LASP

DISTRIBUTED EVENTUALLY CONSISTENT COMPUTATIONS

EN TAL AV

CHRISTOPHER MEIKLEJOHN

RESEARCH WITH:

PETER VAN ROY (UCL)

MOTIVATION

SYNCHRONIZATION IS EXPENSIVE

SYNCHRONIZATION IS SOMETIMES IMPRACTICAL

MOBILE GAMES:

SHARED STATE BETWEEN CLIENTS

CLIENTS GO OFFLINE

http://www.rovio.com/en/news/blog/261/263-million-monthly-active-users-in-december/

INTERNET OF THINGS:

DISJOINT STATE AGGREGATED UPSTREAM CLIENTS GO OFFLINE

Gubbi, Jayavardhana, et al. "Internet of Things (IoT): A vision, architectural elements, and future directions." Future Generation Computer Systems 29.7 (2013): 1645-1660.

NO TOTAL ORDER:

REPLICATED SHARED STATE WITH OFFLINE CLIENTS

CLIENTS NEED TO MAKE PROGRESS

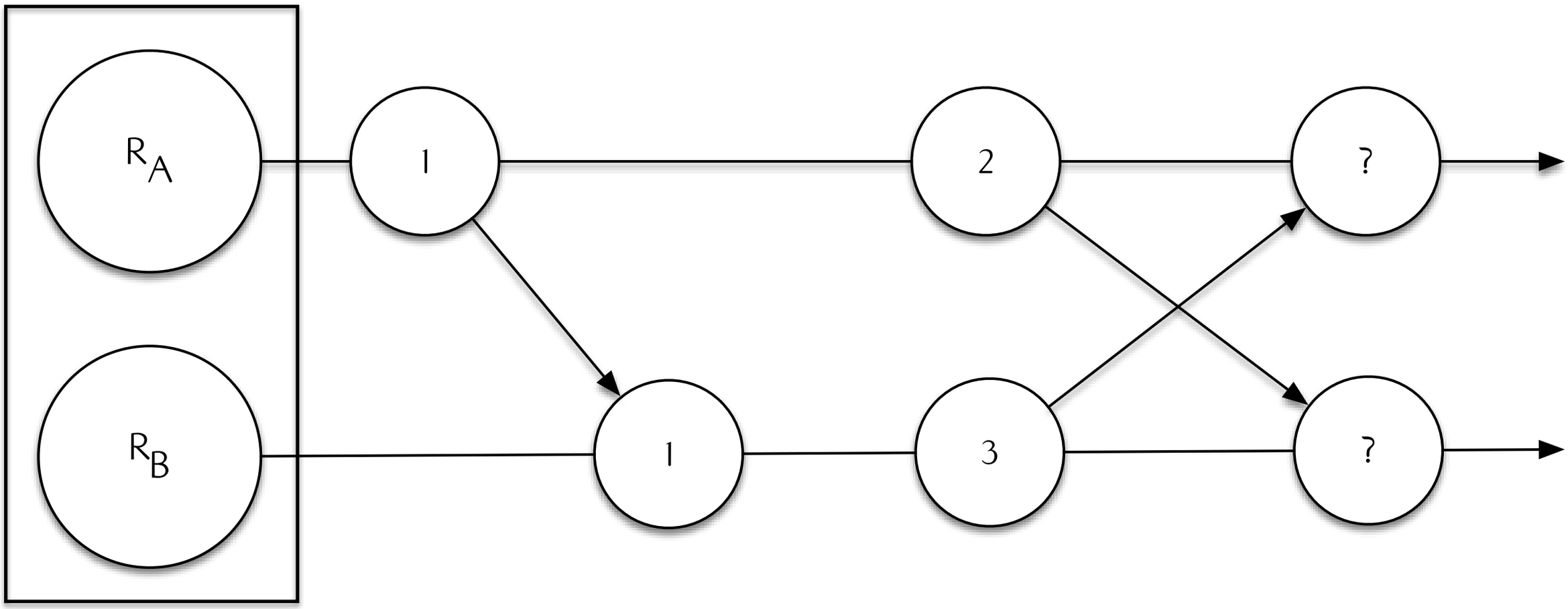
Gilbert, Seth, and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." ACM SIGACT News 33.2 (2002): 51-59.

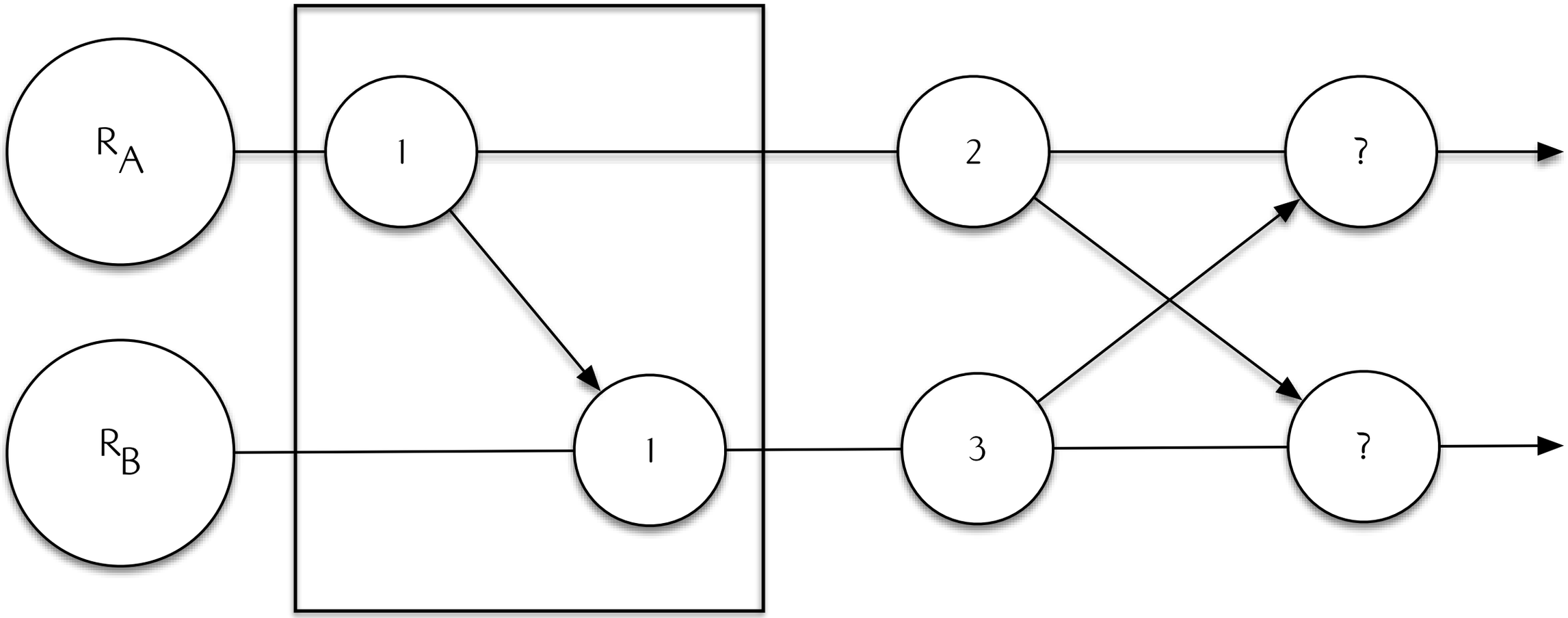
WALL CLOCKS:

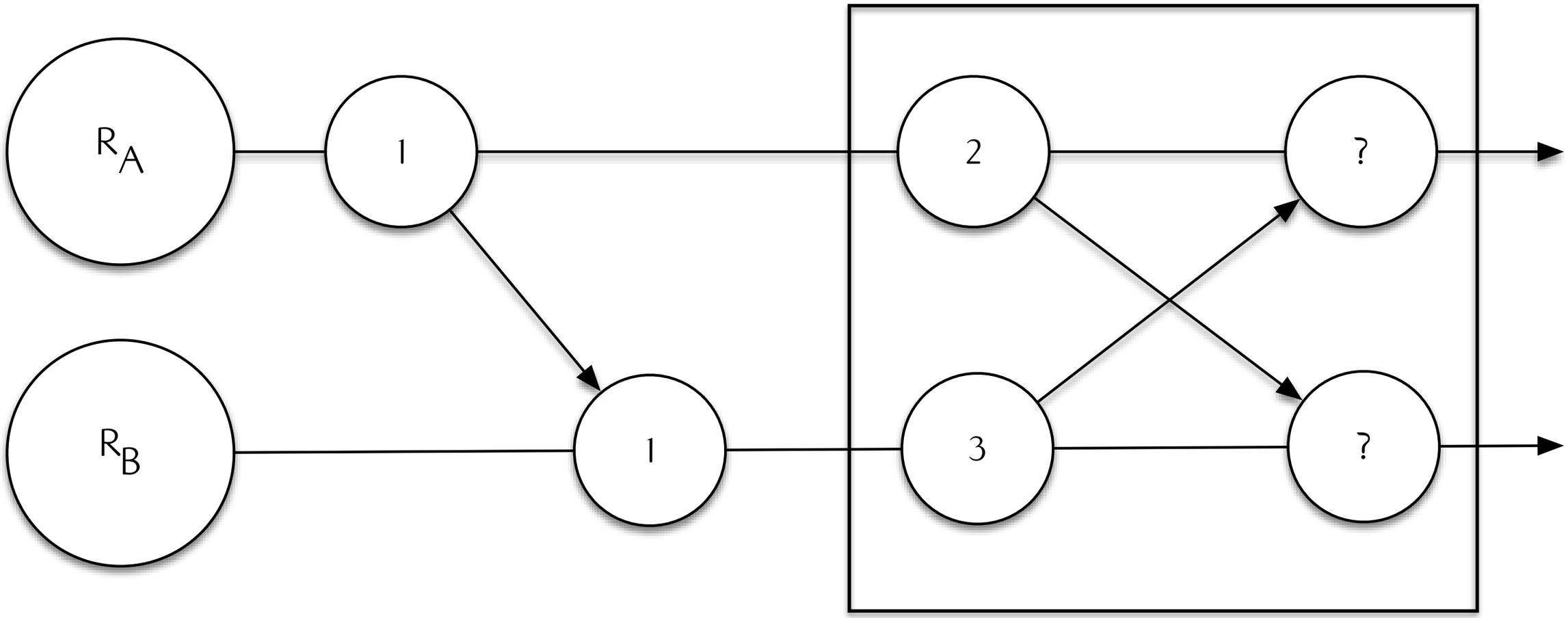
UNRELIABLE AT BEST
NON-DETERMINISTIC IF USED IN COMPUTATIONS

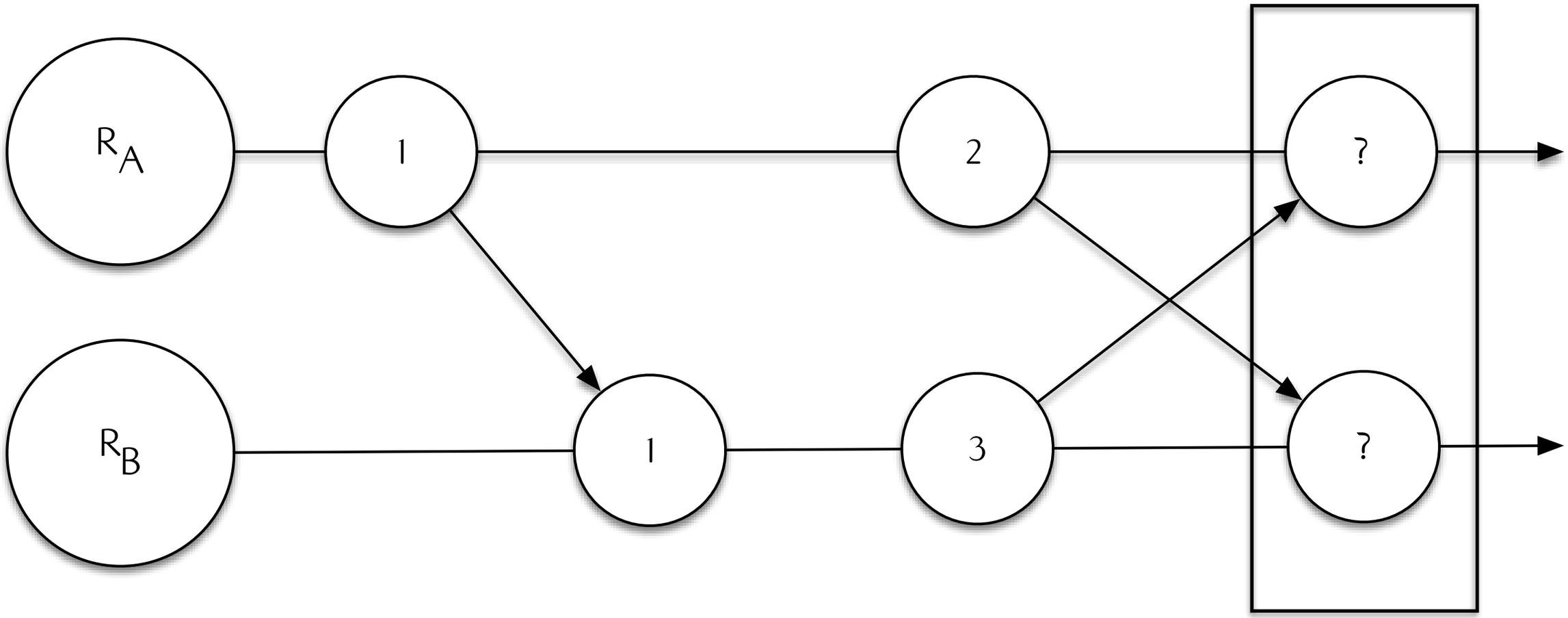
Corbett, James C., et al. "Spanner: Google's globally distributed database." ACM Transactions on Computer Systems (TOCS) 31.3 (2013): 8.

CONCURRENCY RECONCILED BY USER


R_A
R_B
1
1
2
3
?
?


R_A
1
2
?
R_B
1
3
?


R_A
1
2
?
R_B
1
3
?


R_A
1
2
?
R_B
1
3
?

CRDTs

CRDTs PROVIDE DETERMINISTIC RESOLUTION

CRDTs:

MAPS, SETS, COUNTERS, REGISTERS, GRAPHS
DETERMINISTIC RESOLUTION

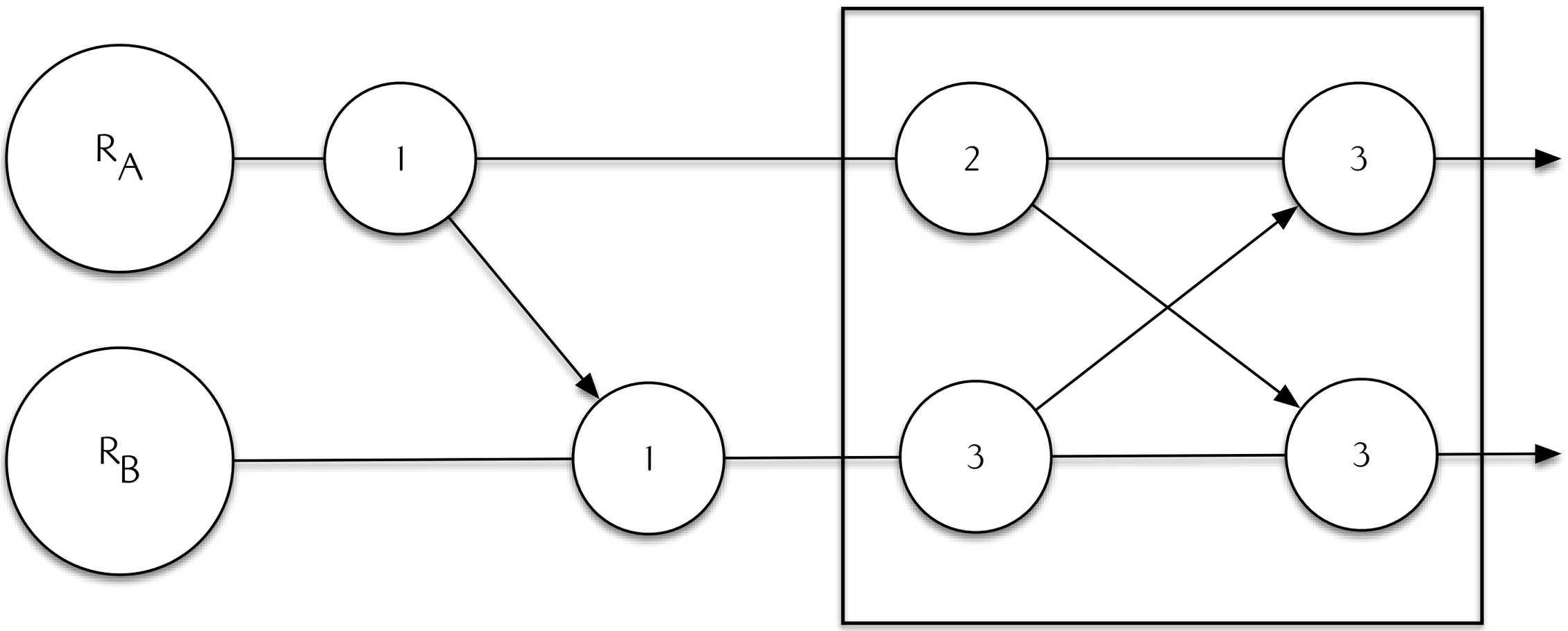
CRDTs REALIZE STRONG EVENTUAL CONSISTENCY

“CORRECT REPLICAS THAT HAVE DELIVERED THE SAME UPDATES HAVE EQUIVALENT STATE”

Shapiro, Marc, et al. "Conflict-free replicated data types." Stabilization, Safety, and Security of Distributed Systems. Springer Berlin Heidelberg, 2011. 386-400.

