1. Cloud Computing
2. Edge Computing (P2P Computing)

Guide to Reliable Distributed Systems:

Building High-Assurance Applications and Cloud-Hosted Services

Book by Ken Birman

2021.

Distributed Systems

- 1. LAN based systems: Banking, ATMs,
- → synchronous computing, time-limit on response, most of research traditional researh studies, uses rpc
- ----------------
- 2. Internet based systems: University mail system, grid computing,
- → asynchronous computing, no-time limit, disconnections and delays, new research activity, depends on web services
- 3. Web based systems: gmail, google docs, cloud computing,
- → asynchronous computing, no-time limit, disconnections and delays, new research activity, depends on web services

Basic Communication Mechanisms

- RPC → Remote Procedure Call → called procedure is on a remote host: It is a basic protocol for distributed system communication (message passing). It is a protocol (not a standard protocol).

- Web Services → Aim to provide a basic communication mechanism on web, to connect remote procedures using open standards

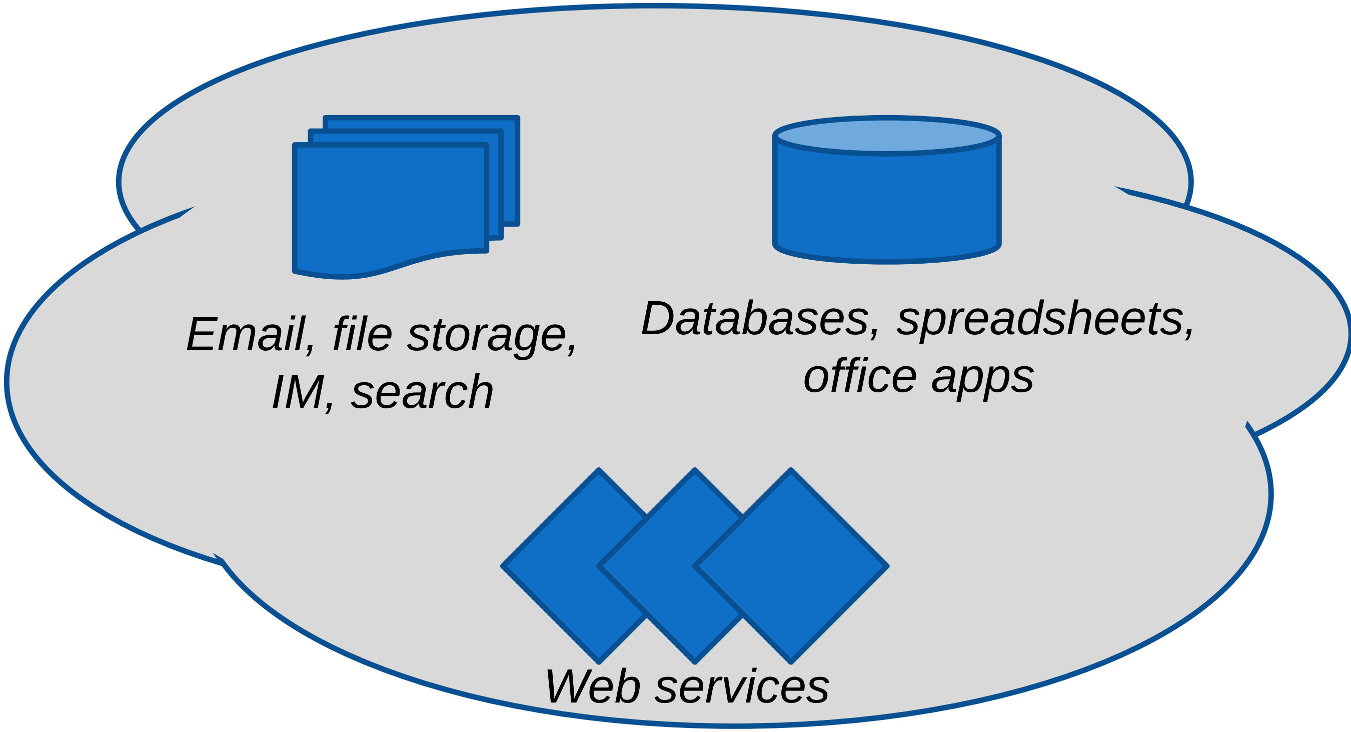
Web Based Distributed Systems: Two revolutions → Side-by-side

- *Cloud computing*: Move computing functions into →large shared data centers (virtualize + externalize)
  - Amazon EC2 → "hosts" data centers for customers
  - Google → runs all sorts of office applications, email, etc
  - Yahoo! → wants to be a one-source computing solution
  - IBM → has a vision of computing "like electric power"

- *Edge computing (peer-to-peer):* → direct interactions among computers (peers) out in the Internet
  - multi-user games
  - VR immersion

Cloud Computing Concept: → EARLY STAGE YEAR 2 IN 2018

Client systems use web technologies



Email, file storage, IM, search

Databases, spreadsheets, office apps

Web services

Google/IBM/Amazon/Yahoo! host the services

Peer-to-Peer (p2p) Systems

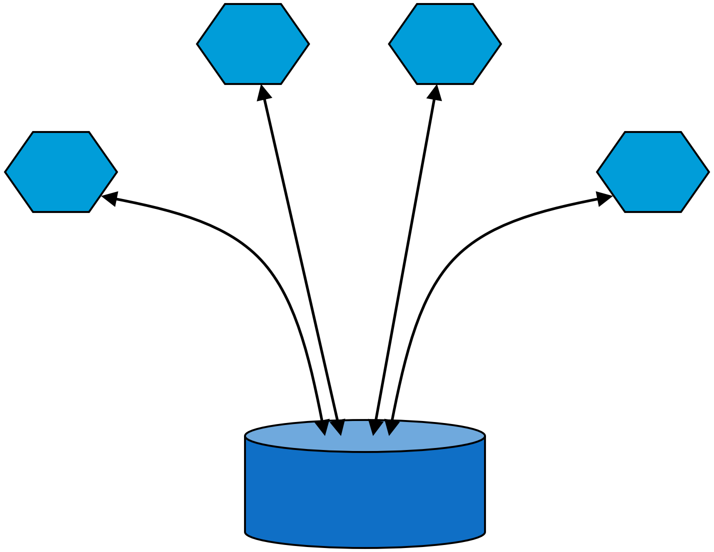
- P2P → a kind of distributed computing system,

