CREASON: Callable and Replicated Shared Objects over NoSQL*

to appear in ICDCS 2017, Atlanta, GA, USA

Pierre Sutra, Etienne Rivière,
Cristian Cotes, Marc Sánchez Artigas, Pedro Garcia Lopez,
Emmanuel Bernard, William Burns and Galder Zamarreño

Télécom SudParis, CNRS, Université Paris-Saclay, France

University of Neuchâtel, Switzerland

Universitat Rovira i Virgili, Tarragona, Spain

Red Hat

*slides by E. Rivière

Building scalable Cloud applications

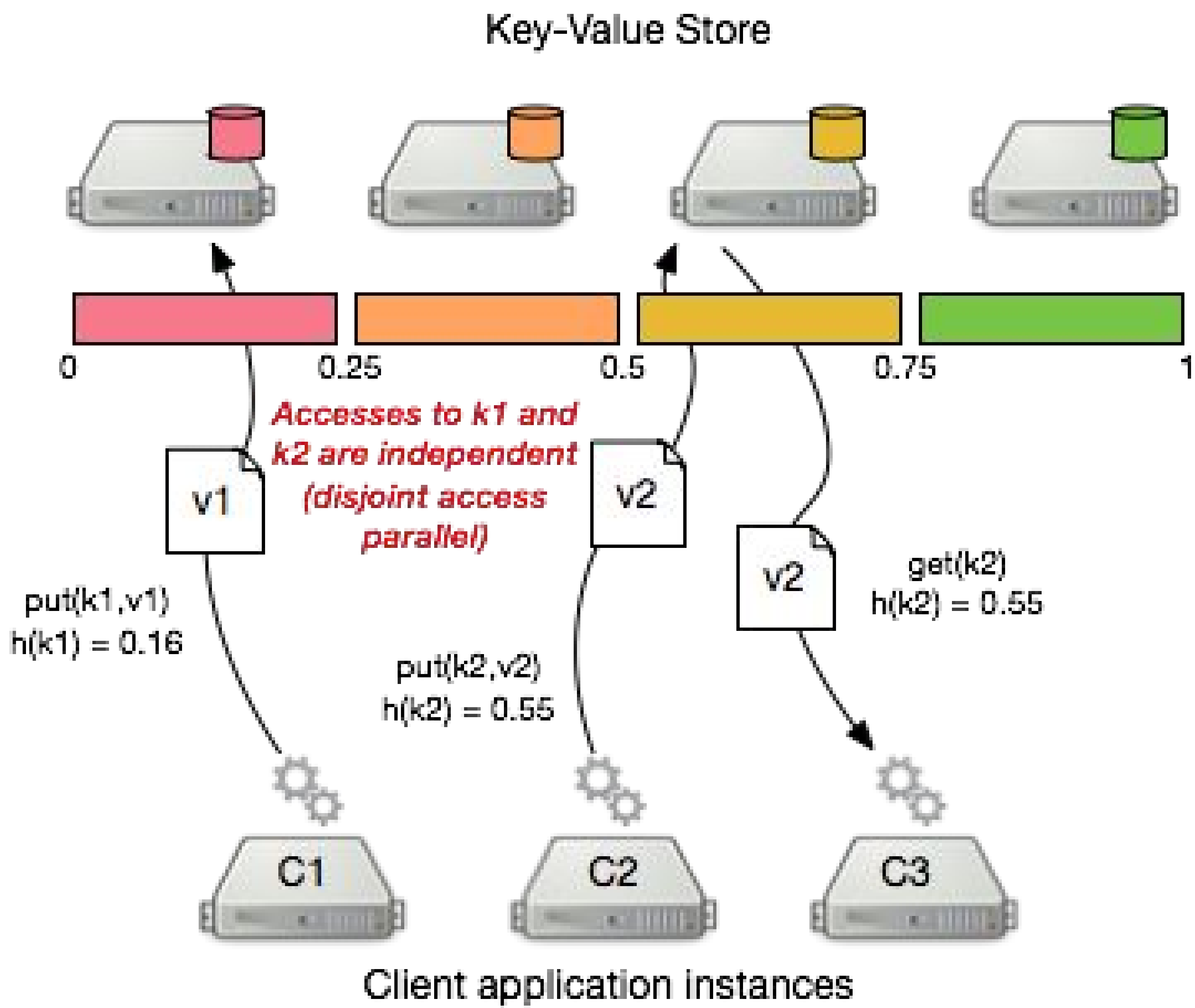
- Cloud applications handle large amounts of clients
  - Large amounts of data: need *scalable* data storage
  - Pay-as-you-go model requires *elastic* scaling
- Failures happen often and must not break service
  - Application data stored in database *persistently*
  - Multiple copies: *consistency* under concurrent operations
- Application design must be *simple* and *scalable*
  - Easy-to-learn programming model and database integration
  - Sharing of data between application instances with database

TELECOM SudParis
INSTITUT Mines-Télécom

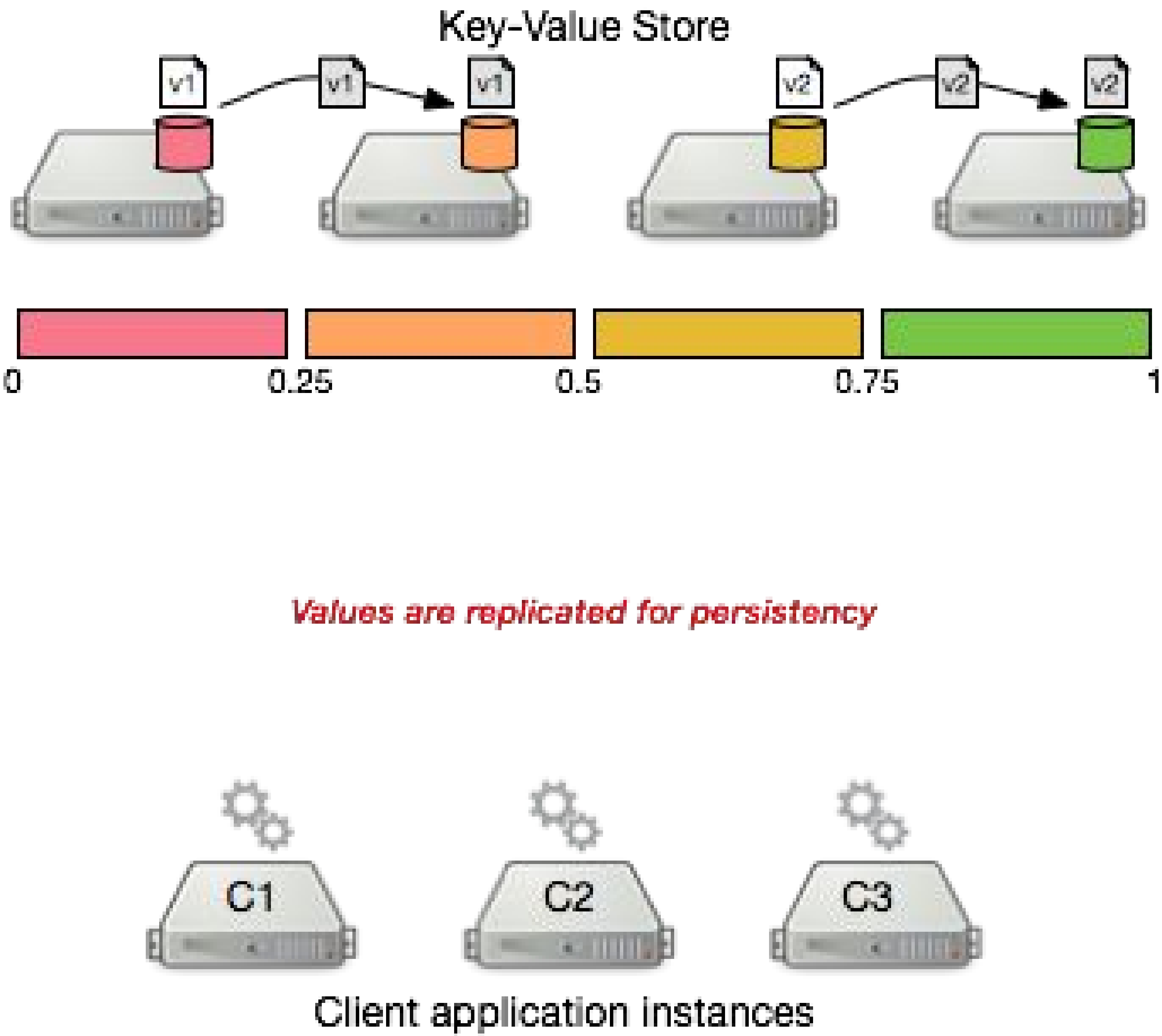
There is not only SQL

- Scaling "traditional" relational (SQL) databases
  - ☹ Limited horizontal scalability, poor support for elasticity
    - Sharding is complex and static, no cross-shard consistency
  - ☹ Fault tolerance with master/slave replication
- NoSQL databases to the rescue
  - ☹ Simpler data schema and querying
    - Only primary index: key/value store, no support for joins
  - ☹ Independent accesses to different keys
  - ☺ Excellent horizontal scalability and elasticity

