© 2016 Jamie Allsop

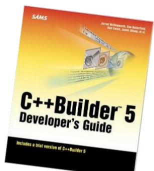
Algorithmic Architecture

Performant Architecture in Evolving Regulatory Environments

Jamie Allsop

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

focused on **C++** & standards work



passionate about **agile**

fiddle with python

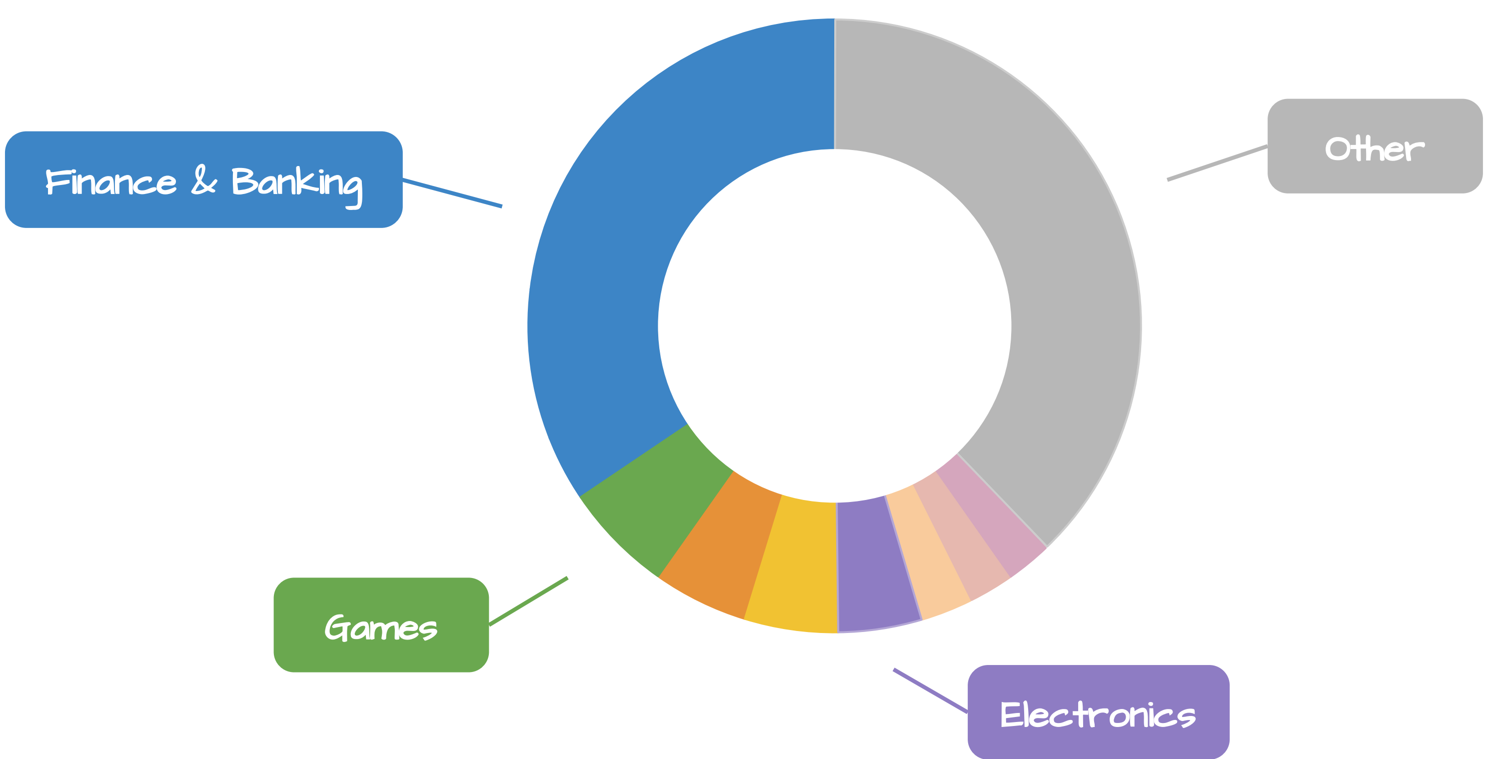
for Scons

ended up at

now director at

© 2016 Jamie Allsop

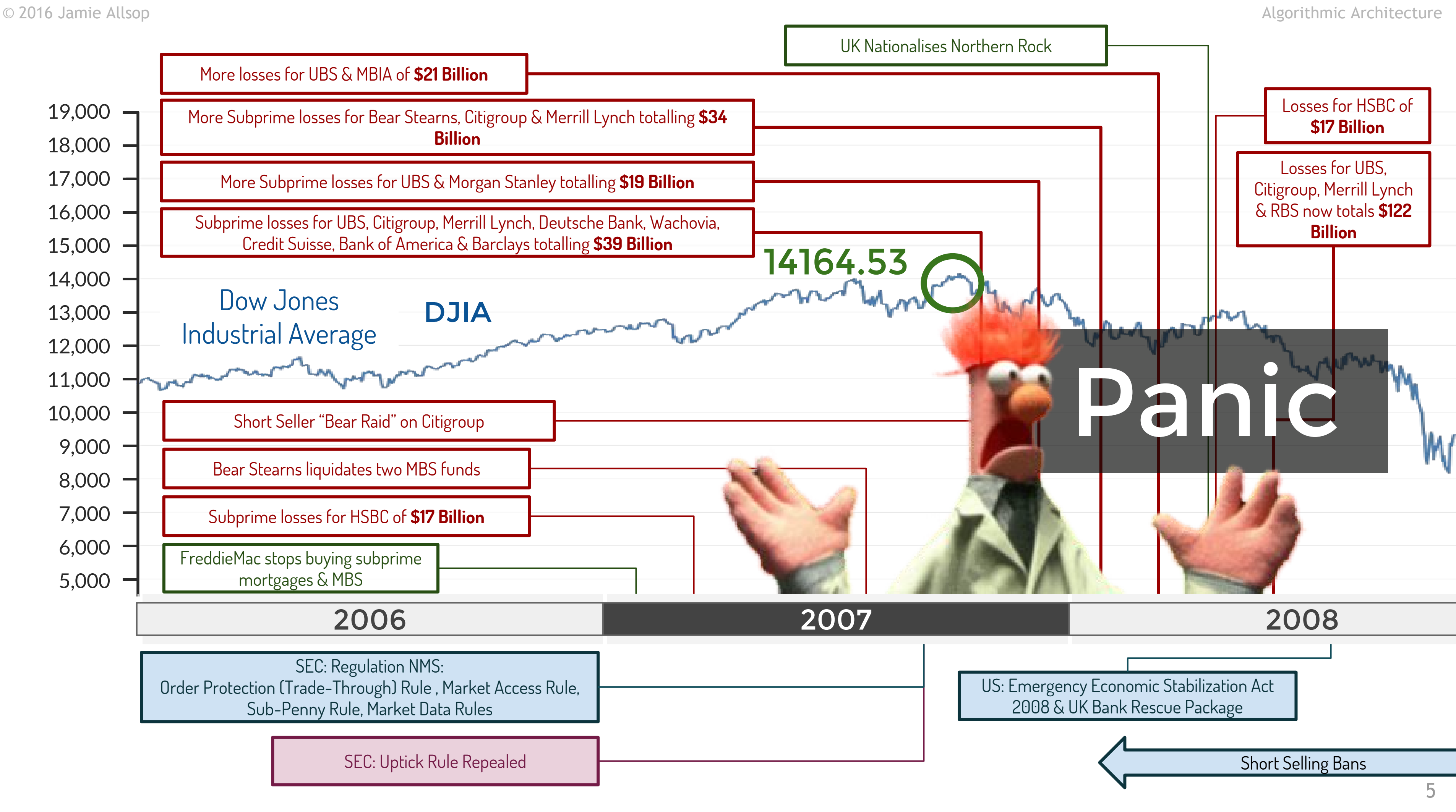
Context (C++)



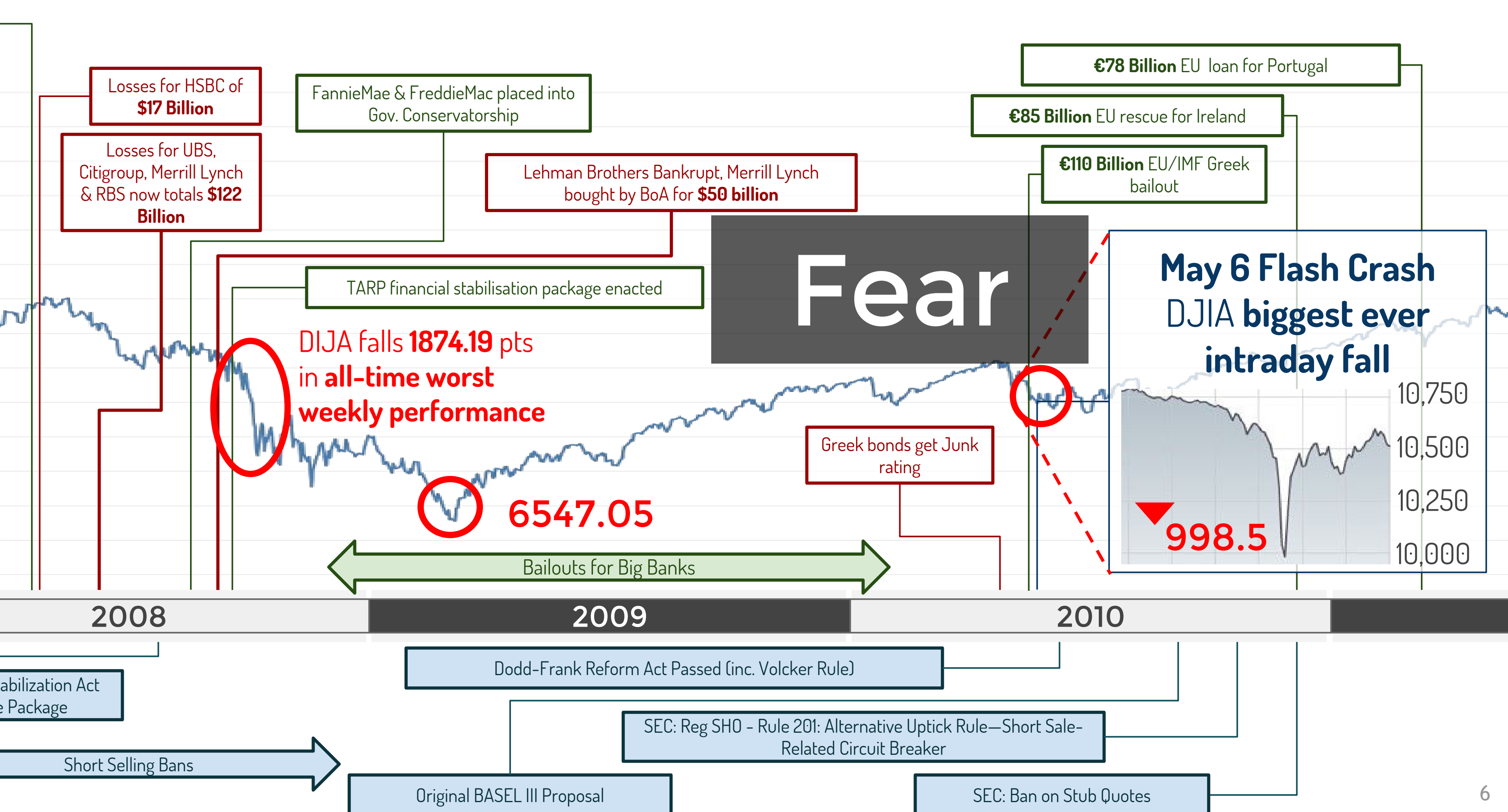
Source: http://blog.jetbrains.com/clion/2015/09/cpp-annotated-summer-edition/

