Designing, Building and Testing Deterministic HFT Systems

QCon Shanghai 2019

Peter Lawrey

CEO, Chronicle Software

主办方 Geekbang. 极客邦科技 InfoQ

收获国内外一线大厂实践 与技术大咖同行成长

✓ 演讲视频 ✓ 干货整理 ✓ 大咖采访 ✓ 行业趋势

关注 QCon 公众号

Peter Lawrey

Chronicle SOFTWARE

Self funded
$2.1m/y revenue

The Central Repository
17m downloads 2018

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

Several Tier 1
banking clients

Java™ Champions

Use Cases

Low latency
FIX Engine
< 20 us 99.9%

Low latency
Replication
< 30 us

Low latency
Persisted messaging
< 1 us 99%

Custom FX Trading
With 3 month ROI

Low latency trading microservice

A process which handles events and transaction in around 10 – 100 microseconds

Ideally, it should be only 1% busy on average.

Chronicle SOFTWARE

Sources of latency

- Network and OS
- Any messaging/persistence
- The core business logic
- Garbage collection

Lowering latency

The less you do the faster it will be. Your process should only do essential work and avoid time spent in abstraction.

Doing less, testing with JMH

Chronicle SOFTWARE

```java
double price = 1.25;
double quantity = 1e6;

@Benchmark
public double multipleAndRoundDouble() {
    double value = price * quantity;
    return Math.round(value * 1e2) / 1e2;
}

@Benchmark
public double multipleAndRoundBigDecimal() {
    return BigDecimal.valueOf(price)
            .multiply(BigDecimal.valueOf(quantity))
            .setScale(2, BigDecimal.ROUND_HALF_UP)
            .doubleValue();
}
```

Doing less

worst	double	BigDecimal
Average	0.052 µS	0.28 µS
1 in 100	0.1 µS	0.4 µS
1 in 1000	0.9 µS	17 µS

