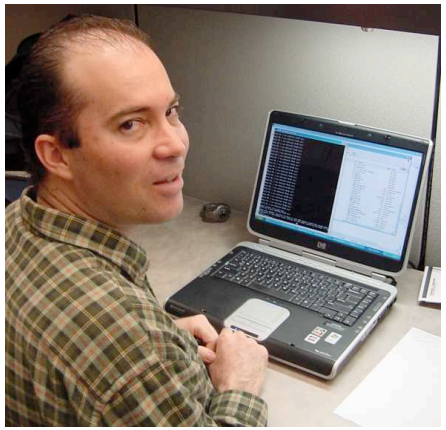
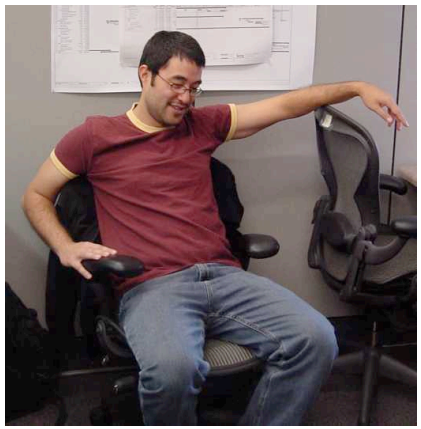
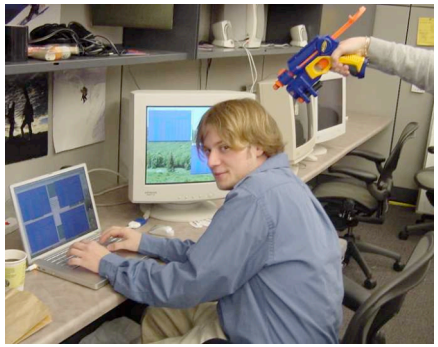
Team SSDMSE06: SuDuelKu

