# Principles of Software Construction: Objects, Design and Concurrency

# Java I/O and an Introduction to Distributed Systems

***15-214***

***toad***

Spring 2013


Christian Kästner **Charlie Garrod**

School of Computer Science

institute for SOFTWARE RESEARCH

# Administrivia

- SVN
  - Commit early, commit often
- Do you want to be a software engineer?

# The foundations of the Software Engineering minor

- Core computer science fundamentals
- Building good software
- Organizing a software project
  - Development teams, customers, and users
  - Process, requirements, estimation, management, and methods
- The larger context of software
  - Business, society, policy
- Engineering experience
- Communication skills
  - Written and oral

# SE minor requirements

- Prerequisite: 15-214
- Two core courses
  - 15-313
  - 15-413
- Three electives
  - Technical
  - Engineering
  - Business or policy
- Software engineering internship + reflection
  - 8+ weeks in an industrial setting, then
  - 17-413

# To apply to be a Software Engineering minor

- Email jonathan.aldrich@cs.cmu.edu and poprocky@cs.cmu.edu
  - Your name, Andrew ID, class year, QPA, and minor/majors
  - Why you want to be a software engineer
  - Proposed schedule of coursework
- Spring applications due this Friday, 12 April 2013
  - Only 15 SE minors accepted per graduating class
- More information at:
  - http://isri.cmu.edu/education/undergrad/

# Administrivia

- SVN
  - Commit early, commit often
- Do you want to be a software engineer?

# Key topics from last Thursday

# Today

- Java I/O fundamentals, continued
  - Basic networking
- Introduction to distributed systems
  - Motivation: reliability and scalability
  - Failure models
  - Techniques for:
    - Reliability (availability)
    - Scalability
    - Consistency

# The fundamental I/O abstraction: a stream of data

- `java.io.InputStream`

```
void          close();
abstract int  read();
int           read(byte[] b);
```

- `java.io.OutputStream`

```
void          close();
void          flush();
abstract void write(int b);
void          write(byte[] b);
```

- Aside: If you have an `OutputStream` you can construct a `PrintStream`:

```
PrintStream(OutputStream out);
PrintStream(File file);
PrintStream(String filename);
…
```

# To read and write arbitrary objects

- Your object must implement the `java.io.Serializable` interface
  - Methods: none!
  - If all of your data fields are themselves `Serializable`, Java can automatically serialize your class
    - If not, will get runtime `NotSerializableException`
- See QABean.java and FileObjectExample.java

