

# THIALFI

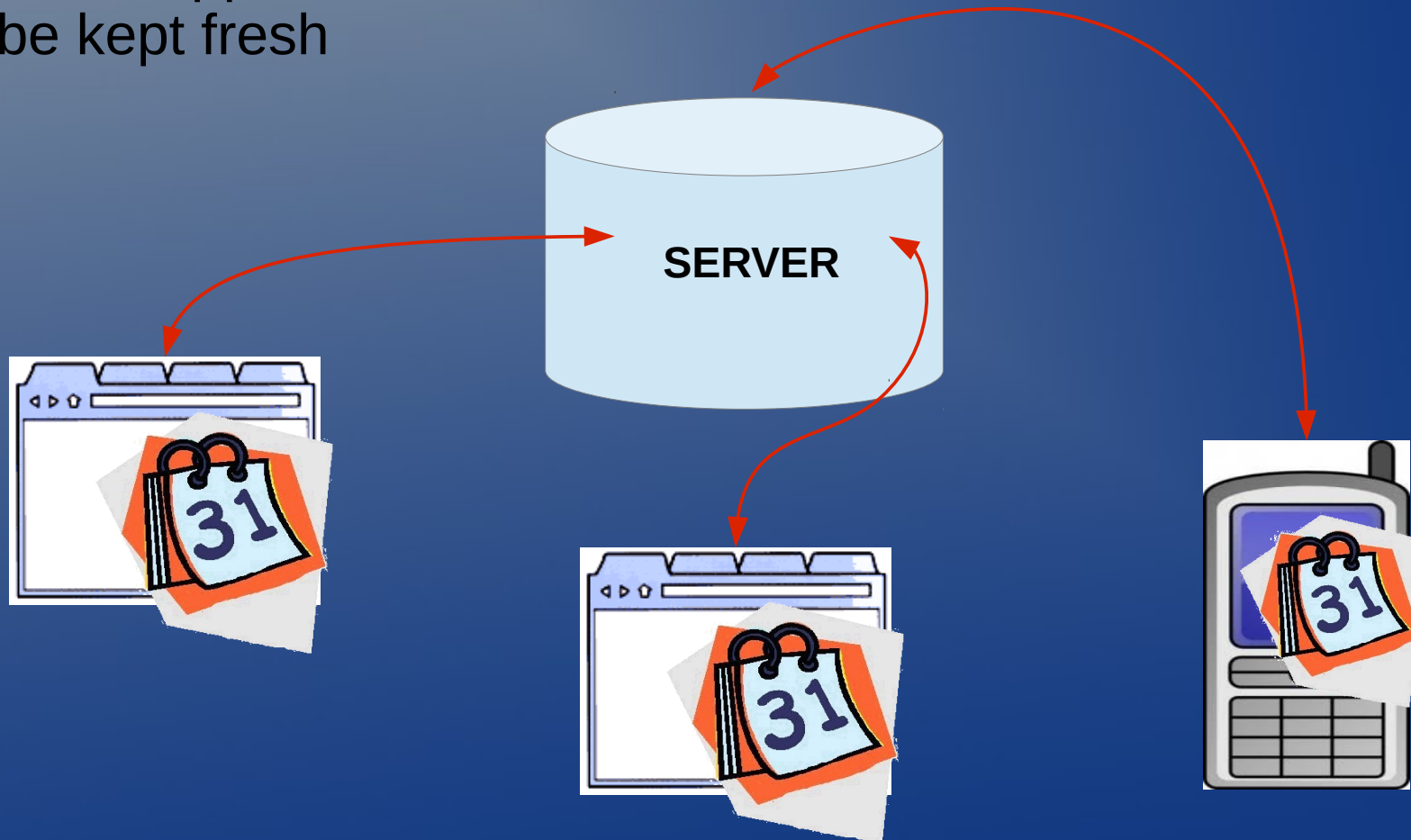
A Client Notification Service for Internet-Scale Applications

Authors: Atul Adya, Gregory Cooper, Daniel Myers, Michael Piatek  
Google, Inc.

Karol Nieniałowski

# Problem

- Ensuring the freshness of client data for applications that rely on cloud infrastructure to store data and mediate sharing
- Client applications maintain a local cache of their data that must be kept fresh



# How to resolve problem ?

- **Polling the servers for changes**
  - Tensions between timeliness and resources consumption
  - Inefficient – imposing significant load on the server
- **Push notifications to clients**
  - Fast but complex
  - Disconnected clients are overwhelmed by a flood of messages

# Design Alternatives

- **Integrating notifications with the storage layer** : track object sharing at the storage layer
  - Diversity of applications, data models, storage systems, ...
  - Complexity
- **Reliable messaging from servers to clients** :
  - Flood of messages upon wakeup
  - Application – specific of message

