M 120: DISTRIBUTED SYSTEMS

Course Overview

*Slides include material provided by Indranil (Indy) Gupta and Ken Birman

What This Course is About

- US Elections
- Movies
- Travel to Mars
- Company Acquisitions
- (Not Kidding)

What This Course is Really About

- Distributed Systems
- How to Design Algorithms for them
- How to Design the Systems
- How they work in real life
- How to build real distributed systems

Our Main Goal Today

To Define the Term Distributed System

Can you name some examples of Operating Systems?

Can you name some examples of Operating Systems?

…

Linux WinXP Vista 7/8 Unix FreeBSD macOS OSX

2K Aegis Scout Hydra Mach SPIN

OS/2 Express Flux Hope Spring

AntaresOS EOS LOS SQOS LittleOS TINOS

PalmOS WinCE TinyOS iOS

…

What is an Operating System?

What is an Operating System?

- Provides abstractions (processes, files)
- Resource manager (scheduler)
- User interface to hardware (device driver)
- Means of communication (networking)
- …

FOLDOC definition

(FOLDOC = Free On-Line Dictionary of Computing)

Operating System - The low-level software which handles the interface to peripheral hardware, schedules tasks, allocates storage, and presents a default interface to the user when no application program is running.

Can you name some examples of Distributed Systems?

Can you name some examples of Distributed Systems?

- Answers that people might give:
  - Client-Server (NFS)
  - The Web
  - The internet
  - A wireless network
  - DNS
  - Gnutella or BitTorrent (peer to peer overlays)
  - A "cloud", e.g., Amazon EC2/S3, Microsoft Azure
  - A datacenter, e.g., NCSA, a Google datacenter, AWS

What is a Distributed System?

FOLDOC definition

A collection of (probably heterogeneous) automata whose distribution is transparent to the user so that the system appears as one local machine. This is in contrast to a network, where the user is aware that there are several machines, and their location, storage replication, load balancing and functionality is not transparent. Distributed systems usually use some kind of client-server organization.

Textbook definitions

- A distributed system is a collection of independent computers that appear to the users of the system as a single computer.

  [Andrew Tanenbaum]

- A distributed system is several computers doing something together. Thus, a distributed system has three primary characteristics: multiple computers, interconnections, and shared state.

  [Michael Schroeder]

Unsatisfactory

- Why are these definitions short?
- Why do these definitions look inadequate to us?
- Because we are interested in the insides of a distributed system
  - Design and implementation
  - Maintenance
  - Algorithmics ("protocols" or "distributed algorithms")

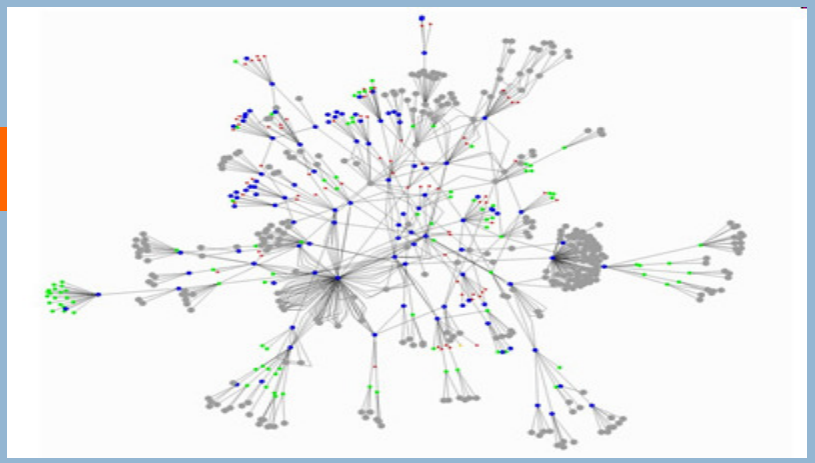
Which is a Distributed System – (A) or (B)?

