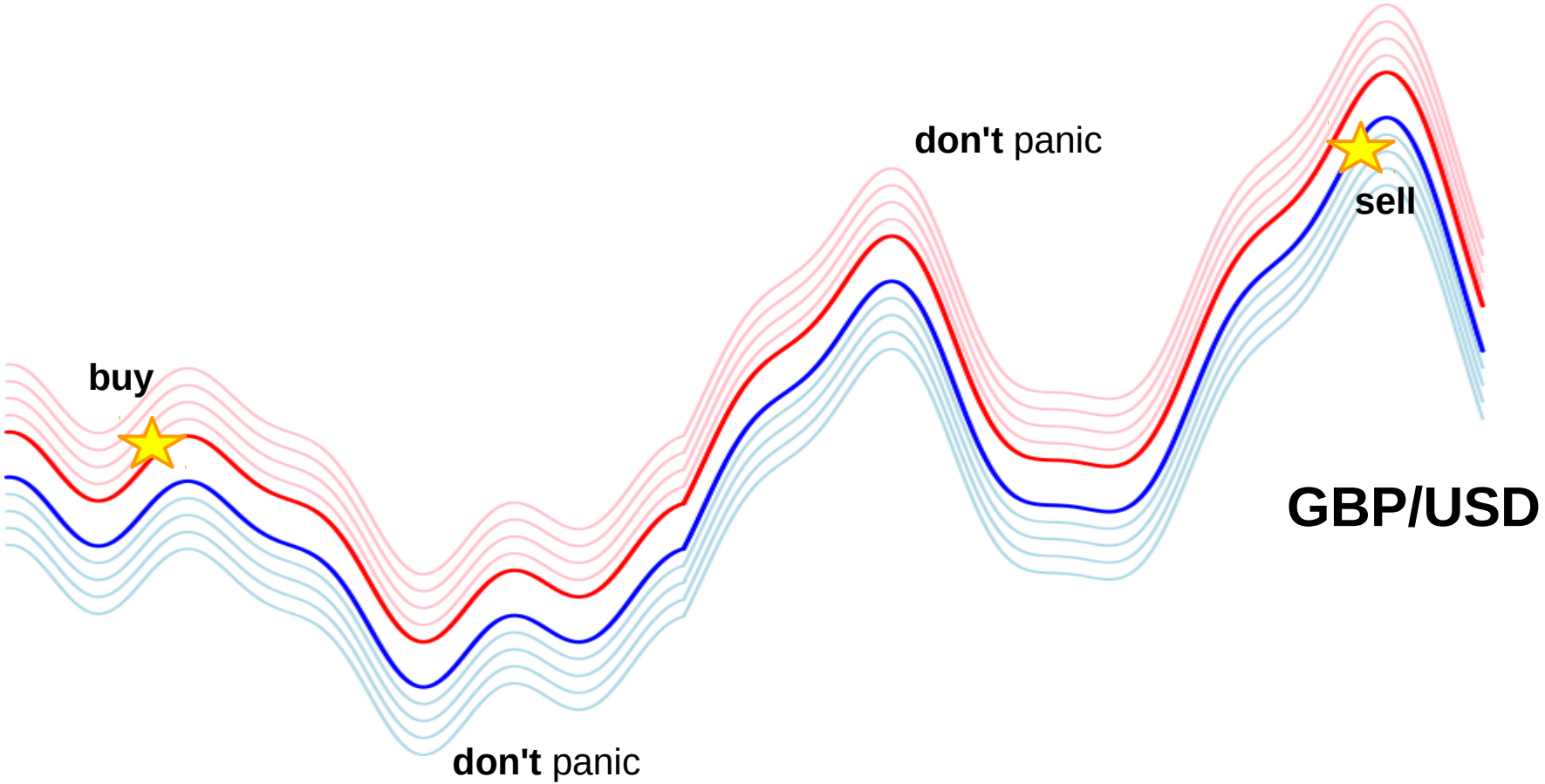
speed › price › transparency

LMAX™ EXCHANGE

Low Latency Trading Architecture

Sam Adams

QCon London, March, 2017



LMAX EXCHANGE

Typical day:

1,000's active clients

100,000's trades occur

100,000,000's orders placed

– very bursty: spikes of 100s / ms

1,000,000,000's market data updates sent

End-to-end latency:

50%: 80 µs

99%: 150 µs

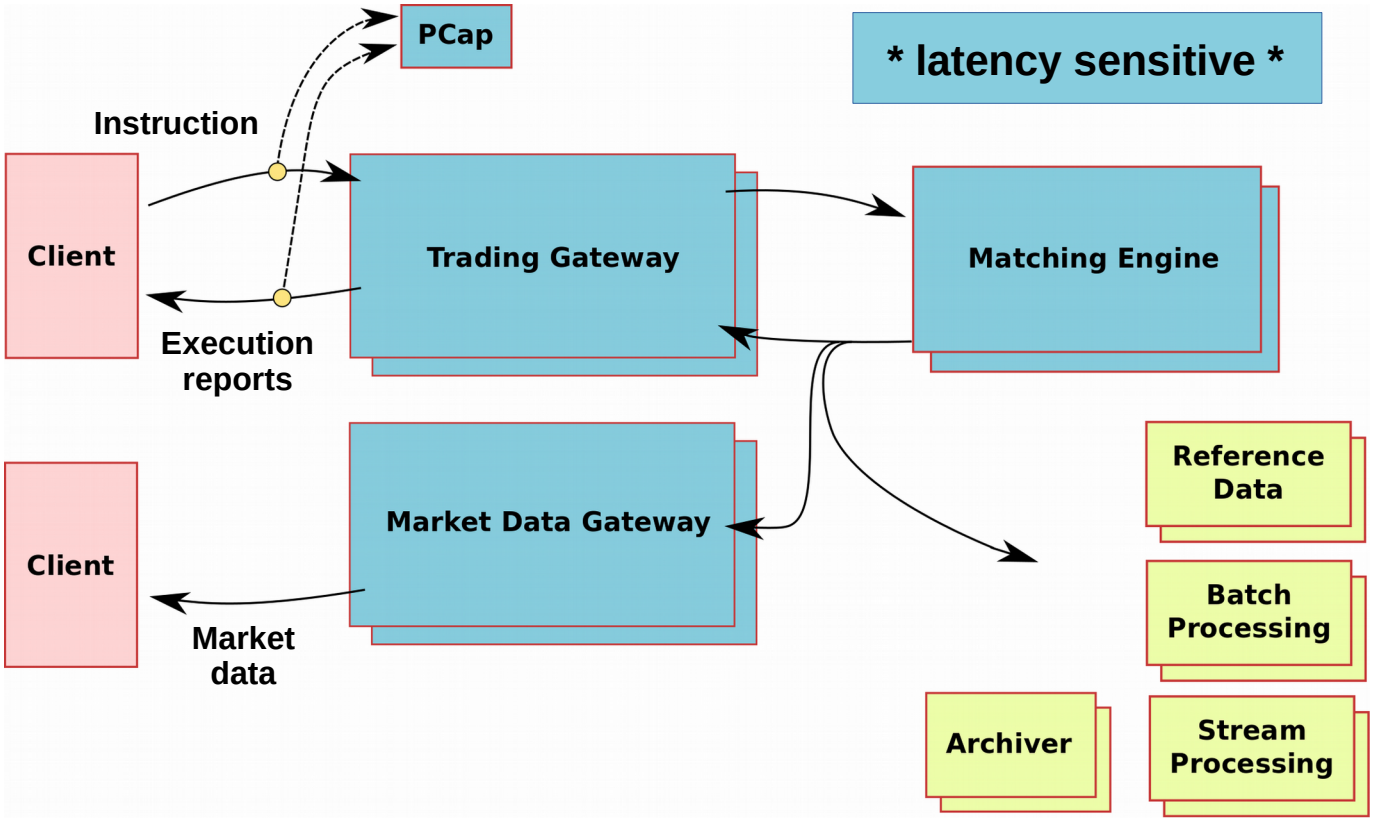
99.99%: 500 µs

Max: 4ms(*)

System Architecture

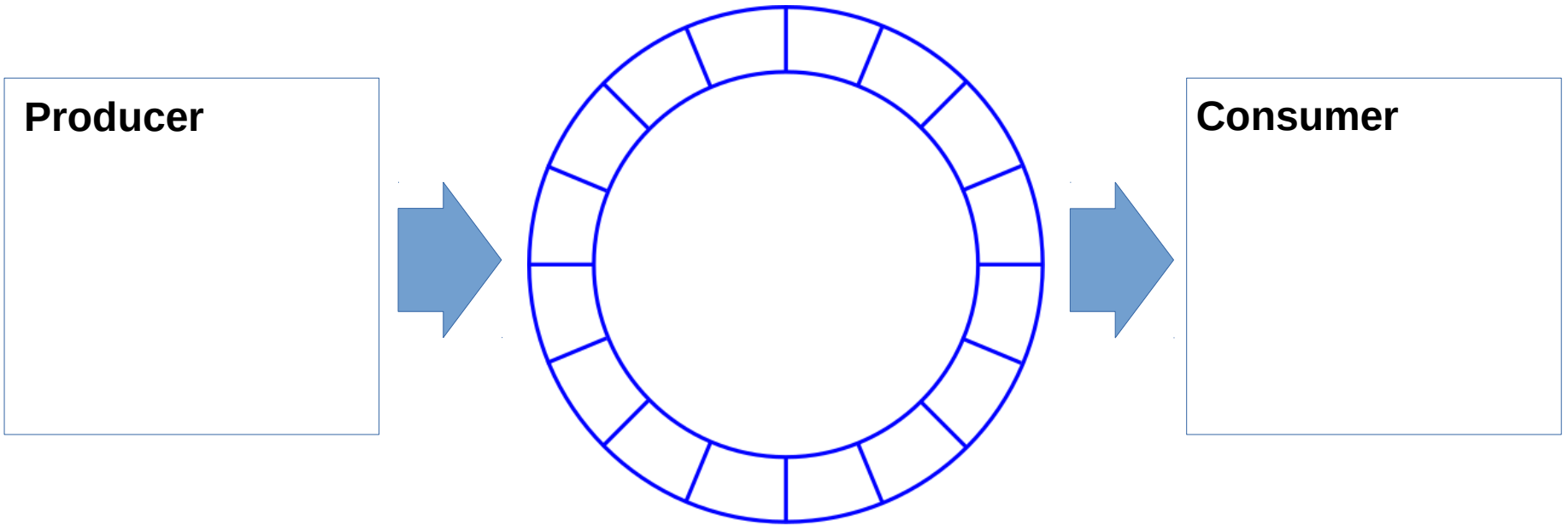
Building low latency applications

LMAX EXCHANGE


PCap
* latency sensitive *
Instruction
Client
Trading Gateway
Matching Engine
Execution reports
Client
Market Data Gateway
Reference Data
Batch Processing
Market data
Archiver
Stream Processing
* throughput matters *

The Disruptor

High performance inter-thread messaging



ArrayBlockingQueue vs Disruptor

```
public class ArrayBlockingQueue<E>
{
  final Object[] items;
  int takeIndex;
  int putIndex;
  int count;

  /** Main lock guarding all access */
  final ReentrantLock lock;
}
```

locking & contention

ArrayBlockingQueue vs Disruptor

```
public class ArrayBlockingQueue<E>
{
  final Object[] items;
  int takeIndex;
  int putIndex;
  int count;

  /** Main lock guarding all access */
  final ReentrantLock lock;
}
```

locking & contention
vs
single writers

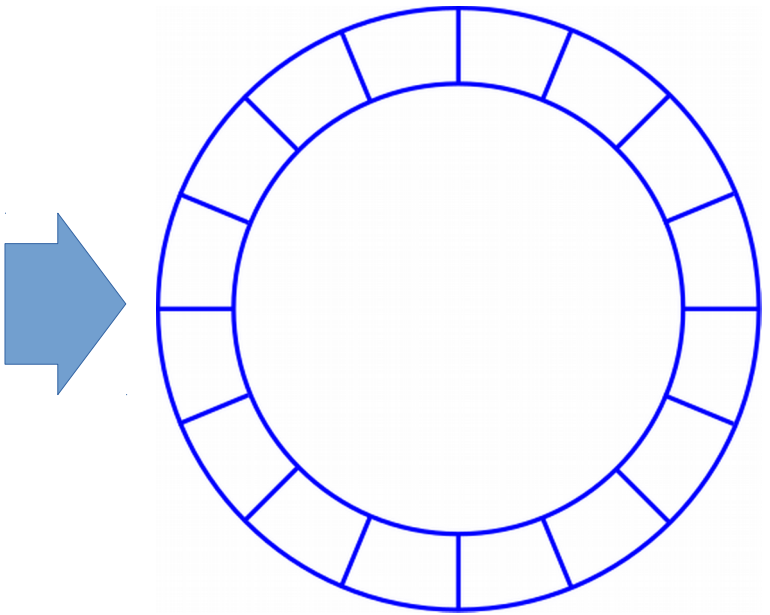
```
public class RingBuffer<E>
  implements DataProvider<E>
{
  // ...
  final long indexMask;
  final Object[] entries;
  final Sequence cursor;
  // ...
}
```

```
public class BatchEventProcessor<E>
{
  final DataProvider<E> dataProvider;
  final Sequence sequence;
}
```

