LMAX Disruptor 3.0

Advanced Patterns and details
(Making the fast, faster)

@mikeb2701

Agenda

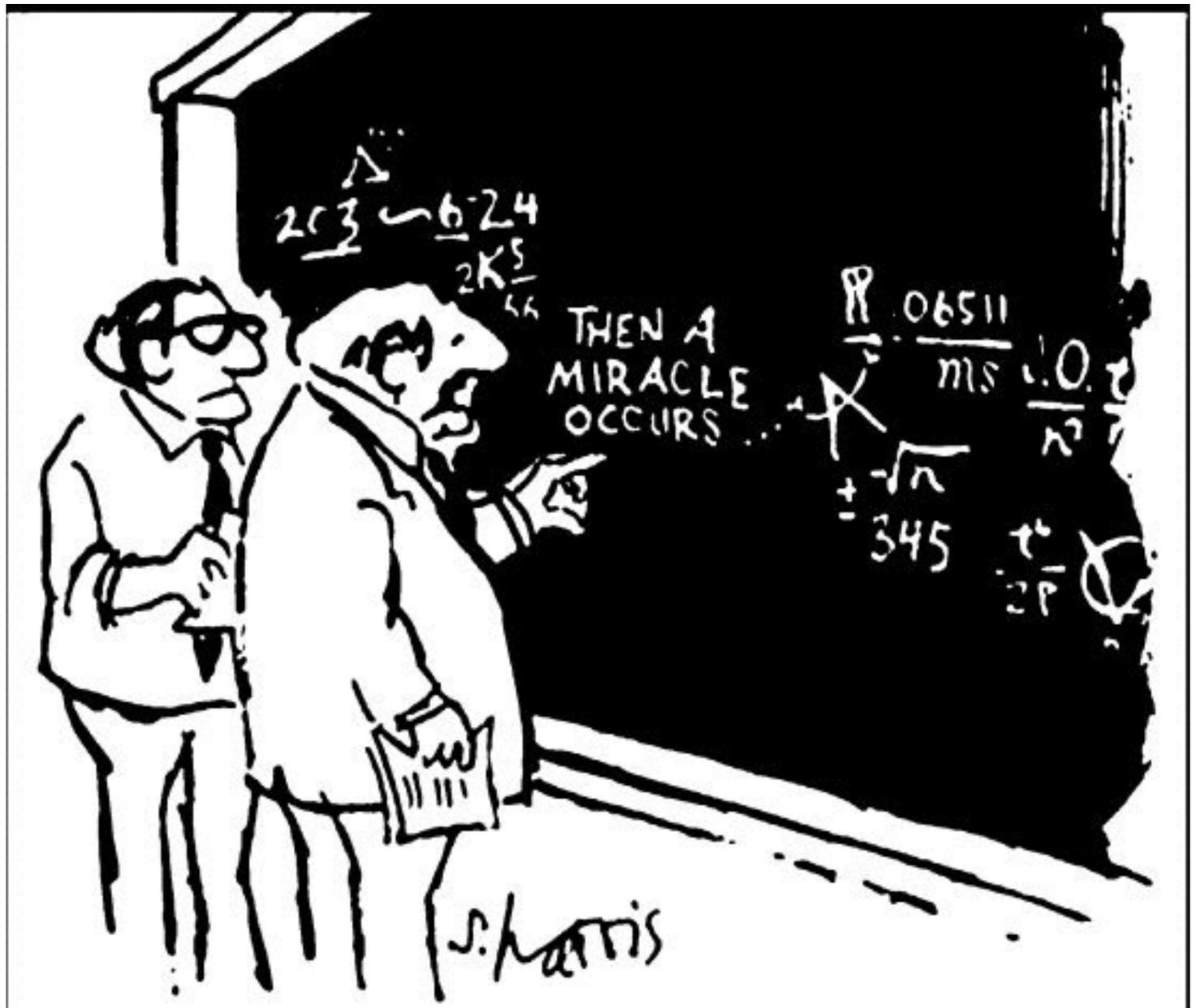
- Prove that memory access is key to software performance

What is LMAX?

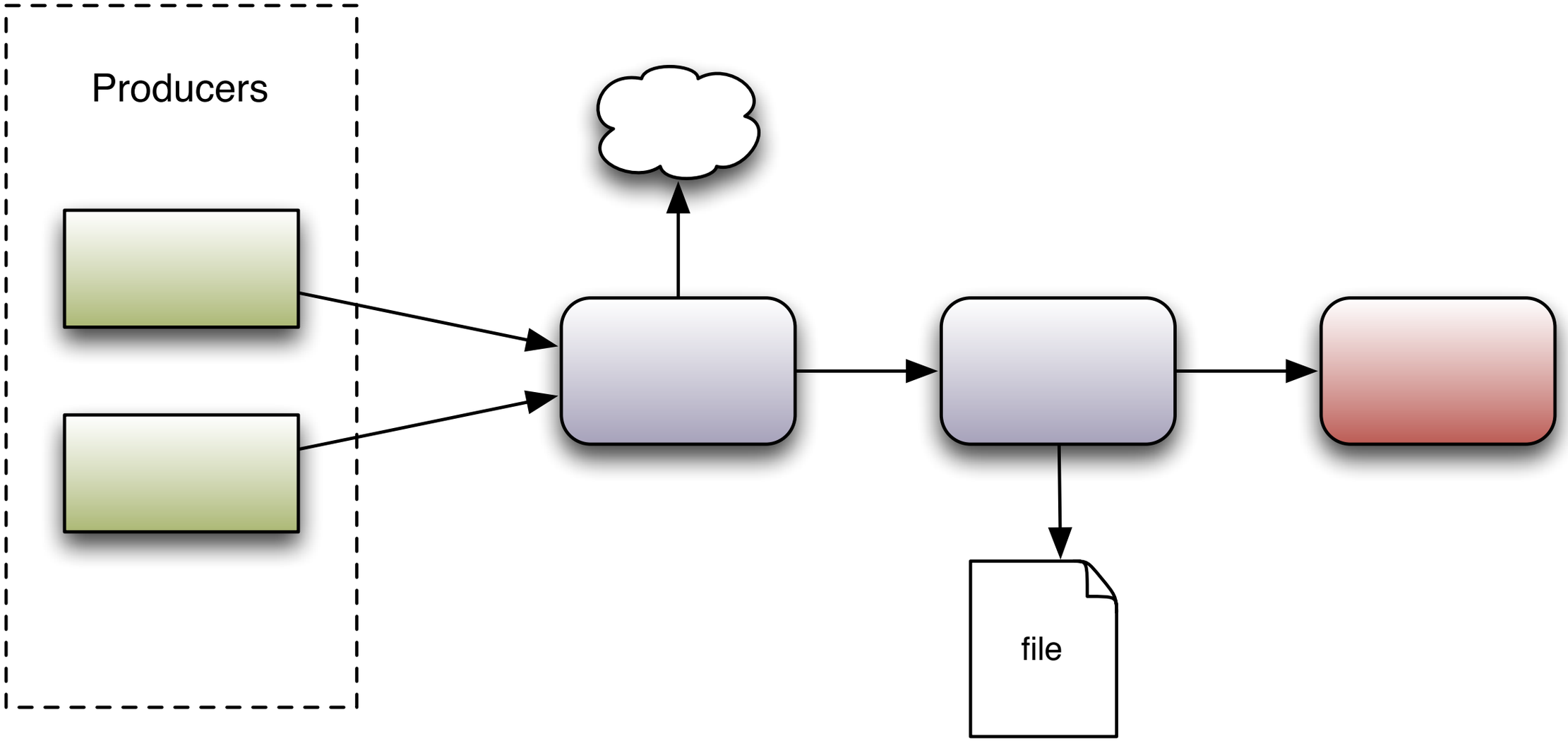
What is LMAX?