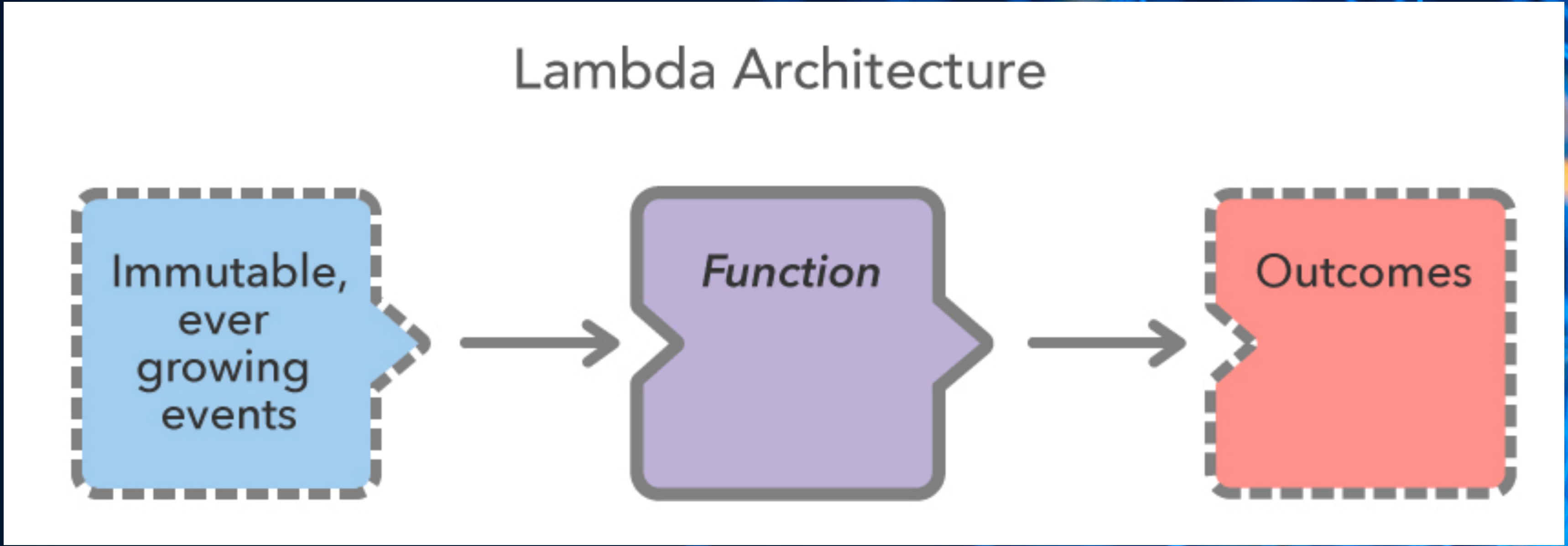
Give yourself a budget

e.g. 24 GB/day

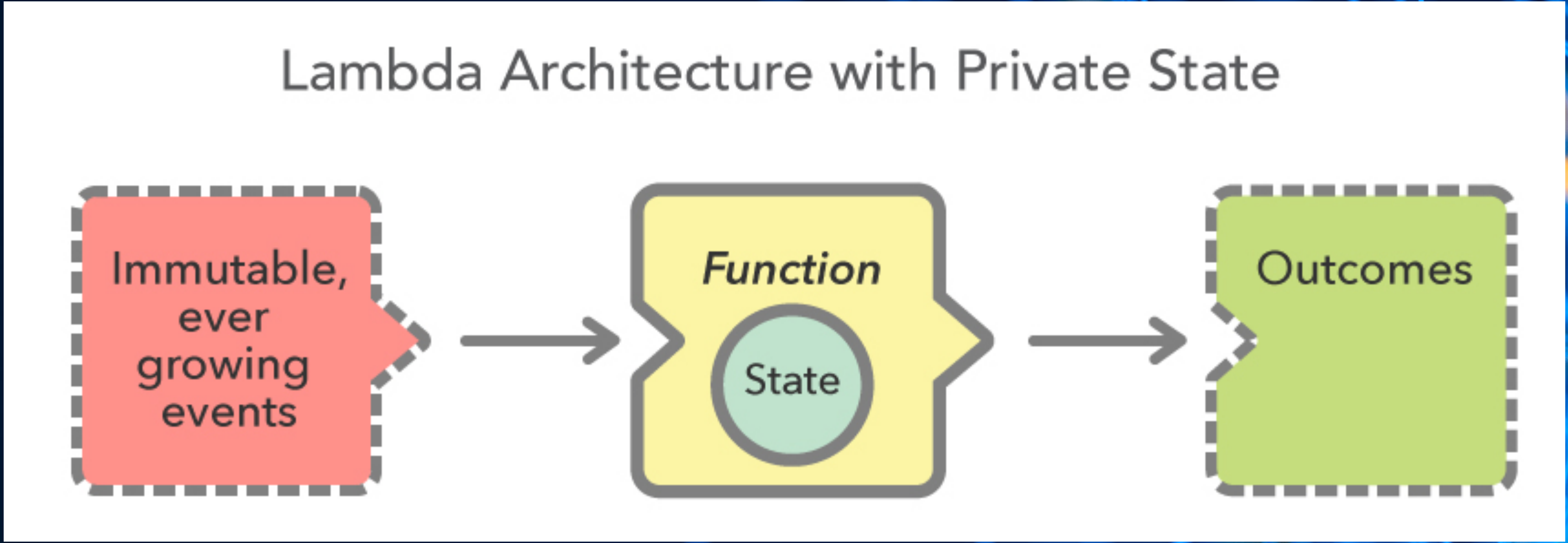
or 1 GB/hour

=> 1 GC a day

Real time processing of transactions

