PUSH
TECHNOLOGY

TechMesh
London
December 2012

- Distributed Systems / HPC guy.

- **Chief Scientist** :- at **Push Technology**

- Alumnus of :-

  Motorola, IONA, Betfair, JPMC, StreamBase.

- School: Trinity College Dublin.
  - BA (Mod). Comp. Sci.+
  - M.Sc. Networks & Distributed Systems

- Responds to: Guinness, Whisky

About me?

Darach@PushTechnology.com

Conversational
Big Data

Big Data. The most important V is Value.

- 4Vs:
  - Volume
  - Velocity
  - Variety
  - Variability

- "It's the connections that matter most"
  - David Evans, Cisco

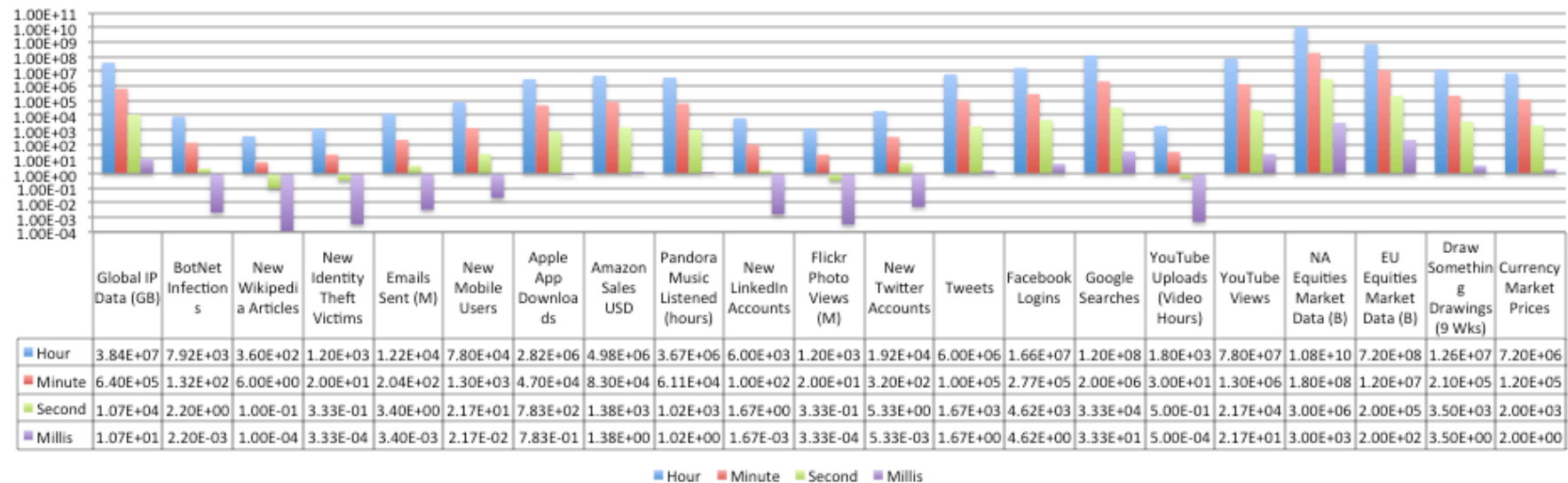| Number of Internet objects	Number of theoretical connections (0.001%)
1	-
100	0
1,000	5
1,000,000	4,999,995
1,000,000,000	4,999,999,995,000
(Today) 10,000,000,000	499,999,999,950,000
(By 2020) 51,597,803,520	13,311,666,640,184,600

Connect all the things

An Internet Minute
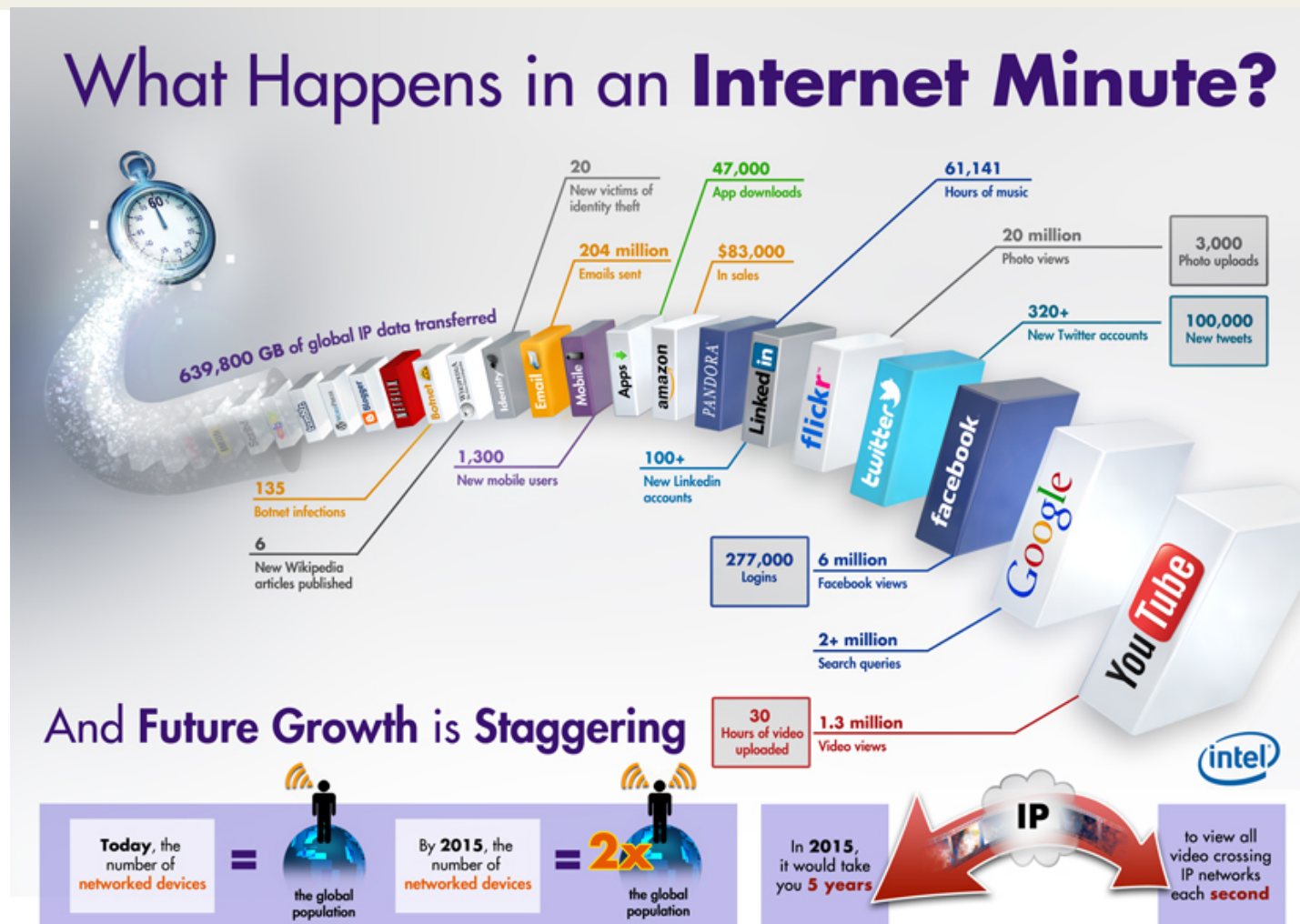
What Happens in an **Internet Minute?**
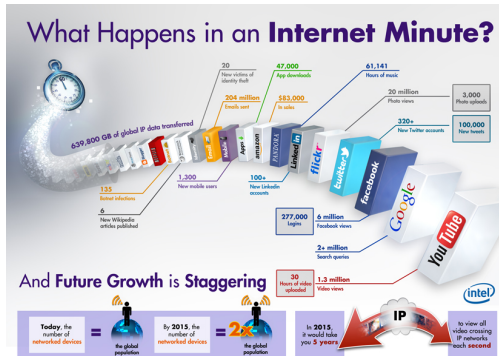
	10k, 1ms	1k, 100us
Hours	12600000	7200000
Minutes	210000	120000
Seconds	3500	2000
Millis	3.5	2

Minutes, Seconds, Millis of Internet time

	Global IP Data (GB)	BotNet Infections	New Wikipedia Articles	New Identity Theft Victims	Emails Sent (M)	New Mobile Users	Apple App Downloads	Amazon Sales USD	Pandora Music Listened (hours)	New LinkedIn Accounts	Flickr Photo Views (M)	New Twitter Accounts	Tweets	Facebook Logins	Google Searches	YouTube Uploads (Video Hours)	YouTube Views	NA Equities Market Data (B)	EU Equities Market Data (B)	Draw Something Drawings (9 Wks)	Currency Market Prices
Hour	3.84E+07	7.92E+03	3.60E+02	1.20E+03	1.22E+04	7.80E+04	2.82E+06	4.98E+06	3.67E+06	6.00E+03	1.20E+03	1.92E+04	6.00E+06	1.66E+07	1.20E+08	1.80E+03	7.80E+07	1.08E+10	7.20E+08	1.26E+07	7.20E+06
Minute	6.40E+05	1.32E+02	6.00E+00	2.00E+01	2.04E+02	1.30E+03	4.70E+04	8.30E+04	6.11E+04	1.00E+02	2.00E+01	3.20E+02	1.00E+05	2.77E+05	2.00E+06	3.00E+01	1.30E+06	1.80E+08	1.20E+07	2.10E+05	1.20E+05
Second	1.07E+04	2.20E+00	1.00E-01	3.33E-01	3.40E+00	2.17E+01	7.83E+02	1.38E+03	1.02E+03	1.67E+00	3.33E-01	5.33E+00	1.67E+03	4.62E+03	3.33E+04	5.00E-01	2.17E+04	3.00E+06	2.00E+05	3.50E+03	2.00E+03
Millis	1.07E+01	2.20E-03	1.00E-04	3.33E-04	3.40E-03	2.17E-02	7.83E-01	1.38E+00	1.02E+00	1.67E-03	3.33E-04	5.33E-03	1.67E+00	4.62E+00	3.33E+01	5.00E-04	2.17E+01	3.00E+03	2.00E+02	3.50E+00	2.00E+00

■ Hour ■ Minute ■ Second ■ Millis

Data On Demand

PUSH TECHNOLOGY

The Last Mile is HARD

Buffer Bloat

Network Saturation

Battery!

Download Bandwidth

- Expected (Mbps)
- Download (Mbps)

Latency

$y = -1.1667x + 45703$

$R^2 = 0.02496$
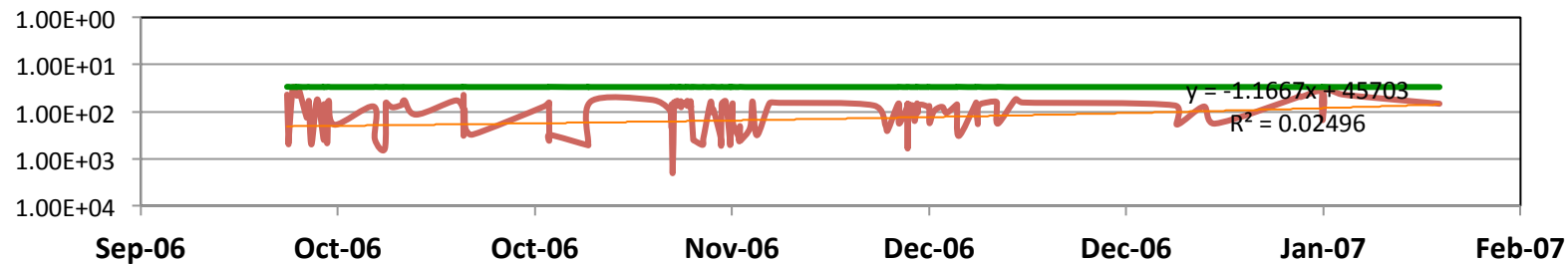
- Latency (ms)
- Expected (ms)
- Linear (Latency (ms))

Data On Demand

PUSH TECHNOLOGY

Conversations

M2M

M2H

Traditional Messaging
MQTT, AMQP

Bidirectional Real
Time Data Distribution

WebSockets

W3C Server Sent
Events

Data On Demand

PUSH
TECHNOLOGY

Traditional Messaging

A

B

ba

bb

Producers

?
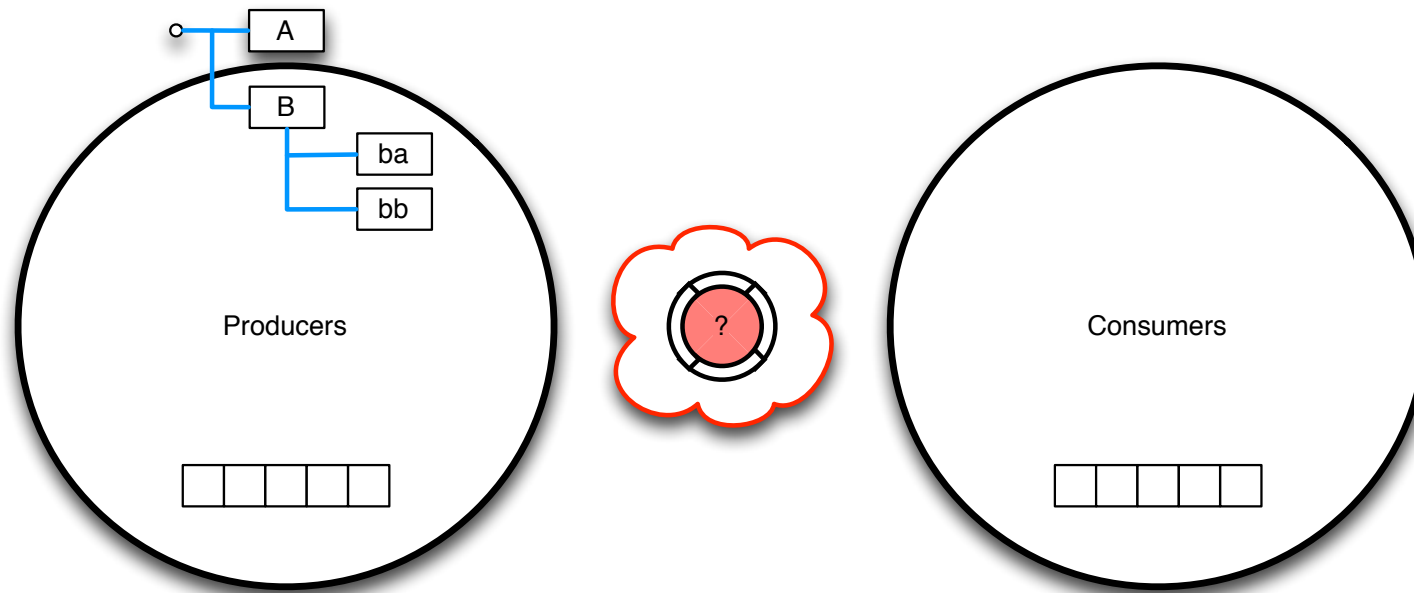
Consumers

Pros

- Loosely coupled.
- All you can eat messaging patterns
- Familiar

Cons

- No data model. Slinging **blobs**
- Fast producer, slow consumer? **Ouch.**
- No data 'smarts'. A blob is a blob.

