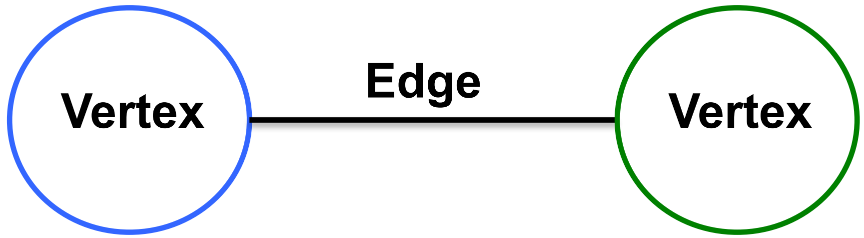
6.886: Graph Analytics

Lecture 1
Introduction

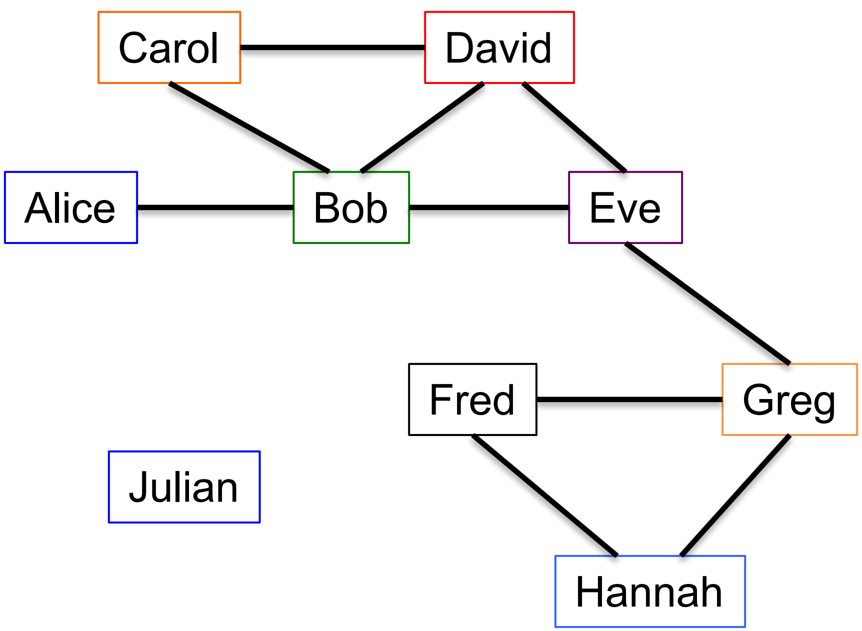
Julian Shun

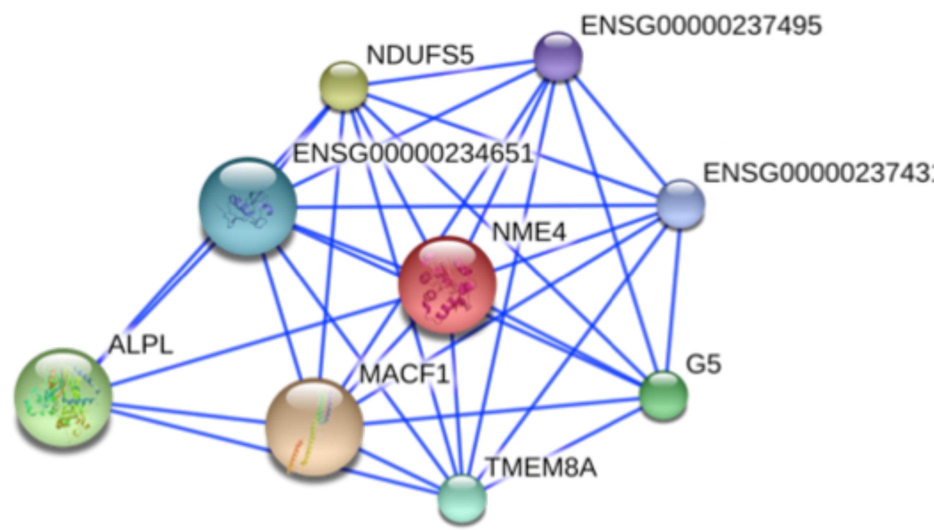
February 7, 2018

What is a graph?