  → "main" service is provided by having the client systems talk directly to one-another

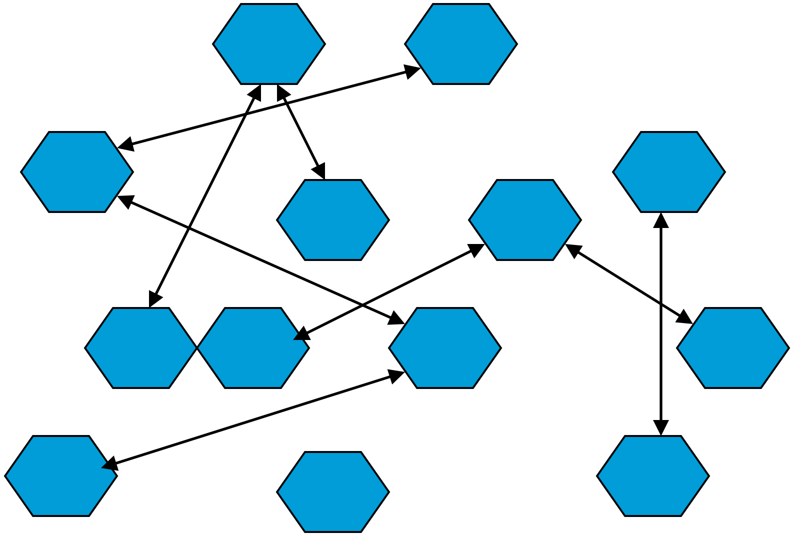
- [Contrast] : Traditional systems are structured with servers at the core and clients around the edges

p2p systems

Standard systems:
Client/Server structured



P2P systems: Clients help
one-another out

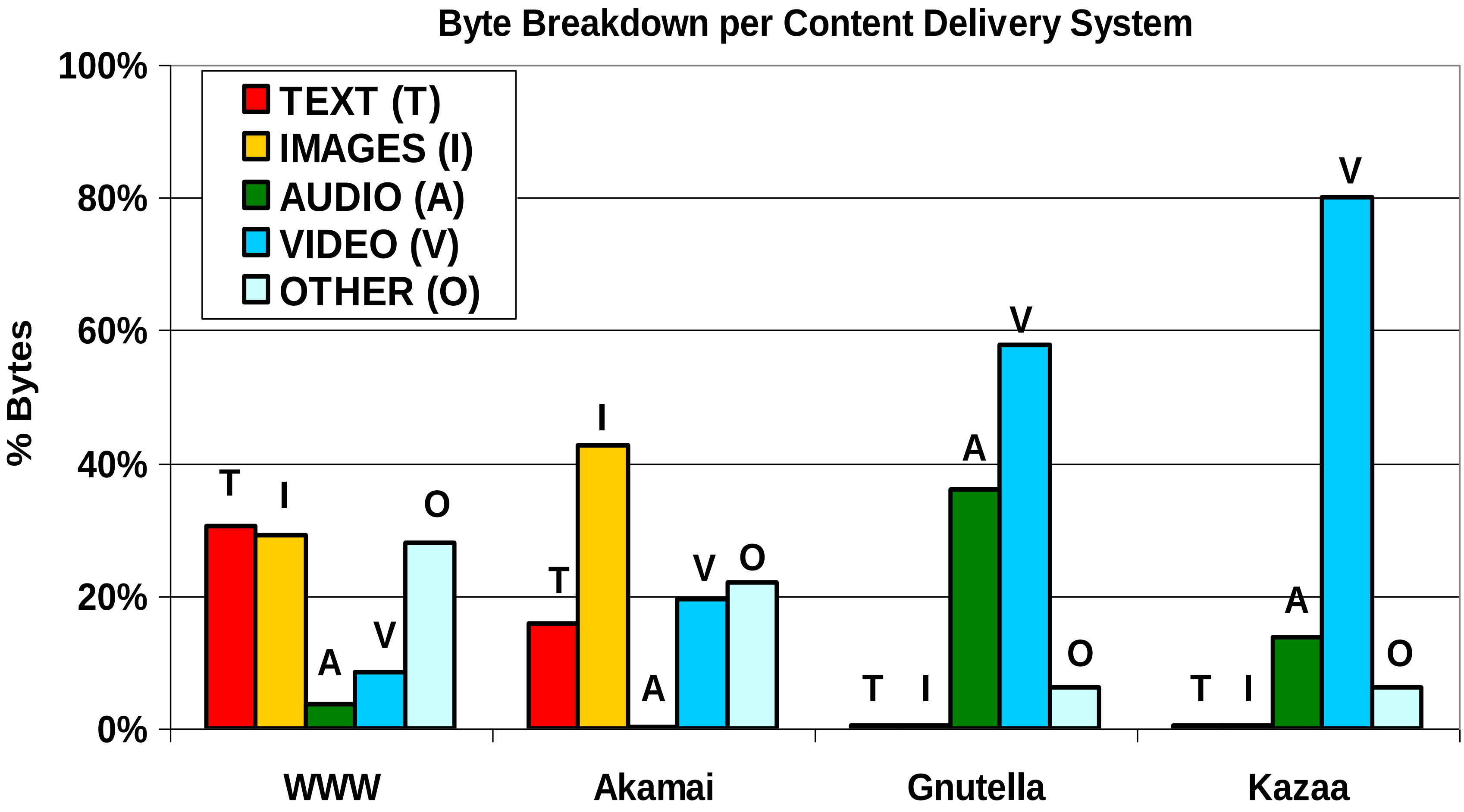


P2P is an “important” topic

- … Topic
- Recording industry → p2p downloads→ big loss of profits
  - Used to be mostly file sharing,

