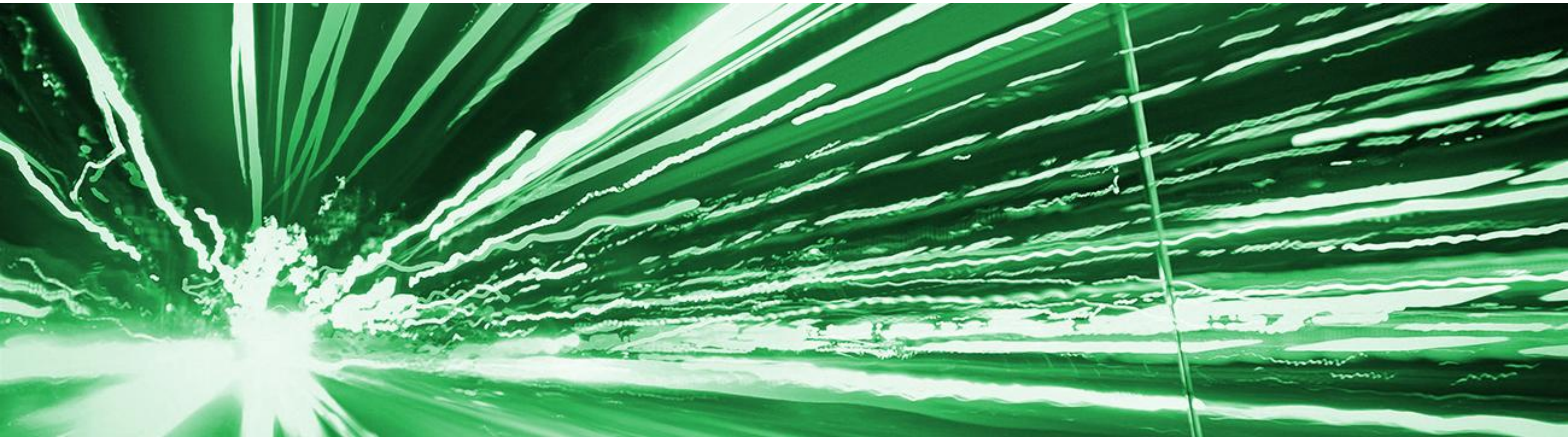


Latency Sensitive Microservices in Java

What Microservices can learn from Trading Systems and visa versa.

Peter Lawrey - CEO of Higher Frequency Trading
Docklands LJC - 2016

Chronicle

Peter Lawrey

Java Developer / Consultant for investment banks and hedge funds for 9 years.

Most answers for Java and JVM on stackoverflow.com

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

Chronicle Software

Build a skeleton high performance system in Java in a one week workshop.

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

```
                              CAP ALT GR1 GR2   sp:0     Sun  01/31/   01:20
MS-DOS V2.11 (UDP)
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
DDPMUP   SYS    14848   9-08-83  11:20a
DDANADMM SYS    14848   9-08-83  10:09a
DDCITOLR SYS    14976   9-08-83  10:19a
DDCNTXM  SYS    14848   9-08-83  10:20a
DDDS180  SYS    14848   9-08-83  10:21a
DDFXHR8  SYS    14976   9-08-83  10:25a
DDFXLRT  SYS    14976   9-08-83  10:23a
DDFXLR8  SYS    14976   9-08-83  10:24a
DDHP7470 SYS    12416   1-01-80  12:15p
DDIDSM   SYS    14848   9-08-83  10:20a
DDNECAPC SYS     7168  12-21-83   3:08p
CRTDUMP  COM    10881   1-31-84   9:30a
READ     ME     12928   2-01-84   3:01p
S        BAT       80   6-27-83
CONFIG   SYS       10   6-27-83
AUTOEXEC BAT        1   5-09-86  12:49p
       78 File(s)    441344 bytes free

A>
```

128 KB RAM

Where do Microservices come from?

UNIX Principle.

Staged Event Driven Architecture.

Service Orientated Architecture.

Lambda Architecture.

Reactive Streams.