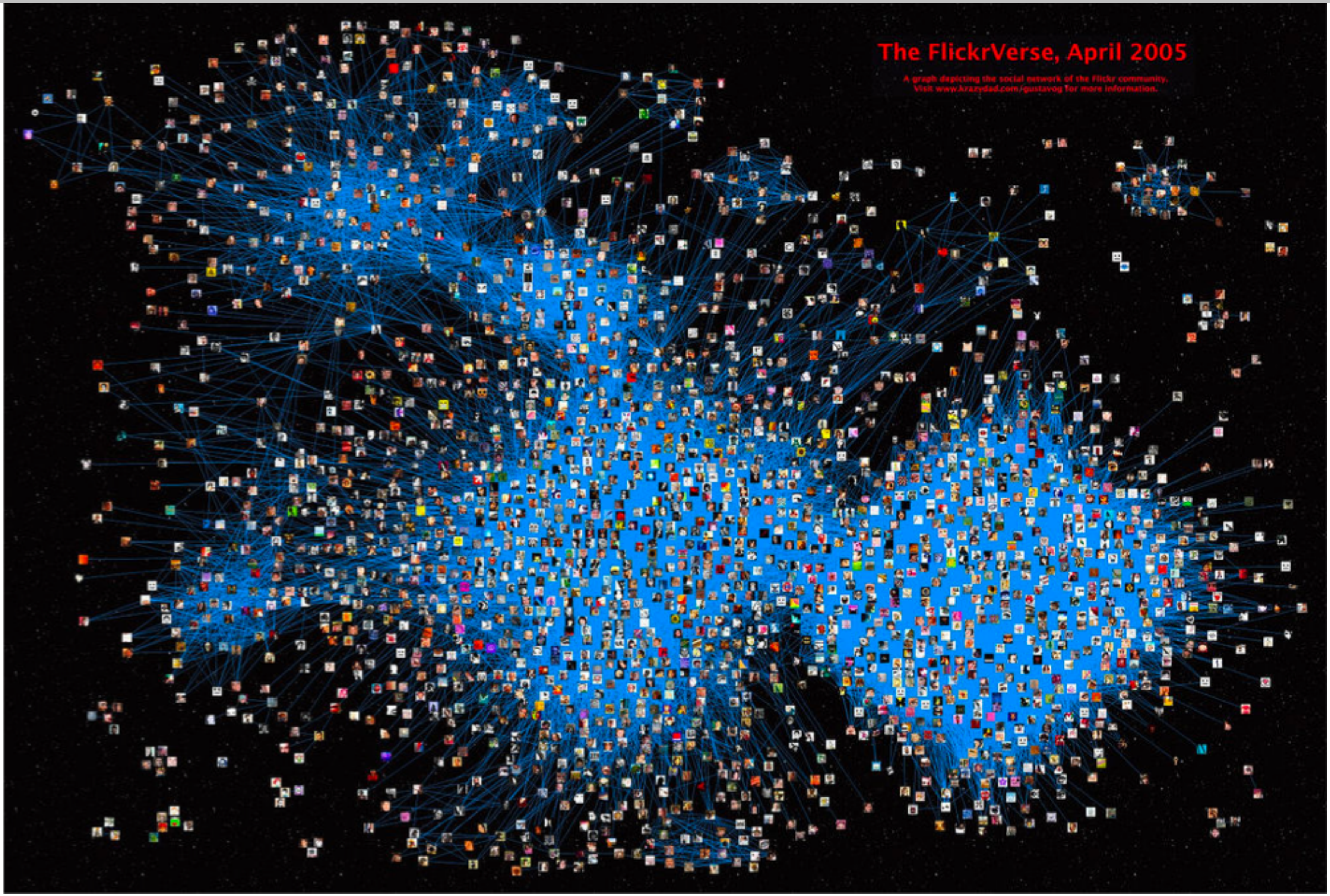
- **Vertices** model objects
- **Edges** model relationships between objects





https://commons.wikimedia.org/wiki/File:Protein_Interaction_Network_for_TMEM8A.png