    but now online radio feeds (RSS feeds)
  - A U. of Wash. study → 80% of their network bandwidth was spent on music/video downloads!
  - DVDs
  - Selected set of many objects were downloaded *many* times
  - Many downloads took <u>months</u> to complete…
  - Most went to a tiny handful of machines

Object type for different systems



Source: Hank Levy. See http://www.cs.washington.edu/research/networking/websys/pubs/osdi_2002/osdi.pdf

P2P - An Overview

- Origins: Illegal fire sharing
- Early academic work: "Distributed hash tables"

P2P - An idea…

- Most of the Distributed Systems study (protocols) are "peer to peer" in a broad sense
  - Lamport was interested in uses direct client-to-client communication
  - Group communication systems often do have servers, but not all need them…

- But the term really has a stronger meaning
  - Denotes systems where the "data that matters" is passed among cooperating client systems
  - And there may be huge numbers of clients

Attributes of p2p systems

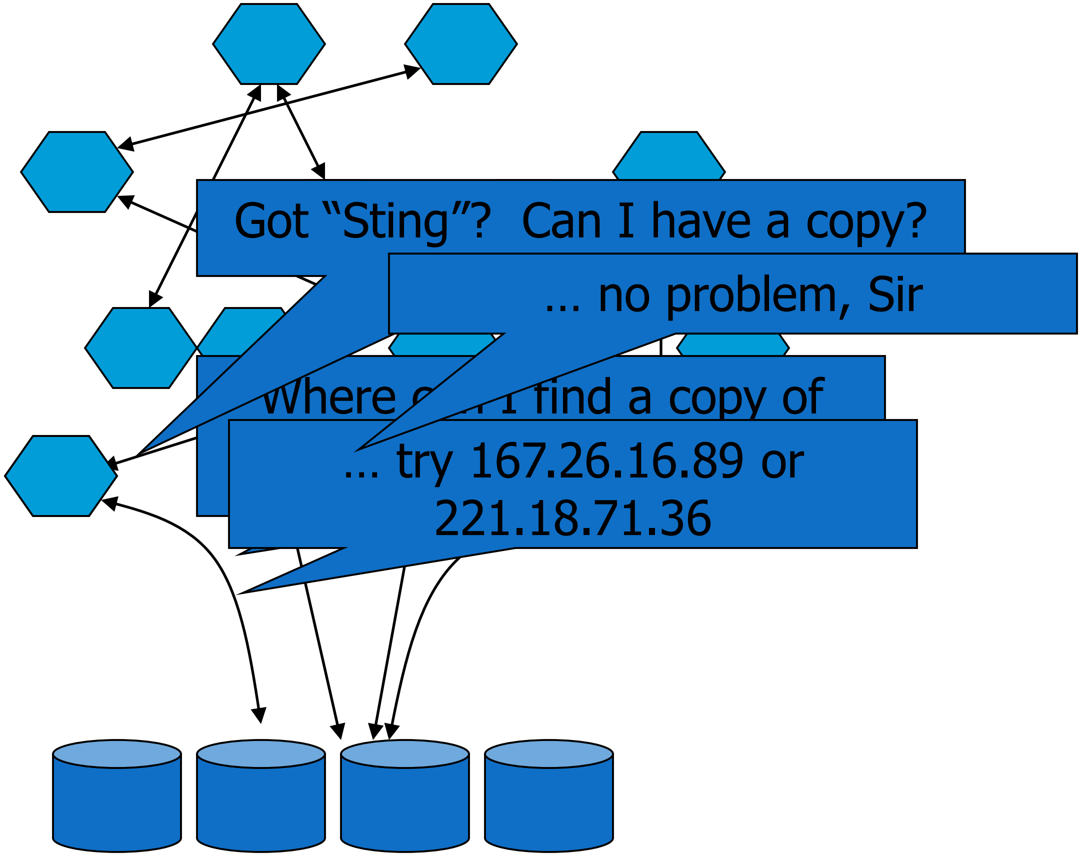
- Enormous Size
  - Hundreds of thousands or millions of client nodes, coming and going rapidly
  - If there are any servers → small in number and have limited roles
- These clients are everywhere
  - Even in Kenya or Nepal… places with low network connectivity

The first peer-to-peer system

- The idea, emerged from the Napster file sharing service
  - In fact, Napster *has* a set of servers
  - Keeps a directory on behalf of clients and orchestrate publicity inserts
  - Servers build the web pages that the users see
  - Actual music and DVD downloads are done from client to client

Napster

Having obtained a top-level page listing peers with copies of music or other content desired, a client can download the files directly from the peer


Got “Sting”? Can I have a copy?
… no problem, Sir
Where can I find a copy of
… try 167.26.16.89 or 221.18.71.36

Data center builds the pages users see when they access Napster

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

What happen on the edge of the network?

