# Wrapping Up CS 256

We Did It!

# Admin:  Last Lecture

- **Course evaluations** in second half of class today
- Will post HW 10 (solution sketch) on GLOW later today
- Help hours before the final:
    - Office hours today 1.30- 3pm, TA hours rest of the day
- **Final**.  24-hour final, will be available on Gradescope from **Thursday May 20 8.30 am** and must be submitted by **May 28, 8.30 pm**
- **Honor Code (final).**
    - Can only refer to course materials (notes/book/GLOW), honor code violation to use external resources, google terms, or discuss exam with others

# Thesis Presentations:  Tomorrow

**10:00 am:  David Lee**, "A Practical Adaptive Quotient Filter"

**10:45 am:  Markus Feng**, " Shard:  Shell for Ad Hoc, Reliable, Distributed Programs"

---Break---

**1:00 pm:  Andrew Thai,**  "A Filtering Approach for NFA Processing on GPUs"

**1:45 pm:  Peter Zhao**,  "Time-Space Tradeoffs in Indexing Problems via Reductions to Function Inversion

# Algorithmic Design Paradigms

- Graph Traversals
  - Can do a lot with just BFS and DFS!
- Greedy Algorithms
  - Simple solution/ hard to show why they work
- Divide & Conquer and Recurrences
  - Using recursion as a powerful design technique
- Dynamic Programming
  - Exponential-seeming problems with polynomial time recursive solutions
- Network Flows
  - Many complicated looking optimization problems in disguise!
- Randomized Algorithms
  - Simple, elegant algorithms that work really well!
- Approximation Algorithms
    - Worst-case guarantees for intractable problems

# Problem Solving Strategies

- How to make an abstract problem concrete
- How different parts of the problem commute/compose together
- **Problem reduction**:  recognizing one problem in another
  - Reduce an instance of Problem **A** to Problem **B**
  - Use algorithm for Problem **B** to solve it
  - Why it works (necessary and sufficient conditions)

