CS2040S
Data Structures and Algorithms
(e-learning edition)

What comes next?

Algorithms Everywhere

Web browser:
Parsing
Substring manipulation (Week 7)
XML trees (Week 5)

Internet routing:
TCP (congestion control)
IP routing
BGP (Bellman-Ford, Week 9)
Content caching (Week 7)
DNS

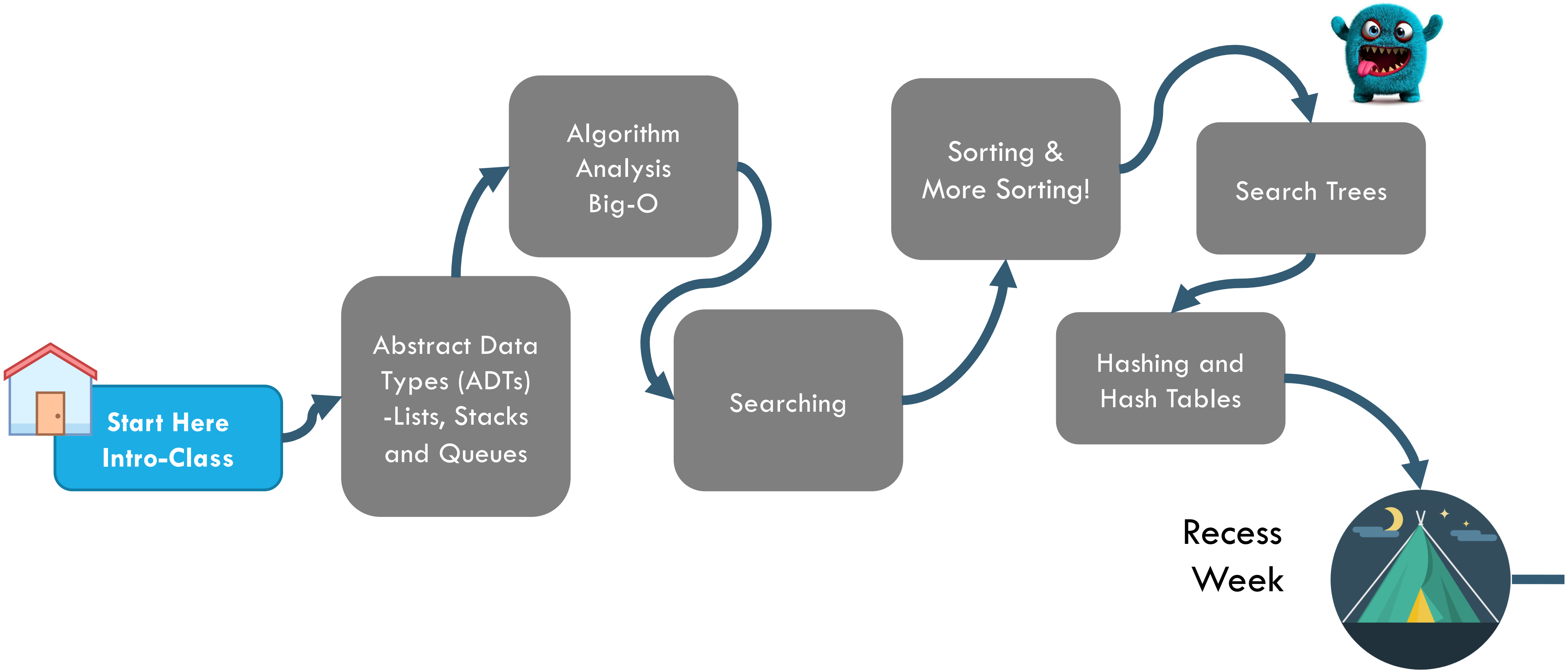
Google:
PageRank
(Week 10)
String matching
(Week 7)

Web servers:
Load balancing (PS 2)
Scheduling (Week 8)
Memory allocation

Database:
B-trees (Week 5)
Search (Week 2)
Sorting (Week 3)

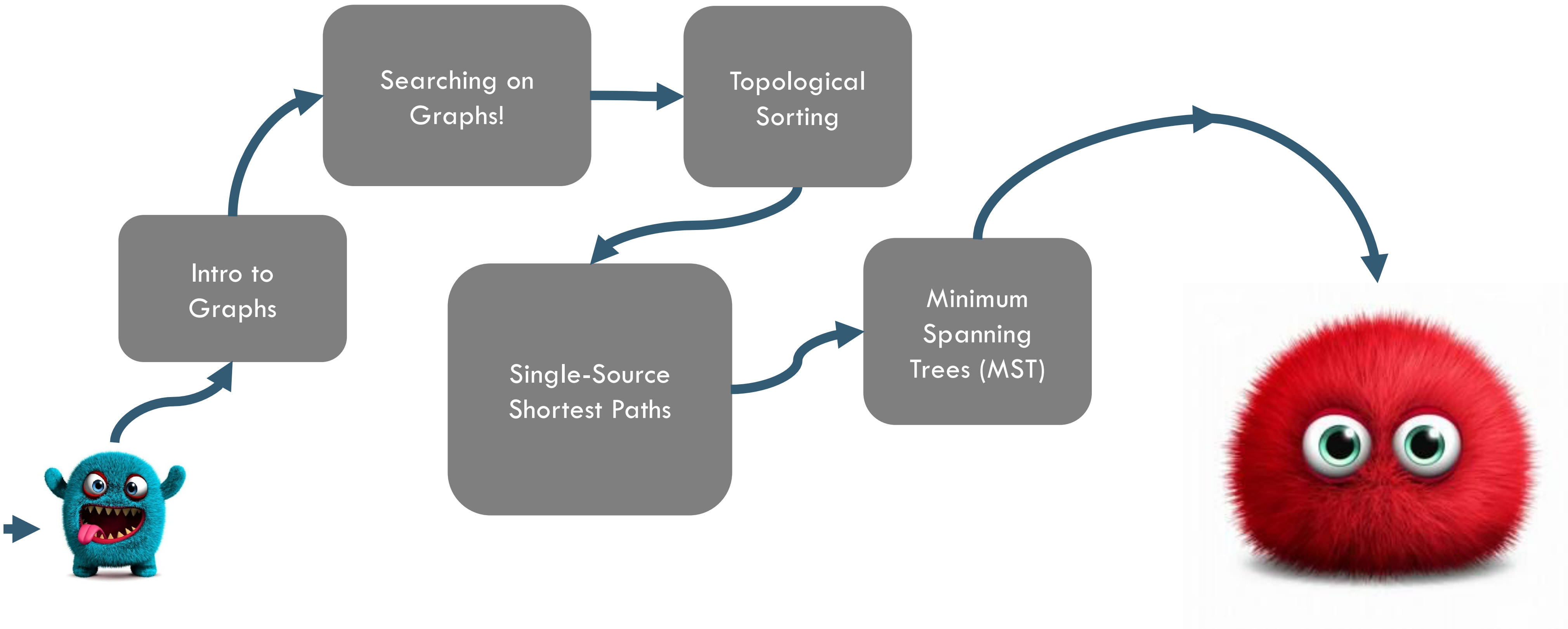
What is CS2040S about?

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:

Fast

Efficient

Cache efficient

Correct

Fair

Elegant

Trade Offs

Modular

Small Memory

Easy to understand

Parallel

Secure

Easy to maintain

Privacy preserving

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 Optimization

Topic 2: Sorting

Topic 3: Trees

Topic 4: Hashing

Topic 5: Shortest Paths

Topic 6: Minimum Spanning Trees

Topic 1: Searching and Optimization

Binary Search:

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

Newton's Method:

- Fast way to search/max/min for convex, twice-differentiable data.

