WELCOME

The Disruptor - High performance messaging with Java

Raffael Schmid

April 28, 2012

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

trivadis
makes IT easier.

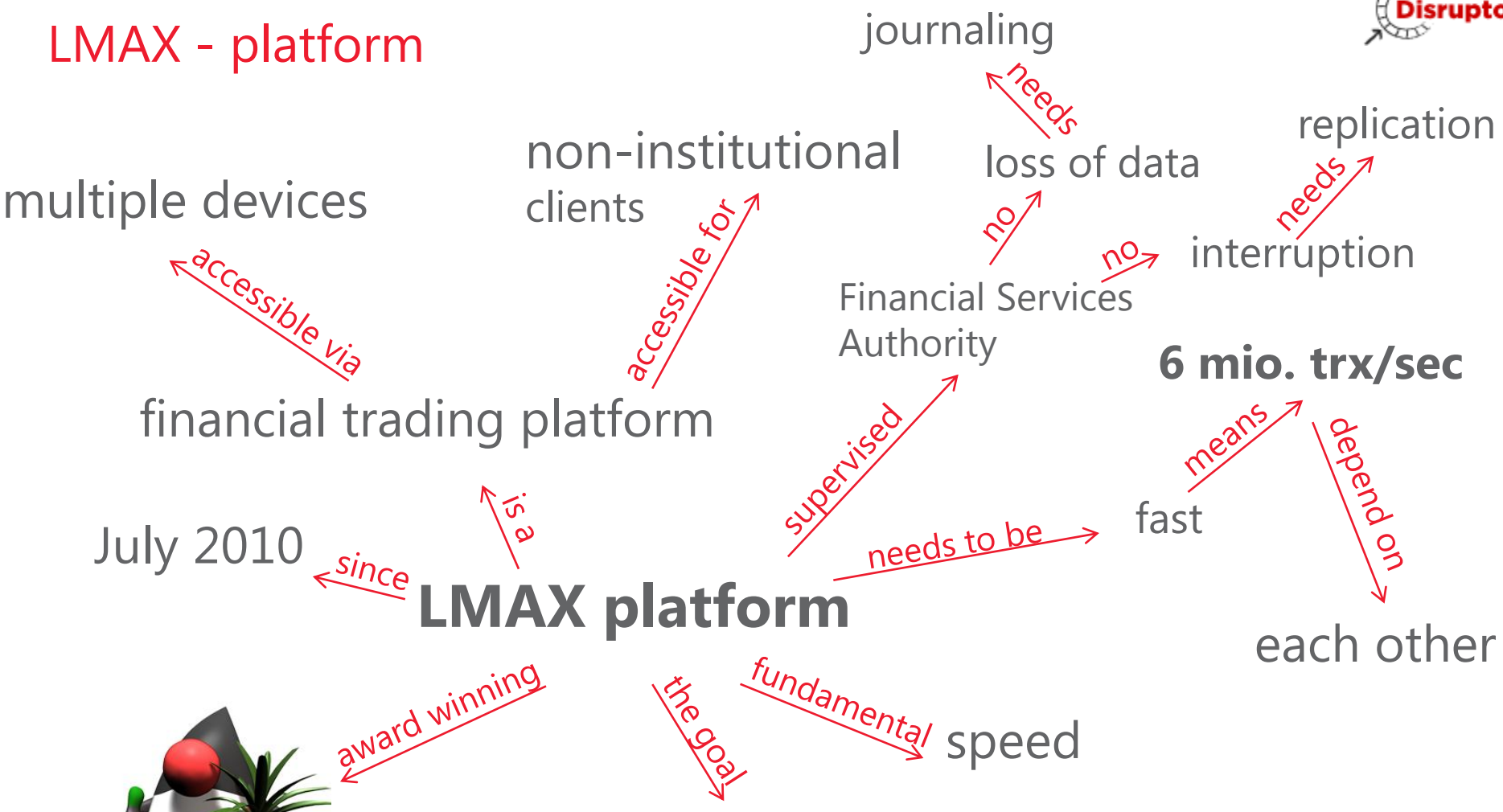
AGENDA

1. About LMAX
2. How to tackle 6’000’000 transactions/second
3. The Disruptor

Where does the Disruptor come from?

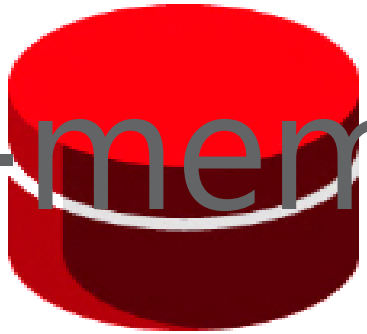
It was initially designed to setup a trading platform called LMAX.

LMAX - platform



How NOT to take 6'000'000 transactions per second?

in-memory



How to tackle 6 mio. transactions / second

Software Transactional Memory

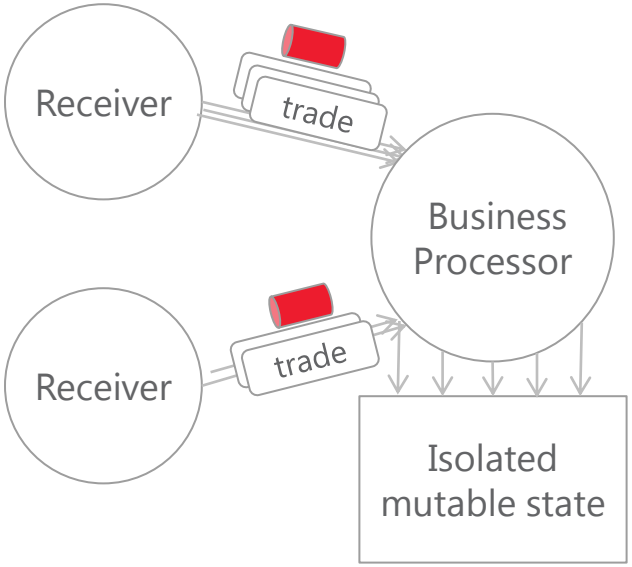
```
Ref<Integer> ref = new Ref<Integer>(0);

Atomic<Integer> atomic = new
                    Atomic<Integer>() {

  public Integer atomically() {

    final int inc = ref.get() + 1;

    ref.set(inc);

    return inc;

  }

};

atomic.execute();
```