…. VR immersion…

Distributed programming by “drag and drop”

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

P2P Computing depends 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

Goals for study

  - 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

Cloud overlap with edge technologies?

- Edge is a world of peer-to-peer solutions
  - BitTorrent, Napster/Gnutella, PPLive, Skype, and even Live Objects

- Edge solutions → Supported by some kind of cloud service

- → In future the integration is to become more significant

Coclusion: [cloud ←→ edge]

- The cloud is good
  - To Store massive amounts of content
  - To Keep precomputed information, account information
  - To 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 of Interest

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

Related comments

- way of using standards and packages

  (learning time 10 weeks)

  - For example,

  Microsoft favors Indigo as their web services solution for Windows platforms

Related comments

- Discuss ways of thinking about distributed systems
- Models for describing protocols
- Ways of proving things about them
- LAN based systems: Distributed Systems
- Internet based systems:
- Web based systems:

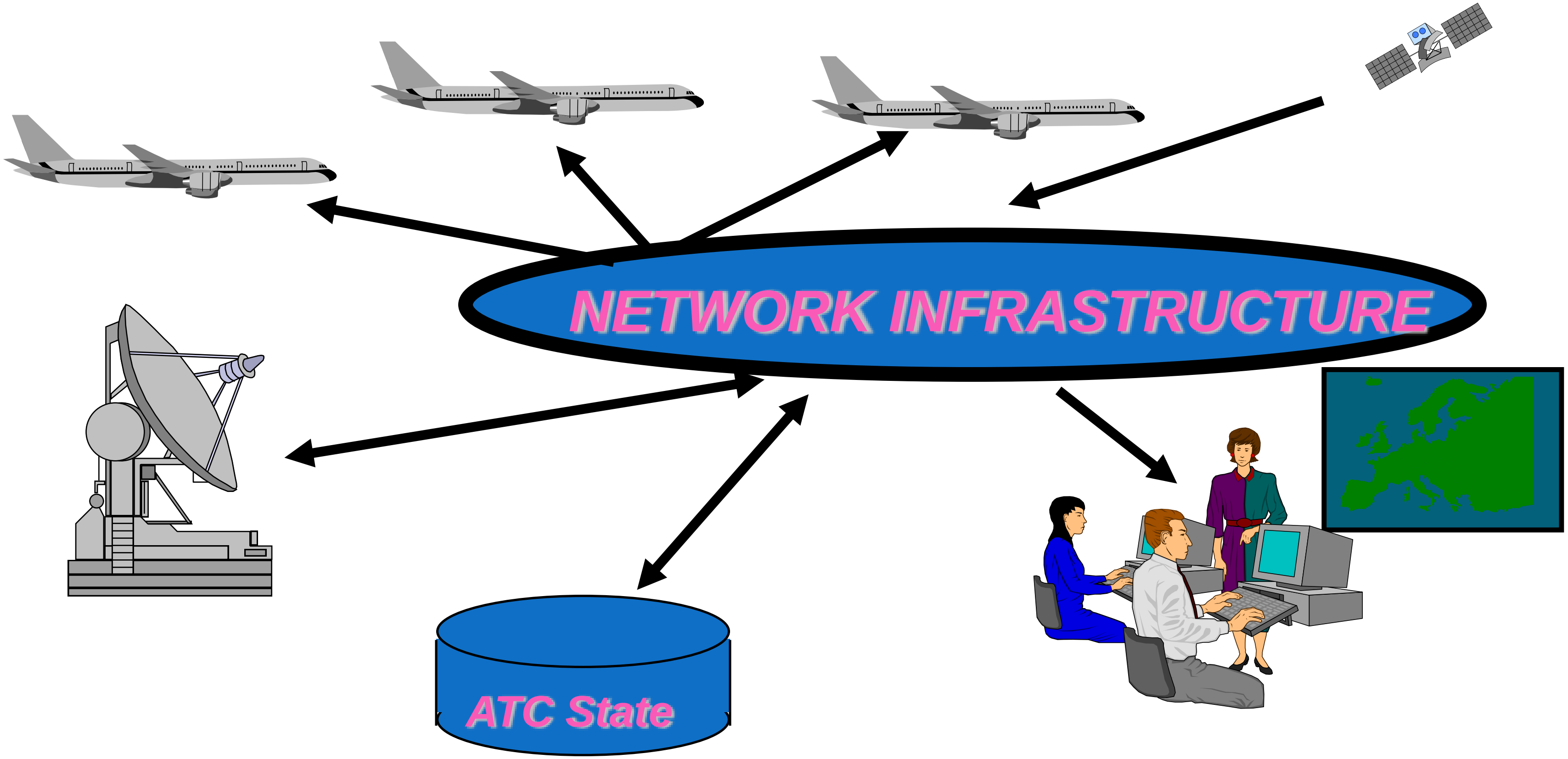
Real Systems

- concrete real systems, like BitTorrent or Chubby, and how they work
- platform standards and structures and how they look
- underlying theory

Look at problems as an example: What do the Cloud-based systems can or cannot do ?

- 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

Air Traffic Control (ATC) Architecture



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

Server Replication

- Service that tracks the status of ATC sectors
  - Client systems → web browsers
  - Server → web service.