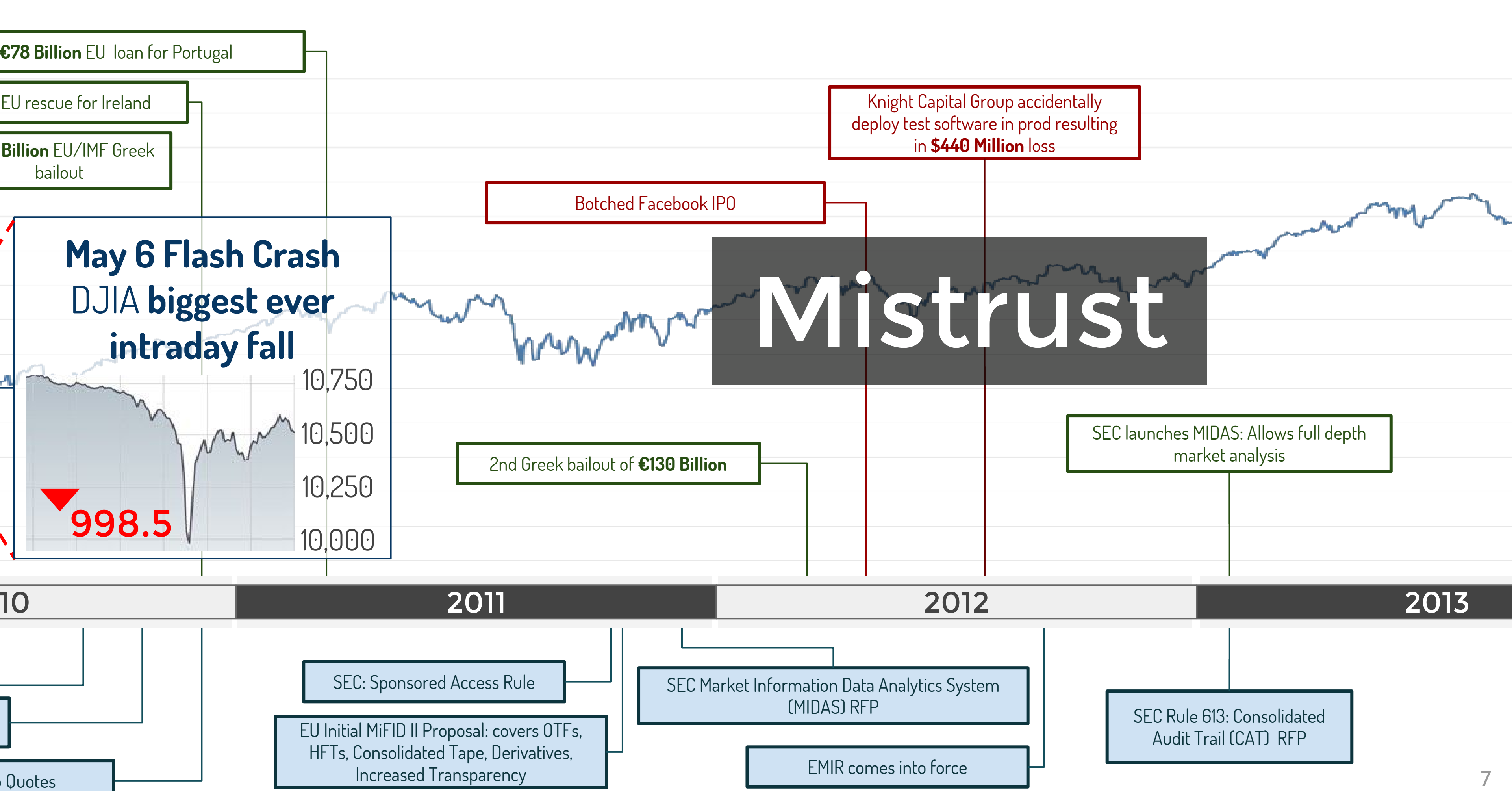
- regulations and change
- problems, people and software
- architecture and performance

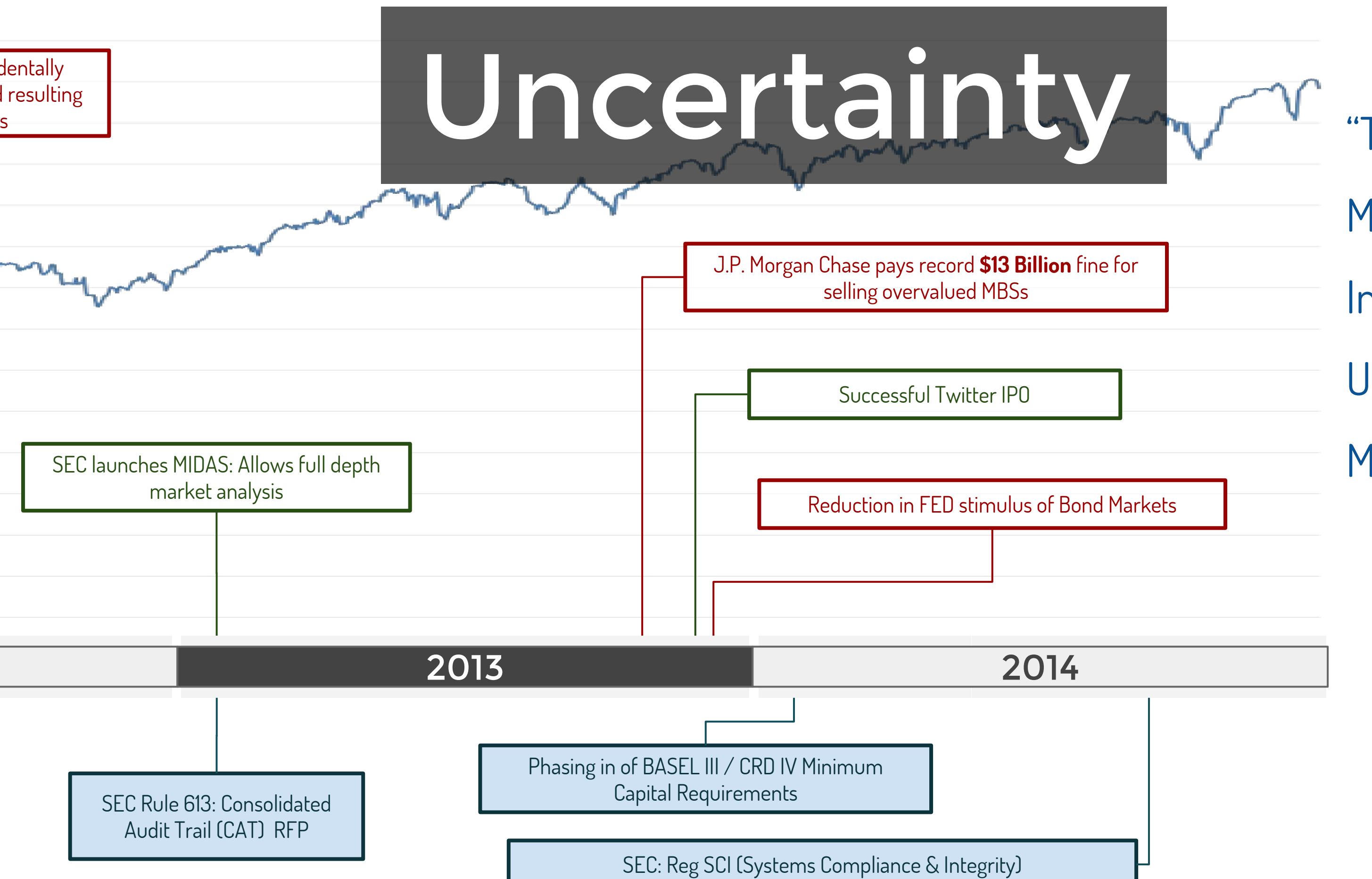
© 2016 Jamie Allsop



© 2016 Jamie Allsop




€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
Knight Capital Group accidentally deploy test software in prod resulting in $440 Million loss
Botched Facebook IPO
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