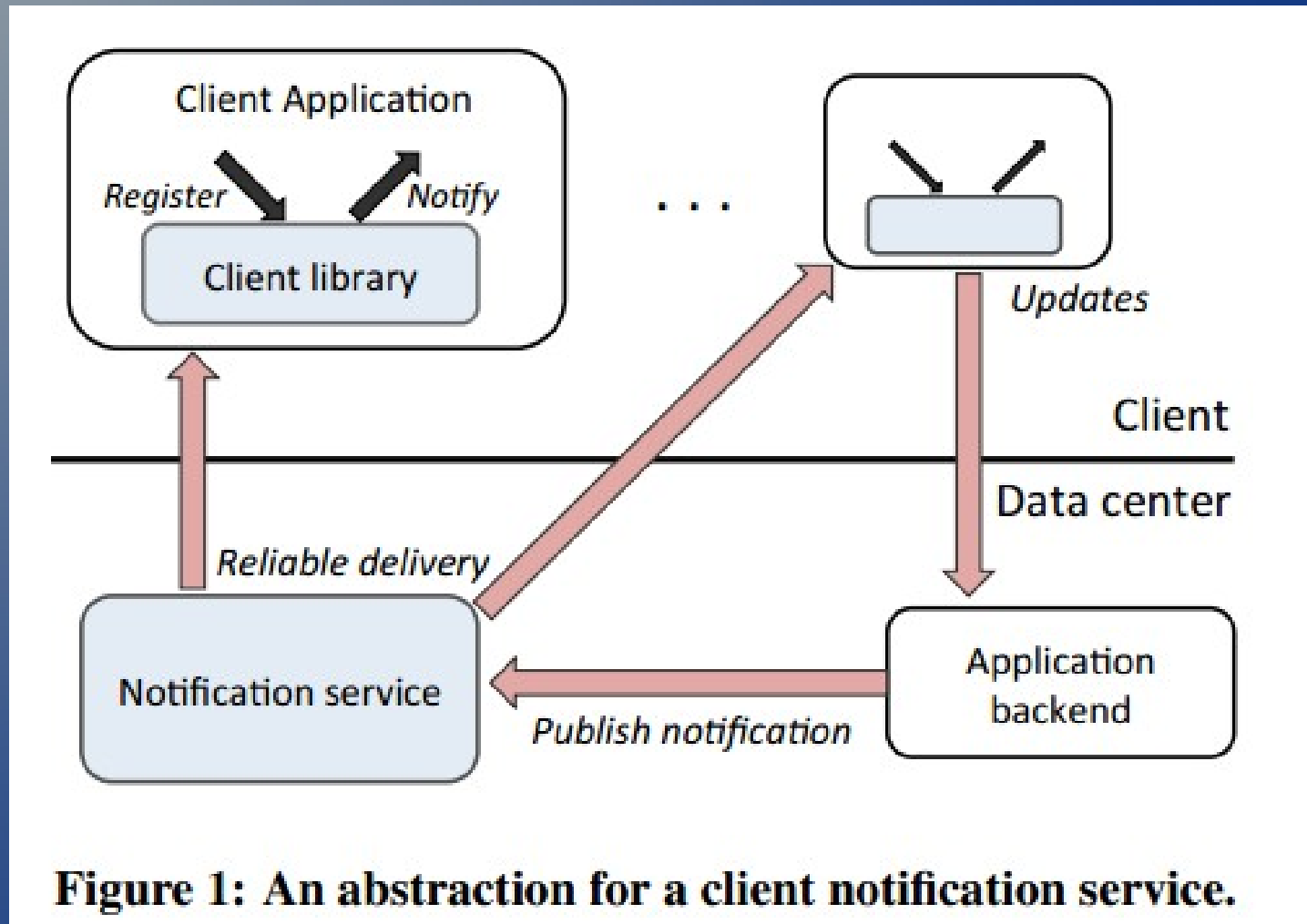
# Requirements

- **Tracking** : Service should know which clients are interested in what data
- **Reliability** : Notifications must be reliable and applications developer cannot be burdened with error detection and recovery mechanism
- **End-to-end** : Service must provide reliability in an end-to-end manner
- **Flexibility** : Service should support applications running in different platforms, written in variety of languages and using different channels.
- **Scalability** : Support millions of users and billions of objects.
- **Fast delivery** : The delay of notification should be small.

**Thialfi**

# Thialfi

- Highly scalable notification system for user-facing applications
- Provides sub-second notification delivery in the common case and clear semantics despite failures
- Supports applications written in a variety of languages (**C++**, **Java**, **JavaScript**)
- Supports applications running on a diverse set of platforms (**Web**, **mobile**, **native desktop apps**)
- Achieve reliability by relying on clients to drive recovery operations





# Client Library

- Library communicates with Thialfi service running in data centers using special protocol
- Registers for objects in Thialfi service
- Invokes callbacks to inform the application of registration changes and to deliver notifications
- **Thialfi delivers only the latest version number to client, NOT application data**
- Thialfi does not provide data synchronization

# Server Infrastructure

- Updates and notify Thialfi when object change
- Publisher library

**Publish**(objectId, versionNumber, source)

- Application backends generate version numbers for each object and ensure that they are monotonically increasing
  - Synchronous stores – by incrementing that numbers
  - Asynchronous stores – ex. by adding time of modification

# Communication

- Thialfi supports multiple communication channels to accommodate application diversity
  - XMPP
  - HTTP
  - Internal RPC
- Messages may be dropped, reordered, or duplicated
- Security :
  - Thialfi does not dictate a particular scheme for securing notifications
  - Thialfi does not mandate a channel security policy

# Client API

```
// Client actions
interface NotificationClient {
    Start(byte[] persistentState);
    Register(ObjectId objectId, long version);
    Unregister(ObjectId objectId);
}

// Client library callbacks
interface NotificationListener {
    Notify(ObjectId objectId, long version);

    NotifyUnknown(ObjectId objectId);

    RegistrationStatusChanged(ObjectId objectId,
        boolean isRegistered);

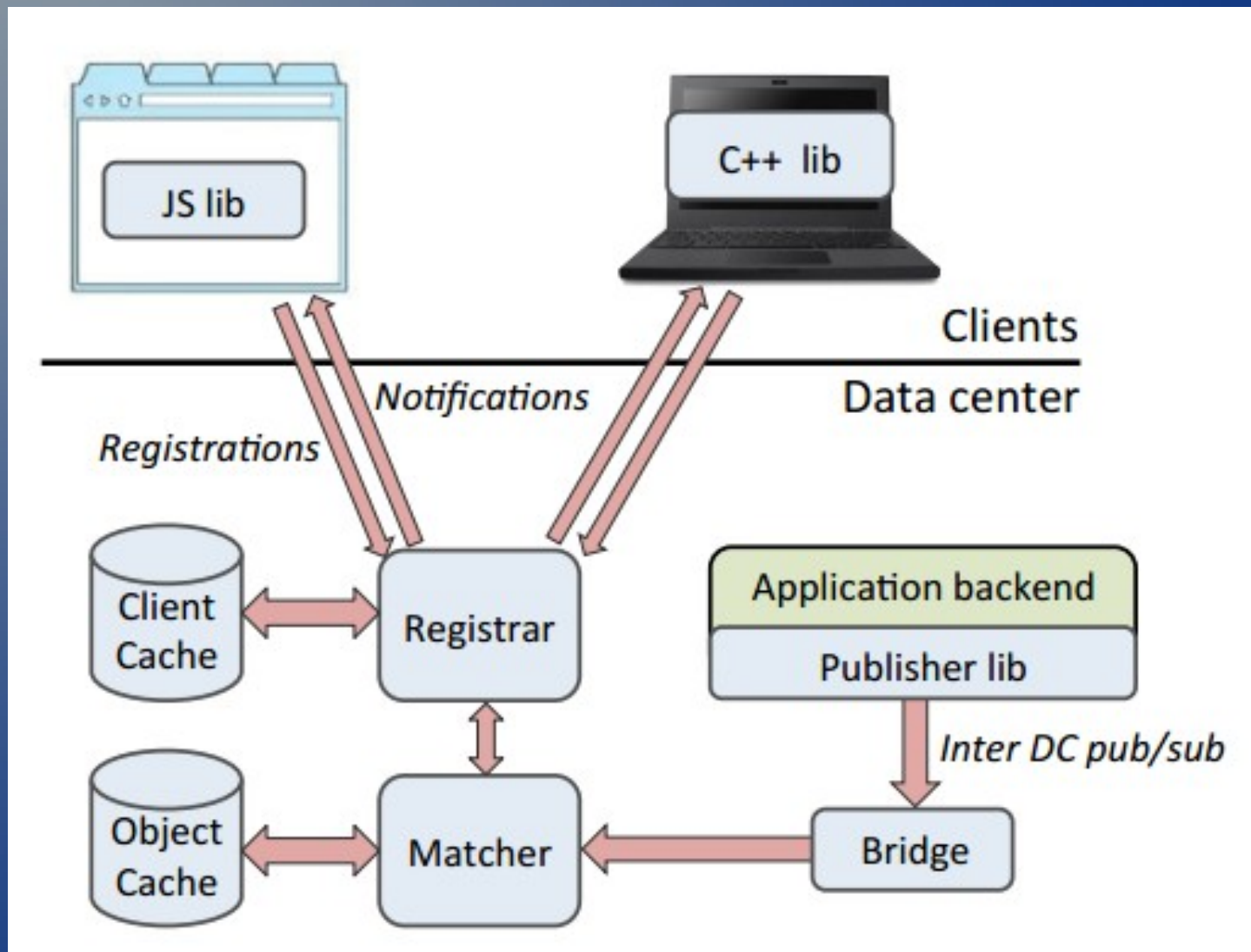
    RegistrationFailure(ObjectId objectId,
        boolean isTransient);

    ReissueRegistrations();

    WriteState(byte[] persistentState);
}
```

Figure 2: The Thialfi client API.

