CORTX DTM, roadmap

~~or Crusoe's personal diary~~

Anatoliy Bilenko

DTM0, Episode I – The Phantom Menace

DTM0, Episode II – Attack of the Clones

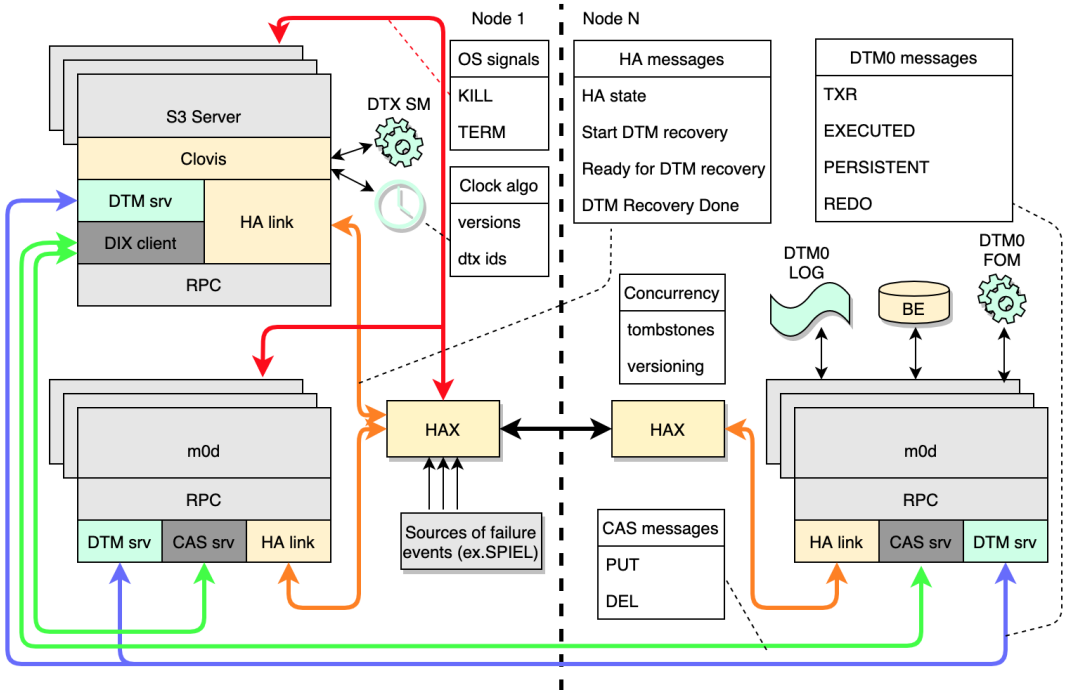
DTM0, Episode 100500 – A New Hope

DTM0, Use-case study

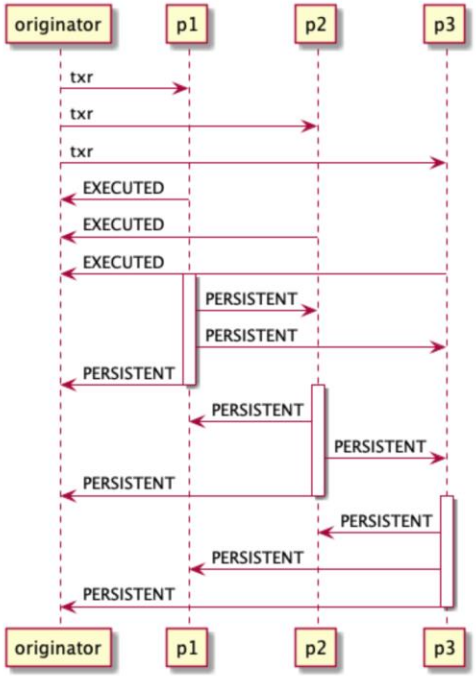
- Happy path
- 1 transient failure of persistent participant and subsequent DTM recovery
- 1 transient failure of persistent participant and subsequent another 1 subsequent failure during recovery
- N transient failures of originators
- N transient failures of originators and k failures of persistent participants
- DTM recovery stop
- DTM during persistent failures
- Client interface w.r.t. distributed transactions
- HA callbacks
- Metadata update
- S3 interraction
- long living tombstones and log records
- m0d recovery with sequential ongoing IO from one client
- m0d recovery without ongoing IO with a single client
- m0d recovery with two clients and ongoing independent IO
- DTM Log truncation
- death of originator
- Dependency tracking
- DTM message delivery during failures
- DTM transaction and record versions
- clock synchronisation
- clock desynchronization
- failure model
- recovery: redo+put/del execution, ordering
- how long with 3-way replication 3 participant can be in transient?
- new originator after fail
- how the states of participants are propagated the dtm logic (confd)
- clean startup
- recovering startup
- Cluster start usecase after shutdown
- Cluster shutdown

