Declarative,
Secure,
Convergent
Edge Computation

Christopher Meiklejohn
Université catholique de Louvain, Belgium

Internet of Things

Internet of Things

but, more generally…

Edge Computation

- Logical extremes
  Pushing both computation and data to the logical extremes of the network

Edge Computation

- **Logical extremes**
  Pushing both computation and data to the logical extremes of the network

- **Arbitrary computation**
  Support arbitrary computations regardless of location of data in the network

Edge Computation

- Logical extremes
  Pushing both computation and data to the logical extremes of the network

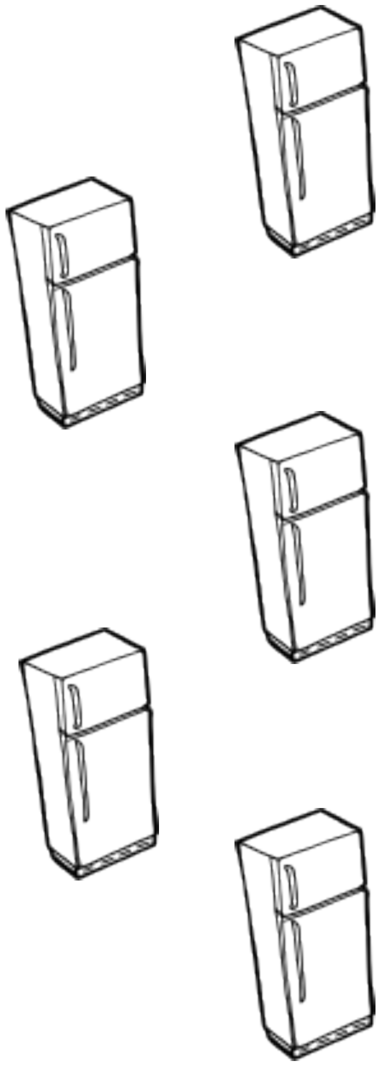
- Arbitrary computation
  Support arbitrary computations regardless of location of data in the network

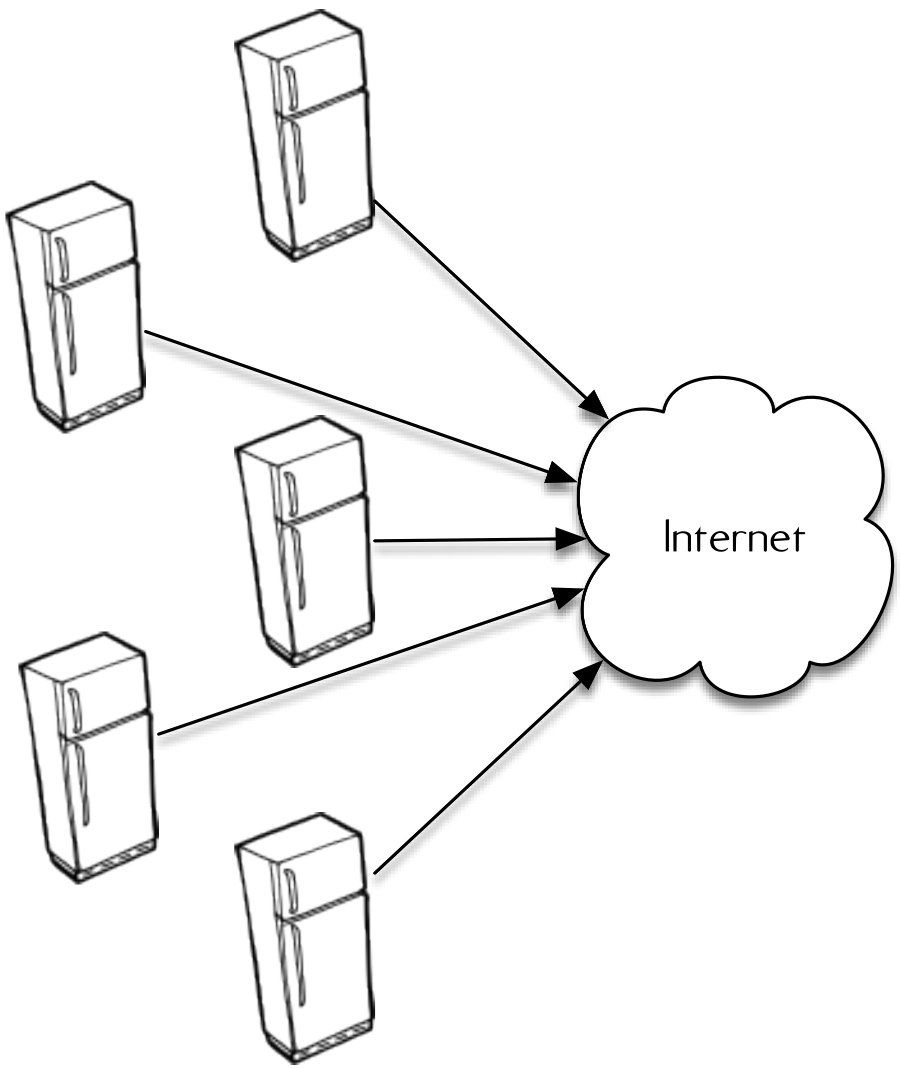
- Self-organizing, resilient
  Directed diffusion, Cornell circa-1990; self-organizing systems that coordinate to complete computations

