Dynamoth: A Scalable Pub/Sub Middleware for Latency-Constrained Applications in the Cloud

Julien Gascon-Samson, Franz-Philippe Garcia, Bettina Kemme, Jörg Kienzle

School of Computer Science, McGill University
Montreal, Canada

Thursday July 2, 2015

Introduction and Background - What is Dynamoth?

- Channel-based pub/sub service in the Cloud

Introduction and Background - What is Dynamoth?

- Channel-based pub/sub service in the Cloud
- For any kind of application, with a specific emphasis on latency-constrained applications

Introduction and Background - What is Dynamoth?

- Channel-based pub/sub service in the Cloud
- For any kind of application, with a specific emphasis on latency-constrained applications
- Can support very large-scale applications

Introduction and Background - What is Dynamoth?

- Channel-based pub/sub service in the Cloud
- For any kind of application, with a specific emphasis on latency-constrained applications
- Can support very large-scale applications
- Can support multiple applications simultaneously

Introduction and Background - What is Dynamoth?

- Channel-based pub/sub service in the Cloud
- For any kind of application, with a specific emphasis on latency-constrained applications
- Can support very large-scale applications
- Can support multiple applications simultaneously
- Scalability / dynamism / extensive load-balancing

Introduction and Background - What is Dynamoth?

- Channel-based pub/sub service in the Cloud
- For any kind of application, with a specific emphasis on latency-constrained applications
- Can support very large-scale applications
- Can support multiple applications simultaneously
- Scalability / dynamism / extensive load-balancing
- Minimizes resource usage

Introduction and Background - What is Dynamoth?

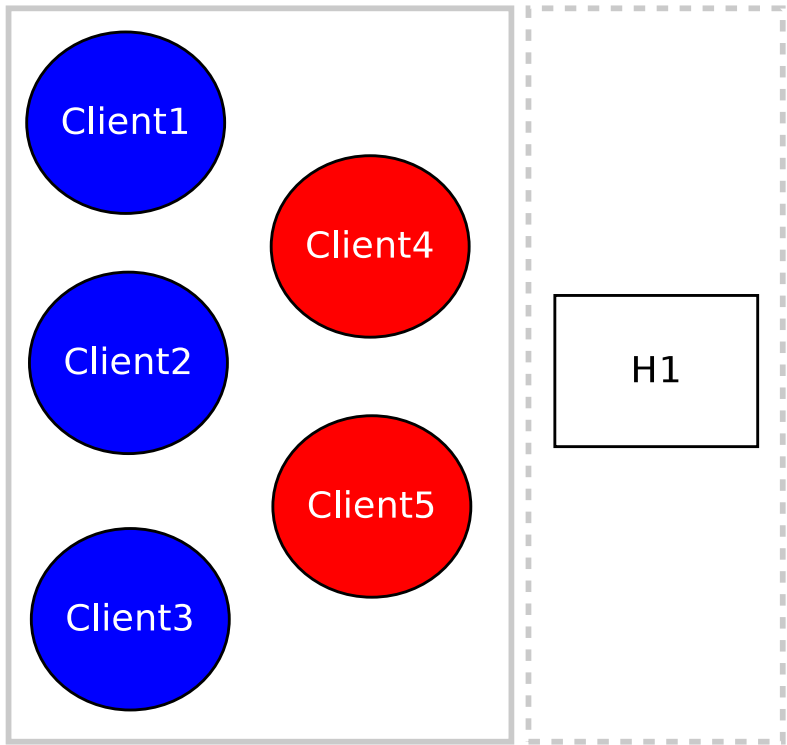
- Channel-based pub/sub service in the Cloud
- For any kind of application, with a specific emphasis on latency-constrained applications
- Can support very large-scale applications
- Can support multiple applications simultaneously
- Scalability / dynamism / extensive load-balancing
- Minimizes resource usage
- Proposes mechanisms to deal with channels that cannot be handled by only one server

Introduction and Background - What is Dynamoth?

- Channel-based pub/sub service in the Cloud
- For any kind of application, with a specific emphasis on latency-constrained applications
- Can support very large-scale applications
- Can support multiple applications simultaneously
- Scalability / dynamism / extensive load-balancing
- Minimizes resource usage
- Proposes mechanisms to deal with channels that cannot be handled by only one server
- Built on top of an unmodified single-server pub/sub middleware (Redis)

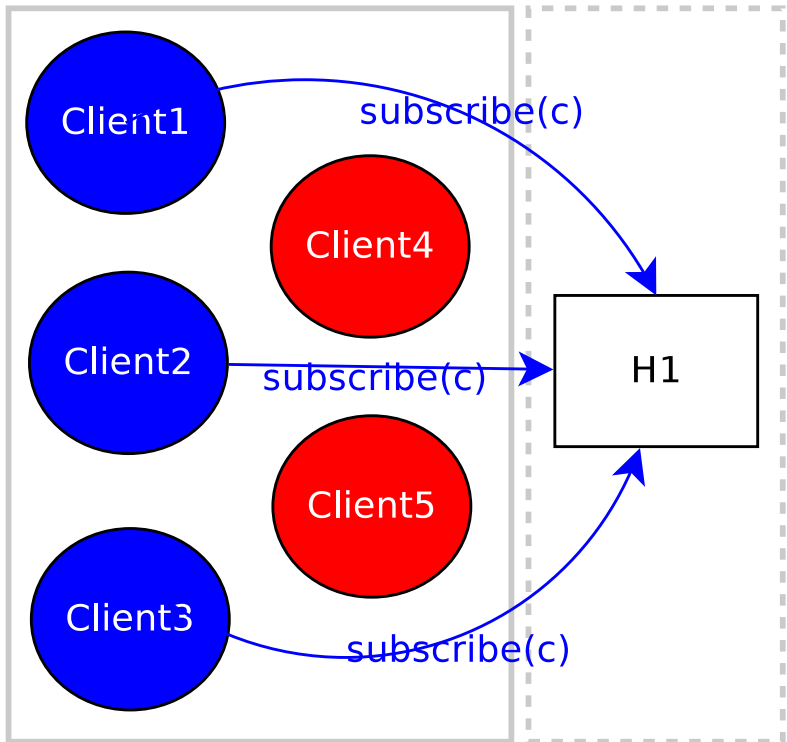
Channel-Based Pub/Sub

- **Subscribers** (in blue) subscribe to channels (topics)
- **Publishers** (in red) publish to channels
- All subscribers of a given channel c will receive all publications sent through c



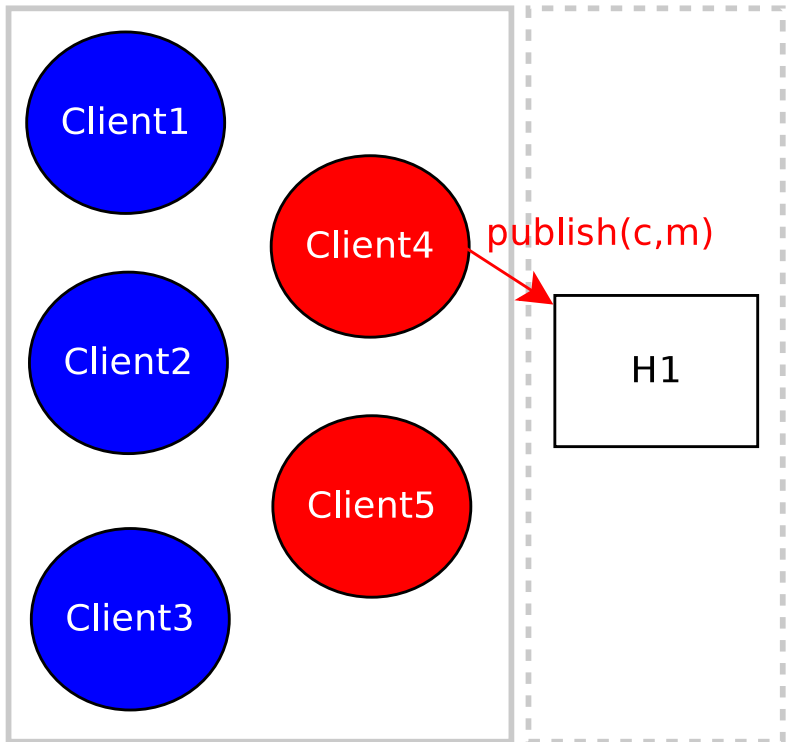
Channel-Based Pub/Sub

- **Subscribers** (in blue) subscribe to channels (topics)
- **Publishers** (in red) publish to channels
- All subscribers of a given channel *c* will receive all publications sent through *c*



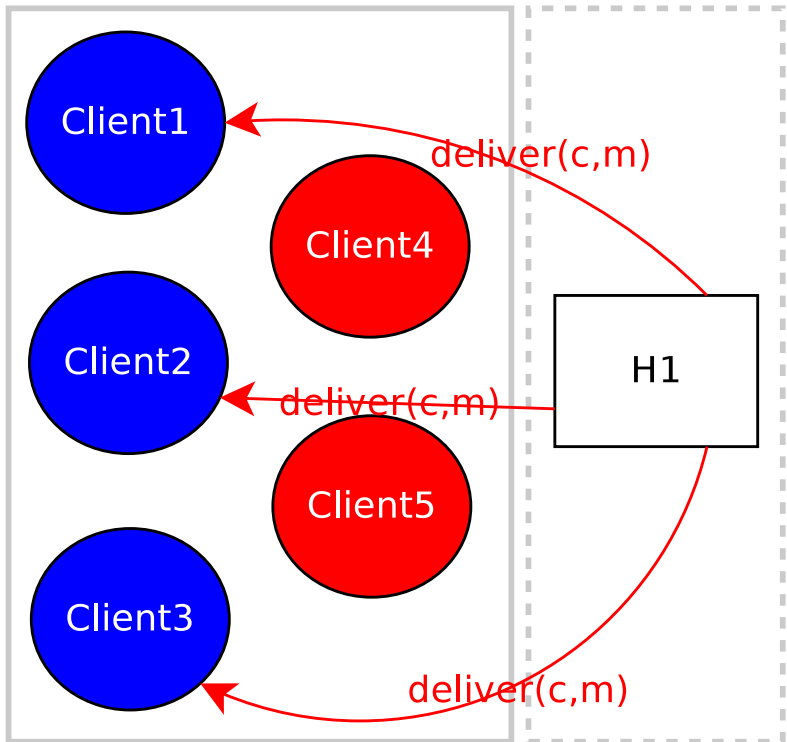
Channel-Based Pub/Sub

- **Subscribers** (in blue) subscribe to channels (topics)
- **Publishers** (in red) publish to channels
- All subscribers of a given channel *c* will receive all publications sent through *c*



Channel-Based Pub/Sub

- **Subscribers** (in blue) subscribe to channels (topics)
- **Publishers** (in red) publish to channels
- All subscribers of a given channel c will receive all publications sent through c



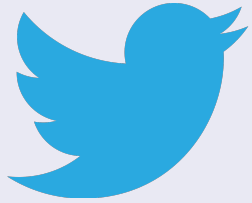
Applications of Channel-Based Pub/Sub

Traffic alert systems

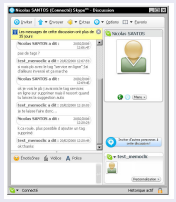
Extreme weather alert systems

Mobile device notif. frameworks

Social networks

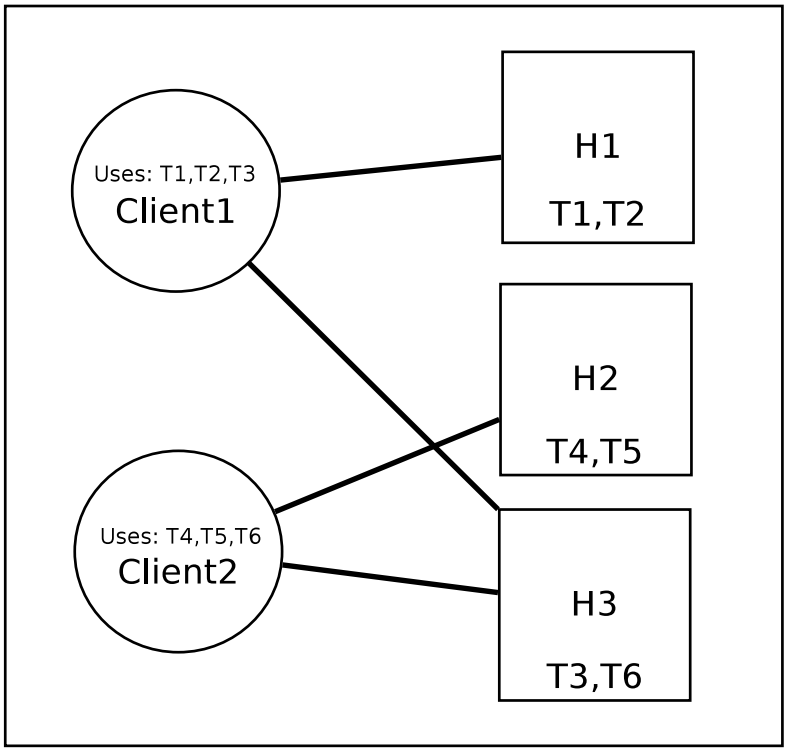


Chat/IM systems



Massive Multiplayer Online Games

Scaling: Consistent Hashing



- Each pub/sub server assigned a set of **virtual identifiers**
- Each channel maps to a virtual identifier (hashing)
- Add a new server: some of the virtual identifiers of each server get "transferred" to new server

Dynamoth - Why?

Addresses the shortcomings of consistent hashing:

1) Channels often have different load

Need a finer-grained channel-to-server approach to provide even load.

Dynamoth - Why?

Addresses the shortcomings of consistent hashing:

1) Channels often have different load

Need a finer-grained channel-to-server approach to provide even load.

2) Dynamic sizing

- Move channels between servers.
- Remove/add servers on-the-fly.
- Reconfiguration: clients (publishers/subscribers) must be aware and react to such changes.
- During reconfiguration, all messages should still be delivered.

Dynamoth - Why?

Addresses the shortcomings of consistent hashing:

1) Channels often have different load

Need a finer-grained channel-to-server approach to provide even load.

2) Dynamic sizing

- Move channels between servers.
- Remove/add servers on-the-fly.
- Reconfiguration: clients (publishers/subscribers) must be aware and react to such changes.
- During reconfiguration, all messages should still be delivered.

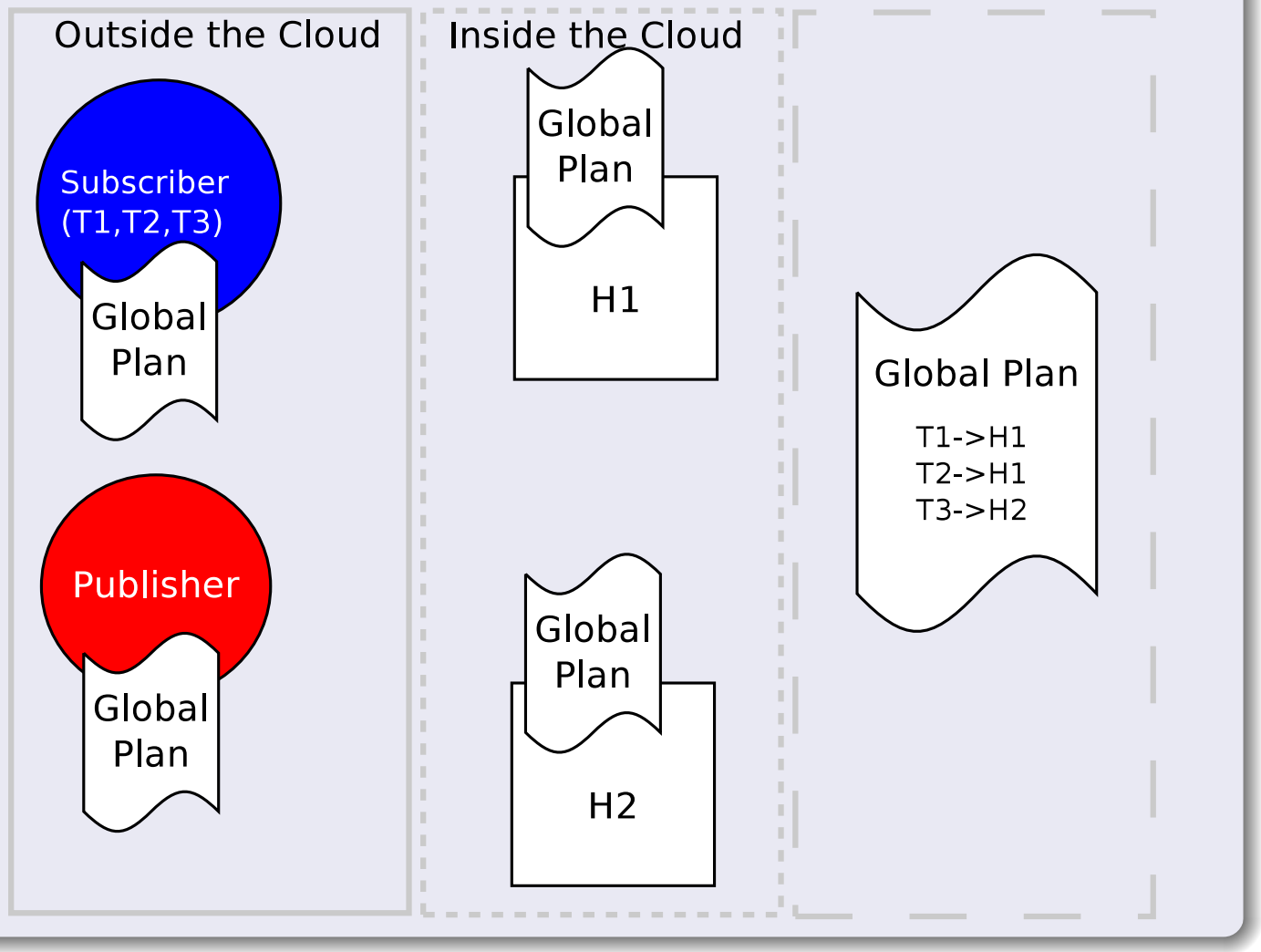
3) Even a single channel can overload a server

We propose channel replication as a way to split the load of a single channel accross multiple servers.

Dynamoth Middleware

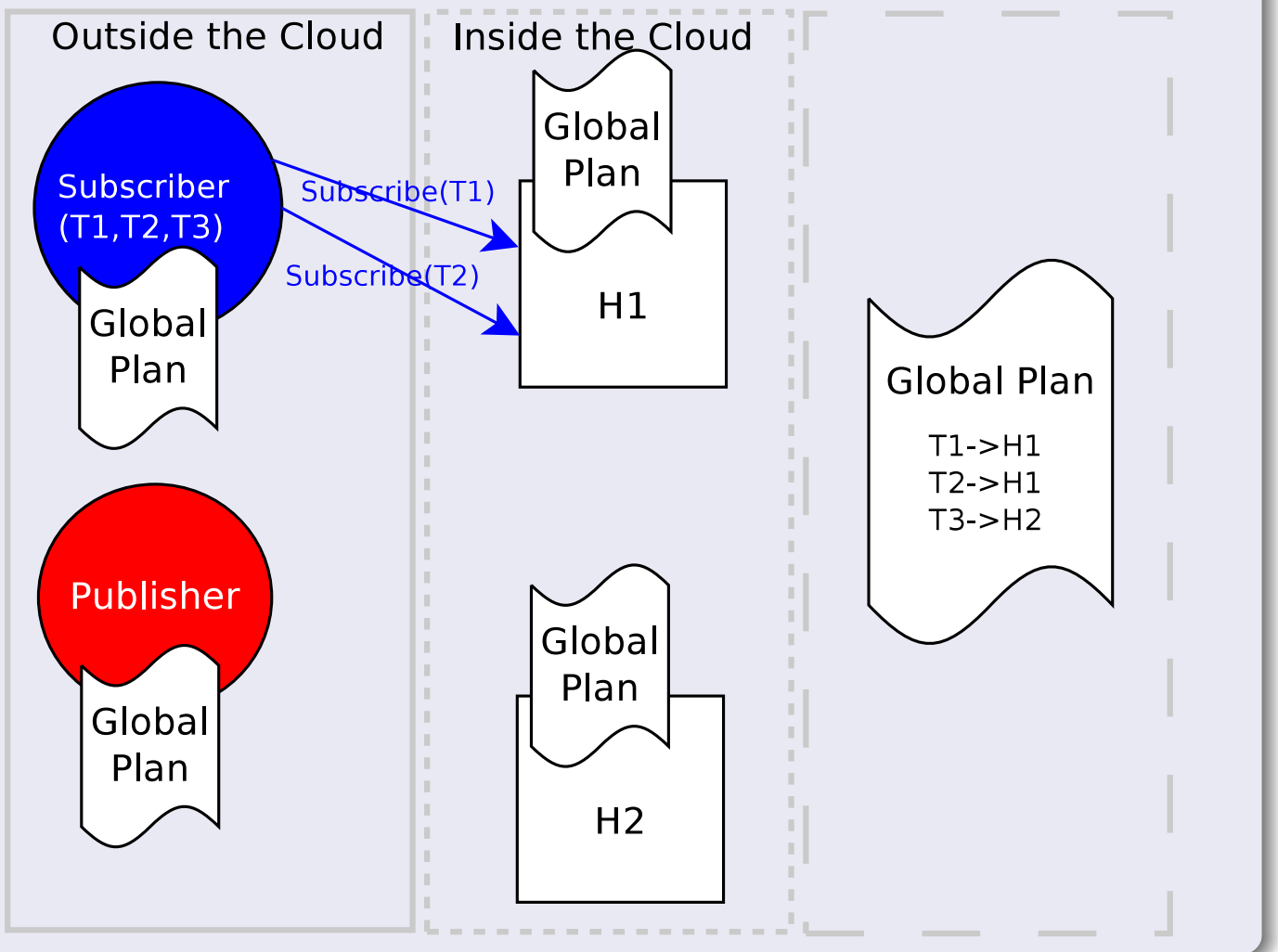
Dynamoth - Plan for Publications and Subscriptions

Initial conditions



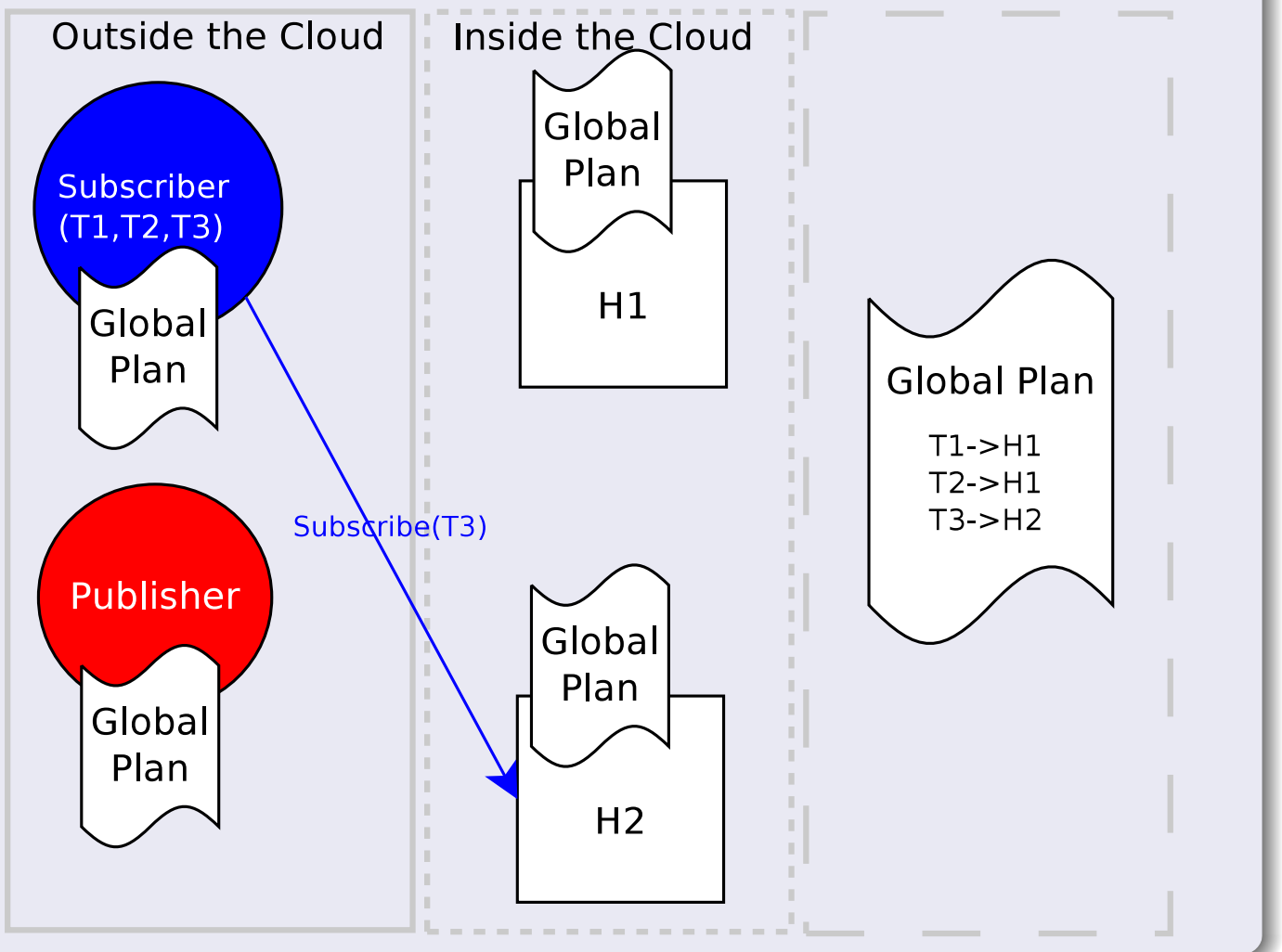
Dynamoth - Plan for Publications and Subscriptions

