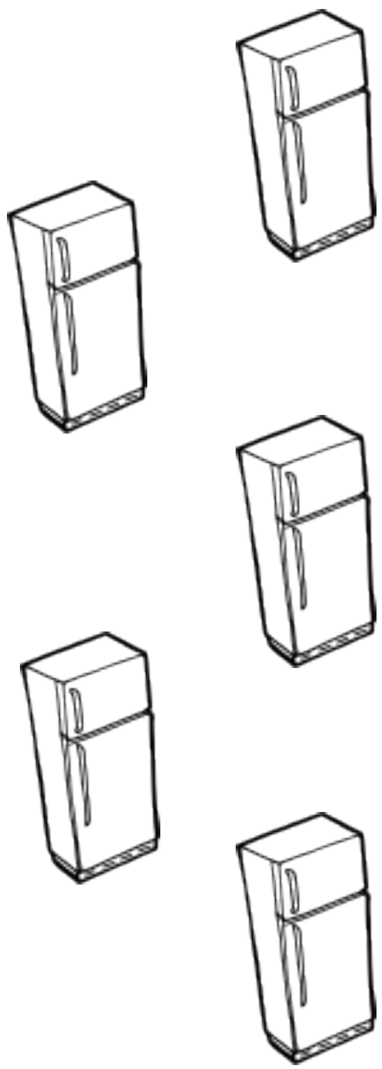
Declarative,
Secure,
Convergent
Edge Computation

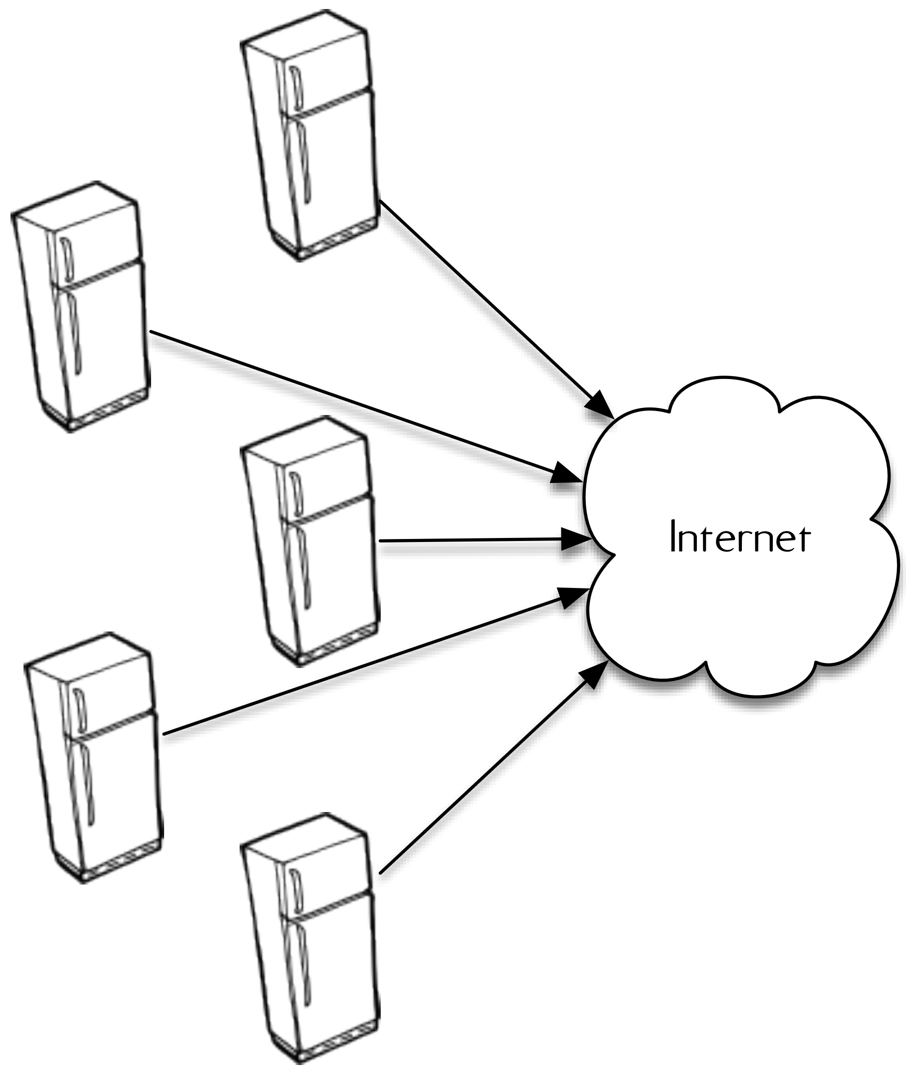
Christopher Meiklejohn

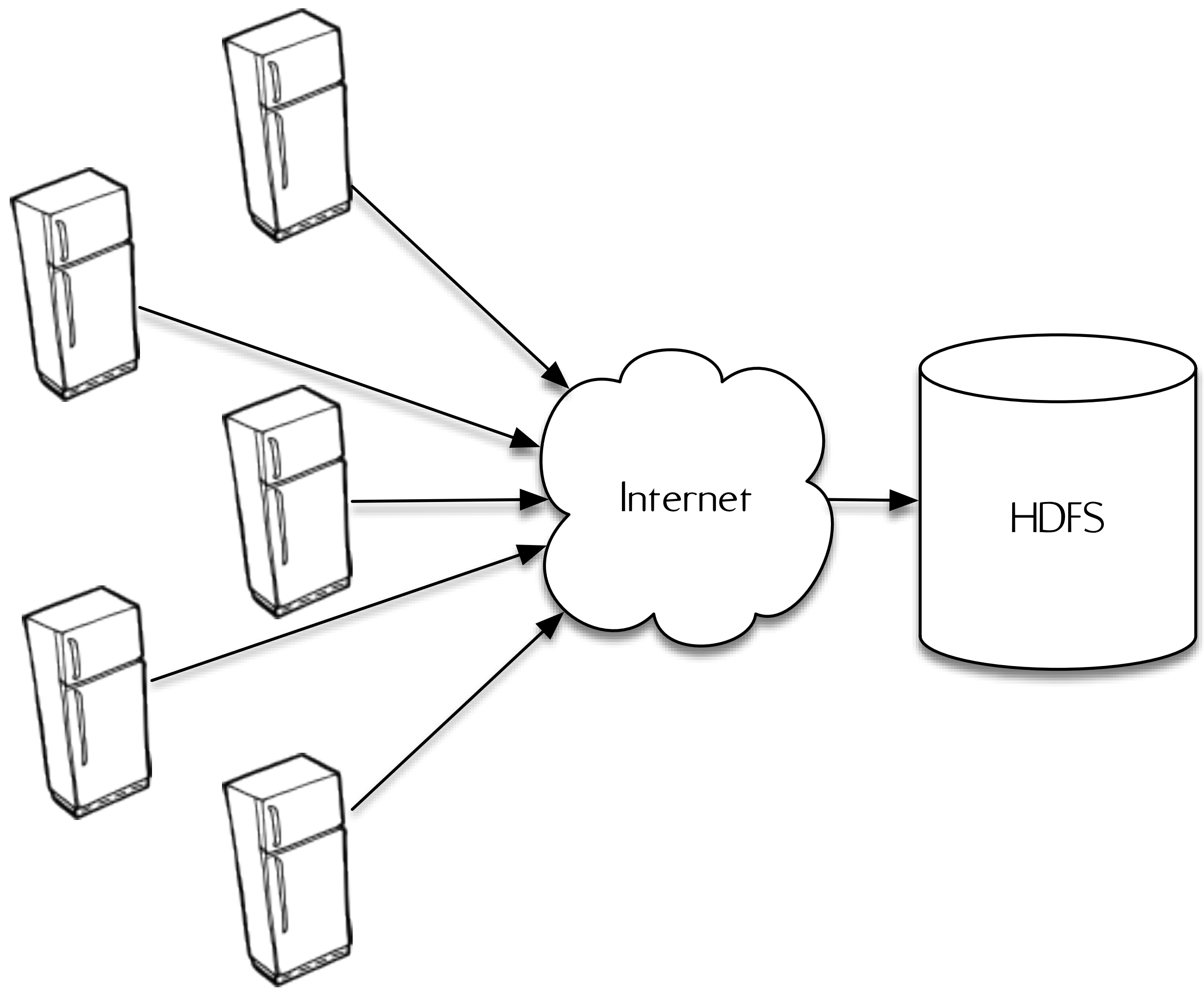
Erlang Factory 2016, March 10th, 2016

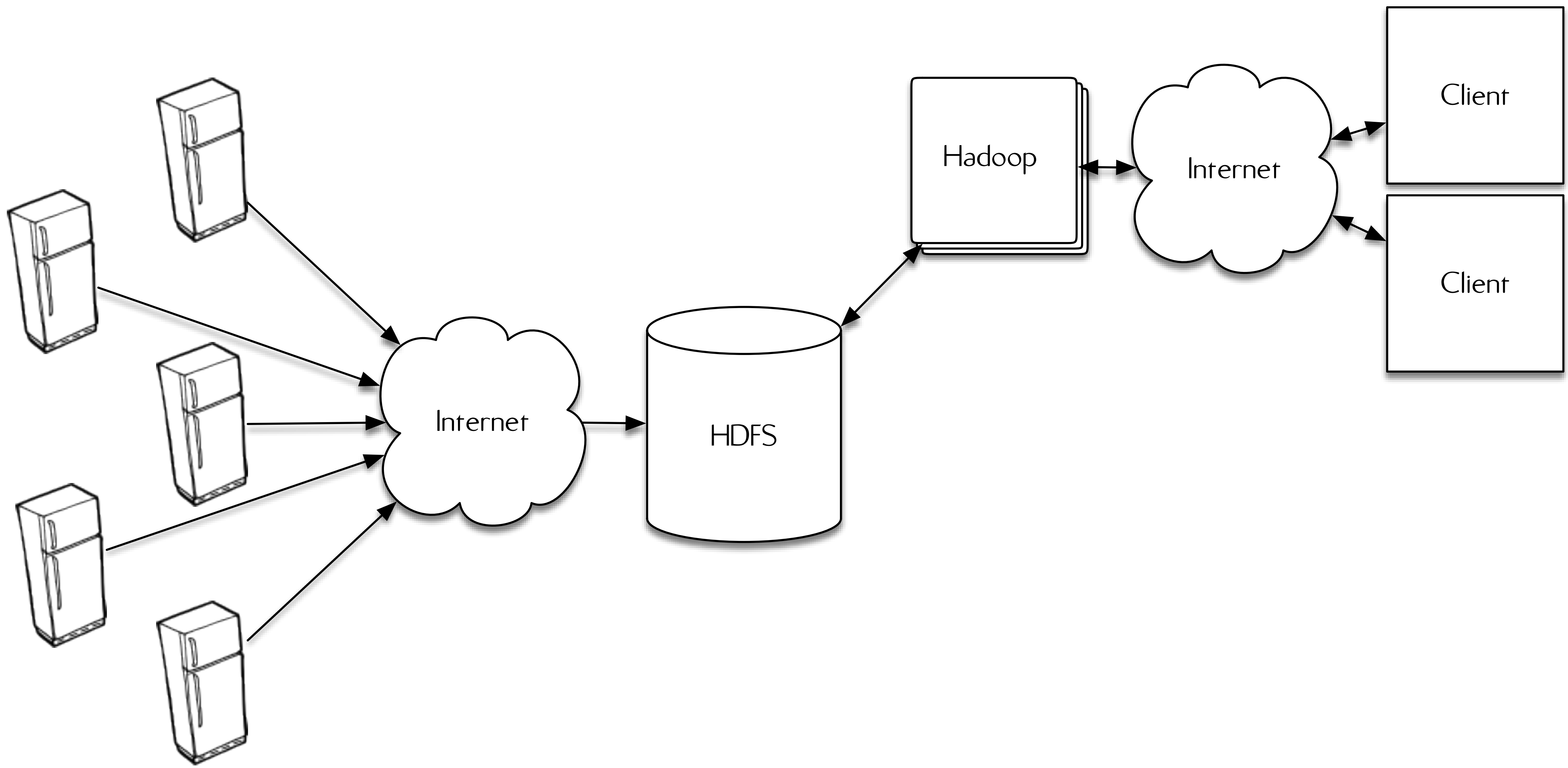
Example Application
Hospital Refrigerators

