Class Introduction

◆

Principles of Systems Design

COS 518 *Advanced Computer Systems*

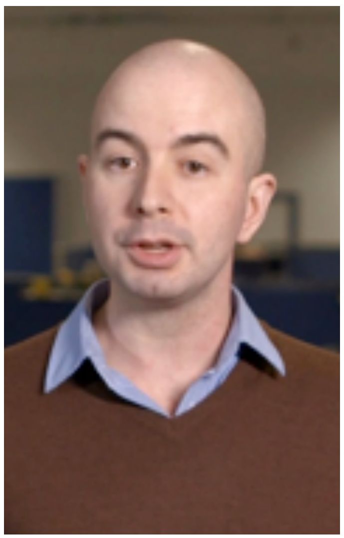
Lecture 1

Kyle Jamieson

Today

- **Welcome to COS-518!**

- **Course staff and office hours:**



Instructor
Kyle Jamieson
COS 305
x8-7477
W 10-11 AM

TA
Logan Stafman
COS 317
850-510-8280
M 10-11 AM

Today

- Welcome to COS-518!

1. **Goals and high-level topics**

2. Course administrivia

3. Systems design
   - "Worse is Better"
   - Lampson's "Hints for Computer System Design"

Goals of this course

- Introduction to
  - – Computer systems **principles**
  - – Computer systems **research**
    - Historical and cutting-edge research
    - How “systems people” think

- Learn how to
  - – **Read** and **evaluate** papers
  - – **Give talks** and evaluate talks
  - – **Build systems** and **write** papers

What is a system?

- **System**
  - Inside v. outside: a system defines an interface with its environment
  - A system achieves specific external behavior
  - A system has many components

- This class is about the design of **computer** systems

- Examples: a PC, a bank ATM, the WWW

- Much of class will operate at the design level
  - Relationships of components
  - Internals of components that help structure

The central problem: Complexity

- Complexity's hard to define, but symptoms include:

1. Large number of **components**
2. Large number of **connections**
3. Irregular **structure**
4. No short description
5. Many people required to design or maintain

Organization of the semester

1. Introduction to systems principles

- Concepts in modularity, abstraction, naming, and communication
  - Lampson's "Worse is Better"
  - Saltzer's end-to-end principles
- Classical computer systems
  - *Plan9* operating system, the Log-Structured File System (*LFS*), the Self-Certifying File System (*SFS*)

Organization of the semester

1. Introduction to systems principles

2. **Distributed systems**

   – Consistency and performance
     • *System R*, Lamport clocks, Saltzer & Kaashoek
     • The *Paxos* algorithm for **distributed consensus**

   – Systems building on this knowledge
     • *CRAQ*, *Spanner*

Organization of the semester

1. Introduction to systems principles

2. Distributed systems

3. **Mobile and Cloud systems**
   - *Sensor Hints*
   - *MAUI* code offload architecture for mobile
   - COMET code offload between VMs
   - Interactive and real-time applications
     - Real-time face recognition
     - Gaming

Organization of the semester

2. Distributed systems

3. Mobile and Cloud Systems

4. **Scaling storage and data processing**
   - Weaker consistency models
     - *Bayou*, *Dynamo*
   - *MapReduce*
   - Back to cloud: Geo-distributed data analytics, latency, and bandwidth

Organization of the semester

3. Mobile and Cloud systems

4. Scaling storage and data processing

5. **Concurrency and performance**
   - Memory and thread management
   - Concurrency in web server and general software design: *Flash*, *SEDA*

Organization of the semester

4. Scaling storage and data processing

5. Concurrency and performance

6. **Security**
   - Ken Thompson's Turing Lecture *Trusting Trust*
   - Saltzer's principles of information protection
   - Guest lecture by Philipp Winter (Tor developer)
   - Untrusted cloud infrastructure (*CryptDB*, *SPORC*)
   - Deniable/anonymous communication (*Denali*)

Organization of the semester

5. Concurrency and performance

6. Security

7. **Project presentations**
   - Open-ended class project
   - Build the software, write it up, present it to the class
   - More details later today…

Today

- Welcome to COS-518!

1. Goals and high-level topics

2. **Course administrivia**

3. Systems design
   - "Worse is Better"
   - Lampson's "Hints for Computer System Design"

Format of this course

1. **Lecture:** Introducing a subject
   - Older "time-tested" papers, and book readings
   - Method of delivery: Read on own, and attend lecture
   - Slides will be posted on web just after lecture

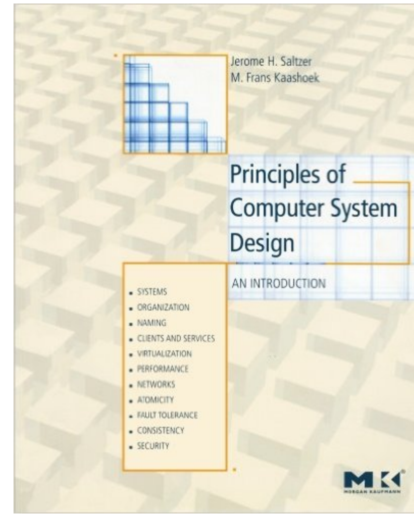
2. **Paper discussion:** Learning about new research directions, results
   - Newer papers from the literature
   - Method of delivery: Read and evaluate one of three papers (using HotCRP review platform)
     - One person presents, others add to discussion