- And why do we care about performance?

Business Logic?

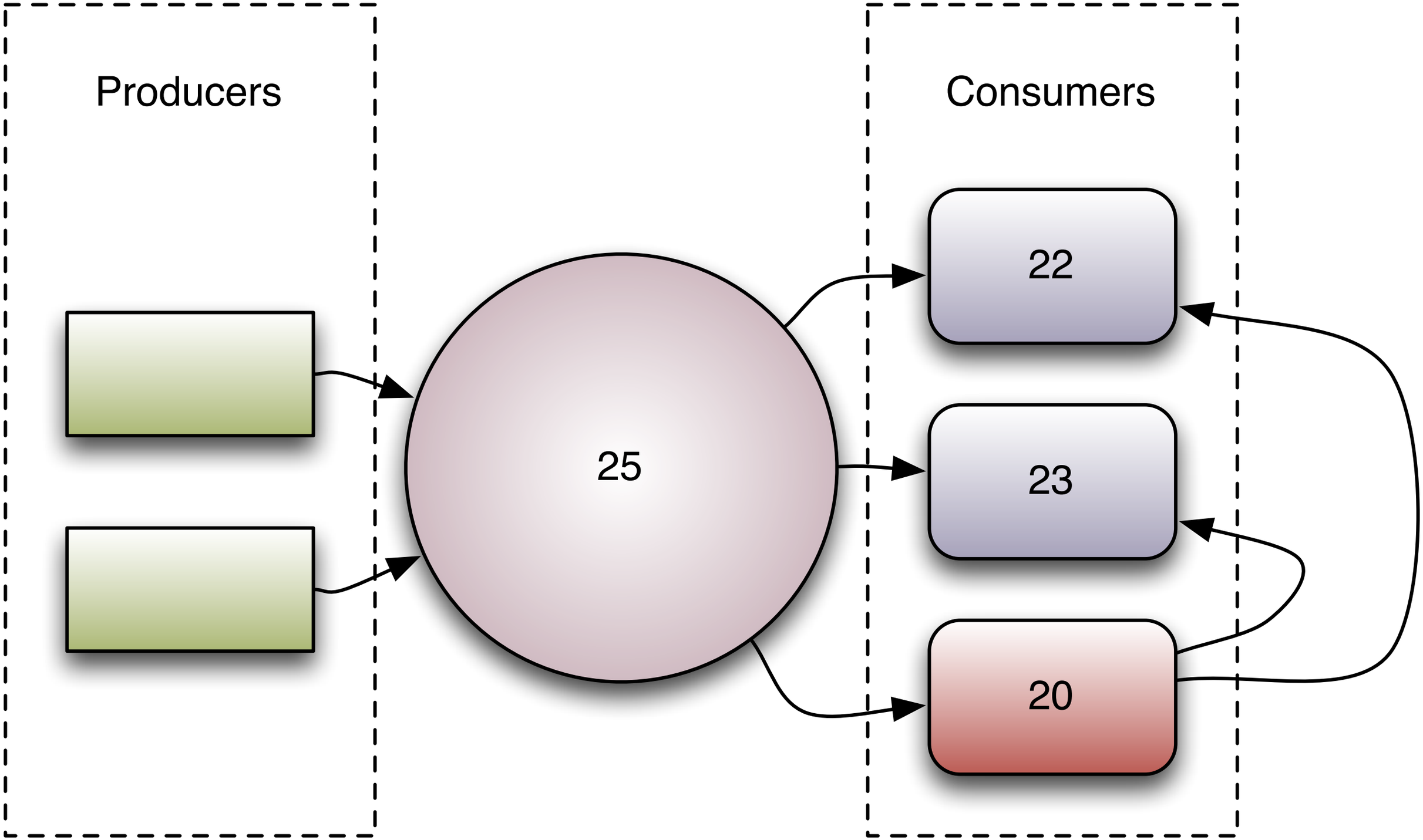


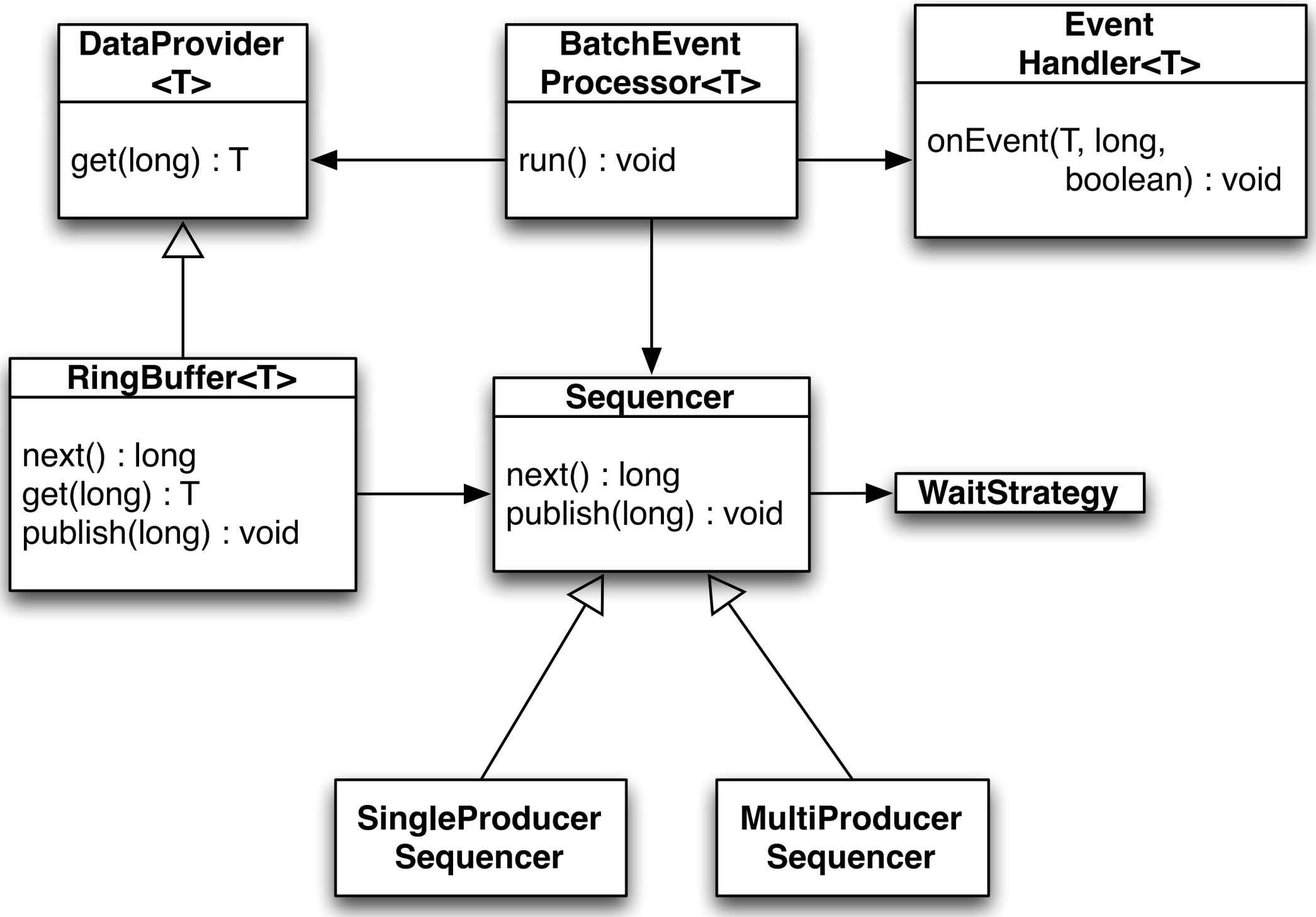
"I think you should be more explicit here in step two."