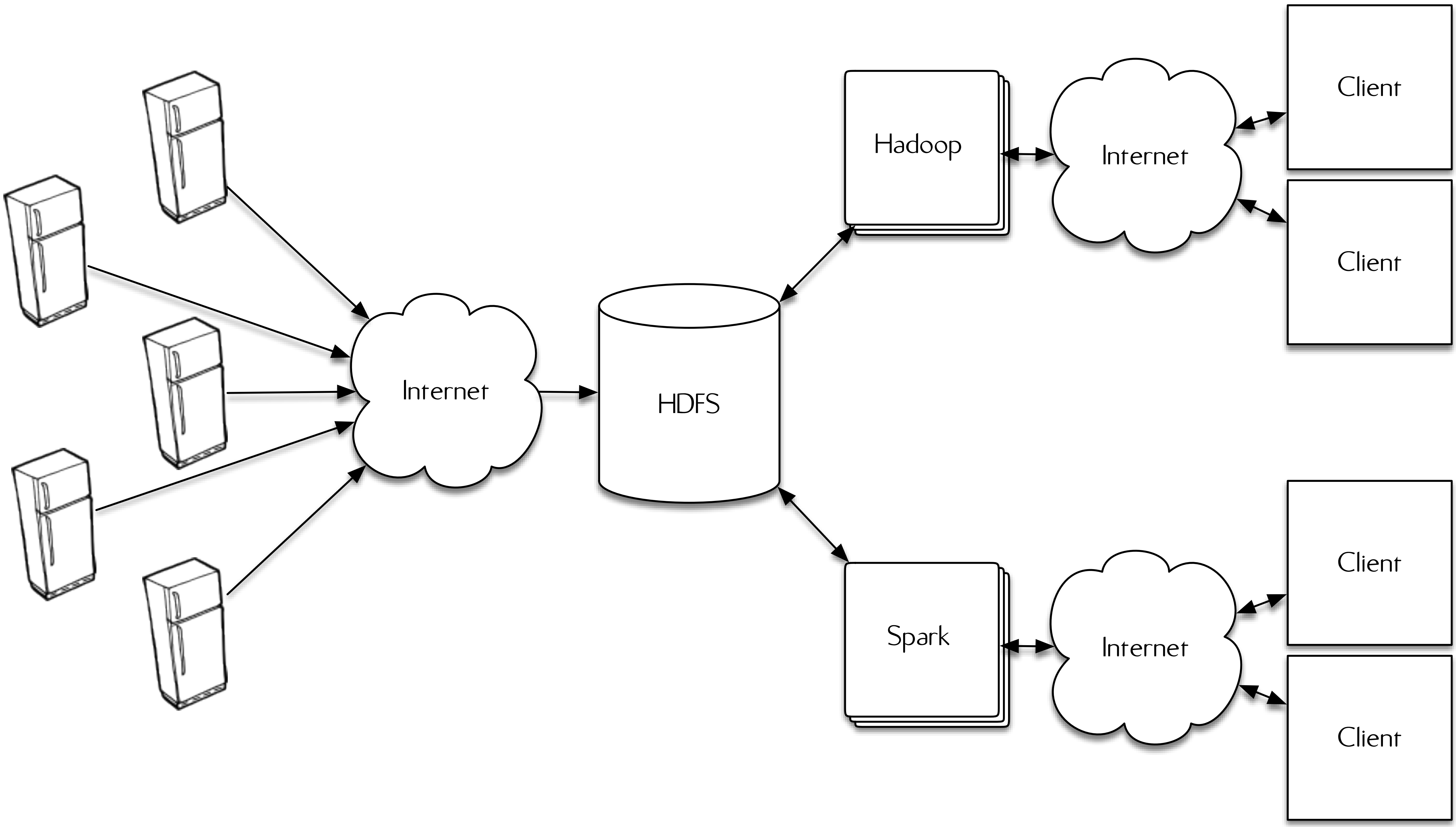
Hospital Refrigerators
Typical Topology



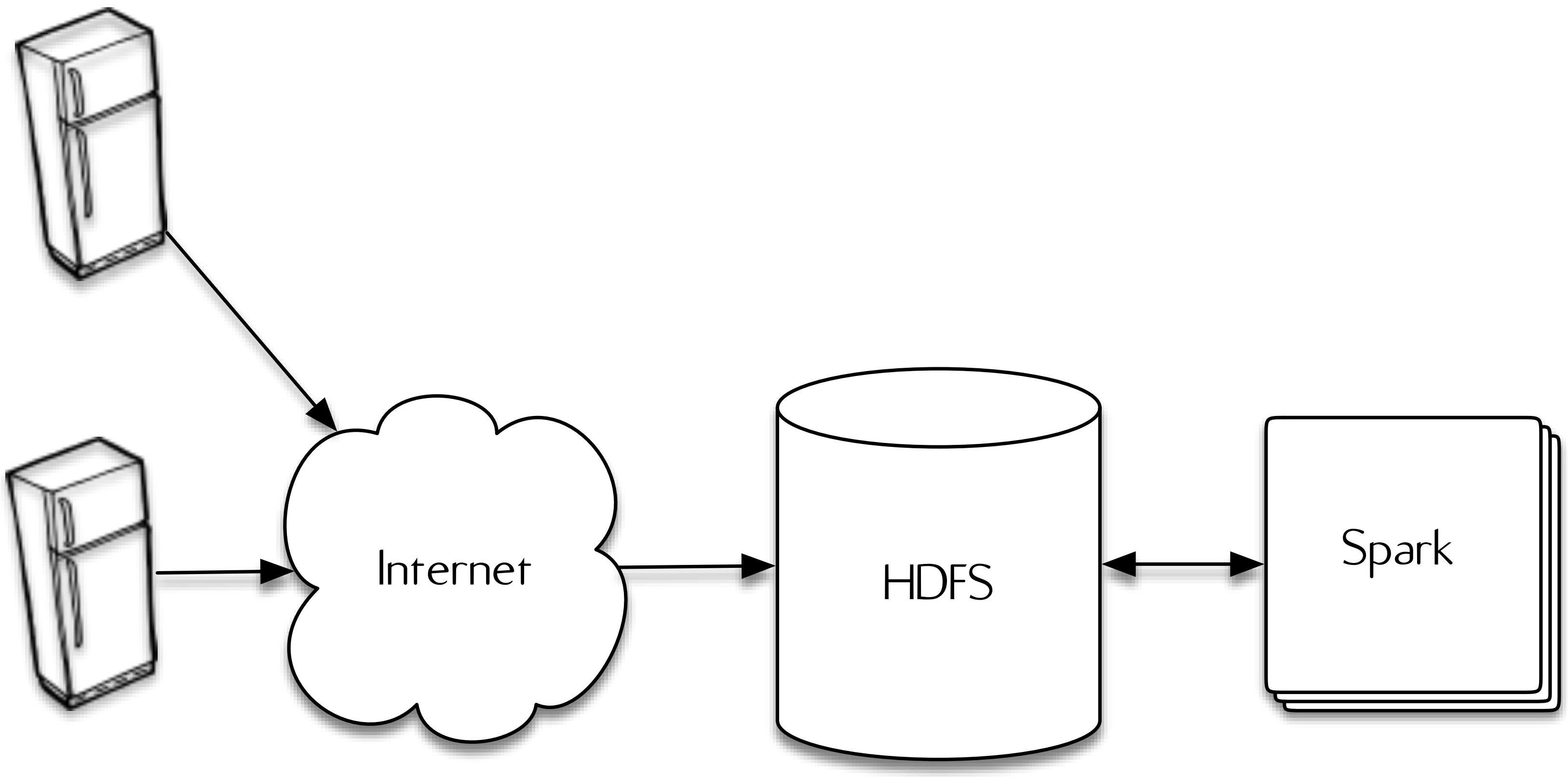

Internet

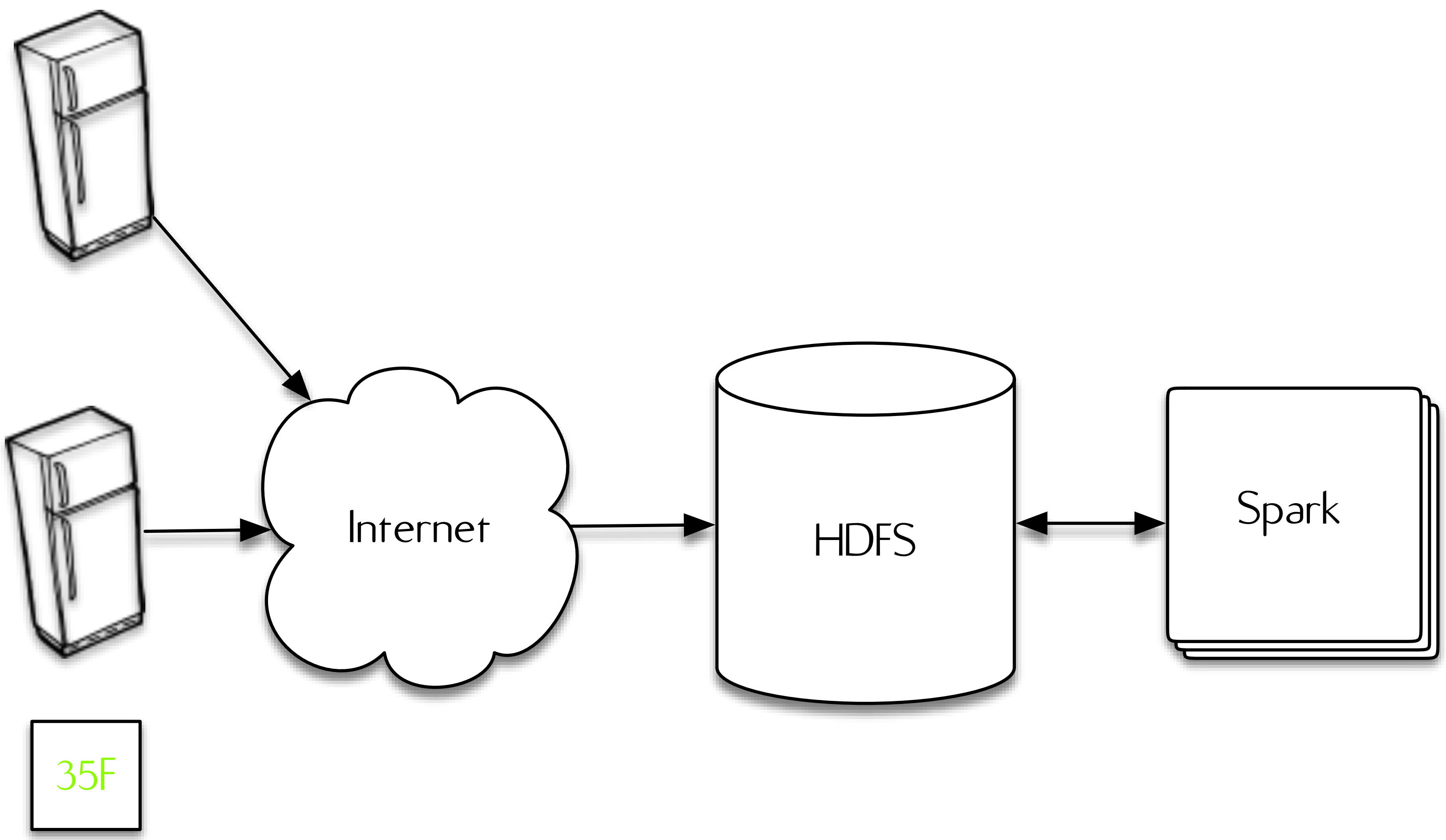

Internet
HDFS

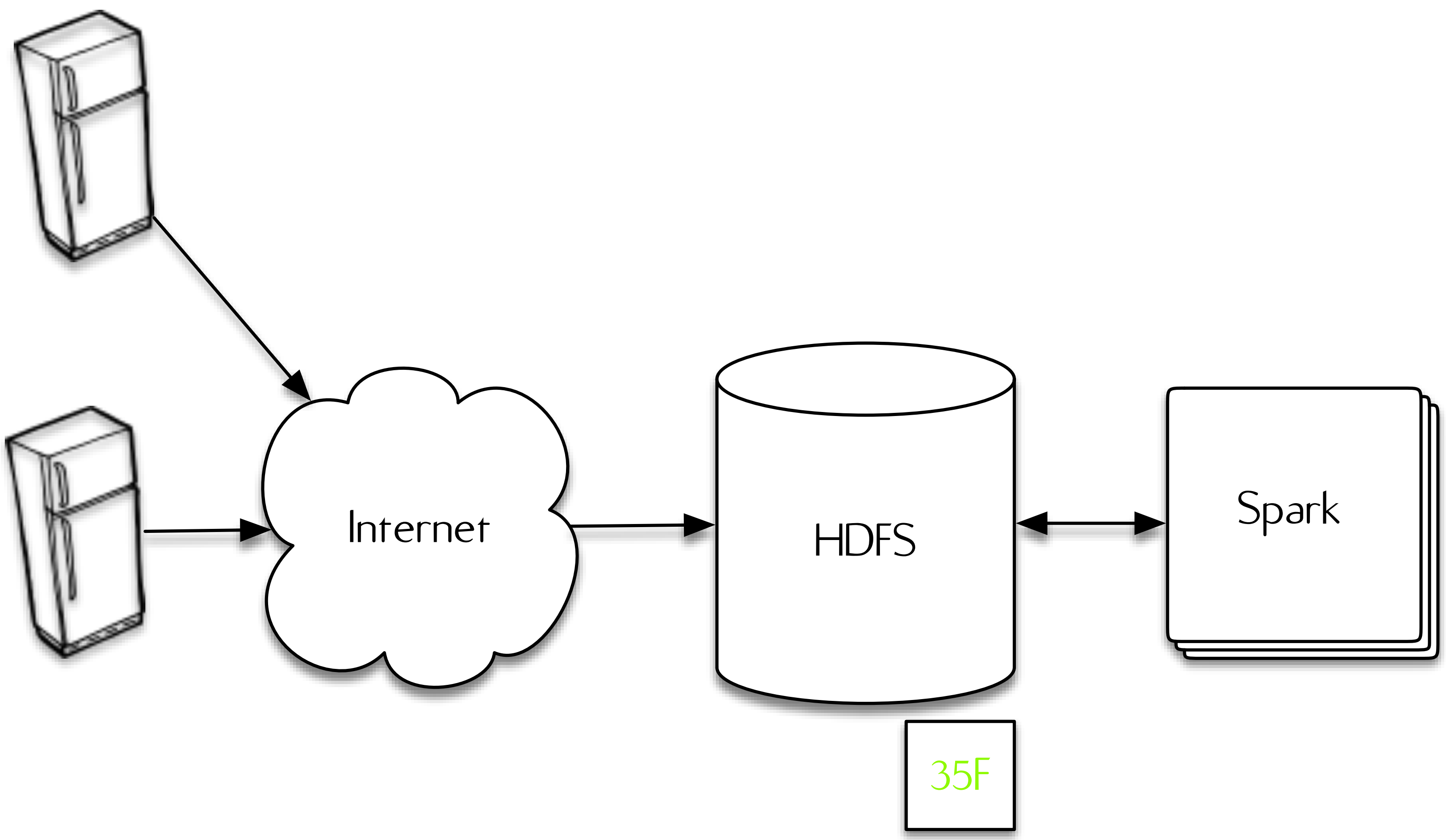

Hadoop
Internet
Client
Client
Internet
HDFS

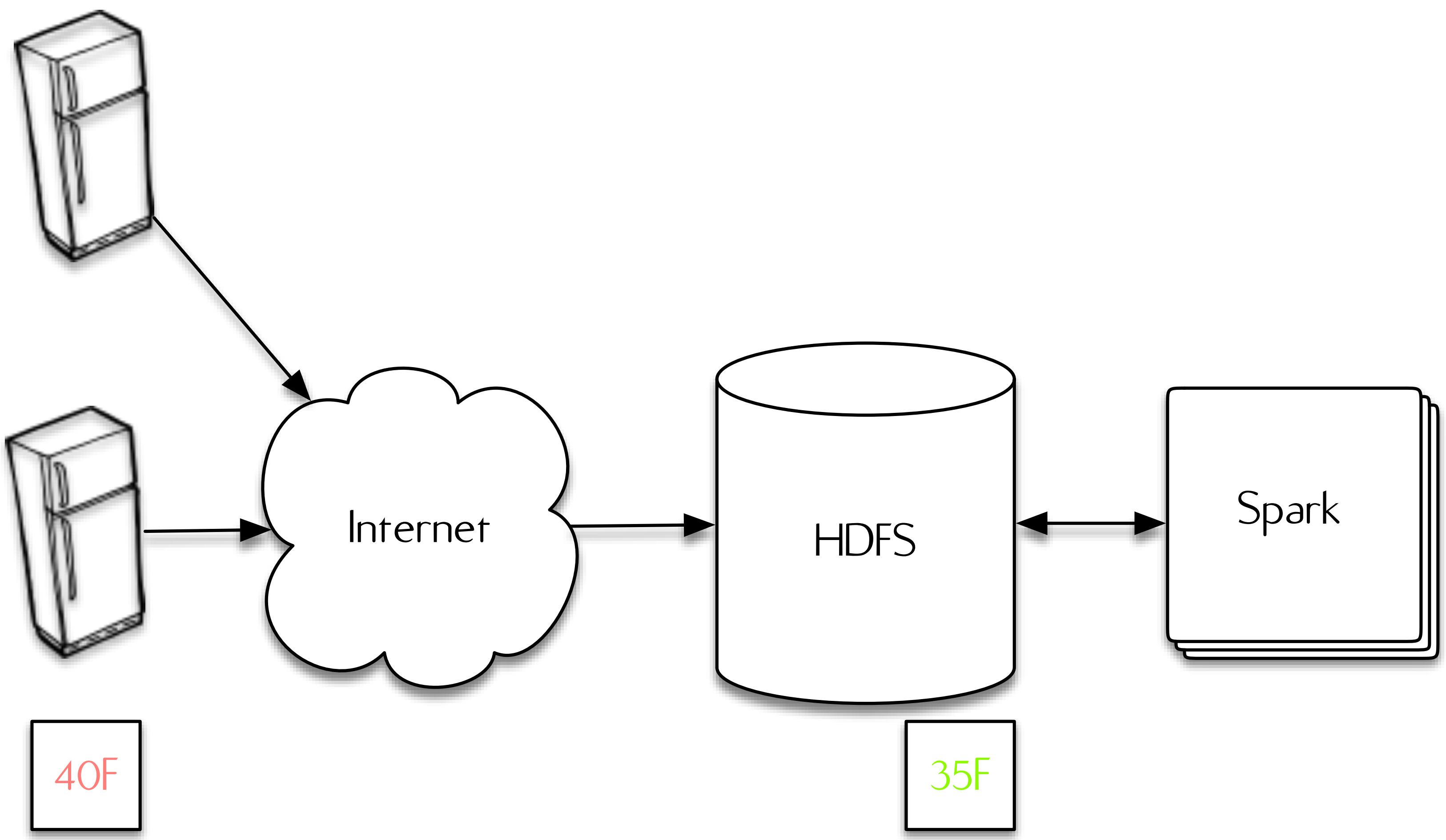

Internet
HDFS
Hadoop
Internet
Client
Client
Spark
Internet
Client
Client

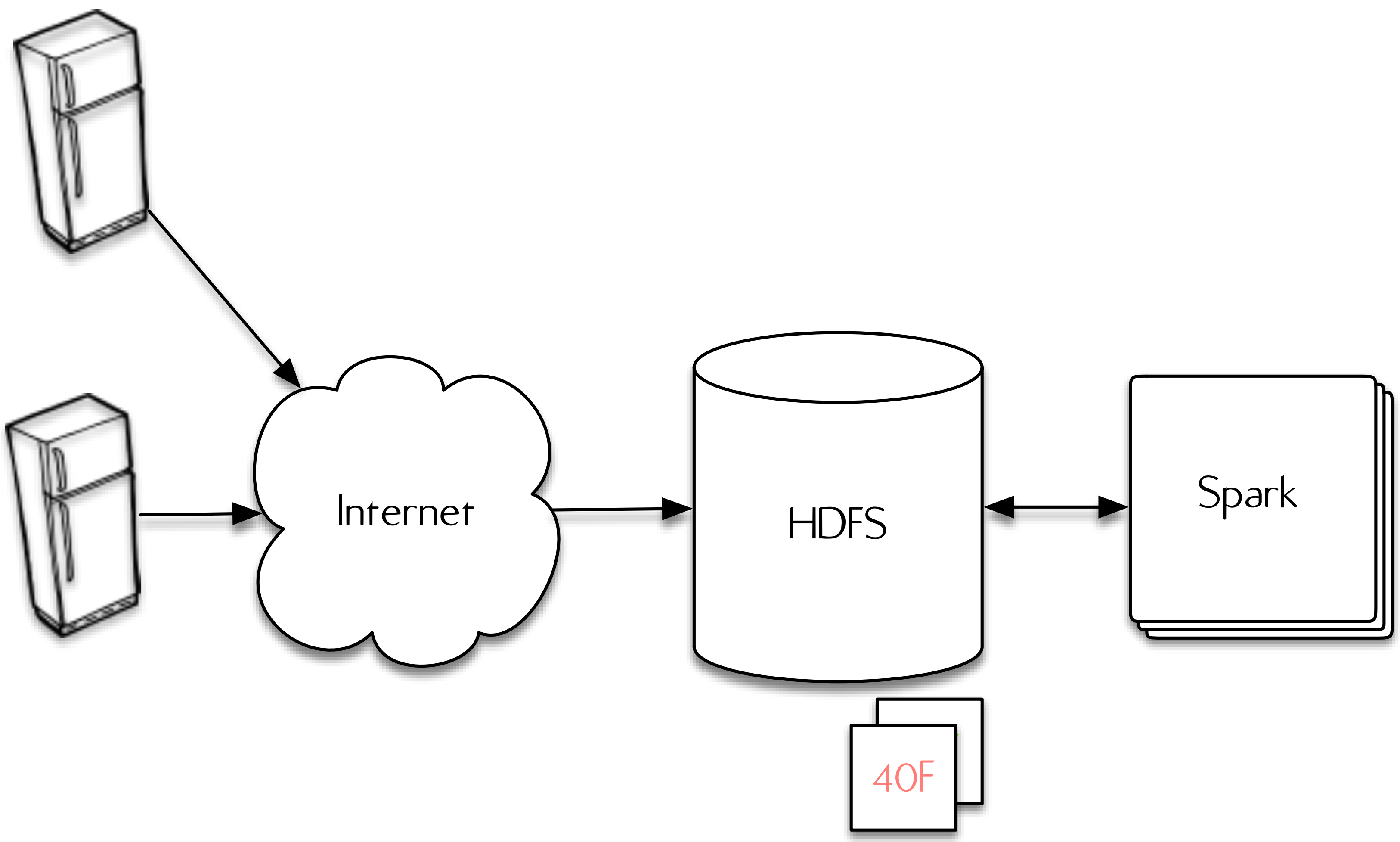
Hospital Refrigerators
Ideal Execution

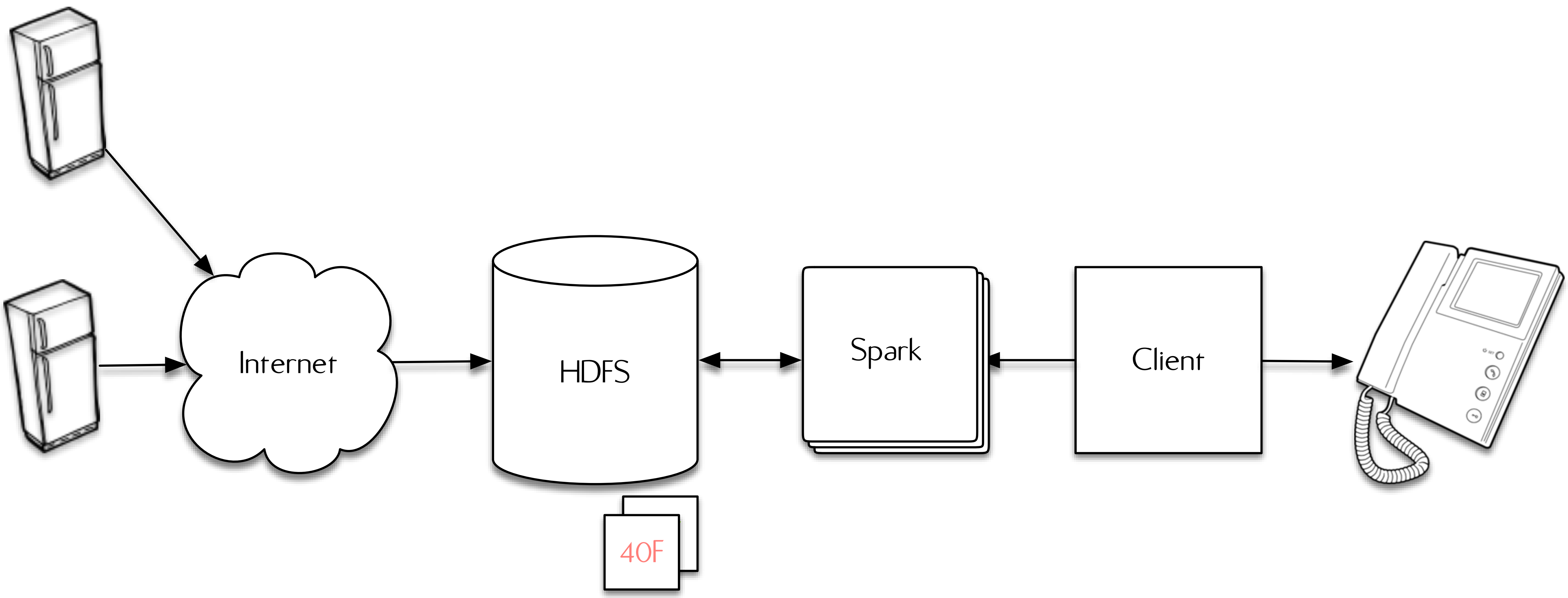

Internet
HDFS
Spark


Internet
HDFS
Spark
35F


Internet
HDFS
Spark
35F

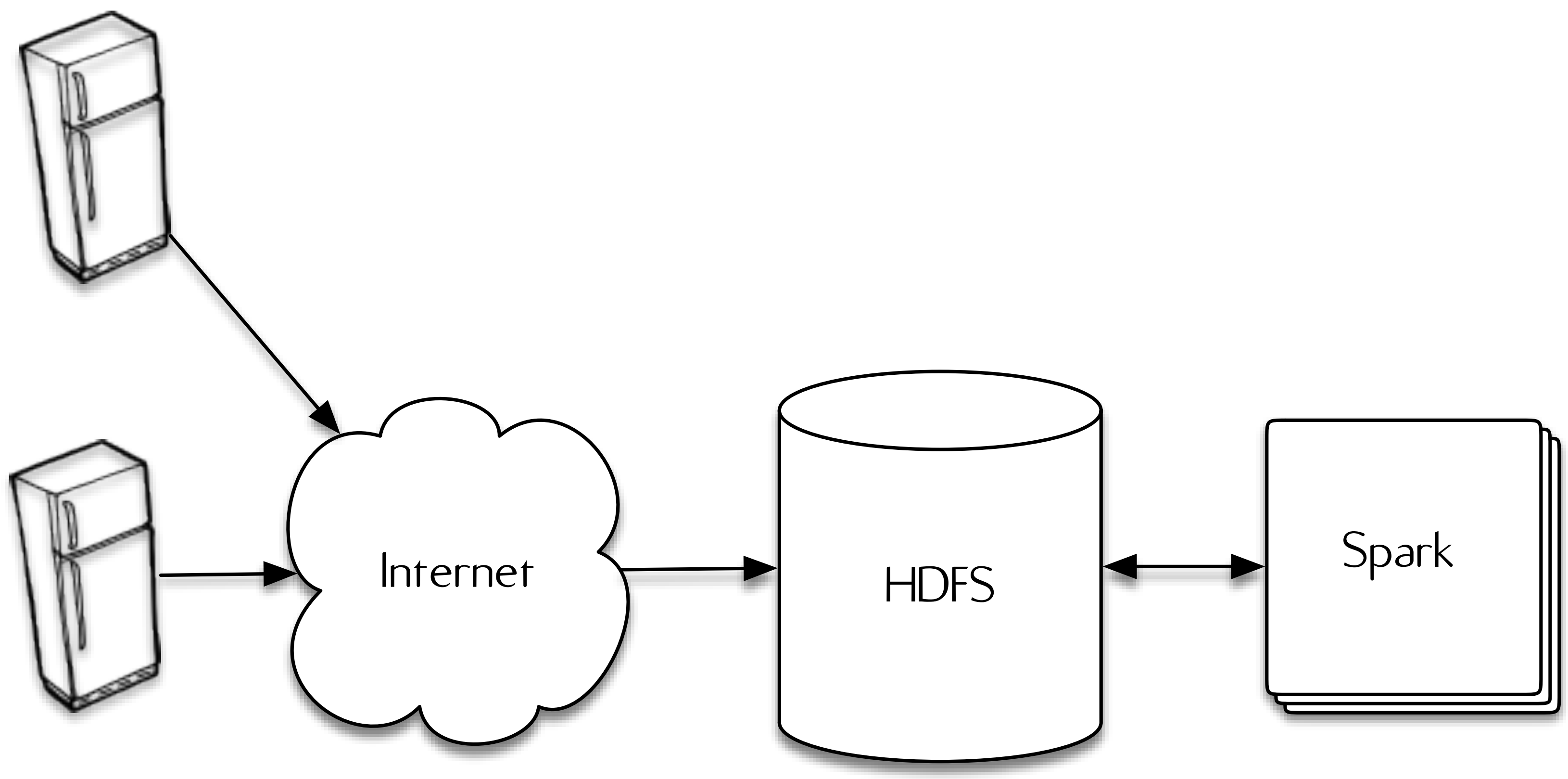

Internet
HDFS
Spark
40F
35F

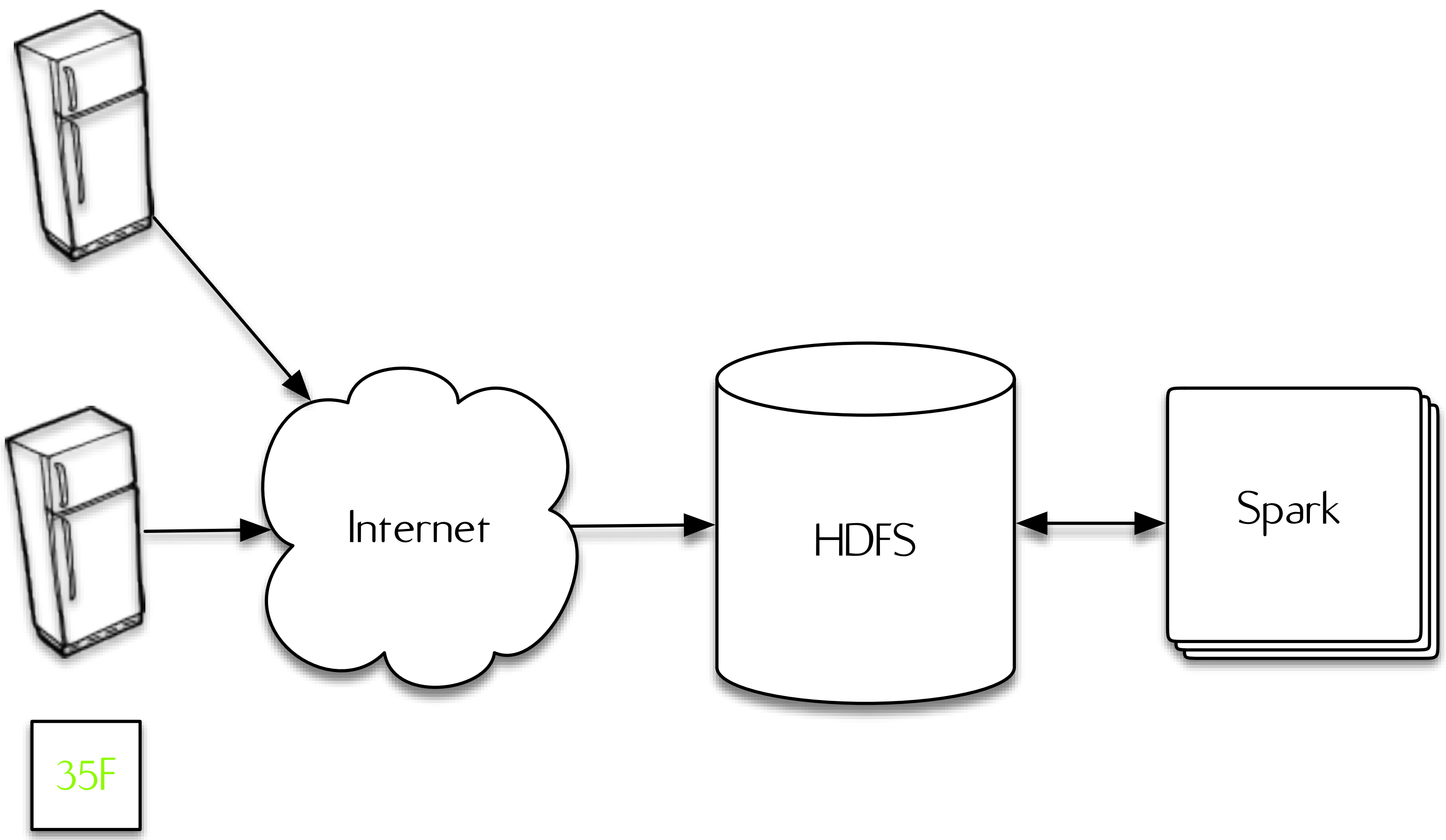

Internet
HDFS
Spark
40F

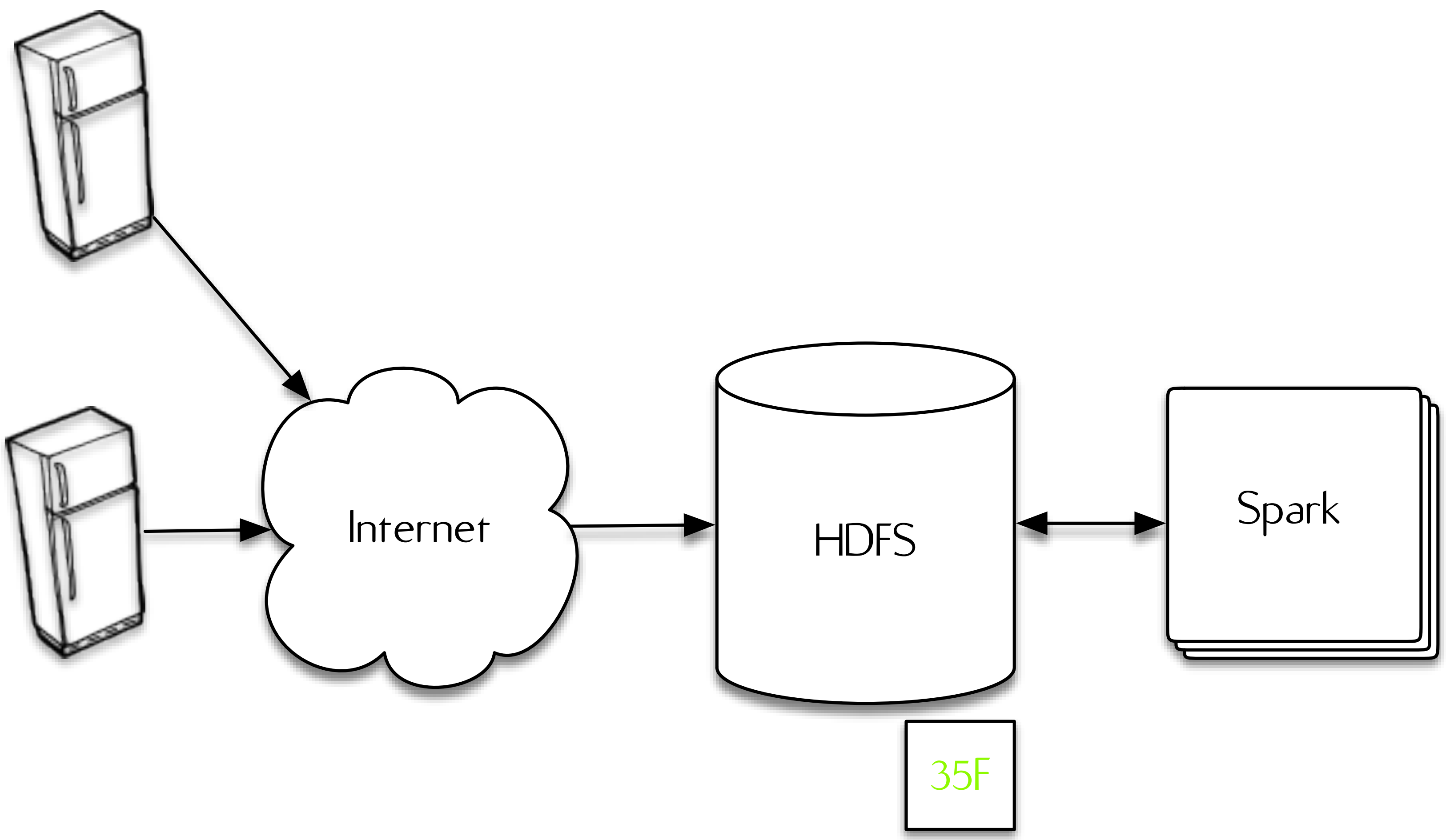

Internet
HDFS
40F
Spark
Client

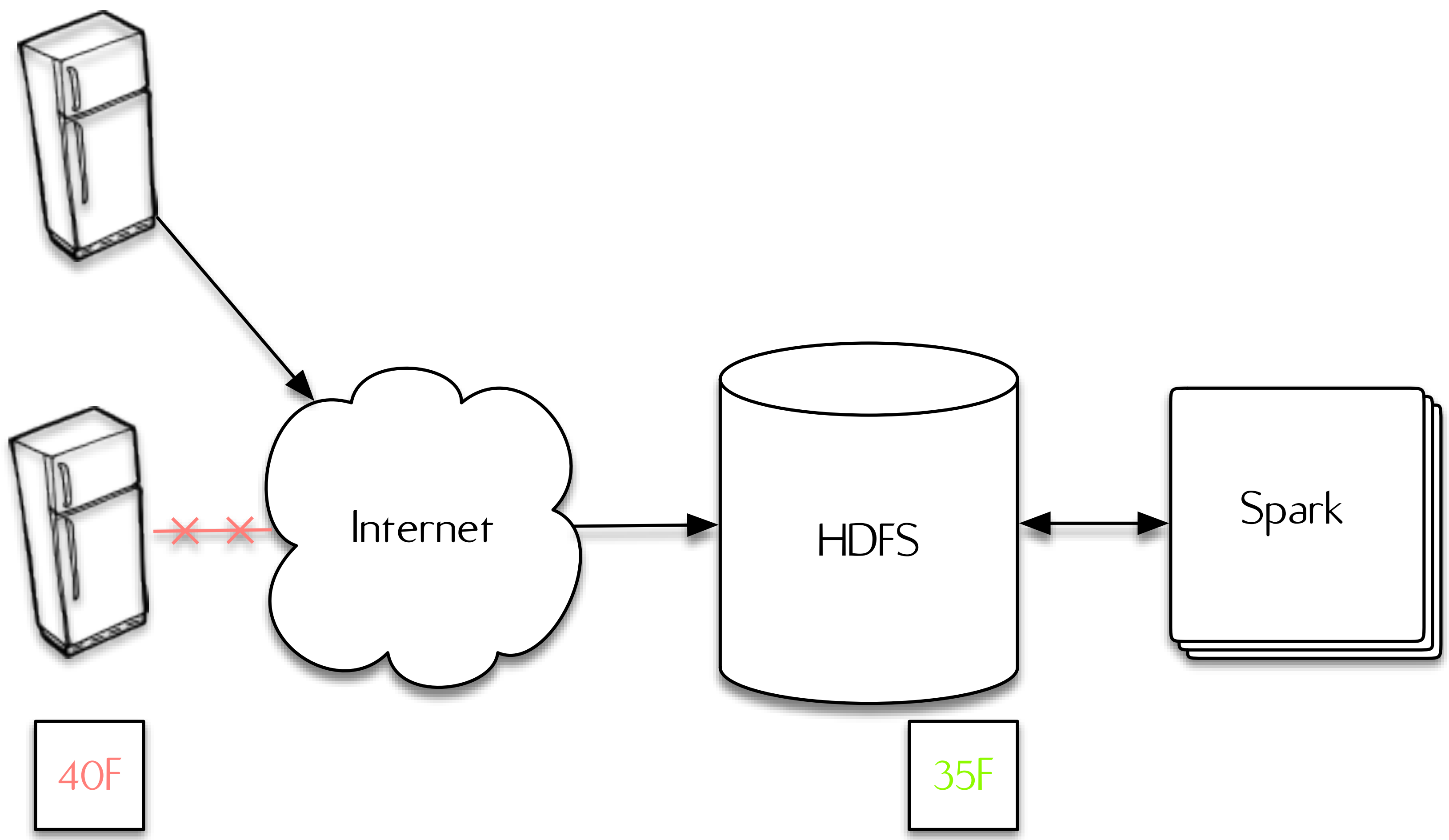
Problem

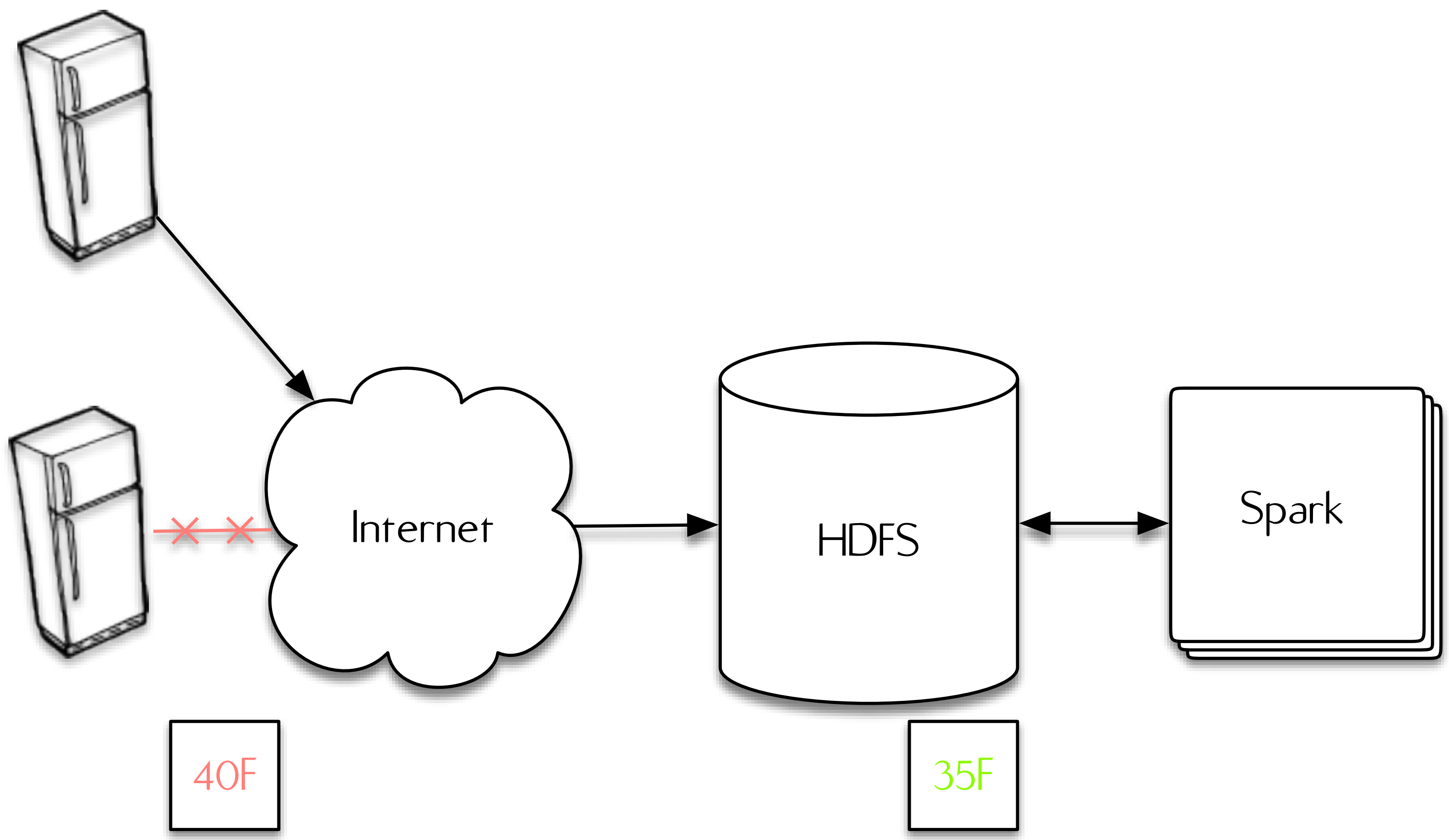
Connectivity

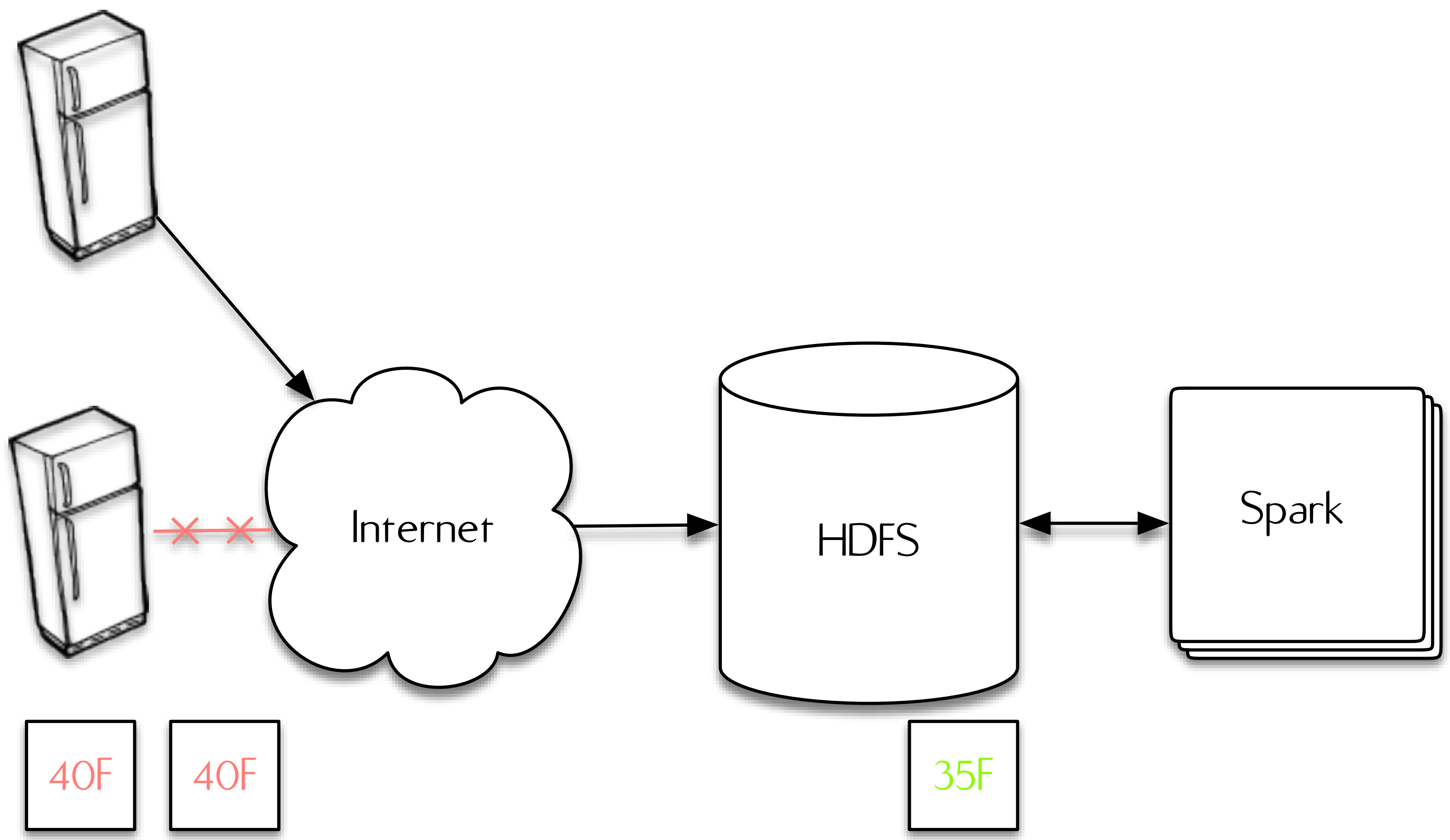

Internet
HDFS
Spark


Internet
HDFS
Spark
35F

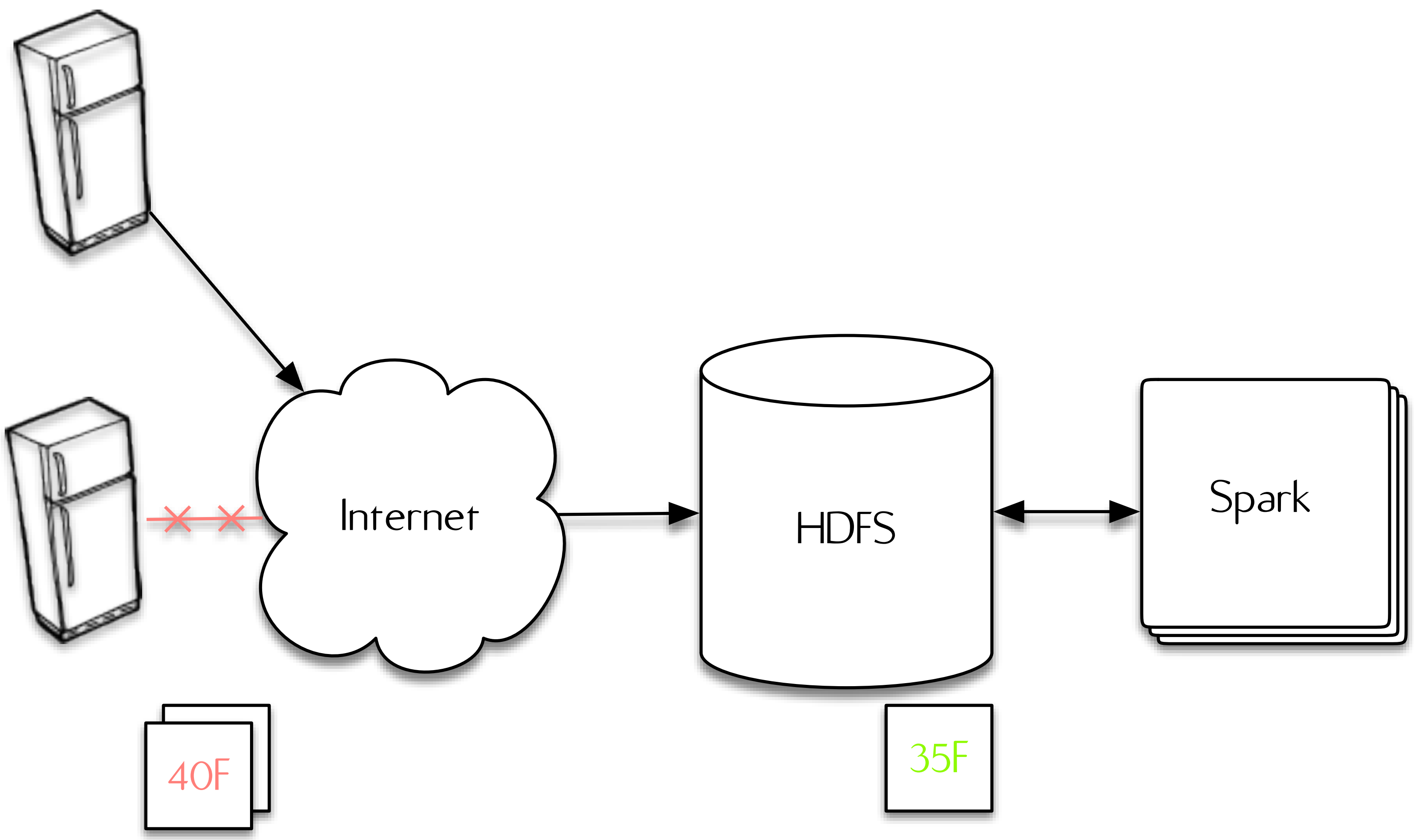

Internet
HDFS
Spark
35F


