Distributed Computing, ACID and the Google Web Toolkit

Lecture 26 – CS 2110 – Fall 2010

Johnathon Schultz

But First…
Life after CS 2110

{ Or, my favorite class is over, what should I do now?

Take More CS Courses

- CS 3310 – Functional Programming
  - My greatest regret from my time at Cornell is not taking this course
  - You will become awesome at programming
- INFO 3300 – Data Driven Web Applications
  - Learn what I'm about to talk about
  - Learn how to learn new programming environments
- CS 3810 – Theory of Computing
  - Prove that something cannot be parsed with Regular Expressions
- CS 2800 – Discrete Structures
  - The math behind CS

- CS 4410 – Operating Systems
  - Concurrency, Scheduling, Filesystems
- CS 4700 – Foundations of A.I.
  - Alpha-beta pruning, A*, Heuristics, ML
- CS 4120 – Compilers
  - Compile a language to JVM bytecode

Take High-Level Courses that you are Interested in

Learn New Languages

- C#
  - Like Java
  - "Microsoft took everything they ever heard of in a programming language and bolted it onto C#"
- C/C++
  - Blazingly fast
  - You will understand how everything works… when you throw pointers into the operating system and segfault

Learn New Languages

- Python
  - Ideal for quick scripts
  - Being able to understand something months after you wrote it without comments
- Perl
  - Even better for quick scripts
  - Will make you awesome at Regex
  - Masochism
  - Not being able to understand something 5 minutes after you wrote it
- PHP
  - Don't

Distributed Computing

{ Or, How I learned to stop worrying and love the bomb

Distributed Computing

- Up until now we've talked about Java on single machine
  - Perhaps with threads to exploit multi-core parallelism
- But suppose that objects could "live" on other machines
  - Then if we could invoke methods on them we could create a distributed program

```
package server;
import javax.jws.WebService;

@WebService
public class HelloImpl {

	/**
	* @param name
	* @return Say hello to the person.
	*/
	public String sayHello(String name){
		return "Hello, " + name + "!";
	}
}
```

Java supports this model, it's called a "Web Services" architecture.

{ Your programs designates certain interfaces and makes them available on the web using Annotations (e.g. @WebService)

Distributed Computing

Talking to the Service

- Before you can write the client you need to run a program called APT
- APT creates:
  - A so-called "WSDL" file that looks like a web page and describes the new service
  - A "schema" for the messages used to talk to the service
  - Java classes to receive requests and "unpack" them, and to send the response back (which "repacks" them)
    - The terminology for this is "Marshalling" and "Unmarshalling"
  - The client "stub" file

Then…

- You start your program on the machine that will be the server
- You also need to wave a magic want to “register” the service with the “Internet Information Service”
  - Or edit the bowels of your Apache configuration files
  - Or setup Tomcat
  - (Really, it’s a choose your own poison situation)
- Then on the client machine you import the service and can then write code to talk to it

Talking to the Web Service

- Done using a client web-service proxy
- When executed, prints:

  Hello Service returned: <Hello My master!>

```
static void Main(string[] args)
{
    HelloServiceClient proxy = new HelloServiceClient();
    String result = proxy.SayHello(“My master”);
    Console.WriteLine(“Hello Service returned: <" + result + “>”);
}
```

Web Browser???

- In fact these solutions literally make your client program behave just like a web browser
  - You can even USE a web browser as a client!
- And they make the server program look like a web site, complete with a URL of its own!
  - And you can point a web browser at that site
- Web services use special HTML (more generally, XML) to send requests and create replies

Java Serialization

- One way to send and receive Java objects is through a process called serialization
- This is a way of writing down an object in text format
- The idea is we can serialize an object, put it into a message to a web service, and receive a serialized object as the result

The Magic of Distributed Computing

- You can write an object oriented application now but instead of all the objects being on one machine
  - Put them any place you like!
- An object becomes a bit like a web page
- If you know how to find it, you can ask it to do stuff!
  - But must pass arguments by “value”, not “reference”

Networking vs. D.C.