LMAX EXCHANGE

Claimed: -1
Published: -1

Producer



Consumer

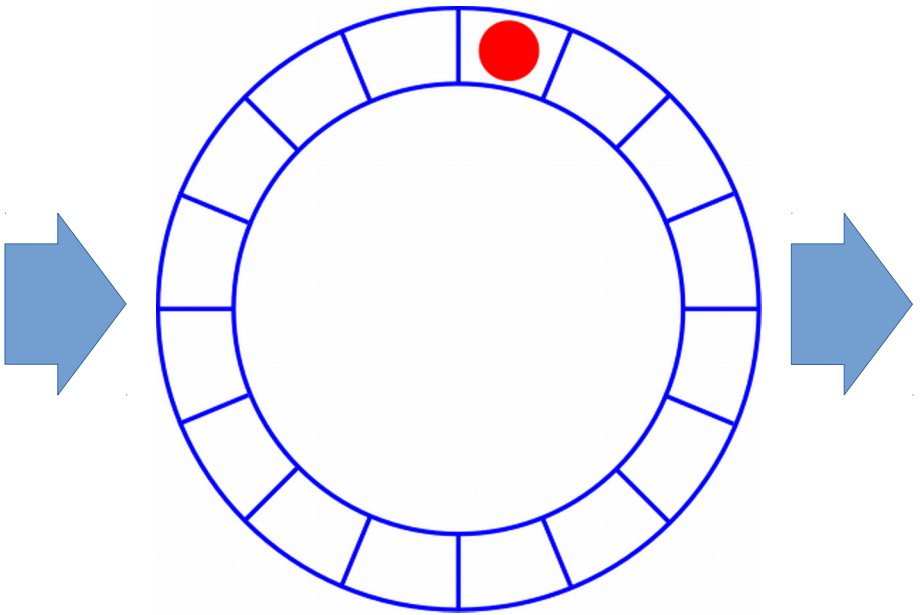
Consumed: -1

Waiting for: 0

Claimed: 0
Published: -1

Producer

Claim slot: 0



Consumer

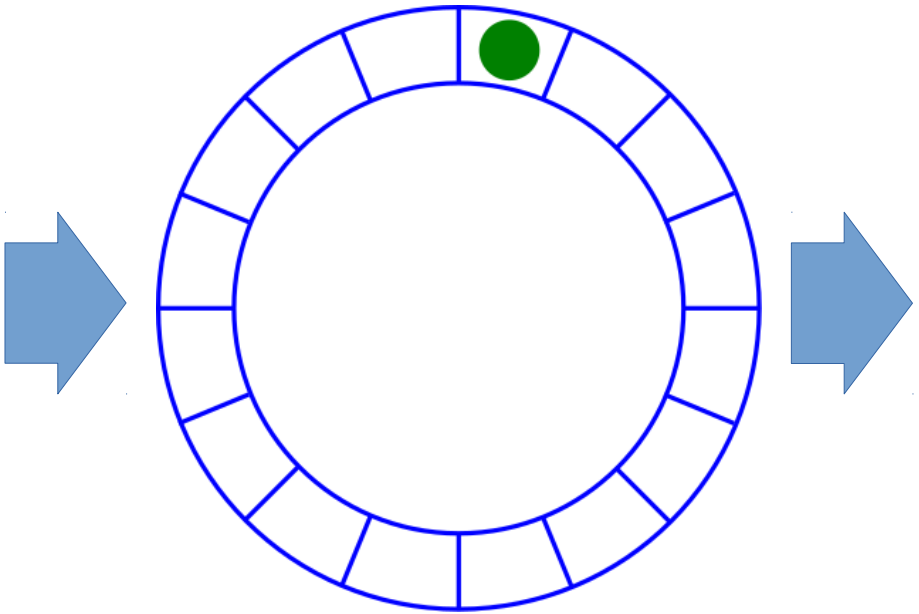
Consumed: -1

Waiting for: 0

Claimed: 0
Published: 0

Producer

Publish slot: 0



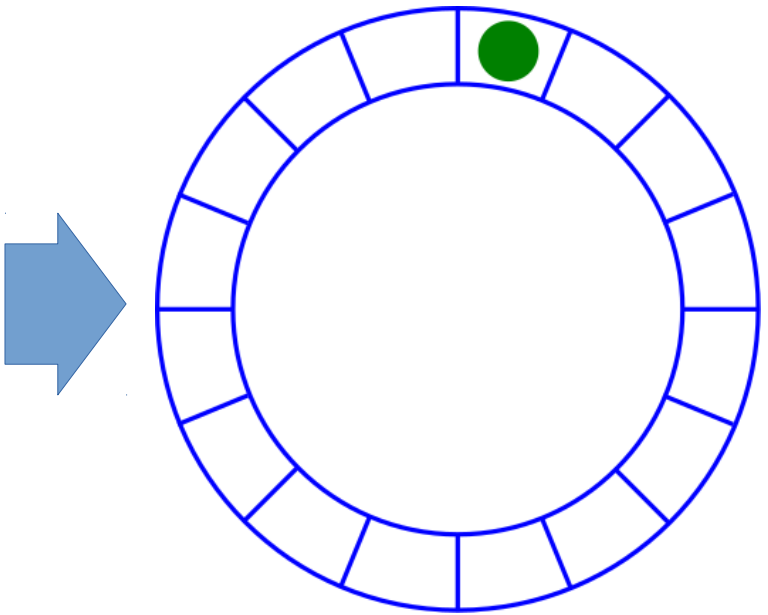
Consumer

Consumed: -1

Waiting for: 0

Claimed: 0
Published: 0

Producer



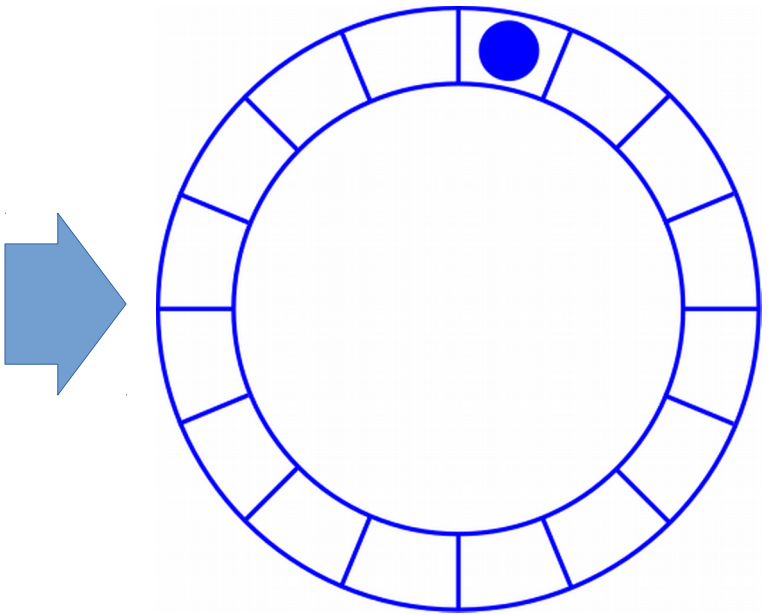
Consumer

Consumed: -1

Available: 0
Processing: 0

Claimed: 0
Published: 0

Producer



Consumer

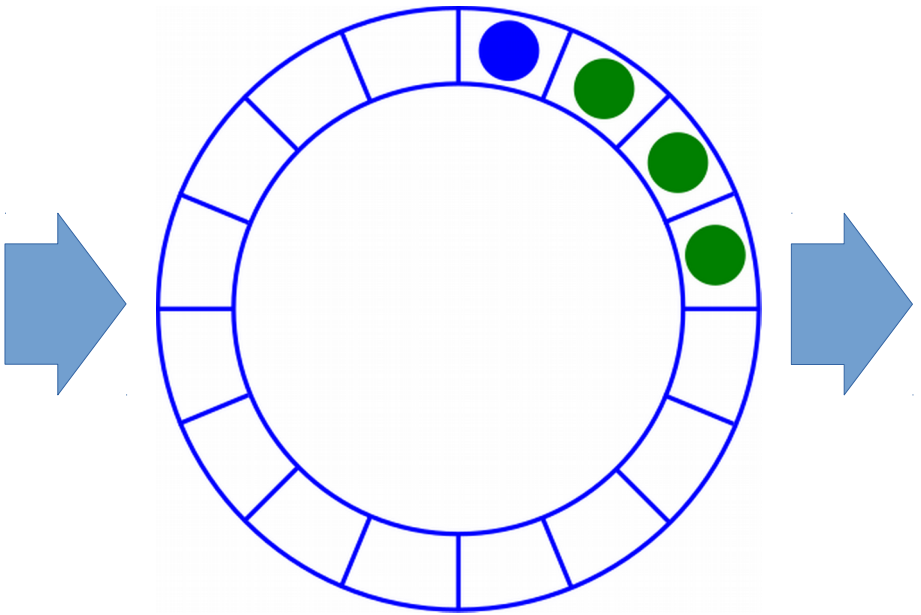
Consumed: 0

Waiting for: 1

Claimed: 3
Published: 3

Producer

Published: 1-3



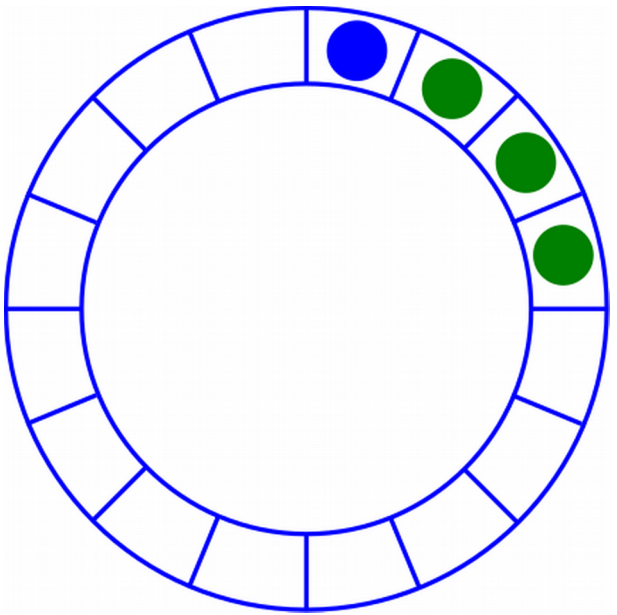
Consumer

Consumed: 0

Waiting for: 1

Claimed: 3
Published: 3

Producer



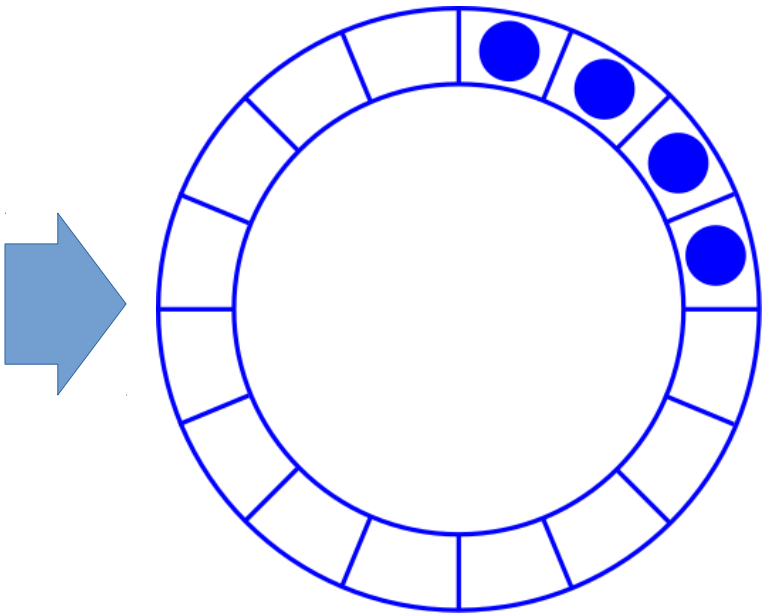
Consumer

Consumed: 0

Available: 3
Processing: 1,2,3

Claimed: 3
Published: 3

Producer

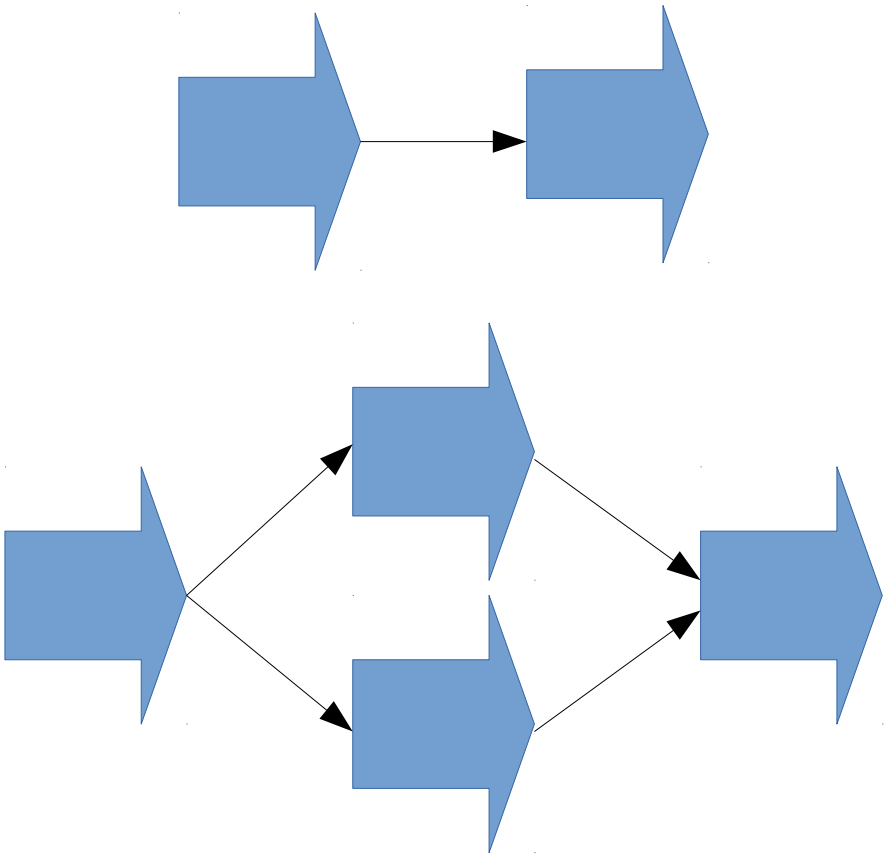


Consumer

Consumed: 3

Waiting for: 4

Supports dependency graphs between consumers

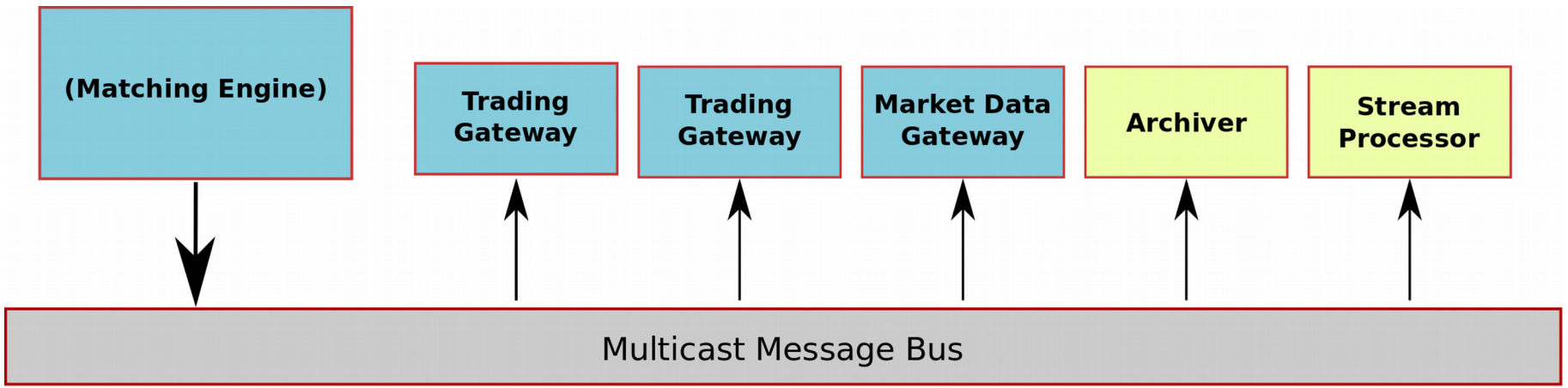


LMAX EXCHANGE

Messaging

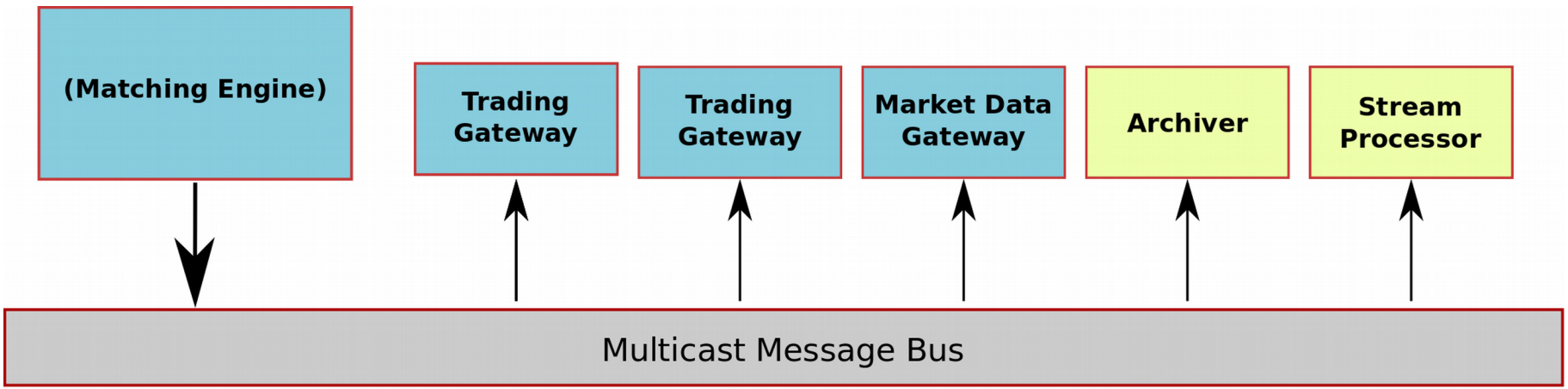
Asynchronous Pub/Sub messaging:

- UDP Multicast: low latency, scalable, **unreliable**
- Services publish / subscribe to topics
    - topic = unique multicast group
- Informatica UMS (aka 29 West LBM) provides * some reliability *



Asynchronous Pub/Sub messaging:

- Push based
- If you miss a message, it is gone
- Late-join: no history



javassist generated proxies to interfaces

Event:

```
long sequence
byte operationIndex
byte[] data
int length
```

```
public interface TradingInstructions
{
    void placeOrder(PlaceOrderInstruction instruction);

    void cancelOrder(CancelOrderInstruction instruction);
}
```

See **`GeneratedRingBufferProxyGenerator`** in disruptor-proxy for inter-thread version
https://github.com/LMAX-Exchange/disruptor-proxy

LMAX EXCHANGE

Event:

```
long sequence
byte operationIndex
byte[] data
int length
```

Publisher proxy:

```
public void placeOrder(PlaceOrderInstruction arg0)
{
  // ...
  event.initialise(sequence, 1);    // operation index
  marshaller.encode(arg0, event.outputStream());
  // ...
}
```

See **`GeneratedRingBufferProxyGenerator`** in disruptor-proxy for inter-thread version
https://github.com/LMAX-Exchange/disruptor-proxy

Subscriber proxy:

Event:

```
long sequence
byte operationIndex
byte[] data
int length
```

```
Invoker invokers[];
TradingInstructions implementation;

public void onEvent(Event event)
{
  Invoker invoker = invokers[event.getOperationIndex()];
  invoker.invoke(event.getInputStream(), implementation);
}
```

```
public void invoke(InputStream input, TradingInstructions implementation)
{
  PlaceOrderInstruction arg0 = marshaller.decode(input);
  implementation.placeOrder(arg0);
}
```

See **GeneratedRingBufferProxyGenerator** in disruptor-proxy for inter-thread version
https://github.com/LMAX-Exchange/disruptor-proxy

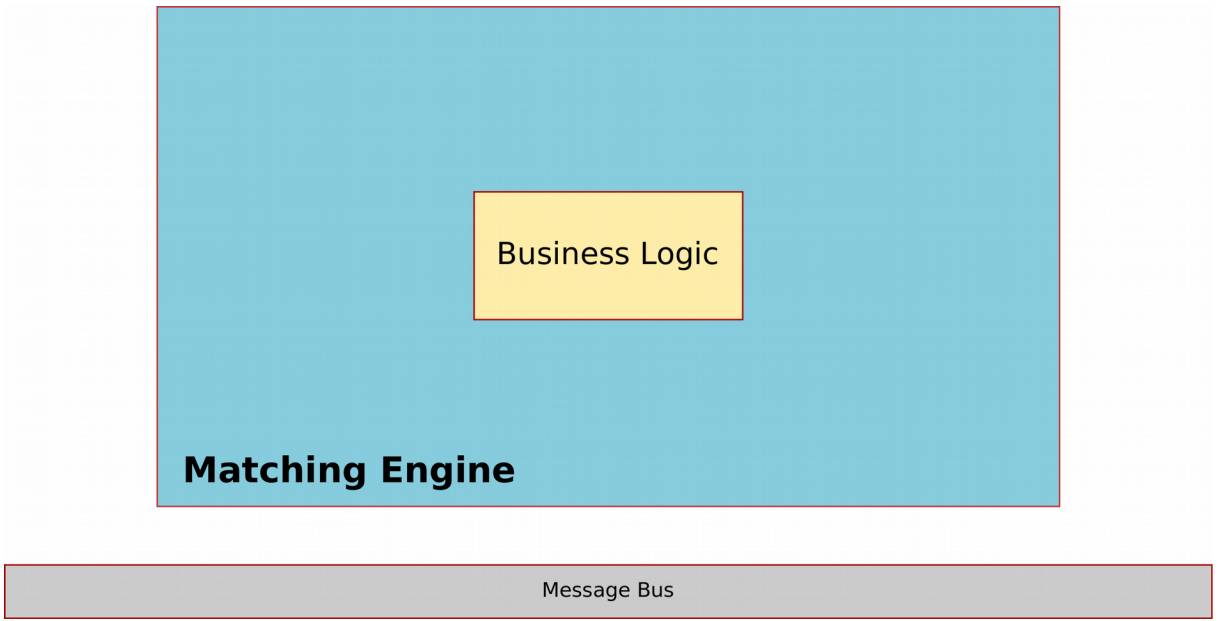
LMAX EXCHANGE

Matching Engine

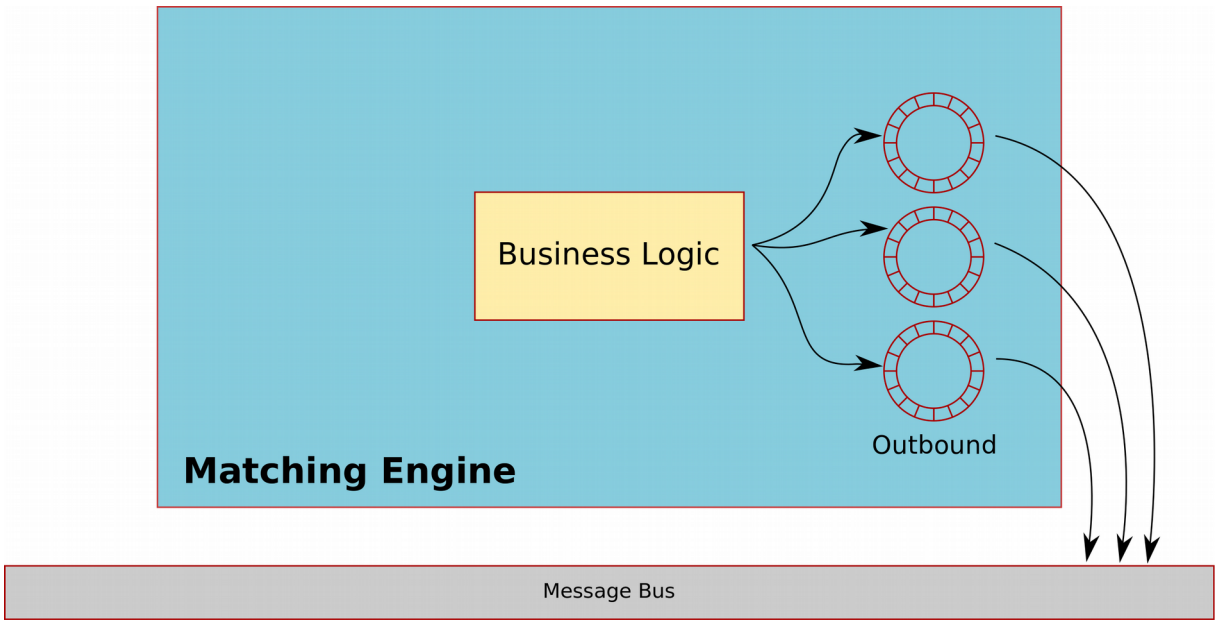
For speed:

All working state held in memory

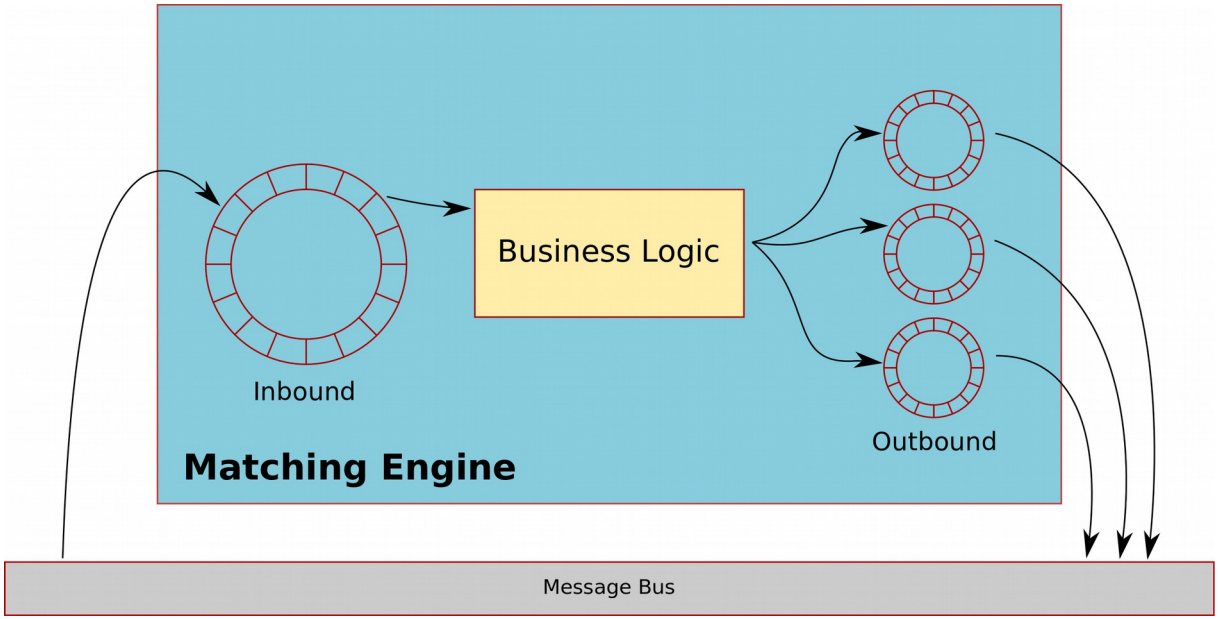
Remove contention: single threaded



Don’t block business logic: buffer for outbound I/O

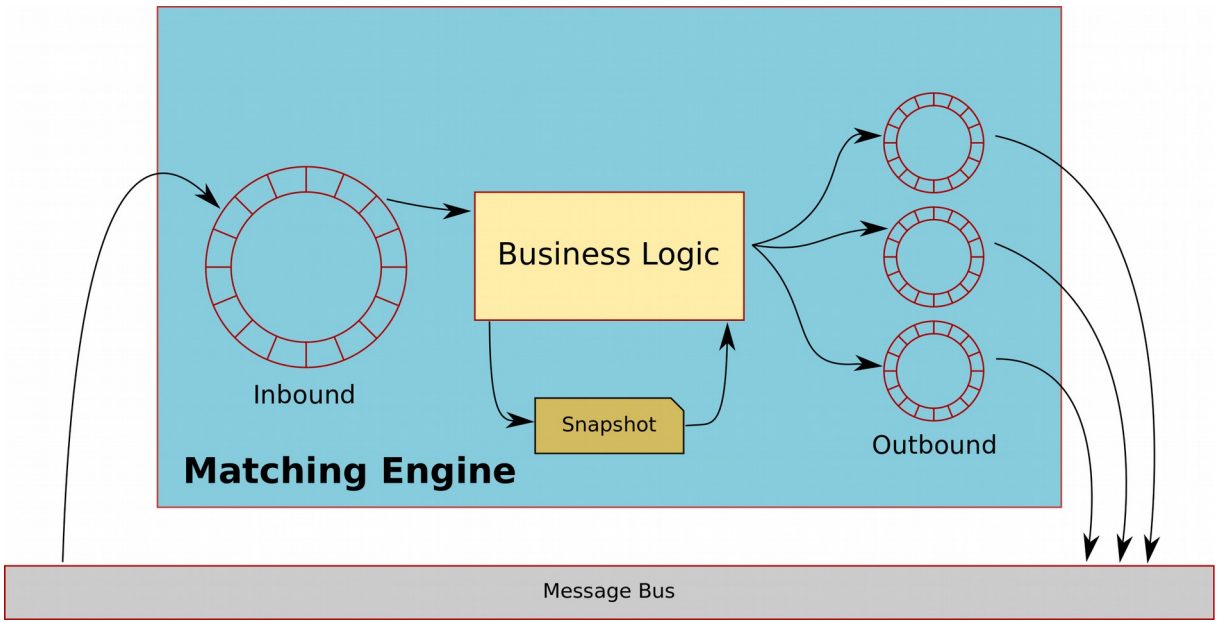


Don’t block network thread: buffer incoming events

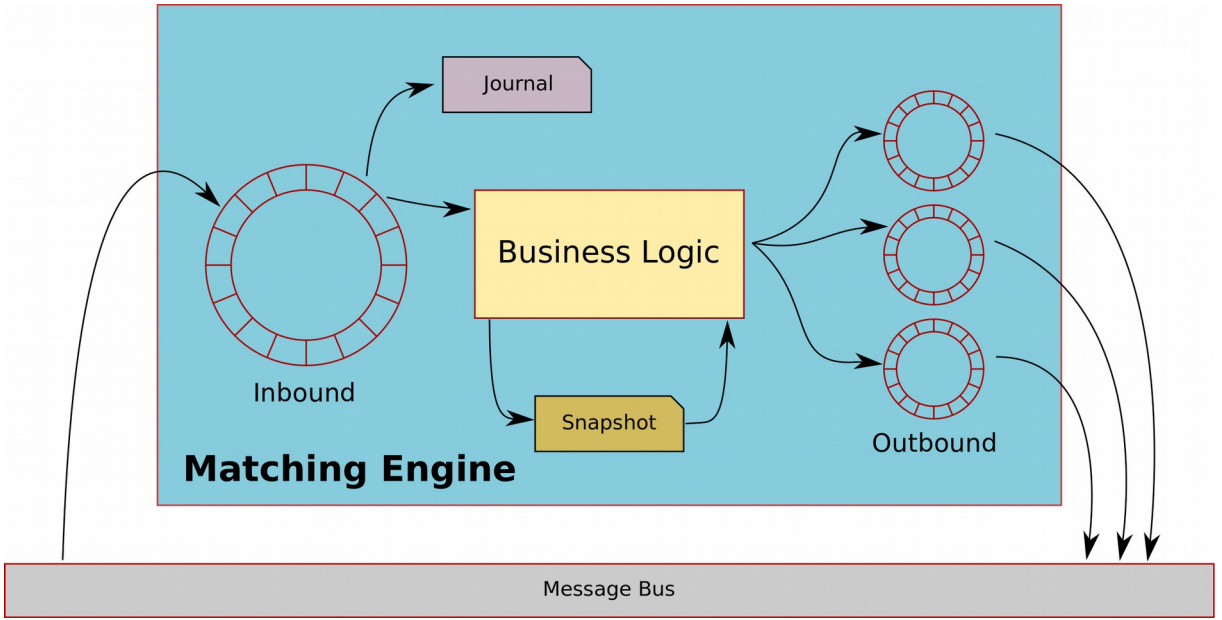


LMAX EXCHANGE

All state in volatile memory:
Save on shutdown / Load on startup



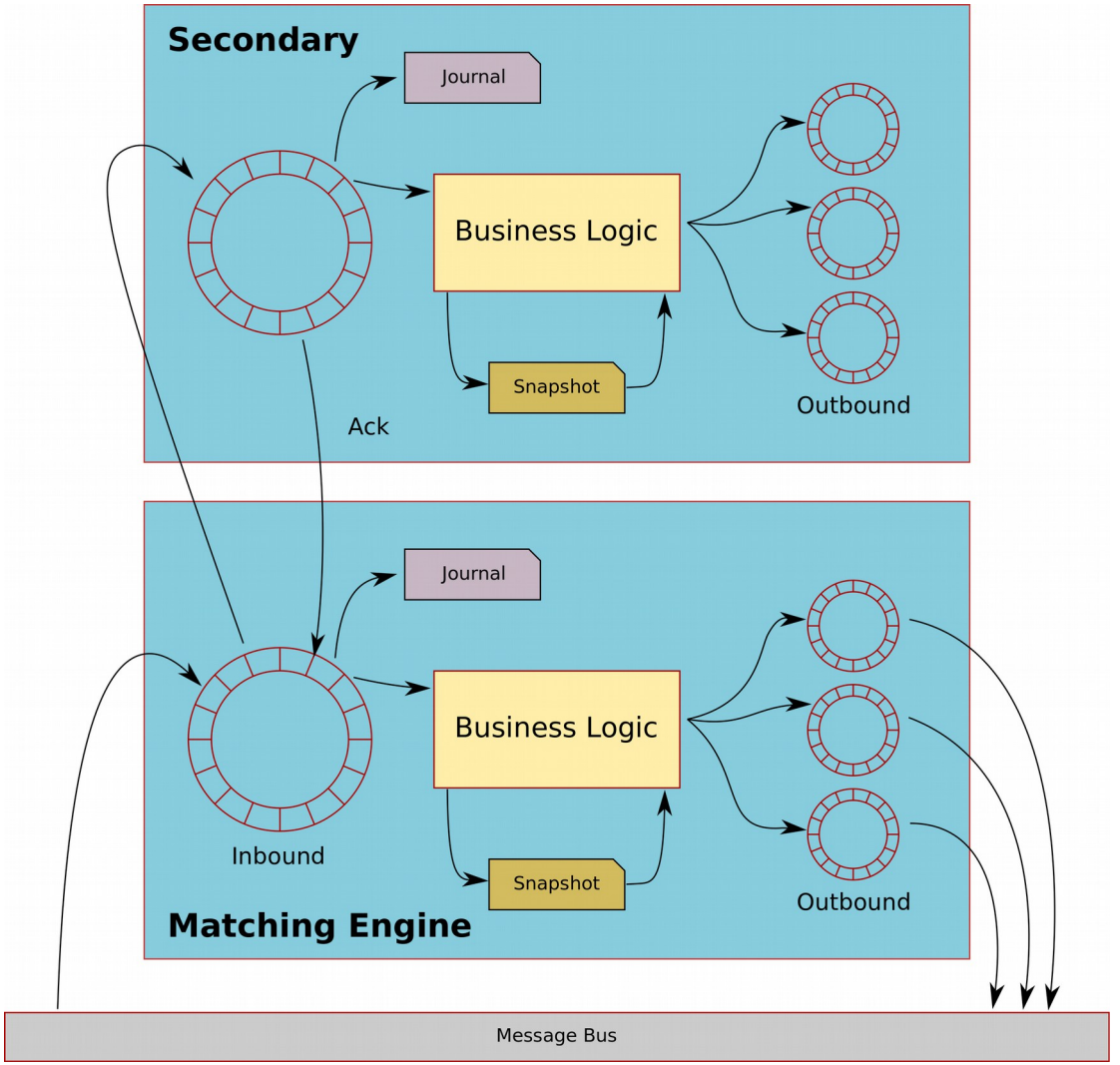
Recover from unclean shutdown
Journal incoming events to disk, replay on startup



Replicate events to hot-standby for resiliency

Manual fail-over

(also to offsite DR)



LMAX EXCHANGE

Holding all your state in memory

No database

No roll-back

Up-front validation is critical

Never throw exceptions
- result is inconsistent state

System must be deterministic

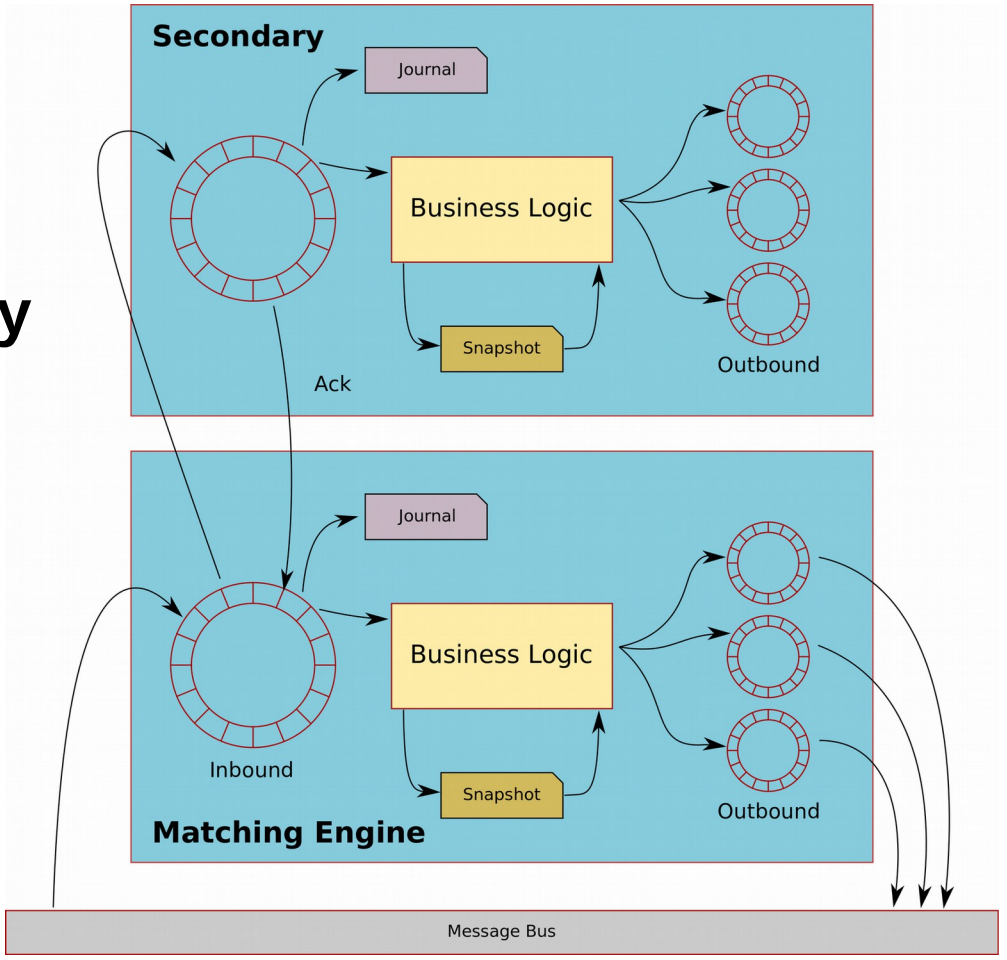
All operations event sourced

time sourced from events
collections must be ordered
no local configuration

Determinism bugs are really nasty

Only an issue if we have
to fail-over or replay

Primary is the source of truth

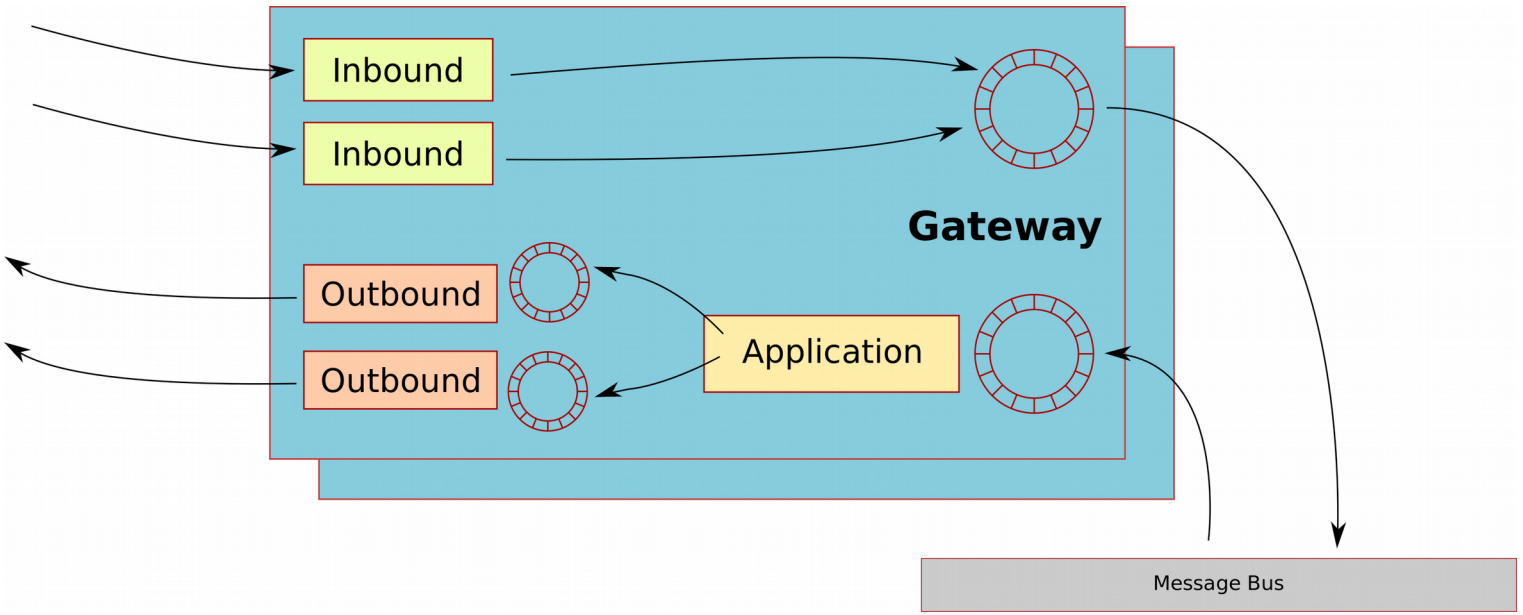


LMAX EXCHANGE

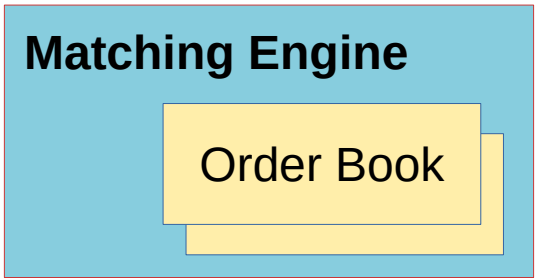
Gateways

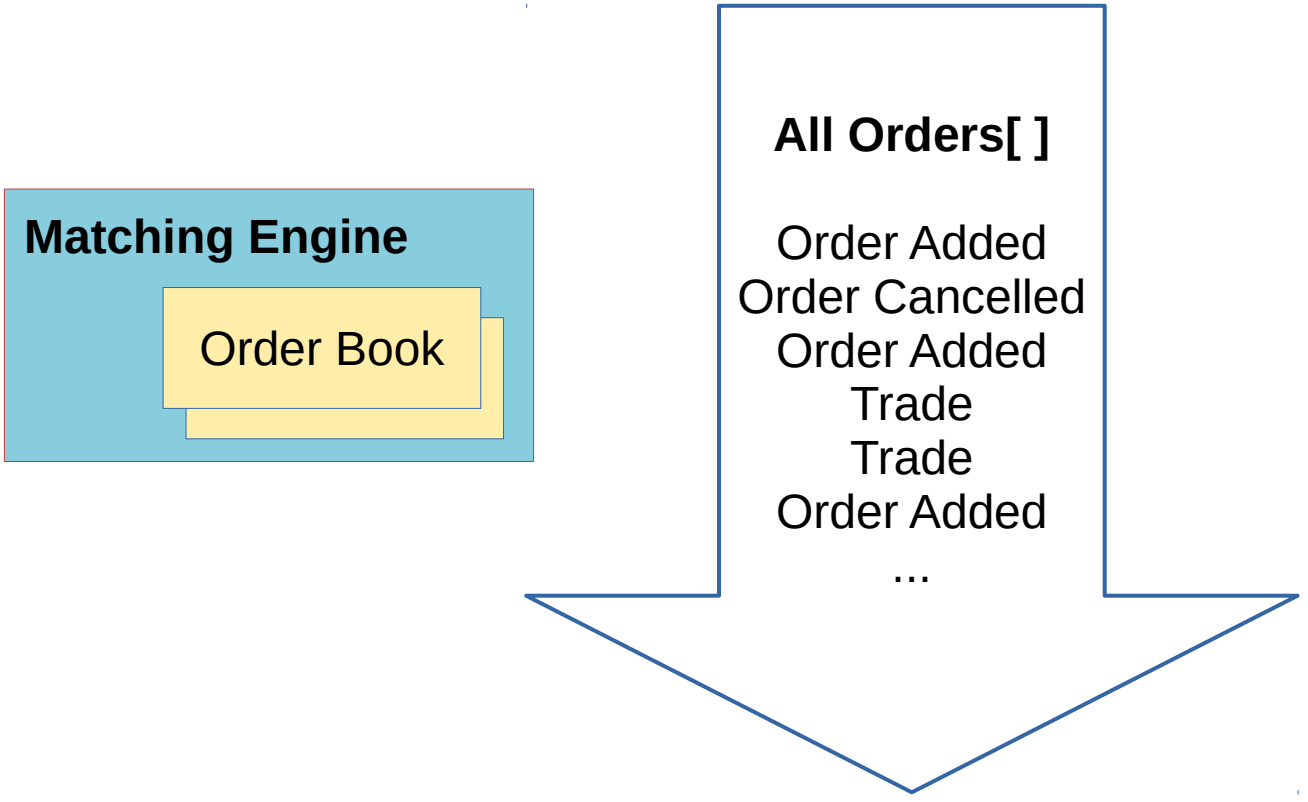
Same principles:

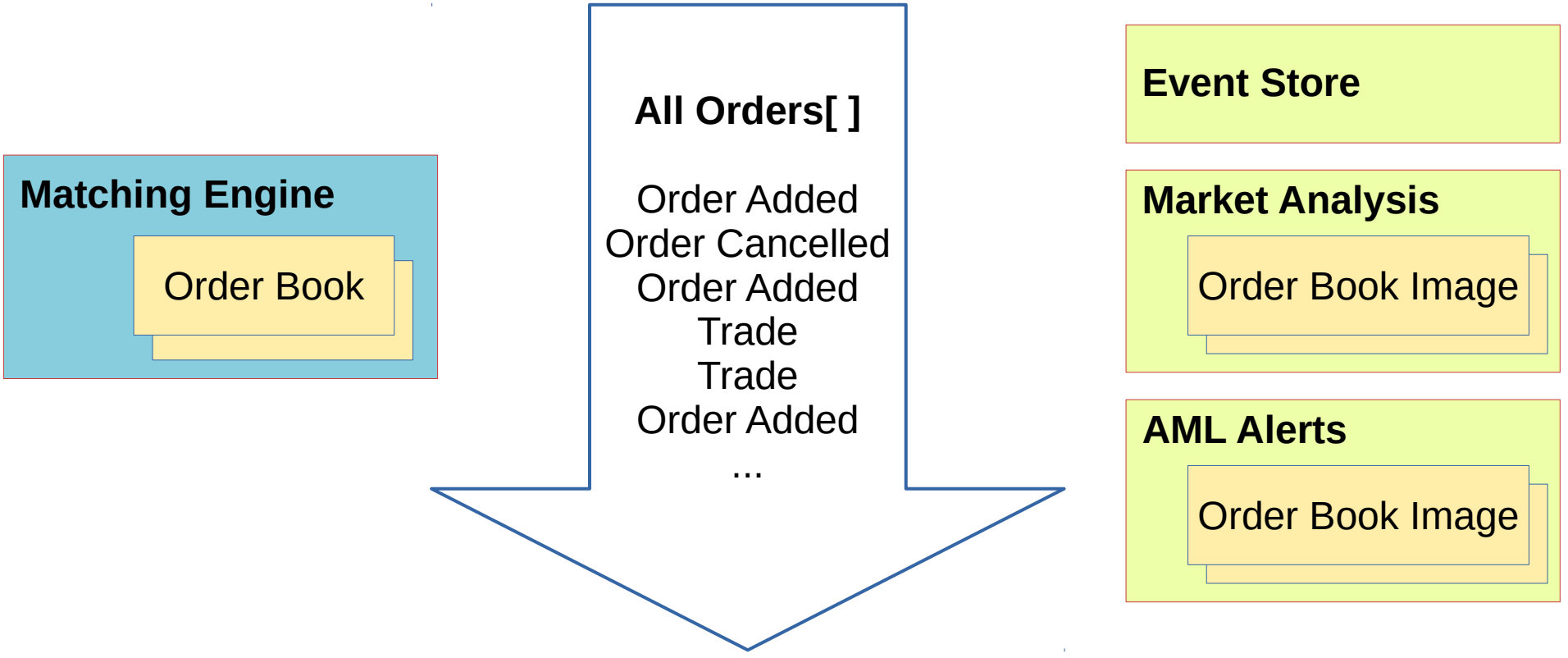
- non-blocking / message passing
- minimise shared state



Stream Processing


Matching Engine
Order Book


Matching Engine
Order Book
All Orders[ ]
Order Added
Order Cancelled
Order Added
Trade
Trade
Order Added
...

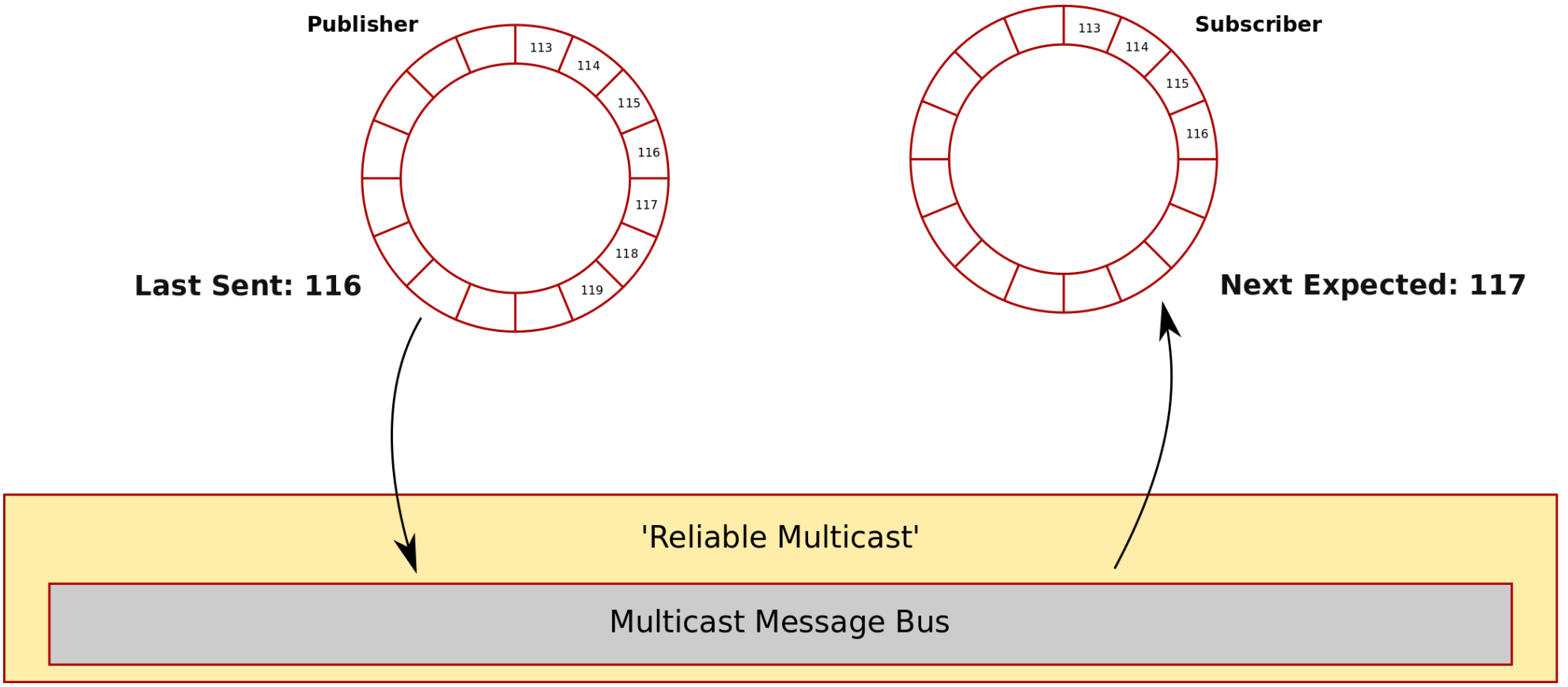

Matching Engine
Order Book
All Orders[ ]
Order Added
Order Cancelled
Order Added
Trade
Trade
Order Added
...
Event Store
Market Analysis
Order Book Image
AML Alerts
Order Book Image

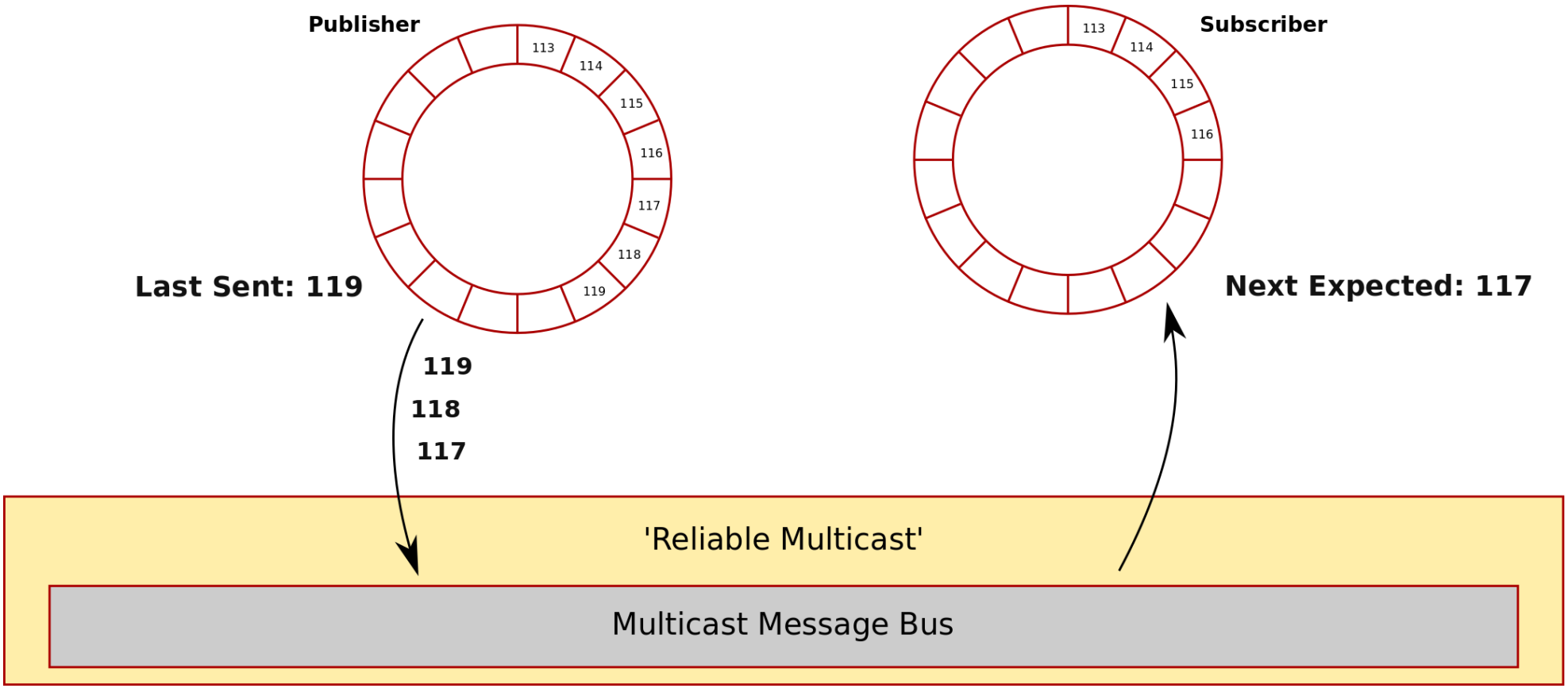
Where latency doesn’t matter...

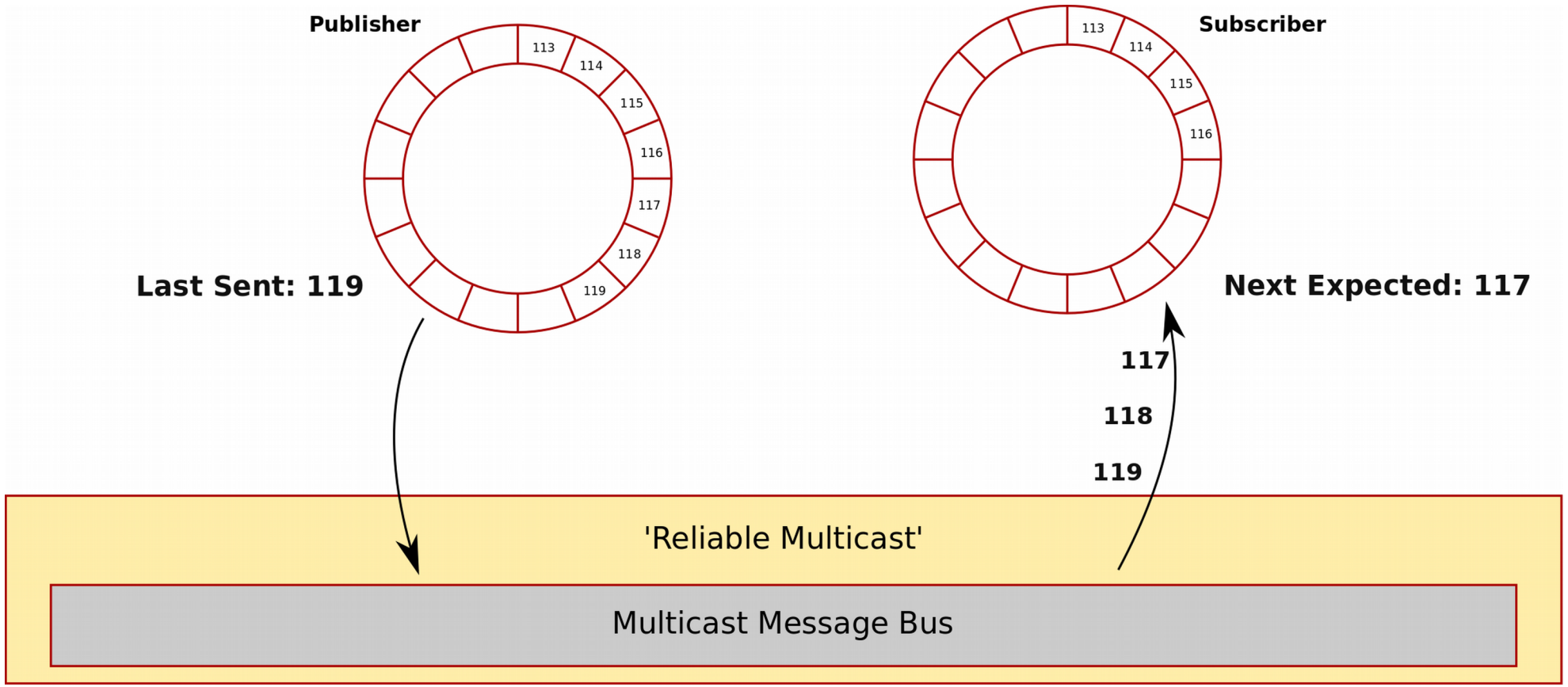
- How big are the bursts?
- Buffers are your friend

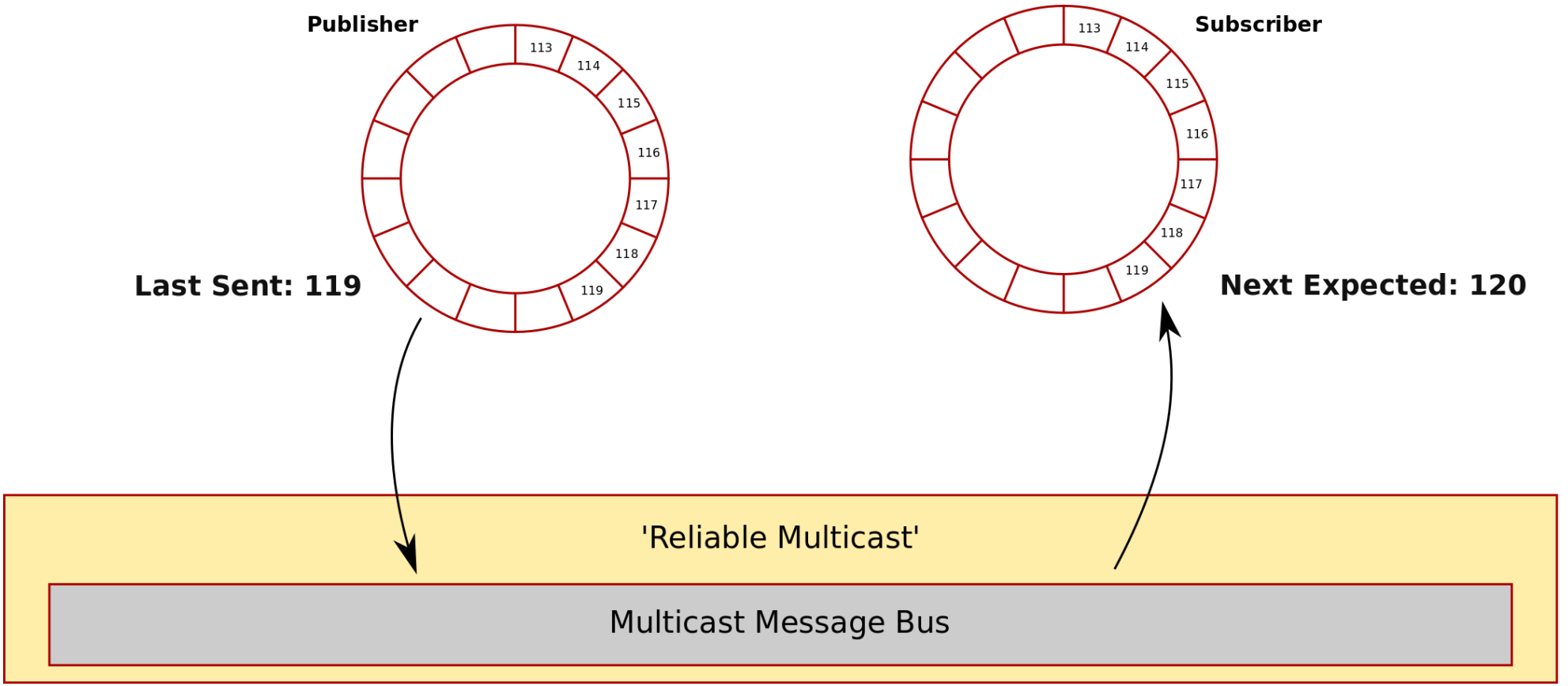
Does data loss matter?