Invented yonks ago...

Before the InterWebs

For 'reliable' networks

For machine to machine

Remember DEC Message Queues?

- That basically. Vomit!

When fallacies were simple

–The network is reliable

–Latency is zero

–Bandwidth is infinite

–There is one administrator

–Topology does not change

–The network is secure

–Transport cost is zero

–The network is homogeneous

Then in 1992, this happened:

The phrase 'surfing the internet' was coined by Jean Poly.

First SMS sent

First base.
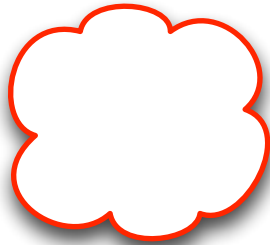
It grew, and it grew

Then in 2007, this happened:

The **god** phone:

Surfing died. Touching happened.

Second base unlocked.

Then in 2007, this happened:

So we took all the things and put them in the internet:

Cloud happened.

So we could touch all the things.

They grew and they grew like all the good things do.

Messaging
Apps
Hardware
Virtualize all the things
Services
Skills,
Specialties

Then in 2007, this happened:

And it grew and it grew.

Like all the good things do.

Messaging
Apps
Hardware
Virtualize all the things
Services
Skills,
Specialties

So we scaled all the things

Big data is fundamentally about extracting value, understanding
implications, gaining insight so we can
learn and improve continuously.

It's nothing new.

big

DATA

But, problem: The bird, basically.

Immediately Inconsistent.

But, Eventually Consistent

… Maybe.

Humans

Are

grumpy

IIBECM

Stop.

−The network is **not** reliable

   nor is it cost free.

−Latency is **not** zero

   nor is it a democracy.

−Bandwidth is **not** infinite

   nor predictable especially the last mile!

−There is **not only** one administrator

   trust, relationships **are** key

−Topology **does** change

   It should, however, converge eventually

−The network is **not** secure

   nor is the data that flows through it

−Transport cost is **not** zero

   but what you **don't do** is **free**

−The network is **not** homogeneous

   nor is it **smart**

Look.

- –**What** and **How** are what geeks do.

- –**Why** gets you paid

- –**Business Value** and **Trust** dictate What and How

- – Policies, Events and Content *implements* ***Business Value***

- –***Science*** *basically. But think like a carpenter:*

- –*Measure twice. Cut once.*

Listen.

- **Every** nuance comes with a set of tradeoffs.

- Choosing the right ones can be hard, but it pays off.

- Context, Environment are critical

- Break all the rules, one benchmark at a time.

- **Benchmark Driven Development**

Action: Data Distribution

Messaging remixed around:

Relevance - Queue depth for conflatable data should be 0 or 1. No more

Responsiveness - Use HTTP/REST for things. Stream the little things

Timeliness - It's relative. M2M != M2H.

Context - Packed binary, deltas mostly, snapshot on subscribe.

Environment - Don't send 1M 1K events to a mobile phone with 0.5mbps.

Data On Demand

PUSH ™
TECHNOLOGY

Action. Virtualize Client Queues

A

B

ba

bb

Producers

Buffer Bloat

Consumers
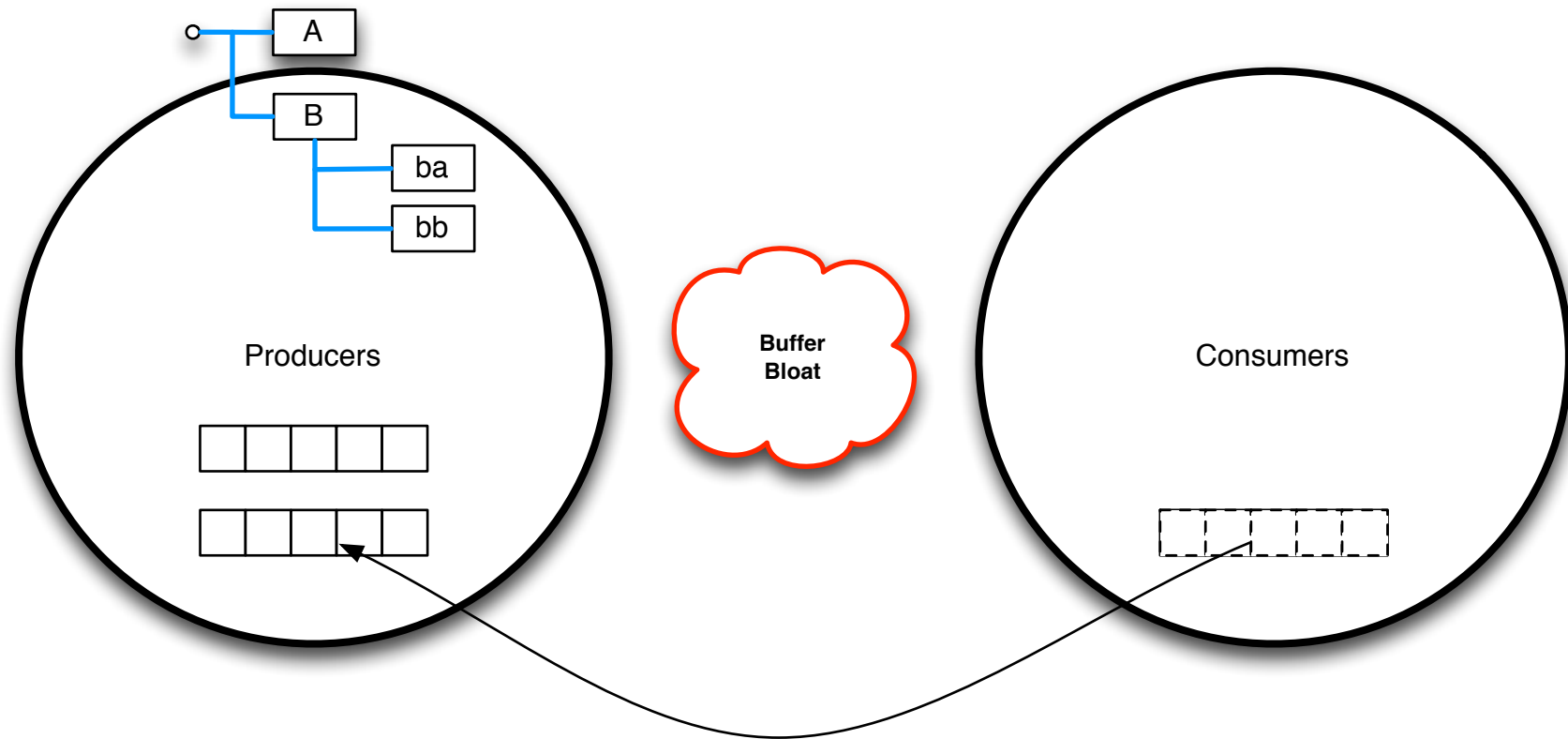
Nuance: Client telemetry. Tradeoff: Durable subscriptions harder

Action. Add data caching

Producers

Is it a cache?

Consumers
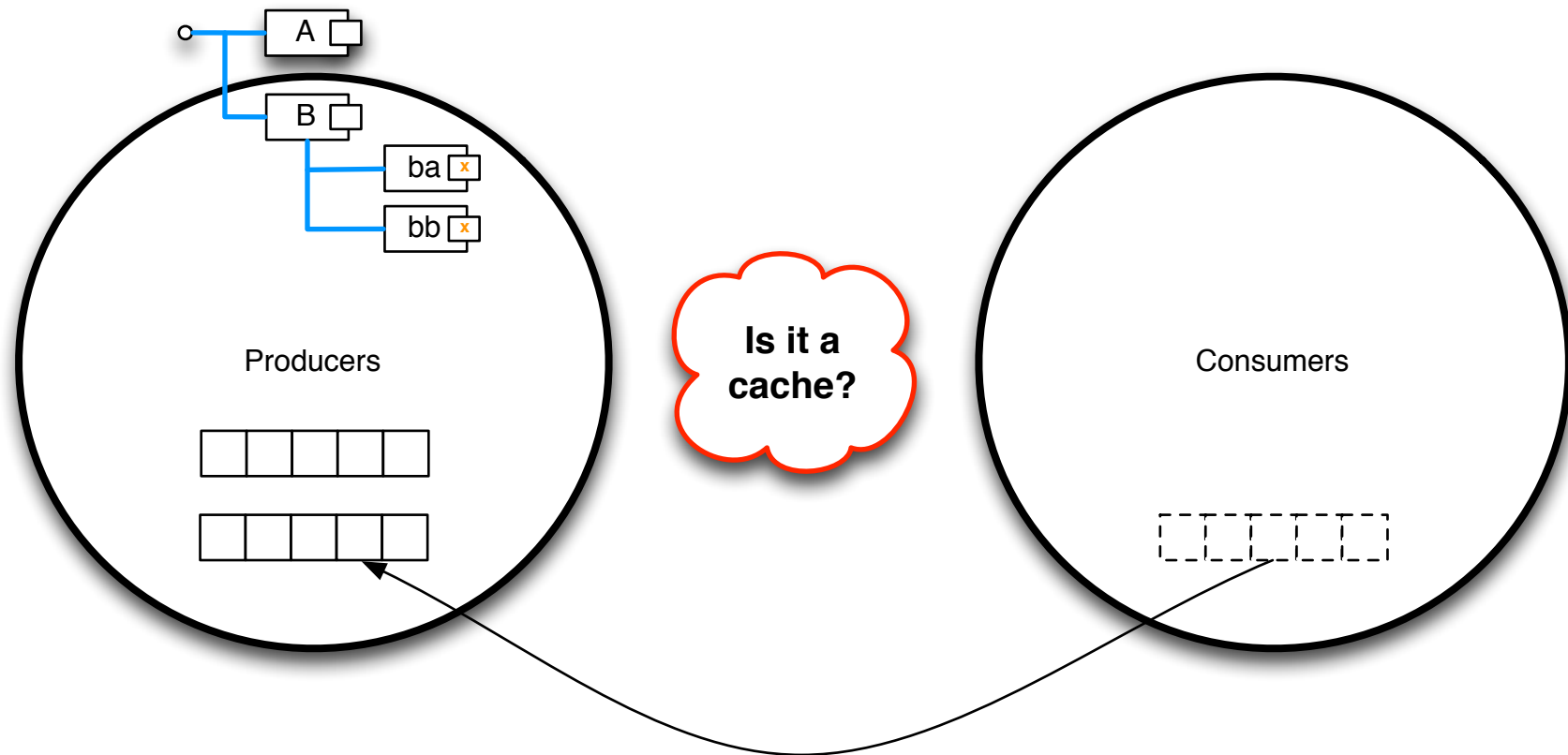
One hop closer to the edge …
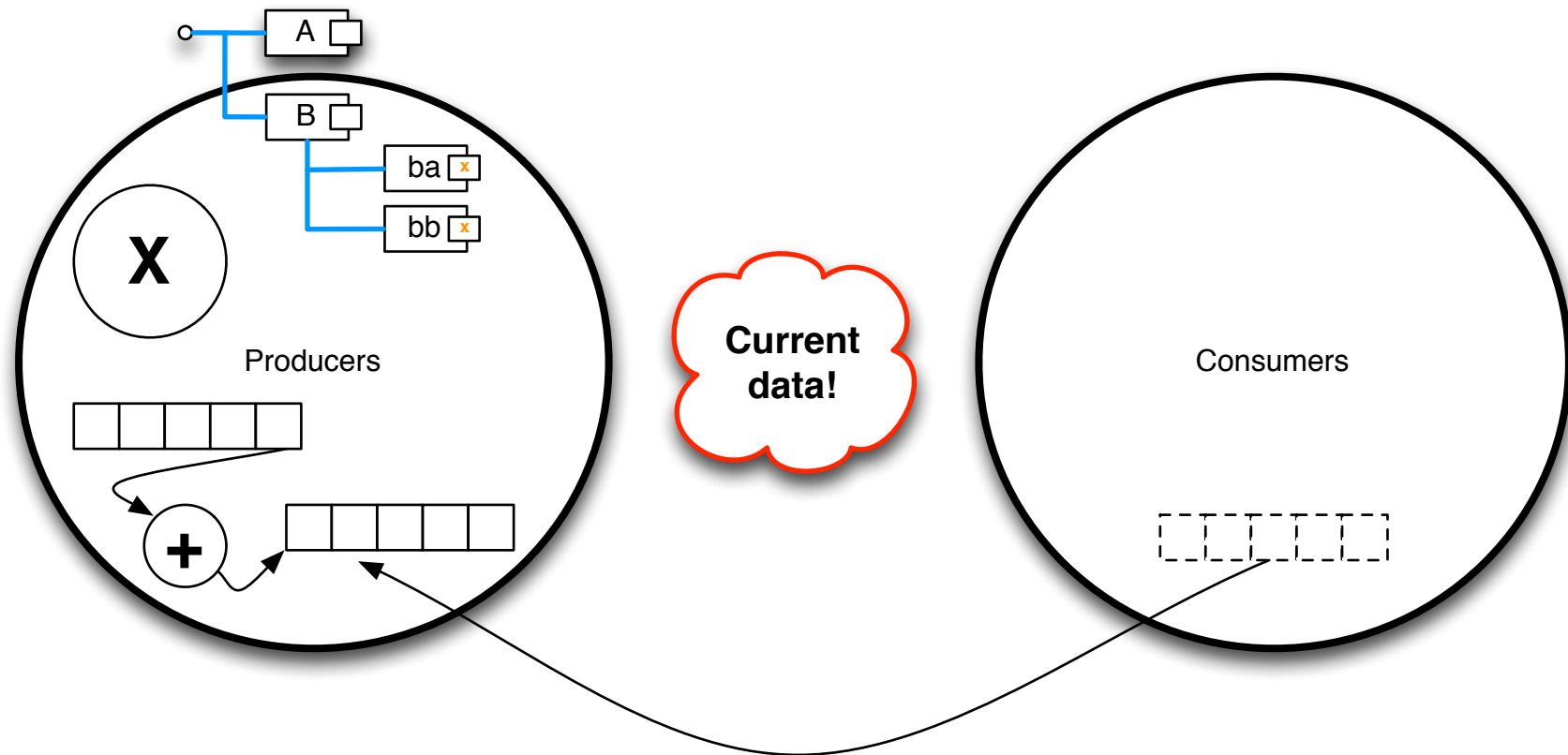
Action. Exploit data structure
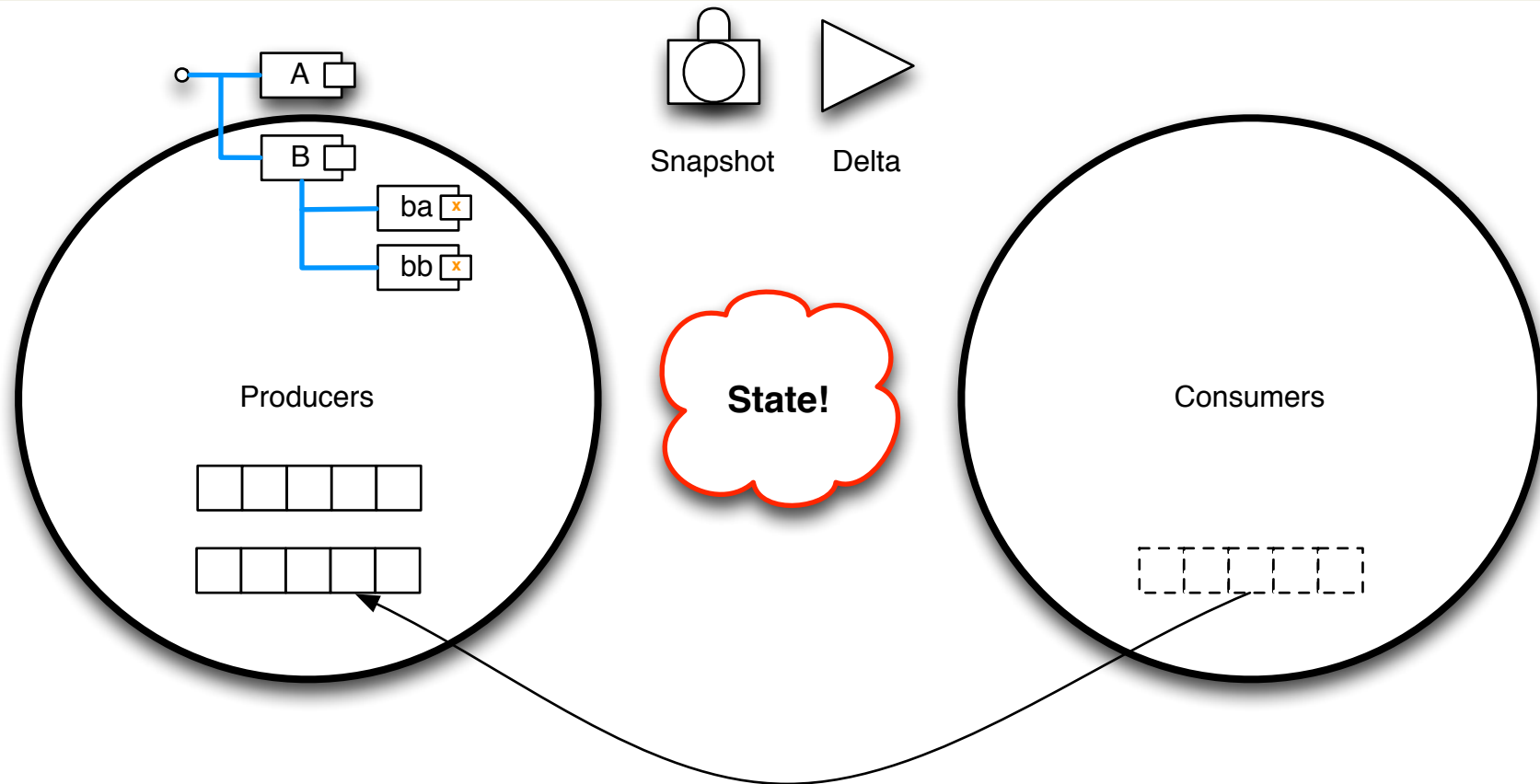
Snapshot recovery. Deltas or Changes mostly. Conserves bandwidth