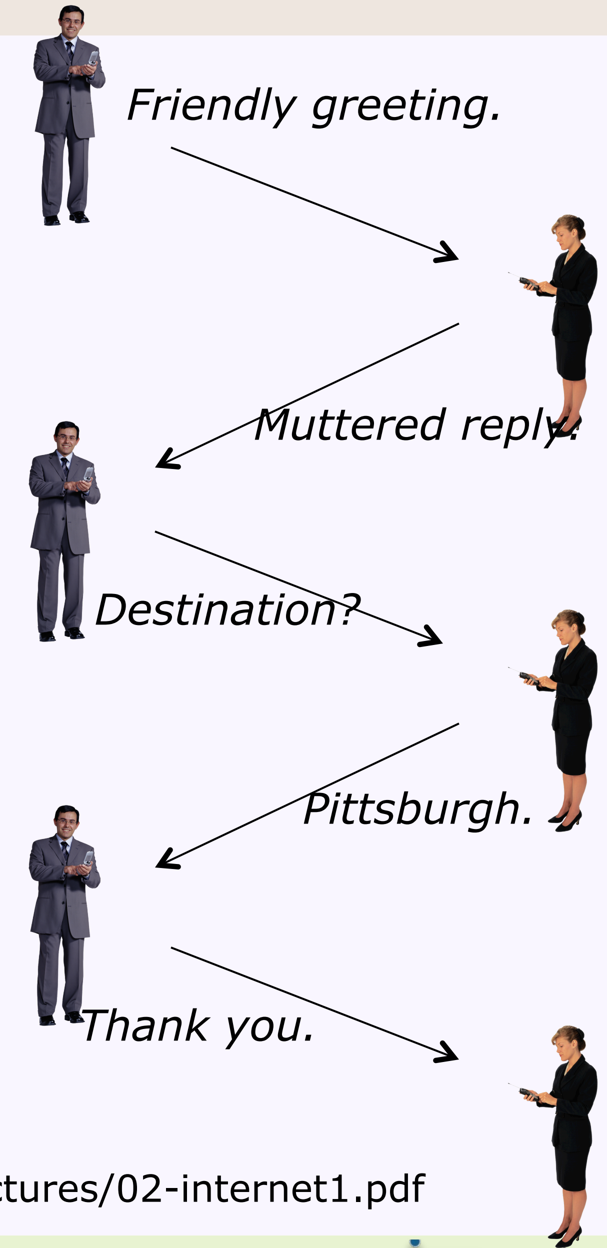
# Our destination: Distributed systems

- Multiple system components (computers) communicating via some medium (the network)
- Challenges:
  - Heterogeneity
  - Scale
  - Geography
  - Security
  - Concurrency
  - Failures

(courtesy of http://www.cs.cmu.edu/~dga/15-440/F12/lectures/02-internet1.pdf

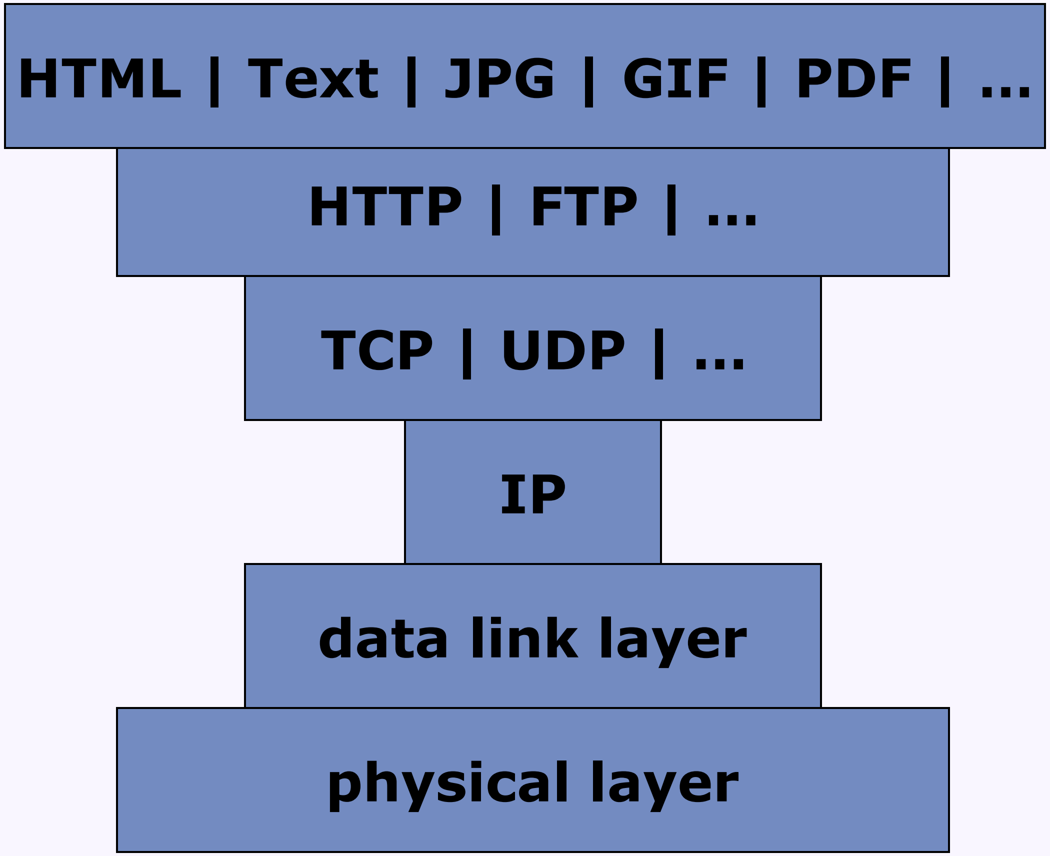
# Communication protocols

- Agreement between parties for how communication should take place
  - e.g., buying an airline ticket through a travel agent




