Distributed Systems

To do …

- Distributed systems
- This class
- Models and architectures
- *Next:* A brief introduction to Go

Image from http://www.vs.inf.ethz.ch/about/zeit.jpg

Welcome!

- What is this all about?
  - A course on distributed systems
  - With an experimental systems approach
  - Focused on large-scale issues

- Our goals
  - Learn about DS – principles & current work
  - Learn Go – a some-what new programming language
  - Build some interesting system
  - Learn/practice reading research papers

What is a *distributed system*?

- Very broad definition
  - Set of independent, interconnected processors that communicate and coordinate their action by exchanging messages
  - … appears to its users as a single coherent system: DNS, 'The Web', BitTorrent, FB …

- This means …
  - Concurrent execution is the norm so coordination is important
  - No global clock and limits to the accuracy you can get when synchronizing multiple clocks
  - Independent failures – all systems can fail, distributed systems fail in new ways

Motivations and everyday examples

- Why do you want one?
  - To share resources – physical resources and information
  - To increase performance (multiple CPU, disks, …)
  - To tolerate faults via replication
  - To connect physically separate entities (people, machines)

- Some clear examples
  - Finance and commerce – e-commerce, bitcoin
  - Information society – web, search, user-generated content
  - Arts and entertainment – Gaming, VoD
  - Transport and logistics – smart roads, cars
  - Environmental management – Monitoring of natural phenomena

Trends driving distributed systems

- Mobile and ubiquitous computing
  - Price/performance/size ratios in computer technology
- Pervasive networking
  - Large number of nodes
  - Increased connectivity \
  - Higher bandwidth
- Content sharing
  - Zettabytes of content and rapidly growing
- Distributed systems as a utility
  - An old idea on a whole new dimension

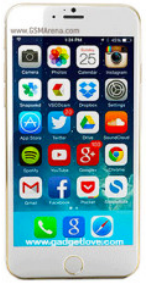
Cheaper & faster machines

IBM 704 Scientific Computing, 1954
40k instructions per second, not sold
but rented at $50,000/month

Apple Lisa, 1983
$24,000 (2015 $)
1MB RAM
Optional 5 or 10MB HDD
Motorola 68000, 5Mhz

IBM AT 80286, 1984
16MB Main Memory
20MB HD
1.5 MIPS on the 10MHz model
$9,600 (2015 $)

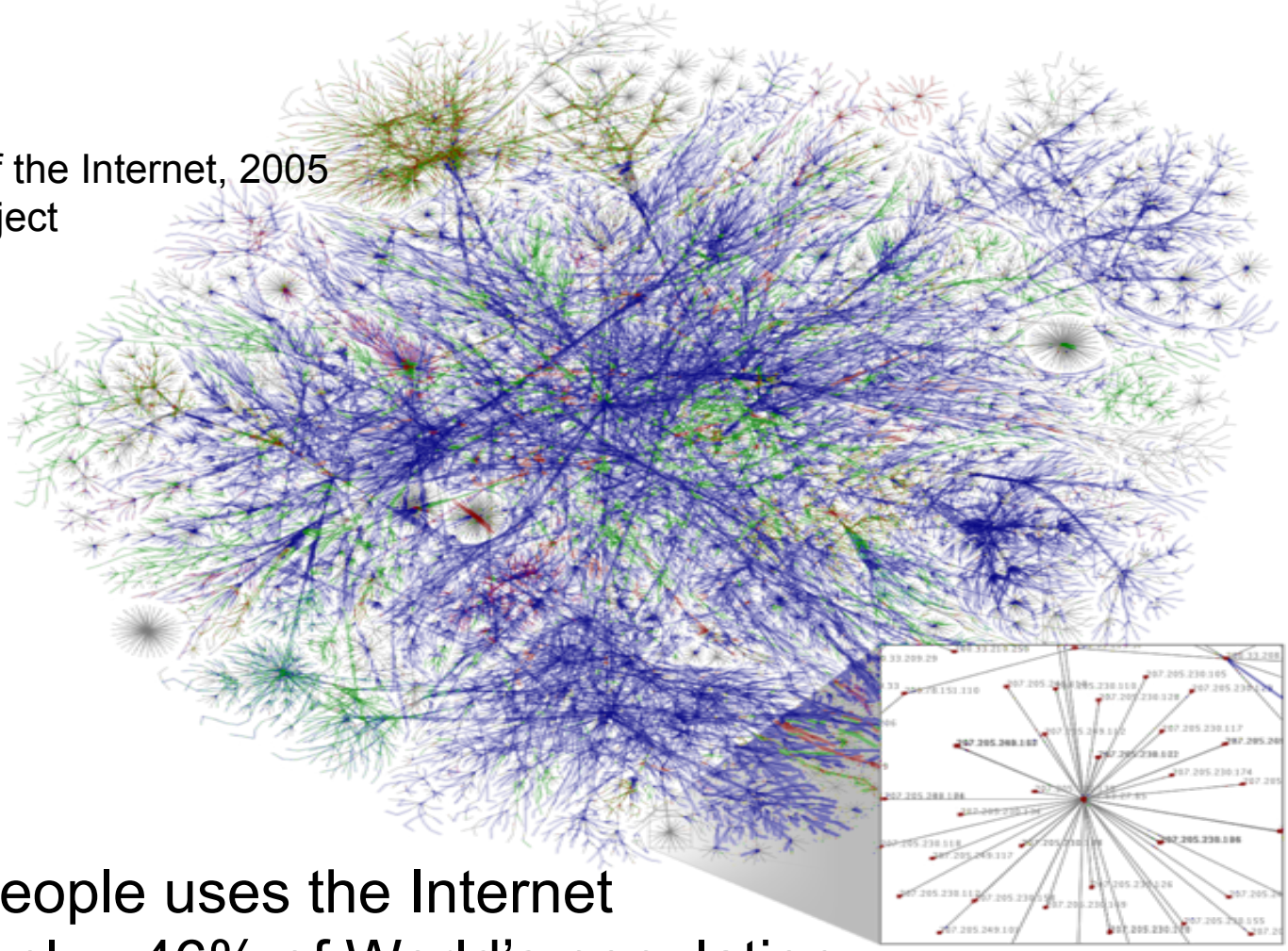
Apple iPhone 6, 2014
128GB Storage
A8 dual-core 64b and M8 chips
$399