- A "networked" application is one that talks to some resources on some other machine
  - Like a file or a web page
  - Network applications *make no promises.*
- We're used to this "model" and know about its quirks
  - You often get timeouts
  - Sometimes your order is dropped, or goes in twice

Distributed Computing

- Some applications (like medical ones) need stronger guarantees:
  - Need to know who the client is
  - And need to "trust" the service
  - May need to protect data against intruders
  - Might want to ensure that the service will be operational even if a crash occurs
- These turn the problem into "distributed computing"

Distributed Promises

- A distributed system makes promises!
  - …. I promise to behave like a non-distributed service that never fails
  - …. I promise you'll never notice effects of concurrency
  - …. I won't reveal data to the wrong people. Really!
  - …. Even evil-doers won't stop me from doing the right thing, all the time

ACID

{ Or, making sure your data doesn't get corroded

Example Problem

- A hospital has five servers
  - They hold medical record "objects"
  - And we want fault-tolerance
- You write an application to let a doctor enter a new medication order
  - "Put this patient on 2 units of Morphine per hour"
  - Need to update the servers
- What if something crashes?

Leads to the idea of a "transaction"

- Idea dates to early work on databases
  - Key concept is that either the operation is done to completion, or it fails and does nothing at all
  - A transaction, by definition, must be
    - **atomic**,
    - **consistent**,
    - **isolated**, and
    - **durable**

- How can a client perform an ACID update?

Atomicity

- Atomicity requires that database modifications must follow an "all or nothing" rule
  - If one part of the transaction fails, the entire transaction fails and the database state is left unchanged
- Transactions can fail for several kinds of reasons:
  1. Hardware failure: A disk drive fails, preventing some of the transaction's database changes from taking effect.
  2. System failure: The user loses their connection to the application before providing all necessary information.
  3. Database failure: E.g., the database runs out of room to hold additional data.
  4. Application failure: The application attempts to post data that violates a rule that the database itself enforces, such as attempting to insert a duplicate value in a column.

Consistency

- The consistency property ensures that any transaction the database performs will take it from one consistent state to another
  - A particular field is for holding integer numbers
  - Two options to maintain consistency when presented with a double value
    - reject attempts to put a double there
    - round the supplied values to the nearest whole number

Isolation

- Isolation refers to the requirement that other operations cannot access data that has been modified during a transaction that has not yet completed
- Think of the threading question from Prelim 2

Durability

- Durability is the ability of the DBMS to recover the committed transaction updates against any kind of system failure (hardware or software).

Failure Examples

Atomicity Failure

- The transaction subtracts 10 from A and adds 10 to B.
- If it succeeds, it would be valid, because the data continues to satisfy the constraint.
- However, assume that after removing 10 from A, the transaction is unable to modify B.
- If the database retains A's new value, atomicity would be violated.
- Atomicity requires that both parts of this transaction complete or neither.

