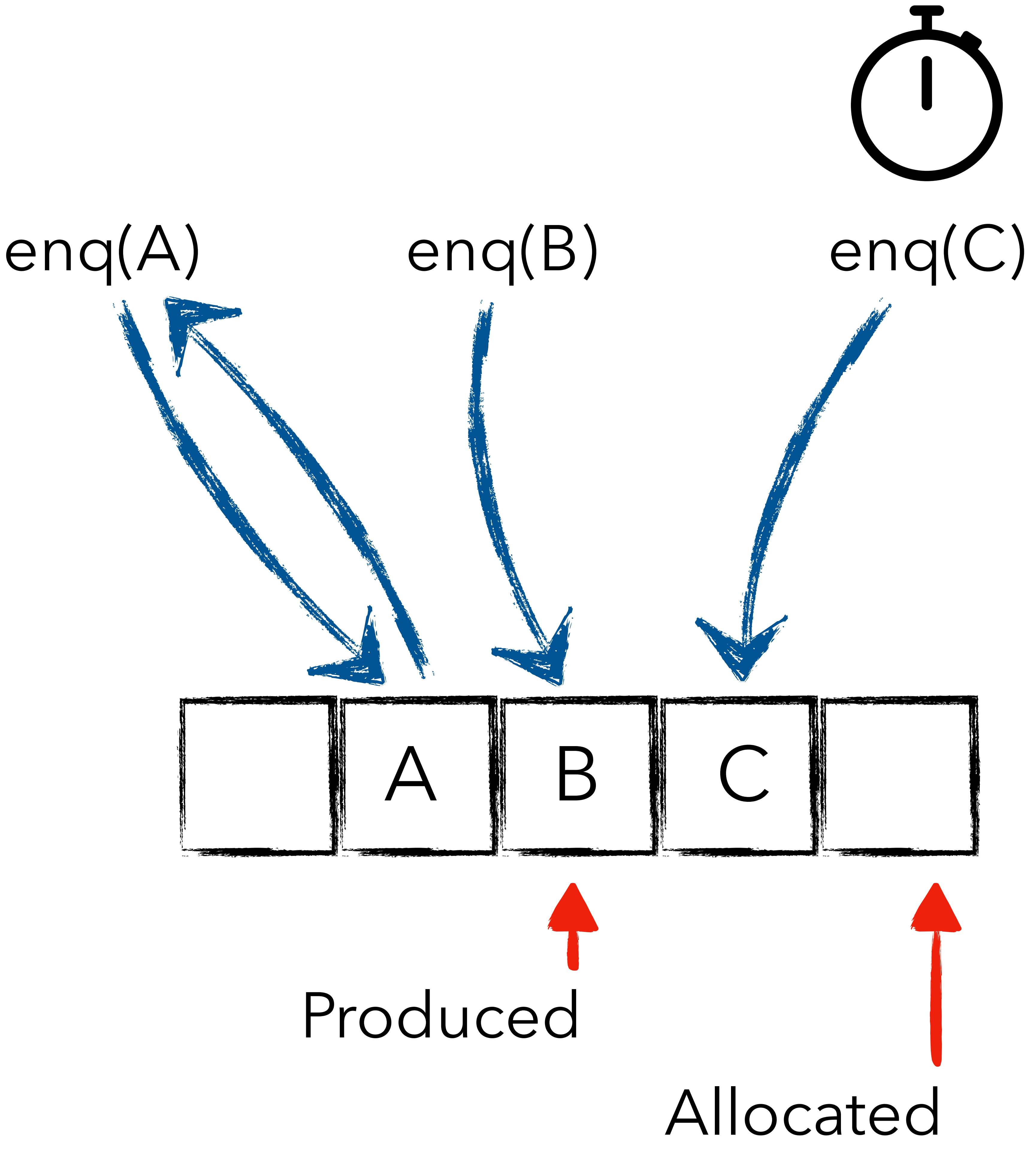
- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

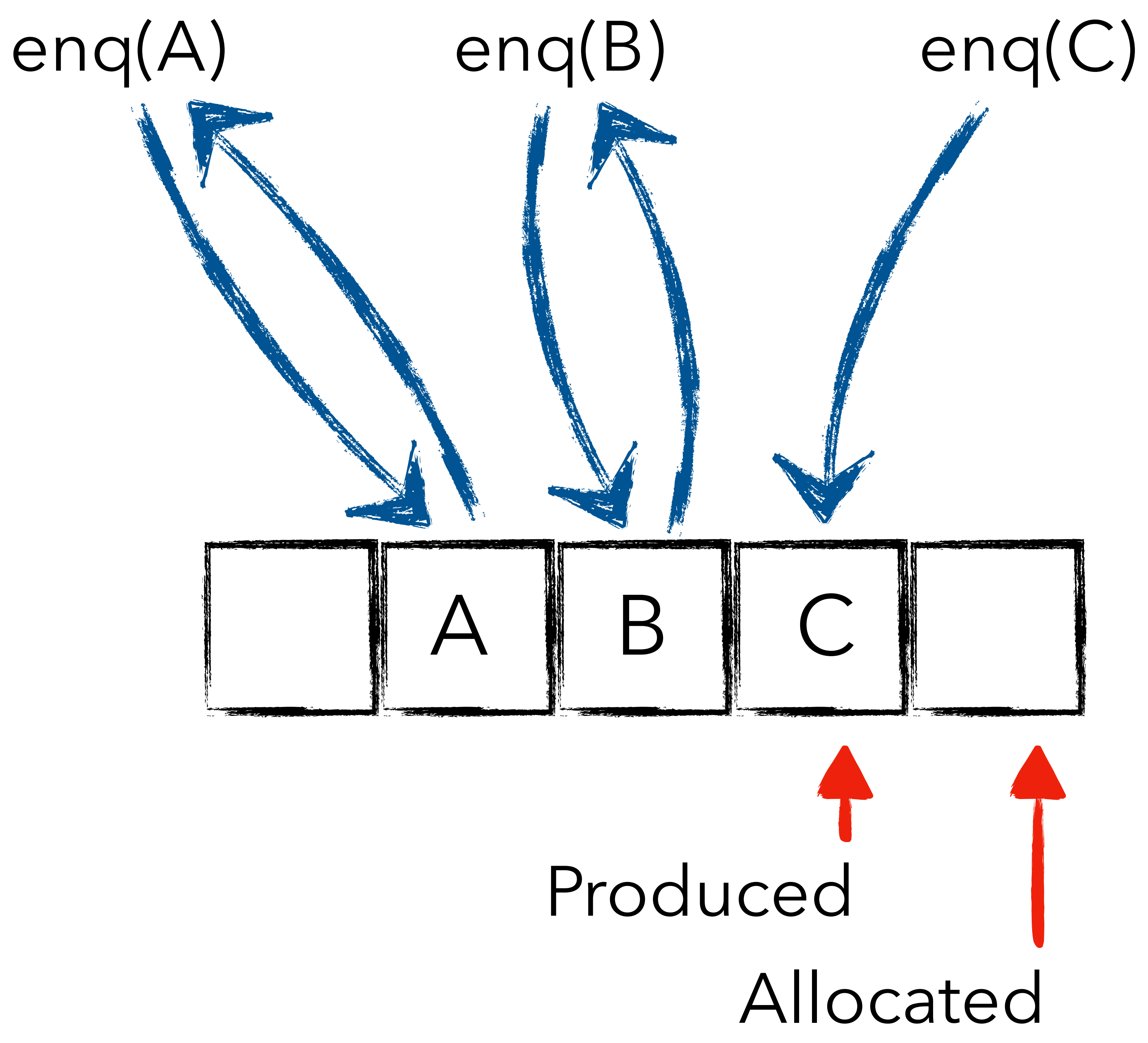
- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

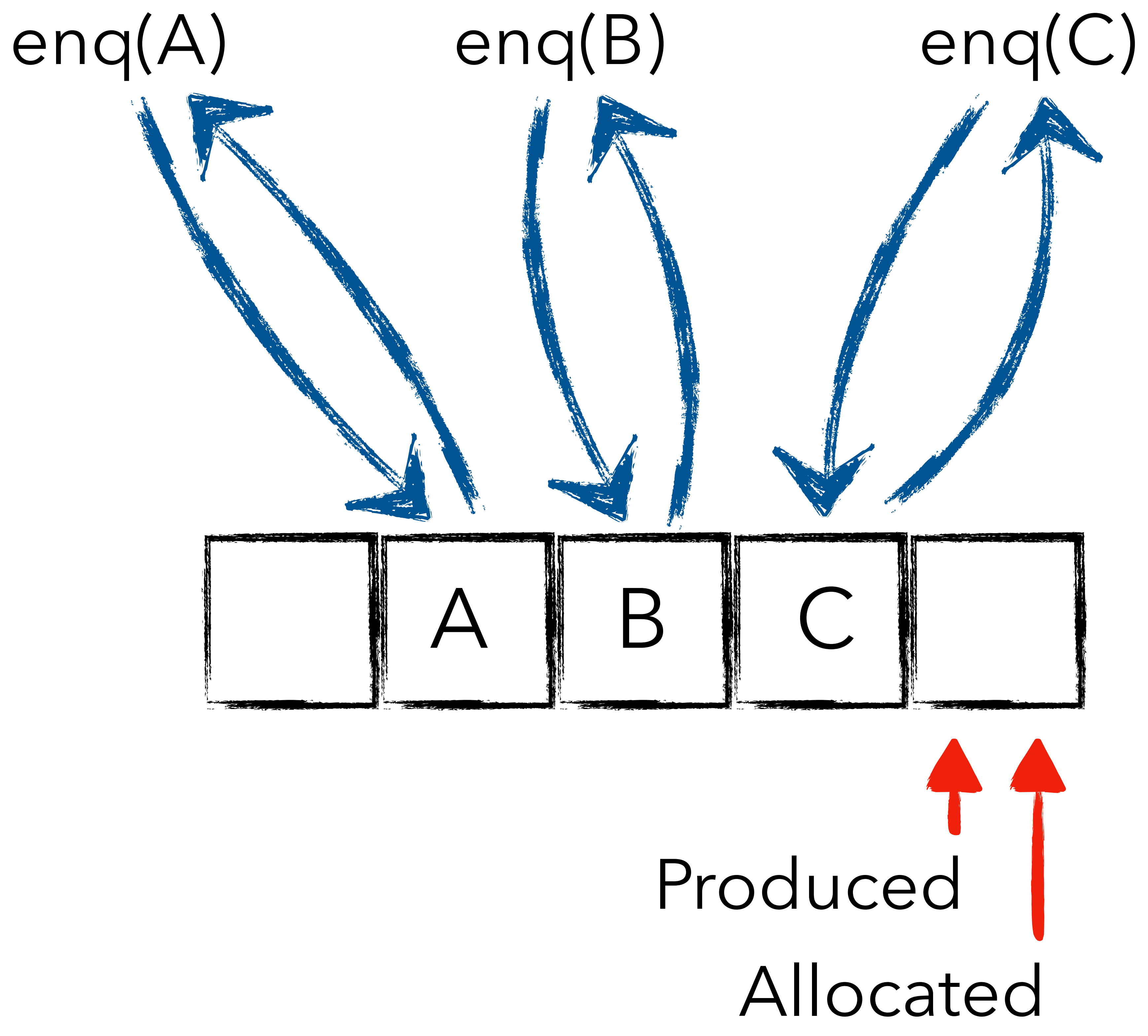
- Problem with initial implementation:
  - In-order operation **limits** performance



Story 2: Oversubscription and out-of-order operations

- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

- Problem with initial implementation:
  - In-order operation **limits** performance

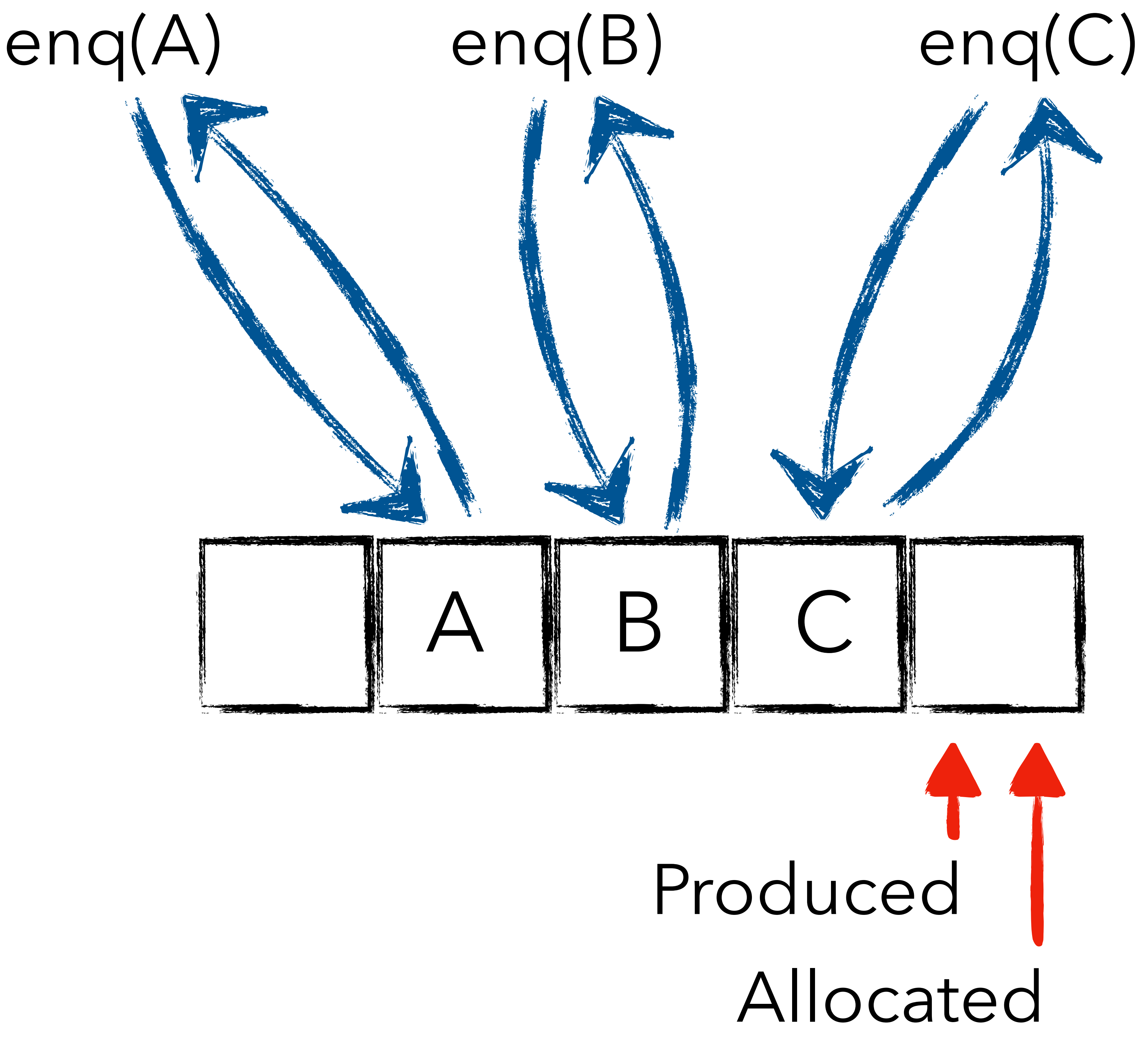


Story 2: Oversubscription and out-of-order operations

- New framework for mobile devices
  - **Few cores and many threads**
  - Communication via ring buffers

- Problem with initial implementation:
  - In-order operation **limits** performance

- Out-of-order operations are **challenging**!
  - See paper for related work



Story 3: Migrating from x86 to Arm

Product stable on x86 for years

- Must migrate to Arm (TaiShan servers)
- Internally uses **old DPDK ring buffer**
- Application grew **intertwined** with ring buffer

Story 3: Migrating from x86 to Arm

Product stable on x86 for years

- Must migrate to Arm (TaiShan servers)
- Internally uses **old DPDK ring buffer**
- Application grew **intertwined** with ring buffer

Decision: **too high risk** of upgrading DPDK ring buffer

Story 3: Migrating from x86 to Arm

Product stable on x86 for years

- Must migrate to Arm (TaiShan servers)
- Internally uses **old DPDK ring buffer**
- Application grew **intertwined** with ring buffer

Decision: **too high risk** of upgrading DPDK ring buffer

Consequence:

- **Annoying weak memory bug**

  due to a few missing fences
- More than **6 person-month to fix it**
- Decision wasn't the best

Story 3: Migrating from x86 to Arm

Product stable on x86 for years

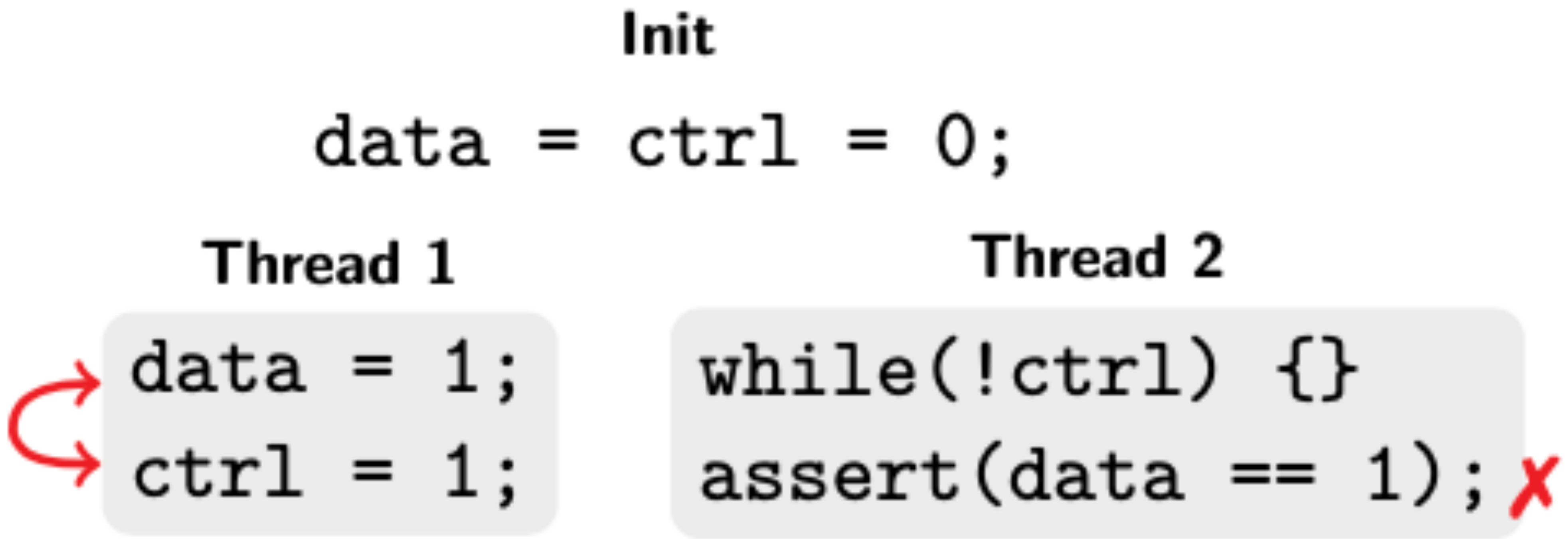
- Must migrate to Arm (TaiShan servers)
- Internally uses **old DPDK ring buffer**
- Application grew **intertwined** with ring buffer

Decision: **too high risk** of upgrading DPDK ring buffer

Consequence:

- **Annoying weak memory bug**

  due to a few missing fences
- More than **6 person-month to fix it**
- Decision wasn't the best



```
              Init
      data = ctrl = 0;
  Thread 1              Thread 2
data = 1;        while(!ctrl) {}
ctrl = 1;        assert(data == 1); ✗
```

Story 3: Migrating from x86 to Arm

Product stable on x86 for years

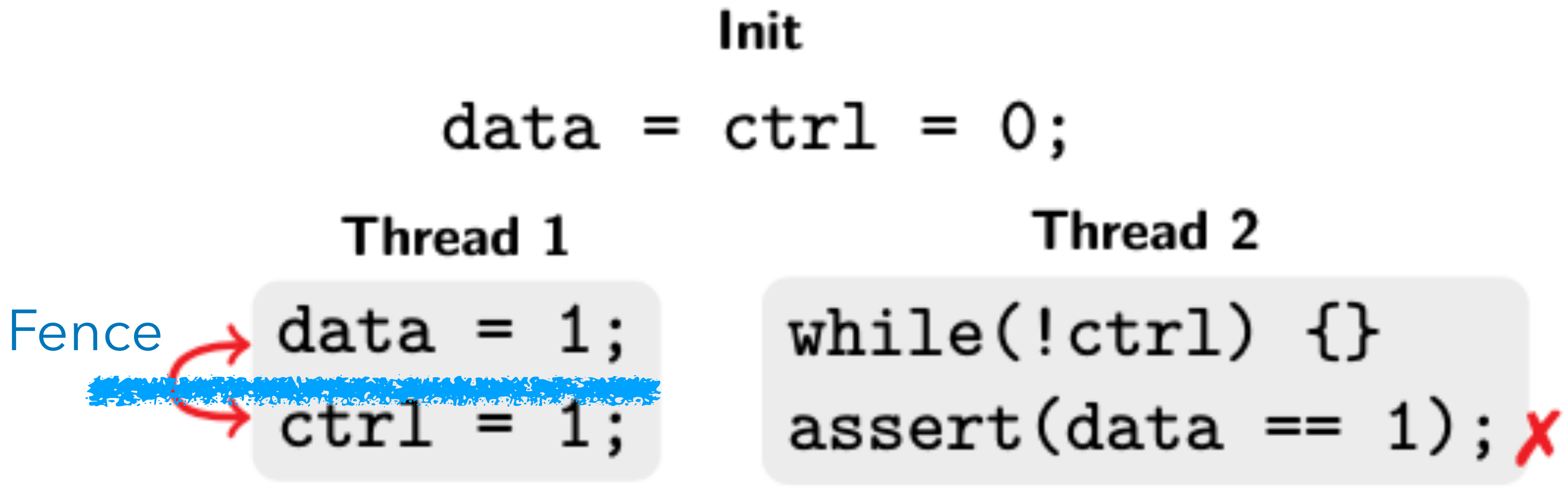
- Must migrate to Arm (TaiShan servers)
- Internally uses **old DPDK ring buffer**
- Application grew **intertwined** with ring buffer