(A)

(A) Facebook Social Network Graph among humans

Source: https://www.facebook.com/note.php?note_id=469716398919

(B)



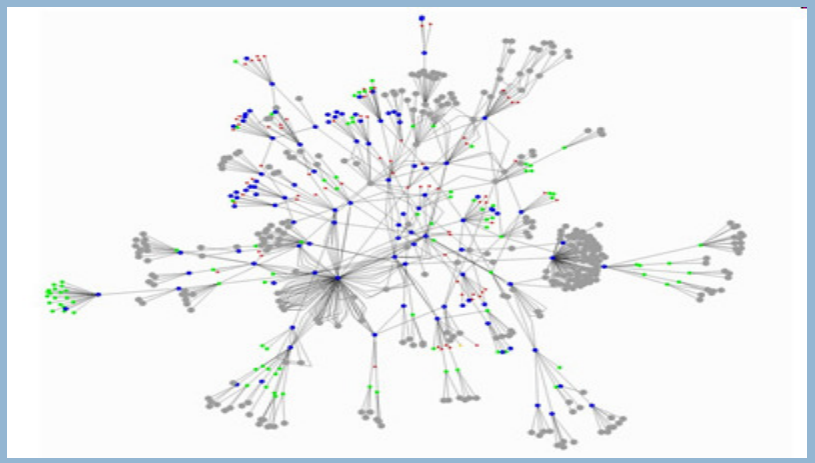
(B) Peer to peer file-sharing system (Gnutella)

A working definition for us (programmers)

A distributed system is a collection of entities, each of which is autonomous, programmable, asynchronous and failure-prone, and which communicate through an unreliable communication medium.

- Entity=a process on a device (PC, PDA)
- Communication Medium=Wired or wireless network
- Our interest in distributed systems involves
  - design and implementation, maintenance, algorithmics

Gnutella Peer to Peer System



What are the "entities" (nodes)?

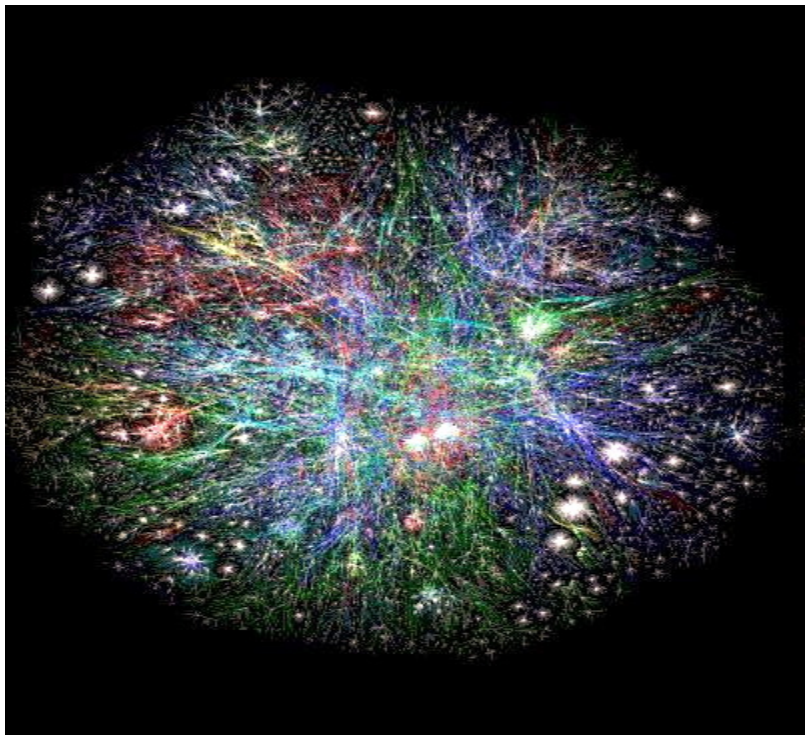
What is the communication medium (links)?

