motr

nikita.danilov@seagate.com

a scalable storage platform

Mero goals

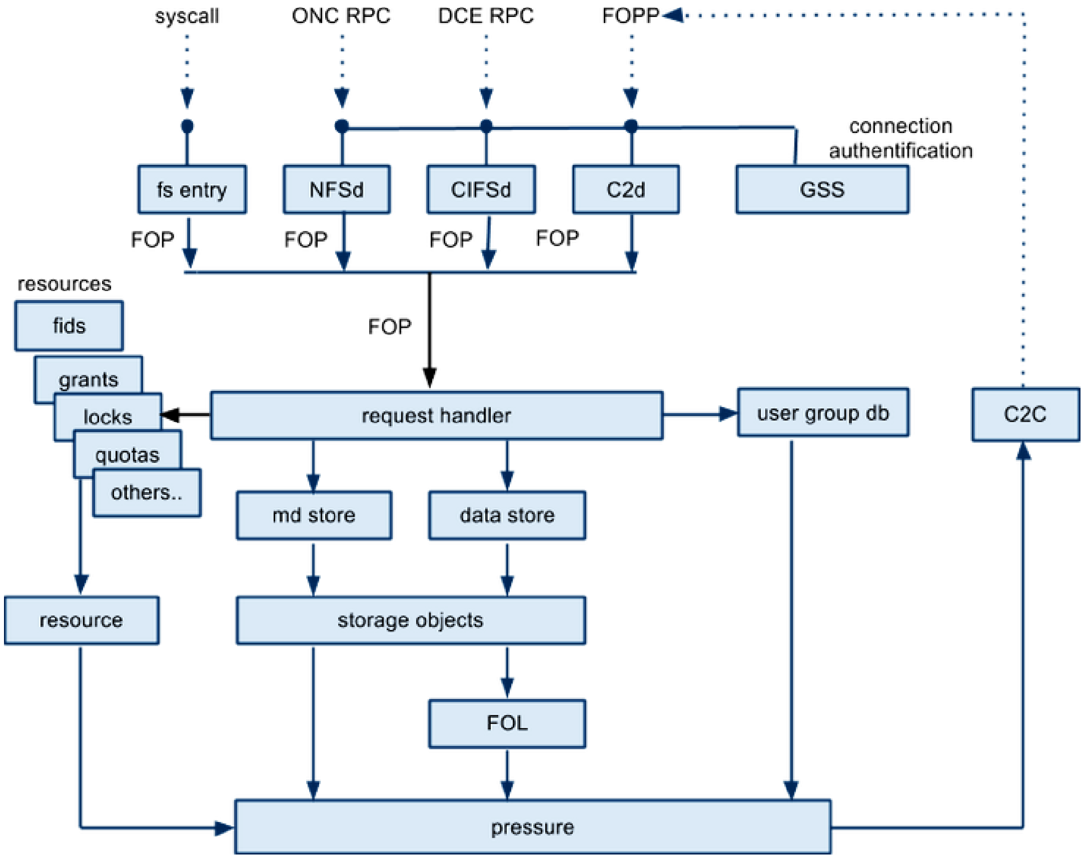
- extensible storage platform
  - observability
  - extensible scalable 3rd party plugins
- horizontally scalable (number of devices and nodes)
- vertically scalable (number of cores)
- flexible deployment

Warning: not everything in this presentation is fully implemented.

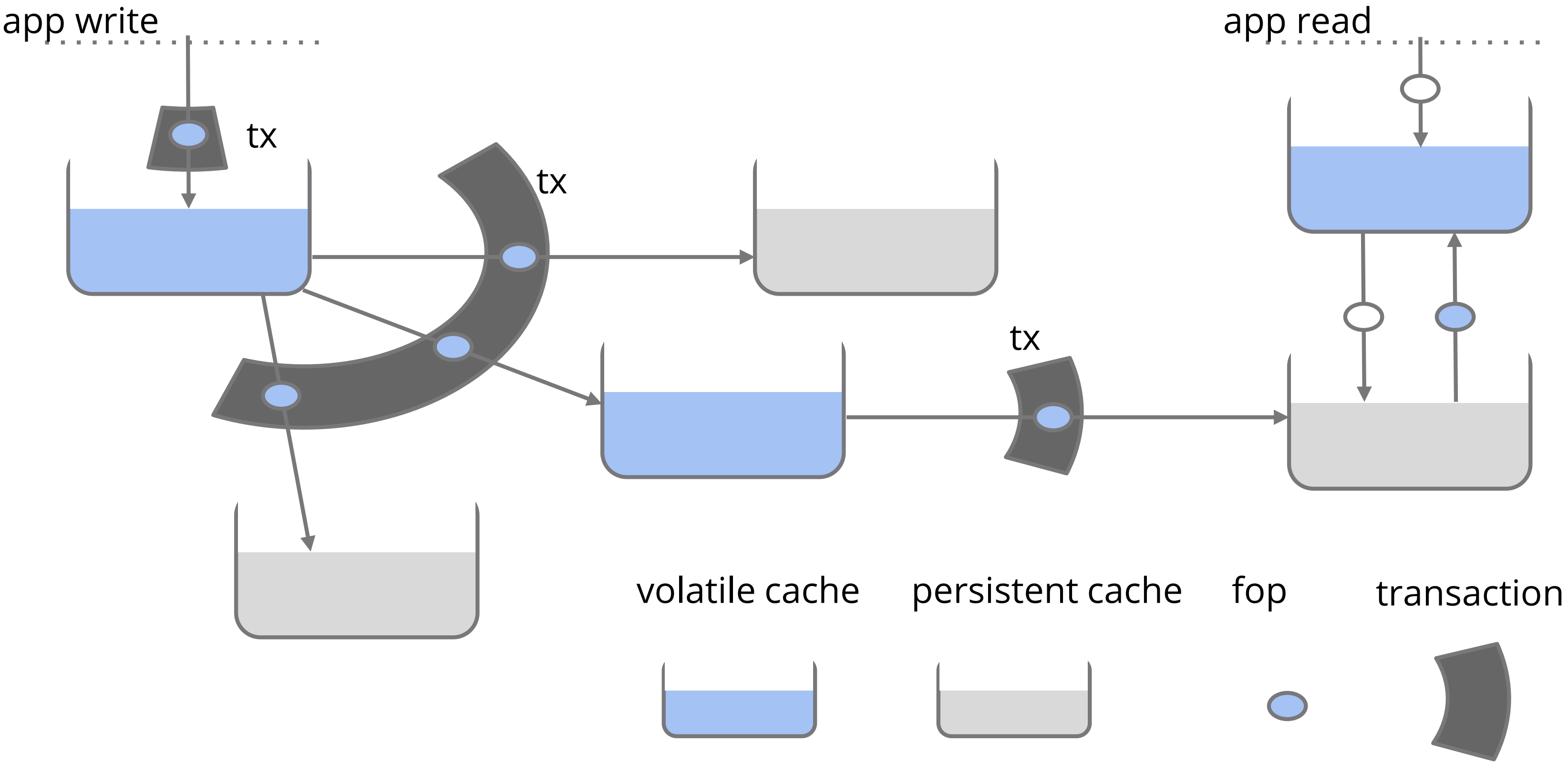
features

- 0-copy, 2-phase IO
- extensible meta-data: distributed key-value store
- layouts (data, meta-data, fault-tolerance)
  - network striping (parity de-clustering)
  - composite: NBA, snapshot, multi-tier
- distributed transactions (consistency)
- resource manager (coherency)
- containers, function shipping
- threadless server design
- fdmi, addb
- user space, portable

structure: instance (single node, process or kernel)



structure: system view



app write

tx

tx

tx

app read

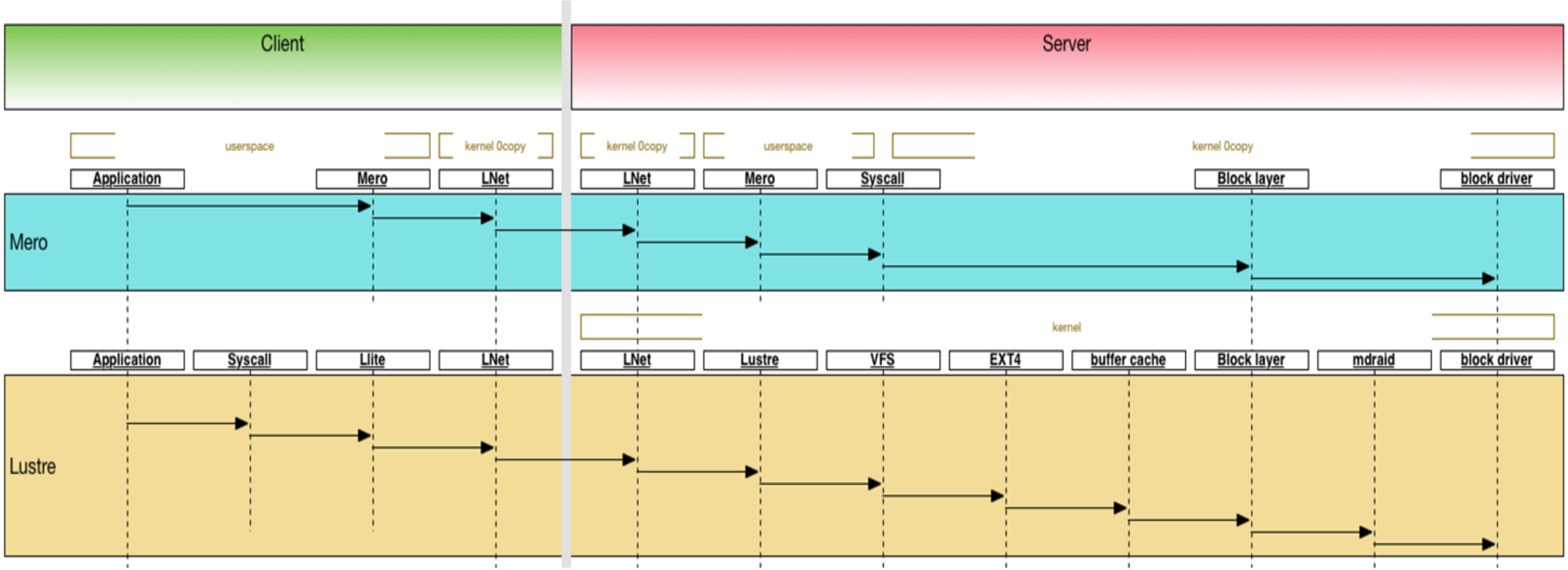
volatile cache

persistent cache

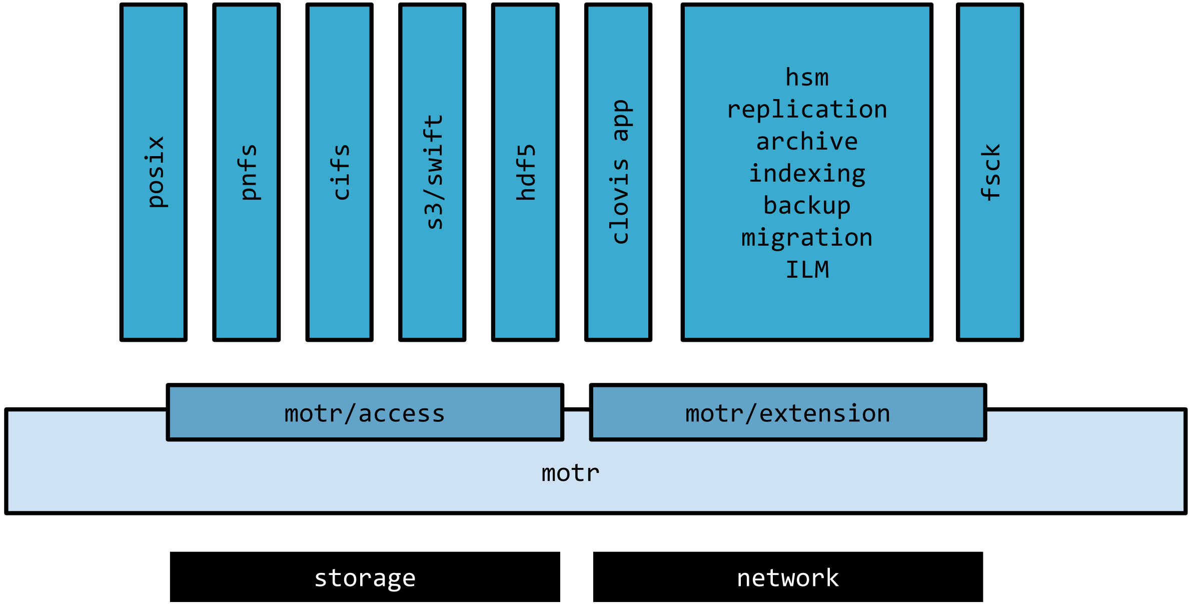
fop

transaction

structure: (non-)layering



structure: components



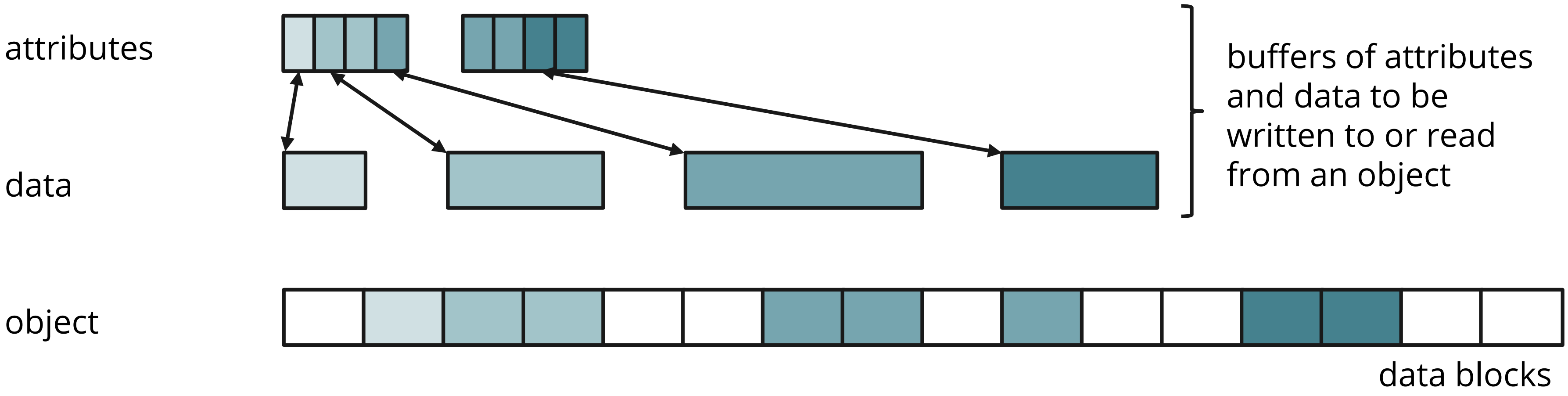
components

- **clovis**
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

clovis: overview

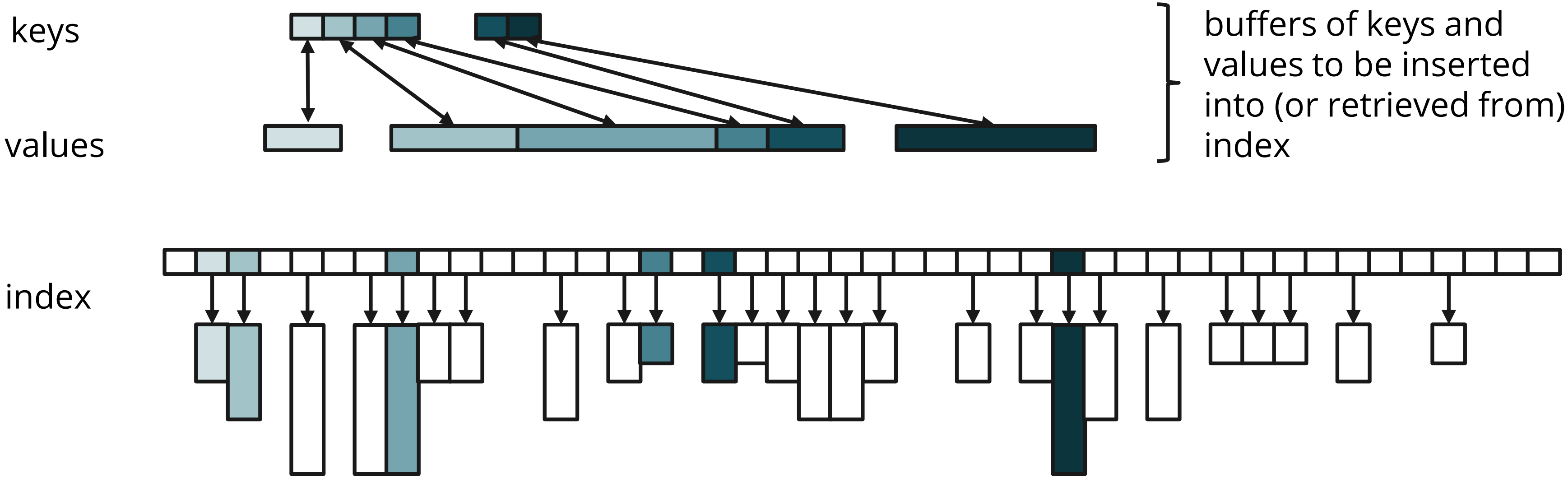
- object: network striped by default
- index: distributed key-value store
- operation: asynchronous, state machine, tracks progress
- transaction: atomic group of operations
- resource: ownership, coherent distributed caching
- layout: placement of data on storage
- 128-bit persistent identifiers, assigned by user
- all entry-points are asynchronous

clovis: object


attributes
data
object
buffers of attributes and data to be written to or read from an object
data blocks

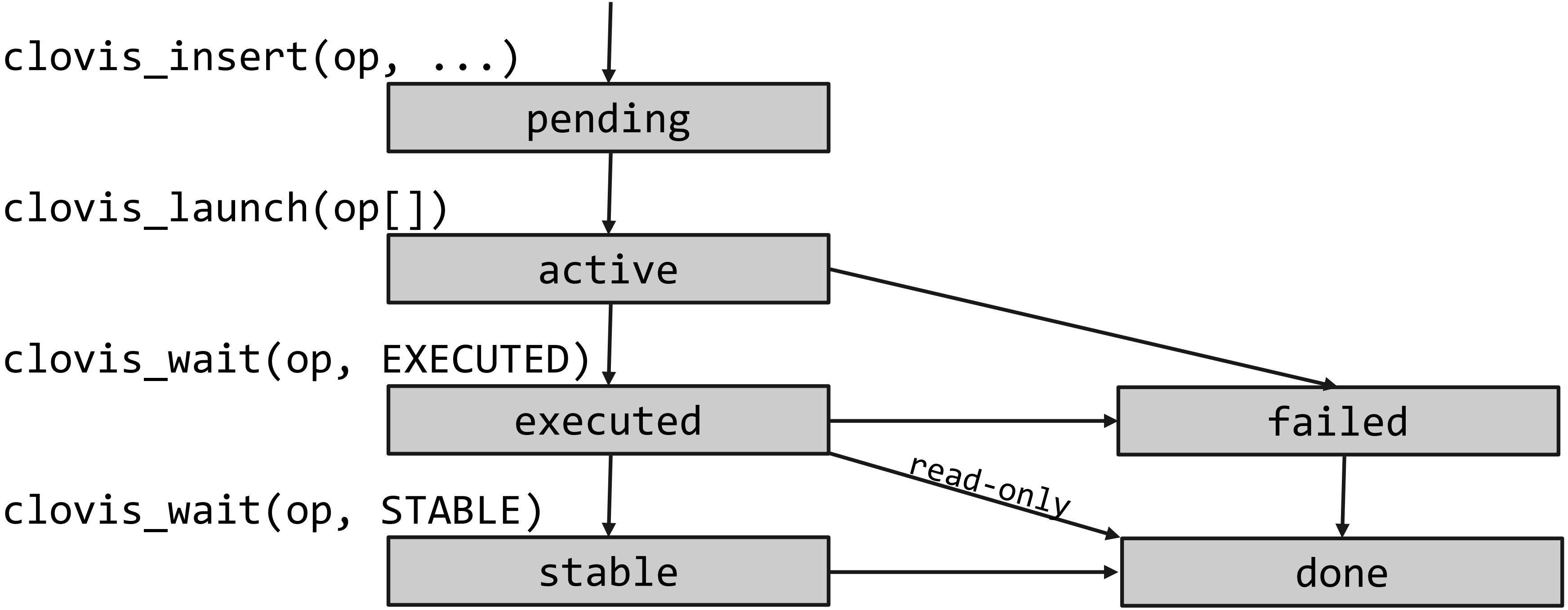
```
clovis_read(op, obj, data_buf_vec, attr_buf_vec, extent_vec)
clovis_write(op, obj, tx, data_buf_vec, attr_buf_vec, extent_vec)
clovis_alloc(op, obj, tx, extent_vec)
clovis_free(op, obj, tx, extent_vec)
```

clovis: index


keys
values
index
buffers of keys and values to be inserted into (or retrieved from) index

```
clovis_lookup(op, index, key_buf_vec, val_buf_vec)
clovis_insert(op, index, tx, key_buf_vec, val_buf_vec)
clovis_next(op, index, key_buf_vec, val_buf_vec)
```

clovis: operation



clovis_insert(op, ...)

clovis_launch(op[])

clovis_wait(op, EXECUTED)

clovis_wait(op, STABLE)

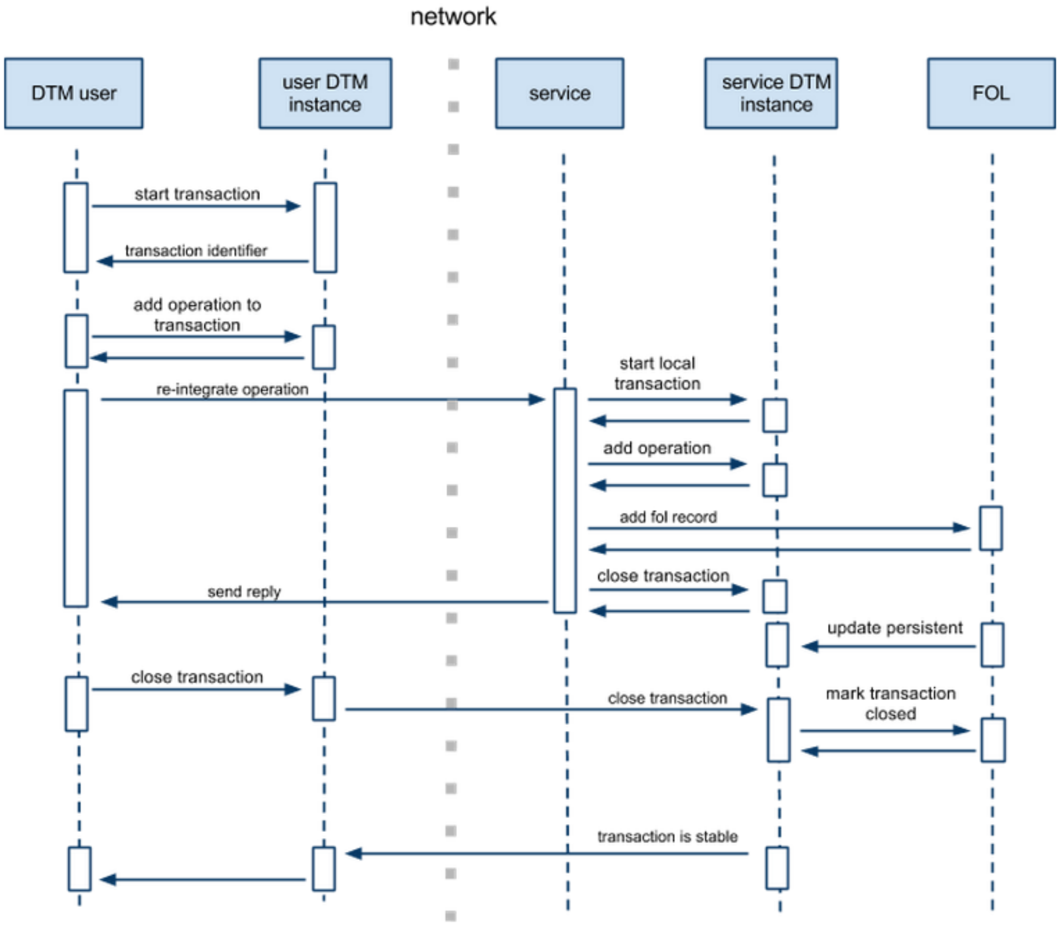
components

- clovis
- **transactions (dtm)**
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

dtm: transactions

- a transaction is a group of operations
- **a**tomicity w.r.t. certain failures (network partitions, node restarts)
- **c**onsistency is defined by user
- guarantees neither **i**solation nor serialisability
- doesn't guarantee synchronous **d**urability: too expensive for small transactions. Asynchronous stabilisation: the user is notified when the transaction becomes stable
- stabilisation ordering, liveness

