Cloud Computing and Edge Computing

Ken Birman

Cornell University. CS5410 Fall 2008.

Welcome to CS5140!

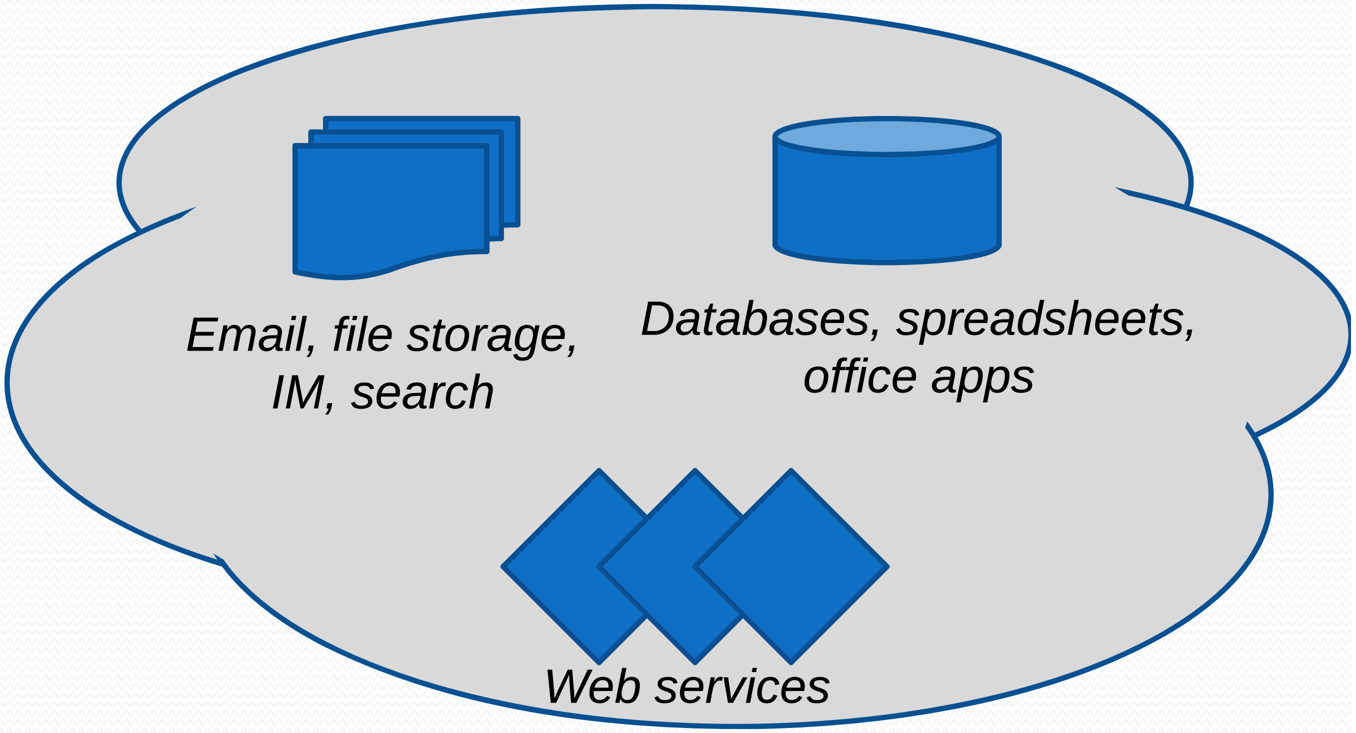
- A course on cloud computing, edge computing, and related systems technologies
- We're using a textbook written by Professor Birman, (a bit out of date). Copies on reserve.
- Grading mostly based on three assignments aimed at hands-on experience with the things we're learning in class
- Background: Java or C++ (or C#), familiar with threads, comfortable writing programs, had an architecture course and an operating systems course.

Two side-by-side revolutions

- *Cloud computing*: trend is to move more and more computing functions into large shared data centers
  - Amazon EC2 “hosts” data centers for customers
  - Google runs all sorts of office applications, email, etc on their systems
  - Yahoo! wants to be a one-source computing solution
  - IBM has a vision of computing “like electric power”
- *Edge computing*: direct interactions among computers (peers) out in the Internet
  - For example, multi-user games, VR immersion

Cloud Computing Concept

Client systems use web technologies


Email, file storage, IM, search
Databases, spreadsheets, office apps
Web services

Google/IBM/Amazon/Yahoo! host the services

Supporting technologies

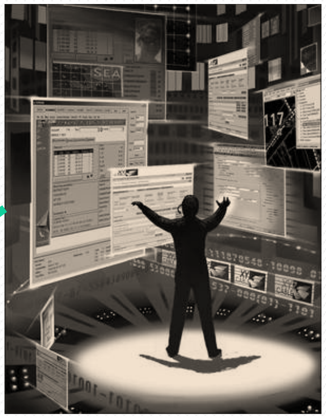
- Infrastructure
  - Core management and scheduling functions
  - Event notification services
  - Storage systems (GFS)
  - Monitoring, debugging, tuning assistance
- Increasingly: *virtualization*

- Cloud “enablers”
  - Map-Reduce
  - BigTable
  - Astrolabe
  - Amazon’s shopping cart
- Even higher level?
  - Tools for building and analyzing massive graphs

… Sadly, we can’t do everything!

