JAX FINANCE

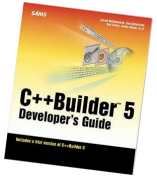
Algorithmic Architecture

Performant Architecture in the Evolving Regulatory Landscape

Jamie Allsop

DSP background with a PhD in **adaptive framework design**

focused on **C++** & standards work BSI | C++ Panel



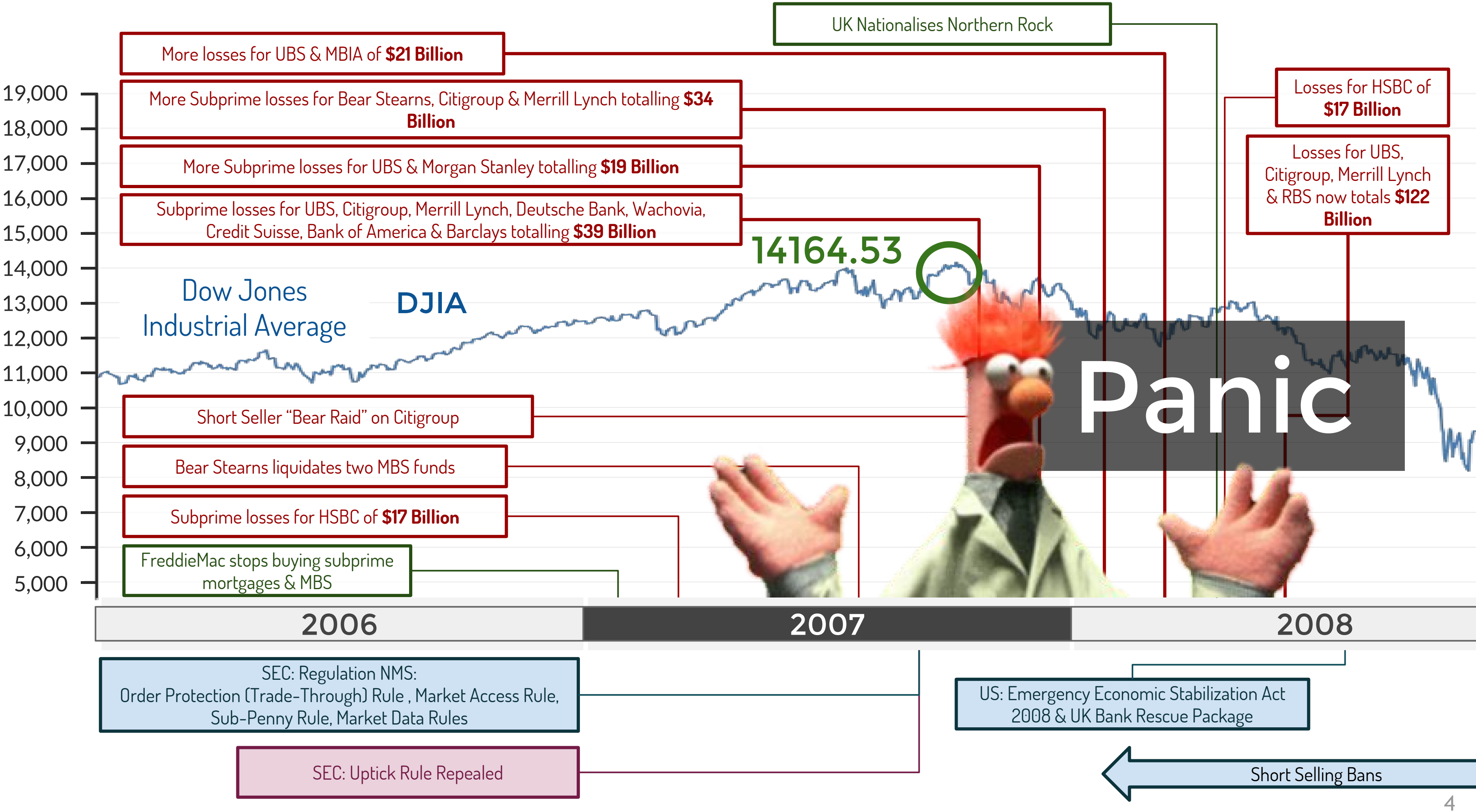
passionate about **agile** agile-trac Agile Integrated Project Collaboration

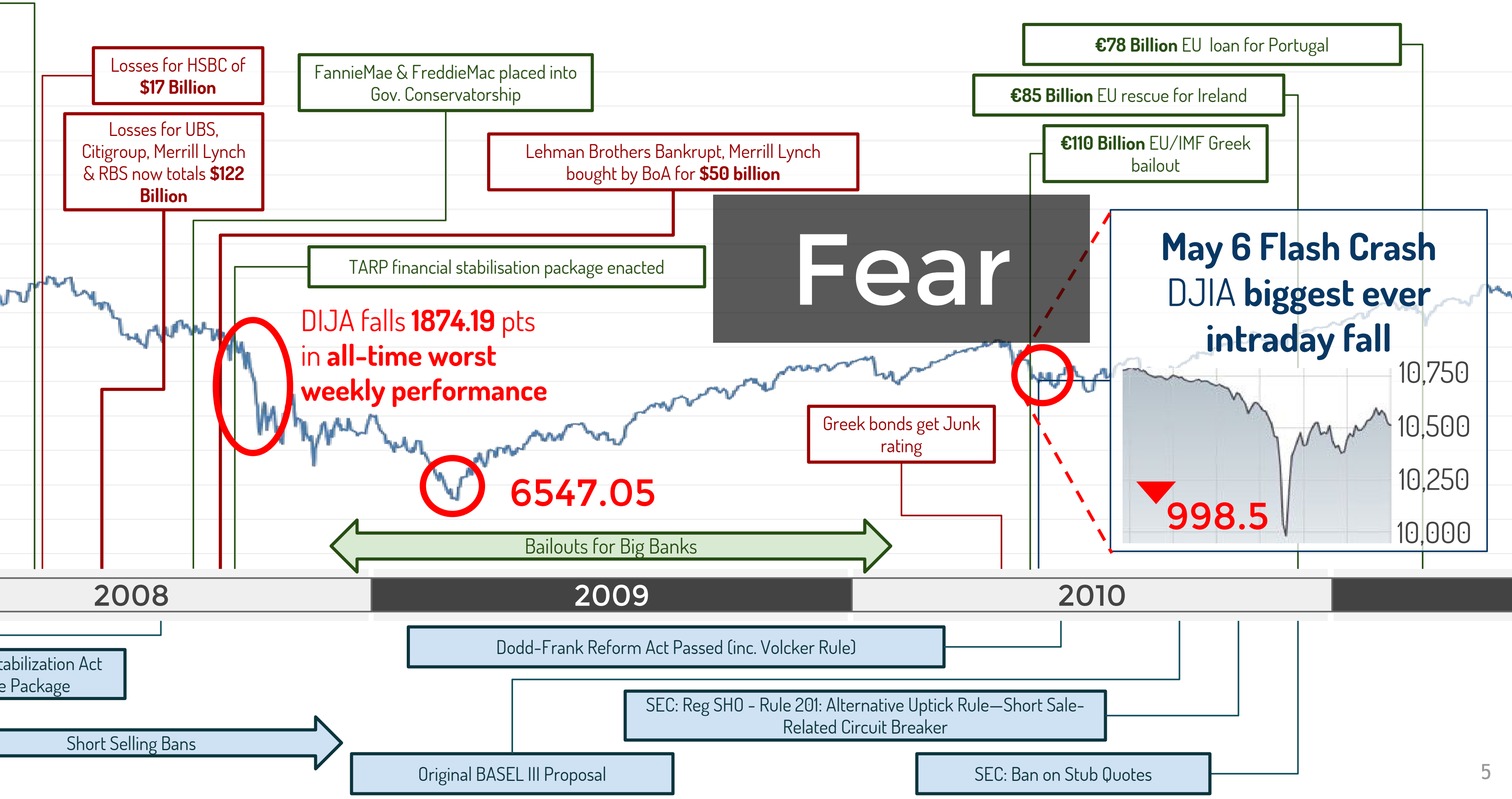
fiddle with python—pypi/**cuppa** for Scons

ended up at NYSE Euronext Powering the exchanging world.

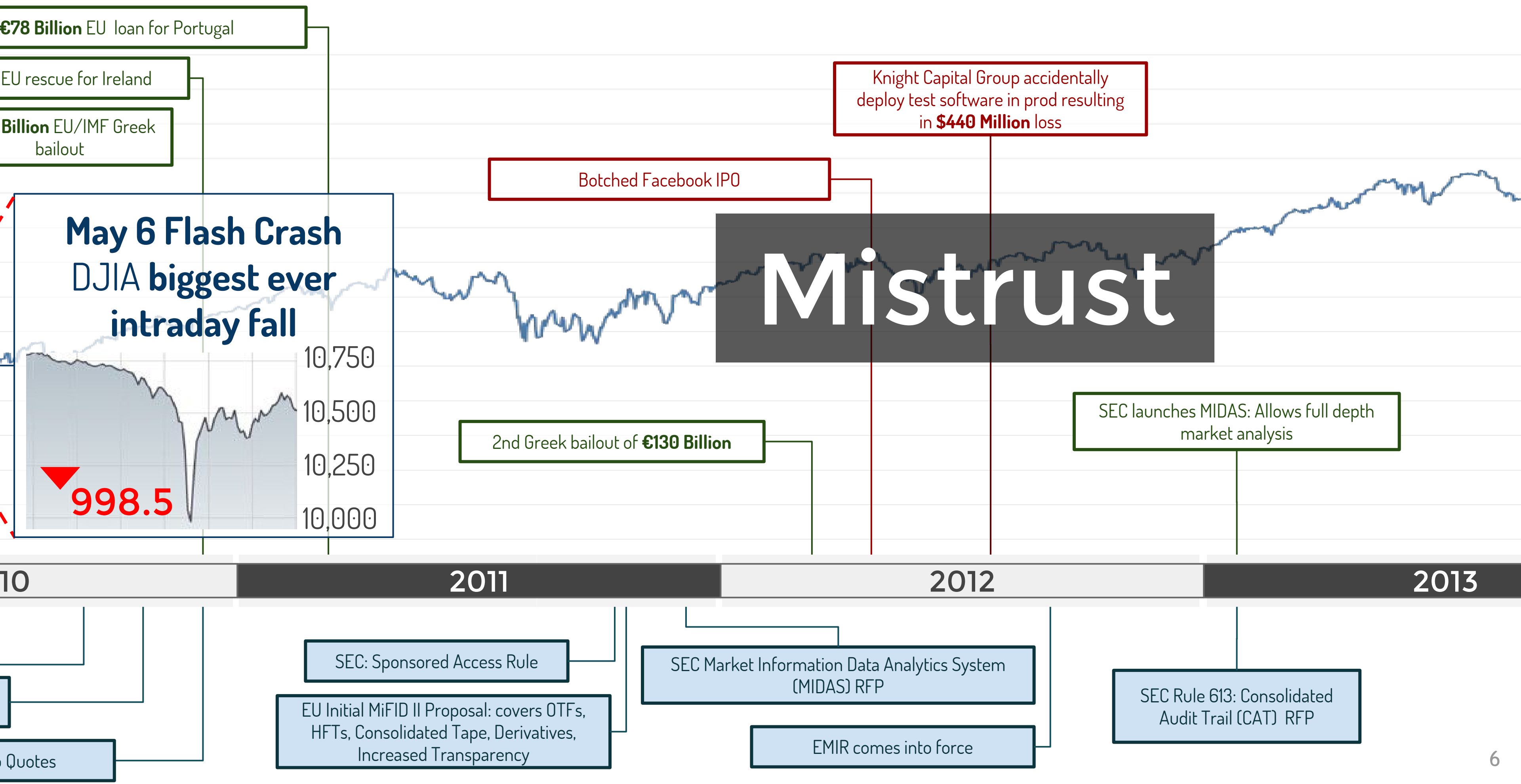
now director at clearpool.io

- regulations and change
- problems, people and software
- architecture and performance

