DATA IS POTENTIAL

CORTX observability overview

Anatoliy Bilenko

SEAGATE

What is it for?

- Systems are large, complex and run related activities in multiple processes on different nodes
- Relevant and structured information related to all such activities needs to be aggregated, related together and drawn preperfrably on a single interactive diagram
- Relate system events and CORTX events togther
- Discover system behavoiur to understand and validate it based on analysis of related state machines lifecycles
- Discover failure, performance, synchronisation scenarios
- Provide a sufficient level of details on every layer of the stack
- Do not interfere with performance
- Provide an interface for data mining to track anomalies in automated way.

Source/Footnote:

Key features. How is it different from others?

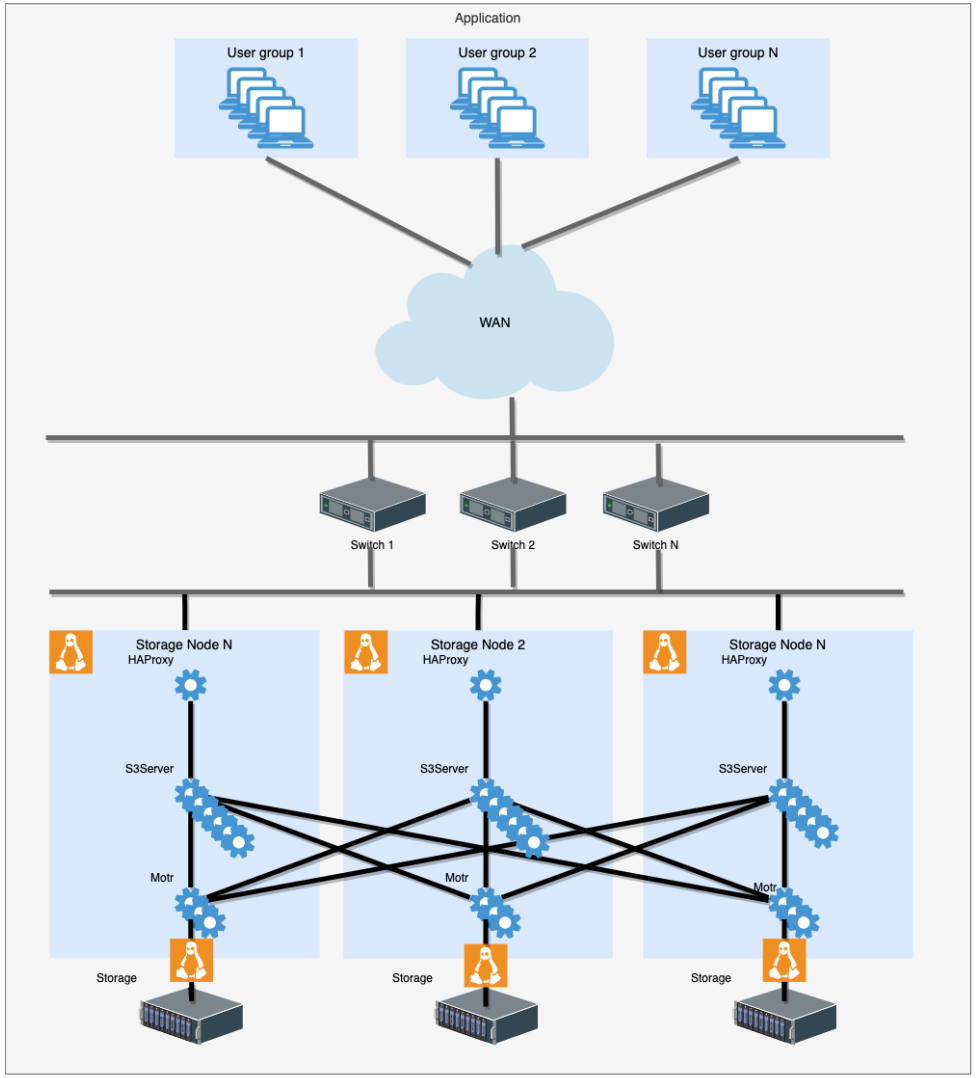
- Blazingly fast aggregation framework based on ADDB embedded into any motr applications (S3 servers, motr servers, motr clients)
- Request based observability (Cover all motr server and S3 server state machines in automated manner) (R.timelines)
- Dynamically explored data schema (R.relations)
- Dynamically explored attributes (R.attr)
- CORTX stack profiling based on statistical framework
- Performance metrics based on queueing theory
- Decorating observability diagrams with system level events:

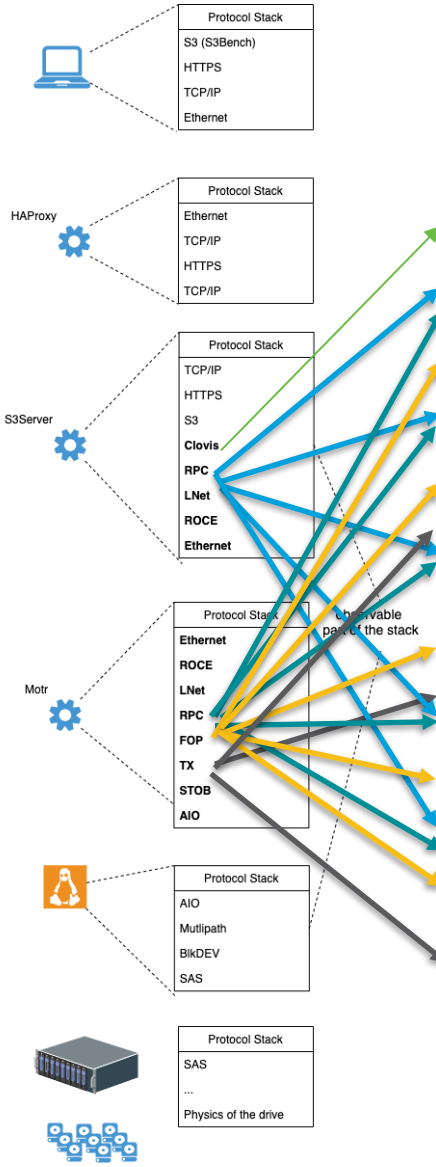
i. CPU on/off
ii. Mutex lock/unlock
iii. CPU intesive places
iv. #PF, CTX swcs.

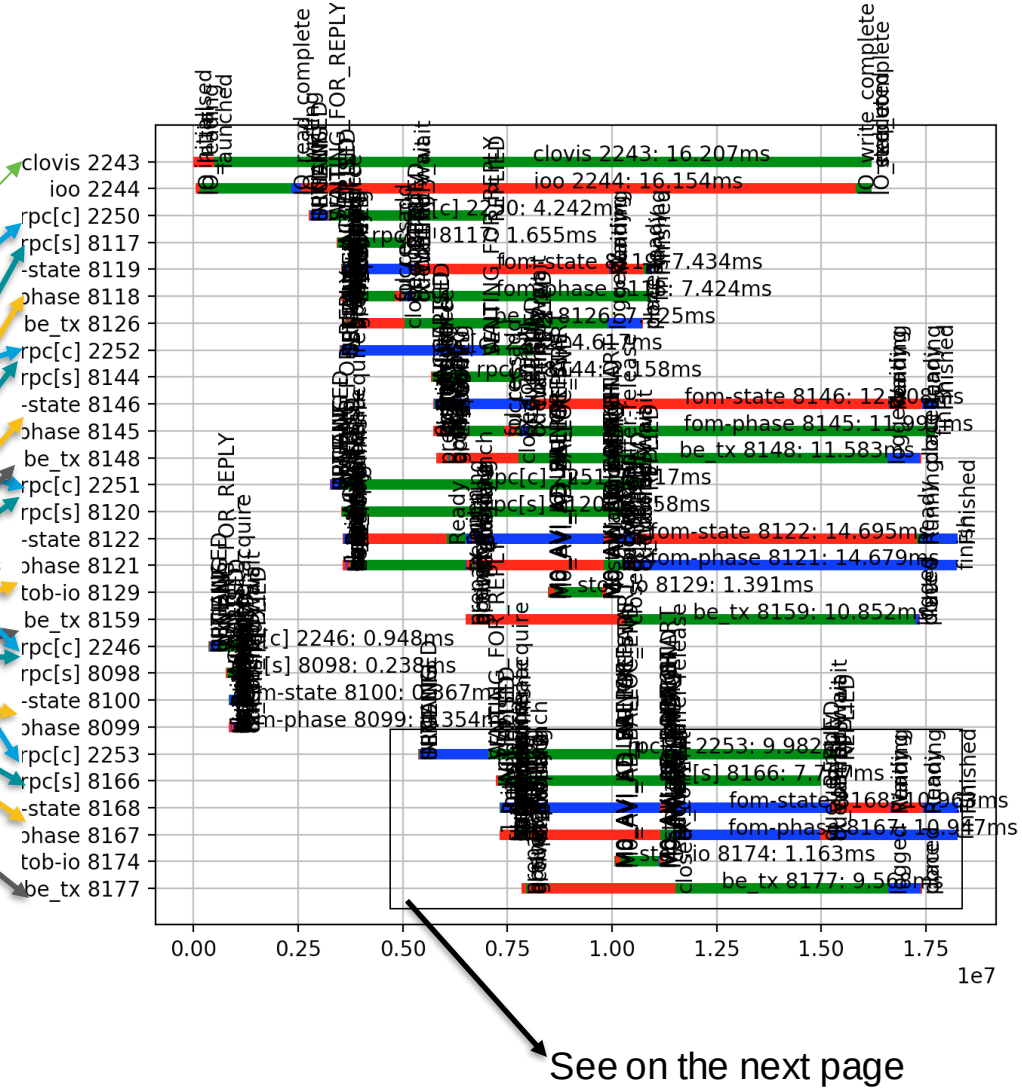
- Workload analysis
- Data mining framework

Source/Footnote:

How is it looks like? Holistic view (1)