Social networks



Collaboration networks

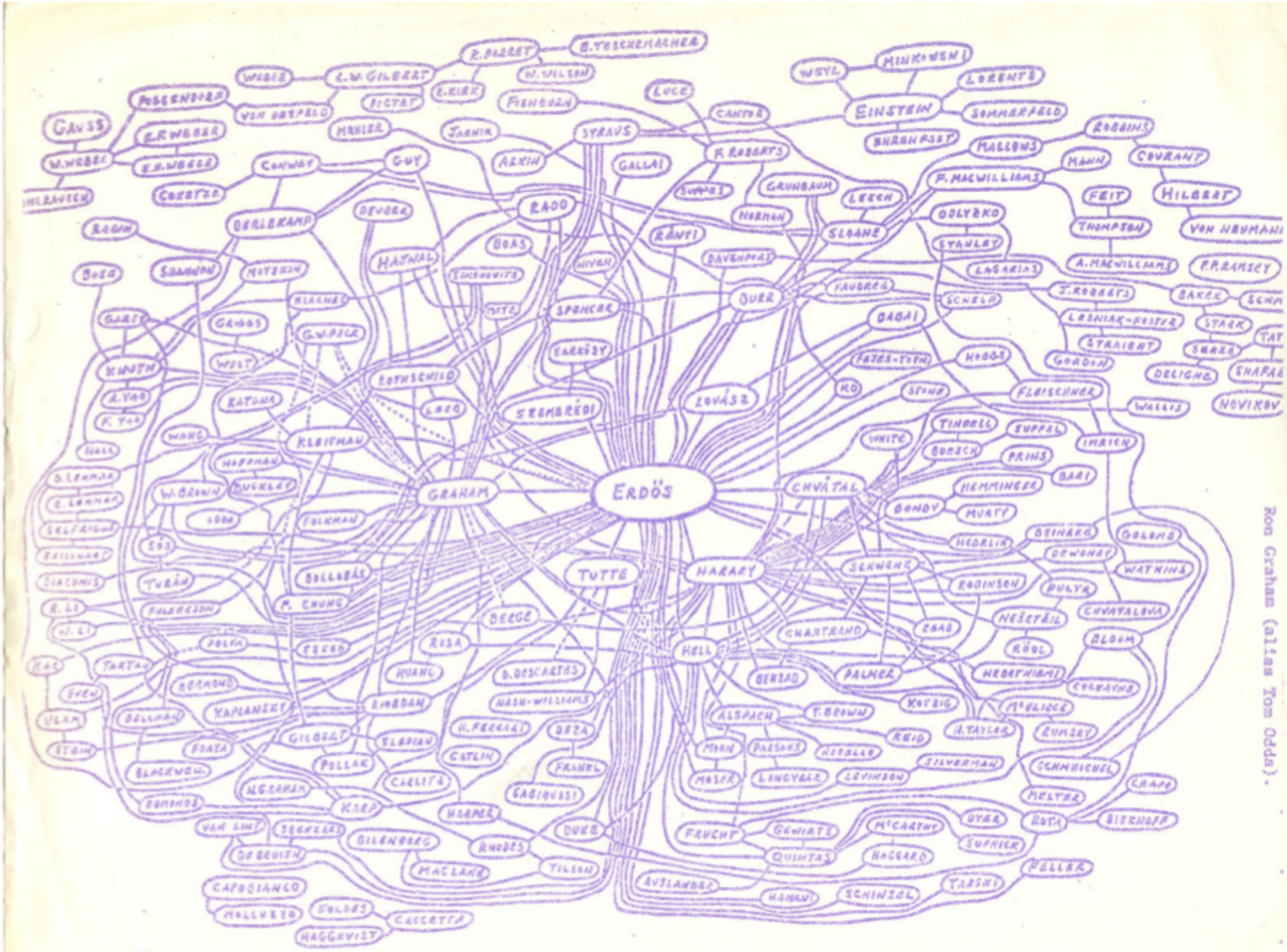
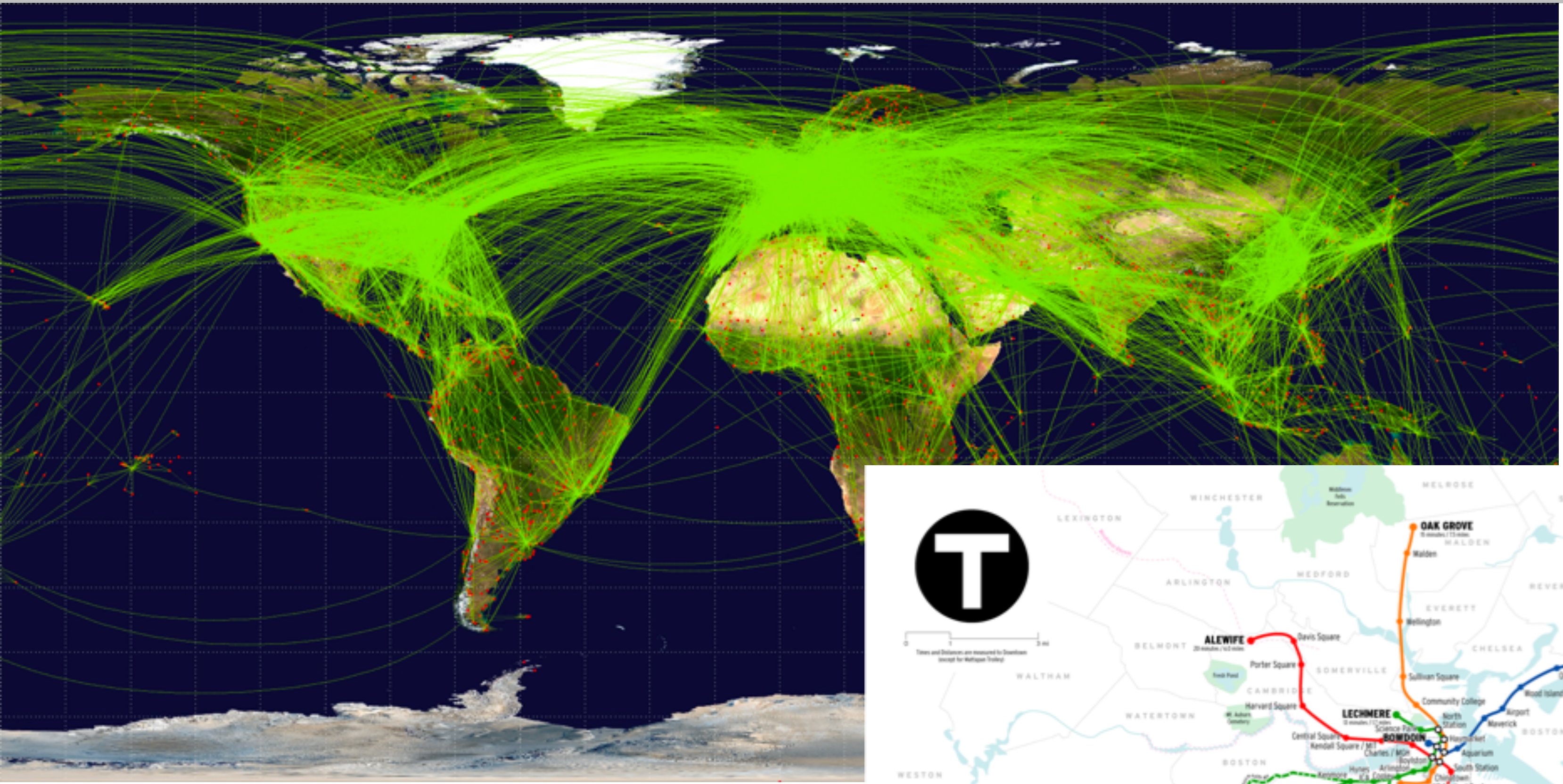