Source: GnuMap Project

Web Domains

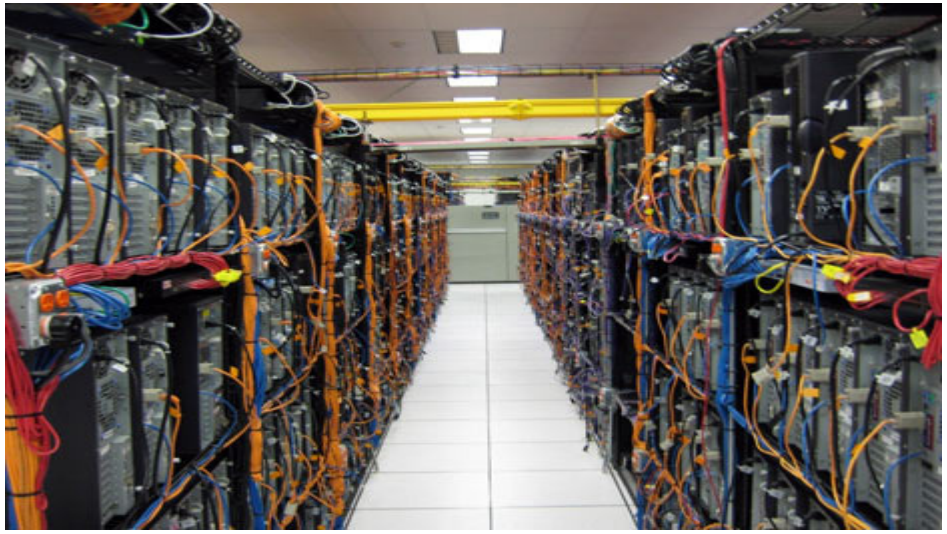
What are the "entities" (nodes)?

What is the communication medium (links)?



Source: http://www.vlib.us/web/worldwideweb3d.html

Datacenter



What are the "entities" (nodes)?

What is the communication medium (links)?

Overall Objective

How can we build distributed systems that do what we need them to do, reliably, accurately, and securely?

Critical apps are built on distributed systems

- If these distributed systems don't work
  - When we need them
  - Correctly
  - Fast enough
  - Securely and privately
- ... then revenue, health and safety, and national security may be at risk!

Examples of mission-critical applications

- Air traffic control
- Healthcare, hospital automation
- Telecommunications infrastructure
- Control of power plants, electric grid
- Banking, stock markets, stock brokerages
- Electronic commerce and electronic cash on the Web (apps built on blockchain technologies)
- Corporate “information” base: a company’s memory of decisions, technologies, strategy
- Military command, control, intelligence systems

Typical Distributed Systems Design Goals

- Common Goals:
  - Heterogeneity – can the system handle a large variety of types of PCs and devices?
  - Robustness – is the system resilient to host crashes and failures, and to the network dropping messages?
  - Availability – are data+services always there for clients?
  - Transparency – can the system hide its internal workings from the users? (warning: term means the opposite of what the name implies!)
  - Concurrency – can the system handle multiple clients simultaneously?
  - Efficiency – is the service fast enough? Does it utilize 100% of all resources?
  - Scalability – can it handle 100 million nodes without degrading service? (nodes=clients and/or servers) How about 6 B? More?
  - Security – can the system withstand hacker attacks?
  - Openness – is the system extensible?

So what makes it hard?