Decision: **too high risk** of upgrading DPDK ring buffer

Consequence:

- **Annoying weak memory bug**

  due to a few missing fences
- More than **6 person-month to fix it**
- Decision wasn't the best



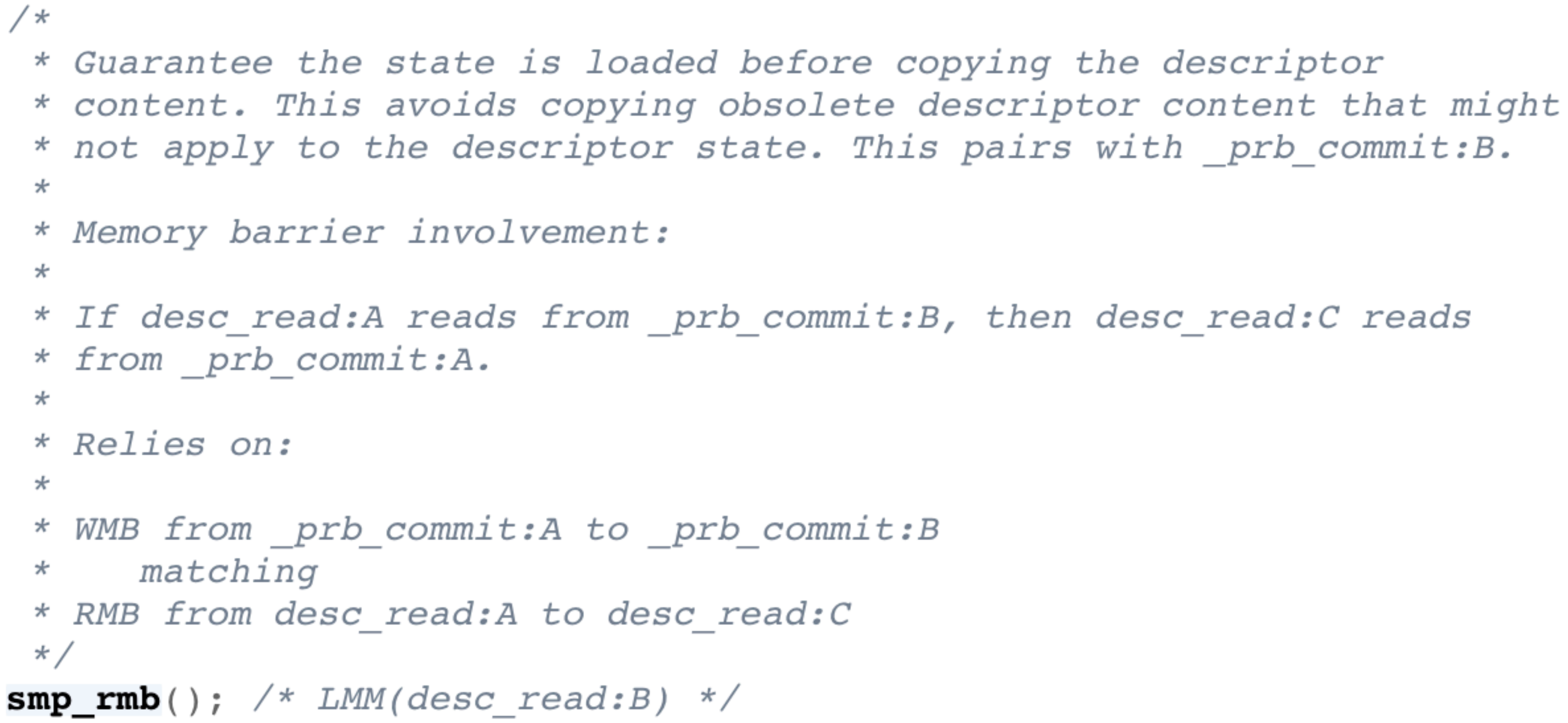
```
              Init
       data = ctrl = 0;

  Thread 1            Thread 2
data = 1;         while(!ctrl) {}
ctrl = 1;         assert(data == 1); ✗
```

How do people develop for WMM?

Think hard and document

For example, printk_ringbuffer

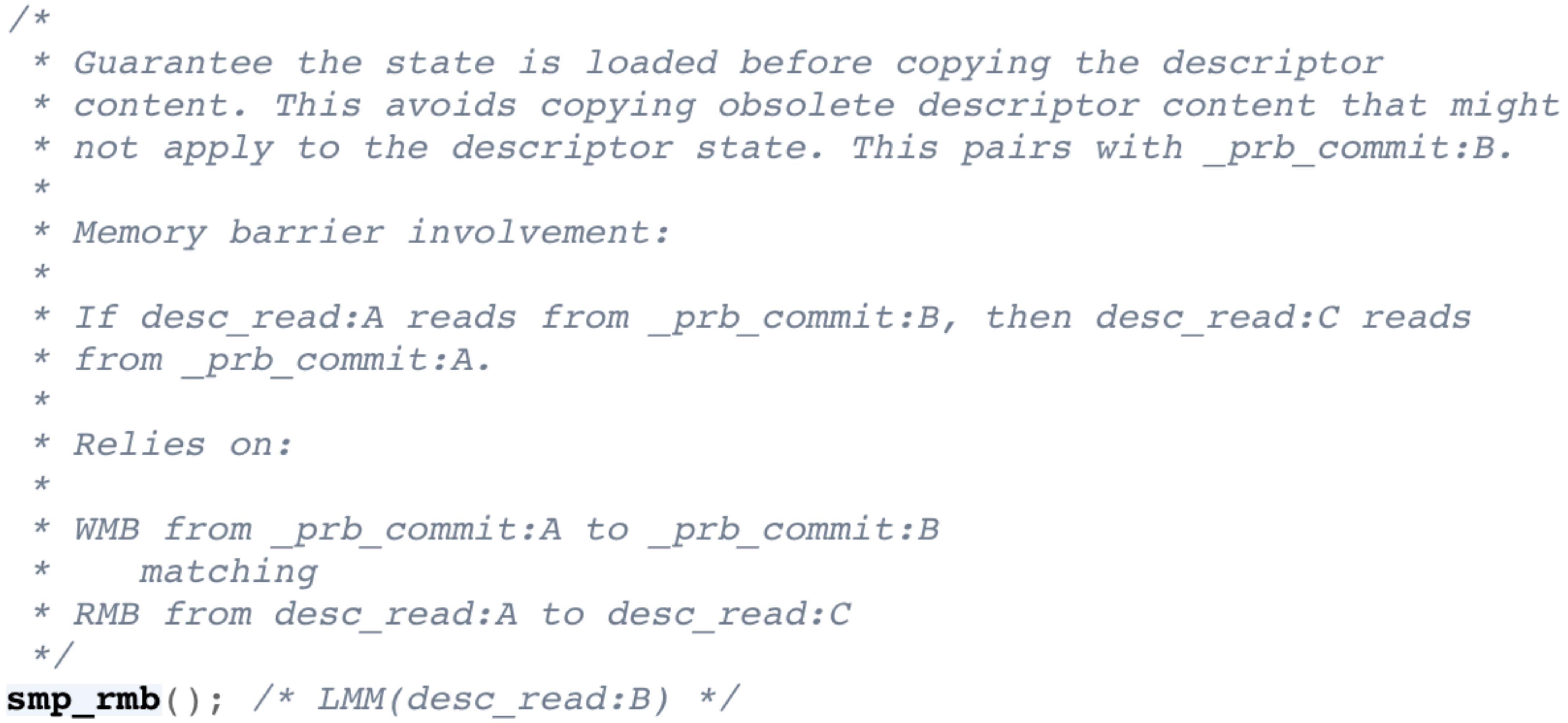


```
/*
 * Guarantee the state is loaded before copying the descriptor
 * content. This avoids copying obsolete descriptor content that might
 * not apply to the descriptor state. This pairs with _prb_commit:B.
 *
 * Memory barrier involvement:
 *
 * If desc_read:A reads from _prb_commit:B, then desc_read:C reads
 * from _prb_commit:A.
 *
 * Relies on:
 *
 * WMB from _prb_commit:A to _prb_commit:B
 *    matching
 * RMB from desc_read:A to desc_read:C
 */
smp_rmb(); /* LMM(desc_read:B) */
```

How do people develop for WMM?

Think hard and document

For example, printk_ringbuffer



```
/*
 * Guarantee the state is loaded before copying the descriptor
 * content. This avoids copying obsolete descriptor content that might
 * not apply to the descriptor state. This pairs with _prb_commit:B.
 *
 * Memory barrier involvement:
 *
 * If desc_read:A reads from _prb_commit:B, then desc_read:C reads
 * from _prb_commit:A.
 *
 * Relies on:
 *
 * WMB from _prb_commit:A to _prb_commit:B
 *    matching
 * RMB from desc_read:A to desc_read:C
 */
smp_rmb(); /* LMM(desc_read:B) */
```

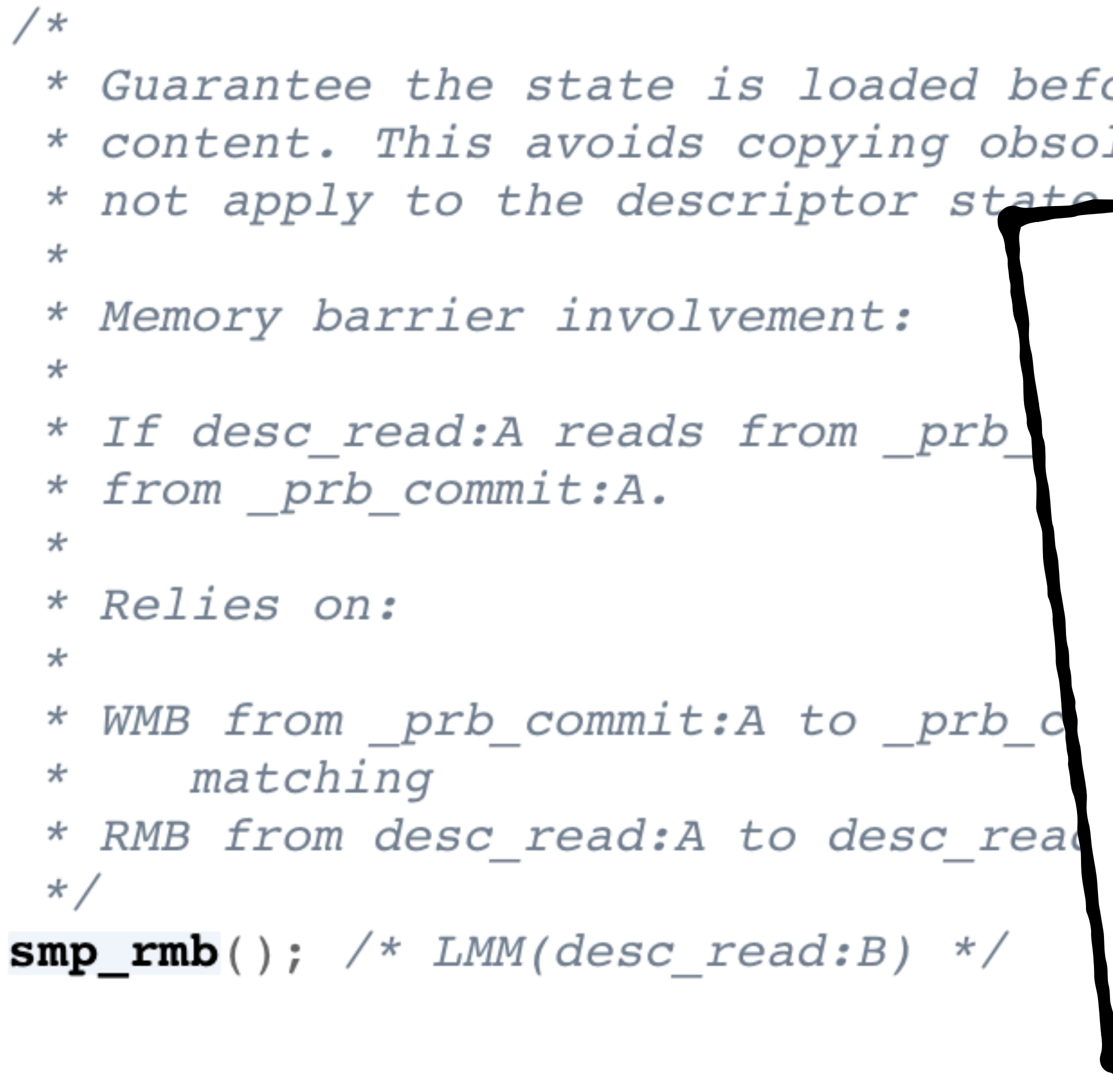
Most just ignore topic

And wait to see what happens

How do people develop for WMM?

Think hard and document

For example, printk_ringbuffer



```
/*
 * Guarantee the state is loaded before copying the descriptor
 * content. This avoids copying obsolete descriptor
 * not apply to the descriptor state
 *
 * Memory barrier involvement:
 *
 * If desc_read:A reads from _prb_
 * from _prb_commit:A.
 *
 * Relies on:
 *
 * WMB from _prb_commit:A to _prb_c
 *    matching
 * RMB from desc_read:A to desc_rea
 */
smp_rmb(); /* LMM(desc_read:B) */
```

Most just ignore topic

And wait to see what happens

What about using tools?

We are in 2022!

There are scalable model checkers for WMM!

e.g., GenMC, Dartagnan, VSync

Our contributions

BBQ: Block-based Bounded Queue

- Novel block-based design
- Focus on enq-deq interference
- Support for out-of-order operations
- Verified for WMMs, pragmatically

Our contributions

BBQ: Block-based Bounded Queue

- Novel block-based design
- Focus on enq-deq interference
- Support for out-of-order operations
- Verified for WMMs, pragmatically

Bonus features

- Single/multi producers/consumers
- Fixed- and variable-sized entries
- Retry-new and drop-old modes
- Use of efficient atomic operations
  - FAA and MAX (ARMv8.1 LSE)
  - No CAS at all if MAX available

Agenda

☑ Motivation

☑ Stories and Challenges

Interference, Out-of-order operations, Correctness on WMMs

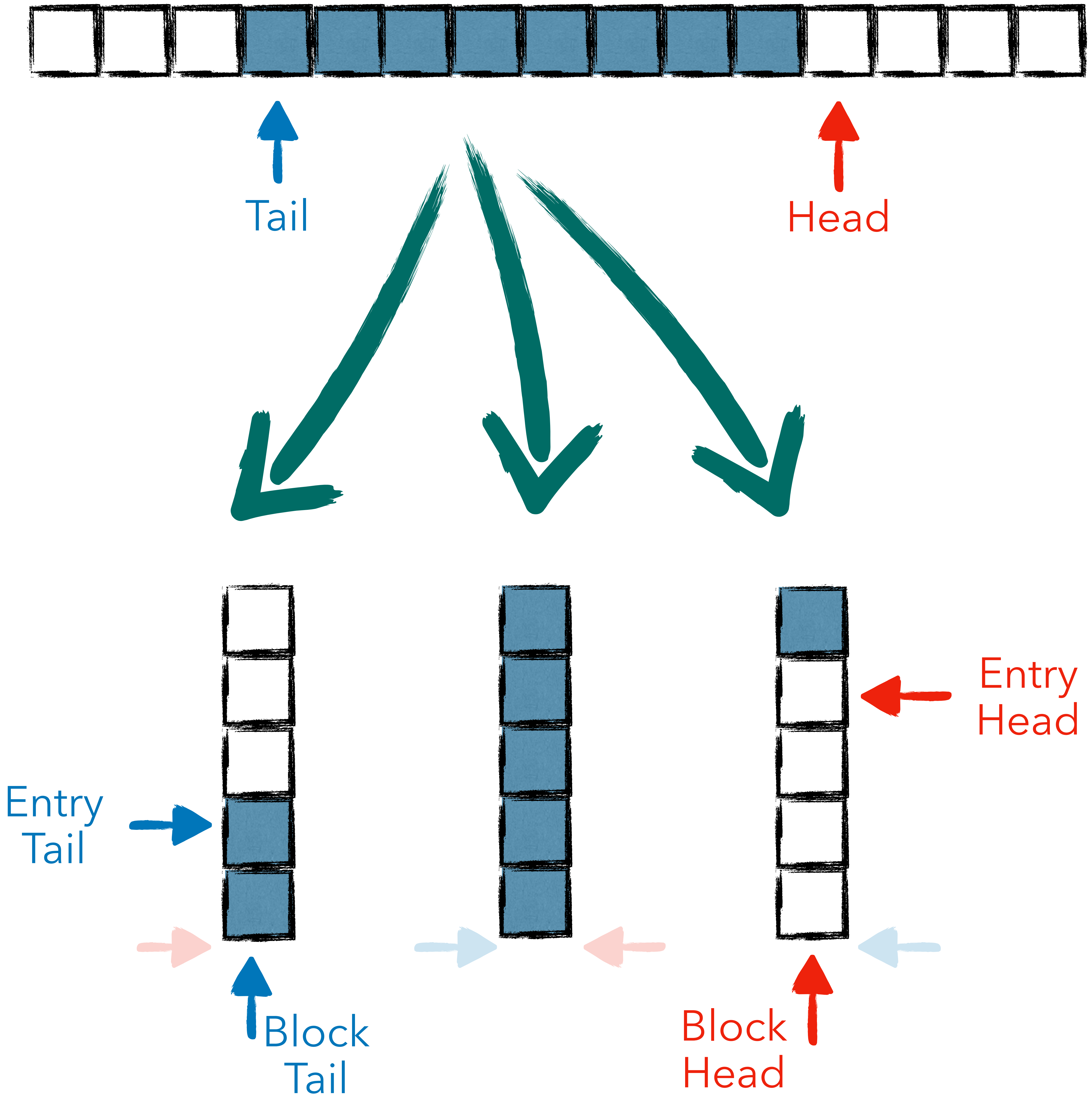
□ BBQ – Block-based Bounded Queue

□ Insights to Tackle the Challenges

□ Selected Evaluation Results

BBQ – Block-based Bounded Queue

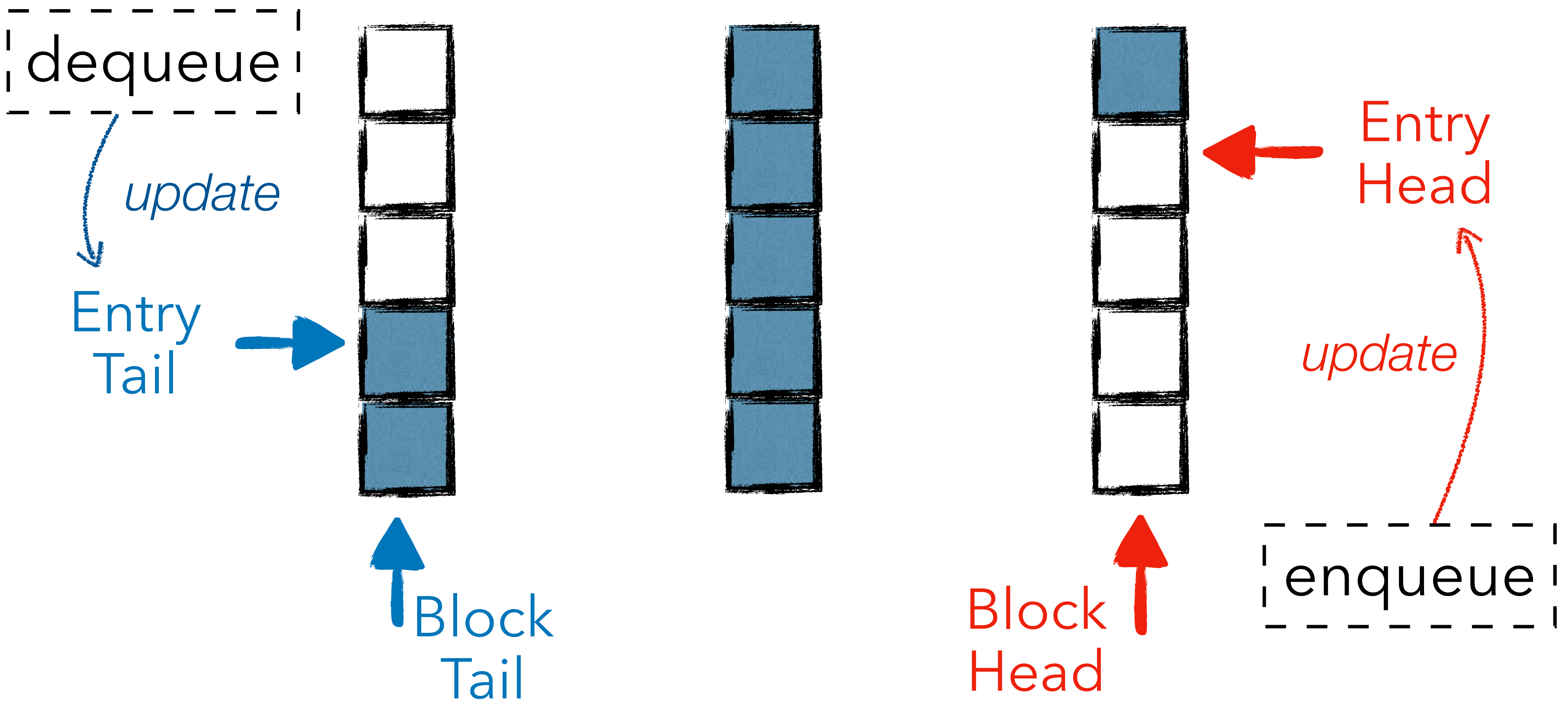
- Ring buffer split into blocks
- Block Head points to current producer block
- Block Tail points to current consumer block
- In each block: Entry Head and Entry Tail



