



A Quest for Predictable Latency

Adventures in Java Concurrency

Martin Thompson - @mjpt777



***If a system does not respond
in a timely manner then it is
effectively unavailable***

1. It's all about the **Blocking**
2. What do we mean by **Latency**
3. Adventures with **Locks** and queues
4. Some **Alternative** FIFOs
5. Where can we go **Next**

1. It's all about the Blocking

What is preventing progress?

A thread is **blocked when it
cannot make progress**

**There are two major causes of
blocking**

Blocking



- **Systemic Pauses**
 - JVM Safepoints (GC, etc.)
 - Transparent Huge Pages (Linux)
 - Hardware (C-States, SMIs)

Blocking



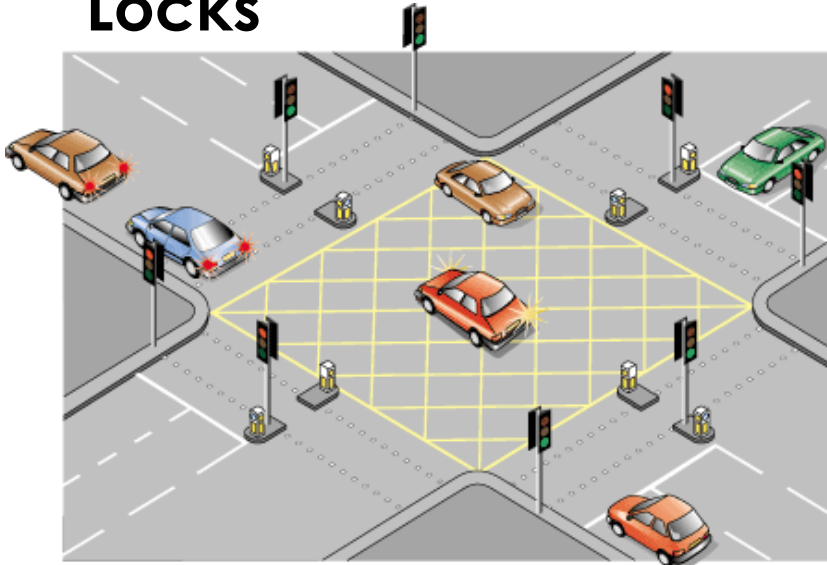
- **Systemic Pauses**
 - JVM Safepoints (GC, etc.)
 - Transparent Huge Pages (Linux)
 - Hardware (C-States, SMIs)



- **Concurrent Algorithms**
 - Notifying Completion
 - Mutual Exclusion (Contention)
 - Synchronisation / Rendezvous

Concurrent Algorithms

Locks



- Call into kernel on contention
- Always blocking
- Difficult to get right

Atomic/CAS Instructions



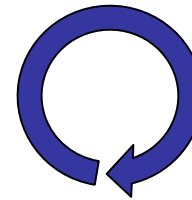
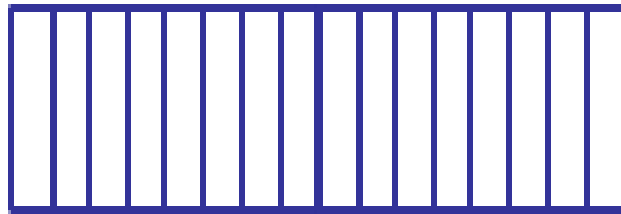
- Execute in user space
- Can be non-blocking
- Very difficult to get right

2. What do we mean by Latency

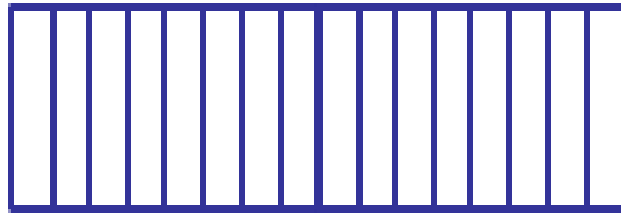
UK Border



Queuing Theory



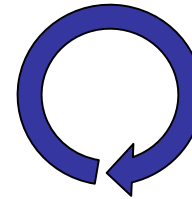
Queuing Theory



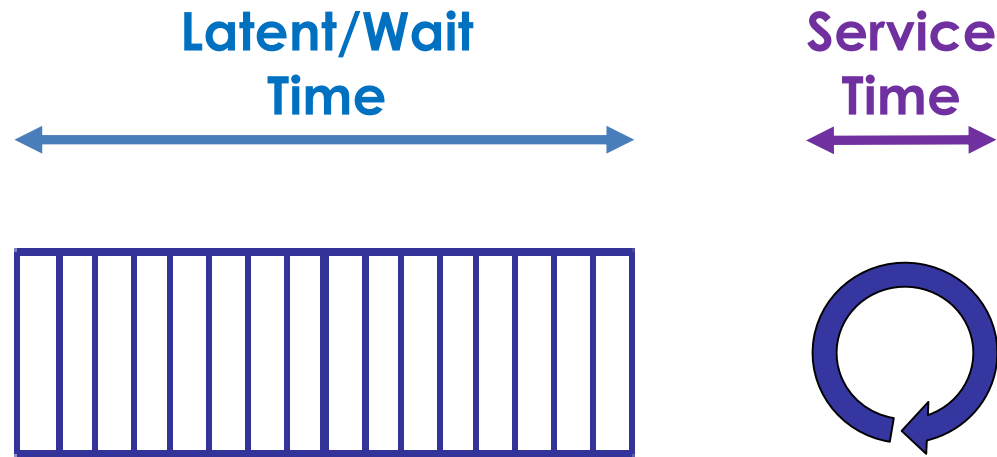
Service
Time



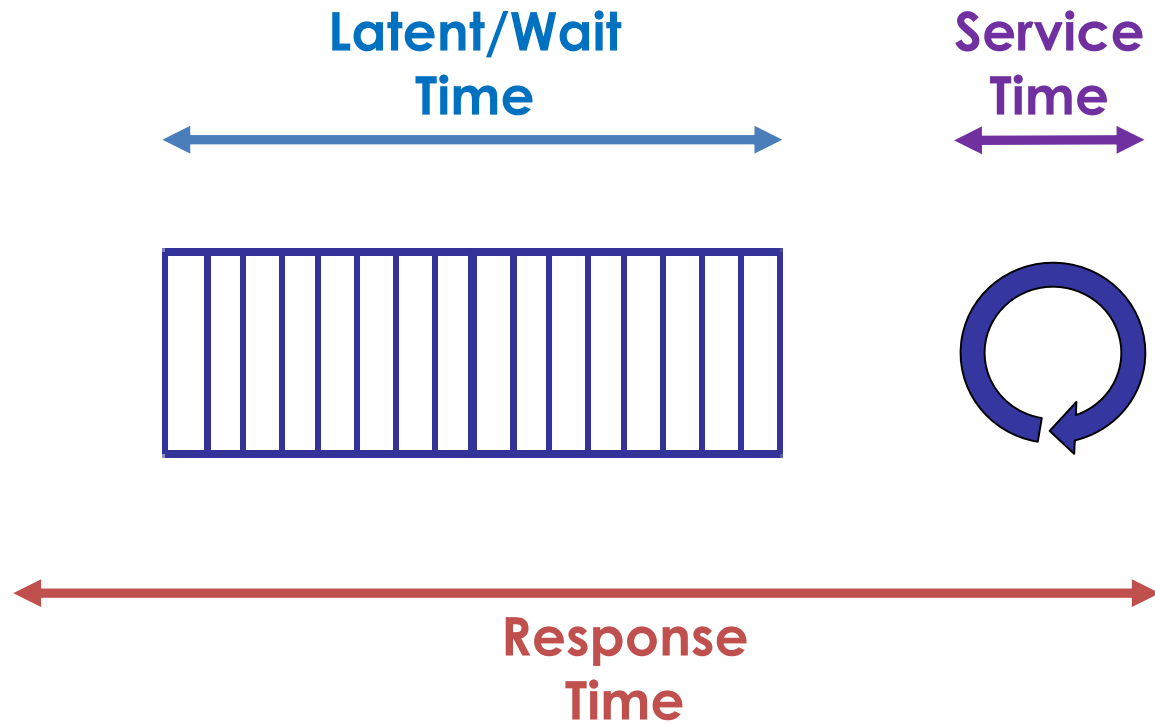
A horizontal double-headed arrow pointing left and right, indicating the duration of service time.



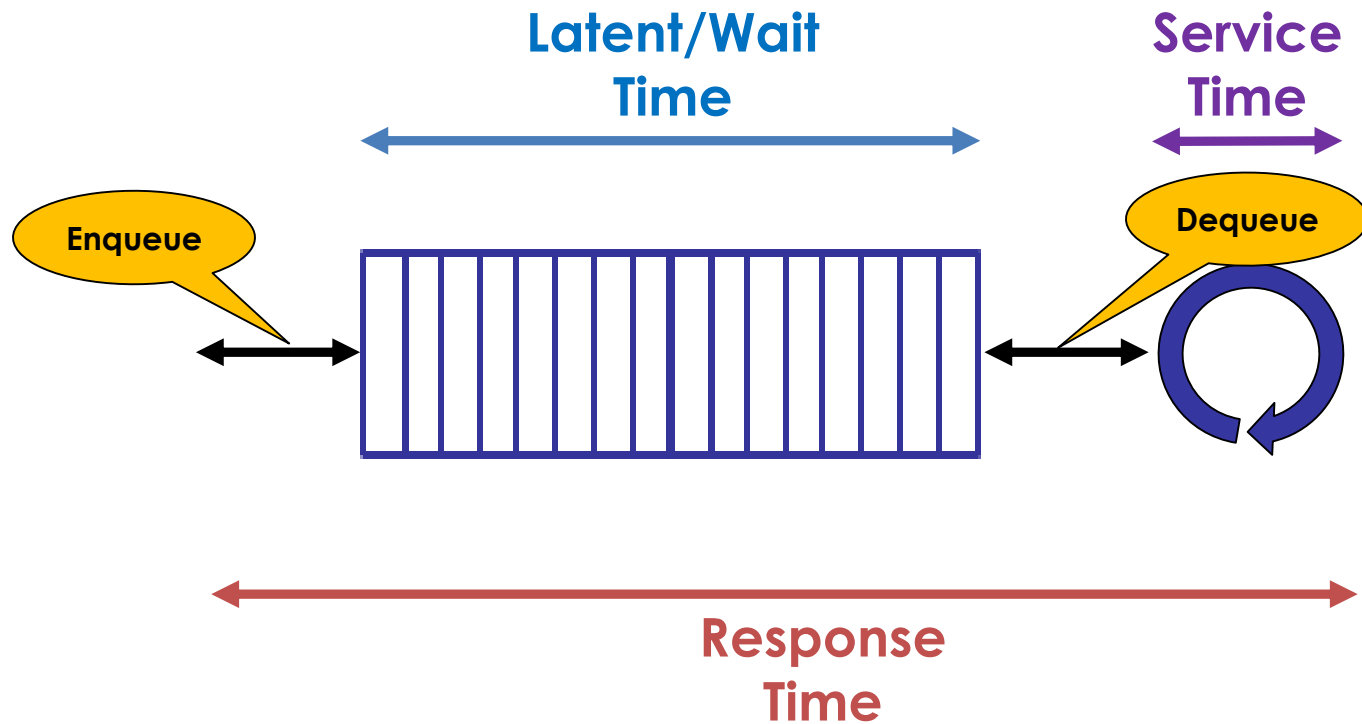
Queuing Theory



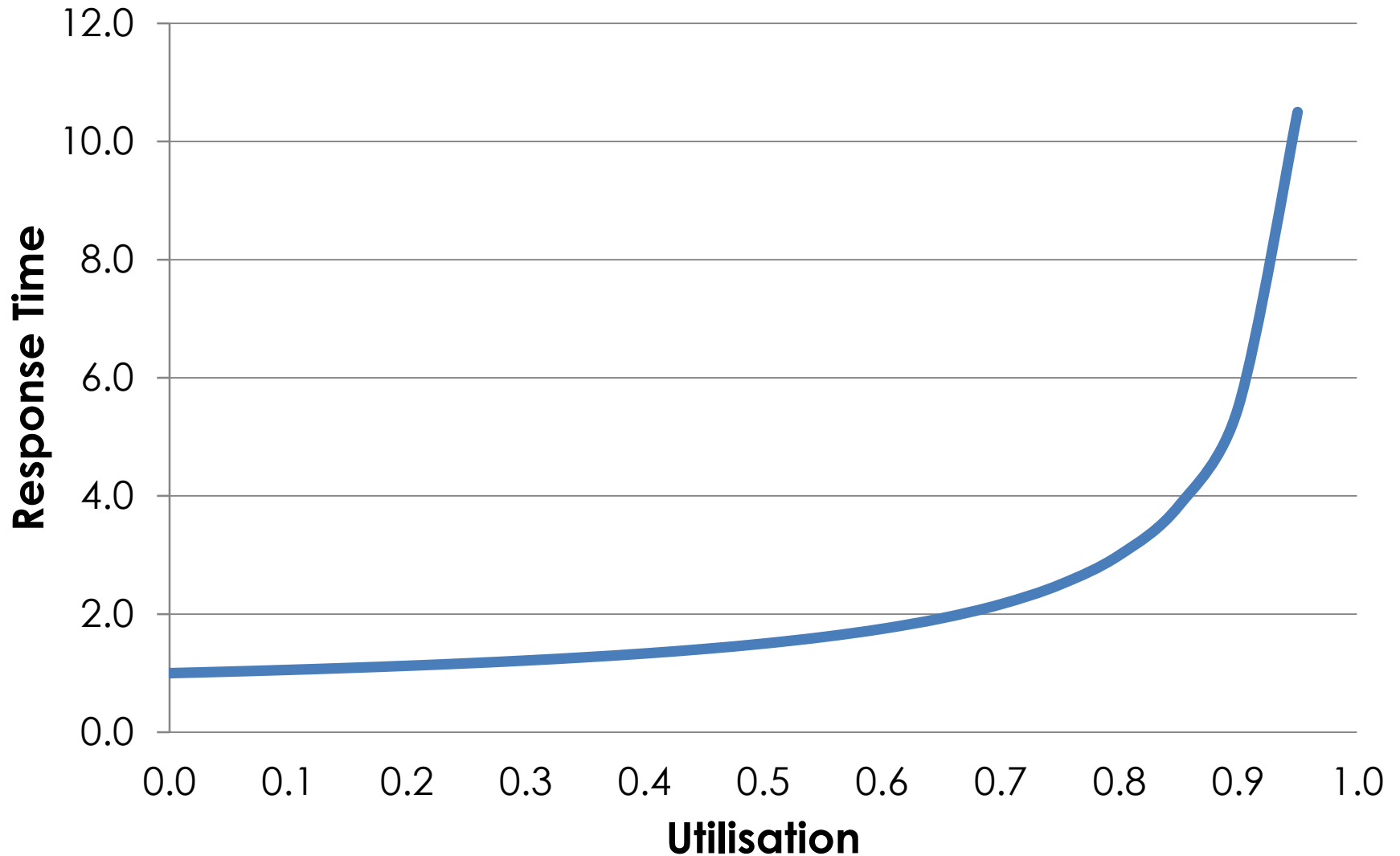
Queuing Theory



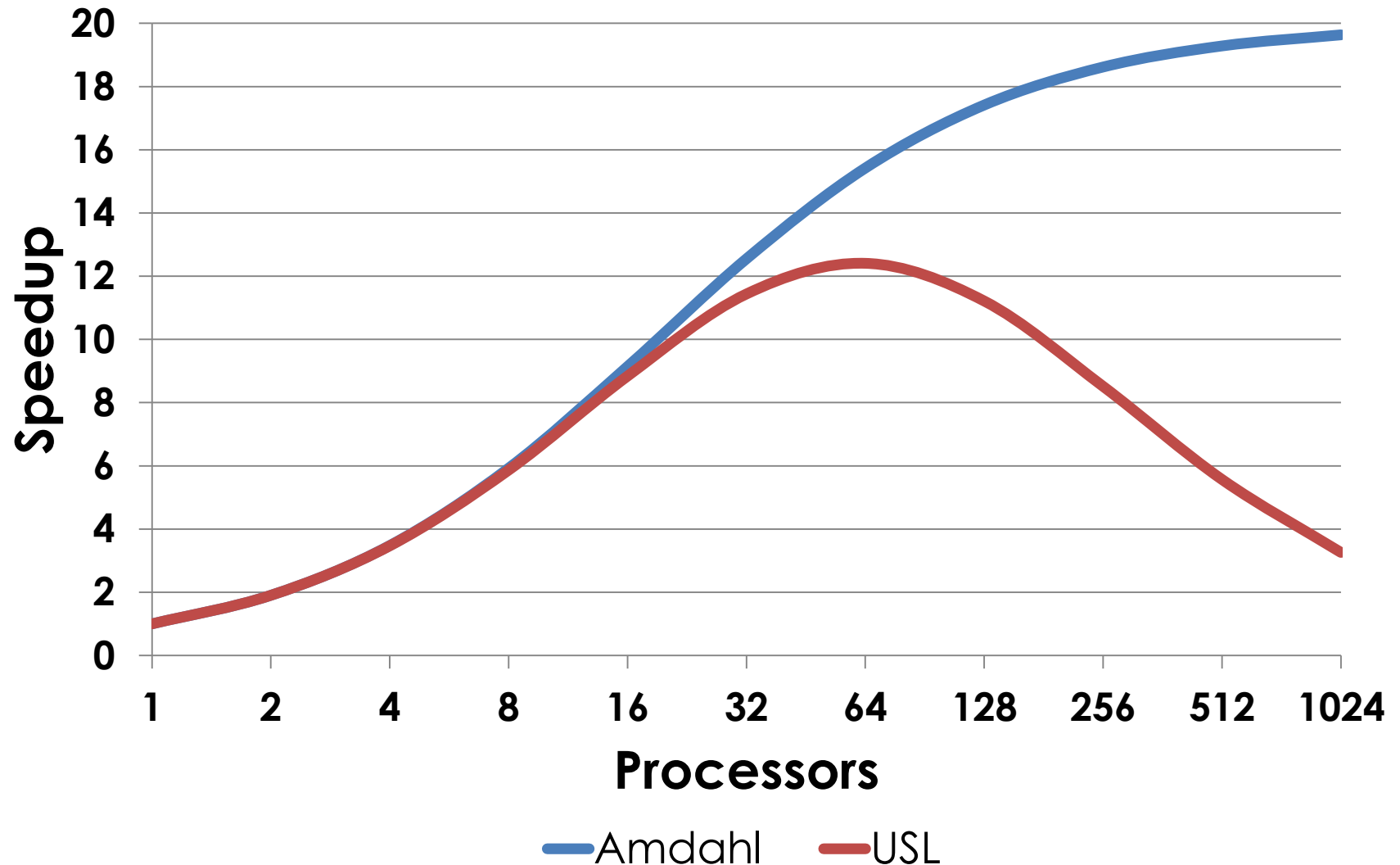
Queuing Theory



Queuing Theory



Amdahl's & Universal Scalability Laws



3. Adventures with Locks and Queues?

***Some queue implementations
spend more time queueing to be
enqueued than in queue time!!!***

The evils of **Blocking**

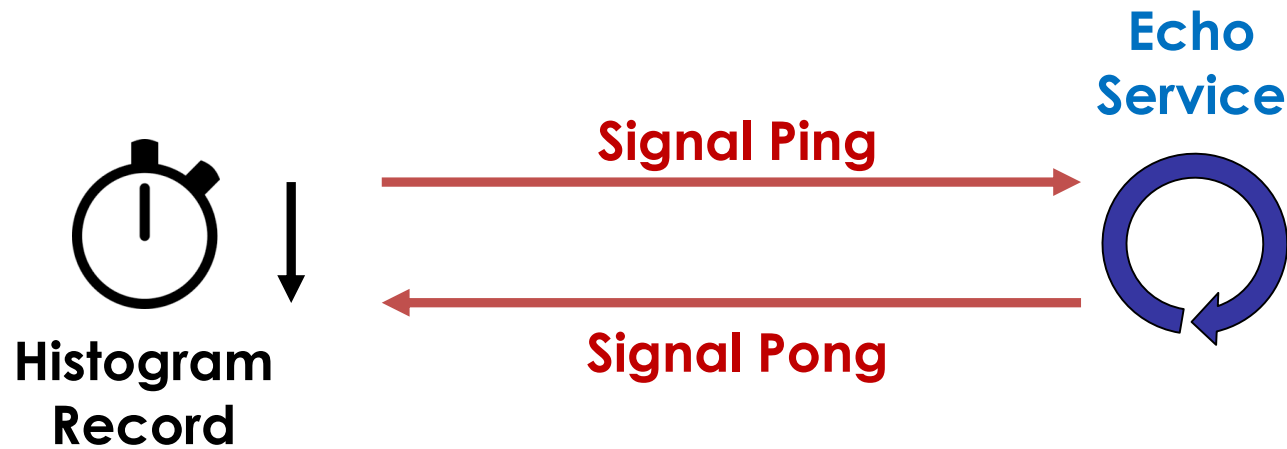
Queue.put()