DTM0, Significant moving parts of the design (1)

Component landscape



Happy path protocol



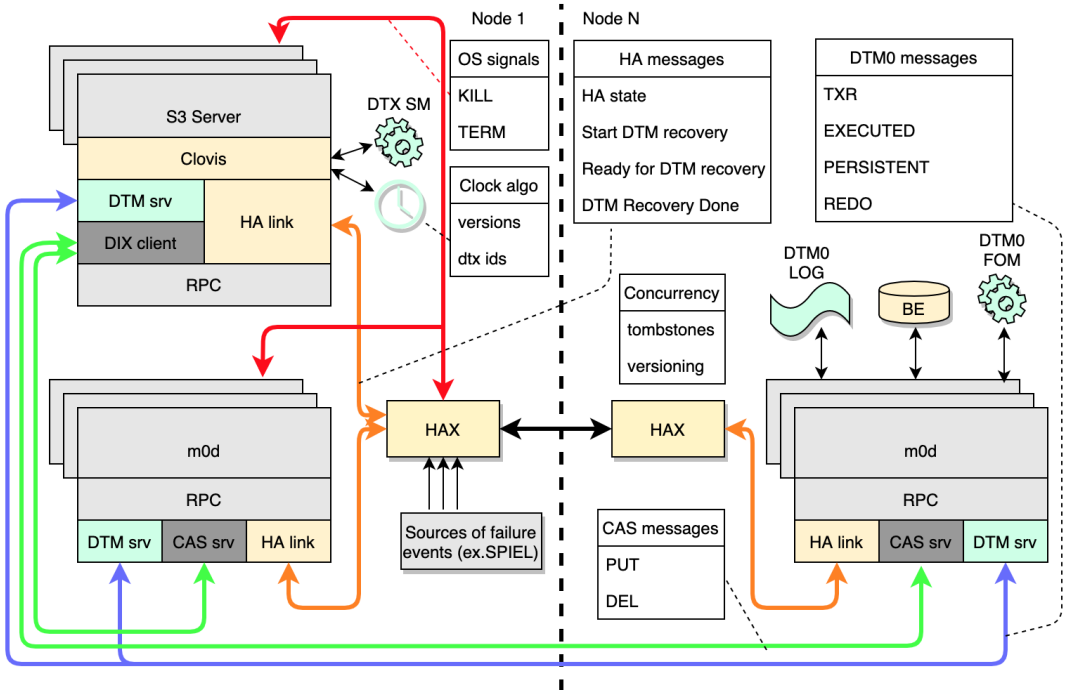

```
on txr receipt
if (!already_executed(txr)) {
    tx_open(be_tx);
    result = execute(be_tx, txr);
    txr.state[self] = EXECUTED;
    log(be_tx, txr);
    tx_close();
}
sender.send(EXECUTED, txr.id,
result);
```

```
on commit (local transaction
    containing txr is logged):
for (process in txr.participants)
    { /* including self */
     /* send with retries until
    reply or receiver failure. */
     process.send(PERSISTENT,
    txr.id, self);
}
```

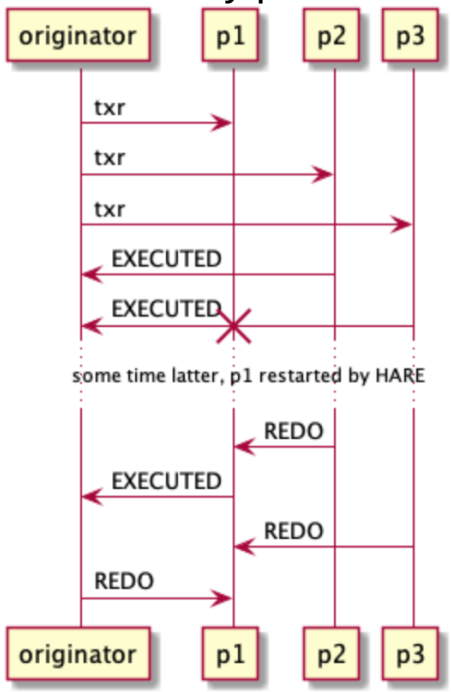
```
on PERSISTENT(txid, process) receipt:
txr = log.find(txid);
tx_open(be_tx);
txr.state[process] = PERSISTENT;
/* +1 to account for txr.state[self] */
if (count(txr.state[], state == PERSISTENT) + 1 > K) {
    wakeup(tx); /* tx is STABLE */
} if (all(txr.state[], state == PERSISTENT ...)) {
    log.prune(txid); /* tx is DONE */
}
tx_close(be_tx);
```

DTM0, Significant moving parts of the design (2)