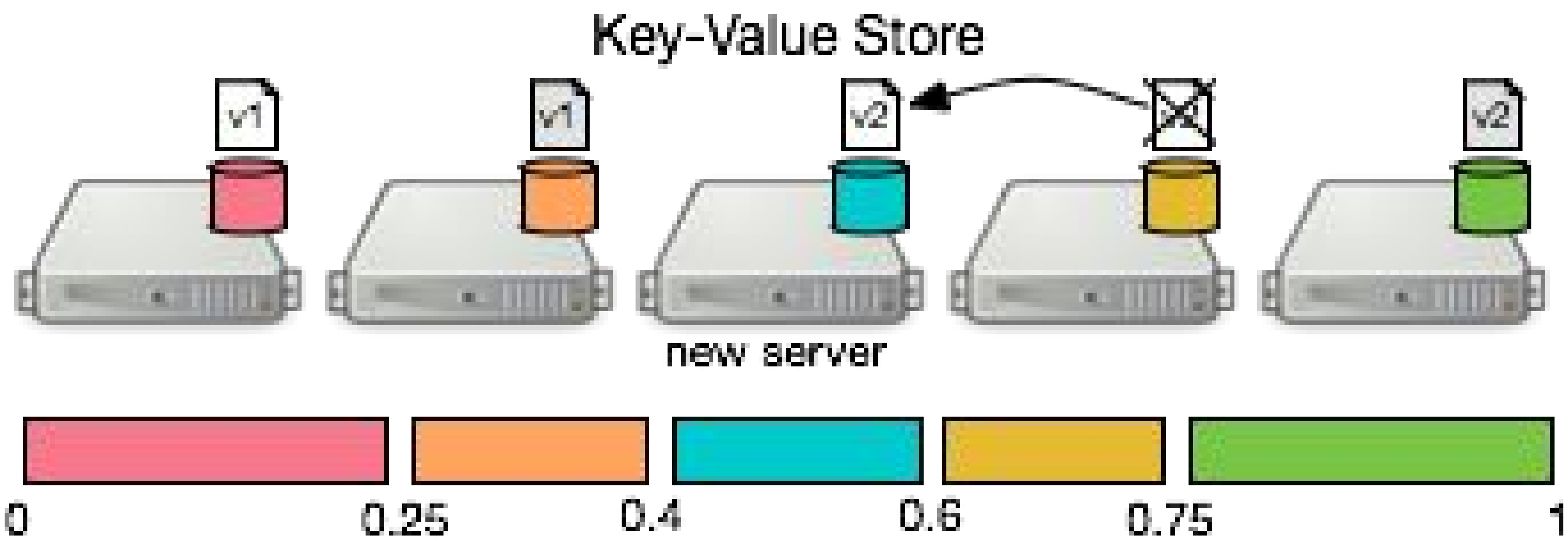
NoSQL for scalable applications



NoSQL for scalable applications

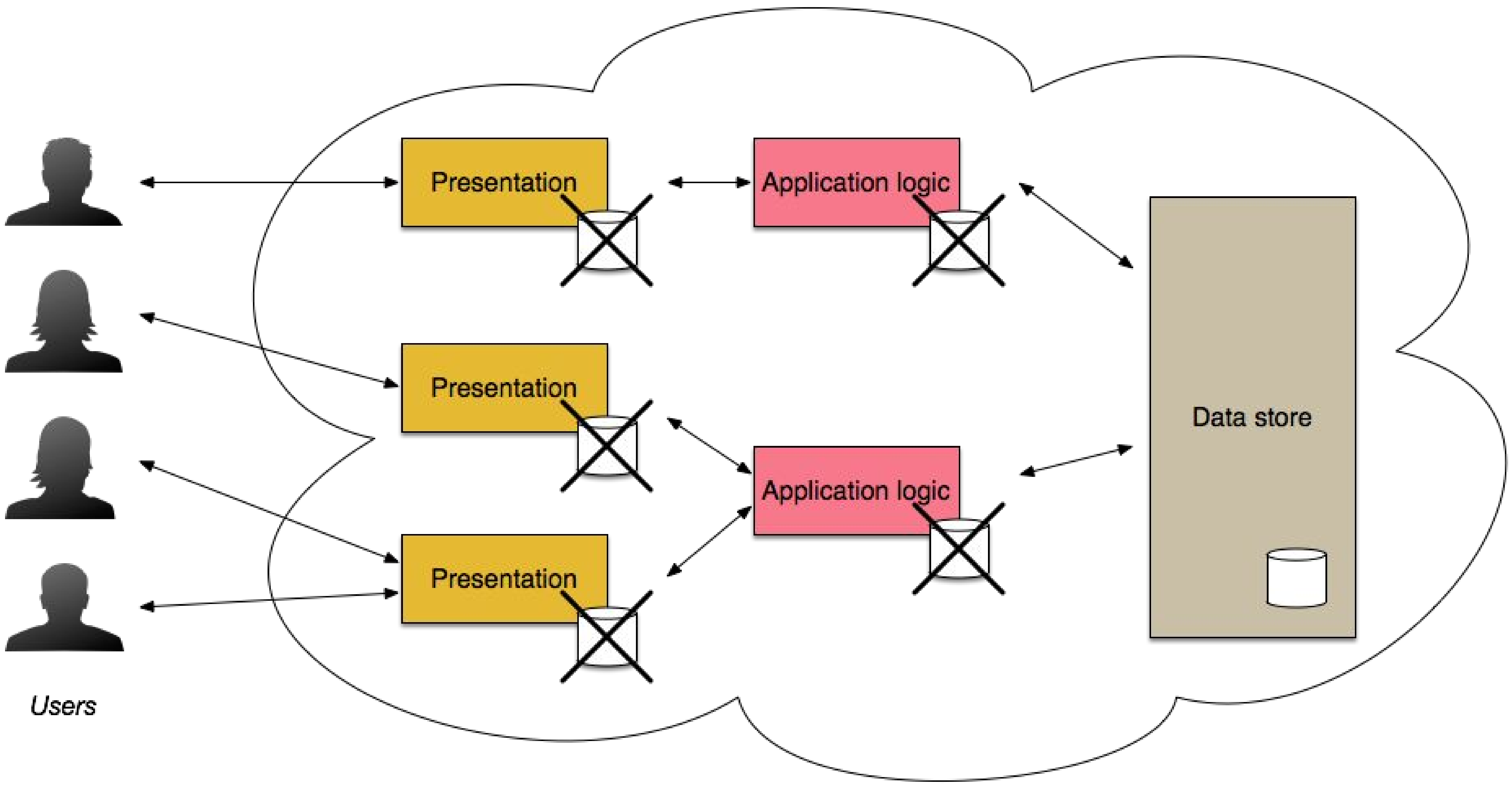


NoSQL for scalable applications



Adding a storage server without service interruption allows horizontal elastic scalability

Typical Cloud-based application



NoSQL databases

- Many flavours of NoSQL
  - General-purpose or {Document,Graph,Column}-oriented
- Interface = variation of a key/value store API
  - Some also support transactions, scans, etc.

NoSQL in an object-oriented application

- Object-oriented programming = prevalent model
- Data shared between application instances

  Objects survive termination of application instances
  & failure of NoSQL servers

- Database storage and in-memory objects use different representations

  But require a *mapping phase* between the two representations
  impedance mismatch

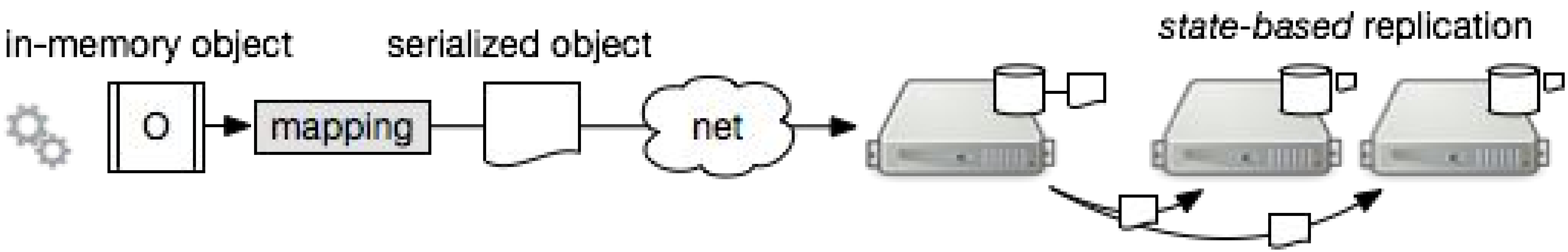
TELECOM SudParis
INSTITUT Mines-Télécom

State-of-the-art: Object-DB mappers

- Object-Relational Mapper
  - Store application objects in relational database
  - Hibernate
  - Integration with OO langage (e.g., Java)

- Object-NoSQL Mapper
  - Maps and store application objects in NoSQL database
  - Hibernate OGM, MongoDB Morphia, Google's Objectify

Client-side Object-NoSQL mapping



- ☹ Access to object: fetch full serialized representation from DB
  - Objects instantiated locally and their methods also called locally
  - Some objects may grow very large
    - Methods may access only a small part of their content
  - Data structure (e.g. graph) traversal = multiple back-and-forth with DB
- ☹ Concurrent accesses to objects with no strong consistency
  - Objectify (part of Google App Engine) not thread-safe

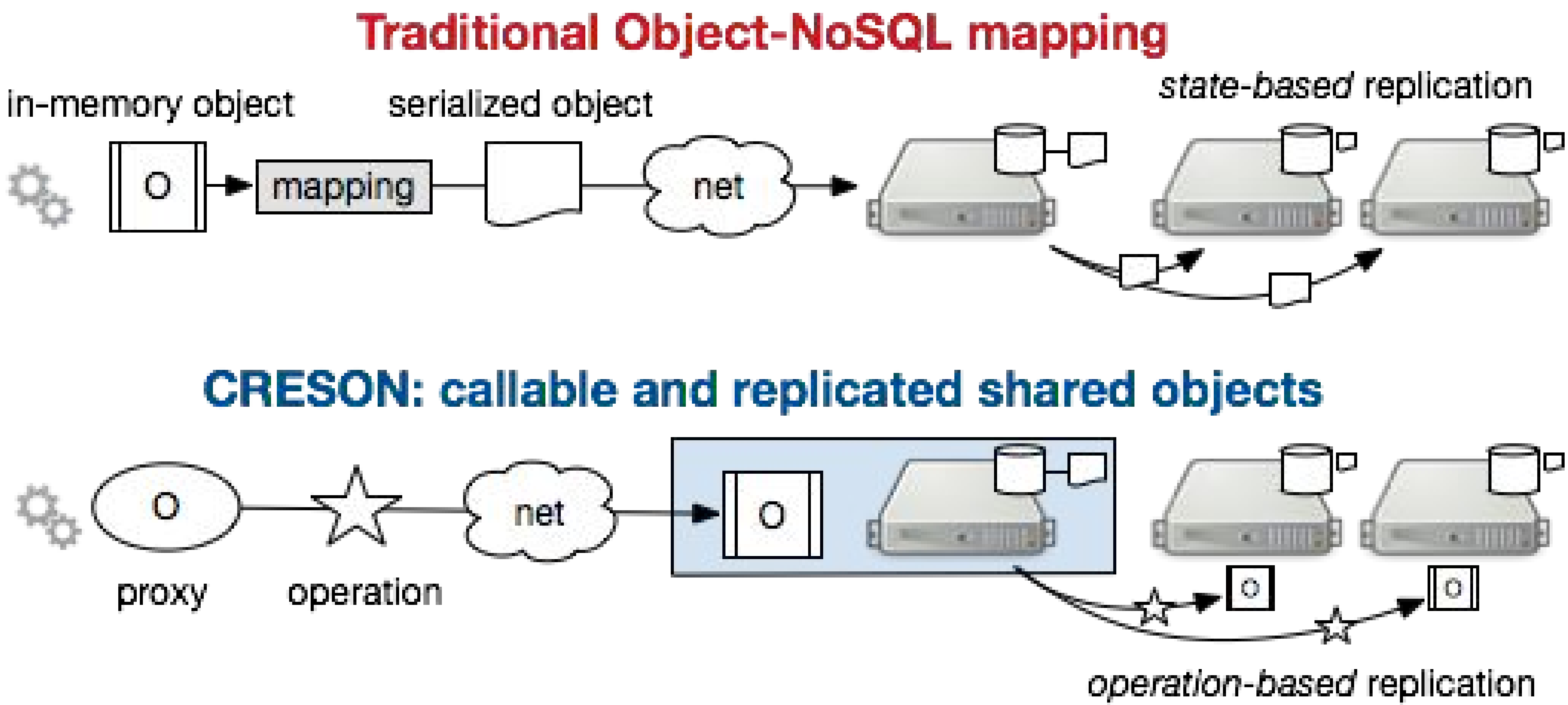
CRESON: objectives

- Support callable objects over NoSQL
  - Application objects instantiated from the DB *at the server side*
    - No shipping of any serialized representation over the network
  - Method calls also performed at the server side

- Dependability and concurrent accesses to shared objects
  - Objects are replicated for persistence
  - Replication happens at the level of operations (method calls)
    - No shipping of full serialized state between replicas
    - Shared objects with *strong consistency* guarantees
    - Including for composed operations accessing multiple objects

TELECOM SudParis
INSTITUT Mines-Télécom

CRESON: server-side mapping



Outline

- Introduction and motivation
- Server-side Object-NoSQL mapping with CRESON
- CRESON design
  - LKVS abstraction
  - Object management components
  - State Machine Replication
  - Guarantees
- Portage of an existing application StackSync to CRESON