Producers
file

What did we want?

- Multicast - Parallel consumers
- Pre-allocation
- Optionally Lock-free


Producers
Consumers
25
22
23
20


DataProvider
<T>
get(long) : T
BatchEvent
Processor<T>
run() : void
Event
Handler<T>
onEvent(T, long,
boolean) : void
RingBuffer<T>
next() : long
get(long) : T
publish(long) : void
Sequencer
next() : long
publish(long) : void
WaitStrategy
SingleProducer
Sequencer
MultiProducer
Sequencer

```
class PlayerMove {
    long id;
    long direction;
    long distance;
}
```

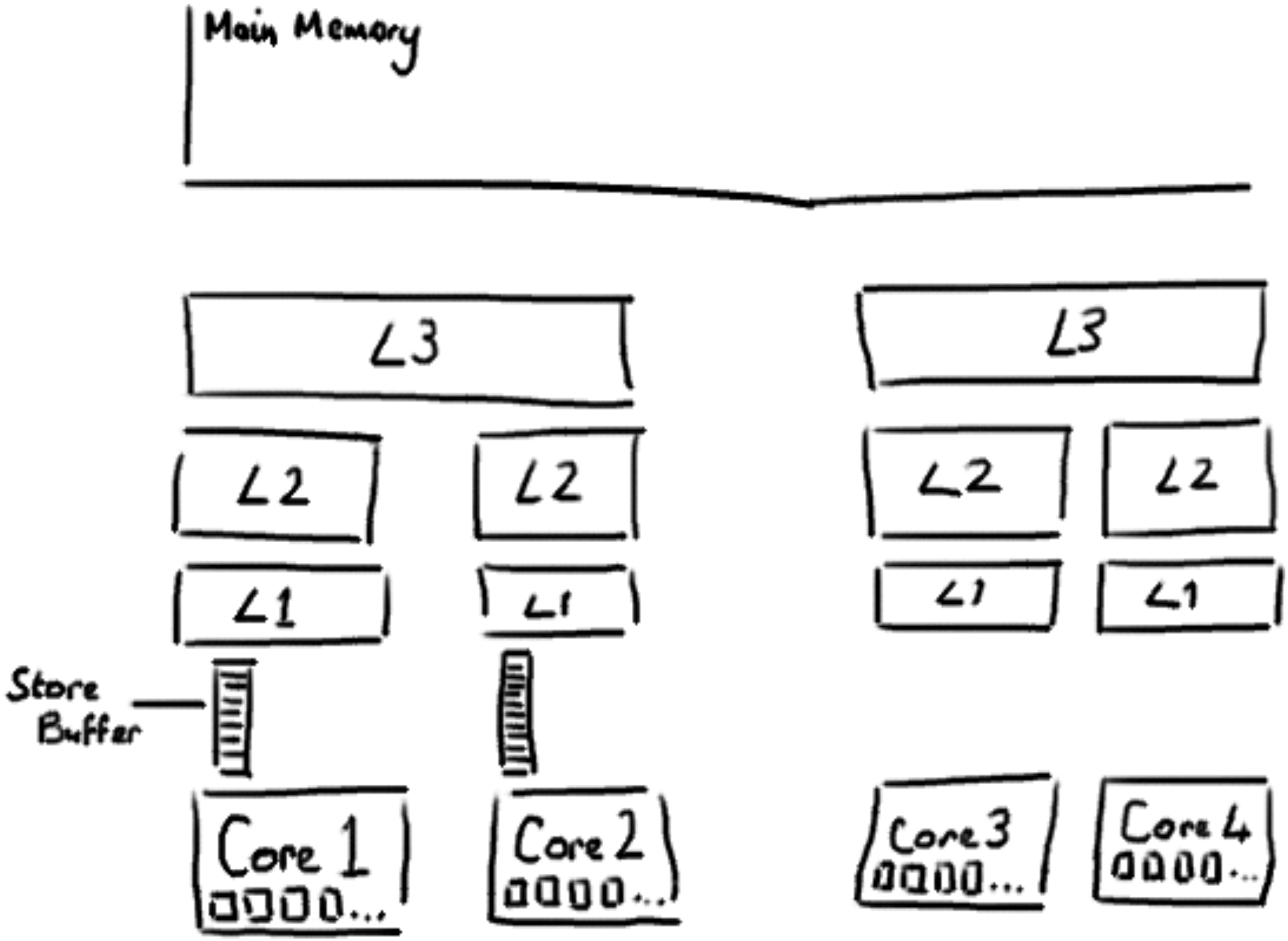
```
class PlayerMoveFactory
implements EventFactory<PlayerMove> {

    public PlayerMove newInstance() {
        return new PlayerMove();
    }
}
```

```
class NetworkHandler {
    RingBuffer<PlayerMove> buffer;

    void handle(ByteBuffer packet) {

        long next = buffer.next();
        try {
            PlayerMove playerMove = buffer.get(next);
            playerMove.id = packet.getLong();
            playerMove.direction = packet.getLong();
            playerMove.distance = packet.getLong();
        } finally {
            buffer.publish(next);
        }
    }
}
```

```
class PlayerHandler
implements EventHandler<PlayerMove> {

    public void onEvent(PlayerMove event,
                        long sequence,
                        boolean onBatchEnd) {

        Player player = findPlayer(event.id);
        player.move(event.direction, event.distance);
    }
}
```

2 Common Cases

- Immutable Reference Objects
- Serialised Objects


Main Memory
L3
L3
L2
L2
L2
L2
L1
L1
L1
L1
Store Buffer
Core 1
Core 2
Core 3
Core 4

- Temporal Locality
- Spatial Locality
- Striding

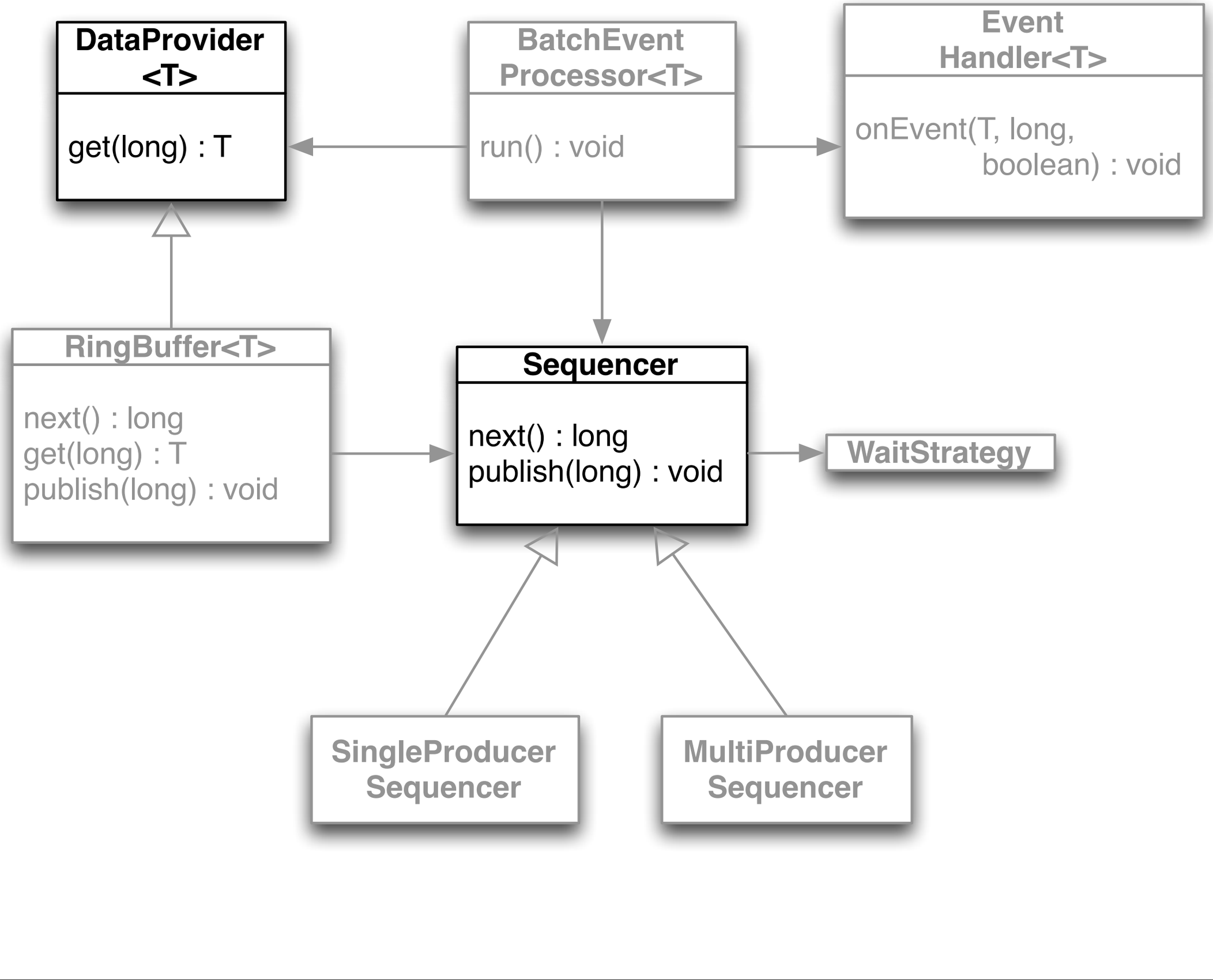
- Temporal Locality
- Spatial Locality
- Striding