Dealing with interferences

Enq-Deq interference

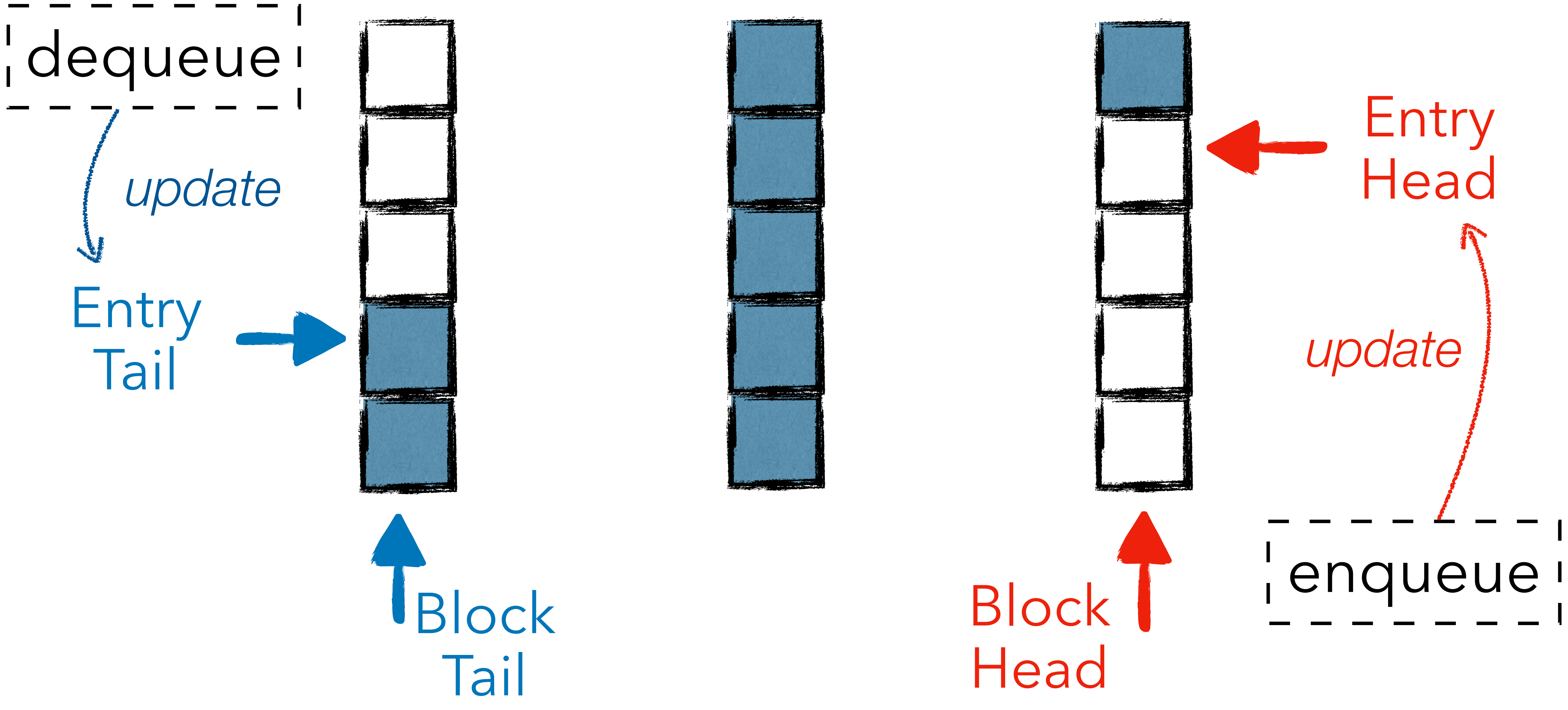
No interference when producer and consumer in different blocks



Dealing with interferences

Enq-Deq interference

No interference when producer and consumer in different blocks

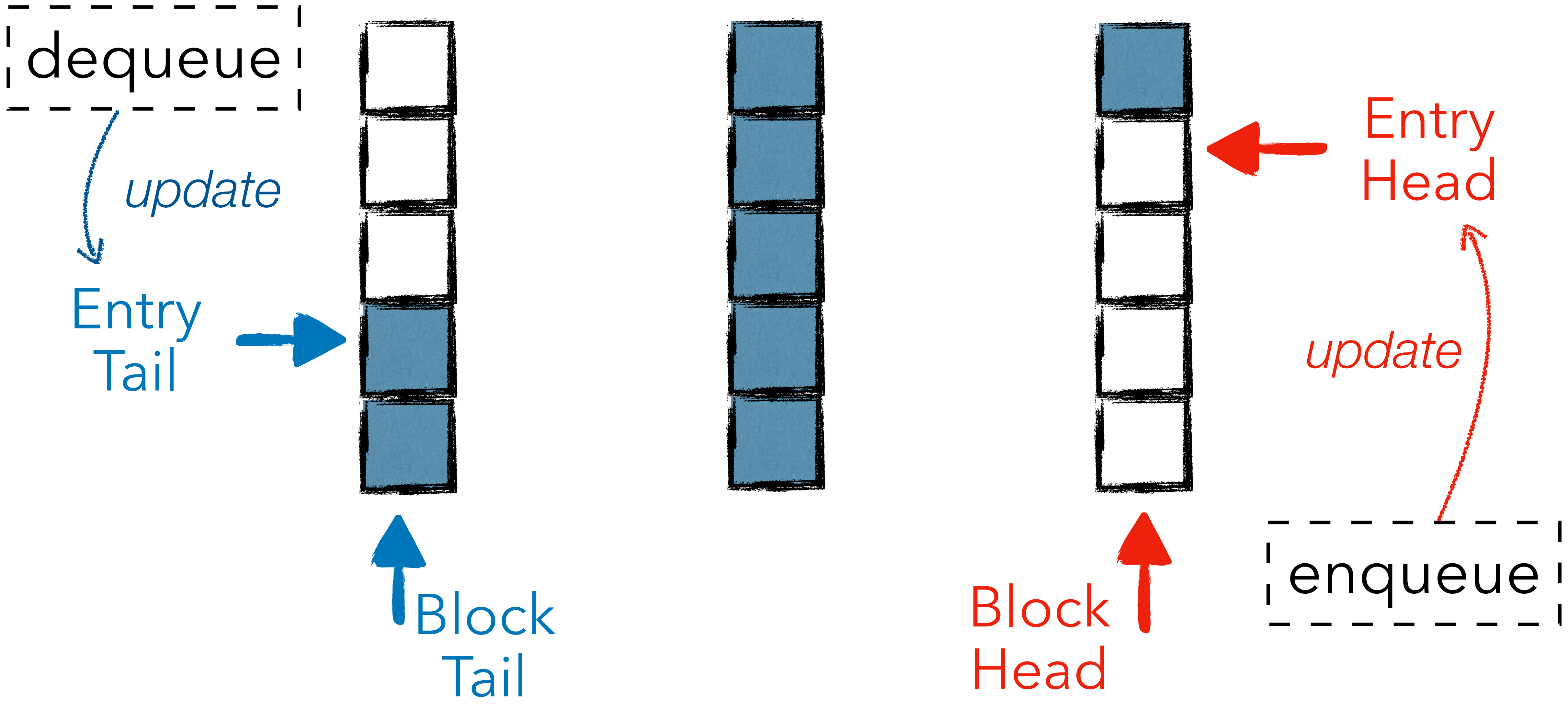


Block head and tail only read when moving to next block

Dealing with interferences

Enq-Deq interference

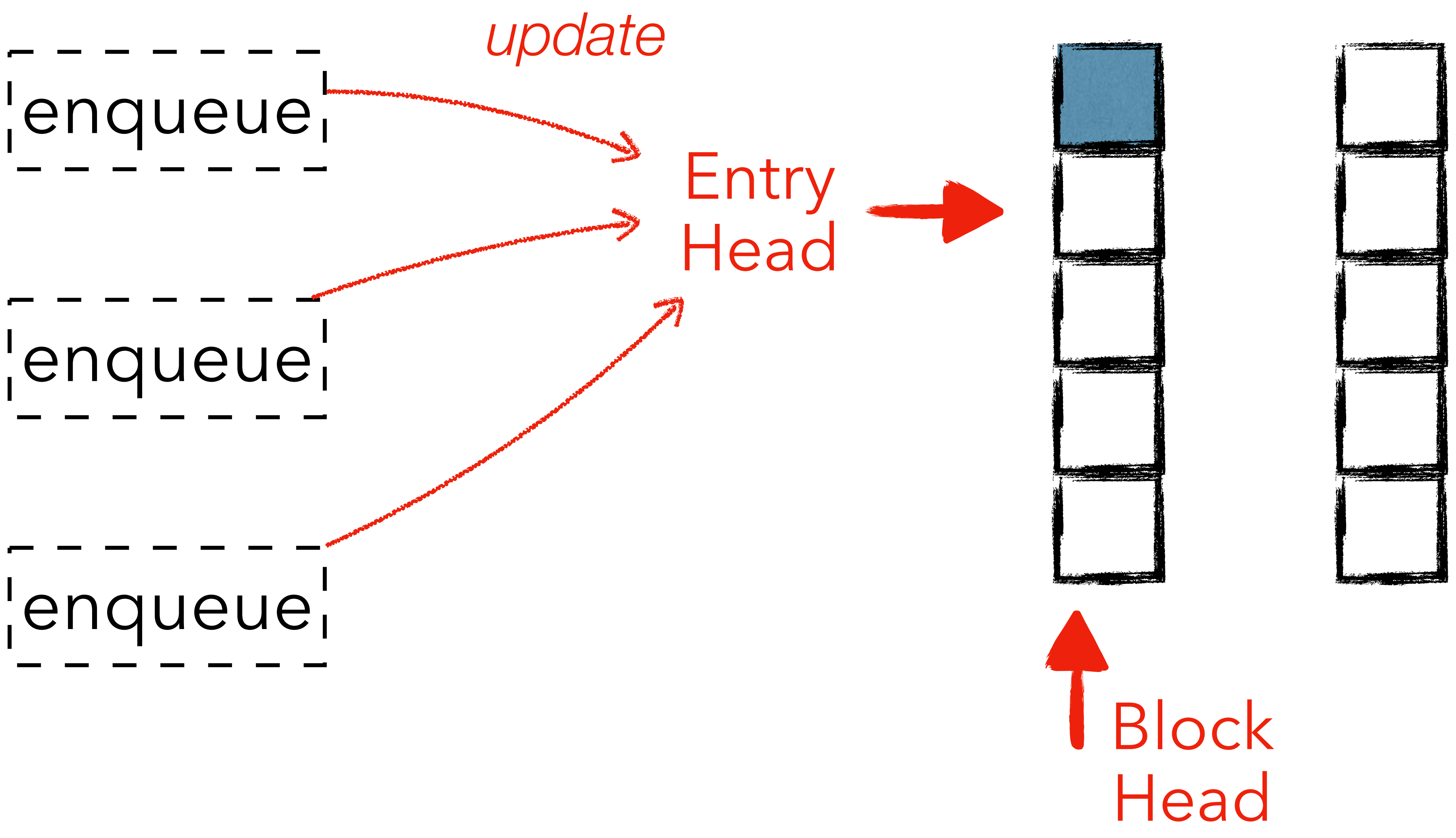
No interference when producer and consumer in different blocks



Block head and tail only read when moving to next block

Enq-Enq interference

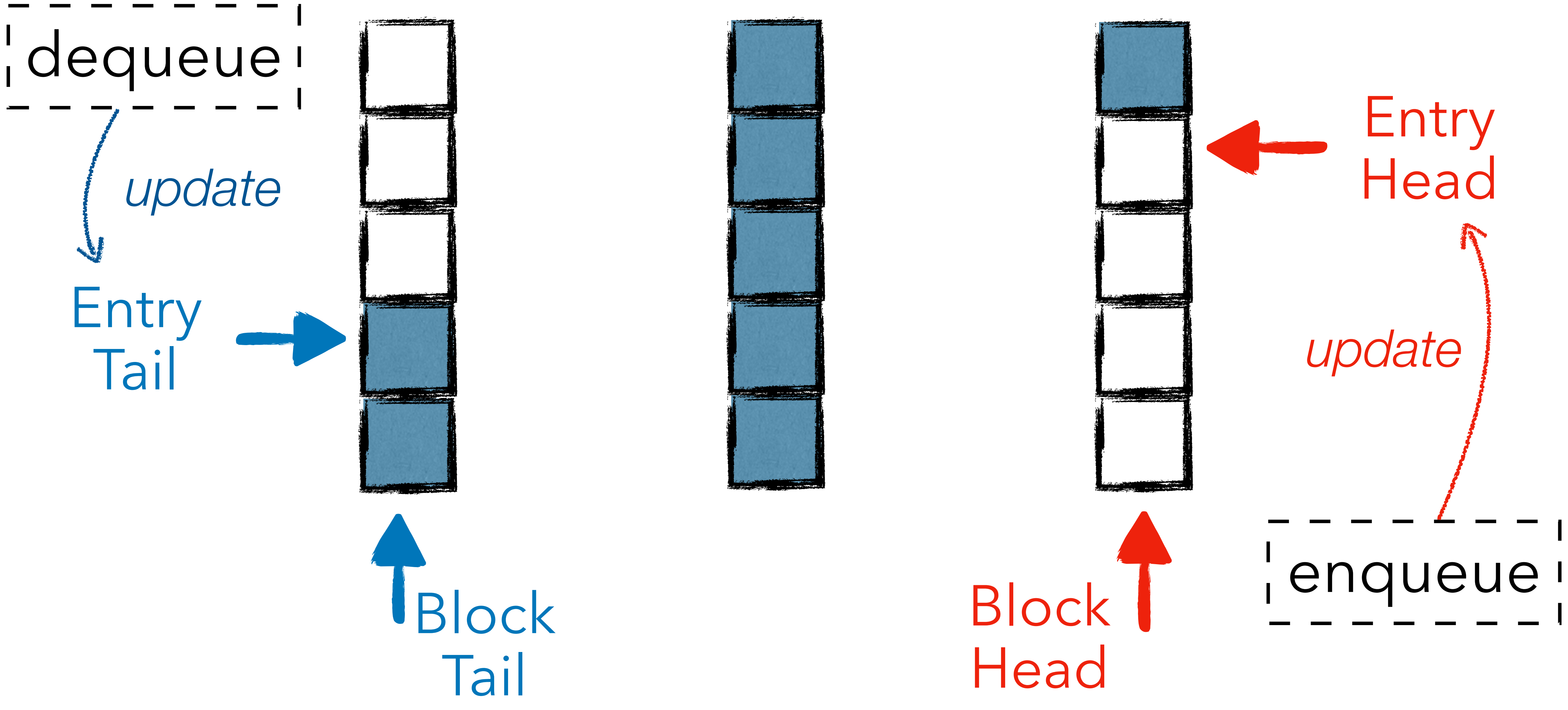
Efficient use of FAA no side effects: neither rollback nor blocking



Dealing with interferences

Enq-Deq interference

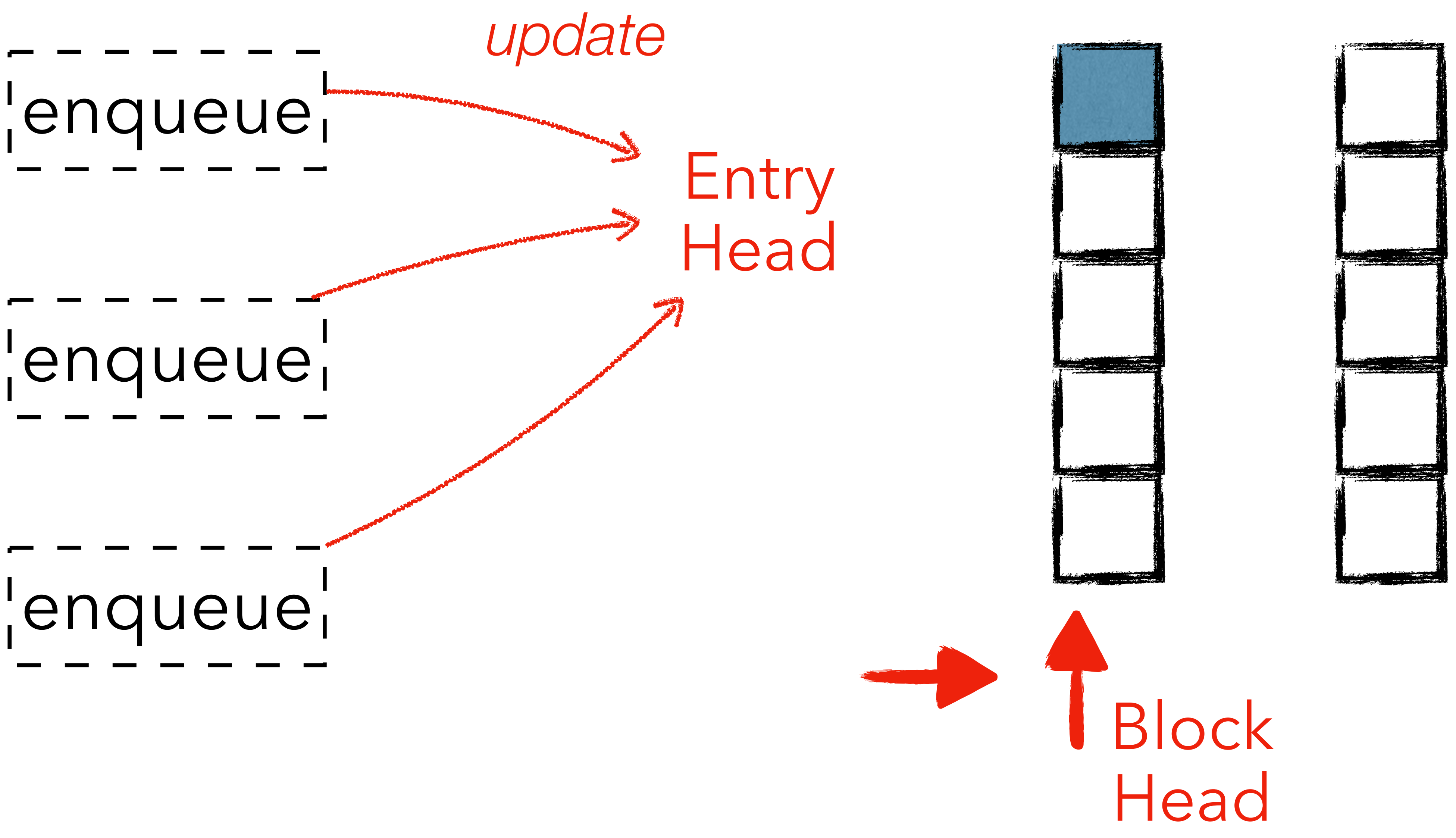
No interference when producer and consumer in different blocks