Isolation Failure

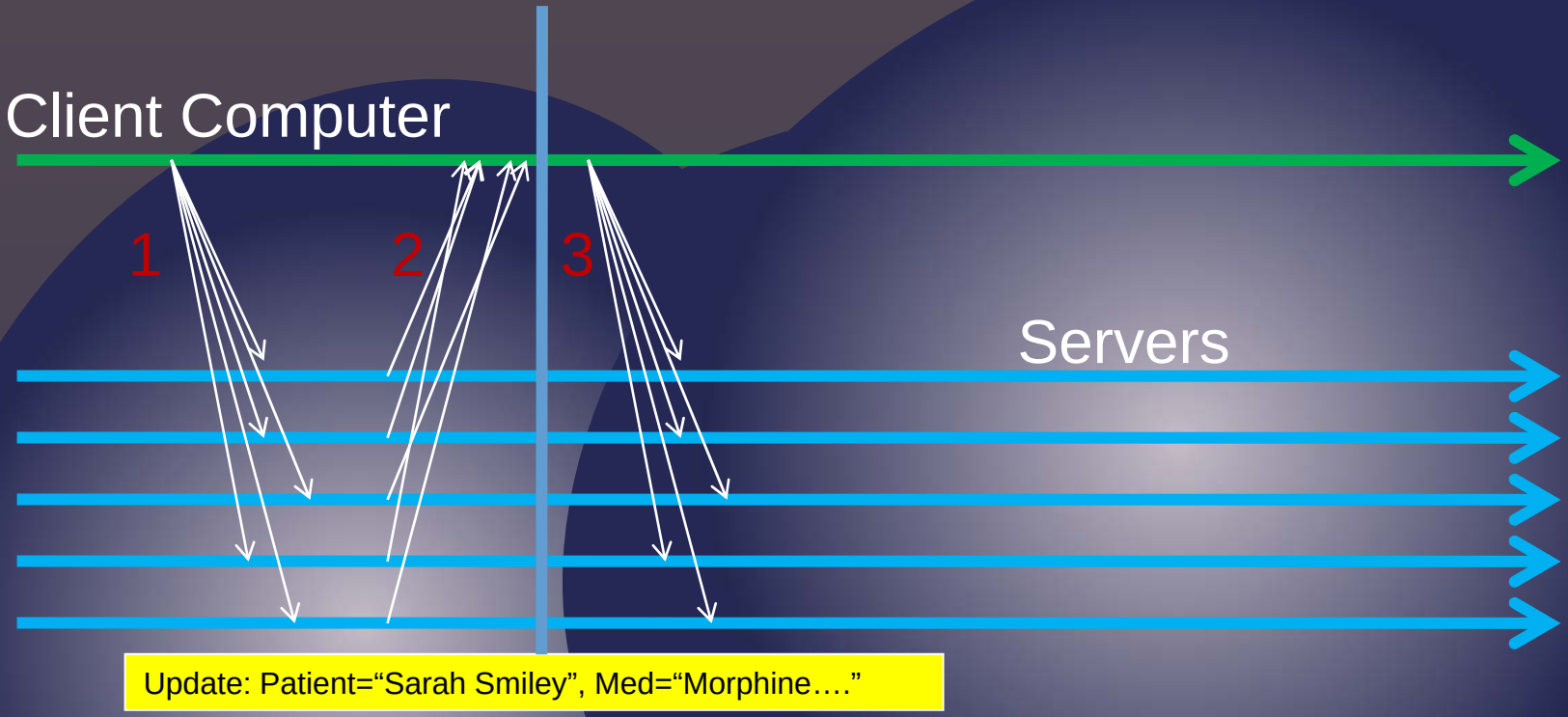
- Consider two transactions. T_1 transfers 10 from A to B. T_2 transfers 10 from B to A. Combined, there are four actions:
  - subtract 10 from A
  - add 10 to B.
  - subtract 10 from B
  - add 10 to A
- If these operations are performed in order, isolation is maintained, although T_2 must wait.
  - Consider what happens, if T_1 fails half-way through. The database eliminates T_1's effects, and T_2 sees only valid data.
- By interleaving the transactions, the actual order of actions might be: $A - 10, B - 10, B + 10, A + 10.$
  - Again consider what happens, if T_1 fails.
  - T_1 still subtracts 10 from A. Now, T_2 adds 10 to A restoring it to its initial value. Now T_1 fails. T_2 subtracts 10 from it. If T_2 is allowed to complete, B's value will be 10 too low, and A's value will be unchanged, leaving an invalid database.
  - This is known as a write-write failure, because two transactions attempted to write to the same data field.

Durability Failure

- Assume that a transaction transfers 10 from A to B.
- It removes 10 from A. It then adds 10 to B.
- At this point, a "success" message is sent to the user.
- However, the changes are still queued in the disk buffer waiting to be committed to the disk.
- Power fails and the changes are lost. The user assumes that the changes have been made, but they are lost.