**Biggest Takeaways:**

- Learning to think algorithmically
- How to formalize your intuition

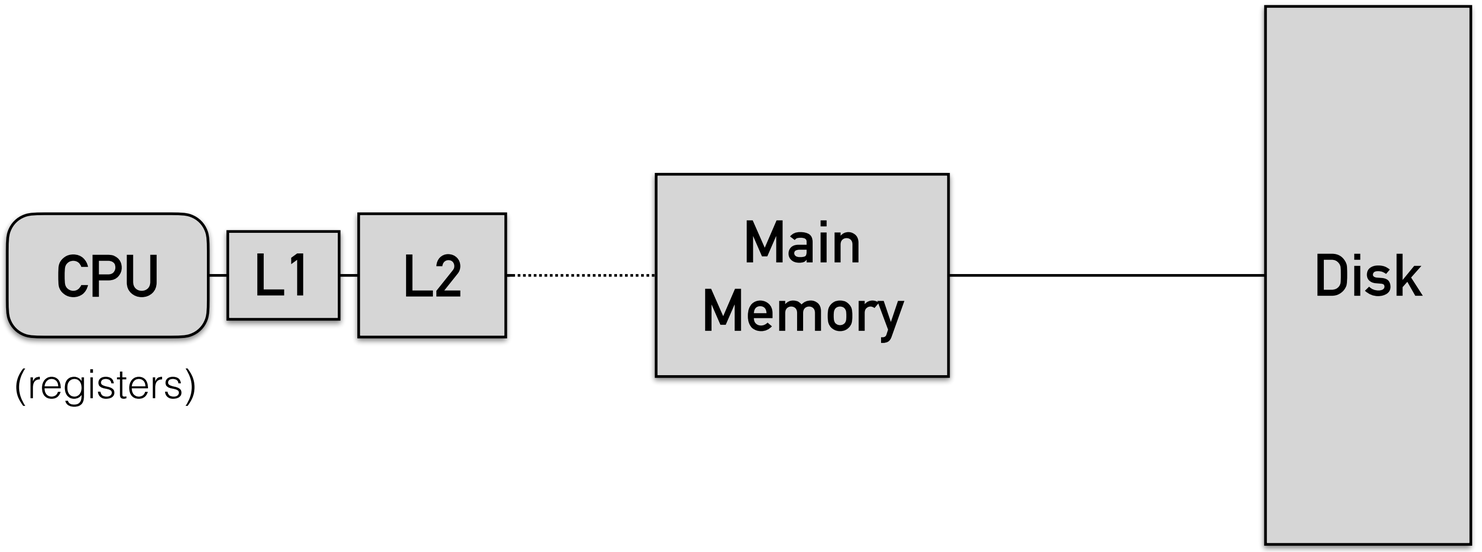
# Flip Side of Design: Intractability

- How do we draw boundaries between what is computable and what is impossible
- No matter how much we'd want it, drawing these boundaries isn't easy
- Brings up the famous P versus NP question
- When we can’t prove something is unconditionally hard, we prove it conditioned on some conjecture we all believe is not true
  - SAT/ 3SAT, Independent Set, Vertex cover
  - Set cover, Hamiltonian Cycle/Path
  - Subset Sum, Graph Coloring, Clique
- Shows the power of mathematical characterization
- ***If you are CS major, you will see these again in CS 361***

# Other Models of Computation

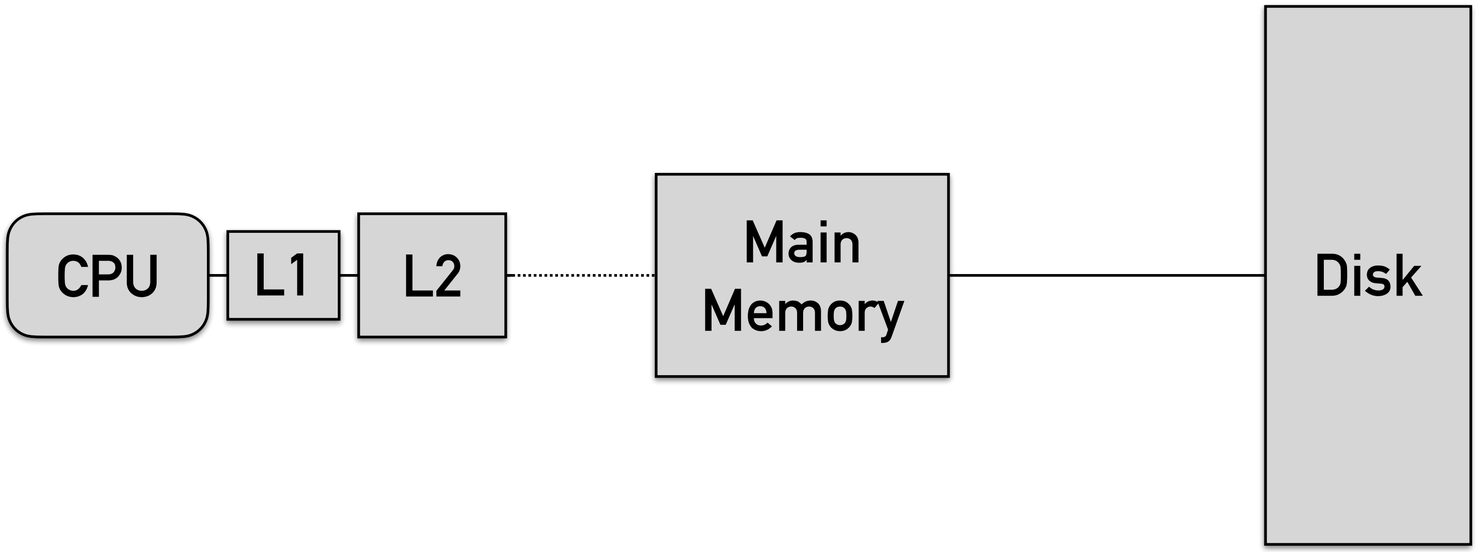
Aka: Applying what we've learned in practice

# Modern Memory Hierarchy




- $\approx$ 10K $\rightarrow$ 100K $\rightarrow$ MBs $\rightarrow$ GBs $\rightarrow$ TBs
- $\approx$ ns $\rightarrow$ 10ns $\rightarrow$ 10ns $\rightarrow$ $\mu$s $\rightarrow$ 10ms
- (Left to right). Bigger but slower latency (distance travelled and physical seek time on disk)
- Bandwidth (amount of data that can transferred) usually matched

