Real-time Framework in Erlang for Computational Advertising

Pero Subasic

AOL Advertising R&D,

Palo Alto, CA

Overview

About online advertising

Problem Outline

Background, existing systems

Infrastructure: hardware, software, data

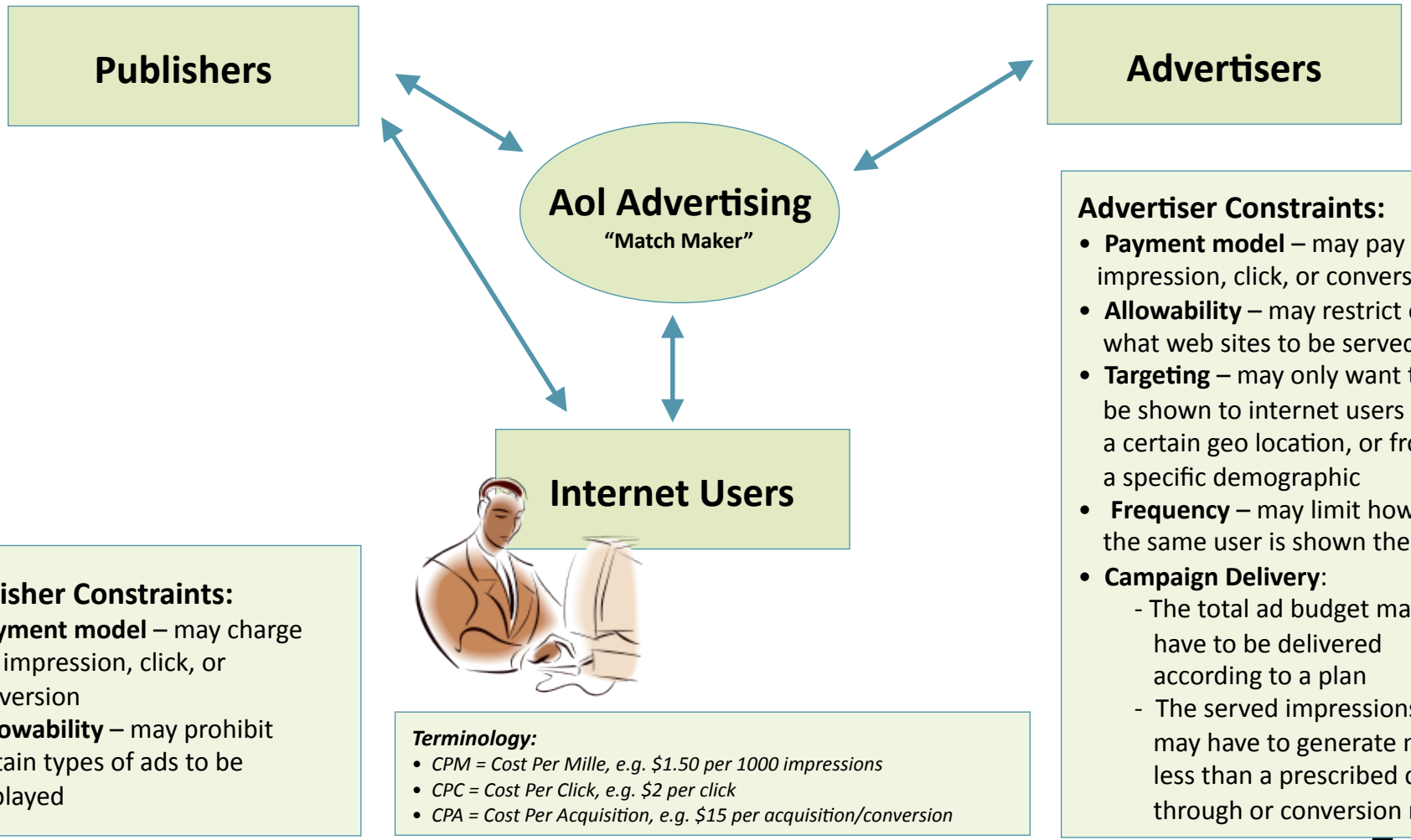
RT processing in existing architecture

Use cases: solutions, experiments and results

Reference architecture

Conclusions, Future

Online Advertising



Publisher Constraints:

- **Payment model** – may charge per impression, click, or conversion
- **Allowability** – may prohibit certain types of ads to be displayed

Terminology:

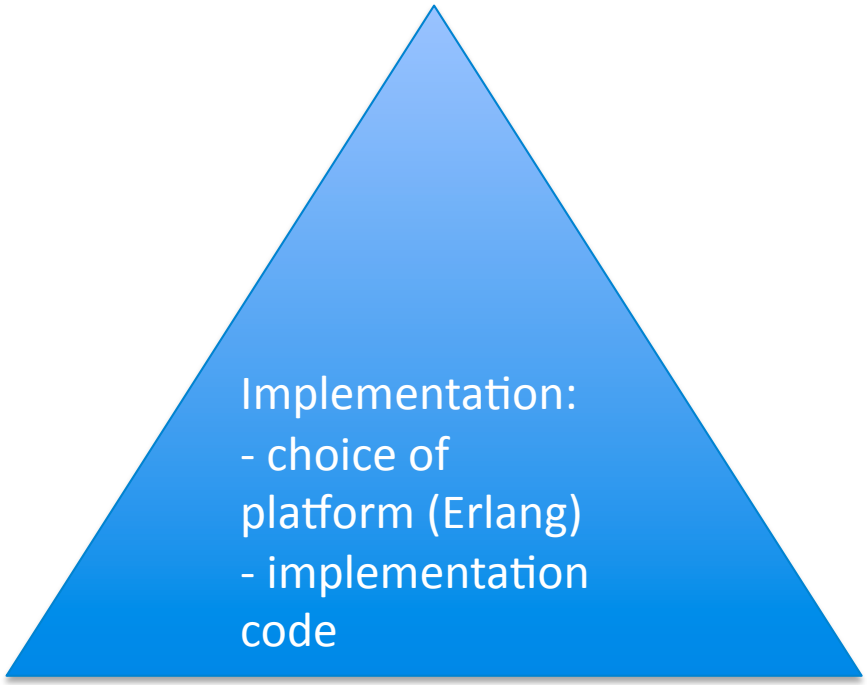
- *CPM = Cost Per Mille, e.g. $1.50 per 1000 impressions*
- *CPC = Cost Per Click, e.g. $2 per click*
- *CPA = Cost Per Acquisition, e.g. $15 per acquisition/conversion*

Advertiser Constraints:

- **Payment model** – may pay per impression, click, or conversion
- **Allowability** – may restrict on what web sites to be served
- **Targeting** – may only want to be shown to internet users in a certain geo location, or from a specific demographic
- **Frequency** – may limit how often the same user is shown the ad
- **Campaign Delivery**:
  - The total ad budget may have to be delivered according to a plan
  - The served impressions may have to generate no less than a prescribed click-through or conversion rate

Performance Factors

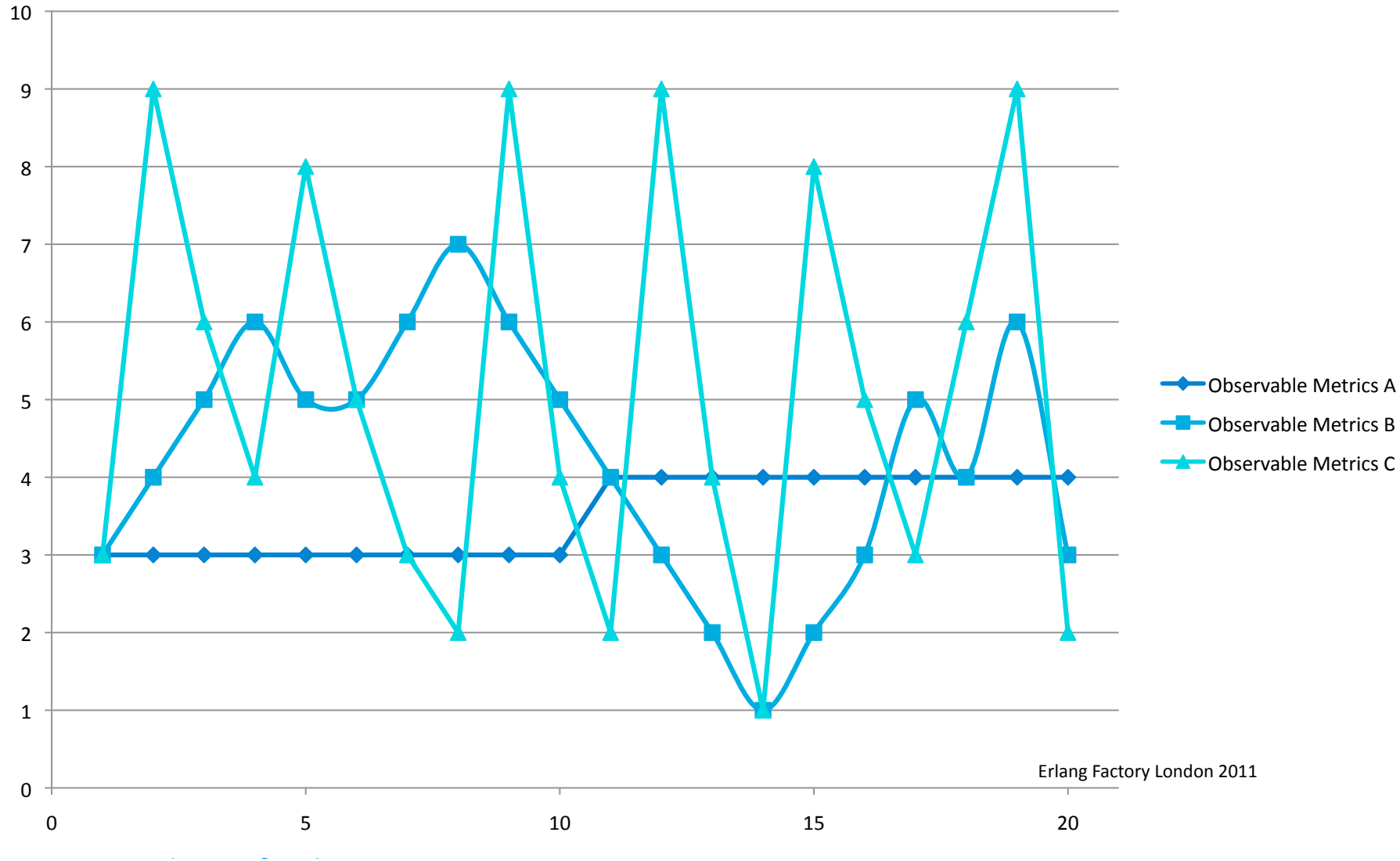
Data:
- volume,
- distribution,
- dynamics