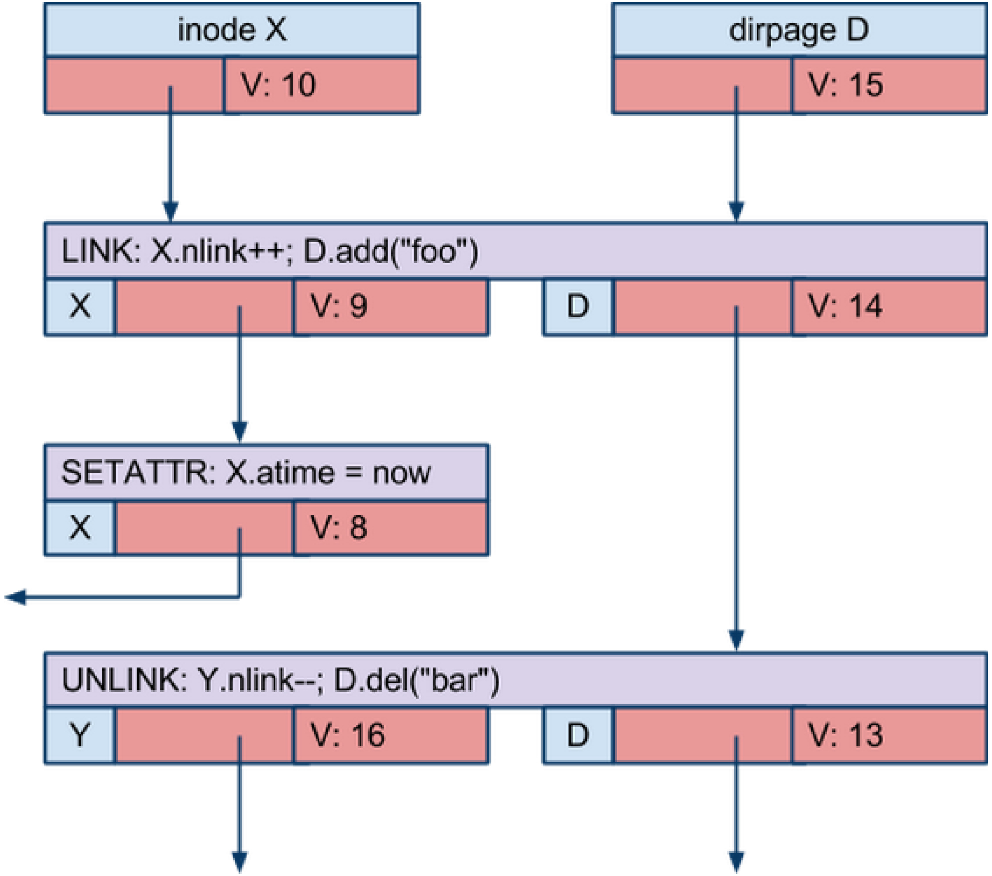
dtm: sequence



dtm: features

- masks certain transient failures
    - network partition, re-ordering, duplication
    - node restart
- write-ahead logging on each server
- undo for data
- redo for meta-data
- stabilisation: global logical clock (epochs)

dtm: versions



inode X

V: 10

dirpage D

V: 15

LINK: X.nlink++; D.add("foo")

X

V: 9

D

V: 14

SETATTR: X.atime = now

X

V: 8

UNLINK: Y.nlink--; D.del("bar")

Y

V: 16

D

V: 13

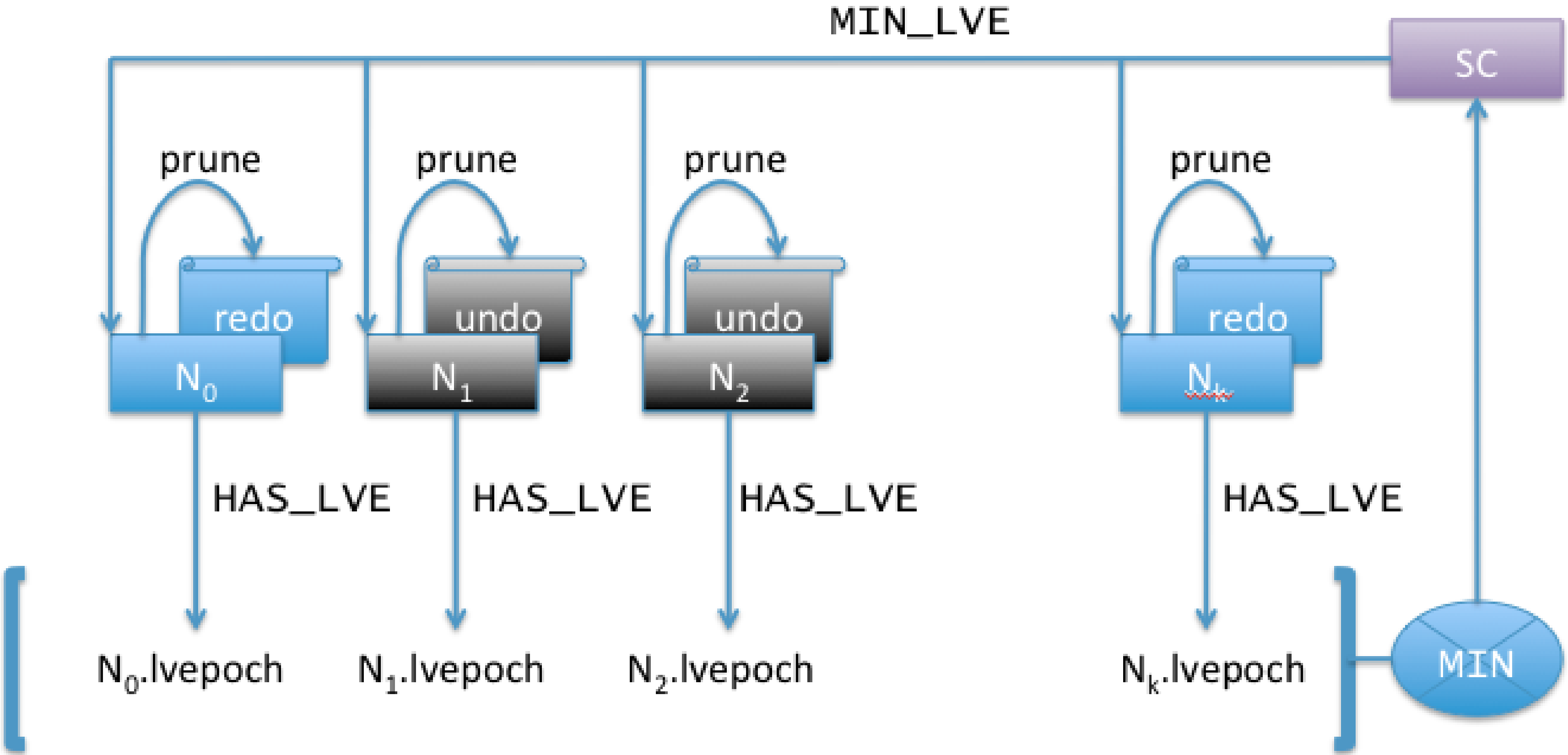
dtm: epochs

- distributed clock to detect operation dependency and ordering (epoch, Fidge-Mattern, Lamport)
- messages are tagged with the logical timestamps:

  `Event1` depends on `Event0`, then `Event0.epoch <= Event1.epoch`

- any node can advance its clock independently
- operations are kept in the persistent log in epoch order, until epoch is stable. Then the log is pruned
- global coordination to determine when an epoch is stable

dtm: epochs


MIN_LVE
SC
prune
prune
prune
prune
redo
undo
undo
redo
N_0
N_1
N_2
N_k
HAS_LVE
HAS_LVE
HAS_LVE
HAS_LVE
N_0.lvepoch
N_1.lvepoch
N_2.lvepoch
N_k.lvepoch
MIN

components

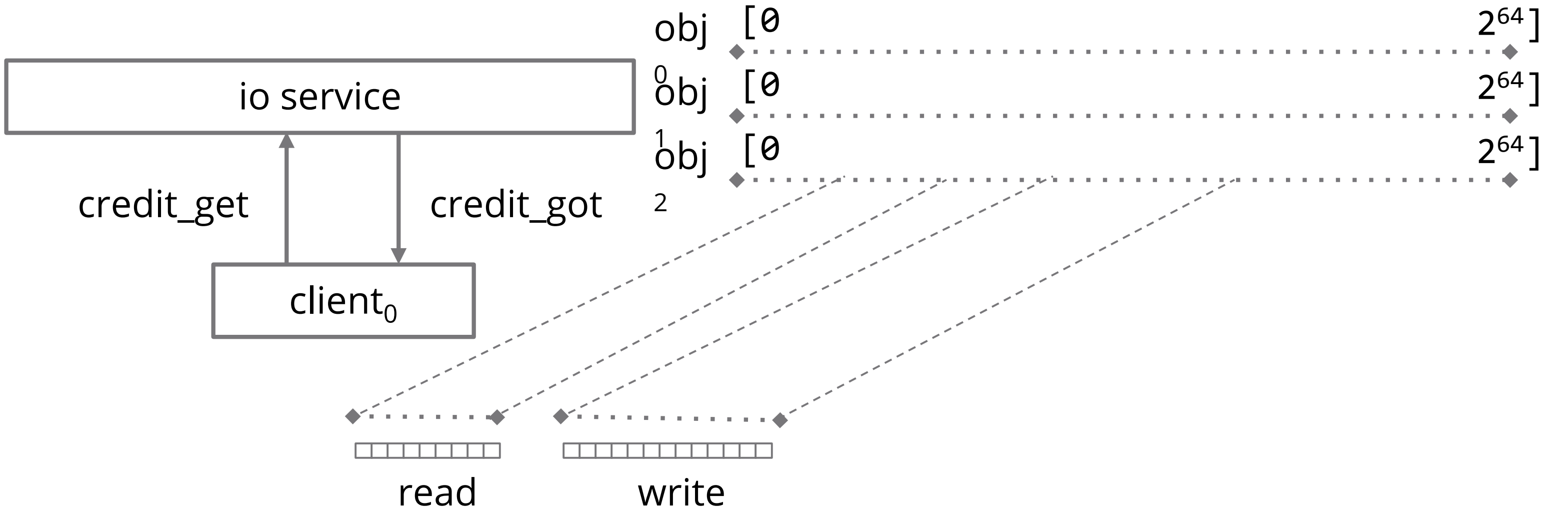
- clovis
- transactions (dtm)
- **resource manager**
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

rm: definition

- resource: anything with ownership. An extent in an object, an entire object, a key in an index, *etc*.
- credit: a right to use a resource in a particular way
- credits control:
  - distributed caching
  - concurrency
- credits can be borrowed and sublet
- resource manager is separate from resource
- resource manager resolves conflicts
- user can define new resource types

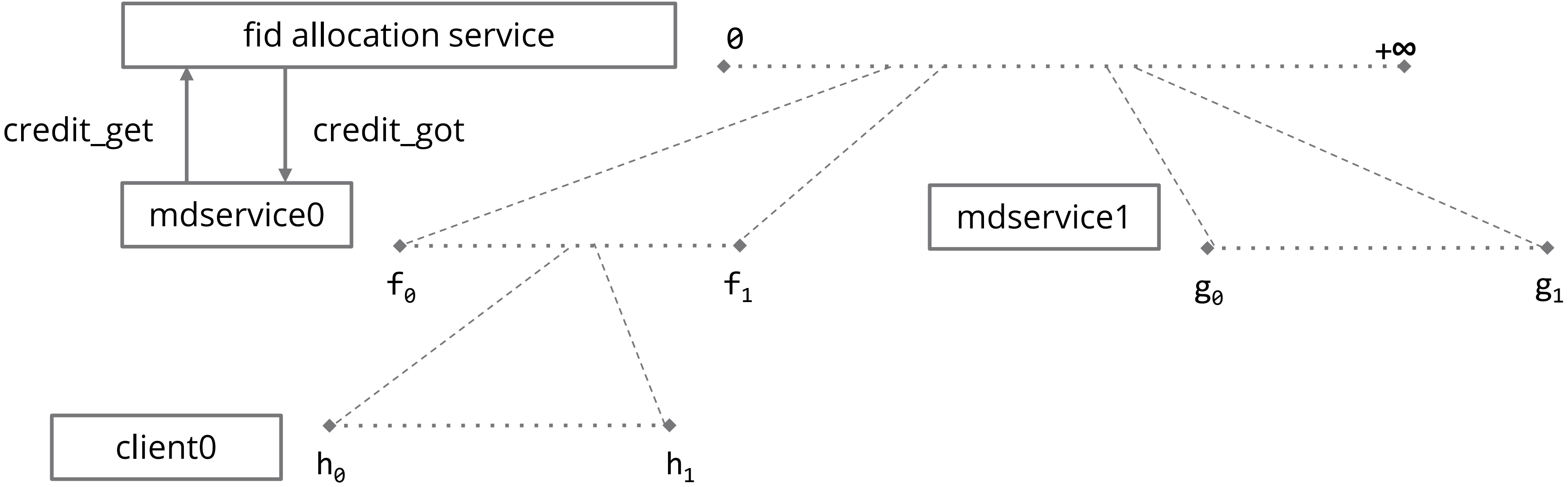
rm: use case

Example: Block extent in a clovis object



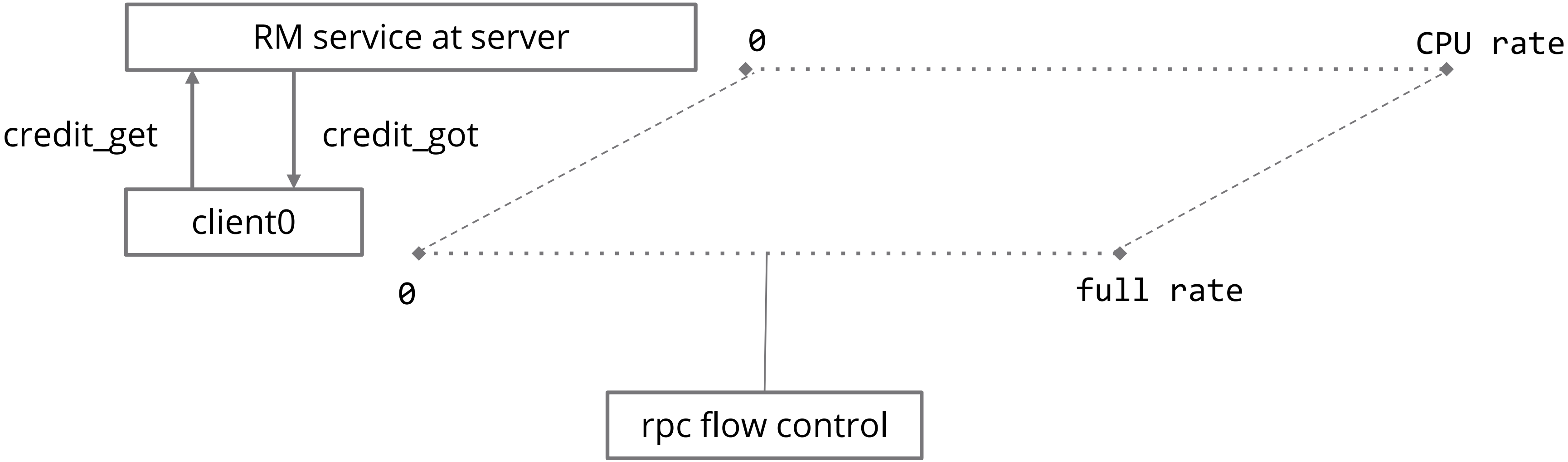
rm: use case

Example: fid extent allocation. Fid: 128 bit.



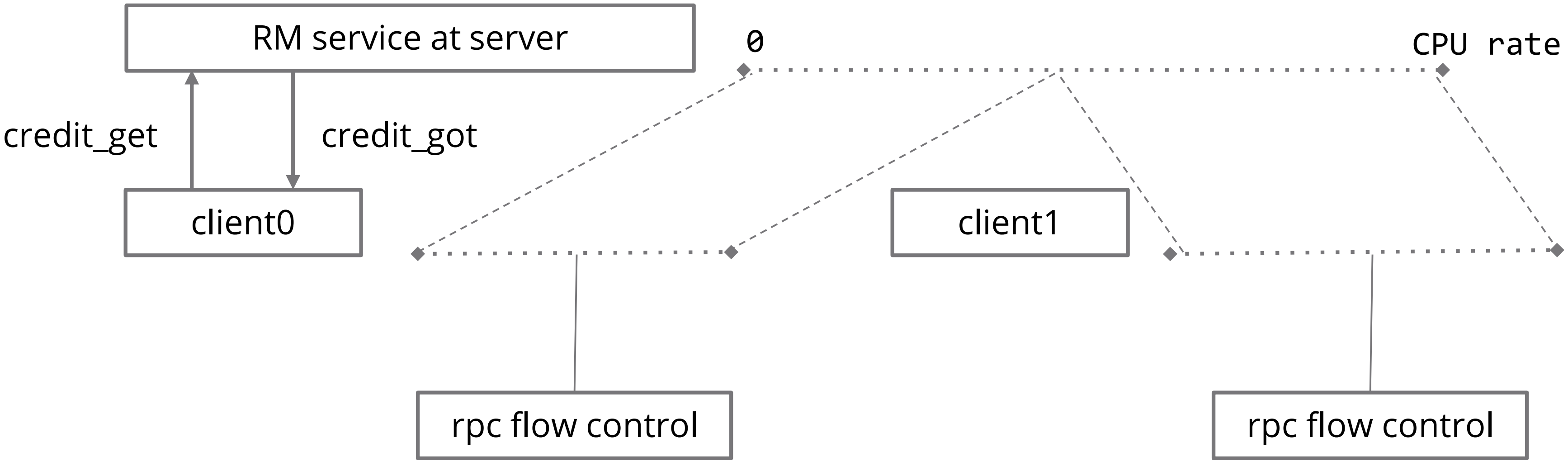
rm: use case

Example: server CPU cycles

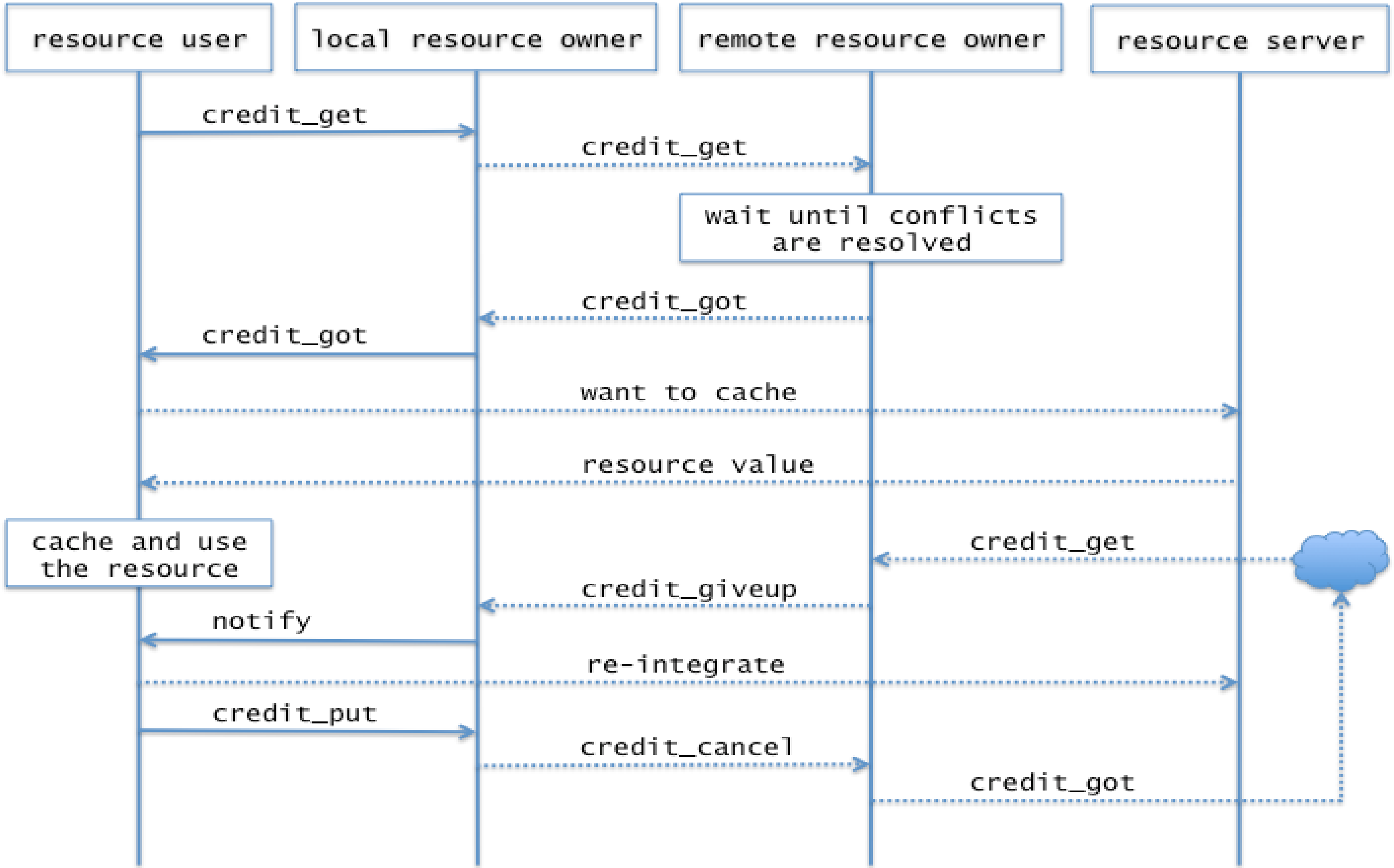


rm: use case

Example: server CPU cycles



rm: sequence


resource user
local resource owner
remote resource owner
resource server
credit_get
credit_get
wait until conflicts are resolved
credit_got
credit_got
want to cache
resource value
cache and use the resource
credit_get
credit_giveup
notify
re-integrate
credit_put
credit_cancel
credit_got

rm: resources

Resource types

- open-files
- file extents
- disk space (grants)
- quotas
- server memory
- server cpu cycles
- file identifiers (fids)
- inode numbers
- network bandwidth
- storage bandwidth
- layout
- cluster configuration
- power

rm: resource manager context

- generic infrastructure:
  - RM service: BORROW, REVOKE, CANCEL
  - client (clovis) interface
- specific resource types:
  - resource and credit names
  - conflict, credit ordering
- RM users:
  - resource acquisition and release logic
  - cache invalidation
  - assignment of RM services