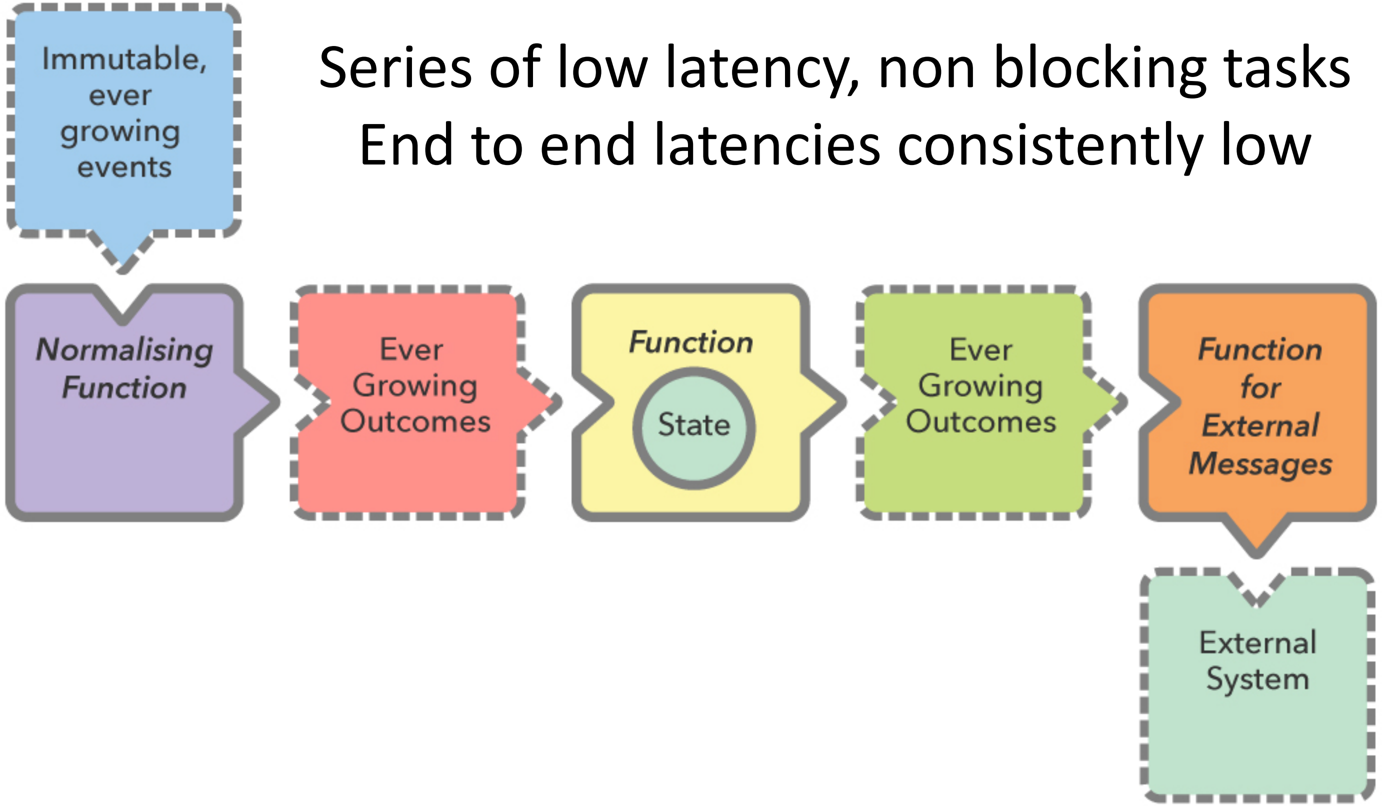


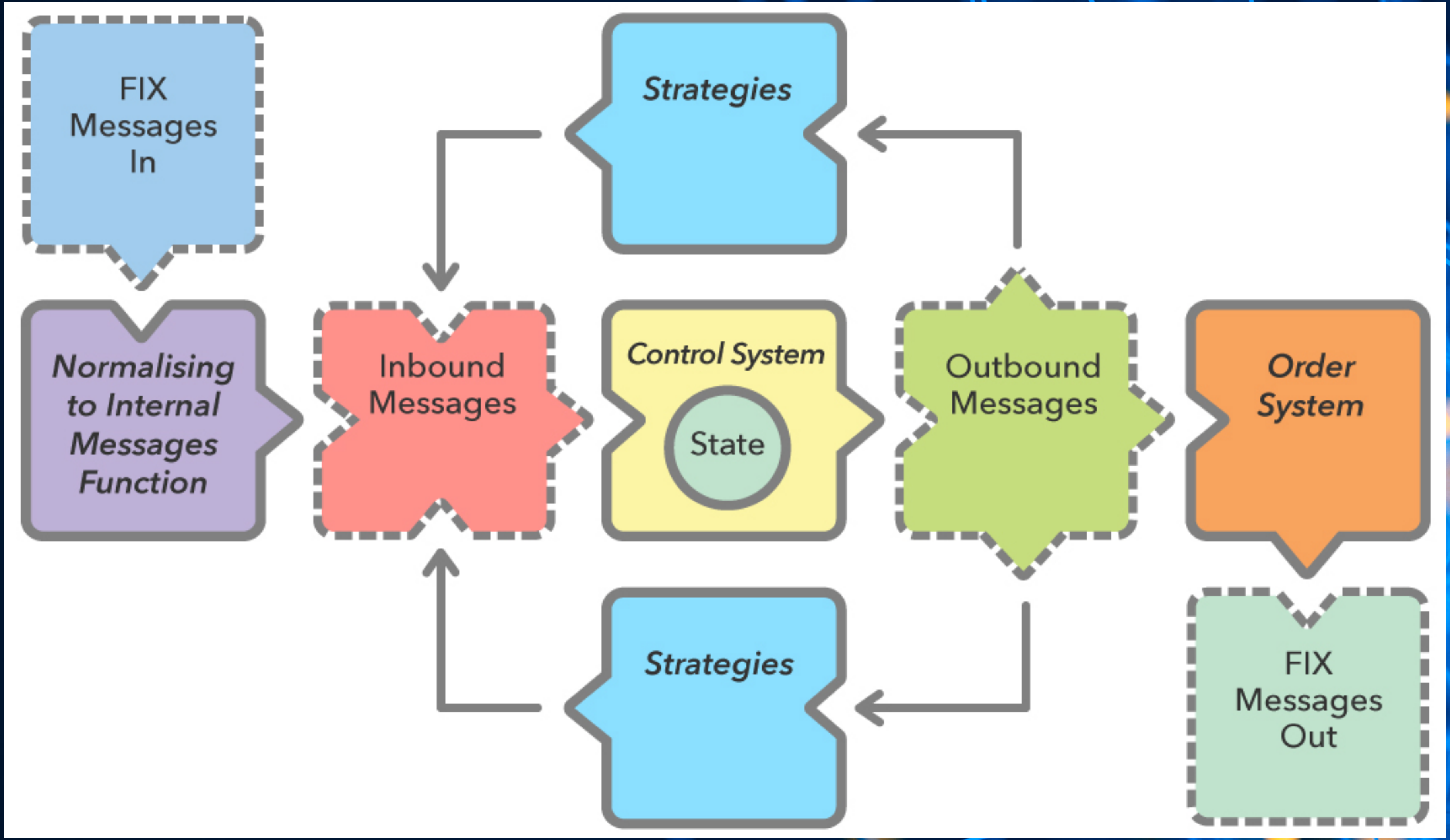
Real time processing of transactions


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