Block head and tail only read when moving to next block

Enq-Enq interference

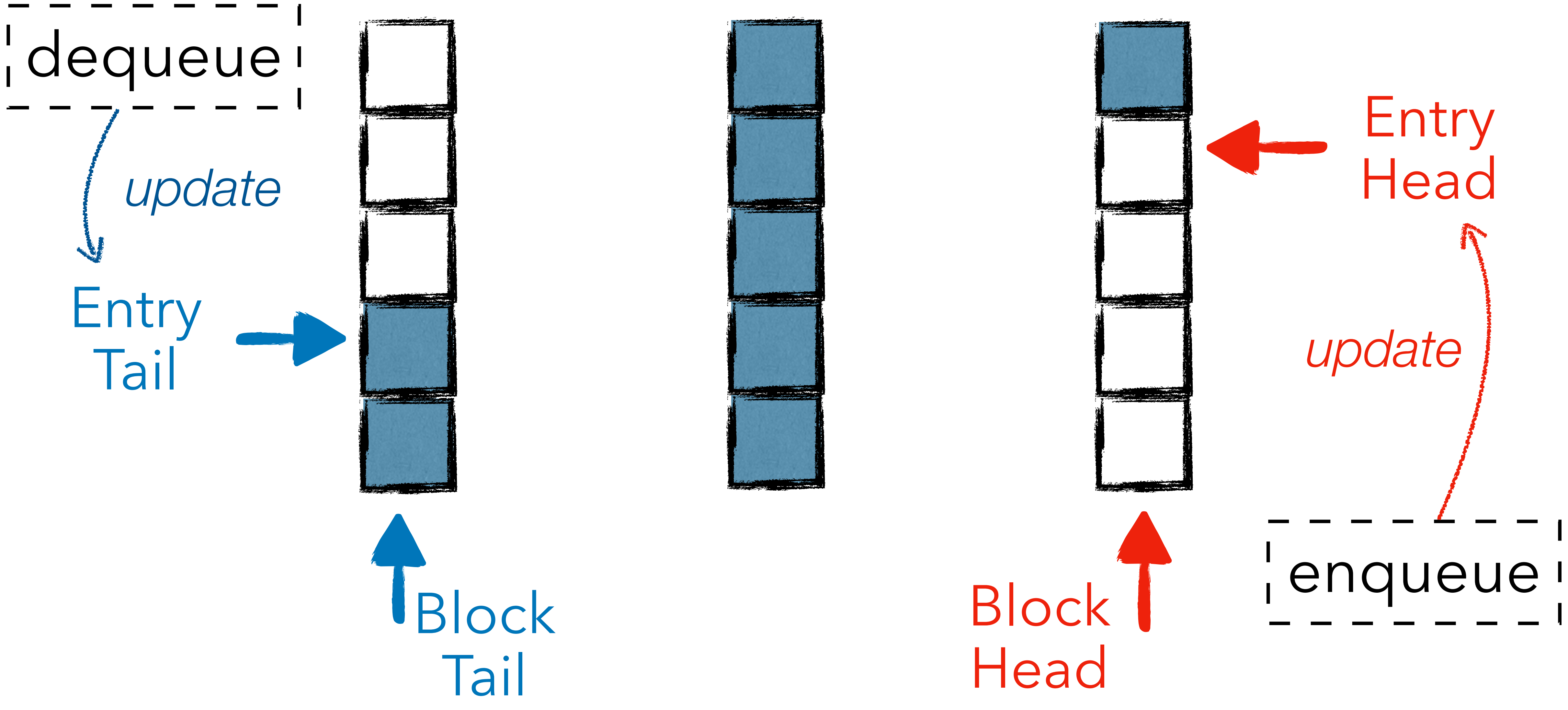
Efficient use of FAA no side effects: neither rollback nor blocking



Dealing with interferences

Enq-Deq interference

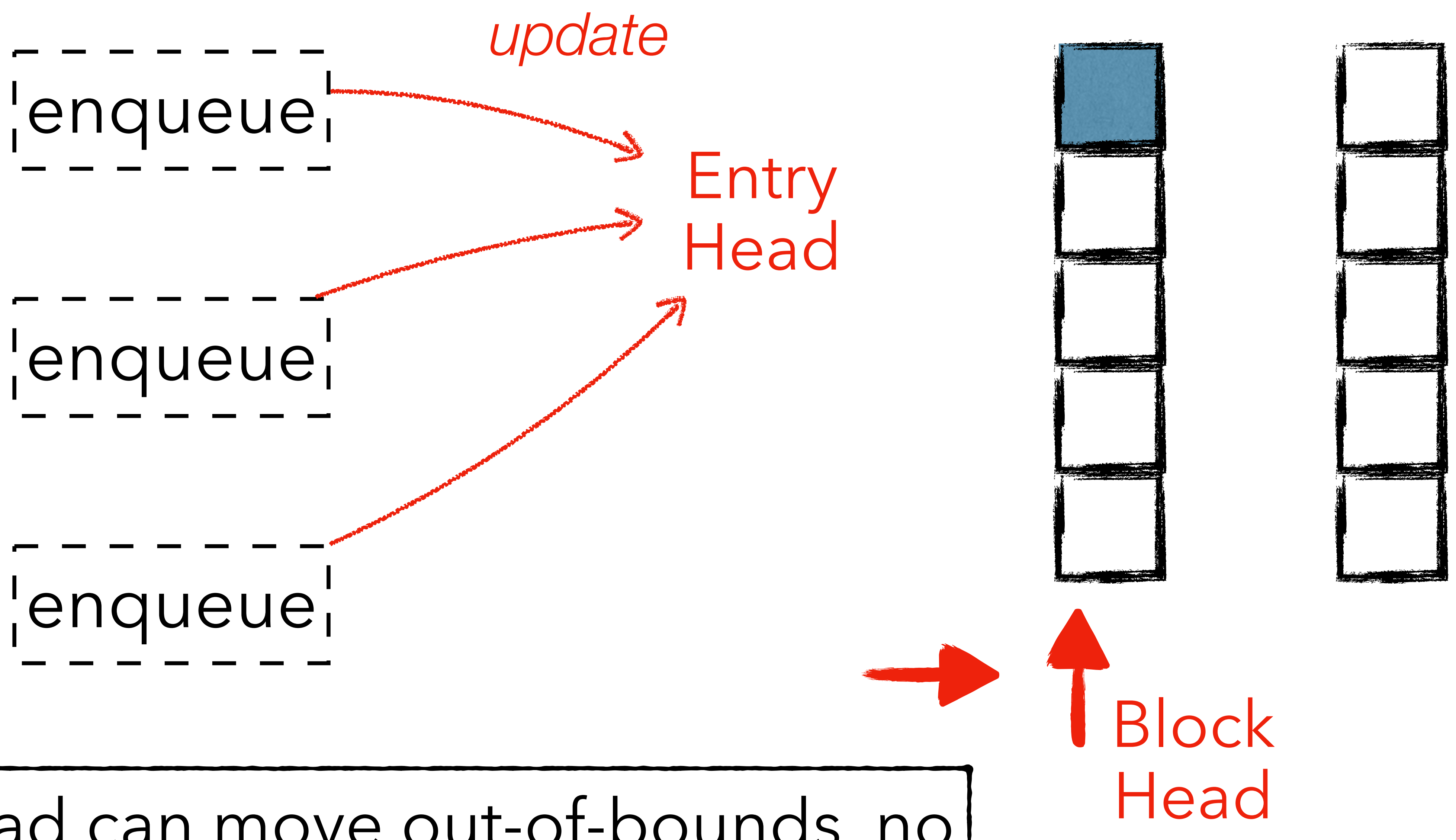
No interference when producer and consumer in different blocks



Block head and tail only read when moving to next block

Enq-Enq interference

Efficient use of FAA no side effects: neither rollback nor blocking

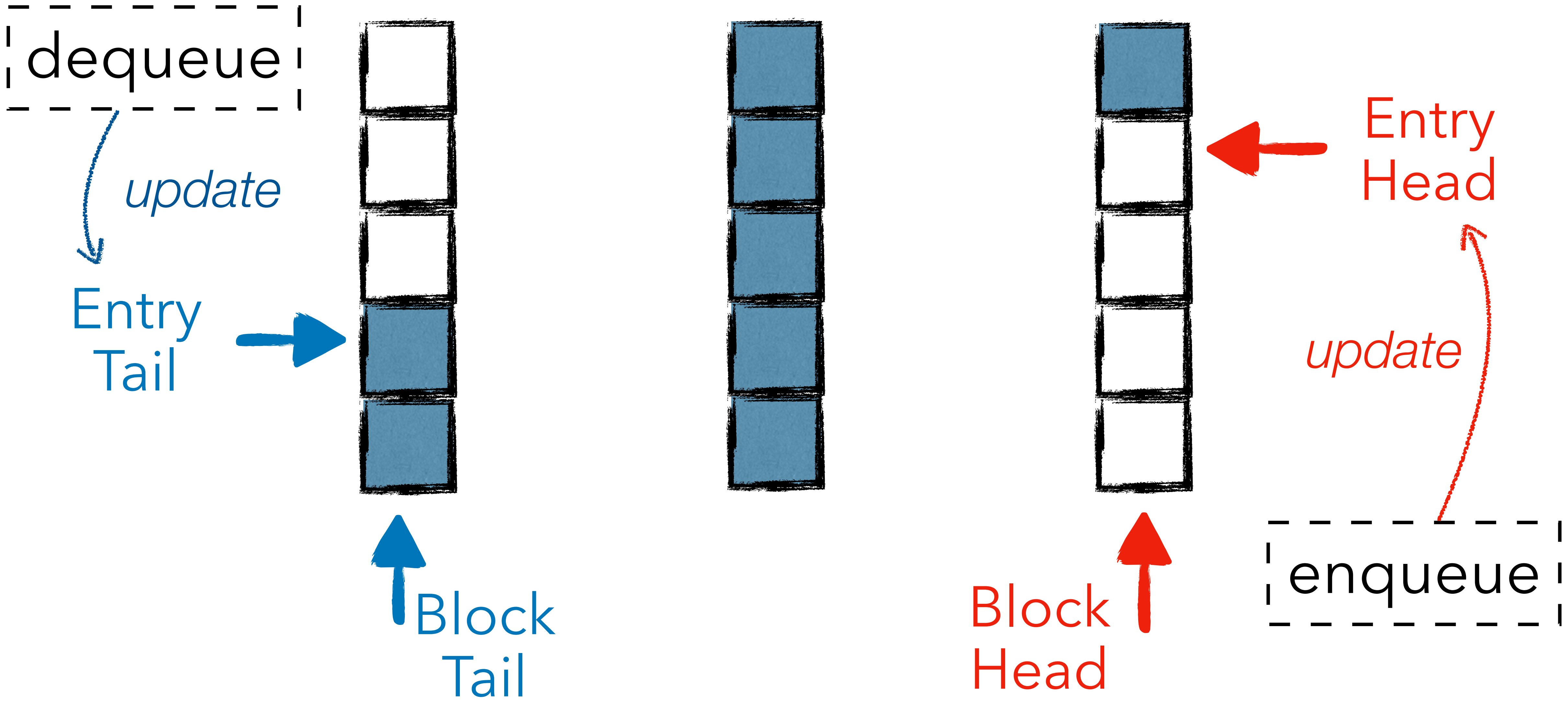


Head can move out-of-bounds, no consequence to following block.

Dealing with interferences

Enq-Deq interference

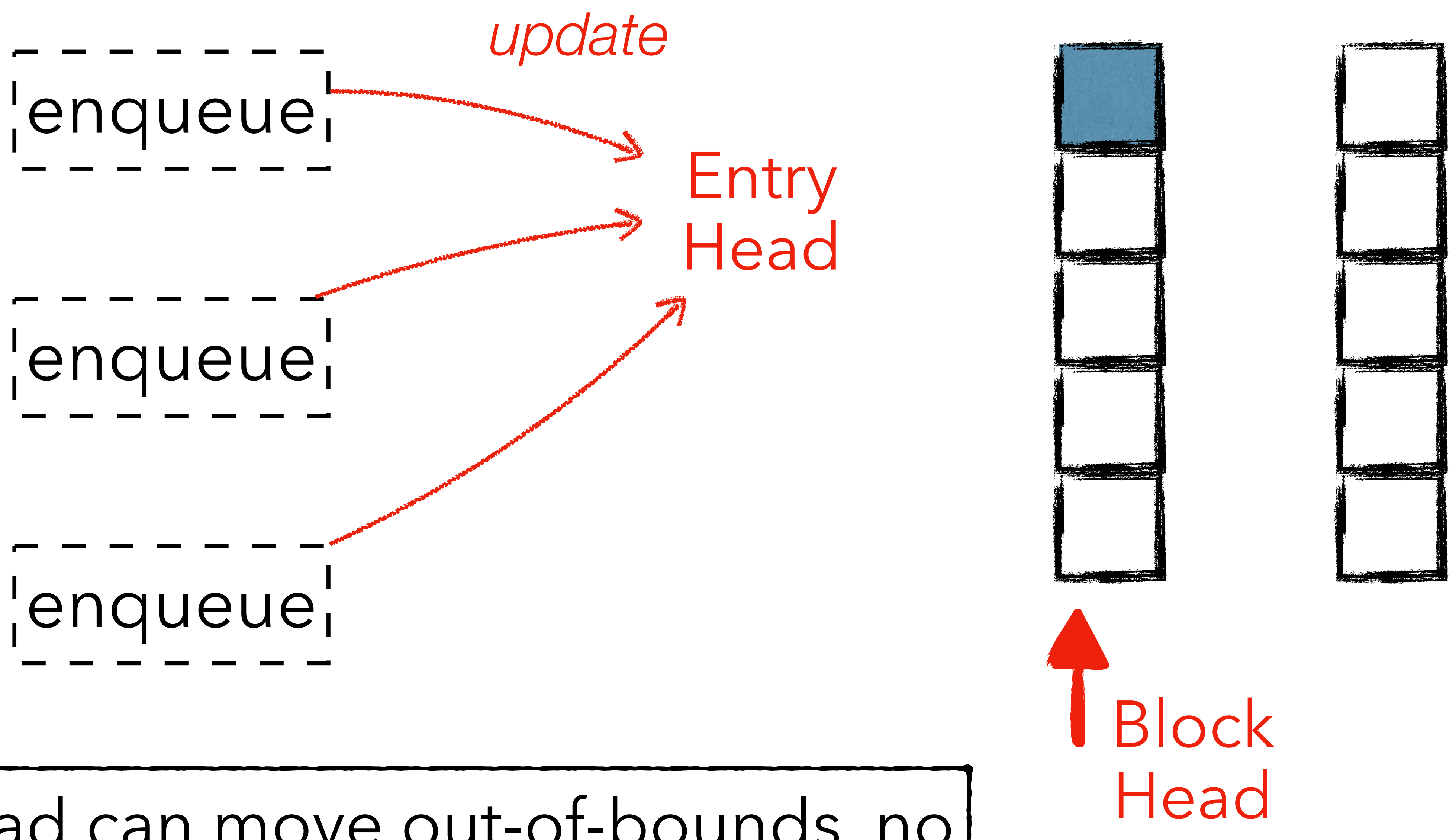
No interference when producer and consumer in different blocks



Block head and tail only read when moving to next block

Enq-Enq interference

Efficient use of FAA no side effects: neither rollback nor blocking



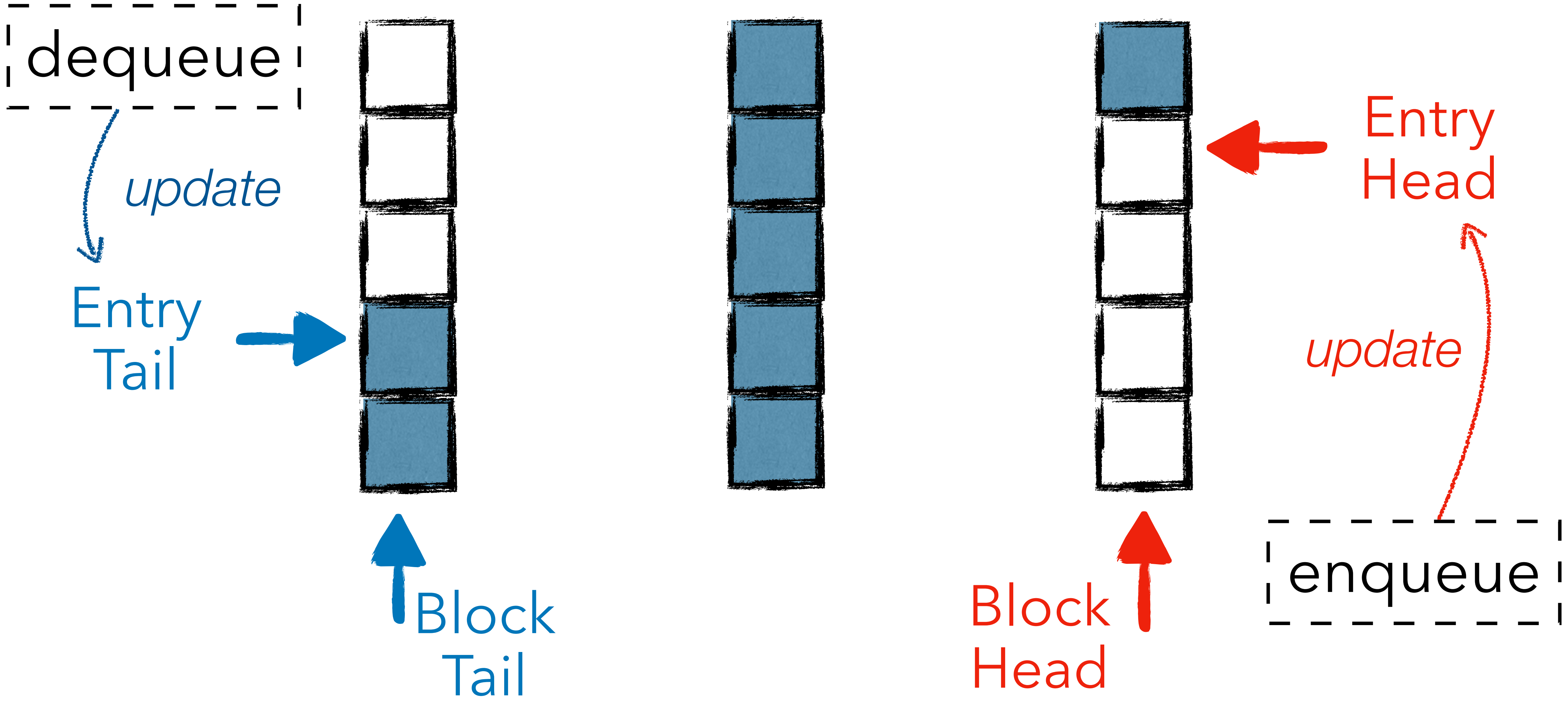
Head can move out-of-bounds, no consequence to following block.

OK →

Dealing with interferences

Enq-Deq interference

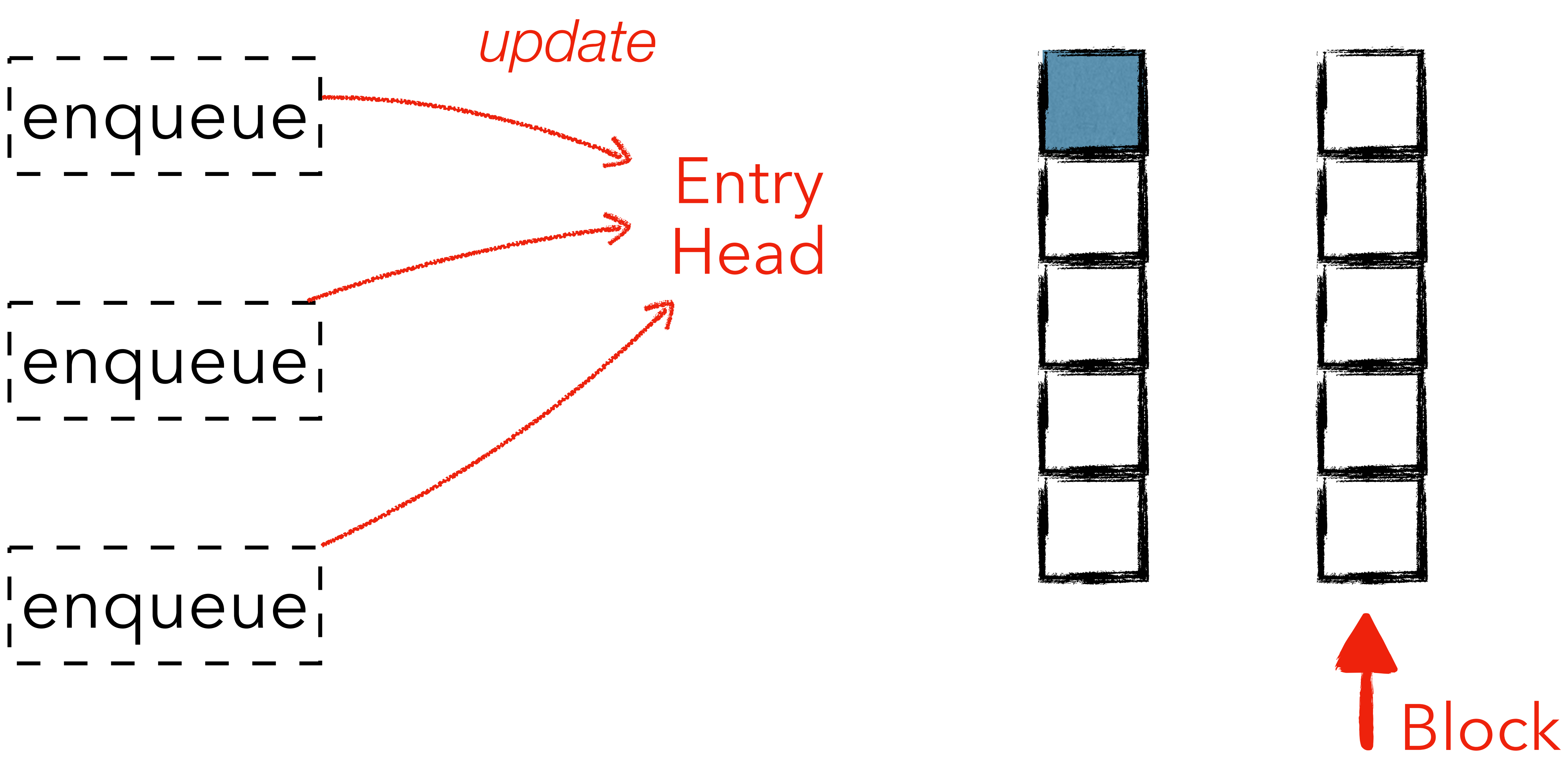
No interference when producer and consumer in different blocks