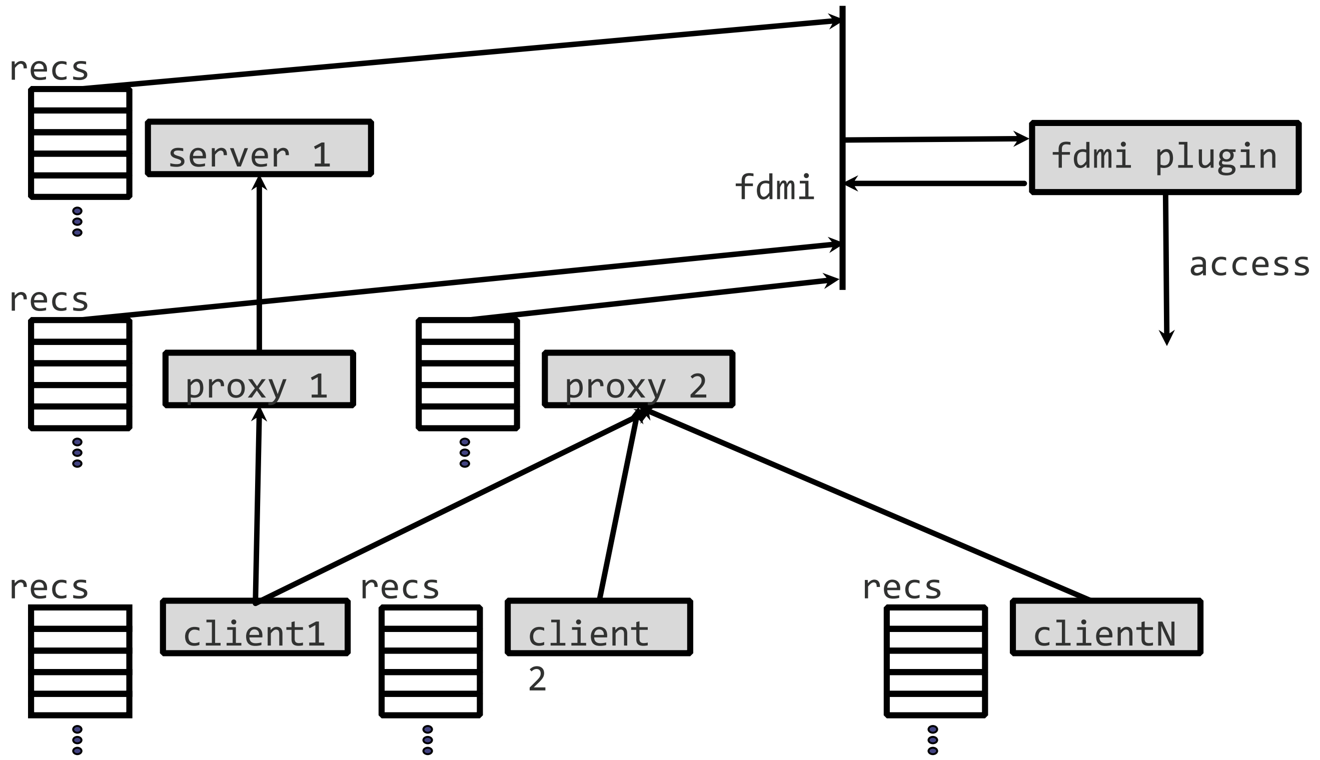
components

- clovis
- transactions (dtm)
- resource manager
- **fdmi**
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

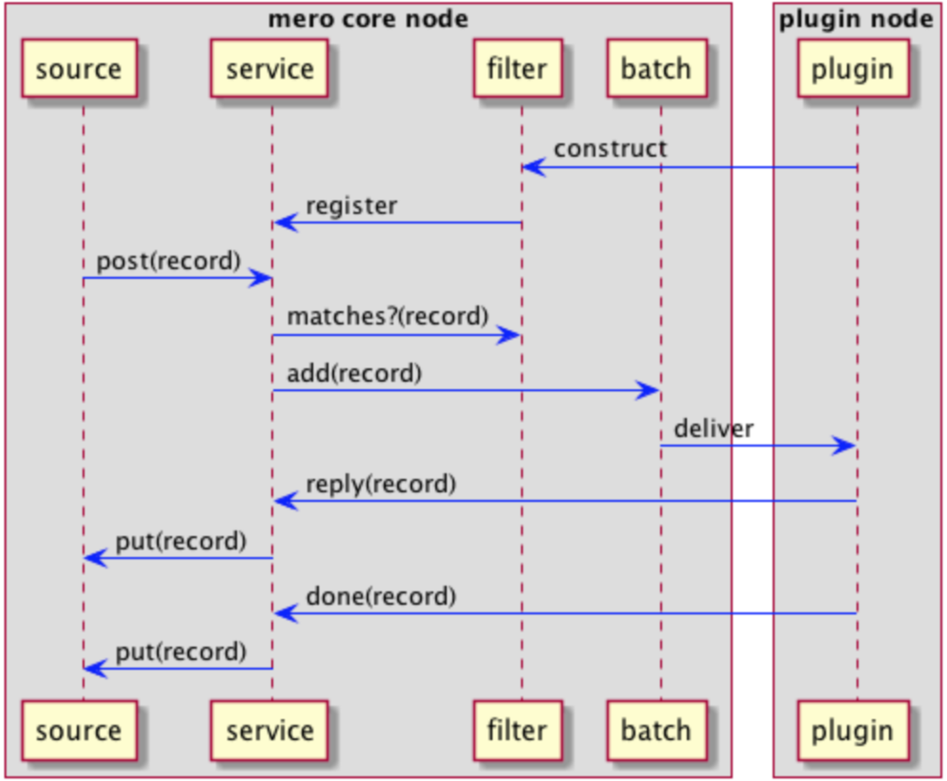
fdmi

- operation log (fol):
  - record each operation on an object
  - log consists of records, log maintained by each node
- file-system definition and manipulation interface (fdmi)
  - scalable publish-subscribe interface
  - subscribe to records matching certain filter
  - map-reduce-style mechanism to deliver matching records to the subscribers
  - transactional delivery (EOS)
  - delivers: fol records, addb records, HA events, others
- horizontal scalability
  - offload plugin processing and data-structures
  - asynchronous processing, batching

fdmi

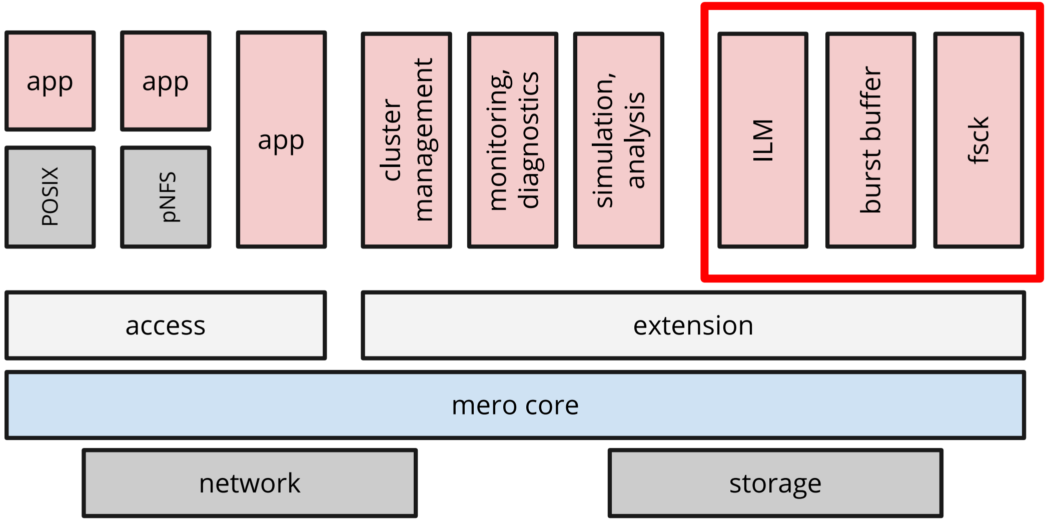

recs
server 1
fdmi
fdmi plugin
access
recs
proxy 1
proxy 2
recs
client1
recs
client
2
recs
clientN

fdmi


mero core node
plugin node
source
service
filter
batch
plugin
construct
register
post(record)
matches?(record)
add(record)
deliver
reply(record)
put(record)
done(record)
put(record)
source
service
filter
batch
plugin

fdmi

- ILM
  - replication
  - migration
  - backup, archival,
  - hsm
- indexing
- fsck
- data re-structuring
  - proxy de-staging
  - RAID re-striping
- guided interfaces
  - profiling
  - prefetching, destaging



components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- **addb**
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

addb

- systems grow larger and more complex
- how well the system is utilised?
- is it failure or expected behaviour?
- is it system or application behaviour?
- sources of data:
    - system logs
    - operating system
    - application traces
- very large amount of collected data
- ... or insufficiently detailed, or both
- difficult to analyse and correlate

addb

- instrumentation on client and server
- data about operation execution and system state
- passed through network
- cross-referenced
- always on (post-mortem analysis, first incident fix)
- simulation (change configuration, larger system, load mix)

```
* 2015-04-20-14:36:13.687531192 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    2015-04-20-14:36:13.629431319, <200000000000003:10000>, count: 8, bvec-nr: 8, ivec-nr: 1, offset: 0
|  stob-io-launch    2015-04-20-14:36:13.666152841, <100000000adf11e:3>, count: 8, bvec-nr: 8, ivec-nr: 8, offset: 65536
```

addb: anatomy of a record

```
* 2015-04-14-15:33:11.998165453 fom-descr service: <7300000000000001:0>, sender: c28baccf27e0001, req-opcode: Read request,
  rep-opcode: Read reply, local: false
|         node             <11186d8bf0e34117:ab1897c062a22573>
|         locality         3
|         thread           7f79e57fb700
|         ast
|         fom              @0x7f795008ed20, 'IO fom', transitions: 0, phase: 0
```

- measurement and context
- timestamped
- labels: identify context
- payload: up to 16 64-bit values,
- interpreted by consumer

addb: sensors and histograms

```
* 2018-04-05-14:27:40.563070315 fom-active nr: 710 min: 0 max: 24 avg: 5.029577 dev: 11.147012
  1 1: 226 3: 313 5: 98 7: 26 9: 20 11: 8 13: 5 15: 6 17: 4 19: 3 21: 0 23: 0 25: 0
|         node             <5e341757d0cf46eb:92a6d6e991b46387>
|         pid              6627
|         locality         0
```

- some events are too frequent
- collapse them into counters
- count last events per locality
- automatically size buckets

```
   :   1 |
 1 : 226 | ***********************
 3 : 313 | *******************************
 5 :  98 | **********
 7 :  26 | ***
 9 :  20 | **
11 :   8 | *
13 :   5 |
15 :   6 |
17 :   4 |
19 :   3 |
21 :   0 |
23 :   0 |
25 :   0 |
```

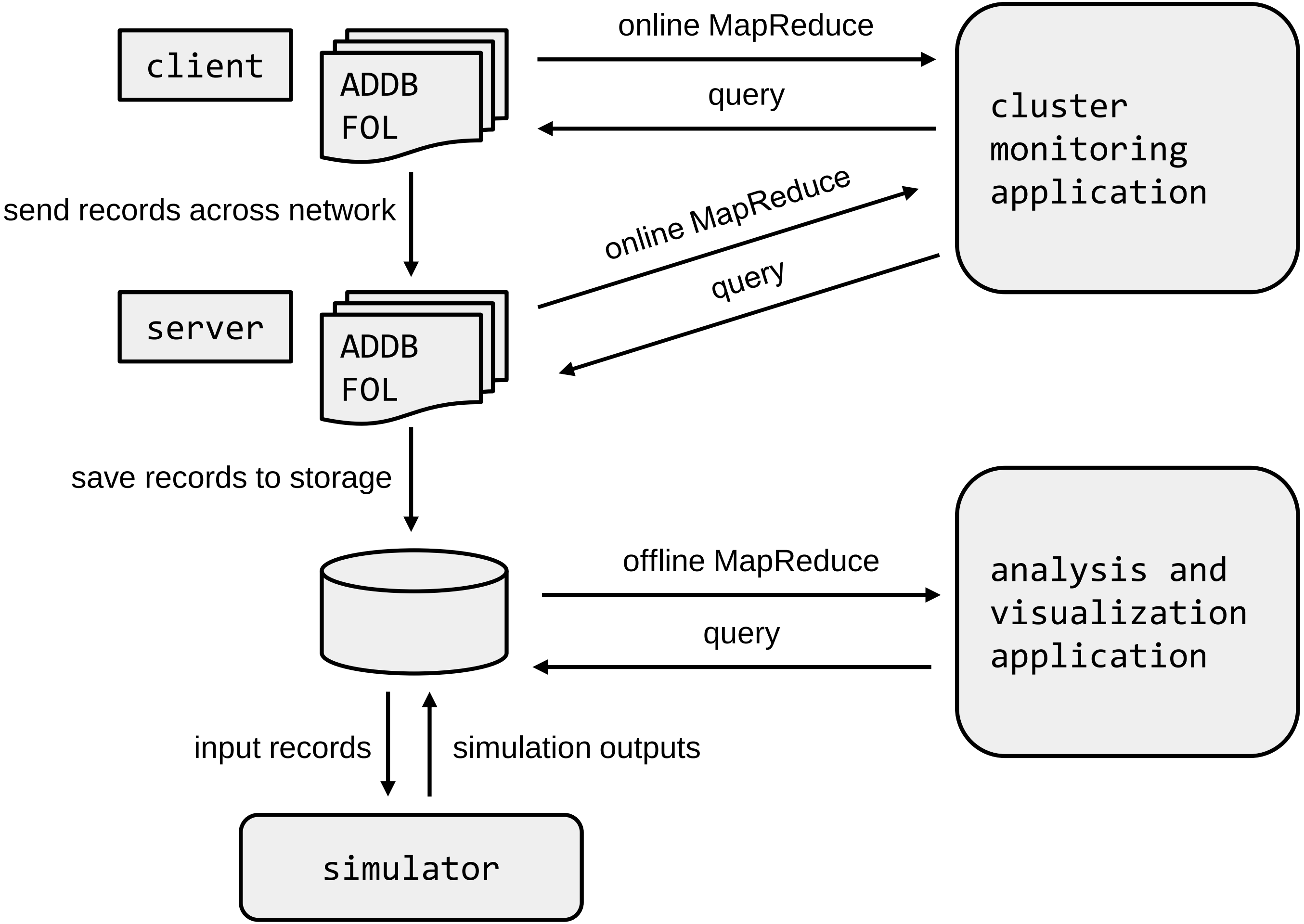
addb: state-machine transitions

```
* 2018-04-05-14:37:09.660378140 be-op/m0_be_op::bo_sm: M0_BOS_ACTIVE -[completed]-> M0_BOS_DONE
  nr: 27788 min: 0 max: 556380 avg: 785.763279 dev: 56005939.461453 datum: 0
  0 0: 25862 2066: 1138 4132: 309 6198: 98 8264: 61 10330: 67 12396: 68
    14462: 30 16528: 12 18594: 10 20660: 23 22726: 29 24792: 81
|         node             <5e341757d0cf46eb:92a6d6e991b46387>
|         pid              6627
|         locality         2
```

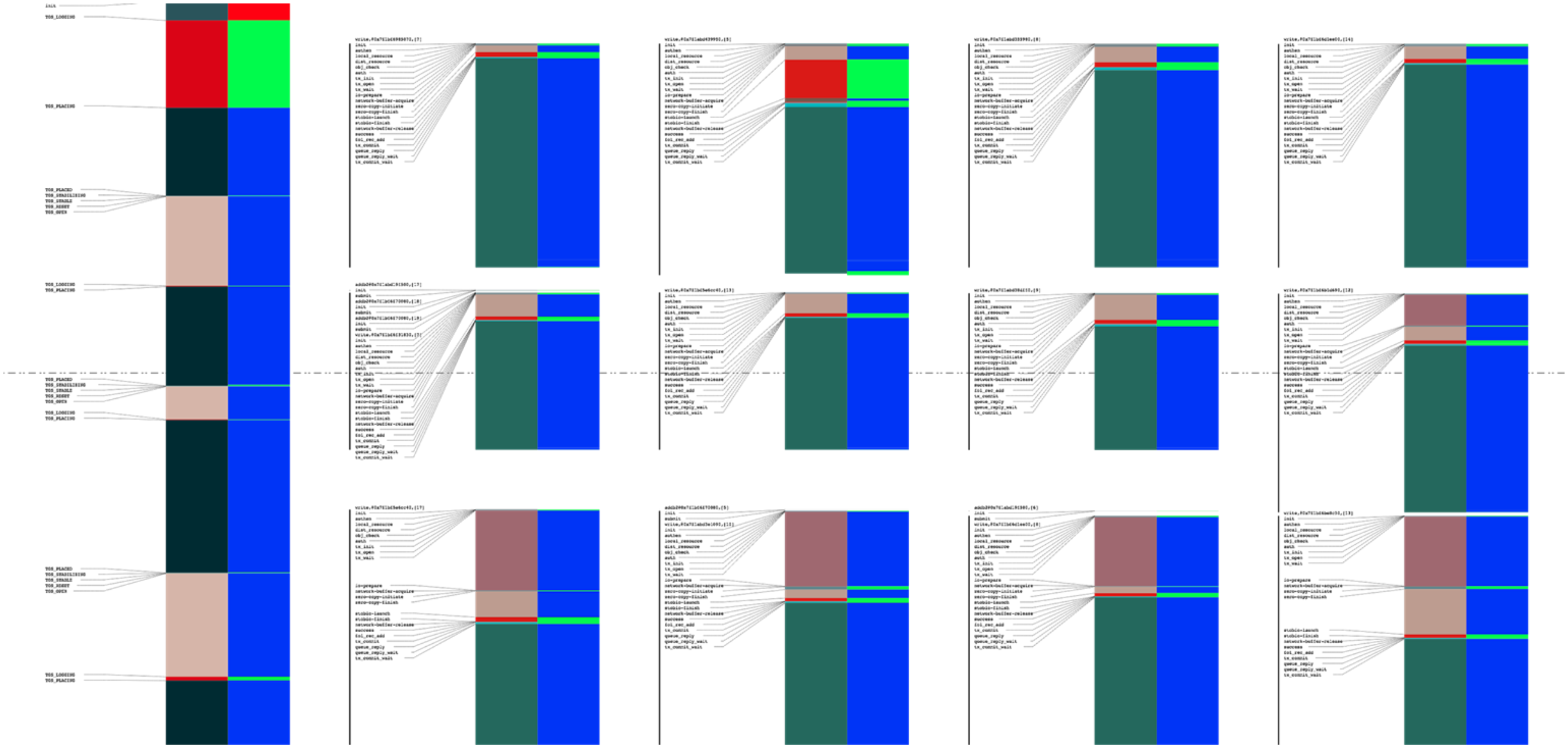
- request processing on clients and servers
- state machines
- state transition delays: in "binary microseconds", second >> 10

```
      :     0 |
    0 : 25862 | **************************
 2066 :  1138 | ************
 4132 :   309 | ***
 6198 :    98 | *
 8264 :    61 | *
10330 :    67 | *
12396 :    68 | *
14462 :    30 |
16528 :    12 |
18594 :    10 |
20660 :    23 |
22726 :    29 |
24792 :    81 | *
```

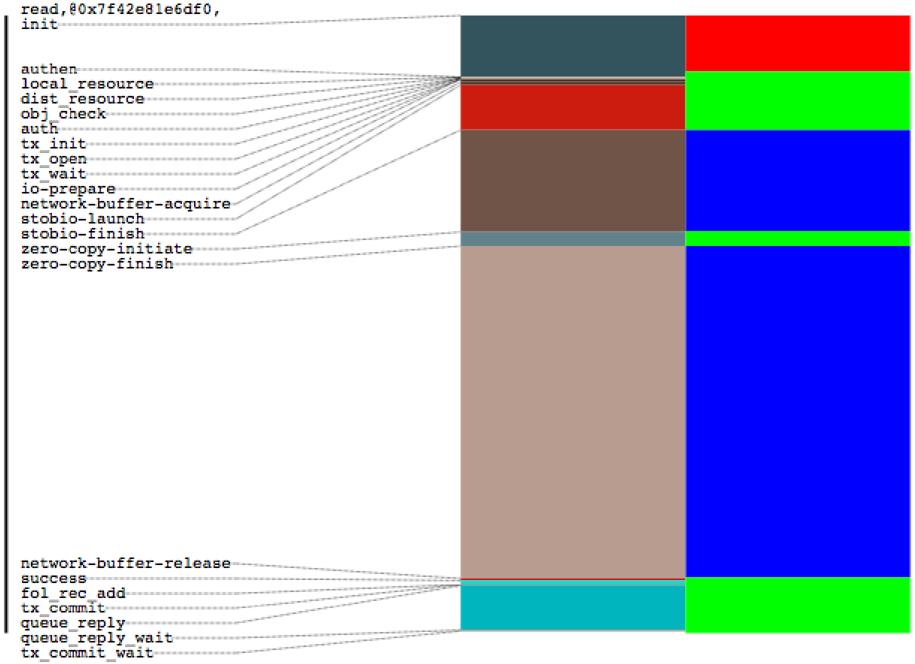
addb


client
ADDB
FOL
online MapReduce
query
cluster
monitoring
application
send records across network
online MapReduce
query
server
ADDB
FOL
save records to storage
offline MapReduce
query
analysis and
visualization
application
input records
simulation outputs
simulator

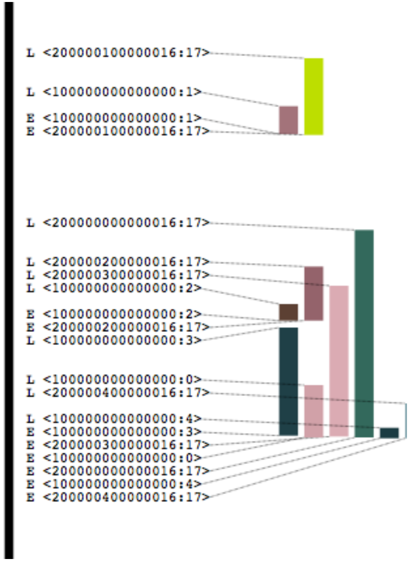
addb


init
TOS_LOGGING
TOS_PLACING
TOS_PLACED
TOS_STABILIZING
TOS_STABLE
TOS_RESET
TOS_OPEN
TOS_LOGGING
TOS_PLACING
TOS_PLACED
TOS_STABILIZING
TOS_STABLE
TOS_RESET
TOS_OPEN
TOS_LOGGING
TOS_PLACING
TOS_PLACED
TOS_STABILIZING
TOS_STABLE
TOS_RESET
TOS_OPEN
TOS_LOGGING
TOS_PLACING
write,@0x7f1b6890070,(7)
init
authen
local_resource
dist_resource
obj_check
auth
tx_init
tx_open
tx_wait
io-prepare
network-buffer-acquire
zero-copy-initiate
zero-copy-finish
stobio-launch
stobio-finish
network-buffer-release
success
fol_rec_add
tx_commit
queue_reply
queue_reply_wait
tx_commit_wait

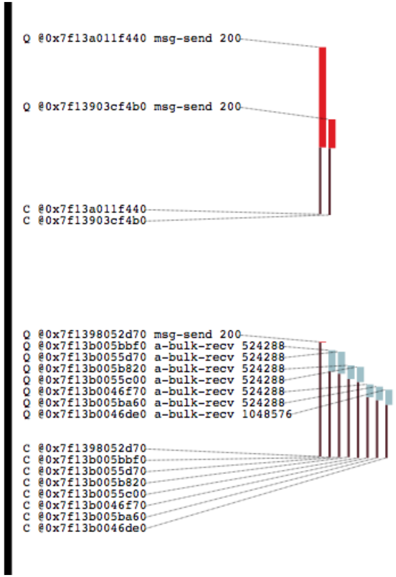
addb



read,@0x7f42e81e6df0,
init
authen
local_resource
dist_resource
obj_check
auth
tx_init
tx_open
tx_wait
io-prepare
network-buffer-acquire
stobio-launch
stobio-finish
zero-copy-initiate
zero-copy-finish
network-buffer-release
success
fol_rec_add
tx_commit
queue_reply
queue_reply_wait
tx_commit_wait



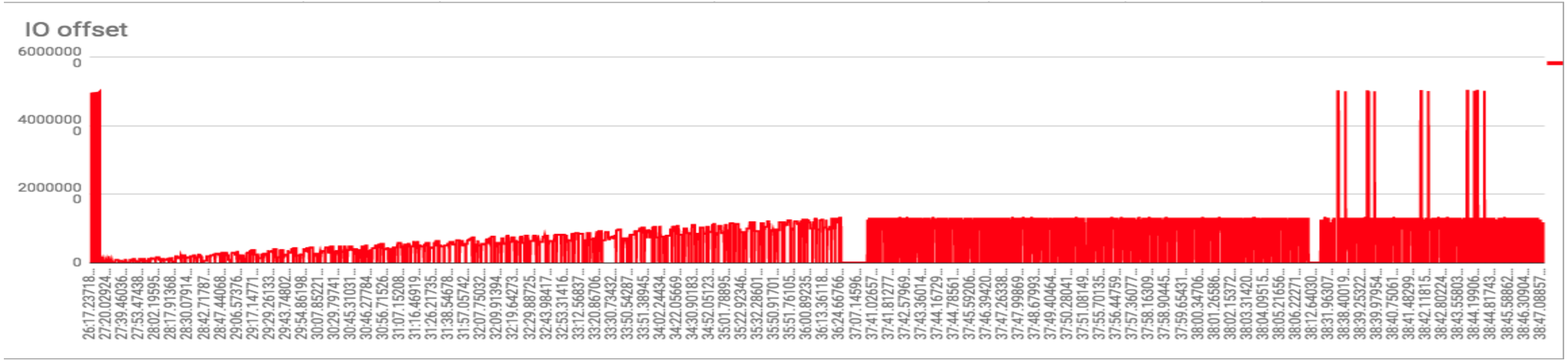
L <200000100000016:17>
L <100000000000000:1>
E <100000000000000:1>
E <200000100000016:17>
L <200000000000016:17>
L <200000200000016:17>
L <200000300000016:17>
L <100000000000000:2>
E <100000000000000:2>
E <200000200000016:17>
L <100000000000000:3>
L <100000000000000:0>
L <200000400000016:17>
L <100000000000000:4>
E <100000000000000:3>
E <200000300000016:17>
E <100000000000000:0>
E <200000000000016:17>
E <100000000000000:4>
E <200000400000016:17>

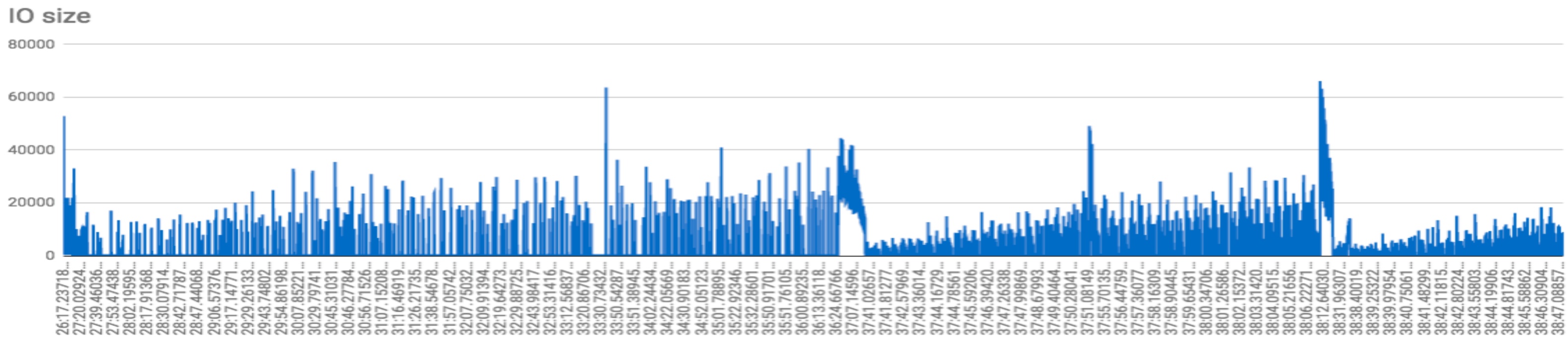


Q @0x7f13a011f440 msg-send 200
Q @0x7f13903cf4b0 msg-send 200
C @0x7f13a011f440
C @0x7f13903cf4b0
Q @0x7f1398052d70 msg-send 200
Q @0x7f13b005bbf0 a-bulk-recv 524288
Q @0x7f13b0055d70 a-bulk-recv 524288
Q @0x7f13b005b820 a-bulk-recv 524288
Q @0x7f13b0055c00 a-bulk-recv 524288
Q @0x7f13b0046f70 a-bulk-recv 524288
Q @0x7f13b005ba60 a-bulk-recv 524288
Q @0x7f13b0046de0 a-bulk-recv 1048576
C @0x7f1398052d70
C @0x7f13b005bbf0
C @0x7f13b0055d70
C @0x7f13b005b820
C @0x7f13b0055c00
C @0x7f13b0046f70
C @0x7f13b005ba60
C @0x7f13b0046de0

addb: ad hoc profiling

```
$ m0addb2dump ... | grep 'stob-io-launch' | awk '{print $2, $6, $12}'
```



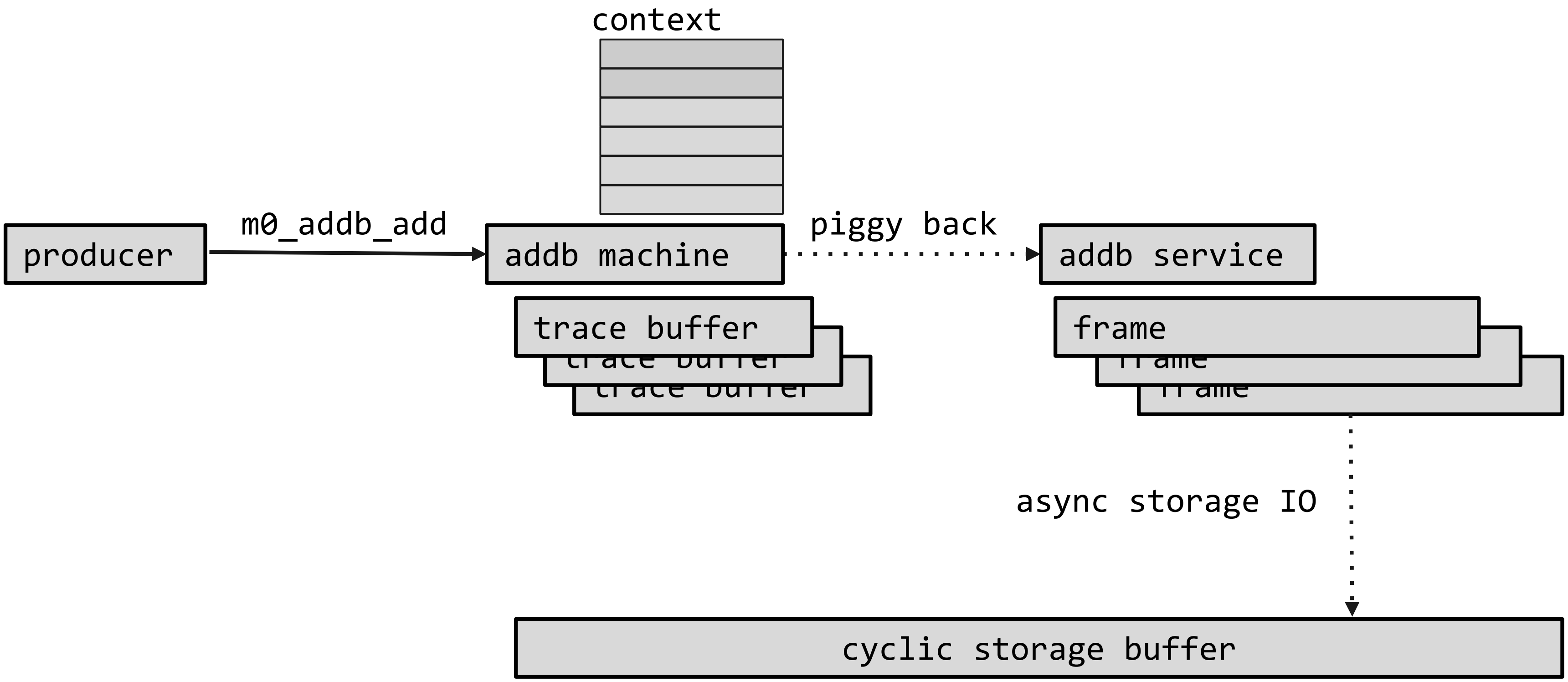


addb: interface

```
/**
 * Adds a label to the current context.
 *
 * @param id    - label identifier
 * @param n     - number of 64-bit values in label payload
 * @param value - payload
 */
void m0_addb_push(uint64_t id, int n, const uint64_t *value);
/**
 * Removes the top-most label in the current context stack.
 *
 * @param id - label identifier
 *
 * @pre "id" must the identifier of the top-most label.
 */
void m0_addb_pop(uint64_t id);
/**
 * Adds one-time measurement in the current context.
 *
 * @param id    - measurement identifier
 * @param n     - number of 64-bit values in measurement payload
 * @param value - payload
 */
void m0_addb_add(uint64_t id, int n, const uint64_t *value);
```

- very simple interface
- binary values only
- stack (LIFO) context
- context management is modular
- separate interface for sensors (not shown)
- usable from system and applications

addb: data flow



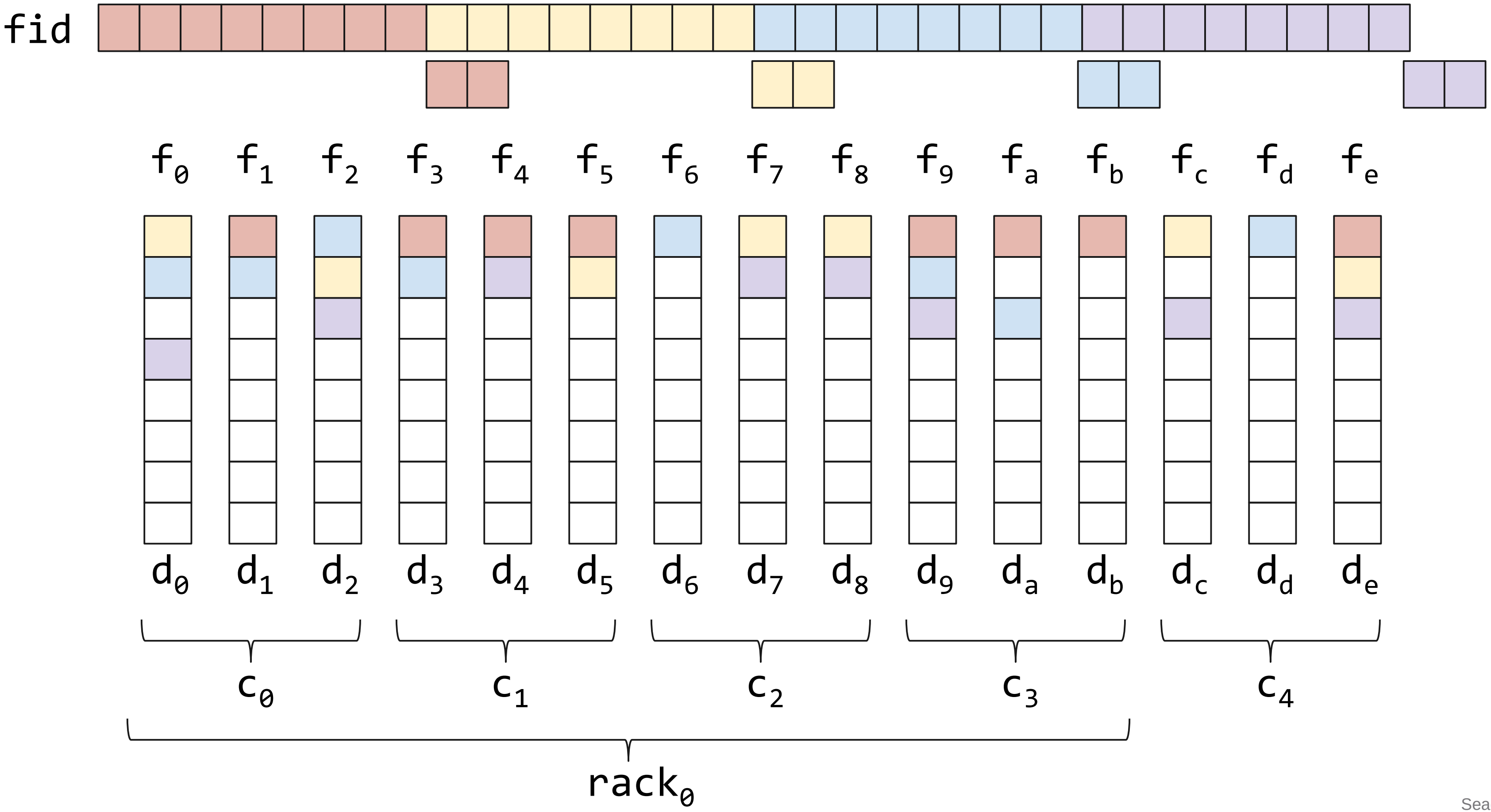
components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- **network raid, layouts**
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

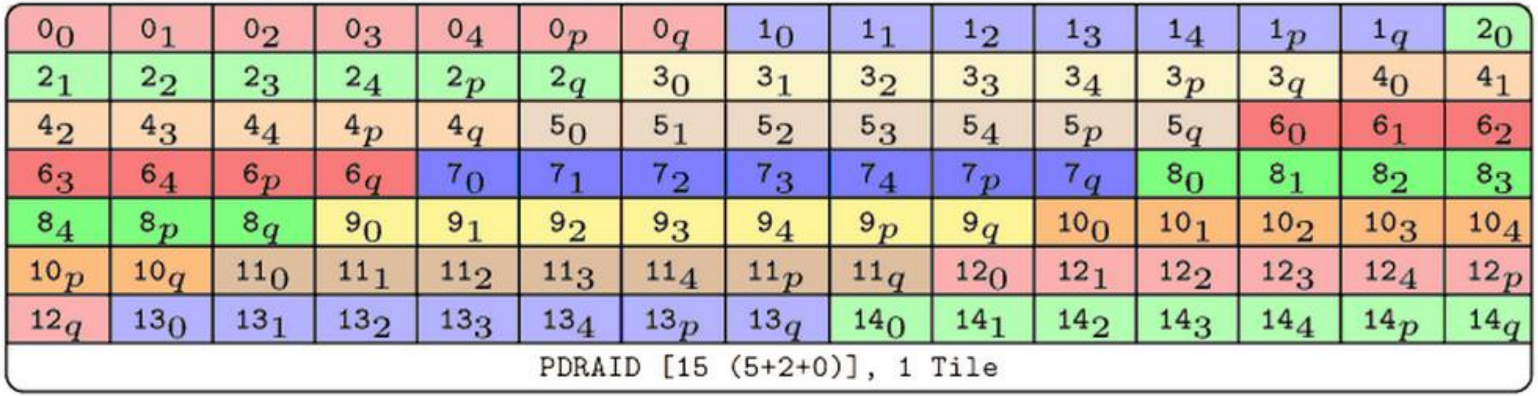
io: layout

- determines how an object is stored in underlying containers
- layouts for data and meta-data
- examples:
    - network striping with parity de-clustering (default)
    - compression
    - encryption
    - de-duplication
    - composite (NBA, small files, migration)

io: parity groups

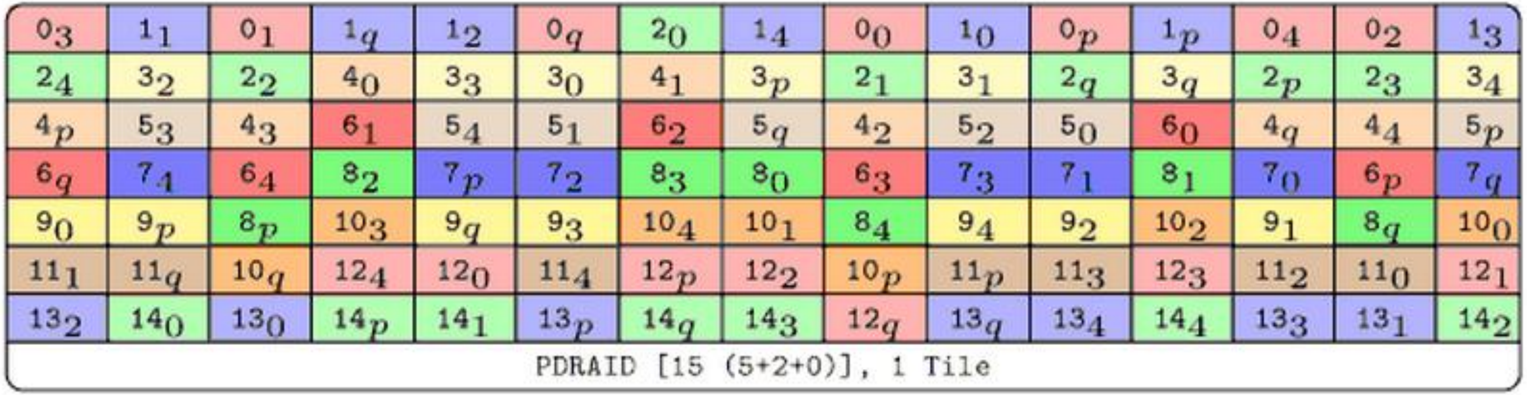


io: permutations



0_0	0_1	0_2	0_3	0_4	0_p	0_q	1_0	1_1	1_2	1_3	1_4	1_p	1_q	2_0
2_1	2_2	2_3	2_4	2_p	2_q	3_0	3_1	3_2	3_3	3_4	3_p	3_q	4_0	4_1
4_2	4_3	4_4	4_p	4_q	5_0	5_1	5_2	5_3	5_4	5_p	5_q	6_0	6_1	6_2
6_3	6_4	6_p	6_q	7_0	7_1	7_2	7_3	7_4	7_p	7_q	8_0	8_1	8_2	8_3
8_4	8_p	8_q	9_0	9_1	9_2	9_3	9_4	9_p	9_q	10_0	10_1	10_2	10_3	10_4
10_p	10_q	11_0	11_1	11_2	11_3	11_4	11_p	11_q	12_0	12_1	12_2	12_3	12_4	12_p
12_q	13_0	13_1	13_2	13_3	13_4	13_p	13_q	14_0	14_1	14_2	14_3	14_4	14_p	14_q

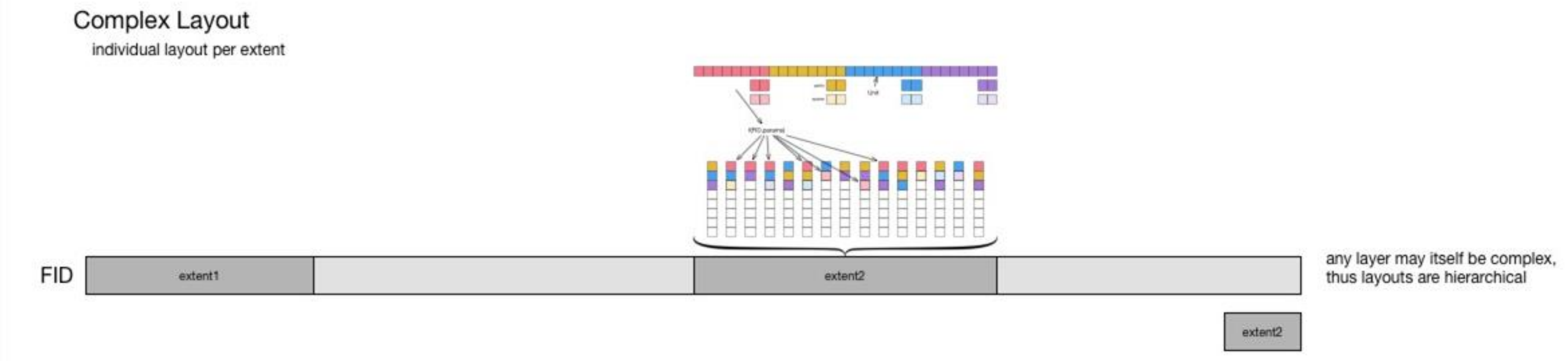
PDRAID [15 (5+2+0)], 1 Tile



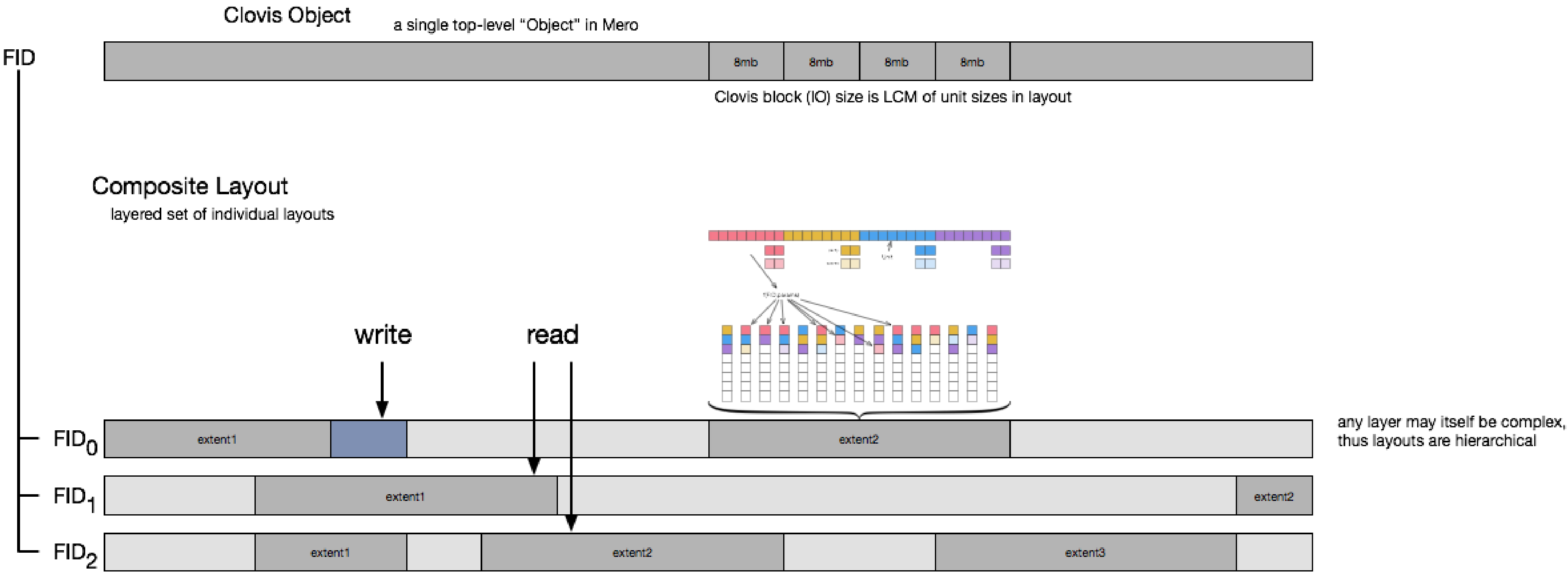
0_3	1_1	0_1	1_q	1_2	0_q	2_0	1_4	0_0	1_0	0_p	1_p	0_4	0_2	1_3
2_4	3_2	2_2	4_0	3_3	3_0	4_1	3_p	2_1	3_1	2_q	3_q	2_p	2_3	3_4
4_p	5_3	4_3	6_1	5_4	5_1	6_2	5_q	4_2	5_2	5_0	6_0	4_q	4_4	5_p
6_q	7_4	6_4	8_2	7_p	7_2	8_3	8_0	6_3	7_3	7_1	8_1	7_0	6_p	7_q
9_0	9_p	8_p	10_3	9_q	9_3	10_4	10_1	8_4	9_4	9_2	10_2	9_1	8_q	10_0
11_1	11_q	10_q	12_4	12_0	11_4	12_p	12_2	10_p	11_p	11_3	12_3	11_2	11_0	12_1
13_2	14_0	13_0	14_p	14_1	13_p	14_q	14_3	12_q	13_q	13_4	14_4	13_3	13_1	14_2