- In CS5140 we don’t have time to cover all of these topics, so we’ll focus on infrastructure tools
  - You can’t build things like Map-Reduce without them!
  - But you won’t learn to use Hadoop (a popular open-source Map-Reduce implementation) in this class
- Even within the infrastructure space, we’ll pick and choose our topics to get at some of the key ideas
- Secondary issue: we also want to look at the edge

Will the next big thing happen on the edge of the network?



…. VR immersion… Distributed programming by “drag and drop”

Pause for a short demo…

http://liveobjects.cs.cornell.edu

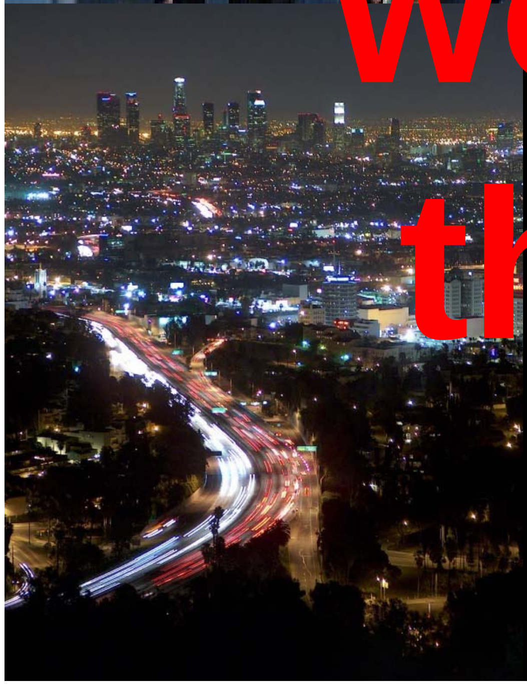
Live objects are…

- An integration tool – a “thin” layer that lets us glue components together into event-driven applications
  - A kind of “drag and drop” programming tool
- Common framework unifies replication technologies

Example Applications	
➢ Photo sharing that works	➢ Games and virtual worlds
➢ Collaboration tools	➢ Emergency response
➢ Office automation	➢ Mobile services
➢ New Internet Services	➢ Coordinated planning
➢ Interactive television	➢ Social networking

But they also depend on data center resources

- Data centers host maps, databases, rendering software
- Think of the “static” content as coming from a data center, and streams of events reflecting real-time content coming directly from sensors and “synthetic content sources”, combined on your end-user node
- All of this needs to scale to massive deployments



It’s a big, big world out on the edge….

Our goals today

- In CS5140, we'll peel back the covers
  - Try and understand major technologies used to implement cloud computing platforms
    - How did IBM/Amazon/Google/etc build their cloud computing infrastructure?
    - What tools do all of these systems employ? How are they implemented, and what are the cost/performance tradeoffs?
    - How robust are they?
  - And also, how to build your own cloud applications
    - Key issue: to scale well, they need to replicate functionality
- The underlying standards: Web Services and CORBA

How does this overlap with edge technologies?

- The edge is a world of peer-to-peer solutions
  - BitTorrent, Napster/Gnutella, PPLive, Skype, and even Live Objects
  - How are these built? What issues need to be addressed when systems live out in the wild (in the Internet)?
- But those edge solutions are invariably supported by some kind of cloud service, and in the future the integration is going to become more significant
- What happens when we graft edge solutions to cloud platforms?

Connecting the cloud to the edge

- The cloud is a good place to
  - Store massive amounts of content
  - Keep precomputed information, account information
  - Run scalable services

- The edge is a good place to
  - Capture data from the real world (sensors, cameras…)
  - Share high-rate video, voice, event streams, “updates”
  - Support direct collaboration, interaction

Topics we'll cover

- [9/3/08] Web Services and SOA standards. CORBA and OO standards
- [9/8/08] Key components of cloud computing platforms
- [9/10/08] Cloud computing applications and Map-Reduce
- [9/15/08] Thinking about distributed systems: Models of time and event ordering
- [9/17/08] Clock synchronization and the limits of real-time
- [9/22/08] Consensus on event ordering: The GMS Service(1)
- [9/24/08] The GMS Service(2)
- [9/29/08] State machine concept. Possible functionality that our GMS can support
- [10/1/08] Replication: basic goals. Ricochet
- [10/6/08] Replication with stronger semantics: Virtual synchrony
- [10/8/08] Replication with consensus semantics: Paxos
- [10/15/08] Transactional subsystems and Web Services support for the transactional model
- [10/20/08] How transactional servers are implemented
- [10/22/08] Gossip-based replication and system monitoring. Astrolabe
- [10/27/08] DHTs. Chord, Pastry, Kelips
- [10/29/08] T-Man
- [11/03/08] Trusted computing issues seen in cloud settings. Practical Byzantine Agreement
- [11/05/08] Interconnecting cloud platforms with Maelstrom. Mirrored file systems.
- [11/10/08] Life on the Edge: Browsers. BitTorrent
- [11/12/08] Sending complex functions to edge systems: Javascript and AJAX
- [11/17/08] In flight web page and data modifications and implications. Web tripwires
- [11/19/08] Pure edge computing: Gnutella
- [11/24/08] Resilient Overlay Networks. PPLive