- Turns Java heap into a transactional data set
- **A**tomicity
- **C**onsistency
- **I**solation: Serializable

does not scale

How to tackle 6 mio. transactions / second

Actors



- Message passing between threads
- Queueing of messages
- Single-threaded BusinessProcessor

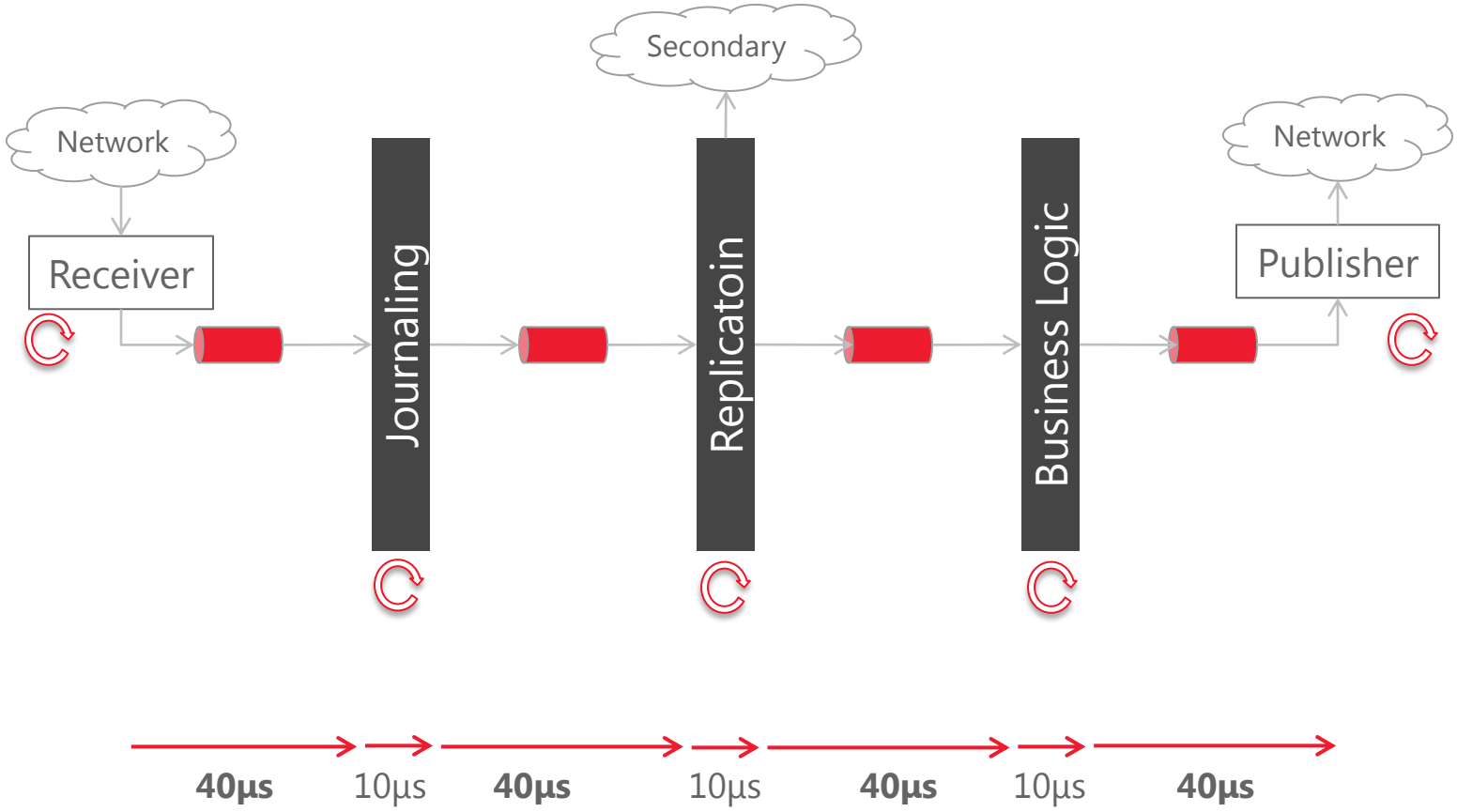
heavy contention on queues

no guarantee for the sequence of events