Gradient Descent:

- Fast way to search/max/min for convex data.
- Fast way to optimize wide variety of functions.

Huge area of research and development:

Primal-Dual Methods

Stochastic gradient descent

Randomized rounding

Gradient Quantization

Parallel SGD

Adagrad

Optimization

Interior Point Methods

Minibatch

Momentum method

Coordinate descent

Subgradient

Online algorithms

Relaxation

Conjugate gradient

Adaptive moment estimation

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

$$\max\ (A+6B)$$

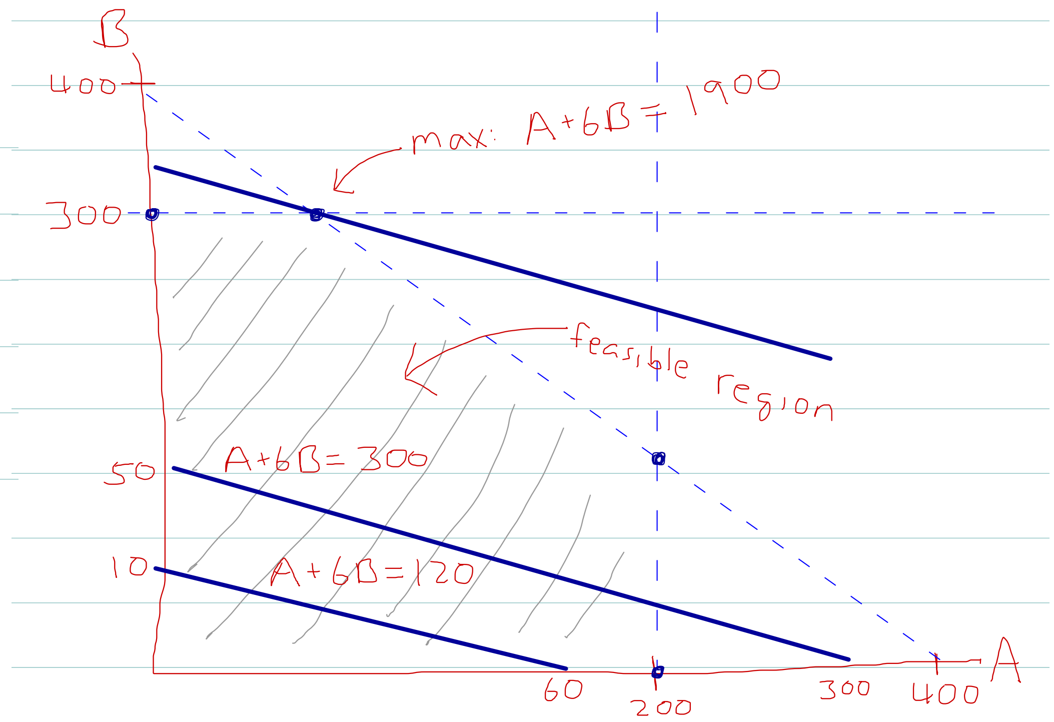
where: $A \le 200$

$B \le 300$

$A + B \le 400$

$A \ge 0$

$B \ge 0$



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 2: Sorting

Fast Sorting Algorithms:

- QuickSort
- MergeSort
- HeapSort

Key properties:

- Running time
- Space usage
- In-place

Sorting Faster!

Multi-pivot QuickSort…

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!

DataBricks TeraSort:

- 206 nodes
- 6,592 cores

Record (2014):

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

DataBricks PetaSort:

- 190 nodes
- 6,080 cores

Record (2014):

DataBricks (Spark) sorts 1PB of data in 234 minutes.

Sorting Faster!

Tencent Sort:

- 512 nodes
- 5,024 cores

Record (2016):

98.8 seconds

Sorting Faster!

It's a race:

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

Topic 3: 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

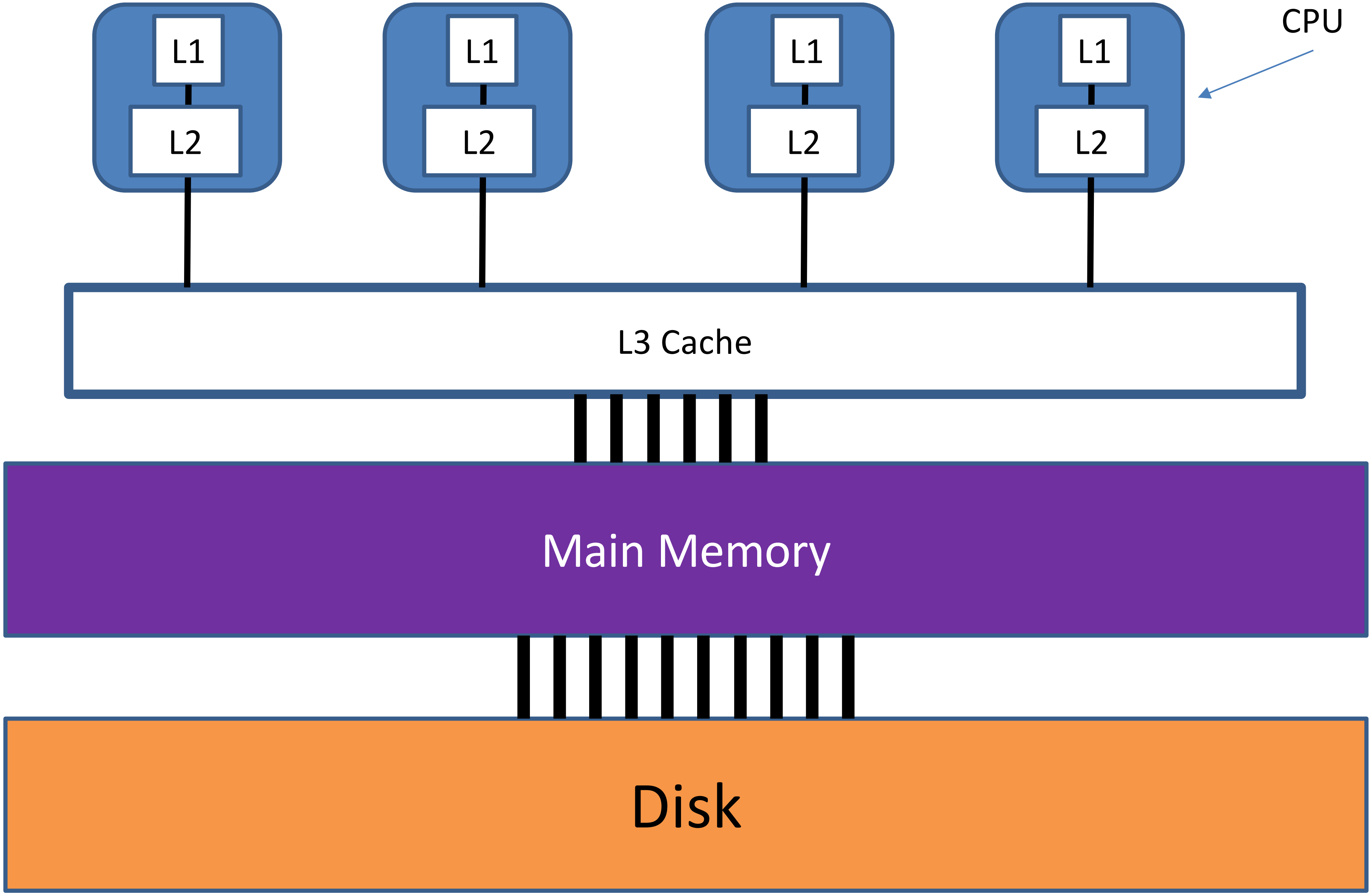
- ⇒ Scan all the data: O(n)
- ⇒ Search: O(n)

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!

