

Dynomite

Yet Another Distributed Key Value Store

@moonpolysoft -
questions

**#dynamite on
irc.freenode.net**

Dynomite!



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

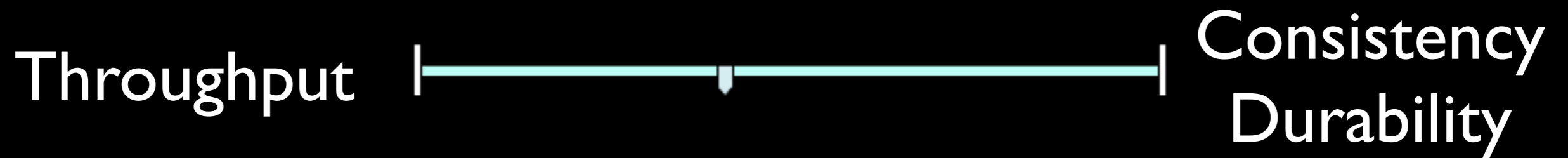
At Powerset

- 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

Latency

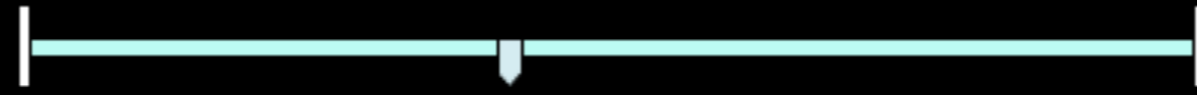


Consistency

R – Read Quorum

Minimum participation
for read

Latency



Durability

W – Write Quorum

Minimum participation
for write

Throughput



Scalability

Q – Partitioning

$$\text{Partitions} = 2^Q$$

QNRW – Defines your
cluster

At Powerset

- Batch writes
- Online reads
- Clients will read many images concurrently

Q - 6
N - 3
R - 1
W - 2

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

```
{ok, 1}
```

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

```
{ok, 1}
```

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

```
{ok, 1}
```

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

```
{ok, 1}
```

Value is **always** Binary

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

```
{ok, 1}
```

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

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

```
(dynamite@galva)2> Terms = binary_to_term(Bin).
```

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

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

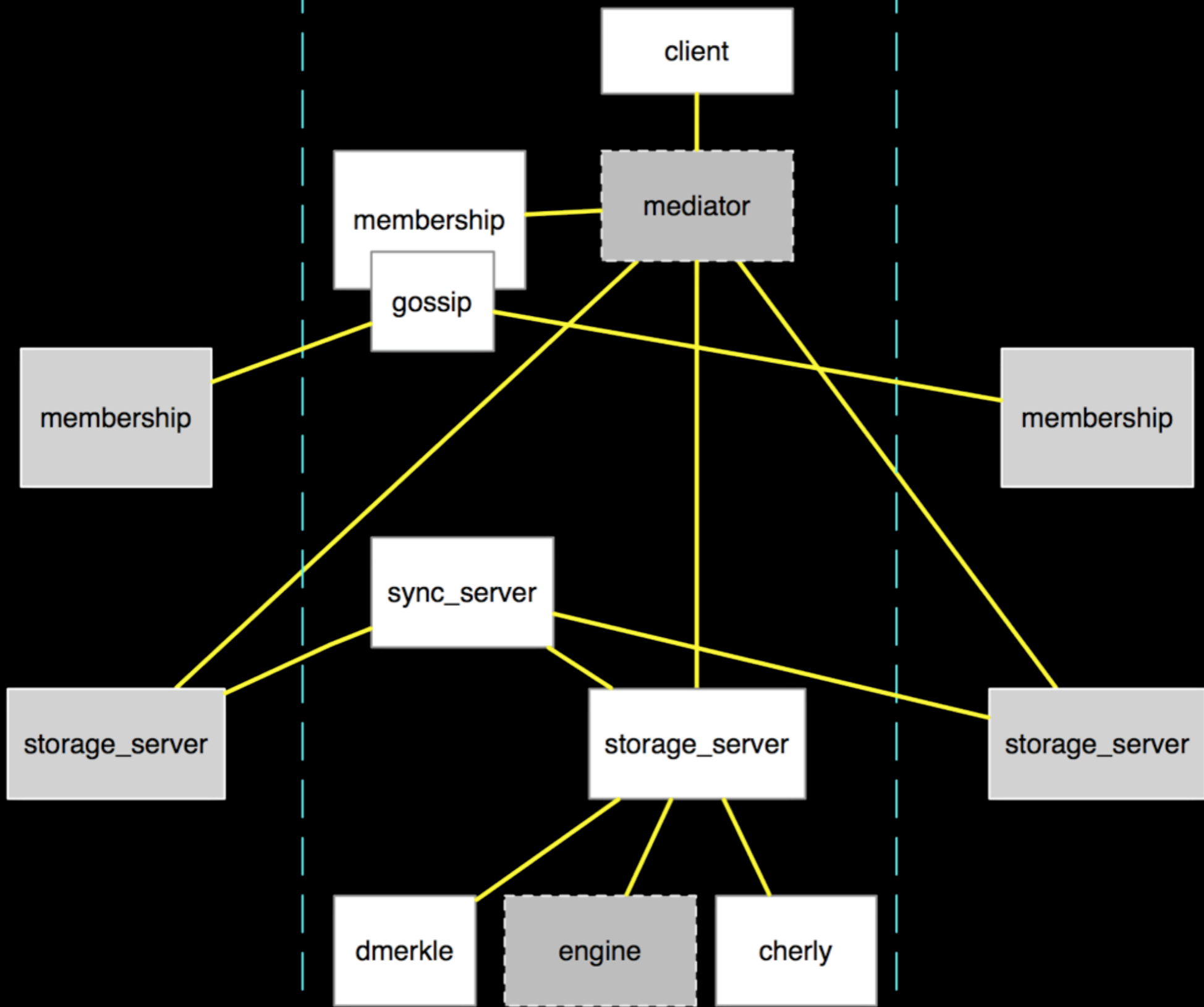
```
{ok, {[{"<0.630.0>", 1240543271.698089}],  
      [ [<<131,108,0,0,0,2,...>> ] ]}}
```

```
(dynamite@galva)2> Terms = binary_to_term(Bin).
```