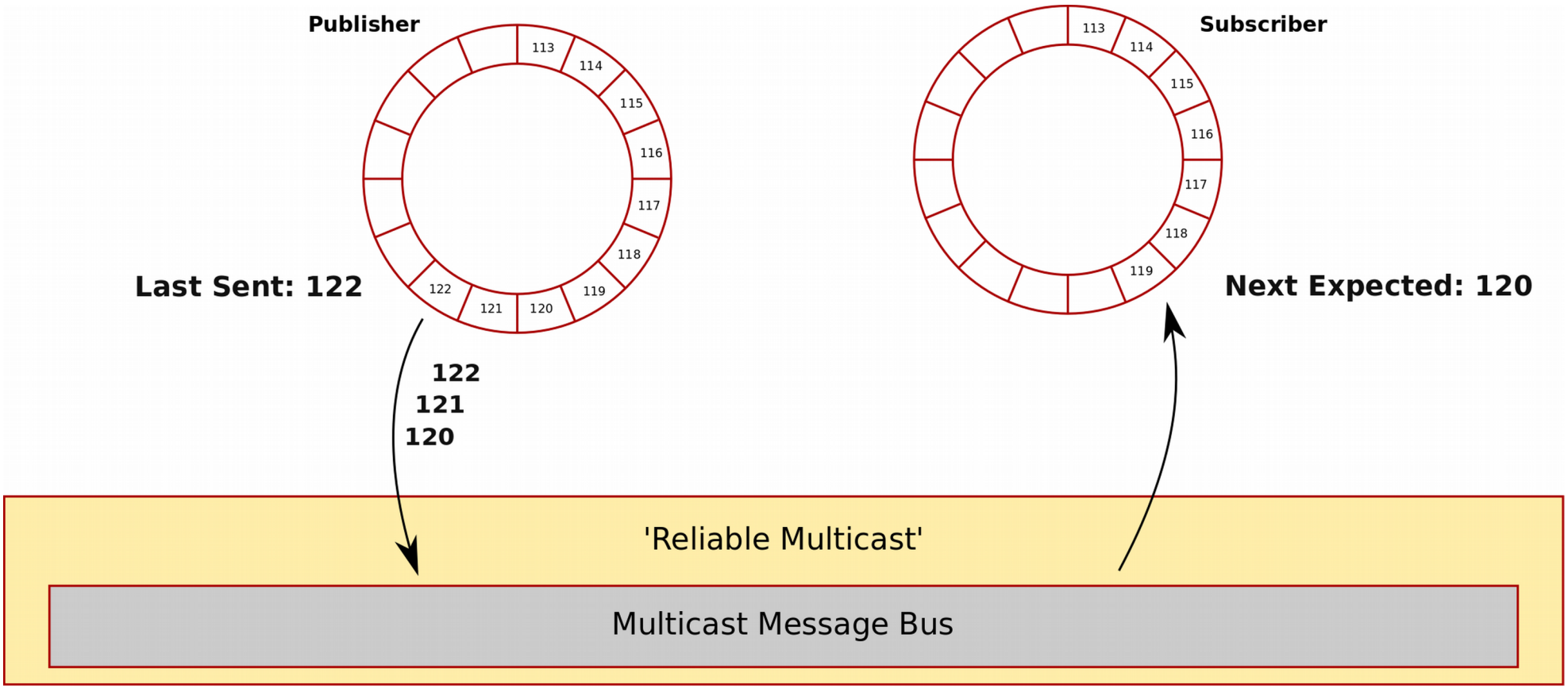
More Reliable Messaging

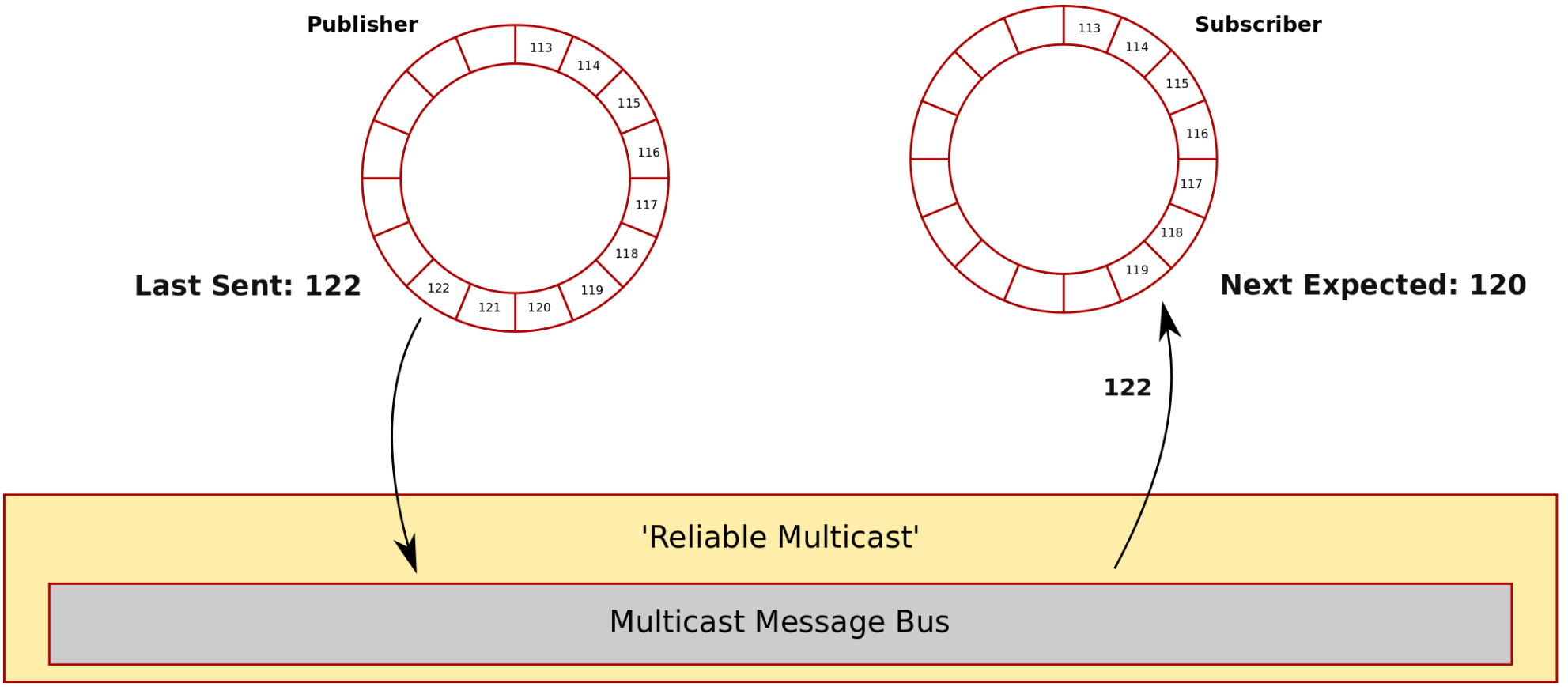

Publisher
113
114
115
116
117
118
119
Last Sent: 116
Subscriber
113
114
115
116
Next Expected: 117
'Reliable Multicast'
Multicast Message Bus

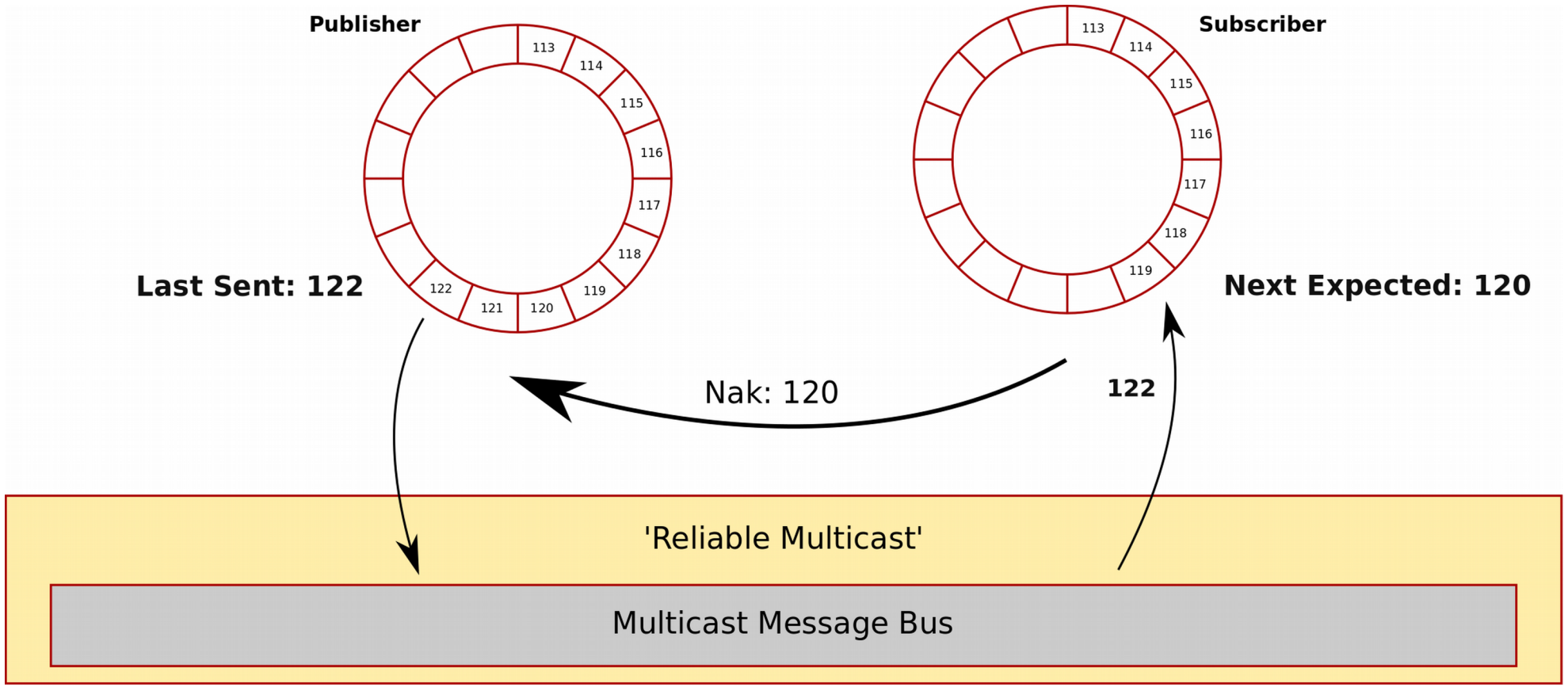

Publisher
113
114
115
116
117
118
119
Last Sent: 119
119
118
117
Subscriber
113
114
115
116
Next Expected: 117
'Reliable Multicast'
Multicast Message Bus

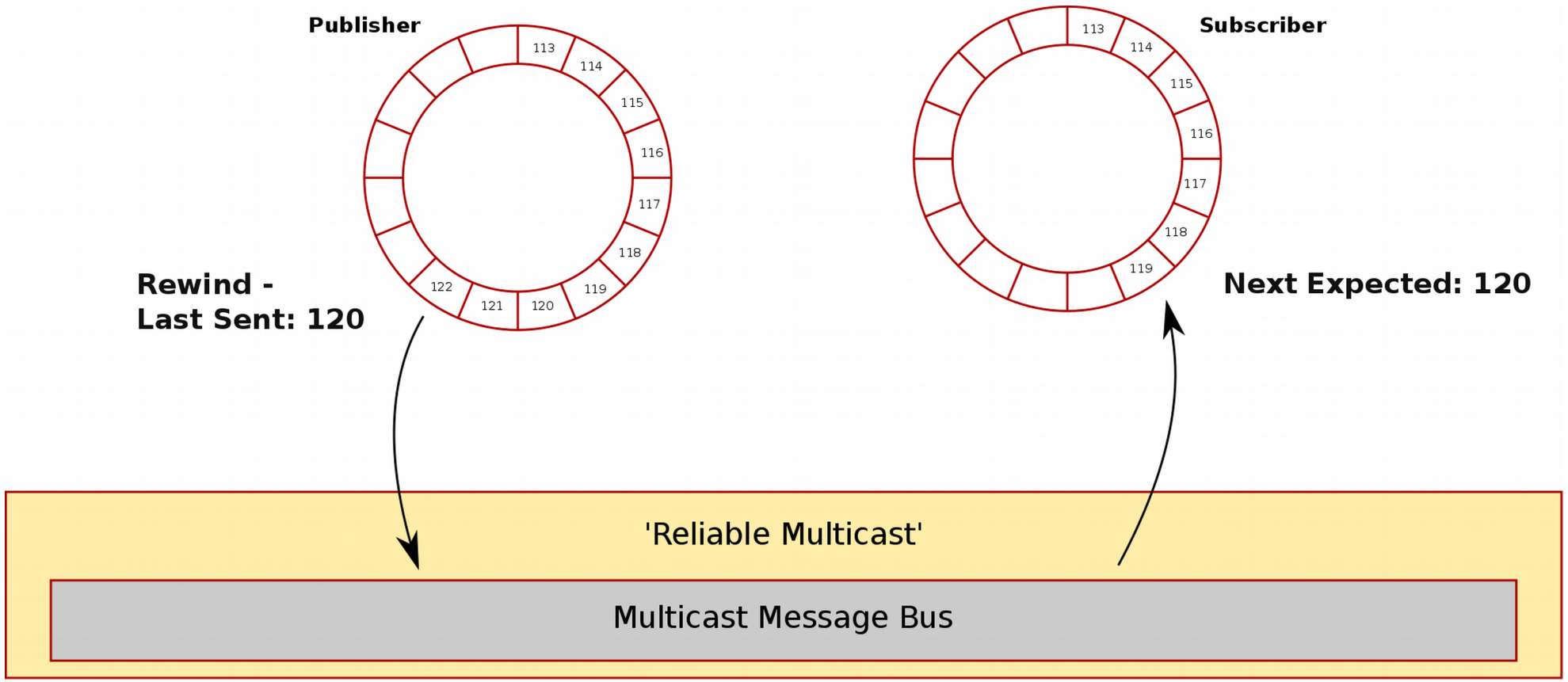

Publisher
113
114
115
116
117
118
119
Last Sent: 119
Subscriber
113
114
115
116
Next Expected: 117
117
118
119
'Reliable Multicast'
Multicast Message Bus

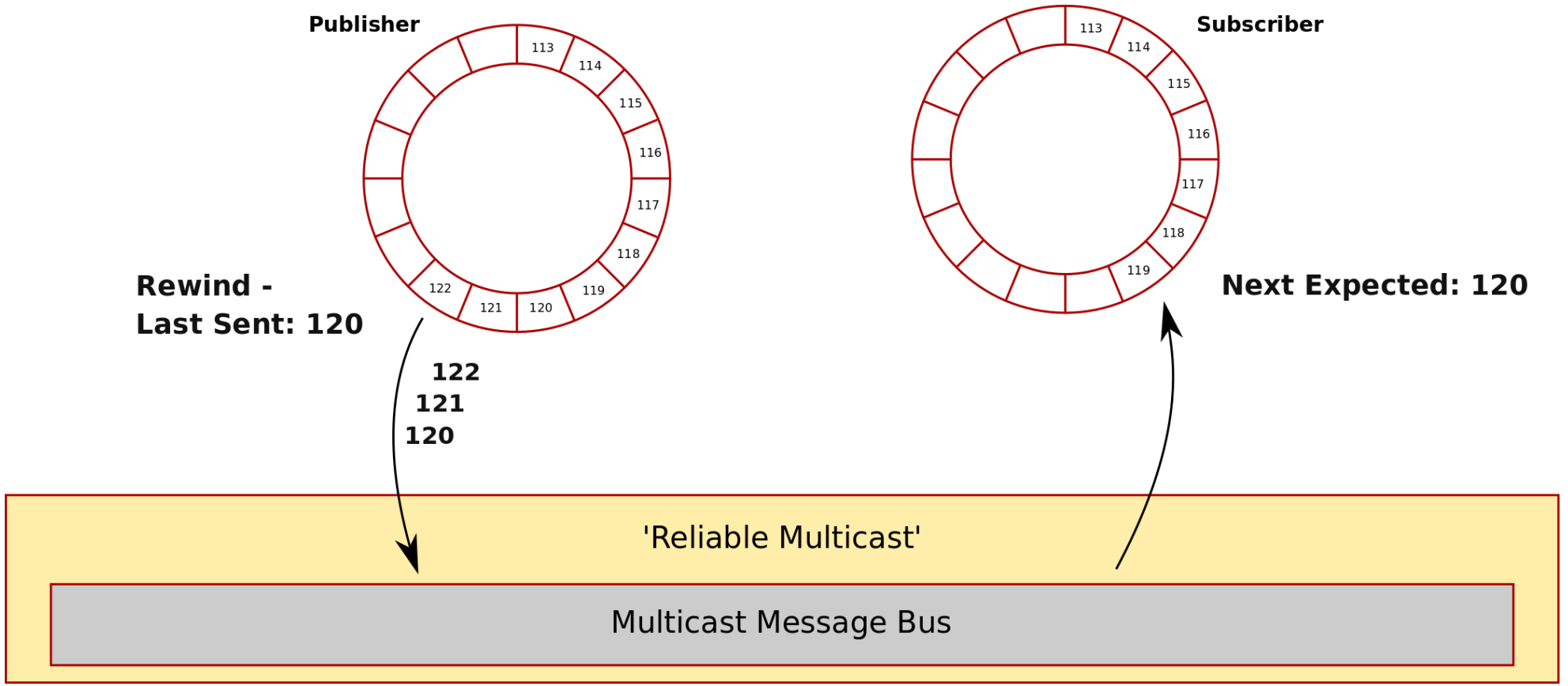

Publisher
113
114
115
116
117
118
119
Last Sent: 119
Subscriber
113
114
115
116
117
118
119
Next Expected: 120
'Reliable Multicast'
Multicast Message Bus


Publisher
113
114
115
116
117
118
119
120
121
122
Last Sent: 122
122
121
120
Subscriber
113
114
115
116
117
118
119
Next Expected: 120
'Reliable Multicast'
Multicast Message Bus


Publisher
113
114
115
116
117
118
119
120
121
122
Last Sent: 122
Subscriber
113
114
115
116
117
118
119
Next Expected: 120
122
'Reliable Multicast'
Multicast Message Bus


Publisher
113
114
115
116
117
118
119
120
121
122
Last Sent: 122
Subscriber
113
114
115
116
117
118
119
Next Expected: 120
Nak: 120
122
'Reliable Multicast'
Multicast Message Bus


Publisher
113
114
115
116
117
118
119
120
121
122
Rewind -
Last Sent: 120
Subscriber
113
114
115
116
117
118
119
Next Expected: 120
'Reliable Multicast'
Multicast Message Bus