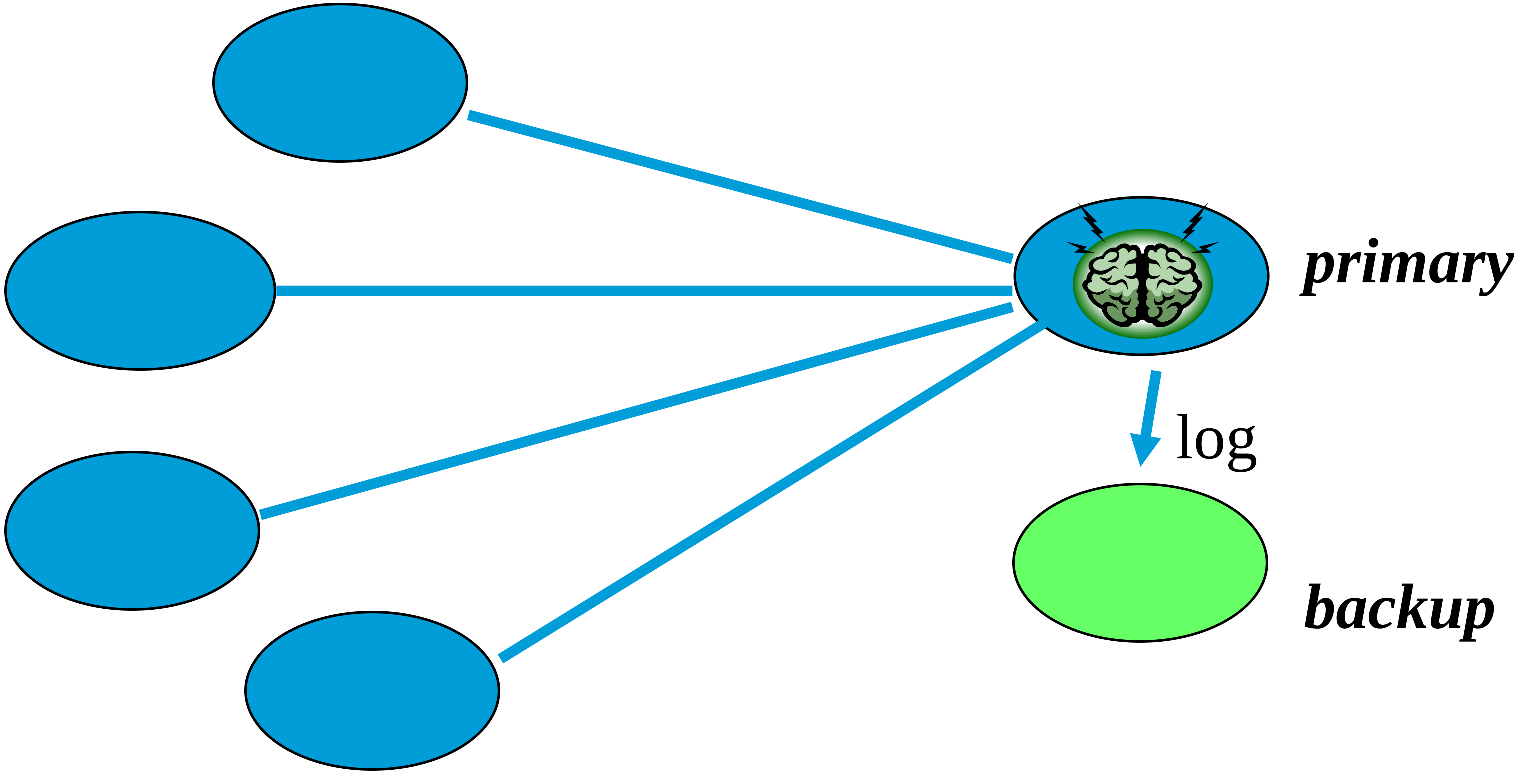
  ATC is a "cloud" but one with special needs: it must speak with "one voice"

- ATC → needs *highly available* servers.
  - Else a crash could leave controller unable to make decisions
  - How can we make a service highly available?