&&

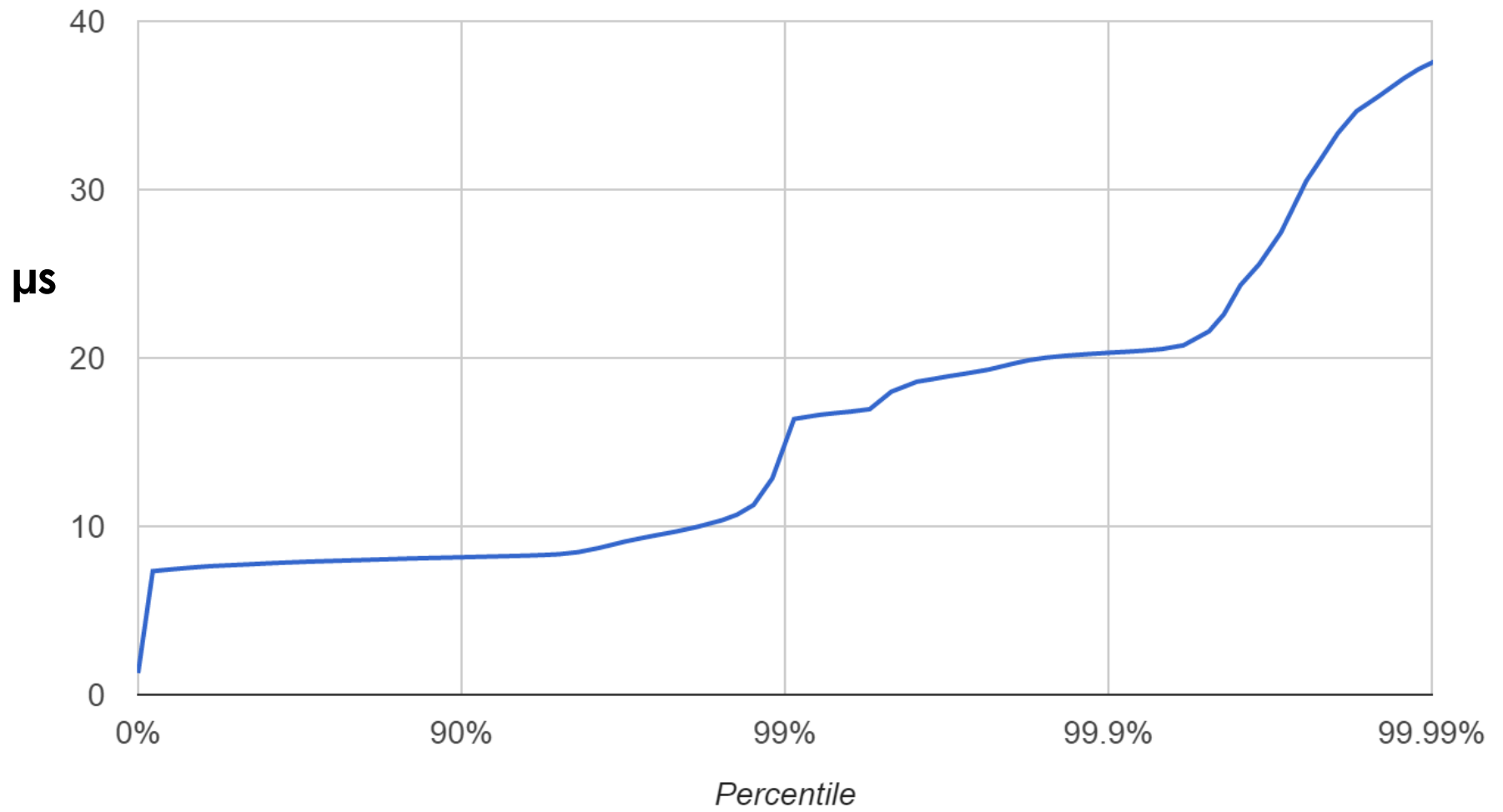
Queue.take()

Condition Variables

Condition Variable RTT

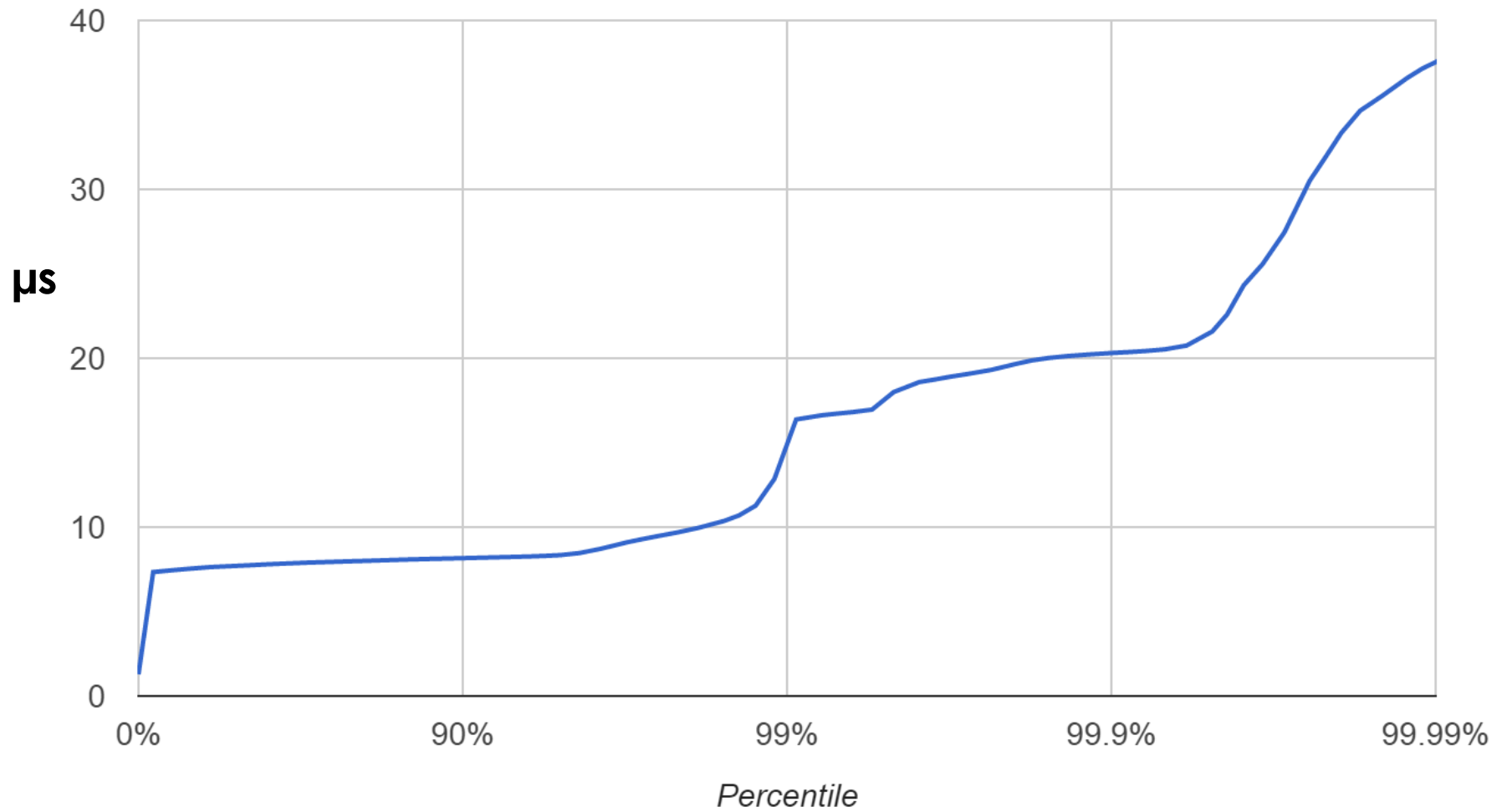


Latency by Percentile Distribution



Latency by Percentile Distribution

Max =
5525.503μs



Bad News

That's the best case scenario!

Are non-blocking APIs any better?

`Queue.offer()`

`&&`

`Queue.poll()`

Context for the Benchmarks

<https://github.com/real-logic/benchmarks>

Java 8u60

Ubuntu 15.04 – “performance” mode

Intel i7-3632QM – 2.2 GHz (Ivy Bridge)

Burst Length = 1: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	167	189
ArrayBlockingQueue	1	645	1,210
	2	1,984	11,648
	3	5,257	19,680
LinkedBlockingQueue	1	461	740
	2	1,197	7,320
	3	2,010	15,152
ConcurrentLinkedQueue	1	281	361
	2	381	559
	3	444	705

Burst Length = 100: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	721	982
ArrayBlockingQueue	1	30,346	41,728
	2	35,631	65,008
	3	50,271	90,240
LinkedBlockingQueue	1	31,422	41,600
	2	45,096	94,208
	3	89,820	180,224
ConcurrentLinkedQueue	1	12,916	15,792
	2	25,132	35,136
	3	39,462	56,768

Burst Length = 100: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	721	982
ArrayBlockingQueue	1	30,346	41,728
	2	35,631	65,008
	3	50,271	90,240
LinkedBlockingQueue	1	31,422	41,600
	2	45,096	94,208
	3	89,820	180,224
ConcurrentLinkedQueue	1	12,916	15,792
	2	25,132	35,136
	3	39,462	56,768

Backpressure?

Size methods?

Flow Rates?

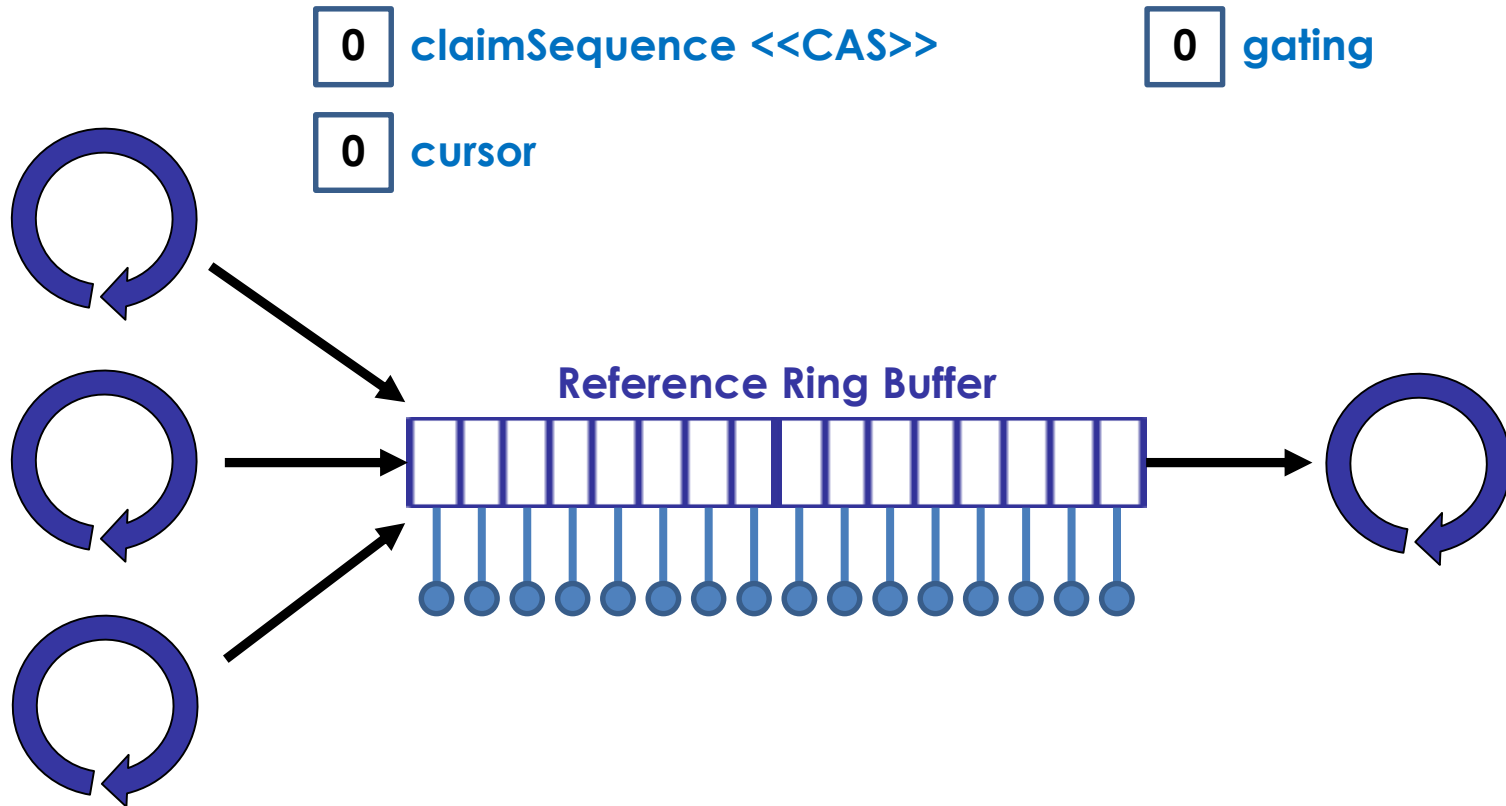
Garbage?

Fan out?