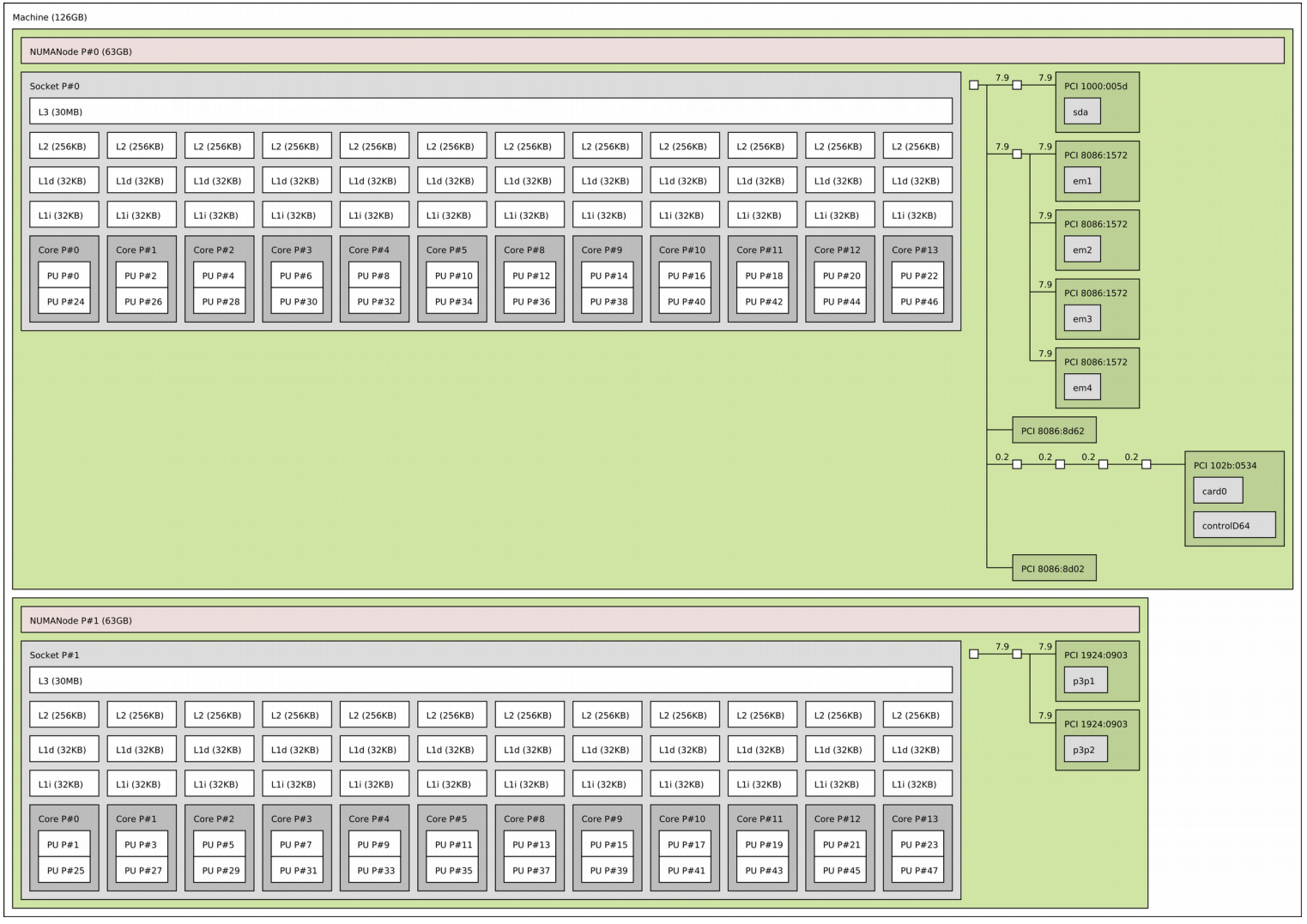

Publisher
113
114
115
116
117
118
119
120
121
122
Rewind -
Last Sent: 120
122
121
120
Subscriber
113
114
115
116
117
118
119
Next Expected: 120
'Reliable Multicast'
Multicast Message Bus

Handling buffer wraps

‘better never than late’
- reset & late join

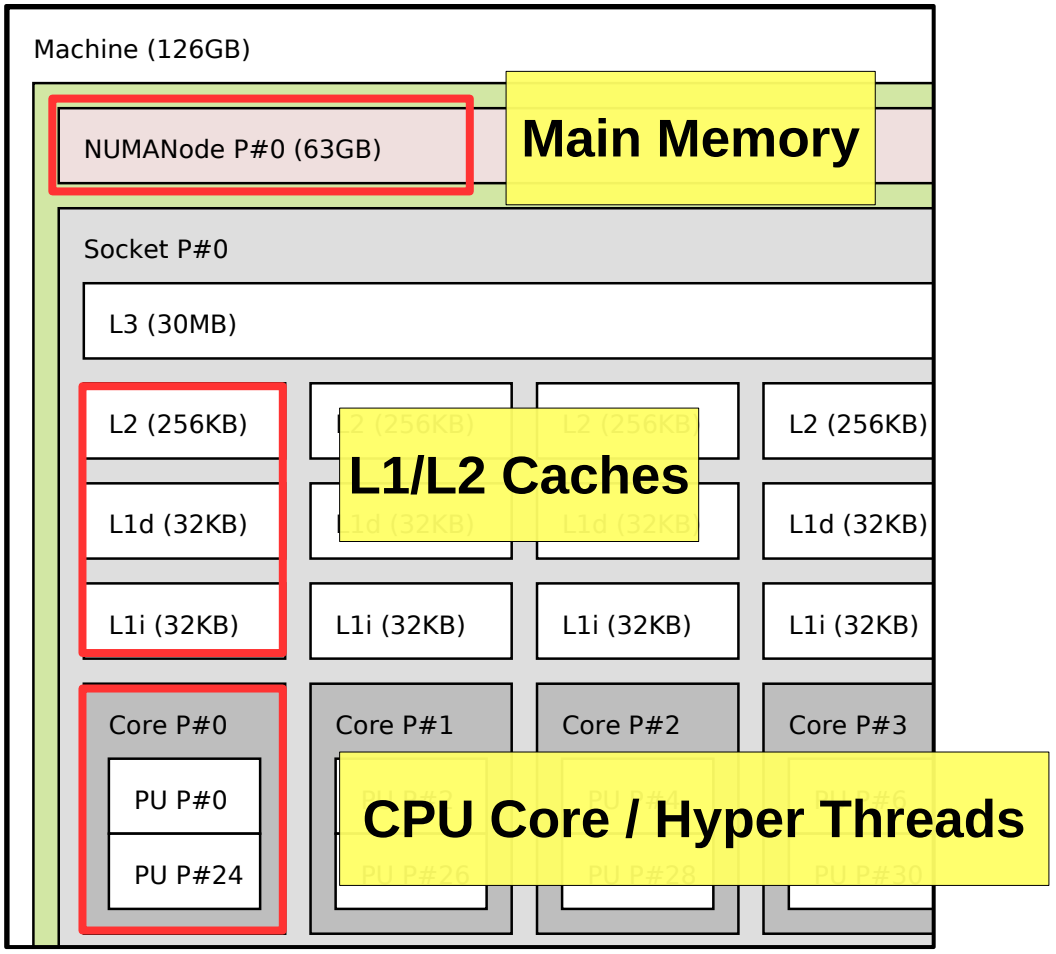
persistent data loss
- recover from event store
- journal replay and gap-fill

Low latency applications: mechanical sympathy



LMAX EXCHANGE

[sam@box ~]$ lstopo


Machine (126GB)
NUMANode P#0 (63GB)
Main Memory
Socket P#0
L3 (30MB)
L2 (256KB)
L1d (32KB)
L1i (32KB)
L1/L2 Caches
Core P#0
PU P#0
PU P#24
Core P#1
Core P#2
Core P#3
CPU Core / Hyper Threads

CPUs are faster than memory

Intel Performance Analysis Guide:

L1 CACHE hit, **4 cycles**

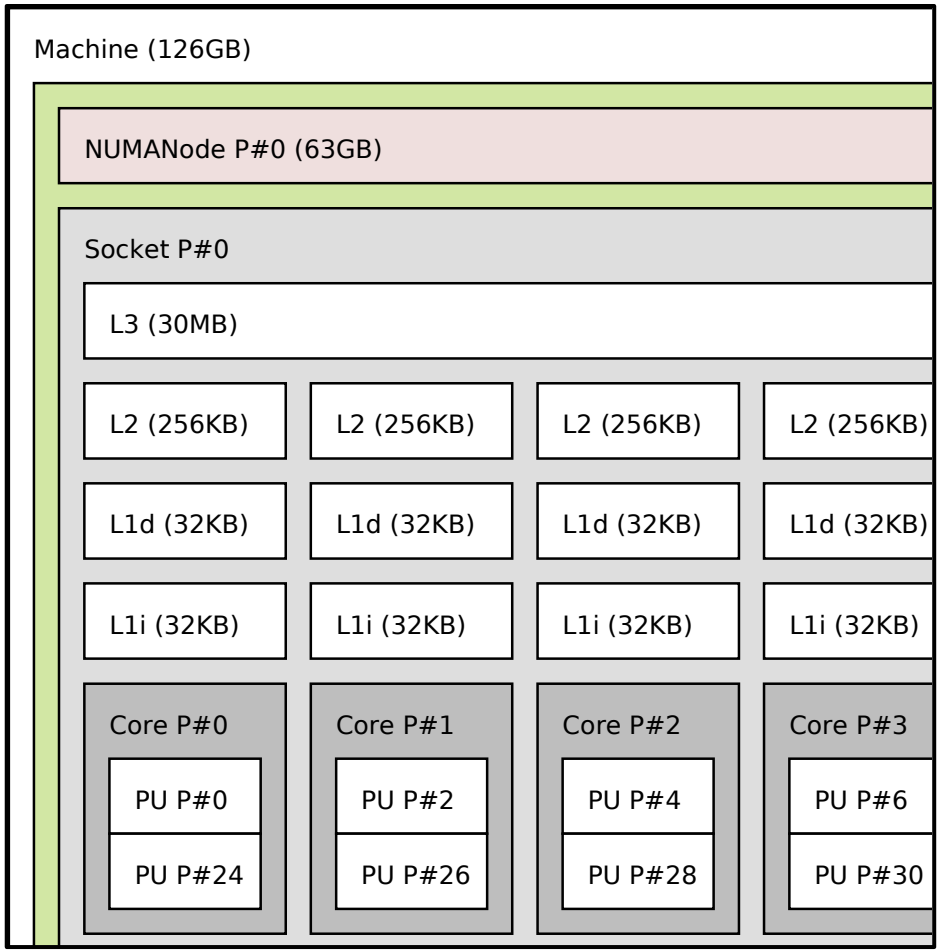
L2 CACHE hit, **10 cycles**

local L3 CACHE hit, **~40-75 cycles**

remote L3 CACHE hit, **~100-300 cycles**

Local Dram **~60 ns**

Remote Dram **~100 ns**



LMAX EXCHANGE

Memory system optimised for:

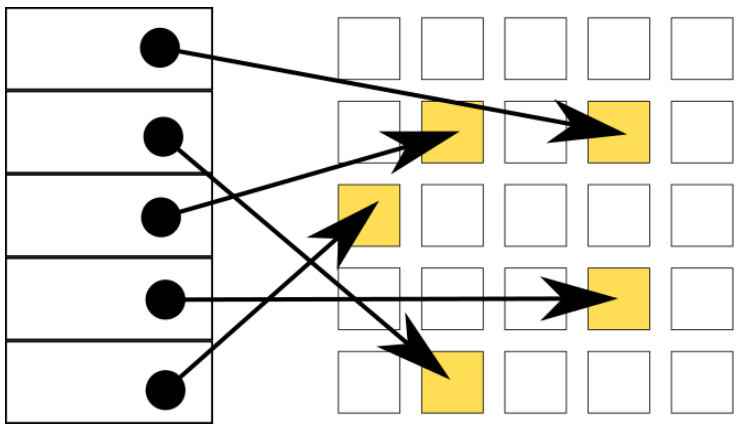
Temporal locality

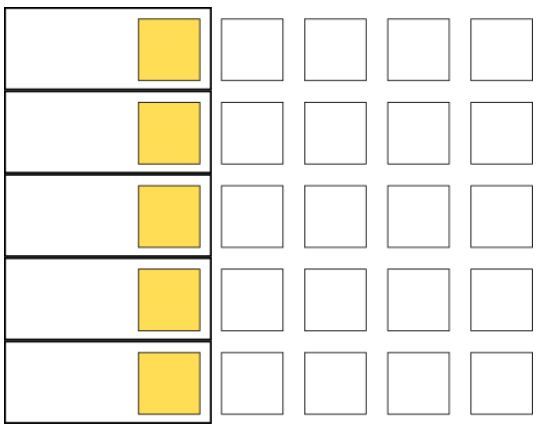
Spatial locality

Equidistant locality

Reference vs Primitives

`Long[]` vs `long[]`





Calculations with money

- `double`: inexact
- `BigDecimal`: expensive

Fixed-point arithmetic with `long`

But I want type-safety...

```
public class Cash
{
    long value;
}
```

Prices, precision: 6dp

1250000L → 1.250000

Quantities, precision: 2dp

1520L → 15.20

```
long price1 = 1250000L;
long quantity1 = 1520L;

// BUG
long price2 = quantity1;
```

Prices, precision: 6dp

1250000L → 1.250000

Quantities, precision: 2dp

1520L → 15.20

With Type Annotations & Units Checker:

```
@Price long price1 = 1250000L;
@Qty   long quantity1 = 1520L;

// Compilation error
@Price long price2 = quantity1;
```

https://checkerframework.org/

LMAX EXCHANGE

java.util vs fastutil

Map<Long,X> vs LongMap<X>

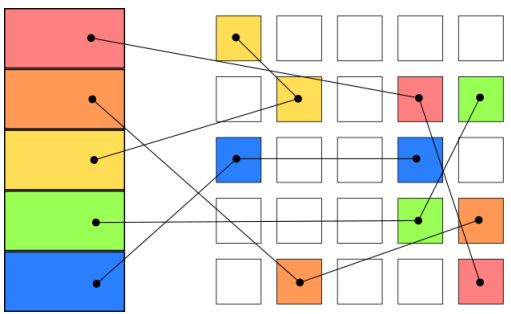
```
public class HashMap<K,V>
{
  Node<K,V>[] table;
  static class Node<K,V>
  {
    K key;
    V value;
    Node<K,V> next;
  }
```

```
public class Long2ObjectOpenHashMap<V>
{
  long[] keys;
  V[] values;
}
```

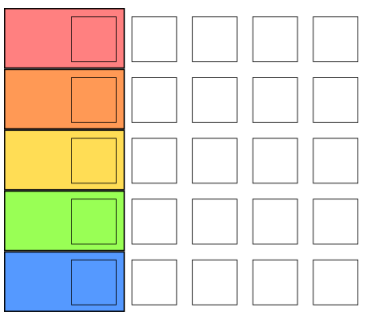
java.util vs fastutil

Map<Long,X> vs LongMap<X>

```
public class HashMap<K,V>
{
  Node<K,V>[] table;
  static class Node<K,V>
  {
    K key;
    V value;
    Node<K,V> next;
  }
```



```
public class Long2ObjectOpenHashMap<V>
{
  long[] keys;
  V[] values;
}
```



False sharing: revisit the Disruptor

```
public class ArrayBlockingQueue<E>
{
  final Object[] items;
  int takeIndex;
  int putIndex;
  int count;

  /** Main lock guarding all access */
  final ReentrantLock lock;
}
```

False sharing: revisit the Disruptor

```
public class RingBuffer
{
  // ...
  final Object[] entries;
  final Sequence cursor;
  // ...
}
```

```
public class Sequence
{
  long p1, p2, p3, p4, p5, p6, p7;
  long value;
  long p9, p10, p11, p12, p13, p14, p15;
}
```

False sharing: revisit the Disruptor

```
public class RingBuffer
{
  // ...
  final Object[] entries;
  final Sequence cursor;
  // ...
}
```

Java 8:

```
public class Sequence
{
  @Contended
  long value;
}
```

Removing Jitter:
GC & Scheduling

GC Options:

Zero garbage

Massive heap, GC when convenient

Commercial JVM – Azul Zing

GC Options:

Zero garbage

Massive heap, GC when convenient

Commercial JVM – Azul Zing

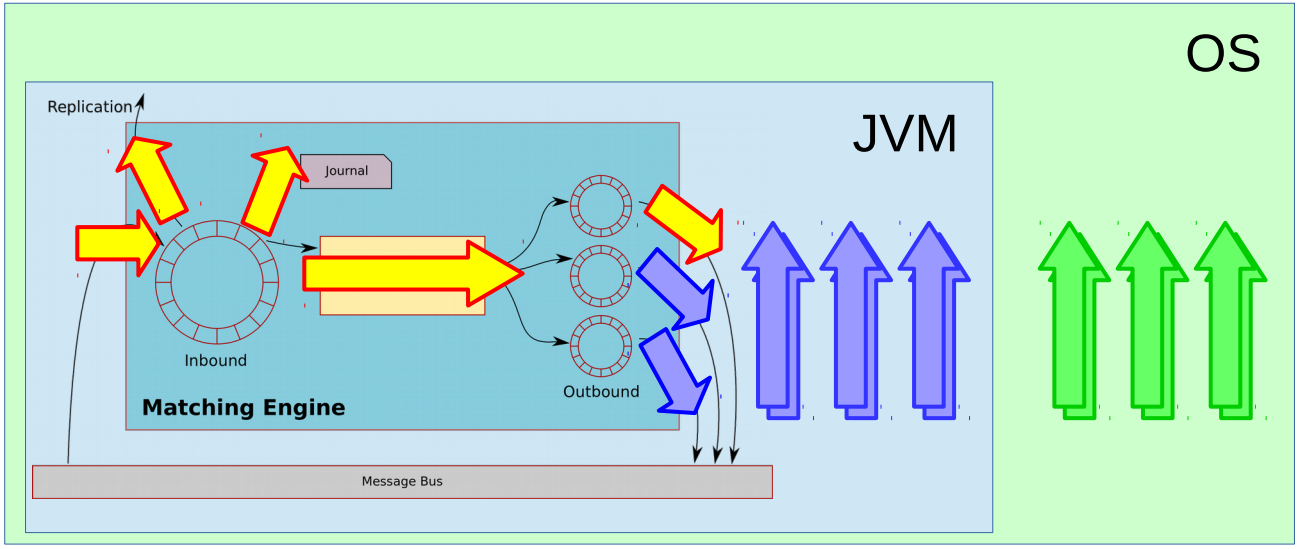
GC Options:

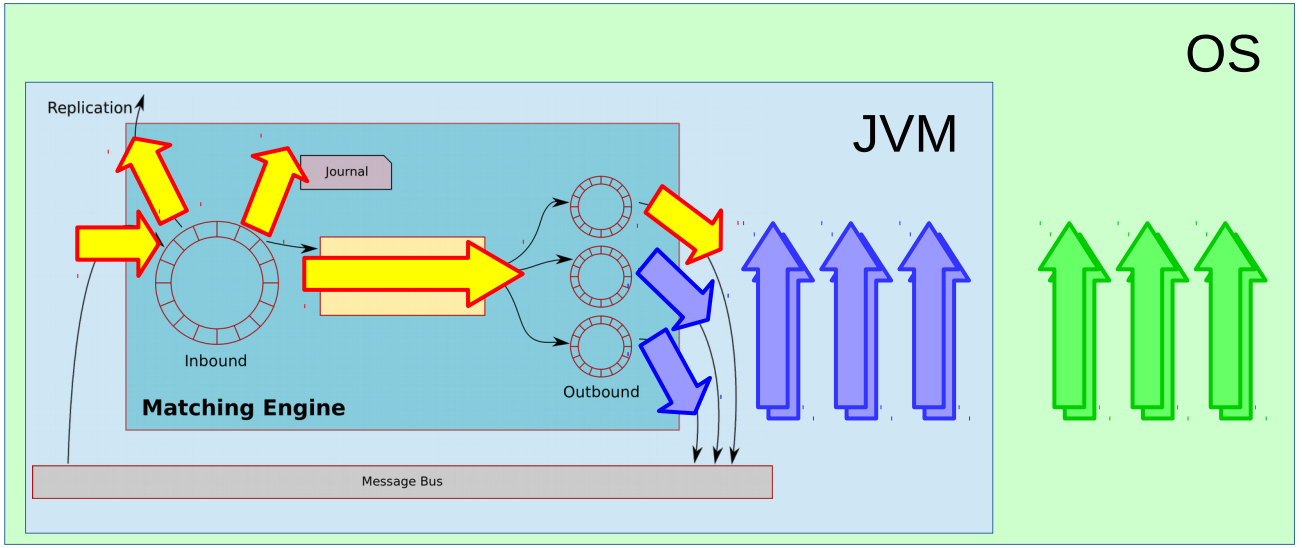
Zero garbage

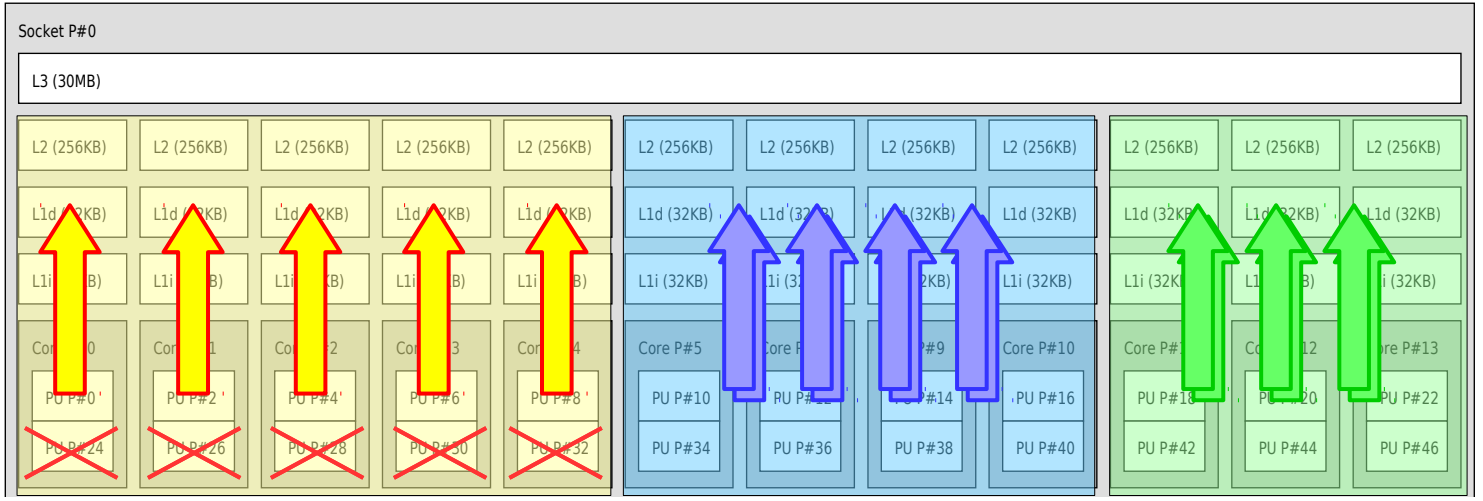
Massive heap, GC when convenient

Commercial JVM – Azul Zing

Avoiding scheduling jitter



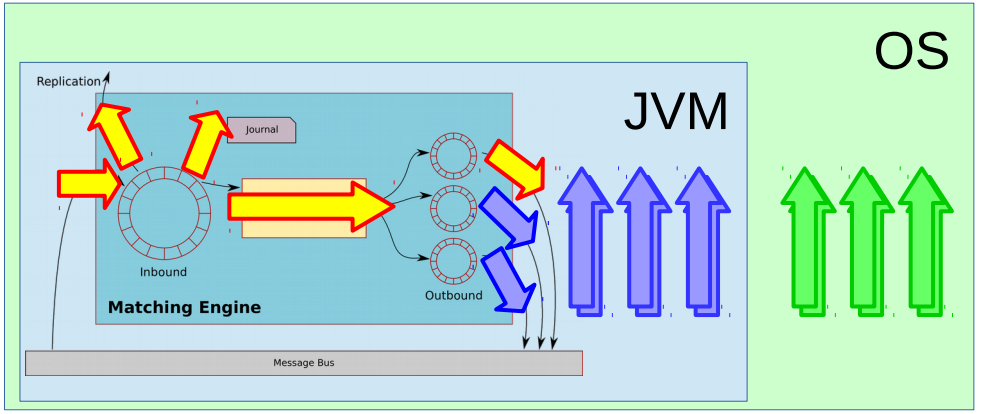


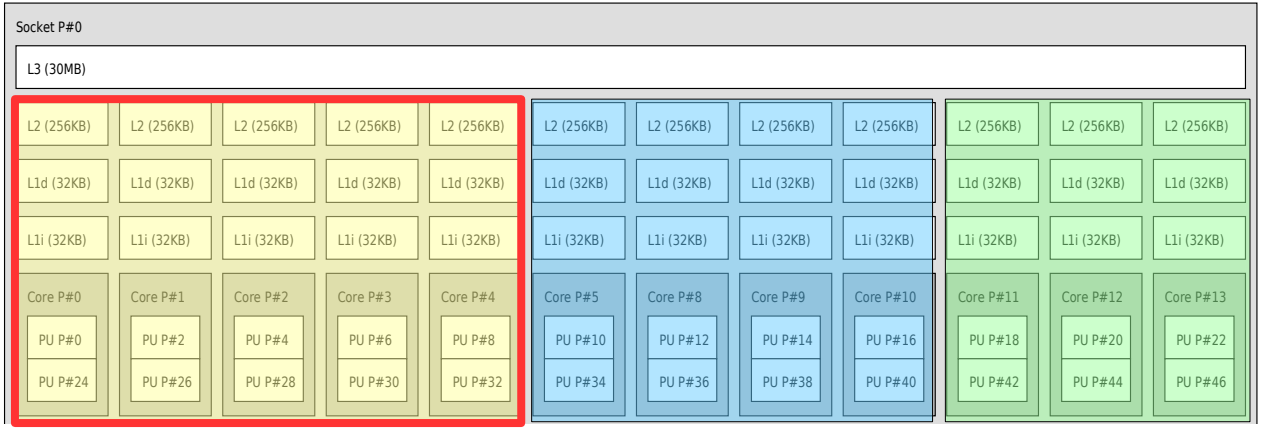


LMAX EXCHANGE

Remove reserved CPUs from the kernel scheduler

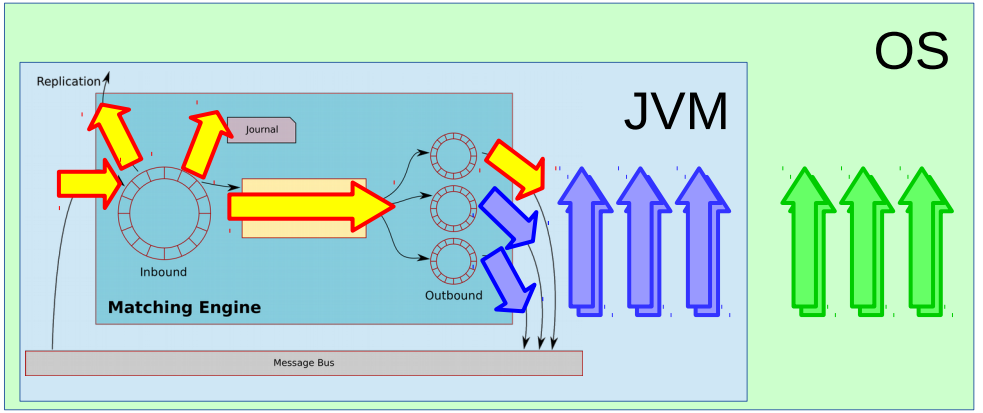
```
isolcpus=0,2,4,6,8,24,26,28,30,32
```

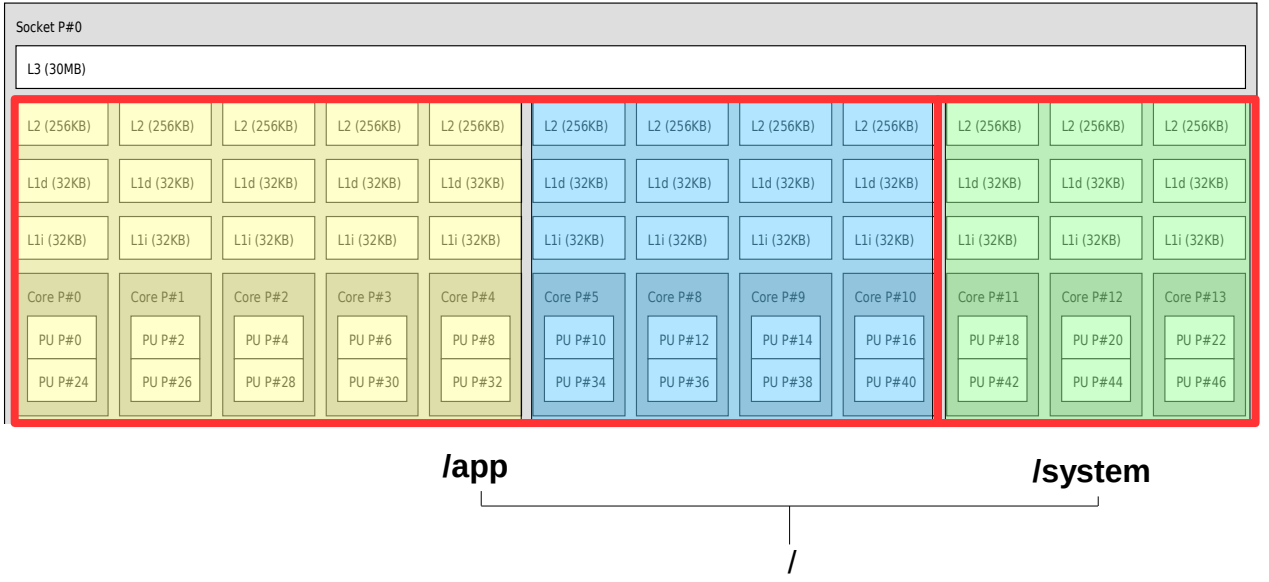




Create CPU sets for system, application

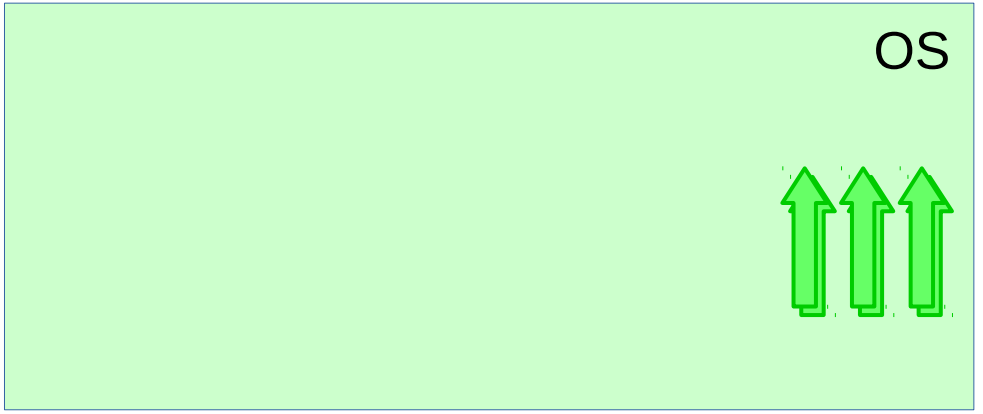
```
# cset set --set=/system --cpu=18,20,...,46

# cset set --set=/app --cpu=0,2,...,40
```

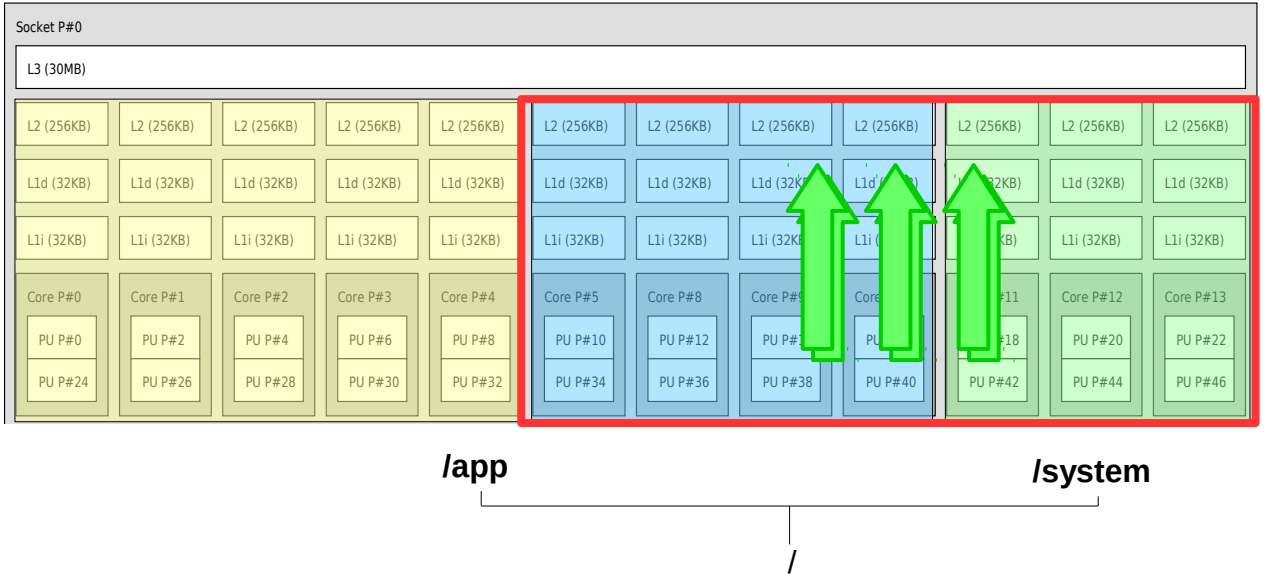




LMAX EXCHANGE

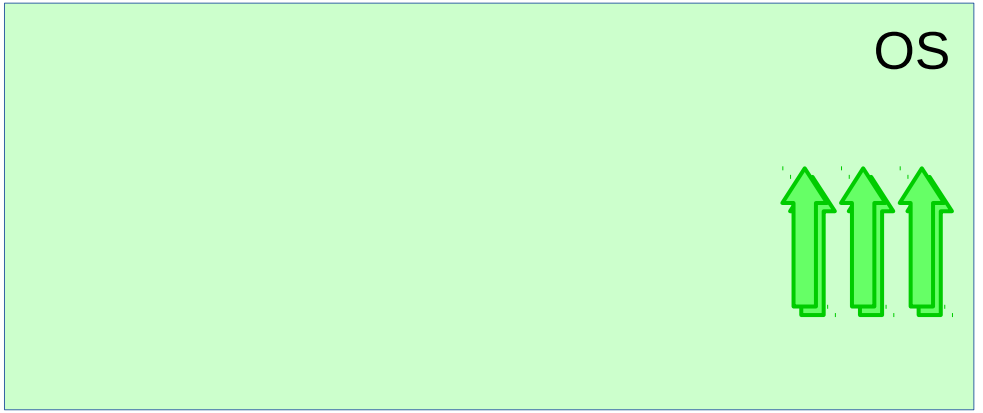
Processes default to the / CPU set

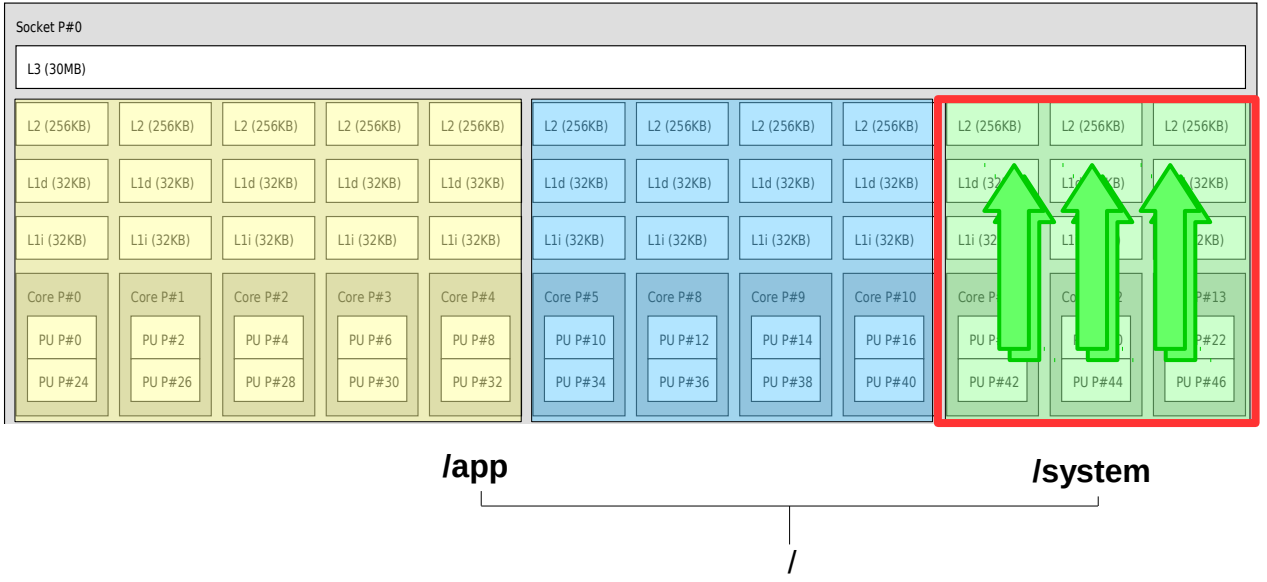




Move all threads into /system CPU set

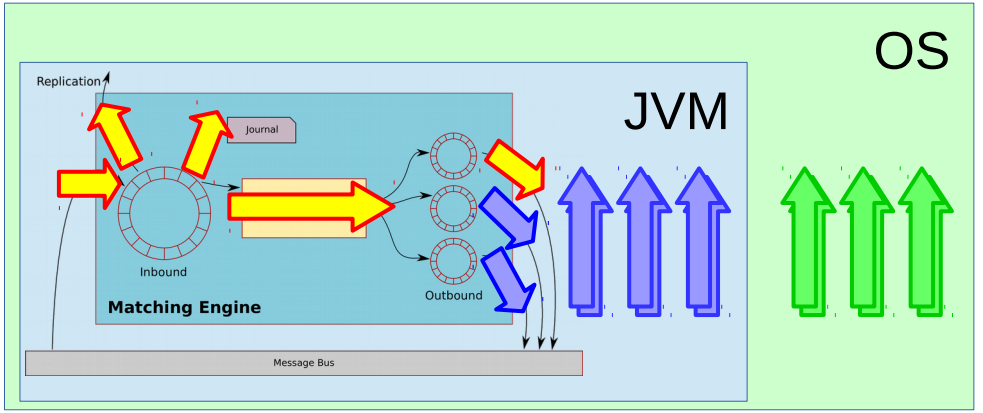
```
# cset proc --move -k --threads --force \
            --from-set=/ --to-set=/system
```