Server Replication

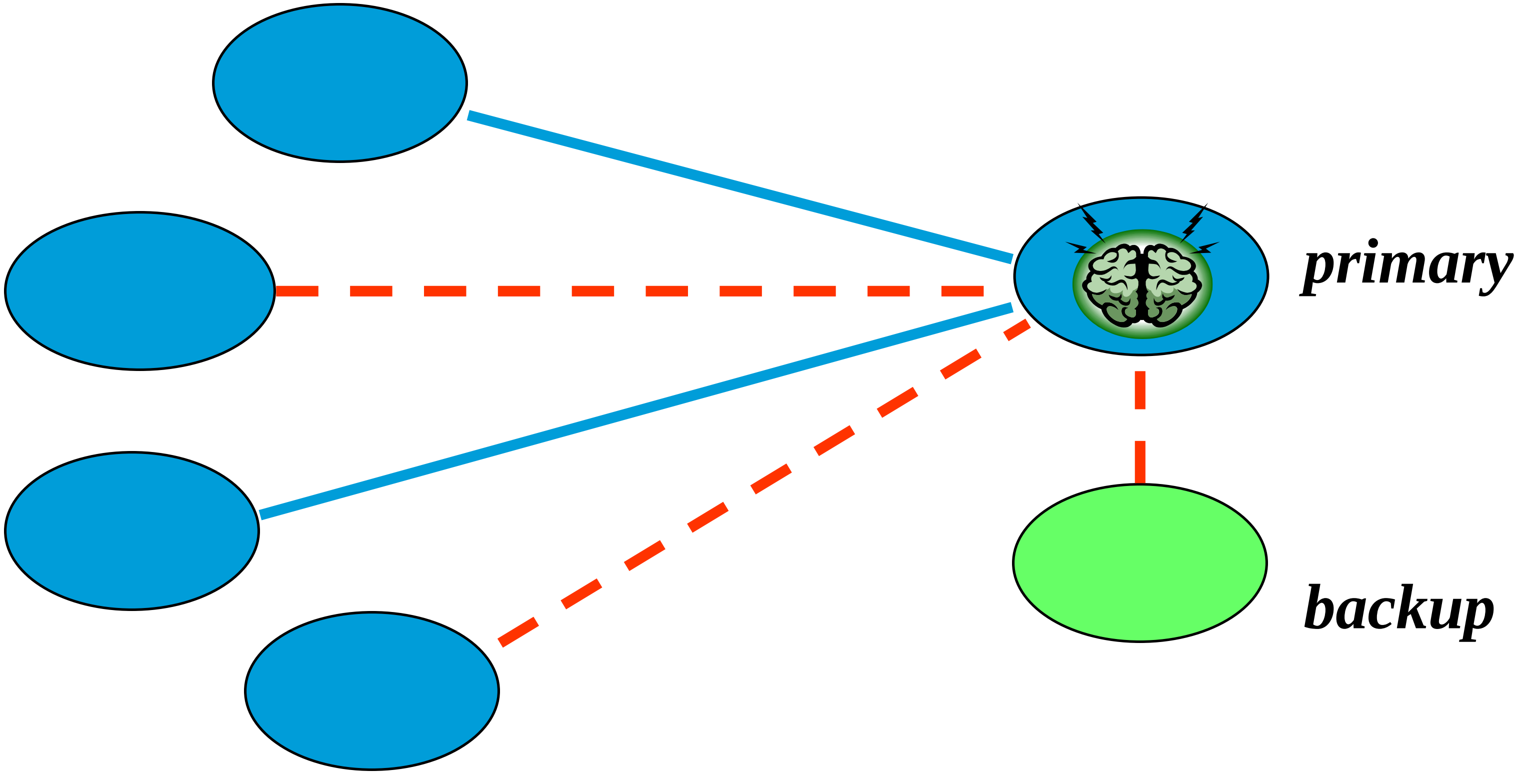
- Key issue: we need to maintain that “one voice” property
  - highly available service needs to be indistinguishable from that of a traditional service running on some single node that just doesn’t fail
- Most obvious option: “primary/backup”
  - We run two servers on separate platforms
  - The primary sends a log to the backup
  - If primary crashes, the backup soon catches up and can take over

A primary-backup scenario…



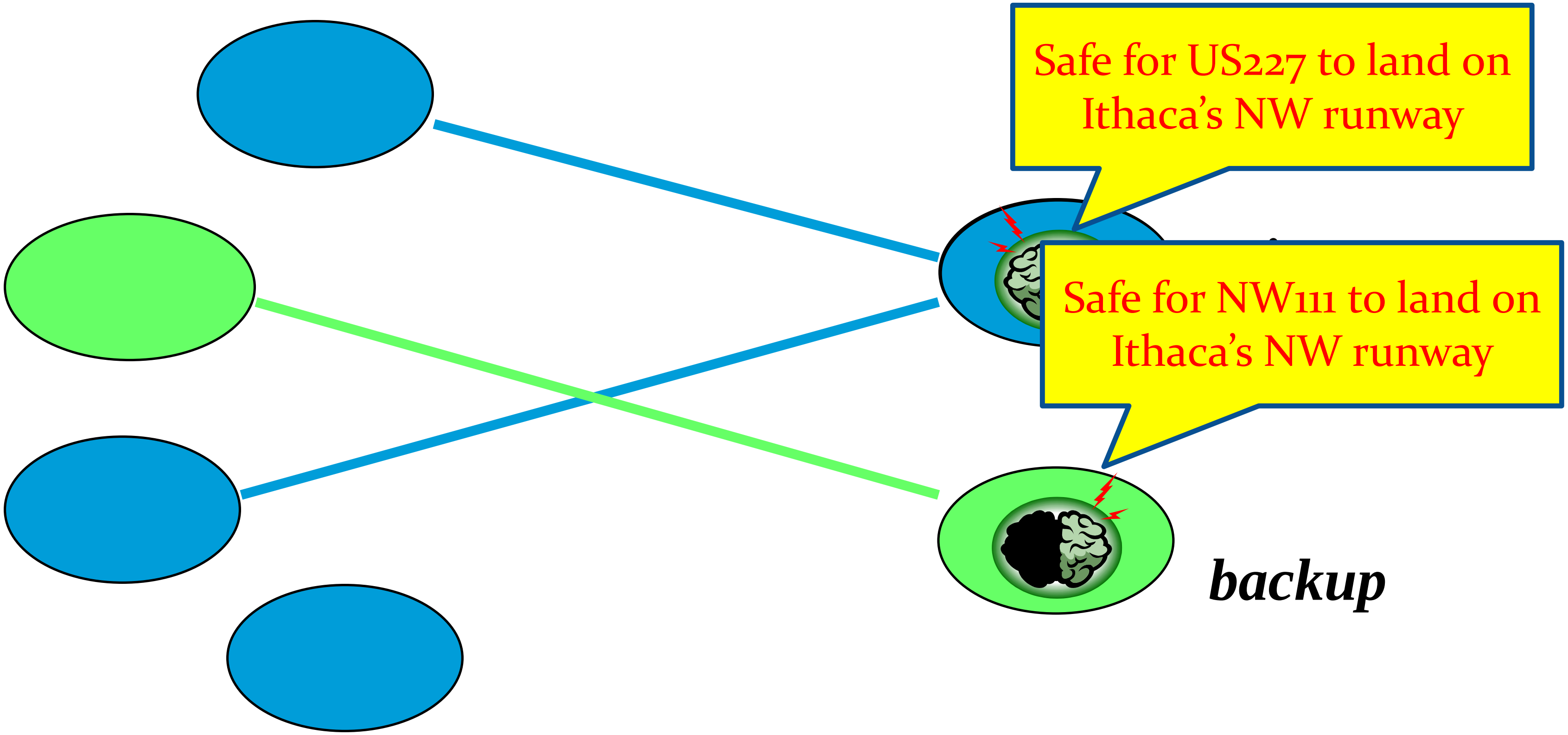
Clients initially connected to primary, which keeps backup up to date. Backup collects the log

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

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.