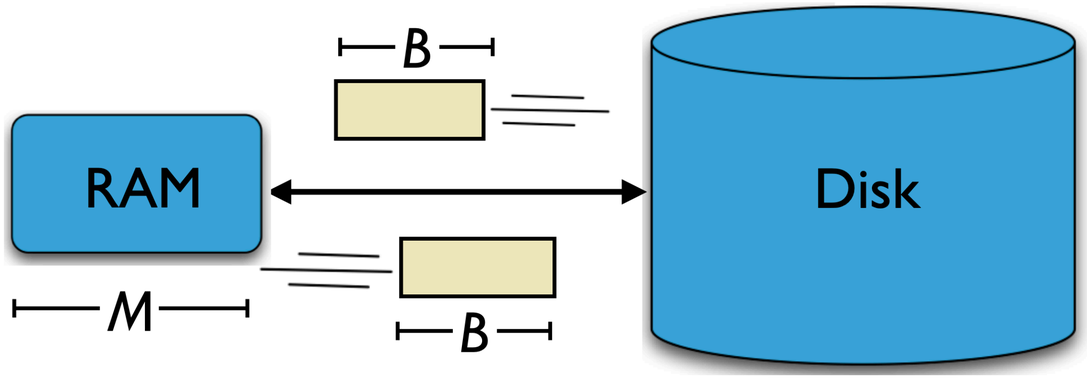
# Modern Memory Hierarchy




- **Blocking.** Idea to mitigate latency.
  - When fetching data, get an entire block containing it
- Cache-efficient algorithms have good locality
  - Use items in same block or nearby blocks (spatial locality)
  - Reuse blocks already in cache (temporal locality)

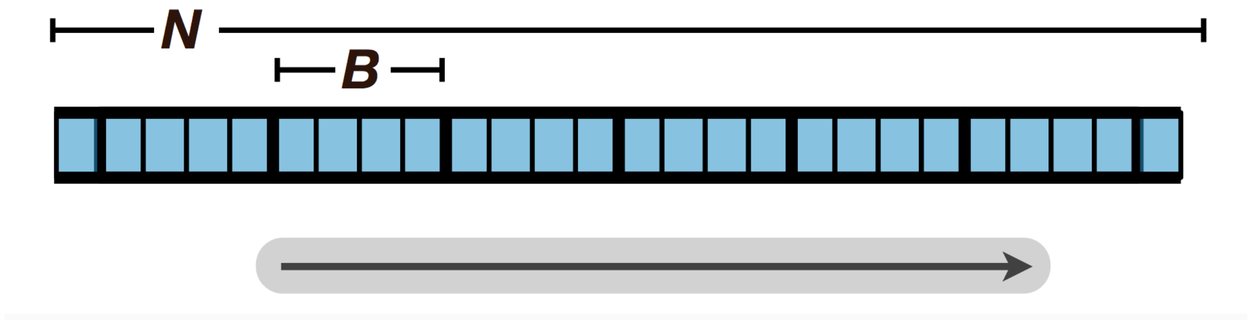
# External-Memory Model [Aggarwal & Vitter 1988]

- Models any two levels of memory hierarchy
- Memory of size $M$, disk of unlimited size
- Data is transferred in blocks of size $B$
- Algorithmic performance metric: number of block transfers (I/Os)
- Clean and useful model to predict cache-efficiency of algorithms
- Usually algorithms don't need to know $M, B$ (cache oblivious)