Subscriptions

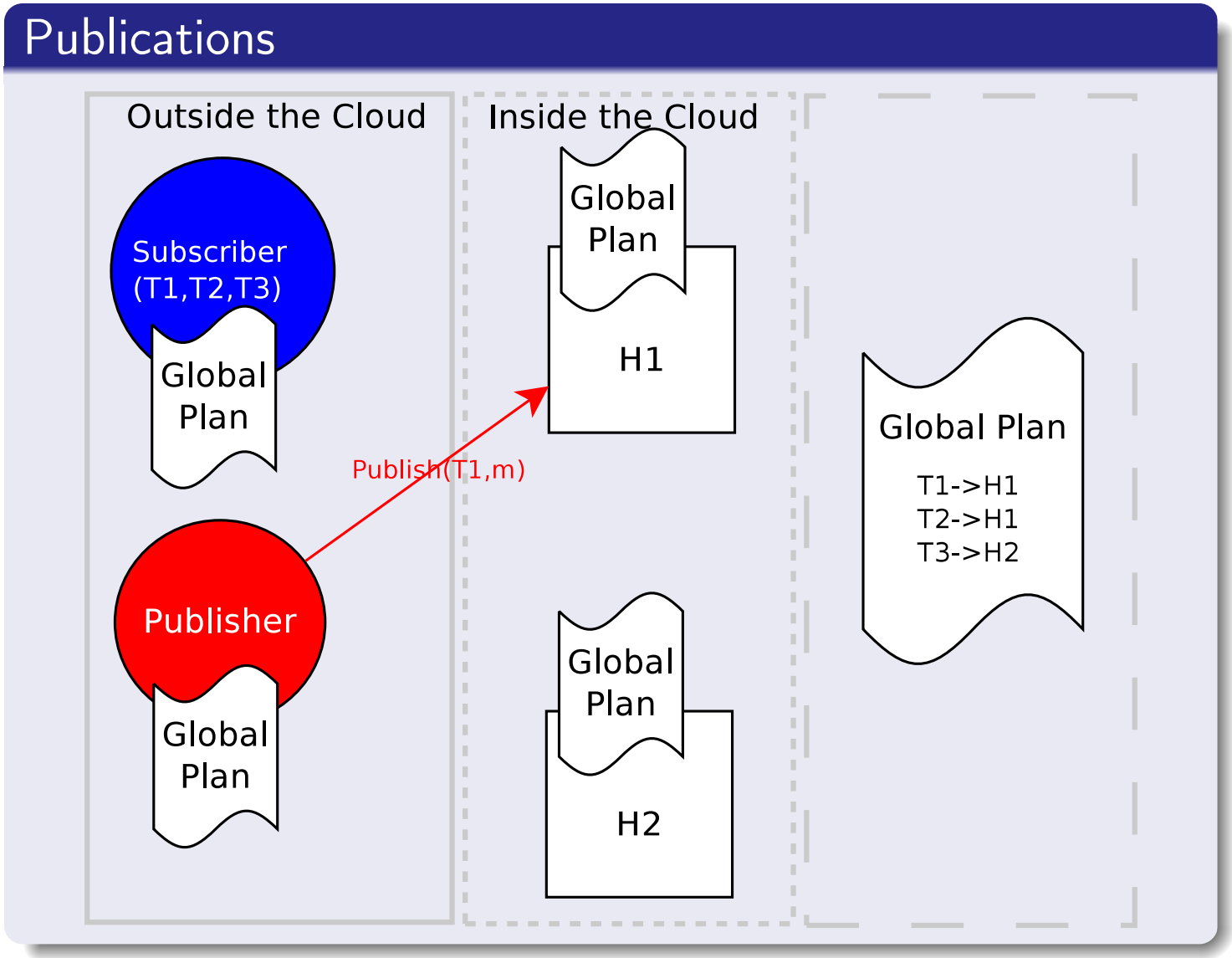


Dynamoth - Plan for Publications and Subscriptions

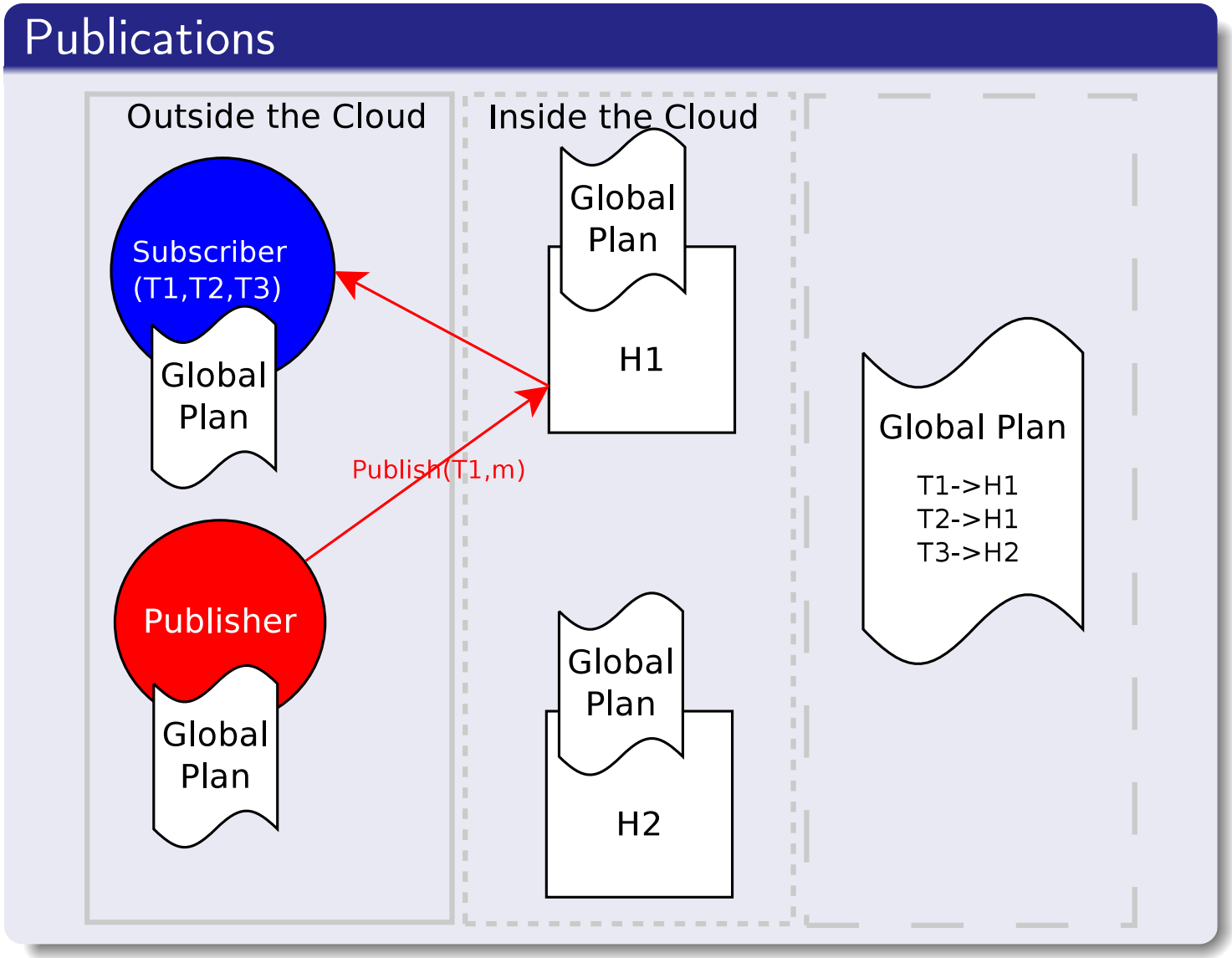
Subscriptions



Dynamoth - Plan for Publications and Subscriptions

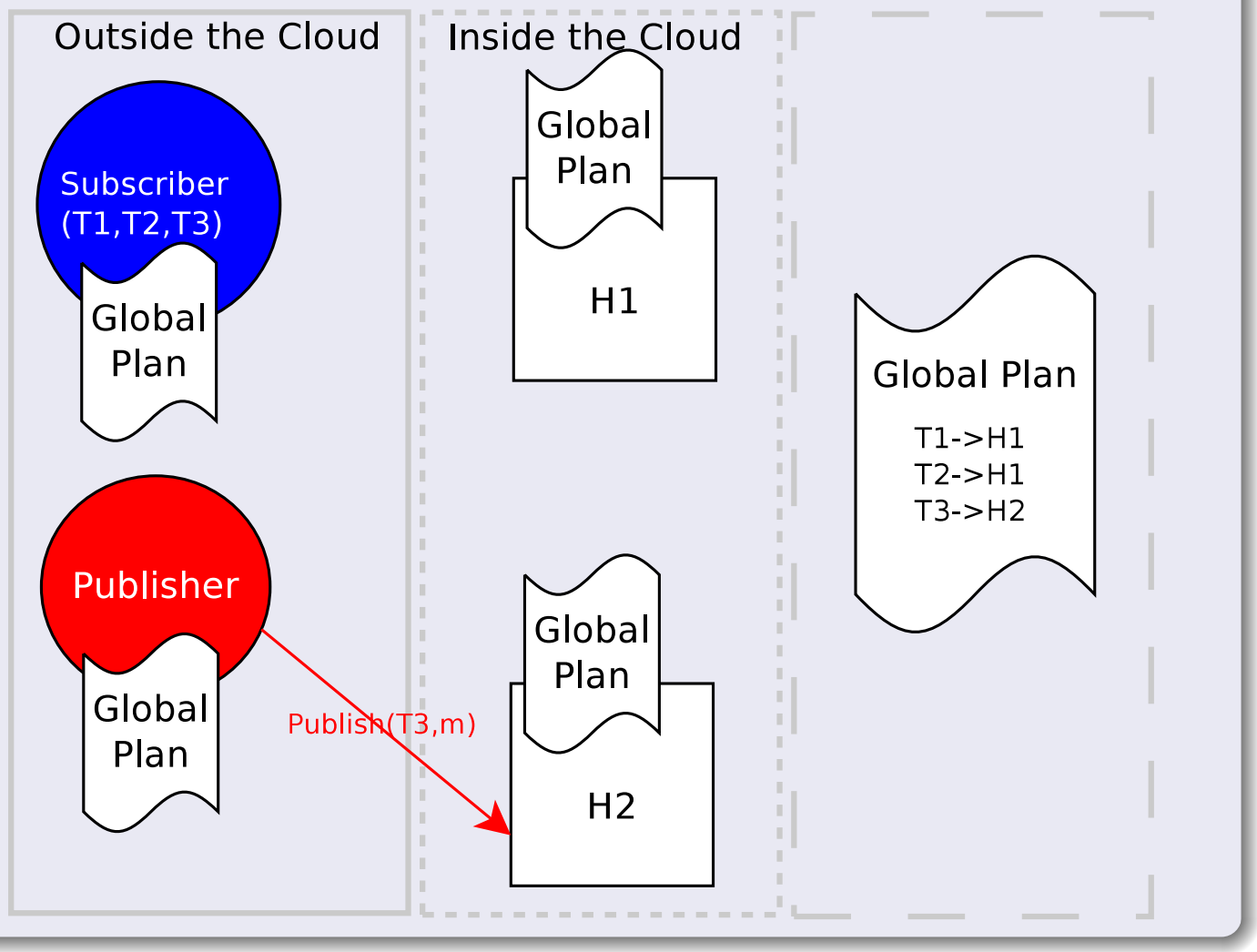


Dynamoth - Plan for Publications and Subscriptions



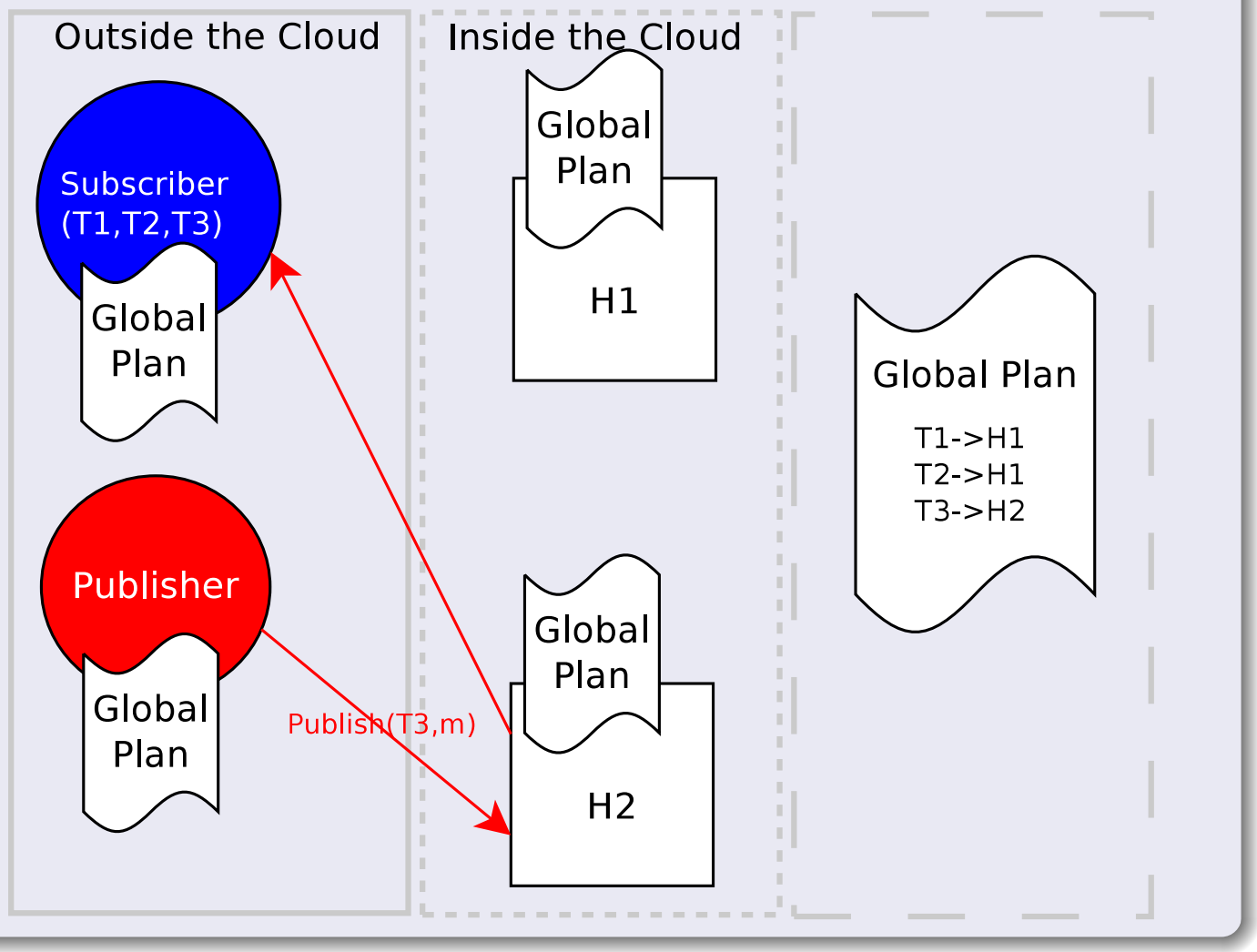
Dynamoth - Plan for Publications and Subscriptions

Publications



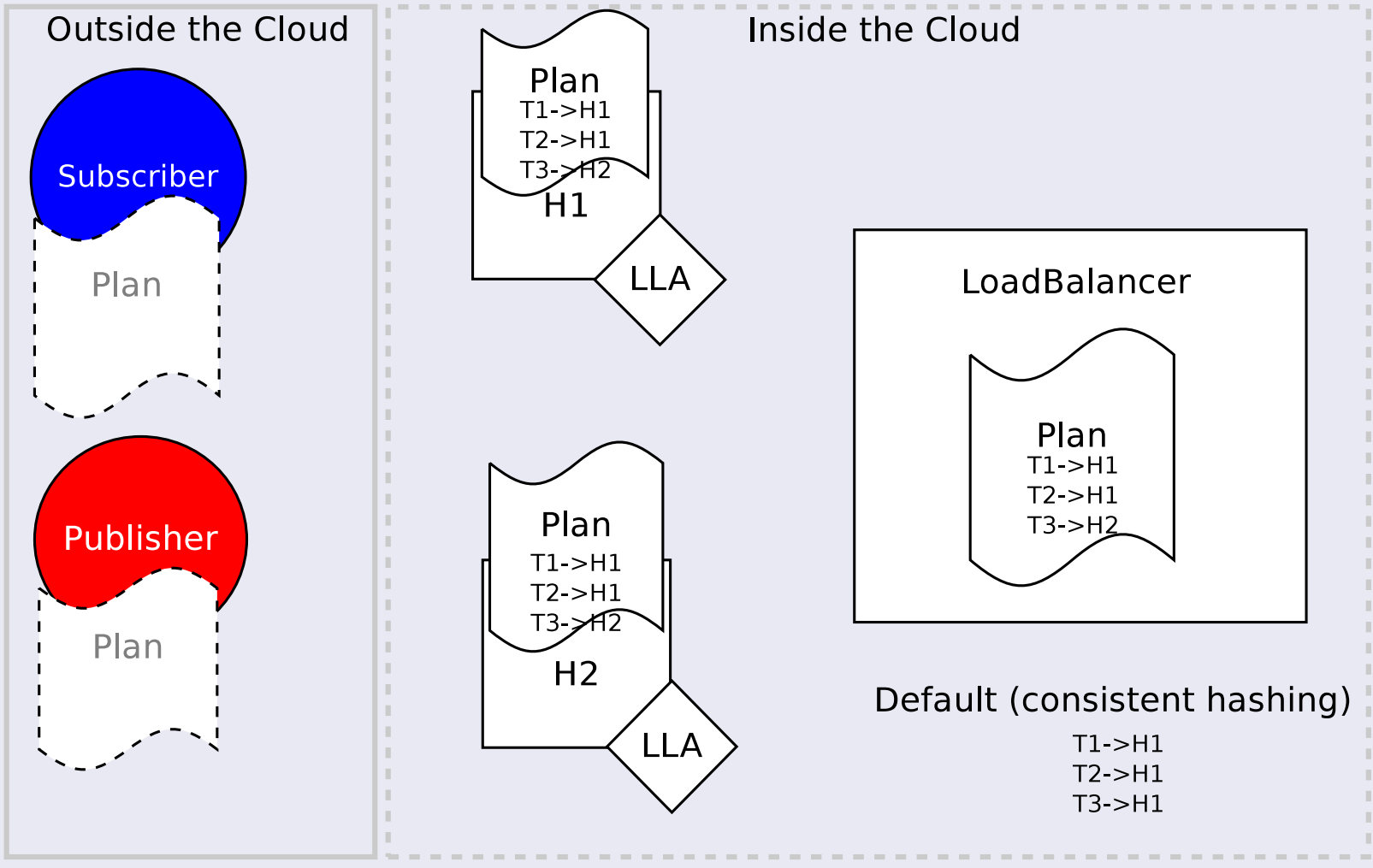
Dynamoth - Plan for Publications and Subscriptions