Action. Behaviors
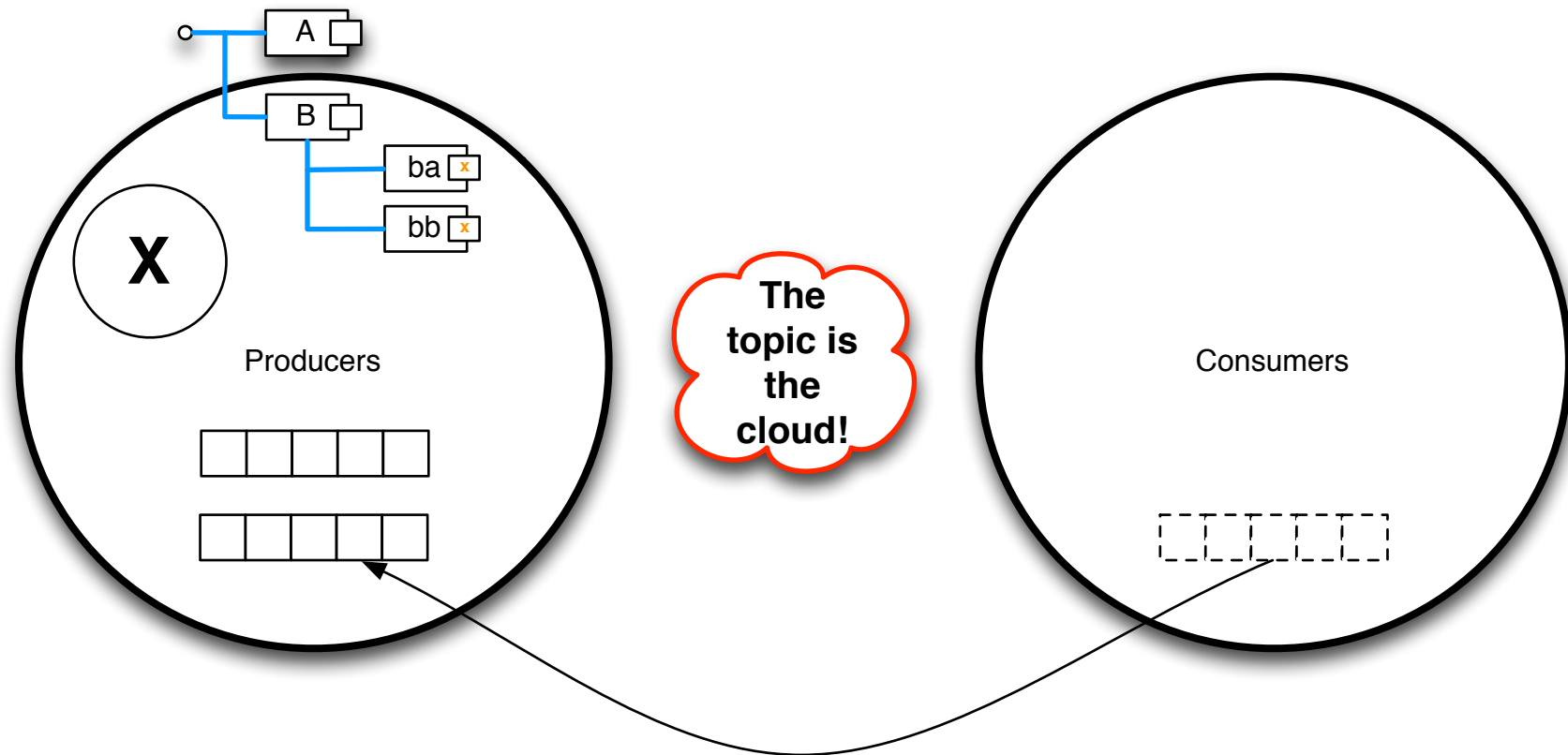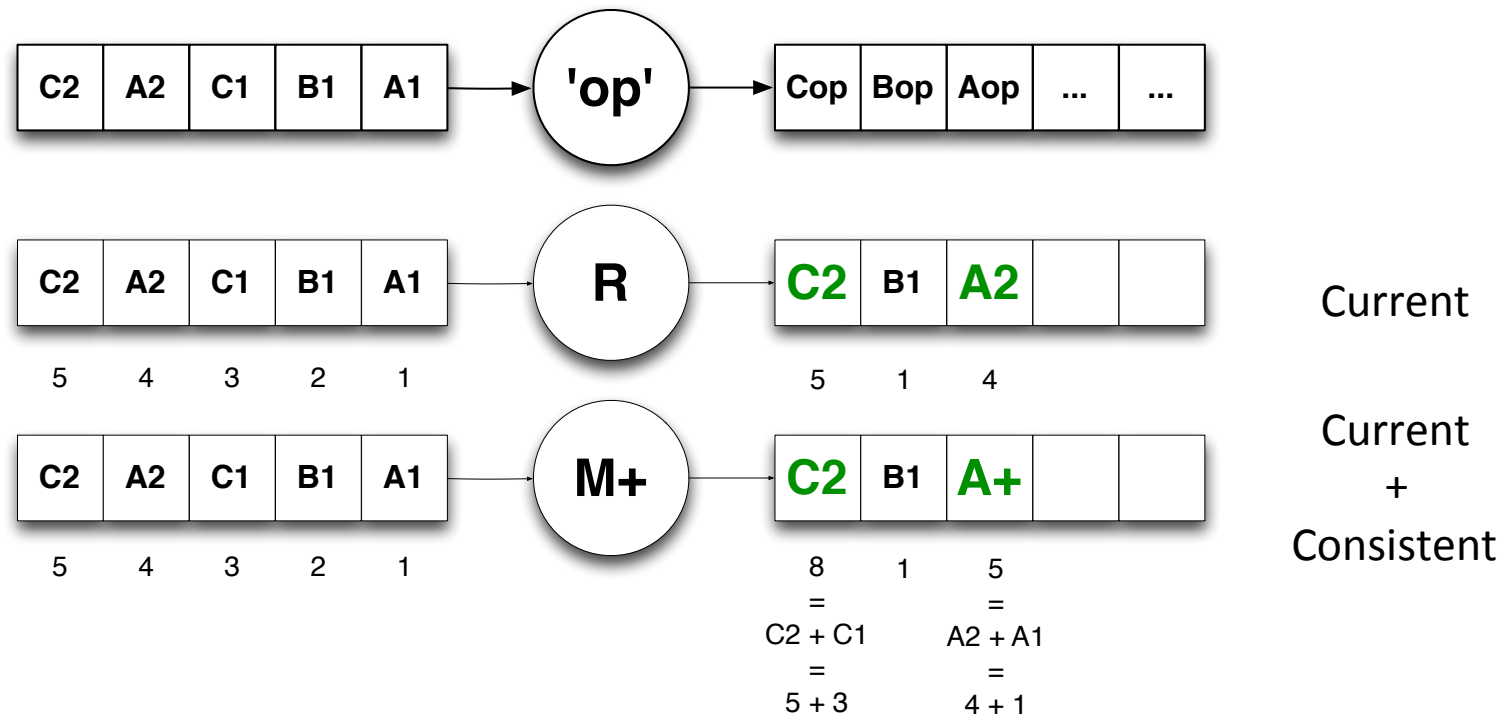

Extensible. Nuance? Roll your own protocols. Tradeoff? 3rd party code in the engine :/

Action. Structural Conflation

Ensures only current + consistent data is distributed. Actively soaks up bloat. Extensible!