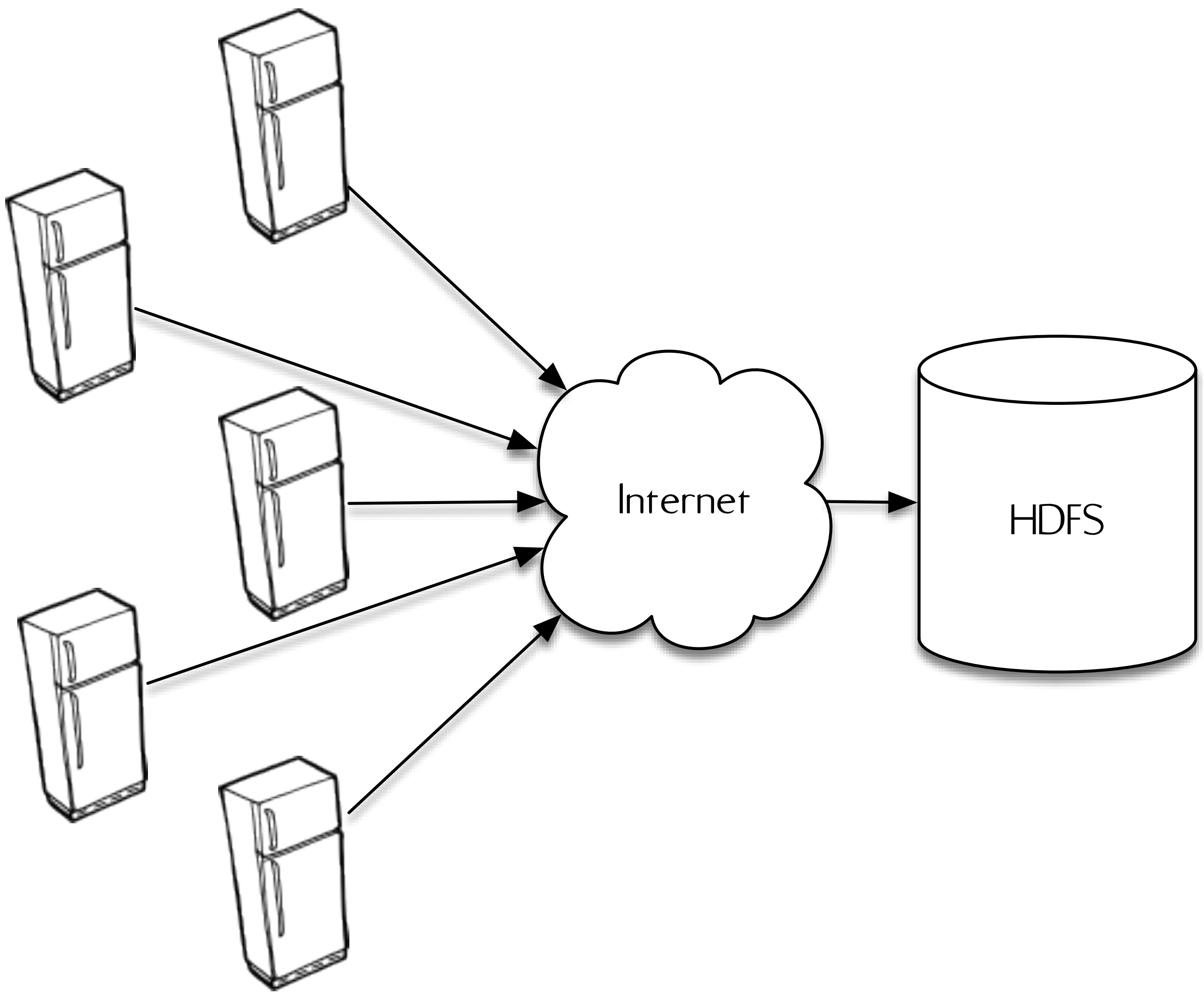
Example Application

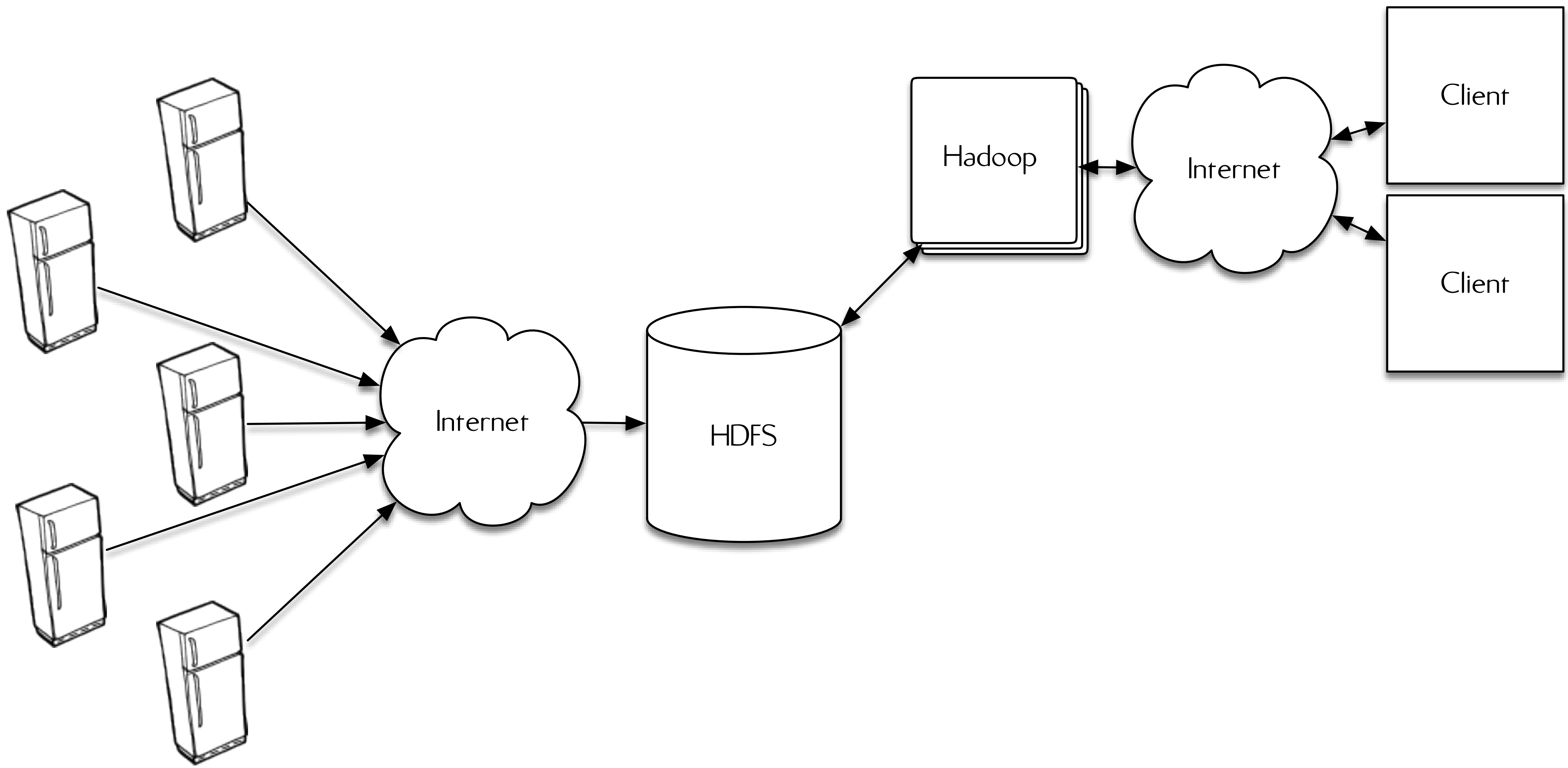
Hospital Refrigerators

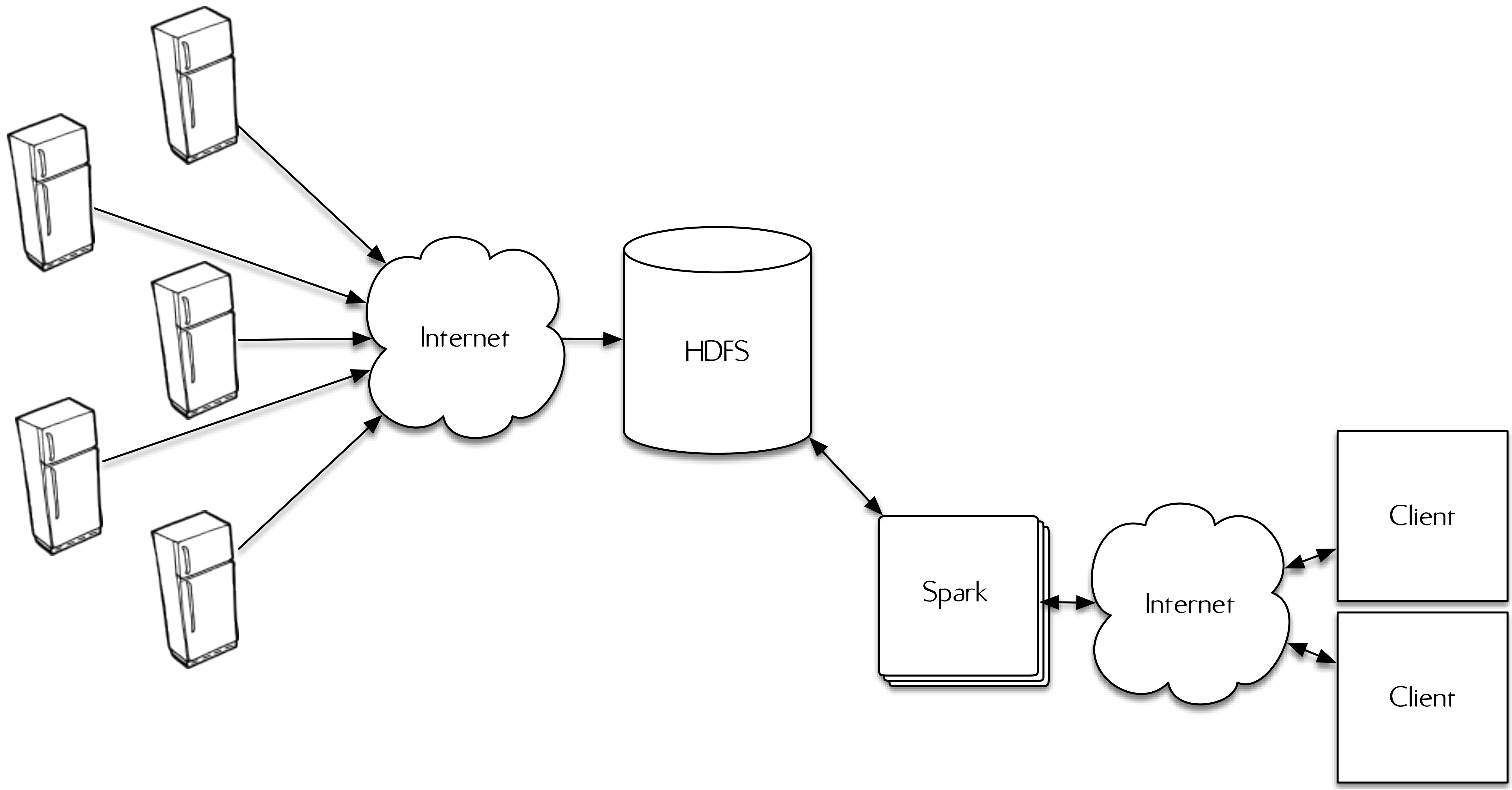
Hospital Refrigerators
Typical Topology



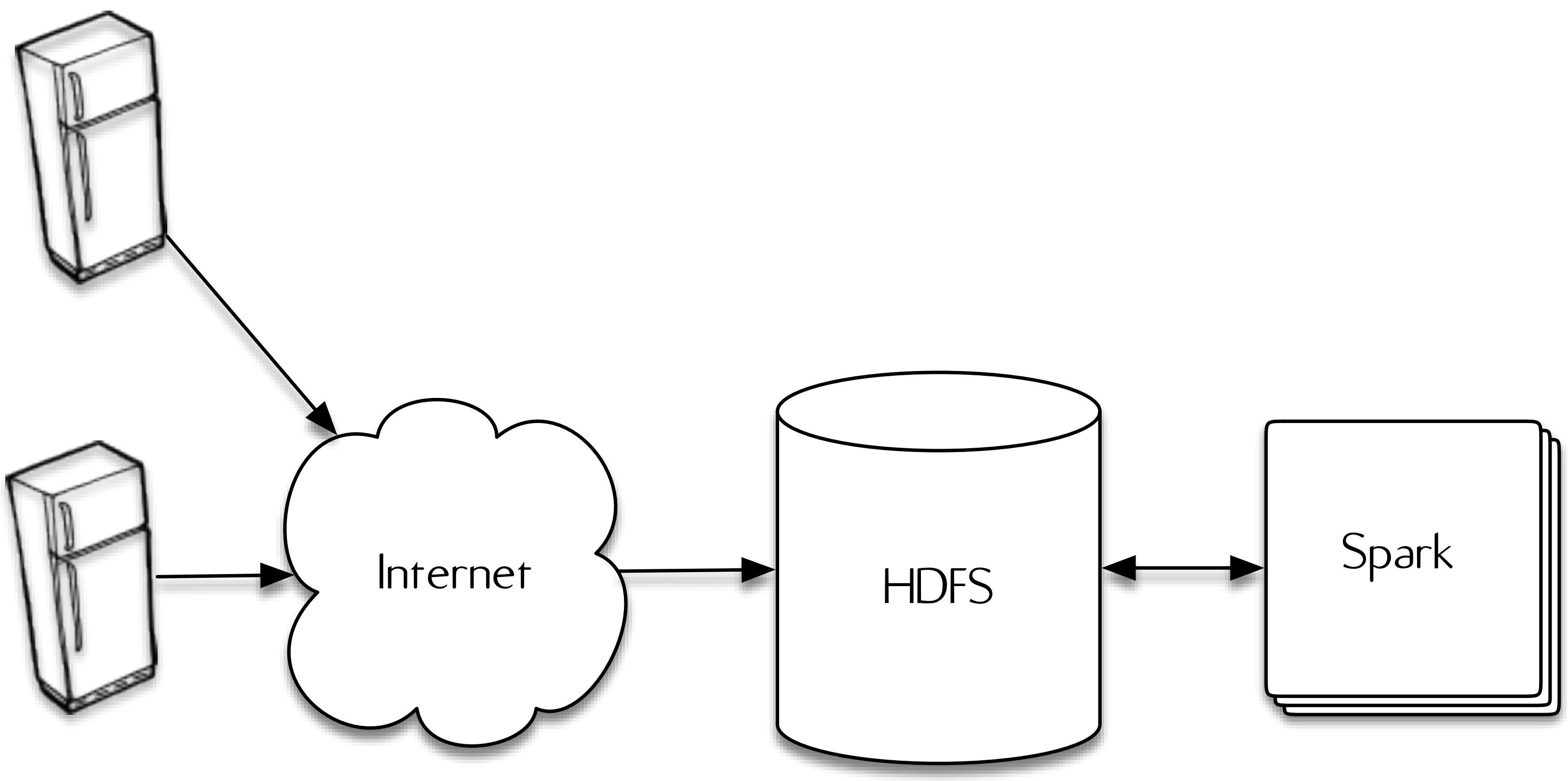

Internet

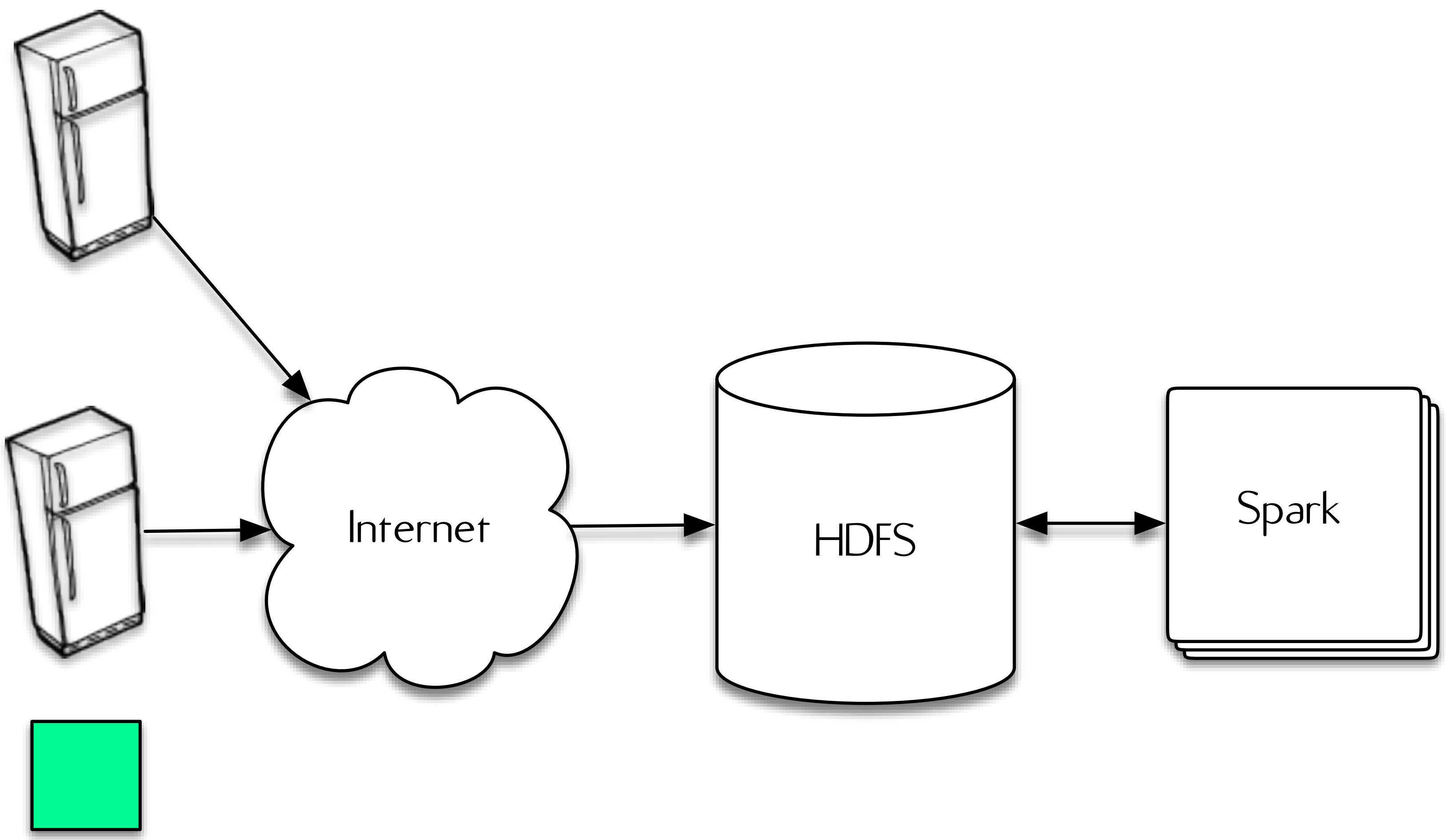

Internet
HDFS

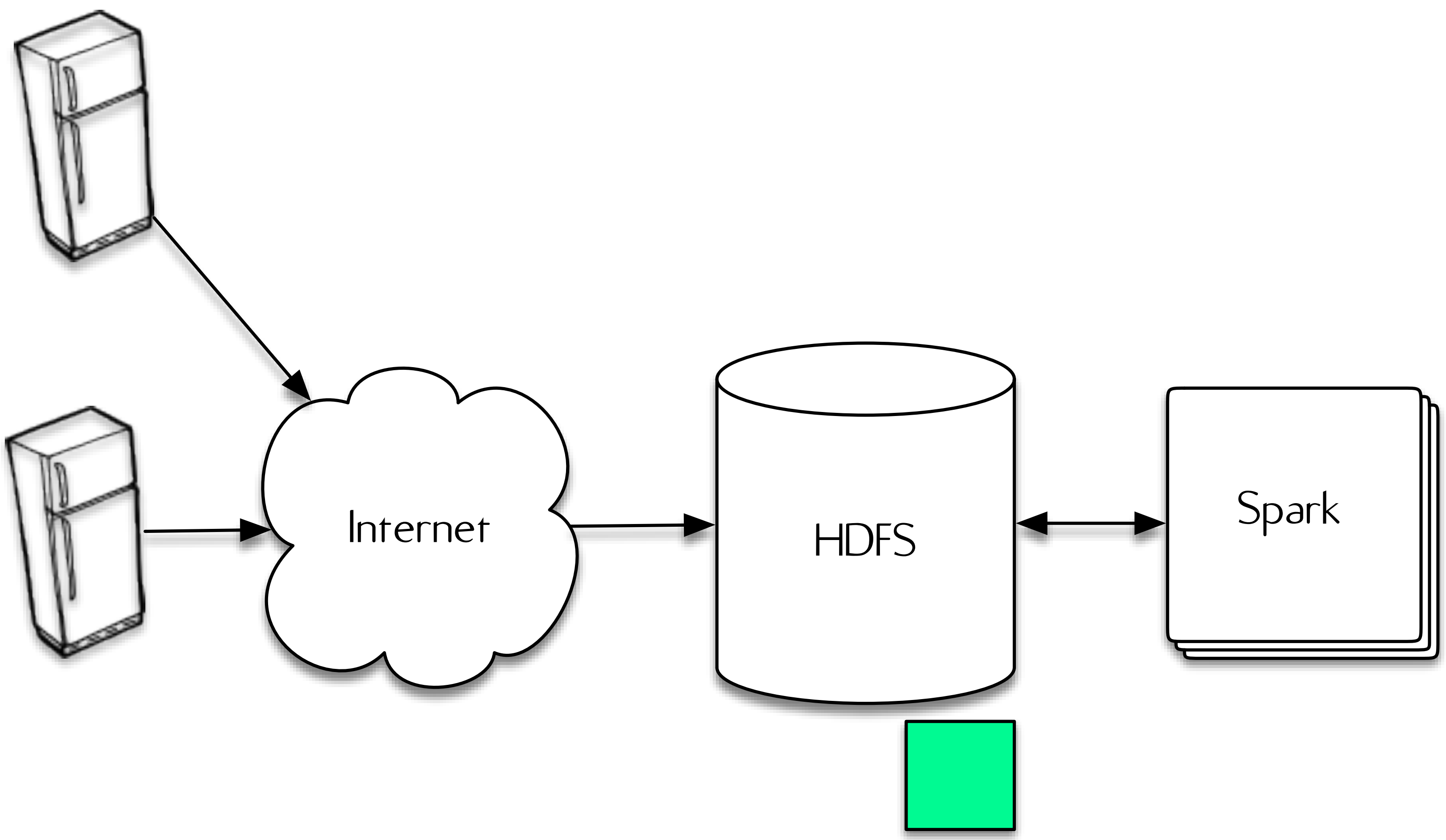

Hadoop
Internet
Client
Client
Internet
HDFS

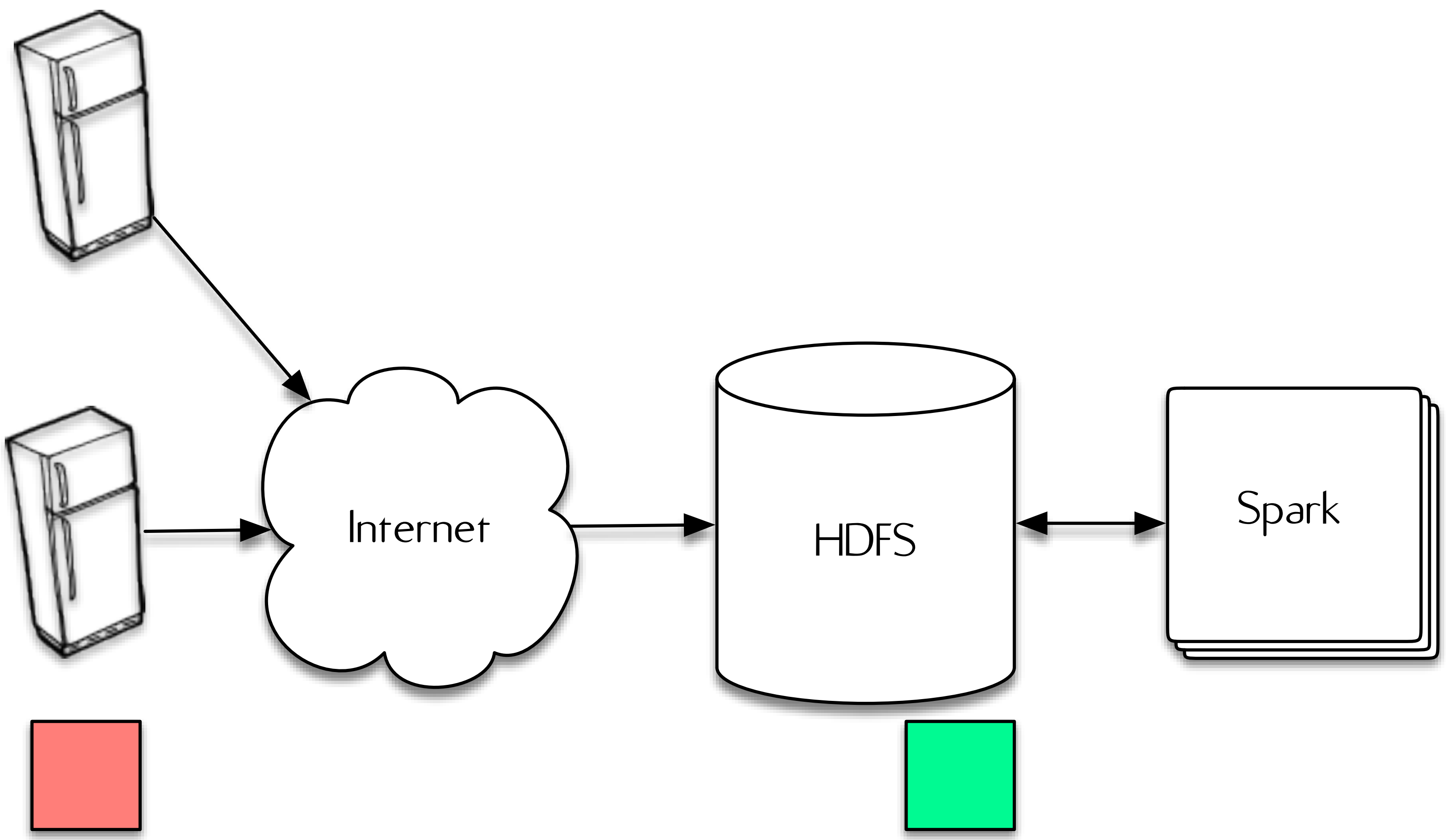

Internet
HDFS
Spark
Internet
Client
Client

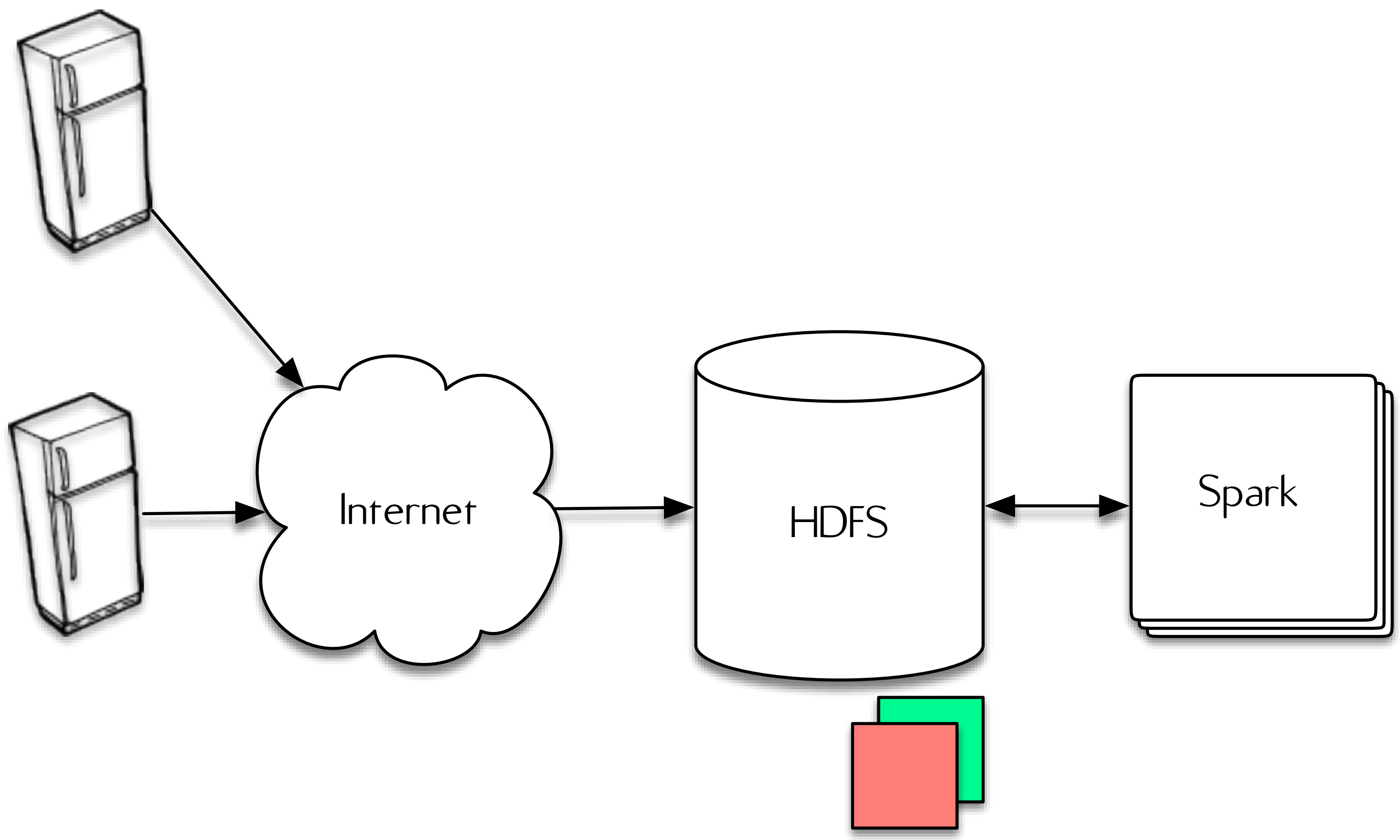
Hospital Refrigerators
Ideal Execution

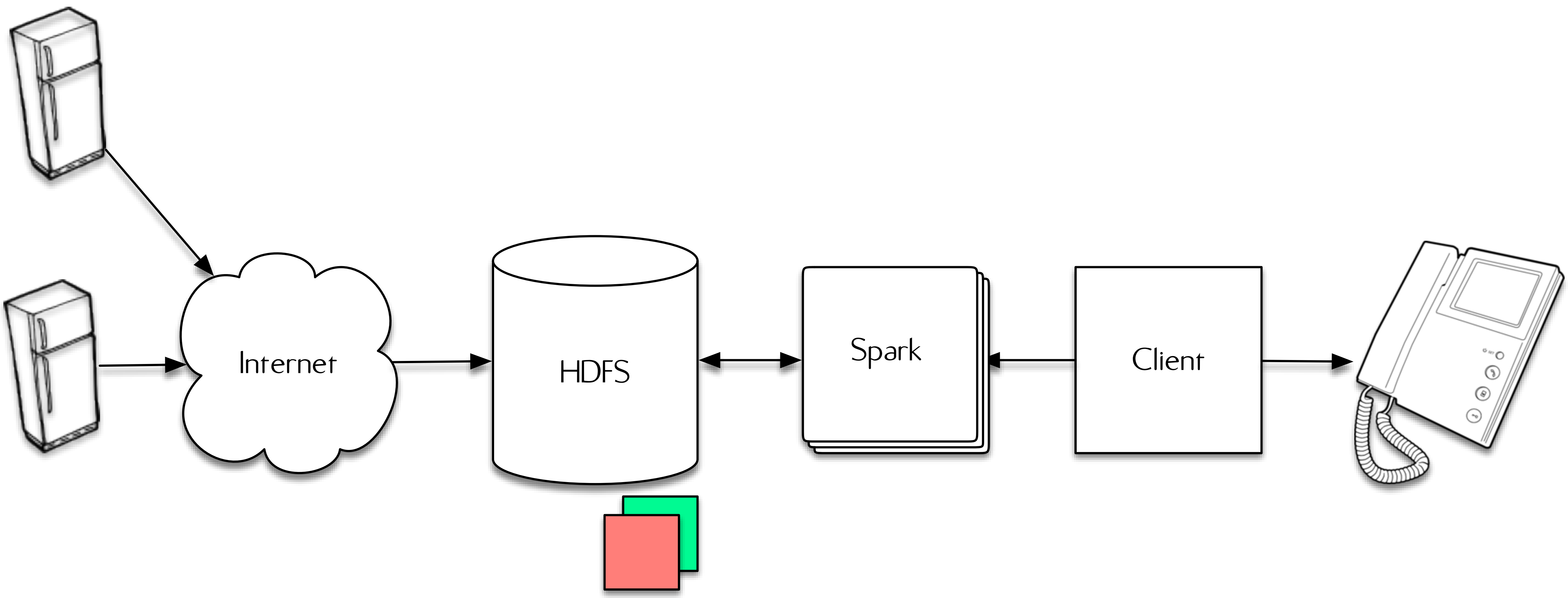

Internet
HDFS
Spark


Internet
HDFS
Spark


Internet
HDFS
Spark

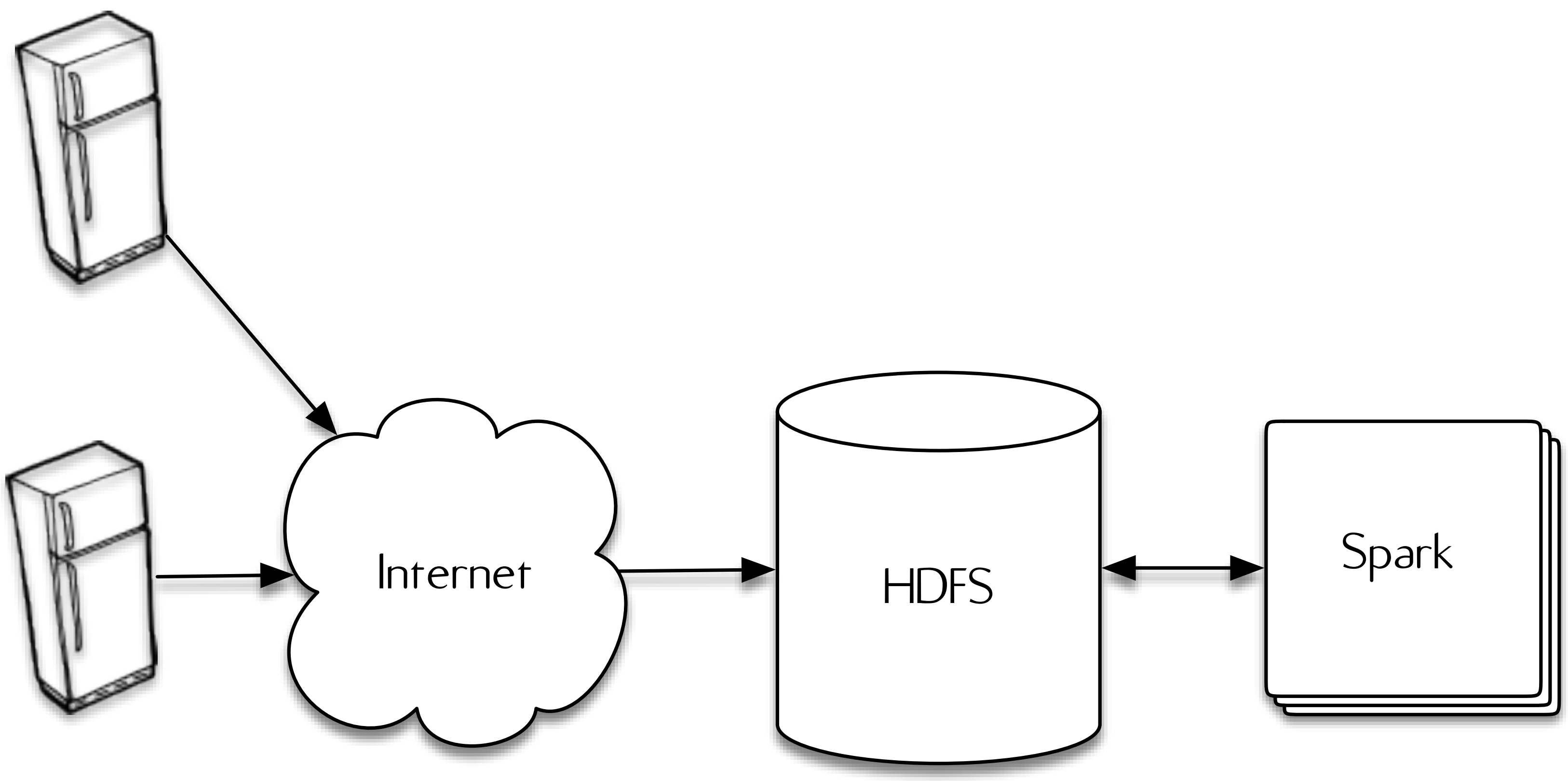

Internet
HDFS
Spark

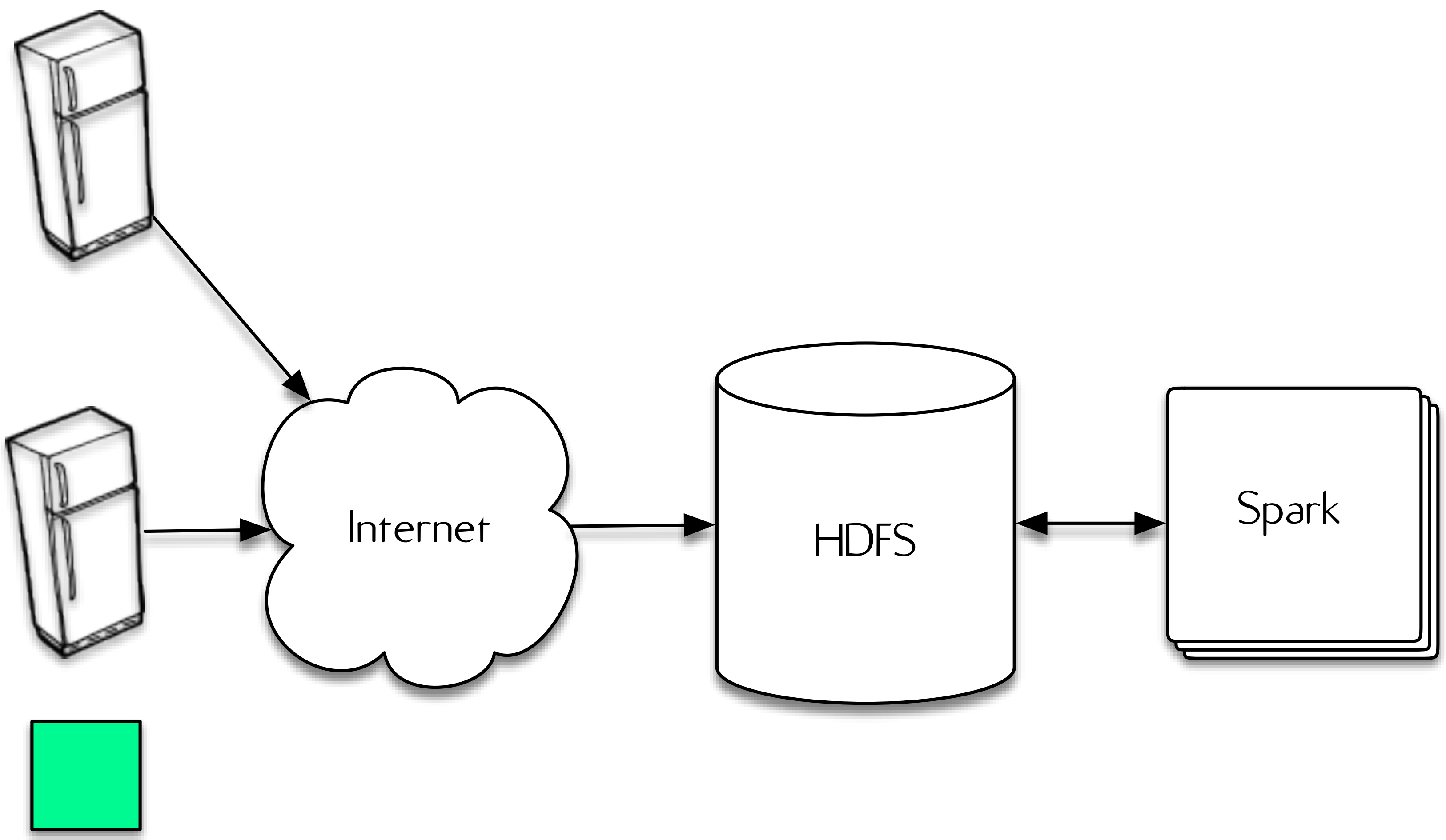

Internet
HDFS
Spark

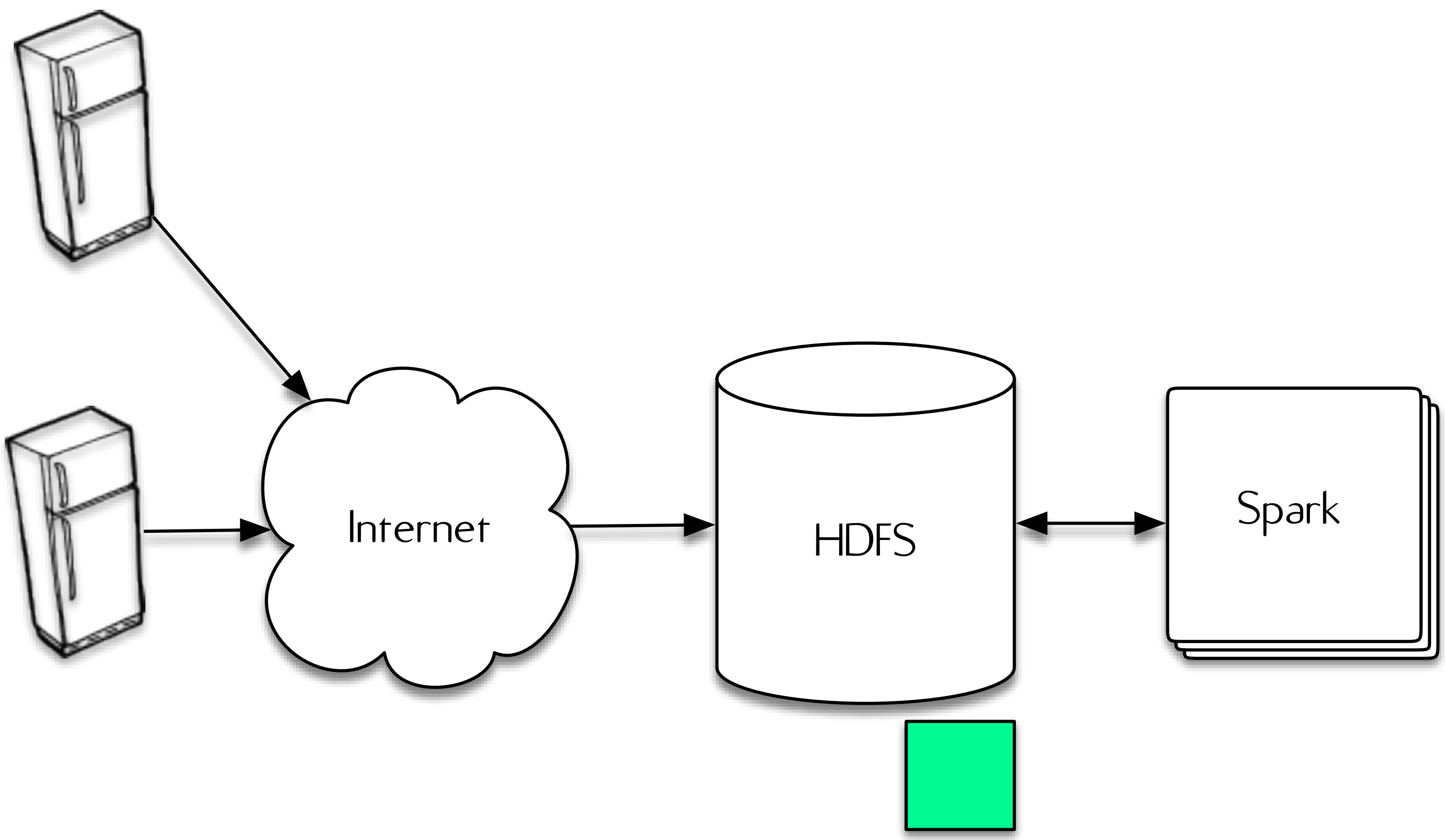

Internet
HDFS
Spark
Client

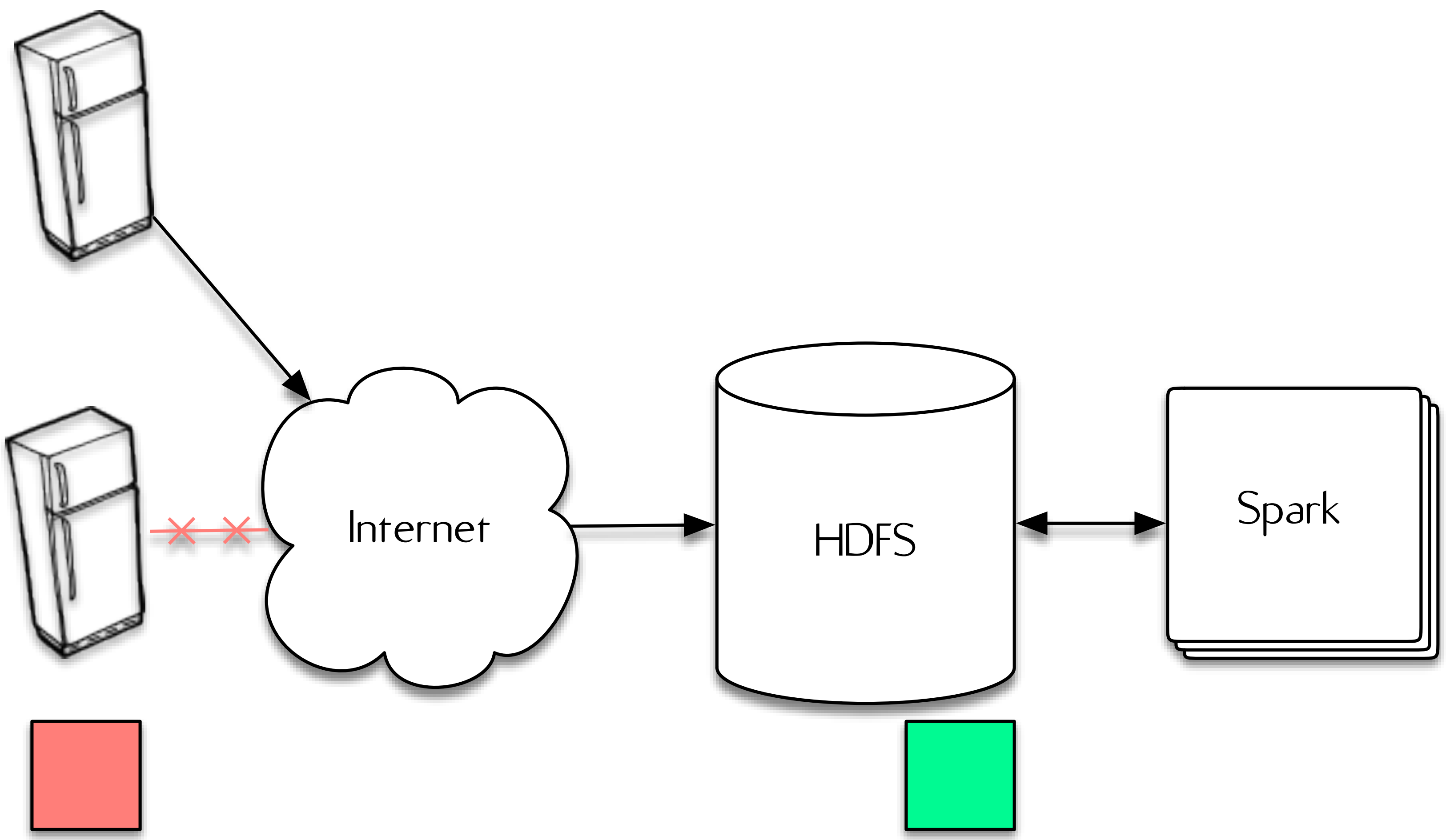
Problem

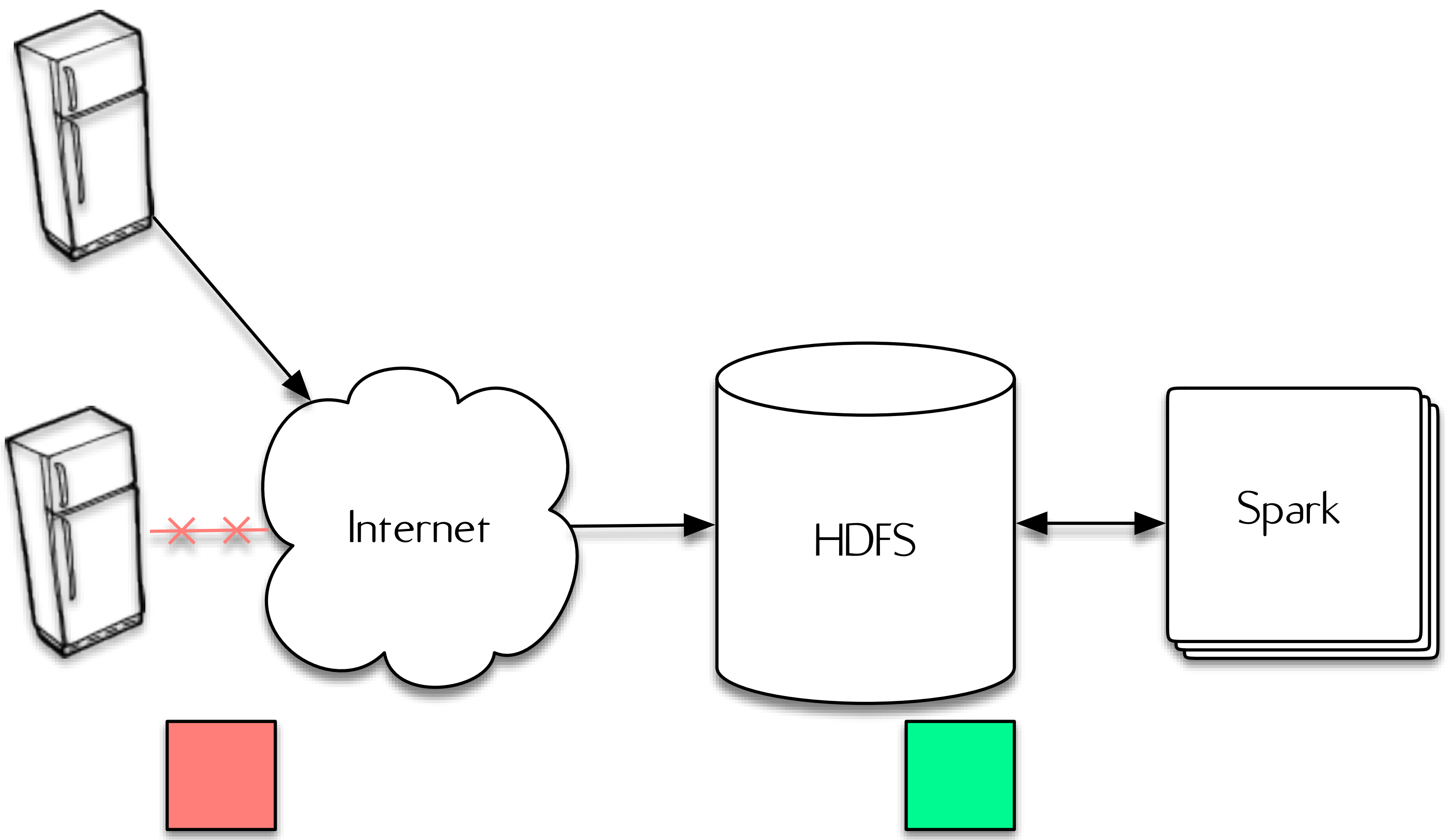
Connectivity

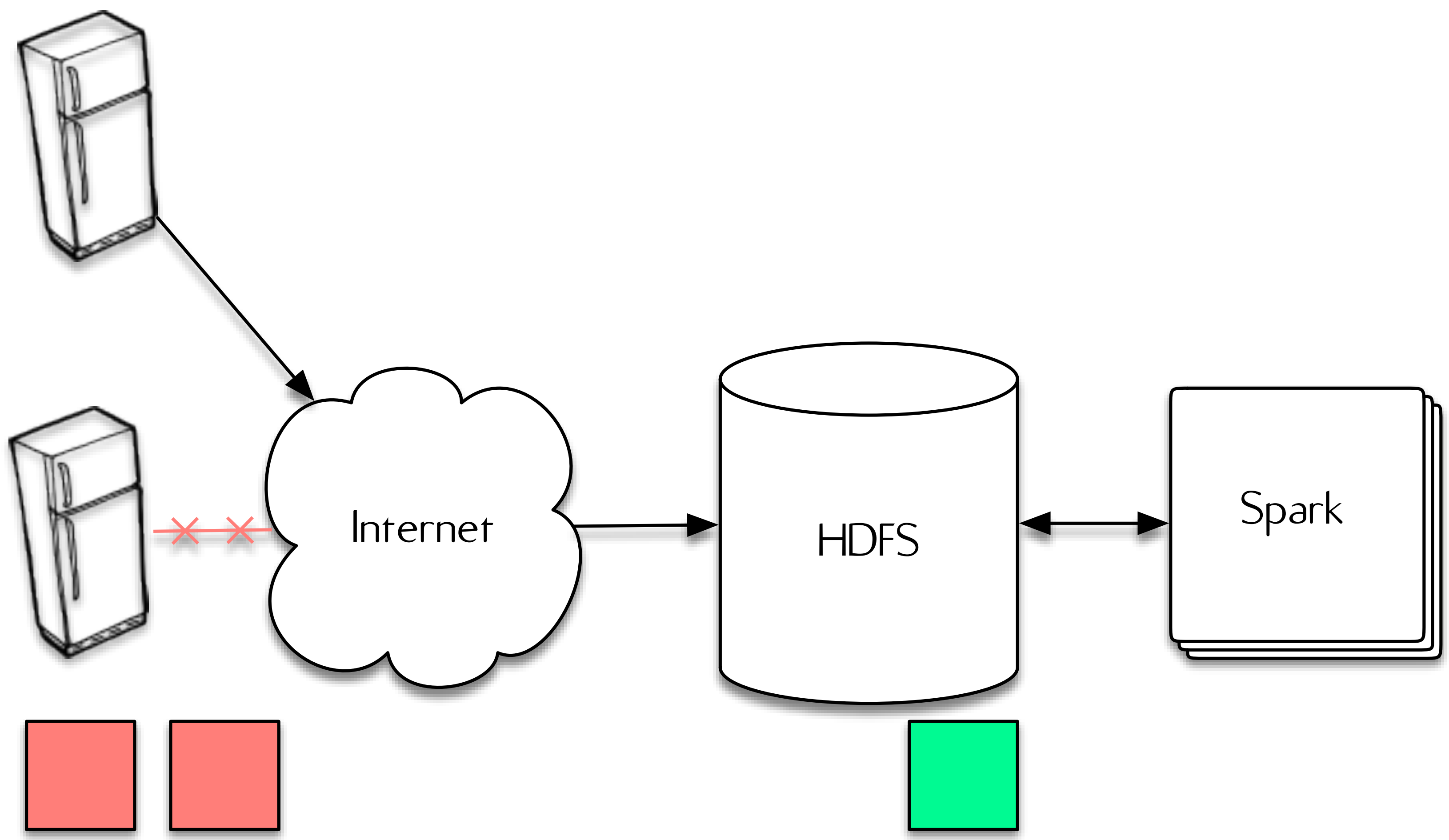

Internet
HDFS
Spark

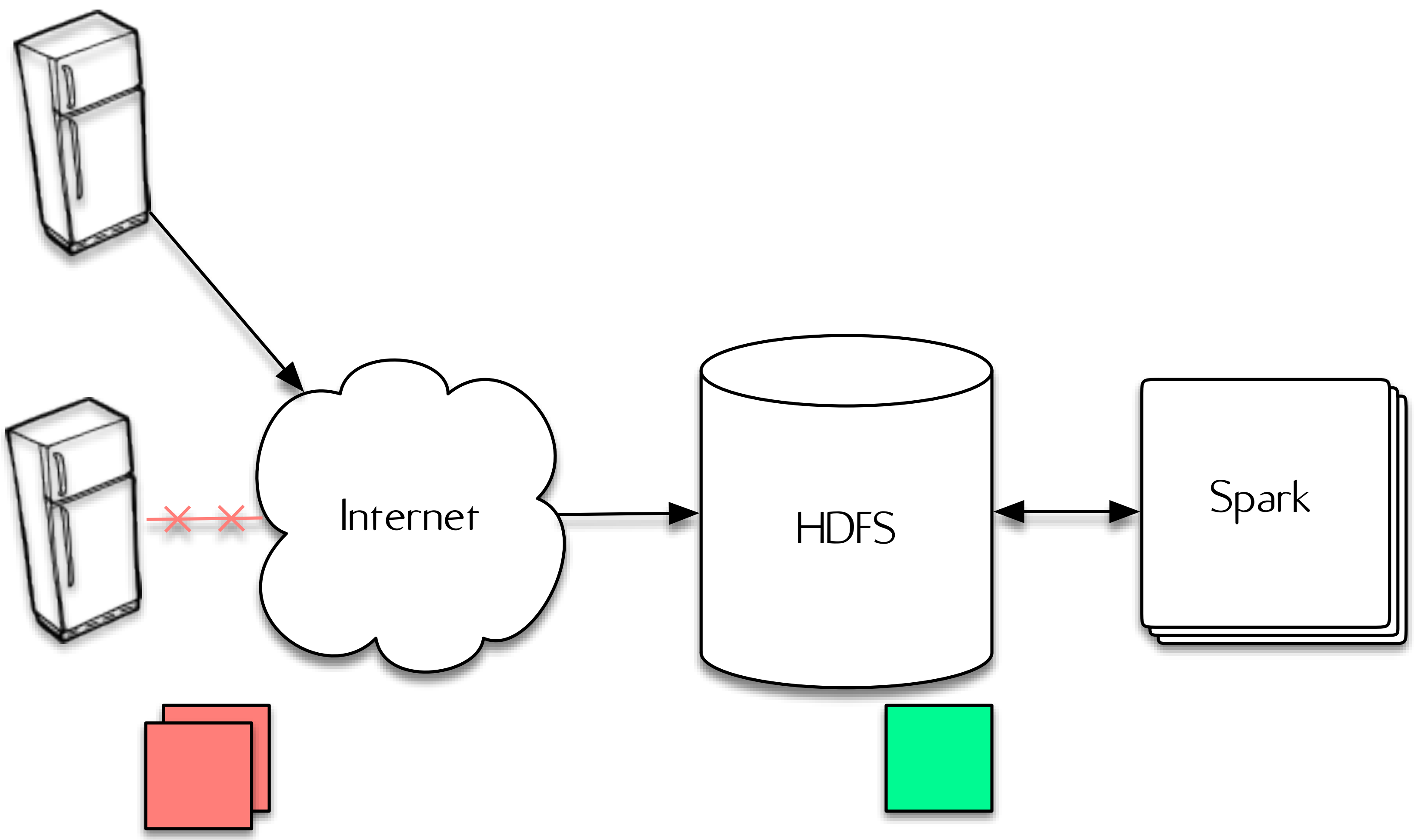

Internet
HDFS
Spark


Internet
HDFS
Spark

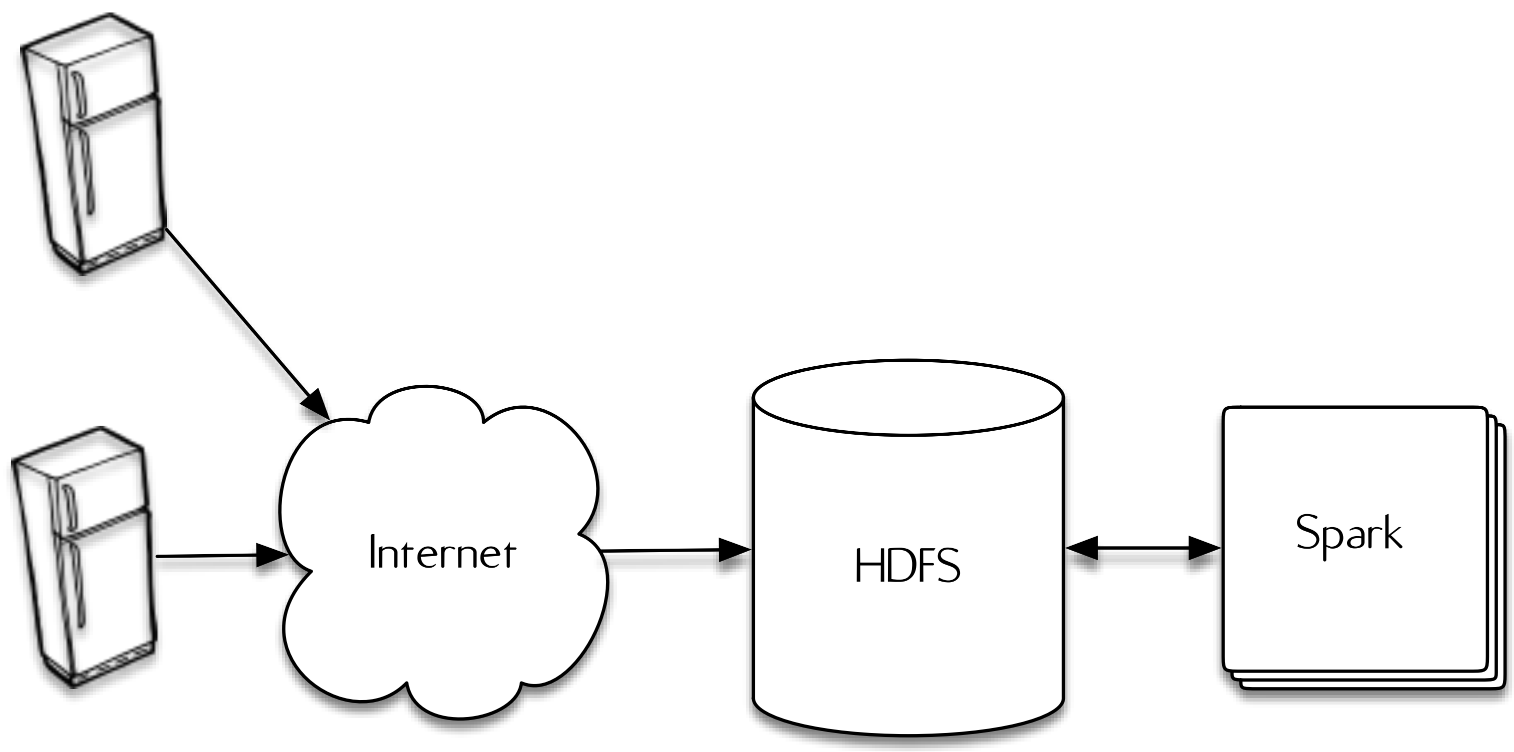

Internet
HDFS
Spark

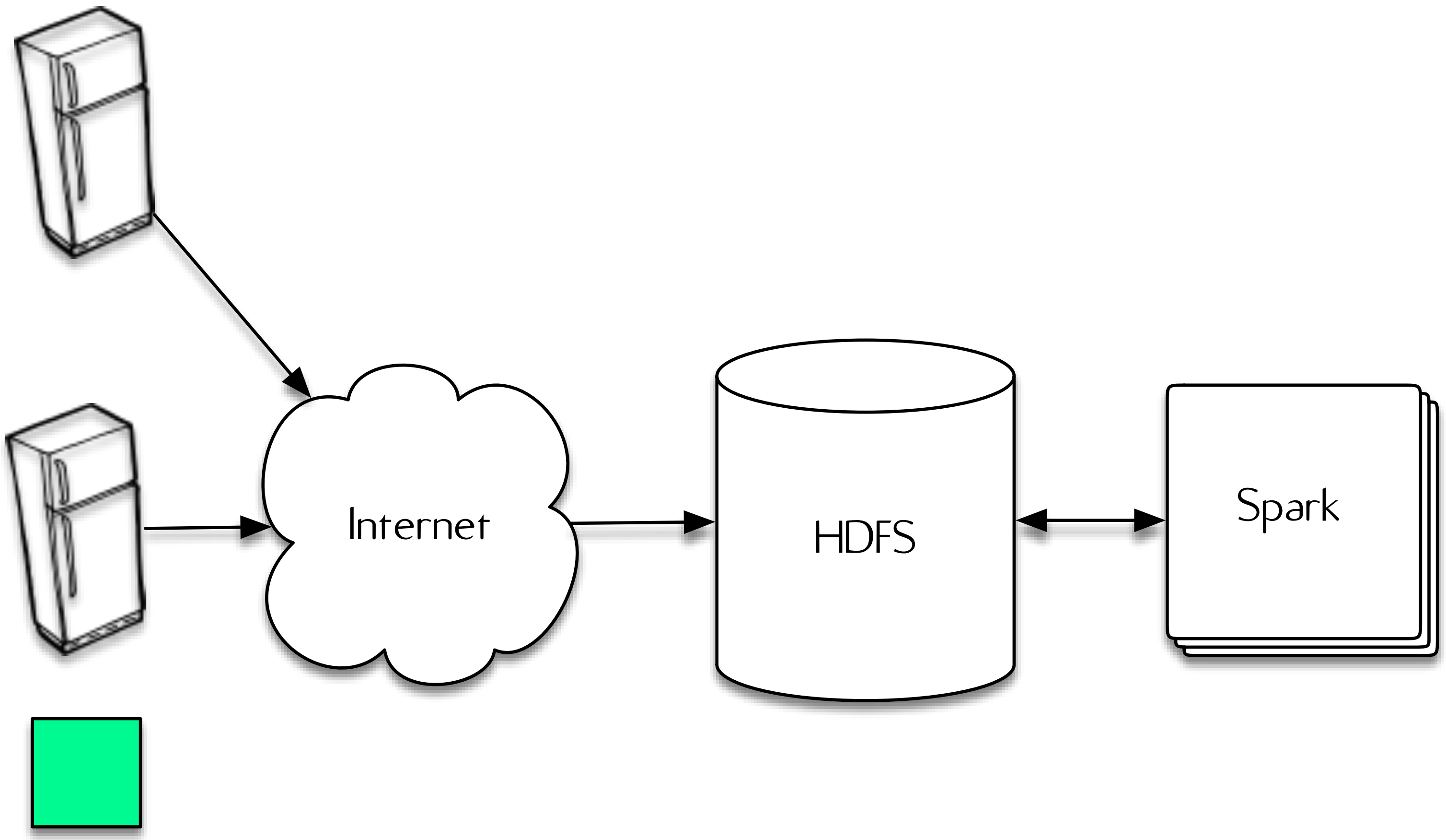

Internet
HDFS
Spark

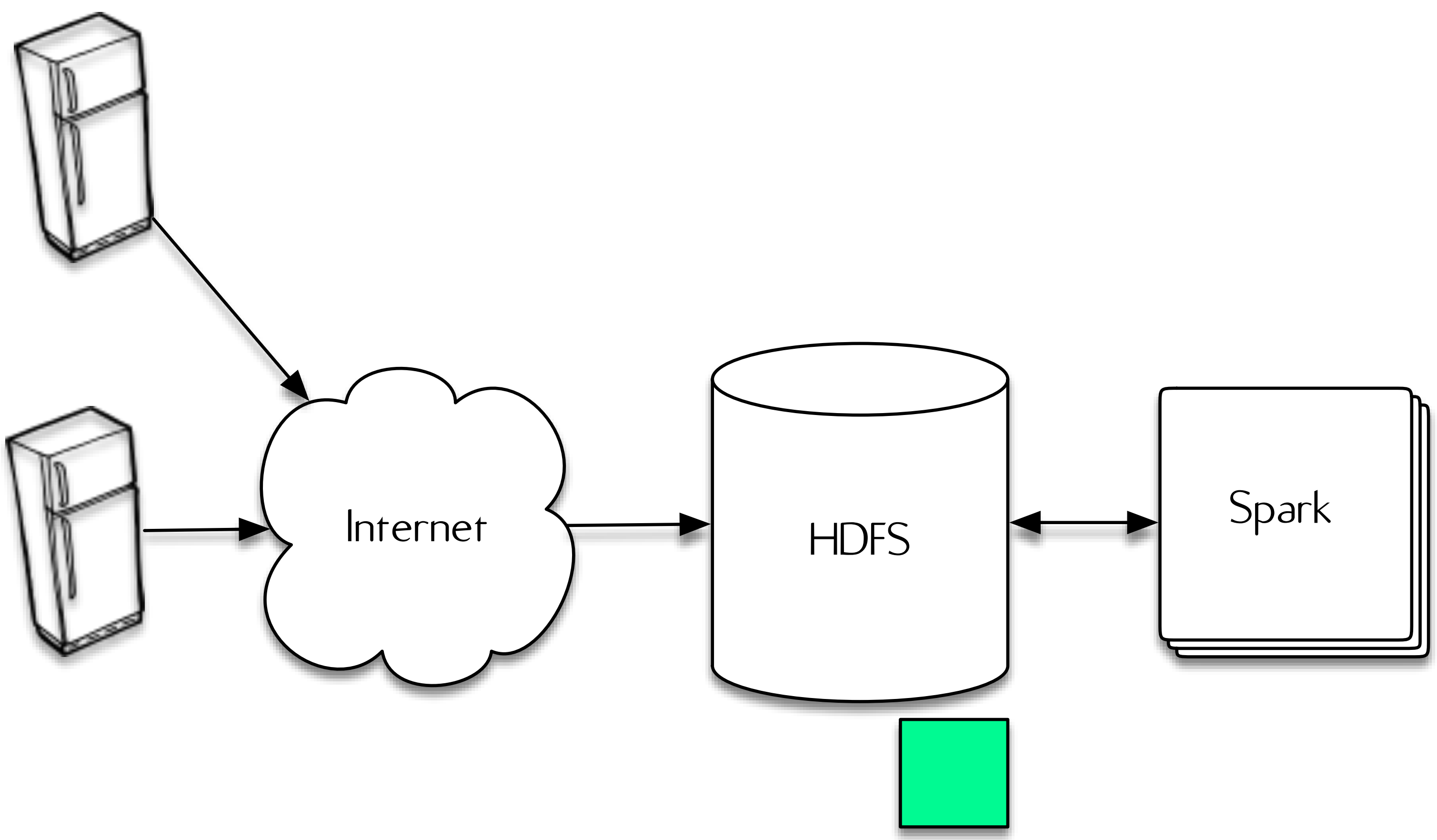

Internet
HDFS
Spark

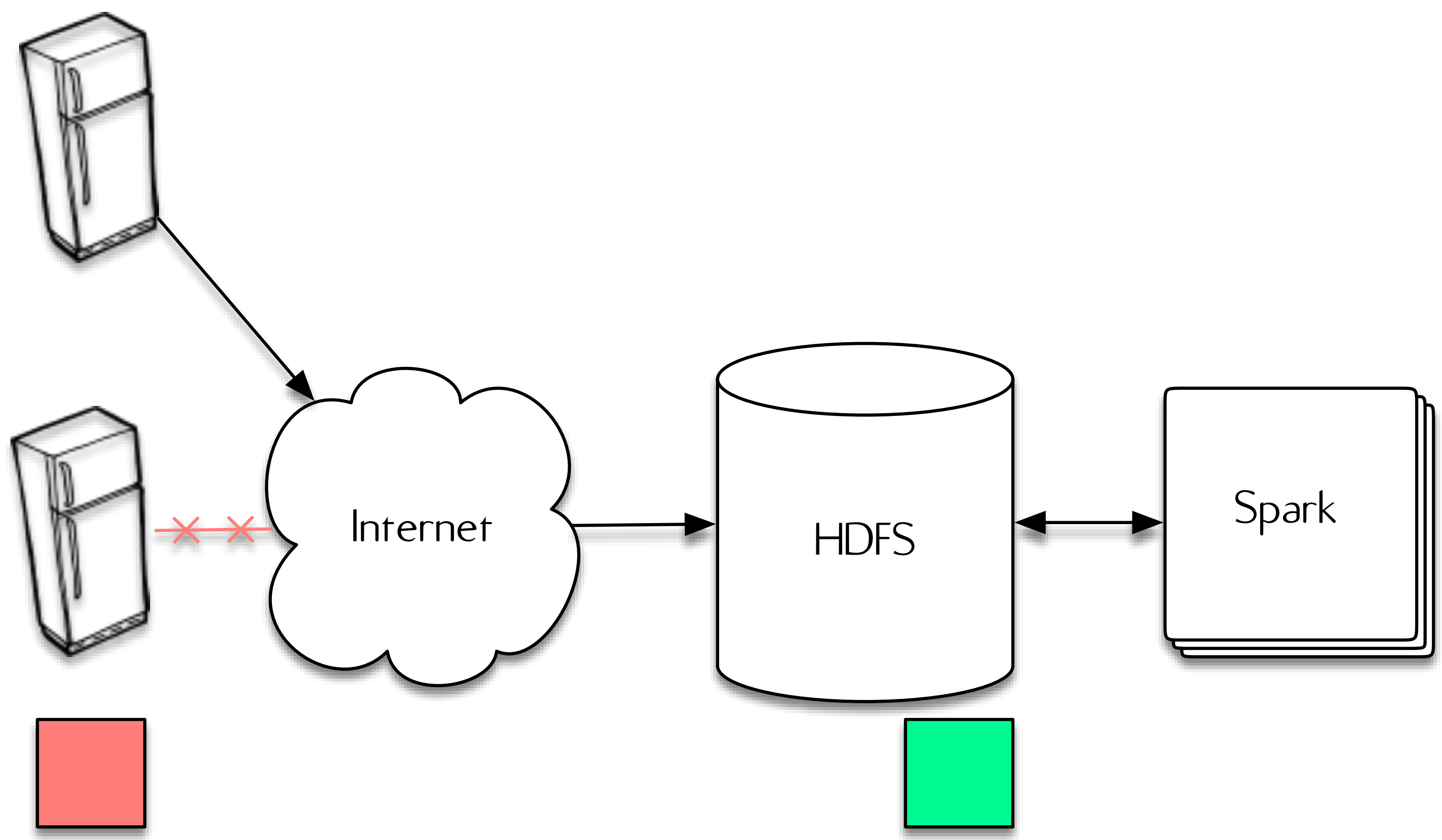

Internet
HDFS
Spark

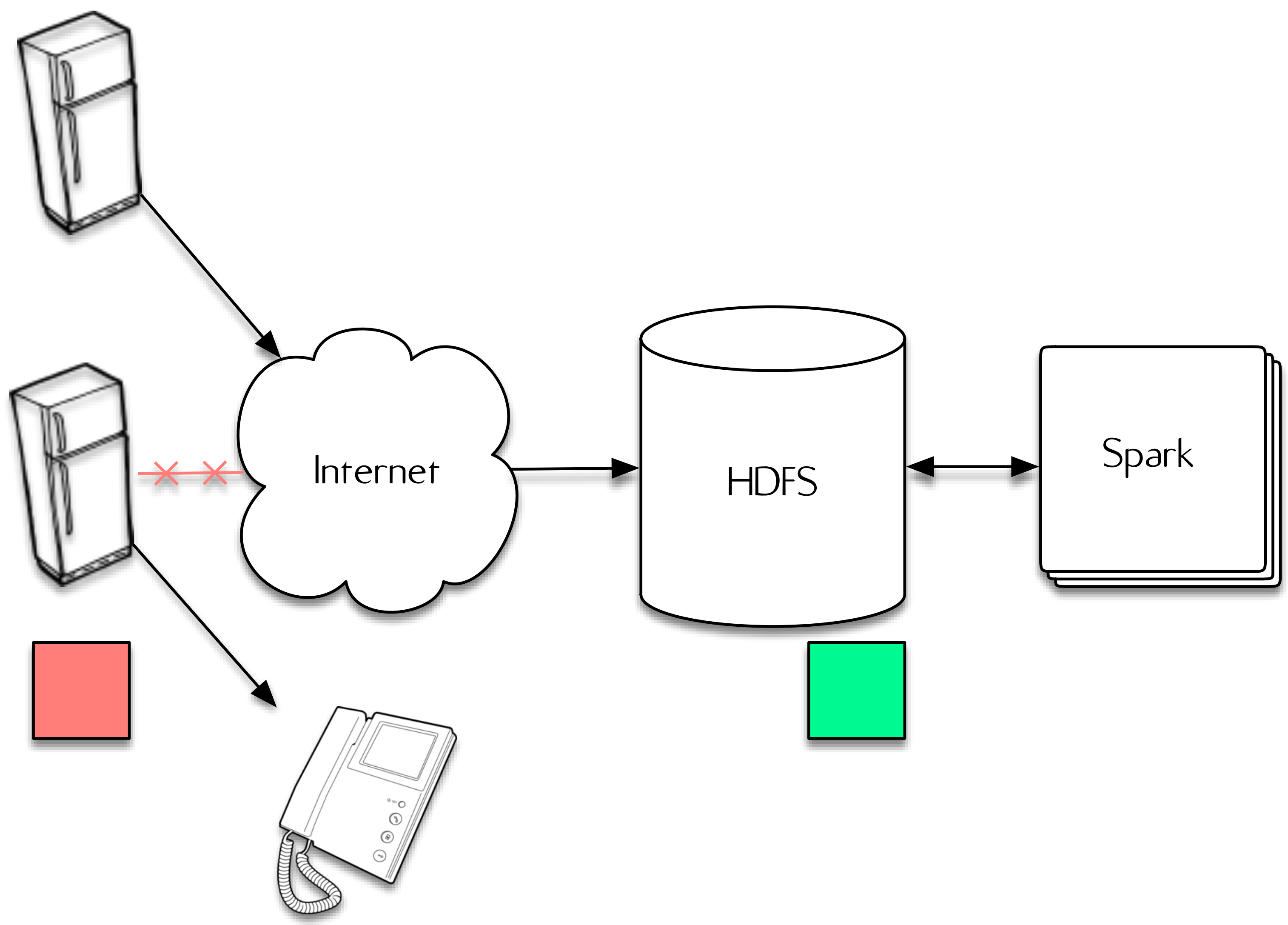
Solution
Local Decisions


Internet
HDFS
Spark


Internet
HDFS
Spark


Internet
HDFS
Spark


Internet
HDFS
Spark


Internet
HDFS
Spark

Local Decisions

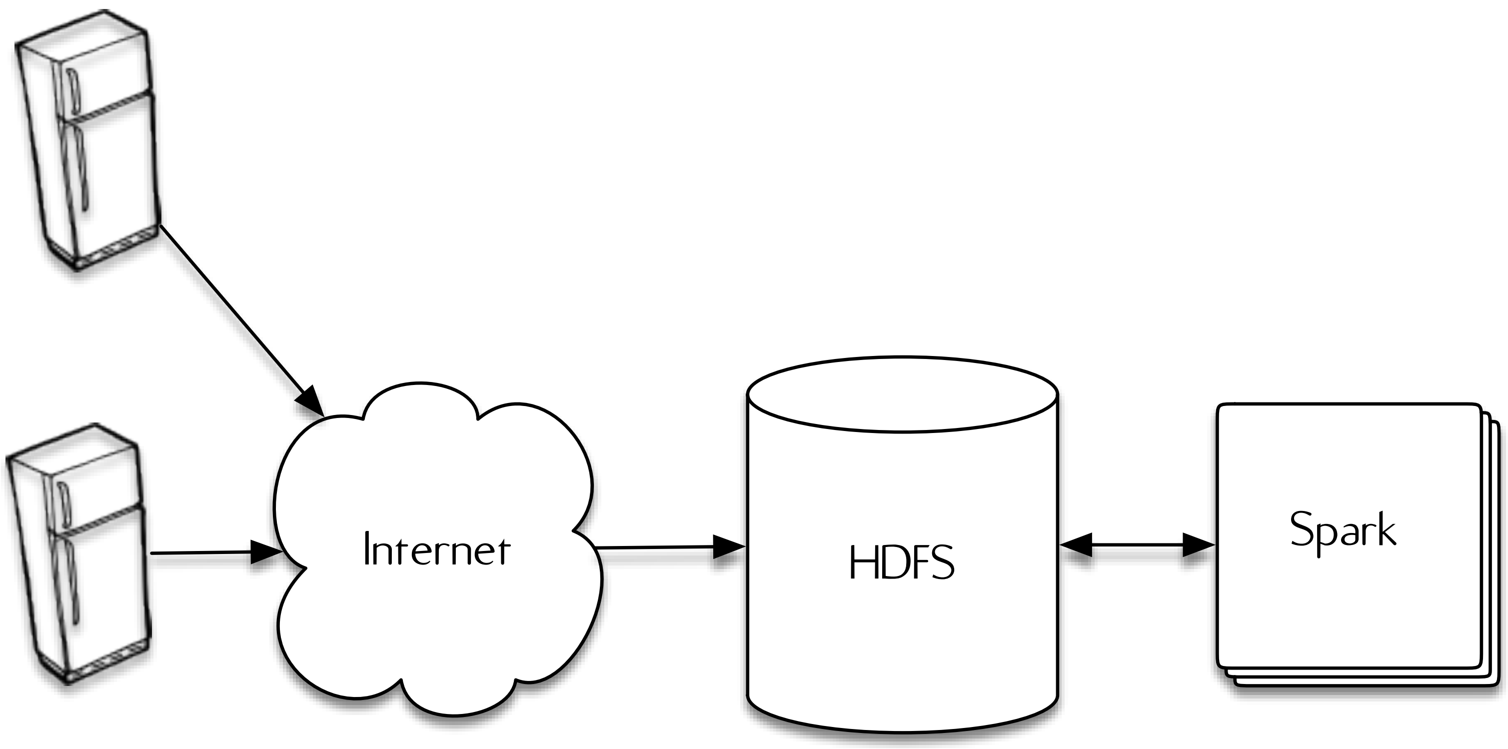
- Not new for backup (80s-90s)
  Backup communication mechanisms for critical systems; POTS backup for ISDN, etc.