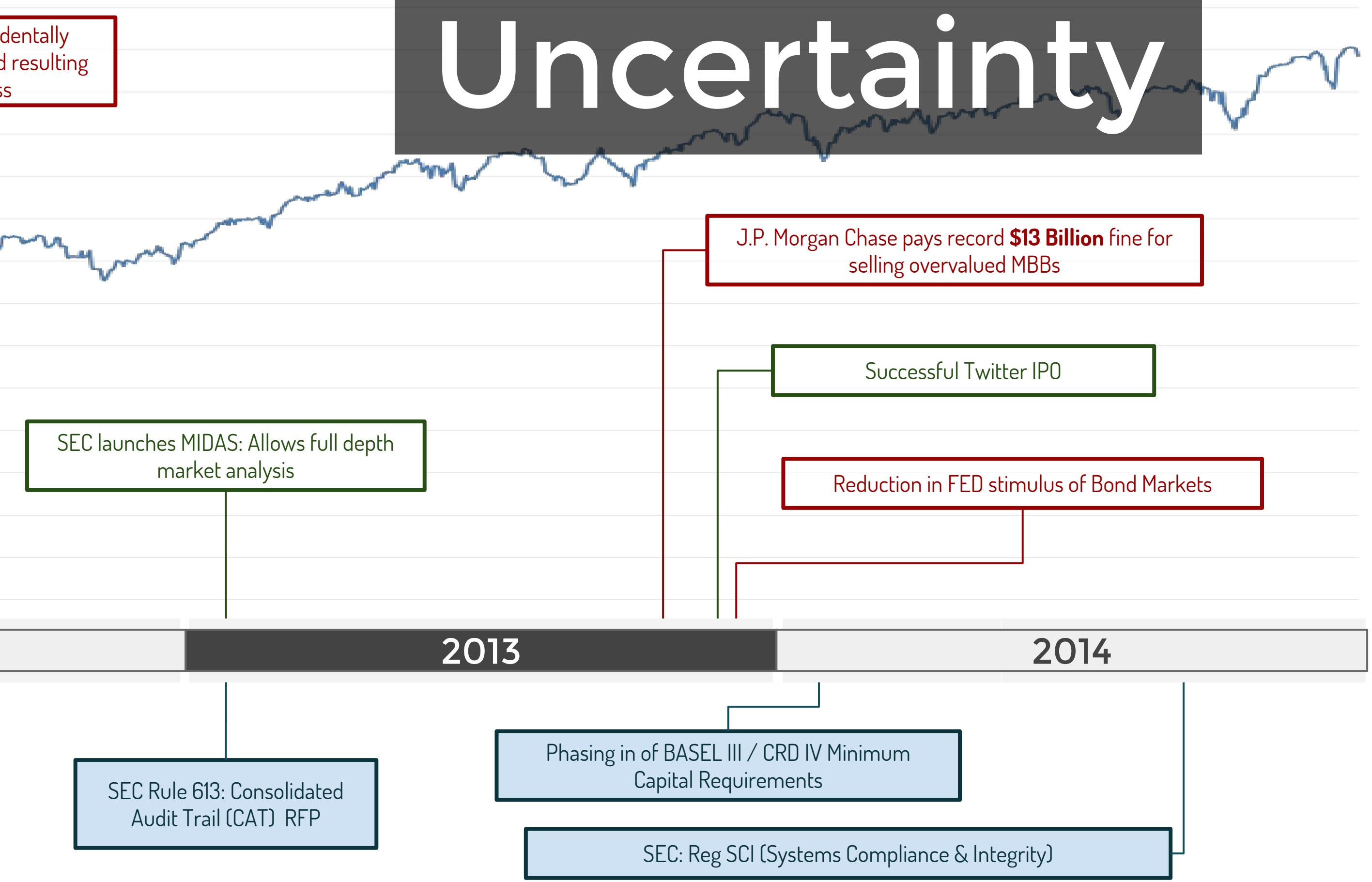

UK Nationalises Northern Rock
More losses for UBS & MBIA of **$21 Billion**
More Subprime losses for Bear Stearns, Citigroup & Merrill Lynch totalling **$34 Billion**
More Subprime losses for UBS & Morgan Stanley totalling **$19 Billion**
Subprime losses for UBS, Citigroup, Merrill Lynch, Deutsche Bank, Wachovia, Credit Suisse, Bank of America & Barclays totalling **$39 Billion**
Losses for HSBC of **$17 Billion**
Losses for UBS, Citigroup, Merrill Lynch & RBS now totals **$122 Billion**
19,000
18,000
17,000
16,000
15,000
14,000
13,000
12,000
11,000
10,000
9,000
8,000
7,000
6,000
5,000
14164.53
Dow Jones Industrial Average
DJIA
Panic
Short Seller "Bear Raid" on Citigroup
Bear Stearns liquidates two MBS funds
Subprime losses for HSBC of **$17 Billion**
FreddieMac stops buying subprime mortgages & MBS
2006
2007
2008
SEC: Regulation NMS: Order Protection (Trade-Through) Rule , Market Access Rule, Sub-Penny Rule, Market Data Rules
US: Emergency Economic Stabilization Act 2008 & UK Bank Rescue Package
SEC: Uptick Rule Repealed
Short Selling Bans



Losses for HSBC of $17 Billion
Losses for UBS, Citigroup, Merrill Lynch & RBS now totals $122 Billion
FannieMae & FreddieMac placed into Gov. Conservatorship
Lehman Brothers Bankrupt, Merrill Lynch bought by BoA for $50 billion
TARP financial stabilisation package enacted
€78 Billion EU loan for Portugal
€85 Billion EU rescue for Ireland
€110 Billion EU/IMF Greek bailout
Fear
DIJA falls 1874.19 pts in all-time worst weekly performance
6547.05
Greek bonds get Junk rating
May 6 Flash Crash
DJIA biggest ever intraday fall
10,750
10,500
10,250
10,000
998.5
Bailouts for Big Banks
2008
2009
2010
tabilization Act
e Package
Short Selling Bans
Dodd-Frank Reform Act Passed (inc. Volcker Rule)
SEC: Reg SHO - Rule 201: Alternative Uptick Rule—Short Sale-Related Circuit Breaker
Original BASEL III Proposal
SEC: Ban on Stub Quotes


€78 Billion EU loan for Portugal
EU rescue for Ireland
Billion EU/IMF Greek bailout
May 6 Flash Crash
DJIA biggest ever intraday fall
10,750
10,500
10,250
10,000
998.5
Botched Facebook IPO
Knight Capital Group accidentally deploy test software in prod resulting in $440 Million loss
Mistrust
2nd Greek bailout of €130 Billion
SEC launches MIDAS: Allows full depth market analysis
10
2011
2012
2013
SEC: Sponsored Access Rule
EU Initial MiFID II Proposal: covers OTFs, HFTs, Consolidated Tape, Derivatives, Increased Transparency
SEC Market Information Data Analytics System (MIDAS) RFP
EMIR comes into force
SEC Rule 613: Consolidated Audit Trail (CAT) RFP
Quotes

Uncertainty



"Too Big to Fail Banks"

Market Volatility

Insufficient Oversight

Unpopular Gov. Bailouts

Mistrust of Technology

Evolving Regulatory Landscape

Regulations are currently seen as the best way to protect the markets and their participants from themselves

But Regulations are a Moving Target

Regulations Change

for many reasons but ultimately they change

stuff happens and regulations are often seen as the answer

regulations create loop-holes that need plugged

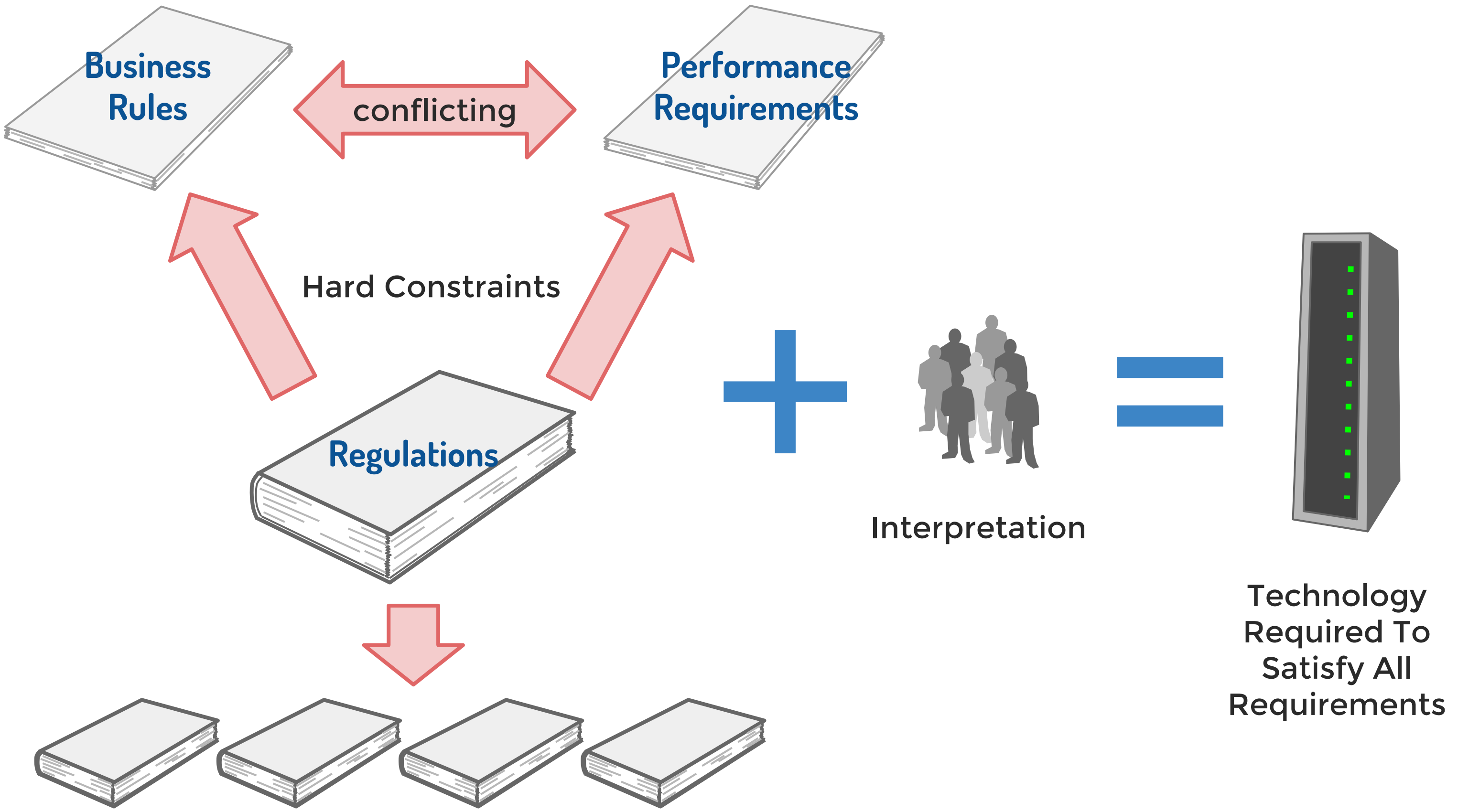
regulations create industries that themselves need regulated

There are often Hard Constraints

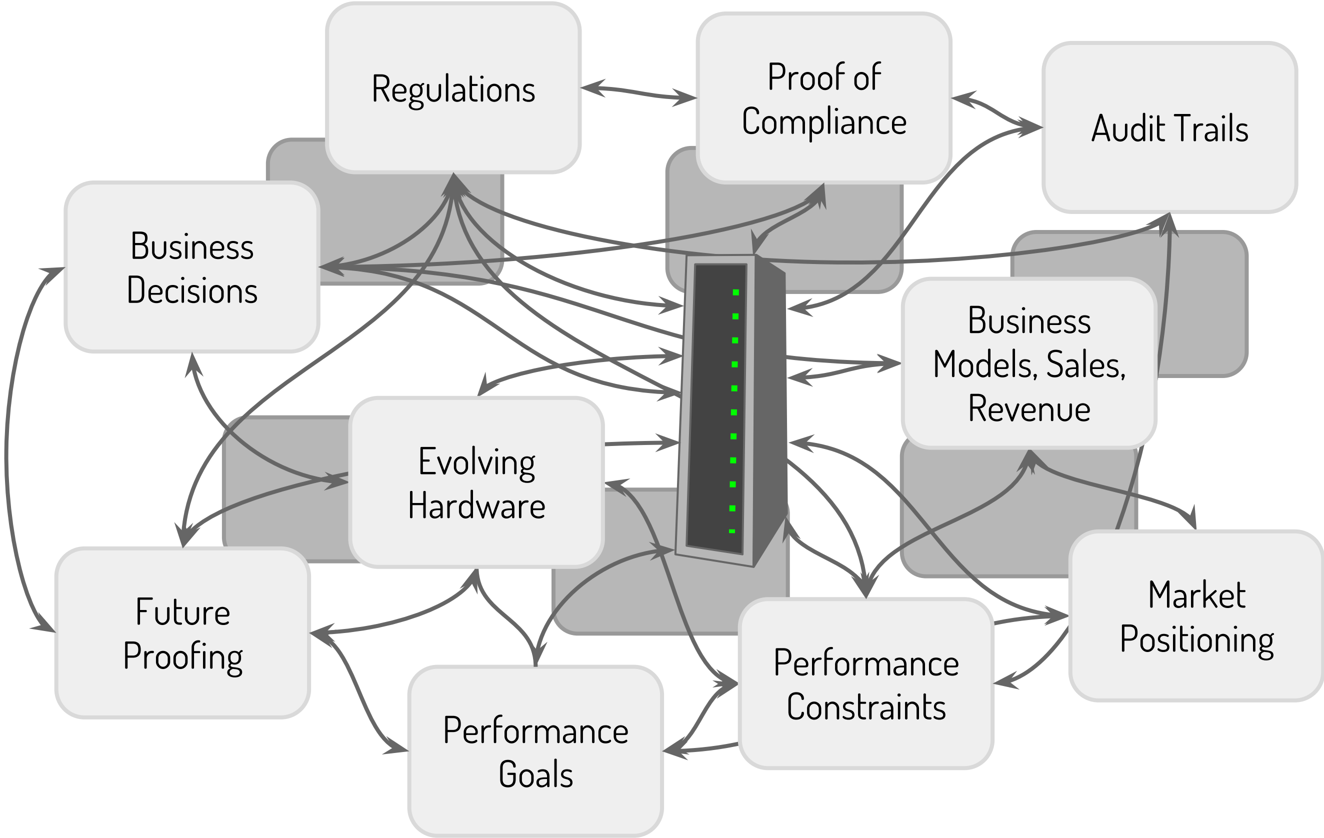
- minimum throughput?
- availability?
- disaster recovery?
- average latency?
- worst case latency?
- proof of compliance?
- audit trails?
- accuracy of data capture?

many constraints driven by regulations

Let’s simplify this…



Addressing Difficult Problems



"We fail more often because we ***solve the wrong problem*** *than because we get the* ***wrong solution to the right problem****"*

— Ackoff 1974

How can we classify problems?