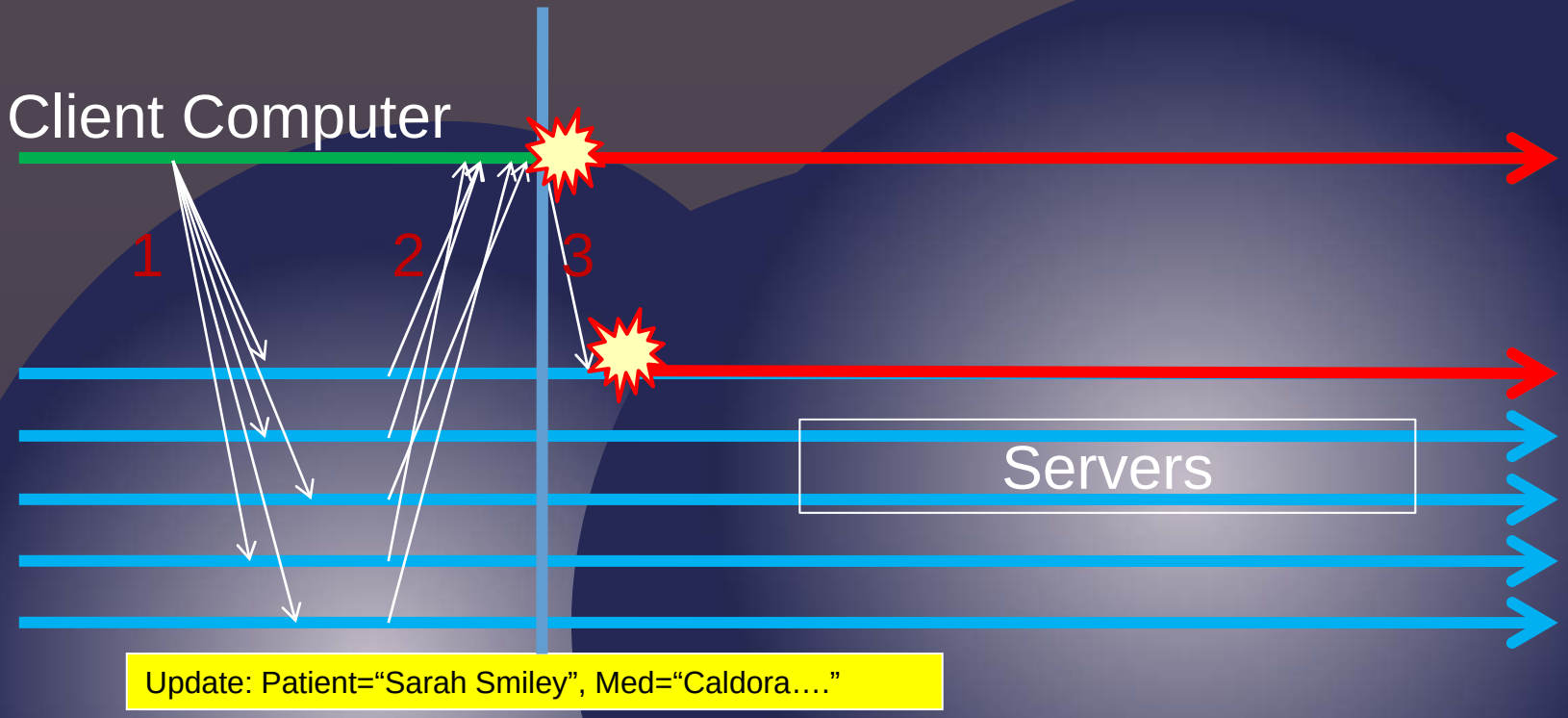
Transactions

{ Or, let's pour ACID all over it