Internet
HDFS
Spark
40F
35F


Internet
HDFS
Spark
40F
35F

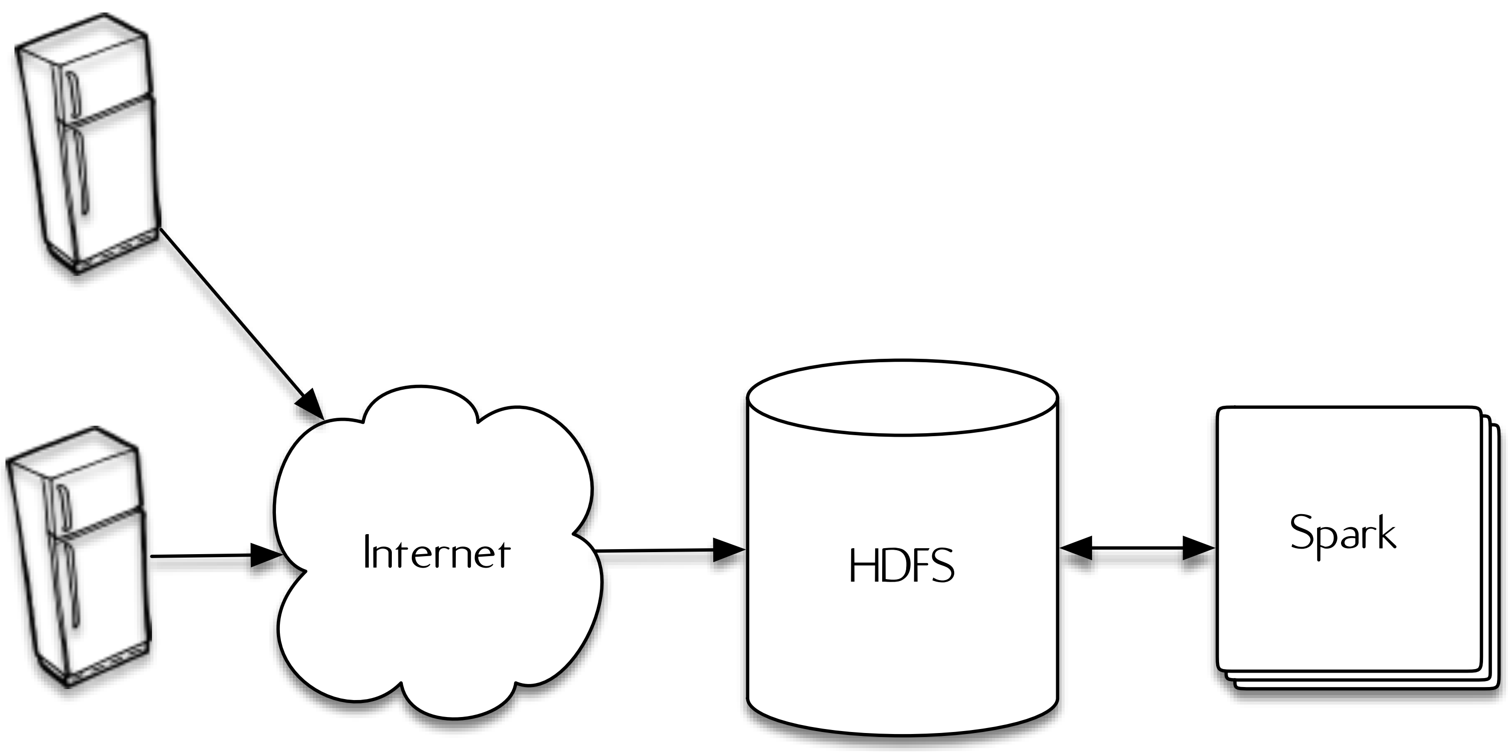


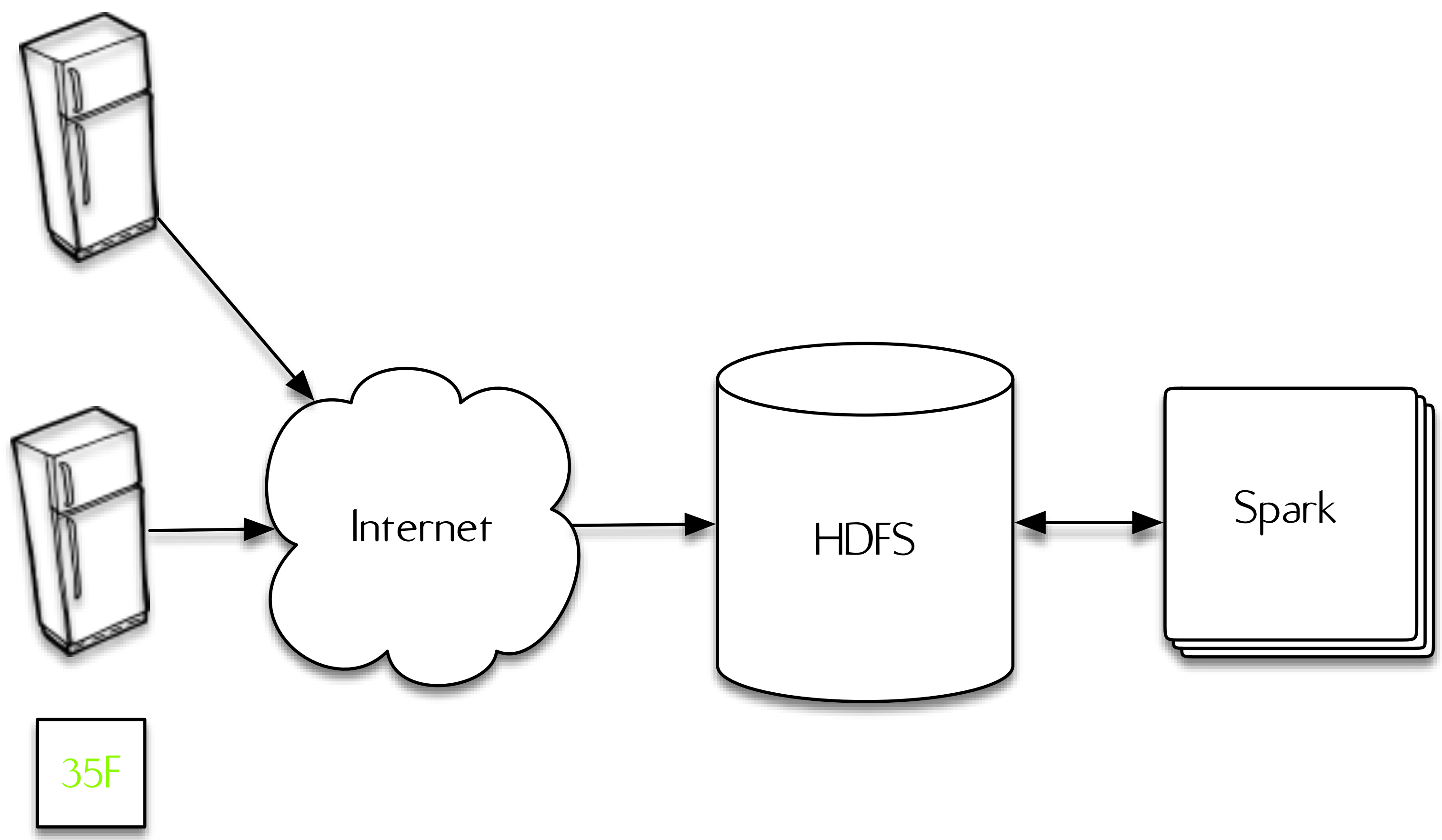
Internet
HDFS
Spark
40F
40F
35F

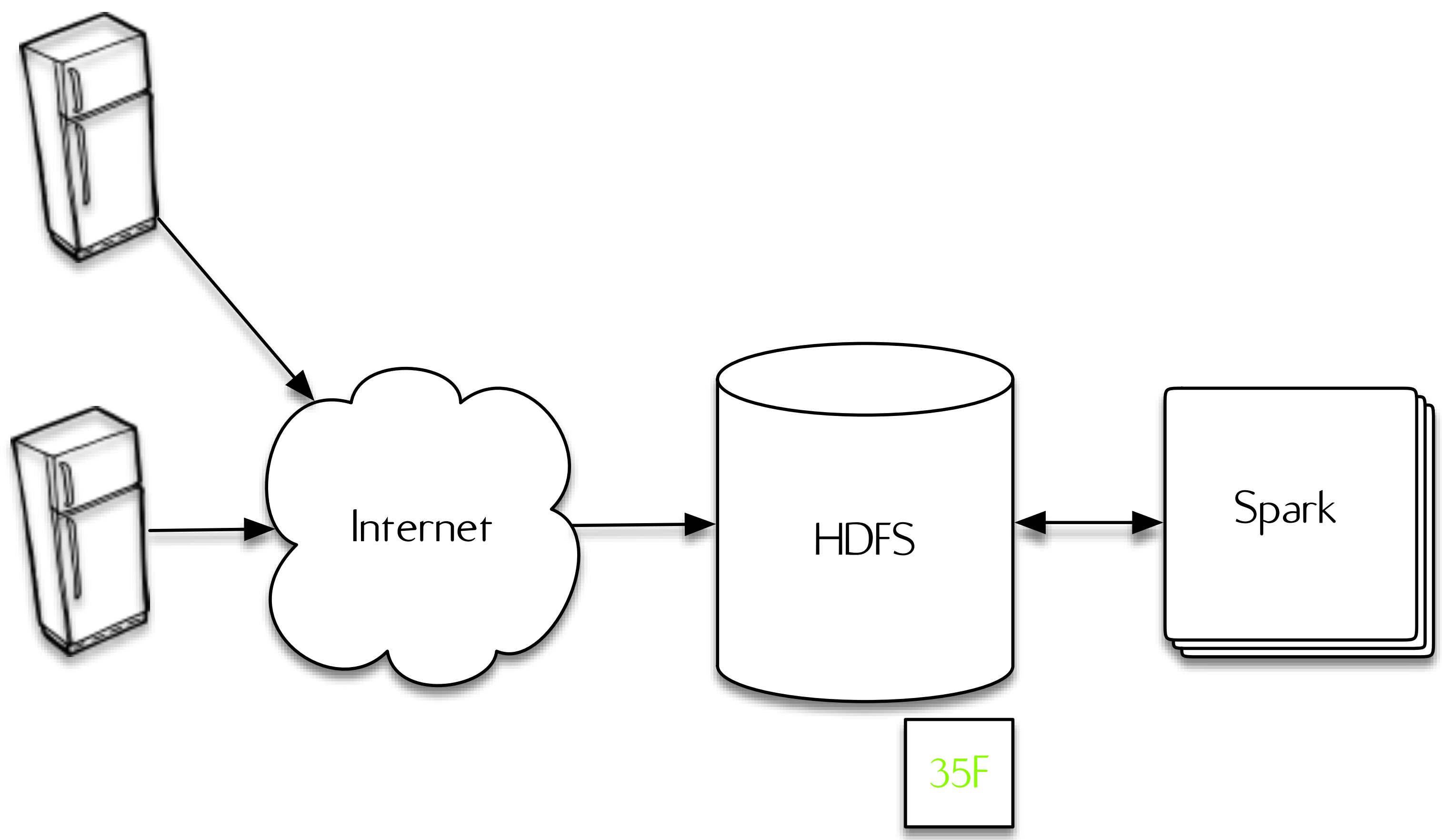

Internet
HDFS
Spark
40F
35F

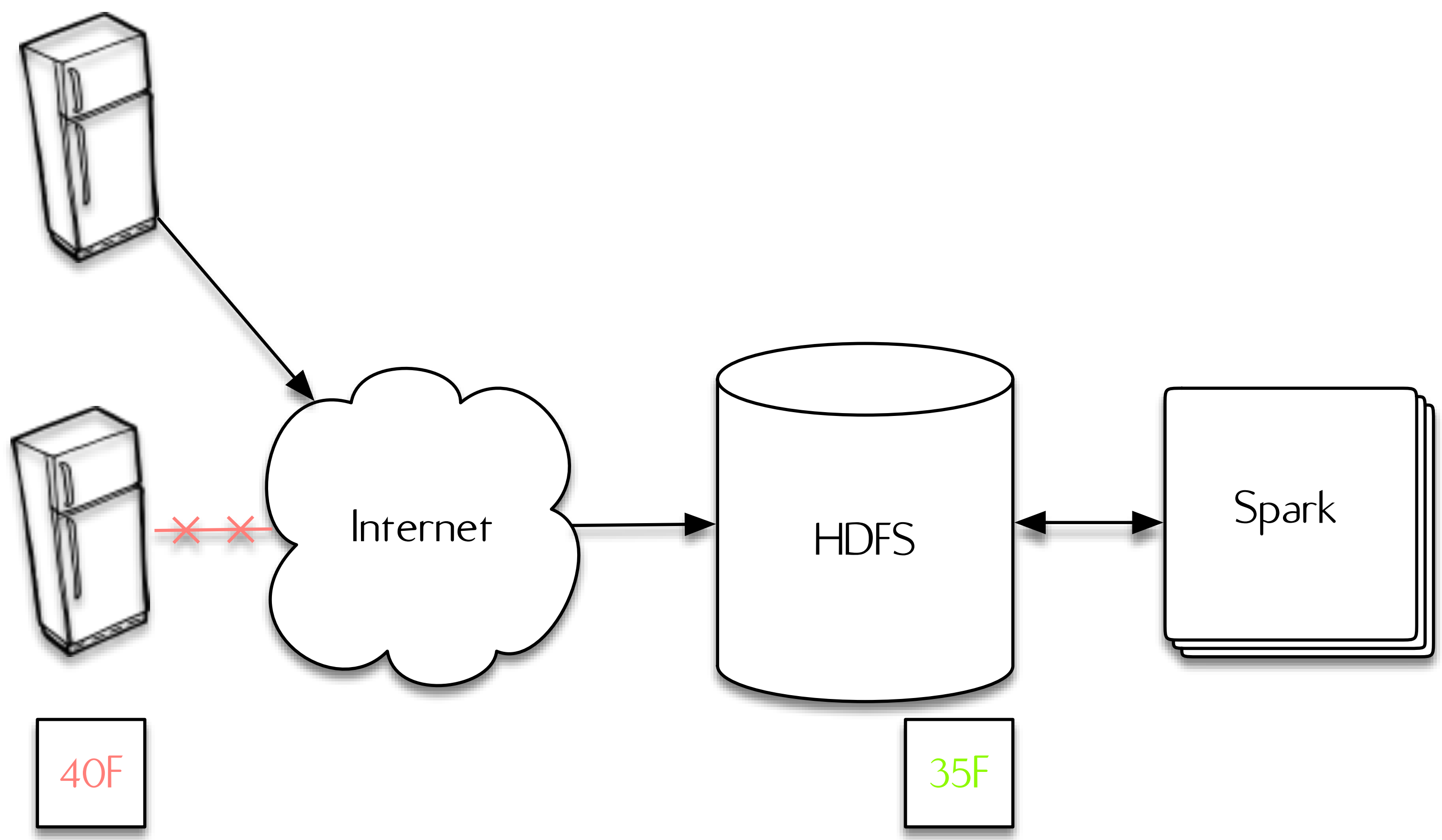
Solution

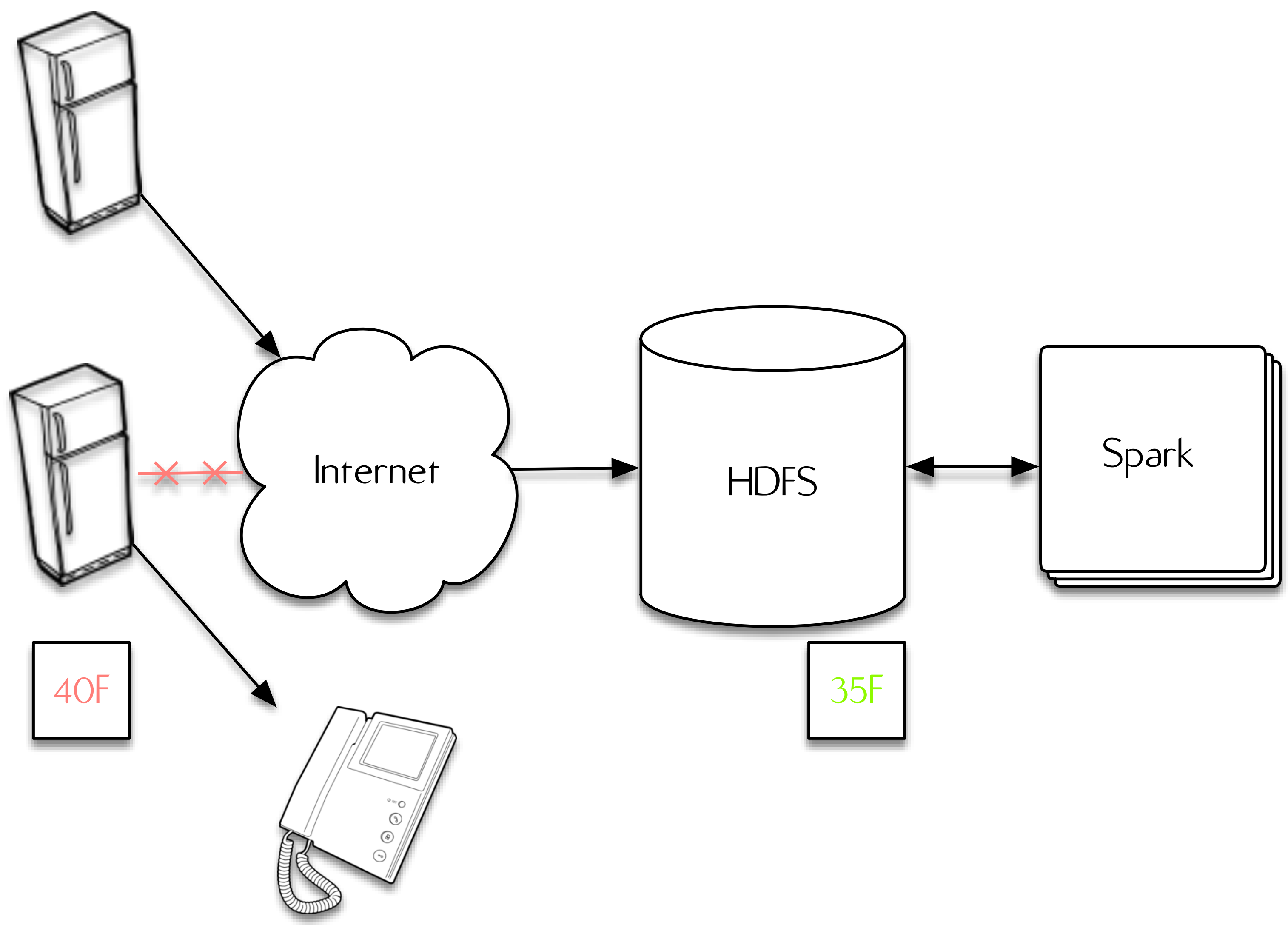
Local Decisions


Internet
HDFS
Spark


Internet
HDFS
Spark
35F

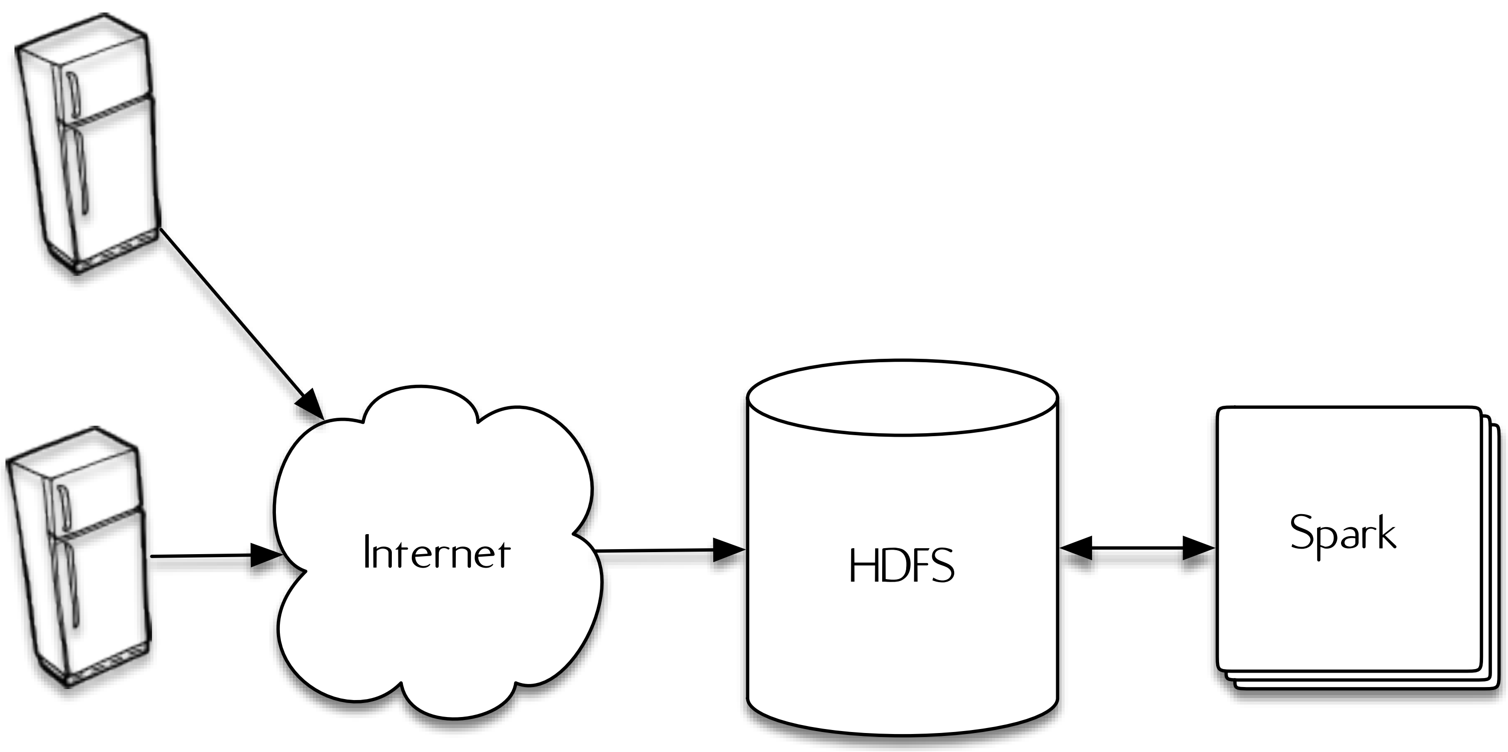

Internet
HDFS
Spark
35F

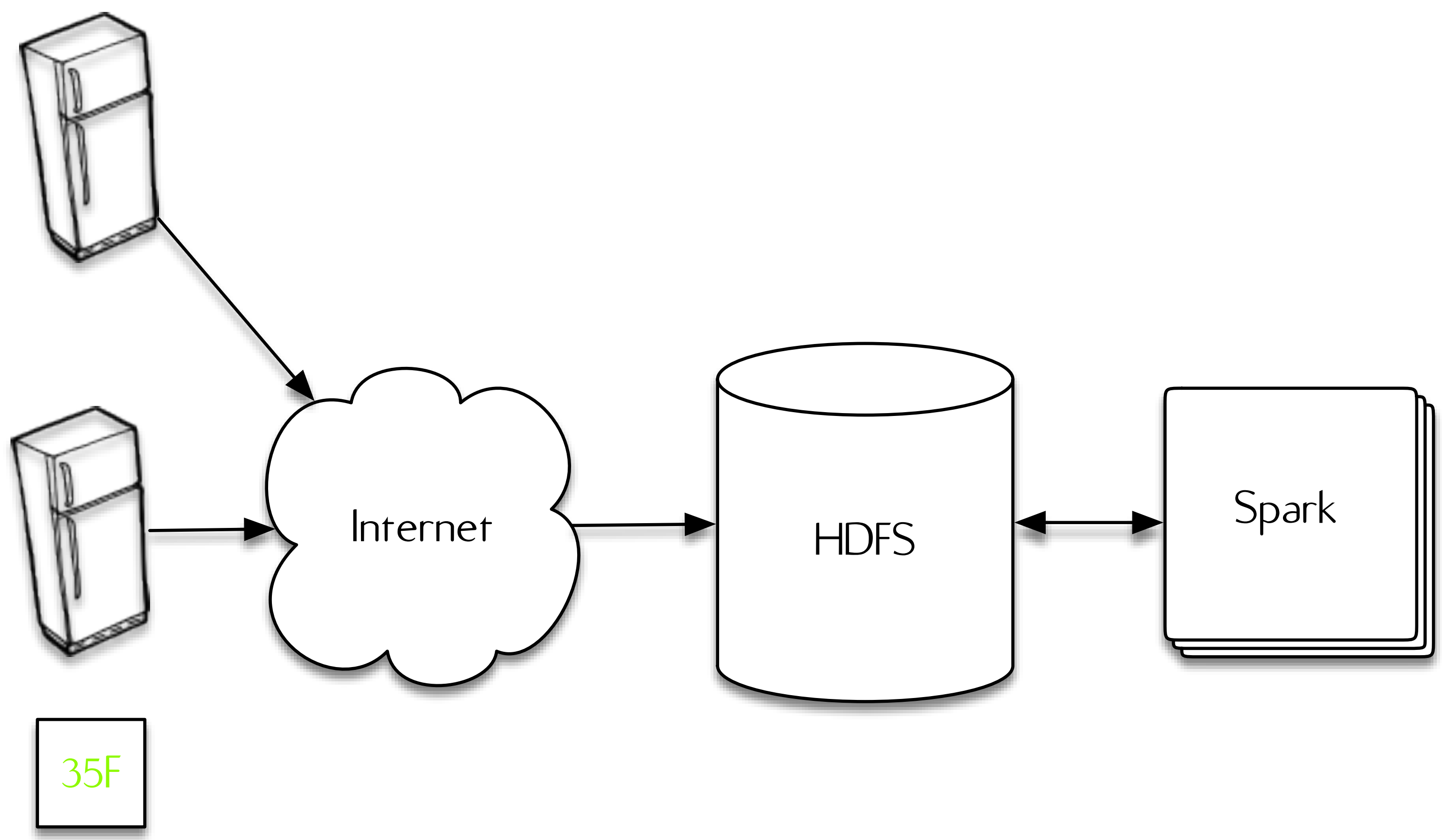

Internet
HDFS
Spark
40F
35F

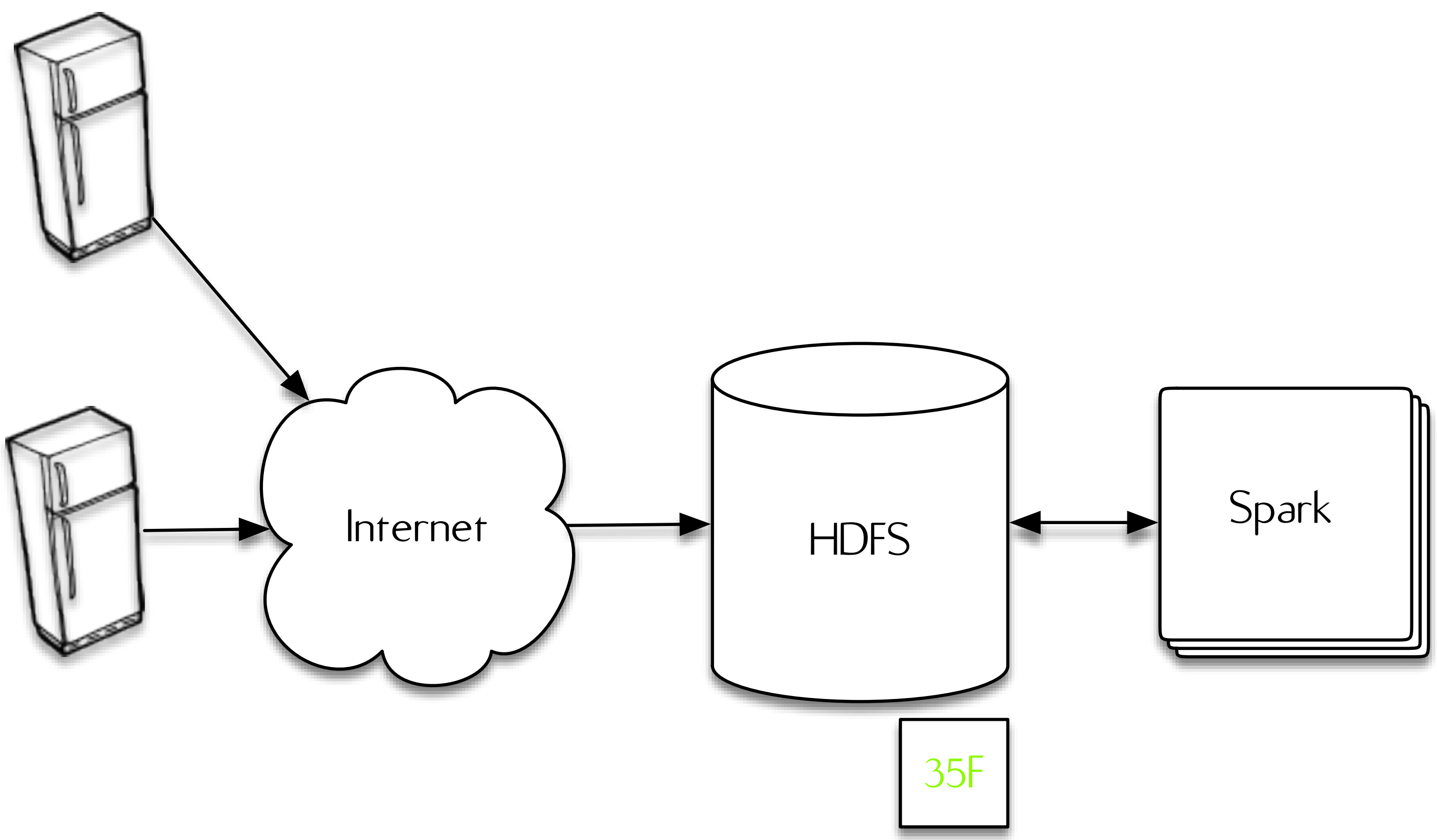

Internet
HDFS
Spark
40F
35F

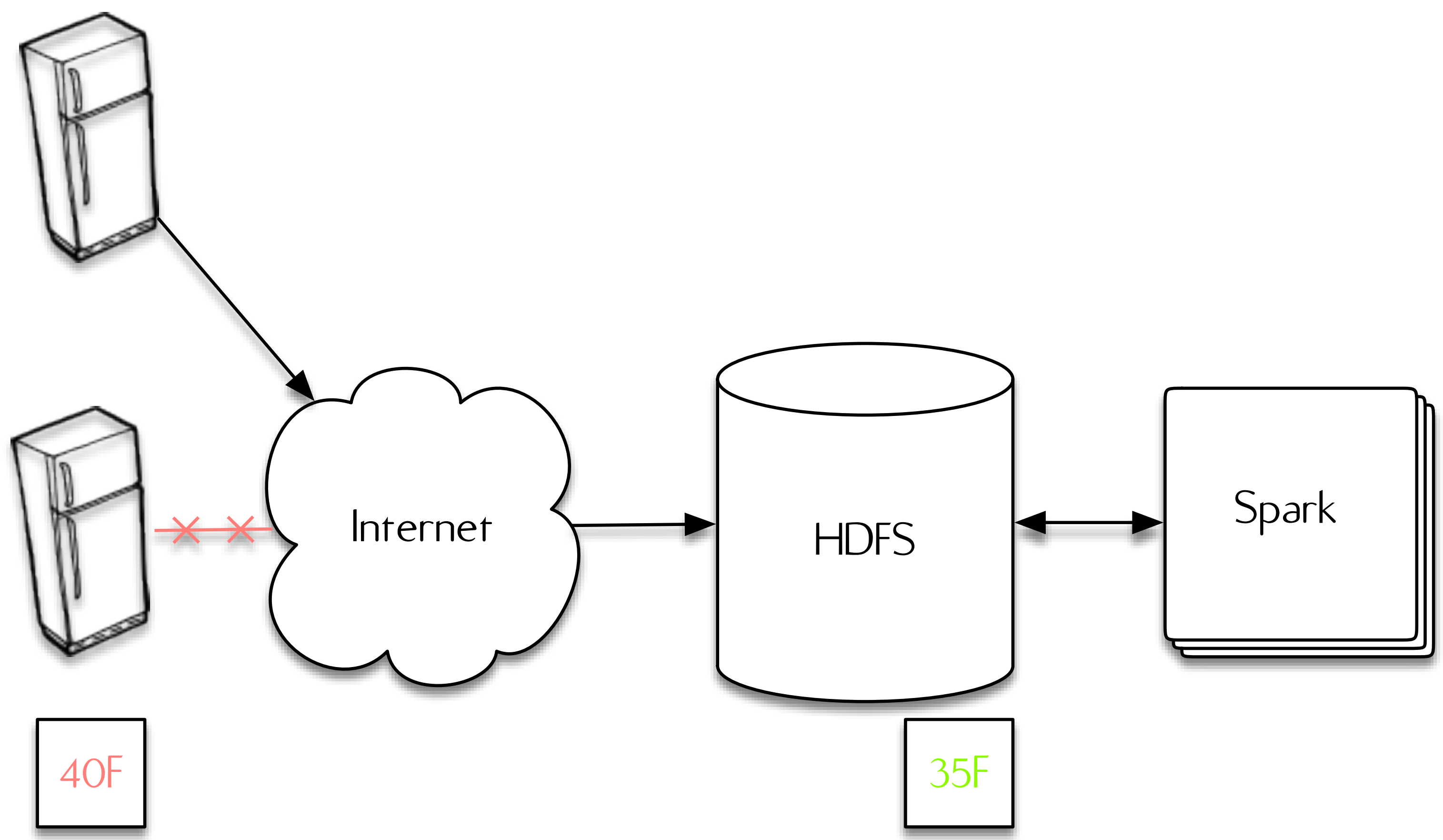
Solution

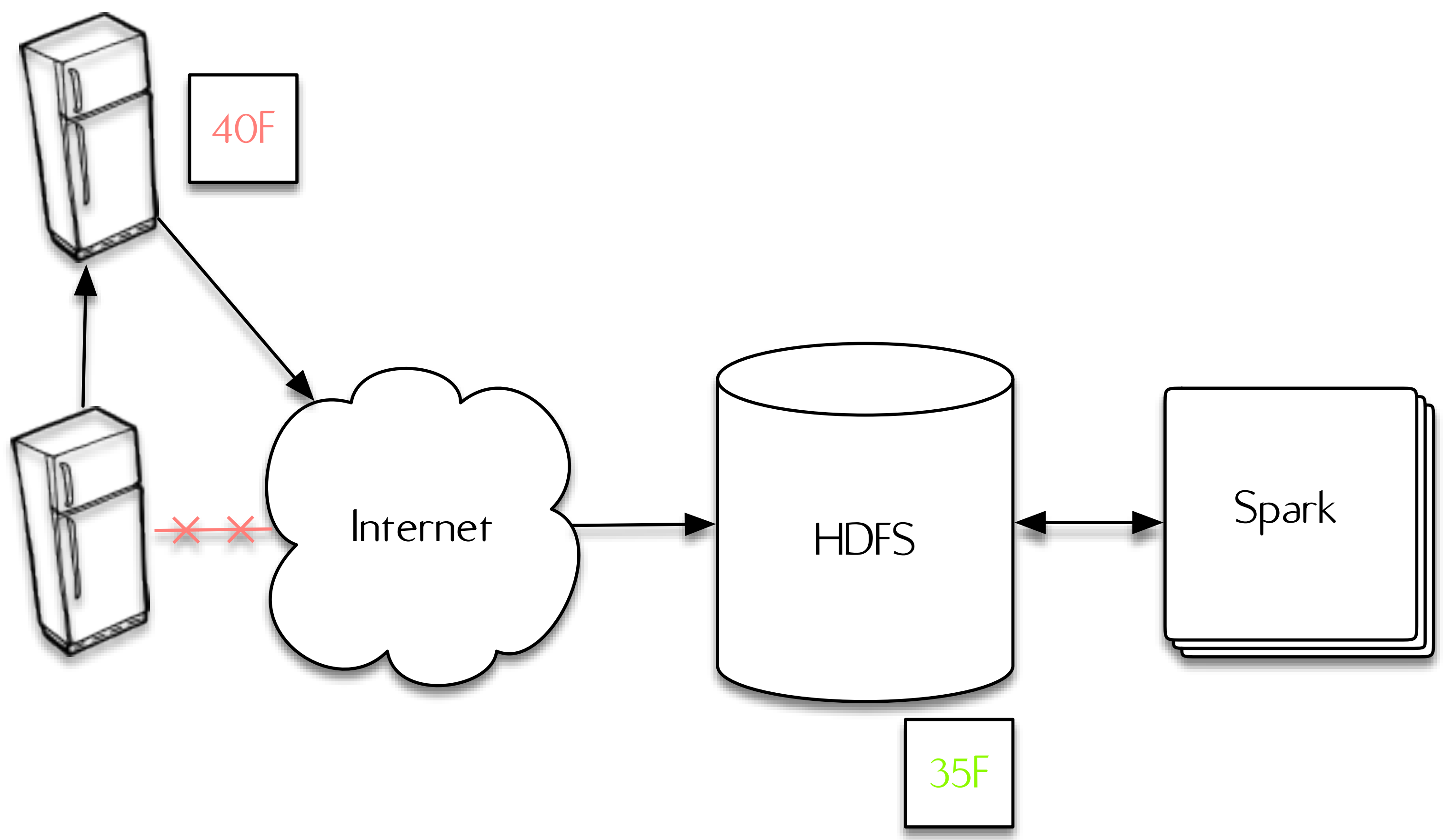
Transitive Dissemination

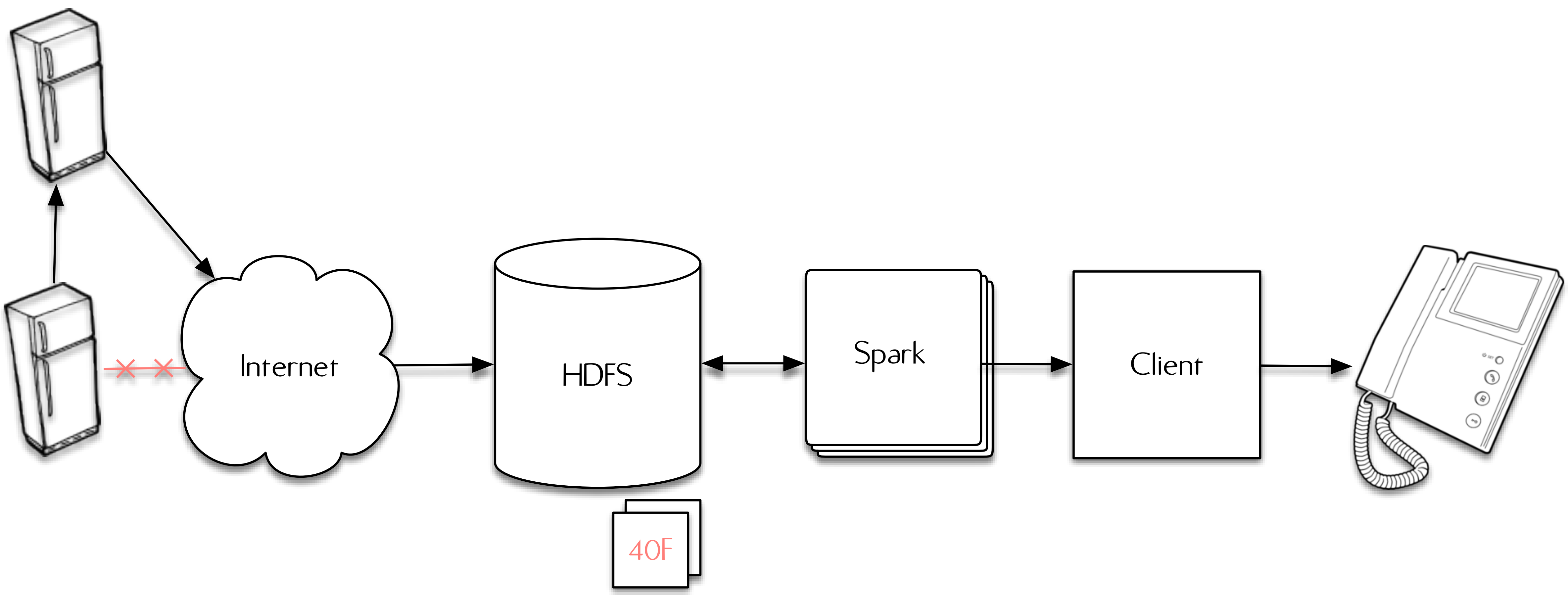

Internet
HDFS
Spark


Internet
HDFS
Spark
35F


Internet
HDFS
Spark
35F


Internet
HDFS
Spark
40F
35F

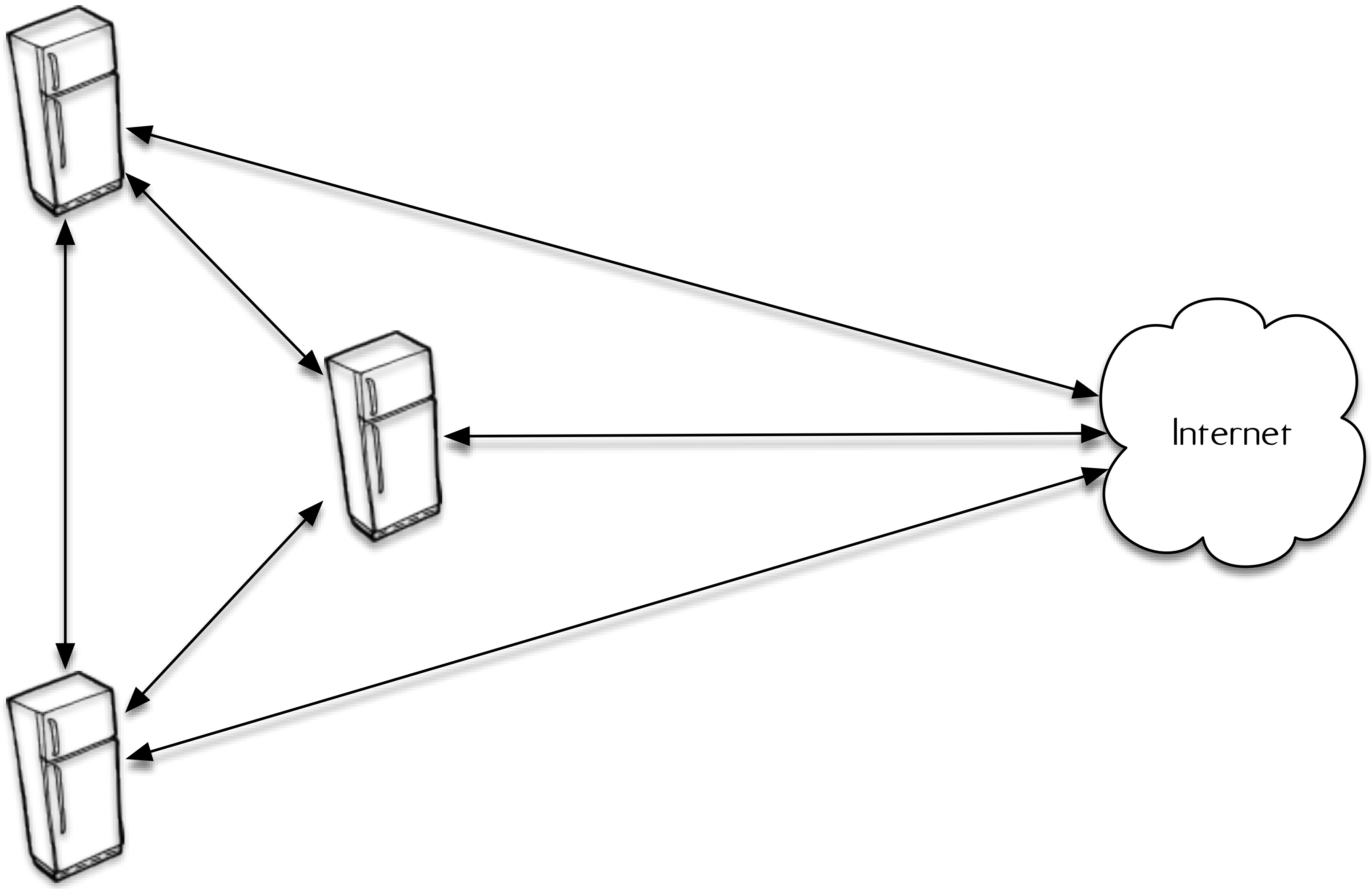

40F
Internet
HDFS
Spark
35F

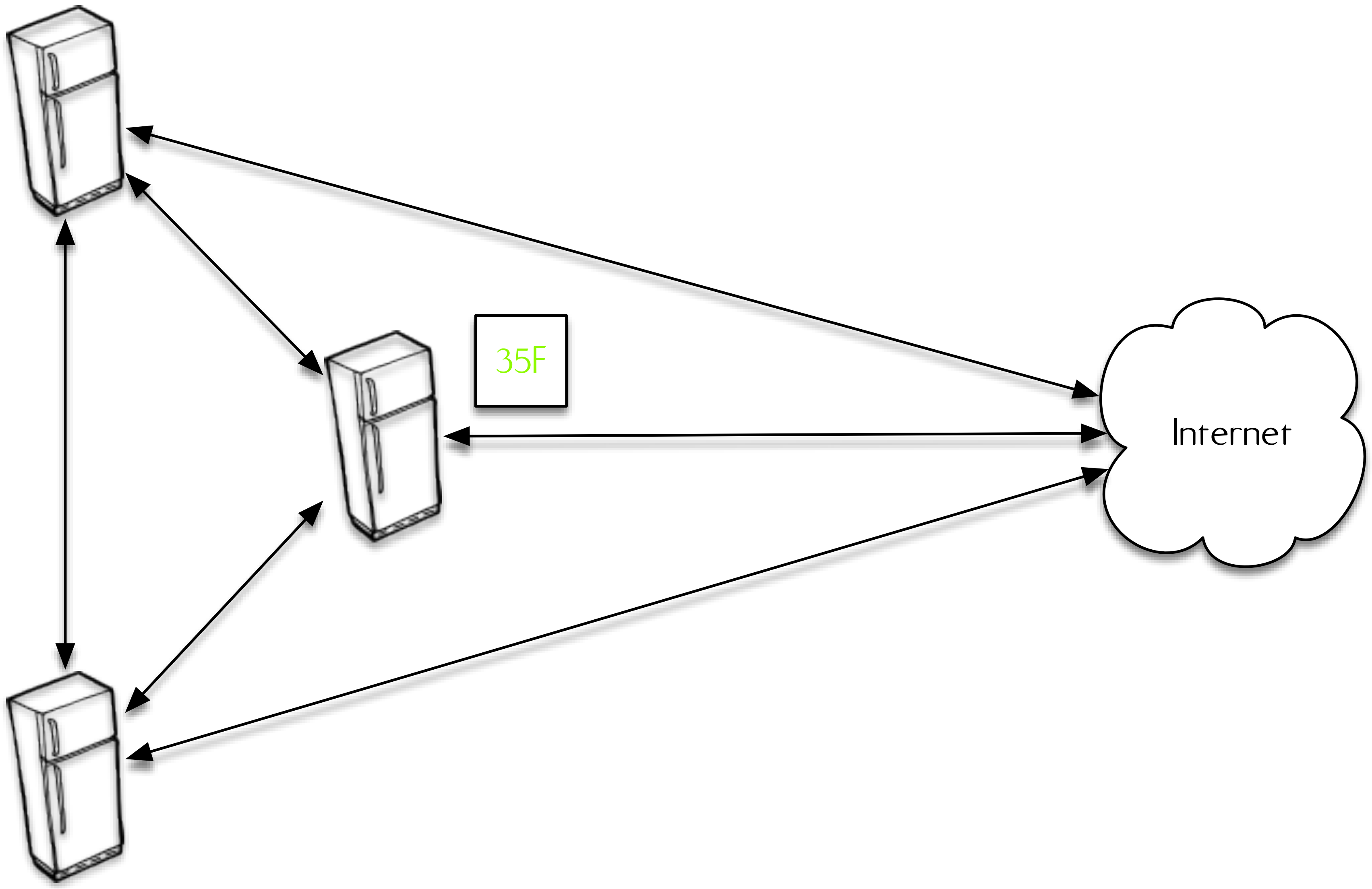


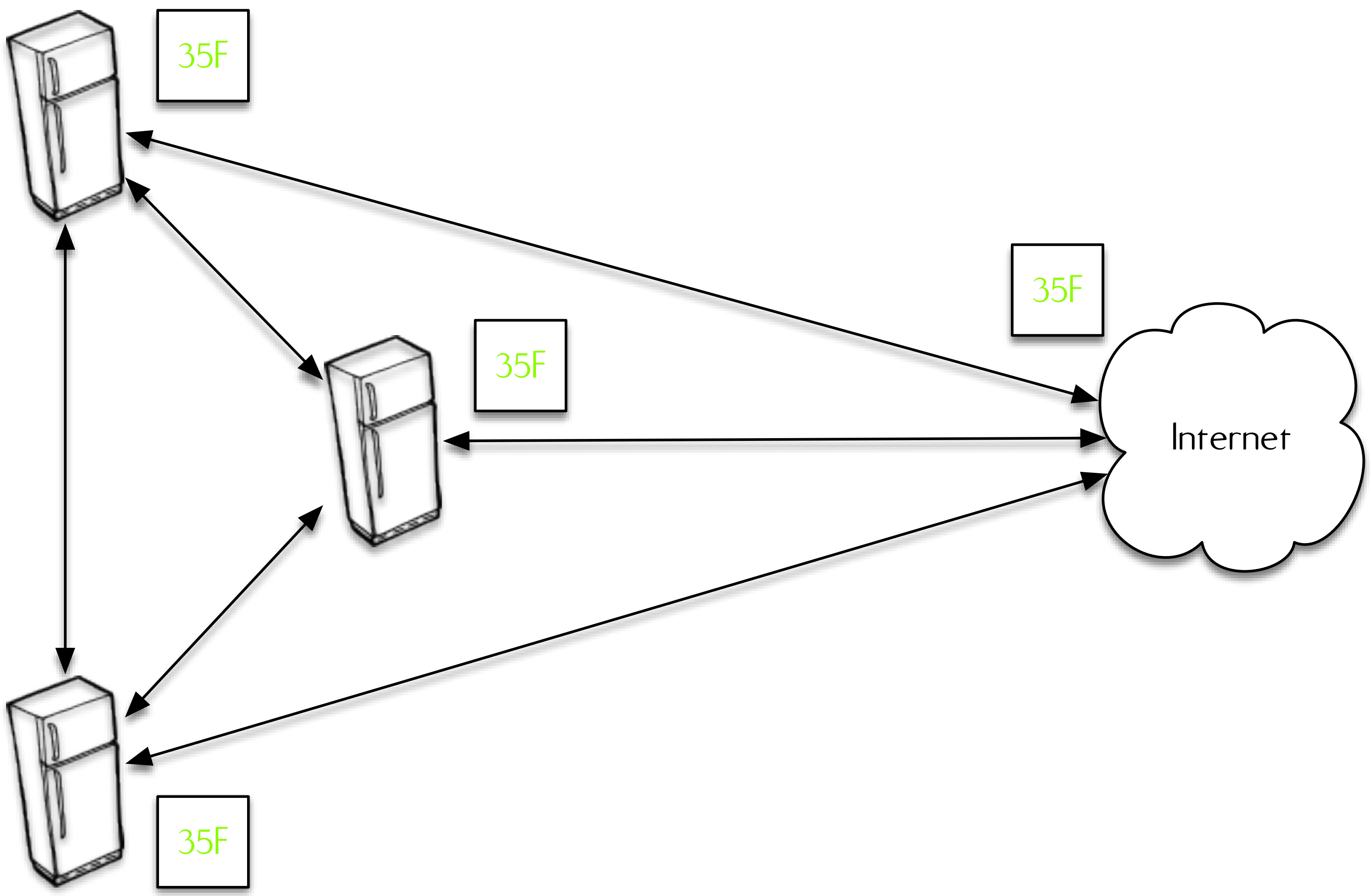
Internet
HDFS
40F
Spark
Client