# Architecture



# Architecture (2)

- **Bridge Servers** are stateless, randomly load-balanced tasks that consume a feed of application-specific update messages from Google's infrastructure pub/sub service, translate them into a standard notification format and assemble them into batches for delivery to Matcher tasks
- **Matchers** consume notifications for objects, match them with the set of registered clients, and forward them to the Registrar for reliable delivery to clients.
- **Registrars** track clients, process registrations, and reliably deliver notifications.

# In-memory Design

# Partitioning by servers

- **Clients** are partitioned over Registrar Servers
- **Objects** are partitioned over Matcher Servers
- **Partitioning Key**
  - hash of a client or object ID
  - Key-space is statistically partitioned to contiguous ranges
  - One range is assigned to each server



# Registrar

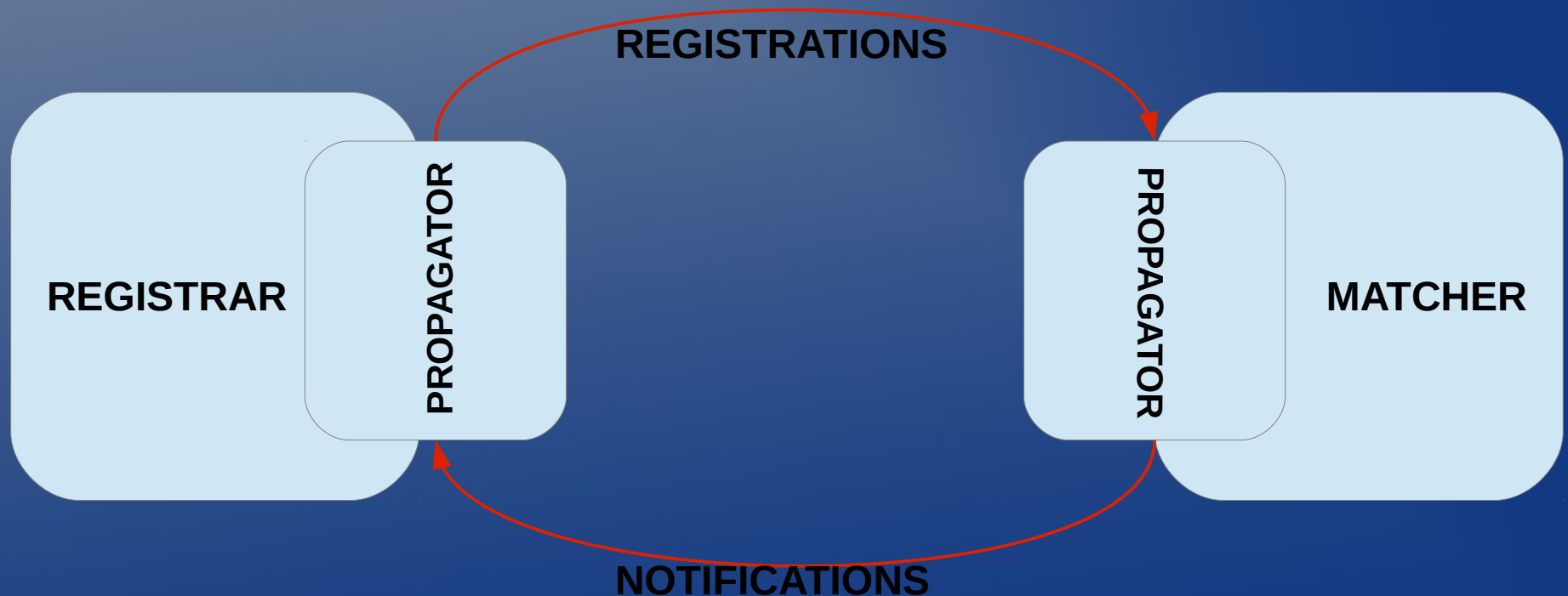
- Track clients
- Process registrations
- Reliably deliver notifications
- Maintain two set:
  - **Registrations** – objects of interest to the client indexed by clientId
  - **Pending notifications** – notifications not yet acknowledged by the client
- Have a monotonically-increasing **sequence number** of each client

# Matcher

- Match notification for objects with registered clients and forward them to the Registrar
- Store set of object with the latest version number provided by application backends
- Maintain copy of the registered clients for each object ( set indexed by objectId )

# Propagators

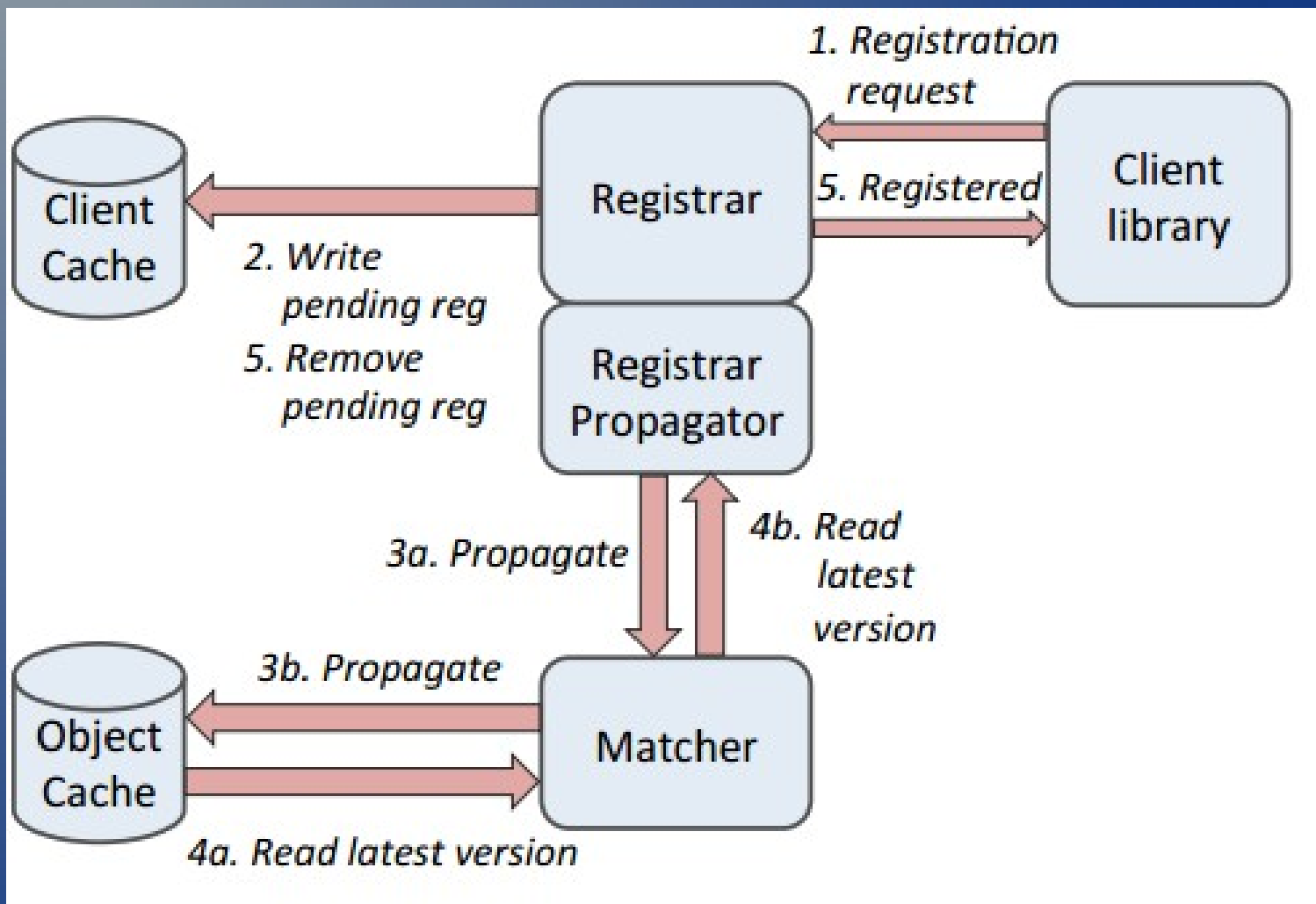
- Asynchronously propagate state between Matchers and Registrars
- Registrar Propagator copies client registration to the Matcher
- Matcher Propagator copies new notifications to the Registrar