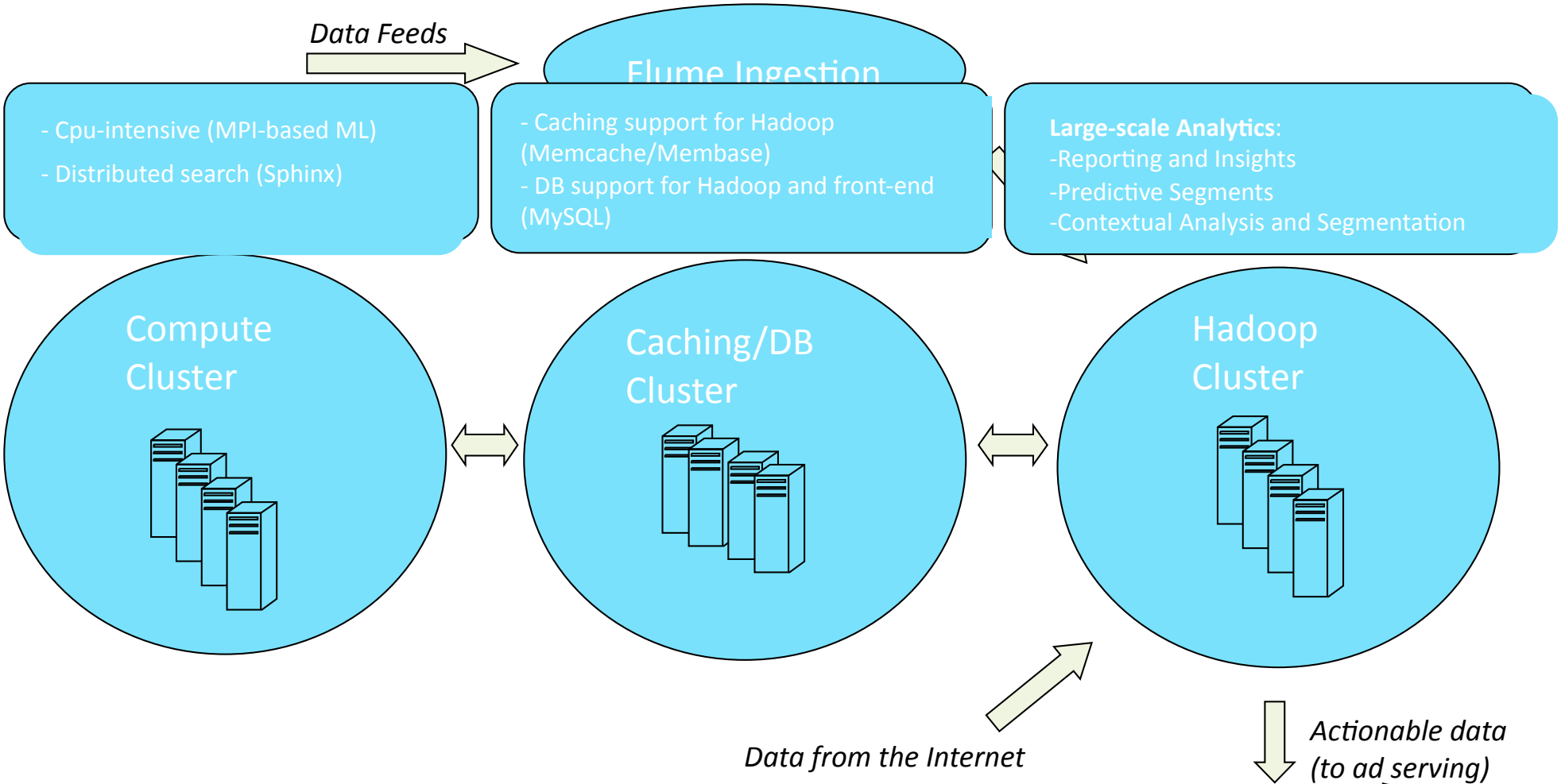
Requirements:
low-latency/RT analytics and optimization