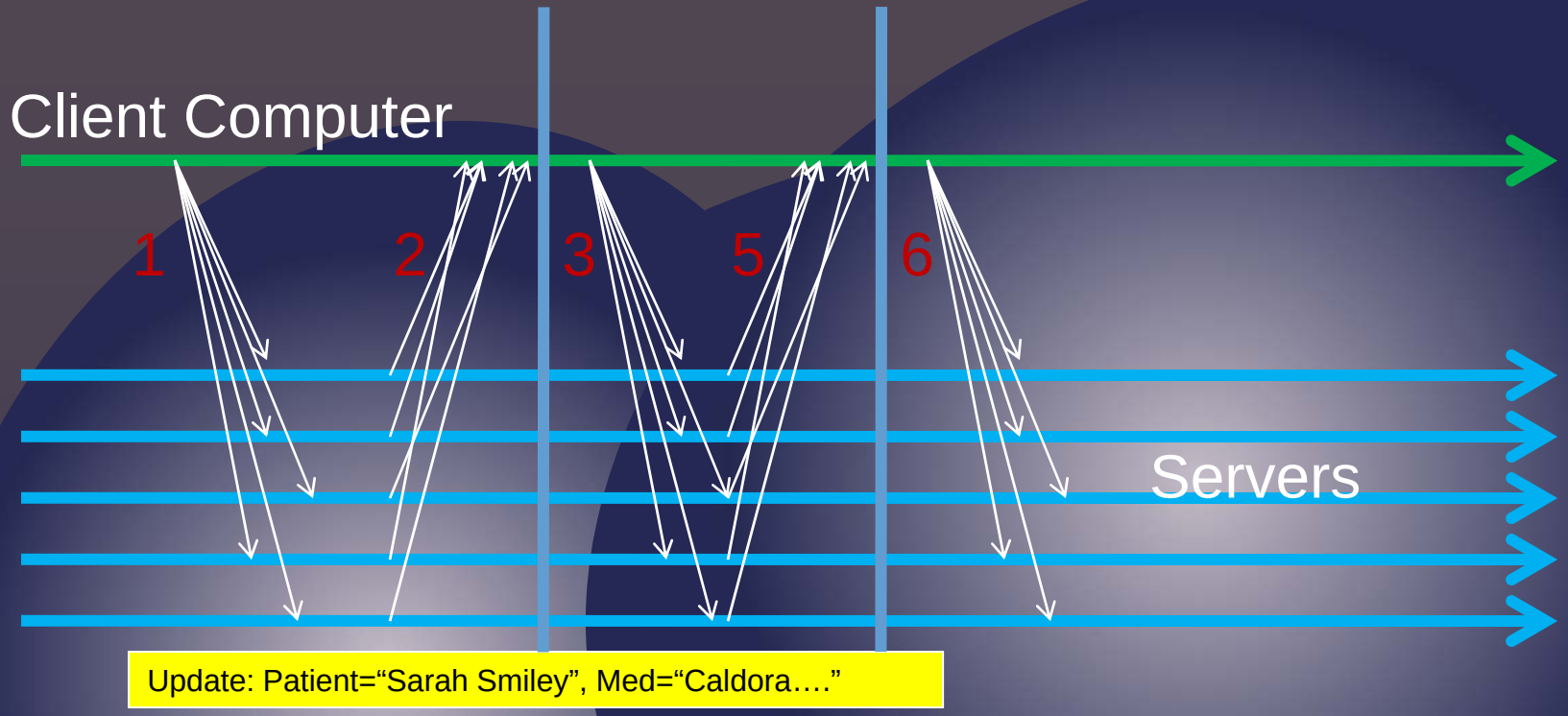
- Idea is to have a "prepare" phase (1, 2) and then a "commit or abort" phase (3)

