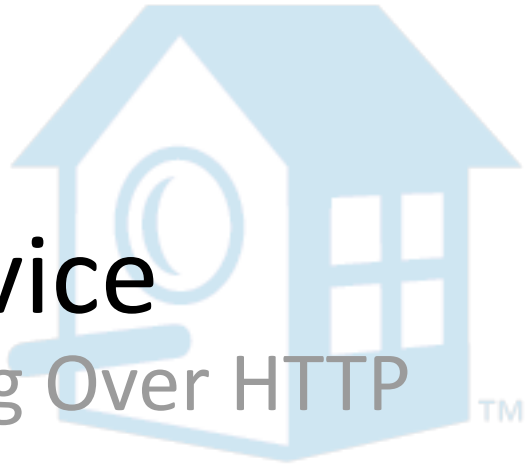
Flaco Lock Service

RESTful Distributed Locking Over HTTP



Jose Falcon

Bryon Jacob

HomeAway®

About Me

- Jose Falcon
- Graduate student @ University of Texas
- Expected graduation: May 2010
- Research interests:
  - Programming languages
  - Artificial intelligence

Outline

- What is synchronization?
- Locks to the rescue!
- Why Flaco?
- Flaco Lock Server
- Flaco Locks
- Reliability With Unreliability
- Demo using cURL
- Flaco Lock Client
- Fitting in @ HomeAway
- Conclusion
- Questions

What is synchronization?

Synchronization

- A problem typically found in multi-threaded (concurrent) environments
- Essentially a problem of coordination
- May appear in many different forms