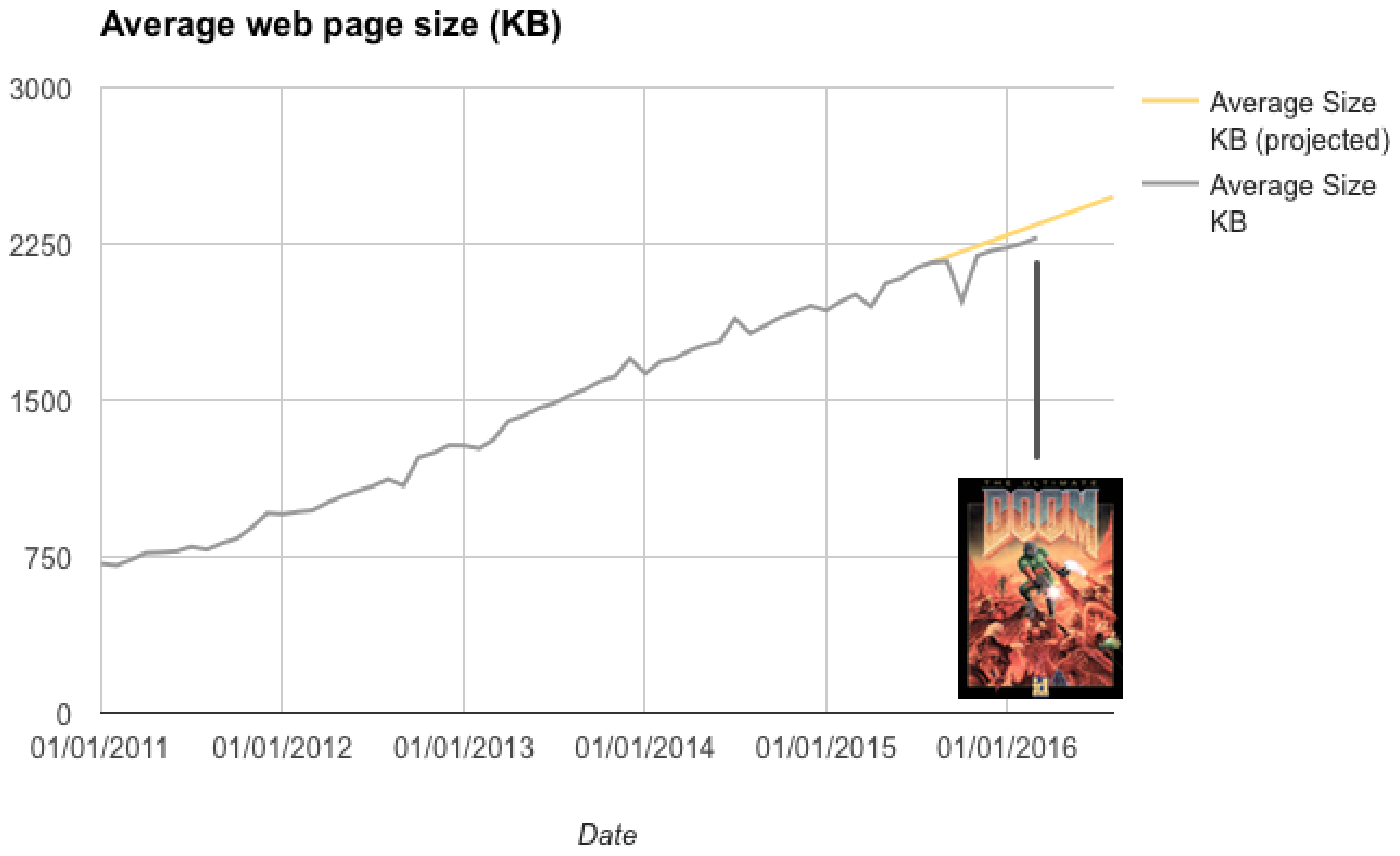
Where do Microservices come from?

Used in building Web applications.

“Micro-Web-Services”

24 fps, ~1 per 40 ms

The web is DOOM



https://mobiforge.com/research-analysis/the-web-is-doom

Microservices denial?

Microservices bring together best practices from a variety of areas.

Most likely you are already using some of these best practices.

Microservices denial?

It sounds like marketing hype.

Microservices denial?

It sounds like marketing hype.

It all sounds pretty familiar.

Microservices denial?

It sounds like marketing hype.

It all sounds pretty familiar.

It just a rebranding of stuff we already do.

Microservices denial?

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.

Microservices denial?

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.

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	★	★★	★★★

Using 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?

You have a view on how much the response time of a system costs your business.

or

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.