Action. Structural Conflation

Replace

- Replace/Overwrite 'A1' with 'A2'
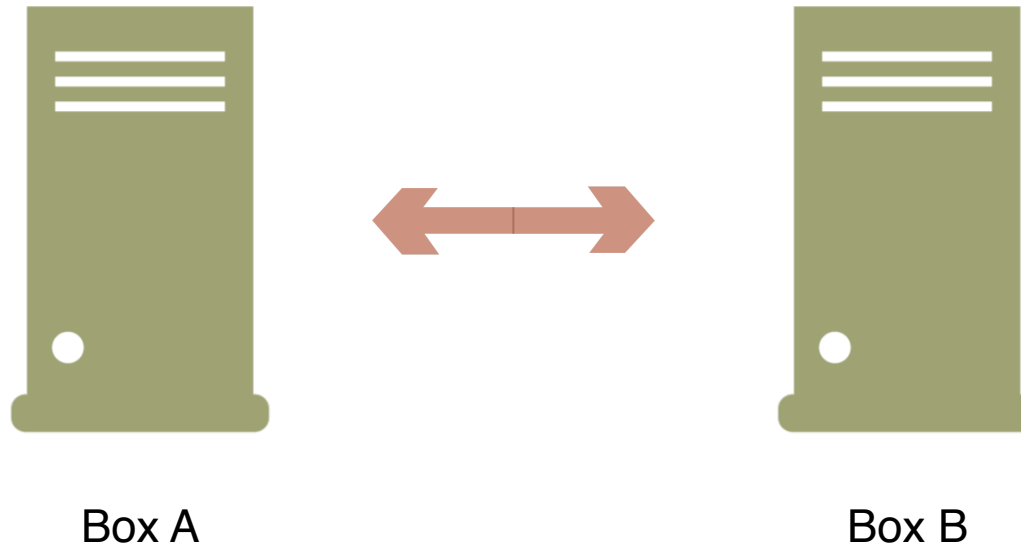- 'Current data right now'
- Fast

Merge

- Merge A1, A2 under some operation.
  - Operation is pluggable
  - Tunable consistency
  - Performance f(operation)

Benchmark Driven Development

'Measuring Embedded CEP'

Performance is **not** a number

Benchmark Models

Box A   Box B

Throughput (Worst case)   **Latency (Best case)**

- Ramp clients continuously   - Really simple. 1 client
- 100 messages per second per client   - Ping – Pong. Measure RTT time.
- Payload: 125 .. 2000 bytes   - Payload: 125 .. 2000 bytes
- Message style vs with conflation
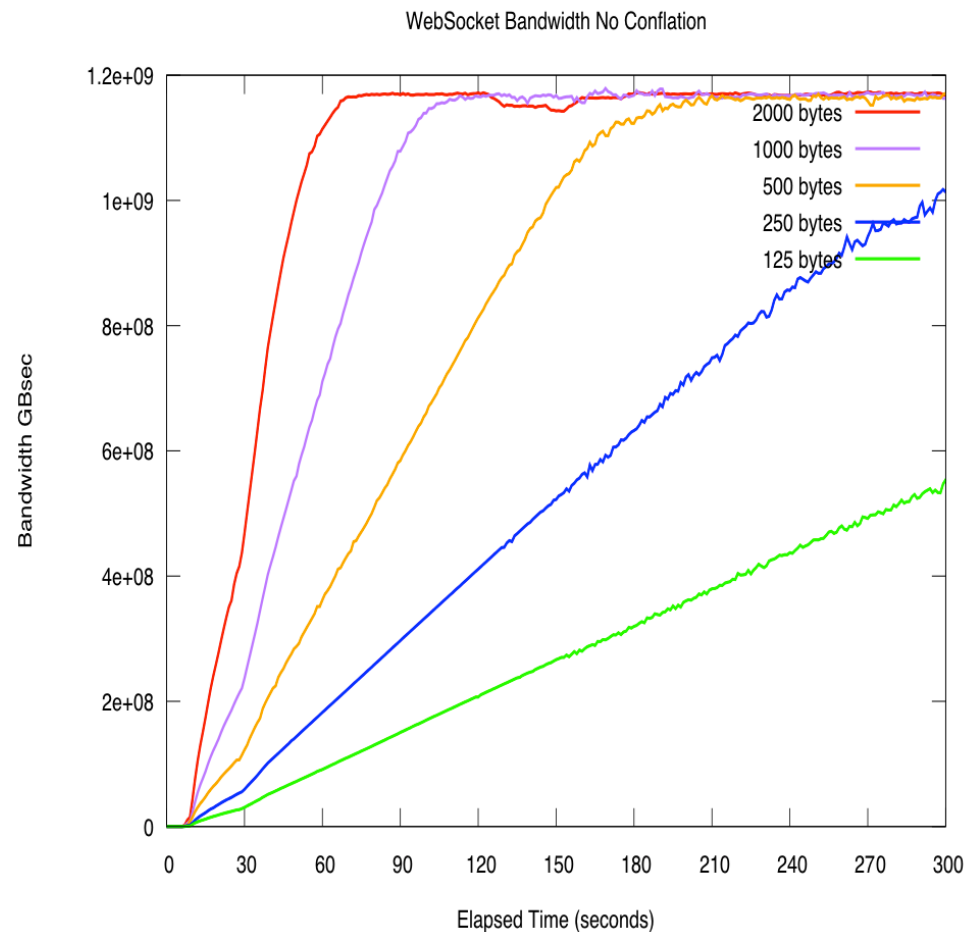
Throughput. Non-conflated

WebSocket Bandwidth No Conflation

- 2000 bytes
- 1000 bytes
- 500 bytes
- 250 bytes
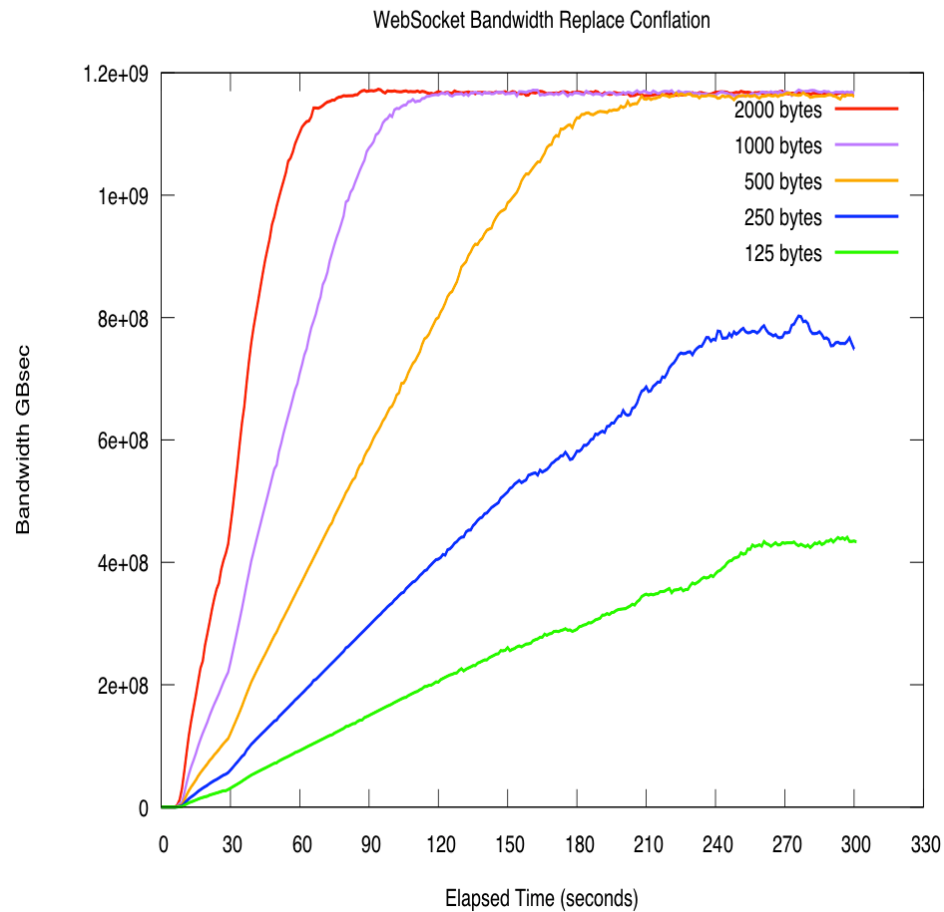- 125 bytes

Bandwidth GBsec

Elapsed Time (seconds)

- Cold start server

- Ramp 750 clients 'simultaneously' at 5 second intervals

- 5 minute benchmark duration

- Clients onboarded linearly until IO (here) or compute saturation occurs.

- What the 'server' sees

Data On Demand

PUSH
TECHNOLOGY

Throughput. Non-conflated

WebSocket Messages Per Second Per Client No Conflation

2000 bytes
1000 bytes
500 bytes
250 bytes
125 bytes

MPS

Elapsed Time (seconds)
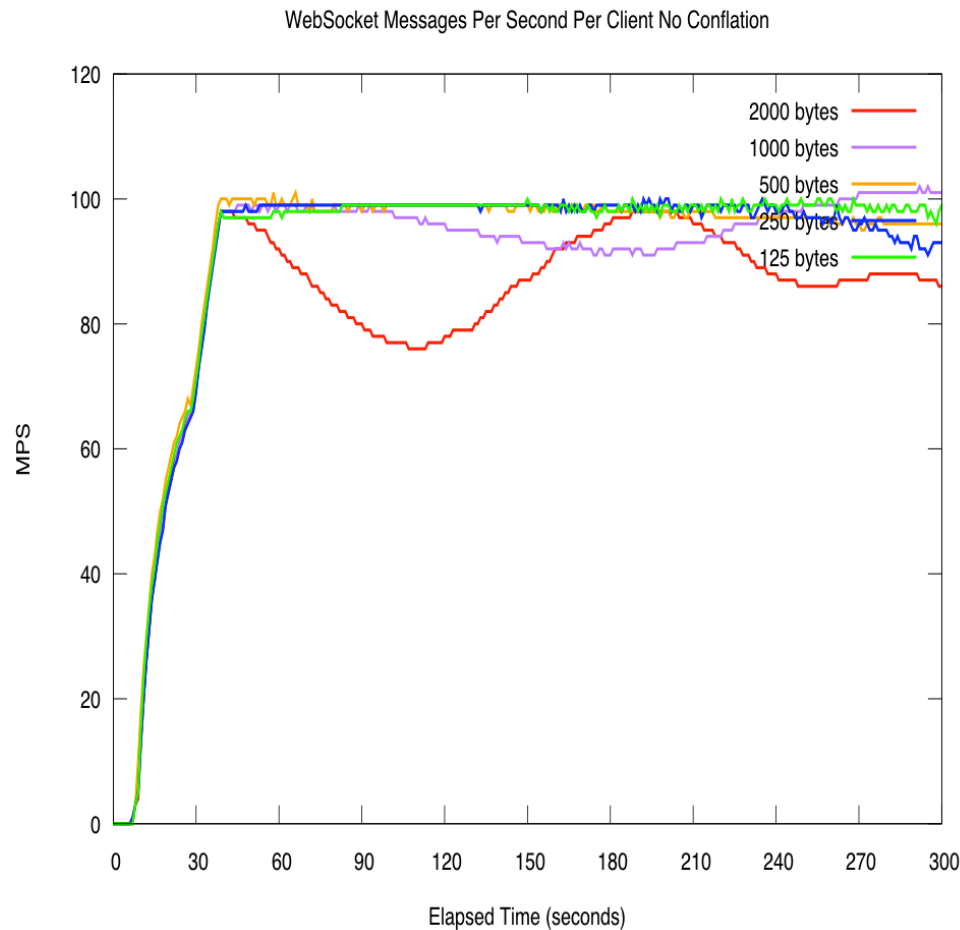
- What the 'client' sees

- At and beyond saturation of some resource?

- Things break!

- New connections fail. Good.
- Long established connections ok.
- Recent connections timeout and client connections are dropped. Good.

- Diffusion handles smaller message sizes more gracefully

- Back-pressure 'waveform' can be tuned out. Or, you could use structural conflation!

Data On Demand

PUSH
TECHNOLOGY

Throughput. Replace-conflated

WebSocket Bandwidth Replace Conflation

- Cold start server

- Ramp 750 clients 'simultaneously' at 5 second intervals

- 5 minute benchmark duration

- Clients onboarded linearly until IO (here) or compute saturation occurs.

- Takes longer to saturate than non-conflated case

- Handles more concurrent connections

- Again, 'server' view

Data On Demand

PUSH
TECHNOLOGY

Throughput. Replace-Conflated

WebSocket Messages Per Second Per Client Replace Conflation
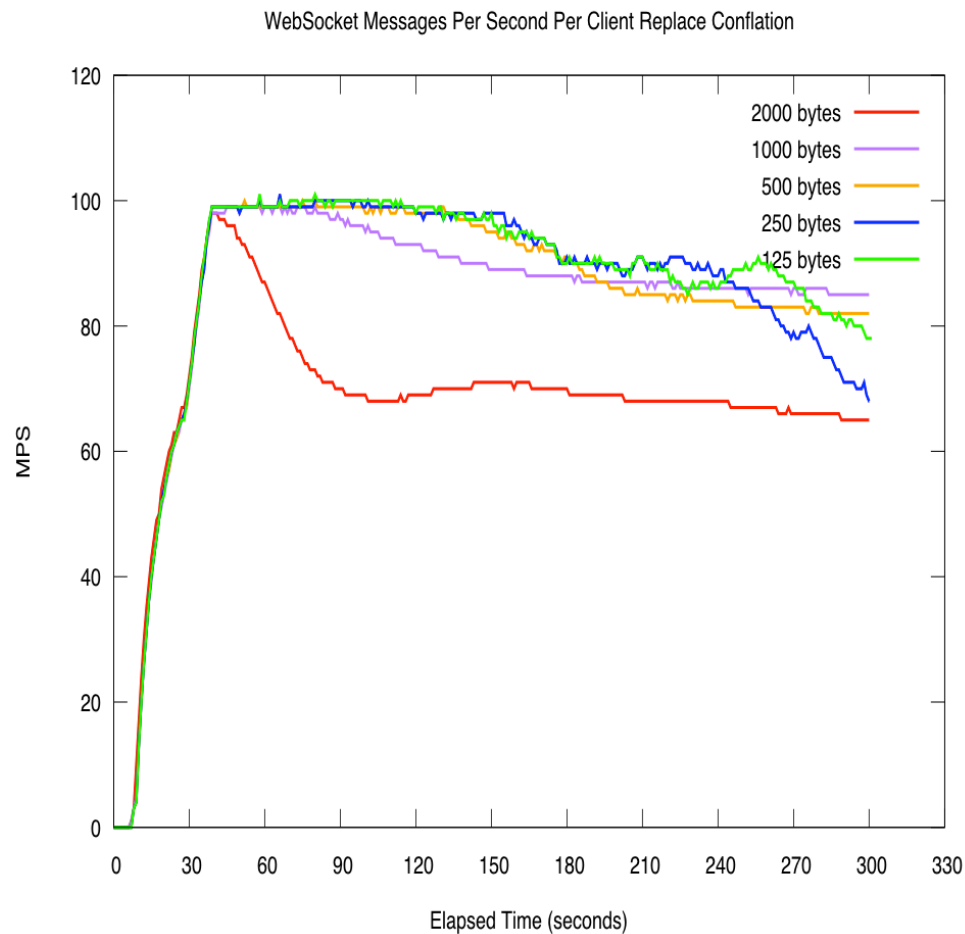
- What the 'client' sees

- Once saturation occurs Diffusion adapts actively by degrading messages per second per client

- This is good. Soak up the peaks through fairer distribution of data.