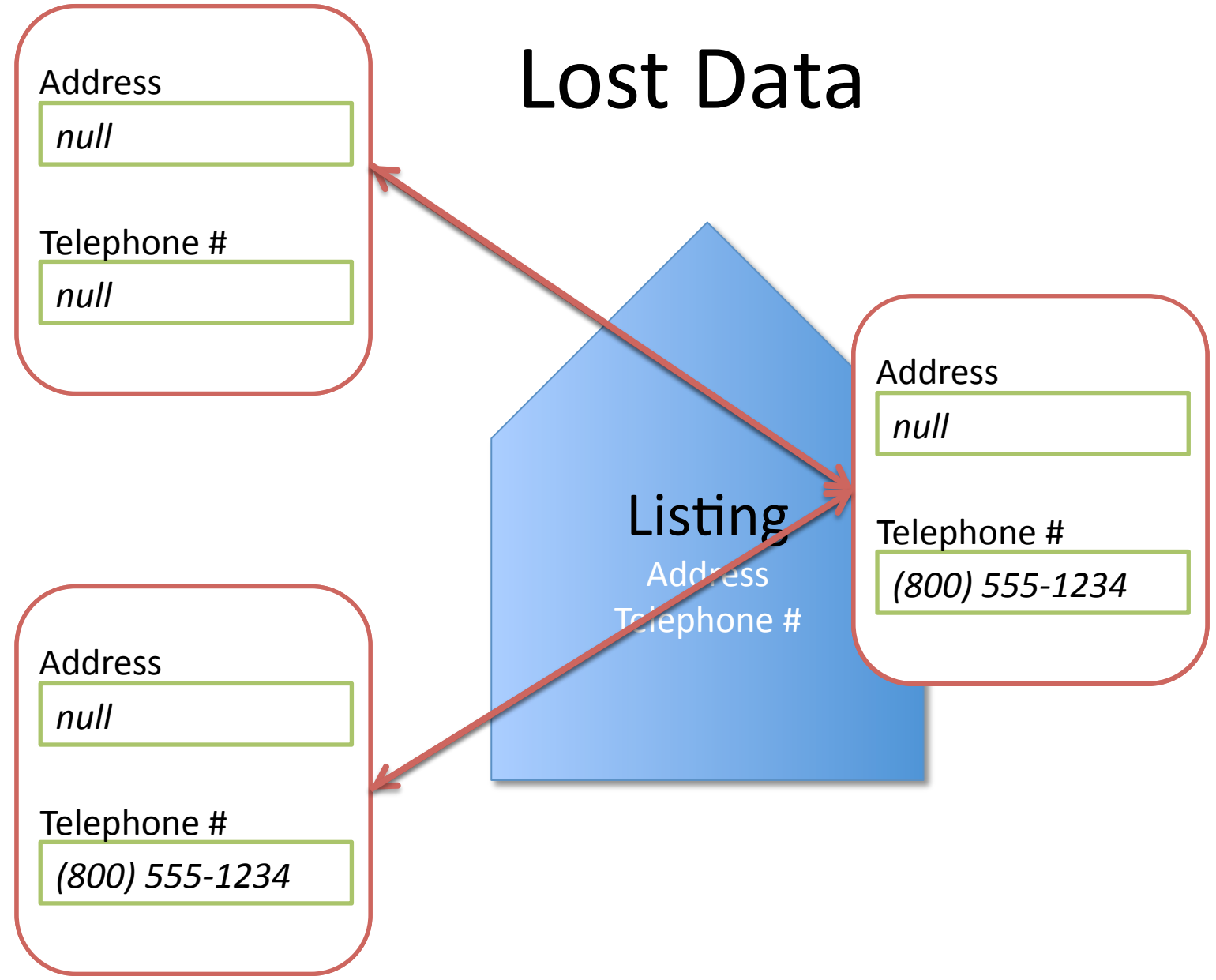
Too Much Milk

What's the Problem?

- Bought too much milk; overworking!
- Collection of "workers", all capable of executing the same task, work without coordination

Single task executed **once** by some worker