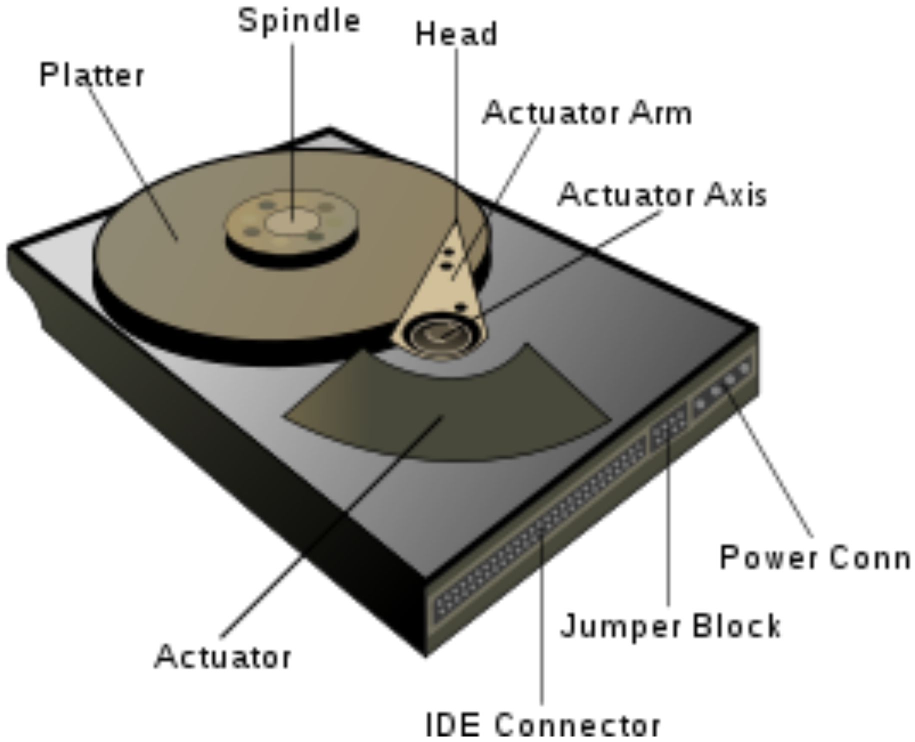
A Real Computer (?)



Disks

Where is most data stored? Hard disk!

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



Two step access:

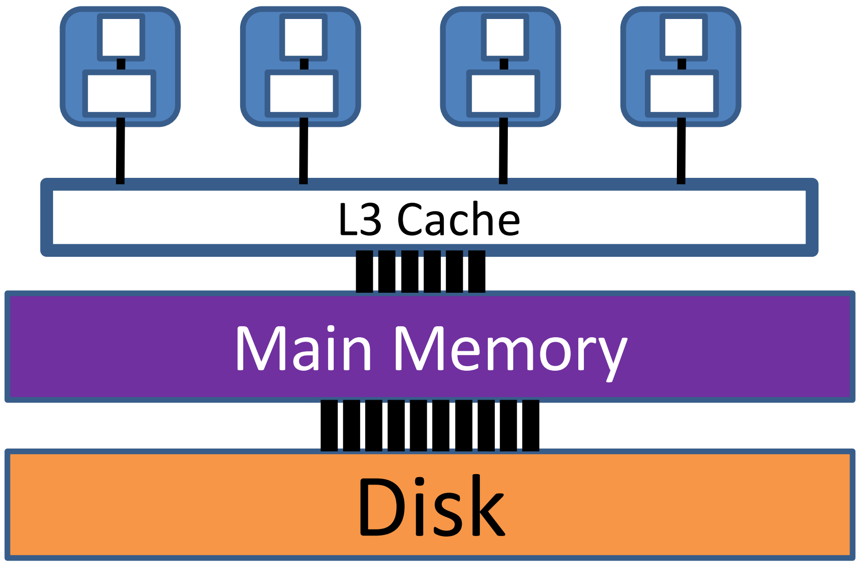
1. seek *(find right track)*
2. read track

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.



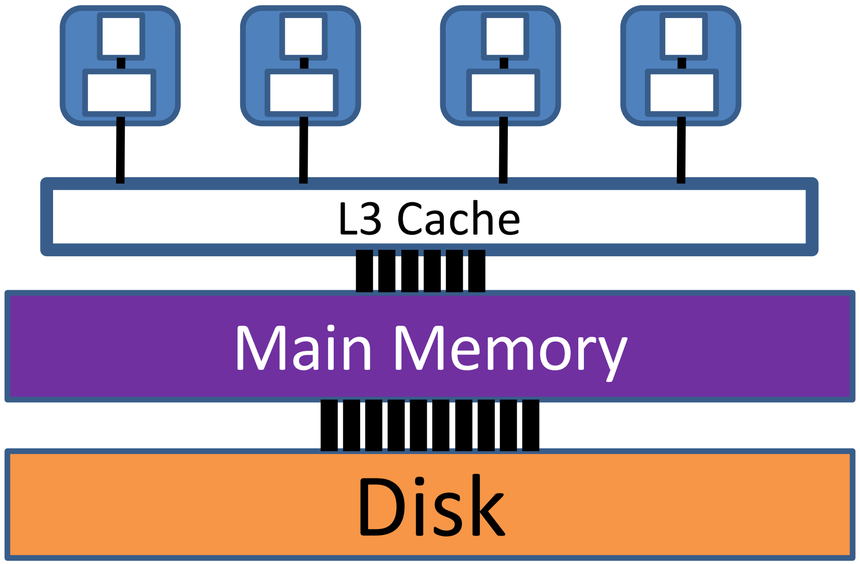
Haswell Architecture

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:

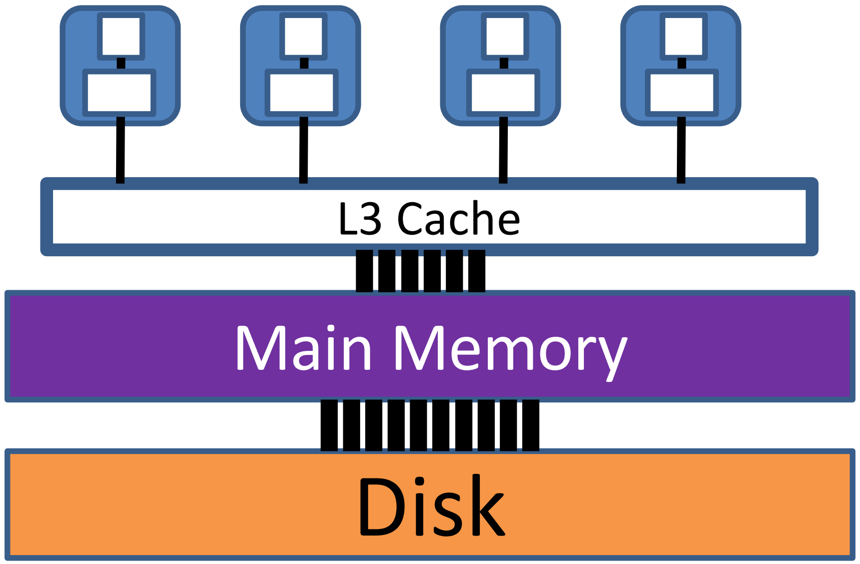
What fraction of operations ”hit” each cache?