Solution to override → a transient fault

- 1. Build a fairly complex network with some routers, multiple network segments, etc

- (i) Run TCP over it in the standard way

- (ii) 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.

As a Result

- 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 "no"… in response to queries!

Can we fix this problem?

- Solution → have the backup unplug the primary
- But less difficult solutions are also possible

Problem: 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!

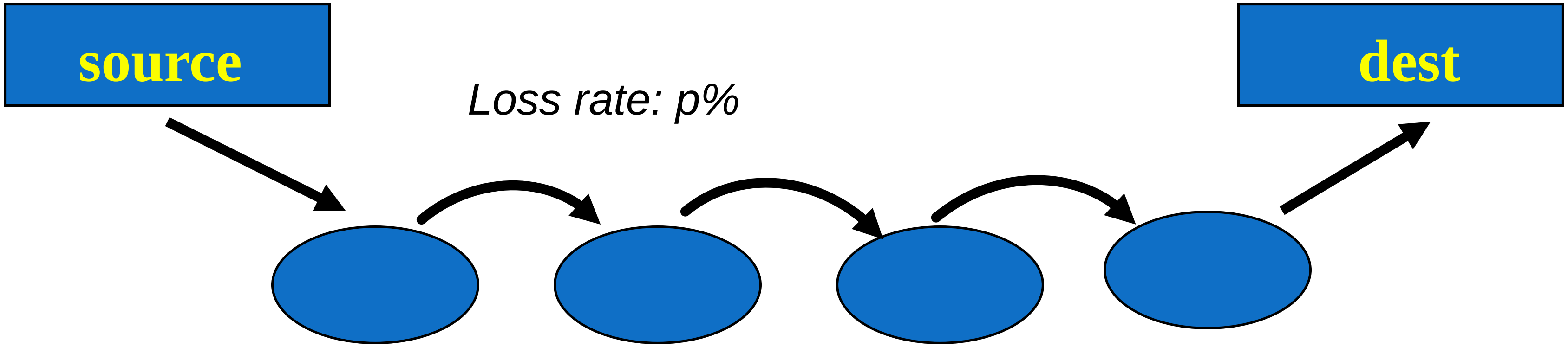
End-to-End argument

- 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 packet 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 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
  - In this example,

    TCP handles errors in ad-hoc, inconsistent ways

- Developers often forced to step outside of the box… and may succeed,

Even this case illustrates choice

- We have many options, if we are willing to change the failure semantics of our platform
  - 1. 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
  - 2. Give backup a way to physically "kill" the primary, e.g. unplug it
    - If backup takes over… primary shuts down
  - 3. 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 → the last of options is good for LANs

Elements of cloud computing

- One :
- Large list of tools and technologies earlier OHP

- A second: a collection of abstractions and assumptions that the cloud needs to implement, and that the developer can then “trust”
  - Example, if the cloud were to implement a failure detection mechanism, the developer could trust, and split brain problems would be avoided
  - Way of thinking in research → *The cloud is a provider of abstractions.* Specific tools implement those abstractions

ATC example…

- 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

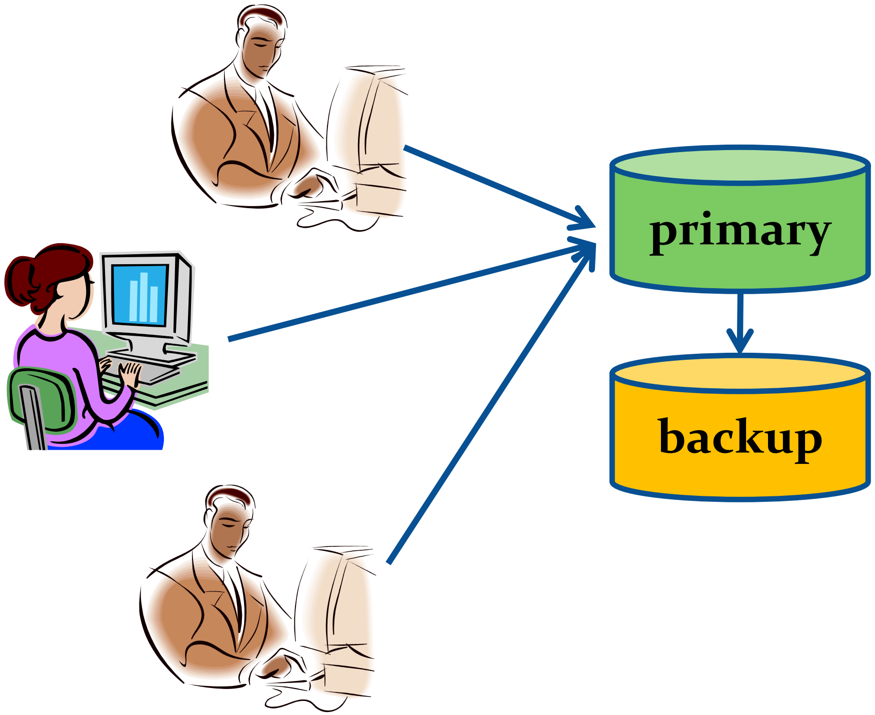
How can a system track its own membership?