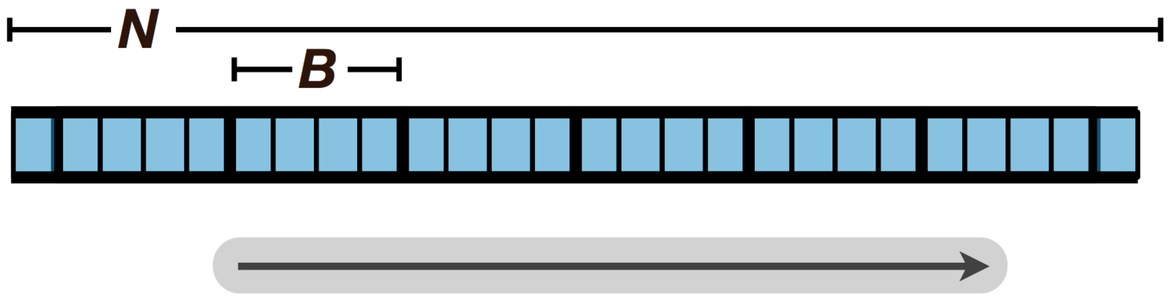
# Sequential Versus Random Access

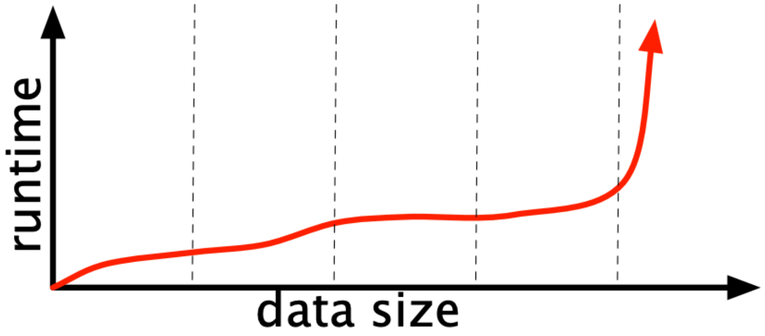
- Disk access usually $10^6$ times more expensive than RAM
- Sequential access patterns are better than random
    - Due to cache misses regardless of where in the memory hierarchy
- Consider traveling a linked list
- Naive: $O(N)$ I/Os (each hop to a new block)
- Smart: $O(N/B)$ I/Os (if sequential nodes stored in a single block)



# Difference Between $N$ and $N/B$

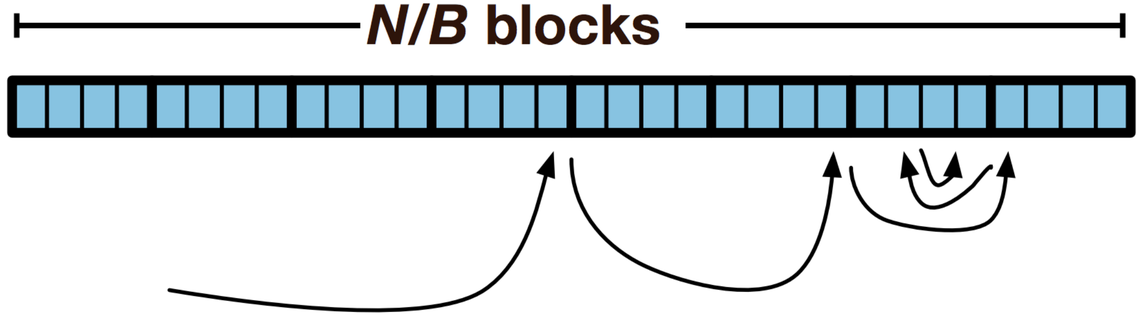
- Consider $N = 256 \times 10^6$, $B = 8000$
  - With disks: 1ms disk access time, SSDs: ~25 microseconds
- $N$ I/Os takes $256 \times 10^3$ sec = **71 hours** disks, ~**4 mins** for SSDs
- $N/B$ I/Os takes 256/8 sec = **32 sec**
- But, most problems developed in RAM model; why don't they thrash?
  - Under the hood OS stuff paging and prefetching
- Massive data and random access pattern can still hurt a lot