Lambda Architecture Services Chained

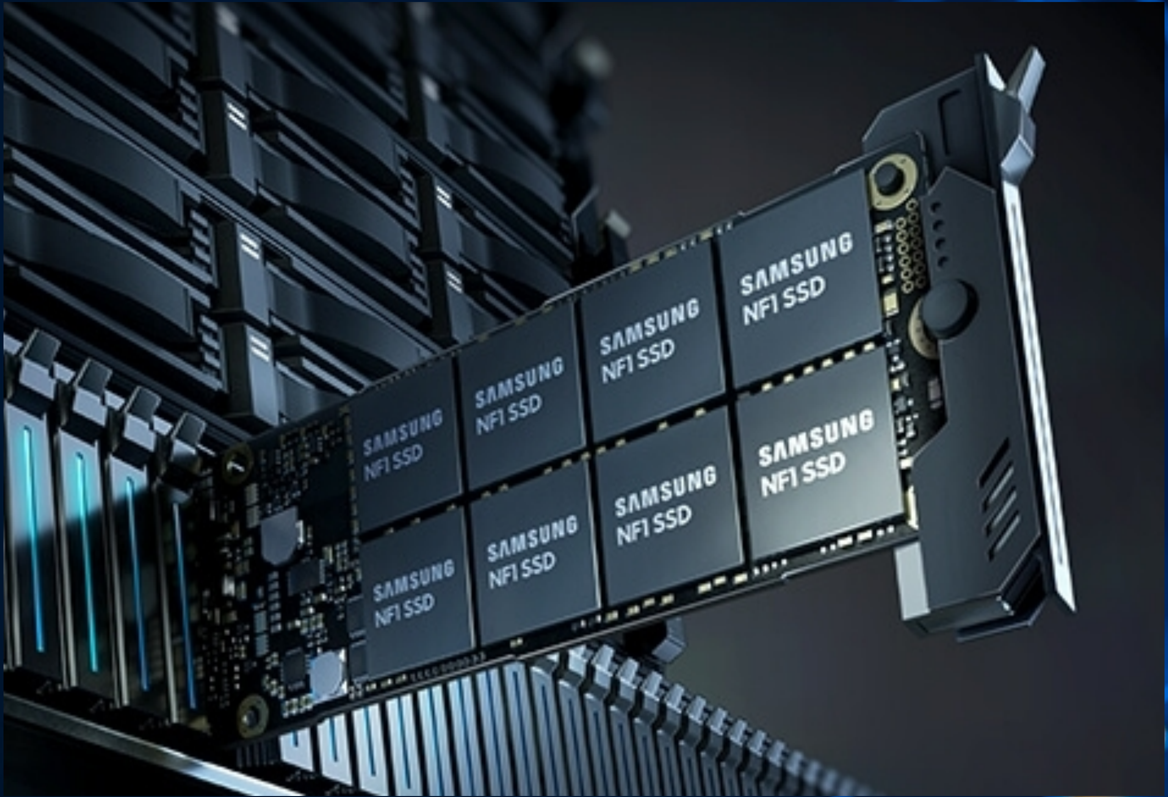
Series of low latency, non blocking tasks
End to end latencies consistently low




