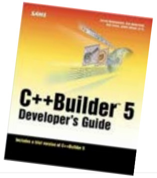
jax FINANCE

Algorithmic Architecture, Real Time AI and the search for Alpha

Jamie Allsop — London 2018

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



BSi | C++ Panel

pypi/cuppa

Market Microstructure

agile-trac
Agile Integrated Project Collaboration

U.S. SECURITIES AND EXCHANGE COMMISSION
MCMXXXIV

clearpool.io
Real Insight, Real Liquidity

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

May 6 Flash Crash

yedup *Be First To Know*

ended up at

NYSE Euronext
Powering the exchanging world.℠

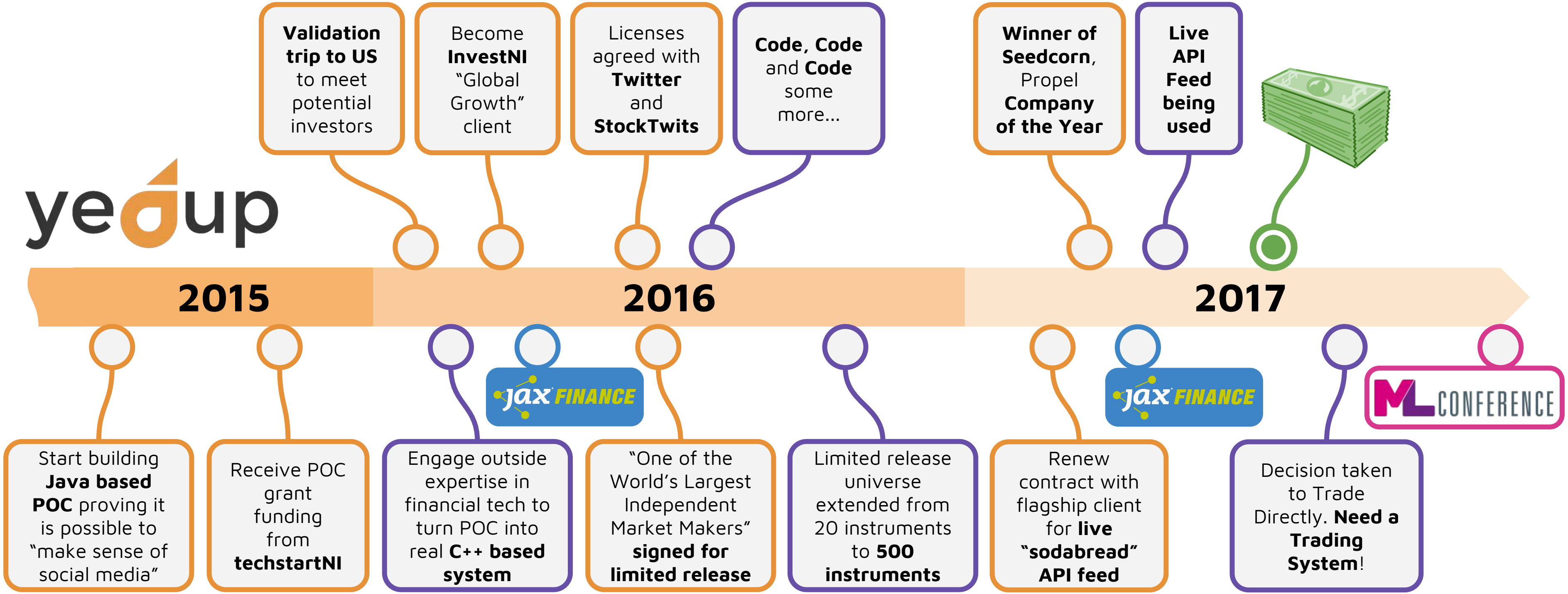
10,750
10,500
10,250
10,000
998.5

TP ICAP

Head of Technology

Keeping Markets in Motion

Backstory…



Start building **Java based POC** proving it is possible to "make sense of social media"

Receive POC grant funding from **techstartNI**

Validation trip to US to meet potential investors

Engage outside expertise in financial tech to turn POC into real **C++ based system**

Become **InvestNI** "Global Growth" client

jax FINANCE

Licenses agreed with **Twitter** and **StockTwits**

"One of the World's Largest Independent Market Makers" **signed for limited release**

Code, Code and **Code** some more…

Limited release universe extended from 20 instruments to **500 instruments**

Renew contract with flagship client for **live "sodabread" API feed**

Winner of Seedcorn, Propel **Company of the Year**

jax FINANCE

Live API Feed being used

Decision taken to Trade Directly. **Need a Trading System**!

ML CONFERENCE

1

Basic Problem — To Trade against Signals Derived from Social Media

Making Sense of Social Media …

It's a New Language

Abbreviations, acronyms, emojis, emphatic spelling. Algorithms are required to **learn the meaning of non-standard strings** and new words as they appear.

Constantly Growing

Around **1500 new words and phrases** appear in the global conversation **each day**. Over time this adds up. Words also have **different meanings in different contexts**, topic domains, and countries.

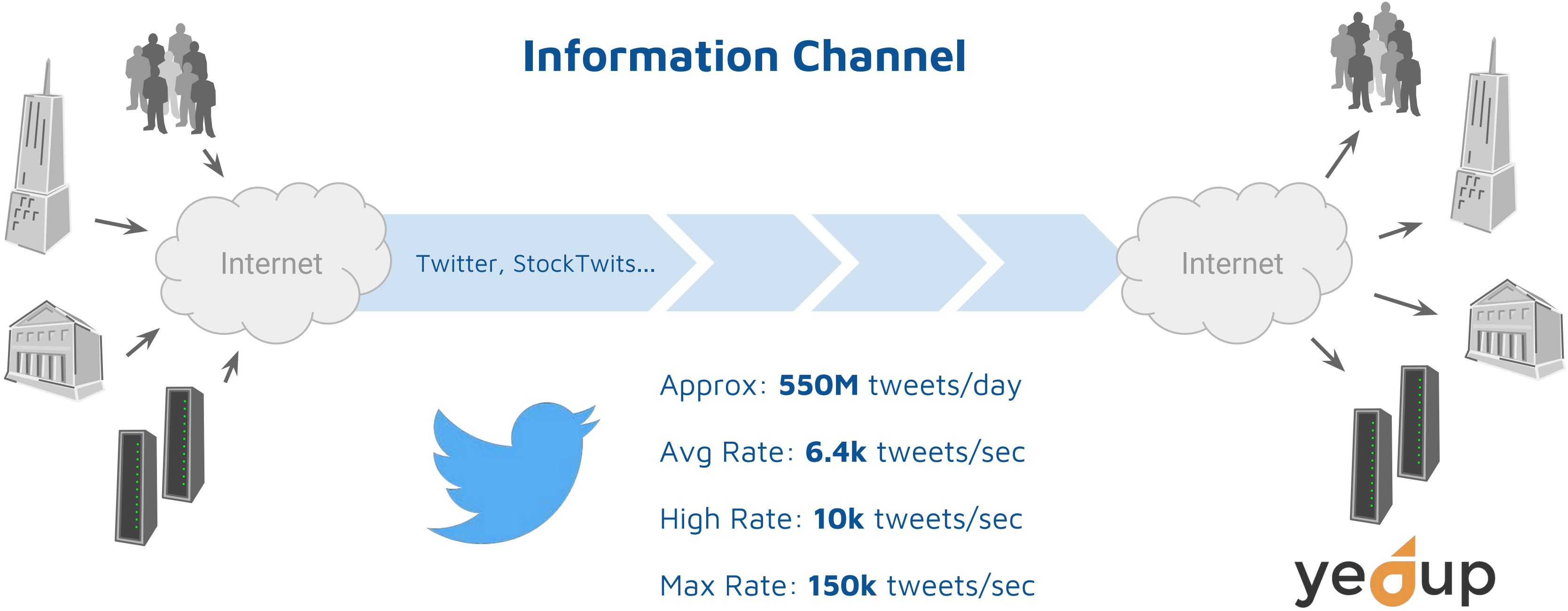
Forever Changing

Algorithms which adapt to keep pace with the way the world expresses itself are needed. These should be **context aware** and be suitable for all **domain specific** applications.

… is not easy!

Well, what is it really?

Information Channel



Approx: **550M** tweets/day

Avg Rate: **6.4k** tweets/sec

High Rate: **10k** tweets/sec

Max Rate: **150k** tweets/sec

… and we want to trade on this

What we Aim For

Real-Time

Process more than 100k social media posts per second, with industry leading low latency. Always deliver results in real time.

Adaptive

Use artificial intelligence to evolve continually to reflect the fluid nature of social media expression and keep pace with the latest lingo.

Language Agnostic

Work with all major languages and script systems. Be able to cover social media channels in Europe, Middle East, Africa, Asia and the Americas.

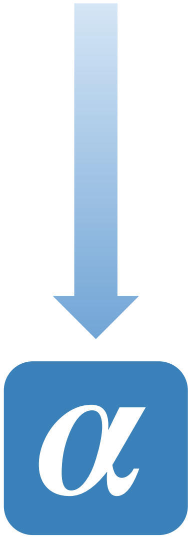
Domain Aware

Machine learning can also capture the domain-specific meanings of certain words and phrases, so the true meaning of what is said is understood in its proper context.

Using a channel

- ✓ I know what I want to hear about so I'll listen for that

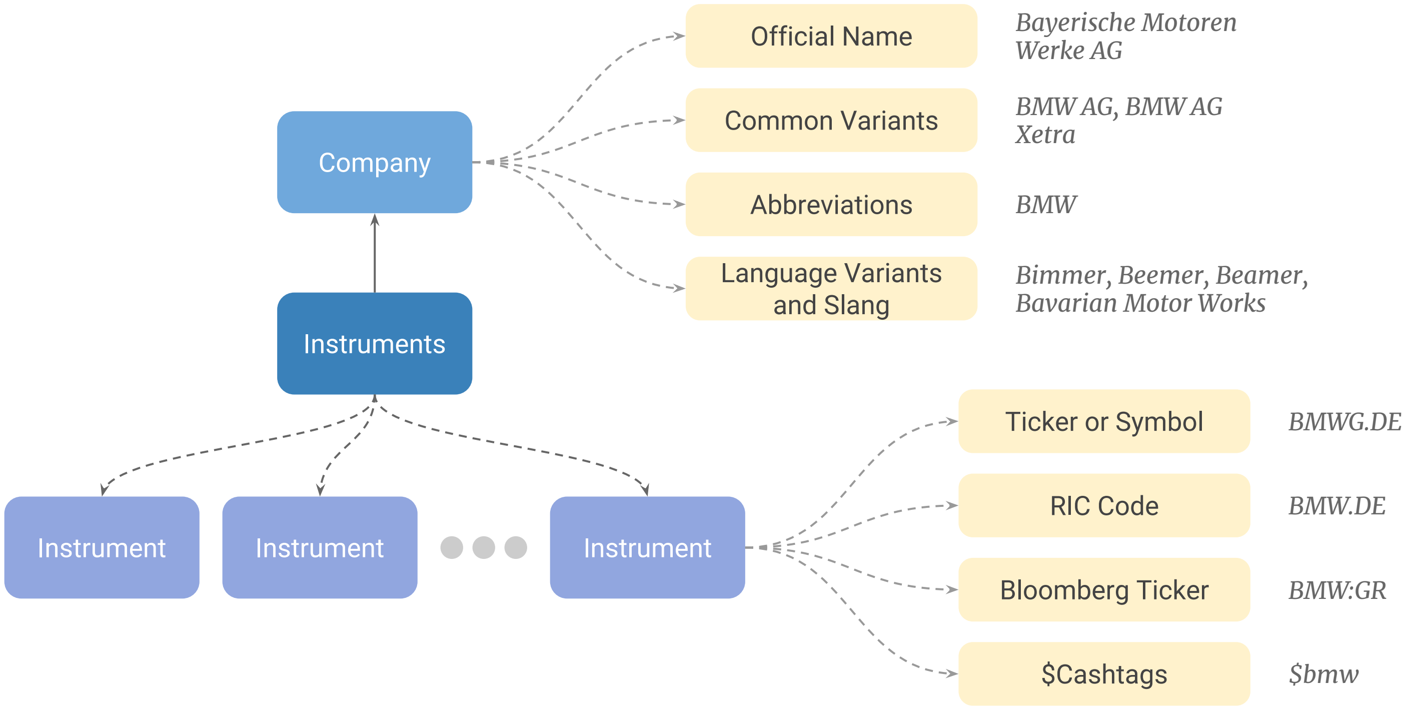
- ⚠ I don't know exactly what I want to hear about but I'll know it when I see it