18-749: Fault-Tolerant Distributed Systems

Saul Jaspan, Lucia de Lascurain, Luis Rios, Yudi Nagata, & Christopher Nelson

Electrical & Computer ENGINEERING

ISRI

Team Members

Lucia de Lascurain

ldelascu@andrew.cmu.edu

Saul Jaspan

Saul.Jaspan@gmail.com

Yudi Nagata

ynagata@andrew.cmu.edu

Christopher Nelson

crnelson@cs.cmu.edu

Luis Rios

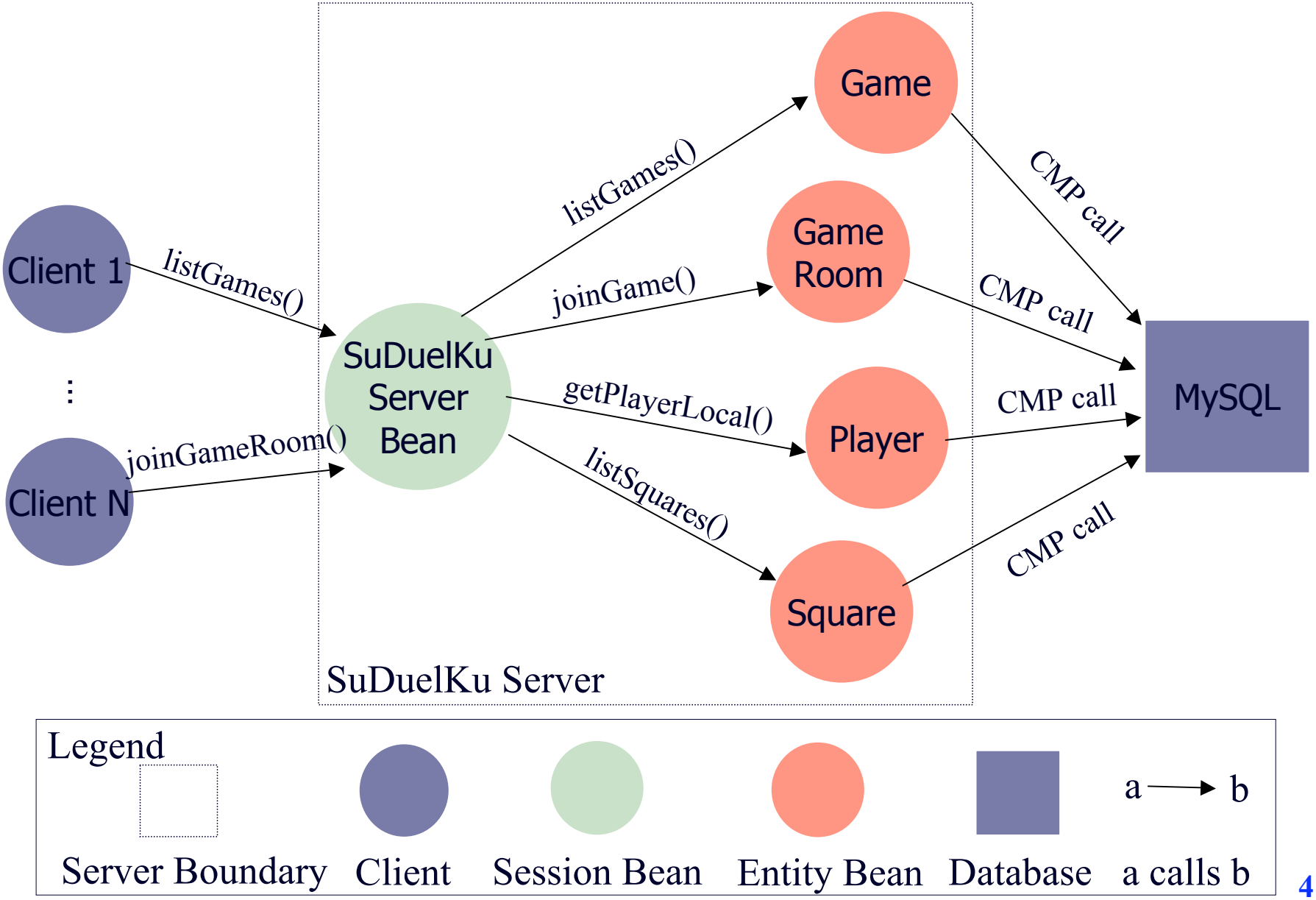
Jriostre@andrew.cmu.edu

http://www.ece.cmu.edu/~ece749/teams-06/team1/html/index.html

Baseline Application

- A real-time, fault-tolerant, high performance game where two or more SuDoKu players can pit their intelligence against each other.
- Su - **Duel** - Ku
- Configuration
  - EJB
  - Linux
  - MySQL
  - Jboss
- Architectural Elements
  - Client(s)
  - Game Server
    - Session Bean
    - Entity Beans
  - Database
- Tools
  - Eclipse
  - LOMBOZ
  - xDoclet
  - Ant

