Darach Ennis (darach@streambase.com) @darachennis

StreamBase Systems

Erlang Factory London - June 10th 2011

Complex Er[jl]ang Processing with StreamBase: A DSL for Low Latency High Frequency Computing

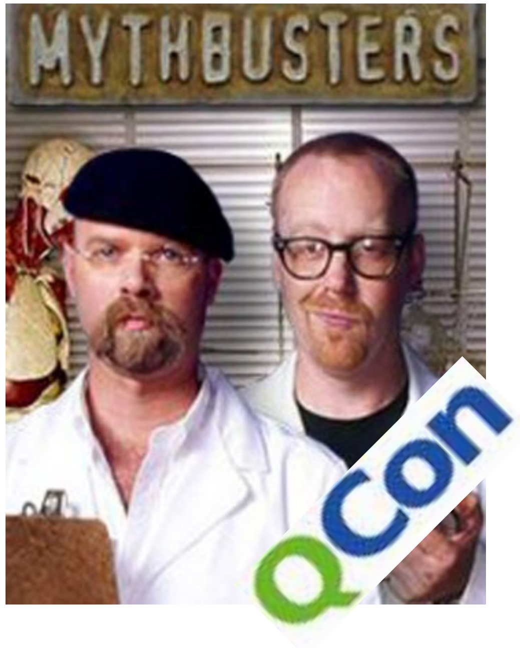
Agenda

- **What is 'Complex Event Processing'**
  - Specifically flow oriented event processing (there are others)
  - Streams & Operators. Windowing, Branching, Combining, Extending
- **A day in the life of a flow programmer**
  - Relativity - Data parallelism, concurrency, latency & throughput
  - Continuity - Continuous Streaming Map Reduce
  - Reliability – High availability, the low latency way
  - Flow, meets Function. Embed Erlang in process via Erjang
- **Integration. Erlang – the ecosystem.**
  - Calling Erlang from StreamBase – Simple & windowed functions
  - Client/Server – Pushing events to/from StreamBase
  - RabbitMQ - Messaging
- **Theft. Erlang – the inspiration. Paxos, in StreamBase**

High Level DSLs : Myth Vs Reality

Myth: **High level domain specific languages are too slow for HFT.**

Reality: **High level domain specific languages can deliver better performance than system programming languages when tailored to a specific task.**



Complex Event Processing aka Event Processing

- **Software organized by events (cf: object/function oriented)**
  - What's an event? What's an object?
    - Something that can trigger processing, can include data.
  - Naturally but not usually represents a "real world" events & observations.
- **Complex Event Processing Platforms**
  - Software stack for event based systems, event driven architectures
  - Event Programming Language – SQL-based, Rules-based, or State-based
  - Commercial and open source: StreamBase, Progress, Microsoft, IBM, Oracle, SAP, Esper, Drools and many more
- **Adopted in financial services and other markets**
  - System monitoring, industrial process control, logistics, defense/ intelligence
- **Other Event Processing Approaches:**
  - Erlang, Scala/Akka, Actors, node.js, .NET Rx

What does a CEP DSL or Language offer?

- **Continuously Observe, Orient, Decide Act (OODA) on event streams**
  - Continuous Incremental Query
  - Pattern matching within or across streams
  - Branch – Split, Causal Split, Filter.
  - Combine - Semi-Join, Union, Gather, Merge, Join, Pattern
  - Windows – Process sets of streaming data
    - Sliding or Tumbling, Overlapping or Non-Overlapping, Gaps or No Gaps
    - Finite (1 second, 1000 tuples), Infinite
    - Emission Policies: On Close, Every odd message
    - Predicate based – Roll your own window type
  - State Management – In memory, CSV files, CSV sockets, RDBMS, Parallel DBMSs, Column Stores, KV stores, NoSQL, NewSQL…
  - Nice to have:
    - Declarative concurrency, Interface Polymorphism, Distribution, Extensible

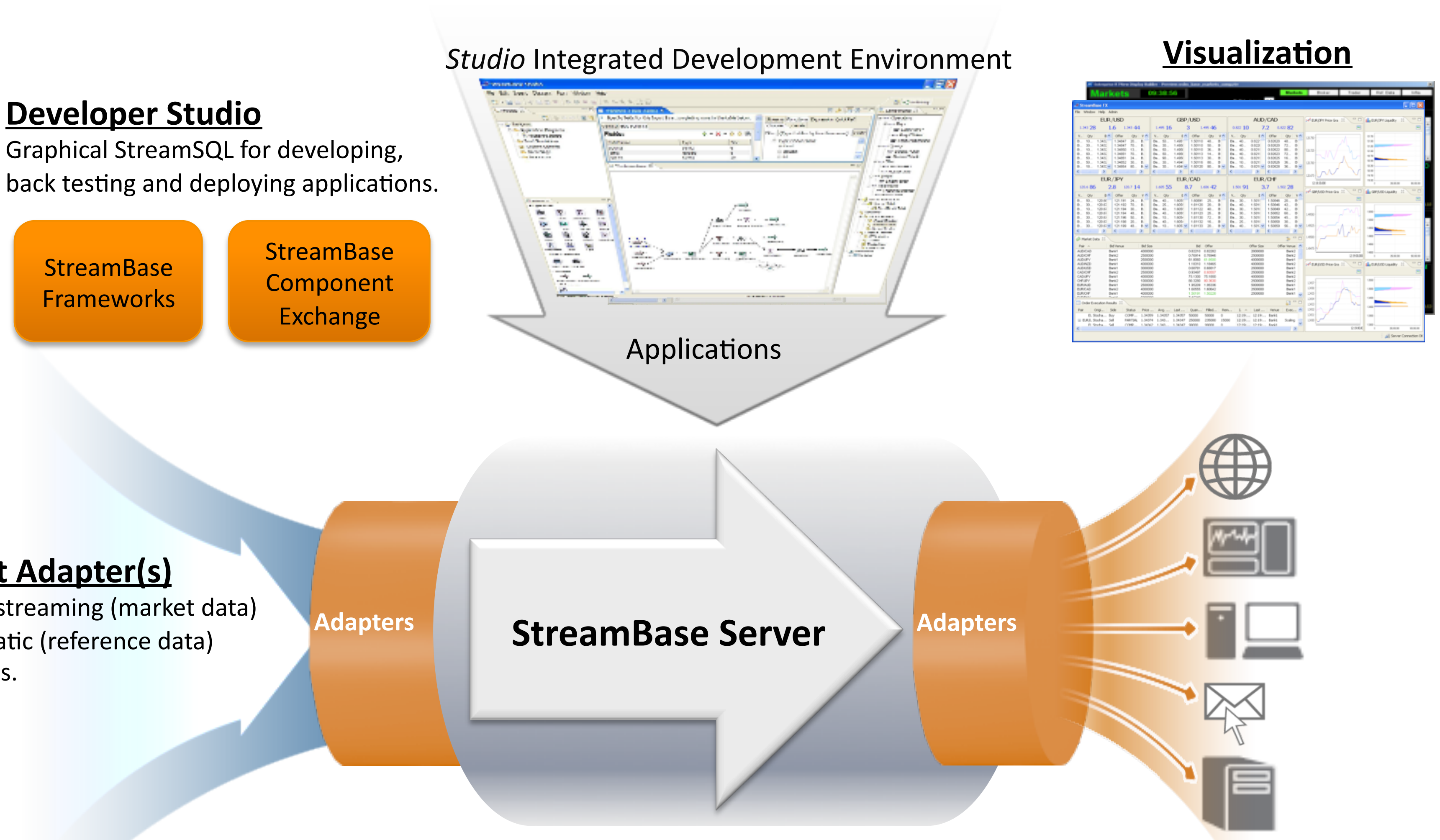
Wikipedia:
John Boyd,
OODA Loop

Challenges for CEP

- **‘Über’ Ultra Low Latency?**
  - Sub-milli is standard, sub-100-micro is desirable. Less is more!
- **Large Data Volumes**
  - Hundreds of thousands of events, thousands of decisions, per thread.
  - Big Data. ~Hundreds of SMP CEP nodes.
- **Demanding Operational Environment**
  - 24x7, 365 – in critical environments (trading, surveillance, utilities)
- **Sophisticated Data Processing (sometimes)**
  - Options pricing, yield curves, risk metrics, smart grid capacity planning, fraud detection.
- **How it’s done (QCON London 2011):**
  - How LMAX did it? http://bit.ly/fUeS0P
  - How we did it? http://bit.ly/hM6NAP <- Our CTO Richard Tibbetts talk

StreamBase Event Processing Platform

Developer Studio
Graphical StreamSQL for developing, back testing and deploying applications.



Input Adapter(s)
Inject streaming (market data) and static (reference data) sources.

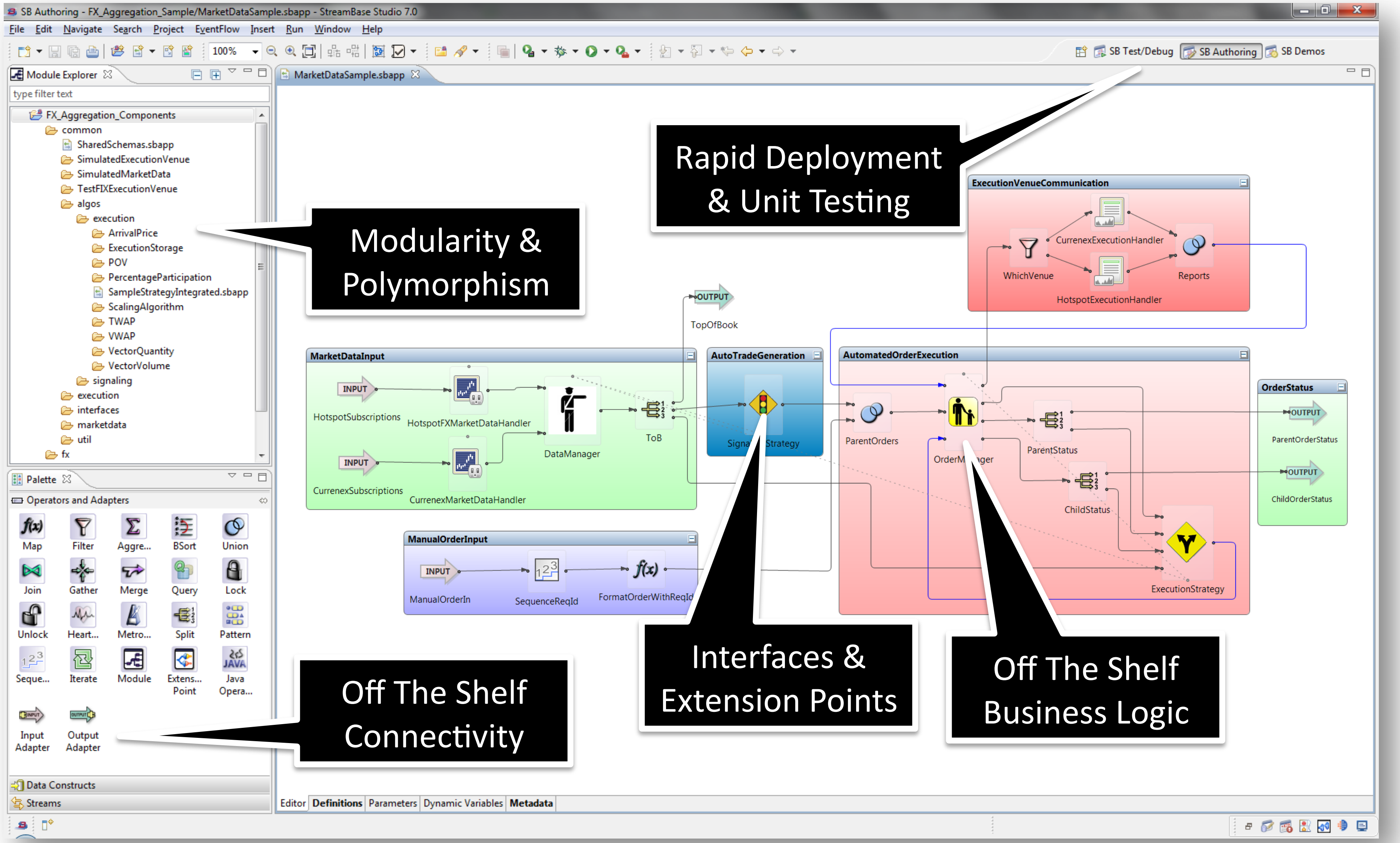
Event Processing Server
High performance optimized engine can process events at market data speeds.

Output Adapter(s)
Send results to systems, users, user screens and databases.

How did we do it?

- **Compilation and Static Analysis**
  - Design the language for it
- **Modular abstraction, interfaces**
  - Quants and Developers Collaborate, share code
- **Bytecode generation and the Janino compiler**
  - Optimized bytecodes, in-memory generation
- **Garbage optimization**
  - Pooling, data class, invasive collections
- **Integrations, C++ and Java plugins**
  - Efficient native interfaces, Hardware acceleration
- **Adapter API, FIX Messaging**
  - Threading and API structure for ultra low latency
- **Parallelism, Clustering, Lanes and Tiers**
  - Scalability, with a latency bias.
- **Modularity through Named Data Formats, Schemas**
  - Sharing data and semantics between apps