- We’ve discussed the idea that a cloud might offer users some form of virtual 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

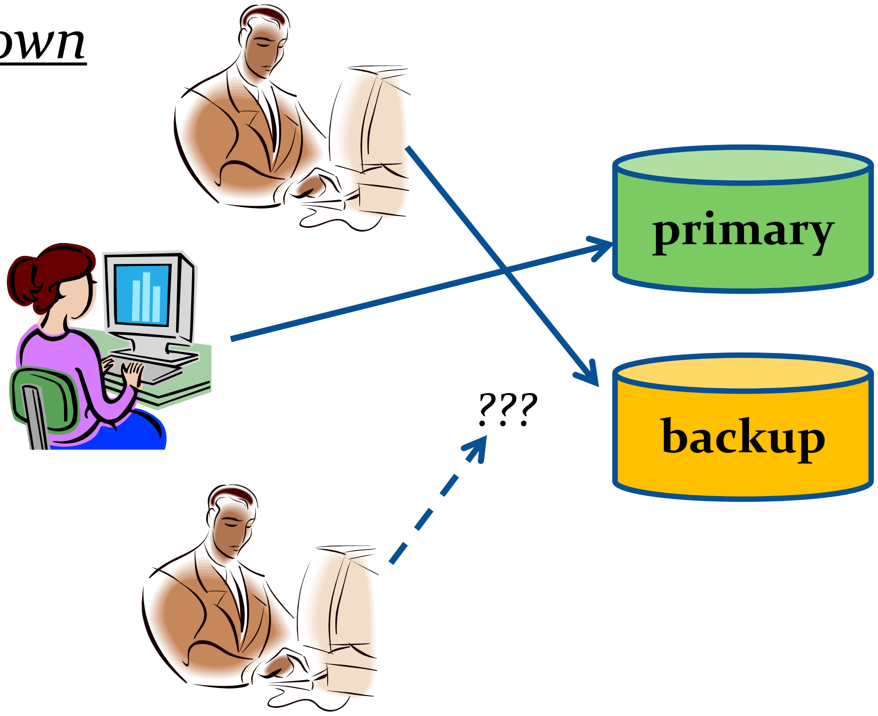
- Each channel has its own timers
- If a timeout occurs, clients "fail-over" to the backup



Split brain scenario

Potential for inconsistency

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



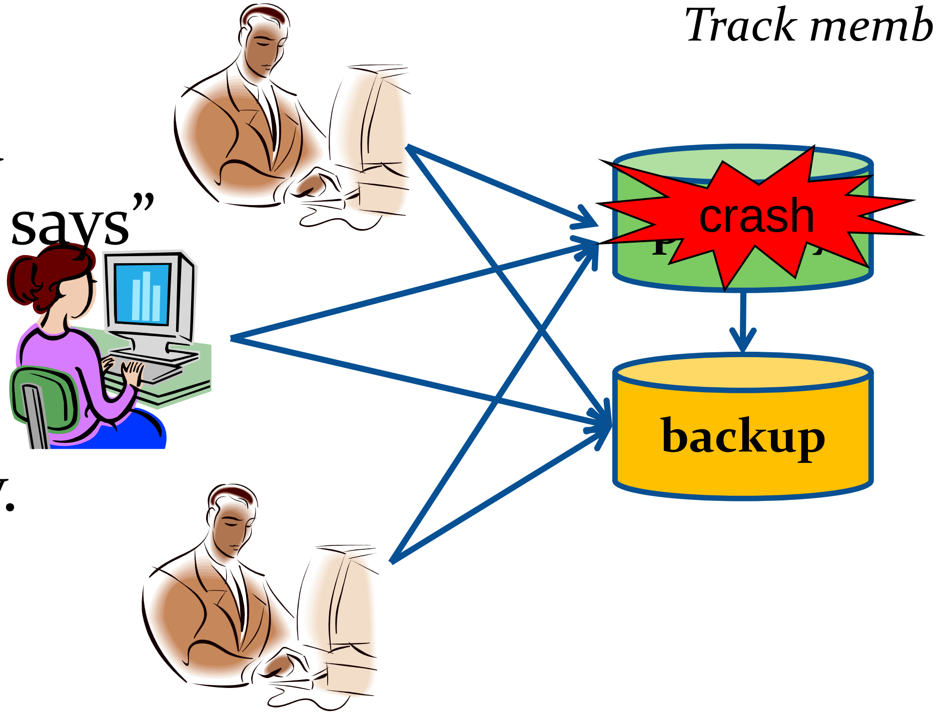
Role of a leader

Hear and obey. The primary is down. !!!

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 → the leader

- 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

- 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.

Back to the edge…

- Issues → in cloud settings.

- 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

  - P2P replication… event notification… building structured overlays… well known applications ← different

Self study work 1

- Please read article and see short demo…,

  write a brief summary – what do you find ?

- 1. Internet Computing, IEEE, Date of Publication: Nov.-Dec. 2007, Volume: 11 , Issue: 6,

  Page(s): 72 – 78
- Live Distributed Objects: Enabling the Active Web
- Author(s): Ostrowski, K. ,Cornell Univ., Birman, K. ; Dolev, D.

2. See live demo :

See short demo…

http://liveobjects.cs.cornell.edu

Self Study work 2

- Programming with Live Distributed Objects
- Authors: Krzysztof Ostrowski Cornell University, Ken Birman Cornell University, Danny Dolev The Hebrew University of Jerusalem, Jong Hoon Ahnn Cornell University, Published: Proceeding ECOOP '08 Proceedings of the 22nd European conference on Object-Oriented Programming Pages 463 - 489 , 2008.