Driving DS – An interconnected world

- 29 Oct 1969 – ARPANET first two interconnected nodes
- 1982 – less than 1000 hosts in ARPANET

Partial map of the Internet, 2005
The Opte Project

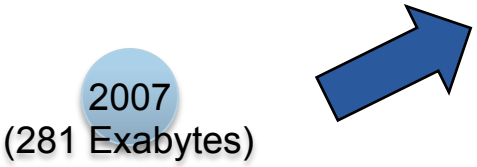


3.4 billion people uses the Internet
And that's only ~46% of World's population

Information growth

Digital universe – data created, captured and replicated – doubling in size every two years

1 Exabyte = 10^{18} bytes, 1,000,000 TB
1 Zettabyte = 10^{21} = 1000 EB



2013
4.4 Zettabytes

Predicted to reach 44 Zettabytes by 2020

EMC, *Digital Universe Report*, 2014

Large-scale distributed systems

1.6 billion in Facebook
305 million monthly active users in Twitter
>70 million Skype users online

Distributed systems challenges

- Making resources available/accessible
- Security – comes with sharing
- Heterogeneity – networks, computer hardware, OS, programming languages, …
- Failures detecting, masking/recovering …
- Scalability in numbers and geographic span
- Transparency
- False assumptions*
  - The network is reliable, secure, and homogeneous, topology does not change
  - Latency is zero, bandwidth is infinite, transport cost is zero
  - There is one administrator

*Based on Peter Detusch (1994) and James Gosling (1997)

Why taking this class? You will learn

- Hard problems, not-obvious solutions
- Active research area
- Behind real systems used by large companies today
  - Google's File system
  - Logical clocks
  - RPCs, RMI, …
  - BitTorrent and Akamai
  - …
- Hands-on approach – build a real distributed system

Course and topic in context

- EECS 340 Networking
  - Network protocols, routing protocols, …
  - Distributed systems: make use of networks
- EECS 343 Operating systems
  - Resource management for single systems
  - Distributed systems: management of distributed resources
- *EECS 345 Distributed systems*
  - Explore solutions to DS problems and challenges
  - Infrastructure software to help build DS and applications

Overview of this course

1. Introduction to Go
2. Networking, communication and overlays
3. Distributed file systems
4. Naming
5. Content distribution networks
6. Time and synchronization
7. Coordination and global state
8. Consensus, replication and fault tolerance
9. Mobile distributed systems

A word about requirements

- We expect you to have taken
  - EECS 213 Intro to computer systems *and*
  - EECS 343 Operating system or 340 Intro to networking
- Some concepts you will need
  - Processes, threads, synchronization, IPC …
  - Basics of network programming
- Some skills you will need
  - Hacking on a Linux/Unix-like environment
  - Solid working experience with C
- *This is a course with a heavy project component!*
  - *Don't take it if you lack the prerequisites*
  - First project should give you a hint