(courtesy of http://www.cs.cmu.edu/~dga/15-440/F12/lectures/02-internet1.pdf

# Abstractions of a network connection



# Packet-oriented and stream-oriented connections

- UDP: User Datagram Protocol
  - Unreliable, discrete packets of data
- TCP: Transmission Control Protocol
  - Reliable data stream

# Internet addresses and sockets

- For IP version 4 (IPv4) host address is a 4-byte number
  - e.g. 127.0.0.1
  - Hostnames mapped to host IP addresses via DNS
  - ~4 billion distinct addresses
- Port is a 16-bit number (0-65535)
  - Assigned conventionally
    - e.g., port 80 is the standard port for web servers
- In Java:
  - `java.net.InetAddress`
  - `java.net.Inet4Address`
  - `java.net.Inet6Address`
  - `java.net.Socket`
  - `java.net.InetSocket`

# Networking in Java

- The java.net.InetAddress:

```
static InetAddress getByName(String host);
static InetAddress getByAddress(byte[] b);
static InetAddress getLocalHost();
```

- The java.net.Socket:

```
Socket(InetAddress addr, int port);
boolean      isConnected();
boolean      isClosed();
void         close();
InputStream  getInputStream();
OutputStream getOutputStream();
```

- The java.net.ServerSocket:

```
ServerSocket(int port);
Socket       accept();
void         close();
…
```

# A simple Sockets demo

- TextSocketClient.java
- TextSocketServer.java
- TransferThread.java

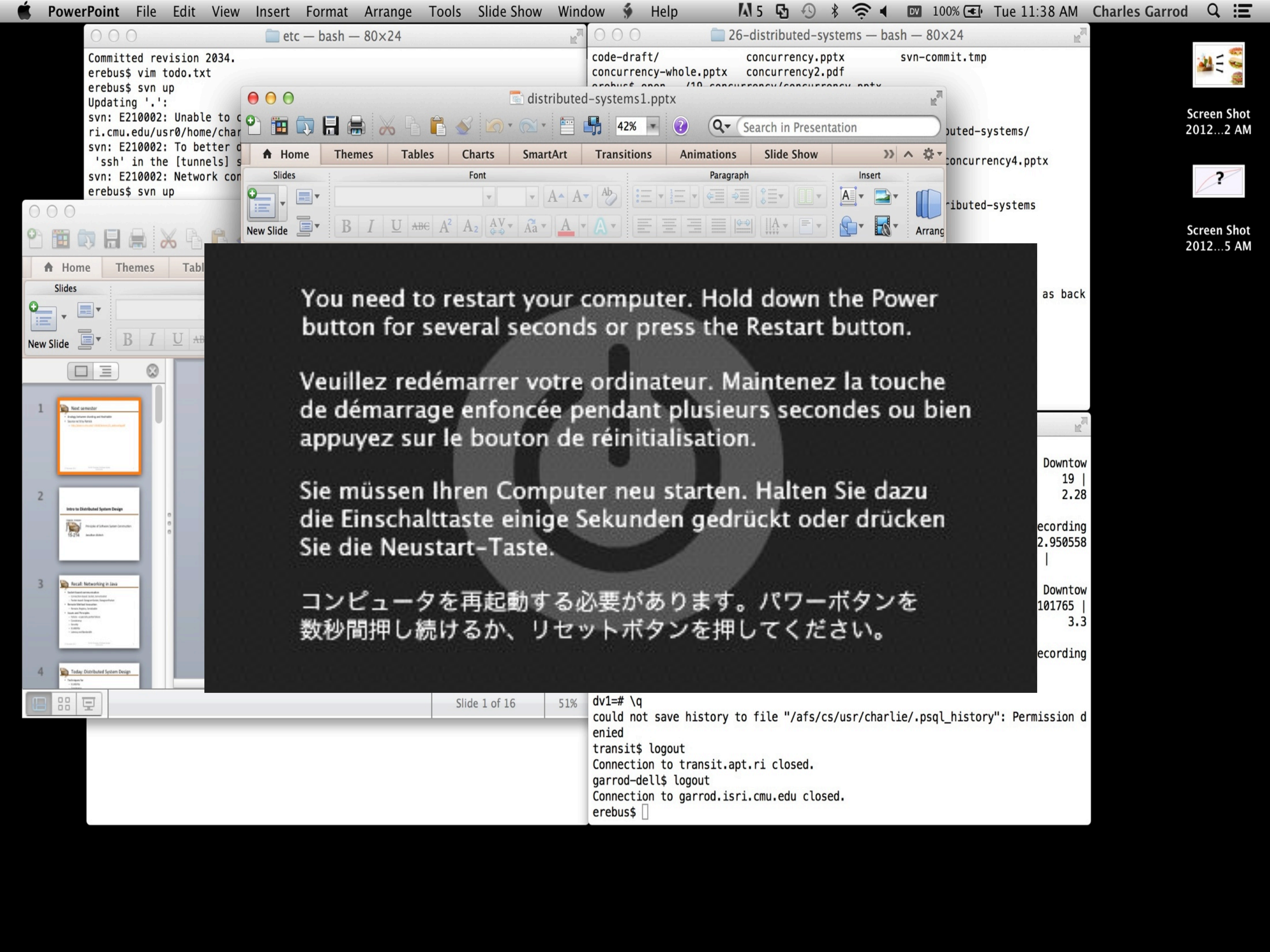
# What do you want to do with your distributed system today?