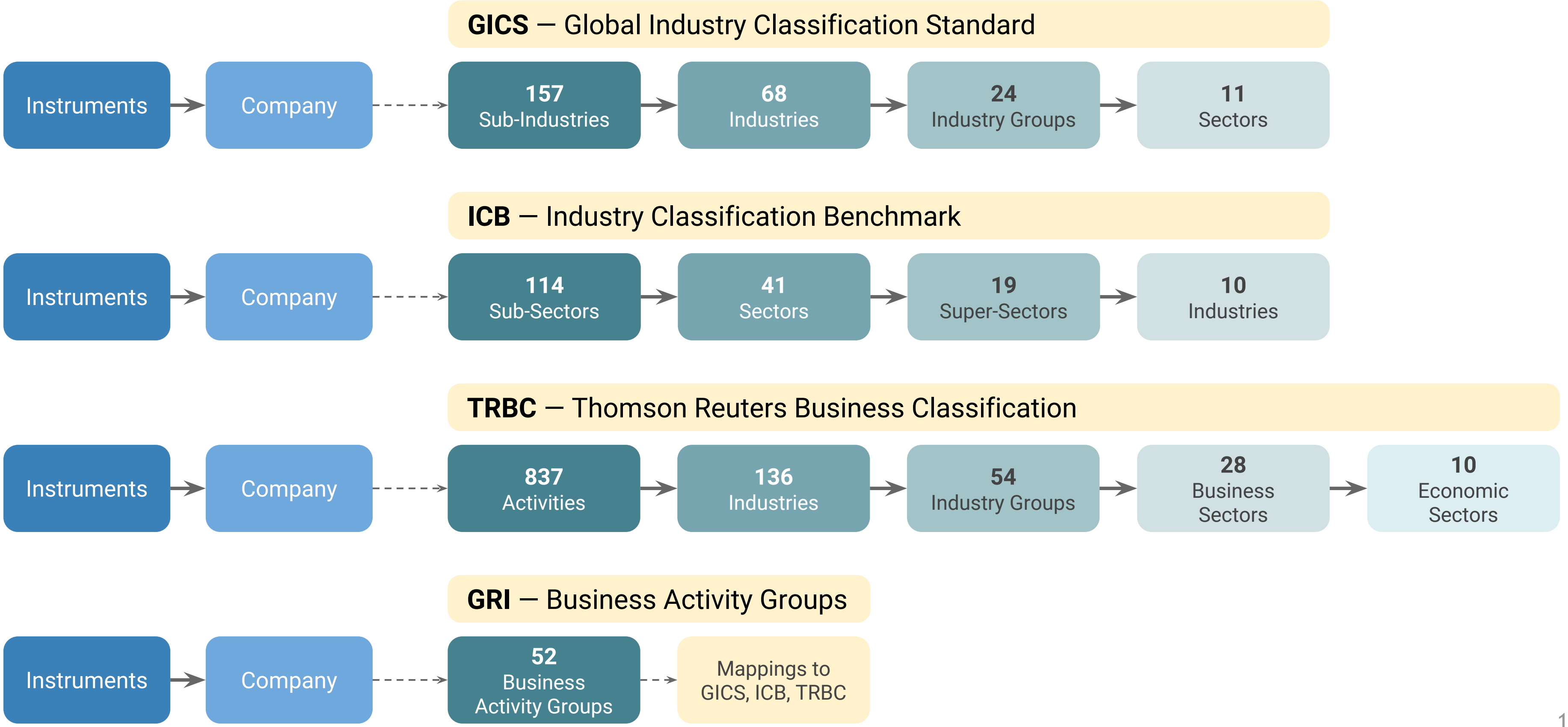
Say What?

- What was said?
- What was it about?
- What was the opinion expressed?
- Who said what was said?
- Who cared about what was said?
- Has anyone said this before?

Which Instrument or Company?



We Trade Financial Instruments



GICS — Global Industry Classification Standard

Instruments → Company → 157 Sub-Industries → 68 Industries → 24 Industry Groups → 11 Sectors

ICB — Industry Classification Benchmark

Instruments → Company → 114 Sub-Sectors → 41 Sectors → 19 Super-Sectors → 10 Industries

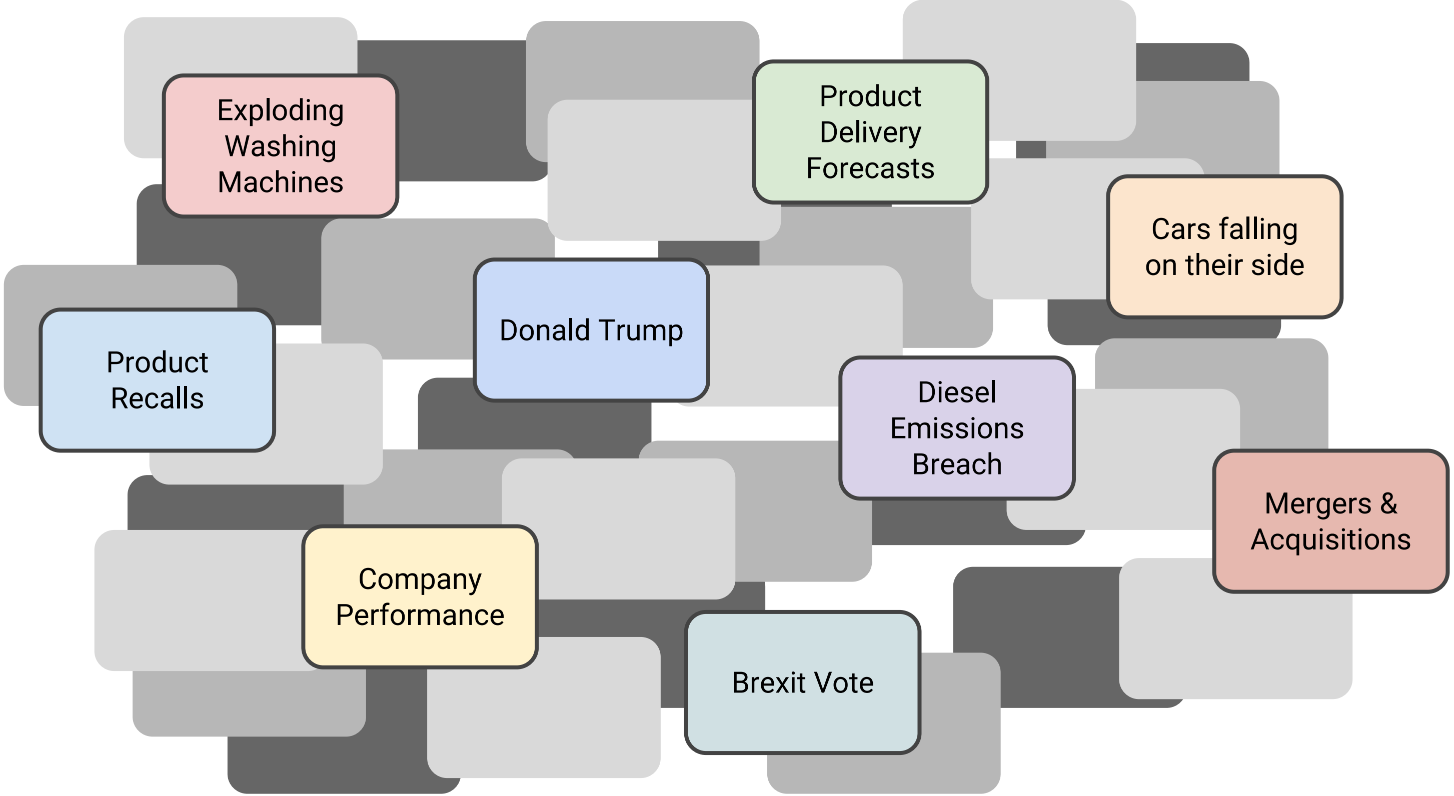
TRBC — Thomson Reuters Business Classification

Instruments → Company → 837 Activities → 136 Industries → 54 Industry Groups → 28 Business Sectors → 10 Economic Sectors

GRI — Business Activity Groups

Instruments → Company → 52 Business Activity Groups → Mappings to GICS, ICB, TRBC

What about it?



Exploding Washing Machines

Product Delivery Forecasts

Cars falling on their side

Donald Trump

Product Recalls

Diesel Emissions Breach

Mergers & Acquisitions

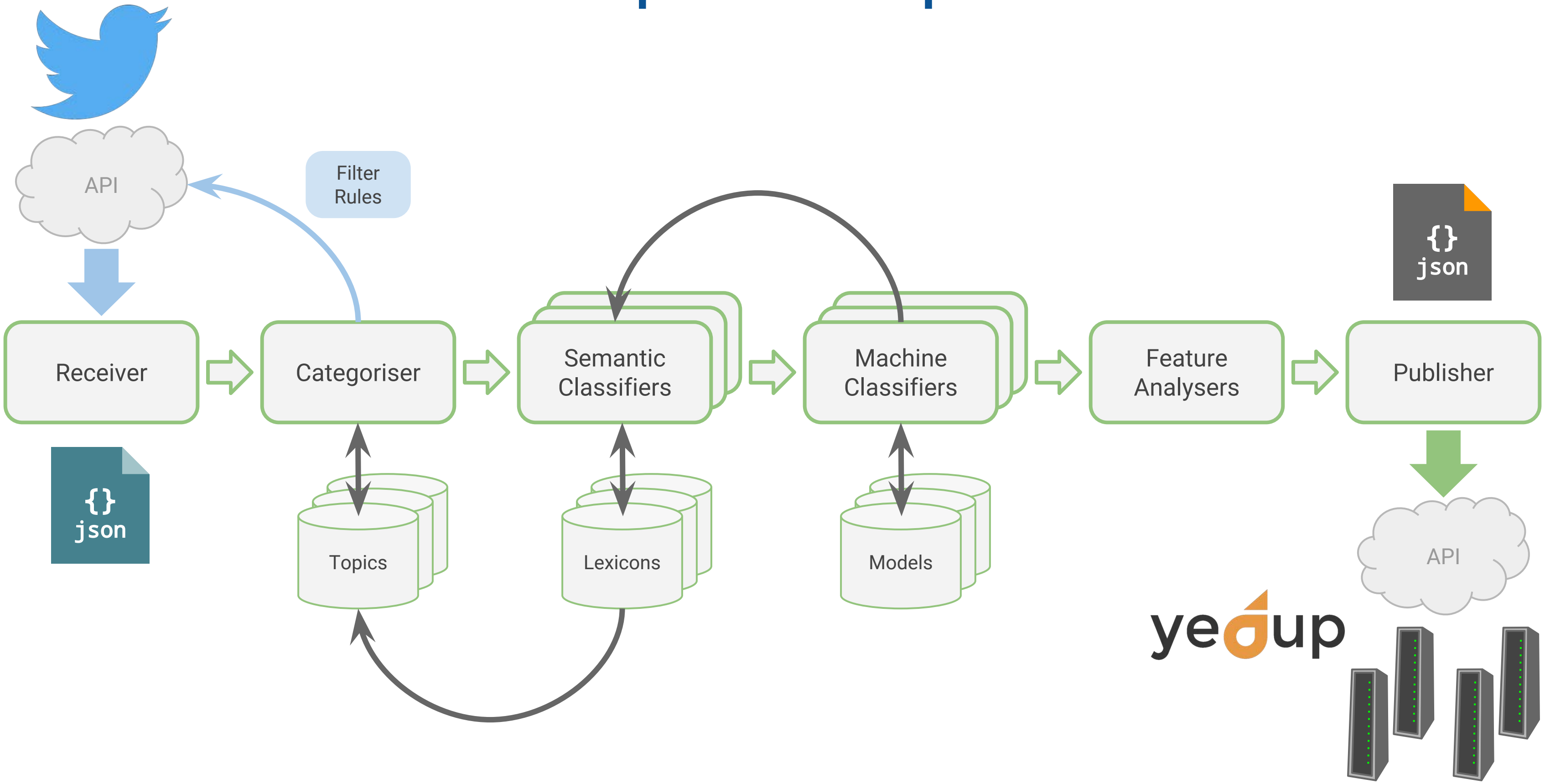
Company Performance

Brexit Vote

2

Solution — Sounds like a good problem for AI?