low predictable latency

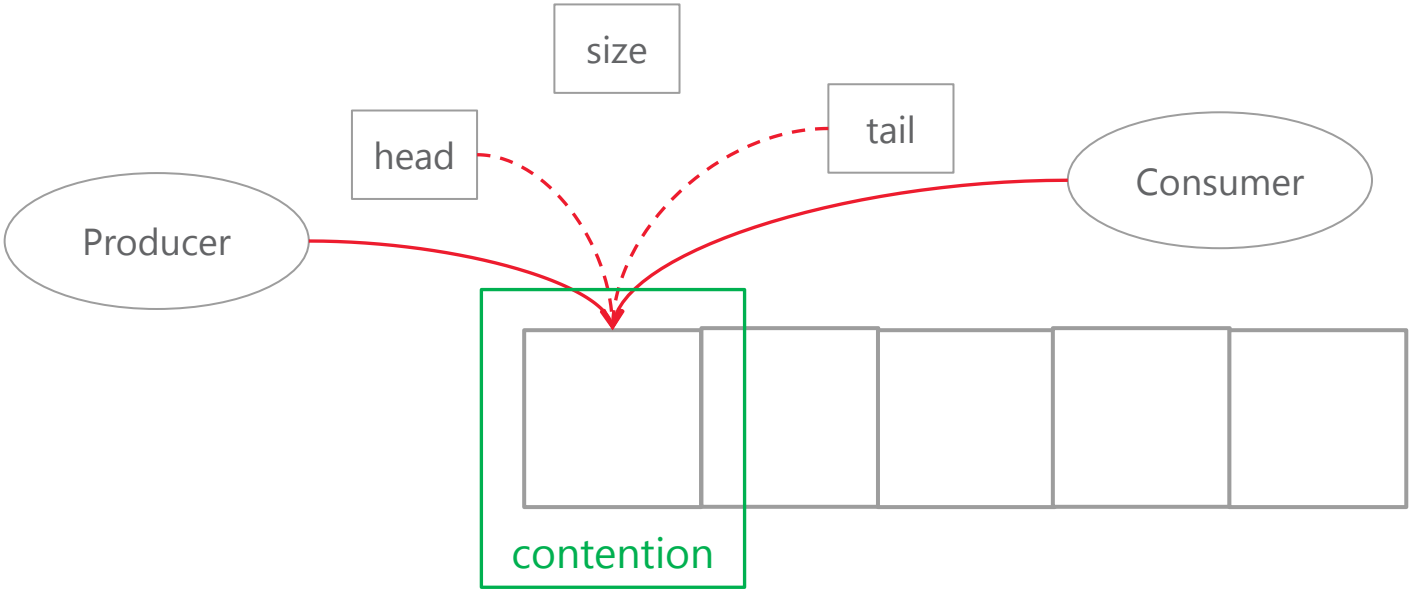
How to tackle 6 mio. transactions / second

Staged Event Driven Architecture



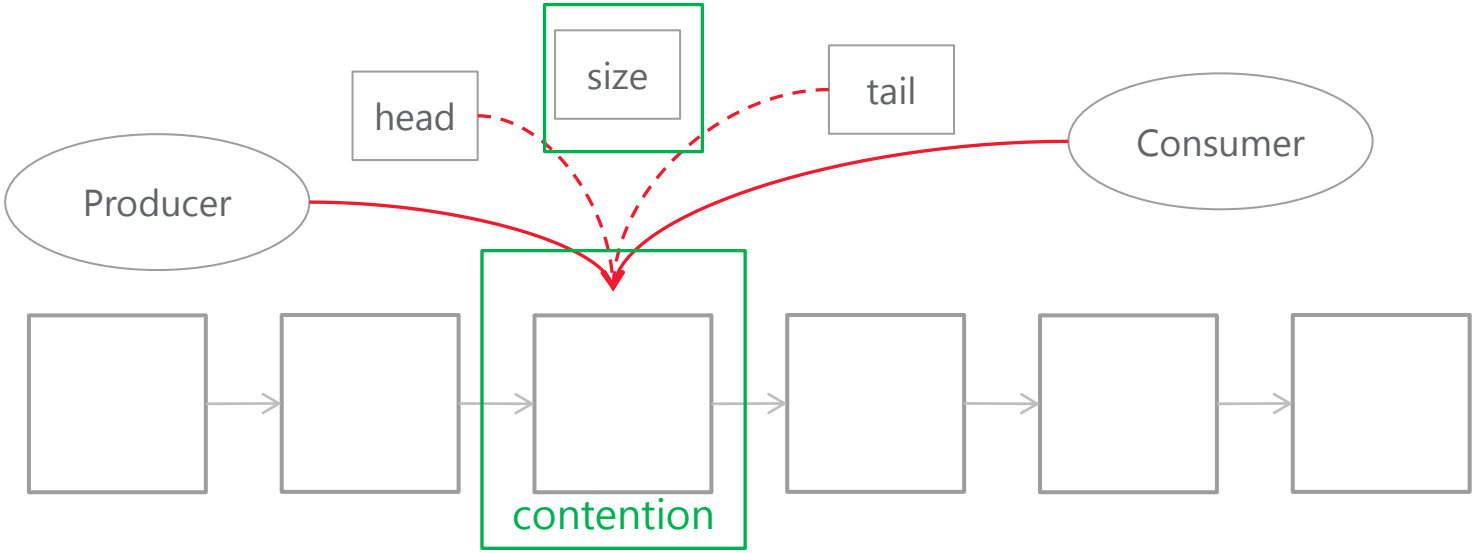
Why queues suck – Array backed

- Queues are never in steady state → **contention** on last or first element
- Head, tail may occupy same cache line → false sharing