⇒ 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)

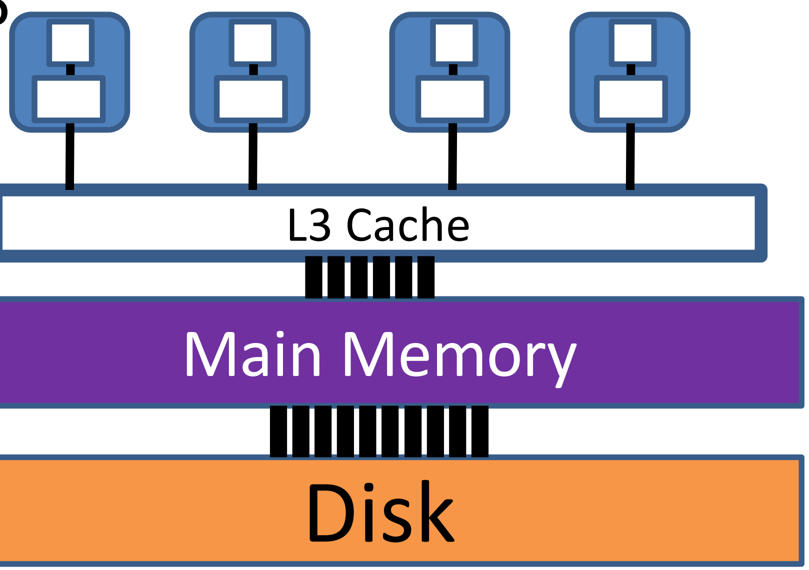
⇒ 0.2% disk (20,000,000 cycles)

Just an example..

What fraction of time for each cache?

⇒ 99.98% waiting for disk

Disk is much, much worse!



Where is the bottleneck?

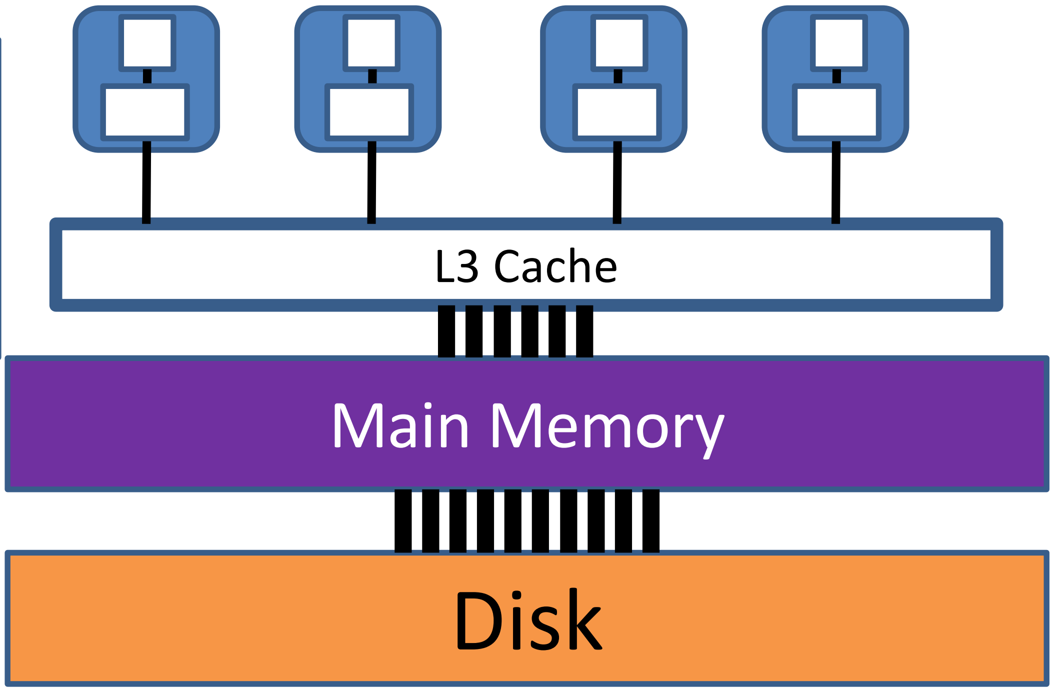
The bottleneck depends on the application:

- Small working set data lives in L1/L2 cache ➔ fast.
- Medium working set data lives in main memory ➔ bottleneck is memory latency.
- Big data lives on disk ➔ bottleneck is disk latency / bandwidth.

For most applications, one level dominates the cost.

(Costs grow fast!

Largest level dominates.)



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.

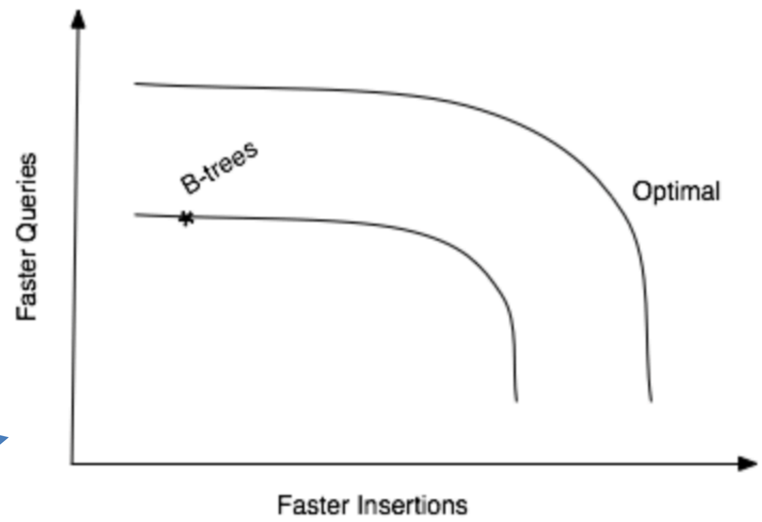
Faster Trees?

Goal:

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

“Write-optimized data structure.”

Percona advertisement graph
(not technical)



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

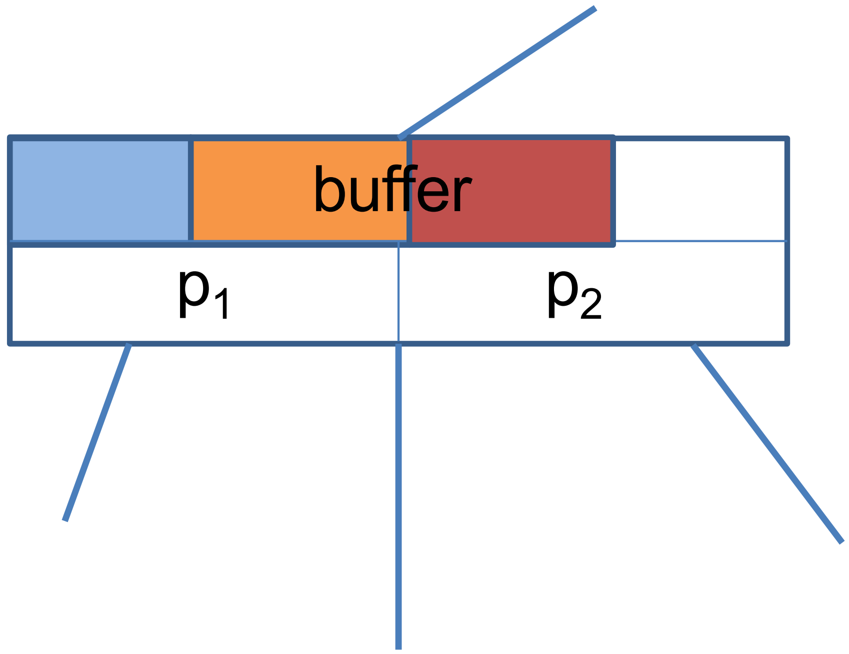
Buffer Tree

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 <u>fast</u> searches, and <u>super-fast</u> insertions/deletions.