# Client Token Management

- Thialfi recognize clients using **client tokens**
- Tokens are composed of two parts:
  - **Client identifiers**
  - **Session identifiers** (Thialfi data center ID)
- Client Library sends periodic **heartbeat** to the Registrar to inform that is still online
  - Interval 20 minutes
  - Messages are small
  - Processing only requires one In-memory operation

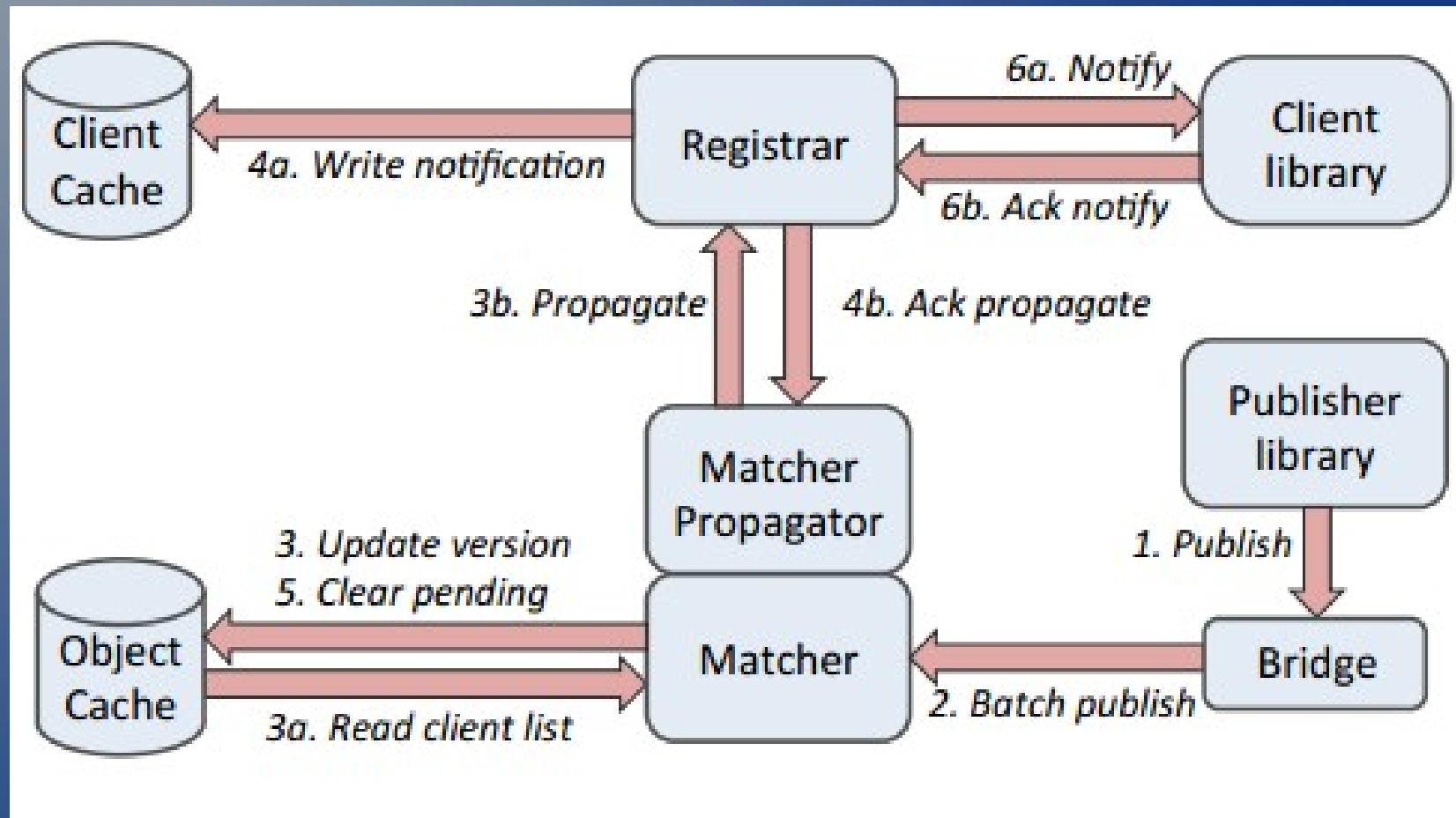
# Registration Operation



# Registration Sync Protocol

- Is used to keep the registrations at the client and the Registrar in sync
- Each message contains digest of the registered objects
- To compute digest is used HMAC-SHA1
- If a discrepancy is detected clients resend its registrations to the server

# Notification Operation



# Handling Failures

- Server Failures
  - Registrar detects errors in synchronization using tokens and Registration Sync Protocol
  - Client registration messages are sufficient to reconstruct the registration state at the Registrar
  - The latest version data at the Matcher is not recovered but it is not generate errors
- Network Failures
  - Sent messages may be lost, duplicated or reordered
  - Clients detect improper message by checking tokens or using Registration Sync Protocol



# Persistent Storage

# Bigtable

- A distributed storage system for managing structured data that is designed to scale to a very large size
- Does not support a full relational data model
- Sparse, distributed multi-dimensional sorted map
- Storage locations are named by { row key, column, version }
- [http://static.googleusercontent.com/external\\_content/untrusted\\_dlcp/research.google.com/pl//archive/bigtable-osdi06.pdf](http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/pl//archive/bigtable-osdi06.pdf)