5. Some alternative FIFOs

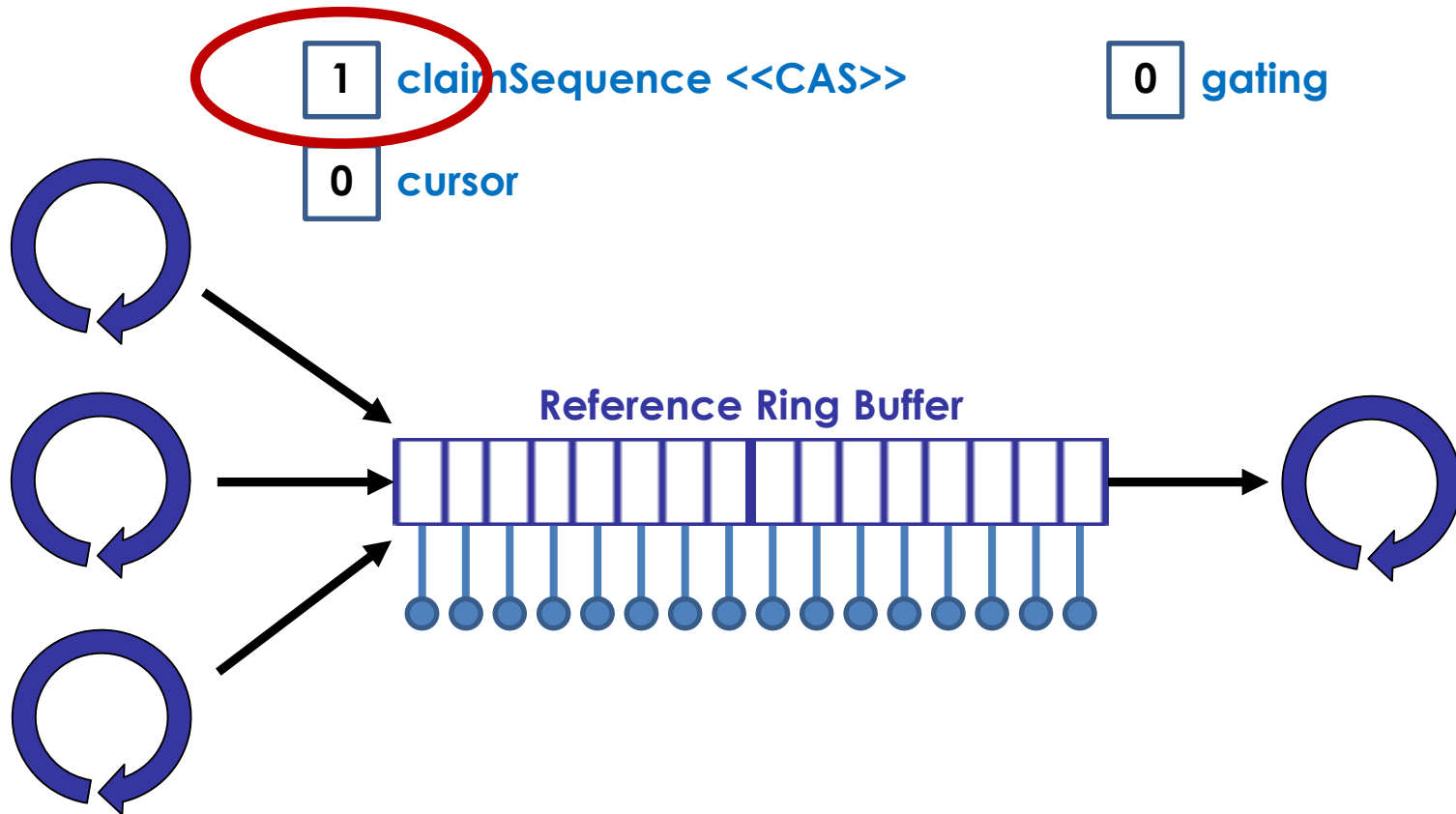
Inter-Thread FIFOs

Disruptor 1.0 – 2.0



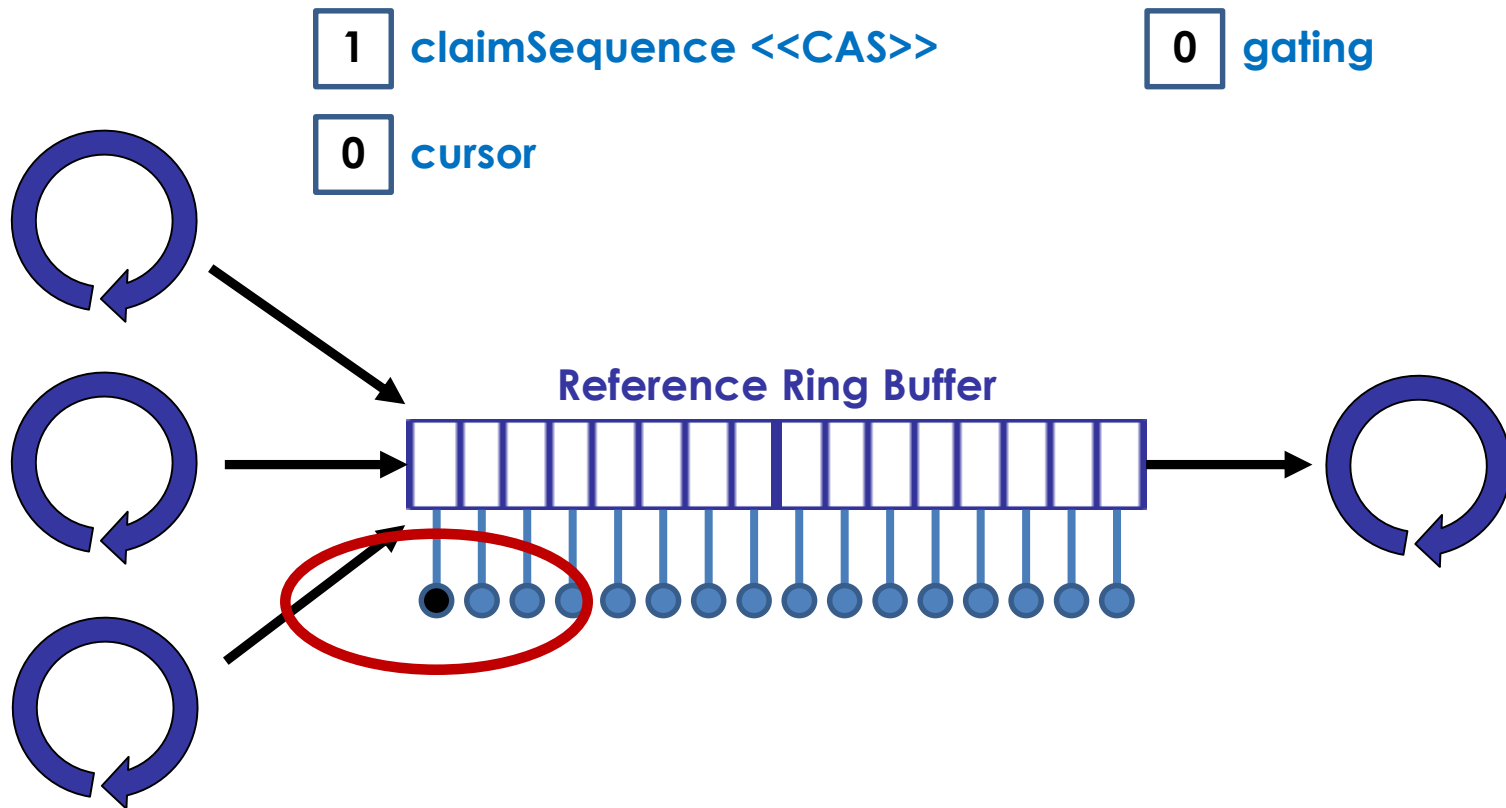
Influences: Lamport + Network Cards

Disruptor 1.0 – 2.0



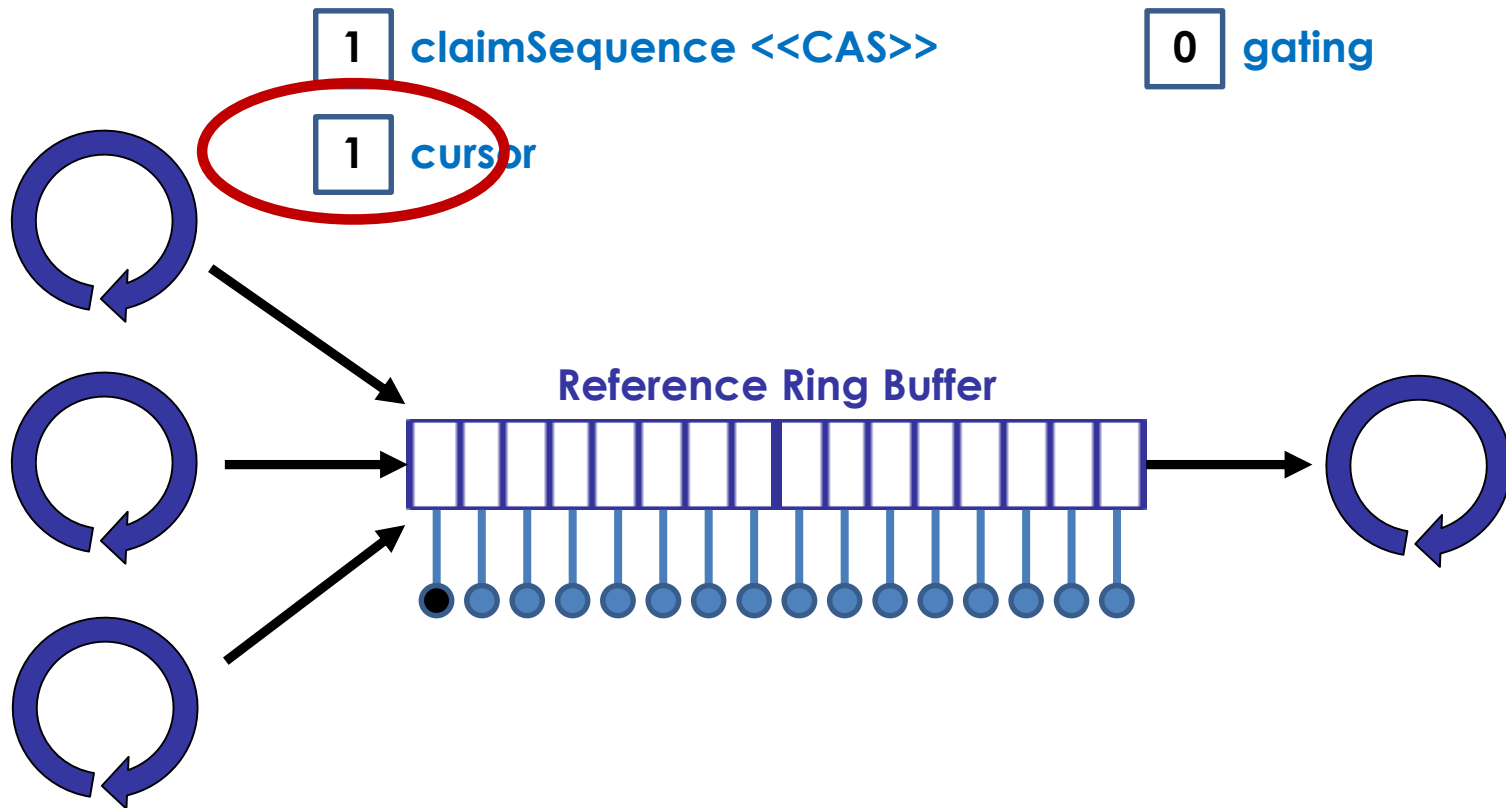
Influences: Lamport + Network Cards

Disruptor 1.0 – 2.0



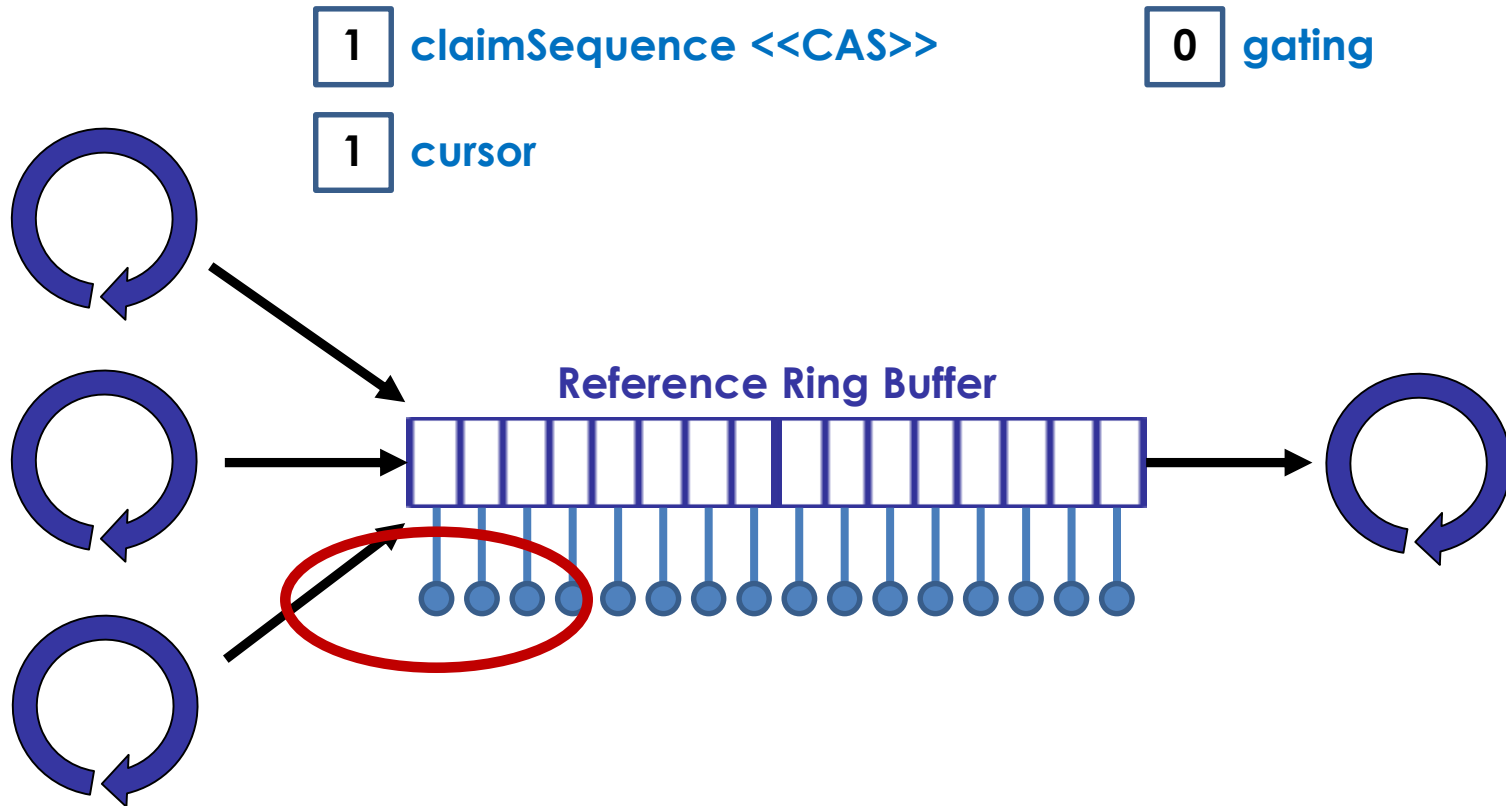
Influences: Lamport + Network Cards

Disruptor 1.0 – 2.0



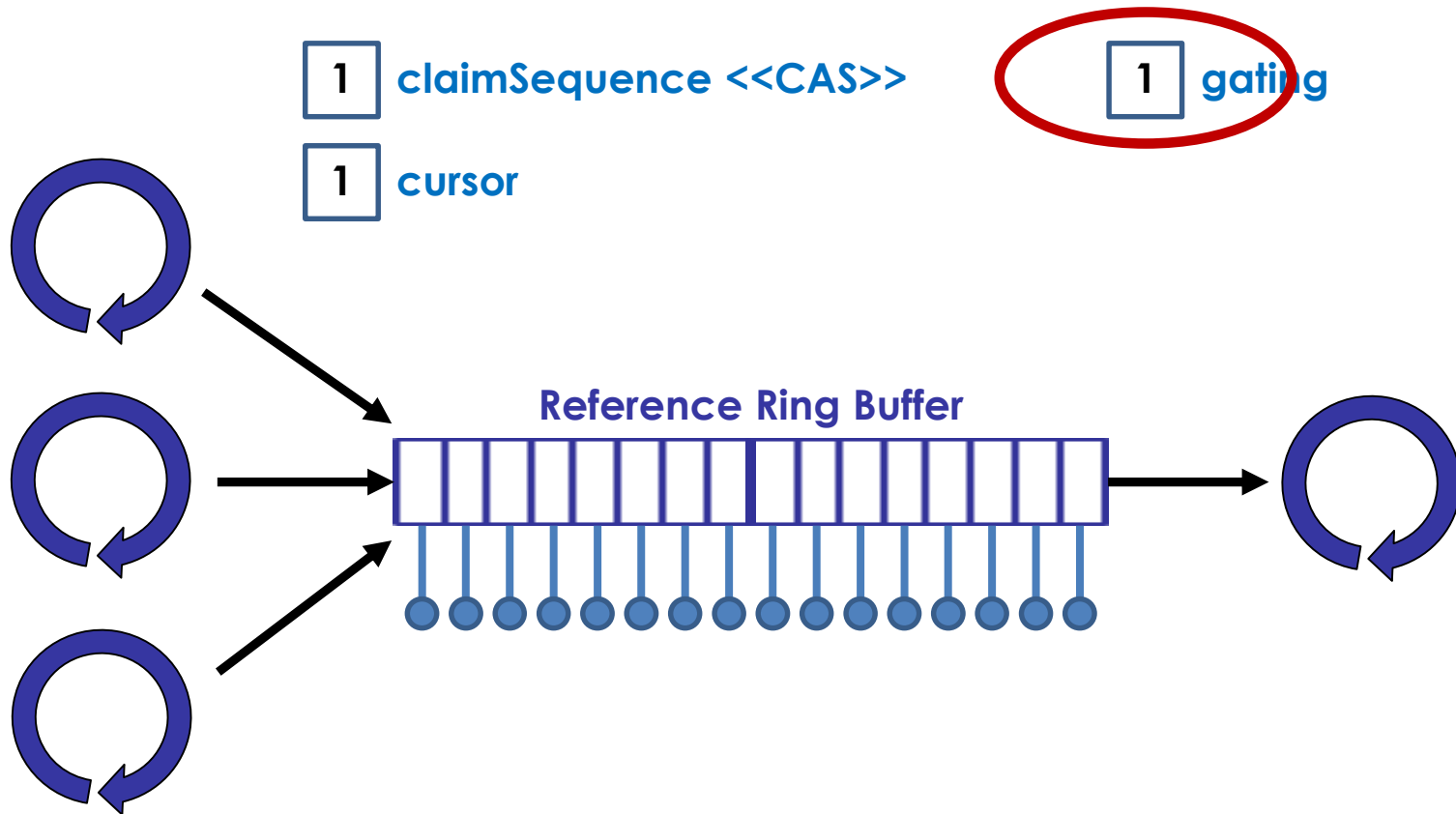
Influences: Lamport + Network Cards

Disruptor 1.0 – 2.0



Influences: Lamport + Network Cards

Disruptor 1.0 – 2.0



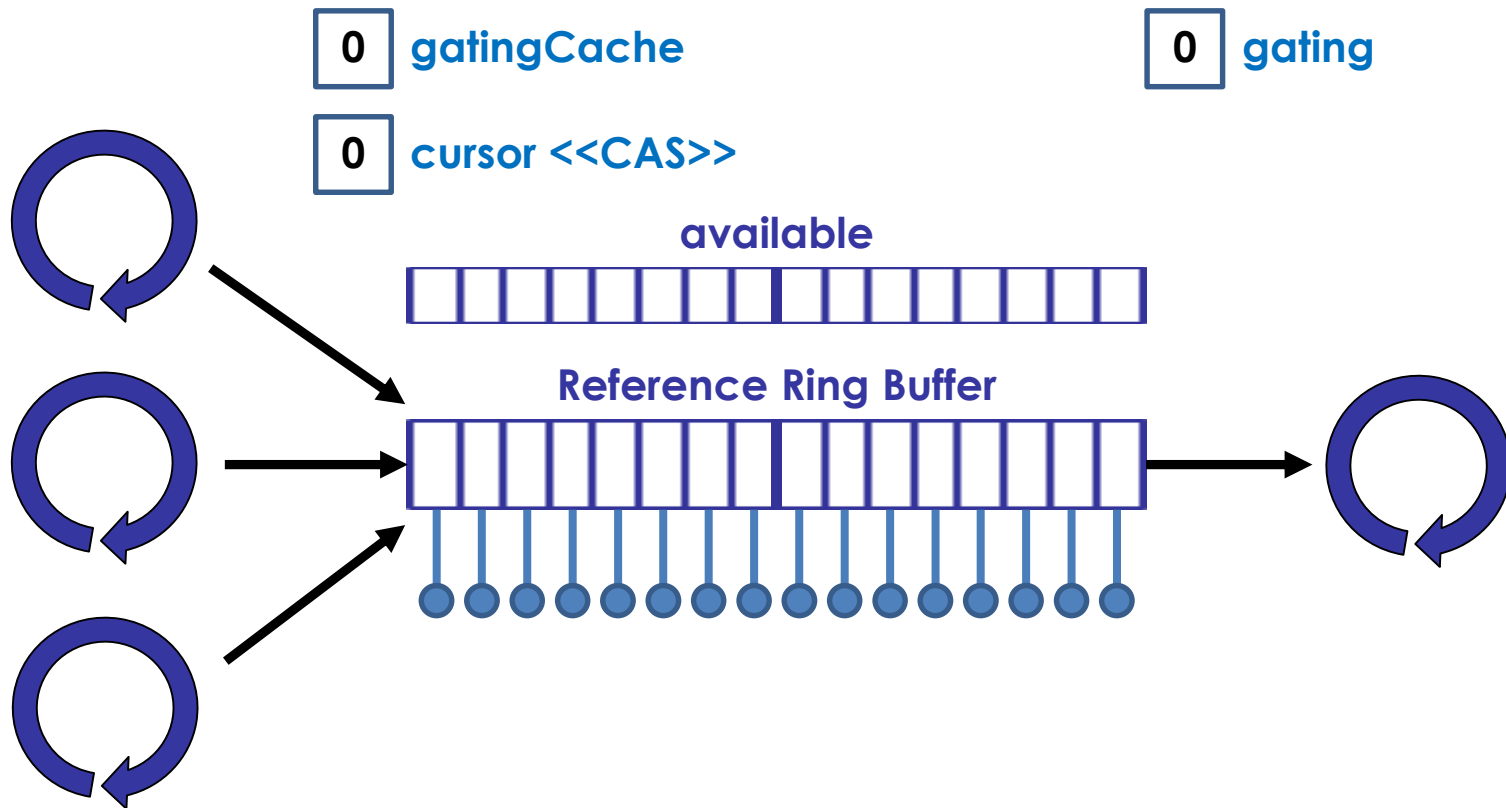
Influences: Lamport + Network Cards

```
long expectedSequence = claimedSequence - 1;

while (cursor != expectedSequence)
{
    // busy spin
}

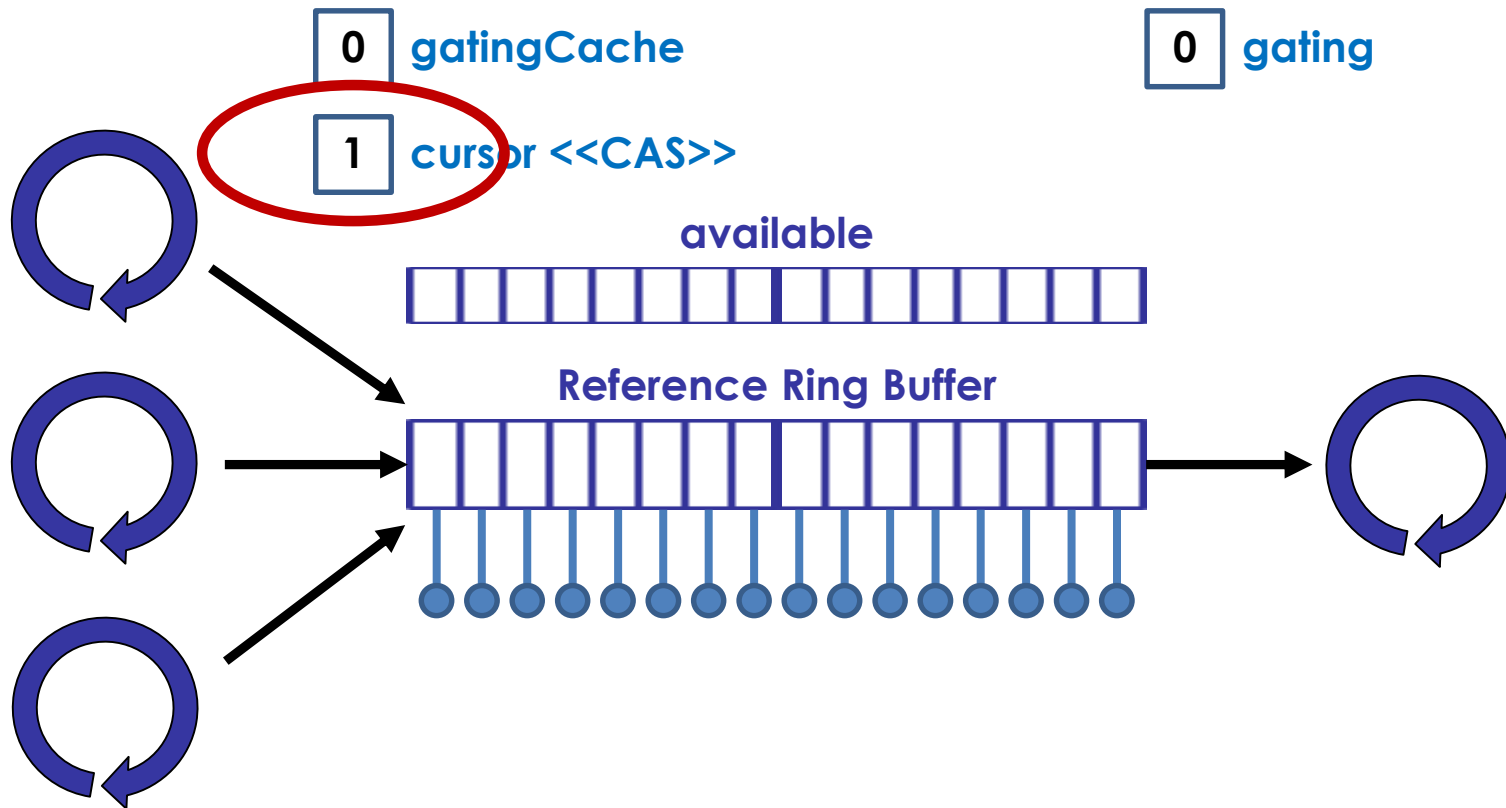
cursor = claimedSequence;
```

Disruptor 3.0



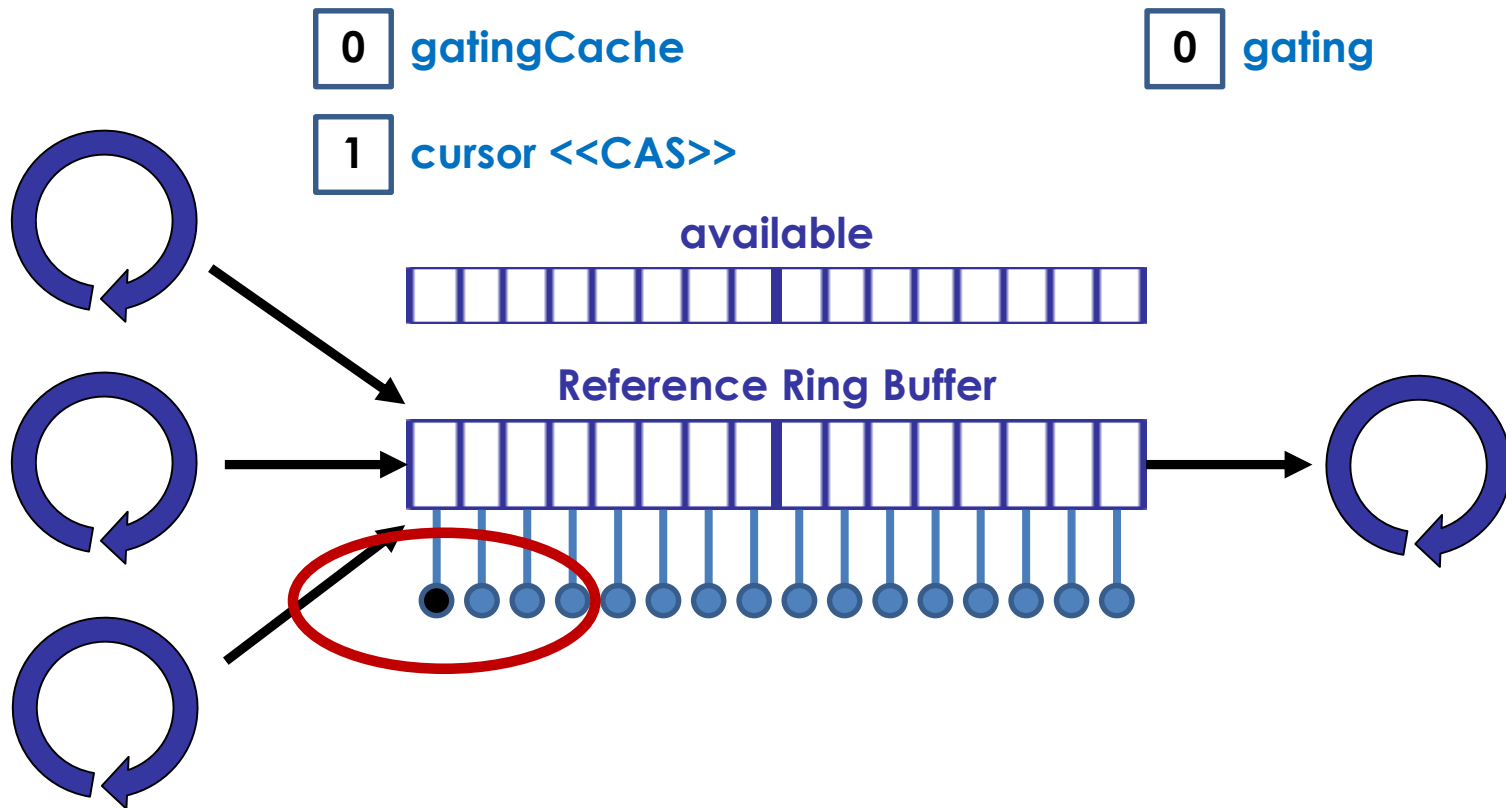
Influences: Lamport + Network Cards + Fast Flow