- Handles spikes in number of connections, volume of data or saturation of other resources using a single load adaptive mechanism
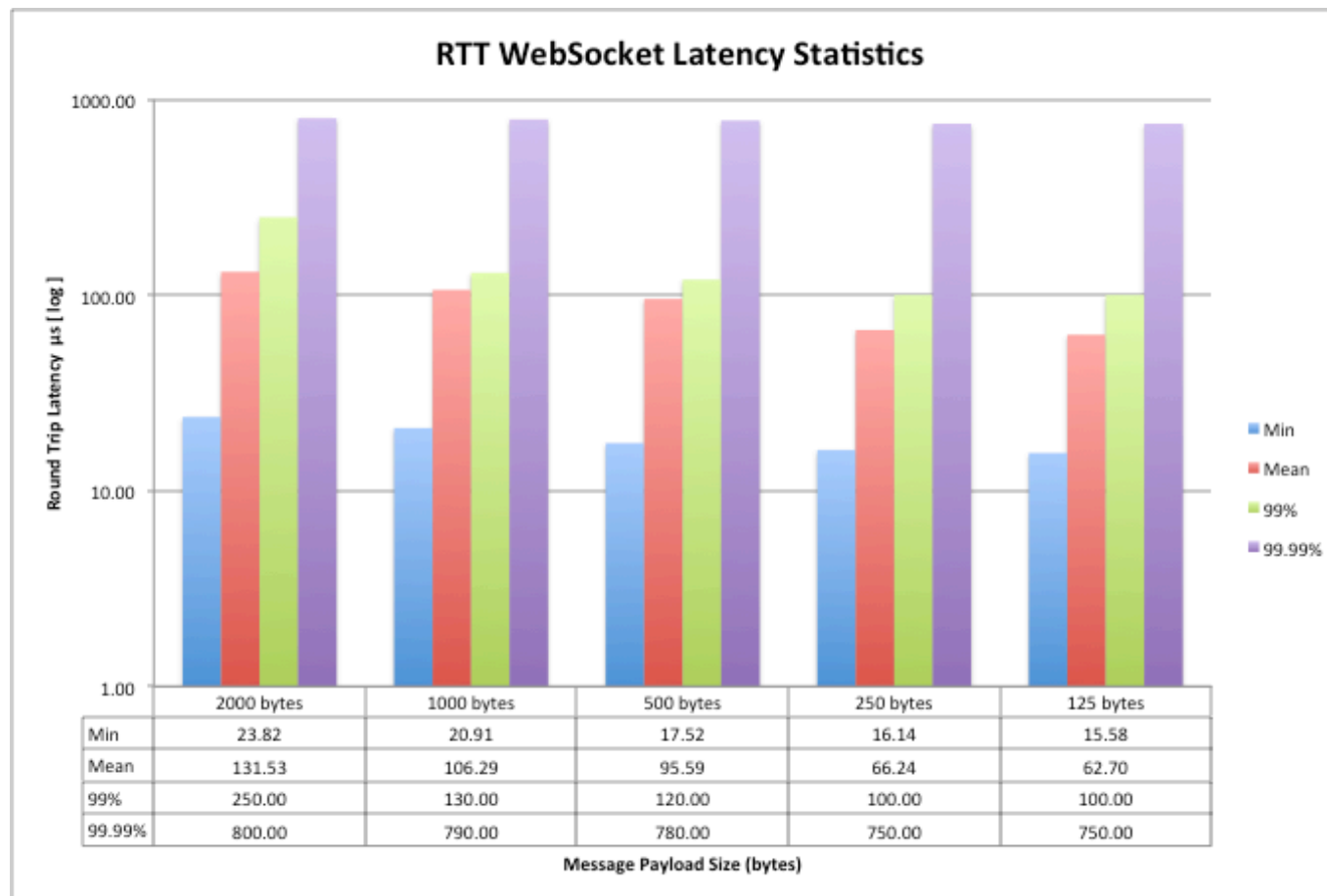
Throughput. Stats

- Data / (Data + Overhead)?
  - 1.09 GB/Sec payload at saturation.
  - 10Ge offers theoretic of 1.25GB/Sec.
  - Ballpark 87.2% of traffic represents business data.

- Benefits?

  - Optimize for business value of distributed data.
  - Most real-world high-frequency real-time data is recoverable
    - Stock prices, Gaming odds
  - Don't use conflation for transactional data, natch!

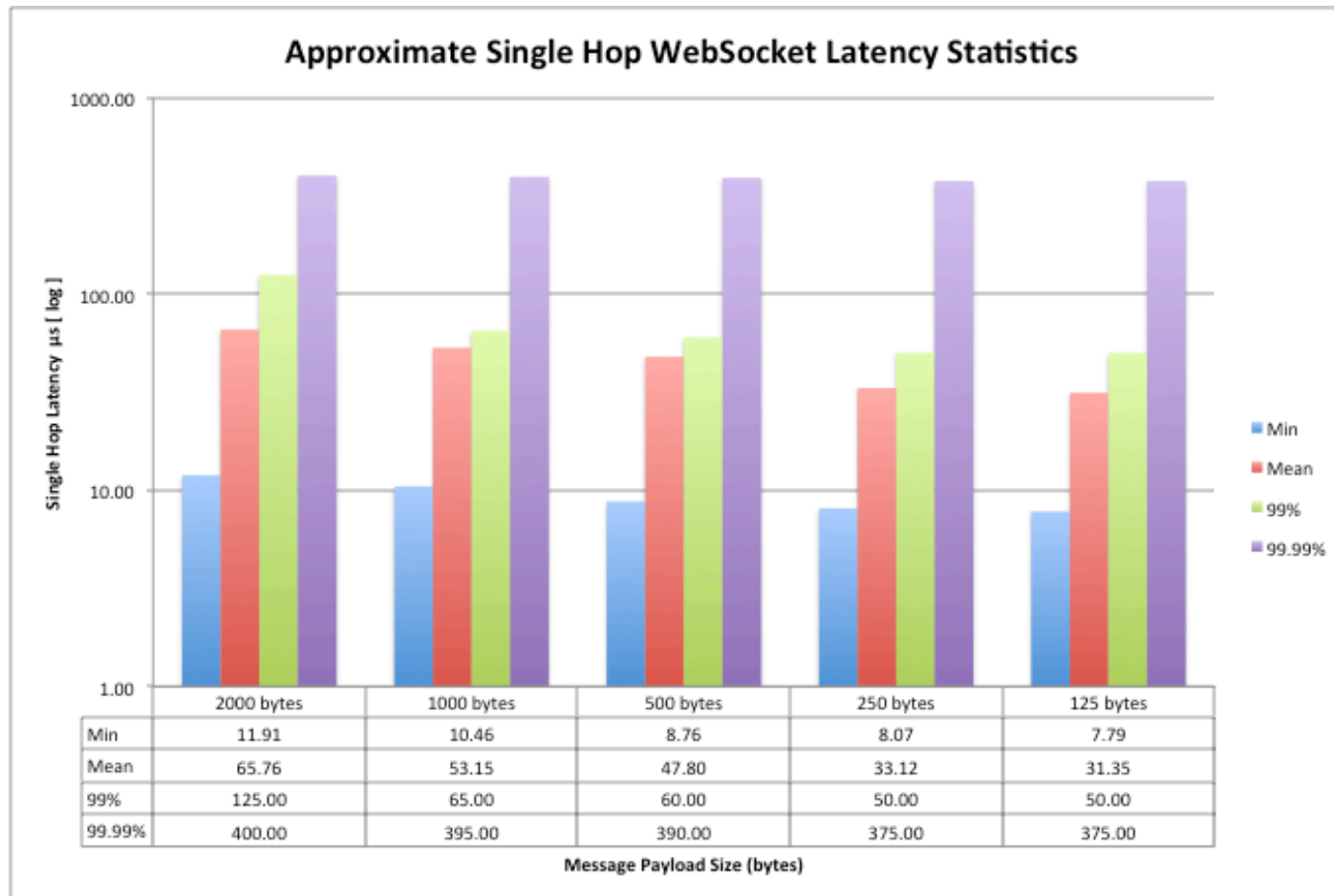# Latency (Best case)

Latency Summary Histogram
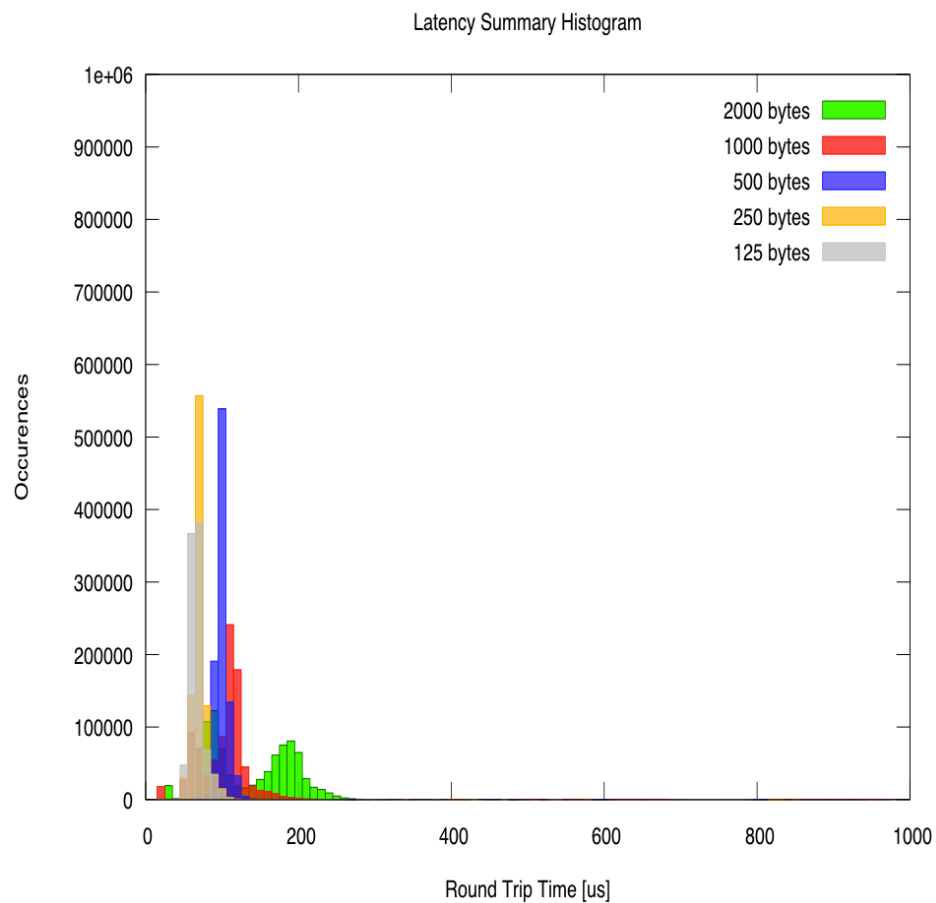
- Cold start server

- 1 solitary client

- 5 minute benchmark duration

- Measure ping-pong round trip time

- Results vary by network card make, model, OS and Diffusion tuning.