Block head and tail only read when moving to next block

Enq-Enq interference

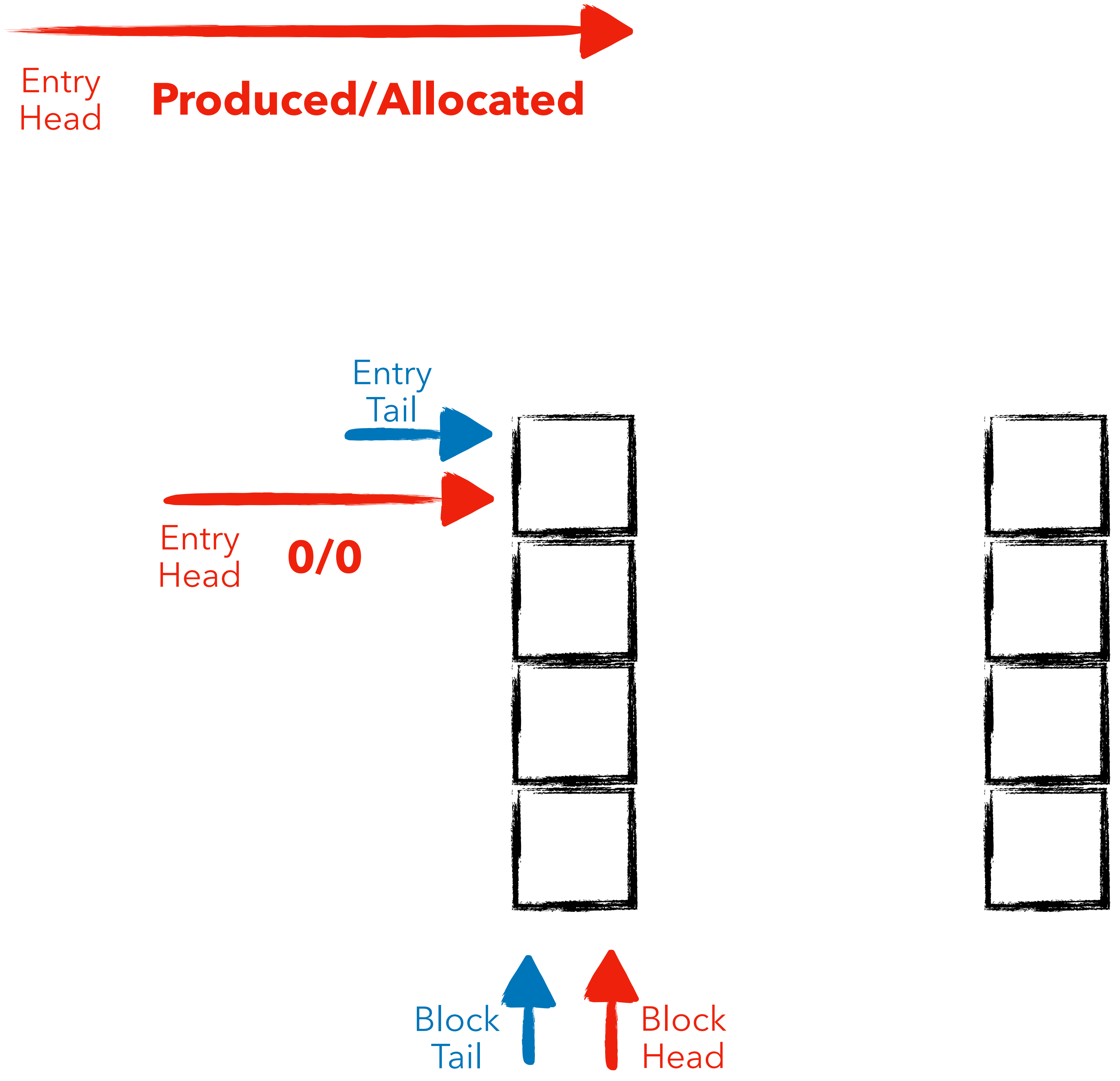
Efficient use of FAA no side effects: neither rollback nor blocking



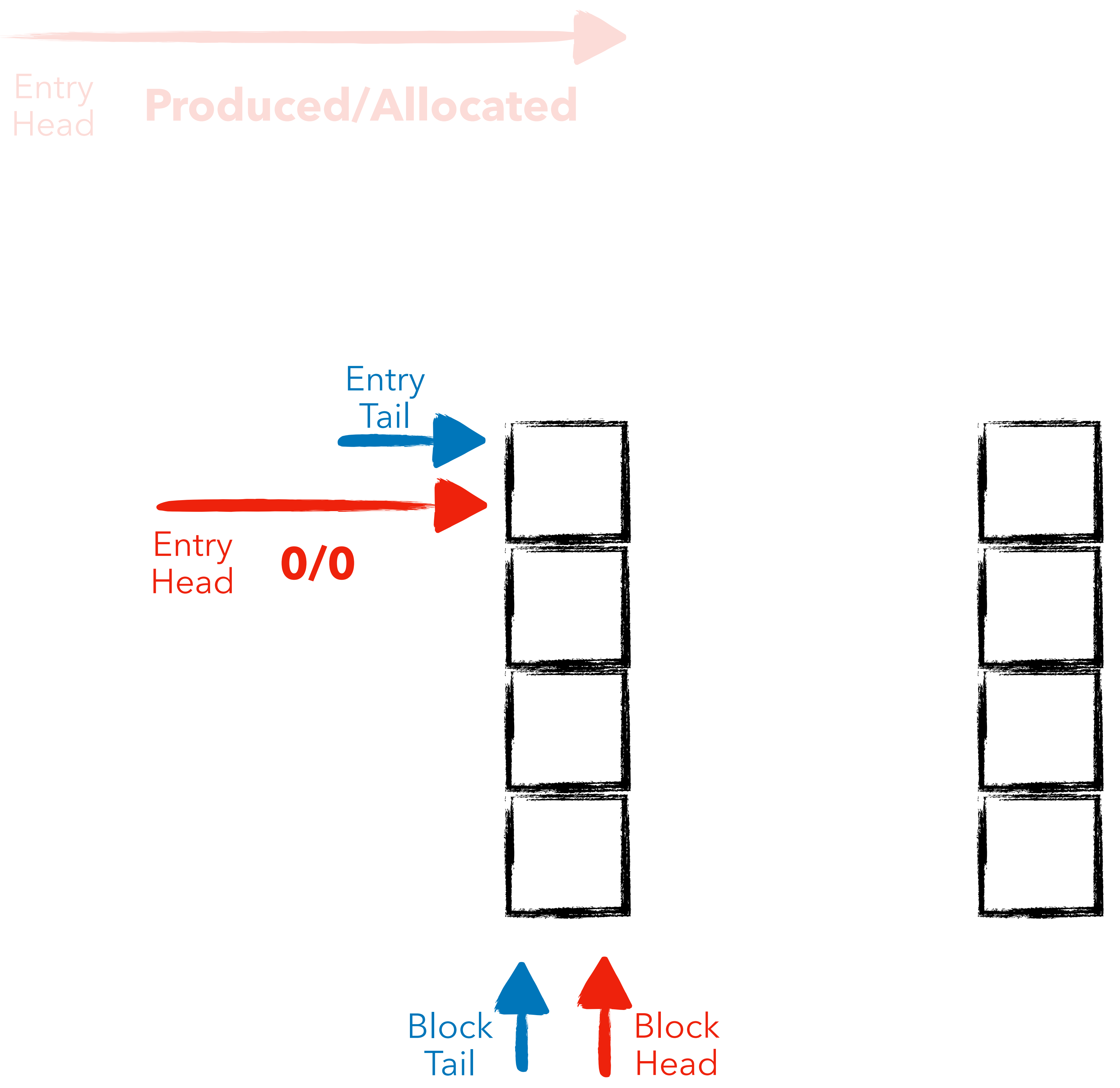
Head can move out-of-bounds, no consequence to following block.