Component landscape



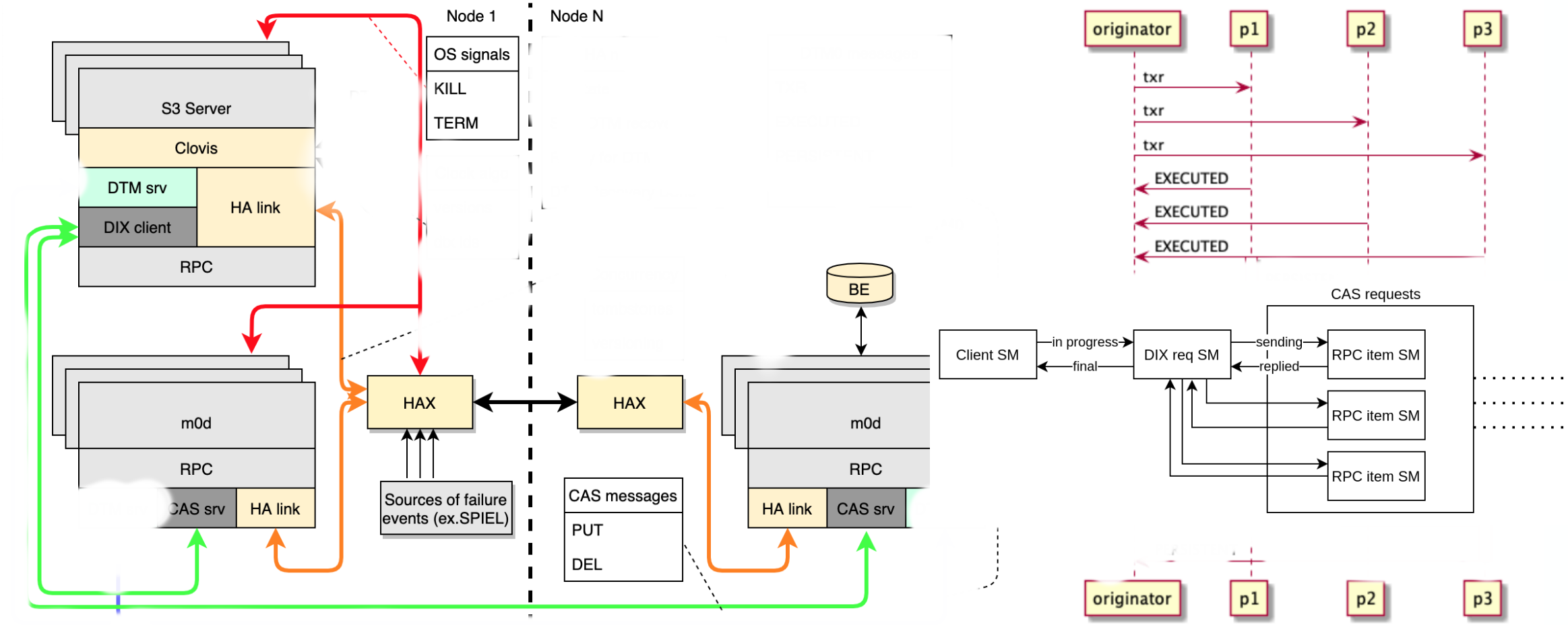
Recovery protocol



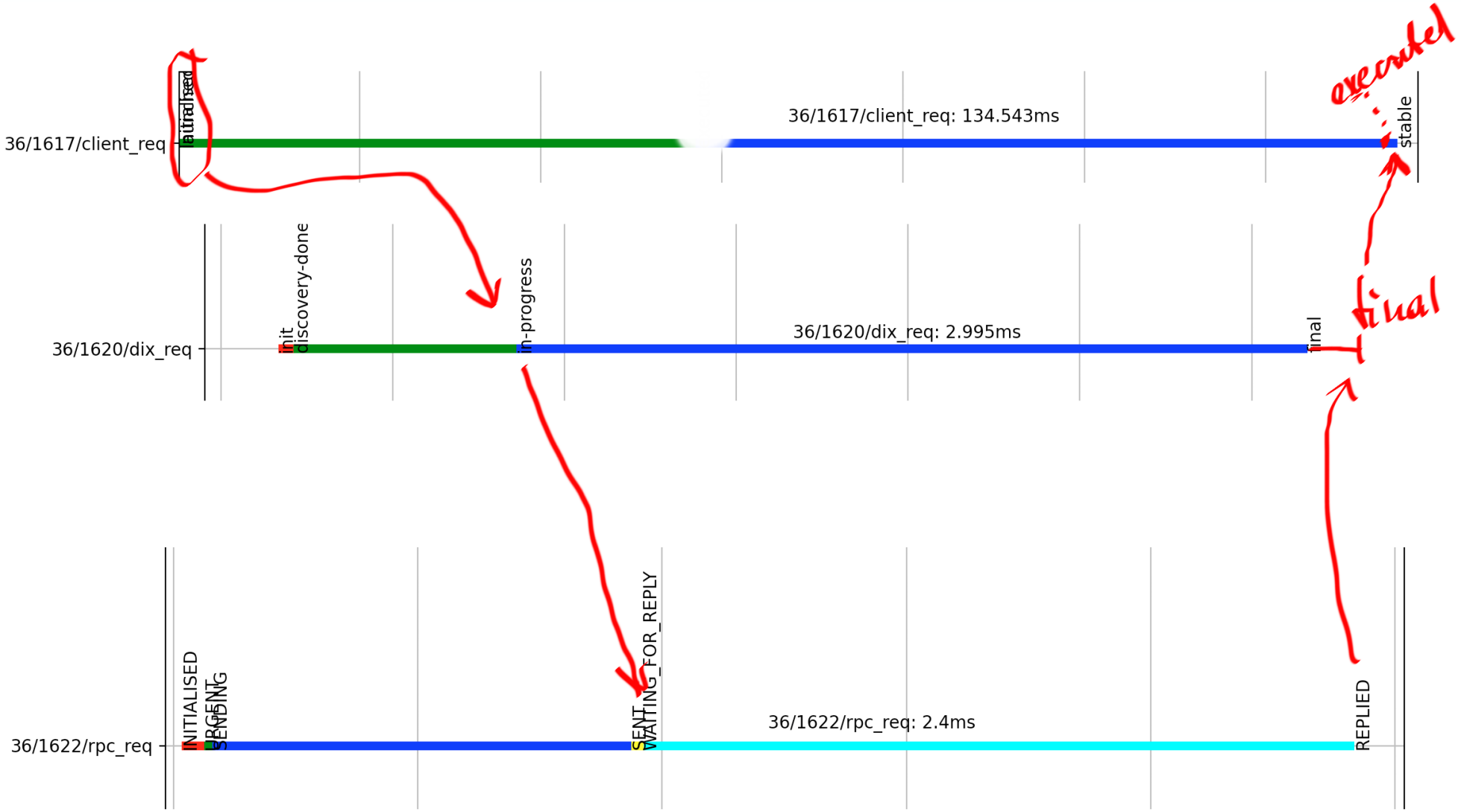

```
on HA.state(process, ONLINE) receipt:
for (txr in log) {
    if (process in txr.participants[] &&
   txr.state[process] < PERSISTENT) {
        process.send(REDO, txr);