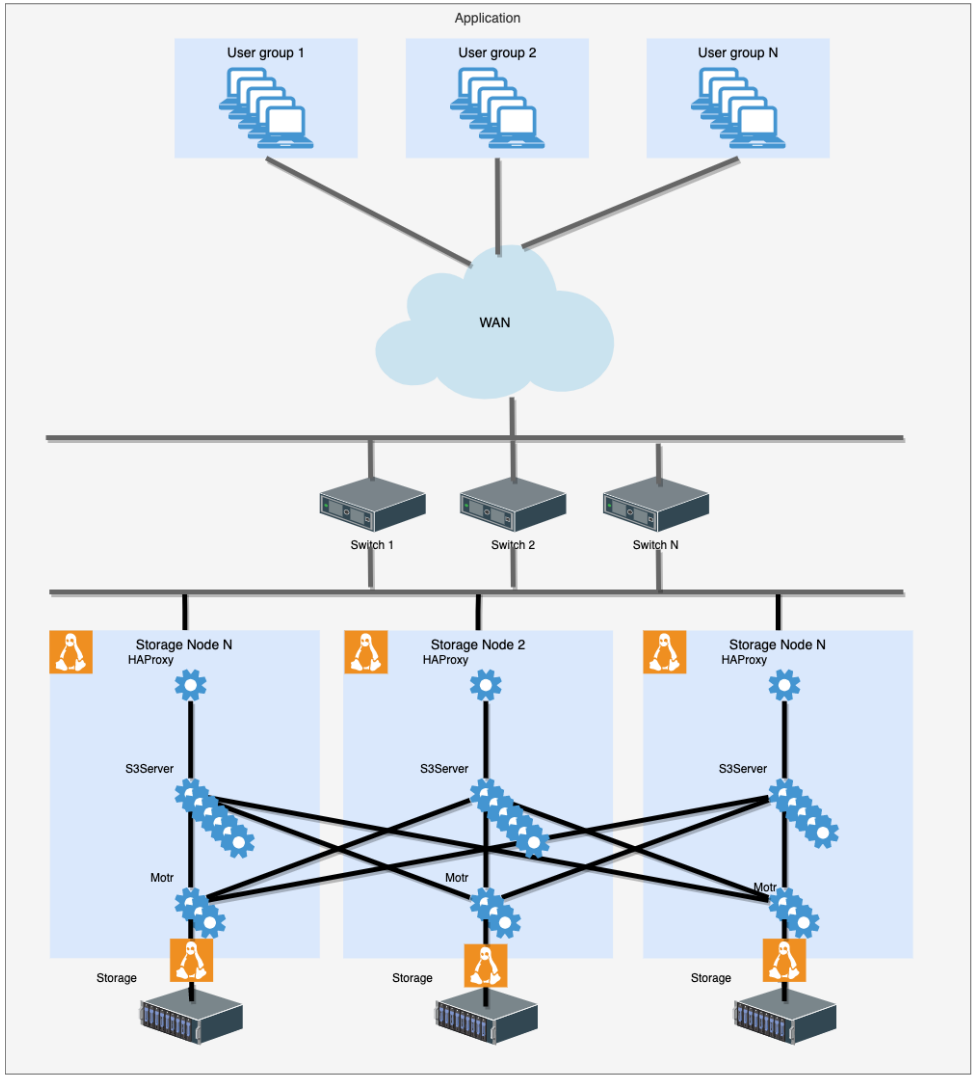


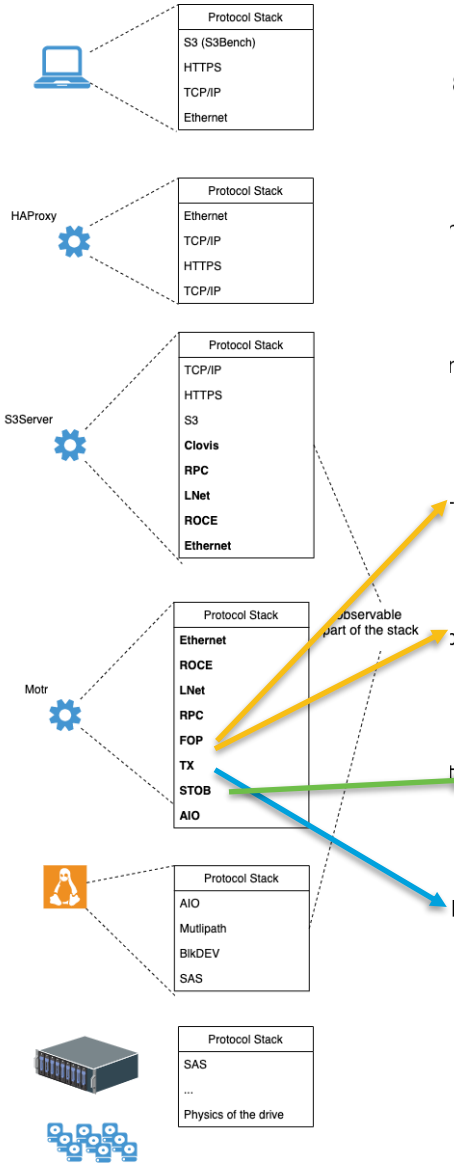


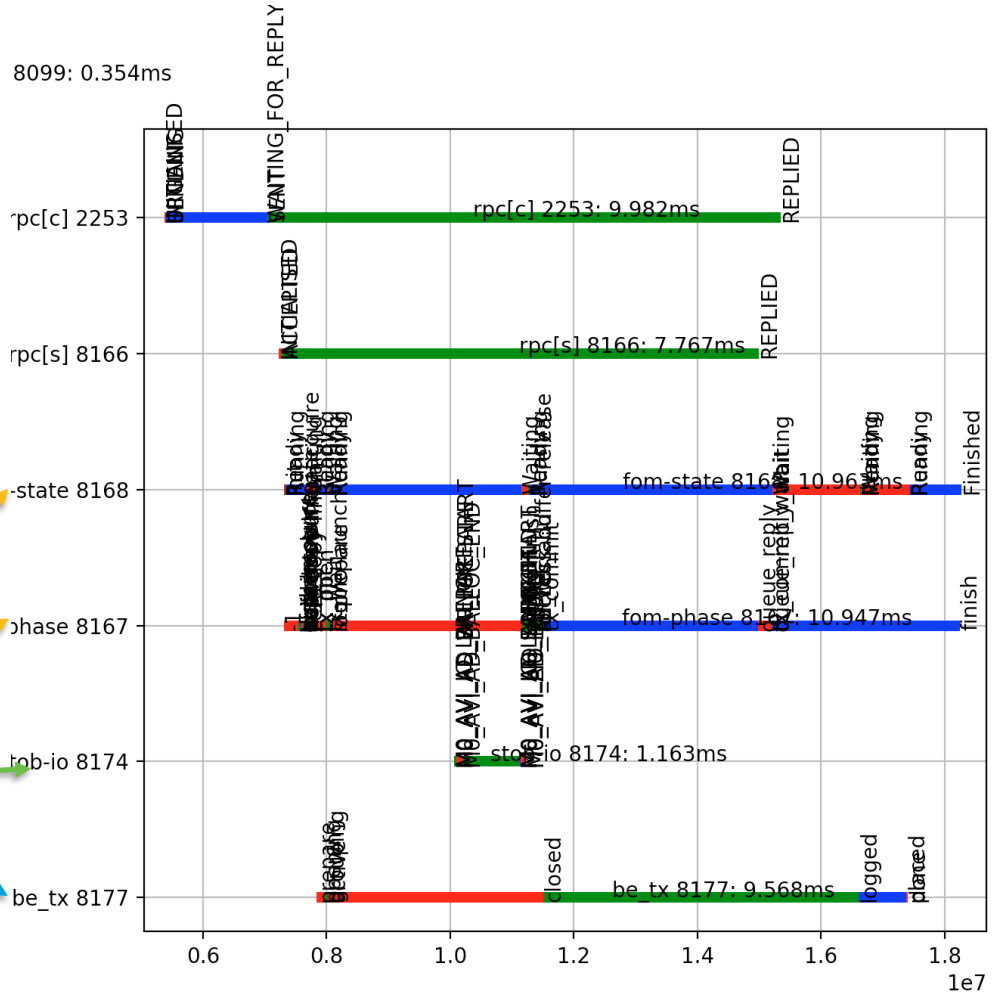
See on the next page

Source/Footnote:

How is it looks like? Interactiveness, easy to traverce all samples (2)