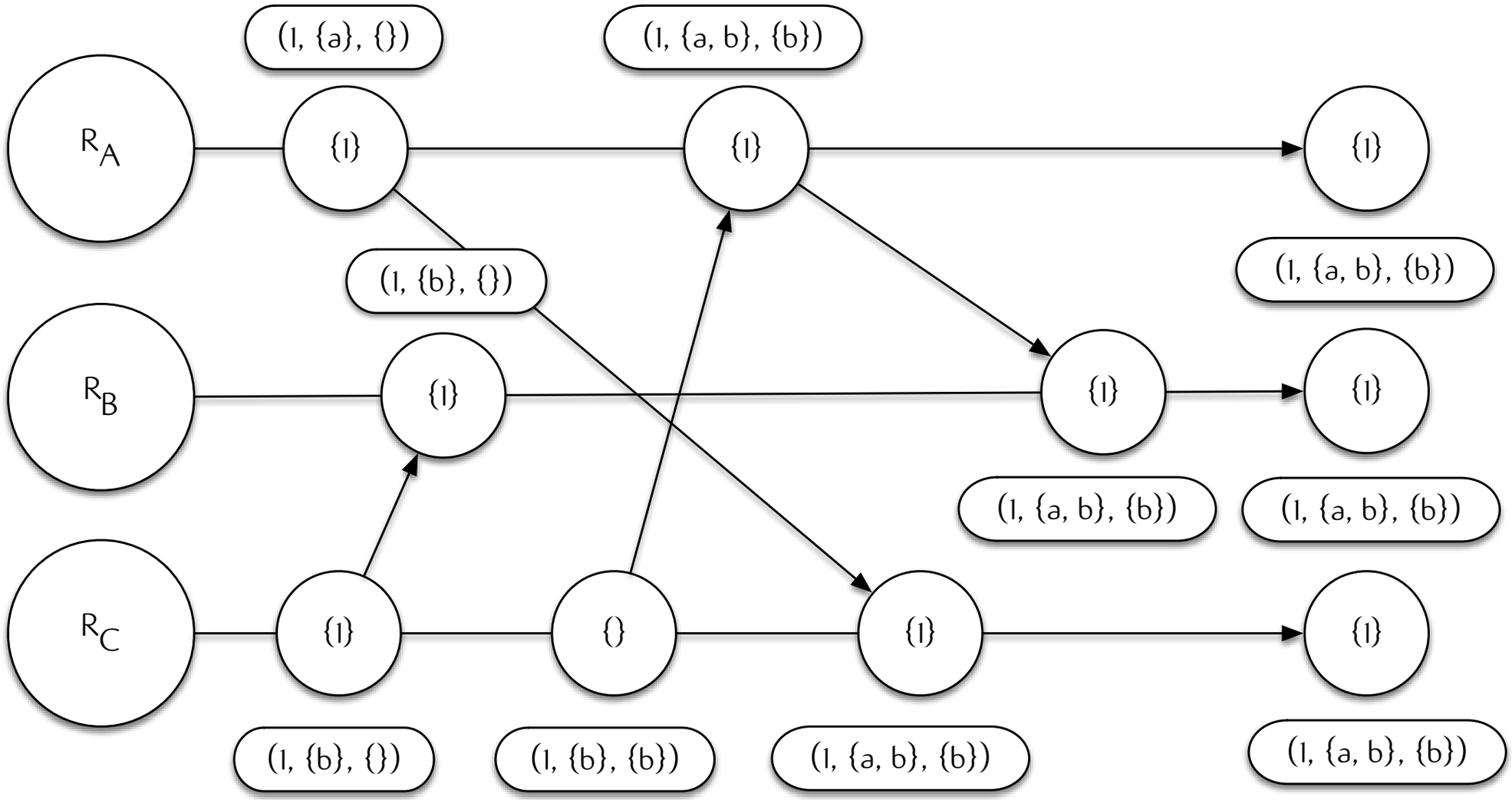
CRDTs EXAMPLE

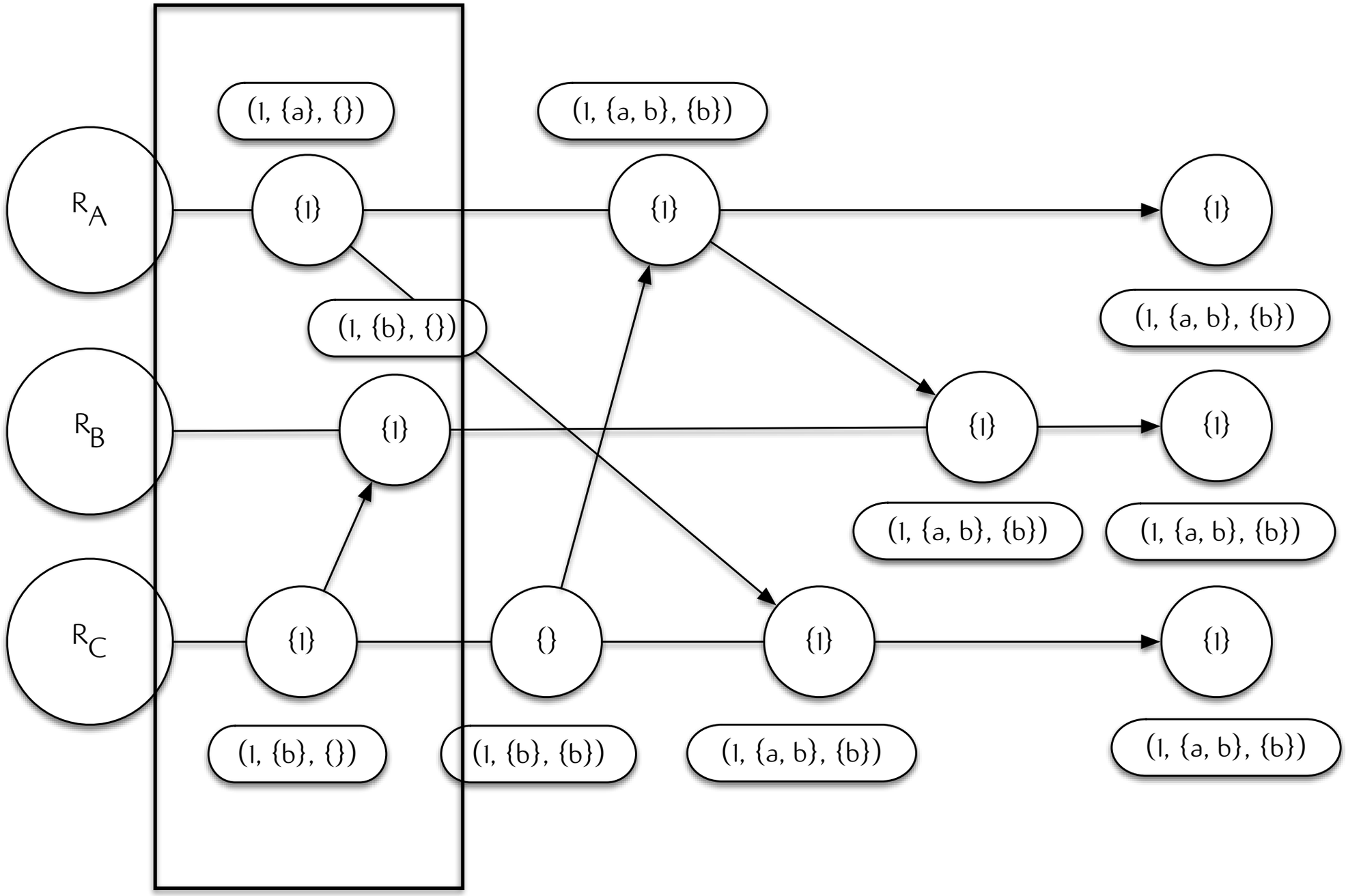
‘MAX’ REGISTER

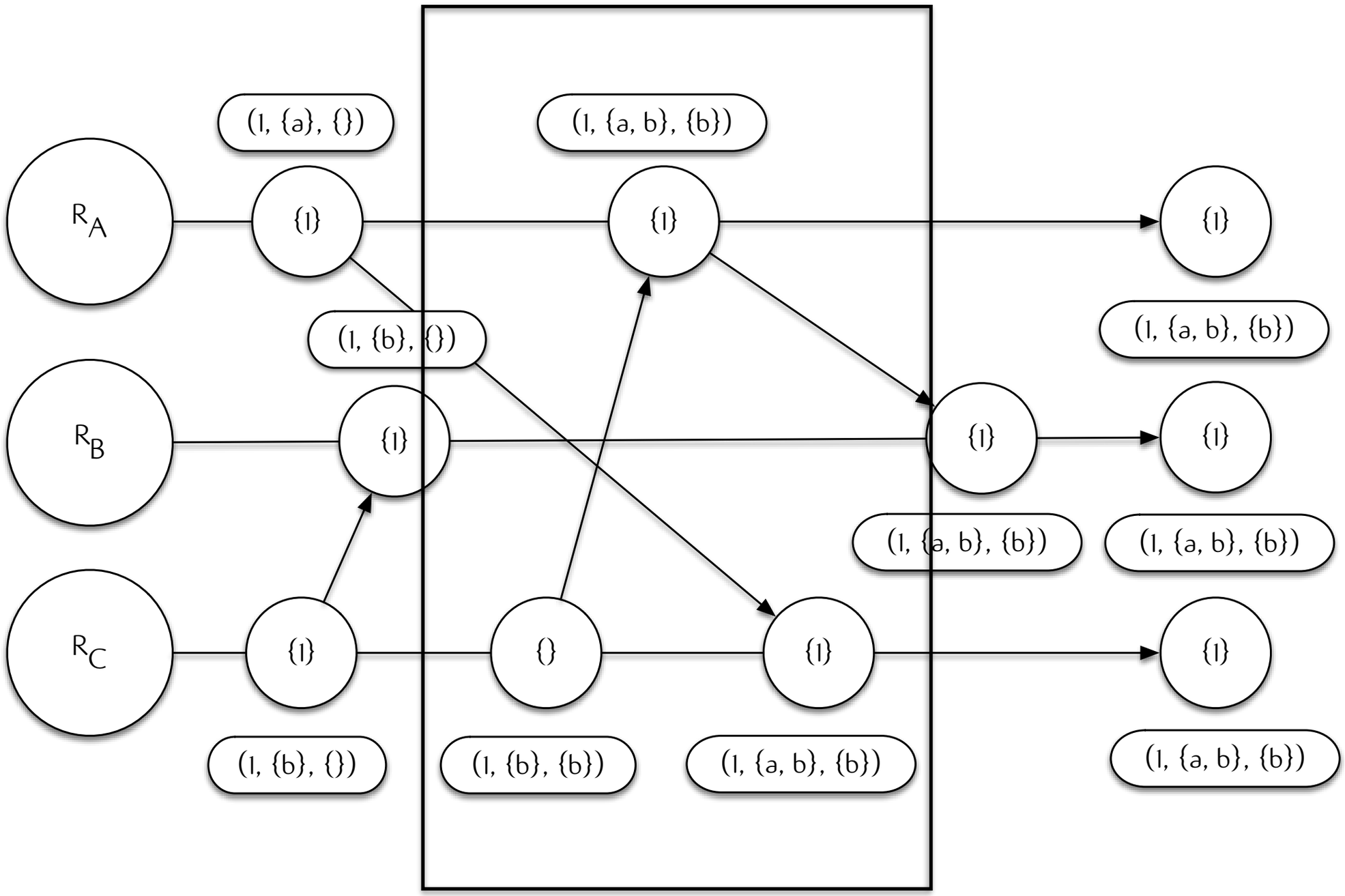

R_A
1
2
3
R_B
1
3
3

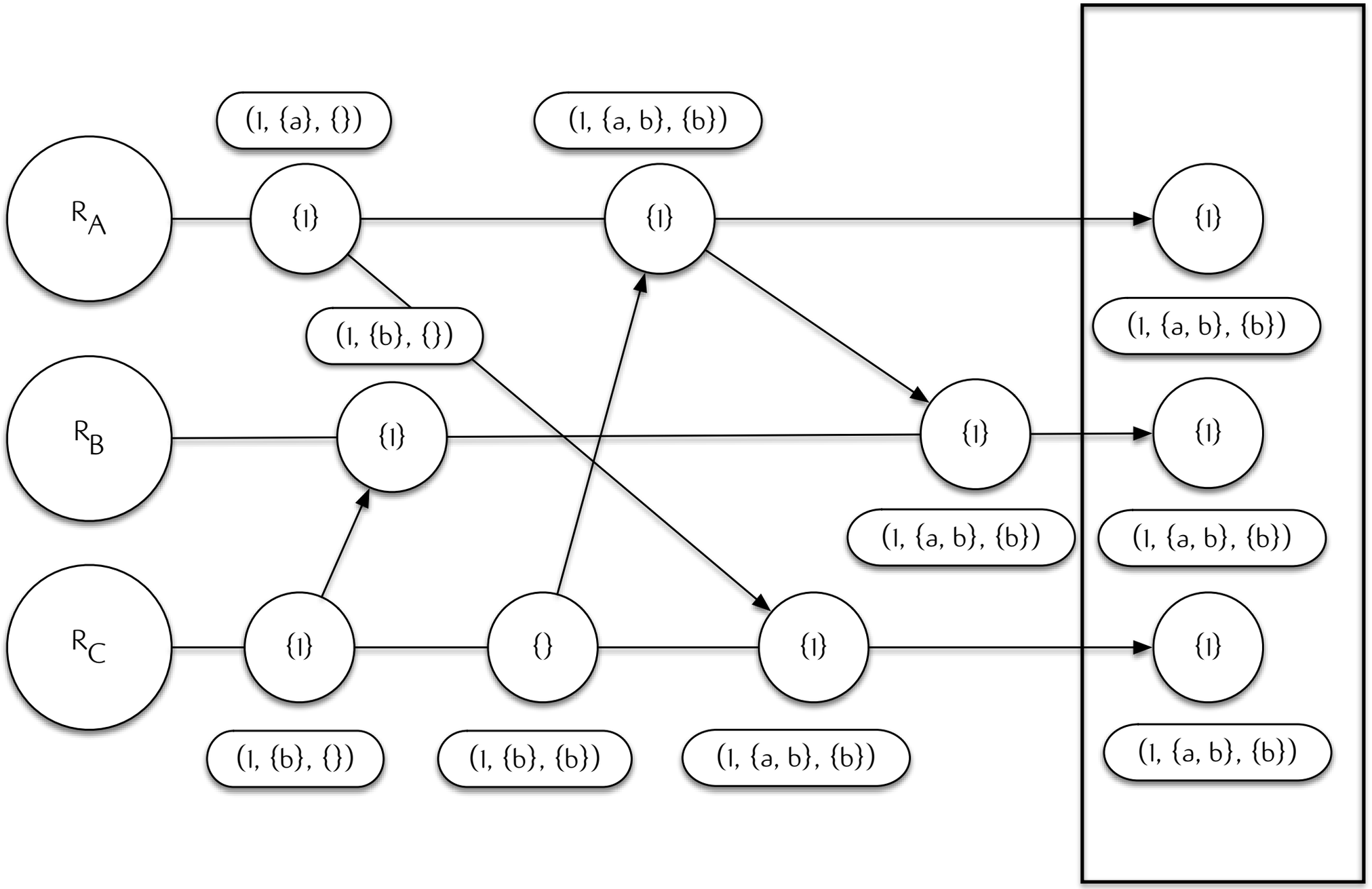
CRDTs EXAMPLE

‘ORSET’ SET

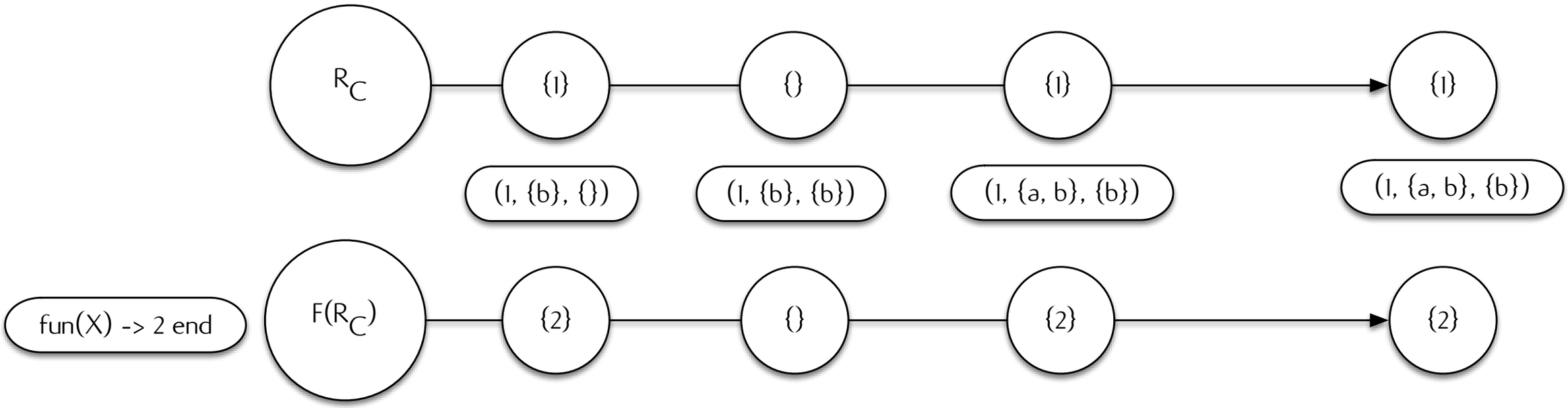

(l, {a}, {})
(l, {a, b}, {b})
R_A
{l}
{l}
{l}
(l, {b}, {})
(l, {a, b}, {b})
R_B
{l}
{l}
{l}
(l, {a, b}, {b})
(l, {a, b}, {b})
R_C
{l}
{}
{l}
{l}
(l, {b}, {})
(l, {b}, {})
(l, {a, b}, {b})
(l, {a, b}, {b})