Data On Demand

PUSH
TECHNOLOGY

Latency. Round Trip Stats

RTT WebSocket Latency Statistics

	2000 bytes	1000 bytes	500 bytes	250 bytes	125 bytes
Min	23.82	20.91	17.52	16.14	15.58
Mean	131.53	106.29	95.59	66.24	62.70
99%	250.00	130.00	120.00	100.00	100.00
99.99%	800.00	790.00	780.00	750.00	750.00

- Min
- Mean
- 99%
- 99.99%

Message Payload Size (bytes)

- Sub-millisecond at the 99.99% percentile

- As low as 15 microseconds

- On average significantly sub 100 microseconds for small data

Latency. Single Hop Stats

Approximate Single Hop WebSocket Latency Statistics

	2000 bytes	1000 bytes	500 bytes	250 bytes	125 bytes
Min	11.91	10.46	8.76	8.07	7.79
Mean	65.76	53.15	47.80	33.12	31.35
99%	125.00	65.00	60.00	50.00	50.00
99.99%	400.00	395.00	390.00	375.00	375.00

- Min
- Mean
- 99%
- 99.99%

- Sub 500 microseconds at the 99.99% percentile
- As low as 8 microseconds
- On average significantly sub 50 microseconds for small data

Latency in perspective

~
125
bytes
(avg)

~
1000
bytes
(avg)

2.4 us   **5.0 us**   **8.0 us**   **50.0 us**
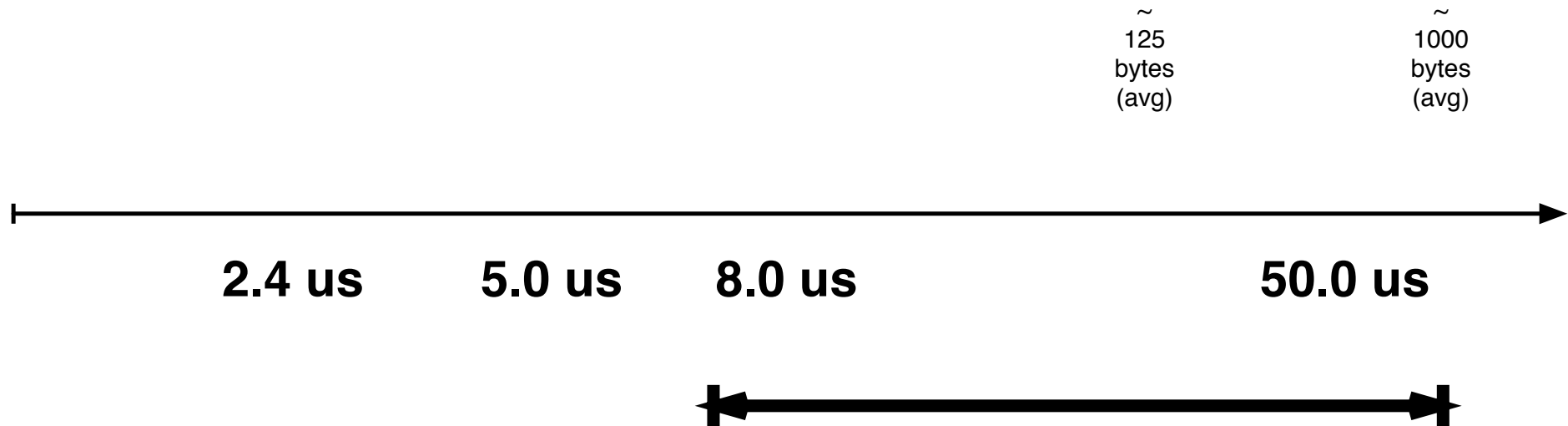
- 2.4 us. A tuned benchmark in C with low latency 10Ge NIC, with kernel bypass, with FPGA acceleration
- 5.0 us. A basic java benchmark – as good as it gets in java
- Diffusion is measurably 'moving to the left' release on release
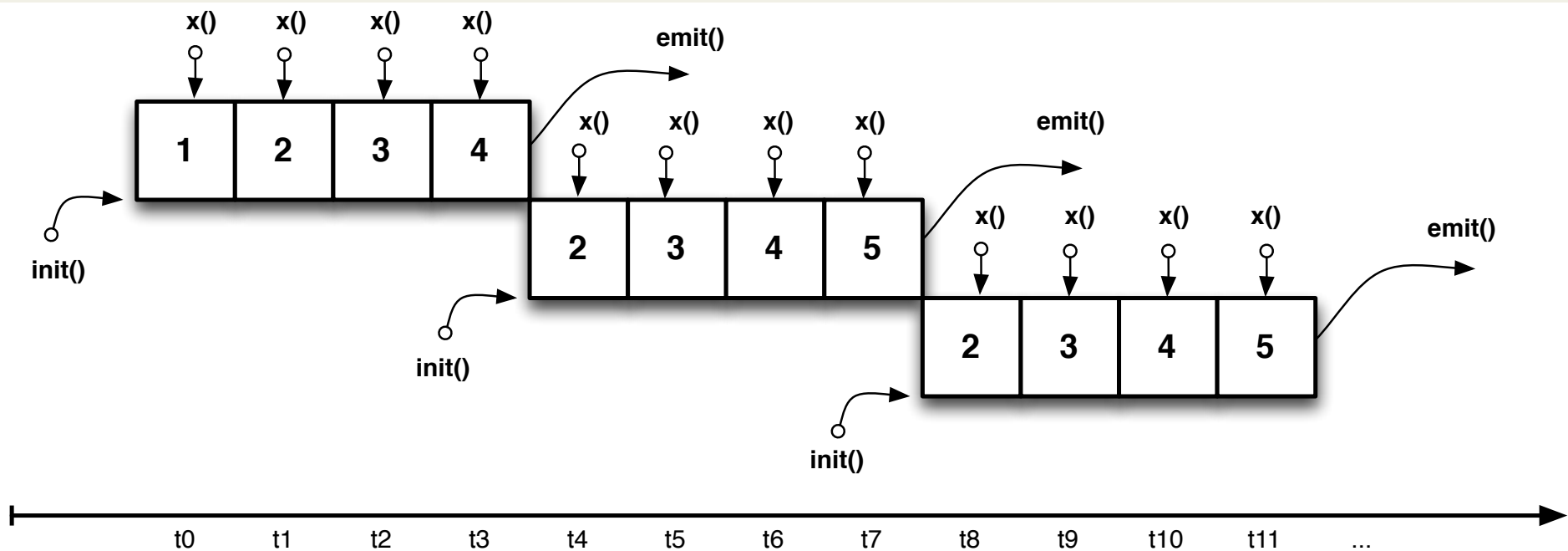- We've been actively tracking and continuously improving since 4.0

Embedded
Event
Processing
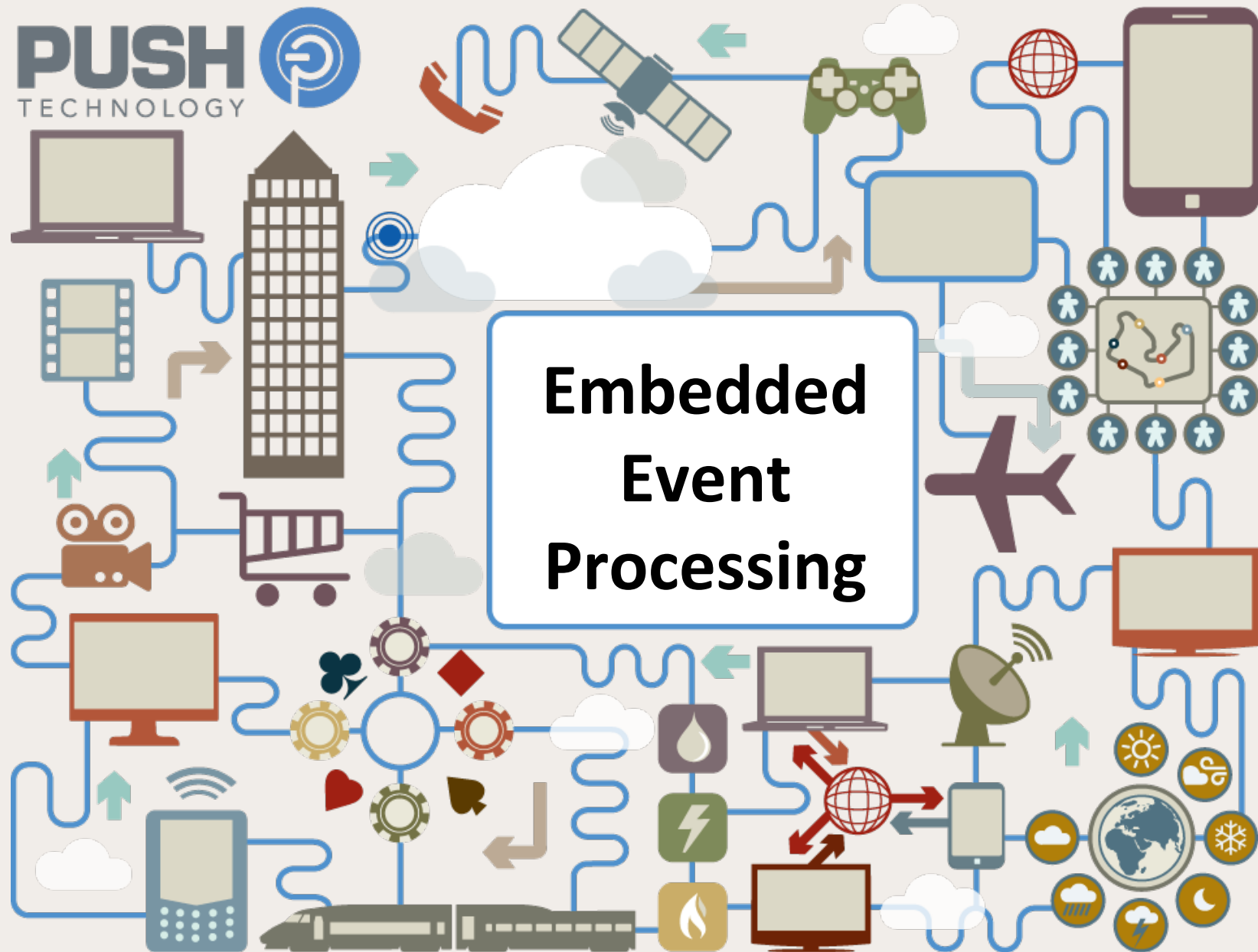
CEP, basically
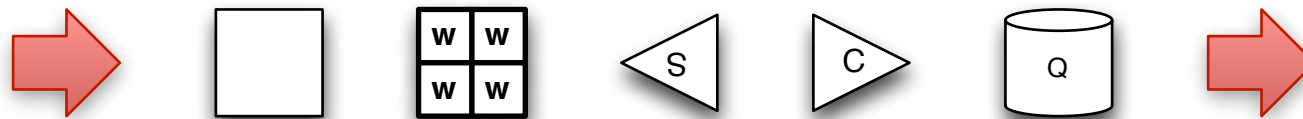
What is eep.js?

- Add aggregate functions and window operations to Node.js
- 4 window types: tumbling, sliding, periodic, monotonic
- Node makes evented IO easy. So just add windows.
- Fast. 8-40 million events per second (upper bound).

Tumbling Windows

What is a tumbling window?

- Every N events, give me an average of the last N events
- Does not overlap windows
- 'Closing' a window, 'Emits' a result (the average)
- Closing a window, Opens a new window

Aggregate Functions

```javascript
CountFunction.prototype = new AggregateFunction();
function CountFunction() {
  var self = this; var count = 0;

  self.init = function() { count = 0; };
  self.accumulate = function(value) { count++; };
  self.emit = function()  { return count; };
  self.make = function() { return new CountFunction(); };
};
```

What is an aggregate function?

- A function that computes values over events.

- The cost of calculations are ammortized per event

- Just follow the above recipe

- Example: Aggregate 2M events (equity prices), send to GPU on emit, receive 2M options put/call prices as a result.

meh: Fumbling Windows

```
start = new Date().getTime();
sum = 0;
for (var i = 0; i < t; i++) {
  var stride = 64*i;
  count = 0;
  for (var j = 0; j < size; j++) {
    sum += cache[stride+j];
    count++;
  }
  avg = sum / size;
}
end = new Date().getTime();
```
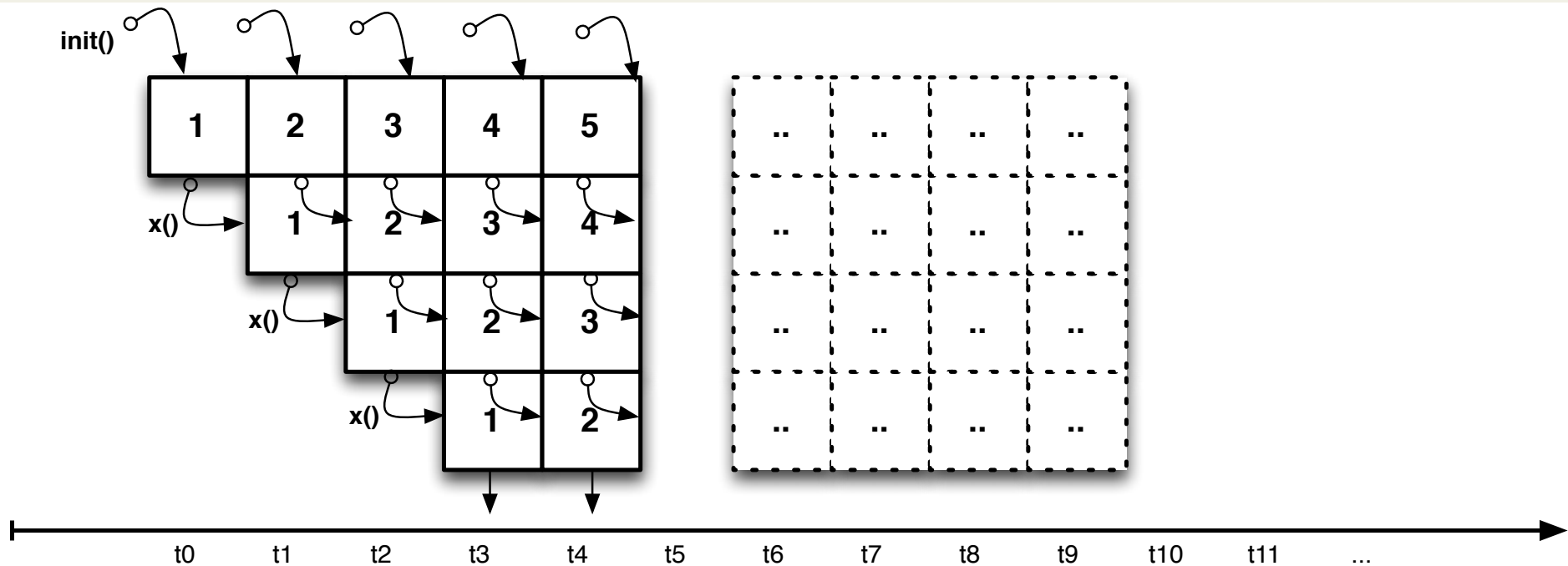
Lesser Fumbling Windows

```
start = new Date().getTime();
var fn = new NullFunction();
for (var i = 0; i < t; i++) {
    var stride = 64*i;
    fn.init();
    for (var j = 0; j < size; j++) {
        fn.accumulate(cache[stride+j]);
    }
    fn.emit();
}
end = new Date().getTime();
```

Event Windows, tumbling.

```javascript
'tumbling window usage example': function(assert) {
  var tumbling  = EventWorld.make().windows().tumbling(new AvgFunction(),
2);
  assert.ok(tumbling != null);

  var results = new Array();
  tumbling.on('emit', function(v) {
    results.push(v);
  });
  tumbling.enqueue(1);
  tumbling.enqueue(2);
  tumbling.enqueue(3);
  tumbling.enqueue(4);
  tumbling.enqueue(5);
  tumbling.enqueue(6);
  assert.same([1.5,3.5,5.5], results);
```

Sliding Windows

What is a sliding window?