StreamBase StreamSQL EventFlow



Rapid Deployment & Unit Testing

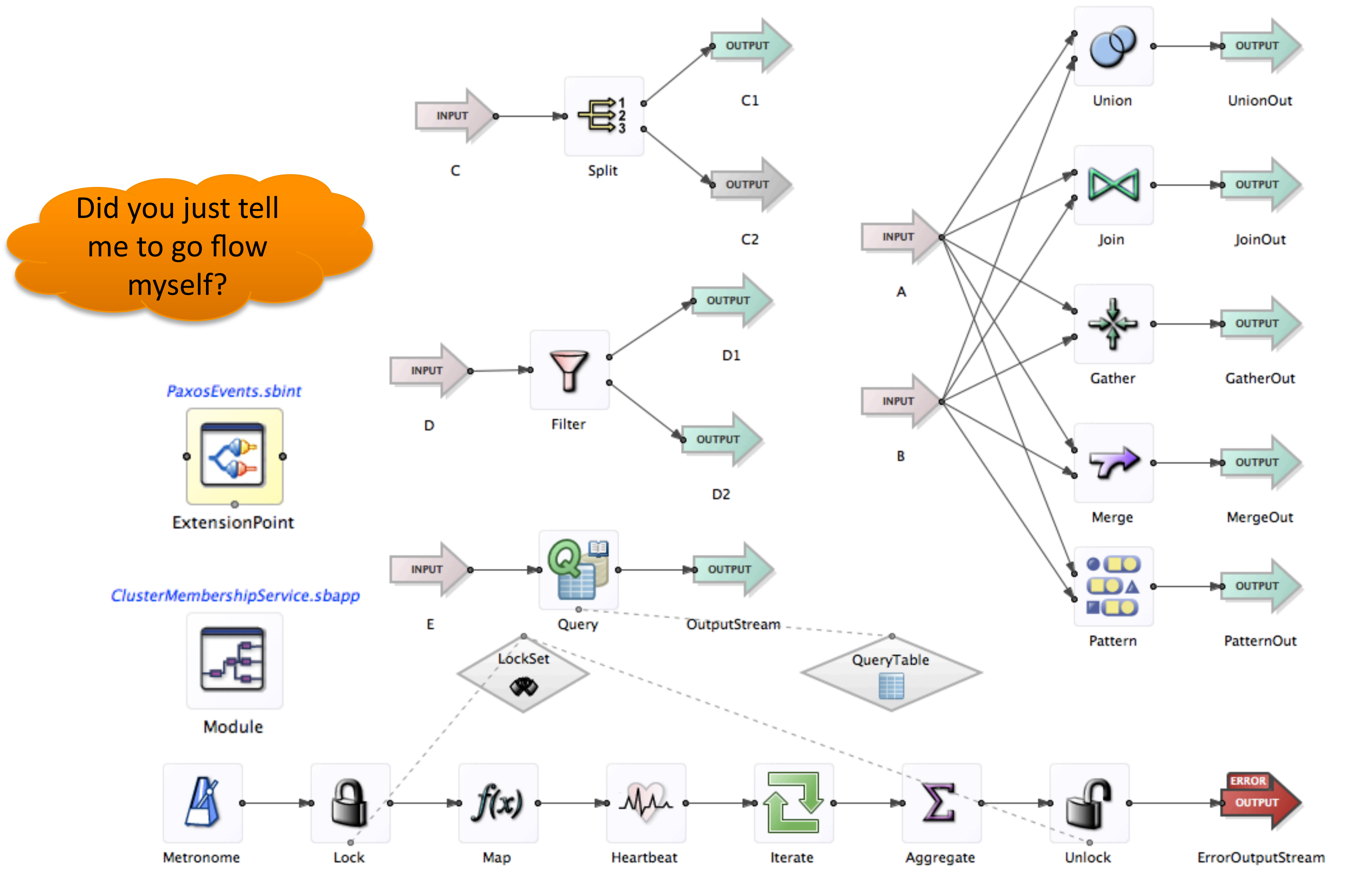
Modularity & Polymorphism

Interfaces & Extension Points

Off The Shelf Business Logic

Off The Shelf Connectivity

Operators – Hi Erlang. Hello StreamBase

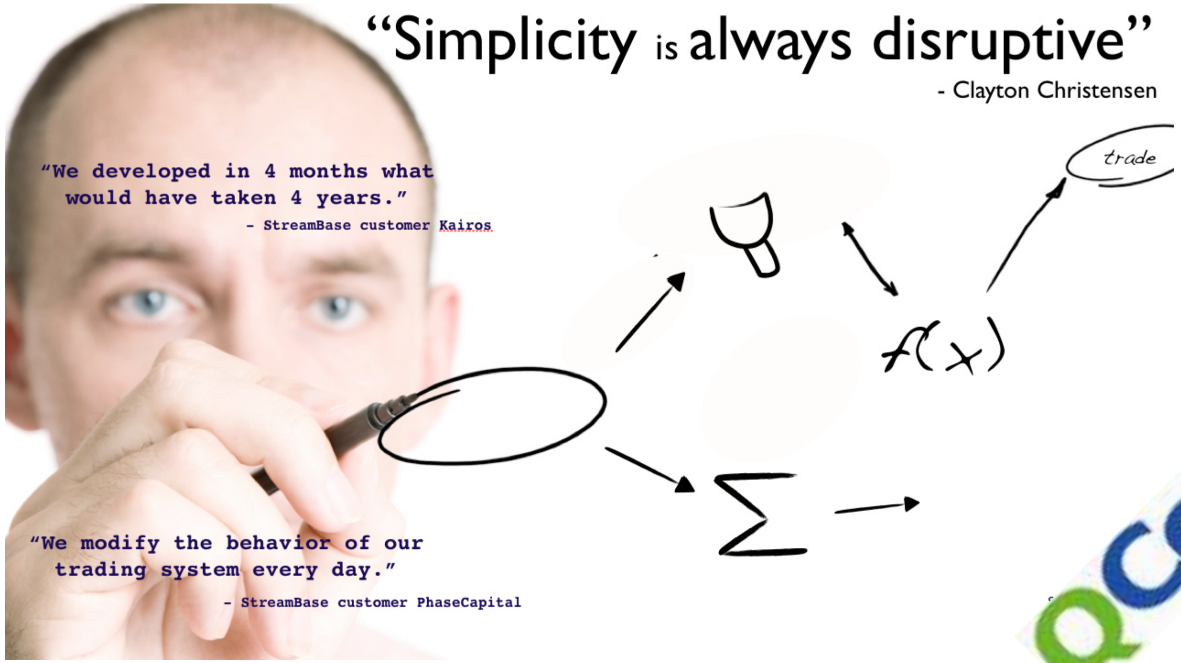


Agenda

- **~~What is 'Complex Event Processing'~~**
  - ~~Specifically flow oriented event processing (there are others)~~
  - ~~Streams & Operators. Windowing, Branching, Combining, Extending~~
- **A day in the life of a flow programmer**
  - Relativity - Data parallelism, concurrency, latency & throughput
  - Continuity - Continuous Streaming Map Reduce
  - Reliability – High availability, the low latency way
  - Flow, meets Function. Embed Erlang in process via Erjang
- **Integration. Erlang – the ecosystem.**
  - Calling Erlang from StreamBase – Simple & windowed functions
  - Client/Server – Pushing events to/from StreamBase
  - RabbitMQ - Messaging
- **Theft. Erlang – the inspiration. Paxos, in StreamBase**

A day in the life… Why a DSL?

- **High level – Windowing, Combination & Pattern Matching Streams**
- **Graphical – 'See' the flow, dependencies, pathways**
- **Fast, Flexible SDLC – Deploy new algorithms, continuously**
- **Understandable – Rise to the abstraction**
- **Flexible**


"Simplicity is always disruptive"
- Clayton Christensen
"We developed in 4 months what would have taken 4 years."
- StreamBase customer Kairos
"We modify the behavior of our trading system every day."
- StreamBase customer PhaseCapital
trade
f(x)
Σ

A day in the life.. More is more!

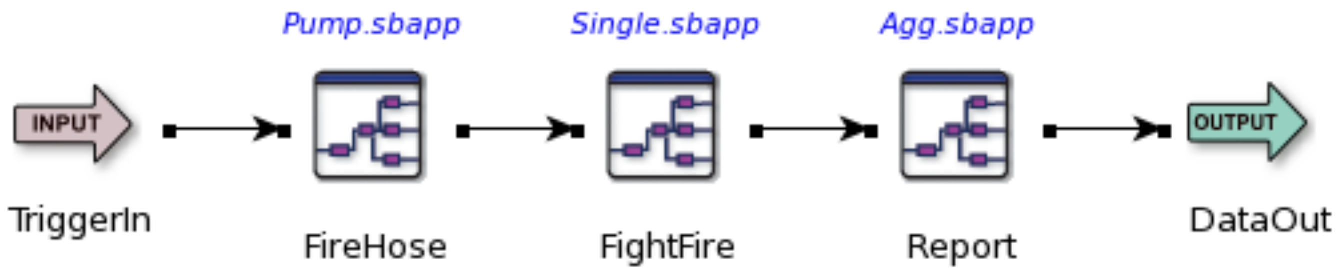
Ultra Low Latency

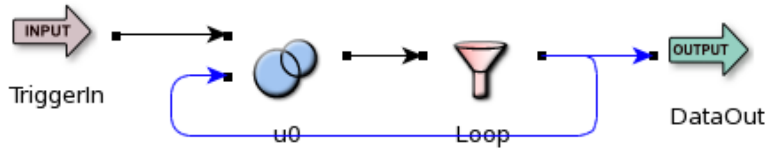
Capital Markets

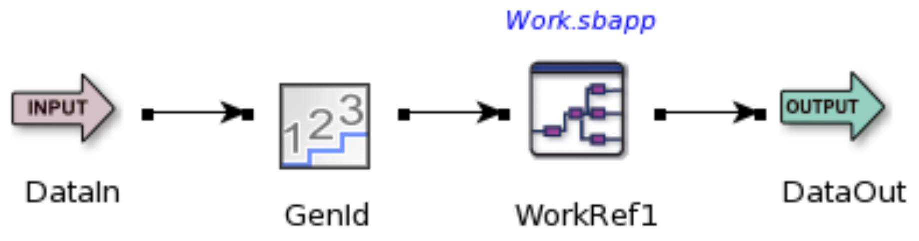
High Throughput

Big Data

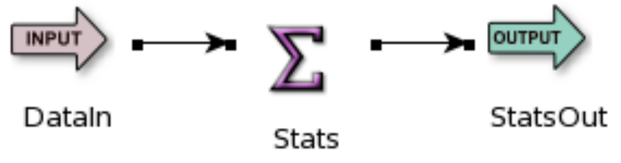
Make it work. Measure it. - Baseline 'Noop' Performance



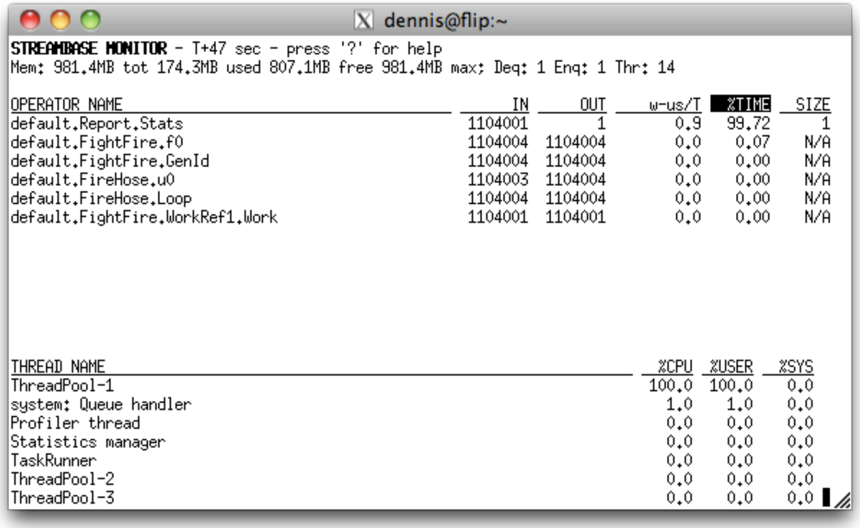




WARNING: Micro-benchmark. YMMV



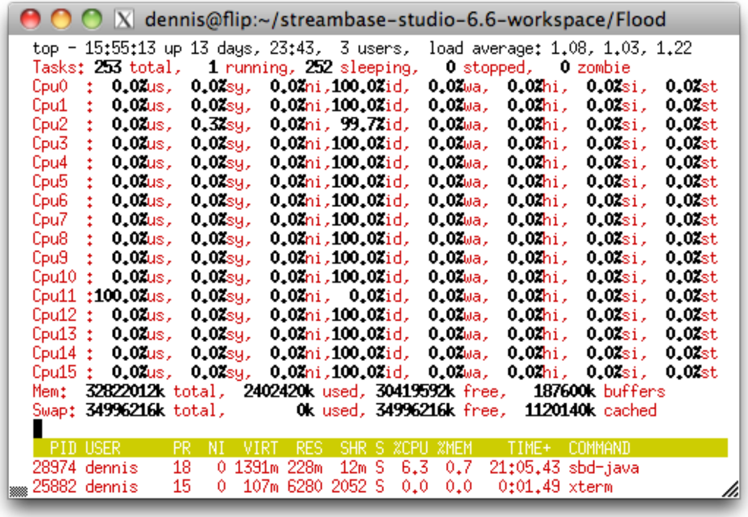
Concurrency – Baseline - Results



dennis@flip:~

```
STREAMBASE MONITOR - T+47 sec - press '?' for help
Mem: 981.4MB tot 174.3MB used 807.1MB free 981.4MB max; Deq: 1 Enq: 1 Thr: 14

OPERATOR NAME                                 IN       OUT   w-us/T  %TIME  SIZE
default.Report.Stats                      1104001        1     0.9   99.72     1
default.FightFire.f0                      1104004  1104004     0.0    0.07   N/A
default.FightFire.GenId                   1104004  1104004     0.0    0.00   N/A
default.FireHose.u0                       1104003  1104004     0.0    0.00   N/A
default.FireHose.Loop                     1104004  1104004     0.0    0.00   N/A
default.FightFire.WorkRef1.Work           1104001  1104001     0.0    0.00   N/A



THREAD NAME                                          %CPU  %USER   %SYS
ThreadPool-1                                        100.0  100.0    0.0
system: Queue handler                                 1.0    1.0    0.0
Profiler thread                                       0.0    0.0    0.0
Statistics manager                                    0.0    0.0    0.0
TaskRunner                                            0.0    0.0    0.0
ThreadPool-2                                          0.0    0.0    0.0
ThreadPool-3                                          0.0    0.0    0.0
```

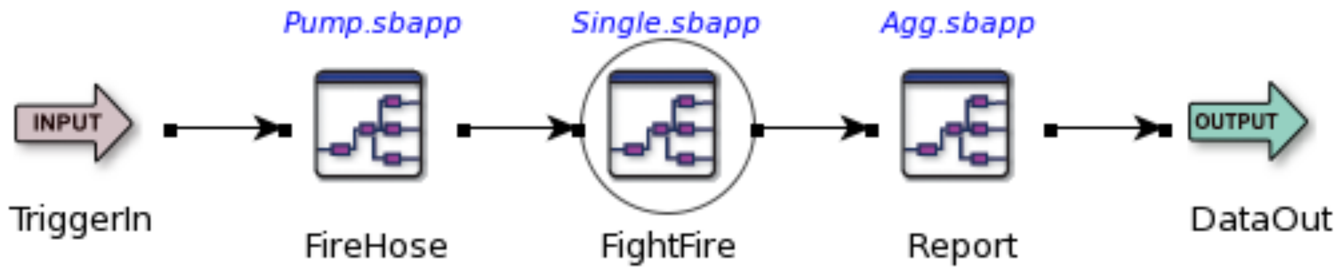
Nice. 1.1 million events/sec



dennis@flip:~/streambase-studio-6.6-workspace/Flood

```
top - 15:55:13 up 13 days, 23:43,  3 users,  load average: 1.08, 1.03, 1.22
Tasks: 253 total,   1 running, 252 sleeping,   0 stopped,   0 zombie
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.0%us,  0.3%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu4  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu5  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu6  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu7  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu8  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu9  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu10 :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu11 :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu12 :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu13 :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu14 :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu15 :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  32822012k total,  2402420k used, 30419592k free,   187600k buffers
Swap: 34996216k total,        0k used, 34996216k free,  1120140k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
28974 dennis    18   0 1391m 228m  12m S  6.3  0.7  21:05.43 sbd-java
25882 dennis    15   0  107m 6280 2052 S  0.0  0.0   0:01.49 xterm
```

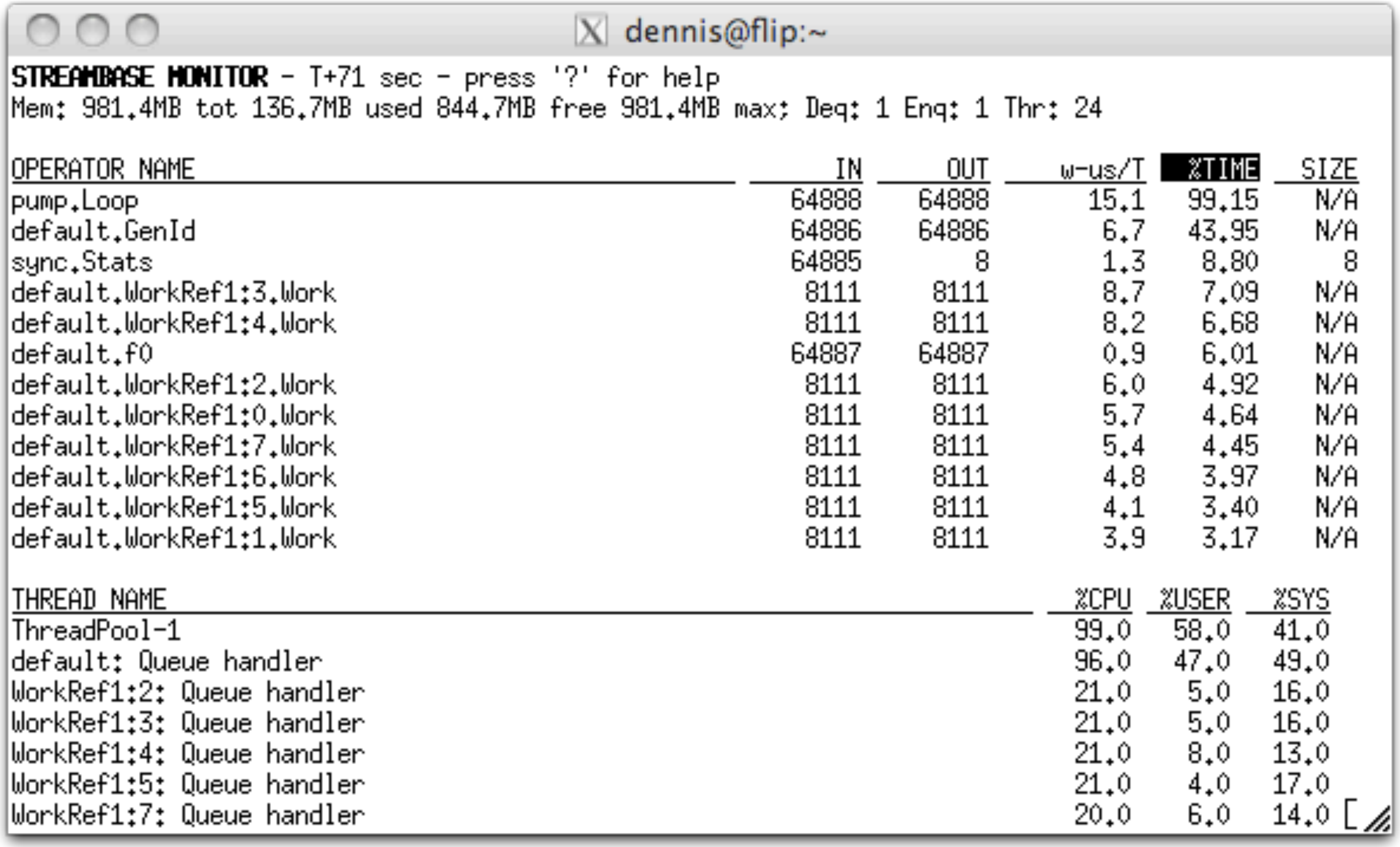
Parallelize it – The 'Convenient, but incorrect' way



Execution Order & Concurrency:

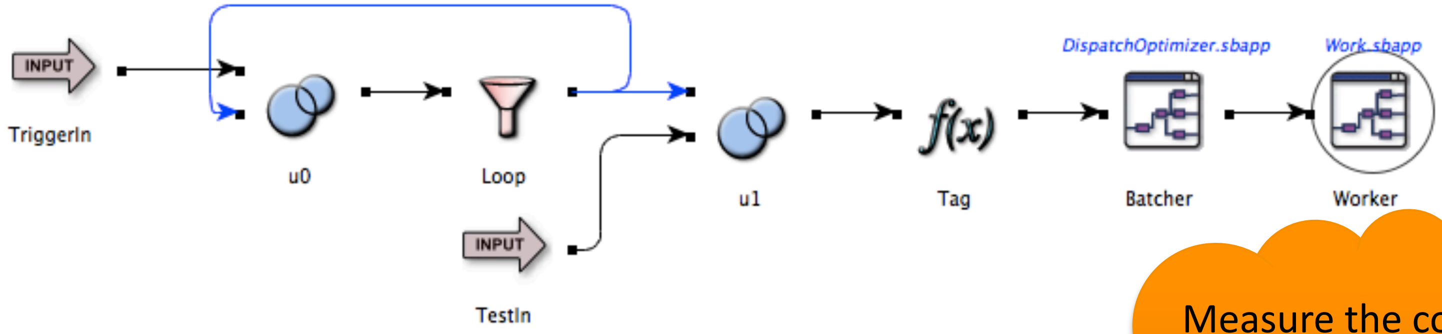
Rule 1 –	Each event processed to completion left-right
Rule 2 –	Branches processed sequentially
Rule 3 –	Outputs are processed sequentially
Rule 4 –	Module output processed to completion immediately
Rule 5 –	One operator executed at a time

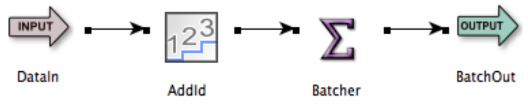
Gah! WTF?!