    }
}
```

```
on HA.state(process, FAILED) receipt:
for (txr in log) {
    if (all(txr.state[], state == PERSISTENT ||
   process.state == FAILED)) {
        log.prune(txid); /* tx is DONE */
    }
}
```

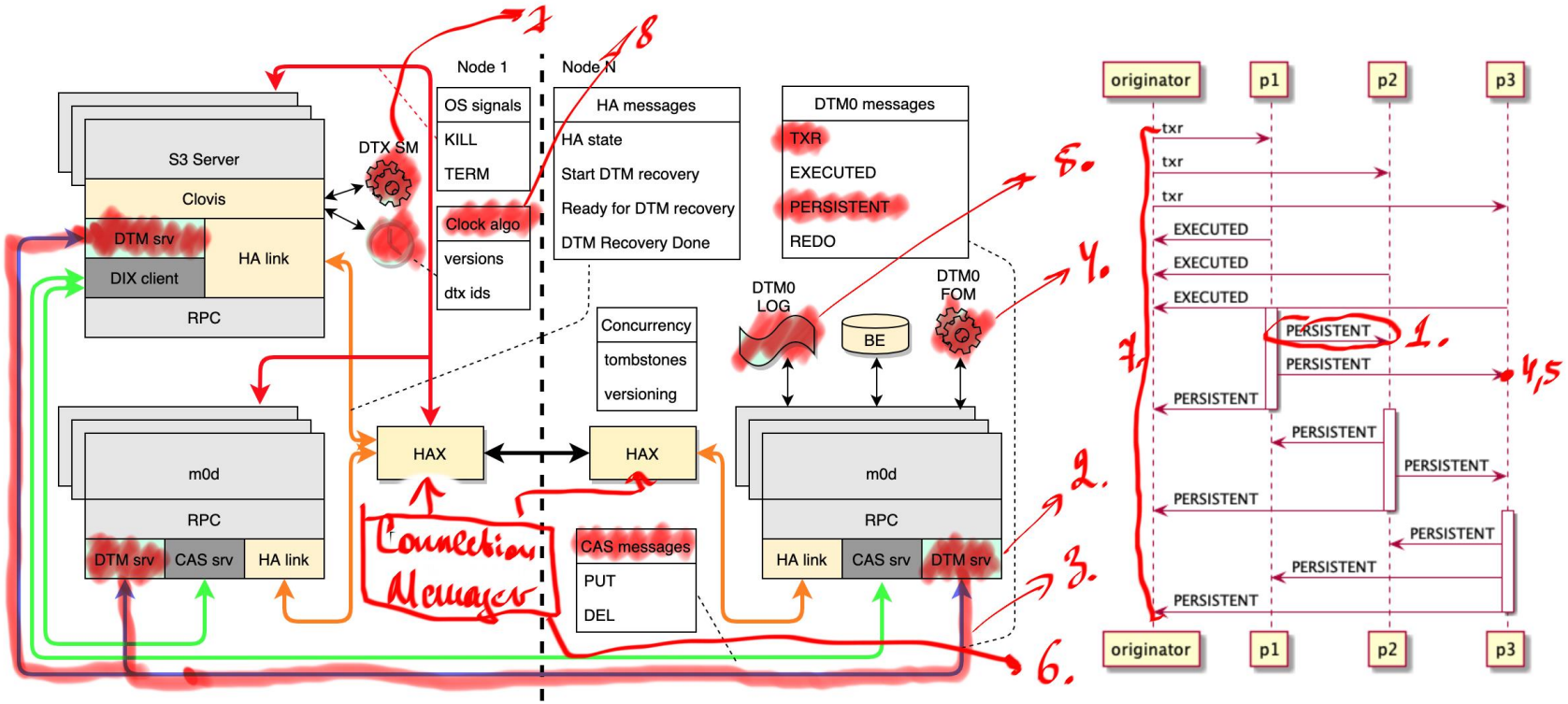
DTM0, What do we have, analysis (1)