# Higher levels of abstraction

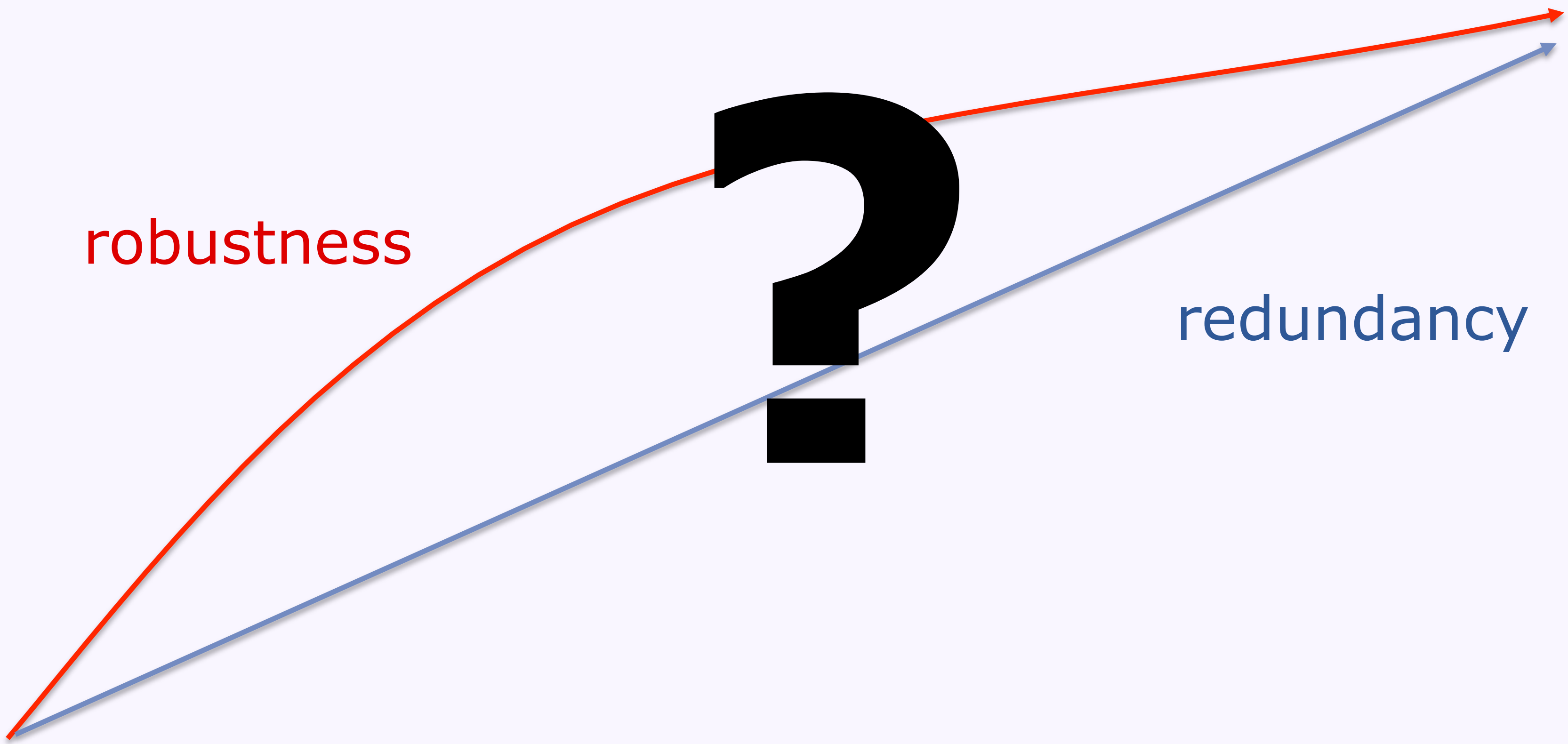
- Application-level communication protocols
- Frameworks for simple distributed computation
  - Remote Procedure Call (RPC)
  - Java Remote Method Invocation (RMI)
- Common patterns of distributed system design
- Complex computational frameworks
  - e.g., distributed map-reduce