Paper discussion: Logistics

- ≈ four working days prior: **Signup deadline** on Piazza to commit to one of the day's papers
  - **One half** of the class signs up for **each** paper
  - First come, first served conflict resolution
- ≈ two working days prior: **Review deadline** on HotCRP to write a paper review
- For the class meeting: Read each others' reviews
- **Once** per student, per term: **Present a paper**
  - Volunteer to present early, or we assign you later

Course text

- **Required text:** Principles of Computer System Design: An Introduction, by J. Saltzer and M. Kaashoek
  - ISBN 978-0-12-374957-4
  - Weekly readings from this text



- First ½ available from Labyrinth Books on Nassau St, and in print and e-reader editions from online retailers

- Download the second ½ for free from MIT Open Courseware

Class communication: website

- Find it at http://cos518.cs.princeton.edu
- Contains **detailed calendar,** meeting times and places, reading assignments and deadlines
  - You're responsible to **check it daily** for reading assignments (not all on class meeting days)
- Website contains links to **Piazza** discussion forum and **HotCRP** paper review system

Class communication: Piazza

- Staff and students **discuss, post questions, and answer questions** on papers and readings
- Receive important announcements from class staff (also forwarded to you by email)
- **Signup today** at http://piazza.com/princeton/fall2015/cos518
  - You must subscribe (class policy)
    - Most grad students already subscribed
- Your responsibility: **check email daily!**

Using Piazza

- Please **post** questions on class material **on Piazza**, rather than emailing course staff:
  - **Faster response,** whole class benefits from seeing your question and its answer
    - Students encouraged to answer student questions!
  - If we think class will benefit from our answer, we may mark private questions as public (preserving privacy and academic integrity)
- When discussing something private (*e.g.*, grades), mark your post as private, so only staff see it!

Course project

- Semester-long, open-ended systems research
  - Groups of two to three per project

- Project schedule:
  - Form groups by Monday, September 28
  - **Idea pitch:** Group meetings with me in early Oct
  - Written proposal: (on HotCRP, others review), early Nov
  - Presentation and prelim v. 0 demo (Dec 14, 16)
  - 5-page paper on v. 1 system (Dean's date, 1/12/16)
    - **Working source code** on github or bitbucket

Project

- Two choices:

1. New research

2. Reimplement system in one of papers we read
   – Give a little twist on it, or evaluate it in a different way, try some of the future work, & c.

- Must be working code!
  – I get to view source in repo

Grading

- 25% class participation

- 25% reading responses ("reviews")
  - Graded on a three-point scale
    - 0: Not submitted or content-free
    - 1: Submitted and intelligible
    - 2: Mostly correct
    - 3: Correct, salient, and complete

- 50% project:
  - 10% checkpoint #1 (proposal)
  - 10% checkpoint #2 (presentation + demo)
  - 30% final report + code

Today

- Welcome to COS-518!

1. Goals and high-level topics

2. Course administrivia

3. **Systems design**
   - Worse is Better
   - Lampson's "Hints for Computer System Design"

Systems challenges common to many fields

1. **Emergent properties (“surprises”)**

   – Properties not evident in **individual** components become clear when **combined** into a system

   – **Millennium bridge,** London example


Tate Modern
Collection 2006

Millennium bridge

- Small lateral movements of the bridge **causes** synchronized stepping, which **leads to** swaying
- Swaying **leads to** more forceful synchronized stepping, **leading to** more swaying
  – Positive feedback loop!
- Nicknamed *Wobbly Bridge* after charity walk on Save the Children
- Closed for two years soon after opening for modifications to be made (**damping**)

Systems challenges common to many fields

1. Emergent properties ("surprises")

2. **Propagation of effects**
   - **Small/local** disruption → **large/systemic** effects
   - Automobile design example (S & K)

Propagation of effects: Auto design

- **Want a better ride** so increase tire size
- Need larger trunk for larger spare tire space
- Need to move the back seat forward to accommodate larger trunk
- Need to make front seats thinner to accommodate reduced legroom in the back seats
- **Worse ride** than before

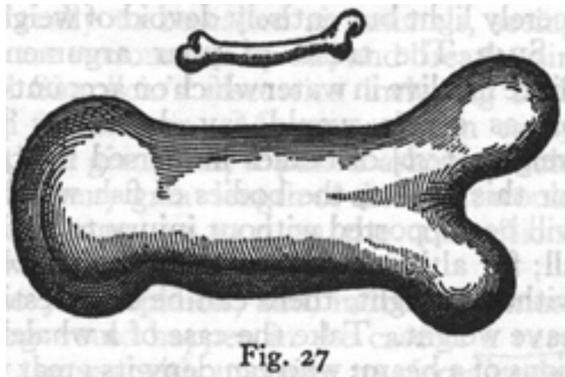
Systems challenges common to many fields

1. Emergent properties (“surprises”)

2. Propagation of effects

3. **Incommensurate scaling**
   – Design for a smaller model may not scale

Galileo in 1638



“To illustrate briefly, I have sketched a bone whose natural length has been increased three times and whose thickness has been multiplied until, for a correspondingly large animal, it would perform the same function which the small bone performs for its small animal…

Thus a small dog could probably carry on his back two or three dogs of his own size; but I believe that a horse could not carry even one of his own size.”