- Like tumbling, except can overlap. But $O(N^2)$, Keep N small.
- Every event opens a new window.
- After N events, every subsequent event emits a result.
- Like all windows, cost of calculation amortized over events

Event Windows, sliding.

```
'sliding window usage example': function(assert) {
  var sliding  = EventWorld.make().windows().sliding(new AvgFunction(), 2);
  assert.ok(sliding != null);

  var results = [];
  sliding.on('emit', function(v) {
    results.push(v);
  });
  sliding.enqueue(1);
  sliding.enqueue(2);
  sliding.enqueue(3);
  sliding.enqueue(4);
  assert.same([1.5,2.5,3.5], results);
  assert.done();
},
```

Periodic Windows
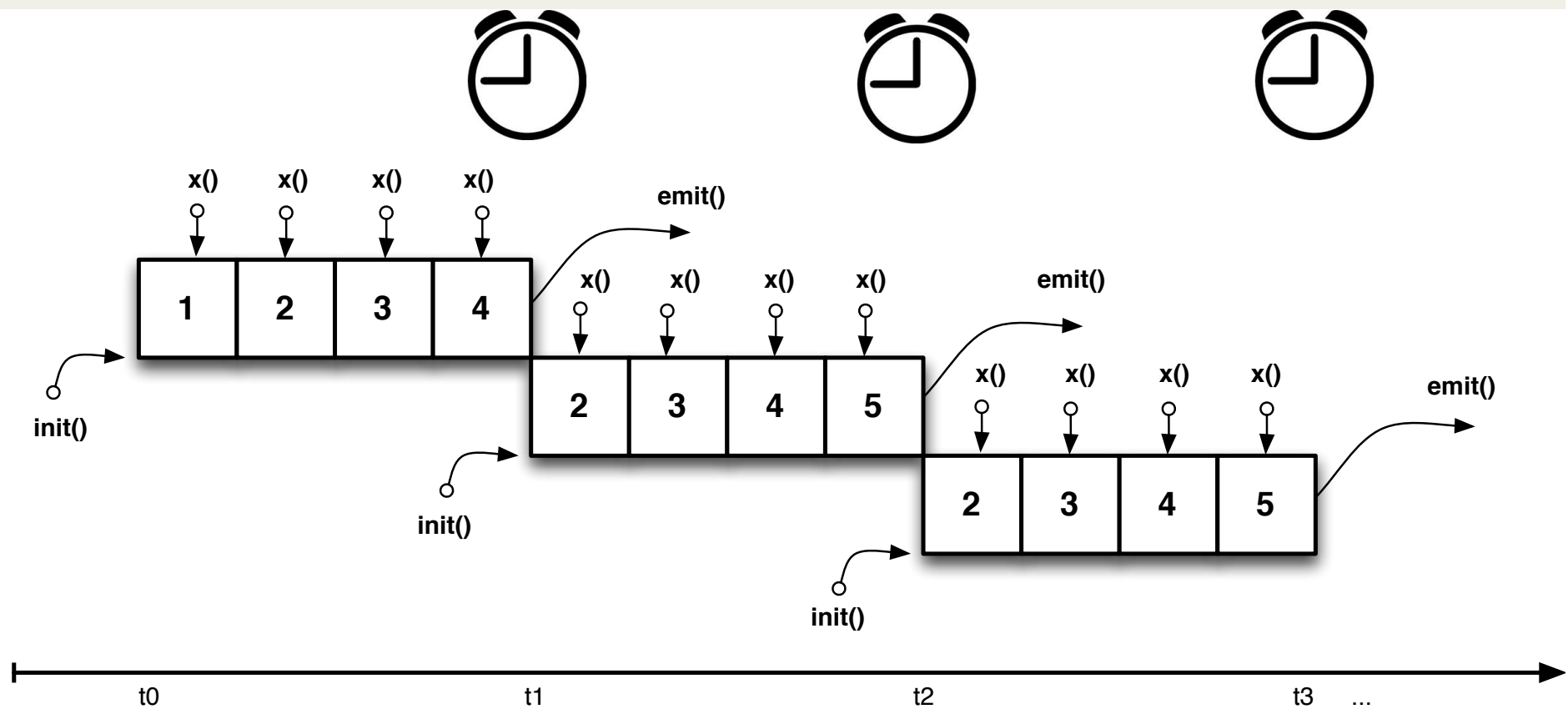
What is a periodic window?

- Driven by 'wall clock time' in milliseconds
- Not monotonic, natch. Beware of NTP

Data On Demand

PUSH ℗™
TECHNOLOGY

Event Windows, periodic.

```javascript
dc.on('open', function() {
  console.log('Connected to: ' + url);

  dc.subscribe(topic);

  ew.on('emit', function(value) {
    console.log('Meps: ' + value);
  });

  setInterval(function() {
    ew.tick();
    dc.send(topic,'Node Dublin ' + ++count);
  }, 1);
});
```

Monotonic Windows
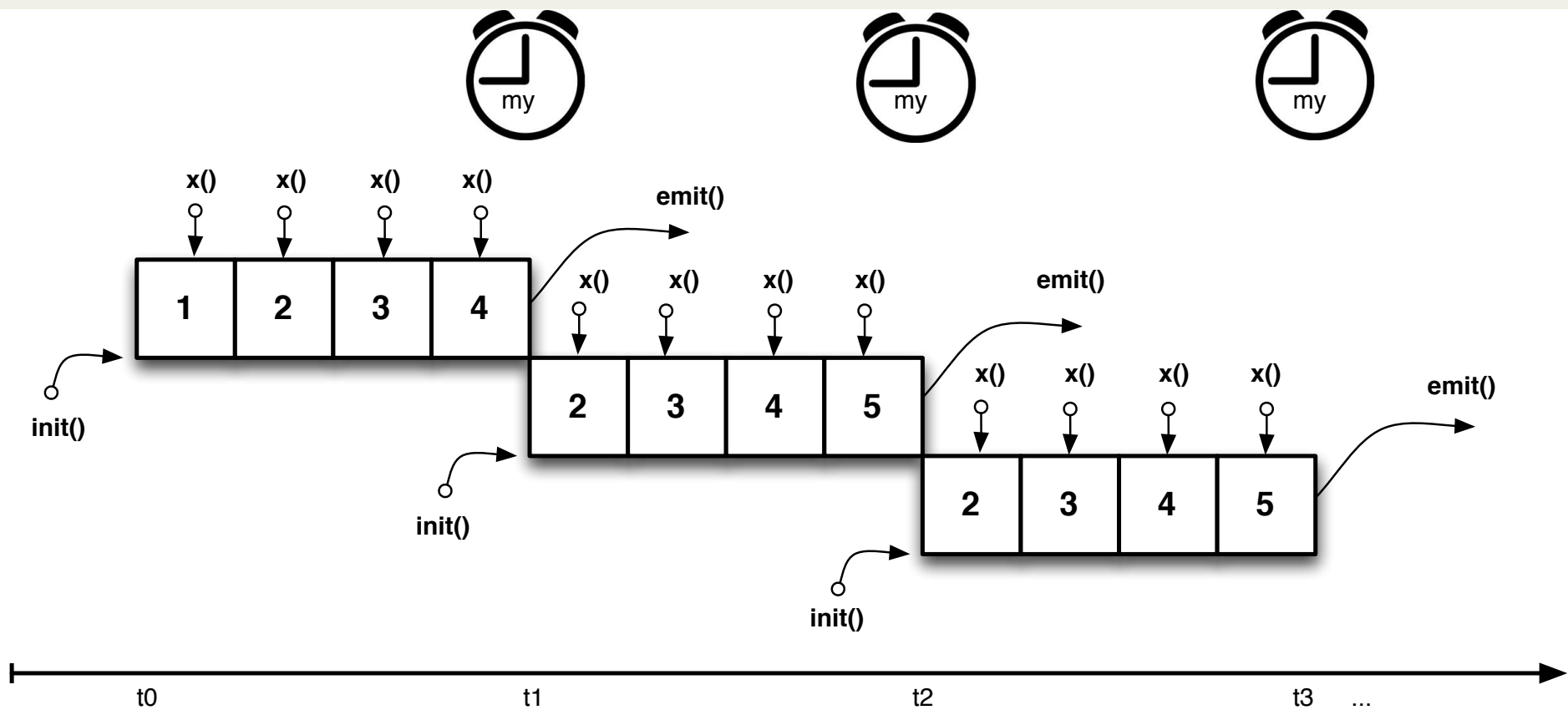
What is a monotonic window?

- Driven mad by 'wall clock time'? Need a logical clock?
- No worries. Provide your own clock! Eg. Vector clock

Data On Demand

PUSH
TECHNOLOGY

Event Windows, monotonic.

```javascript
dc.on('open', function() {
  console.log('Connected to: ' + url);

  dc.subscribe(topic);

  ew.on('emit', function(value) {
    console.log('Meps: ' + value);
  });

  setInterval(function() {
    ew.tick();
    dc.send(topic,'Node Dublin ' + ++count);
  }, 1);
});
```

Event Windows, monotonic.

```javascript
WallClock.prototype = new ClockSource();
function WallClock(interval) {
  var self = this; var at; var mark = null;
  self.tick = function() {
    at = new Date().getTime();
    if (mark == null) {
      mark = at + interval;
    }

    return !((at - mark) < interval);
  };
  self.tock = function(elapsed) {
    var d = at - elapsed;
    if ( d >= interval) {
      mark += interval;
      return true;
    }
    return false;
  }
};
```

Event Windows, monotonic.

```
var monotonic = EventWorld.make().windows().monotonic(
  new AvgFunction(), new WallClock(0)
);

assert.ok(monotonic != null);

var results = [];
monotonic.on('emit', function(v) {
  results.push(v);
});

monotonic.enqueue(5);
assert.same([], results);

monotonic.tick();
assert.same([5], results);
```

EEP for Node.js (eep.js)

Embedded Event Processing:

- Simple to use. Aggregates Functions and Windowed event processing.

- npm install eep. Use it. Fork it. I'm not a JS guy, I'm learning because of Node.js.

- Fast. CEP engines typically handle ~250K/sec.

- For small N (most common) is 34x - 200x faster than commercial CEP engines.

- But, at a small price. **Simple**. No multi-dimensional, infinite or predicate windows

- Reduces a flood of events into a few in near real time

- Can handle 8-40 million events per second (max, on my laptop). YMMV.

- Combinators may be added. [Ugh, if I need combinators]

# Le **Old** Performance?
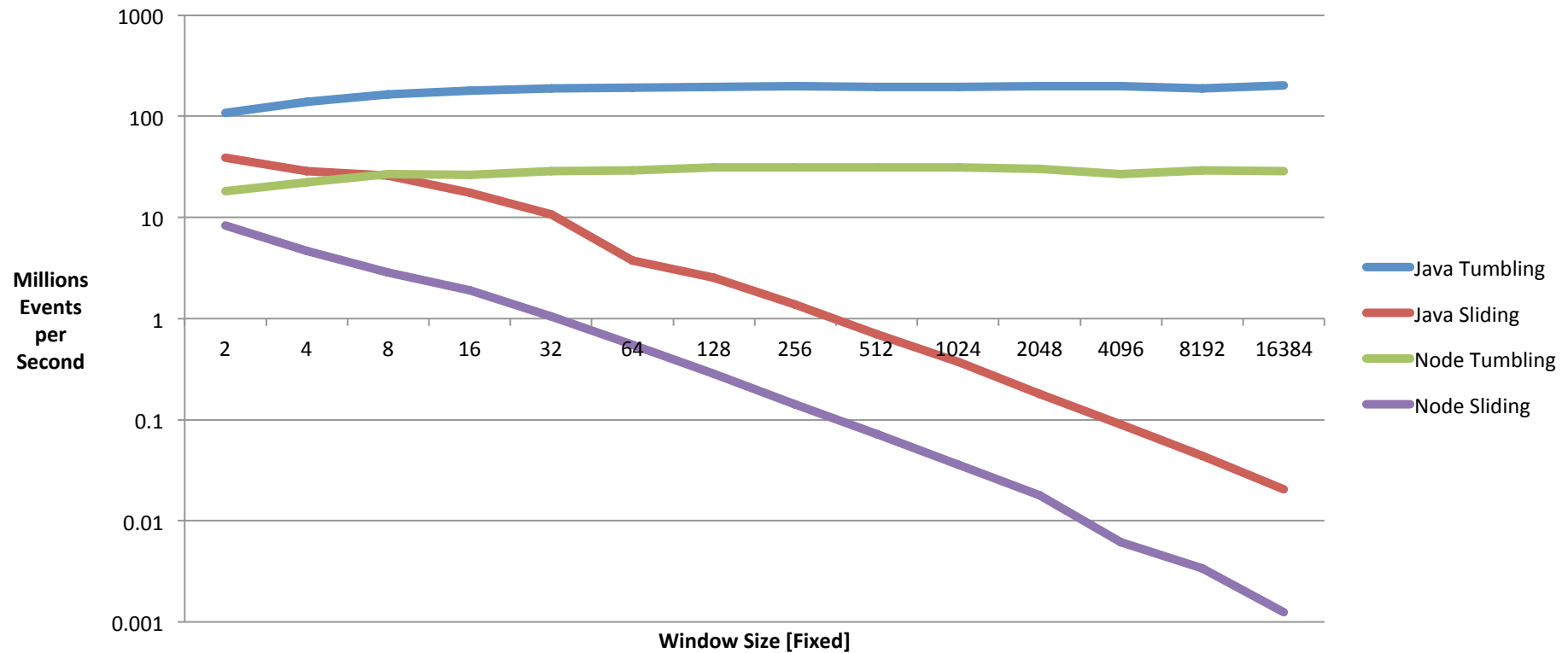
One More Thing ...

- 1 tablespoon of Mechanical Sympathy

- 1 dash of revisiting first principles

- 1 simple observation

- Sliding Windows. $O(N^2)$ <- Wat?