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

mediator:delete/1
mediator:has_key/1

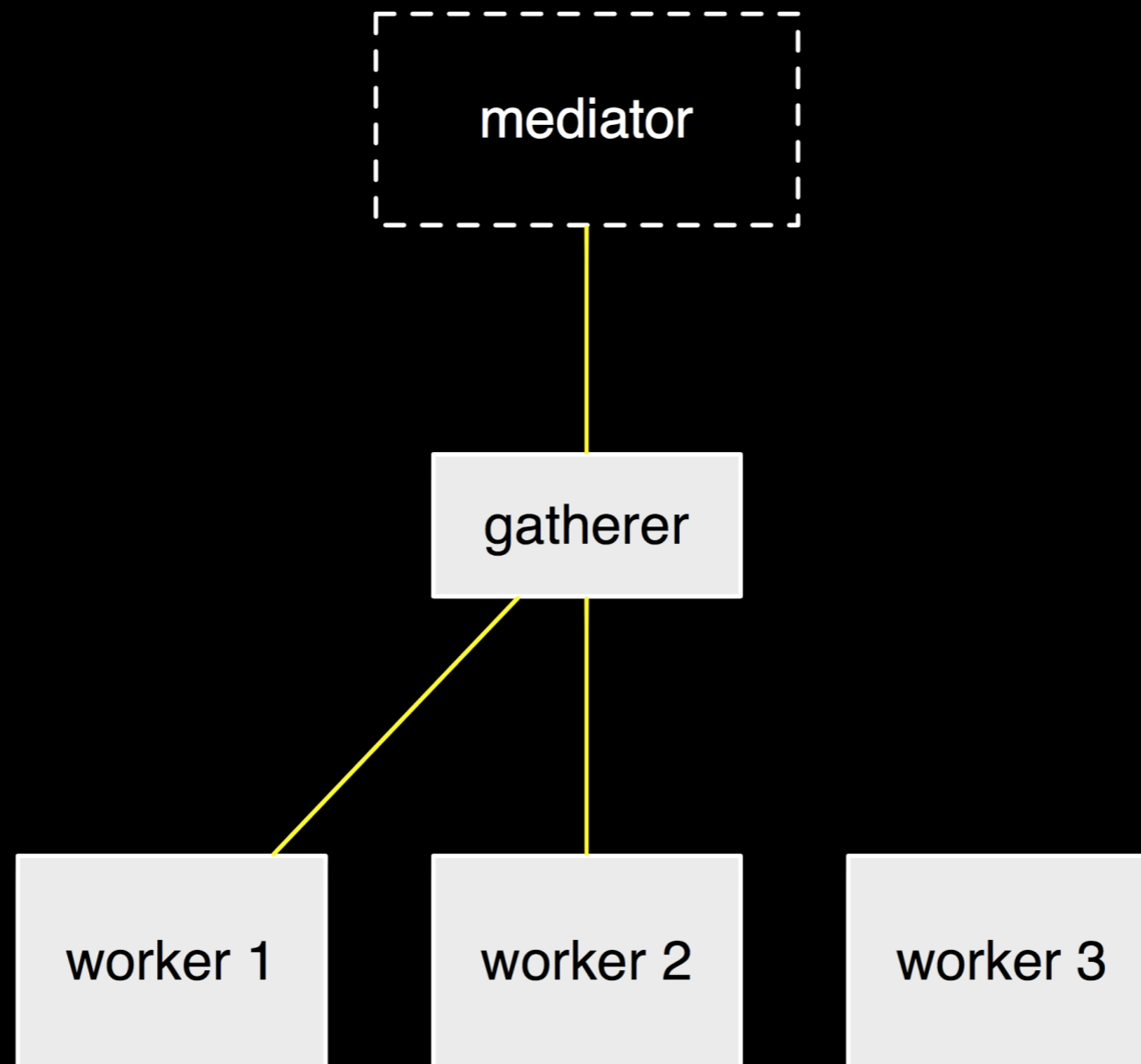
Up In Dem Gutz



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),
  Ret.