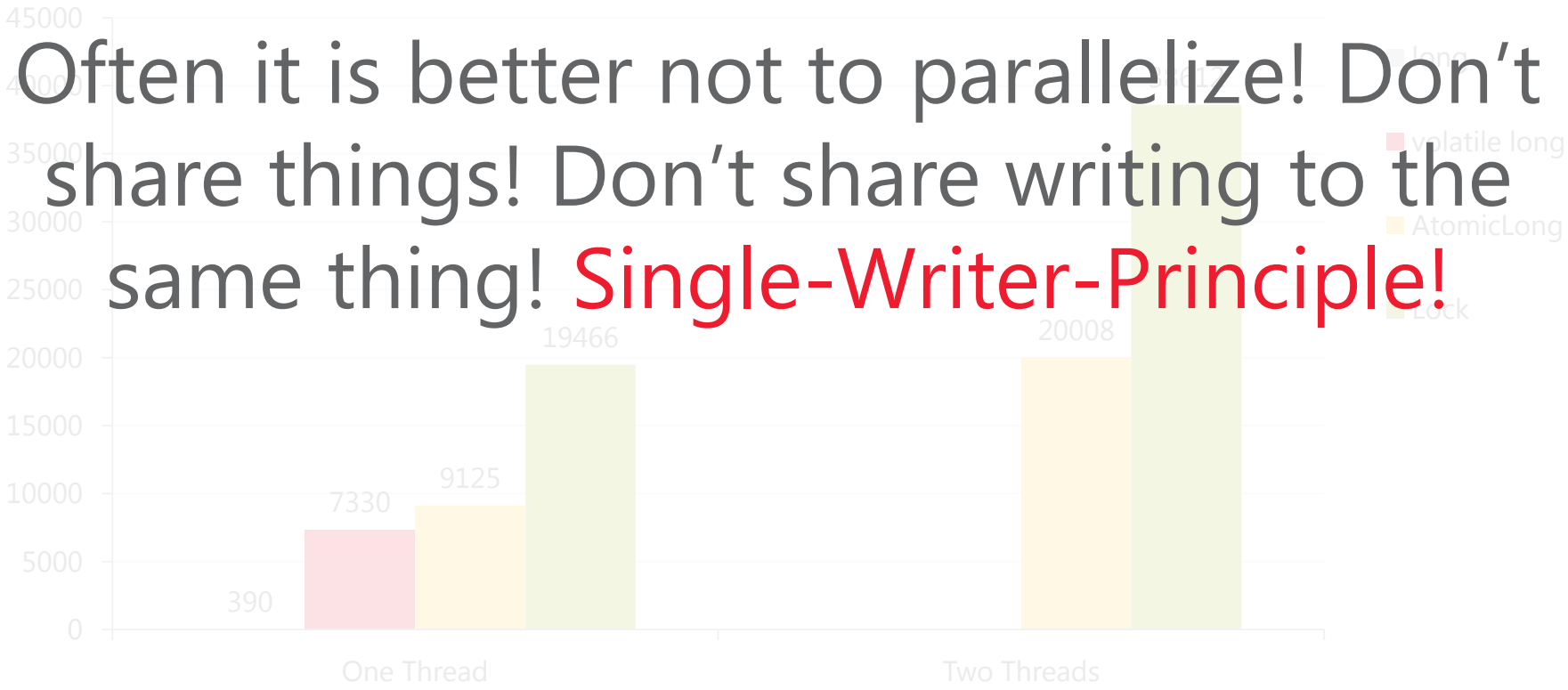
Why queues suck – LinkedList

- Not contiguous → do not support striding (prefetching)
- Needs to be bounded → contention on the size variable



Costs of contention

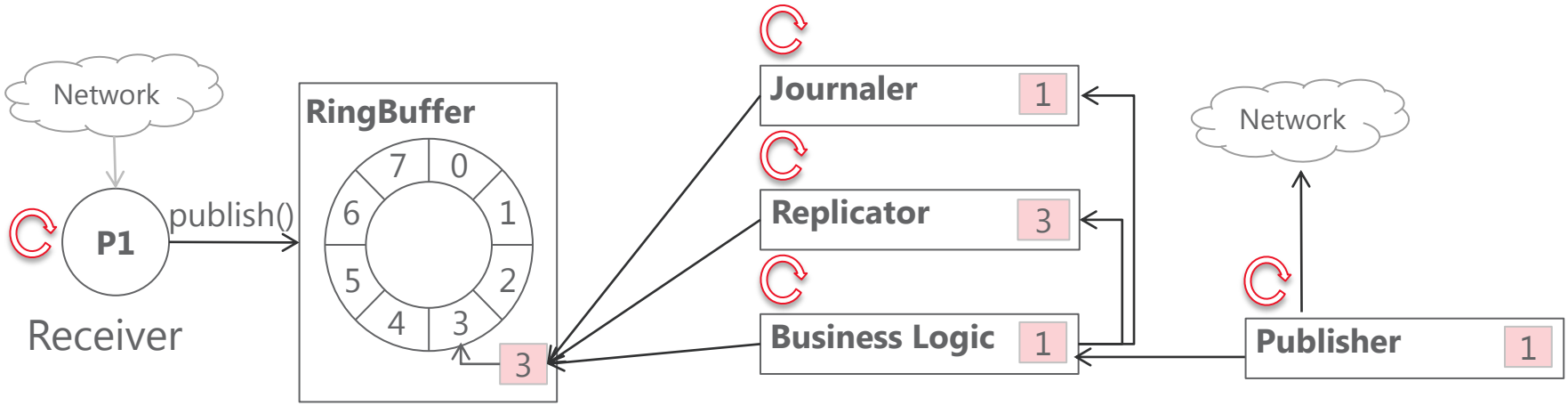
Incrementing a counter 500 000 000 times.



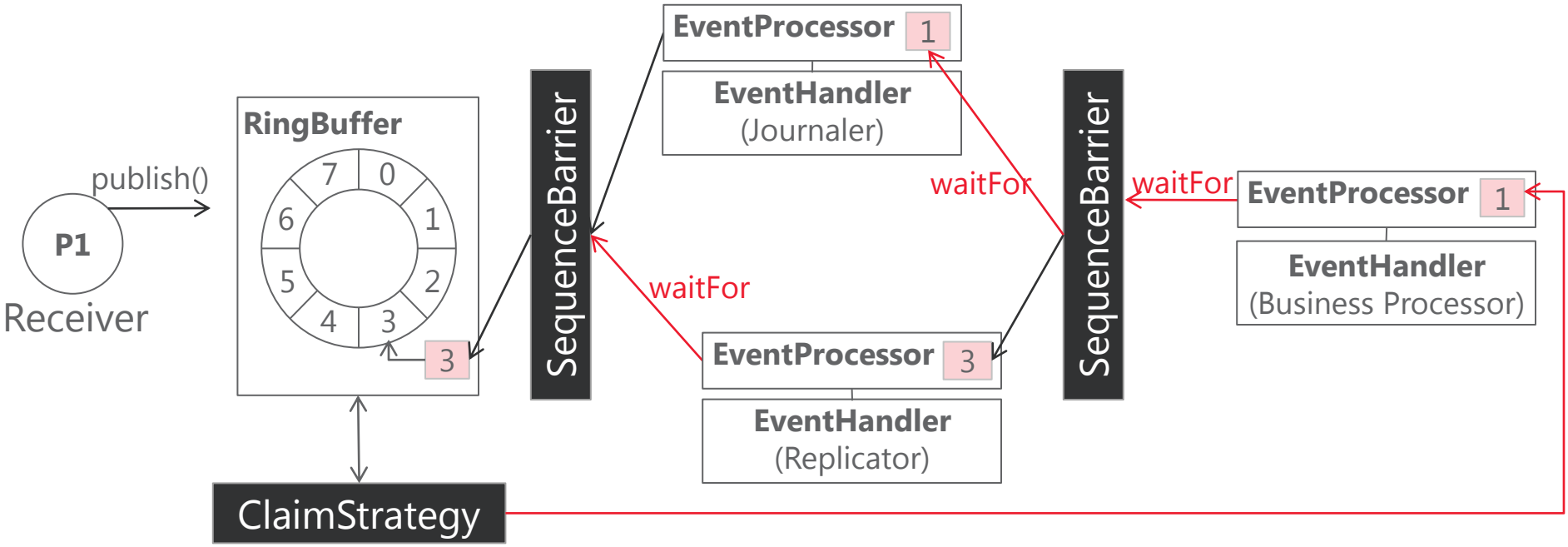
Often it is better not to parallelize! Don't share things! Don't share writing to the same thing! Single-Writer-Principle!