FIX Messages In
Normalising to Internal Messages Function
Inbound Messages
Strategies
Control System
State
Outbound Messages
Strategies
Order System
FIX Messages Out

Store every event model is getting cheaper

Chronicle SOFTWARE



3.84 TB- £830
7.68 TB - £2100
15.36 TB - £4,200

How do we guarantee correctness?

Each microservice should be designed to be deterministic so it produces the same result for given inputs every time.

How do we guarantee correctness?

Chronicle SOFTWARE

Time can be an input for time sensitive actions such as expiry. Time should be an input which is recorded, testable and replayable.

Recovery in a low latency system

The state of any microservice can be reconstructed by replying the inputs. This can take too long so the system can dump a progressive snapshot over N minutes

Recovery in a low latency system

On failover, restart or upgrade a system can read all of it’s inputs and/or the resulting state changes to recreate it’s state.

Replayability reduces time to fix.

Chronicle SOFTWARE

Problems can be recreated quickly by taking the data from production, creating the same state on a development system, debugging and testing any fix.

Testing microservices

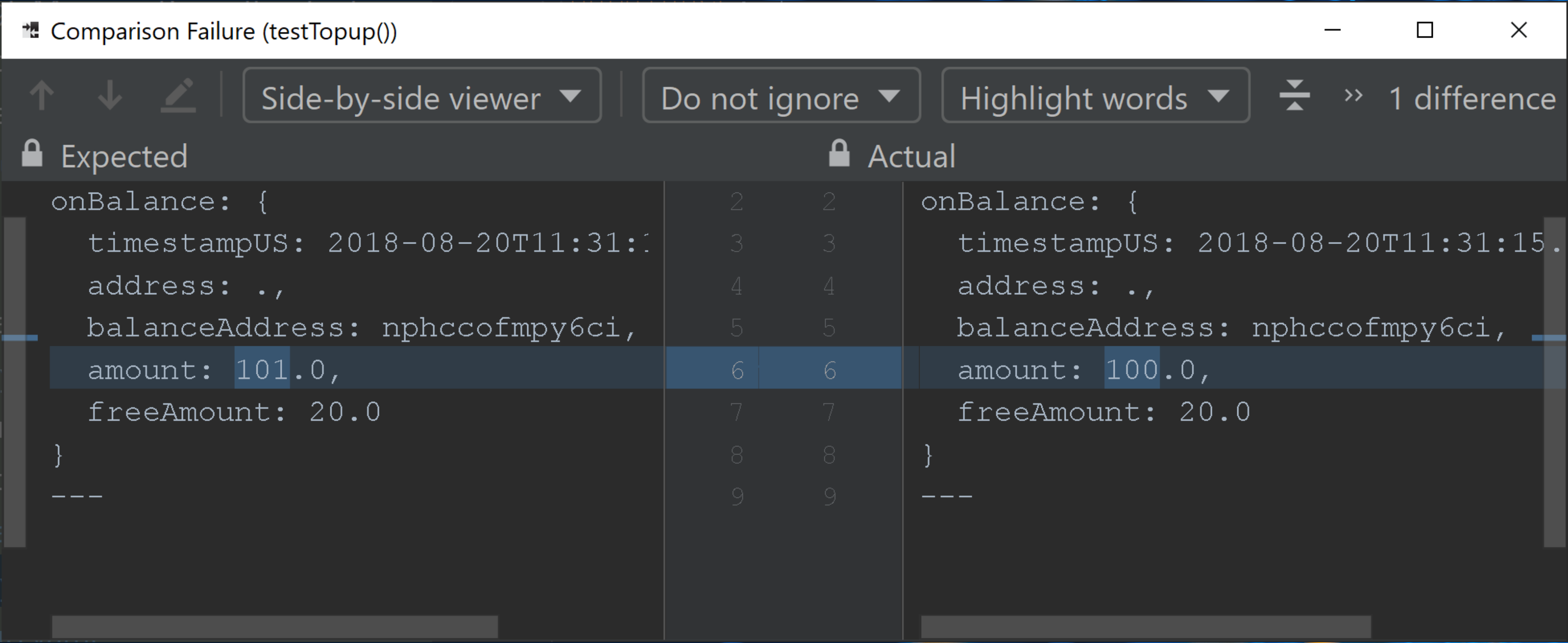
```
# setup.yaml
openingBalance: {
  timestampUS: 2018-08-20T11:31:15.373001,
  address: .,
  balanceAddress: nphccofmpy6ci,
  amount: 100.0
}
---
```

```
# in.yaml
topup: {
  timestampUS: 2018-08-20T11:31:15.379010,
  address: nphccofmpy6ci,
  amount: 20.0,
}
---
```