Rittel & Webber 1973, Ackoff 1974, Roth & Senge 1996, Hancock 2004, Ritchey 2013

Tame Problems

- may be simple or highly complex
- definitive stopping point
- consensus on how to proceed

- can be broken down into parts and solved
- solutions can be determined to be successful ...or not

Messes

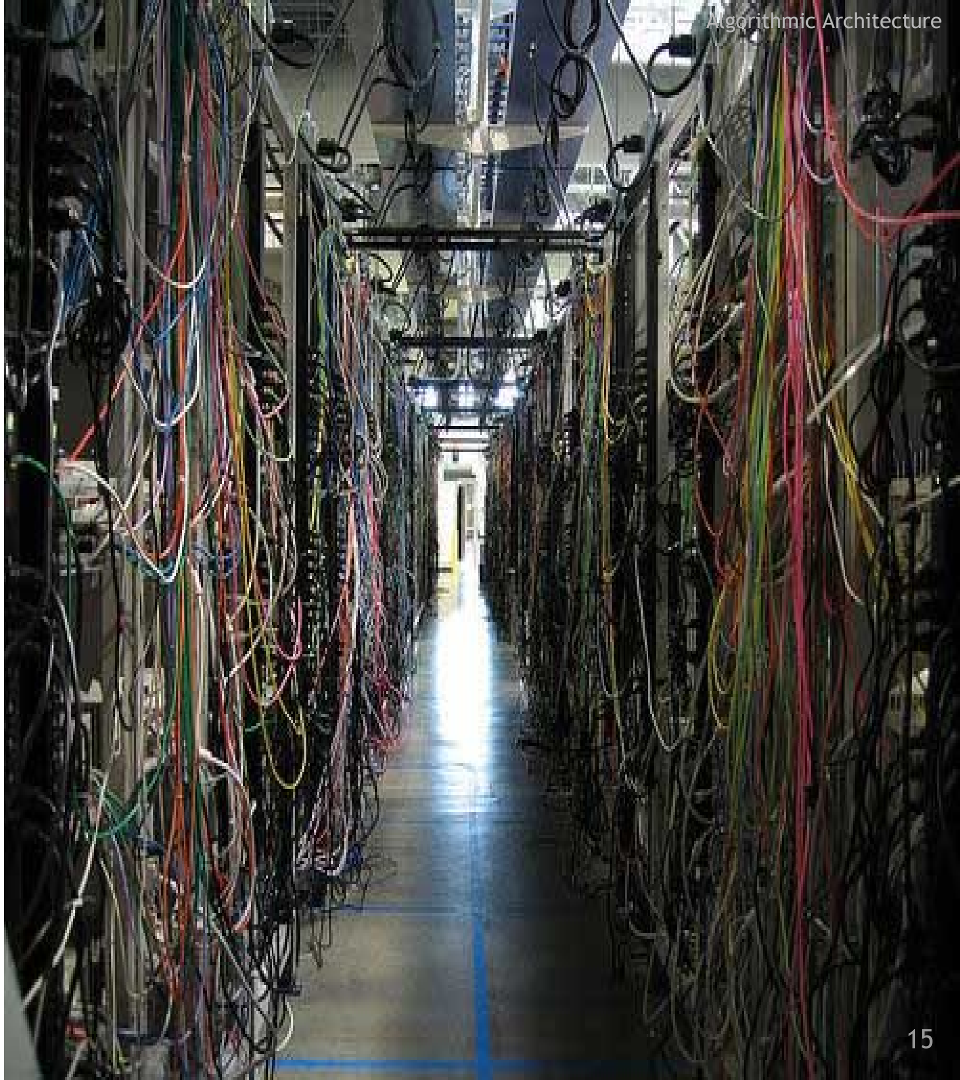
Organised complexity

- clusters of interrelated or interdependent problems

Systems of problems

- problems that cannot be solved in relative isolation from one another

Messes are puzzles – we don't solve them instead we **resolve their complexities**



Tidy up that mess!!!

- not sufficient to just break the system into parts and fix components
- instead look for **patterns** of interactions between parts
- beware of identifying a mess as a tame problem—the evolving mess can be even more difficult to deal with
- **interactive complexity**—what can go wrong?
- **coupling**—the degree to which we cannot stop an impending disaster once it starts

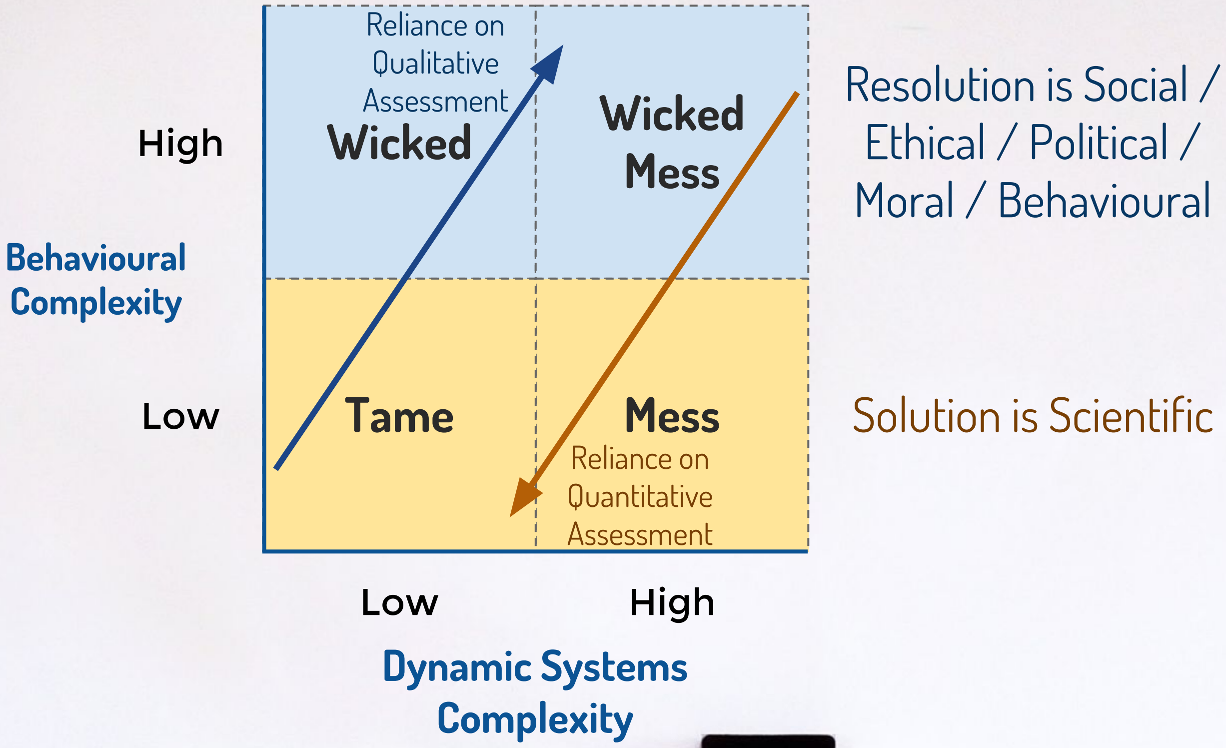
Refactoring vs Bugfixing?

* Conflicting **social** ethics and beliefs
* Smart, informed people **disagree**
* **Divergent** problems with no promise of a solution
* **Evolving** set of **Interlocking** Issues and Constraints
* Many Stakeholders
* Constraints **change over Time**

Know your demons…

- No definitive Problem == No definitive Solution
- Cannot be solved by a Linear or "Waterfall" process
- **Studying** followed by **Taming** does not work
- No stopping rules
- Finished when we **Exhaust Resources**
- Solutions not Right or Wrong but **Better** or **Worse**
- Poor choices create more Wicked Problems

How we deal with problem complexity



Let's consider the question of Healthy Markets

The markets involve people

The markets involve systems

Lots of People and Lots of Systems

Characteristics of a Healthy Market?

Volatility?

Data Access?

Transparency?

Liquidity?

What represents "good liquidity"?

- Tighter Spreads?
- Order Book Depth?
- What about "phantom" Orders?

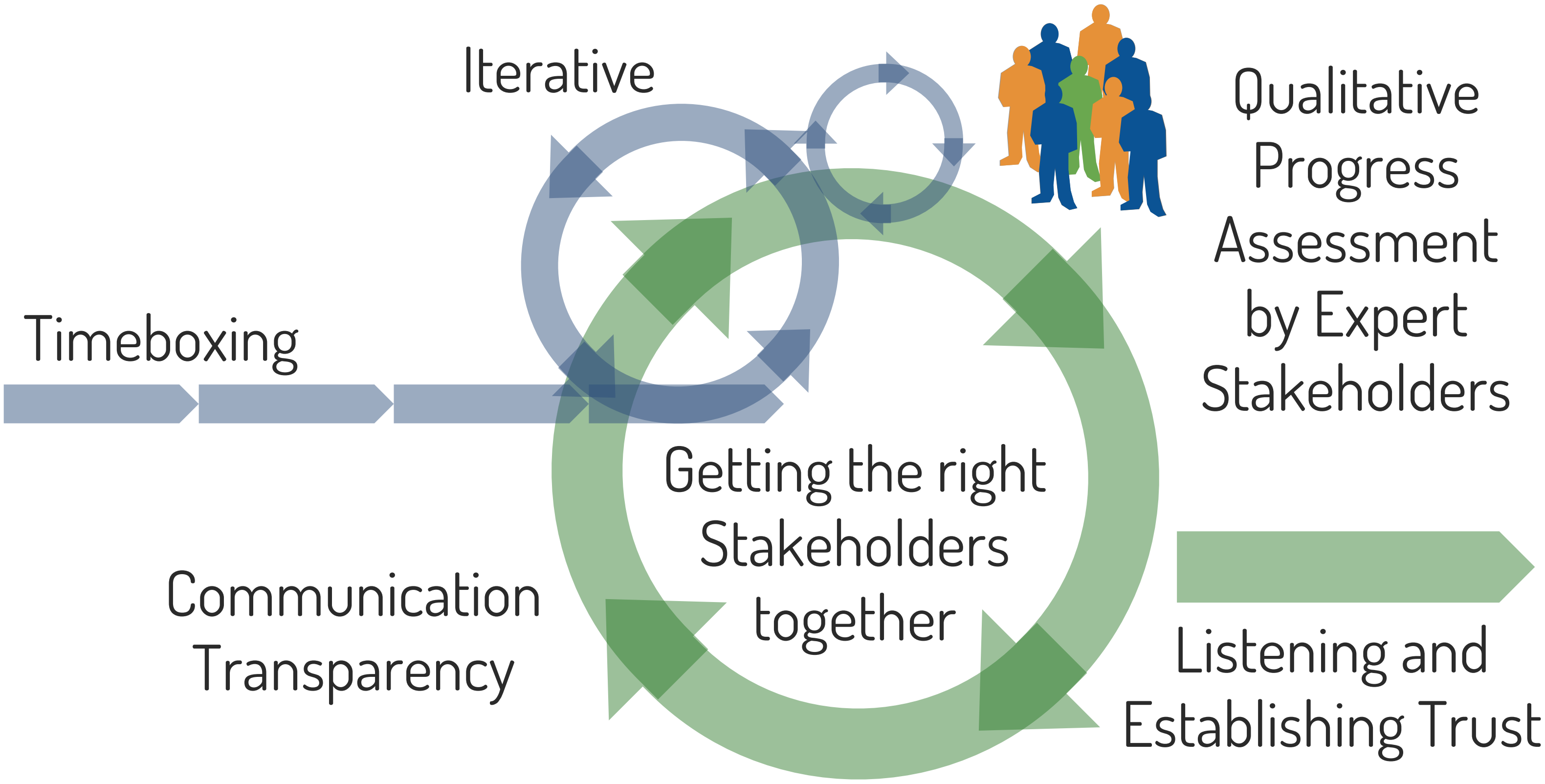
High Behavioural Complexity

High Dynamic System Complexity

Wicked Mess

Regulations Developed to Promote Healthy Markets

Approaches to Wicked Problems



Sounds a lot like Agile Development?

Agile and We're Done?

Remember our focus is on Architecture in the context of Wicked Messes

What do we mean by Architecture?

- The product of Design and Implementation – what you see when you step back and look at your system
- Encoded in the Architecture are the choices made and compromises reached

Whose choices?

Whose compromises?

Another view on Architecture

Marketecture vs Tarchitecture?

Marketecture: Anything that is concerned with how revenue is generated for a product or how it is marketed as working, or how it is sold

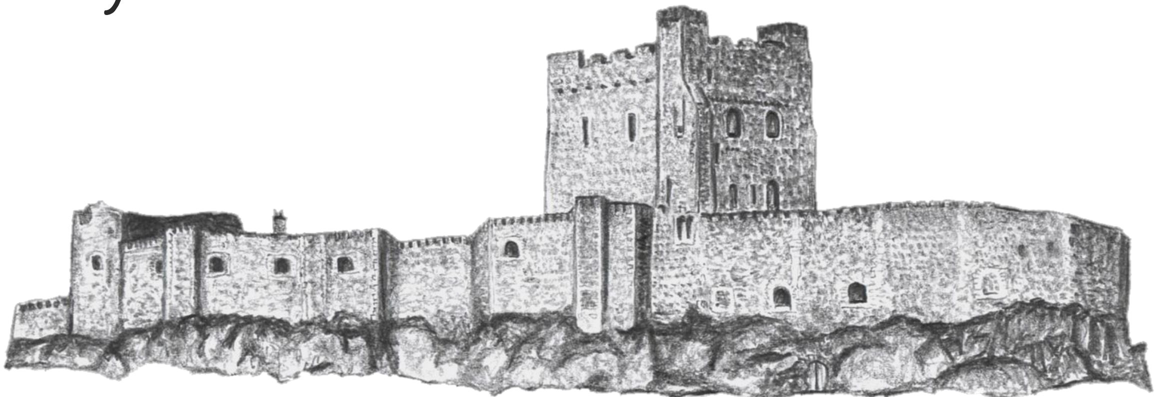
Marketecture **impacts** Tarchitecture

Dangers in evolution

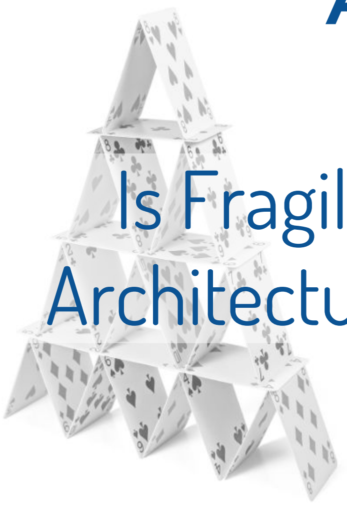
- Marketecture is often driven by decisions that have **no regard for the technical impact**
- Stakeholders change
- Goal posts move
- "Power **without** responsibility"
- Poor choices baked in early
- What's most important?

Architecture General Truths

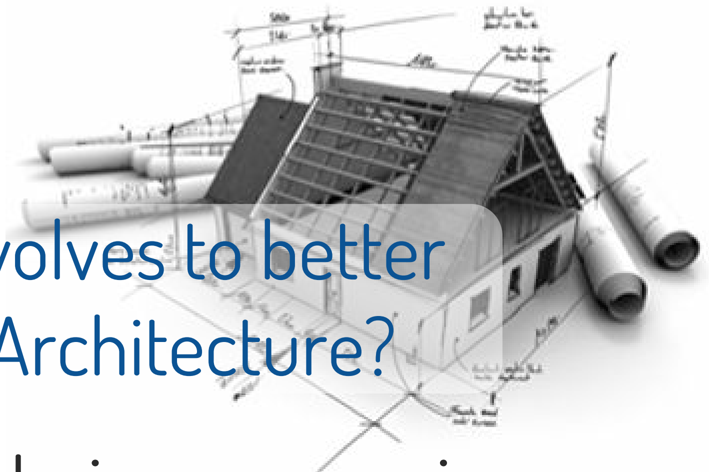
- Is often an observed **sketch** of the system
- Actual architecture exists based on the **source code**
- Pinpointing which aspects contribute to any characteristic of the system can be difficult
- Changing it is usually hard



Agile Architecture



Is Fragile Architecture?