Baseline Architecture



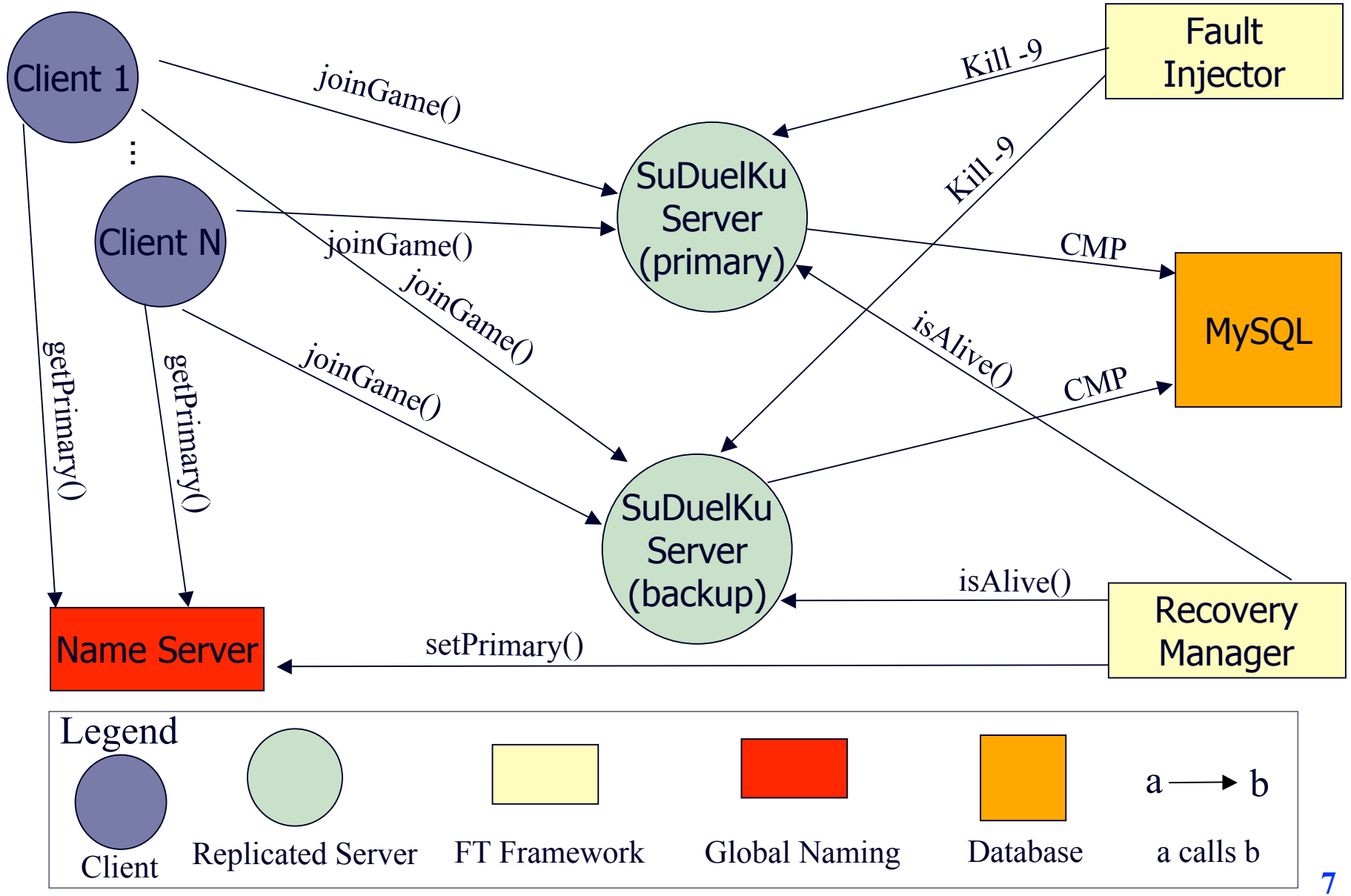
Fault-Tolerance Goals

- Stateless Replicated SuDuelKu Server (each on their own machine)
- Stateless Replicated Global Naming Server (each on their own machine)
- Sacred Machine with
  - Stateful Recovery Manager
  - Stateful Fault-Injector
  - Stateful Database ;-)
  - JMS Server
- Fault-Tolerant Framework
  - Global Recovery Manager
    - Fault Detection
    - Replica Creation
    - Replica Destruction
  - Global Fault Injector