Testing microservices

```
# out.yaml
to: nphccofmpy6ci
onBalance: {
  timestampUS: 2018-08-20T11:31:15.37901,
  address: .,
  balanceAddress: nphccofmpy6ci,
  amount: 100.0,
  freeAmount: 20.0
}
---
```

Testing microservices



Expected:

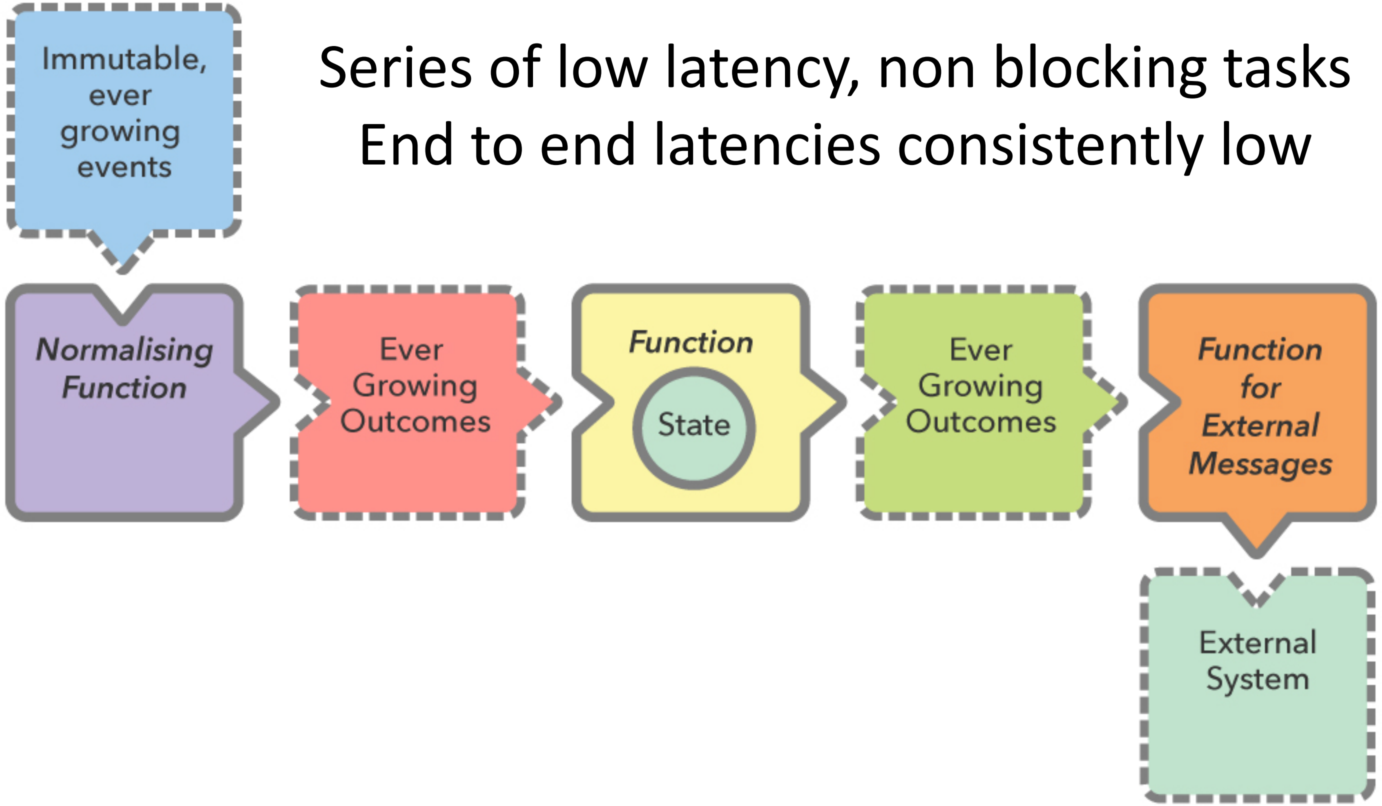
```
onBalance: {
  timestampUS: 2018-08-20T11:31:1
  address: .,
  balanceAddress: nphccofmpy6ci,
  amount: 101.0,
  freeAmount: 20.0
}
---
```

