Microservices for Performance

Building low latency Micro Services and monolith in Java using high performance serialization and messaging

Peter Lawrey - CEO of Higher Frequency Trading

GOTO Chicago - 2016

Chronicle

Peter Lawrey

Java Developer/Consultant for investment banks and hedge funds for 8 years, 23 years in IT.

Most answers for Java and JVM on stackoverflow.com

Founder of the Performance Java User's Group.

Architect of Chronicle Software

Java Champion

- memory
- file-io
- concurrency
- jvm
- string
- arrays
- performance
- multithreading
- java

Chronicle Software

Help companies migrate to high performance Java code.

Sponsor open source projects https://github.com/OpenHFT

Licensed solutions Chronicle-FIX, Chronicle-Enterprise and Chronicle-Queue-Enterprise

Offer one week proof of concept workshops, advanced Java training, consulting and bespoke development.

My First Computer

```
                                     CAP ALT GR1 GR2   sp:0    Sun  01/31/  01:20
MS-DOS V2.11 (UOP)
ASSIGN   SYS       50   1-31-84  12:35p
DDAPCNEW SYS    11648   1-30-84   4:25p
DDNECP2  SYS    14208   1-09-85   4:12p
DDLA50   SYS    14848   9-08-83  10:27a
DDLA100  SYS    14848   9-08-83  10:29a
DDOKI84  SYS    14848   9-08-83  11:18a
DDOKI84  SYS    14848   9-08-83  11:20a
DDPMUP   SYS    14848   9-08-83  10:08a
DDANADVM SYS    14848   9-08-83  10:19a
DDCITOLR SYS    14976   9-08-83  10:20a
DDCNTXM  SYS    14848   9-08-83  10:21a
DDDS100  SYS    14848   9-08-83  10:25a
DDFXHR8  SYS    14976   9-08-83  10:23a
DDFXLR7  SYS    14976   9-08-83  10:24a
DDFXLR8  SYS    14976   1-01-80  12:15p
DDHP7470 SYS    12416   9-08-83  10:26a
DDIDSM   SYS    14848   9-08-83  10:26a
DDNECAPC SYS     7168  12-21-83   3:08p
CRTDUMP  COM    10881   1-31-84   9:36a
READ     ME     12928   2-01-84   3:01p
S        BAT       86   6-27-83
CONFIG   SYS       10   6-27-83
AUTOEXEC BAT        1   5-09-86  12:40p
       70 File(s)   441344 bytes free

A>
```

128 KB of RAM

Where do Microservices come from?

Microservices builds on design principles which have been around for some time.

- UNIX Principle.
- Staged Event Driven Architecture.
- Service Orientated Architecture.
- Lambda Architecture.
- Reactive Streams.
- Used in building Web applications. “Micro-Web-Services”

Performance for GUI applications

A key performance threshold for GUI applications is the speed a human can see. Anything faster than this doesn't matter for a GUI.

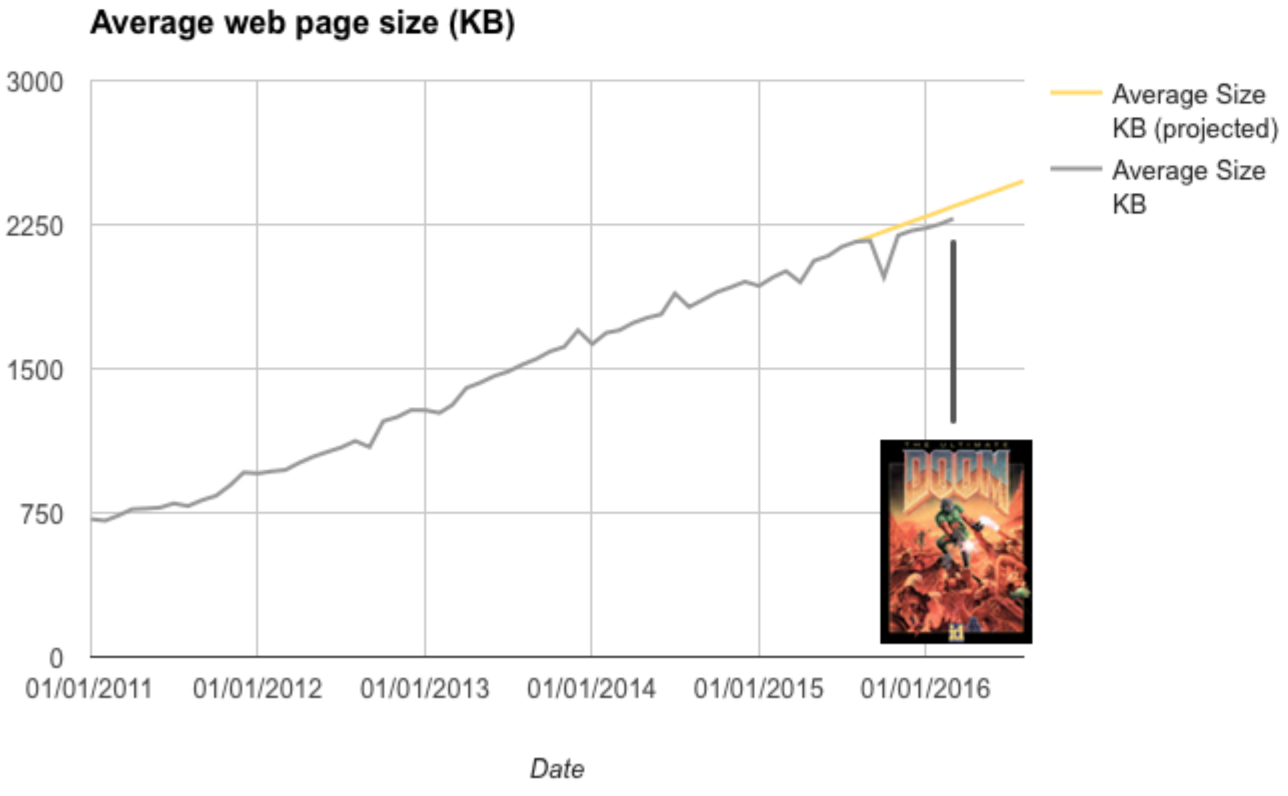
In cinemas, the frame rate used to be 24 frames per second.

This means anything shorter than 40 ms is unlikely to be important.

Chronicle

Computer programs vs Hardware.

GUI programs have tended to not get any faster, rather they attempt to give a richer experience.



goto; conference

Microservices denial?

Microservices bring together best practices from a variety of areas. Most likely you are already using some of these best practices.

Reactions to Microservices

- It sounds like marketing hype.
- It all sounds pretty familiar.
- It just a rebranding of stuff we already do.
- There is room for improvement in what we do.
- There are some tools, and ideas we could apply to our systems without changing too much.

