Where to Go from Here

Announcements

- Office hours Today and Tomorrow:
  - 12-5PM in Gates 160

Goals for this Course

- **Learn how to model and solve complex problems with computers.**
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.

How do we model real-world problems in software?

We're going to need a way to model **sequences of data**.

So let's study stacks, queues, and vectors.

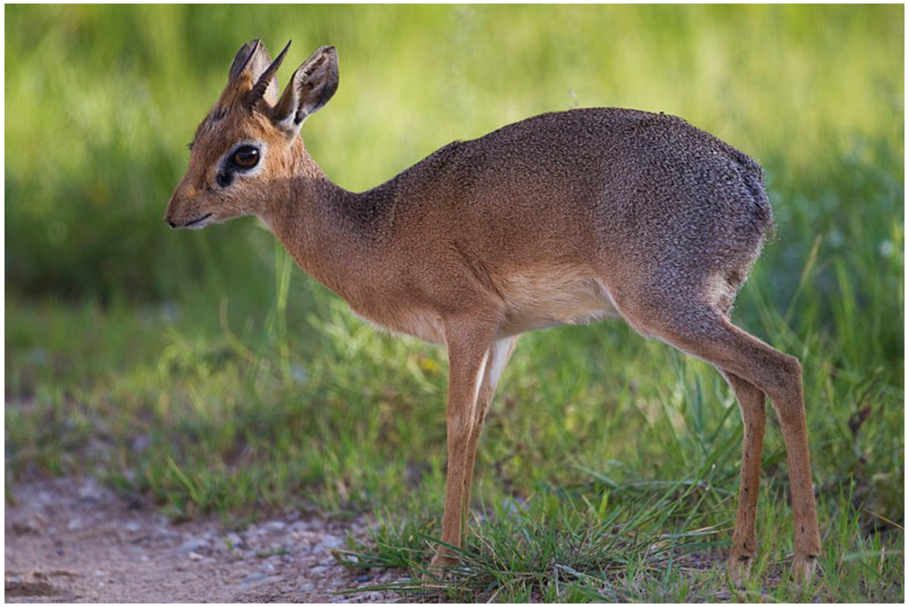
15 - 3 + 4 * 5 / 6

Buying Cell Towers

137 42 95 272 52

What about associative data?
Or unordered data?

This is where maps, sets,
and lexicons come in.



dik·dik

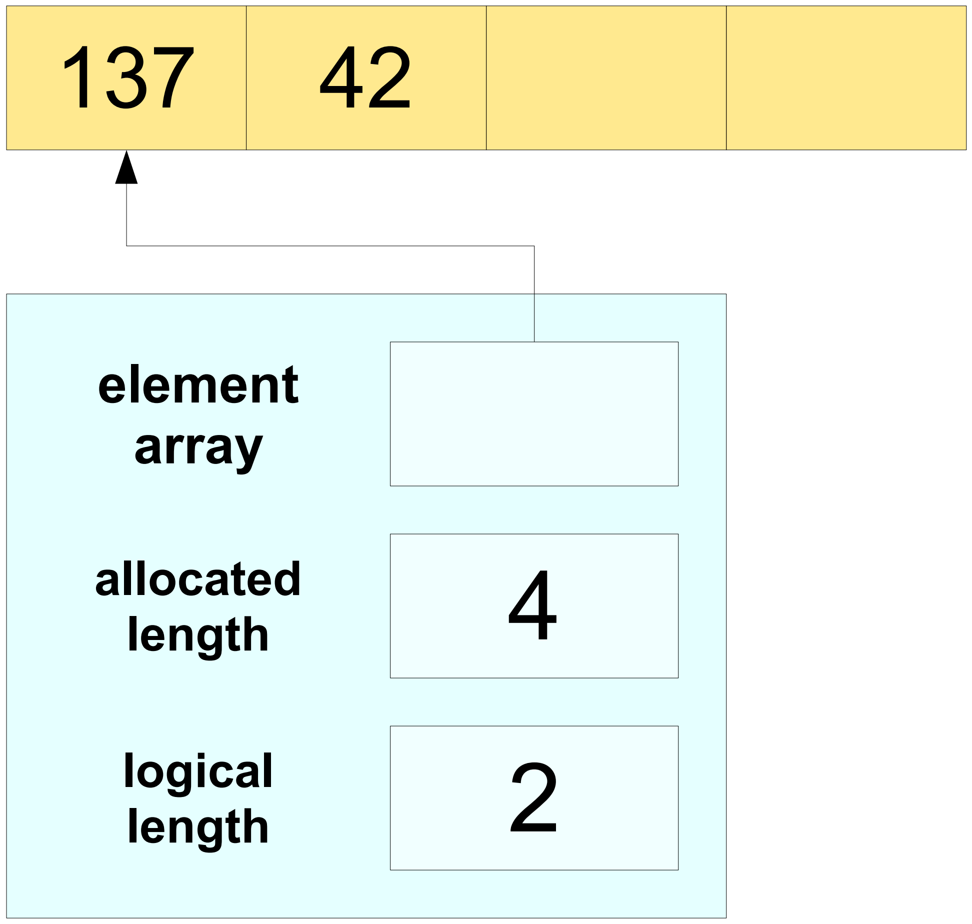
```
opts
post
pots
spot
stop
tops
```

How do we model **networks**?

We can use graphs, which we can build out of those tools we just saw!

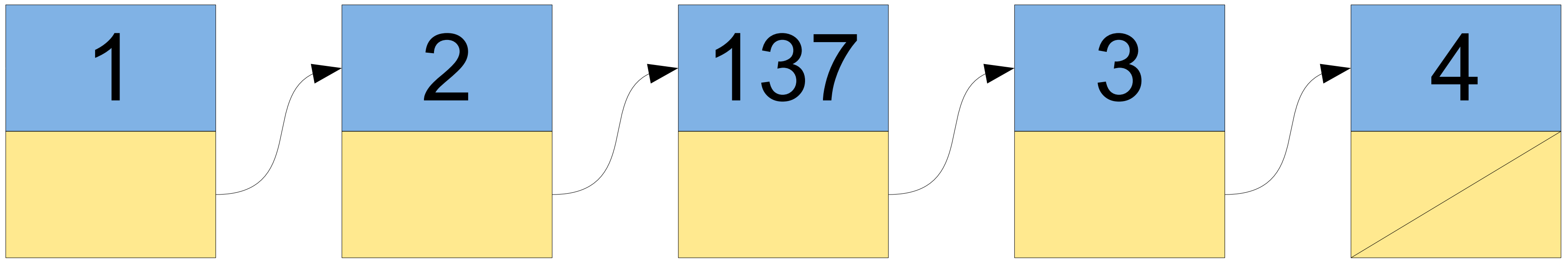
But how does all of this work?

Idea



Linked Lists at a Glance

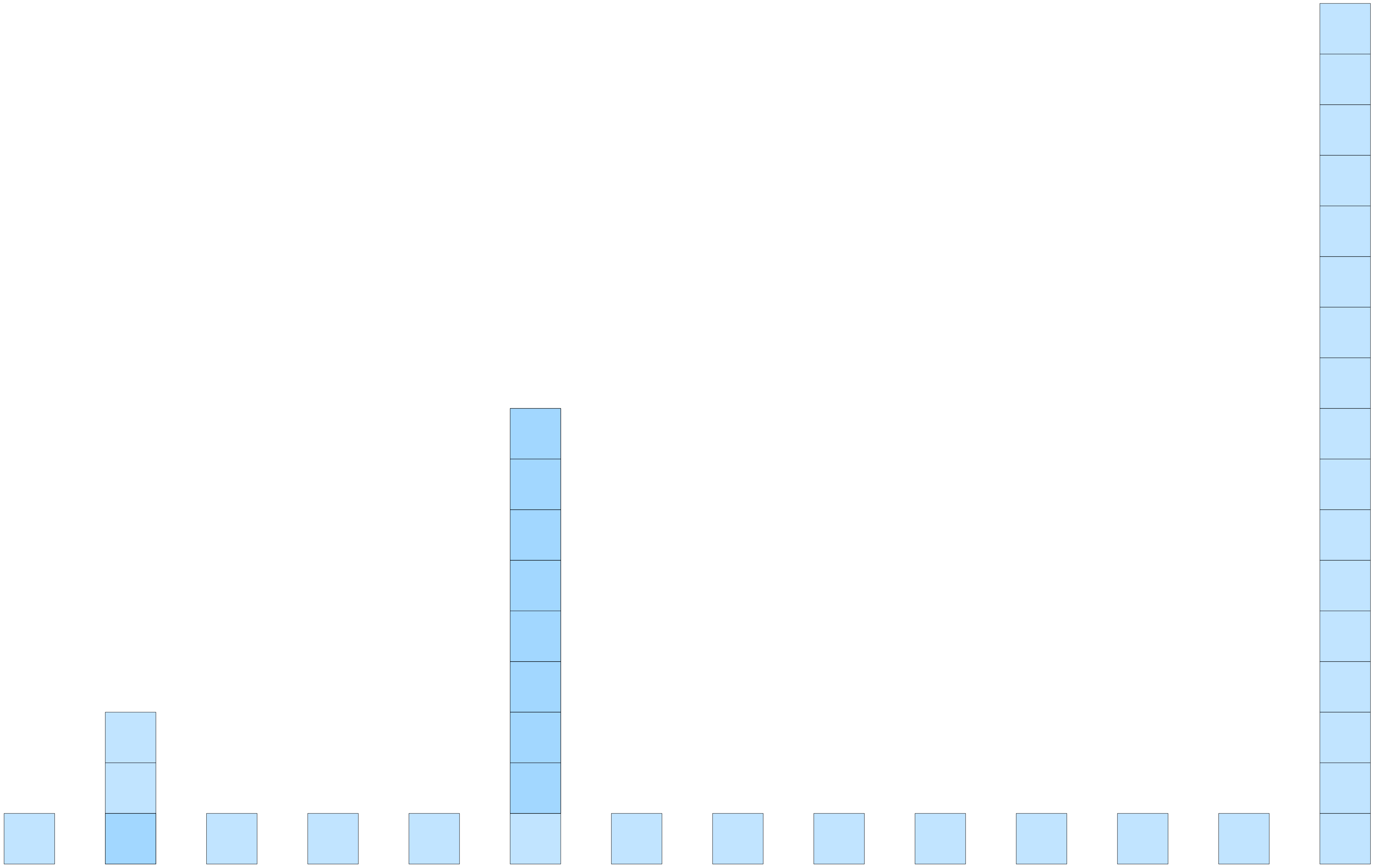
- A **linked list** is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.