Publications



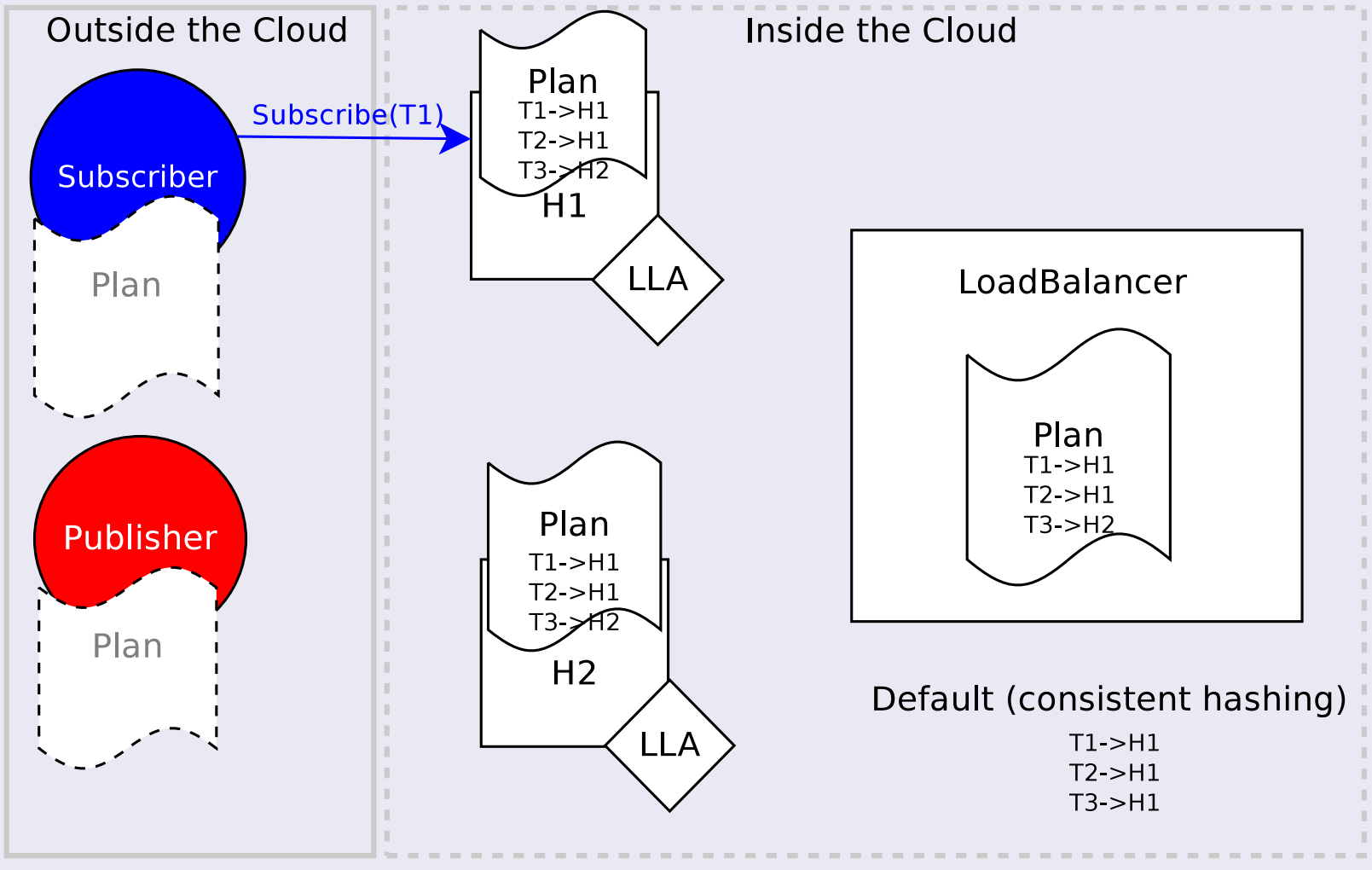
Initial Conditions & Bootstrapping

Initial conditions



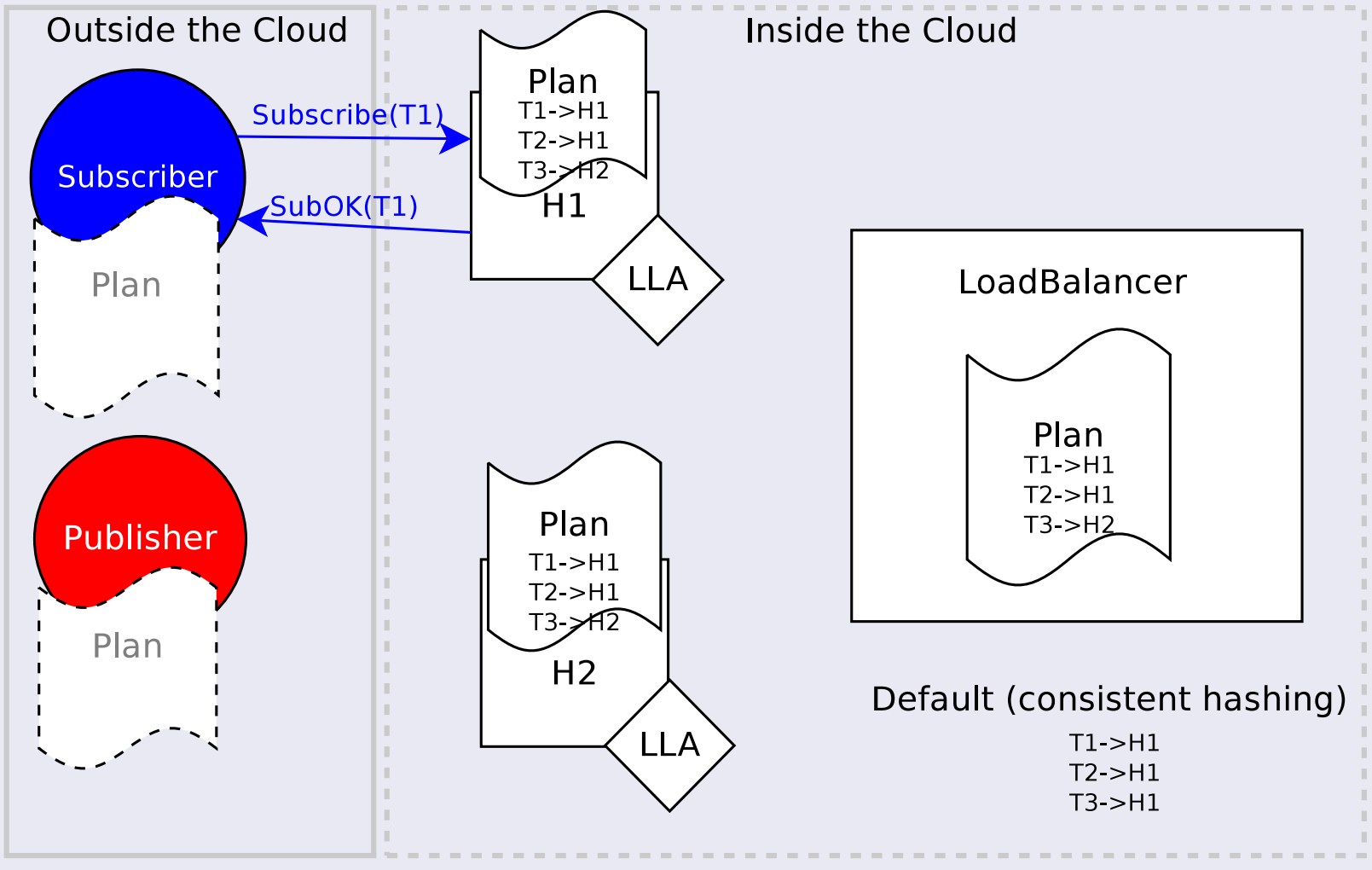
Initial Conditions & Bootstrapping

Subscription: correct server



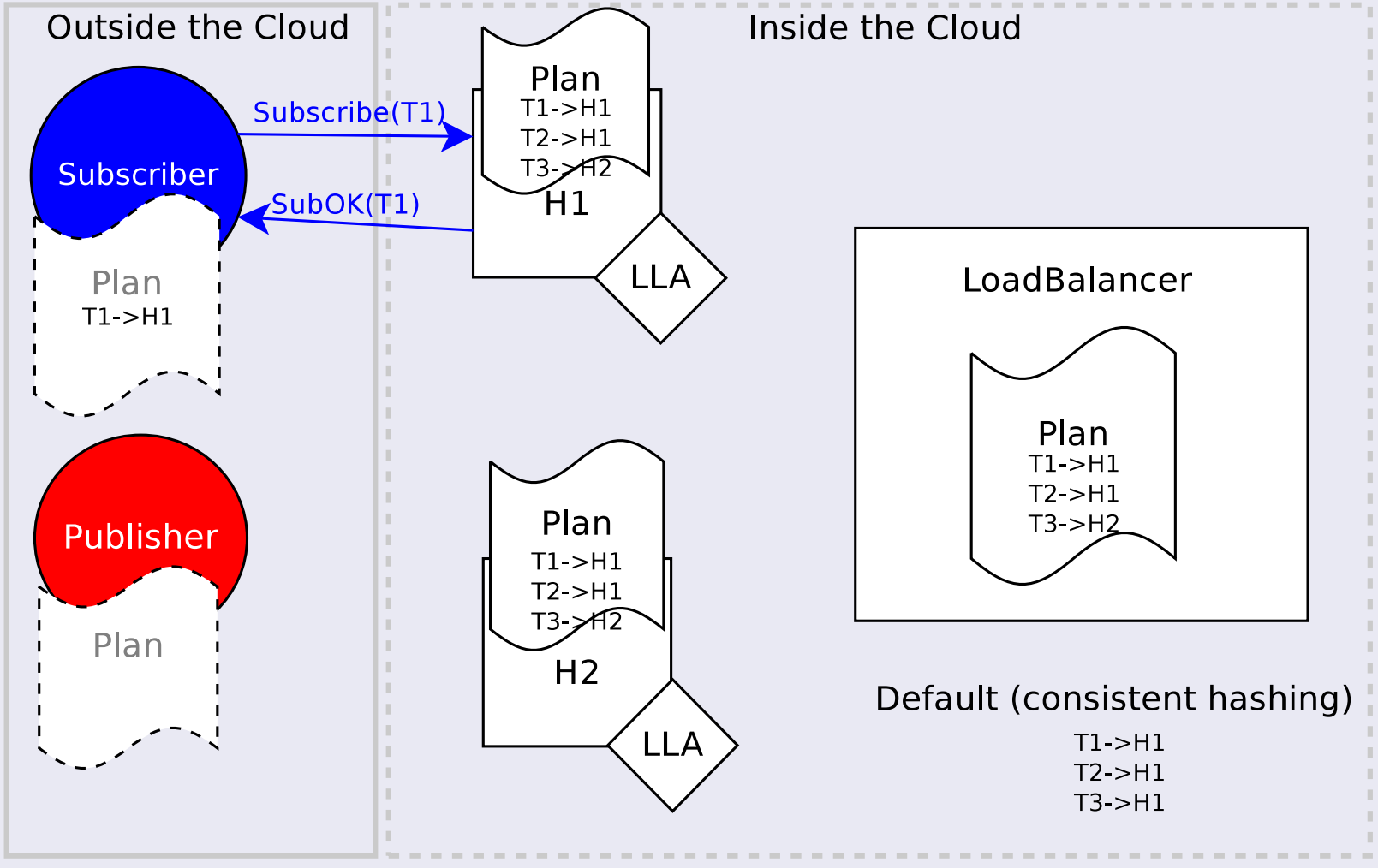
Initial Conditions & Bootstrapping

Subscription: correct server



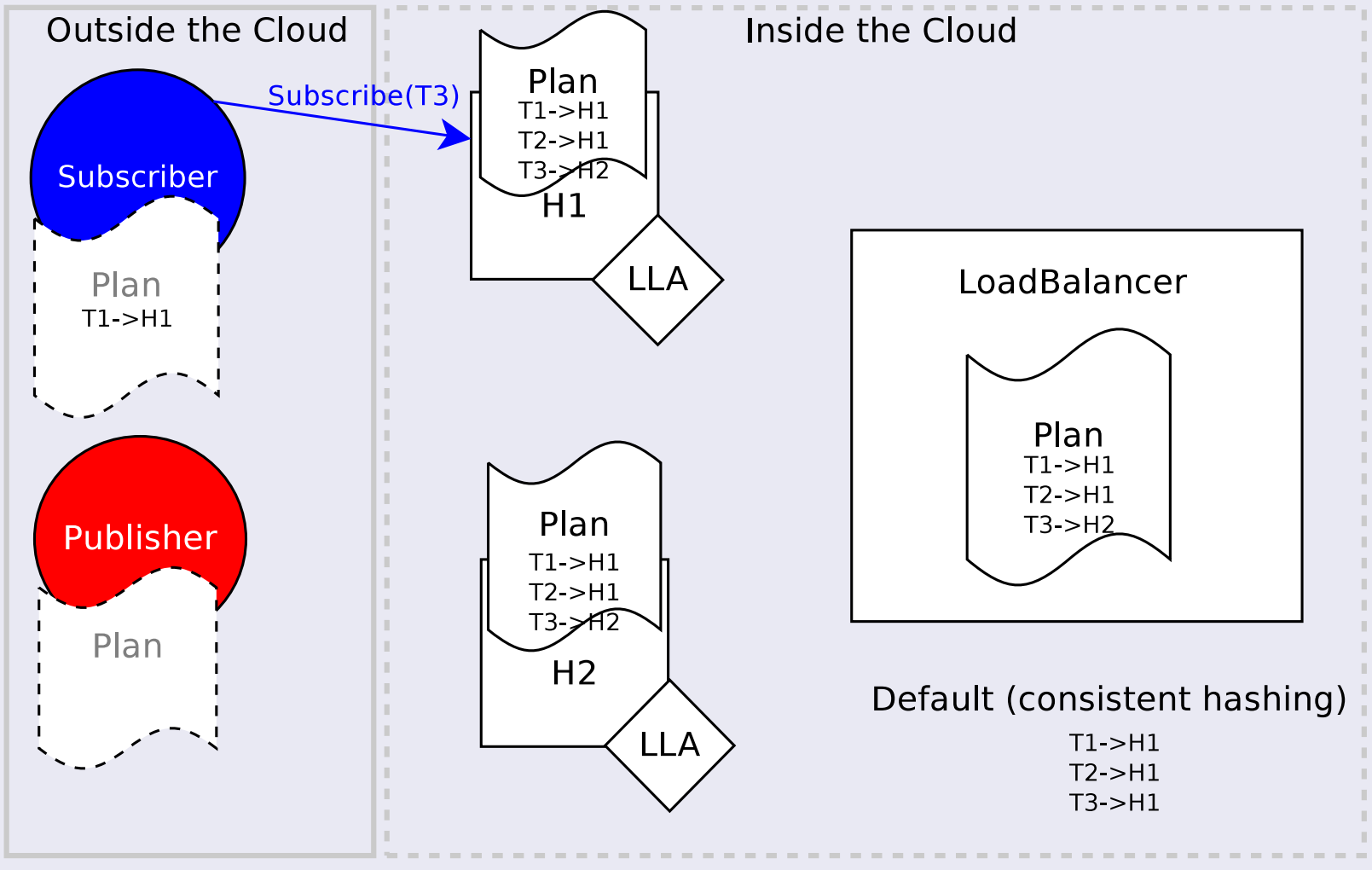
Initial Conditions & Bootstrapping

Subscription: correct server / client plan update



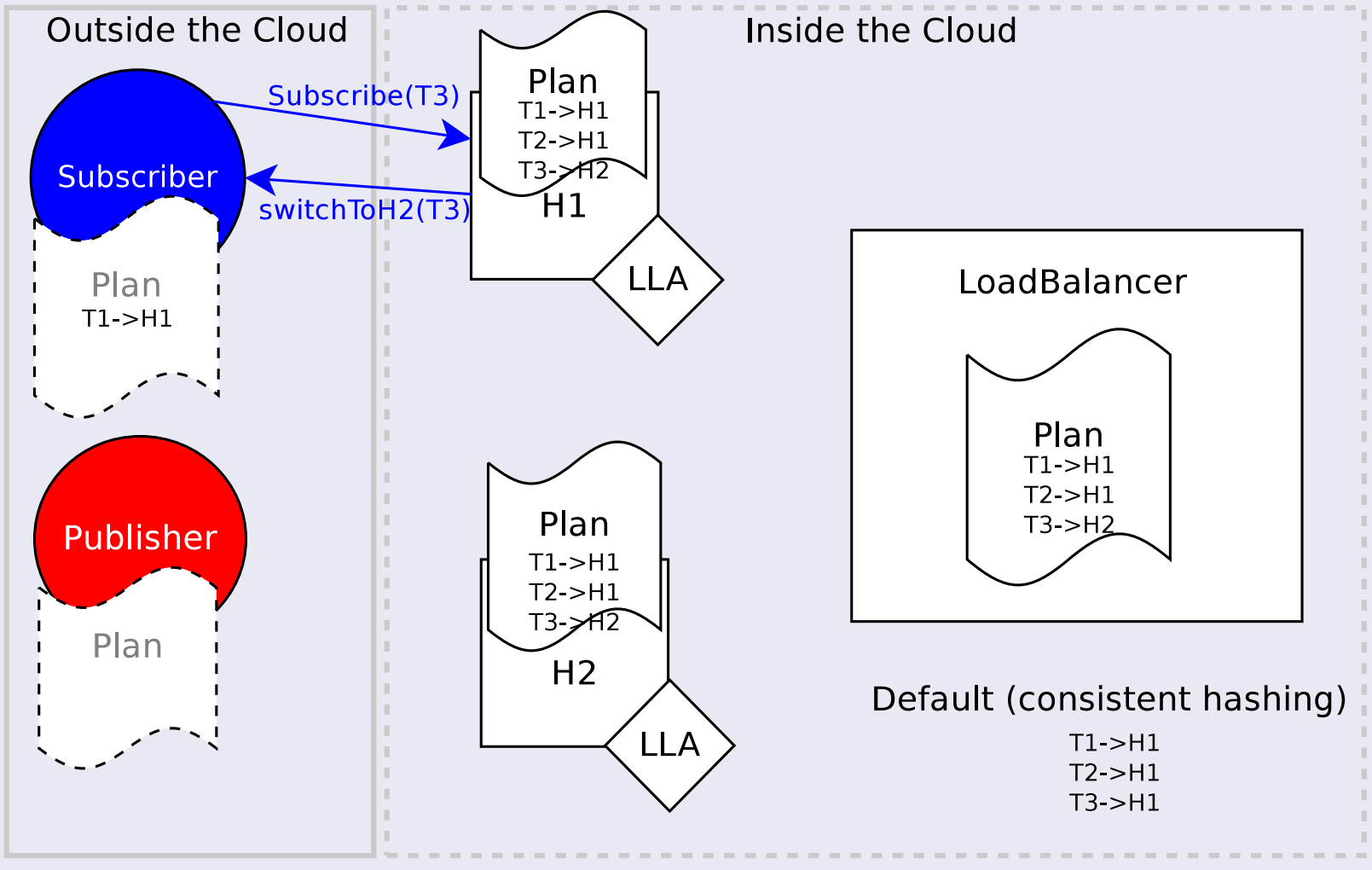
Initial Conditions & Bootstrapping

Subscription: incorrect server



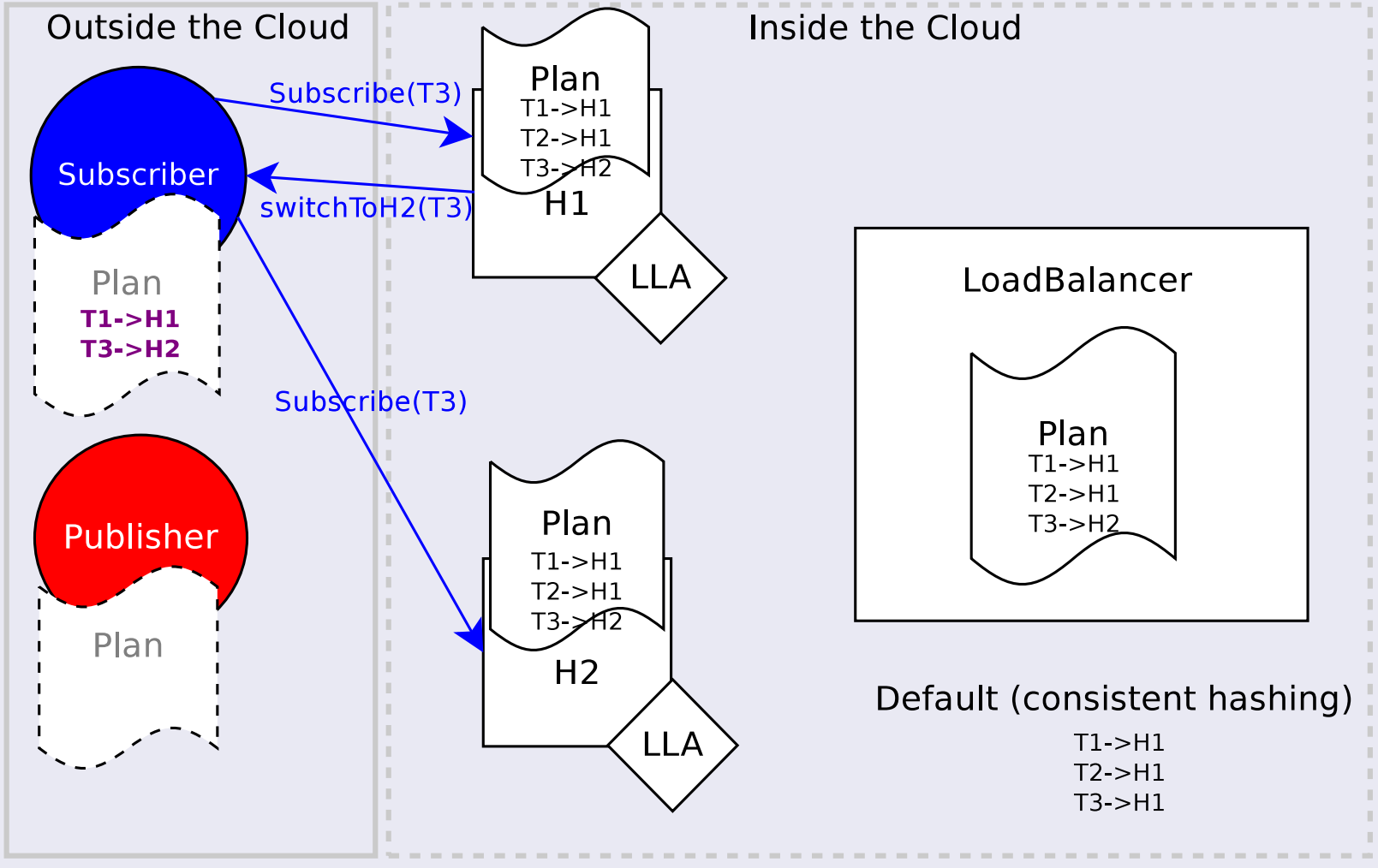
Initial Conditions & Bootstrapping

Subscription: incorrect server



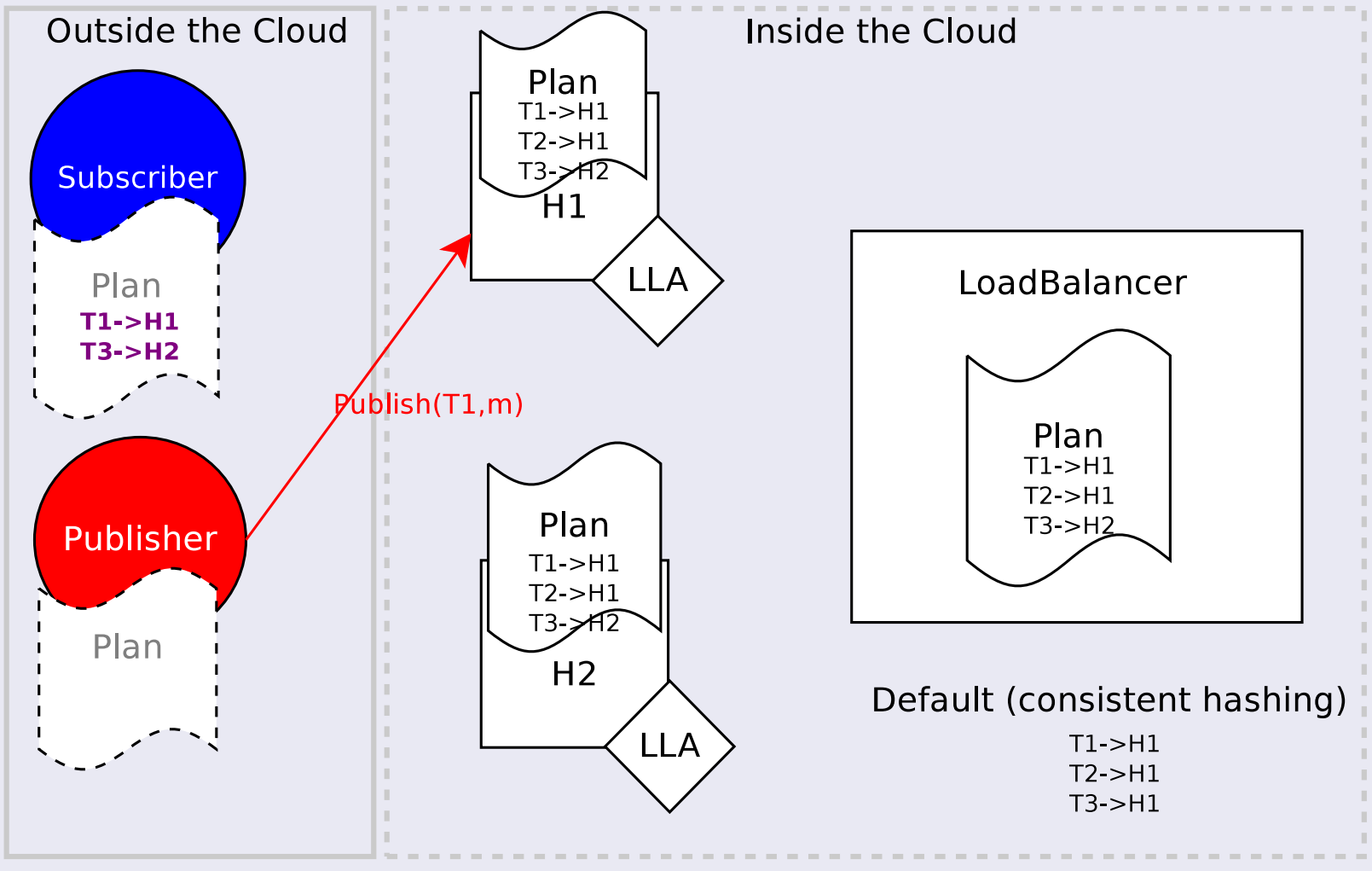
Initial Conditions & Bootstrapping

Subscription: incorrect server / client plan update



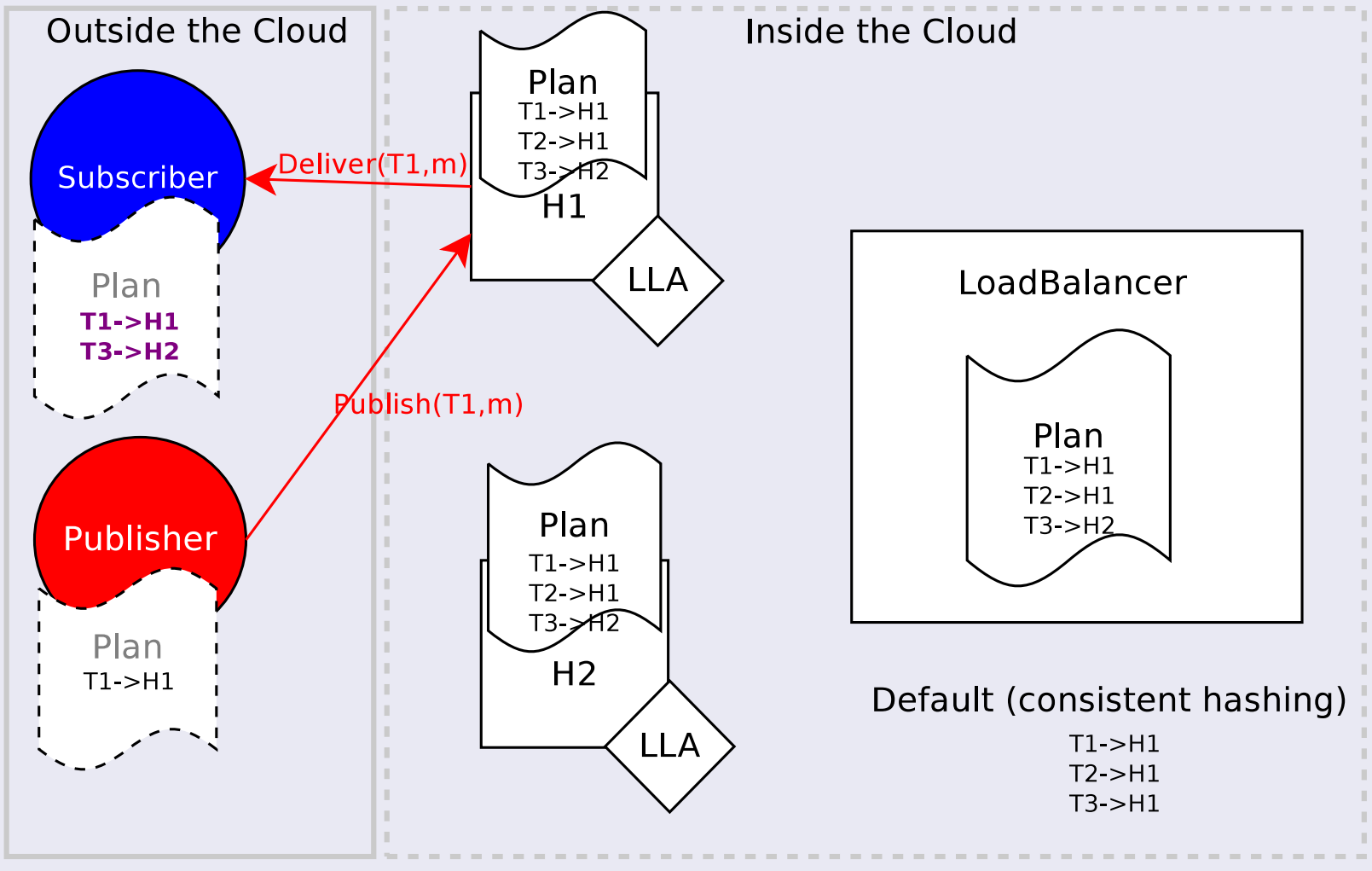
Initial Conditions & Bootstrapping

Publication: correct server



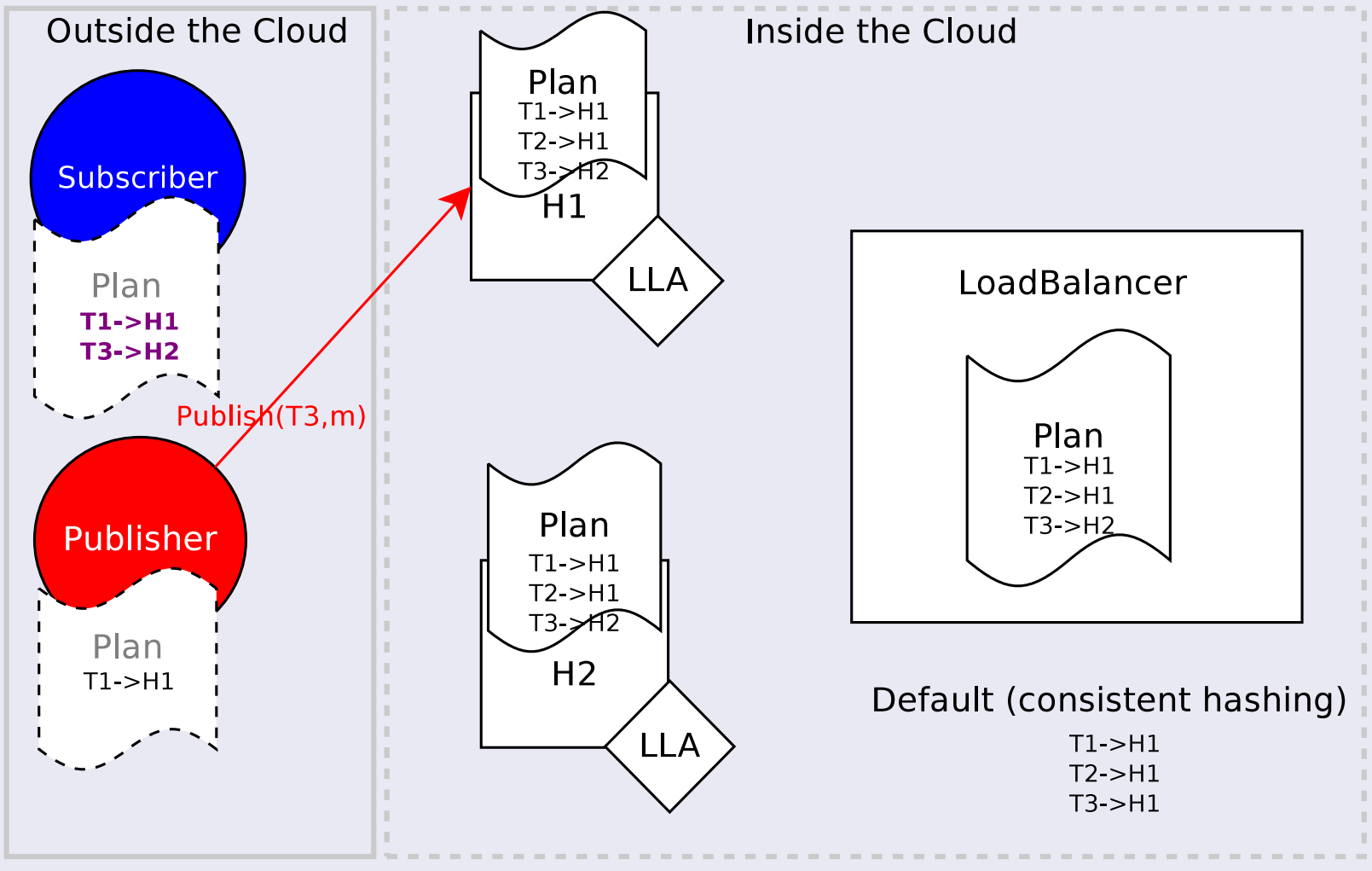
Initial Conditions & Bootstrapping

Publication: correct server



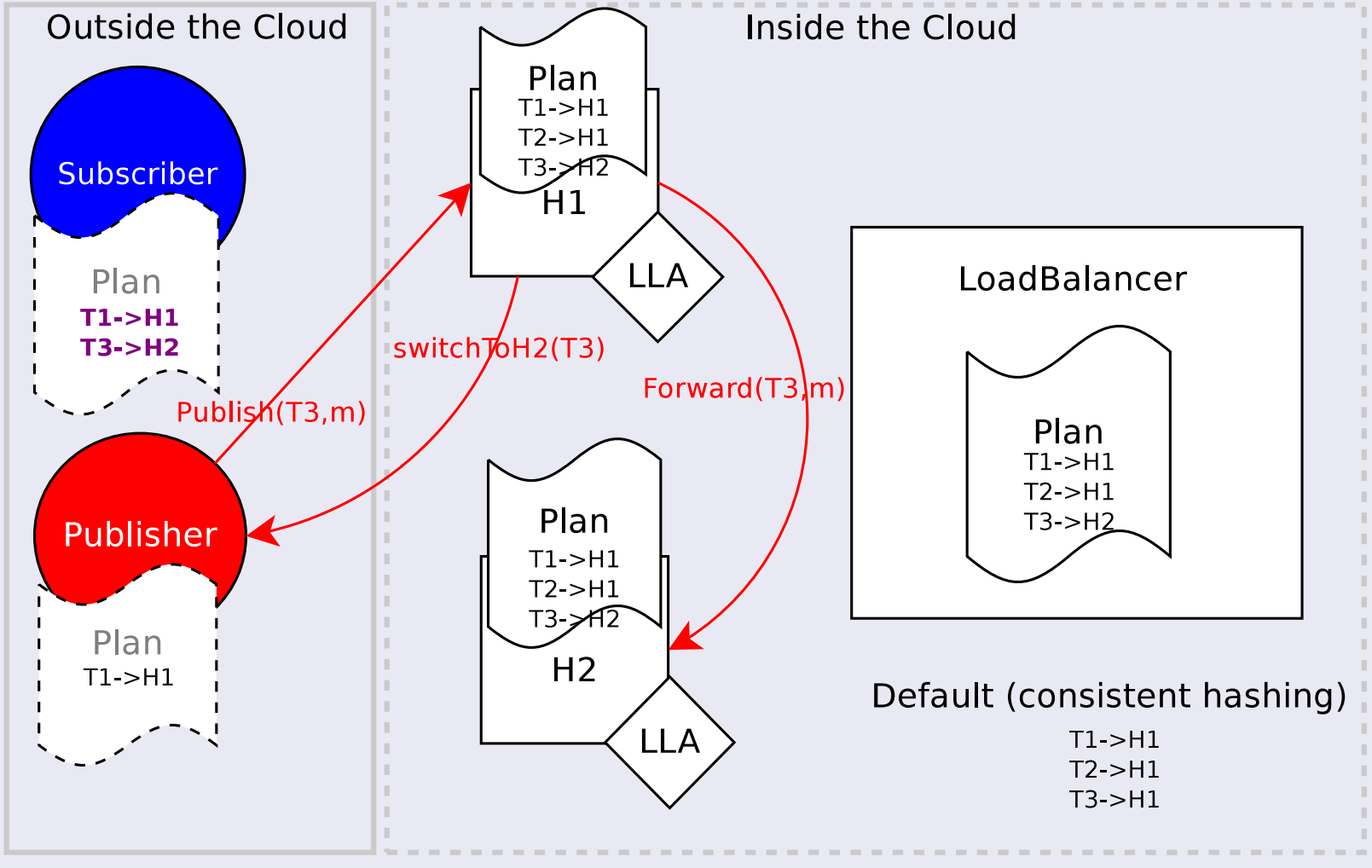
Initial Conditions & Bootstrapping

Publication: incorrect server



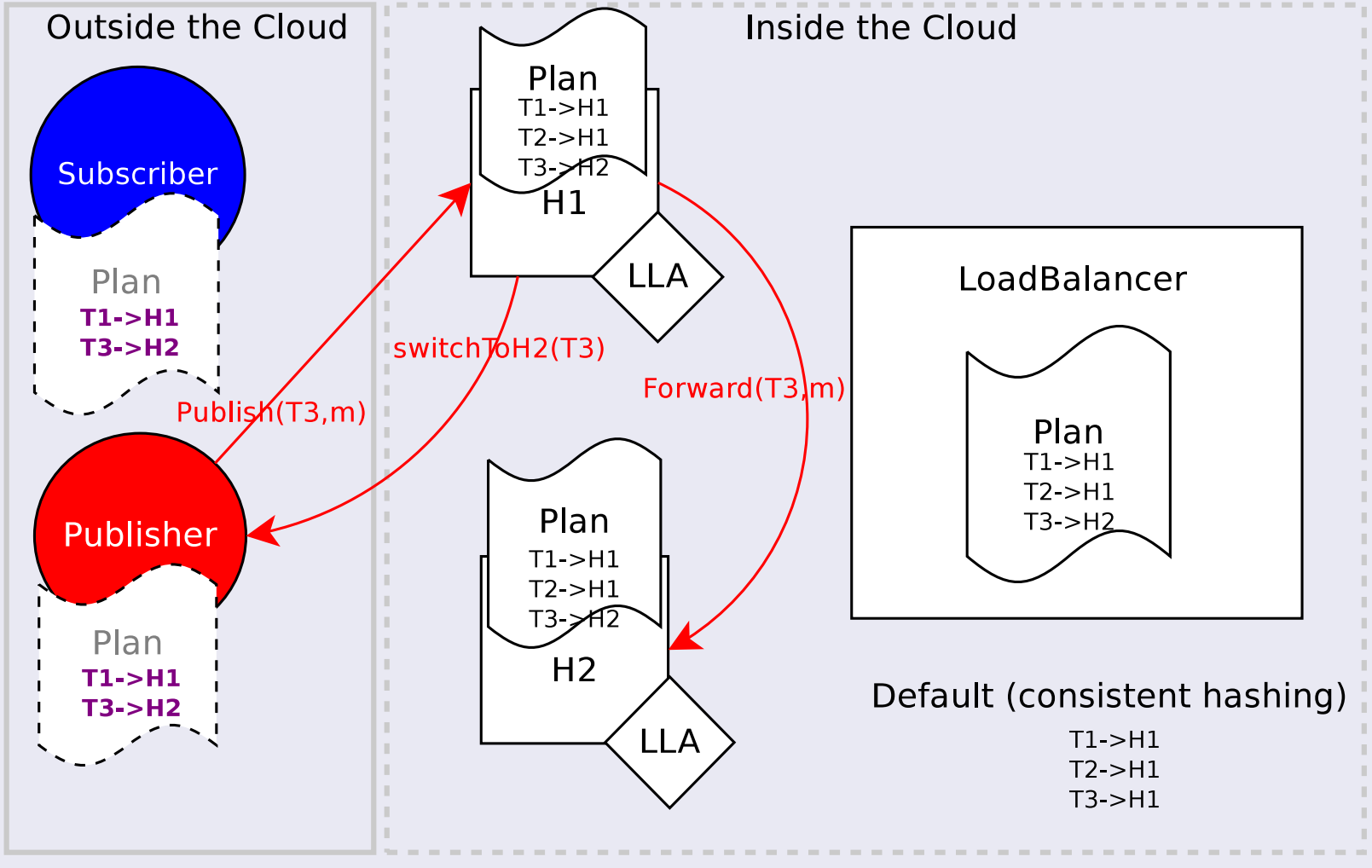
Initial Conditions & Bootstrapping

Publication: incorrect server / forward



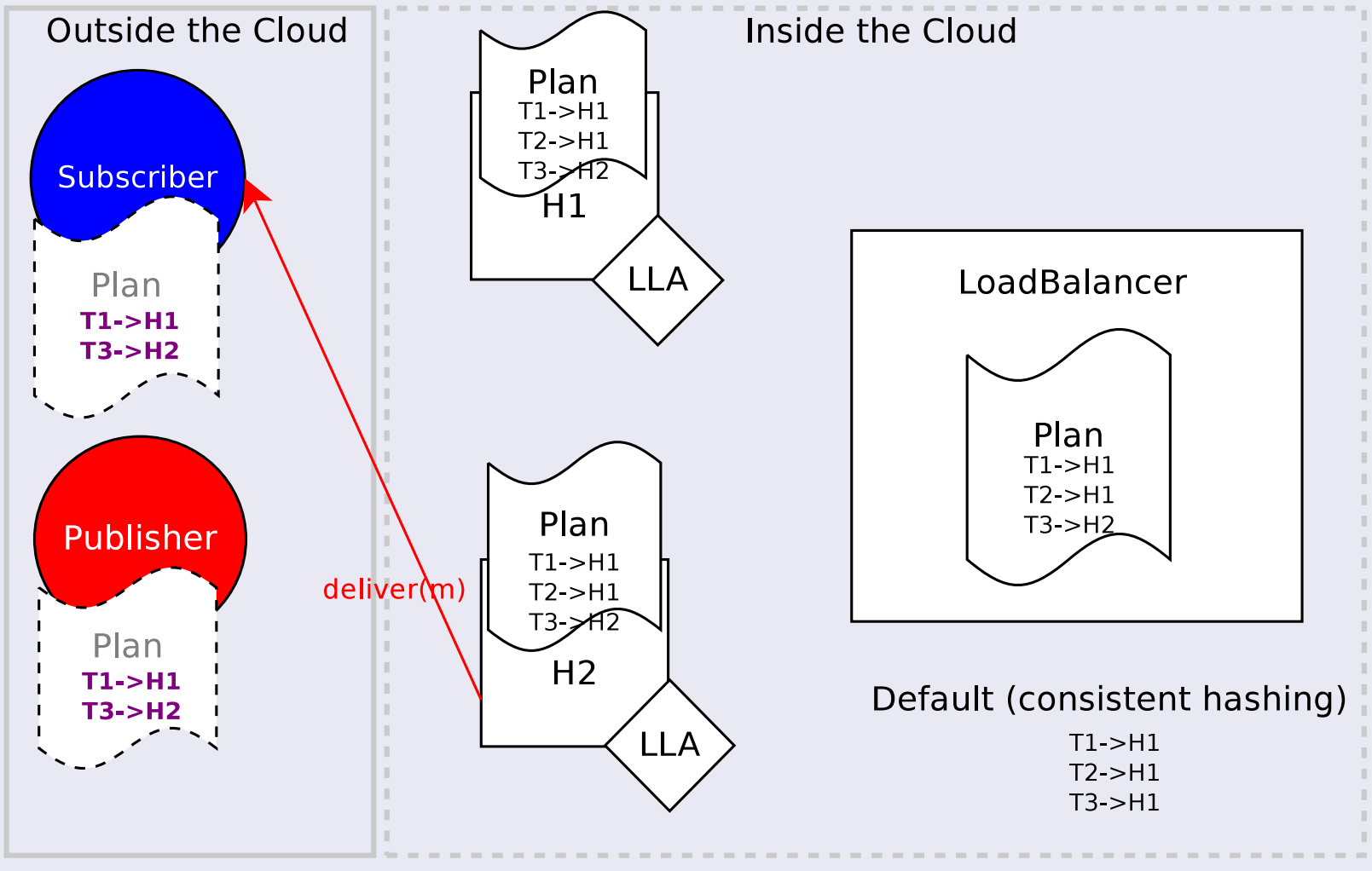
Initial Conditions & Bootstrapping

Publication: incorrect server / client plan update



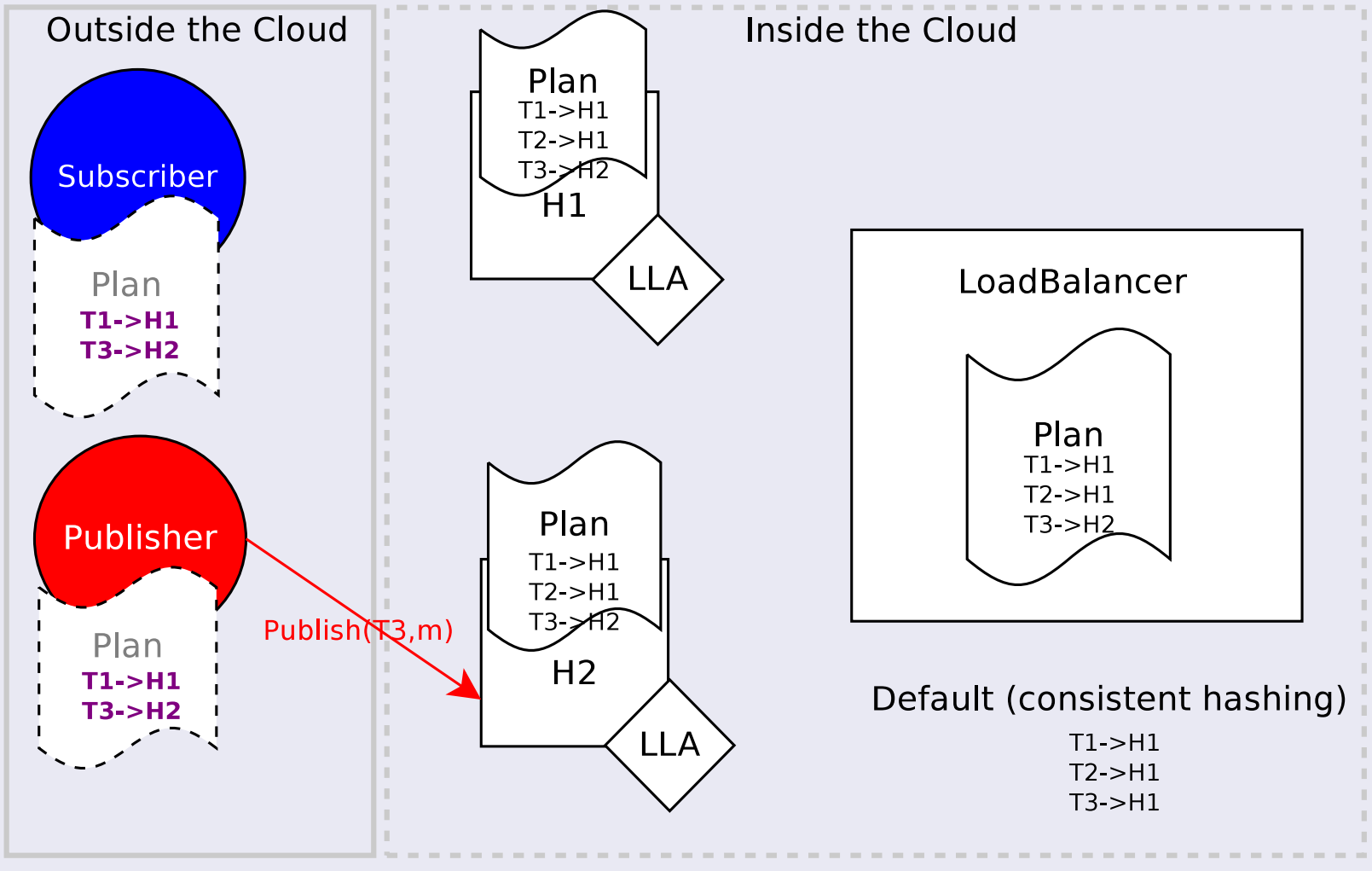
Initial Conditions & Bootstrapping

Publication: incorrect server / message delivery



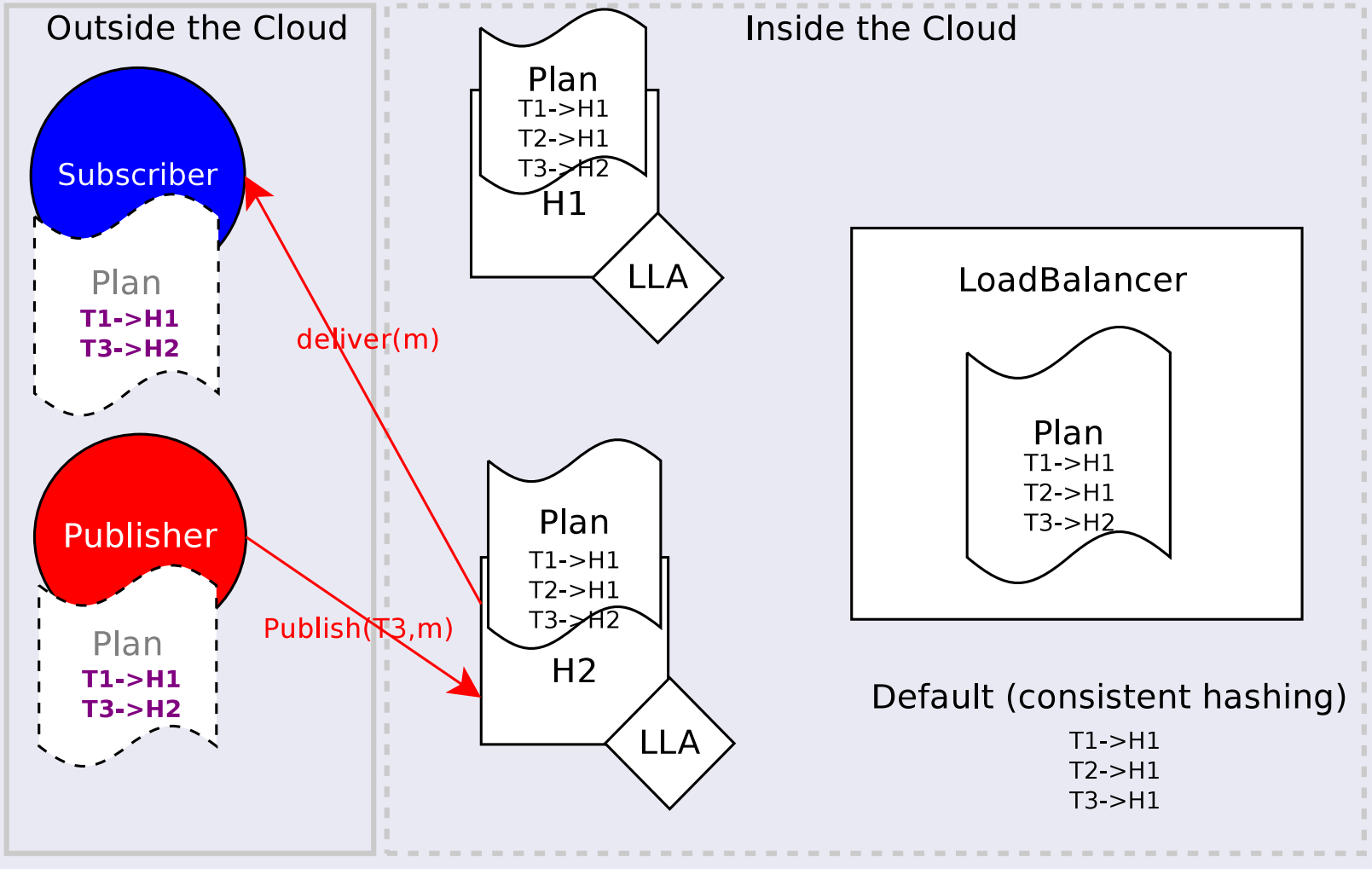
Initial Conditions & Bootstrapping

Publication: correct server (with updated client plan)



Initial Conditions & Bootstrapping

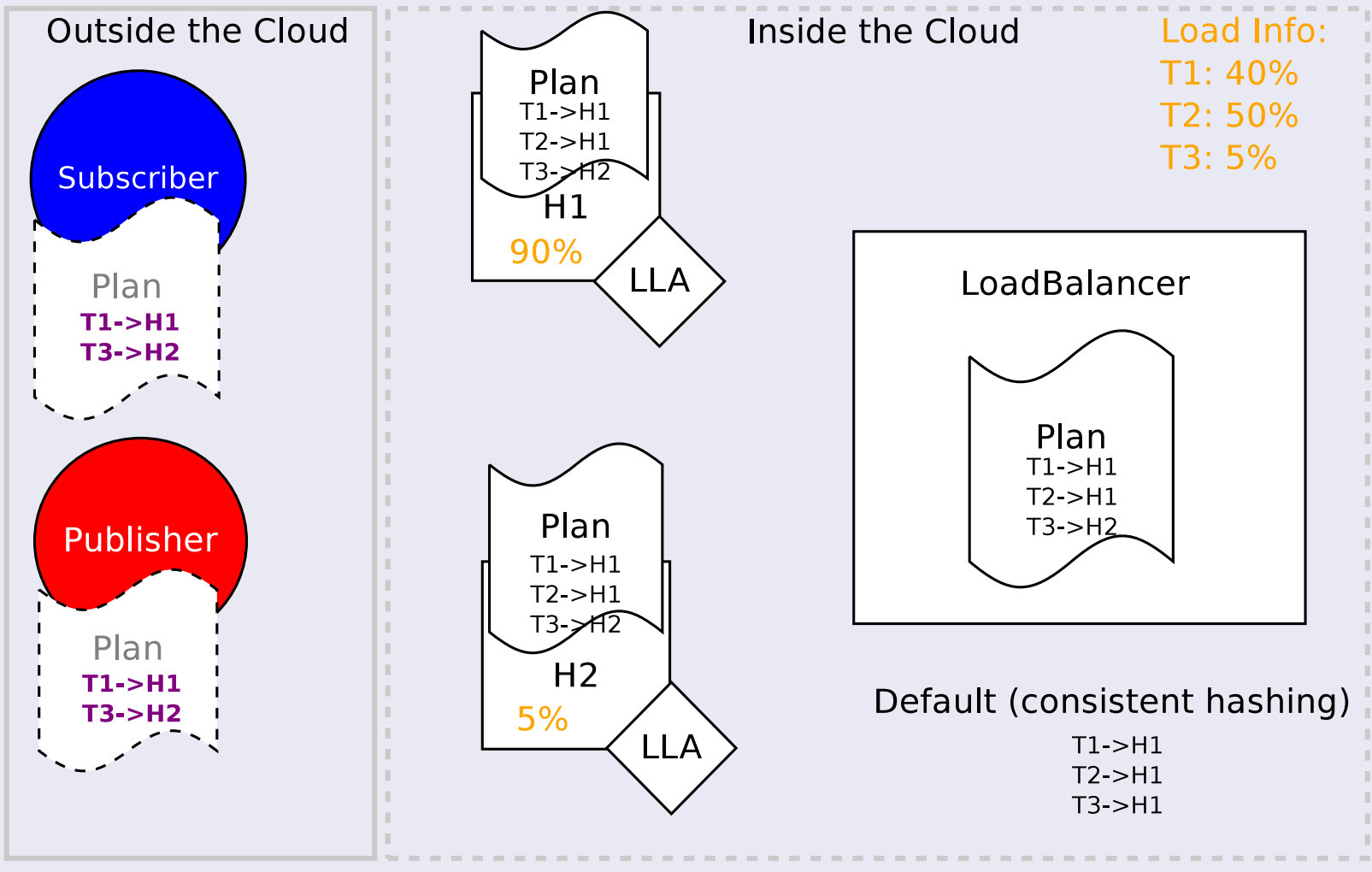
Publication: correct server (with updated client plan)