OK →

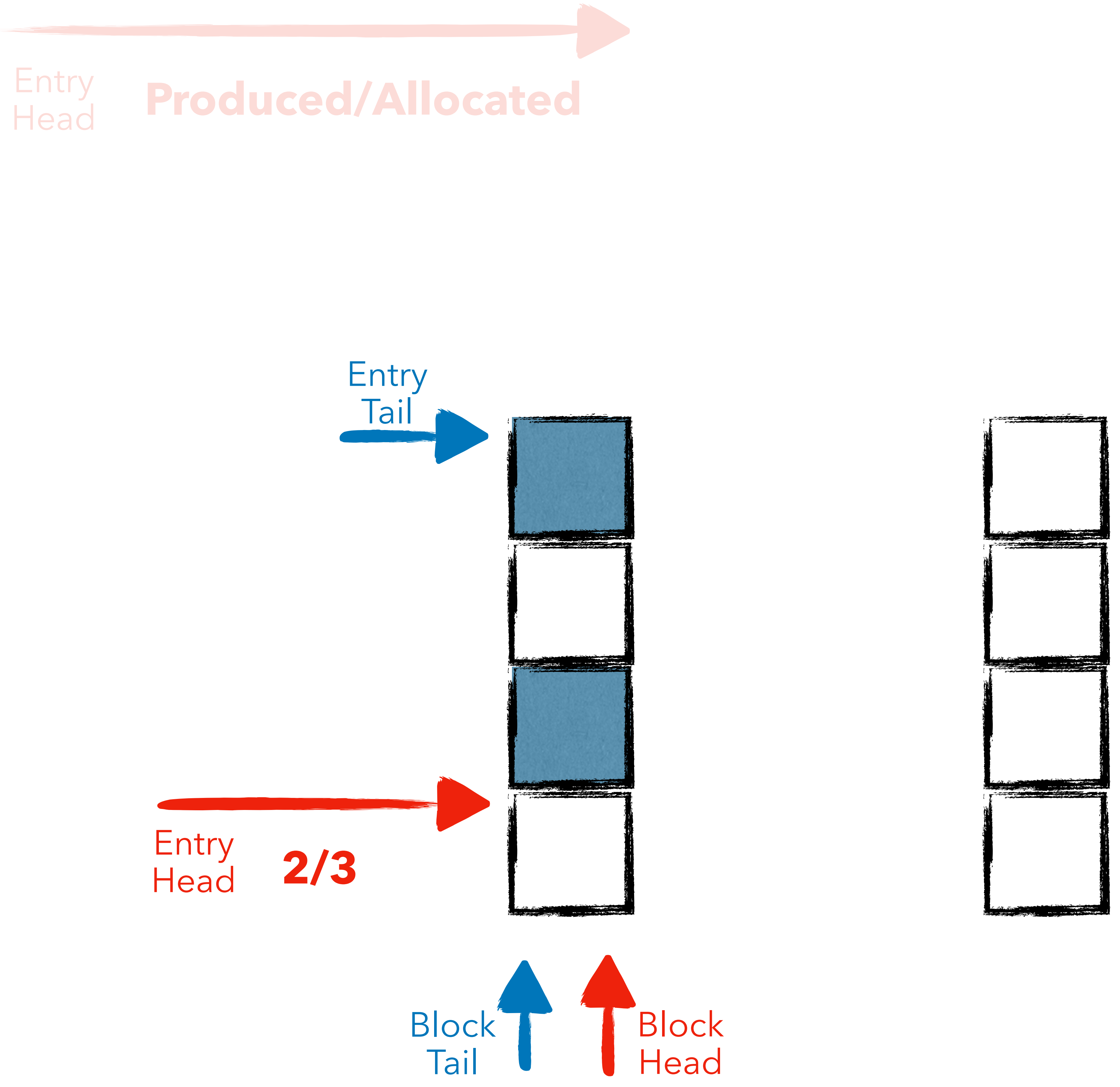
Dealing with out-of-order operations



Dealing with out-of-order operations



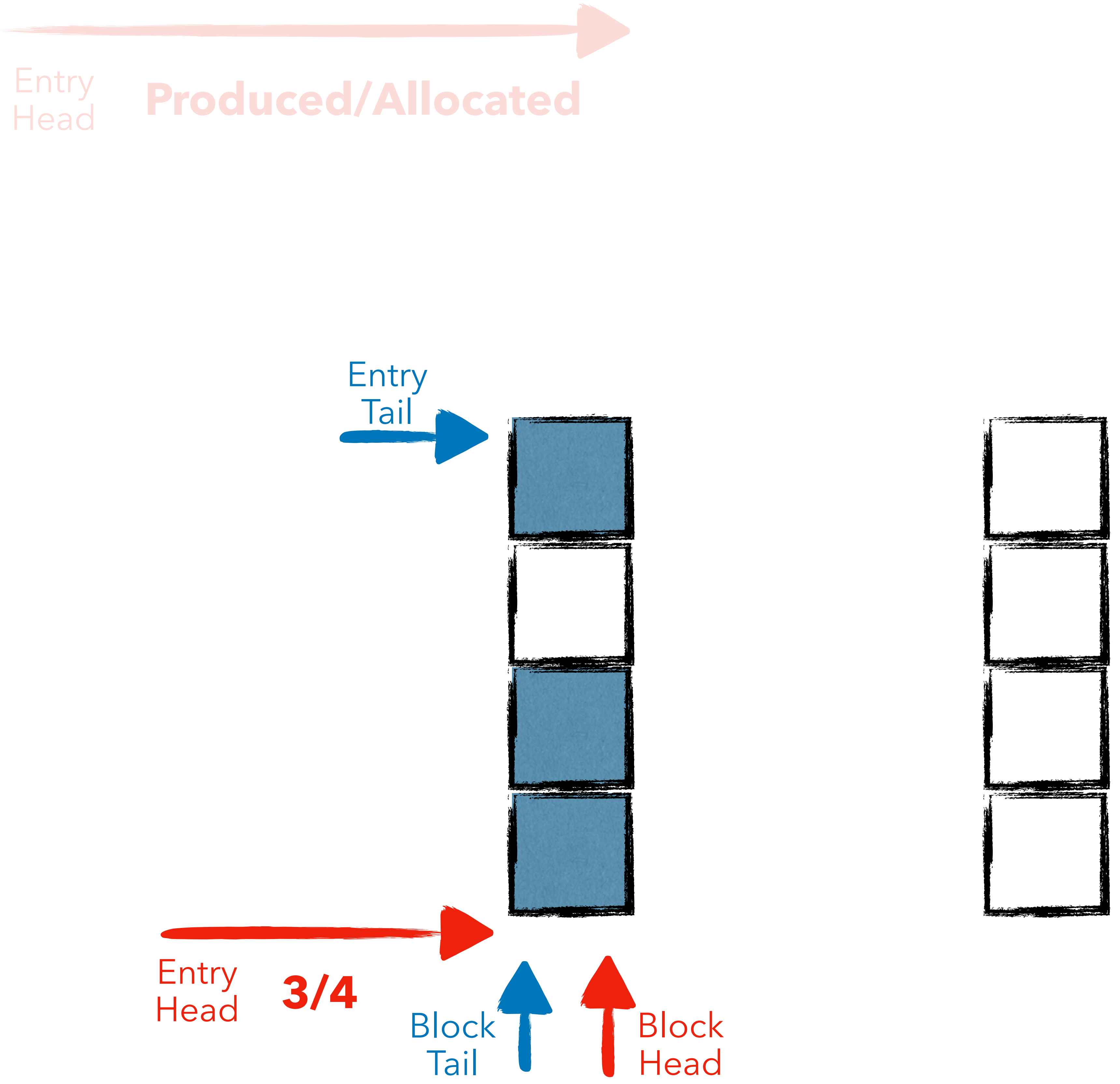
Dealing with out-of-order operations



Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

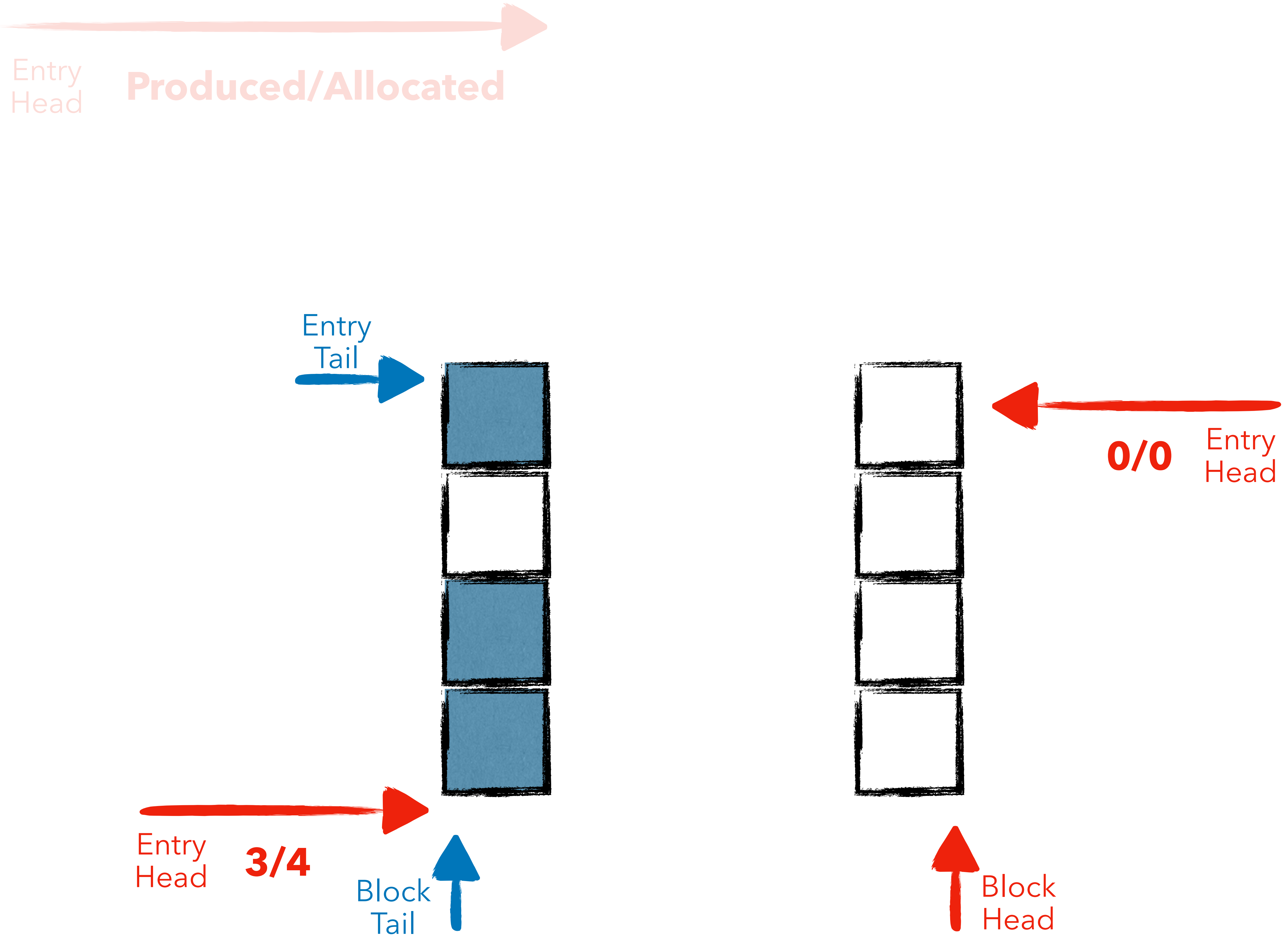
Dealing with out-of-order operations



Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

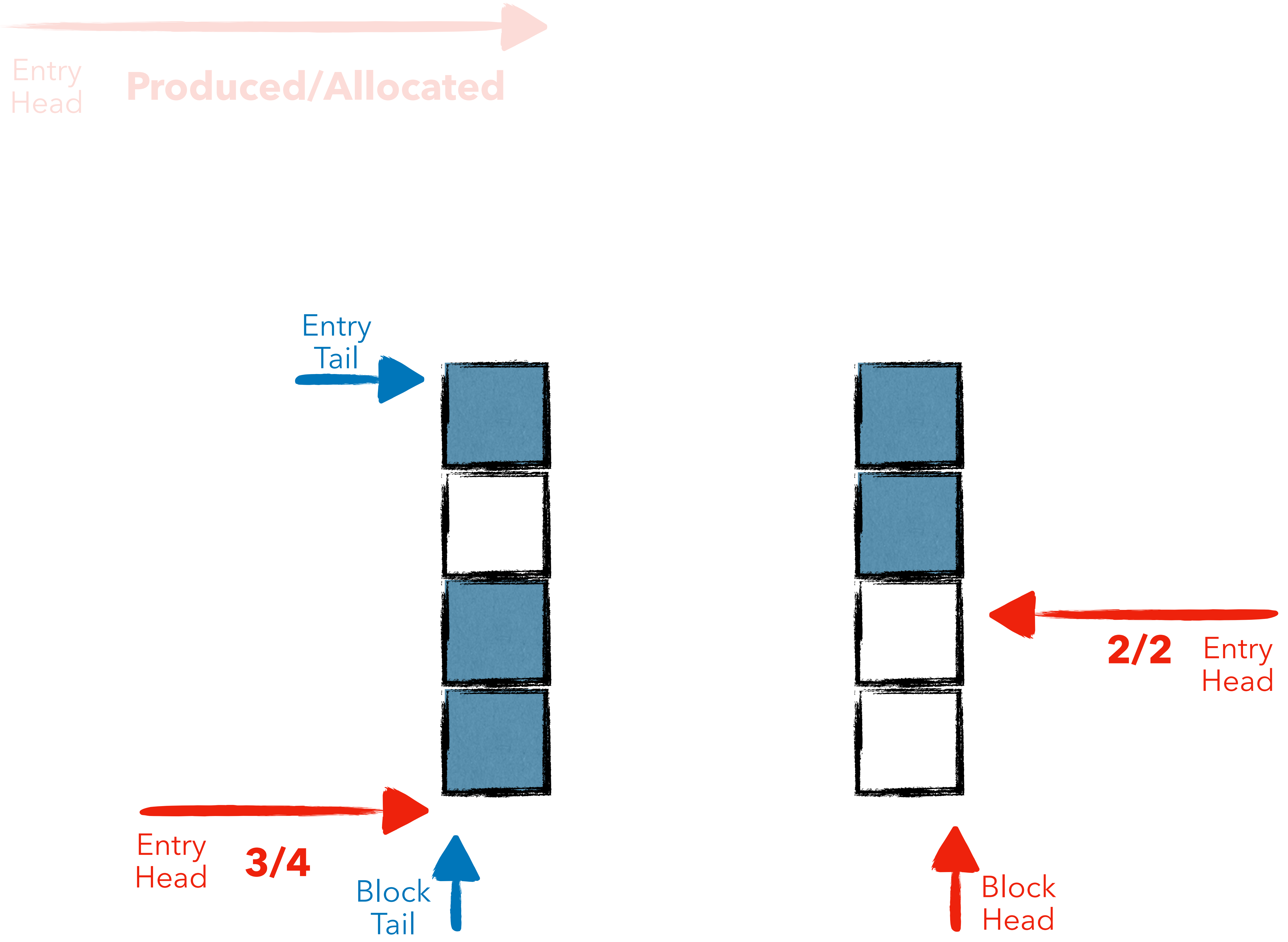
Dealing with out-of-order operations



Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

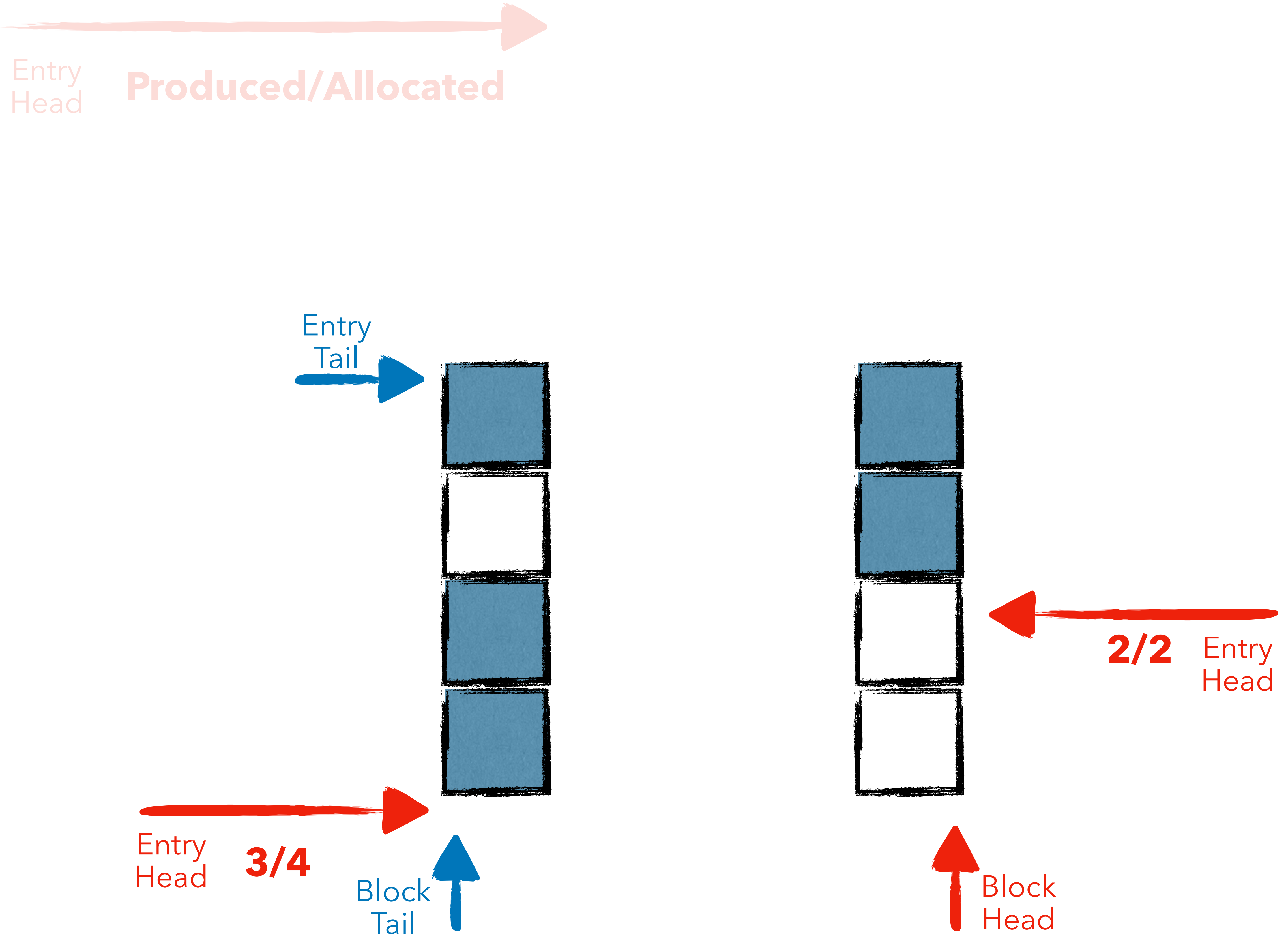
Dealing with out-of-order operations



Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

Dealing with out-of-order operations



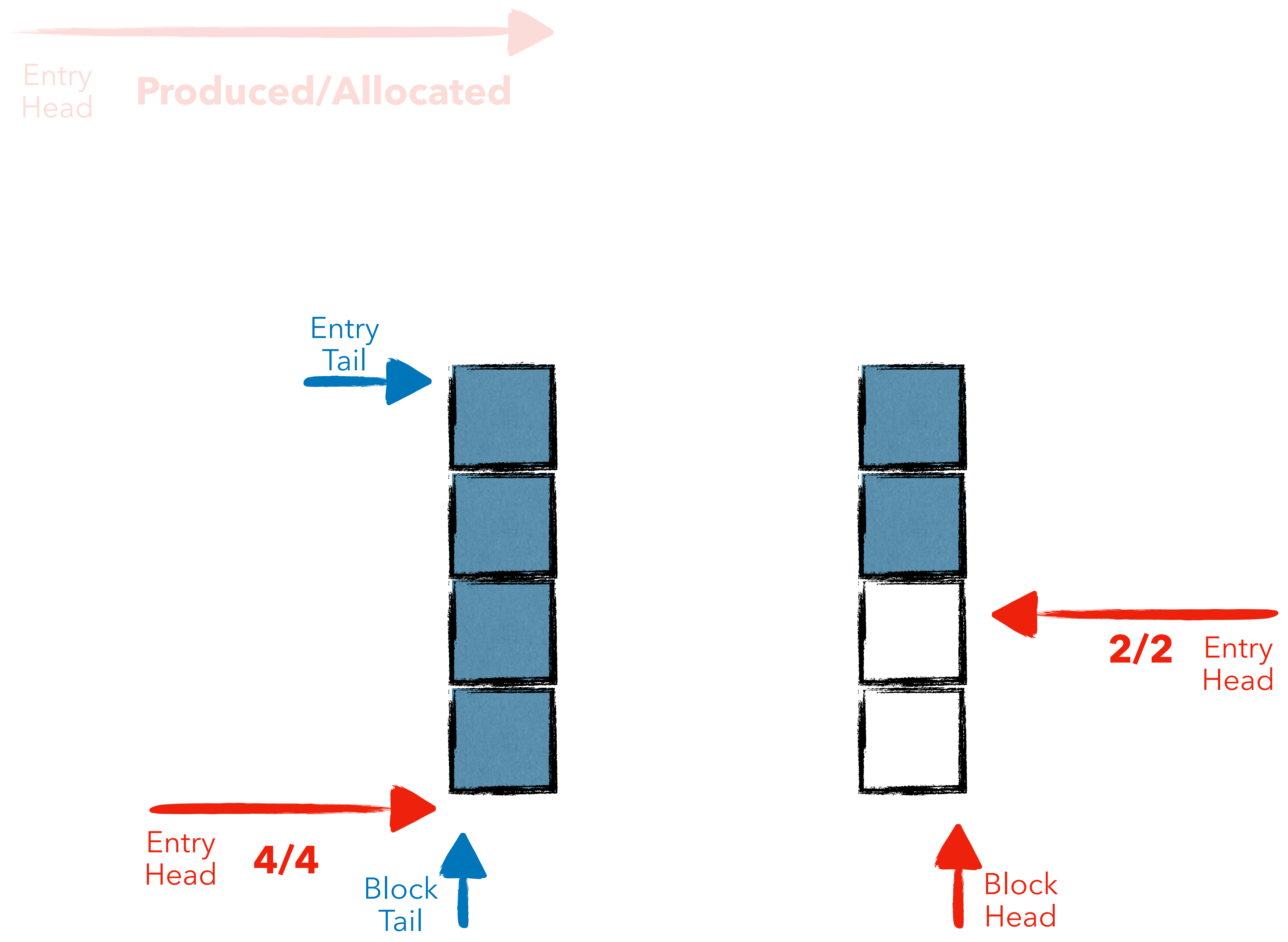
Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

Dequeue calls:

- **return BUSY if an enqueue is ongoing** in same block
- **succeed when block full** or when Produced = Allocated

Dealing with out-of-order operations



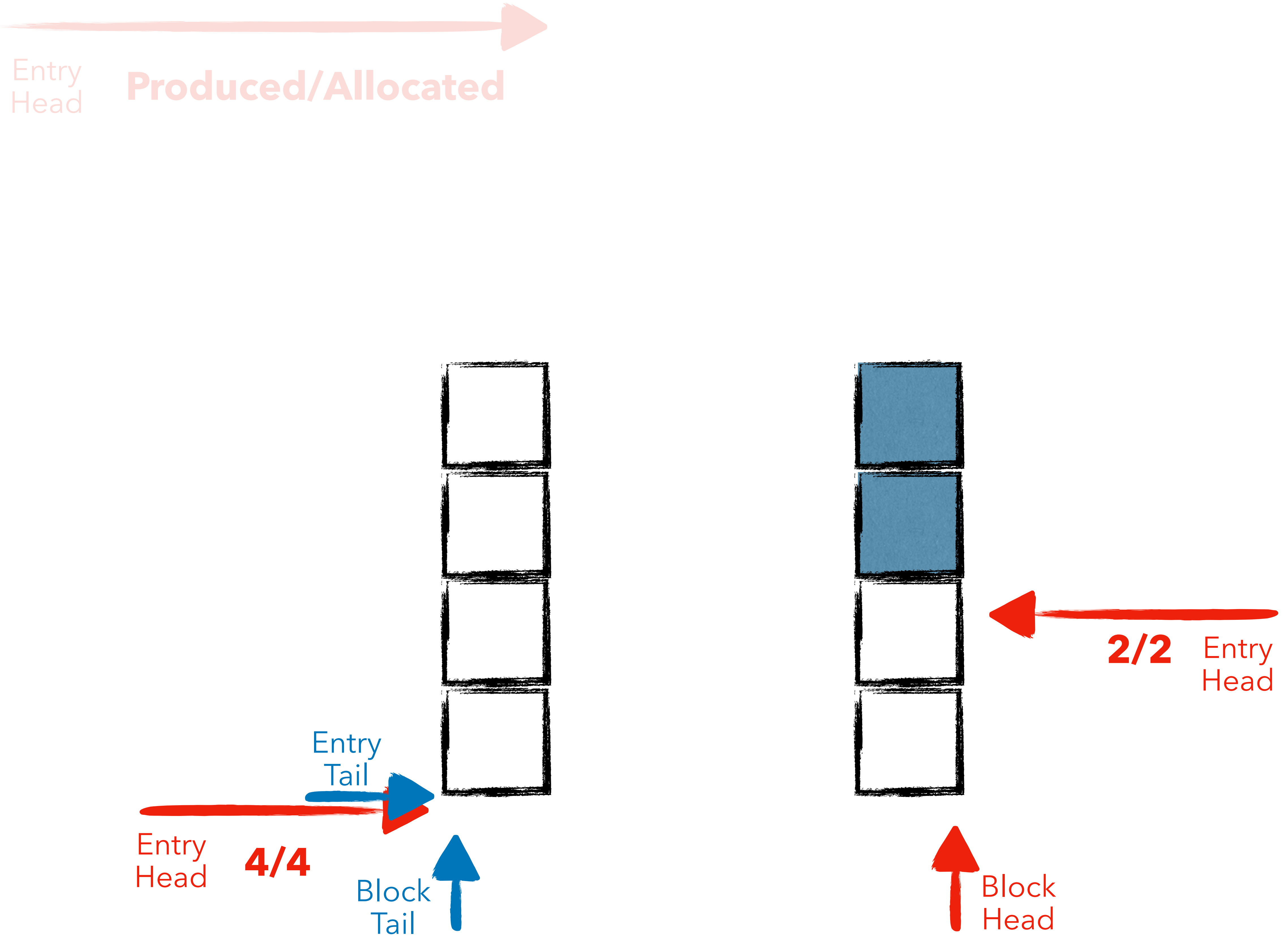
Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

Dequeue calls:

- **return BUSY if an enqueue is ongoing** in same block
- **succeed when block full** or when Produced = Allocated

Dealing with out-of-order operations



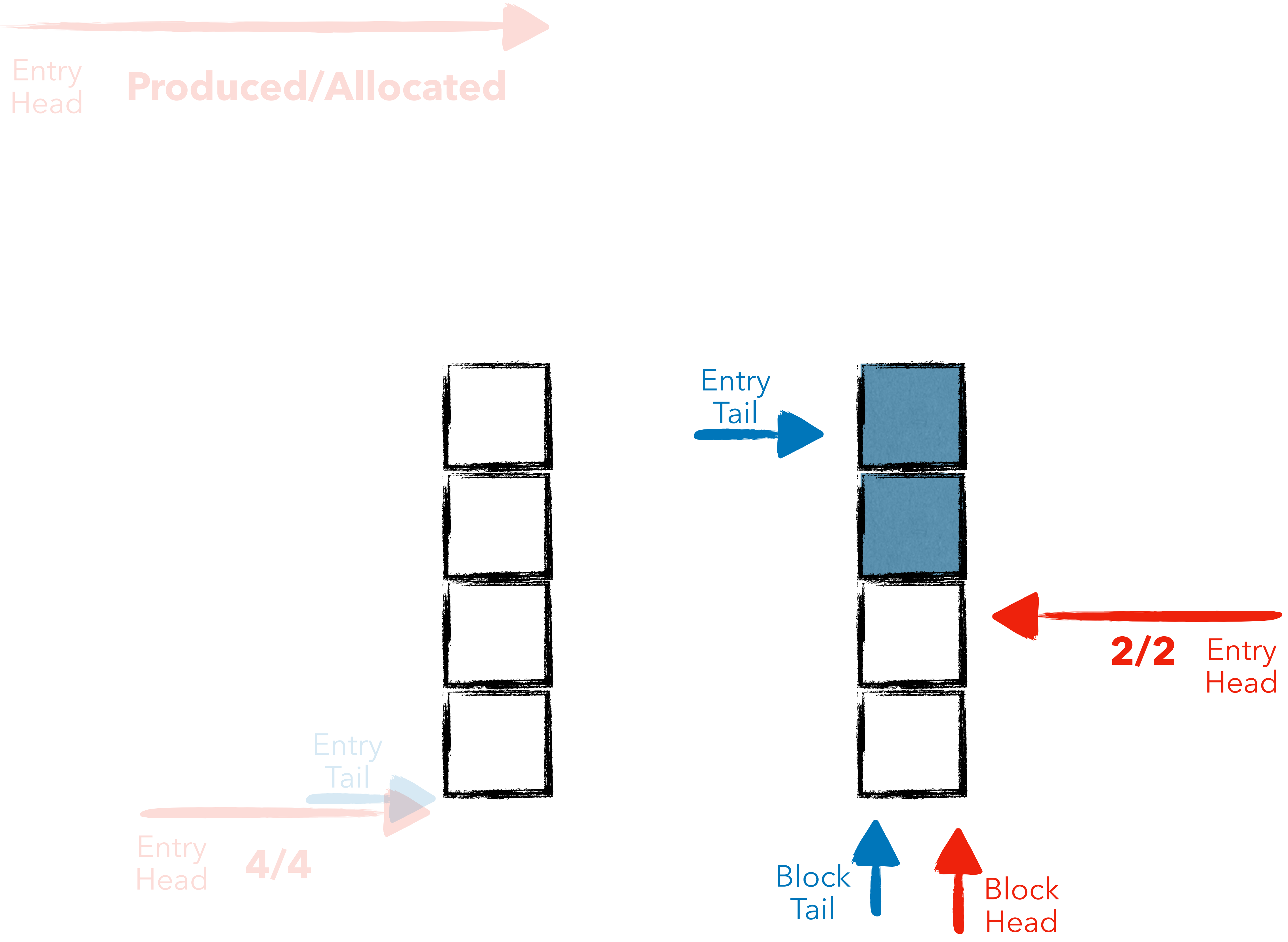
Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

Dequeue calls:

- **return BUSY if an enqueue is ongoing** in same block
- **succeed when block full** or when Produced = Allocated

Dealing with out-of-order operations



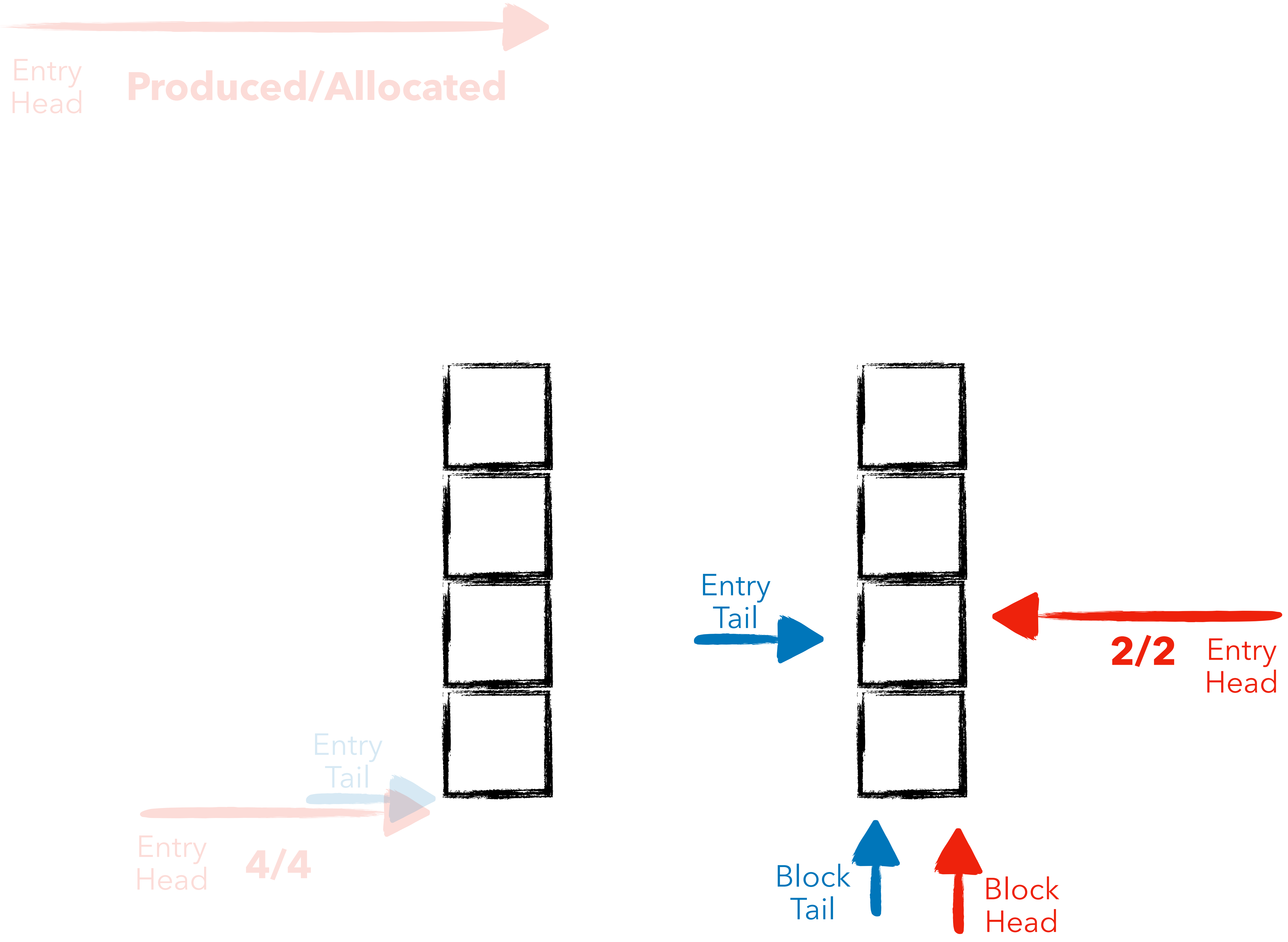
Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