Lost Data



HomeAway®

What's the Problem?

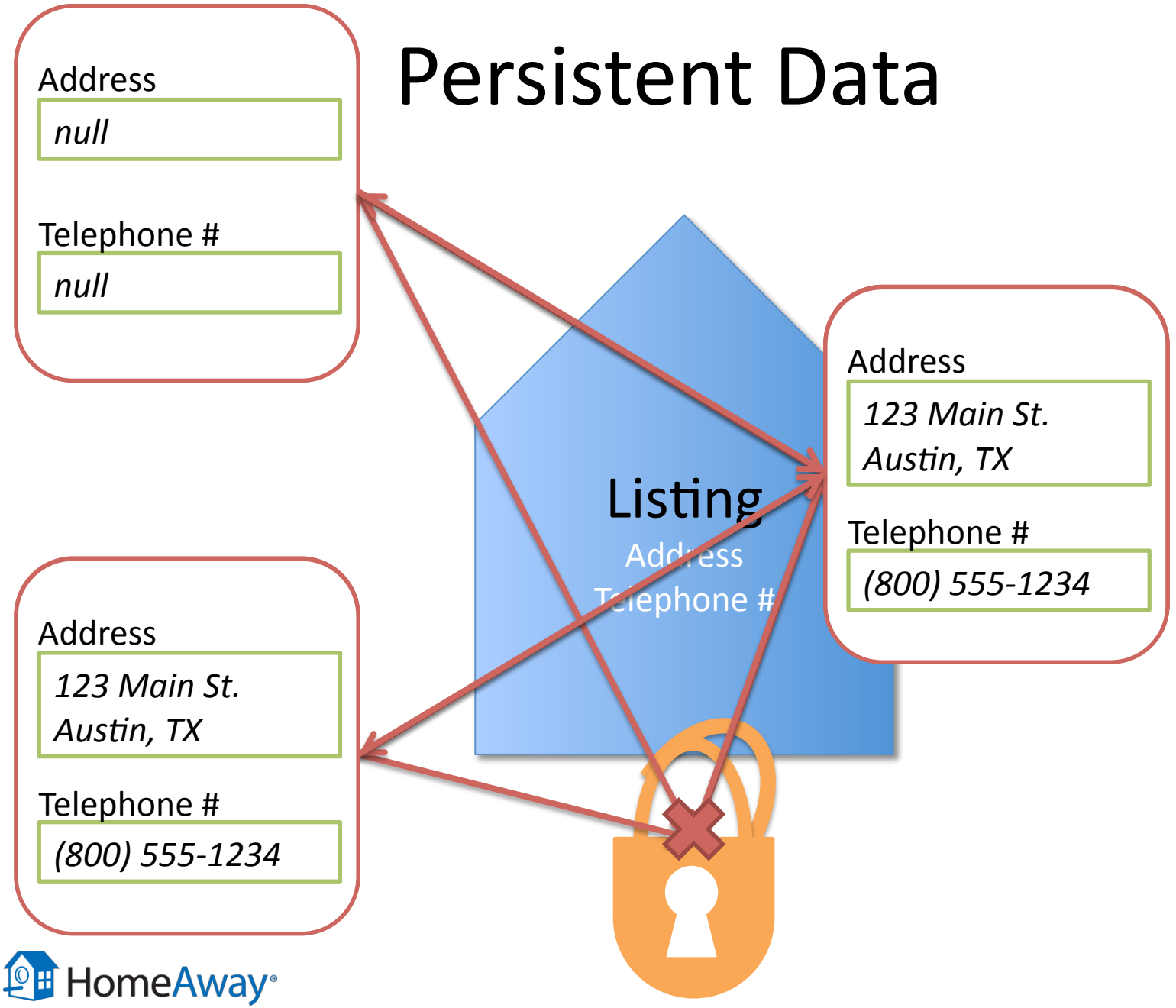
- Accessing "shared object" without considering changes made by other "users"
- The data is inconsistent and out of date!