Load Balancing

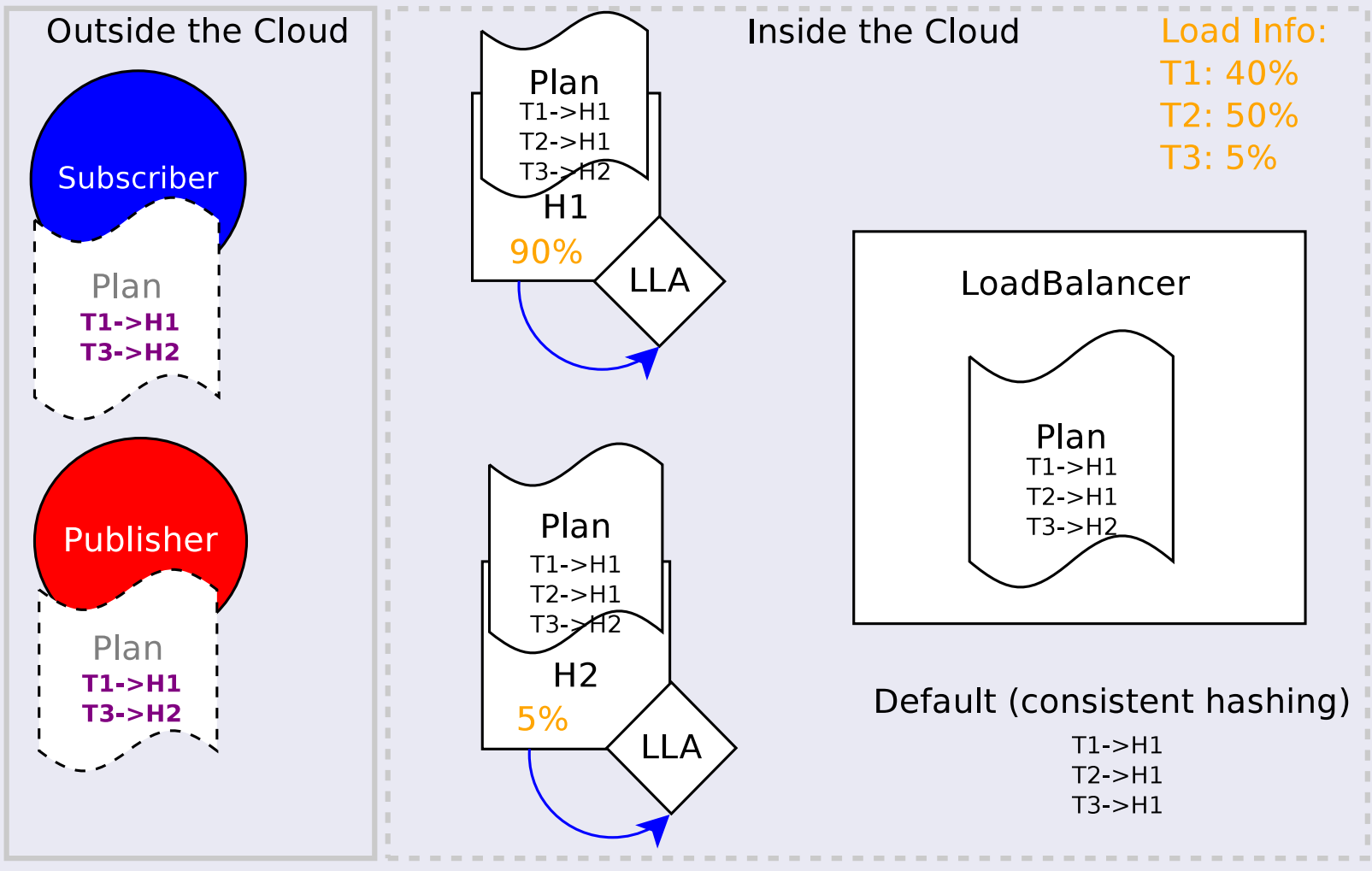
Load Balancing & Reconfiguration

Initial conditions



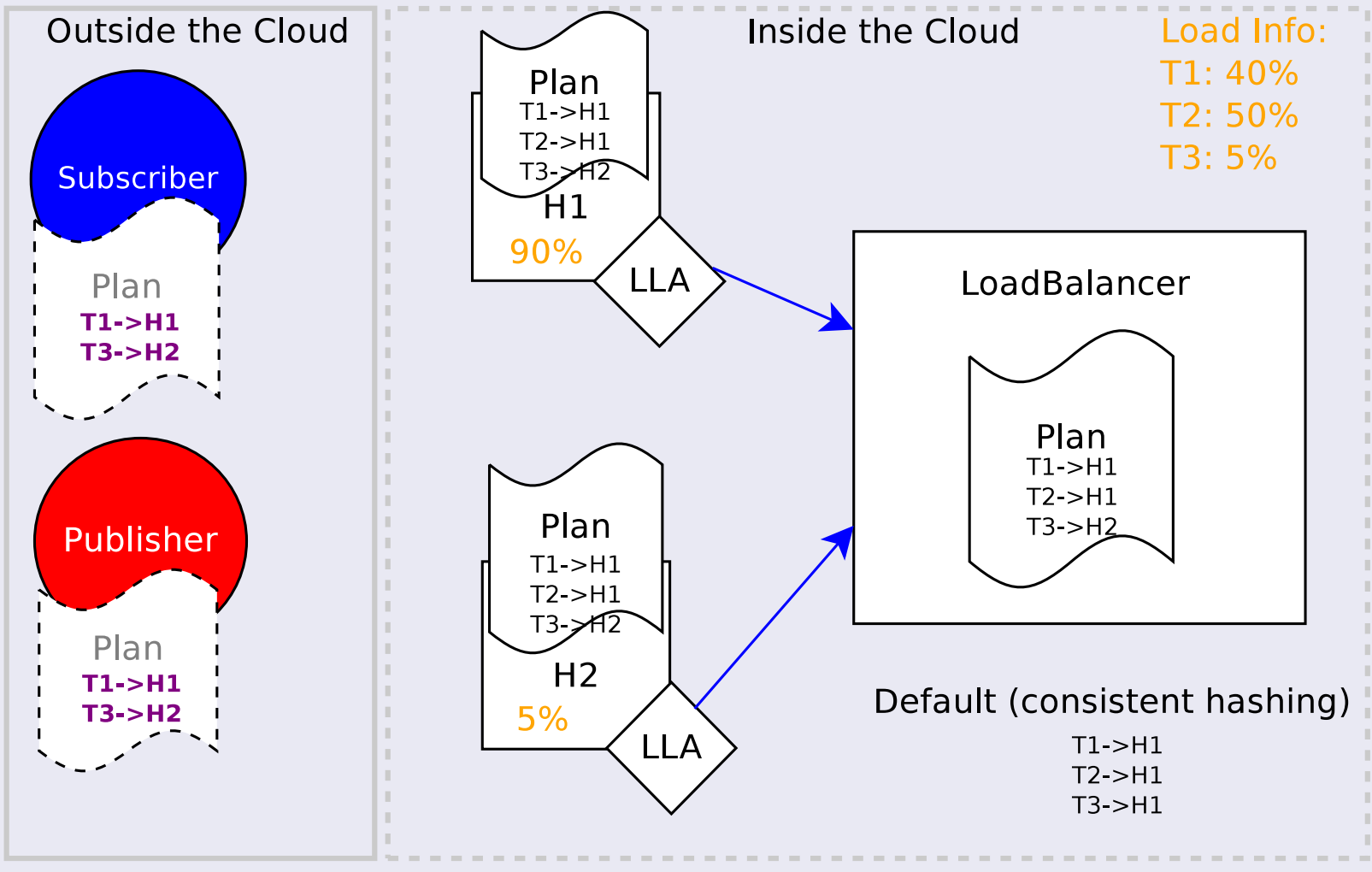
Load Balancing & Reconfiguration

Stats collected by Local Load Analyzer

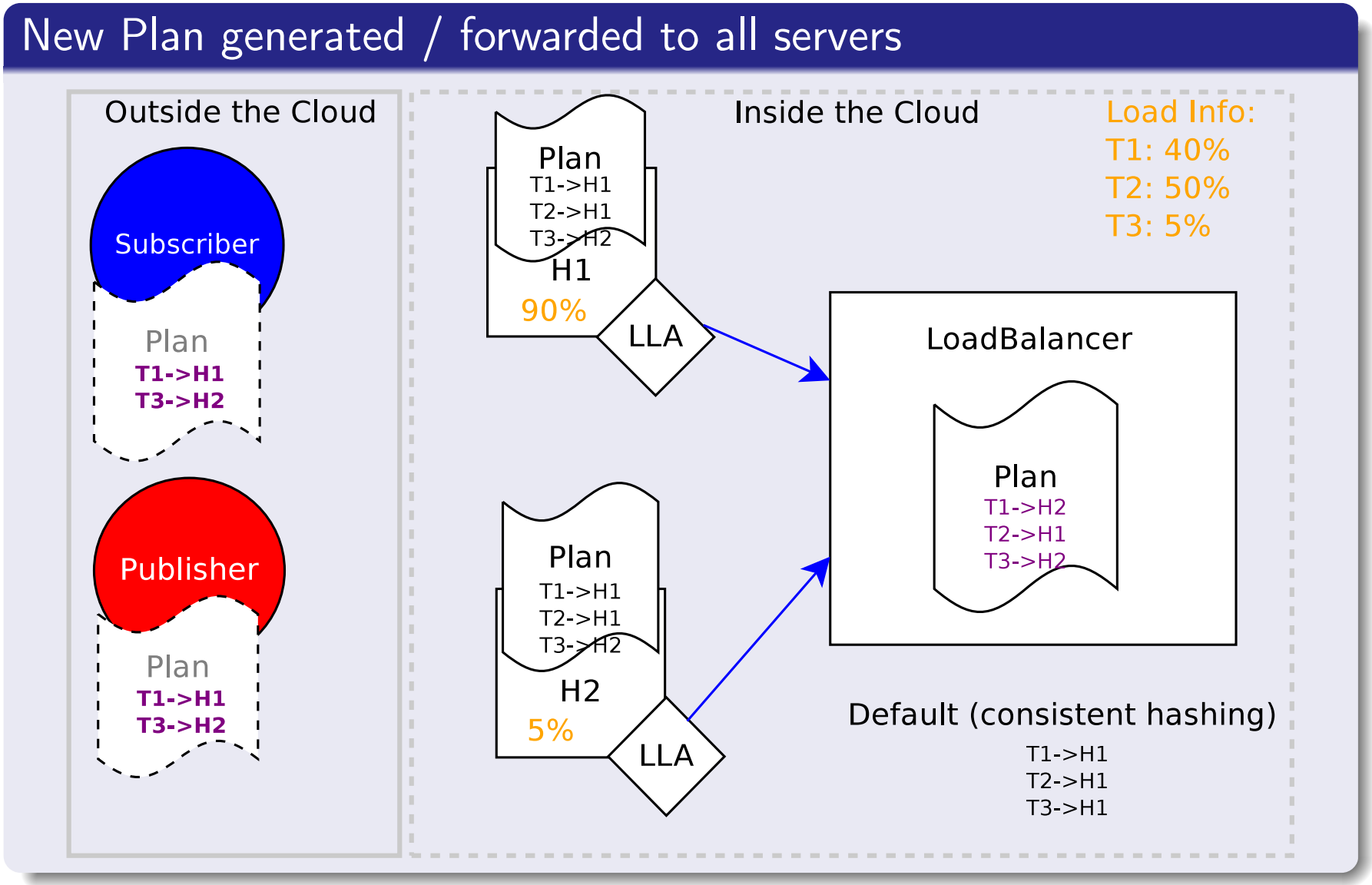


Load Balancing & Reconfiguration

Stats sent to Load Balancer

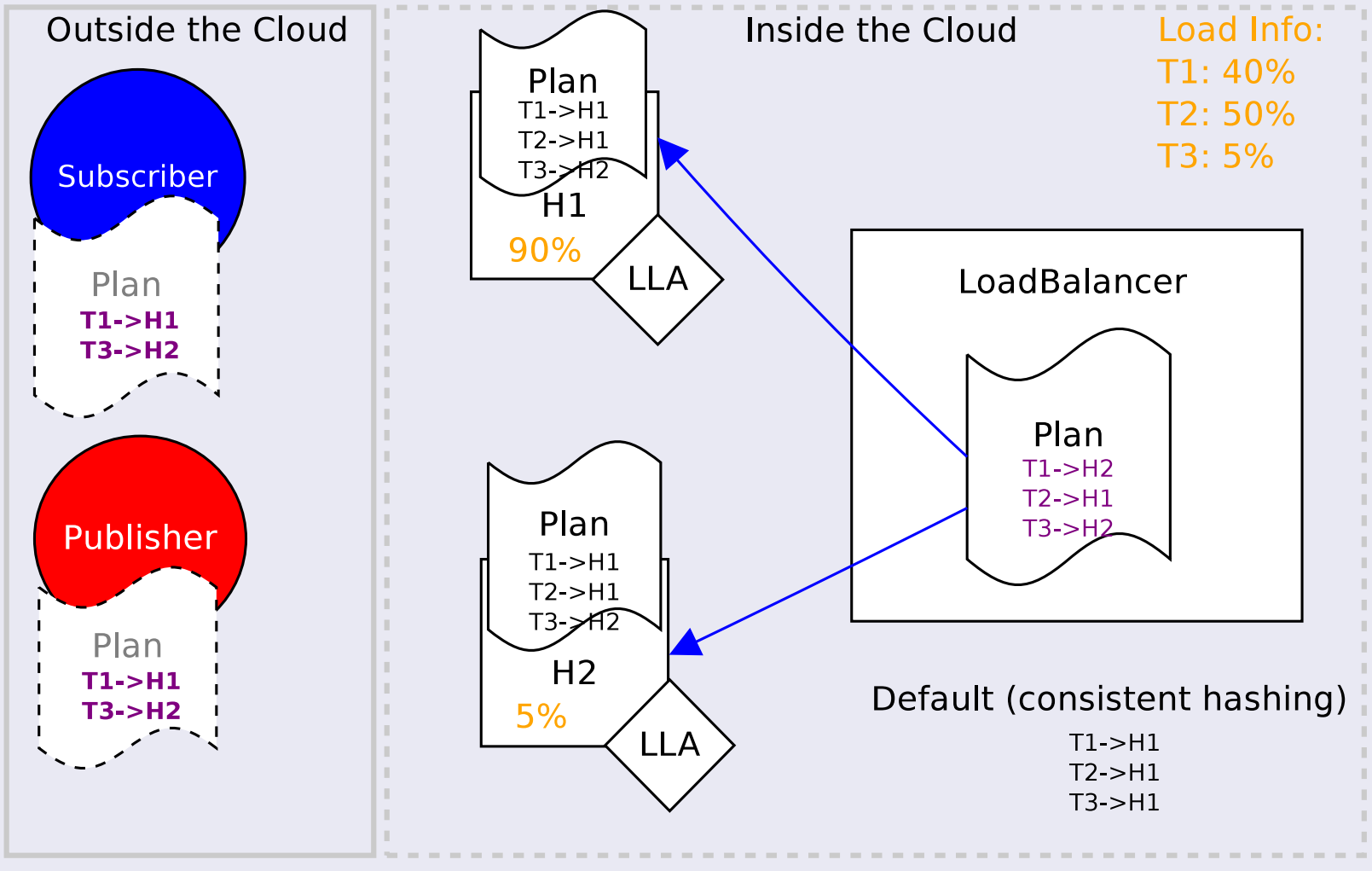


Load Balancing & Reconfiguration



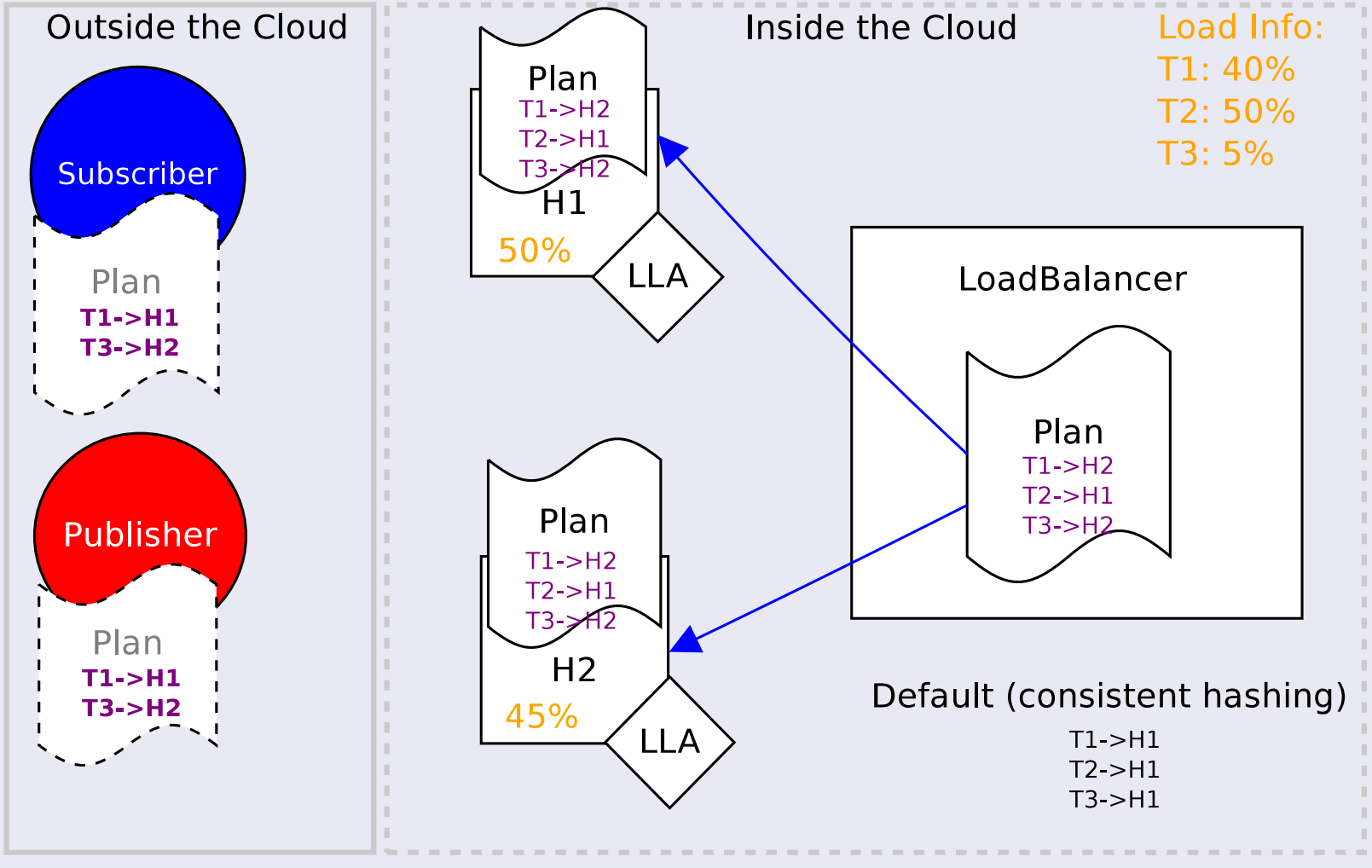
Load Balancing & Reconfiguration

New Plan generated / forwarded to all servers



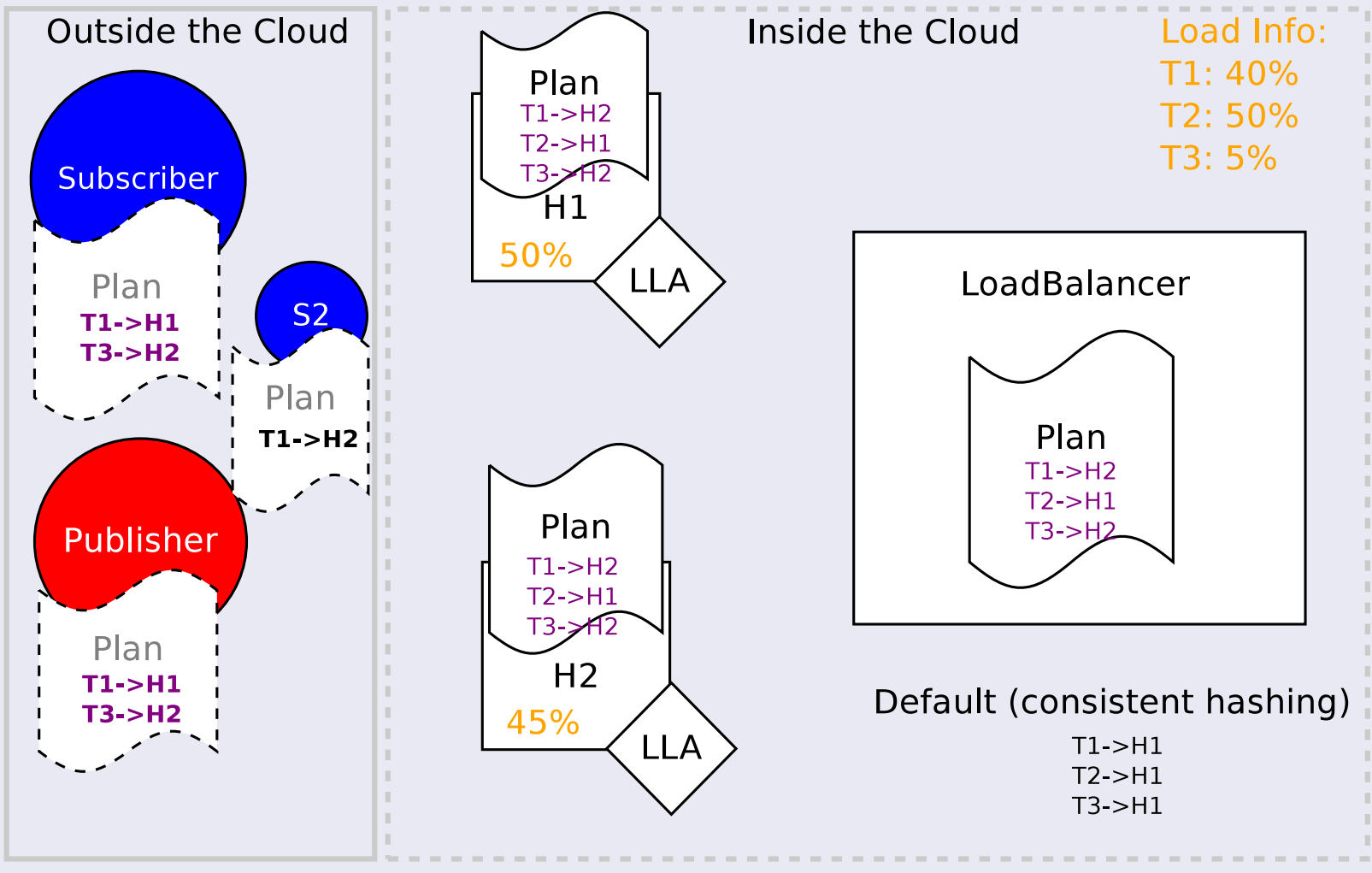
Load Balancing & Reconfiguration

New Plan generated / forwarded to all servers



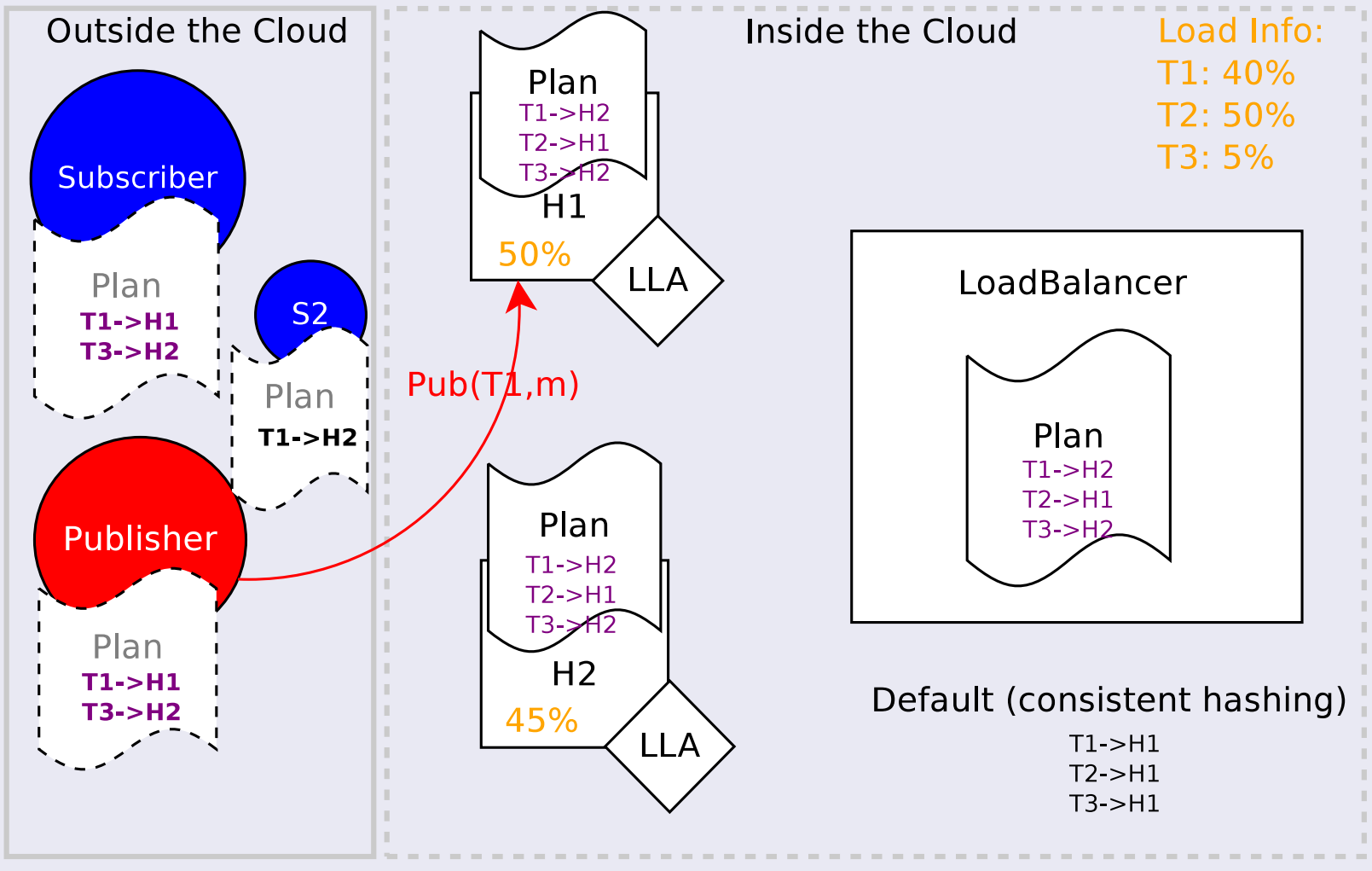
Load Balancing & Reconfiguration

Introduction of new server S_2



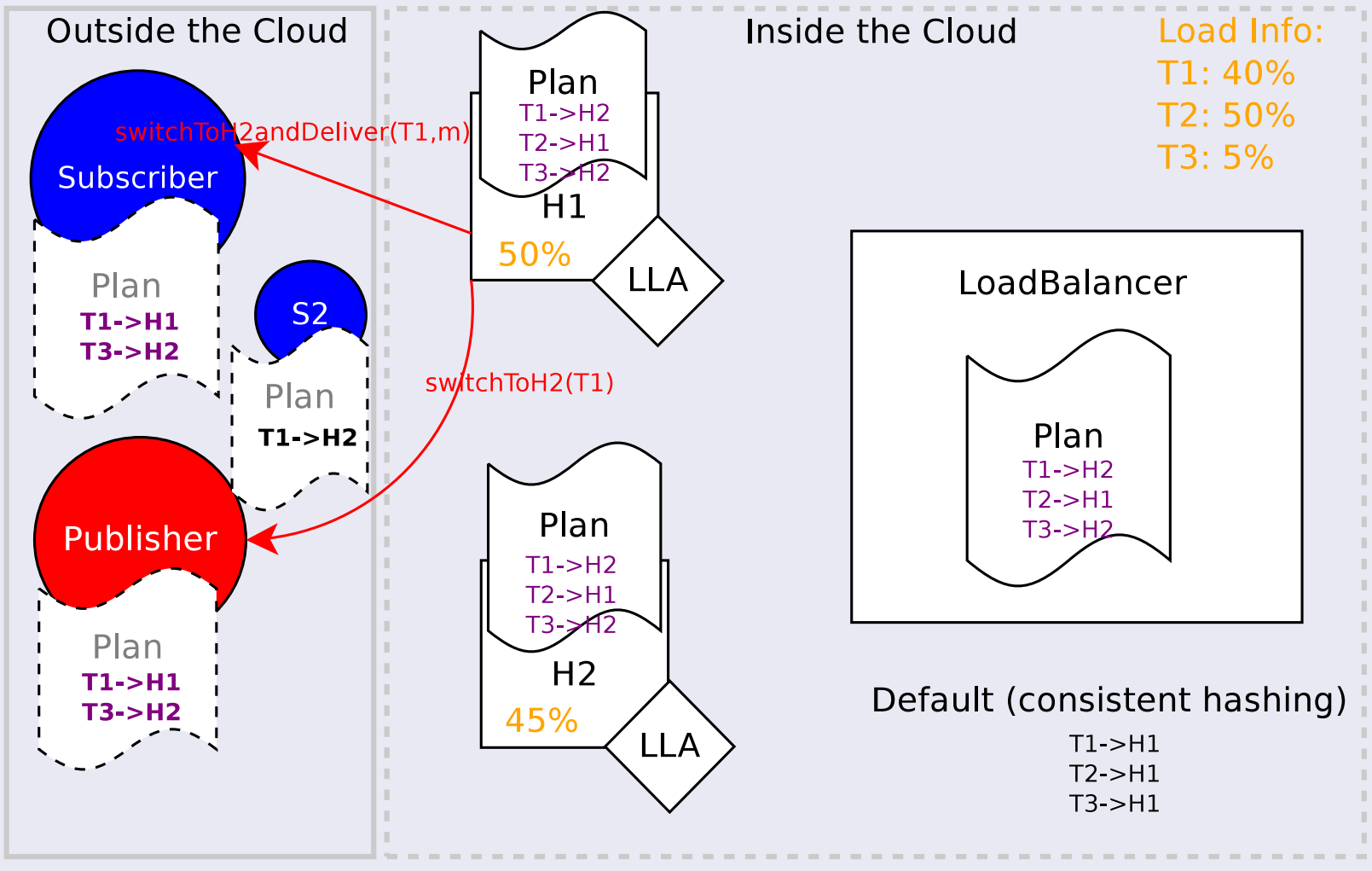
Load Balancing & Reconfiguration

Publication: outdated server



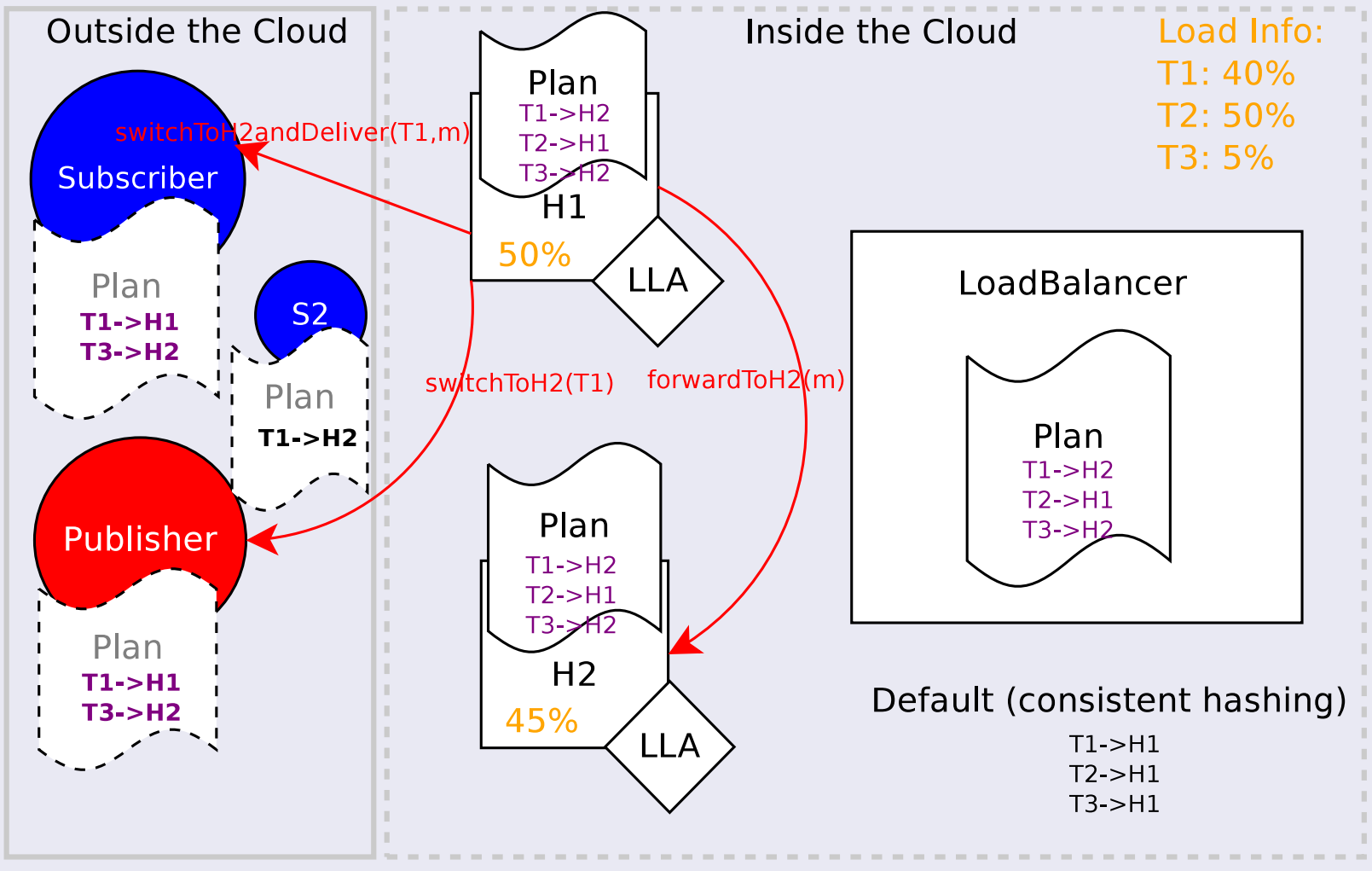
Load Balancing & Reconfiguration

Publication: delivered and request to update client plans

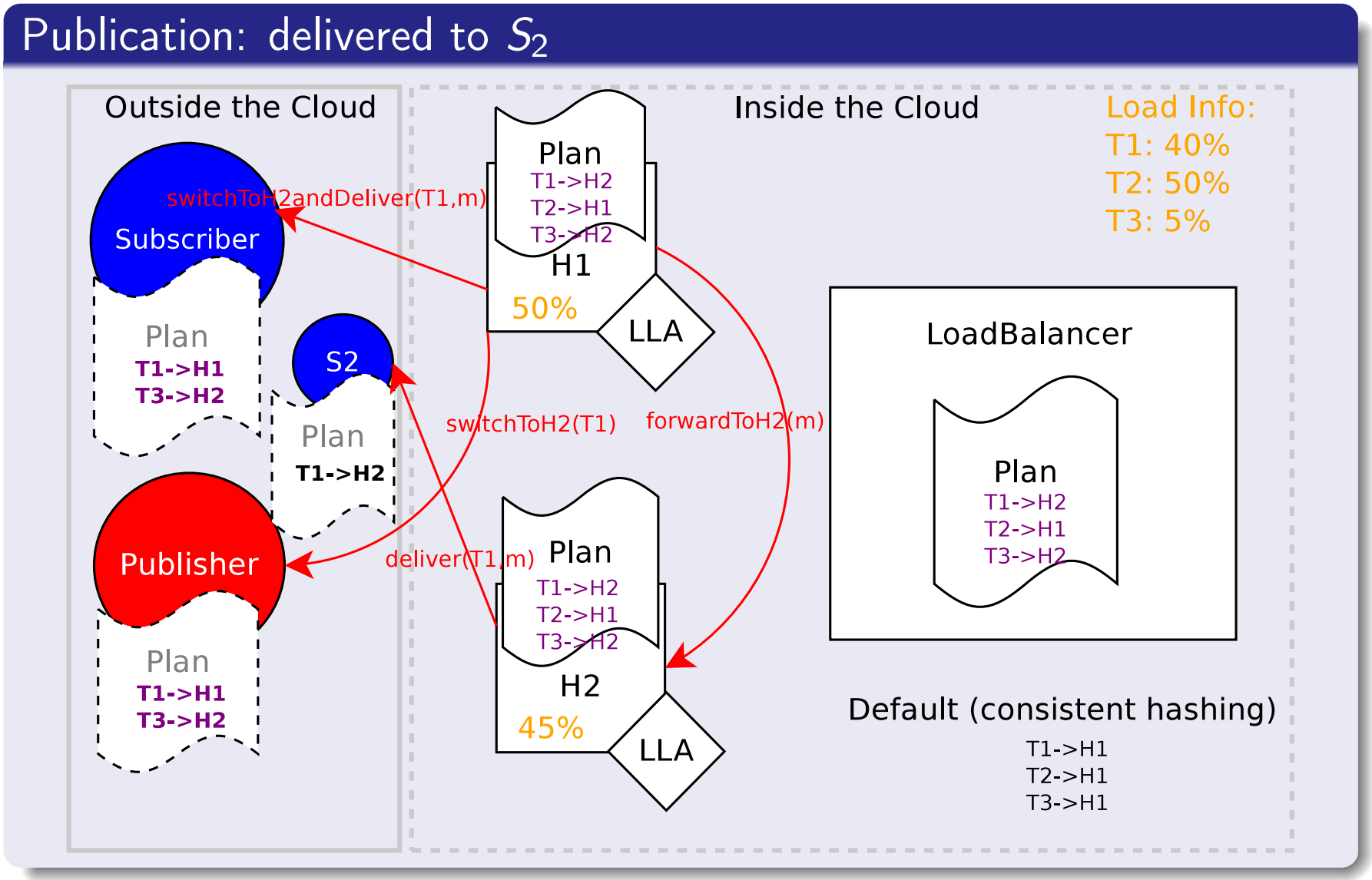


Load Balancing & Reconfiguration

Publication: forwarded to H_2

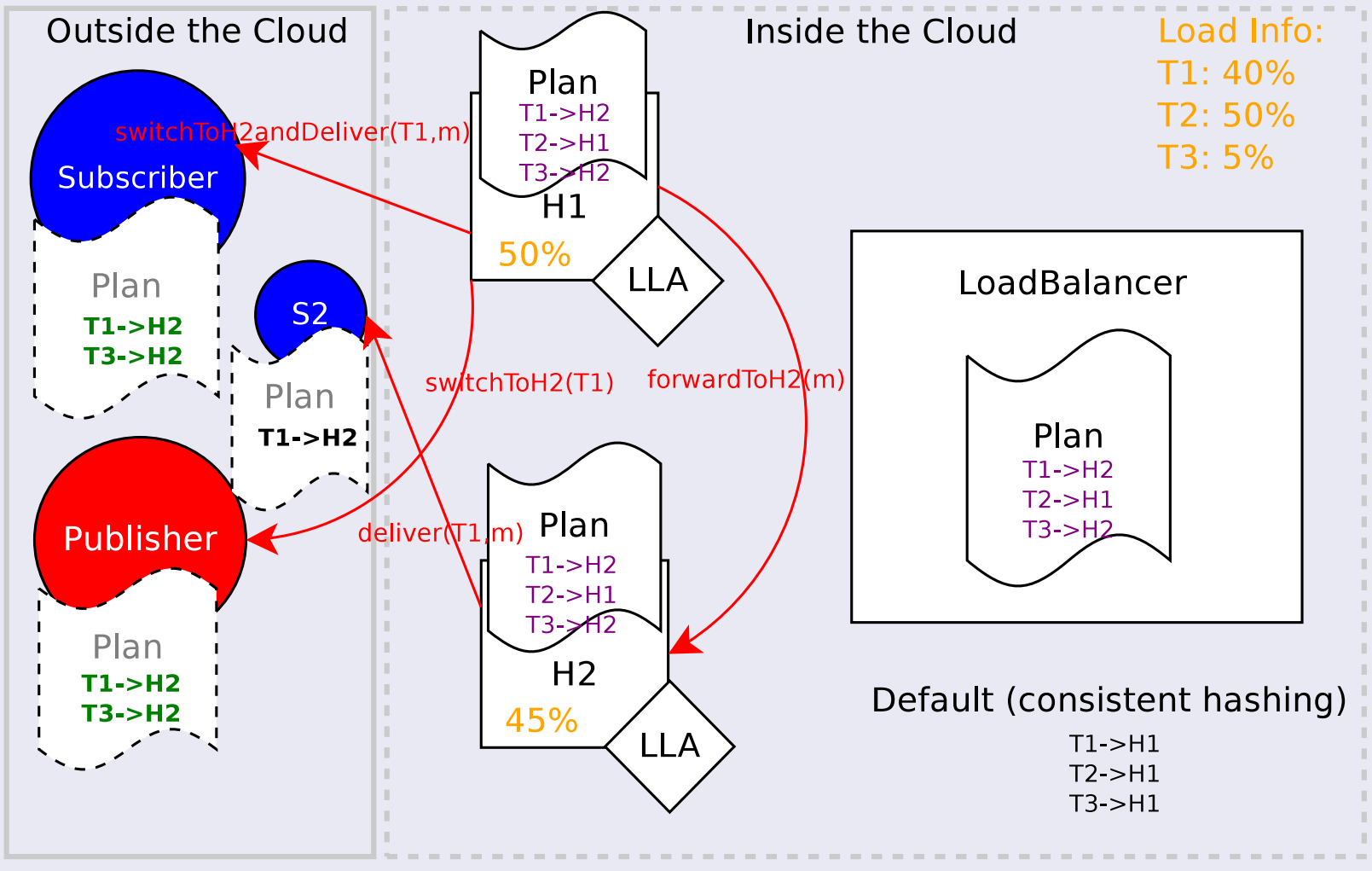


Load Balancing & Reconfiguration



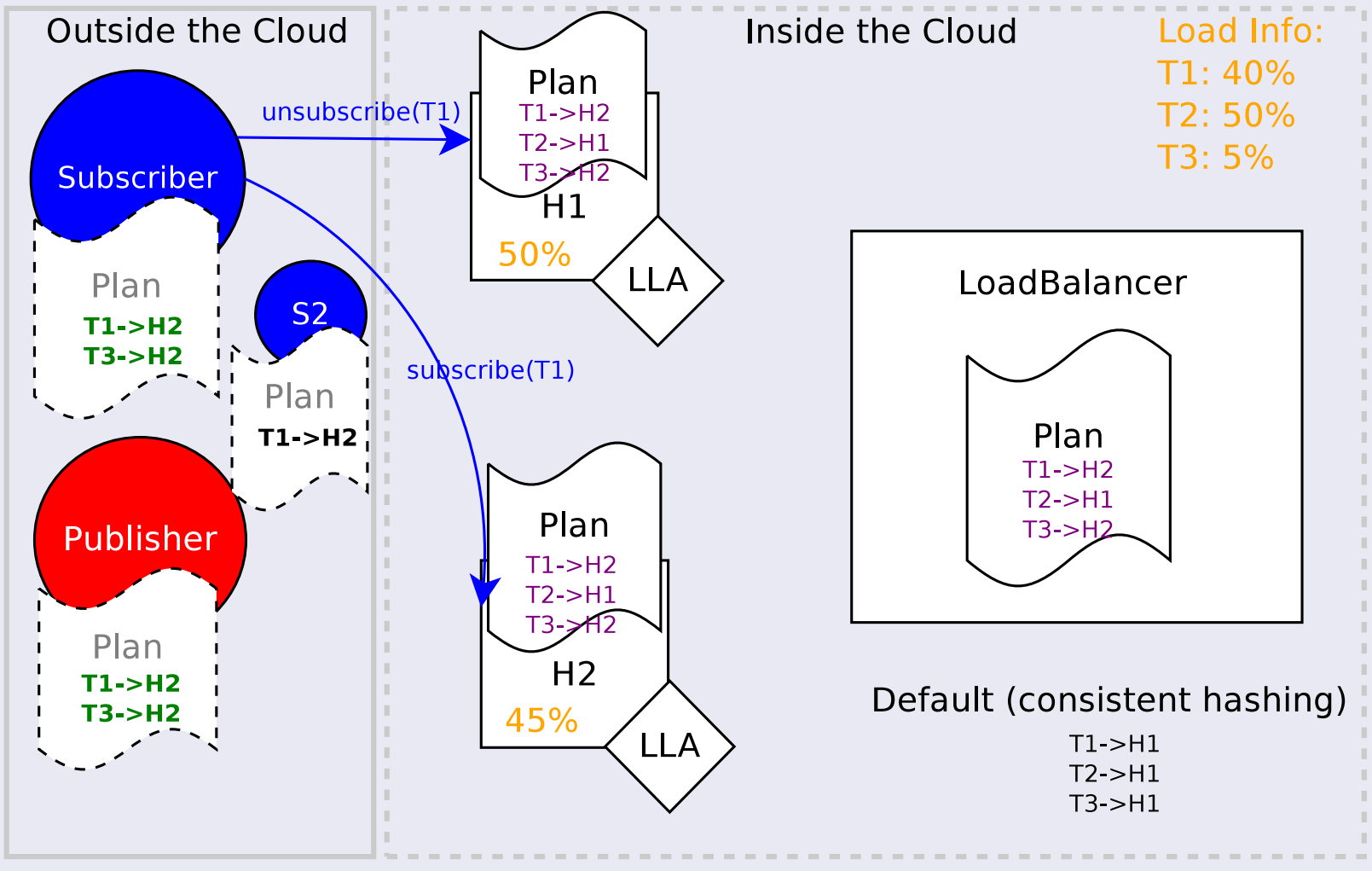