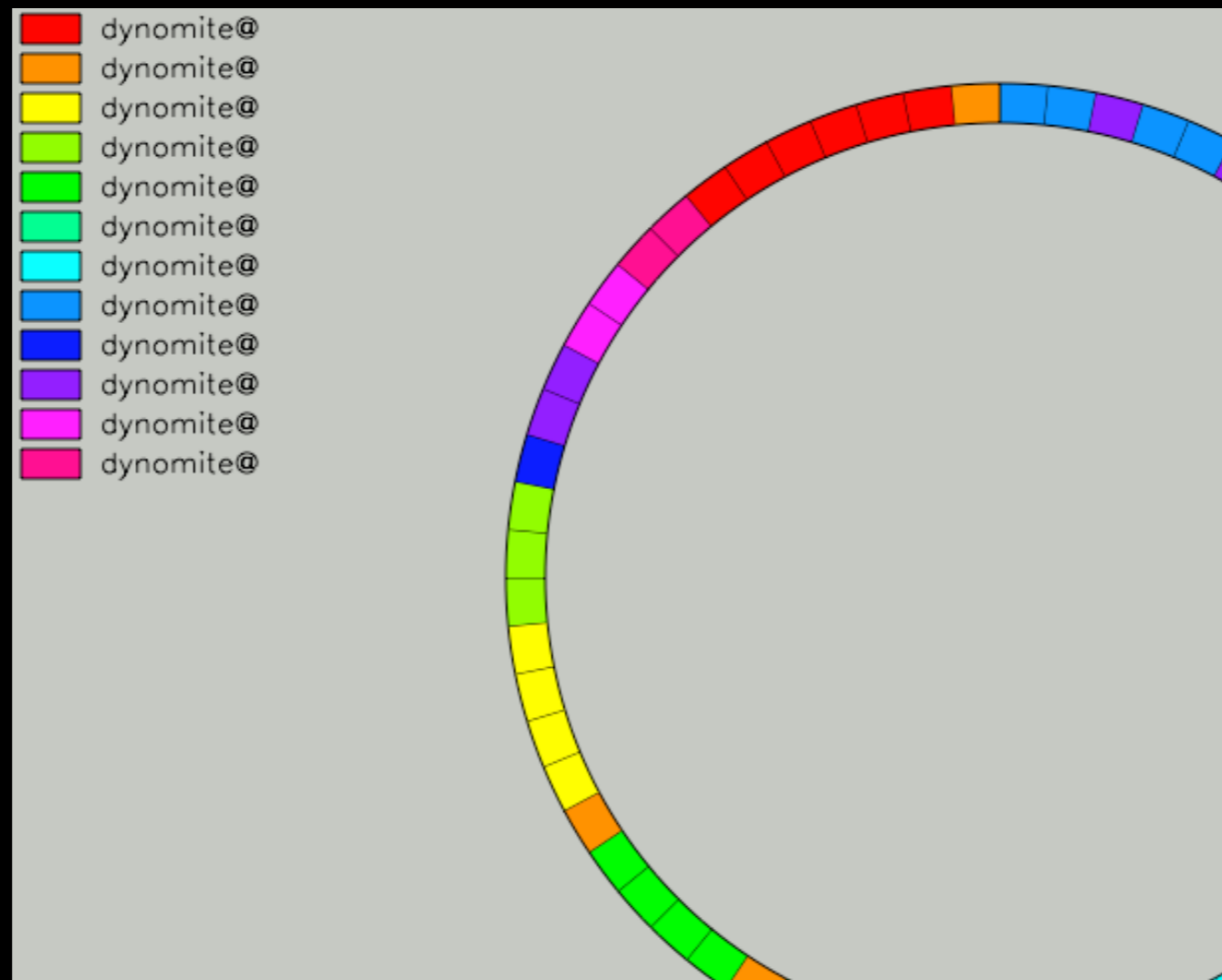
```

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



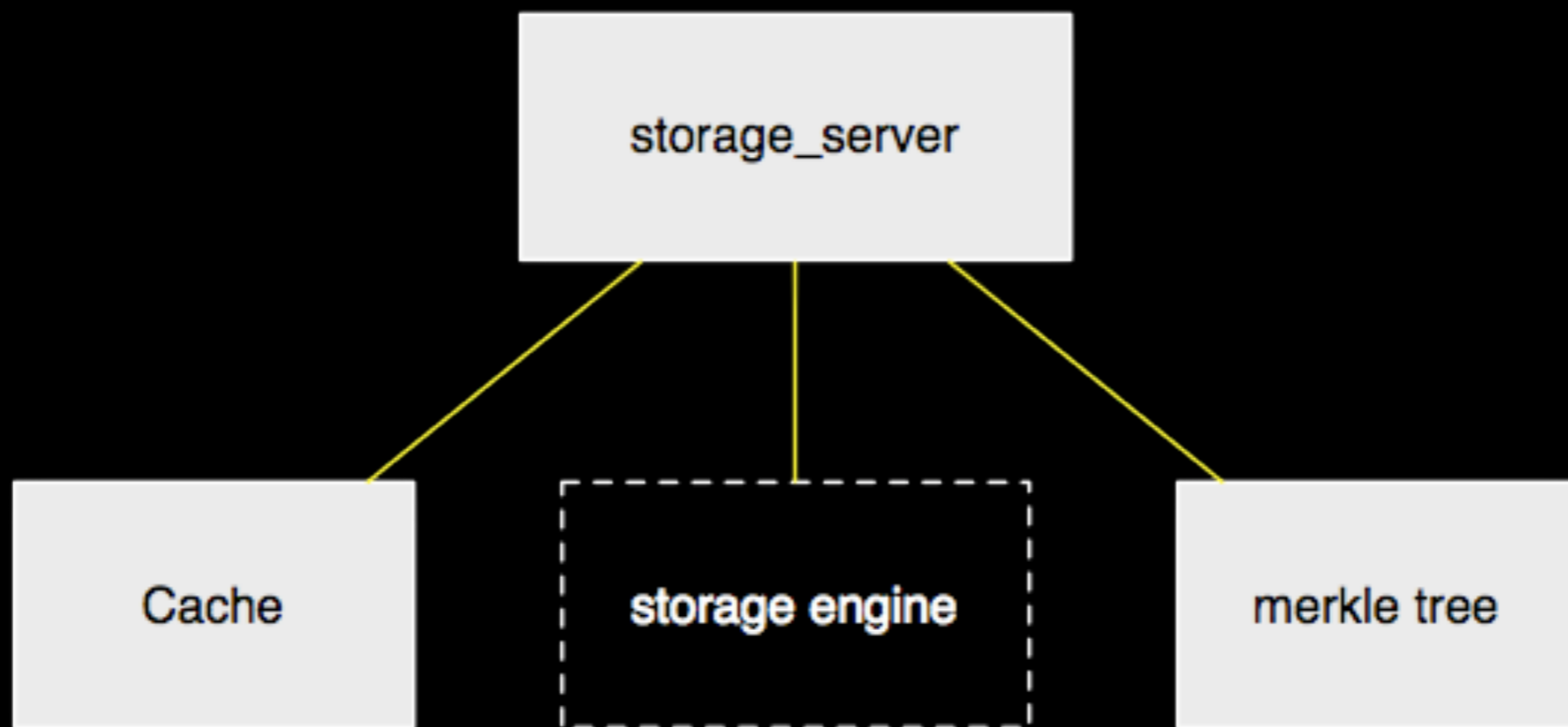
Maps Partitions To Nodes

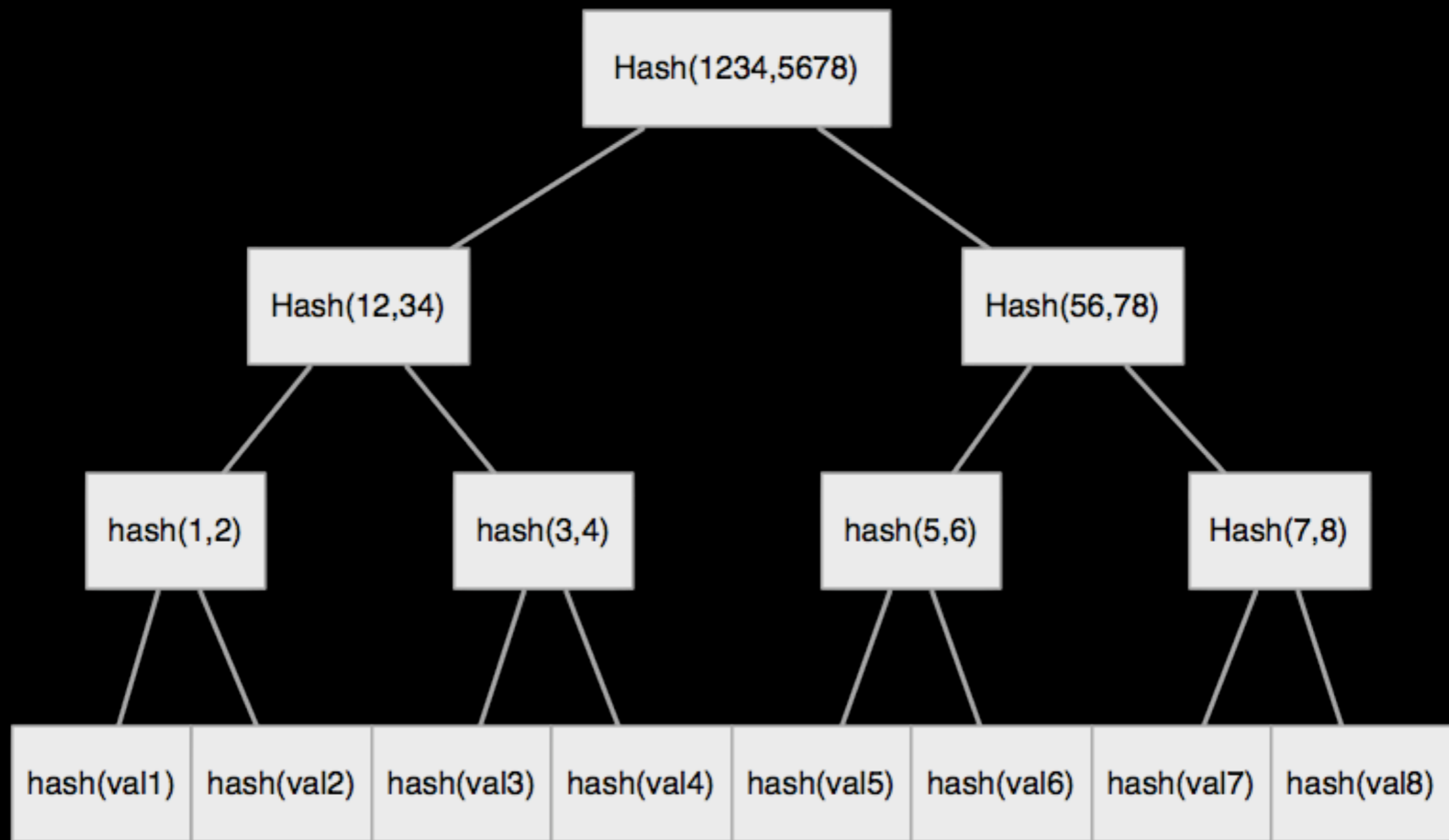
Gossip Girl