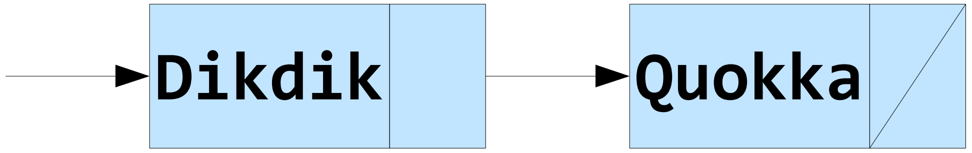


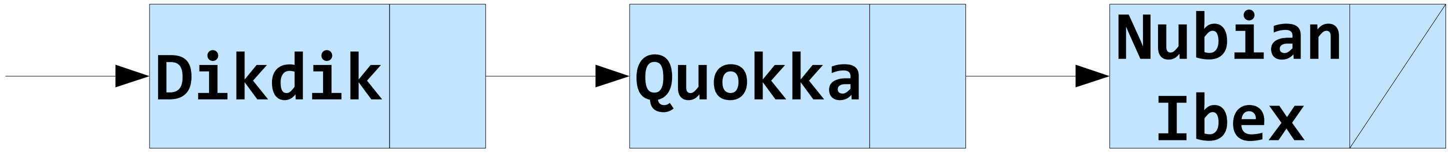
We need a way to compare solutions.

So let's invent big-O notation.

So… how do they compare?