"Write-optimized data structure."

Examples:

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

See: BetrFS

Hot area of DBS research today…

Topic 4: Hash Tables

Two key types of hash tables:

- Chaining
- Open Addressing

Hash Sets / Filters:

- Fingerprint Hash Tables
- Bloom Filters

Better Collision Resolution

Chaining:

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

Cuckoo Hashing:

Neat, new*er* 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:

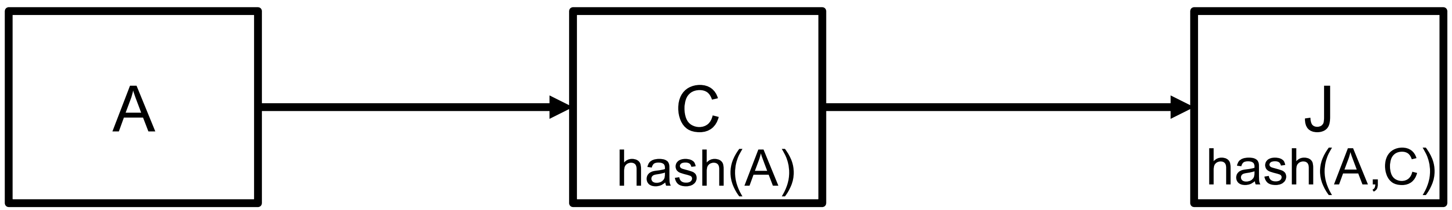
- Example: MD6,
- Example: SHA-512

Are these secure?

Blockchains

Blockchain = Hashchain

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



Topic 4: Hash Tables

Two key types of hash tables:

- Chaining
- Open Addressing

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

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

Topic 5: Shortest Paths

Several situations:

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

Topic 5: Shortest Paths

Key algorithms:

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

Topic 6: 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

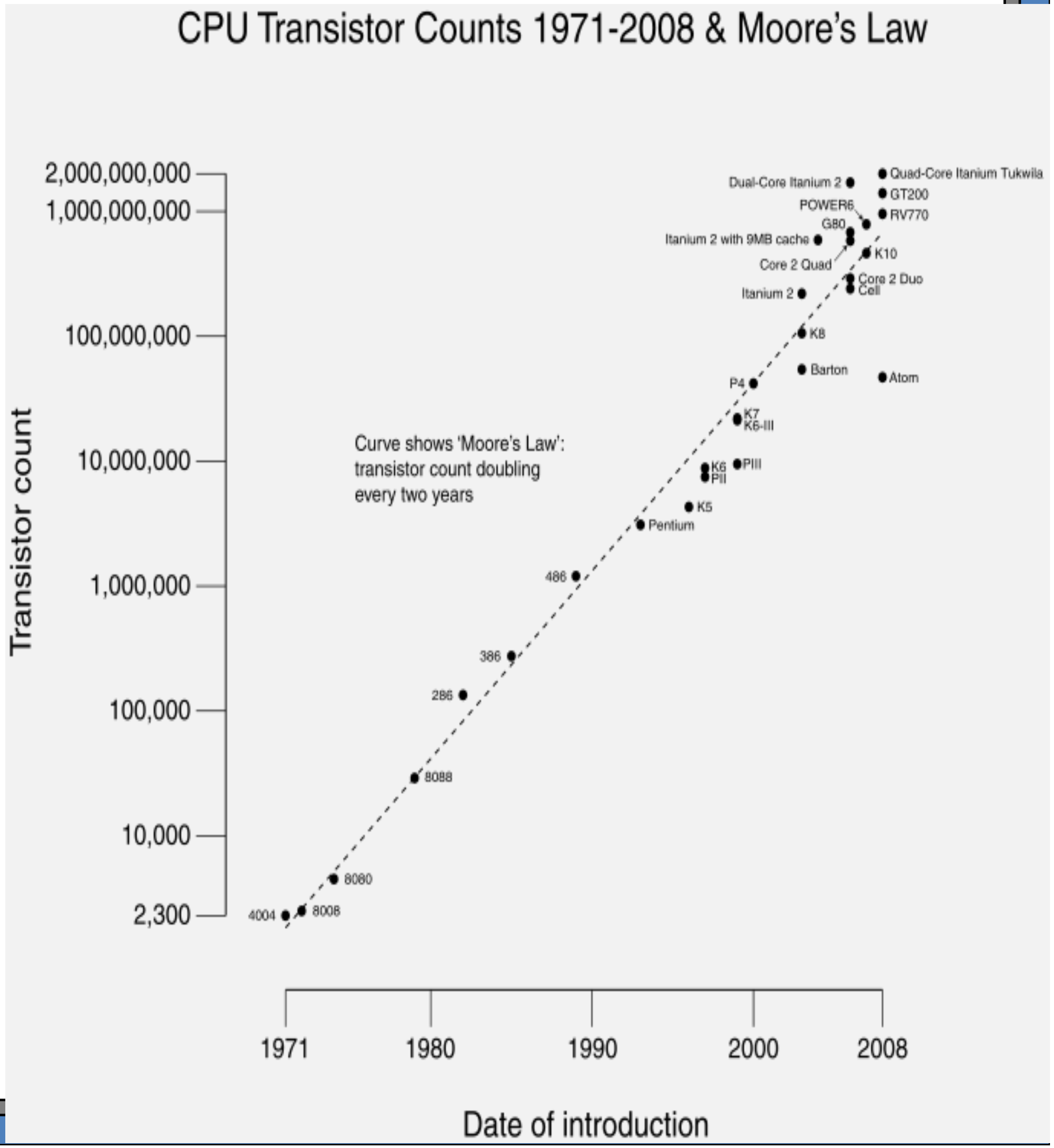
Moore's Law

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

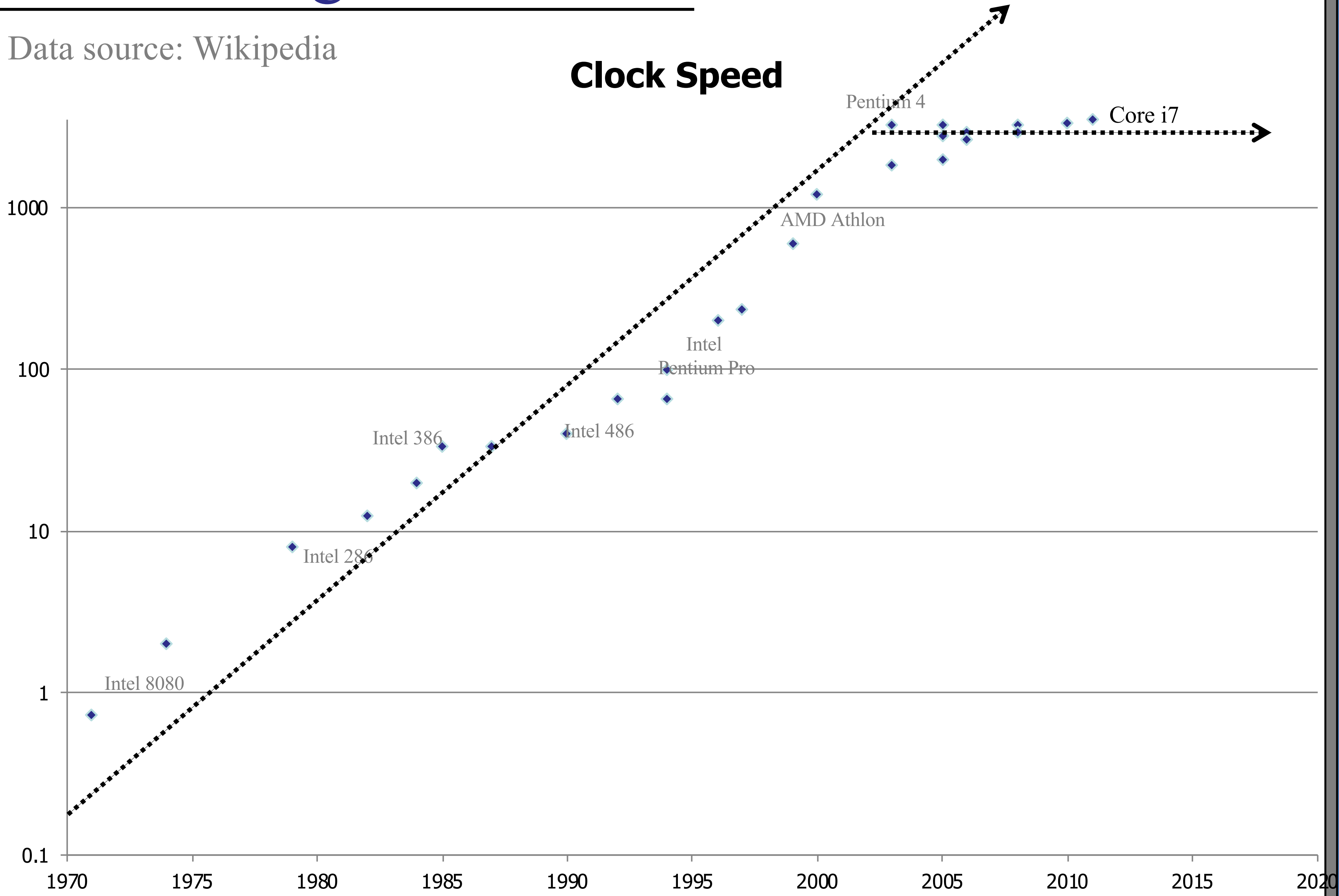
More transisters == faster computers?

- 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

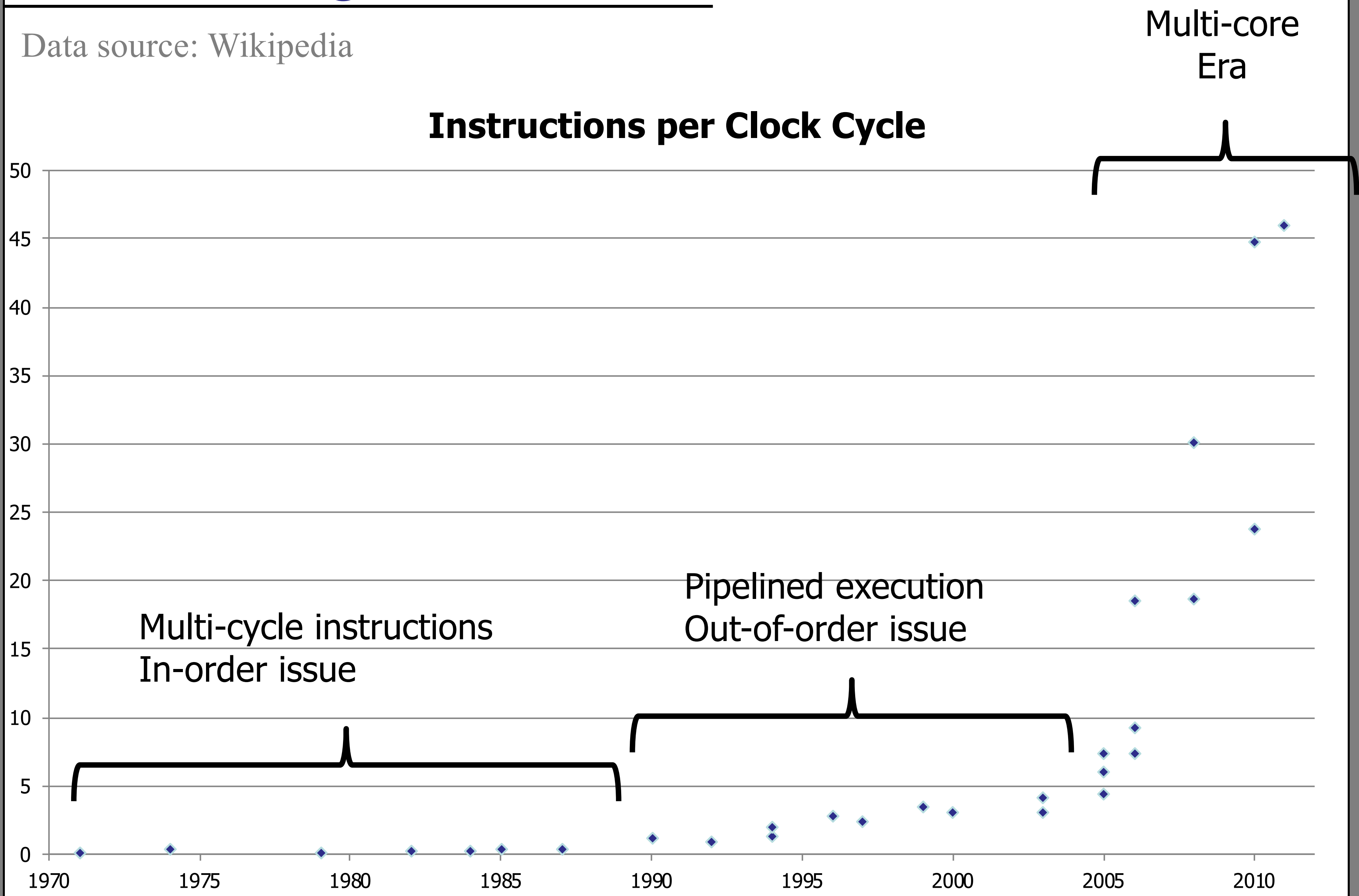
Conclusion:

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

What do we do?