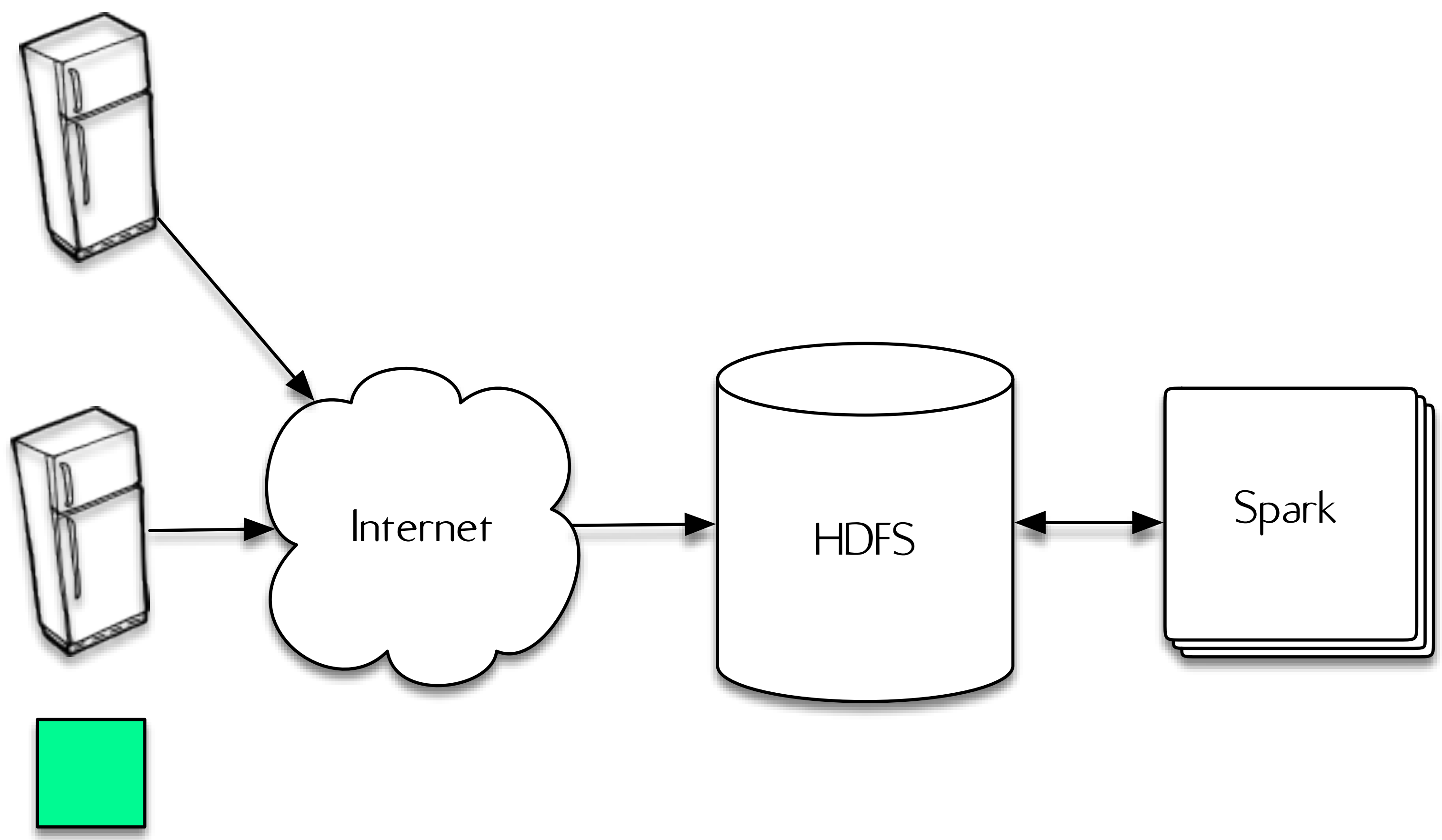
Local Decisions

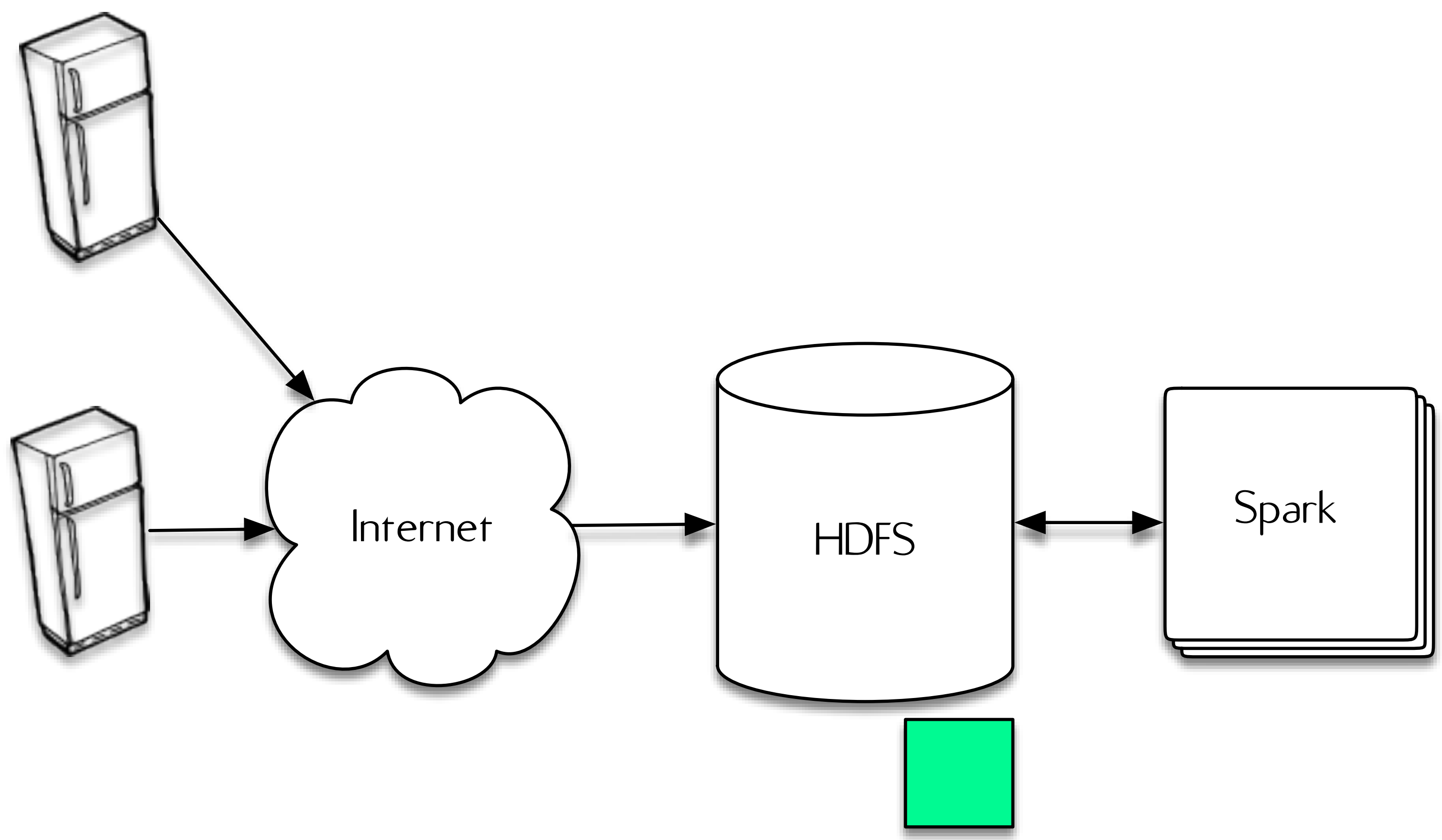
- **Not new for backup (80s-90s)**
  Backup communication mechanisms for critical systems; POTS backup for ISDN, etc.

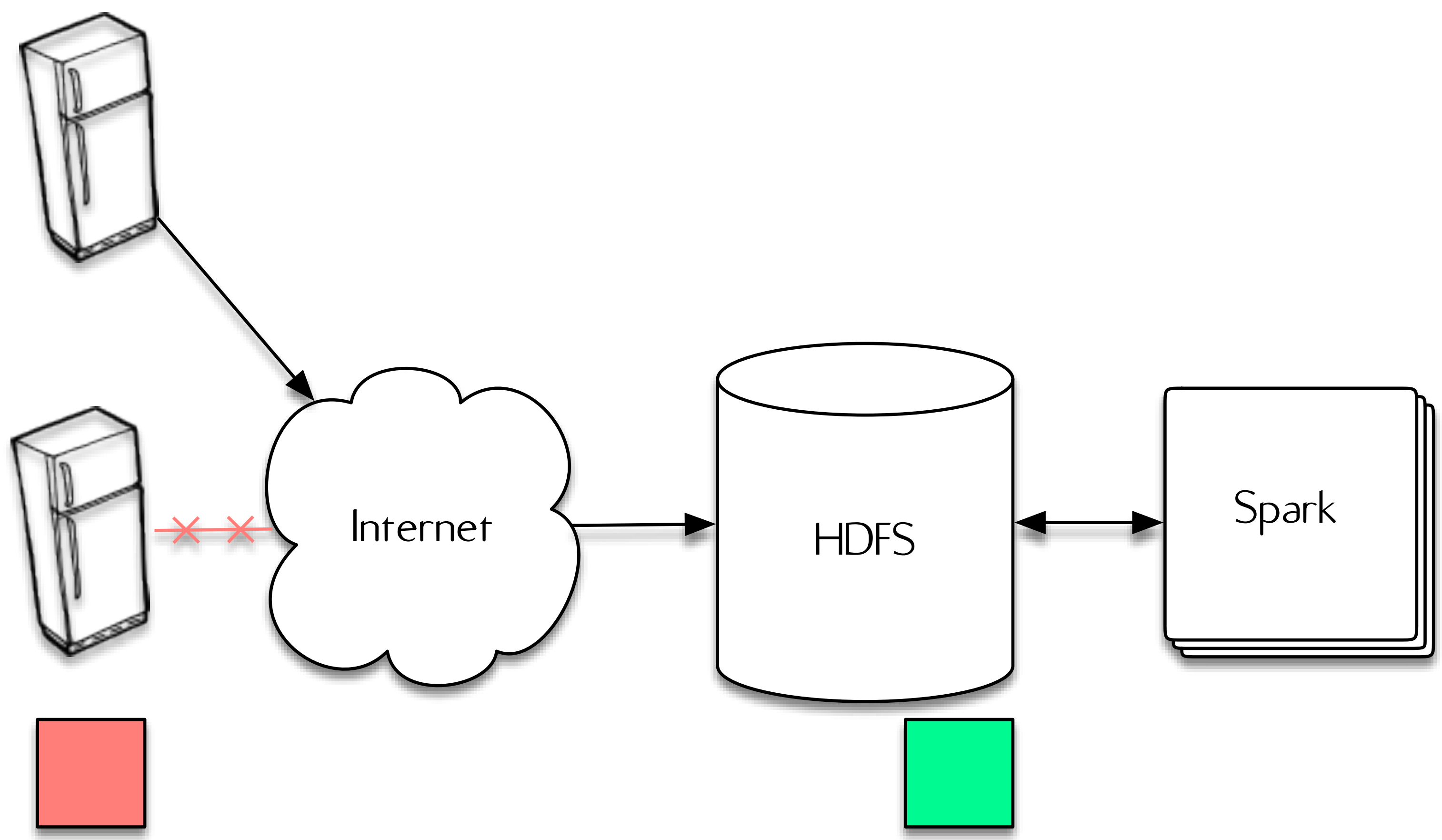
- **Not new for storage (90s-00s)**
  EMC's "phone home" via POTS when disks failed in NAS devices to signal for replacement unit