# Today

- Java I/O fundamentals, continued
  - Basic networking
- Introduction to distributed systems
  - Motivation: reliability and scalability
  - Failure models
  - Techniques for:
    - Reliability (availability)
    - Scalability
    - Consistency

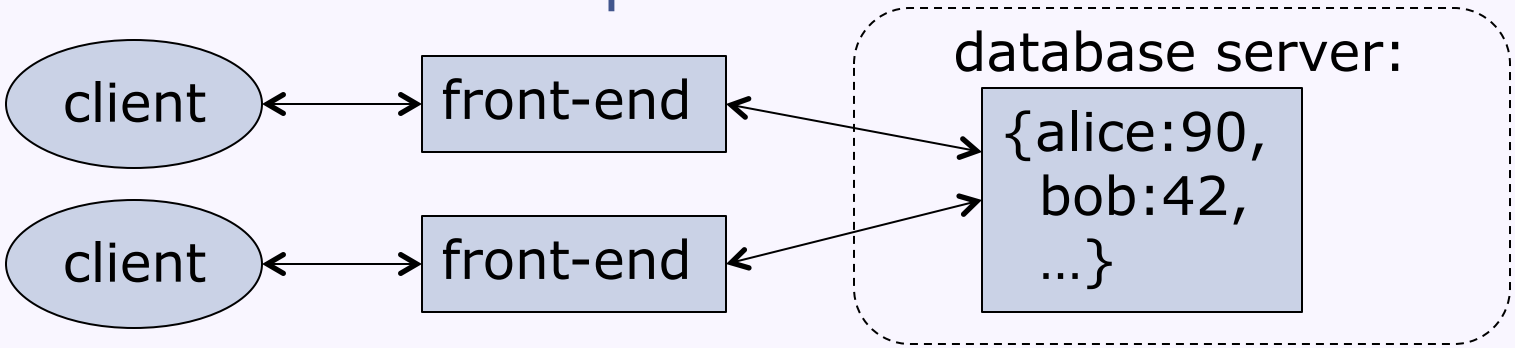

PowerPoint File Edit View Insert Format Arrange Tools Slide Show Window Help
Tue 11:38 AM Charles Garrod
etc — bash — 80×24
Committed revision 2034.
erebus$ vim todo.txt
erebus$ svn up
Updating '.':
26-distributed-systems — bash — 80×24
code-draft/ concurrency.pptx svn-commit.tmp
concurrency-whole.pptx concurrency2.pdf
distributed-systems1.pptx
Search in Presentation
Home Themes Tables Charts SmartArt Transitions Animations Slide Show