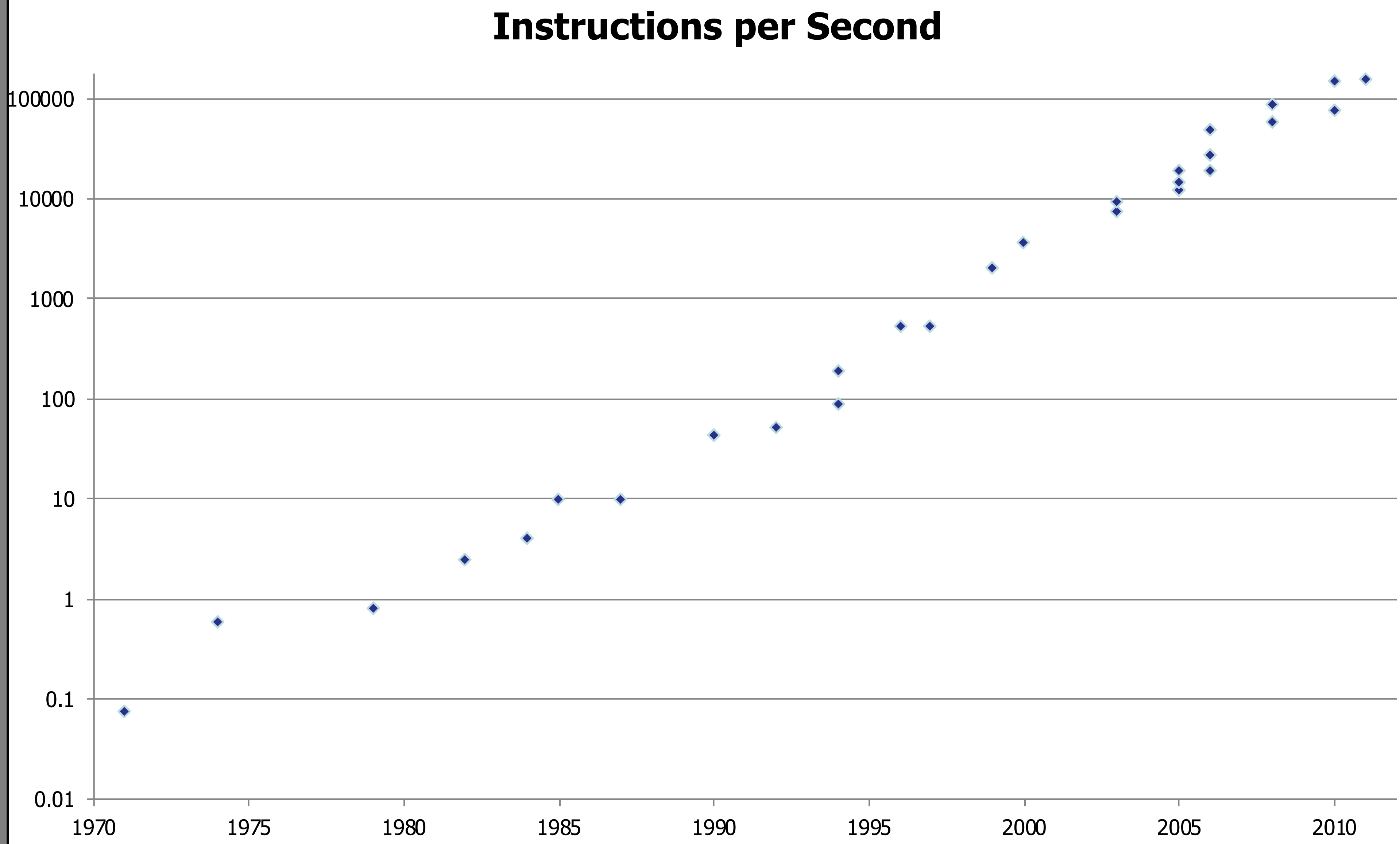
Parallel Algorithms

Data source: Wikipedia



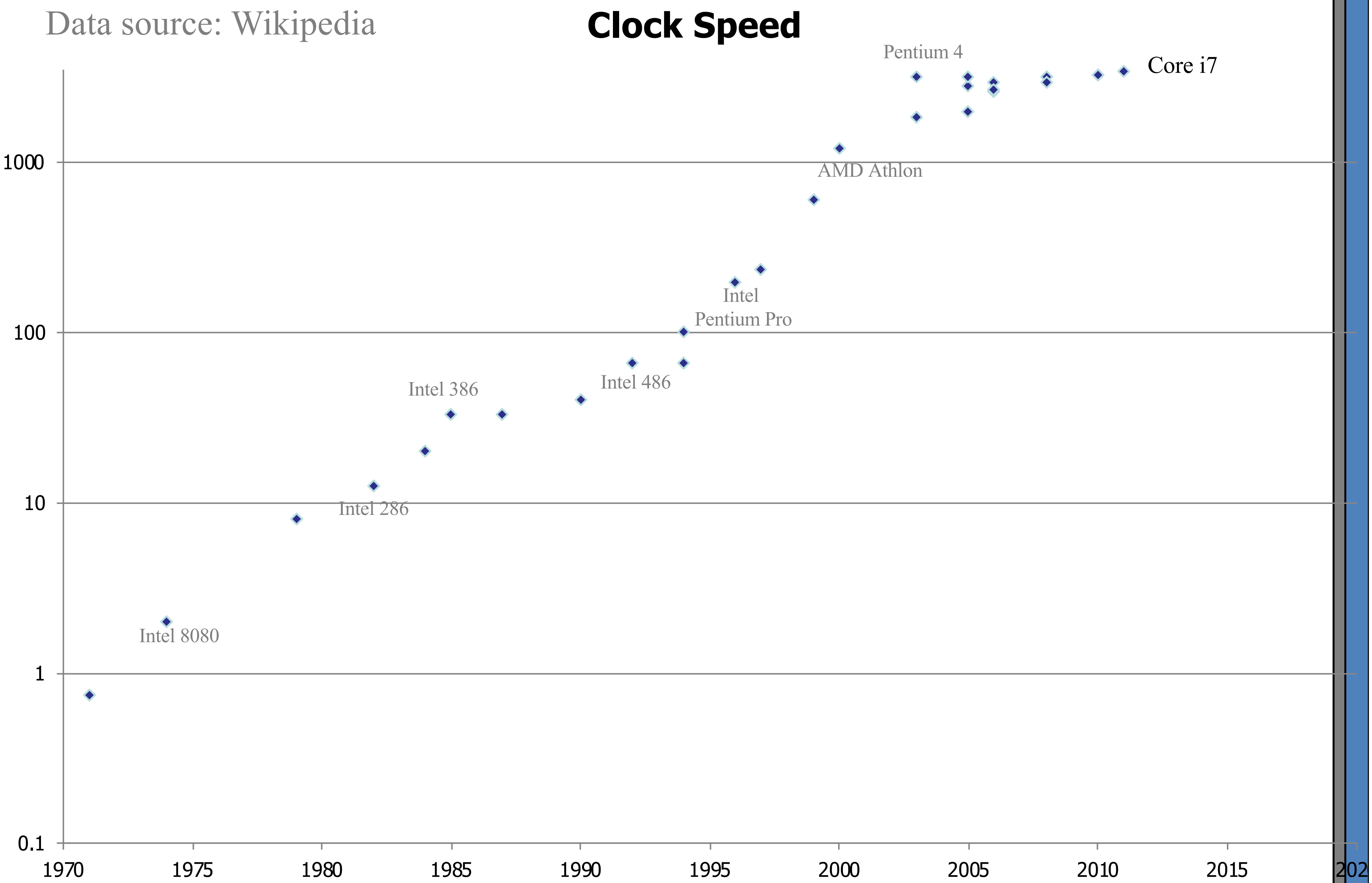
Parallel Algorithms

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.