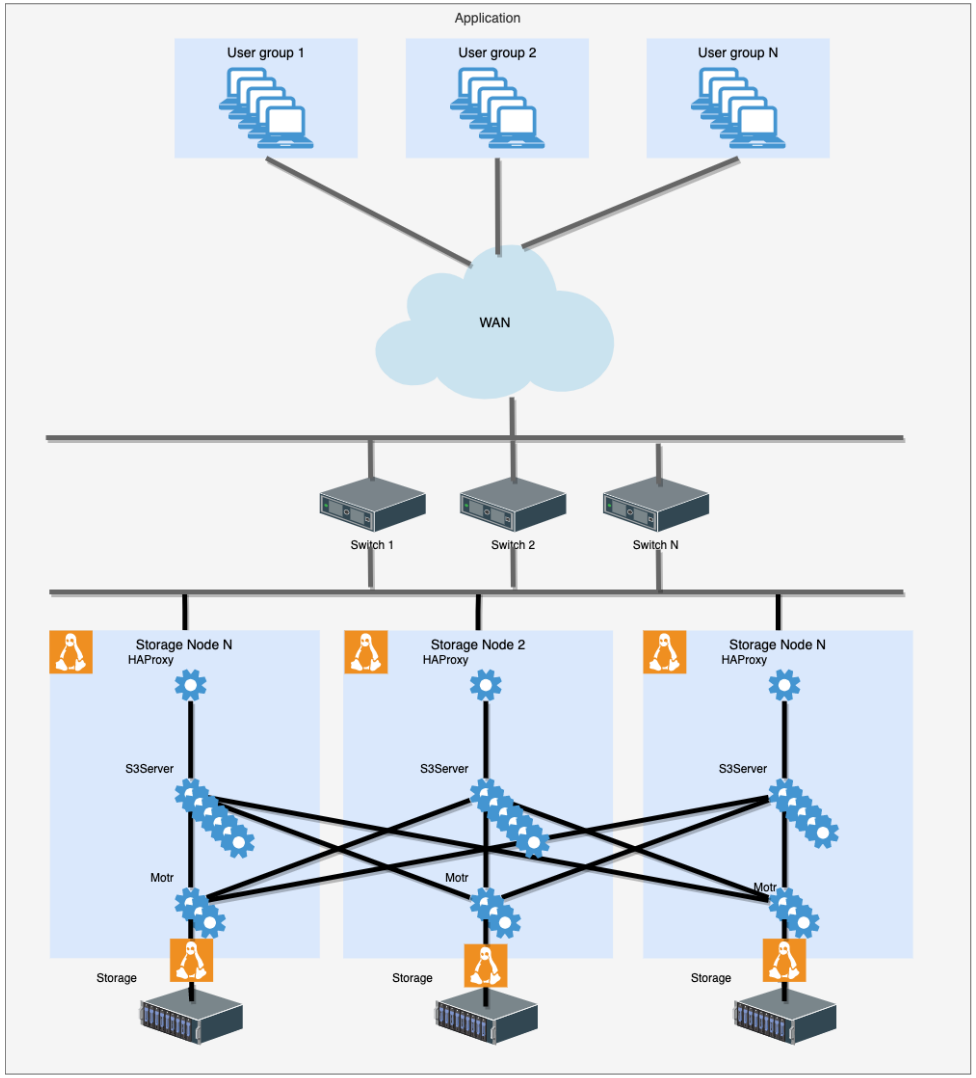


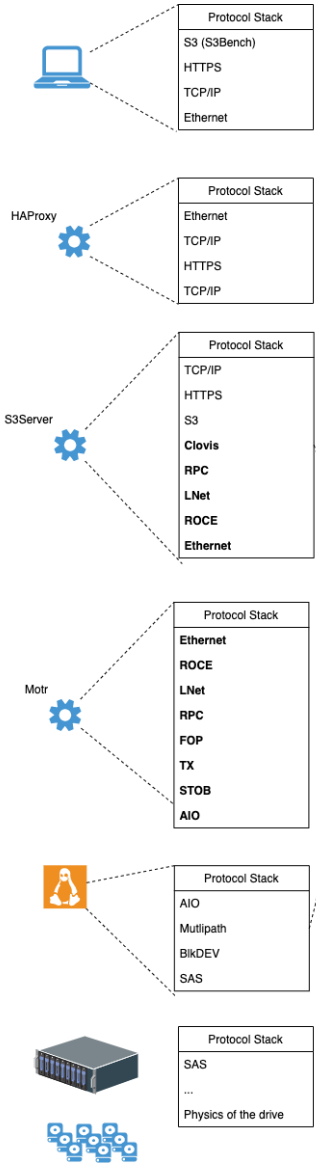


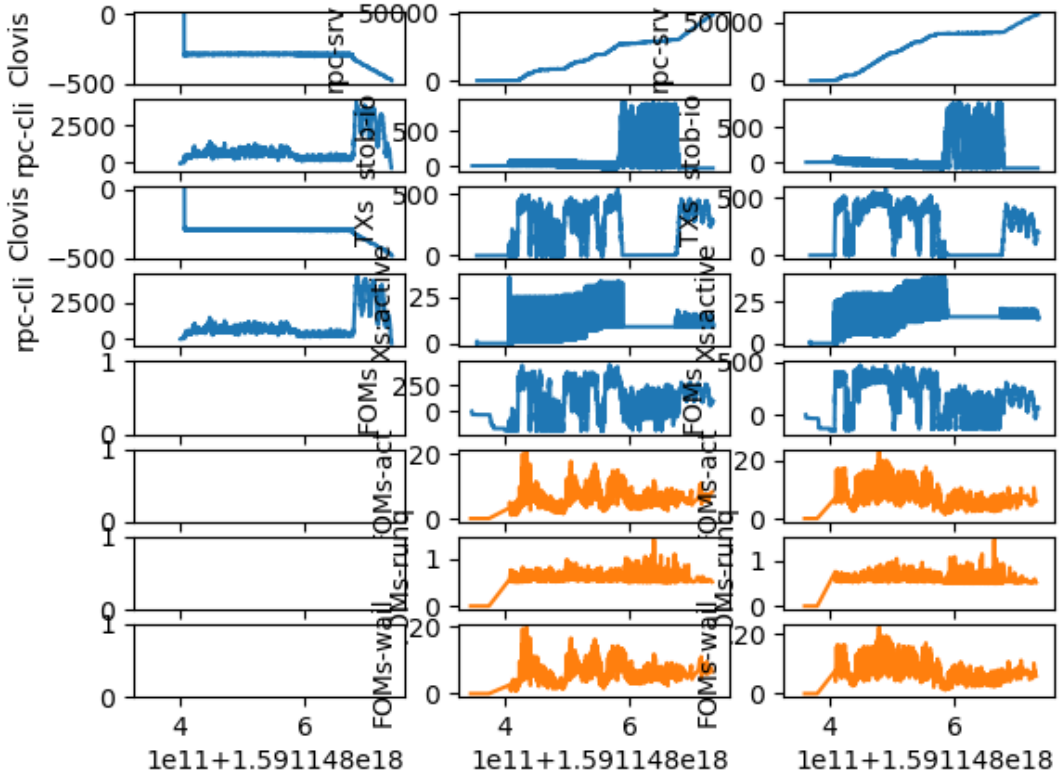


Source/Footnote:

How is it looks like? Layers, queues, workload analysis (3)

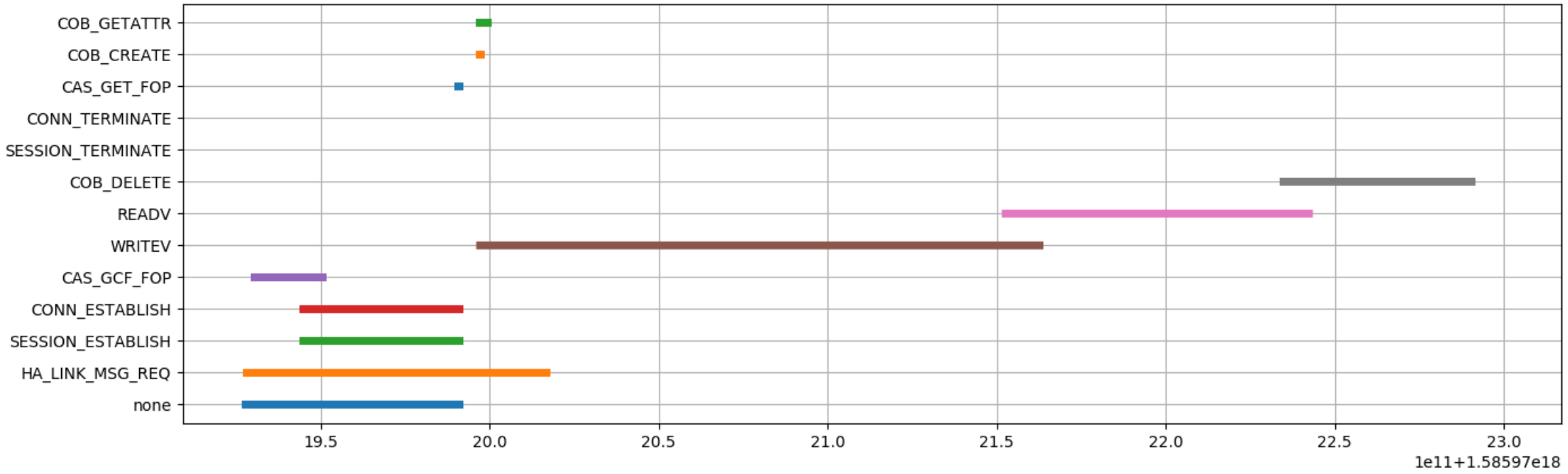






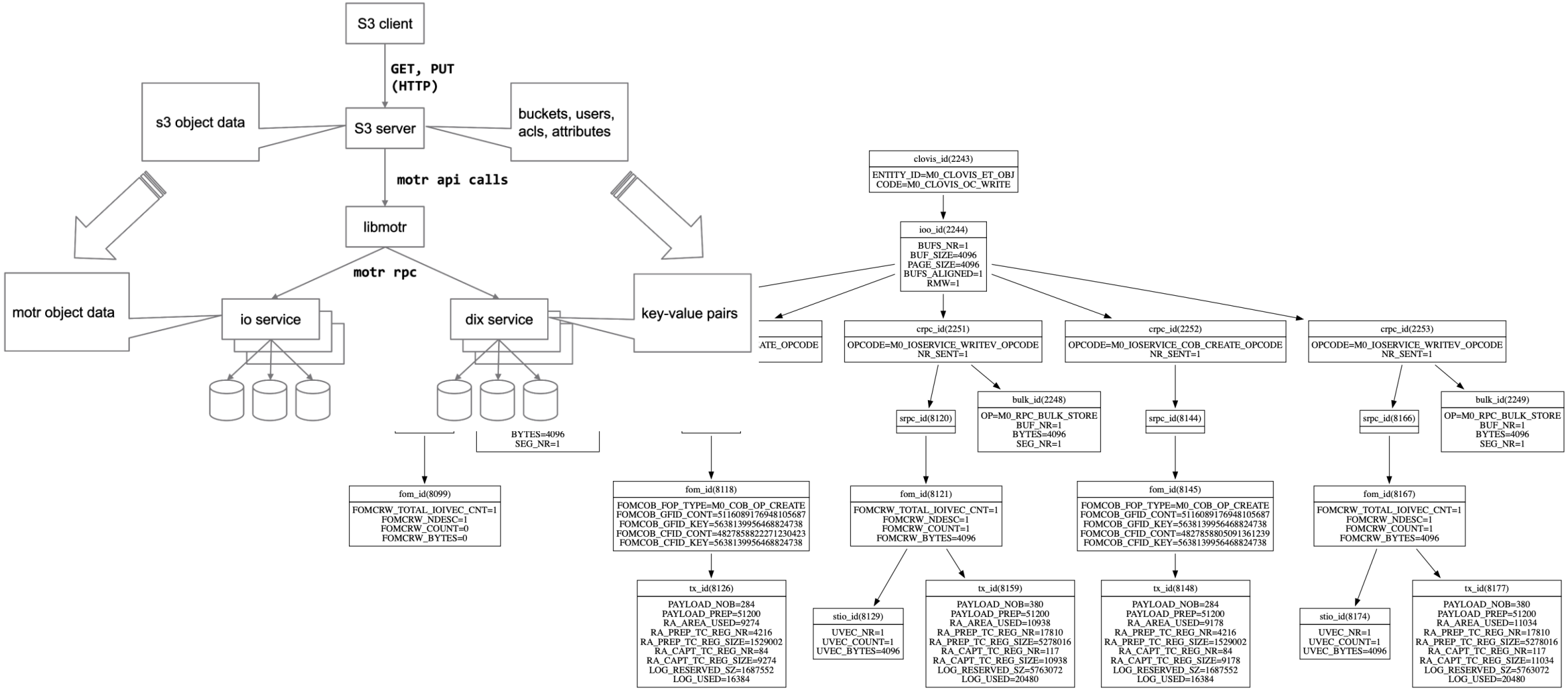
Source/Footnote:

How is it looks like? Workload analysis (4)



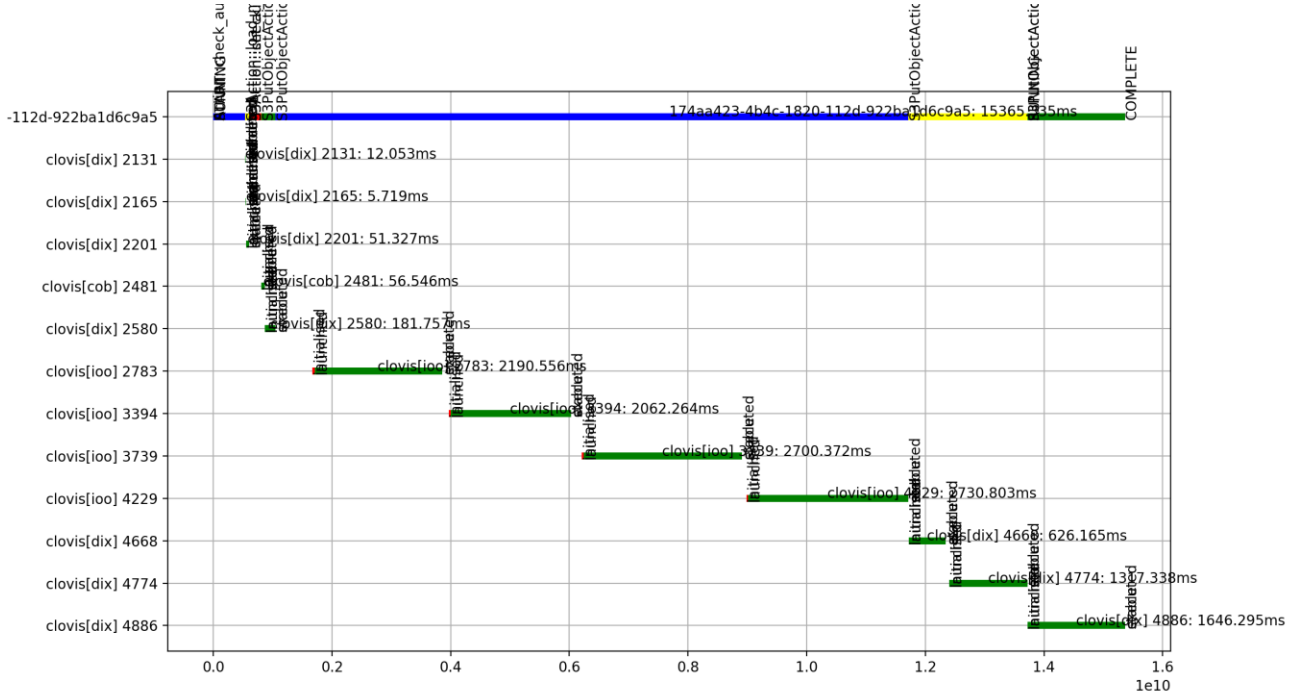
Source/Footnote:

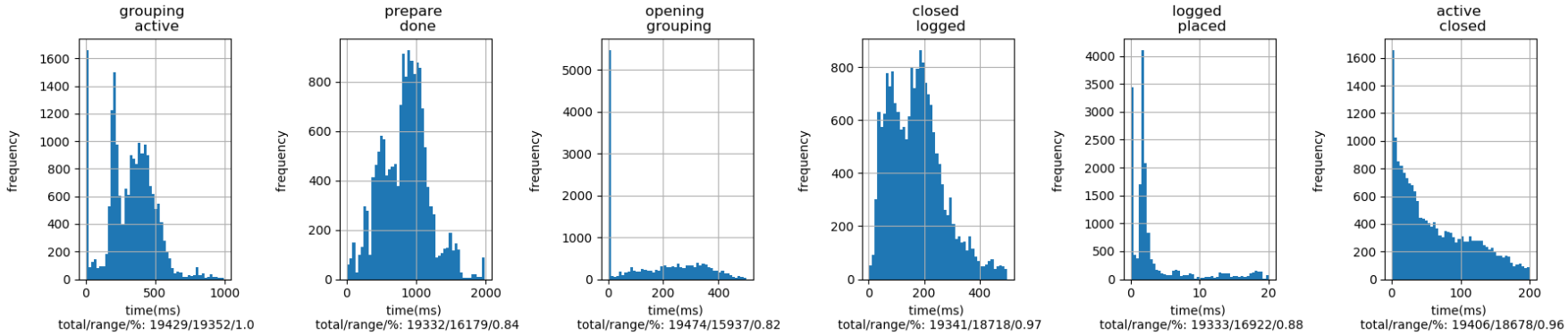
How is it looks like? Flow graphs (5)



Source/Footnote:

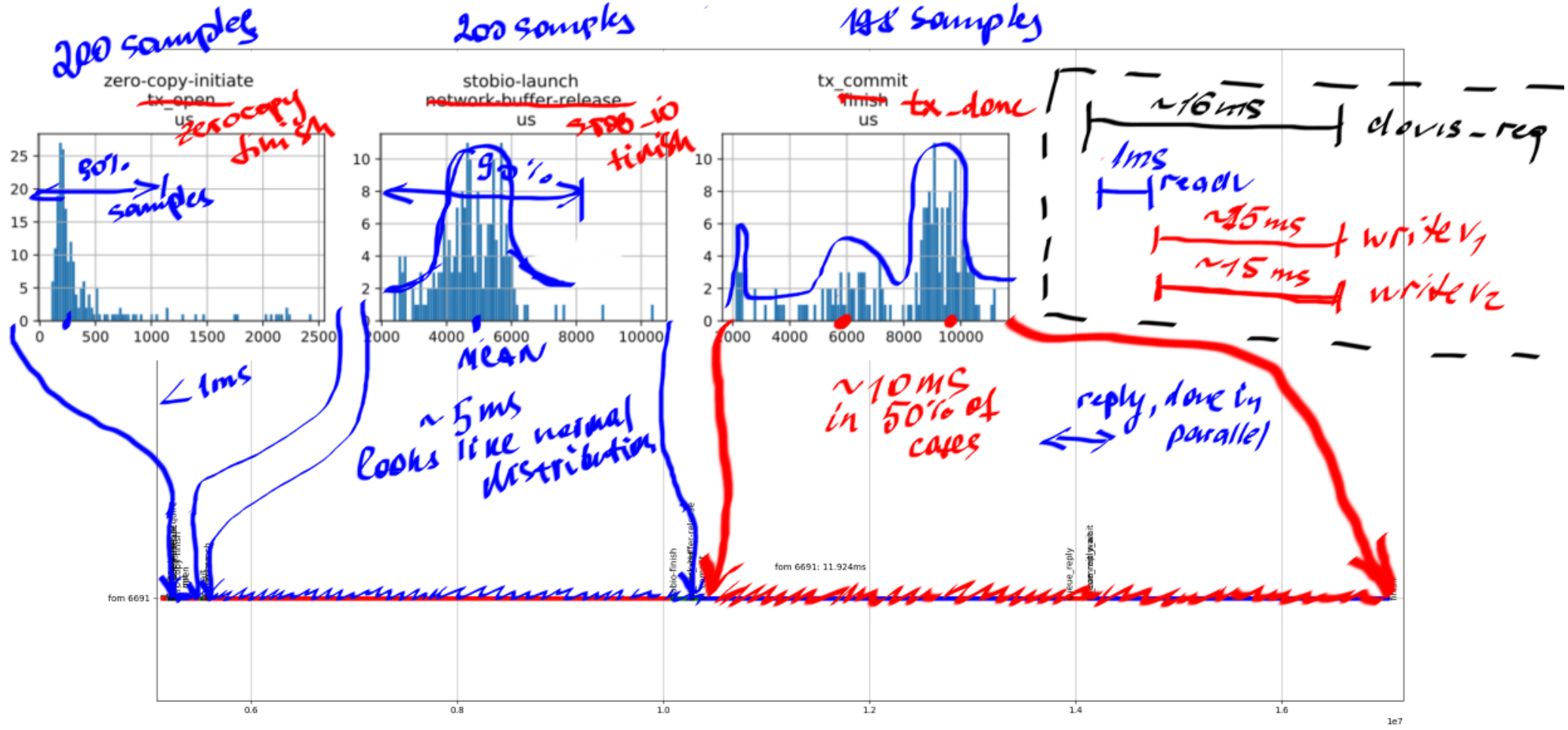
How is it looks like? Performance analysis (6)