Stylistic comments

- One way to approach CS5140 would focus on a how-to way of using standards and packages
  - For example, Microsoft favors Indigo as their web services solution for Windows platforms
  - We could spend 12 weeks learning everything we can about Indigo, do projects using Indigo, etc.
  - You would emerge as an "Indigo expert"

Stylistic comments

- A second extreme would be to completely ignore the web services standards and focus entirely on the theory
  - We would discuss ways of thinking about distributed systems
  - Models for describing protocols
  - Ways of proving things about them

- You would be left to apply these ideas “as an exercise”

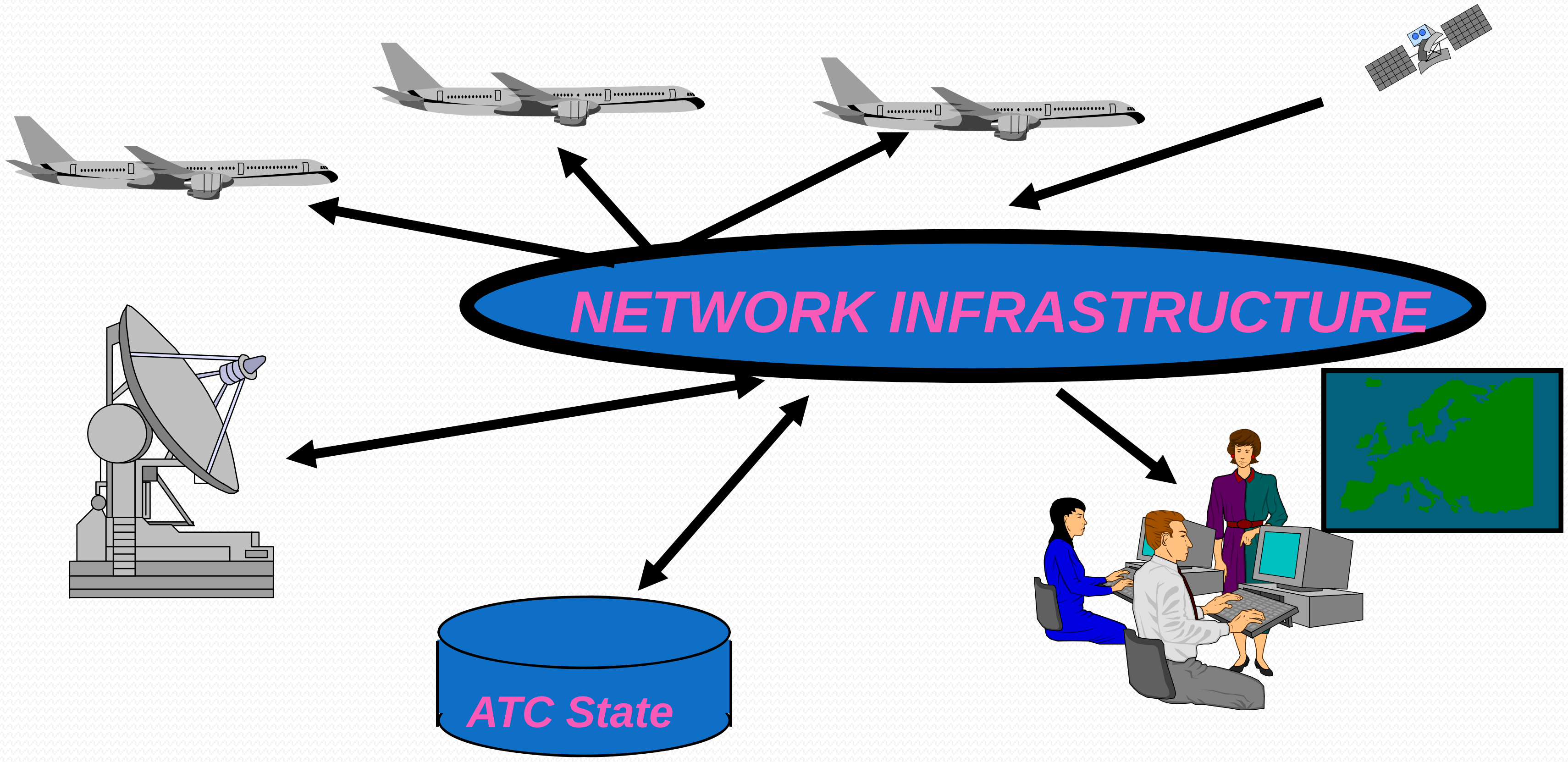
CS5140: Pursuing the middle

- The class will try and live somewhere in the middle
  - About half of our lectures are on very concrete real systems, like BitTorrent or Chubby, and how they work
  - And about half our lectures are concerned with the platform standards and structures and how they look
  - A few lectures focus on the underlying theory
- Homework assignments will involve building real (but simple) distributed systems using these ideas
  - For this, we'll work with Windows platforms & technology

Let’s look at an example

- To illustrate the way the class will operate, let’s look at a typical example of a problem that cuts across these three elements
  - It arises in a standard web services context
  - But it raises harder questions
  - Ultimately, theoretical tools help us gain needed clarity

ATC Architecture



ATC status is a kind of temporal database: for each ATC sector, it tells us what flights might be in that sector and when they will be there