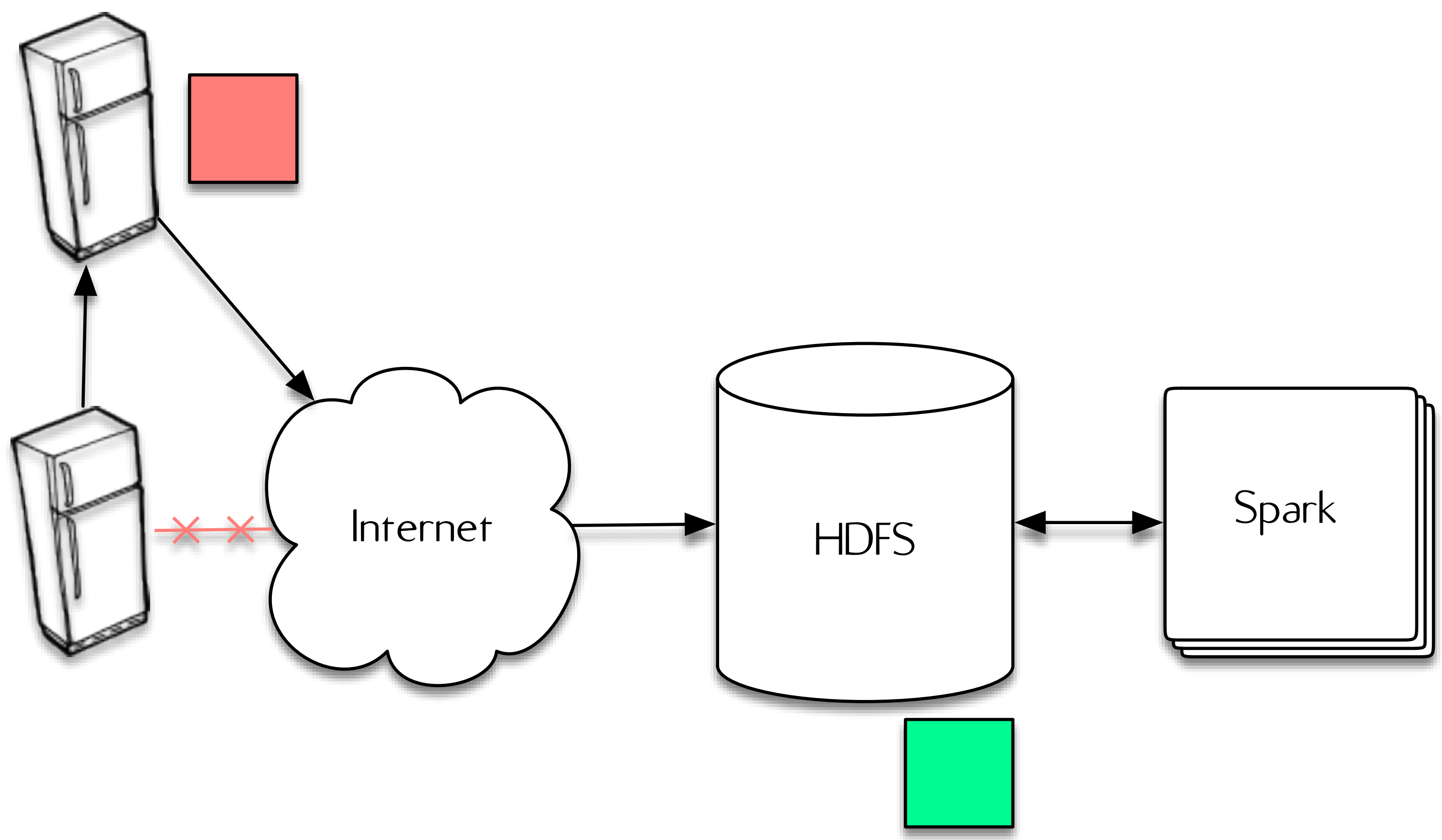
Solution
Transitive Dissemination

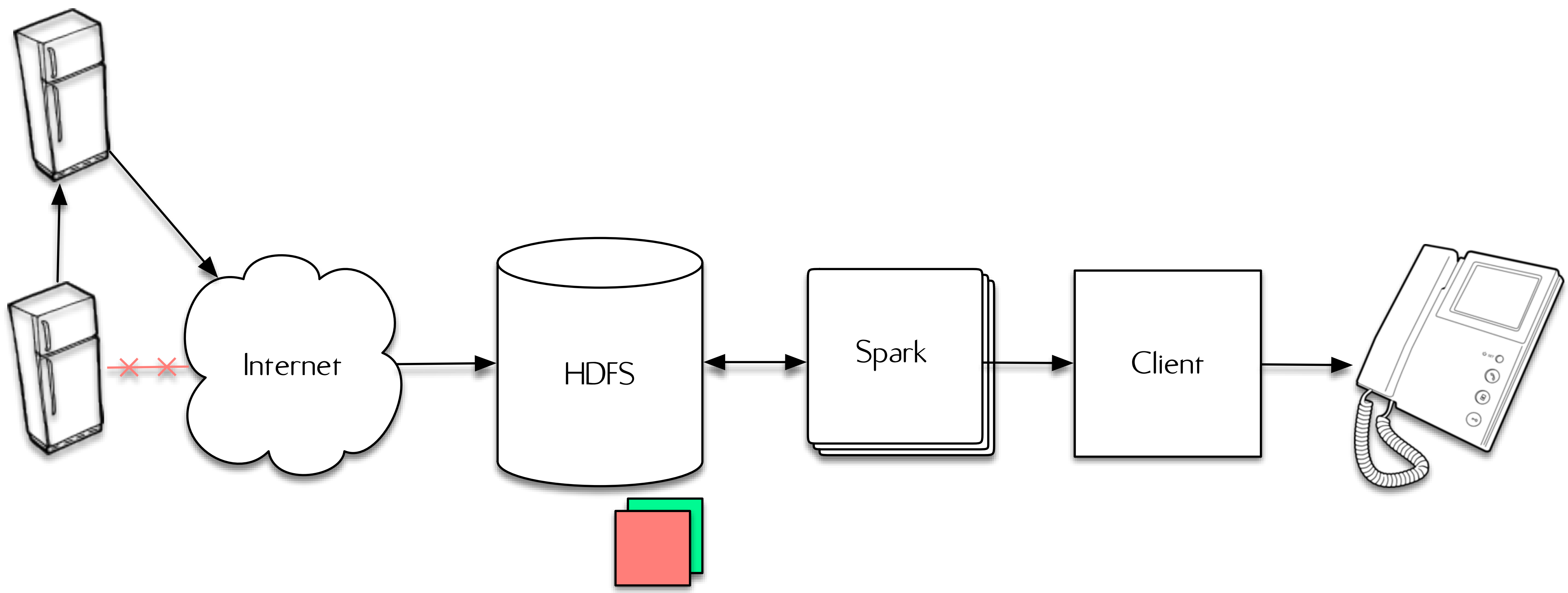

Internet
HDFS
Spark


Internet
HDFS
Spark


Internet
HDFS
Spark

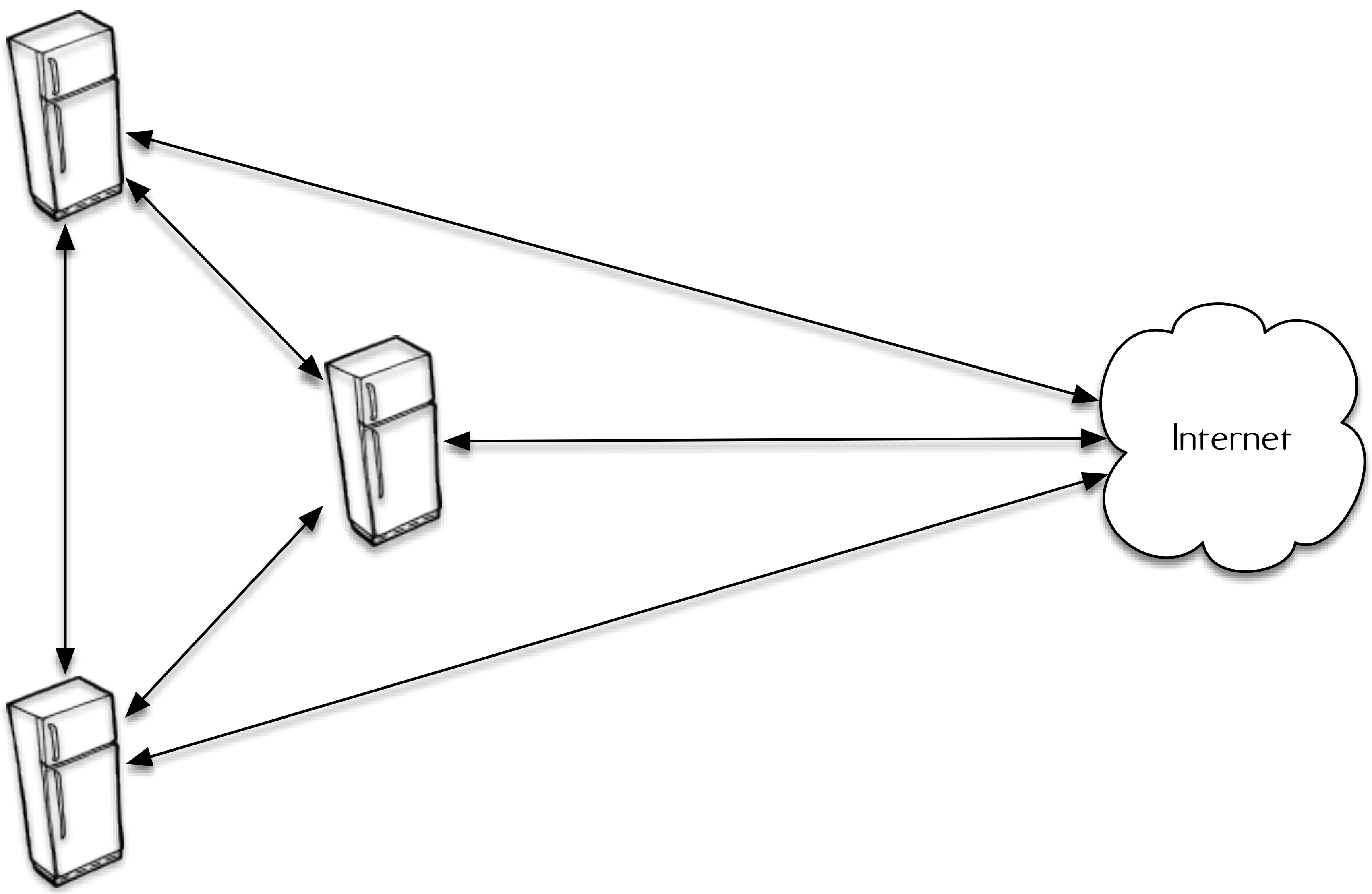

Internet
HDFS
Spark

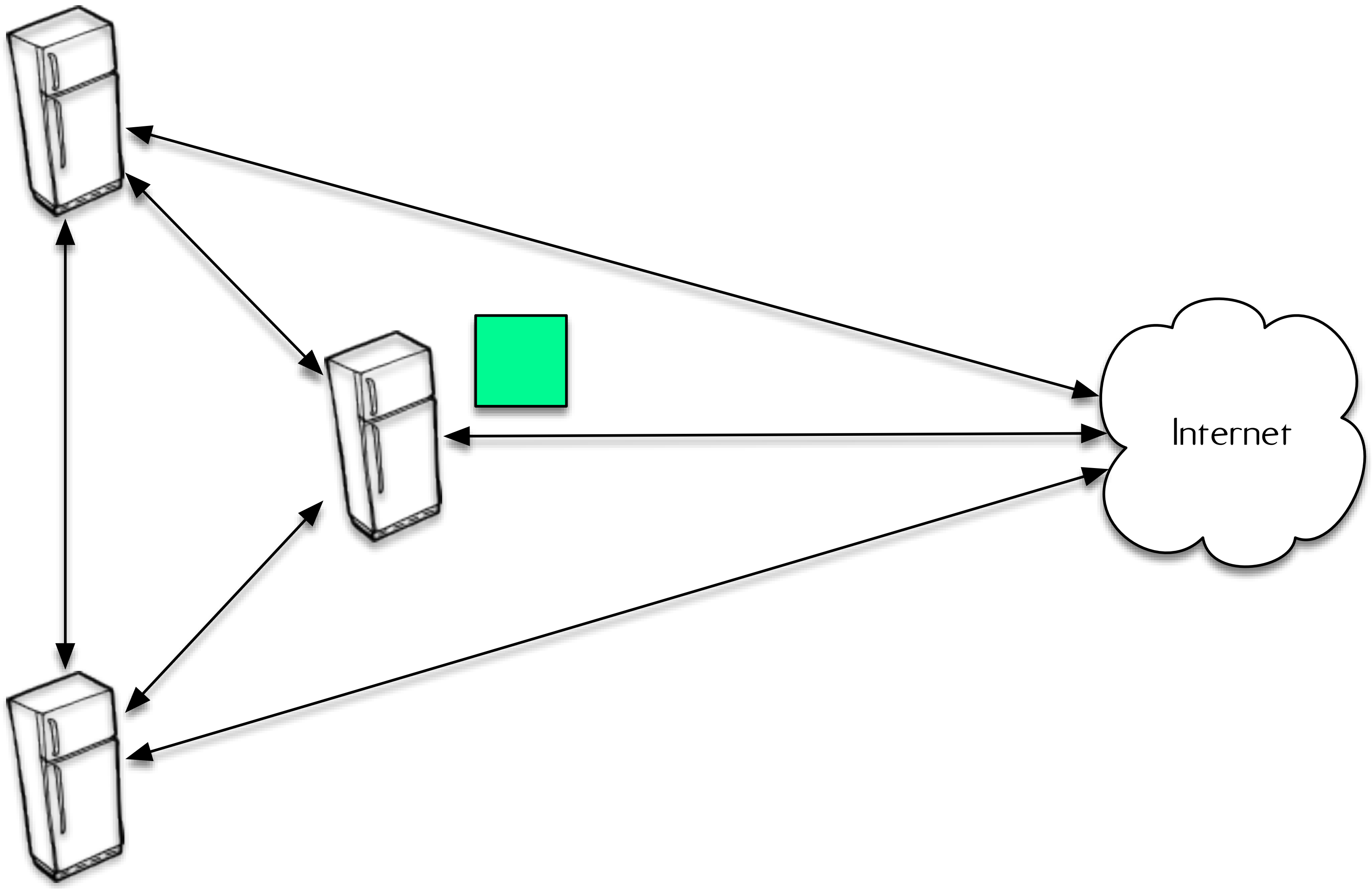

Internet
HDFS
Spark

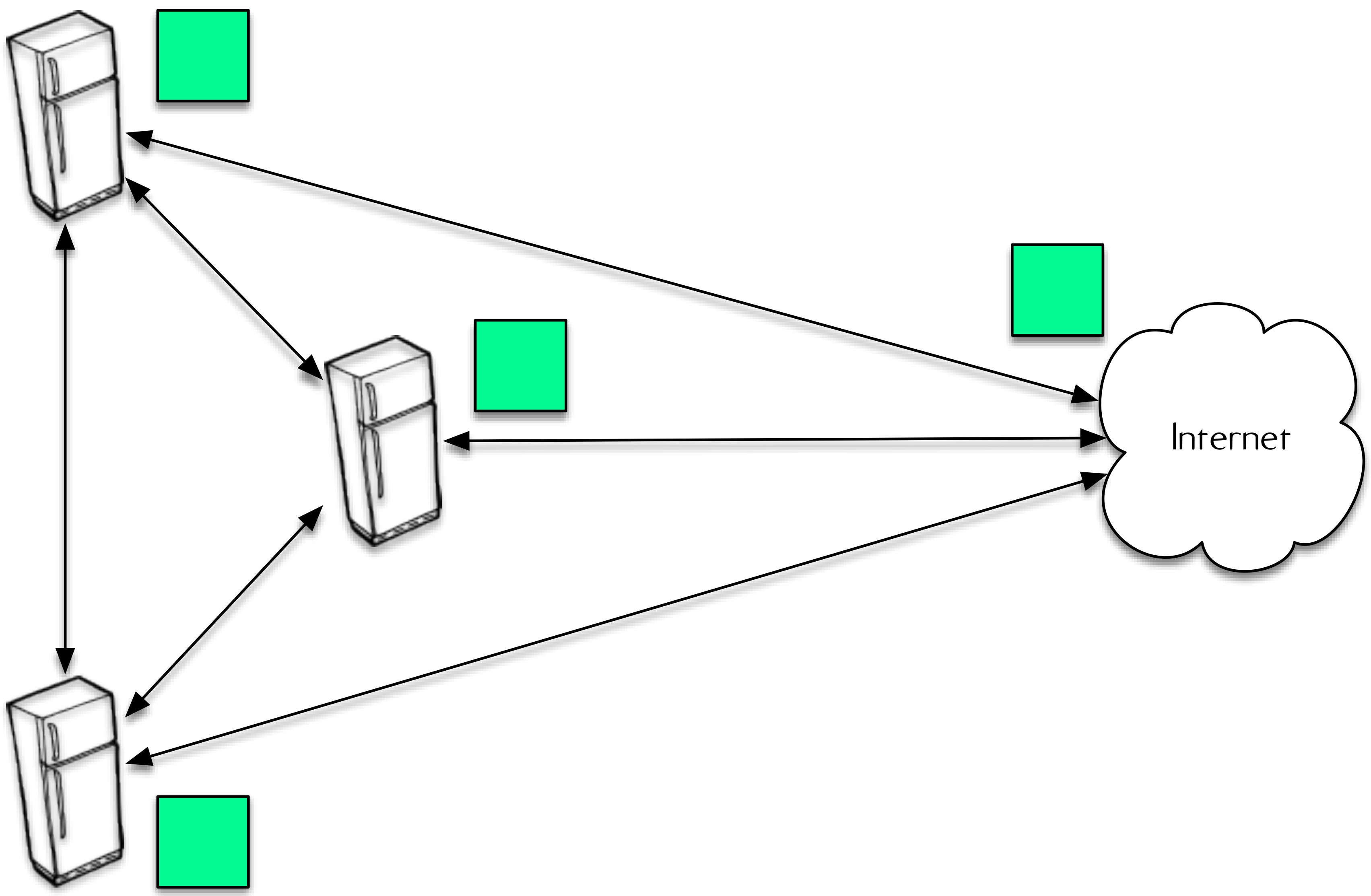


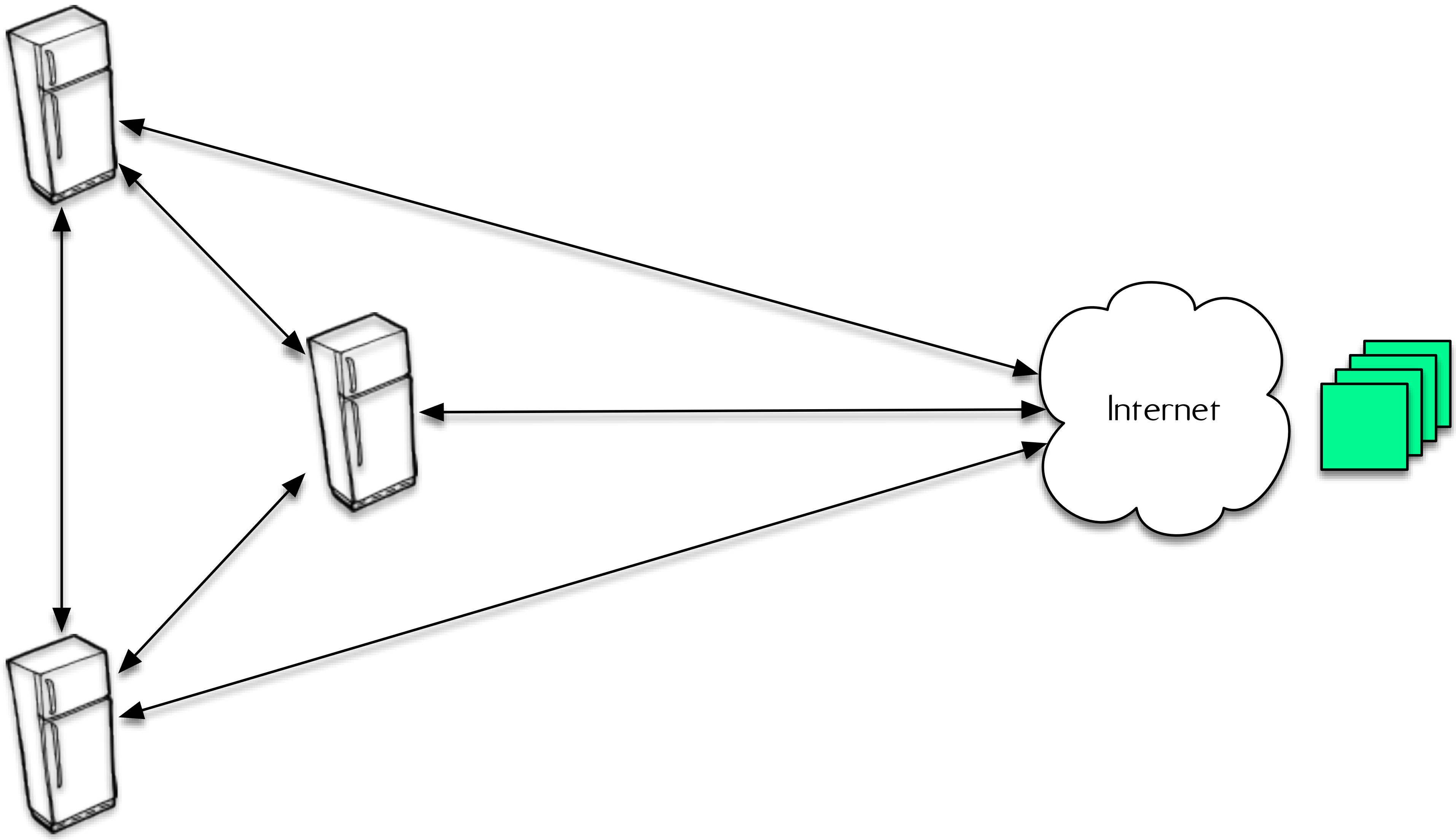
Internet
HDFS
Spark
Client

Problem
State Transmission


Internet

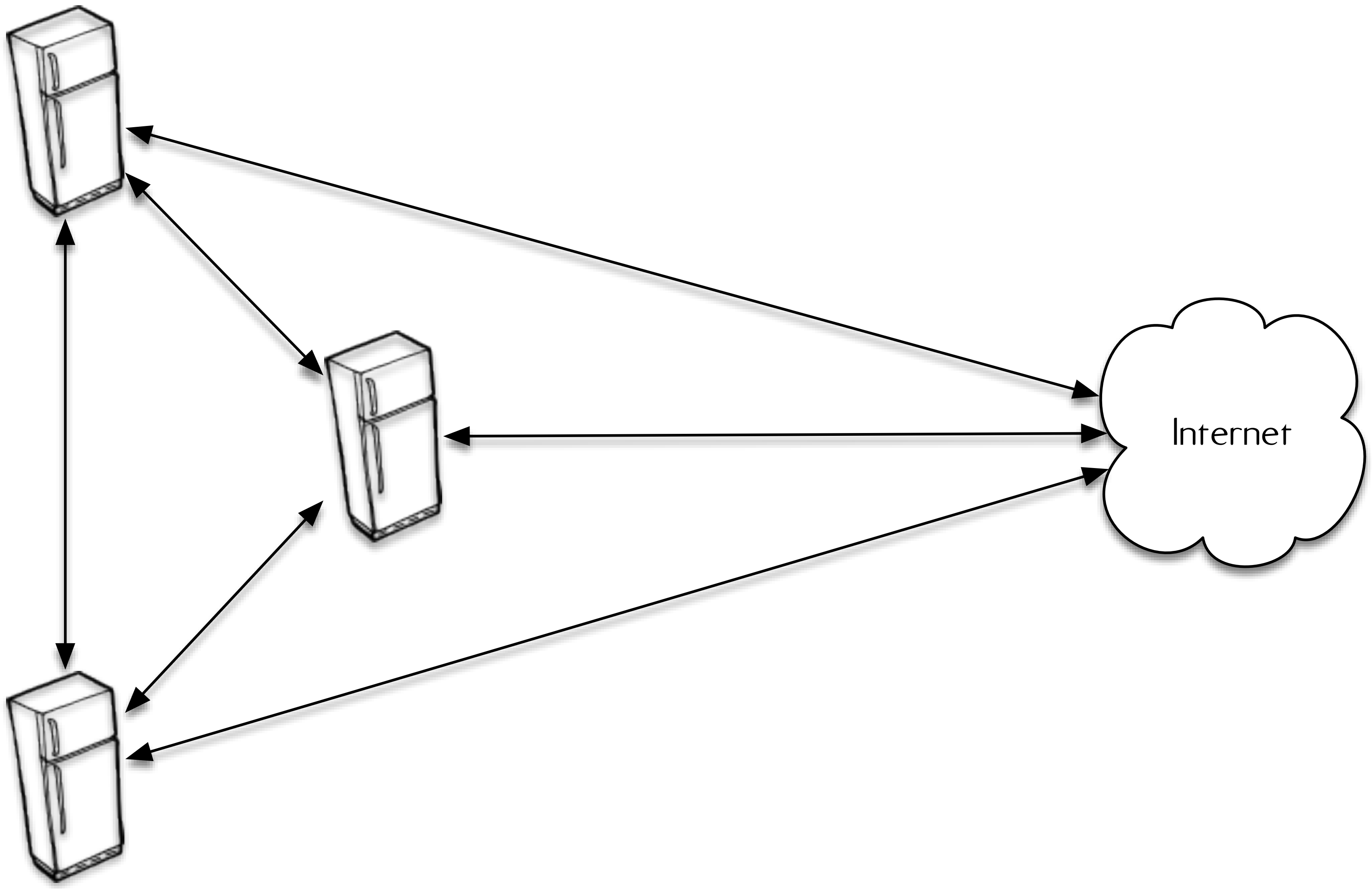

Internet

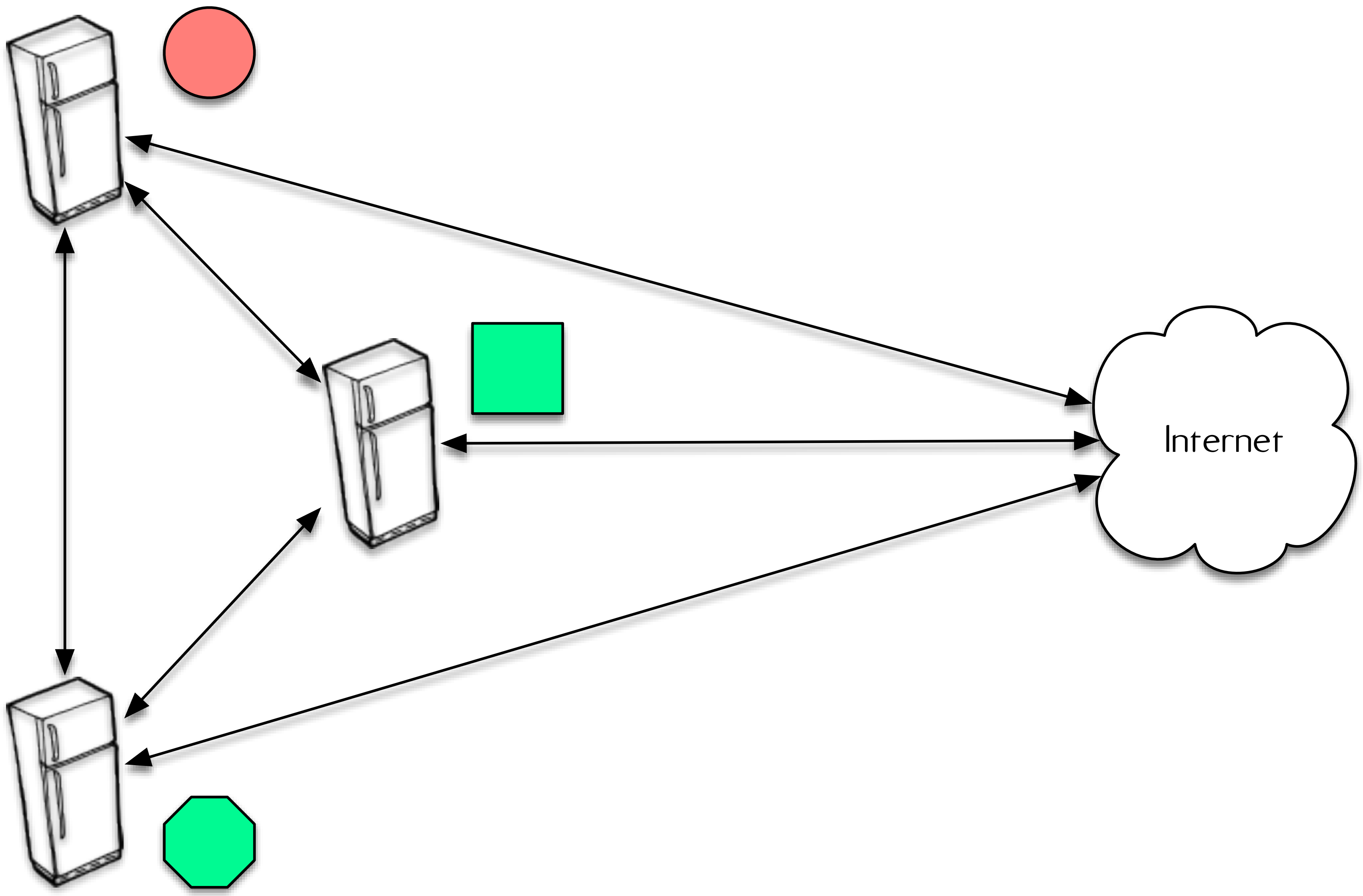

Internet

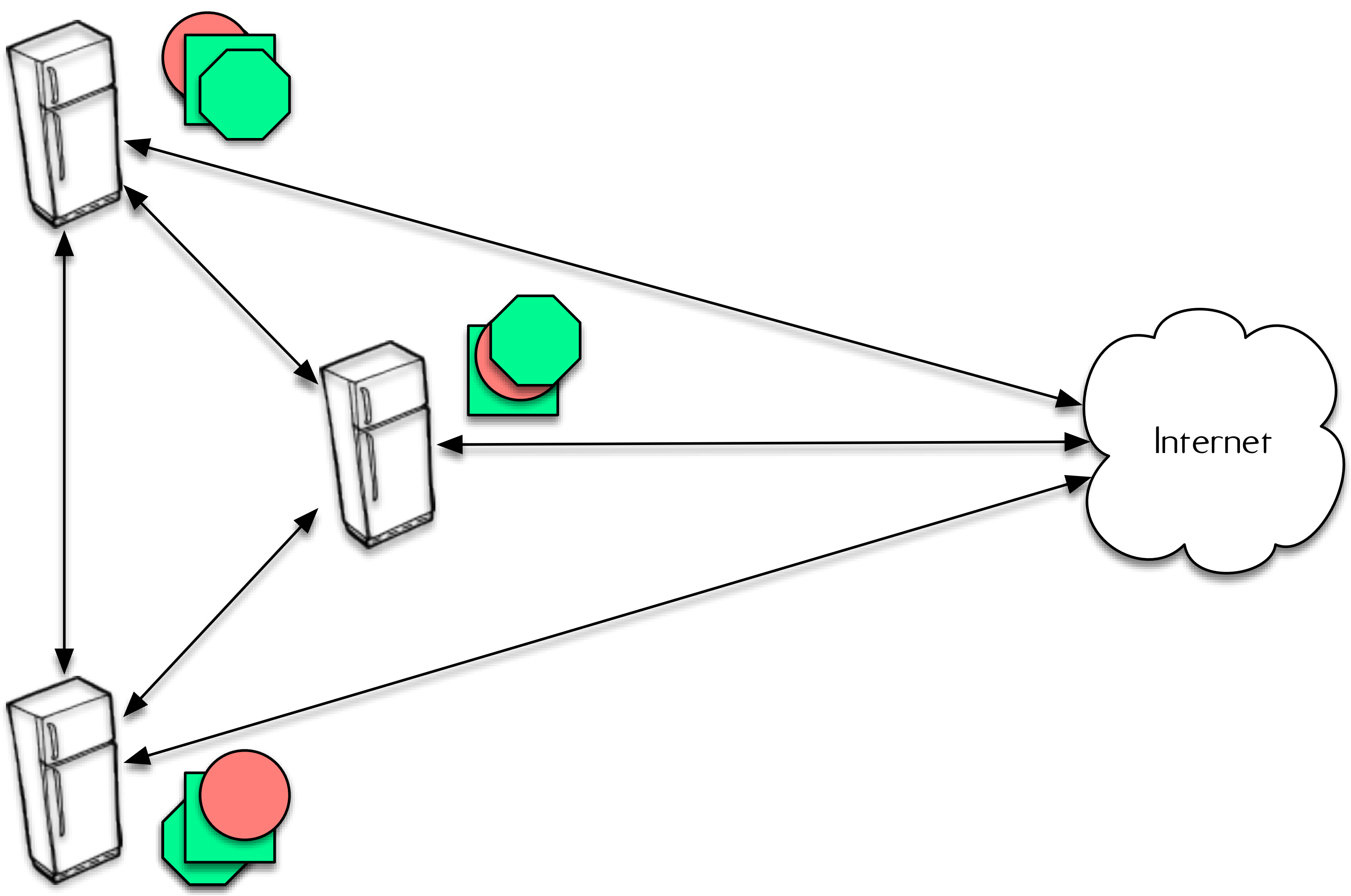

Internet

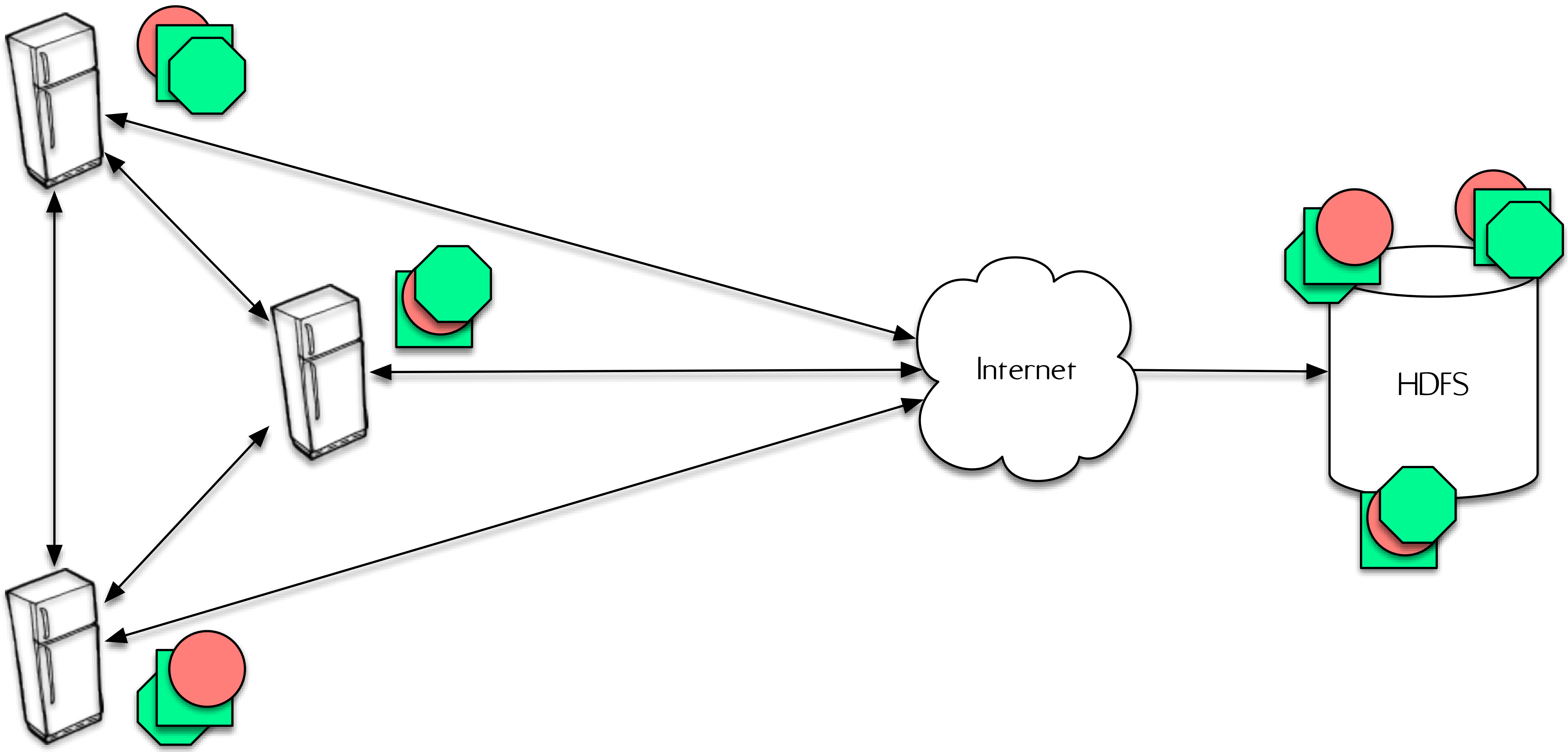
Solution

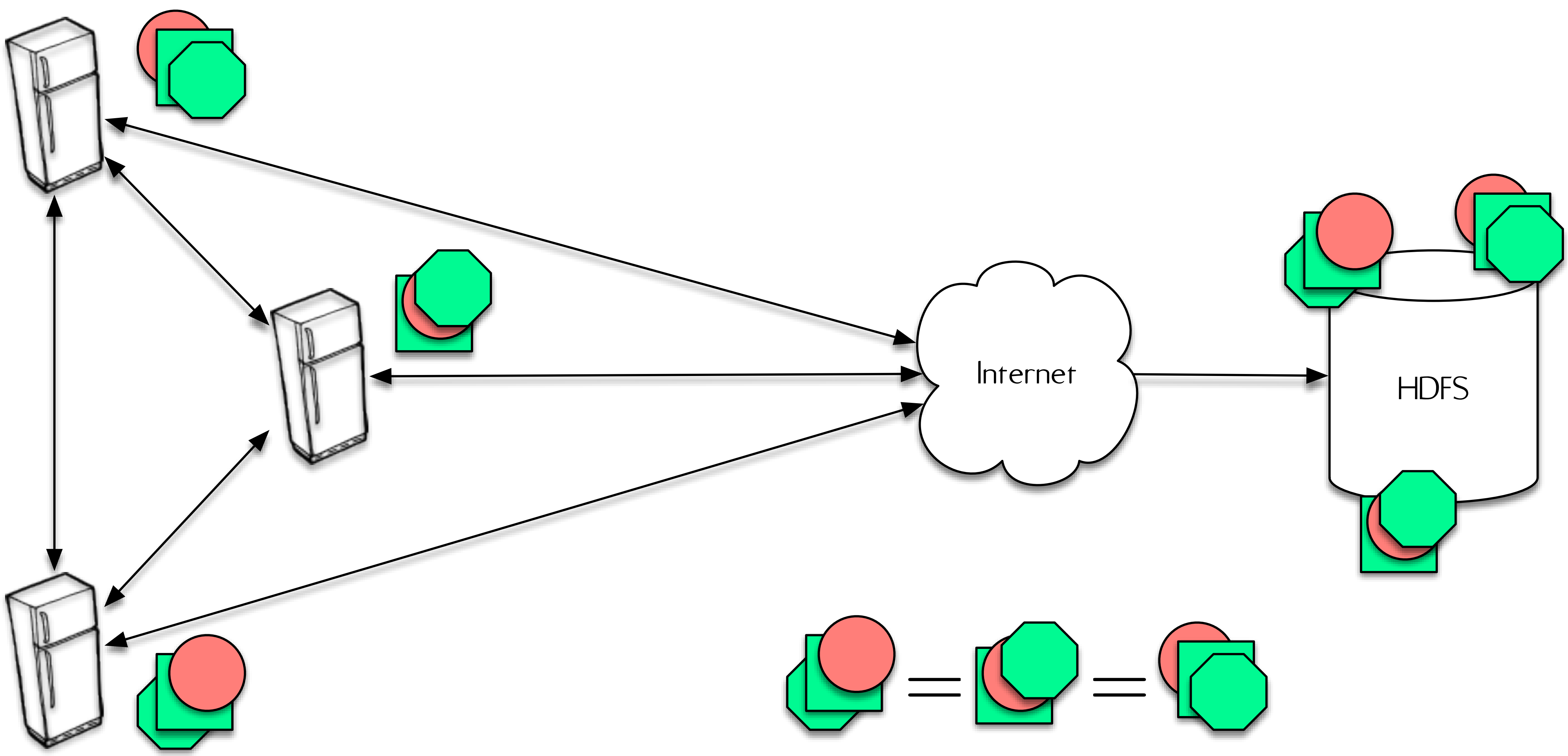
Aggregate Dissemination

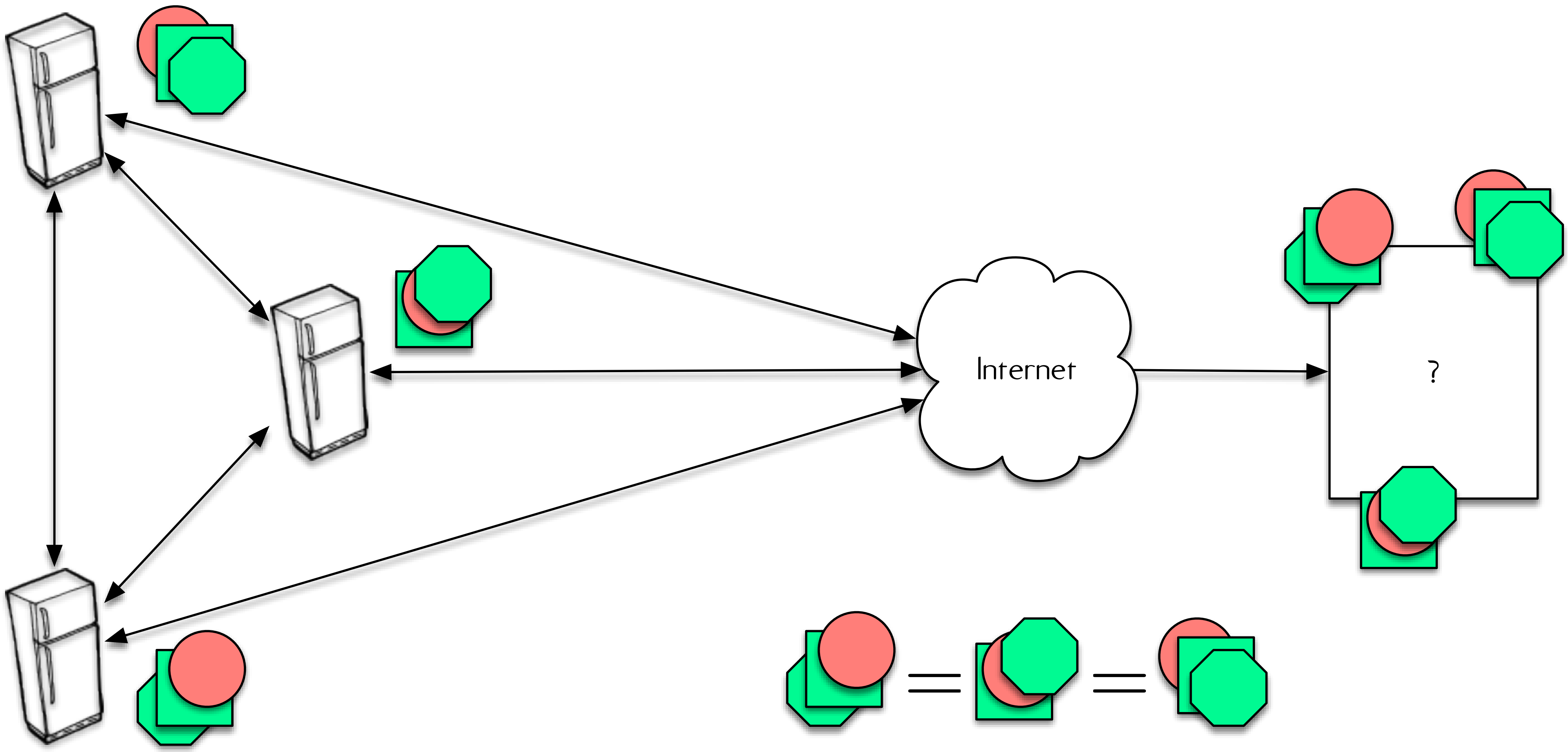

Internet

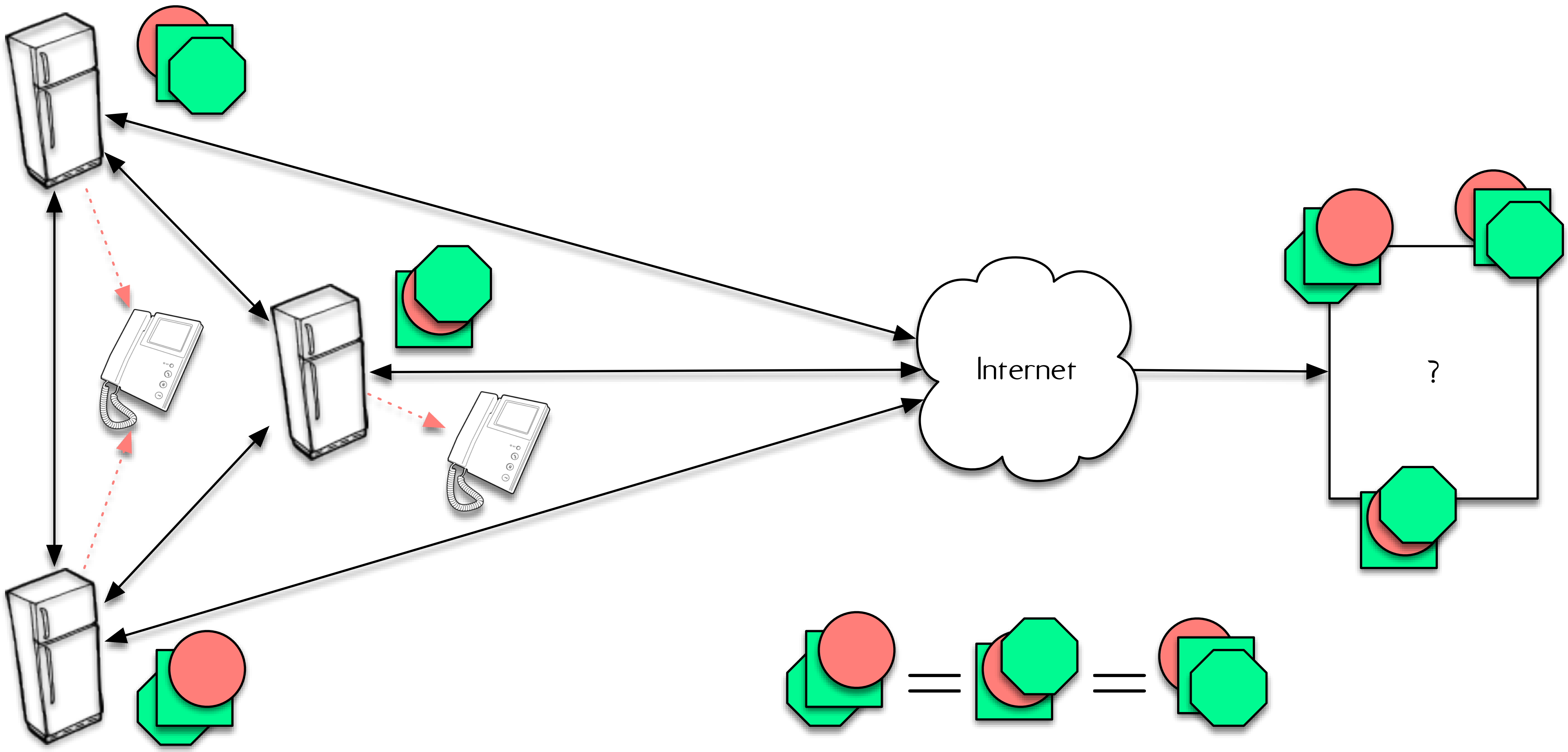

Internet


Internet


Internet
HDFS


Internet
HDFS


Internet
?


Internet
?

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

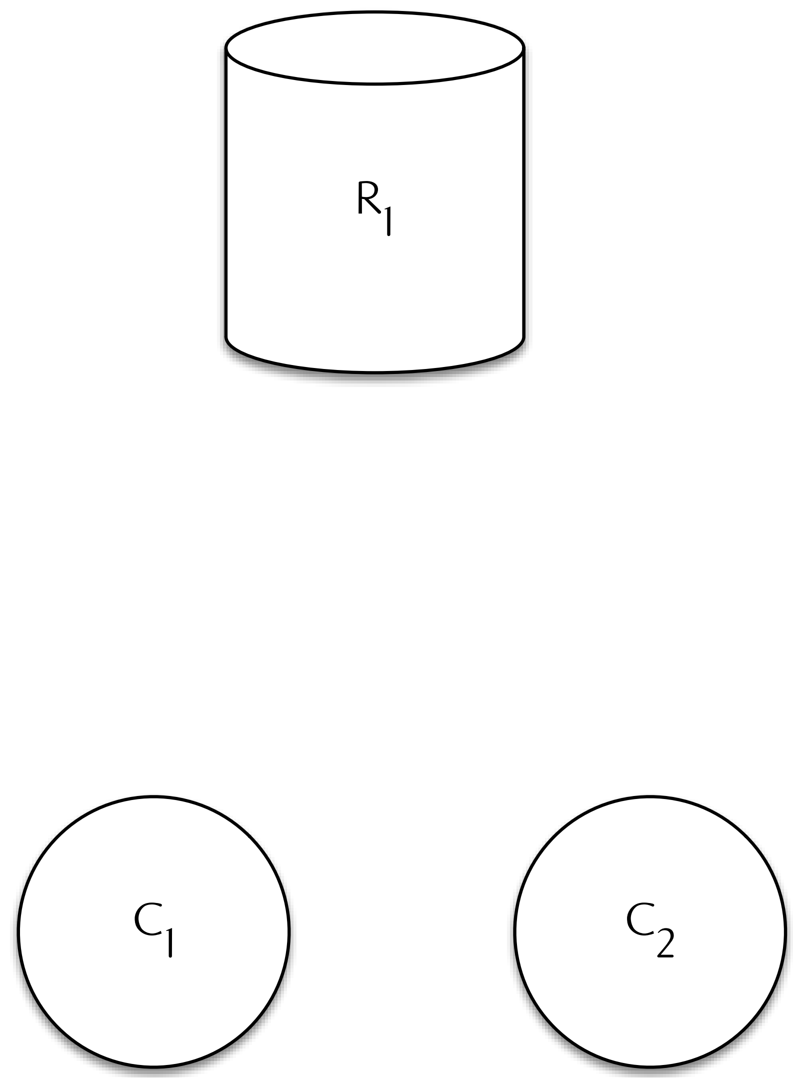
- **Make local decisions**
  Make decisions based on results of local computation