"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

© 2016 Jamie Allsop

Explosive Growth in Regulatory Burden

NATIONAL BANKING ACT

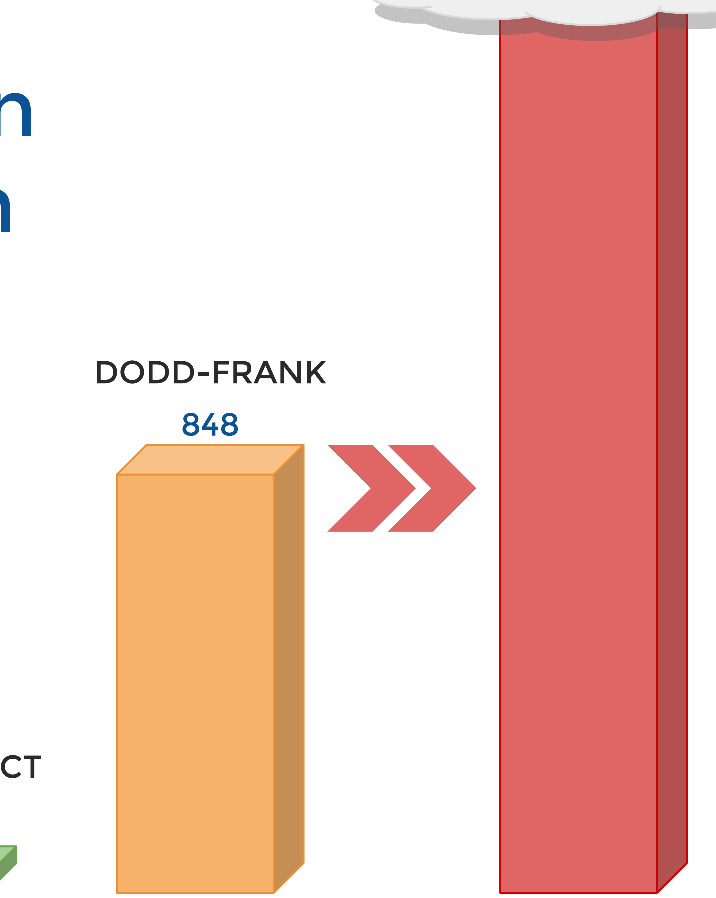
29

FEDERAL RESERVE ACT

32

BANKING ACT

37



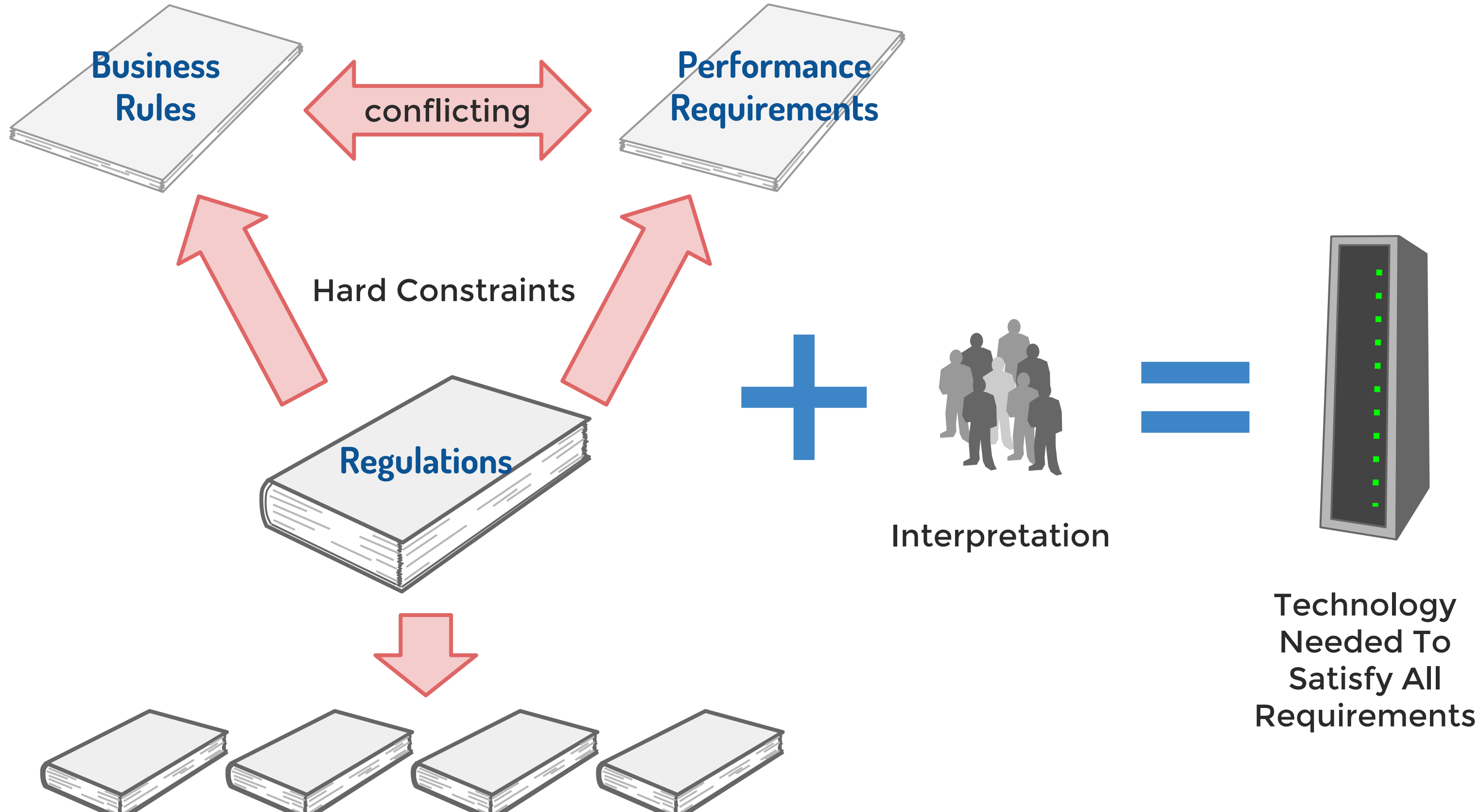
© 2016 Jamie Allsop

There are often Hard Constraints

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

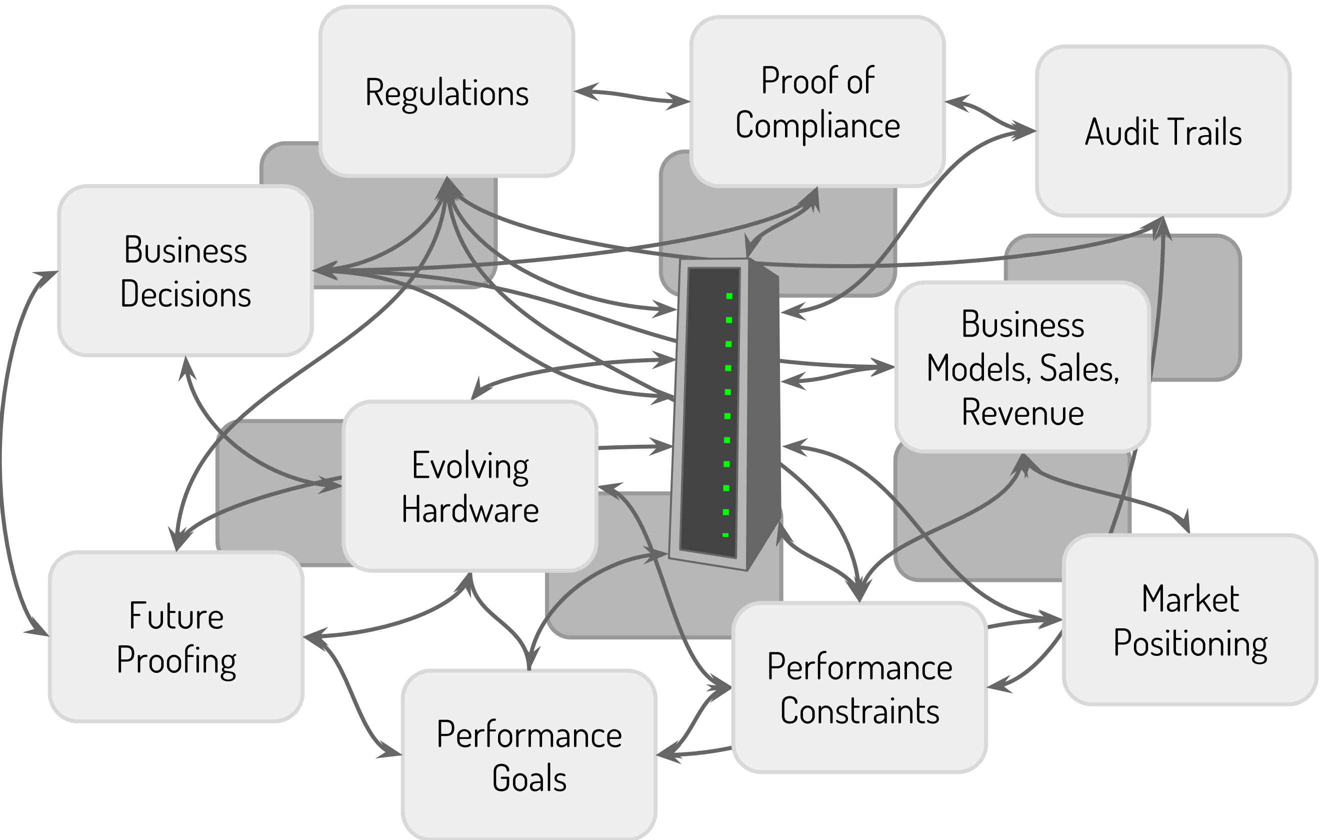
many constraints driven by regulations

Let's simplify this…



© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

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

Gather Data → Analyse Data → Formulate Solution → Implement Solution

© 2016 Jamie Allsop

Messes

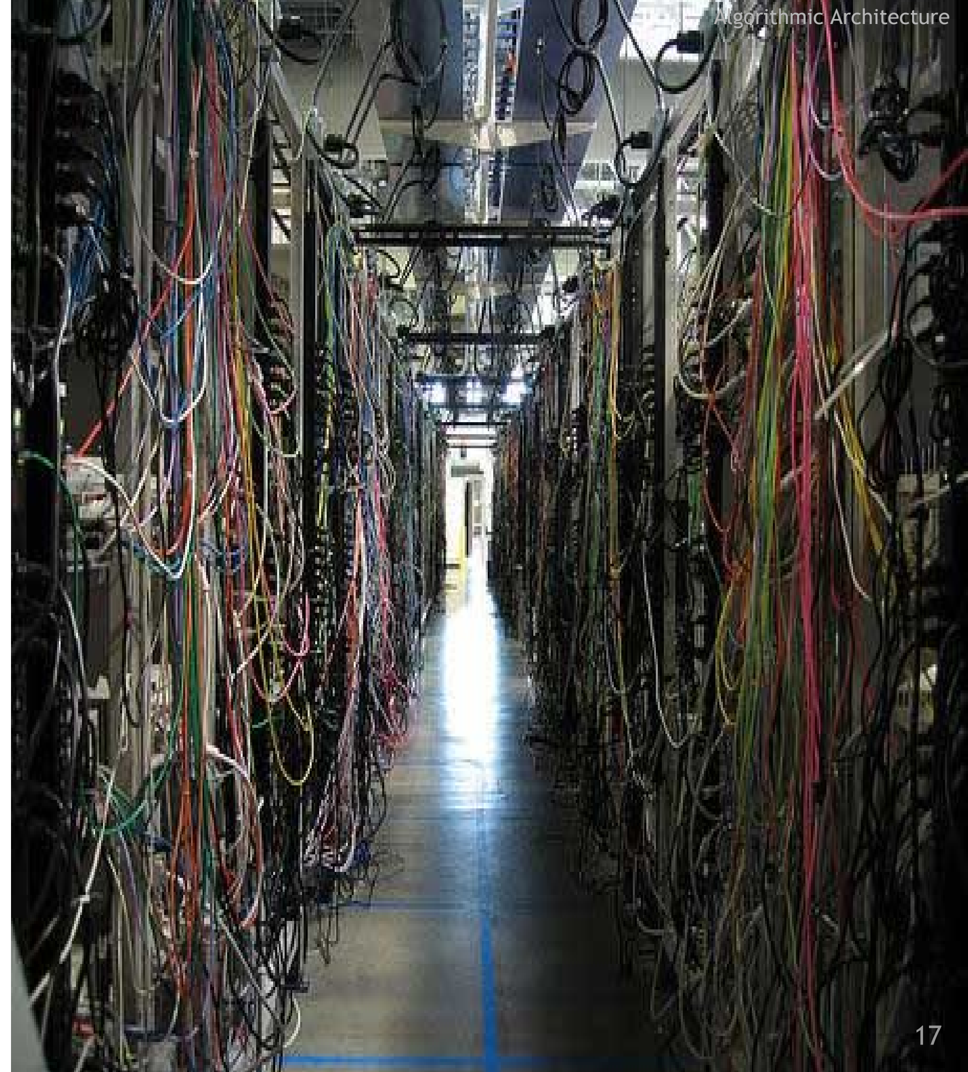
Organised complexity

- 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**



© 2016 Jamie Allsop

Messes are… a 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

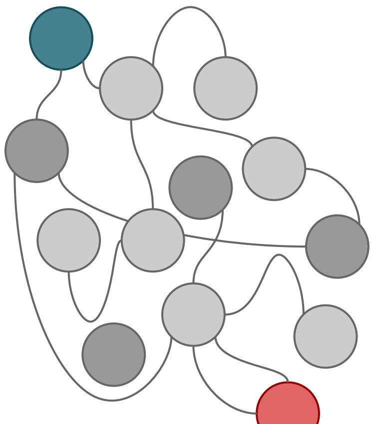
© 2016 Jamie Allsop

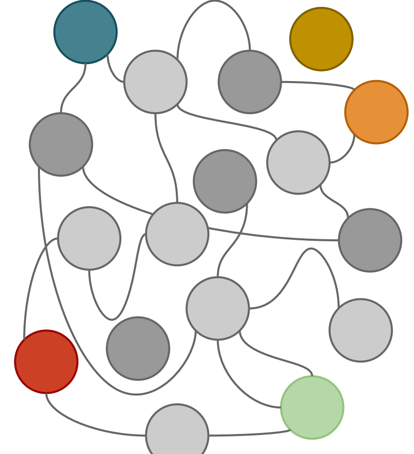
coupling...

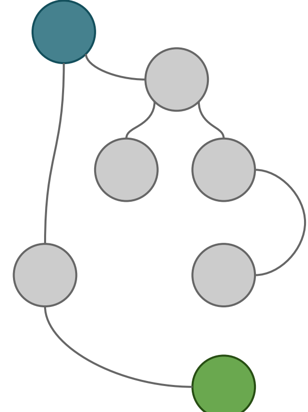
© 2016 Jamie Allsop

Bugfixing?

Refactoring?





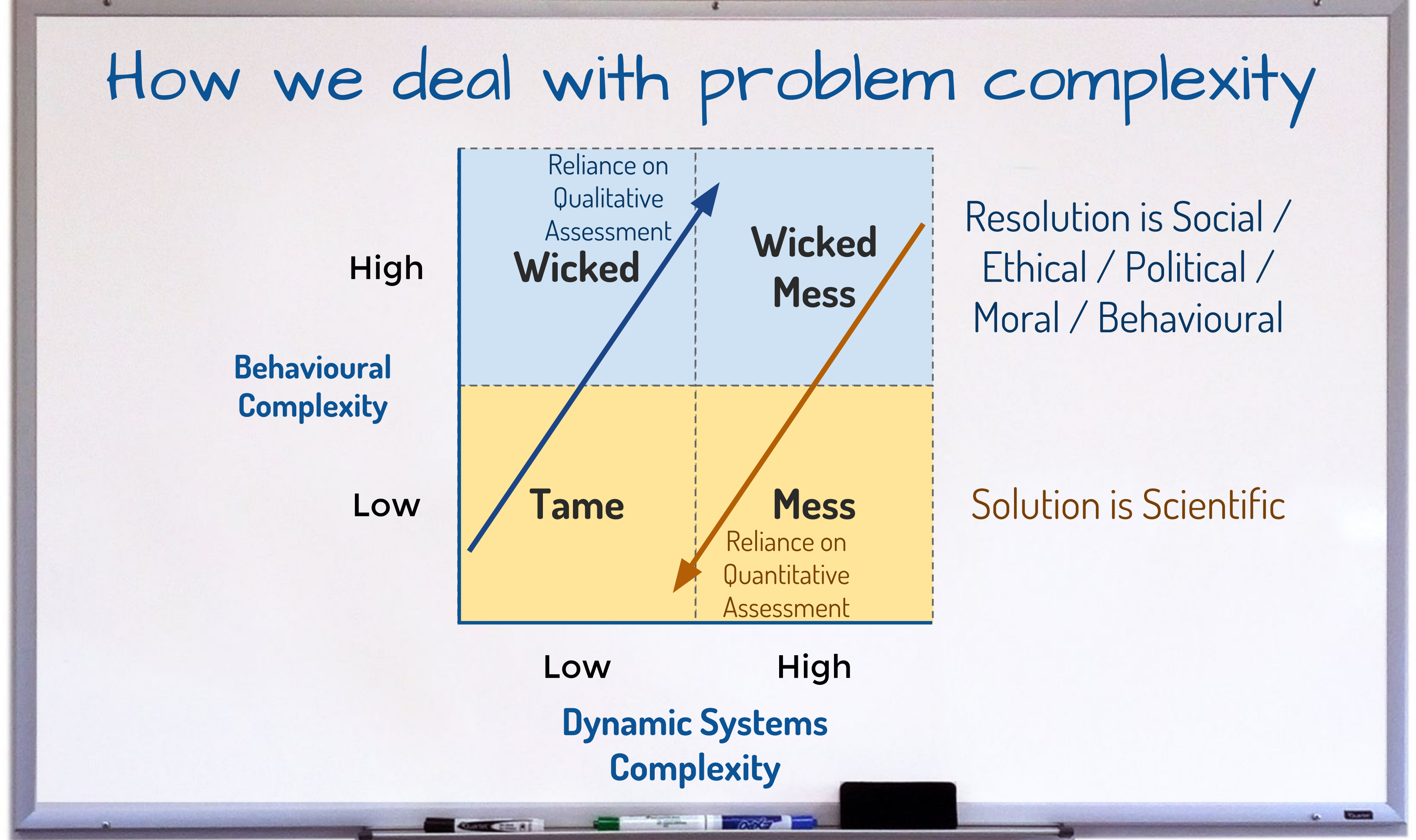


- Conflicting **social** ethics and beliefs
- Smart, informed people **disagree**
- **Divergent** problems with no promise of a solution
- **Evolving** set of **Interlocking** Issues and Constraints
- Constraints **change over Time**
- Many Stakeholders

Wickedness

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
Reliance on Qualitative Assessment
High
Wicked