Server replication

- Let's think about the service that tracks the status of ATC sectors
  - Client systems are like web browsers
  - Server is like a web service. ATC is a "cloud" but one with special needs: it speaks with "one voice"

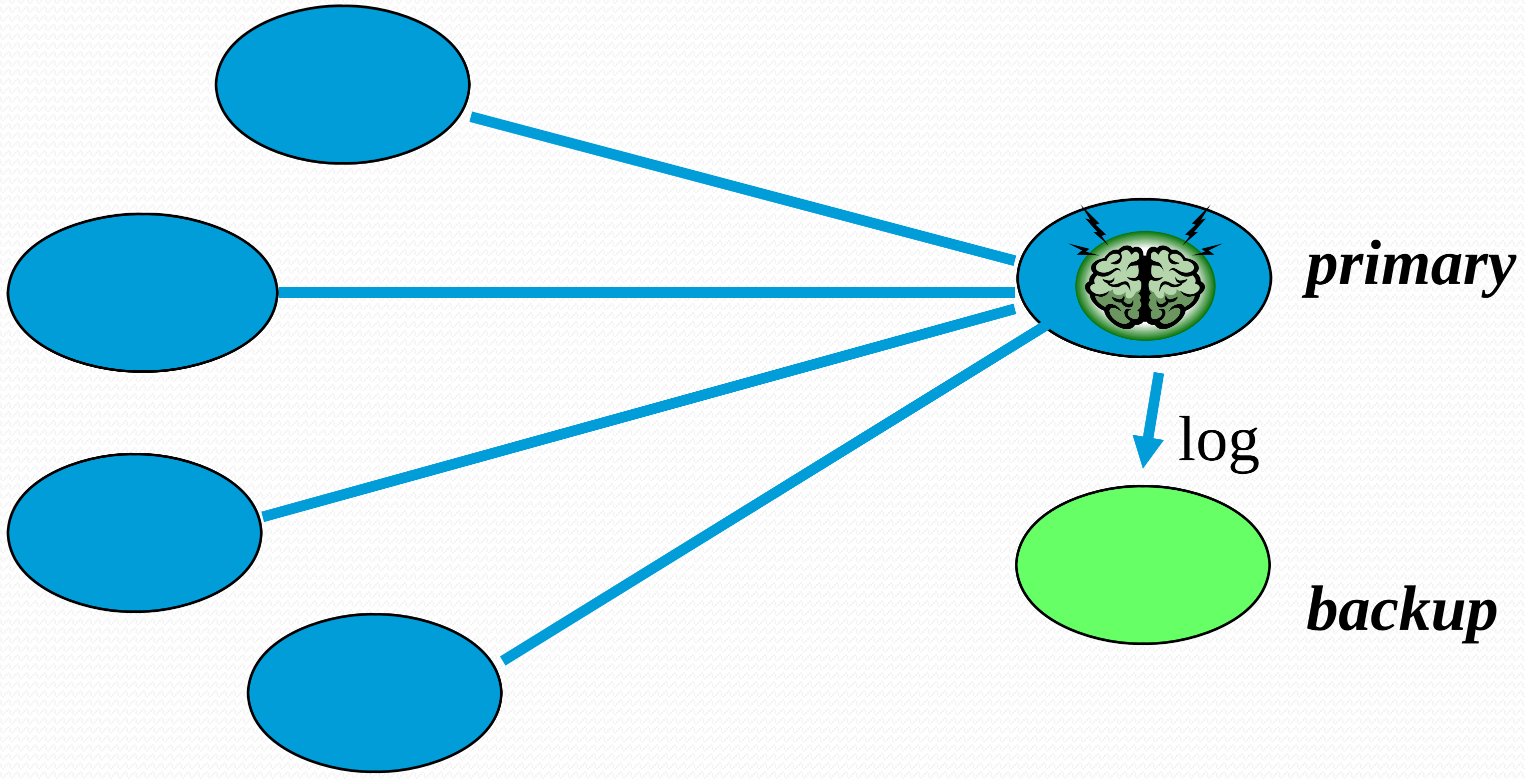
- Now, an ATC needs *highly available* servers.
  - Else a crash could leave controller unable to make decisions
  - So: how can we make a service highly available?

Server replication

- Key issue: we need to maintain that "one voice" property
  - Behavior of our highly available service needs to be indistinguishable from that of a traditional service running on some single node that just doesn't fail