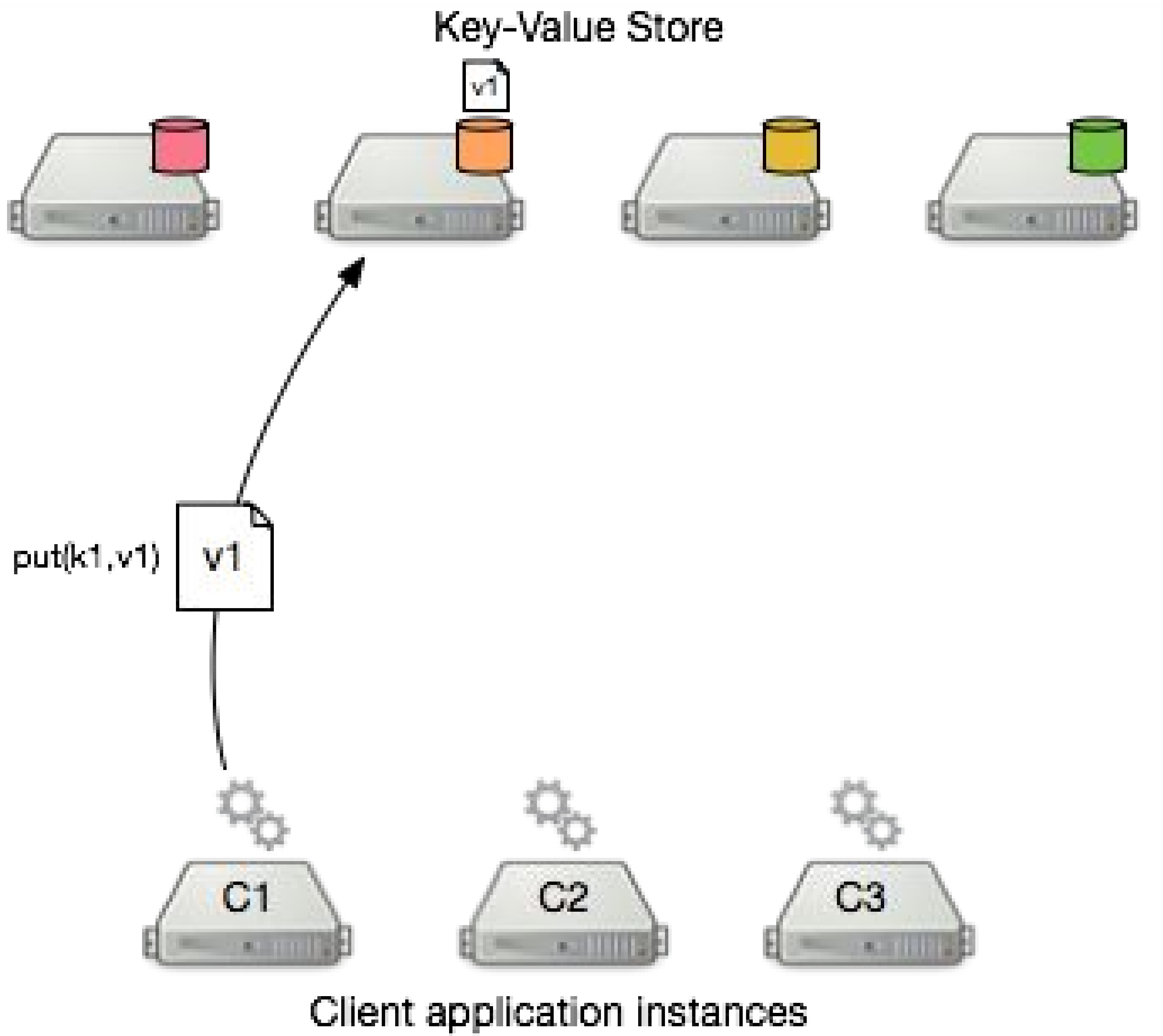
CRESON: components

- LKVS: novel NoSQL storage abstraction
  - ***Listenable*** Key-Value Store
  - Extends key-value API
- Object management logic atop the LKVS
  - Implemented as part of the listener handlers
  - Maintain multiple replicas of the object
  - Implement state-machine replication (operation-based)
- Client-side integration with the Java language
  - Using annotations (similar to JPA)

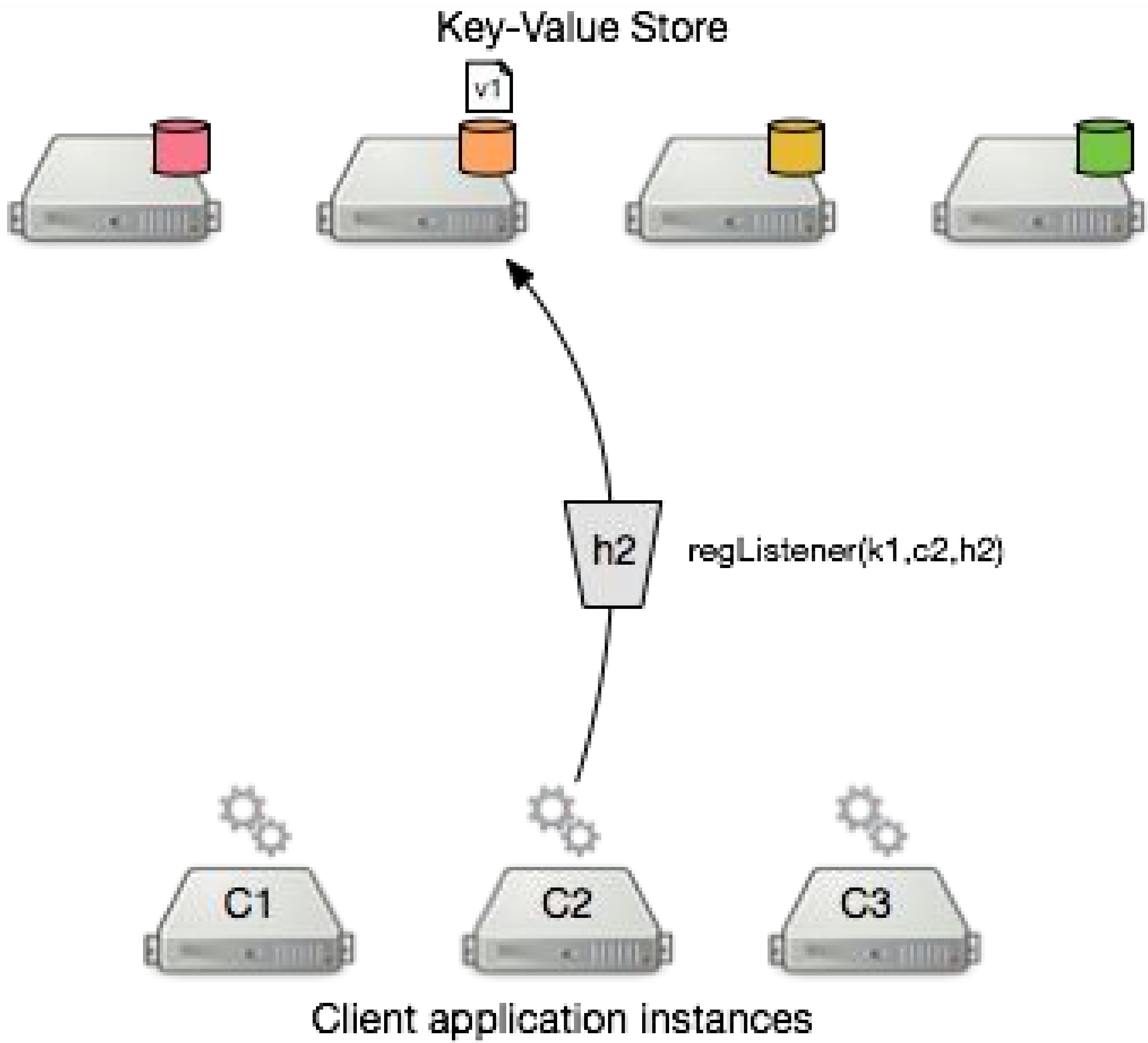
Listenable Key/Value Store

- Classical Key/Value API
  - `void put(K k, V v)`
  - `V get(K k)`
- Two new calls
  - `void regListener(K k, Handler h, Listener l)`
  - `void unregListener(K k, Listener l)`