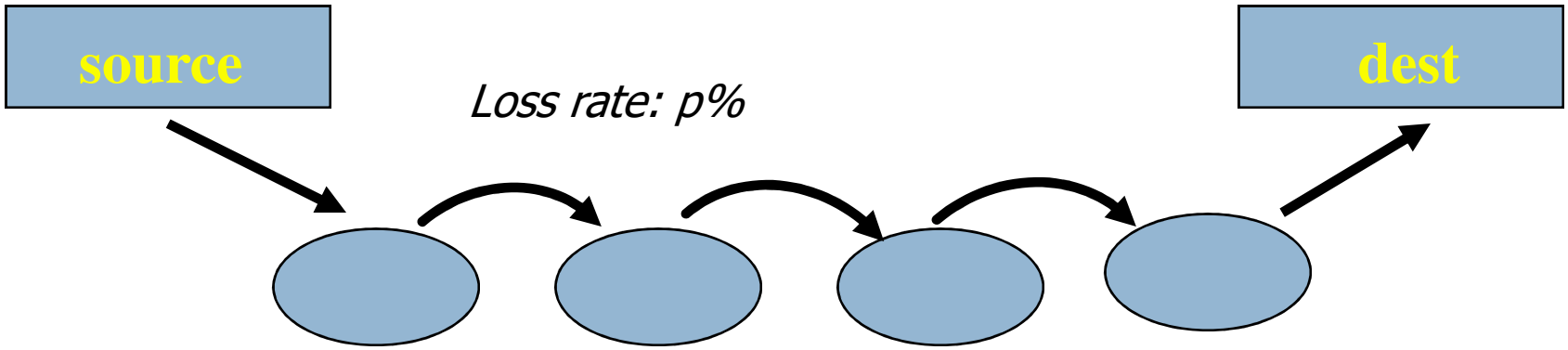
- Many prior/existing web service platforms (e.g., J2EE, .NET)
  - Enable higher productivity and provide "ease of development" but…
  - Lack automated management features
  - Handle errors in ad-hoc, inconsistent ways
  - Offer only one form of fault-tolerance mechanism (transactions), and it isn't compatible with high availability
- Developers often forced to step outside of the box… and might stumble.
  - So why don't platforms standardize such things?

End-to-End argument

- Commonly cited as a justification for **not** tackling reliability in "low levels" of a platform
- Originally posed in the Internet:
  - Suppose an IP packet will take n hops to its destination, and can be lost with probability p on each hop
  - Now, say that we want to transfer a file of k records that each fit in one IP (or UDP) packet
  - Should we use a retransmission protocol running "end-to-end" or n TCP protocols in a chain?

"End-to-End Arguments for Systems Design" by Saltzer, Read, Clark, ICDCS 1981.

End-to-End argument



Probability of successful transit: $(1-p)^n$,
Expected packets lost: $k-k*(1-p)^n$

Saltzer et. al. analysis

- If p is very small, then even with many hops most packets will get through
  - The overhead of using TCP protocols in the links will slow things down and won't benefit us much
  - And we'll need an end-to-end recovery mechanism "no matter what" since routers can fail, too.
- Conclusion: let the end-to-end mechanism worry about reliability

E2E Principle said differently…

- Don't place functionality at lower layer if entities at higher level won't use it
- Don't place functionality at lower layer if can't get it completely right there
  - Higher layer may have knowledge to get it right
  - Classic file transfer example

Generalized End-to-End view?

- Low-level mechanisms should focus on speed, not reliability
- The application should worry about "properties" it needs