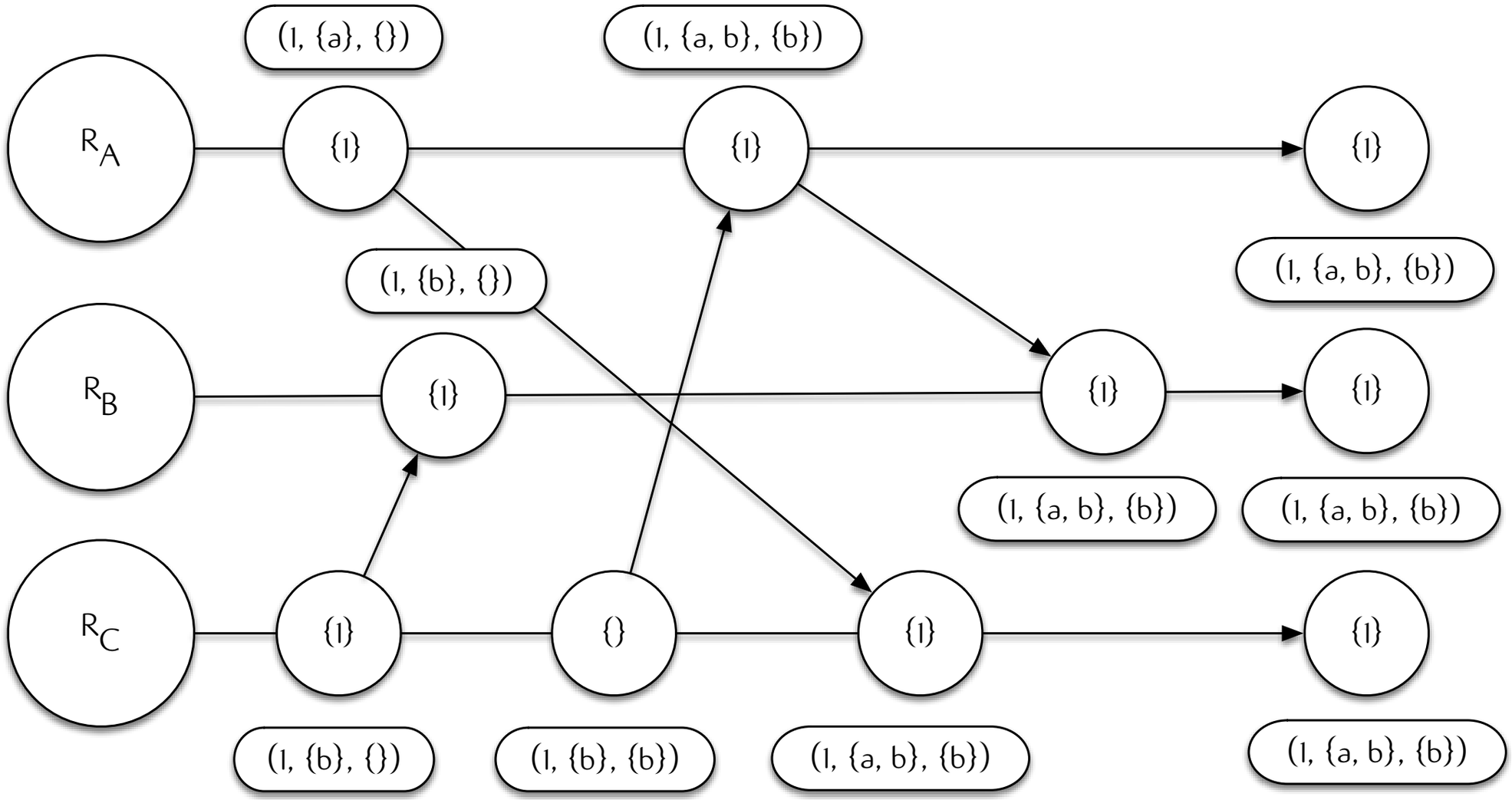

(l, {a}, {})
(l, {a, b}, {b})
R_A
{l}
{l}
{l}
(l, {b}, {})
(l, {a, b}, {b})
R_B
{l}
{l}
{l}
(l, {a, b}, {b})
(l, {a, b}, {b})
R_C
{l}
{}
{l}
{l}
(l, {b}, {})
(l, {b}, {})
(l, {a, b}, {b})
(l, {a, b}, {b})

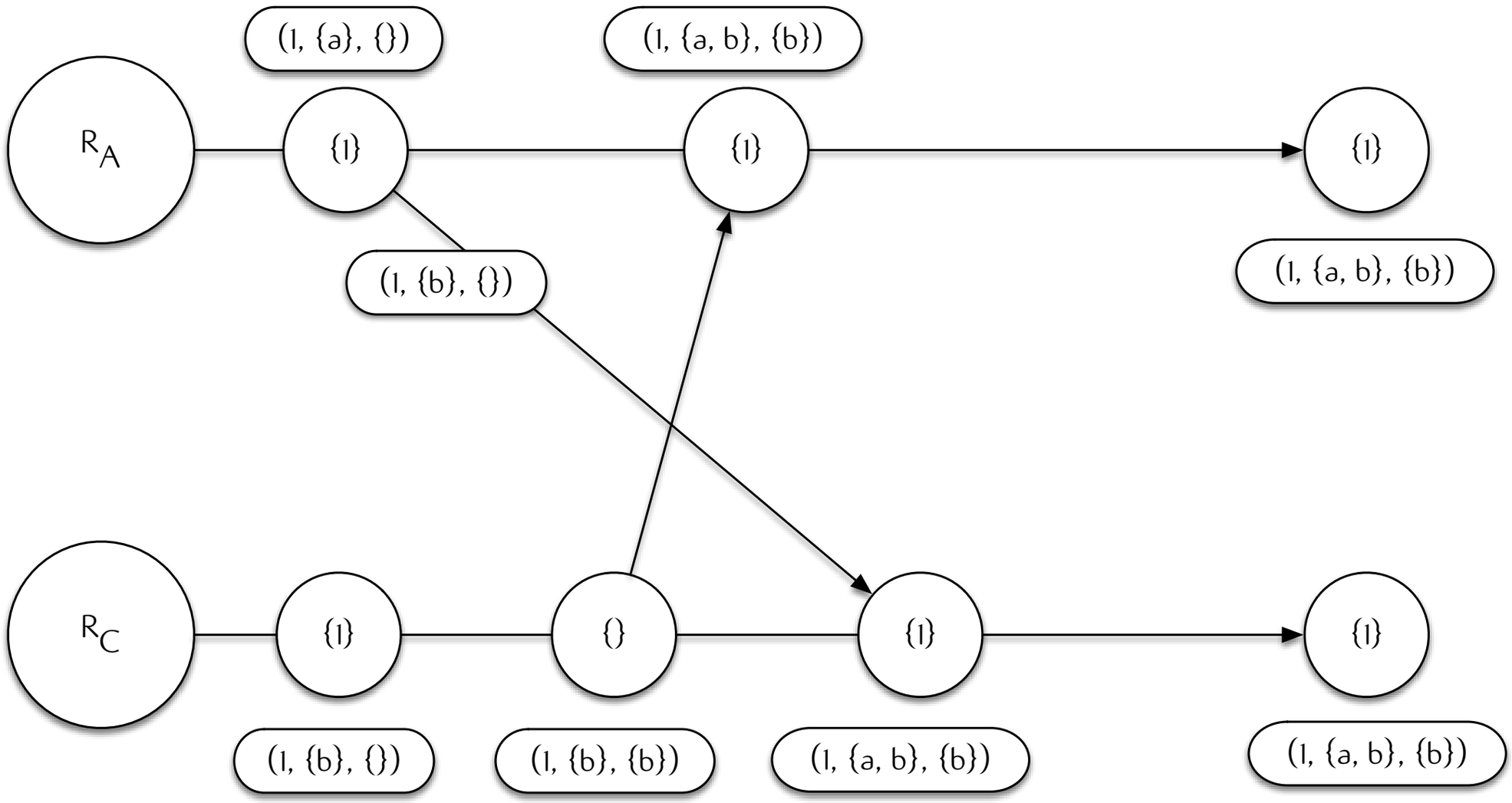

(l, {a}, {})
(l, {a, b}, {b})
R_A
{l}
{l}
{l}
(l, {b}, {})
(l, {a, b}, {b})
R_B
{l}
{l}
{l}
(l, {a, b}, {b})
(l, {a, b}, {b})
R_C
{l}
{}
{l}
{l}
(l, {b}, {})
(l, {b}, {b})
(l, {a, b}, {b})
(l, {a, b}, {b})

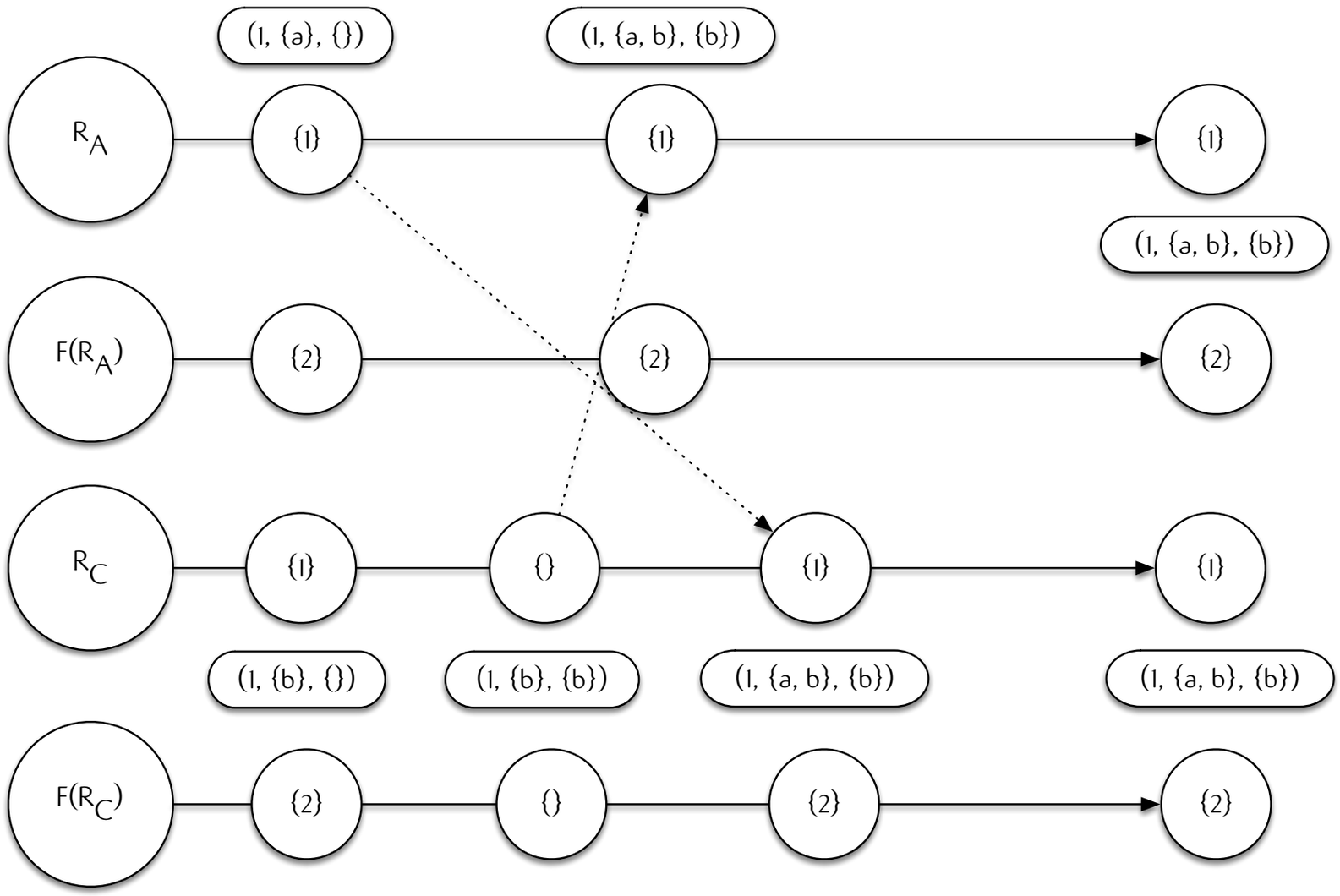

(1, {a}, {})
(1, {a, b}, {b})
R_A
{1}
{1}
{1}
(1, {b}, {})
(1, {a, b}, {b})
R_B
{1}
{1}
{1}
(1, {a, b}, {b})
(1, {a, b}, {b})
R_C
{1}
{}
{1}
{1}
(1, {b}, {})
(1, {b}, {b})
(1, {a, b}, {b})
(1, {a, b}, {b})

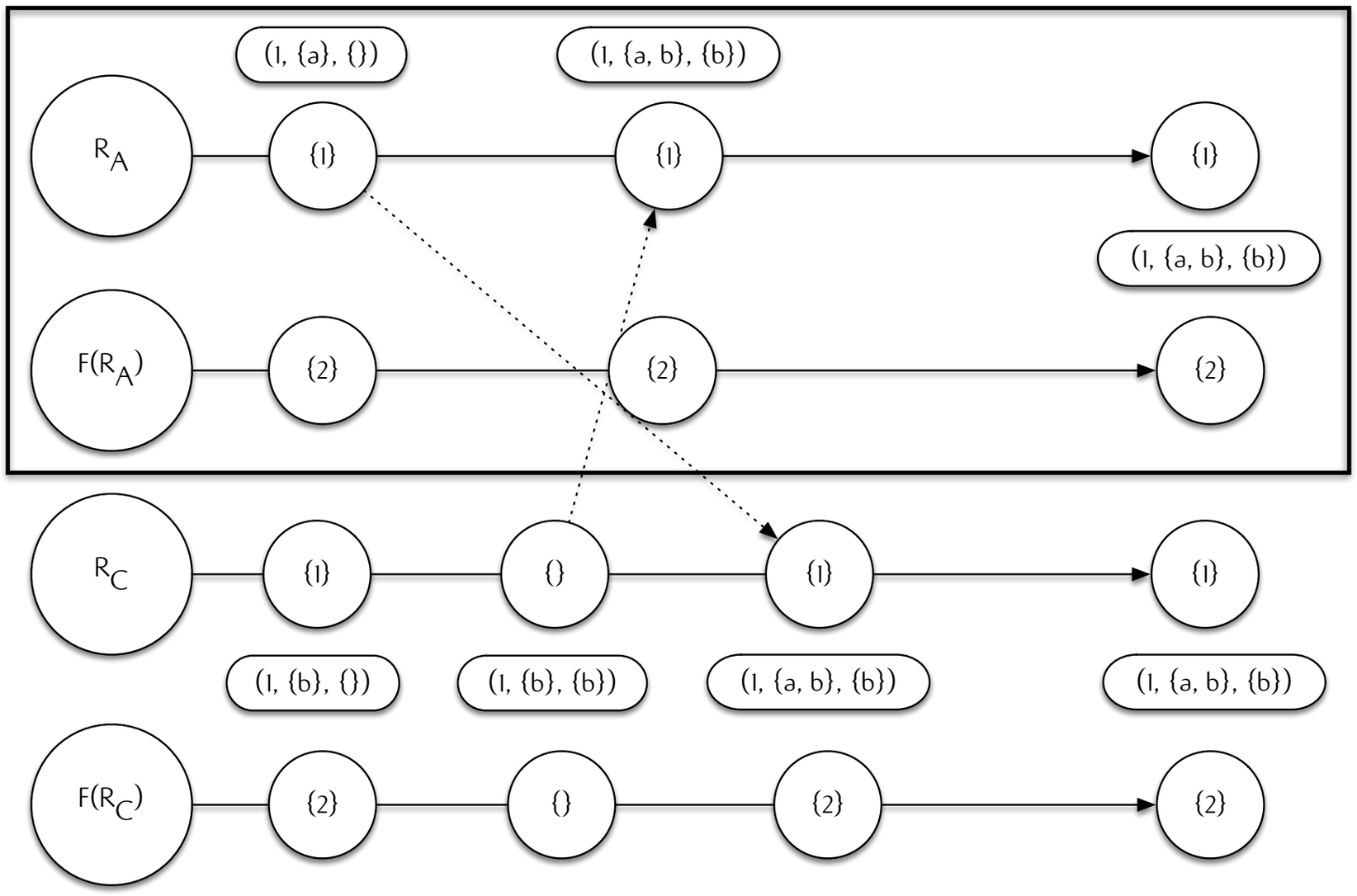
COMPOSITION IS NONTRIVIAL

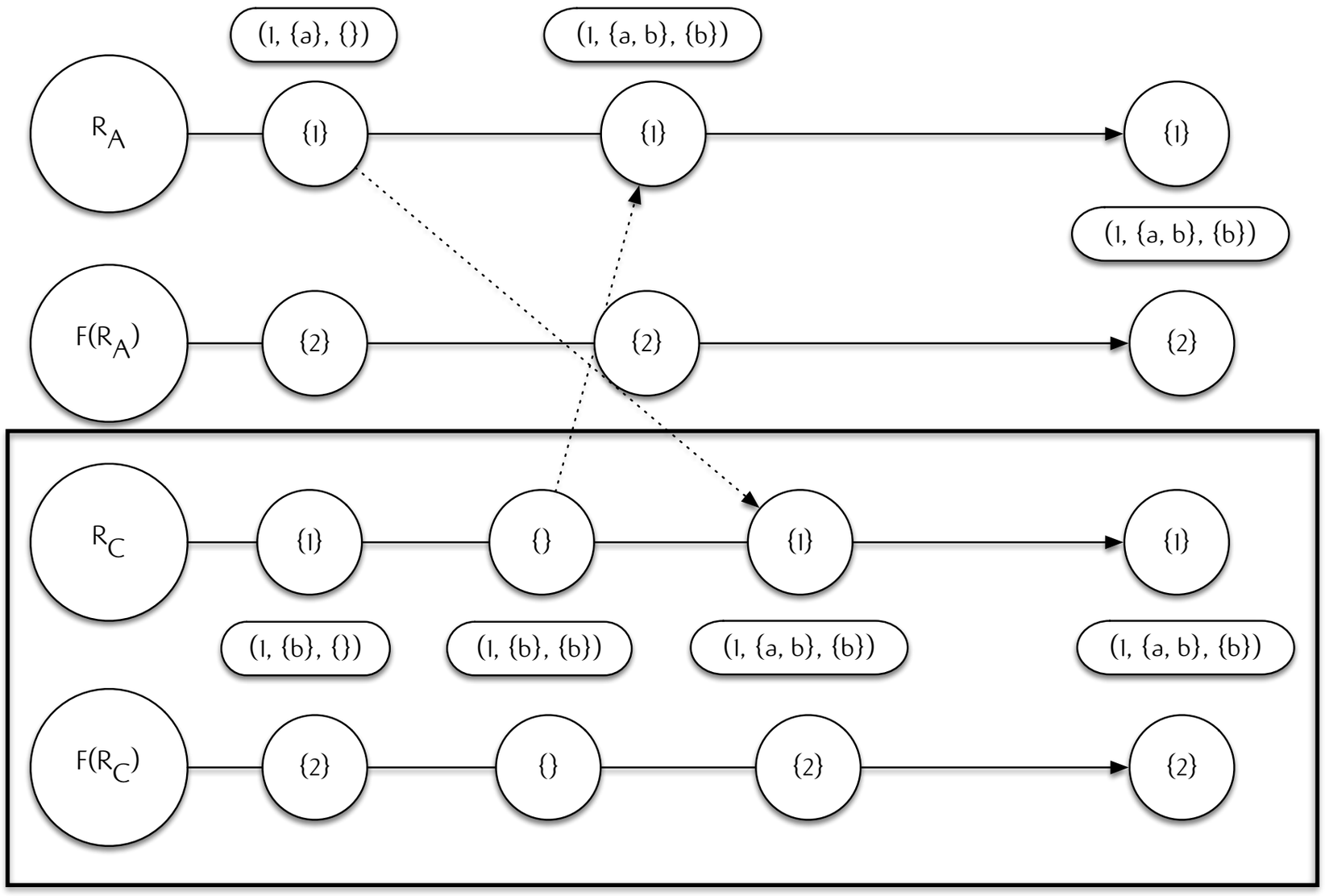

R_C
{l}
{}
{l}
{l}
(l, {b}, {})
(l, {b}, {b})
(l, {a, b}, {b})
(l, {a, b}, {b})
fun(X) -> 2 end
$F(R_C)$
{2}
{}
{2}
{2}