System:
hardware/OS/network capacity and limitations

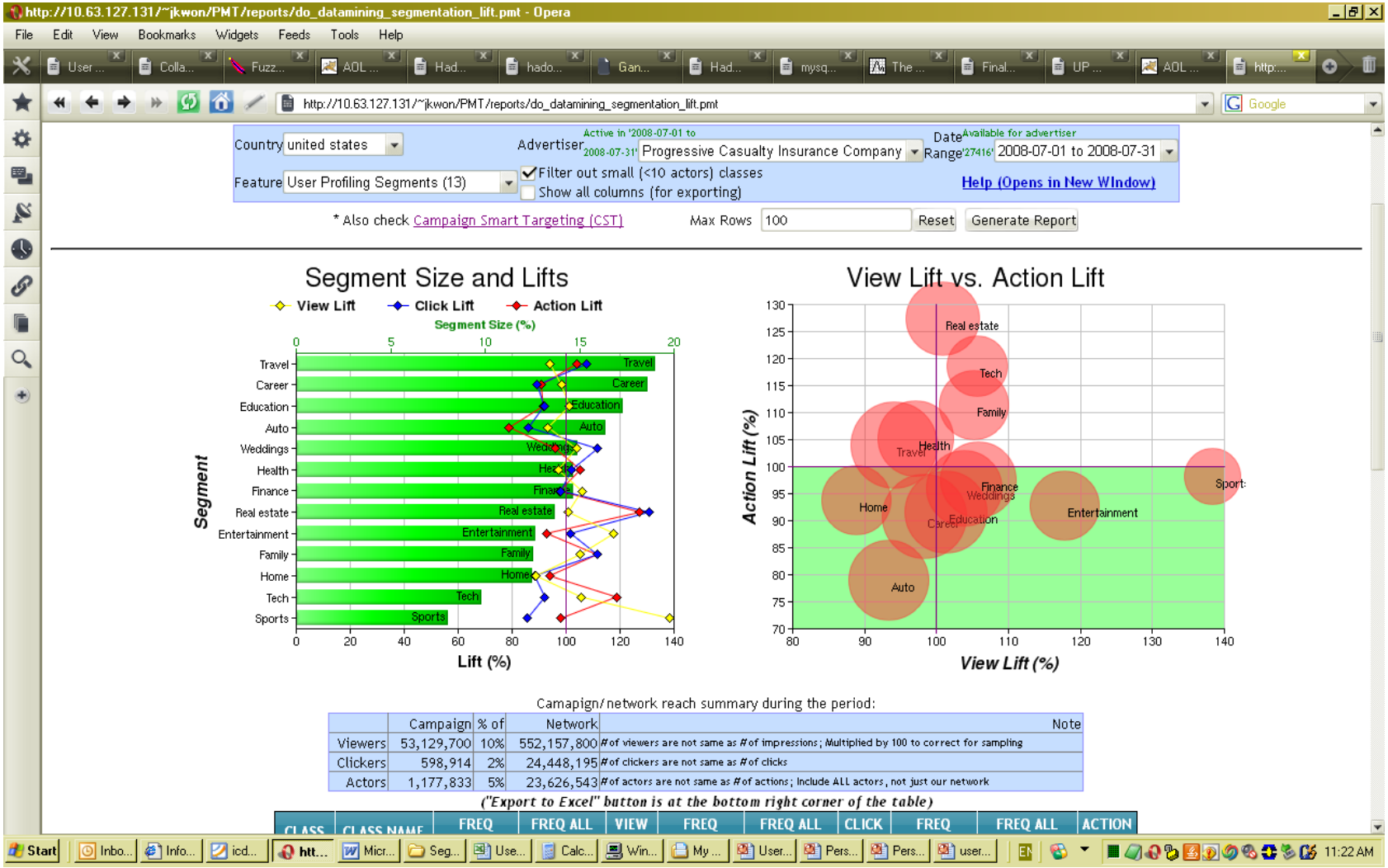
Computing in Time



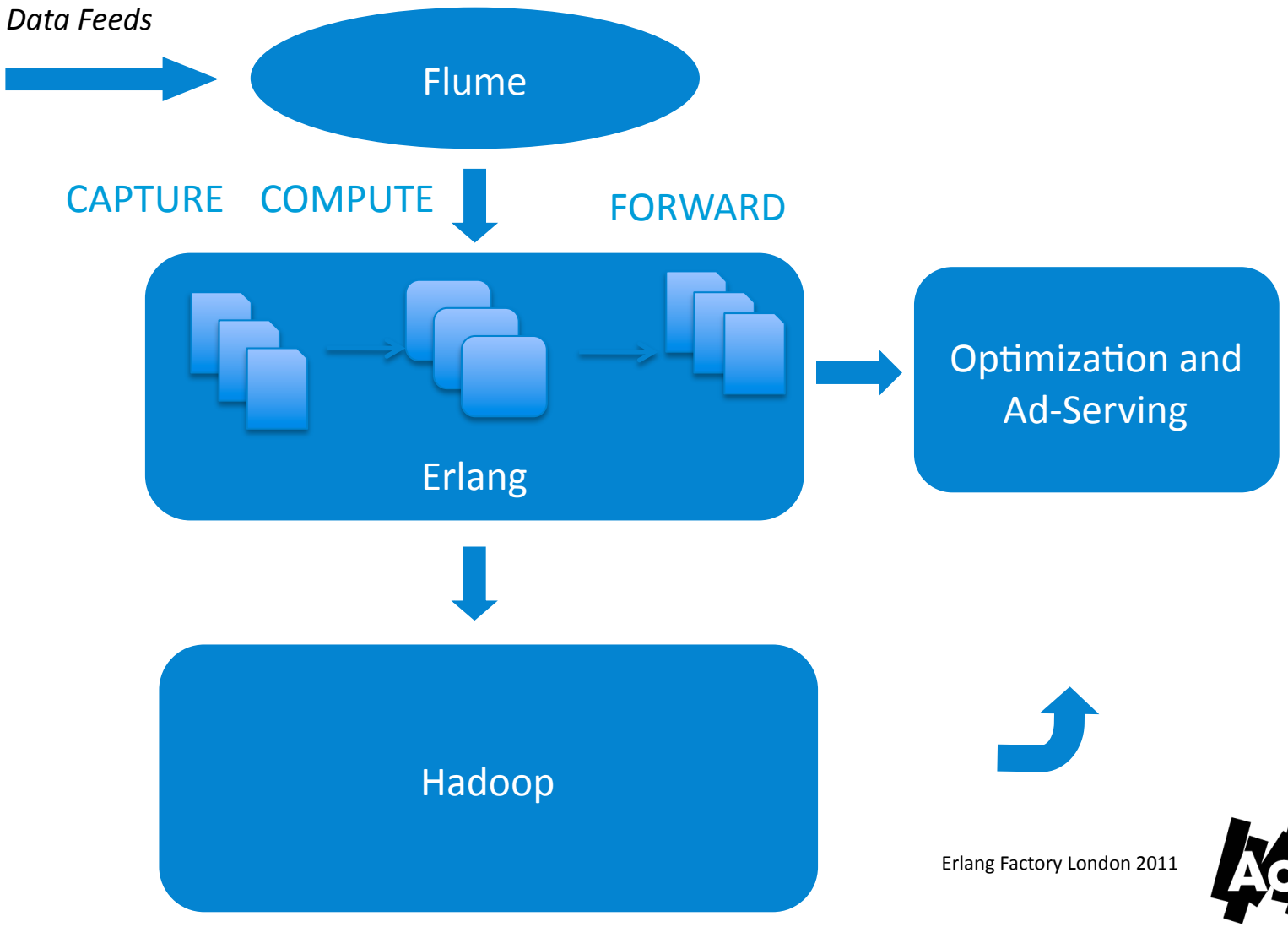
Backend Architecture and Data Flow



SLT lift example with contextual segments, 2008



Capture, Compute and Forward: Stateful Event Processing



Infrastructure: hardware

10 nodes in NTC data center (Mountain View)

16 cores (4 X 4 Quad-core AMD Opteron), 128G RAM, 800MHz/core

GigE on shared switch (125MB/sec theoretical limit; RAM to RAM)

Shared NAS storage: used for file distribution and code staging

Infrastructure: software, data

Software:

Erlang R14B02 (erts-5.8.3),

64 bit,

SMP: 16:16

Data:

Core data set: 1:1 15 minute data feed; smaller sets are generated by random uniform sampling of events

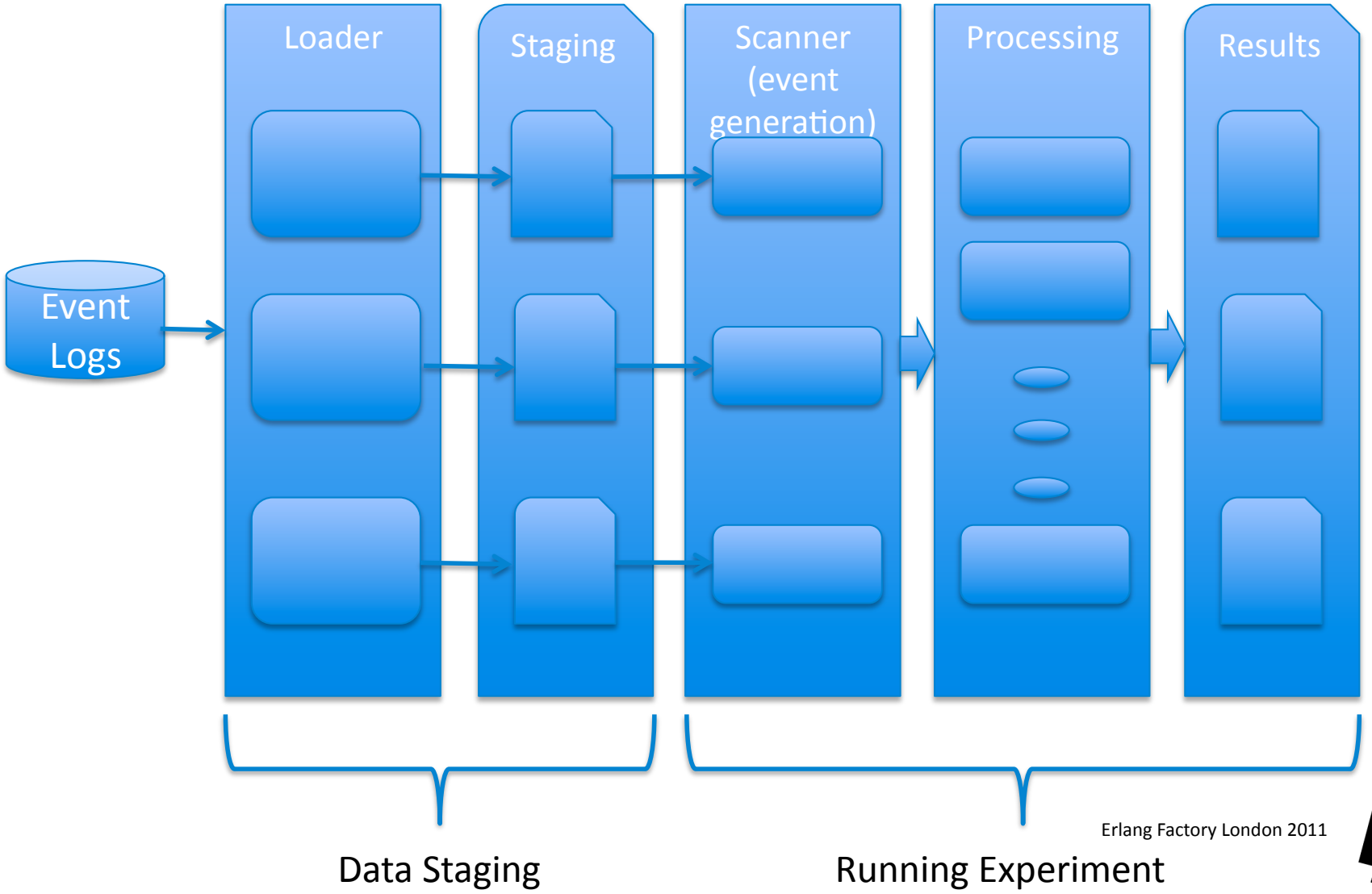
approx. 50-60 mil. events -> 5-6 bil. events per day