- OK to violate the E2E philosophy if E2E mechanism would be much slower

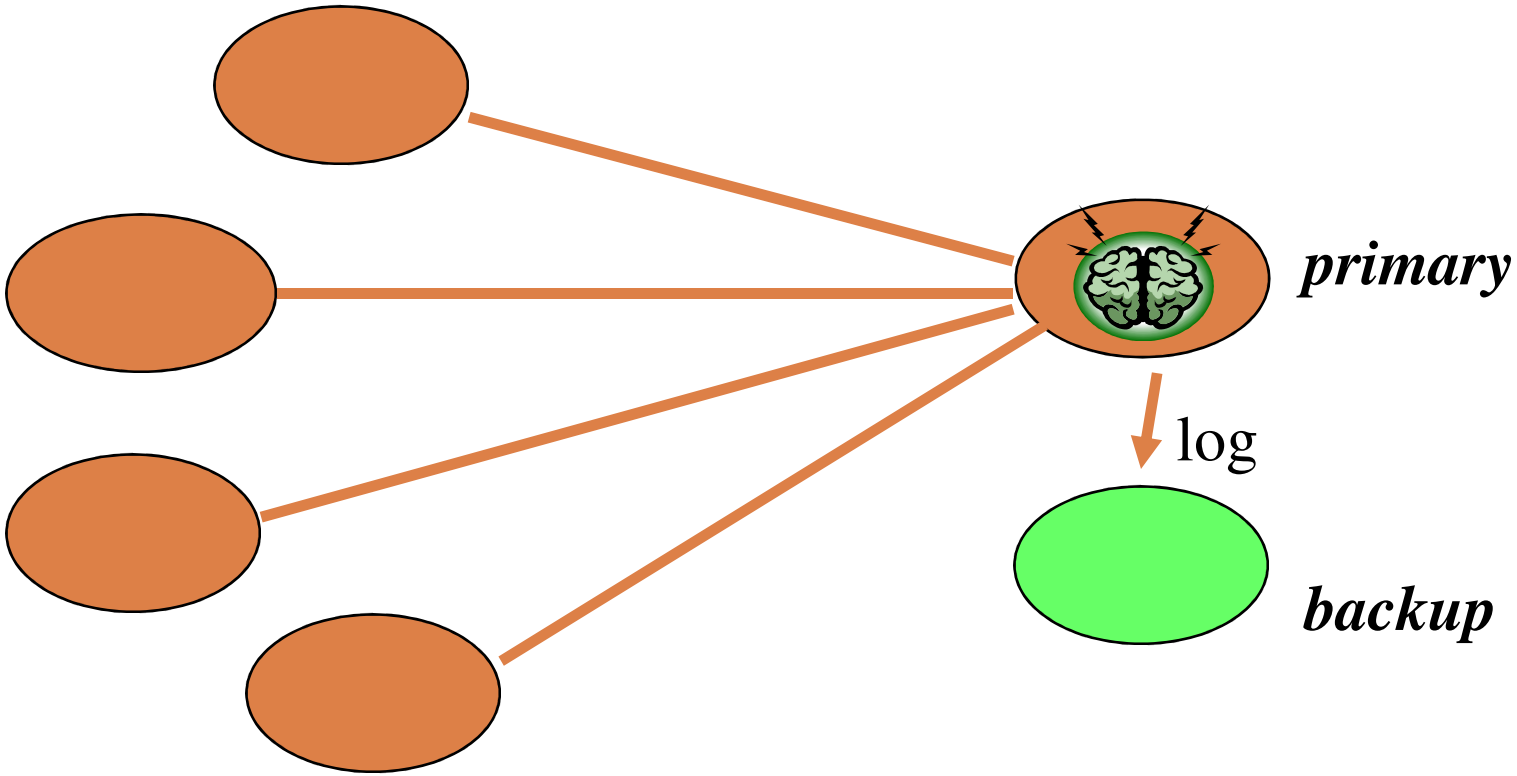
E2E argument is visible in J2EE and .NET

- If something fails, these technologies report timeouts
  - But they also report timeouts when nothing has failed
  - And when they report timeouts, they don't tell you what failed
  - And they don't offer much help to fix things up after the failure, either