Chronicle

Microservices score card

	Today	Quick Wins	6 Months
Simple component based design.	★★	★★☆	★★☆
Distributed by JVM and Machine	★★	★★	★★☆
Service Discovery	★	★☆	★★
Resilience to failure	★☆	★☆	★★
Transport agnostic	★	★☆	★★
Asynchronous messaging.	★☆	★★	★★
Automated, dynamic deployment of services.	★☆	★★	★★☆
Service private data sets.	☆	★☆	★★
Transparent messaging.	☆	★★	★★☆
Independent Teams	★☆	★★	★★
Lambda Architecture	★	★★	★★★

Benefit of Microservices in Trading Systems

Standard techniques for developing and deploying distributed systems

- Shorter time to market.
- Easier to maintain.
- Simpler programming models.

What Microservices can learn from Trading Systems

Trading system have been working with performant distributed systems for years.

- Asynchronous messaging, how to test correctness and performance for latencies you cannot see.
- Building deterministic, highly reproducible systems.

What is low latency?

The term “low latency” can applied to a wide range of situations.

A broad definition might be;

Low latency means you have a view on how much the response time of a system costs your business.

In this talk I will assume;

Low latency means you care about latencies you can only measure as even the worst latencies are too fast to see.

Example of low latency?

An Investment Bank measured the 99.999%ile (worst 1 in 100,000) latency of our Chronicle FIX engine at 450 micro-seconds. Chronicle FIX is written in Java.

This was unacceptable to them. We fixed this bug and dropped it to below 35 micro-seconds.

This was after a socket read to after decoding the message to their data model and persisting it.

To Go Faster Do Less Work

Micro-services design encourages a design where each service does something simple and it does it well.

Your L1/L2 caches are a precious resource. They are limited of only 32 KB (instruction), 32 KB (data) and 256 KB (L2). If you exceed this you are hitting the shared L3 cache which is a scalability problem AND every access is 10x slower or more.

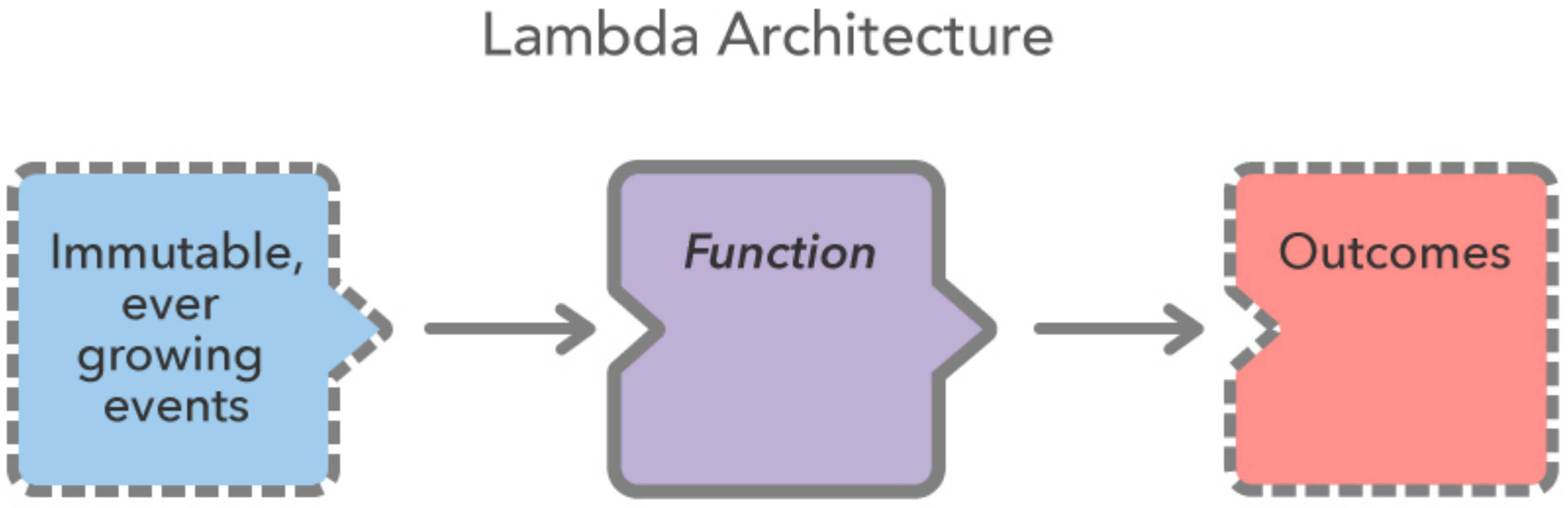
If you want to scale across cores, stay in your L1/L2 cache you want each thread to perform a simple task which fits inside its caches as much as possible.

Where do they overlap.

Microservices and Trading Systems have high level principles of

- Simple component based design.
- Asynchronous messaging.
- Automated, dynamic deployment of services.
- Service private data sets.
- Transparent messaging.
- Teams can develop independently based on well defined contracts.

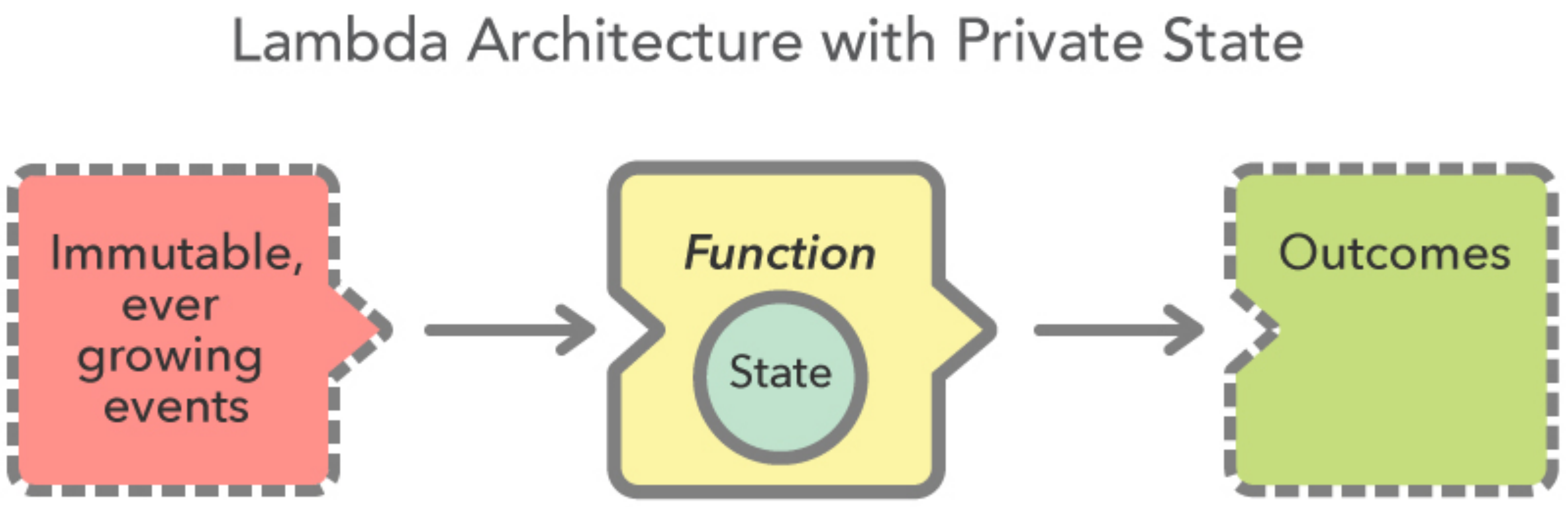
Chronicle



Each output is the result of one input message. This is useful for gateways, both in and out of your system and highly concurrent.