Wicked Mess
Resolution is Social / Ethical / Political / Moral / Behavioural
Behavioural Complexity
Low
Tame
Mess
Solution is Scientific
Reliance on Quantitative Assessment
Low
High
Dynamic Systems 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

Iterative

Timeboxing

Qualitative Progress Assessment by Expert Stakeholders

Getting the right Stakeholders together

Communication Transparency

Listening and Establishing Trust

Regulatory Solutions are Too Slow to React Effectively

© 2016 Jamie Allsop

Triggered and Skewed by Events: Flash Crash and HFTs?

© 2016 Jamie Allsop

ACCEPTED WISDOM

Boundaries for **qualitative assessment** by Expert Stakeholders

Approaches to Wicked Problems

Iterative

Timeboxing

Qualitative Progress Assessment by Expert Stakeholders

Getting the right Stakeholders together

Communication Transparency

Listening and Establishing Trust

Sounds a lot like Agile Development?

© 2016 Jamie Allsop

Agile and We're Done?

© 2016 Jamie Allsop

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?

© 2016 Jamie Allsop

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

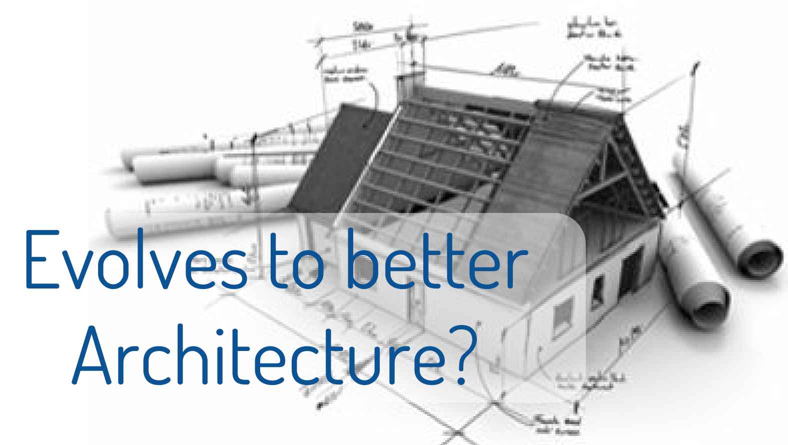
- Marketecture is often driven by decisions that have **no regard for the technical impact**
- Stakeholders change
- Goal posts move
- "Power **without** responsibility"
- Poor choices baked in early
- What's most important?

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

© 2016 Jamie Allsop

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

Can we do better?

© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

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

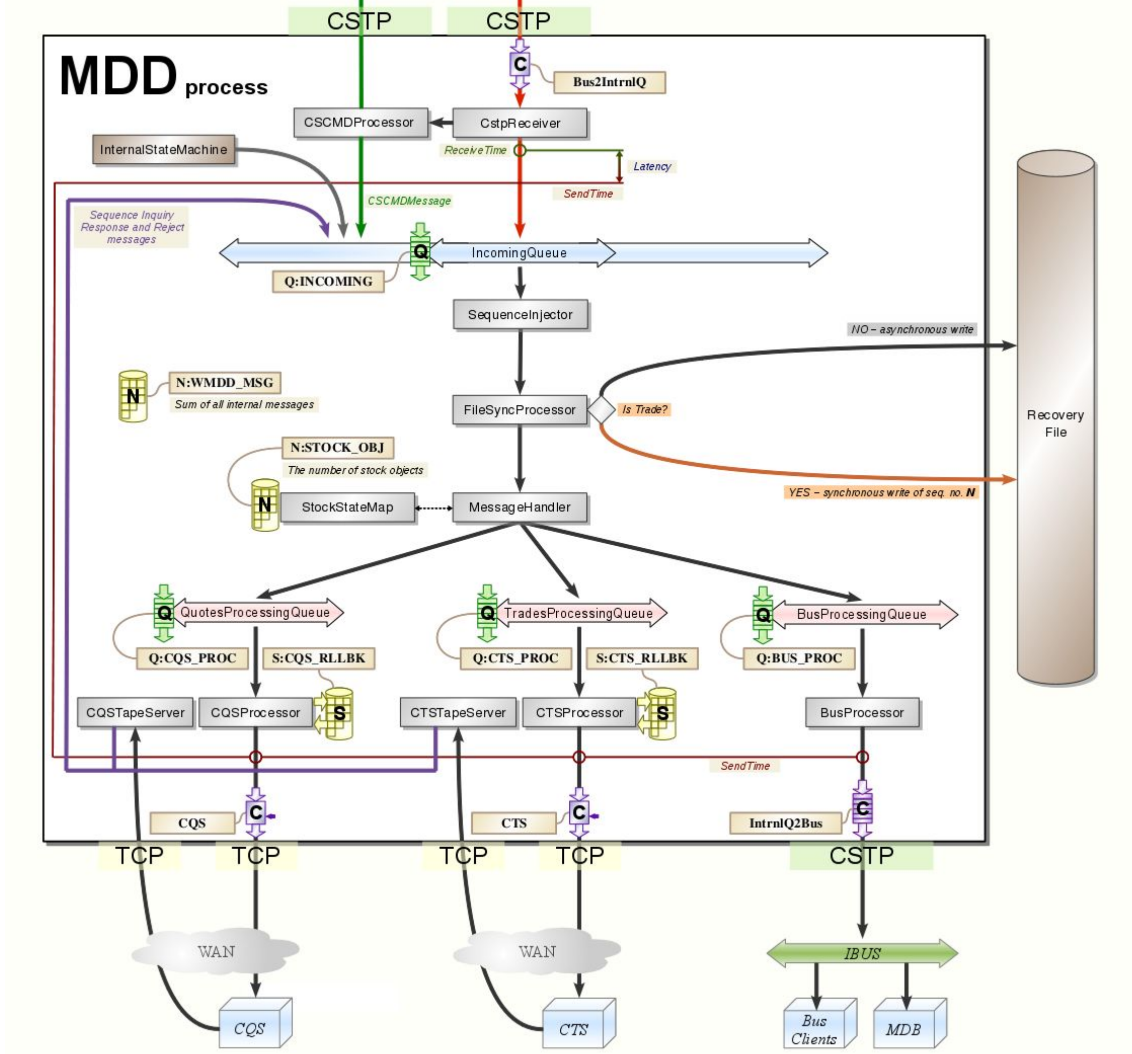
The SEC were presented with architectural overviews of how the systems involved behaved, and how they were evolved

© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

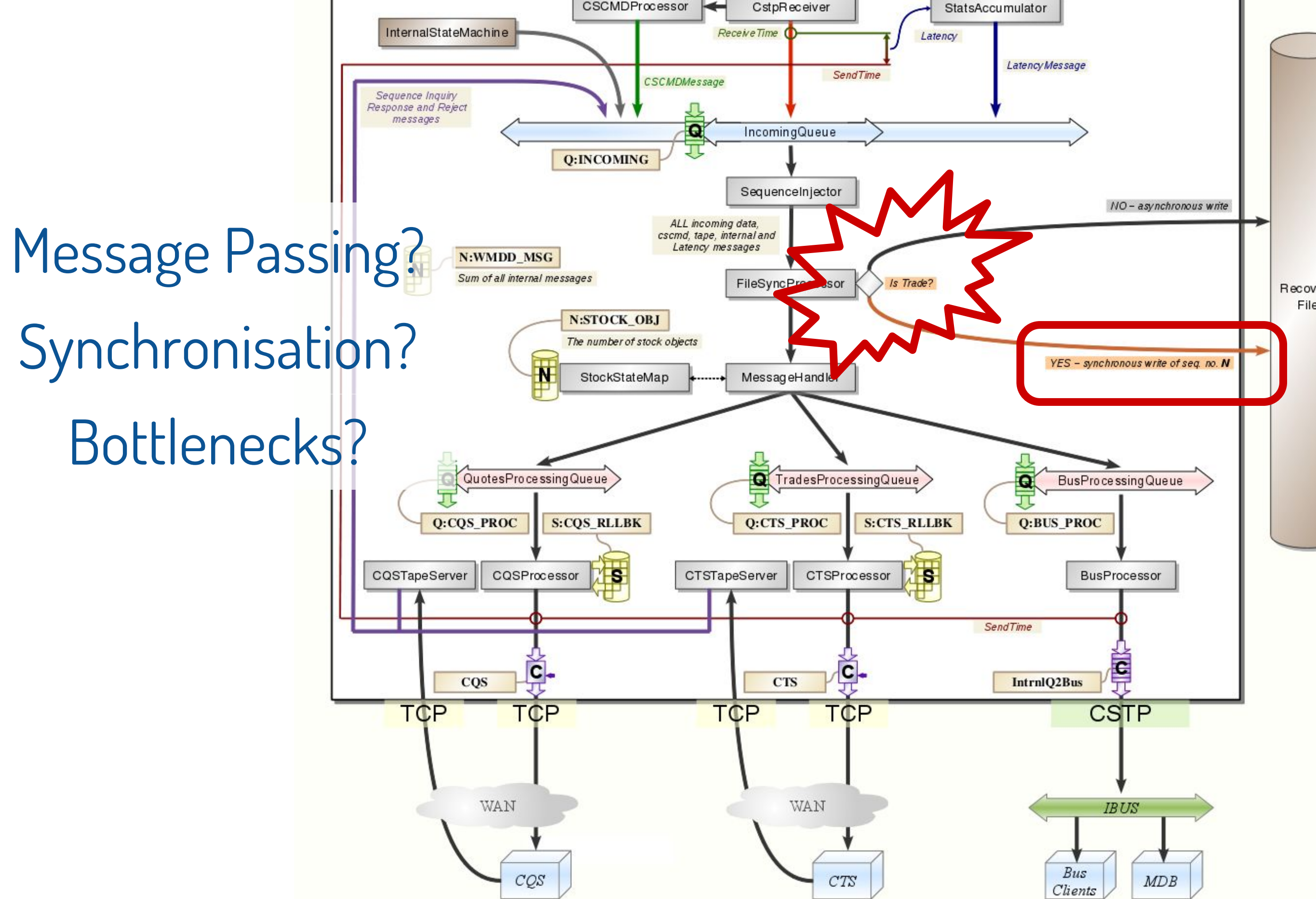
What can we tell from looking at this picture?



1

Data flow
Networking
Queuing
Decisions
Processors
Data stores

© 2016 Jamie Allsop



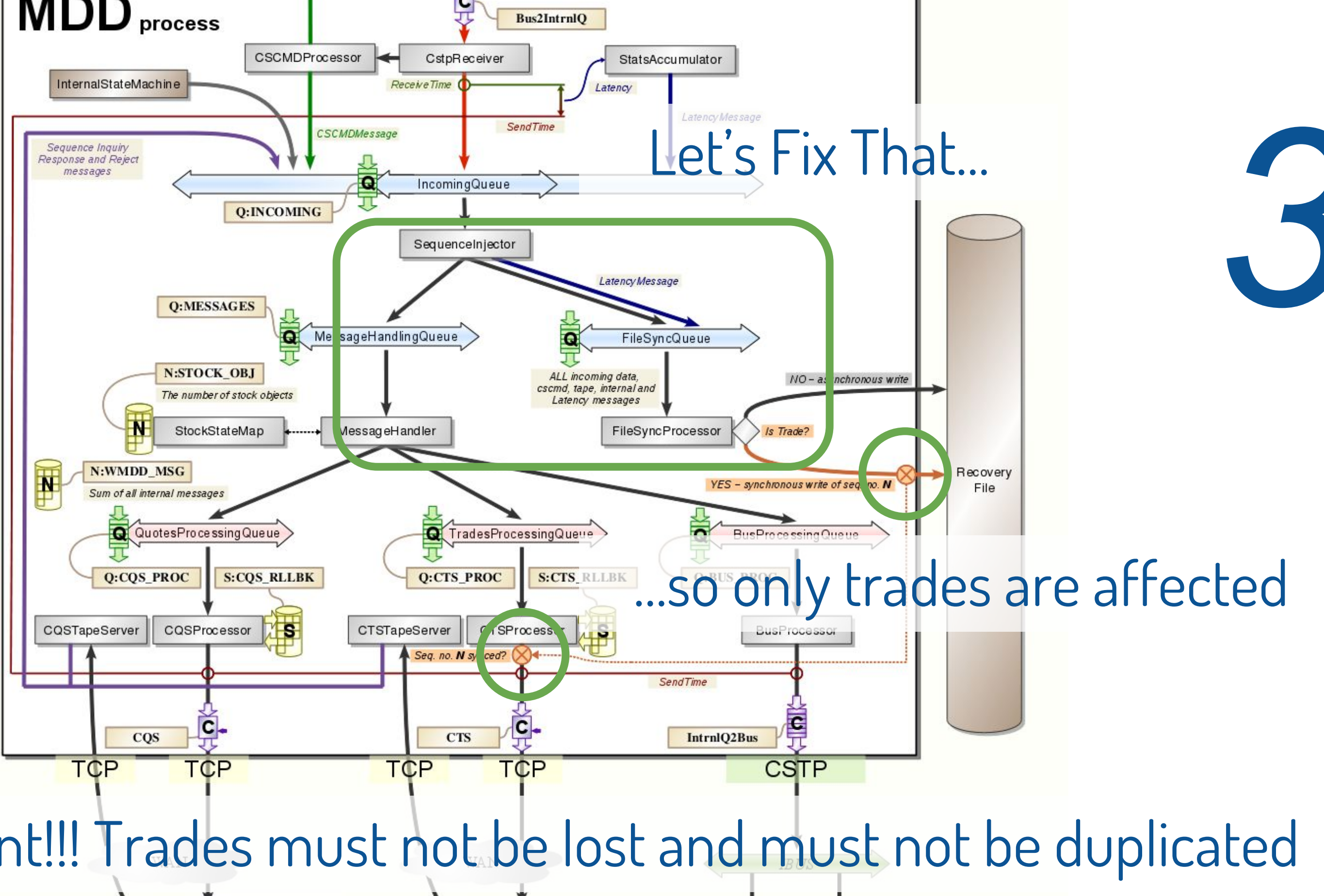
Message Passing?

Synchronisation?

Bottlenecks?