Dequeue calls:

- **return BUSY if an enqueue is ongoing** in same block
- **succeed when block full** or when Produced = Allocated

Dealing with out-of-order operations



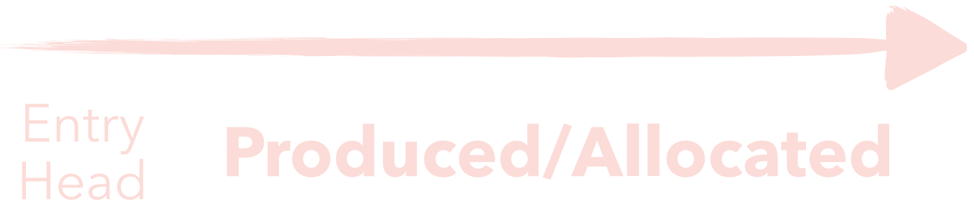
Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

Dequeue calls:

- **return BUSY if an enqueue is ongoing** in same block
- **succeed when block full** or when Produced = Allocated

Dealing with out-of-order operations



Many cool tricks in the paper:

- update block and entry indices at the same time without D-CAS
- Avoid ABA issues with versioning
- Cache block indices for speed

Enqueue calls:

- **do not wait for others** in same block to complete
- can **move to next block** even if current block has ongoing enqueues

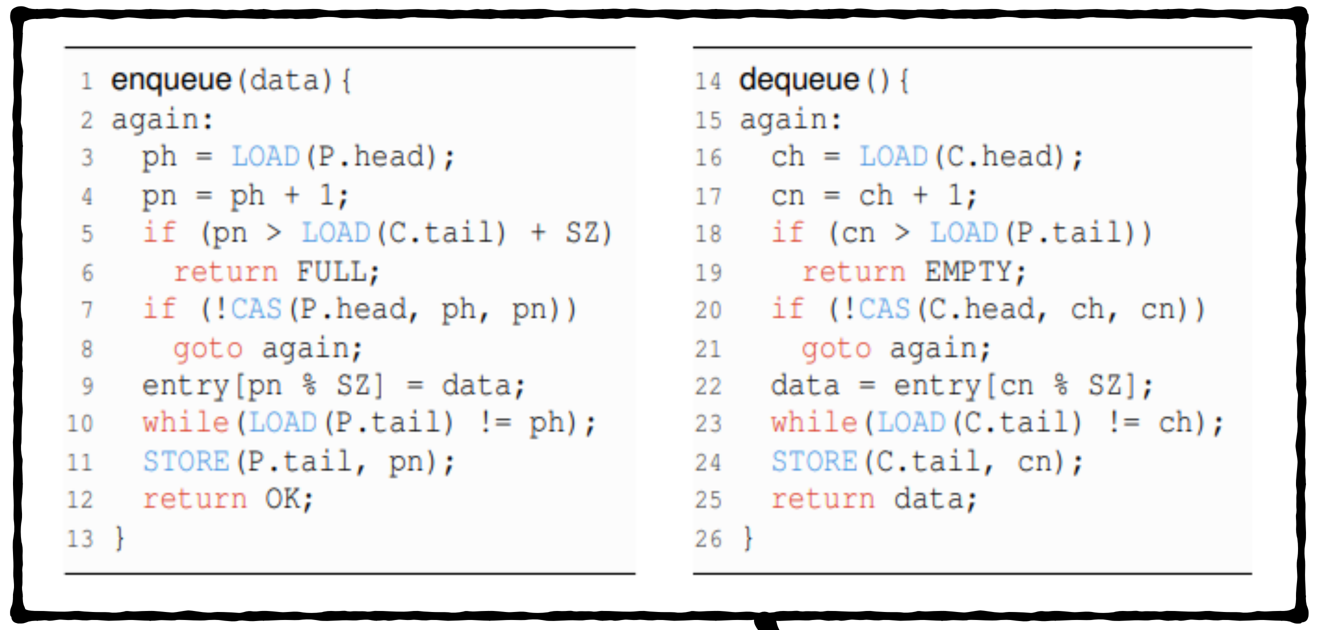
Dequeue calls:

- **return BUSY if an enqueue is ongoing** in same block
- **succeed when block full** or when Produced = Allocated

Correctness on WMMs with **practical** verification

DPDK-like algorithm

~10 atomics



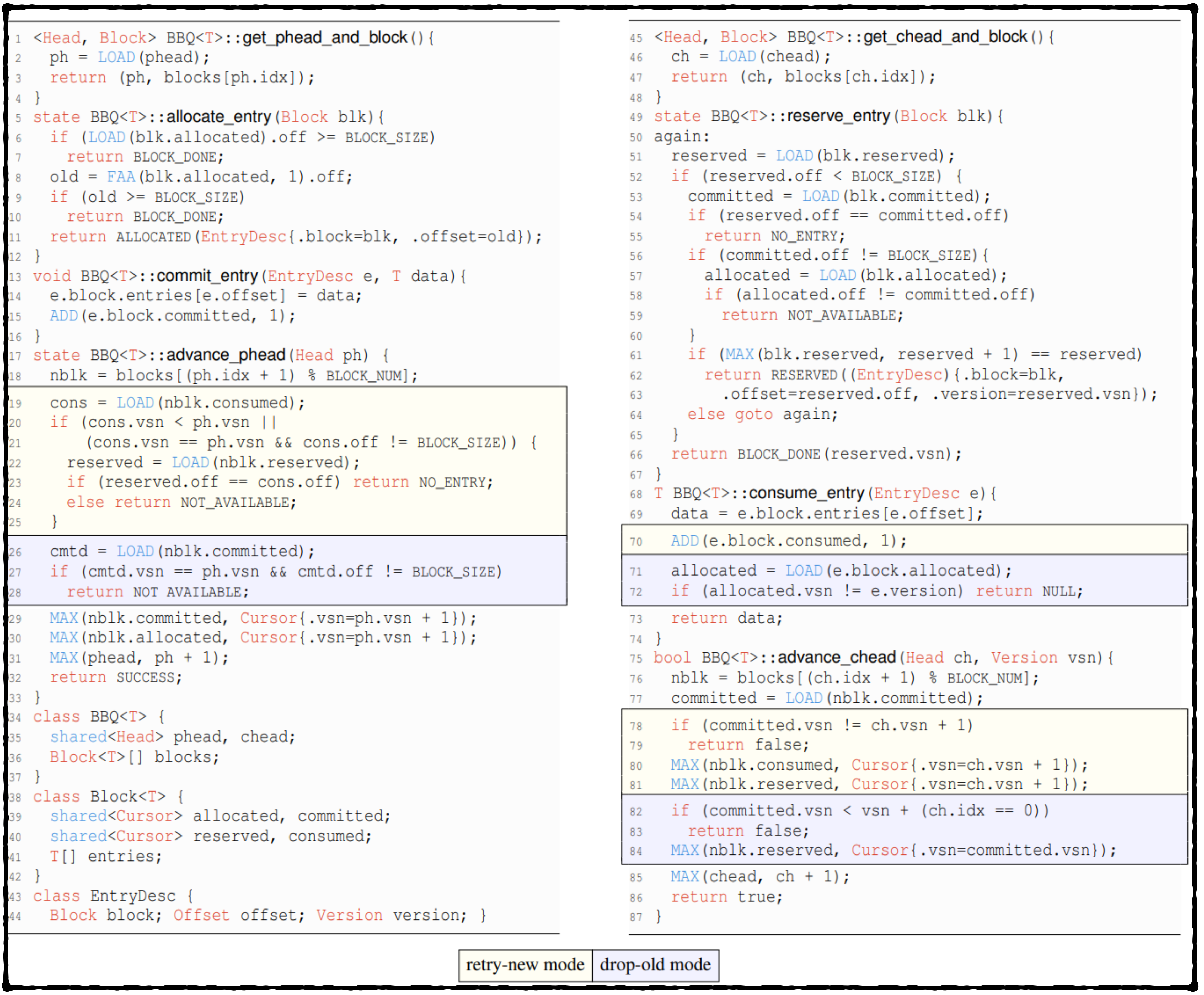
```
enqueue(data){
again:
  ph = LOAD(P.head);
  pn = ph + 1;
  if (pn > LOAD(C.tail) + SZ)
    return FULL;
  if (!CAS(P.head, ph, pn))
    goto again;
  entry[pn % SZ] = data;
  while(LOAD(P.tail) != ph);
  STORE(P.tail, pn);
  return OK;
}
```

```
dequeue(){
again:
  ch = LOAD(C.head);
  cn = ch + 1;
  if (cn > LOAD(P.tail))
    return EMPTY;
  if (!CAS(C.head, ch, cn))
    goto again;
  data = entry[cn % SZ];
  while(LOAD(C.tail) != ch);
  STORE(C.tail, cn);
  return data;
}
```

BBQ is not easy to digest

Part of BBQ

More than 20 atomics



```
<Head, Block> BBQ<T>::get_phead_and_block(){
  ph = LOAD(phead);
  return (ph, blocks[ph.idx]);
}
state BBQ<T>::allocate_entry(Block blk){
  if (LOAD(blk.allocated).off >= BLOCK_SIZE)
    return BLOCK_DONE;
  old = FAA(blk.allocated, 1).off;
  if (old >= BLOCK_SIZE)
    return BLOCK_DONE;
  return ALLOCATED(EntryDesc{.block=blk, .offset=old});
}
void BBQ<T>::commit_entry(EntryDesc e, T data){
  e.block.entries[e.offset] = data;
  ADD(e.block.committed, 1);
}
state BBQ<T>::advance_phead(Head ph) {
  nblk = blocks[(ph.idx + 1) % BLOCK_NUM];
  cons = LOAD(nblk.consumed);
  if (cons.vsn < ph.vsn ||
      (cons.vsn == ph.vsn && cons.off != BLOCK_SIZE)) {
    reserved = LOAD(nblk.reserved);
    if (reserved.off == cons.off) return NO_ENTRY;
    else return NOT_AVAILABLE;
  }
  cmtd = LOAD(nblk.committed);
  if (cmtd.vsn == ph.vsn && cmtd.off != BLOCK_SIZE)
    return NOT AVAILABLE;
  MAX(nblk.committed, Cursor{.vsn=ph.vsn + 1});
  MAX(nblk.allocated, Cursor{.vsn=ph.vsn + 1});
  MAX(phead, ph + 1);
  return SUCCESS;
}
class BBQ<T> {
  shared<Head> phead, chead;
  Block<T>[] blocks;
}
class Block<T> {
  shared<Cursor> allocated, committed;
  shared<Cursor> reserved, consumed;
  T[] entries;
}
class EntryDesc {
  Block block; Offset offset; Version version; }
```

```
<Head, Block> BBQ<T>::get_chead_and_block(){
  ch = LOAD(chead);
  return (ch, blocks[ch.idx]);
}
state BBQ<T>::reserve_entry(Block blk){
again:
  reserved = LOAD(blk.reserved);
  if (reserved.off < BLOCK_SIZE) {
    committed = LOAD(blk.committed);
    if (reserved.off == committed.off)
      return NO_ENTRY;
    if (committed.off != BLOCK_SIZE){
      allocated = LOAD(blk.allocated);
      if (allocated.off != committed.off)
        return NOT_AVAILABLE;
    }
    if (MAX(blk.reserved, reserved + 1) == reserved)
      return RESERVED((EntryDesc){.block=blk,
        .offset=reserved.off, .version=reserved.vsn});
    else goto again;
  }
  return BLOCK_DONE(reserved.vsn);
}
T BBQ<T>::consume_entry(EntryDesc e){
  data = e.block.entries[e.offset];
  ADD(e.block.consumed, 1);
  allocated = LOAD(e.block.allocated);
  if (allocated.vsn != e.version) return NULL;
  return data;
}
bool BBQ<T>::advance_chead(Head ch, Version vsn){
  nblk = blocks[(ch.idx + 1) % BLOCK_NUM];
  committed = LOAD(nblk.committed);
  if (committed.vsn != ch.vsn + 1)
    return false;
  MAX(nblk.consumed, Cursor{.vsn=ch.vsn + 1});
  MAX(nblk.reserved, Cursor{.vsn=ch.vsn + 1});
  if (committed.vsn < vsn + (ch.idx == 0))
    return false;
  MAX(nblk.reserved, Cursor{.vsn=committed.vsn});
  MAX(chead, ch + 1);
  return true;
}
```

retry-new mode	drop-old mode

Correctness on WMMs with **practical** verification

- Long stress testing
  by engineers
- Identification of corner cases
  by WMM experts and engineers
- Only a few corner cases necessary
  queue full/empty, FIFO, wrap-around

- Model check corner cases on WMM
  by engineers
- 3 bugs found model checking them
  Not found while stress testing
- Reproducible on real hardware
  Test cases were built in retrospect

Agenda

☑ Motivation

☑ Stories and Challenges

Interference, Out-of-order operations, Correctness on WMMs

☑ Block-based Bounded Queue (BBQ)

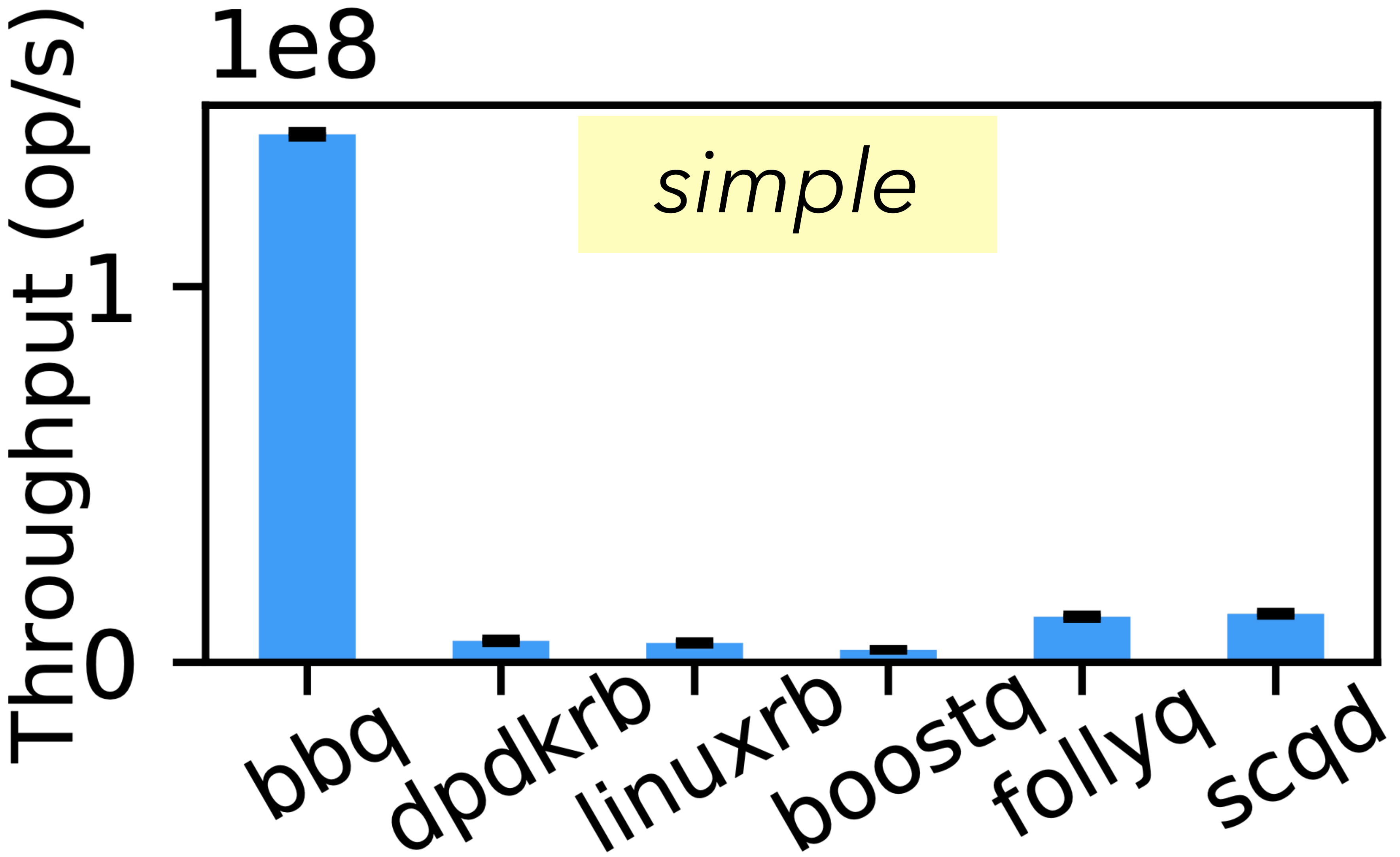
☑ Insights to Tackle the Challenges

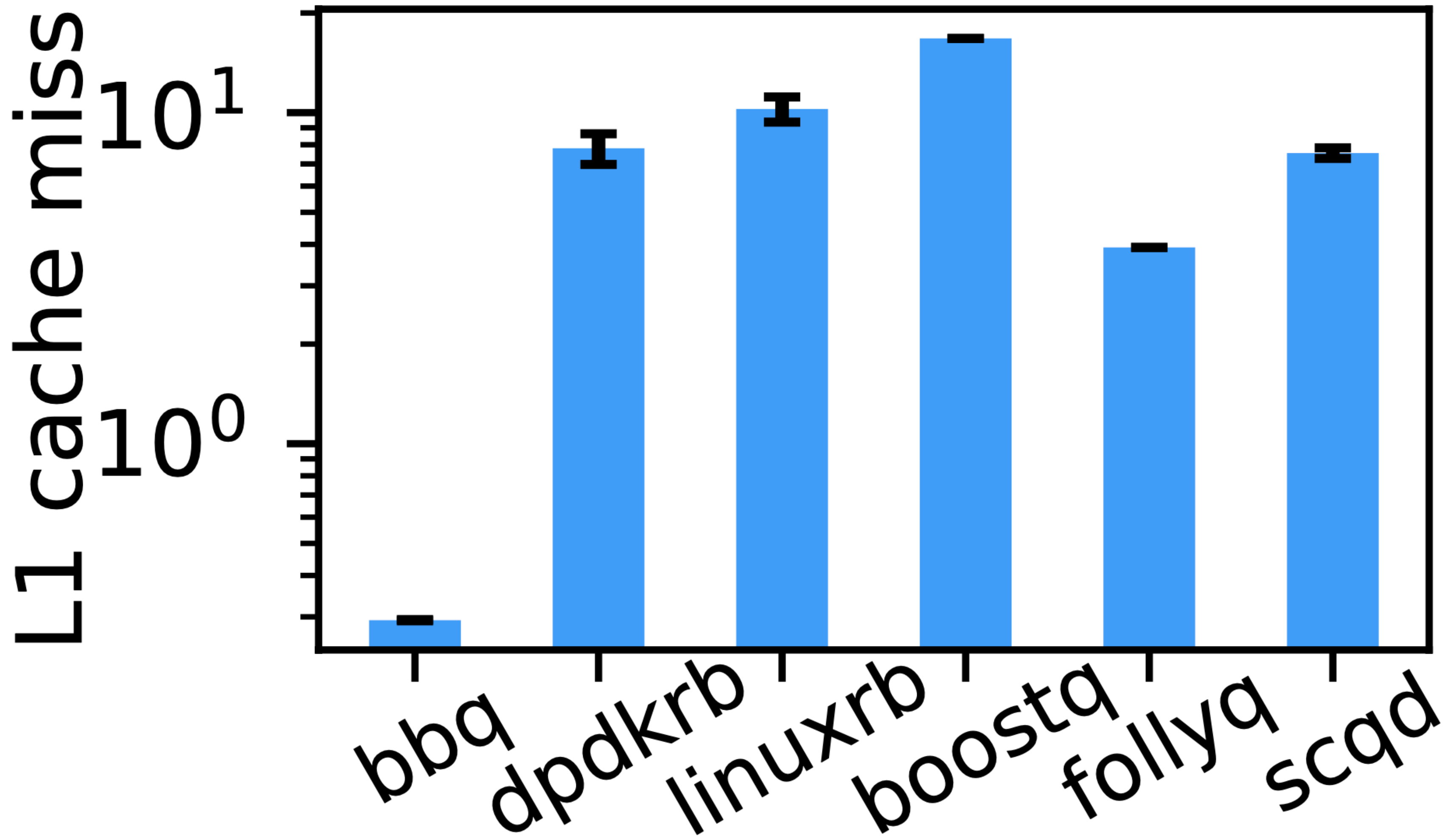
□ Selected Evaluation Results

Micro-benchmark Results – SPSC

Compared against 5 state-of-the-art bounded queues

- x86 machines with 88 hyper-threads
- 8 bytes data size, 32*k* bytes memory usage
- *simple:* 11.3x to 42.4x higher throughput