- Temporal Locality
- Spatial Locality
- Striding

- L1-dcache-loads
- L1-dcache-misses
- LLC-loads
- LLC-misses
- dTLB-loads
- dTLB-misses

- L1-dcache-loads
- L1-dcache-misses
- LLC-loads
- LLC-misses
- dTLB-loads
- dTLB-misses

- L1-dcache-loads
- L1-dcache-misses
- LLC-loads
- LLC-misses
- dTLB-loads
- dTLB-misses

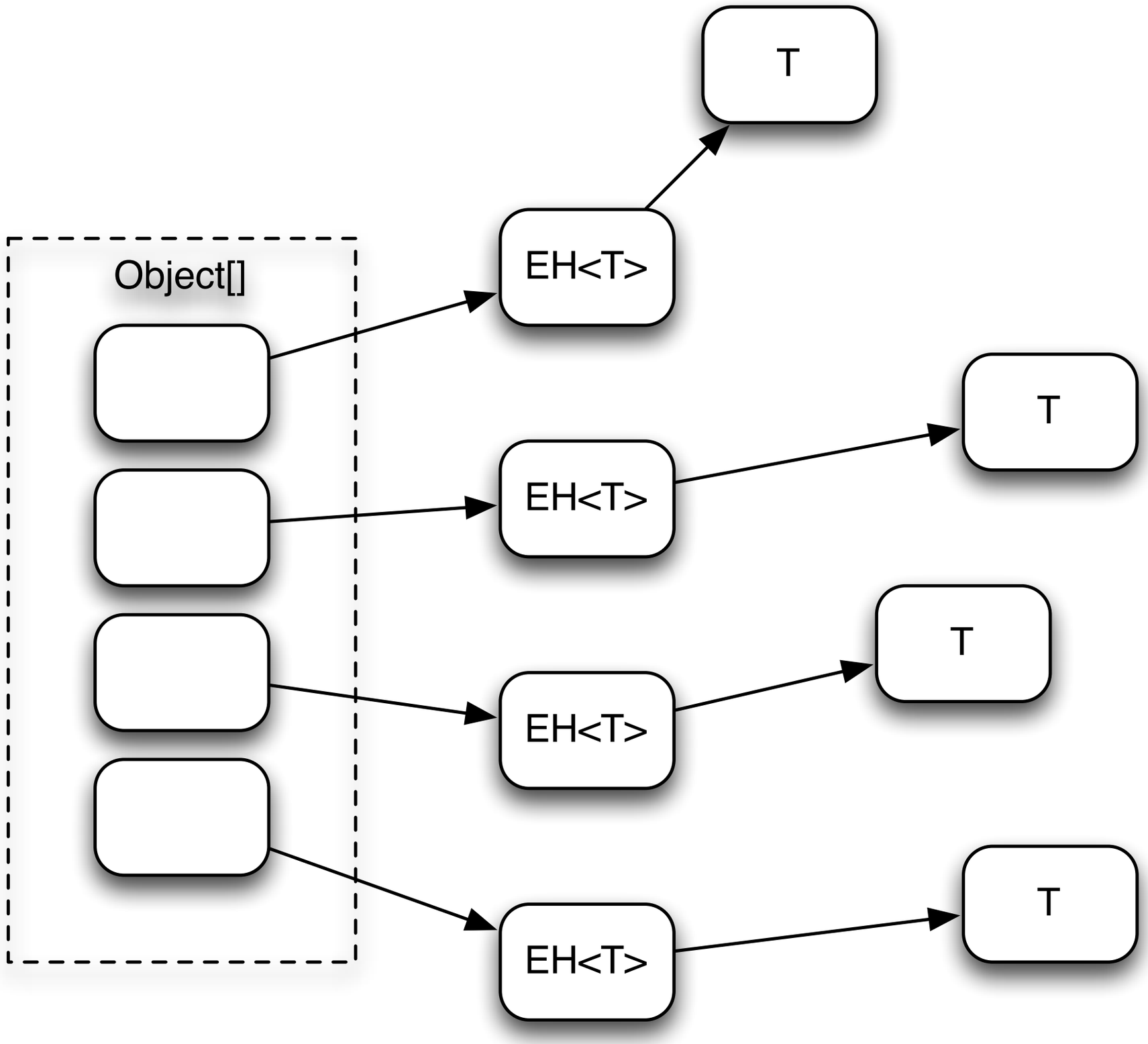

DataProvider
<T>
get(long) : T
BatchEvent
Processor<T>
run() : void
Event
Handler<T>
onEvent(T, long,
boolean) : void
RingBuffer<T>
next() : long
get(long) : T
publish(long) : void
Sequencer
next() : long
publish(long) : void
WaitStrategy
SingleProducer
Sequencer
MultiProducer
Sequencer