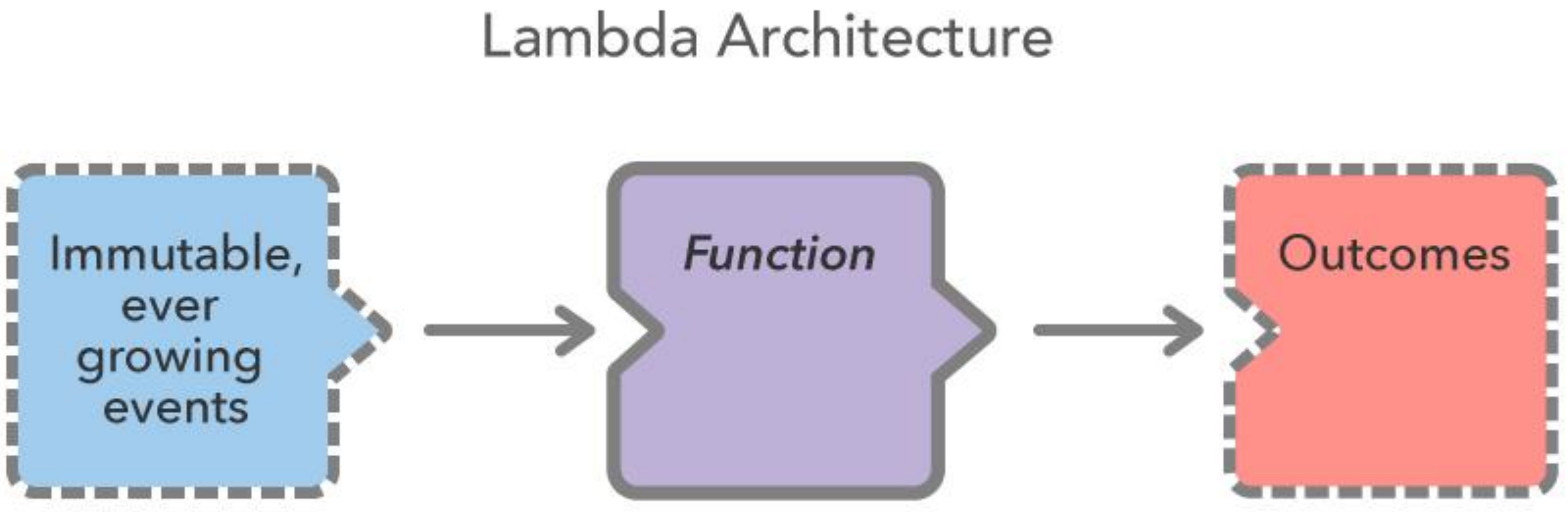
This was unacceptable to them.

We fixed this bug and dropped it to below 35 micro-seconds.

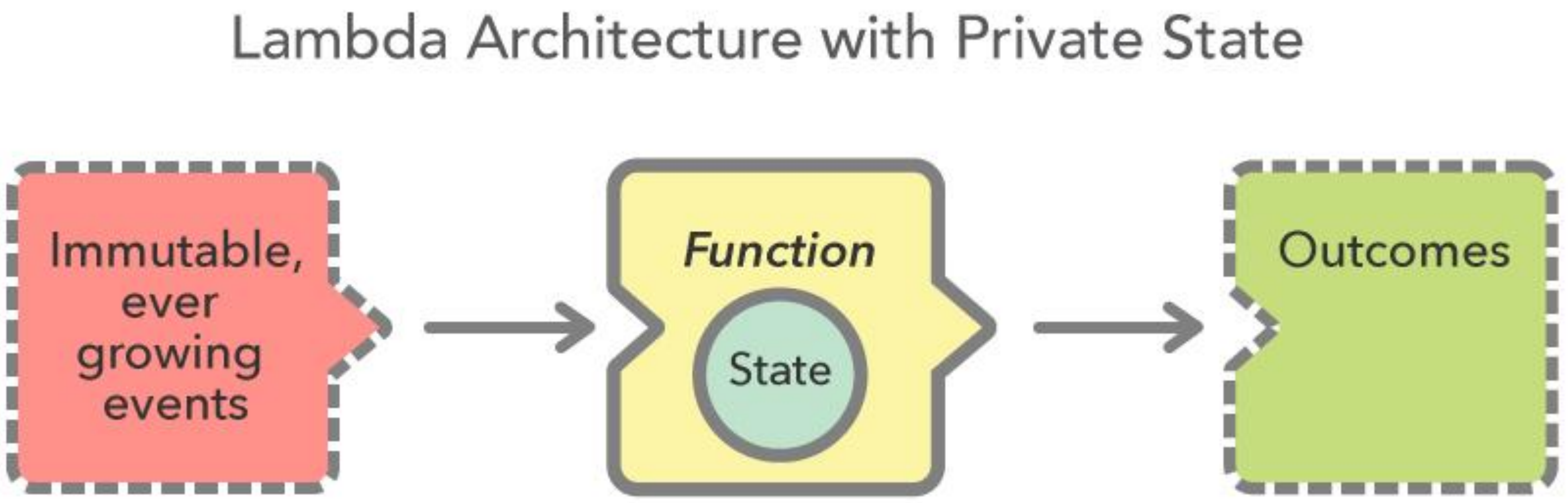
Where do they overlap.