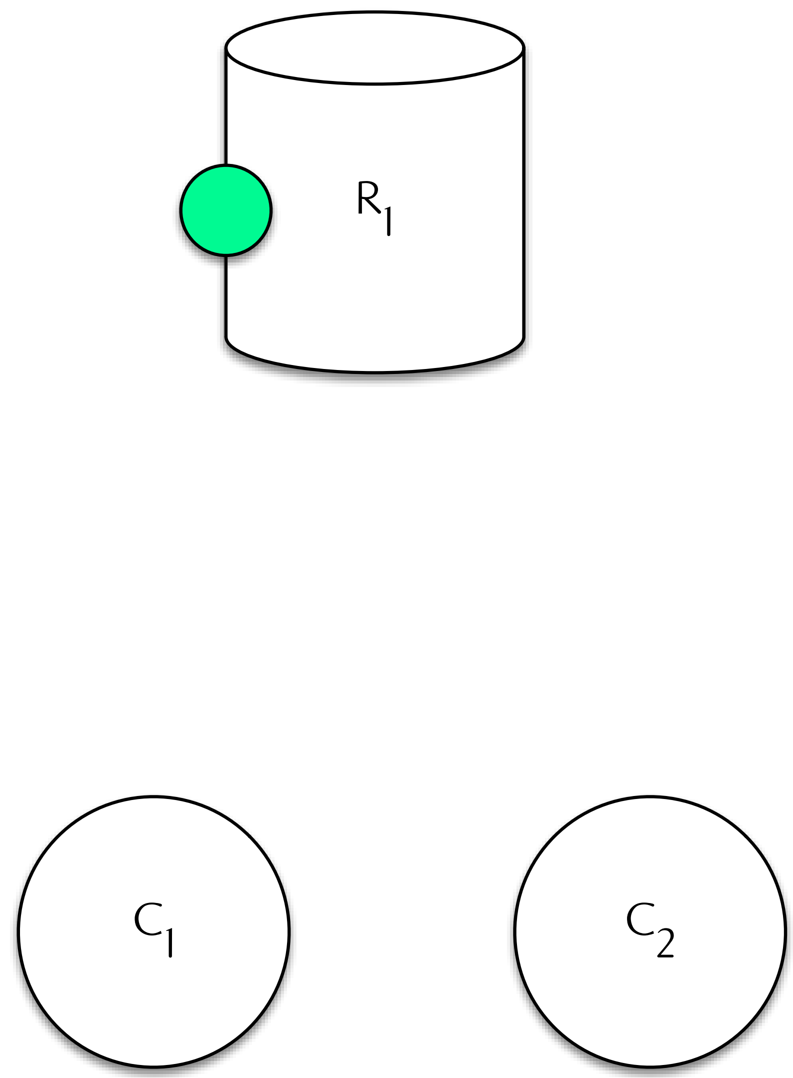
Databases

Consistency Models

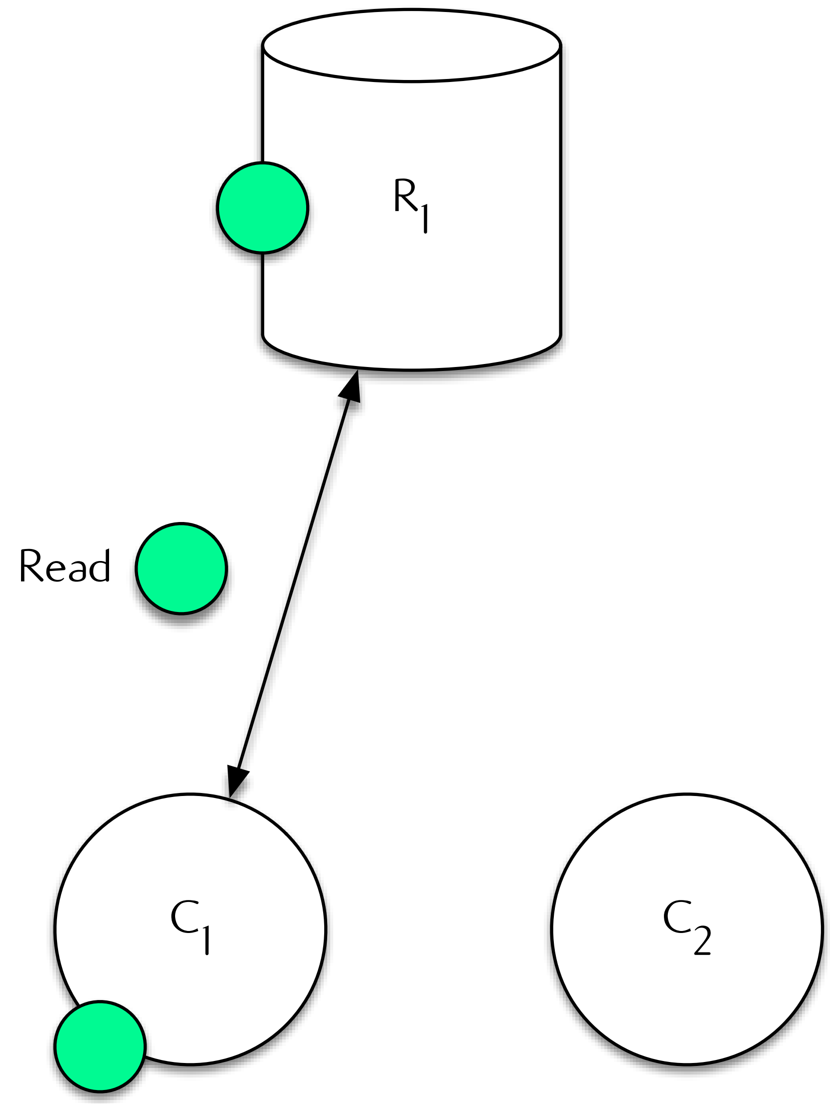
Databases

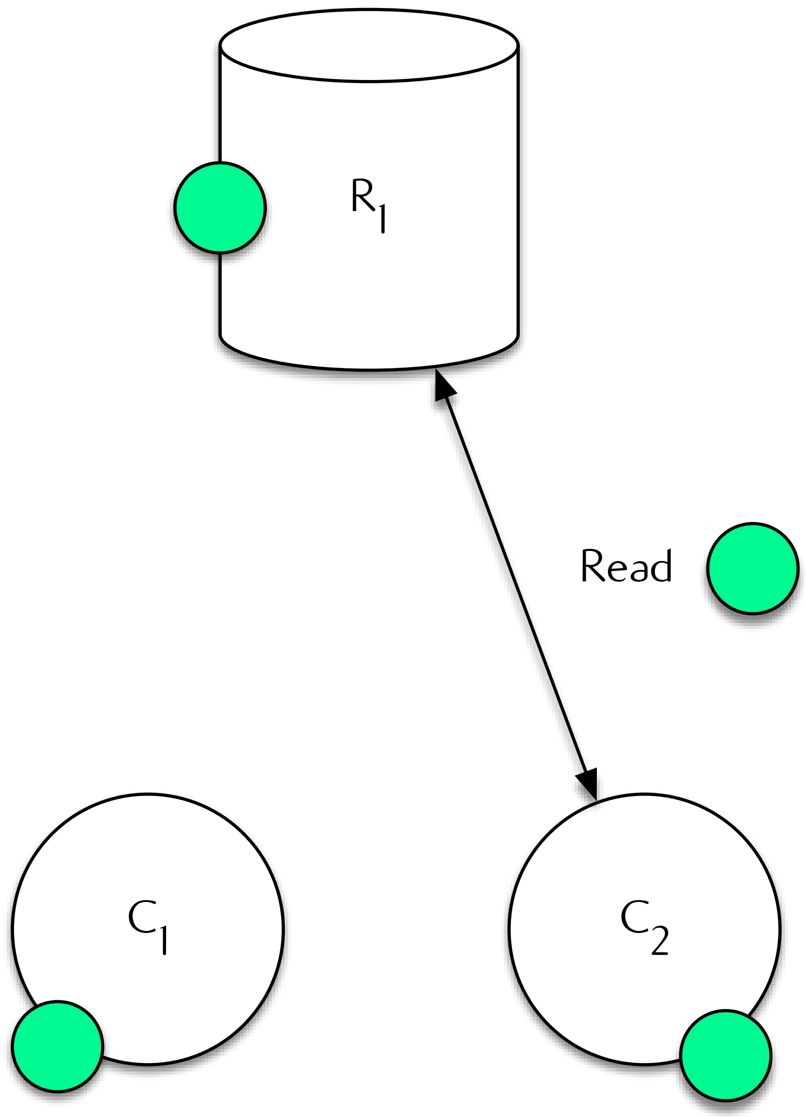
Strong Consistency


R_1
C_1
C_2

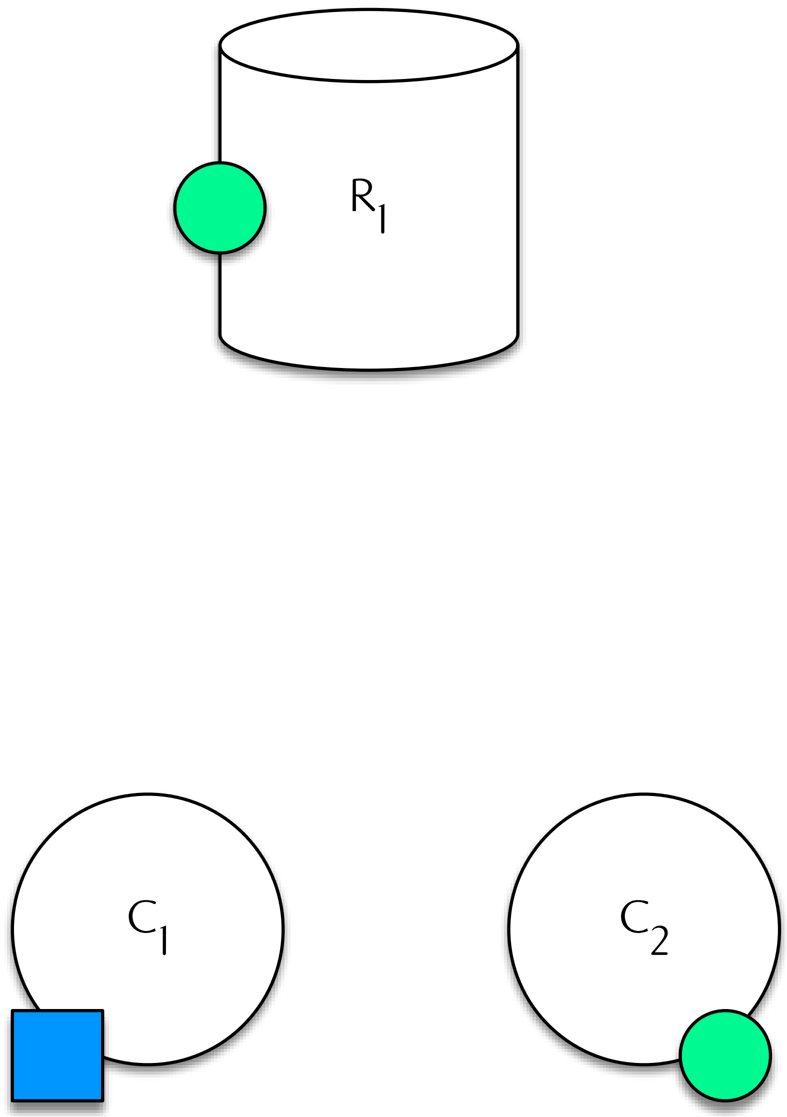


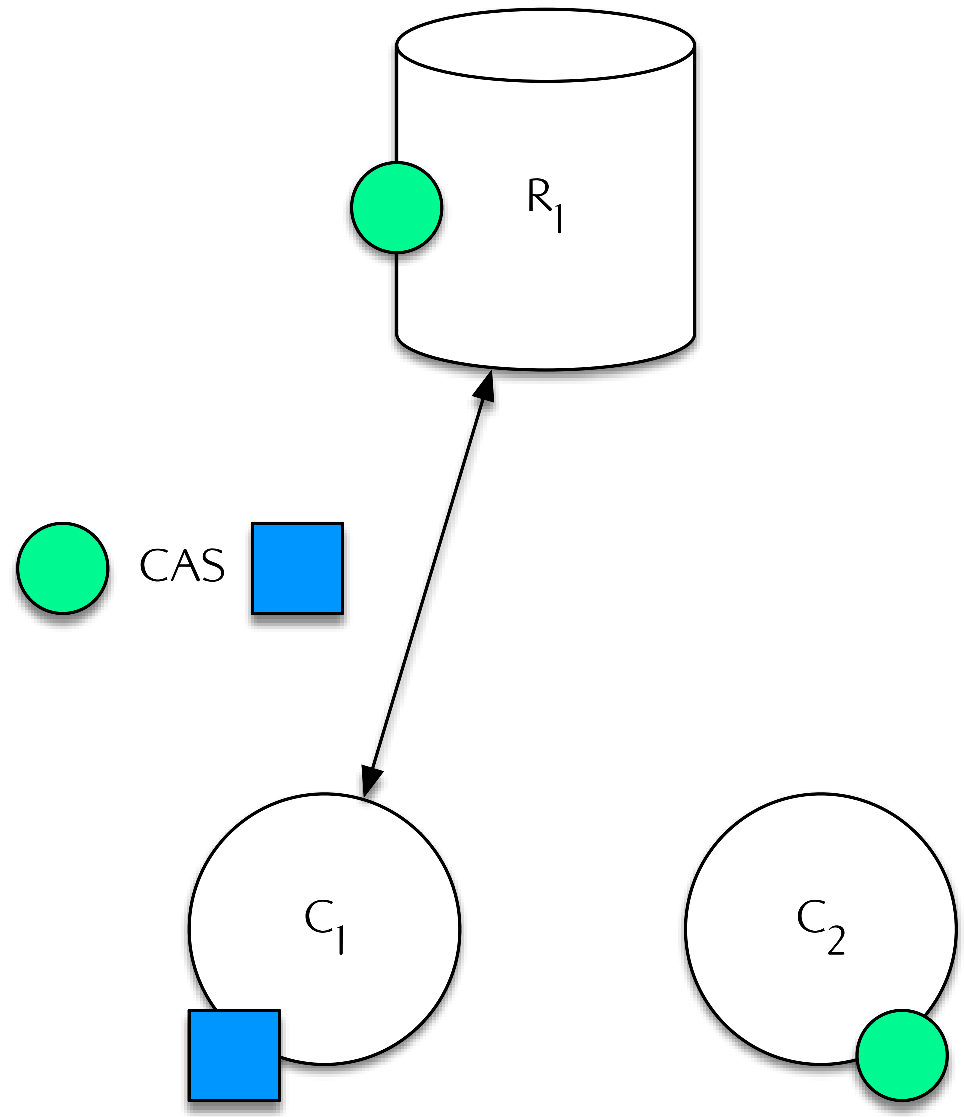
R1
C1
C2

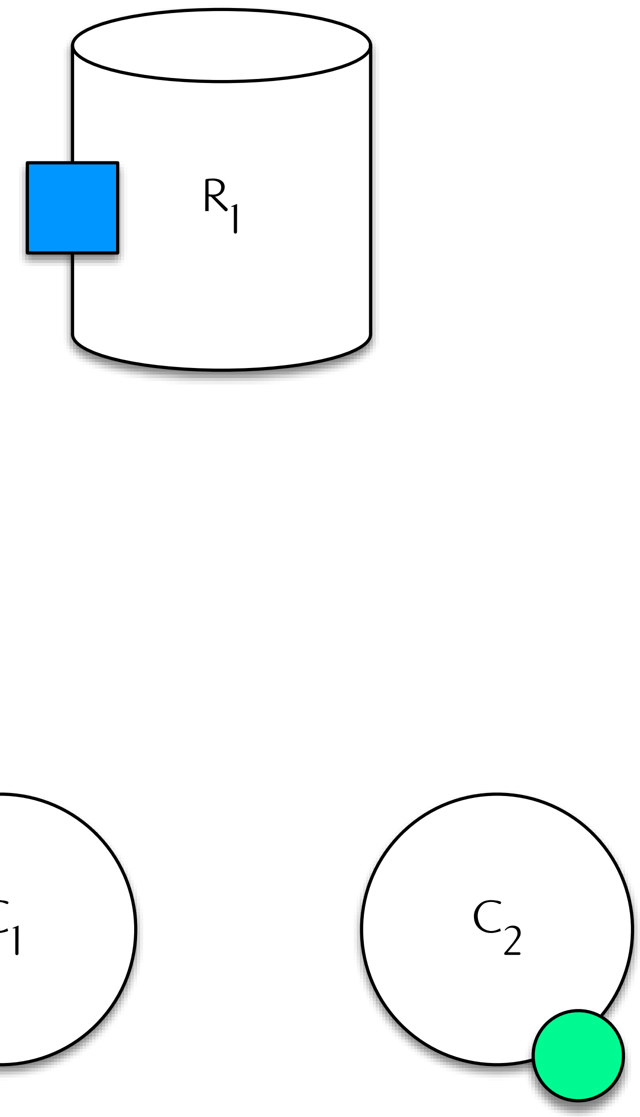

R_1
Read
C_1
C_2

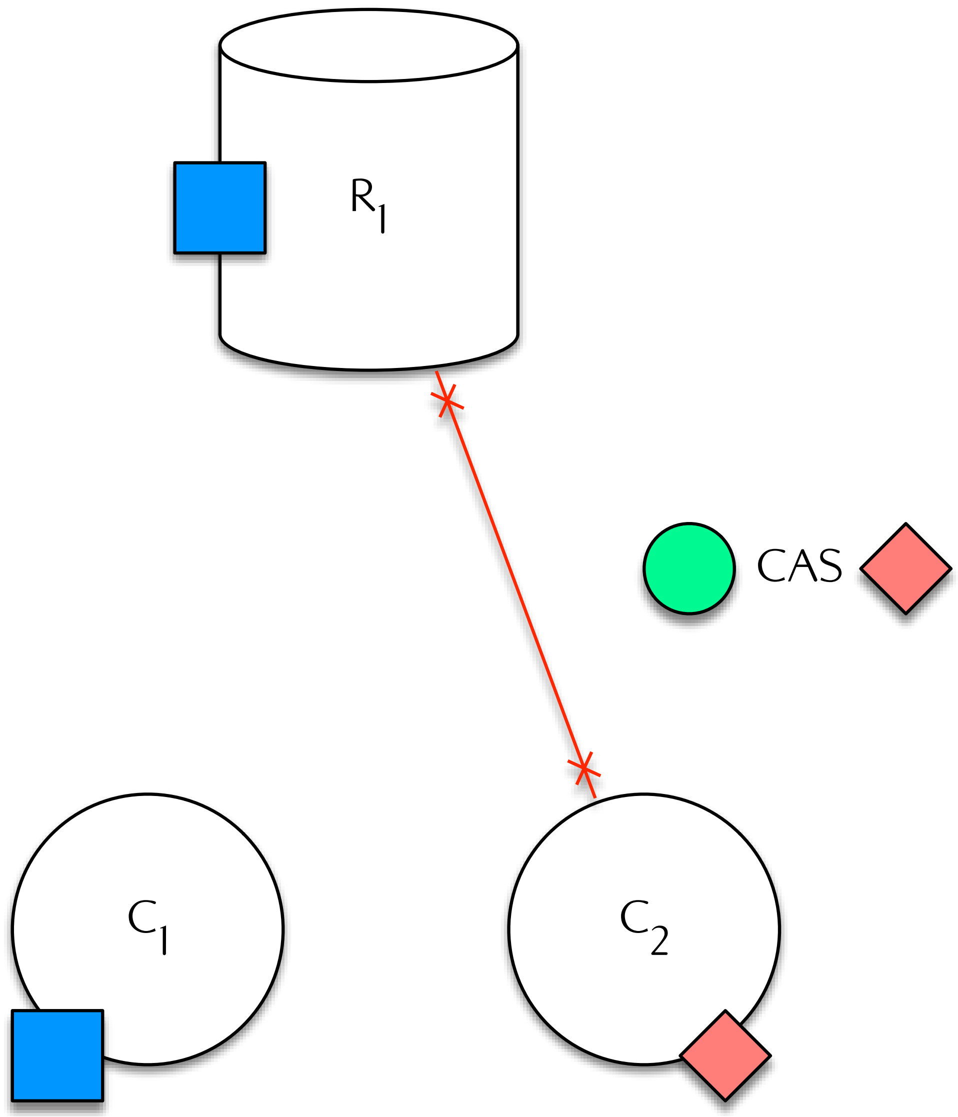


R_1
Read
C_1
C_2

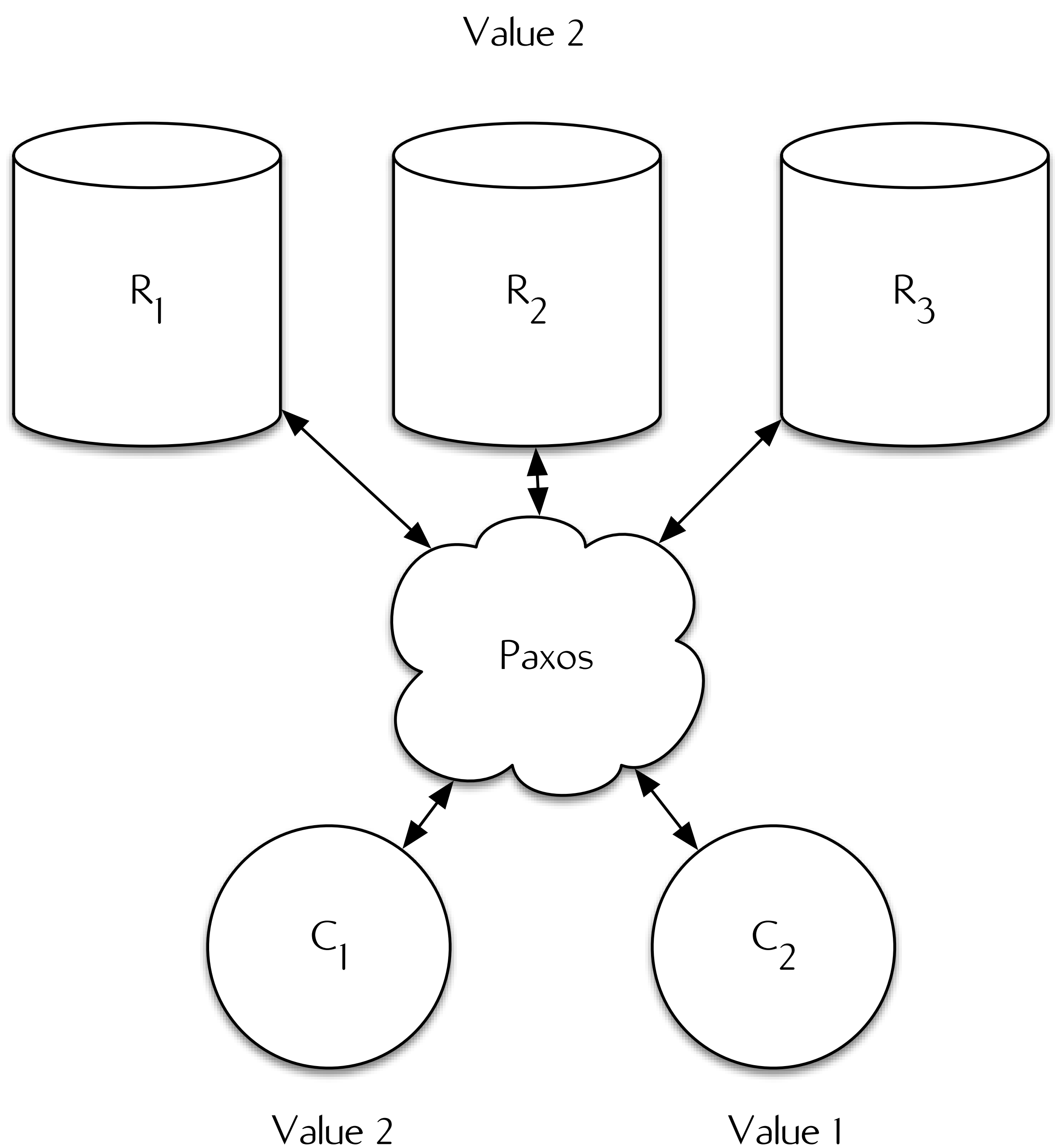

R1
C1
C2


R1
CAS
C1
C2


R_1
C_1
C_2

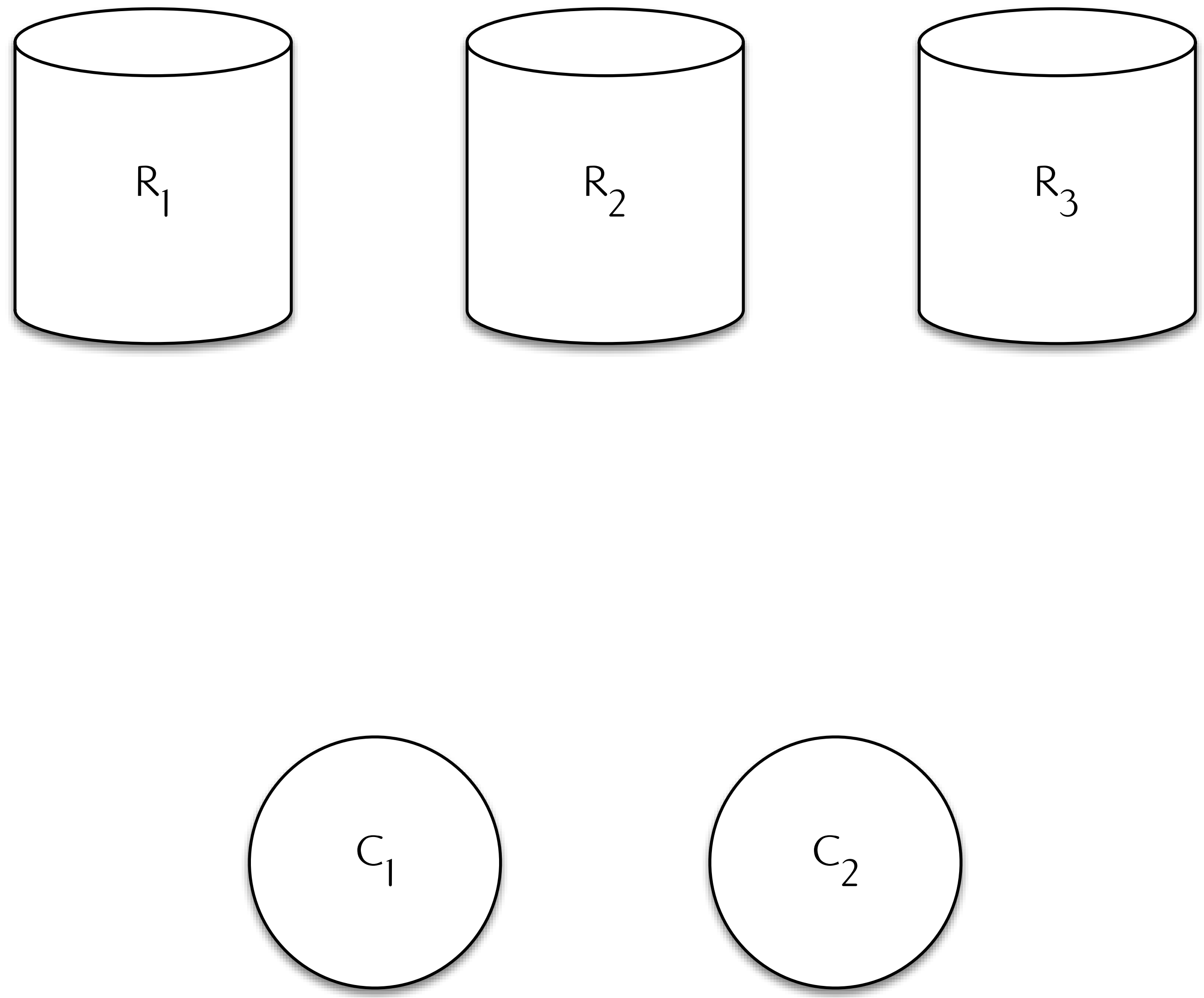

R1
CAS
C1
C2

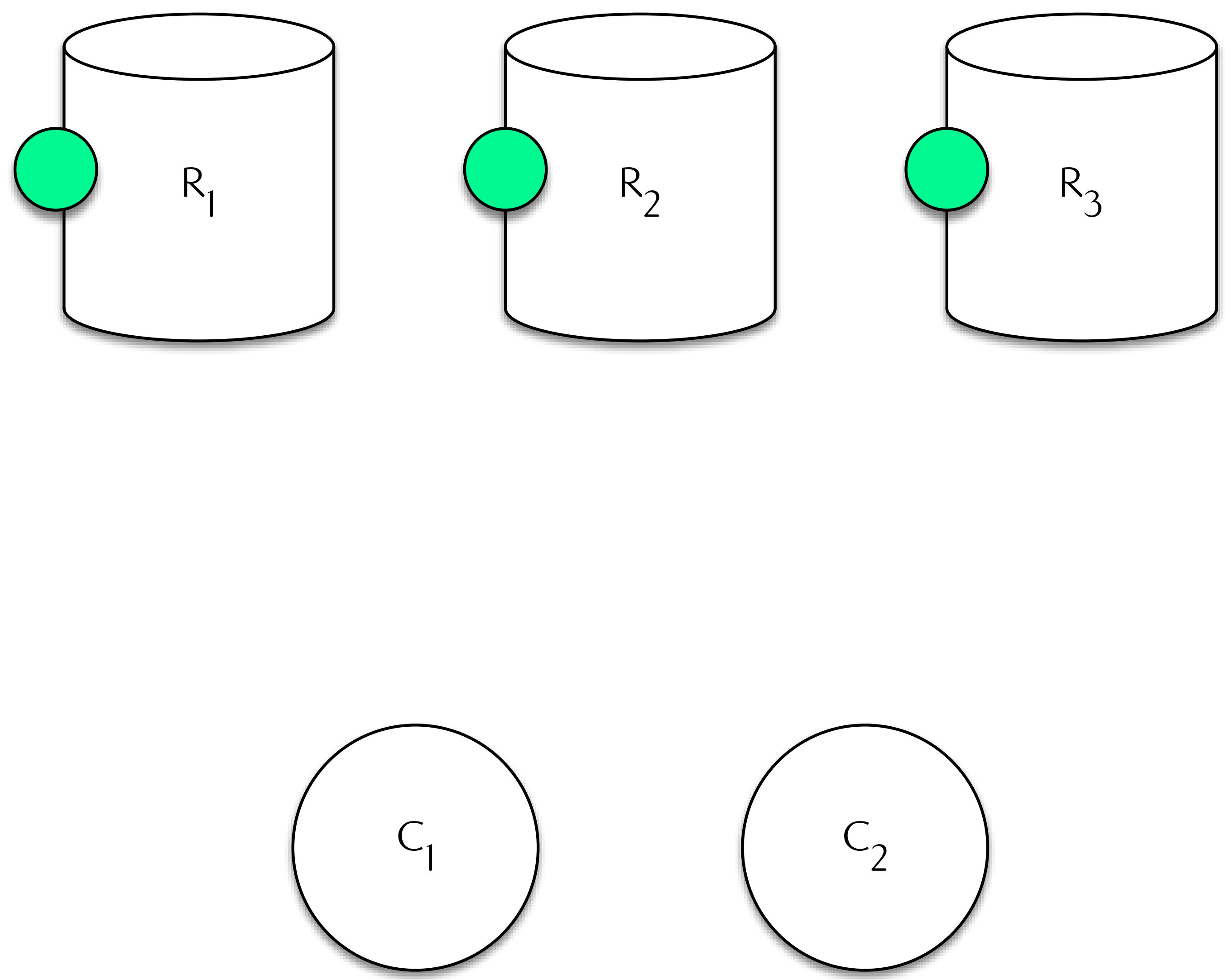
I won’t diagram
the **Paxos** protocol

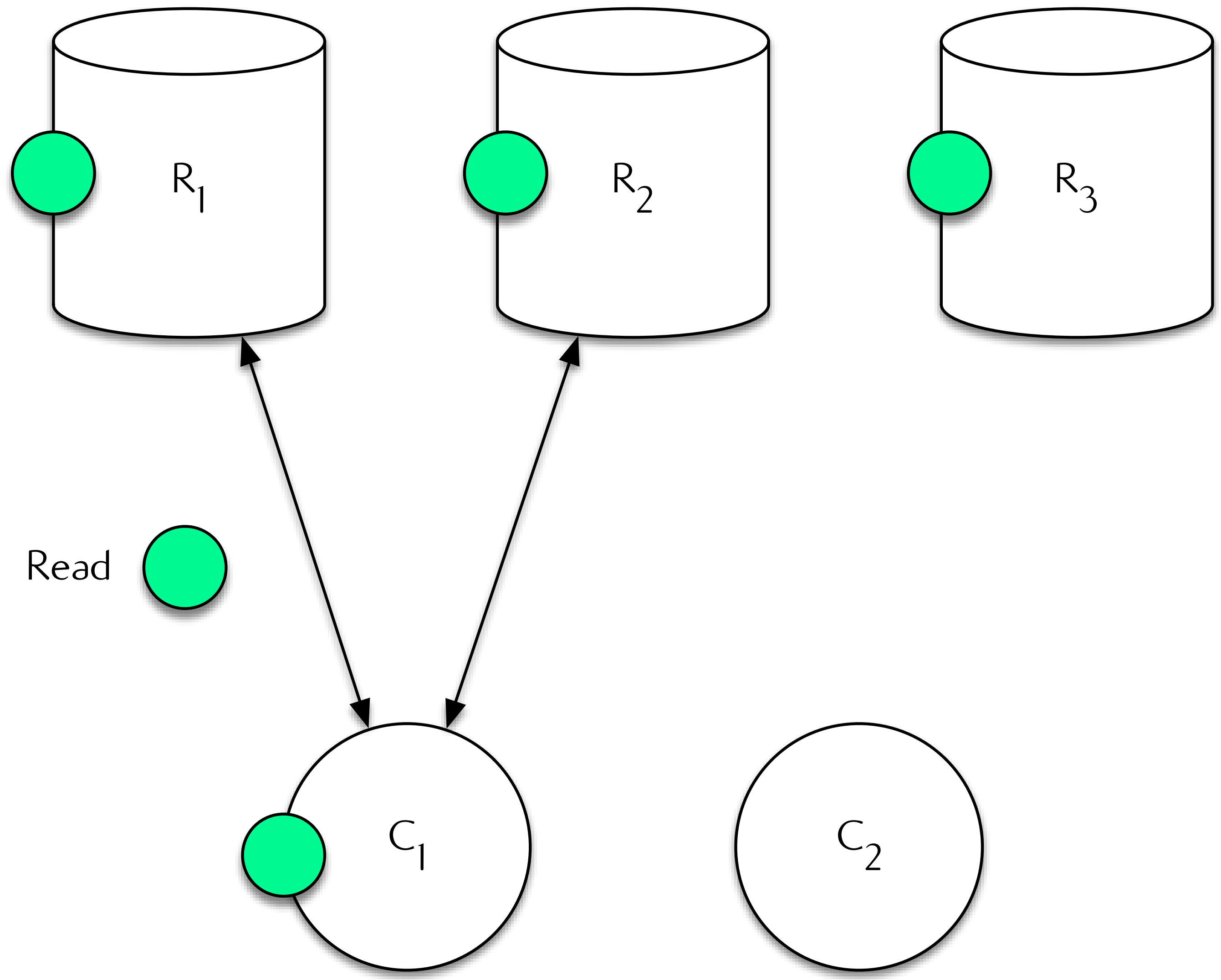

Value 2
R_1
R_2
R_3
Paxos
C_1
C_2
Value 2
Value 1

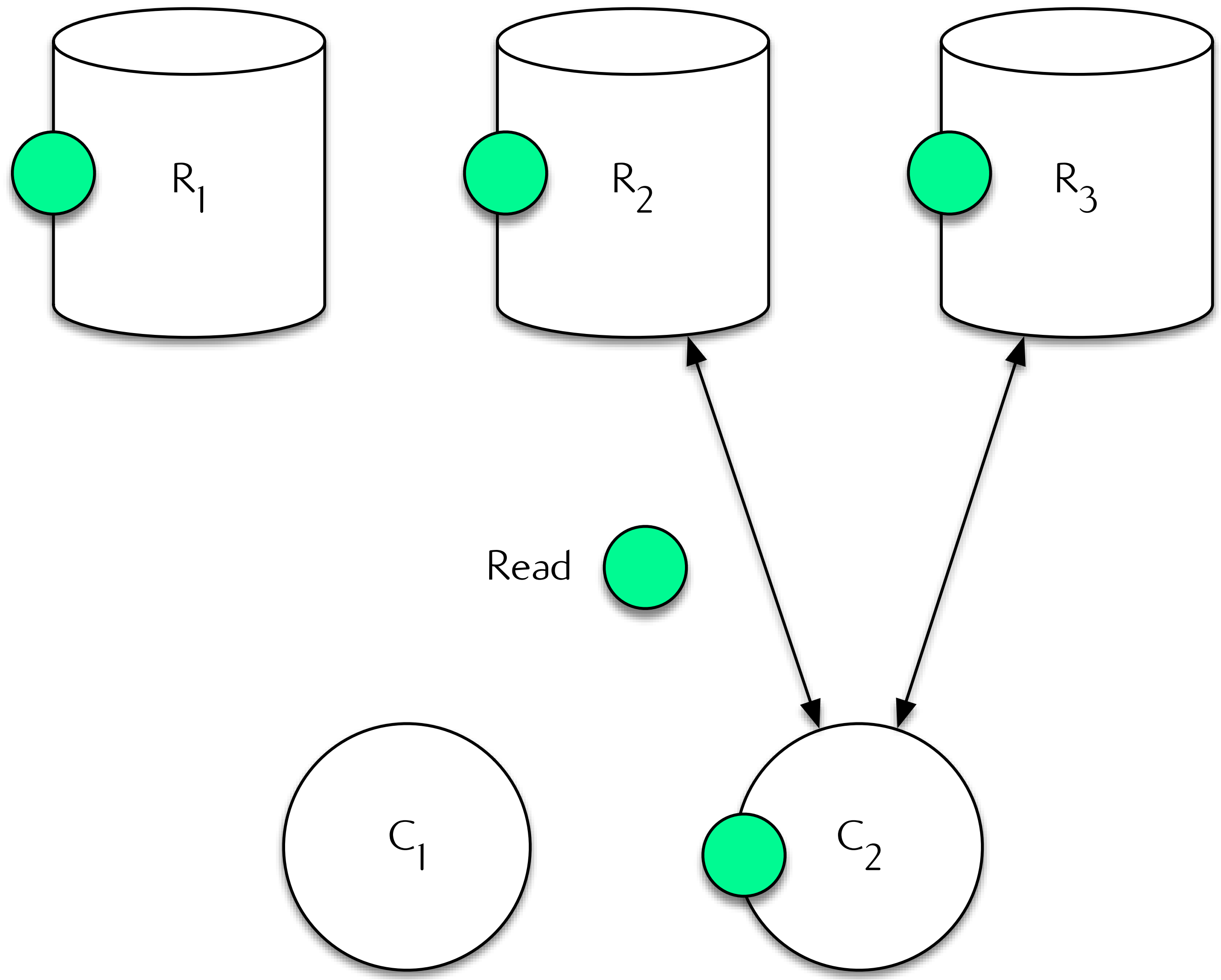
Databases

Eventual Consistency

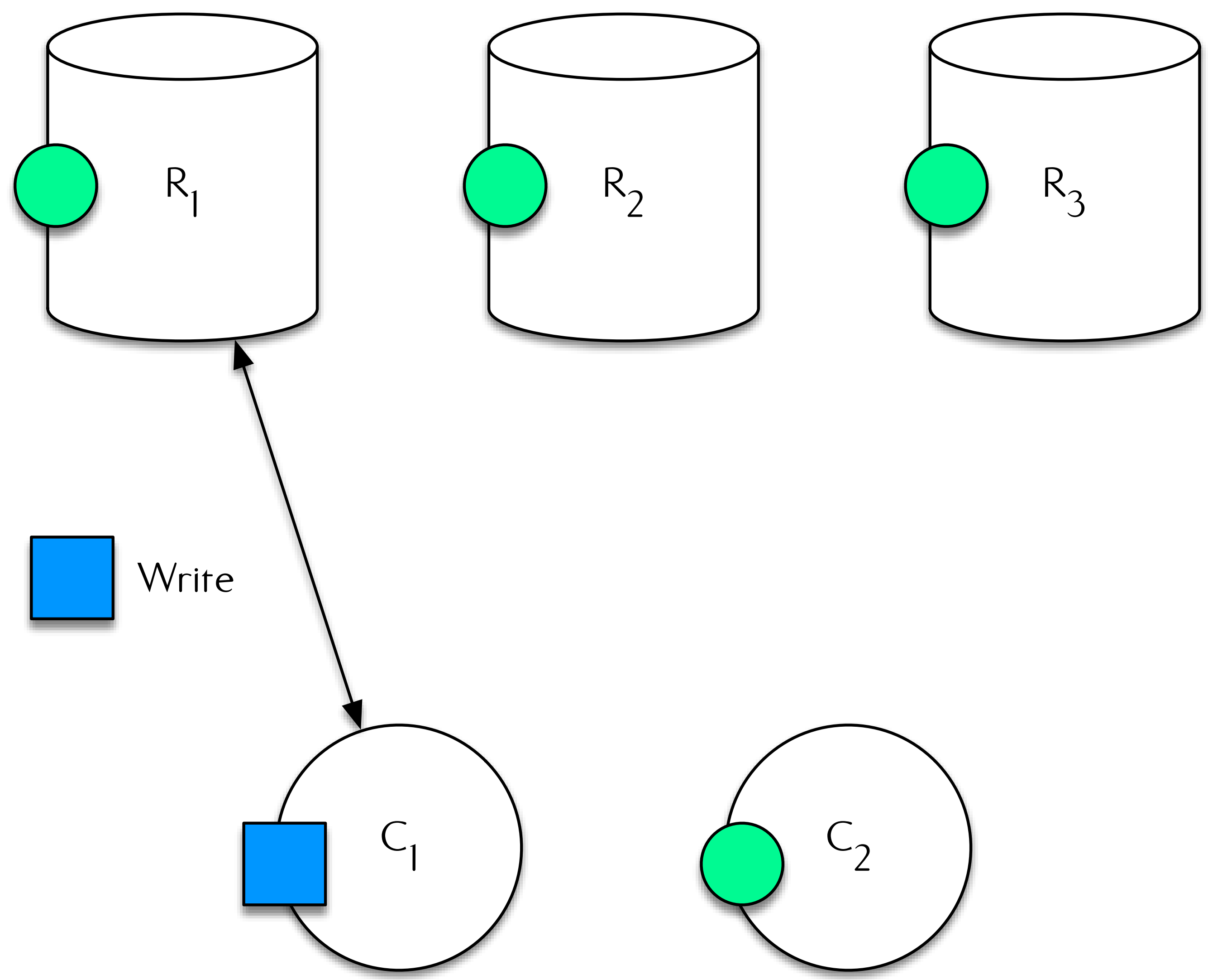

R_1
R_2
R_3
C_1
C_2

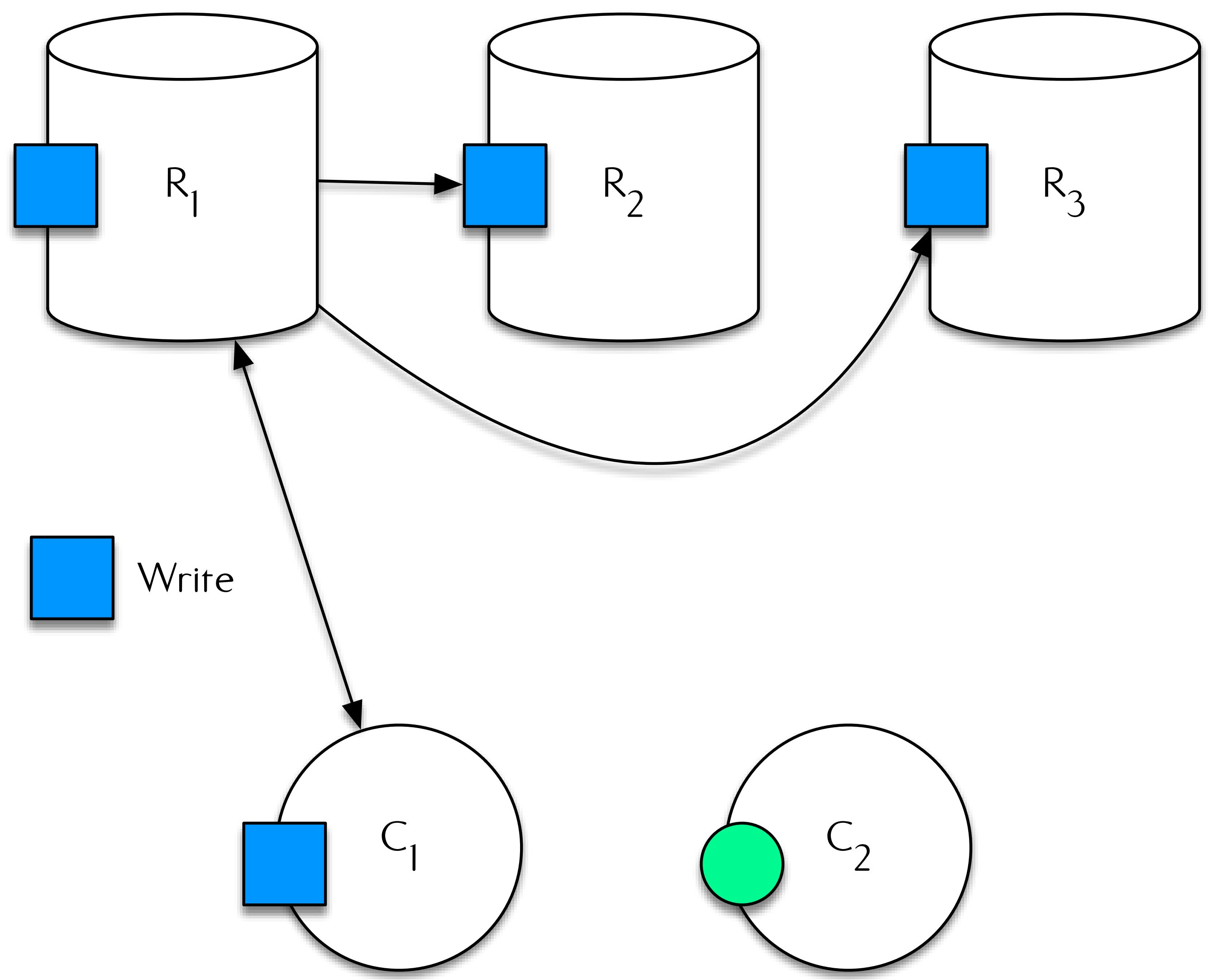

R_1
R_2
R_3
C_1
C_2


R1
R2
R3
Read
C1
C2

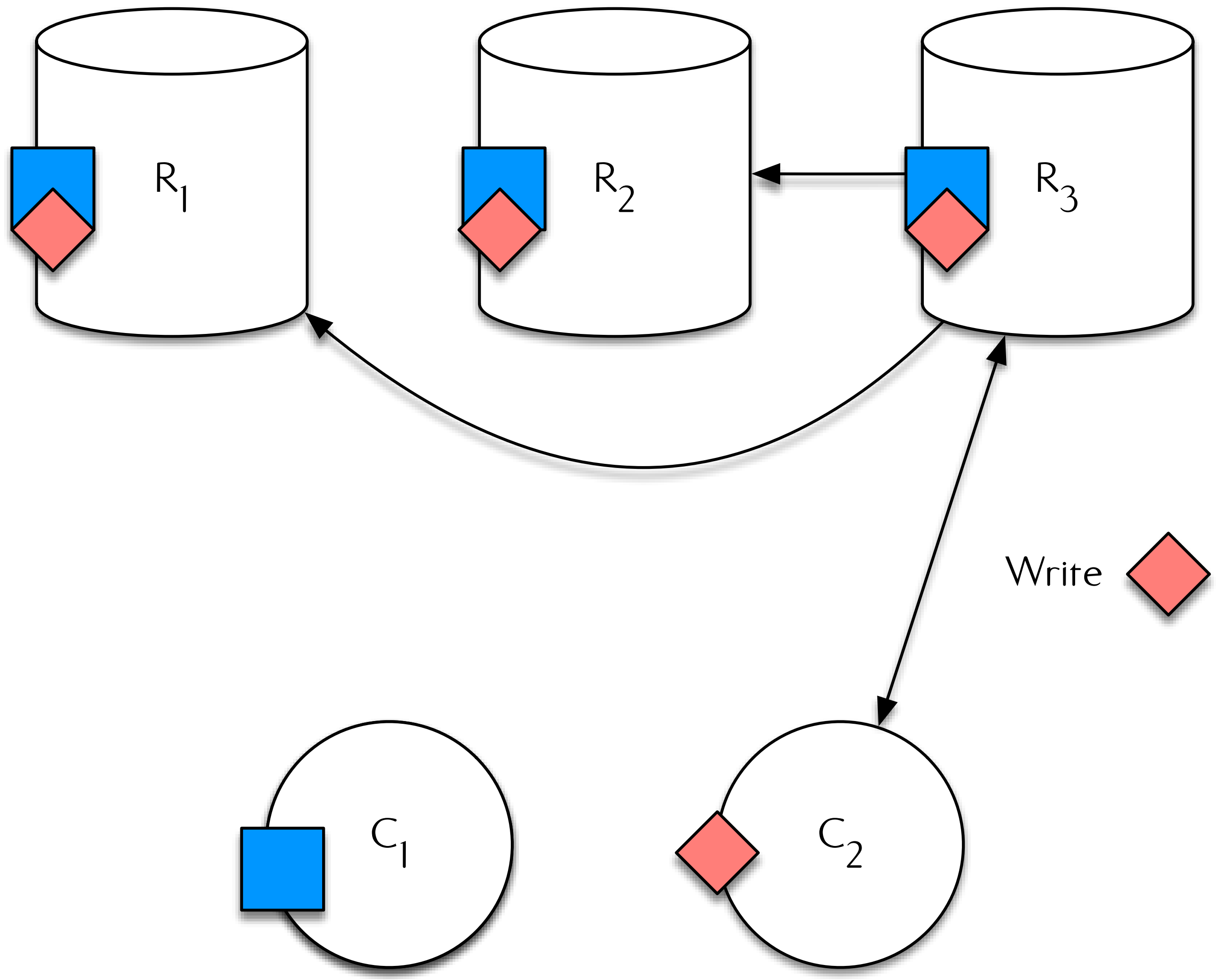


R_1
R_2
R_3
Read
C_1
C_2

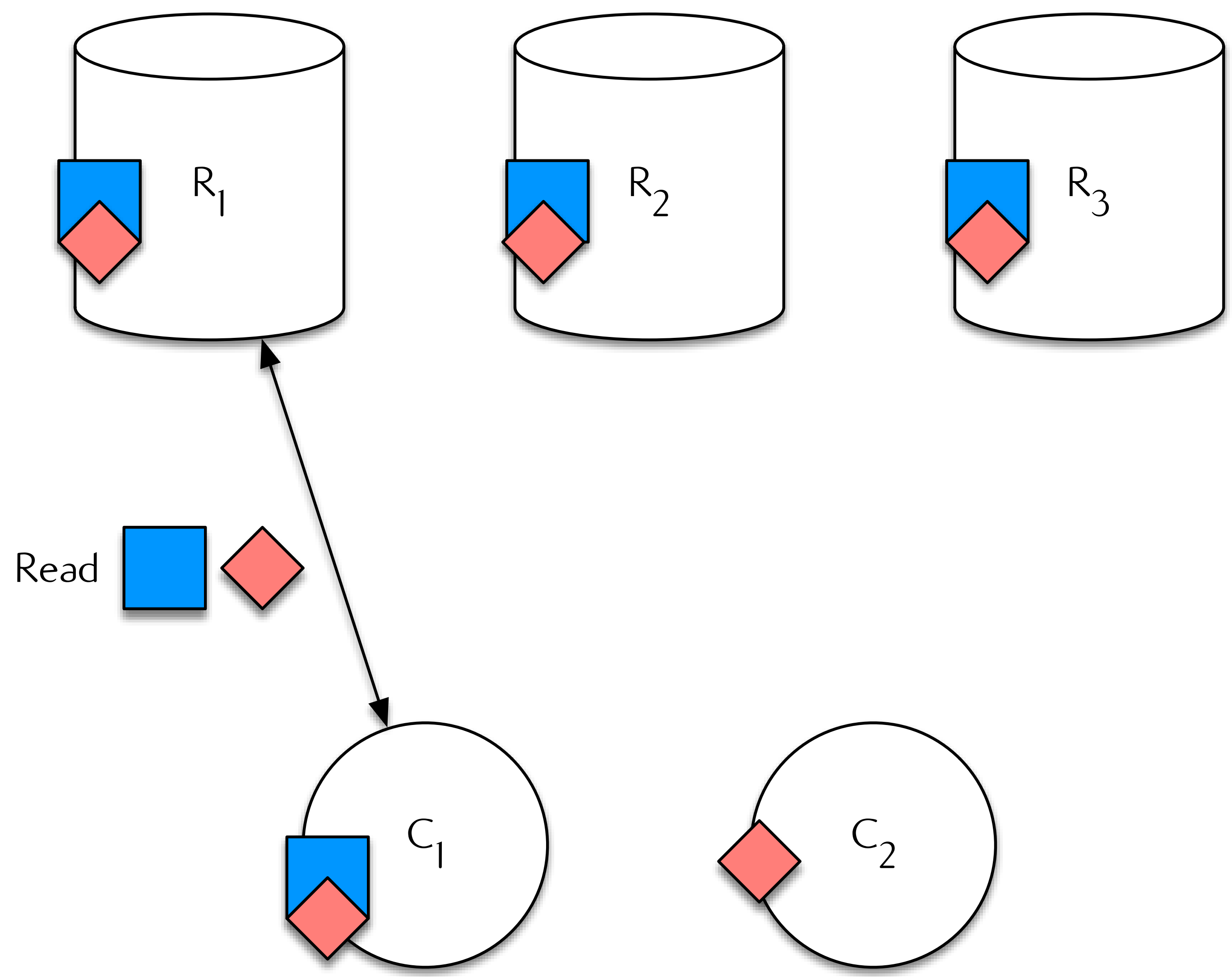

R1
R2
R3
Write
C1
C2

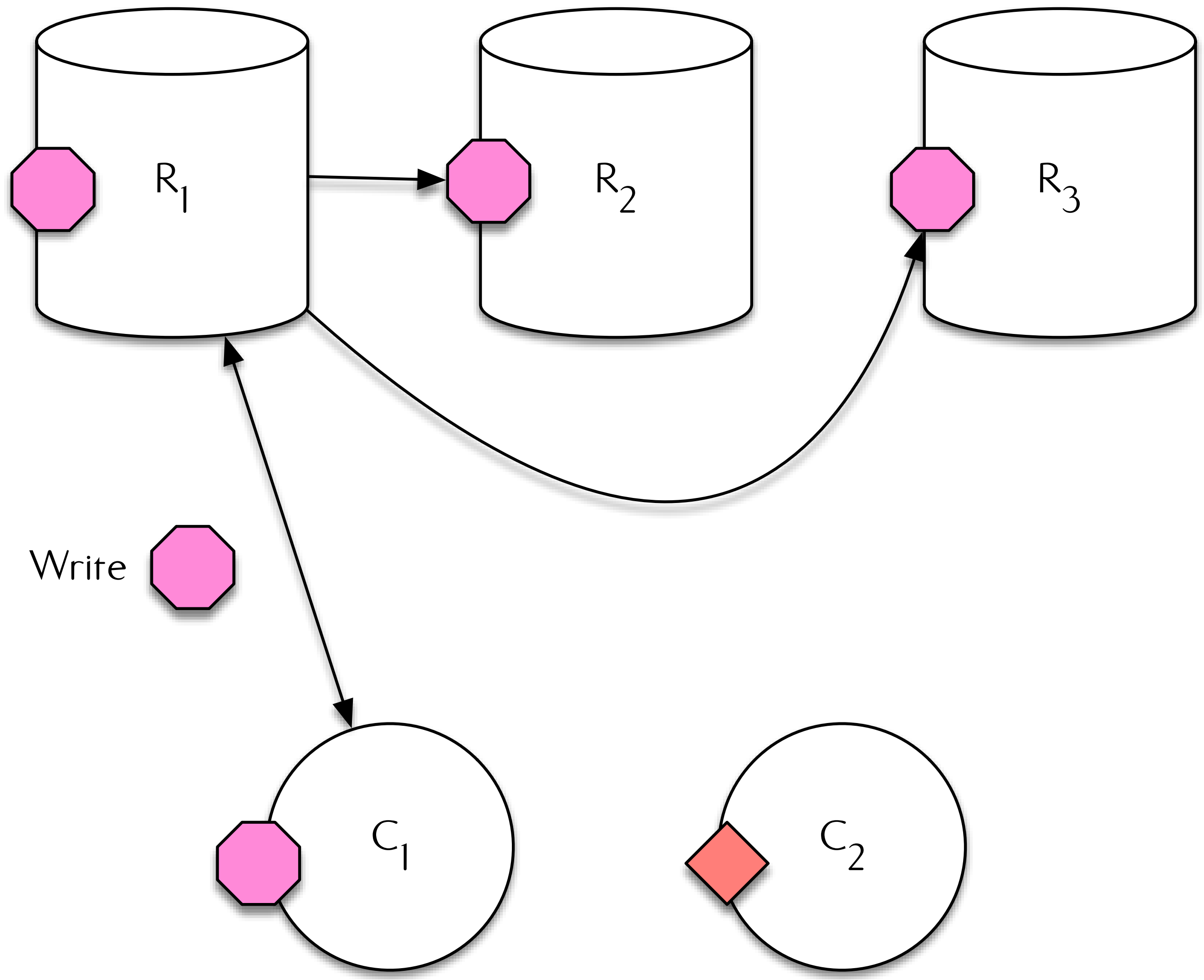


R_1
R_2
R_3
Write
C_1
C_2



R1
R2
R3
Write
C1
C2

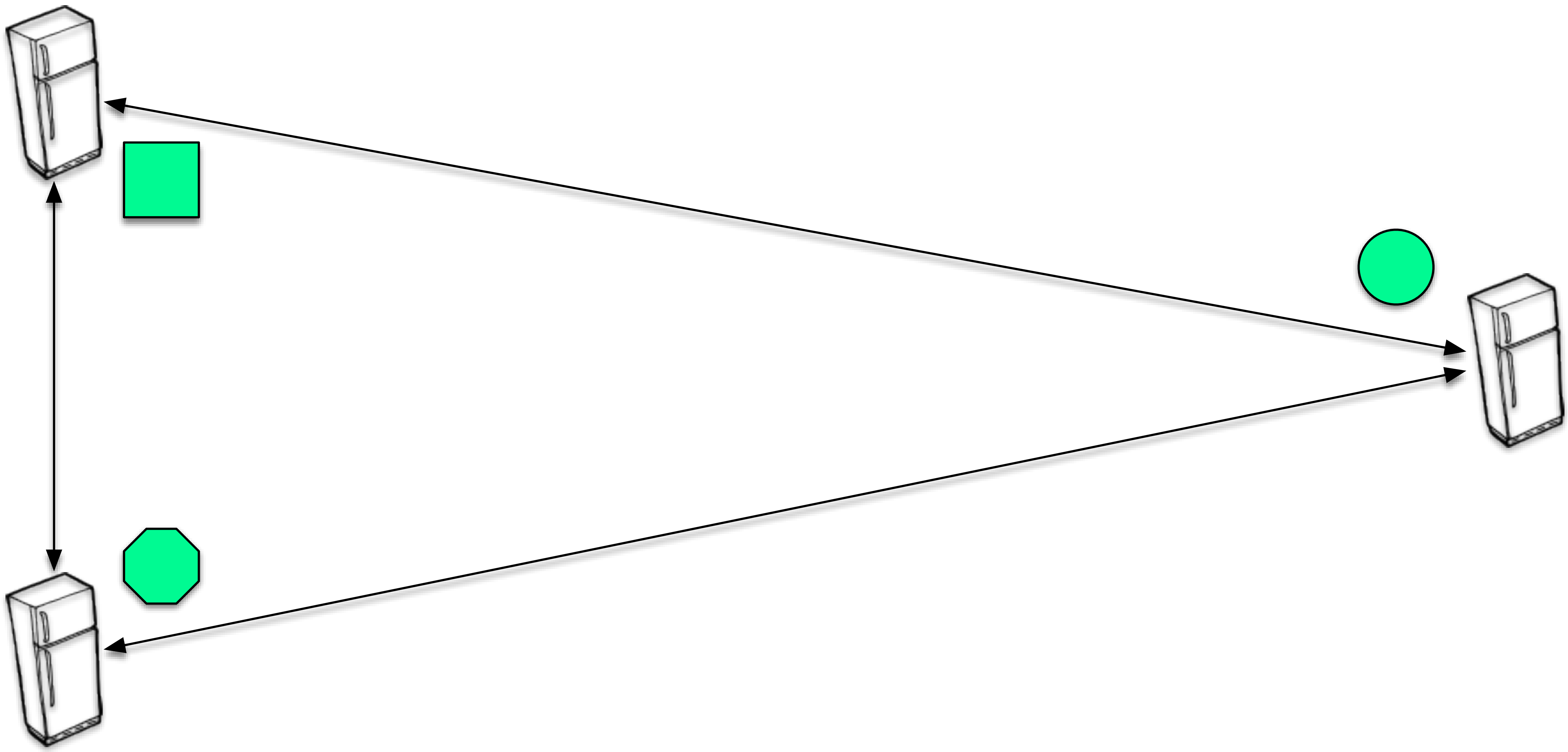

R1
R2
R3
Read
C1
C2

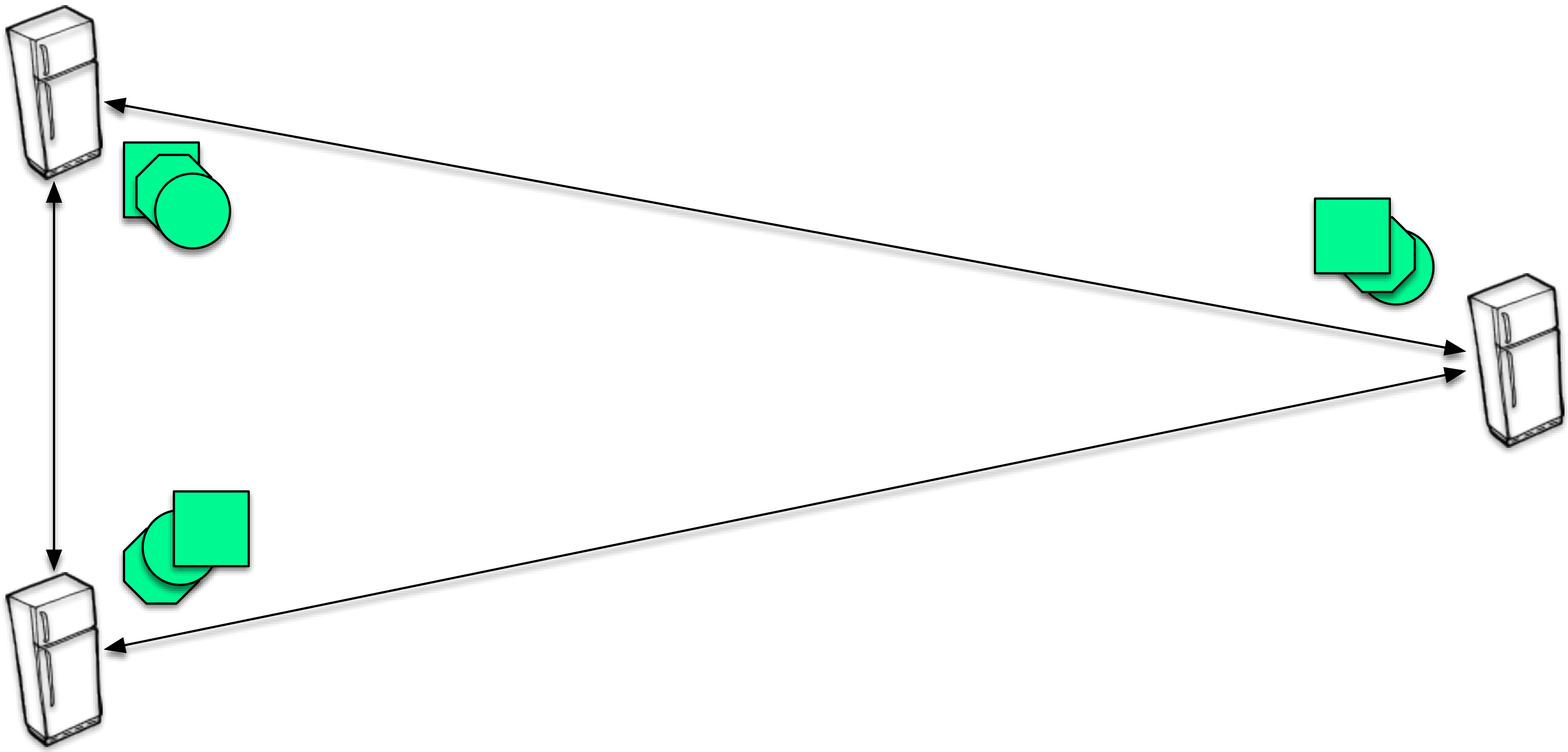


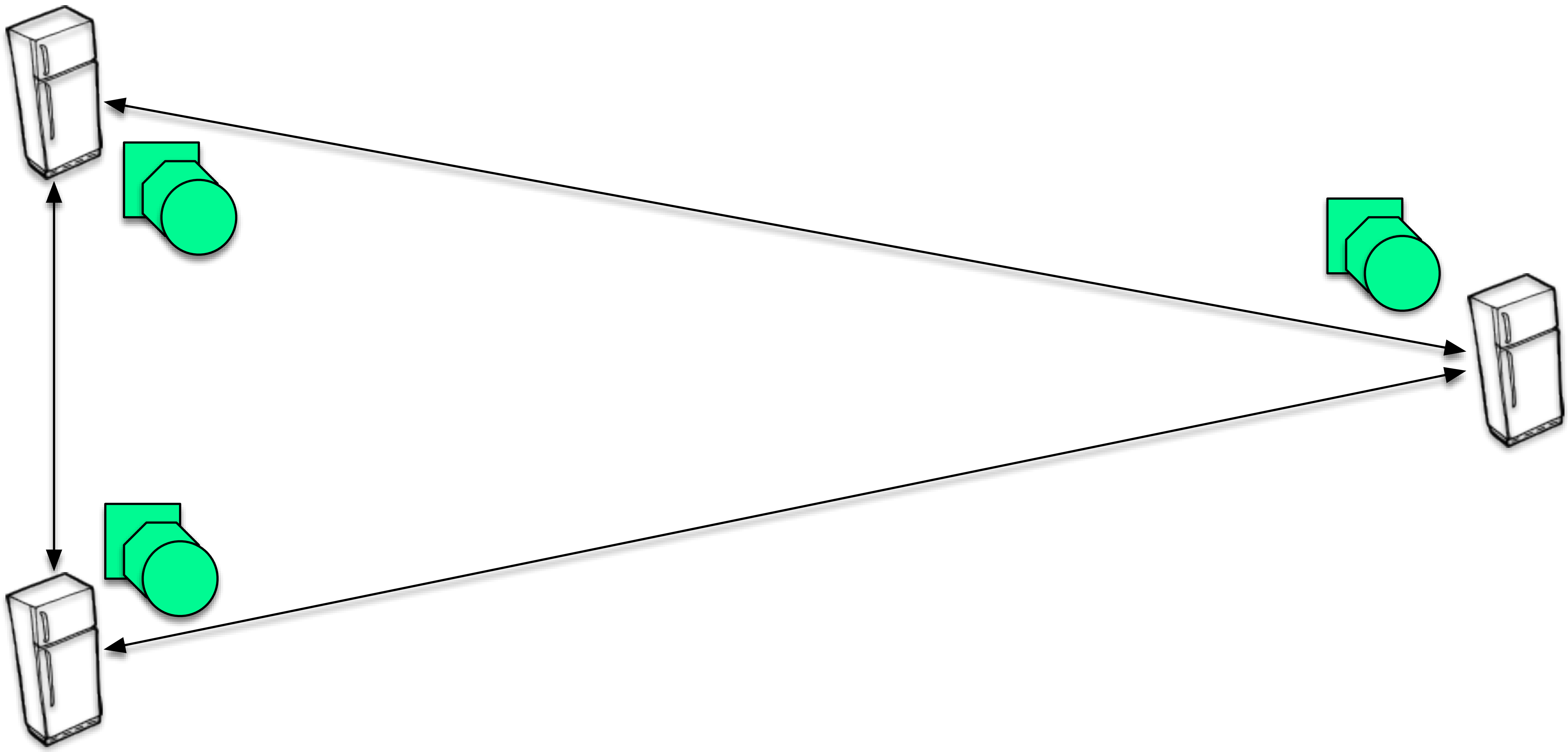
R_1
R_2
R_3
Write
C_1
C_2

Eventual Consistency
As The Model

Clients
Own Their Data

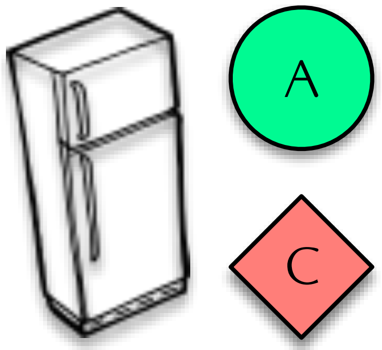


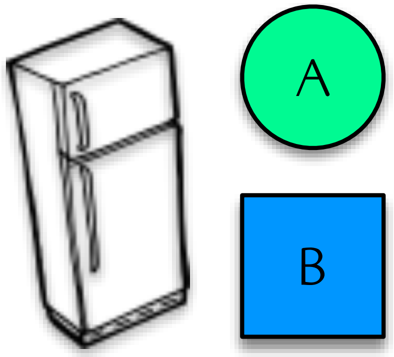


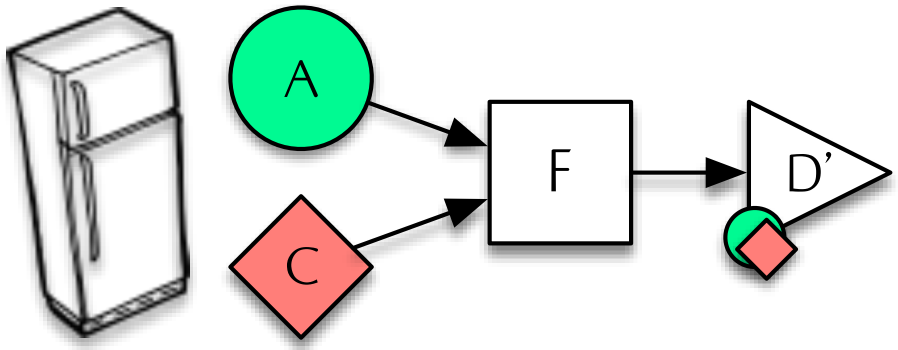


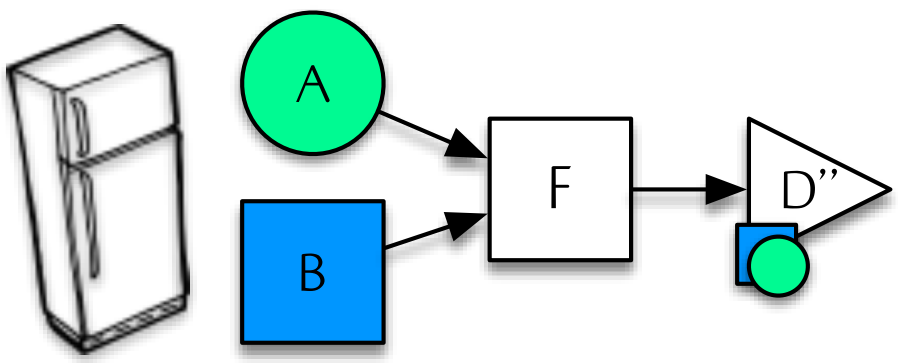
Computations

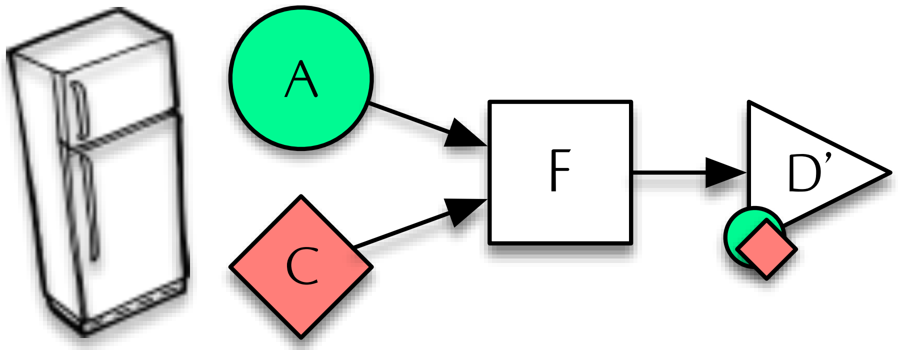
Mergeability & Provenance

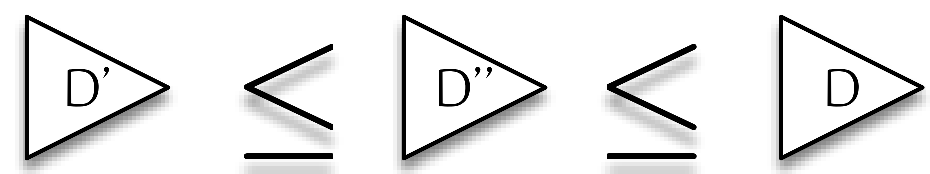

A
C

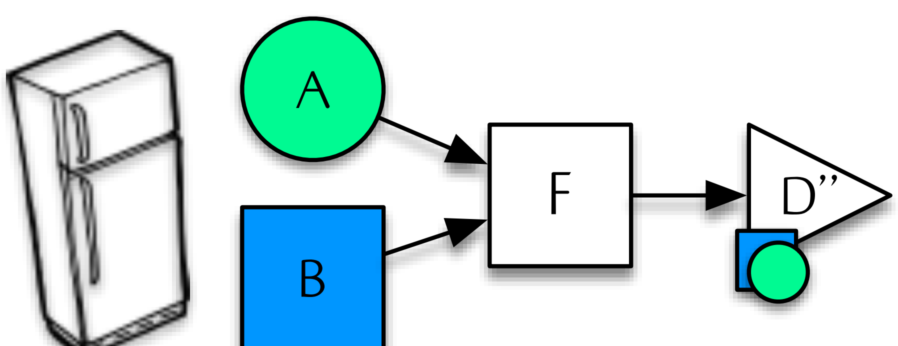

A
B

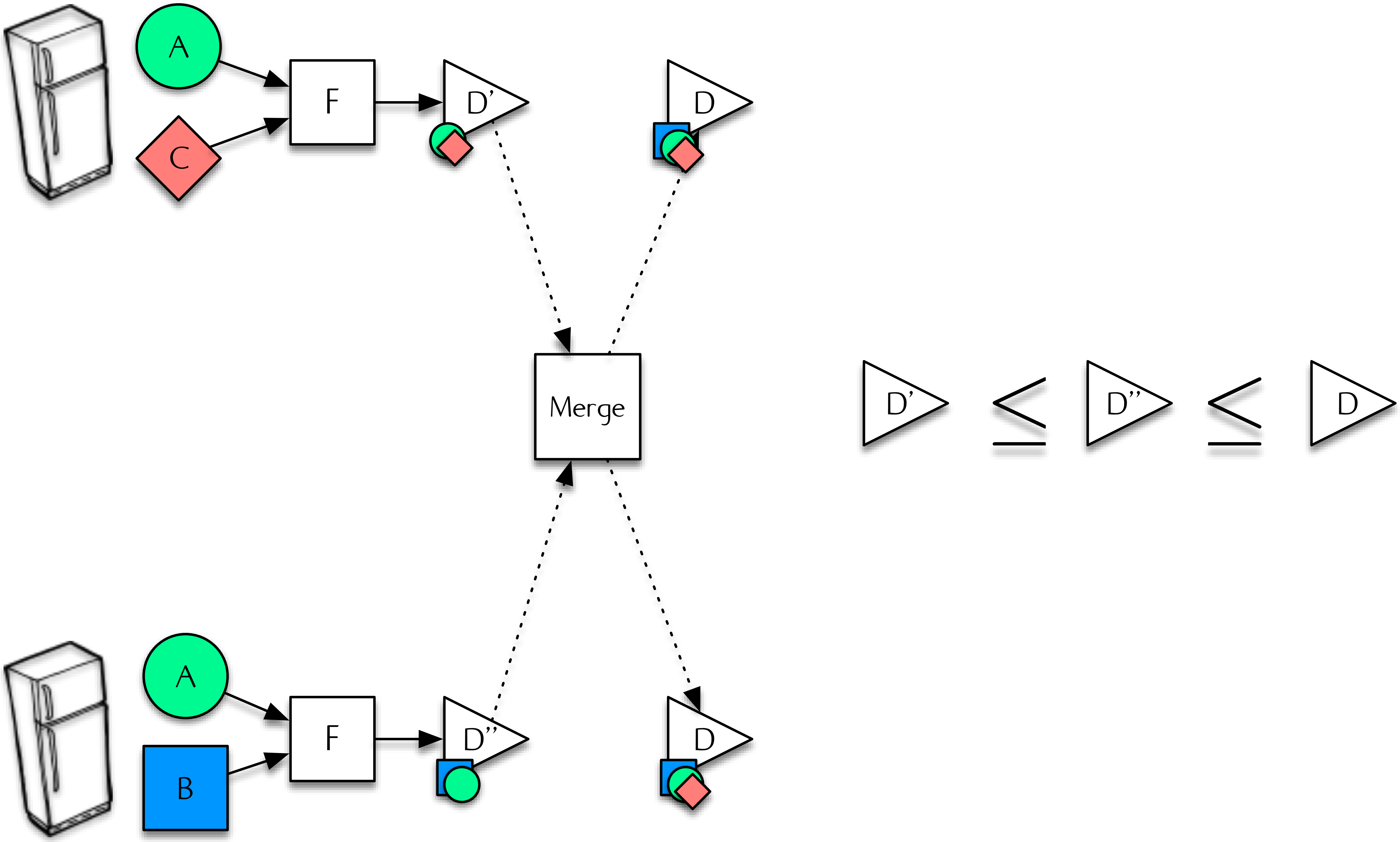

A
C
F
D’