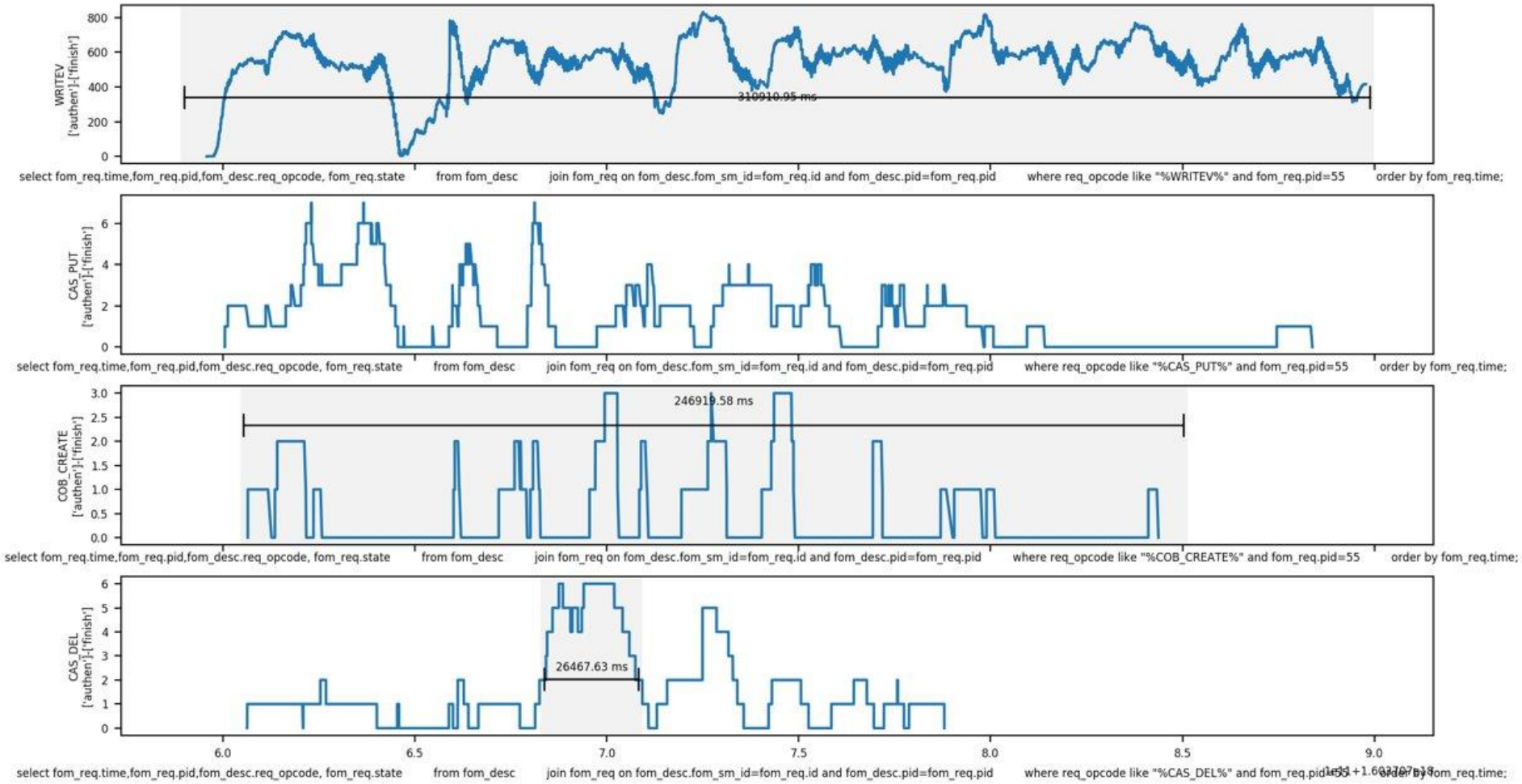
Source/Footnote:

How is it looks like? Performance analysis (7)



Source/Footnote:

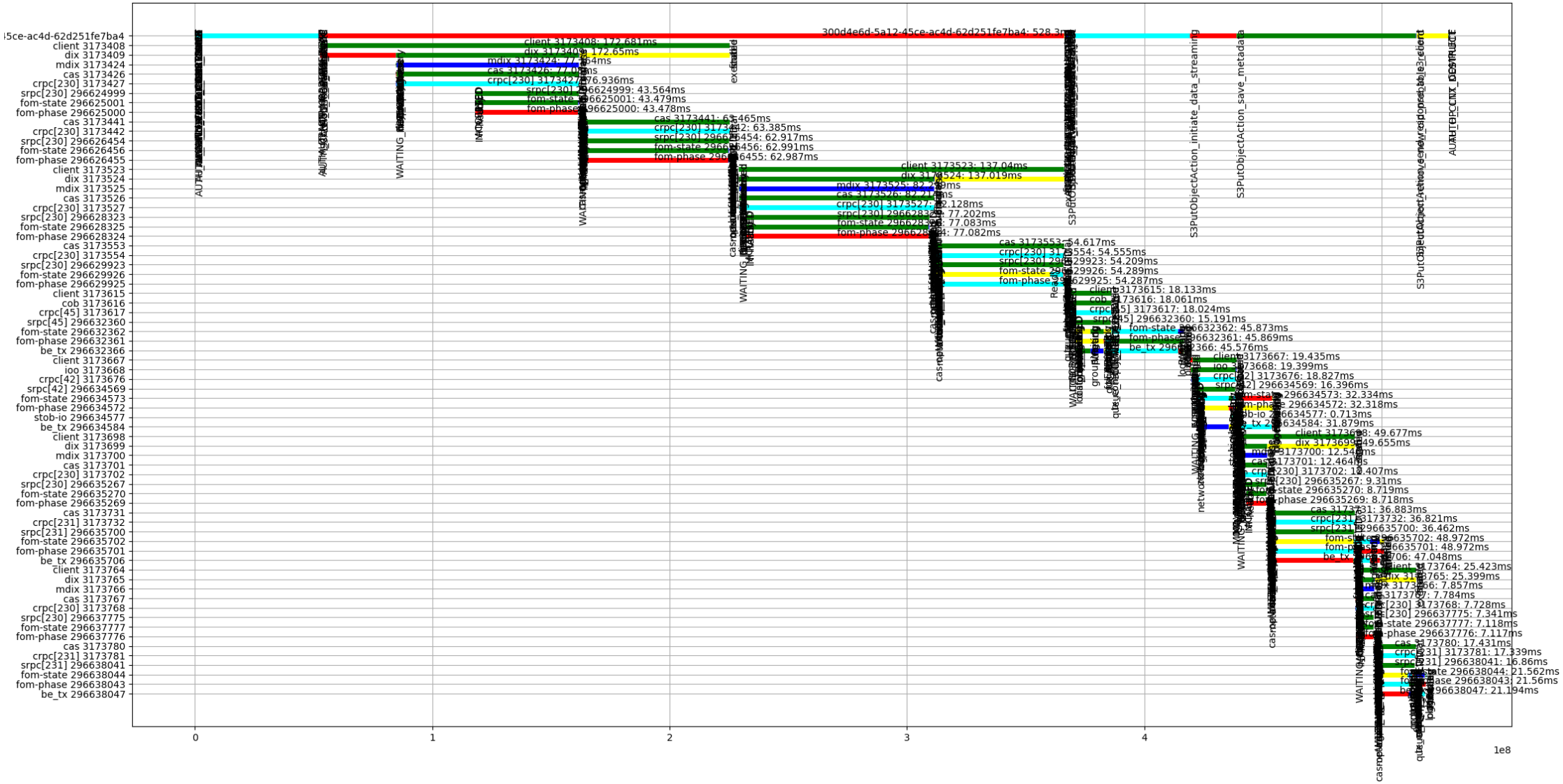
How is it looks like? Data mining framework (9)



- starvation identification
- request level analysis

Source/Footnote:

How is it looks like? Even more sofisticated requests (8)



Source/Footnote:

How to build such an observability system? What's significant?

- Q: What's the observability sample?
- A: it's addb2 sample gathered with m0_addb2_add(id, [uint64_t counters])
- Q:What does sample corresopond to on diagrams?
- A: Every event, for example shown as a "tick" on a timeline, correspond to such sample.
- Q: How to gather samples efficiently? What are physical limitations?
- A: Asume we want to gather observability samples for each request in the workload giving 10GB/s from the node with 1MB blocks. How many samples are needed here? **SamplesNR=LayersNR*SamplesPerLayer*NumberOfRequests.**
- Q: what is it in MB/s?
- A: **SamplesMB/s=SamplesNR*AverageSampleBytesNR.**
- Q:What framework is being used?
- ADDB2.
- Q: What's the instrumentation schema being used?
- A: Schema captures relations, attributes and timeline info which is sufficient to calculate other related metrics offline.
- Q: What's the samples aggregation flow?
- A: We use intermediate sqlite3 db (m0play.db) and aggregation tools (chronometry).

Source/Footnote:

ADDB

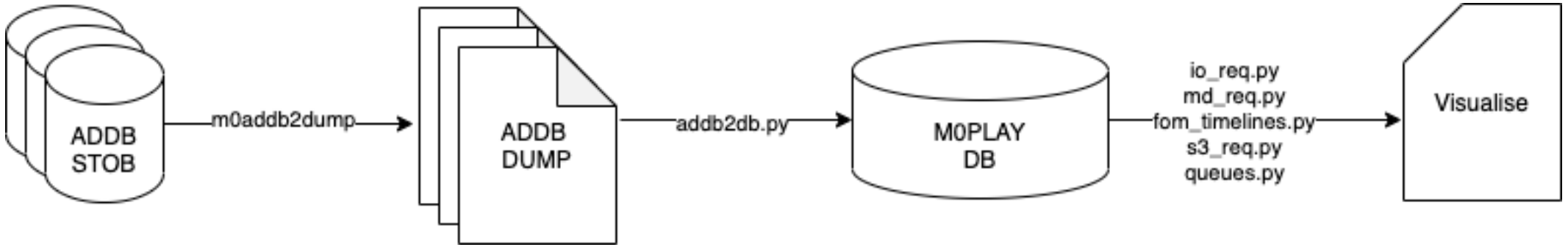
- instrumentation on client and server
- data about operation execution and system state
- passed through network
- cross-referenced
- measurement and context
- timestamped
- labels: identify context
- payload: up to 16 64-bit values,
- interpreted by consumer
- always on (post-mortem analysis, first incident fix)
- simulation (change configuration, larger system, load mix)

```
* 2020-02-20-14:36:13.687531192 alloc size: 40, addr: @0x7fd27c53eb20
|  node             <f3b62b87d9e642b2:96a4e0520cc5477b>
|  locality         1
|  thread           7fd28f5fe700
|  fom              @0x7fd1f804f710, 'IO fom' transitions: 13 phase: Zero-copy finish
|  stob-io-launch   2020-02-20-14:36:13.629431319, <200000000000003:10000>, count: 8, bvec-nr: 8, ivec-nr: 1, offset: 0
|  stob-io-launch   2020-02-20-14:36:13.666152841, <100000000adf11e:3>, count: 8, bvec-nr: 8, ivec-nr: 8, offset: 65536
```

