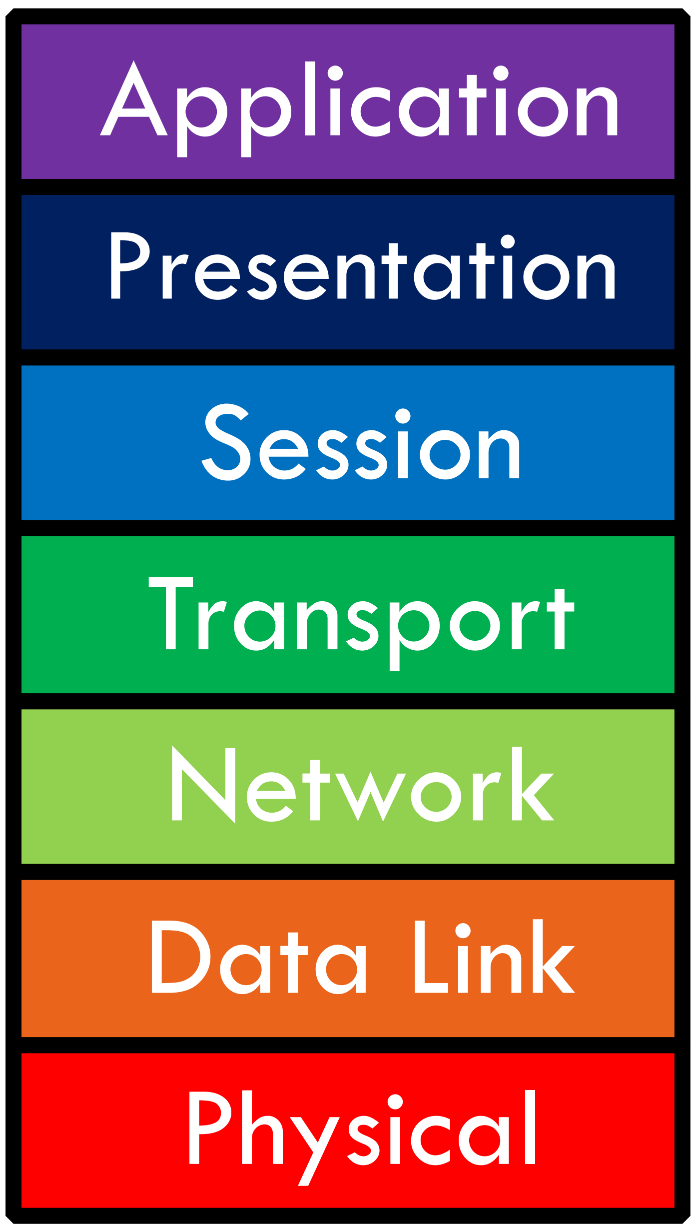
CS 3700
Networks and Distributed Systems

Lecture 13: Distributed Systems

(Based off slides by Rik Sarkar at University of Edinburgh)

Application Layer



- Function:
  - Implement application using network
- Key challenges:
  - Scalability
  - Fault Tolerance
  - Reliability
  - Security
  - Privacy
  - …

What are distributed systems?

- From Wikipedia:

  A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages.

- Essentially, multiple computers working together
  - Computers are connected by a network
  - Exchange information (messages)

- System has a common goal

Definitions

- No widely-accepted definition, but…
- Distributed systems comprised of *hosts* or *nodes* where
  - Each node has its own local memory
  - Hosts connected via a network
- Originally, requirement was *physical distribution*
  - Today, distributed systems can be on same host
  - E.g., VMs on a single host, processes on same machine

Networks vs. Distributed Systems

- Definition similar to definition of a network
  - **Distributed system:** A program (or set of programs) that *use a network* to accomplish a goal
  - **Network:** A system for sending messages (information) between hosts
- Thus, distributed system uses a network
  - Doesn't care about network's implementation
  - But must deal with network's (lack of) guarantees
  - Also, network's naming conventions, etc

Outline

- (Brief) History of distributed systems
- Examples of distributed systems
- Fundamental challenges

History

- Distributed systems developed in conjunction with networks

- Early applications:
  - Remote procedure calls (RPC)
  - Remote access (login, telnet)
  - Human-level messaging (email)
  - Bulletin boards (Usenet)

Early example: Sabre

- Sabre was the earliest airline Global Distribution System
  - The system that they use at the airports

```
2UA388«
UA RESPONSE
0388/19AUG
F YYC/ETD     823A   ON TIME
F DEN/ETA    1046A   L00.01   ETD    1145A  ON TIME
F SAN/ETA     104P   ON TIME

SKED   YYC   ORIG     823A              GTD    21 SHIP 4214
       DEN   1045A   1145A      GTA   B46 GTD **** SHIP 4636
       SAN    104P   TERM       GTA    44
I«
IGD
QC/«
             ON QUEUE AS OF 0643 ON 19AUG FOR BH4C
  G ......3
  S ......7
TOTAL  MESSAGES .......10  SPECIALS ........0  PNRS ........0
_
```

Sabre

- American had a central office with cards for each flight
  - Agent calls in, worker would mark seat sold on card

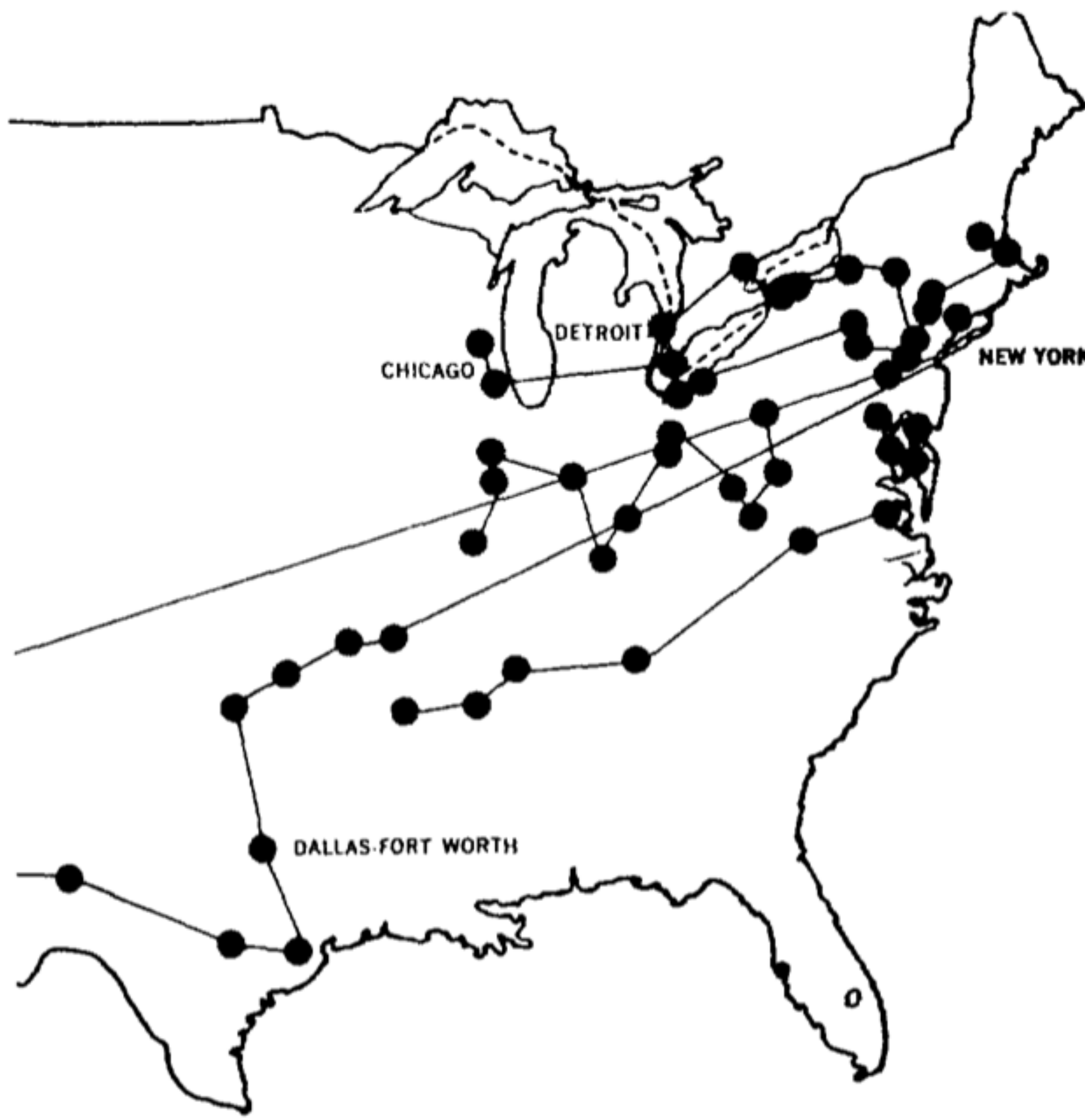
- Built a computerized version of the cards
  - Disk (drum) with each memory location representing number of seats sold on a flight
  - Built network connecting various agencies
  - Distributed terminals to agencies

- Effect: Removed human from the loop

Sabre network

RESERVATIONS PROCESSING SYSTEM

computers speed air travel reservations…



In addition to handling the passenger's reservation, this new IBM system also:

Answers requests for space from other airlines

Advises agents to remind passengers to pick up tickets.

Maintains and processes passengers waiting lists for fully-booked flights.

Supplies fare quotations.

Supplies information on arrival and departure times.

Reminds agents to advise scheduled passengers of any flight changes.

Move towards microcomputers

- In the 1980s, personal computers became popular
  - Moved away from existing mainframes
- Required development of many distributed systems
  - Email
  - Web
  - DNS
  - …
- Scale of networks grew quickly, Internet came to dominate

Today

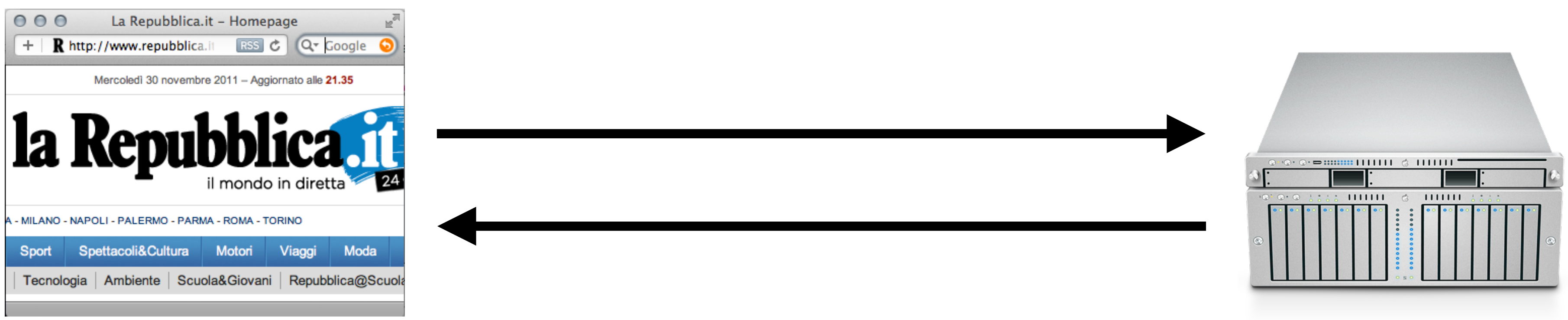
- Growth of pervasive and mobile computing
  - End users connect via a variety of devices, networks
  - More challenging to build systems

- Popularity of “cloud computing”
  - Essentially, can purchase computation as a commodity
  - Many startups don’t own their servers
    - All data stored in the cloud
  - How do we build secure, reliable systems?

Outline

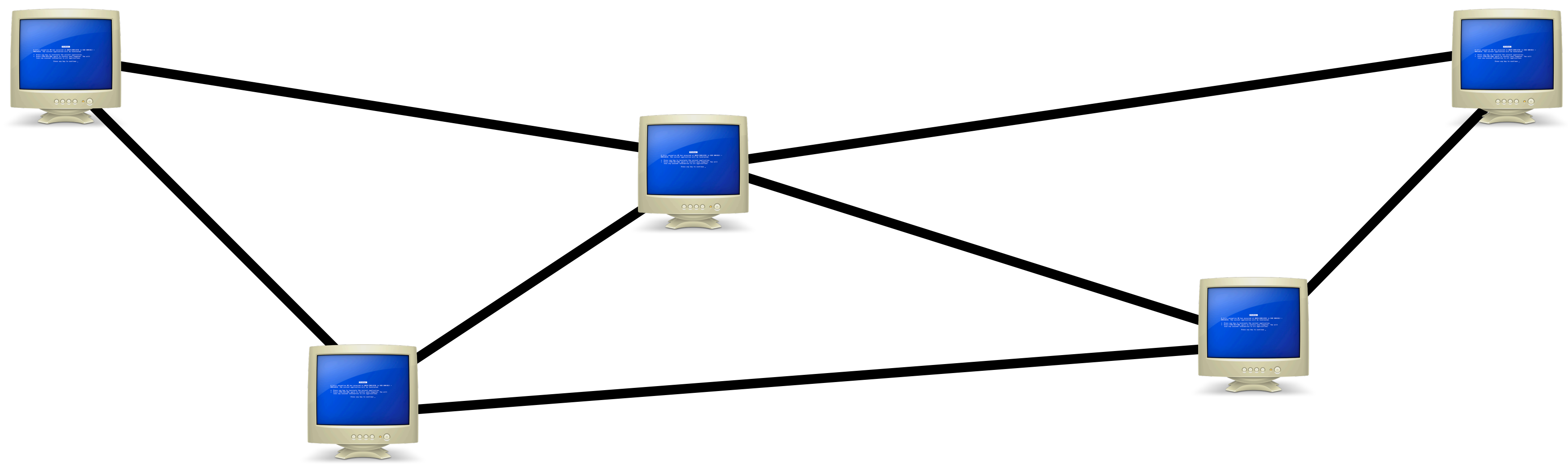
- (Brief) History of distributed systems
- Examples of distributed systems
- Fundamental challenges

Example 1: Web systems



- Web is a widely popular distributed system
- Has two types of entities:
  - **Web browsers:** Clients that render web pages
  - **Web servers:** Machines that send data to clients
- All communication over HTTP

Example 2: Bittorrent



- Popular platform for large content distribution
- All clients "equal"
  - Collaboratively download data
  - Use custom protocol to download
- Robust if any client fails (or is removed)

Example 3: Stock market

- Large distributed system
  - Many players
  - Economic interests *not* aligned
- All transactions must be executed in-order
  - E.g., Facebook IPO
- Transmission delay is a huge concern
  - Hedge funds will buy up rack space closer to datacenter
  - Can arbitrage millisecond differences in delay

Outline

- (Brief) History of distributed systems
- Examples of distributed systems
- Fundamental challenges
- Design decisions

Challenge 1: Global knowledge

- *No host has global knowledge*
- Need to use network to exchange state information
  - Network capacity is limited; can't send everything
- Information may be incorrect, out of date, etc
  - New information takes time to propagate
  - Other things may happen in the meantime
- Fundamental challenge
  - How do detect and address inconsistencies?

Challenge 2: Time

- Time cannot be measured perfectly
  - Hosts have different clocks, skew
  - Network can delay/duplicate messages
- How to determine what happened first?
  - In a game, which player shot first?
  - In a GDS, who bought the last seat on the plane?
- Need to have a more nuanced abstraction of time

Challenge 3: Failures

- A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable. — Leslie Lamport
- Failure is the common case
  - As systems get more complex, failure more likely
  - Must design systems to tolerate failure
- E.g., in Web systems, what if server fails?
  - System need to detect failure, recover

Challenge 4: Scalability

- Systems tend to grow over time
  - How to handle future users, hosts, networks, etc?

- E.g., in a multiplayer game, each user needs to send location to all other users
  - $O(n^2)$ message complexity
  - Will quickly overwhelm real networks
  - Can reduce frequency of updates (with implications)
  - Or, choose nodes who should update each other

Challenge 5: Security

- Distributed systems often have many different entities
  - Often not mutually trusting (e.g., stock market)
  - Economic incentives for abuse

- Systems often need to provide
  - Confidentiality (only intended parties can read)
  - Integrity (messages are authentic)
  - Availability (system cannot be brought down)

Challenge 6: Openness

- Can system be extended/reimplemented?
  - I.e., can I develop a new client?

- Requires specification of system/protocol published
  - Often requires standards body (IETF, etc) to agree
  - Cumbersome process, takes years
    - Many corporations simply publish own APIs

- IETF works off of RFC (*request for comment*)
  - Anyone can publish, propose new protocol

Challenge 7: Concurrency

- Large, complex systems exist in many places:
  - E.g., Web sites replicated across many machines
- Often will have concurrent operations on a single object
  - How to ensure object is in *consistent* state?
  - E.g., bank account: How to ensure I can't overdraw?
- Solutions fall into many camps:
  - Serialization: Make operations happen in defined order
  - Transactions: Detect conflicts, abort
  - Append-only structures: Deal with conflicts later
  - ....

Outline

- (Brief) History of distributed systems
- Examples of distributed systems
- Fundamental challenges
- Design decisions

Distributed system architecture

- Two primary architectures:
  - **Client-server:** System divided into clients (often limited in power, scope, etc) and servers (often more powerful, with more system visibility. Clients send requests to servers.
  - **Peer-to-peer:** All hosts are "equal", or, hosts act as both clients and servers. Peers send requests to each other. More complicated to design, but with potentially higher resilience.

Messaging interface

- Messaging is fundamentally *asynchronous*
  - Client asks network to deliver message
  - Waits for a response

- What should the programmer see?
  - **Synchronous interface:** Thread is "blocked" until a message comes back. Easier to reason about
  - **Asynchronous interface:** Control returns immediately, response may come later. Programmer has to remember all outstanding requests. Potentially higher performance.

Naming

- Need to be able to refer to hosts/processes
- Naming decisions should reflect system organization
  - E.g., with different entities, hierarchal system may be appropriate (entities name their own hosts)
- Naming must also consider
  - Mobility: hosts may change locations
  - Security: how do hosts prove who they are?
  - Scalability: how many hosts can a naming system support?

Rest of the semester

- Will explore a few distributed system basics
  - Handling failures
  - Time/clocks
  - Remote procedure calls
  - Security

- But, most time spent exploring real system
  - Essentially, “case studies”
  - Will explore Web, BitTorrent, Bitcoin in depth
  - Different points in design space, address problems differently