Prevent "users" from accessing stale data

Locks to the rescue!

Not Too Much Milk

Persistent Data



HomeAway®

Locks

- One solution to the synchronization problem
- Simple and intuitive
- Requires:
  - Atomic lock/unlock operations
  - Extreme reliability
  - Cooperation
    - All users must respect the lock!

Why Flaco?

HomeAway®

Current Solution

- At HomeAway locks are created, on demand, in a database
- Problems:
  - Burden to create
  - Requires database connection
  - Overhead for simple applications which don't rely on a DB

What We Want

- RESTful service for easily creating / sharing locks across multiple applications
- Usable in any application
- No dependency on:
  - Database
  - Any specific language
- Additionally:
  - Java client library for easily using Flaco

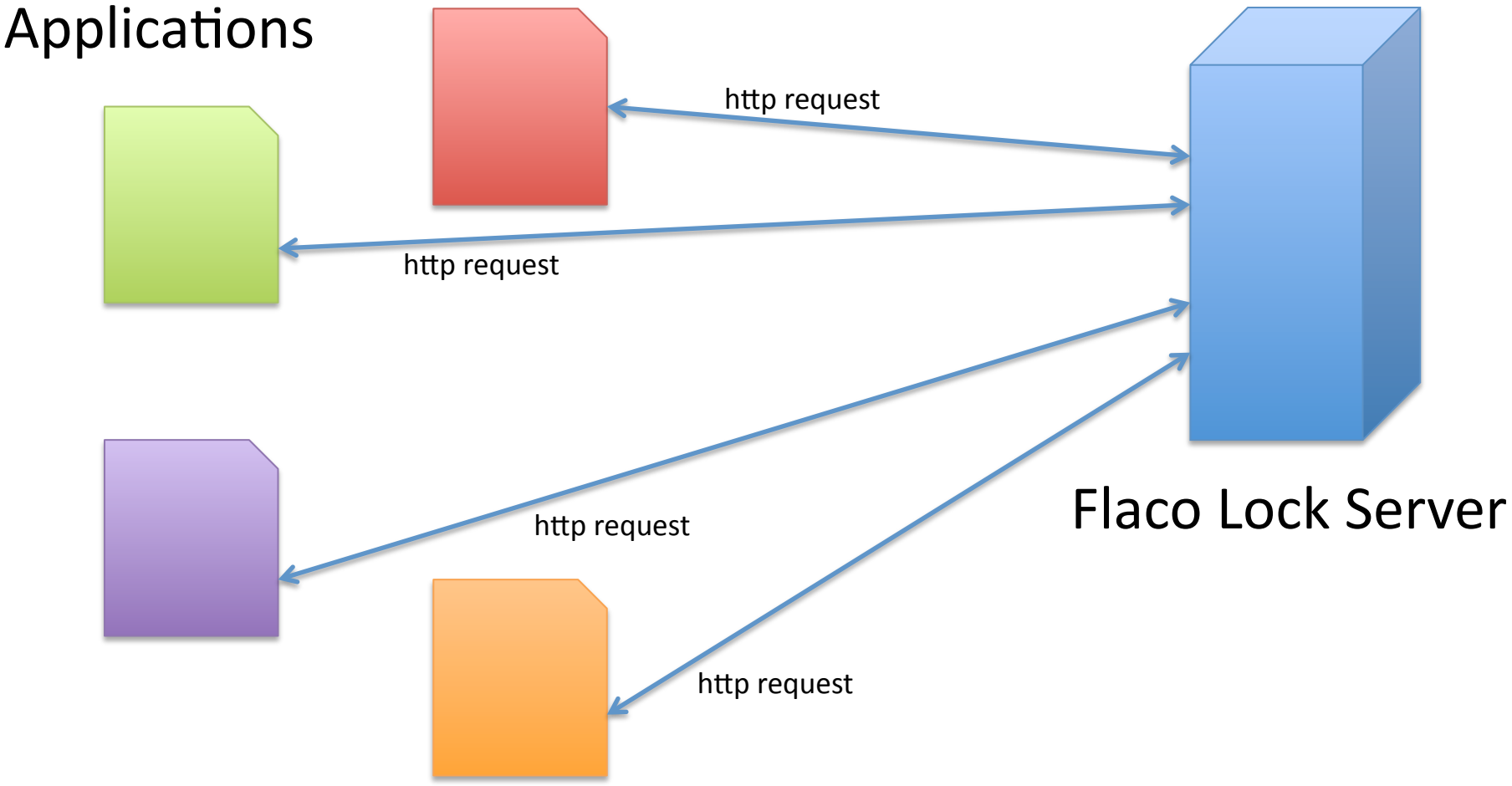
Flaco Lock Server

HomeAway®

Flaco Lock Server

- Centralized service for manipulating logical named locks
- Simplifies the creation of locks
  - All possible locks logically “exist”
- Stores locks in RAM, no need for a database
- Convenient access through REST API
  - Any program/language supporting HTTP requests can synchronize using Flaco

Flaco Lock Service



Flaco Server Details

- Asynchronous
- Basic operations:
  - inquire
  - acquire
  - renew
  - release
  - wait
  - signal
  - signalAll

Inquire

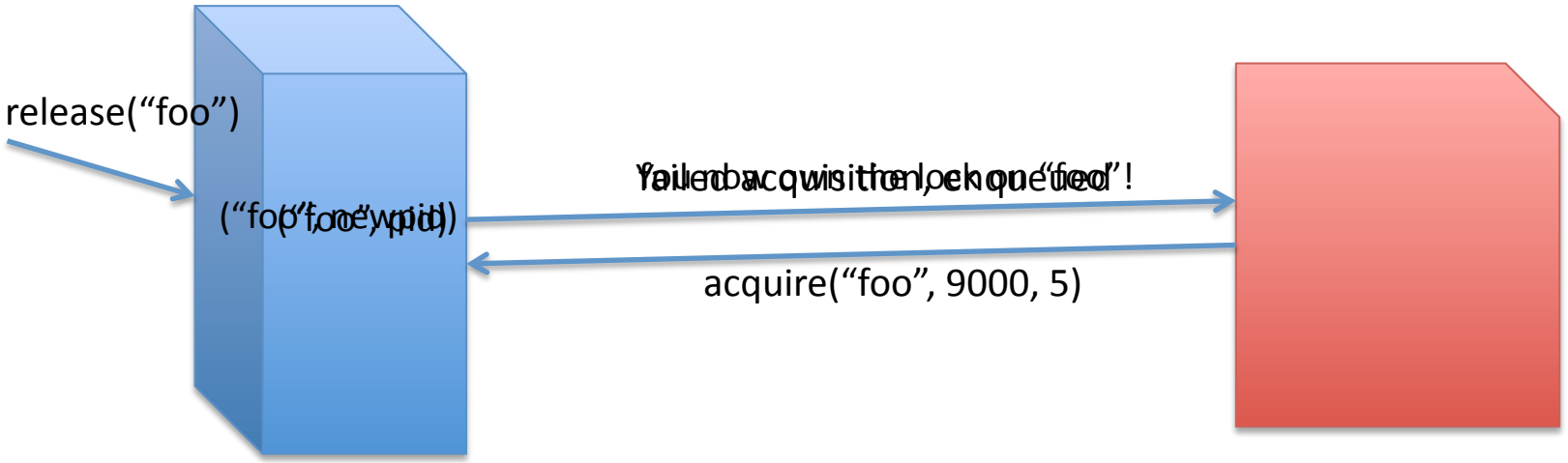
- Returns information regarding the current state of the lock
  - Current owner
  - Current waiters on the “ready queue”
  - Current waiters on “condition queues”