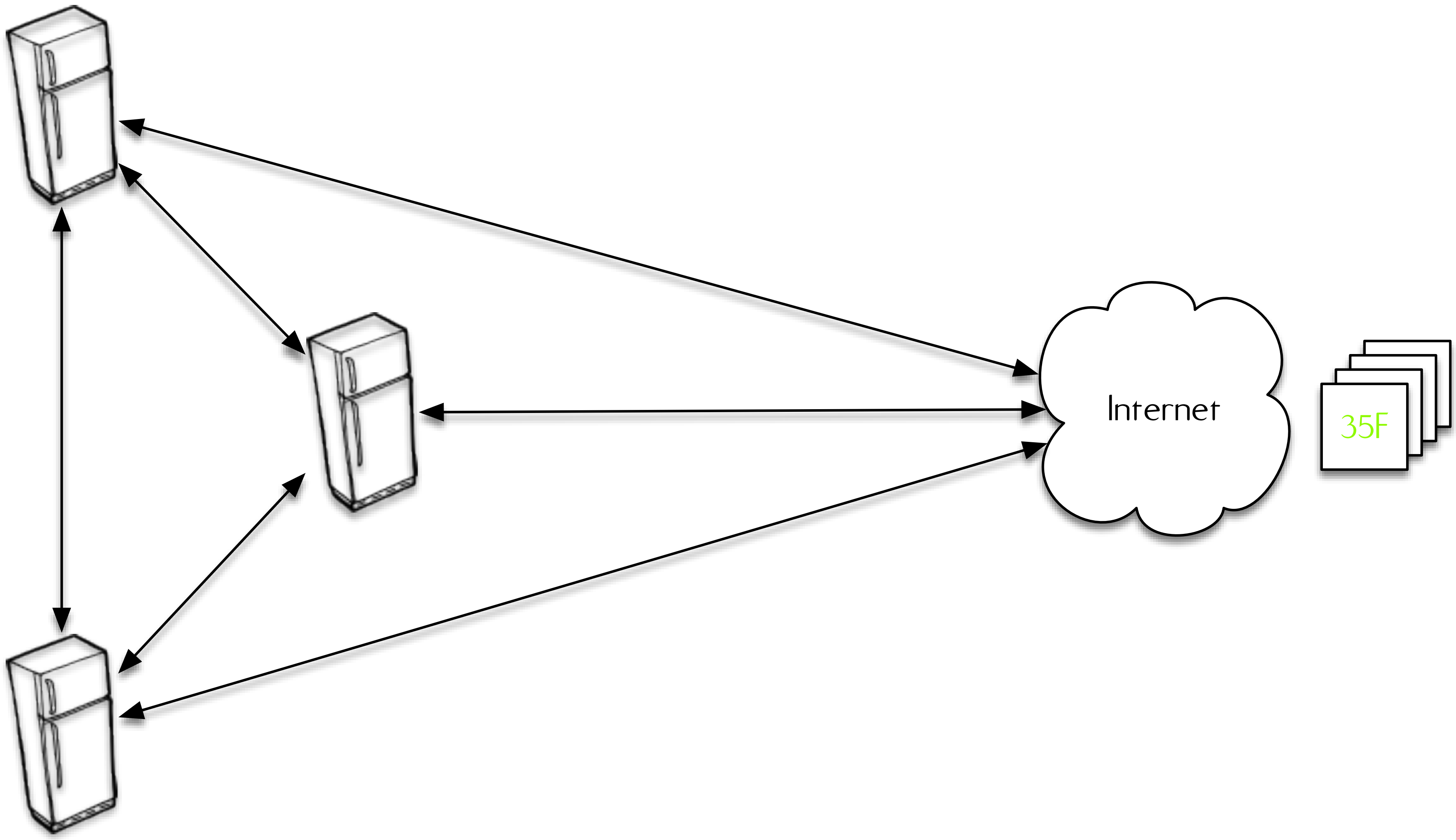
Problem

State Transmission


Internet

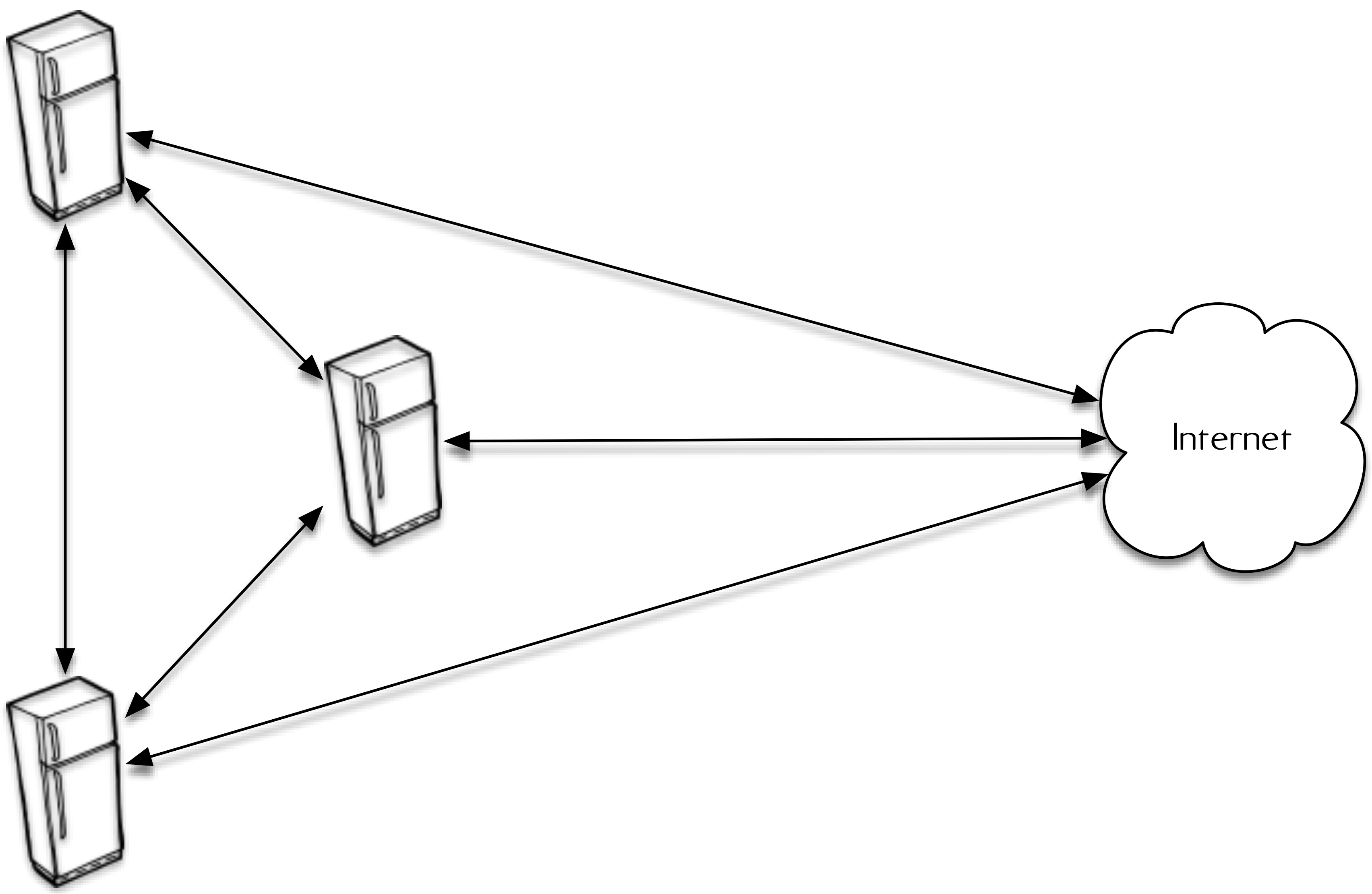

35F
Internet

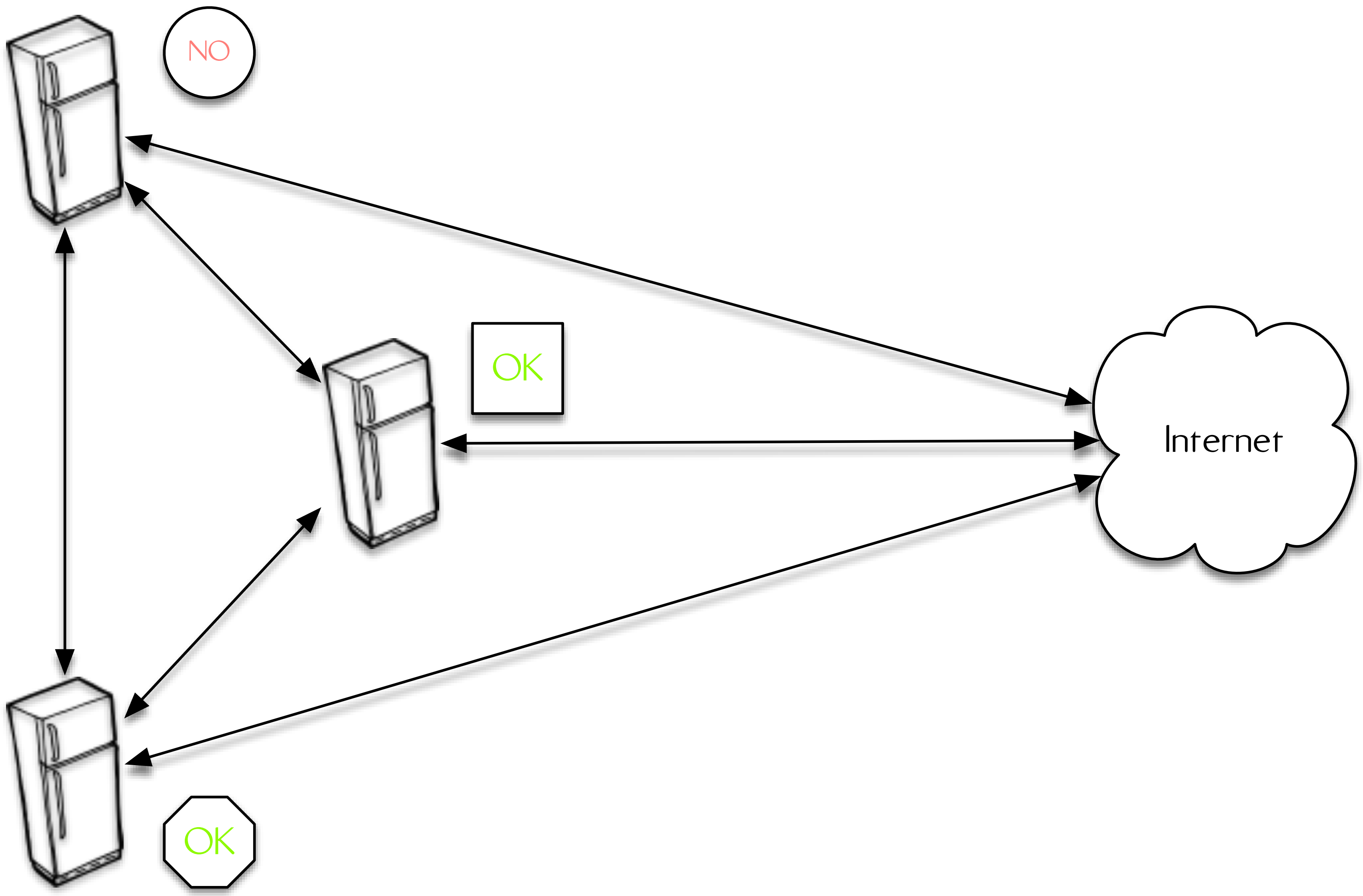

35F
35F
35F
35F
Internet

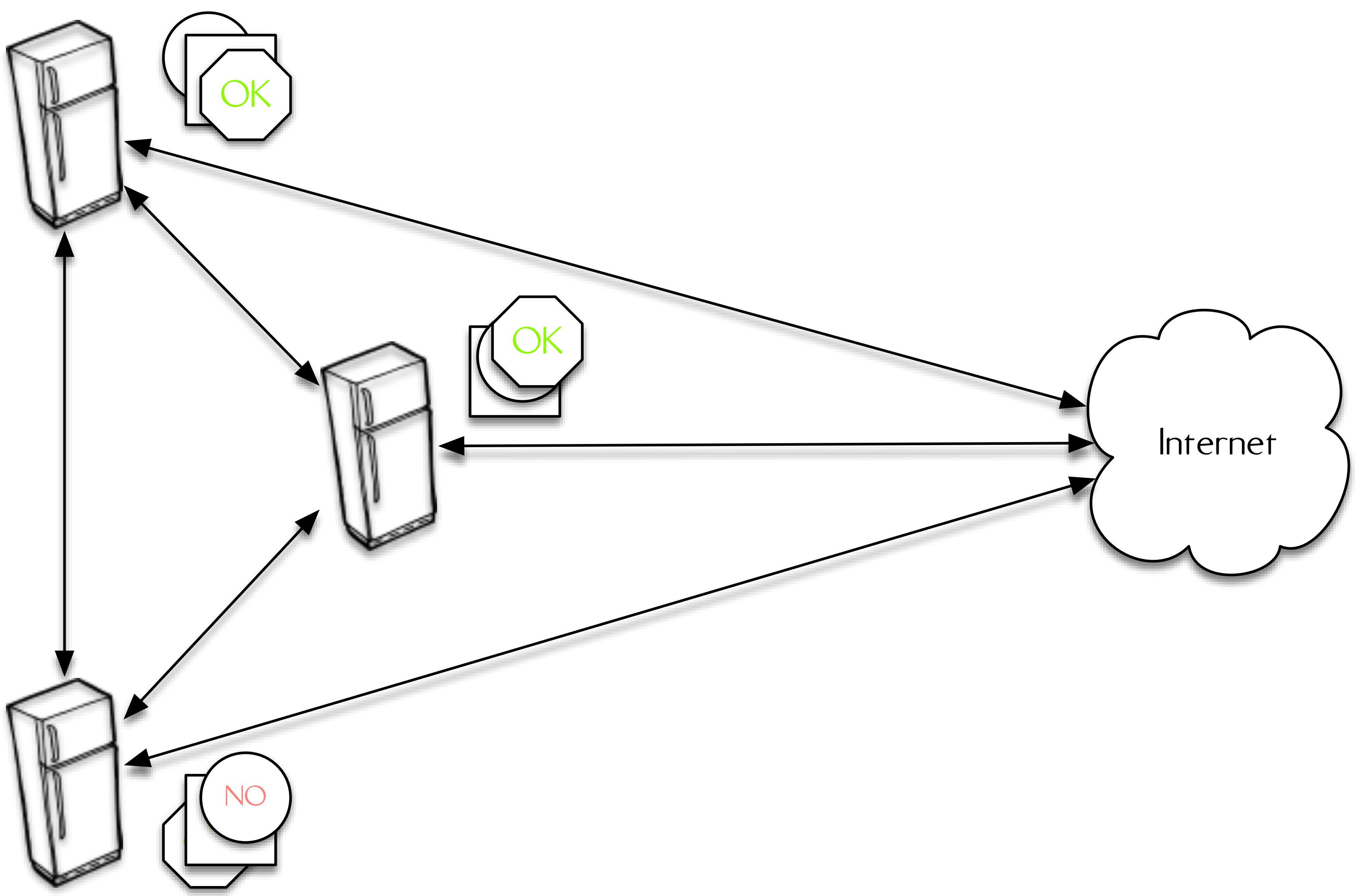

Internet
35F

Solution

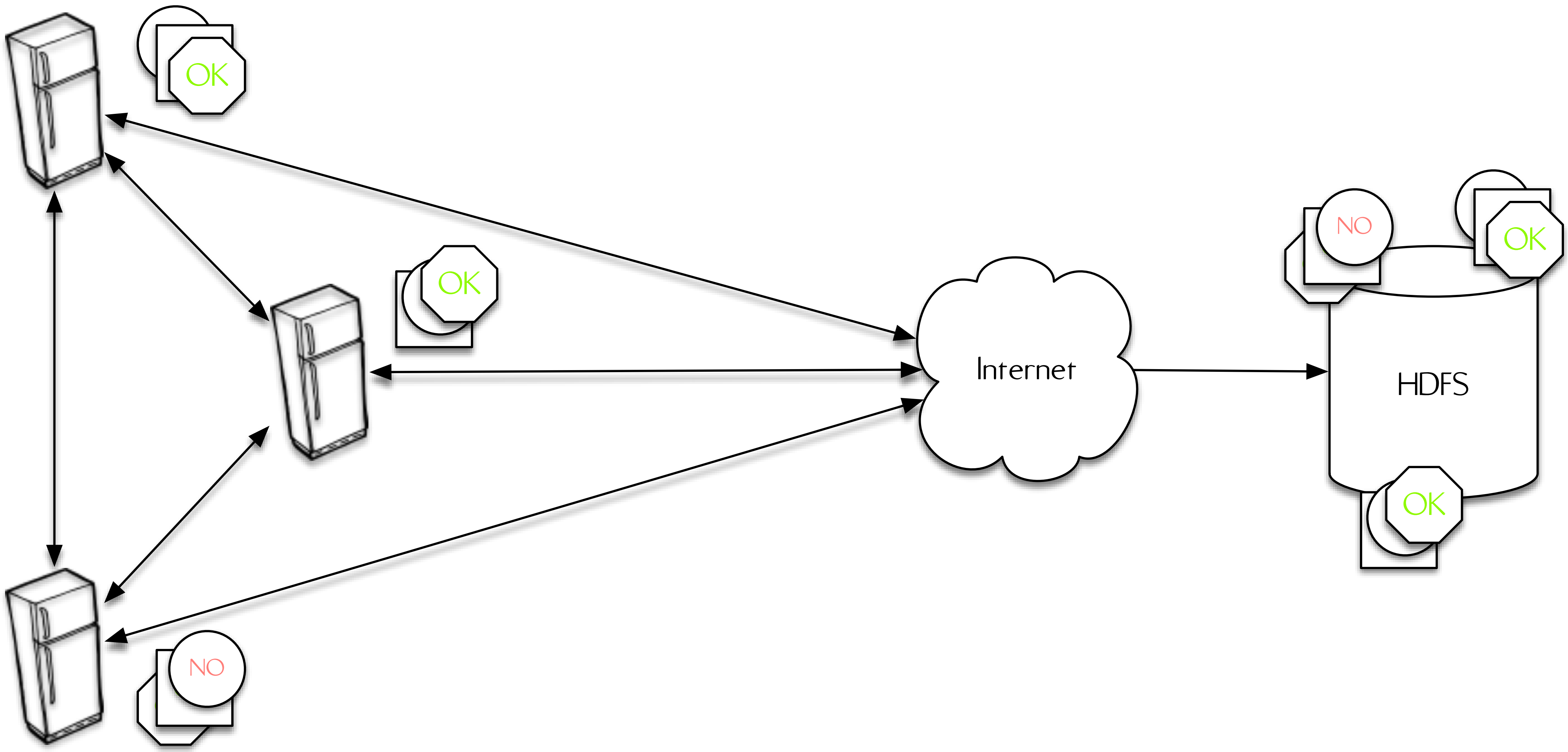
Aggregate Dissemination

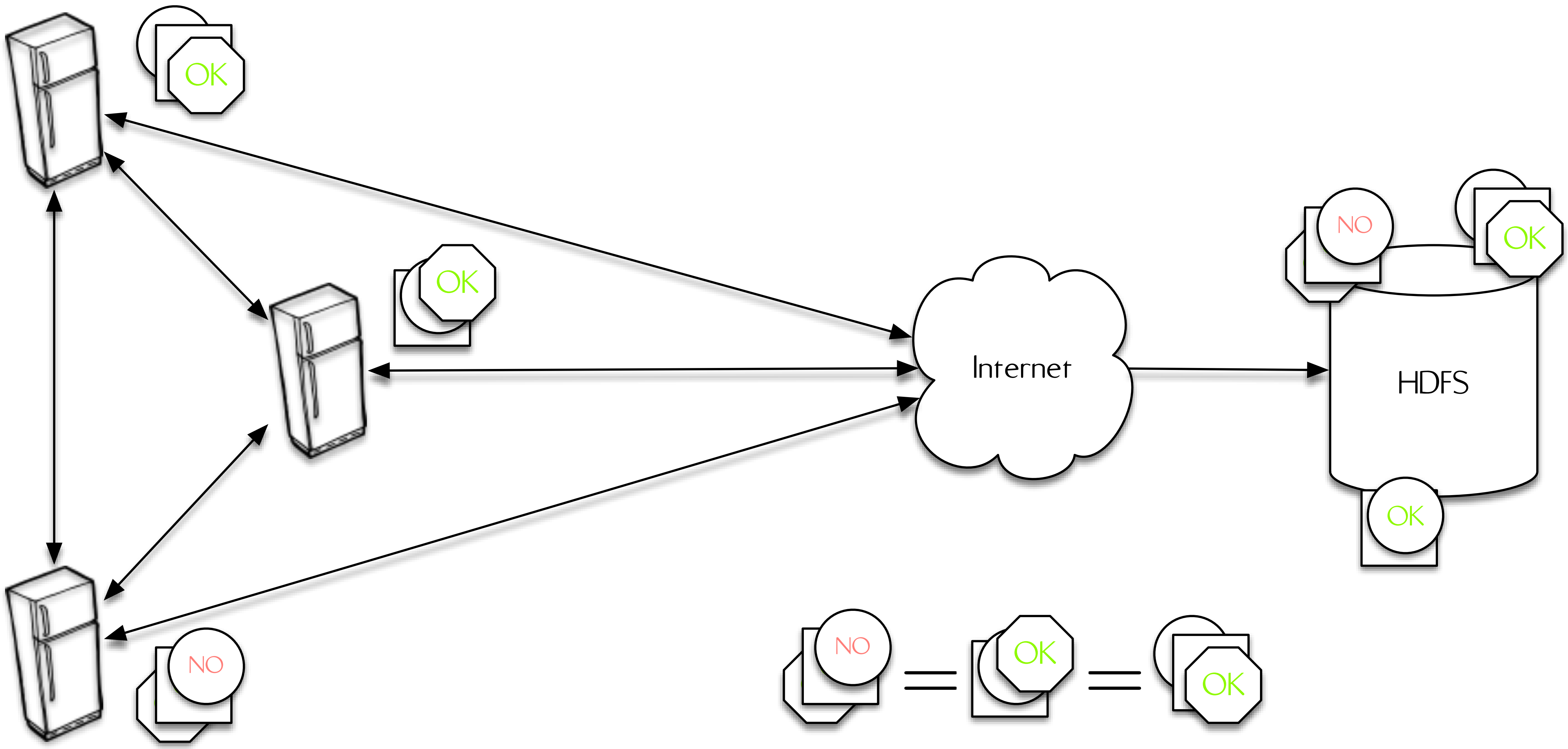

Internet

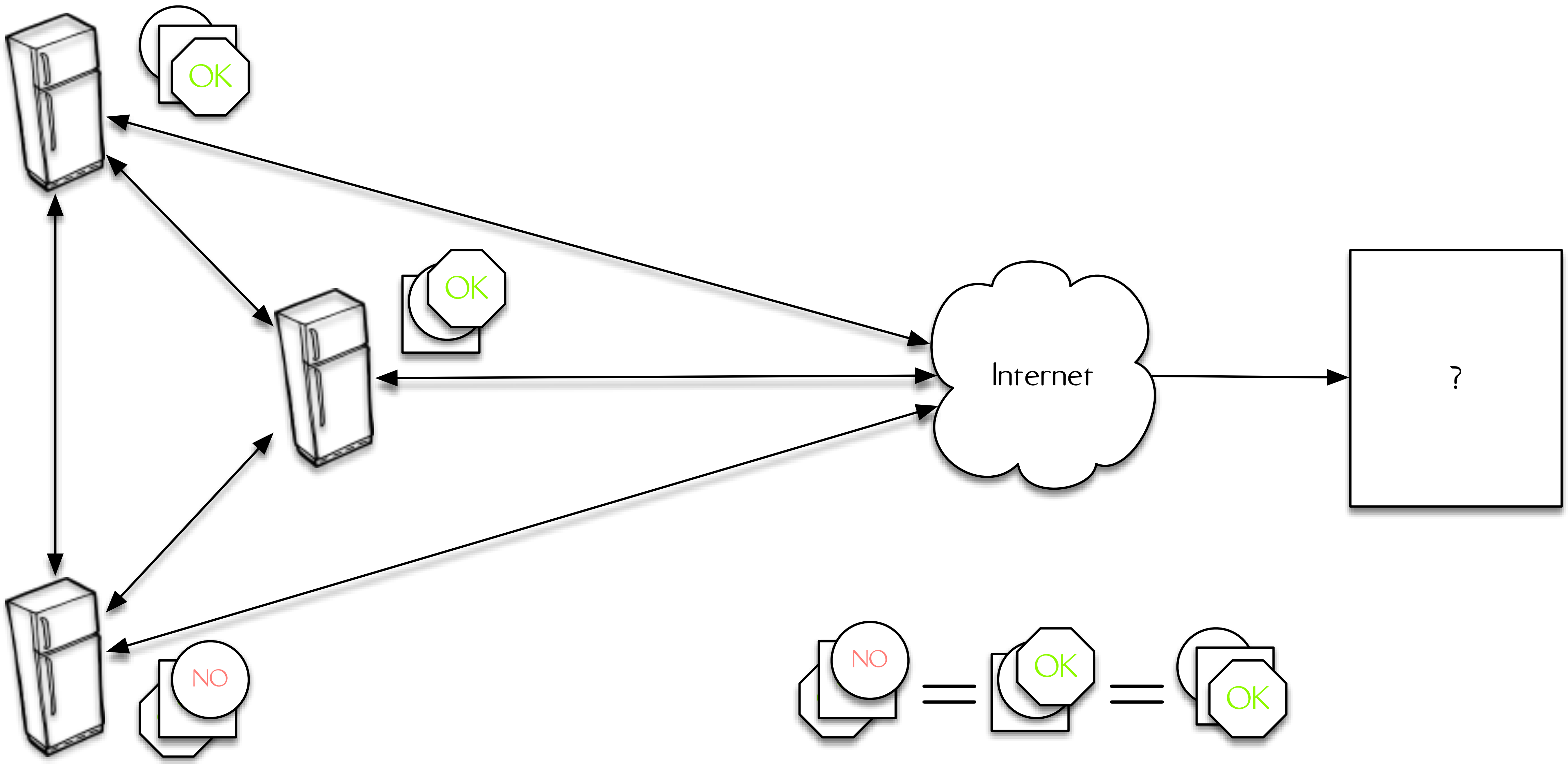

NO
OK
Internet
OK

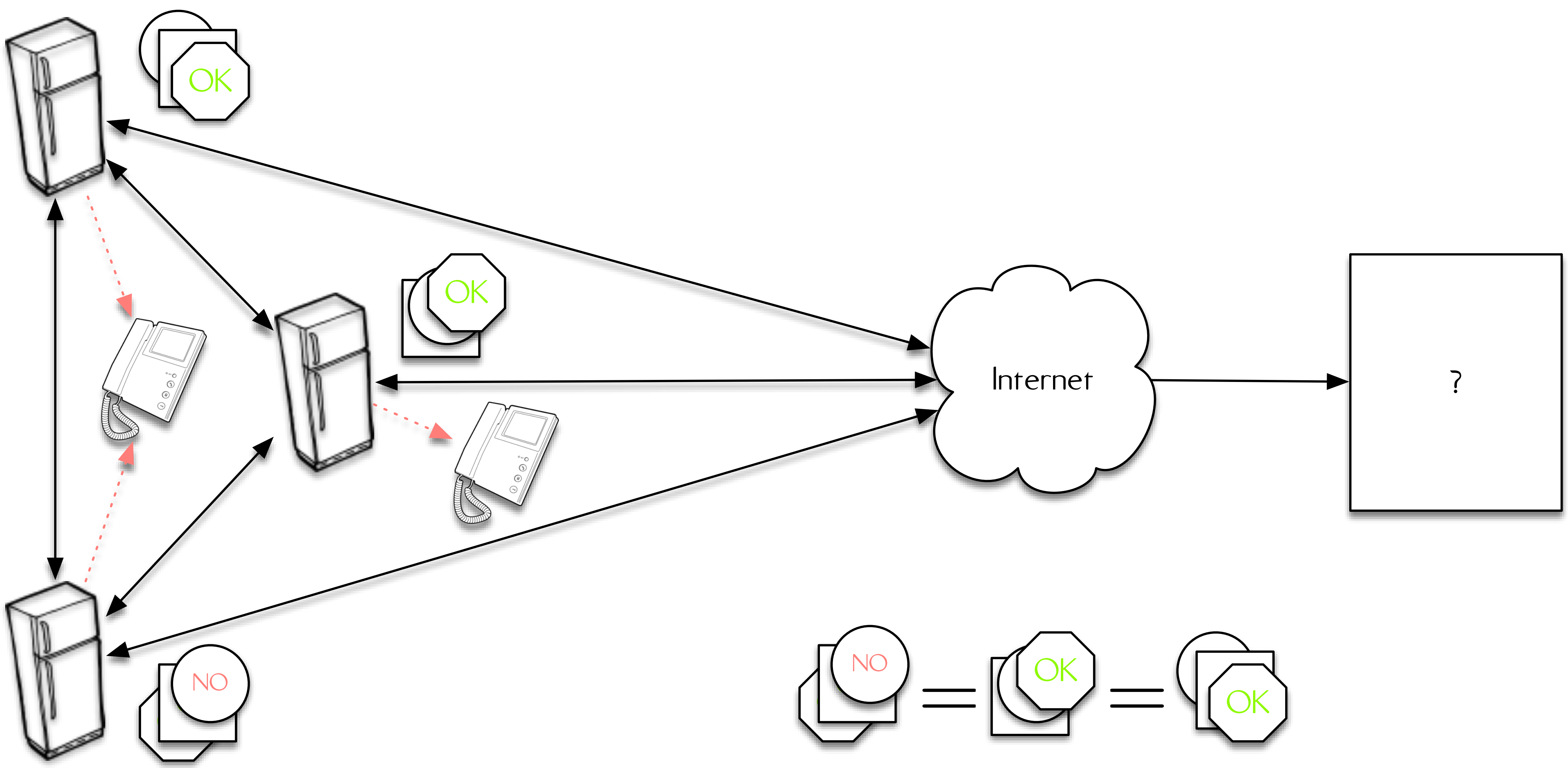


OK
OK
Internet
NO


OK
OK
Internet
NO
OK
HDFS
OK
NO


OK
OK
NO
Internet
HDFS
NO
OK
OK
NO
OK
OK


OK
OK
NO
Internet
?
NO
OK
OK


OK
OK
Internet
?
NO
NO
OK
OK

Local Computation

- **Reduce state transmission**
  Perform some local computation to reduce transmitted state on the wire

Local Computation

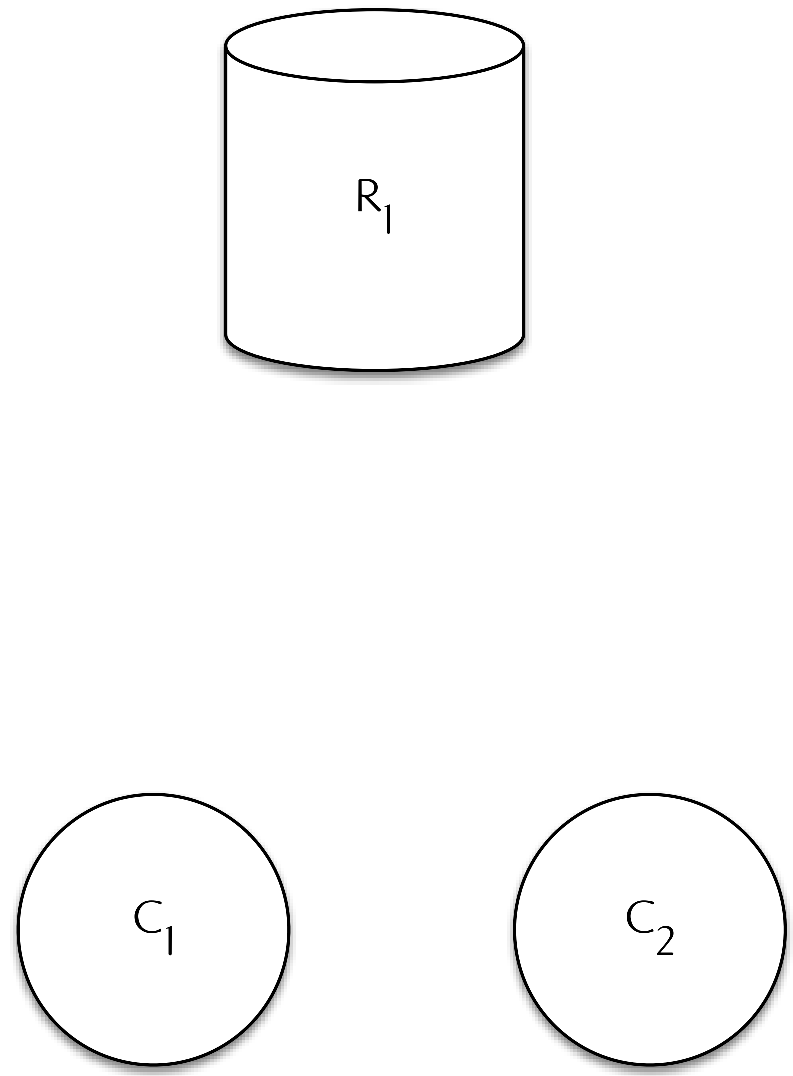
- **Reduce state transmission**
  Perform some local computation to reduce transmitted state on the wire

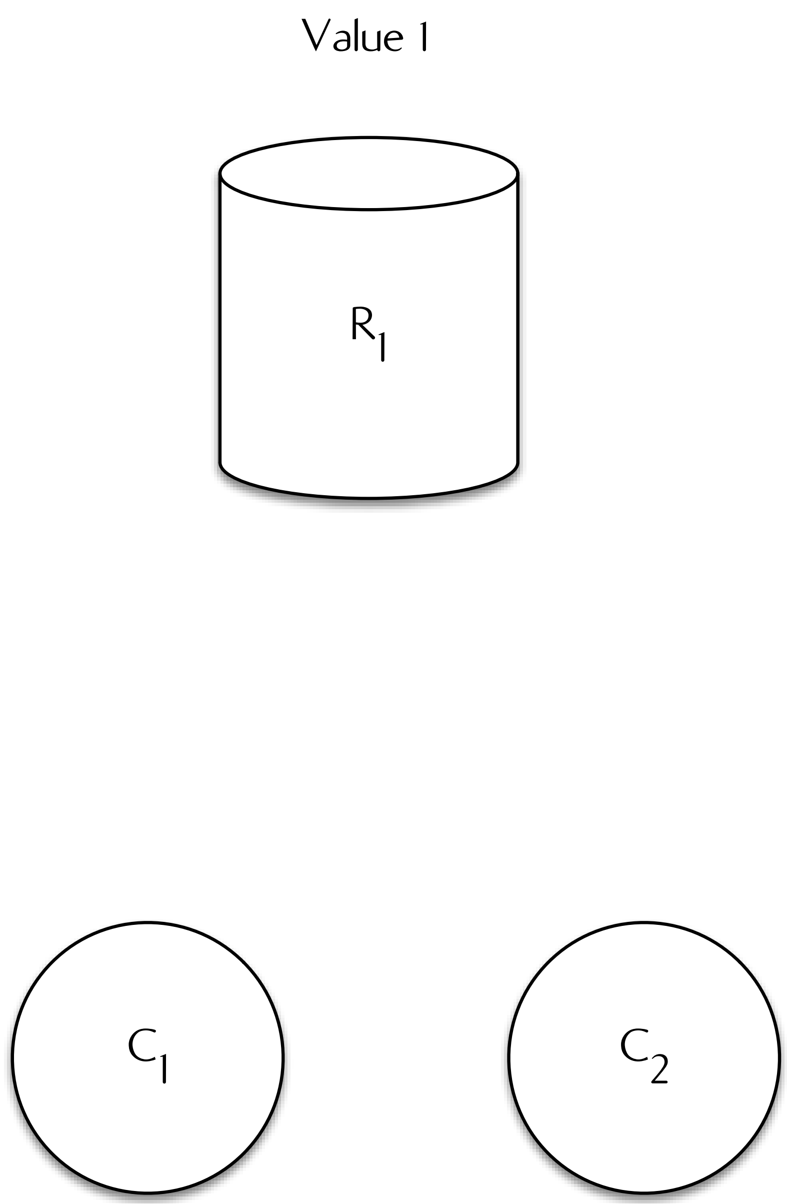
- **Make local decisions**
  Make decisions based on results of local computation

