M120: DISTRIBUTED SYSTEMS

Fundamentals

*Slides are variant of slides provided by Ken Birman

Overview of Lecture

- Fundamentals: terminology and components of a reliable distributed computing system
- Communication technologies and their properties
- Basic communication services
- Internet protocols
- End-to-end argument

Some terminology

- A program is the code you type in
- A process is what you get when you run it
- A message is used to communicate between processes. Arbitrary size.
- A packet is a fragment of a message that might travel on the wire. Variable size but limited, usually to 1400 bytes or less.
- A protocol is an algorithm by which processes cooperate to do something using message exchanges.

More terminology

- A network is the infrastructure that links the computers, workstations, terminals, servers, etc.
  - It consists of routers
  - They are connected by communication links
- A network application is one that fetches needed data from servers over the network
- A distributed system is a more complex application designed to run on a network. Such a system typically has multiple processes that cooperate to do something. Remember our working definition:

> *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.*

Protocol defines

- Types of messages exchanged,
  - e.g., request, response
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

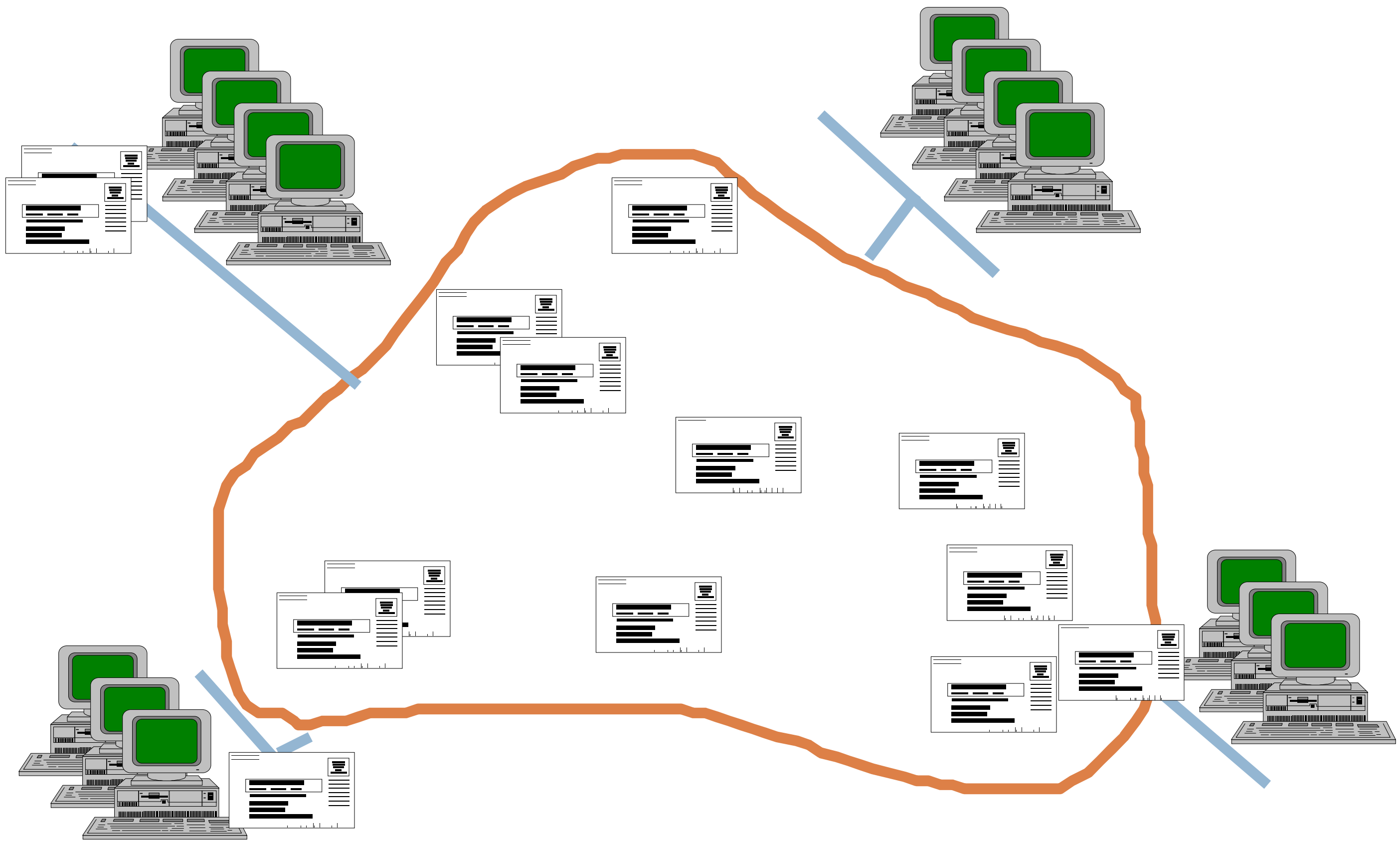
Public-domain protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:

- e.g., KaZaA, Skype

A network is like a "mostly reliable" post office



Why isn't it totally reliable?

- Links can corrupt messages
  - Rare in the high quality links on the Internet "backbone"
  - More common with wireless connections, cable modems, ADSL
- Routers can get overloaded
  - When this happens they drop messages
    - this is very common
- But protocols that retransmit lost packets can increase reliability

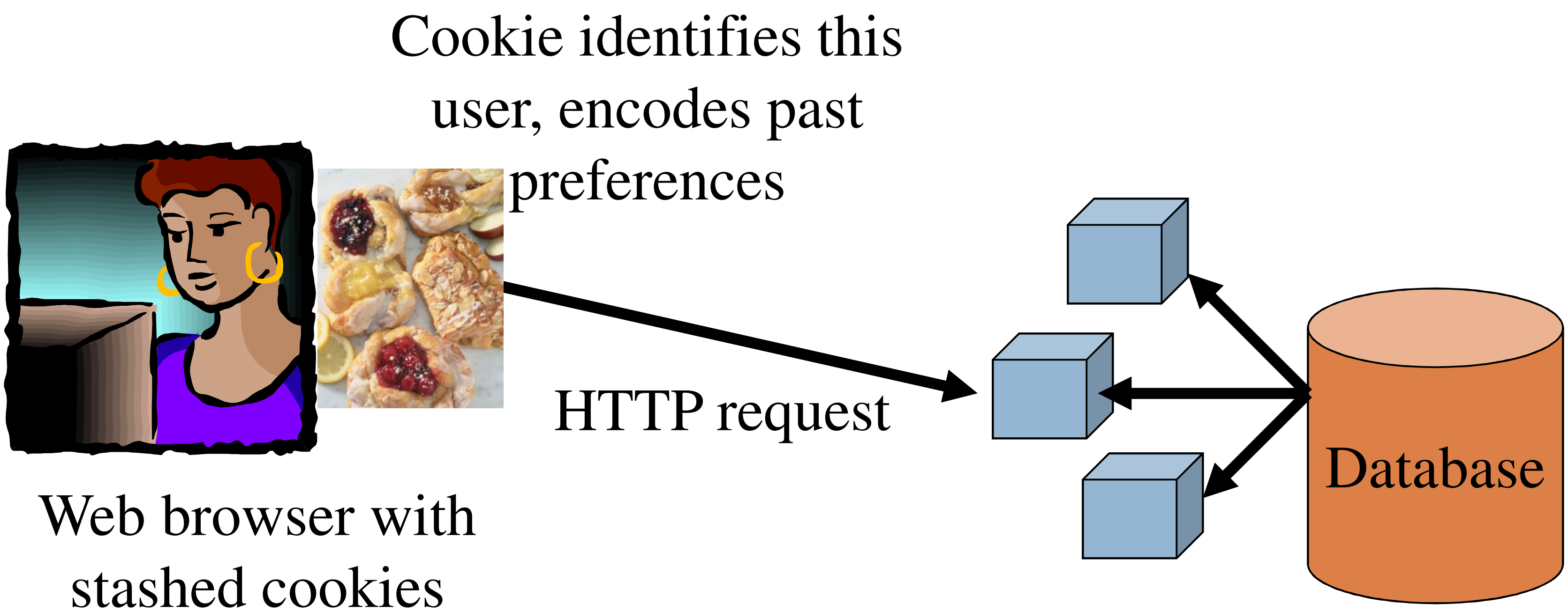
How do distributed systems differ from network applications?

- Distributed systems may have many components but are often designed to mimic a single, non-distributed process running at a single place.
- "State" is spread around in a distributed system
  - Networked application is free-standing and centered around the user or computer where it runs. (E.g. "*web browser*".)
  - Distributed system is spread out, decentralized. (E.g. "*air traffic control system*")

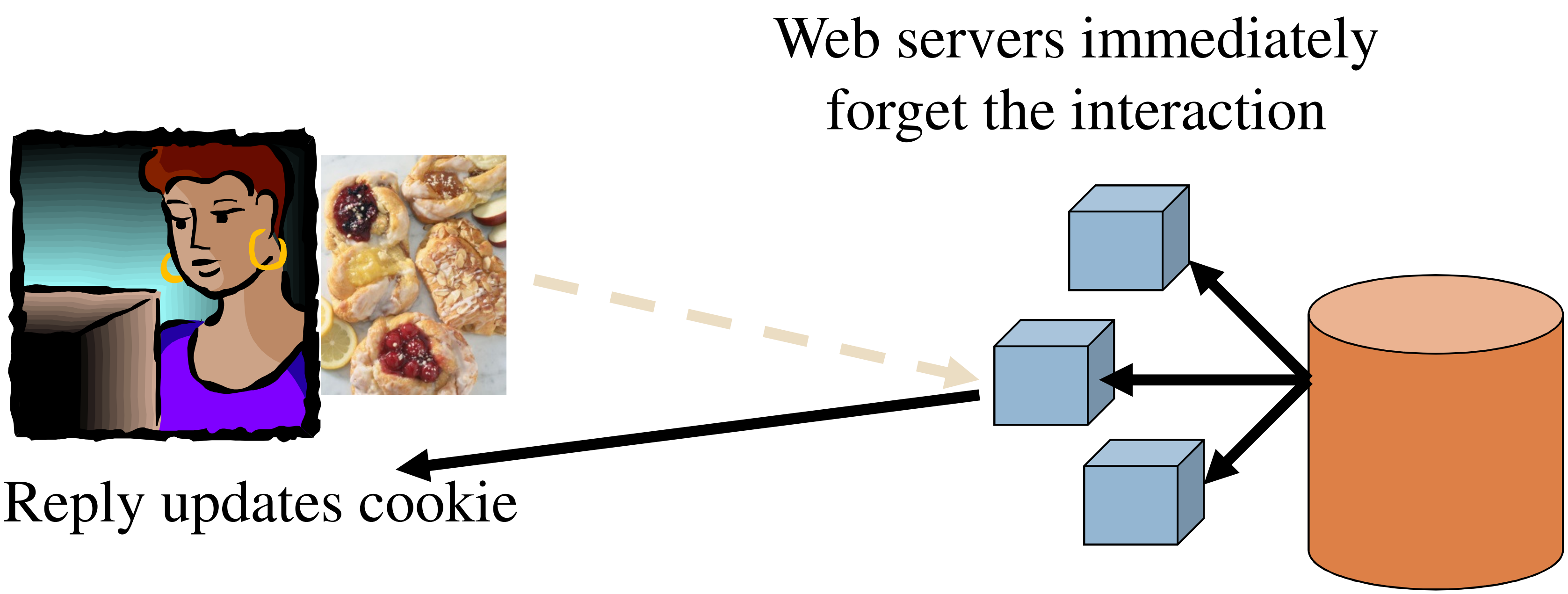
What about the Web?

- Browsers are independent: a browser fetches data you request.
- Web servers don't keep track of who is using them. Each request is self-contained and treated independently of all others.
  - Cookies don't count: they sit on your machine
  - And the database of account info doesn't count either… this is "ancient" history, nothing recent
- … So the web has two network applications that talk to each other
  - The browser on your machine
  - The web server it happens to connect with… which has a database "behind" it

What about the Web?

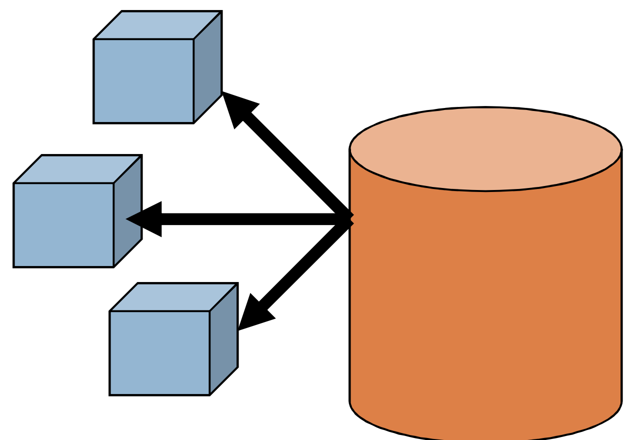


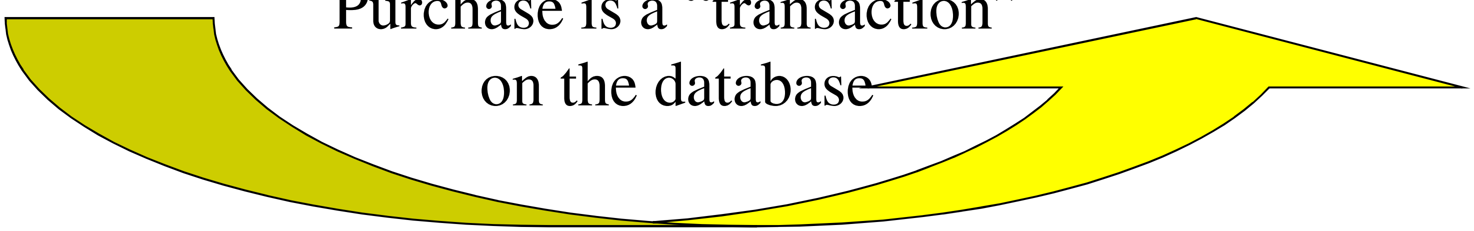
What about the Web?



What about the Web?

Web servers have no memory of the interaction





Purchase is a "transaction" on the database

What about the Web?

- But… the data center that serves your request may be a complex distributed system
  - Many servers and perhaps multiple physical sites
  - Opinions about which clients should talk to which servers
  - Data replicated for load balancing and high availability
  - Complex security and administration policies
- So: we have a "networked application" talking to a "distributed system"

Is the Web "reliable"?

- We want to build distributed systems that can be relied upon to do the correct thing and to provide services according to user expectations
- Not all systems need reliability (e.g., the "old" Web)
  - If a web site doesn't respond, you just try again later
  - If you end up with two wheels of brie, not a big problem!
- Reliability is a growing requirement in critical settings but these remain a small percentage of the overall market for networked computers
  - Reliability requires satisfying multiple properties (next slide).

Reliability is a broad term

- Fault-Tolerance: remains correct despite failures
- High or continuous availability: resumes service quickly after failures, doesn't wait for repairs
- Performance: provides desired responsiveness
- Recoverability: can restart failed components
- Consistency/Correctness: coordinates actions by multiple components, so they mimic a single one
- Security: authenticates access to data, services
- Privacy: protects identity and locations of users

“Failure” also has many meanings

- Halting failures: component simply stops
- Fail-stop: halting failures with notifications
- Omission failures: failure to send/recv. message
- Network failures: network link breaks
- Network partition: network fragments into two or more disjoint subnetworks
- Timing failures: action early/late; clock fails, etc.
- Byzantine failures: arbitrary malicious behavior
  - Most difficult of all to deal with

Examples of failures

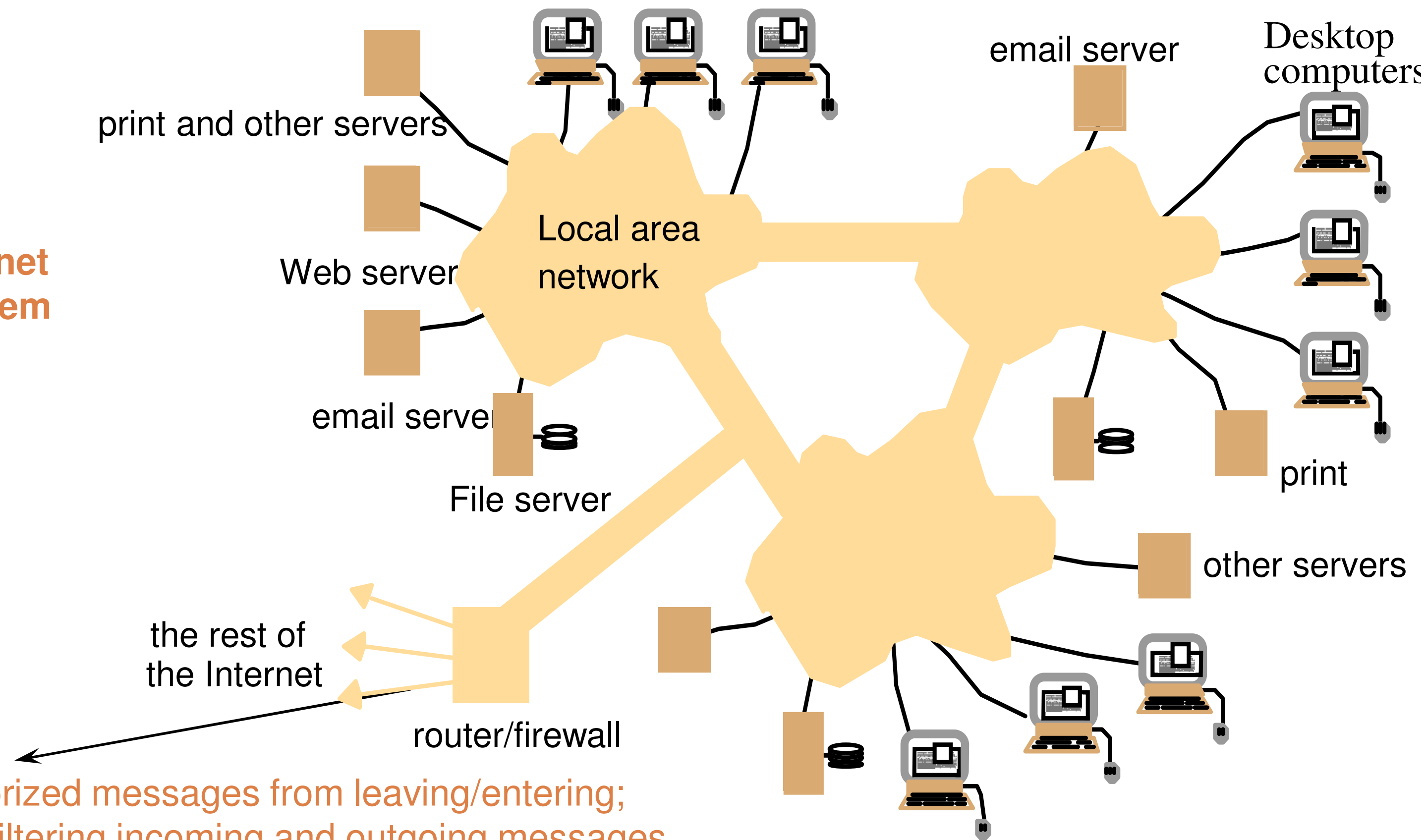
- My PC suddenly freezes up while running a text processing program. No damage is done. This is a halting failure.
- A network file server tells its clients that it is about to shut down, then goes offline. This is a fail-stop failure. (The notification can be trusted.)
- An intruder hacks the network and replaces some parts with fakes. This is a Byzantine failure.

The Internet – Quick Refresher

- Underlies many distributed systems.
- A vast interconnected collection of computer networks of many types.
- Intranets – subnetworks operated by companies and organizations.
- Intranets contain LANs (local area networks).
- WAN – wide area networks, consists of subnets (intranets, LANs, etc.)
- ISPs – Internet Service Providers. Companies that provide modem links and other types of connections to users.
- Intranets (actually the ISPs' core routers) are linked by backbones – network links of large bandwidth, such as satellite connections, fiber optic cables, and other high-bandwidth circuits.
- UC2B? Google Fiber? (MAN = Metropolitan Area Networks)

An Intranet & a distributed system

Running over this Intranet is a distributed file system



prevents unauthorized messages from leaving/entering; implemented by filtering incoming and outgoing messages via firewall "rules" (configurable)

Internet Layering Model

Internet Protocol Stack

Layer 5	Application
Layer 4	Transport
Layer 3	Network
Layer 2	Link
Layer 1	Physical

- Each layer uses the function of the layer below
- Each layer exports functionality to layer above
- This layering of protocol behavior called a "protocol stack"
- Aka, the "TCP/IP stack"

Internet protocol suite

- Can be understood in terms of Internet Layering Model
- Defines "addressing" standard, basic network layer (IP packets, limited to 1400 bytes), and session protocols (TCP, UDP, UDP-multicast)
  - For example, TCP is a "session" protocol
- Includes standard "domain name service" that maps host names to IP addresses
- DNS itself is tree-structured and caches data
- (See two slide sets on class web page for more detailed refreshers on networking basics and DNS)

Major internet protocols

- TCP, UDP, FTP, Telnet
- Email: Simple Mail Transfer Protocol (SMTP)
- News: Network News Transfer Protocol (NNTP)
- DNS: Domain name service protocol
- NIS: Network information service (a.k.a. "YP")
- LDAP: Protocol for talking to the management information database (MIB) on a computer
- NFS: Network file system protocol for UNIX
- X11: X-server display protocol
- Web: HyperText Transfer Protocol (HTTP), and SSL (one of the widely used security protocols)

Networking Stacks

Distributed System Protocols!

Networking Protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	[illegible]
file transfer	ftp [RFC 959]	[illegible]
streaming multimedia	proprietary (e.g. RealNetworks)	[illegible]
remote file server	NFS	TCP or UDP
internet telephony	proprietary (e.g., Skype)	typically UDP

TCP=Transmission Control Protocol
UDP=User Datagram Protocol

(Implemented via sockets)

Typical network hardware options

- Ethernet: 10Mbit CSMA technology, limited to 1400 byte packets. Uses single coax cable.
- FDDI: twisted pair, self-repairing if cable breaks (100Mbit)
- Bridged Ethernet: common in big LAN's, ring with multiple ethernet segments
- Fast Ethernet: 100Mbit version of ethernet. Since 1998, we have seen 1 Gbit Ethernet, 10Gbit, 100Gbit…
- ATM: switching technology for fiber optic paths. Can run at 155Mbits/second or more. Very reliable, but mostly used in telephone systems.

Implications for reliability?

- Protocol designers have problems predicting the properties of local-area networks
- Latencies and throughput may vary widely even in a single installation
- Hardware properties differ widely; often, must assume the least-common-denominator
- Packet loss a minor problem in hardware itself

Hardware evolution

- Over the last two decades:
  - storage capacity has improved by 10,000x
  - CPU speeds by 1,000x
  - core counts by 50x
  - network speeds by 10,000x
  - and I/O latency by 1,000x

Trends: Users

- Then and Now

  Biologists:

  - 1990: were running small single-molecule simulations
  - Today: CERN's Large Hadron Collider producing many PB/year

Trends: Technology

- Doubling Periods – storage: 12 mos, bandwidth: 9 mos, and (what law is this?) cpu compute capacity: 18 mos
- Then and Now
  - Bandwidth
    - 1985: mostly 56Kbps links nationwide
    - 2015: Tbps links widespread
  - Disk capacity
    - Today's PCs have TBs, far more than a 1990 supercomputer

Impact of technology trends

- A discontinuity is currently occurring in communication speeds
  - LANs getting super fast
- Disks (HDDs) have “maxed out” and hence are looking slower and slower and SSDs still expensive for widespread use
- Memory is cheaper (DRAM, NVRAM)
  - Avoid disk for critical path; leave disk for persistence and storage of entire data set
- Memory of remote computers looks “closer and closer” (RDMA)
  - Shift from disk storage towards more use of access to remote objects “over the network”
  - Affects application/system design (e.g., distributed in-memory key value store, distributed file system)
- O/S imposed communication latencies has risen in relative terms over past decade

Reliability versus Performance

- Some think that more reliable means "slower"
  - Indeed, it usually costs time to overcome failure
  - For example, if a packet is lost probably need to resend it, and may need to solicit the retransmission
- But for many applications, performance is a big part of the application itself: too slow means "not reliable" for these!
- Reliable systems thus must look for highest possible performance
- ... but unlike unreliable systems, they can't cut corners in ways that make them faster but flaky

Moving up (the stack) to the Application

- Internet protocol stack stops at the application layer
  - Assumes applications know about one another (ie., can find each other)
  - Client looks up the server… connects… sends a request…response comes back
- But how did the client know which server it wanted?

Discovery Problem

- Consider the problem of *discovering* the right server to connect with
  - Suppose your computer needs to find the Amazon server that will sell you Disney film DVDs (European format)

Why is discovery hard?

- Boston client has opinions
  - You can only play European format, so your search is partly controlled by client goals
- Service has opinions
  - Amazon might have data centers in Europe and in the US and may want your request to go to a particular one
- Once we find the server name we need to map it to an IP address
- And the Internet itself has routing "opinions" too

So… four layers of discovery

- Potentially, we might want to customize each one of these layers to get a given application functionality to work
- The Internet protocol stack doesn’t include any of these layers, so this is an example of a situation where we need much more

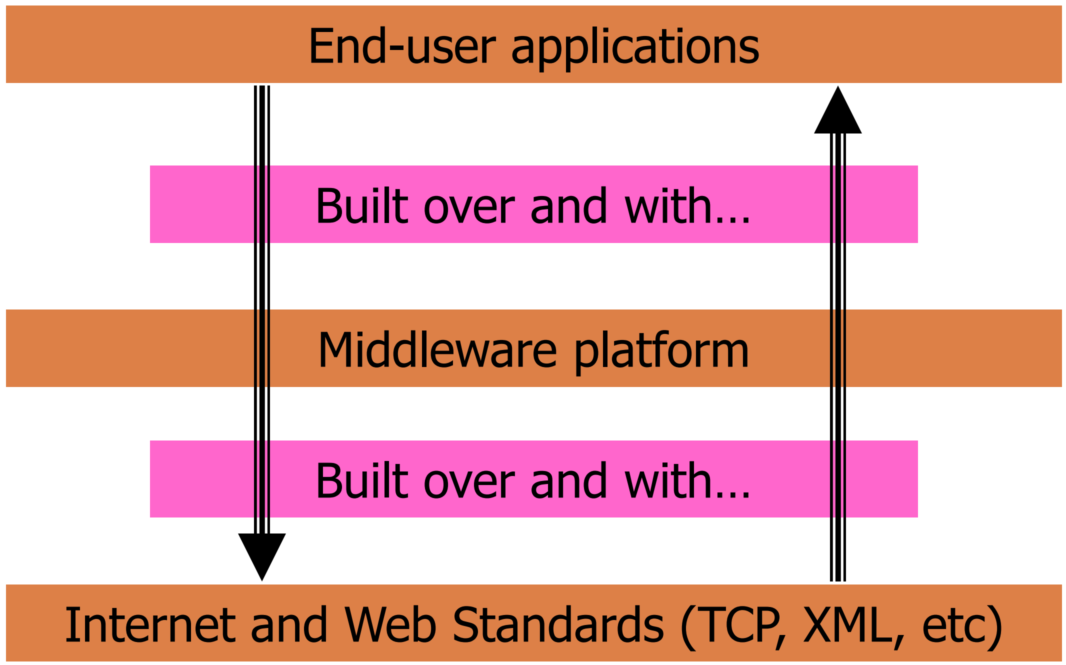
Other things we might need, NOT provided by the stack

- Standard ways to handle
  - Reliability, in all the senses we listed
  - Life cycle management of the service
    - Automated startup of services, if someone asks for one and it isn't running; backup; etc…
    - Automated migration and load-balancing, monitoring, parameter adaptation, self-diagnosis and repair…
  - Tools for integrating legacy applications with new, modern ones

Concept of a middleware platform

- Large software systems that automate many aspects of application management and development
  - CORBA – by now a stable and outmoded platform focused on “objects” (has lost steam)
  - SOAP
  - REST
  - Web Services – service oriented architecture → AWS, microservices, etc.

Layers: Modern perspective



Example

- Imagine a banking system with many programs, one at each branch
- And suppose that only some can talk to others due to firewalls and other restrictions
  - E.g. A can talk to B and B can talk to C, but A can't talk to C

How to handle this?

- In the distant past, people cooked up all sorts of weird hacks
- Today, a standard approach is to build a routing layer
  - Inside the application, it would automatically forward messages towards their destinations
  - Thus A can talk to C (via B)

Once we have this…

- Now we can split our brains, in a good way:
  - Above this routing layer, we write code *as if* routing from anyone to anyone was automatic
  - Inside the routing layer, we implement this functionality
  - Below the routing layer we just do point-to-point messaging where the bank permits it and we never end up trying to send messages over links not available to us

This layering looks elegant!

- It lets us focus attention on issues in one place and simplifies code as a result
- Also helpful when debugging…

- *Platform architectures simply take the same approach further*

Using a platform

- Distributed application developers have often used
  - Java/J2EE: An outgrowth from CORBA which is closely integrated with developer tools and very easy to use
  - Microsoft C# (or C++) on .NET in Visual Studio: similar in concept but focused more on Web Services
  - Software as a Service: cloud-based services
- Often just using their editor and clicking "build and run" is enough to use the service framework!
  - But you inherit its power… and limits…
  - E.g., the reliability model in Web Services doesn't automate data replication