Source/Footnote: from Motr Arch demo

Typical flow

- Start the cluster.
- Run the the test (s3 bench).
- ADDB Samples are gathered into ADDB stobs
- Dump addb samples with m0addbdump.
- Put samples into intermediate db with addb2db.py
- Visualise diagrams



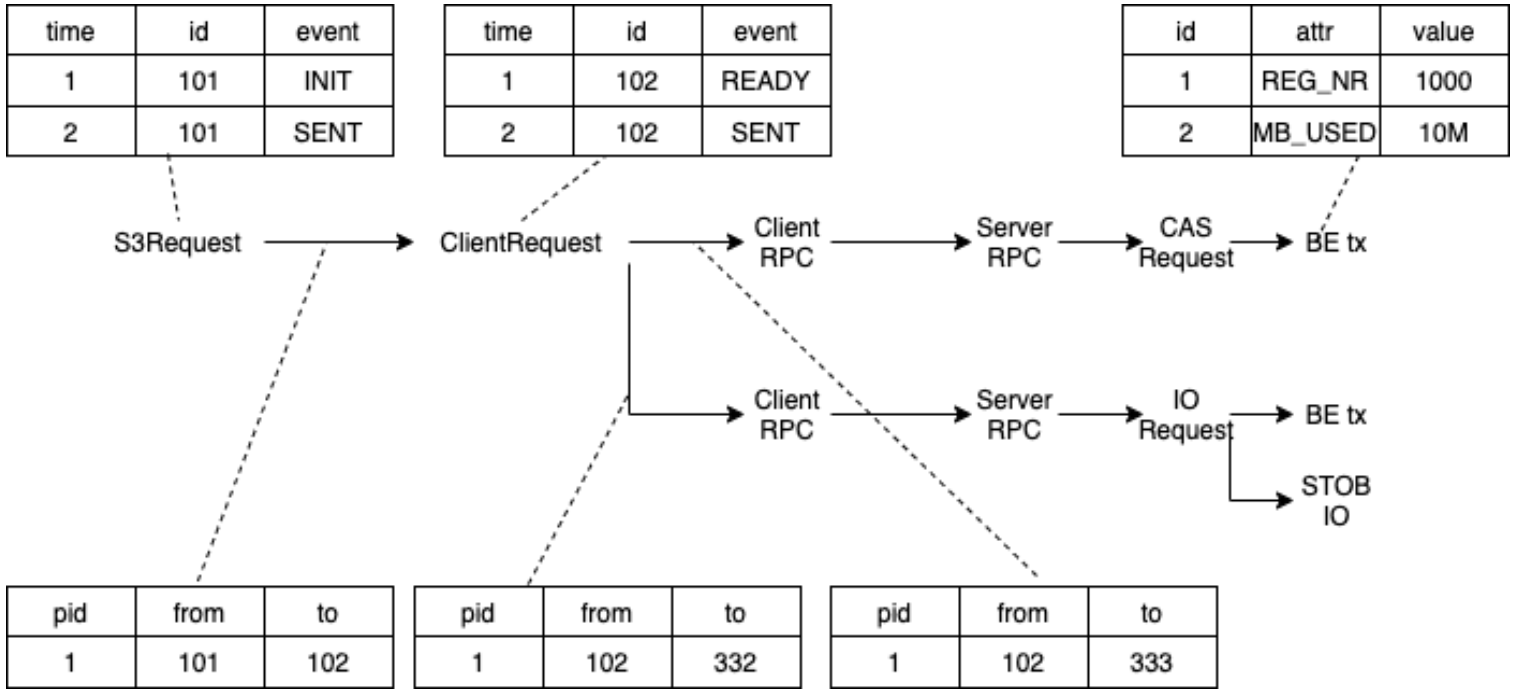
Source/Footnote:

Instrumentation schema

events:
- time
- process_id + event_id
- event_name

relations:
- from (process_id+event_id)
- to (process_id+event_id)

attributes:
- process_id + event_id
- attribute_name
- attribute_value



Instrumentation schema // Examples

events:

```
sqlite> select * from be_tx limit 5;
time|pid|id|state
1589885834546127076|30|2131|prepare
1589885834546138330|30|2131|opening
1589885834546176963|30|2131|grouping
1589885834546605732|30|2131|active
1589885834552517436|30|2131|closed
```

relations:

```
sqlite> select * from fom_to_tx limit 3;
id|pid|fom_id|tx_id
1|30|7499|7501
2|30|7522|7524
3|30|7567|7569
```

attributes:

```
sqlite> select * from attr where name like "%TX_%" limit 3;
id|entity_id|pid|name|val
16|2131|30|M0_AVI_BE_TX_ATTR_PAYLOAD_NOB|0
17|2131|30|M0_AVI_BE_TX_ATTR_PAYLOAD_PREP|0
18|2131|30|M0_AVI_BE_TX_ATTR_RA_AREA_USED|2176
```

- A quite compact instrumentation:

```
$ git grep ADDB2_ADD | grep -v addb2/ | wc -l
  97
```

- A single aggregation place in the code:

```
addb2/dump.c:
struct m0_addb2__id_intrp ids[] = {
     { M0_AVI_NULL,            "null" },
     { M0_AVI_NODE,            "node",            { FID
}},       { M0_AVI_LOCALITY,        "locality",        {
&dec }},
...
```

Source/Footnote:

Instrumentation schema // Examples

events:

```
* 2020-11-11-07:43:09.374671190 tx-state          sm_id: 3440
prepare -[opening]-> opening
|         node             <279b4a1a242c11eb:85c4566f20190800>
|         pid              26596 |          locality         0
|         fom              addr: @0x7f25680a0a80, 'cas-fom'
transitions: 0 phase: init
* 2020-11-11-07:43:09.374856392 tx-state          sm_id: 3440
opening -[grouping]-> grouping
|         node             <279b4a1a242c11eb:85c4566f20190800>
|         pid              26596 |          locality         0
|         ast
* 2020-11-11-07:43:09.374917970 tx-state          sm_id: 3440
grouping -[activated]-> active
|         node             <279b4a1a242c11eb:85c4566f20190800>
|         pid              26596 |          locality         0
|         ast
```

relations:

```
* 2020-11-11-08:01:05.832491882 fom-to-tx         fom_id: 5448,
tx_id: 5452
|         node             <279b4a1a242c11eb:85c4566f20190800>
|         pid              26596 |          locality         0
|         fom              addr: @0x7f25680ab7a0, 'COB
create/delete/getattr' transitions: 0 phase: init
* 2020-11-11-08:01:05.833888977 fom-to-tx         fom_id: 5468,
tx_id: 5472
|         node             <279b4a1a242c11eb:85c4566f20190800>
|         pid              26596 |          locality         0
|         fom              addr: @0x7f25680afdf0, 'COB
create/delete/getattr' transitions: 0 phase: init
```

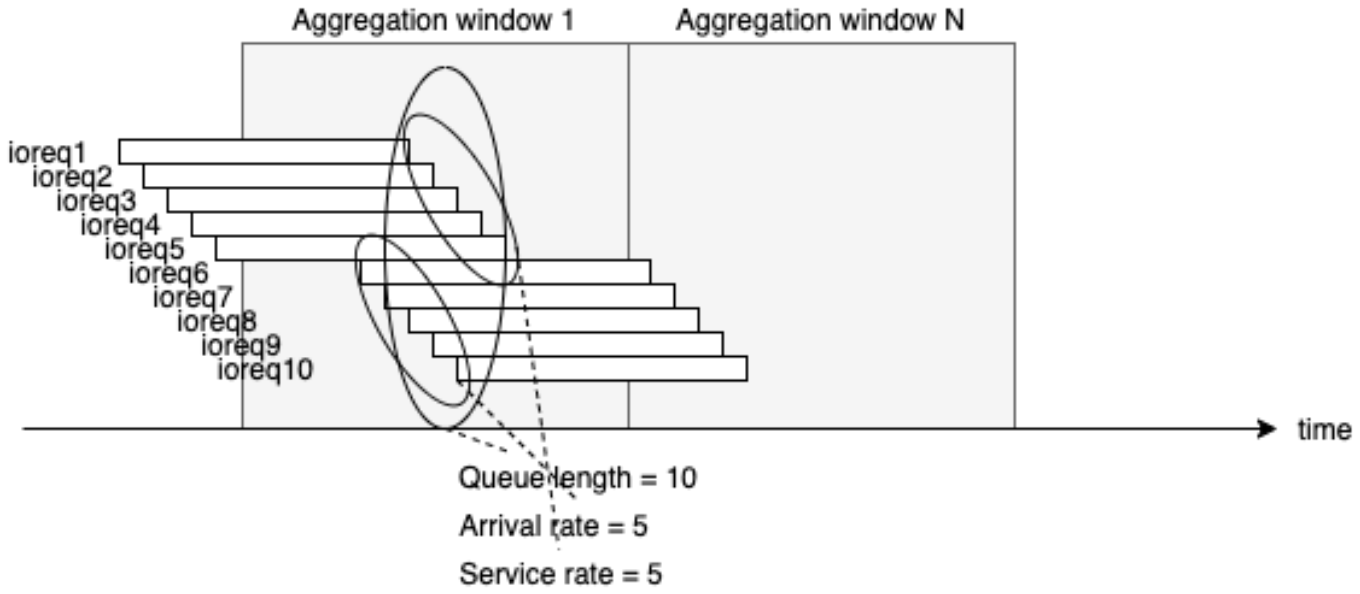
attributes:

```
* 2020-11-11-08:01:05.832917245 attr              entity_id:
5452, M0_AVI_BE_TX_ATTR_PAYLOAD_NOB:
284 |          node             <279b4a1a242c11eb:85c4566f201
90800> |          pid              26596
|         locality         0
|         fom              addr: @0x7f25680ab7a0, 'COB
create/delete/getattr' transitions: 9 phase: tx_wait
* 2020-11-11-08:01:05.832917801 attr              entity_id:
5452, M0_AVI_BE_TX_ATTR_PAYLOAD_PREP:
131072 |          node             <279b4a1a242c11eb:85c4566f
20190800> |          pid              26596
|         locality         0
|         fom              addr: @0x7f25680ab7a0, 'COB
create/delete/getattr' transitions: 9 phase: tx_wait
```