Evolves to better Architecture?

- ➤ Hard choices early so later choices are easier
- ➤ Evolving to an appropriate architecture
- ➤ Deferring choices to last responsible moment
- ➤ Natural calcification along the way

Agile Architecture is a good starting point—evolving to an appropriate architecture

Can we do better?

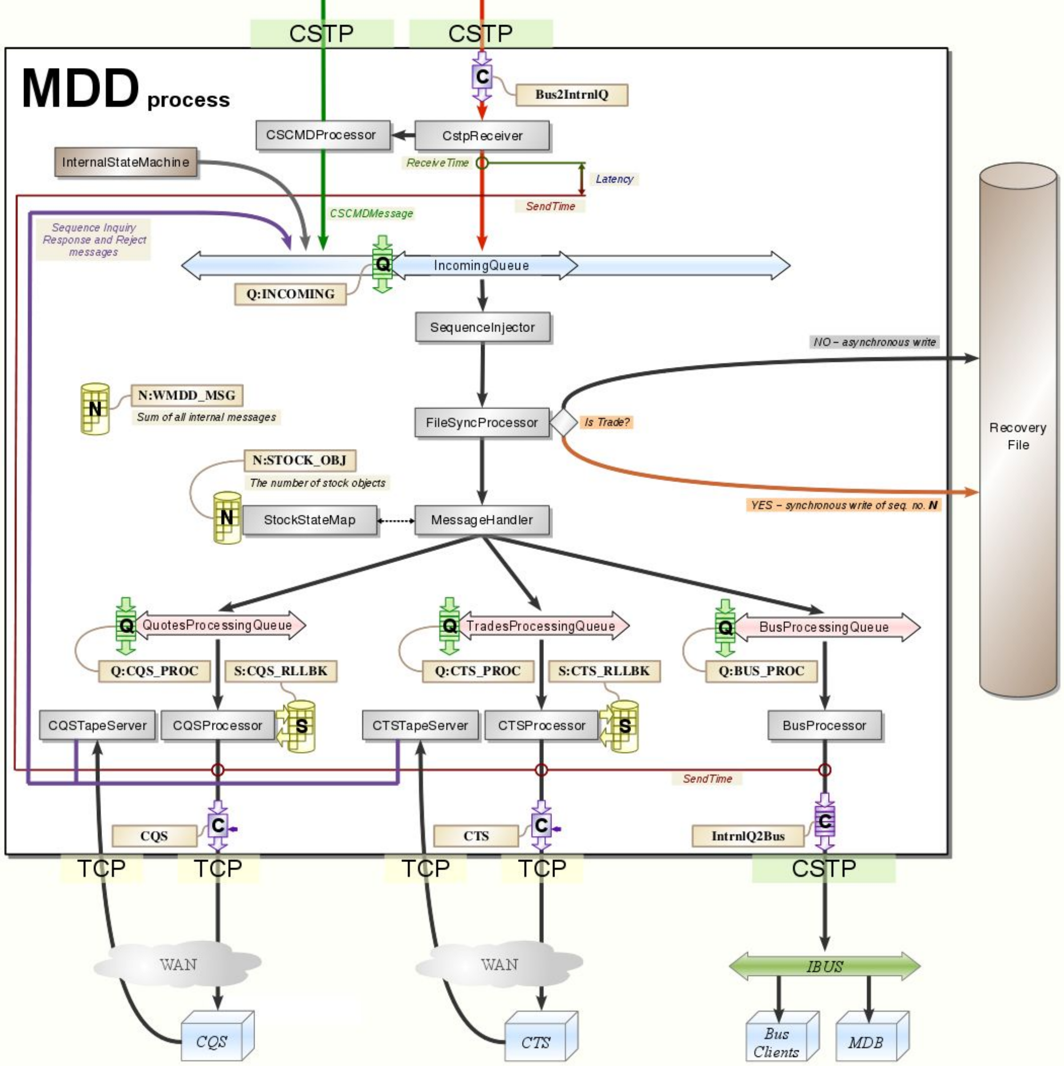
Let's look at a real world example as a starting point

Following the Flash Crash the SEC launched an investigation into the causes

The SEC were presented with architectural overviews of how the systems involved behaved, and how they were evolved

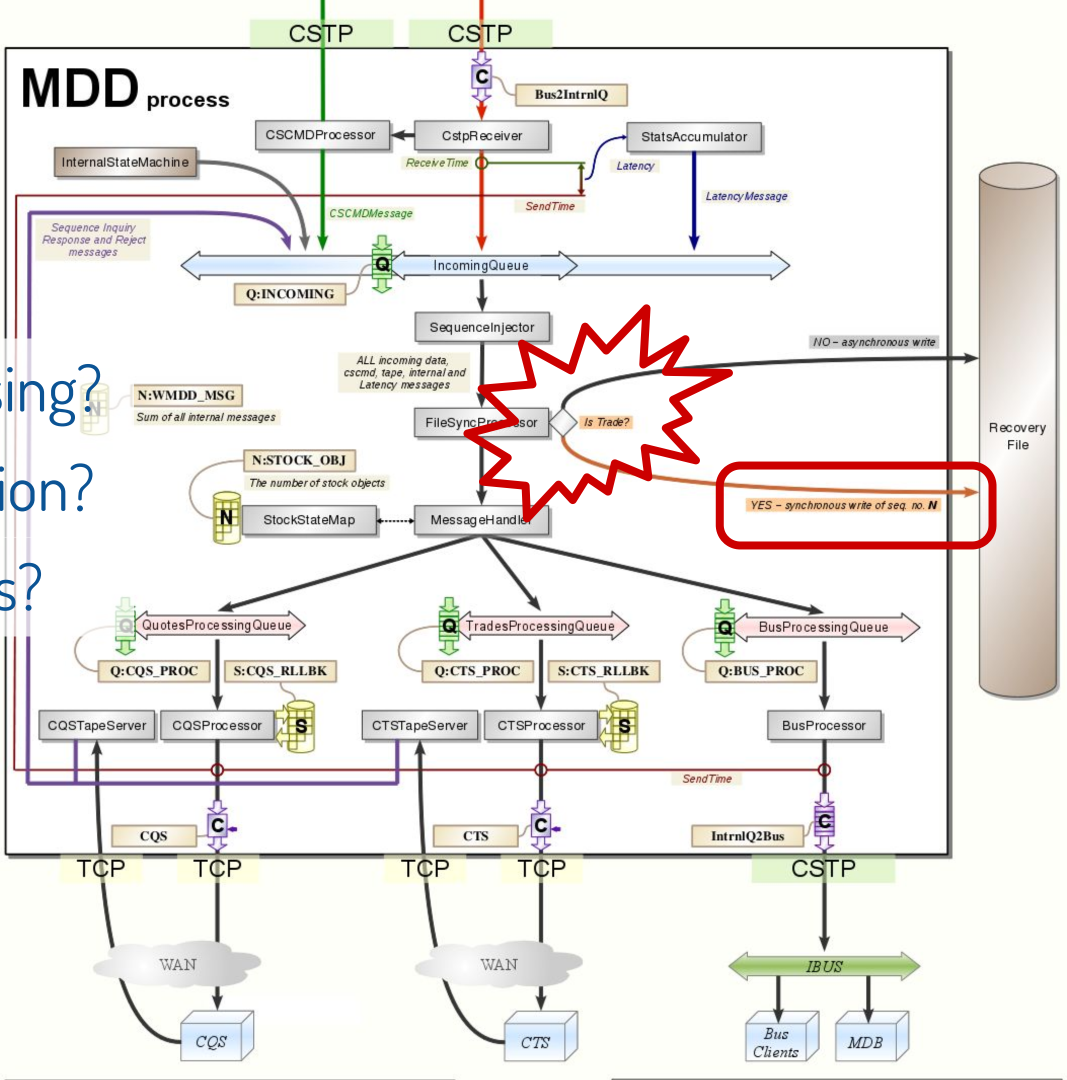
Their focus was on
Market Data Publication
Slow and delayed quoting
was experienced during the
Flash Crash

What can we tell from looking at this picture?



1

Data flow
Networking
Queuing
Decisions
Processors
Data stores

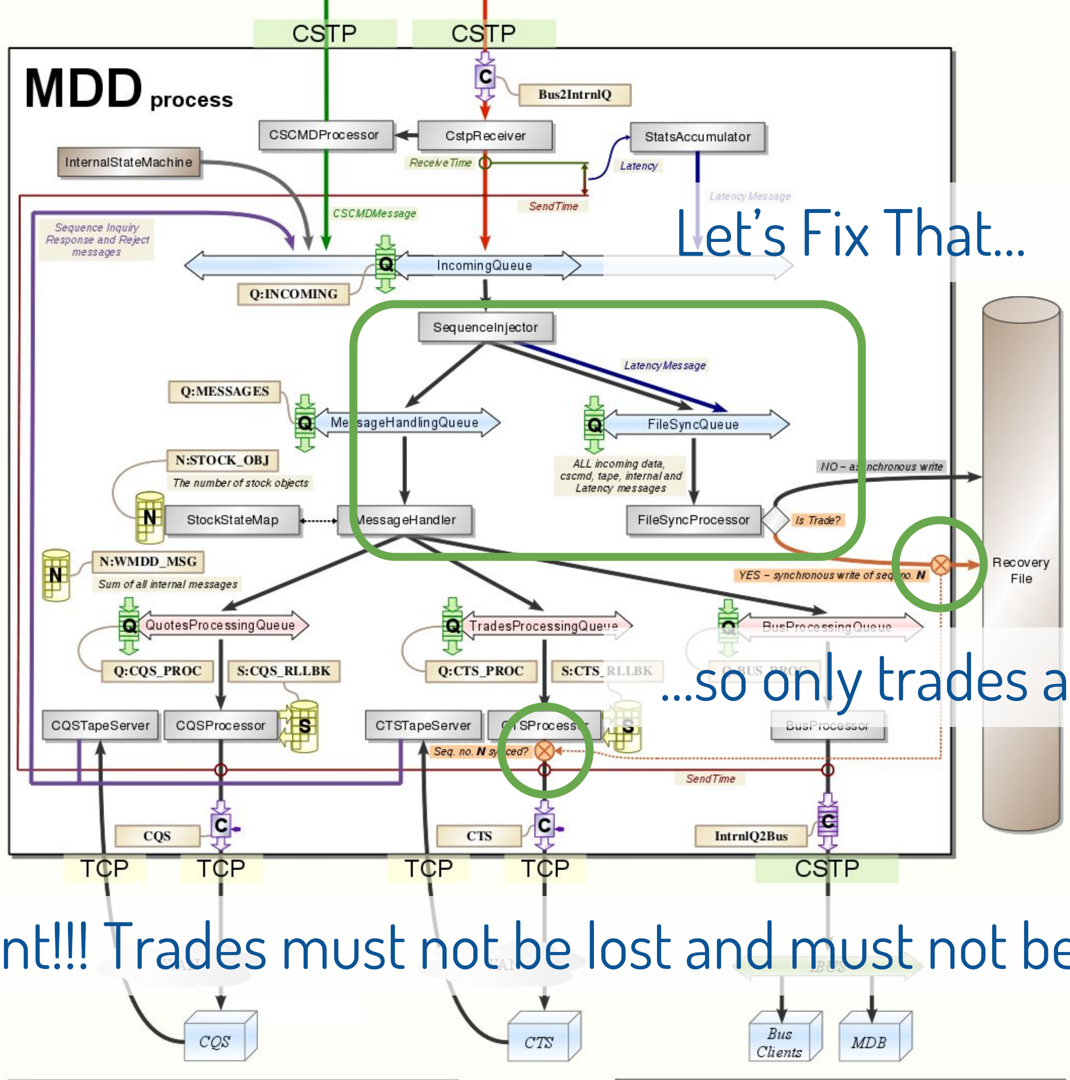


Message Passing?

Synchronisation?

Bottlenecks?

2



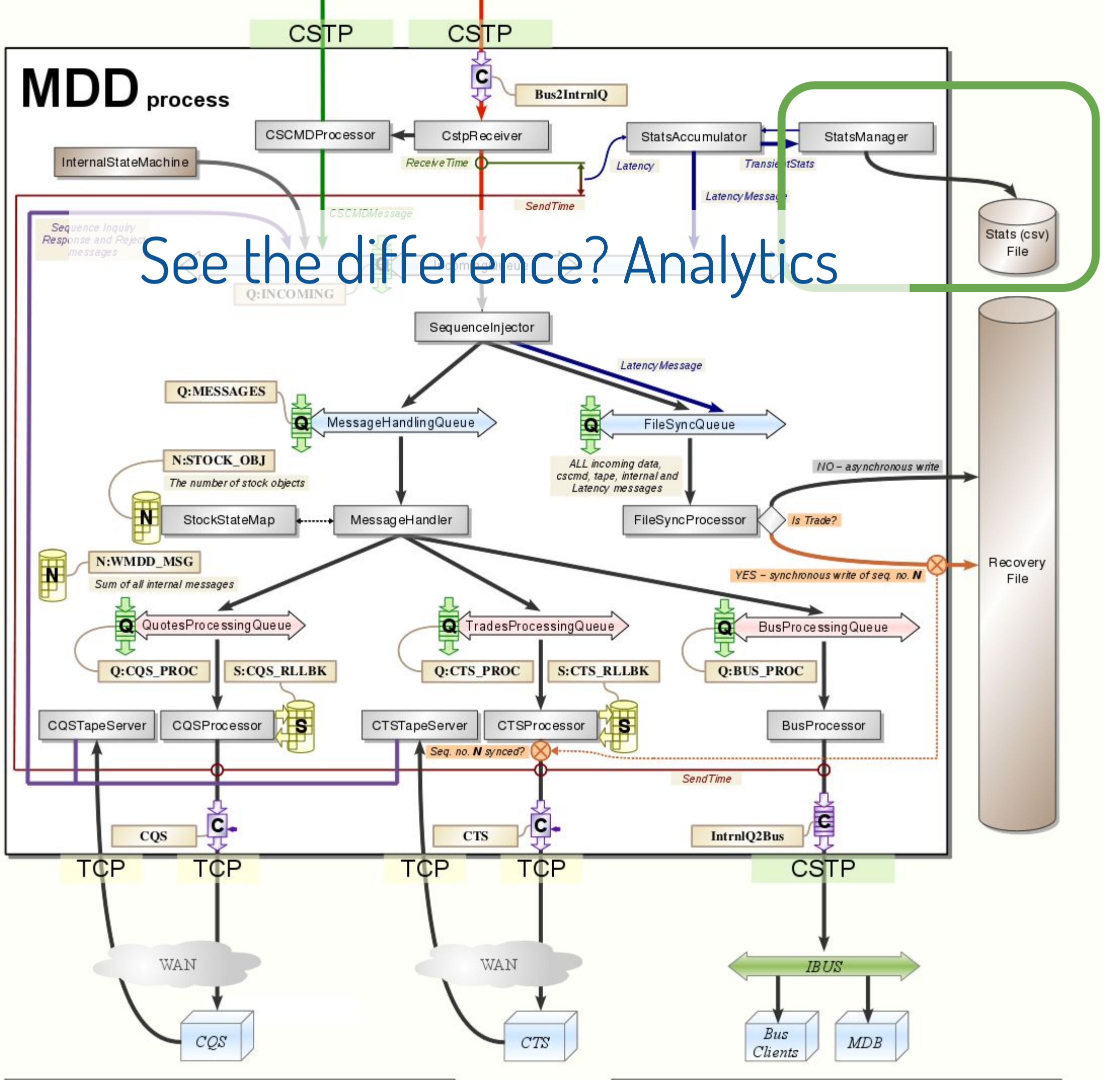
Let's Fix That...