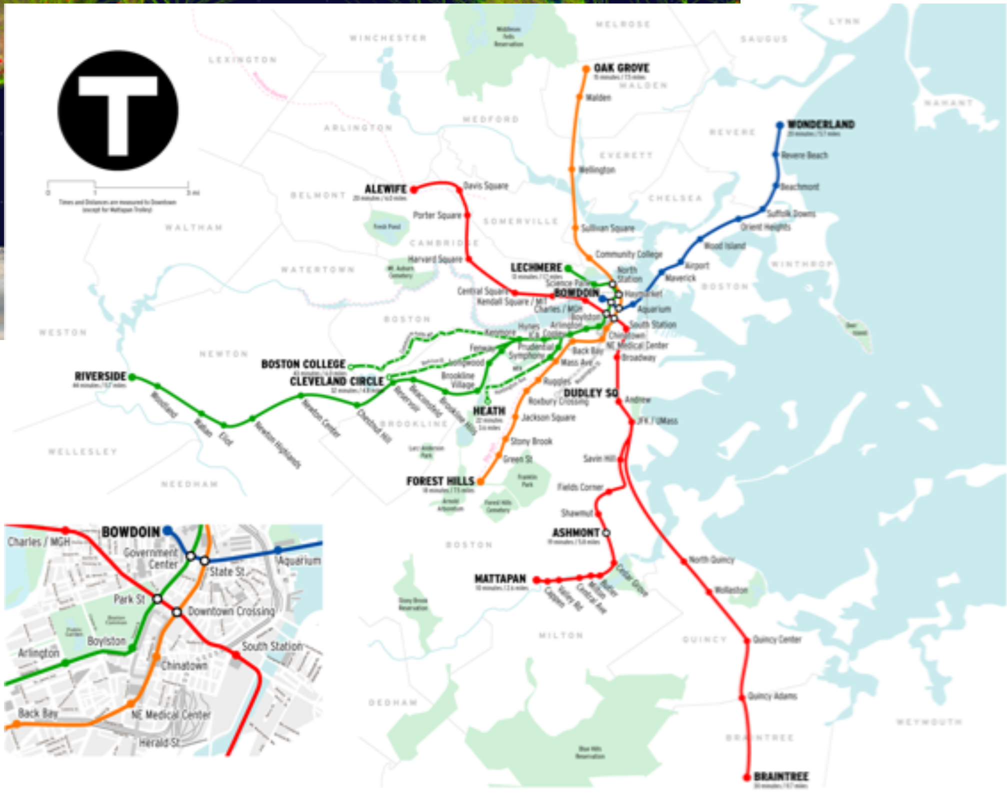