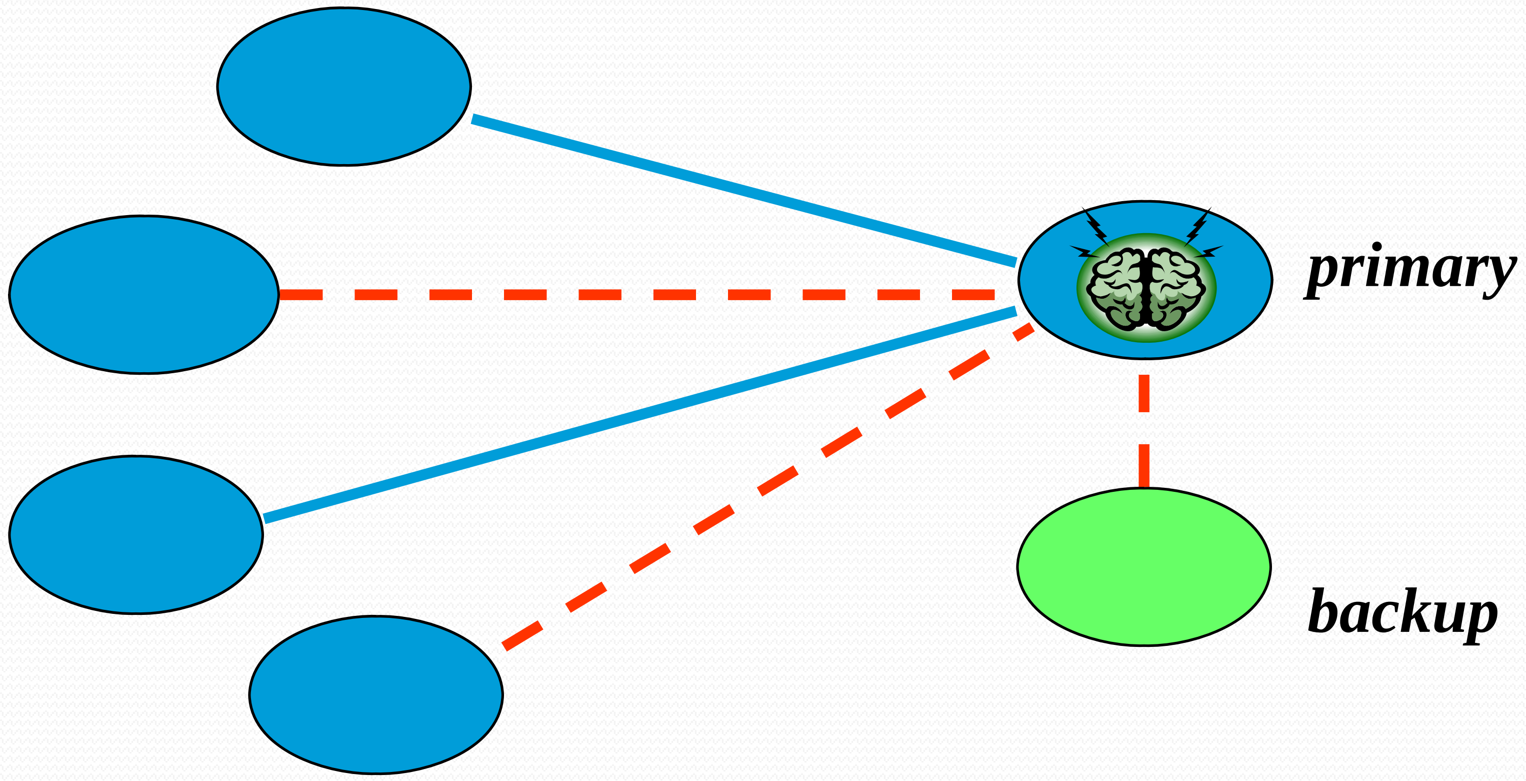
- Most obvious option: "primary/backup"
  - We run two servers on separate platforms
  - The primary sends a log to the backup
  - If primary crashes, the backup soon catches up and can take over

A primary-backup scenario…



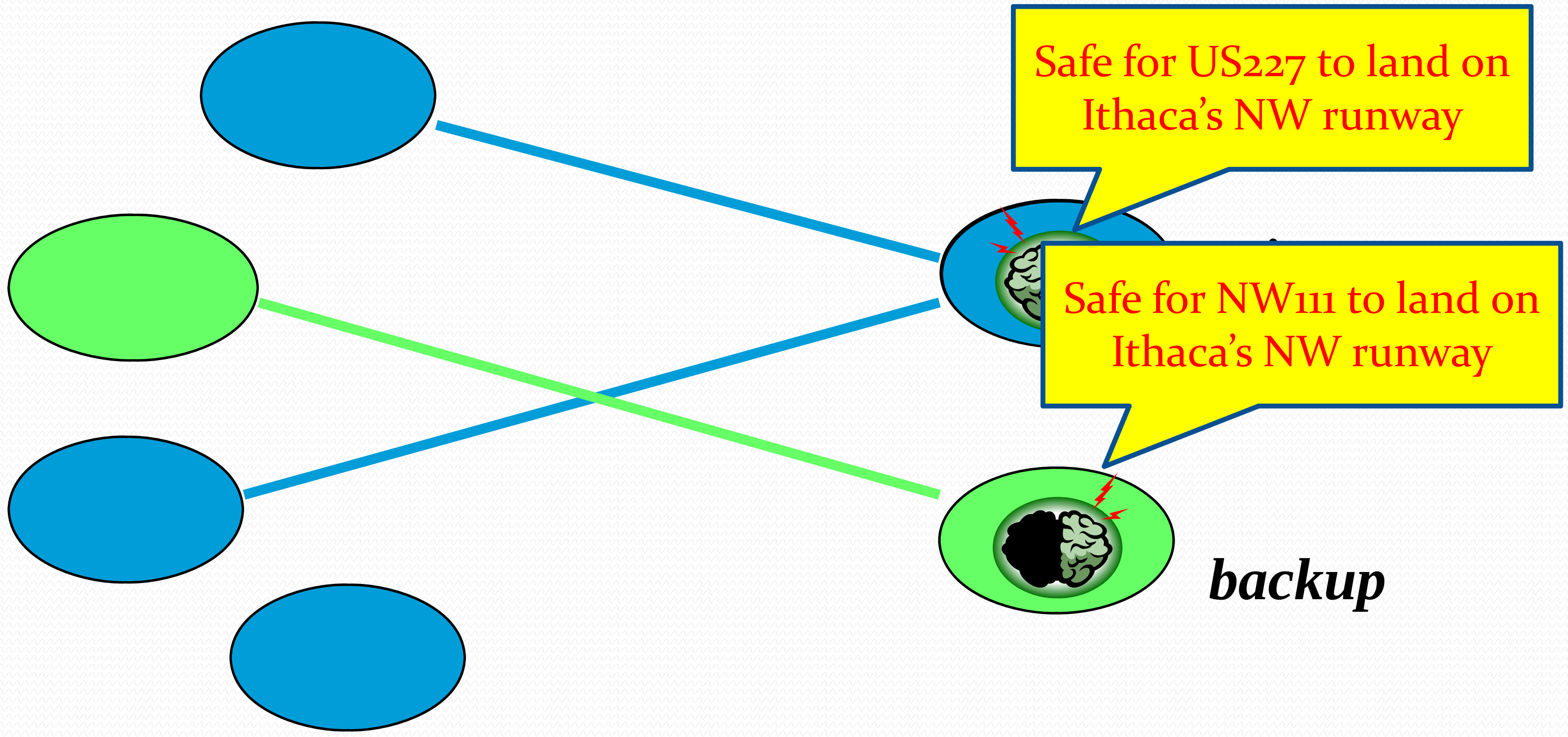
Clients initially connected to primary, which keeps backup up to date. Backup collects the log