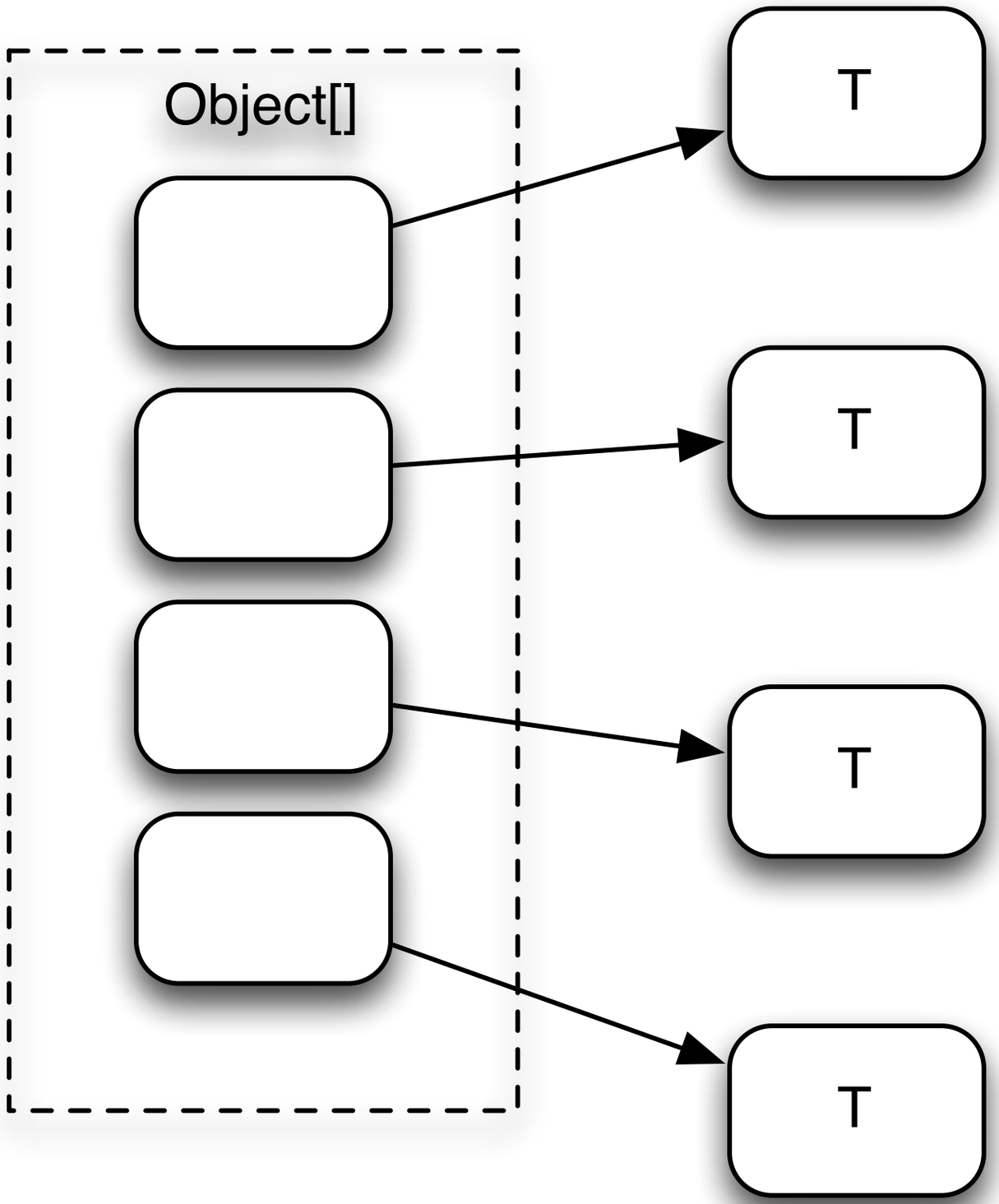
<Immutability>

```
public class SimpleEvent {
    private final long id;
    private final long v1;
    private final long v2;
    private final long v3;

    public SimpleEvent(long id, long v1,
                       long v2, long v3) {
        this.id = id;
        this.v1 = v1;
        this.v2 = v2;
        this.v3 = v3;
    }
}

public class EventHolder {
    public SimpleEvent event;
}
```

```
public class SimpleEvent {
    private final long id;
    private final long v1;
    private final long v2;
    private final long v3;

    public SimpleEvent(long id, long v1,
                       long v2, long v3) {
        this.id = id;
        this.v1 = v1;
        this.v2 = v2;
        this.v3 = v3;
    }
}

public class EventHolder {
    public SimpleEvent event;
}
```

```
EventTranslatorOneArg<EventHolder, SimpleEvent> TRANSLATOR
        = new EventTranslatorOneArg<>() {
    public void translateTo(EventHolder holder,
                            long sequence,
                            SimpleEvent event) {
        holder.event = event;
    }
};
```


Object[]
EH<T>
EH<T>
EH<T>
T
T
T


Object[]
T
T
T
T

```
public interface EventAccessor<T> {
    T take(long sequence);
}

public class CustomRingBuffer<T>
implements DataProvider<EventAccessor<T>>,
           EventAccessor<T> {

    private final Sequencer sequencer;
    private final Object[] buffer;

    public CustomRingBuffer(Sequencer sequencer) {
        this.sequencer = sequencer;
        buffer = new Object[sequencer.getBufferSize()];
    }
```

```
public interface EventAccessor<T> {
    T take(long sequence);
}

public class CustomRingBuffer<T>
implements DataProvider<EventAccessor<T>>,
           EventAccessor<T> {

    private final Sequencer sequencer;
    private final Object[] buffer;

    public CustomRingBuffer(Sequencer sequencer) {
        this.sequencer = sequencer;
        buffer = new Object[sequencer.getBufferSize()];
    }
```

```
public interface EventAccessor<T> {
    T take(long sequence);
}

public class CustomRingBuffer<T>
implements DataProvider<EventAccessor<T>>,
           EventAccessor<T> {

    private final Sequencer sequencer;
    private final Object[] buffer;

    public CustomRingBuffer(Sequencer sequencer) {
        this.sequencer = sequencer;
        buffer = new Object[sequencer.getBufferSize()];
    }
```

```
public interface EventAccessor<T> {
    T take(long sequence);
}

public class CustomRingBuffer<T>
implements DataProvider<EventAccessor<T>>,
           EventAccessor<T> {

    private final Sequencer sequencer;
    private final Object[] buffer;

    public CustomRingBuffer(Sequencer sequencer) {
        this.sequencer = sequencer;
        buffer = new Object[sequencer.getBufferSize()];
    }
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public void put(T t) {
    long next = sequencer.next();
    buffer[index(next)] = t;
    sequencer.publish(next);
}

public T take(long sequence) {
    T t = (T) buffer[index(sequence)];
    buffer[index(sequence)] = null;
    return t;
}

public EventAccessor<T> get(long sequence) {
    return this;
}
```

```
public BatchEventProcessor<EventAccessor<T>>
createHandler(final EventHandler<T> handler) {
    BatchEventProcessor<EventAccessor<T>> processor =
        new BatchEventProcessor<>(this, sequencer.newBarrier(),
            new EventHandler<EventAccessor<T>>() {
                public void onEvent(EventAccessor<T> accessor,
                                    long sequence,
                                    boolean endOfBatch) {
                    handler.onEvent(accessor.take(sequence),
                                    sequence,
                                    endOfBatch);
                }
            });

    sequencer.addGatingSequences(processor.getSequence());

    return processor;
}
```