# Registrar and Matcher Tables

Registrar Table						
Row Key	Client State			Object State		Propagation State
	created	last-seqno	presence	reg-{oid}	log-{oid}	pending
<i>hash(user):user:uuid</i>	<i>appid@0</i>	<i>""@seqno</i>	<i>addr@seqno</i>	<i>""@seqno</i>	<i>""@version</i>	<i>""@seqno</i>

Matcher Table			
Row Key	Object State	Client State	Propagation State
	version	reg-{client-id}	pending
<i>hash(object-id):object-id</i>	<i>appid@version</i>	<i>appid@seqno</i>	<i>""@version</i>

**Table 2: Bigtable layout for server-side state. *a@b* indicates a value *a* at timestamp *b*. *seqno* refers to the sequence number assigned by the Registrar for that particular client.**

# Client Garbage Collection

- Garbage collector deletes a *created* cell from clients' Registrar row when detect that client is offline
- If a garbage collected clients comes back online its created cell will be absent and client need to change ID
- Register Table is periodic scanned for rows without created cell which are deleted afterwards
- Online clients are informed about invalid ID

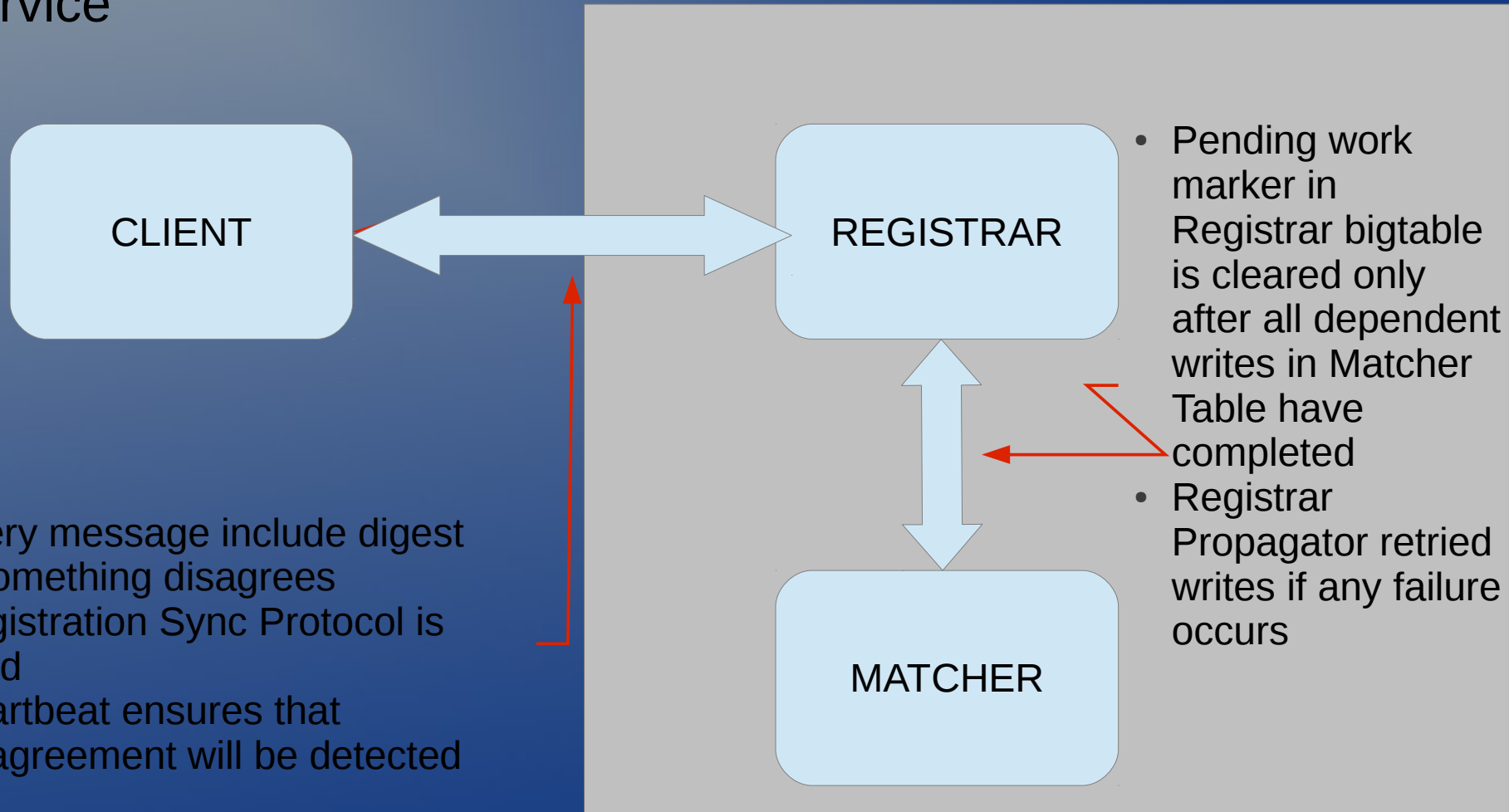
**Correctness**

# Achieving Reliable Delivery

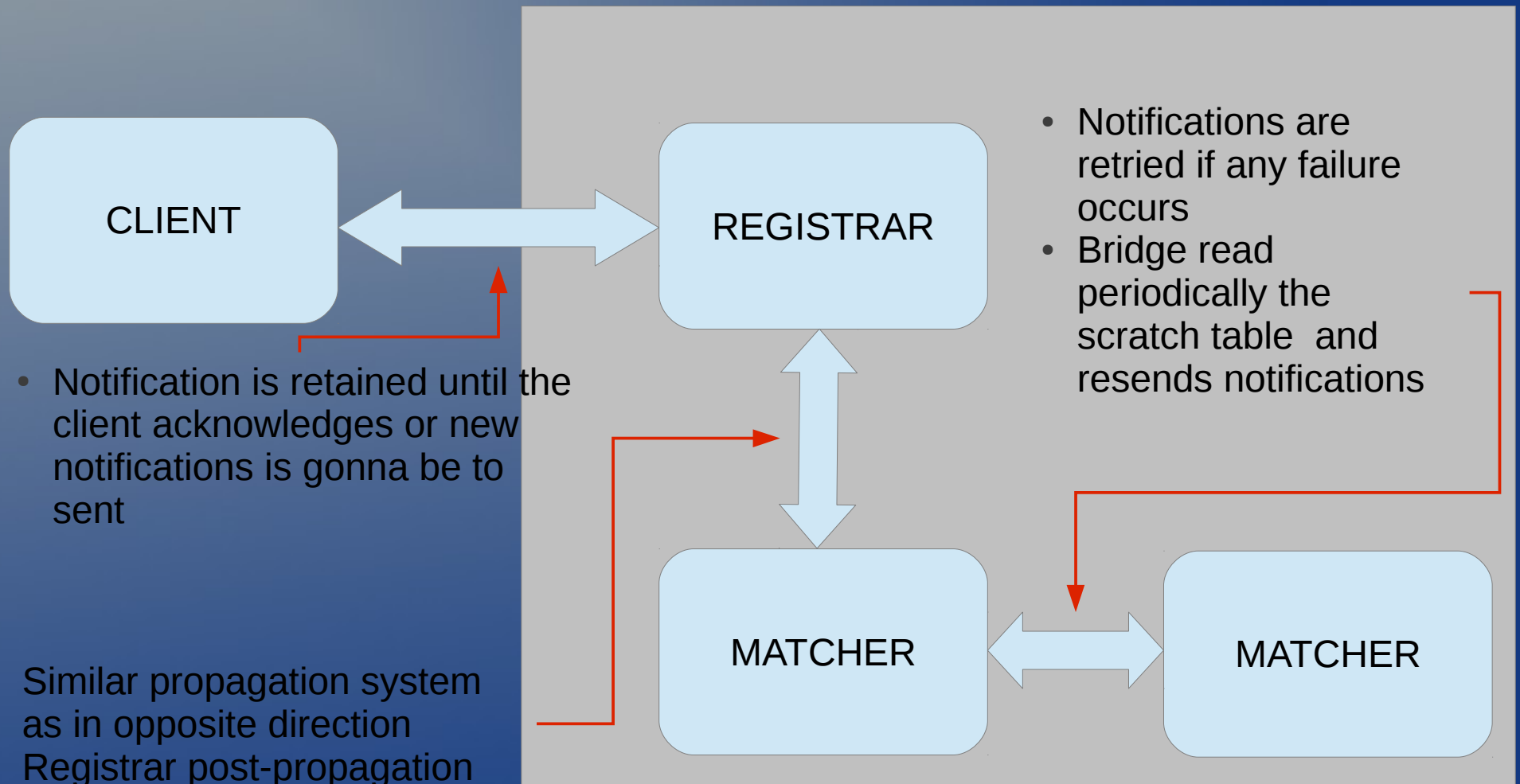
**Reliable delivery property** : If a well behaved client registers for an object X, Thialfi ensures that the client will always eventually learn of the latest version of X