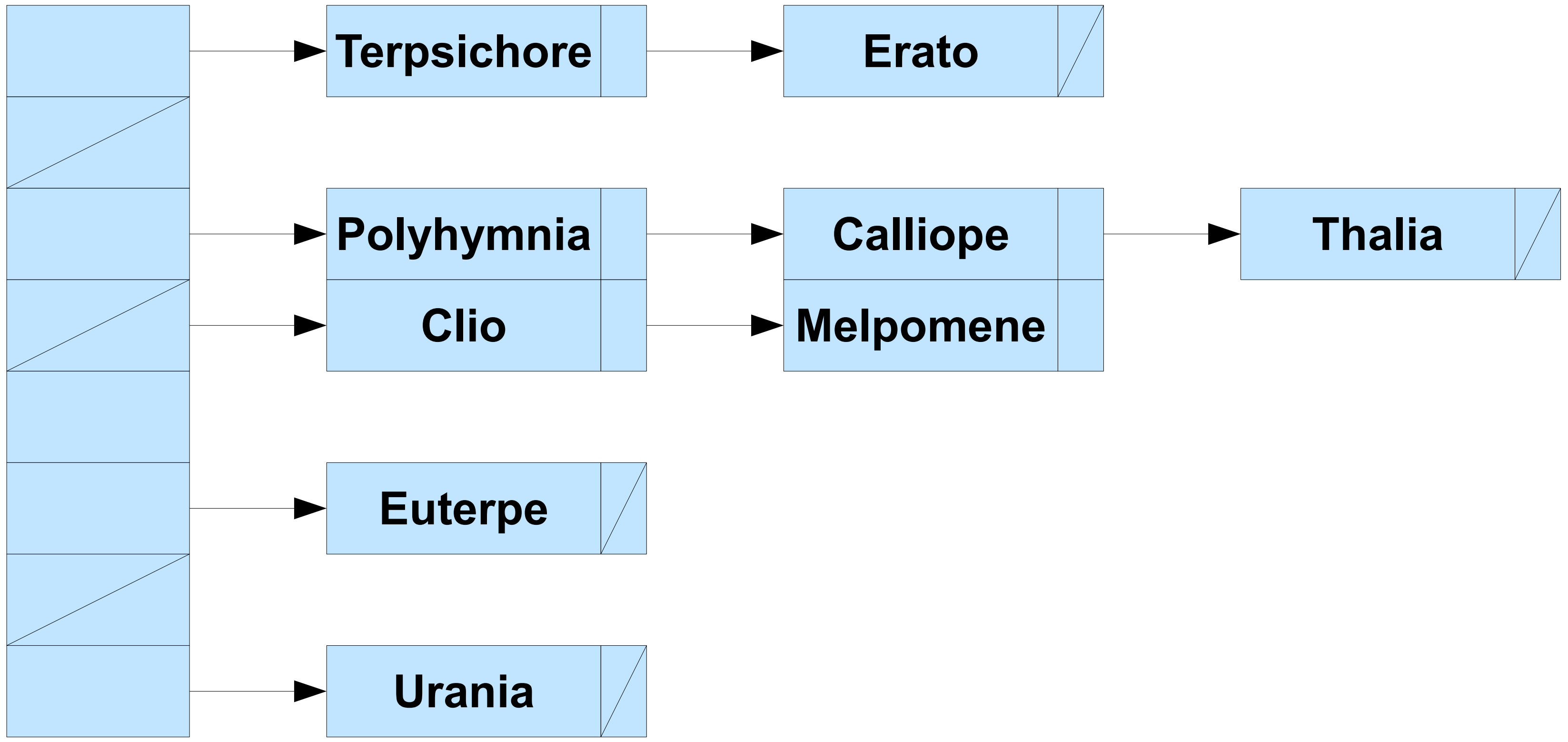


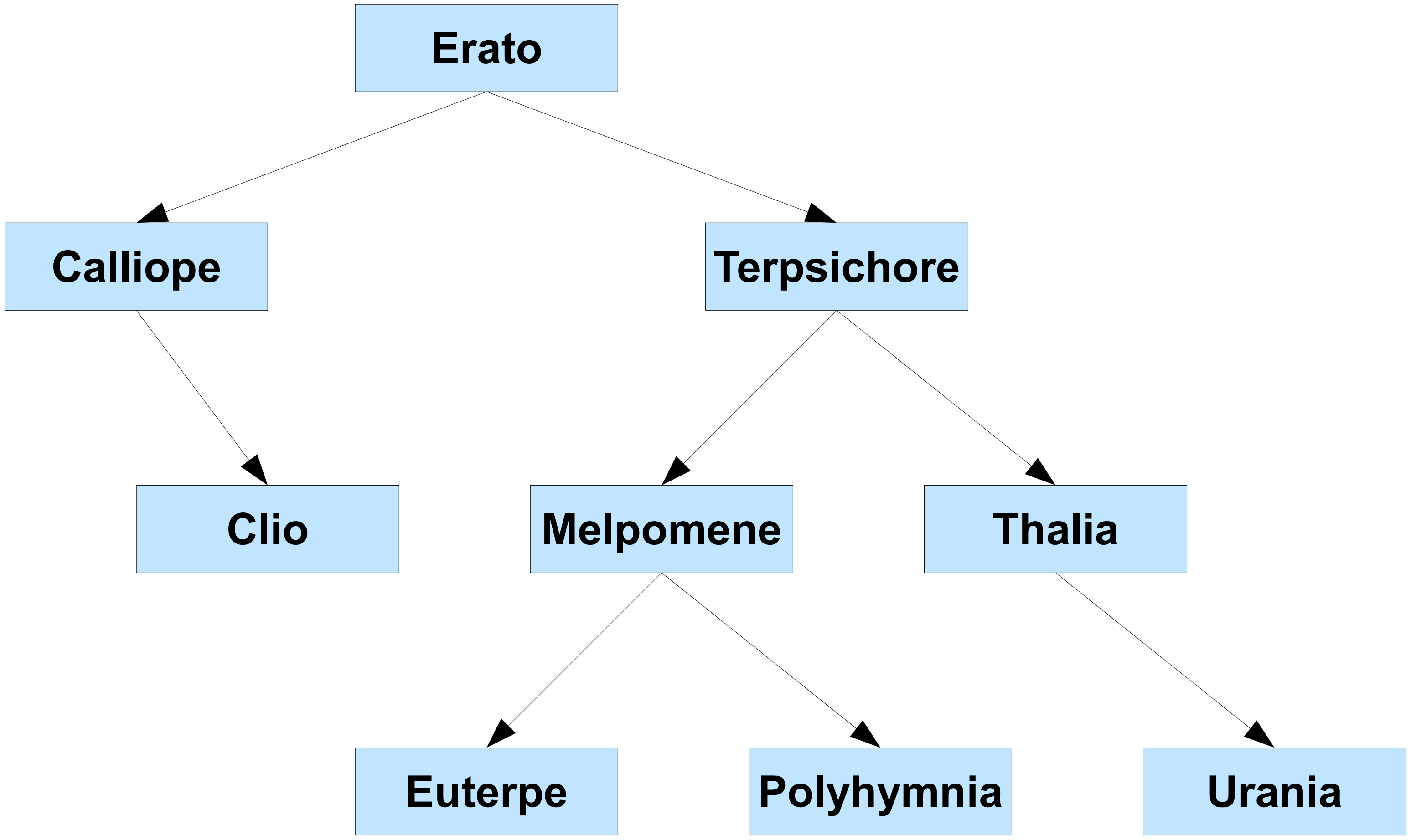

Dikdik
Quokka


Dikdik
Quokka
Nubian Ibex

Neither of the implementations is strictly better than the other!

How might we make maps and sets?


Terpsichore
Erato
Polyhymnia
Calliope
Thalia
Clio
Melpomene
Euterpe
Urania


Erato
Calliope
Terpsichore
Clio
Melpomene
Thalia
Euterpe
Polyhymnia
Urania

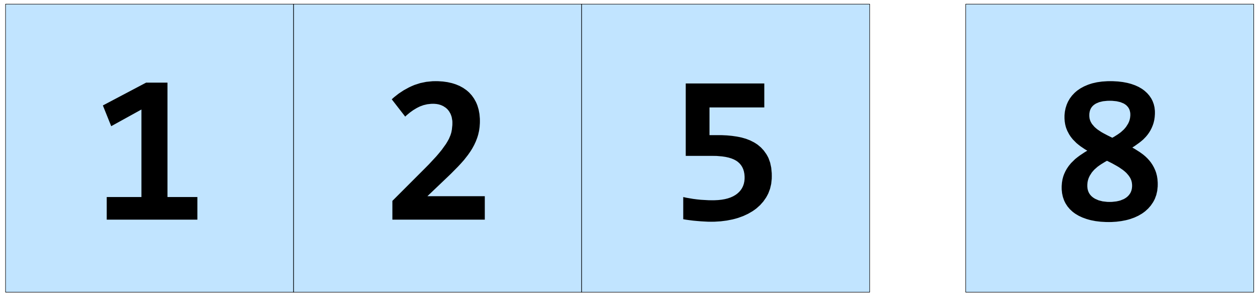
As before, neither of these structures is clearly better than the other!

This hits on a key point:
Engineering is all about trade-offs.

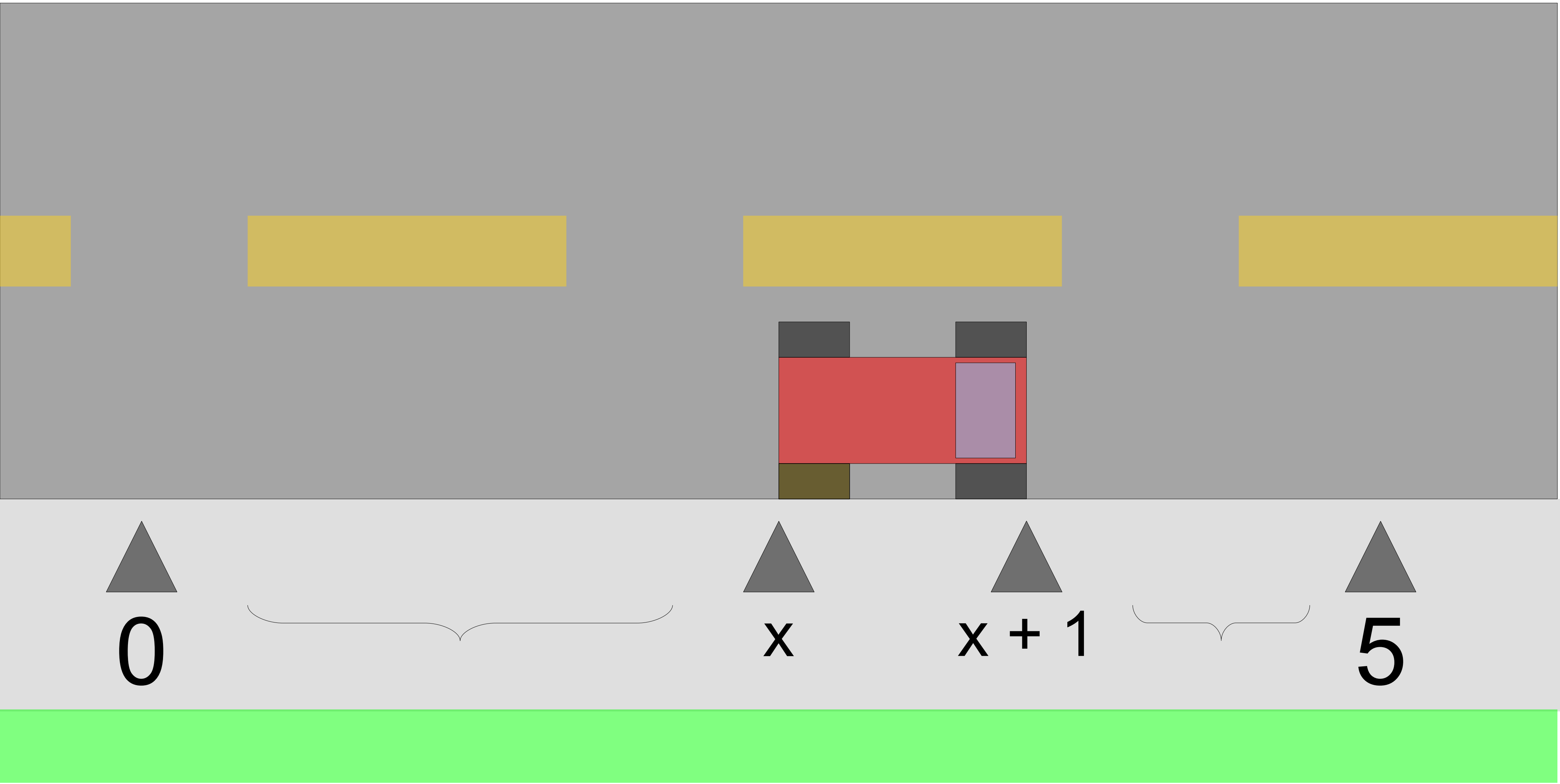
Cool! Now we have a bunch of tools for modeling problems!

So how do we go about solving them?

1	2	5	8


1
2
5
8

Parking Randomly



Nifty! Some problems are self-similar!

So how do we solve them?

```
int sumOfDigits(int n) {
  if (n < 10)
      return n;
  else
      return (n % 10) + sumOfDigits(n / 
10);
}
```

```
void moveTower(int n, char from,
               char temp, char to) {
    if (n > 0) {
        moveTower(n - 1, from, to,
temp);
        moveDisk(from, to);
        moveTower(n - 1, temp, to,
from);
    }
}
```

All we need to do is figure out a base case and a recursive step!

What else can we do with recursion?

```
{ 1, 2, 3 }
{ 1, 2,   }
{ 1,    3 }
{ 1       }
{    2, 3 }
{    2    }
{       3 }
{         }
```

Self-similarity is **everywhere**!

Programming is all about exploring new ways to model and solve problems.

The skills you have just learned will follow you through the rest of your programming career.