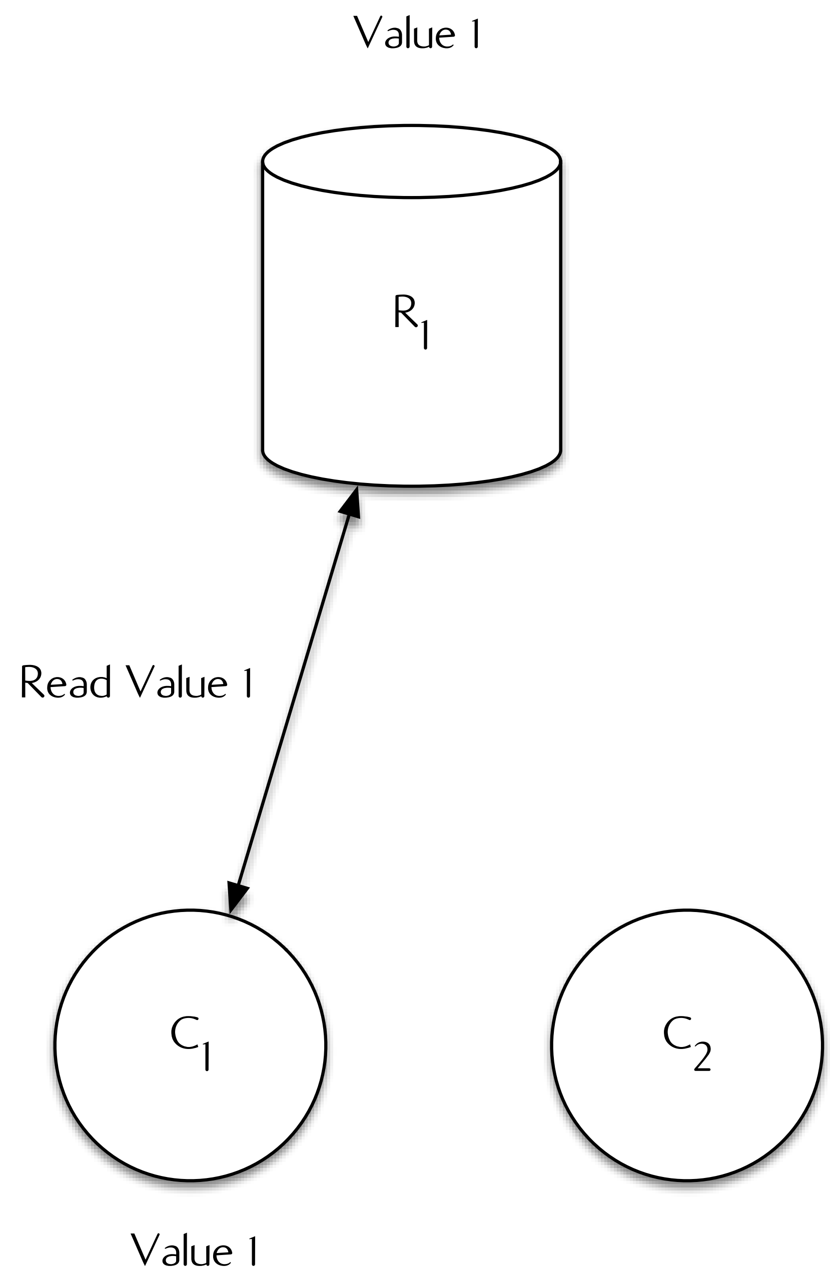
Databases

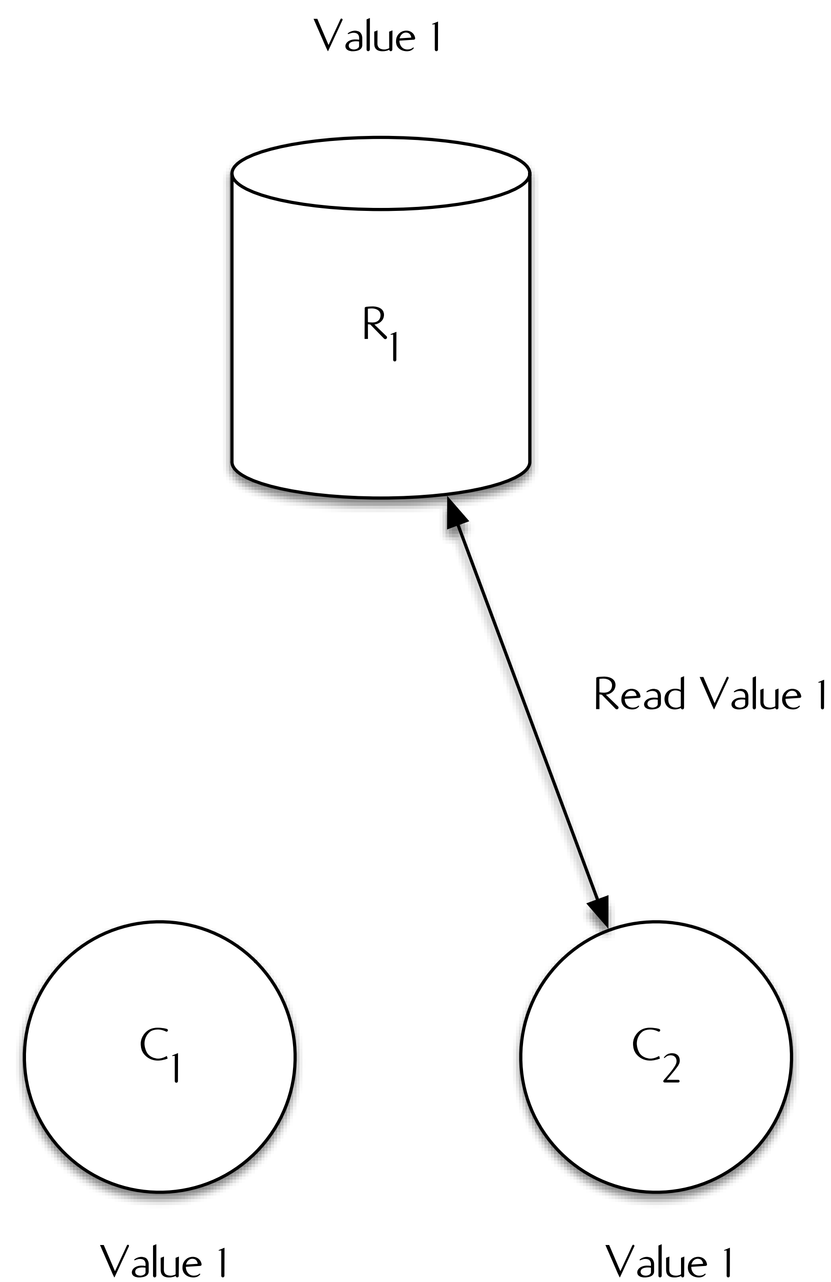
Consistency Models

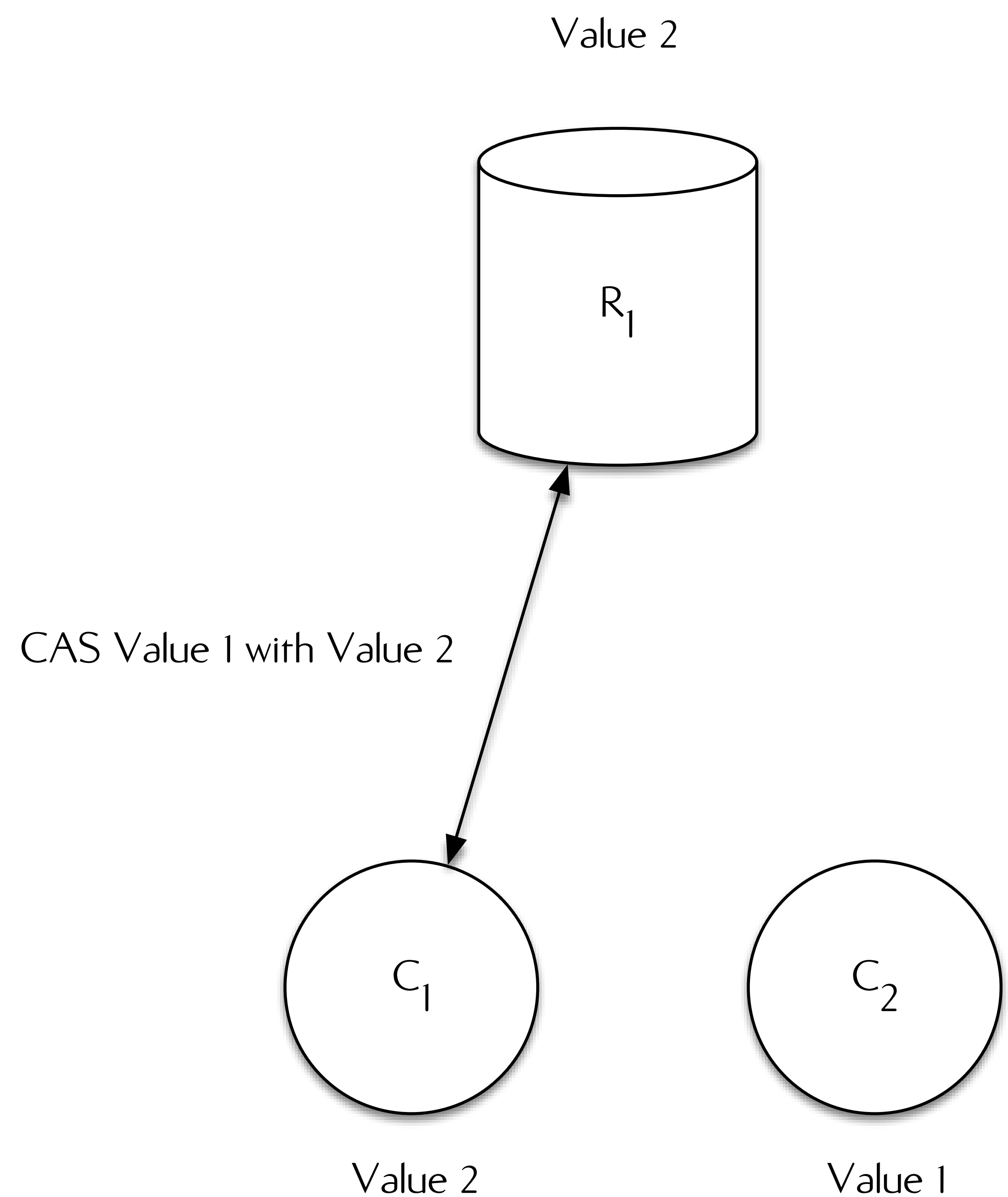
Databases
Strong Consistency

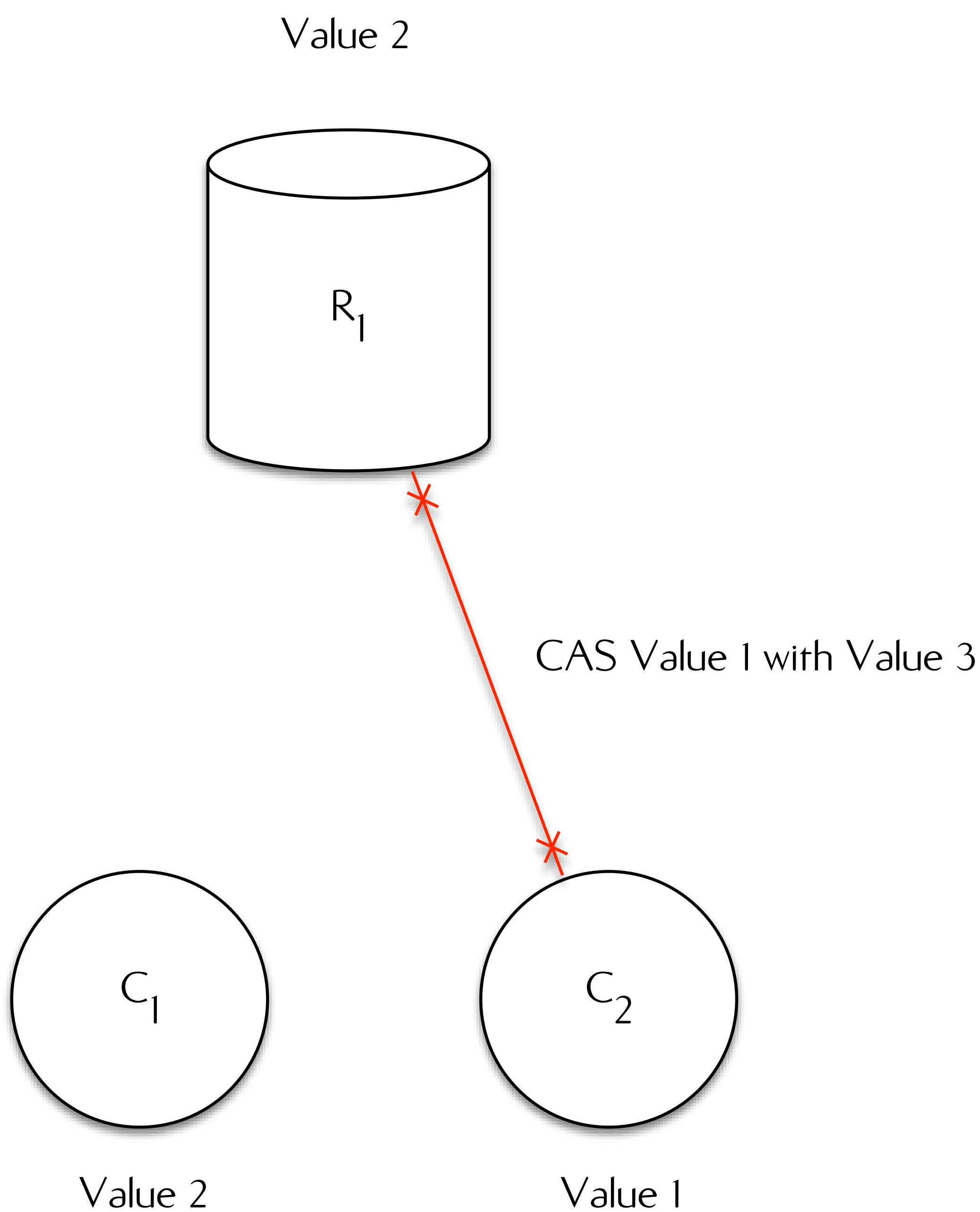

R_1
C_1
C_2


Value 1
R1
C1
C2

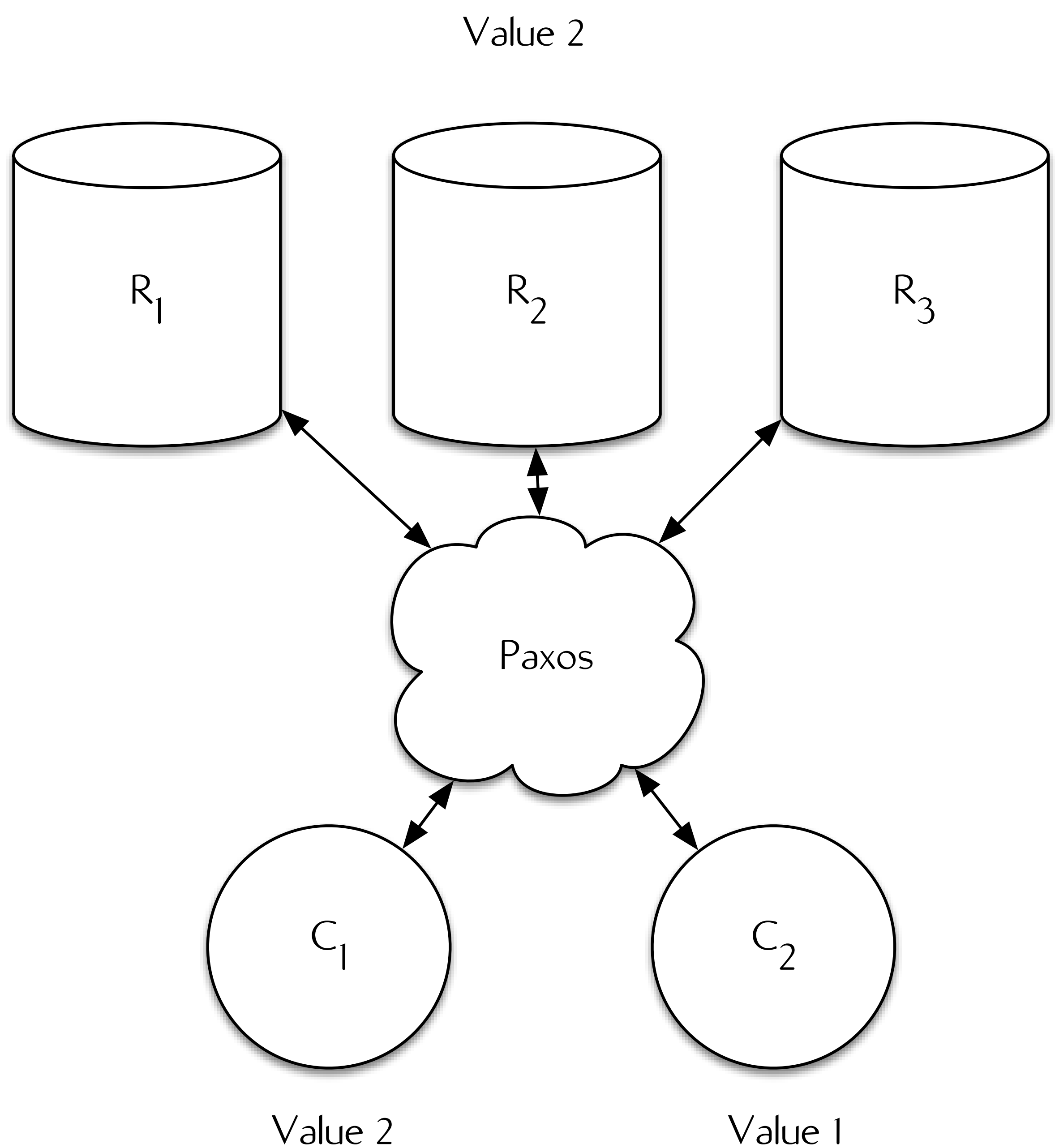

Value 1
R_1
Read Value 1
C_1
C_2
Value 1


Value 1
R1
Read Value 1
C1
C2
Value 1
Value 1

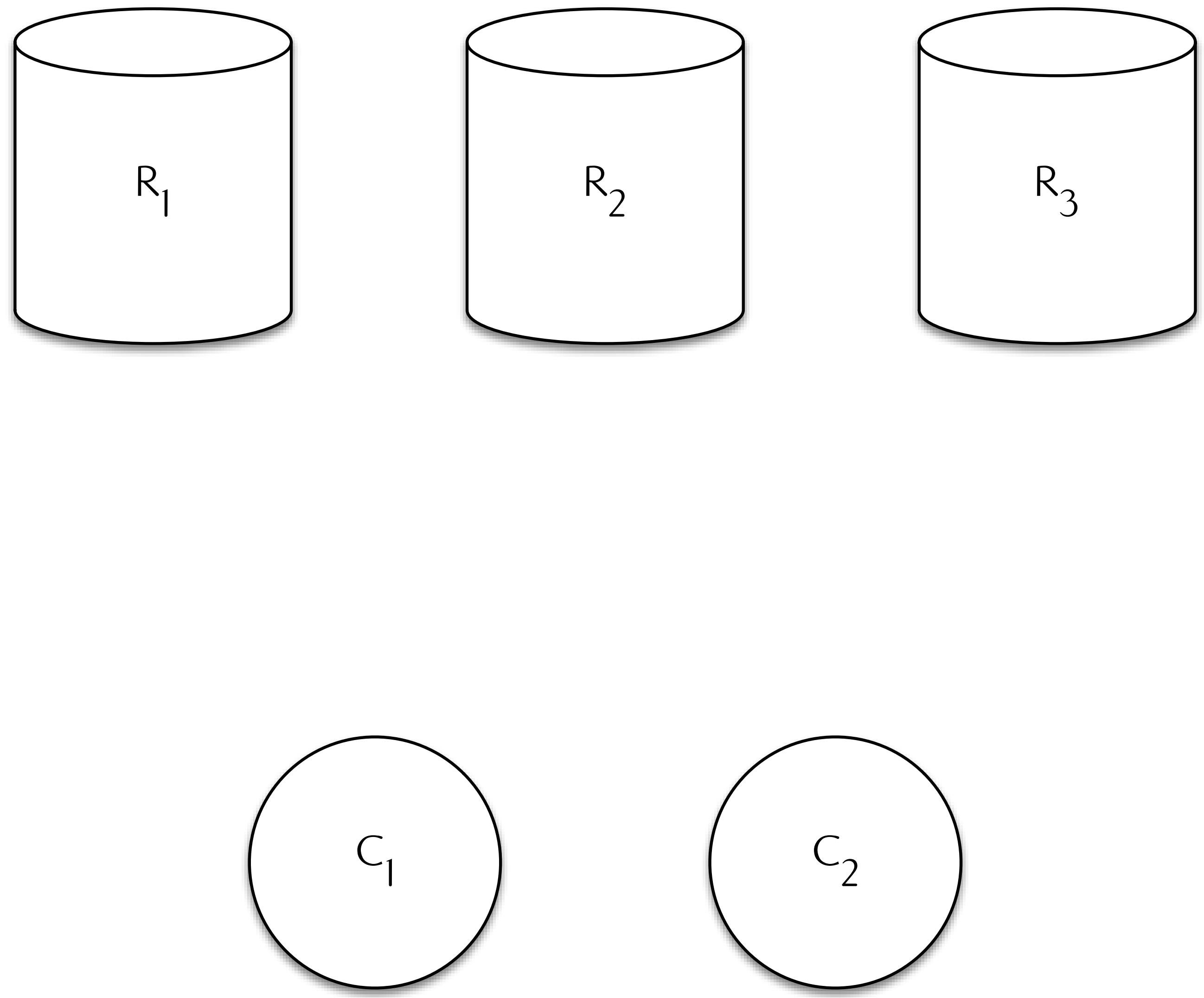

Value 2
R1
CAS Value 1 with Value 2
C1
C2
Value 2
Value 1

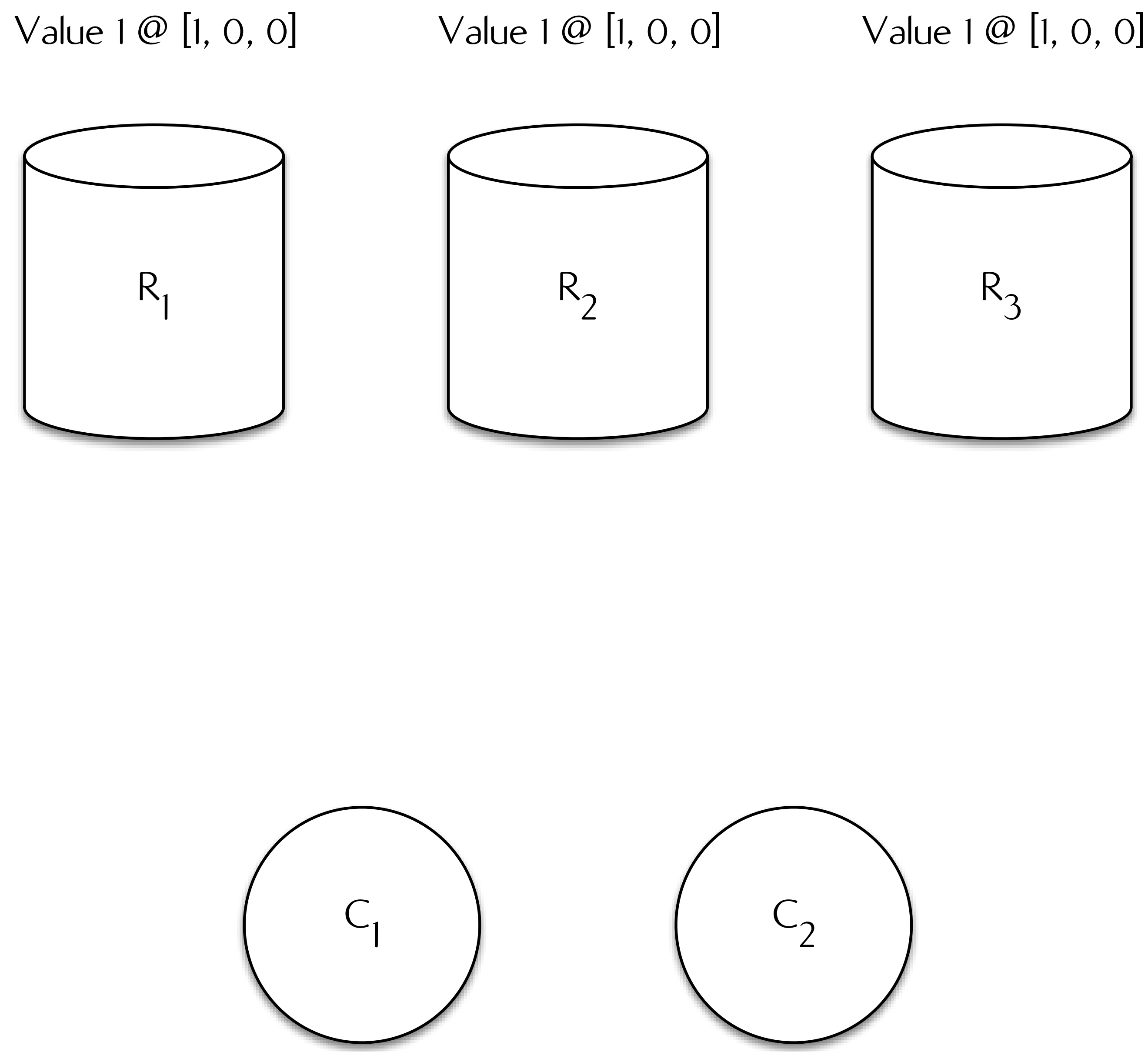

Value 2
R1
CAS Value 1 with Value 3
C1
C2
Value 2
Value 1

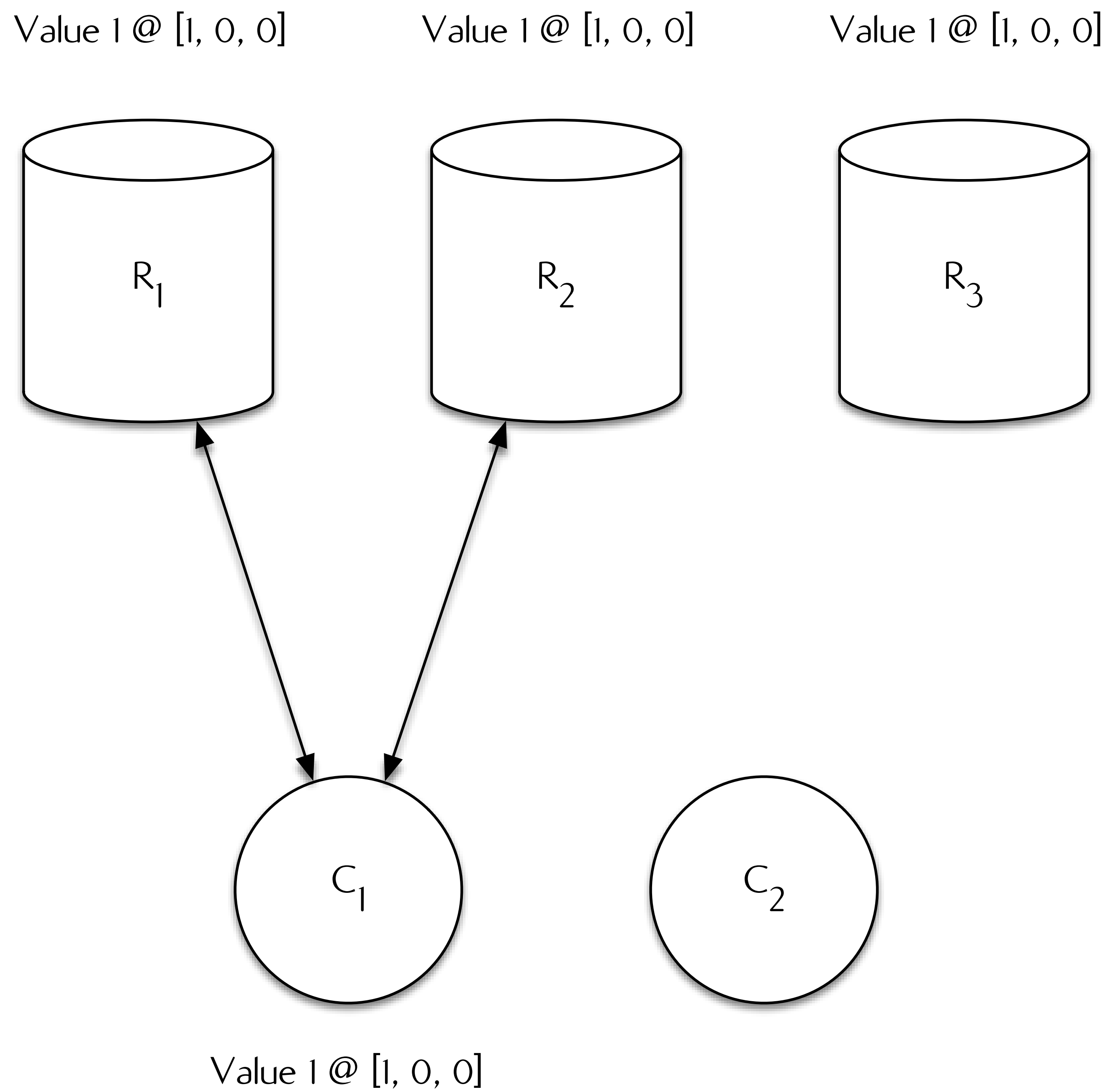
I won't diagram
the **Paxos** protocol

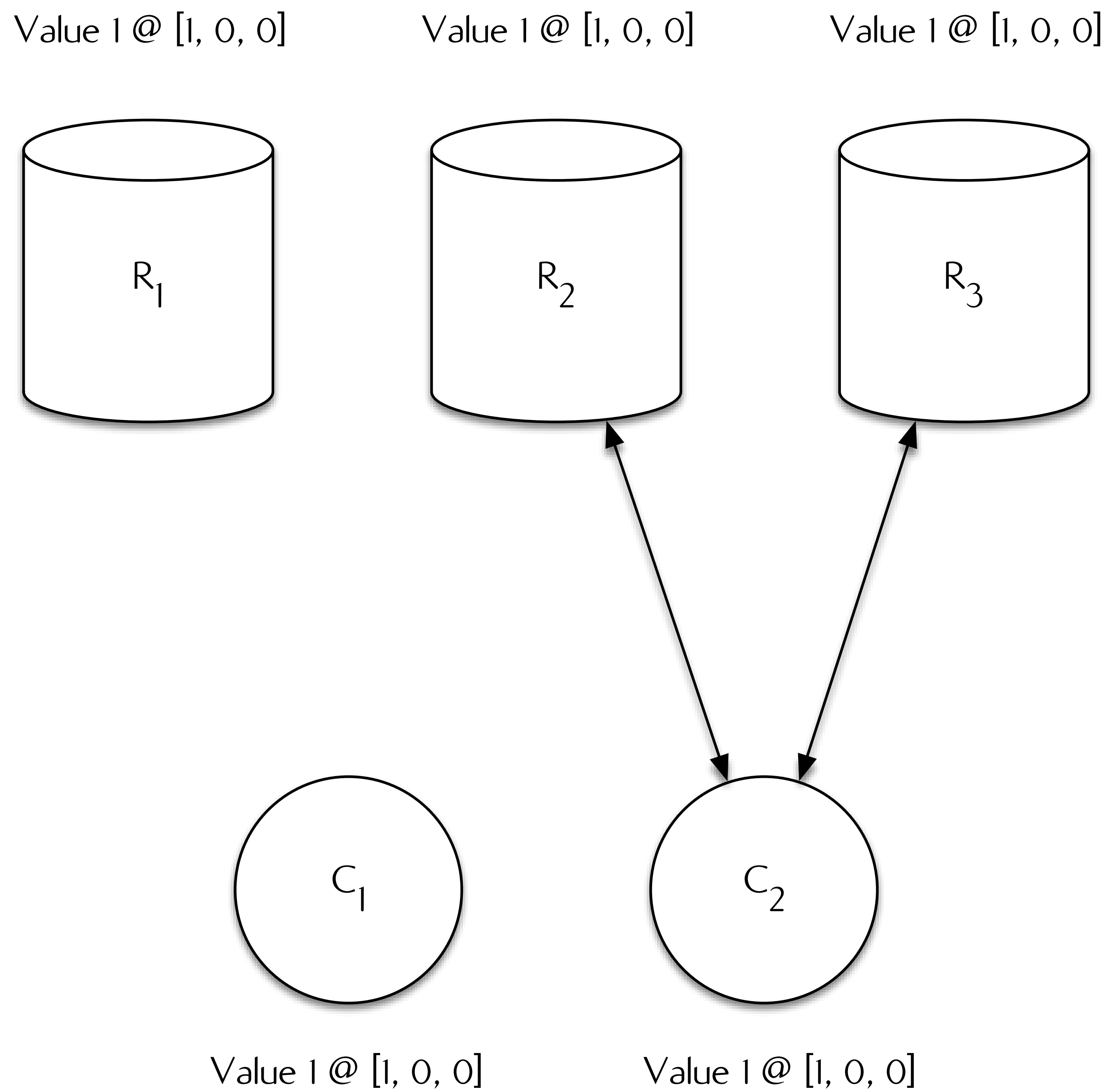

Value 2
R_1
R_2
R_3
Paxos
C_1
C_2
Value 2
Value 1

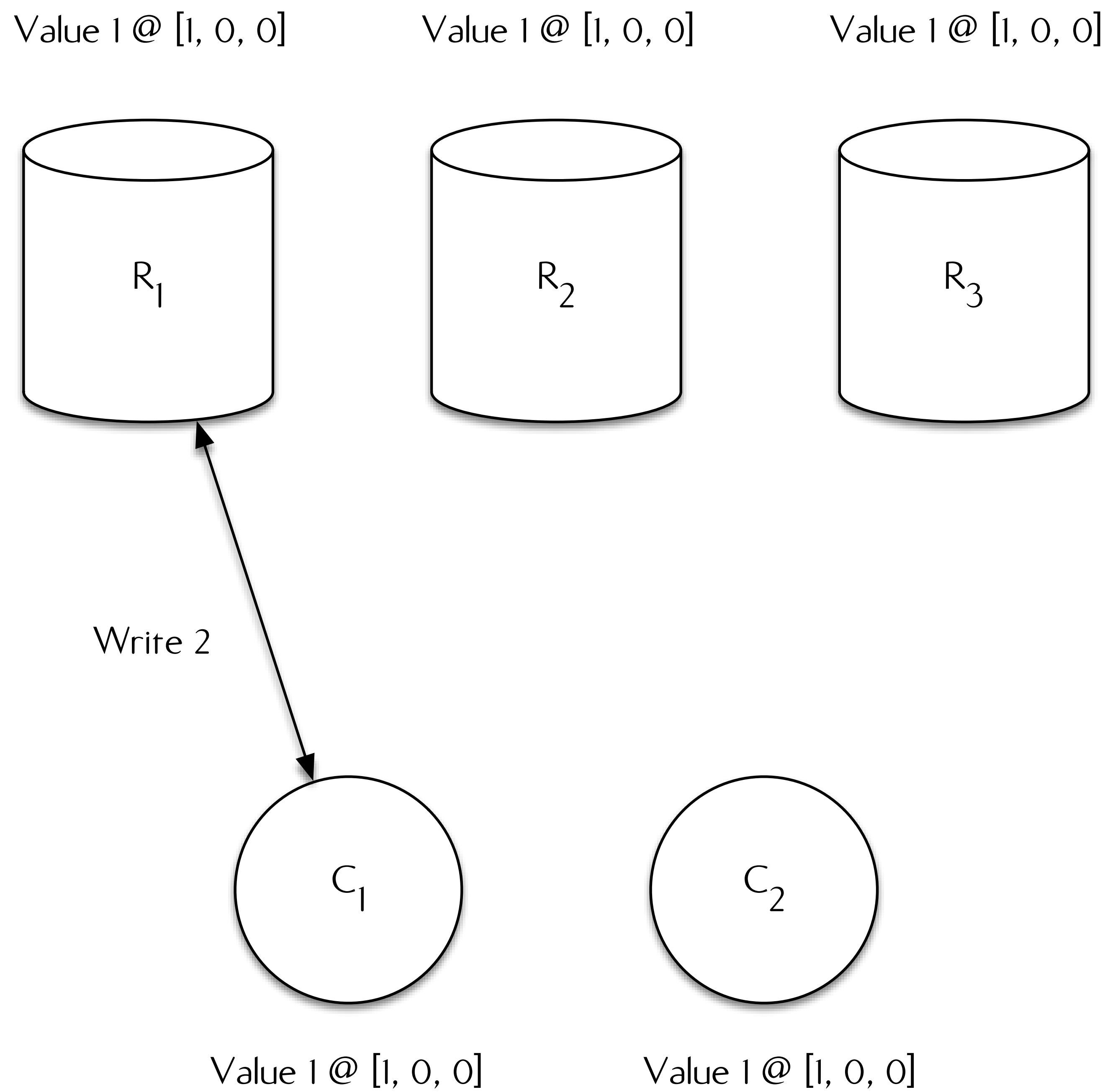
Databases
Eventual Consistency

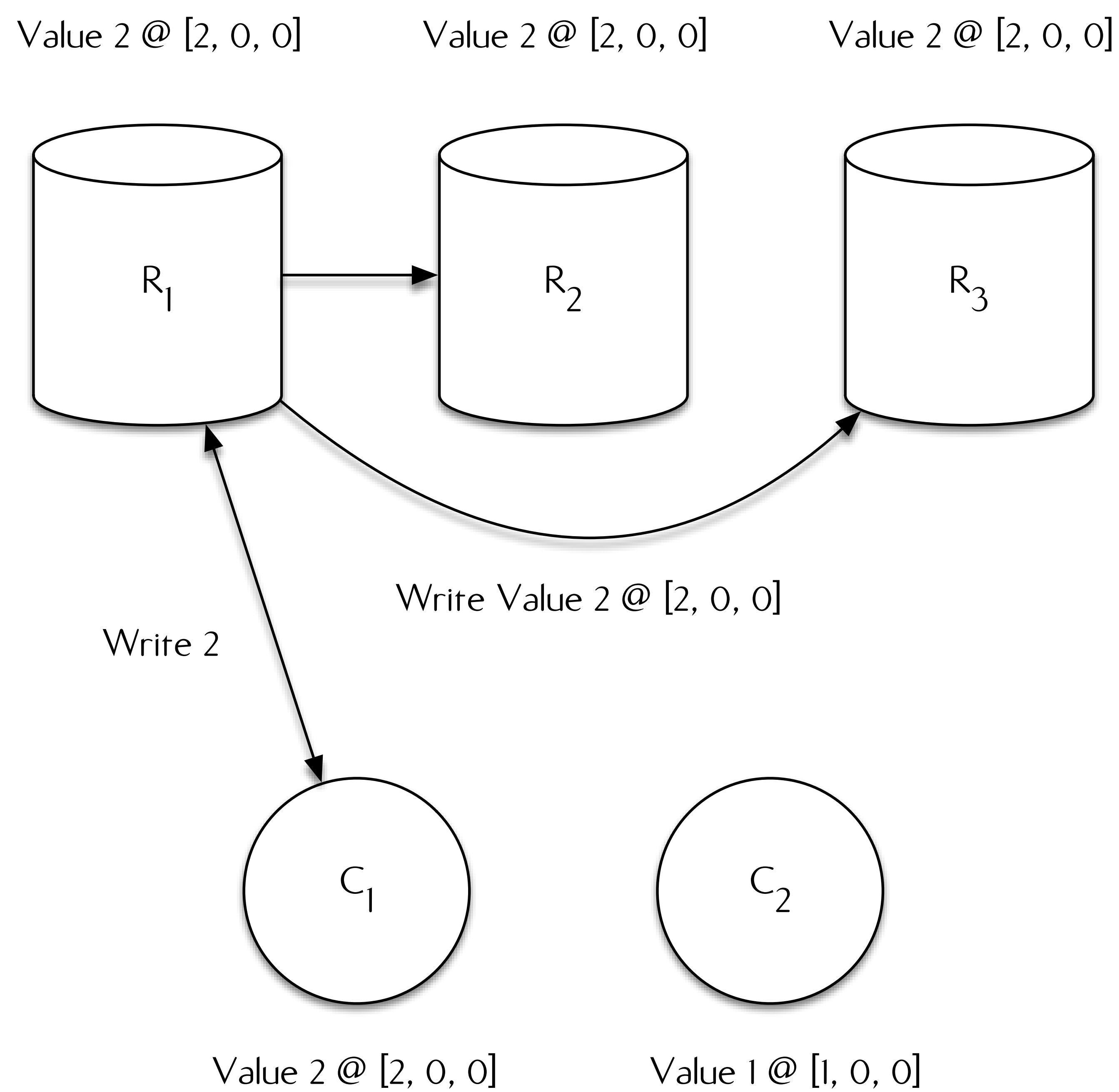

R_1
R_2
R_3
C_1
C_2

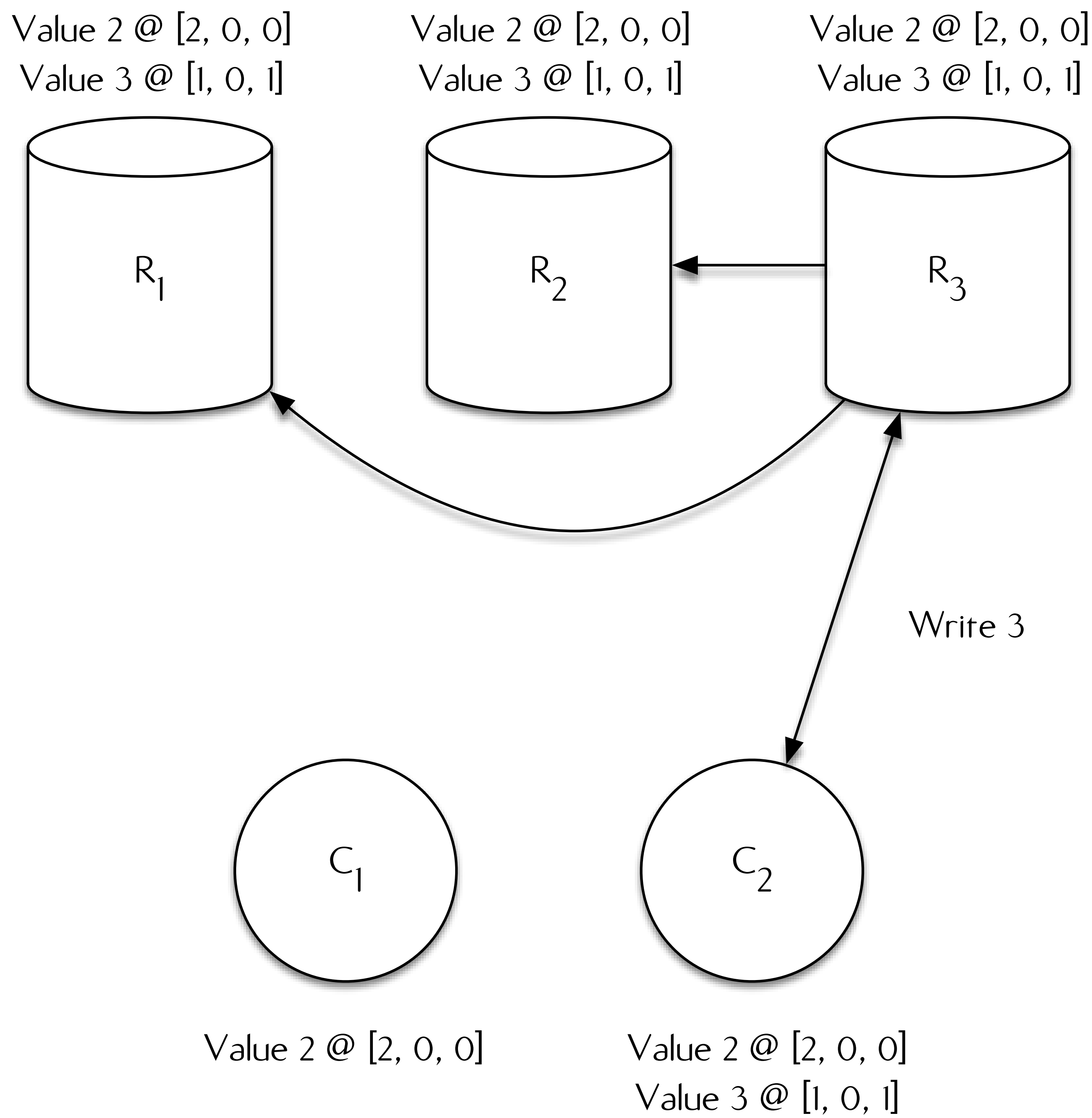

Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]
R1
R2
R3
C1
C2


Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]
R1
R2
R3
C1
C2
Value 1 @ [1, 0, 0]

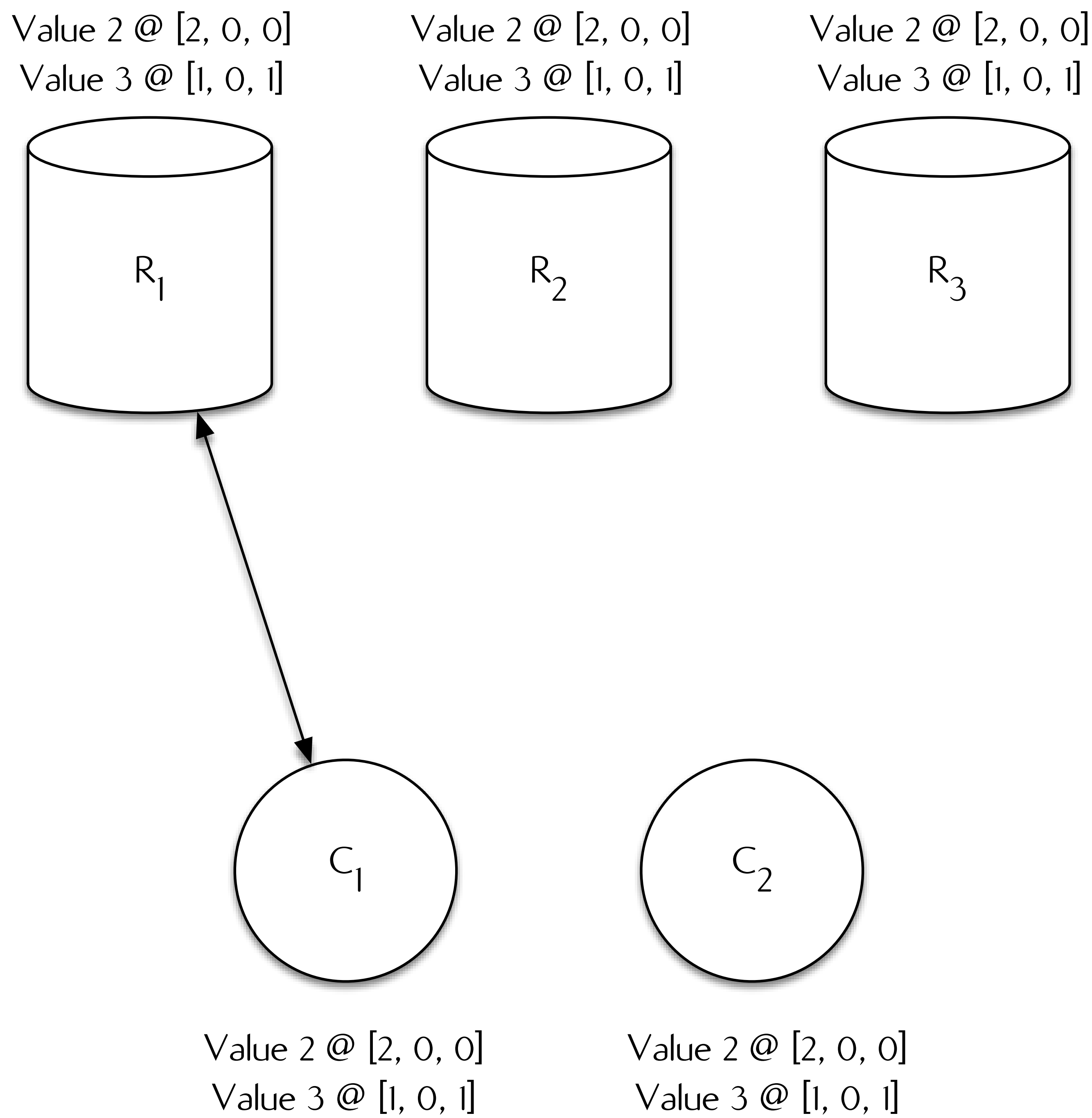

Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]
R1
R2
R3
C1
C2
Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]

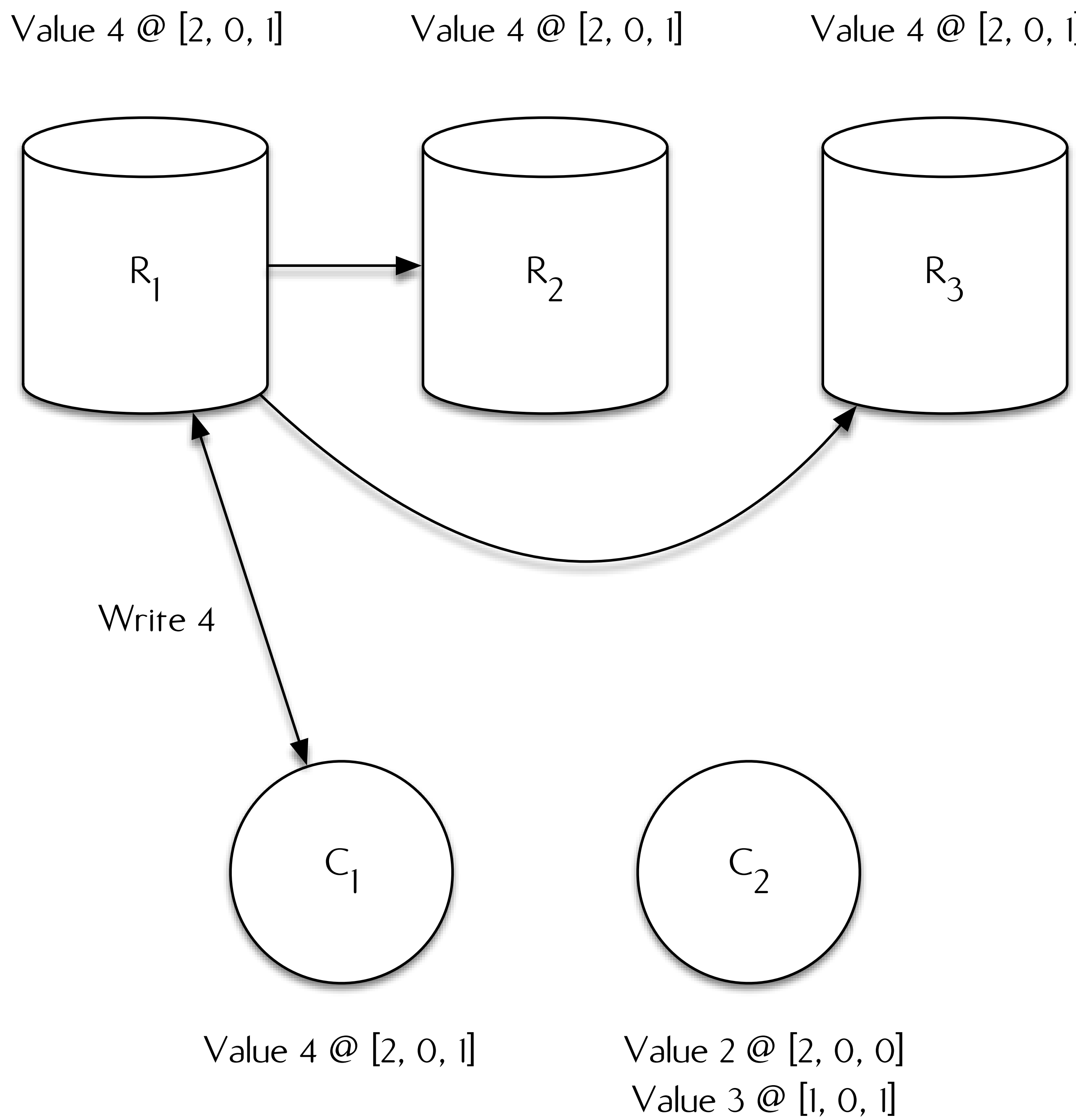

Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]
R1
R2
R3
Write 2
C1
C2
Value 1 @ [1, 0, 0]
Value 1 @ [1, 0, 0]


Value 2 @ [2, 0, 0]
Value 2 @ [2, 0, 0]
Value 2 @ [2, 0, 0]
R1
R2
R3
Write Value 2 @ [2, 0, 0]
Write 2
C1
C2
Value 2 @ [2, 0, 0]
Value 1 @ [1, 0, 0]



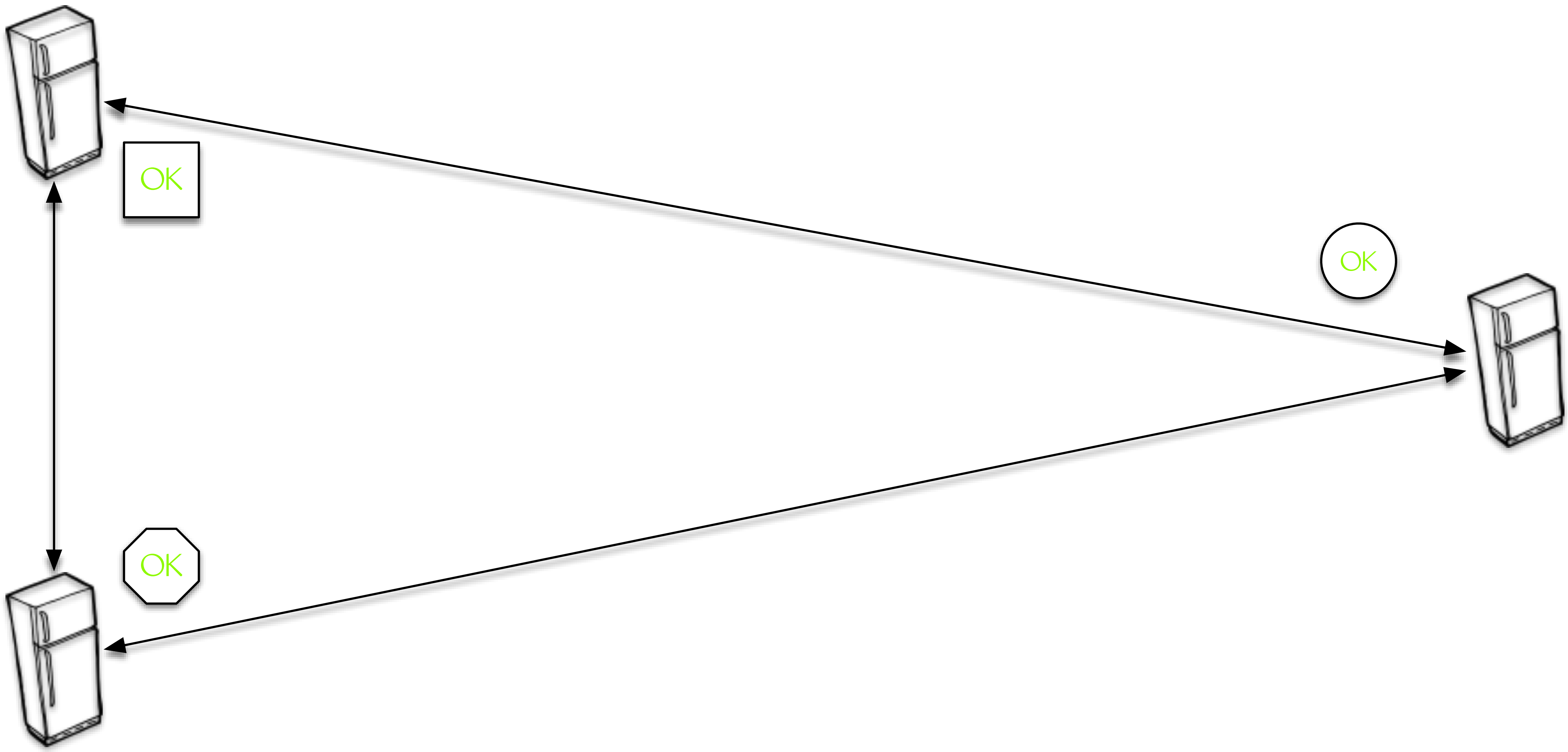
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]
R1
R2
R3
Write 3
C1
C2
Value 2 @ [2, 0, 0]
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]

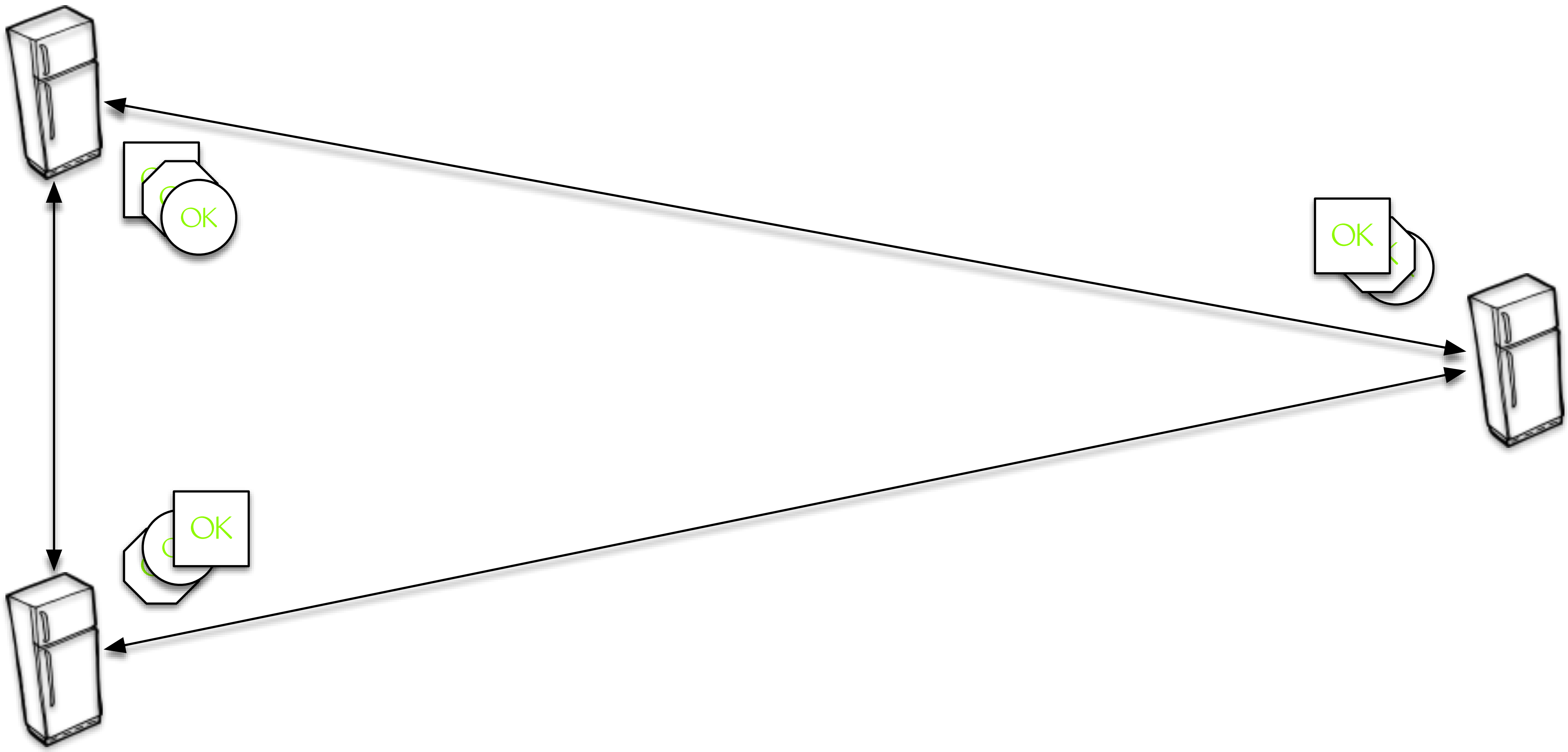

Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]
R1
R2
R3
C1
C2
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]

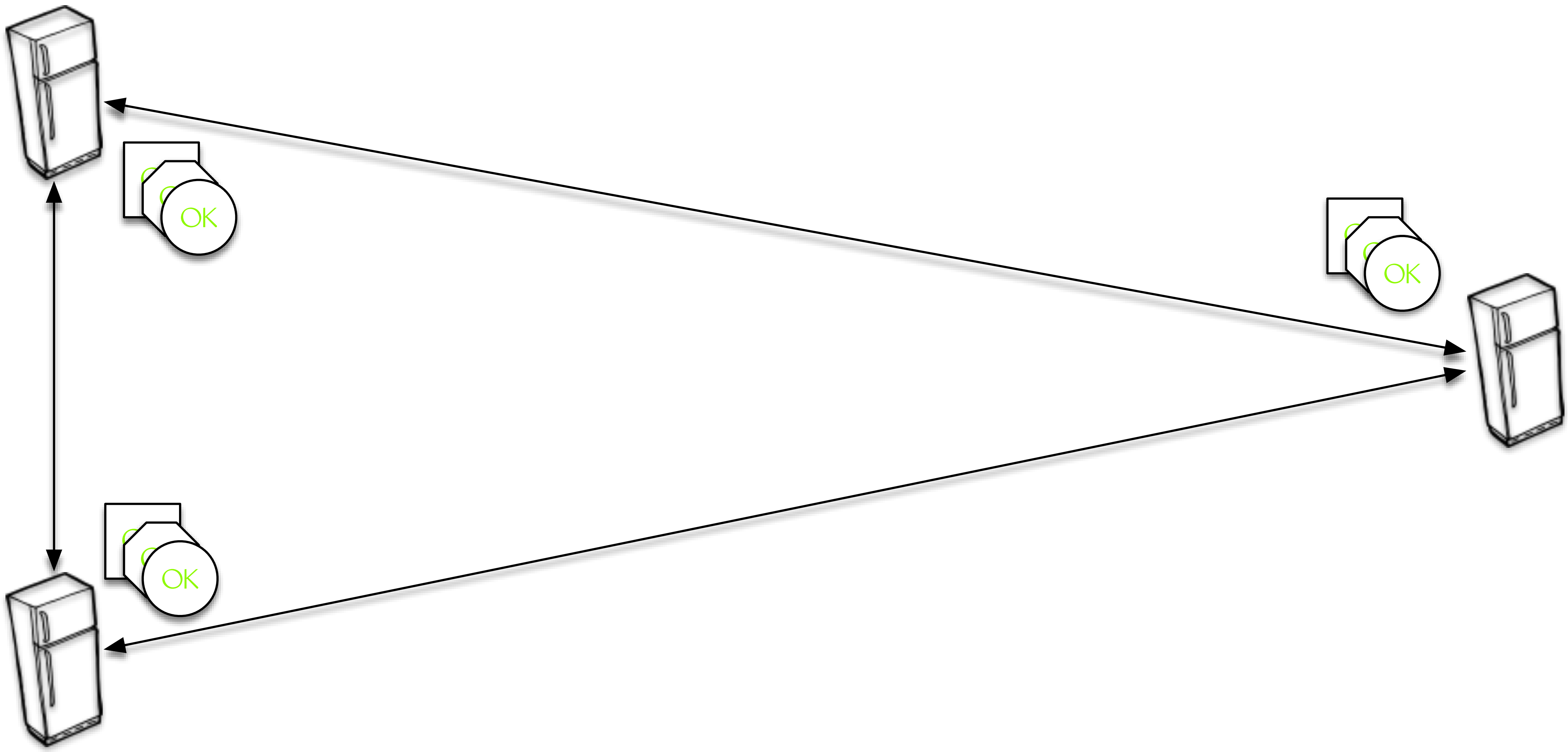

Value 4 @ [2, 0, 1]
Value 4 @ [2, 0, 1]
Value 4 @ [2, 0, 1]
R1
R2
R3
Write 4
C1
C2
Value 4 @ [2, 0, 1]
Value 2 @ [2, 0, 0]
Value 3 @ [1, 0, 1]

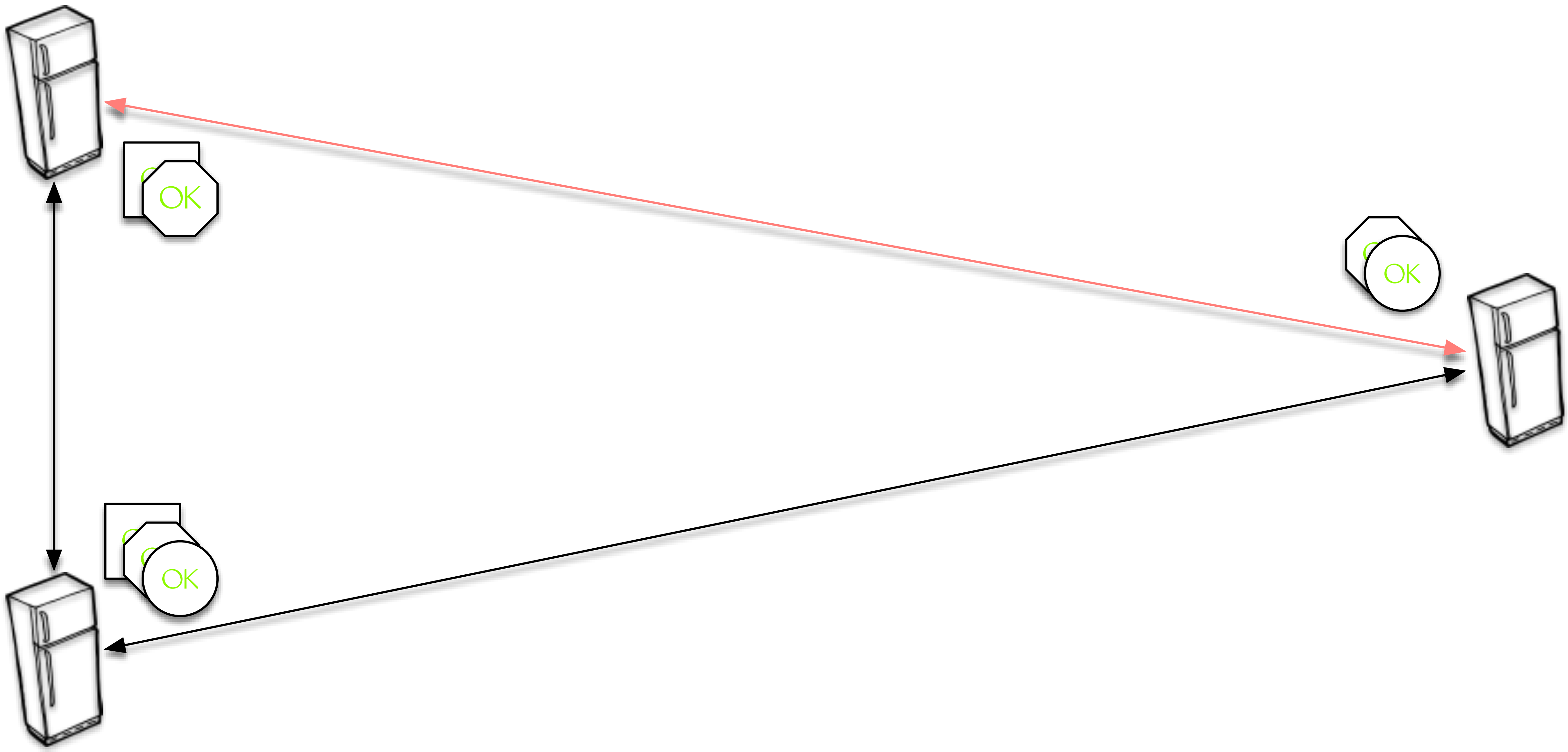
Eventual Consistency
As The Model

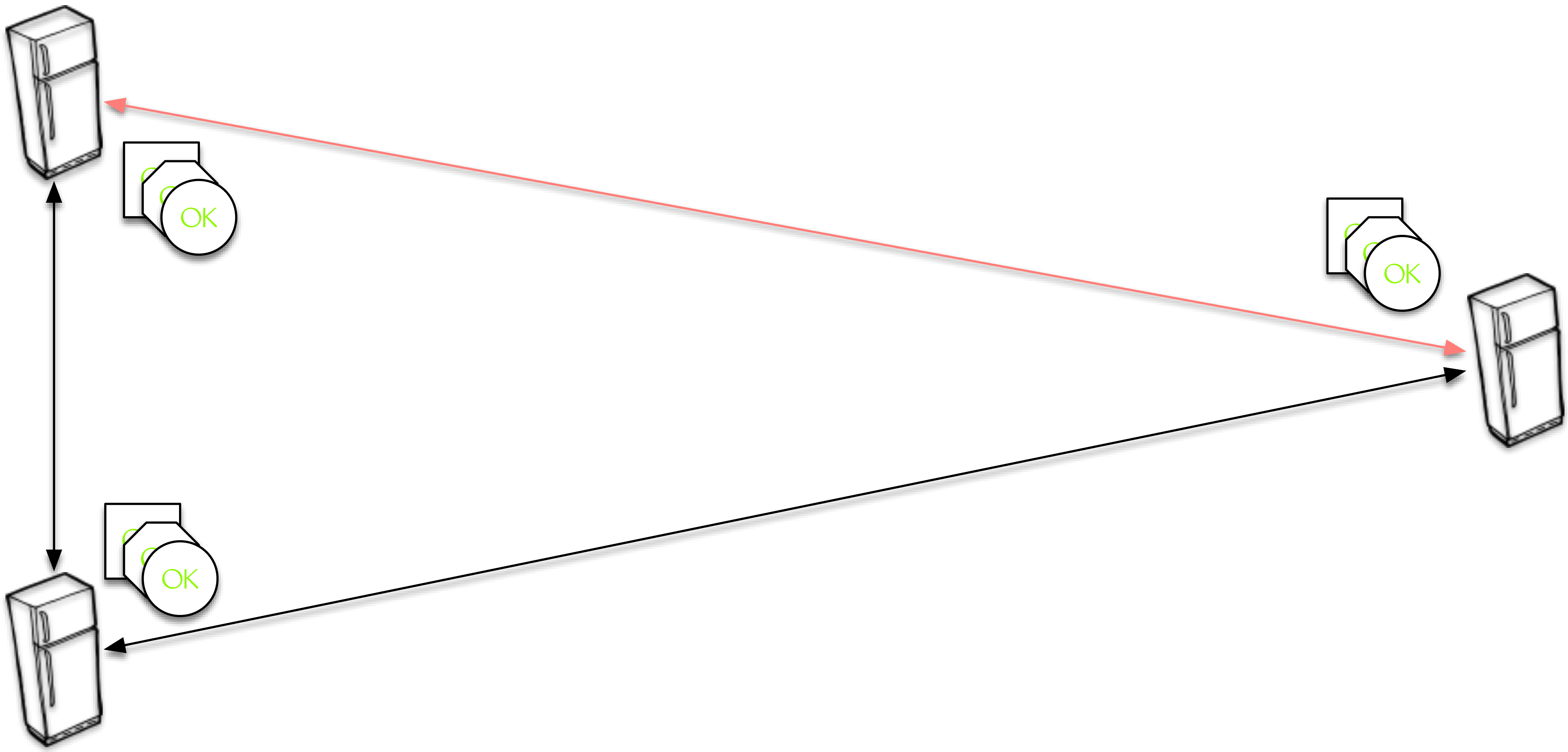
Clients
Own Their Data


OK
OK
OK

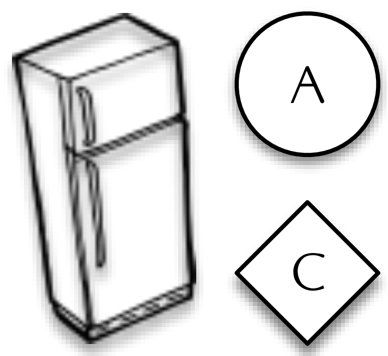

OK
OK
OK

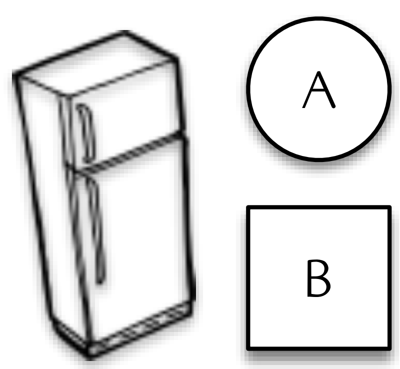

OK
OK
OK

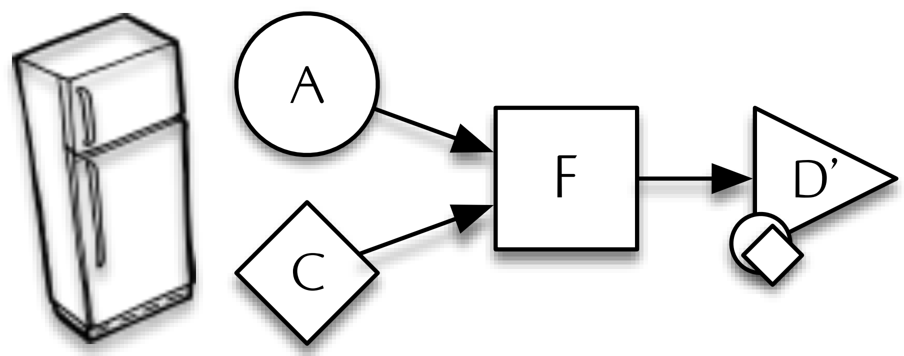

OK
OK
OK

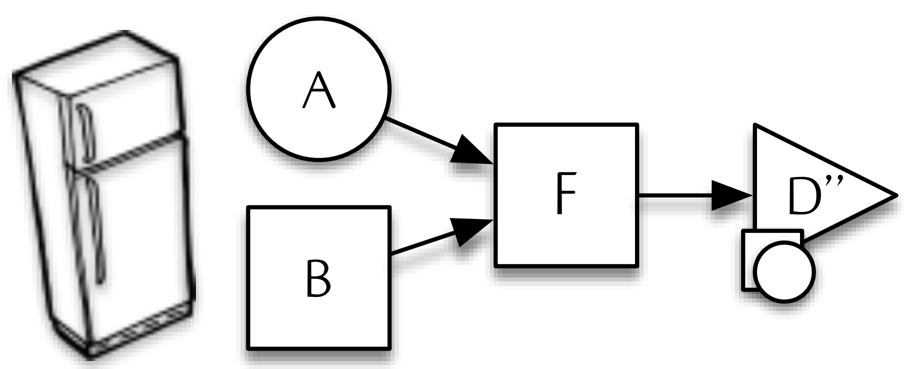

OK
OK
OK

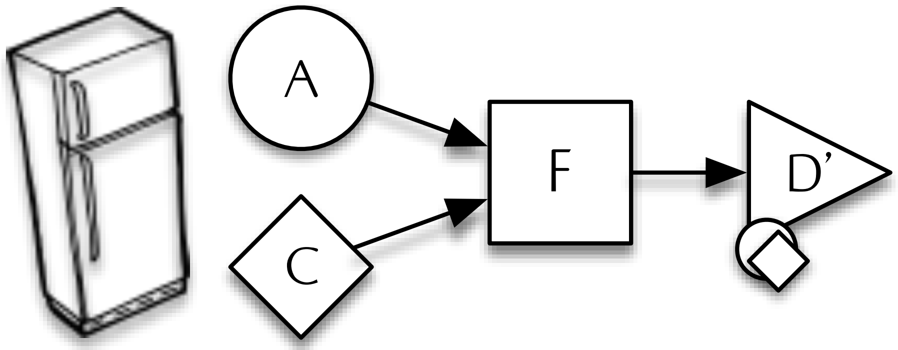
Computations
Mergability & Provenance

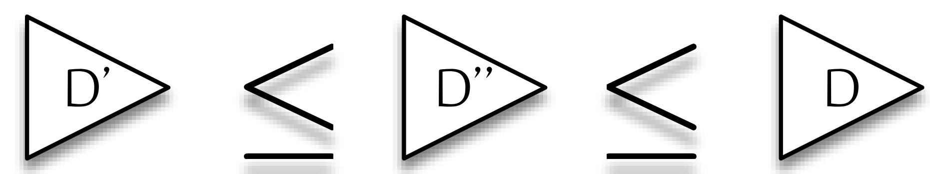

A
C

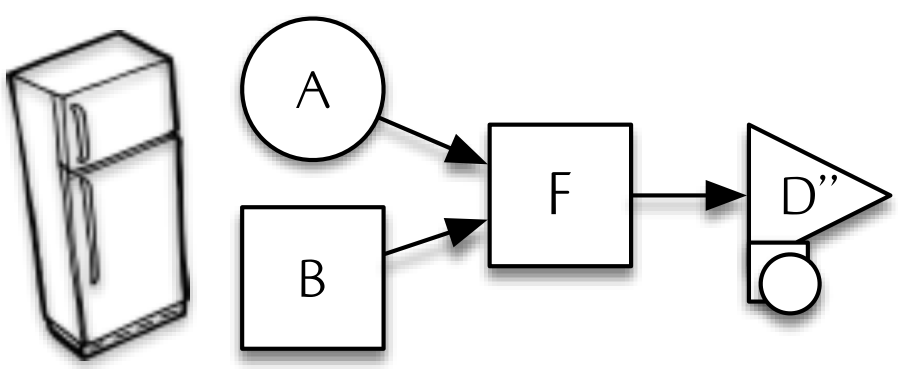

A
B

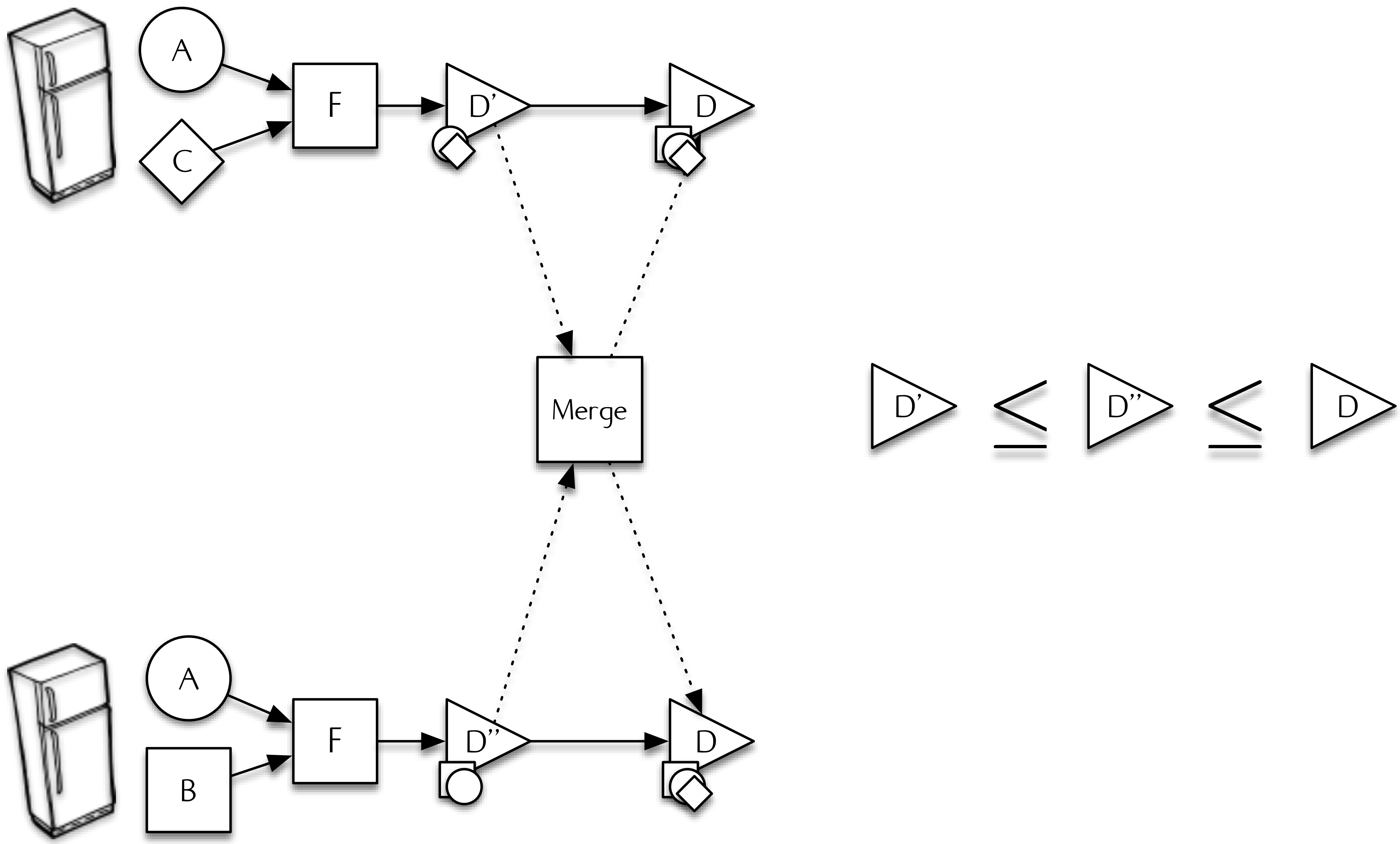

A
C
F
D’