Map-Reduce

Boruvka's Algorithm

Repeat:

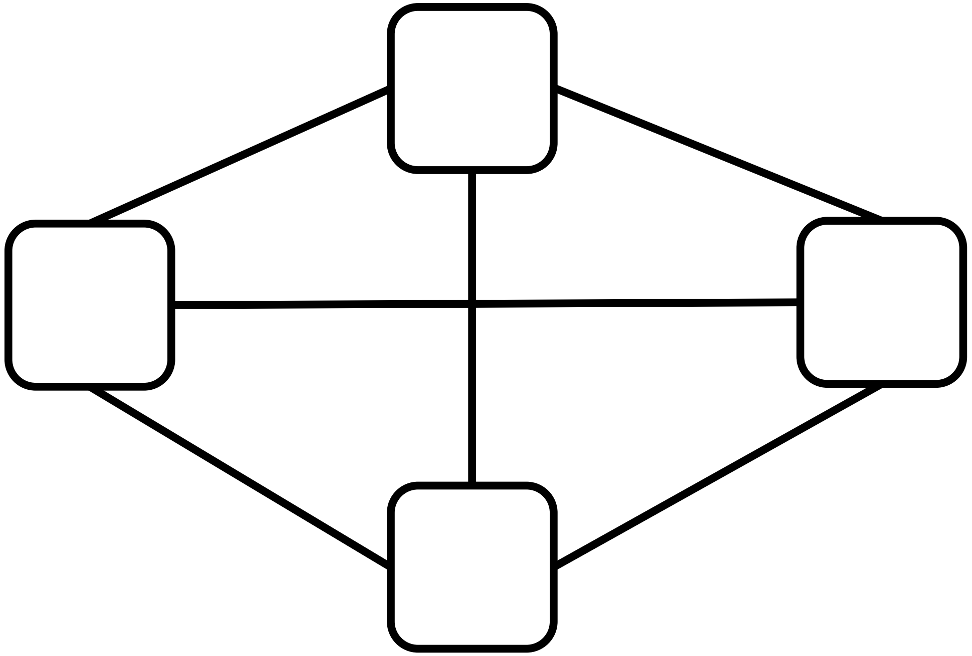
1. Find min-weight-outgoing-edge for each node.
2. Merge components.

Time: $O(\log n)$ rounds of map-reduce

Parallel Algorithms

k-Machine Cluster:

- k servers: system is a collection of cores/CPUs/etc.
- all-to-all communication: communicate via messages
- bandwidth limit B: limited data transfer



k = 4

k-Machine Model

Boruvka's Algorithm

Repeat:

1. Find min-weight-outgoing-edge for each node.
2. Merge components.

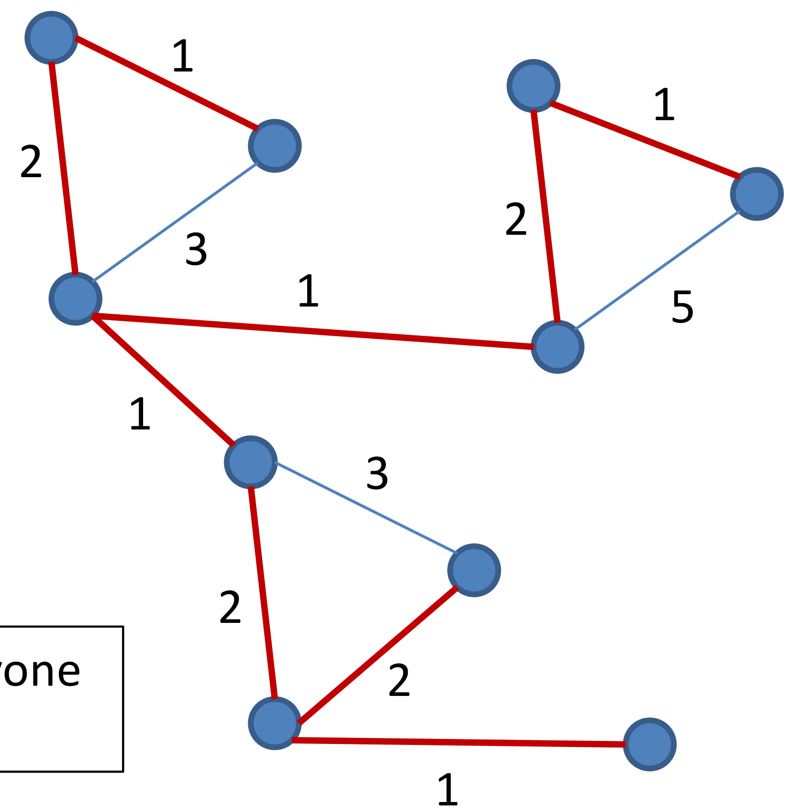
Time: $O(n \log^c(n)/k^2)$

Fully Distributed Model

Network Model

Assume each node in the graph is its own machine.

Each edge in the graph is a real communication edge.



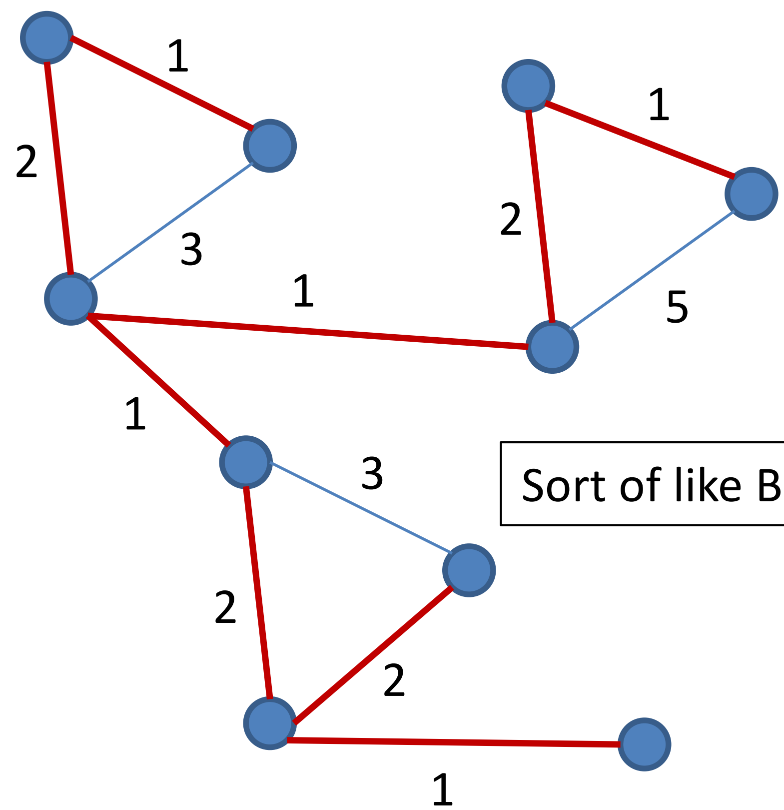
Cannot send message to everyone like in k-machine model.

Fully Distributed Model

Network Model

Assume each node in the graph is its own machine.

Each edge in the graph is a real communication edge.



Sort of like B=log n

Each edge carries 1 message per round.

Fully Distributed Model

Boruvka's Algorithm

Repeat:

1. Find min-weight-outgoing-edge for each node.
2. If component is $< n^{1/2}$ then aggregate MWOE in component. Otherwise aggregate on BFS tree.
3. Merge components.

Time: $O((D + n^{1/2})\log n)$

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 Optimization

Topic 2: Sorting

Topic 3: Trees

Topic 4: Hashing

Topic 5: Shortest Paths

Topic 6: Minimum Spanning Trees

What next?

Algorithms modules:

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

Theory:

- CS4232: Theory of Computation
- CS5230: Computational Complexity

What next?

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?

Specialized modules:

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

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 my ~~office~~ zoom meeting…

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 my ~~office~~ zoom meeting…

And it's over… congratulations!