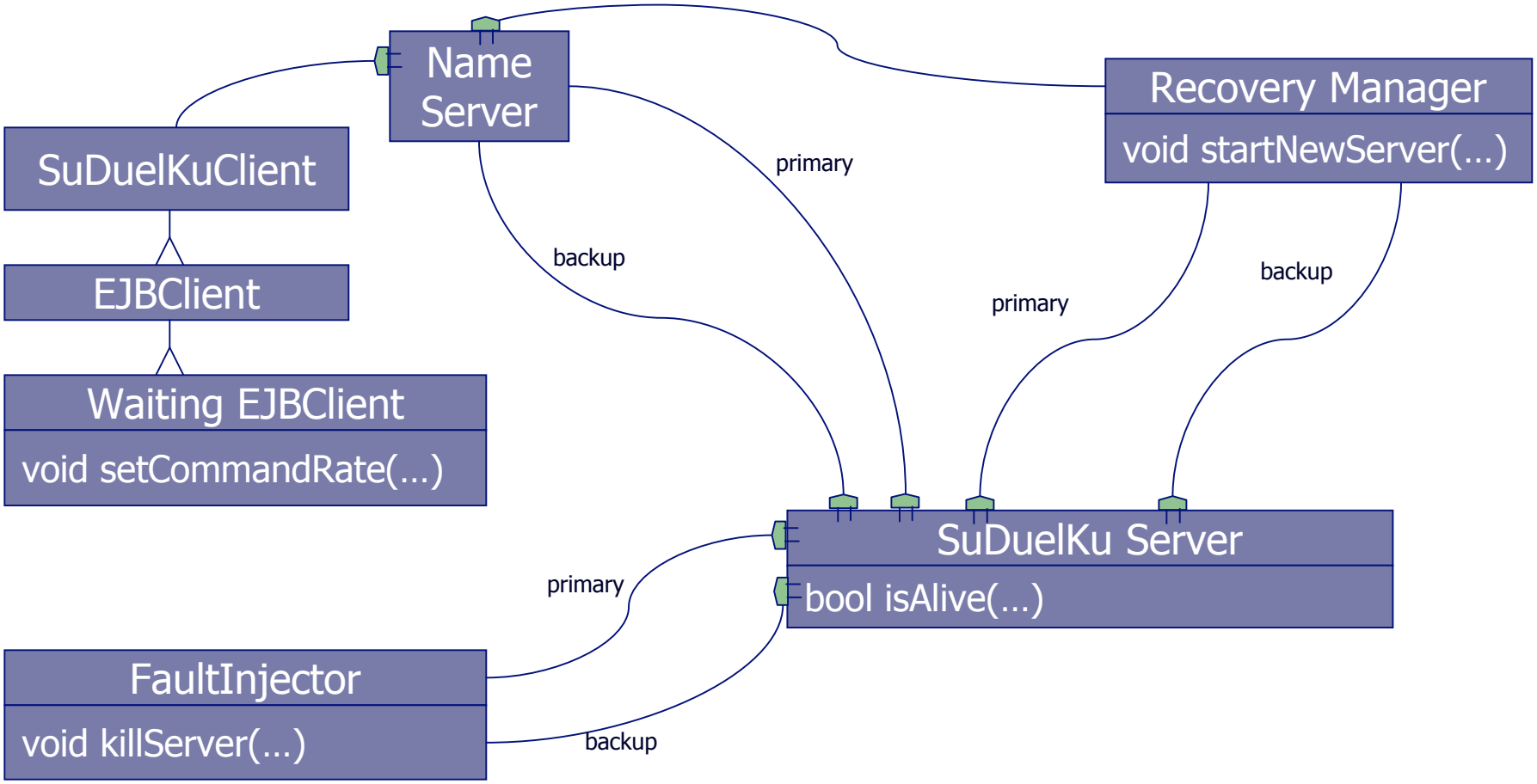
Mechanisms for Fail-Over

- The client asks the global naming server for the primary server
- If the game server goes down, the client gets a *SuduelkuException*.
- After getting an exception, the client waits for a configurable amount of time.
- Meanwhile, the recovery manager tries to execute *isAlive* on the crashed server and gets a *CommunicationException*.
- The recovery manager notifies the naming server that the server is down and sets a new primary server.
- The recovery manager attempts to start the crashed server.
- When the client's waiting time is over, it asks the global naming server for the primary server again.