Load Balancing & Reconfiguration

Publication: client plans updated



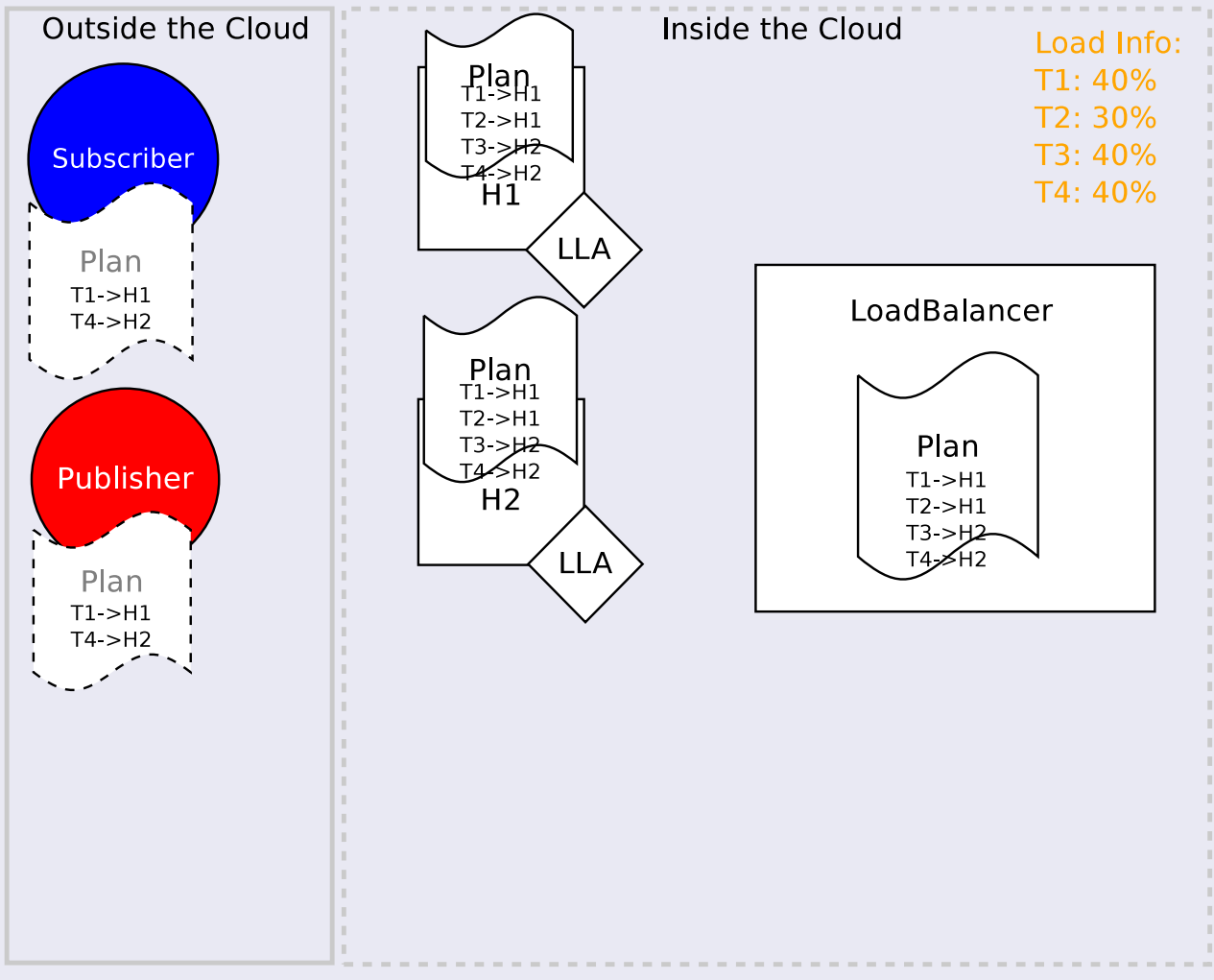
Load Balancing & Reconfiguration

Subscribing to new server and unsubscribing from old server



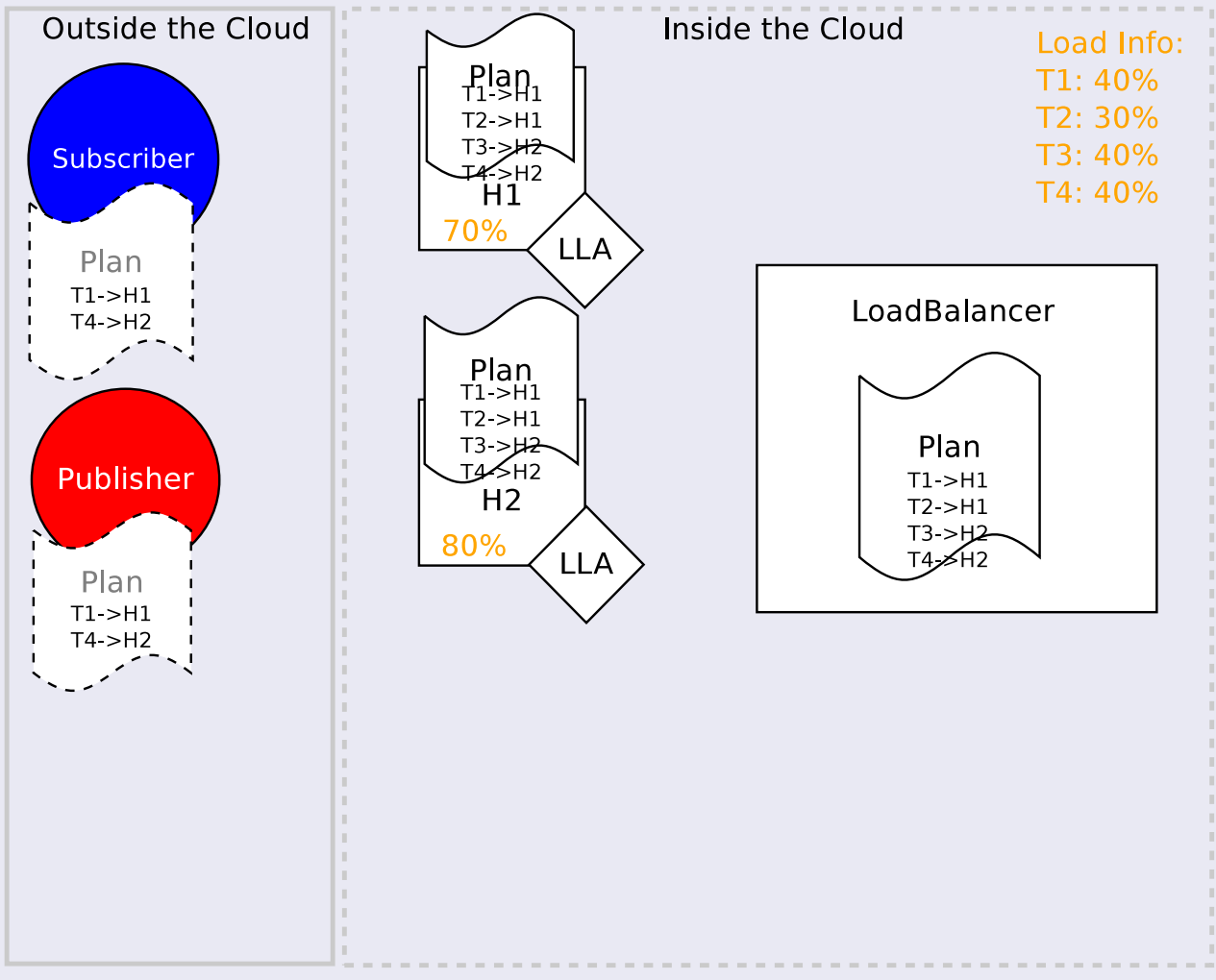
Load Balancing - Adding a new server

Initial conditions



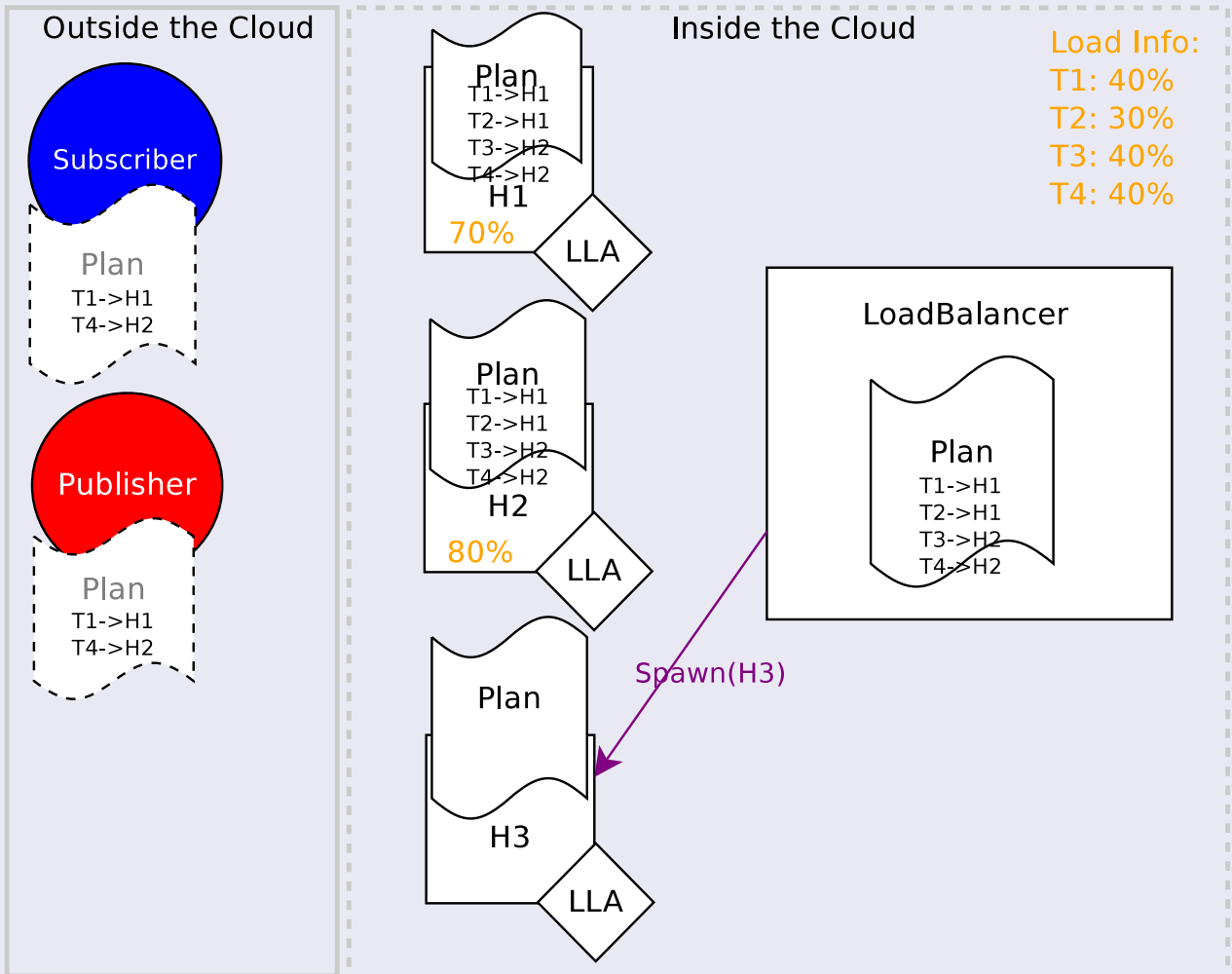
Load Balancing - Adding a new server

Initial conditions



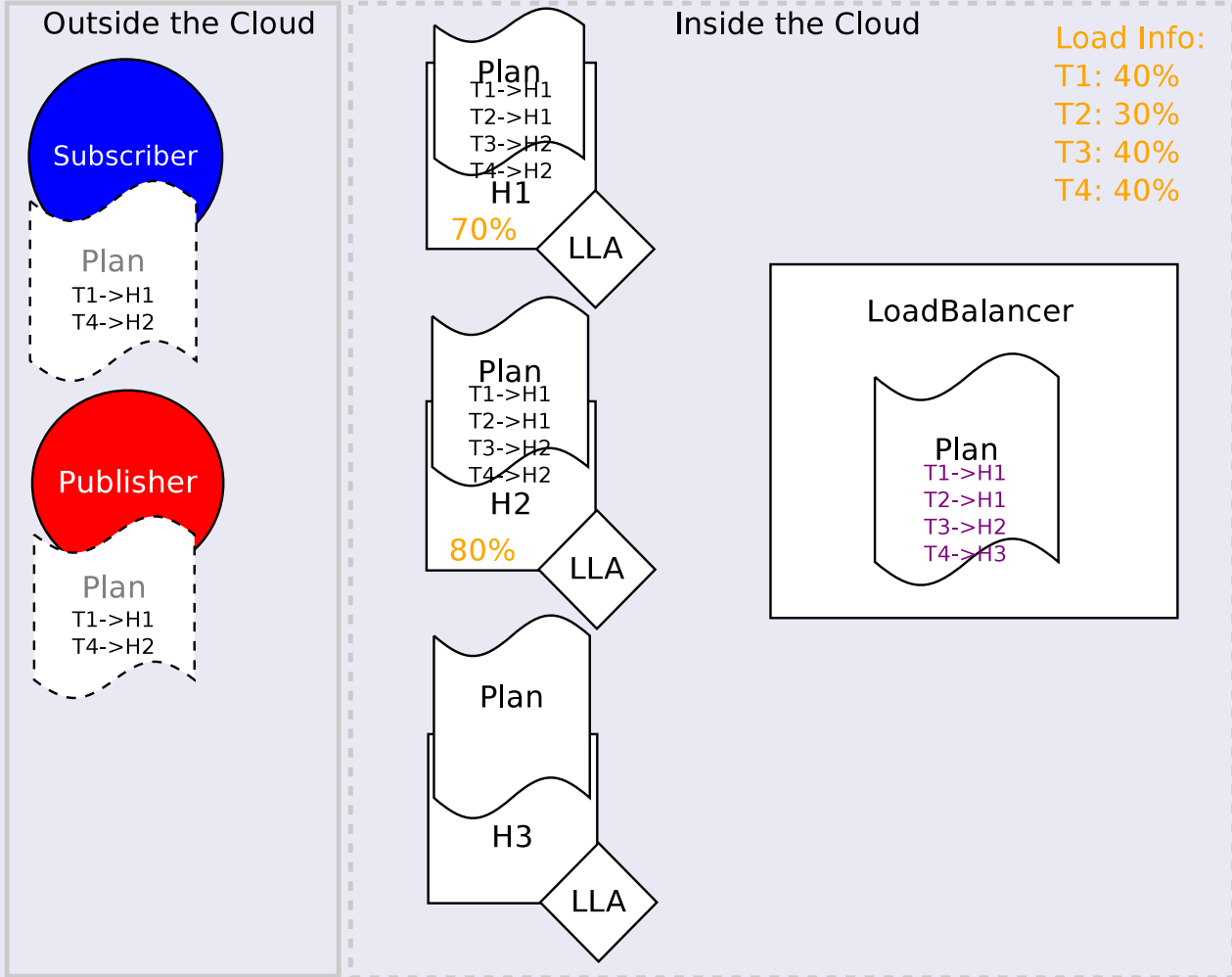
Load Balancing - Adding a new server

Spawning a new server



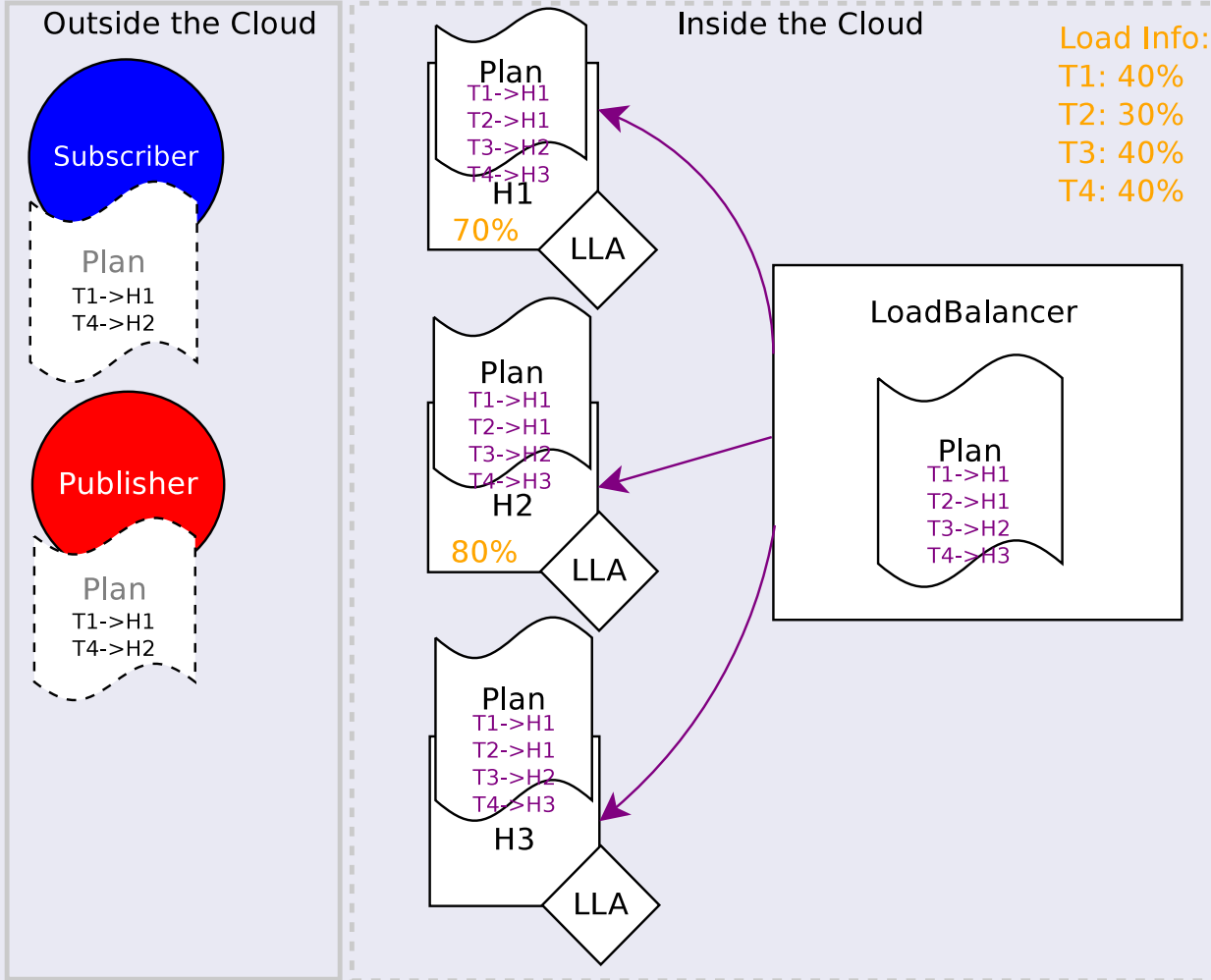
Load Balancing - Adding a new server

Generating a new plan



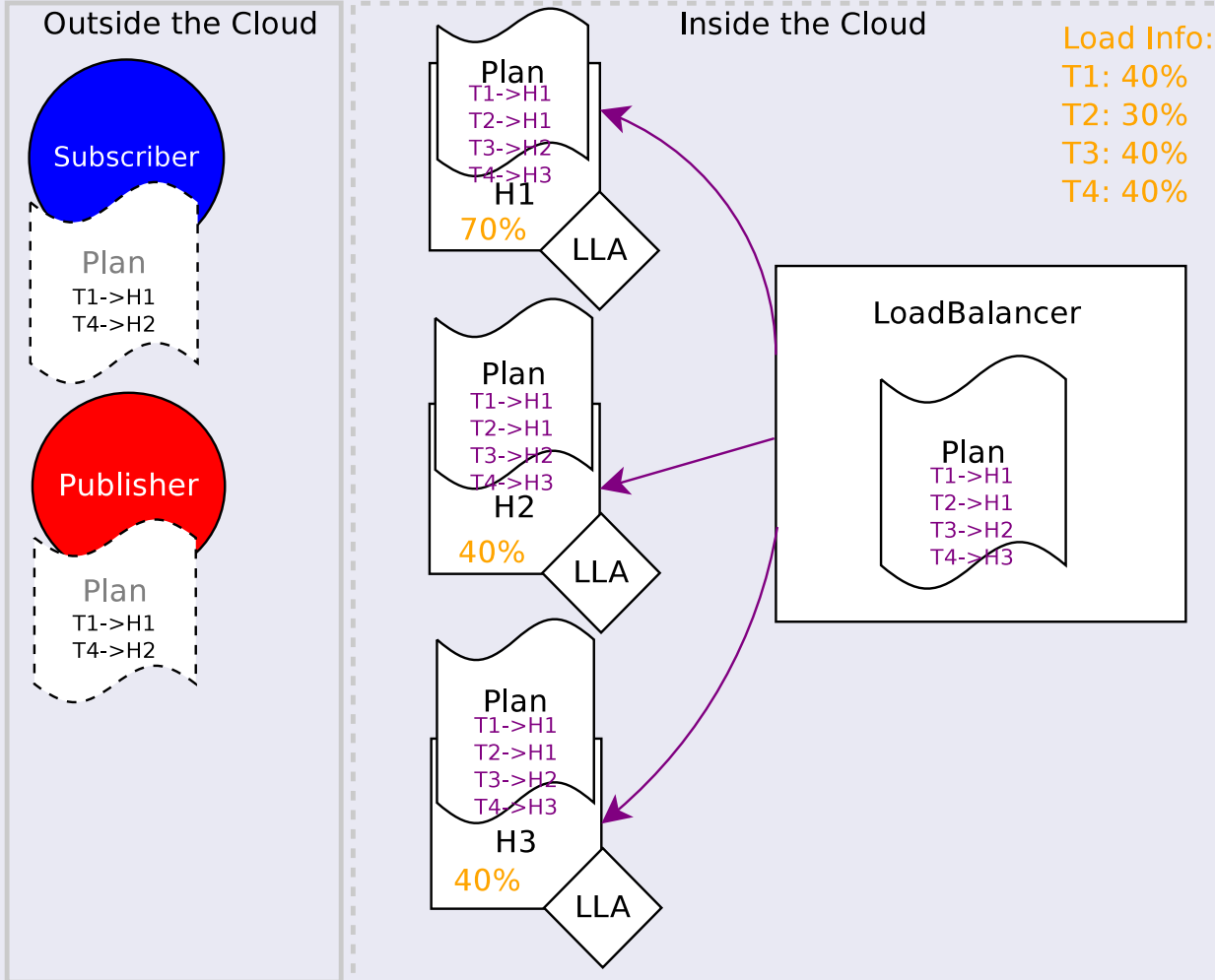
Load Balancing - Adding a new server

Forwarding new plan to all servers

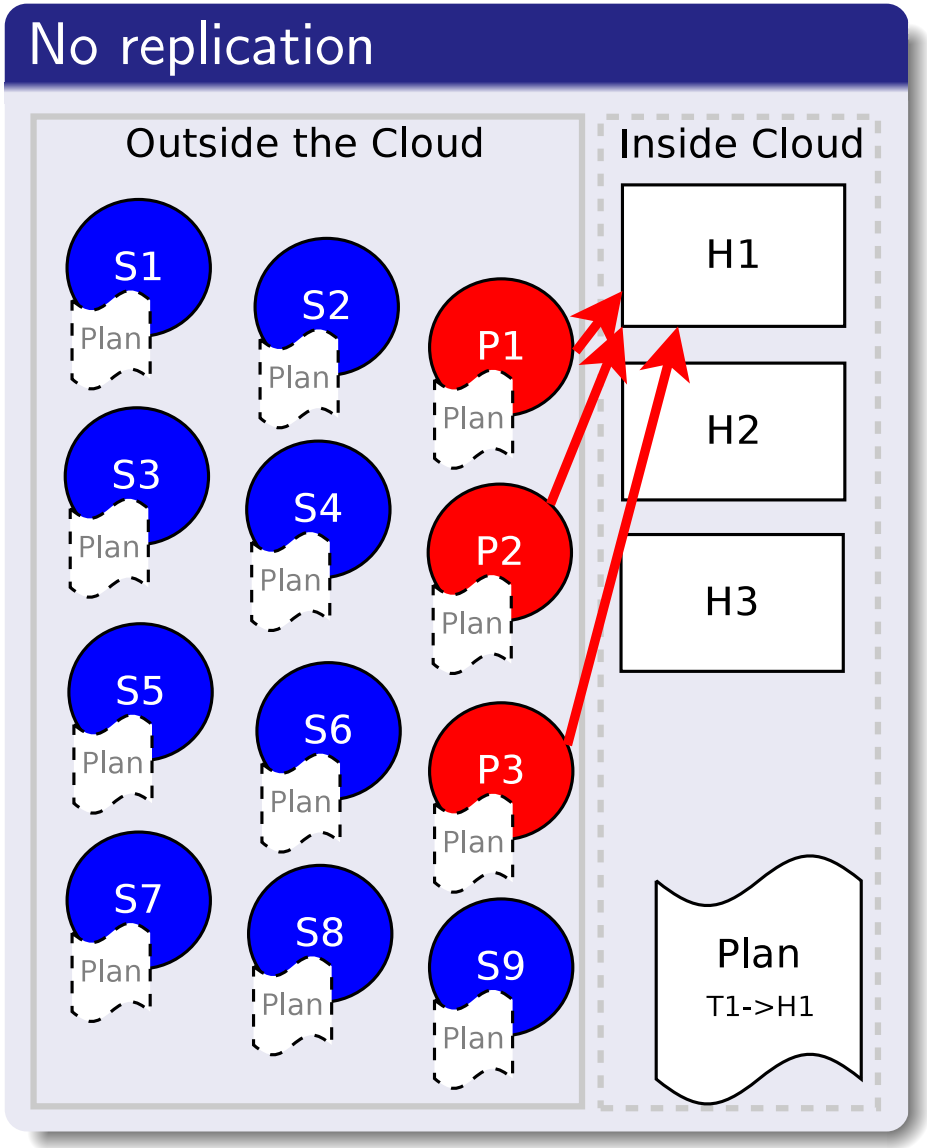


Load Balancing - Adding a new server

Forwarding new plan to all servers

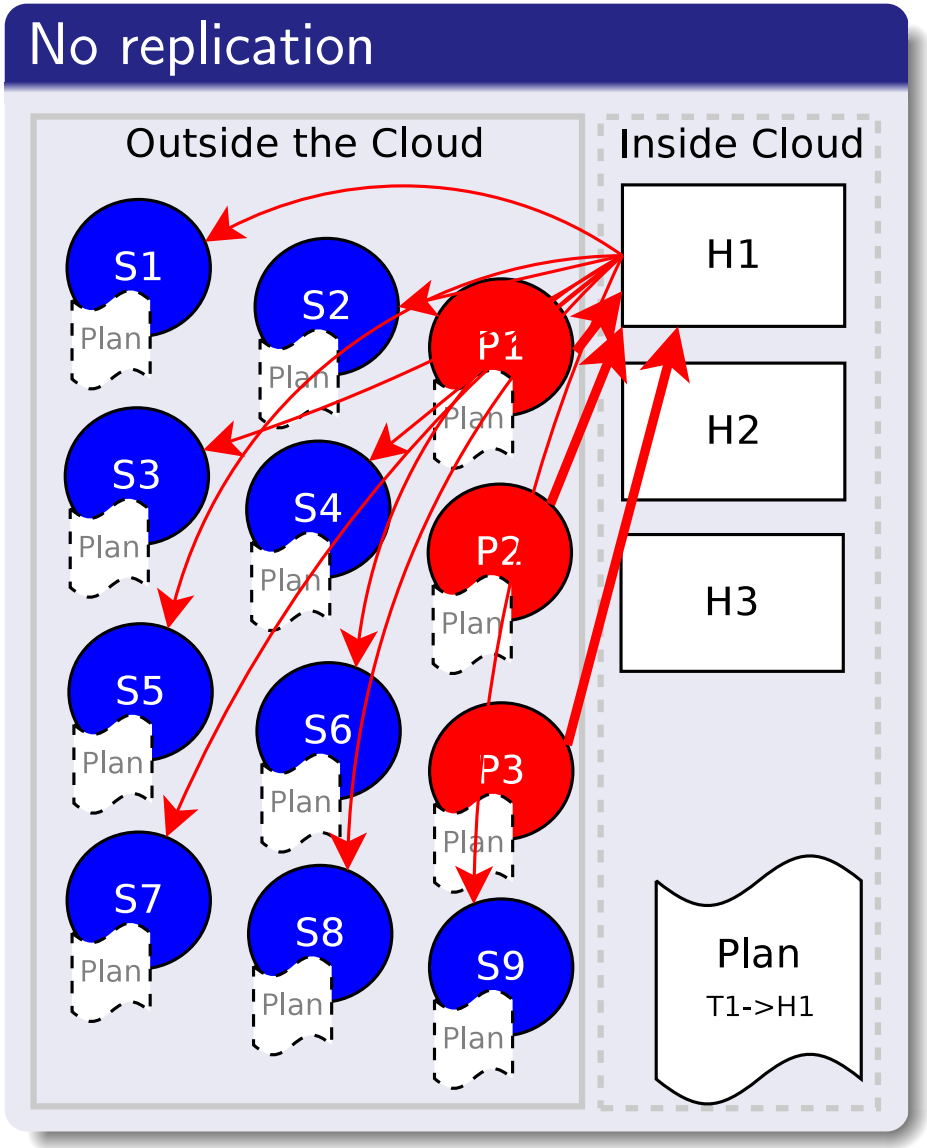


Channel Replication



- Channel T_1 is handled by only one server (H_1)
- All subscribers and publishers use H_1 for channel T_1
- What happens if the load cannot be handled by only one server?

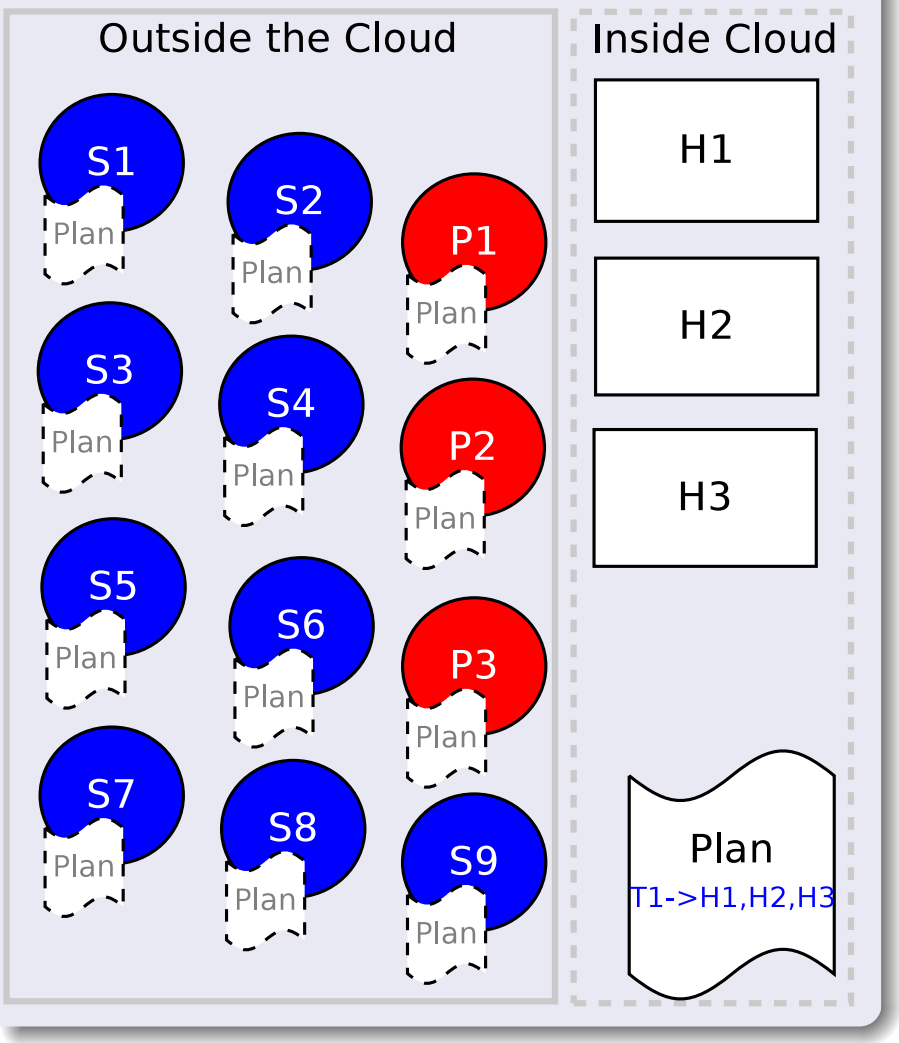
Channel Replication



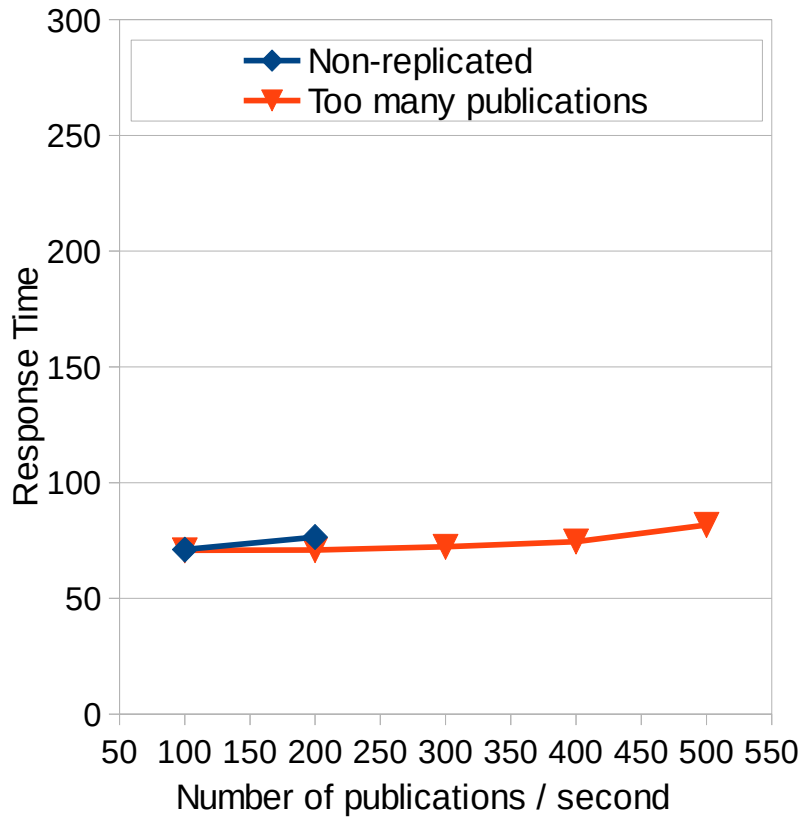