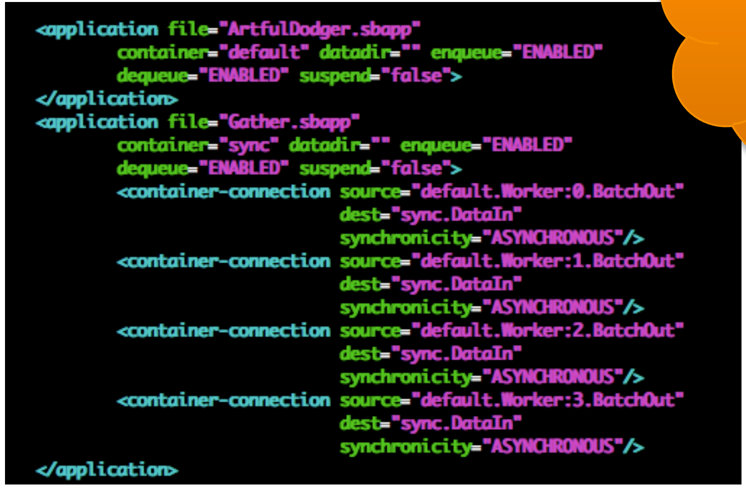


```
STREAMBASE MONITOR - T+71 sec - press '?' for help
Mem: 981.4MB tot 136.7MB used 844.7MB free 981.4MB max; Deq: 1 Enq: 1 Thr: 24

OPERATOR NAME                                       IN      OUT     w-us/T   %TIME   SIZE
pump.Loop                                        64888    64888       15.1   99.15    N/A
default.GenId                                    64886    64886        6.7   43.95    N/A
sync.Stats                                       64885        8        1.3    8.80      8
default.WorkRef1:3.Work                           8111     8111        8.7    7.09    N/A
default.WorkRef1:4.Work                           8111     8111        8.2    6.68    N/A
default.f0                                       64887    64887        0.9    6.01    N/A
default.WorkRef1:2.Work                           8111     8111        6.0    4.92    N/A
default.WorkRef1:0.Work                           8111     8111        5.7    4.64    N/A
default.WorkRef1:7.Work                           8111     8111        5.4    4.45    N/A
default.WorkRef1:6.Work                           8111     8111        4.8    3.97    N/A
default.WorkRef1:5.Work                           8111     8111        4.1    3.40    N/A
default.WorkRef1:1.Work                           8111     8111        3.9    3.17    N/A

THREAD NAME                                                      %CPU   %USER   %SYS
ThreadPool-1                                                     99.0    58.0   41.0
default: Queue handler                                           96.0    47.0   49.0
WorkRef1:2: Queue handler                                        21.0     5.0   16.0
WorkRef1:3: Queue handler                                        21.0     5.0   16.0
WorkRef1:4: Queue handler                                        21.0     8.0   13.0
WorkRef1:5: Queue handler                                        21.0     4.0   17.0
WorkRef1:7: Queue handler                                        20.0     6.0   14.0 [
```

Optimize it! Observation #1

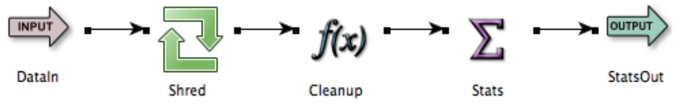




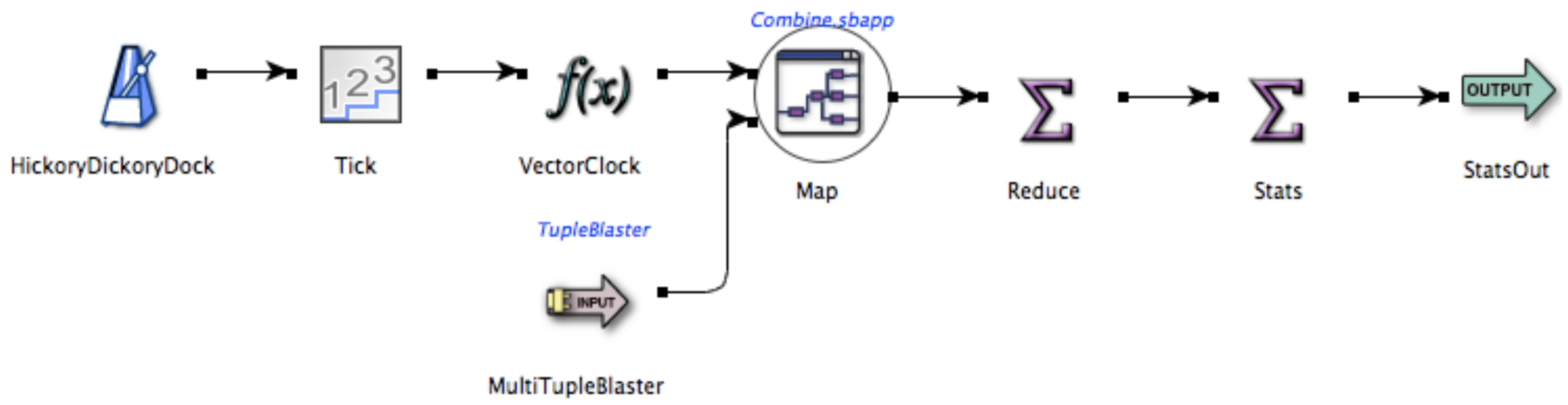


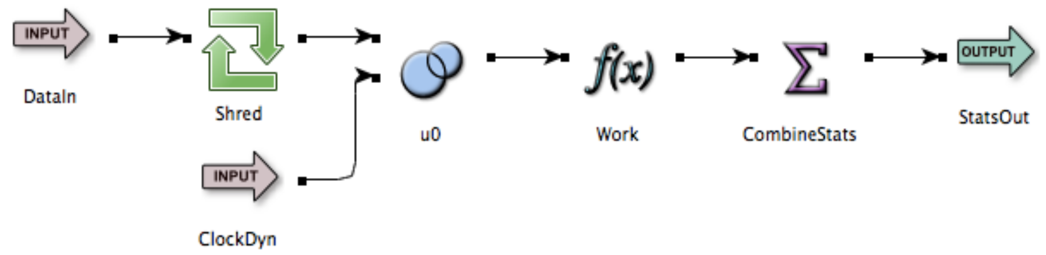
```
<application file="ArtfulDodger.sbapp"
        container="default" datadir="" enqueue="ENABLED"
        dequeue="ENABLED" suspend="false">
</application>
<application file="Gather.sbapp"
        container="sync" datadir="" enqueue="ENABLED"
        dequeue="ENABLED" suspend="false">
        <container-connection source="default.Worker:0.BatchOut"
                              dest="sync.DataIn"
                              synchronicity="ASYNCHRONOUS"/>
        <container-connection source="default.Worker:1.BatchOut"
                              dest="sync.DataIn"
                              synchronicity="ASYNCHRONOUS"/>
        <container-connection source="default.Worker:2.BatchOut"
                              dest="sync.DataIn"
                              synchronicity="ASYNCHRONOUS"/>
        <container-connection source="default.Worker:3.BatchOut"
                              dest="sync.DataIn"
                              synchronicity="ASYNCHRONOUS"/>
</application>
```

Measure the cost of thread context switches. Dodge & Burn!

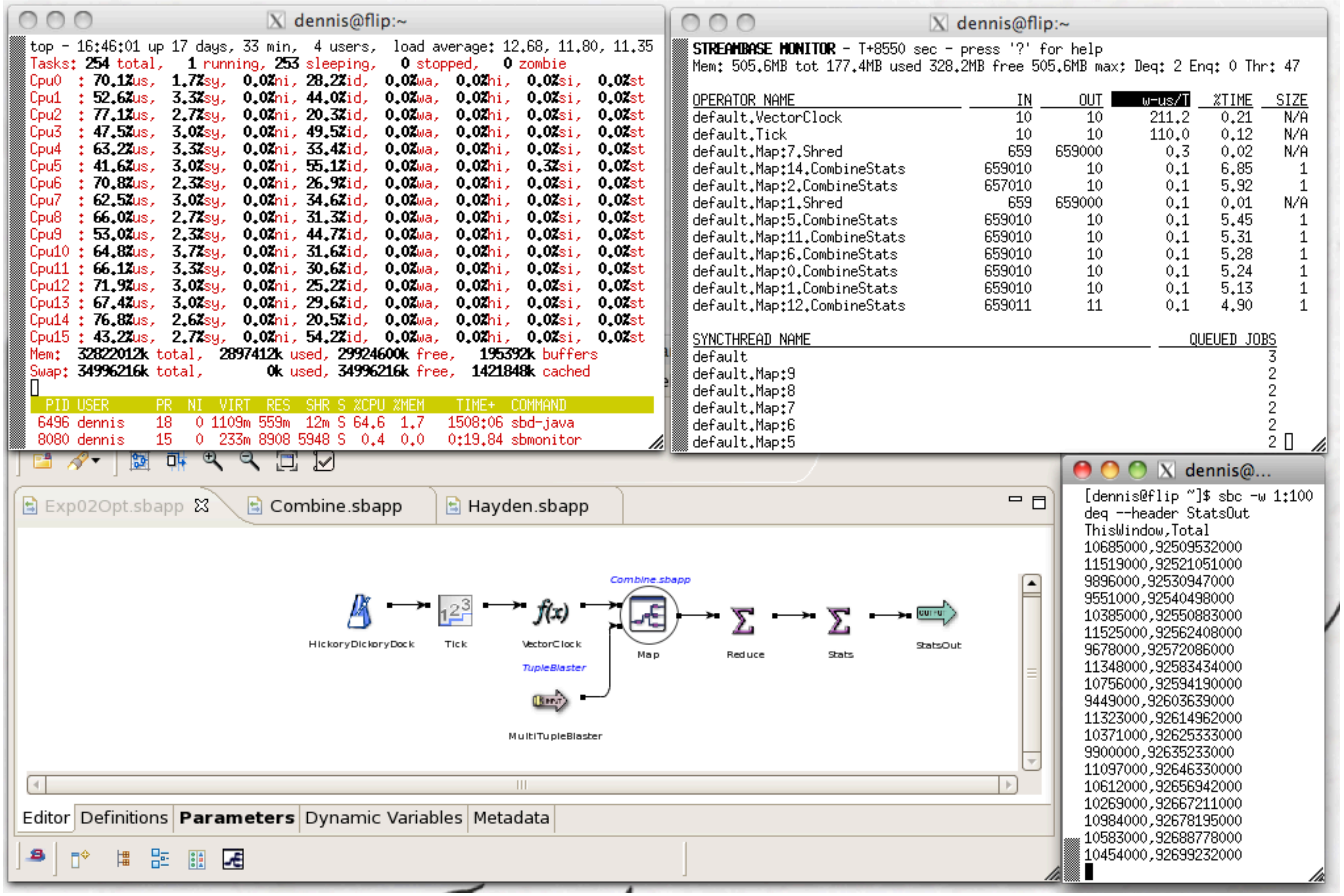


Optimize it! Observation #2 – It's Map/Combine/Reduce





CSMR – A 'pattern' for low latency high throughput?



dennis@flip:~

```
top - 16:46:01 up 17 days, 33 min,  4 users,  load average: 12.68, 11.80, 11.35
Tasks: 254 total,   1 running, 253 sleeping,   0 stopped,   0 zombie
Cpu0  : 70.1%us,  1.7%sy,  0.0%ni, 28.2%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  : 52.6%us,  3.3%sy,  0.0%ni, 44.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  : 77.1%us,  2.7%sy,  0.0%ni, 20.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  : 47.5%us,  3.0%sy,  0.0%ni, 49.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu4  : 63.2%us,  3.3%sy,  0.0%ni, 33.4%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu5  : 41.6%us,  3.0%sy,  0.0%ni, 55.1%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu6  : 70.8%us,  2.3%sy,  0.0%ni, 26.9%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu7  : 62.5%us,  3.0%sy,  0.0%ni, 34.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu8  : 66.0%us,  2.7%sy,  0.0%ni, 31.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu9  : 53.0%us,  2.3%sy,  0.0%ni, 44.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu10 : 64.8%us,  3.7%sy,  0.0%ni, 31.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu11 : 66.1%us,  3.3%sy,  0.0%ni, 30.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu12 : 71.9%us,  3.0%sy,  0.0%ni, 25.2%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu13 : 67.4%us,  3.0%sy,  0.0%ni, 29.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu14 : 76.8%us,  2.6%sy,  0.0%ni, 20.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu15 : 43.2%us,  2.7%sy,  0.0%ni, 54.2%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  32822012k total,  2897412k used, 29924600k free,   195392k buffers
Swap: 34996216k total,        0k used, 34996216k free,  1421848k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6496 dennis    18   0 1109m 559m  12m S 64.6  1.7   1508:06 sbd-java
 8080 dennis    15   0  233m 8908 5948 S  0.4  0.0   0:19.84 sbmonitor
```