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


Safe for US227 to land on Ithaca's NW runway
Safe for NW111 to land on Ithaca's NW runway
backup

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

Oh no! But could this happen?

- How do web service systems detect failures?
  - The specifications don't really answer this question
  - A web client senses a failure if it can't connect to a server, or if the connection breaks
  - And the connections are usually TCP

- So, how does TCP detect failures?
  - Under the surface, TCP sends data in IP packets, and the receiver acknowledges receipt.
  - TCP channels break if a timeout occurs.

Provoking a transient fault

- Build a fairly complex network with some routers, multiple network segments, etc
- Run TCP over it in the standard way
- Now disrupt some core component
  - TCP connections will break *over a 90 second period*
  - So… restore service after perhaps 30 seconds. Some break, but some don't.

Implication?

- We end up with multiple servers that might each think they are in charge of our ATC system!

- An ATC 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!

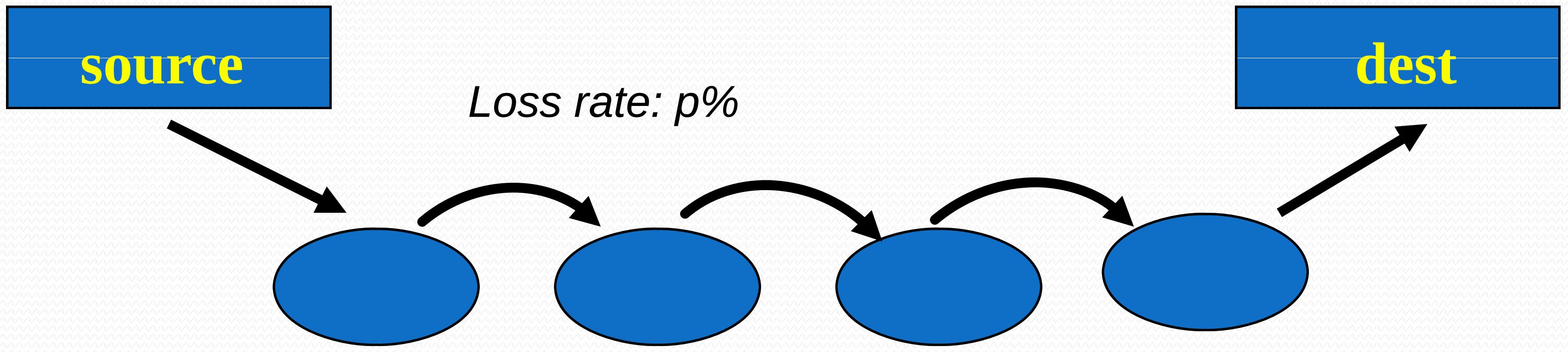
Can we fix this problem?

- Sure. Just have the backup unplug the primary
- But less draconian solutions are also possible
  - We'll look at this issue later in the class
  - Need "agreement" on which machines are up and which have crashed
  - Can't implement "agreement" on a purely 1-to-1 (also called "end-to-end") basis.
    - Separate decisions can always lead to inconsistency
    - So we need a "membership service"… and this is fundamentally not an end-to-end concept!

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 argument



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

Saltzer et. al. analysis

- If p is <u>very</u> 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 often benefit us
  - 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

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 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

- Timeouts and transient faults can't be distinguished
  - Thus we can always detect failures.
  - But we'll sometimes make mistakes.

But why do cloud systems use end-to-end failure detection?