FT-Baseline Architecture 1

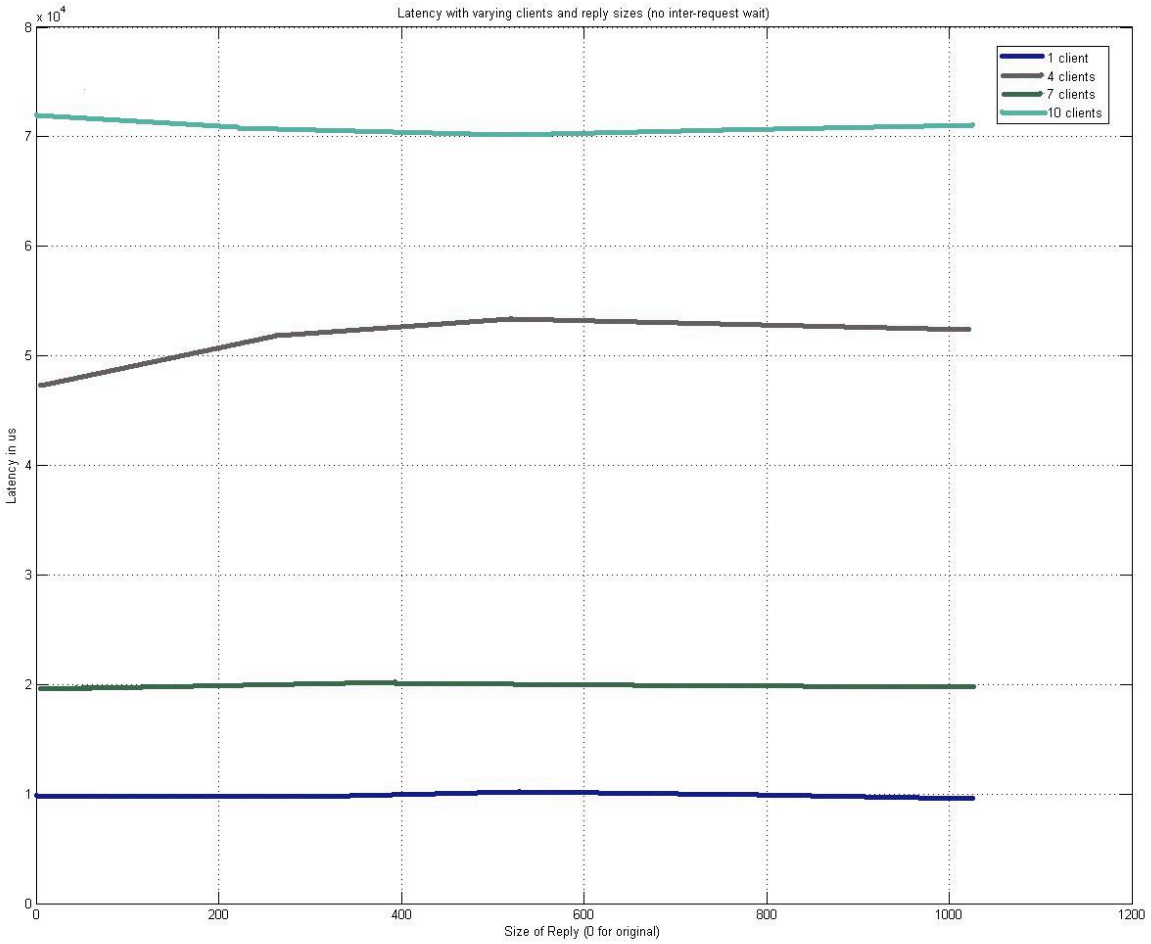


FT-Baseline Architecture 2



Fault-Tolerance Experimentation

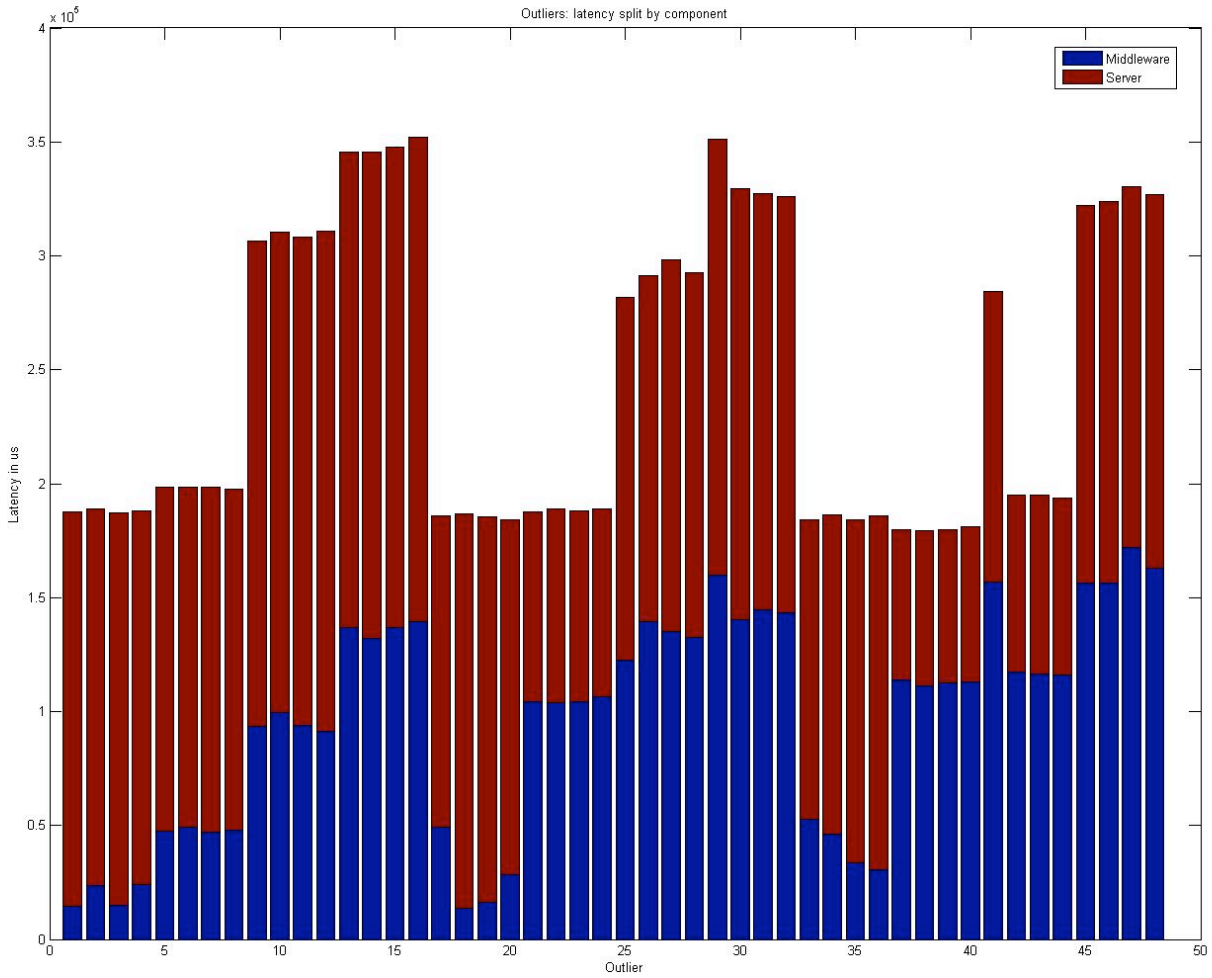
- Latency vs. Size of Reply