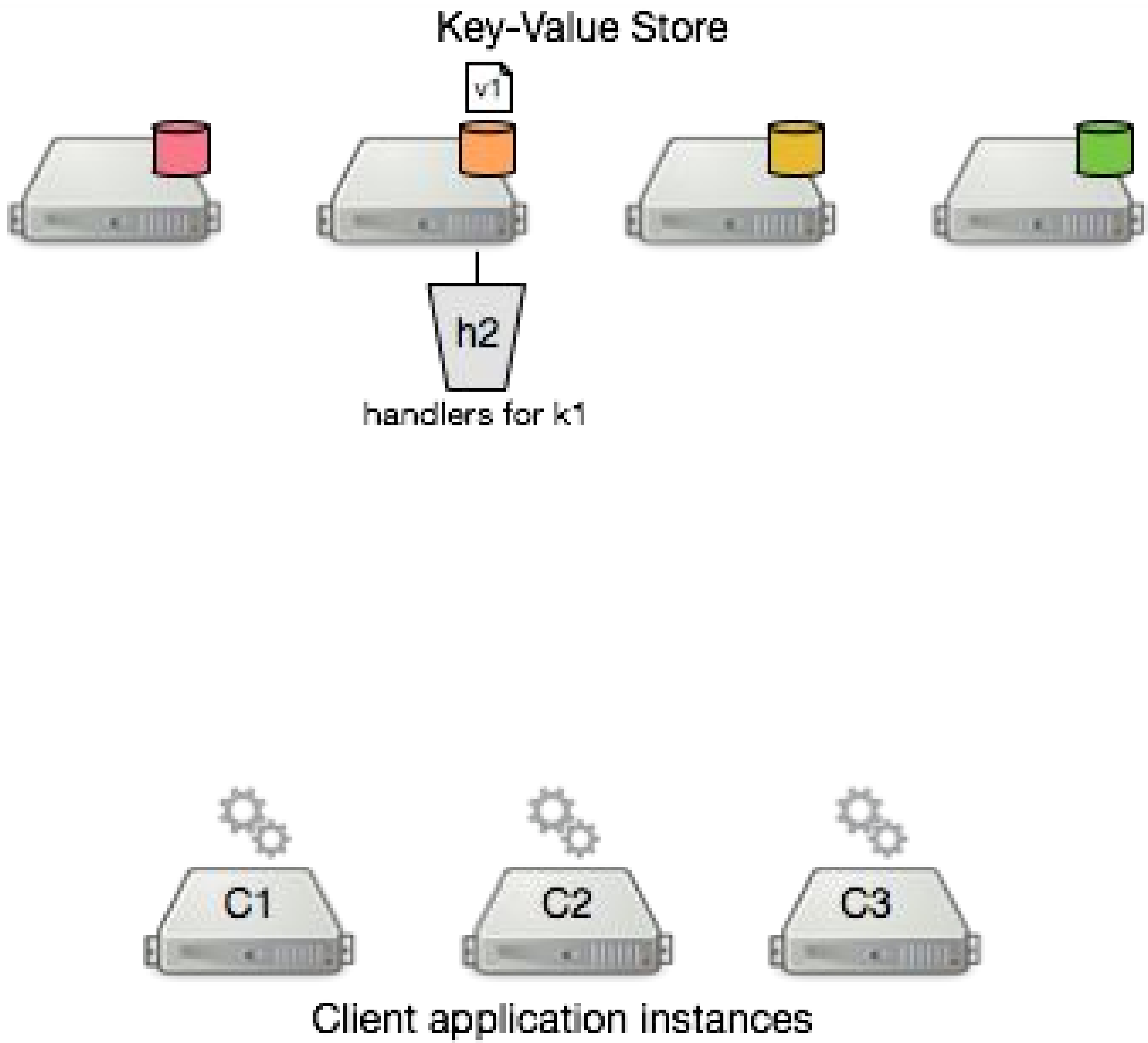
LKVS illustrated



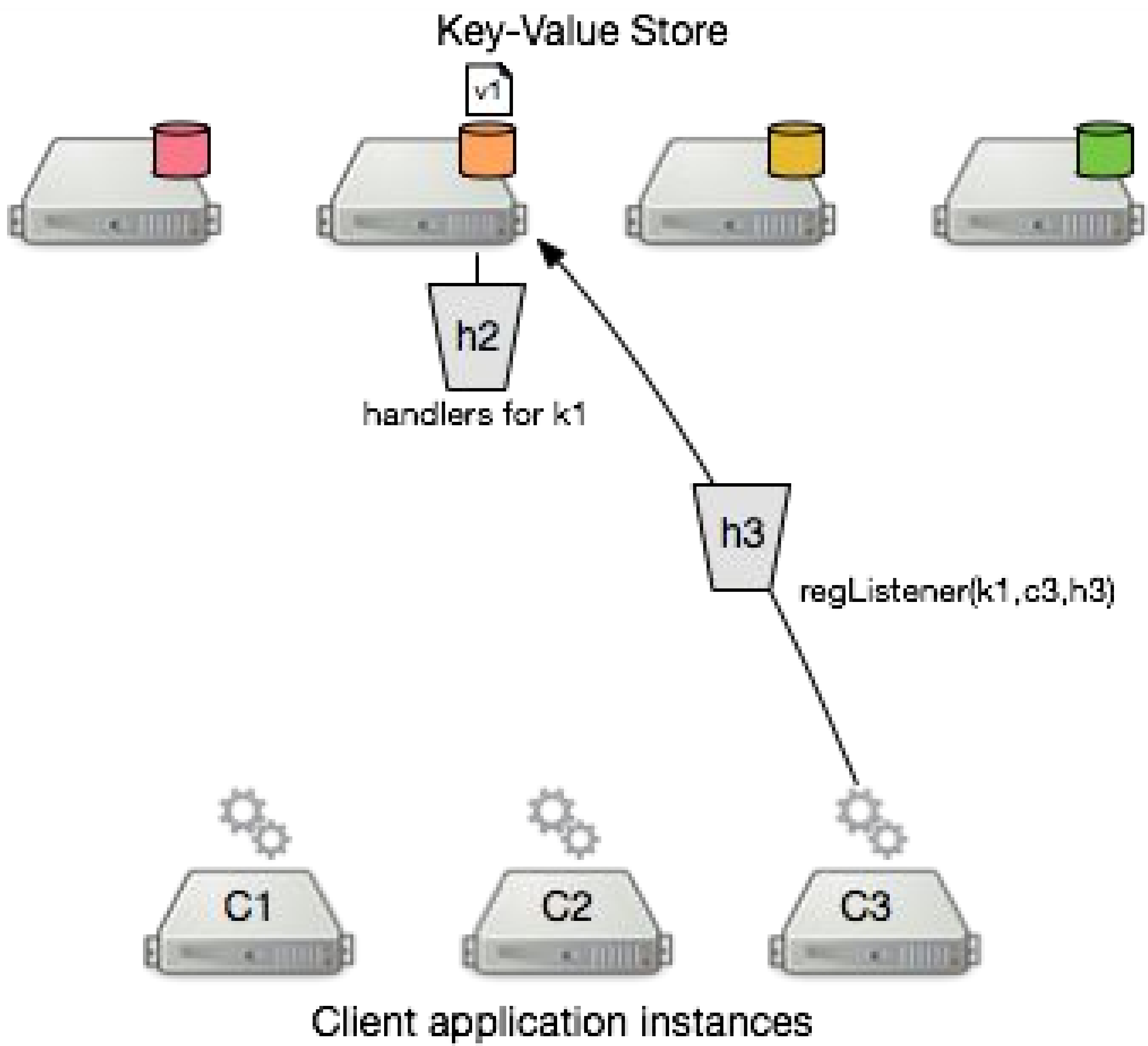
LKVS illustrated



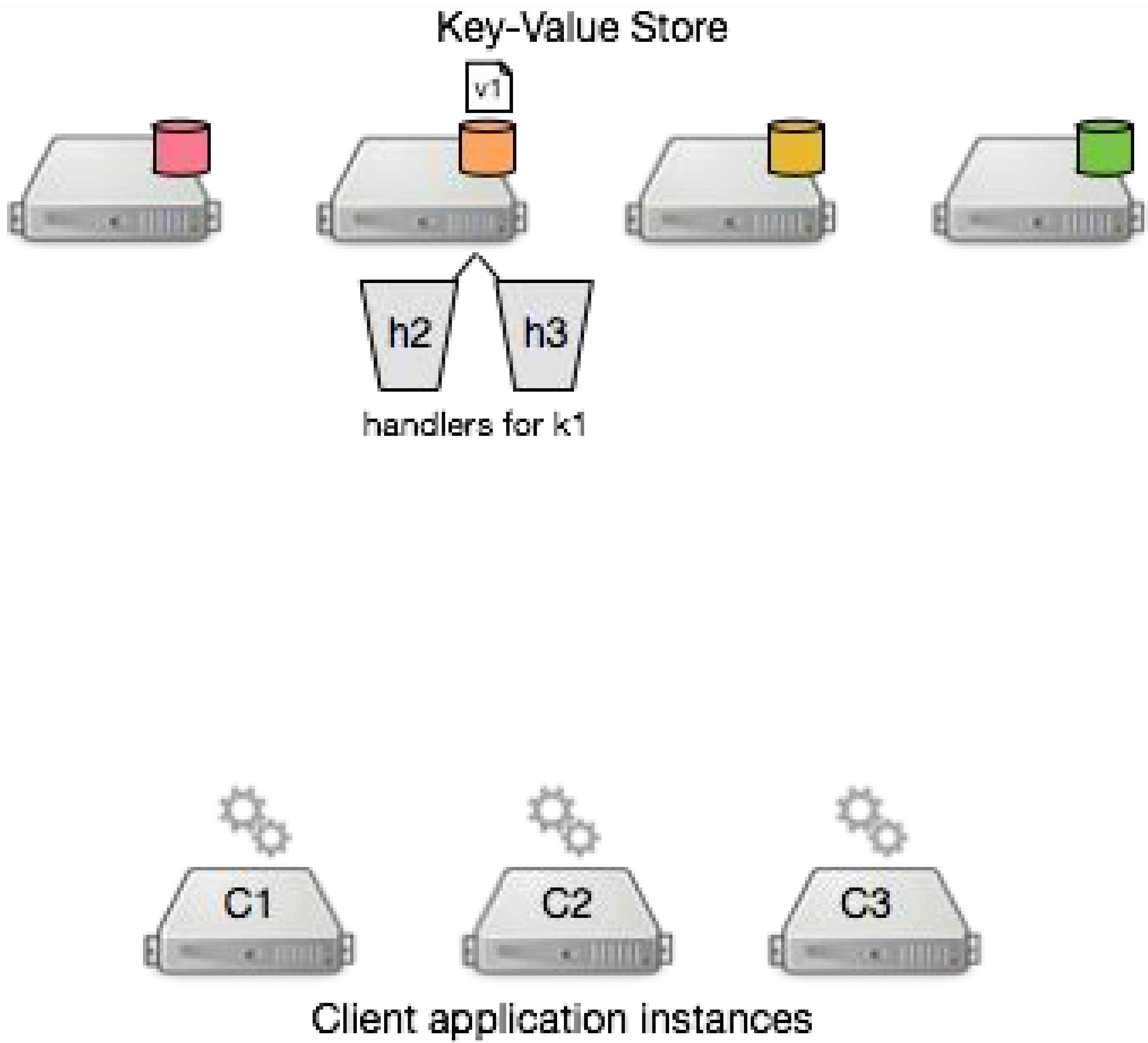
LKVS illustrated



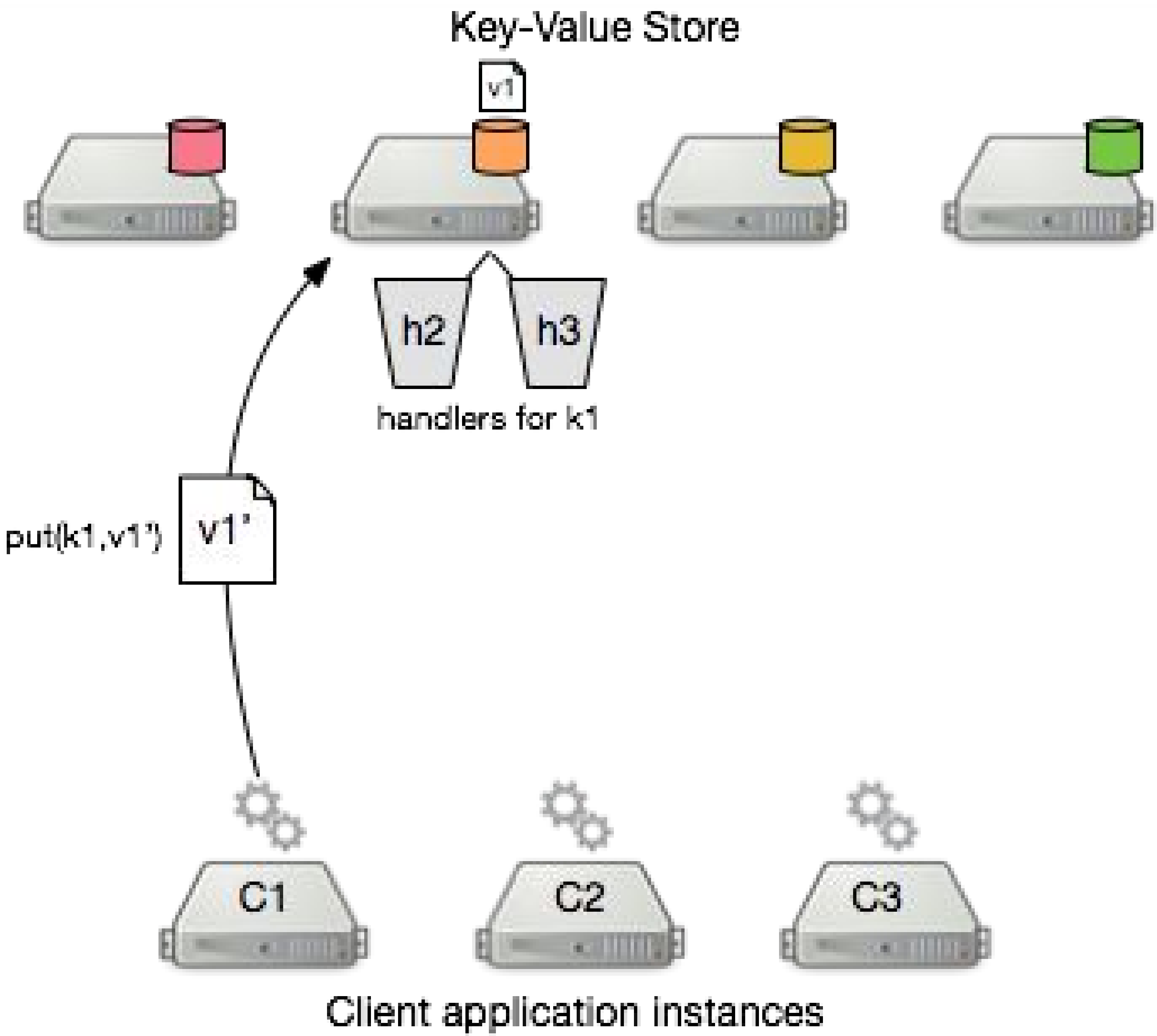
LKVS illustrated



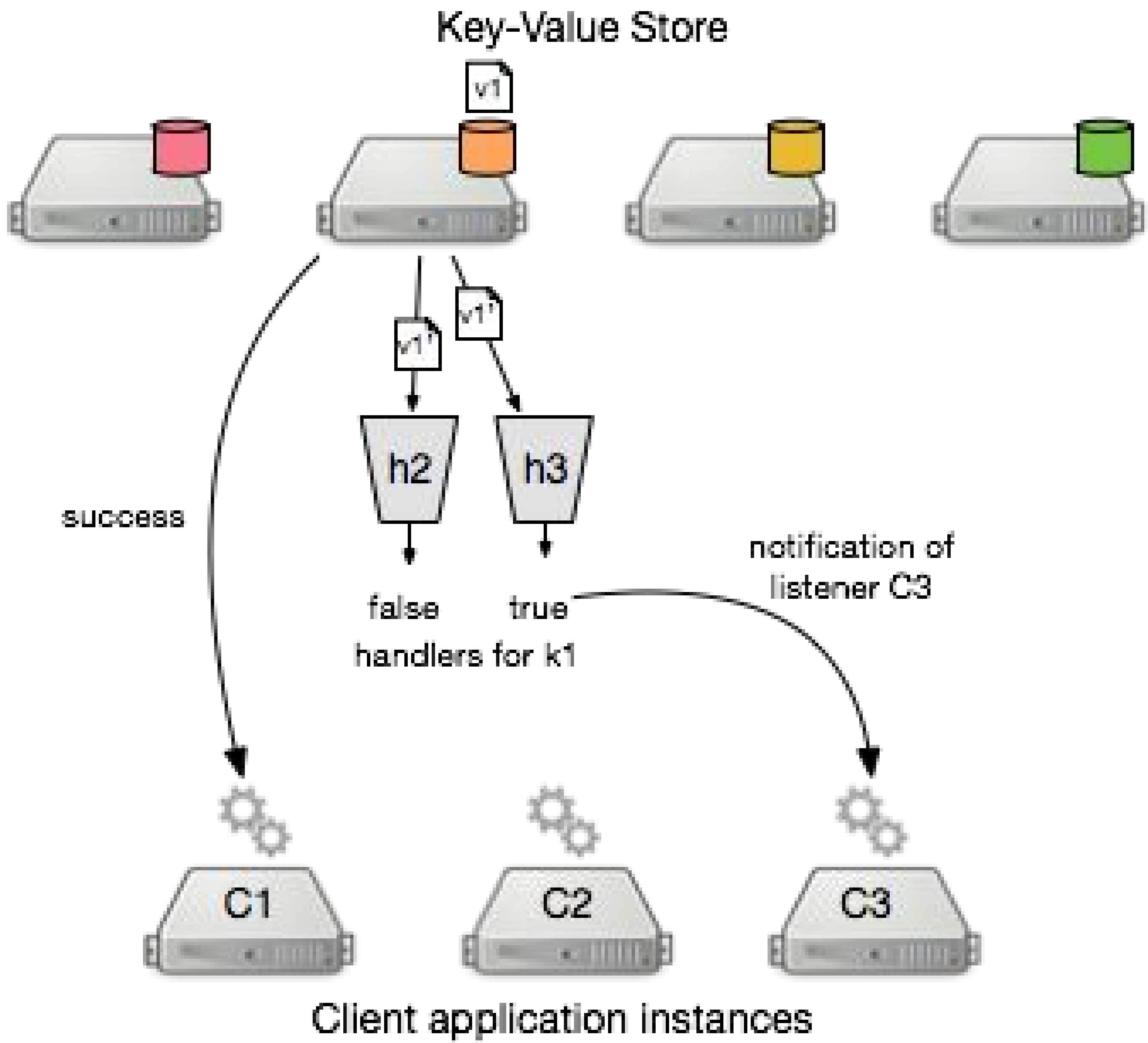
LKVS illustrated



LKVS illustrated

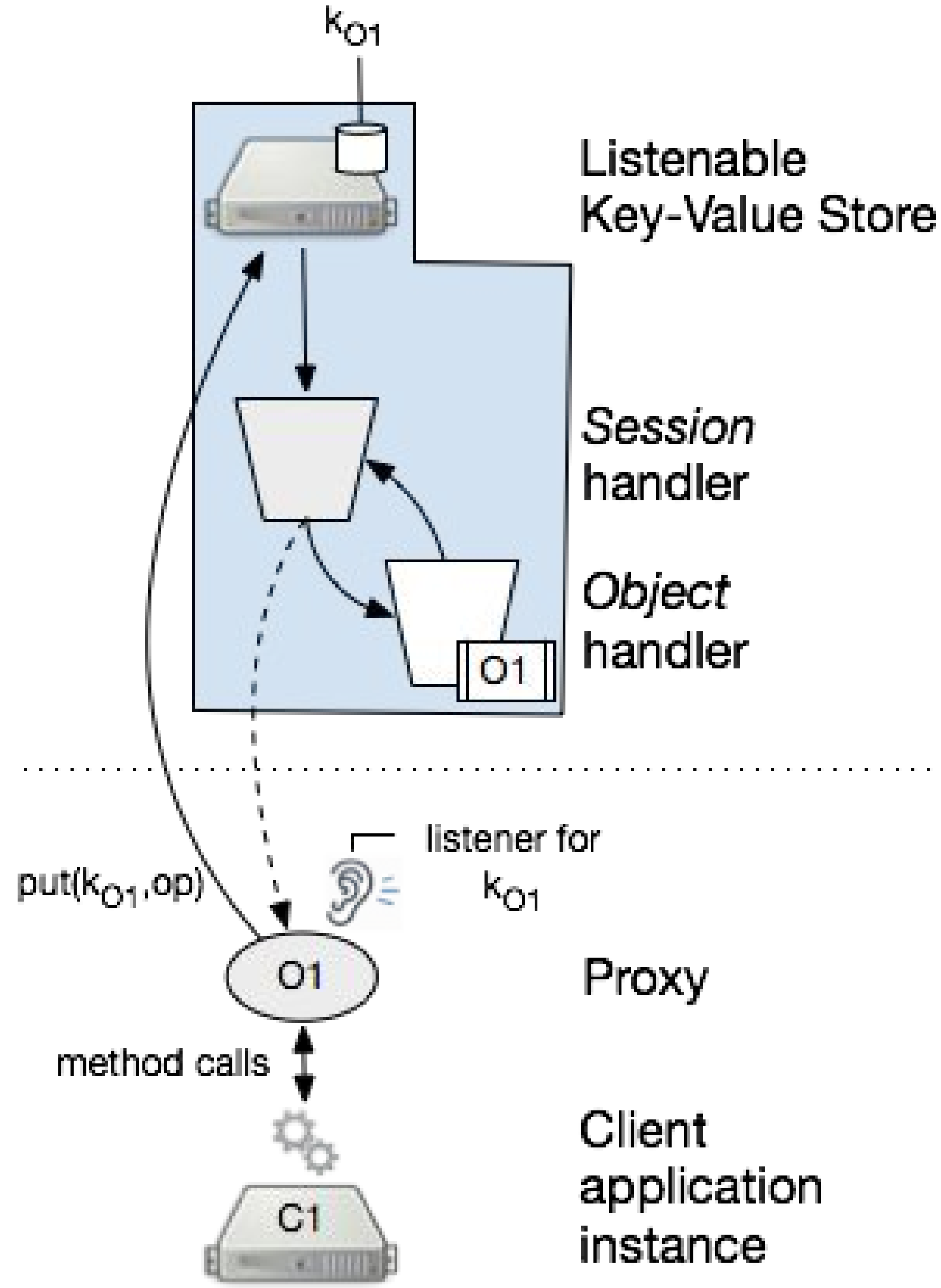


LKVS illustrated



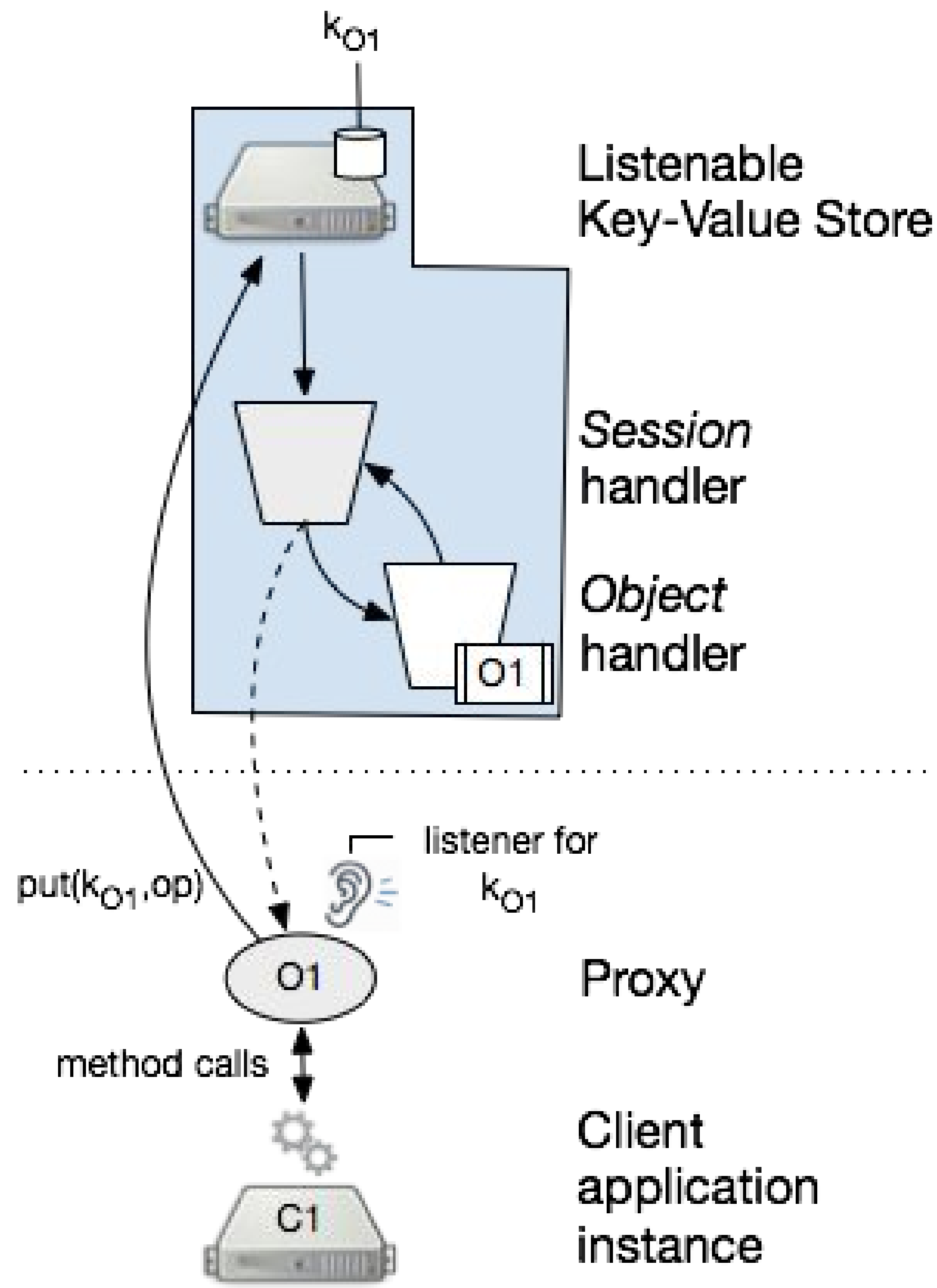
Object management in CRESON (1)

- Client-side *proxy*
- First opening of object for key `k` by a client
  - Not in DB: instantiate new object, server side
  - Serialized in DB: use mapping, server side