Some practical notes

http://www.aqualab.cs.northwestern.edu/classes/eecs-345-s16

- Textbooks and papers
  - A reference book
  - Some classic and some research papers
- Homework assignments as reader enforcers
- Projects
  - … in a second …
- Final exam
- Yes, yes, grades
  - Homework assignments 15% (+5% paper-related questions)
  - Class participation 10%
  - Project 50%
  - Exam 25% (+5% paper-related questions)

Projects

- Using Go
  - A language well suited for DS
  - To ensure the hard problems have to do with DS (e.g., no compiler, libraries, or pointers)
- Three parts
  - Project 0 – 1 week – *Should you continue?*
  - Project 1 & 2 – 6 weeks – Implement the Kademlia DHT
  - Project 3 – 3 weeks – Build a storage system for self-destructive objects on top of your DHT
- First project on your own
- The rest in teams of 2-3 students (3 is preferred, 1 is not allowed)

On projects – Code walkthrough

- For every project, code walkthrough by randomly selected groups with the staff

To get full credit for your project you must be able to carry the walkthrough, showing you understand your own code

- A few points
  - Sampling with replacement – you can do it multiple times
  - To be done in first week after submission
  - All team member must be there, everyone drives at one point

Exam

- In-class final
  - One-page of notes (1 or 2-sides)
  - Papers from reading

- Example questions
  - With asynchronous RPC, a client is blocked until its request has been accepted by the server. To what extent do failures affect the semantic of asynchronous RPC?

  - DNS is susceptible to denial of service attacks, one of the papers discussed in class addresses this problem; describe the problem, explain the key ideas behind the authors solutions and their evaluation approach

On class participation

- A Wong-Baker's like scale

Contributes to the discussion (A)

Seem to follow the discussion (B)

I've seen her in class (C)

She's in class, but probably on FB (D)

Is he actually in this class? (F)

On cheating/plagiarism

- Collaboration is good
- Cheating is a very serious offense
- It's OK to
  - Meet with colleagues
  - Study for exams together
  - Discuss assignments with them
- But, *what you turn in must be your own work*
  - Do not copy code, solution sets, etc. from other people or any other sources (**Web included!**)
  - Do not make your code available for other (even future) students (**e.g., do not post in github, bitbucket, …**)

Go on, take 5'

Talking about DS – models & architectures

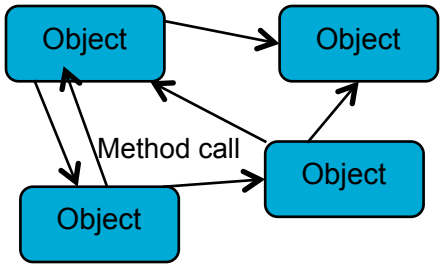
- Models – abstract, simplified but consistent description of a relevant aspect of a system

  "All models are wrong, but some are useful", George E. P. Box

- Distributed systems – types of models
  - Architectural – how to organize the system's software components, how components interact (software architecture)
  - Physical – how to instantiate the set of components on real machines (system architecture)
  - Fundamental – focus on particular aspects such as interactions, failures, security

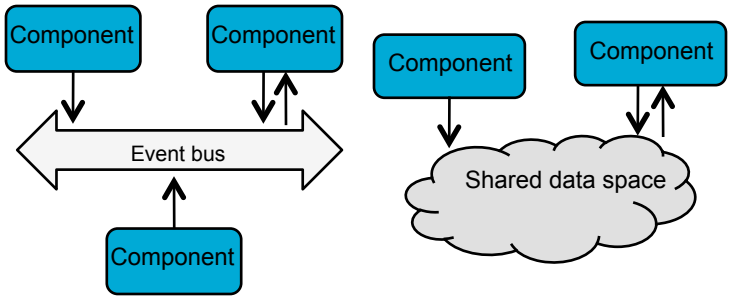
Architectural elements

- Components/entities and their interrelationships
- Communicating entities
  - Systems' perspective – processes, threads or nodes (in some primitive environments, there's no process abstraction)
  - Programming perspective – objects, components or web services



- Communicating paradigms
  - Inter-process communication
  - Remote invocation – RPC, RMI …
  - Indirect communication – pub/sub, distributed shared memory …

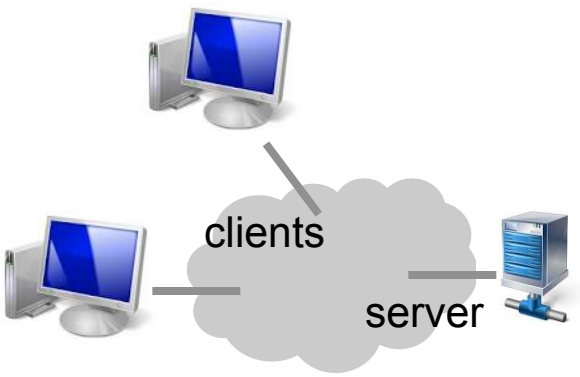
Referential decoupling – Event-based, publish-subscribe