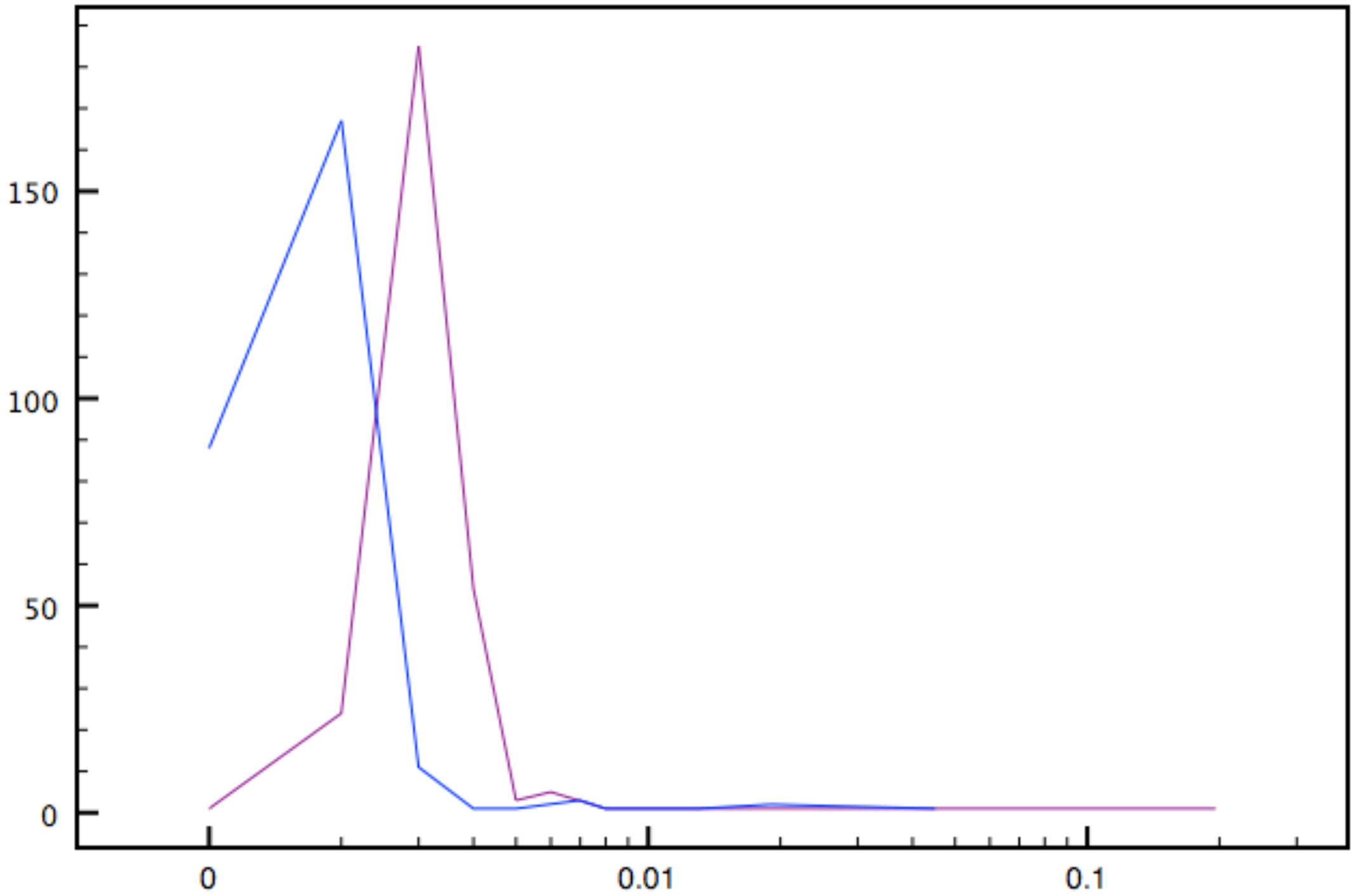
```
public BatchEventProcessor<EventAccessor<T>>
createHandler(final EventHandler<T> handler) {
    BatchEventProcessor<EventAccessor<T>> processor =
        new BatchEventProcessor<>(this, sequencer.newBarrier(),
            new EventHandler<EventAccessor<T>>() {
                public void onEvent(EventAccessor<T> accessor,
                                    long sequence,
                                    boolean endOfBatch) {
                    handler.onEvent(accessor.take(sequence),
                                    sequence,
                                    endOfBatch);
                }
            });

    sequencer.addGatingSequences(processor.getSequence());

    return processor;
}
```

```
public BatchEventProcessor<EventAccessor<T>>
createHandler(final EventHandler<T> handler) {
    BatchEventProcessor<EventAccessor<T>> processor =
        new BatchEventProcessor<>(this, sequencer.newBarrier(),
            new EventHandler<EventAccessor<T>>() {
                public void onEvent(EventAccessor<T> accessor,
                                    long sequence,
                                    boolean endOfBatch) {
                    handler.onEvent(accessor.take(sequence),
                                    sequence,
                                    endOfBatch);
                }
            });

    sequencer.addGatingSequences(processor.getSequence());

    return processor;
}
```

```
public BatchEventProcessor<EventAccessor<T>>
createHandler(final EventHandler<T> handler) {
    BatchEventProcessor<EventAccessor<T>> processor =
        new BatchEventProcessor<>(this, sequencer.newBarrier(),
            new EventHandler<EventAccessor<T>>() {
                public void onEvent(EventAccessor<T> accessor,
                                    long sequence,
                                    boolean endOfBatch) {
                    handler.onEvent(accessor.take(sequence),
                                    sequence,
                                    endOfBatch);
                }
            });

    sequencer.addGatingSequences(processor.getSequence());

    return processor;
}
```

```
public BatchEventProcessor<EventAccessor<T>>
createHandler(final EventHandler<T> handler) {
    BatchEventProcessor<EventAccessor<T>> processor =
        new BatchEventProcessor<>(this, sequencer.newBarrier(),
            new EventHandler<EventAccessor<T>>() {
                public void onEvent(EventAccessor<T> accessor,
                                    long sequence,
                                    boolean endOfBatch) {
                    handler.onEvent(accessor.take(sequence),
                                    sequence,
                                    endOfBatch);
                }
            });

    sequencer.addGatingSequences(processor.getSequence());

    return processor;
}
```