- Latency is independent of size of reply – the network isn't the bottleneck

Fault-Tolerance Experimentation

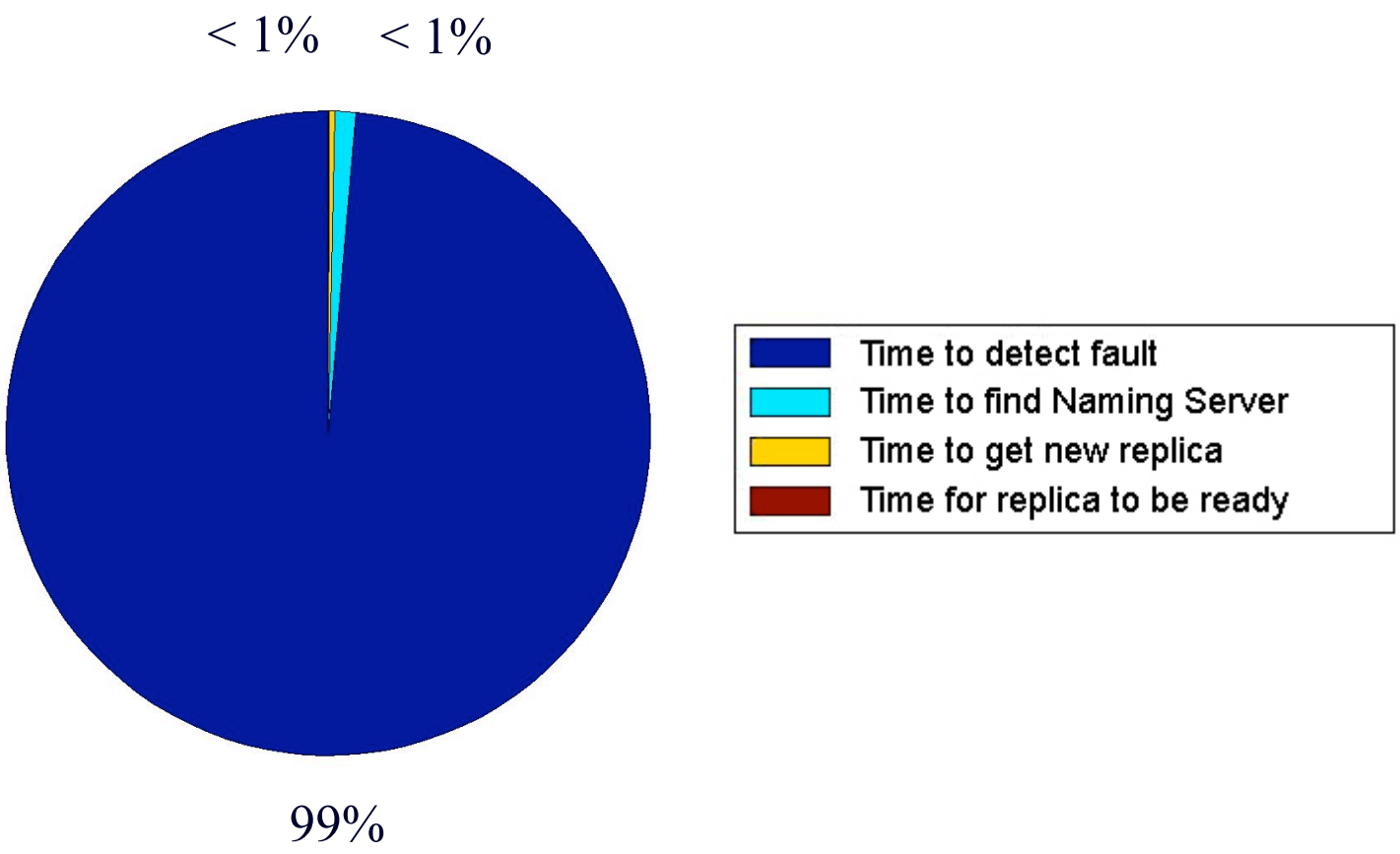
- Latency vs. 3 Sigma Outliers per Experiment