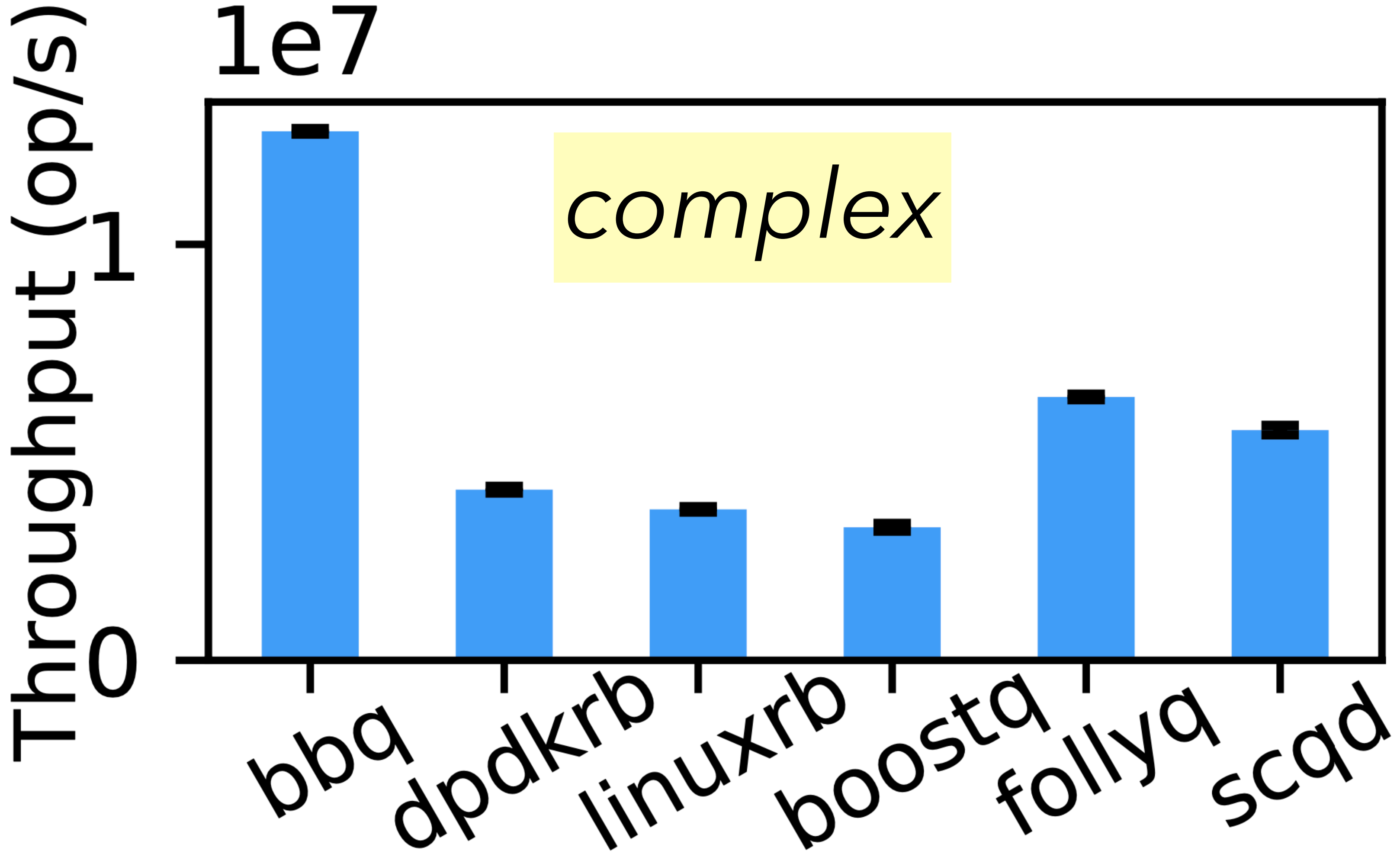


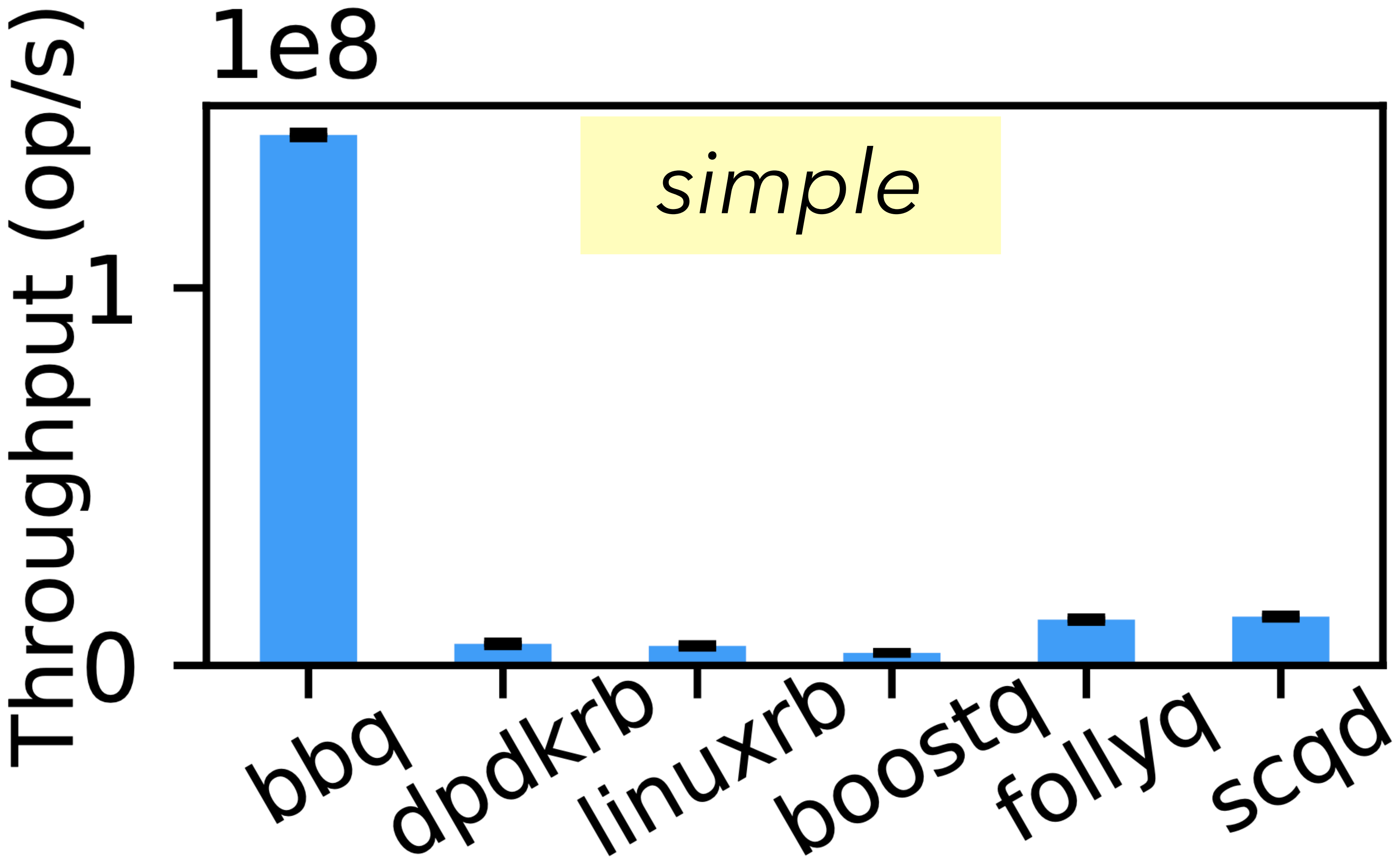


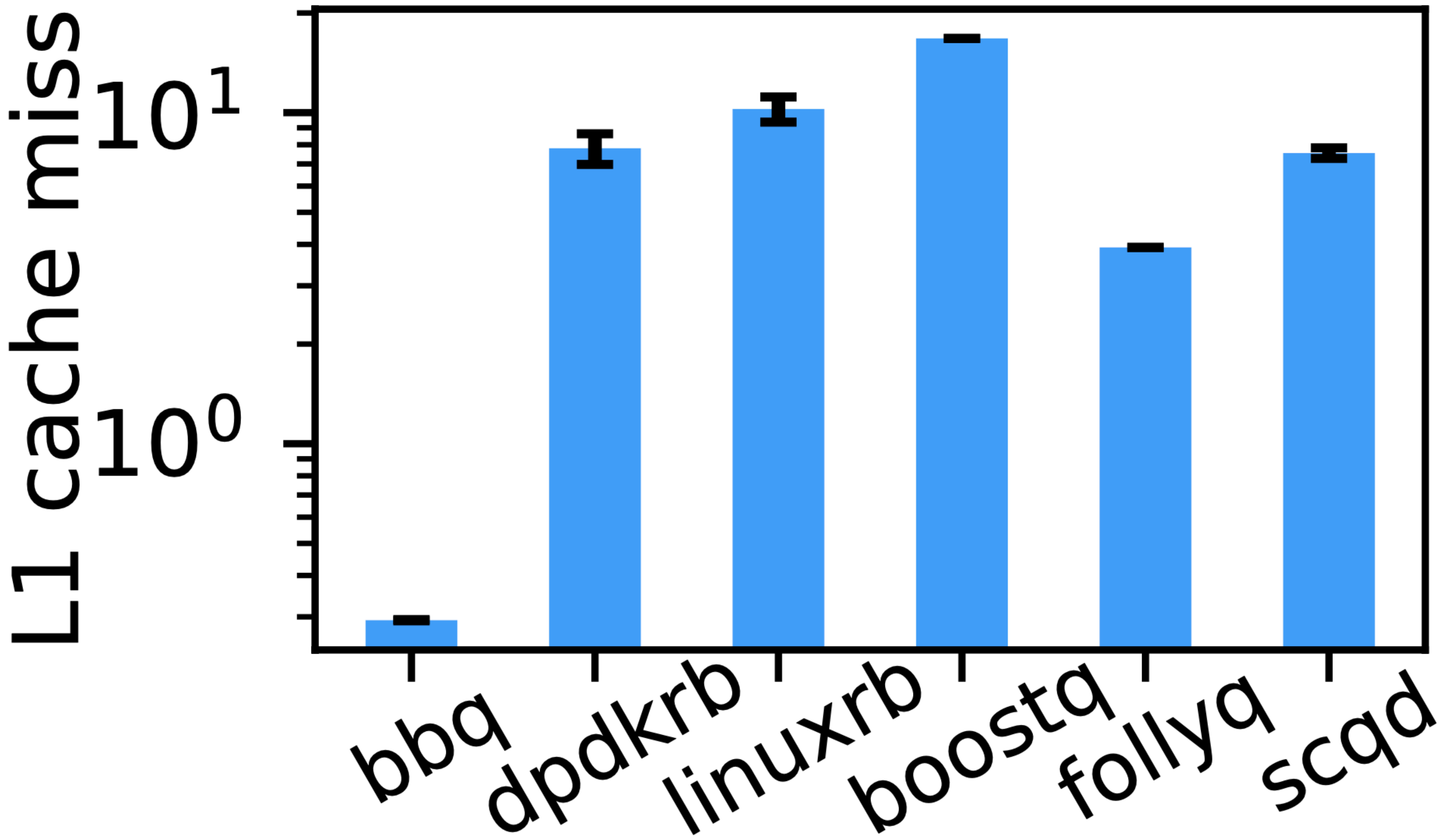
Micro-benchmark Results – SPSC

Compared against 5 state-of-the-art bounded queues

- x86 machines with 88 hyper-threads
- 8 bytes data size, 32*k* bytes memory usage
- *simple*: 11.3x to 42.4x higher throughput
- *complex*: **at least 2x higher than FollyQ**







DPDK Test Suite (DTS) – Multiprocess benchmark

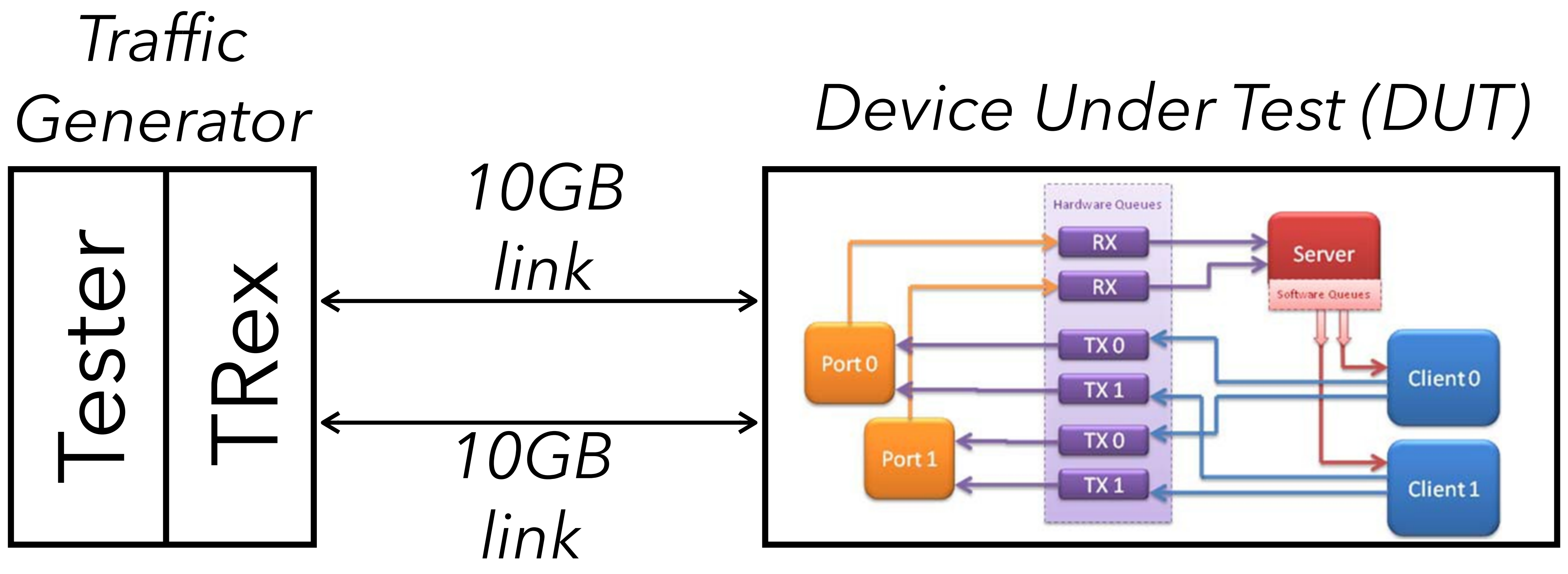
- Device Under Test
  - One server process

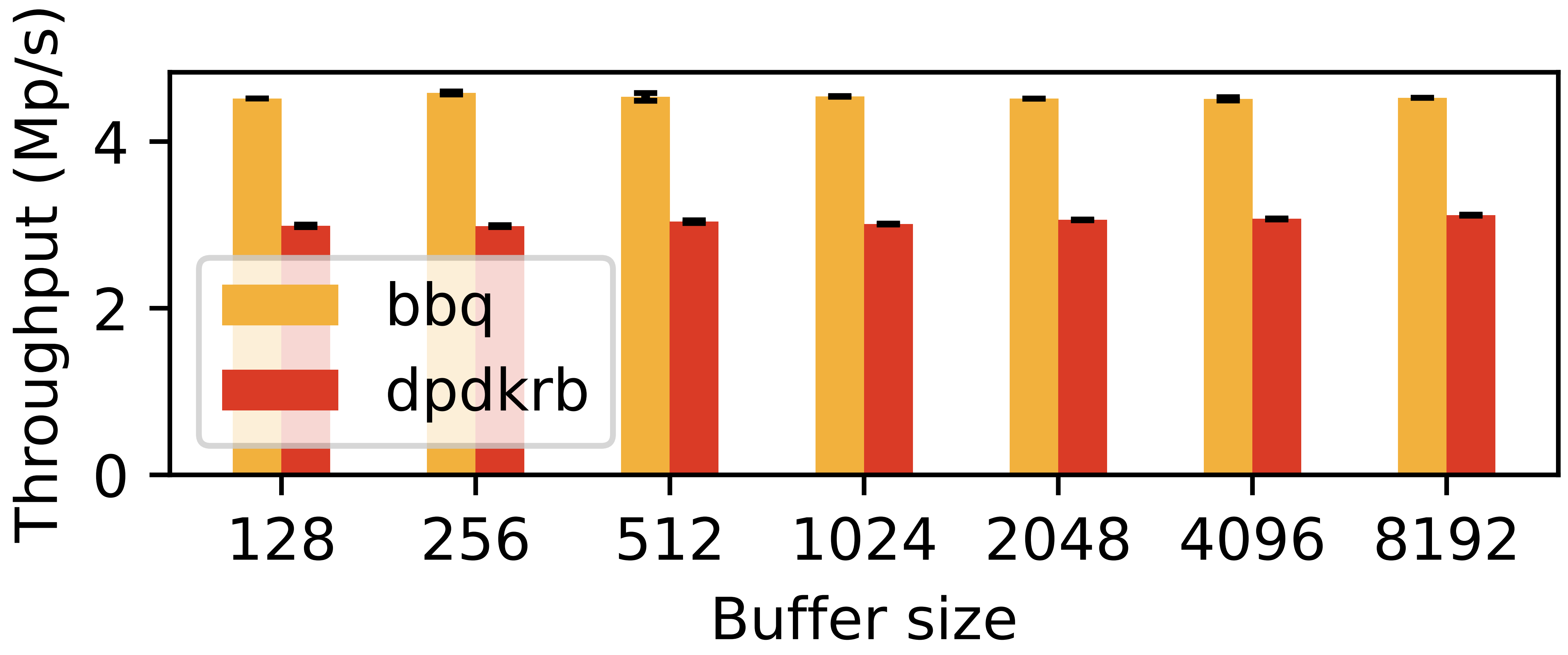
    receiving and distributing packets
  - Two client processes

    performing level-2 packet forwarding

- Tester and traffic generator run on another machine

- BBQ yields **1.5x throughput** of DPDK



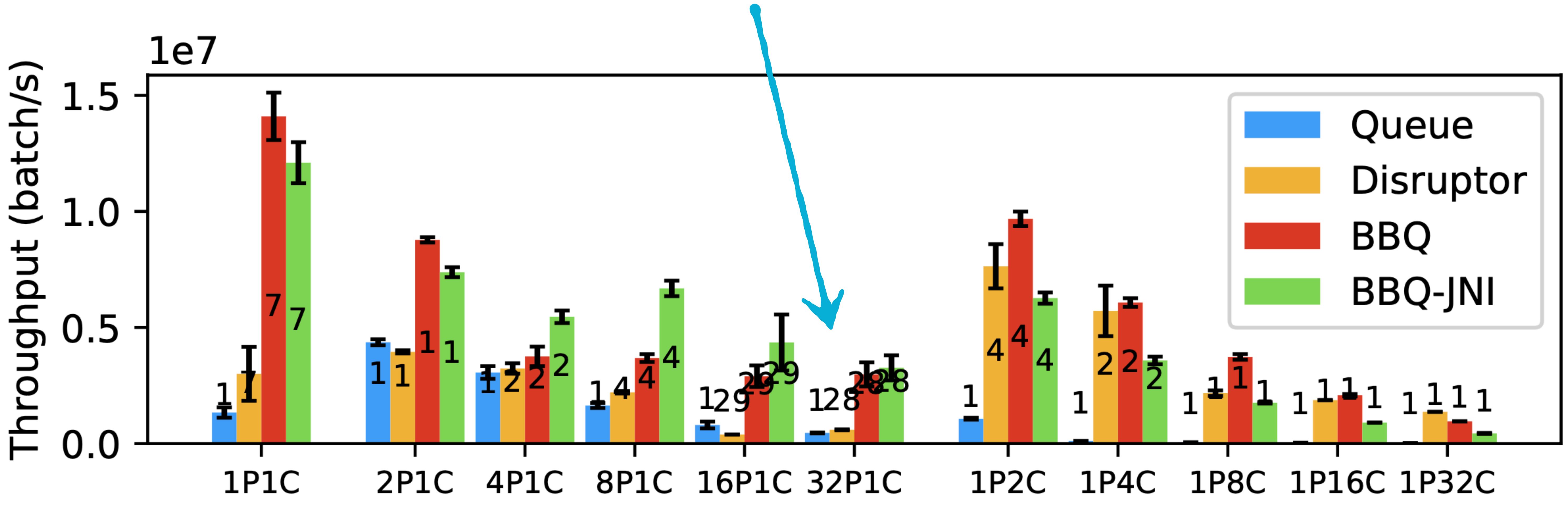


Macro-benchmark Results – Disruptor

- LMAX Disruptor: bounded queue for high-performance trading

- Compared on three official Disruptor benchmarks
  Against Java queue, BBQ in Java, and BBQ in C via JNI

- With 32 producers, **BBQ yields 3 Mop/s and Disruptor 0.6 Mop/s**



Wrap up

BBQ is a novel ring buffer design

- Reduces enq-deq interference
- Supports out-of-order operations
- Model checked for WMMs

Large spectrum of scenarios

- Single/Multi Consumer/Producer
- Retry-new and Drop-old modes
- Etc

Greatly outperforms several industrial ring buffers

Please **look up the paper** for many more results

Thank you! Questions?

(BTW, we are hiring in Dresden and Munich…)

HUAWEI

TECHNISCHE UNIVERSITÄT DRESDEN

SHANGHAI JIAO TONG UNIVERSITY