# Registration states

- Is determined by the client from which it propagates to the Service



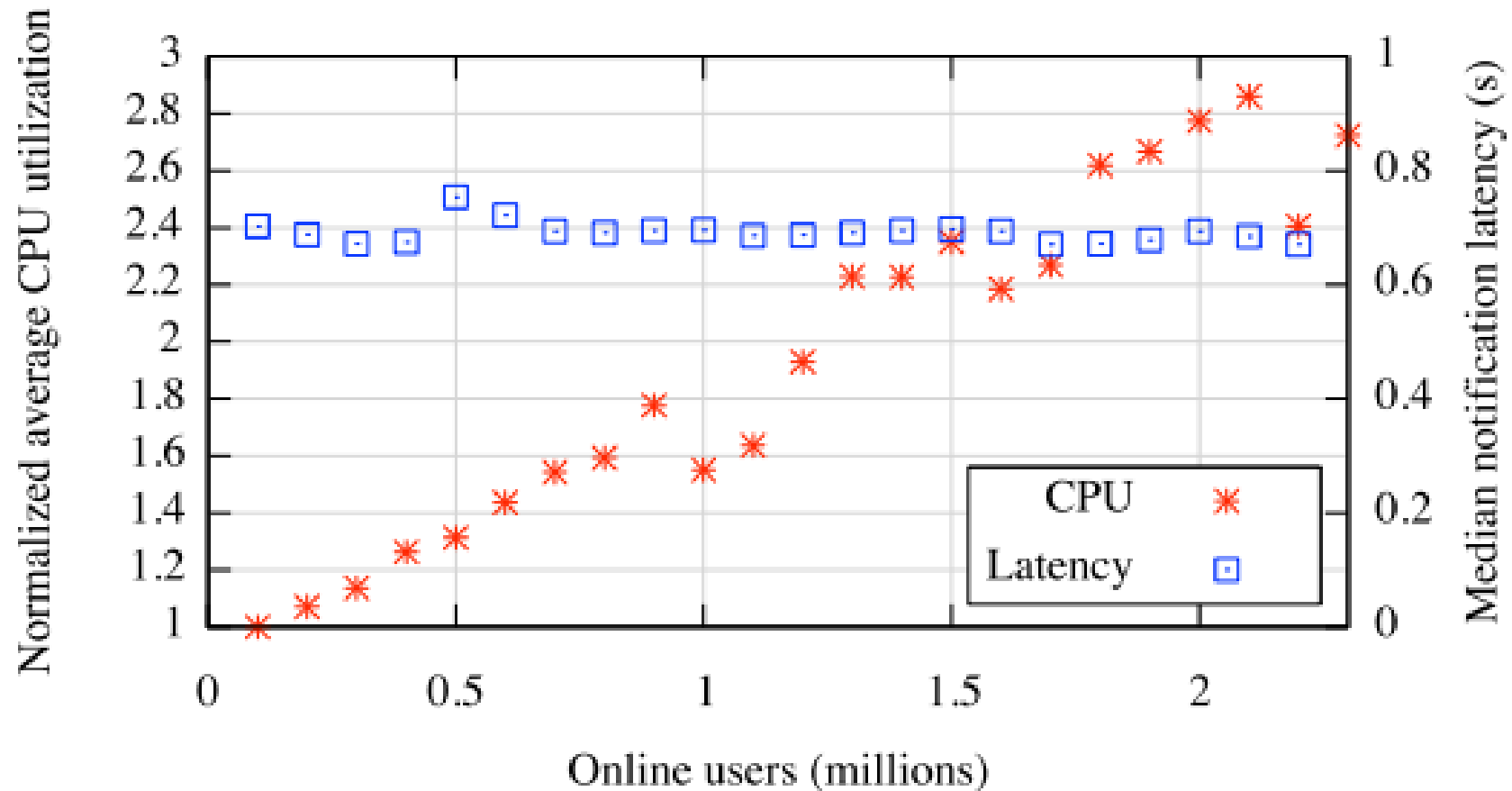
# Notification state



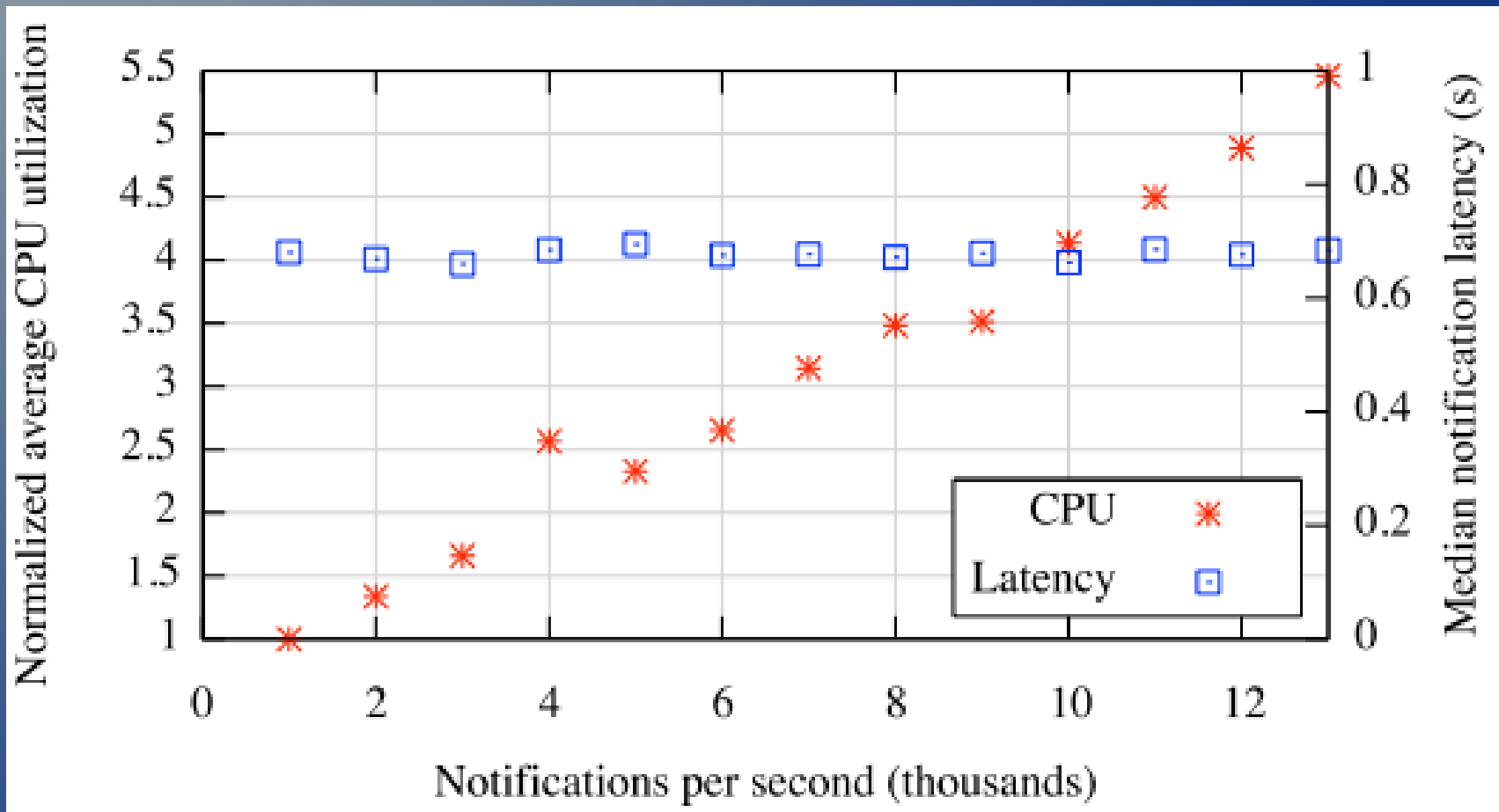