5-6 mil. events per node -> 3 gzipped txt files of 1:10 ad.com data

Event Record

I:1249074734|225|743227|742780|***A.B.C.D***|***KF86001248536890001***3|137|
5686=388359;3152=373261;3139=379838;3548=375138;3153=373262;3775=376196;3
774=376176;3365=374143;3776=376209;4696=378742;4282=378992;7332382035;428
8=378754;4291=382295;3150=378986;3155=378745;7333=396718;4281==396712;317
7=379942;3179=373288;3174=373283;4293=378997;4419=380826;4199=378423;152=
300251;2787=371772;1604=363638;3173=373282;3172=373281;3154=373263;4751=3
82279;4750=382271;3192=373301;5=372255;8=300291;4290=378880;7=200016;2217=
369209;10=377801;3572=380411;6158=402219,390727,391540,390901,402222,39140
4,391501,390726,390899,402220,391500,402223,390897;4472=381048;8751=404702;
8452=402320;7005=394960;8807=404946;8791=404876;8436=402261;7983=399753;7
067=395347;8662=404310;8648=404250;7130=395671;2138=368051;7188=396052,39
6053,396247;4802=383816,382685,389203,391185,396780,391508,396778,396793,39
6792,385390,404716,396775,396781,396777,389202,386263,393788,398844,398848,3
99966,399974,399975,385382,385430,385443;3=540;2=47;1=9;4=21;208=30101;206=3
5636;283=301505;284=301499;285=301492;205=29932;207=29947;140=49999;139=38
7373;408=312321;3190=373299;3170=373279;5687=388364;3145=373254;3164=3732
73;3165=373274;5684=388372;5683=388370;3167=373276|1183|0|0.000689630|
0.000689630|0.000689630|0.000683020|1.000000000|1|x7|
5272=1:93248100;6832=93248100;6828=Scg0OknINDr_sVA_Scg0OrIMMzpJyDQ6sQwz
OsTdeDoBp5c_AVJbc0q4AAAAAgAAAA|http%253a%252f%252frealestate.aol.com
%252fpictures%252ffeature%252freal-estate-news%253fpg%253d6|
MTI0OTA3NDczNDsxOjE0cTkyaGIxZzNqcm1mOjM2NQ

Basic Experimental Setup



Use Cases

Micro-event counter with changing layer configurations

User profiling with changing time window and configurations

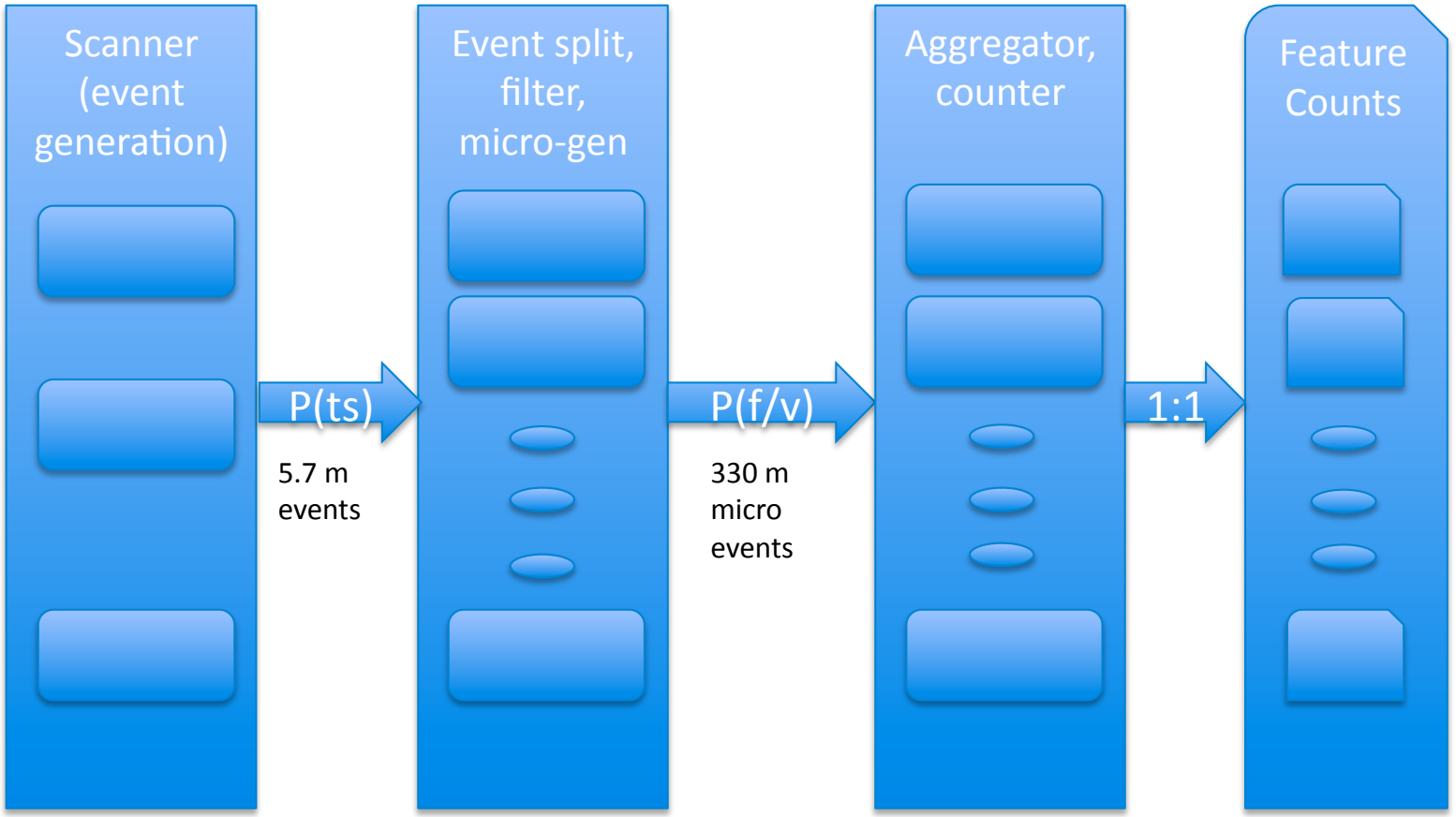
User-level campaign counter

Campaign count aggregation

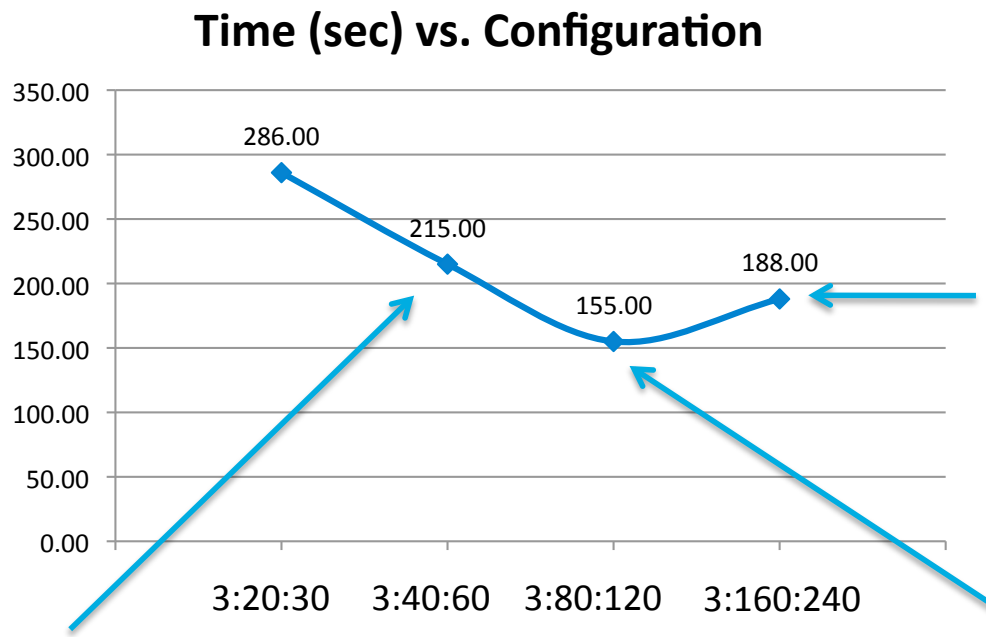
Stress tests:

- High bandwidth distributed message passing
- High data volume/memory requirement test