- Channel T_1 is handled by only one server (H_1)
- All subscribers and publishers use H_1 for channel T_1
- What happens if the load cannot be handled by only one server?

Channel Replication

Repl.: Too Many Publications

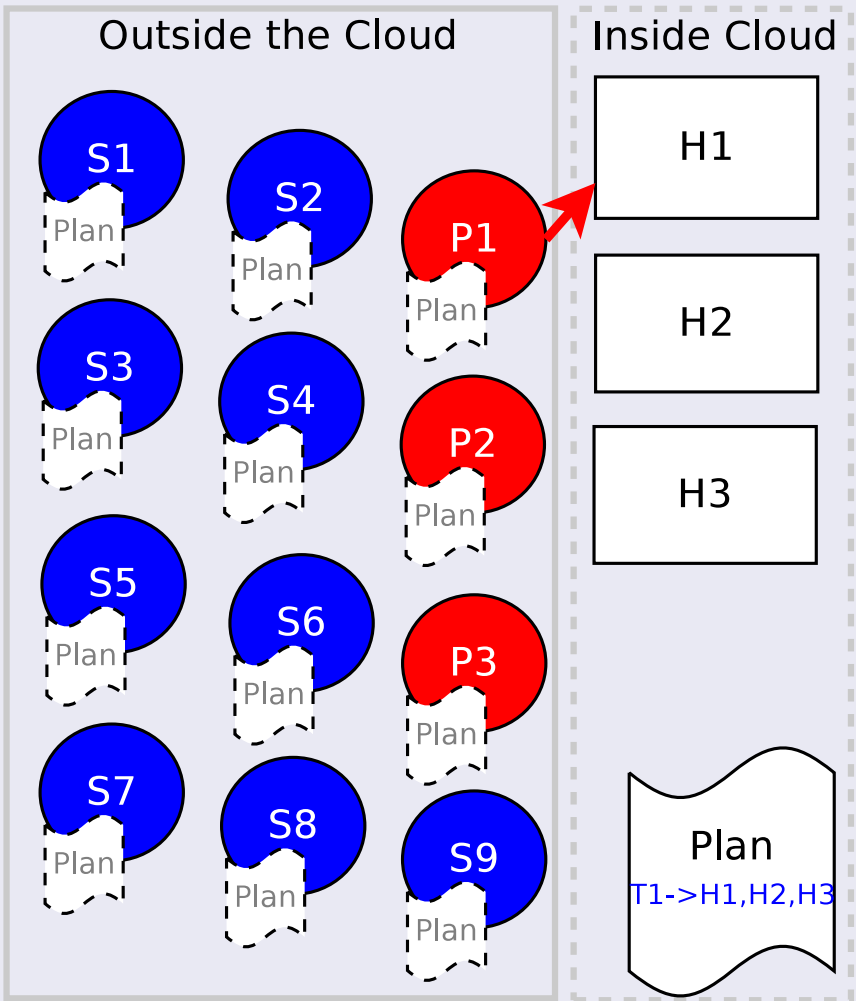


- Publishers *publish to one server*
- Subscribers *subscribe to all servers*

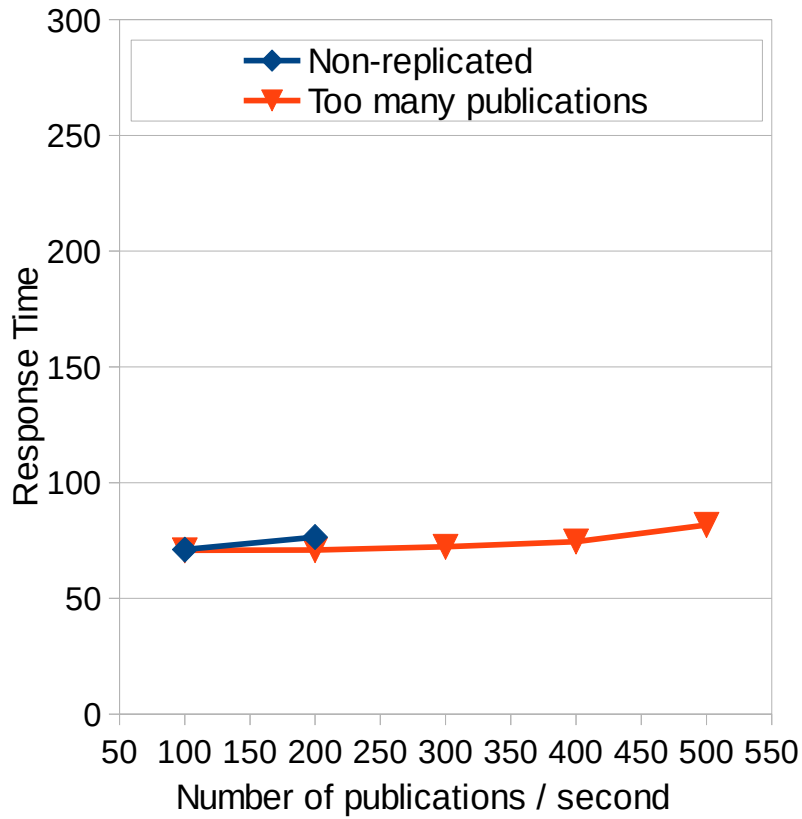


Channel Replication

Repl.: Too Many Publications

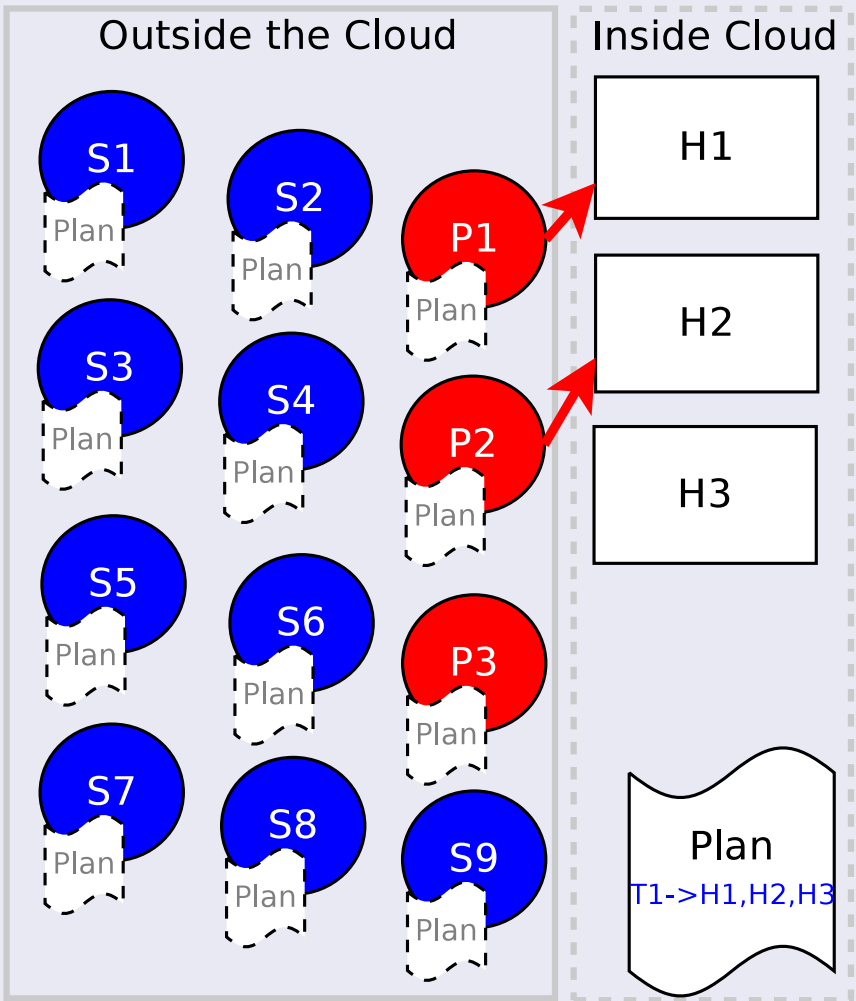


- Publishers *publish to one server*
- Subscribers *subscribe to all servers*

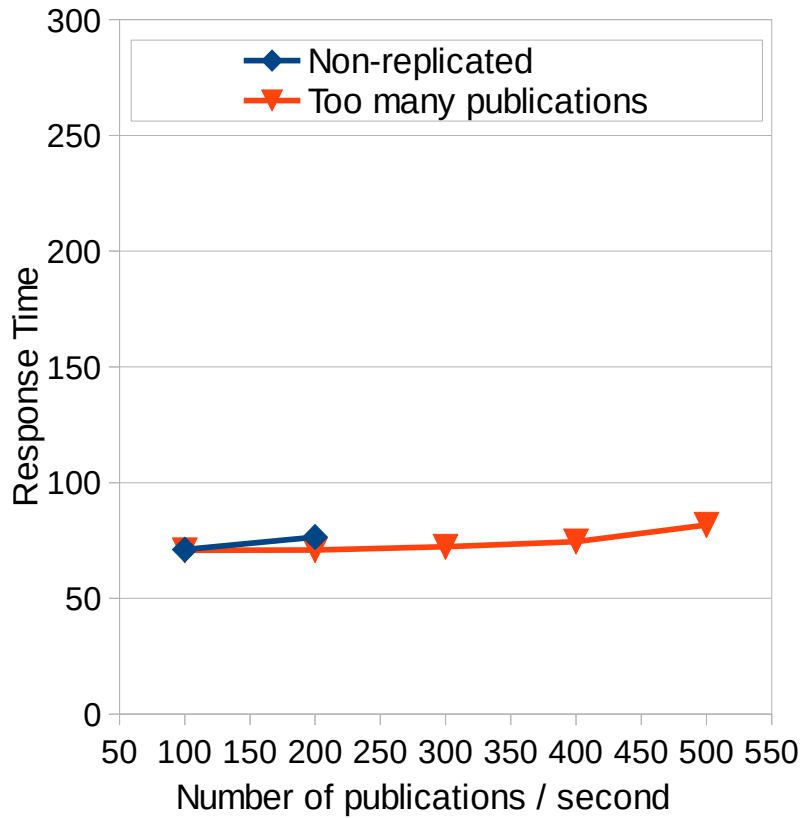


Channel Replication

Repl.: Too Many Publications

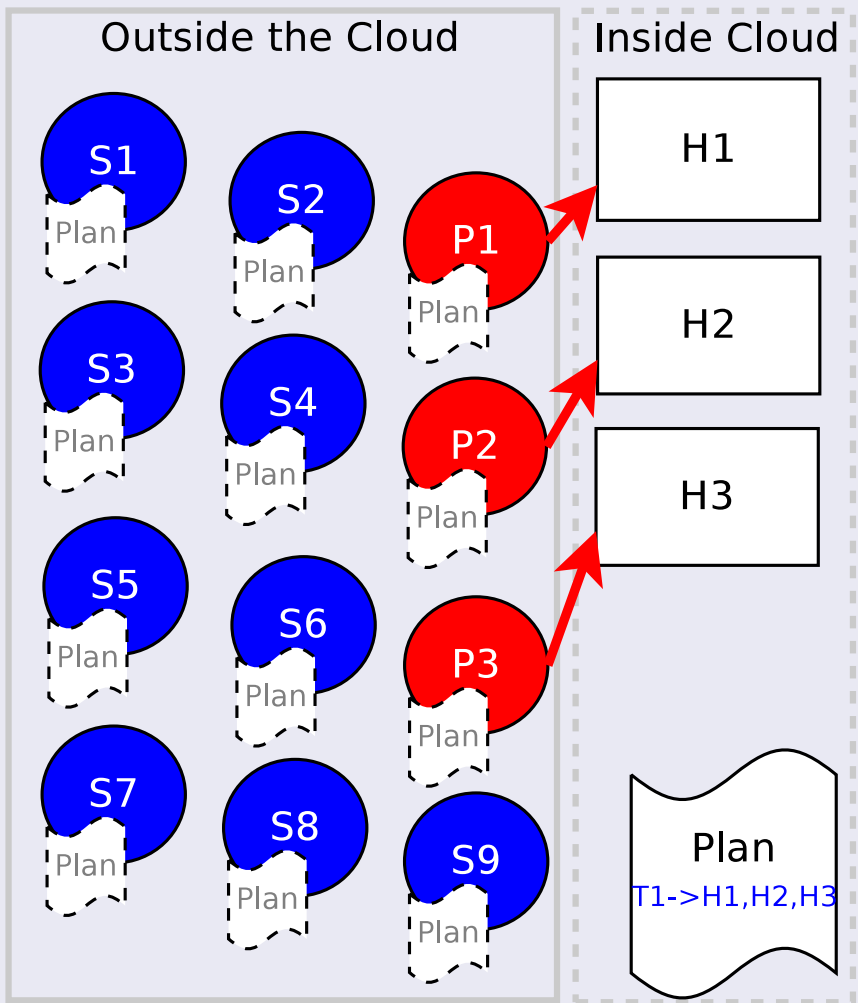


- Publishers *publish to one server*
- Subscribers *subscribe to all servers*

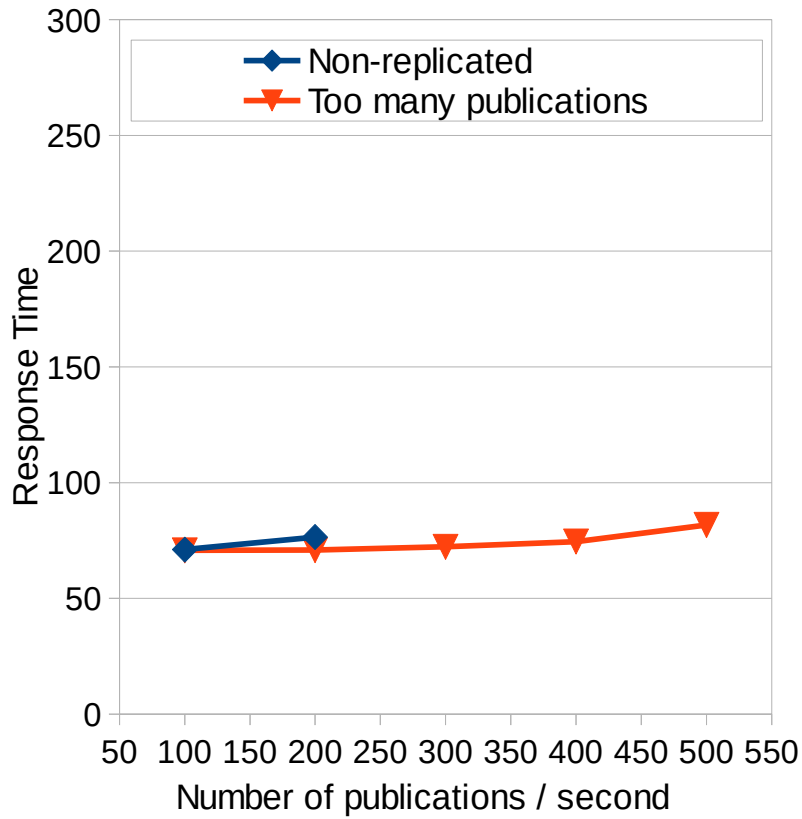


Channel Replication

Repl.: Too Many Publications

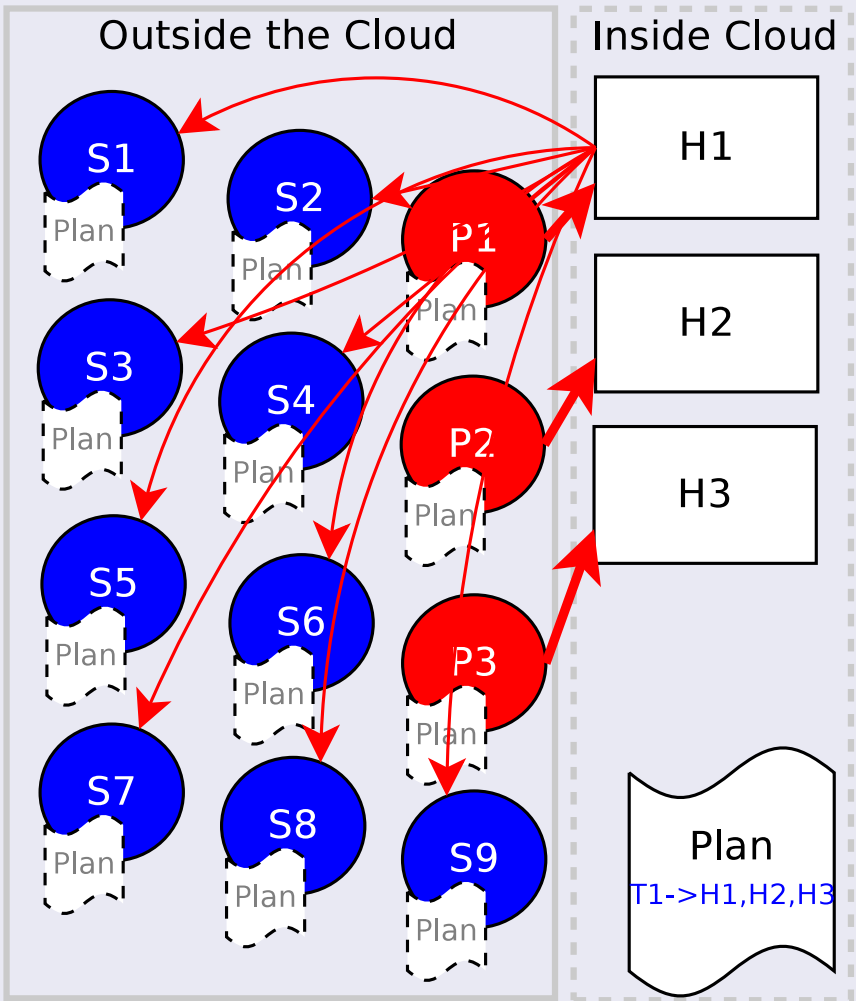


- Publishers *publish to one server*
- Subscribers *subscribe to all servers*

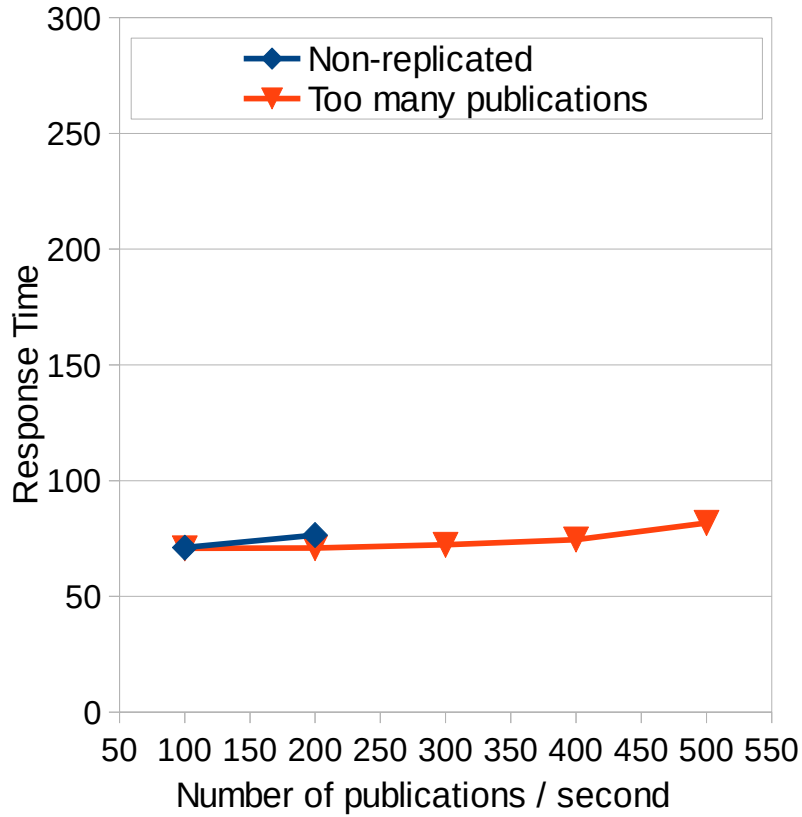


Channel Replication

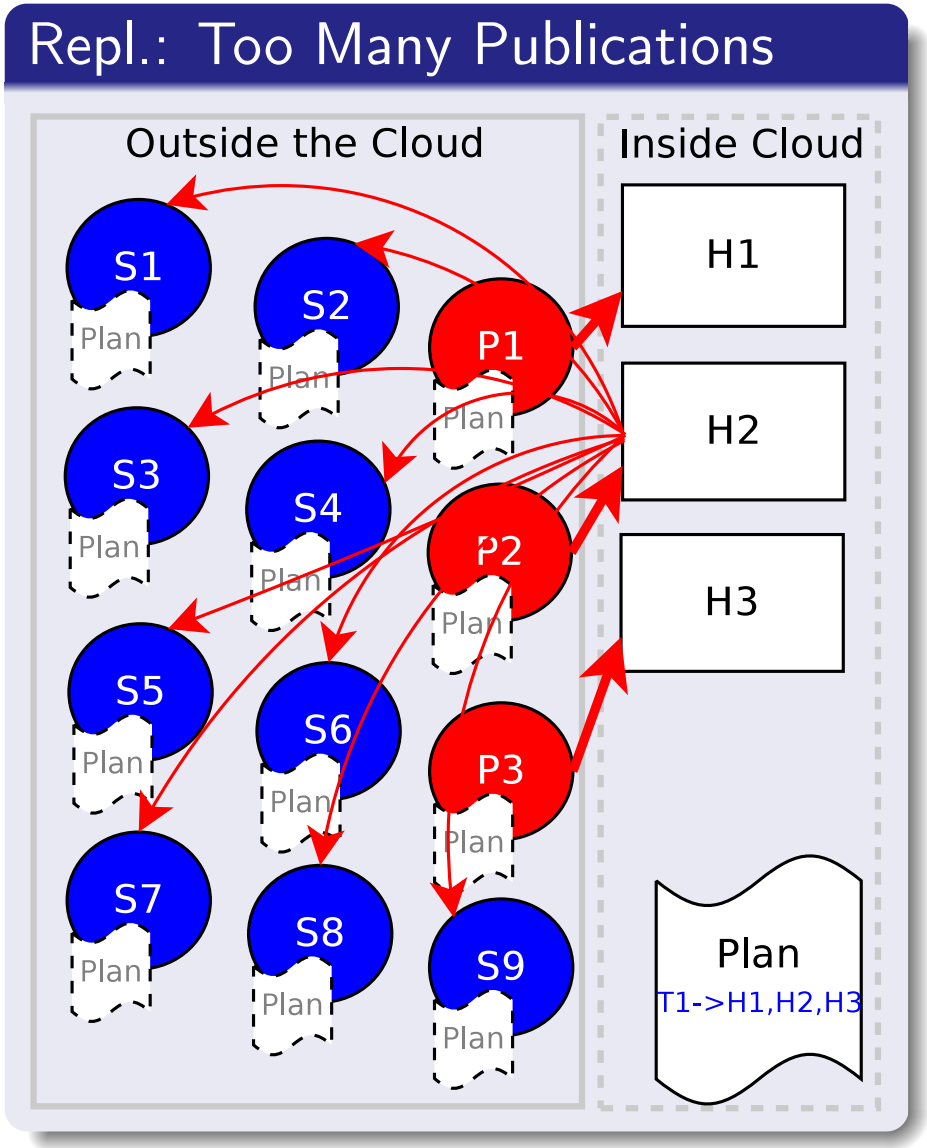
Repl.: Too Many Publications



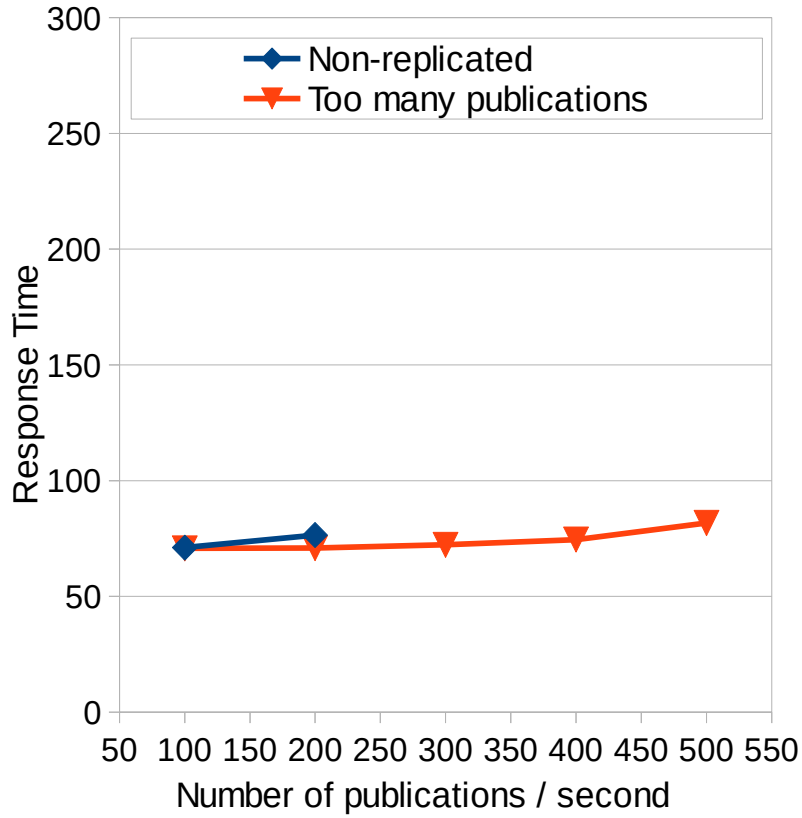
- Publishers *publish to one server*
- Subscribers *subscribe to all servers*