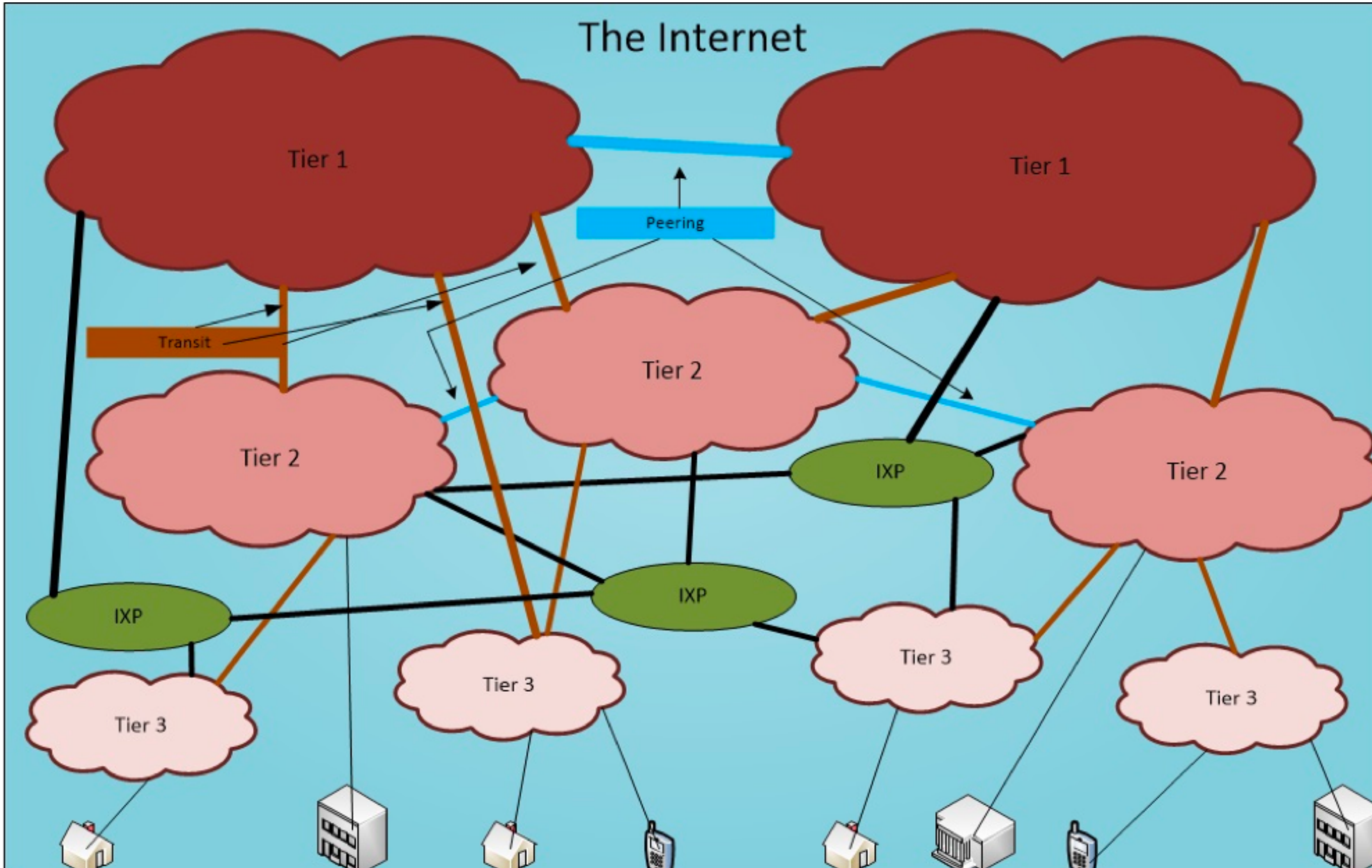
Figure 1
To appear in Topics in Graph Theory (F. Harary, ed.) New York Academy of Sciences (1979).