- Object closed by last client for key `k`
  - Object serialized, server side, stored in DB
- Method calls and object creation/closing are sent with `put()` calls for key `k`
  - Intercepted by handlers registered with key `k`
  - caller receives the result as a notification



Object management in CRESON (2)

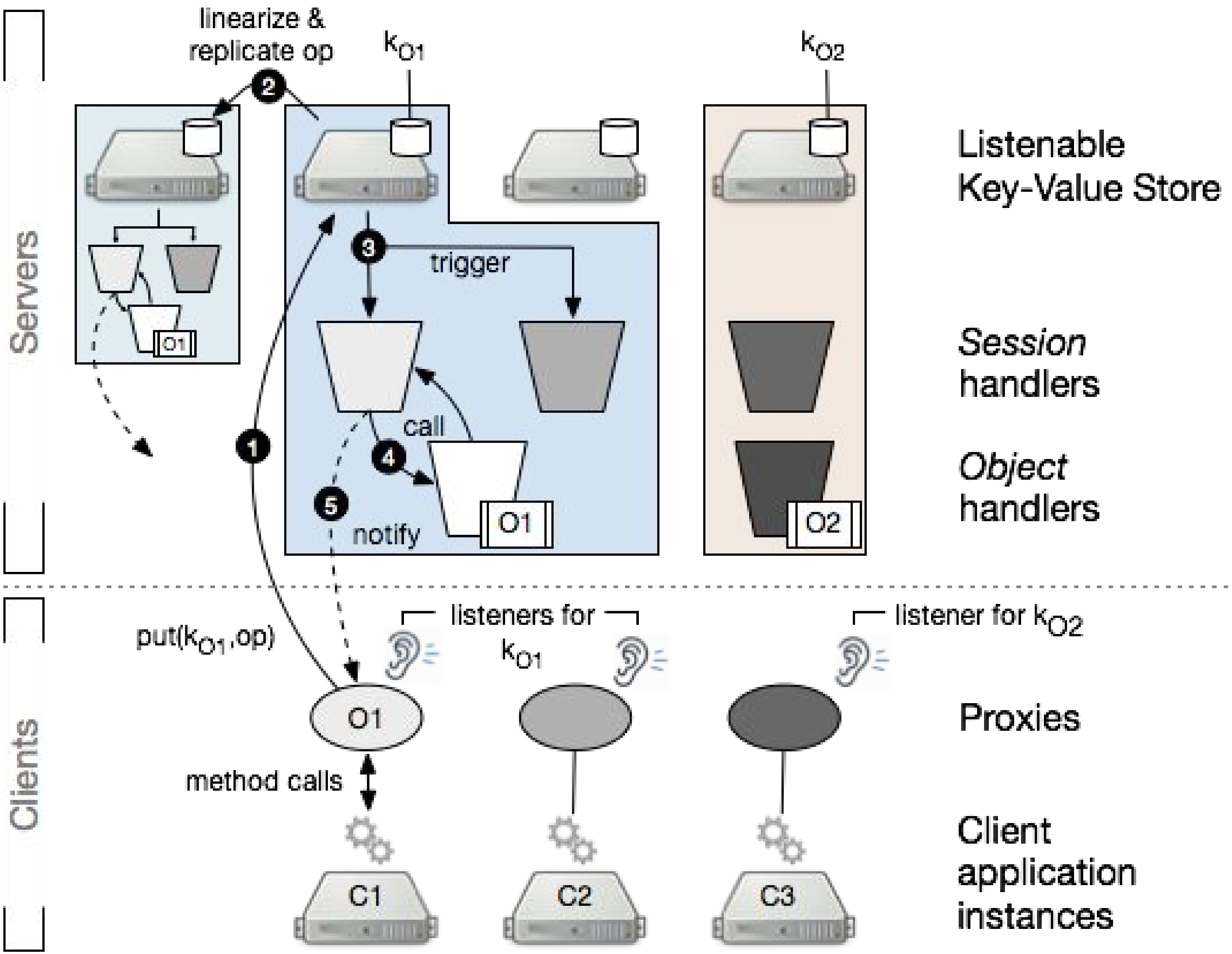
- Two types of handlers for each key
  - One *Session* handler per client
    - Associated with one listener client
    - Ignore operation if from another client
    - Forward to object handler otherwise
  - *Object* handler owns actual object
    - Issues method calls
    - Send return values to session handlers