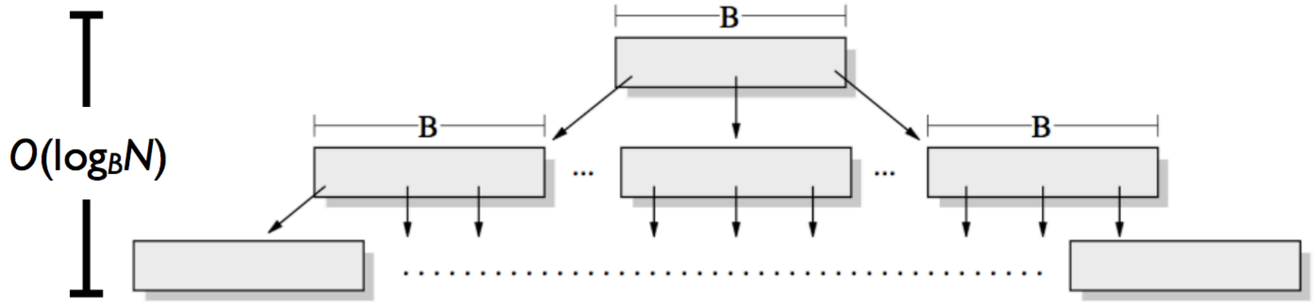


# Examples

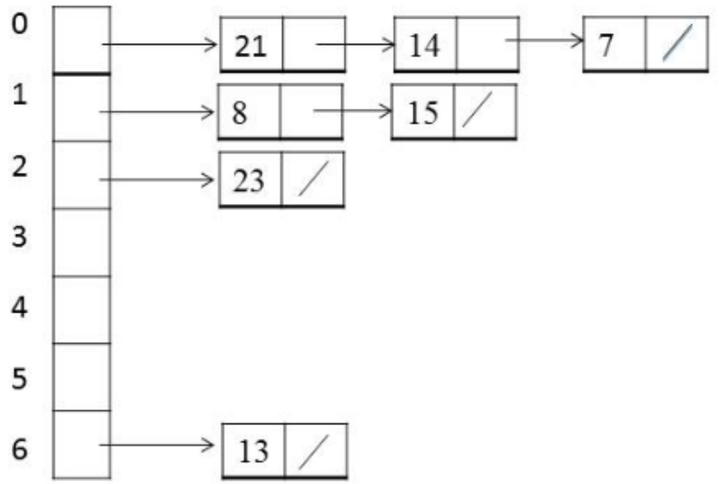
- **Standard binary search.** Cache-efficient or not?
  - Number of I/Os: $O(\log_2(N/B)) \approx O(\log_2 N)$
- Much more I/O efficient to search using a $B$-tree (a $B$-ary) tree

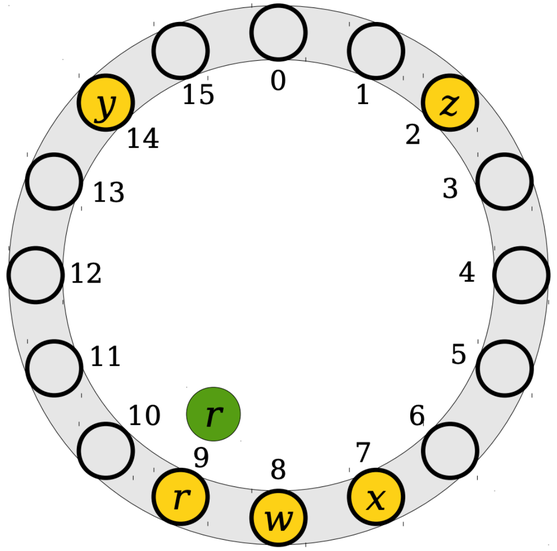


# Examples

- Hashing with **Chaining** vs **Linear Probing**
- Linear probing is much more cache-efficient
- In practice, linear probing is one of the fastest general-purpose hashing strategy
    - Low overhead
    - Excellent locality!
    - Great cache performance

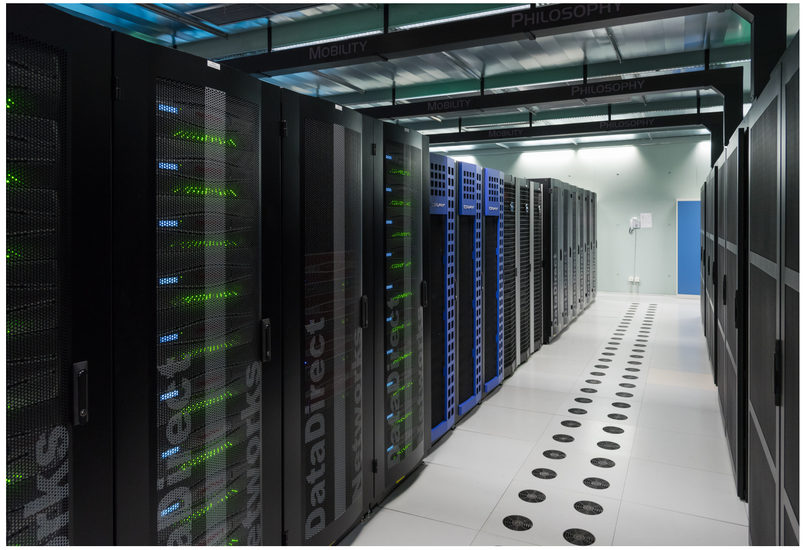


# Operations Aren't the Same

- Addition and subtraction are fast, multiplication is fast
- Division and modulo are slow
- Integers are faster than floats (?)
- Can we model this in theory?
- Not really.  Asymptotics ignore this intentionally
  - Going from $O(2^n)$ to $O(n)$, or even $O(n^2)$ to $O(n \log n)$, is more important
- Occasionally something like: $O(n \log n)$ additions, $O(n)$ divisions

# Parallelism




- Modern computers almost always have multiple compute cores
- If we have $p$ identical “processors” can we speed up our algorithms?
  - Maybe by a factor of $p$?
- Many models for algorithm analysis
  - PRAM is the classical model
  - MapReduce model: massive number of cores, want to minimize communication rounds
  - Many others

# Take Aways

- Big Oh analysis we did is the first strong predictor for performance
- But there are other factors like cache efficiency, that determine how well an algorithm will perform in practice
- When applying these to practice eye towards:
  - Other costs (like cache efficiency)
  - Abilities of modern hardware (parallelism, GPUs)