- Number of clients is the major cause of latency

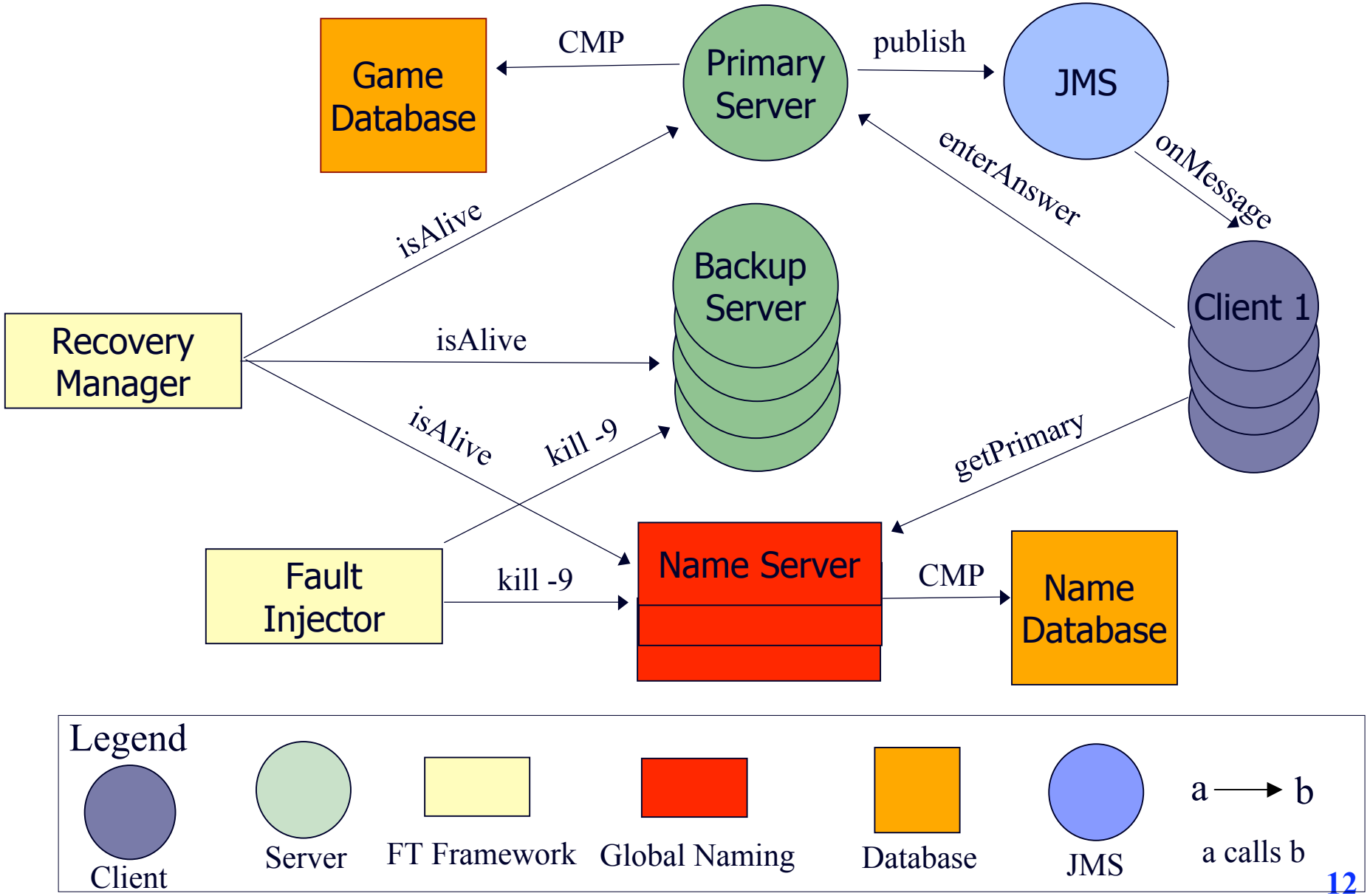
Fail-Over Measurements

- Pie chart break-down of failover time

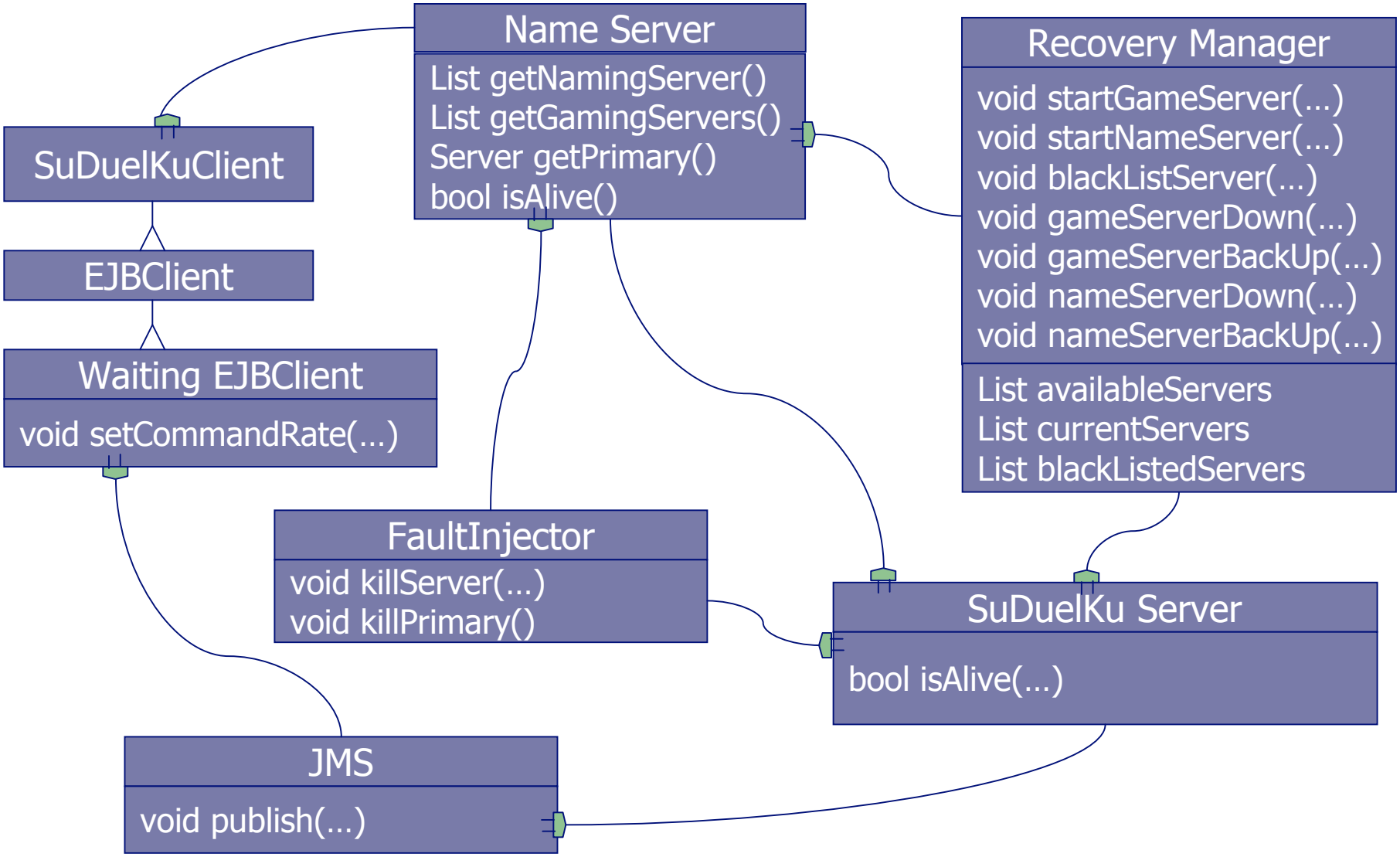


- Fault-Detection is the culprit!

More-FT Architecture 1



More-FT Architecture 2



Other Features

- Fault-Tolerant Naming Server
  - Adds significant complexity to the Recovery Manager state machine
- Callback via JMS
  - JBoss implementation of JMS was good enough
  - Durable subscriptions allowed for clients to be stateless
  - JBoss JMX console helps debugging messaging code
- #1 Recovery Manager - Blacklist problematic servers
  - Good recovery design supports smart recovery tactics
- CLI & GUI !!!
  - Middleware supports clean separation of concerns - Client/Server
  - CLI supports easy scripting of clients
  - GUI supports easy game play!
- Eclipse & Lomboz
  - Tools can have a steep learning curve, but once learned can save time

Insights from Measurements

- Scripting the clients can create unrealistic usage of the system
  - Implementing a usable system with probes will allow for data collection under true system usage
- Stateless servers increase database bottleneck
  - There is a tradeoff between fast data access and failover-time (complexity)
- Reply size did not dramatically impact latency, but request rate did
  - For our application, we may consider sending more data per message to reduce the number of sent messages - thus promoting scalability.
- Running experiments was not as simple as we thought
  - Kerberos authentication expired
  - Ran out of disk space
  - Had trouble keeping multiple clients up
  - Coordination between multiple components: JBoss, database, clients
- Getting representative data is near impossible
  - Expectation management

Open Issues

- Strange behavior after recovering from multiple faults
  - JBoss sporadically reports communication errors when beans are communicating via local interfaces
  - Possibly addressed by using separate JBoss configuration directories - one for our FT Naming Servers and one for our FT Game servers