3

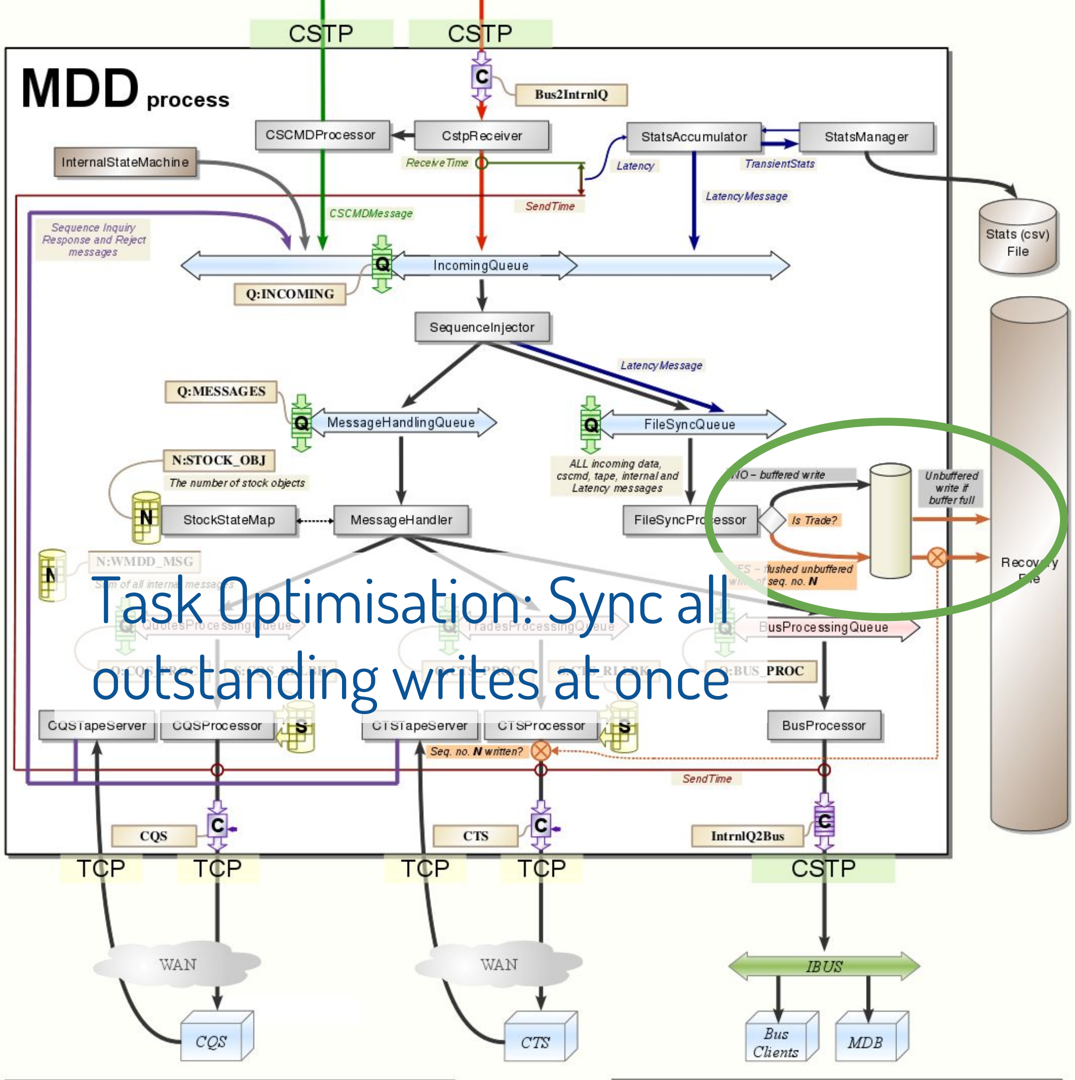
...so only trades are affected

Requirement!!! Trades must not be lost and must not be duplicated

See the difference? Analytics



4



5

Task Optimisation: Sync all outstanding writes at once

There are a lot of things we **cannot** tell from looking at the diagrams

What about...?

How are stale quotes handled during a recovery?

When and why are zero quotes published?

Are the recovery requirements reasonable?

Which version was in production at the time?

Did the system behave correctly?

Is there information to make that determination?

How was memory managed?

How many cores did deployment machines have?

Details, details, details...

Reasons why...?

Risk Averse Business

Correctness the highest priority, then performance

Ultimate priority was performance

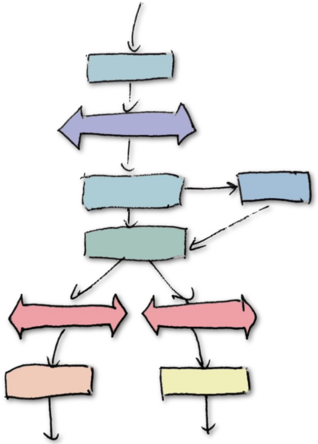
Worst case performance requirements

Architecture should evolve to improve performance

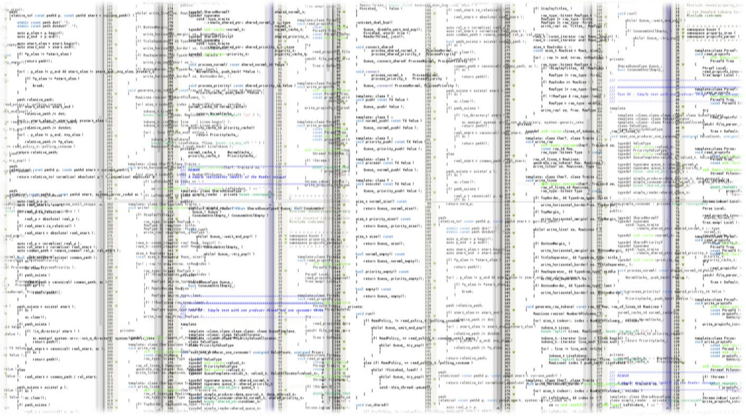
There were 2 versions live in production

A Story…
Not the Whole Story

Nice diagrams typically do not reflect the reality of a code-base



≠

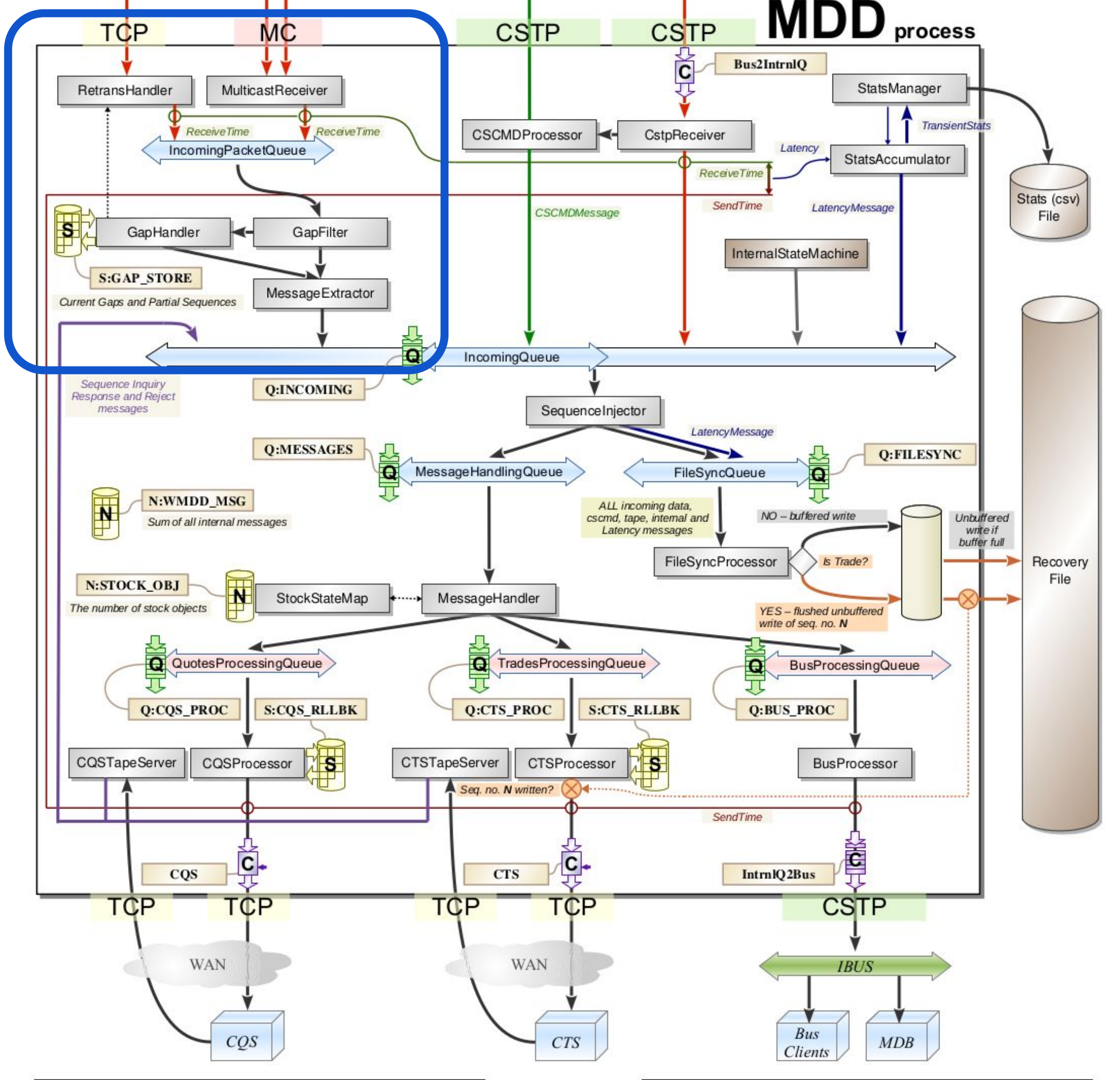


It would be nice if it did

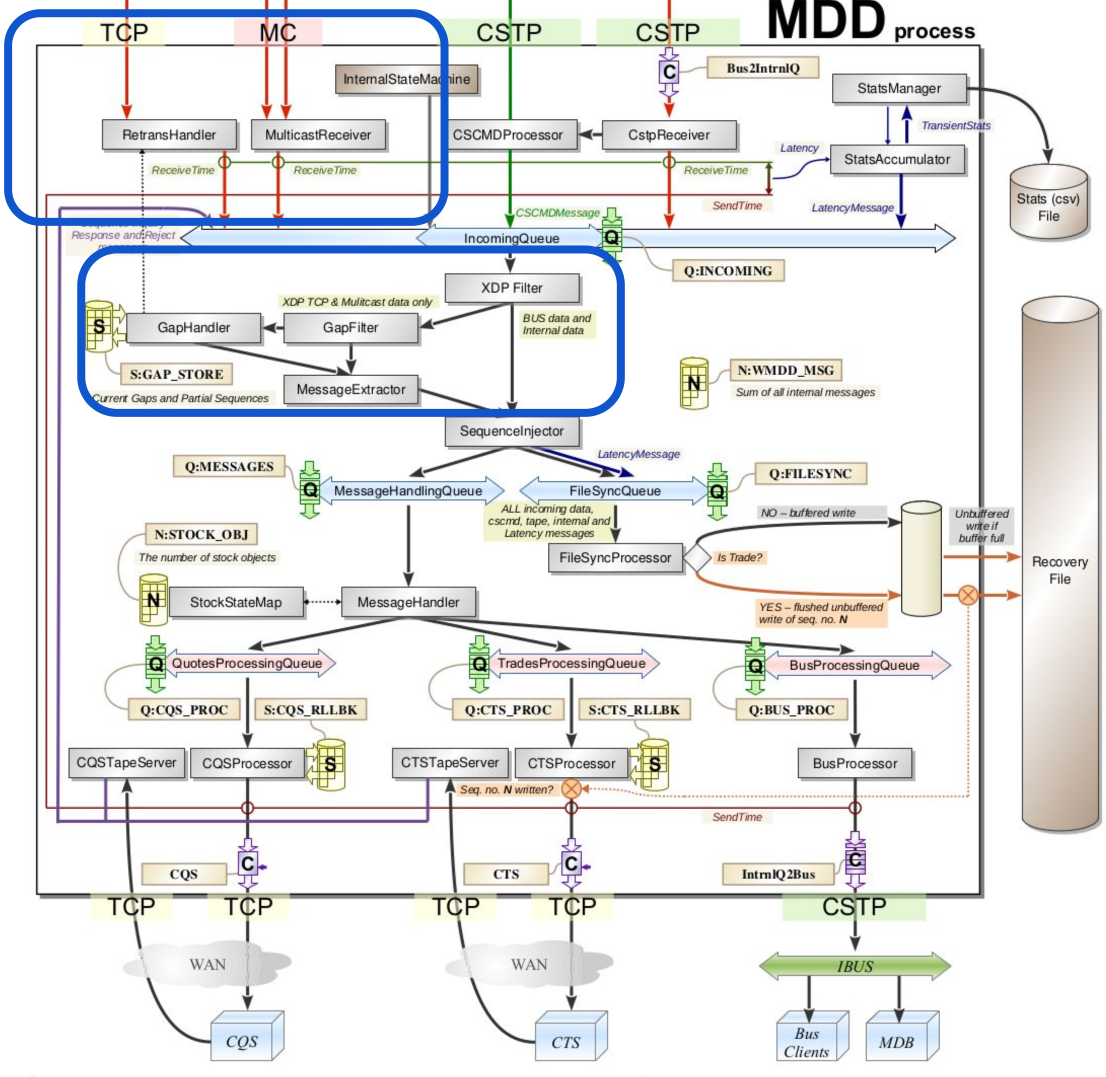
Some things we can conclude

- Performance improved by doing the right thing
- Not by optimising existing behaviour
- Local optimisation only done when solution good enough