Use case 1: micro-event counter with variable layer configurations



Results, Scenario I

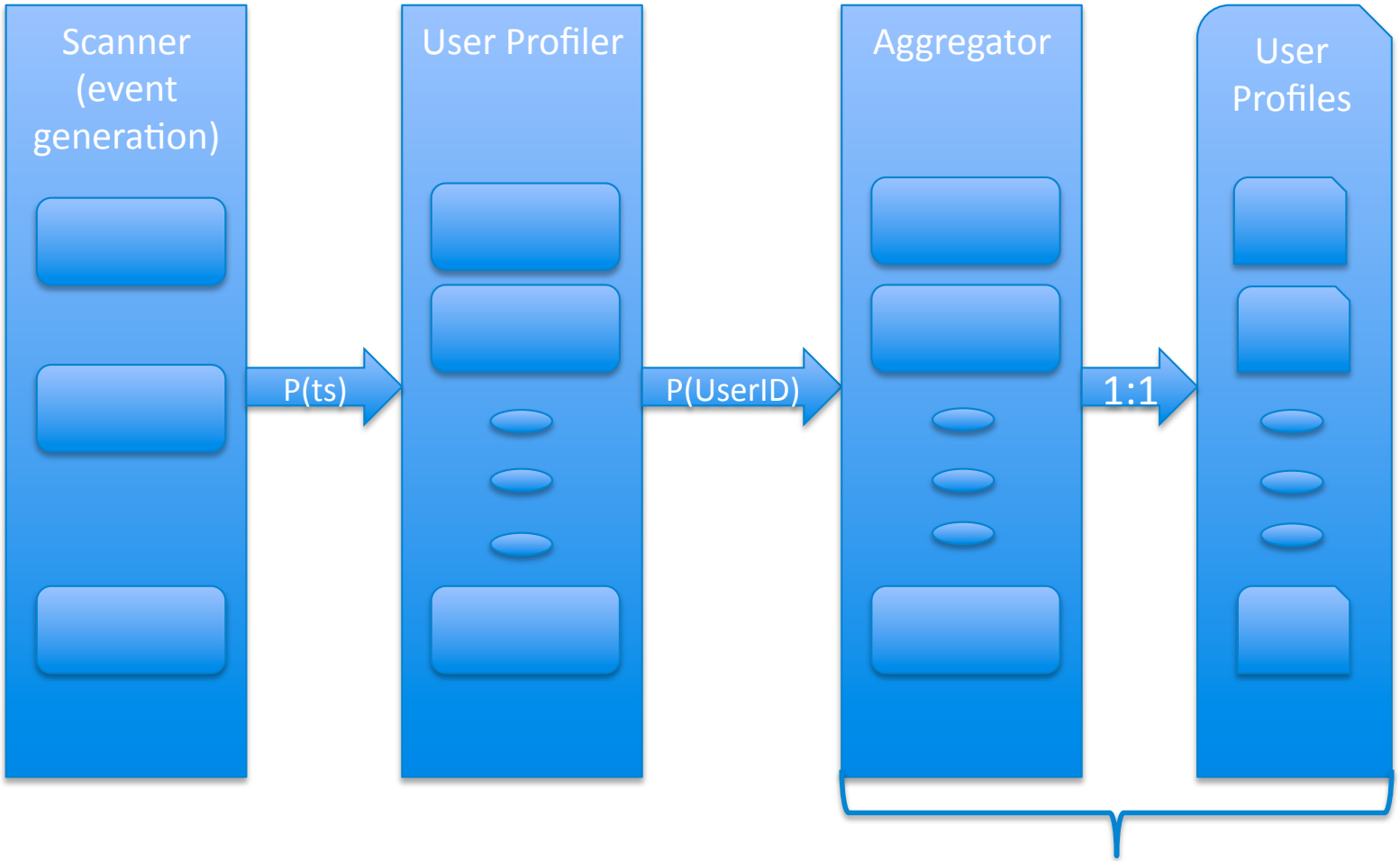


High load, context switching

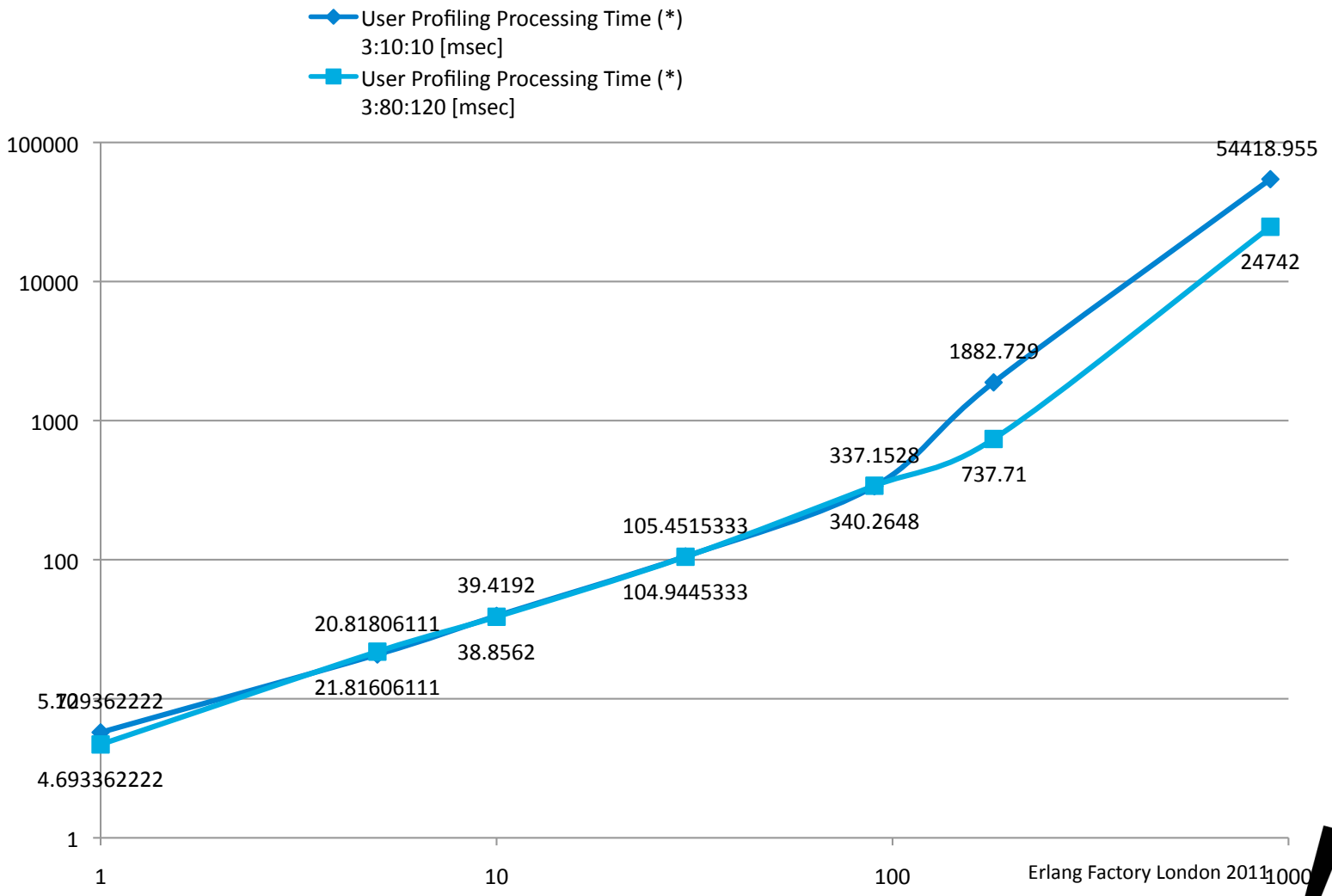
Increased performance and load, better response

2.13 mil. update ops/sec ~ 18K ops/sec/proc, load ~ 15

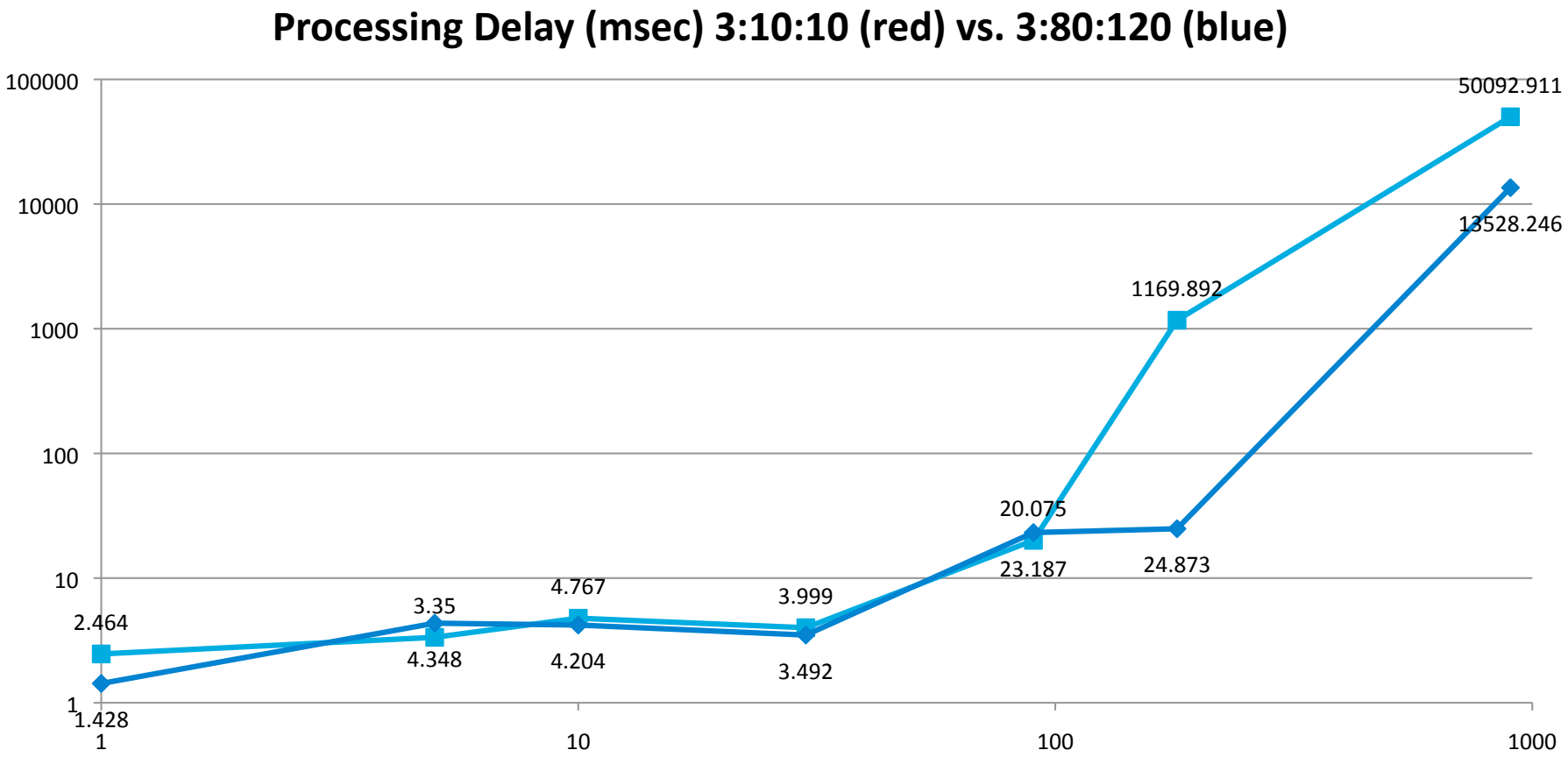
Use case 2: User Profiling with shrinking time window and two configurations