Results in milliseconds, run on a Intel Nehalem 2.93GHz

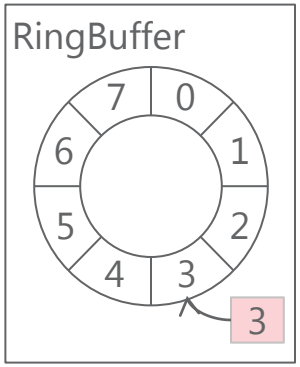
The Disruptor – contention-free design



The Disruptor – contention-free design



The Disruptor – RingBuffer

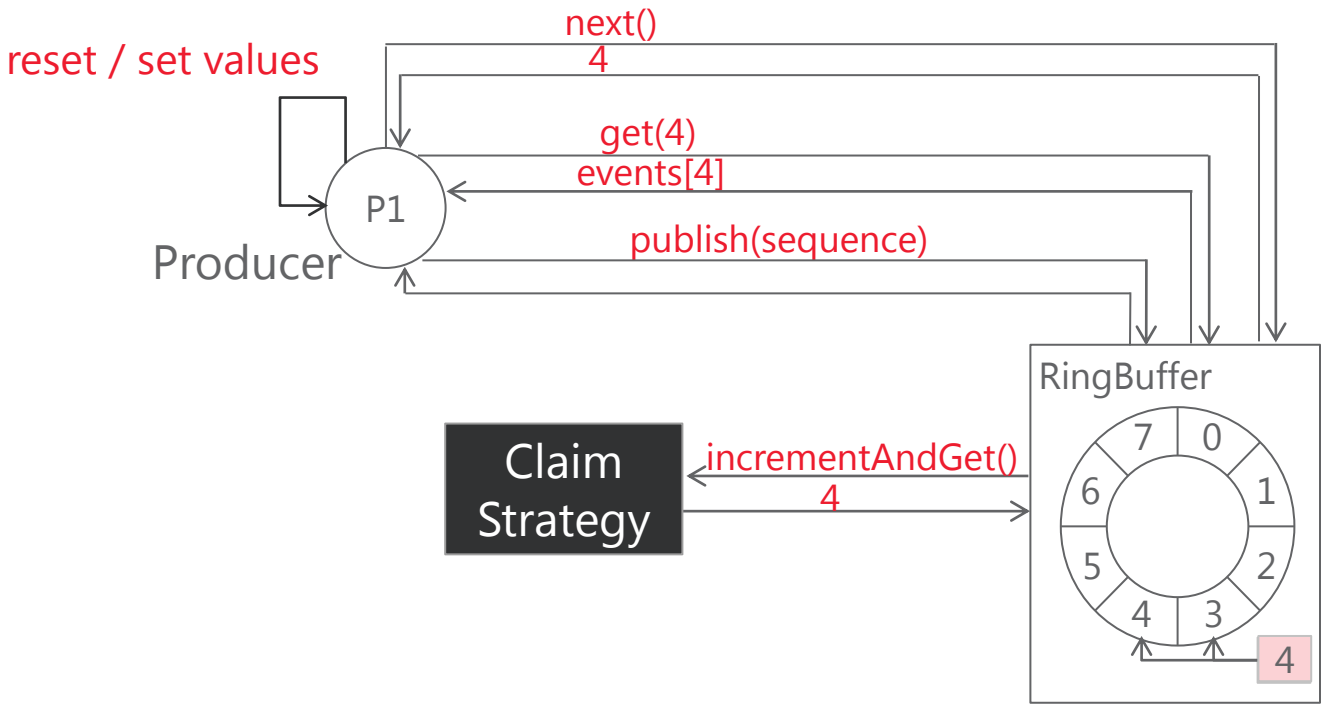


- **Garbage friendly**
- Based on an Array
  - Linear memory access → **prefetching**
  - Multiple entries per cache line
- Reliable messaging
- **Only** the thread writing to the RingBuffer is updating the cursor (Single-Writer-Principle)