—Dialog Concerning Two New Sciences, 2nd Day

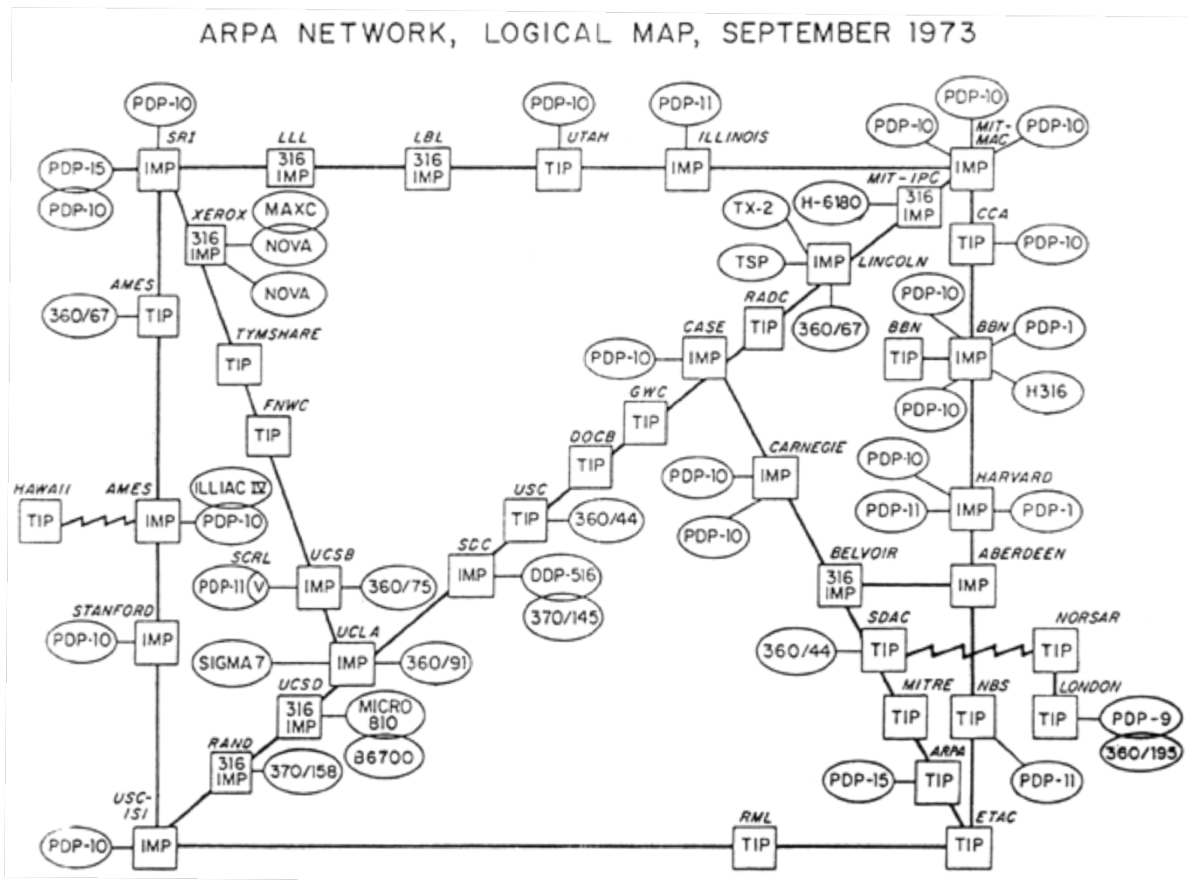
Incommensurate scaling

- **Scaling a mouse into an elephant?**
  - Volume grows in proportion to $O(x^3)$ where x is the linear measure
  - Bone strength grows in proportion to cross sectional area, $O(x^2)$
  - [Haldane, "On being the right size", 1928]

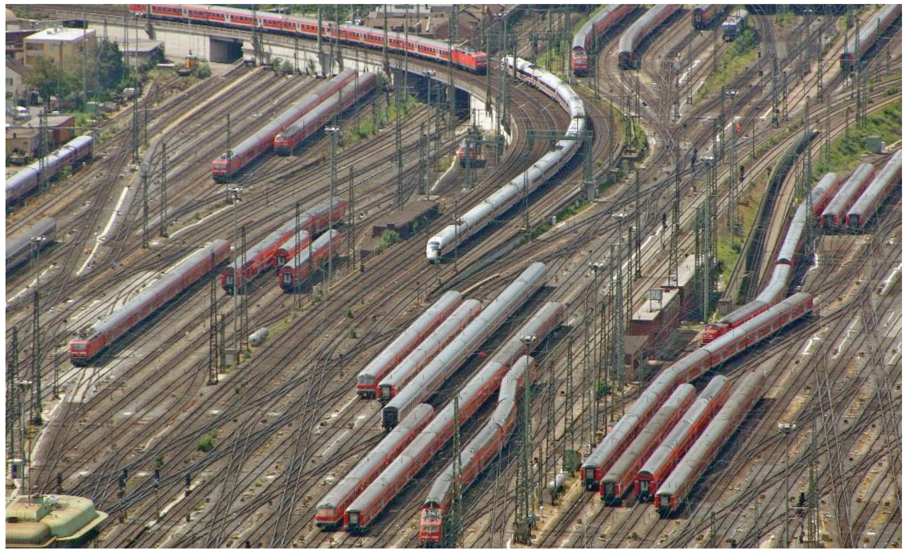
- Real elephant **requires** different skeletal arrangement than the mouse