Conceptual Pipeline



3

Real Problem — Building a Real-time Adaptive System that can Trade

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

— Ackoff 1974

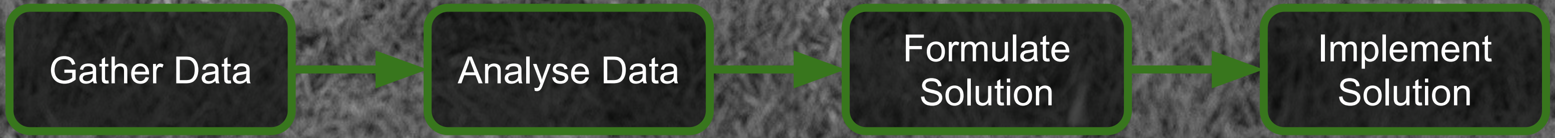
Do we understand problems?

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



Gather Data → Analyse Data → Formulate Solution → Implement Solution

Messes

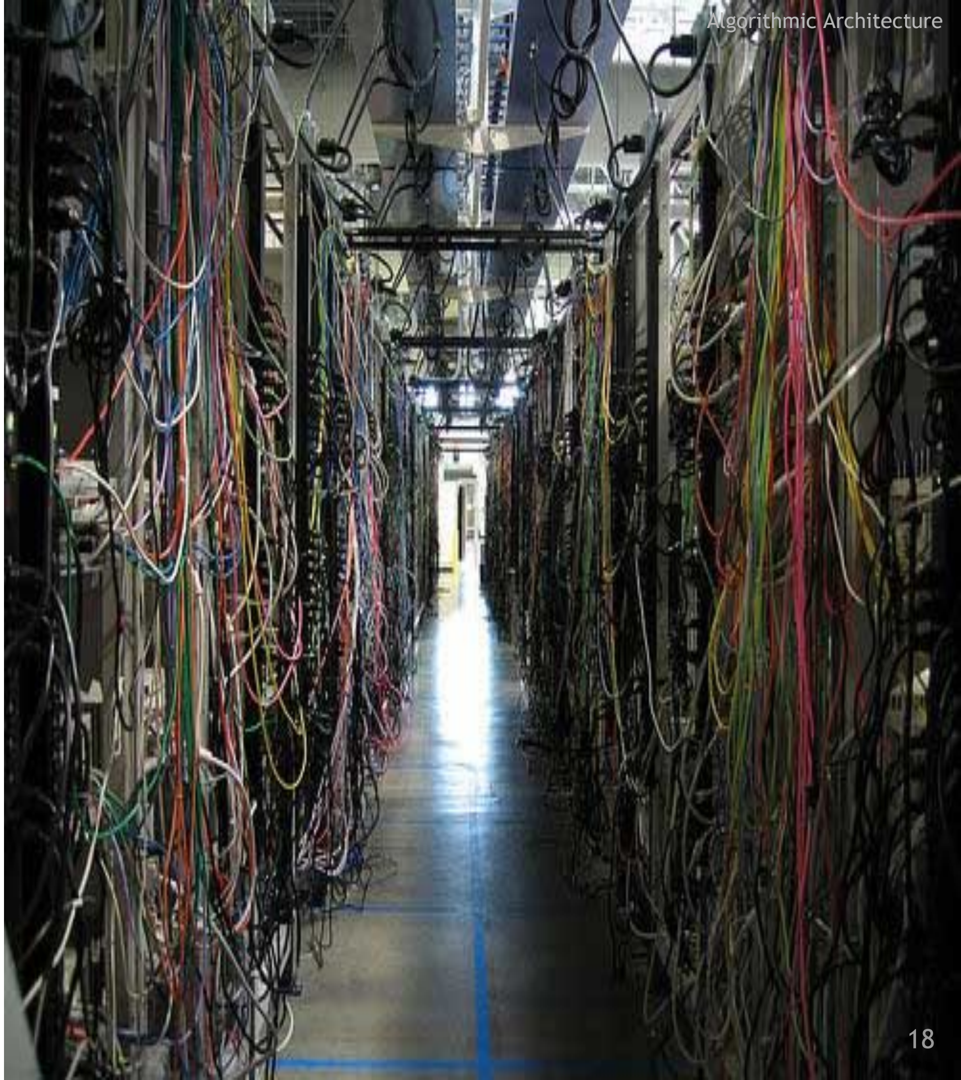
Organised complexity

- clusters of interrelated or interdependent problems

Systems of problems

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

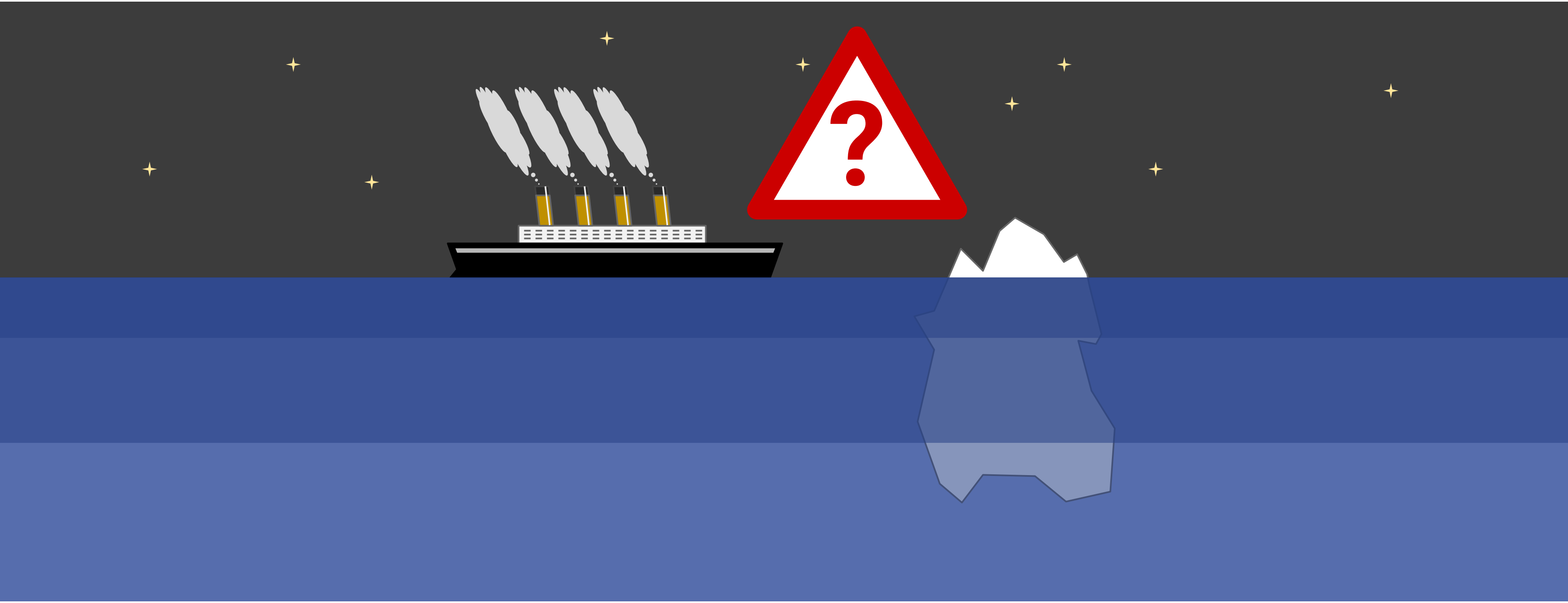
Messes are puzzles – we don’t solve them instead we **resolve their complexities**



Messes are... a Mess

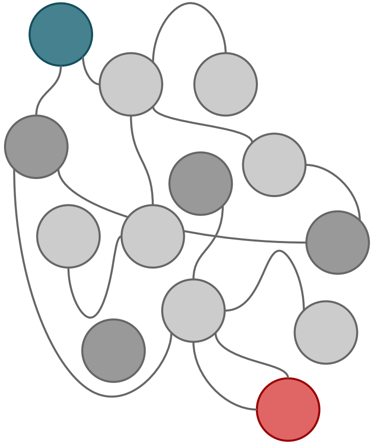
- not sufficient to just break the system into parts and fix components
- instead look for **patterns** of interactions between parts
- beware of identifying a mess as a tame problem—the evolving mess can be even more difficult to deal with
- **interactive complexity**—what can go wrong?
- **coupling**—the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

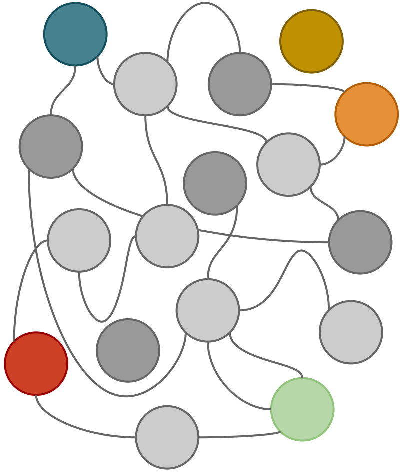
coupling…

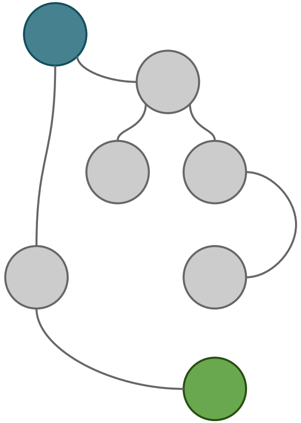


Bugfixing?

Refactoring?