E2E Example: Server replication

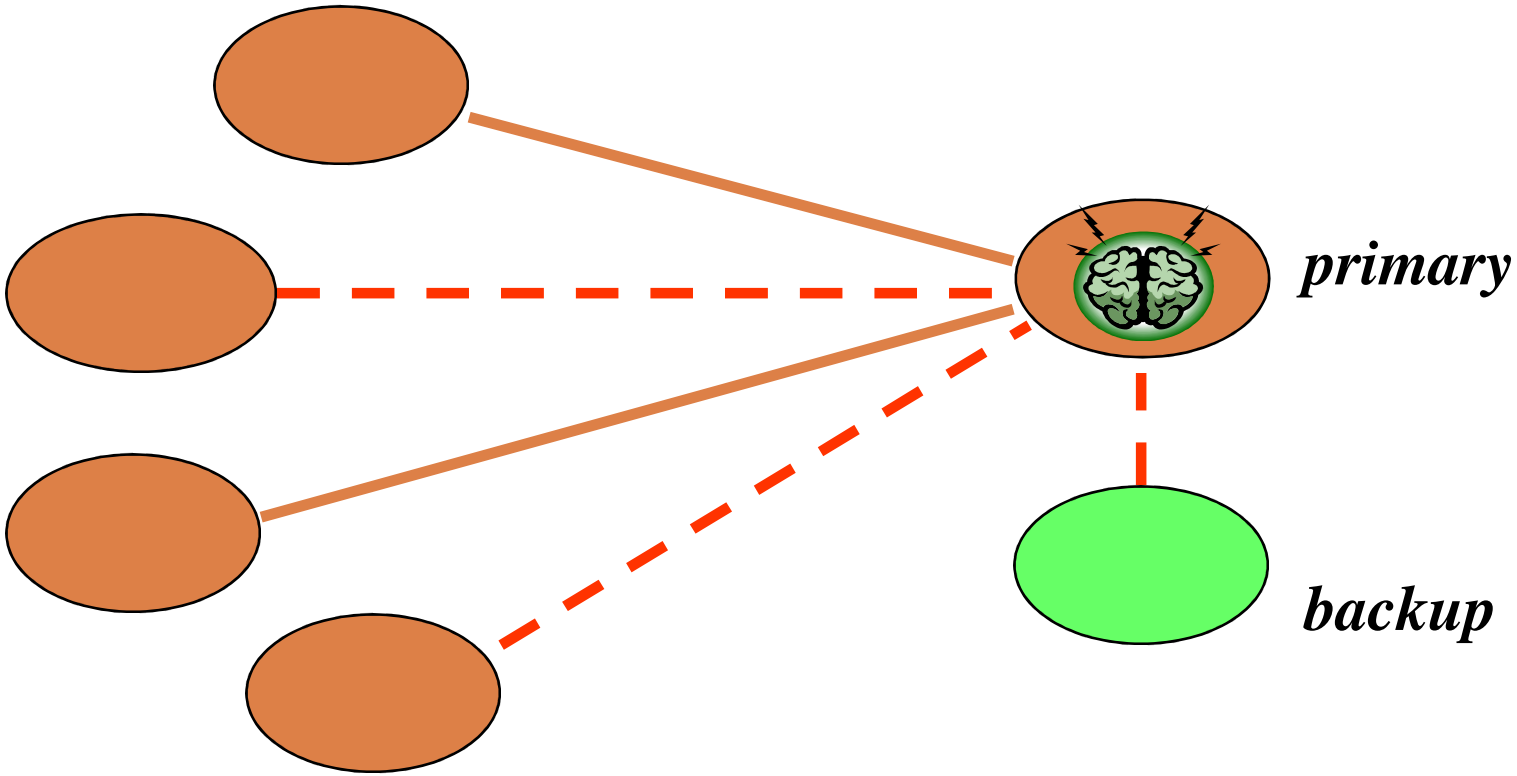
- Suppose that we have an Air Traffic Control service that needs a highly available server.
- One option: "primary/backup"
  - We run two servers on separate platforms
  - The primary sends a log to the backup
  - If primary crashes, the backup soon catches up and can take over