Disruptor 3.0



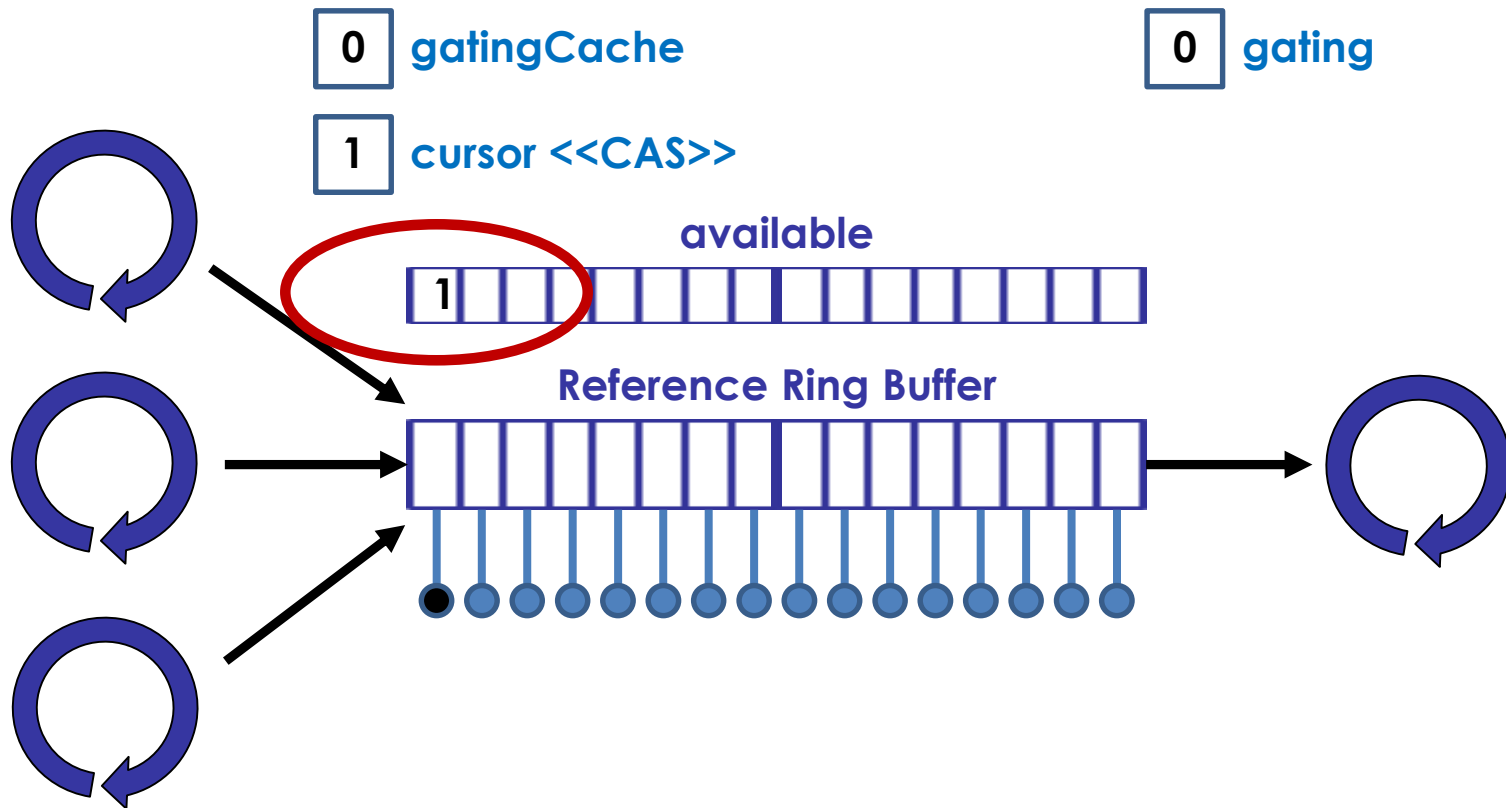
Influences: Lamport + Network Cards + Fast Flow

Disruptor 3.0



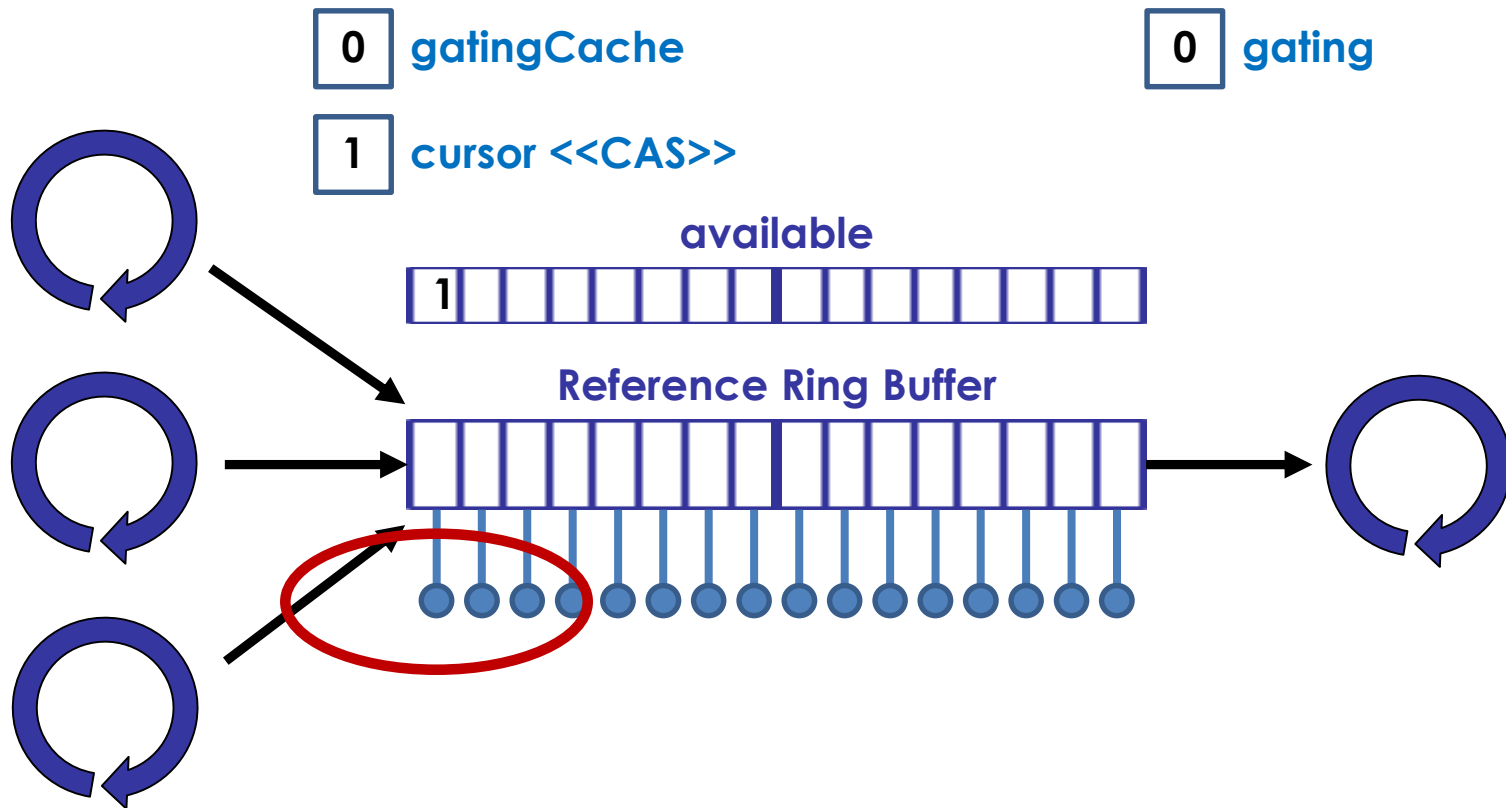
Influences: Lamport + Network Cards + Fast Flow

Disruptor 3.0



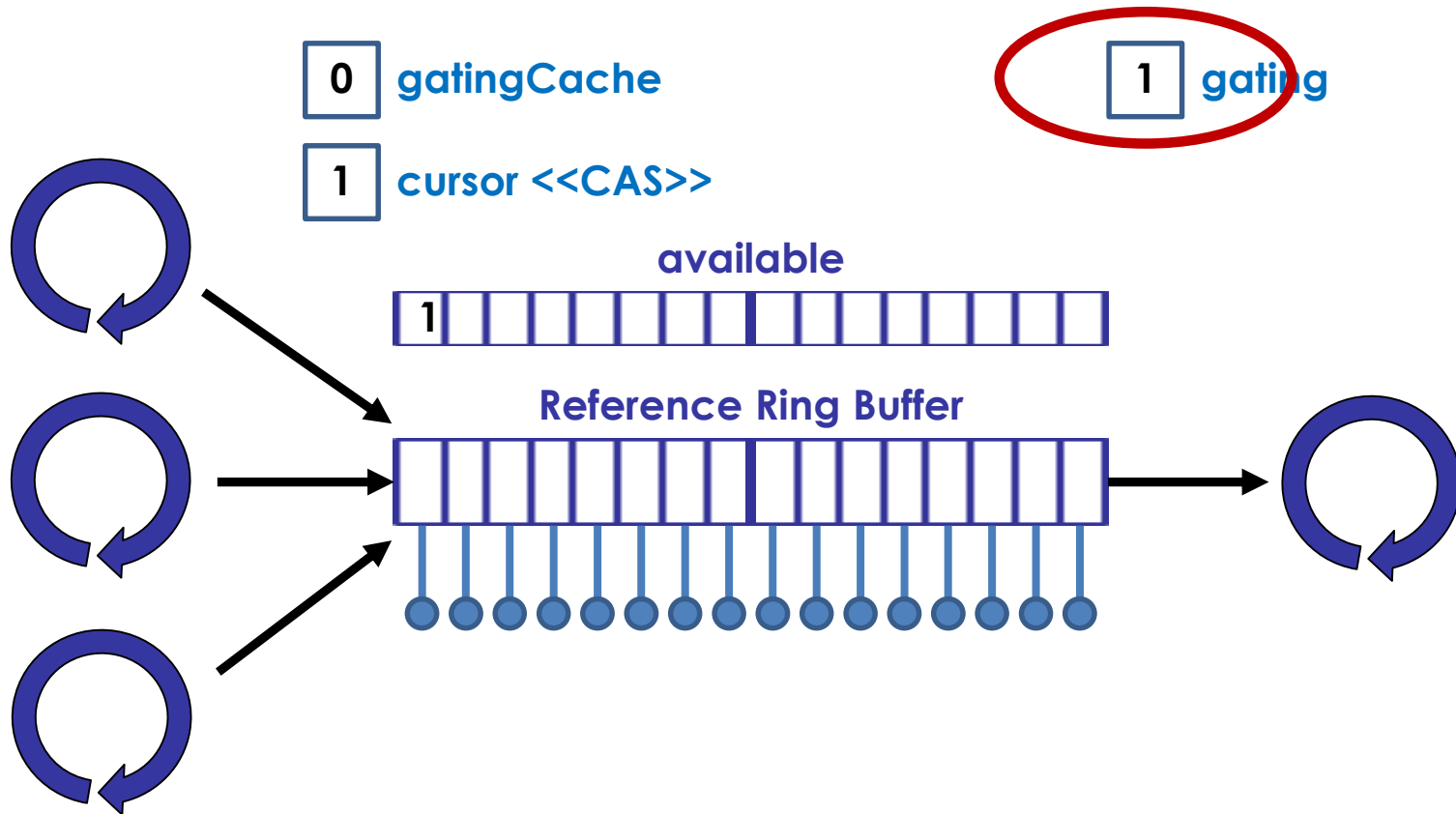
Influences: Lamport + Network Cards + Fast Flow

Disruptor 3.0



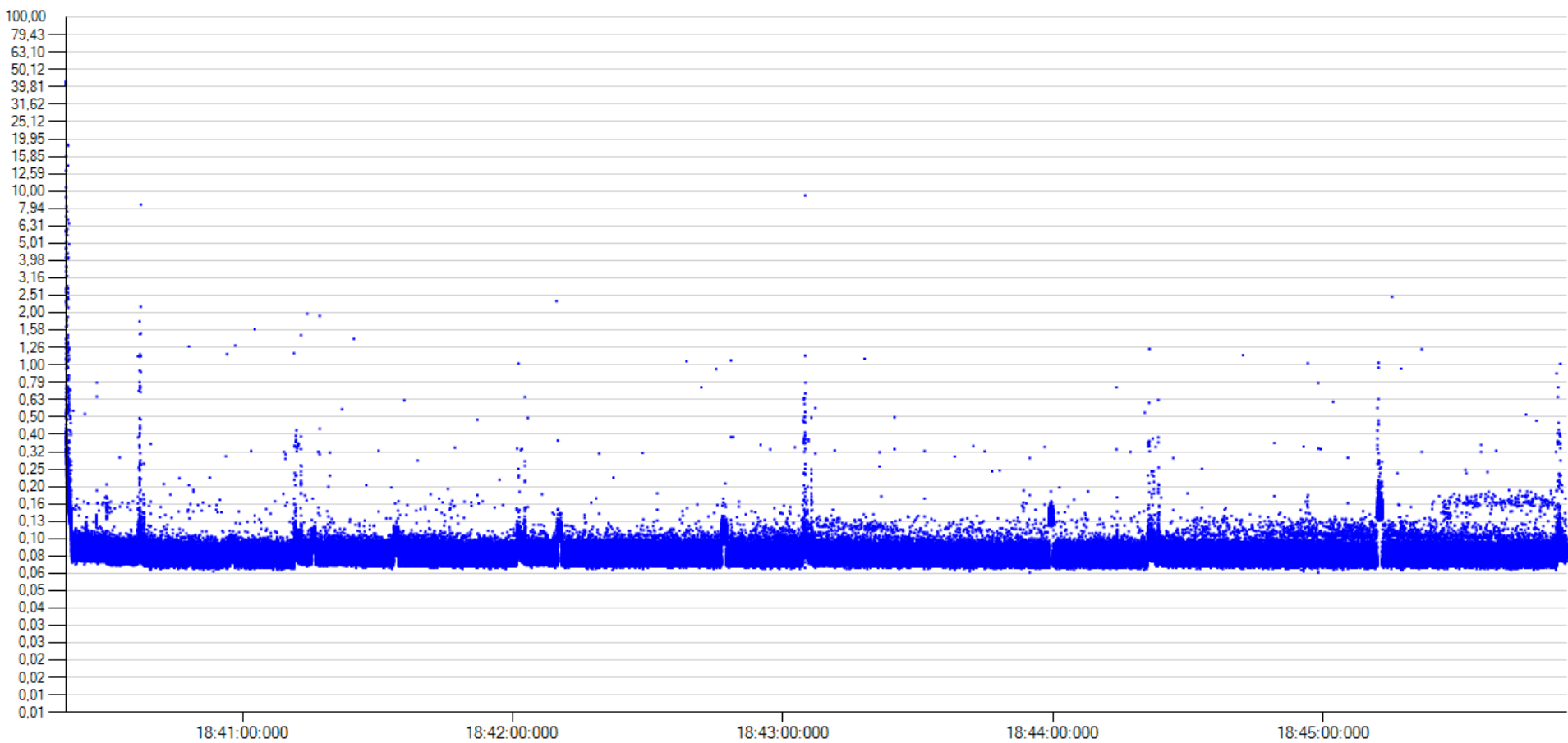
Influences: Lamport + Network Cards + Fast Flow