Two-phase Commit

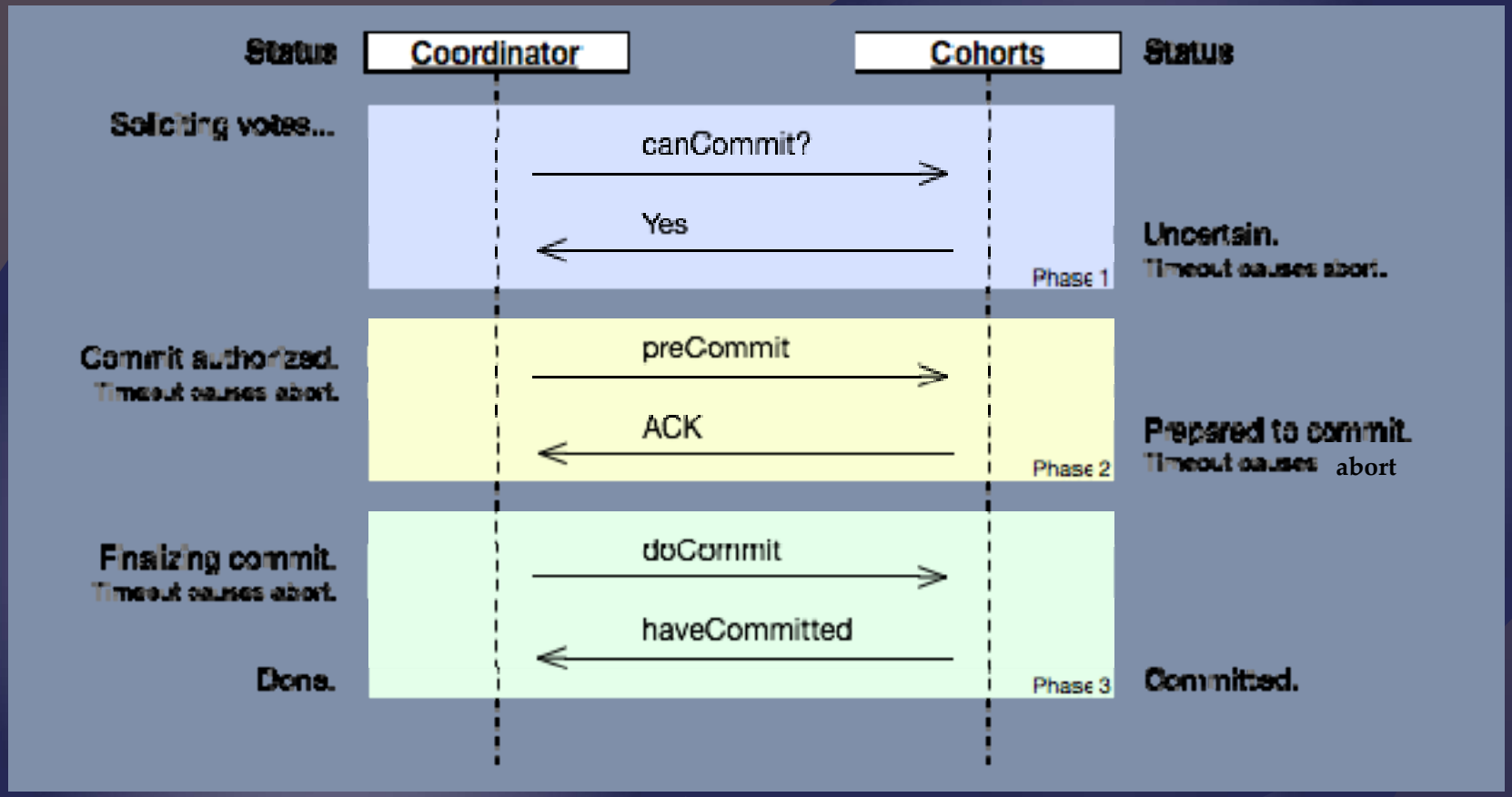


- Suppose the client and one machine crash
  - But client had just enough time to send *one* stage-3 msg
- The remainder of the servers might be in an inconsistent state!

Two-phase Commit Problem



Three-phase Commit



Three-phase Commit

Google Web Toolkit

{ Or, making web programming easier than PHP

What is it?

- Write an AJAX front-end to a Java server in Java
- The GWT cross-compiles your Java to Javascript to run in a browser
- During development, all Java. Use the debugger to find errors
- During release, Javascript runs the same

- Google Wave
- Google Moderator
- Go Grid
- Wirled

Examples

Features

- Build a HTML page like a Java GUI
  - Tools for building the GUI with a GUI
- RPC calls built in
  - Support for asynchronous calls
- Works on top of the Google App Engine
  - Store data in the datastore
  - Store Java objects in a database and run queries on them

Example Time

References

- http://www.artima.com/lejava/articles/threeminutes.html
- http://en.wikipedia.org/wiki/Two-phase_commit_protocol
- http://en.wikipedia.org/wiki/Three-phase_commit_protocol