- ... It's the Algorithm, Stoopid!

It's the Algorithm, Stoopid!

```javascript
//
// Sliding ring buffer, emit on 'close'
var old_enqueue = function(buffer,v) {
  var i = idx;
  // Accumulate event to current and any prior active slots
  do {
    buf[i-- % size].accumulate(v);
  } while(i + size > idx && i >= 0); // <- Surprisingly, CEP engines do this
  // If we've hit our size
  if (idx >= mark) {
    // Find the slot
    var x = (i+1) % size;
    // Emit the result
    var r = buf[x].emit();
    self.emit('emit',r);
    // 'close' the active window. 'open' a new window in that slot
    buf[x].init();
  }
  idx+=1;
};
```
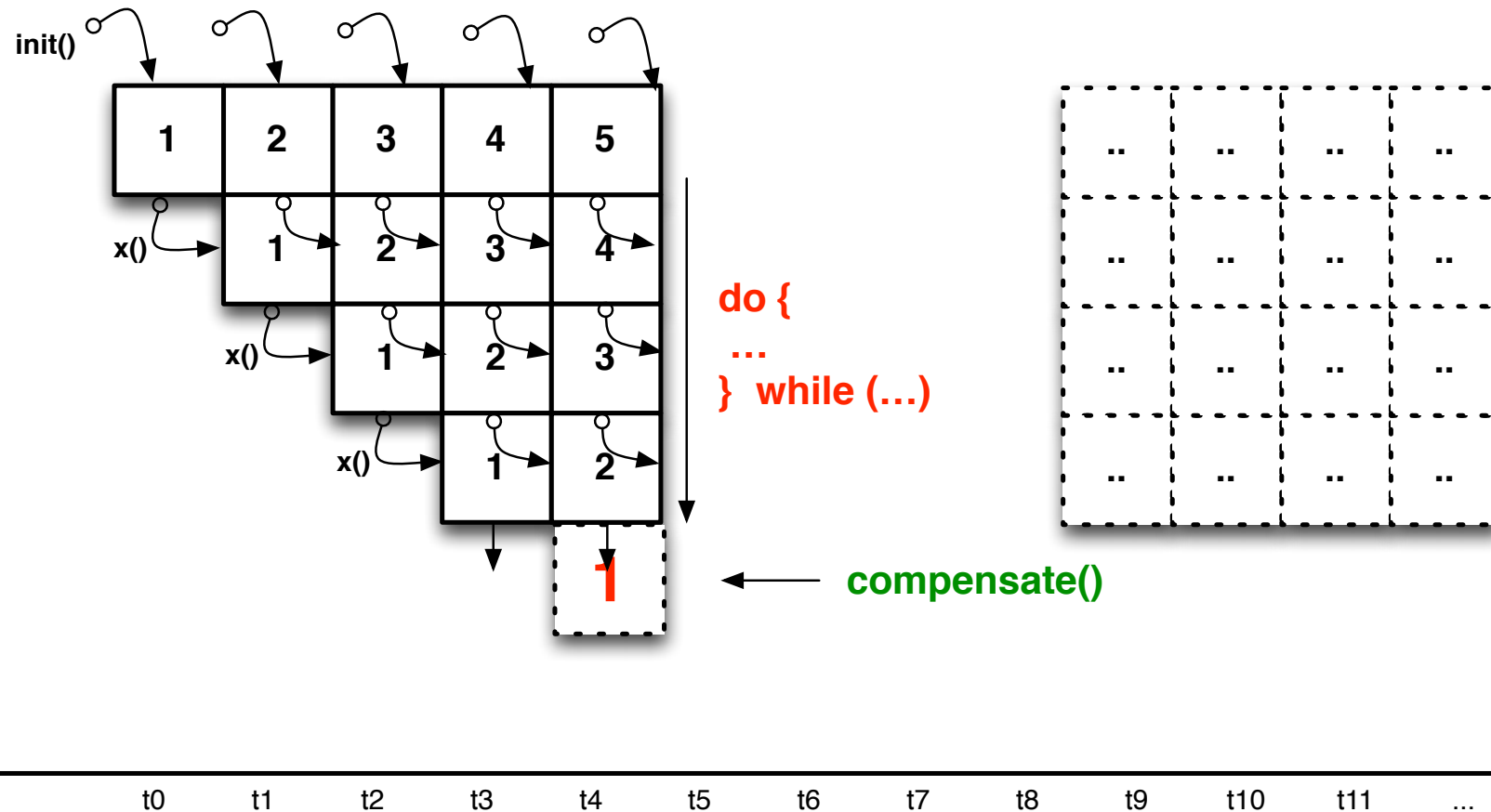
It's the Algorithm, Stoopid!
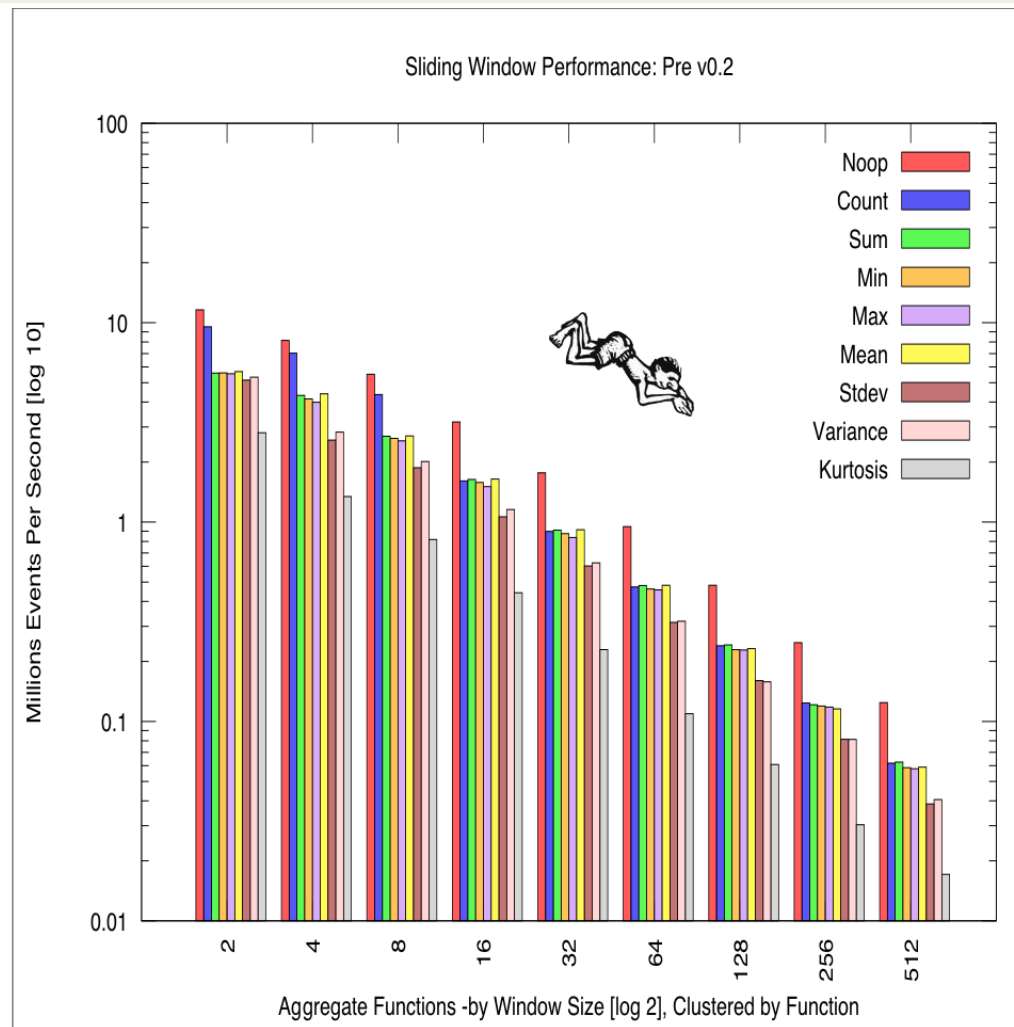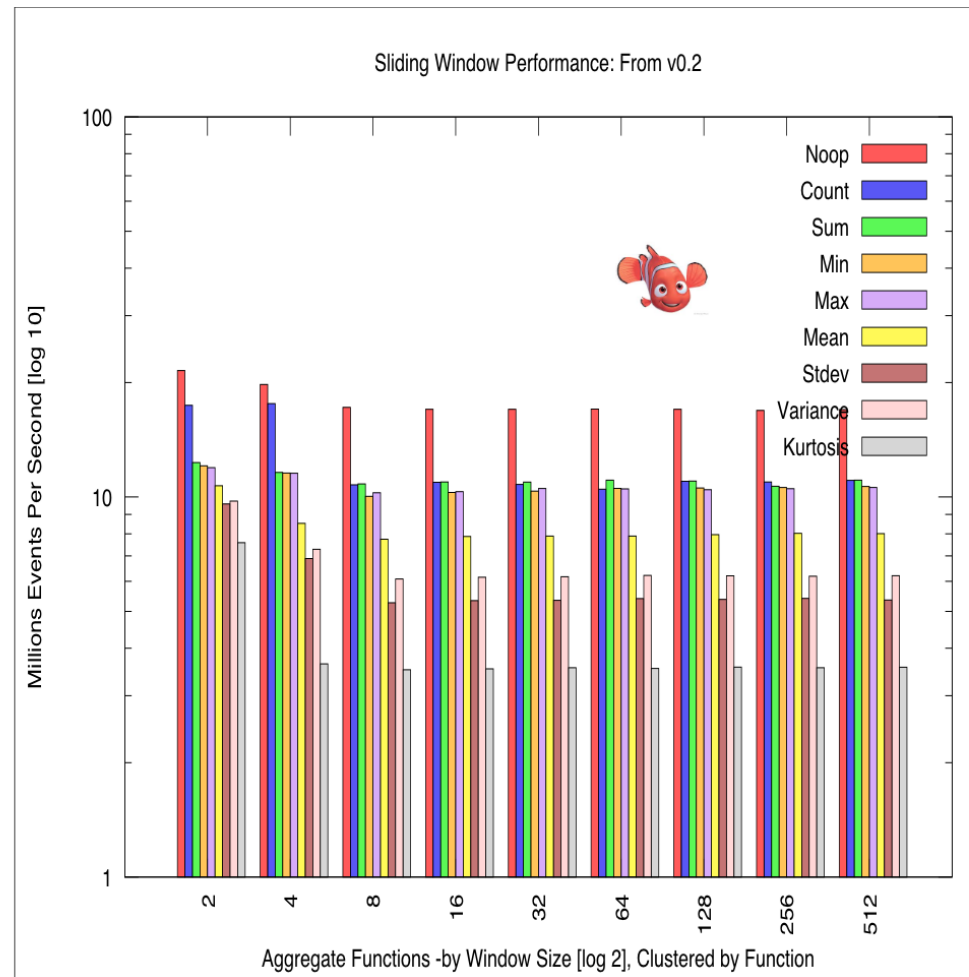
Reversible Aggregation

- Takes us from $O(N^2)$ to $O(N)$ for Sliding windows

```javascript
self.enqueue = function(v) {
  if (idx >= mark) {
    var po = (idx+1) % size;
    fn.accumulate(v);
    self.emit('emit', fn.emit());
    fn.compensate(p[po]);
    p[(idx) % size]=v;
  } else {
    fn.accumulate(v);
    p[idx] = v;
  }
  idx+=1;
};
```

Data On Demand

PUSH ™
TECHNOLOGY

Compensating Aggregates
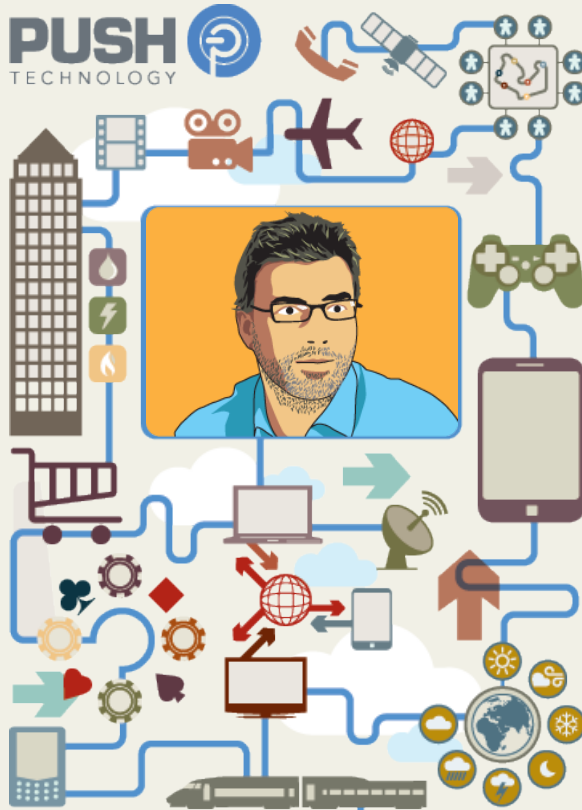
Data On Demand

PUSH
TECHNOLOGY

Le New Performance?

EEP looking forward?

- Streams & Pipes

- Eep for Erlang, Java, ...

- Infinite windows (never open, never close, always 'open')

- Peep – Data parallel eep

  - Continuous Streaming Map Reduce

- Deep – Data analytics with Peep

  - Because I've spent 4+ weeks up to my fourth arse
    in bash, sed, awk, excel and gnuplot.

    Eep all the things, basically.

- Thank you for listening to, having me

- Le twitter: @darachennis

- Github: https://github.com/darach/eep-js

- npm install eep

- EEP.erl 'soon'

- Fork it, Port it, Enjoy it!

Questions?

Darach@PushTechnology.com