State Machine Replication

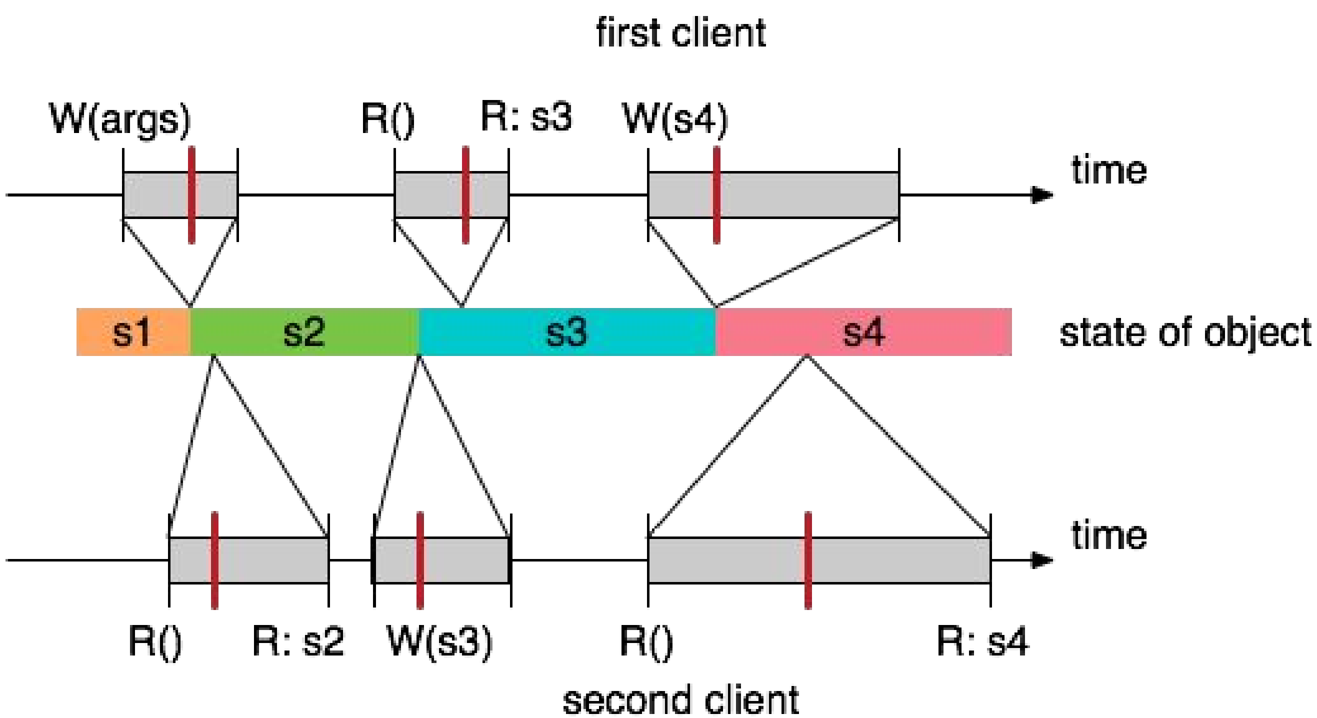
- To survive faults, objects are replicated at the LKVS side
  - Multiple copies of serialized objects
  - Multiple in-memory instances of the same live shared object
- Operation-based replication
  - replicas receive the same stream of operations
  - Order is total,
- Constraint: objects must be *deterministic*
  - Reach unique state from any possible (state, operation) pair
  - Easy to achieve if no use of independent pseudo-random numbers generator

Putting everything together



CRESON guarantees

- ✓ Strong consistency: *linearizability*
- ✓ Wait-freedom for shared objects
- ✓ Composition
  - A shared object can call other objects
  - Maintains linearizability
- ✓ Persistence
- ✓ Disjoint-access parallelism
  - Accesses to distinct objects use distinct LKVS components
- ✓ Elasticity

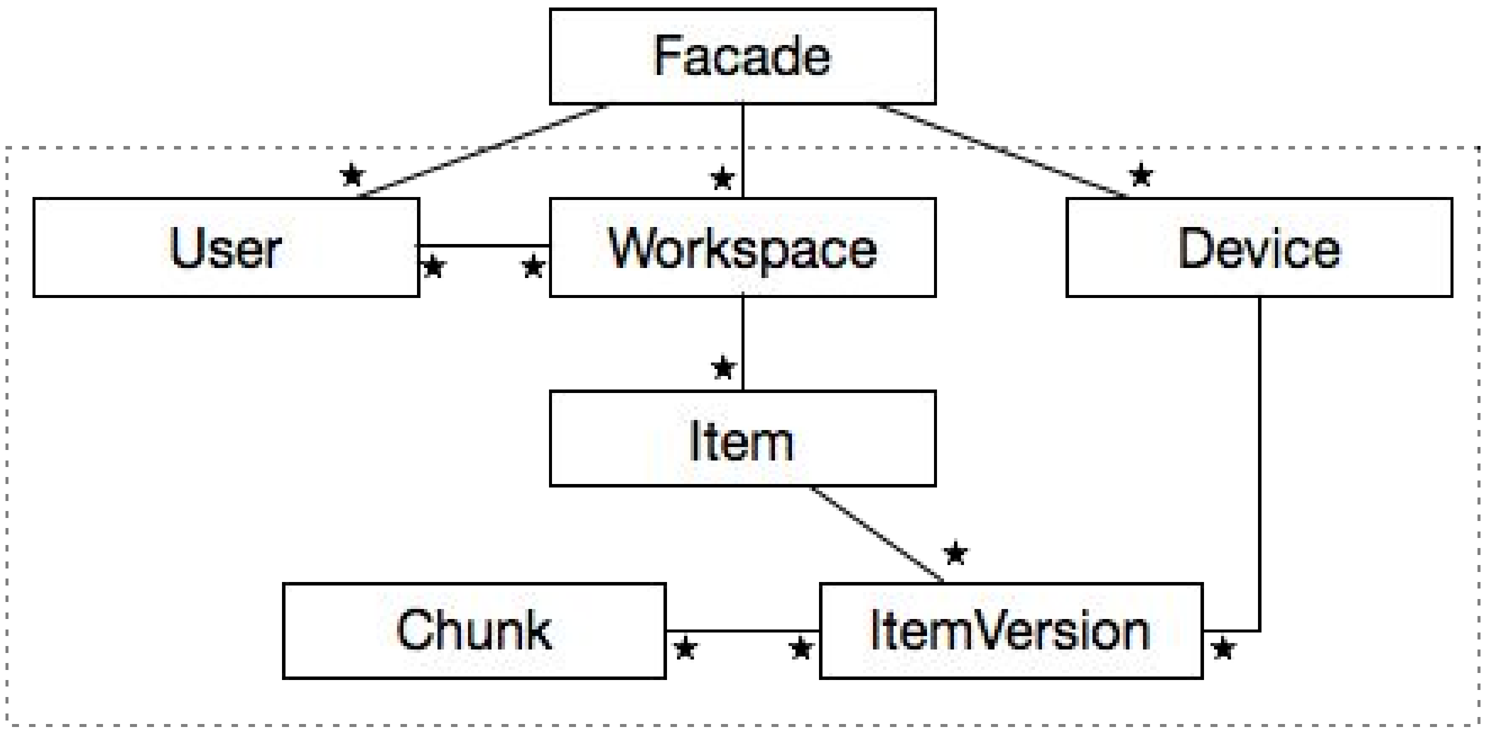


Use case and Interface

- Collaboration with EU project CloudSpaces
  - Open-source Dropbox-like application
  - Synchronization of user file system with cloud-stored file system
  - Sharing of folders and files between users spaces
- Trace collected from Ubuntu U1 personal cloud service
- Data stored in OpenStack Swift
- Metadata requires strongly consistent storage

Original Metadata Management

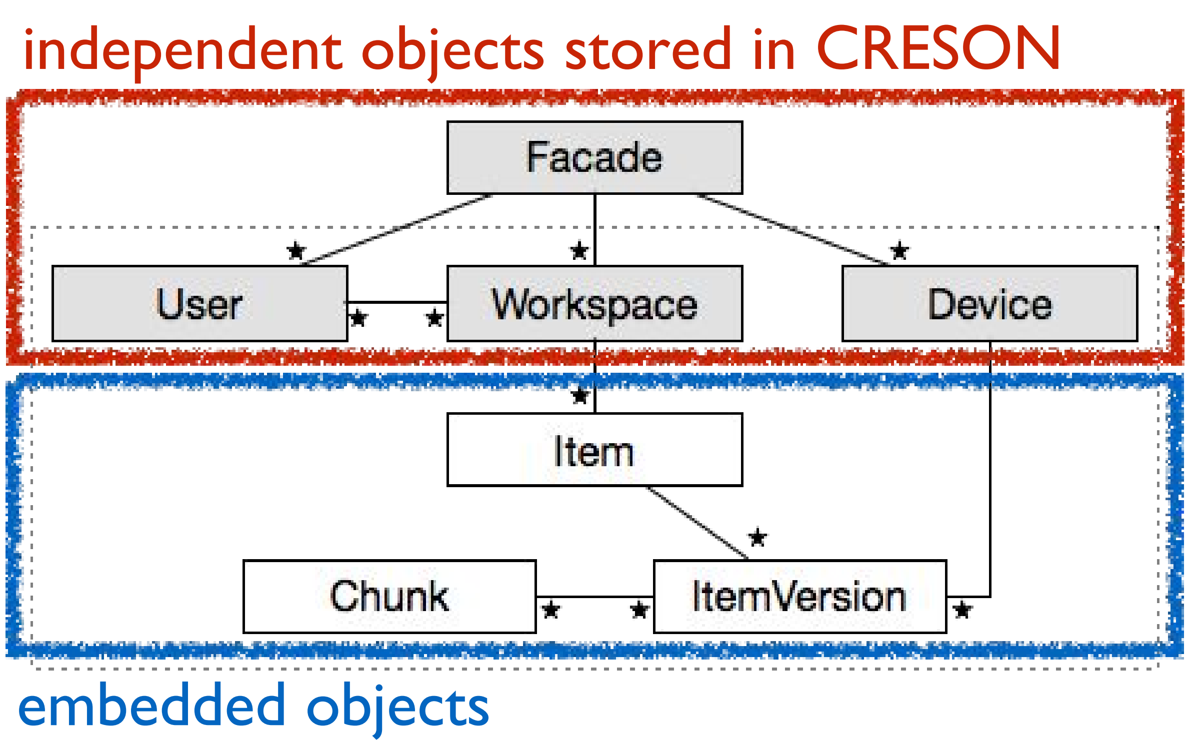
- PostgreSQL relational database
- Performance: use of stored procedures implementing app. logic at server side
- Scalability: sharded (partitioned) database using PL/Proxy
  - No support for elastic scaling
  - No consistency (ACID) guarantees across shards



TELECOM SudParis
INSTITUT Mines-Télécom

Metadata Management with CRESON

- Logic for metadata management re-implemented in plain Java, as methods in StackSync's classes
- Which objects to store in CRESON ?
  - Embedding `Item`, etc. to `Workspace`
- Portage was less than a week of effort
  - Code is simpler and more coherent than with SQL