- 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.

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

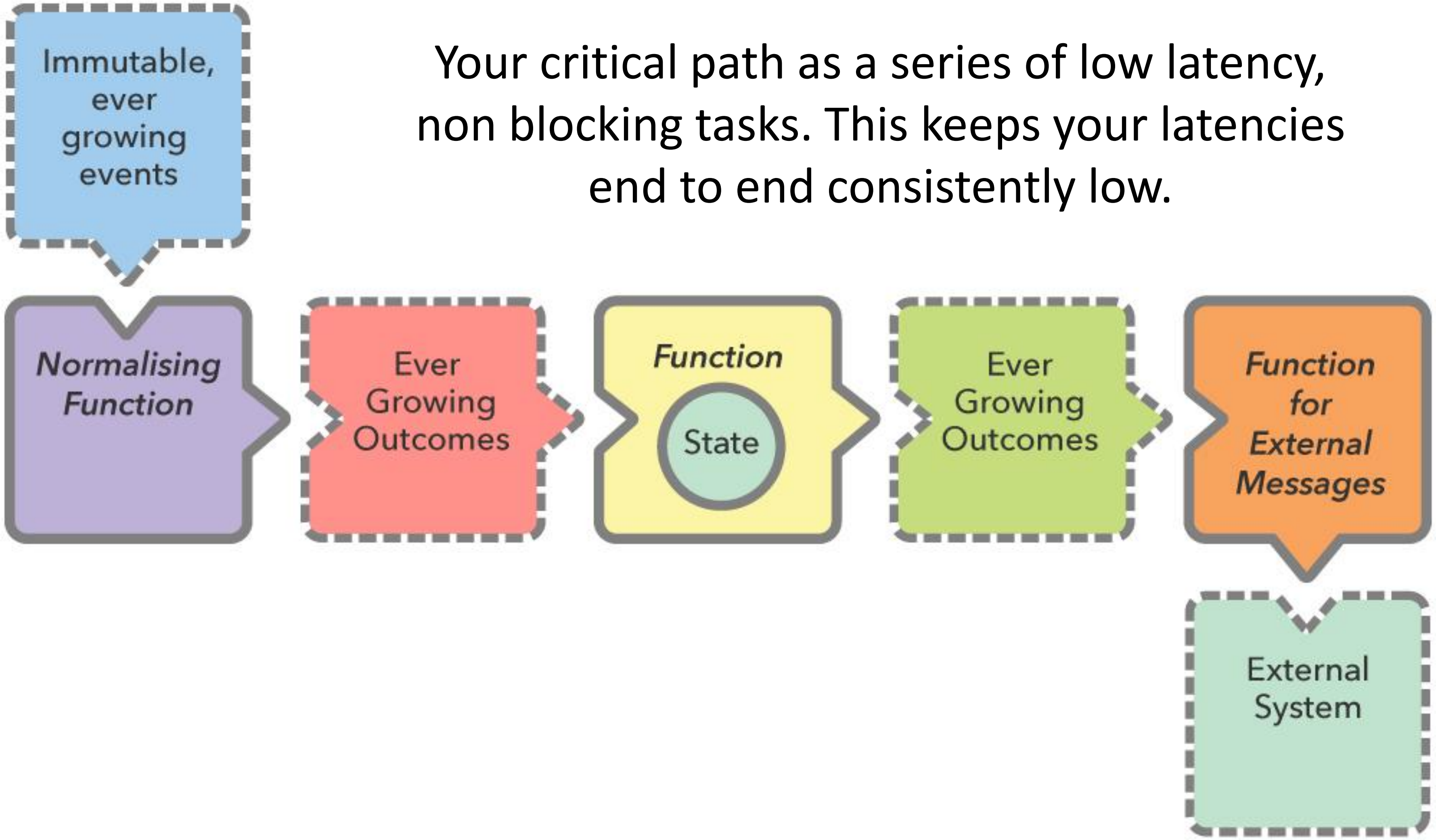


Each output is the result of ALL the inputs. Instead of replying ALL input message each time, the Function could save an accumulated state.