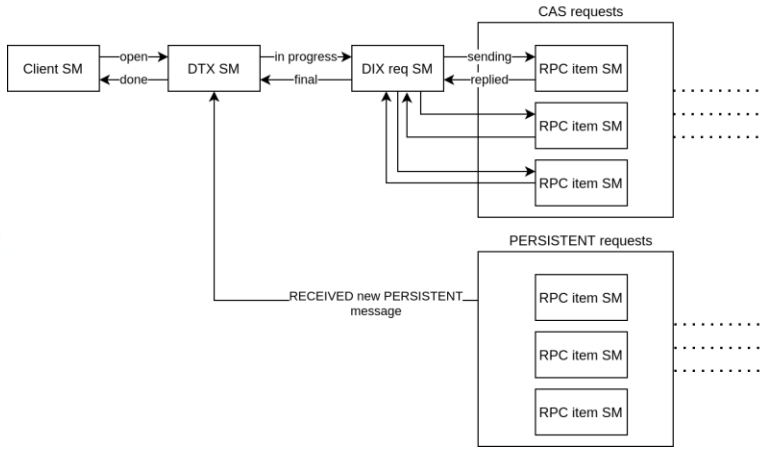


DTM0, What do we have, analysis (2)



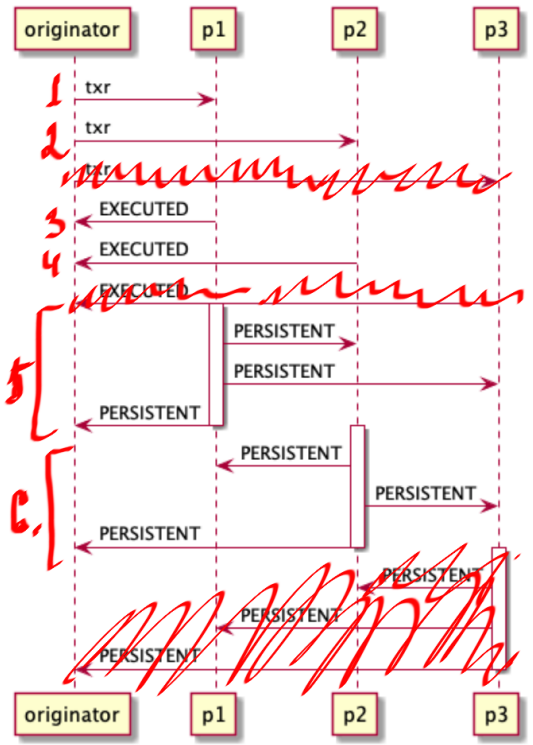
DTM0, What do we NEED to start? Analysis