2

© 2016 Jamie Allsop



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

© 2016 Jamie Allsop

MDD process

CSTP
CSTP
Bus2IntrnlQ
CSCMDProcessor
CstpReceiver
StatsAccumulator
StatsManager
InternalStateMachine
ReceiveTime
Latency
TransientStats
SendTime
LatencyMessage
CSCMDMessage
Sequence Inquiry Response and Reject messages
IncomingQueue
Q:INCOMING
Stats (csv) File
SequenceInjector
LatencyMessage
Q:MESSAGES
MessageHandlingQueue
FileSyncQueue
N:STOCK_OBJ
The number of stock objects
ALL incoming data, cscmd, tape, internal and Latency messages
StockStateMap
MessageHandler
FileSyncProcessor
Is Trade?
Unbuffered write if buffer full
Recovery File
N:WMDD_MSG
BusProcessingQueue
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

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...

© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

A Story…
Not the Whole Story

© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

MDD process

TCP
MC
CSTP
CSTP
Bus2IntrnlQ
RetransHandler
MulticastReceiver
StatsManager
ReceiveTime
ReceiveTime
CSCMDProcessor
CstpReceiver
TransientStats
IncomingPacketQueue
Latency
StatsAccumulator
ReceiveTime
SendTime
LatencyMessage
CSCMDMessage
Stats (csv) File
S
GapHandler
GapFilter
InternalStateMachine
S:GAP_STORE
MessageExtractor
Current Gaps and Partial Sequences
Q
IncomingQueue
Q:INCOMING
Sequence Inquiry Response and Reject messages
SequenceInjector
LatencyMessage
Q:MESSAGES
Q
MessageHandlingQueue
FileSyncQueue
Q
Q:FILESYNC
N
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
N
StockStateMap
MessageHandler
YES – flushed unbuffered write of seq. no. N
Q
QuotesProcessingQueue
Q
TradesProcessingQueue
Q
BusProcessingQueue
Q:CQS_PROC
S:CQS_RLLBK
Q:CTS_PROC
S:CTS_RLLBK
Q:BUS_PROC
CQSTapeServer
CQSProcessor
S
CTSTapeServer
CTSProcessor
S
BusProcessor
Seq. no. N written?
SendTime
C
CQS
C
CTS
C
IntrnlQ2Bus
TCP
TCP
TCP
TCP
CSTP
WAN
WAN
IBUS
CQS
CTS
Bus Clients
MDB

MDD process

TCP
MC
CSTP
CSTP
InternalStateMachine
Bus2IntrnlQ
StatsManager
RetransHandler
MulticastReceiver
CSCMDProcessor
CstpReceiver
TransientStats
Latency
StatsAccumulator
ReceiveTime
ReceiveTime
ReceiveTime
SendTime
LatencyMessage
CSCMDMessage
Stats (csv) File
Response and Reject
IncomingQueue
Q:INCOMING
XDP Filter
XDP TCP & Mulitcast data only
BUS data and Internal data
GapHandler
GapFilter
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

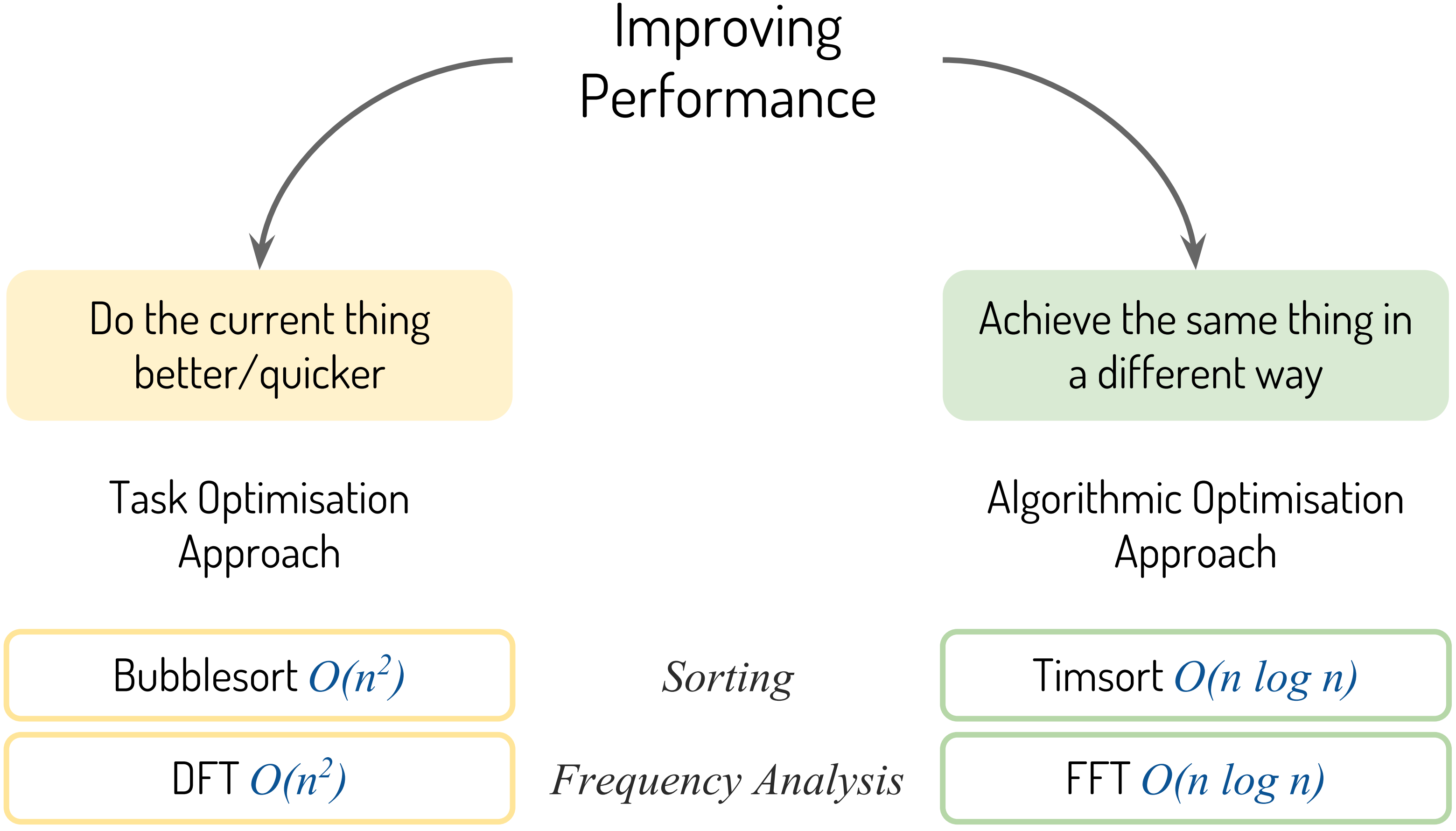
© 2016 Jamie Allsop

MDD process

TCP
MC
CSTP
CSTP
InternalStateMachine
Bus2IntrnlQ
StatsManager
TransientStats
RetransHandler
MulticastReceiver
CSCMDProcessor
CstpReceiver
Latency
StatsAccumulator
Stats (csv) File
ReceiveTime
ReceiveTime
ReceiveTime
SendTime
LatencyMessage
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
PacketFilter
NO – buffered write
Unbuffered write if buffer full
N:STOCK_OBJ
The number of stock objects
MessageExtractor
Non-XDP Packets are passed on directly
FileSyncProcessor
Is Trade?
Recovery File
YES – flushed unbuffered write of seq. no. N
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

© 2016 Jamie Allsop

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

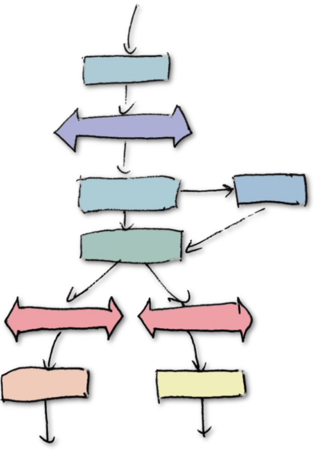
© 2016 Jamie Allsop

Environmental Influences

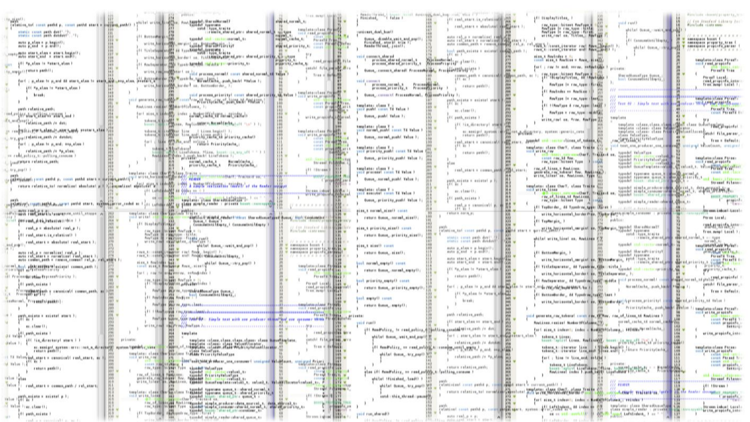
- Architecture for wicked problems typically a "**mess**"
- Many stakeholders and evolving problem domain over time adds "**wickedness**"
- Decomposing and understanding interactions difficult
- Such architecture, good or bad, is often hard to reason about in a way that maps directly to code
- Favours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

© 2016 Jamie Allsop

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"

© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

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

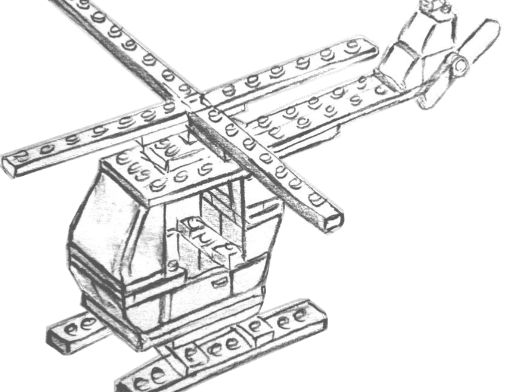
it should be possible to decompose the architecture into vocabulary elements that communicate the intent of the system

- ➤ implies partitioning interfaces

3 Composable

composable components can be assembled together to complete more complex tasks

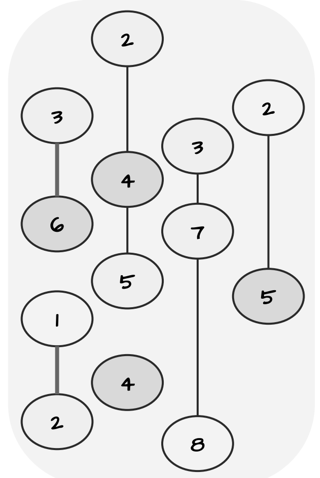
- implies common approach to communication