Results, Scenario 2

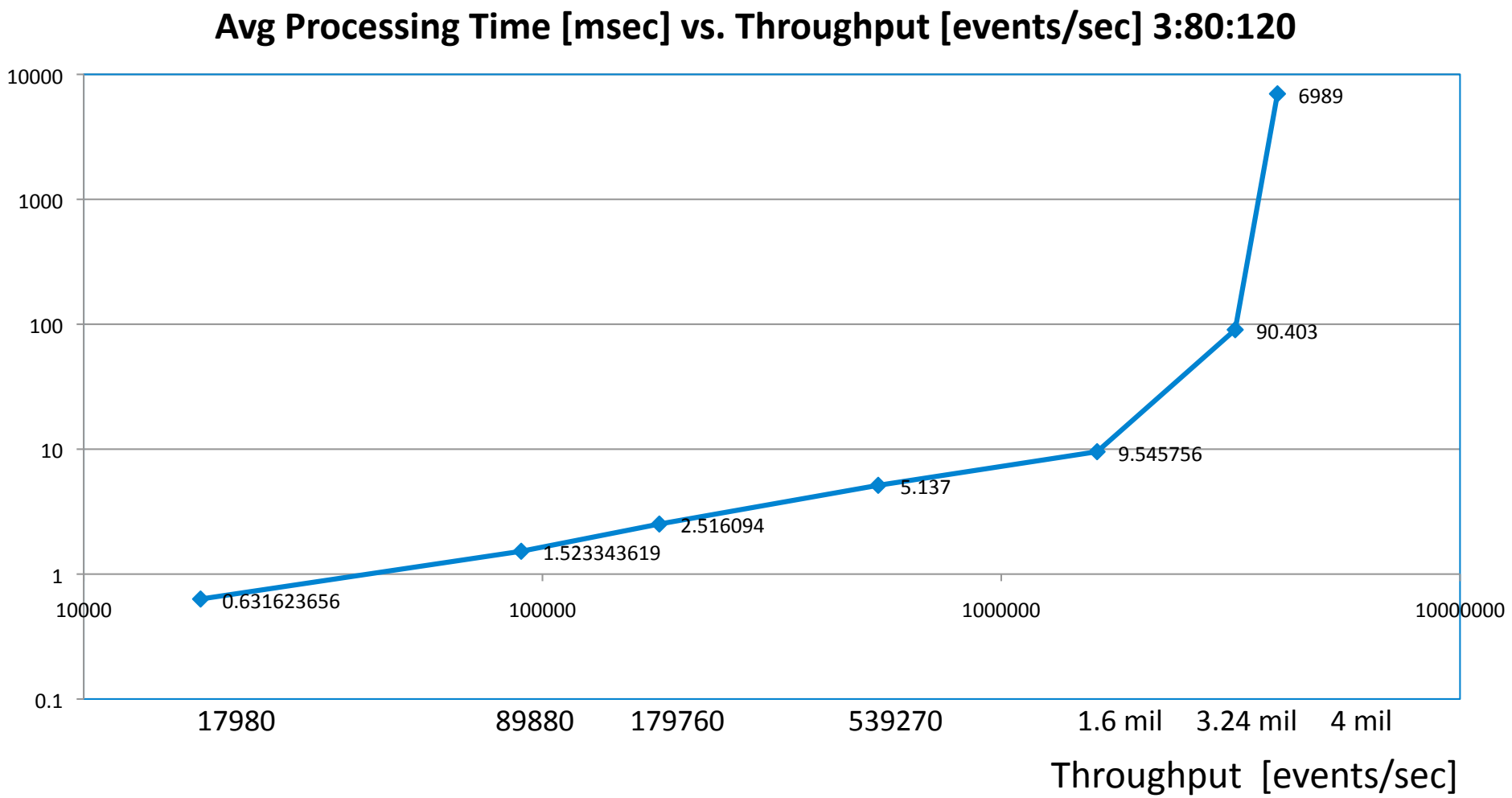


Results, Scenario 2



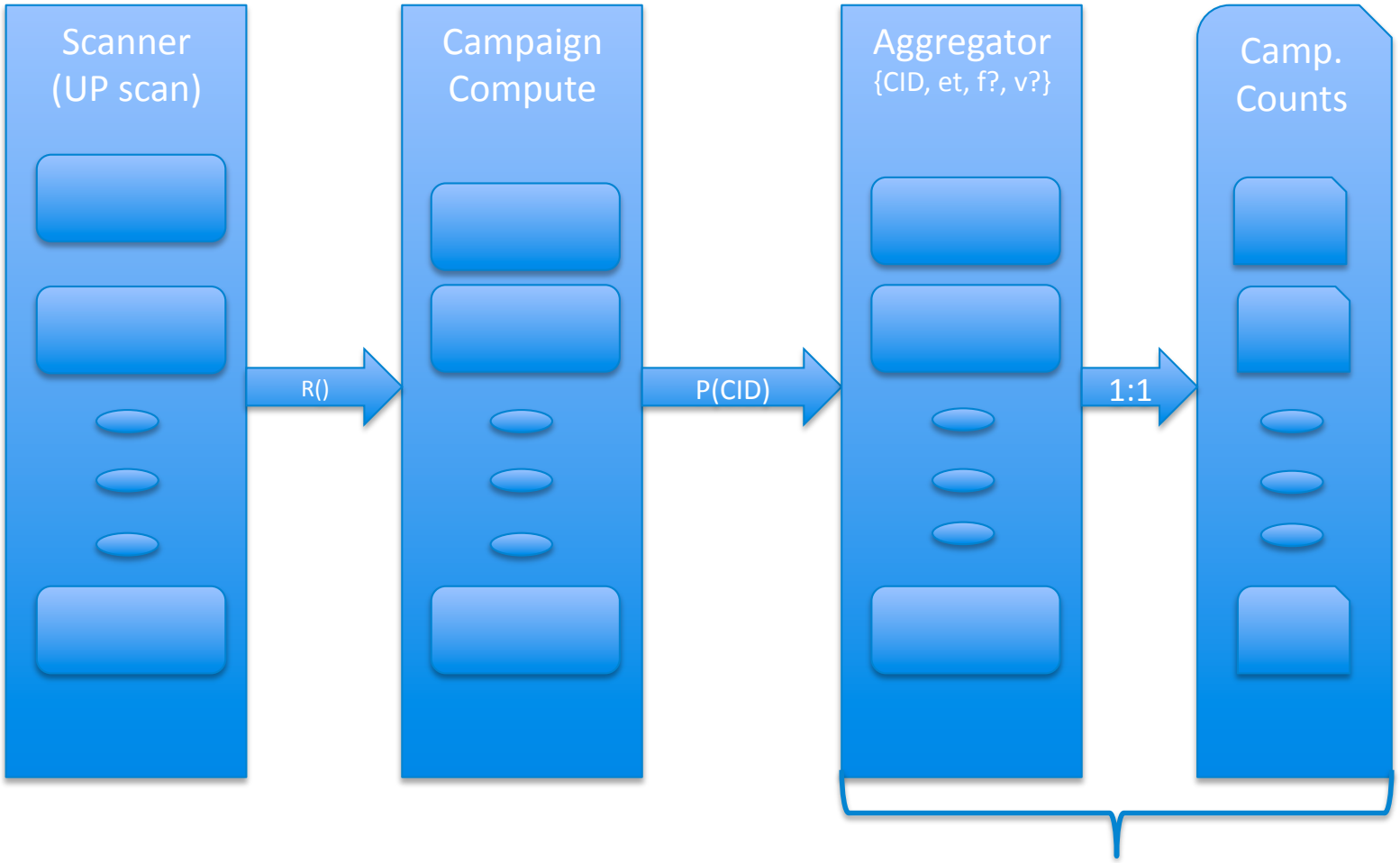
3:10:10 (light blue) vs. 3:80:120 (dark blue)

Results, Scenario 2



3:80:120 (blue)