A
B
F
D’’


A
C
F
D’


D’
≤
D’’
≤
D


A
B
F
D’’


A
C
F
D'
D
Merge
A
B
F
D''
D
D' ≤ D'' ≤ D

Example Application

Preliminary Results

Preliminary Results

- Conflict-Free Replicated Data Types
  Distributed data structures designed for convergence
  [Shapiro *et al.*, 2011]

Preliminary Results

- **Conflict-Free Replicated Data Types**
  Distributed data structures designed for convergence
  [Shapiro *et al.*, 2011]

- **Lattice Processing**
  Make decisions based on results of local computation
  [Meiklejohn & Van Roy, 2015]

Preliminary Results

- Conflict-Free Replicated Data Types
  Distributed data structures designed for convergence
  [Shapiro *et al.*, 2011]

- Lattice Processing
  Make decisions based on results of local computation
  [Meiklejohn & Van Roy, 2015]

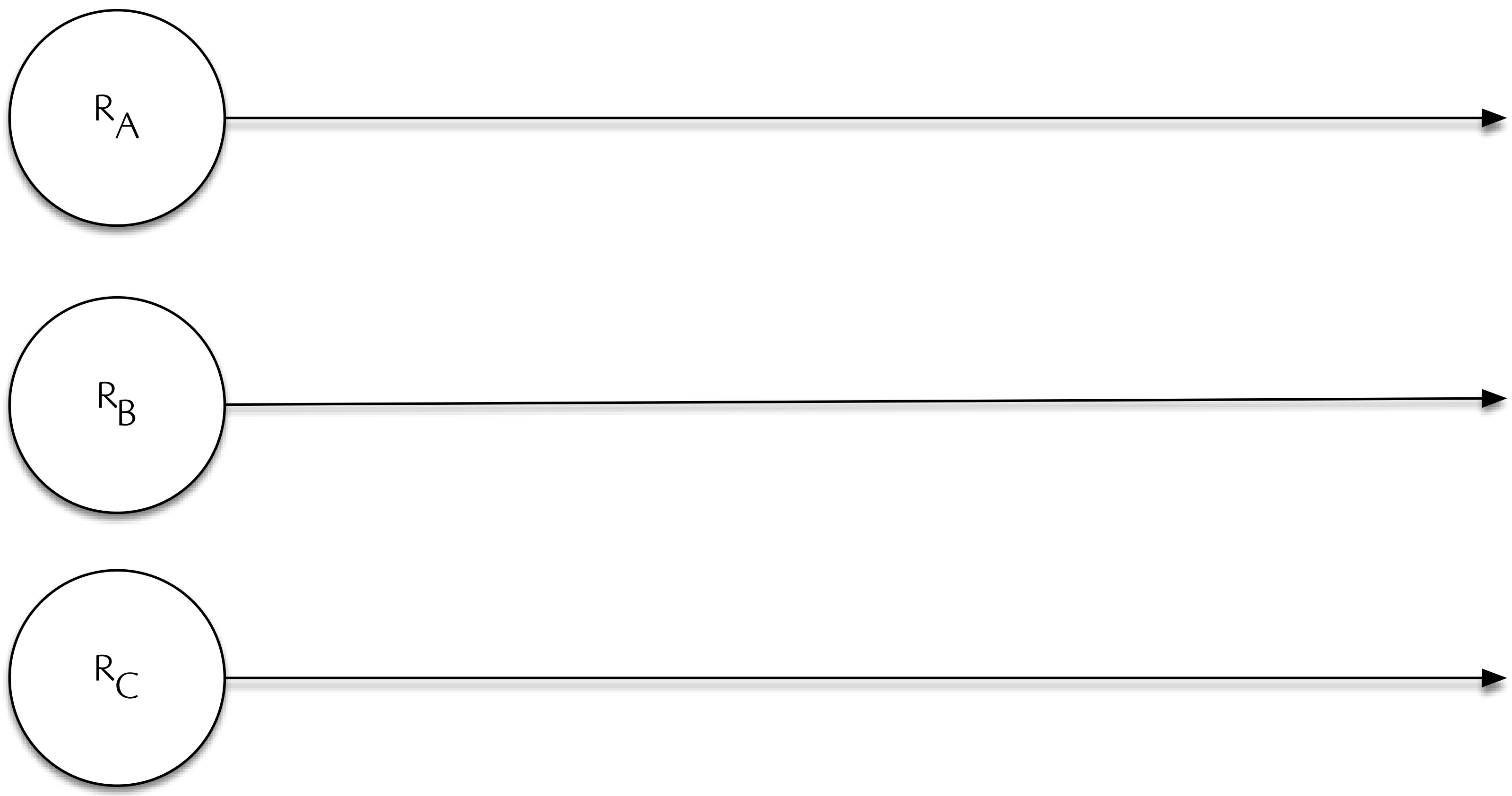
- Selective Hearing
  Epidemic broadcast based runtime system
  [Meiklejohn & Van Roy, 2015/2016]

Conflict-Free
Replicated Data Types

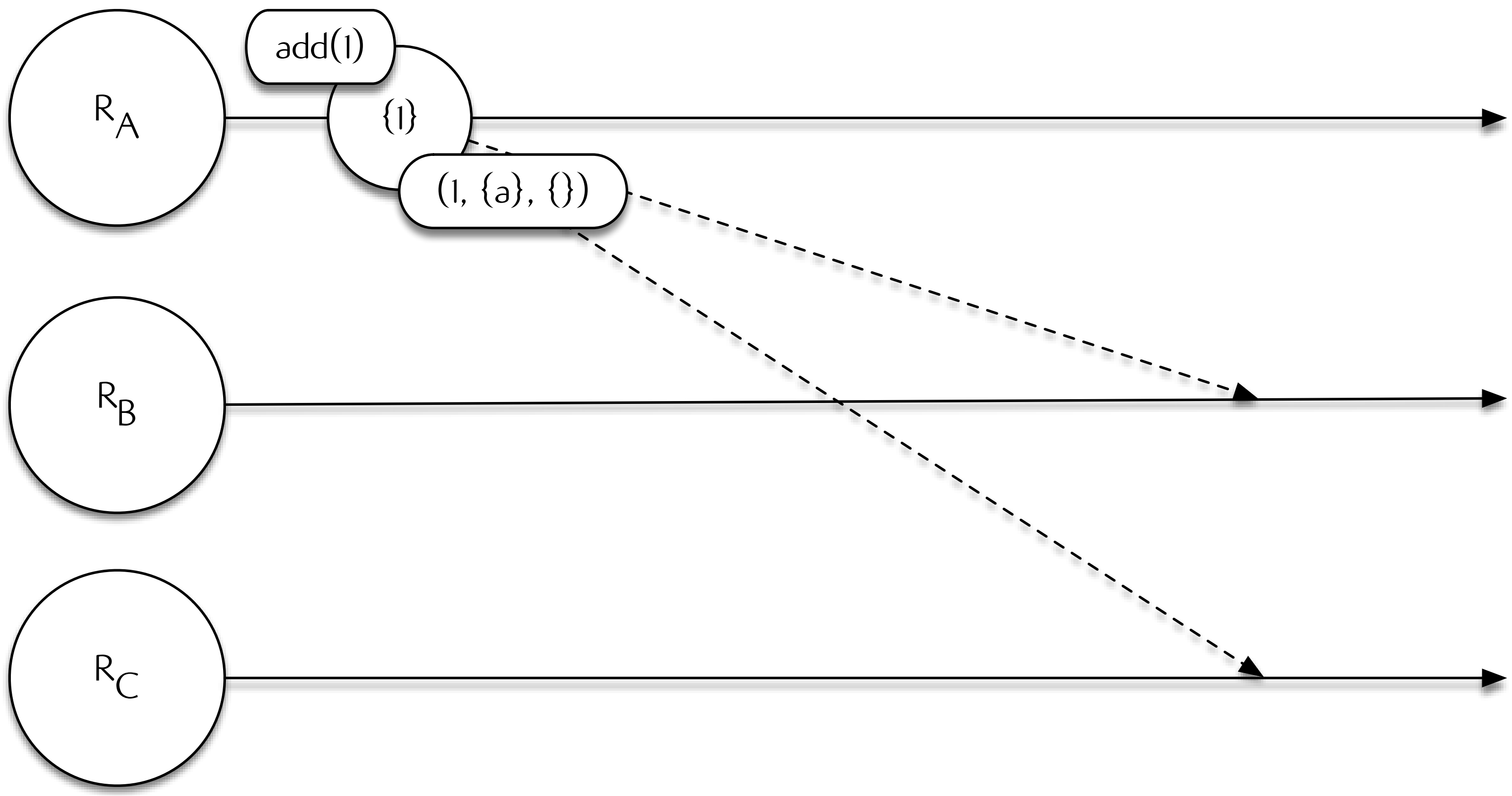
- Collection of types
  Sets, counters, registers, flags, maps

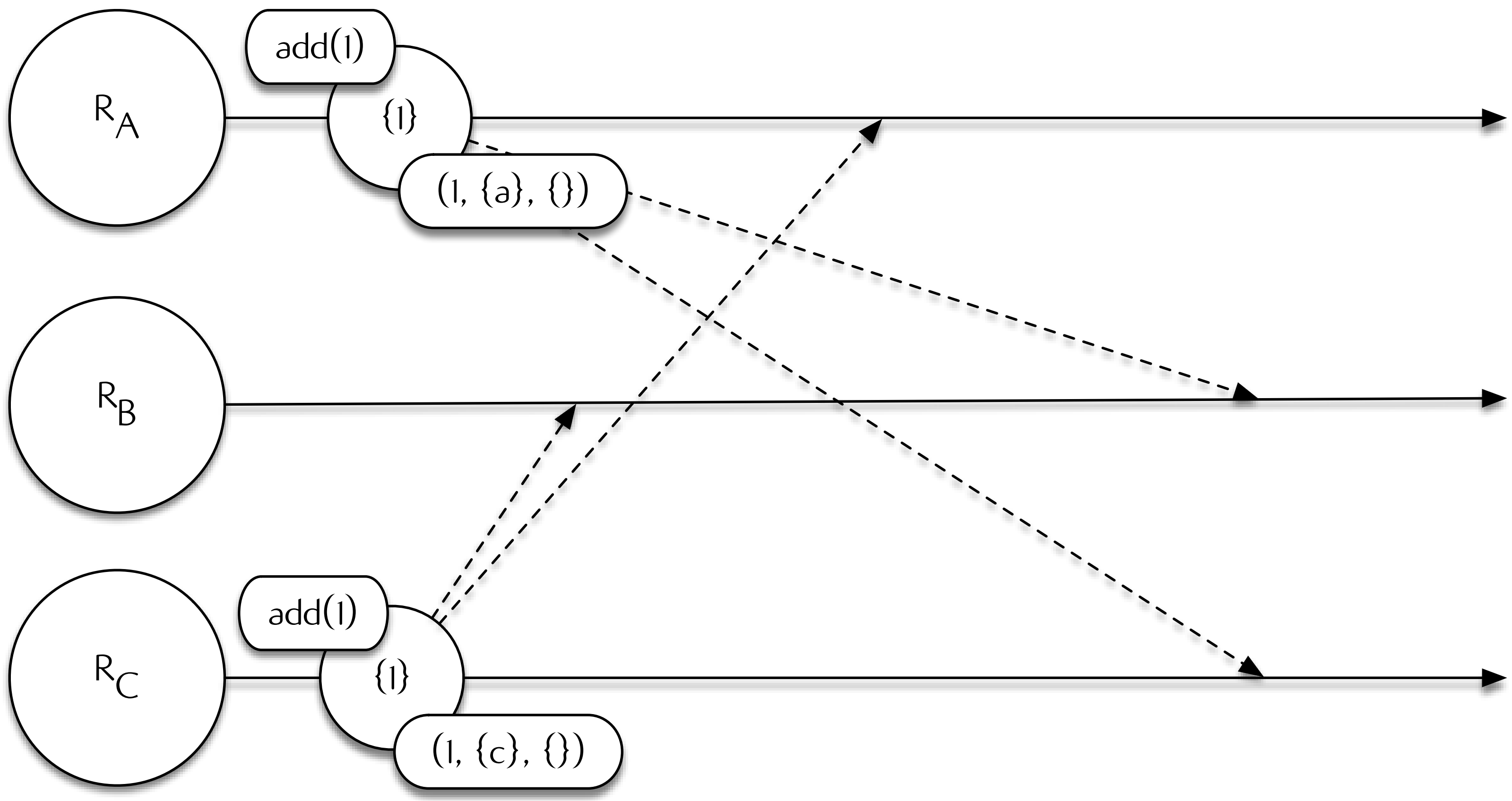
Conflict-Free
Replicated Data Types

- Collection of types
  Sets, counters, registers, flags, maps
- Strong Eventual Consistency
  Objects that receive the same updates, regardless of order, will reach equivalent state

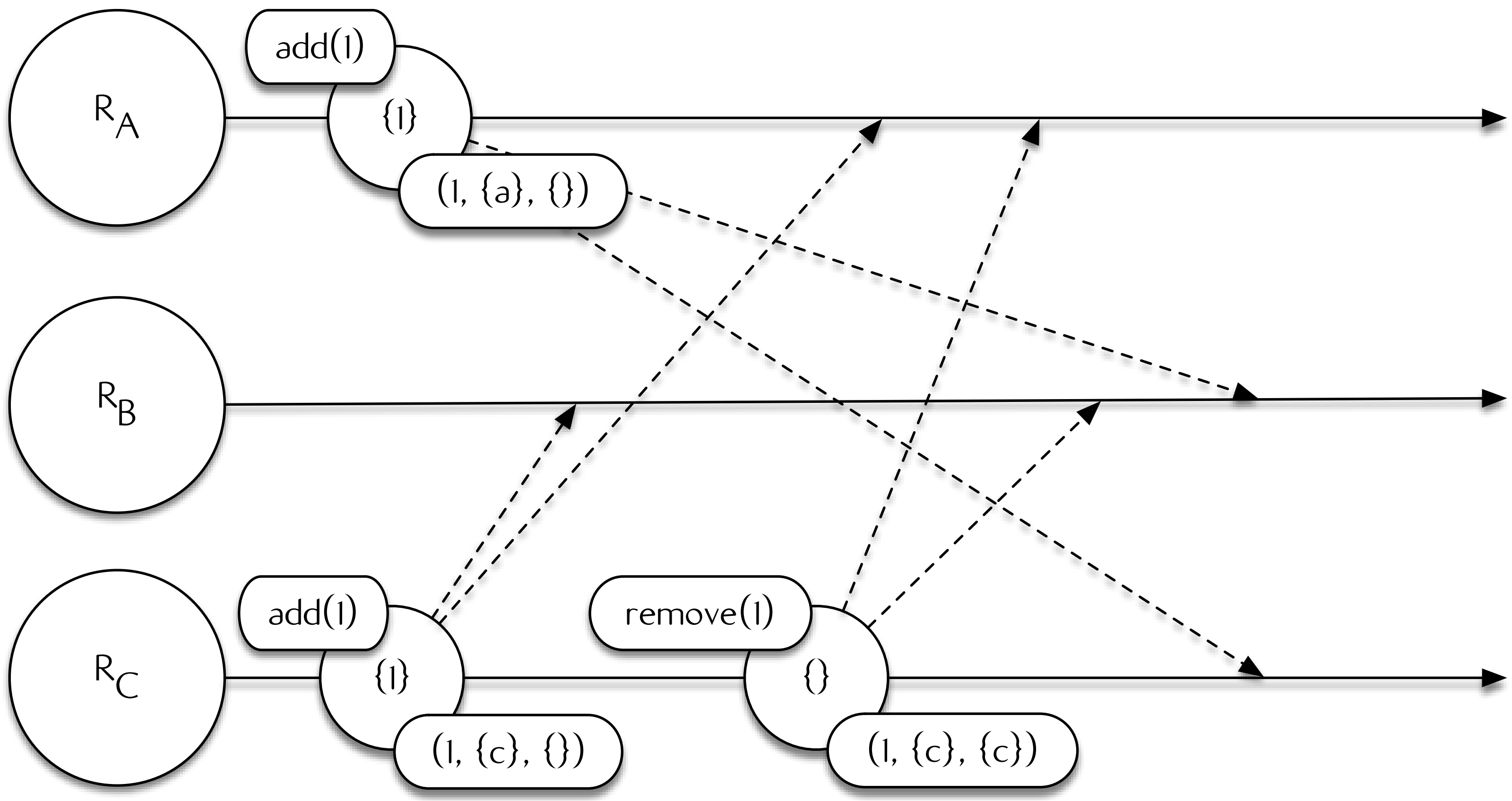


R_A
R_B
R_C

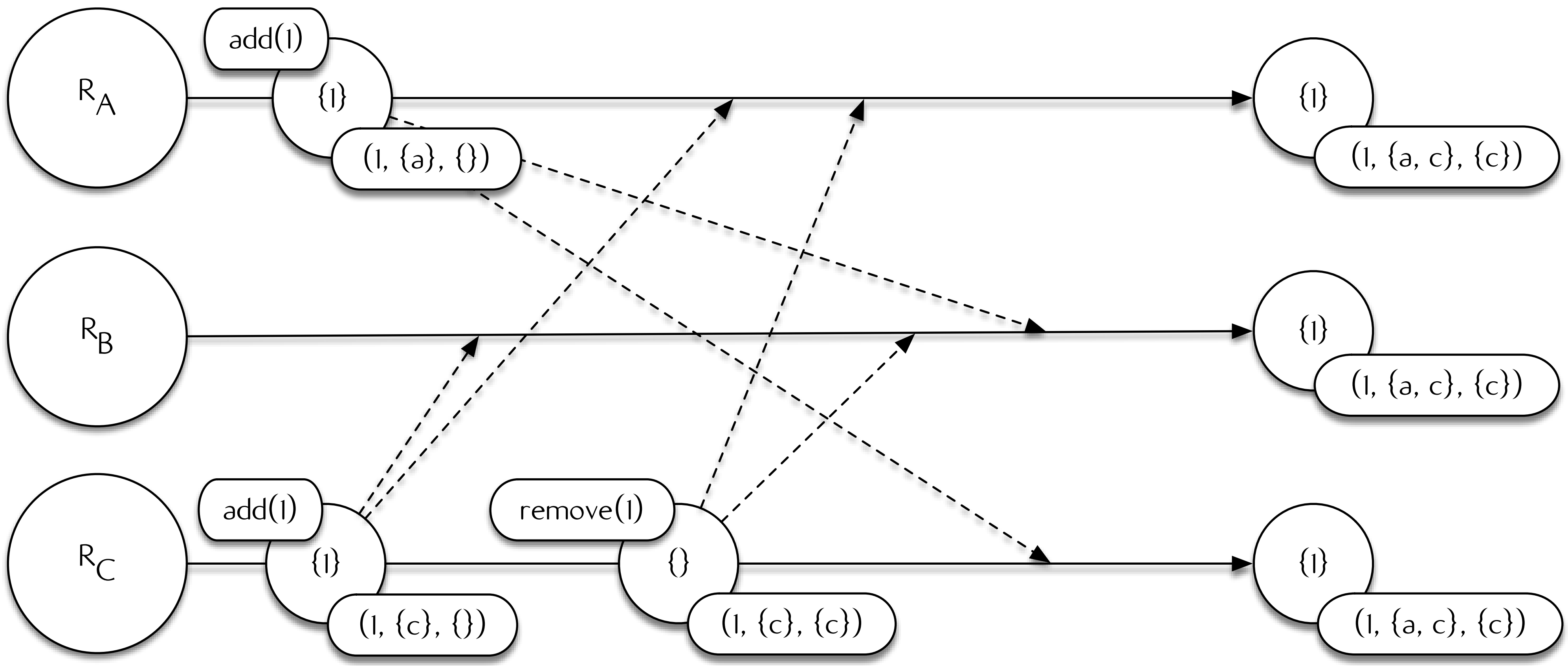

add(1)
R_A
{1}
(1, {a}, {})
R_B
R_C



R_A
add(1)
{1}
(1, {a}, {})
R_B
R_C
add(1)
{1}
(1, {c}, {})



add(1)
R_A
{1}
(1, {a}, {})
R_B
add(1)
R_C
{1}
(1, {c}, {})
remove(1)
{}
(1, {c}, {c})



R_A
add(1)
{1}
(1, {a}, {})
{1}
(1, {a, c}, {c})
R_B
{1}
(1, {a, c}, {c})
R_C
add(1)
{1}
(1, {c}, {})
remove(1)
{}
(1, {c}, {c})
{1}
(1, {a, c}, {c})

Lattice Processing

- **Distributed, deterministic dataflow**
  Distributed, dataflow programming model

Lattice Processing

- **Distributed, deterministic dataflow**
  Distributed, dataflow programming model

- **Convergent data structures**
  Data abstraction is the CRDT

Lattice Processing

- **Distributed, deterministic dataflow**
  Distributed, dataflow programming model

- **Convergent data structures**
  Data abstraction is the CRDT

- **Enables composition**
  Composition preserves SEC

```erlang
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1,2,3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
```

```erlang
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1,2,3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
```

```
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1,2,3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
```

```
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1,2,3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
```

```erlang
%% Create initial set.
S1 = declare(set),

%% Add elements to initial set and update.
update(S1, {add, [1,2,3]}),

%% Create second set.
S2 = declare(set),

%% Apply map operation between S1 and S2.
map(S1, fun(X) -> X * 2 end, S2).
```

Selective Hearing

- Epidemic broadcast protocol
  Runtime system for application state & scope

Selective Hearing

- **Epidemic broadcast protocol**
  Runtime system for application state & scope

- **Peer-to-peer dissemination**
  Pairwise synchronization between peers
  without a central coordinator

Selective Hearing

- **Epidemic broadcast protocol**
  Runtime system for application state & scope

- **Peer-to-peer dissemination**
  Pairwise synchronization between peers without a central coordinator

- **No ordering guarantees on messages**
  Programming model can tolerate message reordering and duplication

What can we build?

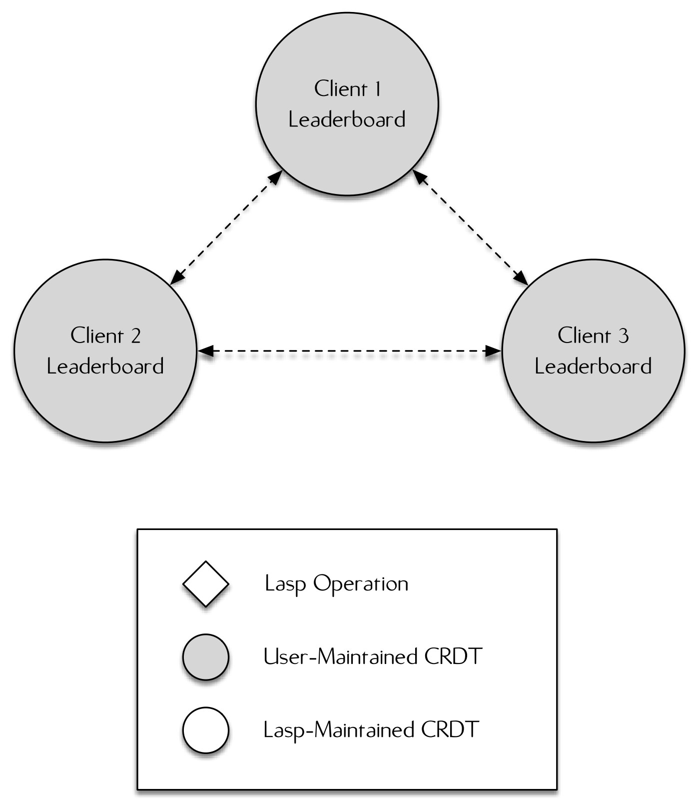
Leaderboard

Leaderboard

- Mobile game platform
  Local leaderboard tracking top-*k* highest scored games

Leaderboard

- **Mobile game platform**
  Local leaderboard tracking top-*k* highest scored games
- **Clients will go offline**
  Clients have limited connectivity and the system still needs to make progress while clients are offline


Client 1
Leaderboard
Client 2
Leaderboard
Client 3
Leaderboard
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT

Leaderboard

- Peer-to-peer dissemination
  Nodes periodically “merge” their state
  with a random peer

Leaderboard

- **Peer-to-peer dissemination**
  Nodes periodically “merge” their state with a random peer
- **Complexity in the data type**
  Each node tracks a top-*k* set of its own games in a bounded set

```erlang
%% Create a leaderboard datatype.
L = declare({top_k, [2]}).

%% Update leaderboard.
update({set, Name, Score}, L).
```

```
%% Create a leaderboard datatype.
L = declare({top_k, [2]}).

%% Update leaderboard.
update({set, Name, Score}, L).
```

```
%% Create a leaderboard datatype.
L = declare({top_k, [2]}).

%% Update leaderboard.
update({set, Name, Score}, L).
```

What if we want to enhance the behavior?

What if we want to enhance the behavior?

Without the creation of a new datatype

What can we build?

Per-User Leaderboard

Per-User Leaderboard

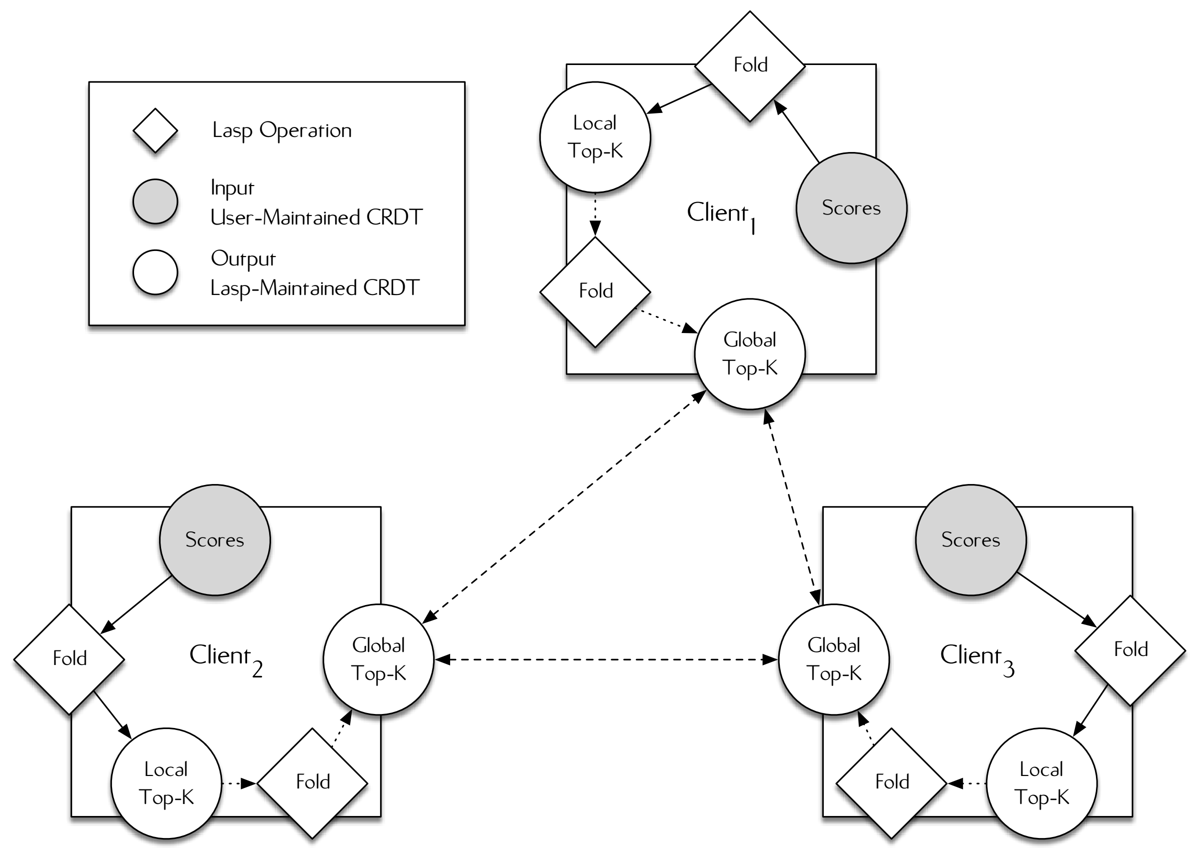
- Enhance existing design
  Only the top score for each user at each device

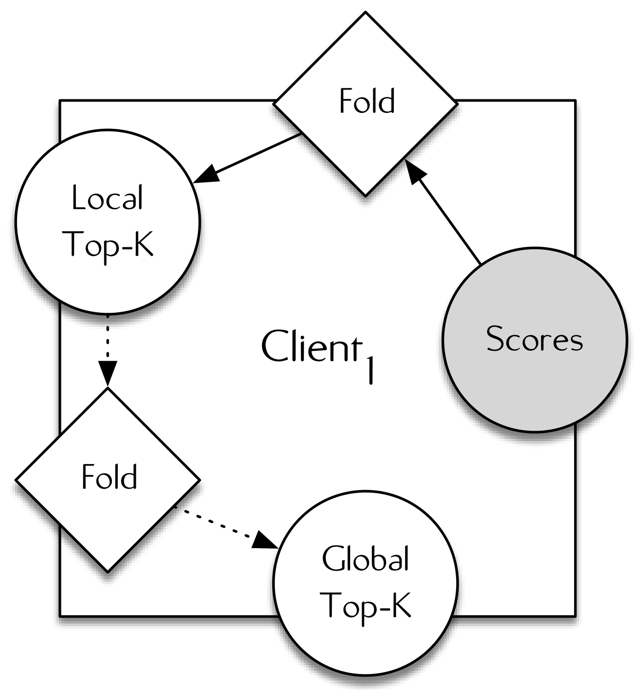
Per-User Leaderboard

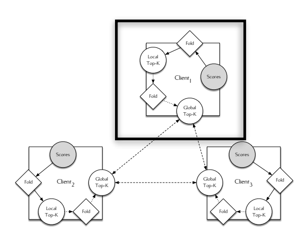
- Enhance existing design
  Only the top score for each user at each device
- Minimize transmitted state
  Prevent transmission of state that is not necessary to perform the computation

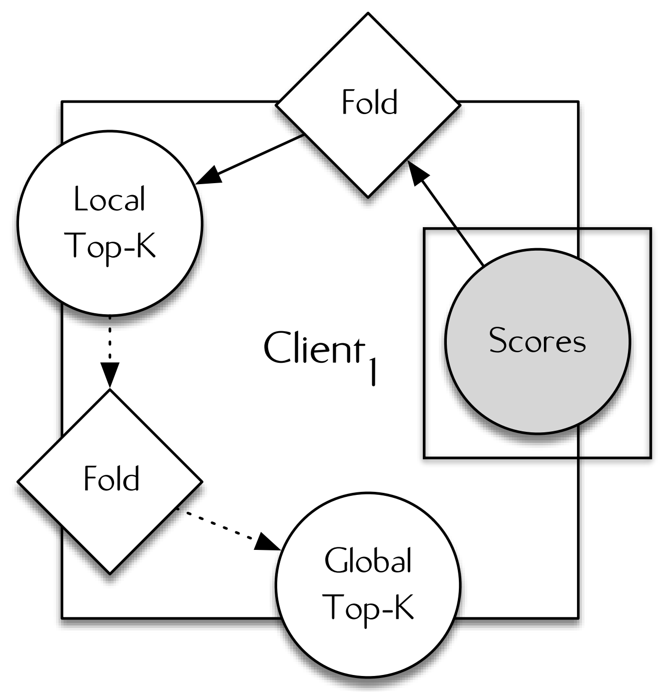
Per-User Leaderboard

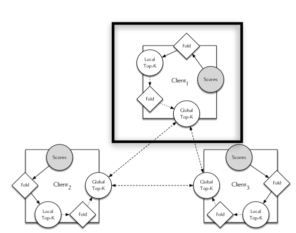
- **Enhance existing design**
  Only the top score for each user at each device
- **Minimize transmitted state**
  Prevent transmission of state that is not necessary to perform the computation
- **Compose data types**
  Build a per-user leaderboard through the composition of existing types

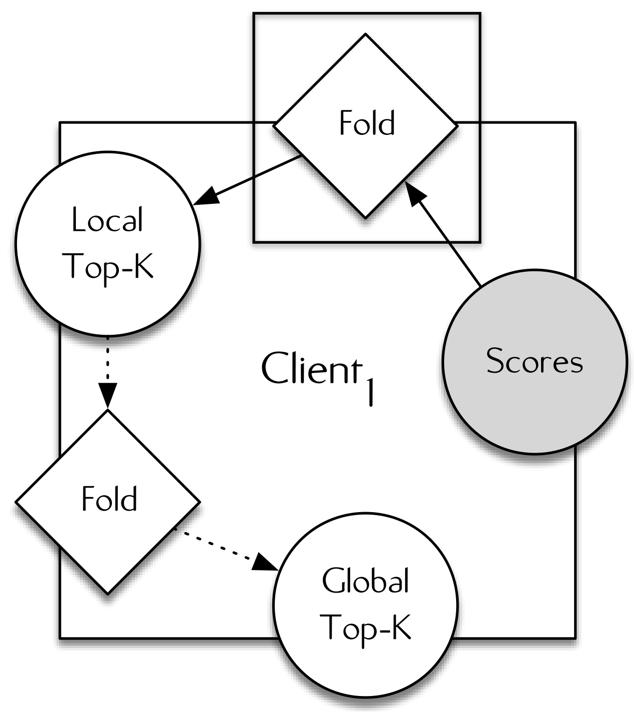

Lasp Operation
Input
User-Maintained CRDT
Output
Lasp-Maintained CRDT
Fold
Local
Top-K
Client1
Scores
Fold
Global
Top-K
Scores
Fold
Client2
Global
Top-K
Local
Top-K
Fold
Scores
Global
Top-K
Client3
Fold
Fold
Local
Top-K

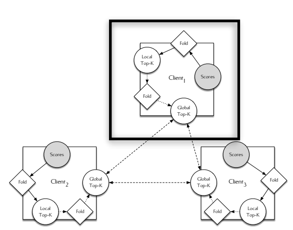

Fold
Local Top-K
Client1
Scores
Fold
Global Top-K

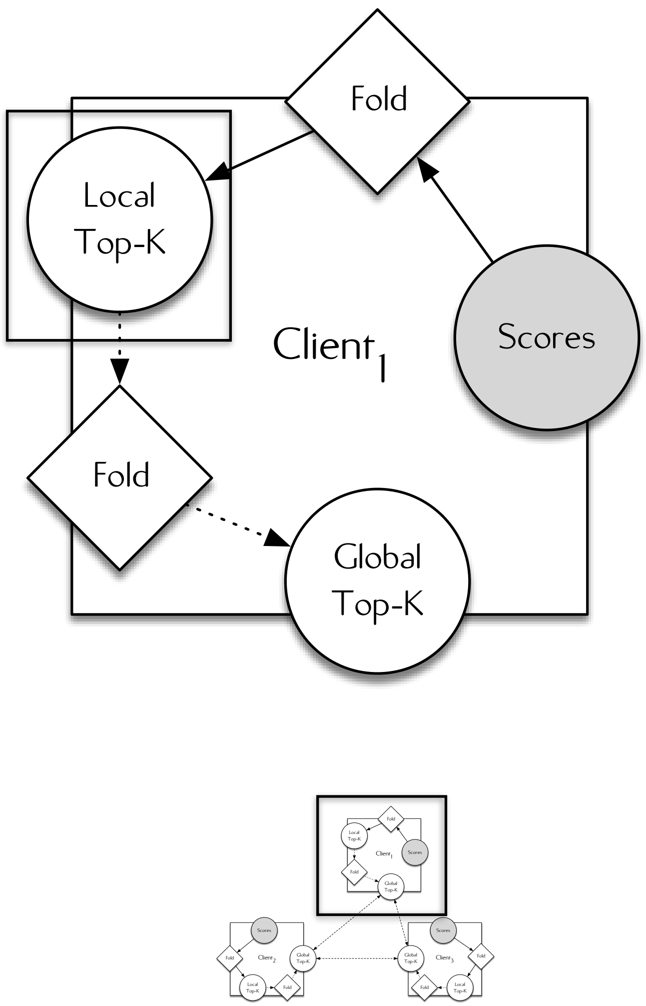


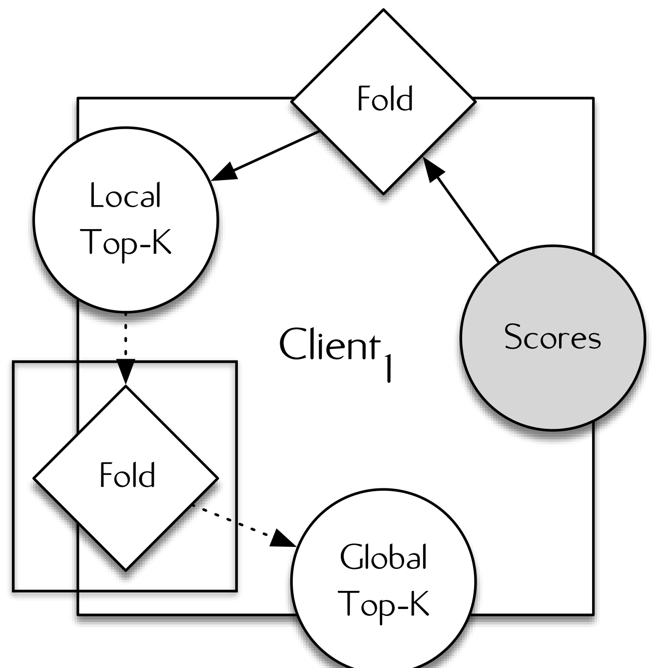

Fold
Local
Top-K
Client$_1$
Scores
Fold
Global
Top-K



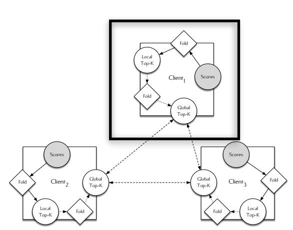

Fold
Local
Top-K
Client1
Scores
Fold
Global
Top-K

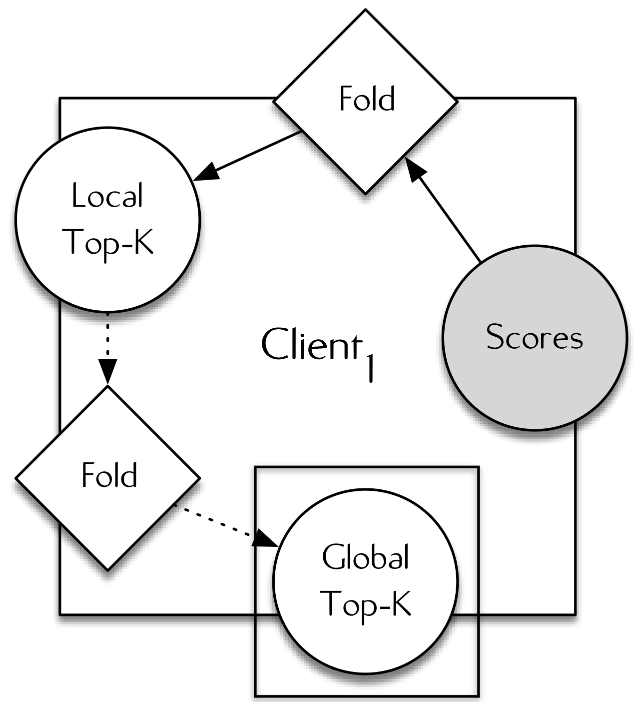


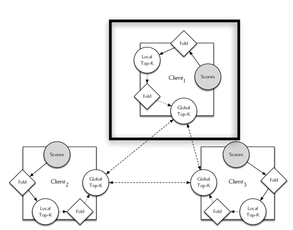

Fold
Local
Top-K
Client$_1$
Scores
Fold
Global
Top-K

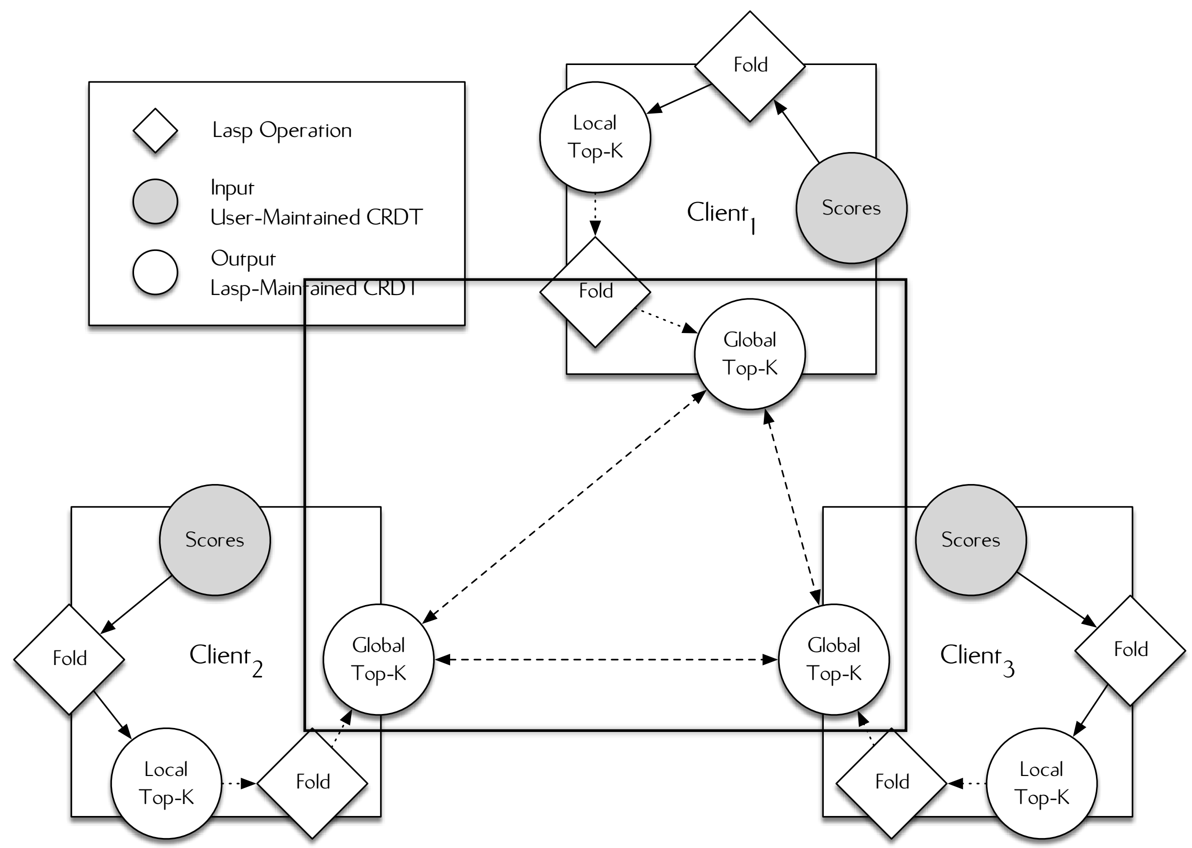


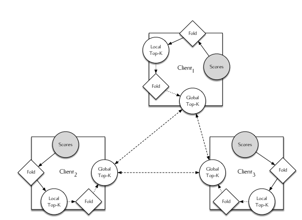
Fold
Local
Top-K
Client1
Scores
Fold
Global
Top-K




Fold
Local
Top-K
Client1
Scores
Fold
Global
Top-K




Lasp Operation
Input
User-Maintained CRDT
Output
Lasp-Maintained CRDT
Fold
Local
Top-K
Client$_1$
Scores
Fold
Global
Top-K
Scores
Fold
Client$_2$
Global
Top-K
Local
Top-K
Fold
Global
Top-K
Client$_3$
Scores
Fold
Fold
Local
Top-K



Per-User Leaderboard

- Dynamically scoped variables
  Variable which take different values depending on where it is executing

Per-User Leaderboard

- Dynamically scoped variables
  Variable which take different values depending on where it is executing

- Dynamically scoped fold operation
  Perform a distributed “reduce” operation that combines the state of a dynamically scoped variables across

```
%% Create a global leaderboard.
G = declare({top_k, [10]}).

%% Create a local leaderboard.
L = declare_dynamic({top_k, [10]}).

%% Create a set of scores.
S = declare_dynamic(set).

%% Compute local top-k list.
fold(S, fun max_by_name/2, L).

%% Compute global top-k list.
fold_dynamic(L, fun max_by_name/2, G).
```

```
%% Create a global leaderboard.
G = declare({top_k, [10]}).

%% Create a local leaderboard.
L = declare_dynamic({top_k, [10]}).

%% Create a set of scores.
S = declare_dynamic(set).

%% Compute local top-k list.
fold(S, fun max_by_name/2, L).

%% Compute global top-k list.
fold_dynamic(L, fun max_by_name/2, G).
```

```
%% Create a global leaderboard.
G = declare({top_k, [10]}).

%% Create a local leaderboard.
L = declare_dynamic({top_k, [10]}).

%% Create a set of scores.
S = declare_dynamic(set).

%% Compute local top-k list.
fold(S, fun max_by_name/2, L).

%% Compute global top-k list.
fold_dynamic(L, fun max_by_name/2, G).
```

```
%% Create a global leaderboard.
G = declare({top_k, [10]}).

%% Create a local leaderboard.
L = declare_dynamic({top_k, [10]}).

%% Create a set of scores.
S = declare_dynamic(set).

%% Compute local top-k list.
fold(S, fun max_by_name/2, L).

%% Compute global top-k list.
fold_dynamic(L, fun max_by_name/2, G).
```

```
%% Create a global leaderboard.
G = declare({top_k, [10]}).

%% Create a local leaderboard.
L = declare_dynamic({top_k, [10]}).

%% Create a set of scores.
S = declare_dynamic(set).

%% Compute local top-k list.
fold(S, fun max_by_name/2, L).

%% Compute global top-k list.
fold_dynamic(L, fun max_by_name/2, G).
```

```
%% Create a global leaderboard.
G = declare({top_k, [10]}).

%% Create a local leaderboard.
L = declare_dynamic({top_k, [10]}).

%% Create a set of scores.
S = declare_dynamic(set).

%% Compute local top-k list.
fold(S, fun max_by_name/2, L).

%% Compute global top-k list.
fold_dynamic(L, fun max_by_name/2, G).
```

Let's look at a larger application

Let’s look at a larger application

With visible non-monotonicity

What can we build?