4 Independently orderable

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

- implies loose coupling



© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

6 Substitutable

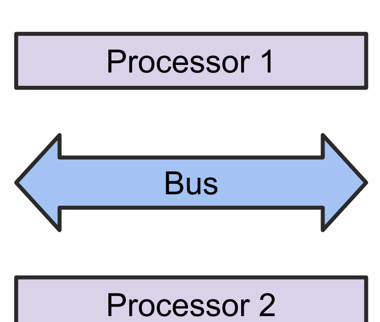
vocabulary elements should be replaceable by differing implementations with differing performance trade-offs

- implies consistent, clean interfaces

Recommendations

- Define **building block vocabulary elements**

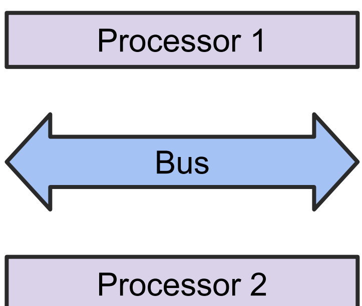
```
template<class DataT>
void process( const DataT& Data );

template<class DataT>
void push( const DataT& Data );

template<class ProcessorT>
void connect( ProcessorT Processor );
```



© 2016 Jamie Allsop

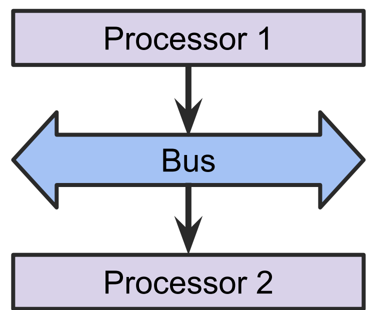
Recommendations

- Define building block vocabulary elements
- Avoid **shared state**



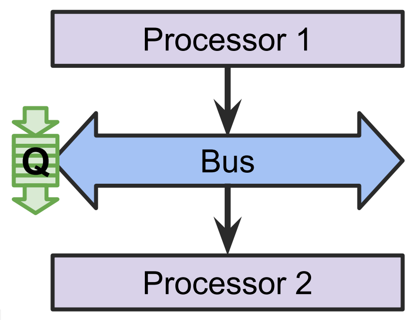
Recommendations

- Define building block vocabulary elements
- Avoid shared state
- Favour **message passing**



Recommendations

- Define building block vocabulary elements
- Avoid shared state
- Favour message passing
- Make **synchronisation points explicit** in the architecture



Synchronisation points are not composable. If you hide them you run the risk of concurrency hazards such as livelocks, starvation, deadlocks, and convoying

© 2016 Jamie Allsop

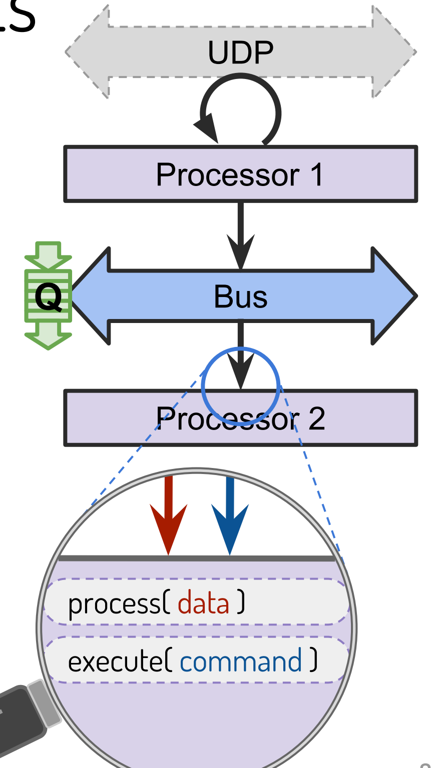
Recommendations

- Define building block vocabulary elements
- Avoid shared state
- Favour message passing
- Make synchronisation points explicit in the architecture
- Support **push** and **pull** models

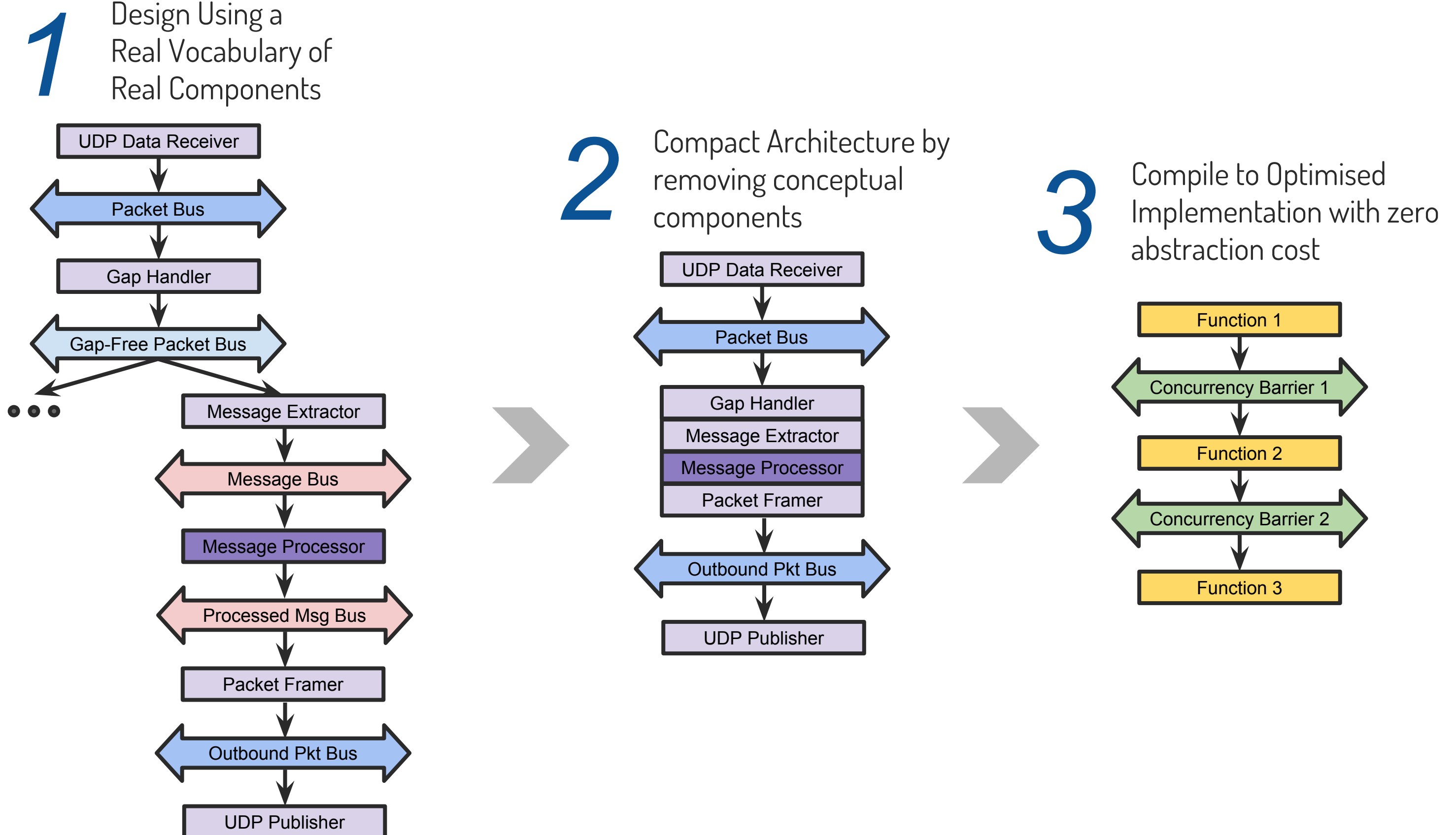
```
enum class read_policy{ on_data, poll };
template<class ProcessorT>
void connect( ProcessorT Processor, read_policy Read );
```