Disruptor 3.0

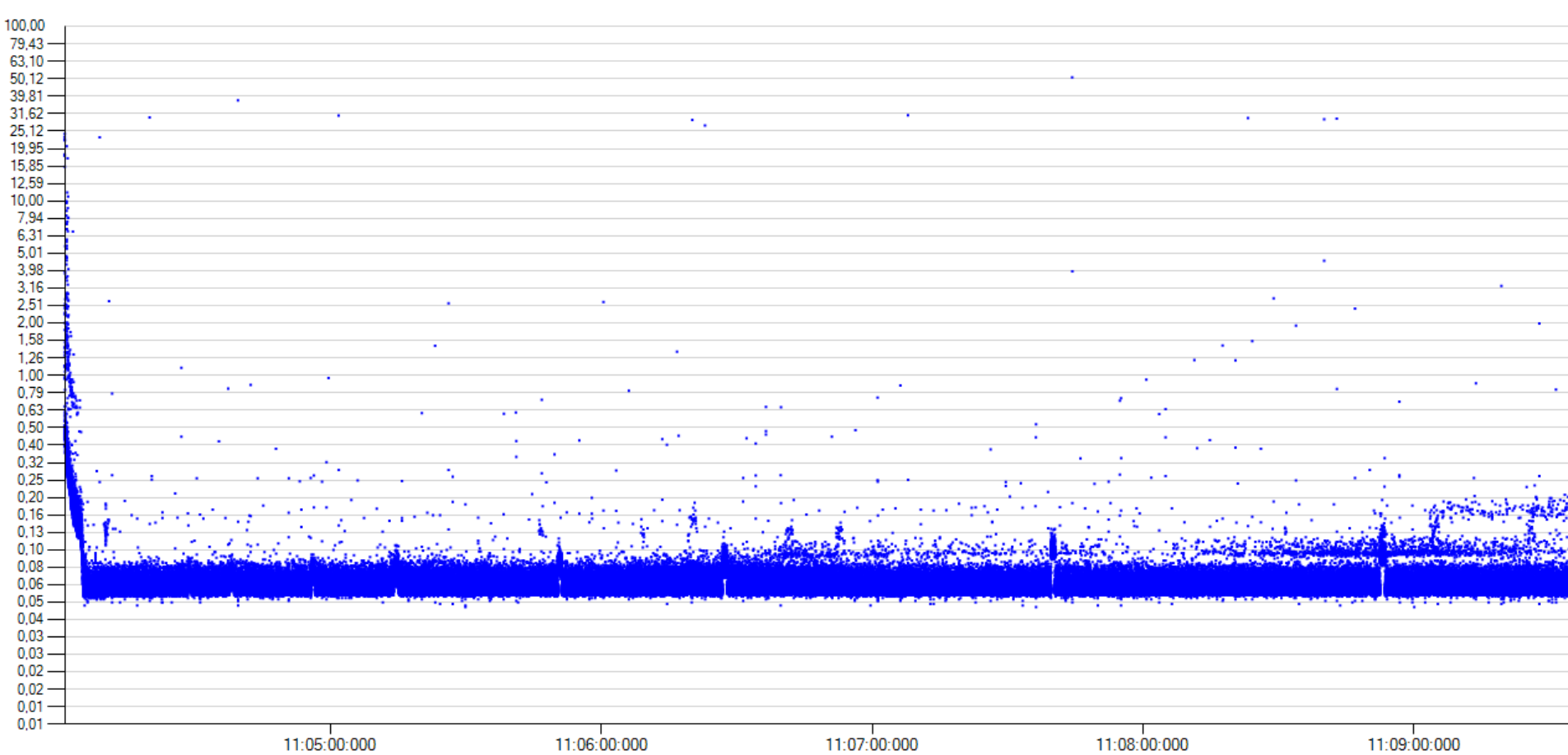


Influences: Lamport + Network Cards + Fast Flow

Disruptor 2.0

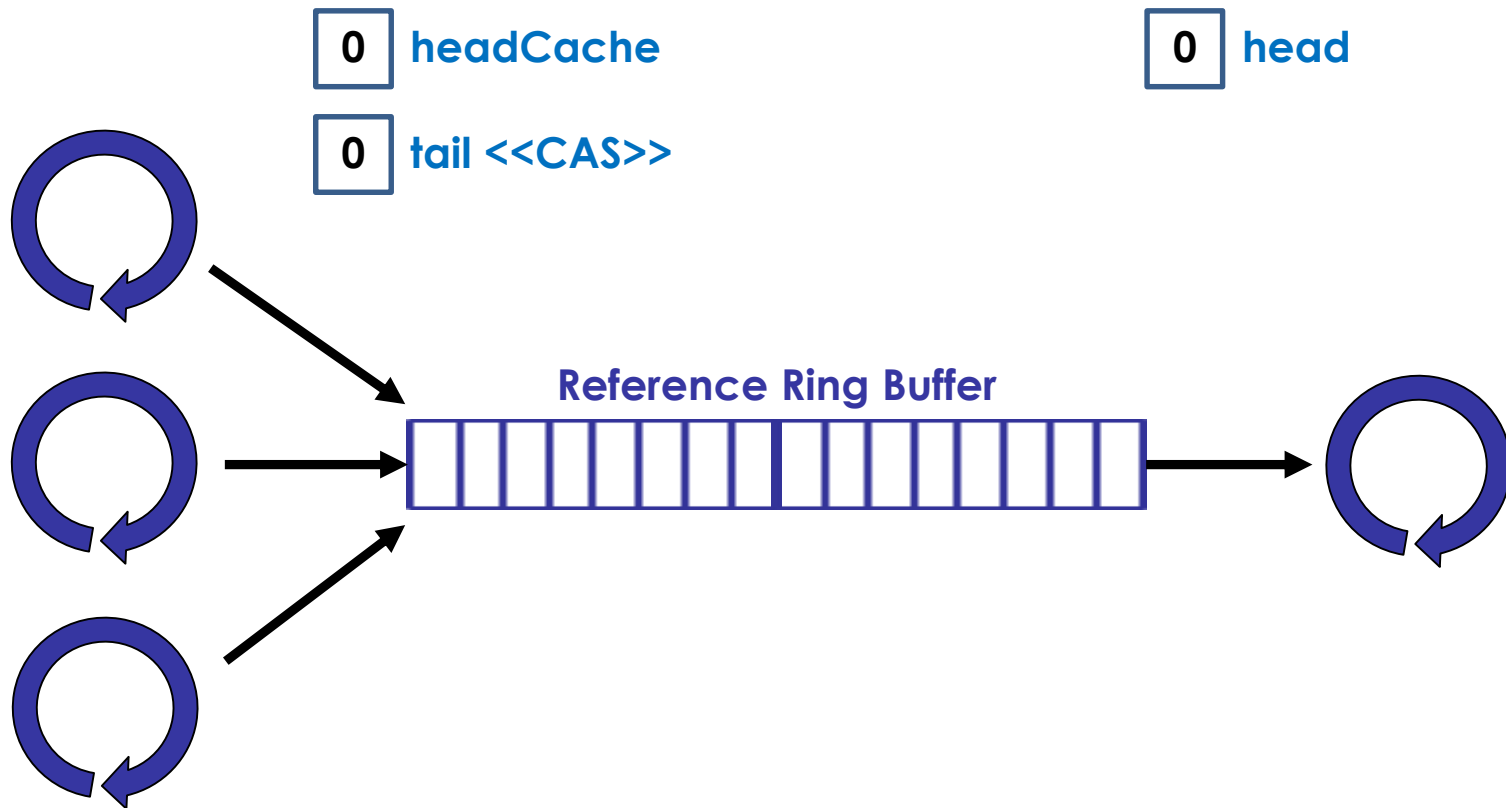


Disruptor 3.0



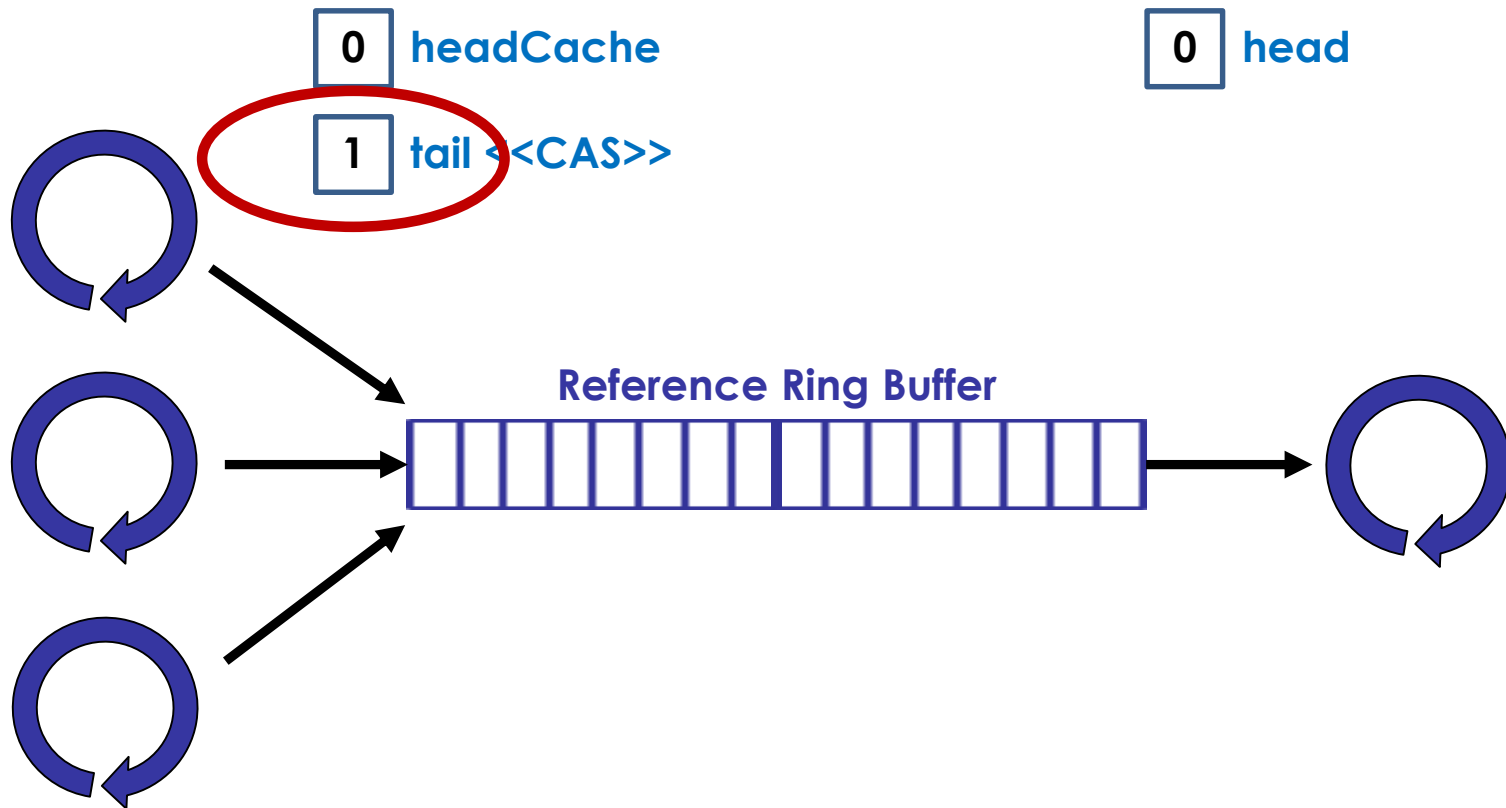
***What if I need something with a
Queue interface?***

ManyToOneConcurrentArrayQueue



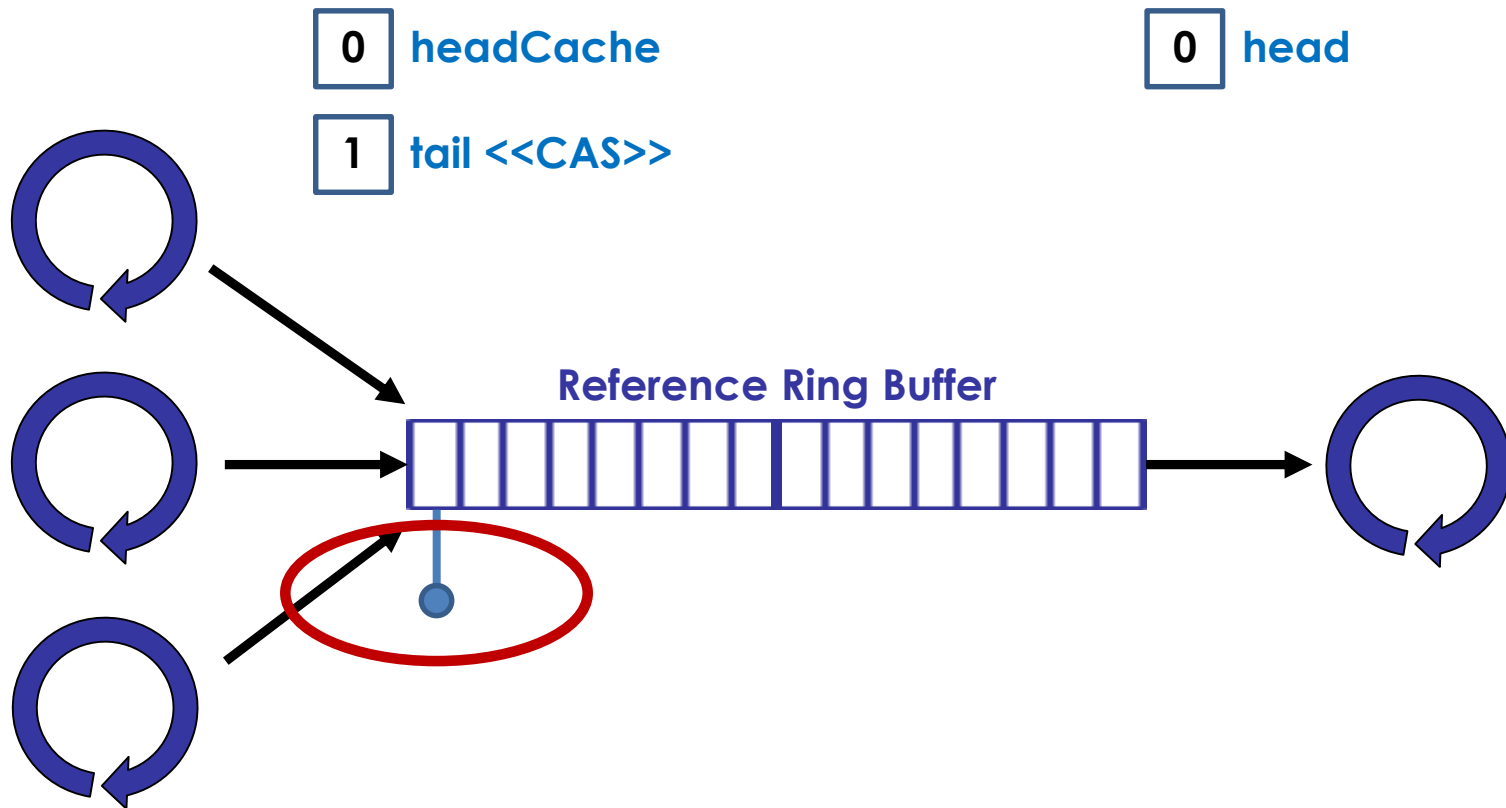
Influences: Lamport + Fast Flow

ManyToOneConcurrentArrayQueue



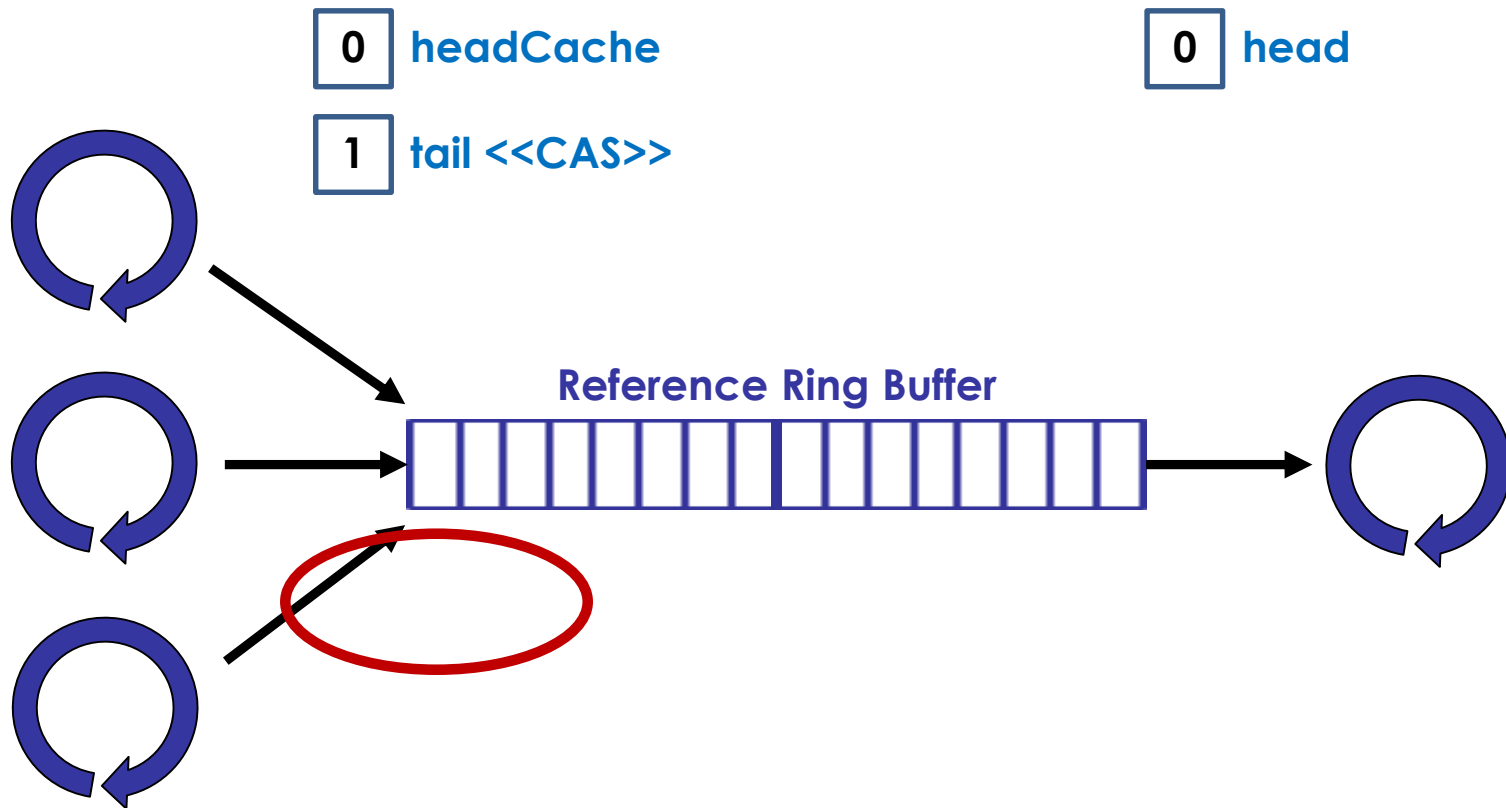
Influences: Lamport + Fast Flow

ManyToOneConcurrentArrayQueue



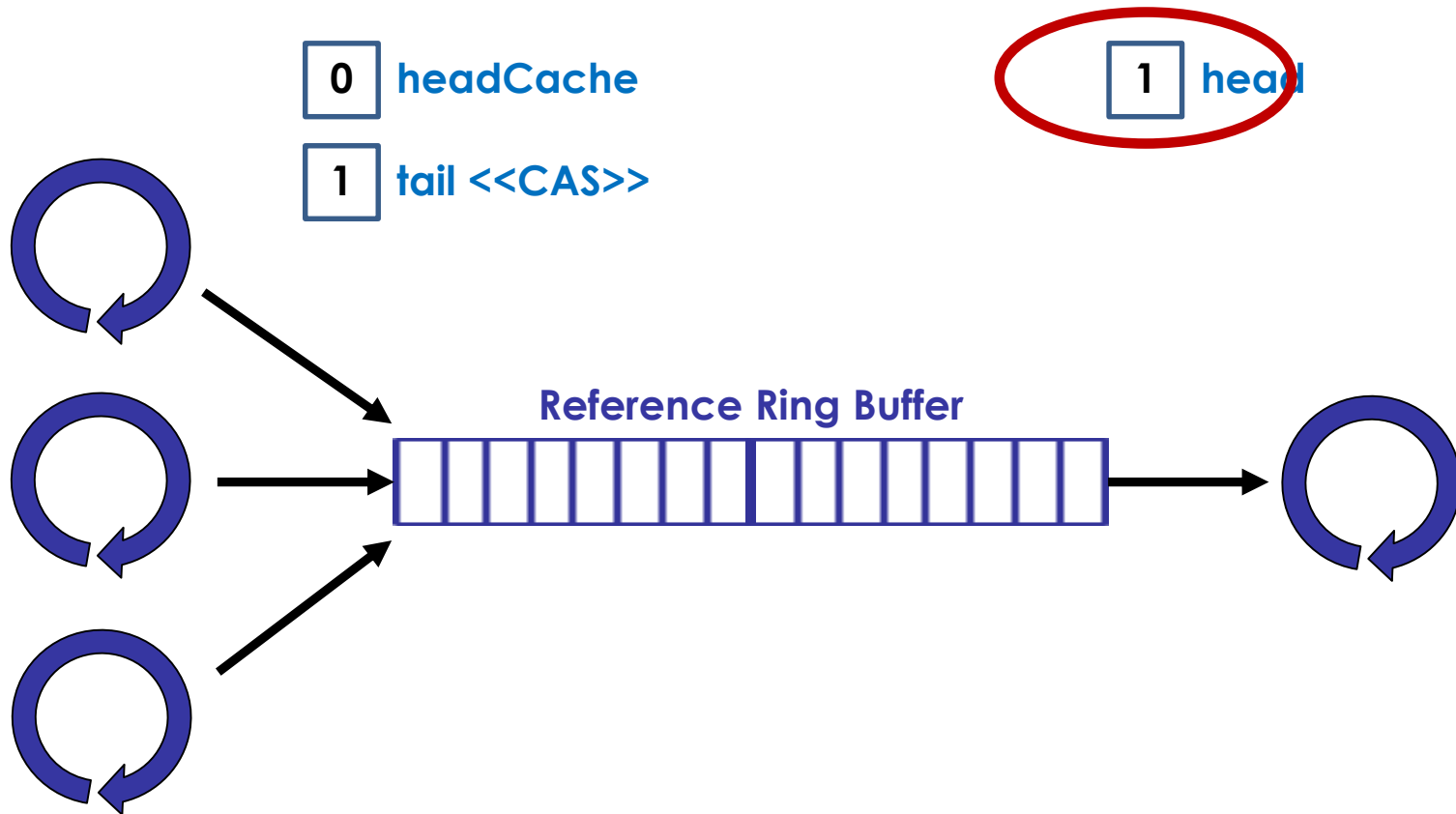
Influences: Lamport + Fast Flow

ManyToOneConcurrentArrayQueue



Influences: Lamport + Fast Flow

ManyToOneConcurrentArrayQueue



Influences: Lamport + Fast Flow

Burst Length = 1: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	167	189
ConcurrentLinkedQueue	1	281	361
	2	381	559
	3	444	705
Disruptor 3.3.2	1	201	263
	2	256	373
	3	282	459
ManyToOneConcurrentArrayQueue	1	173	198
	2	238	319
	3	259	392

Burst Length = 1: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	167	189
ConcurrentLinkedQueue	1	281	361
	2	381	559
	3	444	705
Disruptor 3.3.2	1	201	263
	2	256	373
	3	282	459
ManyToOneConcurrentArrayQueue	1	173	198
	2	238	319
	3	259	392

Burst Length = 100: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	721	982
ConcurrentLinkedQueue	1	12,916	15,792
	2	25,132	35,136
	3	39,462	56,768
Disruptor 3.3.2	1	7,030	8,192
	2	15,159	21,184
	3	31,597	54,976
ManyToOneConcurrentArrayQueue	1	3,570	3,932
	2	12,926	16,480
	3	26,685	51,072

Burst Length = 100: RTT (ns)

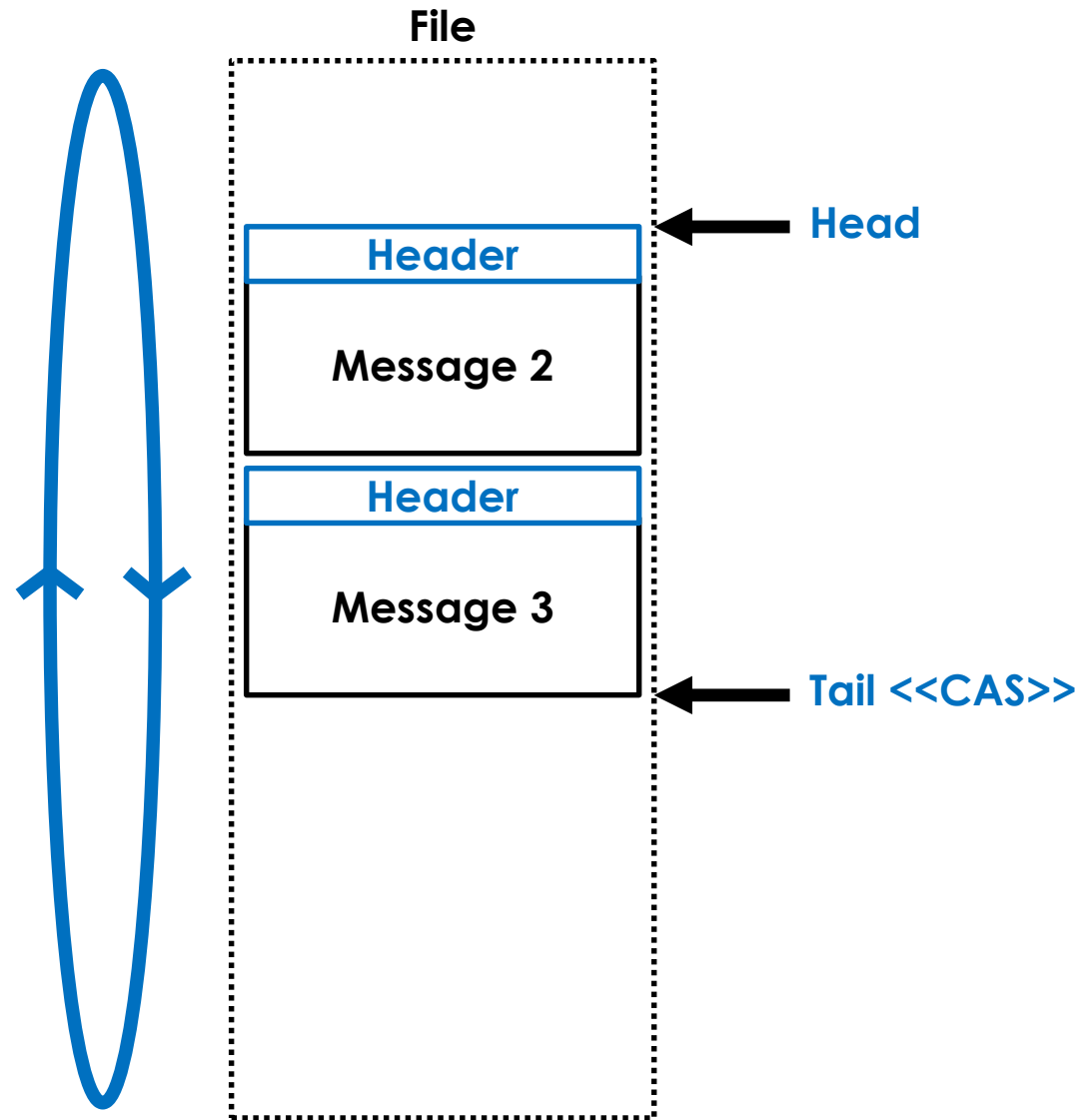
	Prod #	Mean	99%
Baseline (Wait-free)	1	721	982
ConcurrentLinkedQueue	1	12,916	15,792
	2	25,132	35,136
	3	39,462	56,768
Disruptor 3.3.2	1	7,030	8,192
	2	15,159	21,184
	3	31,597	54,976
ManyToOneConcurrentArrayQueue	1	3,570	3,932
	2	12,926	16,480
	3	26,685	51,072

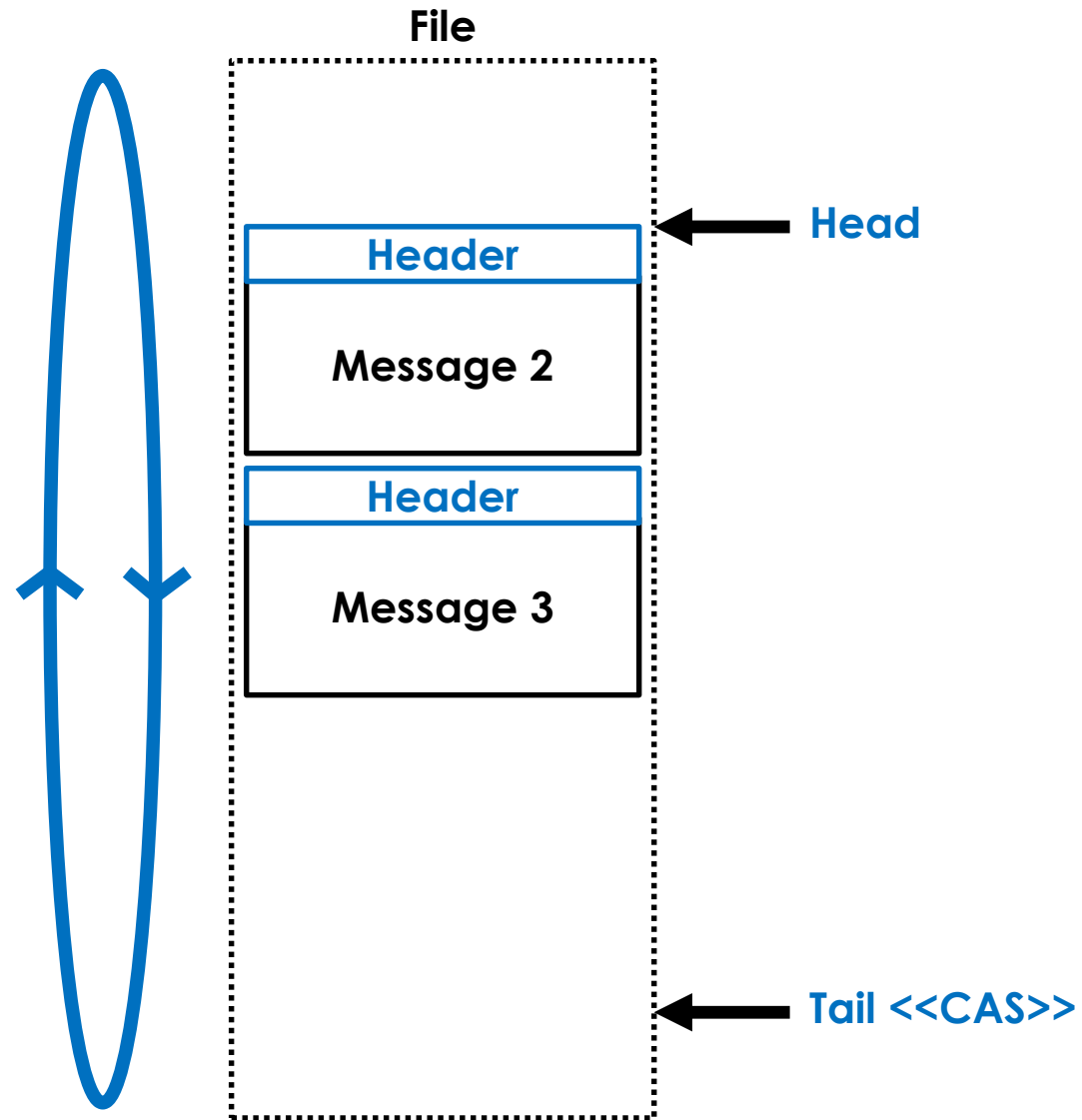
BUT!!!