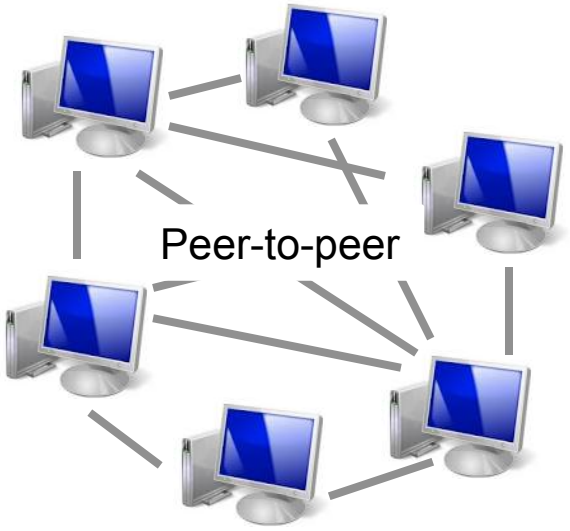


Temporal decoupling – Shared data spaces

Architectural patterns

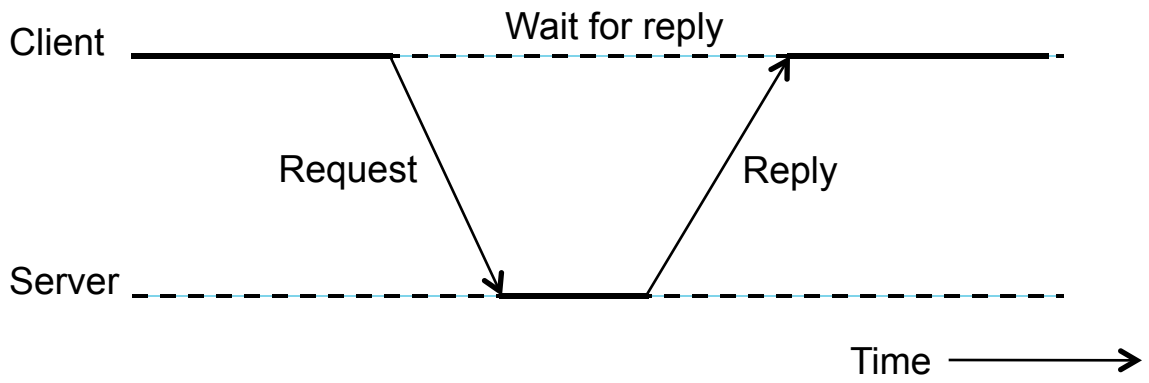
- Roles and responsibilities / system architecture
  - As processes (objects, components, …) interact, they take on given roles that define the architecture, e.g., client-sever or P2P
- A classification based on the symmetry of the interaction between components