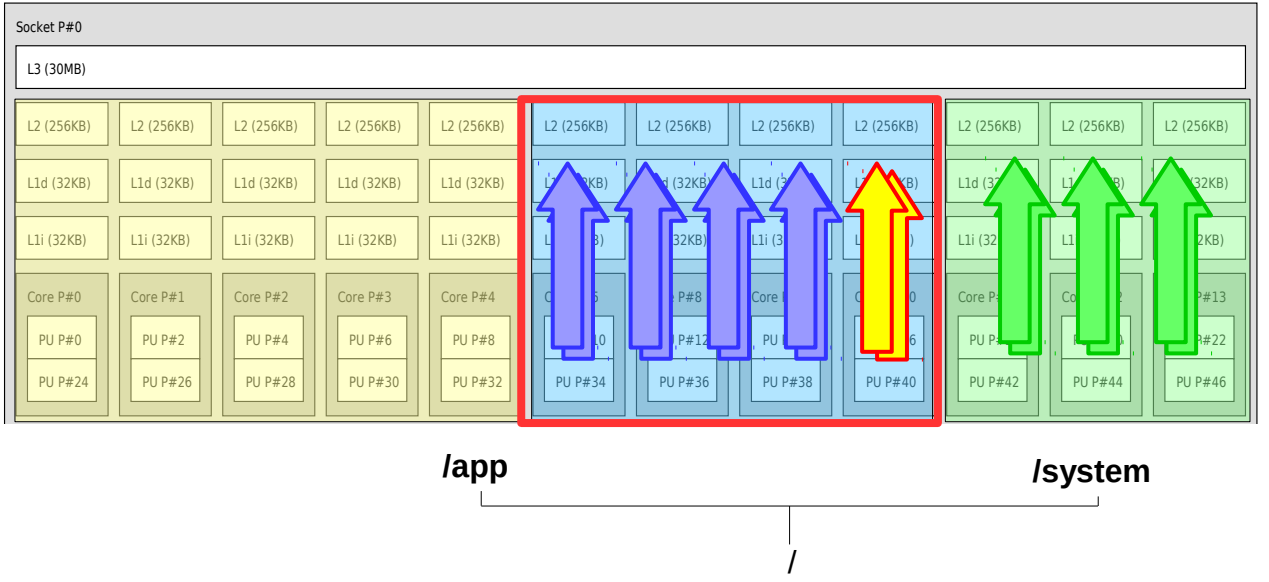




Launch application in /app CPU set, taskset to run in pool

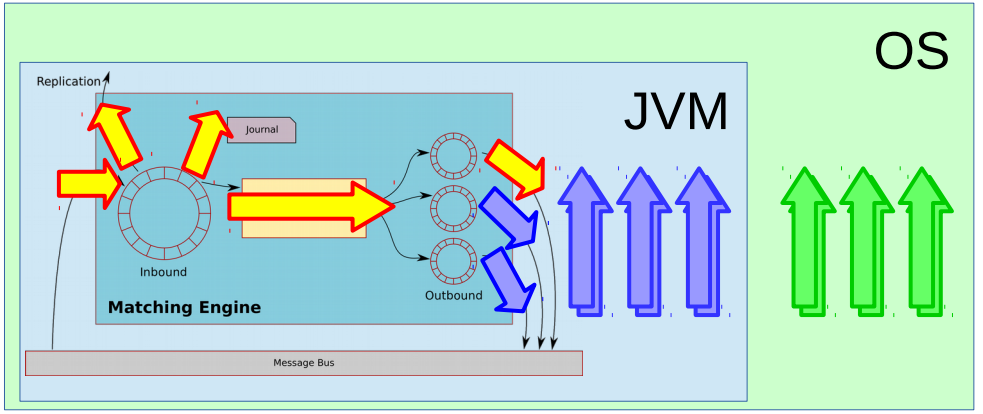
```
$ cset proc --exec /app \
  taskset -cp 10,12...38,40 \
  java <args>
```

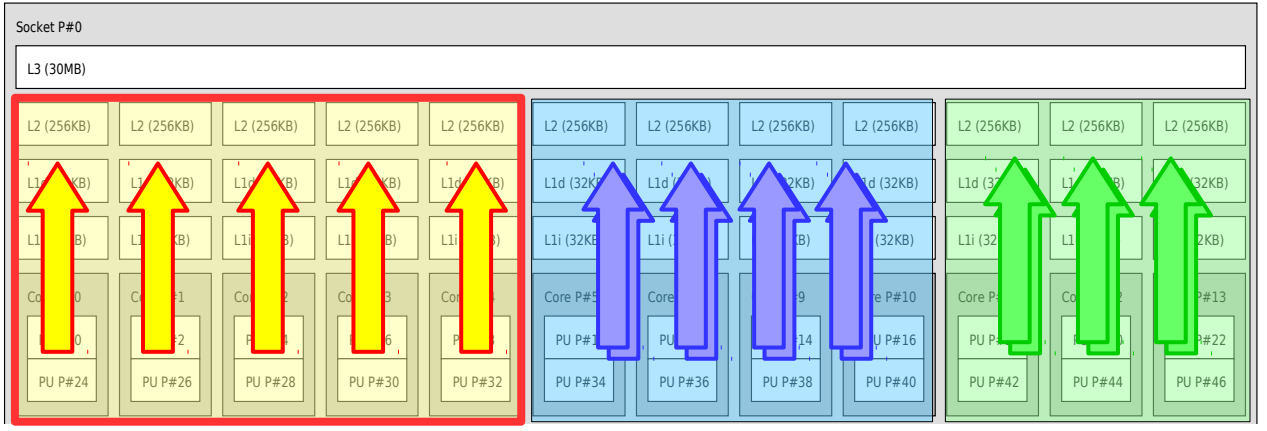




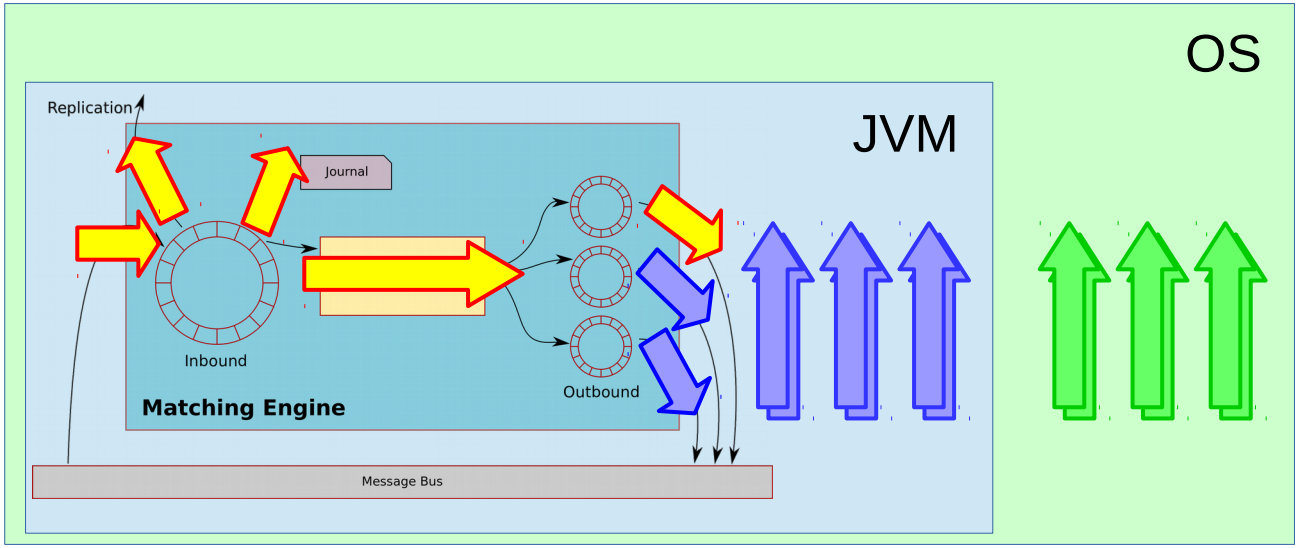
Move critical threads onto their own cores using JNA / JNI

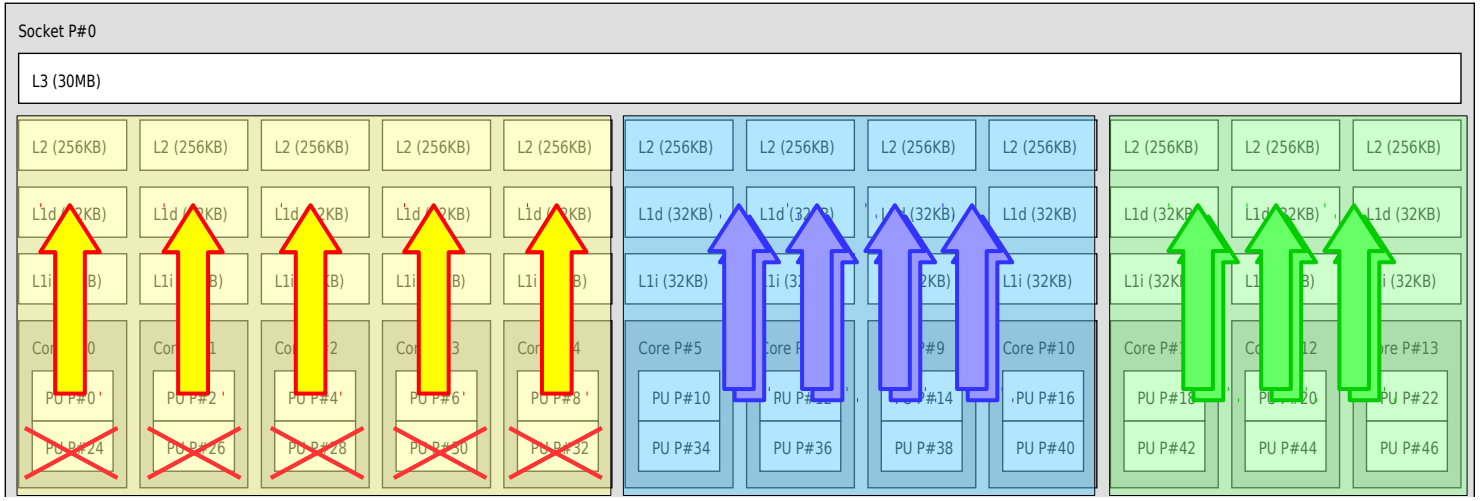
```
sched_set_affinity(0);
sched_set_affinity(2);
...
```





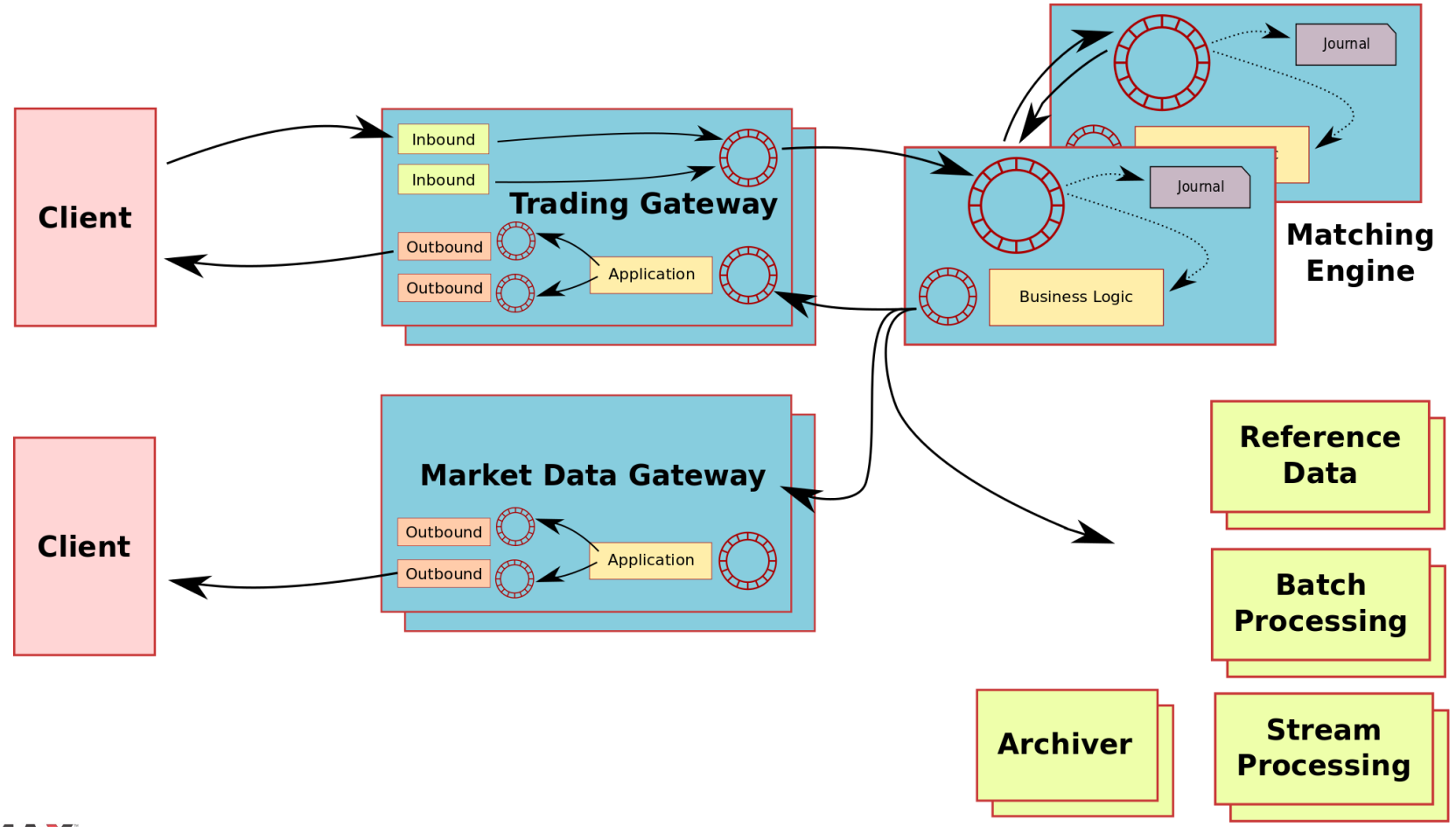
LMAX EXCHANGE



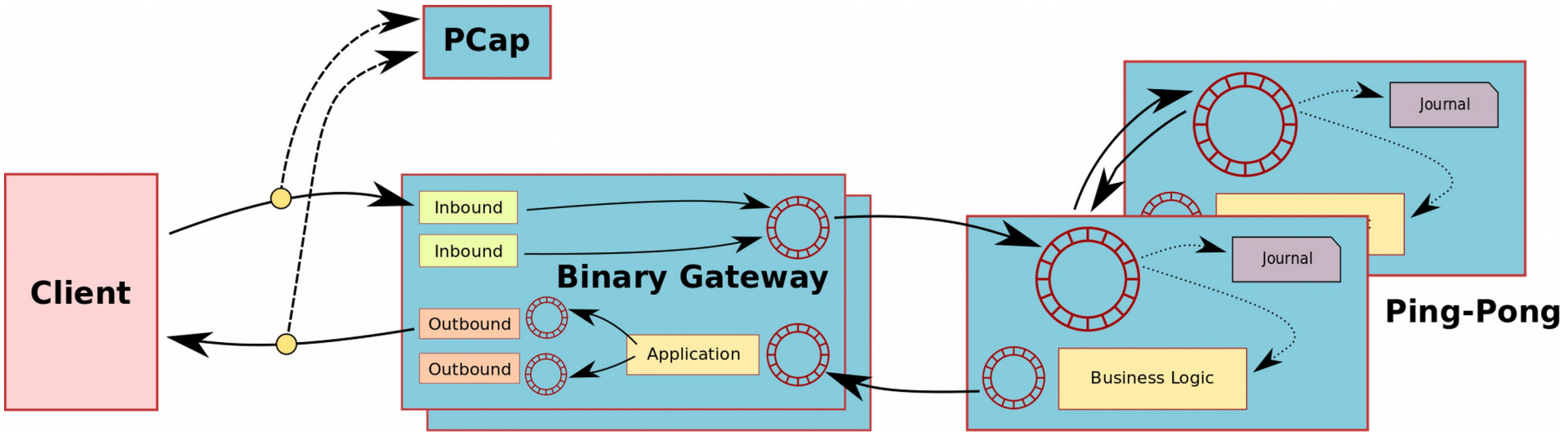


LMAX EXCHANGE

Summary

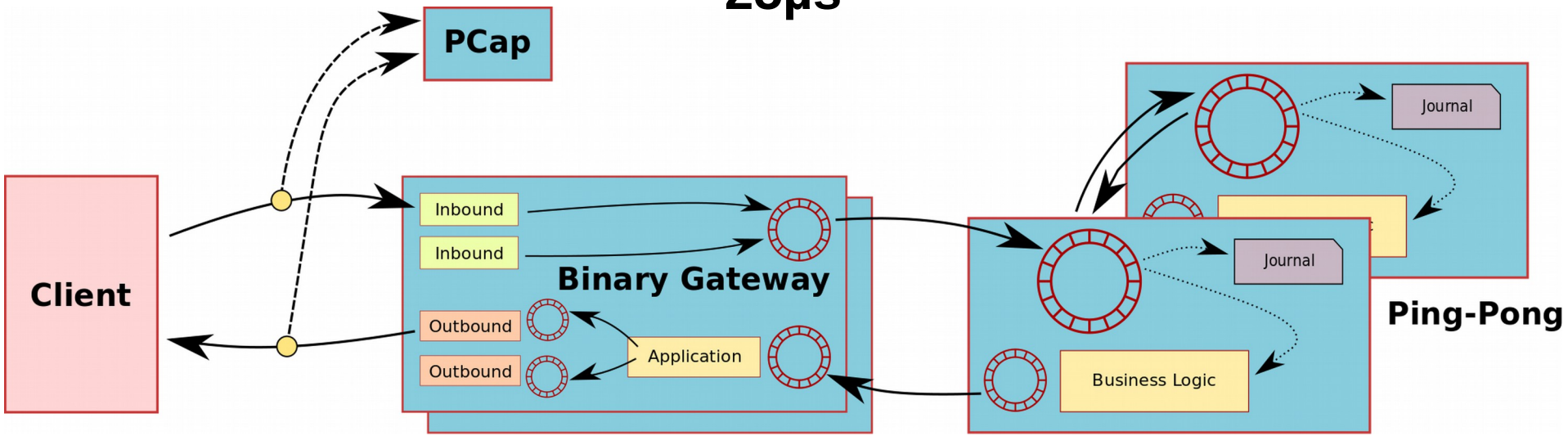

Client
Inbound
Inbound
Trading Gateway
Outbound
Outbound
Application
Journal
Journal
Business Logic
Matching Engine
Client
Market Data Gateway
Outbound
Outbound
Application
Reference Data
Batch Processing
Archiver
Stream Processing

round-trip a correlation ID



round-trip a correlation ID

25µs



Thank You!

sam.adams@lmax.com

https://www.lmax.com/blog/staff-blogs/

p.s. we're hiring!

The End.