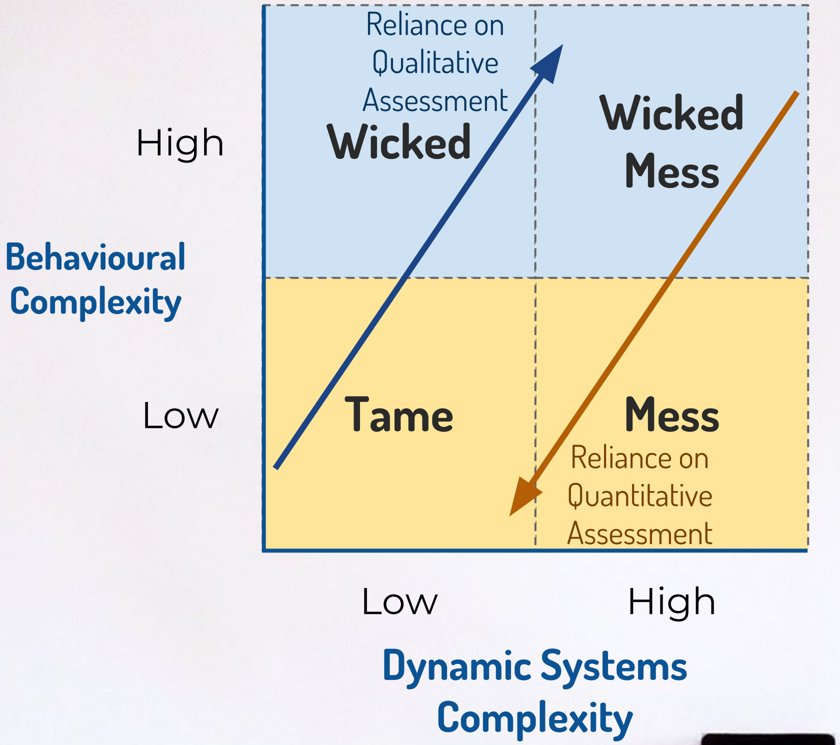
- Conflicting **social** ethics and beliefs
- Smart, informed people **disagree**
- **Divergent** problems with no promise of a solution
- **Evolving** set of **Interlocking** Issues and Constraints
- Constraints **change over Time**
- Many Stakeholders

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

Wicked

Wicked Mess

Tame

Mess

Reliance on Quantitative Assessment

High

Low

Behavioural Complexity

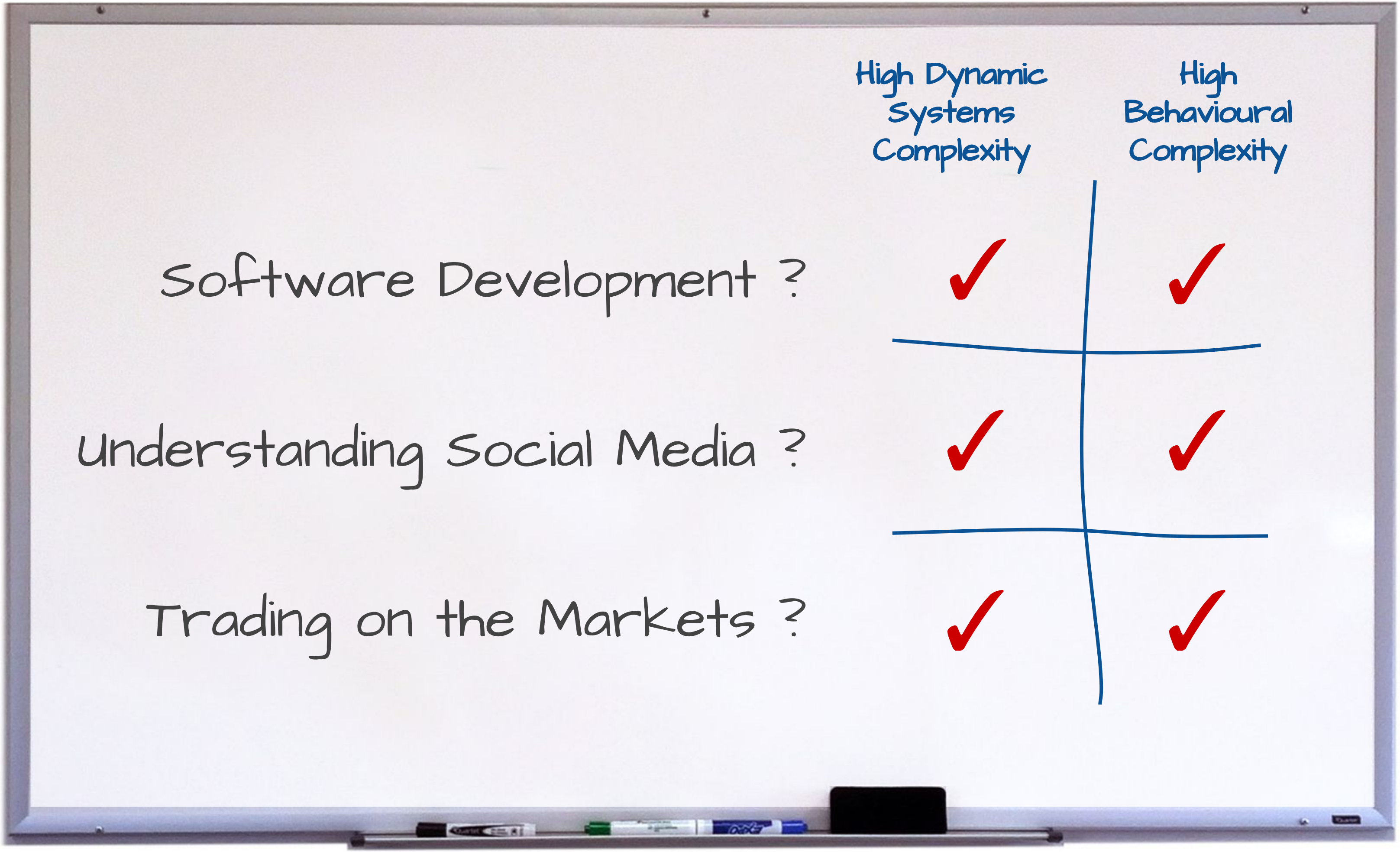
Low

High

Dynamic Systems Complexity

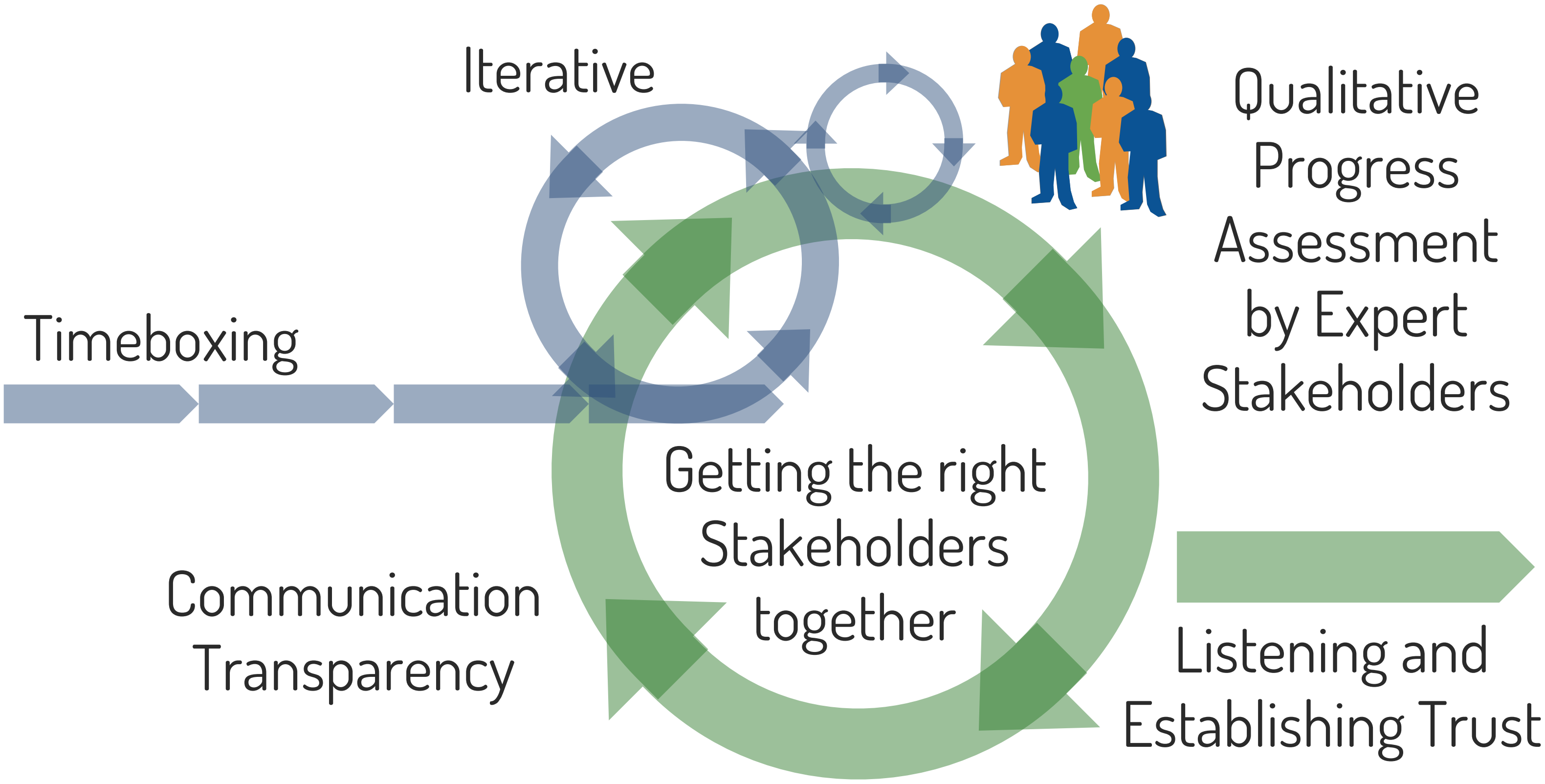
Resolution is Social / Ethical / Political / Moral / Behavioural

Solution is Scientific



	High Dynamic Systems Complexity	High Behavioural Complexity
Software Development ?	✓	✓
Understanding Social Media ?	✓	✓
Trading on the Markets ?	✓	✓