Client-server

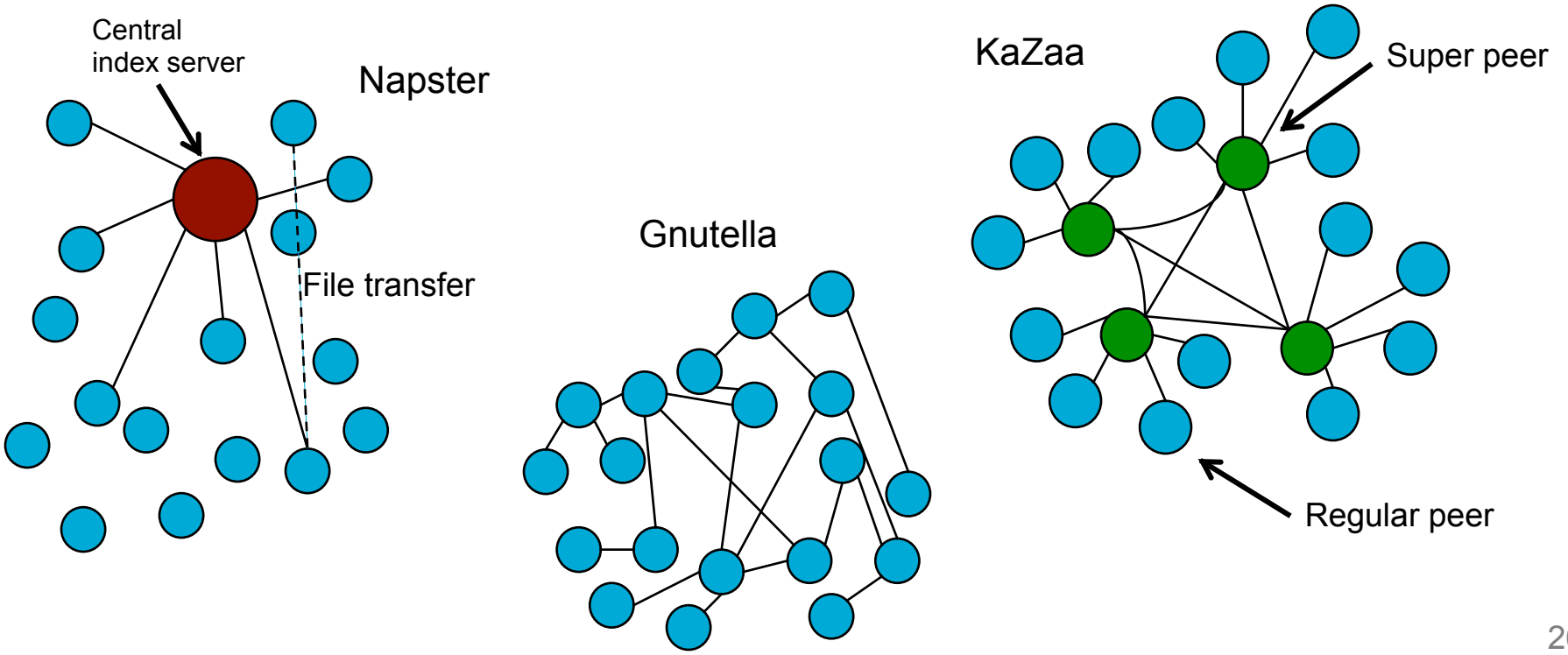
- Basic idea
  - Processes offering services, others using them
  - Client and servers can be on different machines
  - Interaction follows a request/reply model



- Dealing with the scalability issues of servers
  - One service, multiple servers
  - Caching and replication
  - Mobile code and mobile agents

Peer-to-peer – we are all (mostly) equal

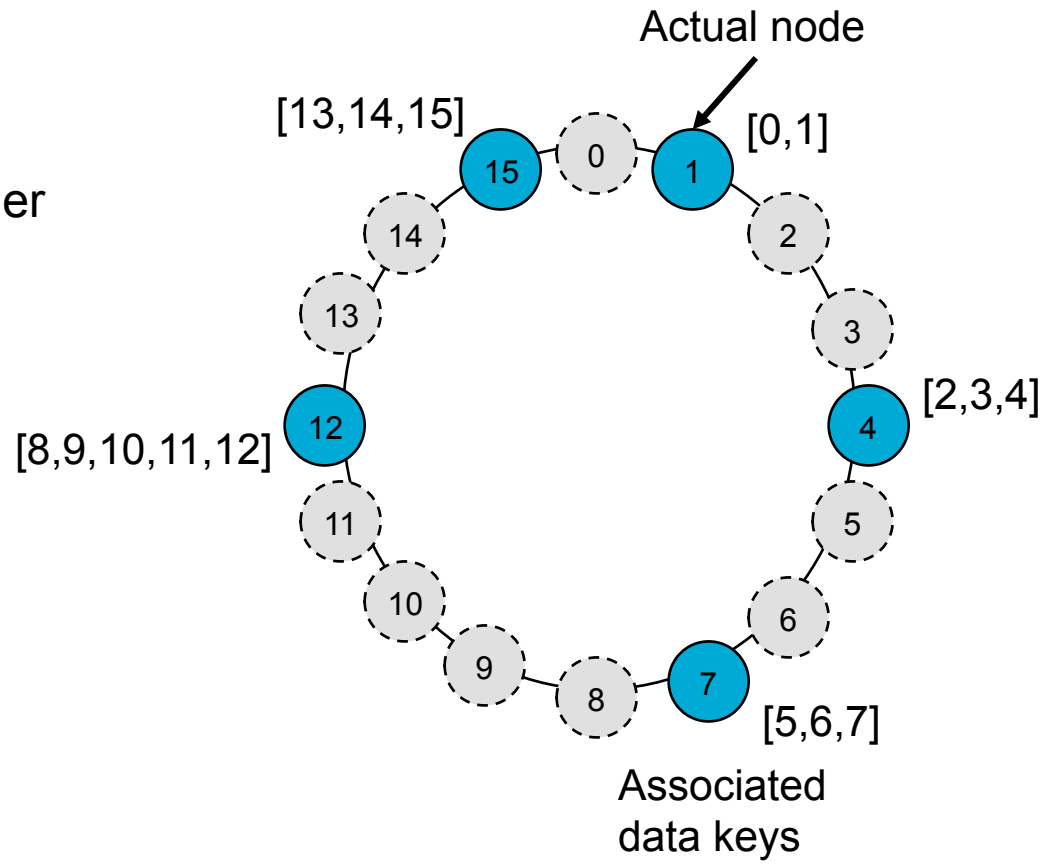
- For scalability, everyone contributes
  - Every client chips in CPU, storage or network resources
- From Napster on …
  - How do you find a given file?
  - Centralized, unstructured and semi-structured models