Other metrics

Kendall's notation

- A: The arrival process
- S: The service time distribution
- c: The number of servers
- K: The number of places in the queue
- N: The calling population
- D: The queue's discipline



More practical metrics

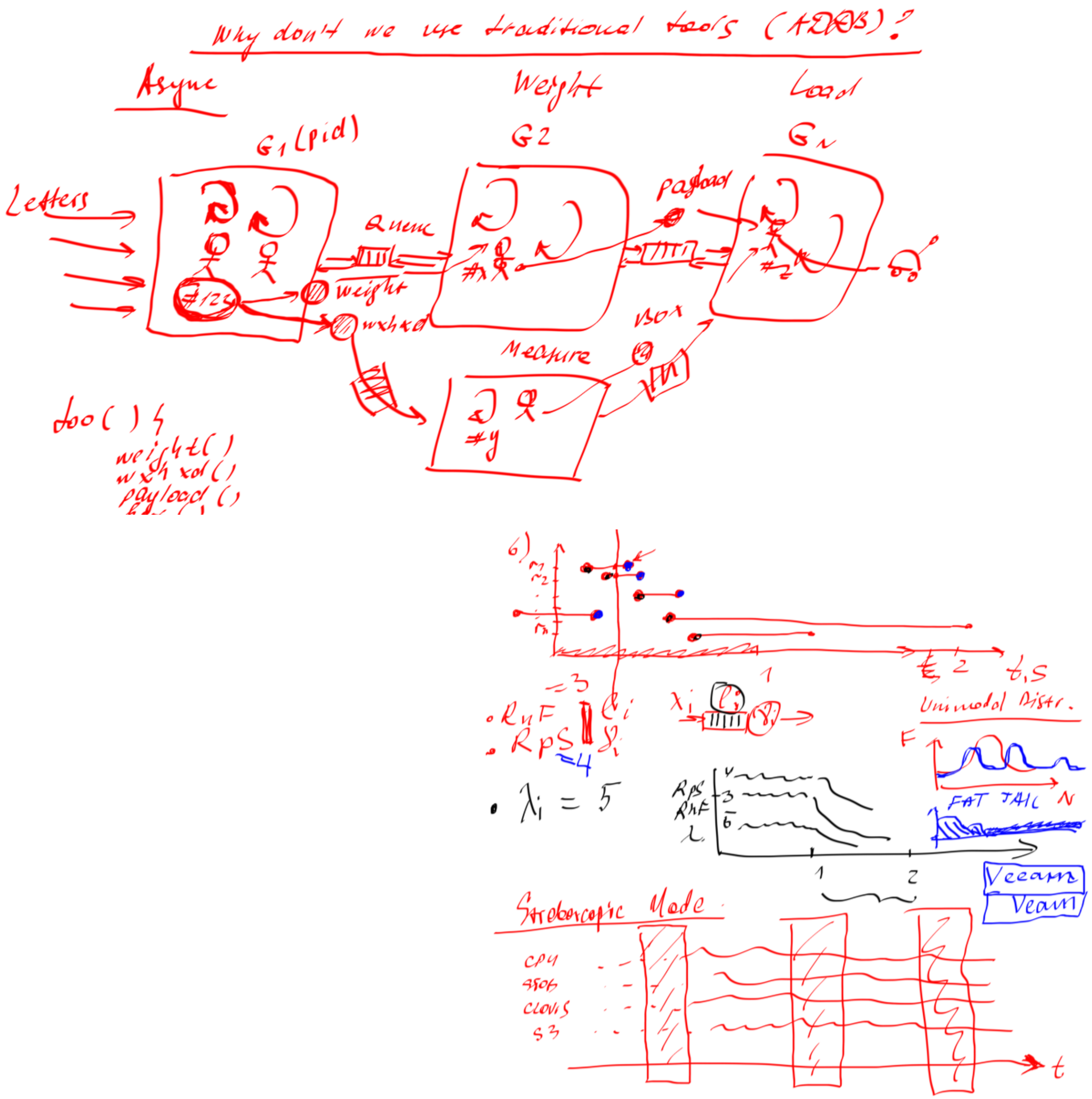
- λ: arraival rate
- l: average queue length (RnF)
- μ: average service time (RPS)

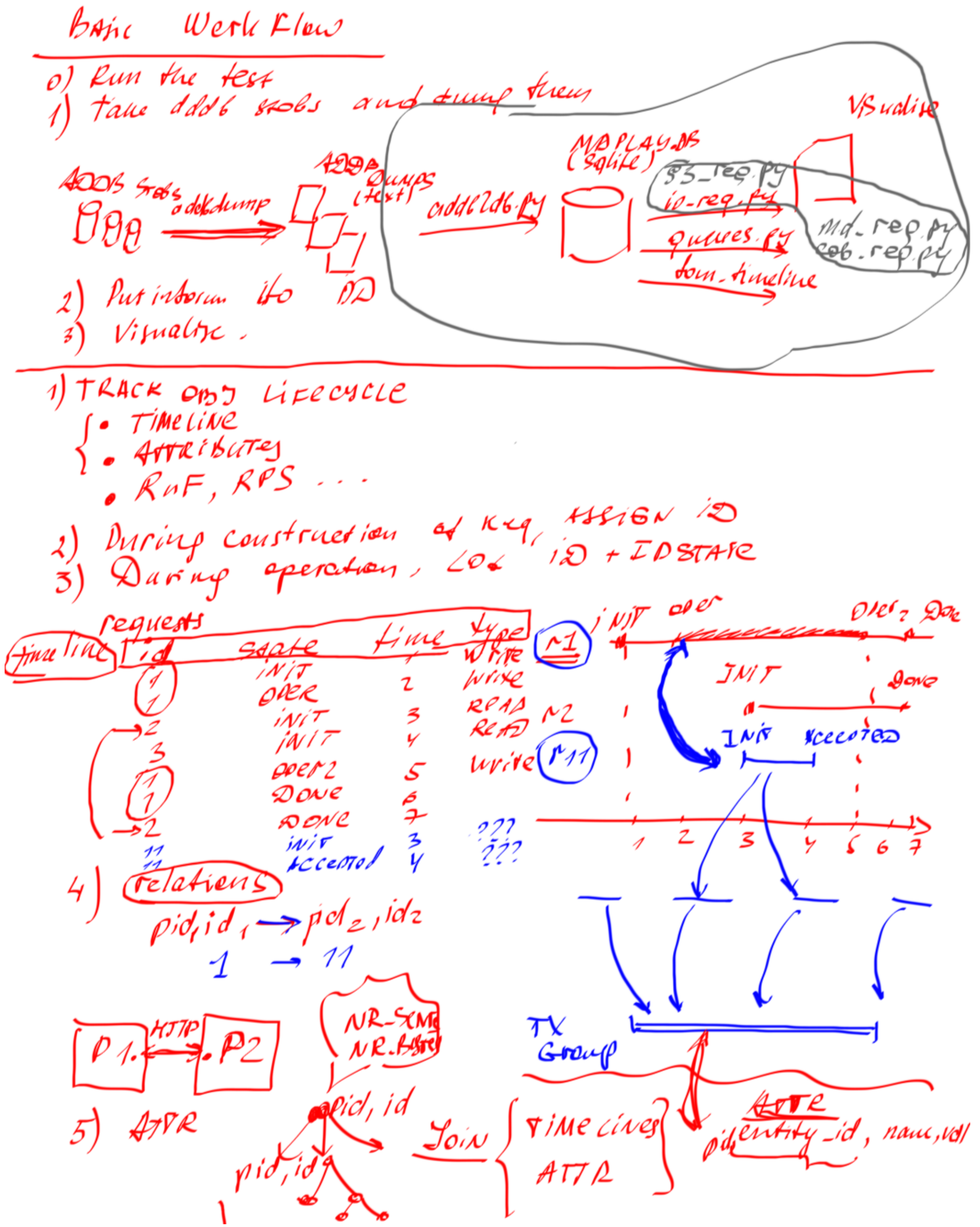
Source/Footnote: https://en.wikipedia.org/wiki/Kendall%27s_notation