Use case 3: Campaign Counts

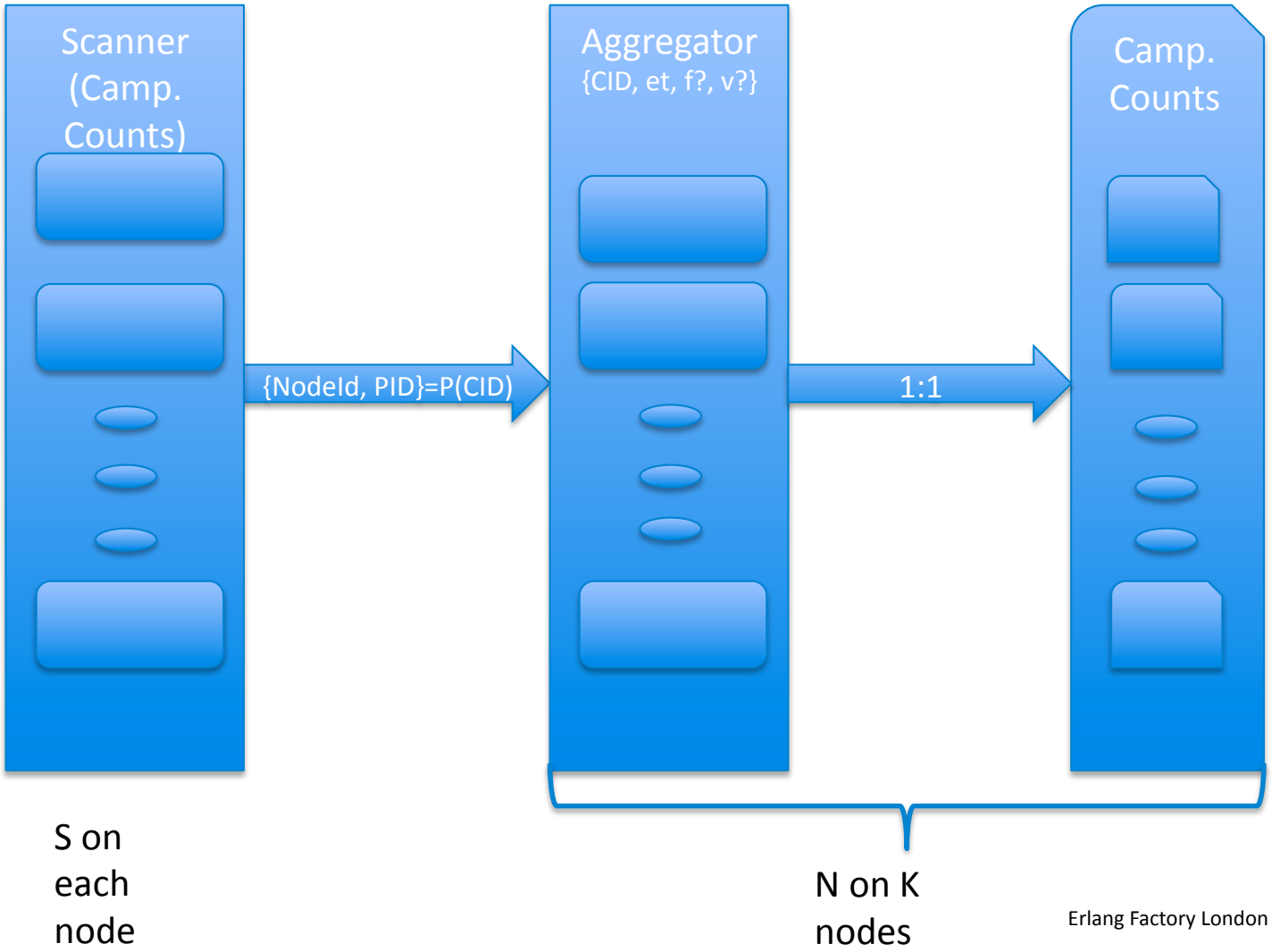


Campaign Compute Worker

```
Campaign compute (assumes one AdMedia table is pre-loaded to ETS
on each node):

for each user
  initialize counted feature set CFS = []
  for each event for the user
    extract mediaID from event record
    map mediaID to campaignID via AdMedia table
    enrich with campaignID
    project to 8 target features
    for each of the 8 features:
      generate tuple {campaignID, eventType, feature, value}
      check whether the tuple is recorded in CFS:
        yes -> skip; // we counted this user for this
                     // {campaignID, eT, f, v} combination
        no -> record the tuple in CFS;
        send a message to aggregator
        {{campaignID, eventType, feature, value}, 1}
```


Use case 4: Cross-node Aggregation



Results, Scenario 2+3+4: Complete Campaign Count Flow

15 minute data volume (about 5 mil. events/node = 50 mil. across all nodes)

Generates counts for I, A, C, L event types and 8 features: country, state, metro, browser, OS, language, domain, connection speed

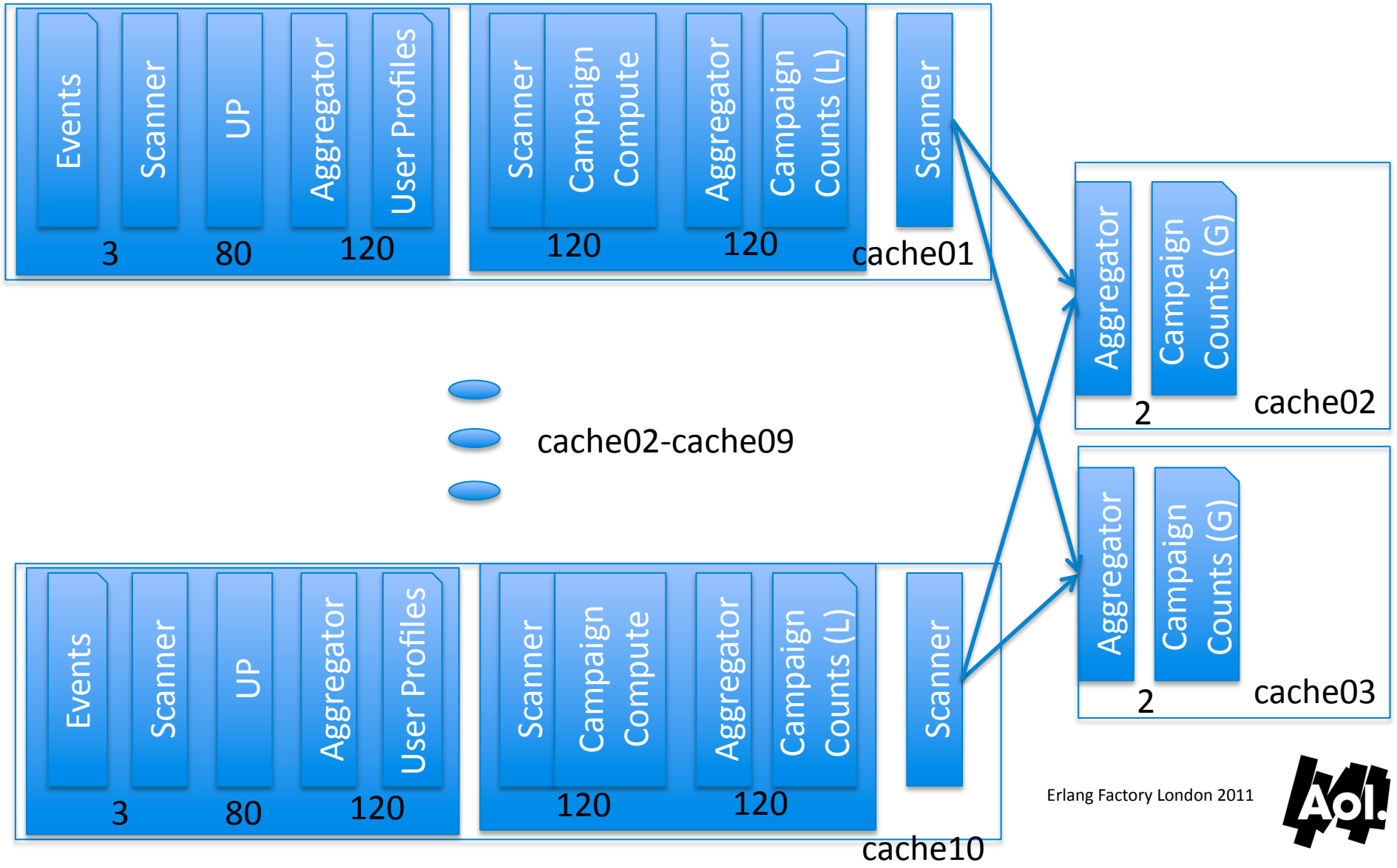
3:80:120 configuration: User Profiling takes 25 sec,

120:120 configuration: Campaign Count ~60 sec.