- Messaging Issues
  - Detect finished games to purge unused topics and subscriptions
  - Identify the exceptions generated by JMS server failure
- Still to come…
  - Fault-Tolerant callbacks
  - Improved server startup time to increase tolerable failure rate
  - Improve scalability to support expected growth in the user community ;-)
  - Calculating player scores
  - Purging complete games from the database
  - Algorithm for game generation

Conclusions (1)

- Enlightenment
  - Middleware does not remove all complexity, it just moves the complexity into a new layer
  - Middleware decisions place constraints on software architecture that must be addressed during design
  - Fault-Tolerance, Real-Time, and High-Performance all involve tradeoffs that must be understood and be made explicitly
- Why we are Team #1 (read #1 Team)
  - Dueling SuDoKu game
  - Fault-Tolerant game server
  - Fault-Tolerant global naming server
  - Callbacks (soon to be FT) within a Fault-Tolerant application
  - An awesome GUI

Conclusions (2)

- New approaches for next time
  - Naming conventions
  - Coding conventions
  - Start implementing probes early
  - Start gathering data early
  - Design for probes
  - Design for a scriptable client
  - Start thinking about transactions early
  - Be aware of concurrency issues

Demo

Questions

Additional thoughts

- How would active replication help?
- What would be the impact of the active replication? Can we keep state in the database?
- How complex would it be to keep server with state? Could there be a mixed style?

- Some more lessons
  - CMP is handy but very complex
  - CMP has limitations with database design (no referential integrity allowed)
  - Experimentation is a laborious process
  - Debugging the server is not as easy as the client
  - JNI works differently on solaris than linux
  - Output to indicate progress of experiments effects the results, but without it, debugging experiments is non-trivial
  - Use a mechanism for disabling debug/probe output