TELECOM SudParis
INSTITUT Mines-Télécom

CRESON interface

- Integration in Java (using AspectJ)
  - using JPA
- `@Entity(key = "id")` annotation
  - Object o of this class stored in CRESON under key `(classname+":"+o.id)`
  - Store static field in CRESON under key `(classname+":"+id)`
    - Only applies to static fields!
- No further action required from developer
- Shared maps (e.g. deviceIndex) are transparently stored as collections in LKVS

```
@Entity(key = "id")
public class Workspace {

  public UUID id;
  private Item root;
  private List<User> users;

  /* ... */

  public boolean isAllowed(User user) {
    return users.contains(user.getId());
  }
}
```

```
@Entity(key = "id")
public class Facade {

  @Entity(key = "deviceIndex")
  public static Map<UUID,Device> devices;

  @Entity(key = "workspaceIndex")
  public static Map<UUID,Workspace> workspaces;

  @Entity(key = "userIndex")
  public static Map<UUID,User> users;

  public UUID id;

  /* ... */

  public boolean add(Device device) {
    return deviceMap.putIfAbsent(
             device.getId(),device) == null;
  }
}
```


TELECOM SudParis
INSTITUT Mines-Télécom

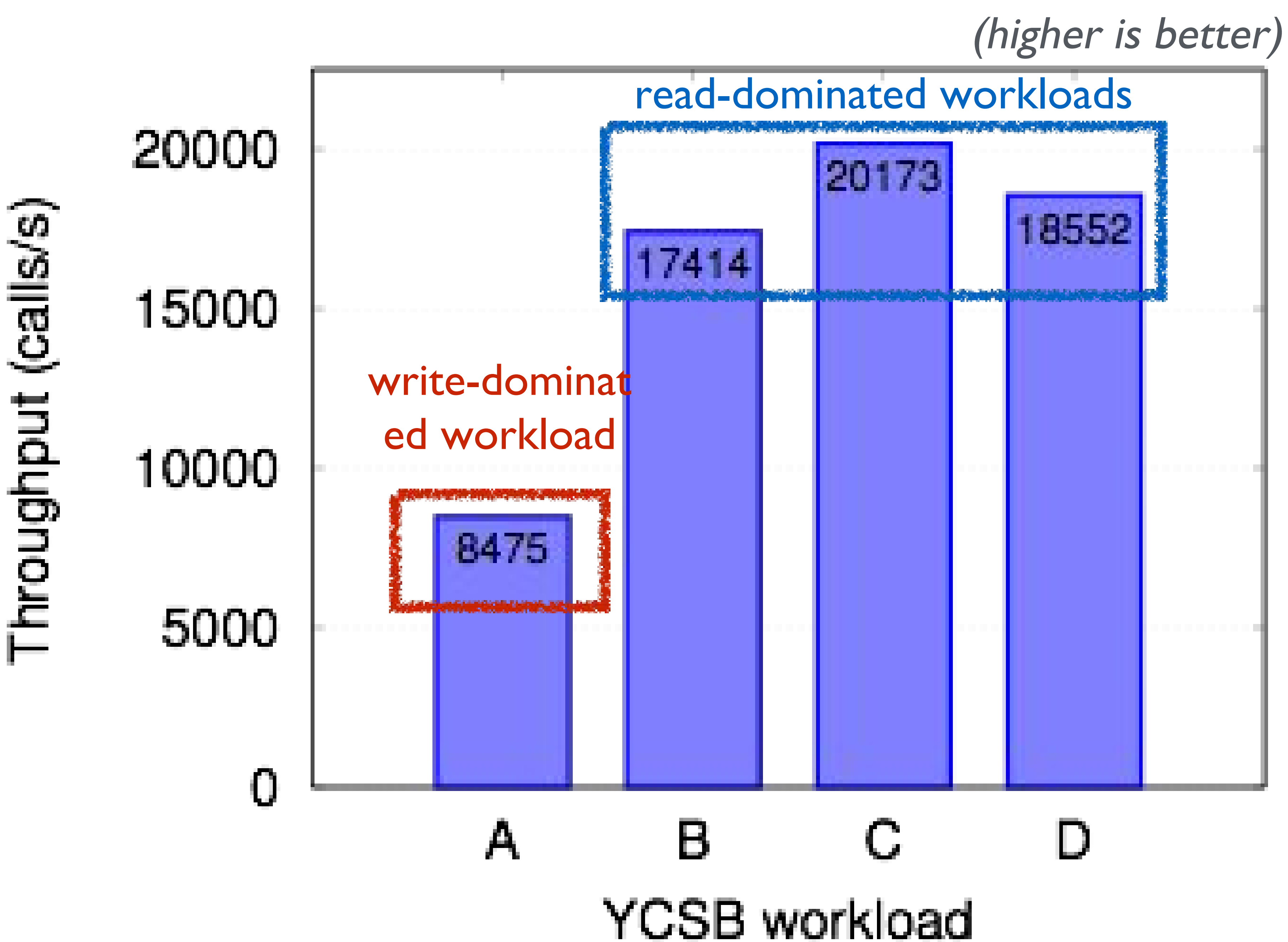
CRESON implementation

- LKVS support added to Infinispan
  - Industrial-grade NoSQL in-memory DB
  - Basis for Red Hat JBoss Data Grid product
  - CRESON integration (staging) as core ISPN feature
- Implementation in Java
  - LKVS = 13,500 SLOC ; CRESON = 4,000 SLOC
- Optimizations (not covered)
  - Listener mutualization
  - Chaining calls idempotency
  - Client-side caching

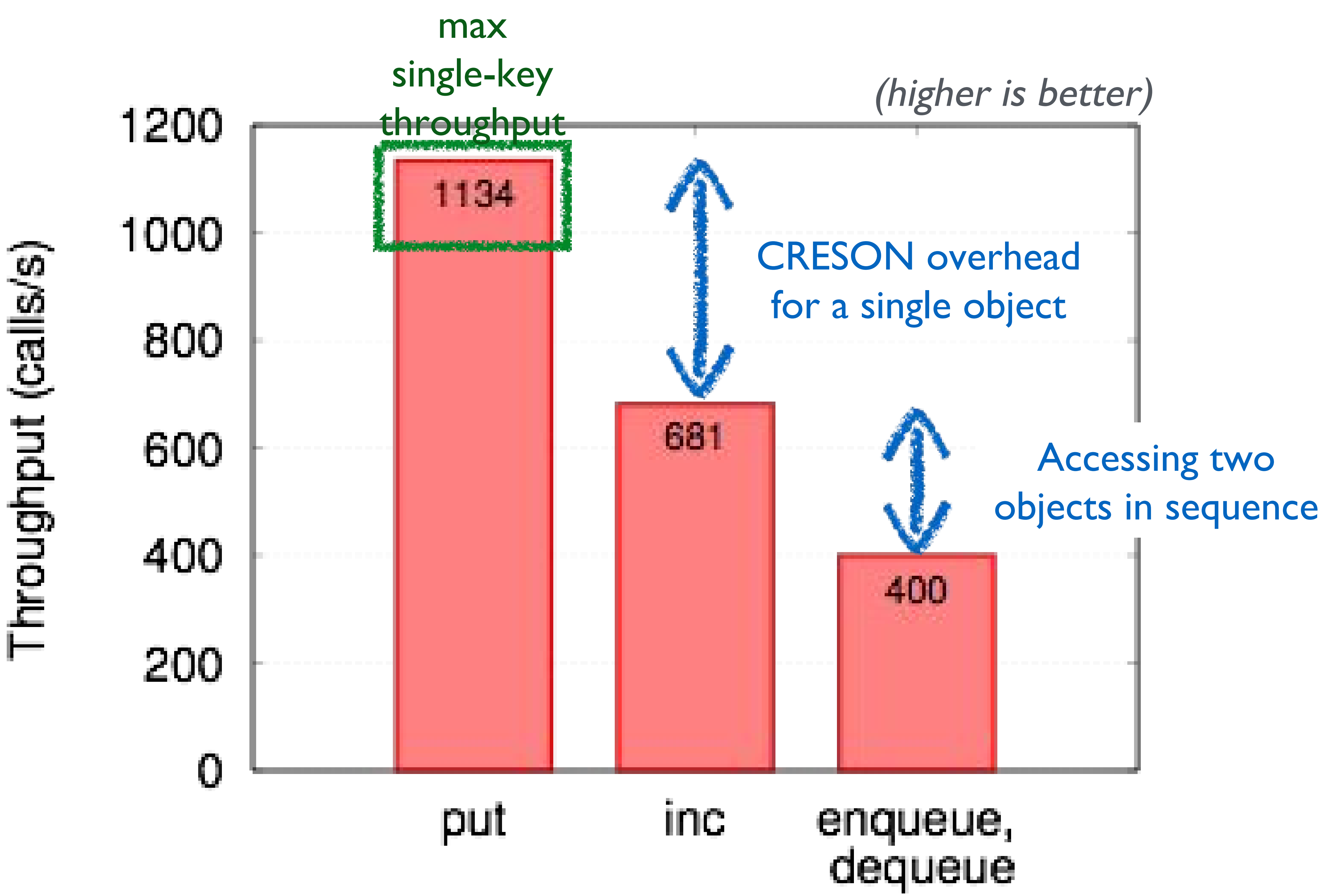
Evaluation

- Cluster of 8-core/8GB Xeon 2.5 GHz, switched 1 Gbps network
- 2 to 6 Infinispan servers (default = 3)
  - Each server maintain a cache of 10^5 recently-used values (serialized objects after their closing)
  - Passivated to disk in the background
  - Replication factor is 2 by default