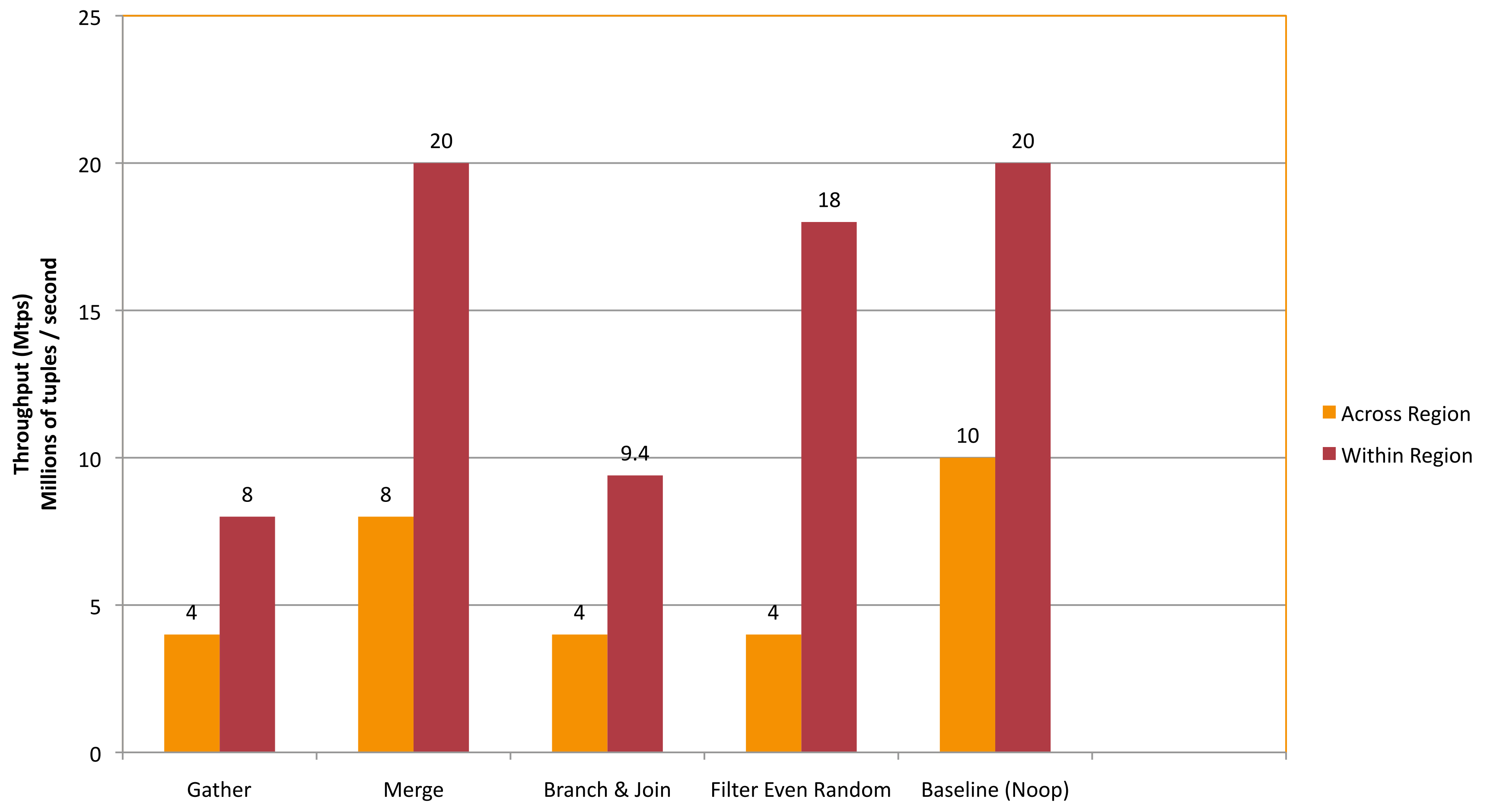
dennis@flip:~

```
STREAMBASE MONITOR - T+8550 sec - press '?' for help
Mem: 505.6MB tot 177.4MB used 328.2MB free 505.6MB max; Deq: 2 Enq: 0 Thr: 47

OPERATOR NAME                        IN      OUT    w-us/T   %TIME   SIZE
default.VectorClock                  10       10     211.2    0.21    N/A
default.Tick                         10       10     110.0    0.12    N/A
default.Map:7.Shred                 659   659000       0.3    0.02    N/A
default.Map:14.CombineStats      659010       10       0.1    6.85      1
default.Map:2.CombineStats       657010       10       0.1    5.92      1
default.Map:1.Shred                 659   659000       0.1    0.01    N/A
default.Map:5.CombineStats       659010       10       0.1    5.45      1
default.Map:11.CombineStats      659010       10       0.1    5.31      1
default.Map:6.CombineStats       659010       10       0.1    5.28      1
default.Map:0.CombineStats       659010       10       0.1    5.24      1
default.Map:1.CombineStats       659010       10       0.1    5.13      1
default.Map:12.CombineStats      659011       11       0.1    4.90      1

SYNCTHREAD NAME                                           QUEUED JOBS
default                                                             3
default.Map:9                                                       2
default.Map:8                                                       2
default.Map:7                                                       2
default.Map:6                                                       2
default.Map:5                                                       2
```

dennis@...

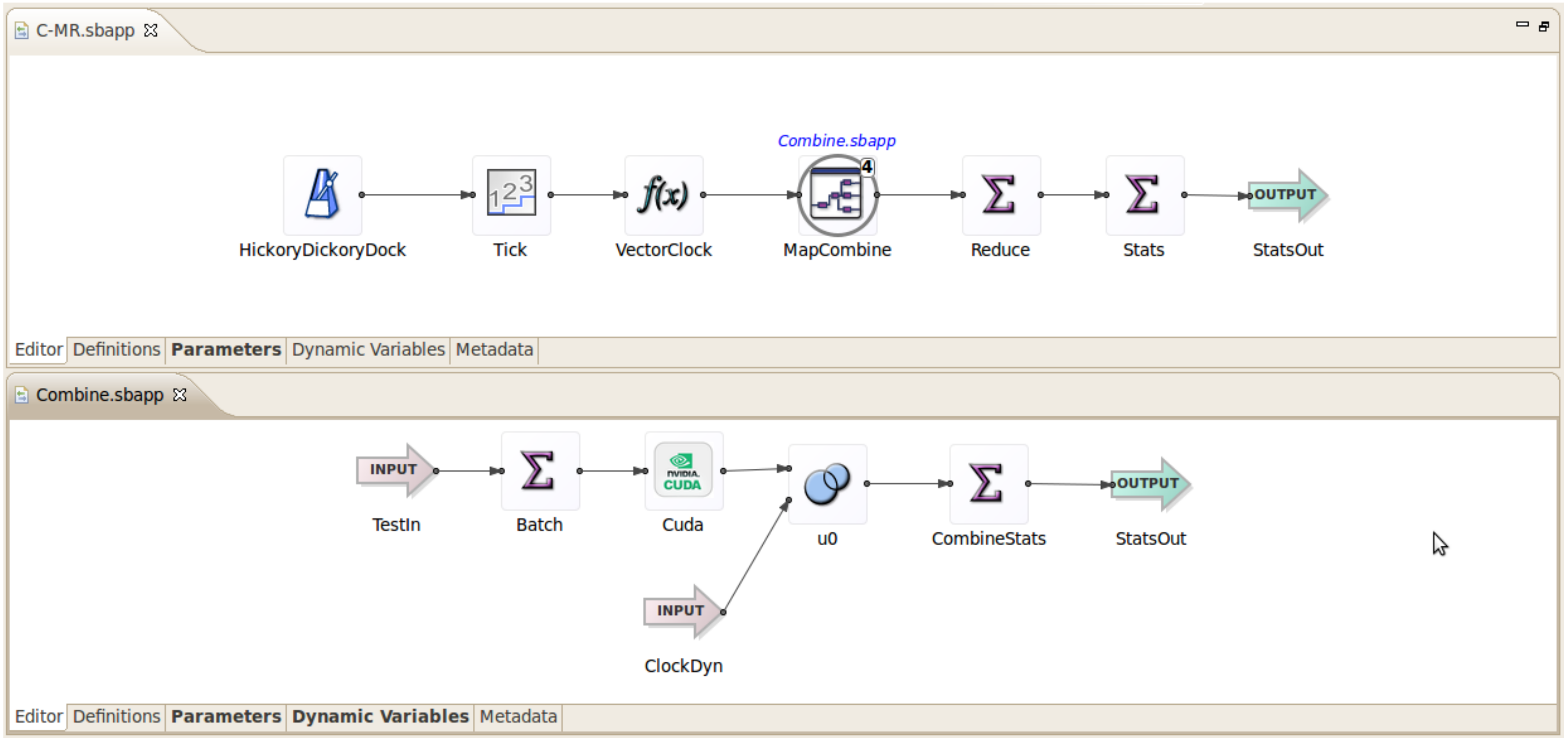
```
[dennis@flip ~]$ sbc -w 1:100
deq --header StatsOut
ThisWindow,Total
10685000,92509532000
11519000,92521051000
9896000,92530947000
9551000,92540498000
10385000,92550883000
11525000,92562408000
9678000,92572086000
11348000,92583434000
10756000,92594190000
9449000,92603639000
11323000,92614962000
10371000,92625333000
9900000,92635233000
11097000,92646330000
10612000,92656942000
10269000,92667211000
10984000,92678195000
10583000,92688778000
10454000,92699232000
```

CSMR – A ‘pattern’ for low latency high throughput? Yup!



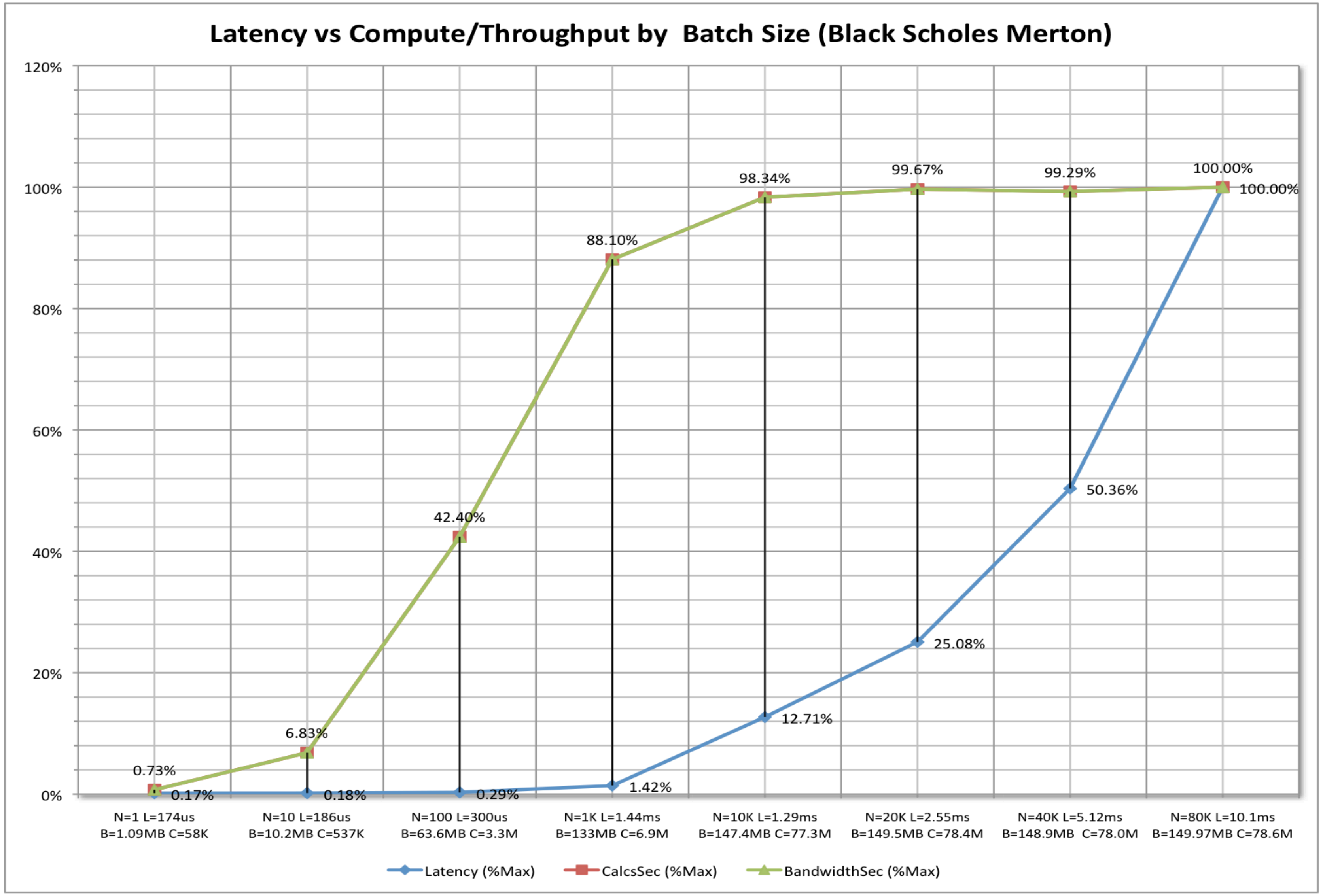
Benchmark HW: Dell RS510, 8 core, 32 GB RAM, 2.4GHz IA-64, OS: RHEL 5u3, SB 7.0.1.2

CSMR + GPUs / FPGAs? Sure!

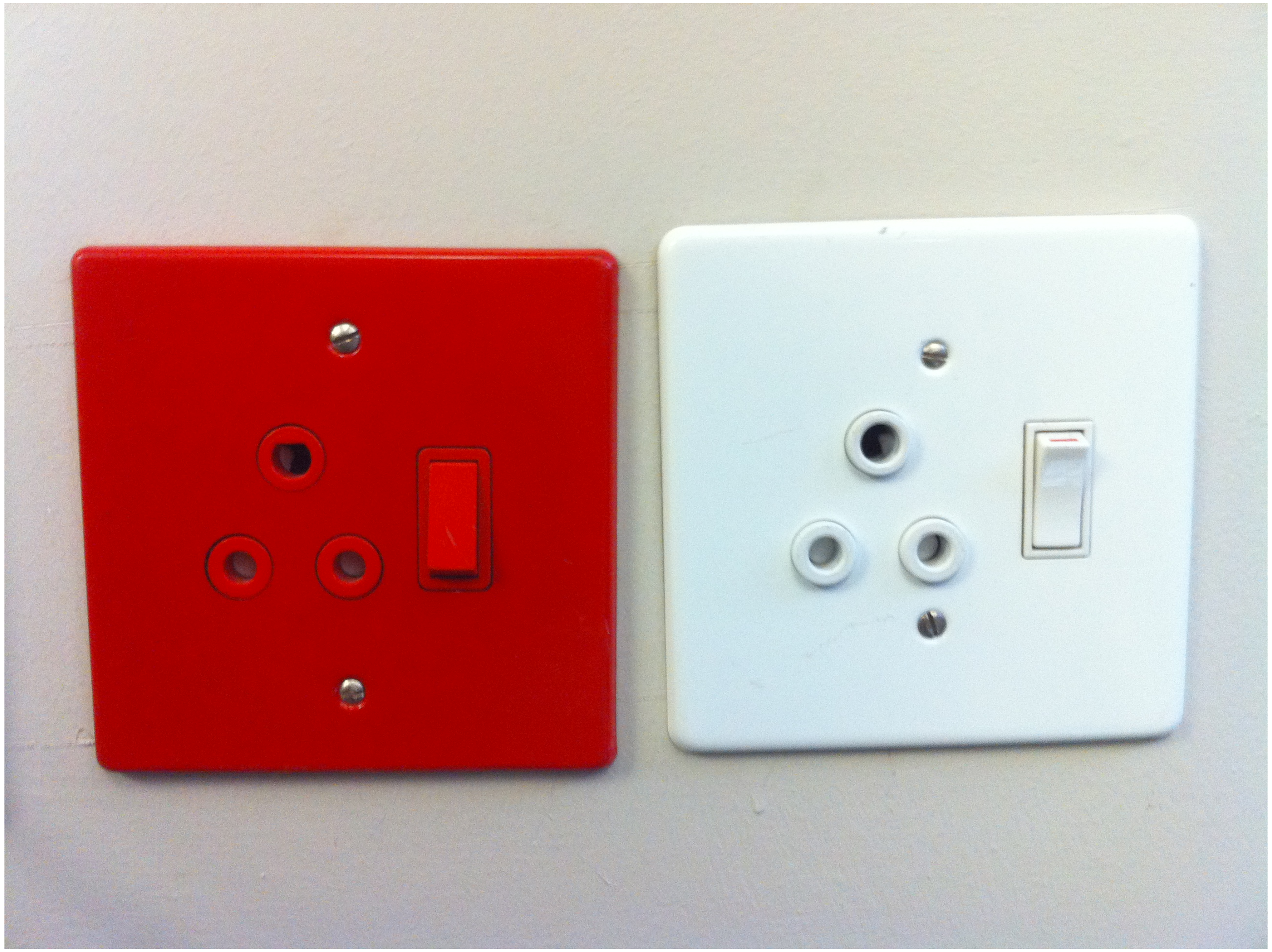


We can build on SMP and cluster wide distribution algorithms to further optimize interactions with accelerated compute technologies. We can exploit accelerated hardware messaging where regular network IO is insufficient for distribution results over 1Ge or regular networking technologies.
The pattern above is a continuous streaming variant of Map/Reduce in EventFlow.

Similar Optimization's apply to GPUs too!