Channel Replication

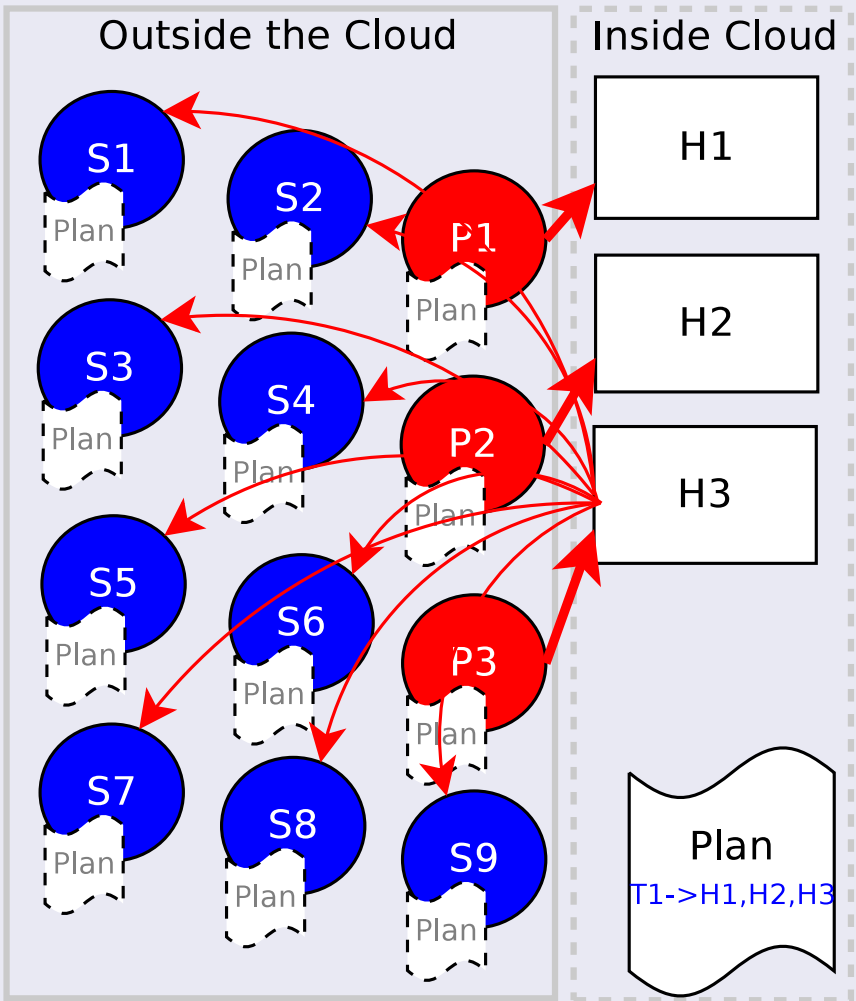


- Publishers *publish to one server*
- Subscribers *subscribe to all servers*

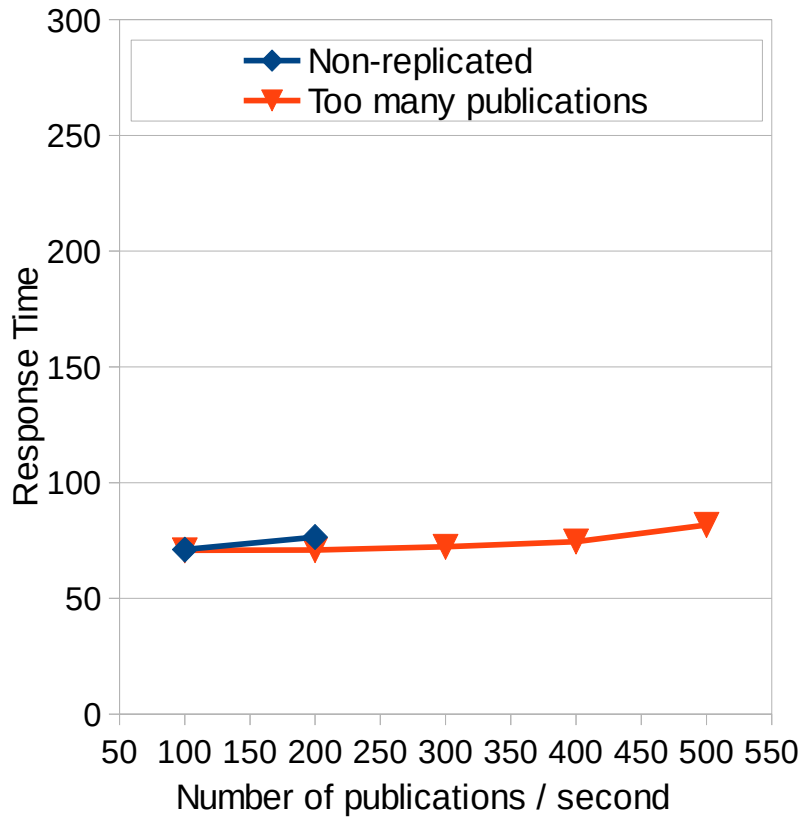


Channel Replication

Repl.: Too Many Publications

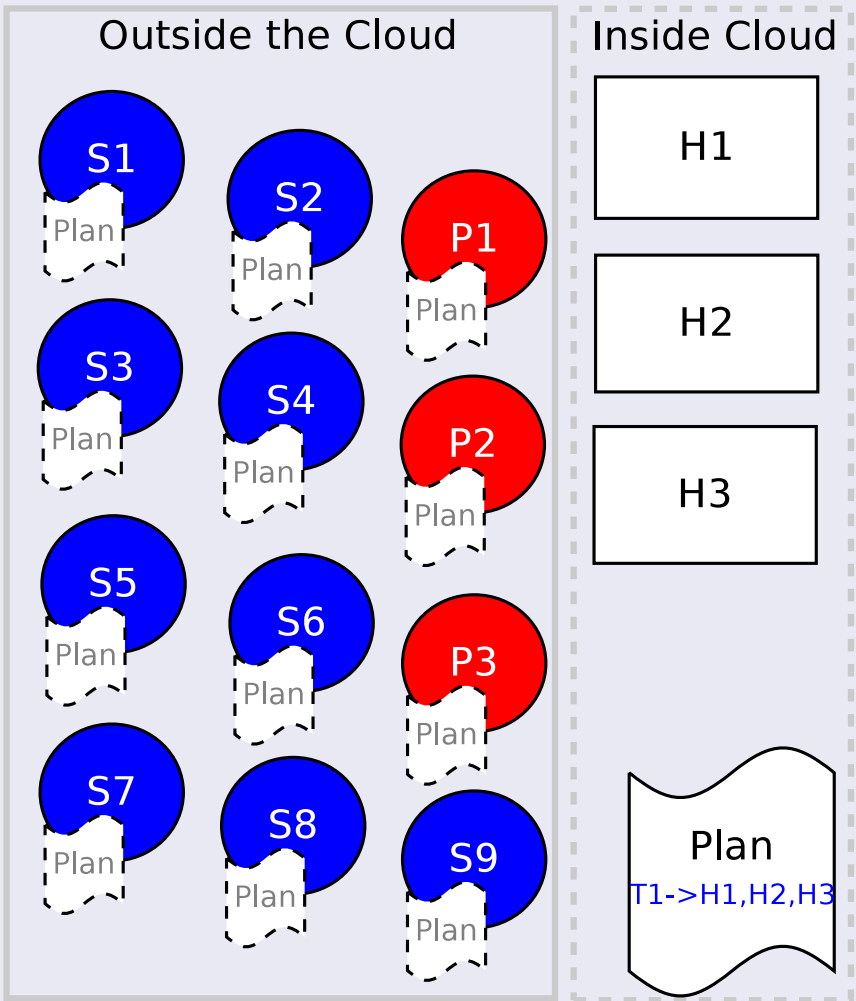


- Publishers *publish to one server*
- Subscribers *subscribe to all servers*

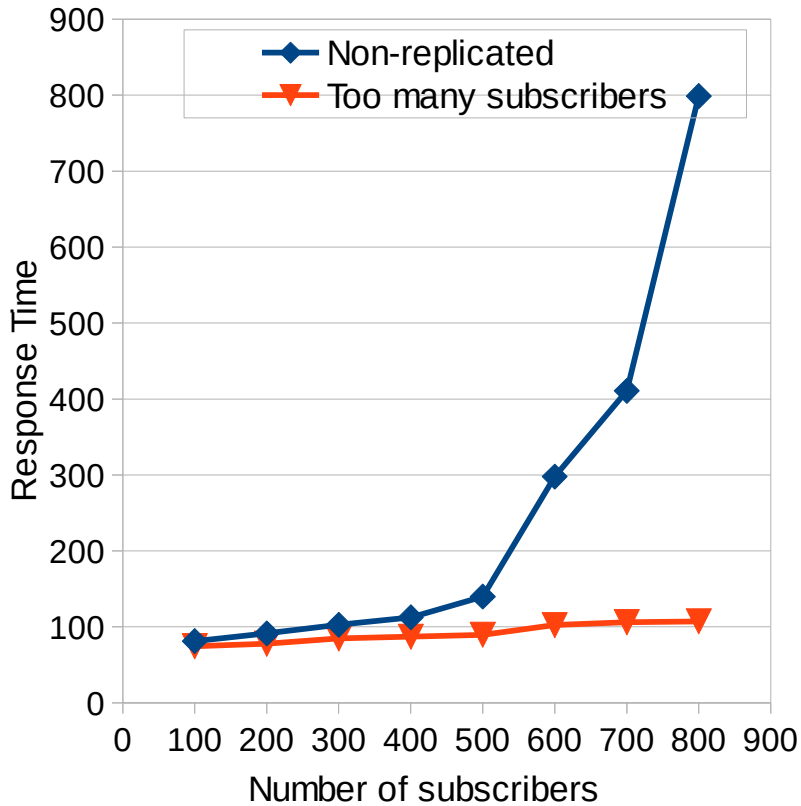


Channel Replication

Repl.: Too Many Subscribers

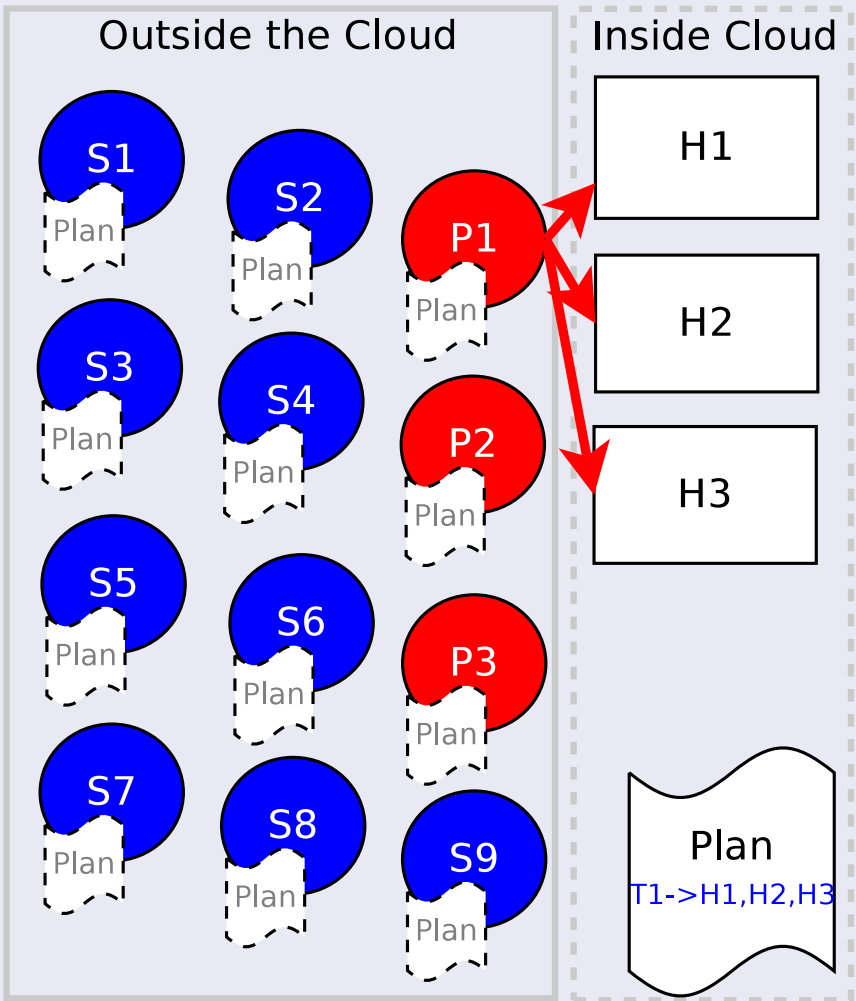


- Publishers *publish to all servers*
- Subscribers *subscribe to one server*

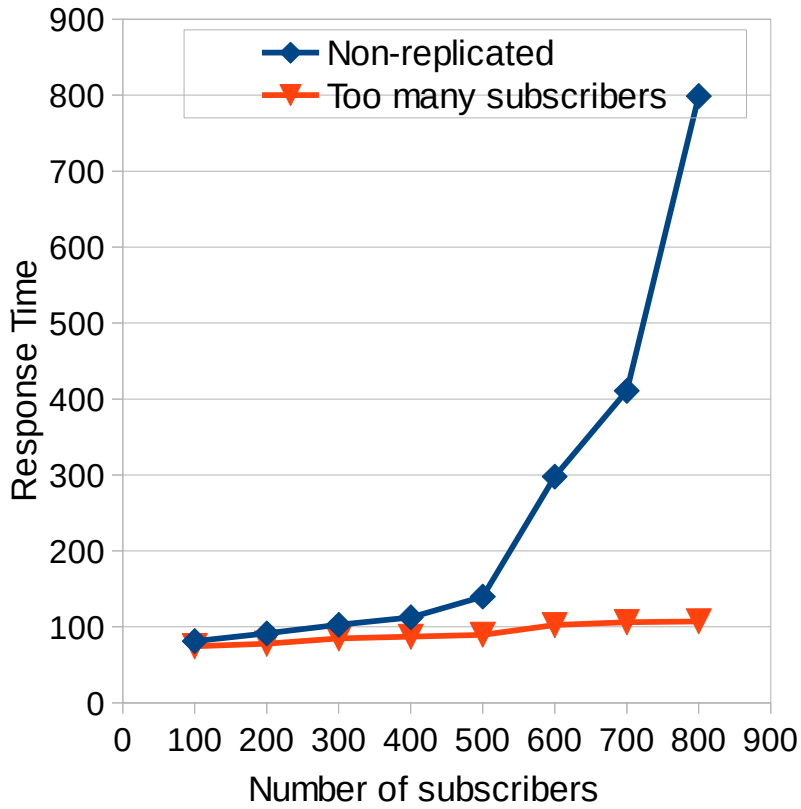


Channel Replication

Repl.: Too Many Subscribers

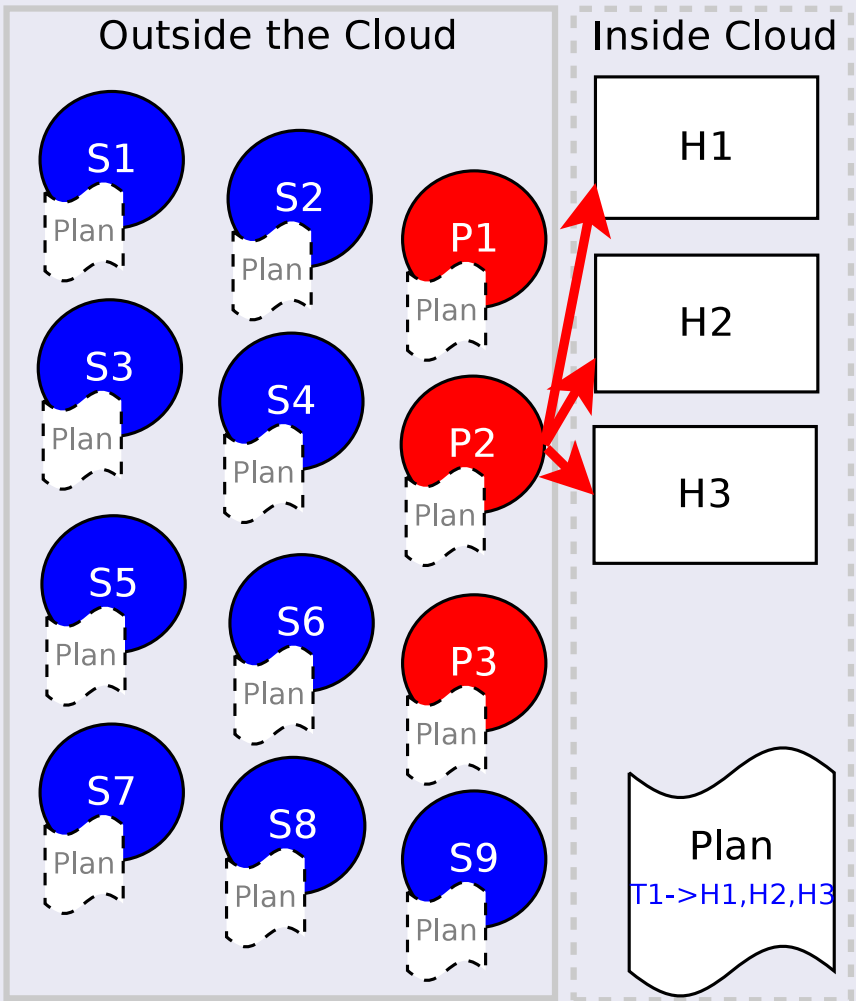


- Publishers *publish to all servers*
- Subscribers *subscribe to one server*

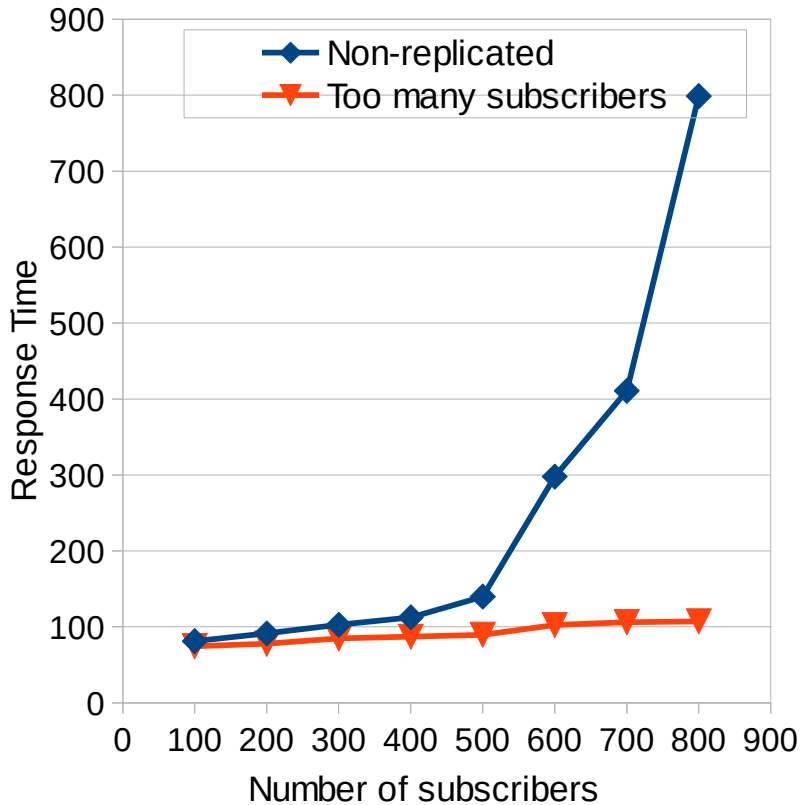


Channel Replication

Repl.: Too Many Subscribers

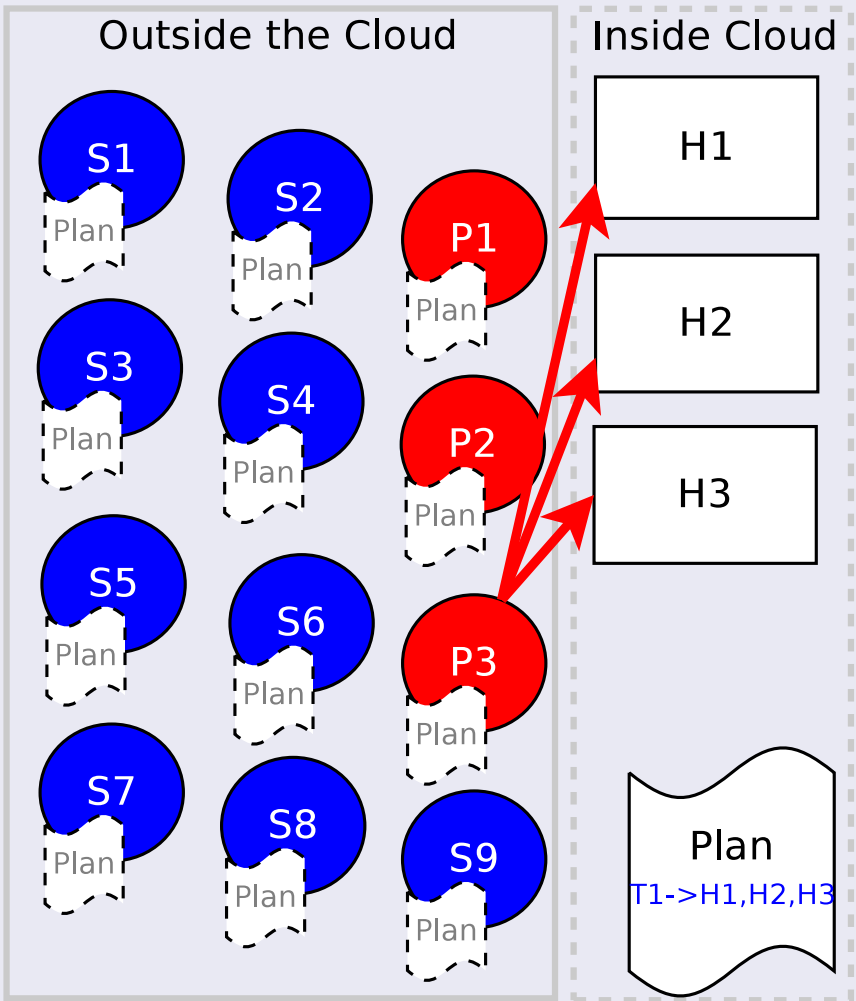


- Publishers *publish to all servers*
- Subscribers *subscribe to one server*

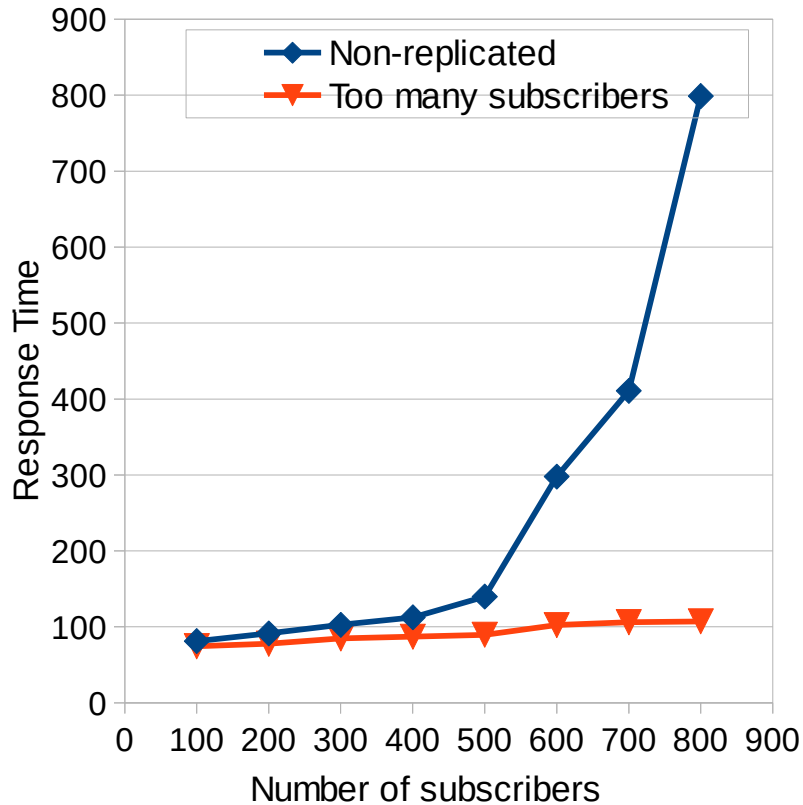


Channel Replication

Repl.: Too Many Subscribers

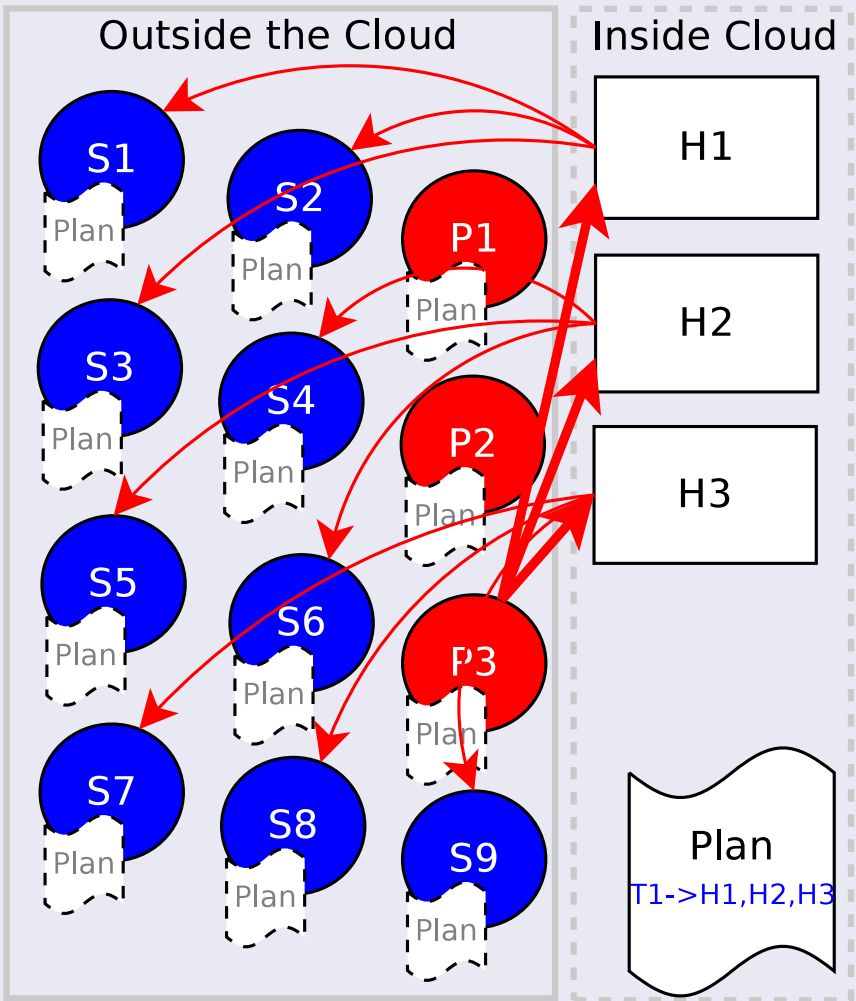


- Publishers *publish to all servers*
- Subscribers *subscribe to one server*

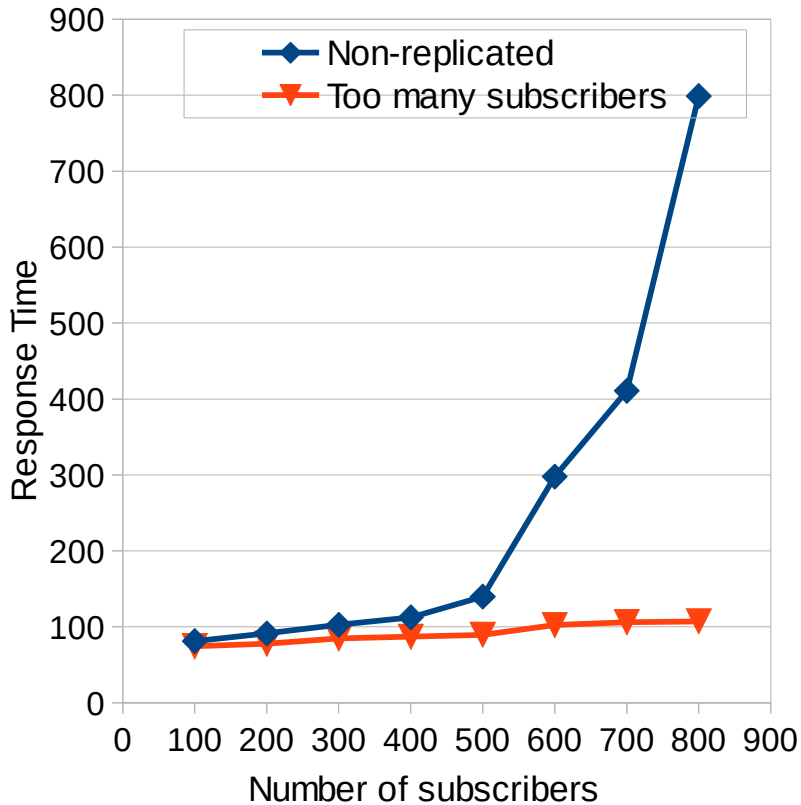


Channel Replication

Repl.: Too Many Subscribers



- Publishers *publish to all servers*
- Subscribers *subscribe to one server*



Load Balancing Algorithmic Model

Load Balancing Algorithm Outline:

1. Channel-Level Load Balancing: check whether any channel should be replicated
    1. Too many publishers / too many subscribers / no replication
    2. If already replicated: increase / decrease number of replicas?
2. System-Level Load Balancing

System-Level Load Balancing

1. Channel-Level Load Balancing
2. System-Level Load Balancing
    1. Servers overloaded?

Load Ratio

$$LR_i = M_i / T_i$$

- M_i: measured outgoing bandwidth of server
- T_i: maximum outgoing bandwidth supported by the server

System-Level Load Balancing

1. Channel-Level Load Balancing
2. System-Level Load Balancing
    1. Servers overloaded?
        1. Migrate channels until Load Ratio $<$ threshold (80%) for all servers

Load Ratio

$$LR_i = M_i / T_i$$

- M_i: measured outgoing bandwidth of server
- T_i: maximum outgoing bandwidth supported by the server

System-Level Load Balancing

1. Channel-Level Load Balancing
2. System-Level Load Balancing
    1. Servers overloaded?
        1. Migrate channels until Load Ratio $<$ threshold (80%) for all servers
        2. If needed: spawn additional servers

Load Ratio

$$LR_i = M_i / T_i$$

- M_i: measured outgoing bandwidth of server
- T_i: maximum outgoing bandwidth supported by the server

System-Level Load Balancing

1. Channel-Level Load Balancing
2. System-Level Load Balancing
    1. Servers overloaded?
        1. Migrate channels until Load Ratio $<$ threshold (80%) for all servers
        2. If needed: spawn additional servers
    2. Servers underloaded (if no servers are overloaded)?