A
B
F
D’’


A
C
F
D’


D’
≤
D’’
≤
D


A
B
F
D’’


A
C
F
D’
D
Merge
D’
≤
D’’
≤
D
A
B
F
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]

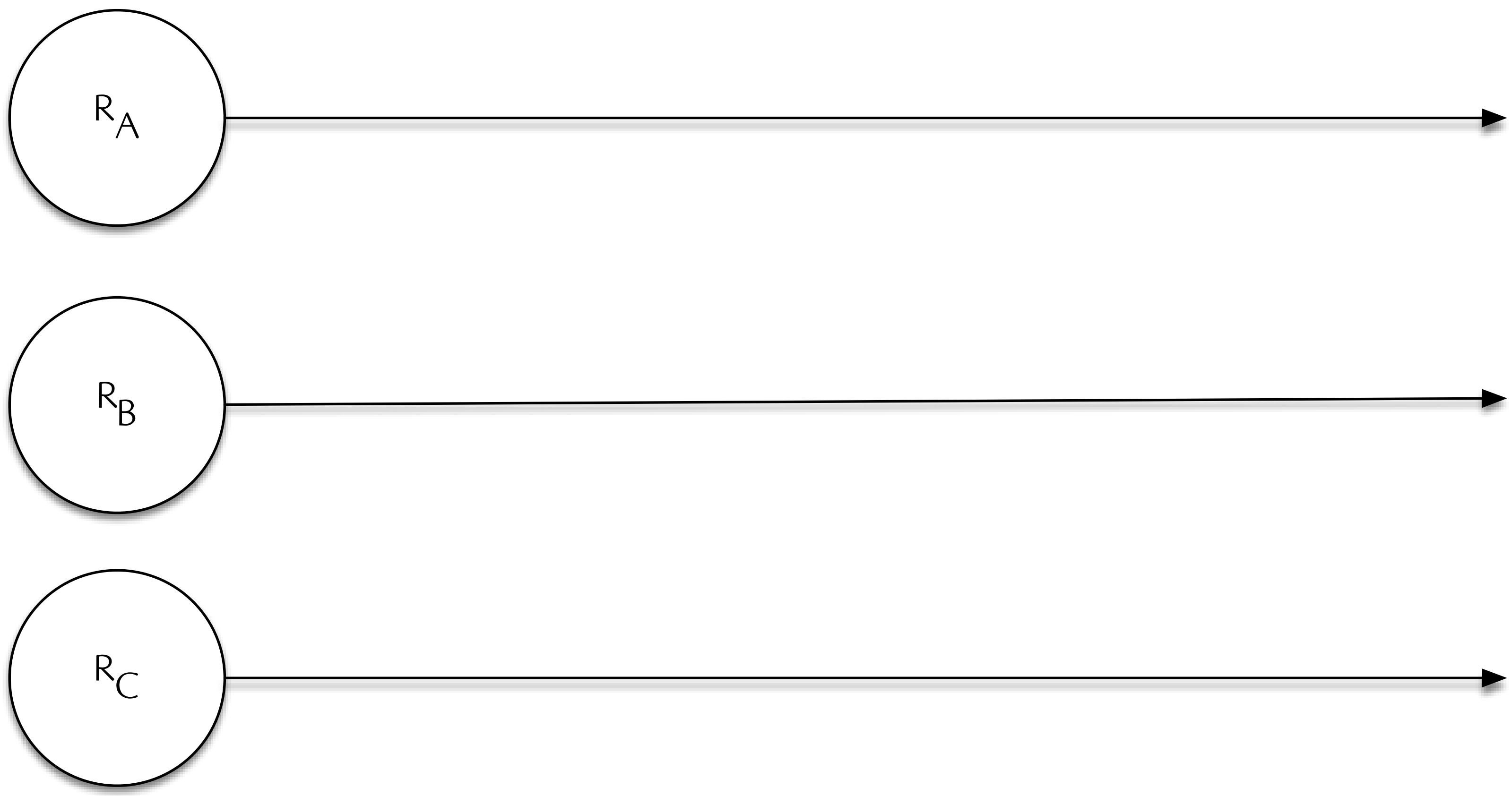
- Selective Hearing
  Scalable, epidemic broadcast based runtime system
  [Meiklejohn & Van Roy, 2015/2016]

Conflict-Free
Replicated Data Types

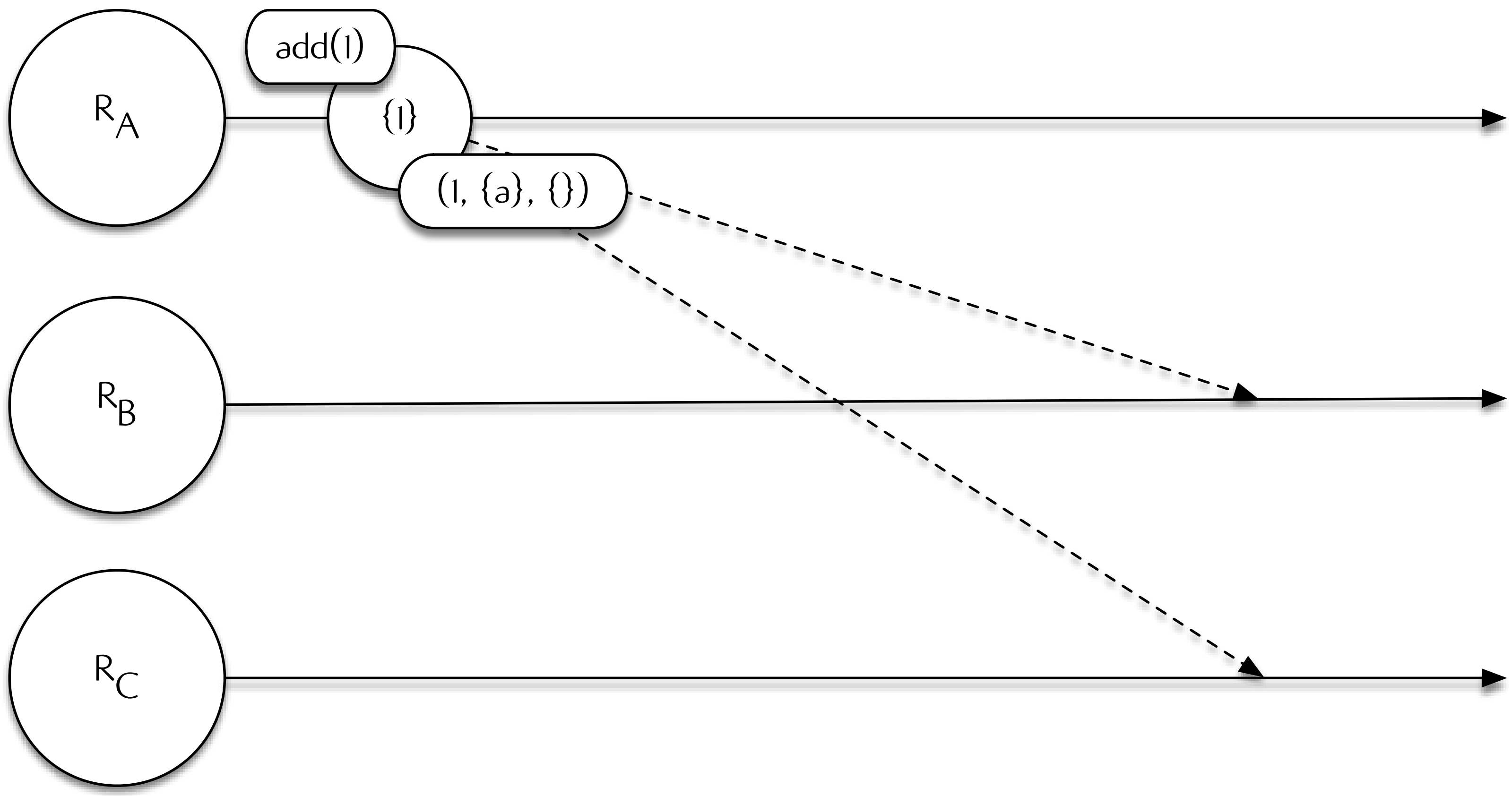
- Collection of types
  Sets, counters, registers, flags, maps

Conflict-Free
Replicated Data Types

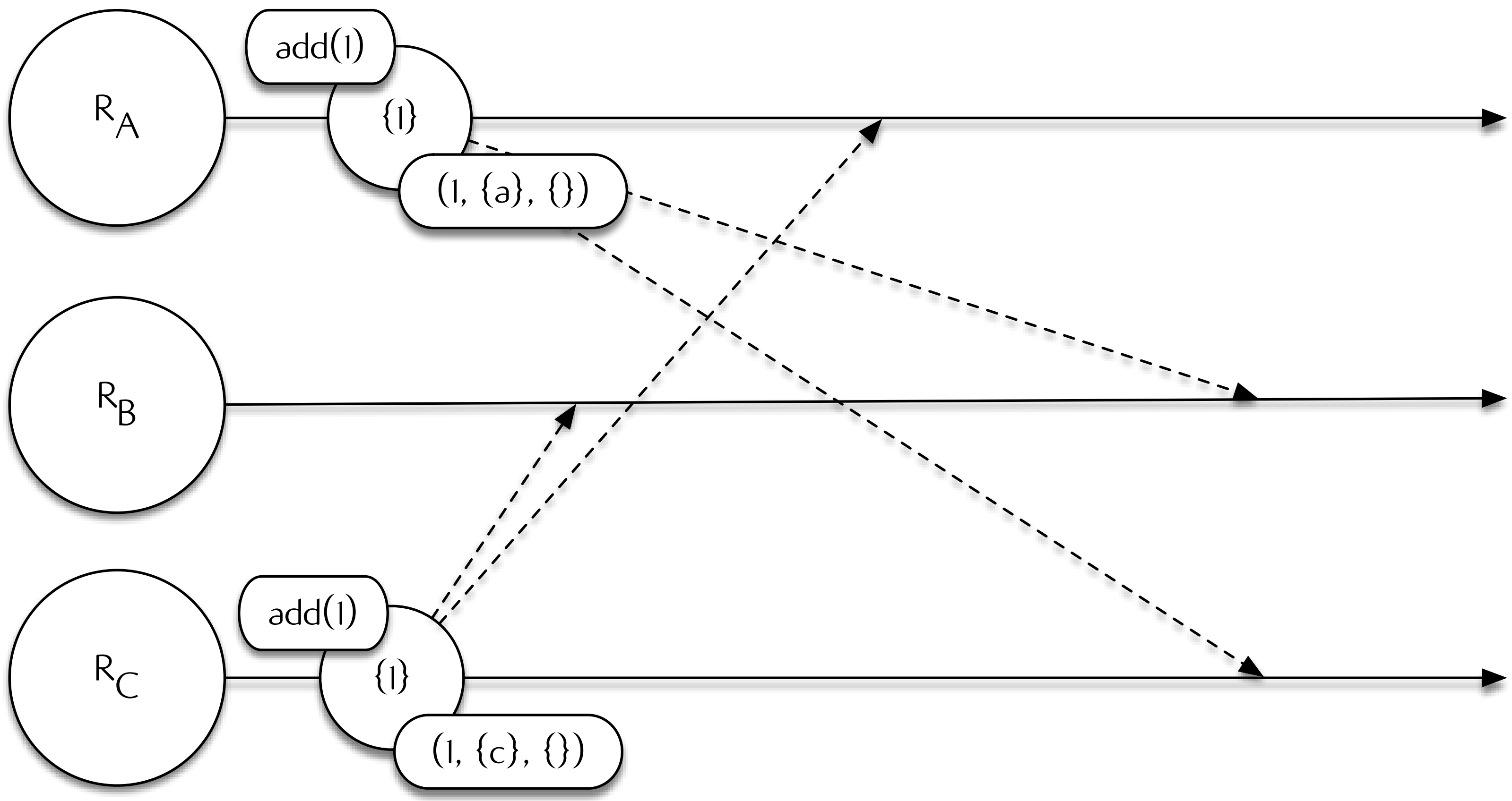
- **Collection of types**
  Sets, counters, registers, flags, maps
- **Strong Eventual Consistency (SEC)**
  Objects that receive the same updates, regardless of order, will reach equivalent state



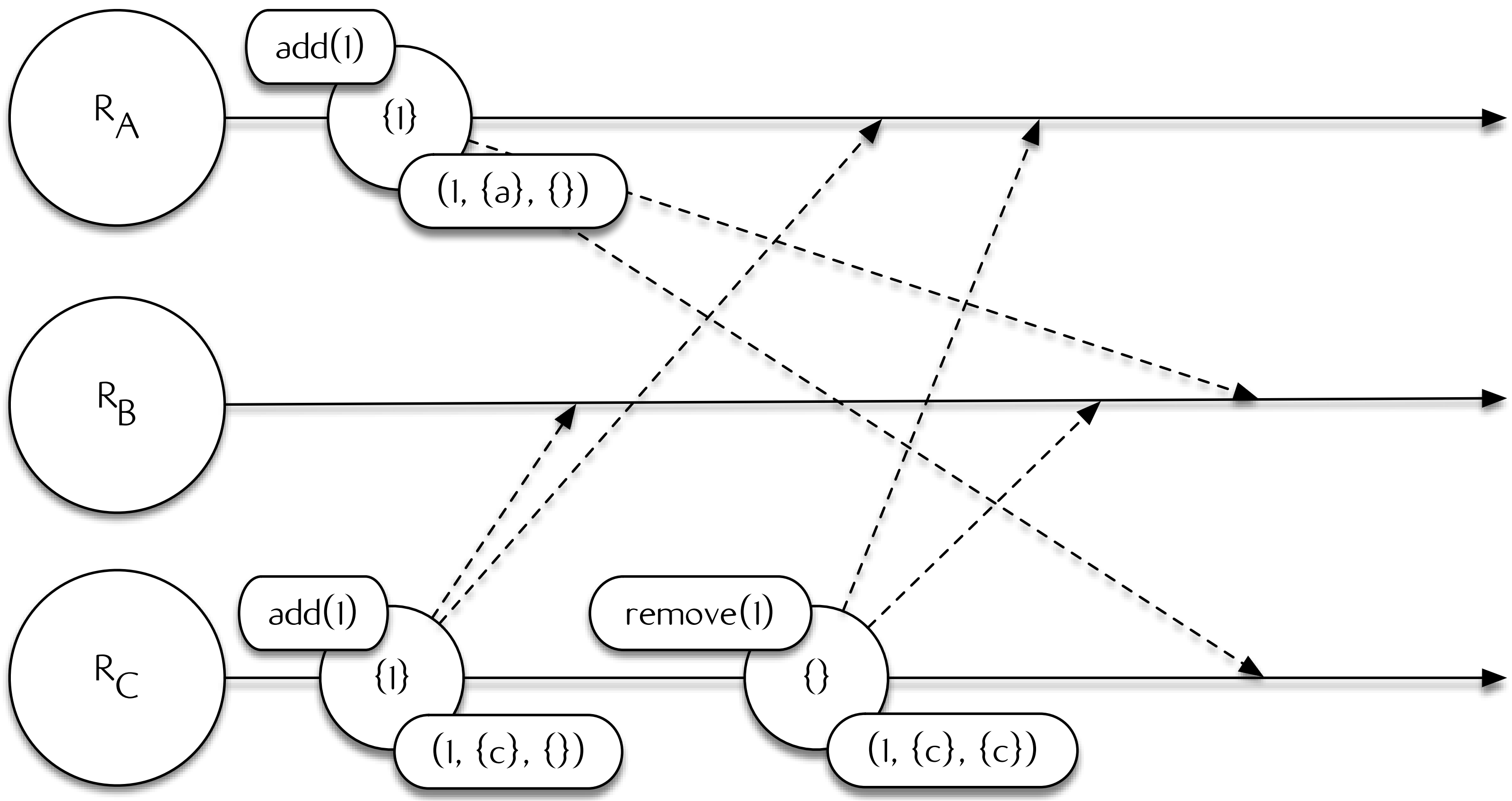
R_A
R_B
R_C

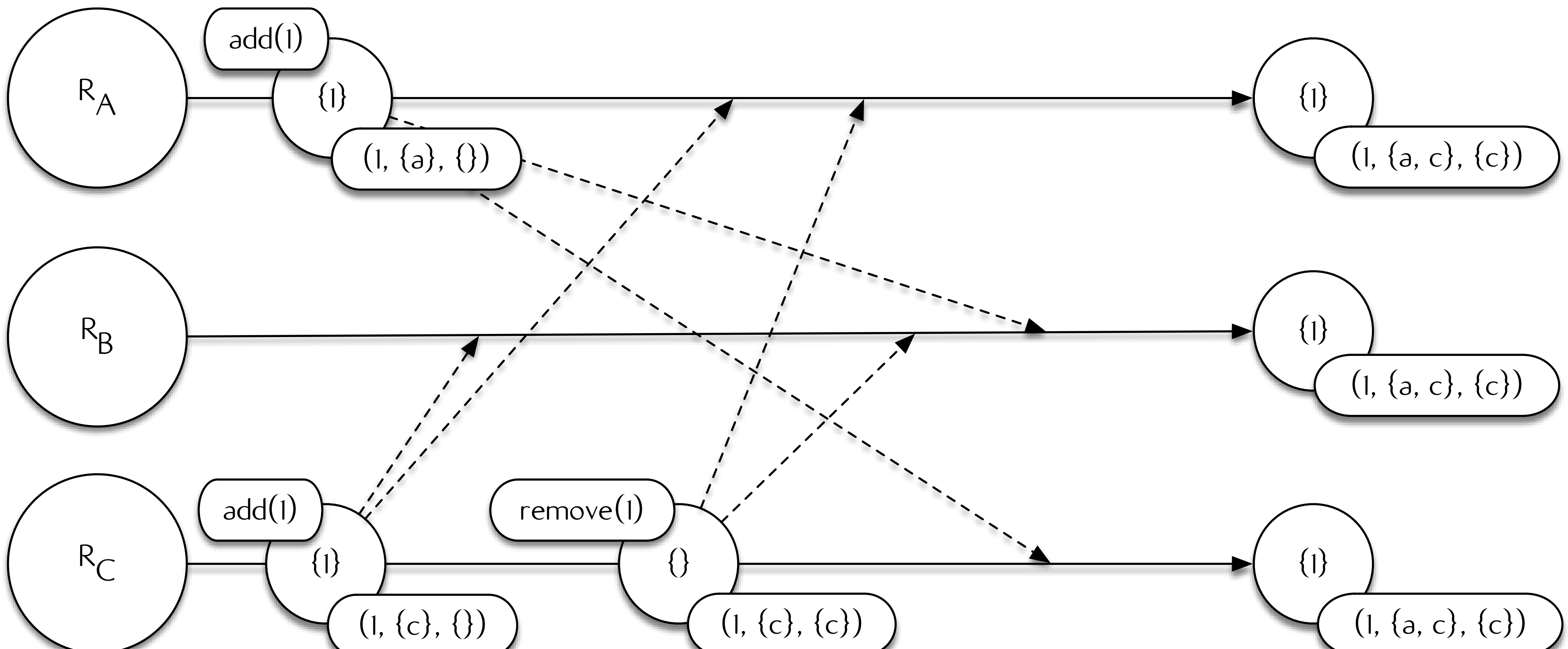


add(1)
{1}
(1, {a}, {})
R_A
R_B
R_C



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


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



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

Lattice Processing

- Distributed dataflow
  Declarative, functional programming model

Lattice Processing

- **Distributed dataflow**
  Declarative, functional programming model

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

Lattice Processing

- Distributed dataflow
  Declarative, functional 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).
```