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

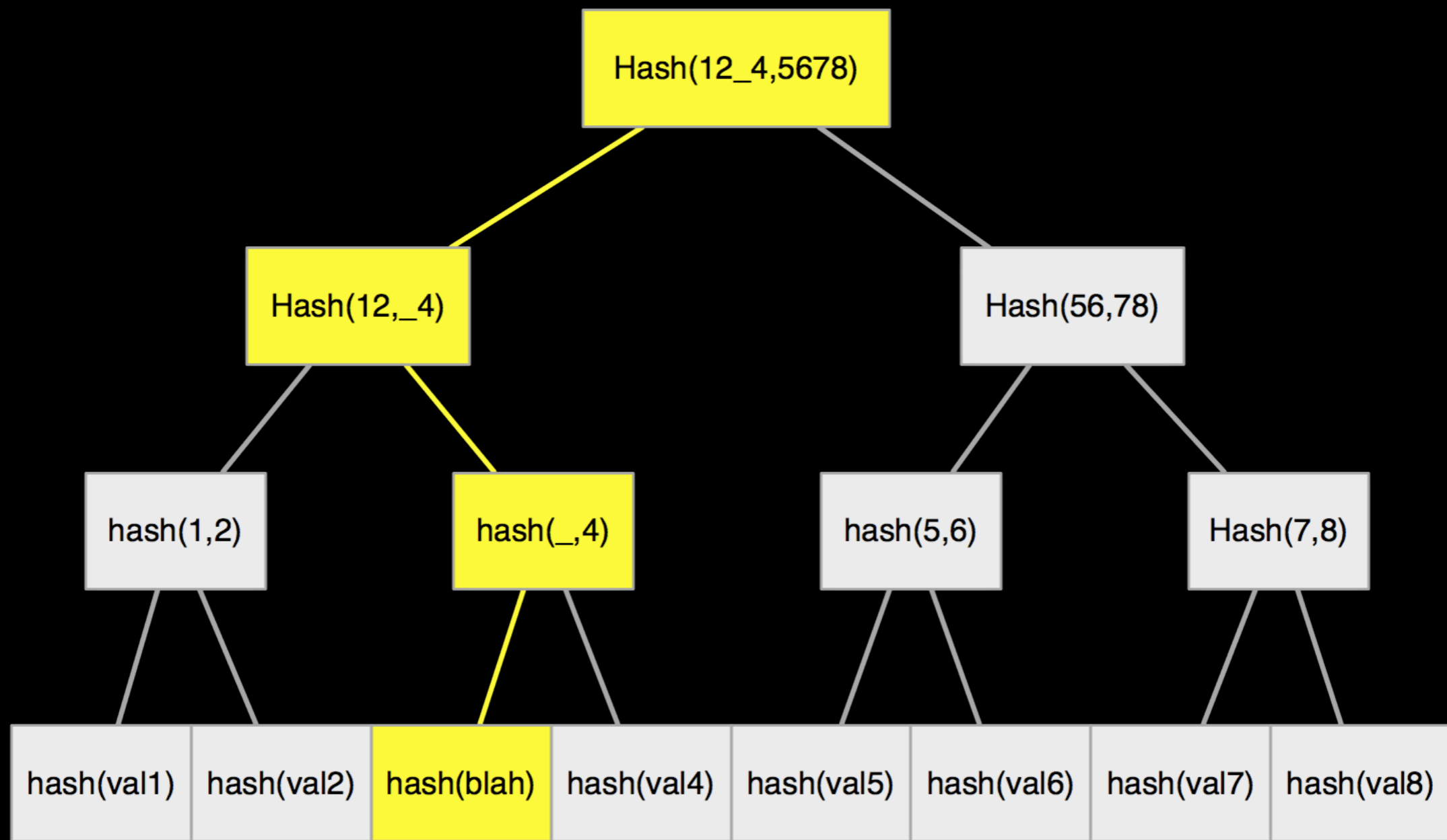
```
gossip_loop(Server) ->
  #membership{nodes=Nodes, node=Node} =
  gen_server:call(Server, state),
  case lists:delete(Node, Nodes) of
    [] -> ok; % no other nodes
    Nodes1 when is_list(Nodes1) ->
      fire_gossip(random_node(Nodes1))
  end,
  SleepTime = random:uniform(5000) + 5000,
  receive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