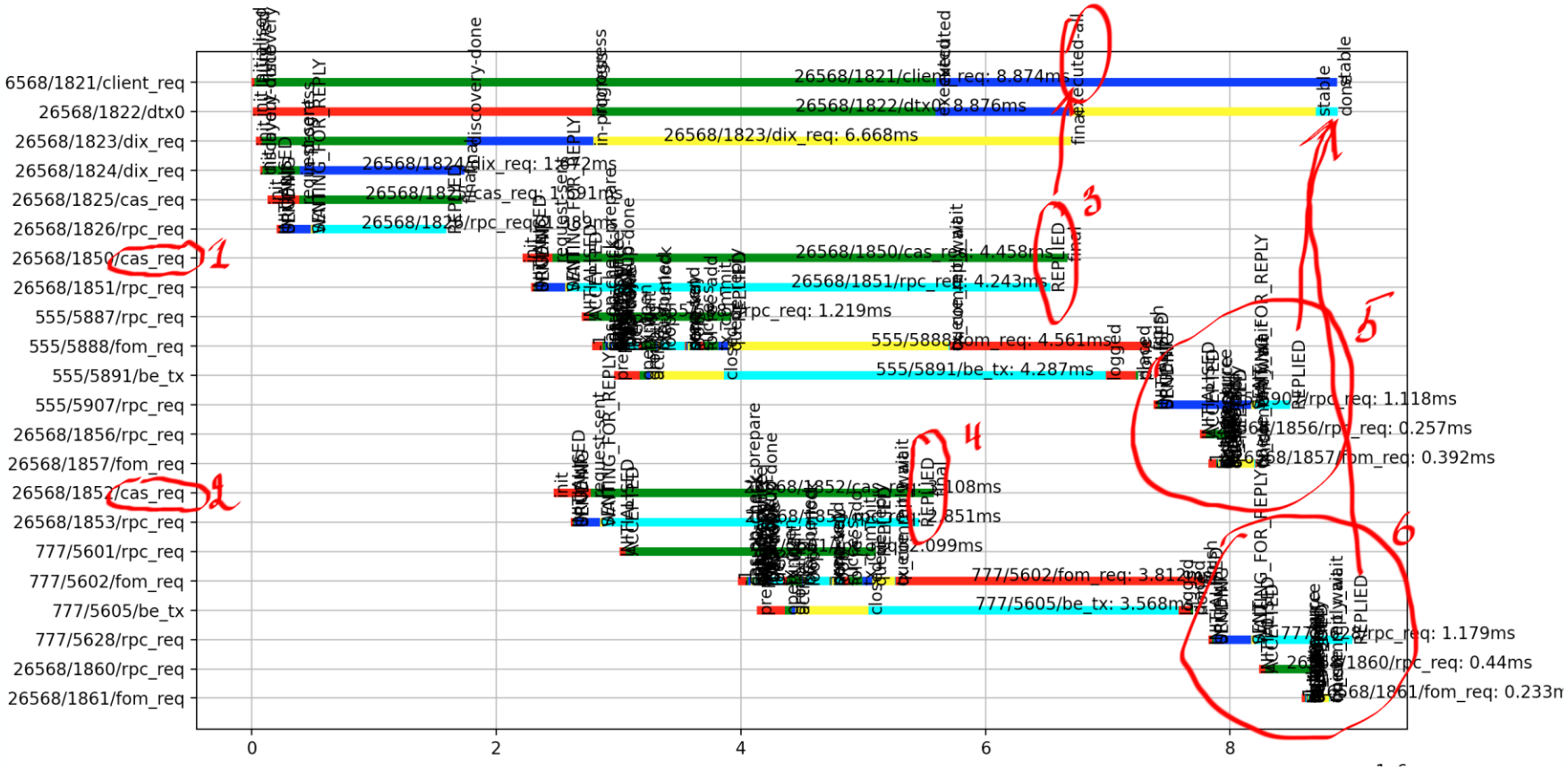




1. Introduce PERSISTENT messages
2. Implement new DTM service
3. Establish connections between DTM services
4. Implement PERSISTENT message processing logic
5. Introduce DTM log to store information needed for the recovery
6. Introduce connection manager kludge to connect DTM services as HARE is not yet fully functional
7. Introduce DTX state machine on the client side.
8. Introduce clock algorithm

DTM0, Happy path scenarios under the microscope





DTM0, One interesting story

- Start and stop scenario (getting rid of connection manager).
- There is a need to send DTM messages in face of failures to counterparts
- To send a message it's needed to establish connection
- There're two options:
  - #A: to establish/close connections on events like process is ONLINE or OFFLINE;
  - #B: to connect lazily on demand when the connection is really needed.

- Caveeat: if the HARE doesn't support EOS that some of related processes will not receive an appropriate amount of HA.ONLINE, HA.OFFLINE messages therefore will not be connected properly.