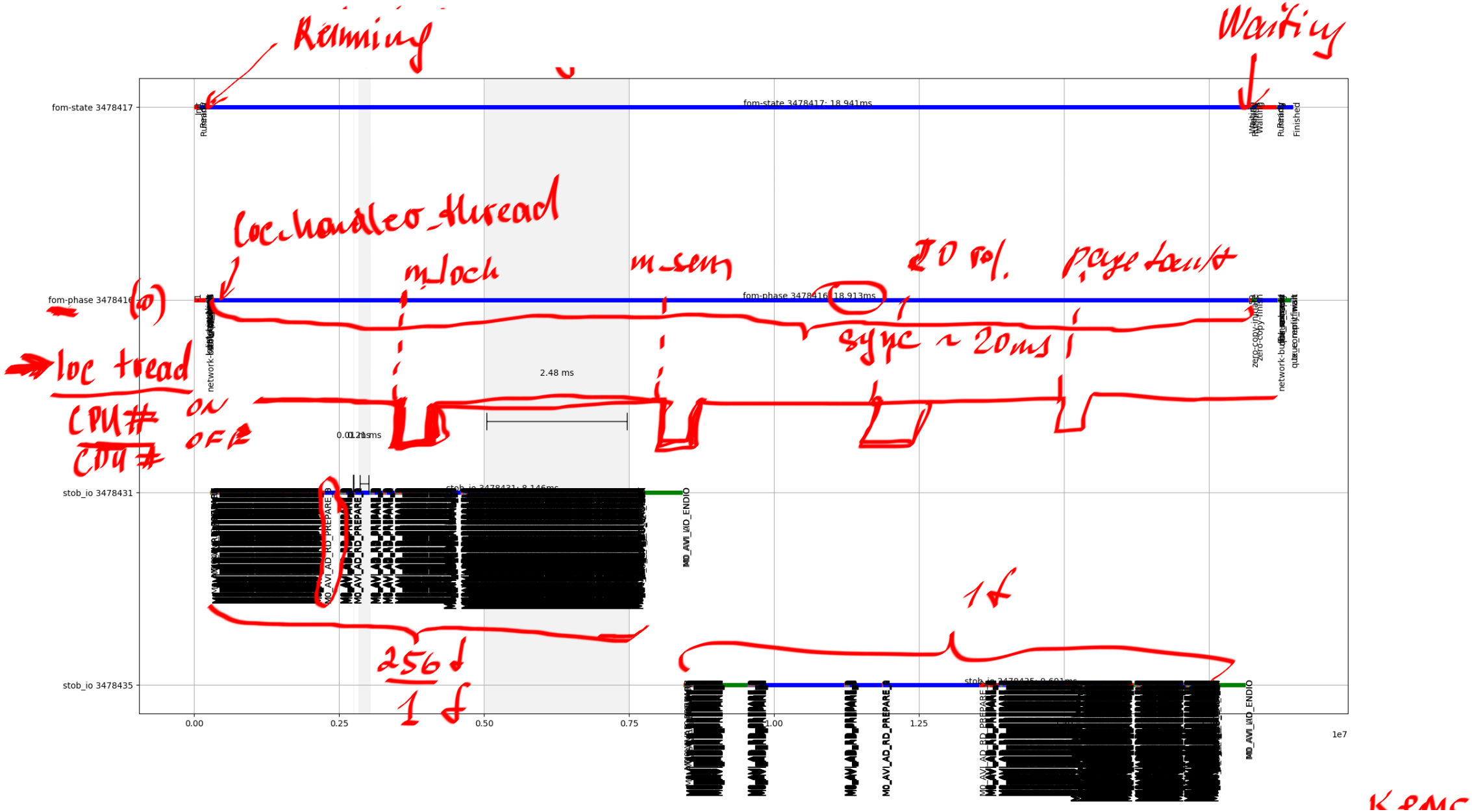
Demo

Questions?

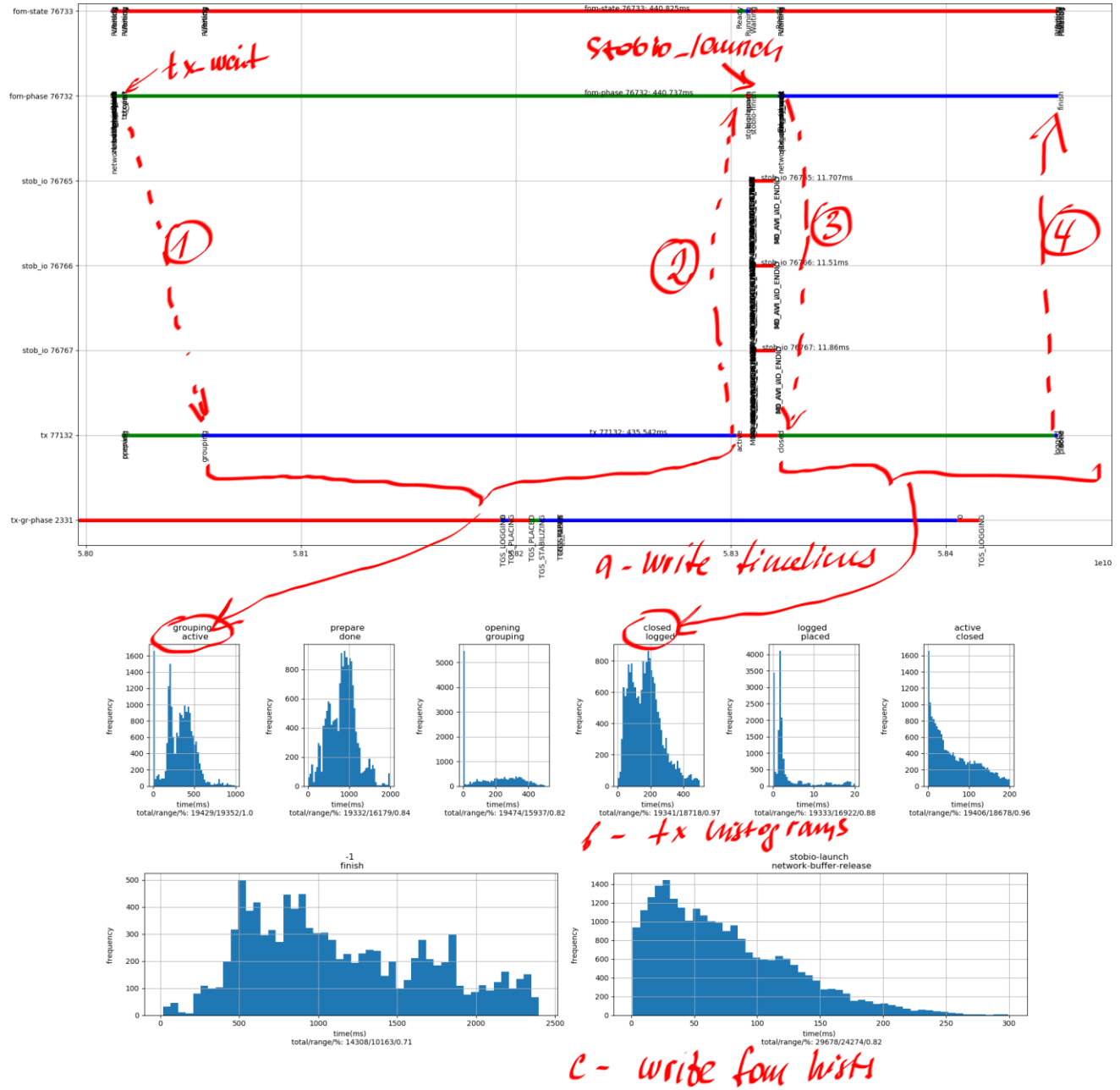
Other notes


Why don't we use traditional tools (ADDB)?
Async
Weight
Load
Letters
Queue
payload
weight
measure
foo( ) {
weight( )
payload( )
RuF
RPS
Unimodal Distr.
FAT TAIL
Stroboscopic Mode
CPU
clovis
S3
t

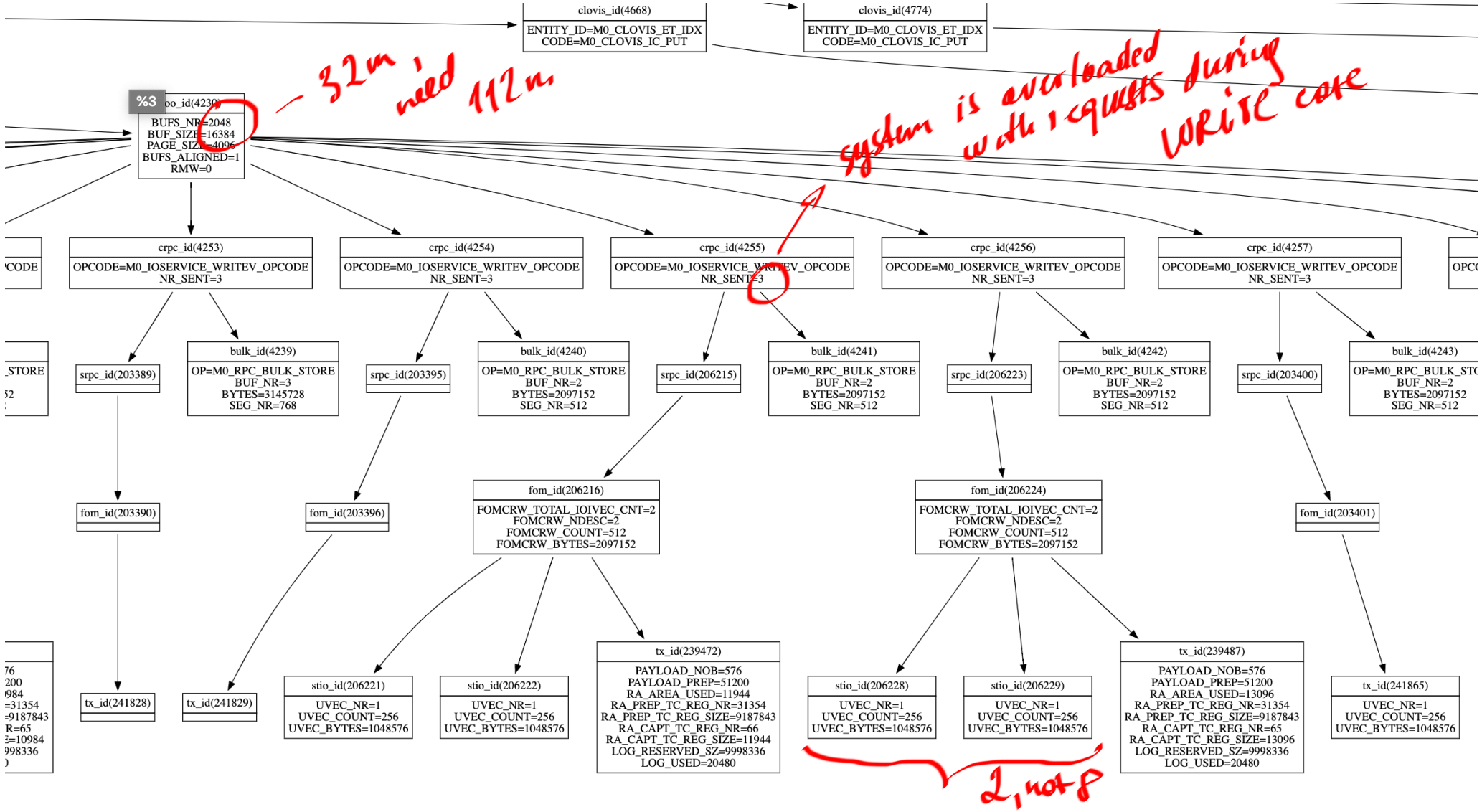

Basic Work Flow
0) Run the test
1) Take addb stobs and dump them
Visualize
ADDB Stobs
ADDB Dumps (text)
addb2db.py
queries.py
join_timeline
2) Put inform. into DB
3) Visualize.
1) TRACK OBJ LIFECYCLE
• TIMELINE
• ATTRIBUTES
• RuF, RPS ...
2) During construction of req, ASSIGN ID
3) During operation, LOG ID + IDSTATE
requests
timeline
id
state
time
type
INIT
OPER
DONE
WRITE
READ
ACCEPTED
4) Relations
pid1, id1 → pid2, id2
5) ATTR
P1.
HTTP
P2
Join
TIMELINES
ATTR
TX Group



Source/Footnote:



Source/Footnote:



Source/Footnote: