Dynomite

Yet Another Distributed Key Value Store

@moonpolysoft - questions

Dynomite!

A Crowded Field

- Cassandra
- Lightcloud
- Memcachedb
- Redis
- Tokyo Tyrannical Cabinet Device Thing
- Voldemort

Who here has written one?

Alpha - 0.6.0

Focus on Distribution

Focus on Performance

- Latency
  - Average ~ 10 ms
  - Median ~ 5 ms
  - 99.9% ~ 1 s

- Throughput
  - R12B ~ 2,000 req/s
  - R13B ~ 6,500 req/s

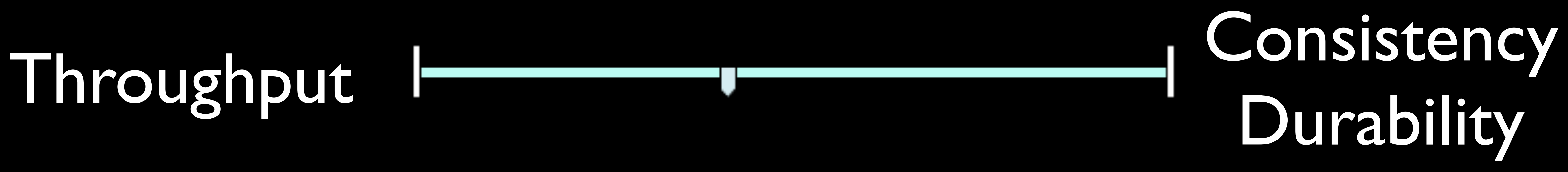
At Powerset

- 12 machine production cluster
- ~6 million images + metadata
- ~2 TB of data including replicas
- ~139KB average size

Production?

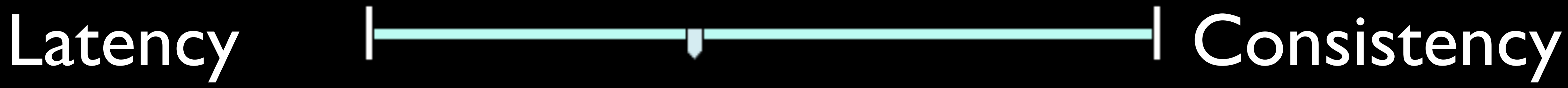
- Data you can afford to lose
- Compatibility will break
- Migration will be provided

The Constants



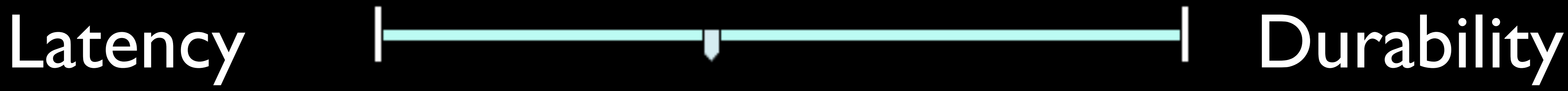
N – Replication

Max Replicas per Partition



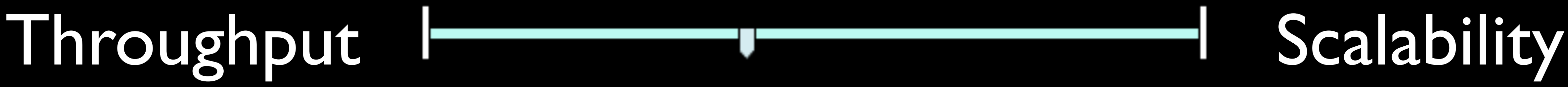
R – Read Quorum

Minimum participation for read



W – Write Quorum

Minimum participation for write



Q – Partitioning

Partitions = 2^Q

QNRW – Defines your cluster

At Powerset

- Batch writes
- Online reads
- Wikipedia pages are obese

Q – 6
N – 3
R – 1
W – 2

```
(dynomite@galva)2> mediator:put(
  "prefs:user:merv",
  undefined,
  term_to_binary(
    [{safe_search, false}, {results, 50}])).
```

```
(dynomite@galva)2> mediator:put(
  "prefs:user:merv",
  undefined,
  term_to_binary(
    [{safe_search, false}, {results, 50}])).
```

```
(dynomite@galva)2> mediator:put(
  "prefs:user:merv",
  undefined,
  term_to_binary(
    [{safe_search, false}, {results, 50}])).
```

```
(dynomite@galva)2> mediator:put(
  "prefs:user:merv",
  undefined,
  term_to_binary(
    [{safe_search, false}, {results, 50}])).
```

Value is always Binary

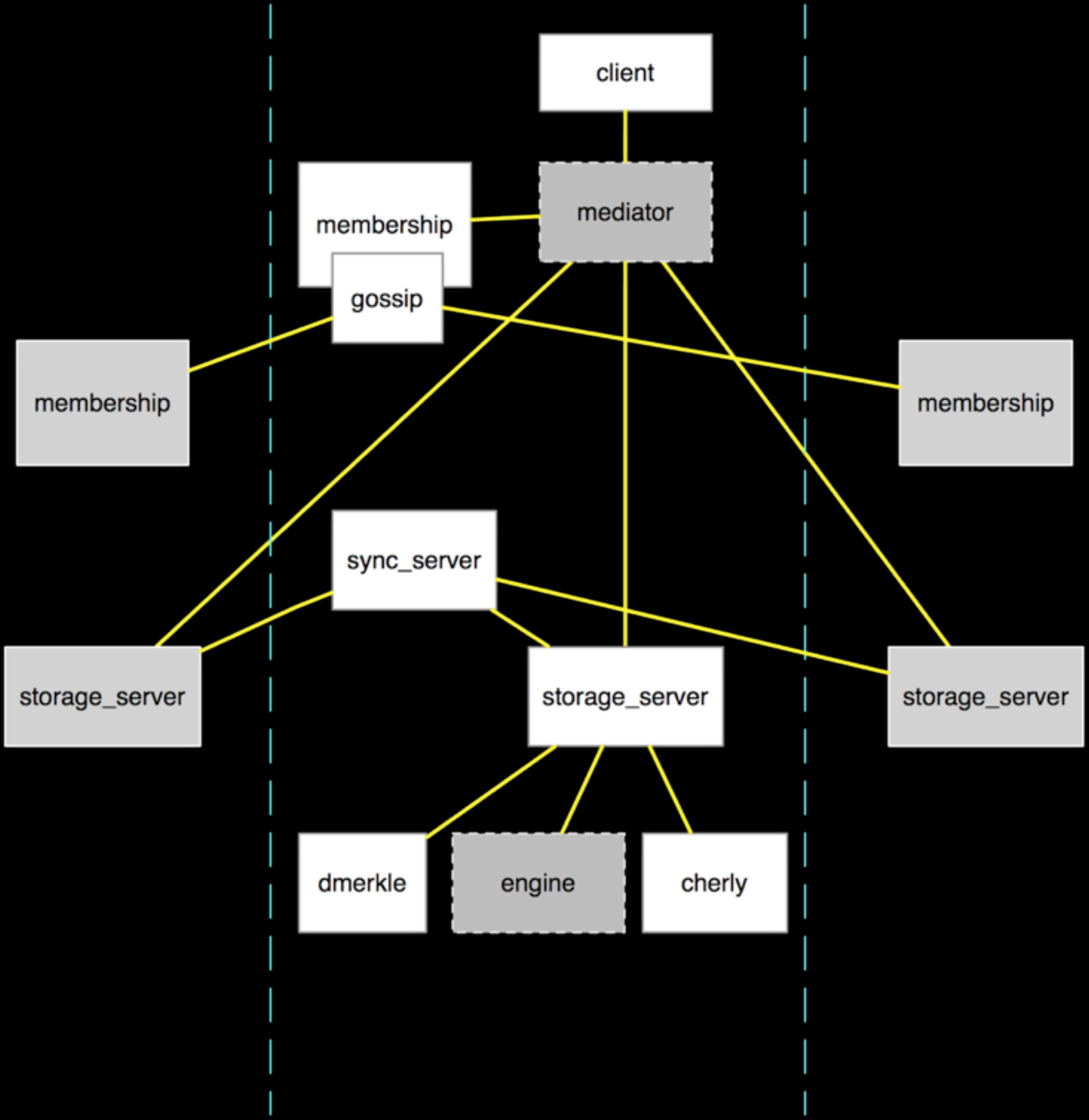
```
(dynomite@galva)2> mediator:put(
  "prefs:user:merv",
  undefined,
  term_to_binary(
    [{safe_search, false}, {results, 50}])).
```

```
(dynomite@galva)1> {ok, {Context, [Bin]}} =
mediator:get("prefs:user:merv").

{ok,{[{"<0.630.0>",1240543271.698089}],
      [<<131,108,0,0,0,2,...>>]}}

(dynomite@galva)2> Terms = binary_to_term(Bin).

[{safe_search,false},{results,50}]
```

```
(dynomite@galva)1> {ok, {Context, [Bin]}} =
mediator:get("prefs:user:merv").

{ok,{[{"<0.630.0>",1240543271.698089}],
      [<<131,108,0,0,0,2,...>>]}}

(dynomite@galva)2> Terms = binary_to_term(Bin).

[{safe_search,false},{results,50}]
```

```
mediator:delete/1
mediator:has_key/1
```

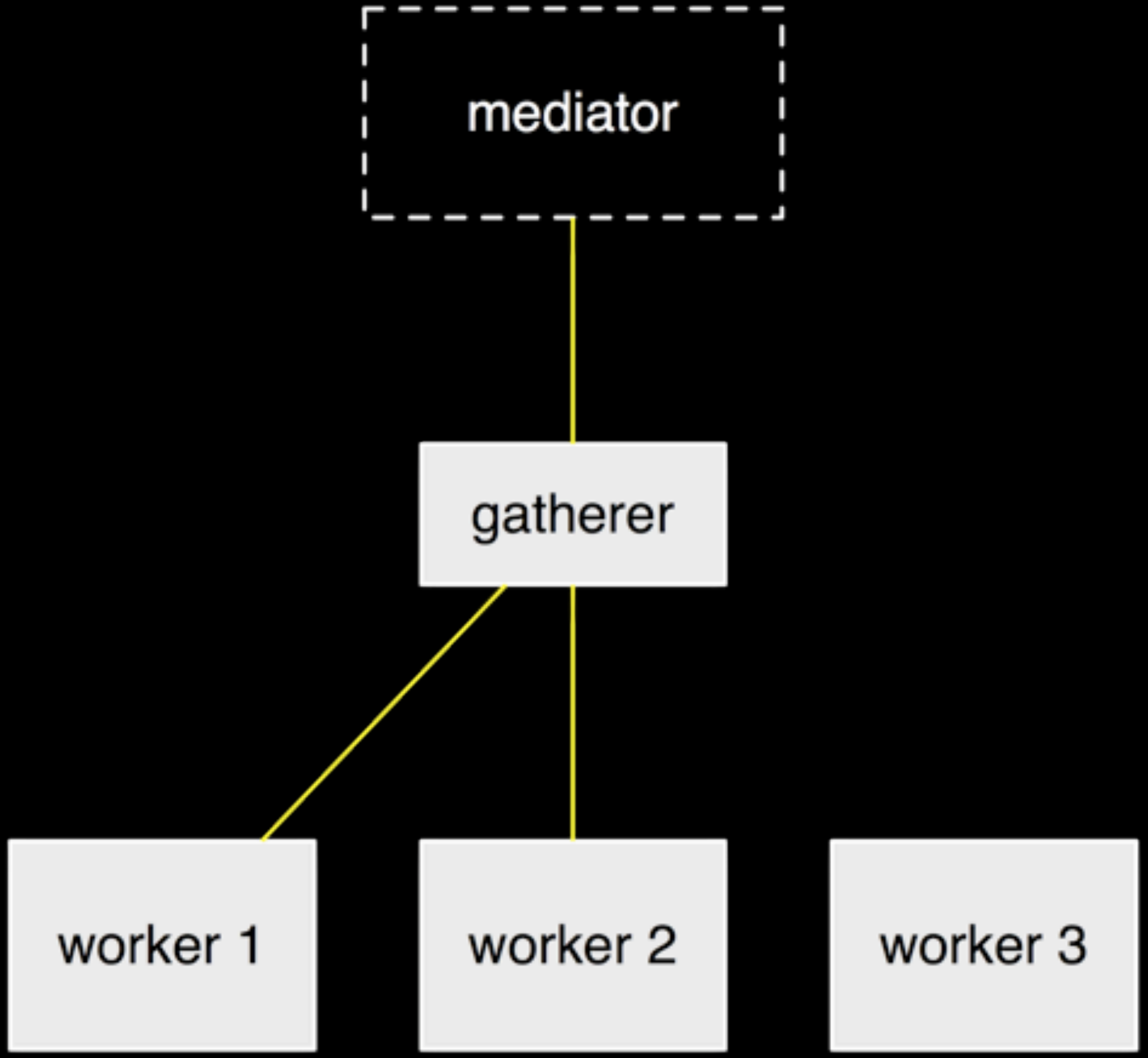
Up In Dem Gutz


client
membership
mediator
gossip
membership
membership
sync_server
storage_server
storage_server
storage_server
dmerkle
engine
cherly

Client Protocols

- Native Erlang Messages
- Thrift
- Protocol Buffers
- Ascii Protocol

Mediator



N – 3 ; R – 2

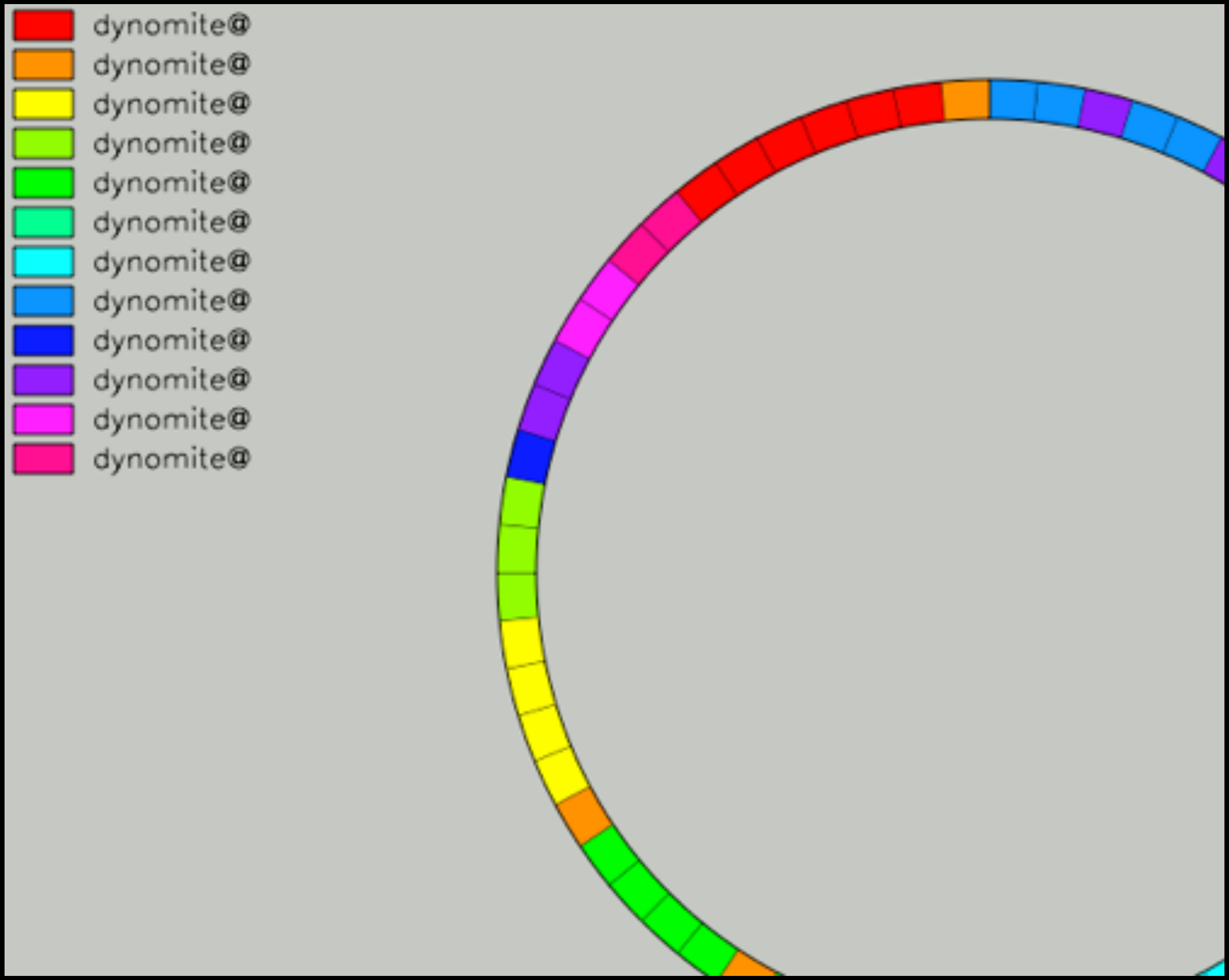
```
pmap(Fun, List, ReturnNum) ->
  N = if
    ReturnNum > length(List) -> length(List);
    true -> ReturnNum
  end,
  SuperParent = self(),
  SuperRef = erlang:make_ref(),
  Ref = erlang:make_ref(),
  %% we spawn an intermediary to collect the results
  %% this is so that there will be no leaked messages sitting in our mailbox
  Parent = spawn(fun() ->
      L = gather(N, length(List), Ref, []),
      SuperParent ! {SuperRef, pmap_sort(List, L)}
    end),
  Pids = [spawn(fun() ->
      Parent ! {Ref, {Elem, (catch Fun(Elem))}}
    end) || Elem <- List],
  Ret = receive
    {SuperRef, Ret} -> Ret
  end,
  % i think we need to cleanup here.
  lists:foreach(fun(P) -> exit(P, die) end, Pids),
```

Membership

Handles Nodes Joining

```
membership:nodes() =/= nodes().
```

- Receive join notification
- Recalculate partition to node mapping
- Start bootstrap routines
- Stops old local storage servers
- Starts new local storage servers
- Install the new partition table



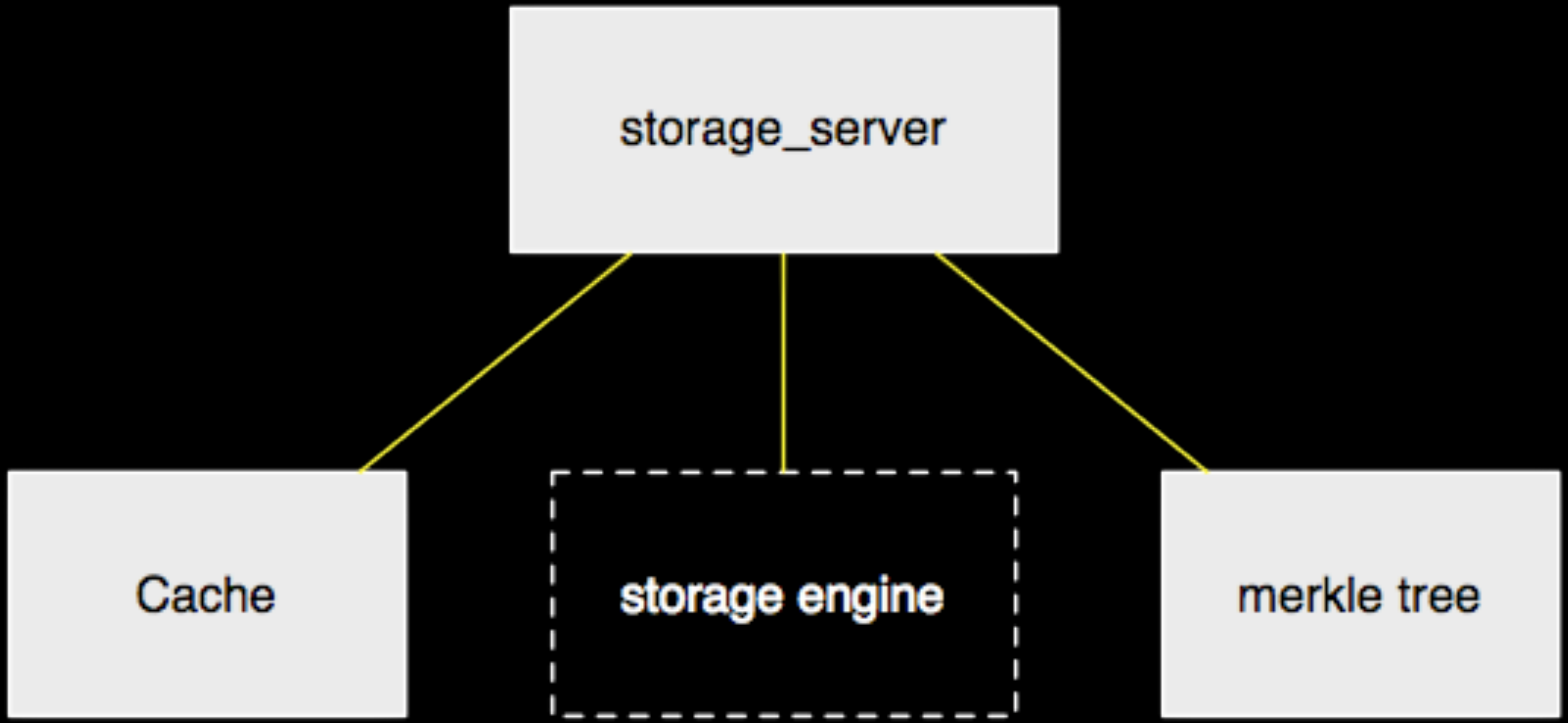
Maps Partitions To Nodes

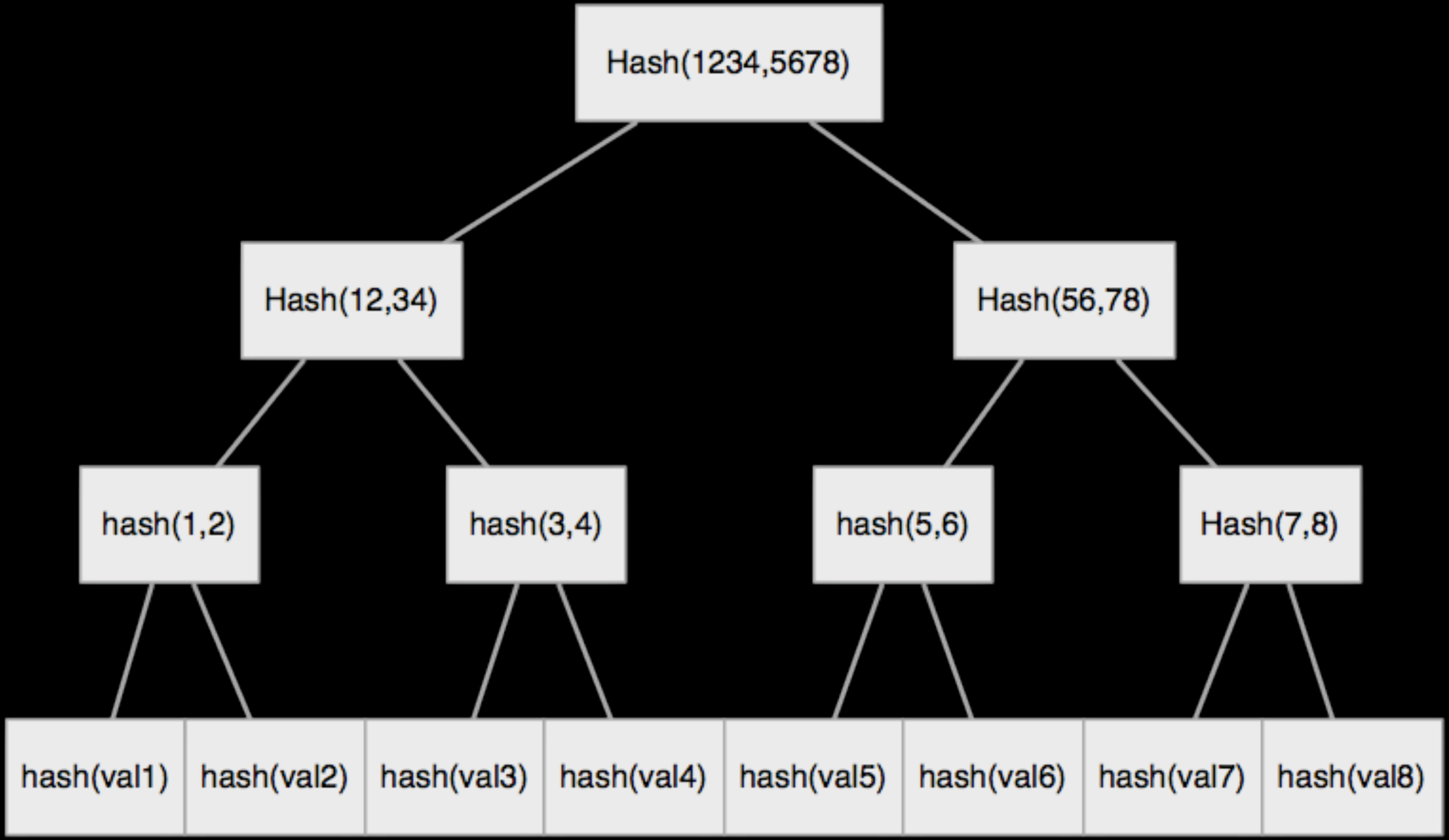
Gossip Girl

- Gossip servers randomly wake up
- Pick another membership server
- Merge membership tables
- Versioned by vector clocks

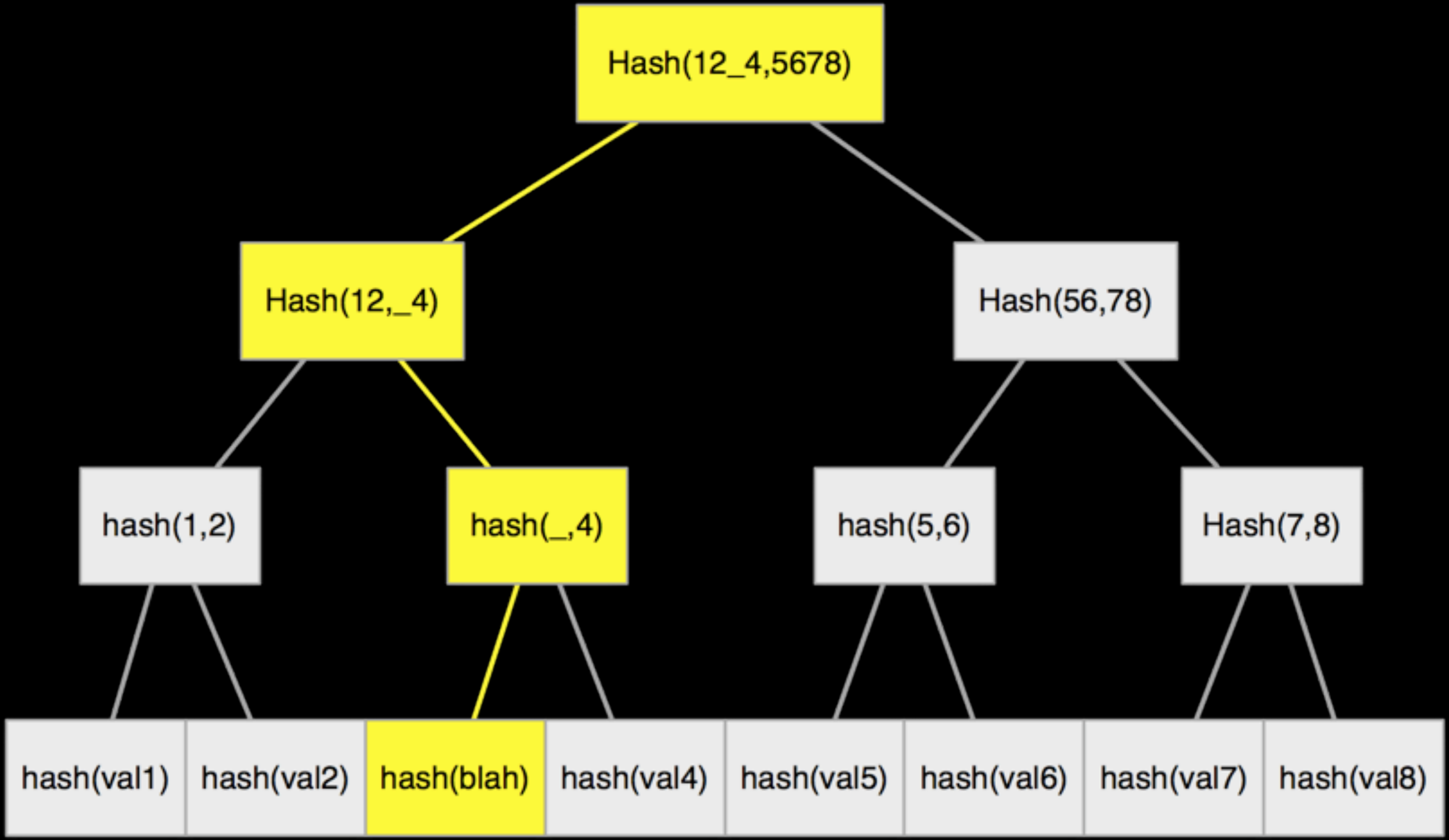
```
gossip_loop(Server) ->
  #membership{nodes=Nodes,node=Node} = gen_server:call(Server, state),
  case lists:delete(Node, Nodes) of
    [] -> ok; % no other nodes
    Nodes1 when is_list(Nodes1) ->
      fire_gossip(random_node(Nodes1))
  end,
  SleepTime = random:uniform(5000) + 5000,
  receive
    stop -> gossip_paused(Server);
    _Val -> ok
  after SleepTime ->
    ok
  end,
  gossip_loop(Server).
```