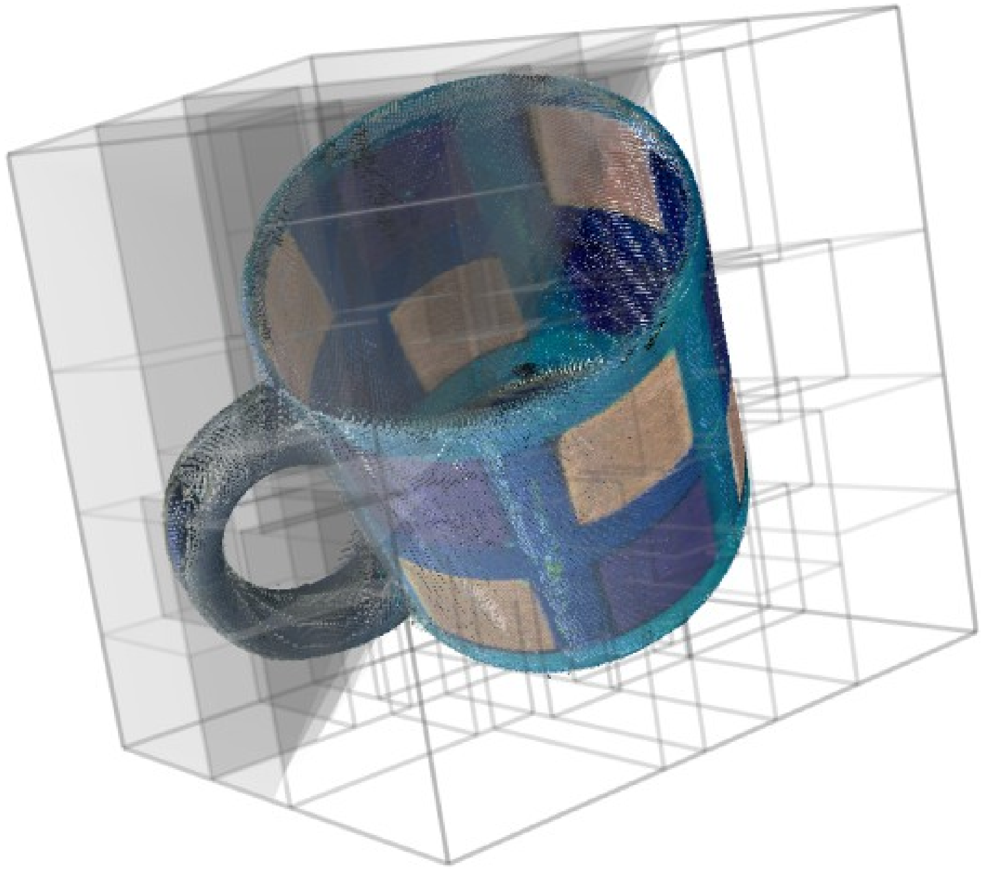
Also, the concepts learned in this class come up *everywhere* in Computer Science.

The material you've learned opens up a *huge* number of new topics to learn!

Trees

Trees

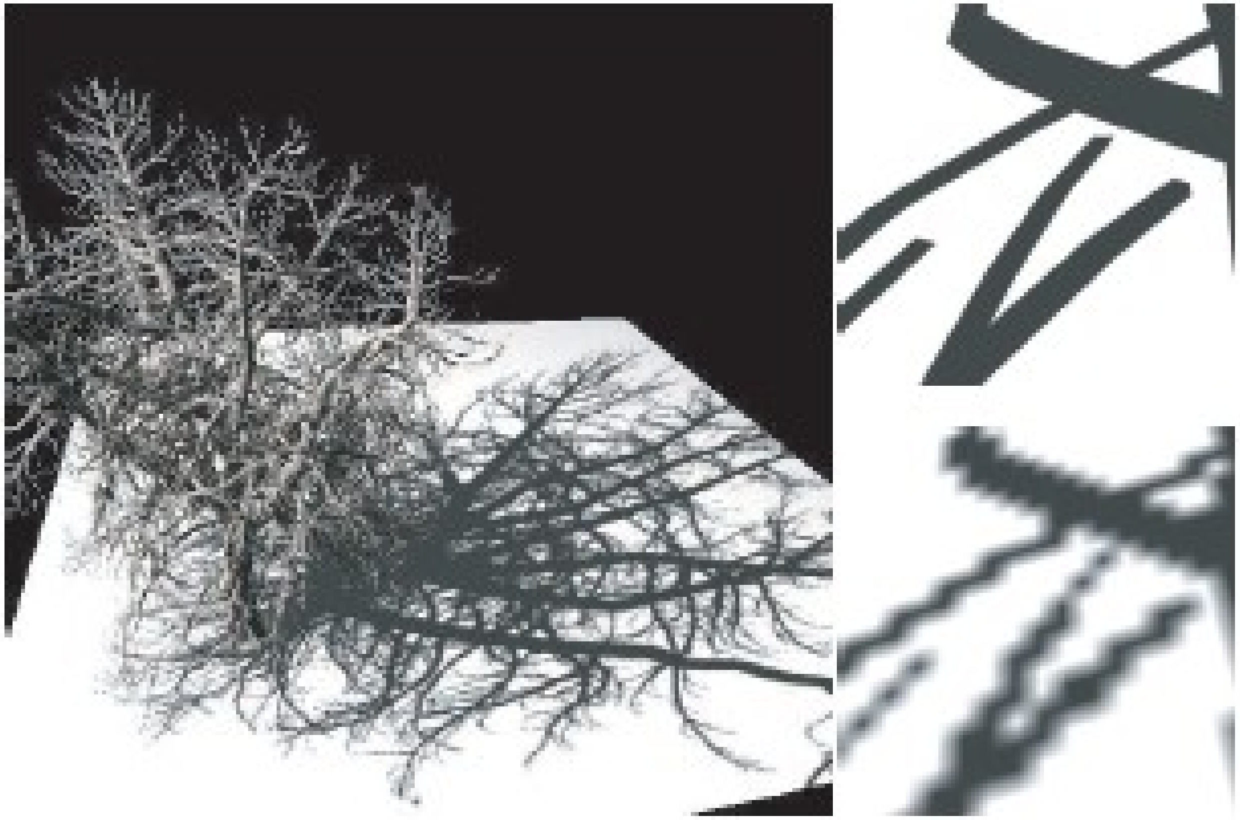
- Trees + Computational Geometry = k-d Trees



http://docs.pointclouds.org/trunk/group__kdtree.html

Trees

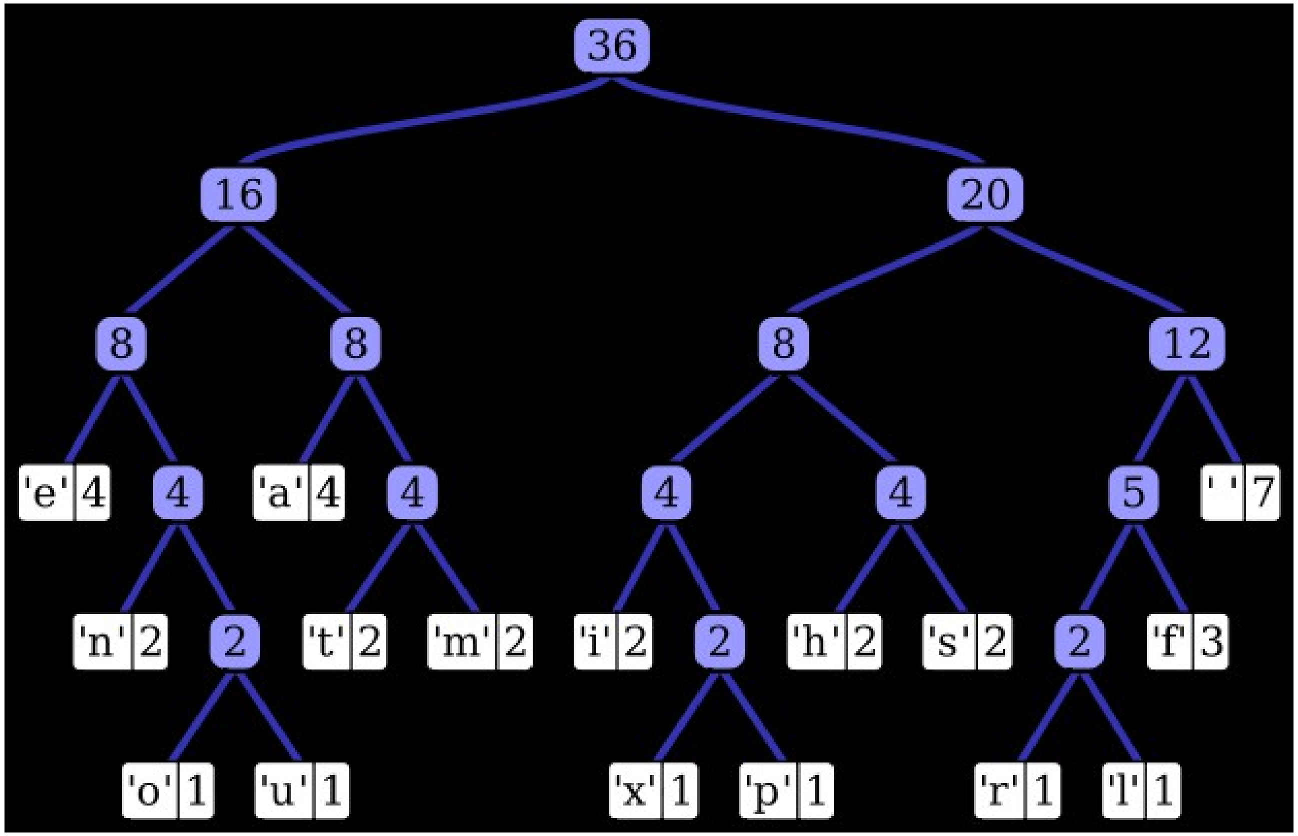
- Trees + Graphics = Adaptive Shadow Maps