Structured P2P

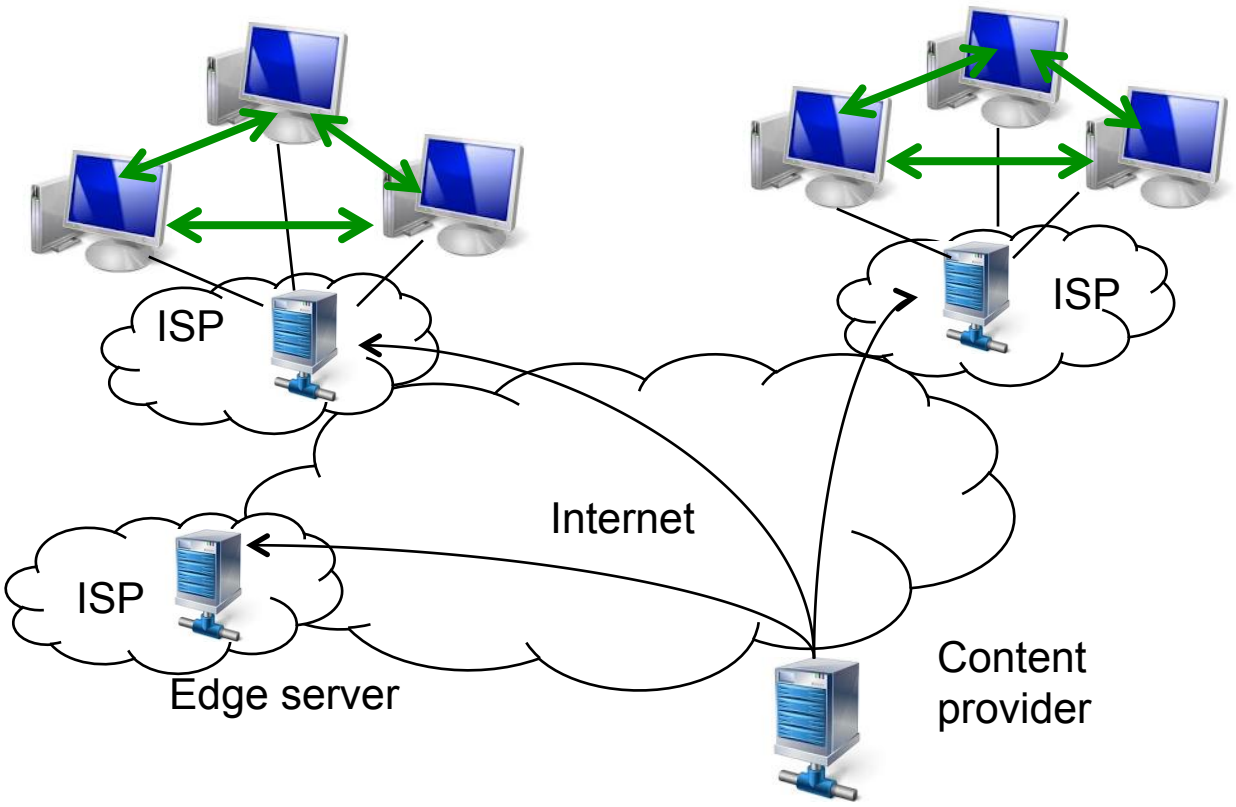
- A bit of structure for more efficient searches
  - E.g. Distributed Hash Tables
  - Use a variant of consistent hashing to assign ownership of each file to a particular peer

Data item with key *k* is mapped to the node with smallest identifier *id >= k (successor(k))*