Storage Server





Merkle Trees



Merkle Trees

$O(\log n)$

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

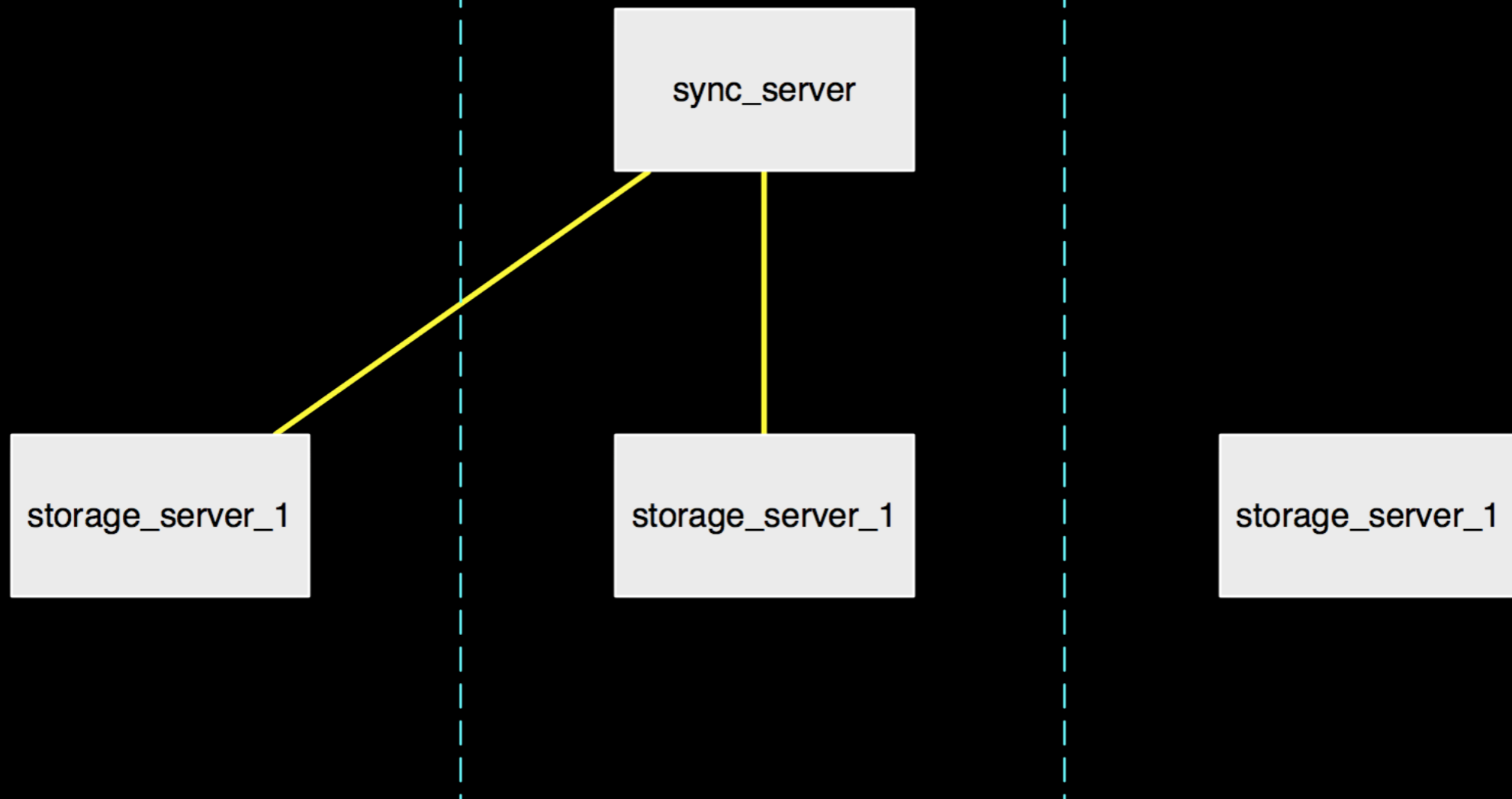
Minimal Transfer

Sync Server

Slave

Master

Slave

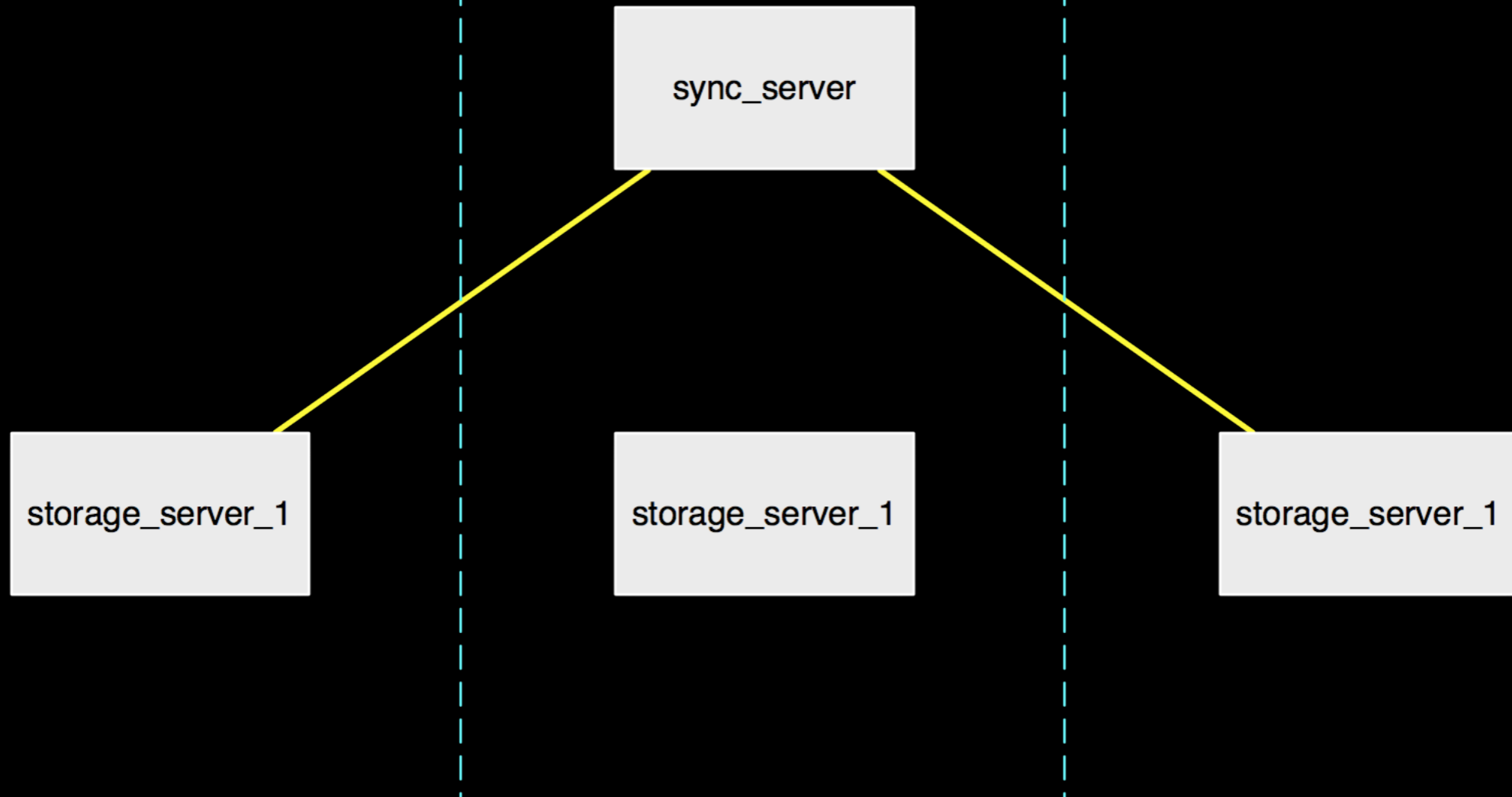


Randomly Wakes Up

Slave

Master

Slave

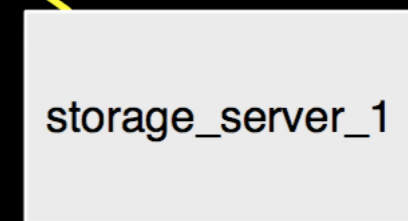
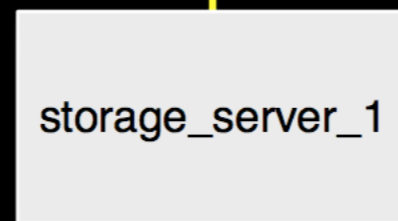
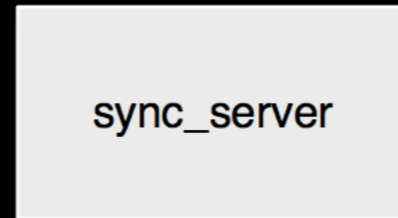
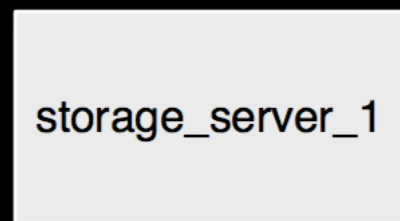


Choses two replicas

Slave

Master

Slave



Transfers Merkle Diff

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

```
(dynamite@yourmom)1> sync_manager:running().
```

```
[{3623878657, 'dynamite@yourmom', 'dynamite@yourdad'},  
 {3892314113, 'dynamite@yourmom', 'dynamite@cableguy'},  
 {4026531841, 'dynamite@yourmom', 'dynamite@mailman'}]
```

The Roadmap

Why Just Key/Value?

To Focus on
Distribution

How to maintain
focus...

And stay relevant?

Imagine a Generic
Distribution Mechanism

Not Tied to Key Value
Lookup

A Dynamo Framework

For Distributed Databases

Distribute any Data

Model

within reason

But How?