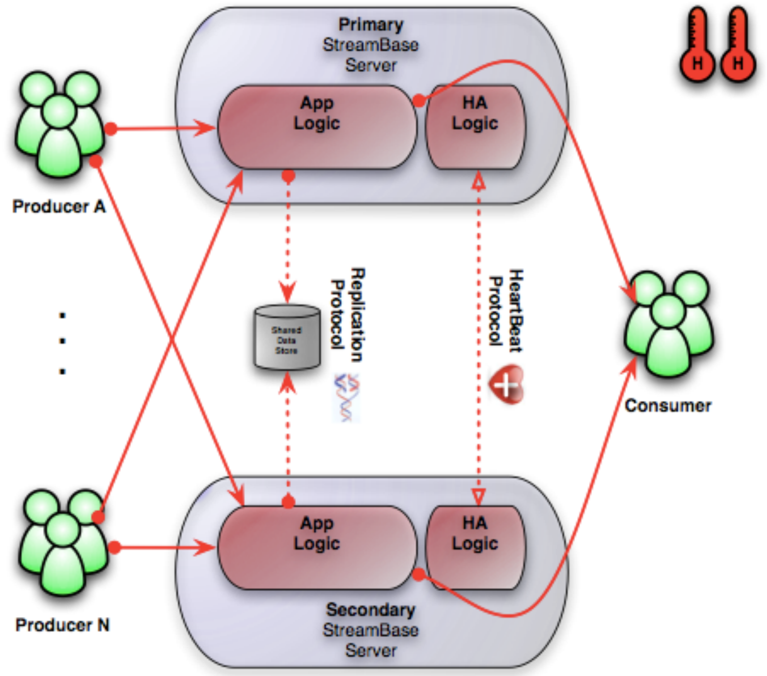


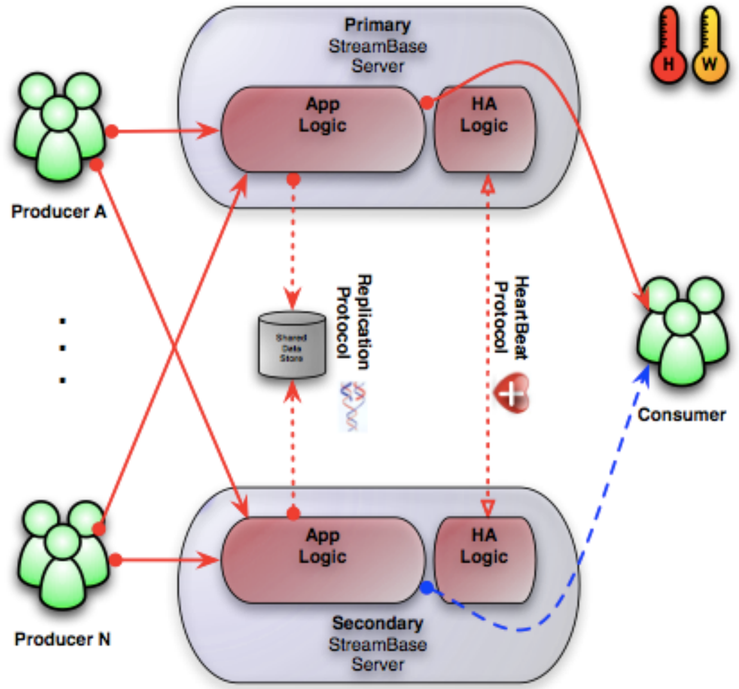
A day in the life.. Reliability



A day in the life.. Reliability

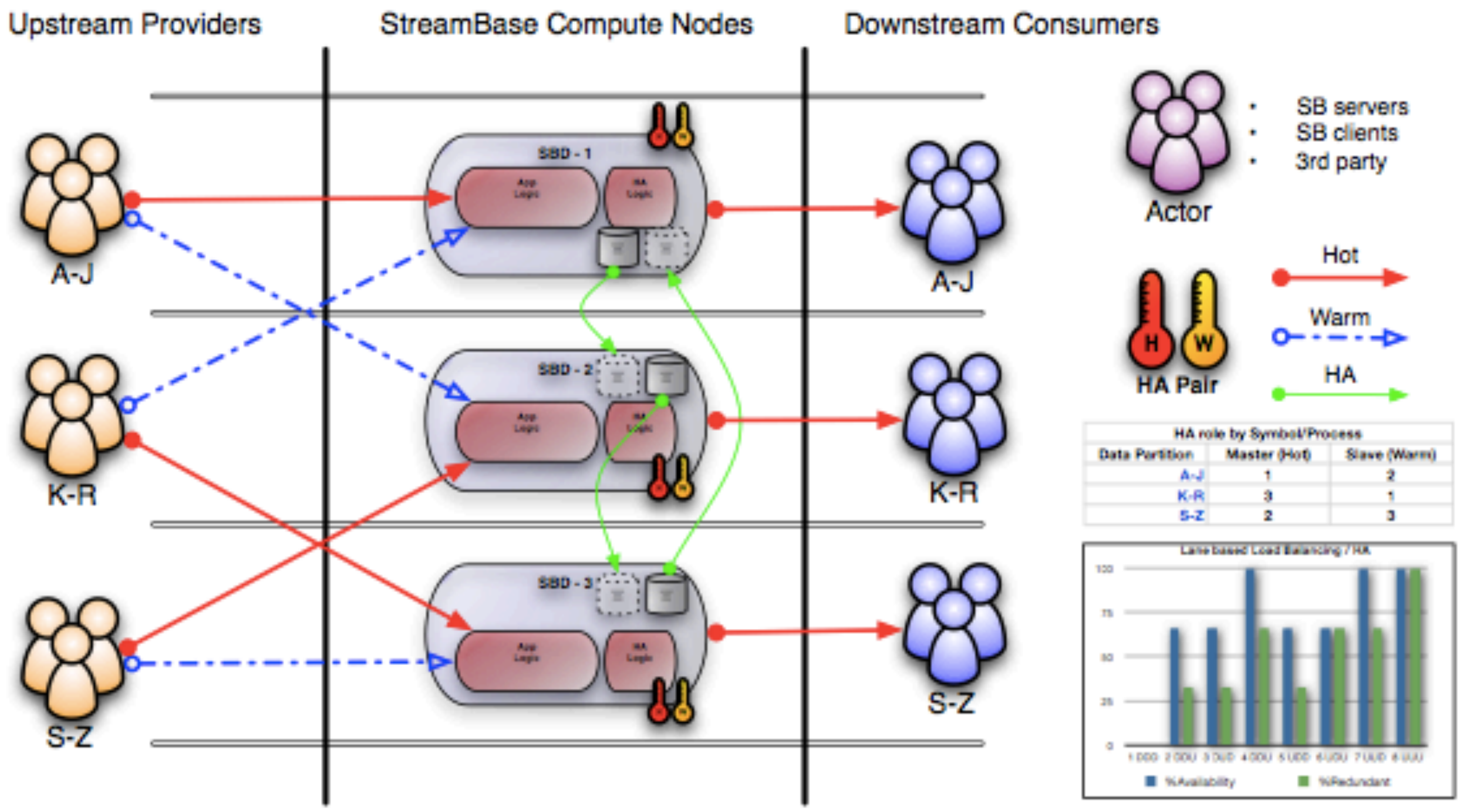
- **Hot/Hot, Hot/Warm, Hot/Cold, None?!**





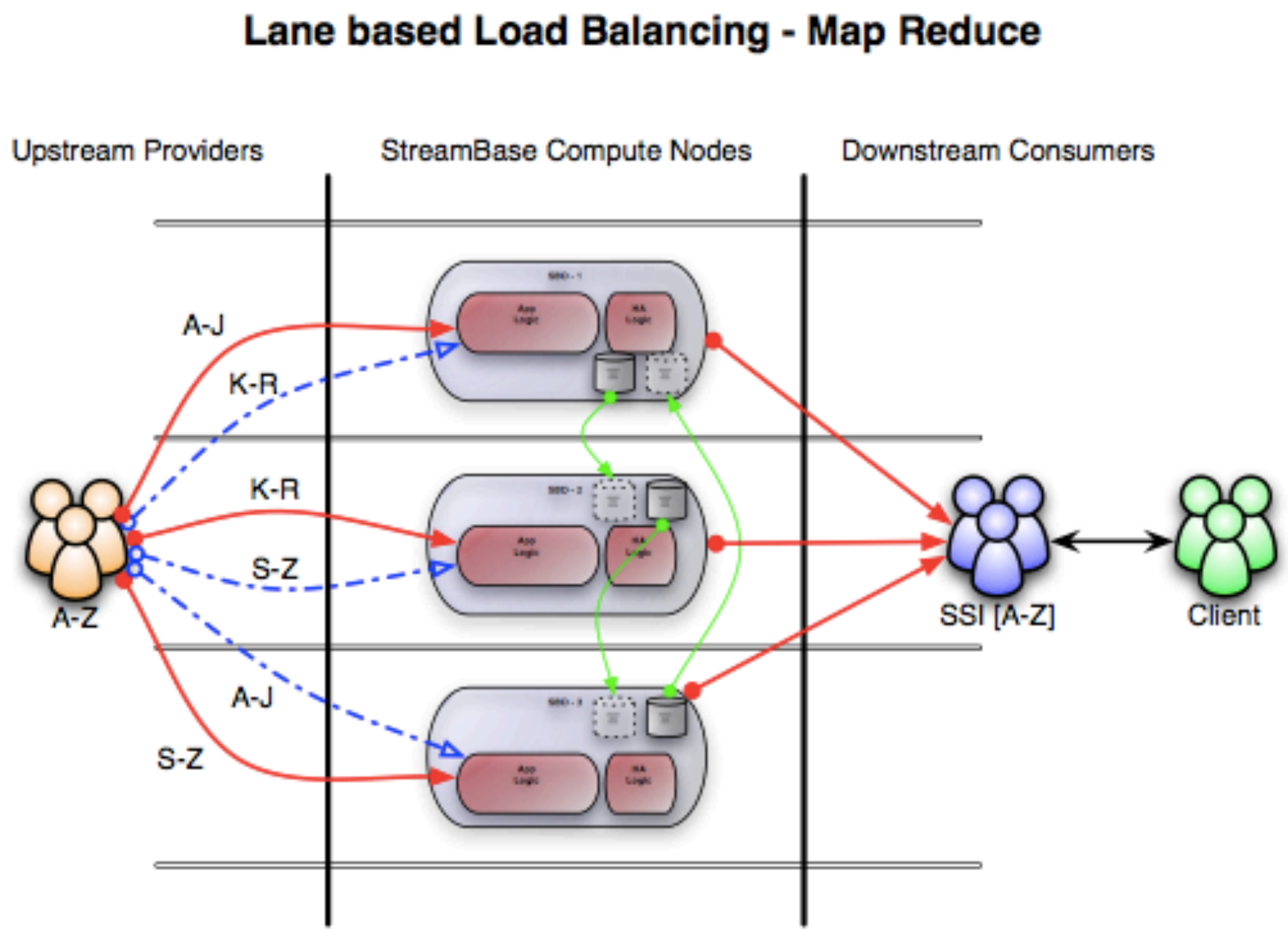
A day in the life.. Reliability

Lane based Load Balancing Overview



HA role by Symbol/Process		
Data Partition	Master (Hot)	Slave (Warm)
A-J	1	2
K-R	3	1
S-Z	2	3

A day in the life.. Reliable CSMR



Agenda

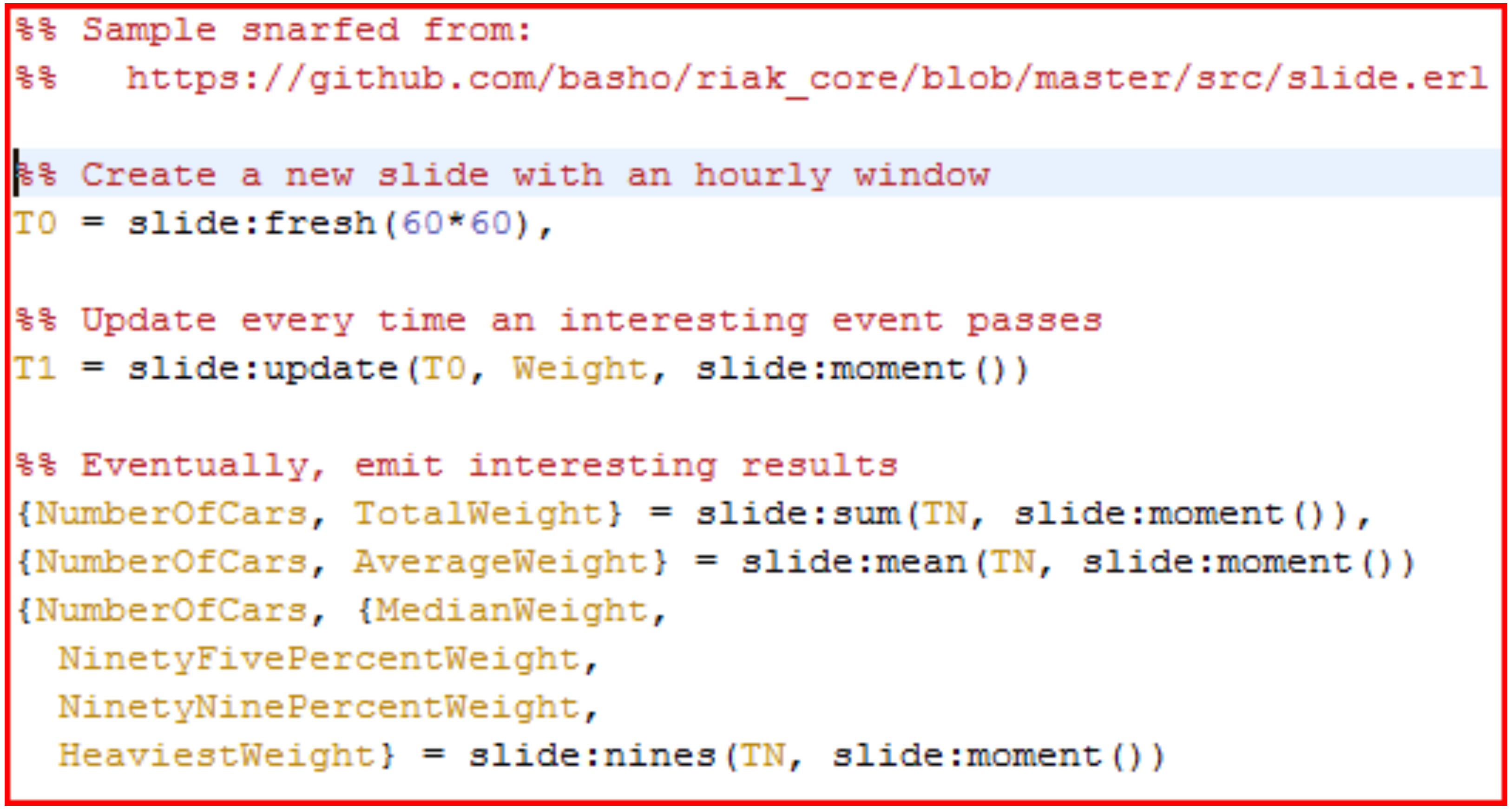
- **~~What is 'Complex Event Processing'~~**
  - ~~Specifically flow oriented event processing (there are others)~~
  - ~~Streams & Operators. Windowing, Branching, Combining, Extending~~
- **~~A day in the life of a flow programmer~~**
  - ~~Relativity - Data parallelism, concurrency, latency & throughput~~
  - ~~Continuity - Continuous Streaming Map Reduce~~
  - ~~Reliability – High availability, the low latency way~~
- **Integration. Erlang – the ecosystem.**
  - Calling Erlang from StreamBase – Simple & windowed functions
  - Client/Server – Pushing events to/from StreamBase
  - RabbitMQ - Messaging
- **Theft. Erlang – the inspiration. Paxos, in StreamBase**

Operators – Hi Erlang. Hello StreamBase

- **Aggregate – A window of moving data**
  - Time based – eg: the last 10 seconds, .001 seconds, day
  - Field based – eg: historical time (timestamp, …)
  - Tuple based – eg: the last 1000 tuples
  - Predicate based – expressions determine when to open, close and/or emit interesting results from a window
- **Multi-dimensional**
  - Give me the last seconds worth of events or the last 10000, whichever happens first
- **Grouping**
  - Partition by Symbol implies a window per Symbol 'concurrently – on the same thread'