Load Ratio

$$LR_i = M_i / T_i$$

- M_i: measured outgoing bandwidth of server
- T_i: maximum outgoing bandwidth supported by the server

System-Level Load Balancing

1. Channel-Level Load Balancing
2. System-Level Load Balancing
    1. Servers overloaded?
        1. Migrate channels until Load Ratio $<$ threshold (80%) for all servers
        2. If needed: spawn additional servers
    2. Servers underloaded (if no servers are overloaded)?
        1. If overall load $<$ a given threshold: slowly remove channels from lowest-loaded server

Load Ratio

$$LR_i = M_i / T_i$$

- M_i: measured outgoing bandwidth of server
- T_i: maximum outgoing bandwidth supported by the server

System-Level Load Balancing

1. Channel-Level Load Balancing
2. System-Level Load Balancing
   1. Servers overloaded?
      1. Migrate channels until Load Ratio $<$ threshold (80%) for all servers
      2. If needed: spawn additional servers
   2. Servers underloaded (if no servers are overloaded)?
      1. If overall load $<$ a given threshold: slowly remove channels from lowest-loaded server
      2. When load of lowest-loaded server reaches 0, despawn it

Load Ratio

$$LR_i = M_i / T_i$$

- M_i: measured outgoing bandwidth of server
- T_i: maximum outgoing bandwidth supported by the server

Experiments

Implementation & Environnement

Implementation

- Built on top of the McGill's Mammoth project
- Around 110 Java classes / 10,000 lines of code
- Uses unmodified Open-Source Redis software for pub/sub
- Experiments done over a simple game (RGame)
  - Large volume of subscriptions and publications

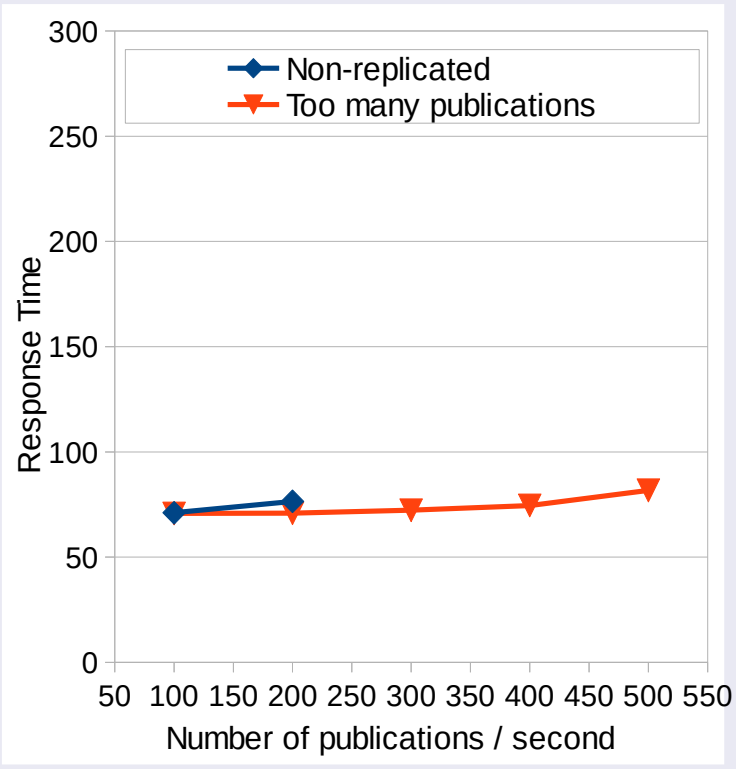
Experimental Setup

Experimental Setup

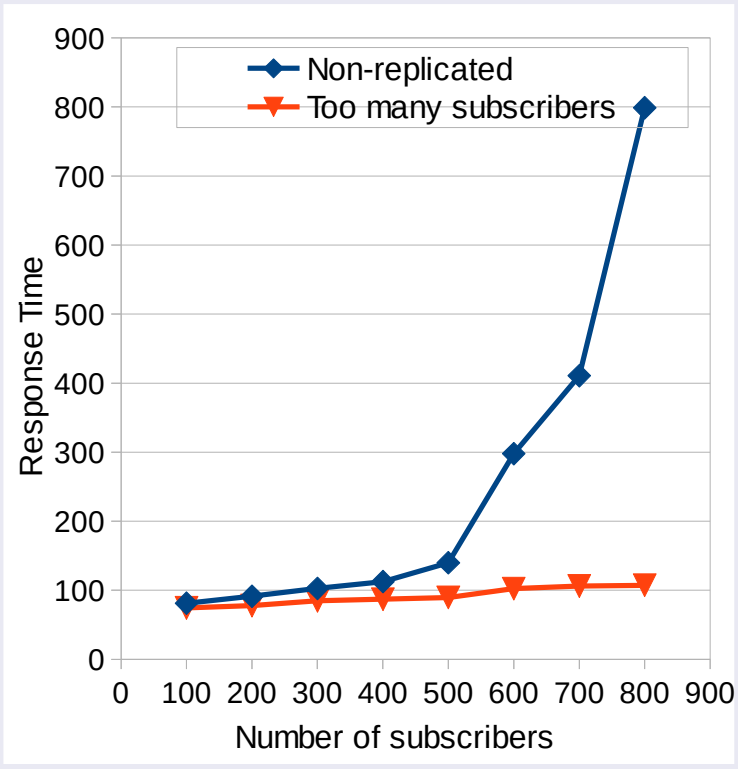
- McGill School of Computer Science lab machines (80)
  - Pub/sub servers + LLA
  - Load balancer
  - Clients - 20 clients per machine
- $>$ 1000 game clients
- Latency Emulation using King Dataset

Experiment 1 - Channel-level Scalability (Replication)

Too Many Publications

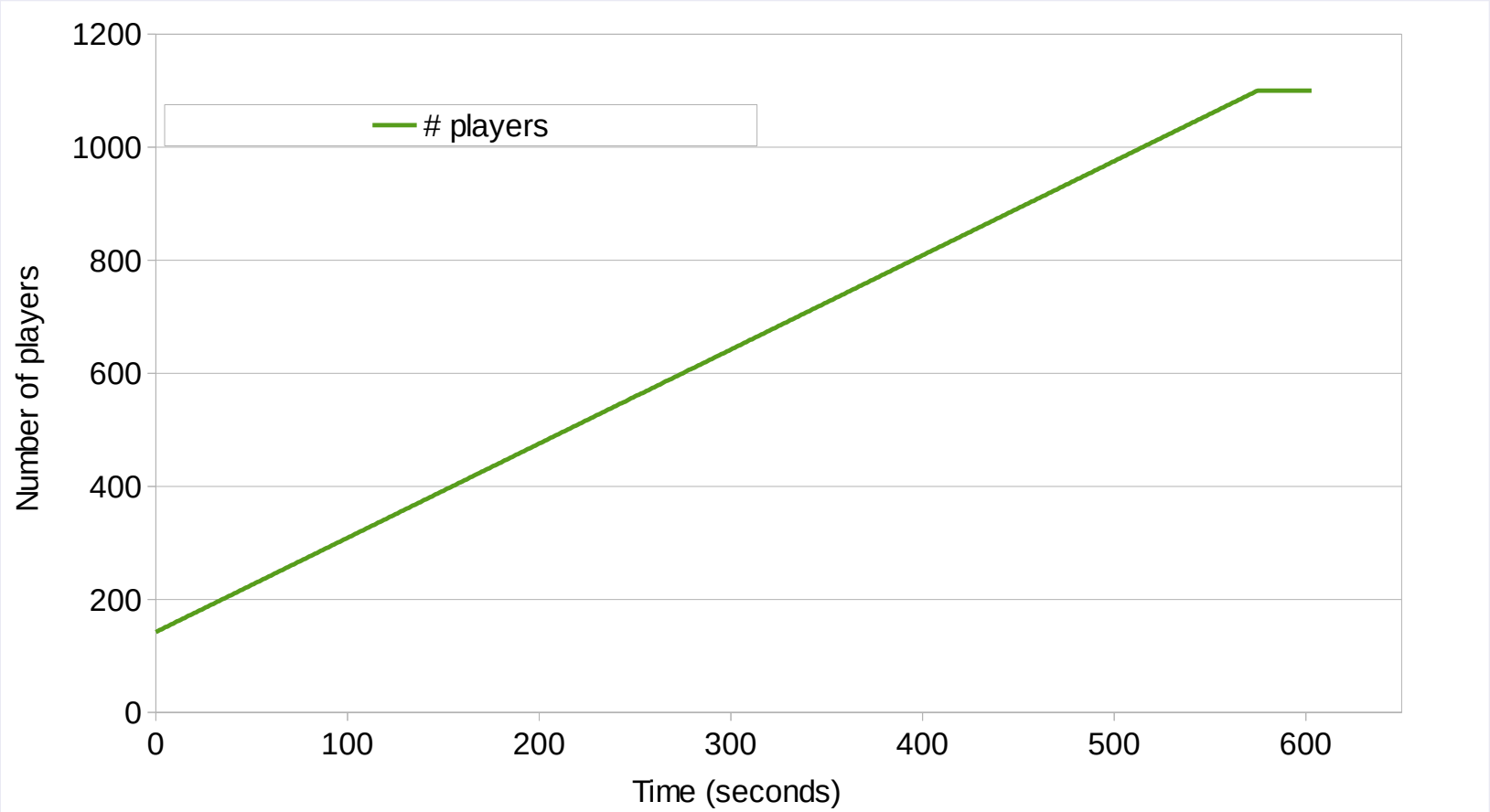


Too Many Subscribers



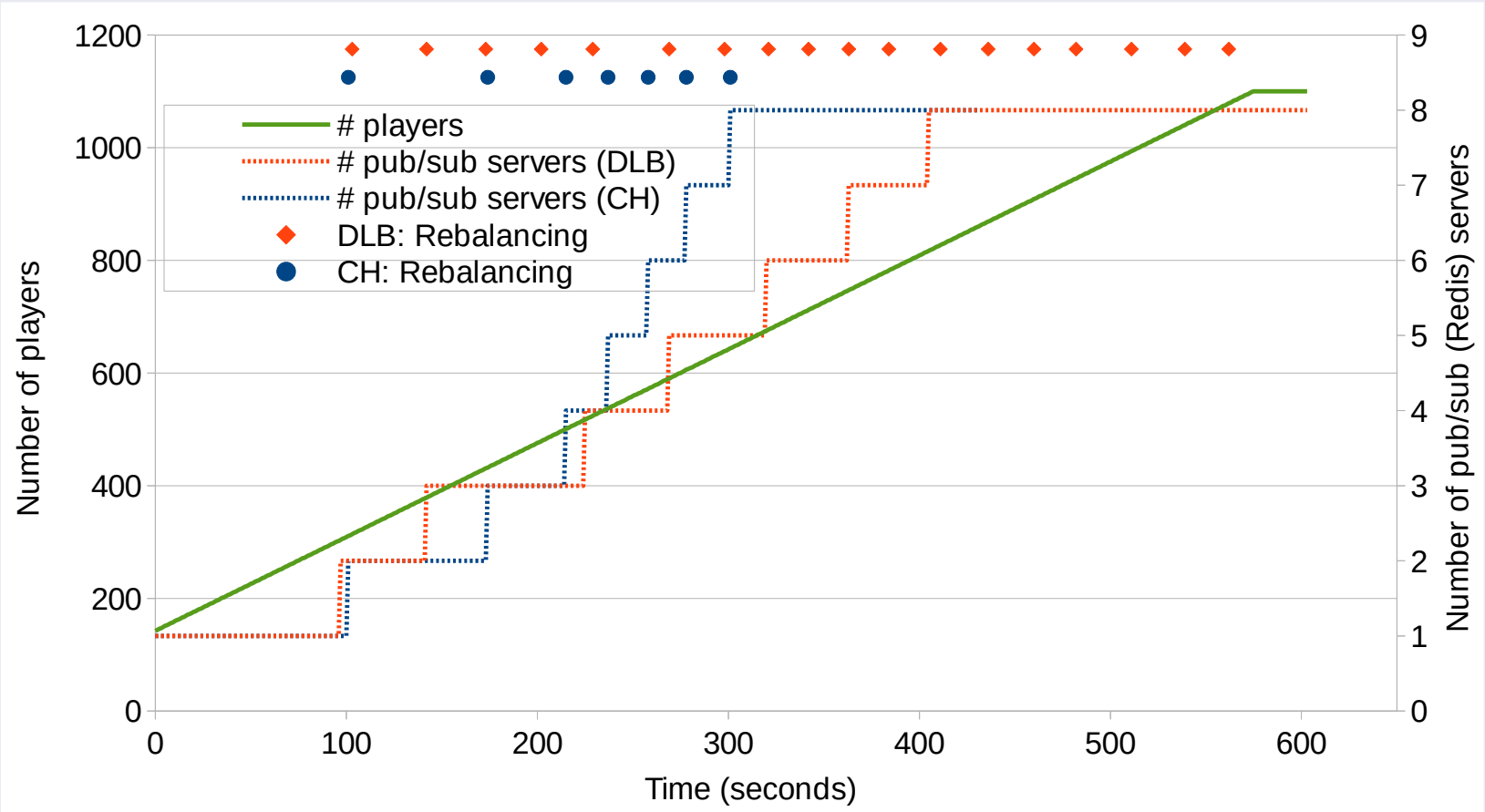
Experiment 2 - Scalability

Number of Players



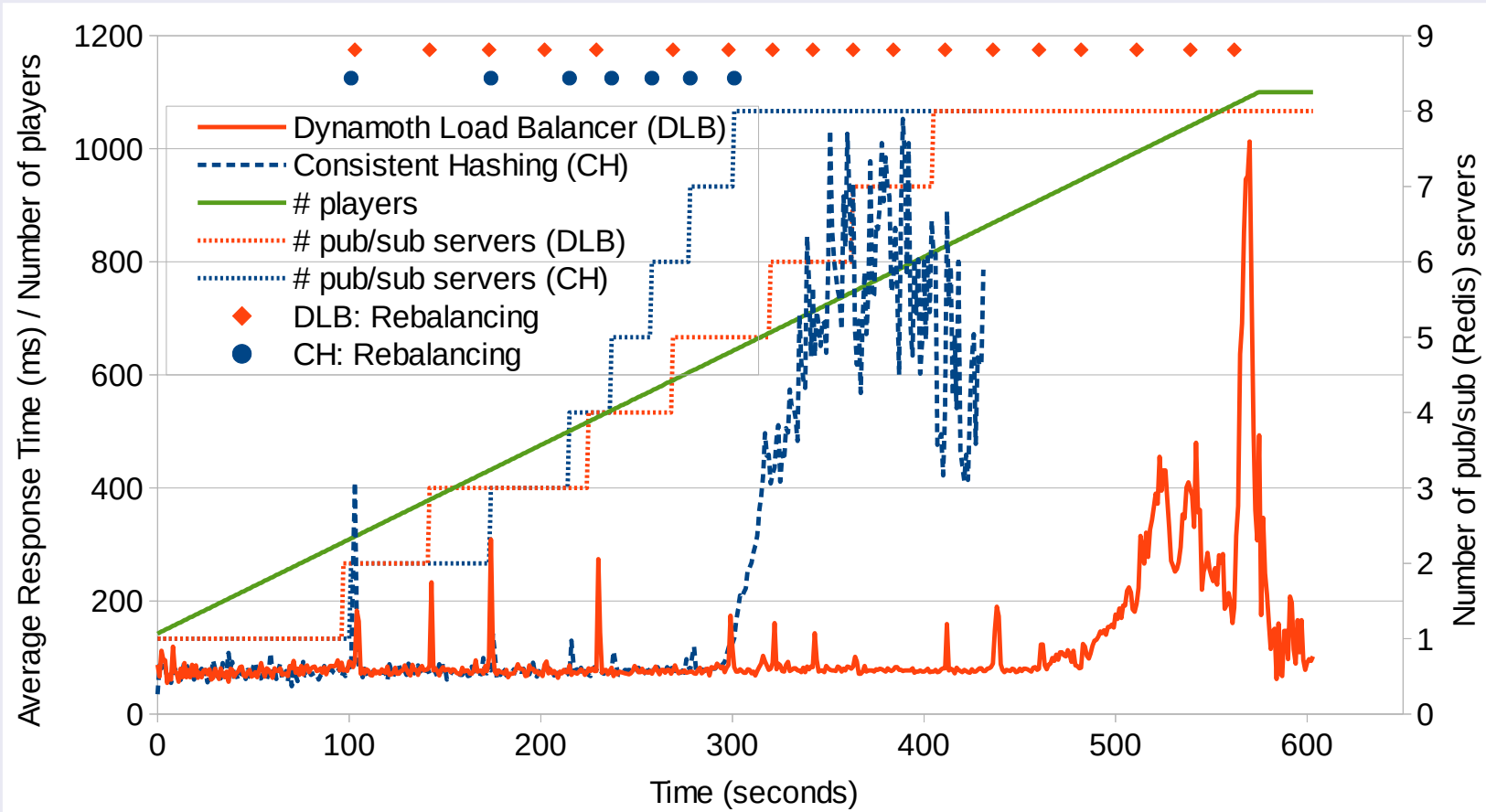
Experiment 2 - Scalability

Load Balancing



Experiment 2 - Scalability

Average Response Time



Conclusion & Future Work

Conclusion & Future Work

Conclusion:

- Service for scalable topic-based pub/sub in the Cloud
- Can handle channels with very high load patterns
- Lazy plan propagation
- Forwarding to prevent message loss while changing plans
- Uses unmodified pub/sub software (ex: Redis)

Conclusion & Future Work

Conclusion:

- Service for scalable topic-based pub/sub in the Cloud
- Can handle channels with very high load patterns
- Lazy plan propagation
- Forwarding to prevent message loss while changing plans
- Uses unmodified pub/sub software (ex: Redis)

Future Work:

- CPU load in Load Balancing (CPU constrained in Cloud environments)
- Cost model to minimize costs in the Cloud
- Reliability & Fault Tolerance
- Large-Scale real-time Wide-Area pub/sub support

Conclusion & Future Work

Thank you for your attention!

Algorithm 1 - Replication

Determining if replication should be used

begin

$P_{\text{ratio}} = \#\text{publications}/\#\text{subscribers}$ $S_{\text{ratio}} = \#\text{subscribers}/\#\text{publications}$;

if $P_{\text{ratio}} > \text{AllSubs}_{\text{threshold}}$ and $\#\text{publications} > \text{Publication}_{\text{threshold}}$ **then**

  $N_{\text{servers}} = P_{\text{ratio}}/\text{AllSubs}_{\text{threshold}}$;

  replicate(ALL_SUBSCRIBERS, N_{servers})

end

else if $S_{\text{ratio}} > \text{AllPubs}_{\text{threshold}}$ and $\#\text{subscribers} > \text{Subscriber}_{\text{threshold}}$ **then**

  $N_{\text{servers}} = S_{\text{ratio}}/\text{AllPubs}_{\text{threshold}}$;

  replicate(ALL_PUBLISHERS, N_{servers})

end

else

  replicate(NO_REPLICATION)

end

end

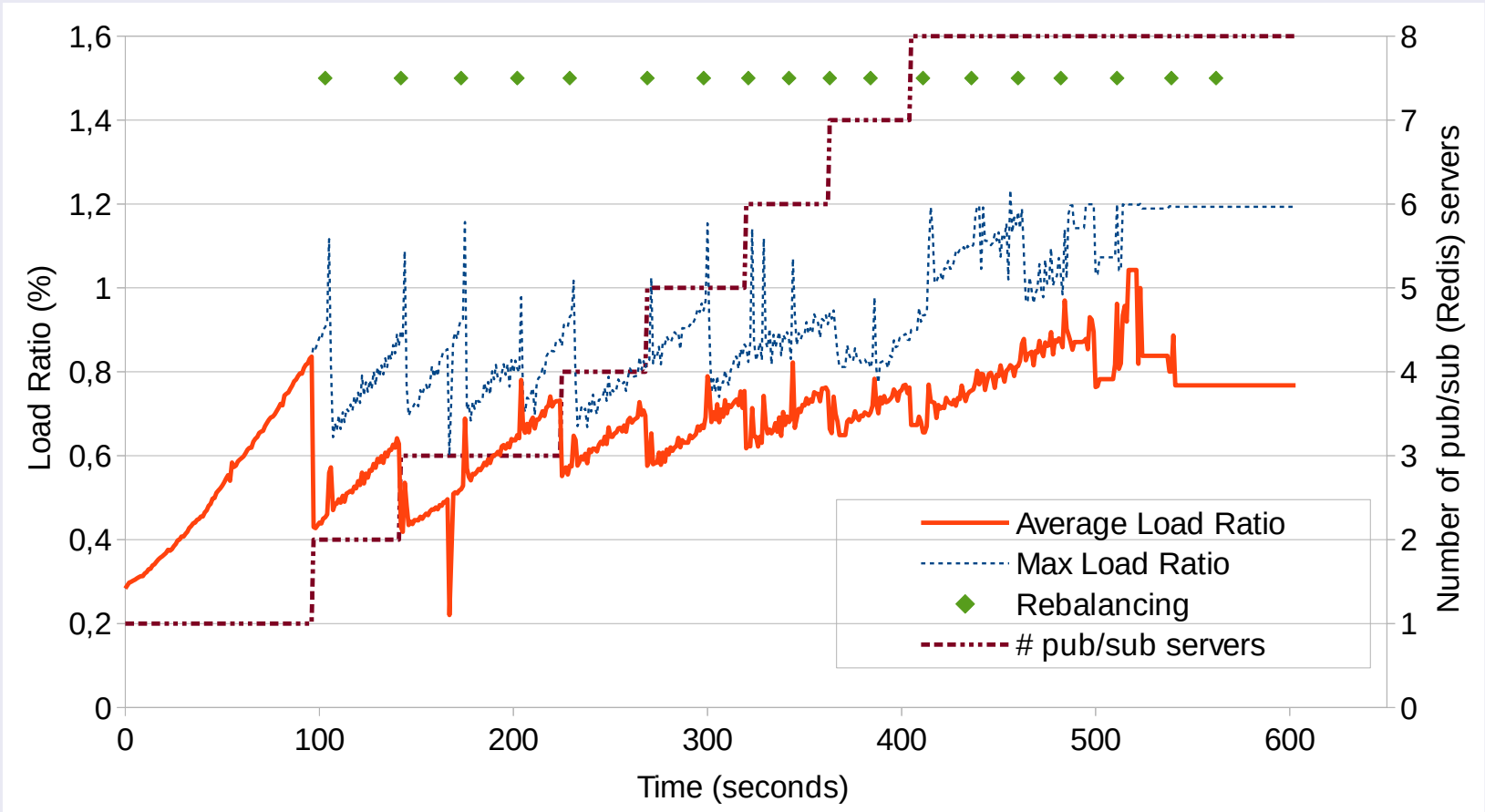
Algorithm 2 - High-load plan

Generating a high-load plan

begin
  P* = P.copy() **while** *true* **do**
    $(H_{max}, LR_{max}) = \max(LR_i \ \forall \ H_i)$;
    if $LR_{max} < LR^{high}$ **then**
      return P^*
    end
    $\overline{LR_{max}} = LR_{max}$;
    while $\overline{LR_{max}} \geq LR^{safe}$ **do**
      $(H_{min}, LR_{min}) = \min(LR_i \ \forall \ H_i)$;
      $c^{out}_{max} = \text{getBusiestChannel}(H_{max})$;
      P*.migrate(c^{out}_{max}, $H_{max} \rightarrow H_{min}$);
      $\overline{LR_{max}} = estimateLR(P*)$
    end
  end
end

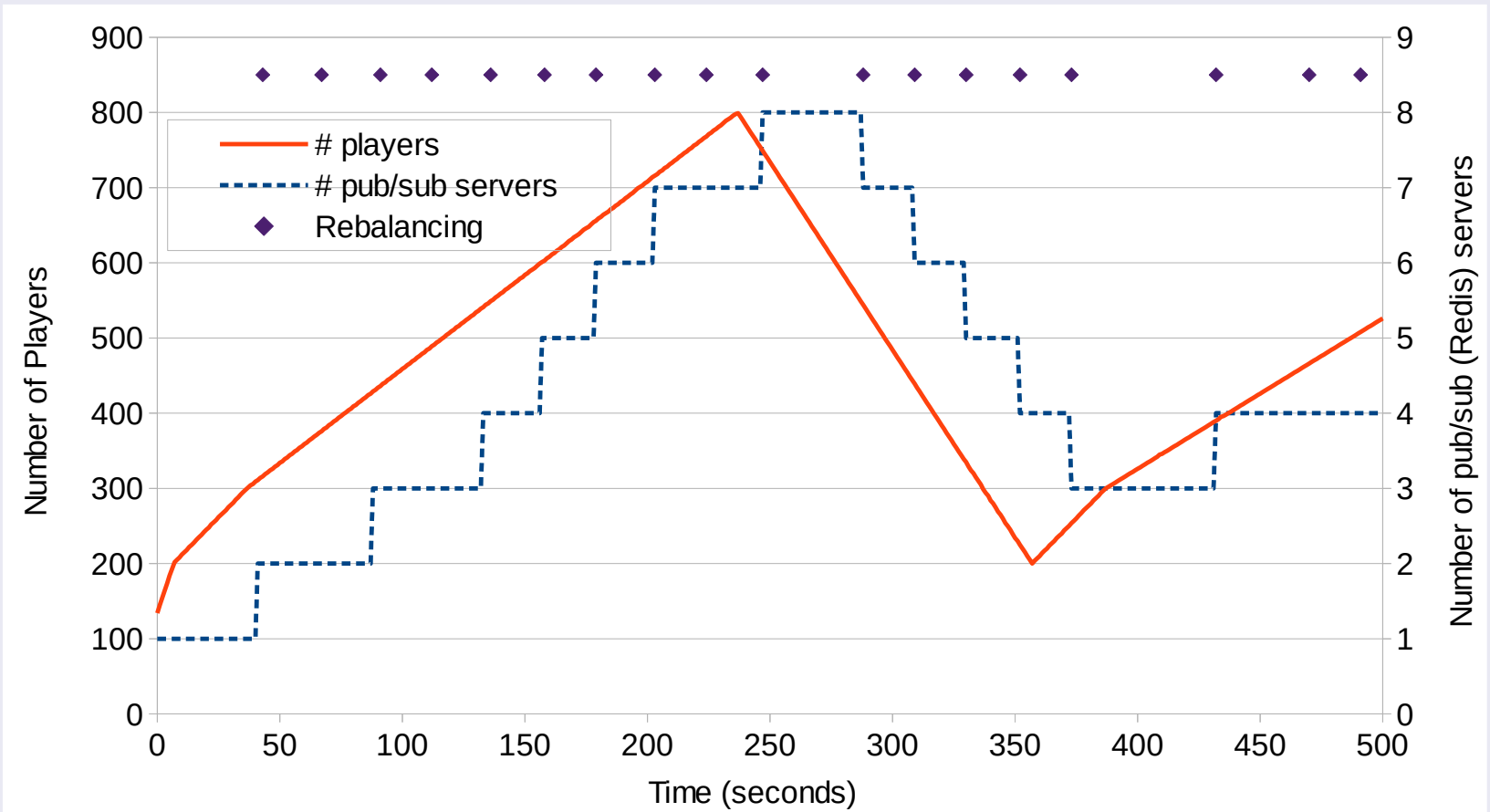
Experiment 2 - Scalability (4)

Dynamoth Load Balancer - Pub/Sub Server Load



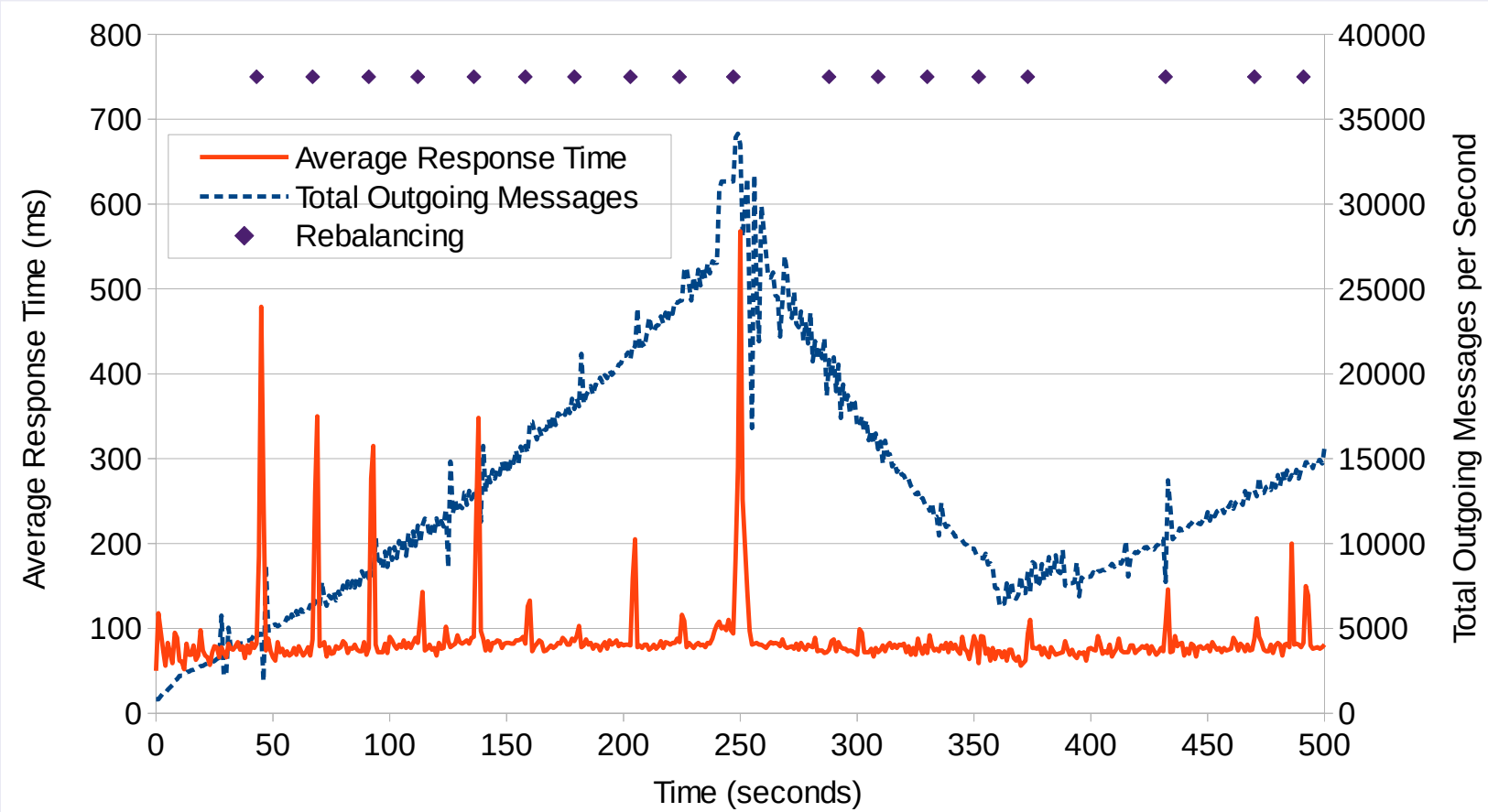
Experiment 3 - Elasticity (1)

Number of Players & Number of pub/sub Servers



Experiment 3 - Elasticity (2)

Avergage Response Time & Outgoing Messages



References

References for slide Applications of Channel-Based Pub/Sub:

1. http://cdn-parismatch.ladmedia.fr/var/news/storage/images/paris-match/actu/societe/samedi-rouge-sur-les-routes-de-france-156207/1585652-1-fre-FR/Samedi-rouge-sur-les-routes-de-France.jpg
2. https://www.drupal.org/files/project-images/gcm-logo.png
3. http://www.memoclic.com/medias/images/contenus/4/1198.jpg
4. http://theloftytraveler.com/wp-content/uploads/2012/03/stormyWeather.jpg
5. https://upload.wikimedia.org/wikipedia/en/thumb/9/9f/Twitter_bird_logo_2012.svg/1267px-Twitter_bird_logo_2012.svg.png
6. (Own image)