- ATC example illustrated a core issue
- Existing platforms
  - Lack automated management features
  - Inherit features of the Internet… even ones that embody inappropriate behavior
  - In this example, we saw that TCP handles errors in ad-hoc, inconsistent ways
- Developers often forced to step outside of the box… and may succeed, but might stumble.
- In CS5140 we'll try and tackle some of these questions in a more principled way

Even this case illustrates choice

- We have many options, if we are willing to change the failure semantics of our platform
  - Just use a single server and wait for it to restart
    - This common today, but too slow for ATC
    - Cloud computing systems usually need *at least* a few seconds
  - 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" and implement this in the cloud computing platform itself
    - System maintains agreement on its own structure
- Later we'll see what the last of these options entails

Elements of cloud computing

- One perspective: the laundry list of tools and technologies we saw earlier

- A second perspective: a collection of abstractions and assumptions that the cloud needs to implement, and that the developer can then "trust"
  - For example, if the cloud were to implement a failure detection mechanism, the developer could trust it, and split brain problems would be avoided
  - We'll generalize this way of thinking. *The cloud is a provider of abstractions.* Specific tools implement those abstractions

ATC example illustrates…

- A form of replication (one form among many)
- An example of a consistency need (one kind of consistency but not the only kind)
- A type of management "implication" associated with that consistency need
- A deeper question of what it means for a system to agree on the state of its own members

How can a system track its own membership?

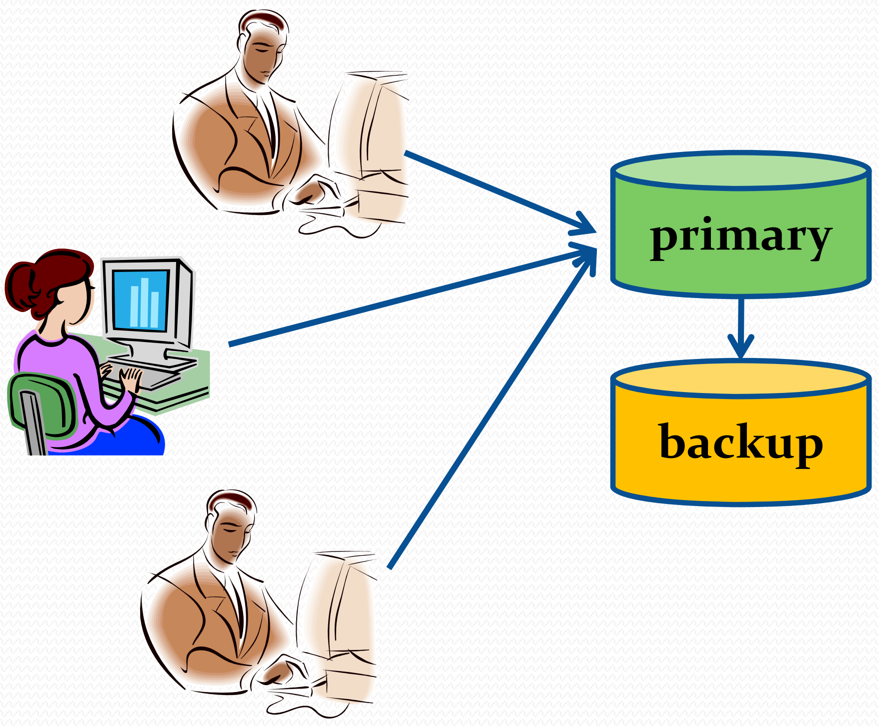
- We’ve discussed the idea that a cloud might offer users some form of VMM abstraction
  - E.g. Amazon.com might tell Target.com “we’ll host your data center” but rather than dedicate one machine for each server Target thinks it needs, Amazon could virtualize the servers and schedule them efficiently

- So… let’s virtualize the concept of failure handling

Typical client-server scenario

TCP used for connections

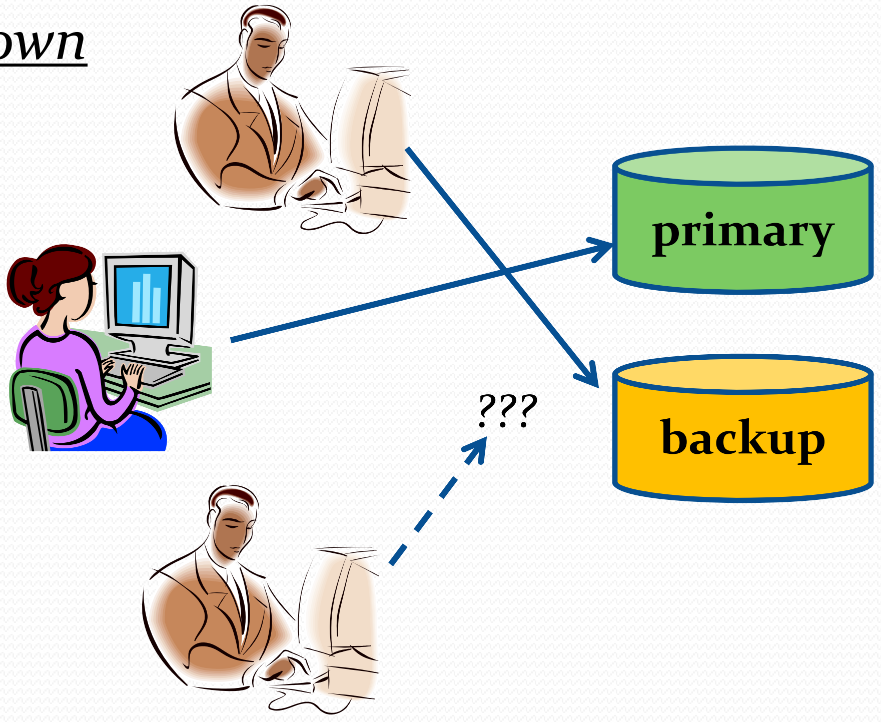
- Each channel has its own timers
- If a timeout occurs, clients "fail-over" to the backup