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

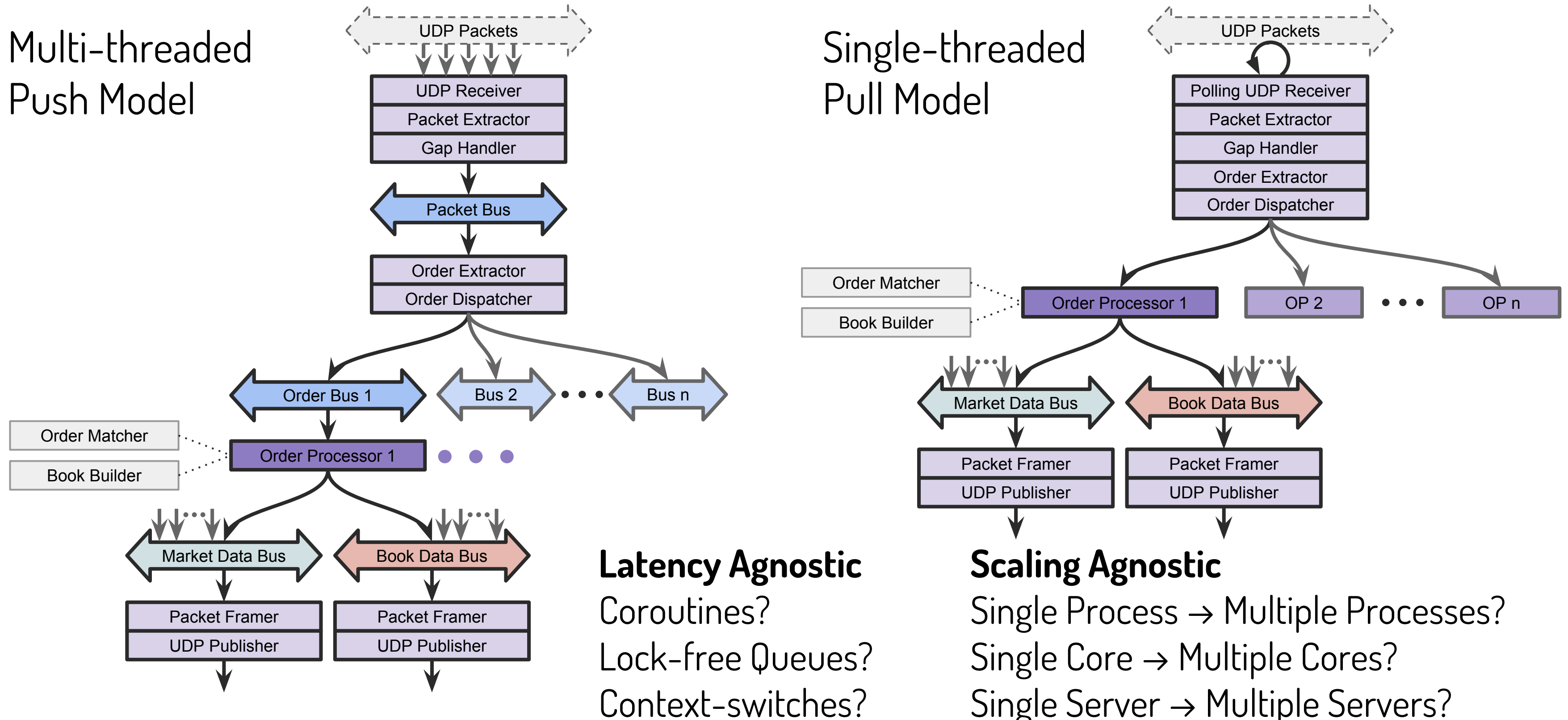


Simple Example



© 2016 Jamie Allsop

Different Performance Trade-offs

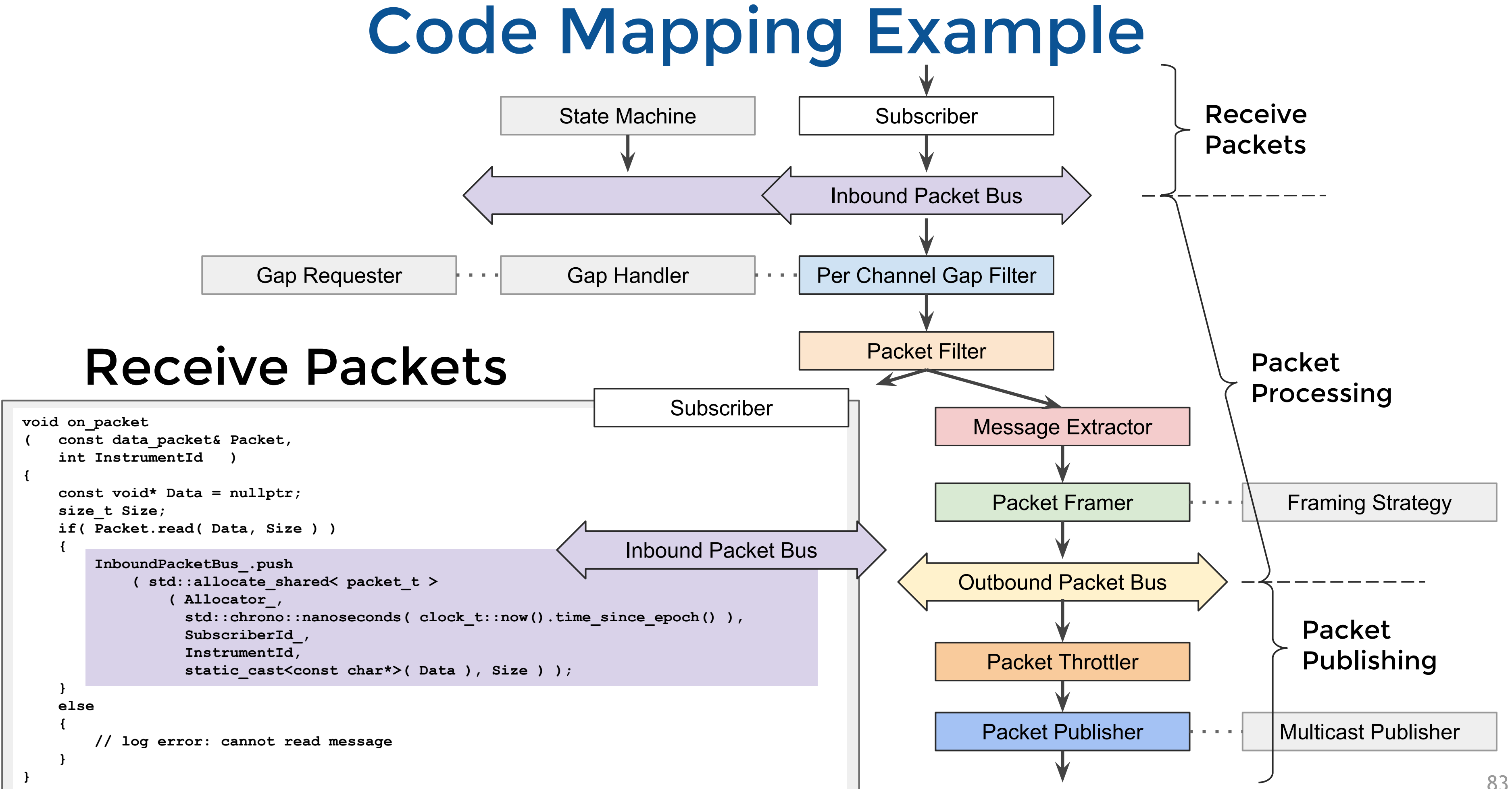


Latency Agnostic
Coroutines?
Lock-free Queues?
Context-switches?

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

© 2016 Jamie Allsop

Code Mapping Example




```
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

```
void process( const shared_inbound_packet& InboundPacket )
{
    if( InboundPacket->seq_num() == ExpectedSeqNum )
    {
        ExpectedSeqNum = InboundPacket->seq_num() + InboundPacket->header().num_msgs();
        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( InboundPacket->seq_num() > ExpectedSeqNum )
    {
        ExpectedSeqNum = GapHandler_.handle_unexpected_packet( InboundPacket, ExpectedSeqNum, ChannelId );
    }
    else if( InboundPacket->seq_num() < ExpectedSeqNum )
    {
        // log and ignore
    }
}
```

Inbound Packet Bus

Per Channel Gap Filter

Packet Filter

Message Extractor

Packet Framer

Outbound Packet Bus

Outbound Packet Bus

Lastly...

Publish Packets

```
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 );
}
```

Outbound Packet Bus

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

© 2016 Jamie Allsop

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

Tier	Examples
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

Projects, Deployments and Configuration

Product Libraries

Domain Specific Libraries

Domain Agnostic Libraries

Standard & Thirdparty Libs

Quality and Technical Knowledge required (↓)

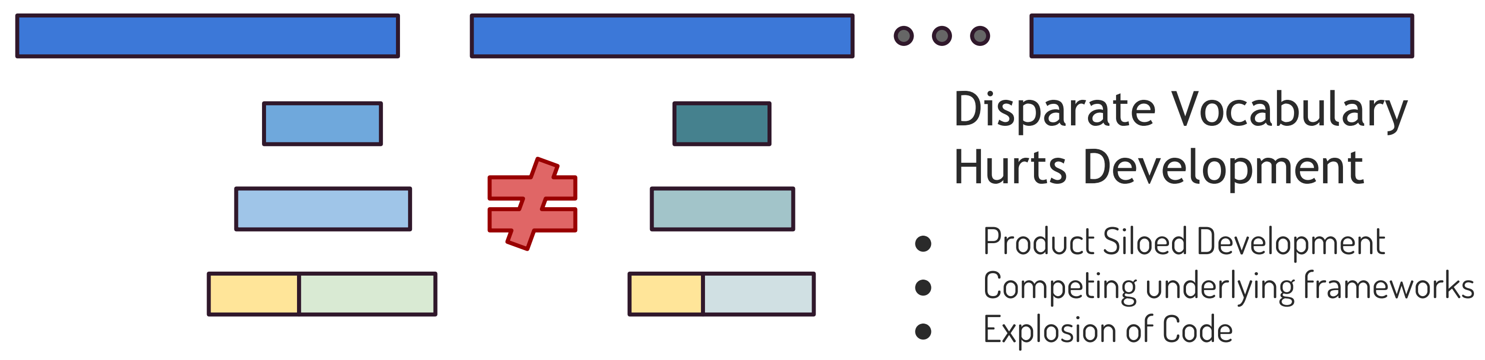
Visibility on progress from a business perspective (↑)

Domain Specific Knowledge (↑)

Domain Agnostic Expertise needed (↓)

Team Lead Possibilities (↑)

Contrast with Disparate Vocabulary

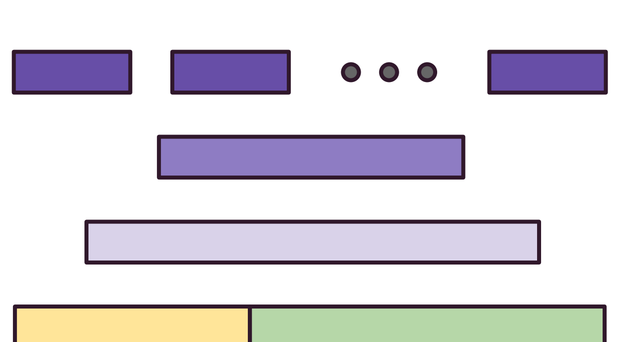


Disparate Vocabulary Hurts Development

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

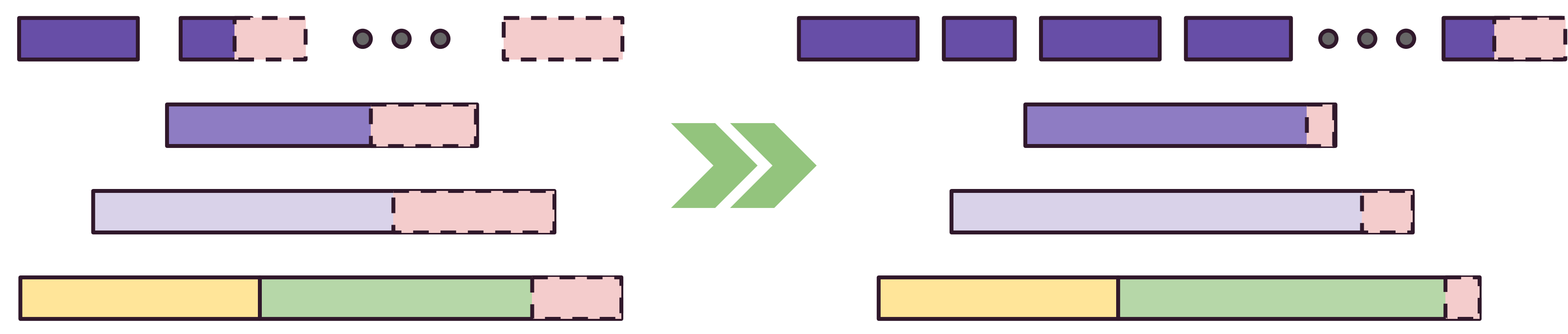
With A Common Vocabulary Less is More

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



© 2016 Jamie Allsop

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)

© 2016 Jamie Allsop

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

© 2016 Jamie Allsop

Thank you for Listening

Questions?

jamie.allsop@clearpool.io
@clearpoolio