Storage Server


storage_server
Cache
storage engine
merkle tree



Merkle Trees



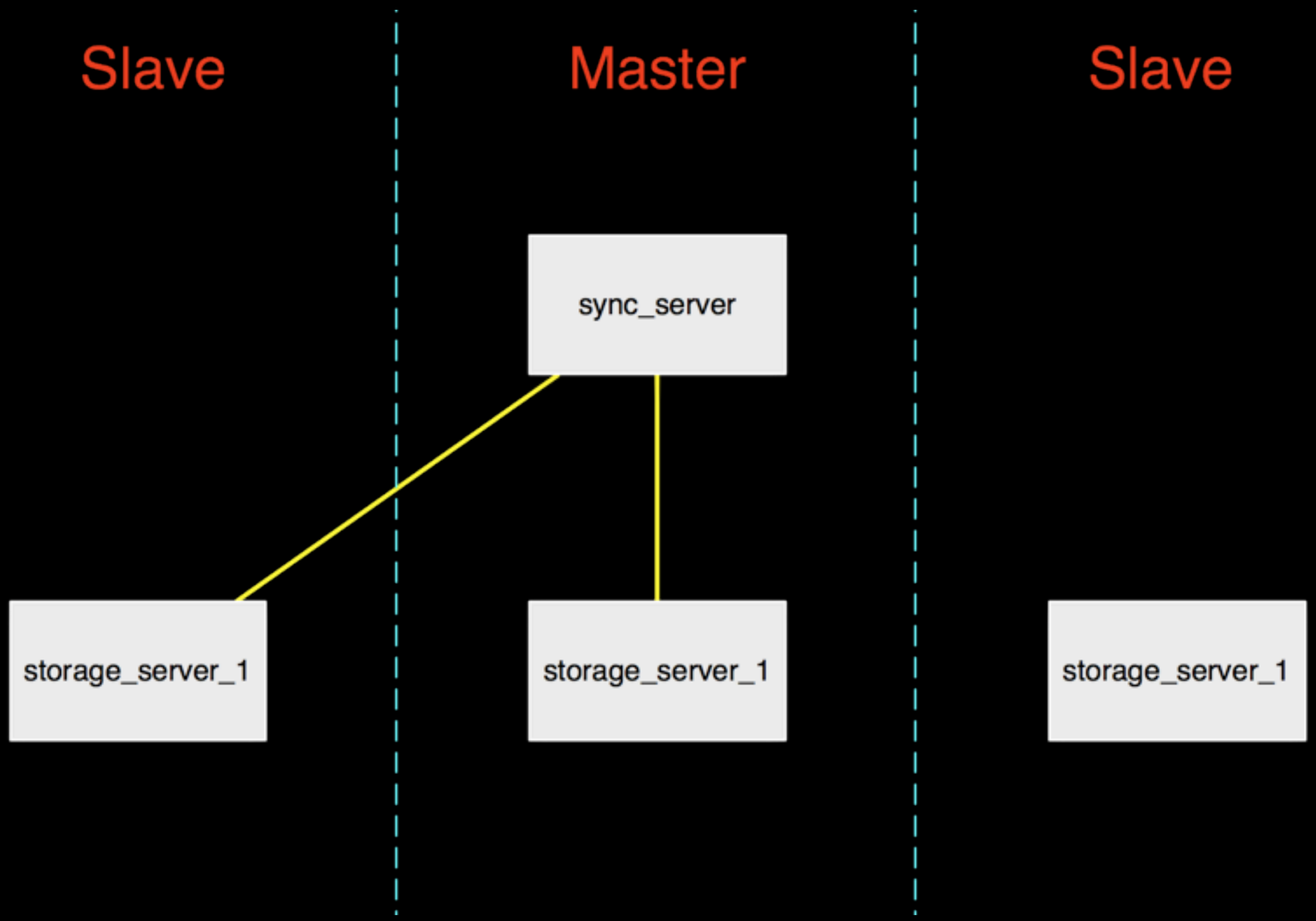
Merkle Trees

$O(\log n)$

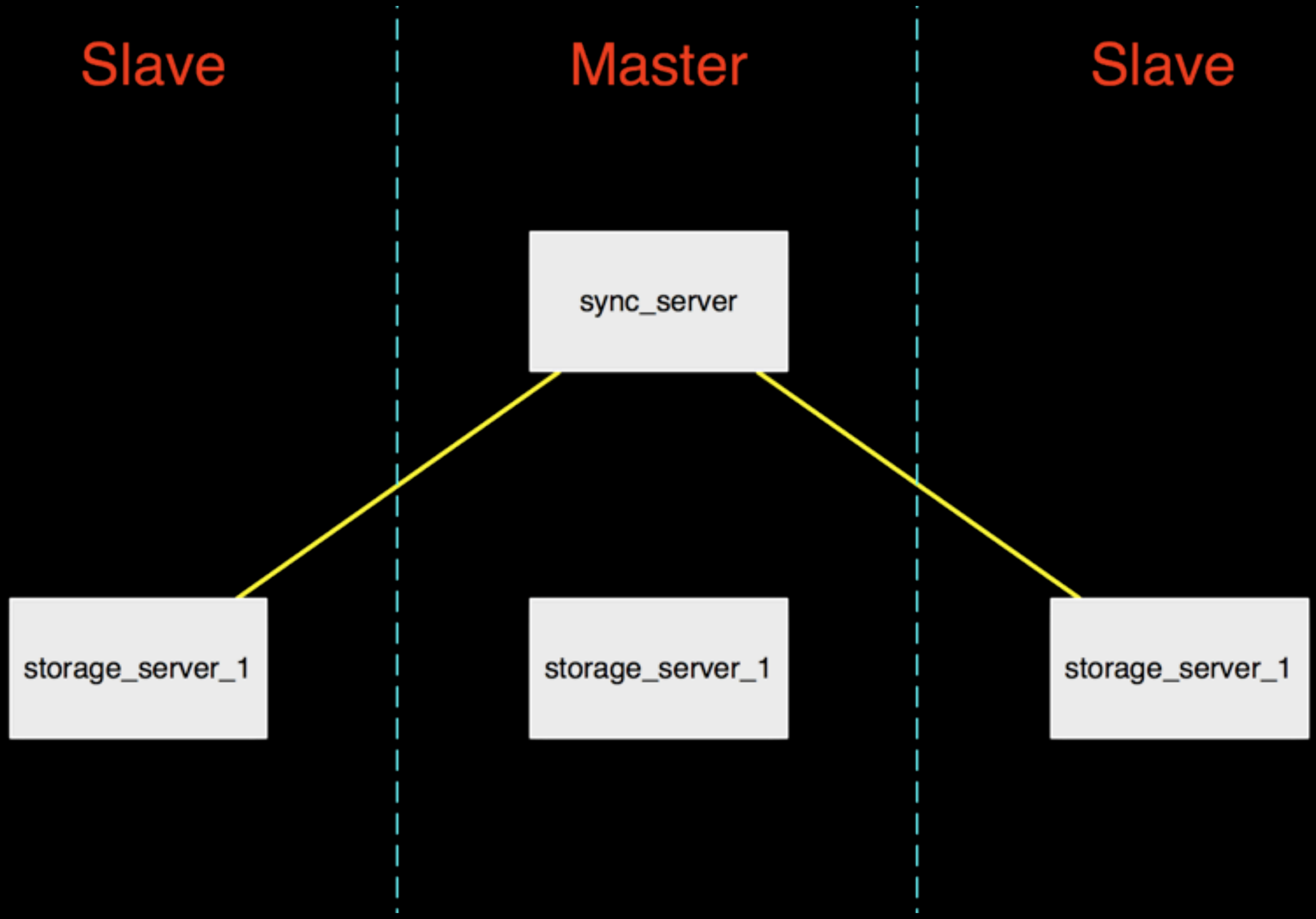
- Implemented on disk as a B-Tree
- Includes buddy-block allocator for key space
- Extremely gnarly code