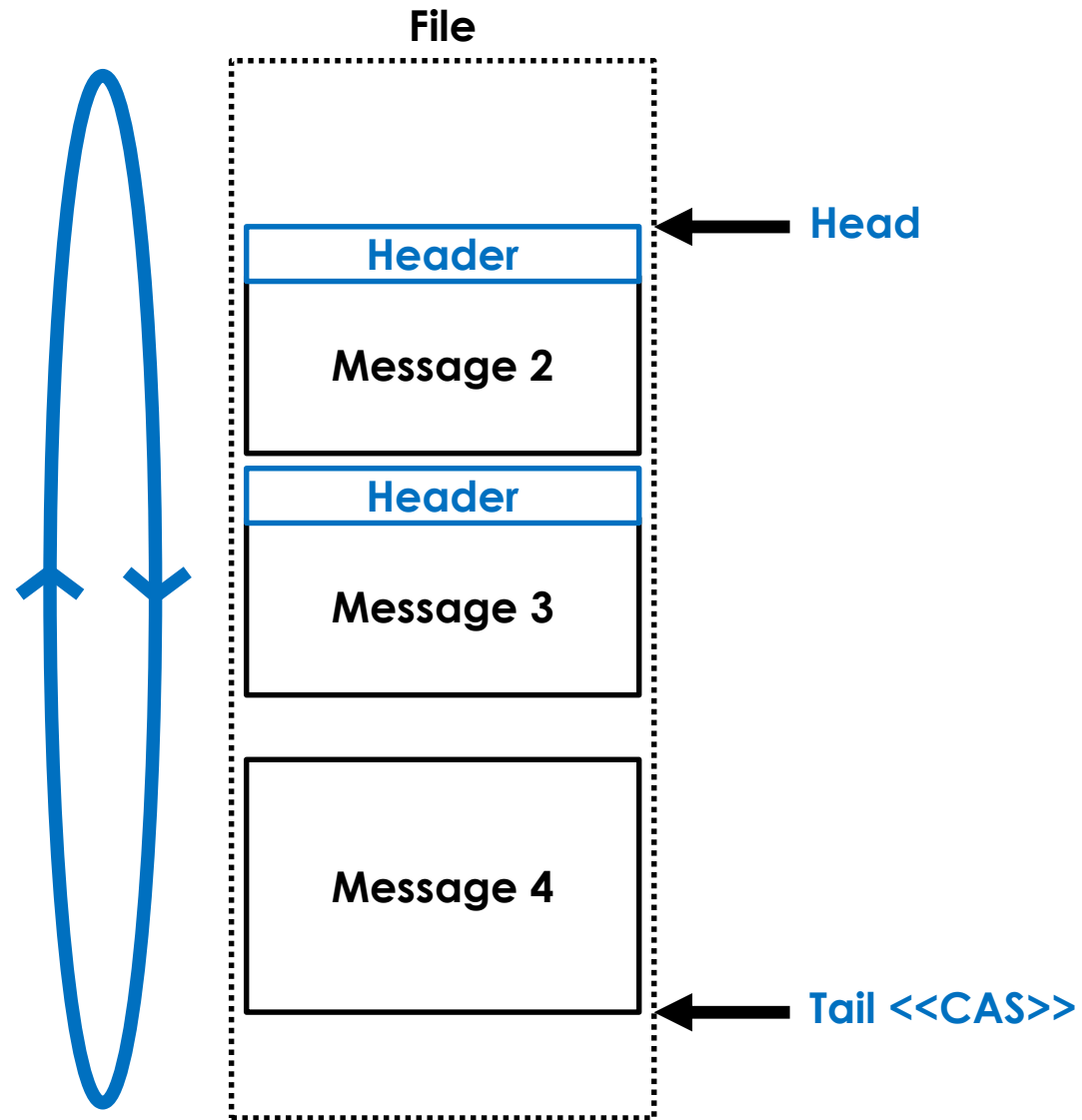
***Disruptor has single producer
and batch methods***