Approaches to Wicked Problems



Iterative

Timeboxing

Qualitative Progress Assessment by Expert Stakeholders

Getting the right Stakeholders together

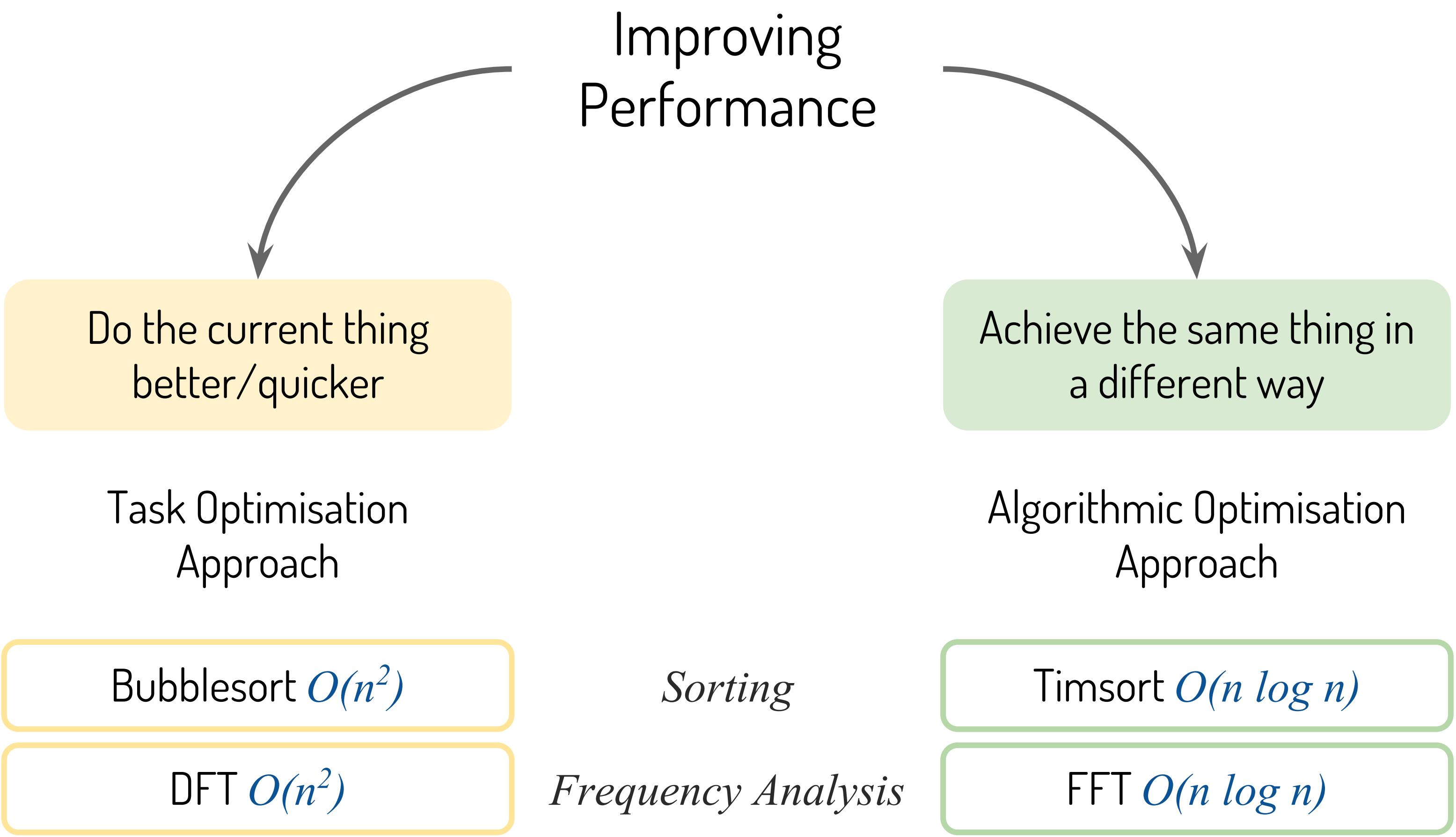
Communication Transparency

Listening and Establishing Trust

Waterfall Solutions are Too Slow to React Effectively

4

What about writing **software with hard constraints** like performance?



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

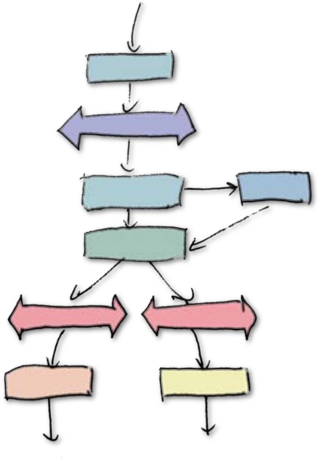
FFT $O(n \log n)$

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
- Pushes us towards **Task Optimisation**

We want to reason about this…



But we can only see this…

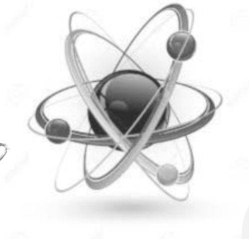
What we really want is an Architecture that

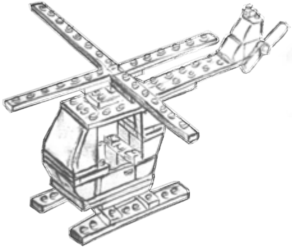
- is based on well defined building blocks
- favours algorithmic optimisation
- has a clear mapping to code
- allows an optimal solution
- is adaptive to a changing environment

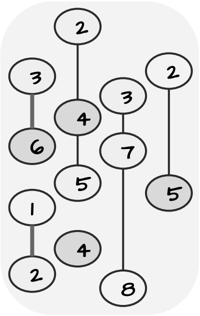
an "Algorithmic Architecture"

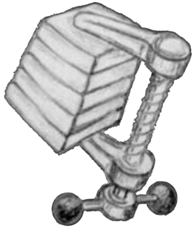
We Achieve This By

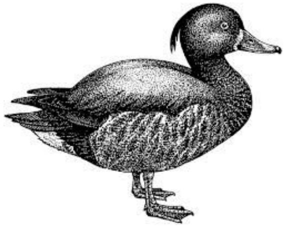
- ➤ Exposing a shared Vocabulary *that can map to code and is*
- ➤ Decomposable
- ➤ Composable
- ➤ Independently Orderable
- ➤ Compactible
- ➤ Substitutable







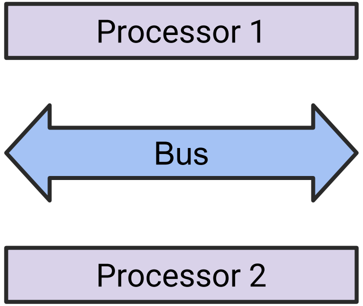




Towards Algorithmic Architecture

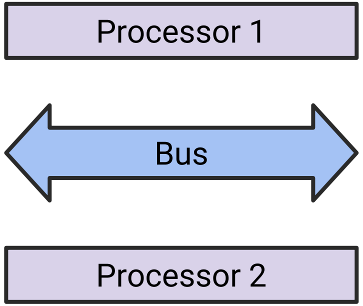
- Define **building block vocabulary elements**

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

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

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



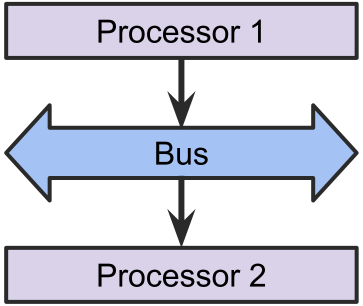
Towards Algorithmic Architecture

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



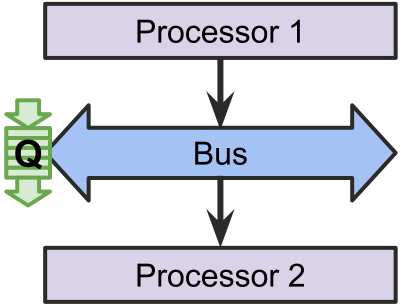
Towards Algorithmic Architecture

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



Towards Algorithmic Architecture

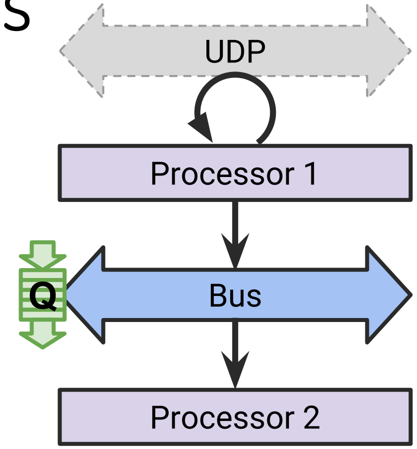
- Define building block vocabulary elements
- Avoid shared state
- Favour message passing
- Make **synchronisation points explicit** in the architecture



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

Towards Algorithmic Architecture

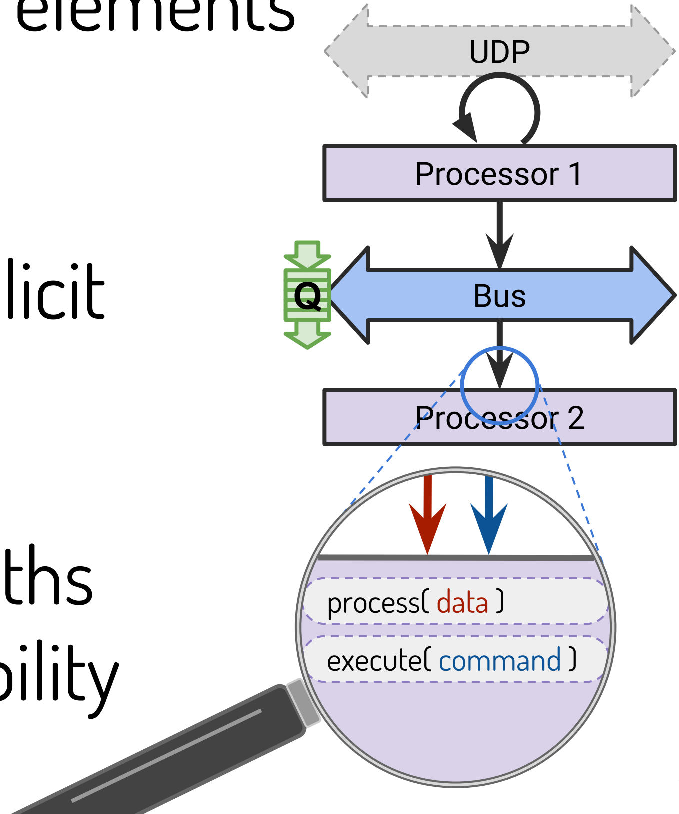
- Define building block vocabulary elements
- Avoid shared state
- Favour message passing
- Make synchronisation points explicit in the architecture
- Support **push** and **pull** models



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

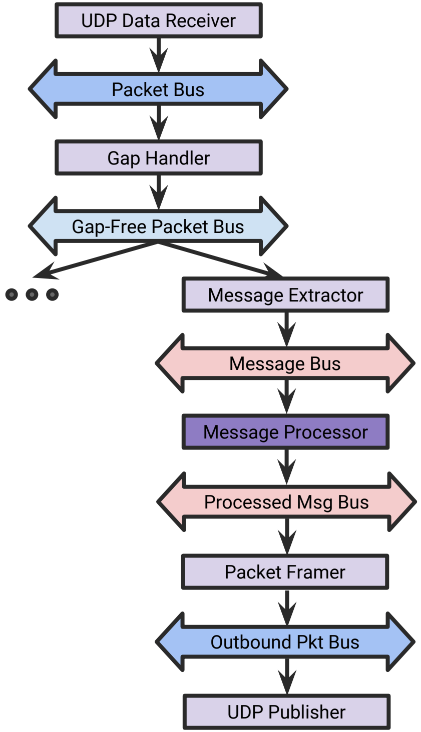
Towards Algorithmic Architecture

- 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 adaptability

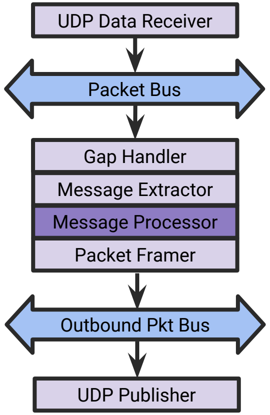


Simple Example

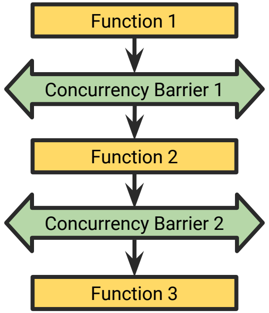
1 Design Using a
Real Vocabulary of
Real Components