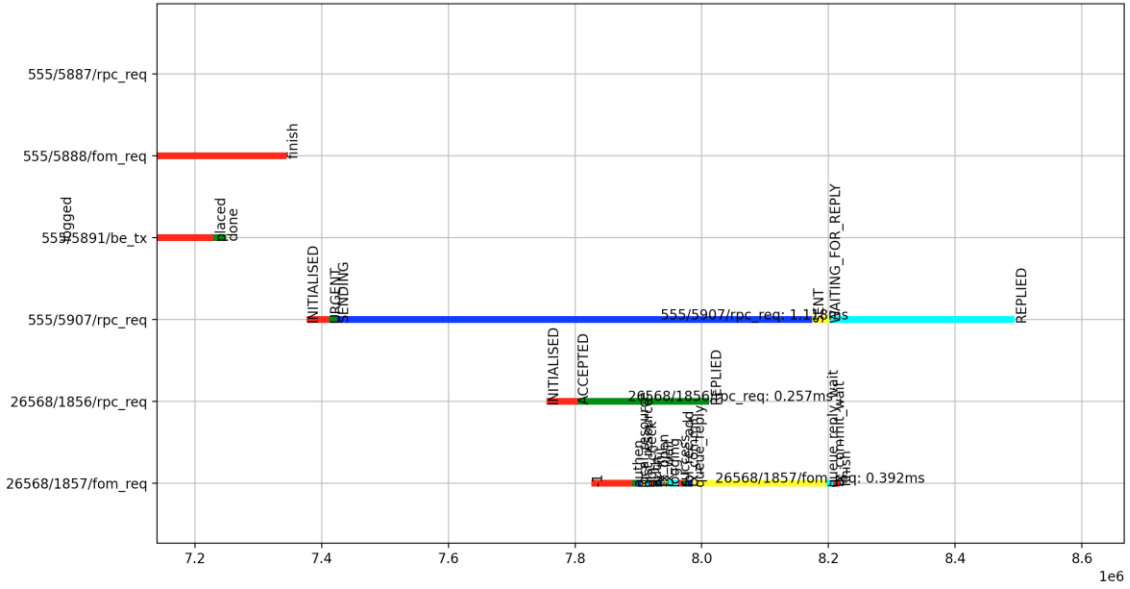
- Conclusion: chose option #B, define RPC failure model 

DTM0, RPC failure model

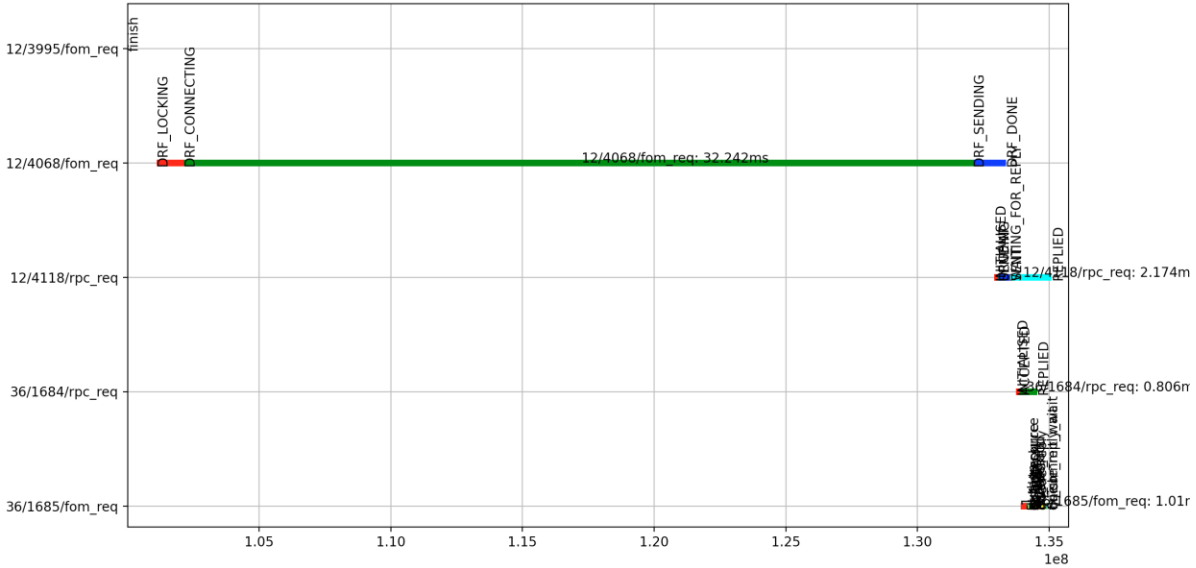
- TL;DL: "timeout is not a failure".
- Assumptions:
  - A message is either request or a reply or one-way.
  - Requests between a pair of processes are delivered in order.
  - Duplicate requests are suppressed (related to DTM0 messages and achieved employing versioning, proper implementation of tx_already_executed() and persistency capturing txr, reply, and other meaningful information).
  - ~~A message can be sent only once.~~
  - Timeout is not a failure and message send timeouts are infinite.
  - States of connections/sessions/messages are defined w.r.t. HA messages.
  - A message can be transmitted if the connection is ACTIVE.
  - HA never fails (if it fails, the process being under HA track never sees this and should be killed by HA or killed itself due to keep-alive algorithm logic).

DTM0, Finally, how does the PERSISTENT message look like?