PDRAID [15 (5+2+0)], 1 Tile

layout: composite


Complex Layout
individual layout per extent
Unit
FID
extent1
extent2
any layer may itself be complex,
thus layouts are hierarchical
extent2

layout: composite



Clovis Object

a single top-level "Object" in Mero

Clovis block (IO) size is LCM of unit sizes in layout

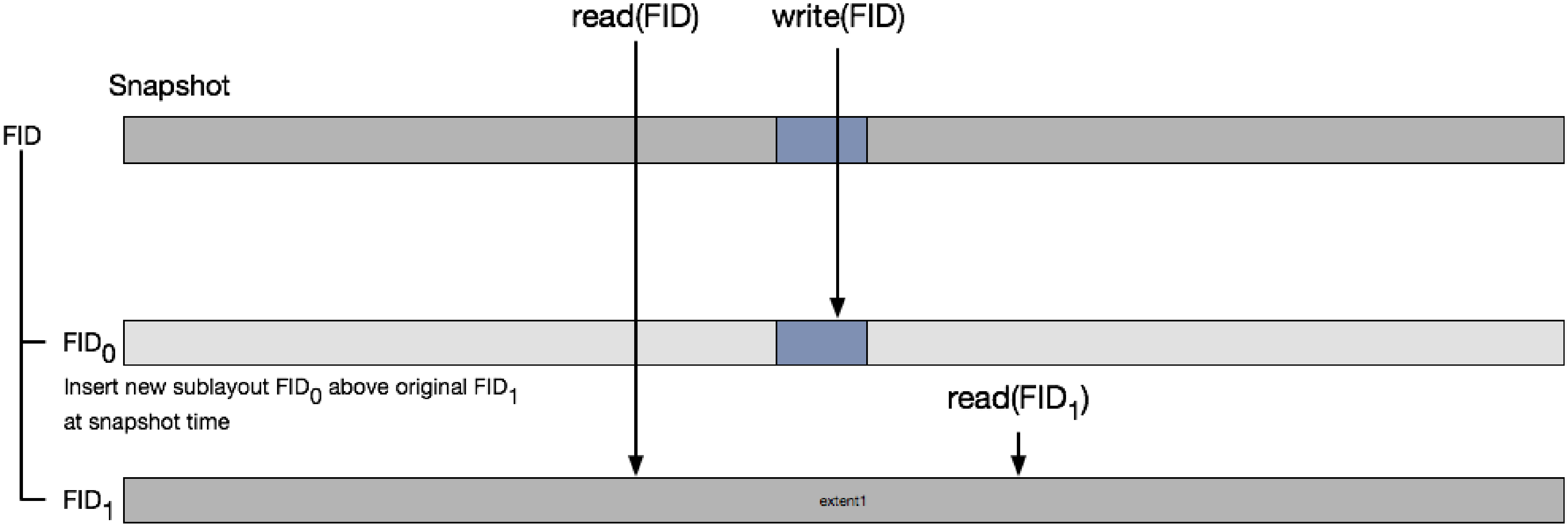
Composite Layout

layered set of individual layouts

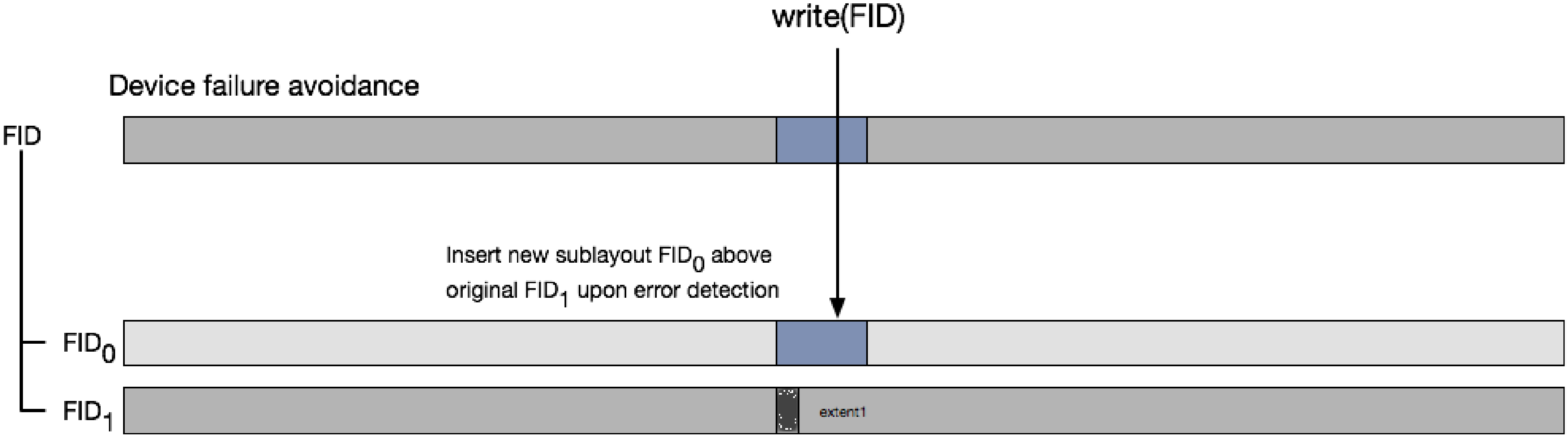
any layer may itself be complex, thus layouts are hierarchical

Reads and writes fall through to first layer with mapped extents. (Newly written blocks are added to read extents.) Layers can also be read/written directly using FID$_{sub}$

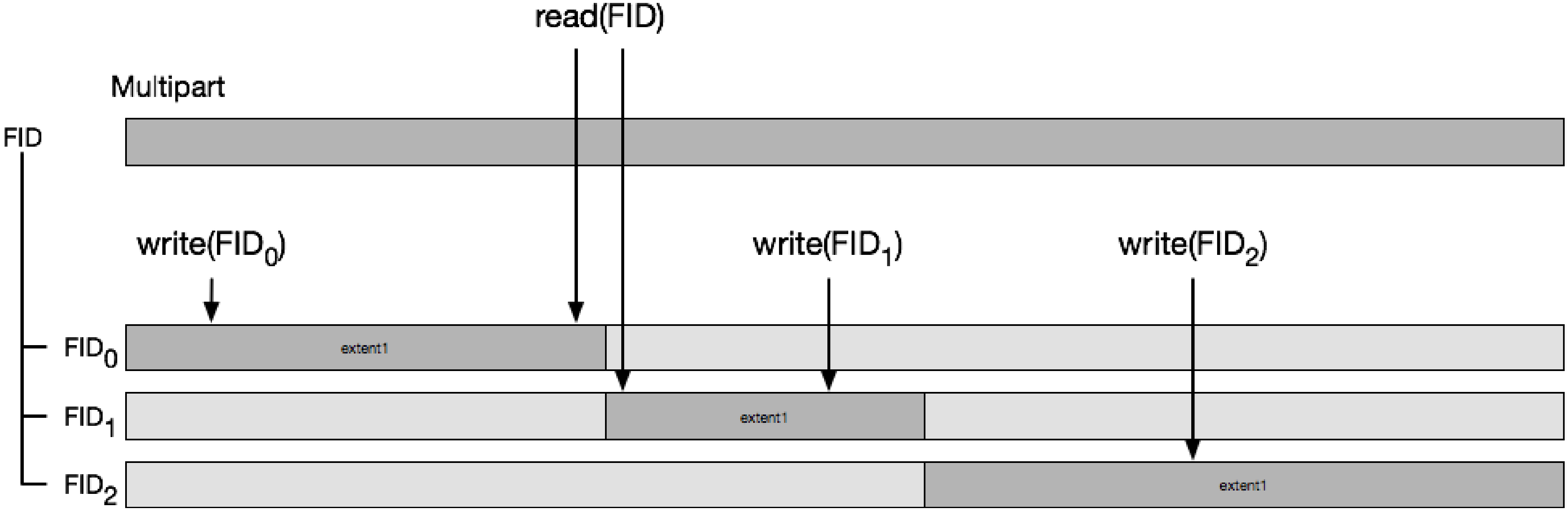
layout: composite: snapshot



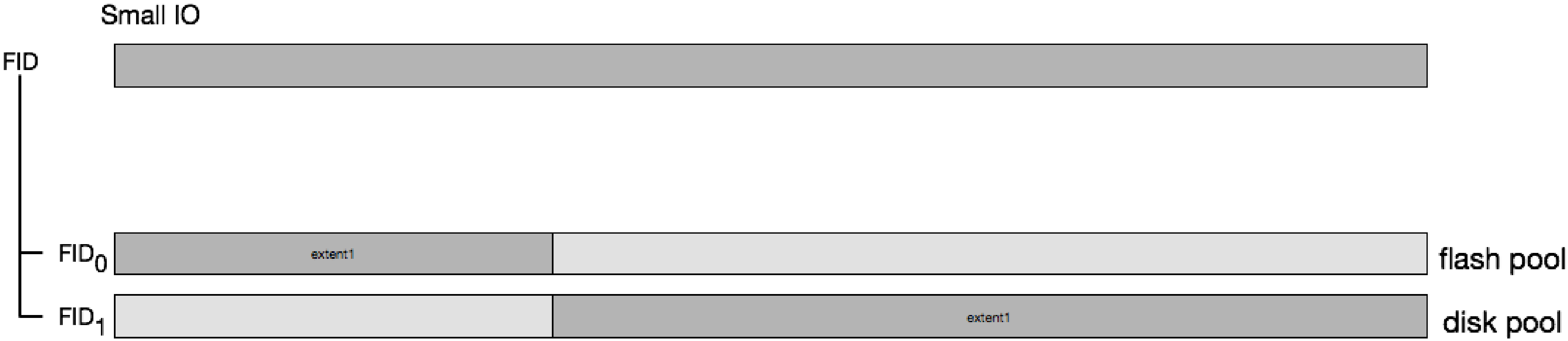
layout: composite: nba


write(FID)
Device failure avoidance
FID
Insert new sublayout FID0 above
original FID1 upon error detection
FID0
FID1
extent1

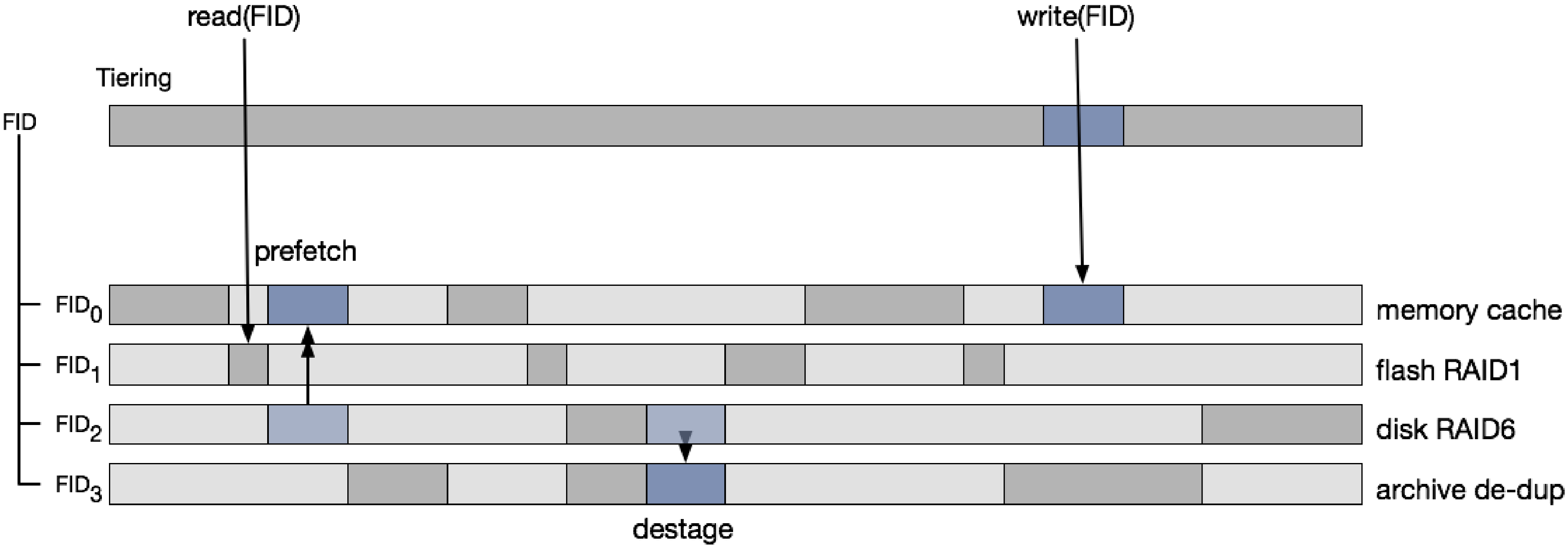
layout: composite: s3 partial upload



layout: composite: small files IO



layout: composite: HSM, tiering



components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- **containers**
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

containers

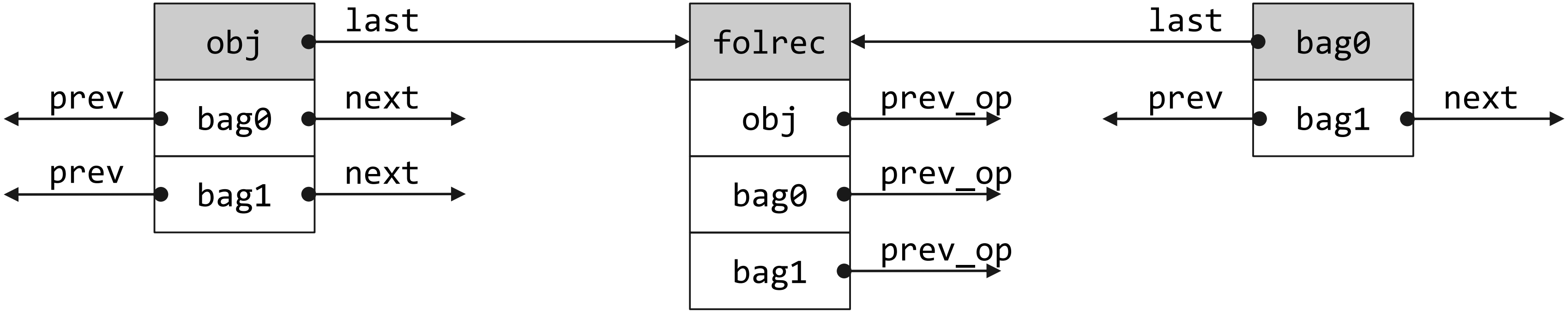
- container: an entity with a fid
- application fully controls containers
  - add an entity to a container;
  - remove an entity from a container;
  - list container elements;
  - execute bulk operation on container contents (compute-in-container)
- arbitrary "topology" (only restriction: no duplicates)

Non-properties:

- separate fid-space for contents
- strong isolation guarantees

containers implementation

- component containers, bag:
  - entities with fids,
  - lazily created when an entity is added to the container
- all local entities are linked together
- file operations log records (folrecs) for operations on container contents are linked



containers rationale

- efficiently identify containers to which an object belongs
- flexible container nesting
- concurrent mass-operations on containers
- container history is traceable
- fdmi filters on containers
- function shipping to a container

Don't have:

- ordering of container contents
- ordered enumeration
- "implicit" containers (*e.g.*, "all objects on this server")

containers use case

- application creates a container to track "entities of interest"
- entities are added to the container
- fdmi plugin is registered to receive notifications of all updates to the container
- examples:
    - hsm: fileset of hot objects
    - replication: source fileset

components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- **function shipping**
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

function shipping

- move computation closer to data (compute-in-storage)
- reduce network transmission overhead
- application structuring mechanism
- bulk computation on all Clovis entities:
  - object: function on data blocks
  - index: function on key-value records
  - container: function on member fids
- built-in fault tolerance

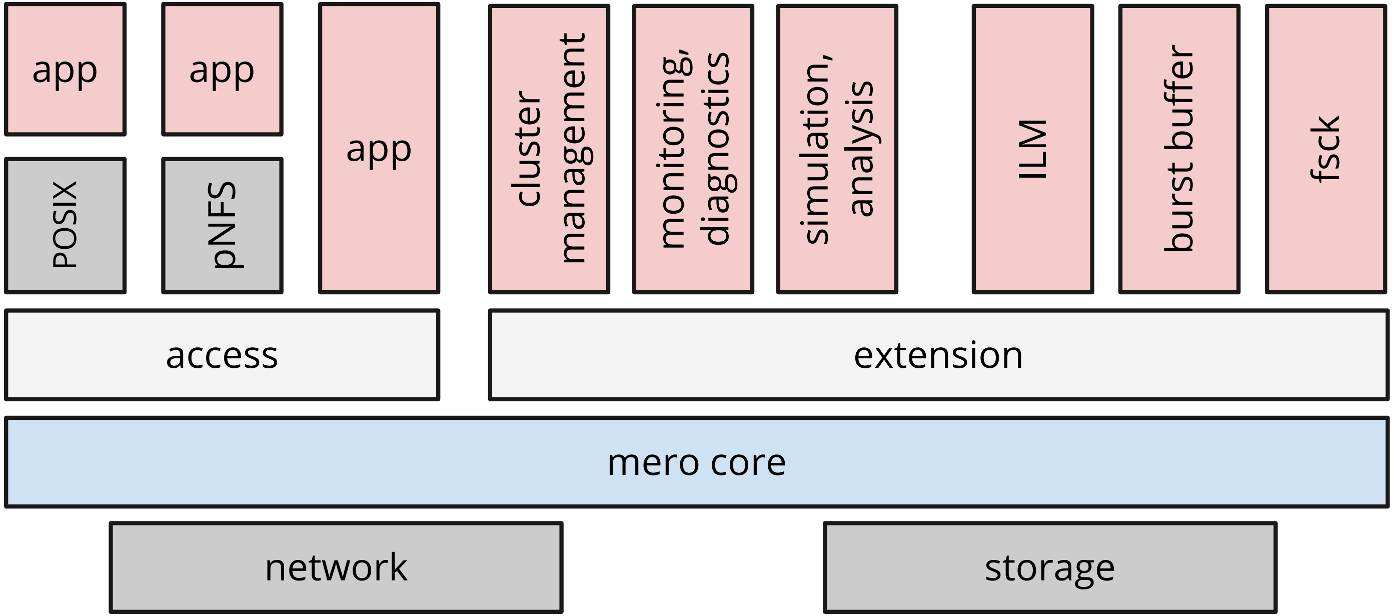
function shipping implementation

- computations are first class Clovis entities
  - unique fid, globally addressable
  - registered dynamically by an application
- low level trusted mechanism:
  - dynamically load shared library into Mero service process
  - invoke computations remotely, argument-result passing
- untrusted mechanism:
  - run untrusted code (*e.g.*, Python) in a separate address space
- client uses layouts to start execution and recover from failures

components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- **lingua franca**
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

lingua franca



Multiple front-ends:

interoperability, common meta-data mechanism

lingua franca

- POSIX file system: *via* pNFS
  - straightforward semantics
  - concurrency:
    - concurrent reads
    - one writer
    - concurrent writers
- POSIX file system: *via* S3
  - interpret pathname as an URL (object key)
  - mapping of security and ownership attributes
- S3: *via* POSIX: parse URL as a pathname
- POSIX file system: *via* HDF5

lingua franca

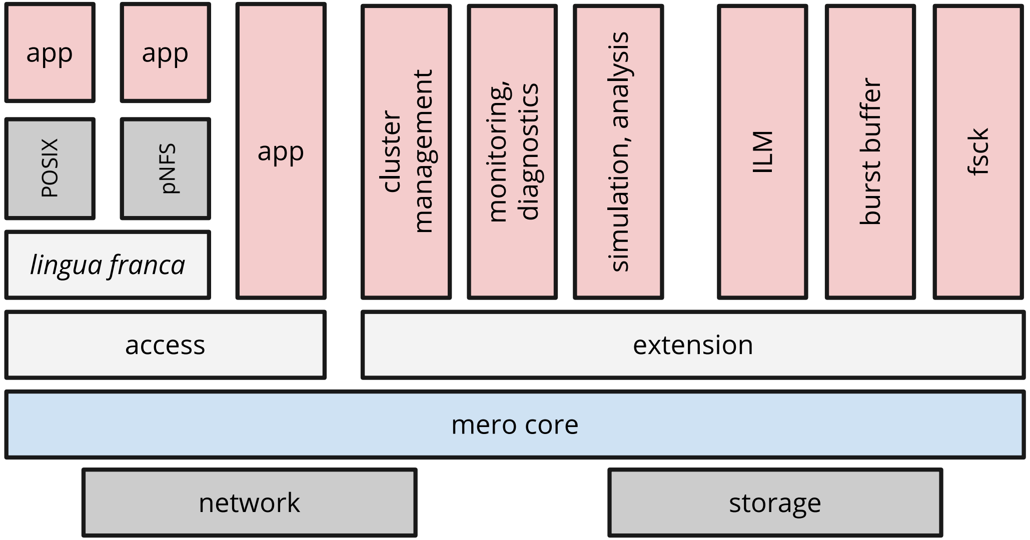
- S3 object: "blue_bucket/finance-9a84c723ee89f4b723b46cc5f1642b3"
- mount S3 store as POSIX

  ```
    $   cd blue_bucket
    $   ls -l finance-9a84c723ee8*

  -rw------- 1 satoshi satoshi 285 January 03 2009 finance-9a84c723ee89f4b723b46cc5f1642b3
  ```

- enumerate and locate objects
- access attributes
  - system attributes: layout, containers;
  - common attributes: size;
  - front-end specific attributes: POSIX nlink
- efficiency: remote meta-data lookups are expensive

lingua franca



POSIX, pNFS, CIFS, S3/SWIFT, MPI Object IO, HDF5, MySQL, block device

lingua franca implementation

- what are the *entities* managed by an FE?
- how entities are named?
- how entities are organised: tree, graph, array?
- how attributes are associated with entities?
  - different FEs have different sets of native attributes
  - an FE wants to add attributes to foreign entities
  - an FE wants to interpret foreign attributes
  - some attributes are shared by multiple FEs
- core system has its own attributes

lingua franca implementation

- a domain of files (entities), a file identified by a 128-bit *fid*
- two indices:
  - name-space (ns): records file attributes
  - object index (oi): maps from the file fid to all its names

ns index structure:

key	value
FRONTENDID+FNAME+ATTRCLASS+ATTRID	attribute value

oi index structure:

FID+FRONTENDID+FNAMEID	FRONTENDID+FNAME

lingua franca implementation: S3

- FRONTENDID: 3
- ATTRCLASS: 3
- FNAME: Object-URI: bucketname+NUL+object_key+NUL

object: "app_bucket/statement.xls", fid 0x60000:0x17ae76d0f

ns index:

key	value	comment
3app_bucket\0statement.xls\0!fid	0x60000:0x17ae76d0f	the fid of the object "statement.xls"
3app_bucket\0statement.xls\03content-length	3201526	object size in bytes
3app_bucket\0statement.xls\03content-md5	0x62ae2f12137738a9:0x173f5224f81446aa	md5 checksum
3app_bucket\0statement.xls\03last-modified	2017-04-26-15:29:30.85267	last modification timestamp
3app_bucket\0statement.xls\03...		other S3 attributes

lingua franca implementation: S3

object: "app_bucket/statement.xls", fid `0x60000:0x17ae76d0f`

oi index:

key	value	comment
`3:0000000000060000:0000000017ae76d0:30`	`3app_bucket\0statement.xls`	name of the object

lingua franca implementation: POSIX

- `FRONTENDID: P`
- `ATTRCLASS: P`
- `FNAME: parent_directory_fid+name_in_the_directory`

object: "/etc/passwd", fid `0x70000:0x322e1673fd`

parent directory: "/etc", fid: `0x70000:0x18203a6485`

ns index:

key	value	comment
`P:70000:18203a6485:passwd!fid`	`0x70000:0x322e1673fd`	the fid of the object "`/etc/passwd`"
`P:70000:18203a6485:passwdPsize`	`16523`	file size
`P:70000:18203a6485:passwdPatime`	`2017-04-26-15:29:30.85267`	access time
`P:70000:18203a6485:passwdP...`		other POSIX attributes

lingua franca implementation: POSIX

object: "/etc/passwd", fid `0x70000:0x322e1673fd`

oi index:

key	value	comment
`P:0000000000070000:0x000000322e1673fd:P0`	`P:70000:18203a6485:passwd`	name of the object
`P:0000000000070000:0x000000322e1673fd:P1`	`P:70000:18203a6485:passwd.1`	another name: hard-link

```
# ln /etc/passwd /etc/passwd.1
```

lingua franca implementation: features

- each attribute is a separate key-value pair: flexibility
- new attributes can be added to existing files
  - without breaking compatibility
- attributes can be enumerated (NEXT operation)
  - an FE can selectively handle attributes it understands
- contents of a directory can be enumerated
- new names can be added to a file

components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- **integrity checking**
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

integrity checking

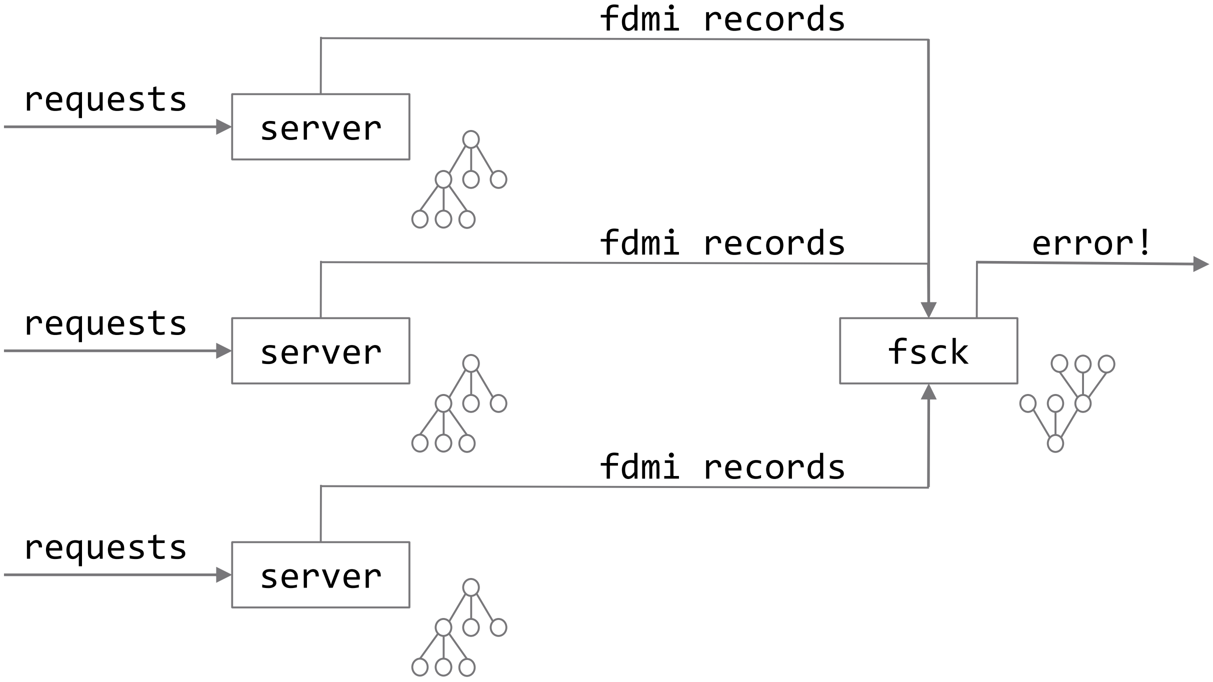
- redundancy, fancy metadata: not an answer (has been tried)
    - bugs (more important over time)
    - recovery from catastrophic failures
- traditional fsck
    - not distributed
    - specific to particular meta-data format
    - does not scale
        - time
        - space

integrity checking

- need scalable integrity checking
- run it all the time
- on dedicated separate nodes (horizontal scalability)
  - maintain redundant "inverse" meta-data
  - update meta-data to match system evolution (fdmi)
  - detect inconsistencies
  - report, recover from redundancy
  - recover catastrophic failures
- usual redundancy: parity, checksums, background scrub
- fdmi: transactional coherence with the main state

- parallel programming

integrity checking



inverse meta-data

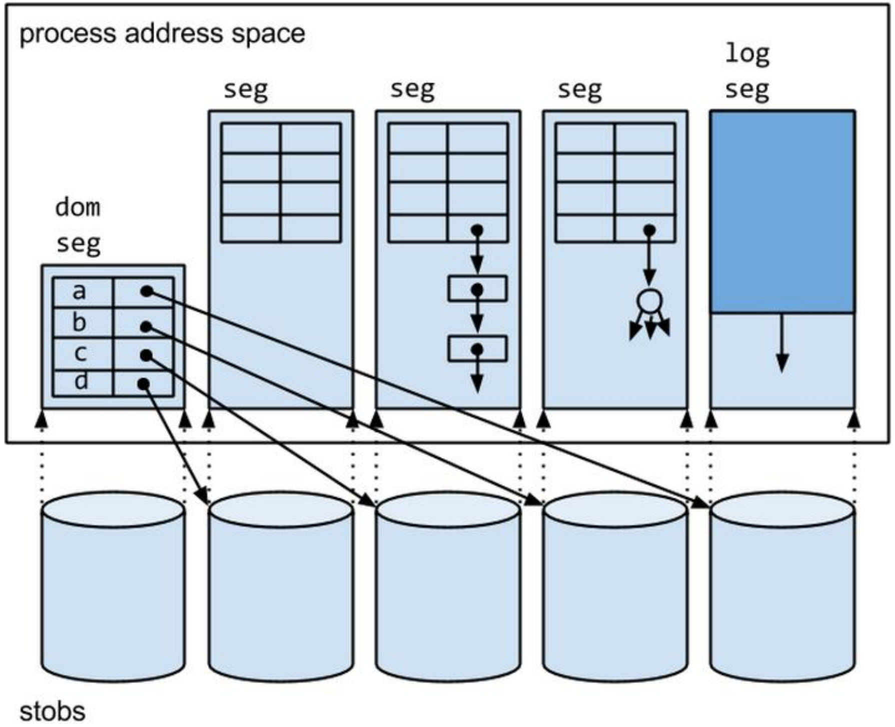
- block allocation
- pdclust structure
- key distribution
- b-tree structure
- application specific invariants
  - POSIX tree
  - hdf5

components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- **meta-data back-end**
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- security

backend

- segment
- transaction
- domain
- container
- WAL
- redo-only
- allocator
- btree



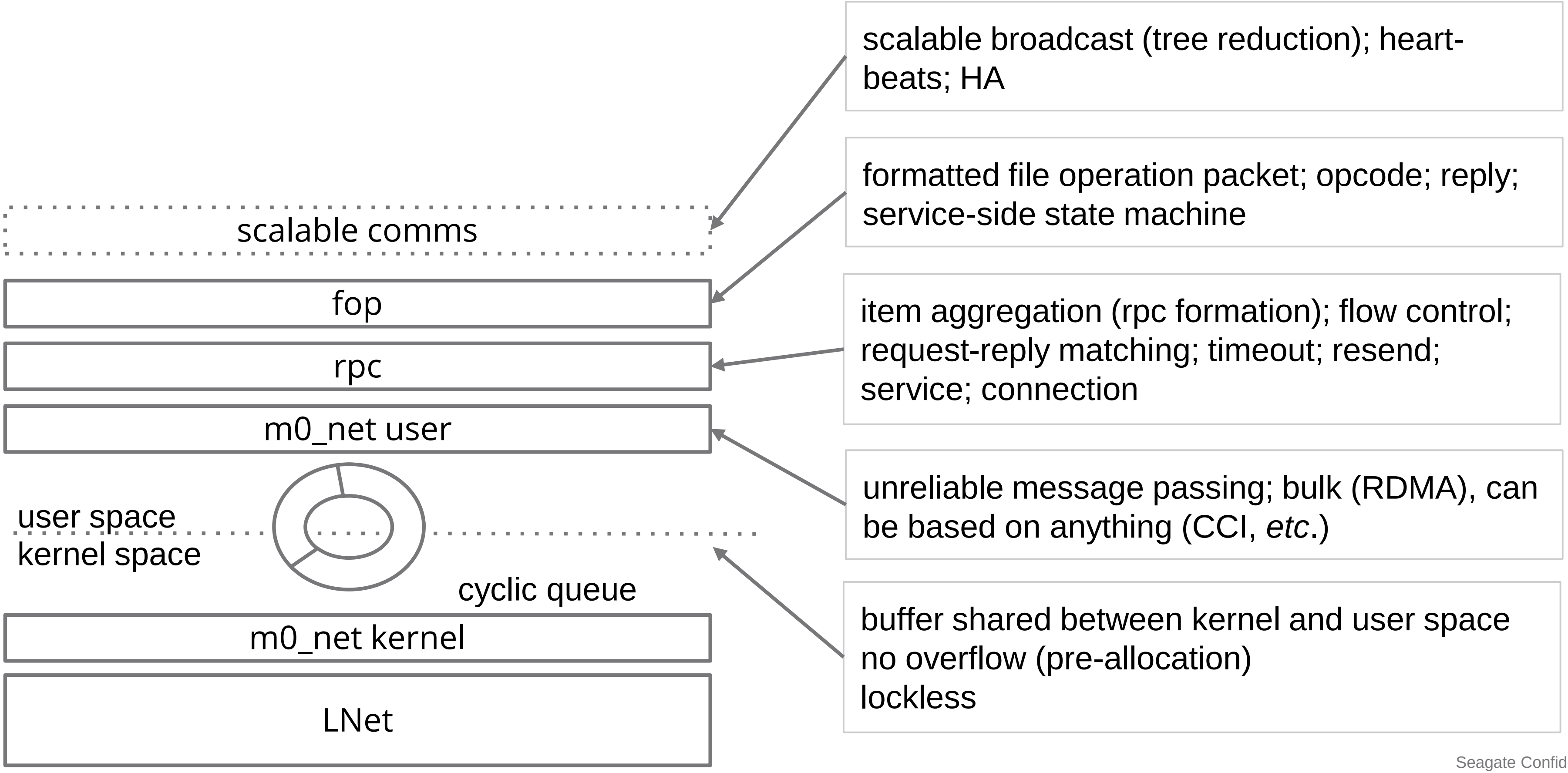
components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- **network, rpc, fop, HA**
- fom, reqh
- device io (stob)
- network raid repair
- security

comm

- network: LNet, 0-copy, unreliable message passing
- rpc
  - message packing (formation),
  - request-reply semantics
  - retransmit
- xcode: serialisation library
- fop: operation packet
- *reduce-broadcast*
  - communication and aggregation tree
  - provided by HA

comm



scalable comms

fop

rpc

m0_net user

user space

kernel space

cyclic queue

m0_net kernel

LNet

scalable broadcast (tree reduction); heart-beats; HA

formatted file operation packet; opcode; reply; service-side state machine

item aggregation (rpc formation); flow control; request-reply matching; timeout; resend; service; connection

unreliable message passing; bulk (RDMA), can be based on anything (CCI, *etc.*)

buffer shared between kernel and user space
no overflow (pre-allocation)
lockless

comm: xcode

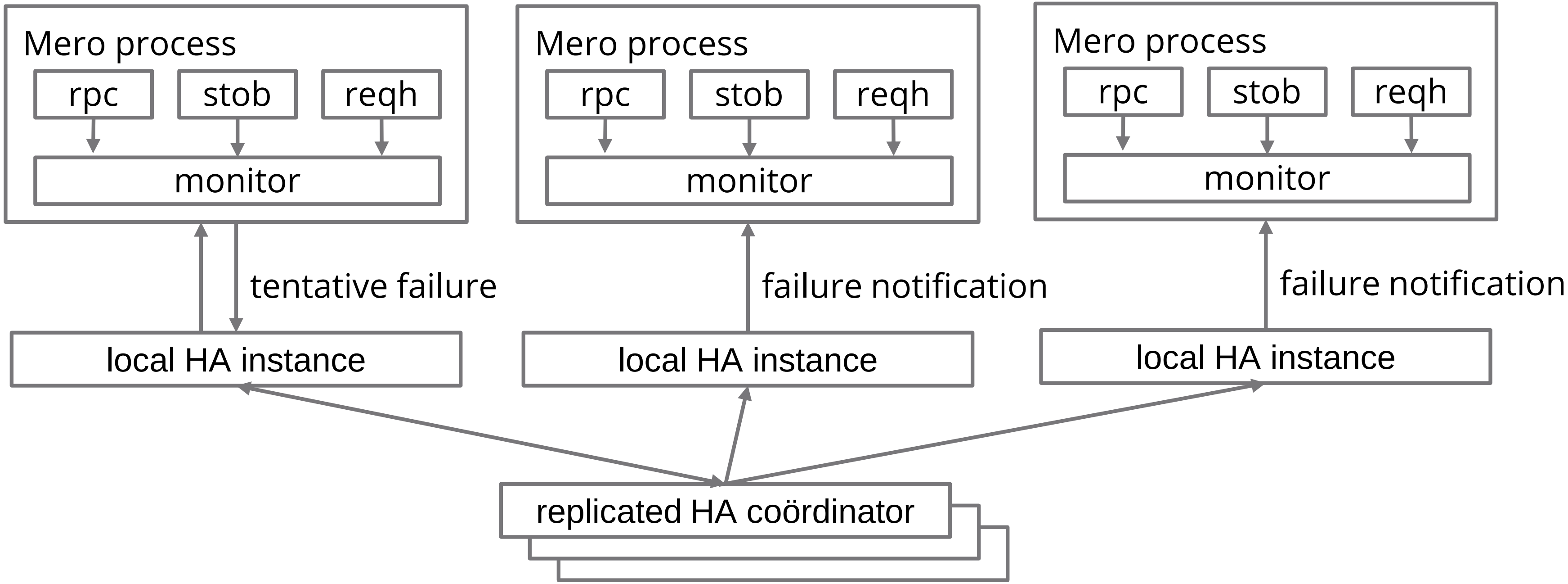
- a library adding a (modest) amount of reflection to C
- annotate header file foo.h
- gccxml: C parser
- generate foo_xc.h, foo_xc.c

```
struct m0_foo {
         uint32_t       f_nr;
         struct m0_bar *f_bar;
} M0_XCA_RECORD;
```

```
static struct m0_xcode_type m0_foo_xc = {
         .xct_name   = "m0_foo",
         .xct_sizeof = sizeof (struct m0_foo),
         .xct_children = {
                   [0] = { "f_nr",  &M0_XT_U32, offsetof(struct m0_foo, f_nr) },
                   [1] = { "f_bar", &m0_bar_xc, offsetof(struct m0_foo, f_bar) }
};
```

- m0_xcode_{en,de}code(), m0_xcode_{find,print,read}()

comm: HA

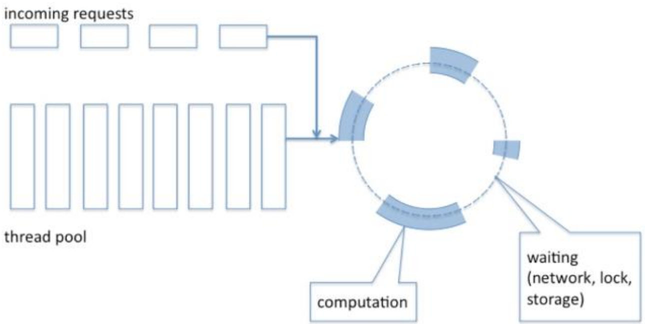

Mero process
rpc
stob
reqh
monitor
Mero process
rpc
stob
reqh
monitor
Mero process
rpc
stob
reqh
monitor
tentative failure
failure notification
failure notification
local HA instance
local HA instance
local HA instance
replicated HA coördinator

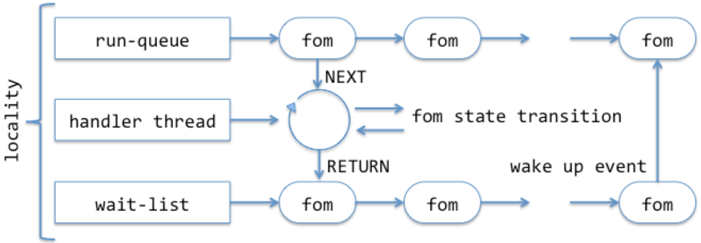
components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- **fom, reqh**
- device io (stob)
- network raid repair
- security

fom

- thread-per-request:
  - multiple cores, NUMA,
  - locking,
  - cache ping-pong,
  - c10K, many threads
- reqh:
  - thread per core
  - non-blocking scheduler
  - locality of reference
  - load balancing
  - long-term scheduling





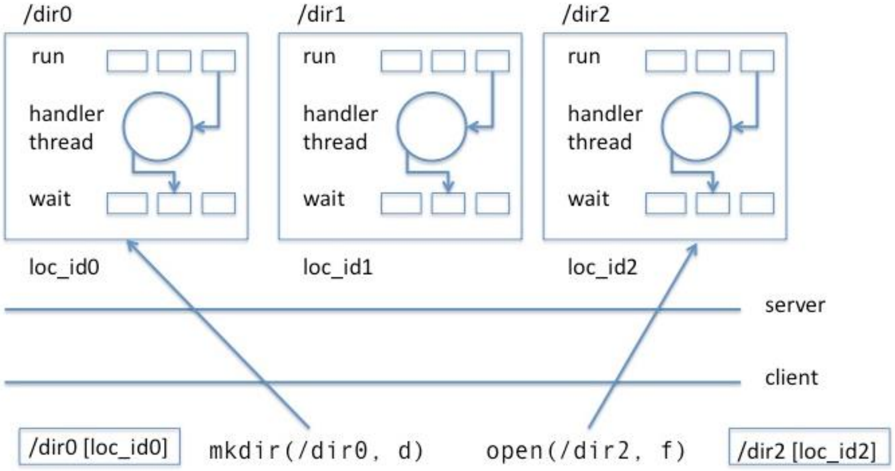
fom: request handler

```
while (!locality->shutdown) {
        while (have_work) {
                network_drain();
                stob_drain();
                fom_transition();
        }
        wait_for_work();
}
```

```
fom_transition(locality *l) {
        for_each(fom, l->run_queue) {
                good = goodness(l, fom);
                if (good > max_goodness) {
                             best = fom;
                             max_goodness =
fom_good;
                      }
        }
        best->state(); /* state transition */
}
```

```
goodness(locality *l, fom *f) {
        if (abs(fom->next_block_nr, l->elevator_pos) < threshold)
                result += 1;
        if (l->pending_rpc[fom->next_nid] > 0)
                result += 1;
        result += take_deadline_into_account(l, f);
        result += take_priority_into_account(l, f);
        ...
}
```

fom: load balancing



- each locality is a small server
- clients talk to particular locality, using opaque identifier
- change identifier for load-balancing

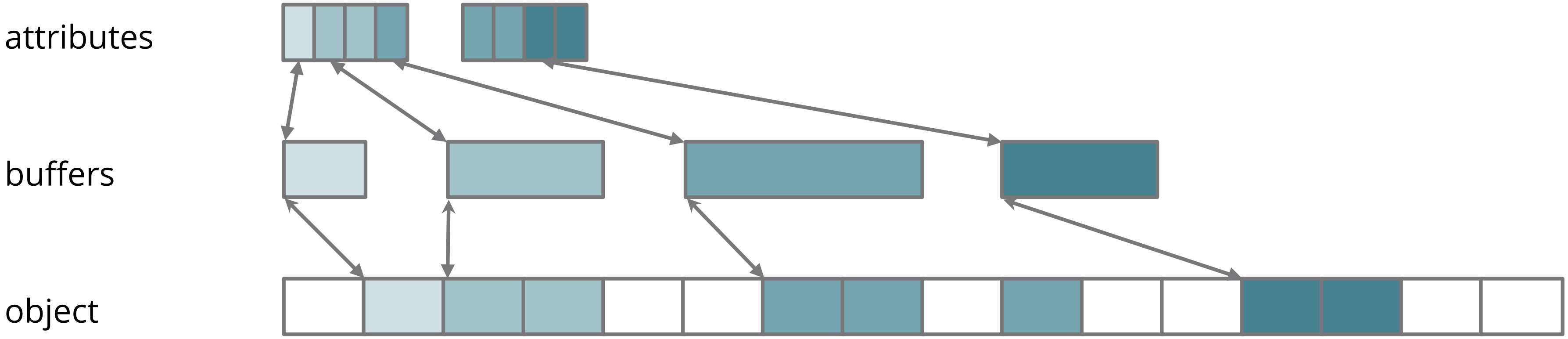
components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- **device io (stob)**
- network raid repair
- security

stob

- array of data-blocks, [0, 2^{64}), initially a hole
- create, delete, read, write, alloc, free operations
- IO at block granularity
- no usual meta-data (attributes, *etc*.)
- block attributes can be used for checksums, encryption keys, hash fingerprints
- scatter-gather-scatter operations: data and block attributes

stob: IO



stob: implementations

- linuxstob (*aka* devstob)
  - stob = file
  - aio
- adstob (allocation data stob)
  - multiple stobs stored in a backend stob
  - block allocator balloc
    - based on ext4 mballoc