Incommensurate scaling: Scaling routing in the Internet

- Just **39 hosts** as the **ARPA net** back in **1973**



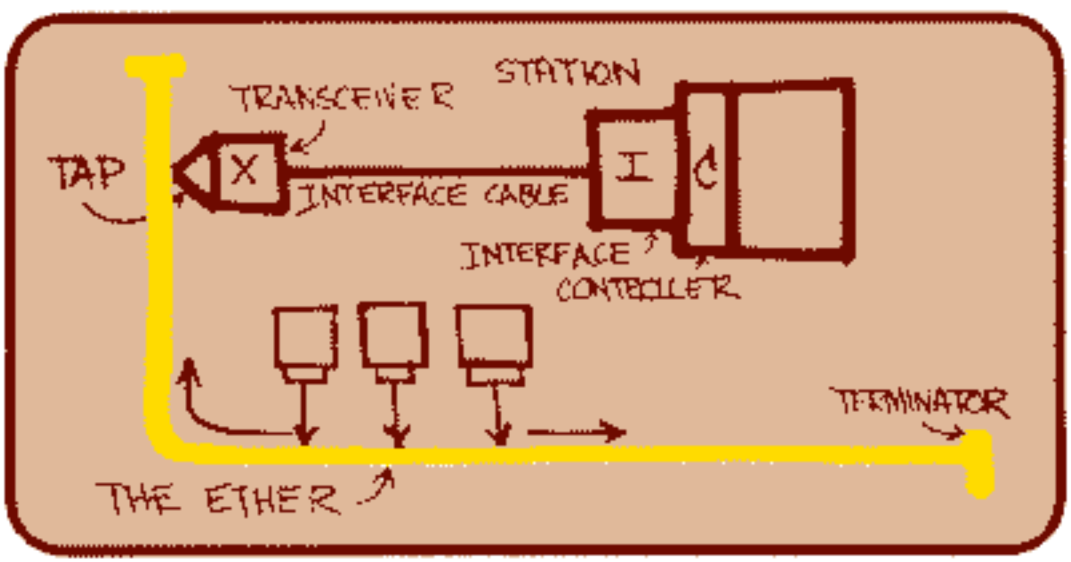
Incommensurate scaling: Scaling routing in the Internet



- Total size of routing tables (for shortest paths): **$O(n^2)$**
- Today's Internet: Techniques to **cope with scale**
  - **Hierarchical routing** on network numbers
    - 32 bit address =16 bit network # and 16 bit host #
  - Limit # of hosts/network: **Network address translation**

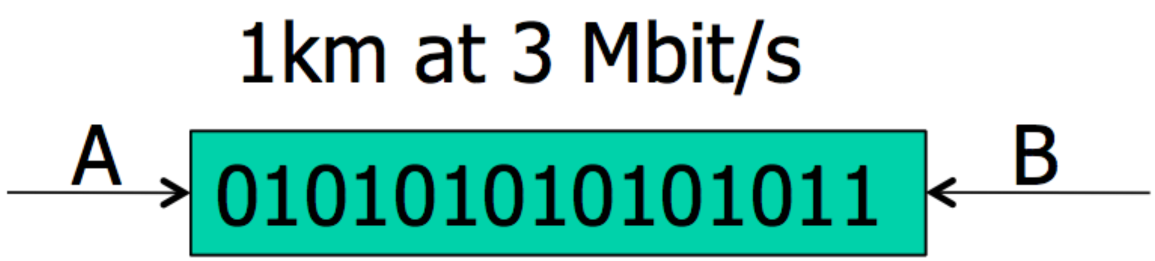
Incommensurate Scaling: Ethernet

- All computers share single cable
- Goal is reliable delivery
- **Listen-while-send** to avoid collisions



Will listen-while-send detect collisions?

- 1 km at 60% speed of light is 5 μs
  - **A** can send 15 bits before first bit arrives at **B**
- Therefore **A** must keep sending for 2 × 5 μs
  - To detect collision if **B** sends when first bit arrives
- Therefore, minimum packet size is 2 × 5 μs × 3 Mbit/s = 30 bits



From the experimental Ethernet to the Ethernet standard

- Experimental Ethernet design: **3 Mbit/s**
  - – Default header is 5 bytes = 40 bits
  - – No problem with detecting collisions
- First Ethernet standard: **10 Mbit/s**
  - – Must send for 2 × 20 μs = 400 bits
    - But header is just 112 bits
  - – **Need for a minimum packet size!**
- **Solution:** **Pad packets** to at least 50 bytes

Systems challenges common to many fields

1. Emergent properties ("surprises")
2. Propagation of effects
3. Incommensurate scaling
4. **Trade-offs**
   – Many design constraints present as trade-offs
   – Improving one aspect of a system diminishes performance elsewhere

Binary classification trade-off

- Have a ***proxy signal*** that imperfectly captures **real signal of interest**

- **Example:** Household smoke detector

		Real categories	
		fire	no fire
Proxy categories	detector signals	TA: fire extinguished	FA: false alarm
	detector quiet	FR: house burns down	TR: all quiet

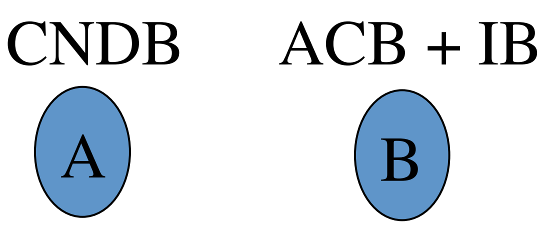
Sources of complexity

1. Cascading and interacting requirements

– **Example:** Telephone system

- Features: Call Forwarding, reverse billing (900 numbers), Call Number Delivery Blocking, Automatic Call Back, Itemized Billing

– **A** calls **B, B** forwards to 900 number, who pays?



