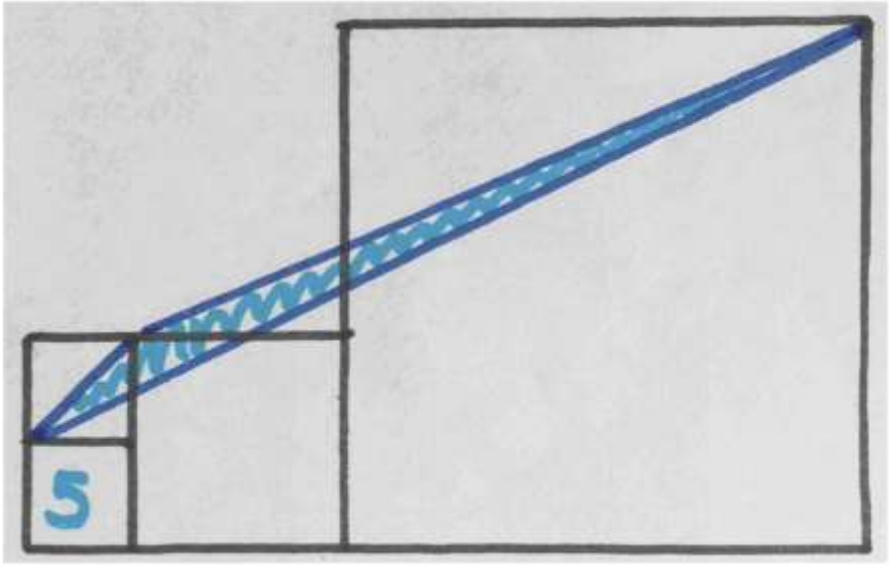
CS2040S
Data Structures and Algorithms
Conclusion

Puzzle of the Week:

The area of the bottom left square is 5. What's the area of the blue triangle?



Catriona Agg
https://twitter.com/cshearer41/status/1027844515338616832

Thanks to the tutors

for all their hard work!

CS2040S
Data Structures and Algorithms

What comes next?

Today's Plan

- ❑ **Announcements**
- ❑ **What comes next?**
- ❑ **Top 8 Algorithm Tips**
- ❑ **Wrap**

Announcements

Coursemology level

- **Today**: 67% at level 30 or above, 39% at level 32 or above.

- **Remains**: 1 problem EXP, 1 tutorial EXP, 1 recitation EXP, other assorted EXP

Final Exam:

- Date: Wednesday April 26
- Time: 13:00 – 15:00
- Location: MPSH 2-A, 2-B

(check specific hall assignment)

Final Exam: Covid Issues

- Check NUS policies
- Testing no longer *required*
- Testing still encouraged!
- If you feel unwell, please test.
- If you test positive, please do not come to take the exam in person.

Final Exam:

- Topics: *everything*
- Cheat sheet: Bring 1 (double-sided, A4) sheet of notes.
- Calculator/phone: NO
- Previous years papers: last 2 years uploaded

Final Exam:

Format: *100% MCQ (similar to midterm assessment)*

** NO SHORT QUESTIONS (unlike previous iterations)*

Review Sessions: *(optional)*

Grad TAs will hold review sessions next week.

Each session will be on one topic from this class

(e.g., Theory, Searching, Sorting, Trees, Hashing, Graphs, etc.)

Ask questions in advance on Coursemology survey.

Show up to discuss answers!

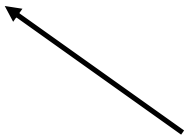
Announcements

Teaching Feedback:

- Please fill in departmental teaching feedback.

- Please fill in Coursemology feedback survey.

  (One last chance at a few XP!)



Open tomorrow…

Problem Sets:

Last chance for submissions: Friday, April 14, 11:59pm.

Tech Interview Preparation for Summer (TIPS)

Overview

SoC will be conducting a **10-week** programme this summer to ensure that our freshmen are ***well-prepared*** *to* handle technical (coding) interviews when they apply for **technical internships**.

Team

Senior students with a wide range of internship experiences from companies like Meta, Google, Stripe, Asana, Indeed, Shopee, ByteDance, etc.

Programme Outline

- Resume preparation & reviews
- Technical interview skills & techniques
- Weekly technical interview preparation questions
- Weekly mock interviews
- Career & internship sharings from senior students & alumni from different companies
- Career discussion panels with recruiters & tech chiefs from tech companies with a local presence.

Tech Interview Preparation for Summer (TIPS)

Weekly Commitment

- Attend weekly sharing session on **Monday evening 8pm - 9.30pm** (23rd May - 25th July)
- Complete at least **7 LeetCode questions**
- Complete one **mock interview** with a classmate from Week 3 onwards

***** in order to make sure everyone in the program is responsible and committed to the peer activity (e.g. mock interview), failure in completing the weekly commitment may result students being removed from the programme

FAQ

Q: Can I join this if I have Orbital / CVWO / Internship / anything else?

A: Yes! As long as you are able to complete the weekly commitment.

For any enquiries, please email soc-tips@googlegroups.com.

Sign up now on https://forms.gle/jnom3Ugyu6g6KRyU9!

Other practice material for technical interview prep:

https://github.com/yangshun/tech-interview-handbook
https://leetcode.com/
https://nus.kattis.com/courses/CS2040

Today's Plan

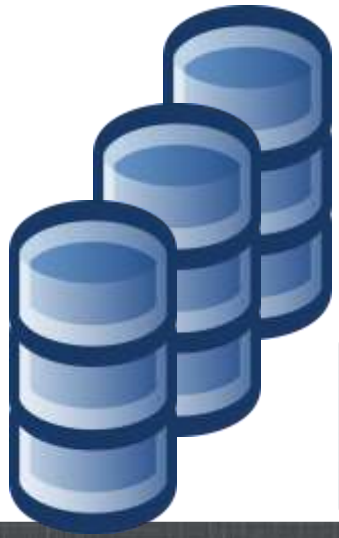
- ❑ **Announcements**
- ❑ **What comes next?**
- ❑ **Top 8 Algorithm Tips**
- ❑ **Wrap**

Algorithms Everywhere

Web browser:
Parsing
Substring manipulation
XML trees

Internet routing:
TCP (congestion control)
IP routing
BGP (Bellman-Ford)
Content caching
DNS