GC Pause Histogram - Nehalem Server



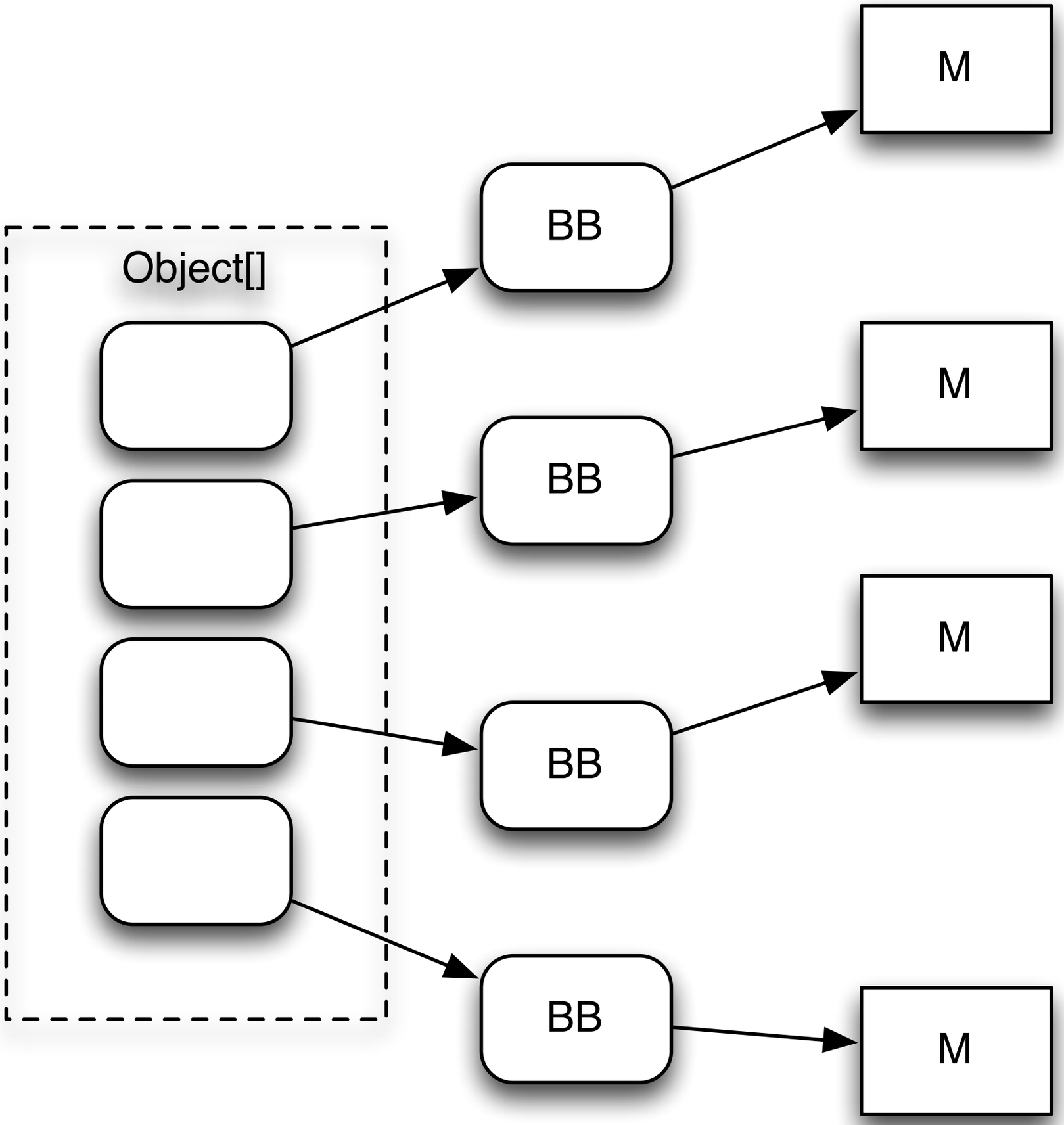
Naive: 11.55s

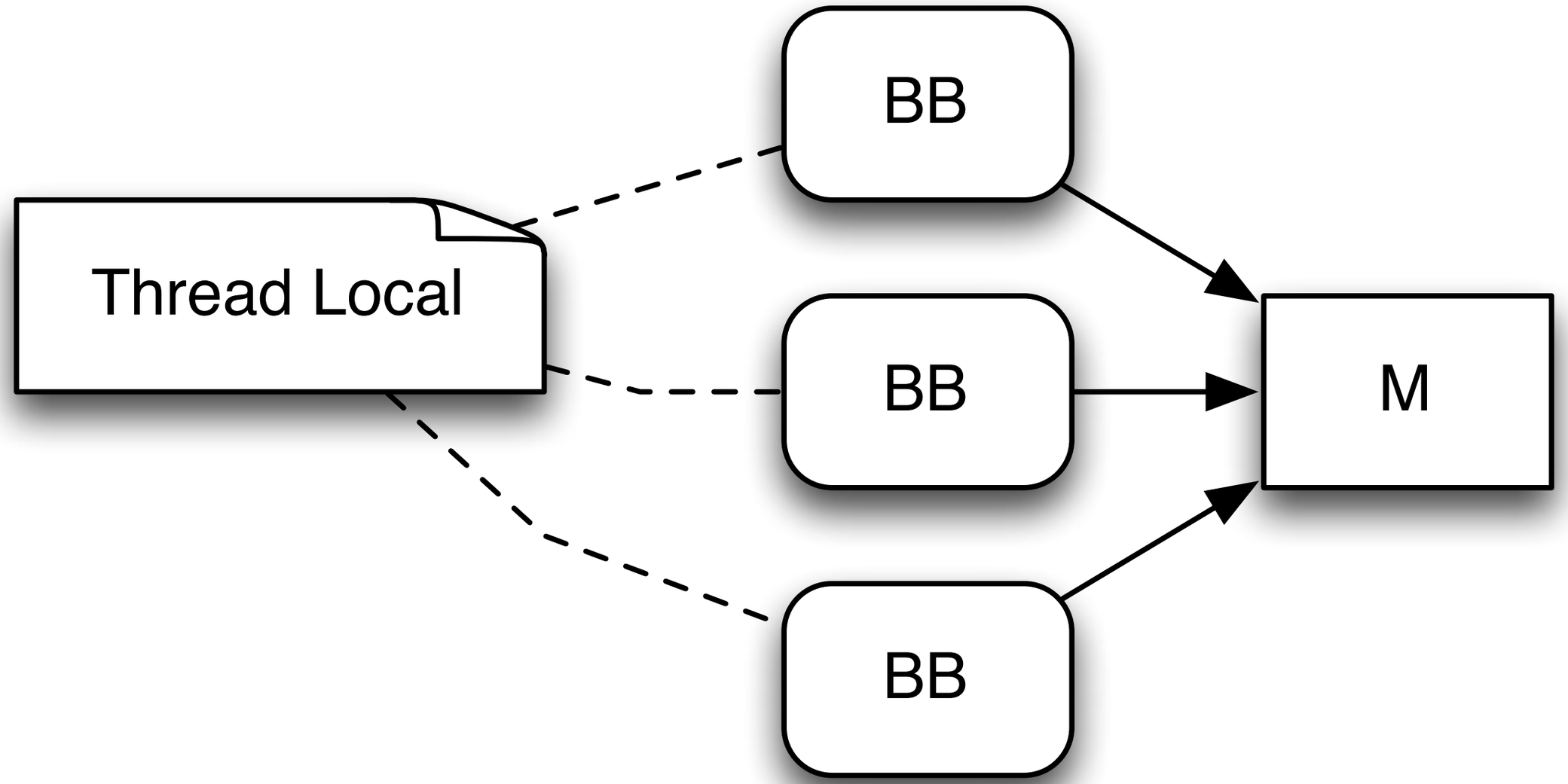
Custom: 6.80s

CPU Cache Stats - Nehalem Server

	Simple	Custom
L1-dcache-loads	13,382,248,612	8,007,239,157
L1-dcache-misses	442,342,087 (3.31%)	533,198,971 (6.66%)
LLC-loads	83,664,103	271,897,732
LLC-misses	**47,647,402 (56.95%)**	**10,928,138 (4.02%)**
dTLB-loads	13,432,146,520	8,002,371,051
dTLB-misses	2,328,319 (0.02%)	1,671,855 (0.02%)