- **A** calls **B, B** is busy
- Once **B** done, **B** calls **A**
- **A's** number appears on **B's** bill

Interacting Features

- Each feature has a spec
- An interaction is bad if feature X breaks feature Y
- These bad interactions may be fixable…
  - But there are so many interactions to consider: huge source of complexity.
  - Perhaps more than n^2 interactions, *e.g.* triples
  - Cost of **thinking about / fixing interaction** gradually grows to dominate software costs
- Complexity is super-linear

Sources of complexity

1. Cascading and interacting requirements

2. **Maintaining high utilization of a scarce resource**
   - **Example:** Single-track railroad line running through a long canyon
     - Might use a pullout and signal to allow bidirectional ops
     - But now need careful scheduling
     - **Emergent property:** Train length < pullout length

Coping with complexity

1. **Modularity**
   - Divide system into ***modules,*** consider each separately
   - Well-defined interfaces give flexibility and isolation
     - Hide implementation, thus, it can be freely changed

- Example: **bug count** in a large, ***N*-line** codebase
  - Bug count $\propto N$
  - Debug time $\propto N \times$ bug count $\propto \boldsymbol{N^2}$
- Now divide the N-line codebase into ***K*** modules
  - Debug time $\propto (N/K)^2 \times K = \boldsymbol{N^2/K}$

Coping with complexity

1. Modularity

2. **Abstraction**
   - The ability of any module to treat other modules like a “black box”
     - Just based on the other module’s interface
     - Without regard for the other’s internal implementation

   - Symptoms:
     - Fewer interactions between modules
     - Less ***propagation of effects*** between modules