Windows in the Wild - #1 Riak

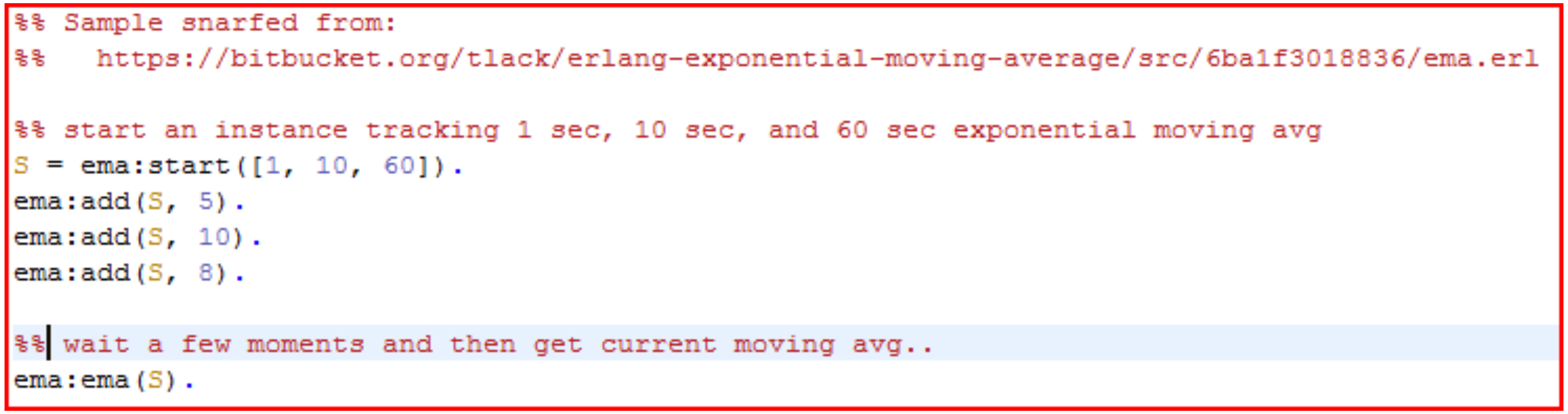
- **Riak Core:** https://github.com/basho/riak_core
  - Mixes the ‘when’ window dimension (time) with the what ‘aggregation’



```
%% Sample snarfed from:
%%   https://github.com/basho/riak_core/blob/master/src/slide.erl

%% Create a new slide with an hourly window
T0 = slide:fresh(60*60),

%% Update every time an interesting event passes
T1 = slide:update(T0, Weight, slide:moment())

%% Eventually, emit interesting results
{NumberOfCars, TotalWeight} = slide:sum(TN, slide:moment()),
{NumberOfCars, AverageWeight} = slide:mean(TN, slide:moment())
{NumberOfCars, {MedianWeight,
  NinetyFivePercentWeight,
  NinetyNinePercentWeight,
  HeaviestWeight} = slide:nines(TN, slide:moment())
```

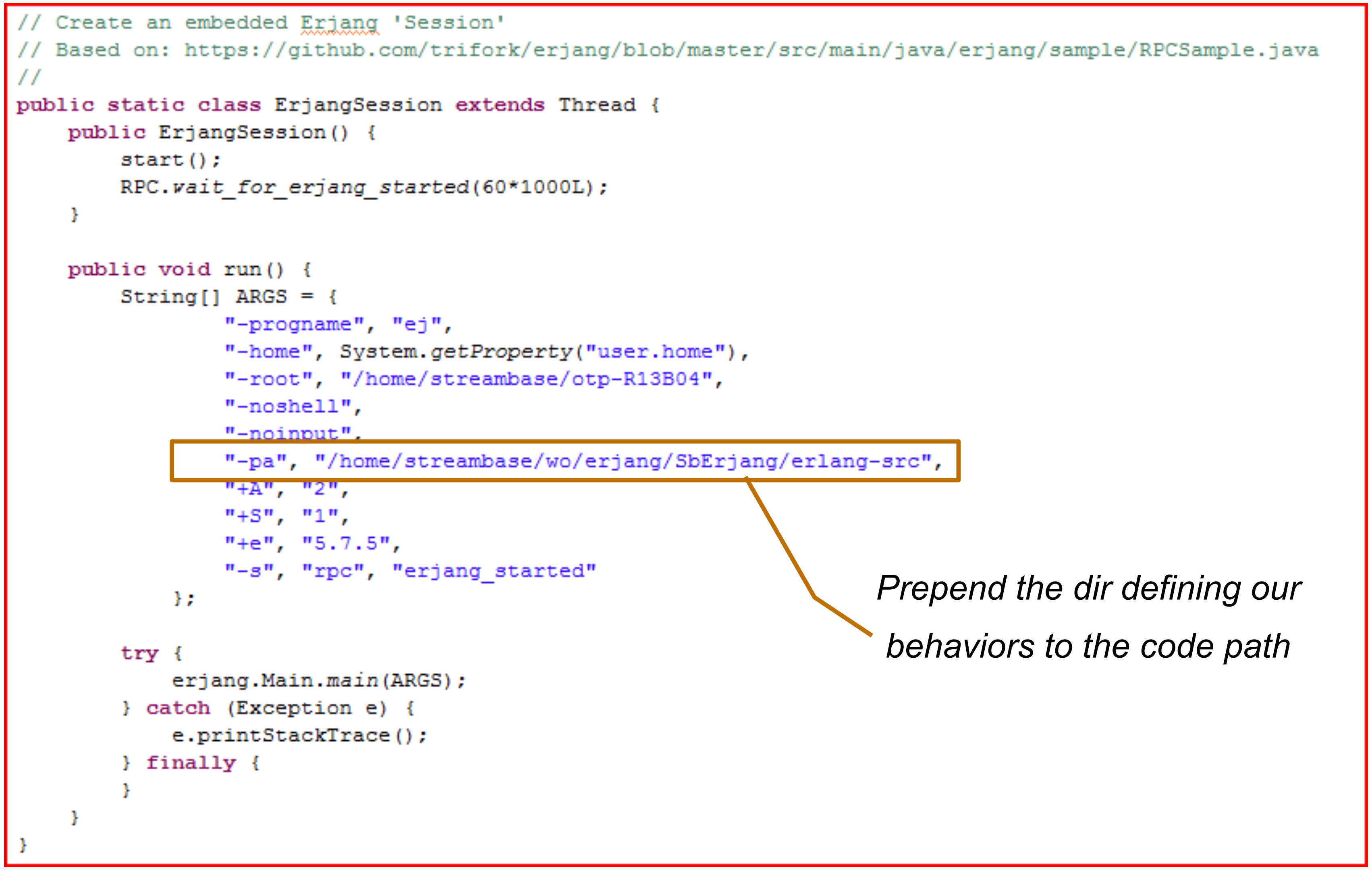
Windows in the Wild - #2 tlack on BitBucket

- **Thomas Lackner - tlack - EMA:**
  - https://bitbucket.org/tlack/erlang-exponential-moving-average/overview
  - Mixes the ‘when’ window dimension (time) with the what ‘aggregation’ and number of occurrences ‘how many’



```
%% Sample snarfed from:
%%   https://bitbucket.org/tlack/erlang-exponential-moving-average/src/6ba1f3018836/ema.erl

%% start an instance tracking 1 sec, 10 sec, and 60 sec exponential moving avg
S = ema:start([1, 10, 60]).
ema:add(S, 5).
ema:add(S, 10).
ema:add(S, 8).

%% wait a few moments and then get current moving avg..
ema:ema(S).
```

Embedding Erlang (Erjang)

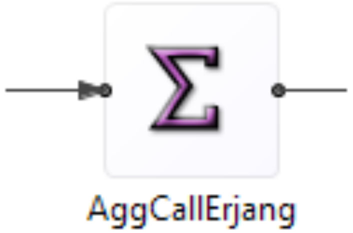


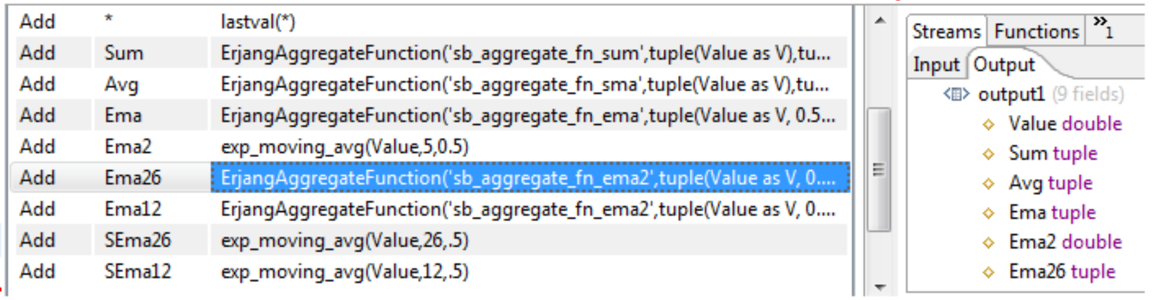
```java
// Create an embedded Erjang 'Session'
// Based on: https://github.com/trifork/erjang/blob/master/src/main/java/erjang/sample/RPCSample.java
//
public static class ErjangSession extends Thread {
    public ErjangSession() {
        start();
        RPC.wait_for_erjang_started(60*1000L);
    }

    public void run() {
        String[] ARGS = {
                "-progname", "ej",
                "-home", System.getProperty("user.home"),
                "-root", "/home/streambase/otp-R13B04",
                "-noshell",
                "-noinput",
                "-pa", "/home/streambase/wo/erjang/SbErjang/erlang-src",
                "+A", "2",
                "+S", "1",
                "+e", "5.7.5",
                "-s", "rpc", "erjang_started"
            };

        try {
            erjang.Main.main(ARGS);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
        }
    }
}
```

Prepend the dir defining our behaviors to the code path

Define Behavioral Contracts

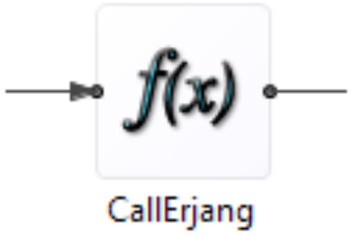
```
%% Contract between StreamBase and erlang (erjang) embedded
%% Enables StreamBase aggregate plugin/extension functions to be written in erlang
%%

-module (sb_aggregate_fn).
-export([behaviour_info/1]).

behaviour_info(callbacks) ->
  [
    {init,1},
    {accumulate,2},
    {emit,1}
  ];

behaviour_info(_Other) ->
    undefined.
```

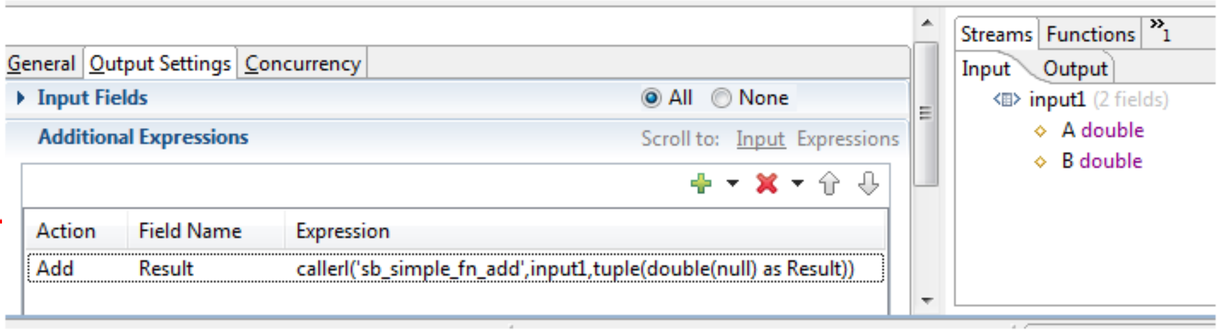




Add	*	lastval(*)
Add	Sum	ErjangAggregateFunction('sb_aggregate_fn_sum',tuple(Value as V),tu...
Add	Avg	ErjangAggregateFunction('sb_aggregate_fn_sma',tuple(Value as V),tu...
Add	Ema	ErjangAggregateFunction('sb_aggregate_fn_ema',tuple(Value as V, 0.5...
Add	Ema2	exp_moving_avg(Value,5,0.5)
Add	Ema26	ErjangAggregateFunction('sb_aggregate_fn_ema2',tuple(Value as V, 0....
Add	Ema12	ErjangAggregateFunction('sb_aggregate_fn_ema2',tuple(Value as V, 0....
Add	SEma26	exp_moving_avg(Value,26,.5)
Add	SEma12	exp_moving_avg(Value,12,.5)

```
%% Contract between StreamBase and erlang (erjang) embedded
%% Enables StreamBase simple plugin/extension functions to be written in erlang
%%

-module (sb_simple_fn).

-export([behaviour_info/1]).

behaviour_info(callbacks) ->
  [{callerl,1}];

behaviour_info(_Other) ->
    undefined.
```

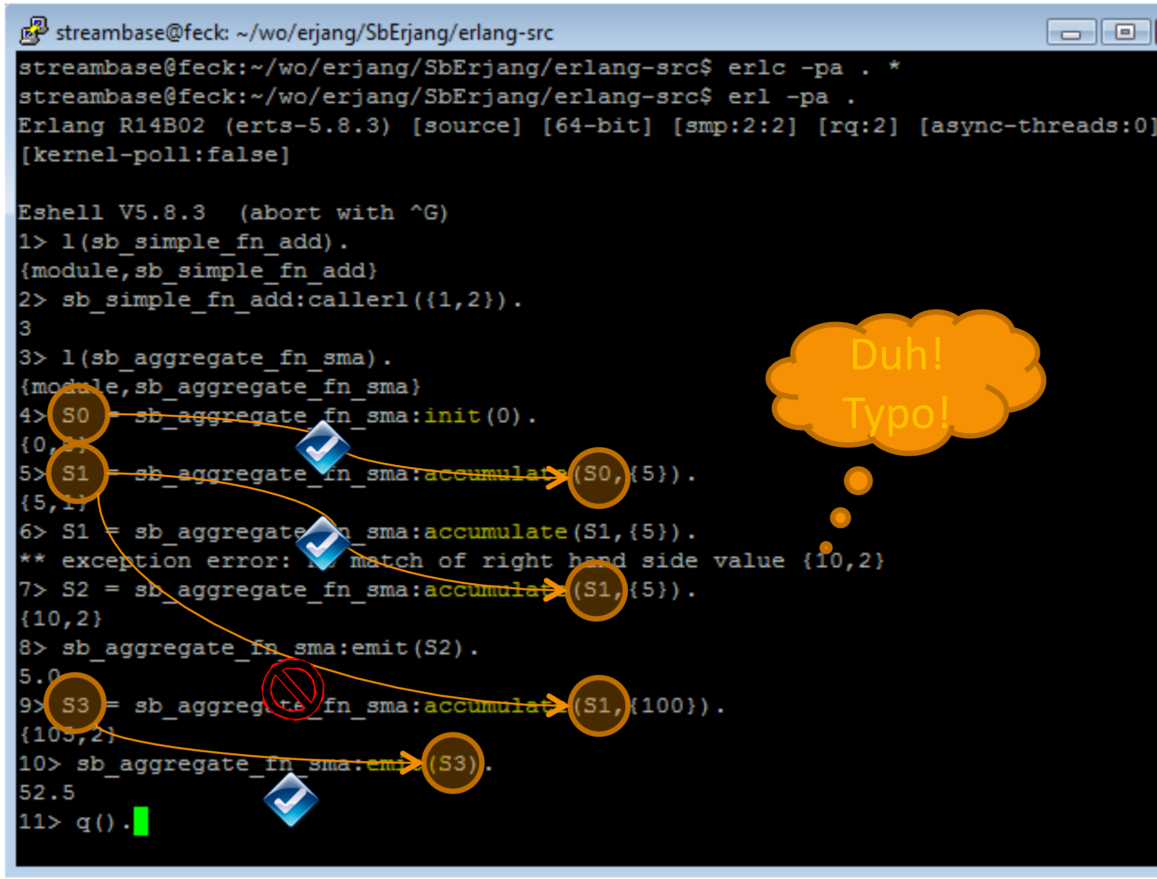




Action	Field Name	Expression
Add	Result	callerl('sb_simple_fn_add',input1,tuple(double(null) as Result))