```
%% 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).
```

```
%% 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).
```

Delta-based **Dissemination**

- Delta-state based CRDTs
  Reduces state transmission for clients

Delta-based **Dissemination**

- Delta-state based CRDTs
  Reduces state transmission for clients

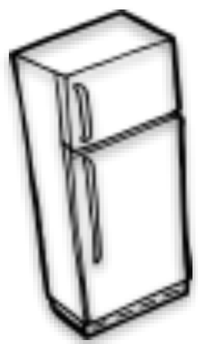
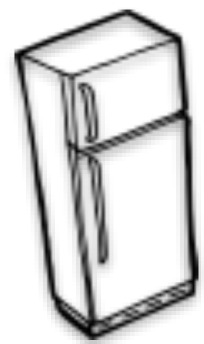
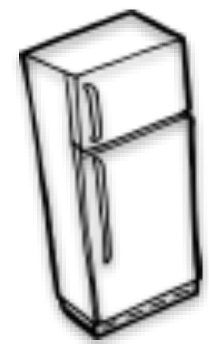
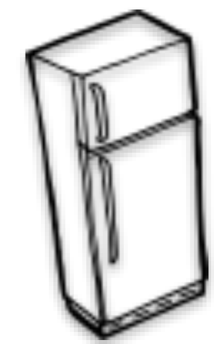
- Operate locally
  Objects are mutated locally; deltas buffered locally and periodically gossiped

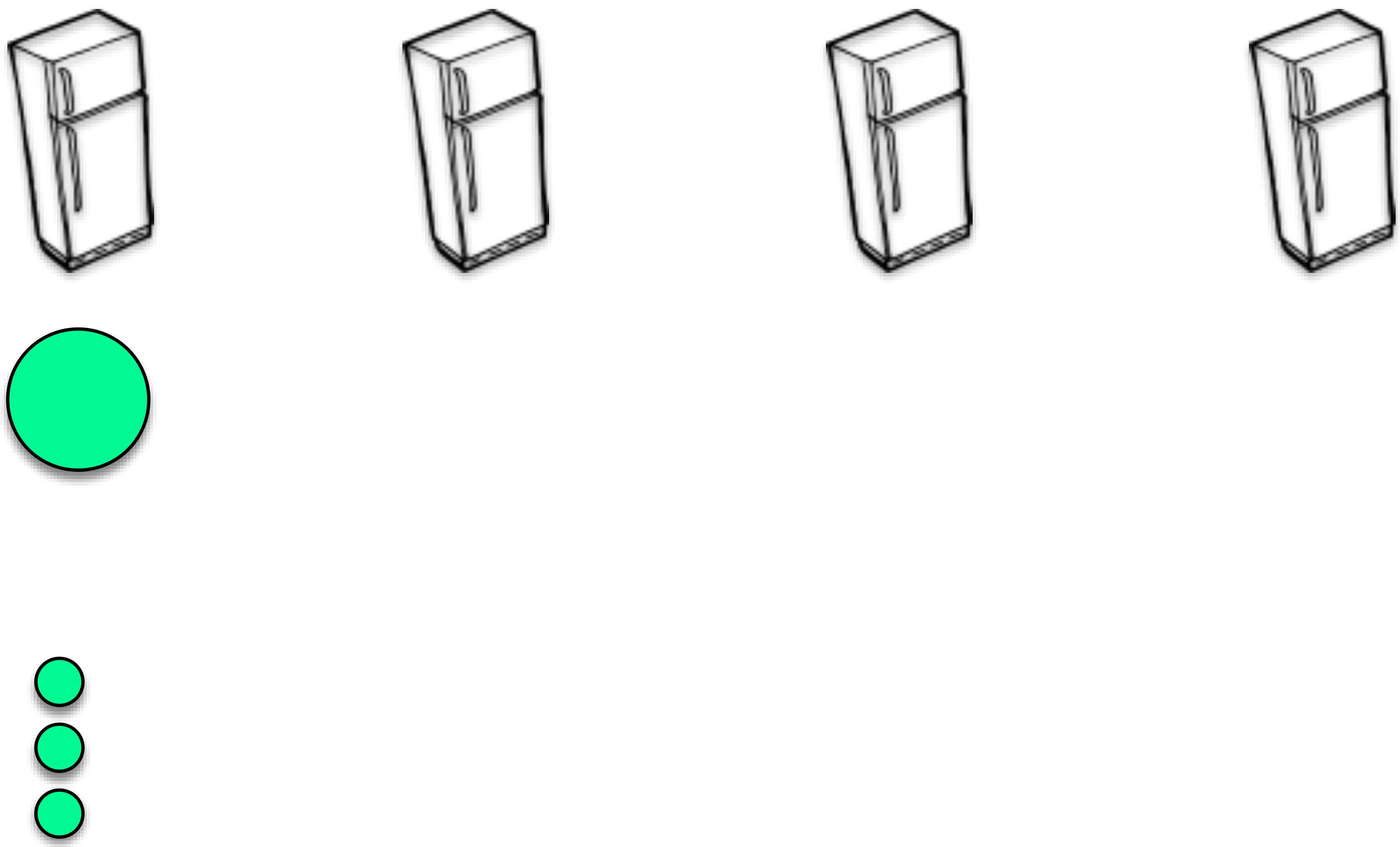
Delta-based **Dissemination**

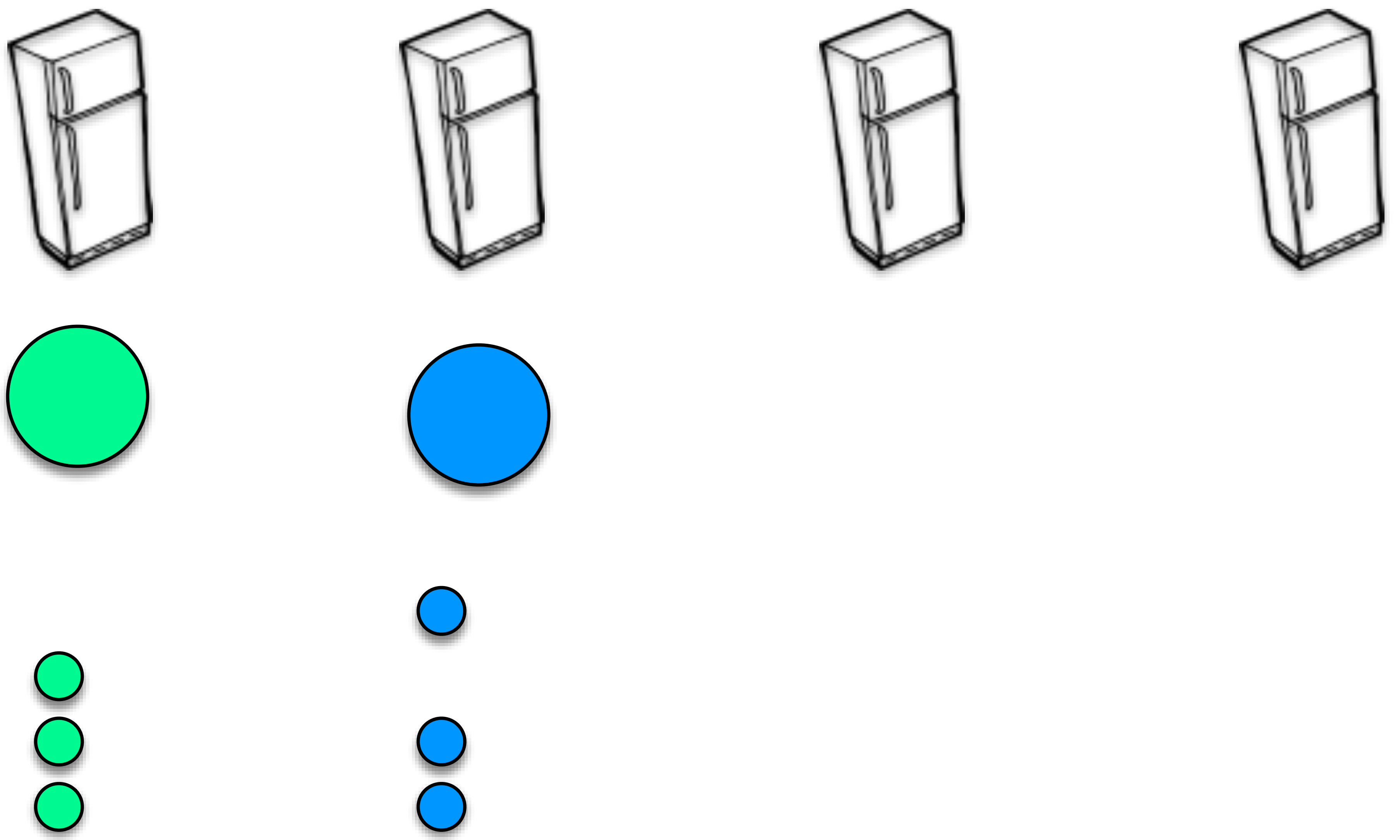
- Delta-state based CRDTs
  Reduces state transmission for clients

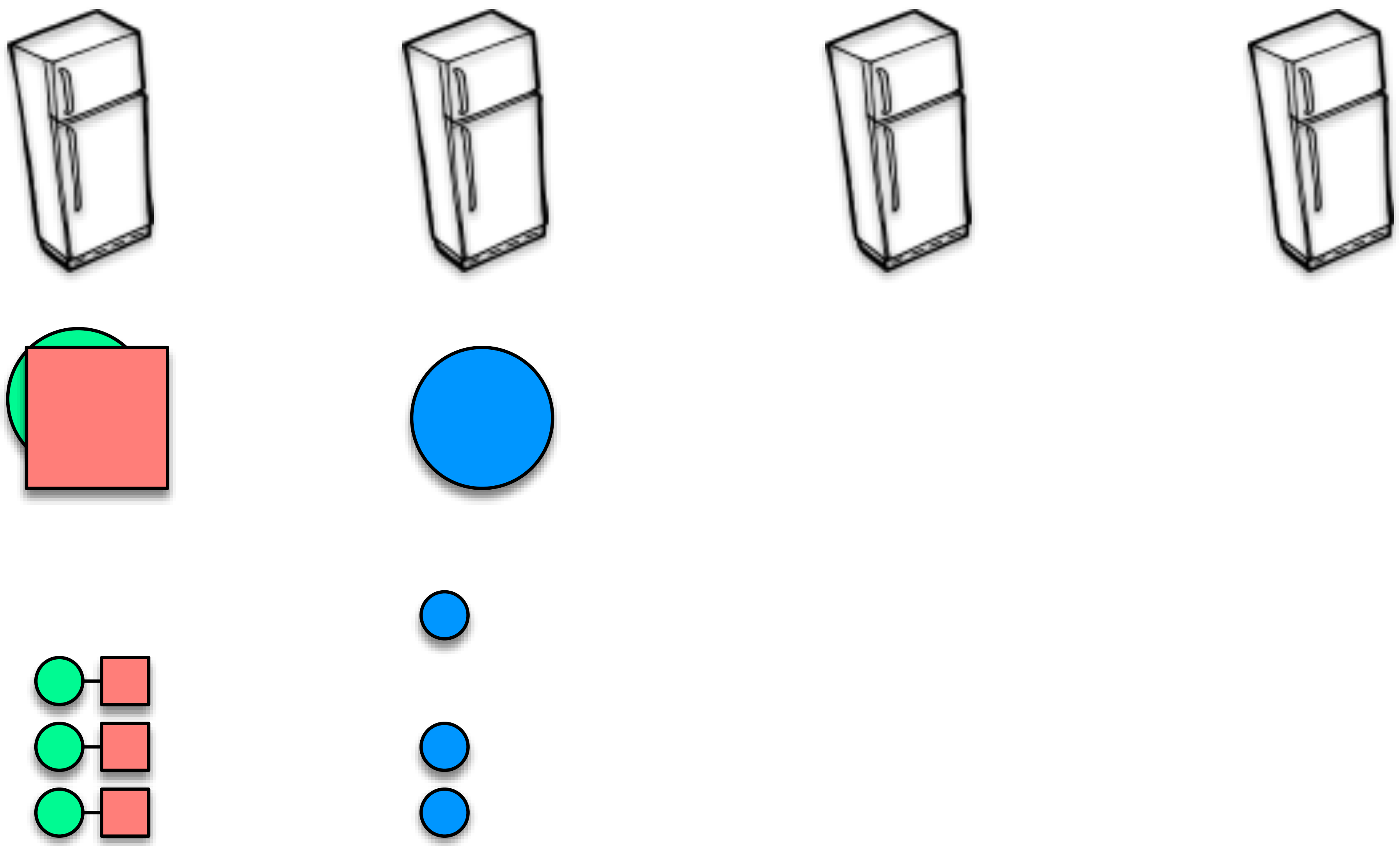
- Operate locally
  Objects are mutated locally; deltas buffered locally and periodically gossiped

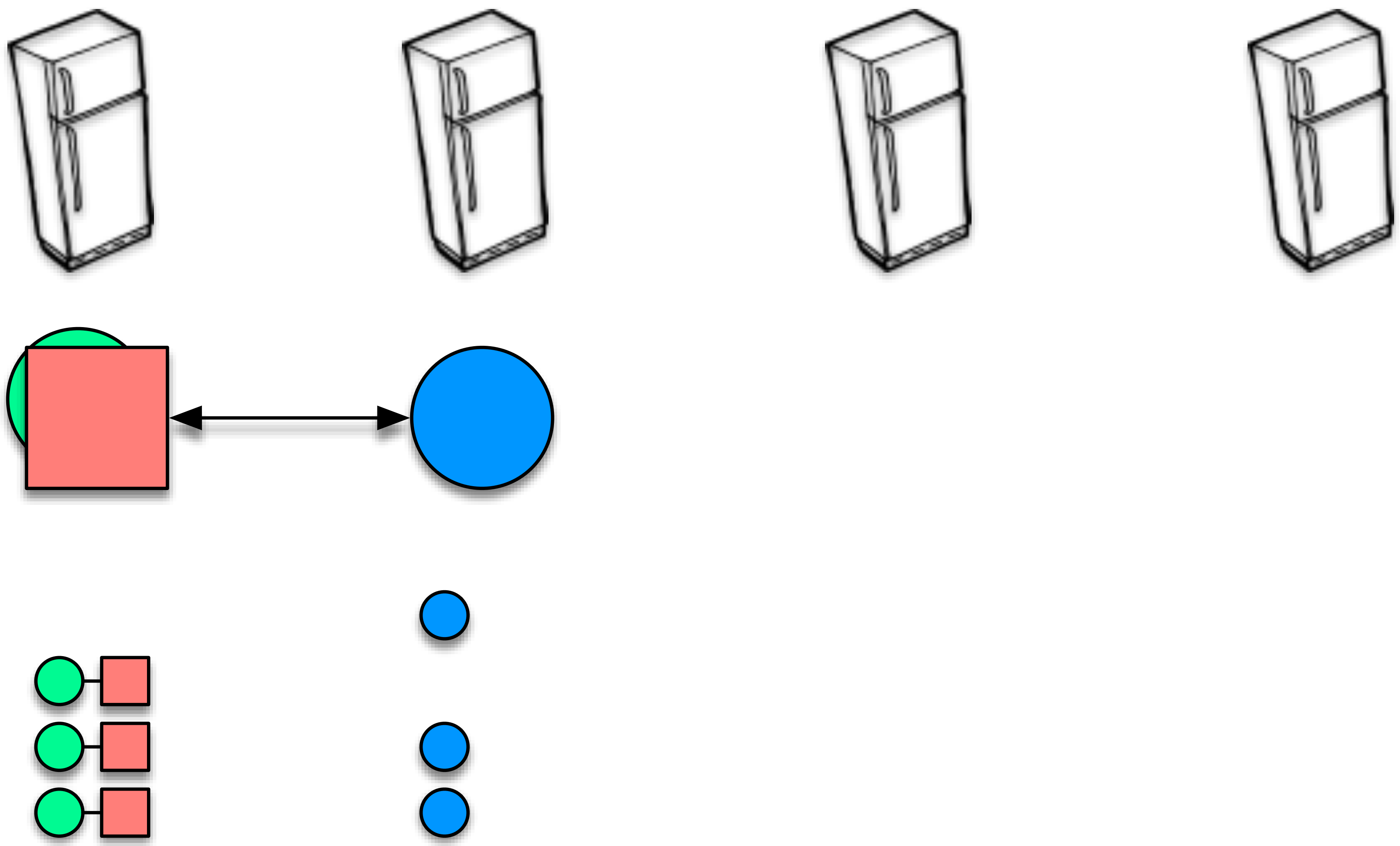
- Only fixed number of clients
  Clients resort to full state synchronization when they've been partitioned too long

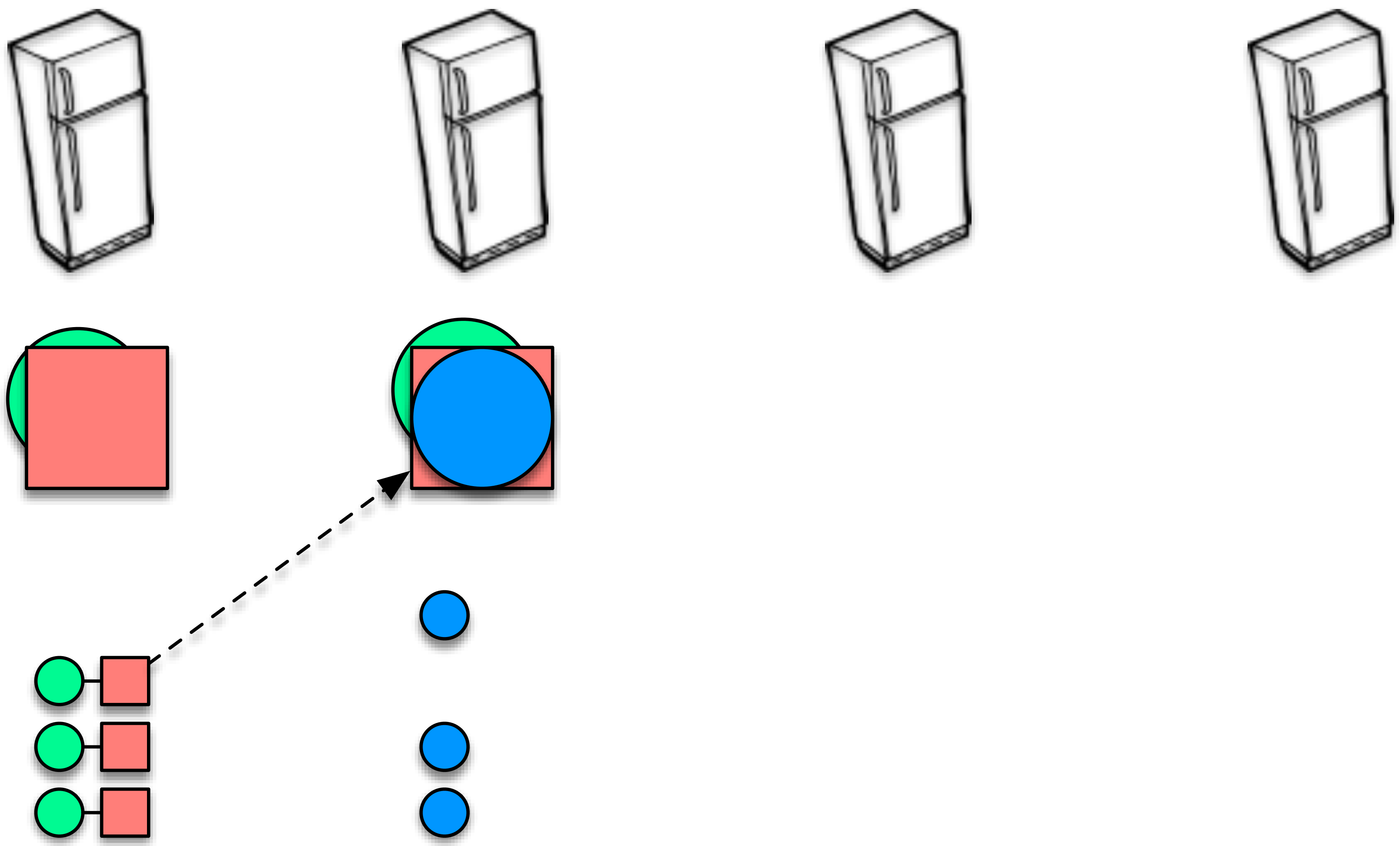





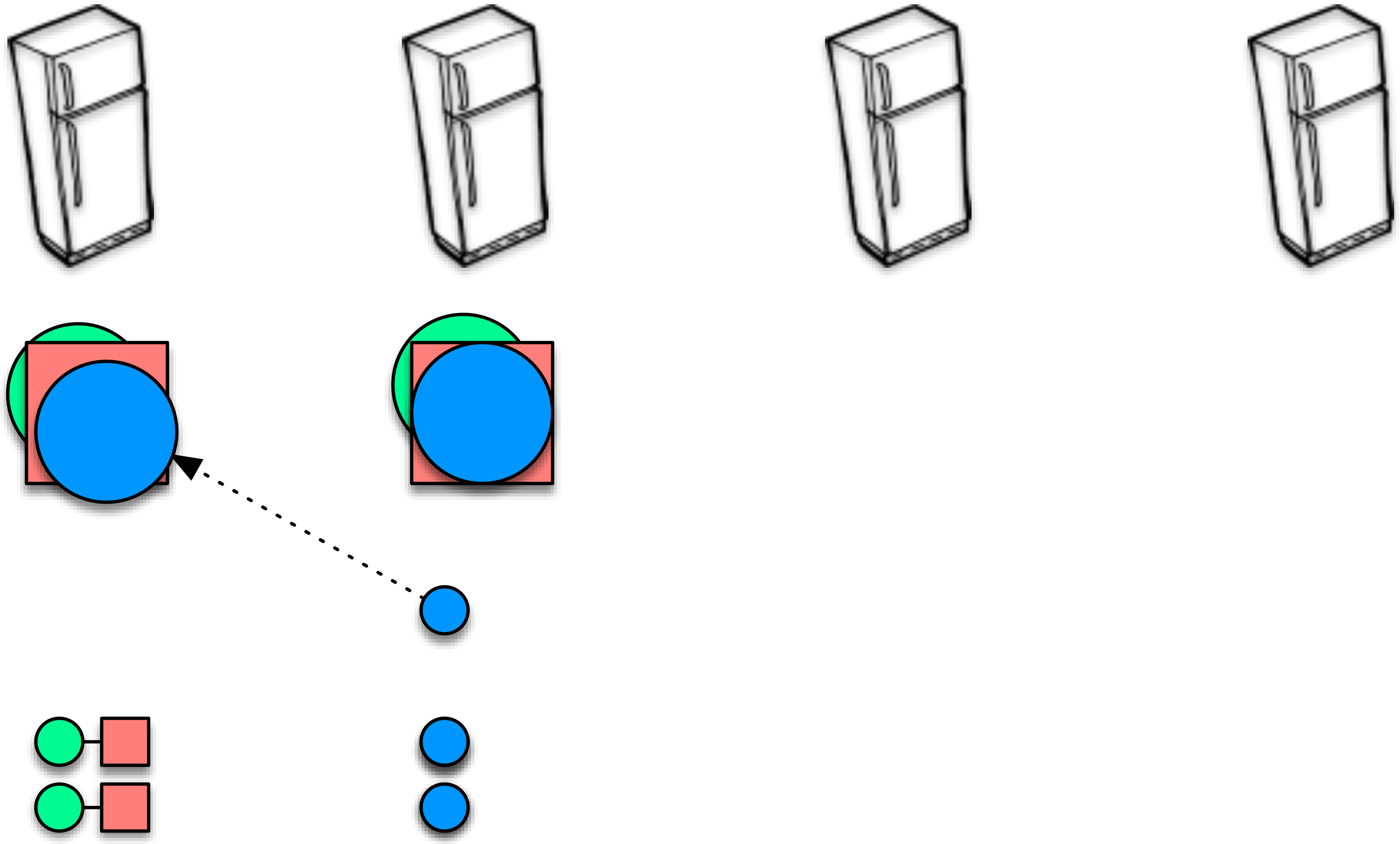


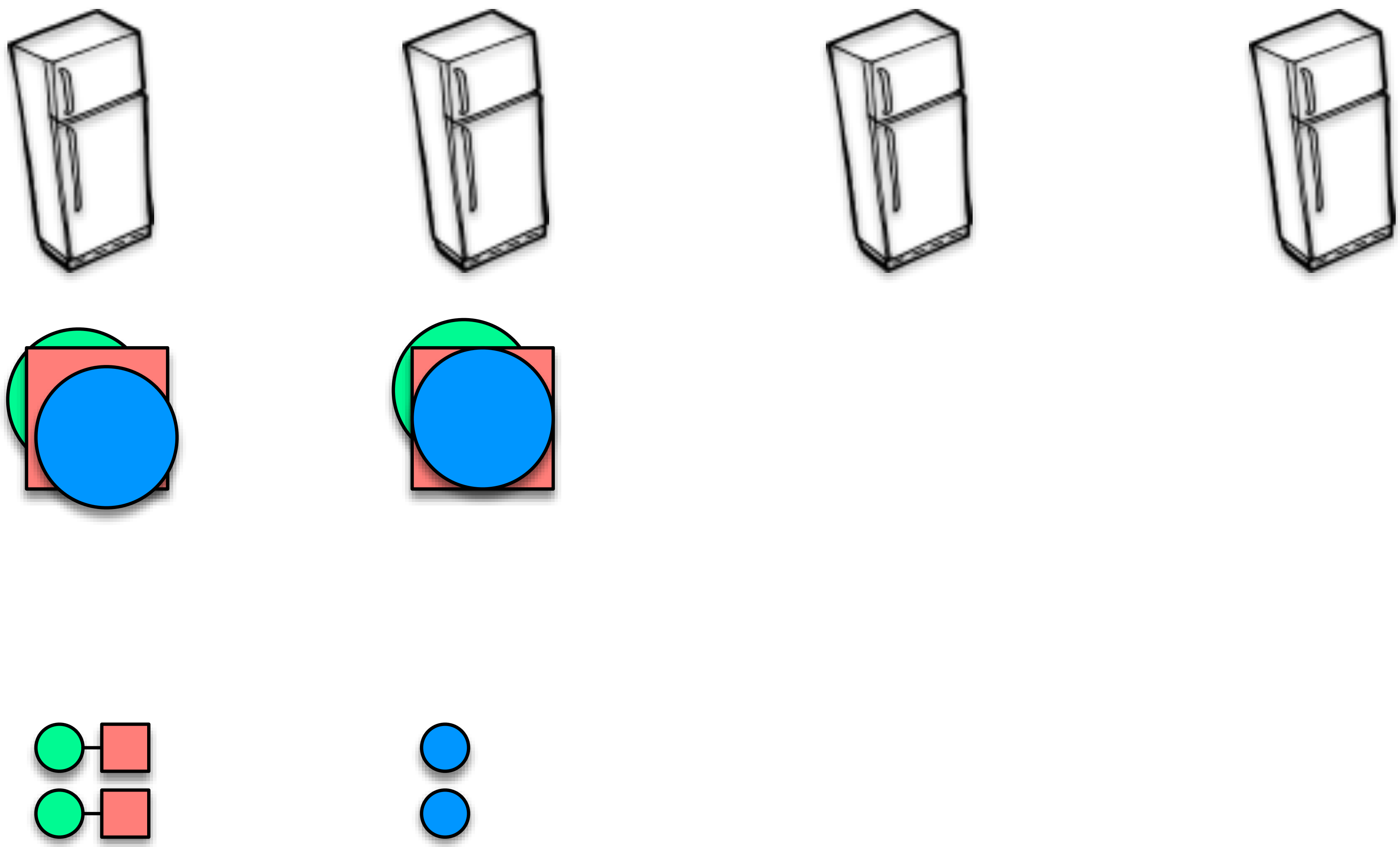












Selective Hearing

- Epidemic broadcast protocol
  Runtime system for application state & scope

Selective Hearing

- Epidemic broadcast protocol
  Runtime system for application state & scope

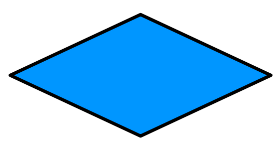
- Peer-to-peer dissemination
  Pairwise synchronization between peers
  without a central coordinator

Selective Hearing

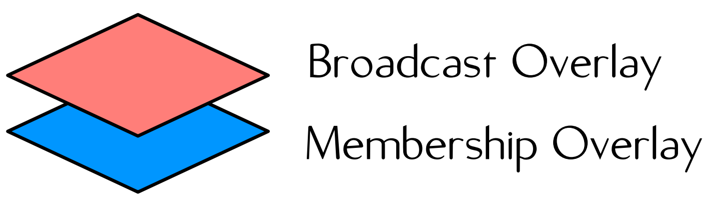
- Epidemic broadcast protocol
  Runtime system for application state & scope

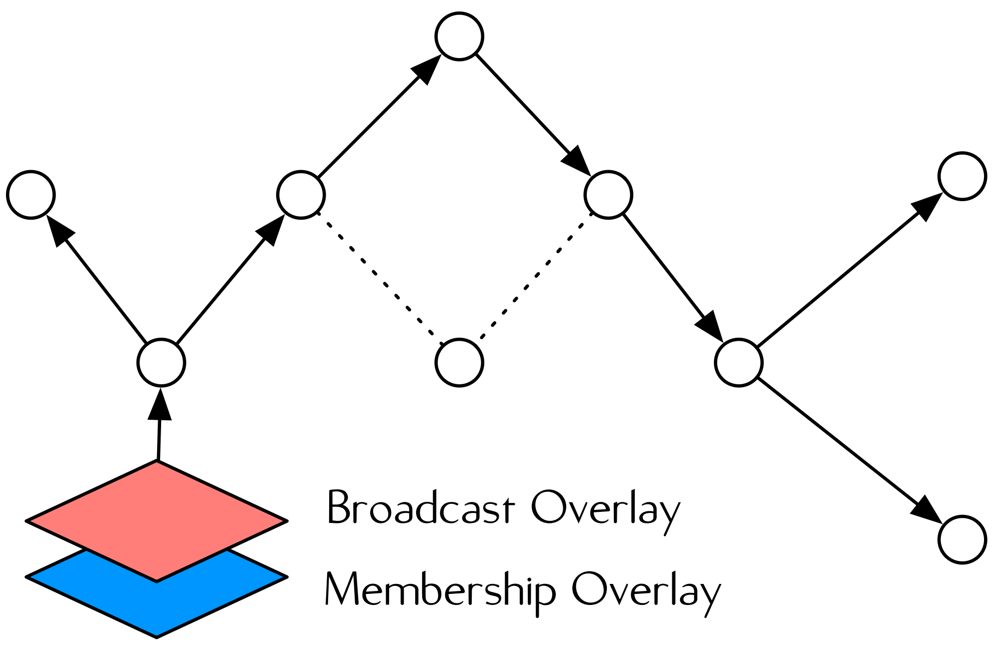
- Peer-to-peer dissemination
  Pairwise synchronization between peers without a central coordinator

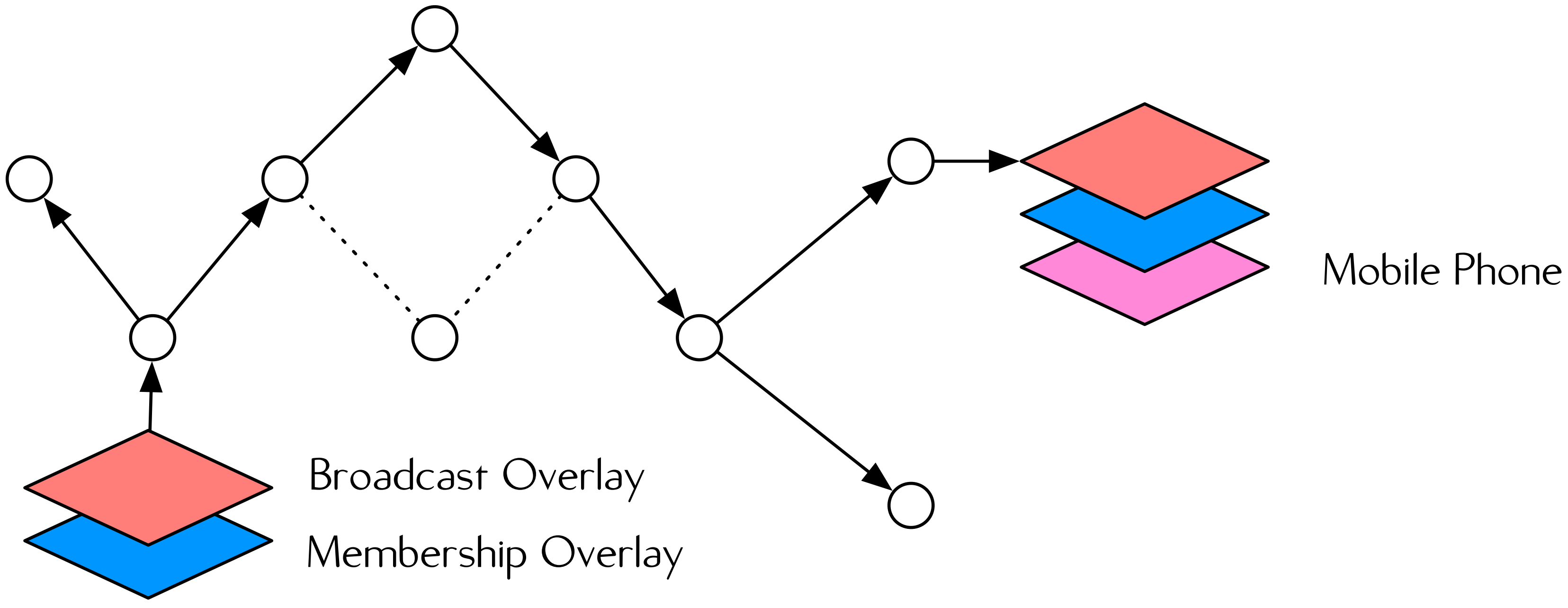
- No ordering guarantees on messages
  Programming model can tolerate message reordering and duplication

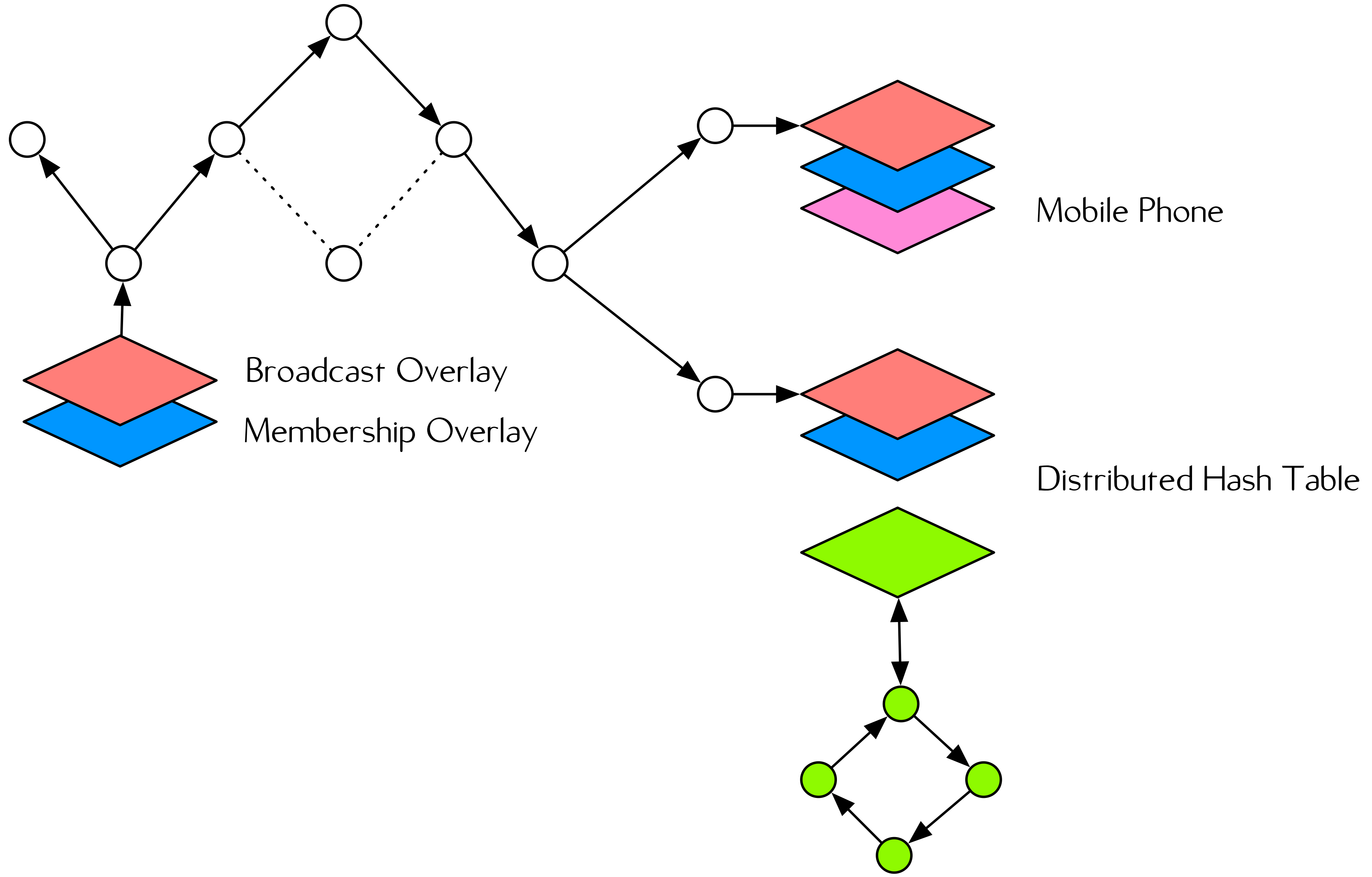


Membership Overlay

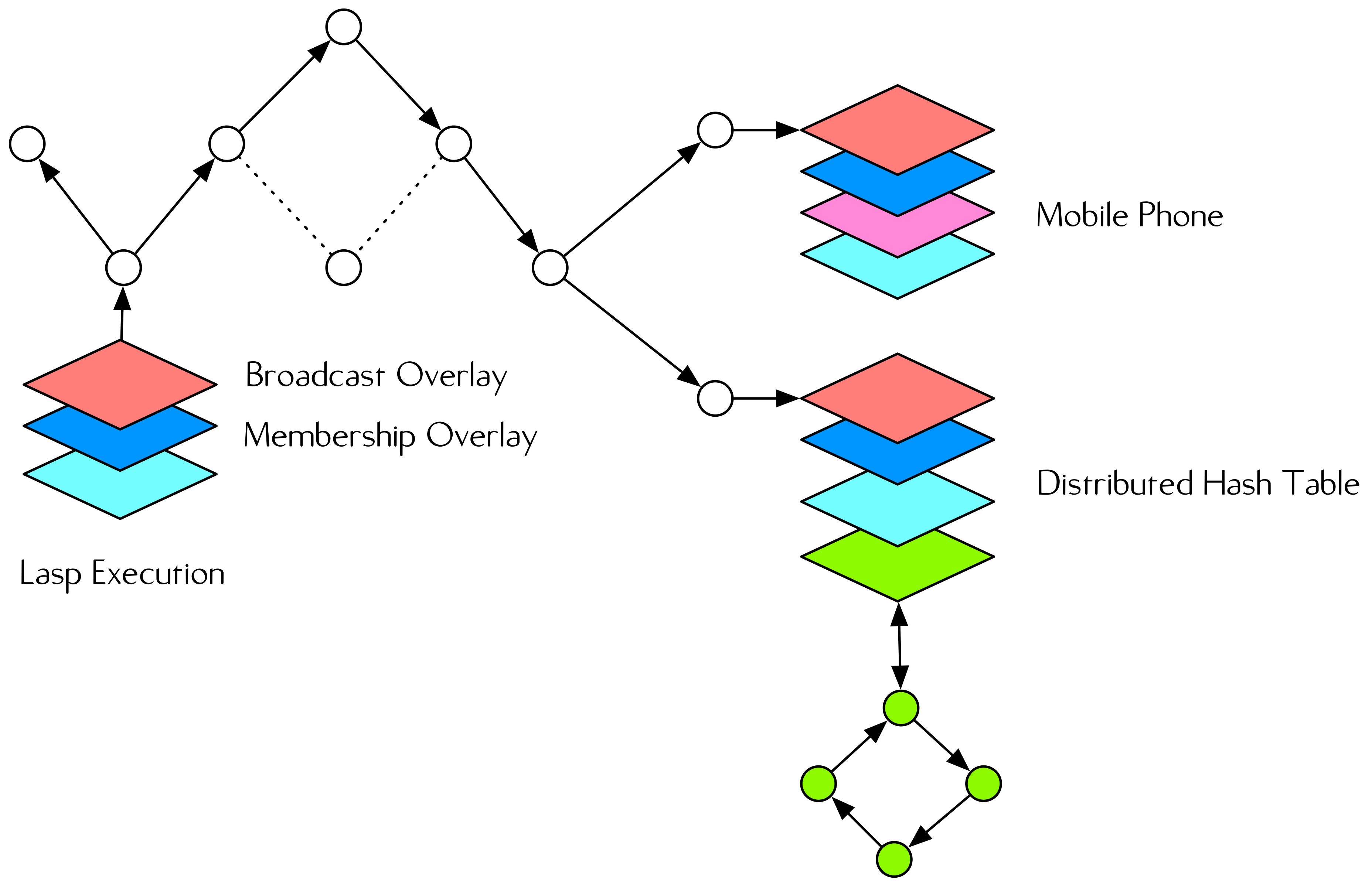

Broadcast Overlay
Membership Overlay


Broadcast Overlay
Membership Overlay


Broadcast Overlay
Membership Overlay
Mobile Phone



Broadcast Overlay
Membership Overlay
Mobile Phone
Distributed Hash Table



Broadcast Overlay
Membership Overlay
Lasp Execution
Mobile Phone
Distributed Hash Table

What can we build?
Advertisement Counter

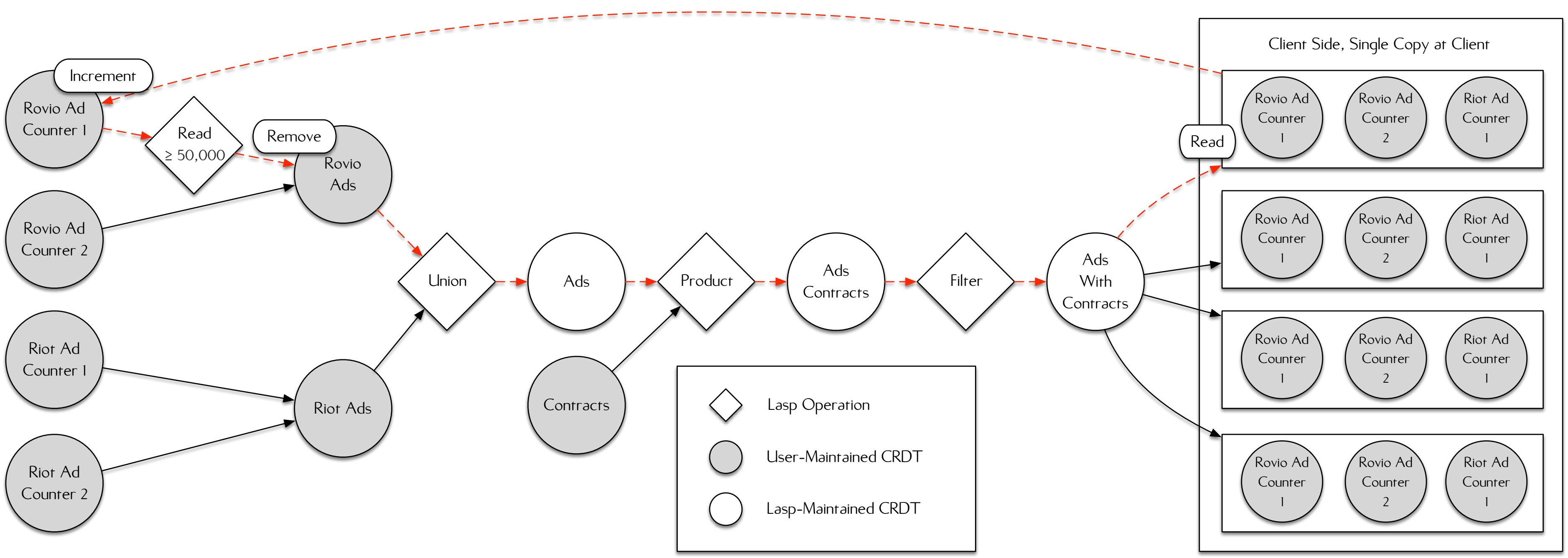
Advertisement Counter

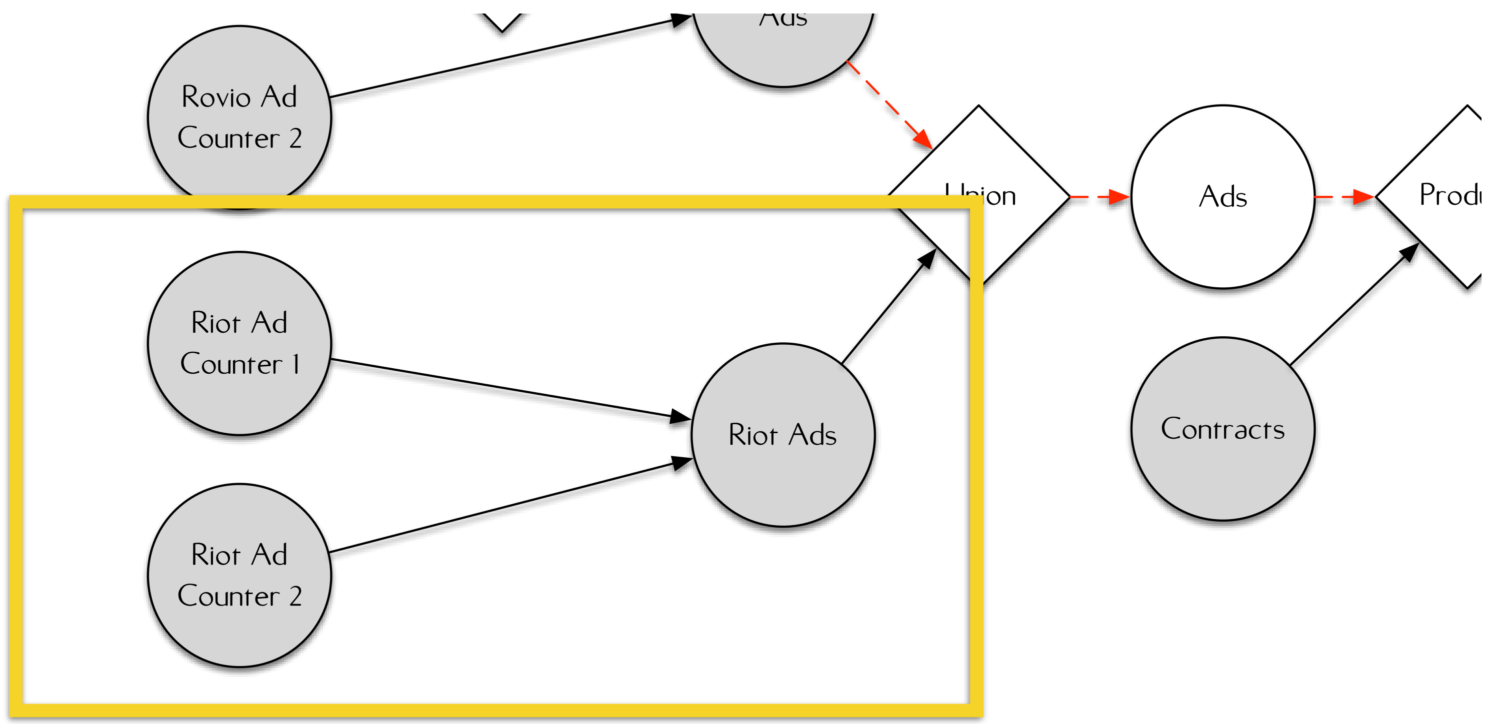
- Mobile game platform selling advertisement space
  Advertisements are paid according to a minimum number of impressions

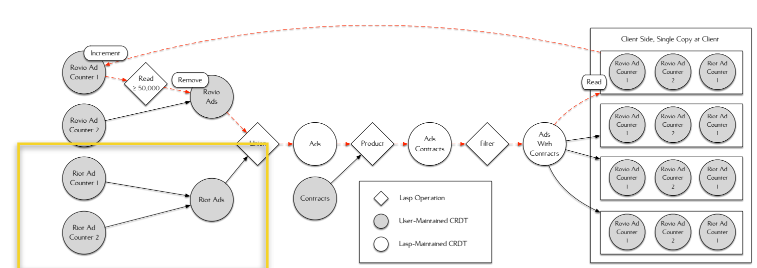
Advertisement Counter

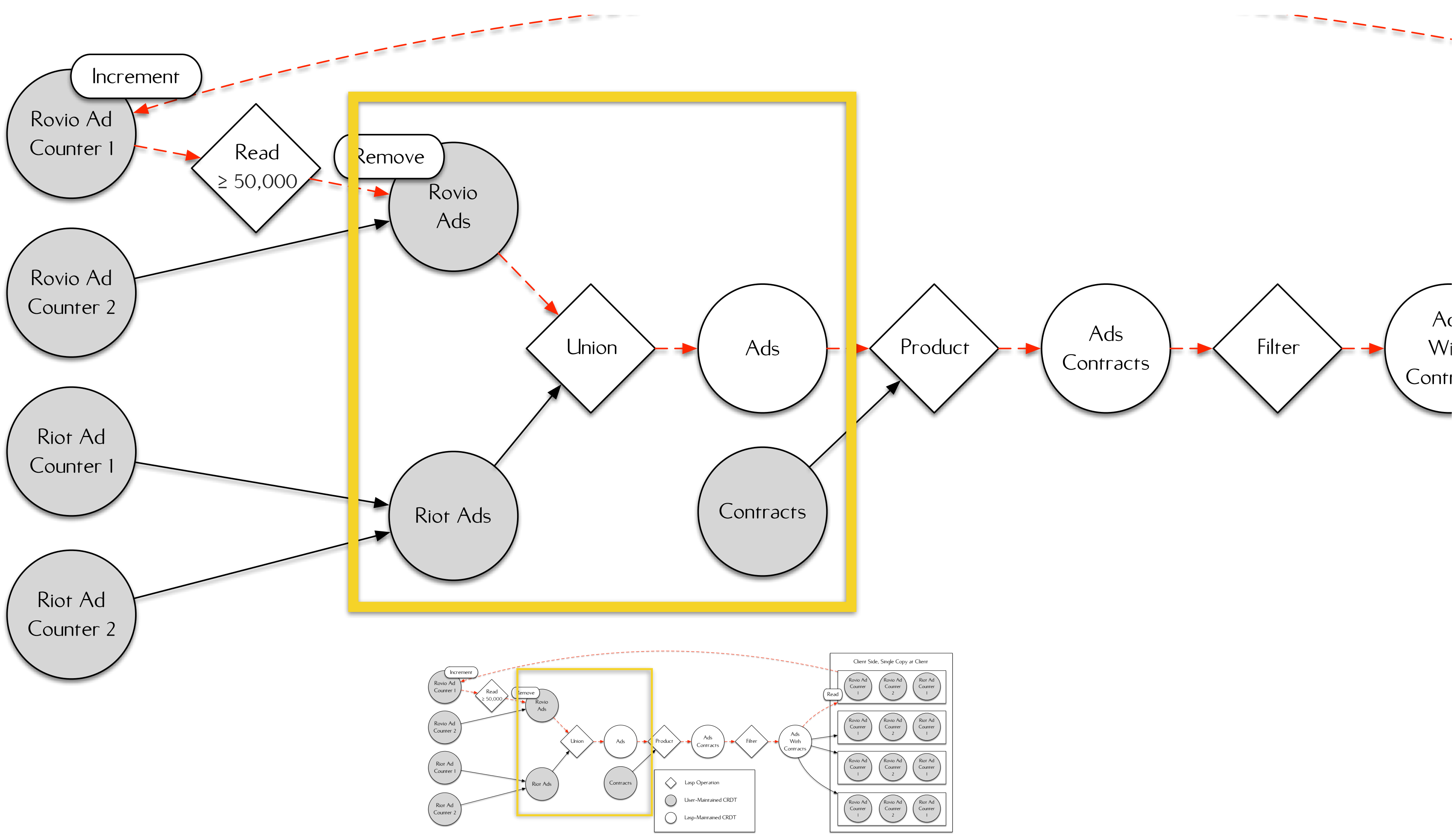
- **Mobile game platform selling advertisement space**
  Advertisements are paid according to a minimum number of impressions

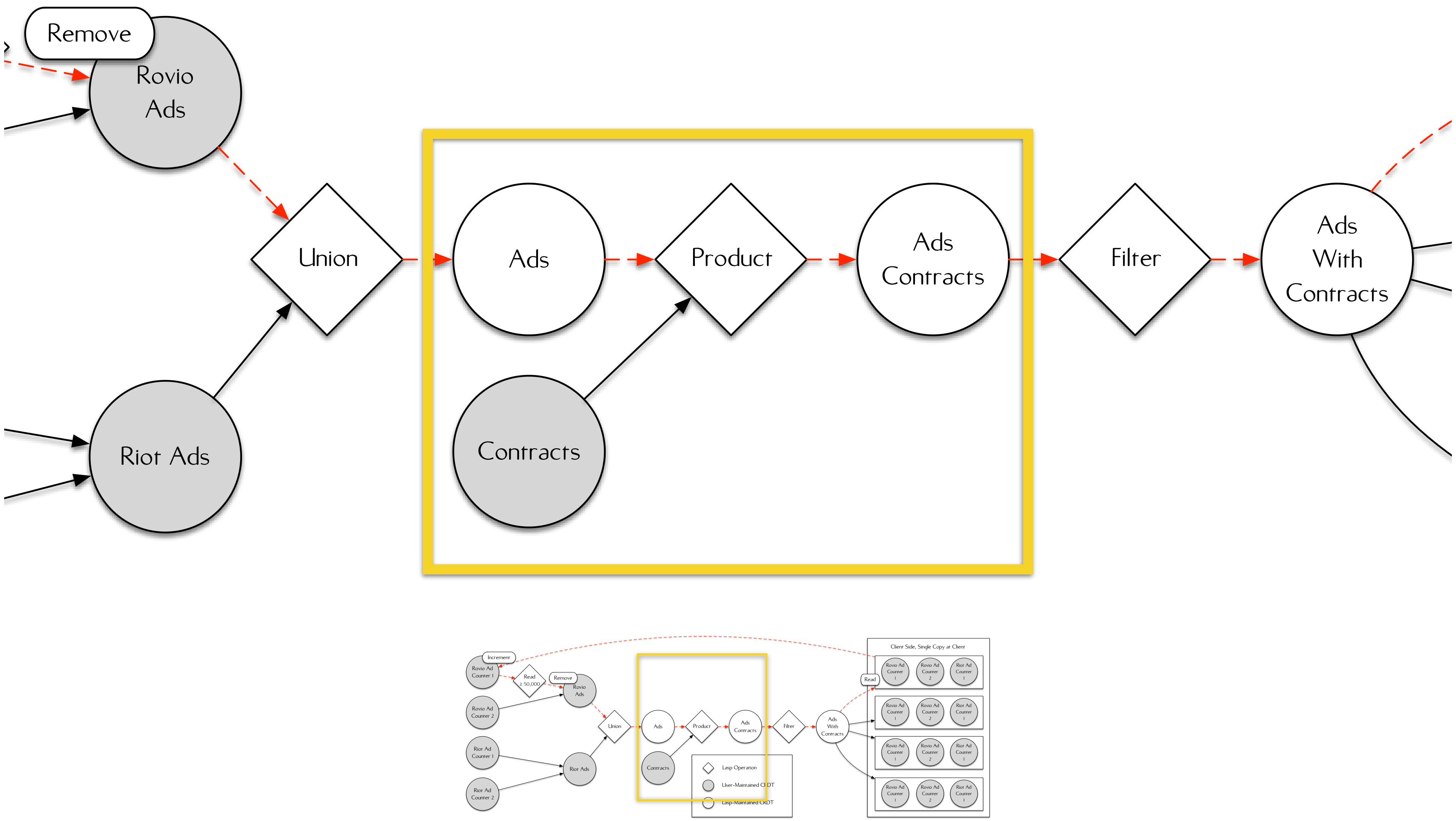
- **Clients will go offline**
  Clients have limited connectivity and the system still needs to make progress while clients are offline

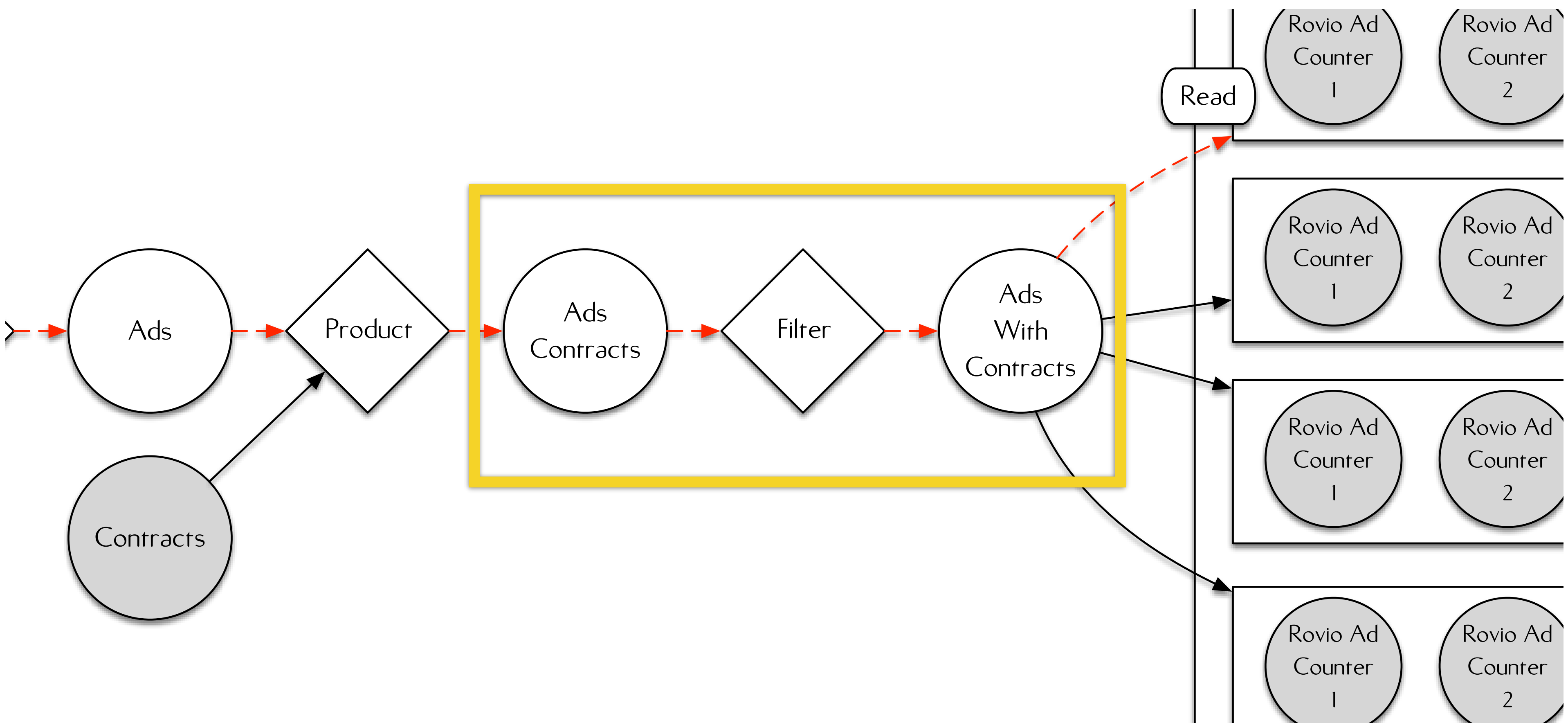

Increment
Rovio Ad Counter 1
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter 2
Riot Ad Counter 1
Riot Ads
Riot Ad Counter 2
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Client Side, Single Copy at Client
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Lasp Operation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

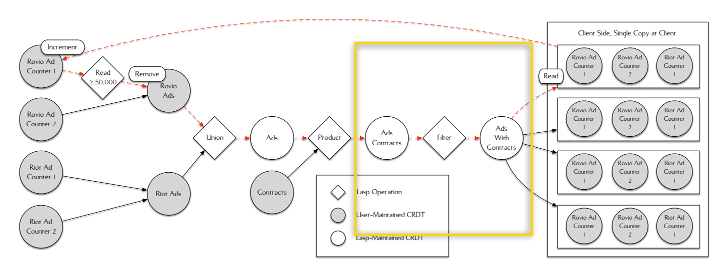

Rovio Ad Counter 2
Riot Ad Counter 1
Riot Ad Counter 2
Riot Ads
Ads
Contracts

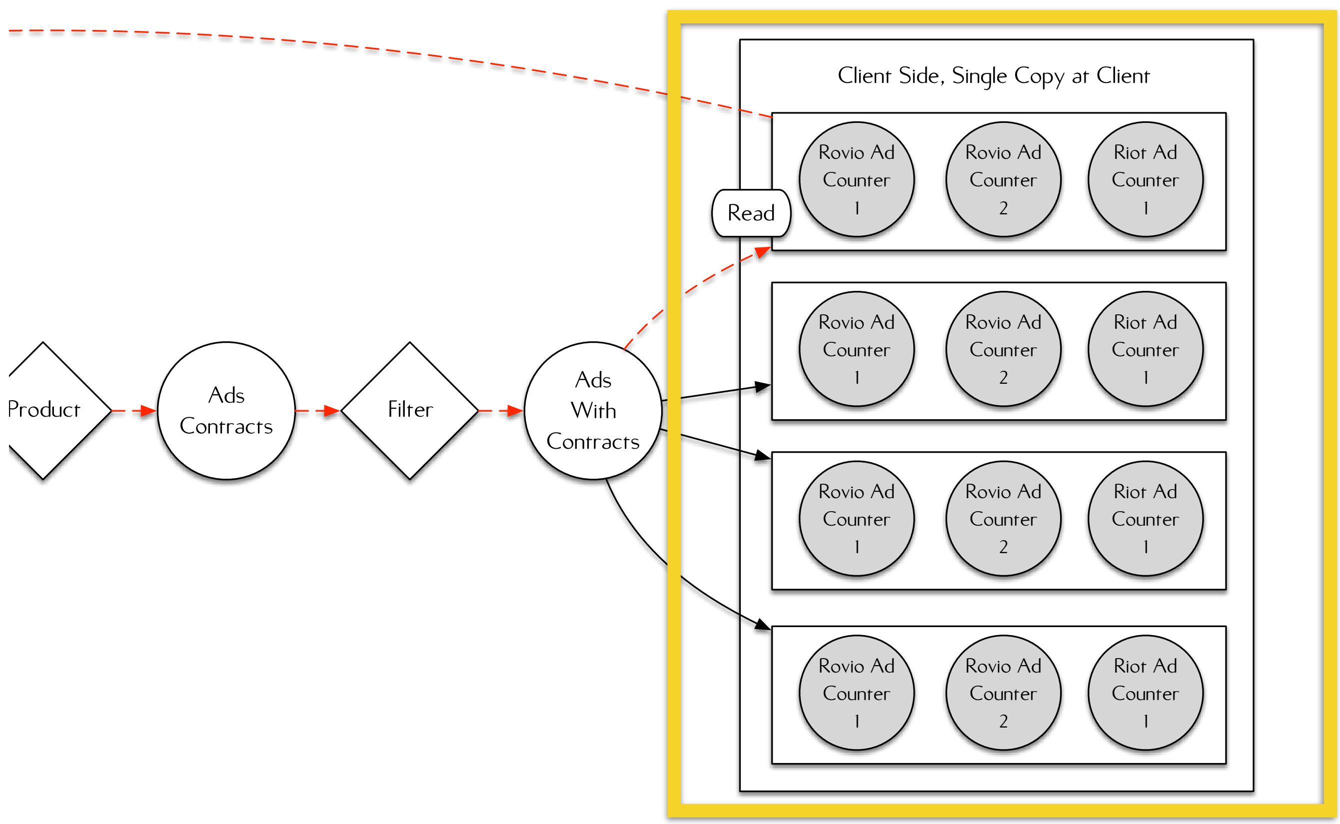


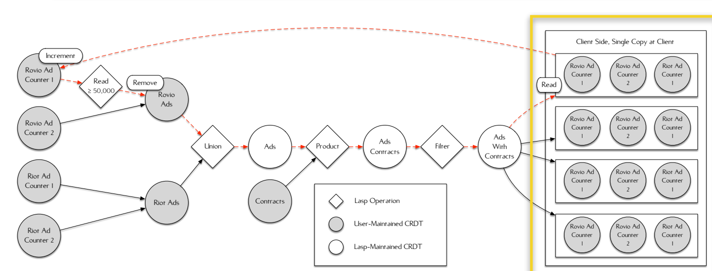

Increment
Rovio Ad Counter 1
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter 2