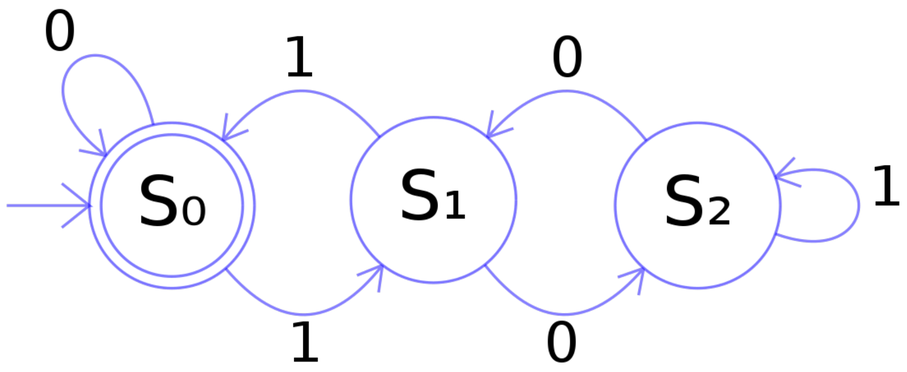
# Beyond CSCI 256:
# Similar Courses

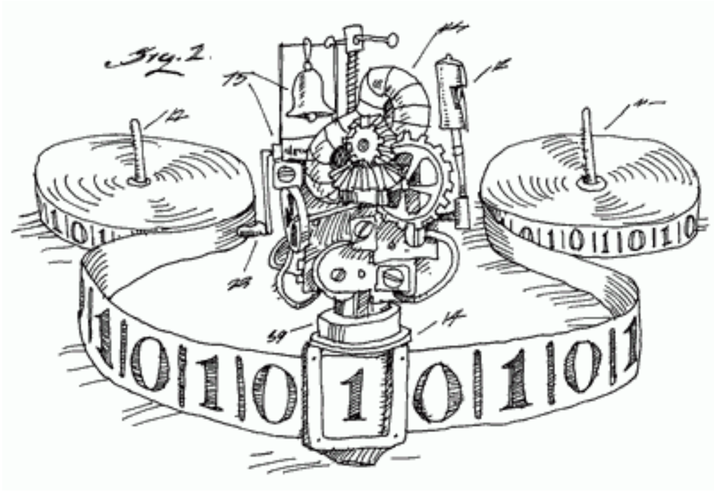
# Beyond CS 256:
## CSCI 361: Theory of Computation

Required course if you are CS major

- Formal framework for investigating both the computability and complexity of problems
- Several models of computation including finite automata, regular languages, context-free grammars, and Turing machines
- These models provide a mathematical basis for the study of computability theory