http://www.cs.utah.edu/~jmk/images/dynamicShadowMaps.jpg

Trees

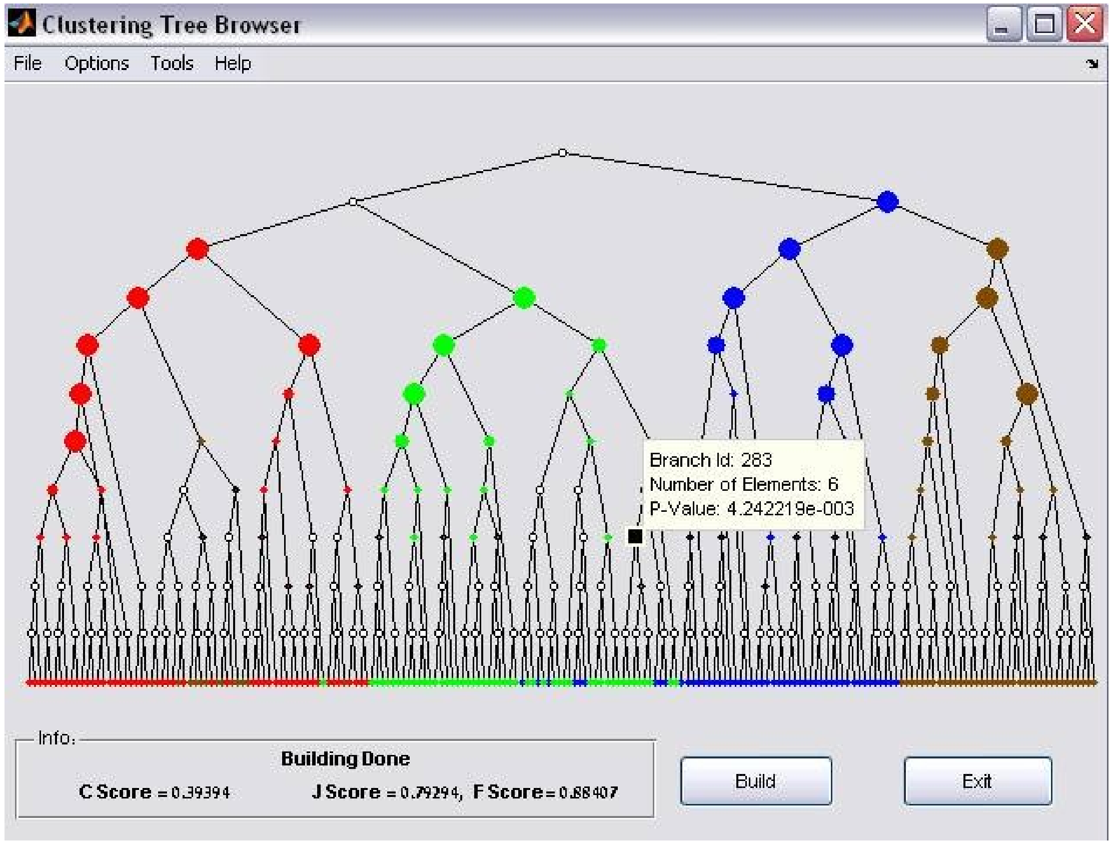
- Trees + Information Theory = Huffman Encoding



http://en.wikipedia.org/wiki/File:Huffman_tree_2.svg

Trees

- Trees + Machine Learning = Hierarchical Clustering



Branch Id: 283
Number of Elements: 6
P-Value: 4.242219e-003

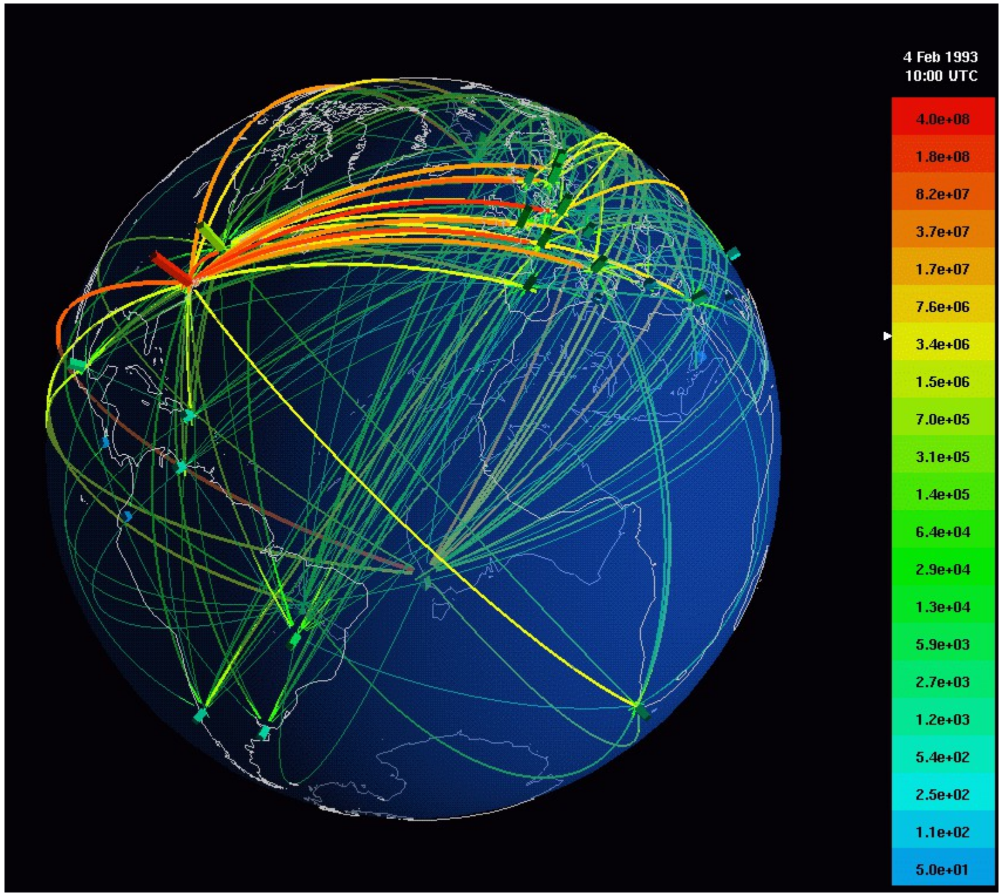
Info:
Building Done
C Score = 0.39394 J Score = 0.79294, F Score = 0.88407