Acquire

- Two types of acquisition:
  - Poll
  - Callback
- Poll
  - Simply awards the lock if it is available
- Callback
  - Timed acquisition with a callback server

Callback Acquire

- If the lock is available, award ownership
- Else queue the requester in the “ready queue”
- The server will notify the requester when she is eligible for acquisition at the given callback



HomeAway

Renew

- Limited time ownership
- Must renew lease prior to lease timeout
- Failure to renew results in auto-release
- Ensures liveliness of owner

Release

- Removes current owner
- Awards ownership to next “alive” waiter
- Sends notifications to callback URLS

Condition Variables

- Implicit condition
  - All locks have an implicit condition
- Explicit condition
  - Support arbitrary conditions on names
- Allow complex interactions between owners

Wait, Signal/All

- Wait
  - Removes ownership of the lock
  - Placed on condition “wait queue”
- Signal/All
  - Removes single/(all) waiter(s) from the condition “wait queue” to the “ready queue”

REST API

- Base URL:
  - http://flaco.homeaway.com/api/<lockName>
- GET
  - Inquires about the state of the lock
- POST
  - Mapped to various actions:
    - “acquire”
    - “renew”
    - …

Acquisition Status

- Determine if a given PID is still pending acquisition
- Base URL:
  - http://flaco.homeaway.com/api/status/<PID>

Flaco Locks

Flaco Lock Details

- Lock security
- Reentrancy
- PID may not “multi-block”
- Hierarchical locks

Lock Security

- Public/Private key paradigm
  - PID/Credentials
  - Each “entity” that acquires a lock must have a server unique PID
  - Manipulations on a lock require lock credentials

Reentrant Locks

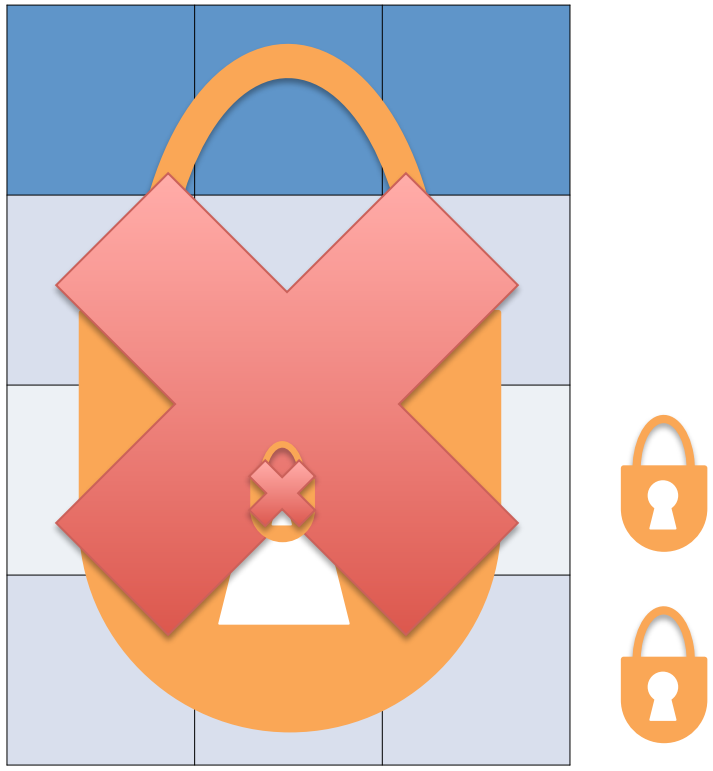
- Requests for the same lock by the same PID increment a reentrancy depth
- Releasing a lock decrements this count
- Locks are released when the depth reaches zero

Multi-Block

- Owners that are logically blocked cannot function as if they are not
- May not:
  - Wait twice on the same lock
  - Acquire/wait on another lock