Advertisement Counter

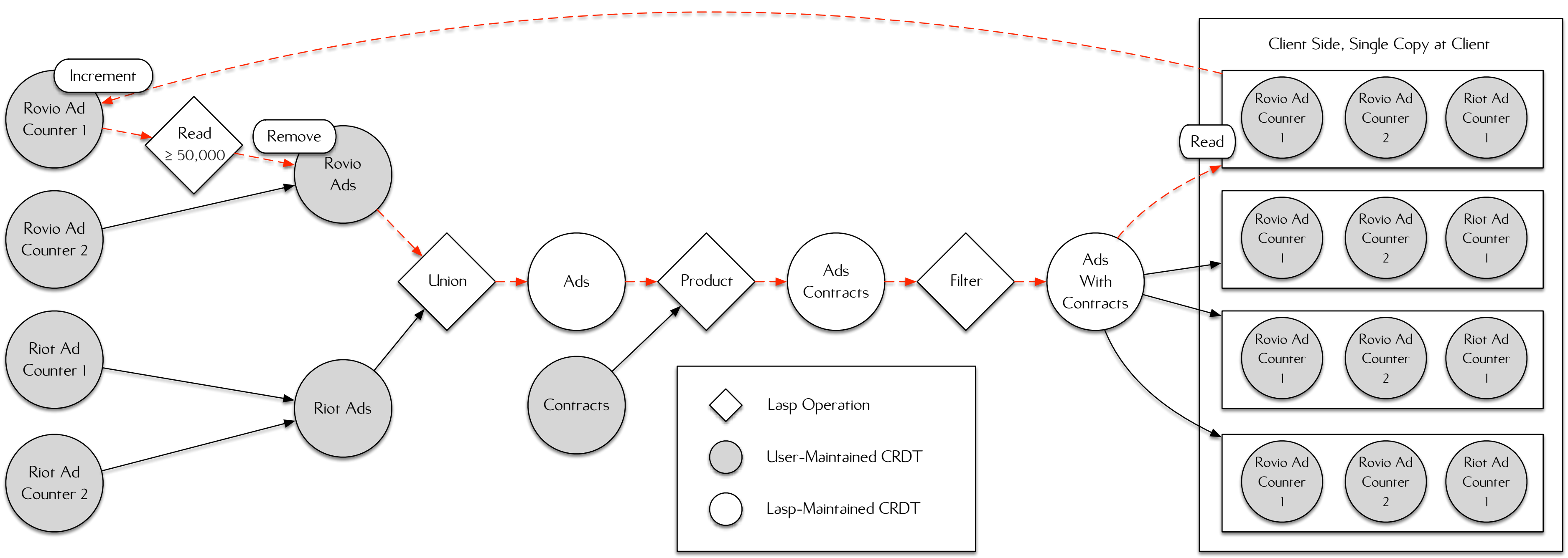
Advertisement Counter

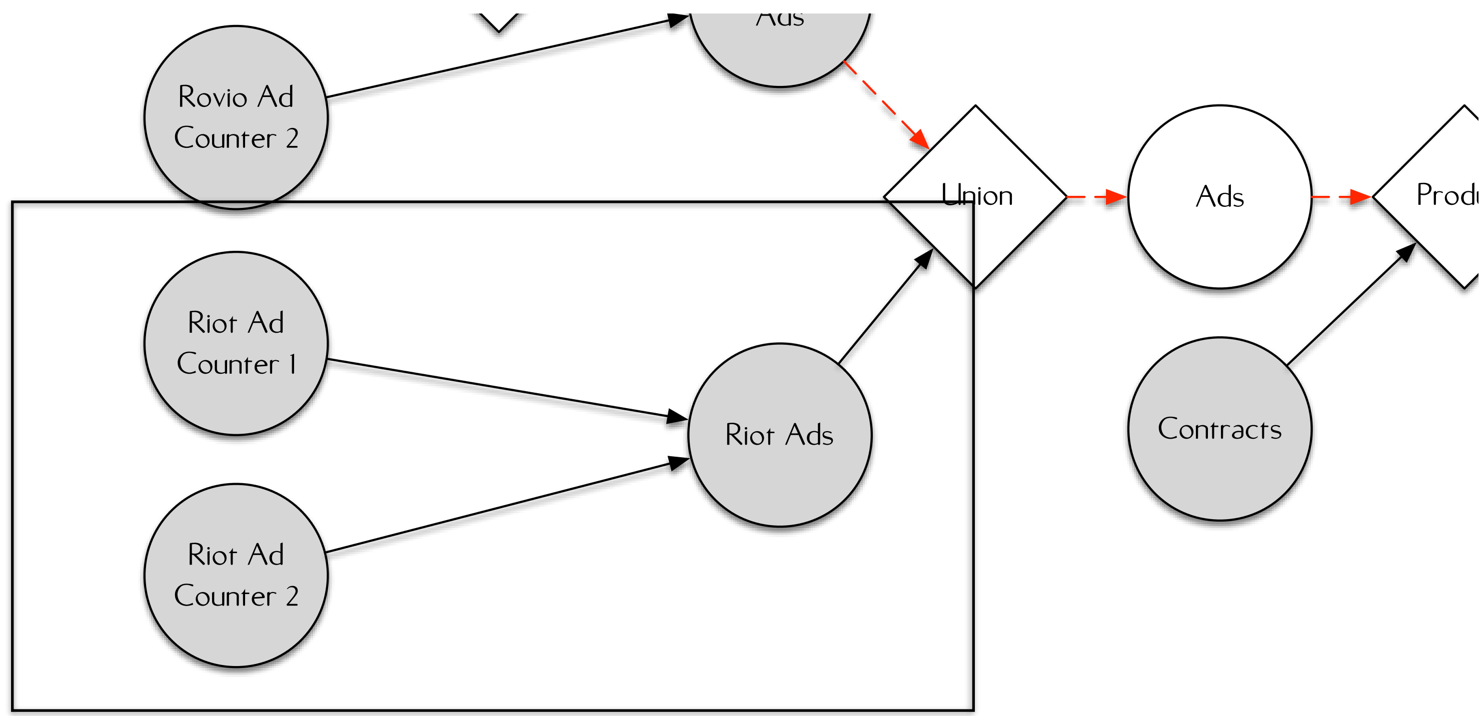
- Mobile game platform selling advertisement space
  Advertisements are paid according to a minimum number of impressions

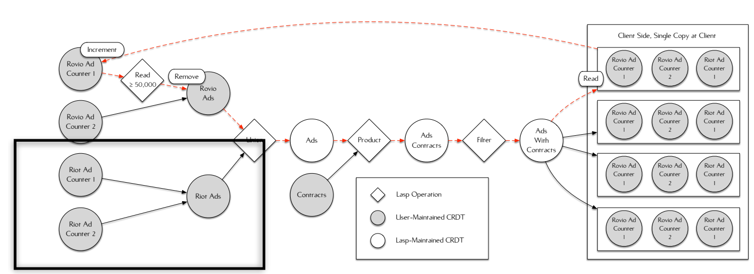
Advertisement Counter

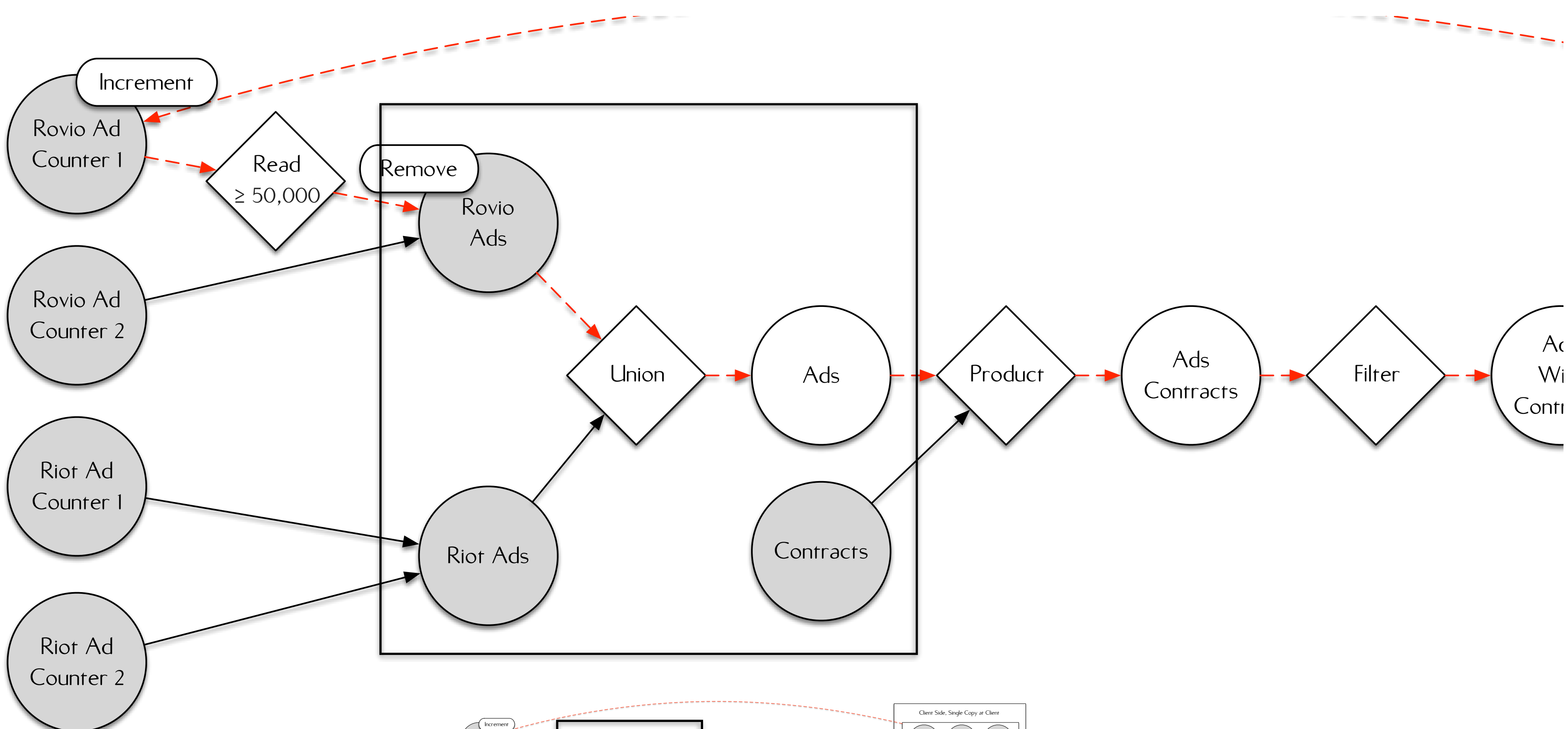
- **Mobile game platform selling advertisement space**
  Advertisements are paid according to a minimum number of impressions

- **Clients will go offline**
  Clients have limited connectivity and the system still needs to make progress while clients are offline

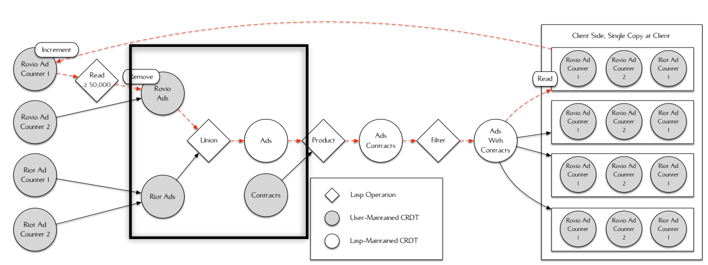

Client Side, Single Copy at Client
Increment
Rovio Ad Counter 1
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter 2
Riot Ad Counter 1
Riot Ad Counter 2
Riot Ads
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT

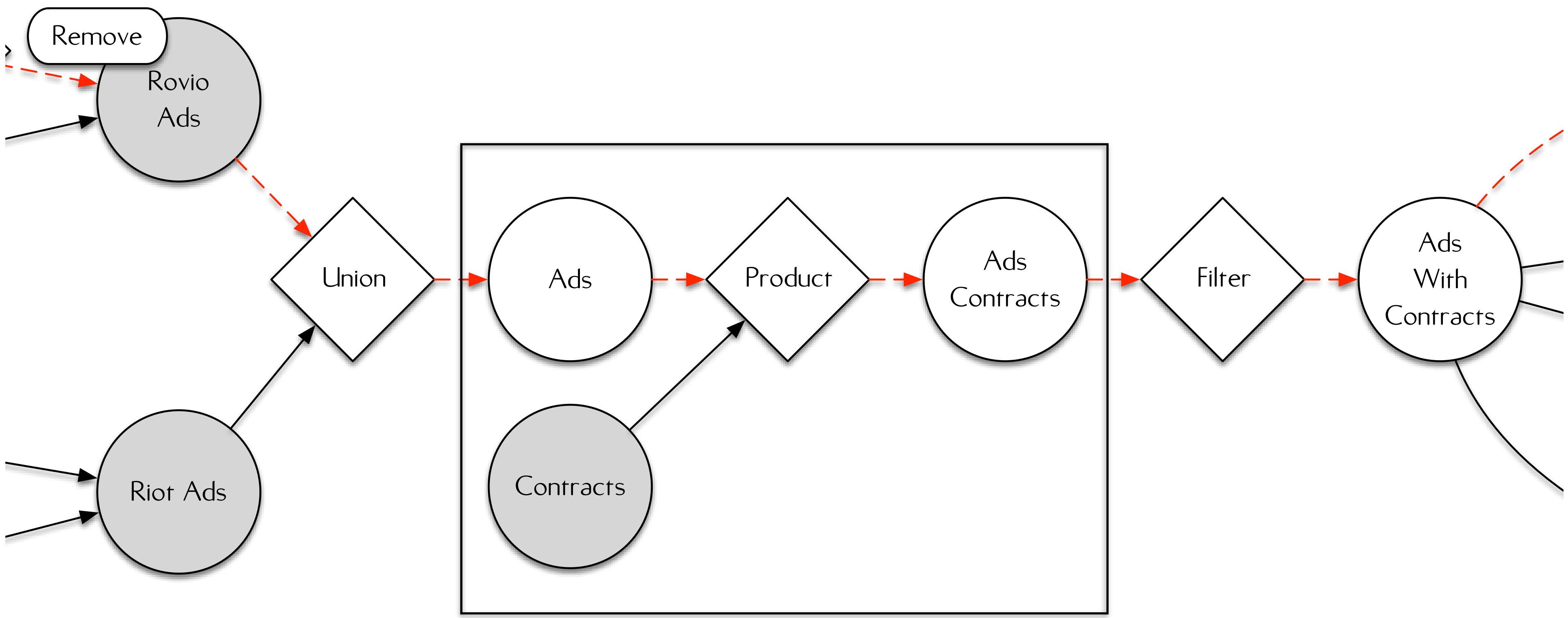

Ads
Rovio Ad Counter 2
Union
Ads
Riot Ad Counter 1
Riot Ad Counter 2
Riot Ads
Contracts

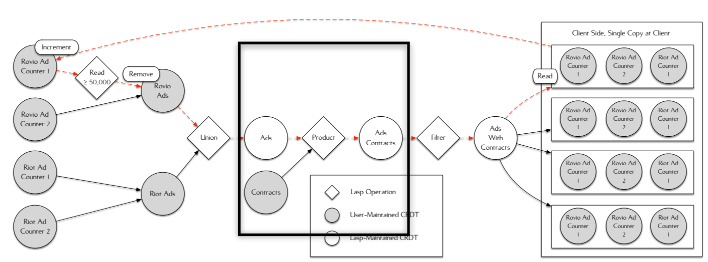


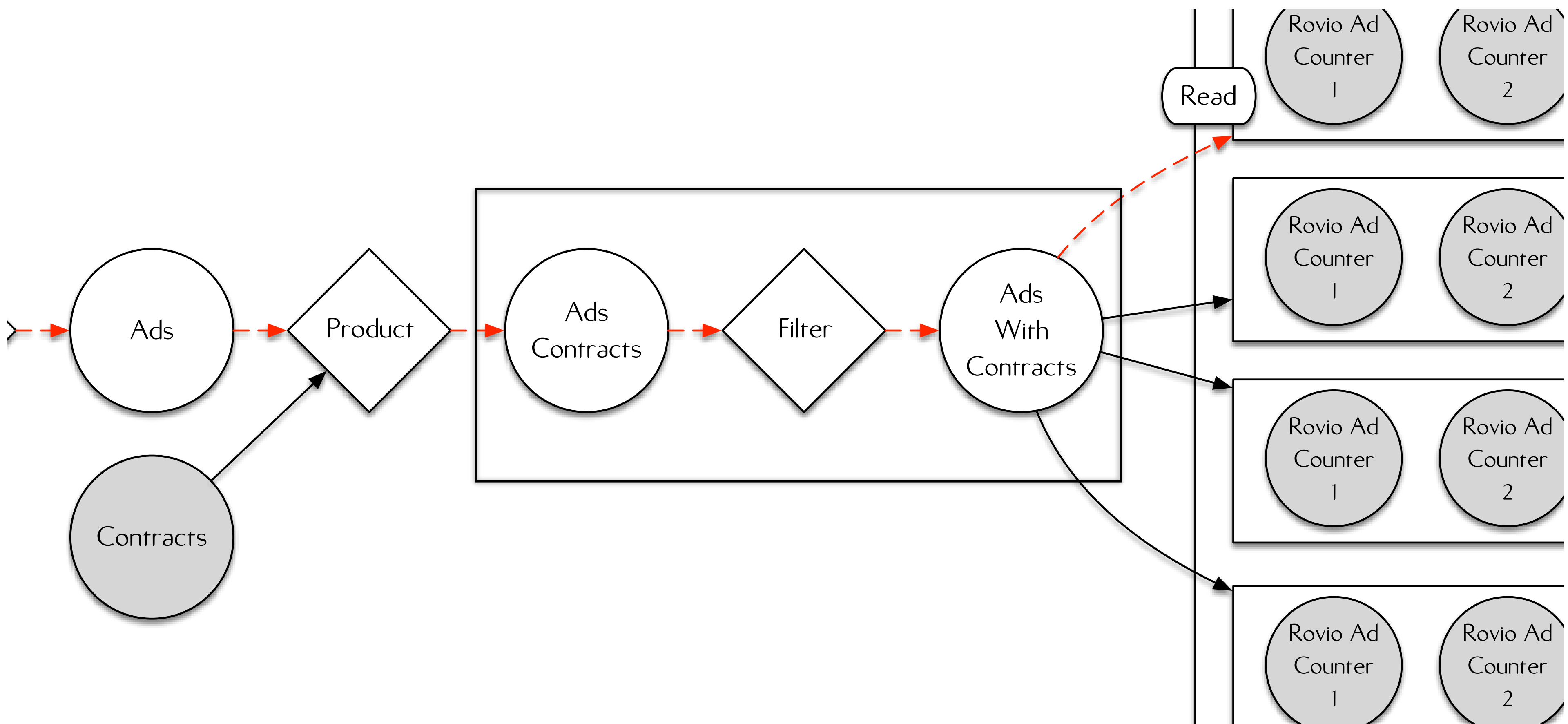


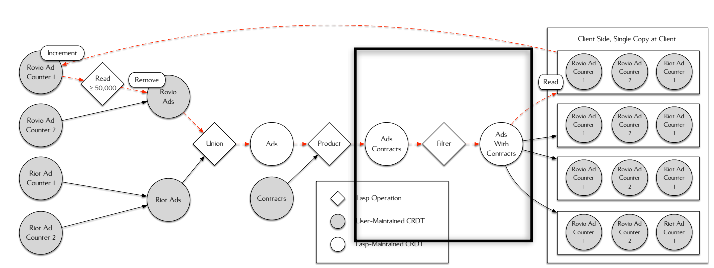
Increment
Rovio Ad Counter 1
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter 2
Union
Ads
Product
Ads Contracts
Filter
Riot Ad Counter 1
Riot Ads
Contracts
Riot Ad Counter 2

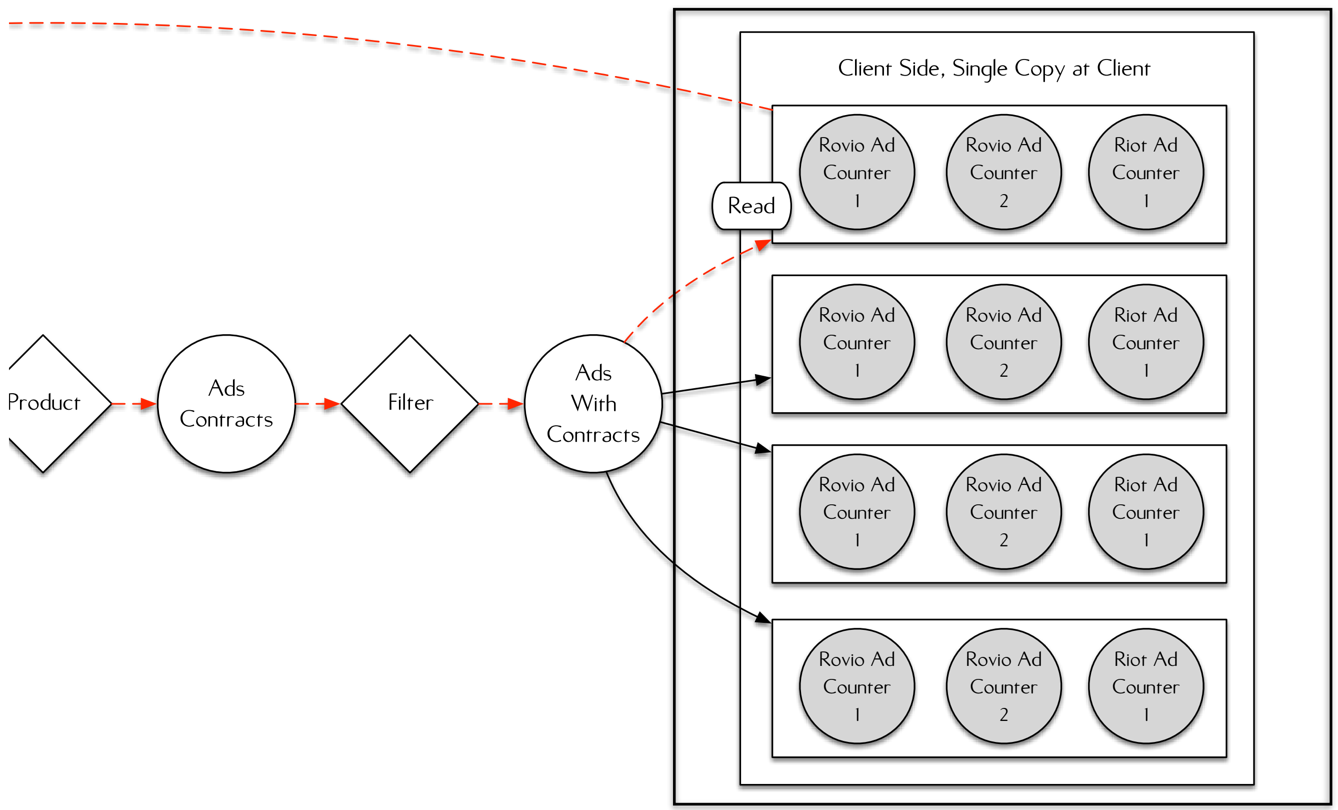


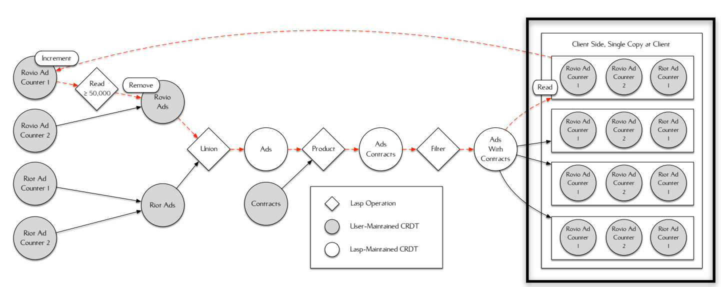

Remove
Rovio Ads
Riot Ads
Union
Ads
Contracts
Product
Ads Contracts
Filter
Ads With Contracts

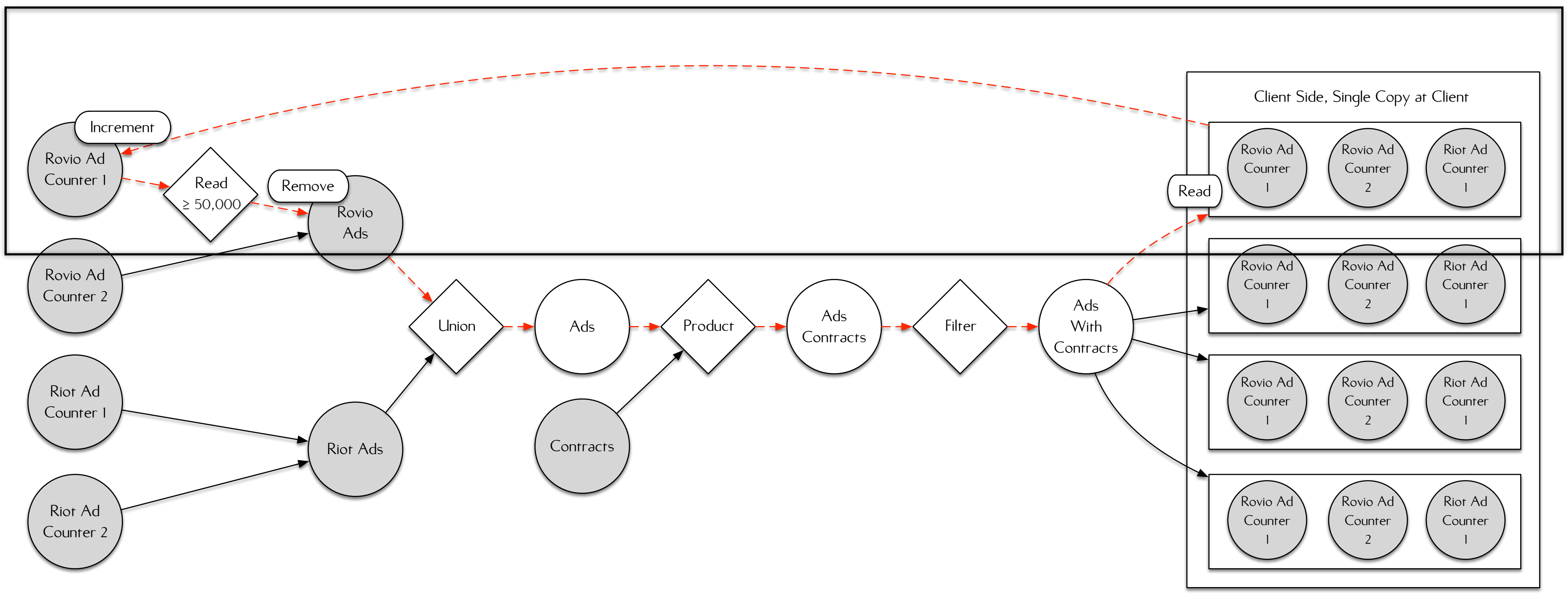


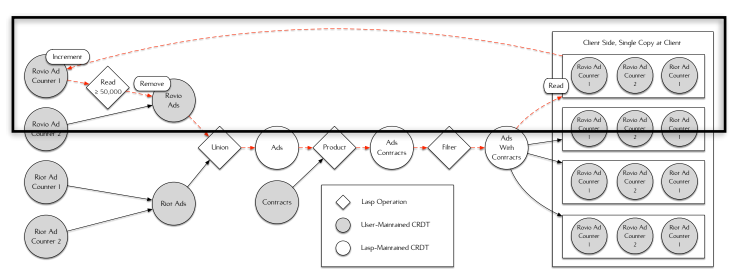

Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Rovio Ad Counter 1
Rovio Ad Counter 2
Rovio Ad Counter 1
Rovio Ad Counter 2
Rovio Ad Counter 1
Rovio Ad Counter 2
Rovio Ad Counter 1
Rovio Ad Counter 2

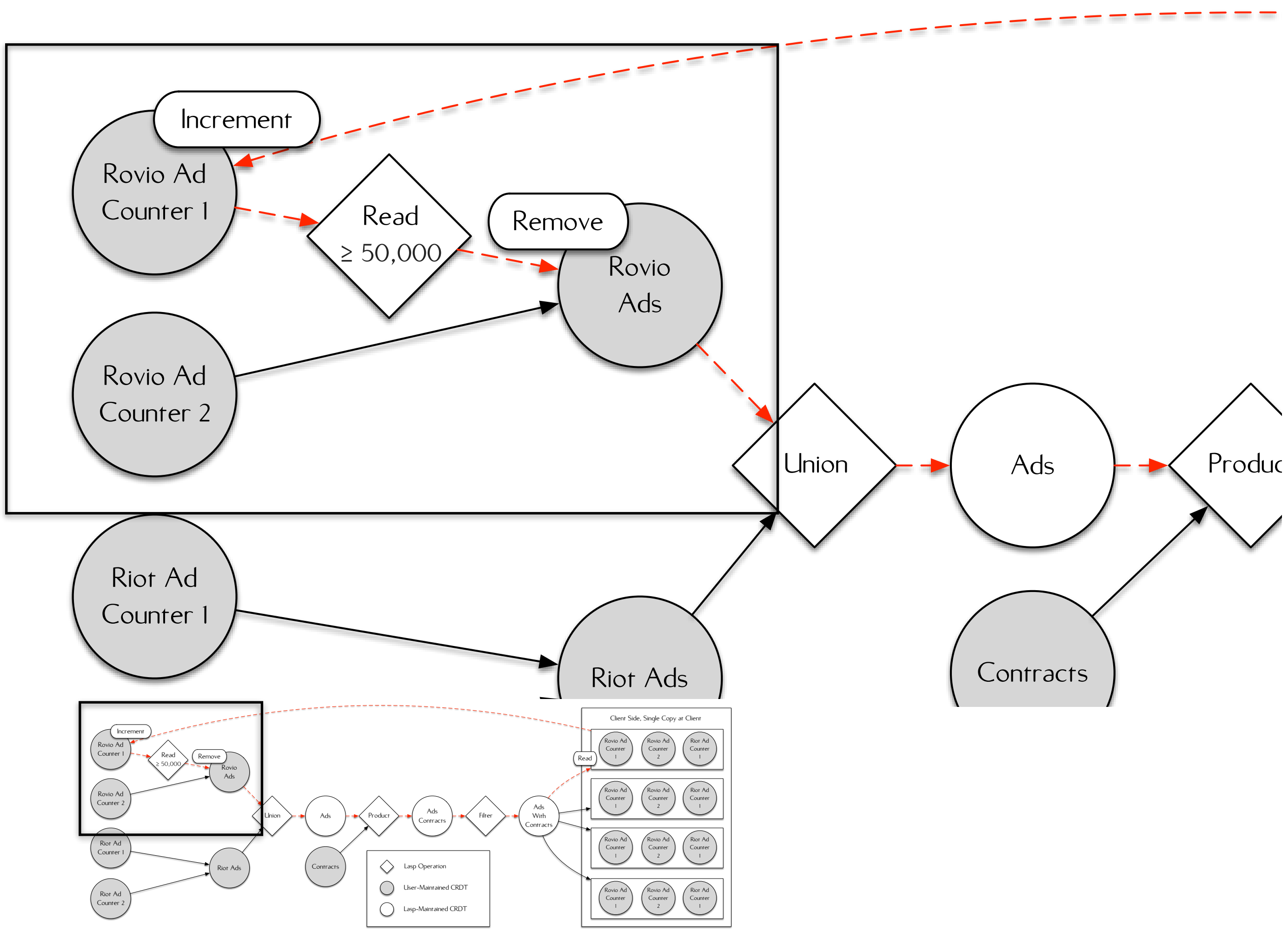


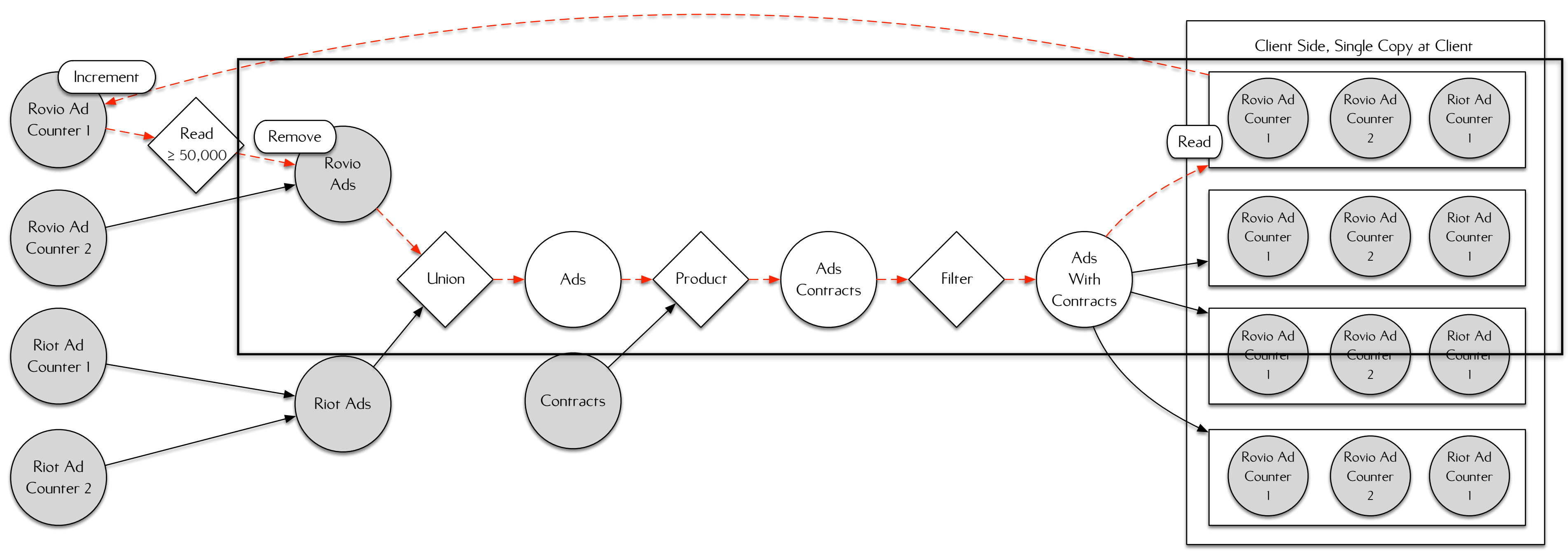

Client Side, Single Copy at Client
Read
Product
Ads Contracts
Filter
Ads With Contracts
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1



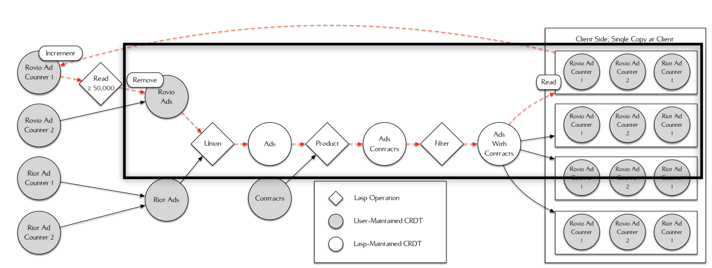

Client Side, Single Copy at Client
Increment
Rovio Ad Counter 1
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter 2
Riot Ad Counter 1
Riot Ad Counter 2
Riot Ads
Union
Ads
Contracts
Product
Ads Contracts
Filter
Ads With Contracts
Read
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1




Increment
Rovio Ad Counter 1
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter 2
Union
Ads
Produc
Riot Ad Counter 1
Riot Ads
Contracts



Client Side, Single Copy at Client
Increment
Rovio Ad Counter 1
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter 2
Union
Ads
Product
Ads Contracts
Filter
Ads With Contracts
Read
Riot Ad Counter 1
Riot Ads
Riot Ad Counter 2
Contracts
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1



Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT

Advertisement Counter

- Completely monotonic
  Disabling advertisements and contracts are all modeled through monotonic state growth

Advertisement Counter

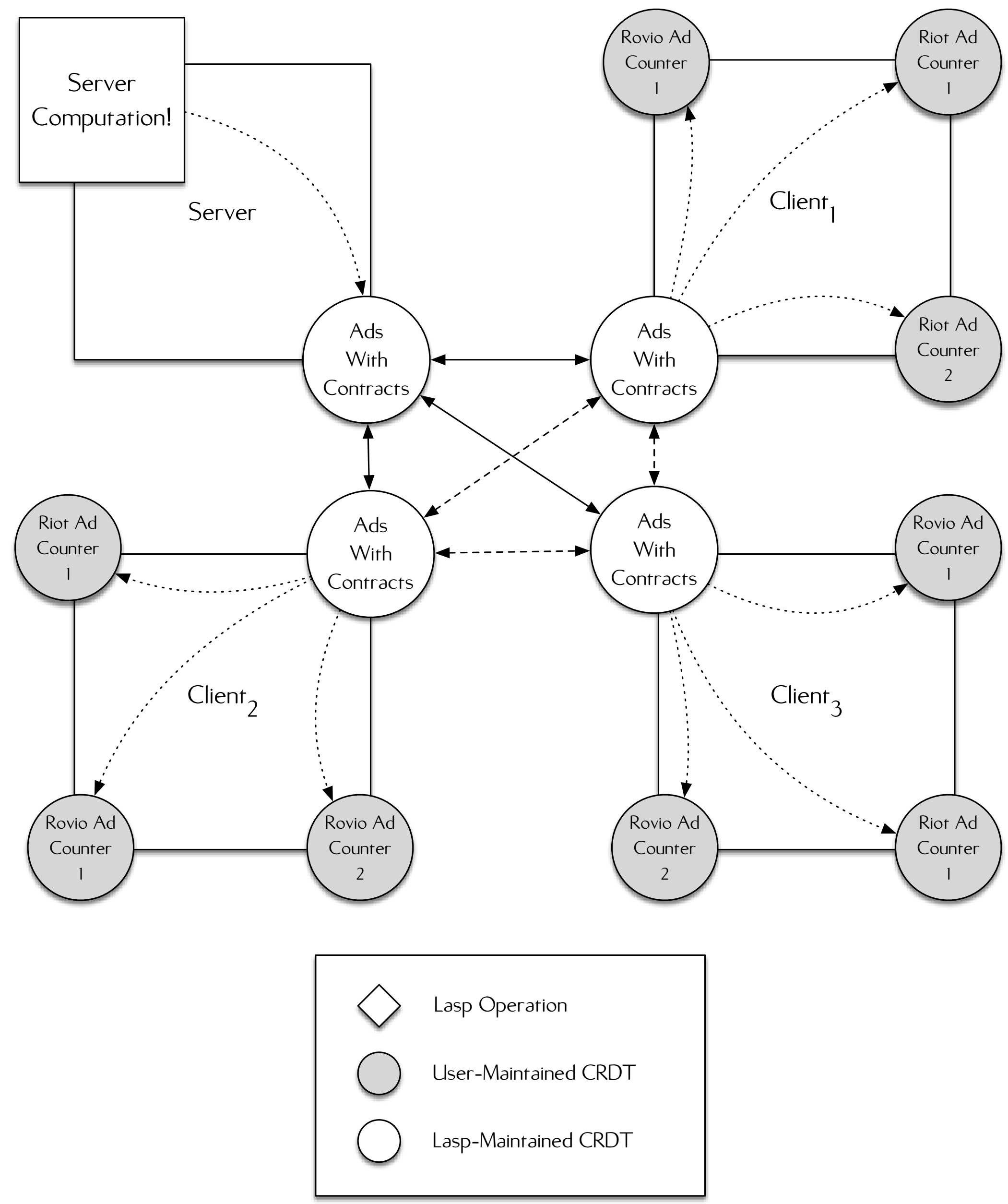
- Completely monotonic
  Disabling advertisements and contracts are all modeled through monotonic state growth
- Arbitrary distribution
  Use of convergent data structures allows computational graph to be arbitrarily distributed