Final aggregation across all nodes: 10 X 120 -> 2 X 2, 5.28 secs

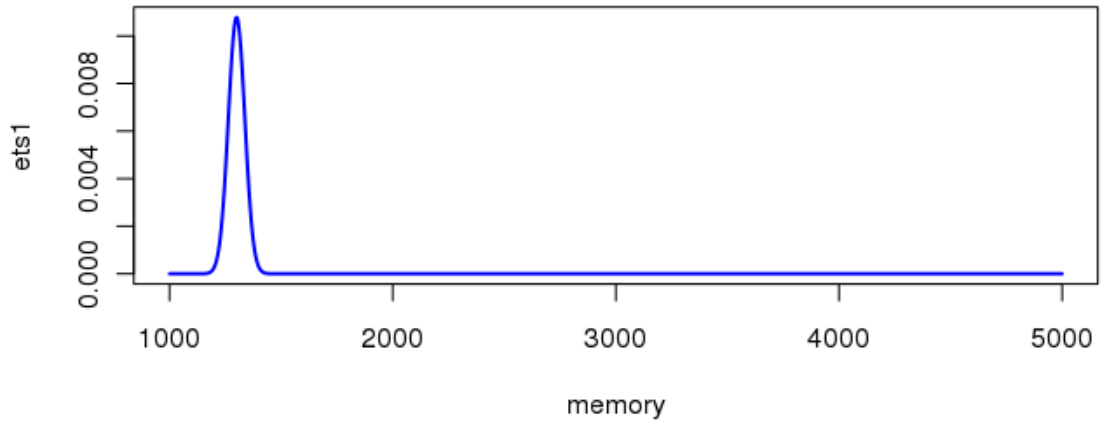
Total: ~90 secs (10% of data frame)

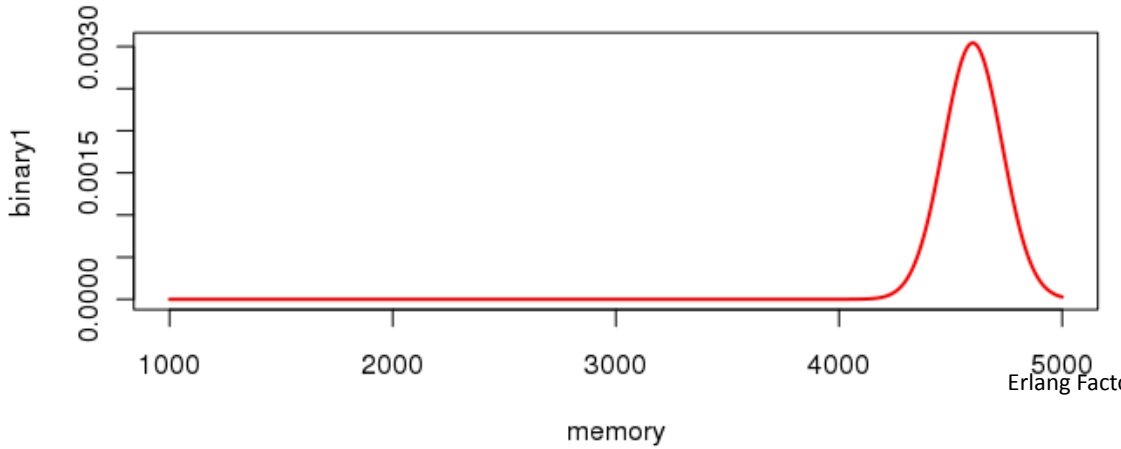
Full Campaign Count Flow



Ganglia Monitoring

Time and Memory Distribution across Physical Nodes





Architecture

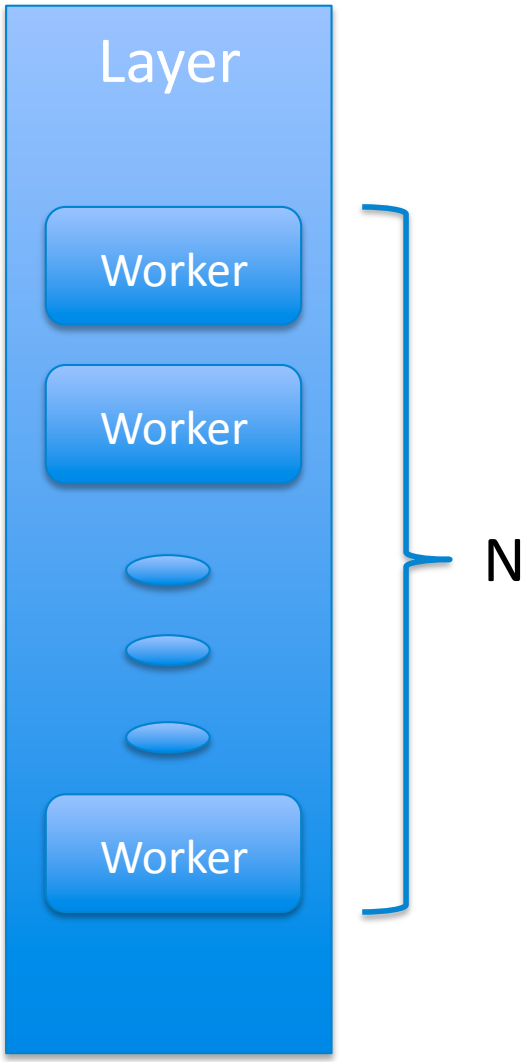
Flow – Layer – Worker hierarchy

ETS/DETS tables for staging and intermediaries

Parameters:

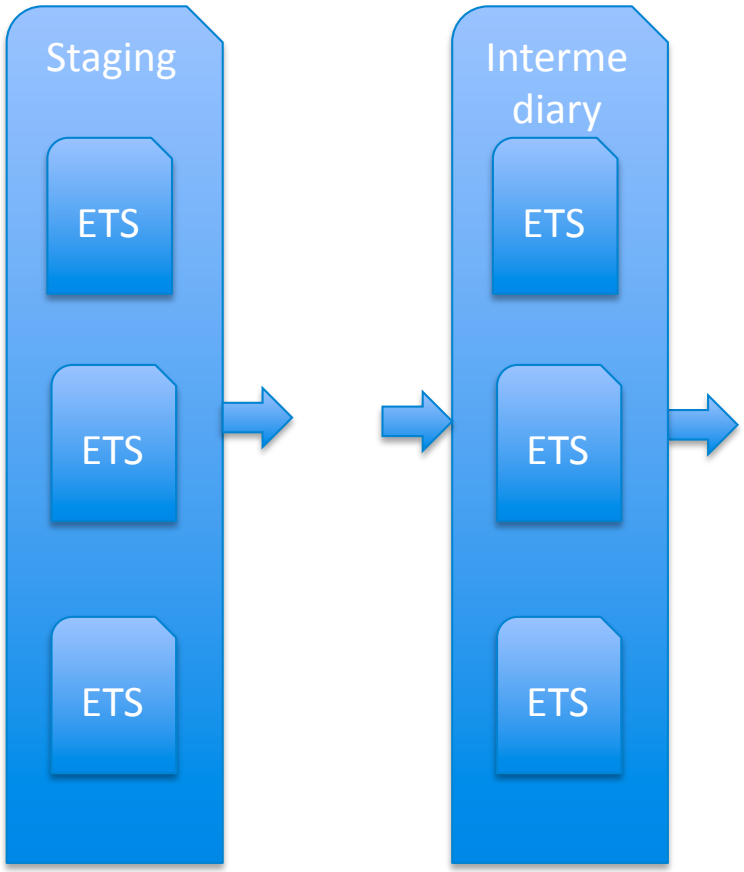
- Layer size
- Layer identification
- Layer input, output data/format and connectivity with adjacent layers
- Mapping functions between layers: partitioning (e.g hashing), load balancing (round-robin, random uniform, etc.)
- Compute function (user-defined, external)
- Layer to physical node mapping

Layer



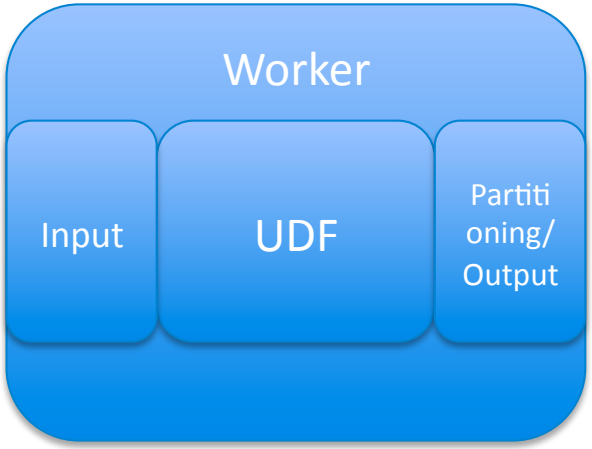
- Is a supervisor
- Supervises workers
- Elastic in number of workers
- Workers perform uniform functions
- Connects to other layers or staging/intermediary tables
- Maps to physical nodes

Staging and Intermediaries



- Storage for crucial data sets
- Staging used as input for one or more flows
- Intermediary: output from previous layer, input to the next one; e.g. implementing barrier concept in MR
- Optional persistence with DETS
- Both can be used for staging input for multiple downstream layers/flows
- Important for memory capacity planning

Workers



Input layer: predefined for ETS scan

UDF assumes intermediary/ staging table record format

UDF defined externally, still Erlang

Partitioning/load balancing function: predefined or custom

Output to intermediary or another layer

All functionality local to a node

Future

3 months:

Expose campaign counts via GUI: demonstrate segment lifts in real time

More experimentation, expanding to more use cases, improving code base

Code refactoring, modularization to reflect architecture specs closer to framework/platform

6 months:

Redundancy and failover

In-stream analytics algorithms

Advanced large-scale algorithms via GPU/pteracuda

Everybody talks Erlang!