<Serialised>


Object[]
BB
BB
BB
BB
M
M
M
M


BB
Thread Local
BB
M
BB

```
public class OffHeapRingBuffer
implements DataProvider<ByteBuffer> {
    private final Sequencer sequencer;
    private final int entrySize;
    private final ByteBuffer buffer;

    ThreadLocal<ByteBuffer> perThreadBuffer =
            new ThreadLocal<ByteBuffer>() {
                protected ByteBuffer initialValue() {
                    return buffer.duplicate();
                }
            };

public OffHeapRingBuffer(Sequencer sequencer,
                         int entrySize) {
    this.sequencer = sequencer;
    this.entrySize = entrySize;
    buffer = ByteBuffer.allocateDirect(
        sequencer.getBufferSize() * entrySize);
}
```

```
public class OffHeapRingBuffer
implements DataProvider<ByteBuffer> {
    private final Sequencer sequencer;
    private final int entrySize;
    private final ByteBuffer buffer;

    ThreadLocal<ByteBuffer> perThreadBuffer =
            new ThreadLocal<ByteBuffer>() {
                protected ByteBuffer initialValue() {
                    return buffer.duplicate();
                }
            };

public OffHeapRingBuffer(Sequencer sequencer,
                         int entrySize) {
    this.sequencer = sequencer;
    this.entrySize = entrySize;
    buffer = ByteBuffer.allocateDirect(
        sequencer.getBufferSize() * entrySize);
}
```

```
public class OffHeapRingBuffer
implements DataProvider<ByteBuffer> {
    private final Sequencer sequencer;
    private final int entrySize;
    private final ByteBuffer buffer;

    ThreadLocal<ByteBuffer> perThreadBuffer =
            new ThreadLocal<ByteBuffer>() {
                protected ByteBuffer initialValue() {
                    return buffer.duplicate();
                }
            };

    public OffHeapRingBuffer(Sequencer sequencer,
                             int entrySize) {
        this.sequencer = sequencer;
        this.entrySize = entrySize;
        buffer = ByteBuffer.allocateDirect(
            sequencer.getBufferSize() * entrySize);
    }
```

```
public class OffHeapRingBuffer
implements DataProvider<ByteBuffer> {
    private final Sequencer sequencer;
    private final int entrySize;
    private final ByteBuffer buffer;

    ThreadLocal<ByteBuffer> perThreadBuffer =
            new ThreadLocal<ByteBuffer>() {
                protected ByteBuffer initialValue() {
                    return buffer.duplicate();
                }
            };

    public OffHeapRingBuffer(Sequencer sequencer,
                             int entrySize) {
        this.sequencer = sequencer;
        this.entrySize = entrySize;
        buffer = ByteBuffer.allocateDirect(
            sequencer.getBufferSize() * entrySize);
    }
```

```
public class OffHeapRingBuffer
implements DataProvider<ByteBuffer> {
    private final Sequencer sequencer;
    private final int entrySize;
    private final ByteBuffer buffer;

    ThreadLocal<ByteBuffer> perThreadBuffer =
            new ThreadLocal<ByteBuffer>() {
                protected ByteBuffer initialValue() {
                    return buffer.duplicate();
                }
            };

    public OffHeapRingBuffer(Sequencer sequencer,
                             int entrySize) {
        this.sequencer = sequencer;
        this.entrySize = entrySize;
        buffer = ByteBuffer.allocateDirect(
            sequencer.getBufferSize() * entrySize);
}
```

```
public ByteBuffer get(long sequence) {
    int index = index(sequence);
    int position = index * entrySize;
    int limit = position + entrySize;

    ByteBuffer byteBuffer = perThreadBuffer.get();
    byteBuffer.position(position).limit(limit);

    return byteBuffer;
}
```

```
public ByteBuffer get(long sequence) {
    int index = index(sequence);
    int position = index * entrySize;
    int limit = position + entrySize;

    ByteBuffer byteBuffer = perThreadBuffer.get();
    byteBuffer.position(position).limit(limit);

    return byteBuffer;
}
```

```
public ByteBuffer get(long sequence) {
    int index = index(sequence);
    int position = index * entrySize;
    int limit = position + entrySize;

    ByteBuffer byteBuffer = perThreadBuffer.get();
    byteBuffer.position(position).limit(limit);

    return byteBuffer;
}
```

```
public ByteBuffer get(long sequence) {
    int index = index(sequence);
    int position = index * entrySize;
    int limit = position + entrySize;

    ByteBuffer byteBuffer = perThreadBuffer.get();
    byteBuffer.position(position).limit(limit);

    return byteBuffer;
}
```

```
public ByteBuffer get(long sequence) {
    int index = index(sequence);
    int position = index * entrySize;
    int limit = position + entrySize;

    ByteBuffer byteBuffer = perThreadBuffer.get();
    byteBuffer.position(position).limit(limit);

    return byteBuffer;
}
```

```
public void put(byte[] data) {
    long next = sequencer.next();
    try {
        get(next).put(data);
    } finally {
        sequencer.publish(next);
    }
}
```

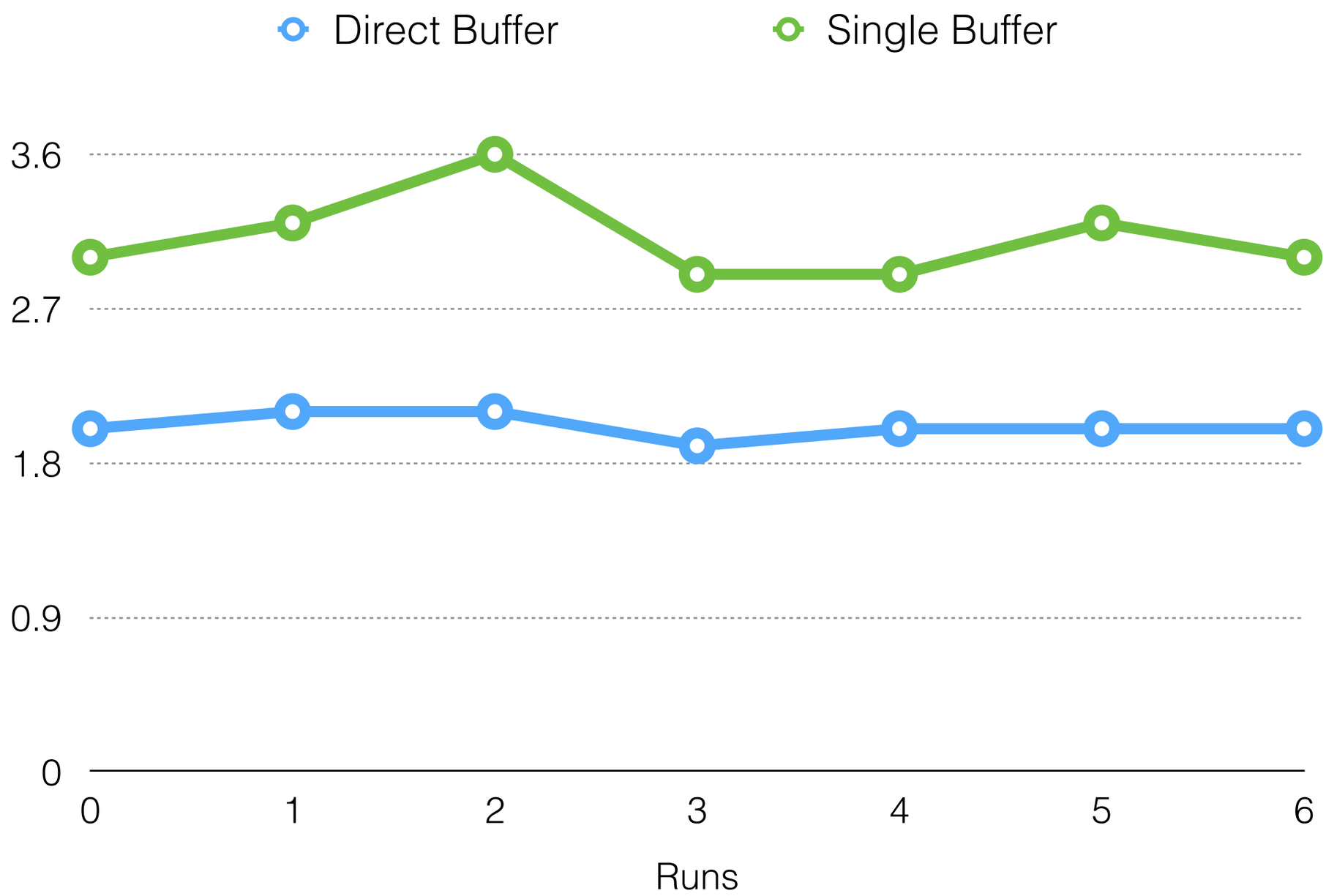
```
public void put(byte[] data) {
    long next = sequencer.next();
    try {
        get(next).put(data);
    } finally {
        sequencer.publish(next);
    }
}
```

```
public void put(byte[] data) {
    long next = sequencer.next();
    try {
        get(next).put(data);
    } finally {
        sequencer.publish(next);
    }
}
```

```
public void put(byte[] data) {
    long next = sequencer.next();
    try {
        get(next).put(data);
    } finally {
        sequencer.publish(next);
    }
}
```

```
public class BufferEventHandler
implements EventHandler<ByteBuffer> {

    public void onEvent(ByteBuffer buffer,
                        long sequence,
                        boolean endOfBatch) {
        // Do stuff...
    }
}
```

Throughput (MOps/sec) - Nehalem Server



CPU Cache Stats - Nehalem Server

	Direct Buffer	Single Buffer
L1-dcache-loads	52,194,082,109	35,236,312,790
L1-dcache-misses	1,251,459,098 (2.40%)	599,414,551 (1.70%)
LLC-loads	**520,972,754**	**23,697,987**
LLC-misses	479,054,619 (91.95%)	17,083,178 (72.09%)
dTLB-loads	52,206,185,418	35,230,048,700
dTLB-misses	137,088,875 (0.26%)	922,520 (0.00%)

In Conclusion…

<Q&A>

- http://lmax-exchange.github.io/disruptor/
- We're Hiring: recruitment@lmax.com