http://adios.tau.ac.il/clustree/Clustree_files/image014.jpg

Graphs

Graphs

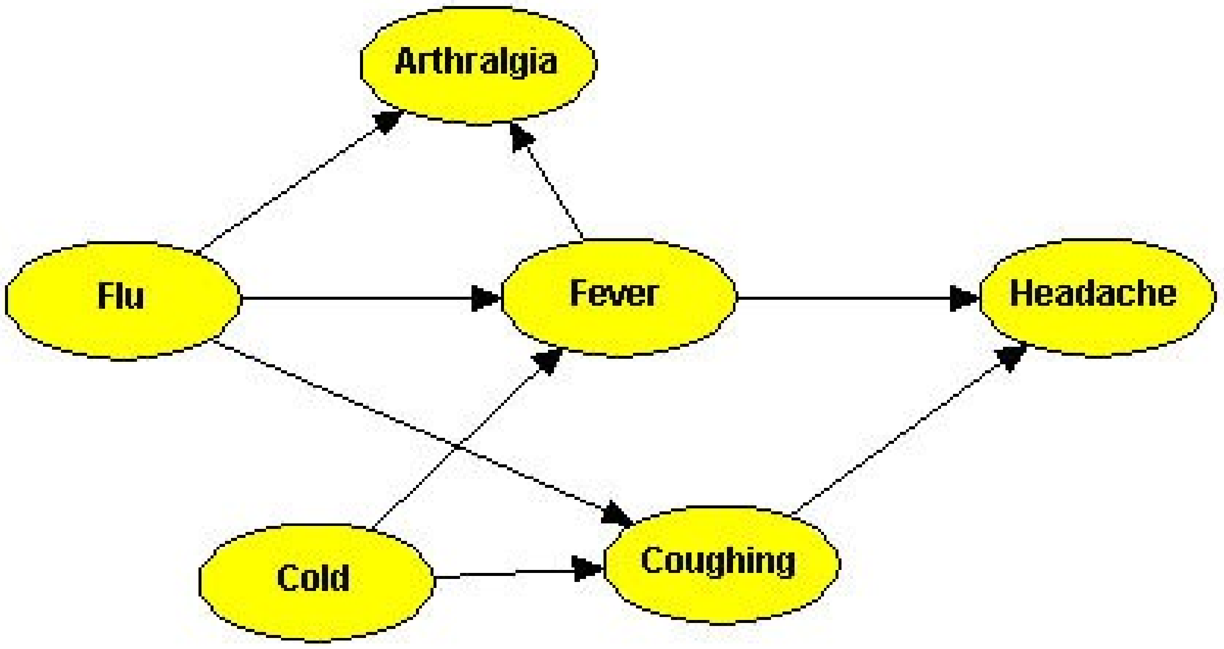
- Graphs + Networking = Computer Networks



http://icawww1.epfl.ch/cn2/0910/

Graphs

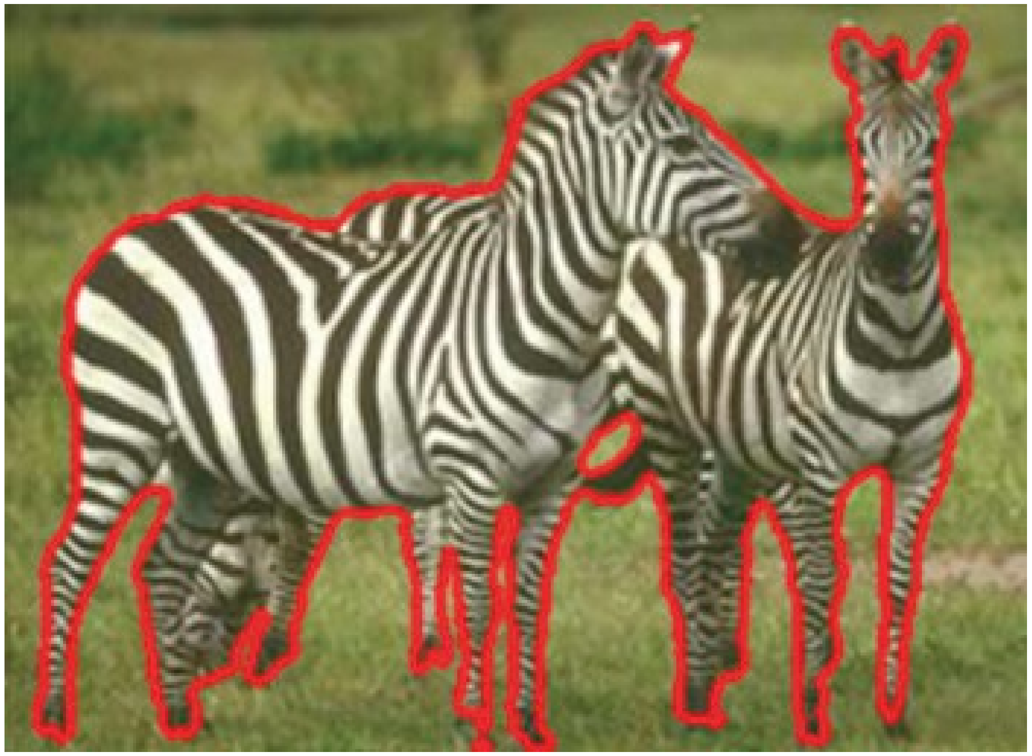
- Graphs + Machine Learning = Probabilistic Graphic Models