2 Compact Architecture by
removing conceptual
components

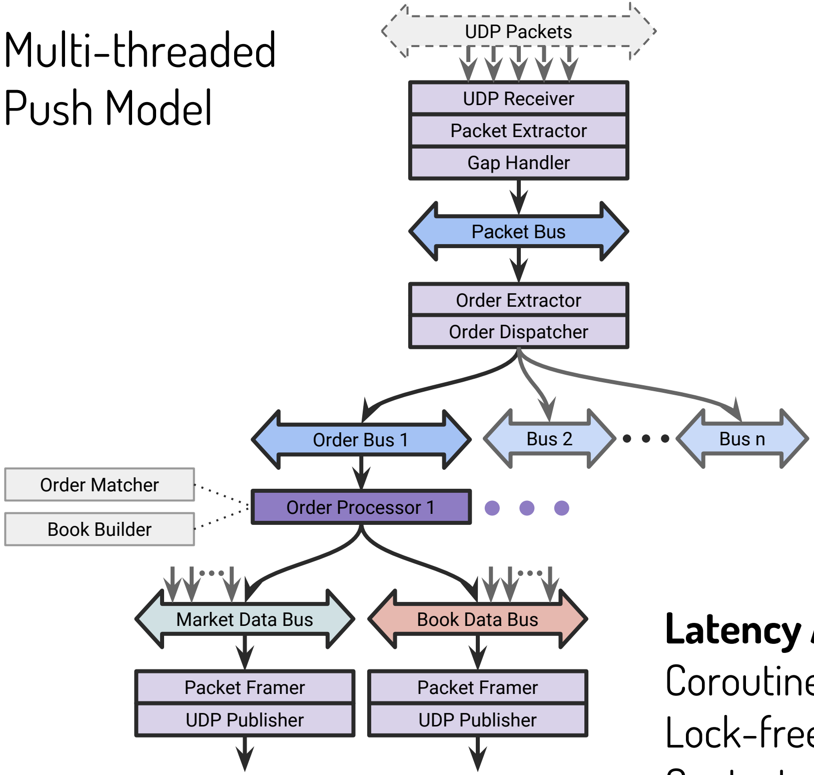


3 Compile to Optimised
Implementation with zero
abstraction cost

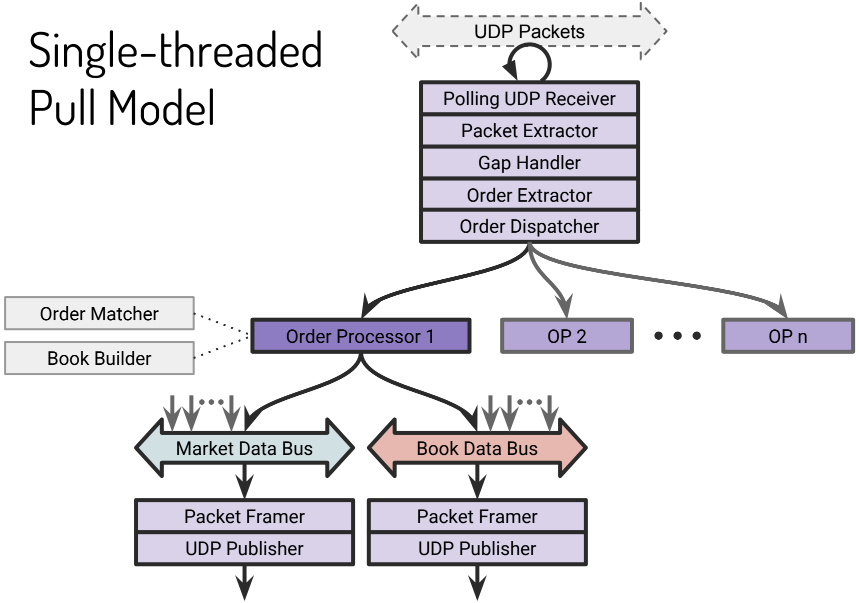


Different Performance Trade-offs

Multi-threaded
Push Model



Single-threaded
Pull Model

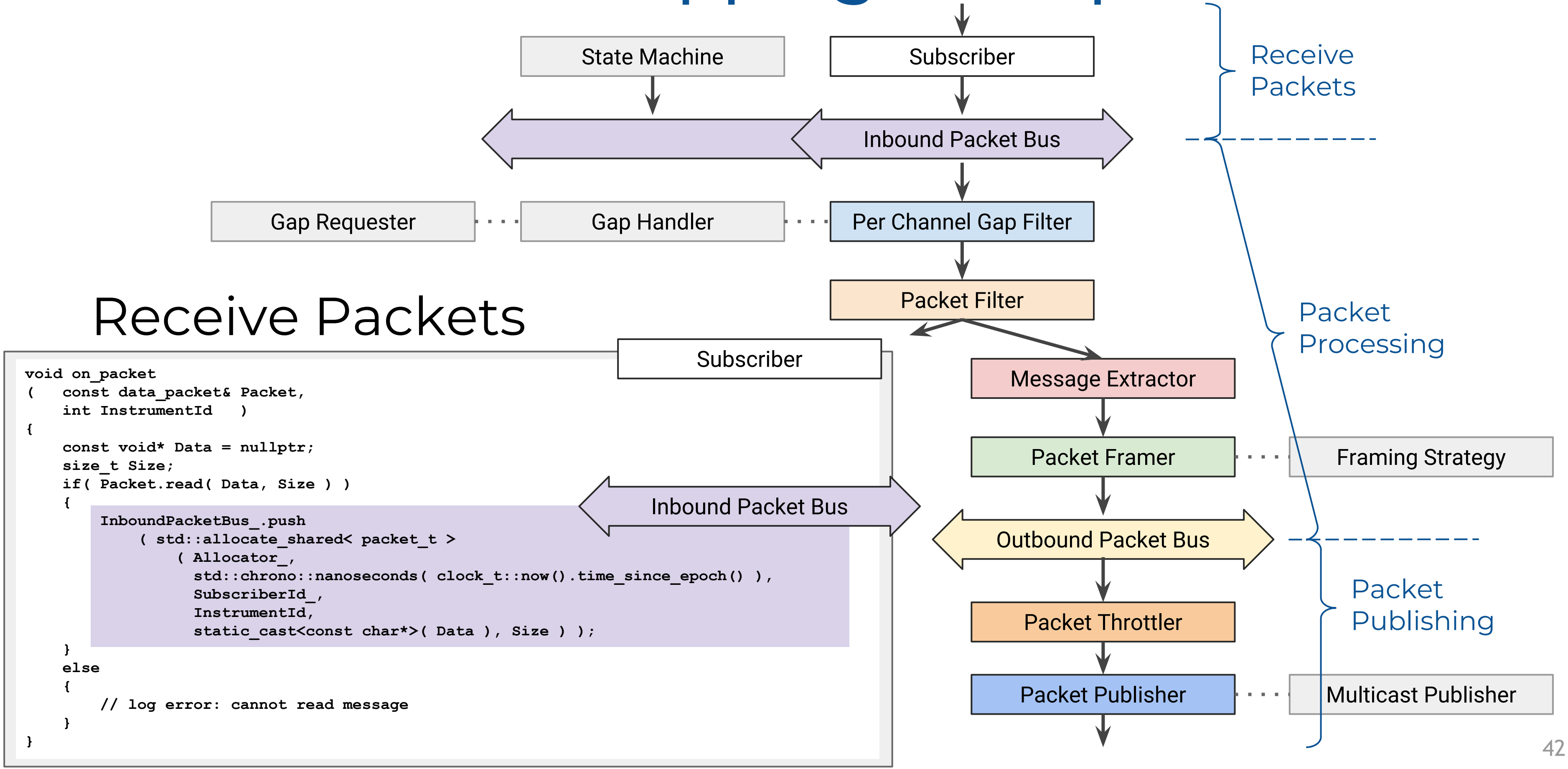


Latency Agnostic
Coroutines?
Lock-free Queues?
Context-switches?