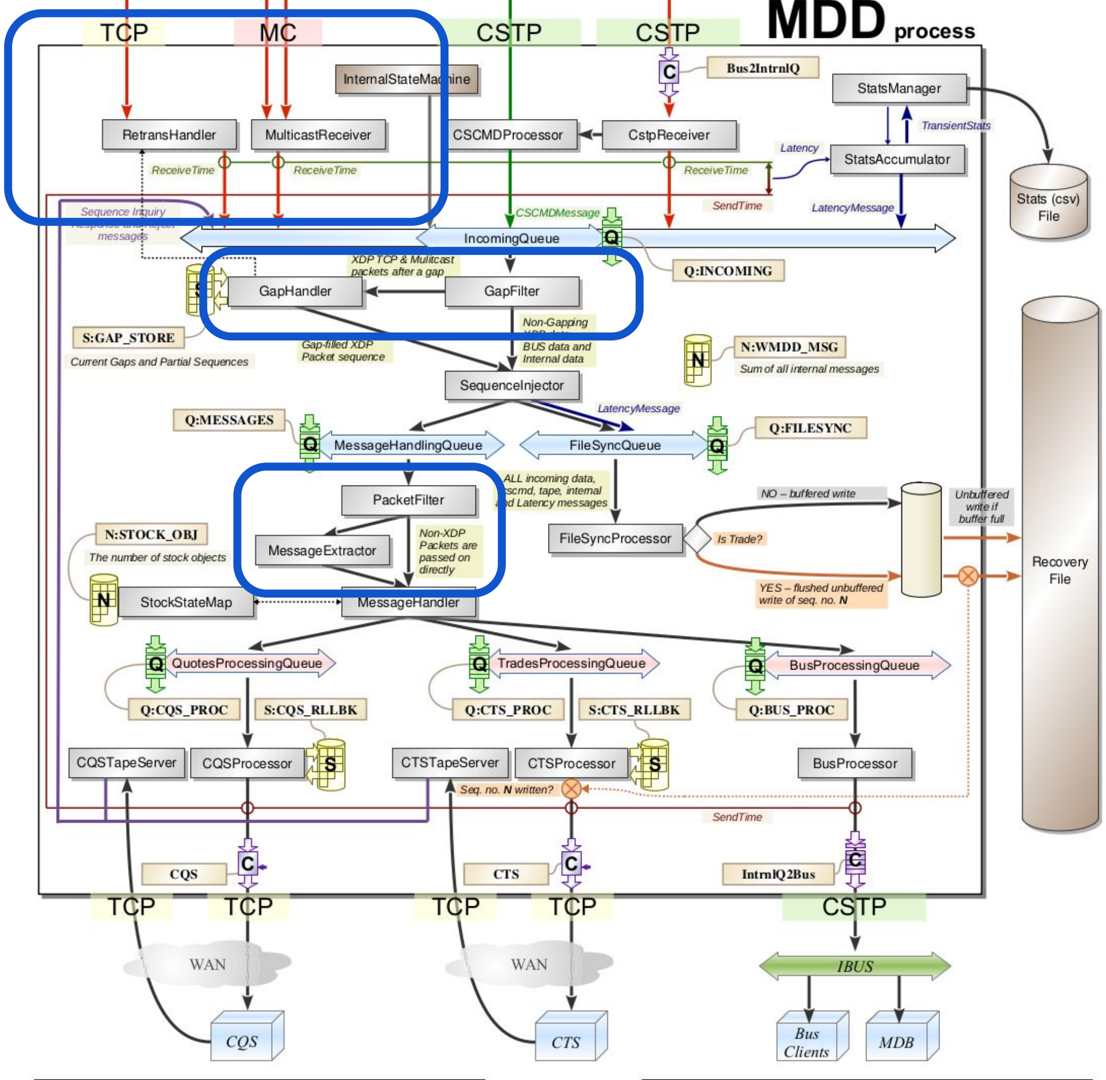
Let's look at some possible future systems that all do the same thing...


MDD process
TCP
MC
CSTP
CSTP
Bus2IntrnlQ
RetransHandler
MulticastReceiver
StatsManager
ReceiveTime
ReceiveTime
TransientStats
IncomingPacketQueue
CSCMDProcessor
CstpReceiver
Latency
StatsAccumulator
ReceiveTime
SendTime
Stats (csv) File
CSCMDMessage
LatencyMessage
GapHandler
GapFilter
InternalStateMachine
S:GAP_STORE
Current Gaps and Partial Sequences
MessageExtractor
IncomingQueue
Q:INCOMING
Sequence Inquiry Response and Reject messages
SequenceInjector
LatencyMessage
Q:MESSAGES
MessageHandlingQueue
FileSyncQueue
Q:FILESYNC
N:WMDD_MSG
Sum of all internal messages
ALL incoming data, cscmd, tape, internal and Latency messages
NO – buffered write
Unbuffered write if buffer full
FileSyncProcessor
Is Trade?
Recovery File
N:STOCK_OBJ
The number of stock objects
StockStateMap
MessageHandler
YES – flushed unbuffered write of seq. no. N
QuotesProcessingQueue
TradesProcessingQueue
BusProcessingQueue
Q:CQS_PROC
S:CQS_RLLBK
Q:CTS_PROC
S:CTS_RLLBK
Q:BUS_PROC
CQSTapeServer
CQSProcessor
CTSTapeServer
CTSProcessor
BusProcessor
Seq. no. N written?
SendTime
CQS
CTS
IntrnlQ2Bus
TCP
TCP
TCP
TCP
CSTP
WAN
WAN
IBUS
CQS
CTS
Bus Clients
MDB

6


MDD process
TCP
MC
CSTP
CSTP
InternalStateMachine
Bus2IntrnlQ
StatsManager
RetransHandler
MulticastReceiver
CSCMDProcessor
CstpReceiver
TransientStats
Latency
StatsAccumulator
Stats (csv) File
ReceiveTime
ReceiveTime
ReceiveTime
SendTime
LatencyMessage
CSCMDMessage
Response and Reject
IncomingQueue
Q:INCOMING
XDP Filter
XDP TCP & Mulitcast data only
GapHandler
GapFilter
BUS data and Internal data
S:GAP_STORE
MessageExtractor
Current Gaps and Partial Sequences
N:WMDD_MSG
Sum of all internal messages
SequenceInjector
LatencyMessage
Q:MESSAGES
MessageHandlingQueue
FileSyncQueue
Q:FILESYNC
ALL incoming data, cscmd, tape, internal and Latency messages
NO – buffered write
Unbuffered write if buffer full
N:STOCK_OBJ
The number of stock objects
FileSyncProcessor
Is Trade?
Recovery File
StockStateMap
MessageHandler
YES – flushed unbuffered write of seq. no. N
QuotesProcessingQueue
TradesProcessingQueue
BusProcessingQueue
Q:CQS_PROC
S:CQS_RLLBK
Q:CTS_PROC
S:CTS_RLLBK
Q:BUS_PROC
CQSTapeServer
CQSProcessor
CTSTapeServer
CTSProcessor
BusProcessor
Seq. no. N written?
SendTime
CQS
CTS
IntrnlQ2Bus
TCP
TCP
TCP
TCP
CSTP
WAN
WAN
IBUS
CQS
CTS
Bus Clients
MDB

7


MDD process
TCP
MC
CSTP
CSTP
InternalStateMachine
Bus2IntrnlQ
StatsManager
RetransHandler
MulticastReceiver
CSCMDProcessor
CstpReceiver
TransientStats
Latency
StatsAccumulator
ReceiveTime
ReceiveTime
ReceiveTime
SendTime
LatencyMessage
Stats (csv) File
Sequence Inquiry
Response and reject messages
CSCMDMessage
IncomingQueue
Q:INCOMING
XDP TCP & Mulitcast packets after a gap
GapHandler
GapFilter
S:GAP_STORE
Current Gaps and Partial Sequences
Gap-filled XDP Packet sequence
Non-Gapping XDP data
BUS data and Internal data
N:WMDD_MSG
Sum of all internal messages
SequenceInjector
LatencyMessage
Q:MESSAGES
MessageHandlingQueue
FileSyncQueue
Q:FILESYNC
ALL incoming data, cscmd, tape, internal and Latency messages
NO – buffered write
Unbuffered write if buffer full
PacketFilter
N:STOCK_OBJ
The number of stock objects
MessageExtractor
Non-XDP Packets are passed on directly
FileSyncProcessor
Is Trade?
YES – flushed unbuffered write of seq. no. N
Recovery File
StockStateMap
MessageHandler
QuotesProcessingQueue
TradesProcessingQueue
BusProcessingQueue
Q:CQS_PROC
S:CQS_RLLBK
Q:CTS_PROC
S:CTS_RLLBK
Q:BUS_PROC
CQSTapeServer
CQSProcessor
CTSTapeServer
CTSProcessor
BusProcessor
Seq. no. N written?
SendTime
CQS
CTS
IntrnlQ2Bus
TCP
TCP
TCP
TCP
CSTP
WAN
WAN
IBUS
CQS
CTS
Bus Clients
MDB

8

The same thing in a different way with different trade-offs: Performance trade-offs

Improving Performance

Do the current thing better/quicker		Achieve the same thing in a different way
Task Optimisation Approach		Algorithmic Optimisation Approach
Bubblesort $O(n^2)$	*Sorting*	Timsort $O(n \log n)$
DFT $O(n^2)$	*Frequency Analysis*	FFT $O(n \log n)$

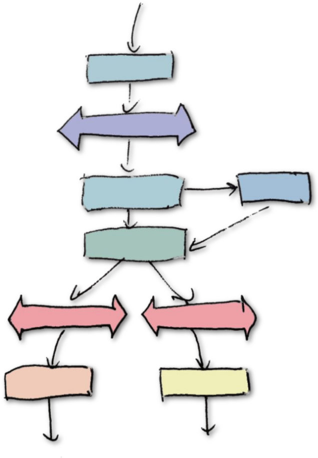
Prefer to optimise at the highest level possible

The fastest way to do something is not do it at all

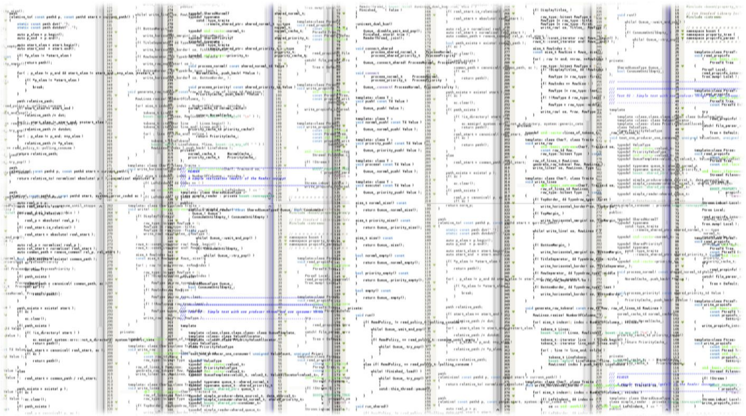
Environmental Influences

- Architecture for wicked problems typically a "**mess**"
- Many stakeholders and evolving problem domain over time adds "**wickedness**"
- Decomposing and understanding interactions difficult
- Such architecture, good or bad, is often hard to reason about in a way that maps directly to code
- Favours **Task Optimisation**

We want to reason about this…



But we can only see this…



What we really want is an Architecture that

- favours algorithmic optimisation
- has a clear mapping to code
- allows an optimal solution
- is adaptive to a changing environment

an "Algorithmic Architecture"

Relies on being able to decompose the Architecture into discrete elements treating them as Building Blocks

We Achieve This By

- Exposing a Vocabulary *that can map to code and is*
- Decomposable
- Composable
- Independently Orderable
- Compactible
- Substitutable

1 Expose a vocabulary

the first step in moving towards an algorithmic architecture is to identify a vocabulary suitable for the domain

- implies decomposability
- implies extensibility

Must be a **common** vocabulary

A common vocabulary's primary concern is not ensuring the best use in the description of a possible solution—rather it is focused on ensuring that all stakeholders can communicate sufficiently their position within it—it is **shared**

Must be **domain specific**

The vocabulary must support natural domain specific terms as understood by most stakeholders—it is not sufficient to simply adopt a general vocabulary based on general design patterns (but they help)

Identify **concepts**

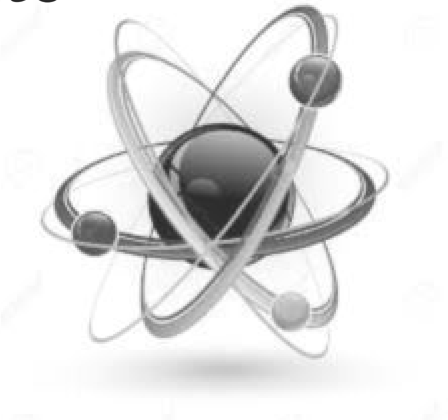
Focus on identifying **concepts** over specific realisations. Refinement to more concrete terms is best reserved for supporting substitutable elements in an architecture

Vocabulary Checklist

- must add in clarity of communication
- should have consensus on basic meanings
- does not need to be complete
- but should be sufficient to model basic systems
- may capture concepts at **different** levels in a system
- should be possible to describe a system
- vocabularies can grow and evolve

2 Decomposable

it should be possible to decompose the architecture into vocabulary elements that communicate the intent of the system