Google:
PageRank
String matching

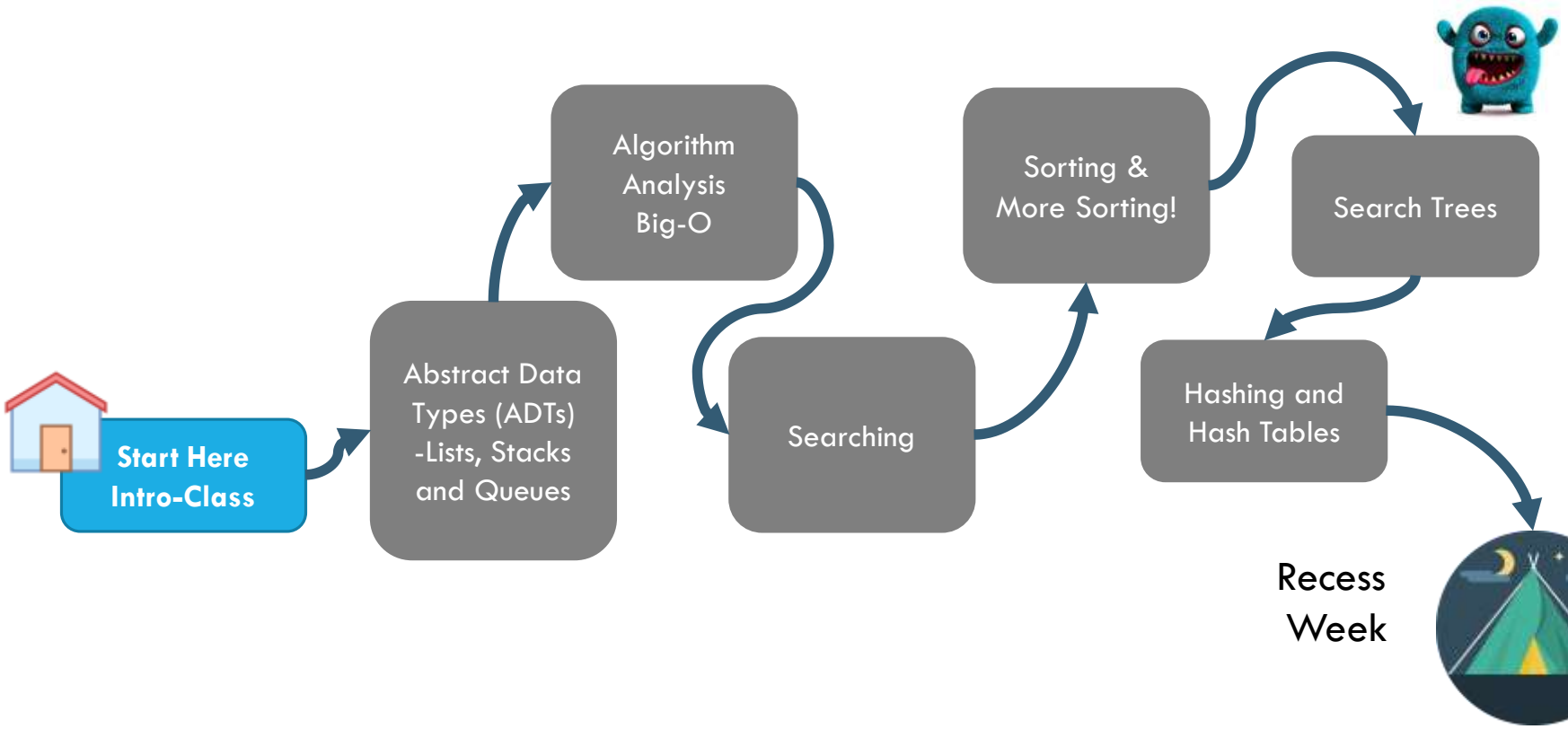


Web servers:
Load balancing
Scheduling
Memory allocation

Database:
B-trees
Search
Sorting

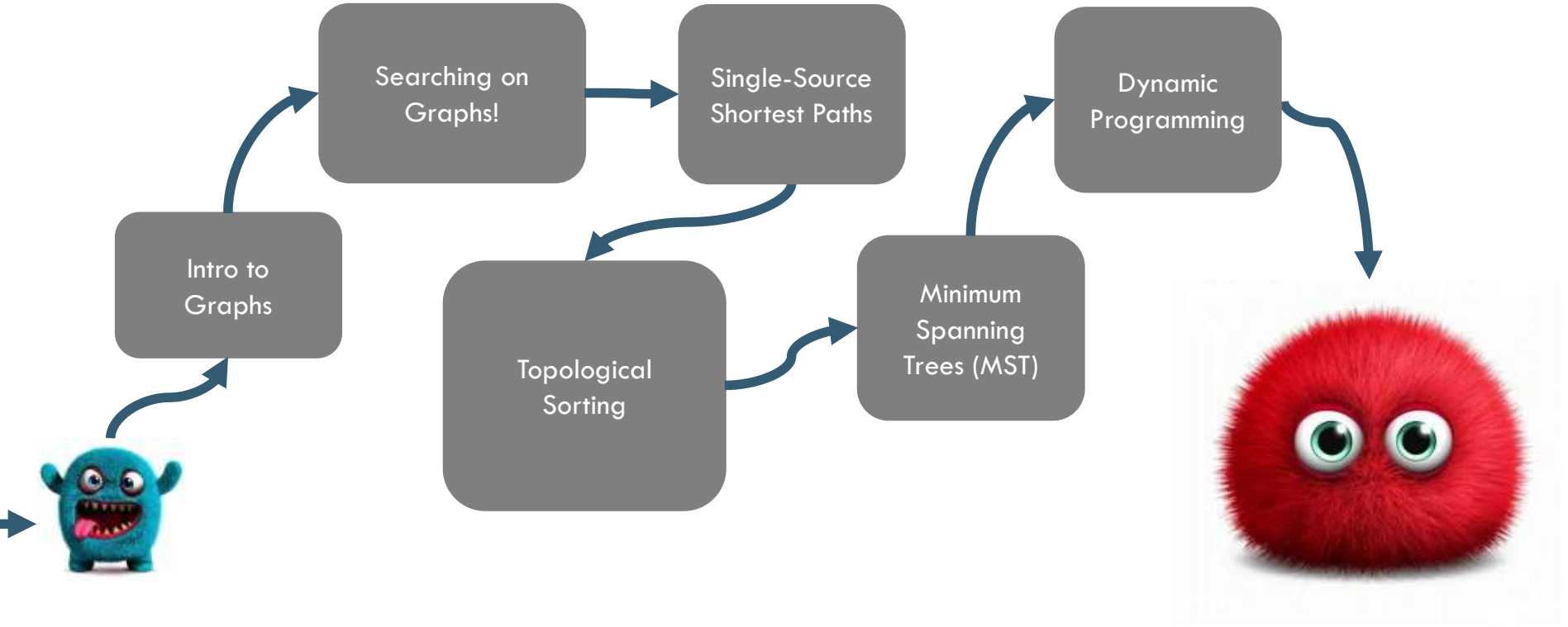
What is CS2040S about?

COURSE STRUCTURE



PART I: ORGANIZING YOUR DATA

COURSE STRUCTURE



PART II: MODELLING AND SOLVING PROBLEMS

Data Structures

Problem Solving

Algorithms

Desirable features of your algorithm:

How do you choose the right algorithm for the right problem?

How do you design new algorithms for new problems?

FRAMEWORK: ALGORITHM & DATA STRUCTURE

What problem does it solve?

How does it work?

How to implement it?

What is its asymptotic performance?

What is its real world performance?

What are the trade-offs?

GOALS

By the end of this course, you should be able to:

- **Apply algorithmic thinking and techniques** for solving computational problems.
- **Describe the structure and operation** of different **data structures and algorithms** under the standard computational model.
- **Assess the suitability** of different data-structures and algorithms for a specific computational problem.
- **Adapt existing data-structures and algorithms** to solve specific computational problems.

What comes next?

What's next?

Topic 1: Searching and Sorting

Topic 2: Trees

Topic 3: Hashing

Topic 4: Shortest Paths

Topic 5: Minimum Spanning Trees

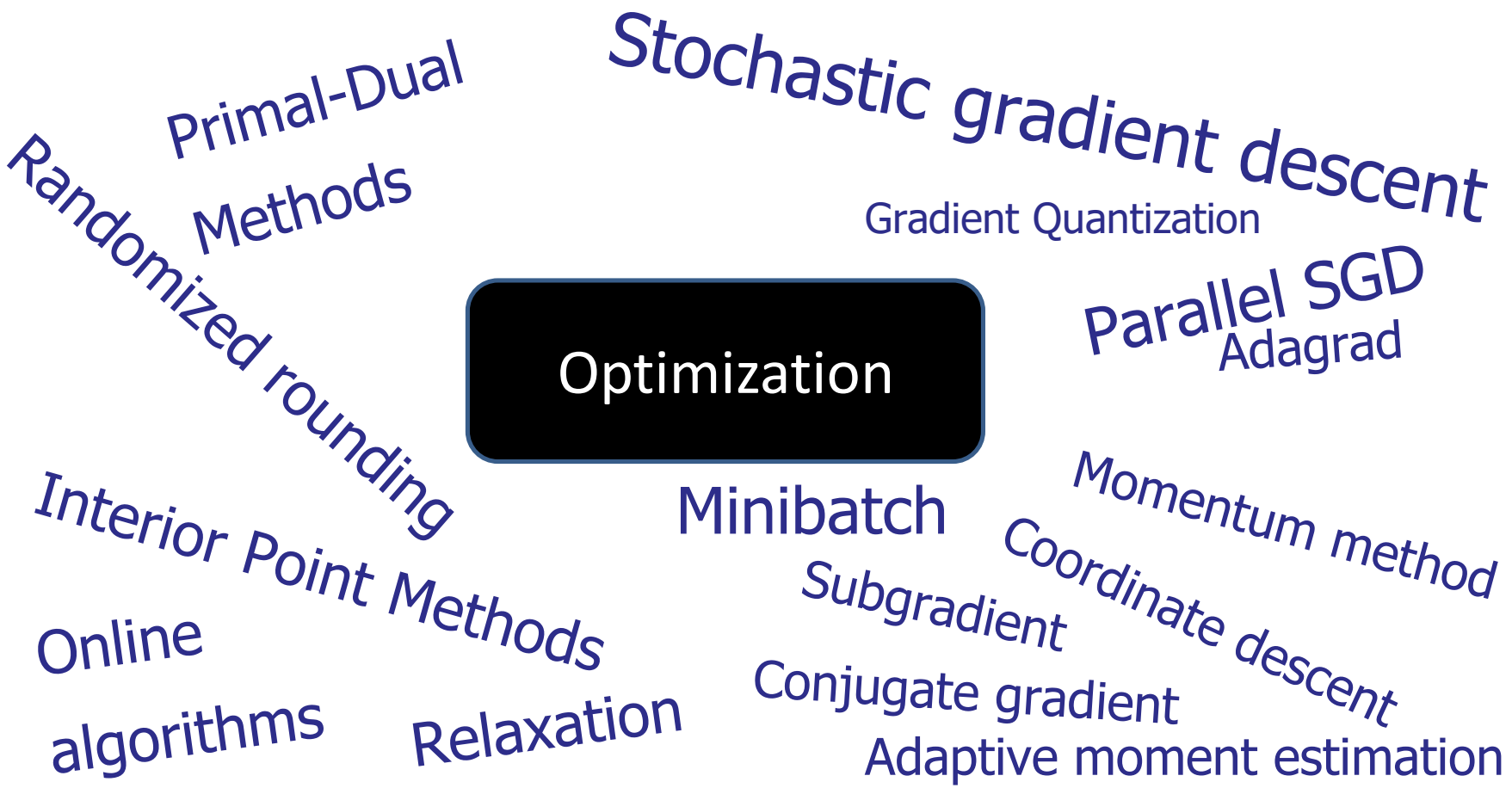
Topic 1: Searching and Sorting

Binary Search:

- Fast way to search monotonic data.
- Fast was to find max/min for convex data.

➔ Optimization

Huge area of research and development:



Optimization algorithms:
General techniques.

Machine learning:
How to train a model?