Entries created once per slot

Only producers update the cursor

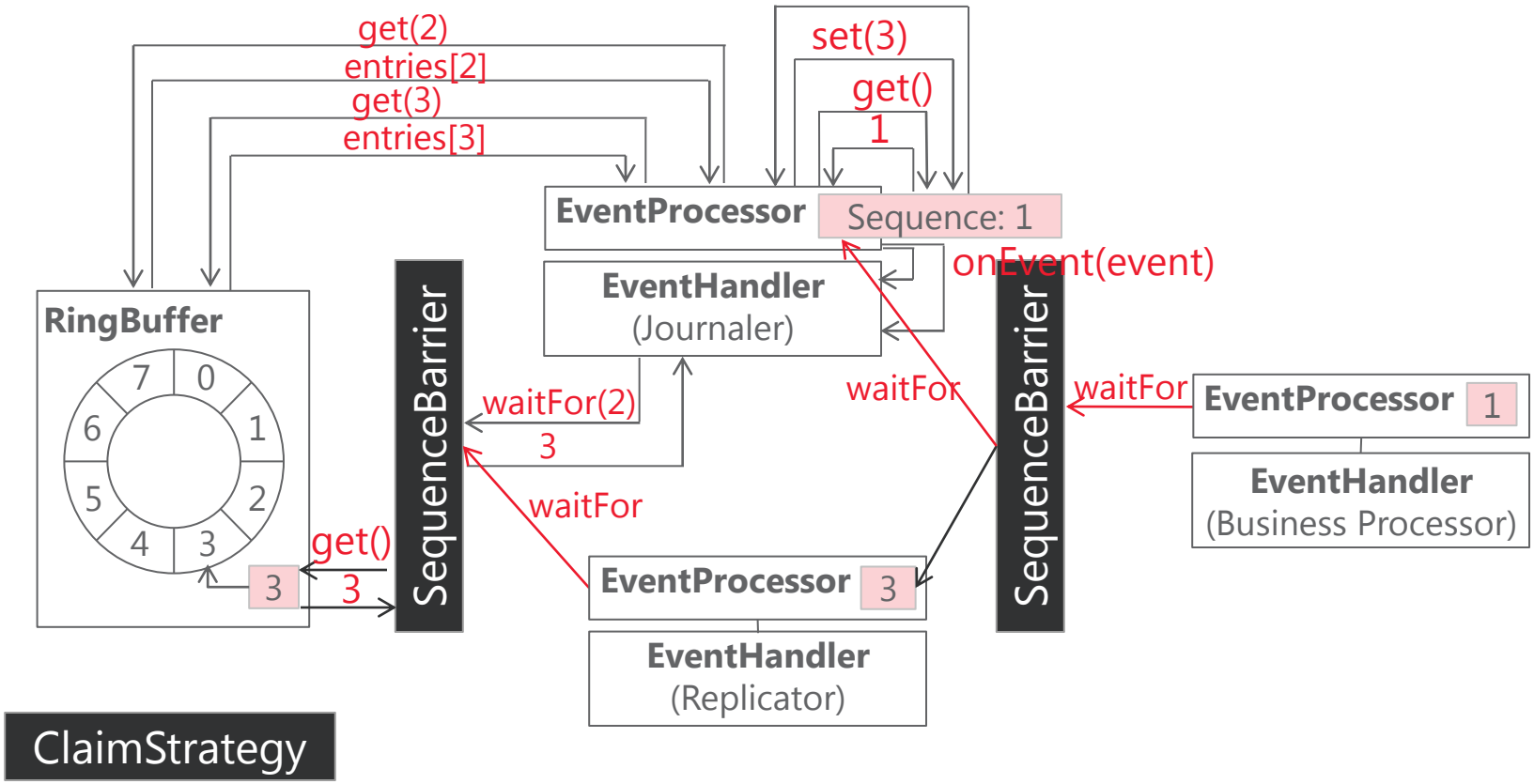
The Disruptor – writing to the RingBuffer



Two-Phase-Commit: claim, publish

ClaimStrategy inhibits wrapping

The Disruptor – Reading from the RingBuffer



Fields on a event should be written by only one EventProcessor!

Setup the Disruptor

```
RingBuffer<TradeEvent> buffer = new RingBuffer<TradeEvent>(
                                        eventFactory,
                                        claimStrategy,
                                        waitStrategy);

SequenceBarrier ringBarrier   = buffer.newBarrier();
```

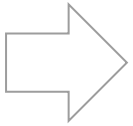
```
BatchEventProcessor<TradeEvent> replication = new BatchEventProcessor<TradeEvent>(
                                                  buffer,
                                                  ringBarrier,
                                                  new ReplicationHandler());
BatchEventProcessor<TradeEvent> journaling = new BatchEventProcessor<TradeEvent>(
                                                  buffer,
                                                  ringBarrier,
                                                  new JournalingHandler());
```

```
SequenceBarrier journalingReplicationBarrier = buffer.newBarrier(
                                                   replProcessor.getSequence(),
                                                   journProcessor.getSequence());
```

Publish messages

```
BatchEventProcessor<TradeEvent> businessLog = new BatchEventProcessor<TradeEvent>(
                                                    buffer,
                                                    journalingReplicationBarrier,
                                                    blHandler);
buffer.setGatingSequences(businessLog.getSequence());
```

```
long sequence = buffer.next();
buffer.get(sequence).setValue(i);
buffer.publish(sequence);
```



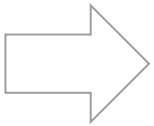
2-phase commit

Setup the Disruptor using the DSL

```
Disruptor<TradeEvent> disruptor = new Disruptor<TradeEvent>(
                                        TradeEvent.EVENT_FACTORY,
                                        EXECUTOR,
                                        claimStrategy,
                                        waitStrategy);
```