http://song.bayesian.net/index.php/Bayesian_net

Graphs

- Graphs + Computer Vision = Image Segmentation

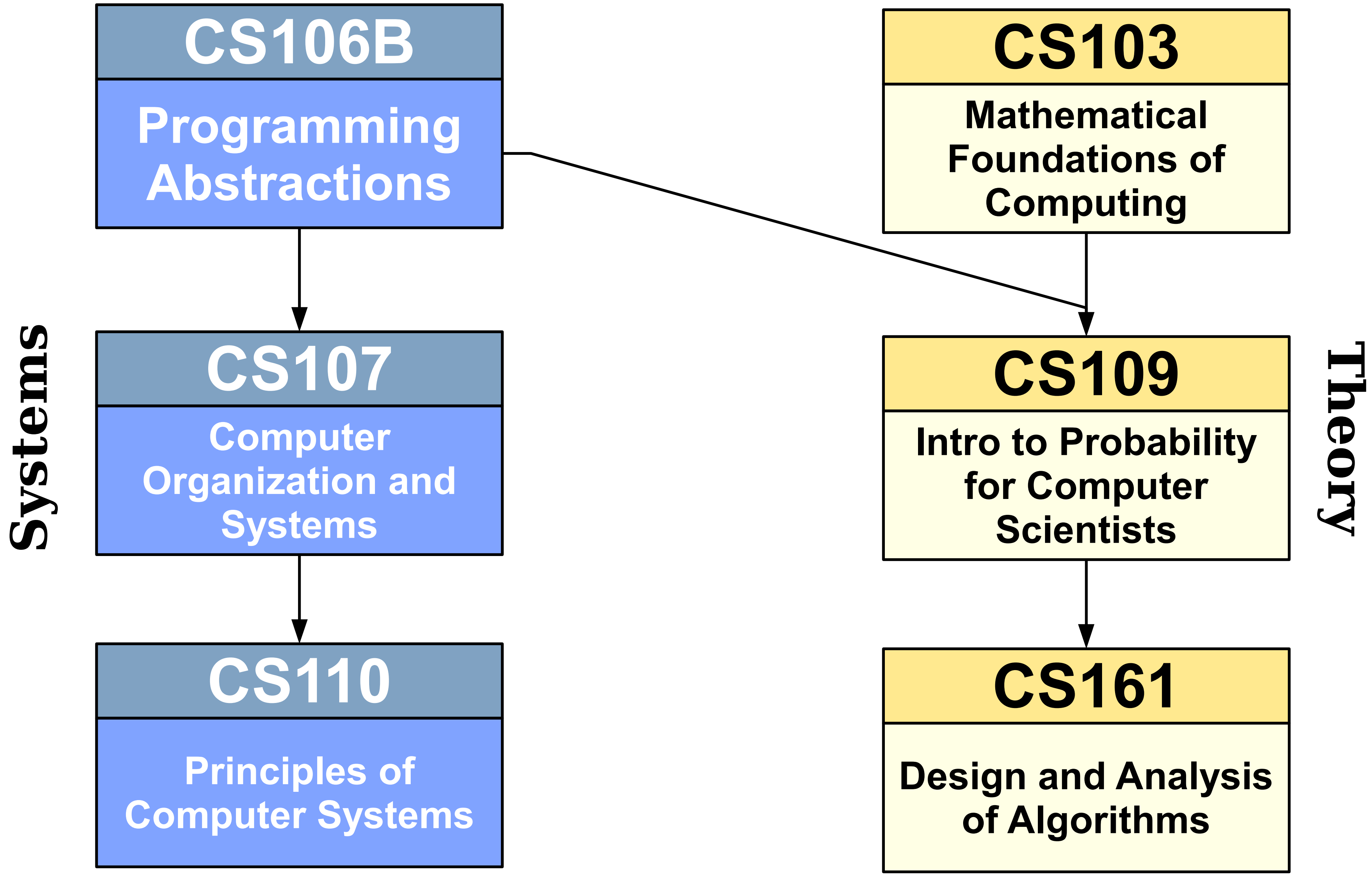


http://ailab.snu.ac.kr/intro.php

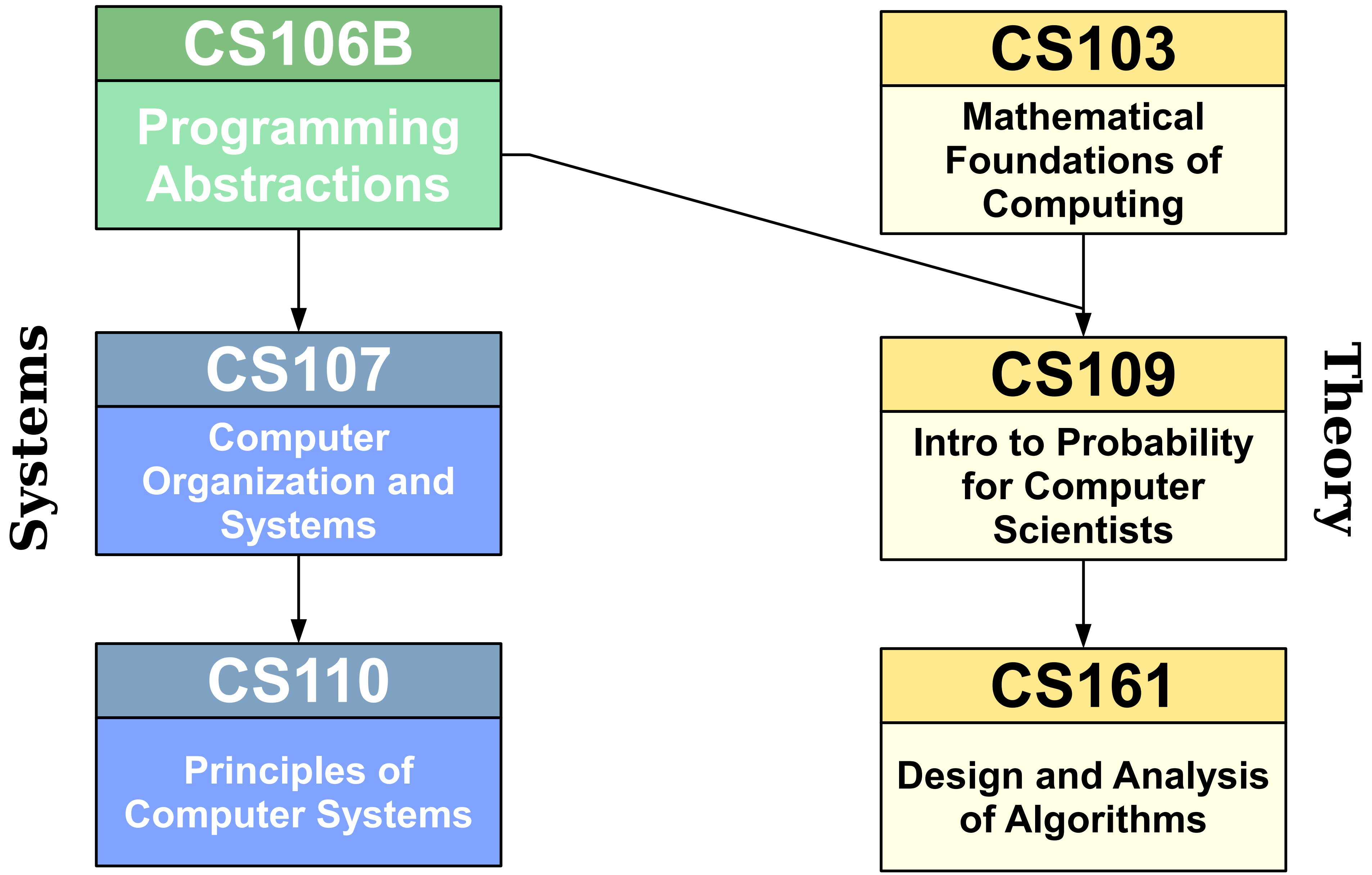
So what comes next?

Courses to Take

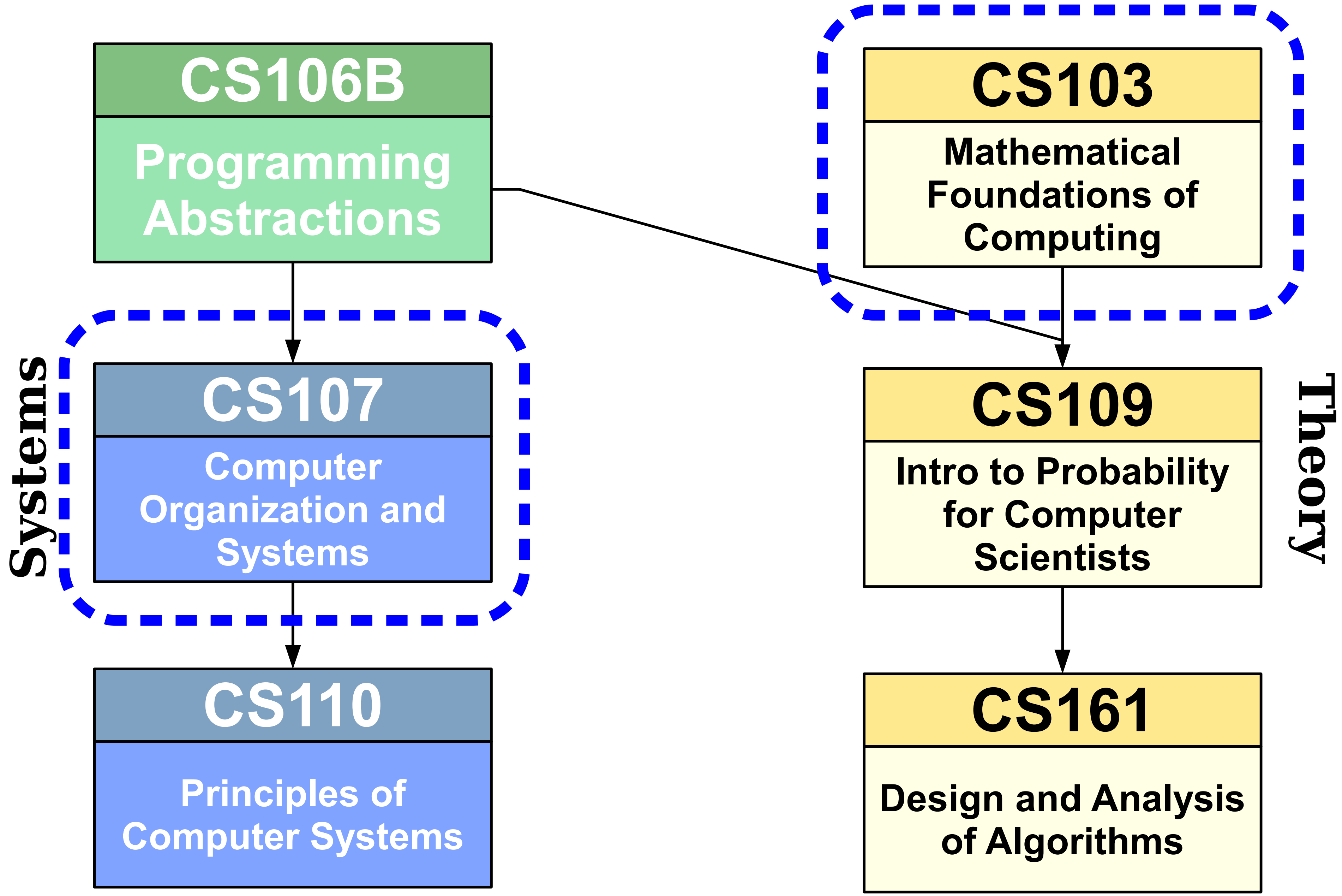
The CS Core


Systems
CS106B
Programming Abstractions
CS107
Computer Organization and Systems
CS110
Principles of Computer Systems
Theory
CS103
Mathematical Foundations of Computing
CS109
Intro to Probability for Computer Scientists
CS161
Design and Analysis of Algorithms