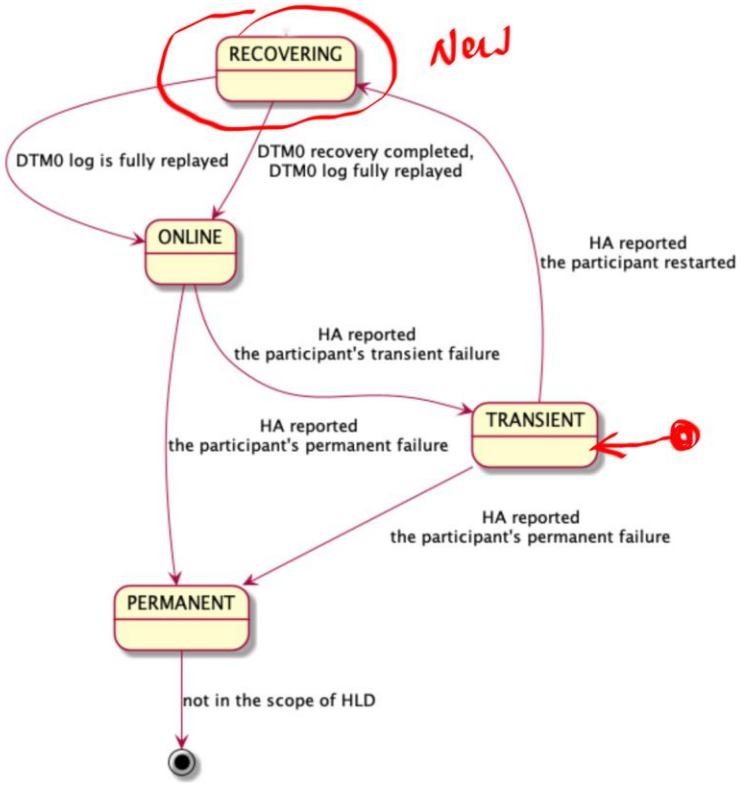
BEFORE

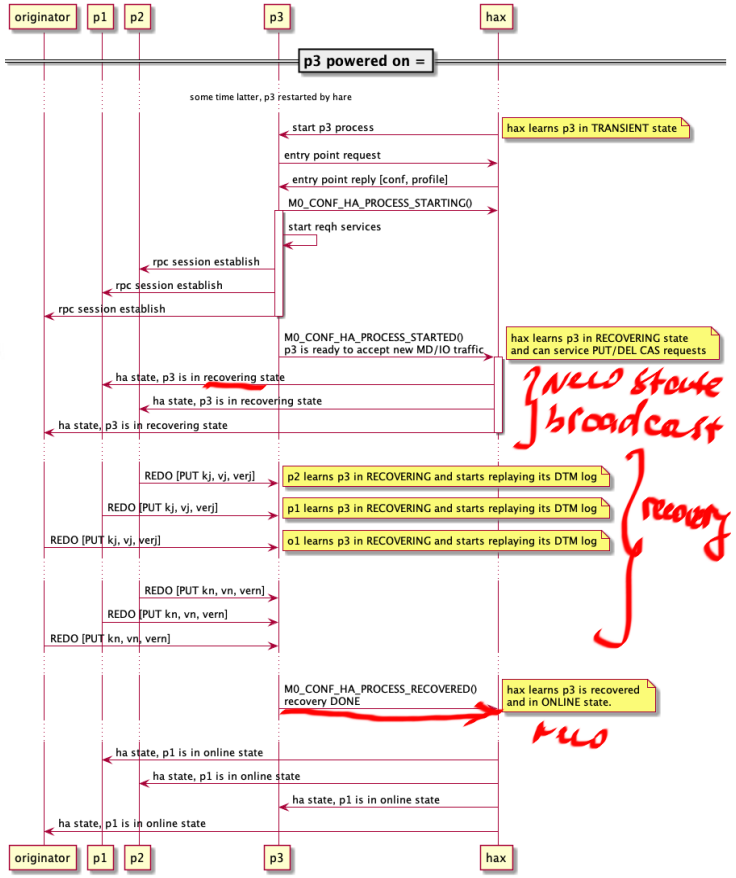


AFTER



DTM0, Recovery: we just started. What was identified?





- Add new RECOVERING state to all motr processes;
- Add new messages in MOTR<->HARE interaction;
- HARE need to suport EOS.

DTM0, Again, where we are?

- [DONE] DTM0 happy path scenario;
- [DONE] DTM0 start/stop scenarios;
- [IN PROGRESS] DTM0 simple recovery scenario;
- [TODO] Basic S3 integration;
- [TODO] DTM0 test;
- [TODO] DTM0 basic deployment;
- [TODO] (future) Metadata update;
- [TODO] (future) DTM0 for IO.

QUESTIONS?