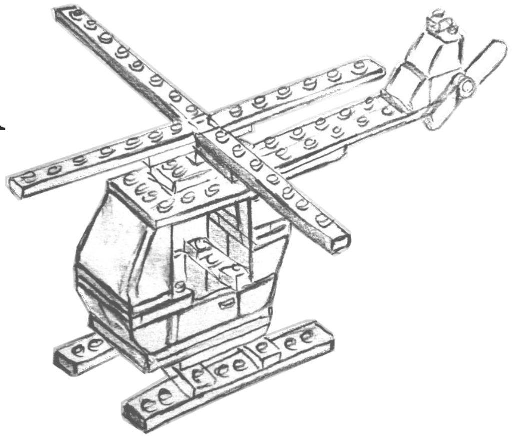


➤ implies partitioning interfaces

3 Composable

composable components can be assembled together to complete more complex tasks

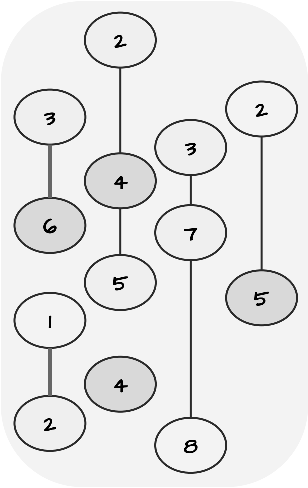
- implies common approach to communication



4 Independently orderable

it should be possible to re-order components of the architecture that do not have an ordering relationship

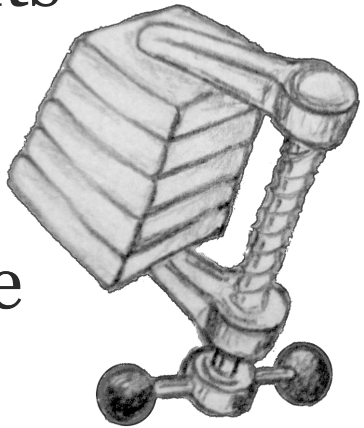
➤ implies loose coupling



5 Compactible

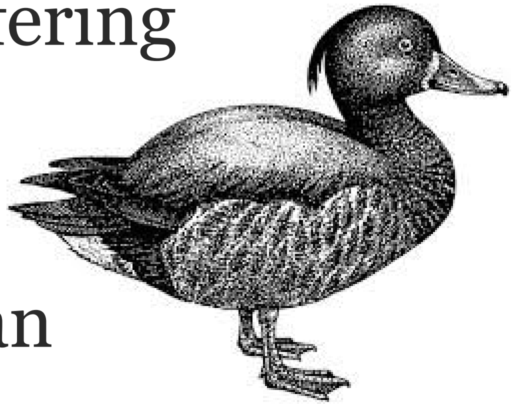
it should be possible to compact the architecture such that placeholder vocabulary elements can be optimised away

- implies facilities to offset the cost of abstraction



6 Substitutable

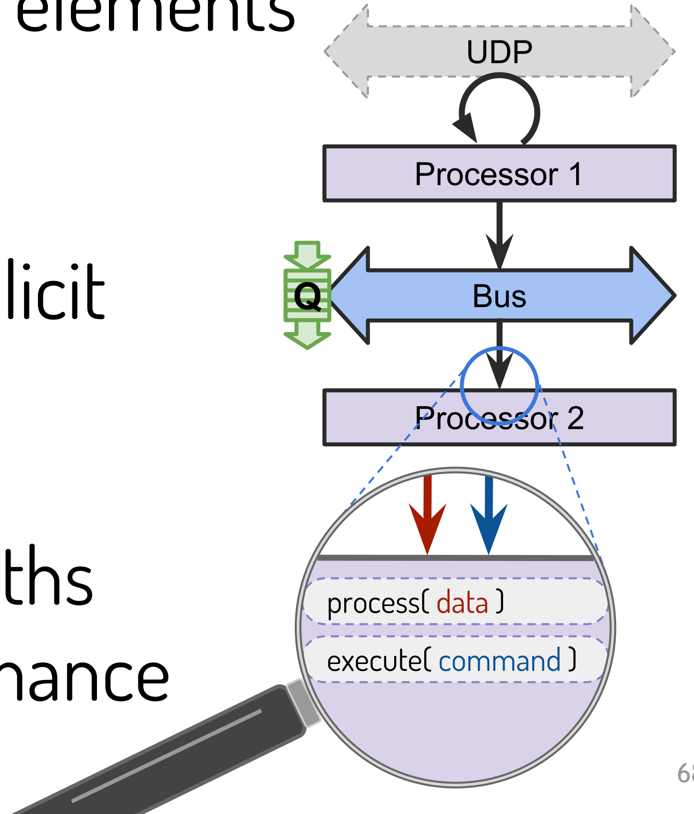
vocabulary elements should be replaceable by differing implementations with differing performance trade-offs



- implies consistent, clean interfaces

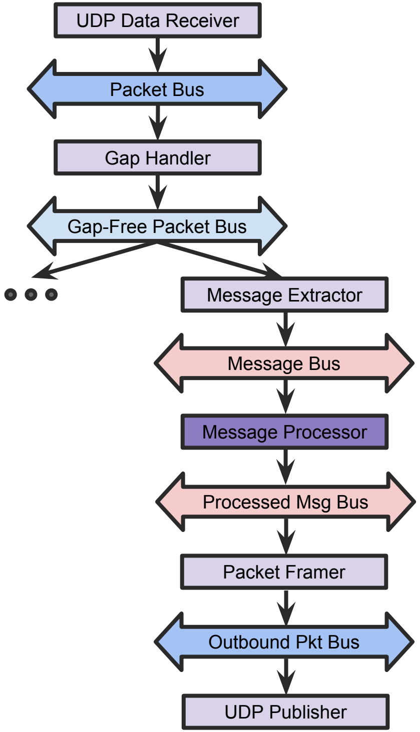
Recommendations

- Define building block vocabulary elements
- Avoid shared state
- Favour message passing
- Make synchronisation points explicit in the architecture
- Support push and pull models
- Separate Data and Command paths
- Static Polymorphism for Performance

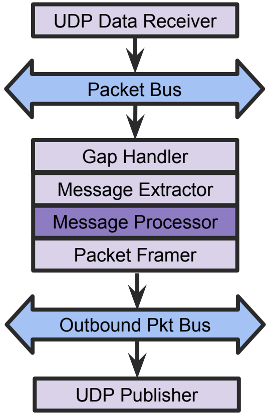


Simple Example

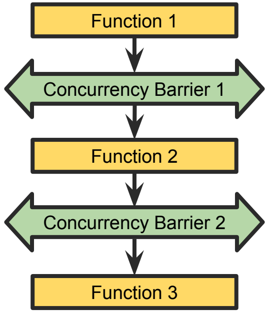
1 Design Using a Real Vocabulary of Real Components



2 Compact Architecture by removing conceptual components

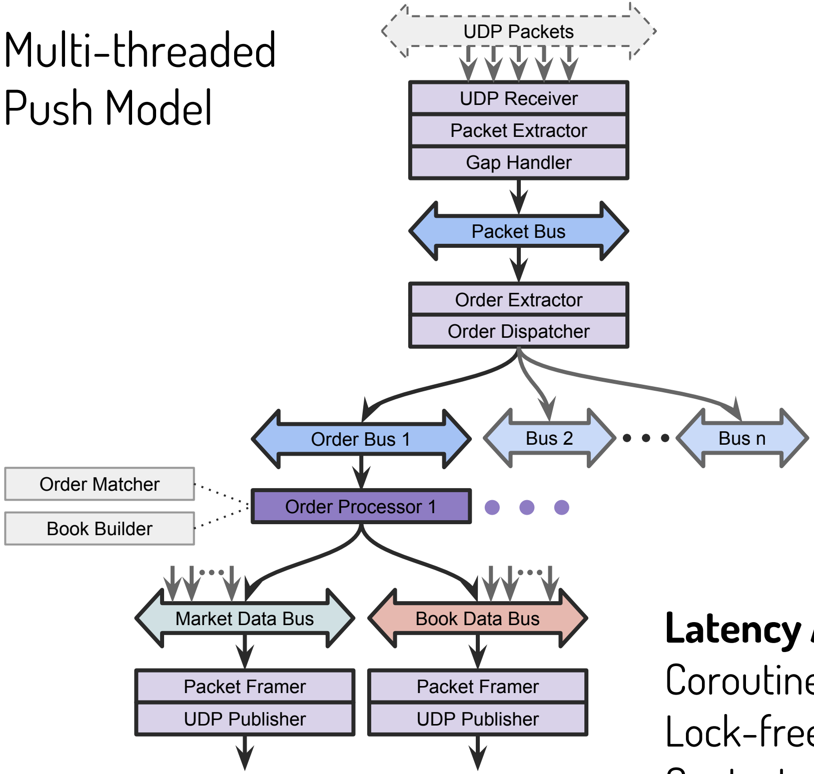


3 Compile to Optimised Implementation with zero abstraction cost

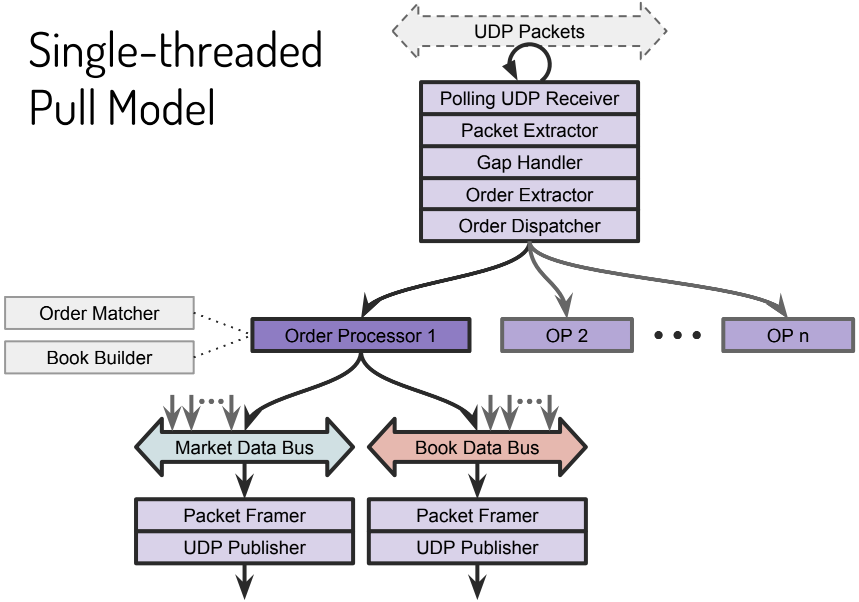


Different Performance Trade-offs

Multi-threaded
Push Model



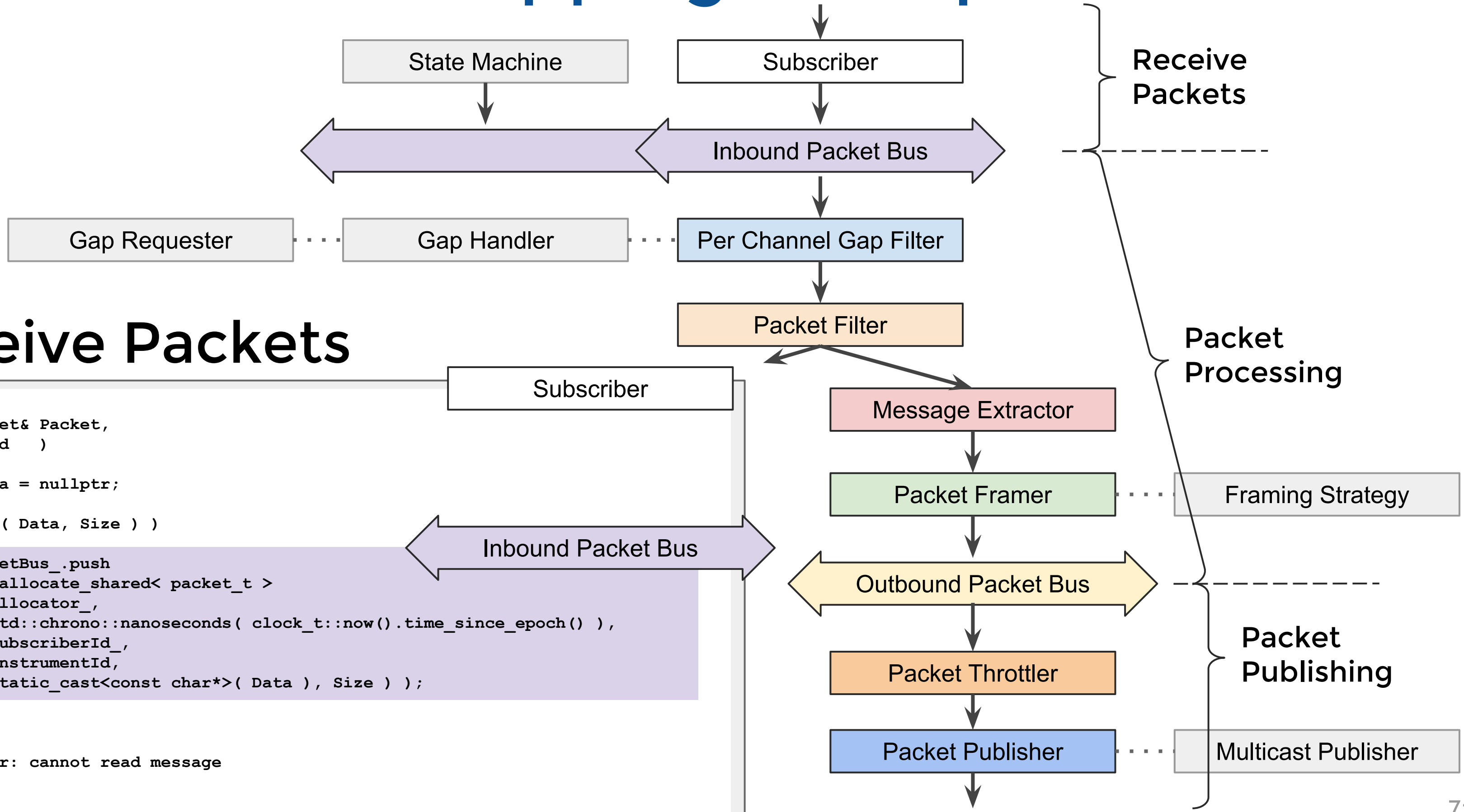
Single-threaded
Pull Model



Latency Agnostic
Coroutines?
Lock-free Queues?
Context-switches?