# EVALUATION

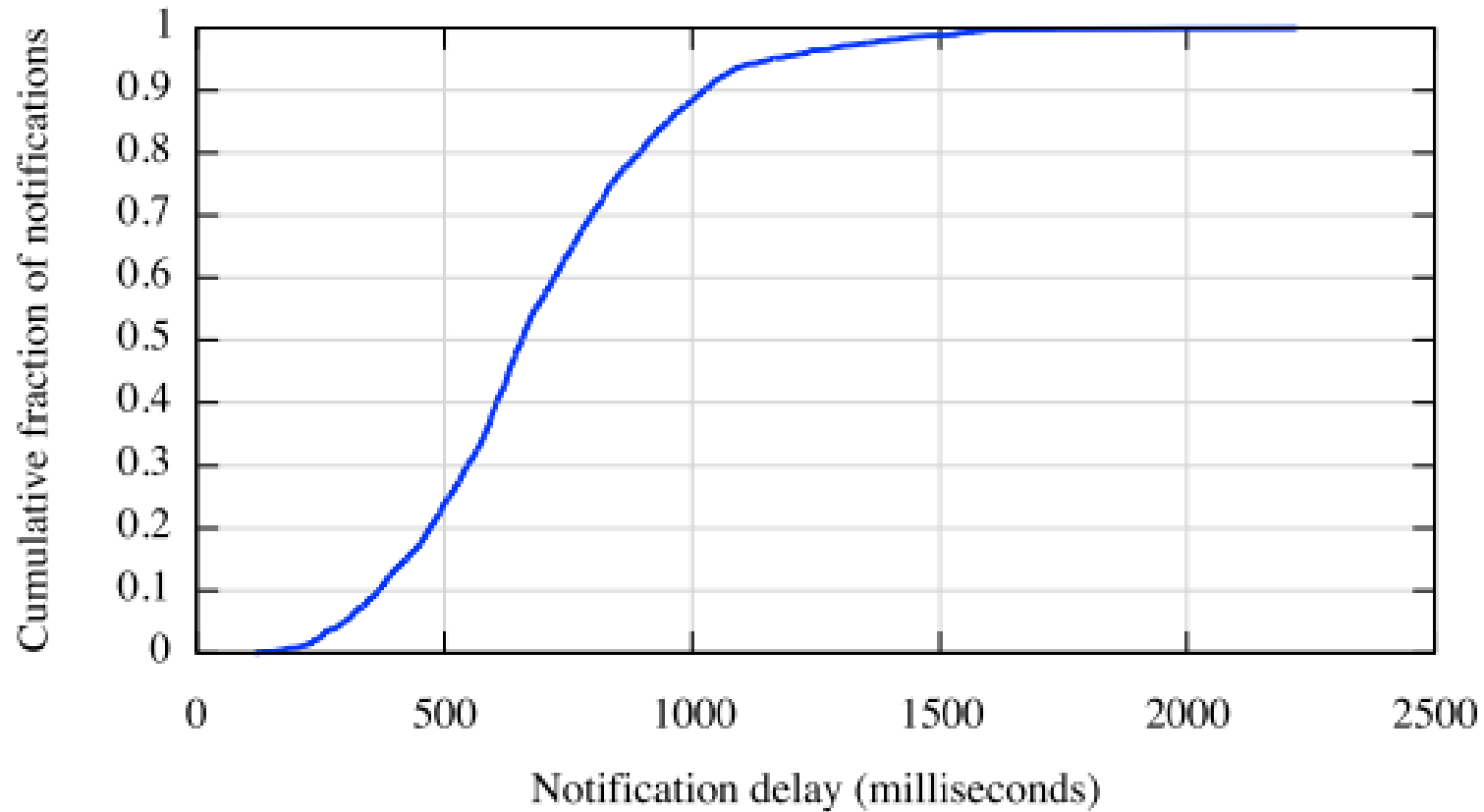
# Scalability – Active Users



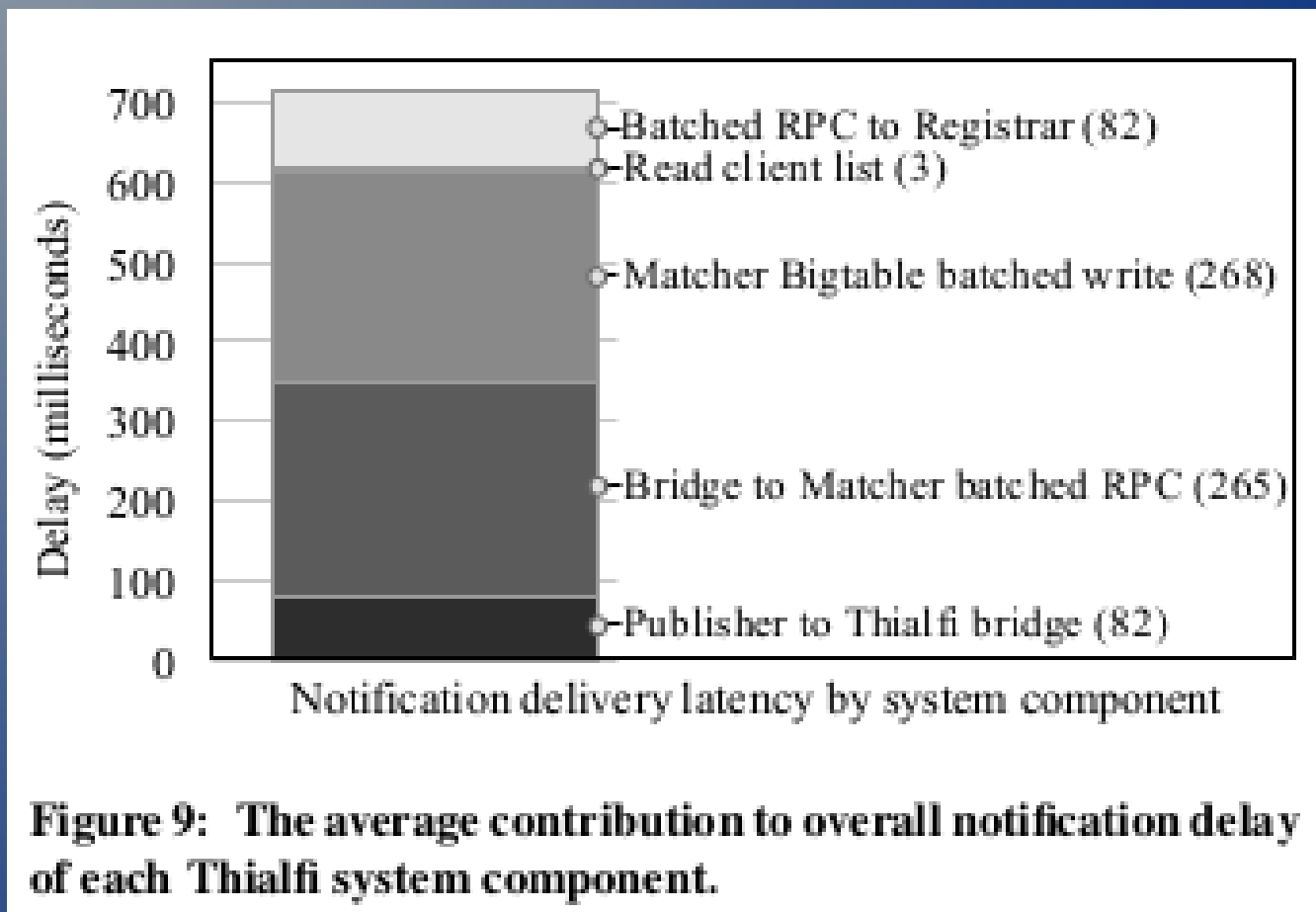
# Scalability - Notifications



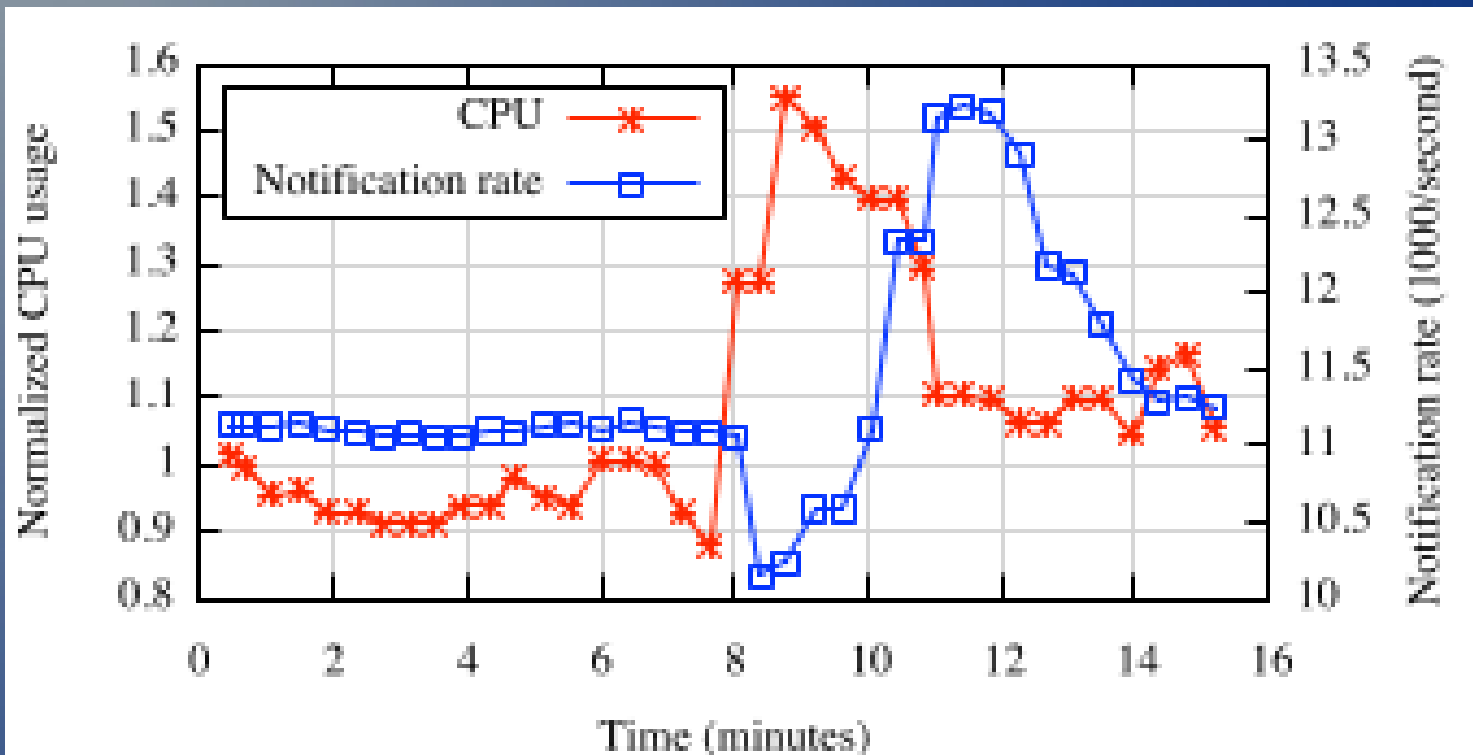
# Performance



# Performance (2)



# Fault-tolerance



**Figure 10: CPU usage and notification rate during the sudden failure of a Thialfi data center.**

**THANK YOU !**