Minimal Transfer

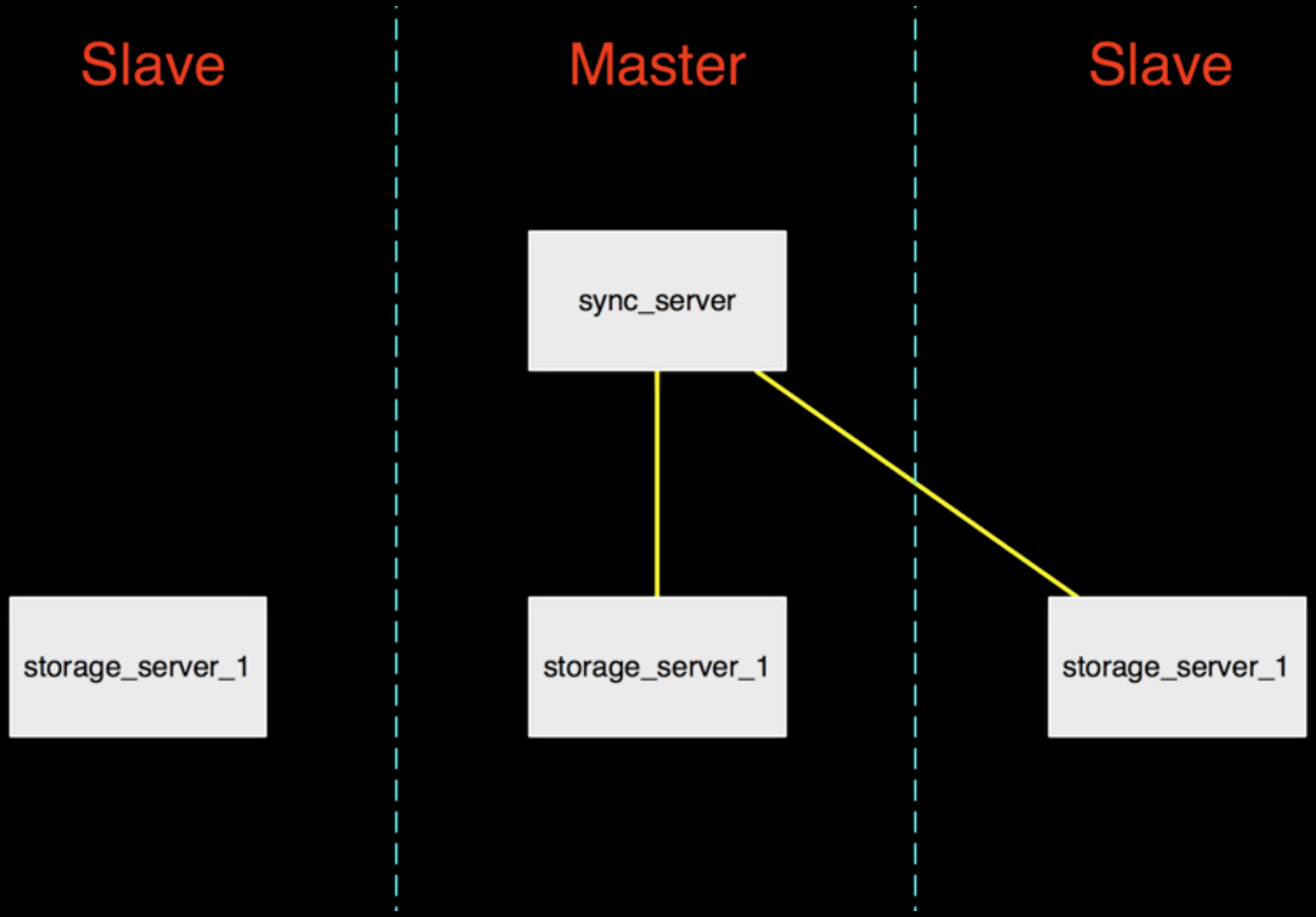
Sync Server



Randomly Wakes Up



Choses two replicas



Transfers Merkle Diff

```
[cliff@yourmom ~]$ dynomite console

(dynomite@yourmom)1> sync_manager:running().

[{3623878657,'dynomite@yourmom','dynomite@yourd
ad'},

{3892314113,'dynomite@yourmom','dynomite@cableg
uy'},
```

Problem Areas

Membership

Problems

- New partitions online before they are ready
- A priori knowledge of what is running
- Possible to dilute a cluster into uselessness
- Migrations are tremendously painful

Direction

- Storage servers rally with local membership server
- Use monitors on local storage
- Alerts for dangerous situations

Merkle

Erlang is good at many things

Low level storage ain't one

Direction

- Rewrite Dmerkle storage in C
- Use async driver pool
- Build in more crash recovery to the format

Web Console

Direction

- More detailed visualizations of cluster health
- Perform “safe” ops tasks

And other things I haven’t thought of

Demo!

Spare a patch, friend?

- http://github.com/cliffmoon/dynomite
- http://wiki.github.com/cliffmoon/dynomite

Q and or A