Actual:

```
onBalance: {
  timestampUS: 2018-08-20T11:31:15.
  address: .,
  balanceAddress: nphccofmpy6ci,
  amount: 100.0,
  freeAmount: 20.0
}
---
```

Lambda Architecture Services Chained

Series of low latency, non blocking tasks
End to end latencies consistently low



Order example

```
exchangeConfigEvent: {
  timestampUS: 1970-01-01T00:00:00,
  address: config,
  currencies: {
    ? BTC_USD: { tickSize: 1.0, minOrderSize: 0.0001 }
  }
}
---
indexMid: {
  indexMid: 500,
  symbol: BTC_USD_P0,
}
```

Order example

```
createOrder: {
  clOrdID: clOrdID4,
  ordType: '2',
  eventTime: 2019-01-24T16:57:39.843143,
  side: BUY,
  originalCounterCcyQty: 100.0,
  price: 200.0,
  symbol: BTC_USD
}
```

Order example

```
createOrder: {
  clOrdID: clOrdID6,
  ordType: '2',
  eventTime: 2019-01-24T16:57:39.843146,
  side: SELL,
  originalCounterCcyQty: 100.0,
  price: 210.0,
  symbol: BTC_USD
}
```

Order example

```
trade: {
  eventTime: 2019-01-24T16:57:39.843148,
  tradeID: MEir800000,            # the unique trade id
  aggressorClOrdID: clOrdID6,
  initiatorClOrdID: clOrdID4,
  price: 210.0,                   # The price that the trade archived
  qty: 0.47619,                   # the order quantity in counter currency,
  initiatorSide: SELL,            # side from the initiator perspective
  symbol: BTC_USD_P0,             # BTC_USD_P0 is the perpetual BTC USD swa
  counterCcyQty: 100.0,           # the order qty in the counter currency,
}
```

Order example

```
trade: {
  eventTime: 2019-01-24T16:57:39.843148,
  tradeID: MEir800000,        # the unique trade id
  aggressorClOrdID: clOrdID6,
  initiatorClOrdID: clOrdID4,
  price: 210.0,               # The price that the trade archived
  qty: 0.47619,               # the order quantity in counter currency,
  initiatorSide: SELL,        # side from the initiator perspective
  symbol: BTC_USD_P0,         # BTC_USD_P0 is the perpetual BTC USD swa
  counterCcyQty: 100.0,       # the order qty in the counter currency,
}
```

Order Example

```
public interface CreateOrderListener {
    void createOrder(CreateOrder createOrder);
}
```

```
public interface TradeListener {
    void trade(Trade trade);
}
```

```java
public class CreateOrder extends AbstractEvent<CreateOrder> {
    @LongConversion(CurrencyPair.class)
    @Comment(SYMBOL)
    private long symbol;

    @LongConversion(UniqueID.class)
    @Comment(CL_ORD_ID)
    private long clOrdID;

    @Comment(TIME_STAMP)
    @LongConversion(MicroTimestampLongConverter.class)
    private long timestampUS;

    @Comment(ADDRESS)
    @LongConversion(AddressLongConverter.class)
    private long address;

    @LongConversion(MicroDurationLongConverter.class)
    private long ttl;
```