Optimization Algorithms

Linear Programming:

- How to optimize a linear function subject to linear constraints.
- E.g., simplex method

max $(A+6B)$

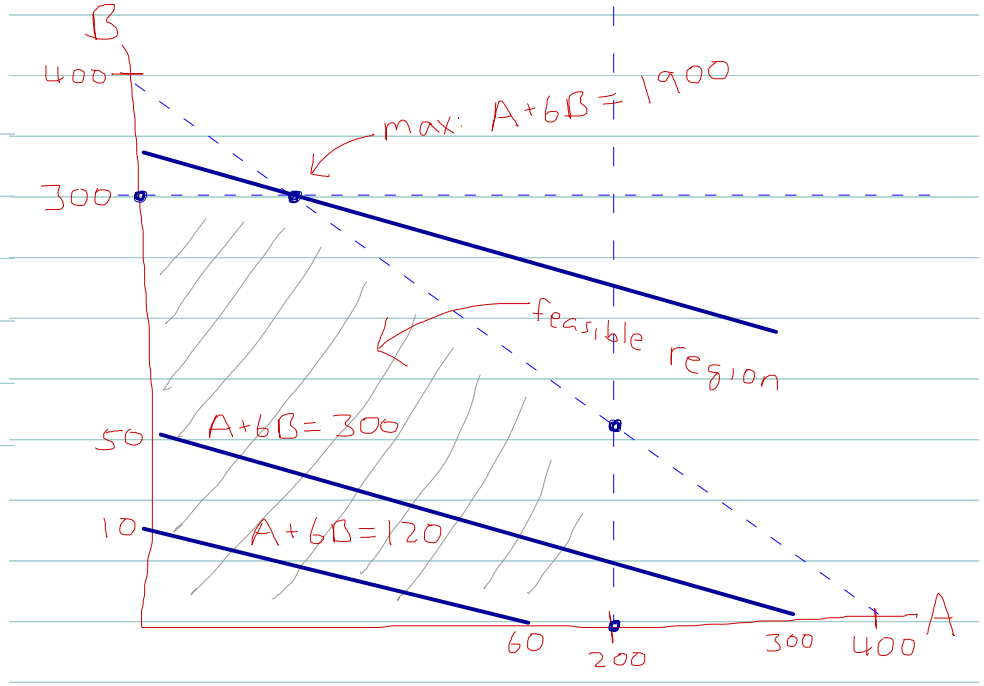
where: $A \leq 200$

$B \leq 300$

$A + B \leq 400$

$A \geq 0$

$B \geq 0$



Optimization Algorithms

Linear Programming:

- How to optimize a linear function subject to linear constraints.
- E.g., simplex method

Applications:

- Graph algorithms (e.g., max-flow)
- Approximation algorithms (e.g., weighted vertex cover, weighted set cover, multicommodity flow)
- And many, many real-world problems.

Optimization Algorithms

Linear Programming:

- How to optimize a linear function subject to linear constraints.
- E.g., simplex method

And more…

- Semidefinite programming
- Integer linear programming (NP-hard)
- Quadratic programming (NP-hard)
- Constraint satisfaction (NP-hard)

Topic 1: Searching & Sorting

Fast Sorting Algorithms:

- QuickSort
- MergeSort
- HeapSort

Key properties:

- Running time
- Space usage
- In-place

Sorting Faster!

Yahoo TeraSort:

- Each node has:
  - 8 cores: 2GHz
  - 8 GB RAM
  - 4 disks: 4TB each
- 40 nodes / rack (interconnect: 1GB/s switch)
- 25-100 racks (interconnect: 8GB/s switch)

➔ ~ 16,000 cores

Sorting Faster!

Yahoo TeraSort:

- Each node has:
  - 8 cores: 2GHz
  - 8 GB RAM
  - 4 disks: 4TB each
- 40 nodes / rack (interconnect: 1GB/s switch)
- more racks (interconnect: 8GB/s switch)

➔ 50,400 cores

2013:
Yahoo (Hadoop) sorts 100TB of data in 72 minutes.

Sorting Faster!

Tencent Sort:

- 512 nodes
- 5,024 cores

Record (2016):

Tencent sorts 100TB of data in <100 seconds.

DataBricks TeraSort:

- 206 nodes
- 6,592 cores

Record (2014):

DataBricks (Spark) sorts 100 TB of data in 23 minutes.

Sorting Faster!

It’s a race:

- High performance clusters.
- Hardware interconnect.
- Parallel performance.

Topic 2: Search Trees

Many types of search trees:

- **AVL trees and** Red-Black trees
- **B-trees**
- **Skip Lists**
- **Tries**
- **Interval Trees, Order Statistics Trees**
- **Range Trees, kd-Trees**
- Splay trees

Key tree-related questions:

High dimensional data:

- How should we store higher dimensional data?
- How should we index higher dimensional data?
- How should we search higher dimensional data?

Examples:

R-trees, spatial indices, quadtrees, Z-order curve, …

Key tree-related questions:

Performance?

Predicting Performance

Example: 100 TB of data

1) Store data sorted in an array

⇒ Scan all the data: O(n)

⇒ (Binary) search: O(log n)

2) Store data in a linked list

⇒ Scan all the data: O(n)

⇒ Search: O(n)

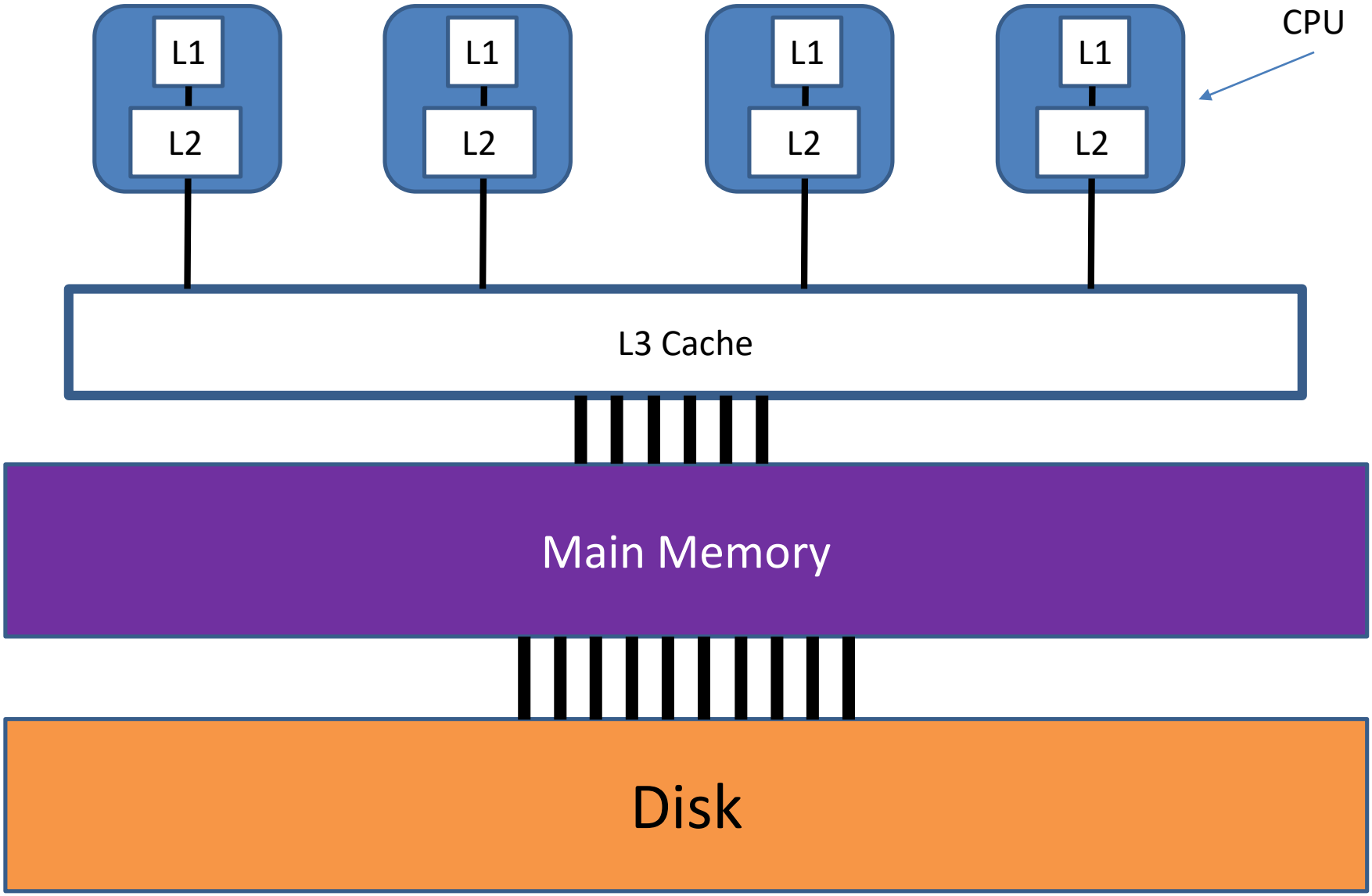
3) Store data in a red-black tree

⇒ Scan all the data: O(n)

⇒ Search: O(log n)

Analysis is not predicting performance very well!