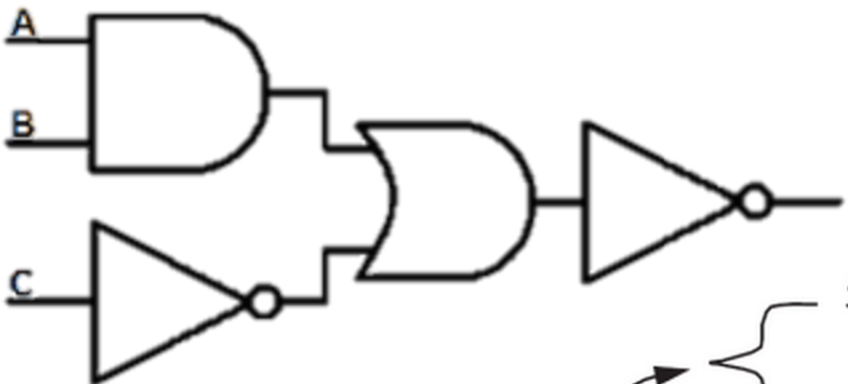
Coping with complexity

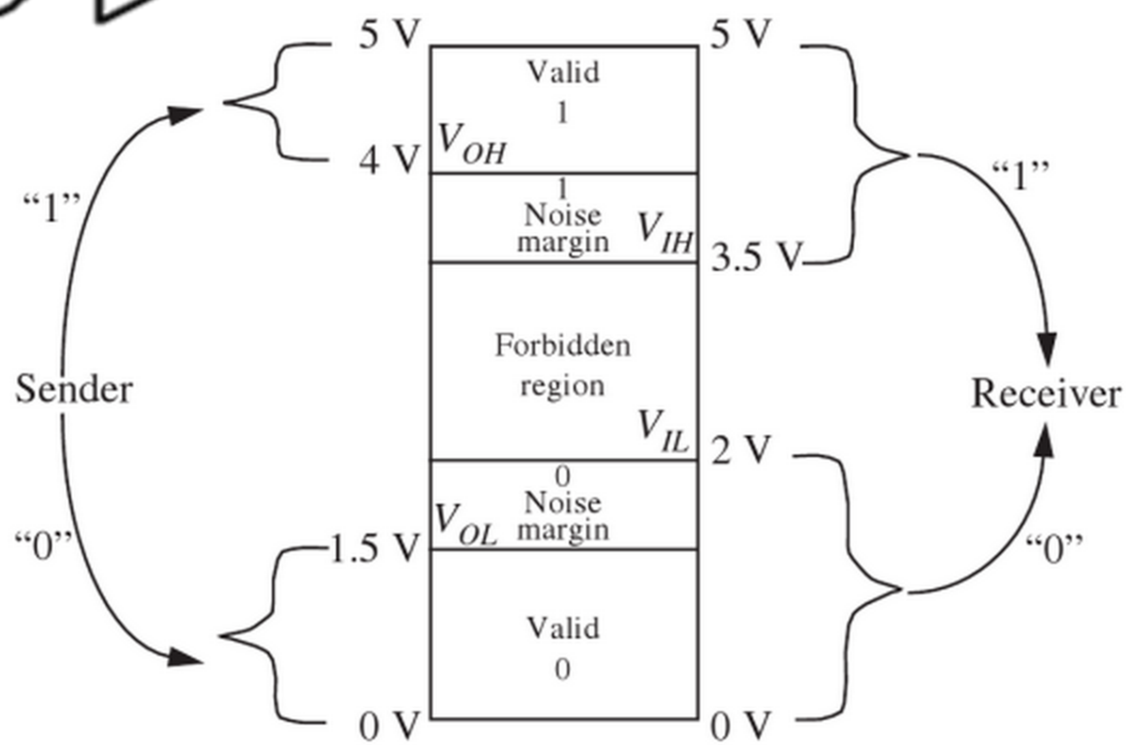
1. Modularity

2. **Abstraction**

   – **The Robustness Principle:** Be tolerant of inputs and strict on outputs

Robustness principle in action: The digital abstraction





Coping with complexity

1. Modularity

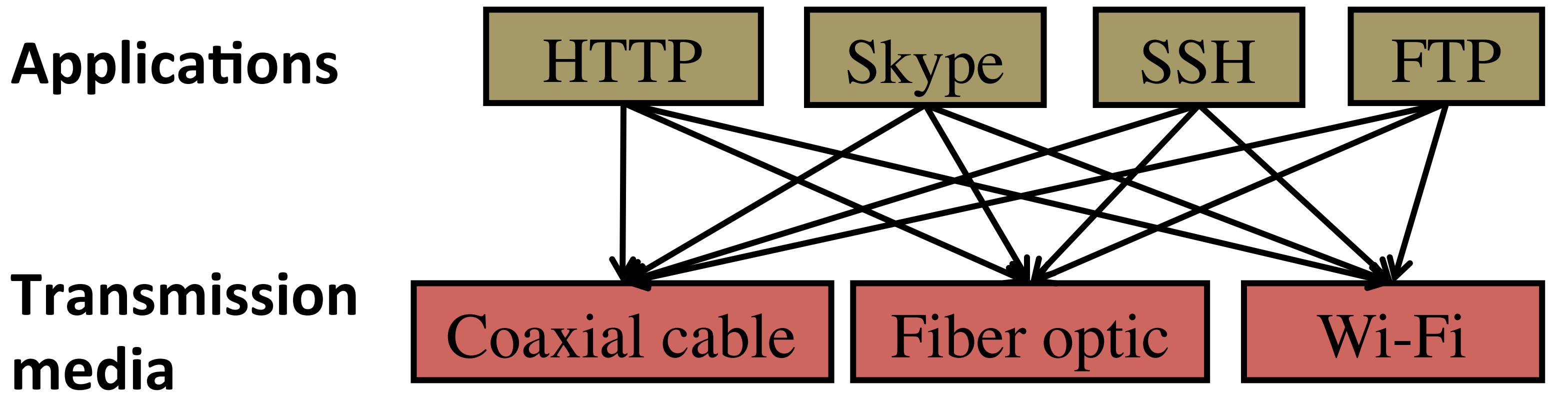
2. Abstraction

3. **Hierarchy**
   - Start with small group of modules, assemble
     - Assemble those assemblies, *& c.*
   - Reduces connections, constraints interactions

Coping with complexity

1. Modularity
2. Abstraction
3. Hierarchy
4. **Layering**
   - A form of modularity
   - Gradually build up a system, layer by layer
   - **Example: Internet protocol stack**