Split brain Syndrome…



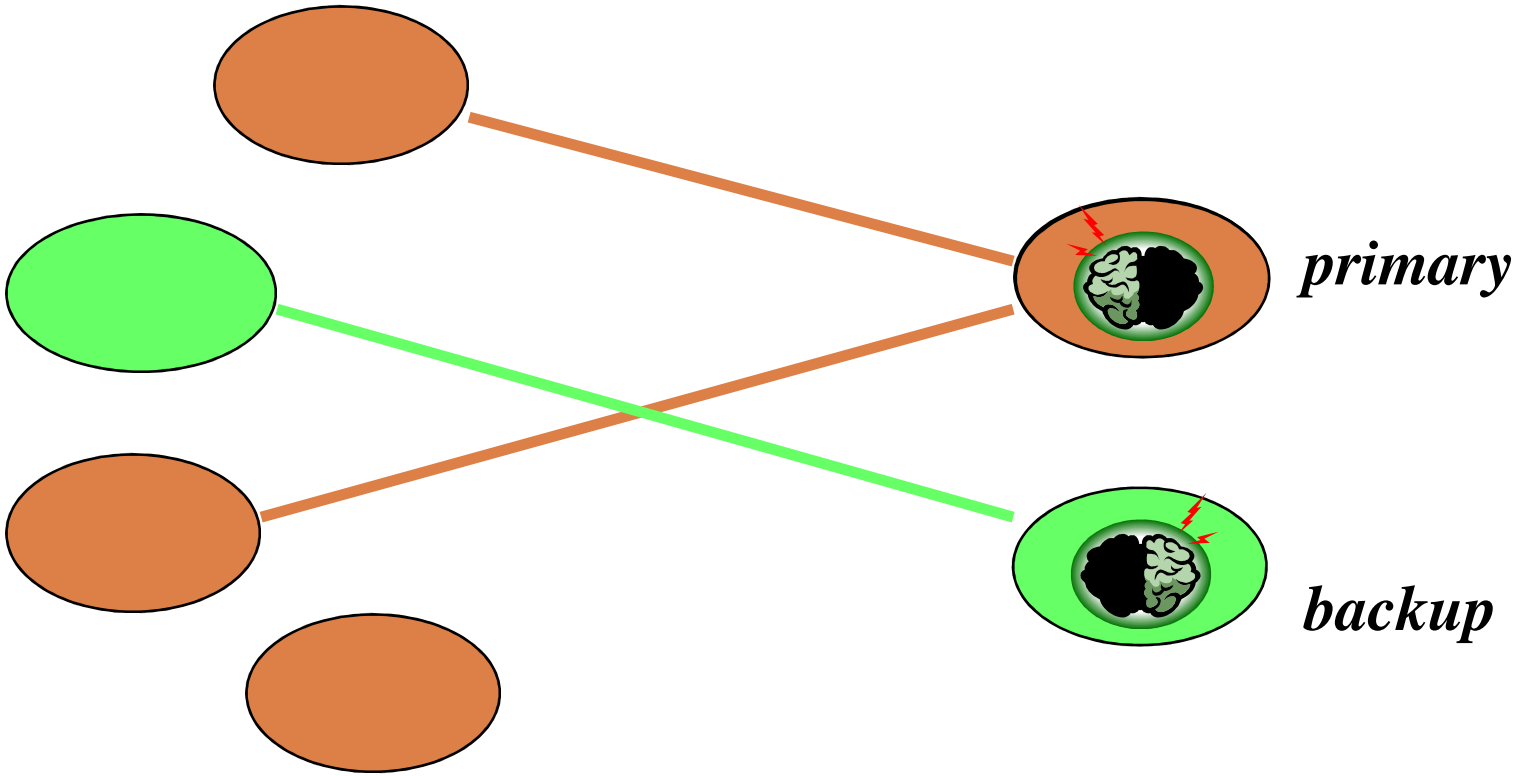
Clients initially connected to primary, which keeps backup up-to-date. Backup collects the log

Split brain Syndrome…



Transient problem causes some links to break but not all.
Backup thinks it is now primary, primary thinks backup is down

Split brain Syndrome



Some clients still connected to primary, but one has switched to backup and one is completely disconnected from both

Implication?

- Air Traffic System with a split brain could malfunction disastrously!
  - For example, suppose the service is used to answer the question “is anyone flying in such-and-such a sector of the sky”
  - With the split-brain version, each half might say “nope”… in response to different queries!

Can we fix this problem?

- No, if we insist on an end-to-end solution
  - We need some form of "agreement" on which machines are up and which have crashed
  - Can't implement "agreement" on a purely 1-to-1 (hence, end-to-end) basis.
    - Separate decisions can lead to inconsistency
    - So we need a "membership service"… and this is fundamentally not an end-to-end concept!

Can we fix this problem?

- Yes, many options, once we accept this
  - Just use a single server and wait for it to restart
    - This is common today, but too slow for ATC
  - Give backup a way to physically "kill" the primary, e.g. unplug it
    - If backup takes over… primary shuts down
  - Or require some form of "majority vote"
    - Maintains agreement on system status
- Split-brain is just one example the types of issues a distributed systems developer must handle
  - Bottom line? You need to anticipate the issue… and to implement a solution.