**Prerequisite: CS 256**

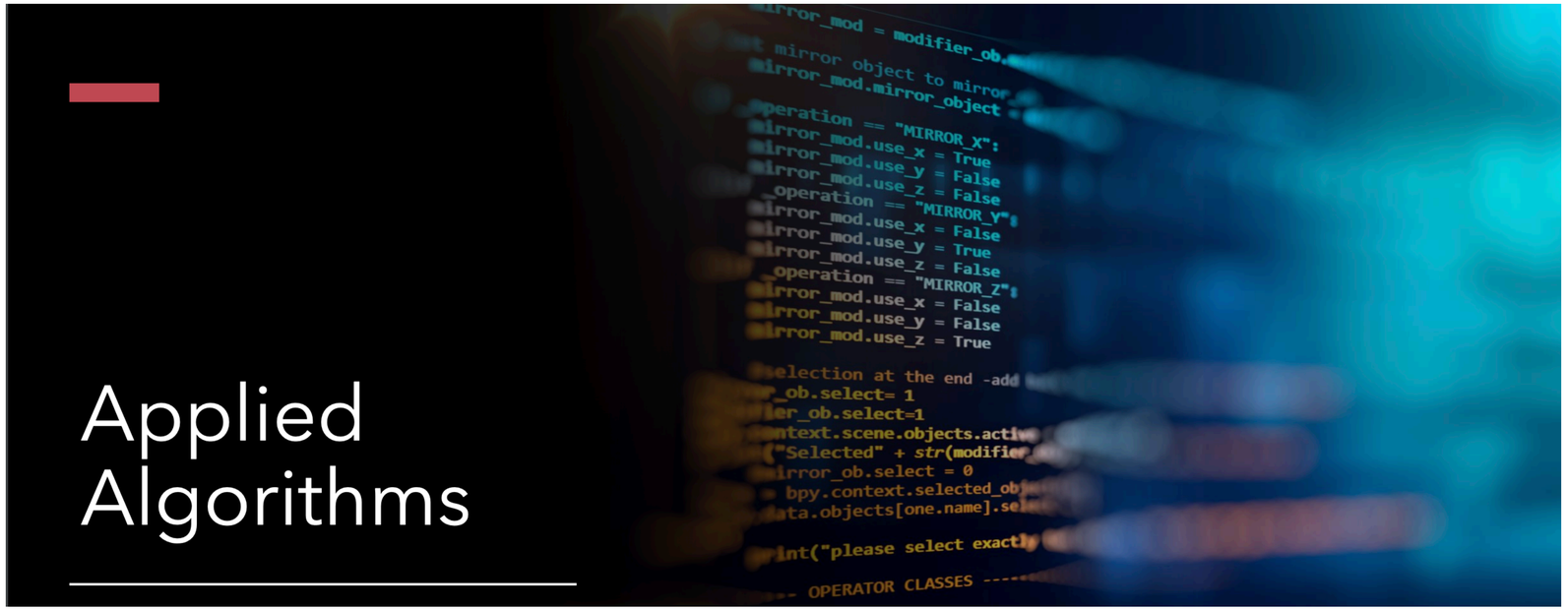


# Beyond CS 256:
# CSCI 358: Applied Algorithms

- Bridging the gap between theoretical running time and writing fast code in practice (taking advantage of modern hardware)
- Learn about the most useful tools in a coder's toolkits
- How to implement algorithms in an practically-efficient way
- Learn new algorithmic techniques not covered in 256

**Prerequisites:**
**CS 256 and CS237**

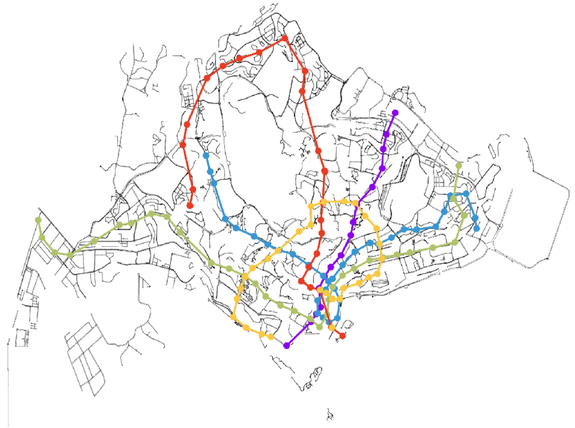


# Beyond CS 256:
# CSCI 357: Algorithmic Game Theory

Topics in game theory/mechanism design from a computational perspective

- how to design algorithms that incentivize truthful behavior, that is, where the participants have no incentive to cheat?
- Overarching goal is to understand and analyze selfish behavior and whether it can or should influence system design

**Prerequisite: CS 256**

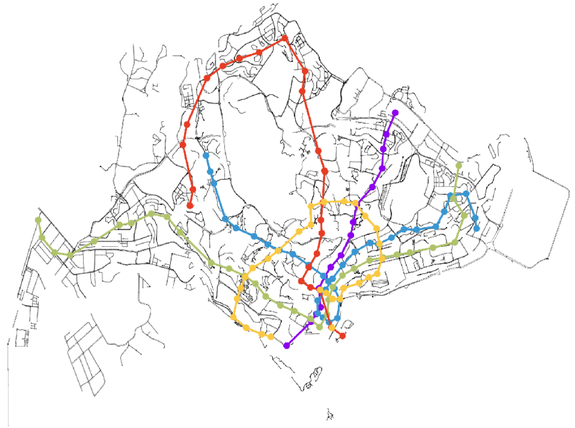



Source: https://courses.csail.mit.edu/6.851/spring12/

# Beyond CS 256:
# CSCI 357: Algorithmic Game Theory

*"The next time we bemoan people exploiting loopholes to subvert the intent of the rule makers, instead of asking 'What's wrong with these people?" let's instead ask, `What's wrong with the rules?' and then adopt a scientifically principled approach to fixing them. "*