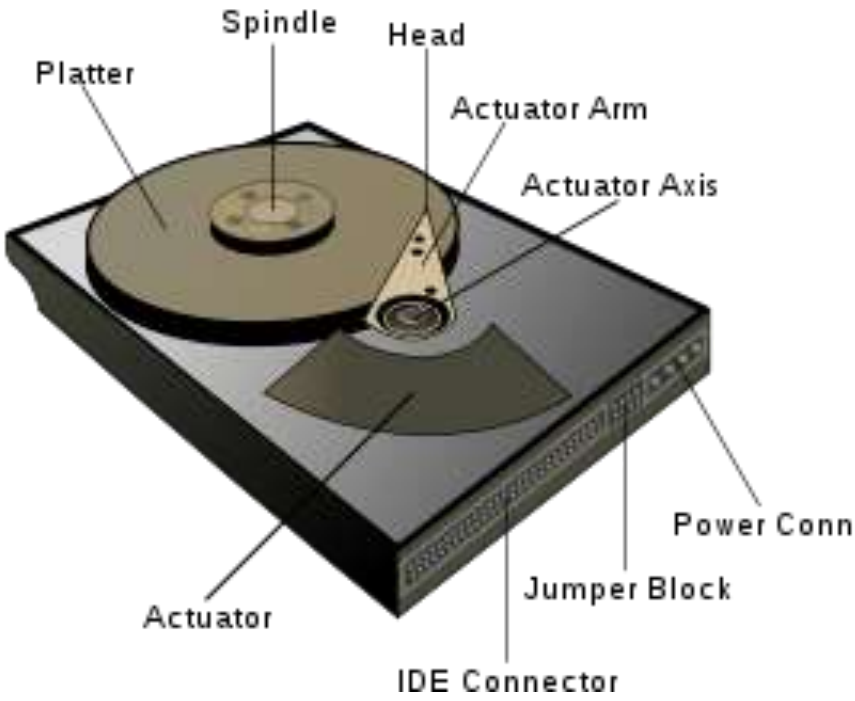
A Real Computer (?)



Disks

Where is most data stored? Hard disk!

- Magnetic
- Mechanical
- Slow (6000rpm = 10ms)



Two step access:

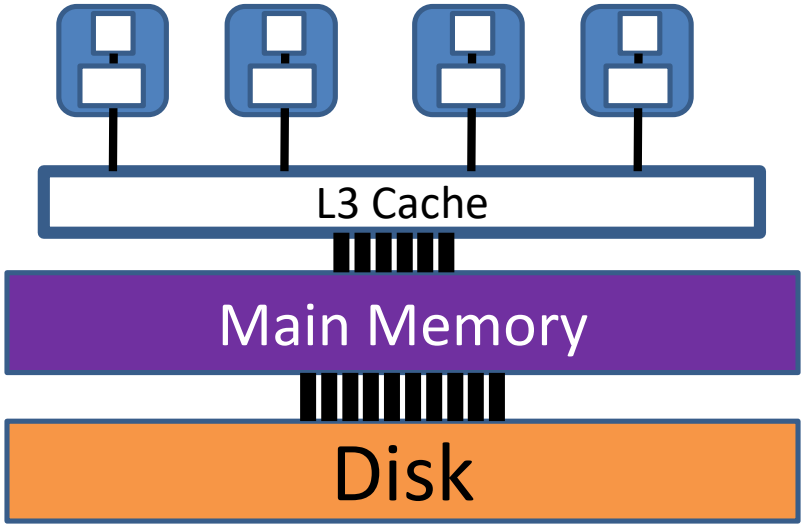
1. seek *(find right track)*
2. read track

Haswell Architecture (2-18 cores)

Memory Type	size	line size	clock cycles
L1 cache	64 KB	64 B	~4
L2 cache	256 KB	64 B	~10
L3 cache	2-40 MB	64 B	40-74
L4 (optional)	128 MB		
Main Memory	< 128 GB	16 KB	~200-350
SSD Disk	BIG	Variable (e.g., 16KB)	~20,000
Disk	BIGGER	Variable (e.g., 16KB)	~20,000,000

Notes:

- Several other ”caches” e.g., TLB, micro-op cache, instruction cache, etc.
- L1/L2 caches are per core.
- L3/L4 cache are shared per socket.
- Main memory shared cross socket.



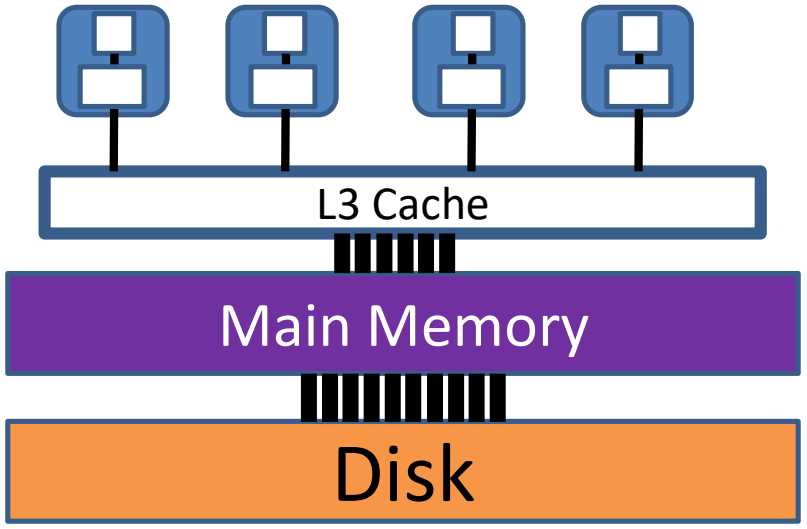
Haswell Architecture

A simple example calculation:

What fraction of operations ”hit” each cache?

- ⇒ 90% L1 hit rate (4 cycles)
- ⇒ 8% L2 hit rate (10 cycles)
- ⇒ 2% main memory (300 cycles)

Just an example..



Haswell Architecture

A simple example calculation:

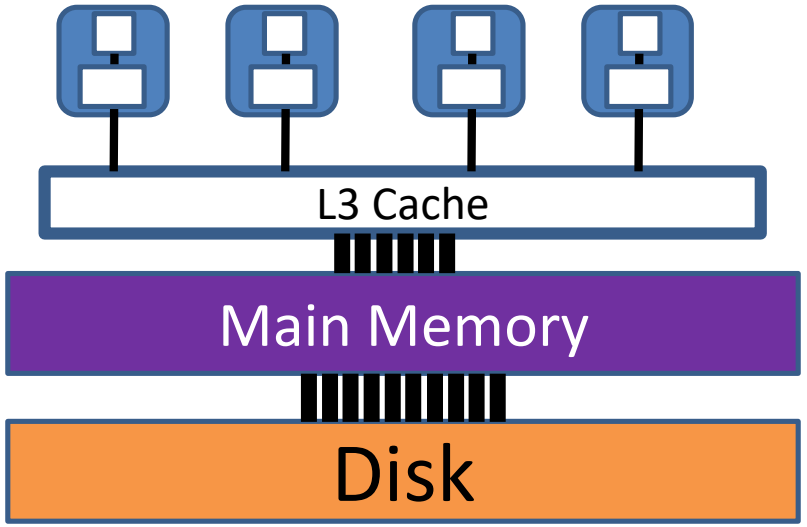
What fraction of operations ”hit” each cache?

- ⇒ 90% L1 hit rate (4 cycles)
- ⇒ 8% L2 hit rate (10 cycles)
- ⇒ 2% main memory (300 cycles)

Just an example..

What fraction of time for each cache?

- ⇒ 35% waiting for L1
- ⇒ 8% waiting for L2
- ⇒ 57% waiting for main memory



Conclusion:

98% cache hit ➔

57% waiting on main memory

Haswell Architecture

A simple example calculation:

What fraction of operations ”hit” each cache?

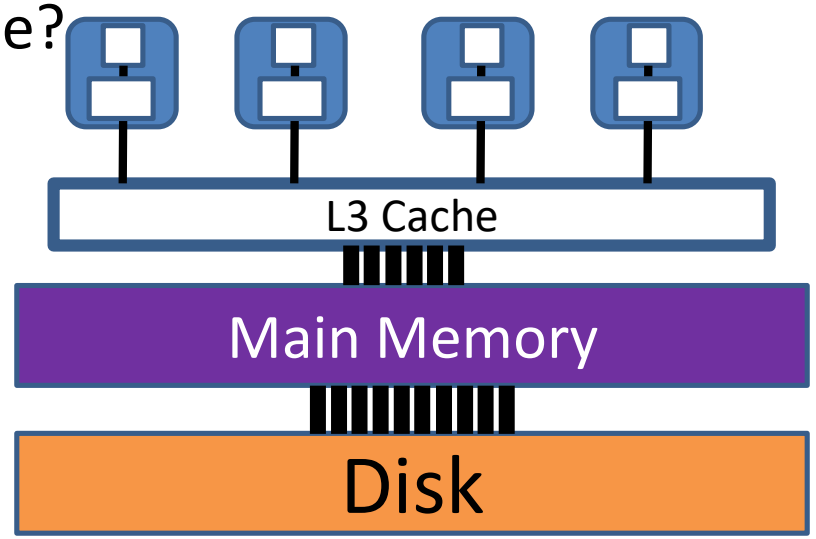
- ⇒ 90% L1 hit rate (4 cycles)
- ⇒ 8% L2 hit rate (10 cycles)
- ⇒ 1.8% main memory (300 cycles)
- ⇒ 0.2% disk (20,000,000 cycles)

Just an example..

What fraction of time for each cache?

- ⇒ 99.98% waiting for disk

Disk is much, much worse!