goto; conference

Chronicle


Lambda Architecture with Private State
Immutable, ever growing events
Function
State
Outcomes

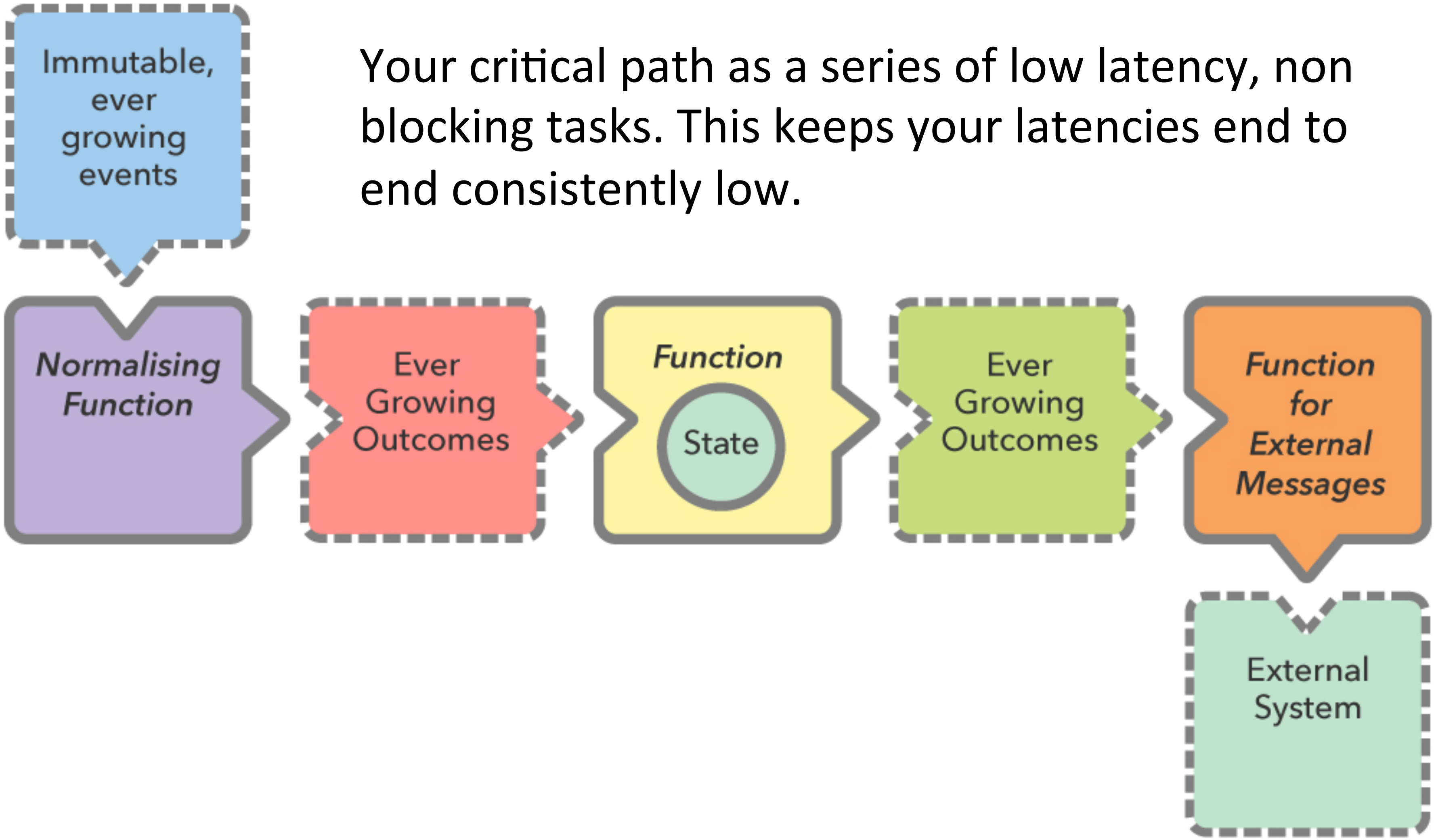
Each output is the result of ALL the inputs. Instead of replying ALL input message each time, the Function could save an accumulated state (which can be recreated by replaying inputs)

goto; conference

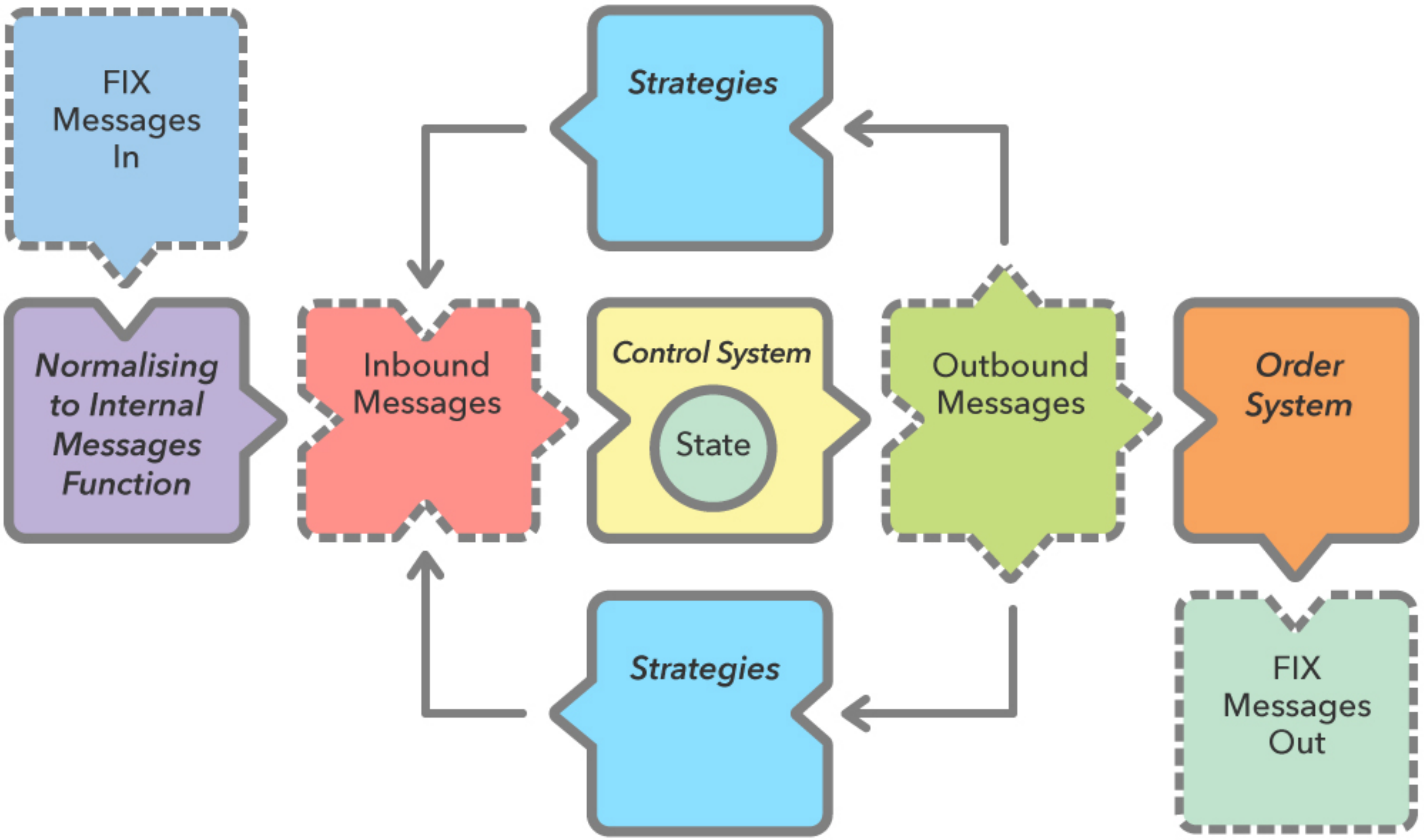
Chronicle

Lambda Architecture Services Chained

Your critical path as a series of low latency, non blocking tasks. This keeps your latencies end to end consistently low.



Lambda Architecture Services with Feedback



What do we mean by a Distributed System.

Usually a distributed system will have a processes run across multiple machines.

In a high performance space you want to thinking about how your threads are distributed within your machine.

In particular, NUMA regions can be critical especially for the JVM which has a GC which assumes random access to all memory.

Even if you are working within a single NUMA region, it can be worth looking at the distribution of your application within a region.

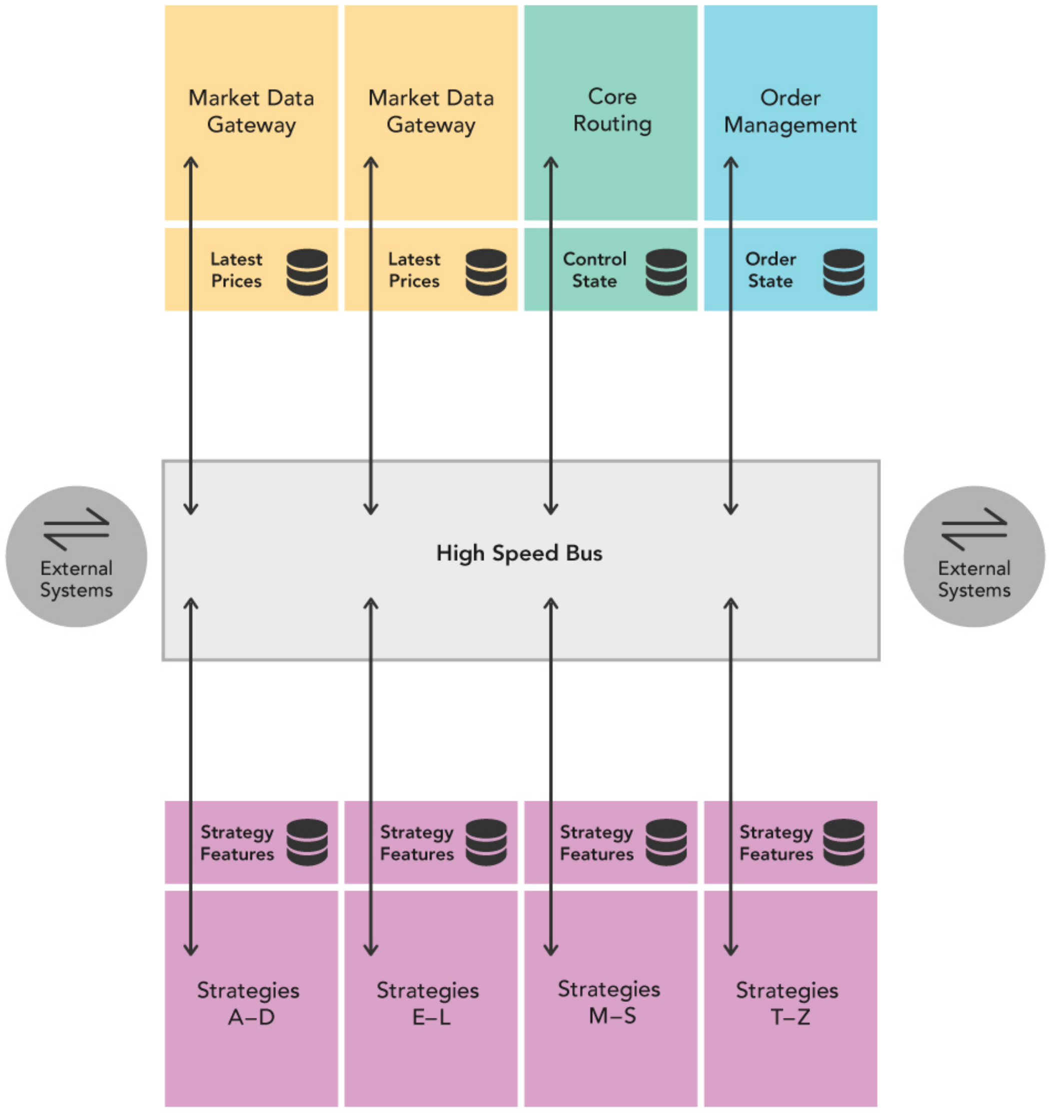
A Computer is a Distributed System.

When you are considering short time scales of 10 micro-seconds or less, you have to consider that each core as a processor of it's own.

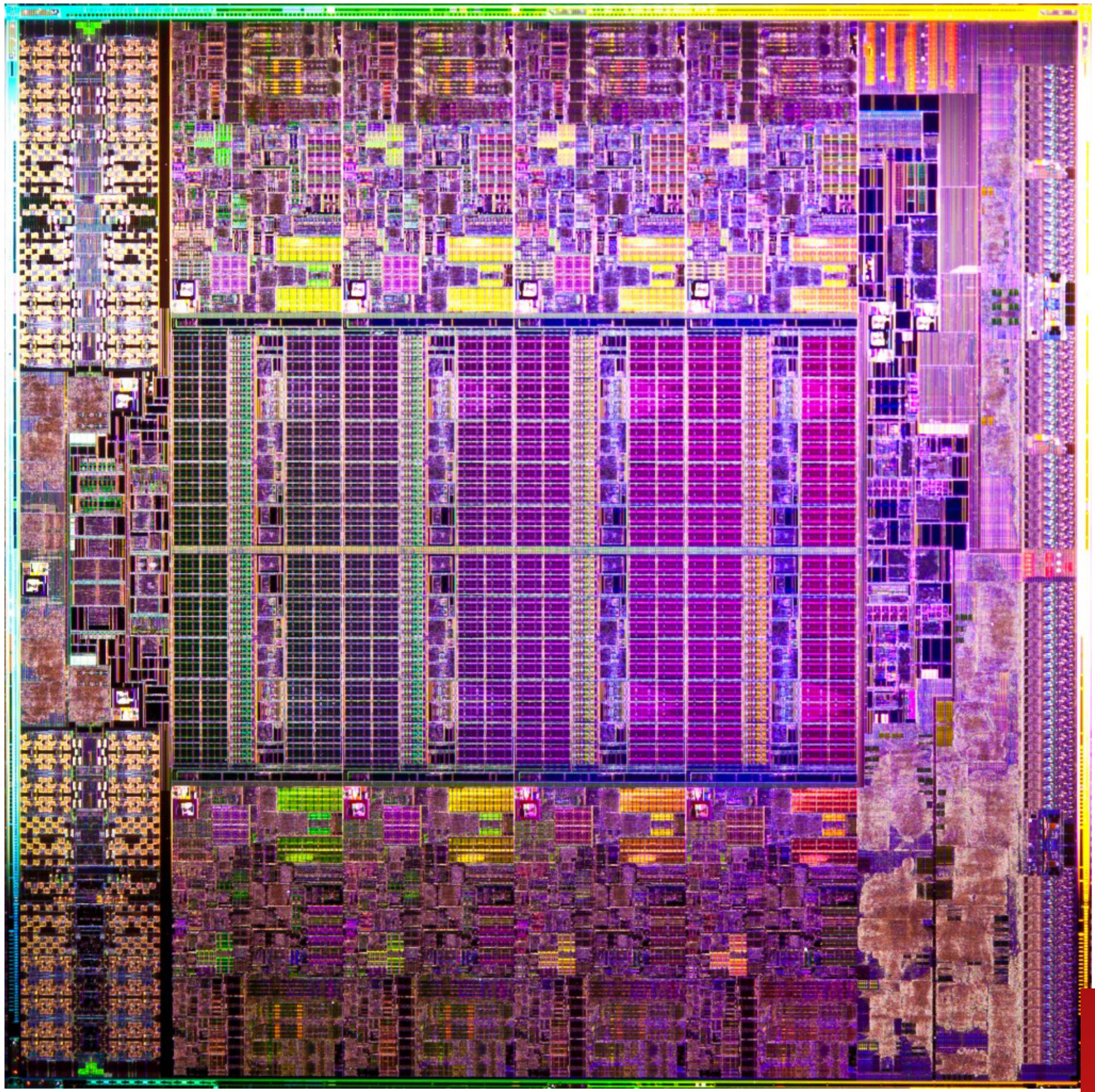
Each core

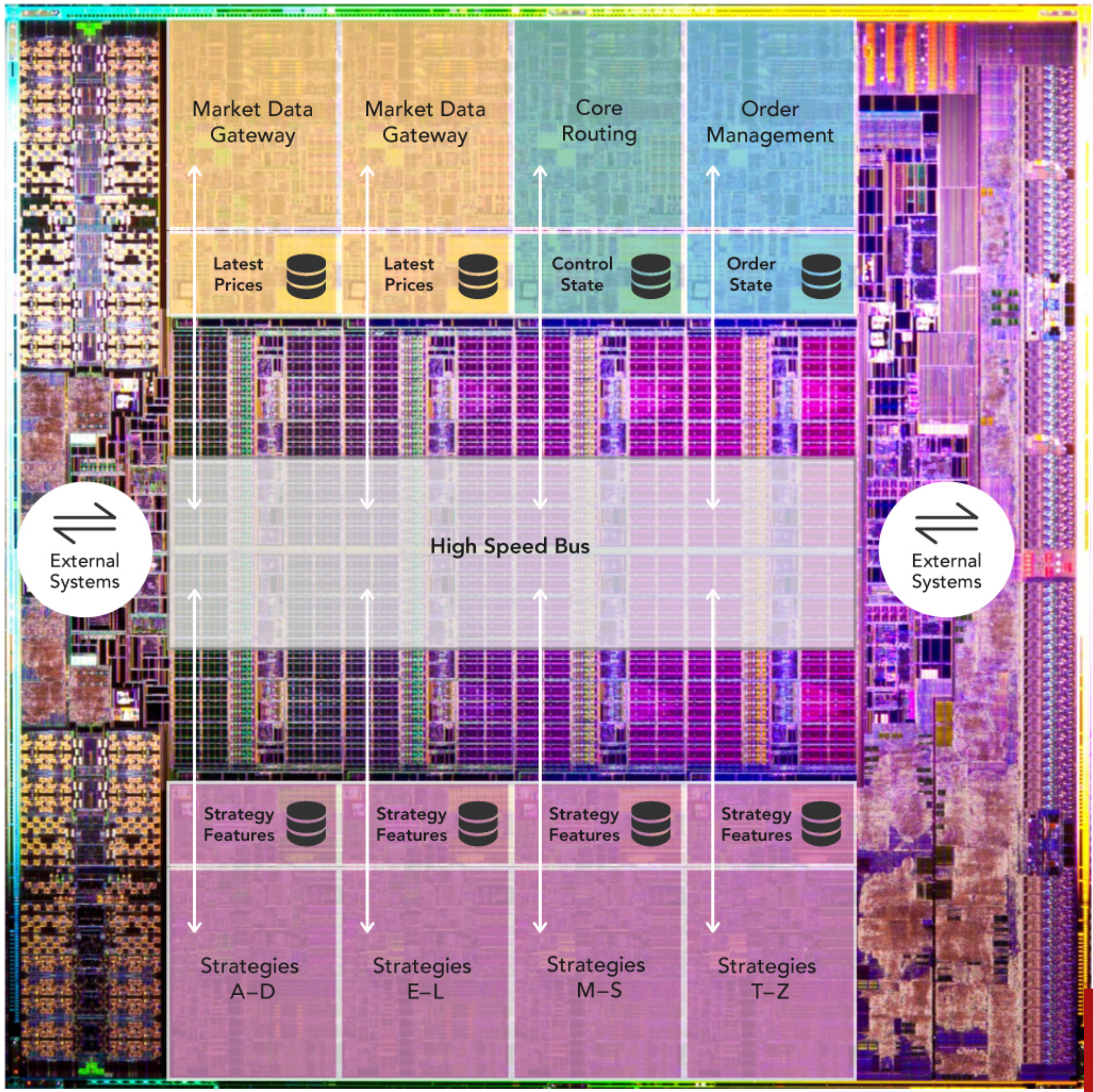
- has it's own memory (L1 & L2 caches)
- can run independently
- communicates with other cores via a L2 cache coherence bus.

Chronicle



goto; conference




Market Data Gateway
Market Data Gateway
Core Routing
Order Management
Latest Prices
Latest Prices
Control State
Order State
External Systems
High Speed Bus
External Systems
Strategy Features
Strategy Features
Strategy Features
Strategy Features
Strategies A–D
Strategies E–L
Strategies M–S
Strategies T–Z

Designing for Micro-services

Micro-services need to have;

- Isolation, minimises contention of state.
- Asynchronous messages, minimises the impact of delays.
- One thing and do it well, stay in your private CPU caches.
- Services should be addressable.
- Transparency of what your services are doing. *

Micro-services come in systems must be testable in isolation.

* Transparency is needed if you are to remove redundant work.

Testing and Debugging Micro-services

You want micro-services which are easy to unit test and debug. However, your framework and infrastructure can get in the way. They are not helping.

You need to be able to run your services as stand alone components. These components can be tested, integrated and debugged without the framework or infrastructure so you can see where the source of your issues are.

Why not use a framework?

In this example we look at how to implement these services without a framework. Frameworks are very good for getting started but are not so good if the framework doesn't do exactly what you want. In particular they are not very good at doing less.

In performant systems, how easy it is to remove something the program doesn't have to be doing is just as important as how easy it is to add some functionality.

The key is transparency in what your services are doing.

Turning a Monolith into Micro-Services

You need to have good component based design whether you have Micro-Services or a Monolith with clear separation of concerns. Micro-services make good component design even more important as they won't well work unless you do this.

To turn your components into services, you need to add a transport. This transport should be interchangeable in fact it should be optional and have no impact if it's not there.

Component + Transport = Service.

Let’s look at an example

Say we have a market data component which combines prices and this feeds another component which is your order manager.

A more details discussion of this example is on my blog
https://vanilla-java.github.io/

The full code is https://github.com/Vanilla-Java/Microservices/

Starting with a simple contract

A simple contract for a service which takes asynchronous messages is an interface. Each message has a method name and it takes one or more arguments.

Say we have a component which consumes one sided prices

```
public interface SidedMarketDataListener {
    void onSidedPrice(SidedPrice sidedPrice);
}
```

And this produces top of book prices with bid and ask

```
public interface MarketDataListener {
    void onTopOfBookPrice(TopOfBookPrice price);
}
```

A Data Transfer Object

AbstractMarshallable provides an equals, hashCode and toString.

```
public class SidedPrice extends AbstractMarshallable {
    String symbol;
    long timestamp;
    Side side;
    double price, quantity;

    public SidedPrice(String symbol, long timestamp, Side side, double price, double quantity) {
        init(symbol, timestamp, side, price, quantity);
    }

    public SidedPrice init(String symbol, long timestamp, Side side, double price, double quantity) {
        this.symbol = symbol;
        this.timestamp = timestamp;
        this.side = side;
        this.price = price;
        this.quantity = quantity;
        return this;
    }
}
```

Deserializable toString()

For it to deserialize the same object, no information can be lost, which useful to creating test objects from production logs.

```
SidedPrice sp = new SidedPrice("Symbol", 123456789000L,
                                        Side.Buy, 1.2345, 1_000_000);
assertEquals("!SidedPrice {\n" +
        "  symbol: Symbol,\n" +
        "  timestamp: 123456789000,\n" +
        "  side: Buy,\n" +
        "  price: 1.2345,\n" +
        "  quantity: 1000000.0\n" +
        "}\n", sp.toString());

// from string
SidedPrice sp2 = Marshallable.fromString(sp.toString());
assertEquals(sp2, sp);
assertEquals(sp2.hashCode(), sp.hashCode());
```

Writing a simple component

We have a component which implements our contract and in turn calls another interface with the result (if there is one)

```
public class SidedMarketDataCombiner implements SidedMarketDataListener {
    final MarketDataListener mdListener;
    final Map<String, TopOfBookPrice> priceMap = new TreeMap<>();

    public SidedMarketDataCombiner(MarketDataListener mdListener) {
        this.mdListener = mdListener;
    }

    public void onSidedPrice(SidedPrice sidedPrice) {
        TopOfBookPrice price = priceMap.computeIfAbsent(sidedPrice.symbol,
TopOfBookPrice::new);
        if (price.combine(sidedPrice))
            mdListener.onTopOfBookPrice(price);
    }
}
```

Mocking our simple component

We can use standard mocking tools such as EasyMock and I easy test this from my IDE.

```
@Test
public void testOnSidedPrice() {
    // what we expect to happen
    SidedPrice sp = new SidedPrice("Symbol", 123456789000L,
                                                      Side.Buy, 1.2345, 1_000_000);
    SidedMarketDataListener listener = createMock(SidedMarketDataListener.class);
    listener.onSidedPrice(sp);
    replay(listener);

    // what happens
    listener.onSidedPrice(sp);

    // verify we got everything we expected.
    verify(listener);
}
```

Testing our simple component

We can mock the output listener of our component.

```
MarketDataListener listener = createMock(MarketDataListener.class);
listener.onTopOfBookPrice(new TopOfBookPrice("EURUSD", 123456789000L,
                                      1.1167, 1_000_000, Double.NaN, 0));
listener.onTopOfBookPrice(new TopOfBookPrice("EURUSD", 123456789100L,
                                      1.1167, 1_000_000, 1.1172, 2_000_000));
replay(listener);

SidedMarketDataListener combiner = new SidedMarketDataCombiner(listener);
combiner.onSidedPrice(new SidedPrice("EURUSD", 123456789000L,
                                      Side.Buy, 1.1167, 1e6));
combiner.onSidedPrice(new SidedPrice("EURUSD", 123456789100L,
                                      Side.Sell, 1.1172, 2e6));

verify(listener);
```

Testing multiple components

```
// what we expect to happen
OrderListener listener = createMock(OrderListener.class);
listener.onOrder(new Order("EURUSD", Side.Buy, 1.1167, 1_000_000));
replay(listener);

// build our scenario
OrderManager orderManager = new OrderManager(listener);
SidedMarketDataCombiner combiner = new SidedMarketDataCombiner(orderManager);

// events in
orderManager.onOrderIdea(new OrderIdea("EURUSD", Side.Buy, 1.1180, 2e6)); // not expected to

combiner.onSidedPrice(new SidedPrice("EURUSD", 123456789000L, Side.Sell, 1.1172, 2e6));
combiner.onSidedPrice(new SidedPrice("EURUSD", 123456789100L, Side.Buy, 1.1160, 2e6));

combiner.onSidedPrice(new SidedPrice("EURUSD", 123456789100L, Side.Buy, 1.1167, 2e6));

orderManager.onOrderIdea(new OrderIdea("EURUSD", Side.Buy, 1.1165, 1e6)); // expected to trig

verify(listener);
```

Adding a transport

Any messaging system can be used as a transport. You can use

- REST or HTTP
- JMS or Akka
- Aeron or a UDP based transport.
- Raw TCP or UDP.
- Chronicle Queue.

Why use Chronicle Queue

Chronicle Queue v4 has a number of advantages

- Broker less, only the OS needs to be up.
- Low latency, less than 10 microseconds 99% of the time.
- Persisted, giving your replay and transparency.
- Can replace your logging improving performance.
- Kernel Bypass, Shared across JVMs with a system call for each message.

What does Chronicle Queue look like?

```
--- !!meta-data #binary
header: !SCQStore { wireType: !WireType BINARY, writePosition: 777, roll: !SCQSRoll
{ length: 86400000, format: yyyyMMdd, epoch: 0 }, indexing: !SCQSIndexing
{ indexCount: !int 8192, indexSpacing: 64, index2Index: 0, lastIndex: 0 } }
# position: 227
--- !!data #binary
onOrderIdea: { symbol: EURUSD, side: Buy, limitPrice: 1.118, quantity: 2000000.0 }
# position: 306
--- !!data #binary
onTopOfBookPrice: { symbol: EURUSD, timestamp: 123456789000, buyPrice: NaN, buyQuantity:
0, sellPrice: 1.1172, sellQuantity: 2000000.0 }
# position: 434
--- !!data #binary
onTopOfBookPrice: { symbol: EURUSD, timestamp: 123456789100, buyPrice: 1.116,
buyQuantity: 2000000.0, sellPrice: 1.1172, sellQuantity: 2000000.0 }
# position: 566
--- !!data #binary
onTopOfBookPrice: { symbol: EURUSD, timestamp: 123456789100, buyPrice: 1.1167,
buyQuantity: 2000000.0, sellPrice: 1.1172, sellQuantity: 2000000.0 }
# position: 698
--- !!data #binary
onOrderIdea: { symbol: EURUSD, side: Buy, limitPrice: 1.1165, quantity: 1000000.0 }
...
# 83885299 bytes remaining
```

Measuring the performance?

Measure the write latency with JMH (Java Microbenchmark Harness)

```
Percentiles, us/op:
     p(0.0000) =      2.552 us/op
    p(50.0000) =      2.796 us/op
    p(90.0000) =      5.600 us/op
    p(95.0000) =      5.720 us/op
    p(99.0000) =      8.496 us/op
    p(99.9000) =     15.232 us/op
    p(99.9900) =     19.977 us/op
    p(99.9990) =    422.475 us/op
    p(99.9999) =    438.784 us/op
   p(100.0000) =    438.784 us/op
```

Java Latency Benchmark Harness

JMH is an excellent tool, why add another?

JLBH measures

- Timings in asynchronous threads.
- Timings from any point in the process to another.
- Works with a set throughput.
- Accounts for coordinated omission.
- Has a background thread to report the general noise.

Java Latency Benchmark Harness

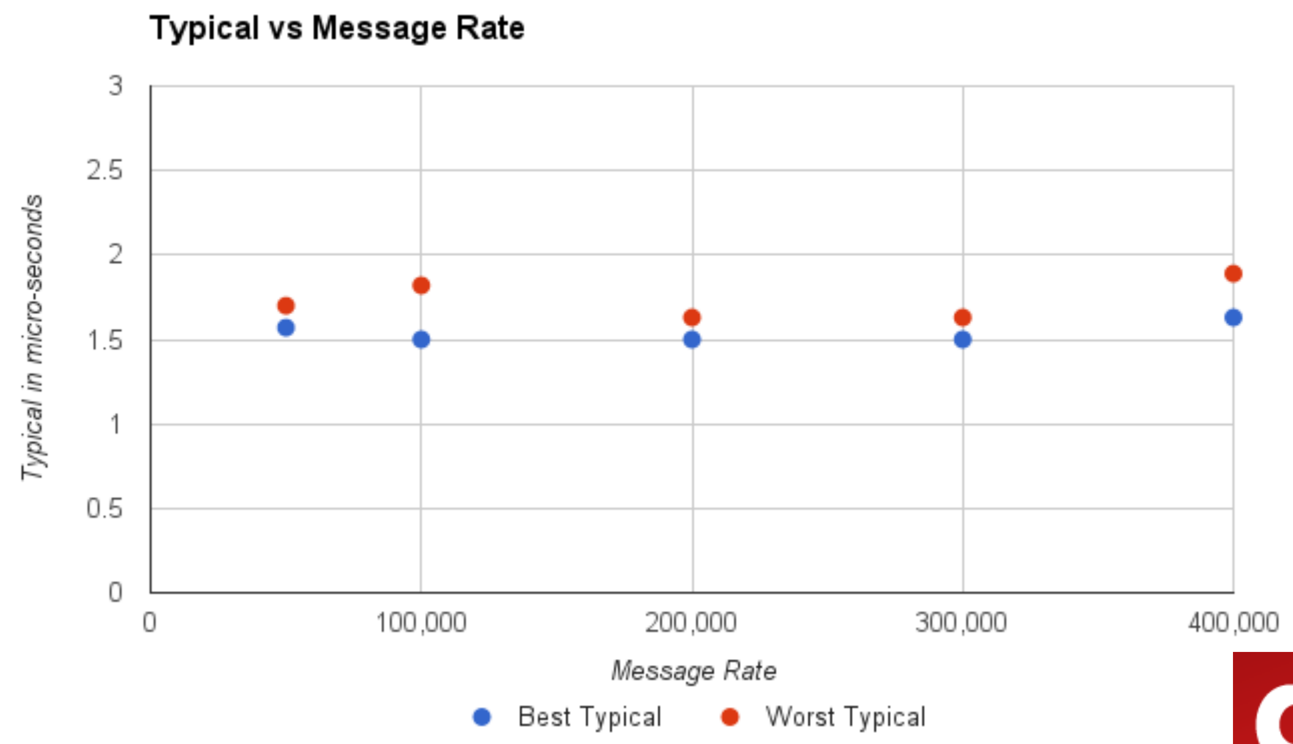
Multiple services running in their own threads can be chained and performance tested individually as well as end to end.

```
@Override
public void init(JLBH jlbh) {
    serviceIn = SingleChronicleQueueBuilder.binary(queueIn).build()
                        .createAppender().methodWriter(Service.class);

    service2 = new ServiceWrapper<>(queueIn, queue2,
                        new ServiceImpl(jlbh.addProbe("Service 2")));

    service3 = new ServiceWrapper<>(queue2, queue3,
                        new ServiceImpl(jlbh.addProbe("Service 3")));

    serviceOut = new ServiceWrapper<>(queue3, queueOut,
                        new ServiceImpl(jlbh.addProbe("Service Out"), jlbh));
}
```

Typical performance

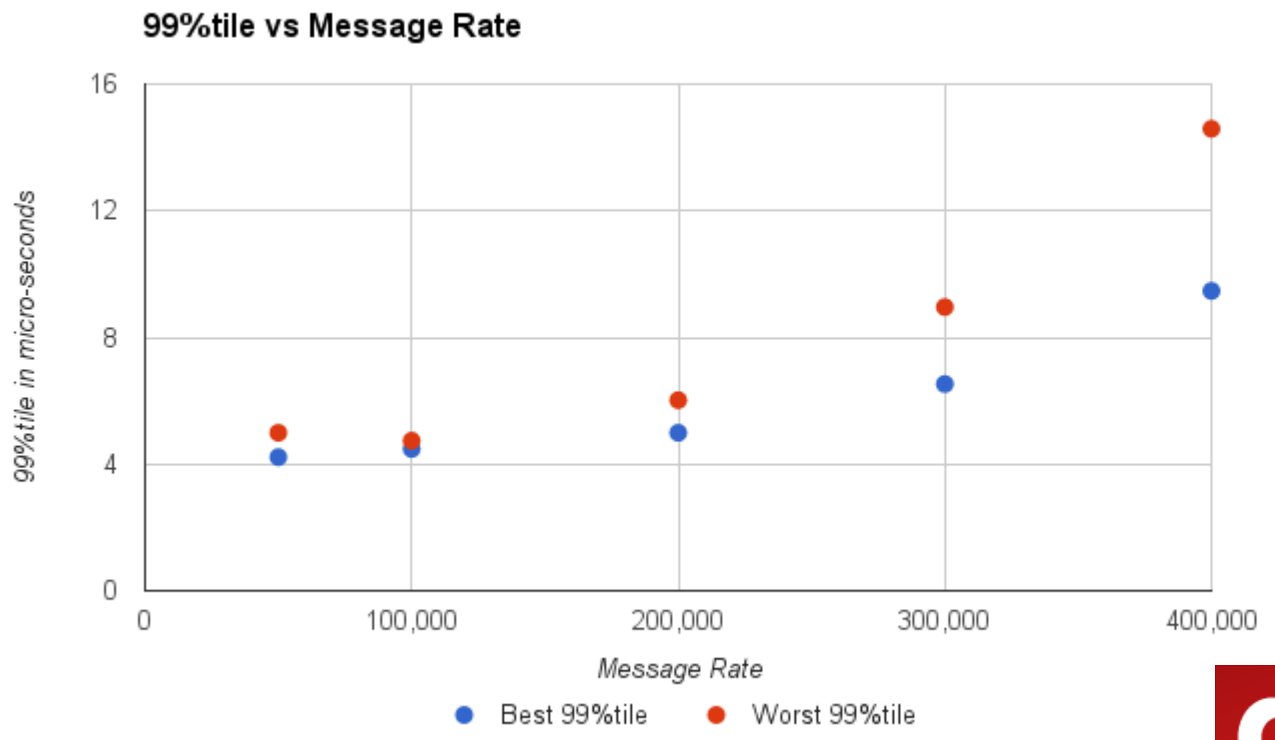
For a trivial service, the typical performance is consistent.

Tested on a E5-2650 v2. Fifteen 2 minute tests were compared.



Looking at the nines

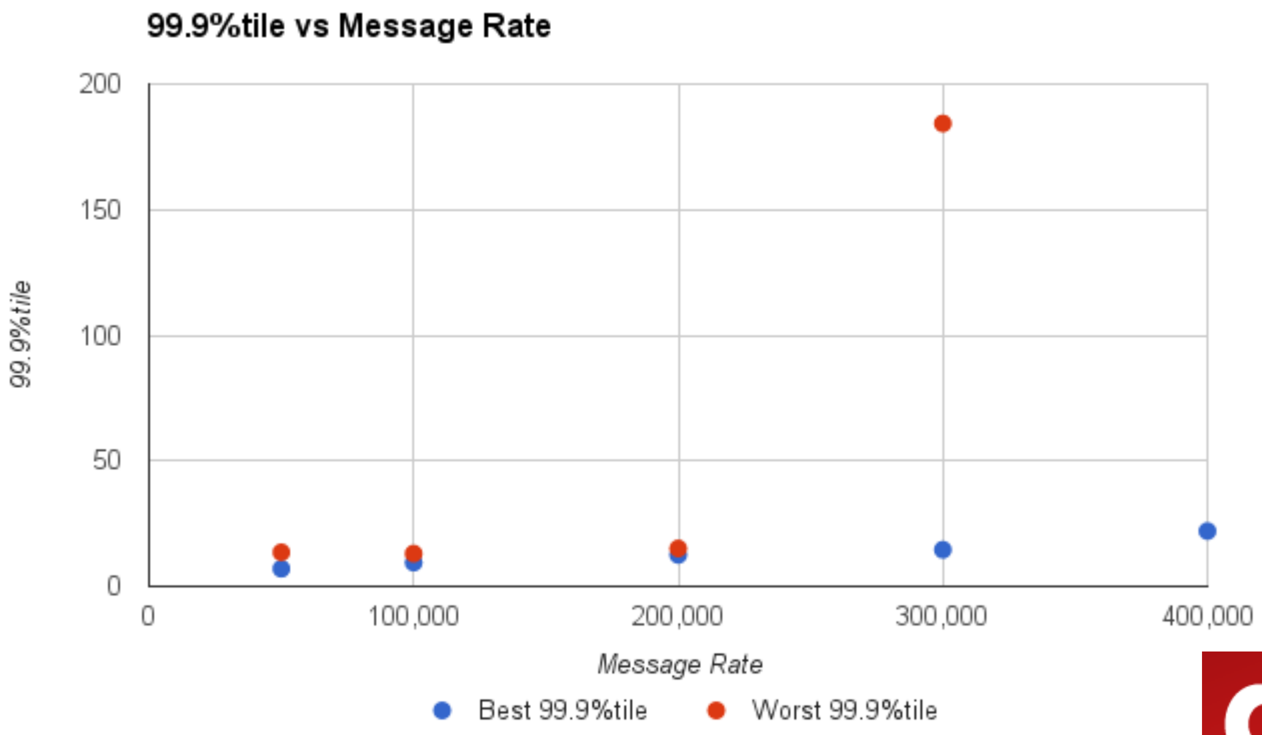
For a trivial service, the typical performance is consistent. At 99%, the throughput has an impact on latency.



Looking at the nines

At 99.9%, the throughput makes a big difference to the latency.

At 400K/s the 99.9%ile hit 19 ms in one 2 minute run.



No Flow Control?

Chronicle Queue is a producer centric solution, unlike most messaging systems which are consumer centric.

- many publishers don't support flow control (e.g. market data) or don't want it (reporting to compliance systems)
- Flow control can hide the worst latencies.
- Flow control means an asynchronous consumer is iteracting with it’s producer.

Without flow control, the producer and consumer behave the same whether tested stand alone or running at the same time as the producer is never impacted by a slow consumer.

Chronicle Queue Enterprise

We have a licensed version of Chronicle Queue which includes

- A ring buffer to minimise the jitter of persisting messages.
- Confirmed multi-node replication. The writer and reader can know or wait for a message to be successfully replicated.
- Replication throttling, and traffic shaping.
- Compressing of files on rolling.

Coming soon, multi-master multi-node queue.

Where can I try this out?

Low Latency Microservices examples

https://github.com/Vanilla-Java/Microservices

The OSS Chronicle products are available

https://github.com/OpenHFT/

Contact us for a trail version of Chronicle-FIX, Chronicle-Enterprise or Chronicle-Queue-Enterprise.

sales@chronicle.software

Chronicle

FIX – Micro seconds customisable FIX Engine

Enterprise – Monitoring, Traffic Shaping, Security

Queue-Enterprise – Confirmed Replication Distributed Queue

Journal – Custom Data Store, Key-Queue

Engine – Customisable Data Fabric, Reactive Live Queries

Queue – Persist every event

Map – Persisted Key-Value

Wire – YAML, Binary YAML, JSON, CSV, Raw data

Network – Remote access

Bytes – 64-bit off heap native + memory mapped files

Threads – Low latency

Core – Low level access to OS and JVM

In summary

- Microservices doesn’t mean you can do things differently, only improve what you are doing already.
- Introduce the Best Practices which make sense for you.
- You will have some Best Practices already.
- Trading Systems are distributed systems, even if on one machine.
- Lambda Architecture is simple so use it as much as possible, though it won’t solve all problems.

Q & A

Blog: http://vanilla-java.github.io/

http://chronicle.software

@ChronicleUG

sales@chronicle.software

https://groups.google.com/forum/#!forum/java-chronicle