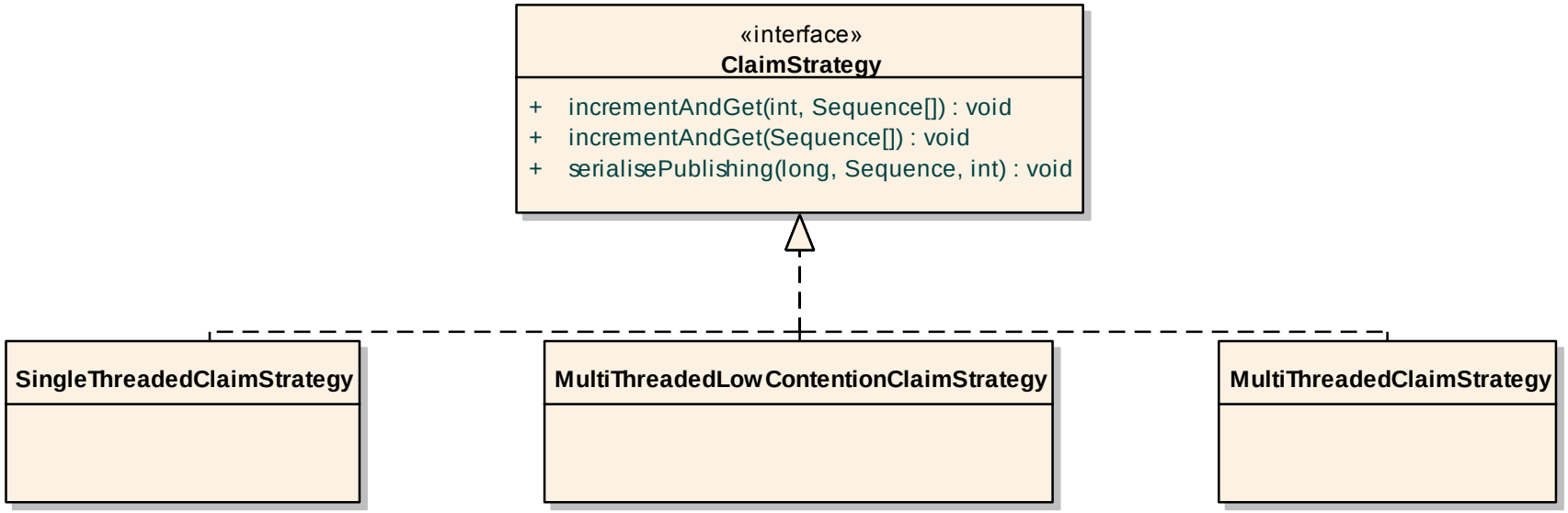
```
disruptor.handleEventsWith( new ReplicationHandler(), new JournalingHandler())
                    .then(  new BusinessLogicHandler());

disruptor.start();
```

```
disruptor.publishEvent(eventTranslator);
```



implicit Two-Phase-Commit

Which ClaimStrategy should I use?

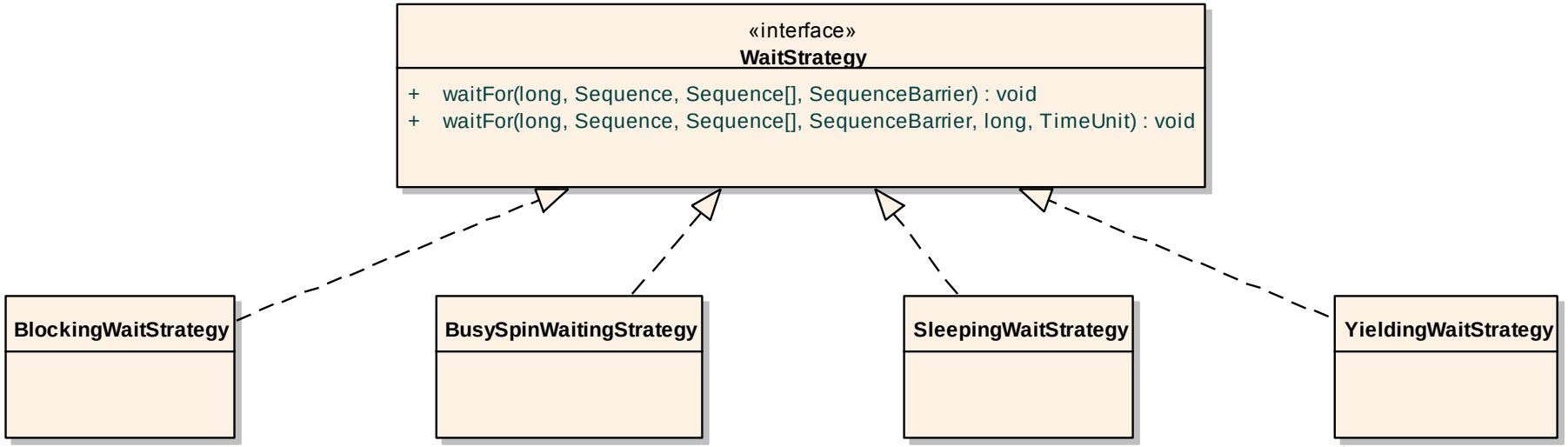


- assumes that **no one else** is publishing to the RingBuffer
- **doesn't use Compare and Swap (AtomicLong) for sequencing**

- assumes that there are **multiple publishers**
- based on an **AtomicLong**

- assumes that there are **multiple publishers**
- forgives when the multiple publisher threads are highly contended
- based on an **AtomicLong**