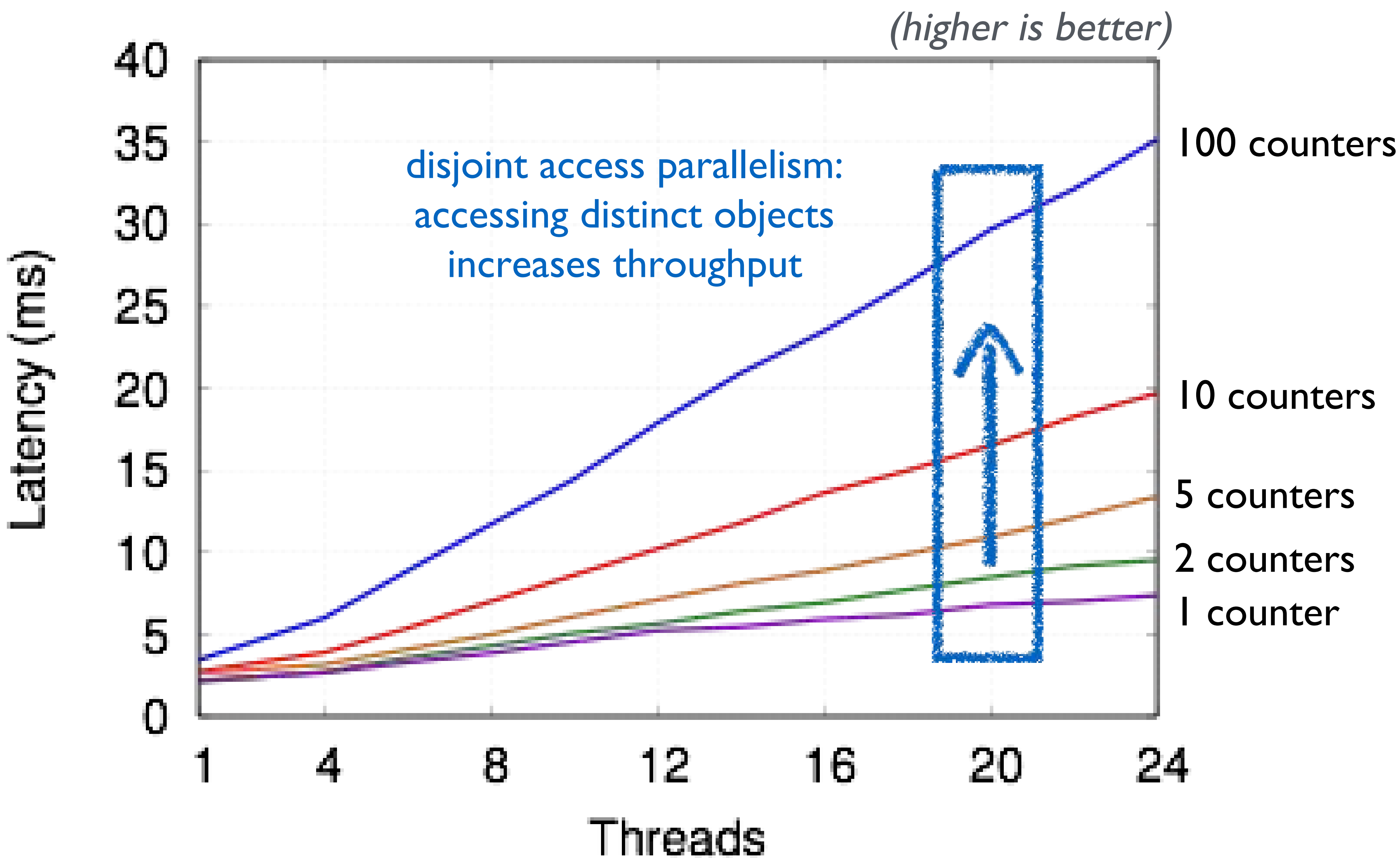
Base Infinispan performance



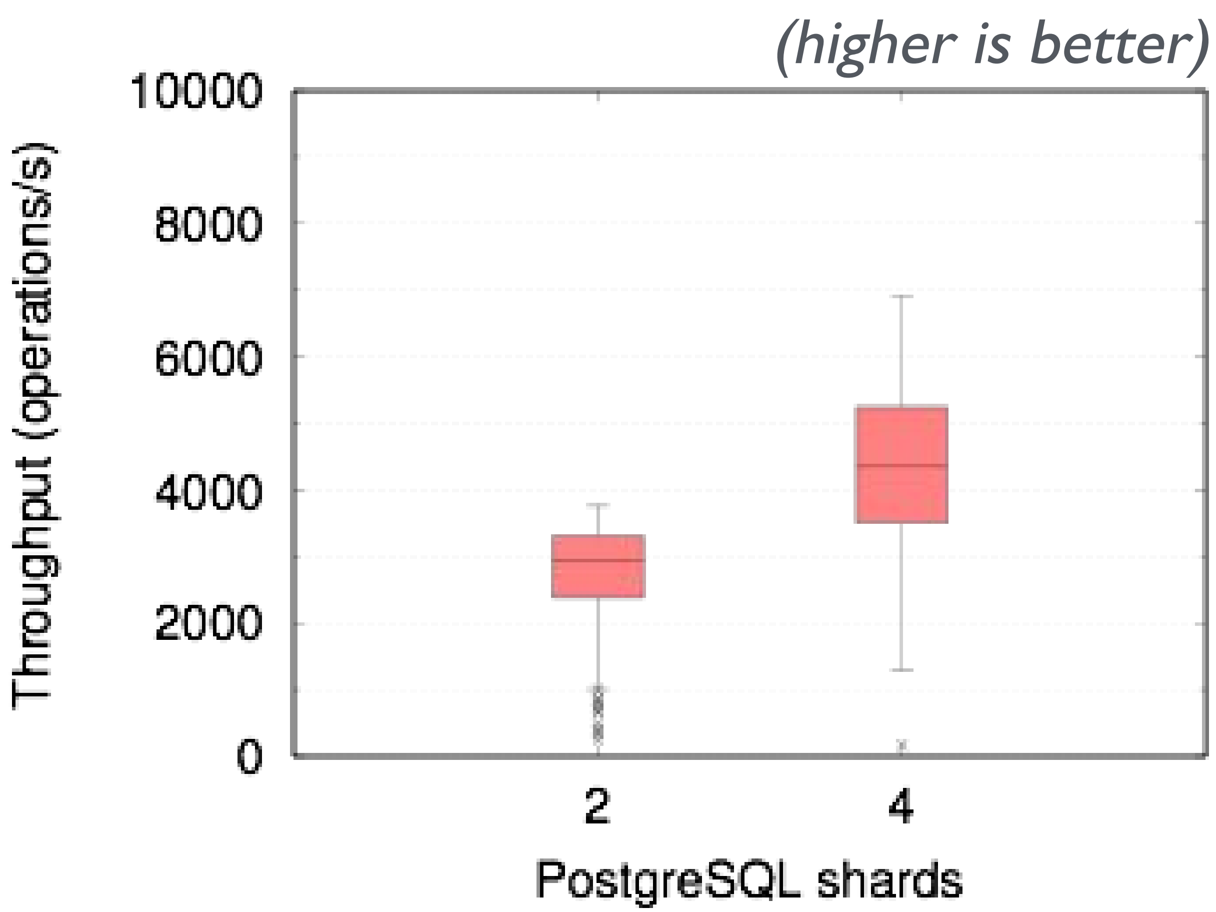
Single-object performance



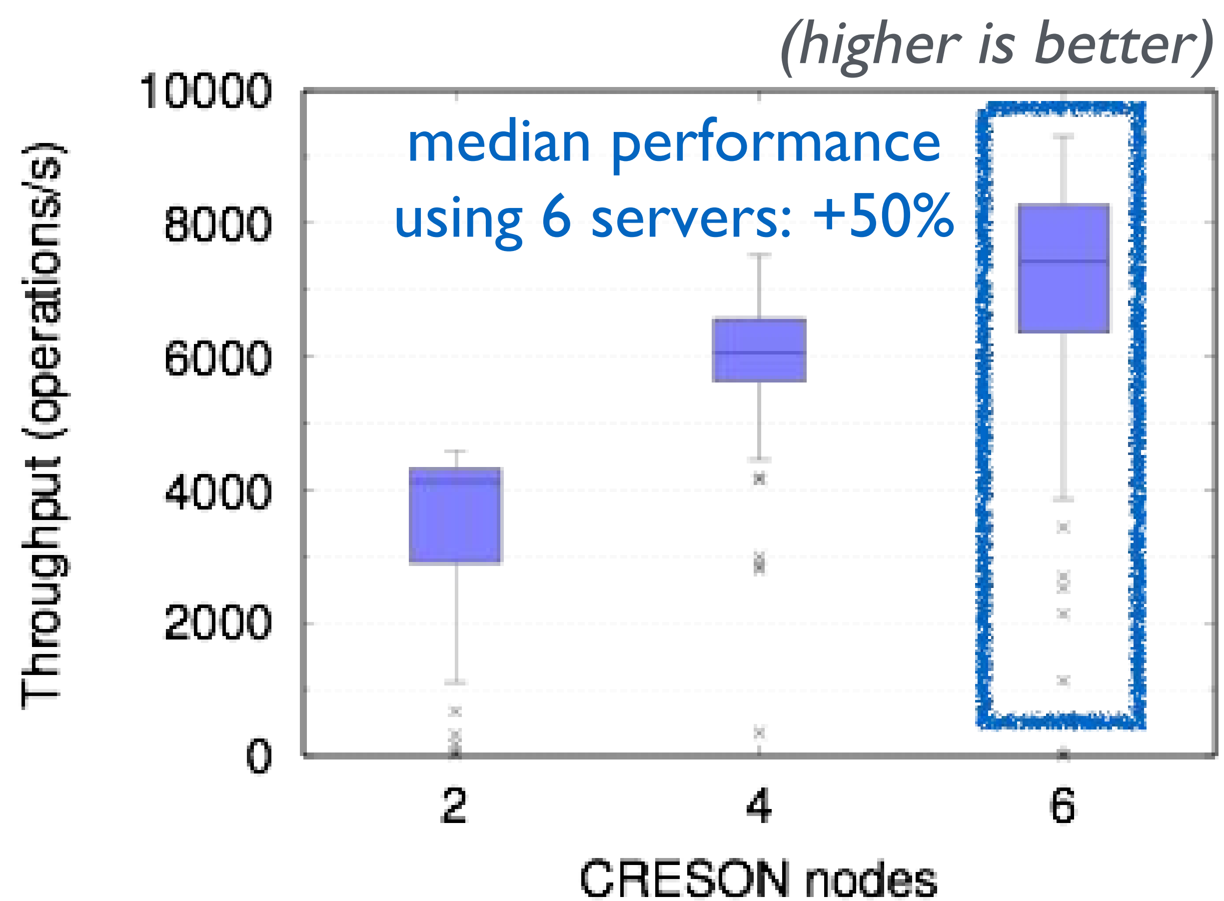
Performance with multiple objects



StackSync performance: throughput

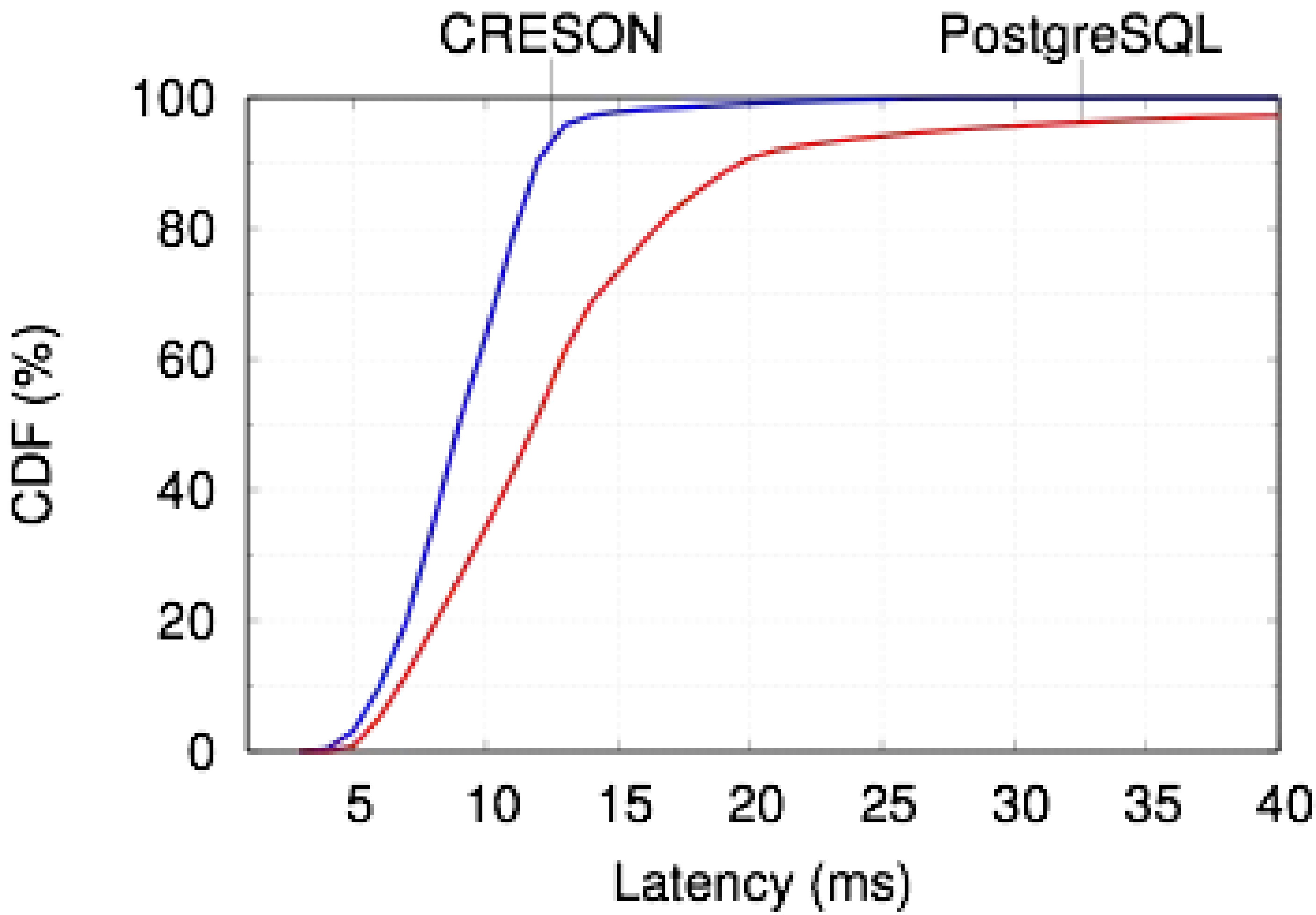


+ 2 additional PL/Proxy nodes



StackSync performance: latency

(leftmost is better)



Conclusion

- NoSQL databases: scalability, elasticity and performance
  but object-SQL mapping is costly

- CRESON = callable shared objects NoSQL
  - Novel LKVS abstraction
  - Simple programming model

- Better performance and elasticity than PostgreSQL

- Future work: support for queries over objects