Union
Ads
Product
Ads Contracts
Filter
Riot Ad Counter 1
Riot Ads
Contracts
Riot Ad Counter 2

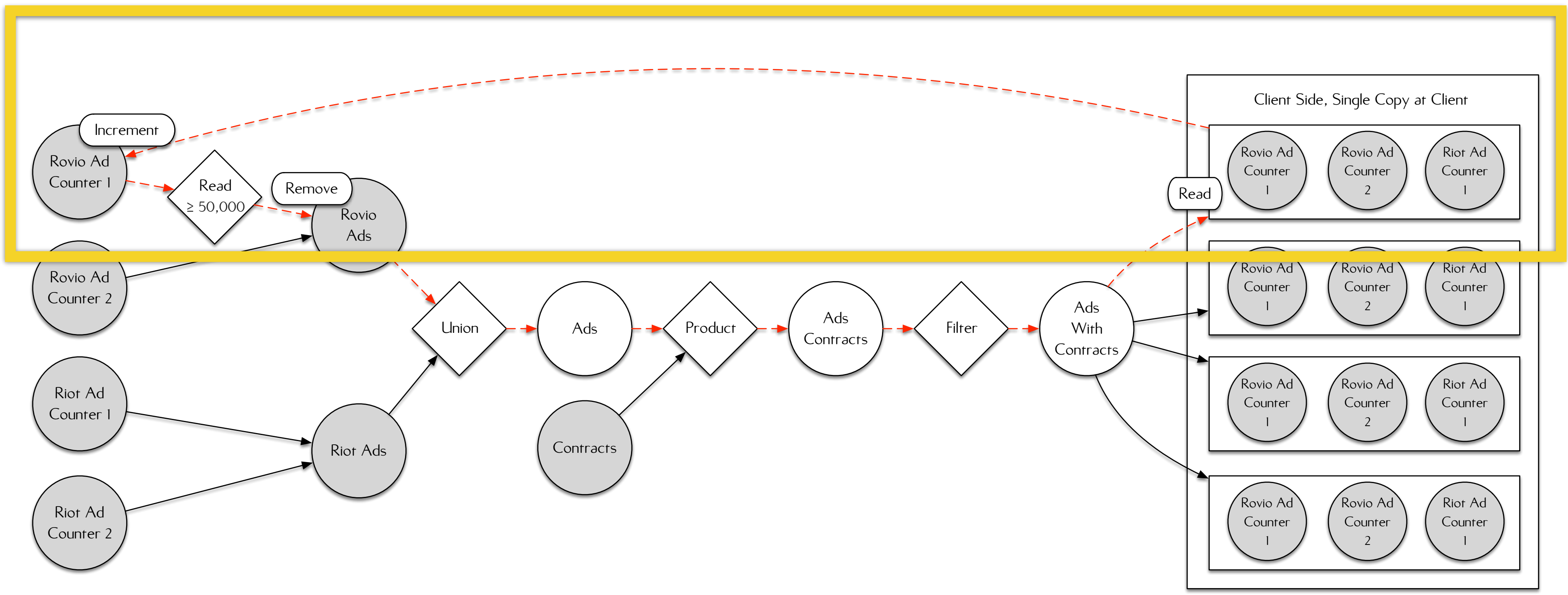

Remove
Rovio Ads
Union
Riot Ads
Ads
Contracts
Product
Ads Contracts
Filter
Ads With Contracts

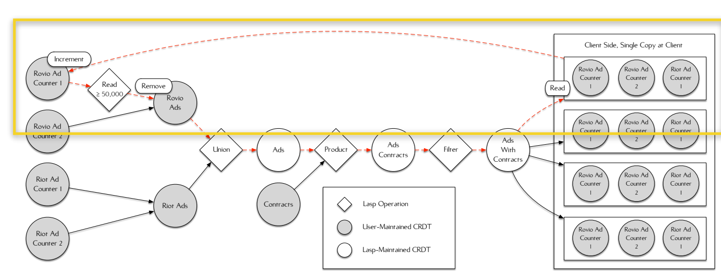

Ads
Product
Ads
Contracts
Filter
Ads
With
Contracts
Contracts
Read
Rovio Ad
Counter
1
Rovio Ad
Counter
2
Rovio Ad
Counter
1
Rovio Ad
Counter
2
Rovio Ad
Counter
1
Rovio Ad
Counter
2
Rovio Ad
Counter
1
Rovio Ad
Counter
2

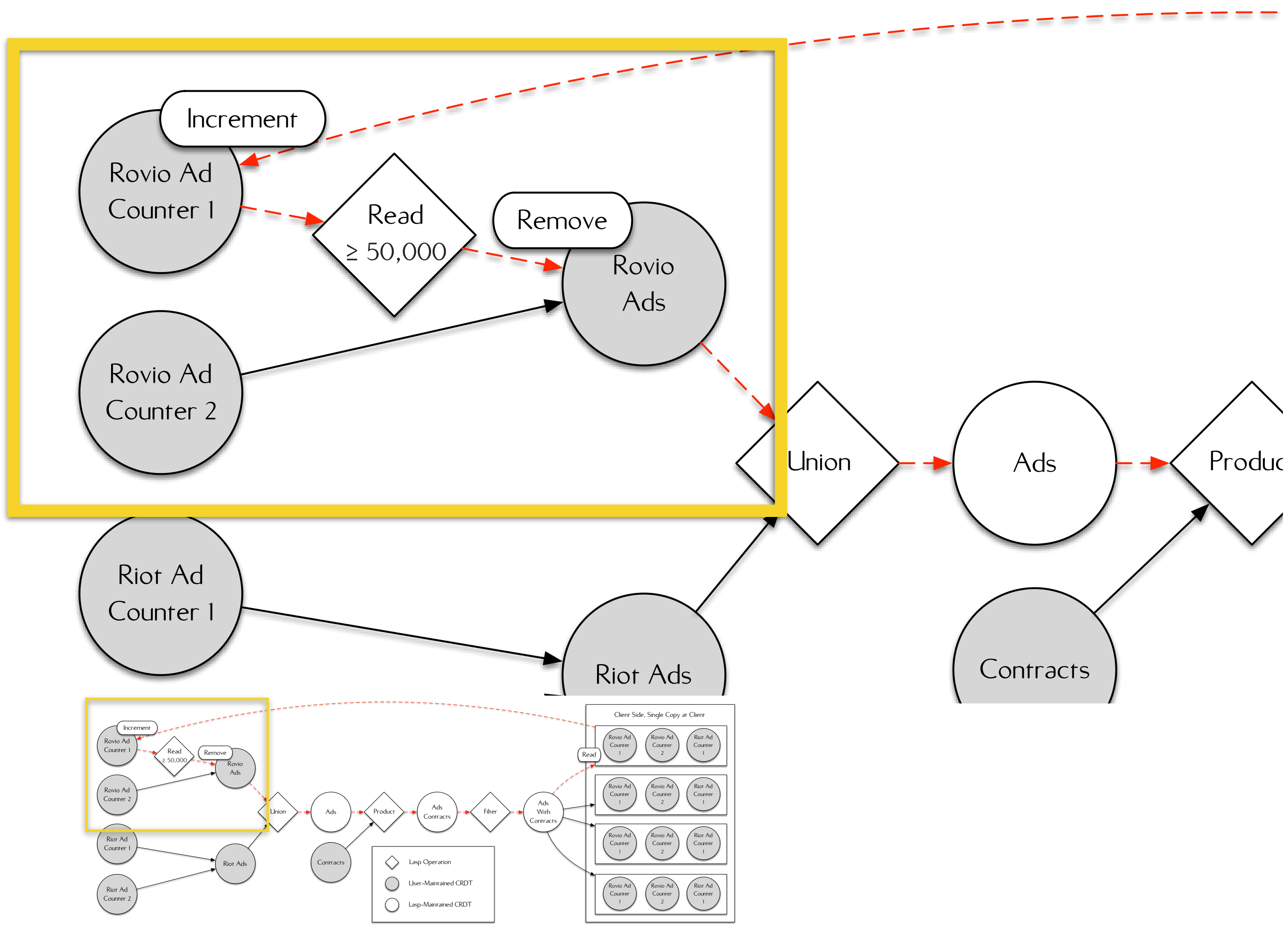


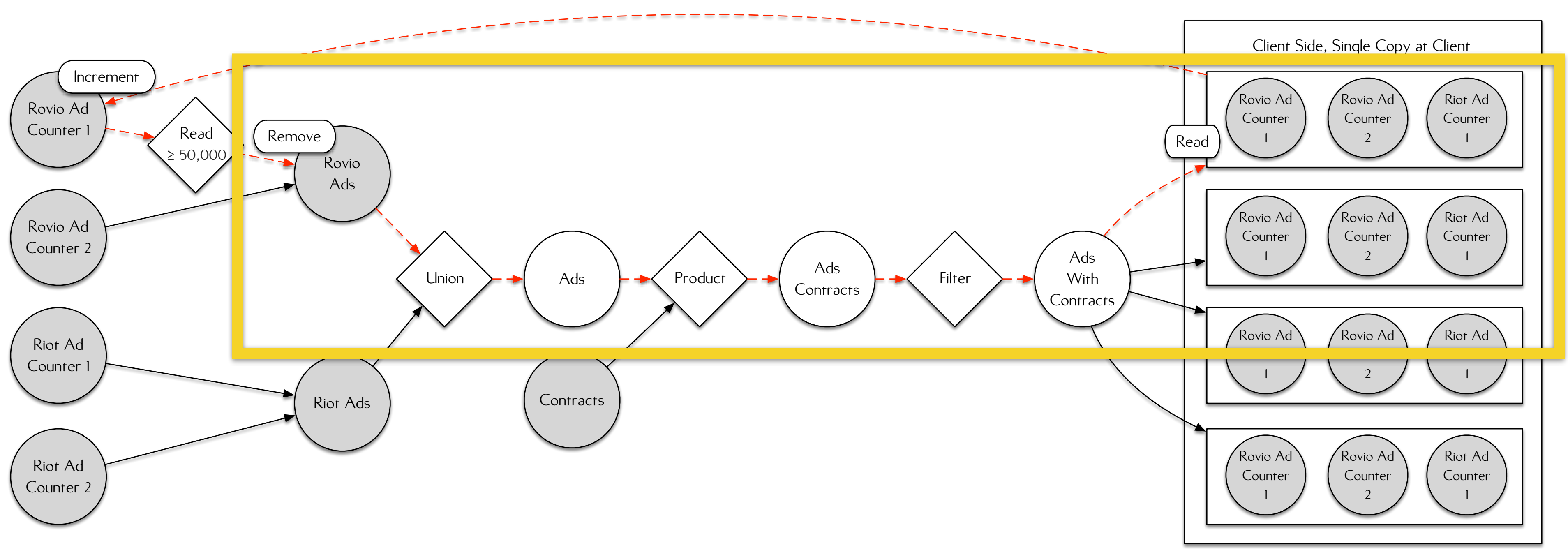

Client Side, Single Copy at Client
Read
Product
Ads Contracts
Filter
Ads With Contracts
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad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

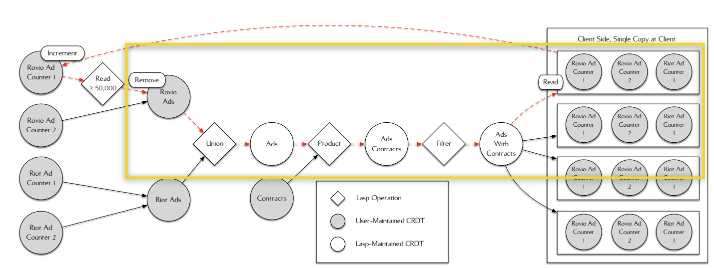



Client Side, Single Copy at Client
Increment
Rovio Ad Counter 1
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter 2
Riot Ad Counter 1
Riot Ad Counter 2
Riot Ads
Union
Ads
Contracts
Product
Ads Contracts
Filter
Ads With Contracts
Read
Rovio Ad Counter 1
Rovio Ad Counter 2
Riot Ad Counter 1




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
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



Advertisement Counter

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

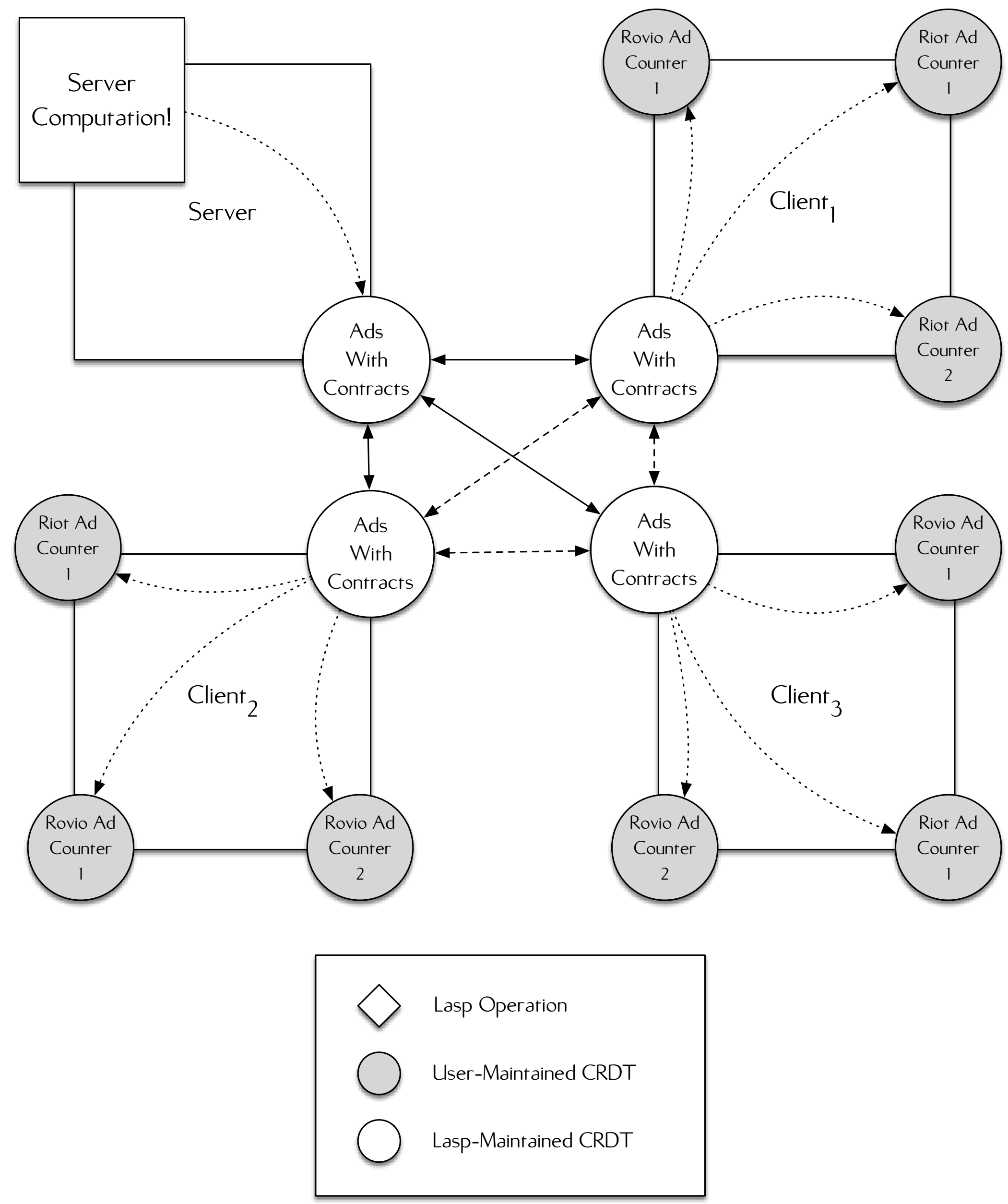
Advertisement Counter

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

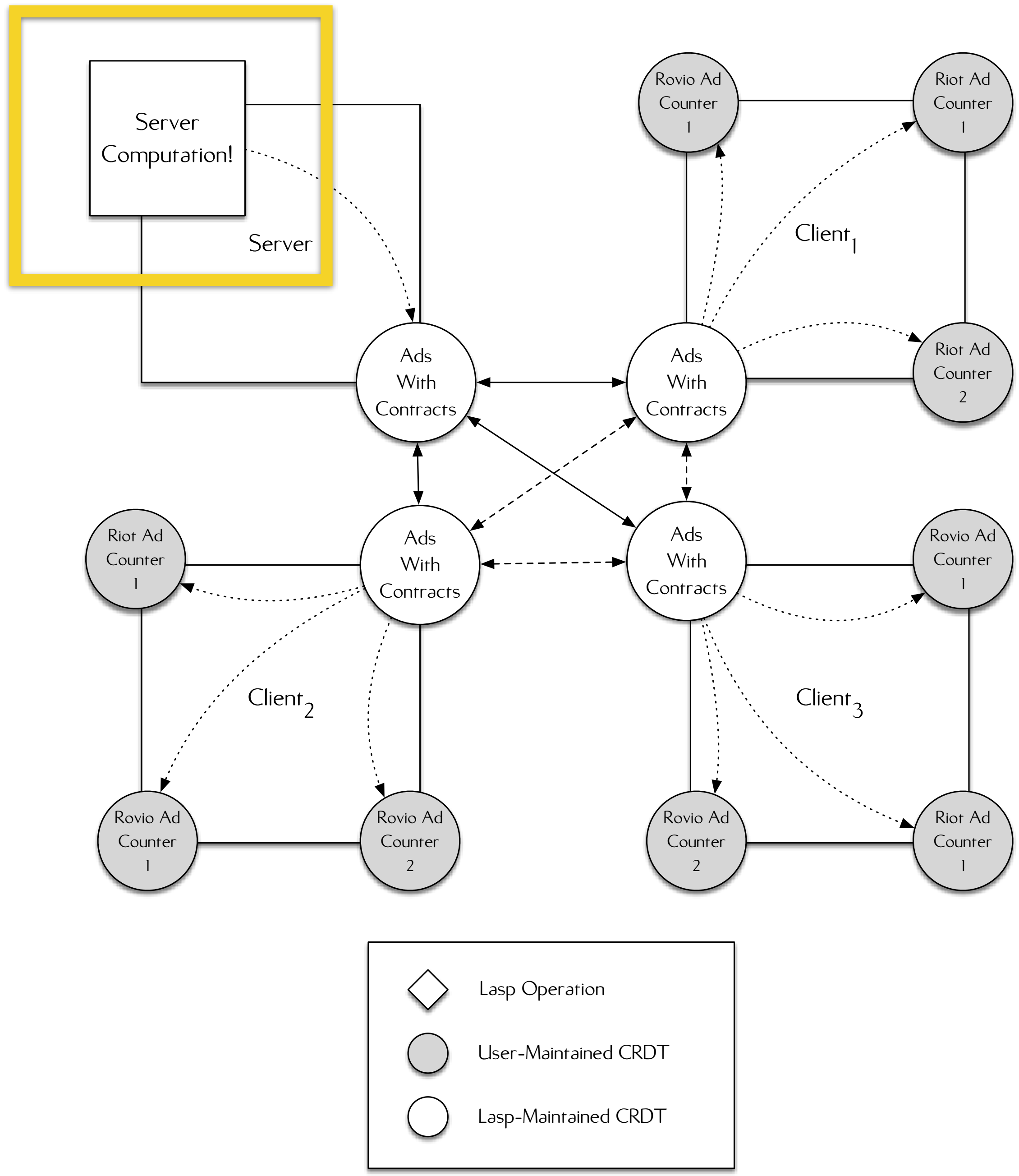
- Arbitrary distribution
  Use of convergent data structures allows computational graph to be arbitrarily distributed

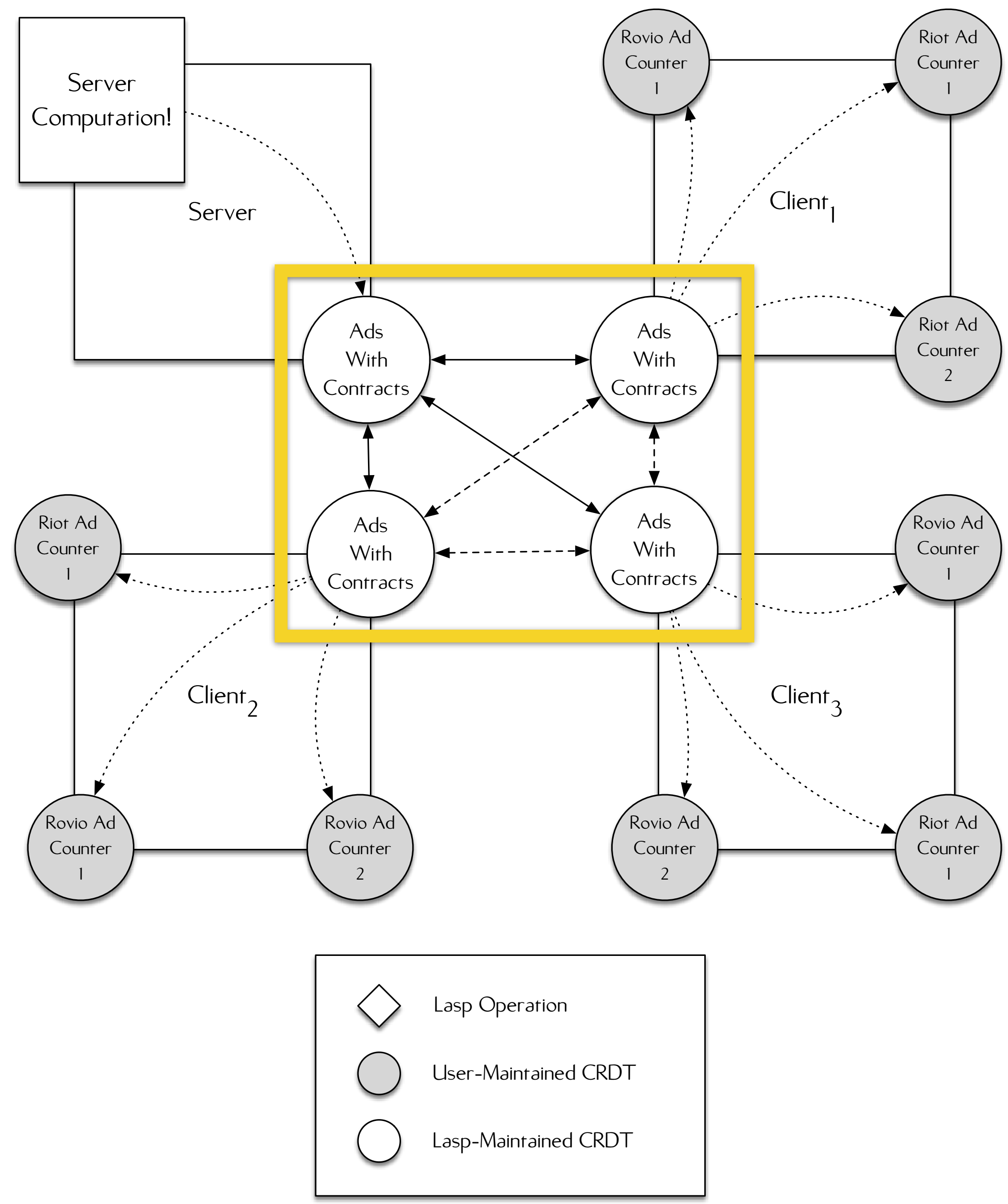
Advertisement Counter

- 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
Client$_1$
Ads With Contracts
Riot Ad Counter 2
Riot Ad Counter 1
Ads With Contracts
Ads With Contracts
Rovio Ad Counter 1
Client$_2$
Client$_3$
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
Rovio Ad Counter 1
Riot Ad Counter 1
Client1
Ads With Contracts
Ads With Contracts
Riot Ad Counter 2
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
Rovio Ad Counter 1
Riot Ad Counter 1
Client1
Riot Ad Counter 2
Ads With Contracts
Ads With Contracts
Ads With Contracts
Ads With Contracts
Riot Ad Counter 1
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

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 built up from zero synchronization

We’ve built up from zero synchronization

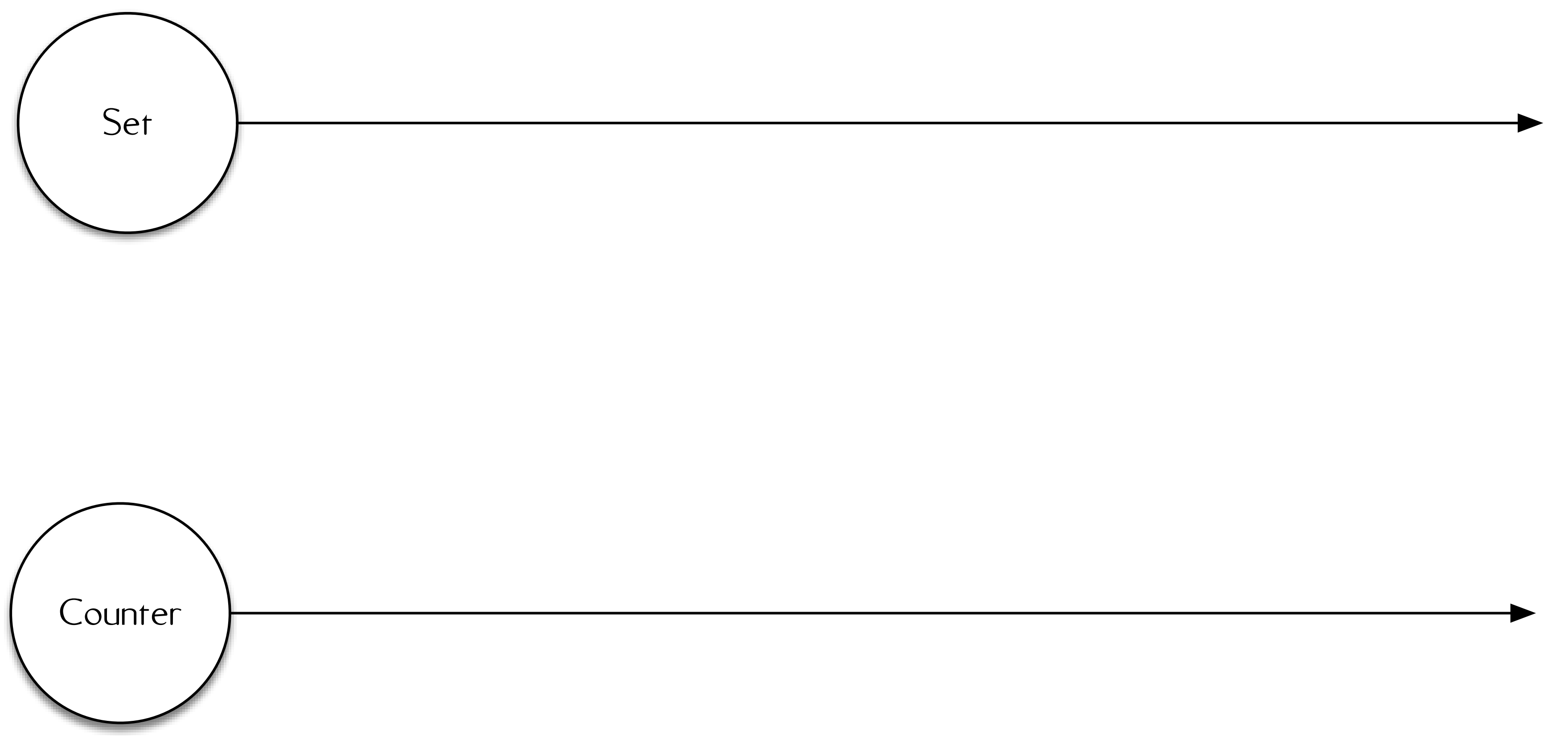
Instead of working to remove synchronization

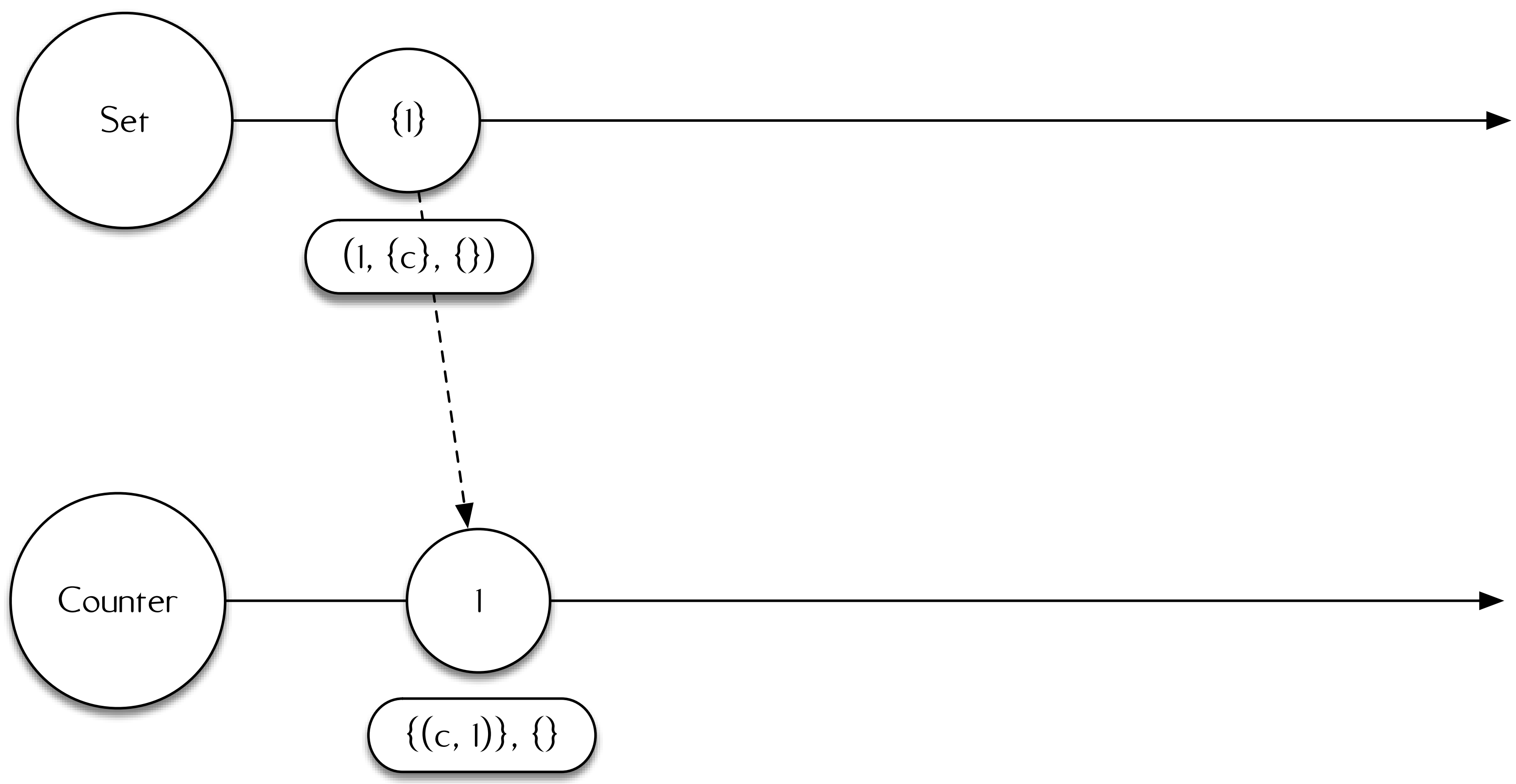
Challenges

Looking Ahead

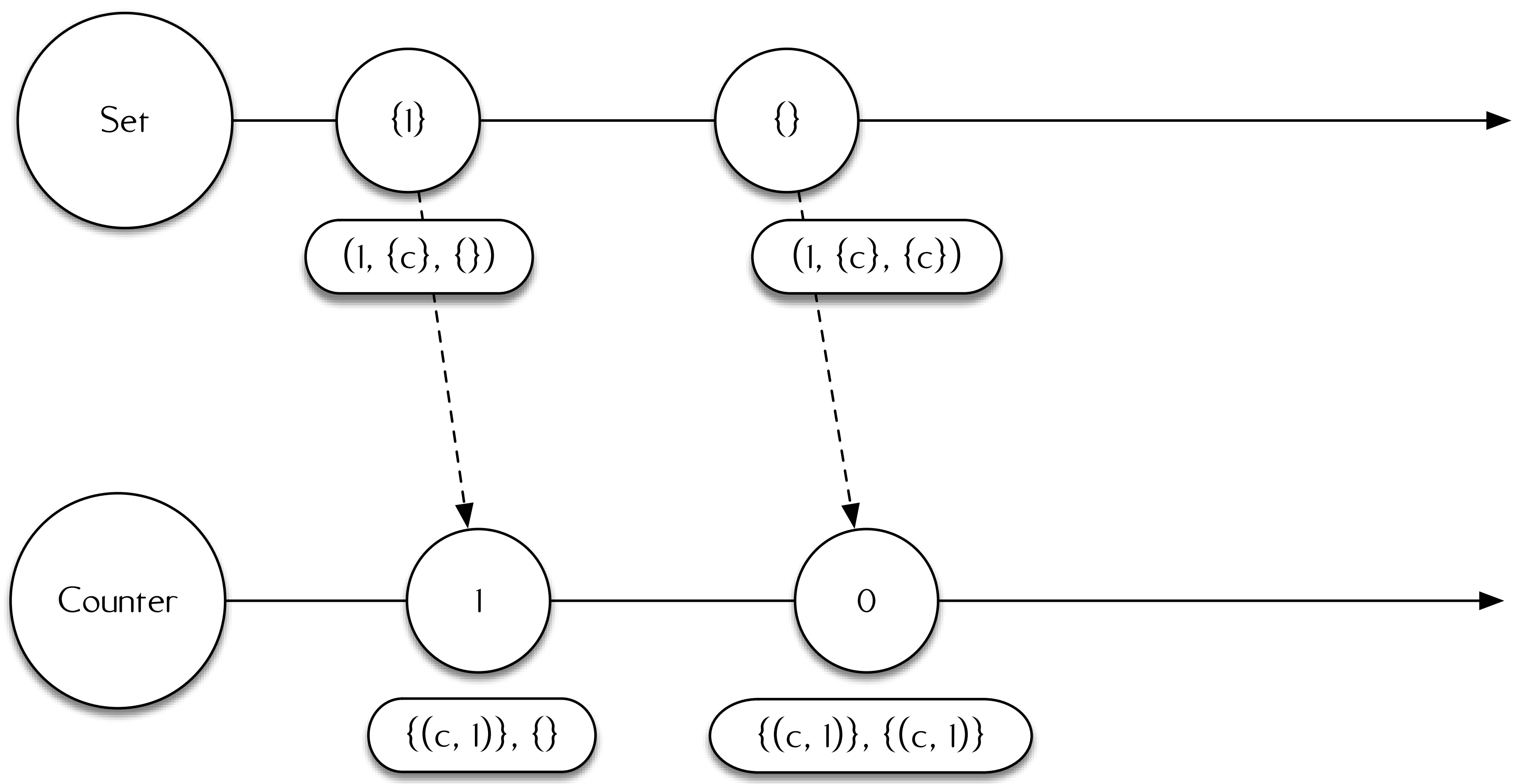
Causality

State Explosion

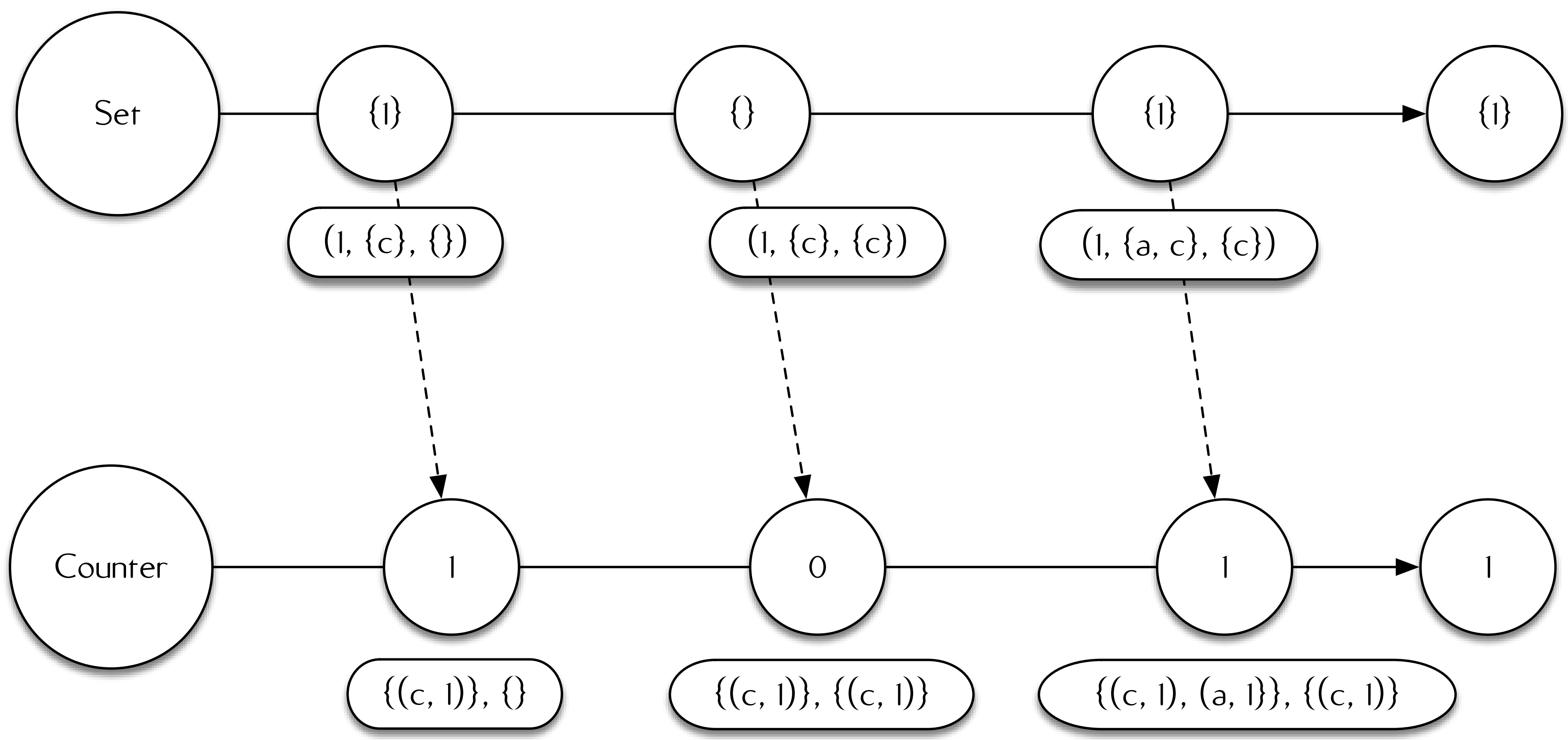

Set
Counter



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

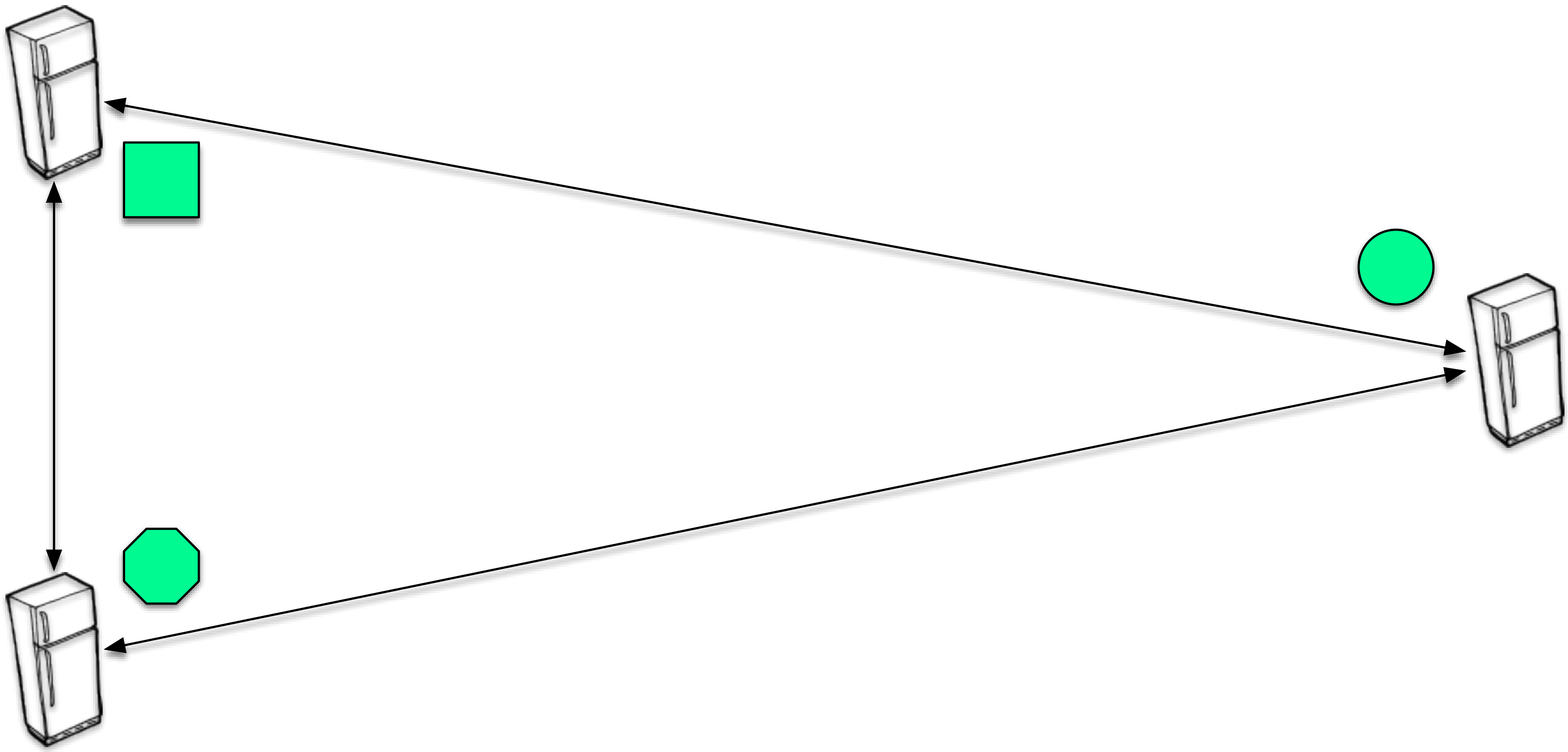


Set
{1}
{}
(1, {c}, {})
(1, {c}, {c})
Counter
1
0
{(c, 1)}, {}
{(c, 1)}, {(c, 1)}


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



Computations

Expressiveness

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

How do I learn more?

Publications

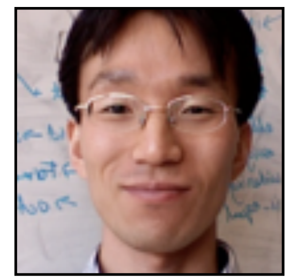
- “Lasp: A Language for Distributed, Coordination-Free Programming”
  ACM SIGPLAN PPDP 2015
- “Selective Hearing: An Approach to Distributed, Eventually Consistent Edge Computation”
  IEEE W-PSDS 2015
- “The Implementation and Use of a Generic Dataflow Behaviour in Erlang”
  ACM SIGPLAN Erlang Workshop ’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

Three independently **successful** techniques.

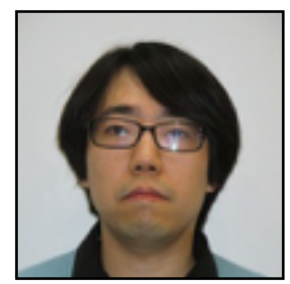
Can we combine them into a **cohesive** programming environment for distributed programming?

Thanks!










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