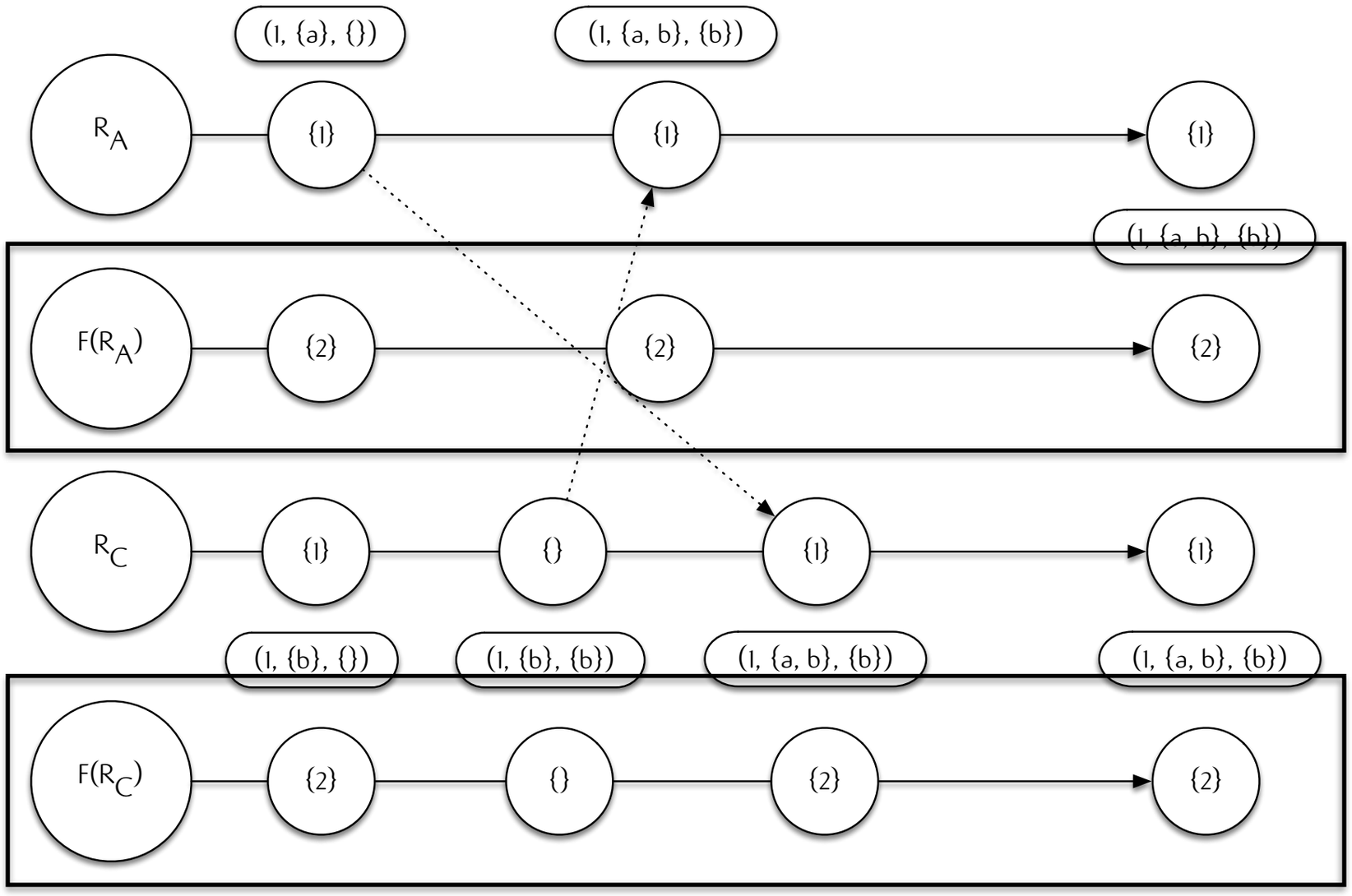

(l, {a}, {})
(l, {a, b}, {b})
R_A
{l}
{l}
{l}
(l, {b}, {})
(l, {a, b}, {b})
R_B
{l}
{l}
{l}
(l, {a, b}, {b})
(l, {a, b}, {b})
R_C
{l}
{}
{l}
{l}
(l, {b}, {})
(l, {b}, {})
(l, {a, b}, {b})
(l, {a, b}, {b})


(l, {a}, {})
(l, {a, b}, {b})
R_A
{l}
{l}
{l}
(l, {b}, {})
(l, {a, b}, {b})
R_C
{l}
{}
{l}
{l}
(l, {b}, {})
(l, {b}, {b})
(l, {a, b}, {b})
(l, {a, b}, {b})


(1, {a}, {})
(1, {a, b}, {b})
R_A
{1}
{1}
{1}
(1, {a, b}, {b})
F(R_A)
{2}
{2}
{2}
R_C
{1}
{}
{1}
{1}
(1, {b}, {})
(1, {b}, {b})
(1, {a, b}, {b})
(1, {a, b}, {b})
F(R_C)
{2}
{}
{2}
{2}


(I, {a}, {})
(I, {a, b}, {b})
R_A
{I}
{I}
{I}
(I, {a, b}, {b})
$F(R_A)$
{2}
{2}
{2}
R_C
{I}
{}
{I}
{I}
(I, {b}, {})
(I, {b}, {b})
(I, {a, b}, {b})
(I, {a, b}, {b})
$F(R_C)$
{2}
{}
{2}
{2}


(1, {a}, {})
(1, {a, b}, {b})
R_A
{1}
{1}
{1}
(1, {a, b}, {b})
$F(R_A)$
{2}
{2}
{2}
R_C
{1}
{}
{1}
{1}
(1, {b}, {})
(1, {b}, {b})
(1, {a, b}, {b})
(1, {a, b}, {b})
$F(R_C)$
{2}
{}
{2}
{2}


(1, {a}, {})
(1, {a, b}, {b})
R_A
{1}
{1}
{1}
(1, {a, b}, {b})
F(R_A)
{2}
{2}
{2}
R_C
{1}
{}
{1}
{1}
(1, {b}, {})
(1, {b}, {b})
(1, {a, b}, {b})
(1, {a, b}, {b})
F(R_C)
{2}
{}
{2}
{2}

COMPOSITION:

USER OBSERVABLE VALUE VS. STATE METADATA MAPPING IS NONTRIVIAL WITHOUT MAPPING METADATA; UNMERGABLE

Brown, Russell, et al. "Riak dt map: A composable, convergent replicated dictionary." Proceedings of the First Workshop on Principles and Practice of Eventual Consistency. ACM, 2014.
Conway, Neil, et al. "Logic and lattices for distributed programming." Proceedings of the Third ACM Symposium on Cloud Computing. ACM, 2012.
Meiklejohn, Christopher. "On the composability of the Riak DT map: expanding from embedded to multi-key structures." Proceedings of the First Workshop on Principles and Practice of Eventual Consistency. ACM, 2014.

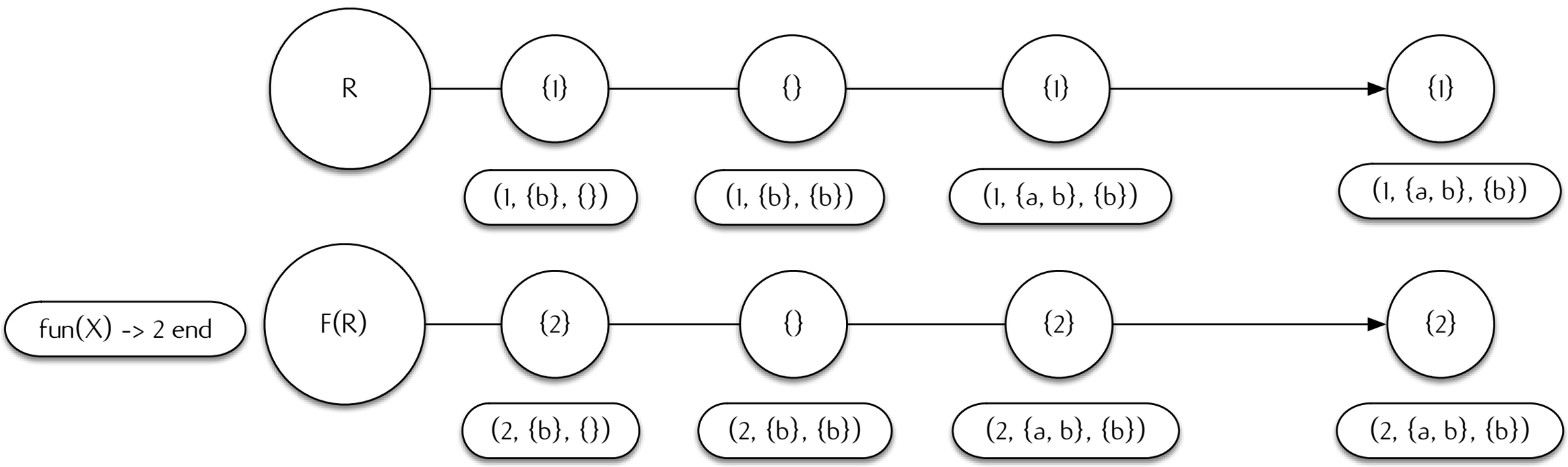
LASP

WHAT IS LASP

LASP

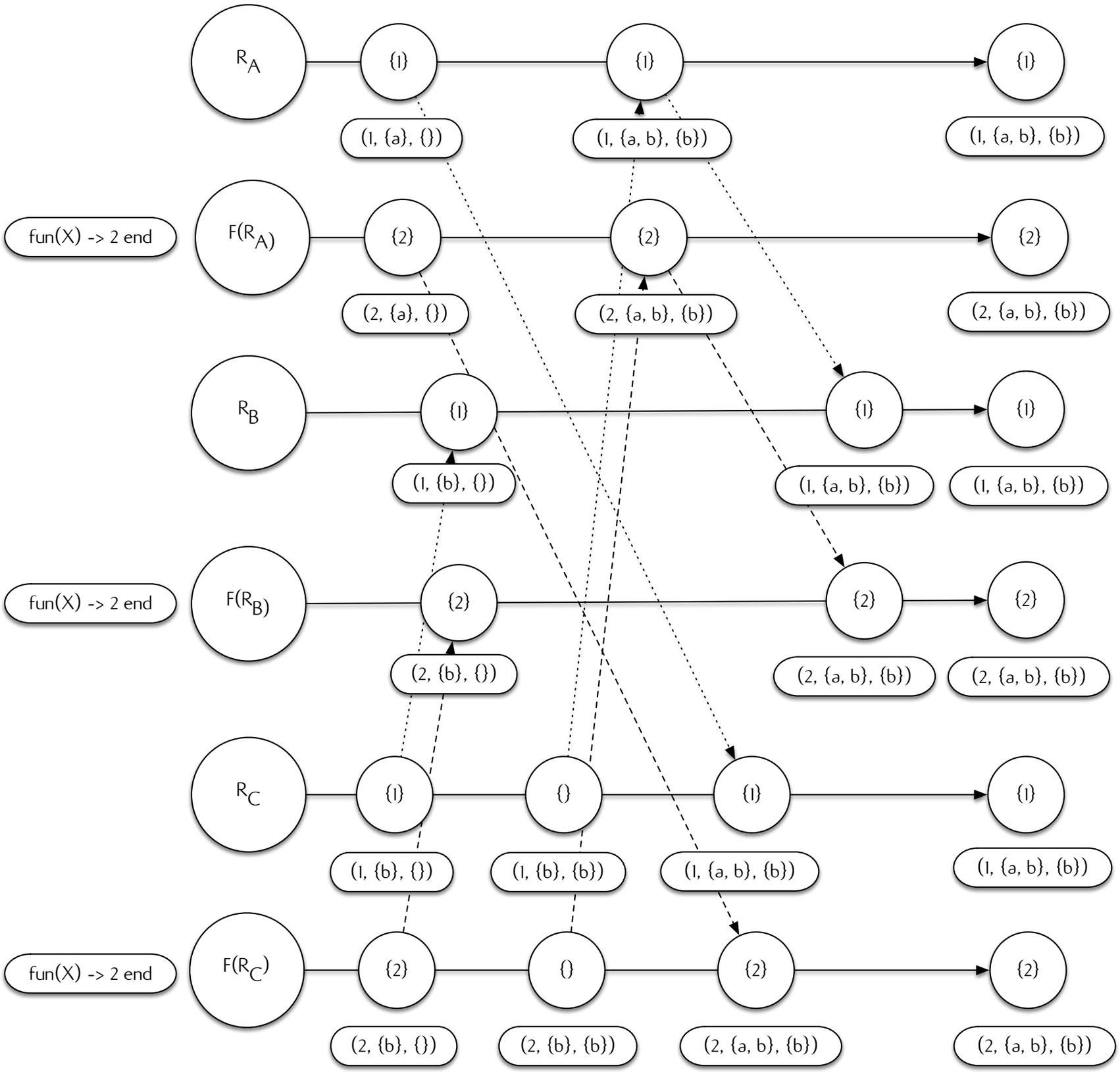
LATTICE PROCESSING

CENTRALIZED SEMANTICS:
DATATYPE COMPOSITION


R
{l}
{}
{l}
{l}
(l, {b}, {})
(l, {b}, {b})
(l, {a, b}, {b})
(l, {a, b}, {b})
fun(X) -> 2 end
F(R)
{2}
{}
{2}
{2}
(2, {b}, {})
(2, {b}, {b})
(2, {a, b}, {b})
(2, {a, b}, {b})

DISTRIBUTED SEMANTICS:

DATATYPE COMPOSITION


R_A
{I}
{I}
{I}
(I, {a}, {})
(I, {a, b}, {b})
(I, {a, b}, {b})
fun(X) -> 2 end
$F(R_A)$
{2}
{2}
{2}
(2, {a}, {})
(2, {a, b}, {b})
(2, {a, b}, {b})
R_B
{I}
{I}
{I}
(I, {b}, {})
(I, {a, b}, {b})
(I, {a, b}, {b})
fun(X) -> 2 end
$F(R_B)$
{2}
{2}
{2}
(2, {b}, {})
(2, {a, b}, {b})
(2, {a, b}, {b})
R_C
{I}
{}
{I}
{I}
(I, {b}, {})
(I, {b}, {b})
(I, {a, b}, {b})
(I, {a, b}, {b})
fun(X) -> 2 end
$F(R_C)$
{2}
{}
{2}
{2}
(2, {b}, {})
(2, {b}, {b})
(2, {a, b}, {b})
(2, {a, b}, {b})

DISTRIBUTED SEMANTICS:
SYSTEM COMPOSITION

CENTRALIZED RUNTIME:
SINGLE NODE MODEL

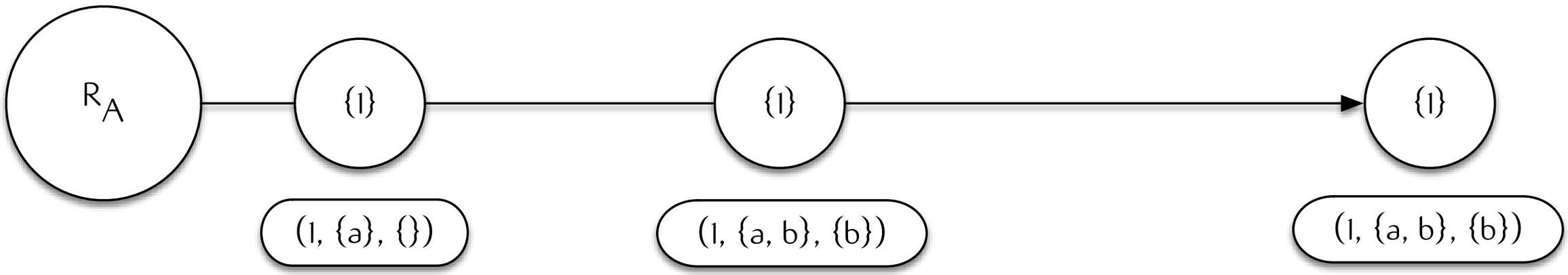
DISTRIBUED RUNTIME:
MULTI-NODE MODEL

SEMANTICS

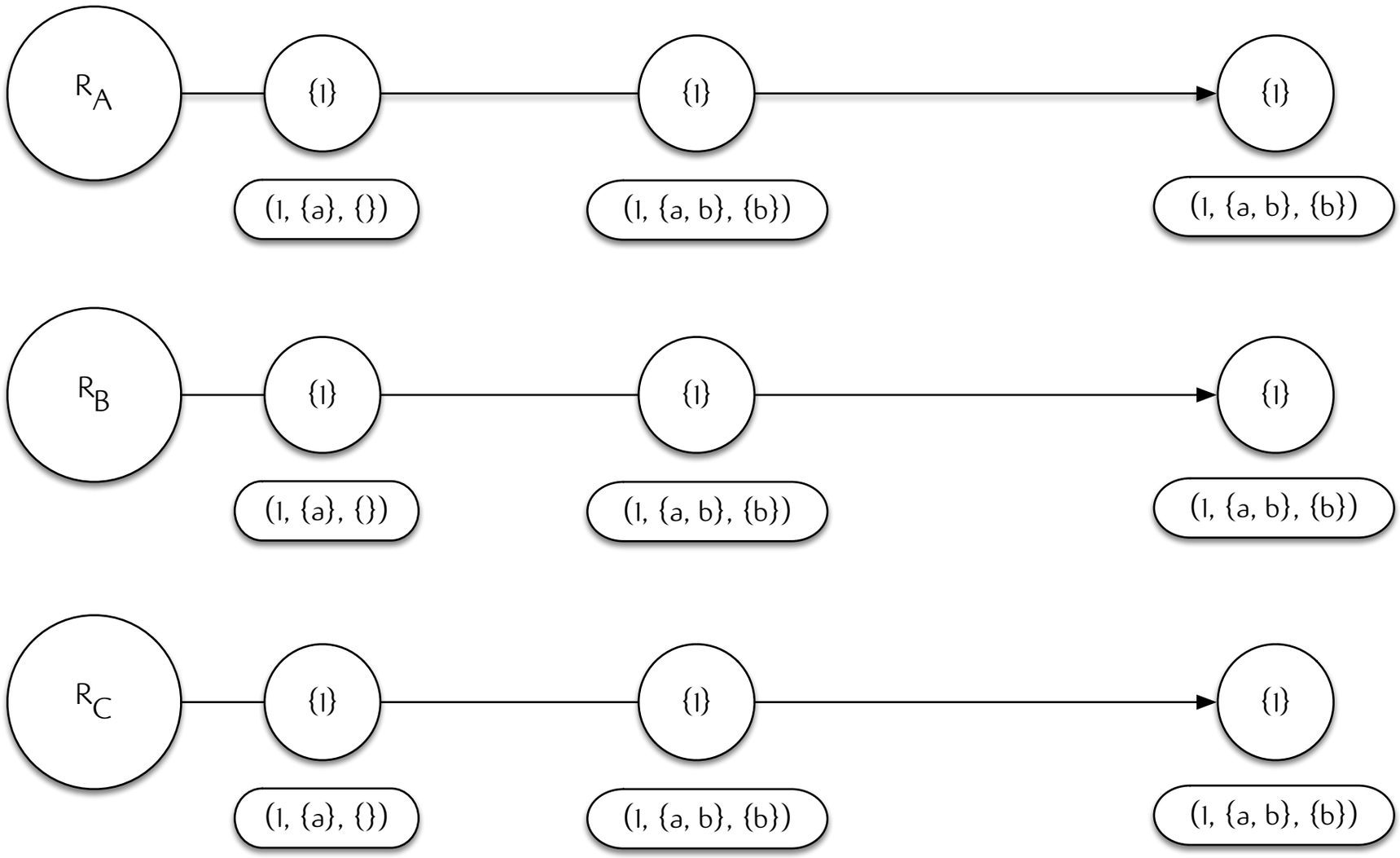
STREAMS

STREAMS:

CENTRALIZED EXECUTION

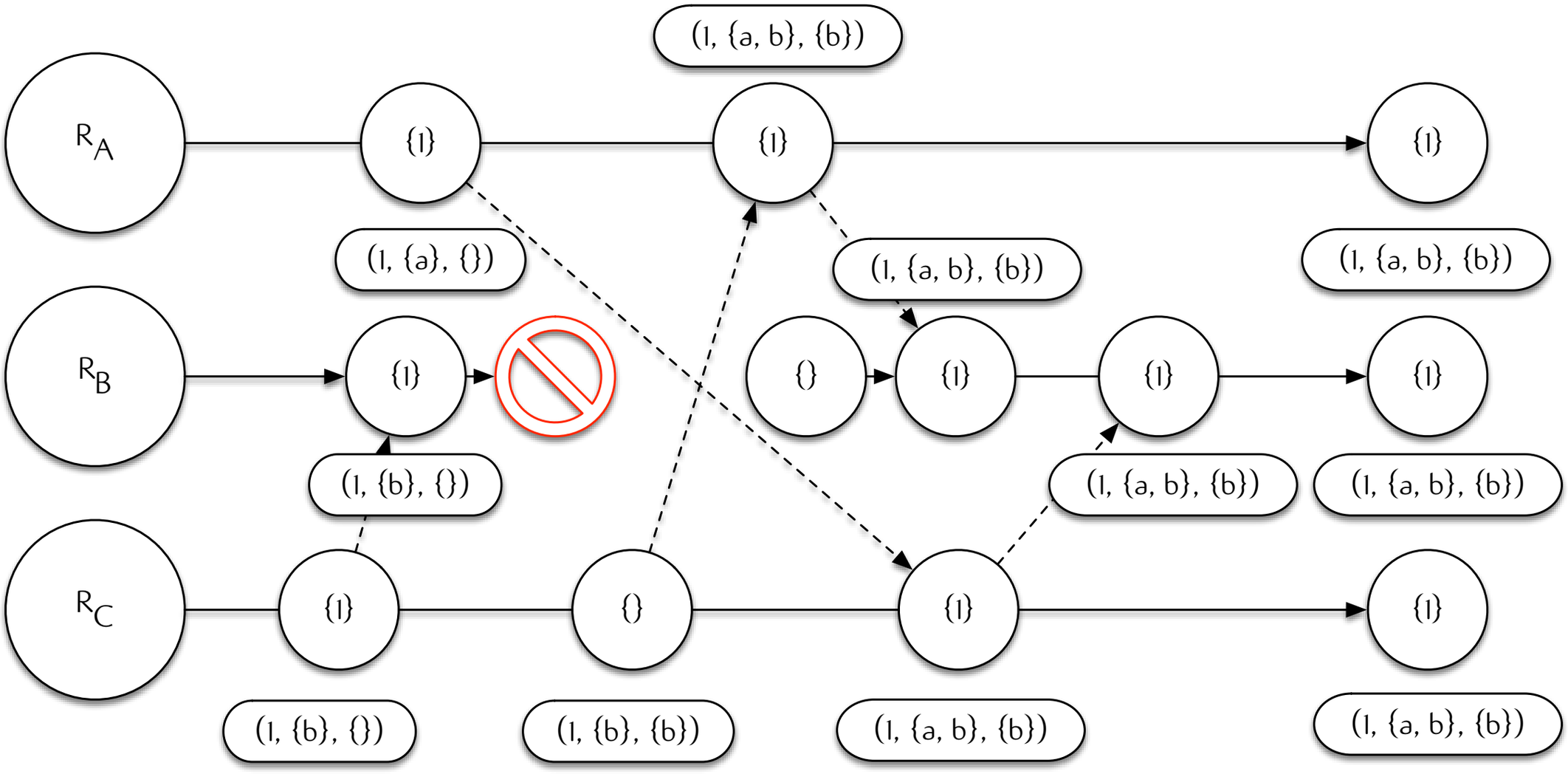

R_A
{1}
{1}
{1}
(1, {a}, {})
(1, {a, b}, {b})
(1, {a, b}, {b})

STREAMS:
DISTRIBUTED EXECUTION


R_A
{l}
{l}
{l}
(l, {a}, {})
(l, {a, b}, {b})
(l, {a, b}, {b})
R_B
{l}
{l}
{l}
(l, {a}, {})
(l, {a, b}, {b})
(l, {a, b}, {b})
R_C
{l}
{l}
{l}
(l, {a}, {})
(l, {a, b}, {b})
(l, {a, b}, {b})

STREAMS:

CONCURRENCY; FAILURE; ANTI-ENTROPY


R_A
R_B
R_C
{I}
{}
(I, {a, b}, {b})
(I, {a}, {})
(I, {b}, {})
(I, {b}, {b})

STREAMS:

DISTRIBUTED EXECUTION DETERMINISTIC
REQUIRES EVENTUAL DELIVERY; ANTI-ENTROPY

DECLARE:

CREATE A STREAM OF A GIVEN CRDT TYPE

INITIALIZE STATE AT BOTTOM VALUE

BIND:

ASSIGN A VALUE TO THE STREAM
MERGE OF CURRENT AND NEW VALUE

MONOTONIC READ

MONOTONIC READ:

ENSURES FORWARD PROGRESS
BLOCKS ON INFLATIONS (OR STRICT INFLATIONS)

COMPOSITIONS ARE PROCESSES:

BLOCK FOR CHANGE IN INPUT
COMPUTE CHANGE
PROPAGATE CHANGE TO OUTPUT

FUNCTIONAL

FUNCTIONAL:

MAP; FILTER; FOLD

SET-THEORETIC

SET-THEORETIC:

PRODUCT; INTERSECTION; UNION

ARCHITECTURE

LASP STORE

SHARED VARIABLE STORE

LASP BACKENDS

LEVELDB, BITCASK, ETS

LASP CRDTs

PROVIDED BY RIAK DT

LASP ARCHITECTURE

CENTRALIZED SEMANTICS

STORE:

SHARED VARIABLE STORE

PROCESSESS SYNCHRONIZE ON VARIABLES

LASP ARCHITECTURE

DISTRIBUTED SEMANTICS

STORE:

REPLICATED, SHARDED VARIABLE STORE

PROCESSESS SYNCHRONIZE ON VARIABLES

REPLICATED:

DISTRIBUTED WITH RIAK CORE

QUORUM REQUESTS; ANTI-ENTROPY PROTOCOL

HYBRID:

DISTRIBUTE PROGRAMS; R/W WITH LOCAL STORE
CENTRALIZED EXECUTION

EXAMPLES

```
%% Create initial set.
{ok, S1} = lasp:declare(Type),

%% Add elements to initial set and update.
{ok, _} = lasp:update(S1, {add_all, [1,2,3]}, a),

%% Create second set.
{ok, S2} = lasp:declare(Type),

%% Apply map.
ok = lasp:map(S1, fun(X) -> X * 2 end, S2),
```

```
%% Create initial set.
{ok, S1} = lasp_core:declare(Type, Store),

%% Add elements to initial set and update.
{ok, _} = lasp_core:update(S1, {add_all, [1,2,3]}, a, Store),

%% Create second set.
{ok, S2} = lasp_core:declare(Type, Store),

%% Apply map.
ok = lasp_core:map(S1, fun(X) -> X * 2 end, S2, Store),
```

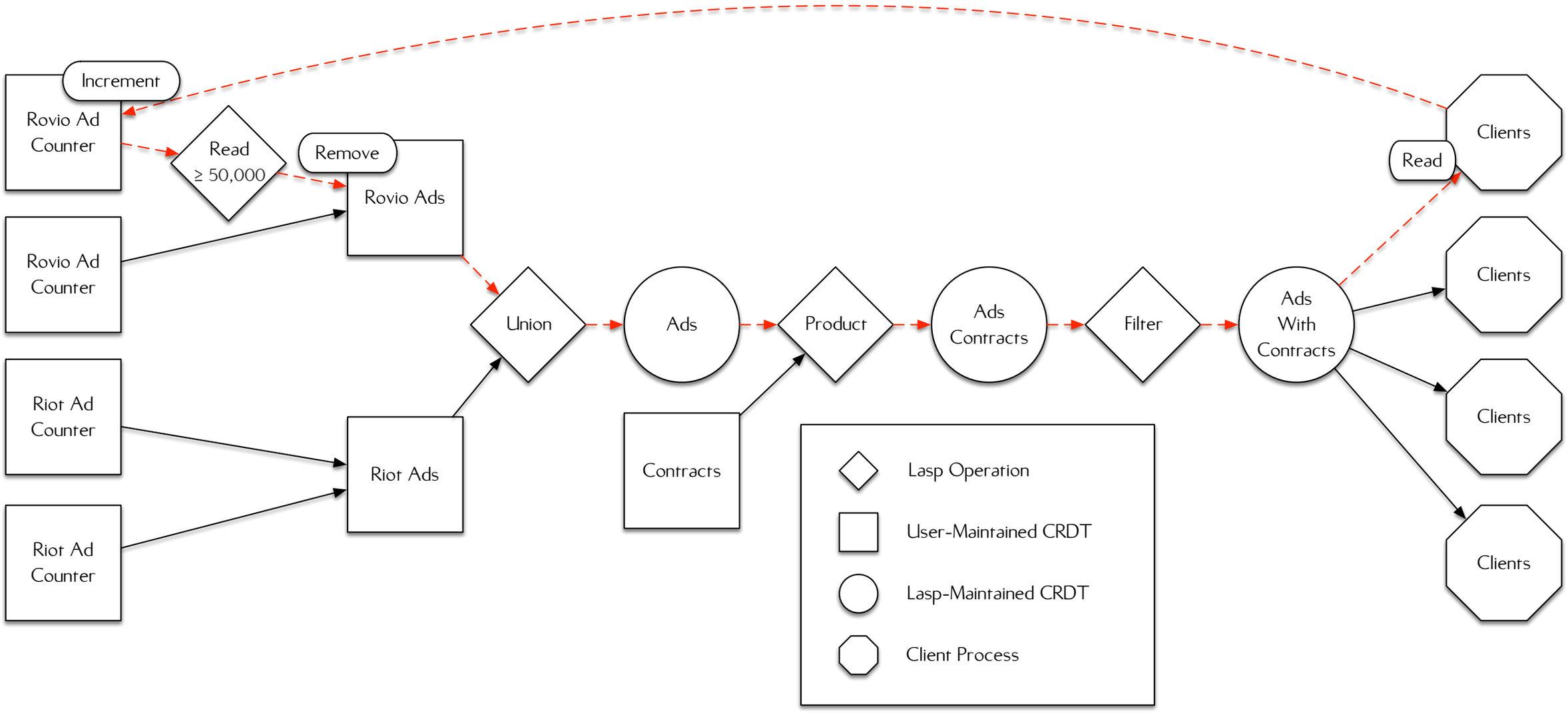
AD COUNTER

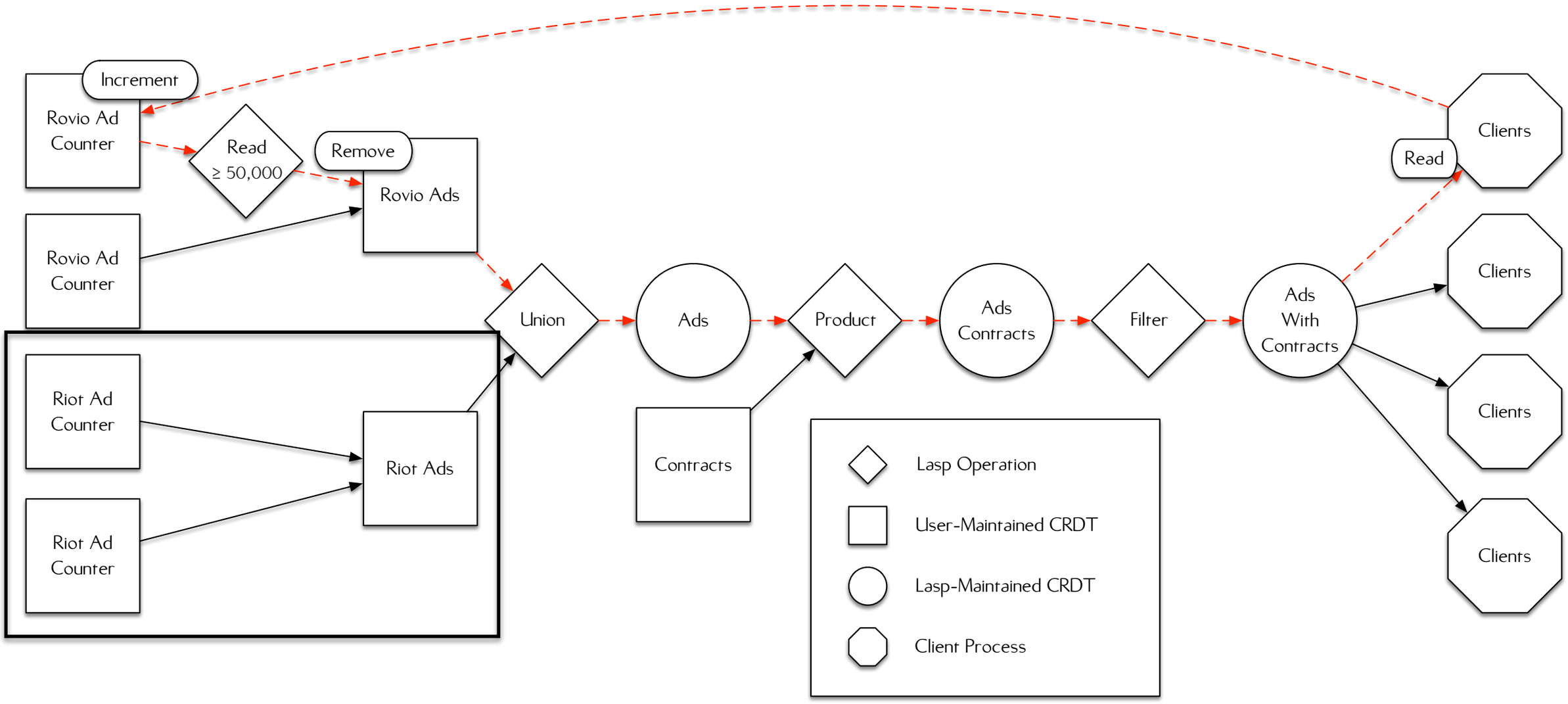
AD COUNTER:

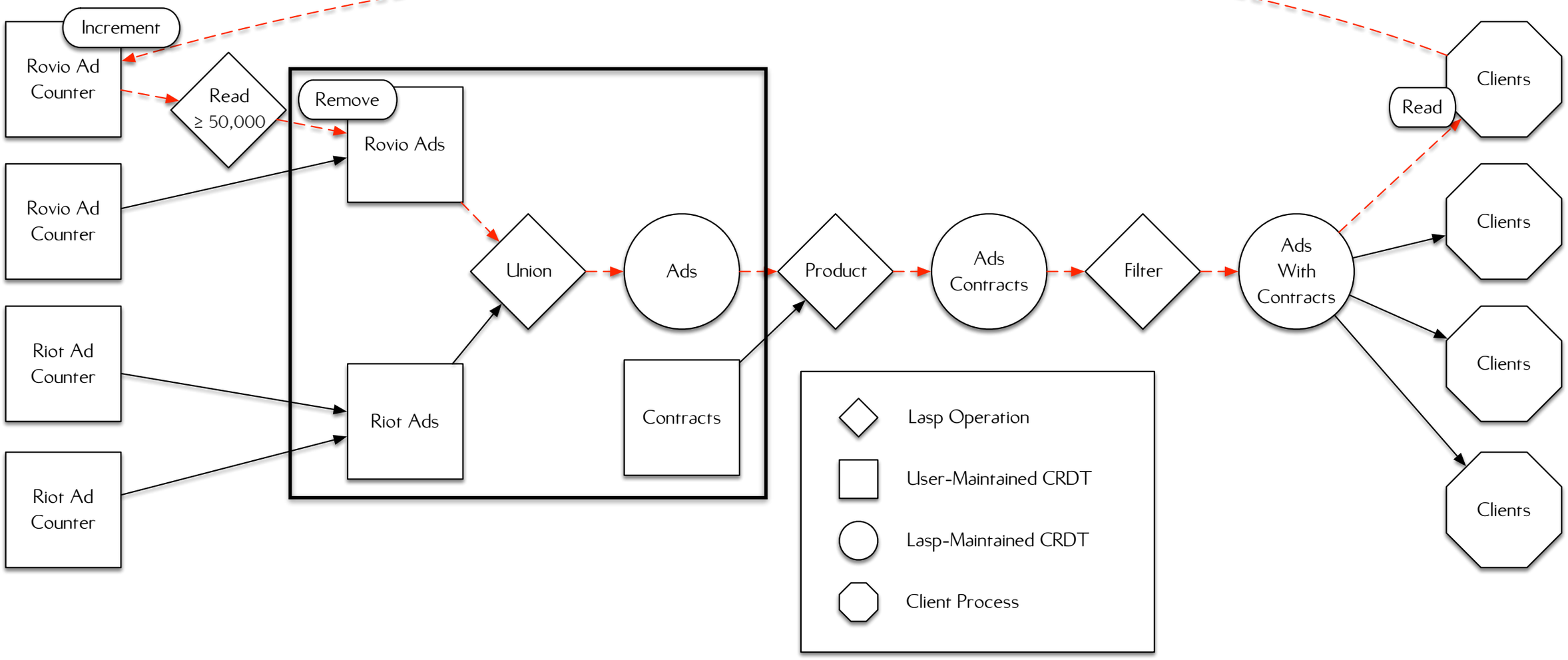
TRACKS AD IMPRESSIONS
PUSHES ADVERTISEMENTS TO THE CLIENT
DISABLES AD AT 50,000+ IMPRESSIONS
CLIENTS DISPLAY ADS WHEN OFFLINE

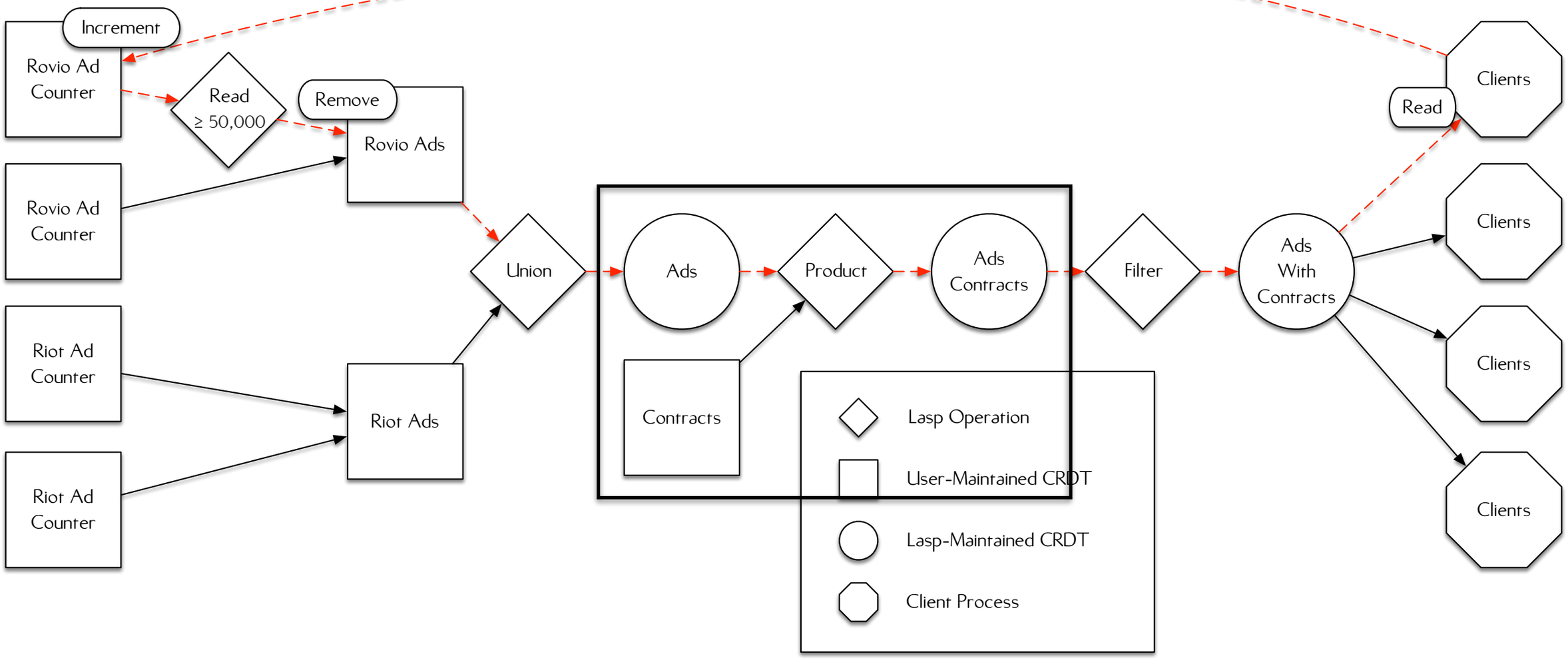
AD COUNTER

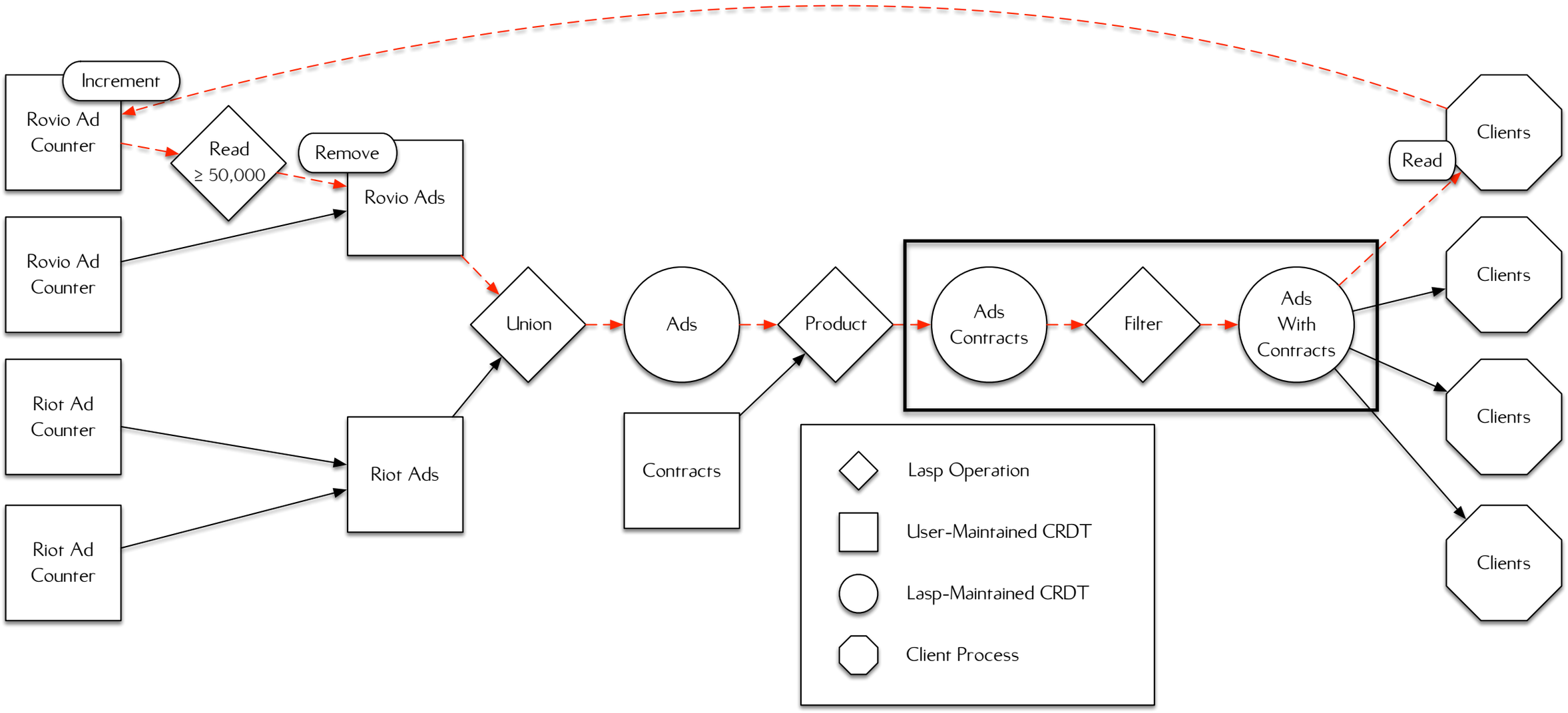
INFORMATION FLOW

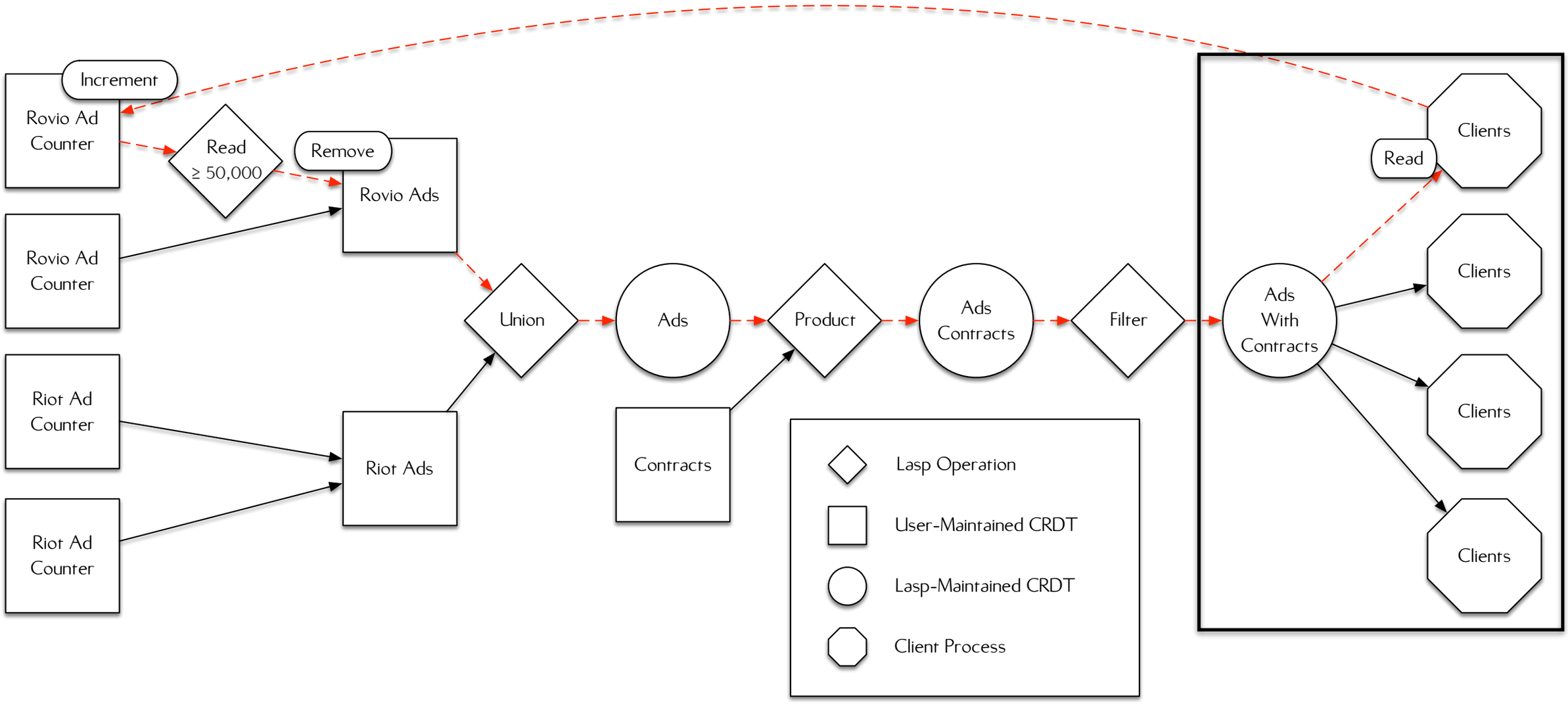

Increment
Rovio Ad Counter
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process

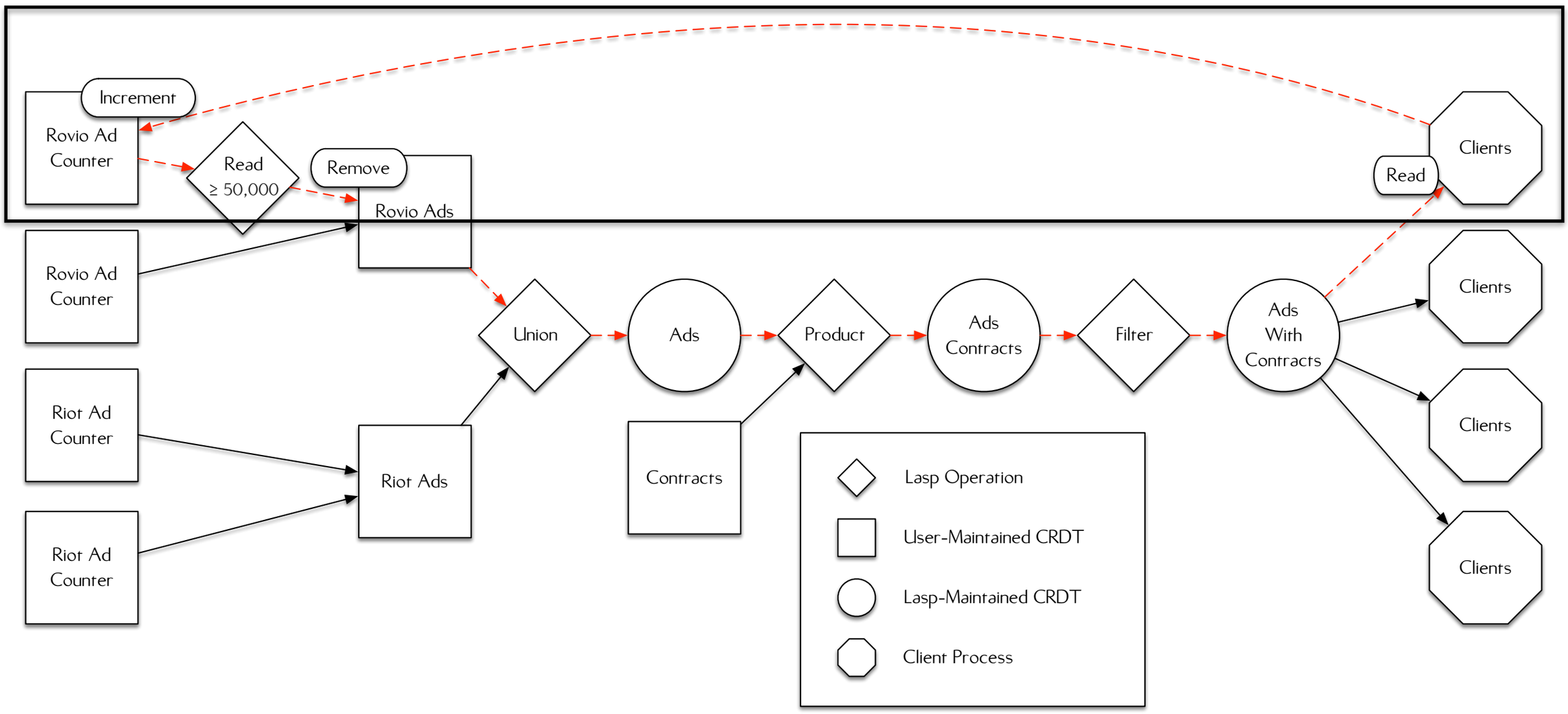

Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process

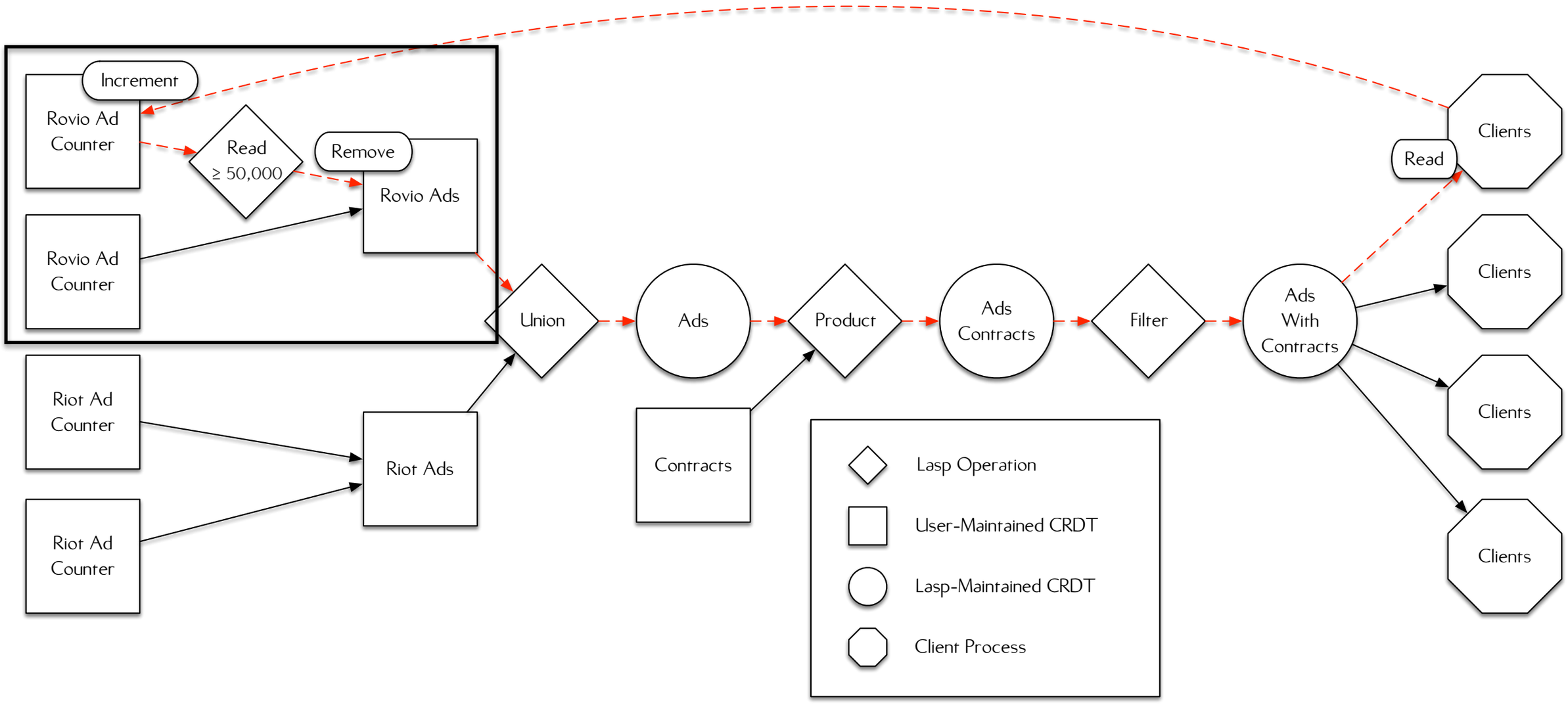

Increment
Rovio Ad Counter
Read
≥ 50,000
Rovio Ad Counter
Riot Ad Counter
Riot Ad Counter
Remove
Rovio Ads
Riot Ads
Union
Ads
Contracts
Product
Ads Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process