B-trees

Basic facts

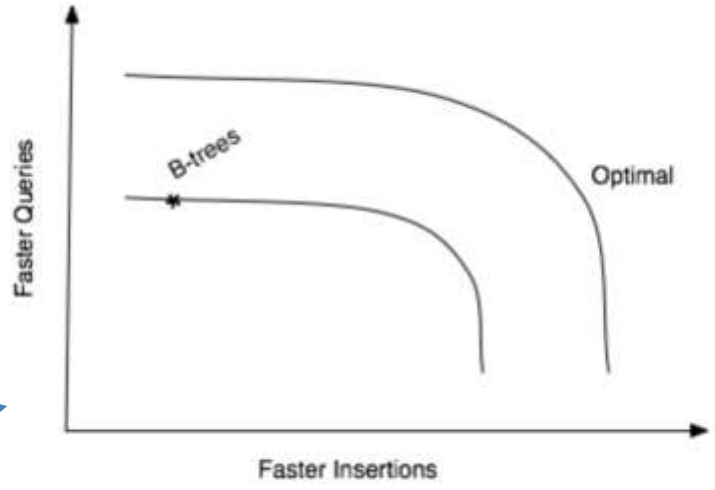
- One of the most important data structures out there today. (Variants used in all major databases.)
- Very fast. (Not just asymptotic analysis, but in practice nearly impossible to beat a well-implemented B-tree.)
- Benefit comes both from good cache performance, low overhead, good parallelization, etc.

Faster Trees?

Goal:

A external memory data structure with fast searches, and super-fast insertions/deletions.

"Write-optimized data structure."



B-trees are not on the optimal insert/query tradeoff curve

Percona advertisement graph
(not technical)

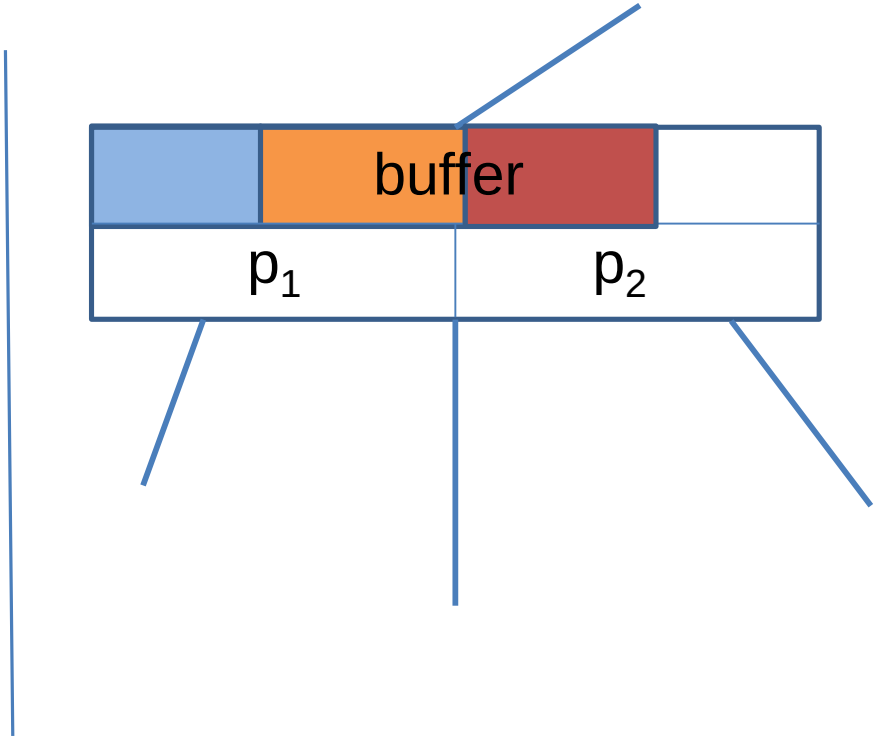
Buffer Tree

Summary

Cost of operations:

insert/delete: $O\left(\frac{1}{B}\log n\right)$

search: $O\left(\log n\right)$



Faster Trees?

Goal:

A external memory data structure with fast searches, and super-fast insertions/deletions.

"Write-optimized data structure."

Examples:

- LSM: log-structured merge trees
- COLA: cost-oblivious lookahead array

Hot area of DBS research today…

Topic 3: Hash Tables

Two key types of hash tables:

- Chaining
- Open Addressing

Better Collision Resolution

Chaining:

- O(1) *expected* search
- O(1) *worst-case* insertion

Cuckoo Hashing:

Neat, newer hashing method!

- O(1) *worst-case* search
- O(1) *expected* insertion

More realistic hash functions

How well does linear probing really work?

Does open addressing work with realistic hash functions?

- Example: limited independence hash functions
- Example: tabular hashing

YES: it works really well!

Better hash functions

Faster hash functions:

- Example: xxHash, etc.
- Example: tabular hashing

Fastest today??
For GPUs??

Cryptographic hash functions:

- Example: MD6,
- Example: SHA-512

Are these secure?

Topic 3: Hash Tables

Use case of a hash table: maintain a set of items

Hash Sets / Filters:

- Fingerprint Hash Tables
- Bloom Filters

Better filters

Quotient Filters:

- Optimal trade-off error vs. space.
- Practical and easy to implement

Cuckoo Filters:

- Interesting alternative to a Bloom Filter

Optimizations: Bloomier filters, compact approximators…

Key question: minimize space, minimize error

Better filters

Learned Bloom Filters:

- Can we use machine learning to train better Bloom Filters?

A Model for Learned Bloom Filters, and Optimizing by Sandwiching

Michael Mitzenmacher
School of Engineering and Applied Sciences
Harvard University
michaelm@eecs.harvard.edu

Abstract

Recent work has suggested enhancing Bloom filters by using a pre-filter, based on applying machine learning to determine a function that models the data set the Bloom filter is meant to represent. Here we model such *learned Bloom filters*, with the following outcomes: (1) we clarify what guarantees can and cannot be associated with such a structure; (2) we show how to estimate what size the learning function must obtain in order to obtain improved performance; (3) we provide a simple method, sandwiching, for optimizing learned Bloom filters; and (4) we propose a design and analysis approach for a learned Bloomier filter, based on our modeling approach.

- Can we get a better trade-off between common-case and worst-case performance?

Applications of Filters

Caches:

- What is in your cache?
- Check bloom filters before accessing cache.

Learned Bloom Filters:

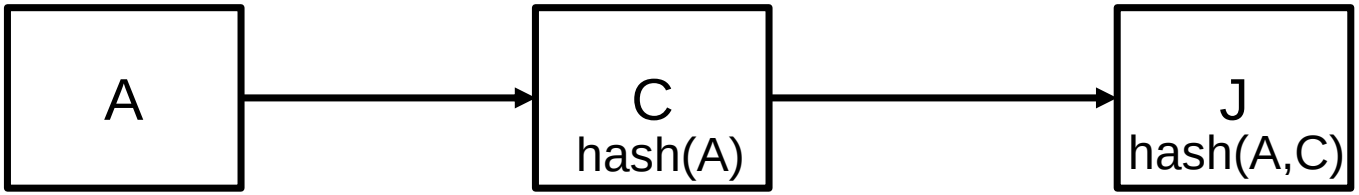
- Use machine learning to choose most useful items to store in filter.
- Then we need less space!

See: https://papers.nips.cc/paper/7328-a-model-for-learned-bloom-filters-and-optimizing-by-sandwiching.pdf

Blockchains

Blockchain = Hashchain

- proof-of-work == inverting hash function
- hash summarizes chain (see: Merkle trees!)



Topic 4: Shortest Paths

Several situations:

- Unweighted graphs
- Directed acyclic graphs
- Graphs with negative weights
- Graph with positive weights
- All-pairs shortest paths

Topic 4: Shortest Paths

Key algorithms:

- BFS
- Topological sort
- Bellman-Ford
- Dijkstra
- Floyd-Warshall

Topic 5: Minimum Spanning Trees

Key algorithms:

- Prim's
- Kruskal's
- Boruvka's

Sub-components:

- Union-Find
- Priority Queues

Key Questions

Big Data

Parallelism

Distributed Network Applications

Parallel Algorithms

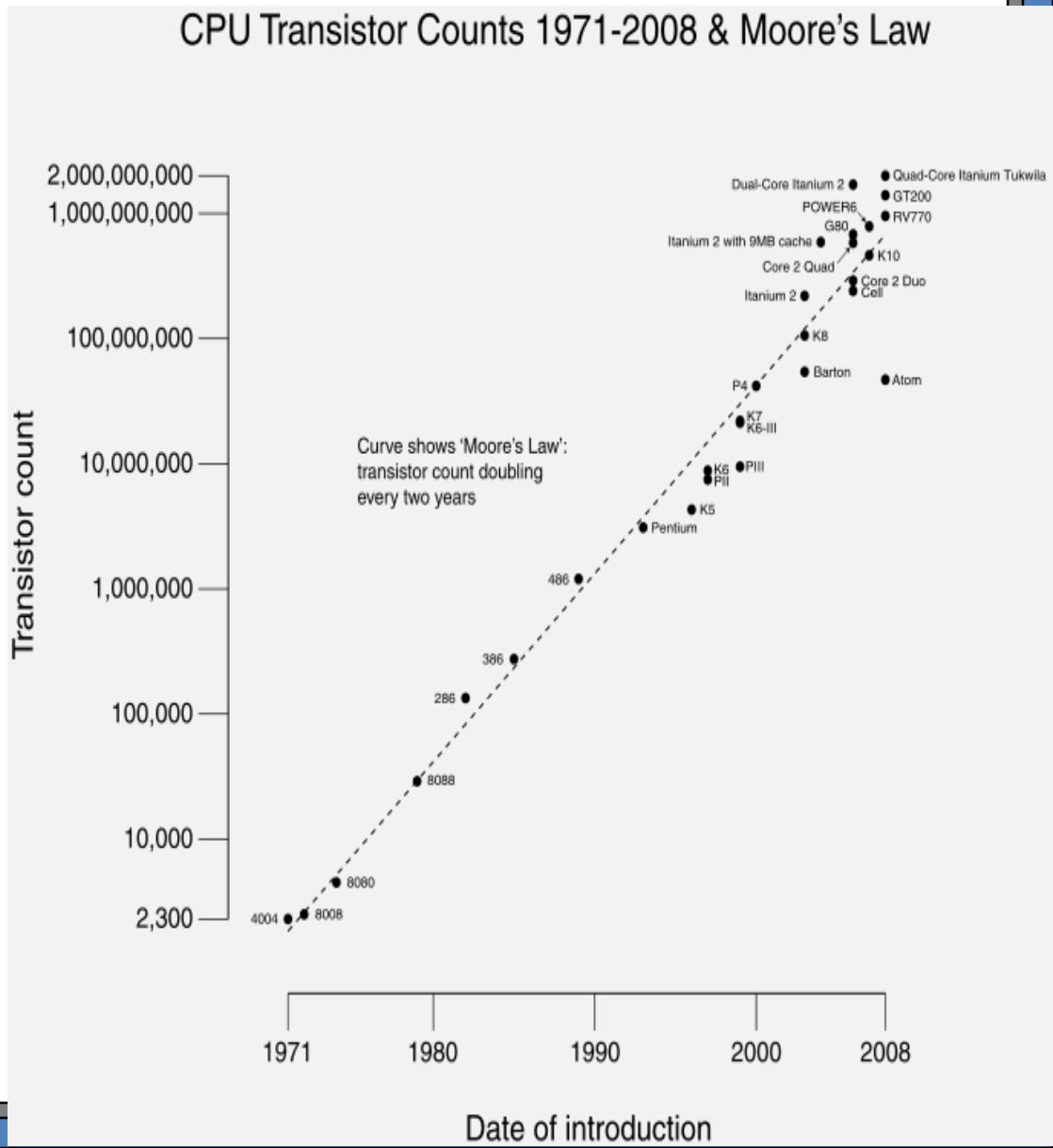
Moore's Law

Number of transistors doubles every 2 years!

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year... Certainly over the short term this rate can be expected to continue, if not to increase." Gordon Moore, 1965

Limits will be reached in 10-20 years…maybe.

Source: Wikipedia



Parallel Algorithms

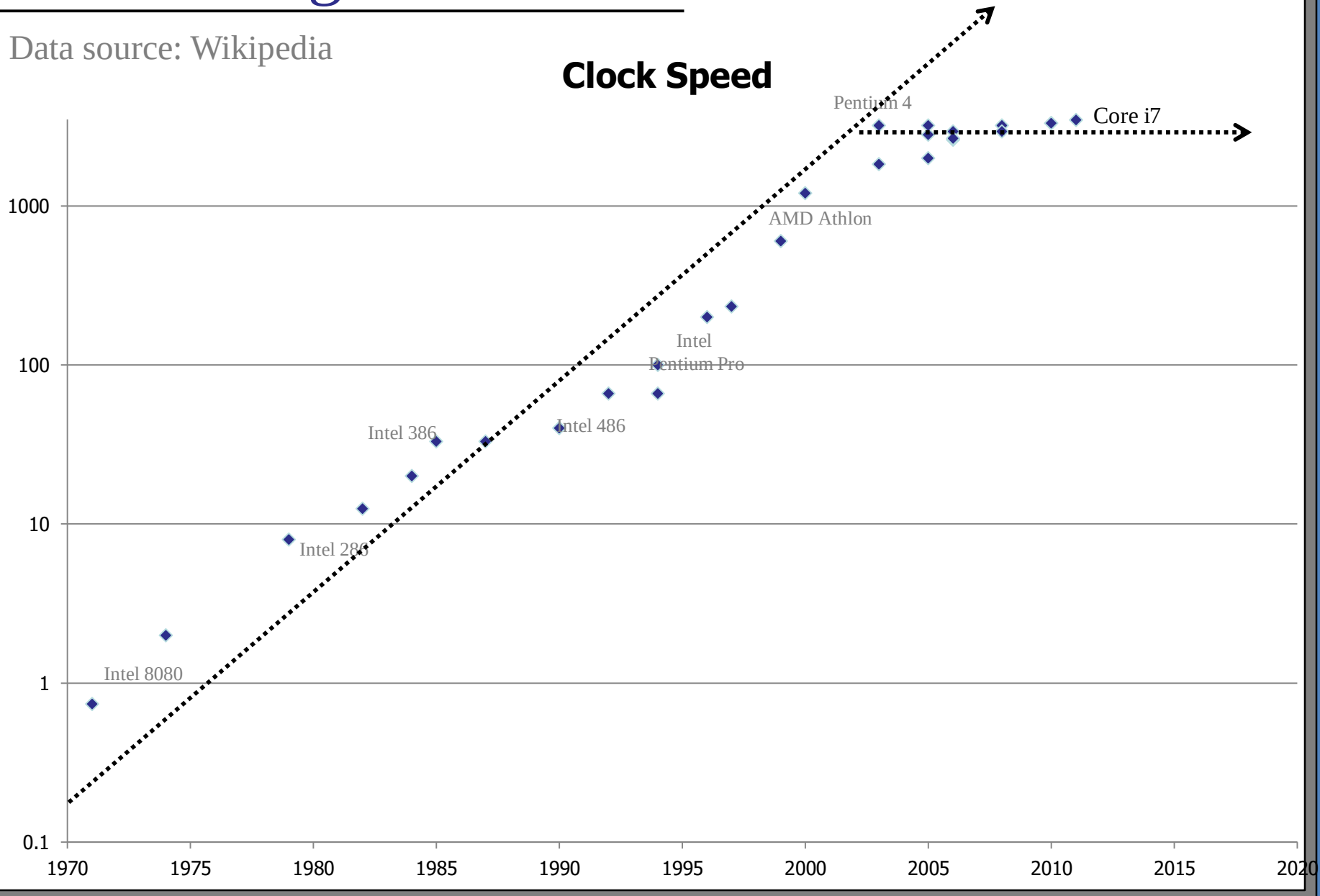
More transisters == faster computers?

- More transistors per chip ➔ smaller transistors.
- Smaller transistors ➔ faster
- Conclusion:

  Clock speed doubles every two years, also.

Parallel Algorithms

Data source: Wikipedia



Parallel Algorithms

What to do with more transistors?

- More functionality
  - GPUs, FPUs, specialized crypto hardware, etc.
- Deeper pipelines
- More clever instruction issue (out-of-order issue, scoreboarding, etc.)
- More on chip memory (cache)

Limits for making faster processors?

Parallel Algorithms

Problems with faster clock speeds:

- Heat
  - Faster switching creates more heat.
- Wires
  - Adding more components takes more wires to connect.
  - Wires don't scale well!
- Clock synchronization
  - How do you keep the entire chip synchronized?
  - If the clock is too fast, then the time it takes to propagate a clock signal from one edge to the other matters!

Parallel Algorithms

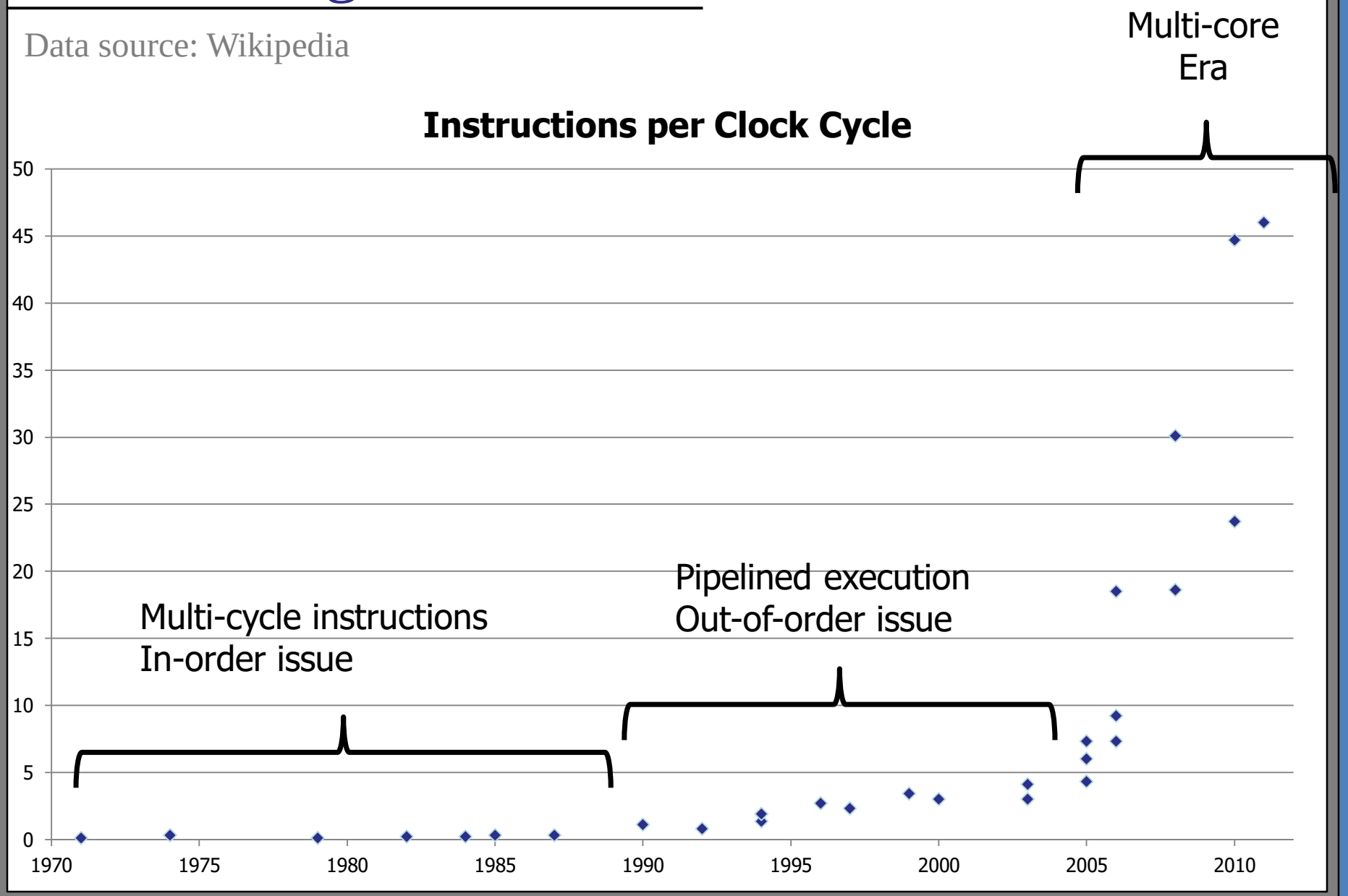
Conclusion:

- We have lots of new transistors to use.
- We can't use them to make the CPU faster.

What do we do?

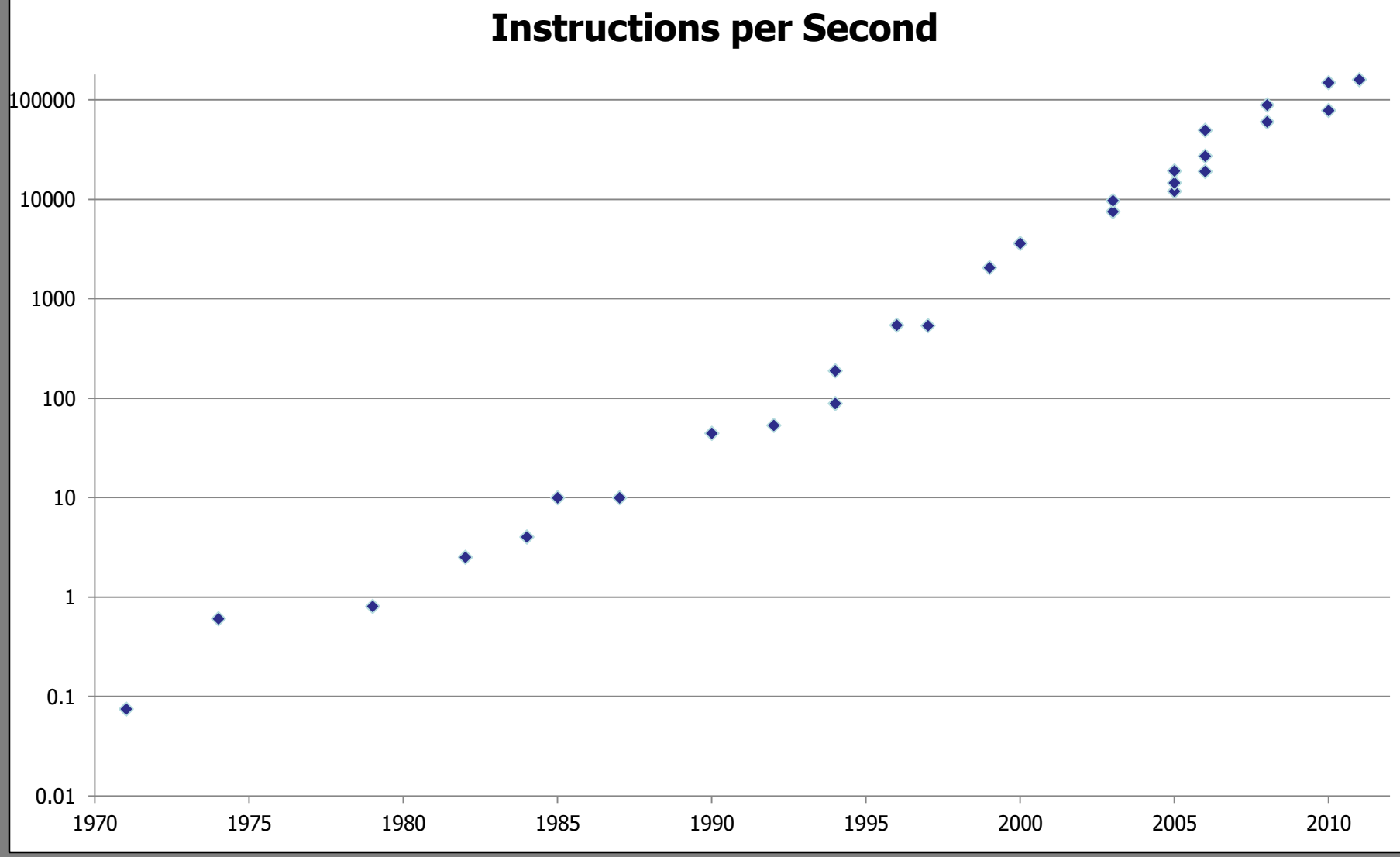
Parallel Algorithms

Data source: Wikipedia



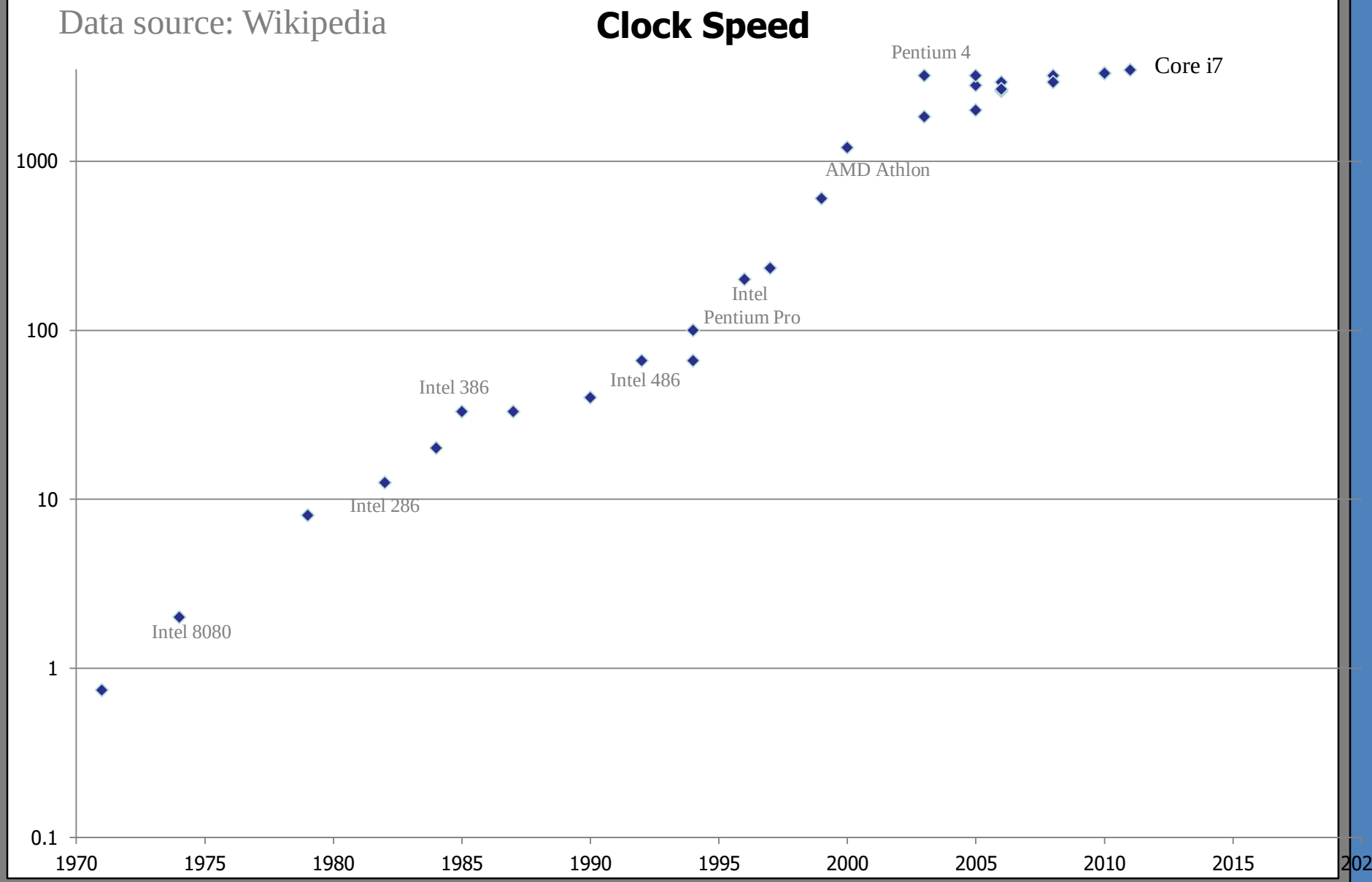
Parallel Algorithms

Data source: Wikipedia



Parallel Algorithms

Data source: Wikipedia



Parallel Algorithms

To make an algorithm run faster:

- Must take advantage of multiple cores.
- Many steps executed at the same time!