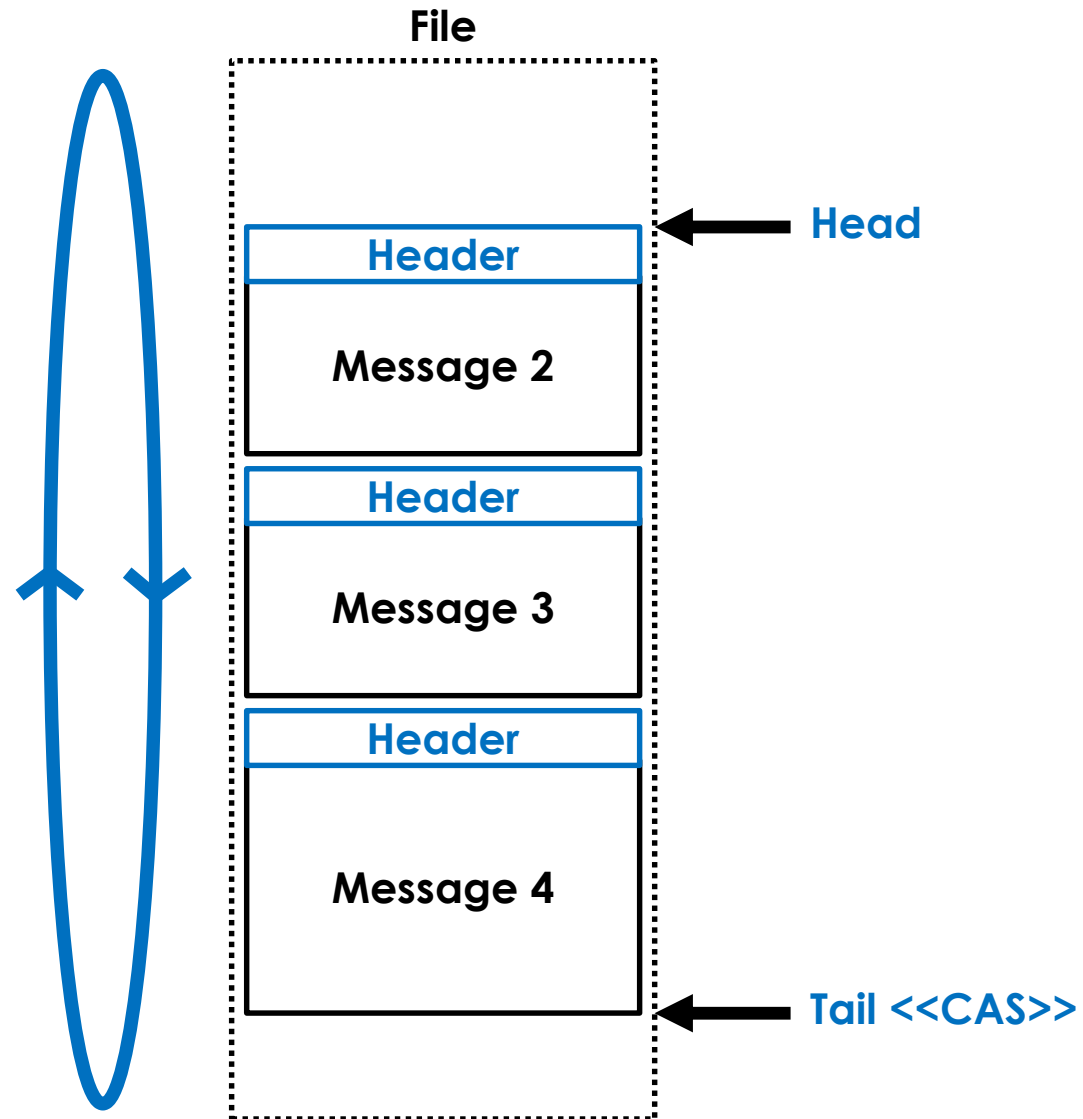
Inter-Process FIFOs

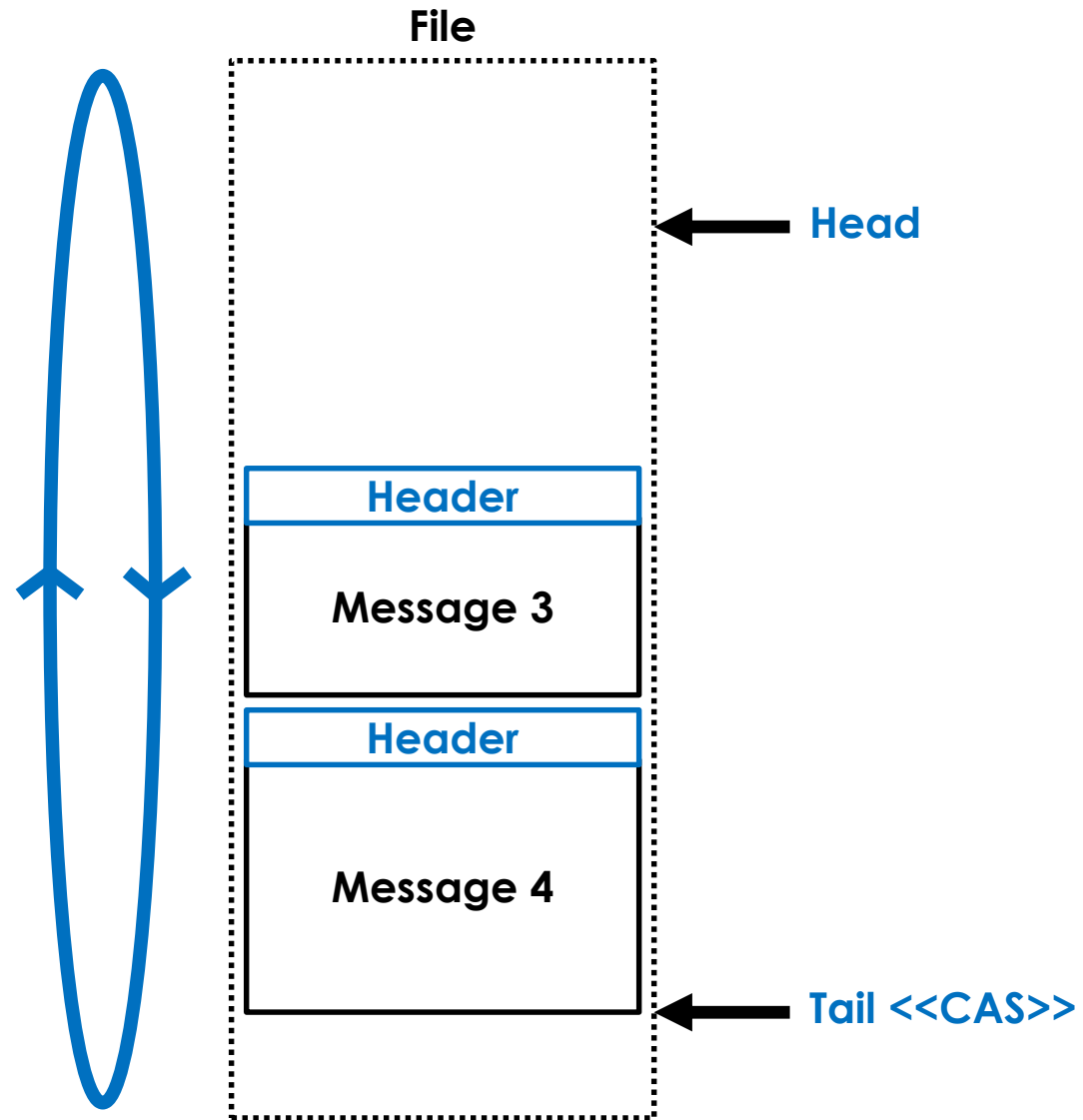
ManyToOneRingBuffer

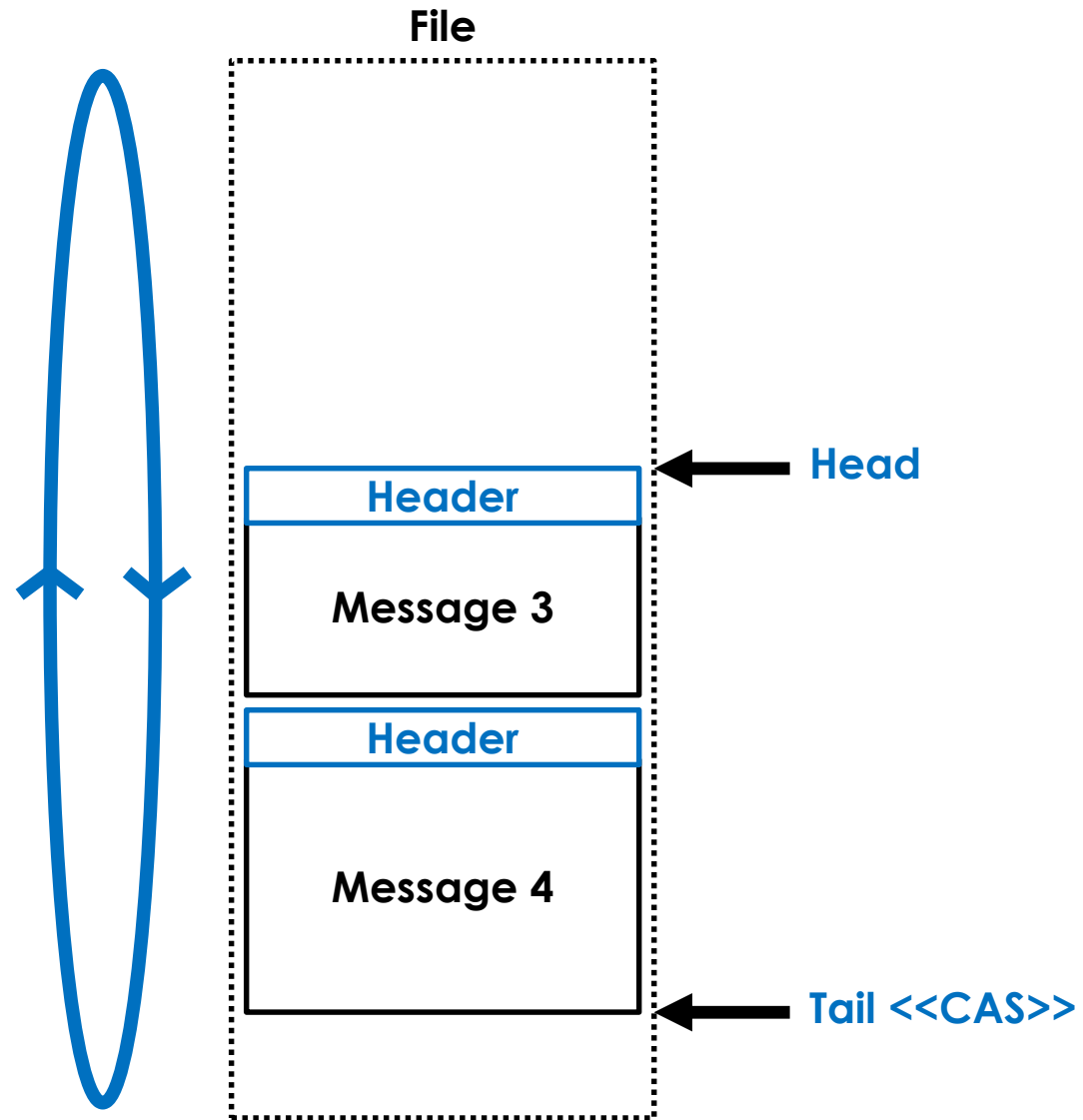




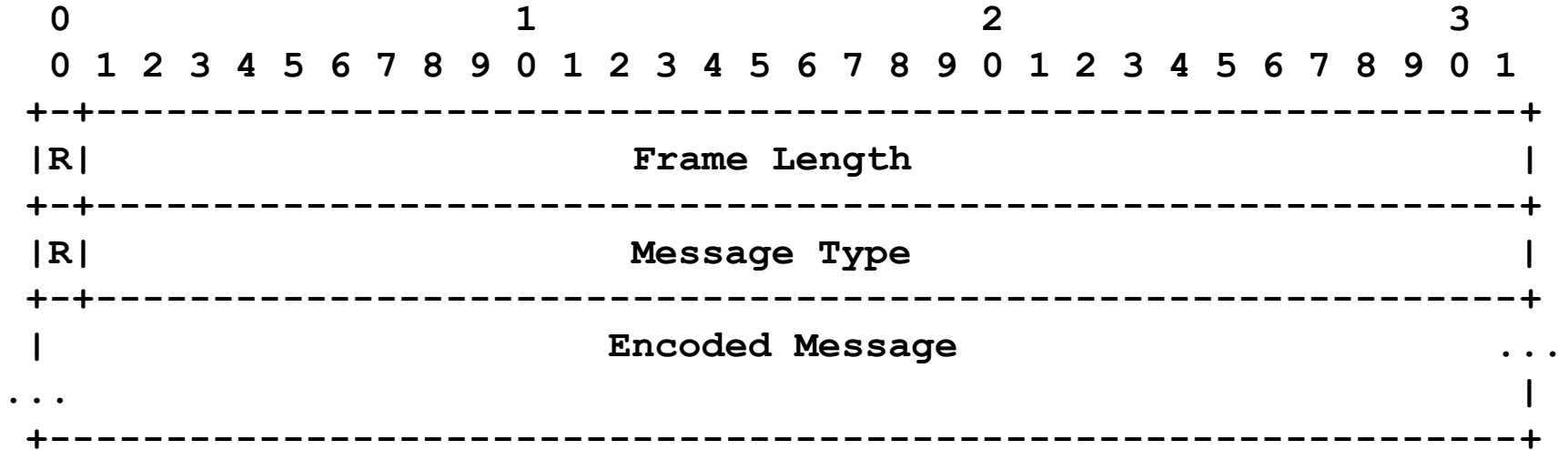






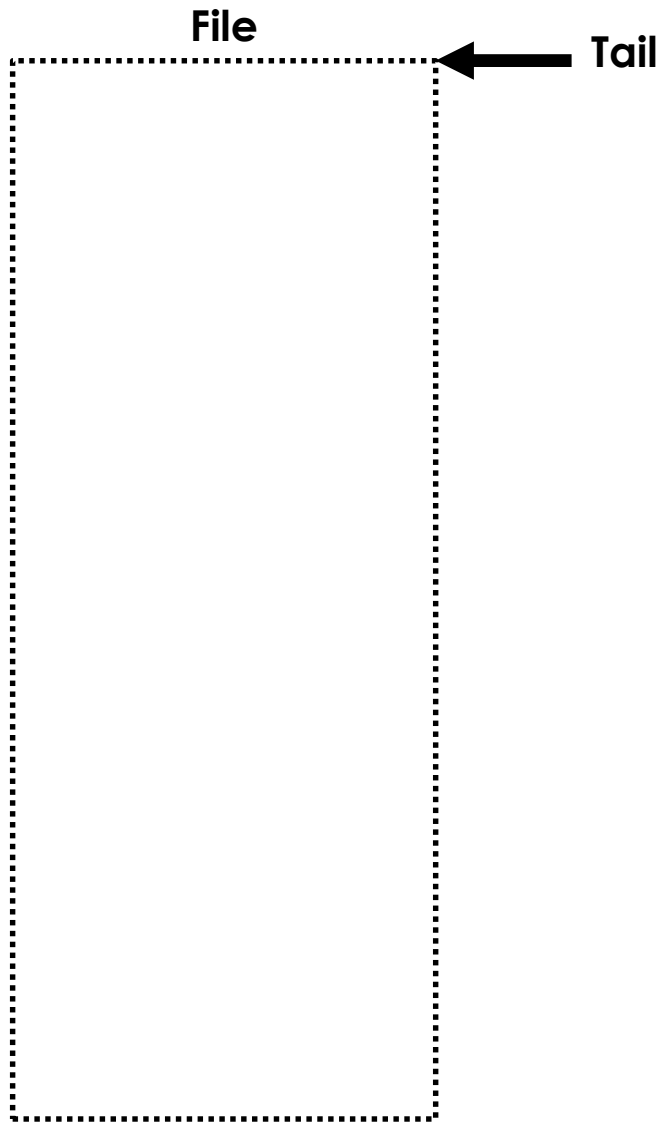


MPSC Ring Buffer Message Header

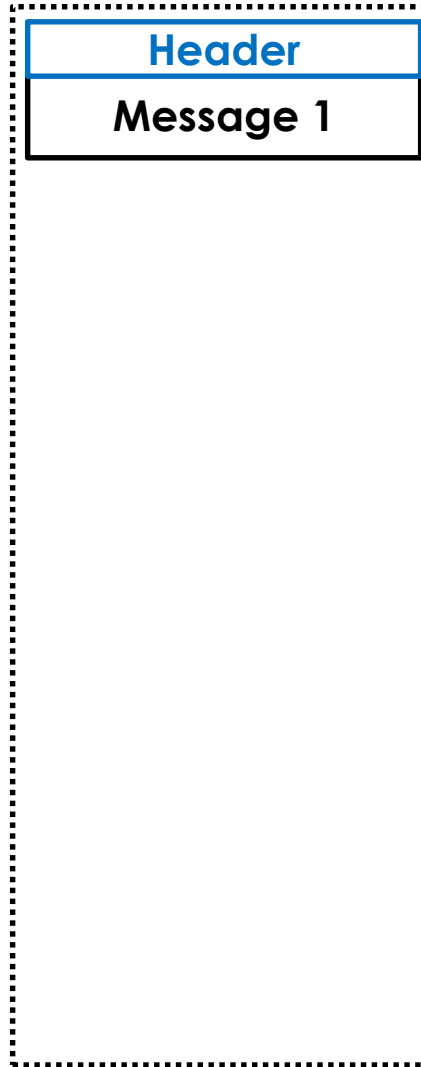


***Takes a throughput hit on zero'ing
memory but latency is good***

Aeron IPC



File

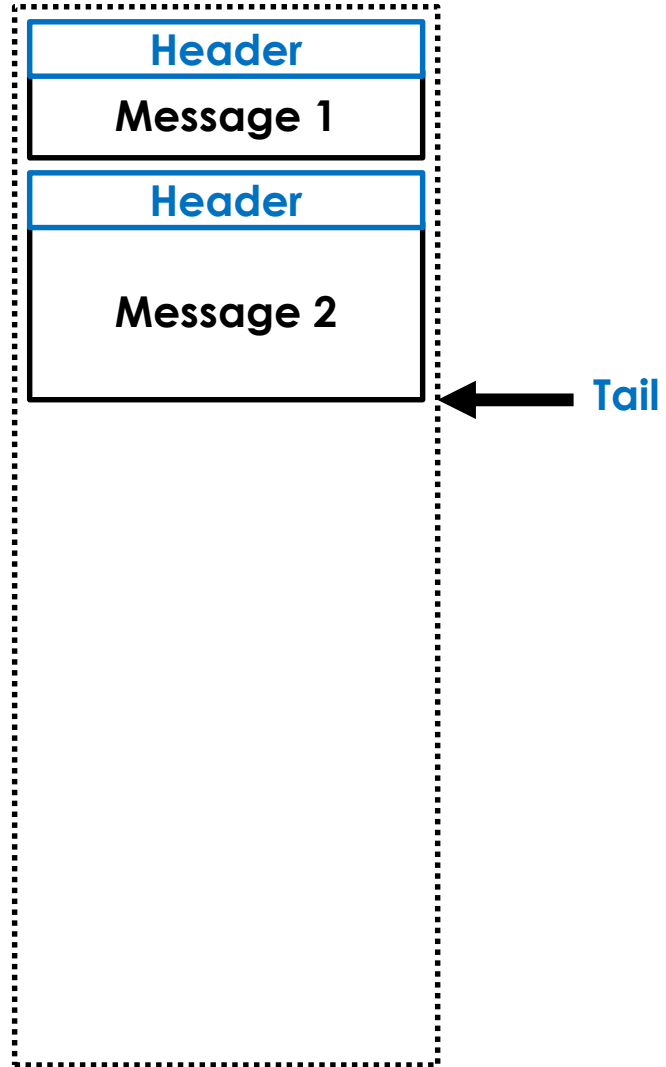


Header

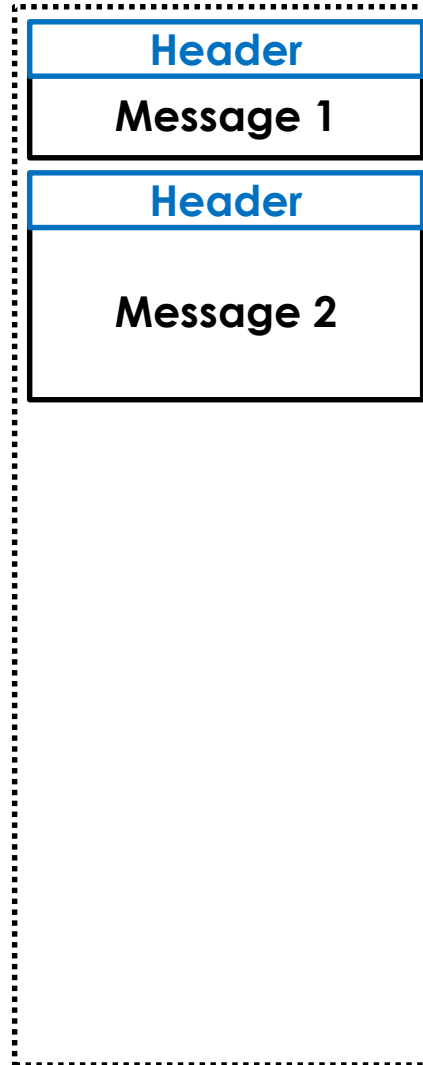
Message 1

Tail

File

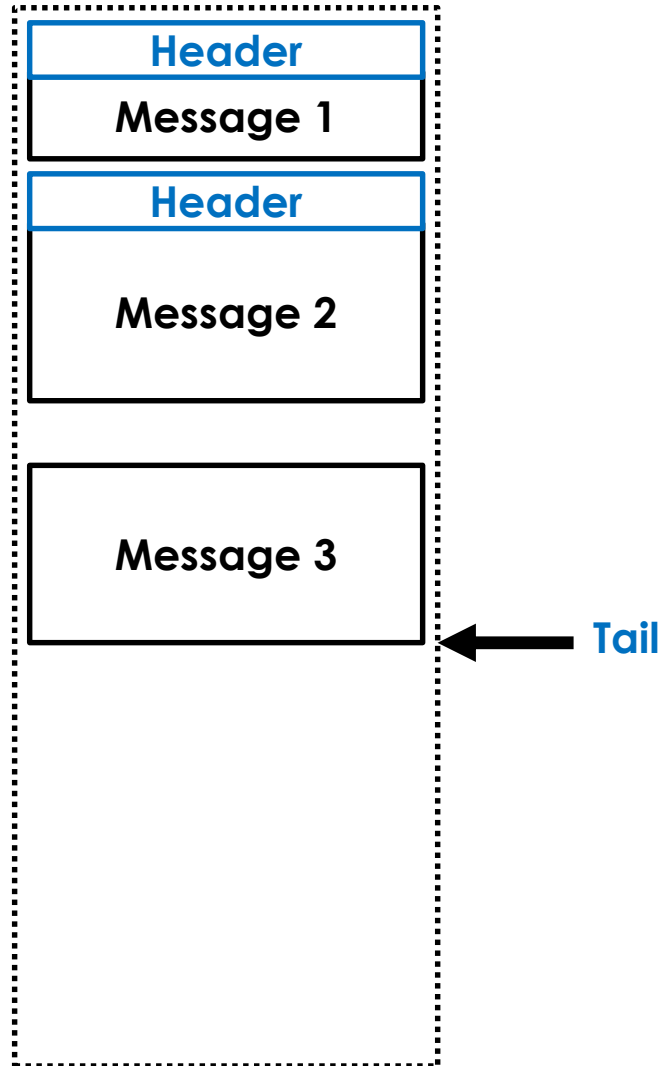


File

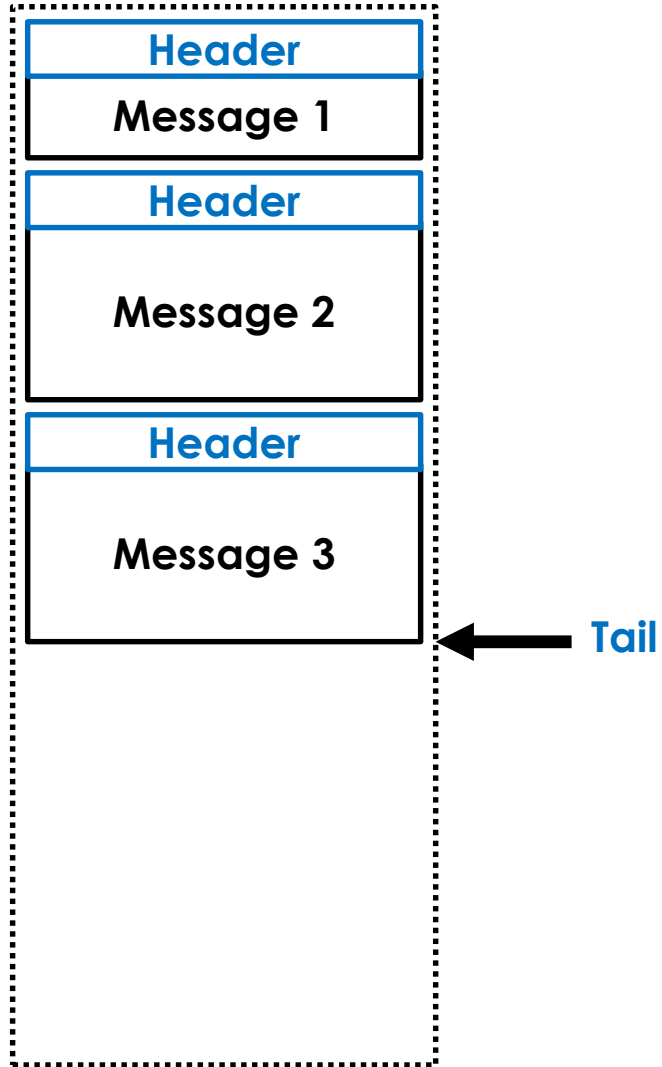


← Tail

File



File

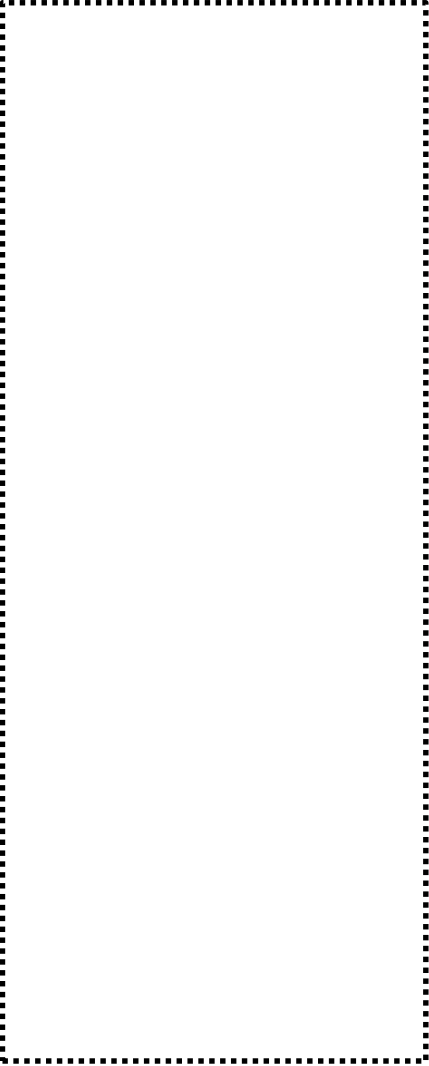


***One big file that
goes on forever?***

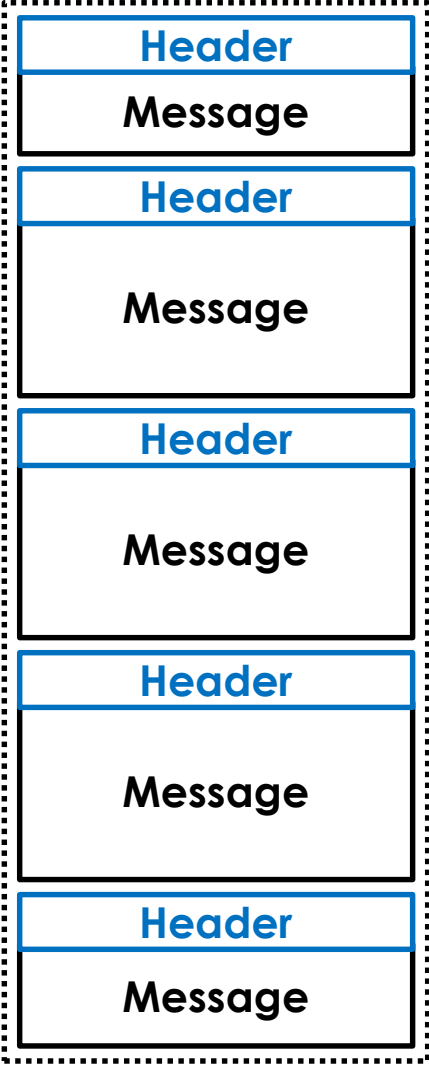
No!!!

***Page faults, page cache churn,
VM pressure, ...***

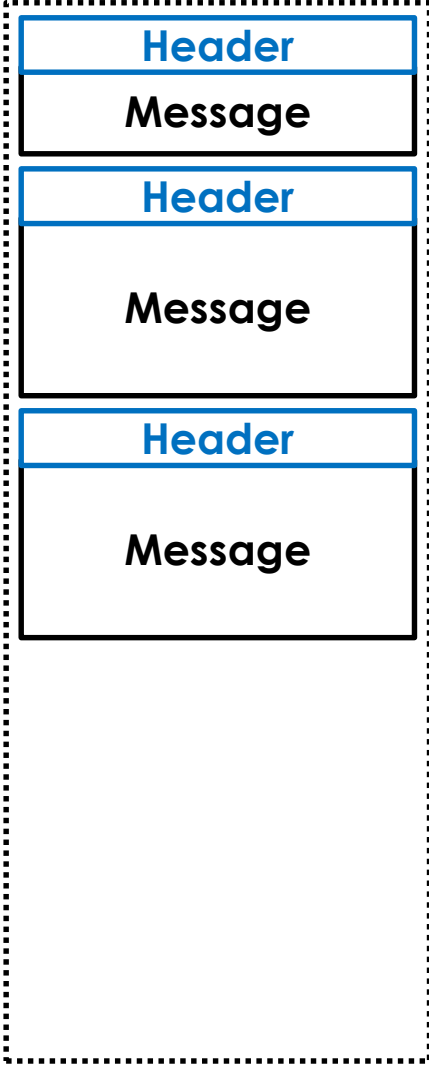
Clean



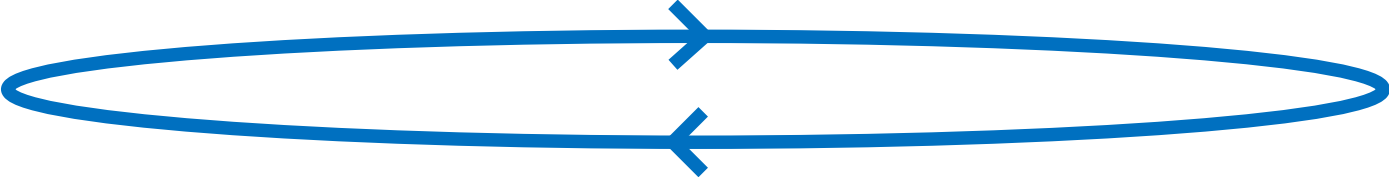
Dirty



Active

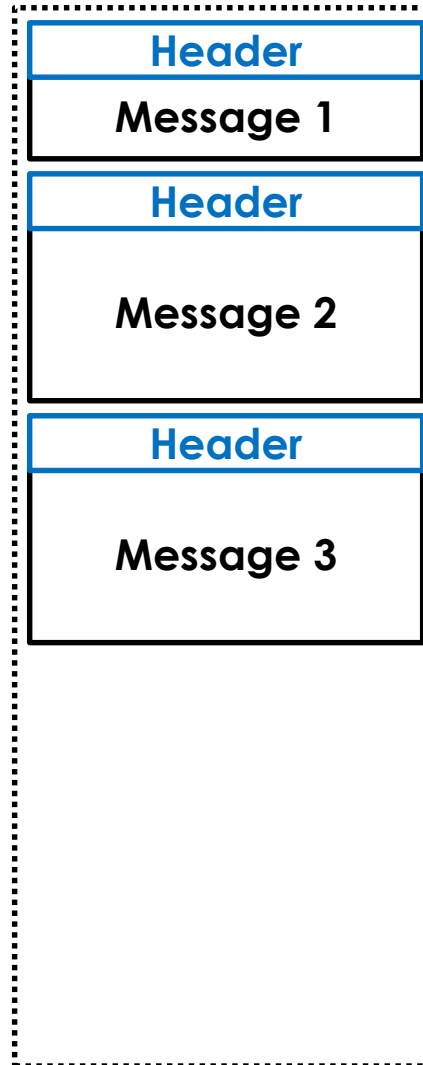


← Tail



Do publishers need a CAS?