Typical Distributed Systems Design Goals

- Common Goals:
  - Heterogeneity – can the system handle a large variety of types of PCs and devices?
  - Robustness – is the system resilient to host crashes and failures, and to the network dropping messages?
  - Availability – are data+services always there for clients?
  - Transparency – can the system hide its internal workings from the users? (warning: term means the opposite of what the name implies!)
  - Concurrency – can the server handle multiple clients simultaneously?
  - Efficiency – is the service fast enough? Does it utilize 100% of all resources?
  - Scalability – can it handle 100 million nodes without degrading service? (nodes=clients and/or servers) How about 6 B? More?
  - Security – can the system withstand hacker attacks?
  - Openness – is the system extensible?

"Important" Distributed Systems Issues

- No global clock; no single global notion of the correct time (asynchrony)
- Unpredictable failures of components: lack of response may be due to either failure of a network component, network path being down, or a computer crash (failure-prone, unreliable)
- Highly variable bandwidth: from 16Kbps (slow modems or Google Balloon) to Gbps (Internet2) to Tbps (in between DCs of same big company)
- Possibly large and variable latency: few ms to several seconds
- Large numbers of hosts: 2 to several million –

"Important" Issues = Headache?

- If you're already complaining that the list of topics we've discussed so far has been perplexing…
  - You're right!
  - It was meant to be (perplexing)

- The Goal for the Rest of the Course: see enough examples and learn enough concepts so these topics and issues will make sense

Class Logistics

- Professor: Mema Roussopoulou
  - Office: A38
  - Office hours: by appt
- Please enroll in class Piazza forum (Required)
- Course info on readings, office hours, etc. at:

http://cgi.di.uoa.gr/~mema/courses/m120/m120.html

Class Logistics (2)

- Graduate course:
  - Prerequisites:
    - Undergrad Operating Systems
    - Computer Networks
    - Advanced OS
- Class attendance required

Textbooks and readings

- *Distributed Systems: Principles and Paradigms,* by Andrew Tanenbaum and Maarten Van Steen; Prentice Hall
- *Reliable Distributed Systems,* by Ken Birman; Springer Verlag

- We'll also read papers: Web page has references and links

Class Logistics (3)

- Lectures
  - Based on concepts in textbooks
- Paper discussions
  - Write a review (in specified format)
  - Reviews are **due via email before class time (ASCII text is fine)**
- Paper presentations
- Project

Paper Discussions

- Participate in class discussions on the papers
  - Be prepared to discuss questions/comments in detail
  - Send email before class with questions if shy asking about a particular detail
  - Print out papers and bring them to class

Paper Presentations

- You will present one or more papers during the semester
  - 25-30 minute presentation
  - Explain overall contributions of the paper
    - What problem does the paper focus on?
    - Why is this problem important?
    - What solution do the authors propose?
    - How do the authors evaluate the solution?
  - Lead the class discussion on the paper
    - This is an **interactive** class – you are not meant to drone on
    - Solicit opinions from others
    - I will help you
  - Send presentation day preferences to me to be assigned papers

Final Project

- Full specification to be defined ***by the students***
  - Problem definition
  - Approach
  - Related work
- Deliverables:
  - design document
  - programming library/framework
    - … including documentation
  - demonstrator application
- I will give you a list you can choose from

Real Research Projects

- Internet voting system
  - Vote collection backend
- Mobile phone data collection system
  - Sensors on phones, multiple data collection points per city, global archival backend
- Online health monitoring system
  - Real-time data streaming, querying, and response

Other example projects

- Fault-tolerant key-value store
  - replicated data
- File distribution service:
  - read-only
  - peer-to-peer
  - Lookup protocol + parallelized I/O
- Implement a protocol/algorithm covered in class

You can work in small teams

- Teams of 2-3 members
  - 3-person teams should tackle a more ambitious problem and will also face some coordination challenges
  - Experience is like working in commercial setting

Project Deadlines

- April 28: Project proposal
- June 7: Interim reports
- July 7: Final project, write-up, and demo/presentation

Grading

- Reviews: 10%
- Class Participation: 10%
- Paper Presentations 20%
- Final Project: 60%