Split brain scenario

Potential for inconsistency

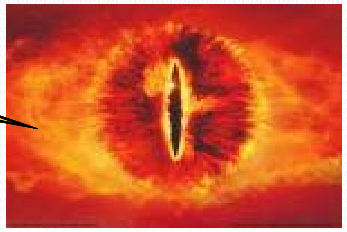
- Each client makes *<u>its own</u>* decisions
- Outcome can be inconsistent
- Concern: "split brain" behavior



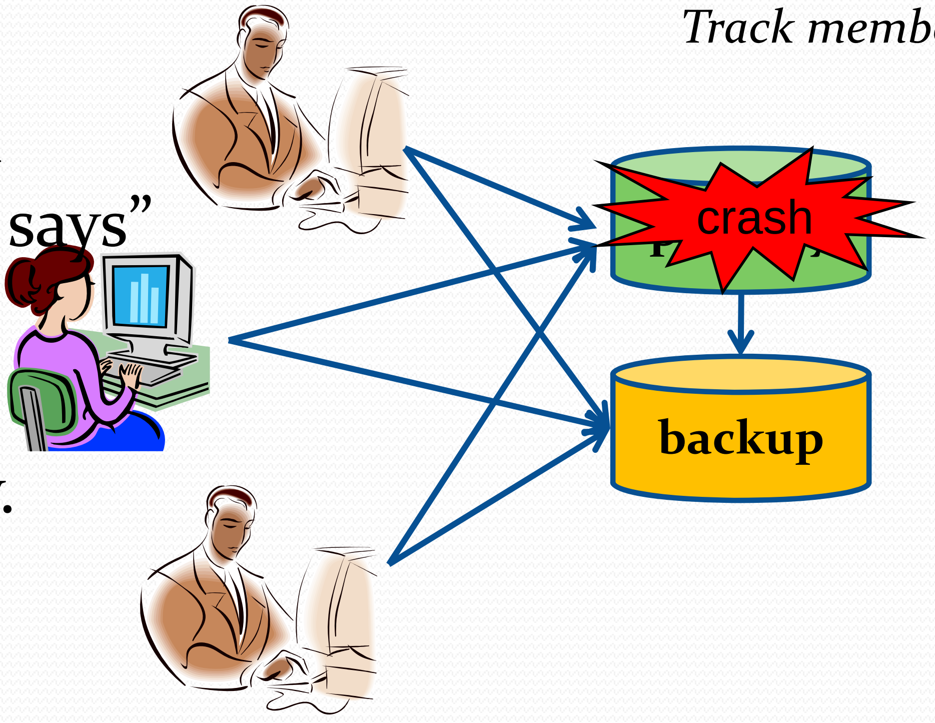
Role of an Oracle

An all-seeing eye.

- Clients must obey it
- If the oracle makes a mistake, we "do as it says" anyhow
- This eliminates our fear of inconsistency.
- Now we just hope mistakes are rare!



Track membership



Oracle

- A kind of all-seeing eye that *dictates* the official policy for the whole data center
  - If the Oracle says that node X is up, we keep trying to talk to node X
  - If the Oracle says node X is down, we give up

- An Oracle imposes a form of consistency

Oracle

- Later we'll see that an Oracle can be implemented in a decentralized way
  - Some (perhaps all) nodes run a service
  - The service elects a leader, and the leader makes decisions (if you think a node is faulty, you tell it)
  - If the leader fails, a majority election is used to elect a new leader

- By continously requiring majority consent, we guarantee that split-brain cannot occur.

Stepping back

- So one goal of CS5140 is to look at these issues on the boundary of
  - What technologies can and can’t do
  - What we can do to overcome or evade the limits
  - The associated theory
- A second goal is to think about how to structure systems into more standard pieces

Back to the edge…

- Today's focus was on issues see in cloud settings.

- But similar questions arise in peer-to-peer systems used for file sharing, telephony, games, and even live objects
  - Not the identical notion of consistency and the style of solutions is different
  - We'll look at P2P replication… event notification… building structured overlays… well known applications

Wrapping up

- CS5140 lectures will look at technologies and issues such as the ones just reviewed
- CS5140 projects will
  - Give you hands-on experience using Windows to build web services and clients that talk to them
  - And some limited experience using the solutions we identify in class in the context of those services
- Grading: Mostly based on the three assignments
  - First two will be done individually; third can be done in small teams

Meng credit

- The CS5140 project can be used for Meng project credit
  - You must sign up for CS7900 credit with Ken Birman
  - We recommend 3 credits, letter grade
  - Your grade will be identical in CS5140, CS7900
  - You are expected to tackle a slightly more ambitious problem to get the extra credit
- Typically, cs7900 entails doing a more detailed experimental evaluation of assignment three and reporting your findings