Slides Font Paragraph Insert
New Slide
Screen Shot 2012...2 AM
Screen Shot 2012...5 AM
You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.
Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.
Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.
コンピュータを再起動する必要があります。パワーボタンを数秒間押し続けるか、リセットボタンを押してください。
Slide 1 of 16
51%
dv1=# \q
could not save history to file "/afs/cs/usr/charlie/.psql_history": Permission denied
transit$ logout
Connection to transit.apt.ri closed.
garrod-dell$ logout
Connection to garrod.isri.cmu.edu closed.
erebus$

# Aside: The robustness vs. redundancy curve



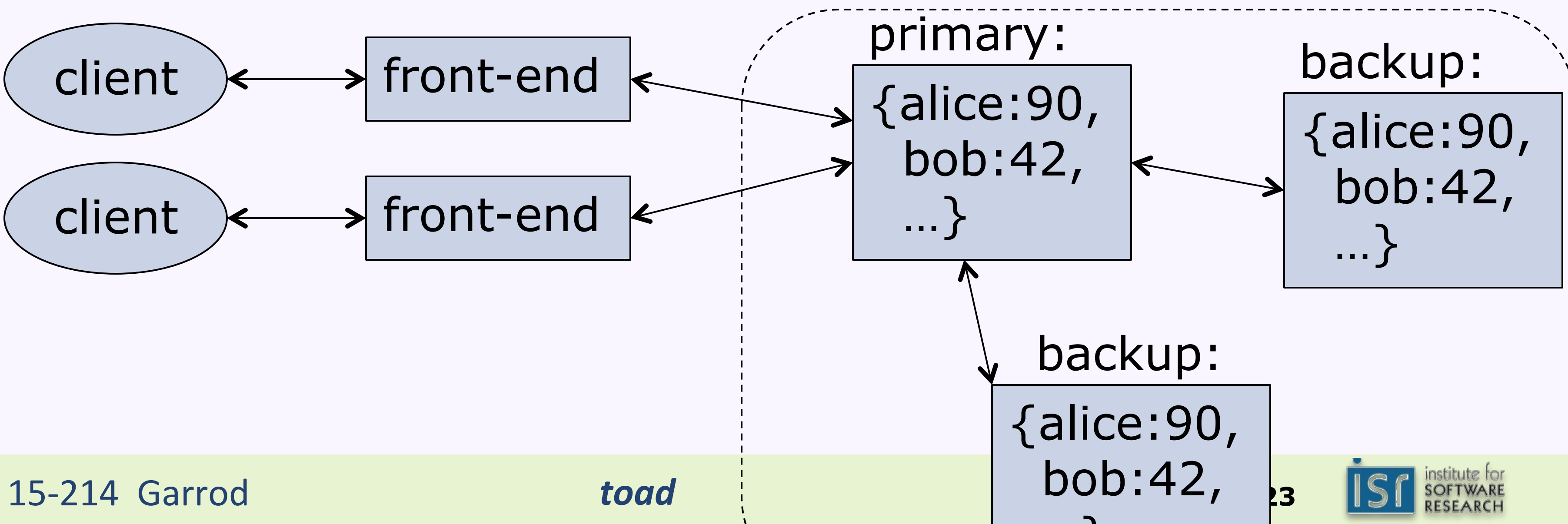
# A case study: Passive primary-backup replication

- Architecture before replication:




  - Problem: Database server might fail

- Solution: Replicate data onto multiple servers



# Passive primary-backup replication protocol

1. Front-end issues request with unique ID to primary DB
2. Primary checks request ID
   - If already executed request, re-send response and exit protocol
3. Primary executes request and stores response
4. If request is an update, primary DB sends updated state, ID, and response to all backups
   - Each backup sends an acknowledgement
5. After receiving all acknowledgements, primary DB sends response to front-end

# Issues with passive primary-backup replication

# Issues with passive primary-backup replication

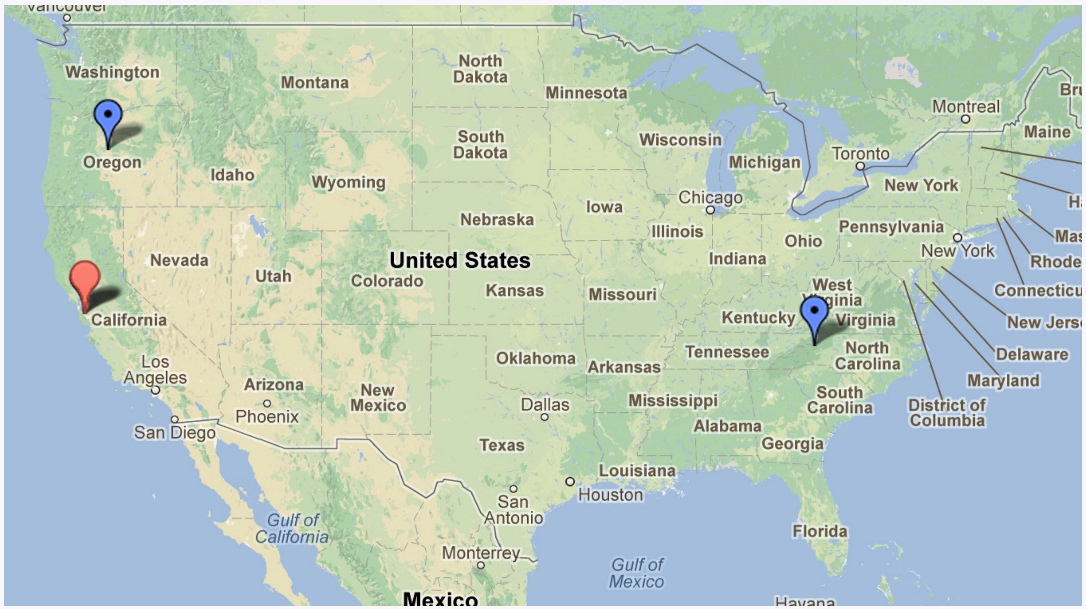
- Many subtle issues with partial failures
- If primary DB crashes, front-ends need to agree upon which unique backup is new primary DB
  - Primary failure vs. network failure?
- If backup DB becomes new primary, surviving replicas must agree on current DB state
- If backup DB crashes, primary must detect failure to remove the backup from the cluster
  - Backup failure vs. network failure?
- If replica fails* and recovers, it must detect that it previously failed
- …

# More issues…

- Concurrency problems?
  - Out of order message delivery?
    - Time…
- Performance problems?
  - 2n messages for n replicas
  - Failure of any replica can delay response
  - Routine network problems can delay response
- Throughput problems?
  - All replicas are written for each update, but primary DB responds to every request
  - Does not address the scalability challenge

# Aside: Facebook and primary-backup replication

- Variant for scalability only:
  - Read-any, write-all
  - Palo Alto, CA is primary replica




  - A 2010 conversation:

    Academic researcher: What would happen if X occurred?

    Facebook engineer: We don't know. X hasn't happened yet…but it would be bad.