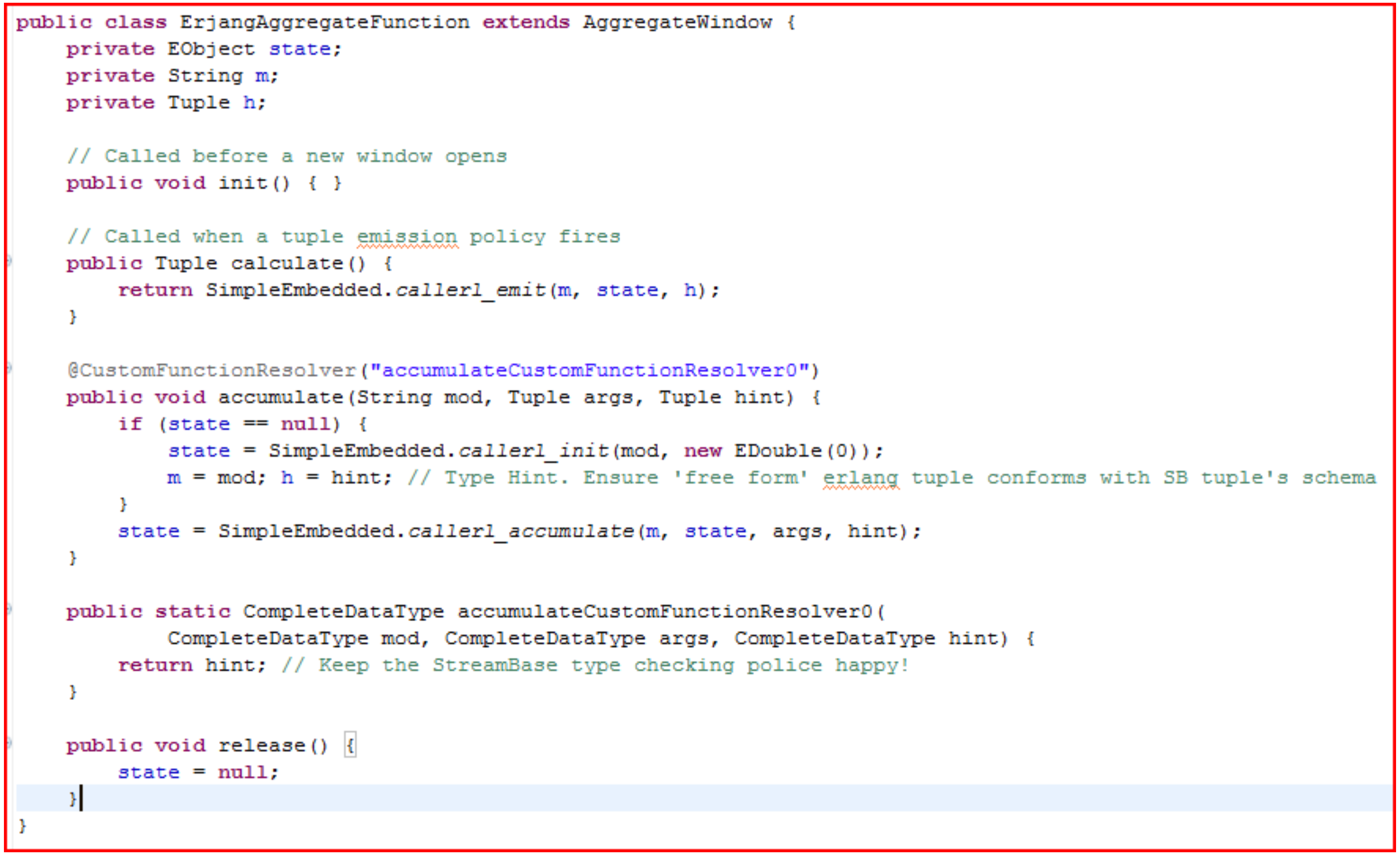
Implement & Test in Erlang (or Erjang!)

```
%% Sample: Exponential Moving Average (EMA)

-module (sb_aggregate_fn_ema2).
-behaviour(sb_aggregate_fn).
-export([init/1,accumulate/2,emit/1]).

init(_State) -> {[], [], 100.0}.

accumulate(State,{A,Weight,Capacity}) ->
     {Values,Weights,W} = State,
     Exp = W * (1.0 - Weight),
     { bounded_list:append(Values,A,Capacity),
            bounded_list:append(Weights,W,Capacity),
            Exp }.

emit(State) ->
     {Values,Weights, _W} = State,
     lists:sum([ V * VW || {V, VW} <- lists:zip(lists:reverse(Values),Weights)])/lists:sum(Weights).
```



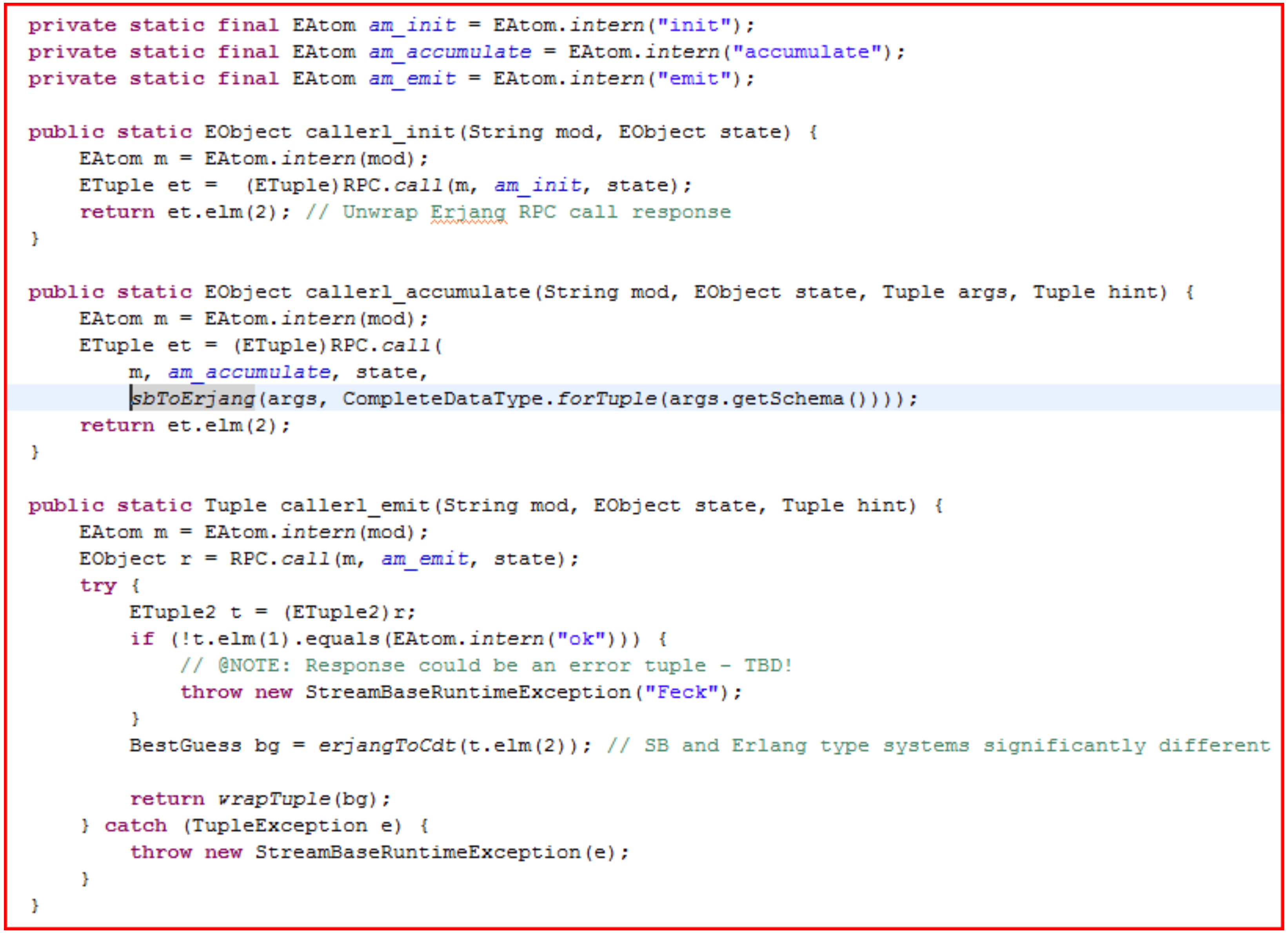
```
streambase@feck:~/wo/erjang/SbErjang/erlang-src$ erlc -pa . *
streambase@feck:~/wo/erjang/SbErjang/erlang-src$ erl -pa .
Erlang R14B02 (erts-5.8.3) [source] [64-bit] [smp:2:2] [rq:2] [async-threads:0]
[kernel-poll:false]

Eshell V5.8.3  (abort with ^G)
1> l(sb_simple_fn_add).
{module,sb_simple_fn_add}
2> sb_simple_fn_add:callerl({1,2}).
3
3> l(sb_aggregate_fn_sma).
{module,sb_aggregate_fn_sma}
4> S0 = sb_aggregate_fn_sma:init(0).
{0,[illegible]}
5> S1 = sb_aggregate_fn_sma:accumulate(S0,{5}).
{5,1}
6> S1 = sb_aggregate_fn_sma:accumulate(S1,{5}).
** exception error: no match of right hand side value {10,2}
7> S2 = sb_aggregate_fn_sma:accumulate(S1,{5}).
{10,2}
8> sb_aggregate_fn_sma:emit(S2).
5.0
9> S3 = sb_aggregate_fn_sma:accumulate(S1,{100}).
{105,2}
10> sb_aggregate_fn_sma:emit(S3).
52.5
11> q().
```

Expose to StreamBase [Call by Behaviour/Mod] #1



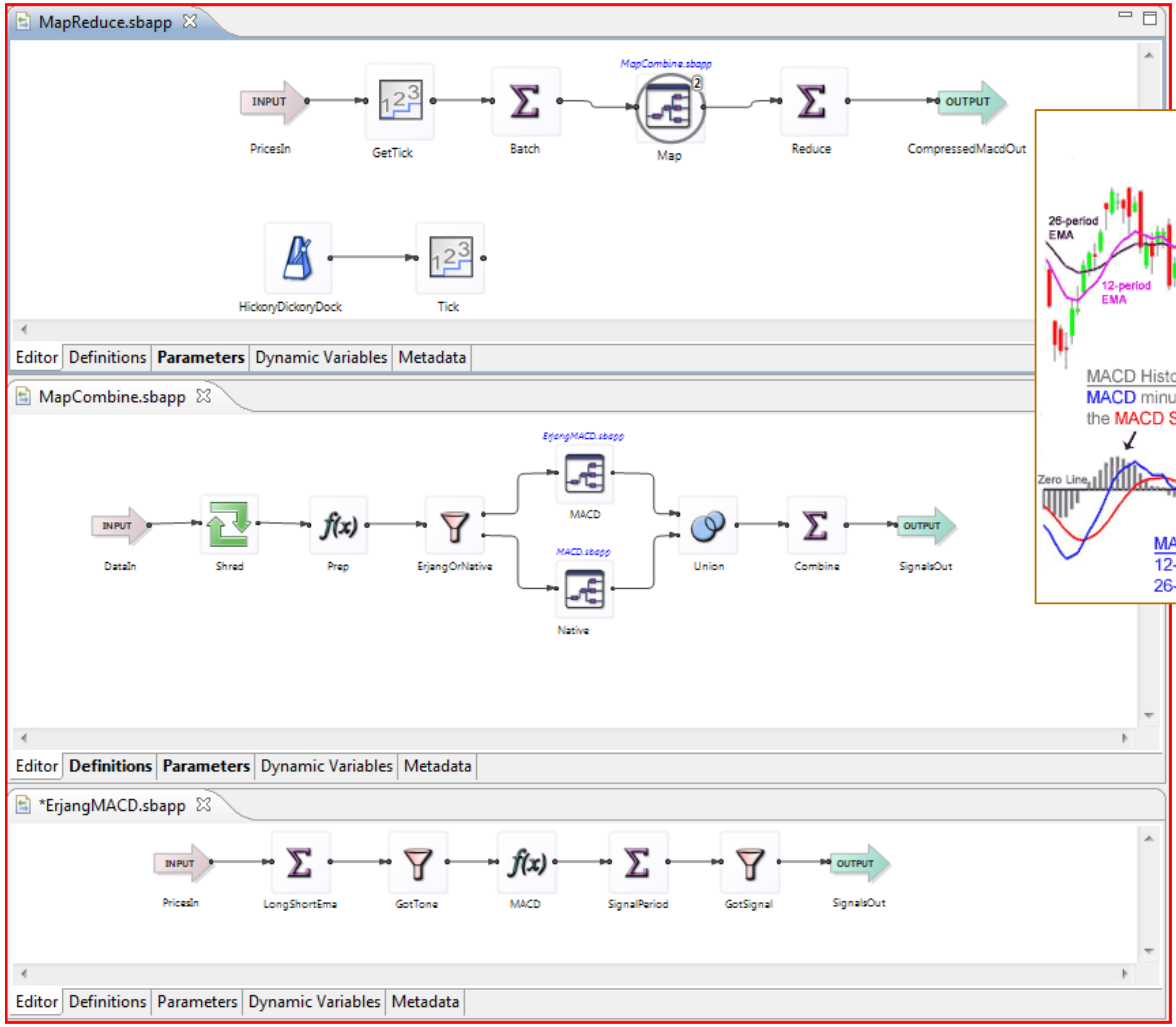
```
public class ErjangAggregateFunction extends AggregateWindow {
    private EObject state;
    private String m;
    private Tuple h;

    // Called before a new window opens
    public void init() { }

    // Called when a tuple emission policy fires
    public Tuple calculate() {
        return SimpleEmbedded.callerl_emit(m, state, h);
    }

    @CustomFunctionResolver("accumulateCustomFunctionResolver0")
    public void accumulate(String mod, Tuple args, Tuple hint) {
        if (state == null) {
            state = SimpleEmbedded.callerl_init(mod, new EDouble(0));
            m = mod; h = hint; // Type Hint. Ensure 'free form' erlang tuple conforms with SB tuple's schema
        }
        state = SimpleEmbedded.callerl_accumulate(m, state, args, hint);
    }

    public static CompleteDataType accumulateCustomFunctionResolver0(
            CompleteDataType mod, CompleteDataType args, CompleteDataType hint) {
        return hint; // Keep the StreamBase type checking police happy!
    }

    public void release() {
        state = null;
    }
}
```

Expose To StreamBase [Call by Behaviour/Mod] #2

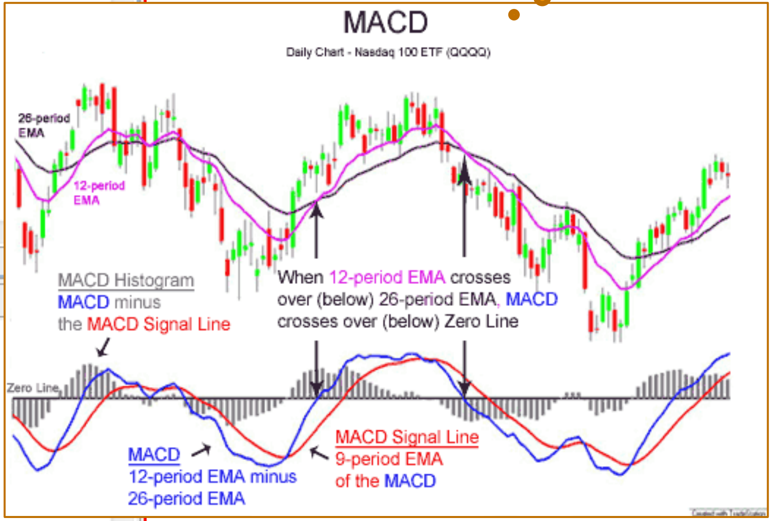


```
private static final EAtom am_init = EAtom.intern("init");
private static final EAtom am_accumulate = EAtom.intern("accumulate");
private static final EAtom am_emit = EAtom.intern("emit");

public static EObject callerl_init(String mod, EObject state) {
    EAtom m = EAtom.intern(mod);
    ETuple et =  (ETuple)RPC.call(m, am_init, state);
    return et.elm(2); // Unwrap Erjang RPC call response
}

public static EObject callerl_accumulate(String mod, EObject state, Tuple args, Tuple hint) {
    EAtom m = EAtom.intern(mod);
    ETuple et = (ETuple)RPC.call(
        m, am_accumulate, state,
        sbToErjang(args, CompleteDataType.forTuple(args.getSchema())));
    return et.elm(2);
}

public static Tuple callerl_emit(String mod, EObject state, Tuple hint) {
    EAtom m = EAtom.intern(mod);
    EObject r = RPC.call(m, am_emit, state);
    try {
        ETuple2 t = (ETuple2)r;
        if (!t.elm(1).equals(EAtom.intern("ok"))) {
            // @NOTE: Response could be an error tuple - TBD!
            throw new StreamBaseRuntimeException("Feck");
        }
        BestGuess bg = erjangToCdt(t.elm(2)); // SB and Erlang type systems significantly different

        return wrapTuple(bg);
    } catch (TupleException e) {
        throw new StreamBaseRuntimeException(e);
    }
}
```

Use, Deploy & Run



Wikipedia: MACD

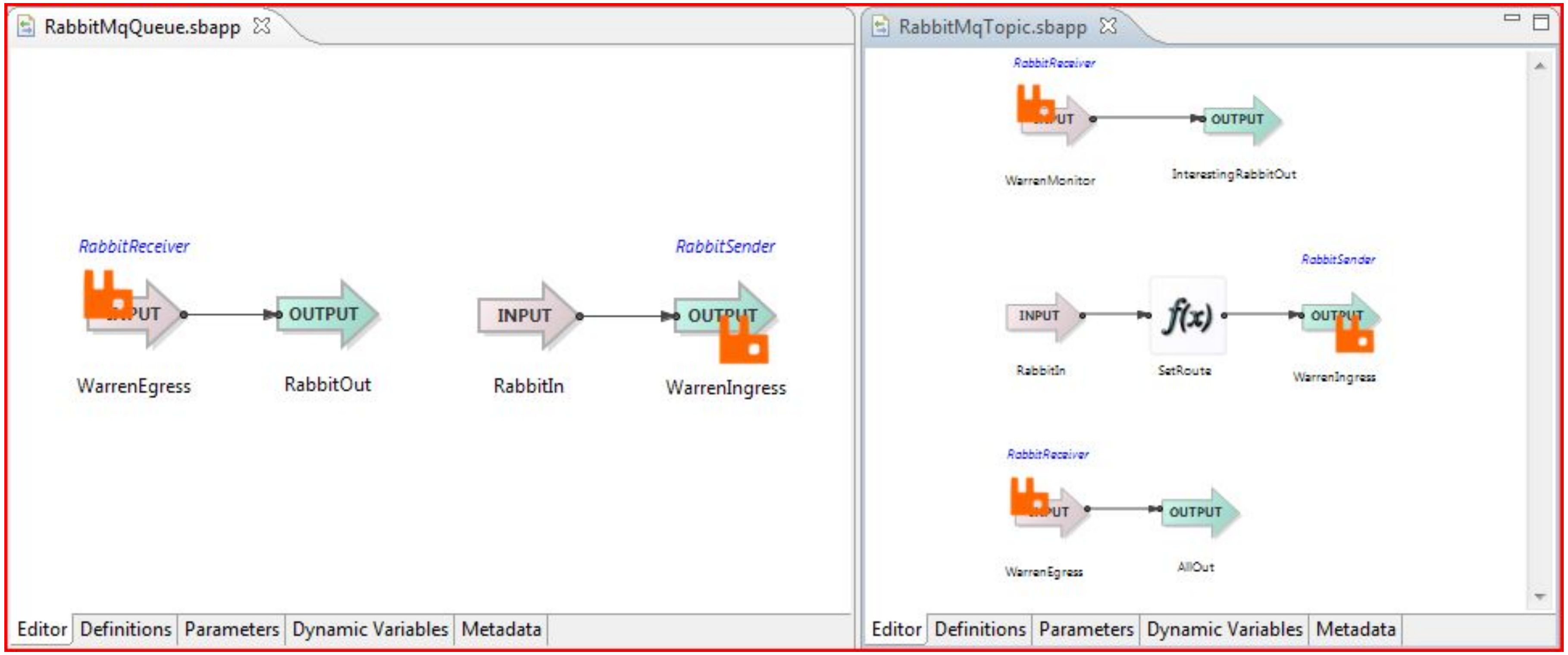


Example:

Continuous Streaming Map/Reduce with 1-second MACD compression. Linearly SMP scalable.

Just add boxes to scale!

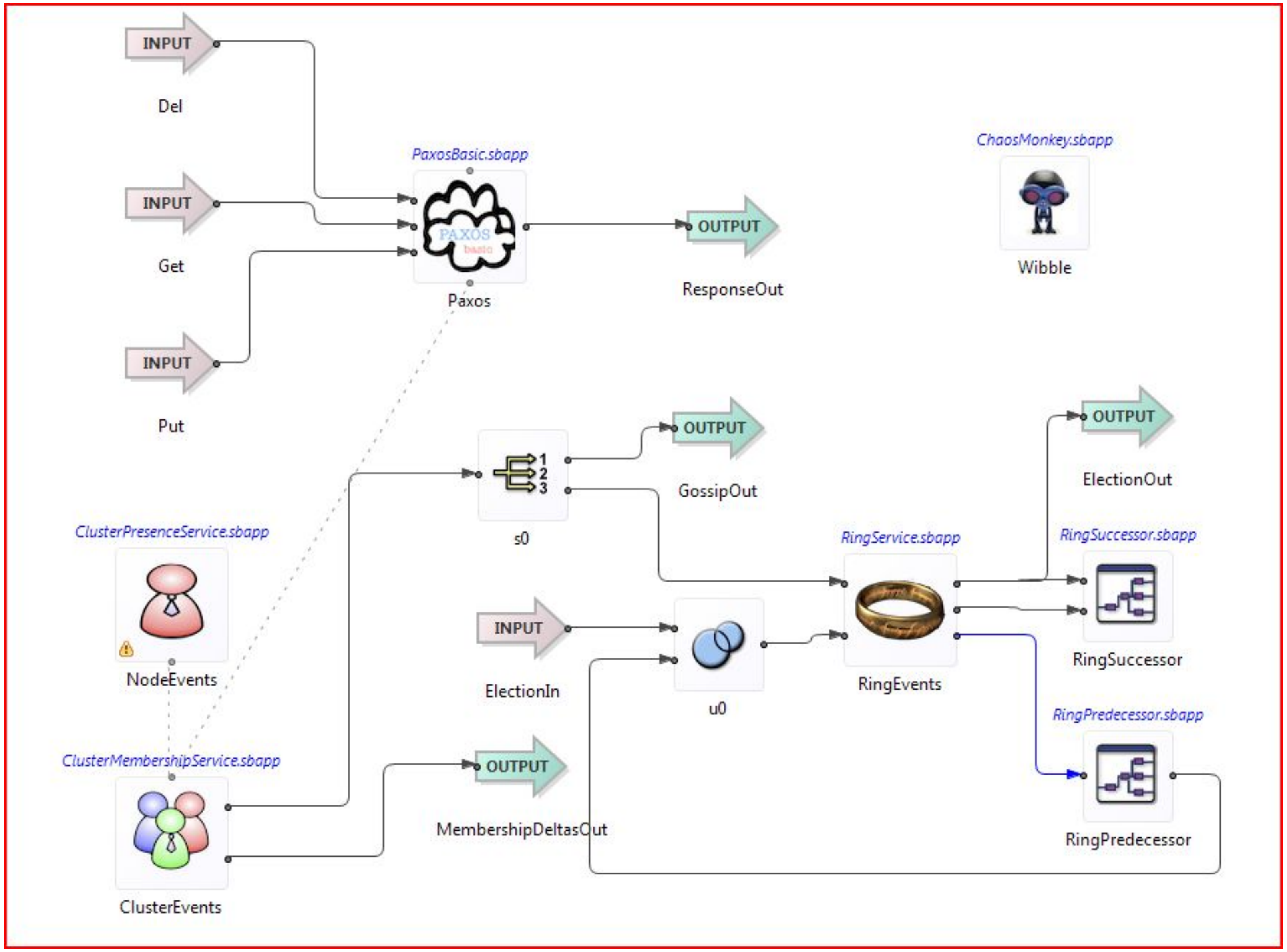
Run, Rabbit, Run, Rabbit, …



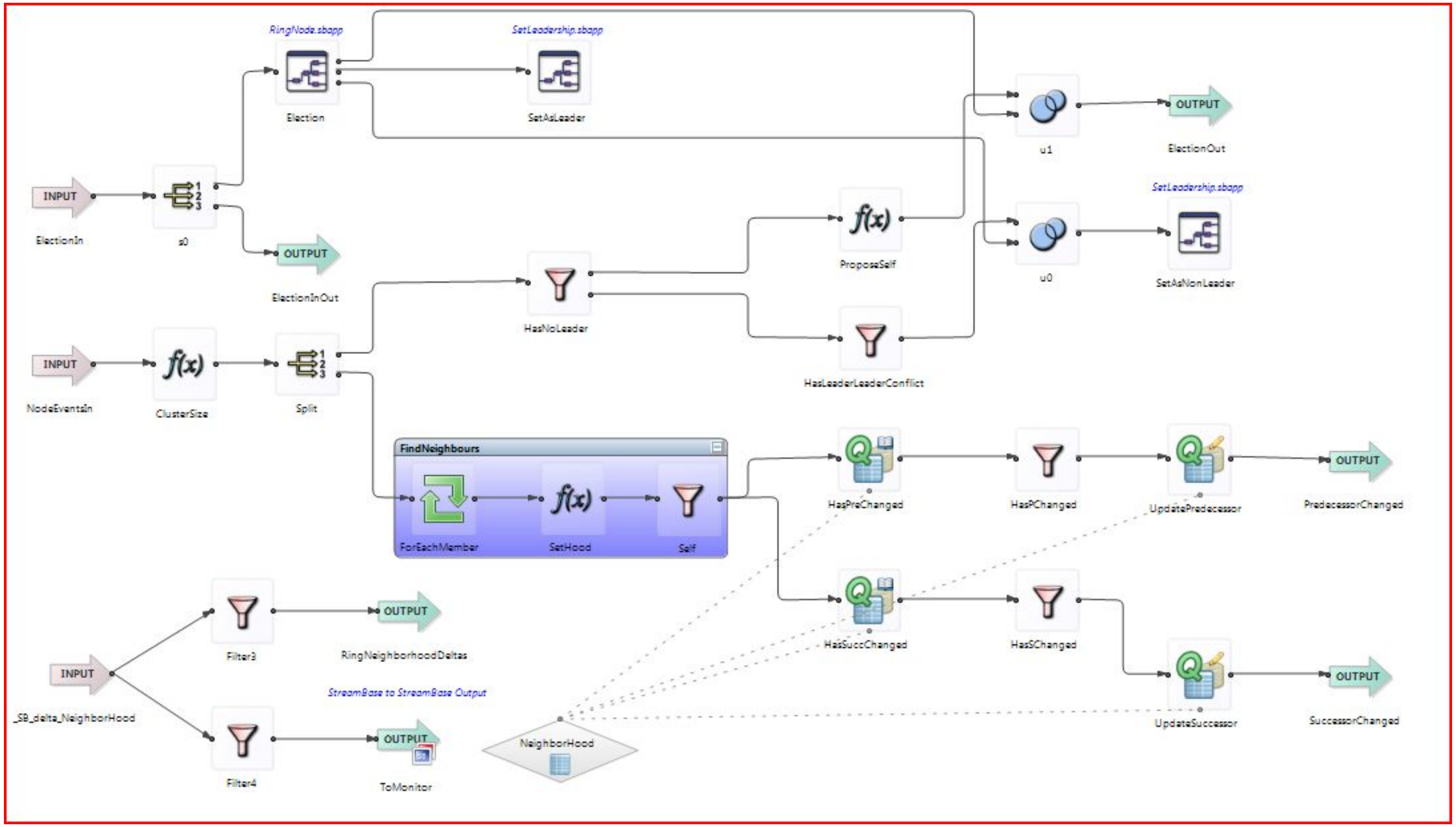
Agenda

- **~~What is 'Complex Event Processing'~~**
  - ~~Specifically flow oriented event processing (there are others)~~
  - ~~Streams & Operators. Windowing, Branching, Combining, Extending~~
- **~~A day in the life of a flow programmer~~**
  - ~~Relativity - Data parallelism, concurrency, latency & throughput~~
  - ~~Continuity - Continuous Streaming Map Reduce~~
  - ~~Reliability – High availability, the low latency way~~
- **~~Integration. Erlang – the ecosystem.~~**
  - ~~Calling Erlang from StreamBase – Simple & windowed functions~~
  - ~~Client/Server – Pushing events to/from StreamBase~~
  - ~~RabbitMQ - Messaging~~
- **Theft. Erlang – the inspiration. Paxos, in StreamBase**

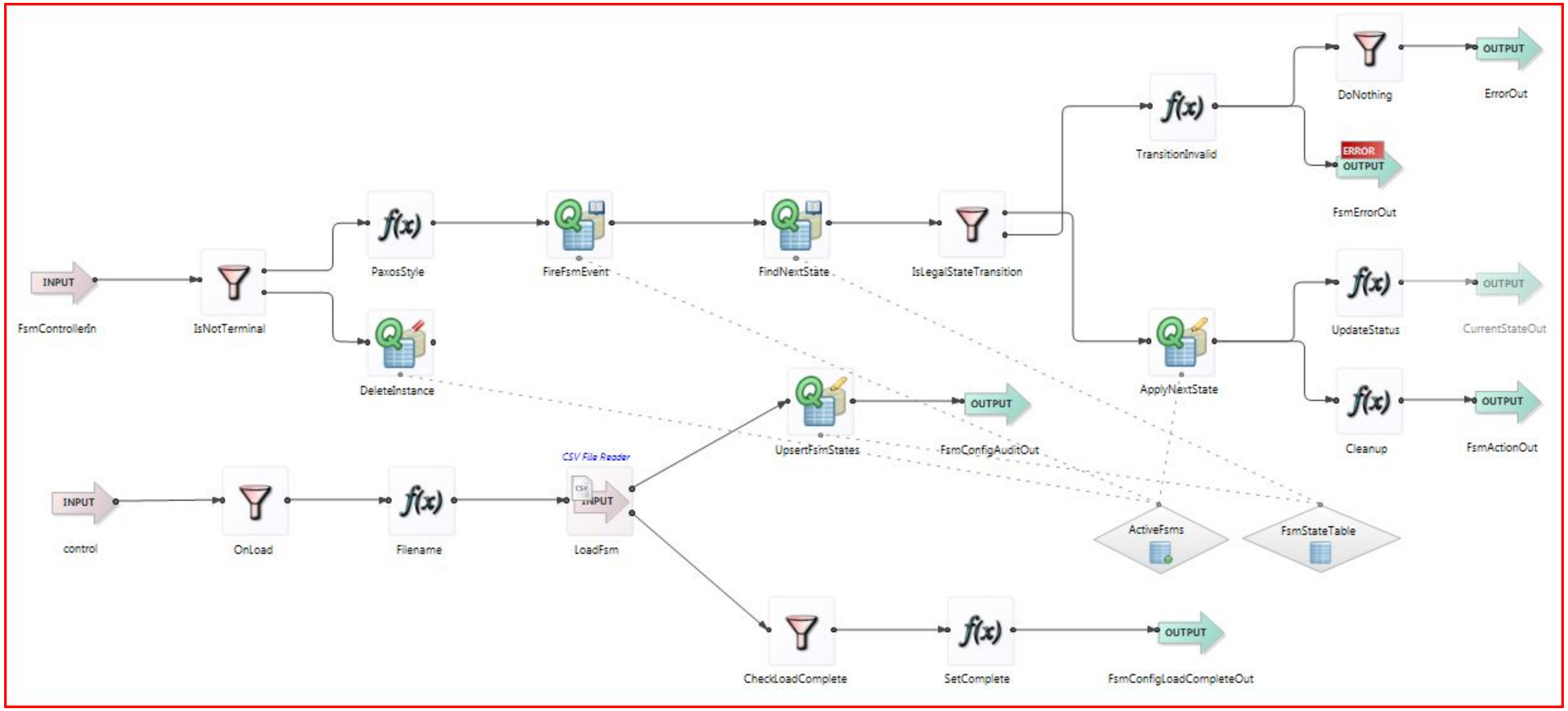
SB Paxos – Entrypoint (cc @kevsmith!)



SB Paxos – Autonomic, Self Healing Ring



SB Paxos – Generic FSM ‘Behaviour’



SB Paxos – Generic FSM ‘Behaviour’ – Paxos FSM

```
StateMap,State,Transition,NextState,Action,Description,
BasicPaxosMap,Initial,Bootstrap,Proposer,Prepare,"..."
BasicPaxosMap,Initial,PromiseOk,Accepter,PromiseOk,"..."
BasicPaxosMap,Initial,PromiseNotOk,Done,PromiseNotOk,"..."
BasicPaxosMap,Initial,Accepted,Learner,Response,"When ..."
BasicPaxosMap,Proposer,Accept,Proposer,Accept,"..."
BasicPaxosMap,Proposer,PromiseNotOk,Proposer,DoNothing,"..."
BasicPaxosMap,Proposer,Response,Done,Gone,DoStop,"..."
BasicPaxosMap,Accepter,Accept,Learner,Accept,"..."
BasicPaxosMap,Accepter,Accepted,Learner,Response,"..."
BasicPaxosMap,Learner,Accepted,Done,Response,"..."
BasicPaxosMap,Learner,Done,Gone,DoStop,"..."
```

Shameless Plugs

- **StreamBase**
  - You could build one of these yourself, or use ours…
  - Download and test out the full product http:/www.streambase.com
  - Build something and submit to the StreamBase Component Exchange http://sbx.streambase.com
  - Contact us to buy or to an OEM partner, offices London, Boston, New York
  - We're hiring
  - We're training
    - http://www.streambase.com/developers-training-events.htm
- **DEBS – Distributed Event Based Systems**
  - Academic (ACM) Conference outside NYC in July
- **EPTS – Event Processing Technology Society**
  - http://ep-ts.org industry consortium

Questions?

Acknowledgements

- **Erlang, Erlang Solutions, Erlang UG London, & Erlang Factory**
  - http://www.erlang.org/
  - http://www.erlang-solutions.com/
  - http://www.erlang-solutions.com/etc/usergroup/london
  - http://www.erlang-factory.com/
- **Erjang – The Java based Erlang Virtual Machine**
  - https://github.com/trifork/erjang/wiki/
- **erlIDE**
  - http://erlide.sourceforge.net/
- **@tibbetts – I ~~stole~~ borrowed some of his QCon slides!**
- **Download StreamBase and tell us what you think:**
  - http://www.streambase.com

Download StreamBase and More Information

http://www.streambase.com

Questions?