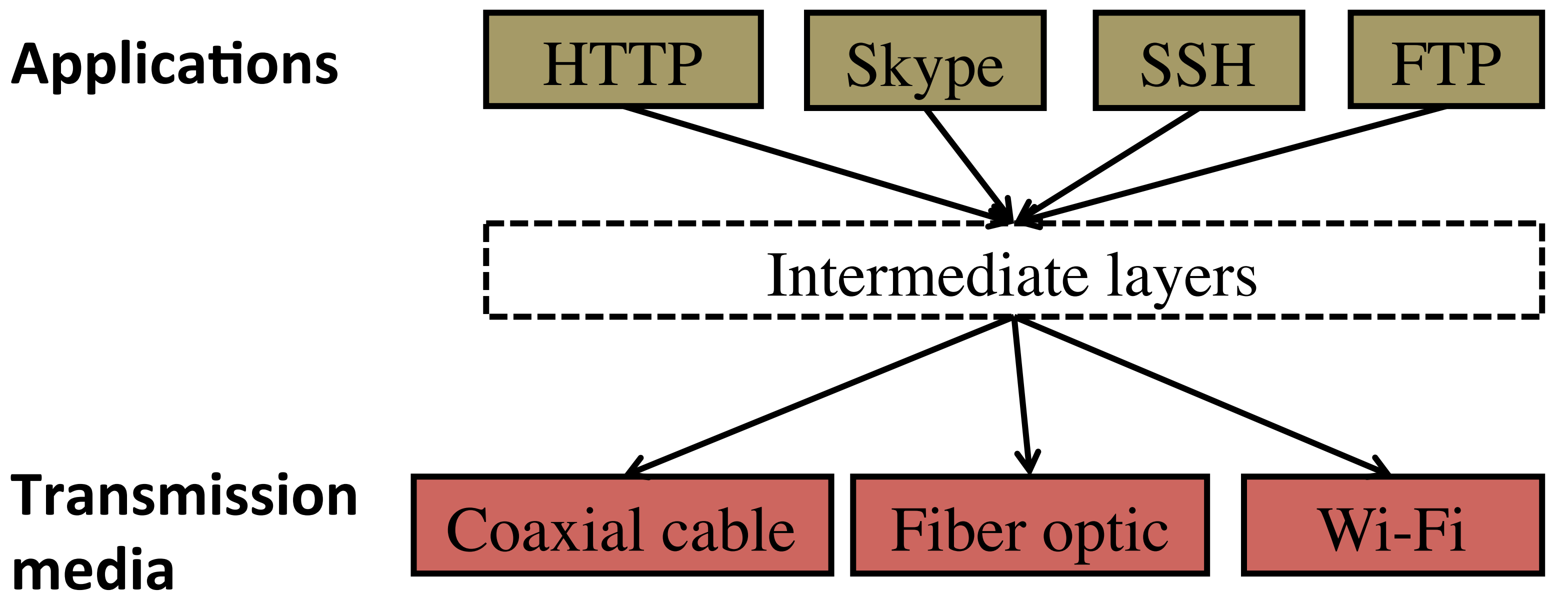
Layering on the Internet: The problem



- Re-implement every application for every new underlying transmission medium?
- Change every application on any change to an underlying transmission medium (and vice-versa)?

- **No!** But how does the Internet design avoid this?

Layering on the Internet: Intermediate layers provide a solution



- Intermediate layers provide a set of abstractions for applications and media
- New applications or media need only implement for intermediate layer's interface

Computer systems: The same, but different

1. **Often unconstrained by physical laws**
   - Computer systems are **mostly digital**
   - **Contrast:** **Analog** systems have **physical limitations** (degrading copies of analog music media)
   - Back to the **digital static discipline**
     - Static discipline **restores signal levels**
     - Can therefore **scale** microprocessors to billions of gates, encounter new, **interesting emergent properties**

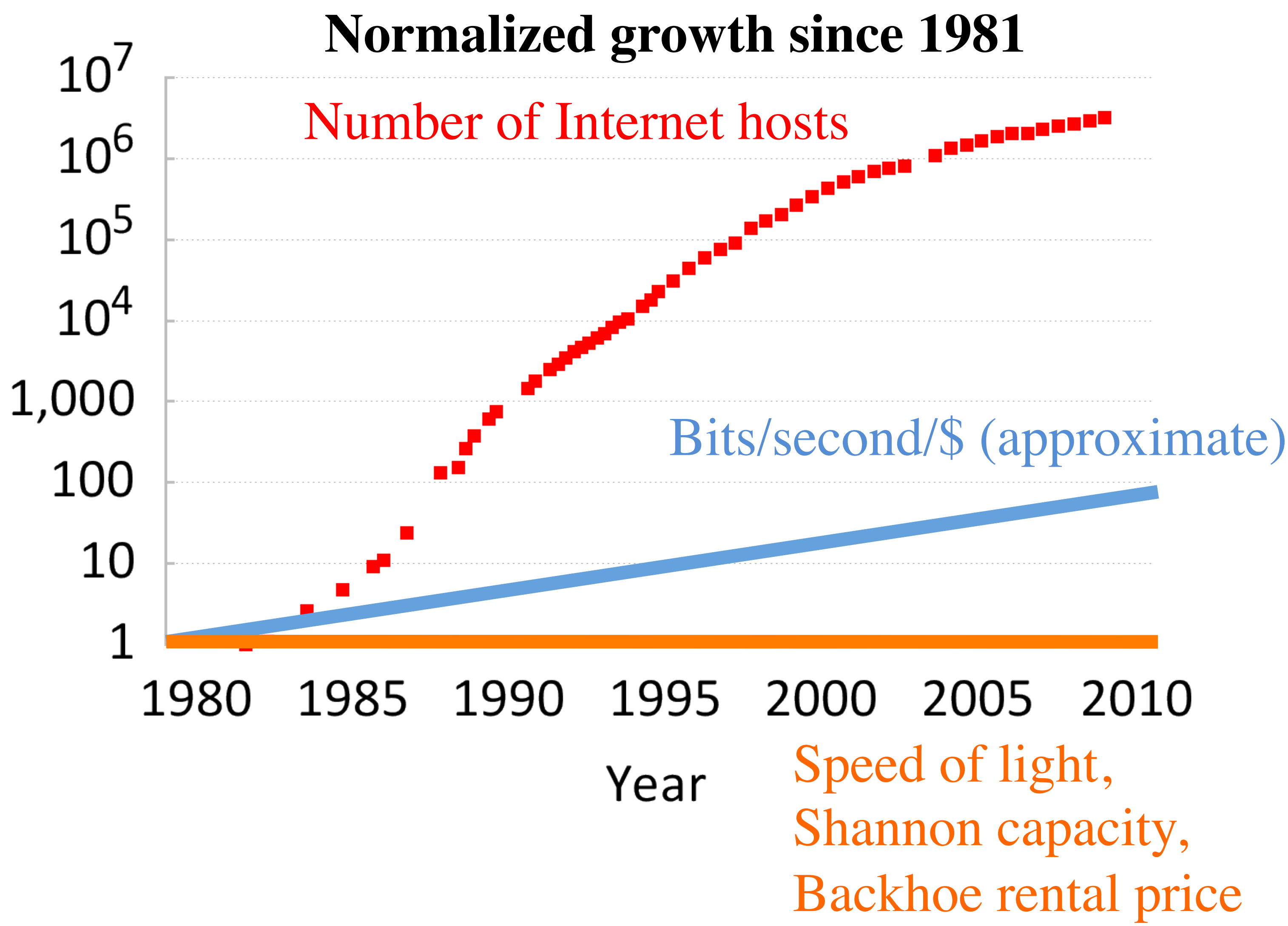
Computer systems: The same, but different