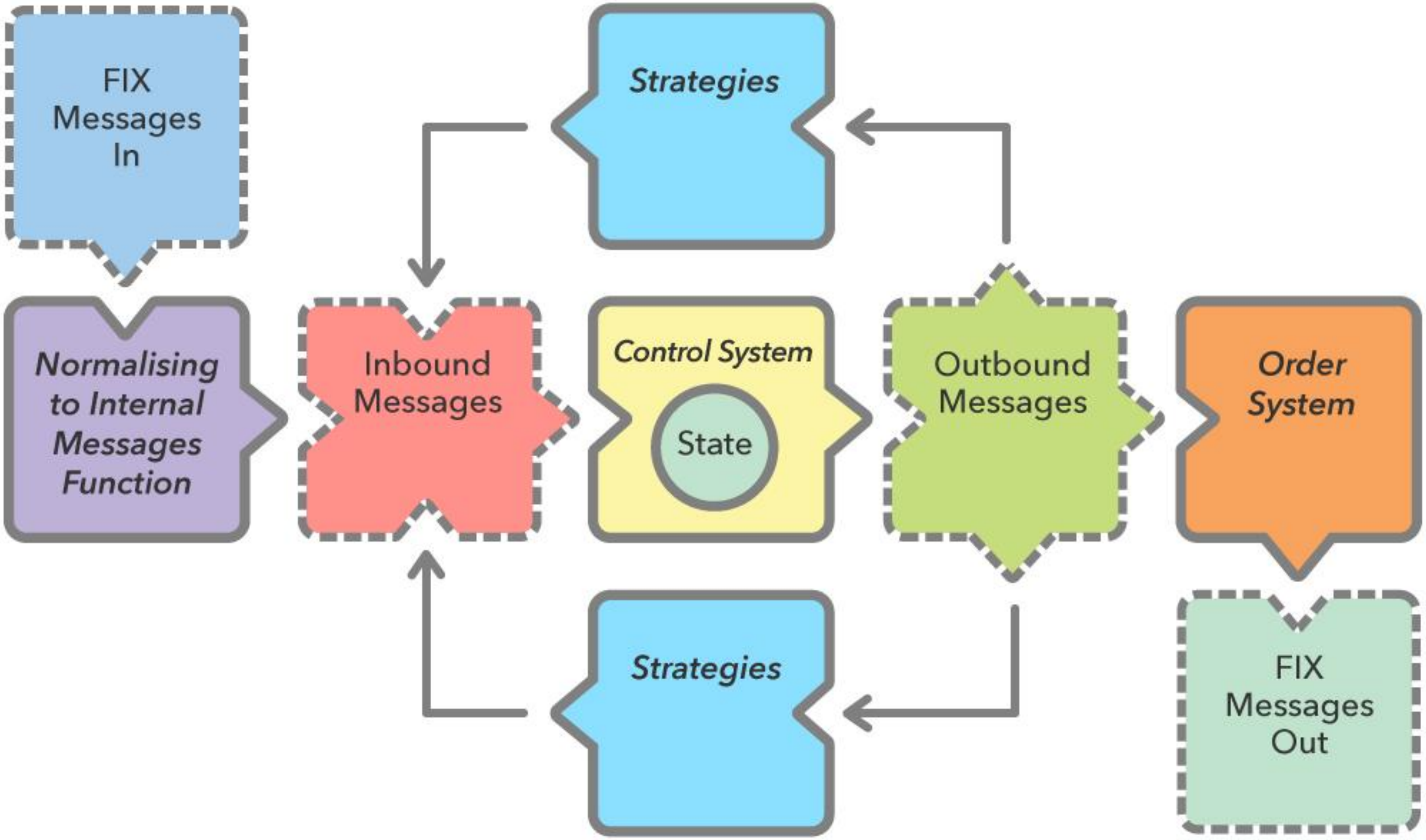

Lambda Architecture with Private State
Immutable, ever growing events
Function
State
Outcomes

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



To go faster use private data

Micro-services do something simple
with privately held data.