Hybrid models

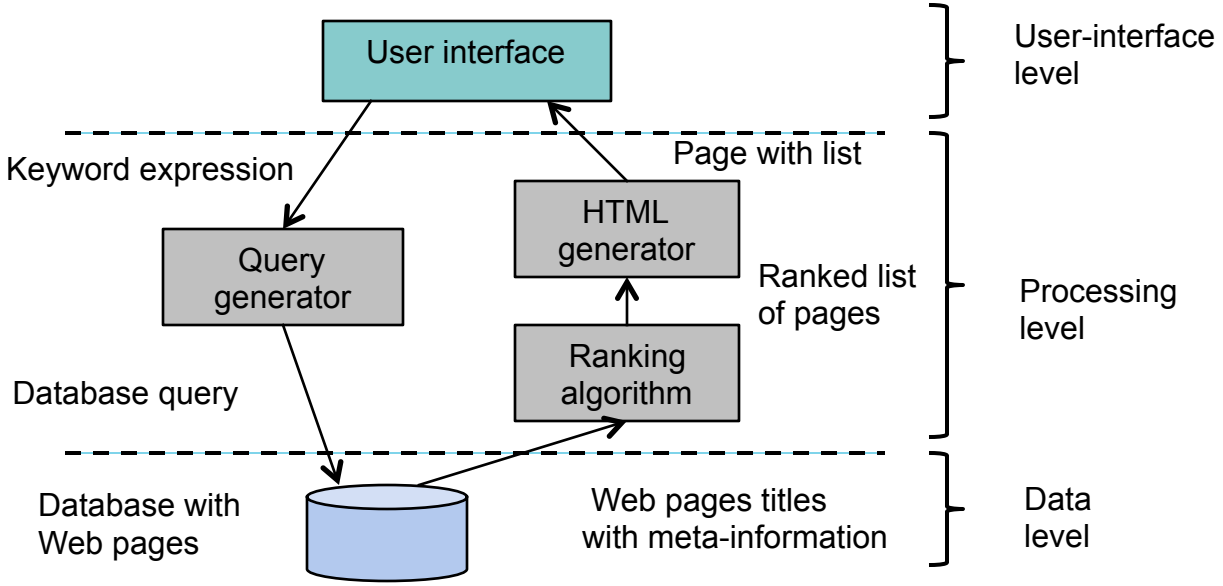
- Edge-server systems – typical of content distribution networks (CDNs)
  - Clients get content through an edge server
- Peer-assisted CDNs



Layers and tiers

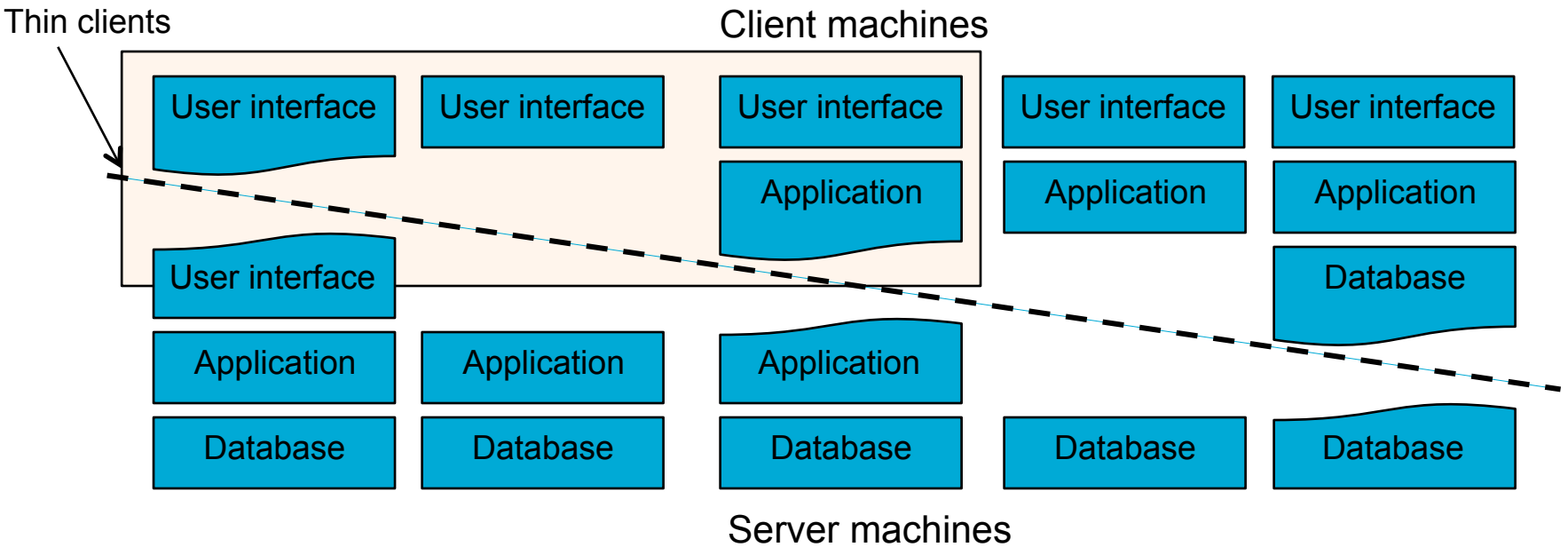
- Server for some and client of others
- A typical three-layer model for database access
  - User-interface – UI part of an application
  - Processing – functionality of an application without data
  - Data – data to be manipulated through the application
    - Responsible for ensuring data is persistent and consistent
    - Many times a relational database