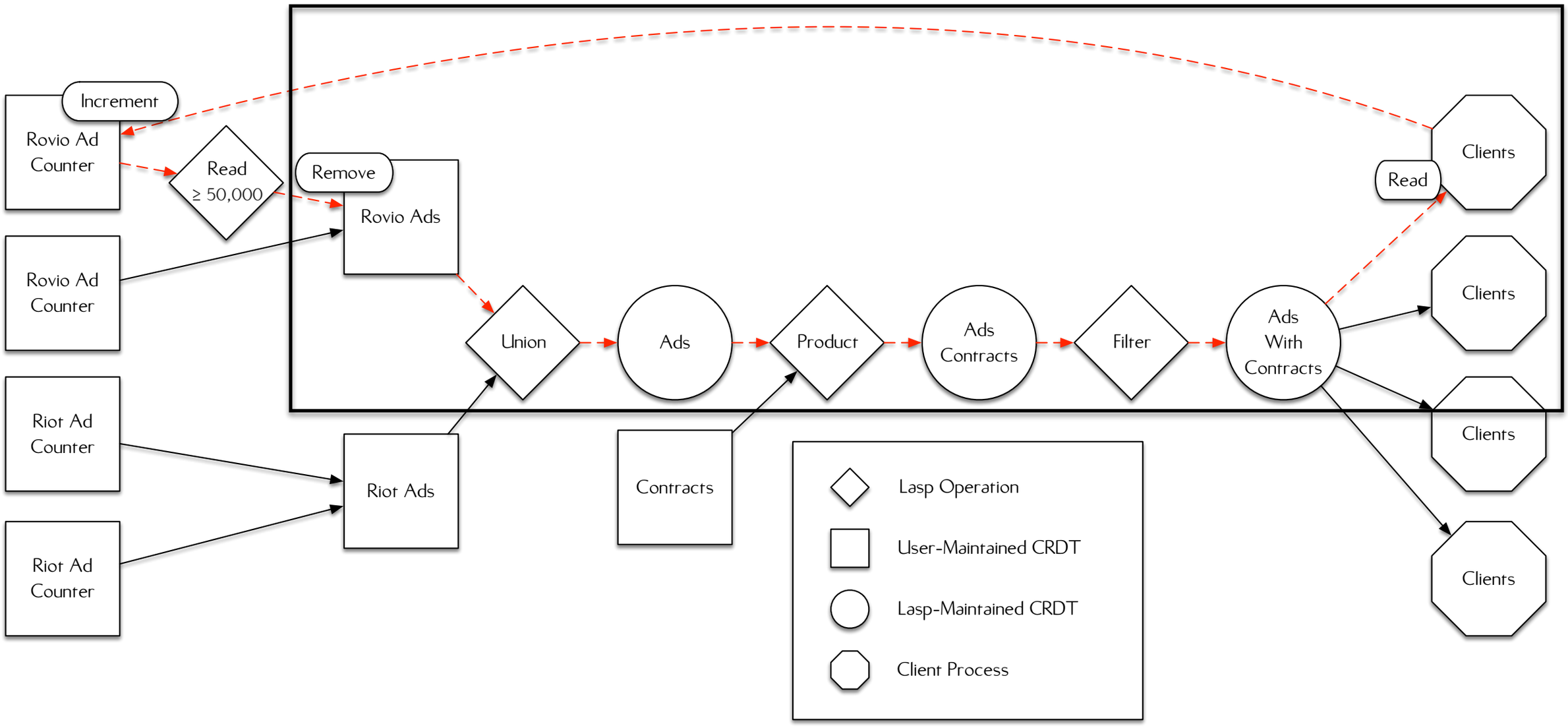

Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Ads Contracts
Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process


Increment
Rovio Ad Counter
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process


Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Ads Contracts
Filter
Contracts
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process


Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Read
Clients
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Ads Contracts
Filter
Ads With Contracts
Clients
Clients
Clients
Contracts
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process


Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process

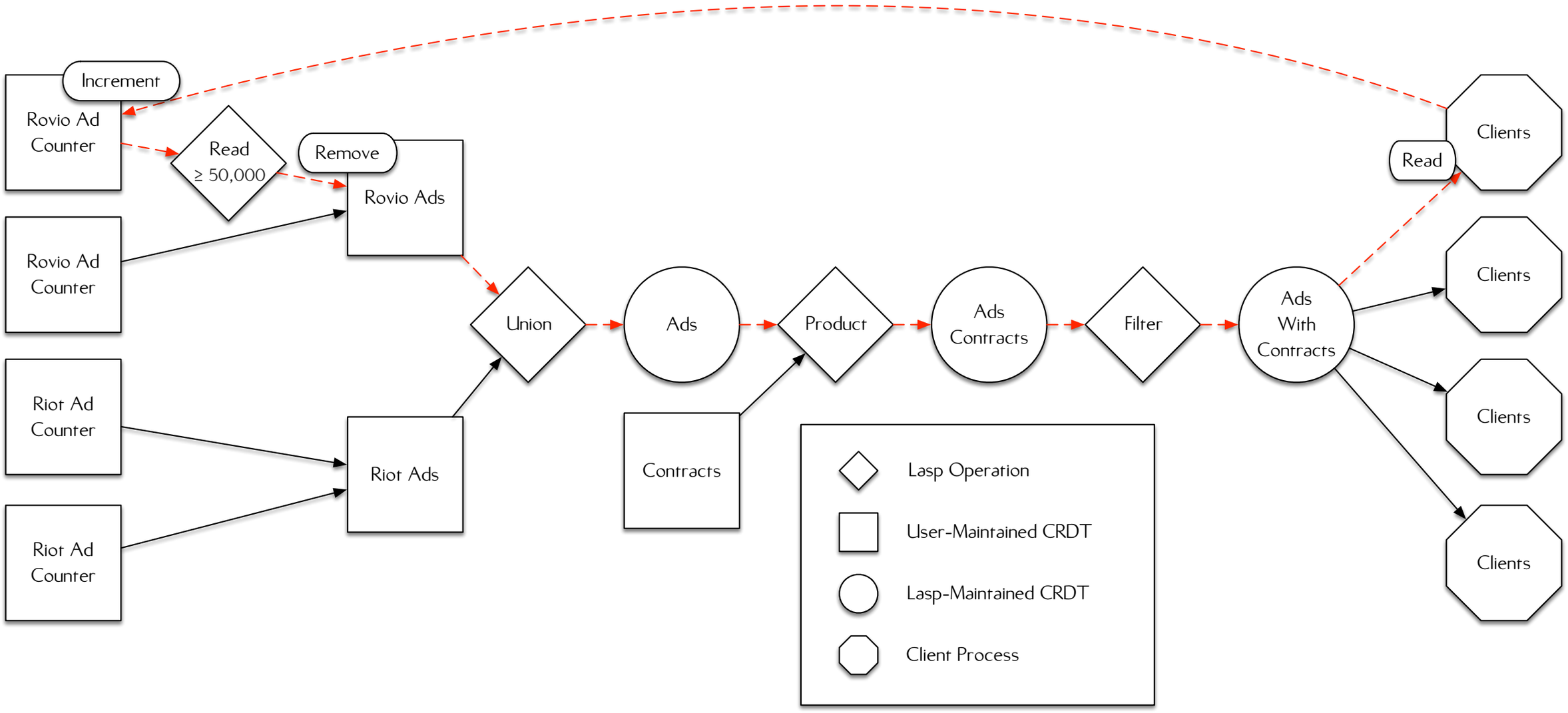

Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Ads Contracts
Filter
Ads With Contracts
Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process

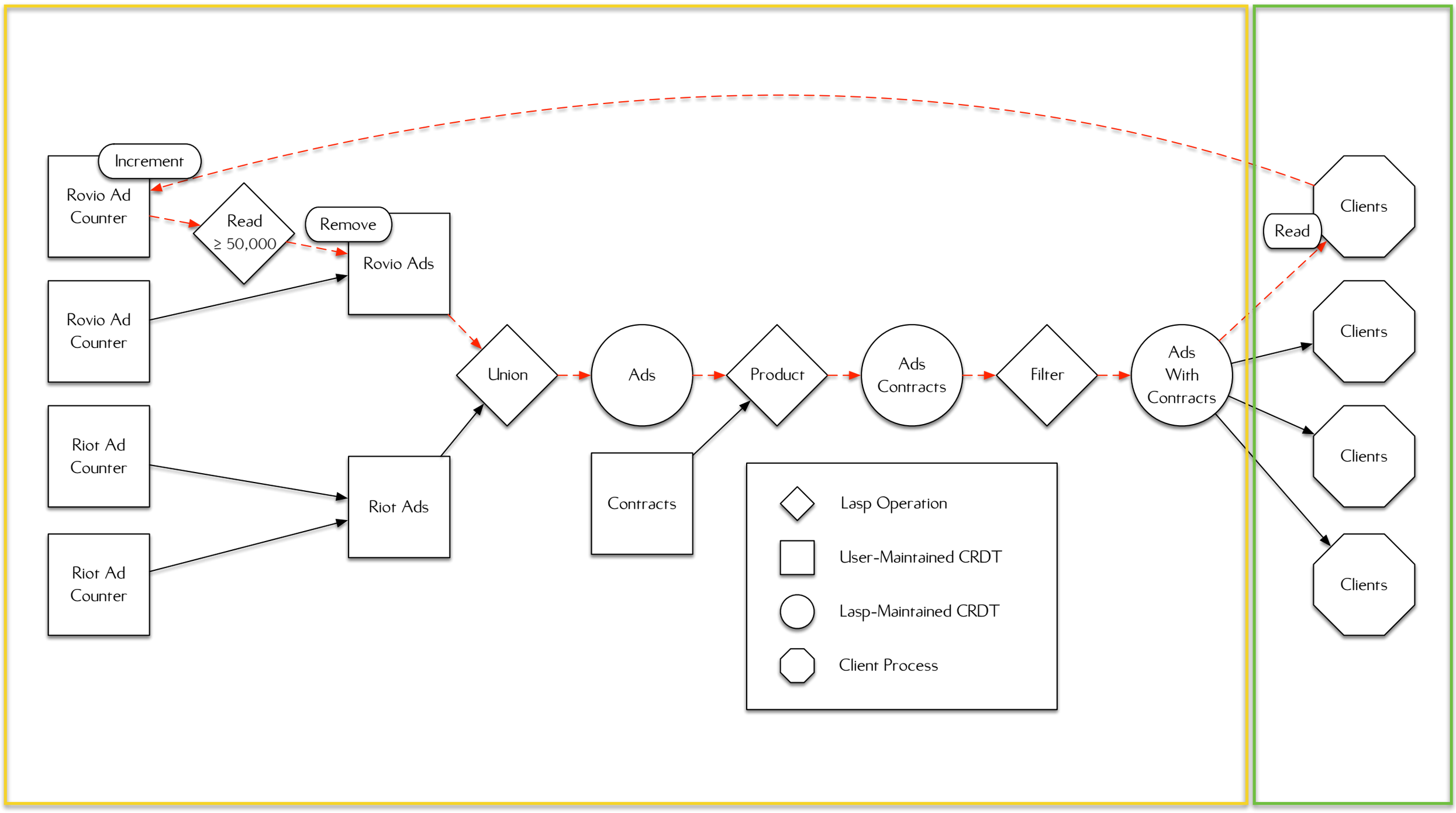
INFORMATION FLOW:

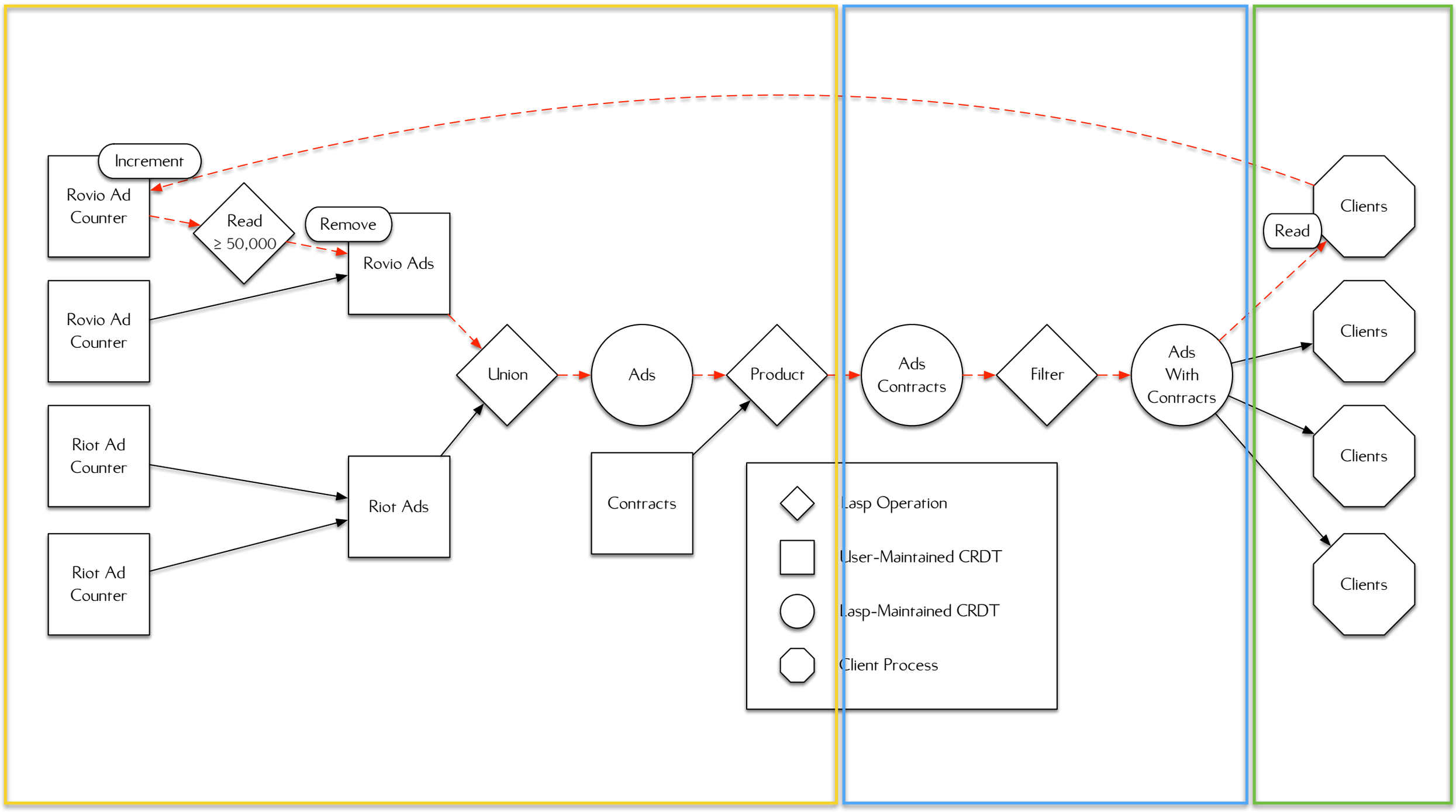
MONOTONIC
METADATA TO PREVENT DUPLICATE PROPAGATION

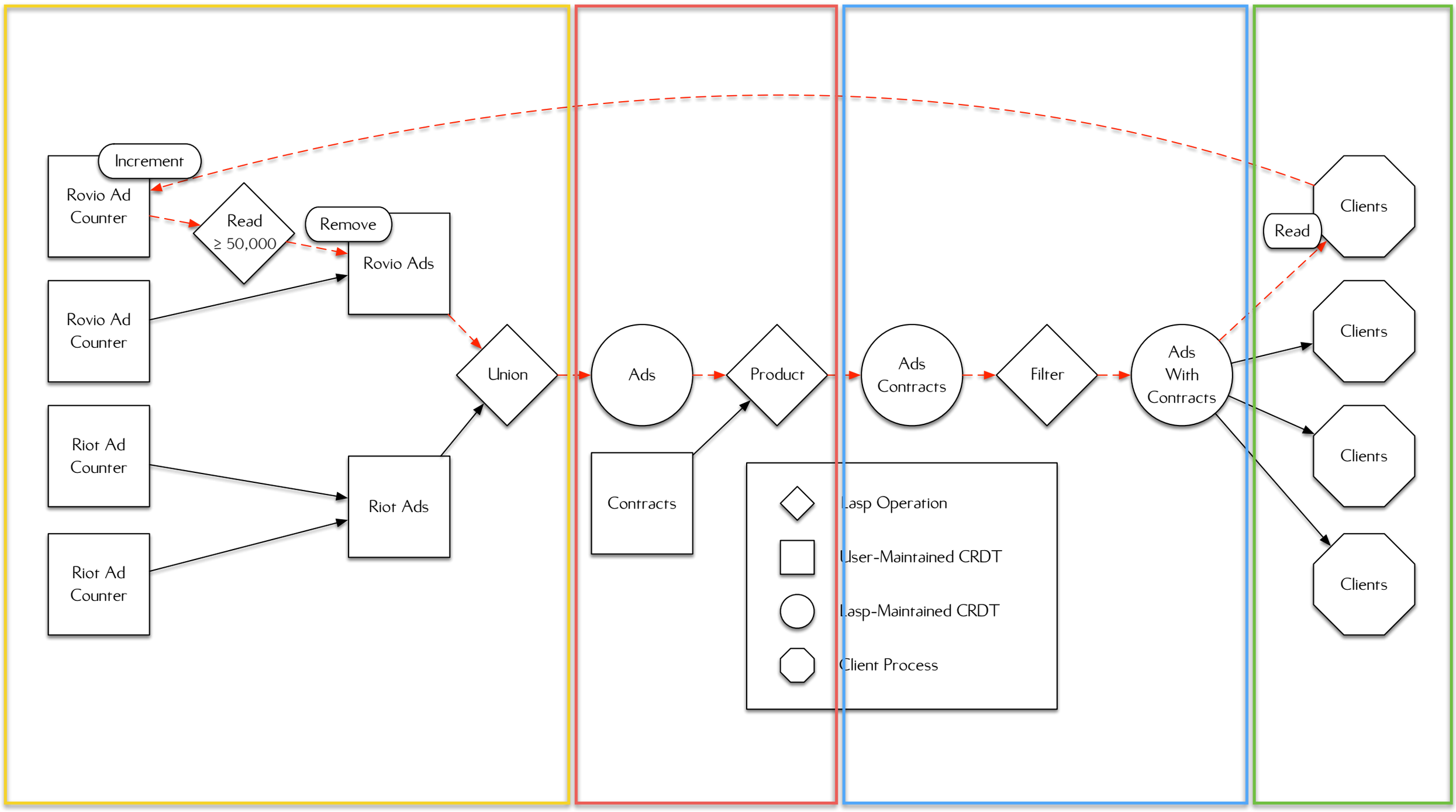
Title Text

AD COUNTER DISTRIBUTION


Increment
Rovio Ad Counter
Read ≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process


Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Ads Contracts
Filter
Ads With Contracts
Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process


Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process


Increment
Rovio Ad Counter
Read
≥ 50,000
Remove
Rovio Ads
Rovio Ad Counter
Riot Ad Counter
Riot Ads
Riot Ad Counter
Union
Ads
Product
Contracts
Ads Contracts
Filter
Ads With Contracts
Read
Clients
Clients
Clients
Clients
Lasp Operation
User-Maintained CRDT
Lasp-Maintained CRDT
Client Process

DISTRIBUTION BOUNDARIES:

ARBITRARY

ALLOWS COMPOSITION OF ENTIRE SYSTEM

AD COUNTER EXAMPLE CODE

```
%% @doc Client process; standard recursive looping server.
client(Id, AdsWithContracts, PreviousValue) ->
    receive
        view_ad ->
            %% Get current ad list.
            {ok, {_, _, AdList0}} = lasp:read(AdsWithContracts, PreviousValue),
            AdList = riak_dt_orset:value(AdList0),

            case length(AdList) of
                0 ->
                    %% No advertisements left to display; ignore
                    %% message.
                    client(Id, AdsWithContracts, AdList0);
                _ ->
                    %% Select a random advertisement from the list of
                    %% active advertisements.
                    {#ad{counter=Ad}, _} = lists:nth(
                            random:uniform(length(AdList)), AdList),

                    %% Increment it.
                    {ok, _} = lasp:update(Ad, increment, Id),
                    lager:info("Incremented ad counter: ~p", [Ad]),

                    client(Id, AdsWithContracts, AdList0)
            end
    end.
```

```
%% @doc Server functions for the advertisement counter.  After 5 views,
%%      disable the advertisement.
%%
server({#ad{counter=Counter}=Ad, _}, Ads) ->
    %% Blocking threshold read for 5 advertisement impressions.
    {ok, _} = lasp:read(Counter, 5),

    %% Remove the advertisement.
    {ok, _} = lasp:update(Ads, {remove, Ad}, Ad),

    lager:info("Removing ad: ~p", [Ad]).
```

```
%% Generate a series of unique identifiers.
RovioAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Rovio Ad Identifiers are: ~p", [RovioAdIds]),

TriforkAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Trifork Ad Identifiers are: ~p", [TriforkAdIds]),

Ids = RovioAdIds ++ TriforkAdIds,
lager:info("Ad Identifiers are: ~p", [Ids]),

%% Generate Rovio's advertisements.
{ok, RovioAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(RovioAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, RovioAdIds),

%% Generate Trifork's advertisements.
{ok, TriforkAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(TriforkAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, TriforkAdIds),

%% Union ads.
{ok, Ads} = lasp:declare(?SET),
ok = lasp:union(RovioAds, TriforkAds, Ads),

%% For each identifier, generate a contract.
{ok, Contracts} = lasp:declare(?SET),
lists:map(fun(Id) ->
        {ok, _} = lasp:update(Contracts,
                              {add, #contract{id=Id}},
                              undefined)
        end, Ids),

%% Compute the Cartesian product of both ads and contracts.
{ok, AdsContracts} = lasp:declare(?SET),
ok = lasp:product(Ads, Contracts, AdsContracts),

%% Filter items by join on item it.
{ok, AdsWithContracts} = lasp:declare(?SET),
FilterFun = fun({#ad{id=Id1}, #contract{id=Id2}}) ->
    Id1 =:= Id2
end,
ok = lasp:filter(AdsContracts, FilterFun, AdsWithContracts),

%% Launch a series of client processes, each of which is responsible
%% for displaying a particular advertisement.

%% Generate a OR-set for tracking clients.
{ok, Clients} = lasp:declare(?SET),

%% Each client takes the full list of ads when it starts, and reads
%% from the variable store.
lists:map(fun(Id) ->
        ClientPid = spawn_link(?MODULE, client,
                               [Id, AdsWithContracts, undefined]),
        {ok, _} = lasp:update(Clients,
                              {add, ClientPid},
                              undefined)
        end, lists:seq(1,5)),

%% Launch a server process for each advertisement, which will block
%% until the advertisement should be disabled.

%% Create a OR-set for the server list.
{ok, Servers} = lasp:declare(?SET),

%% Get the current advertisement list.
{ok, {_, _, AdList0}} = lasp:read(AdsWithContracts),
AdList = riak_dt_orset:value(AdList0),

%% For each advertisement, launch one server for tracking it's
%% impressions and wait to disable.
lists:map(fun(Ad) ->
        ServerPid = spawn_link(?MODULE, server, [Ad, Ads]),
        {ok, _} = lasp:update(Servers,
                              {add, ServerPid},
                              undefined)
        end, AdList),
```

```
%% Generate a series of unique identifiers.
RovioAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Rovio Ad Identifiers are: ~p", [RovioAdIds]),

TriforkAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Trifork Ad Identifiers are: ~p", [TriforkAdIds]),

Ids = RovioAdIds ++ TriforkAdIds,
lager:info("Ad Identifiers are: ~p", [Ids]),

%% Generate Rovio's advertisements.
{ok, RovioAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(RovioAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, RovioAdIds),

%% Generate Trifork's advertisements.
{ok, TriforkAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(TriforkAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, TriforkAdIds),

%% Union ads.
{ok, Ads} = lasp:declare(?SET),
ok = lasp:union(RovioAds, TriforkAds, Ads),

%% For each identifier, generate a contract.
{ok, Contracts} = lasp:declare(?SET),
lists:map(fun(Id) ->
        {ok, _} = lasp:update(Contracts,
                              {add, #contract{id=Id}},
                              undefined)
        end, Ids),
```

```
%% Compute the Cartesian product of both ads and contracts.
{ok, AdsContracts} = lasp:declare(?SET),
ok = lasp:product(Ads, Contracts, AdsContracts),

%% Filter items by join on item it.
{ok, AdsWithContracts} = lasp:declare(?SET),
FilterFun = fun({#ad{id=Id1}, #contract{id=Id2}}) ->
    Id1 =:= Id2
end,
ok = lasp:filter(AdsContracts, FilterFun, AdsWithContracts),

%% Launch a series of client processes, each of which is responsible
%% for displaying a particular advertisement.

%% Generate a OR-set for tracking clients.
{ok, Clients} = lasp:declare(?SET),

%% Each client takes the full list of ads when it starts, and reads
%% from the variable store.
lists:map(fun(Id) ->
        ClientPid = spawn_link(?MODULE, client,
                               [Id, AdsWithContracts, undefined]),
        {ok, _} = lasp:update(Clients,
                              {add, ClientPid},
                              undefined)
        end, lists:seq(1,5)),

%% Launch a server process for each advertisement, which will block
%% until the advertisement should be disabled.

%% Create a OR-set for the server list.
{ok, Servers} = lasp:declare(?SET),

%% Get the current advertisement list.
{ok, {_, _, AdList0}} = lasp:read(AdsWithContracts),
AdList = riak_dt_orset:value(AdList0),

%% For each advertisement, launch one server for tracking it's
%% impressions and wait to disable.
lists:map(fun(Ad) ->
        ServerPid = spawn_link(?MODULE, server, [Ad, Ads]),
        {ok, _} = lasp:update(Servers,
                              {add, ServerPid},
                              undefined)
        end, AdList),
```

```
%% Generate a series of unique identifiers.
RovioAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Rovio Ad Identifiers are: ~p", [RovioAdIds]),

TriforkAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Trifork Ad Identifiers are: ~p", [TriforkAdIds]),

Ids = RovioAdIds ++ TriforkAdIds,
lager:info("Ad Identifiers are: ~p", [Ids]),

%% Generate Rovio's advertisements.
{ok, RovioAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(RovioAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, RovioAdIds),

%% Generate Trifork's advertisements.
{ok, TriforkAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(TriforkAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, TriforkAdIds),

%% Union ads.
{ok, Ads} = lasp:declare(?SET),
ok = lasp:union(RovioAds, TriforkAds, Ads),

%% For each identifier, generate a contract.
{ok, Contracts} = lasp:declare(?SET),
lists:map(fun(Id) ->
        {ok, _} = lasp:update(Contracts,
                              {add, #contract{id=Id}},
                              undefined)
        end, Ids),
```

```
%% Compute the Cartesian product of both ads and contracts.
{ok, AdsContracts} = lasp:declare(?SET),
ok = lasp:product(Ads, Contracts, AdsContracts),

%% Filter items by join on item it.
{ok, AdsWithContracts} = lasp:declare(?SET),
FilterFun = fun({#ad{id=Id1}, #contract{id=Id2}}) ->
    Id1 =:= Id2
end,
ok = lasp:filter(AdsContracts, FilterFun, AdsWithContracts),

%% Launch a series of client processes, each of which is responsible
%% for displaying a particular advertisement.

%% Generate a OR-set for tracking clients.
{ok, Clients} = lasp:declare(?SET),

%% Each client takes the full list of ads when it starts, and reads
%% from the variable store.
lists:map(fun(Id) ->
        ClientPid = spawn_link(?MODULE, client,
                               [Id, AdsWithContracts, undefined]),
        {ok, _} = lasp:update(Clients,
                              {add, ClientPid},
                              undefined)
        end, lists:seq(1,5)),

%% Launch a server process for each advertisement, which will block
%% until the advertisement should be disabled.

%% Create a OR-set for the server list.
{ok, Servers} = lasp:declare(?SET),

%% Get the current advertisement list.
{ok, {_, _, AdList0}} = lasp:read(AdsWithContracts),
AdList = riak_dt_orset:value(AdList0),

%% For each advertisement, launch one server for tracking it's
%% impressions and wait to disable.
lists:map(fun(Ad) ->
        ServerPid = spawn_link(?MODULE, server, [Ad, Ads]),
        {ok, _} = lasp:update(Servers,
                              {add, ServerPid},
                              undefined)
        end, AdList),
```

```
%% Generate a series of unique identifiers.
RovioAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Rovio Ad Identifiers are: ~p", [RovioAdIds]),

TriforkAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Trifork Ad Identifiers are: ~p", [TriforkAdIds]),

Ids = RovioAdIds ++ TriforkAdIds,
lager:info("Ad Identifiers are: ~p", [Ids]),

%% Generate Rovio's advertisements.
{ok, RovioAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(RovioAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, RovioAdIds),

%% Generate Trifork's advertisements.
{ok, TriforkAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(TriforkAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, TriforkAdIds),

%% Union ads.
{ok, Ads} = lasp:declare(?SET),
ok = lasp:union(RovioAds, TriforkAds, Ads),

%% For each identifier, generate a contract.
{ok, Contracts} = lasp:declare(?SET),
lists:map(fun(Id) ->
        {ok, _} = lasp:update(Contracts,
                              {add, #contract{id=Id}},
                              undefined)
        end, Ids),

%% Compute the Cartesian product of both ads and contracts.
{ok, AdsContracts} = lasp:declare(?SET),
ok = lasp:product(Ads, Contracts, AdsContracts),

%% Filter items by join on item it.
{ok, AdsWithContracts} = lasp:declare(?SET),
FilterFun = fun({#ad{id=Id1}, #contract{id=Id2}}) ->
    Id1 =:= Id2
end,
ok = lasp:filter(AdsContracts, FilterFun, AdsWithContracts),

%% Launch a series of client processes, each of which is responsible
%% for displaying a particular advertisement.

%% Generate a OR-set for tracking clients.
{ok, Clients} = lasp:declare(?SET),

%% Each client takes the full list of ads when it starts, and reads
%% from the variable store.
lists:map(fun(Id) ->
        ClientPid = spawn_link(?MODULE, client,
                               [Id, AdsWithContracts, undefined]),
        {ok, _} = lasp:update(Clients,
                              {add, ClientPid},
                              undefined)
        end, lists:seq(1,5)),

%% Launch a server process for each advertisement, which will block
%% until the advertisement should be disabled.

%% Create a OR-set for the server list.
{ok, Servers} = lasp:declare(?SET),

%% Get the current advertisement list.
{ok, {_, _, AdList0}} = lasp:read(AdsWithContracts),
AdList = riak_dt_orset:value(AdList0),

%% For each advertisement, launch one server for tracking it's
%% impressions and wait to disable.
lists:map(fun(Ad) ->
        ServerPid = spawn_link(?MODULE, server, [Ad, Ads]),
        {ok, _} = lasp:update(Servers,
                              {add, ServerPid},
                              undefined)
        end, AdList),
```

```
%% Generate a series of unique identifiers.
RovioAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Rovio Ad Identifiers are: ~p", [RovioAdIds]),

TriforkAdIds = lists:map(fun(_) -> druuid:v4() end, lists:seq(1, 10)),
lager:info("Trifork Ad Identifiers are: ~p", [TriforkAdIds]),

Ids = RovioAdIds ++ TriforkAdIds,
lager:info("Ad Identifiers are: ~p", [Ids]),

%% Generate Rovio's advertisements.
{ok, RovioAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(RovioAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, RovioAdIds),

%% Generate Trifork's advertisements.
{ok, TriforkAds} = lasp:declare(?SET),
lists:map(fun(Id) ->
        %% Generate a G-Counter.
        {ok, CounterId} = lasp:declare(?COUNTER),
        %% Add it to the advertisement set.
        {ok, _} = lasp:update(TriforkAds,
                              {add, #ad{id=Id, counter=CounterId}},
                              undefined)

        end, TriforkAdIds),

%% Union ads.
{ok, Ads} = lasp:declare(?SET),
ok = lasp:union(RovioAds, TriforkAds, Ads),

%% For each identifier, generate a contract.
{ok, Contracts} = lasp:declare(?SET),
lists:map(fun(Id) ->
        {ok, _} = lasp:update(Contracts,
                              {add, #contract{id=Id}},
                              undefined)
        end, Ids),

%% Compute the Cartesian product of both ads and contracts.
{ok, AdsContracts} = lasp:declare(?SET),
ok = lasp:product(Ads, Contracts, AdsContracts),

%% Filter items by join on item it.
{ok, AdsWithContracts} = lasp:declare(?SET),
FilterFun = fun({#ad{id=Id1}, #contract{id=Id2}}) ->
    Id1 =:= Id2
end,
ok = lasp:filter(AdsContracts, FilterFun, AdsWithContracts),

%% Launch a series of client processes, each of which is responsible
%% for displaying a particular advertisement.

%% Generate a OR-set for tracking clients.
{ok, Clients} = lasp:declare(?SET),

%% Each client takes the full list of ads when it starts, and reads
%% from the variable store.
lists:map(fun(Id) ->
        ClientPid = spawn_link(?MODULE, client,
                               [Id, AdsWithContracts, undefined]),
        {ok, _} = lasp:update(Clients,
                              {add, ClientPid},
                              undefined)
        end, lists:seq(1,5)),

%% Launch a server process for each advertisement, which will block
%% until the advertisement should be disabled.

%% Create a OR-set for the server list.
{ok, Servers} = lasp:declare(?SET),

%% Get the current advertisement list.
{ok, {_, _, AdList0}} = lasp:read(AdsWithContracts),
AdList = riak_dt_orset:value(AdList0),

%% For each advertisement, launch one server for tracking it's
%% impressions and wait to disable.
lists:map(fun(Ad) ->
        ServerPid = spawn_link(?MODULE, server, [Ad, Ads]),
        {ok, _} = lasp:update(Servers,
                              {add, ServerPid},
                              undefined)
        end, AdList),
```

RELATED WORK

RELATED WORK

DISTRIBUTED OZ

RELATED WORK

DERFLOW$_L$

RELATED WORK

BLOOM[L]

RELATED WORK

LVARS

RELATED WORK

D-STREAMS

RELATED WORK

SUMMINGBIRD

FUTURE WORK

FUTURE WORK

INVARIANT PRESERVATION

FUTURE WORK

CAUSAL+ CONSISTENCY

FUTURE WORK

ORSWOT OPTIMIZATION

FUTURE WORK

DELTA STATE-CRDTs

FUTURE WORK

OPERATION-BASED CRDTs

FUTURE WORK

DEFORESTATION

SOURCE

GITHUB.COM/CMEIKLEJOHN/LASP

ERLANG WORKSHOP 2014

DERFLOW
DISTRIBUTED DETERMINISTIC DATAFLOW PROGRAMMING FOR ERLANG

PAPOC / EUROSYS 2015

LASP
A LANGUAGE FOR DISTRIBUTED, EVENTUALLY CONSISTENT COMPUTATIONS WITH CRDTs

SYNCFREE

SYNCFREE IS A EUROPEAN RESEARCH PROJECT TAKING PLACE FOR 3 YEARS, STARING OCTOBER 2013, AND IS FUNDED BY THE EUROPEAN UNION, GRANT AGREEMENT N°609551.

FIN