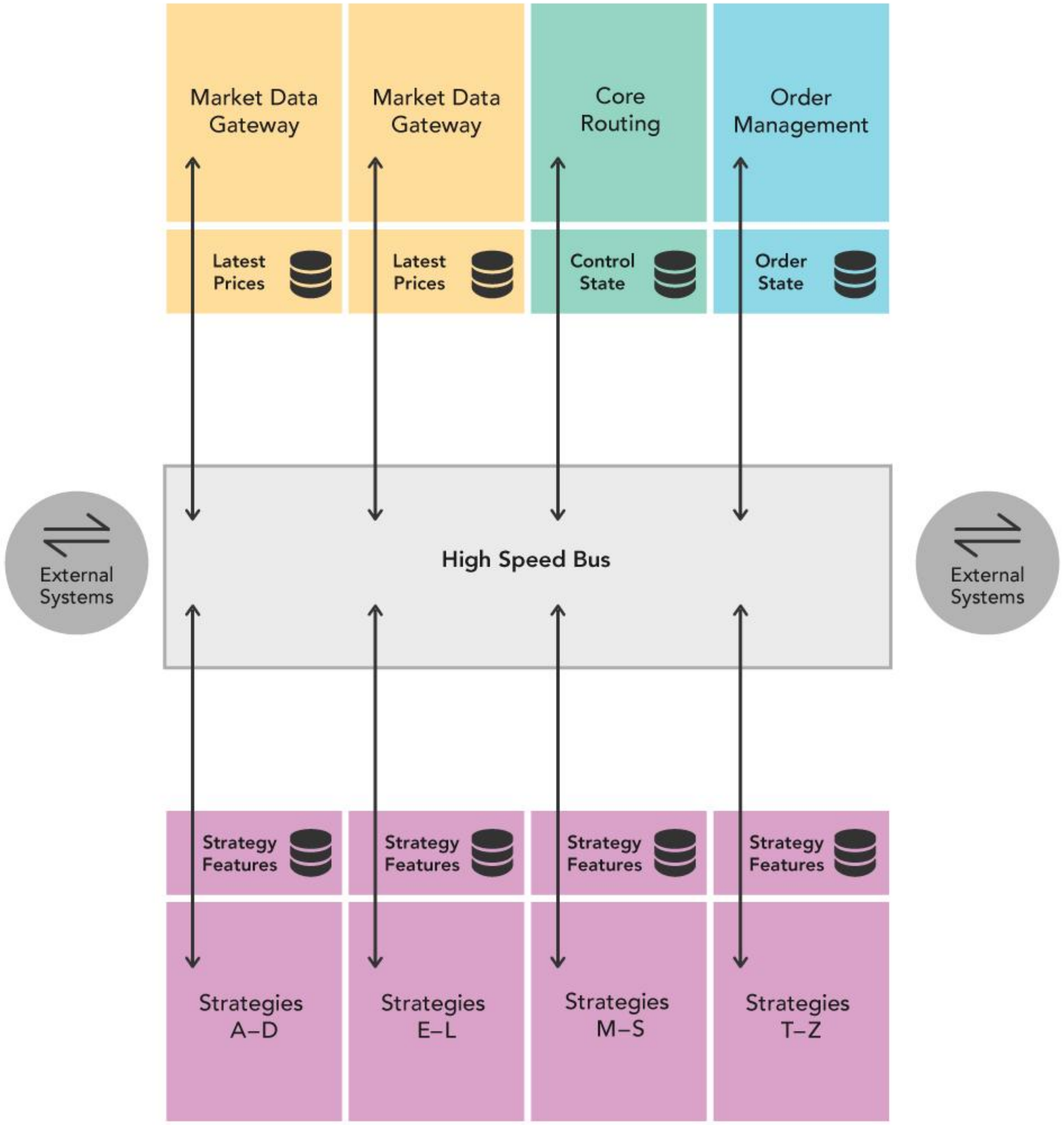
Cache	Size	Clock Cycles	Private
L1 Instruction	32 KB	3	Yes
L1 Data	32 KB	3	Yes
L2 Cache	256 KB	10	Yes
L3 Cache	1 MB – 48 MB	40 - 70	NO