Decisions to be made - WaitStrategy



Puts current thread to state waiting until requested entry gets available

- ➕ CPU resource
- ➖ Throughput

Spins until requested entry gets available

- ➖ CPU resource
- ➕ Throughput

spin (100 times)→ yield (100 times) → timed waiting

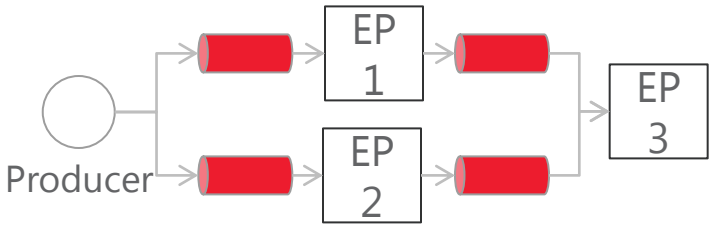
- 〓 CPU resource
- 〓 Throughput

yields current thread

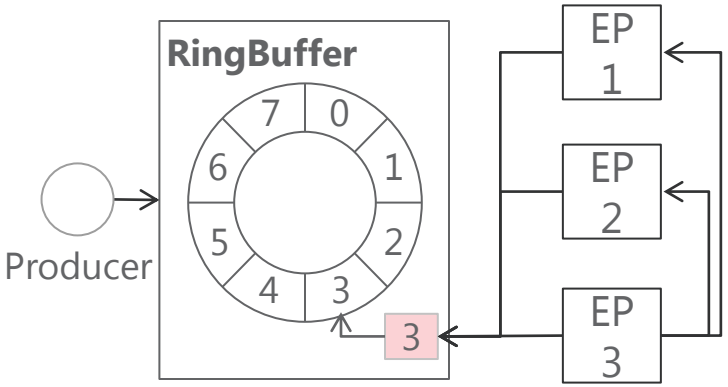
- 〓 CPU resource
- 〓 Throughput

Performance - LinkedBlockingQueue vs Disruptor

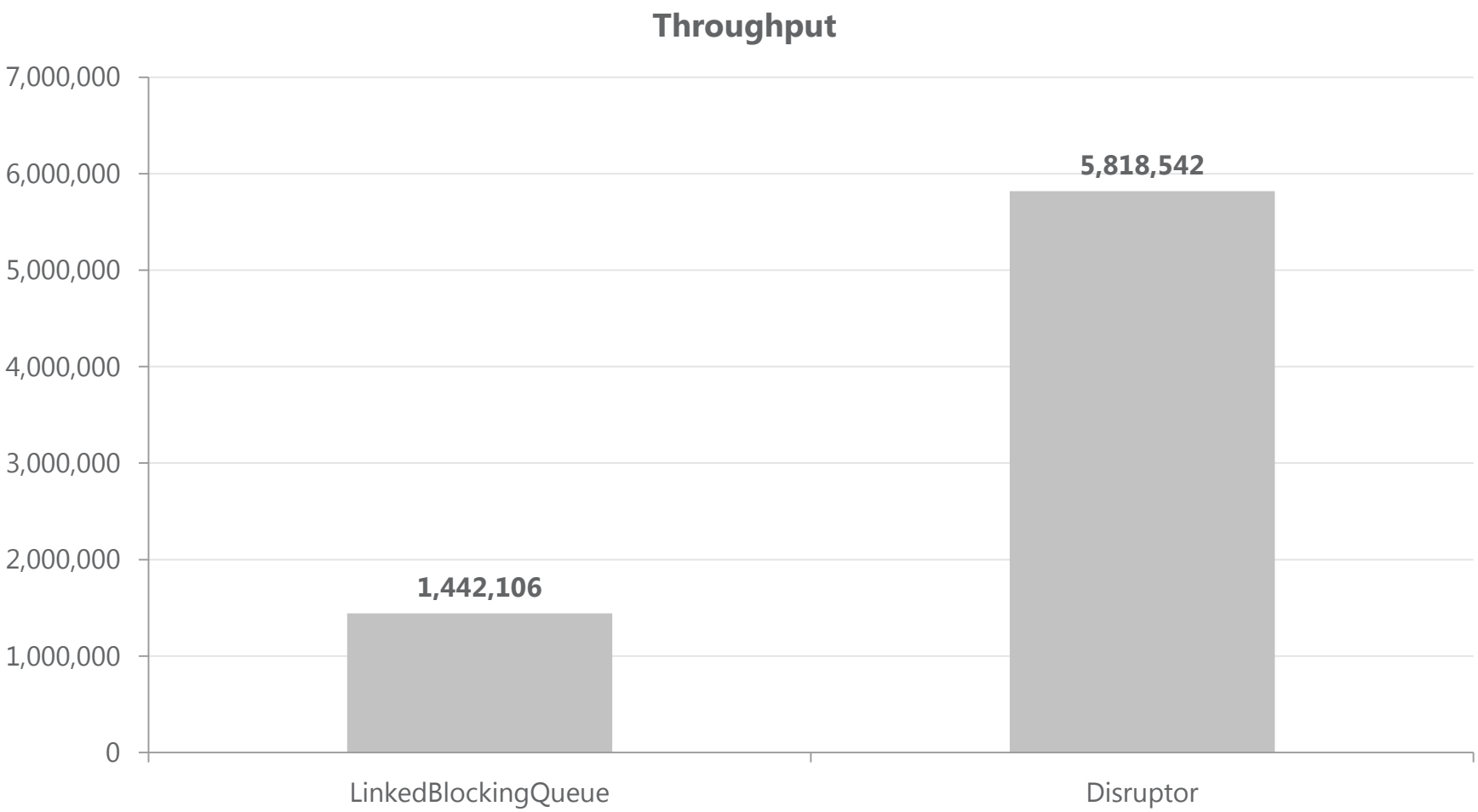
LinkedBlockingQueue



Disruptor



Performance - Throughput



Why is it so fast

1. No Locks
2. minimized use of memory barriers
3. One single place of contention: if multiple producers write to the RingBuffer
4. Multiple producers, consumers share the same data structure
5. Recycling of events→ less garbage → less garbage collection pauses
6. tracking sequences at each place + cache line padding → no false sharing → no unexpected contetion

Sources

- Blogs
  - Martin Thompson
    http://mechanical-sympathy.blogspot.com
  - Michael Barker
    http://bad-concurrency.blogspot.com
  - Trisha Gee
    http://mechanitis.blogspot.com
- Articles
  - Technical Paper
    http://disruptor.googlecode.com/files/Disruptor-1.0.pdf
  - Java Best Practices – Queue battle and the Linked ConcurrentHashMap
    http://www.javacodegeeks.com/2010/09/java-best-practices-queue-battle-and.html
- Presentations
  - Understanding the Disruptor: A Beginners Guide to Hardcore Concurrency
    http://www.youtube.com/watch?v=DCdGIxBbKU4

Questions

THANK YOU.

Trivadis AG

Raffael Schmid

Europa-Strasse 5
8152 Glattbrugg

Mobile +41 79 699 70 09
Tel. +41 44 808 70 20
Fax +41 44 808 70 21

raffael.schmid@trivadis.com
www.trivadis.com

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

trivadis
makes IT easier.