Hierarchical locks

trips

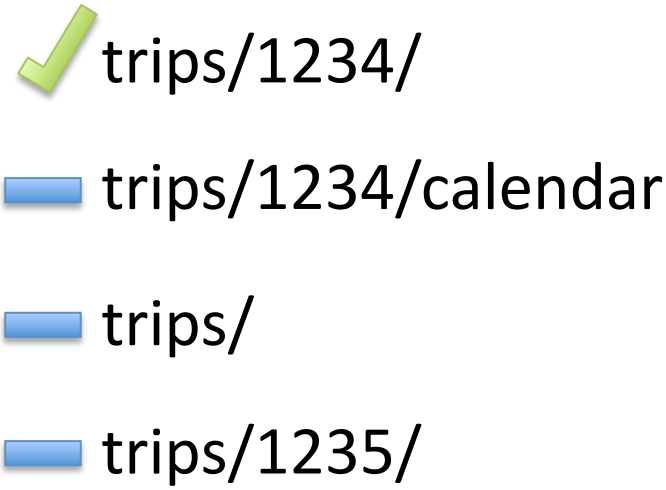


✓ trips/1234/

✓ trips/1234/calendar

Fair!

- Locks are handed out according to when they were requested!
- Removes possibility of starvation



Reliability With Unreliability

Problem

How do we maintain lock reliability with an inherently unreliable network connection?!

Reliability

- Not unique to Flaco!
  - All distributed locking systems using a network connection will encounter this problem
- What if…
  - A client stops communicating?
    - Then she never renews her lease and the server will remove her as owner (at some point), guaranteed
    - Ensures progress
  - The server goes down?
    - A backup server picks up where the other left off

Demo using cURL

Flaco Lock Client

What We Want

- Java library for interacting with Flaco
- Abstraction of the REST API
- Replace “local synchronization” with “distributed synchronization” without significant code change

Lock Client Interface

- A lock client interacts with a lock server
- Single method:
  - getLock(lockName)
- Returns a distributed lock object for easy manipulation

Distributed Lock Interface

- Basic interface:
  - lock()
  - lockInterruptibly()
  - tryLock(timeout)
  - unlock()

Standard Java Lock Interface

  - getNamedCondition(conditionName)
  - doSynchronized(callable)
  - check()

Java Client API

- Java implementation handles all interaction with the Flaco Server
- Maintains an “auto-renew” thread
- Manages internal “callback-server”

getNamedCondition

- Distributed version of Java's getNewCondition by providing names to conditions
- Returns a distributed condition object for the provided name
- Prevents "busy-waiting"

doSynchronized

- Distributed version of *synchronized* keyword
- Executes some computation within the scope of the lock

```
lock.lock()
lock.doSynchronized(new Callable<Void> {
    // perform some complicated action
});
    lock.unlock();
}
```

check

- Determines if the lock object still holds the lock at the server

Code Snippet

```
LockClient lc = new FlacoLockClient("flacoserver");
DistributedLock myDistrLock = lc.getLock("foobar");
```

```
myDistrLock.lock();
try {
    // set the phone number to (800) 555-1234
} finally {
    myDistrLock.unlock();
}
```

Standard "locking" template in Java

Interrupting Code Execution

- Imagine a broken connection with the server
  - How can we interrupt the executing code?!
- Possible, though difficult
- Rely on throwing exceptions on lock manipulations to break execution
  - All operations on a lock or condition variable may throw a *LostLockException* indicating that the lock has been lost at the server

Example

```
LockClient lc …

myDistrLock.lock();
try {
   distributedCount++;
   // do some complicated action here
   myDistrCond.signalAll()
} catch (LostLockException e) {
   // undo some complicated action here
   distributedCount--;
} finally {
   myDistrLock.unlock();
}
```

Fitting in @ HomeAway

Advantages of Flaco

- Extremely easy to use
- No database dependency
- No language dependency
- REST API
- Performance benefits (hypothesis)

Flaco @ HomeAway

- "Core" service
- Improve reliability of feed processing
- Expose synchronization to scripts
- Use in *Mesa*
  - Distributed file system currently being built
- Outside of HomeAway
  - Flaco will be open source

Thanks

- Bryon Jacob (*mentor)*
- Alex Victoria
- Raul Mireles
- The entire services team
- HomeAway

Thank you!!

Questions??