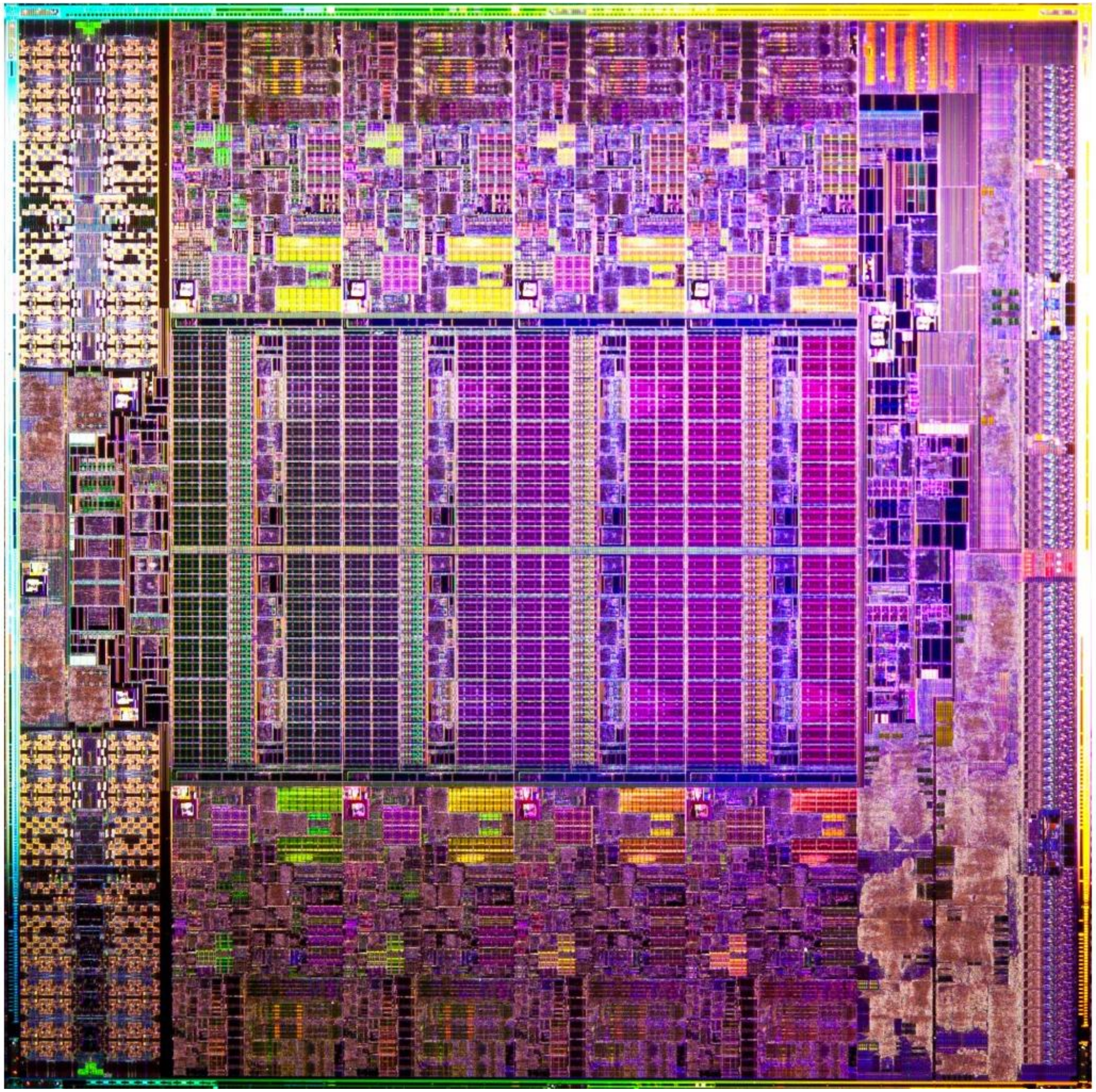
A Computer is a Distributed System.