1. Often unconstrained by physical laws

2. **Unprecedented *d*(technology)/*dt***
   – Many examples:
     • Magnetic disk storage price per gigabyte
     • RAM storage price per gigabyte
     • Optical fiber transmission speed

   – **Result:** Incommensurate scaling, with system redesign consequences

Incommensurate scaling on the Internet



Summary and lessons

- **Expect surprises** in system design
- There is **no small change** in a system
- 10-100× increase? ⇒ perhaps re-design
- Complexity is **super-linear** in system size
- Performance cost is super-linear in system size
- Reliability cost is super-linear in system size
- **Technology's high rate of change** induces incommensurate scaling

Today

- Welcome to COS-518!

1. Goals and high-level topics

2. Course administrivia

3. **Systems design**
   - **“Worse is Better”**
     - **Richard P. Gabriel (known for Common Lisp)**

   - Lampson’s “Hints for Computer System Design”

Setting: The two approaches

MIT approach

- **Simplicity:** Must be simple in both implementation, and **especially interface**
- **Correctness:** Must be **absolutely** correct in **all** aspects
- **Completeness:** Must cover **all** reasonably expected cases, even to **detriment** of simplicity

New Jersey approach

- **Simplicity:** Must be simple in both interface and **especially implementation**
- **Correctness:** Must be correct, but slightly better to be **simple**
- **Completeness:** Cover **as many** cases **as is practical**
  - Can sacrifice for other property, **must** sacrifice for **simplicity**

Worse is better!

- In your favorite language, what does the following compute (suppose `x` is an integer): `x + 1`
  - **Scheme:** Always calculates an integer value one larger than `x`
  - **Most others** including **C:** Something like `(x + 1) mod` 2^{32}

- **C:** **simple** implementation, **complex** interface
  - This is the **key tradeoff** that Gabriel describes
  - Probably not what the programmer actually wanted
  - But, **it works in the common case,** and most languages follow the New Jersey approach!

Worse is worse!

- Consider **`fgets(char *s, int n, FILE *f)`** **versus** **`gets(char *s)`**
  - **`fgets`** **limits** the length of the string stored to the size specified by **`n`**
  - **`gets`** stores into **`s`** **however many** characters from `stdin` are ready for input

- *Which is the MIT approach? Which is the New Jersey approach?*

- **`gets`** has been implicated in many **buffer overflow security exploits**
  - For security, “the right thing” is the only thing!

Today

- Welcome to COS-518!

1. Goals and high-level topics

2. Course administrivia

3. **Systems design**
   - Worse is Better
     - Richard P. Gabriel (known for Common Lisp)

   - **Lampson's "Hints for Computer System Design"**
     - **Butler Lampson (Turing, MSFT Fellow, Alto, 2PC, …)**
     - **SOSP 1993 conference**

Systems versus algorithms

- Computer **systems** **differ from** **algorithms**
  - External interfaces are less precisely designed, more **complex,** more **likely to change**
  - Much **more internal structure**, interfaces
  - Measure of success much less clear
- And, principles of computer system design are much **more heuristic, less mathematical**

Interfaces

- Most of Lampson's hints depend on notion of ***interface***
  - Separates ***clients*** of an abstraction from the ***implementation*** of that abstraction

- **Defining interfaces** is the most important part of system design

- Interfaces should be:

1. Simple
2. Complete
3. Admit a sufficiently small and fast implementation

Keep it simple

- In other words, follow the New Jersey approach:
- Do **one thing at a time,** and do it well
- **Don't generalize:** generalizations are usually wrong
  – Generalization leads to unexpected complexity
- Interface **mustn't promise more** than the implementation knows how to deliver

Continuity

- *As a system changes, how do you manage change?*
- **Keep basic interfaces stable**
- If you do change interfaces, **keep a place to stand**
  – ***Compatibility package*** (a.k.a. ***shim layer***) implementing old interface atop new interface

Implementation

- **Plan to throw one away (you will anyhow)**
  - Brooks' observation in The Mythical Man-Month
  - It pays to revisit old design decisions with the benefit of hindsight

- **Keep secrets** of the implementation
  - **Assumptions** about the implementation that clients are **not allowed** to make
    - In other words, things that **can change**

- Instead of generalizing, **use a good idea again**

Handling all the cases

- Handle **normal** and **worst** cases **separately:**
  - The **normal** case **must be fast;**
  - The **worst** case must **make some progress**

A possibly-missing hint:

- **Use indirection**
  – Go through an intermediary to an object

- Examples:
  – **Virtual memory**
  – Compiler's **intermediate representation** (between high-level and machine languages)
  – We'll see another example when we discuss **System R** (Lecture 3)

For next time…

- **Today:** Read S&K assigned reading, "Worse is Better" and Lampson's "Hints"

- **Monday 9/21** paper discussion:
  - The Log-Structured File System
  - Plan 9 Operating system

- **Excellent papers,** so an **opportunity:** Sign up to present on Monday by emailing TA today
  - **Mandatory: Everyone sign up to review** one of the two papers by the **end of the day today**
    - If no volunteers, we will randomly assign a presenter tomorrow morning!