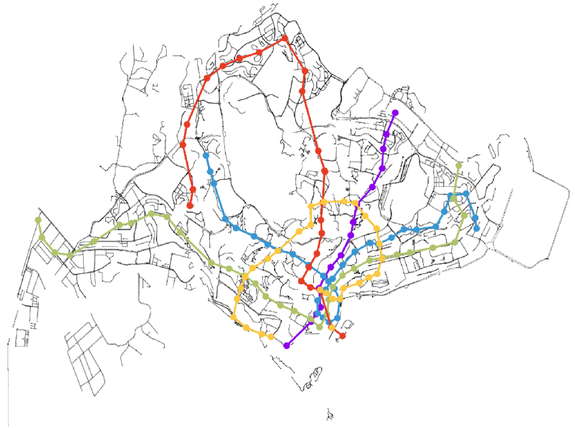
— Hartline and Kleinberg

# Beyond CS 256:
# CSCI 357: Algorithmic Game Theory

Examples of topics:

- auction design, efficiency of equilibria, network games
- two-sided markets, crowdsourcing markets
- incentives in computing applications such as file sharing, cryptocurrencies, etc
- computational social choice: selfish voting, resource allocation, etc.

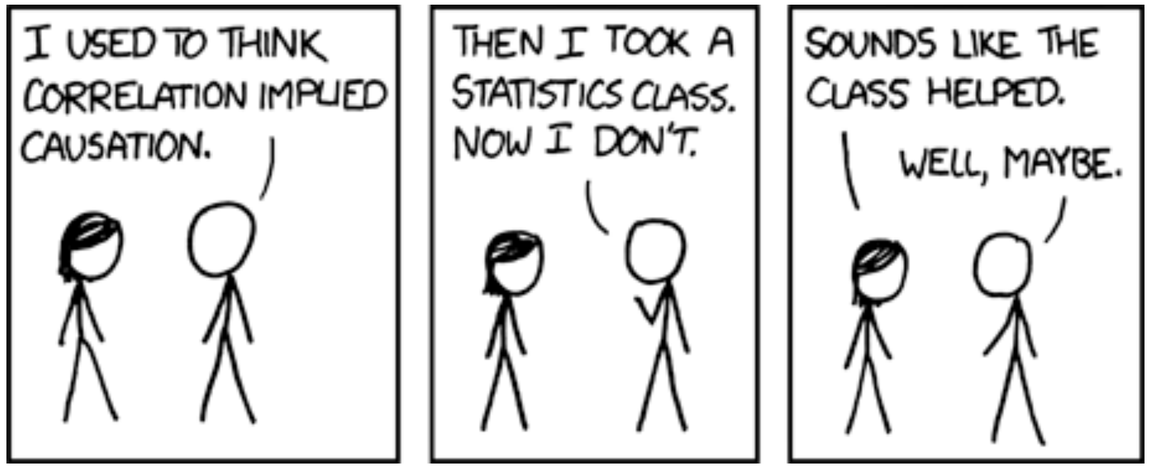



# Beyond CS 256:
## CSCI 379: Causal Inference

Does X cause Y? If so, how? And what is the strength of this causal relation? Teasing apart causation from association in messy real-world data; Topics in causal inference including

- causal graphical models and unsupervised learning of the structure of these models
- expression of causal quantities as functions of observed data
- and robust/efficient estimation of these quantities using statistical and machine learning methods

**Prerequisite:**
**CSCI 136, and either CSCI 256 or STAT 201/202**




https://images.app.goo.gl/idnuvemJmVrLn7oN9

# Course Evaluations

# About Course Evaluations

**Purpose**.

- Your feedback will help **improve this course** for other students taking it in the future
- Help shape the CS curriculum

Who sees the evaluation results:

  - Senior faculty in Dept & College
  - Maybe Maud
  - Me (only *after* I put in grades)

- Remember that the evaluations are **confidential**

# Course Evaluations: the Script

- You may skip questions that you don't wish to answer, and there is no penalty for choosing not to participate.
- All of your answers are confidential and I will only receive a report on your responses after I have submitted all grades for this course.
- While evaluations are open, I will receive information on how many students have filled out the evaluations, but I won't know which of you have and haven't completed them.
- I won't know which responses are associated with which student unless you identify yourself in the comments.

# Blue Sheets!

No longer blue or a sheet

"*Due to the pandemic, physical blue comment sheets will not be distributed. Instead, students will see a digital equivalent at the conclusion of the evaluations.*"

**Purpose.** Comments or feedback directed only to me (no one else reads these but me)

*To access the online evaluations, log into* ***Glow*** *(glow.williams.edu) using your regular Williams username and password (the same ones you use for your Williams email account). On your Glow dashboard you'll see a course called "****Course Evaluations****." Click on this and then follow the instructions you see on the screen. If you have trouble finding the evaluation, you can ask a neighbor for help or reach out to ir@williams.edu.*

# All the Best
# at
# Williams and Beyond

pls fill out course evals

# Acknowledgments


- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)