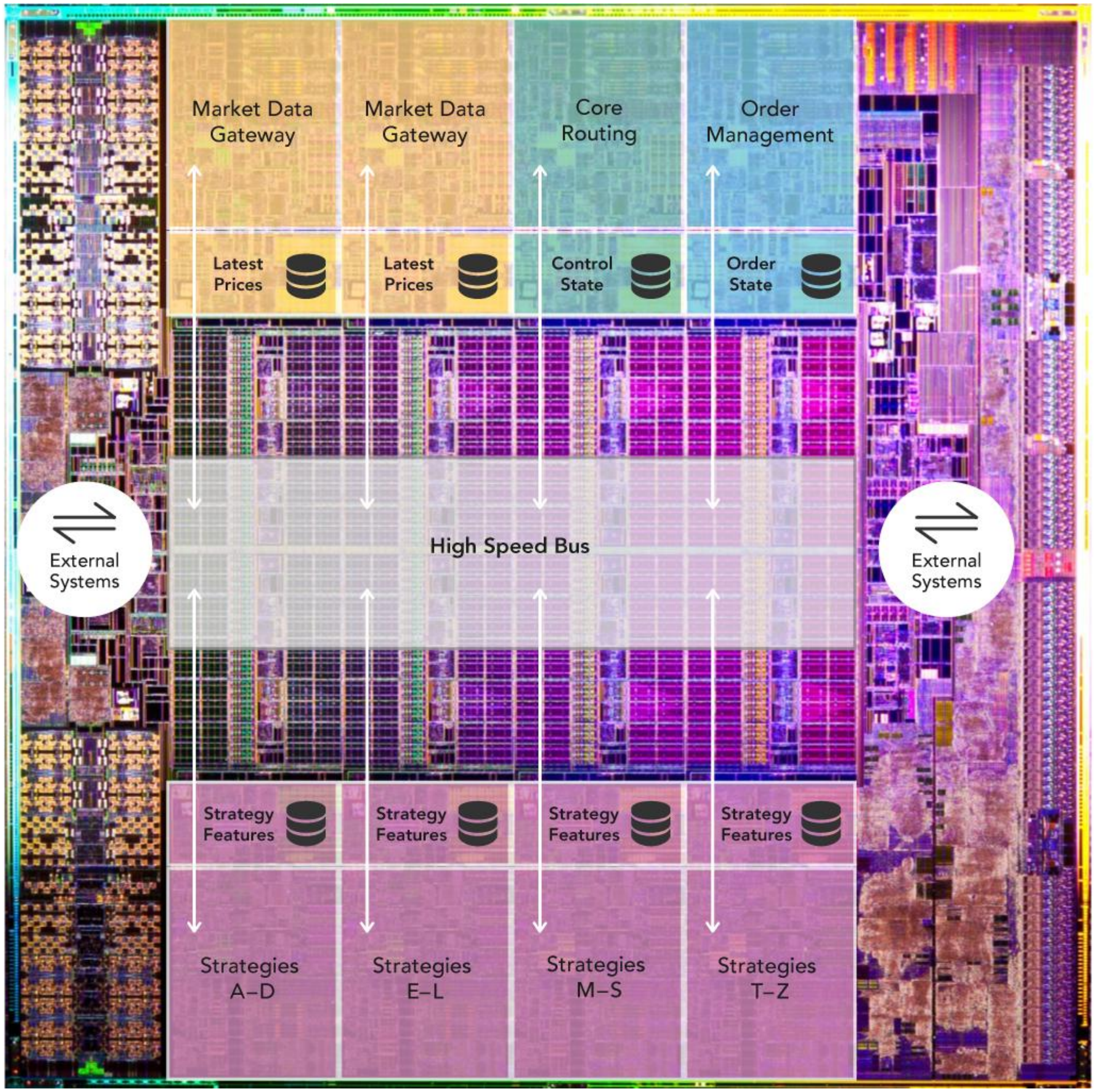
When you are considering short time scales of 10 micro-seconds or less, you have to consider that each core as a processor of it’s own.

Each core

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


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




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

Testing and Debugging Microservices

Frameworks can make testing and debugging harder.

You need to be able to test and debug your components without the framework, or a transport.

Turning a Monolith into Microservices

Business Component + Transport = Service.

Starting with a simple contract

An asynchronous message has a type, a payload and doesn't return a result.

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

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

A Data Transfer Object

```
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) {
        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 a result

```
public class SidedMarketDataCombiner
        implements SidedMarketDataListener {

    final MarketDataListener mdListener;

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

Writing a simple component

The component calculates a result, using private state.

```java
final Map<String, TopOfBookPrice> priceMap = new TreeMap<>();

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

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

We can mock the output listener of our component.

```
// 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);
```

Testing multiple components

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

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));

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

verify(listener);
```

Adding a transport

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

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

Making messages transparent

```
orderManager.onOrderIdea(
   new OrderIdea("EURUSD", Side.Buy, 1.1180, 2e6));
```

```
--- !!data #binary
onOrderIdea: {
    symbol: EURUSD,
    side: Buy,
    limitPrice: 1.118,
    quantity: 2000000.0
}
```

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.

```
--- !!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
```

No Flow Control?

Market Data

Compliance

In summary

Microservices doesn’t mean you have to do everything differently, only improve what you are doing already.

In summary

Microservices doesn’t mean you have to do everything differently, only improve what you are doing already.

Introduce the Best Practices which make sense for you.

In summary

Microservices doesn’t mean you have to do everything differently, only improve what you are doing already.

Introduce the Best Practices which make sense for you.

You will have some Best Practices already.

In summary

Microservices doesn’t mean you have to do everything 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 on one machine.

In summary

Microservices doesn’t mean you have to do everything 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.

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/

Chronicle

Q & A

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

http://chronicle.software

@ChronicleUG

sales@chronicle.software

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