Parallel Algorithms

Different types of parallelism:

- multicore
  - on-chip parallelism: synchronized, shared caches, etc.
- multisocket
  - closely coupled, highly synchronized, shared caches
- cluster / data center
  - connected by a high-performance interconnect
- distributed networks
  - slower interconnect, less tightly synchronized

Parallel Algorithms

Map-Reduce:

- Target: high-performance clusters
- Focus: data (not computation)

Inventor: Google

- processing web data

Today: ubiquitous (Amazon, Yahoo, Facebook, etc,.)

- Hadoop, etc.

Map-Reduce Model

Basic round:

1. Map: process each (key, value) pair
2. Shuffle: group items by key
3. Reduce: process items with same key together

Plan:

Load data from disk.

Execute several rounds.

Save (key, value) pairs, sorted by key.

Parallel Algorithms

A huge amount of ongoing research on how to process big graphs fast in parallel…

A huge amount of ongoing research on how to efficiently find shortest paths, spanners, MSTs, and more in a network…

What's next?

Topic 1: Searching and Sorting

Topic 2: Trees

Topic 3: Hashing

Topic 4: Shortest Paths

Topic 5: Minimum Spanning Trees

What next?

I want to learn more neat algorithms!

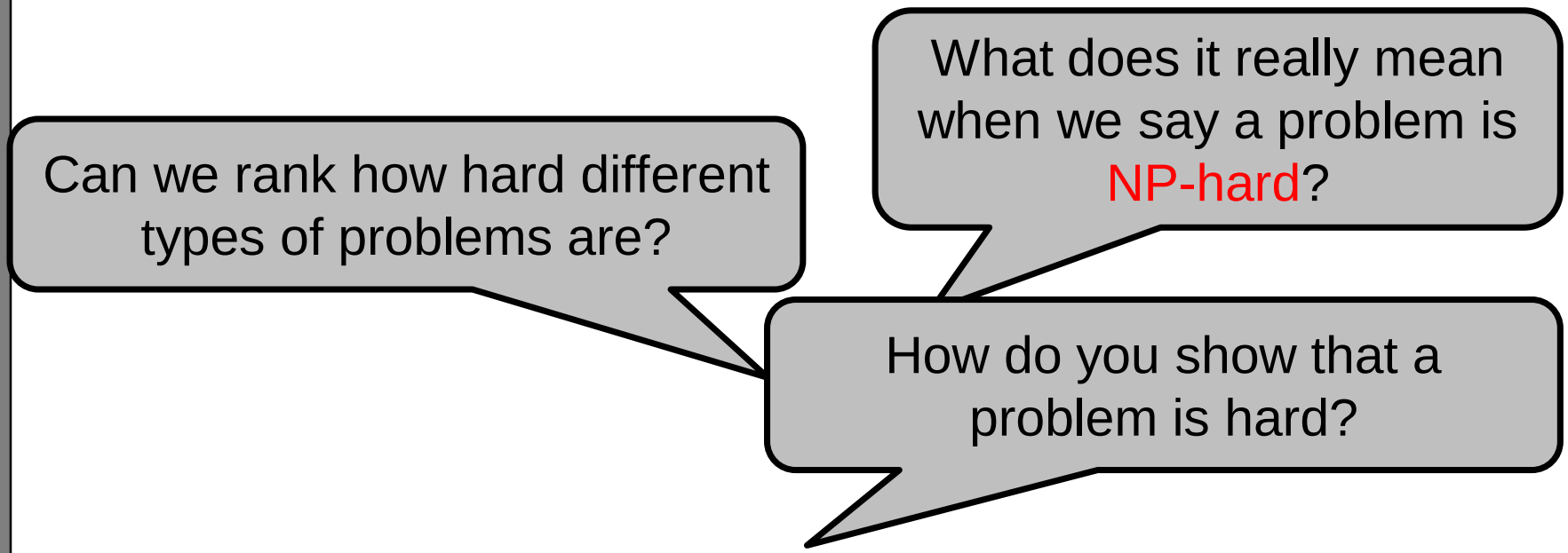
Algorithms modules:

- CS3230: Design and Analysis of Algorithms
- CS3233: Competitive Programming
- CS4231: Parallel and Distributed Algorithms
- CS4234: Optimization Algorithms
- CS4261 Algorithmic Mechanism Design
- CS4268 Quantum Computing
- CS5234: Combinatorial and Graph Algorithms
- CS5237: Computational Geometry
- CS5330: Randomized Algorithms

What next?

Theory:

- CS4232: Theory of Computation
- CS3234: Logic for Proofs and Programs
- CS4269 Fundamentals of Logic in Computer Science
- CS5230: Computational Complexity



What next?

I want to build cool systems!

Software engineering modules:

- CS2103: Software engineering
- CS4211: Formal methods for software engineering
- CS4218: Software testing and debugging

System design and programming modules:

- CS3216: Software development on evolving platforms
- CS3217: Software engineering on modern application platforms

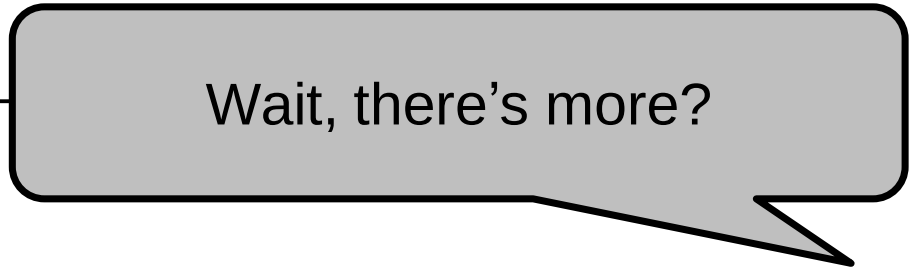
What next?

I want to build an AGI that will take over the world and subjugate the human race!

Machine Learning and AI:

- CS2109: Intro to Machine Learning and AI
- CS4243 Computer Vision and Pattern Recognition
- CS4244 Knowledge Representation and Reasoning
- CS4246 AI Planning and Decision Making
- CS4248 Natural Language Processing

What next?



Specialized modules:

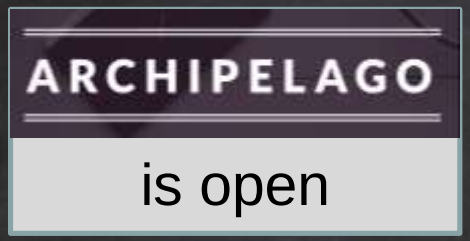
- Distributed Systems
- Computer Security
- Game Design
- Computer Graphics
- Computational Biology
- Wireless computing and sensor networks
- Etc…

What next?

Research:

- UROP (Undergraduate Research Opportunities)
- Turing Programme
- FYP

Quick Review…



Which of these quotes is NOT by a famous computer scientist?

Who is known for his paper: "Goto Statement Considered Harmful."

What algorithm was originally invented to improve Russian-to-English translation?

What problem did the most ancient known algorithm solve?

What was the first object-oriented programming language?

In what year did Adelson-Velsky and Landis publish their breakthrough paper on AVL trees?

Algorithm Design Tips

8

Small examples

- At least 5 items/nodes.
- Avoid special cases.
- Understand why problem is hard.
- Look for useful structure.

7

Naïve solutions:

- Simple algorithms are useful.
- Baseline: complicated algorithm should be better than the simple one!
- De-optimize your algorithm.

6

Special cases:

- Try easier versions of the problem.
- Trees? Planar graphs?
- Do those cases generalize?

5

Best possible solution?

- What is your target.
- Look for natural "lower bounds."
- How good is good enough?

4

Metrics:

- How to measure goodness of your solution?
- What do you want?
- Can you get better performance by looking at a different metric?

3

Invariants

- What is always true?
- What does the problem require?
- What does the algorithm guarantee?

2

Binary search:

- Many, many algorithms can be optimized by using binary search.

1

Divide-and-conquer:

- Look for subproblems.
- Look for structure.
- Wishful thinking: if only we could solve X, then we could…

The End

Goals:

- Apply algorithmic thinking and techniques.
- Describe the structure and operation of different data structures.
- Assess the suitability of different data-structures and algorithms.
- Adapt existing data-structures and algorithms.

The End

Goals:

- Apply algorithmic thinking and techniques.
- Describe the structure and operation of different data structures.
- Assess the suitability of different data-structures and algorithms.
- Adapt existing data-structures and algorithms.

I hope everyone has had fun…

The End

Goals:

- Apply algorithmic thinking and techniques.
- Describe the structure and operation of different data structures.
- Assess the suitability of different data-structures and algorithms.
- Adapt existing data-structures and algorithms.

I hope everyone has had fun…

If you ever want to chat about algorithms, just drop by either of our offices…

And it's over… congratulations!