A simple example – an Internet search engine



Multi-tiered architectures

- Organization and placement
- Three logical layers with multiple instantiations
- How much to place on each side?
  - Thinner or fatter clients
    - Fat's good – Short(er) response times and leverage clients' resources
    - Thin's good – Easy to manage



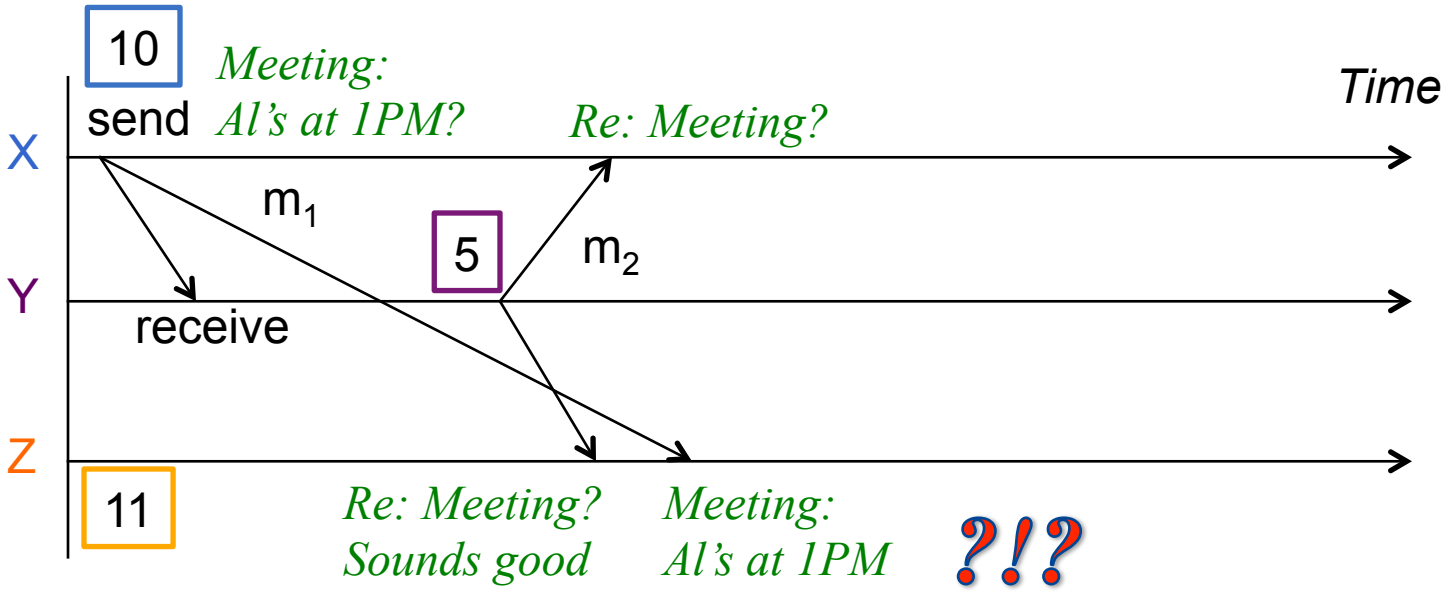
Fundamental models – Interaction

- Focused on fundamental properties, making explicit all relevant assumptions, to reason about
- Interaction models
  - Synchronous - Time to execute step of a process, transmit a message and the drift rate of clocks have known upper bounds
  - Asynchronous – There are no bounds

- Is asynchronous realistic? Is there any “send” that will take >1,260 secs (longest delay to Mars)?!
  - Not the point – General solutions that won’t fail if the assumptions (delay) fail

Fundamental models

- Event ordering, ignoring clocks
- Process interaction through message exchanges
  - Communication performance is often a limiting characteristic
  - No global clock or global notion of time
    - Clocks drift at different rates
    - Logical clocks and some basic maybe enough



Fundamental models

- Agreeing on failures – failure models
  - Failures of entities and communication channels
  - To understand the effect of failures
- Omission failures
  - Process or communication channel fails to do what's expected
  - Crashed – halted for good, it's not responding!
  - *How do you detect a crash in an asynchronous system?*
- Arbitrary or Byzantine failures
  - Anything is possible, e.g., server may send a wrong response
  - *How do you reach consensus?*
- Timing failures in synchronous systems
  - E.g., process or message takes longer than stated bounds

Next time

- A brief introduction to Go (by Zach Bischof)
- Before
  - Take the Go tour if you haven't!
  - Join Piazza
  - Look for a team