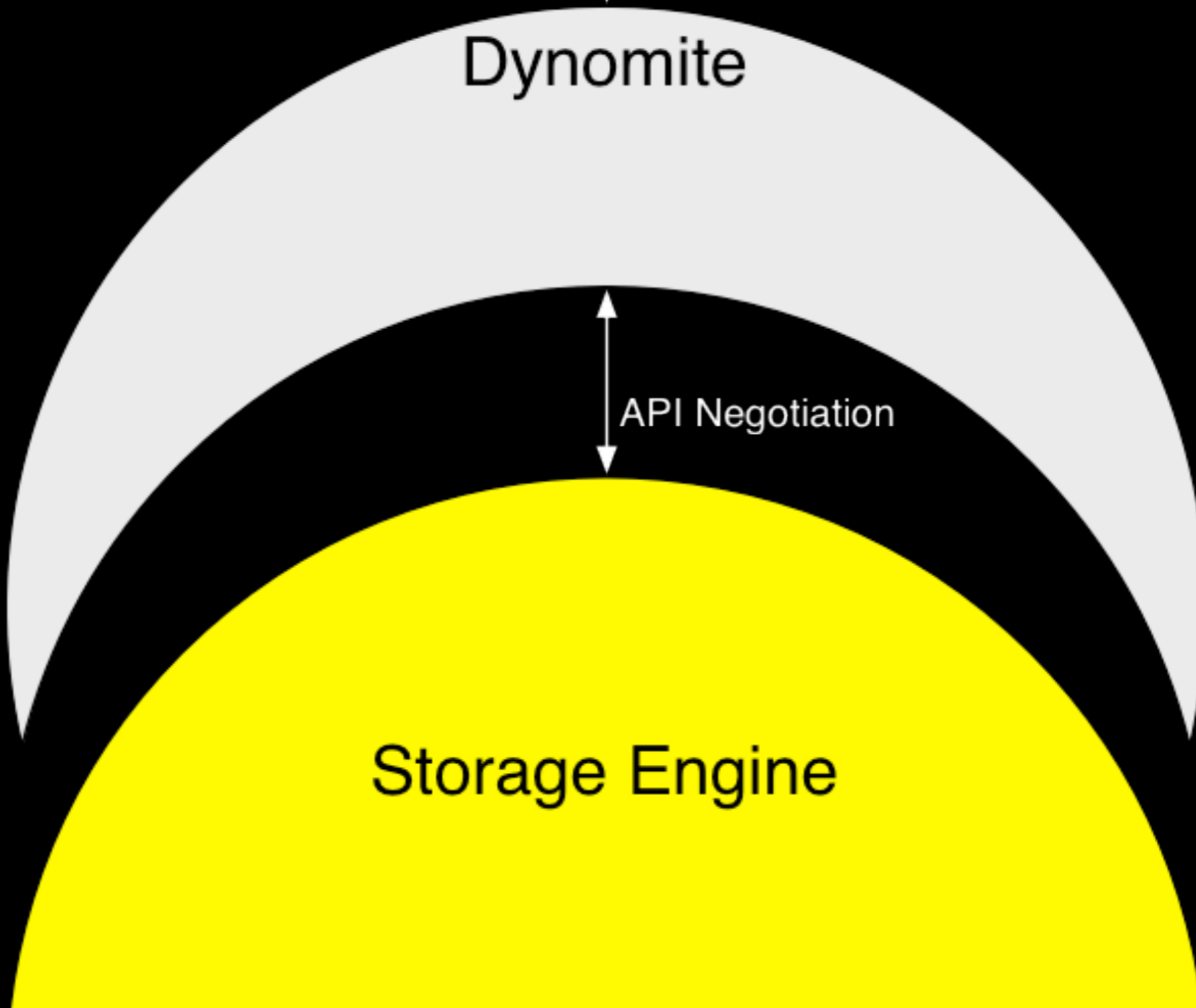
Client Application

Dynomite Client API

Dynomite

API Negotiation

Storage Engine



Storage Engine

- Persists data according to its data model
- Runs queries locally
- Provides a means to stitch data back together during read repair
- Can provide hashing
- Publishes its own API

Dynomite

- Distributes queries and updates to nodes
- Invokes the correct read repair phase
- Handles node membership transparently
- Provides network interface for clients
- Versions data

API Negotiation

- Storage Engine Publishes an API
- API Methods declare
 - Data Types
 - Conflict resolution
 - Result Collation
 - Partition Strategy

Client Protocol

- Dynamite is a front end to storage
- Clients can query API capabilities
- Mediator follows distribution semantics
 - Single partition
 - Range of partitions
 - Quorum requirements

```
api_info() ->
  {api_info,
   HashFun      = fun(Keys),
   MethodSpecs = [api_spec()]
  }

api_spec() = {
  ReadWrite      = read | write | cas,
  Plurality      = single | many,
  Name           = atom(),
  Arguments      = [argument_type_spec()],
  PartitionStrategy = single | many,
  CollationFun   = fun(ResultsA, ResultsB) | undefined,
  ResolutionFun  = fun(ContextA, ValueA, ContextB, ValueB) | undefined
}

argument_type_spec() = {
  Key           = true | false,
  Name          = atom(),
  DataType      = abstract_data_type()
}
```

Plz to Contribute

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

Q and or A