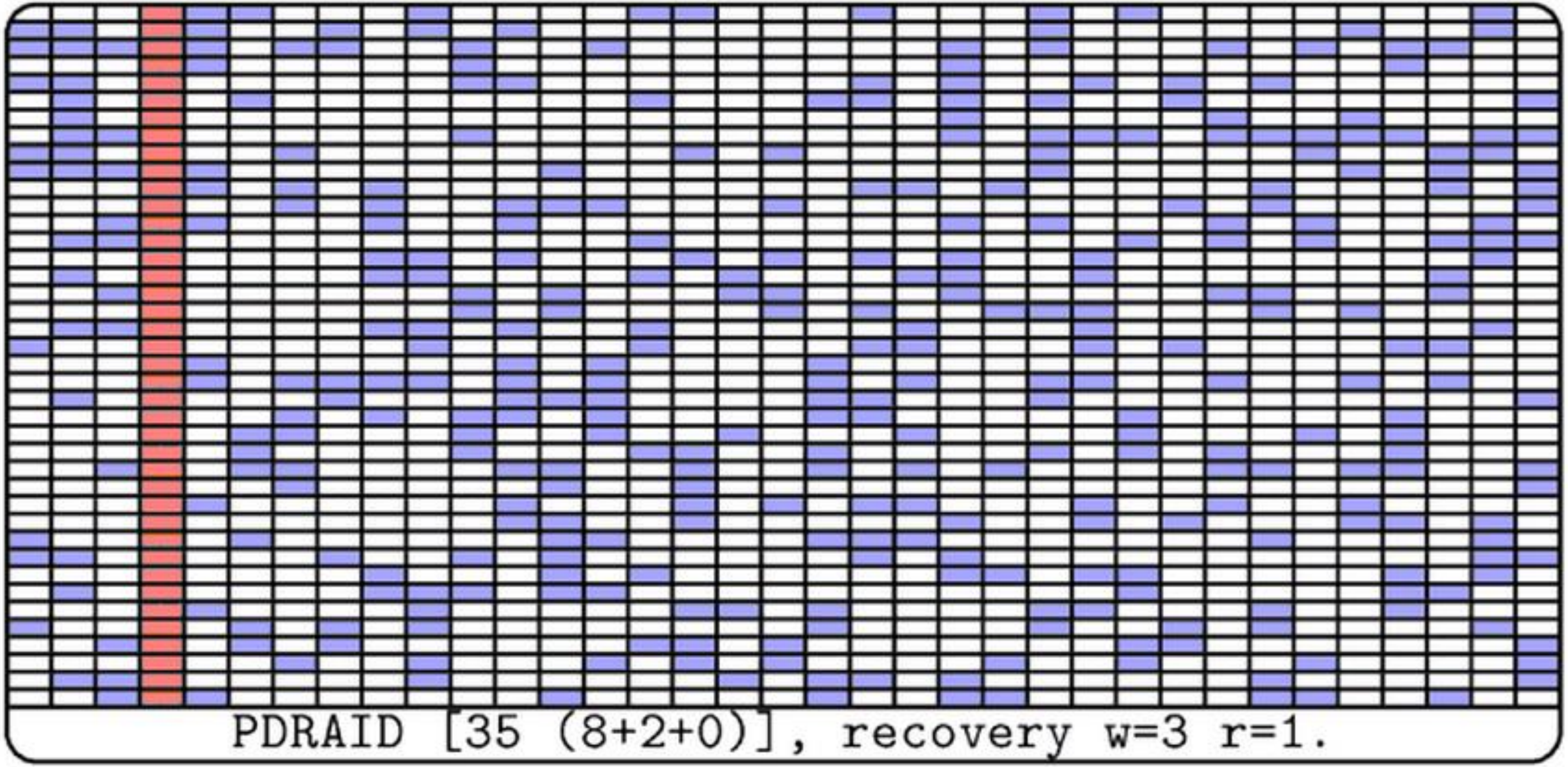
components

- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- **network raid repair**
- security

sns

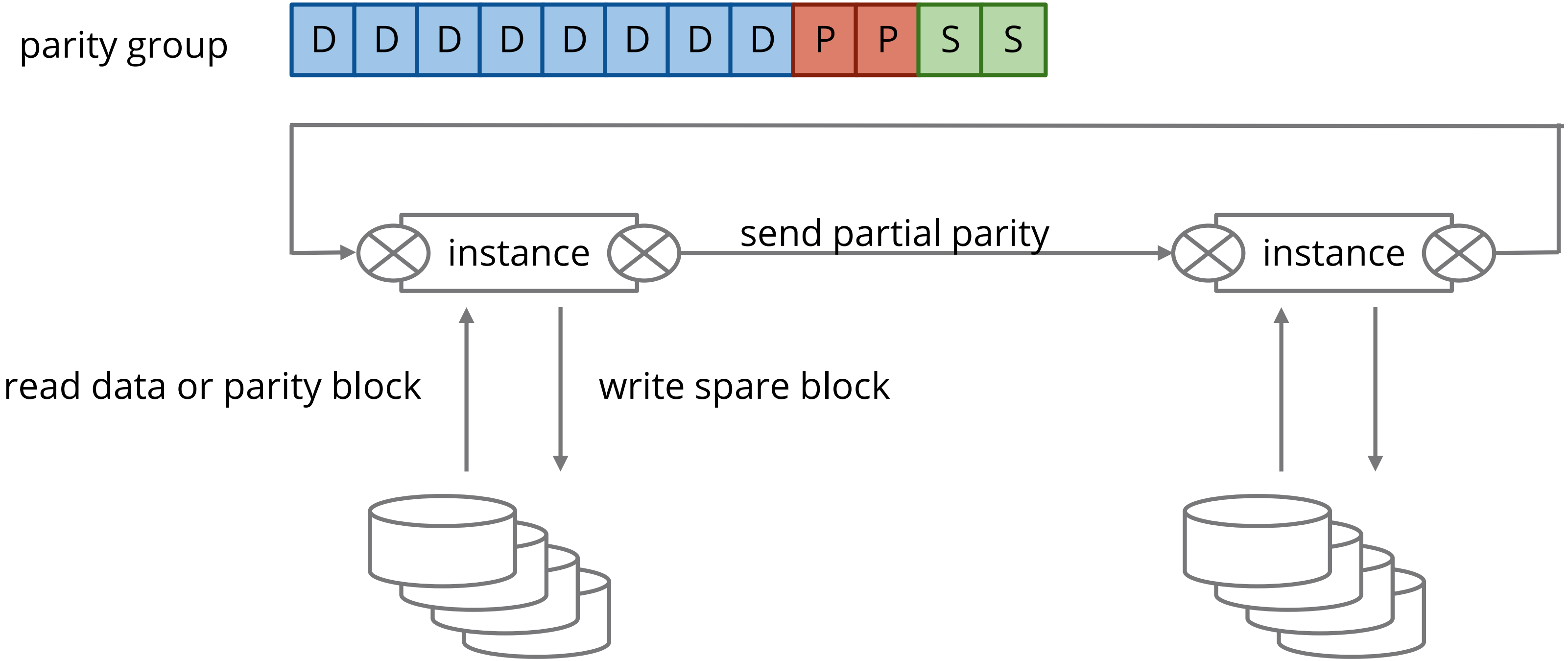
- guaranteed IO performance during repair
- fast repair
- copy machine
- repair
- rebalance
- pool
- *flattening*

sns: repair



PDRAID [35 (8+2+0)], recovery w=3 r=1.

sns: copy machine



parity group

D D D D D D D D P P S S

instance

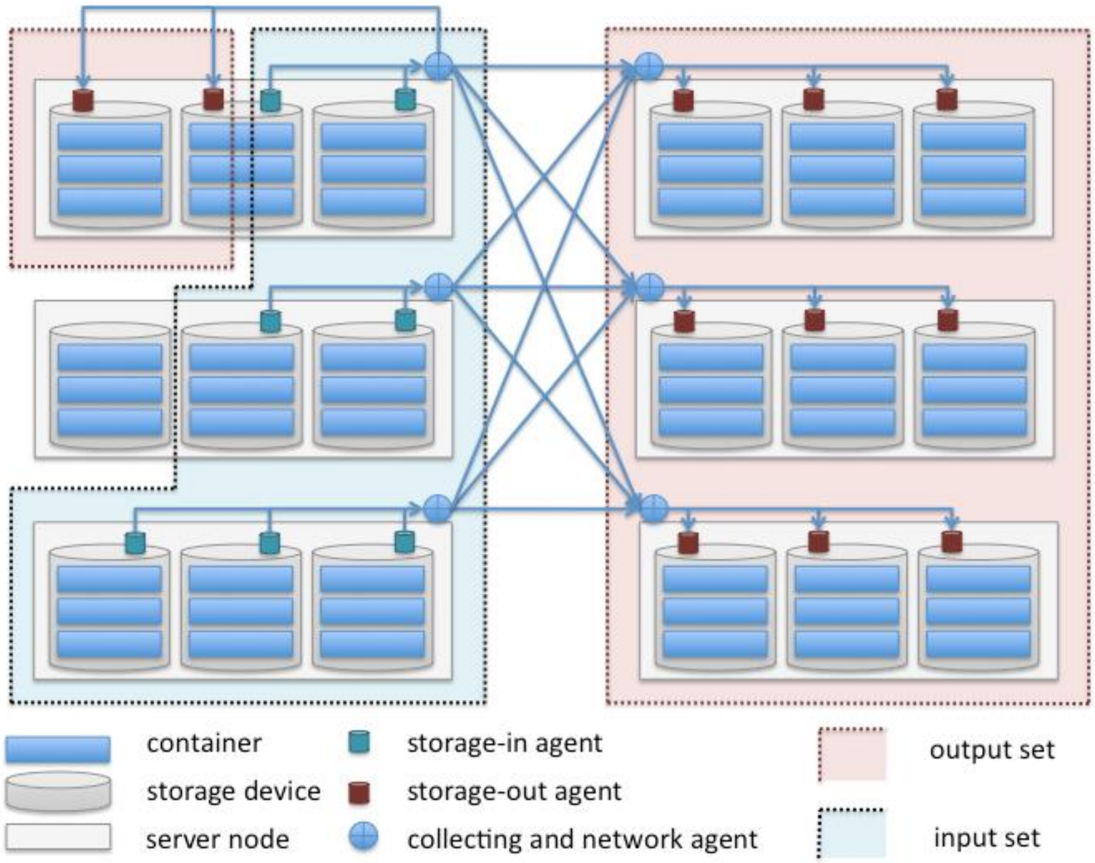
send partial parity

instance

read data or parity block

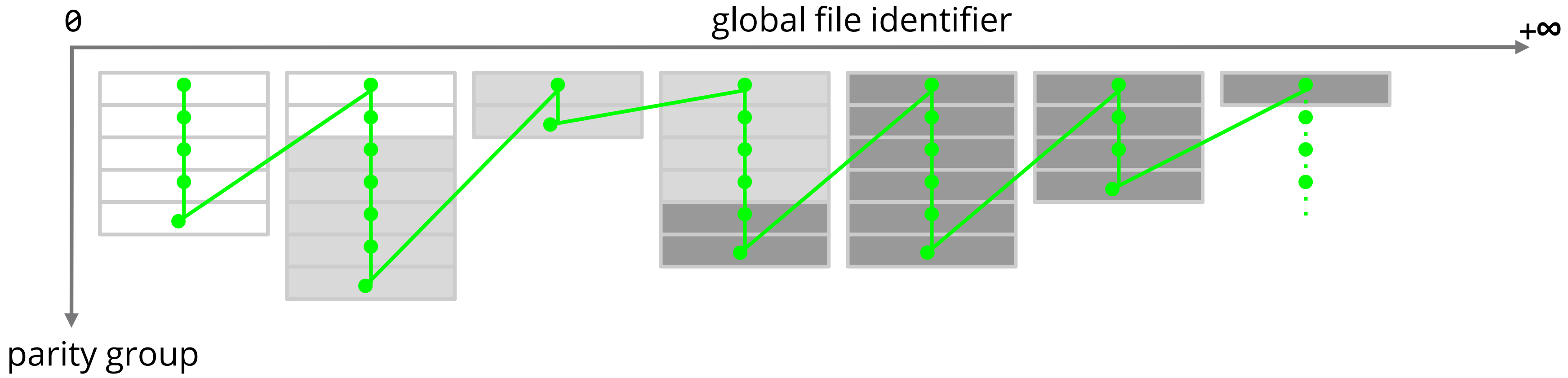
write spare block

sns: network


container
storage device
server node
storage-in agent
storage-out agent
collecting and network agent
output set
input set

sns: coördination

copy machine sliding window

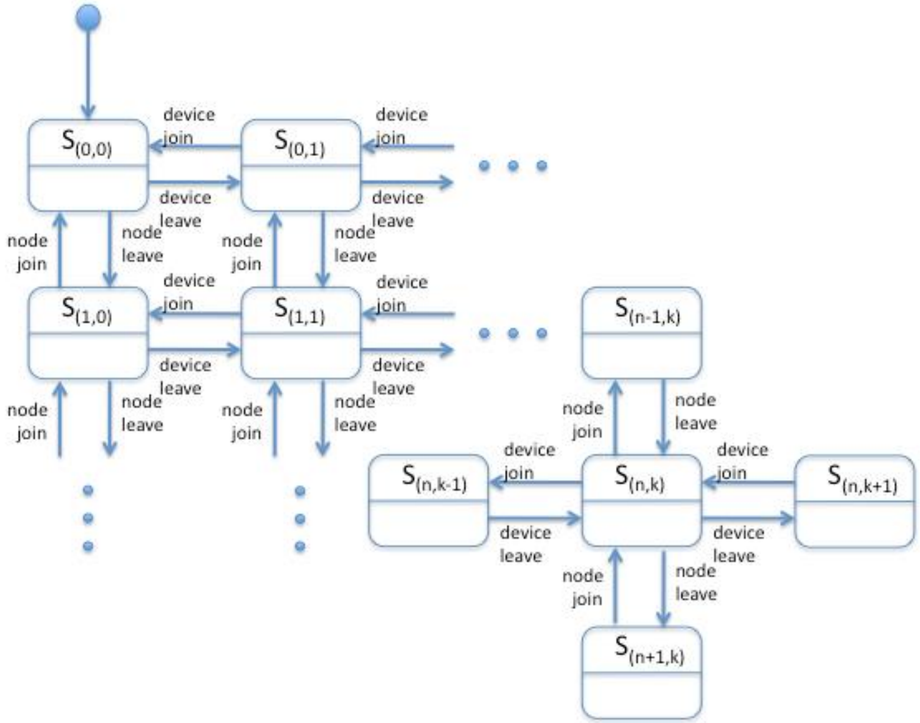


░ under repair

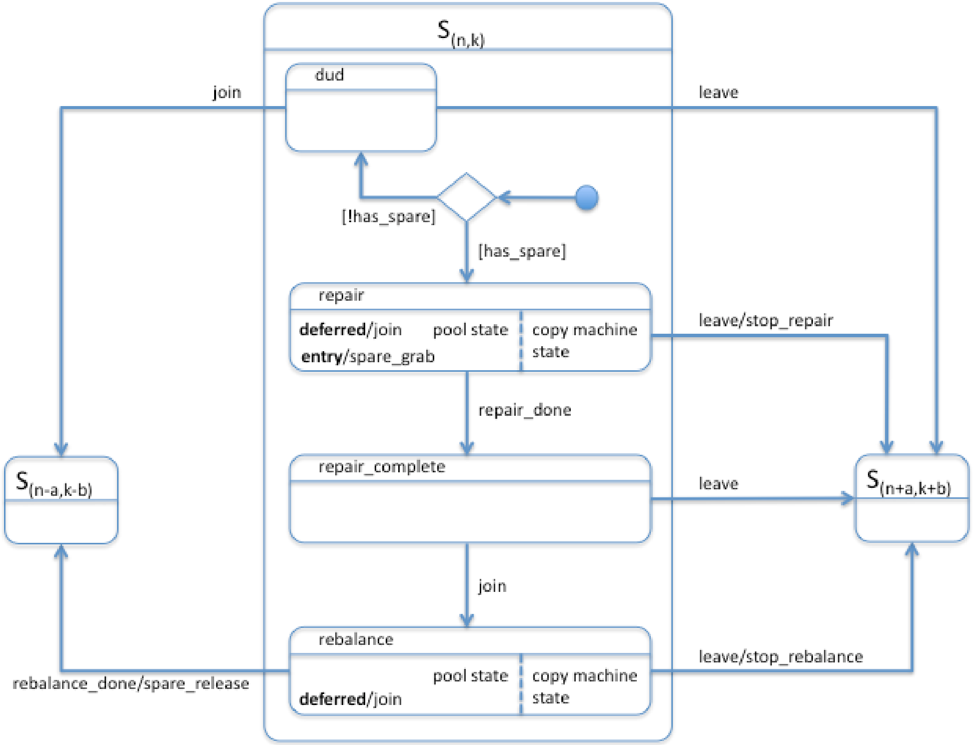
▓ still not repaired

□ repaired

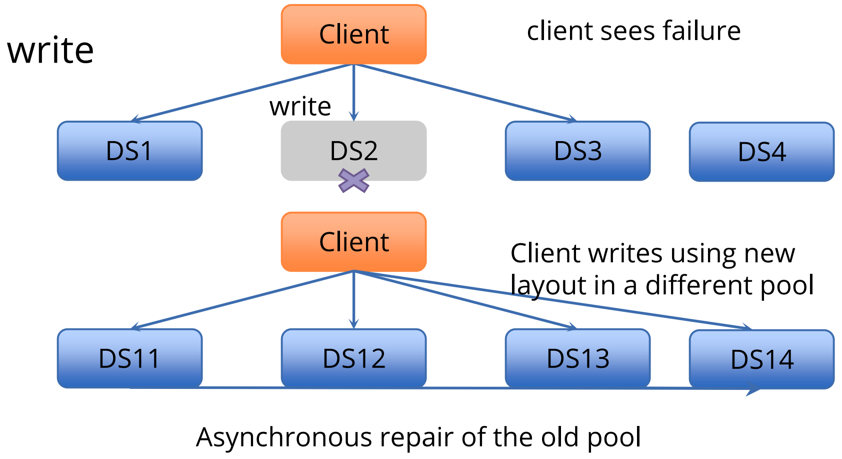
sns: failure state machine



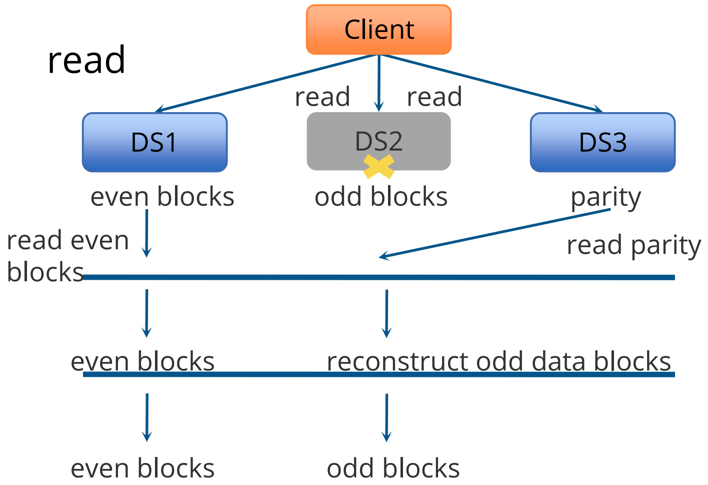
sns: failure in detail



sns: nba



write
Client
client sees failure
write
DS1
DS2
DS3
DS4
Client
Client writes using new layout in a different pool
DS11
DS12
DS13
DS14
Asynchronous repair of the old pool

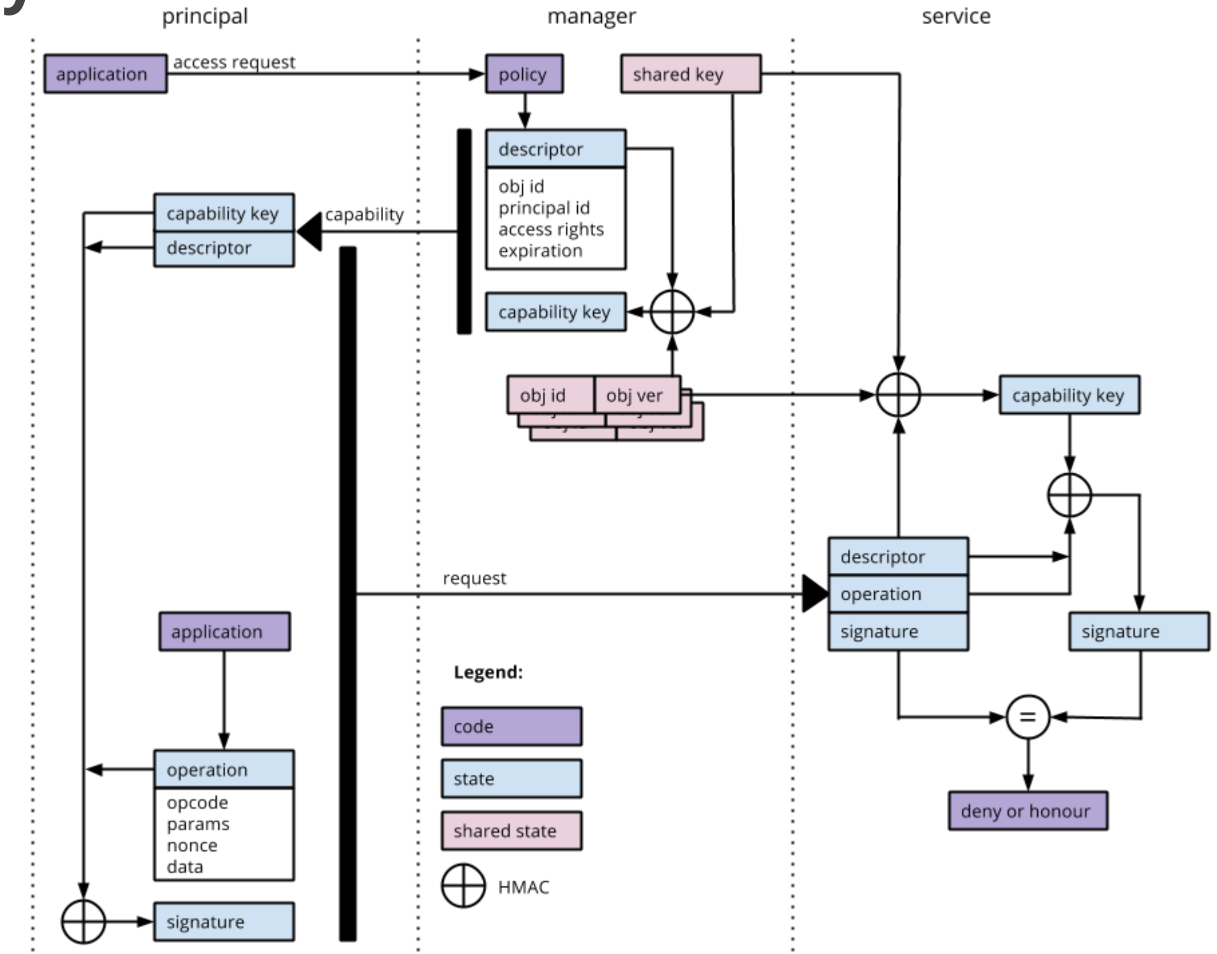


- client senses failure by timeout, notifies HA
- how new layout is selected? Layout formula
- composite layout with list of extents
- flattening: restore simple layout

components

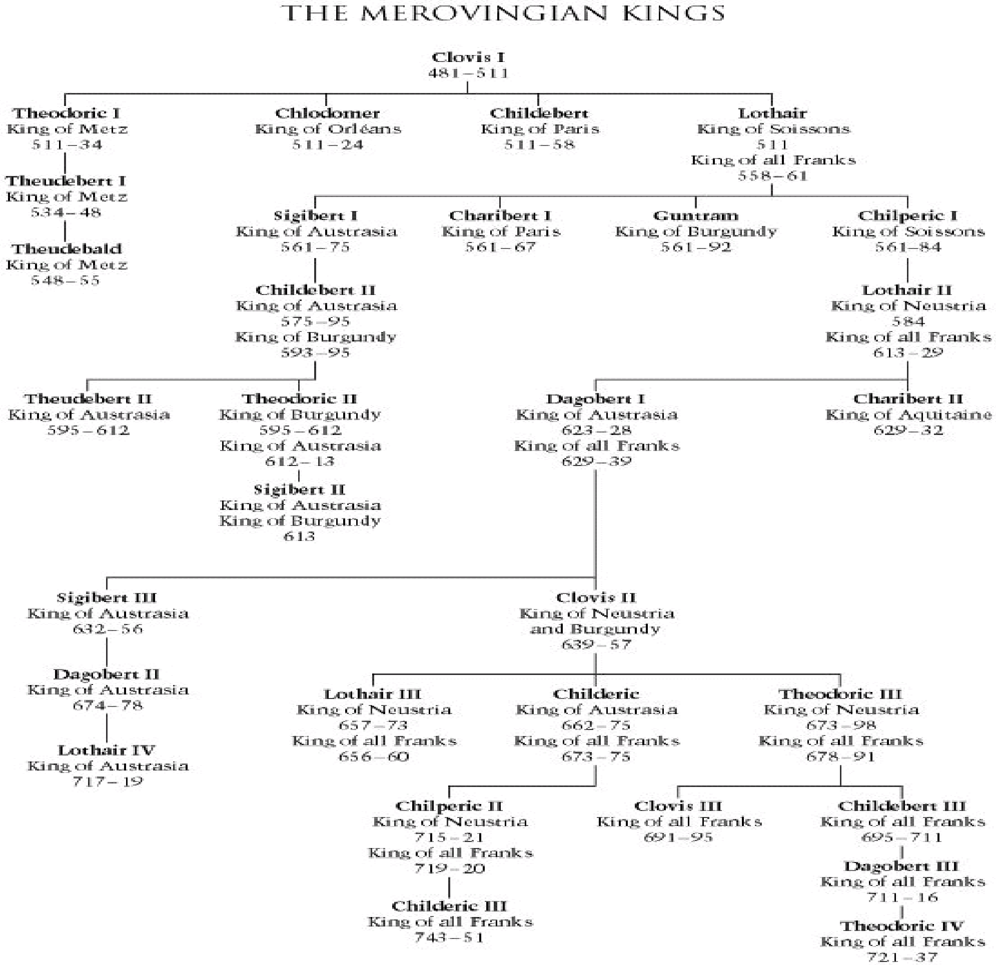
- clovis
- transactions (dtm)
- resource manager
- fdmi
- addb
- network raid, layouts
- containers
- function shipping
- lingua franca
- integrity checking
- meta-data back-end
- network, rpc, fop, HA
- fom, reqh
- device io (stob)
- network raid repair
- **security**

security



principal
manager
service
application
access request
policy
shared key
descriptor
obj id
principal id
access rights
expiration
capability key
descriptor
capability
capability key
obj id
obj ver
capability key
request
descriptor
operation
signature
signature
=
deny or honour
application
operation
opcode
params
nonce
data
signature
Legend:
code
state
shared state
HMAC

questions?