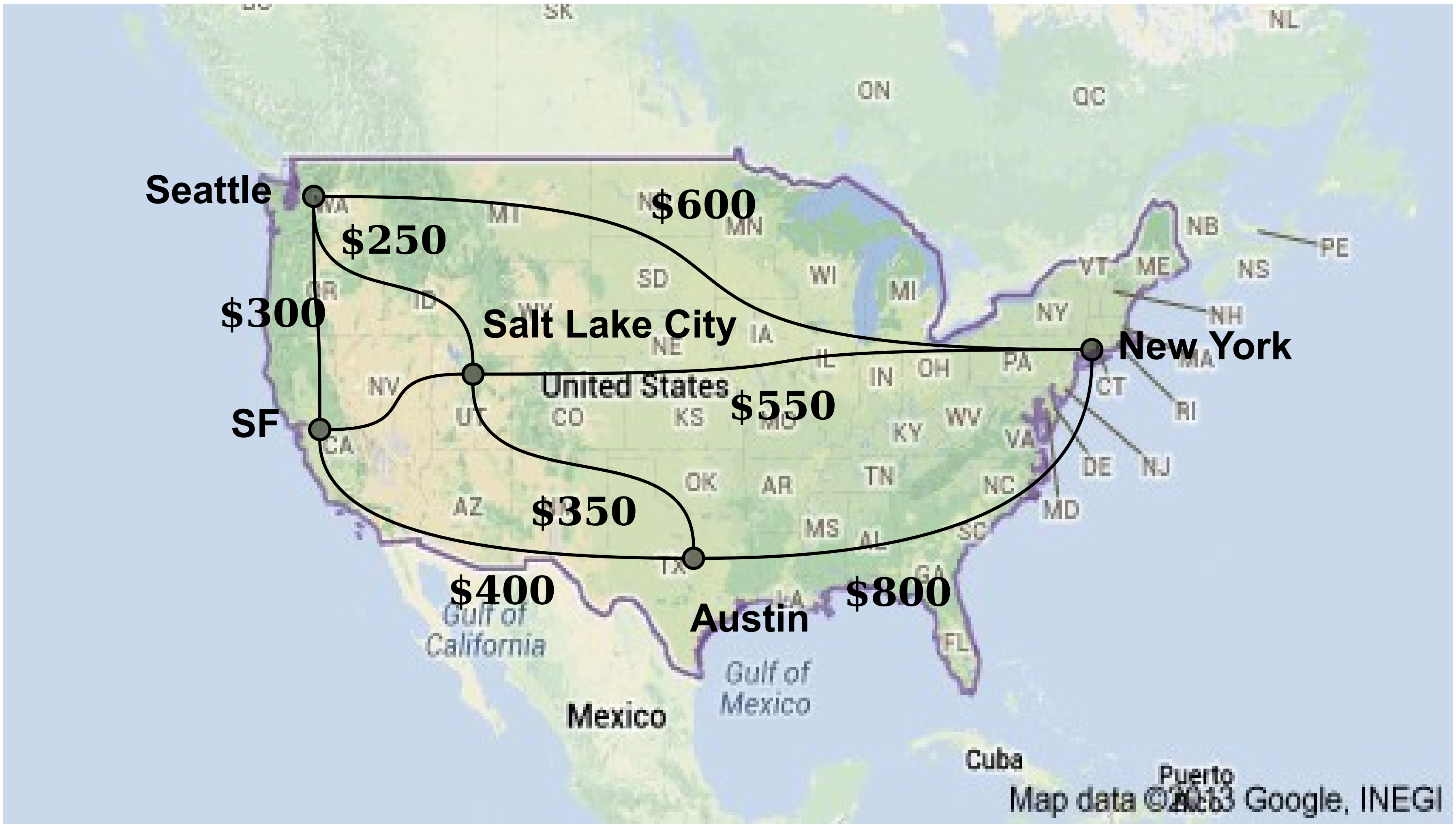
The CS Core


CS106B
Programming Abstractions
CS103
Mathematical Foundations of Computing
Systems
CS107
Computer Organization and Systems
CS109
Intro to Probability for Computer Scientists
Theory
CS110
Principles of Computer Systems
CS161
Design and Analysis of Algorithms

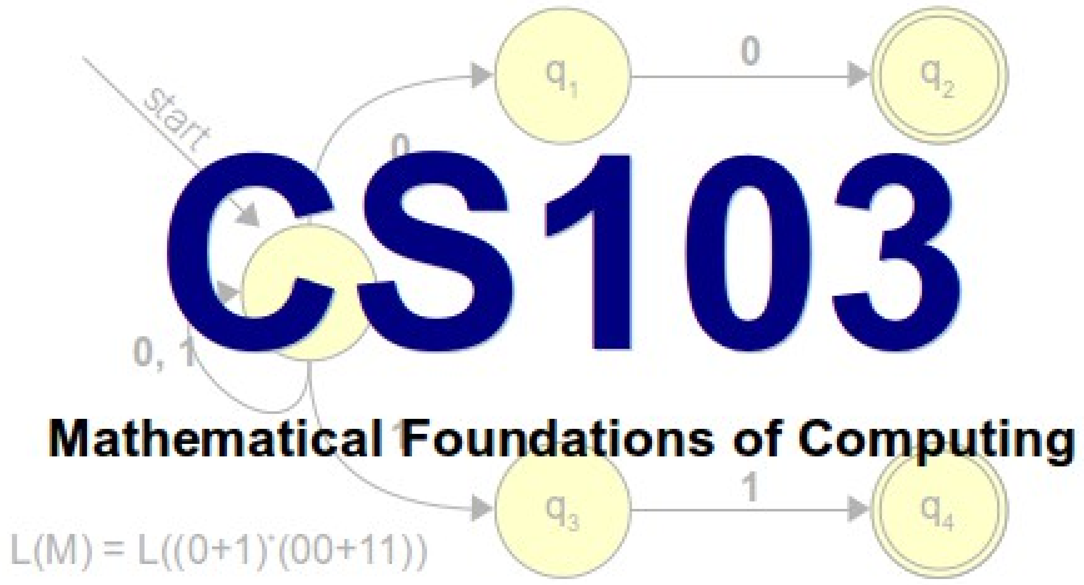
The CS Core


CS106B
Programming Abstractions
CS107
Computer Organization and Systems
CS110
Principles of Computer Systems
CS103
Mathematical Foundations of Computing
CS109
Intro to Probability for Computer Scientists
CS161
Design and Analysis of Algorithms
Systems
Theory

Traveling Salesperson



https://www.google.com/maps/vt/data=VLHX1wd2Cgu8wR6jwyh-km8JBWAkEzU4,2bUCUBVs3YYr-KB4ccFl-1Q1nWYcyKzmW0Ggf8ar4OOyEuuN9txRnTiKzlvmH6qy6B4vSoZvopndG7VjMIsOIDayhdkqKblOykP1wZYm9RcF8-Y6pkecPwDi3xc98B3gNGLchfR7xnPKzCGEmRocrv9OczmELzORvRseZHLyjWOvL0GzUeg0WFJGA4Y



CS103

Mathematical Foundations of Computing

Can computers solve all problems?

Why are some problems harder than others?

How can we be certain about this?

It's all Bits and Bytes!

K	01001011
I	01001001
R	01010010
K	01001011
'	00100111
S	01010011
	00100000
D	01000100
I	01001001
K	01001011
D	01000100
I	01001001
K	01001011

It's all Bits and Bytes!

K	01001011
I	01001001
R	01010010
K	01001011
'	00100111
S	01010011
	00100000
D	01000100
I	01001001
K	01001011
D	01000100
I	01001001
K	01001011

```
01001011010010010101001001001011
00100111010100110010000001000100
01001001010010110100010001001001
01001011
```

CS107
Computer Organization and Systems

How do we encode text, numbers, programs, etc. using just 0s and 1s?

Where does memory come from? How is it managed?

How do compilers, debuggers, etc. work?

What CS107 Isn't

- CS107 is ***not*** a litmus test for whether you can be a computer scientist.
  - You can be a *great* computer scientist without enjoying low-level systems programming.
- CS107 is ***not*** indicative of what programming is "really like."
  - CS107 does a lot of low-level programming. You don't have to do low-level programming to be a good computer scientist.
- CS107 is ***not*** soul-crushingly impossibly hard.
  - It's hard. It does not eat kittens.
- *Don't be afraid to try CS107!*

Other CS Courses

CS108

Object-Oriented Systems Design

- ***How do you build large software systems in a team?***
- Introduction to
  - Unit-testing frameworks
  - Object-oriented design.
  - Multithreaded applications.
  - Databases and web applications.
  - Source control.
- Excellent if you're interested in learning industrial programming techniques.

CS193

- Many offerings throughout the year, focused on specific technologies:
  - CS193A: Android Programming
  - CS193C: Client-Side Web Technologies
  - CS193I: iOS Programming
  - CS193L: Lua Programming
  - CS193P: iPhone and iPad programming
- Great for learning particular technologies.

CS147

Intro to Human-Computer Interaction

- ***How do you design software to be usable?***
- ***What are the elements of a good design?***
- ***How do you prototype and test out systems?***

The CS Minor

The CS Coterm

Outside Stanford

Learning More

- MOOCs: Coursera, Udacity, and edX all offer CS courses.
    - e.g. Check out Udacity's Web Development course at https://www.udacity.com/course/cs253.
- Explore and play around! There are great resources for getting unstuck on the internet.
    - e.g. Stack Overflow (http://www.stackoverflow.com) is a place to ask for help on programming-related questions.
- Read a book! There are great books that introduce most programming languages and frameworks.

Taking Classes

- Want to come back to Stanford? Take classes through SCPD!

My Email Address

adgress@cs.stanford.edu

Some Words of Thanks