File

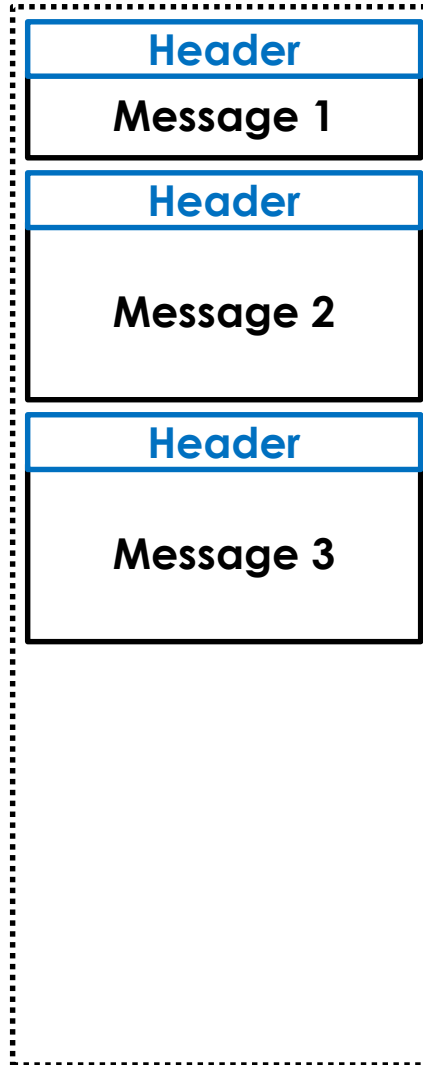


Tail <<XADD>>

Message X

Message Y

File

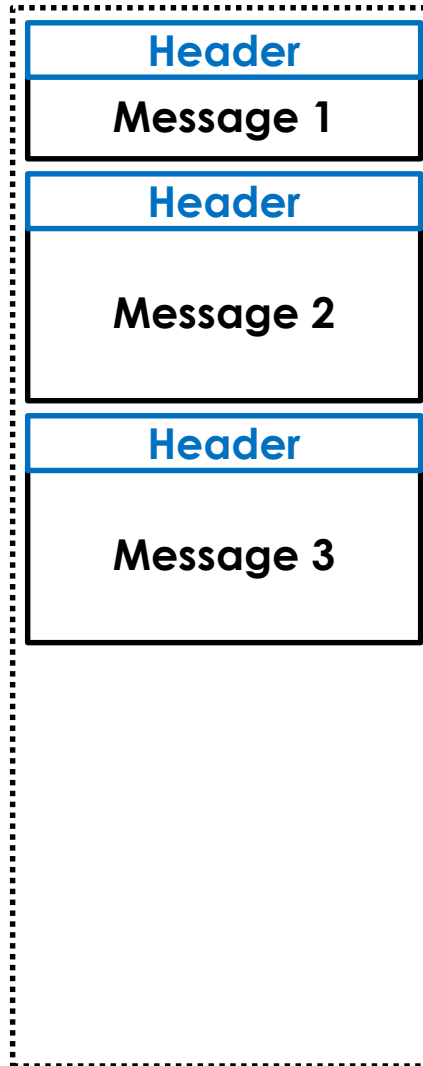


Message X

Message Y

← Tail <<XADD>>

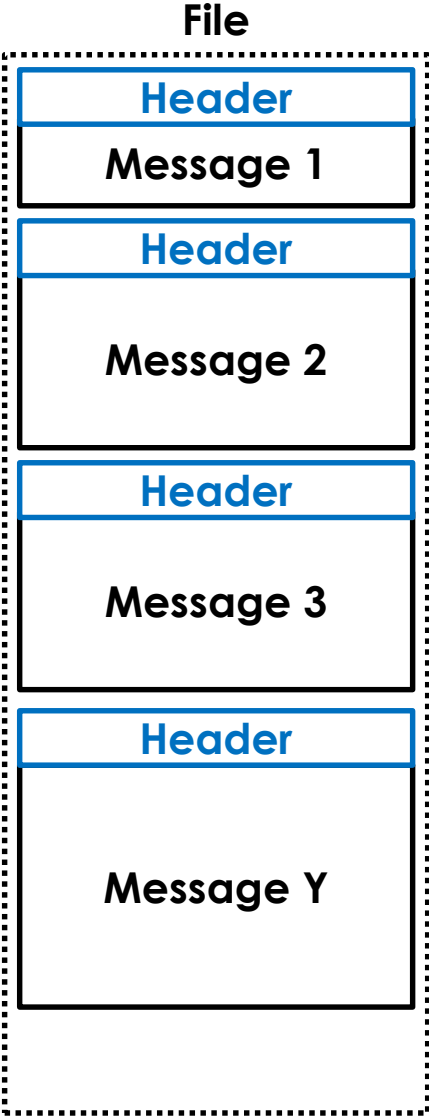
File



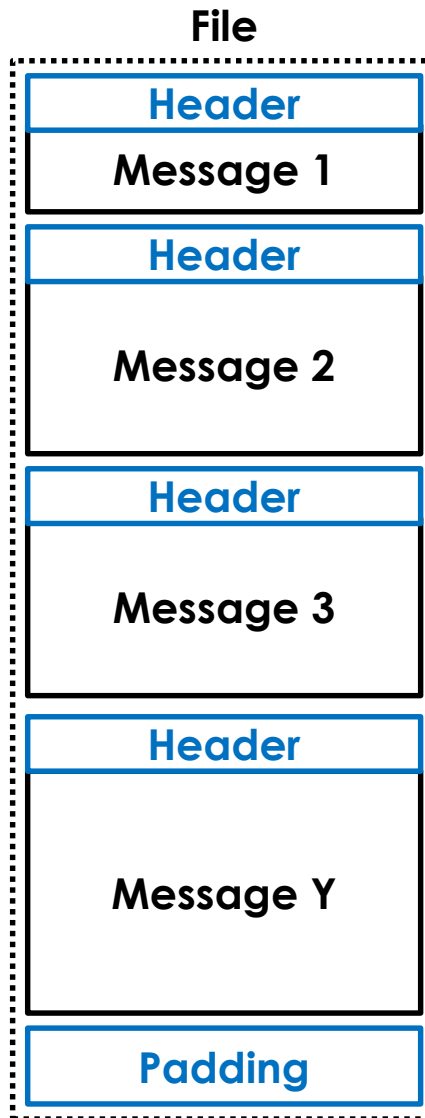
Message X

Message Y

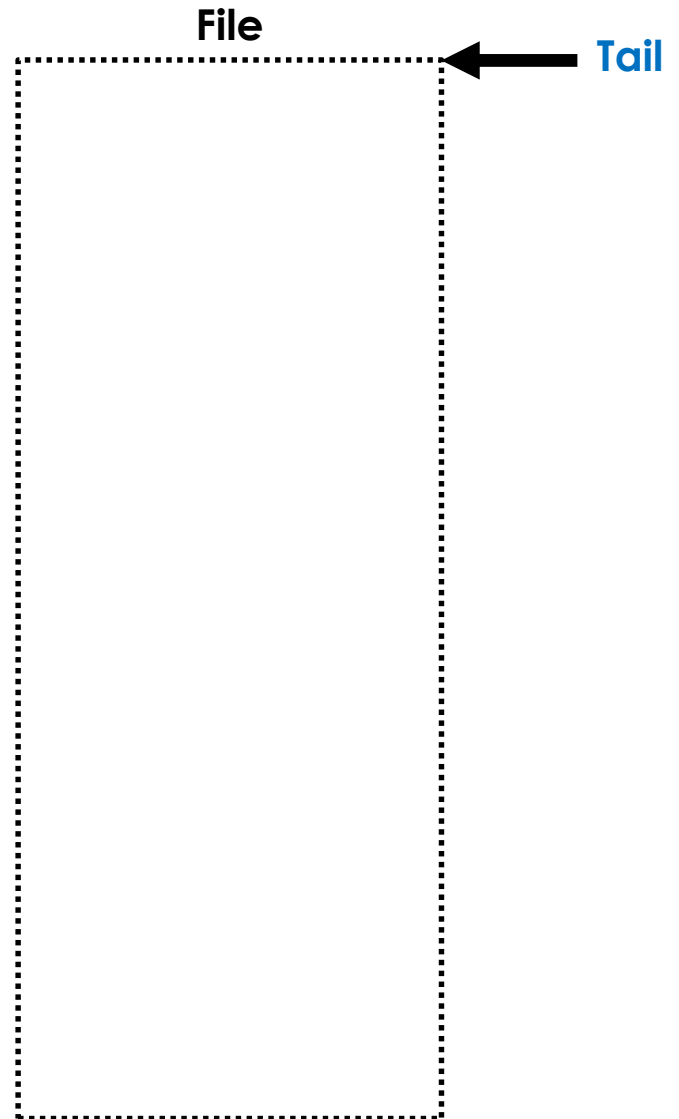
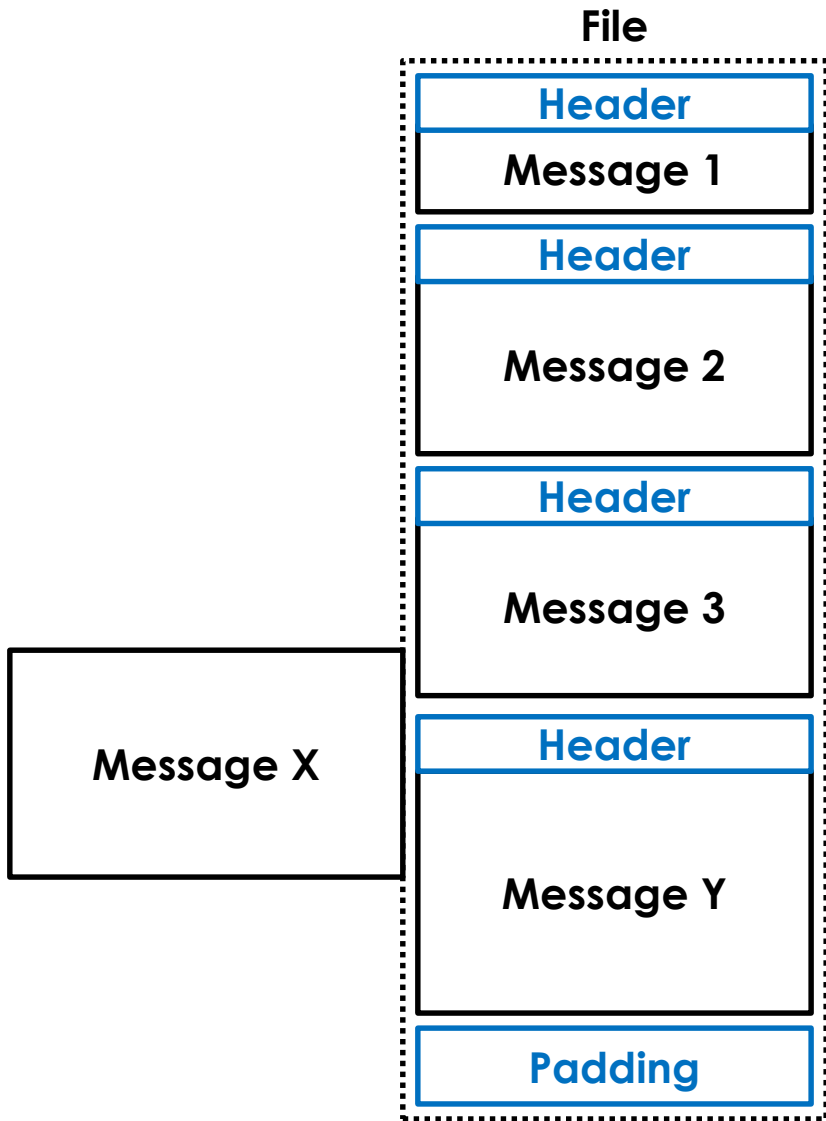
← Tail



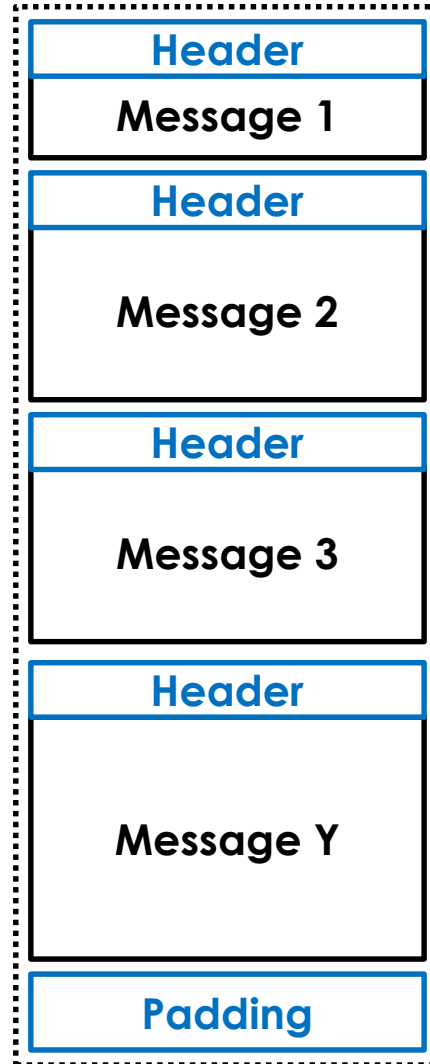
← Tail



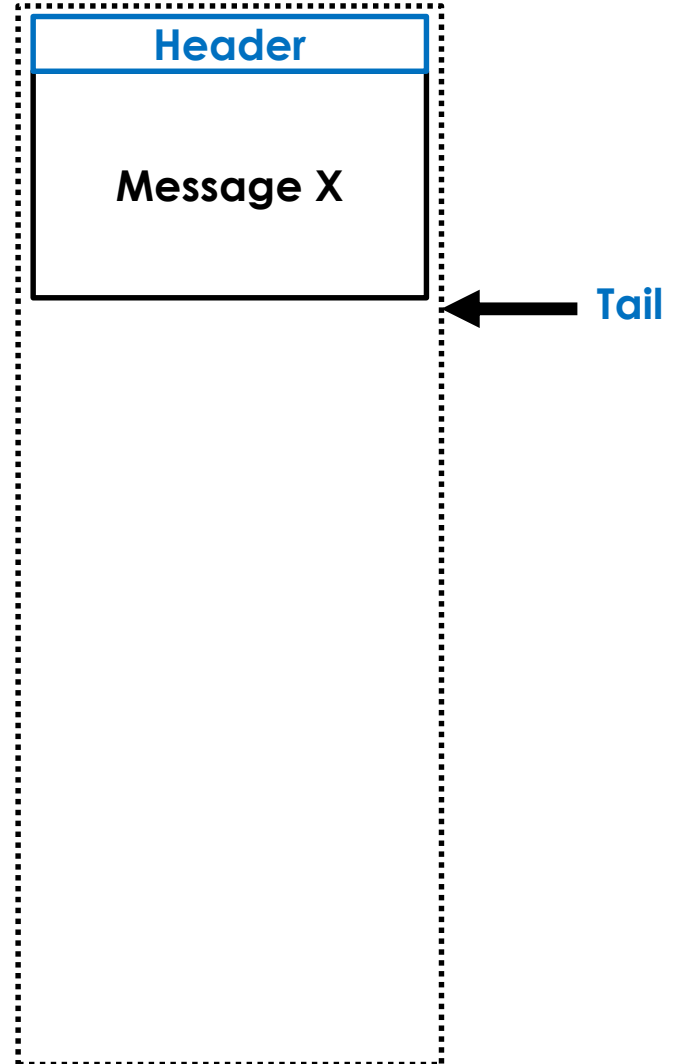
← Tail



File



File



***Have a background thread do
zero'ing and flow control***

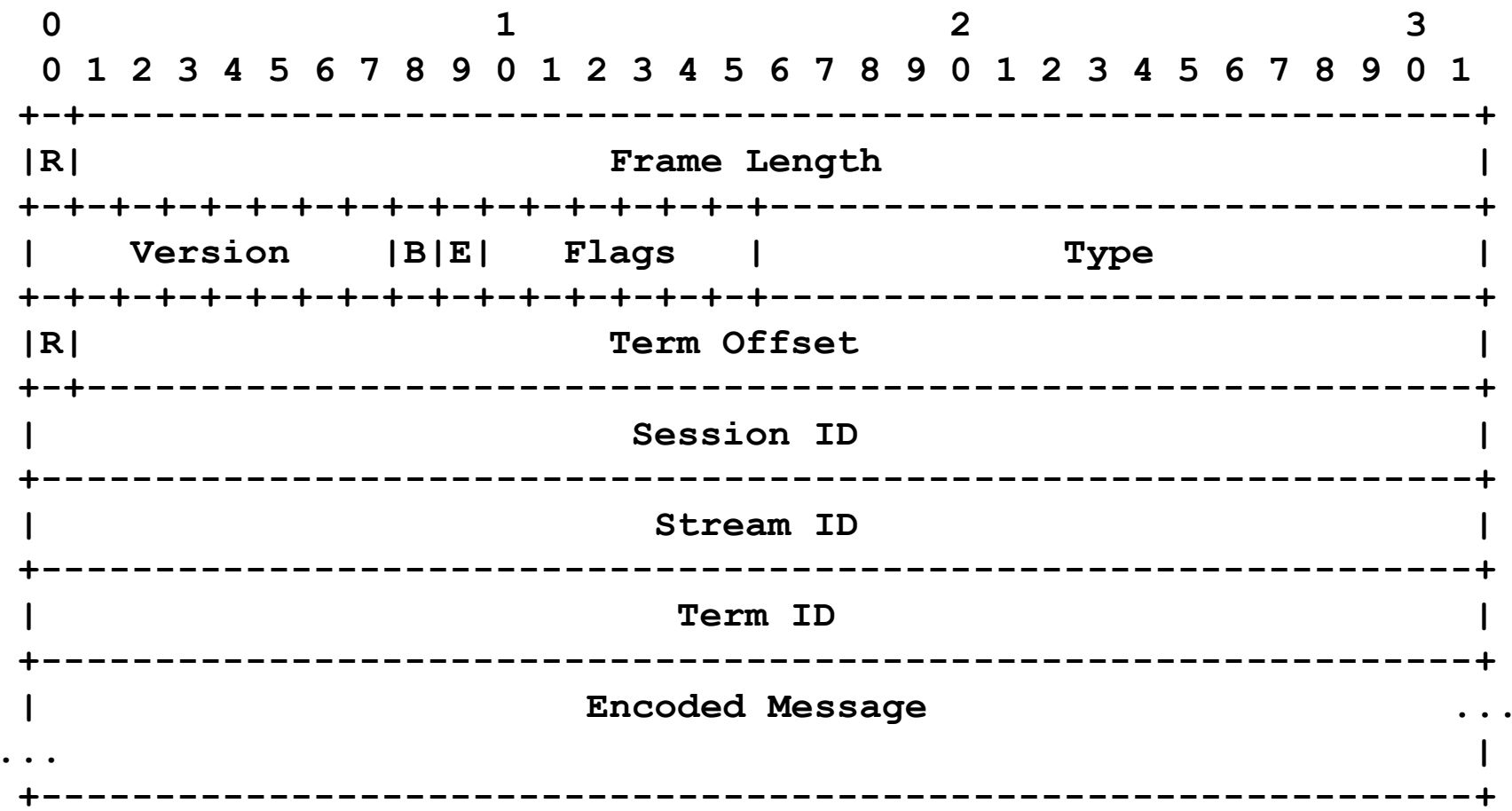
Burst Length = 1: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	167	189
ConcurrentLinkedQueue	1	281	361
	2	381	559
	3	444	705
ManyToOneRingBuffer	1	170	194
	2	256	279
	3	283	340
Aeron IPC	1	192	210
	2	251	286
	3	290	342

Burst Length = 1: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	167	189
ConcurrentLinkedQueue	1	281	361
	2	381	559
	3	444	705
ManyToOneRingBuffer	1	170	194
	2	256	279
	3	283	340
Aeron IPC	1	192	210
	2	251	286
	3	290	342

Aeron Data Message Header



Burst Length = 100: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	721	982
ConcurrentLinkedQueue	1	12,916	15,792
	2	25,132	35,136
	3	39,462	56,768
ManyToOneRingBuffer	1	3,773	3,992
	2	11,286	13,200
	3	27,255	34,432
Aeron IPC	1	4,436	4,632
	2	7,623	8,224
	3	10,825	13,872

Burst Length = 100: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	721	982
ConcurrentLinkedQueue	1	12,916	15,792
	2	25,132	35,136
	3	39,462	56,768
ManyToOneRingBuffer	1	3,773	3,992
	2	11,286	13,200
	3	27,255	34,432
Aeron IPC	1	4,436	4,632
	2	7,623	8,224
	3	10,825	13,872

Less “False Sharing”

-

Inlined data vs Reference Array

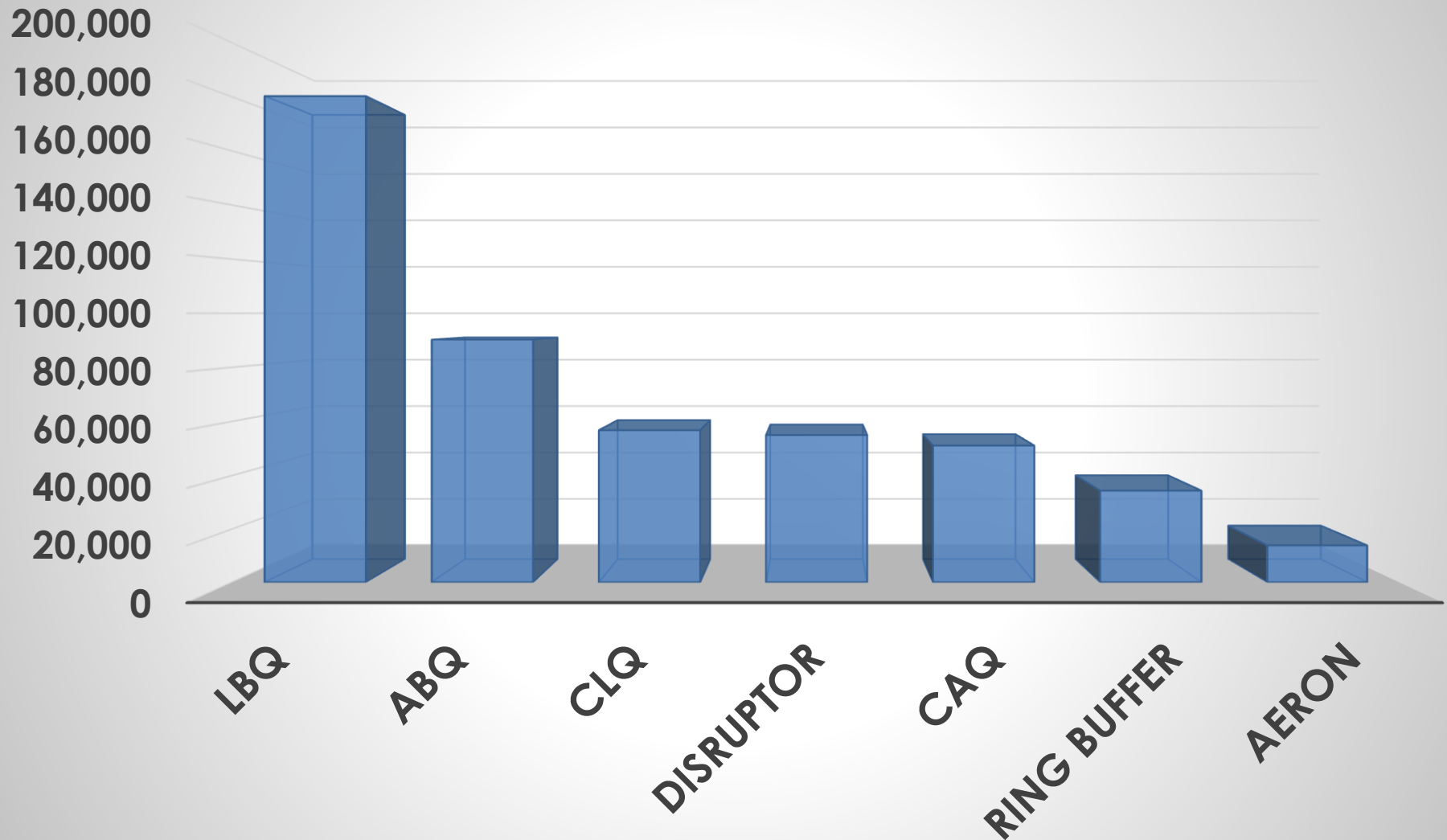
Less “Card Marking”

***Is avoiding the spinning “CAS”
loop is a major step forward?***

Burst Length = 100: RTT (ns)

	Prod #	Mean	99%
Baseline (Wait-free)	1	721	982
ConcurrentLinkedQueue	1	12,916	15,792
	2	25,132	35,136
	3	39,462	56,768
ManyToOneRingBuffer	1	3,773	3,992
	2	11,286	13,200
	3	27,255	34,432
Aeron IPC	1	4,436	4,632
	2	7,623	8,224
	3	10,825	13,872

Burst Length = 100: 99% RTT (ns)



Logging is a Messaging Problem

What design should we use?

***5. Where can we go
Next?***

C vs Java



C vs Java



- **Spin Wait Loops**
 - `Runtime.onSpinWait()`

C vs Java



- **Spin Wait Loops**
 - `Runtime.onSpinWait()`
- **Data Dependent Loads**
 - **Heap Aggregates - ObjectLayout**
 - **Stack Allocation – Value Types**

C vs Java



- **Spin Wait Loops**
 - `Runtime.onSpinWait()`
- **Data Dependent Loads**
 - Heap Aggregates - `ObjectLayout`
 - Stack Allocation – Value Types
- **Memory Copying**
 - Baseline against `memcpy()` for differing alignment

C vs Java



- **Spin Wait Loops**
 - `Runtime.onSpinWait()`
- **Data Dependent Loads**
 - Heap Aggregates - `ObjectLayout`
 - Stack Allocation – Value Types
- **Memory Copying**
 - Baseline against `memcpy()` for differing alignment
- **Queue Interface**
 - Break conflated concerns to reduce blocking actions

In closing...

2TB RAM and 100+ vcores!!!



X1 INSTANCE



INTEL XEON E7 V3

Where can I find the code?

<https://github.com/real-logic/benchmarks>

<https://github.com/real-logic/Agrona>

<https://github.com/real-logic/Aeron>

Questions?

Blog: <http://mechanical-sympathy.blogspot.com/>

Twitter: @mjpt777

***“Any intelligent fool can make things bigger,
more complex, and more violent.***

***It takes a touch of genius, and a lot of courage,
to move in the opposite direction.”***

- Albert Einstein