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

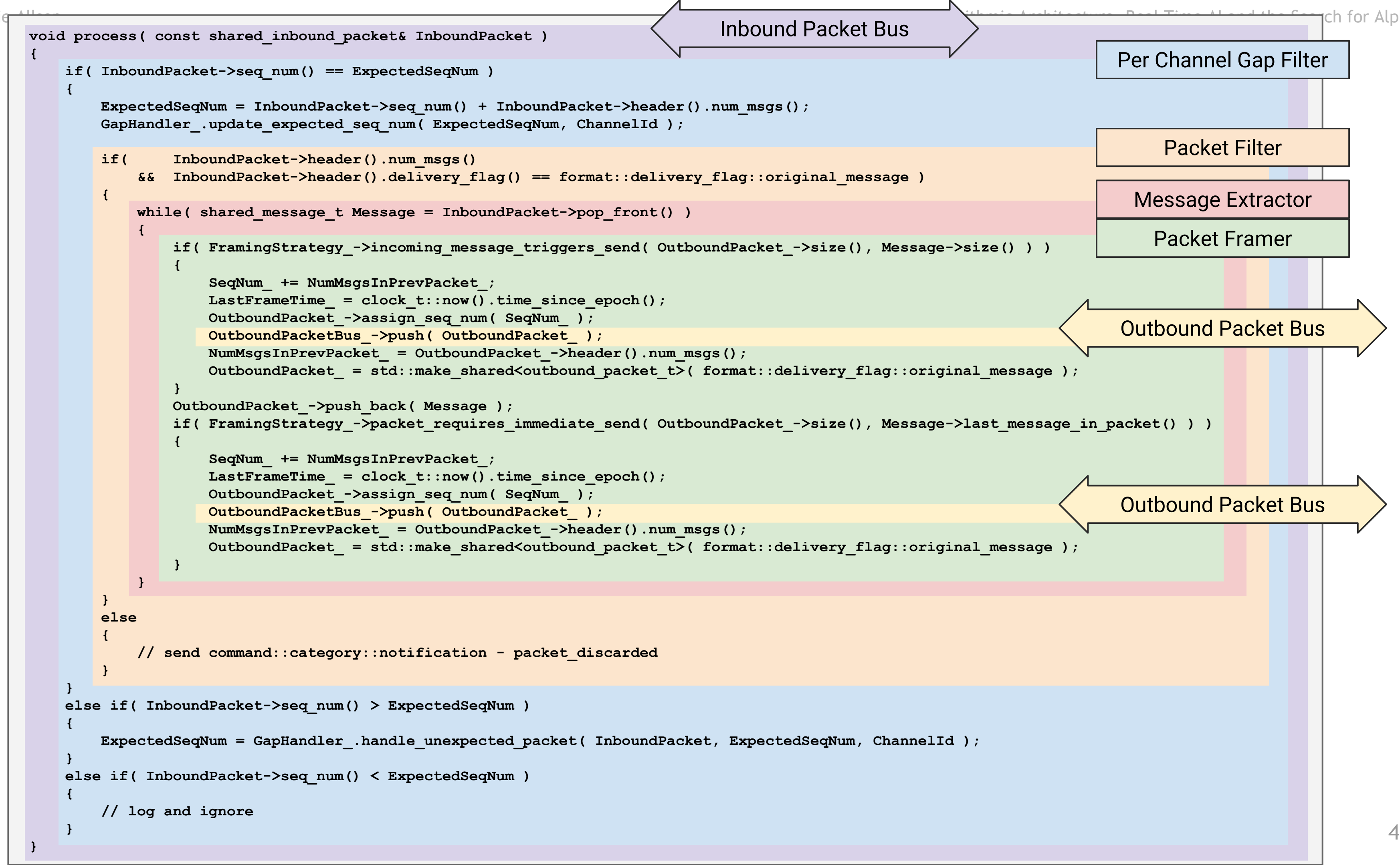
Code Mapping Example

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



```
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

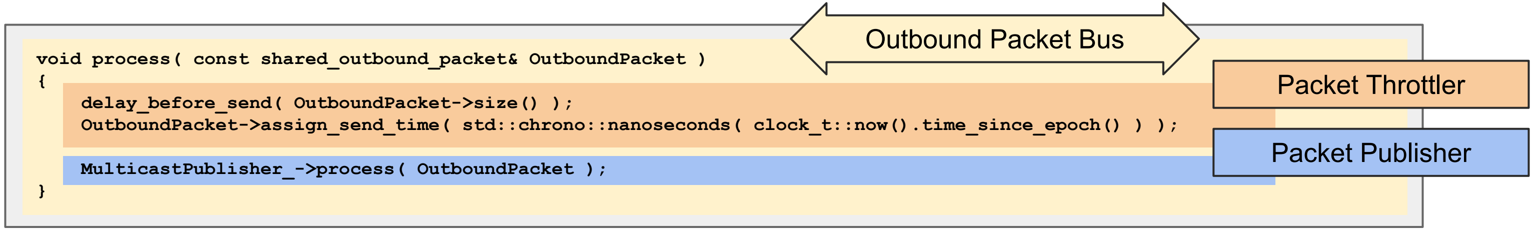
Packet Framer

Outbound Packet Bus

Outbound Packet Bus

Lastly...

Publish Packets



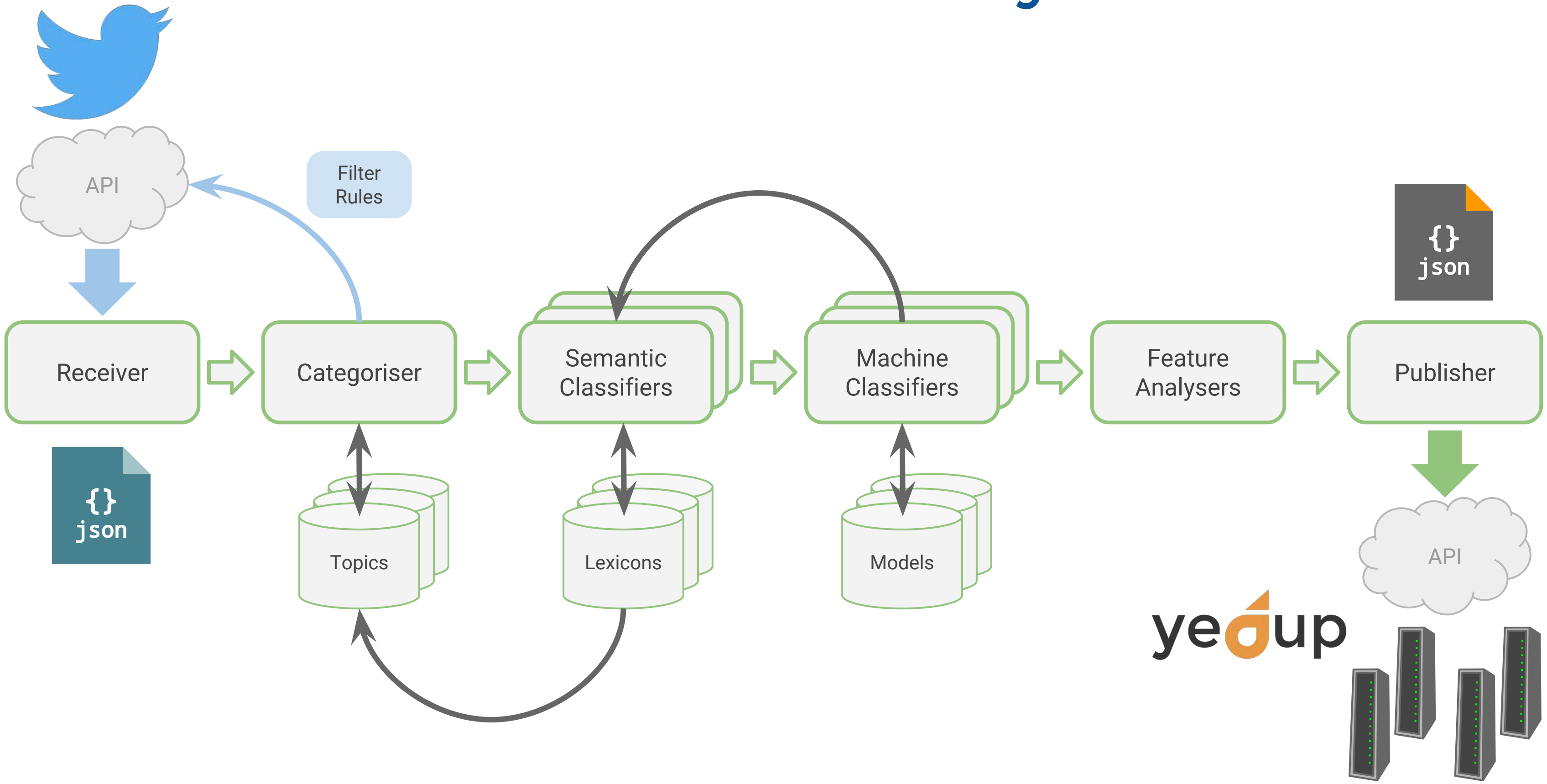
```
void process( const shared_outbound_packet& OutboundPacket )
{
    delay_before_send( OutboundPacket->size() );
    OutboundPacket->assign_send_time( std::chrono::nanoseconds( clock_t::now().time_since_epoch() ) );

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

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

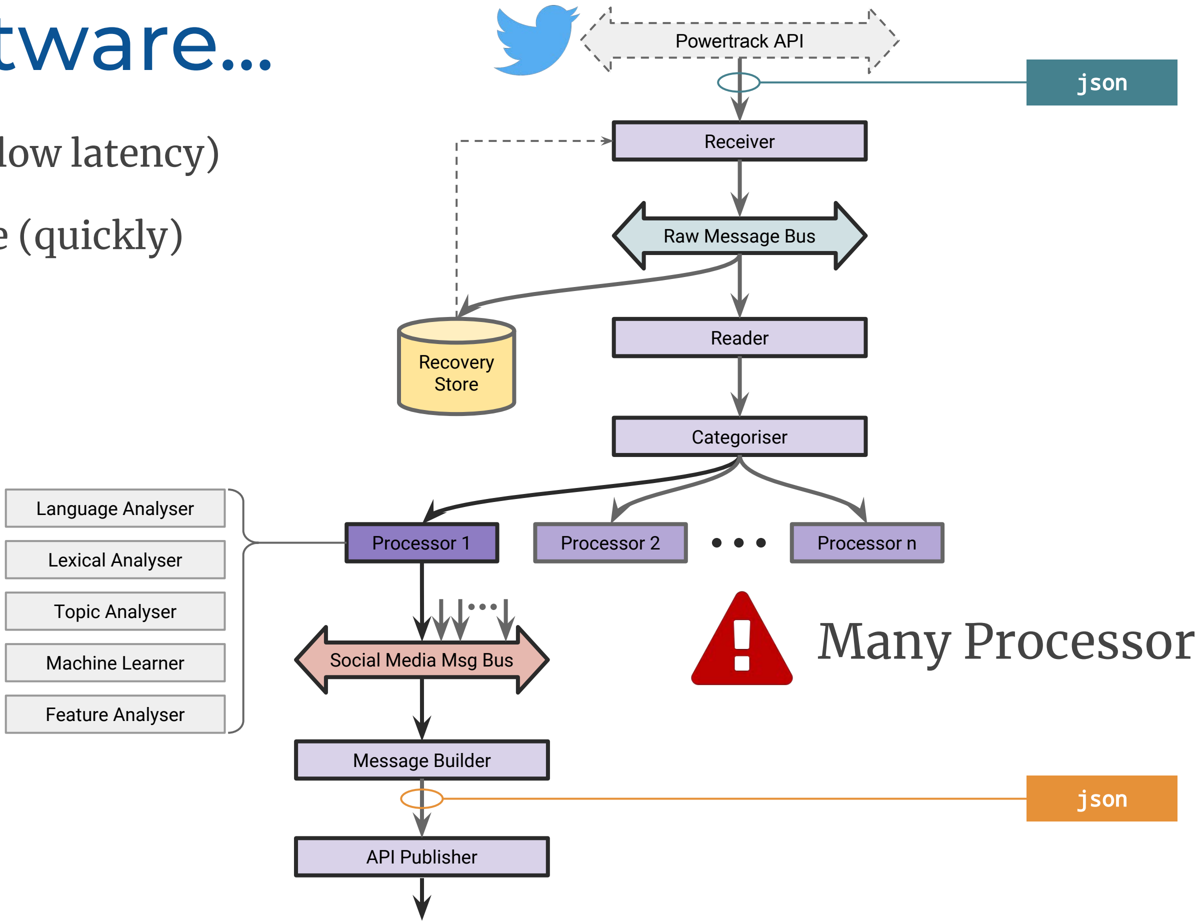
5
Algorithmic Architecture, Real Time AI and the Search for Alpha…

Build a Real-Time System



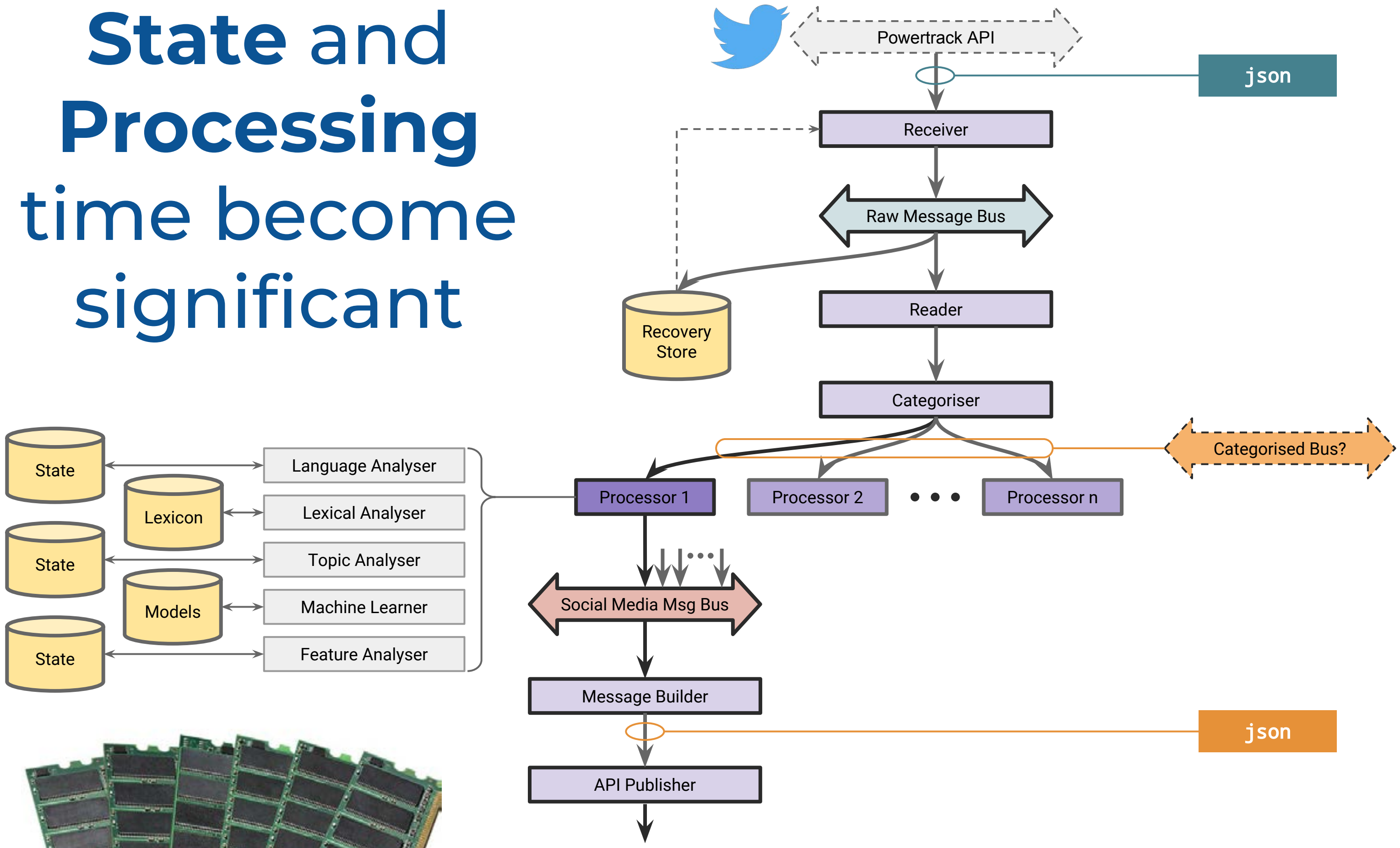
As software...