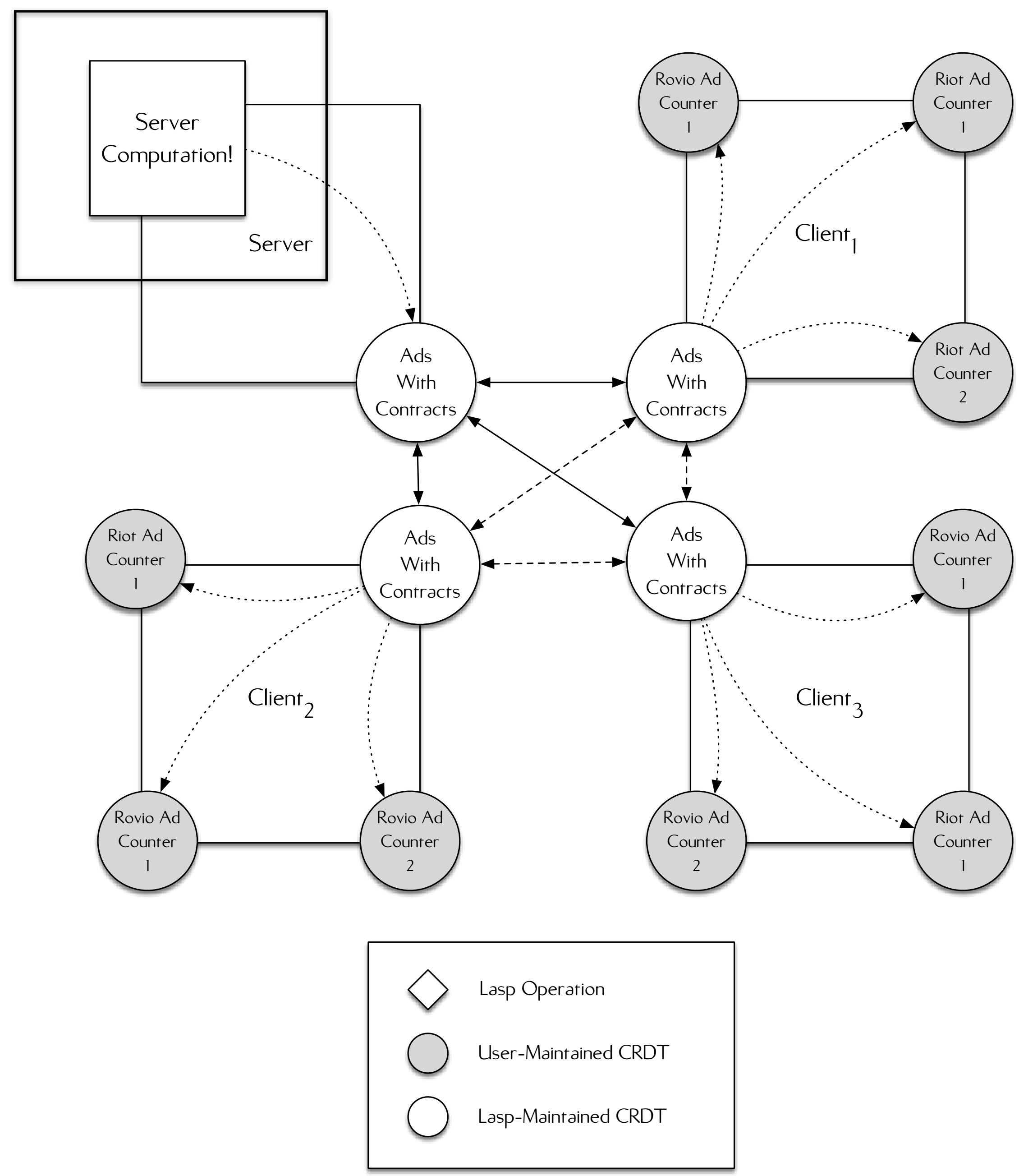
Advertisement Counter

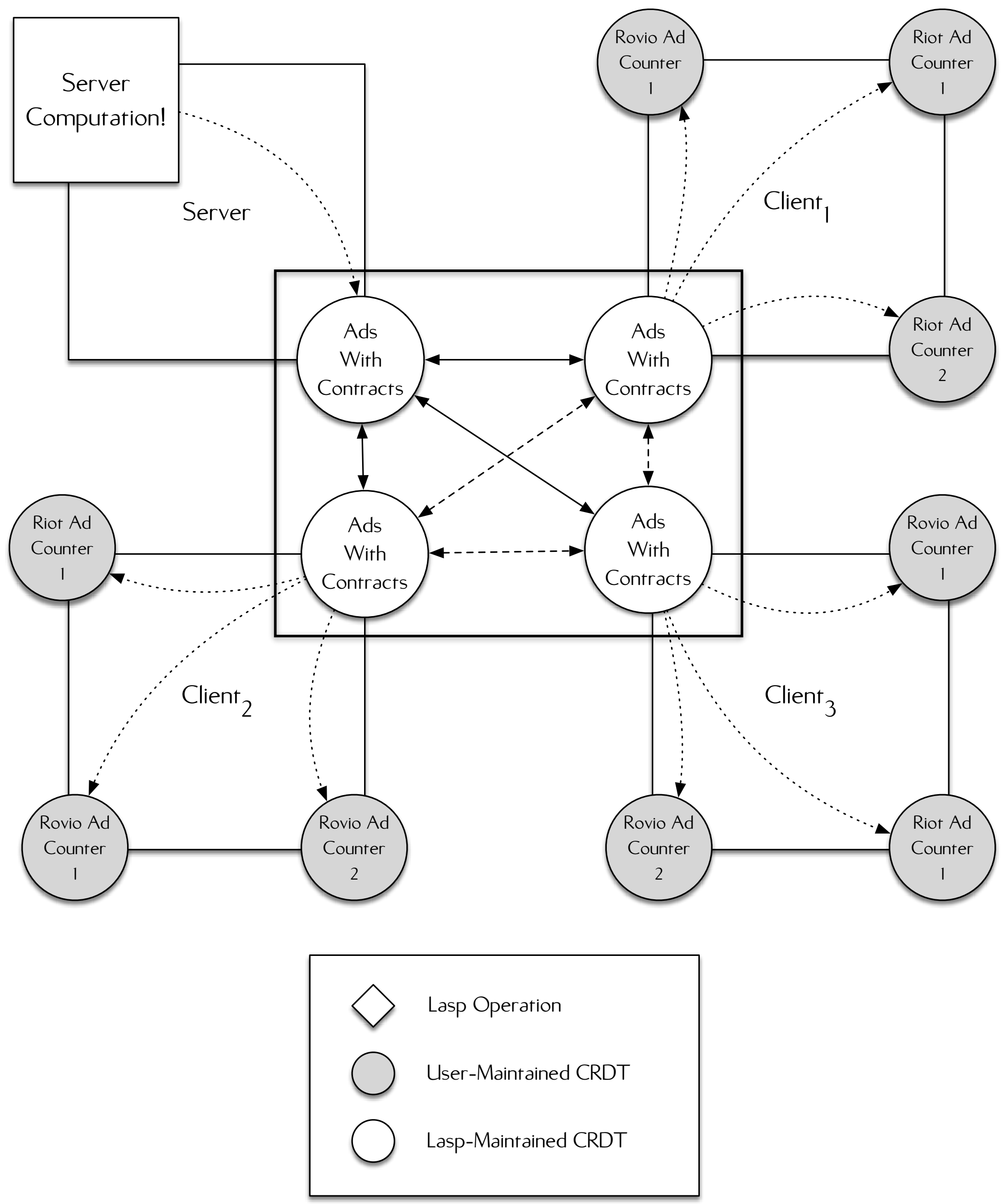
- Completely monotonic
  Disabling advertisements and contracts are all modeled through monotonic state growth

- Arbitrary distribution
  Use of convergent data structures allows computational graph to be arbitrarily distributed

- Divergence
  Divergence is a factor of synchronization period


Server Computation!
Server
Ads With Contracts
Rovio Ad Counter 1
Riot Ad Counter 1
Client1
Riot Ad Counter 2
Ads With Contracts
Riot Ad Counter 1
Ads With Contracts
Ads With Contracts
Rovio Ad Counter 1
Client2
Client3
Rovio Ad Counter 1
Rovio Ad Counter 2
Rovio Ad Counter 2
Riot Ad Counter 1
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT


Server
Computation!
Server
Ads With Contracts
Ads With Contracts
Ads With Contracts
Ads With Contracts
Client$_1$
Client$_2$
Client$_3$
Rovio Ad Counter 1
Riot Ad Counter 1
Riot Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT


Server Computation!
Server
Client1
Client2
Client3
Ads With Contracts
Ads With Contracts
Ads With Contracts
Ads With Contracts
Rovio Ad Counter 1
Riot Ad Counter 1
Riot Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Rovio Ad Counter 2
Riot Ad Counter 1
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT

Advertisement Counter

- “Servers” as peers to “clients”
  Servers are peers to clients that perform additional computation

Advertisement Counter

- “Servers” as peers to “clients”
  Servers are peers to clients that perform additional computation
  - Any node can disable an advertisement under this model given enough information

Advertisement Counter

- “Servers” as peers to “clients”
  Servers are peers to clients that perform additional computation
  - Any node can disable an advertisement under this model given enough information
- “Servers” as trusted nodes
  Serve as a location for performing “exactly once” side-effects

Advertisement Counter

- **“Servers” as peers to “clients”**
  Servers are peers to clients that perform additional computation
  - Any node can disable an advertisement under this model given enough information
- **“Servers” as trusted nodes**
  Serve as a location for performing “exactly once” side-effects
  - Billing customers must be done at a central point by a trusted node in the system

We’ve build up from zero synchronization

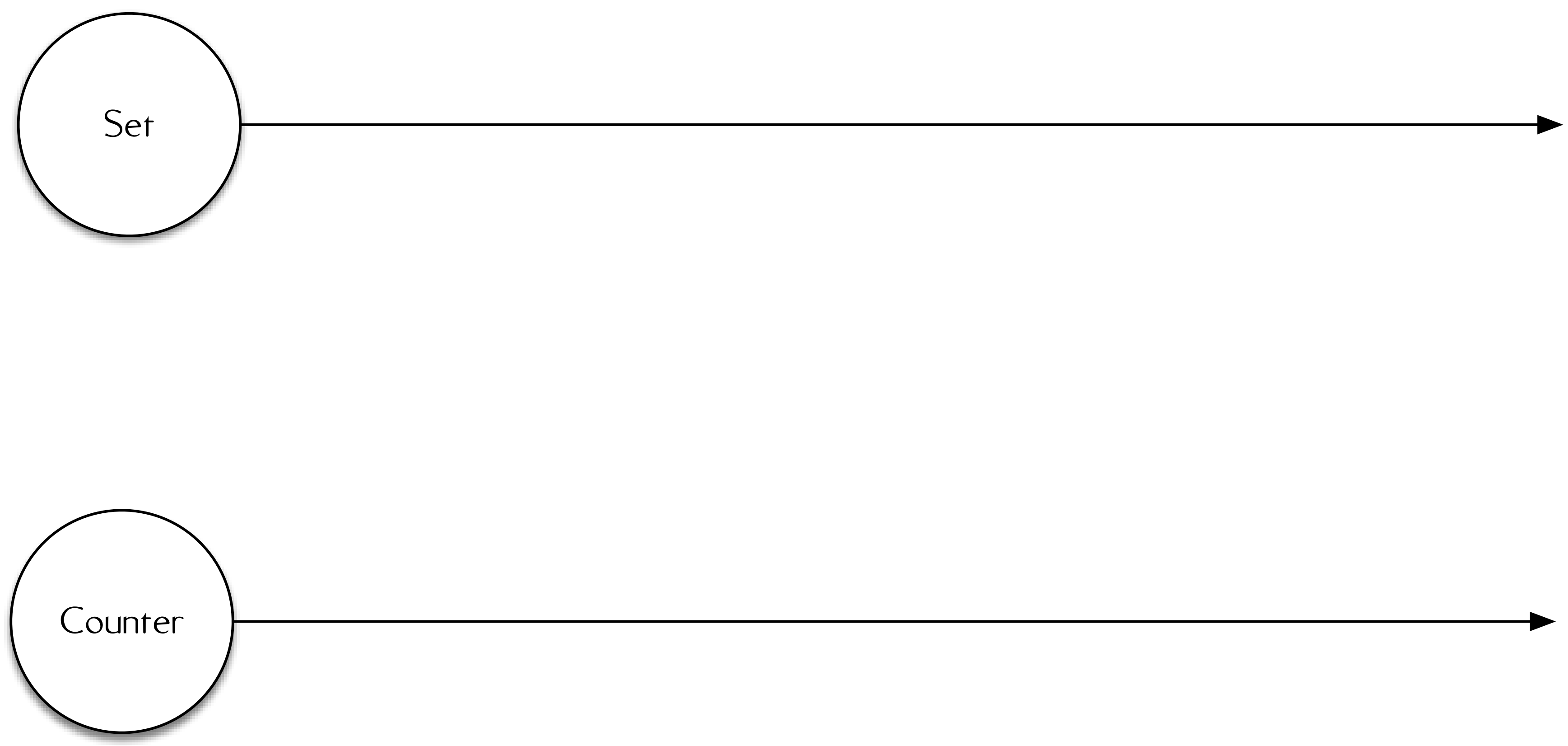
We’ve build up from zero synchronization

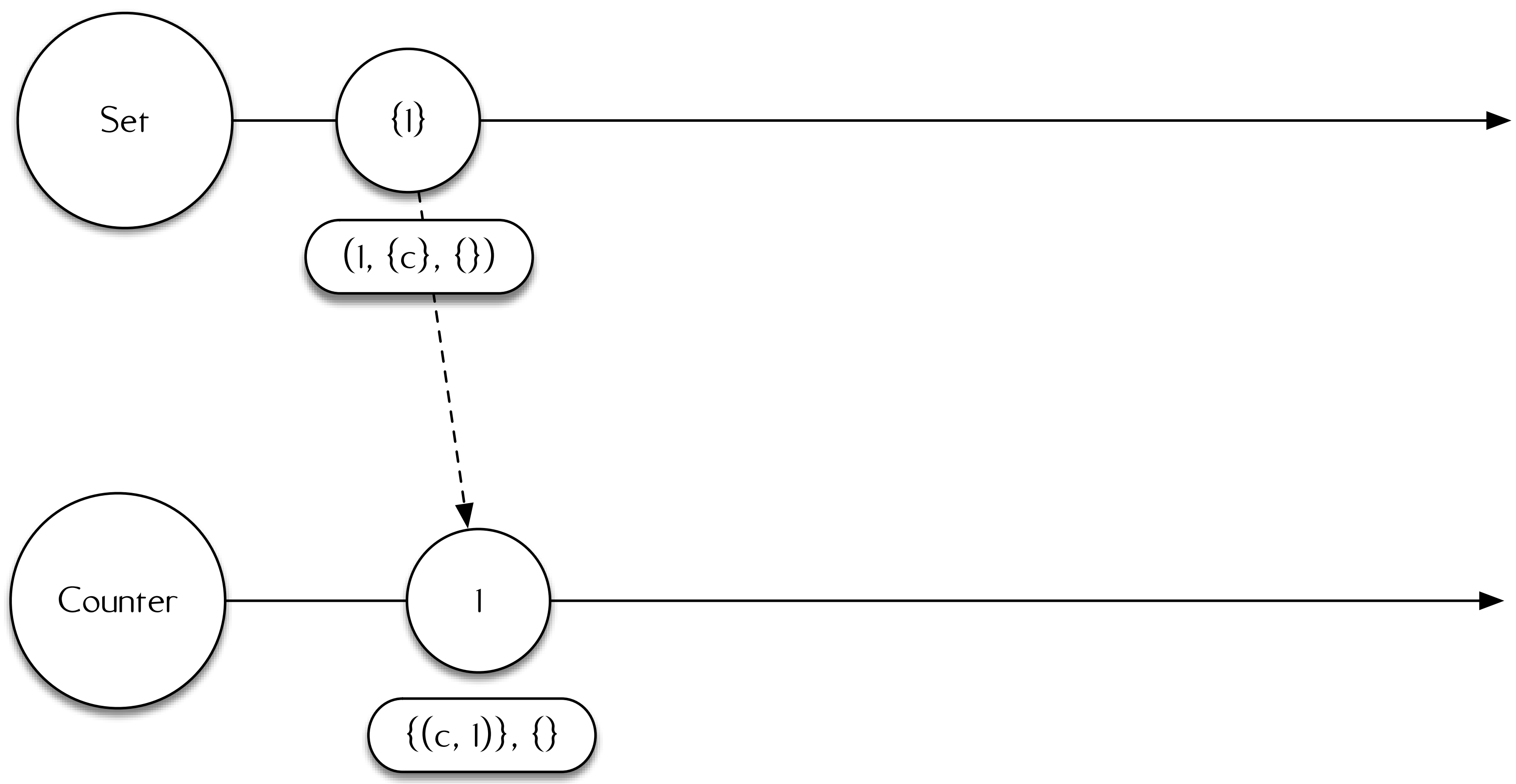
Instead of working to remove synchronization

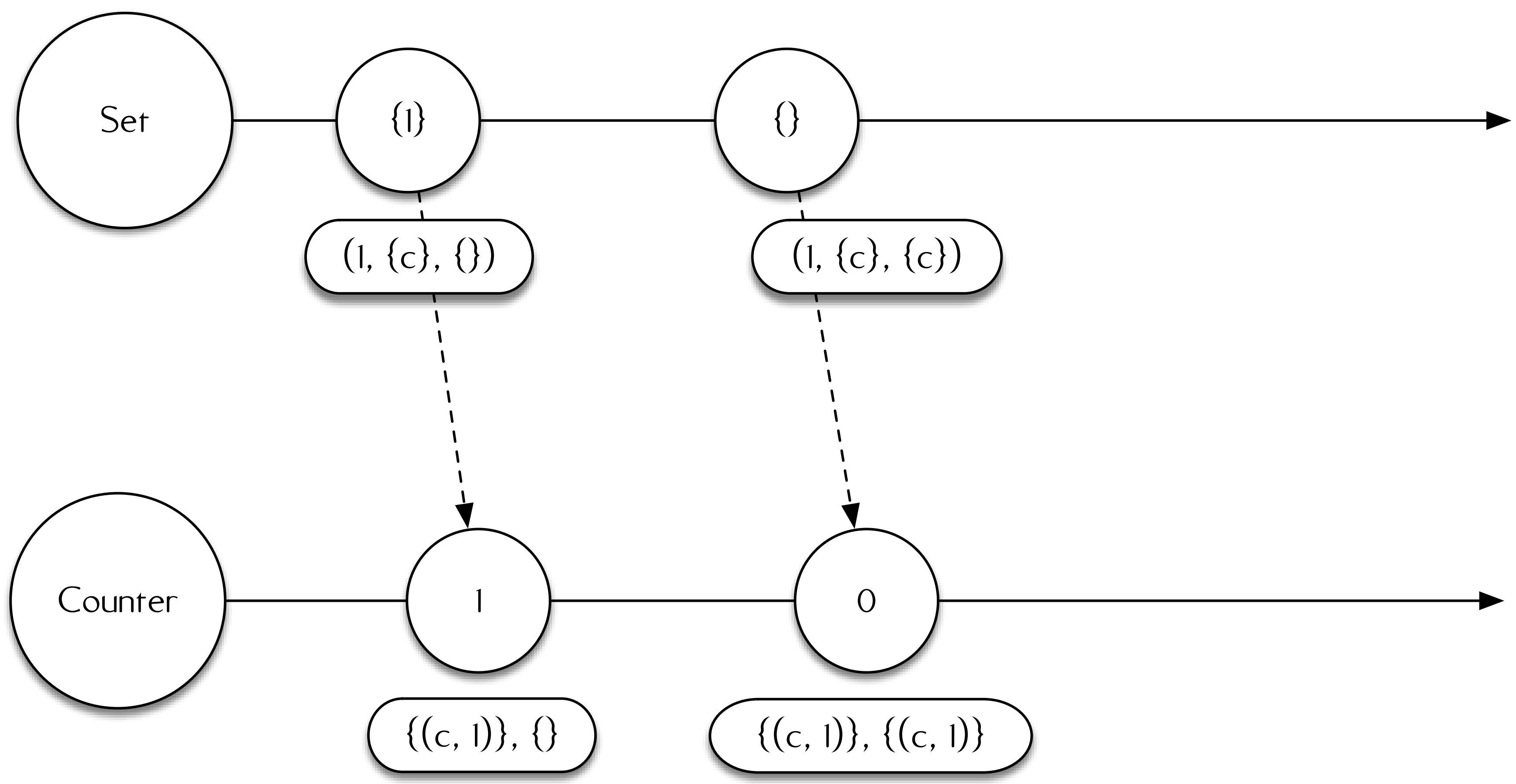
Challenges
Looking Ahead

Causality

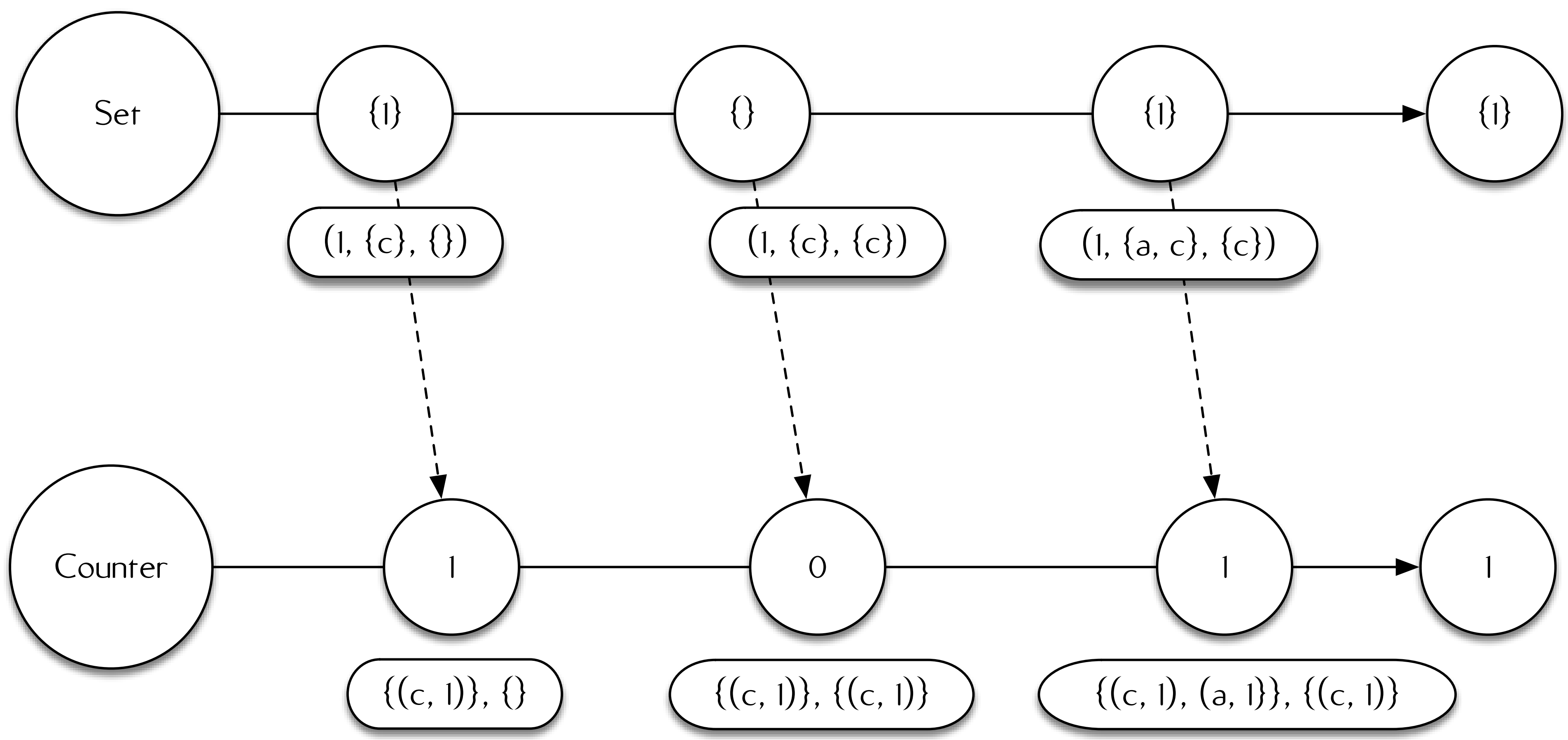
State Explosion


Set
Counter


Set
{1}
(1, {c}, {})
Counter
1
{(c, 1)}, {}



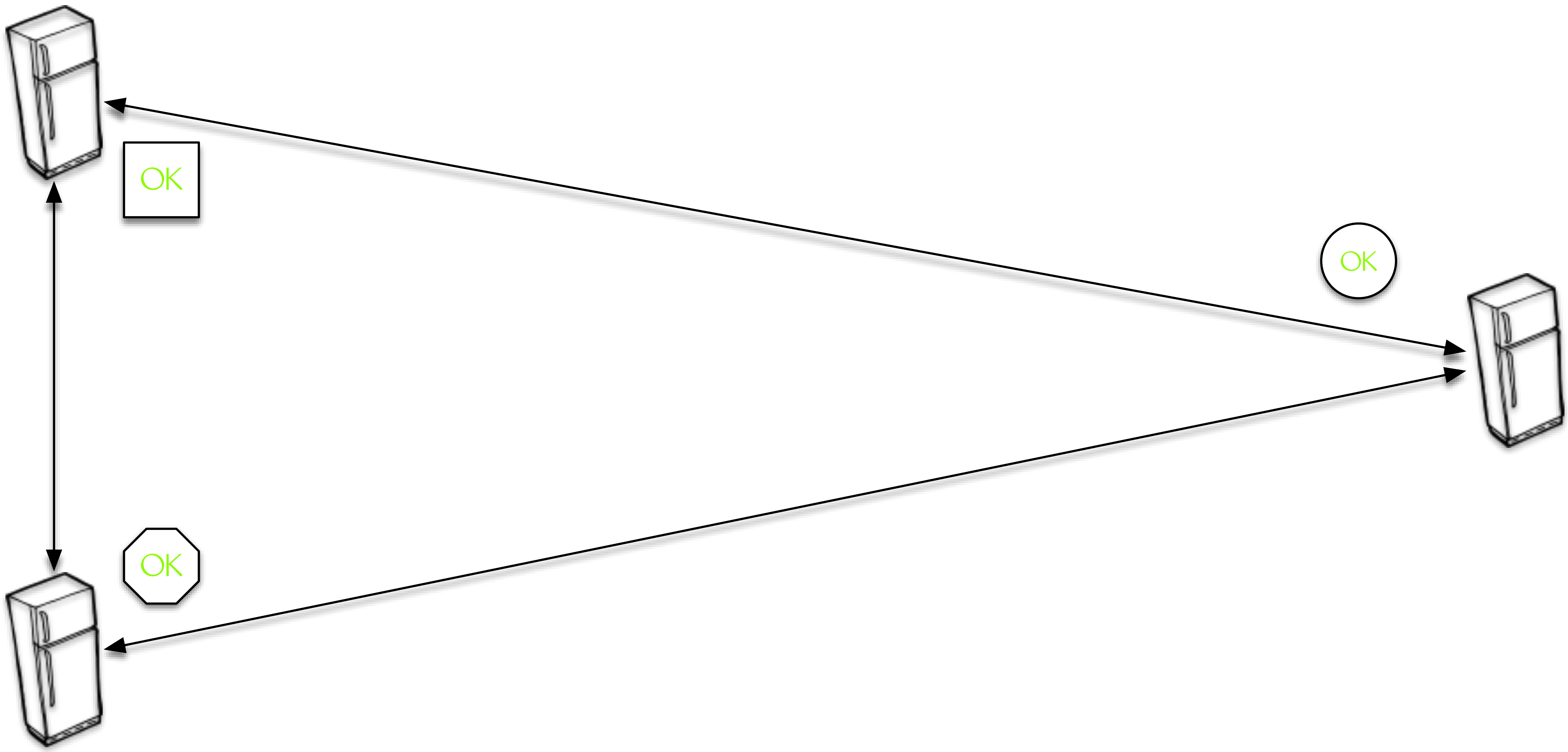
Set
{1}
{}
(1, {c}, {})
(1, {c}, {c})
Counter
1
0
{(c, 1)}, {}
{(c, 1)}, {(c, 1)}



Set
{1}
{}
{1}
{1}
(1, {c}, {})
(1, {c}, {c})
(1, {a, c}, {c})
Counter
1
0
1
1
{(c, 1)}, {}
{(c, 1)}, {(c, 1)}
{(c, 1), (a, 1)}, {(c, 1)}

Security

Computing at the Edge


OK
OK
OK

Computations

Expressiveness

How restrictive is a programming model where operations must be **associative**, **commutative**, and **idempotent**?

What's new?

Since Erlang Factory 2015

- Erlang 18, rebar3, Common Test
  Faster test suite, on Erlang 18 with rebar3

Since Erlang Factory 2015

- Erlang 18, rebar3, Common Test
  Faster test suite, on Erlang 18 with rebar3

- No Riak Core
  Plumtree distribution replaces the Riak Core distribution system

Since Erlang Factory 2015

- **Erlang 18, rebar3, Common Test**
  Faster test suite, on Erlang 18 with rebar3

- **No Riak Core**
  Plumtree distribution replaces the Riak Core distribution system

- **No NIFs**
  Allows us to cross-compile to other platforms and not worry about NIF scheduling

Semantics Improvements

- Delta-State Based Conflict-Free Replicated Data Types
  Optimized state dissemination by shipping deltas
  [Almeida *et al.*, 2016]

Semantics Improvements

- Delta-State Based Conflict-Free Replicated Data Types
  Optimized state dissemination by shipping deltas
  [Almeida *et al.*, 2016]

- Causal CRDTs
  Optimized, garbage-free data structure support
  [Almeida *et al.*, 2016; Meiklejohn 2016, in review]

Semantics Improvements

- Delta-State Based Conflict-Free Replicated Data Types
  Optimized state dissemination by shipping deltas
  [Almeida *et al.,* 2016]

- Causal CRDTs
  Optimized, garbage-free data structure support
  [Almeida *et al.,* 2016; Meiklejohn 2016, in review]

- Fold
  New semantics for a more expressive fold operation for arbitrary computation over sets
  [Meiklejohn 2016, in review]

Runtime Improvements

- Delta-State Based Anti-Entropy
  Optimized AAE mechanism based on deltas
  [Almeida *et al.*, 2016]

Runtime Improvements

- Delta-State Based Anti-Entropy
  Optimized AAE mechanism based on deltas
  [Almeida *et al.*, 2016]

- Mesos and Docker Enabled
  Run large-scale Lasp clusters on Mesos with Marathon
  [Meiklejohn and Yoo 2016, in review]

Runtime Improvements

- Delta-State Based Anti-Entropy
  Optimized AAE mechanism based on deltas
  [Almeida *et al.,* 2016]

- Mesos and Docker Enabled
  Run large-scale Lasp clusters on Mesos with Marathon
  [Meiklejohn and Yoo 2016, in review]

- Loquat
  Epidemic broadcast, partially replicated with
  Decentralized Information Flow Control
  [Meiklejohn 2016, in review]

Lasp Simulator

- Docker Containers
  Docker containers for EPMD and Lasp runtime system

Lasp Simulator

- Docker Containers
  Docker containers for EPMD and Lasp runtime system

- Service Discovery
  Mechanisms for clustering Erlang nodes based on either Marathon application definitions of Mesos-DNS

Lasp Simulator

- Docker Containers
  Docker containers for EPMD and Lasp runtime system

- Service Discovery
  Mechanisms for clustering Erlang nodes based on either Marathon application definitions of Mesos-DNS

- Instrumentation
  Transmission instrumentation and divergence measurement

Lasp Simulator

- Plumtree VM-to-VM
  VM-to-VM communication performed using the Plumtree epidemic broadcast protocol

Lasp Simulator

- Plumtree VM-to-VM
  VM-to-VM communication performed using the Plumtree epidemic broadcast protocol

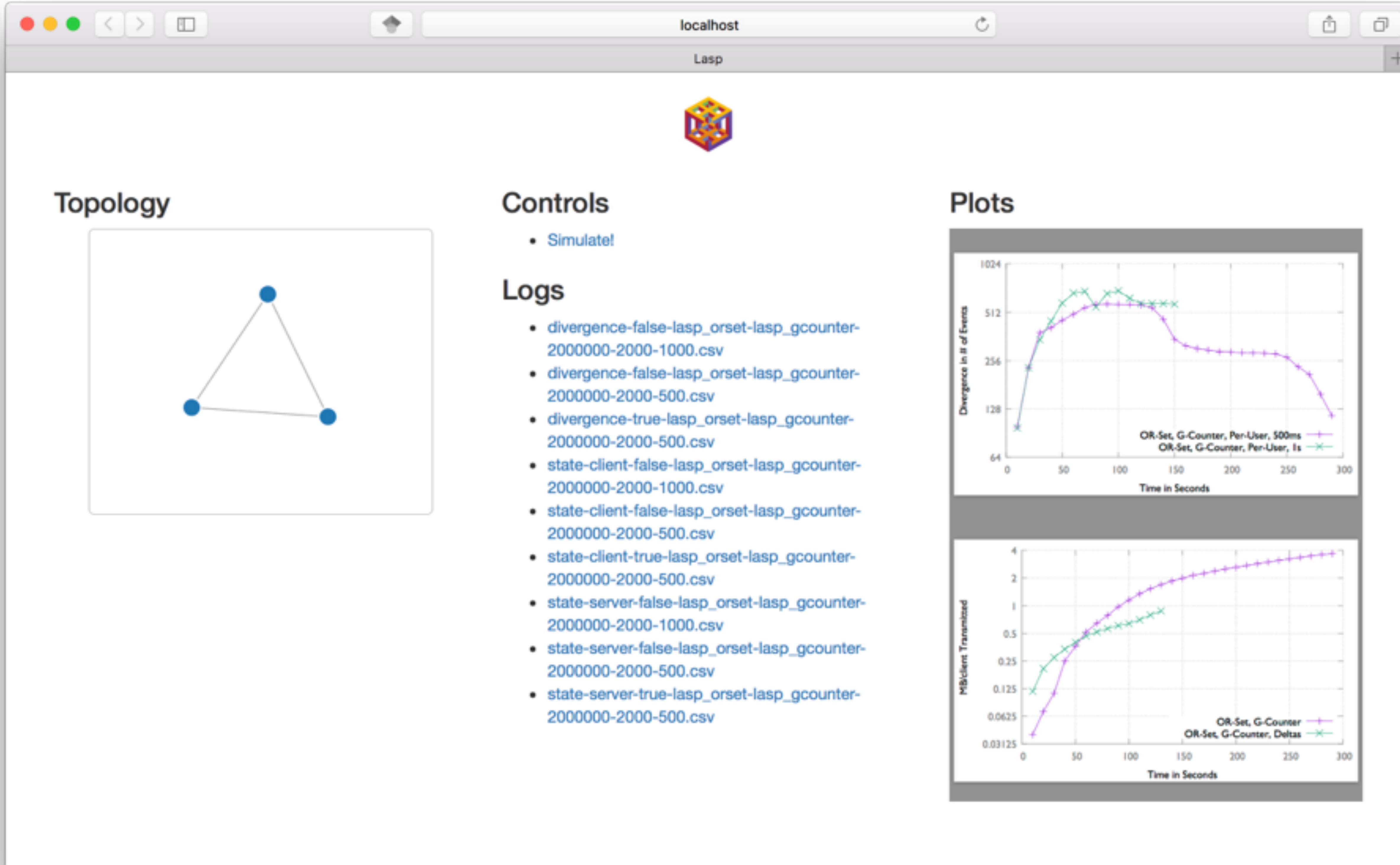
- Clients-as-processes
  Multiple clients per virtual machine, acting as mobile/IoT devices that periodically simulate and can be partitioned

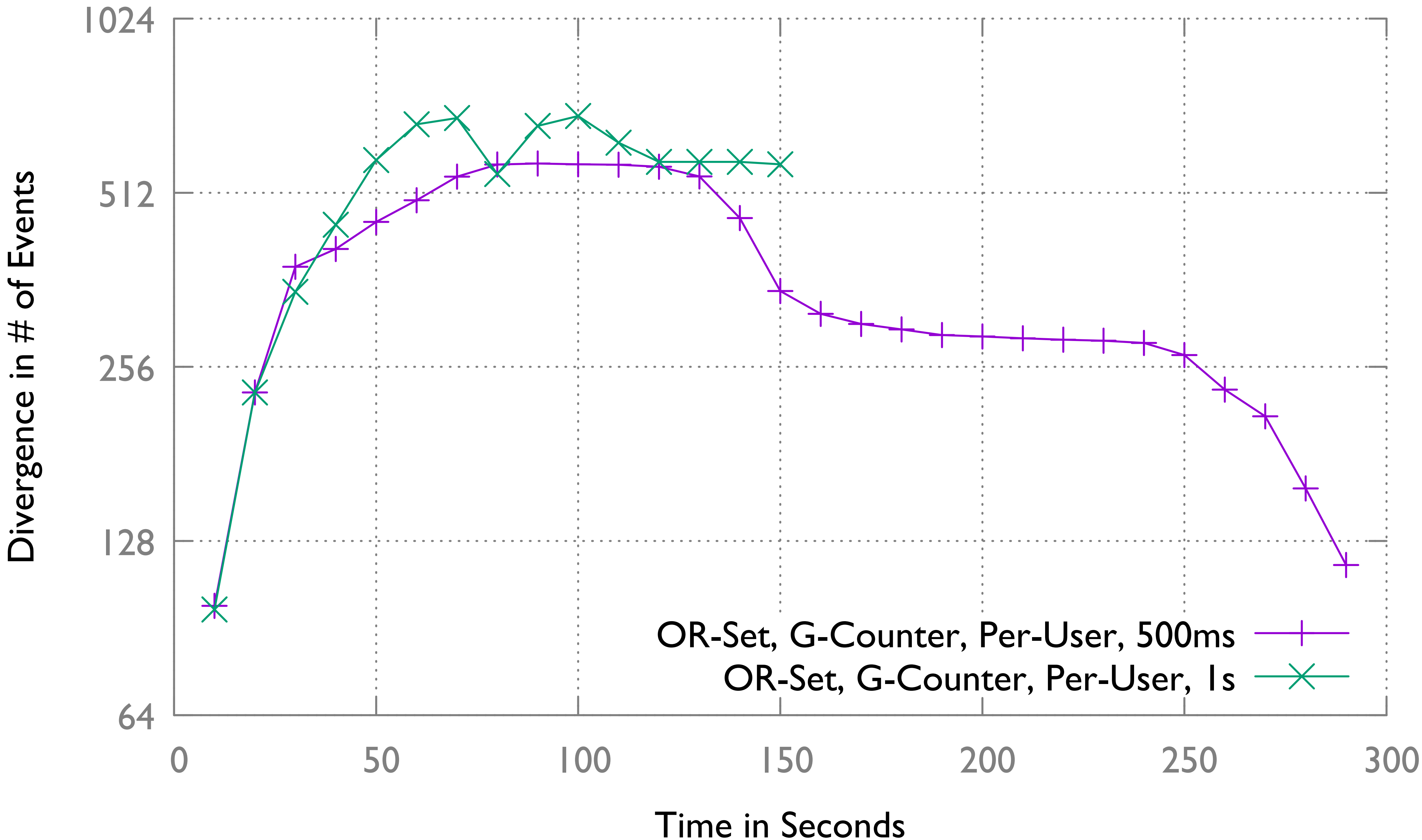
Lasp Simulator

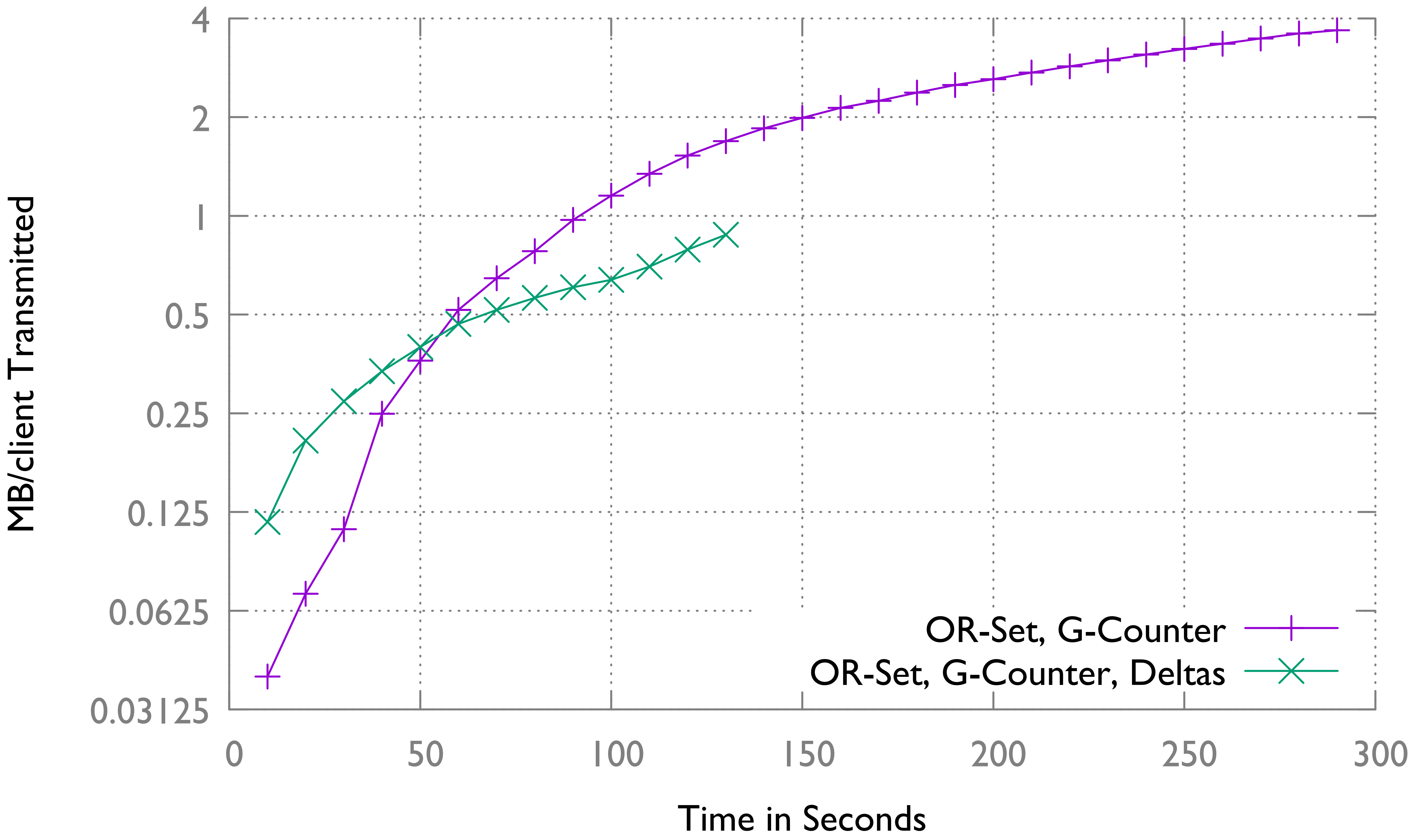
- **Plumtree VM-to-VM**
  VM-to-VM communication performed using the **Plumtree** epidemic broadcast protocol

- **Clients-as-processes**
  Multiple clients per virtual machine, acting as mobile/IoT devices that periodically simulate and can be partitioned

- **“Design in the small, run in the large”**
  Runtime configuration change for client-to-VM ratio, allows for **single laptop design of multi-machine evaluations** of the programming model


localhost
Lasp
Topology
Controls
Simulate!
Logs
divergence-false-lasp_orset-lasp_gcounter-2000000-2000-1000.csv
divergence-false-lasp_orset-lasp_gcounter-2000000-2000-500.csv
divergence-true-lasp_orset-lasp_gcounter-2000000-2000-500.csv
state-client-false-lasp_orset-lasp_gcounter-2000000-2000-1000.csv
state-client-false-lasp_orset-lasp_gcounter-2000000-2000-500.csv
state-client-true-lasp_orset-lasp_gcounter-2000000-2000-500.csv
state-server-false-lasp_orset-lasp_gcounter-2000000-2000-1000.csv
state-server-false-lasp_orset-lasp_gcounter-2000000-2000-500.csv
state-server-true-lasp_orset-lasp_gcounter-2000000-2000-500.csv
Plots
Divergence in # of Events
Time in Seconds
OR-Set, G-Counter, Per-User, 500ms
OR-Set, G-Counter, Per-User, 1s
MB/client Transmitted
Time in Seconds
OR-Set, G-Counter
OR-Set, G-Counter, Deltas


1024
512
256
128
64
0
50
100
150
200
250
300
Divergence in # of Events
Time in Seconds
OR-Set, G-Counter, Per-User, 500ms
OR-Set, G-Counter, Per-User, 1s



4
2
1
0.5
0.25
0.125
0.0625
0.03125
MB/client Transmitted
0
50
100
150
200
250
300
Time in Seconds
OR-Set, G-Counter
OR-Set, G-Counter, Deltas

What's next?

Google Summer of Code (and my Ph.D.!)

- Partial Evaluation
  Optimize execution based on analysis and annotations where we can determine local-vs-remote usage

Google Summer of Code
(and my Ph.D.!)

- Partial Evaluation
  Optimize execution based on analysis and annotations where we can determine local-vs-remote usage

- Optimizations
  General optimizations of the Erlang implementation of Lasp to improve performance of the runtime system

Google Summer of Code (and my Ph.D.!)

- Partial Evaluation
  Optimize execution based on analysis and annotations where we can determine local-vs-remote usage

- Optimizations
  General optimizations of the Erlang implementation of Lasp to improve performance of the runtime system

- Elixir / Macros
  Provide a nicer way for working with Lasp, outside of the current syntax.

How do I learn more?

Publications

- “Lasp: A Language for Distributed, Coordination-Free Programming”
  ACM SIGPLAN PPDP 2015
- “Selective Hearing: An Approach to Distributed, Eventually Consistent Edge Computation”
  IEEE W-PSDS 2015
- “The Implementation and Use of a Generic Dataflow Behaviour in Erlang”
  ACM SIGPLAN Erlang Workshop ’15
- “Lasp: A Language for Distributed, Eventually Consistent Computations with CRDTs"
  PaPoC 2015
- “Declarative, Sliding Window Aggregations for Computations at the Edge"
  IEEE EdgeCom 2016

Thanks!

Christopher Meiklejohn
@cmeik
http://www.lasp-lang.org