# Types of failure behaviors

- Fail-stop
- Other halting failures
- Communication failures
  - Send/receive omissions
  - Network partitions
  - Message corruption
- Performance failures
  - High packet loss rate
  - Low throughput
  - High latency
- Data corruption
- Byzantine failures

# Common assumptions about failures

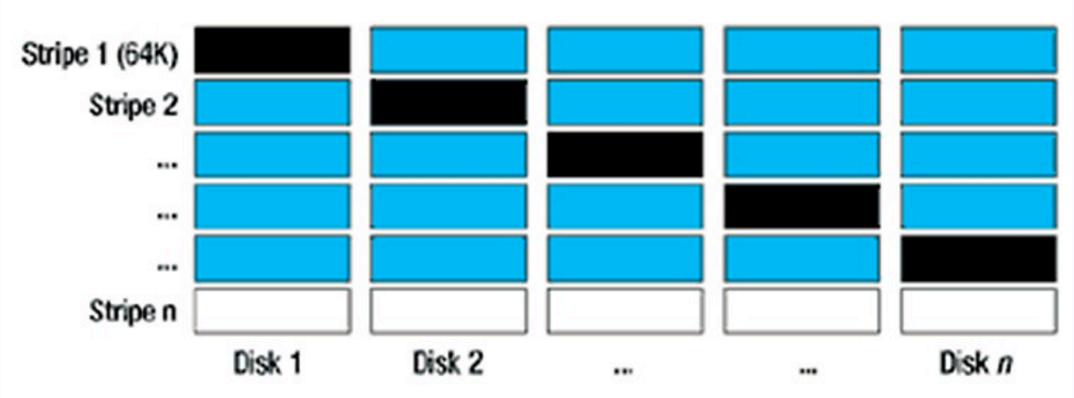
- Behavior of others is fail-stop (ugh)
- Network is reliable (ugh)
- Network is semi-reliable but asynchronous
- Network is lossy but messages are not corrupt
- Network failures are transitive
- Failures are independent
- Local data is not corrupt
- Failures are reliably detectable
- Failures are unreliably detectable

# Some distributed system design goals

- The end-to-end principle
  - When possible, implement functionality at the end nodes (rather than the middle nodes) of a distributed system
- The robustness principle
  - Be strict in what you send, but be liberal in what you accept from others
    - Protocols
    - Failure behaviors
- Benefit from incremental changes
- Be redundant
  - Data replication
  - Checks for correctness

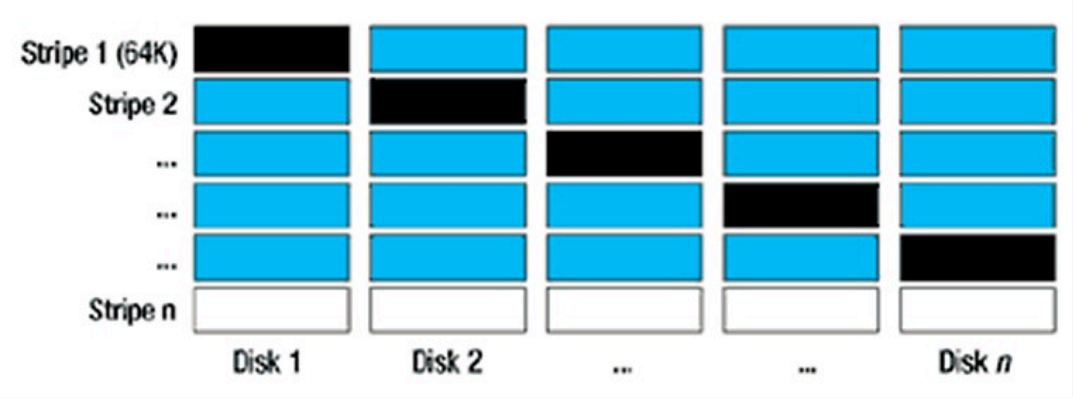
# A case of contradictions:  RAID

- RAID:  Redundant Array of Inexpensive Disks
  - Within a single computer, replicate data onto multiple disks
  - e.g., with 5 1TB disks can get 4TB of useful storage and recover from any single disk failure



# A case of contradictions: RAID

- RAID: Redundant Array of Inexpensive Disks
  - Within a single computer, replicate data onto multiple disks
  - e.g., with 5 1TB disks can get 4TB of useful storage and recover from any single disk failure




- Aside: Does Google use RAID?