- ✓ Real Time (low latency)
- ✓ Recoverable (quickly)
- ✓ Adaptive
- ✓ Repeatable



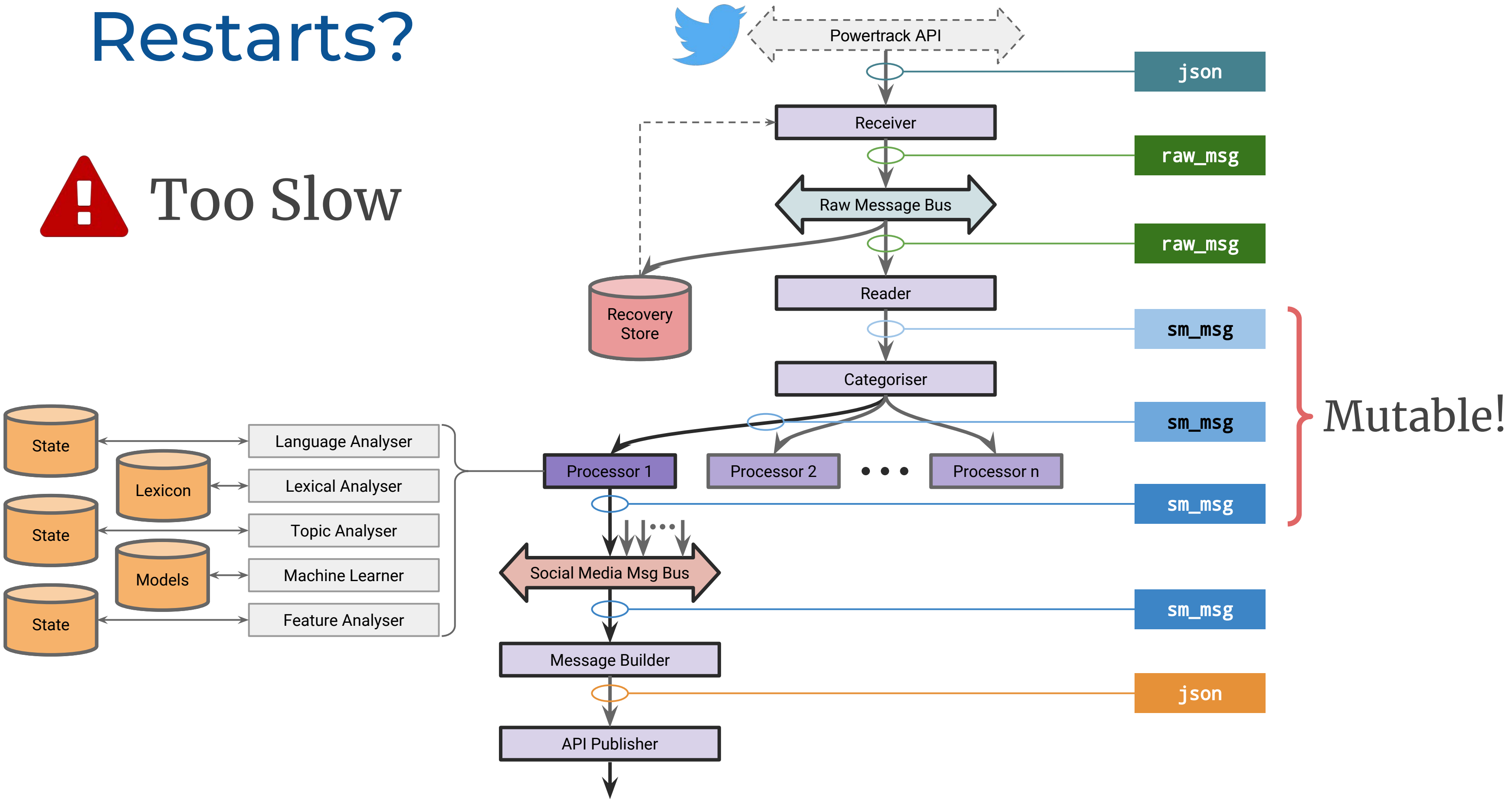
Many Processors Needed

State and **Processing** time become significant

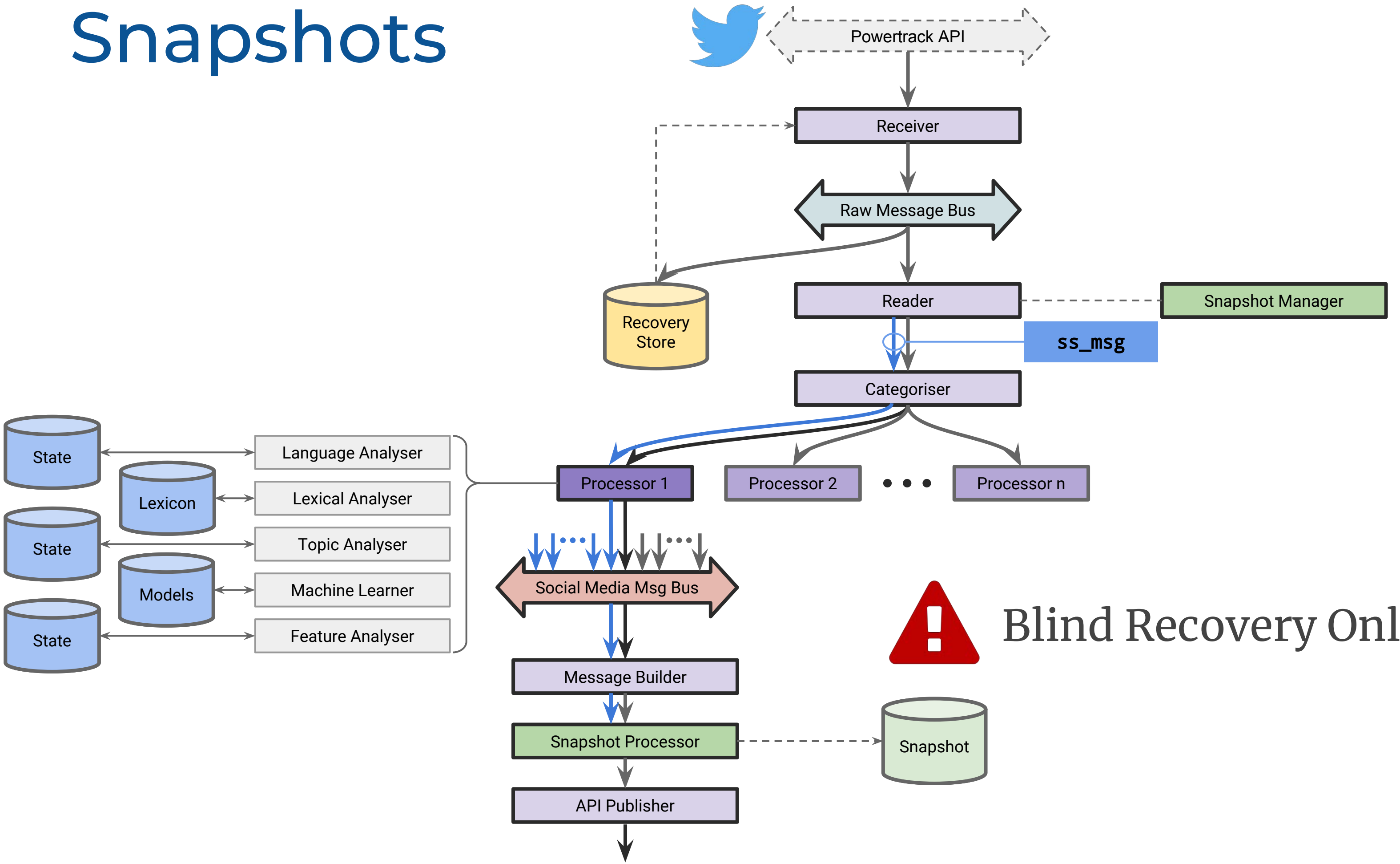


Restarts?

⚠ Too Slow



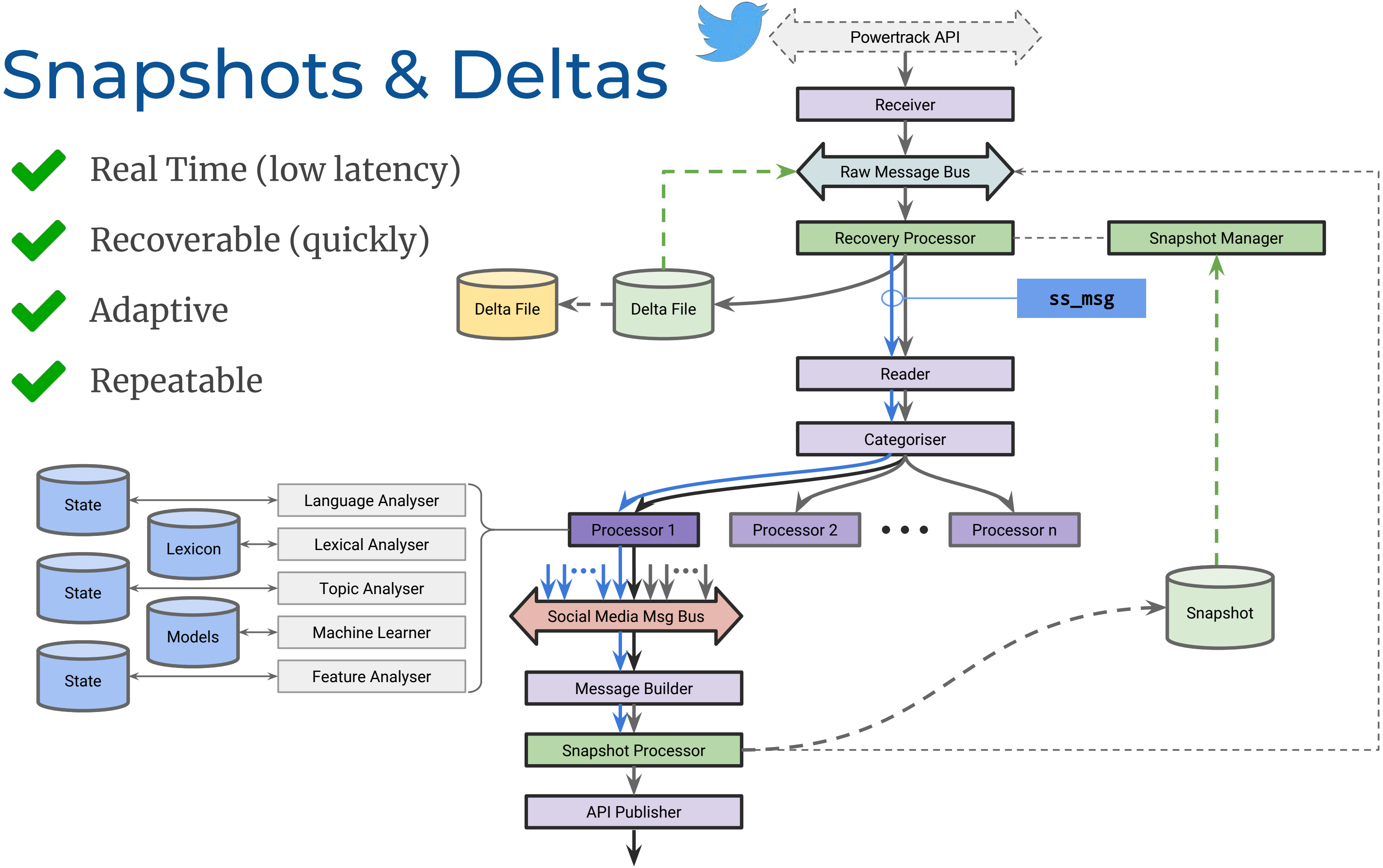
Snapshots



Blind Recovery Only

Snapshots & Deltas

- ✓ Real Time (low latency)
- ✓ Recoverable (quickly)
- ✓ Adaptive
- ✓ Repeatable



March 2017

Profit = 3-6 %

Sharpe Ratio = 11+

jax FINANCE

Thank you

clearpool.io

Real Insight, Real Liquidity

jamie.allsop@clearpool.io