Scaling Agnostic
Single Process → Multiple Processes?
Single Core → Multiple Cores?
Single Server → Multiple Servers?

Code Mapping Example



Receive Packets

```
void on_packet
(   const data_packet& Packet,
    int InstrumentId    )
{
    const void* Data = nullptr;
    size_t Size;
    if( Packet.read( Data, Size ) )
    {
        InboundPacketBus_.push
            ( std::allocate_shared< packet_t >
                ( Allocator_,
                  std::chrono::nanoseconds( clock_t::now().time_since_epoch() ),
                  SubscriberId_,
                  InstrumentId,
                  static_cast<const char*>( Data ), Size ) );
    }
    else
    {
        // log error: cannot read message
    }
}
```

Packet Processing

Inbound Packet Bus

Per Channel Gap Filter

Packet Filter

Message Extractor

Packet Framer

Outbound Packet Bus

Outbound Packet Bus

```
void process( const shared_inbound_packet& InboundPacket )
{
    if( SeqNum == ExpectedSeqNum )
    {
        ExpectedSeqNum = SeqNum + NumMsgs;
        GapHandler_.update_expected_seq_num( ExpectedSeqNum, ChannelId );

        if(     InboundPacket->header().num_msgs()
            &&  InboundPacket->header().delivery_flag() == format::delivery_flag::original_message )
        {
            while( shared_message_t Message = InboundPacket->pop_front() )
            {
                if( FramingStrategy_->incoming_message_triggers_send( OutboundPacket_->size(), Message->size() ) )
                {
                    SeqNum_ += NumMsgsInPrevPacket_;
                    LastFrameTime_ = clock_t::now().time_since_epoch();
                    OutboundPacket_->assign_seq_num( SeqNum_ );
                    OutboundPacketBus_->push( OutboundPacket_ );
                    NumMsgsInPrevPacket_ = OutboundPacket_->header().num_msgs();
                    OutboundPacket_ = std::make_shared<outbound_packet_t>( format::delivery_flag::original_message );
                }
                OutboundPacket_->push_back( Message );
                if( FramingStrategy_->packet_requires_immediate_send( OutboundPacket_->size(), Message->last_message_in_packet() ) )
                {
                    SeqNum_ += NumMsgsInPrevPacket_;
                    LastFrameTime_ = clock_t::now().time_since_epoch();
                    OutboundPacket_->assign_seq_num( SeqNum_ );
                    OutboundPacketBus_->push( OutboundPacket_ );
                    NumMsgsInPrevPacket_ = OutboundPacket_->header().num_msgs();
                    OutboundPacket_ = std::make_shared<outbound_packet_t>( format::delivery_flag::original_message );
                }
            }
        }
        else
        {
            // send command::category::notification - packet_discarded
        }
    }
    else if( SeqNum > ExpectedSeqNum )
    {
        ExpectedSeqNum = GapHandler_.handle_unexpected_packet( InboundPacket, ExpectedSeqNum, ChannelId );
    }
    else if( SeqNum < ExpectedSeqNum )
    {
        // log and ignore
    }
}
```

Lastly...

Publish Packets

Outbound Packet Bus

```
void process( const shared_outbound_packet& OutboundPacket )
{
    delay_before_send( OutboundPacket->size() );
    OutboundPacket->assign_send_time( std::chrono::nanoseconds( clock_t::now().time_since_epoch() ) );

    MulticastPublisher_->process( OutboundPacket );
}
```

Packet Throttler

Packet Publisher

Vocabulary elements map directly to code

- Code still lives in separate ‘modules’
- Maintained and tested separately
- Communication through building block interfaces
- Abstraction cost removed but clarity retained
- Easy to change, fix, replace

Additional Benefits of a Common Vocabulary

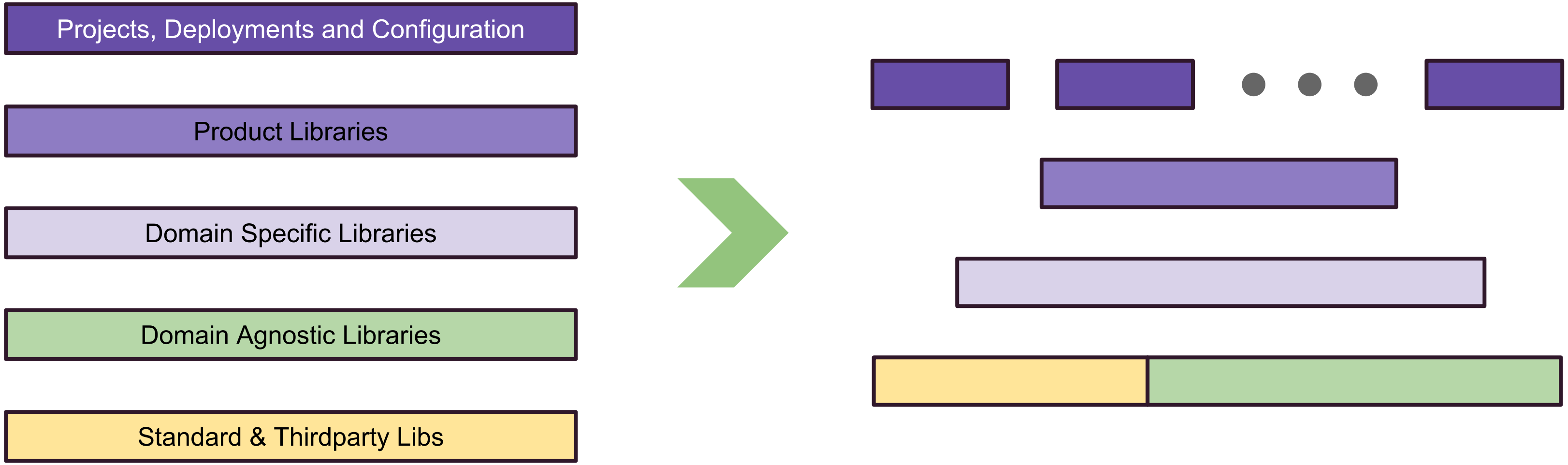
Common Vocabulary → Tiered Structure

Source code is arranged in tiers facilitating a layered development structure and allowing critical code to retain high quality and performance

Projects, Deployments and Configuration	New York Equities Platform
Product Libraries	matching_engine, gateway
Domain Specific Libraries	gap_handler, format, session
Domain Agnostic Libraries	multicast, bus, concurrency
Standard & Thirdparty Libs	boost, std, asio

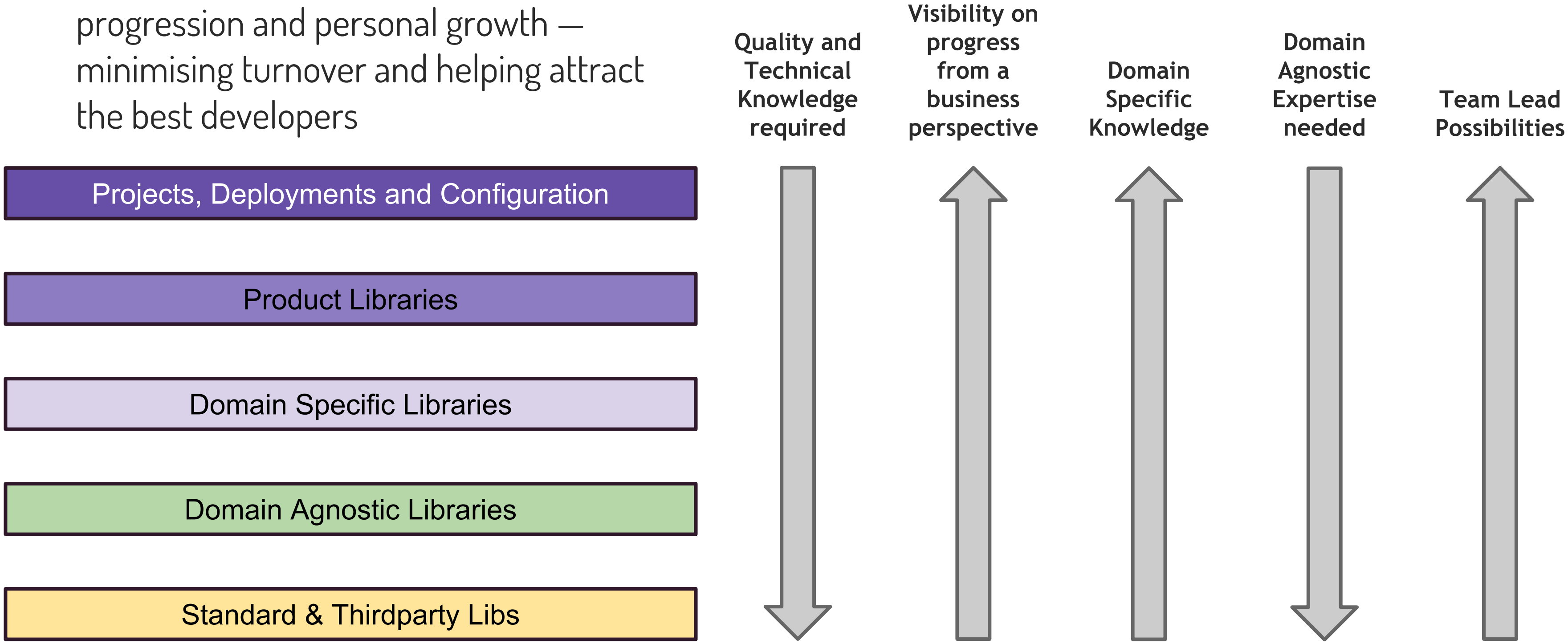
Stable Foundations

Tiers form a pyramid of code with the foundations formed by re-usable components and libraries of well tested code

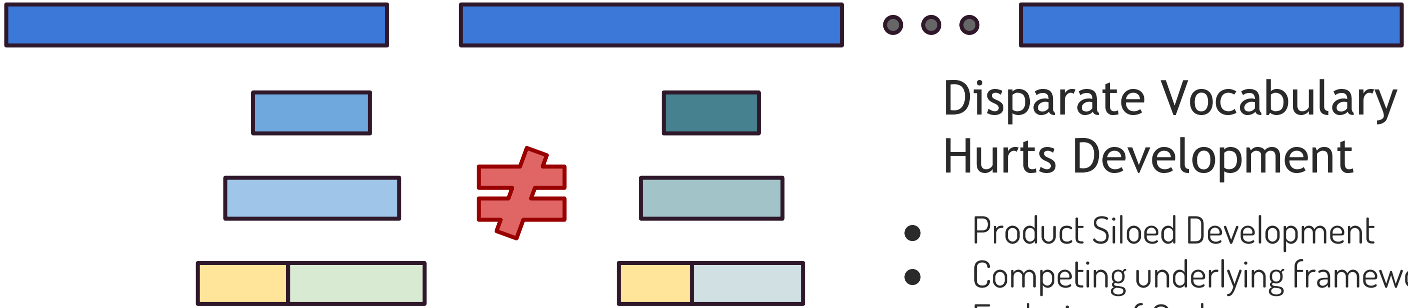


Developer Growth

- Allows different experience and skillsets to be catered to throughout the team
- Provides clear opportunities for progression and personal growth — minimising turnover and helping attract the best developers



Contrast with Disparate Vocabulary



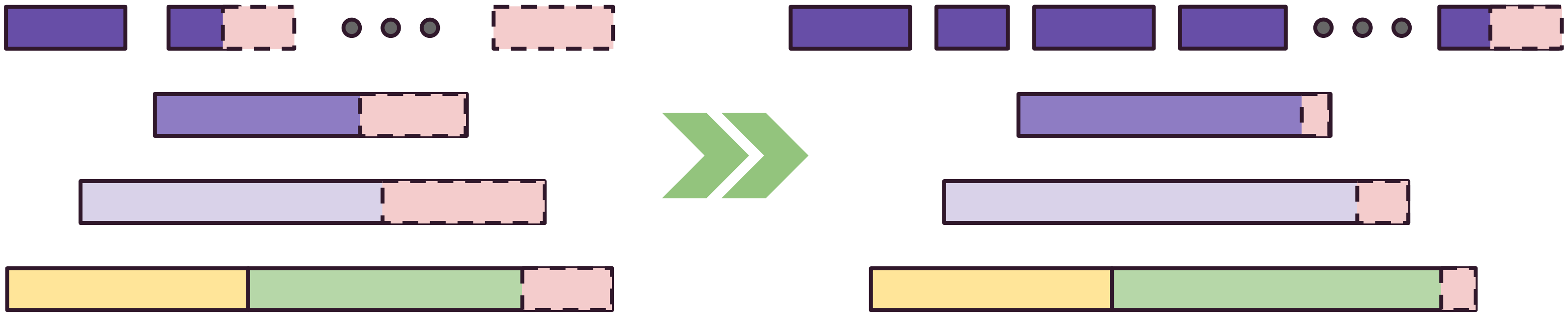
Disparate Vocabulary Hurts Development

- Product Siloed Development
- Competing underlying frameworks
- Explosion of Code

With A Common Vocabulary Less is More

- Possible to adopt a Core Framework
- Product Building Focused more on Assembly
- Scales across Teams and Geographies
- Developer and Business share the vocabulary

Accelerated Development



Products based on shared framework

- Development rate increases over time
- Framework stabilises over time
- Developer turnover less impact

Minimal Toolchain possible

- Hiring Easier
- Maintenance Easier
- Faster Learning

C++ (core language, high perf, servers), **Python** (web-server, scripting, builds, test), **Javascript** (web-clients), **SCSS** (presentation), **Postgresql** (data storage)

We favour a more holistic view of development — one that puts people as a central aspect of architecture

Final Thoughts

In a highly regulated, ever-changing, environment with extreme performance constraints it is increasingly difficult to avoid full system rewrites to meet changing requirements

Algorithmic architecture is primarily about adhering to certain principles and concepts where the goal is to facilitate clear understanding within complex and changing problem domains

The goal of those principles is to allow optimisation (and general improvement) of an architecture to occur at the highest level possible—the architecture itself—allowing adaptivity and evolution

Thank you for Listening

clearpool.io

jamie.allsop@clearpool.io