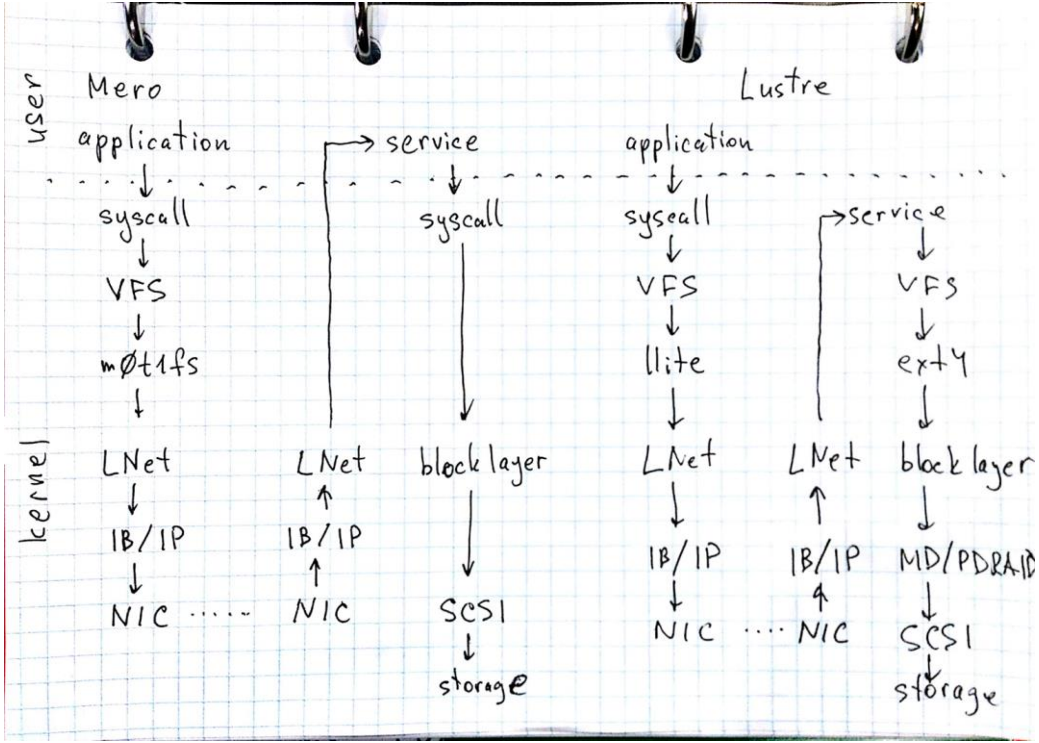


THE MEROVINGIAN KINGS

use cases

- IO data-flow end-to-end
- lookup path through all layers
- LOMO/WOMO: server and client failure

data-flow


user
kernel
Mero
Lustre
application
service
application
syscall
syscall
syscall
service
VFS
VFS
VFS
mØt1fs
llite
ext4
LNet
LNet
block layer
LNet
LNet
block layer
IB/IP
IB/IP
IB/IP
IB/IP
MD/PDRAID
NIC
NIC
SCSI
NIC
NIC
SCSI
storage
storage

lookup: m0t1fs

```
$ ls -l foo
```

- `stat("foo", &stbuf);`
  - `m0t1fs_lookup(dir, dentry, nd)`
    - `fop = m0_fop_alloc(&m0_fop_lookup_fopt);`
    - `m0_rpc_post(fop)`
    - `m0_rpc_item_wait_for_reply(fop)`
    - `m0t1fs_iget()`

lookup: rpc out

- `m0_rpc_post(fop)`
    - `m0_rpc_item_send()`
        - `m0_rpc_item_start_timer()`
        - `m0_rpc_frm_enq_item()`
            - `frm_insert()`
                - **`queue add [URGENT, BOUND]`**
            - `frm_balance()`
                - **`if (ready) frm_fill_packet()`**
                    - `m0_rpc_packet_add_item()`
                - **`frm_packet_ready()`**
                    - `m0_net_buffer_add()`

lookup: rpc in

- buf_recv_cb() (= [M0_NET_QT_MSG_RECV])
    - net_buf_received()
        - packet_received()
            - item_received()
                - **m0_rpc_slot_reply_received()**
                    - item_find(item_xid(reply))
                    - req->rio_replied()
                    - m0_rpc_item_change_state(req, M0_RPC_ITEM_REPLIED)

- m0_rpc_item_wait_for_reply()

lookup: network out

- m0_net_buffer_add()
  - m0_net_tm_tlink_init_at_tail(buf, ql)
  - nx_ops->xo_buf_add(buf) (= nlx_xo_buf_add)
    - M0_NET_QT_MSG_SEND
    - nlx_core_buf_msg_send()
      - nlx_ucore_ioctl(M0_LNET_BUF_MSG_SEND)

lookup: network in

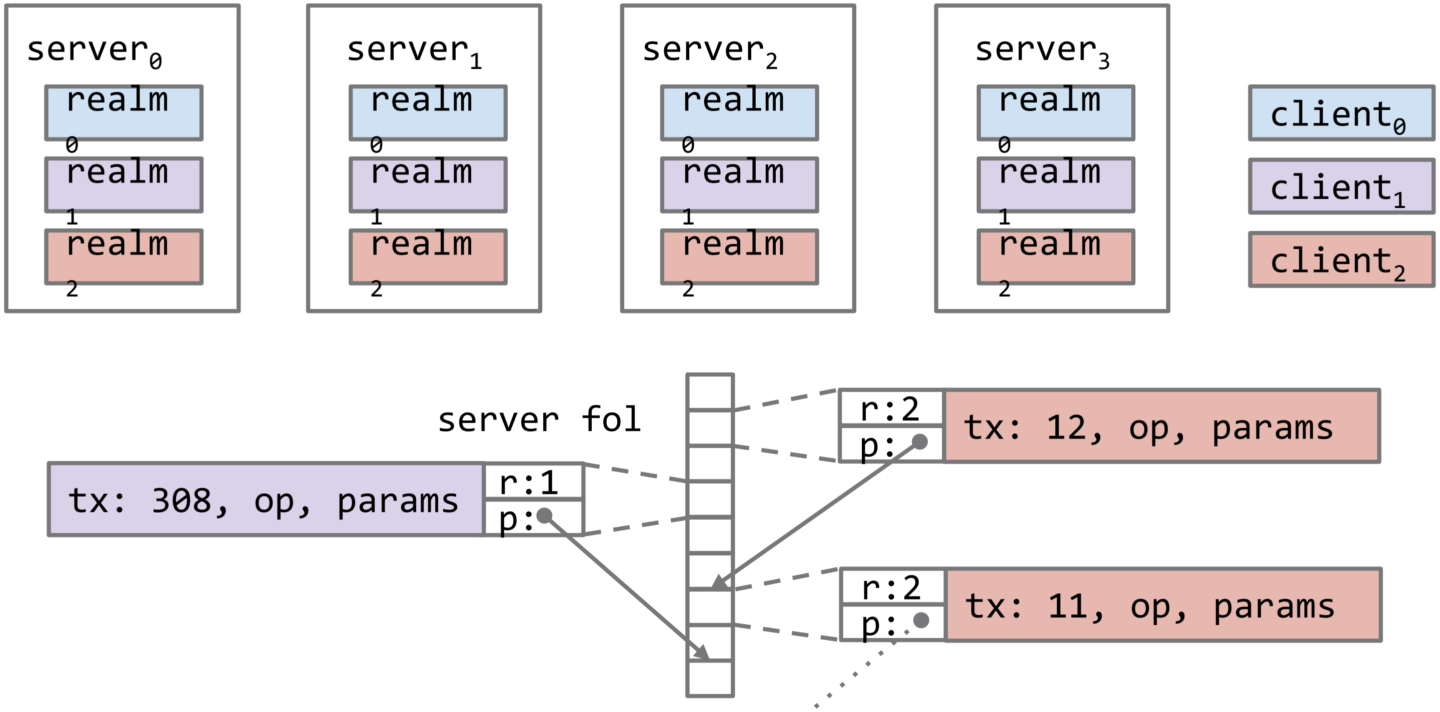
- nlx_tm_ev_worker()
    - !queue.empty() or semaphore_down()
    - queue get
    - m0_net_tm_event_post()
        - **buf_recv_cb() (= [M0_NET_QT_MSG_RECV])**

lookup: service side

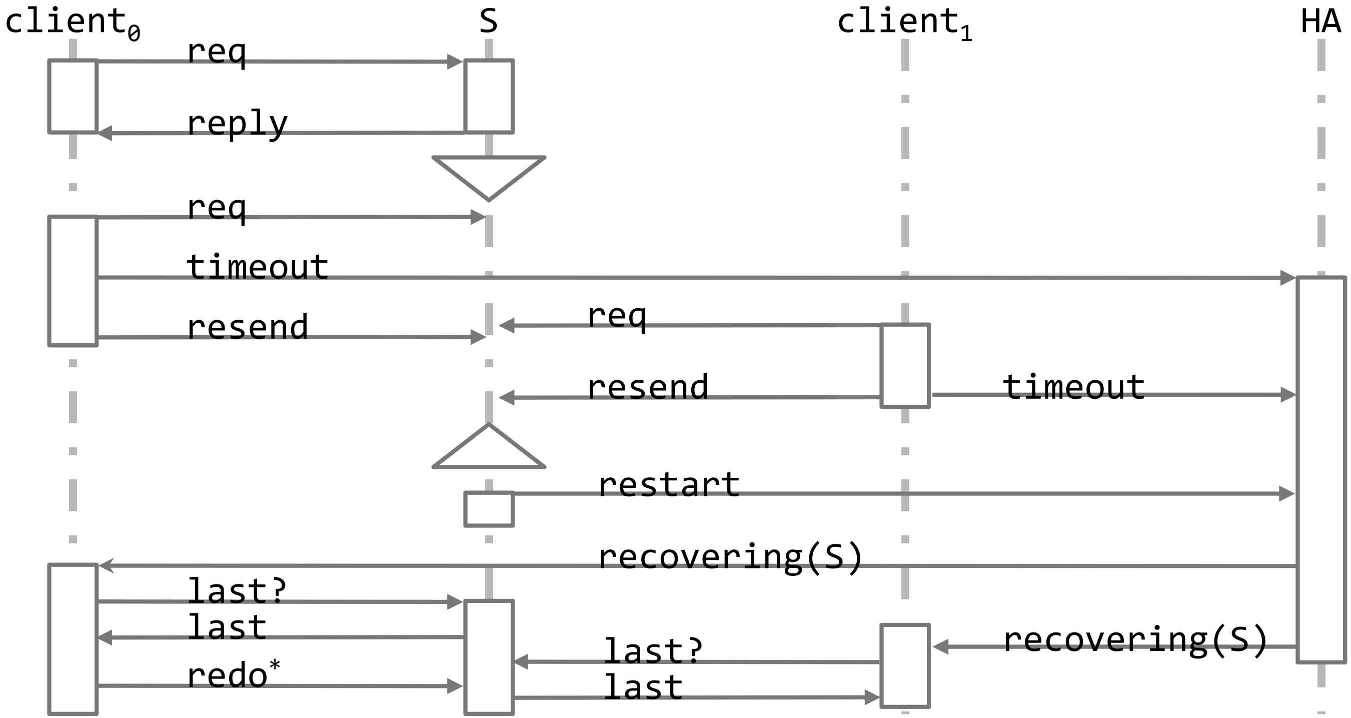
- m0_rpc_item_dispatch()
  - m0_reqh_fop_handle()
    - **fop->fto_create(fop, &fom)**
    - **m0_fom_queue(fom)**

- loc_handler_thread()
  - fom_exec(fom)
    - **fom->fo_ops->fo_tick(fom)**
      - m0_md_tick_getattr()
        - **m0_fom_tick_generic()**
        - **m0_cob_locate()**
          - m0_be_btree_lookup()

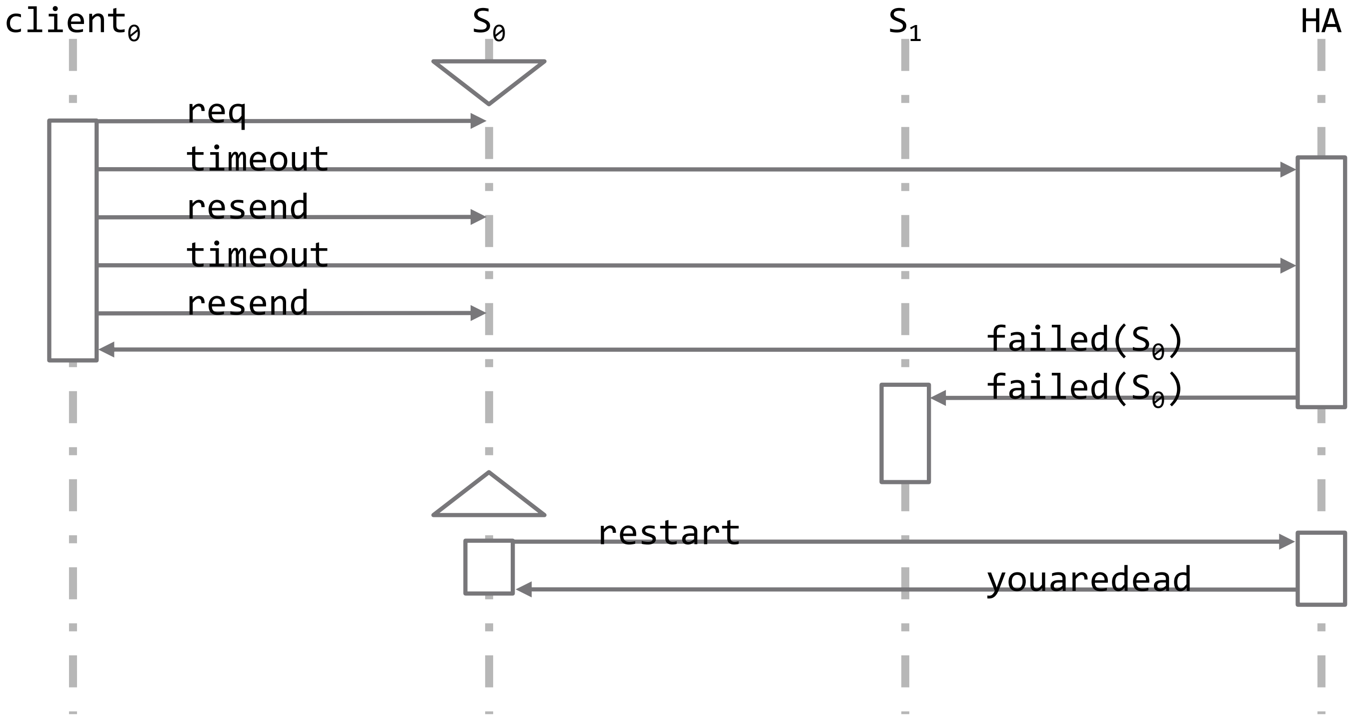
LOMO/WOMO realms



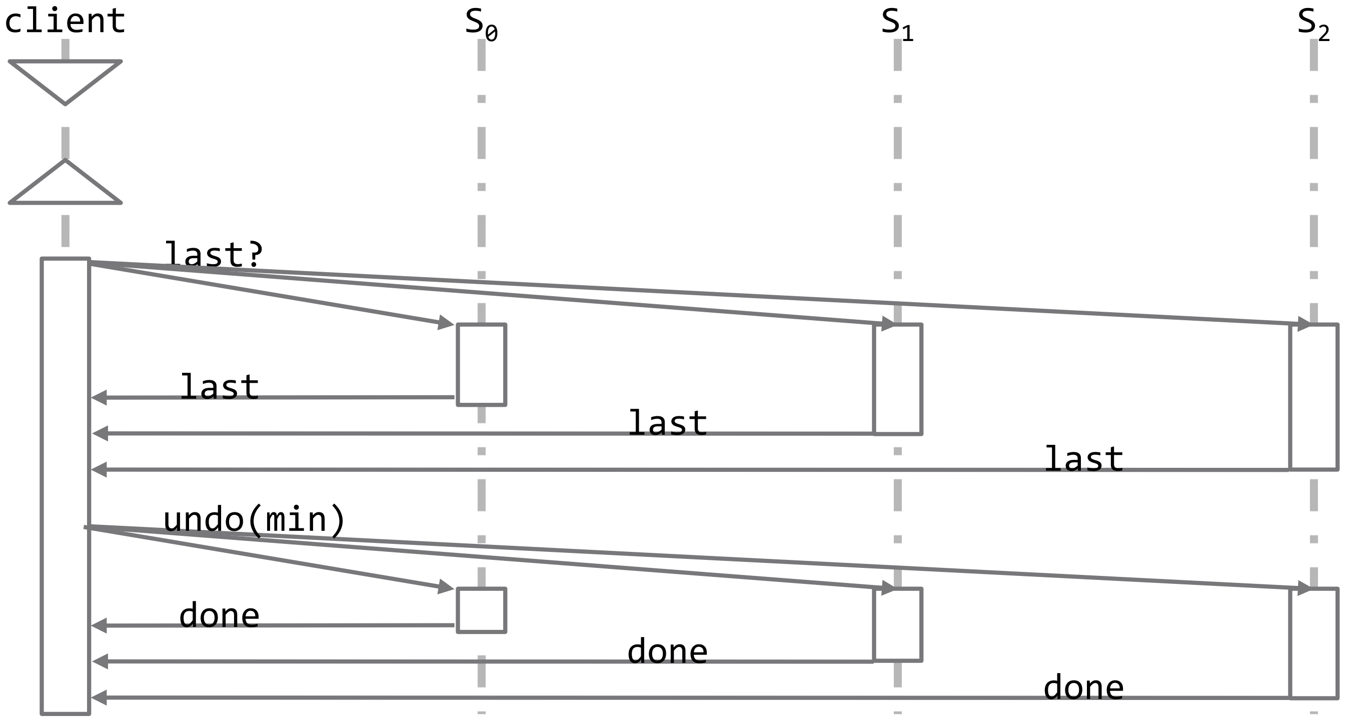
transient server failure



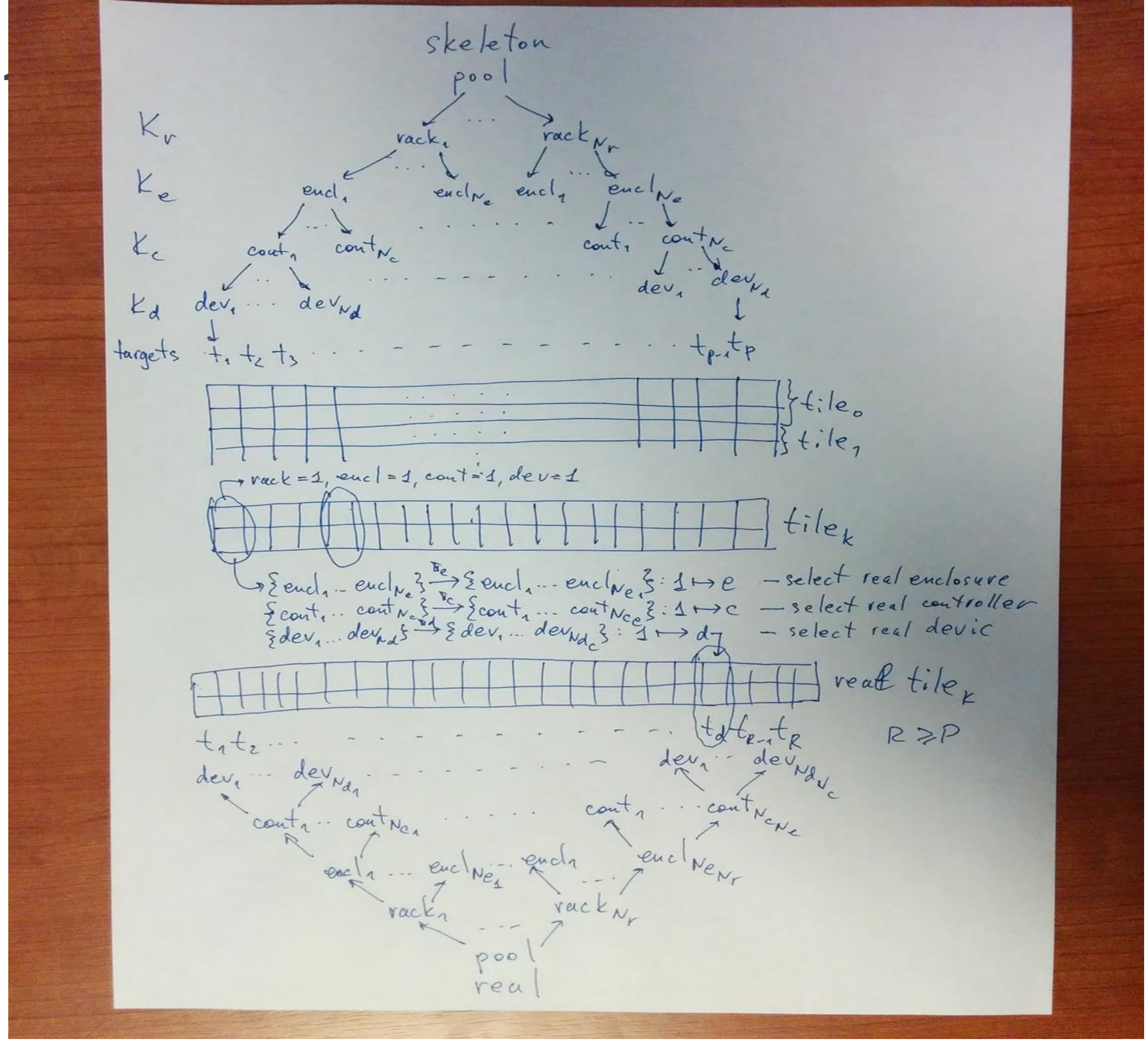
permanent server failure



client failure



io:


skeleton
pool
K_r
K_e
K_c
K_d
$rack_1$
$rack_{N_r}$
$encl_1$
$encl_{N_e}$
$cont_1$
$cont_{N_c}$
dev_1
dev_{N_d}
targets
t_1 t_2 t_3
t_{P-1} t_P
$tile_0$
$tile_1$
rack = 1, encl = 1, cont = 1, dev = 1
$tile_k$
$\{encl_1 .. encl_{N_e}\} \xrightarrow{\pi_e} \{encl_1 ... encl_{N_{e_1}}\} : 1 \mapsto e$ — select real enclosure
$\{cont_1 .. cont_{N_c}\} \xrightarrow{\pi_c} \{cont_1 ... cont_{N_{ce}}\} : 1 \mapsto c$ — select real controller
$\{dev_1 ... dev_{N_d}\} \xrightarrow{\pi_d} \{dev_1 ... dev_{N_{d_c}}\} : 1 \mapsto d$ — select real devic
real tile$_k$
t_1 t_2
t_d t_{R-1} t_R
$R \geq P$
dev_1
$dev_{N_{d_1}}$
$dev_{N_{dN_c}}$
$cont_1$
$cont_{N_{e_1}}$
$cont_{N_{cN_e}}$
$encl_1$
$encl_{N_{e_1}}$
$encl_{N_{eN_r}}$
$rack_1$
$rack_{N_r}$
pool
real