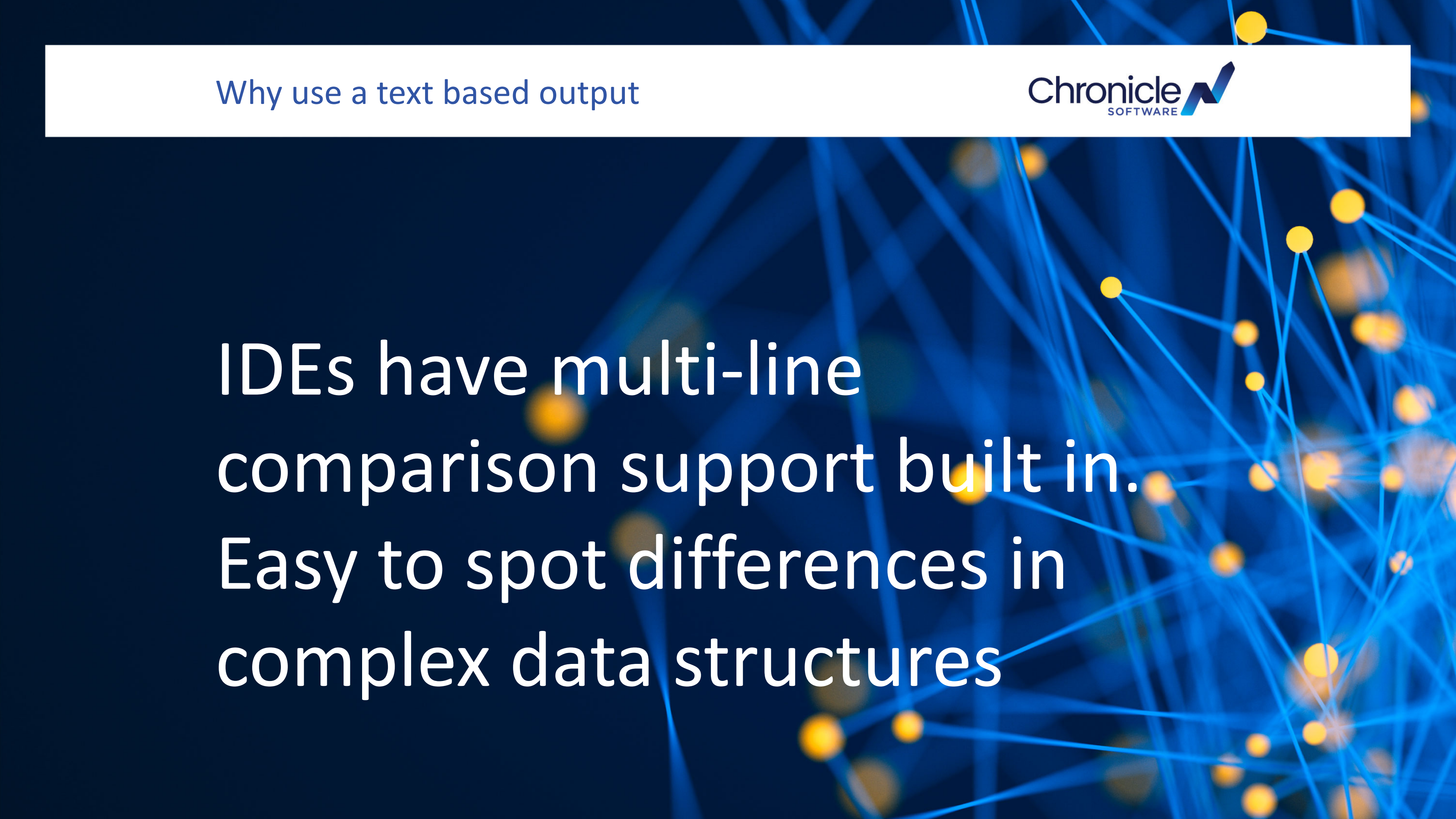
Order Example

```
public class Trade extends AbstractEvent<Trade> {

    @LongConversion(UniqueID.class)
    @Comment(TRADE)
    private long tradeID;

    @LongConversion(UniqueID.class)
    @Comment(CL_ORD_ID)
    private long aggressorClOrdID;

    @LongConversion(UniqueID.class)
    @Comment(CL_ORD_ID)
    private long initiatorClOrdID;
```



Why use a text based output

Chronicle SOFTWARE

IDEs have multi-line comparison support built in. Easy to spot differences in complex data structures

Why use a text based output

Easy to fix test by copy-pasting the expected result

Why use a text based output

Chronicle SOFTWARE

Easy to regress all the test at once and compare the differences on check in, rather than alter each test

Questions

https://www.linkedin.com/in/peterlawrey/

Geekbang> 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播

关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货