Transportation networks



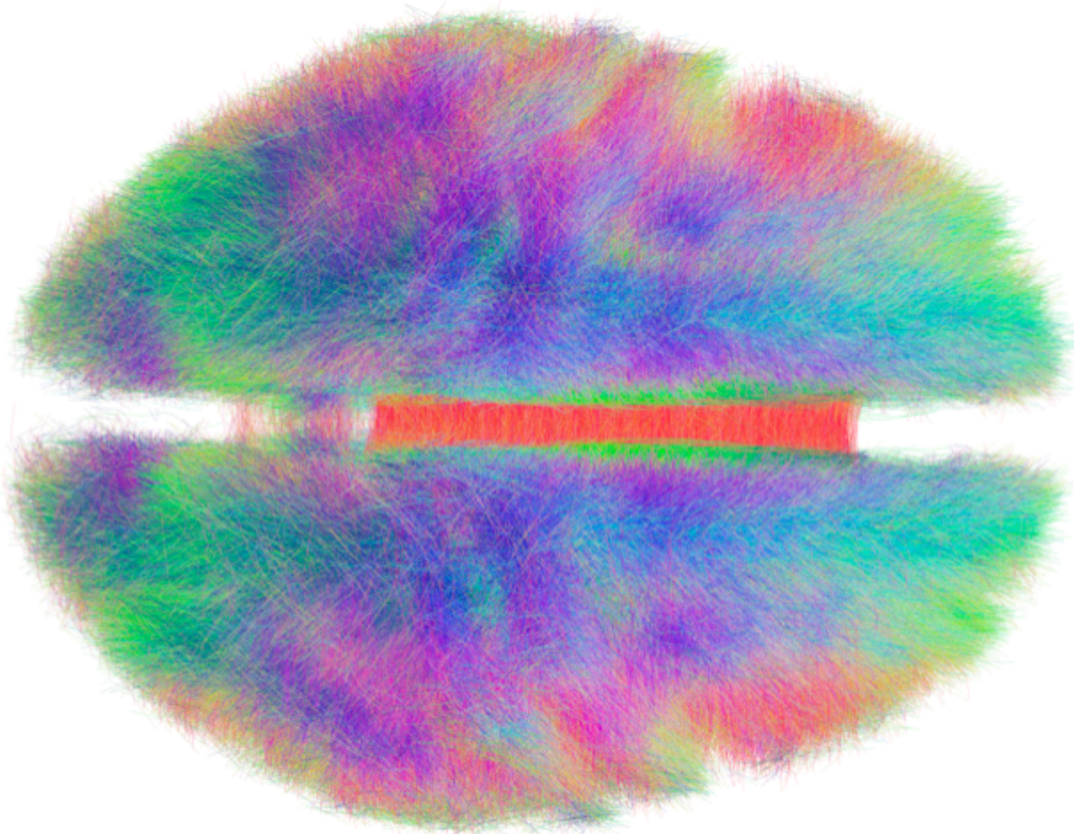


Computer networks



Source: rawbytes.com

Connectomics



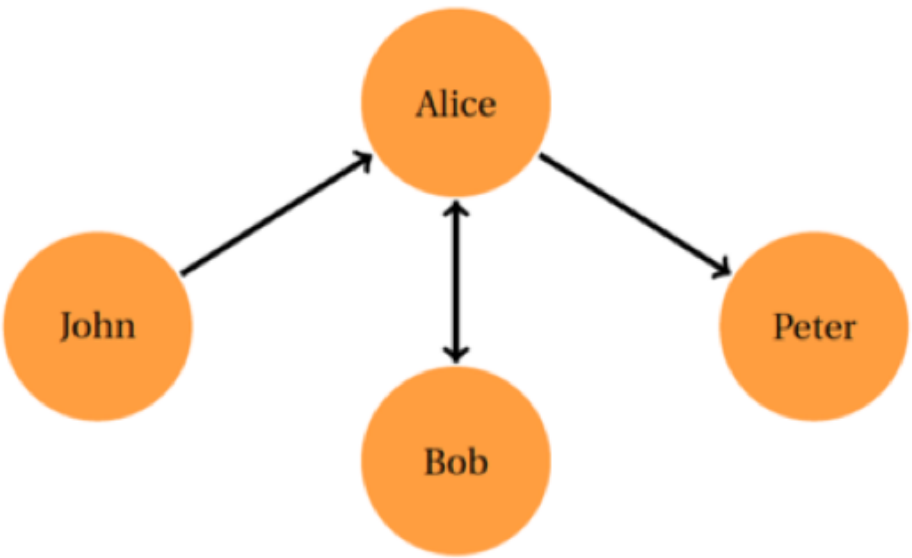
- Vertices are neurons, edges are synapses
- Roughly 10^{11} neurons and 10^{15} synapses in human brain

Other Applications

- Financial transaction networks
- Economic trade networks
- Food web
- Various types of biological networks
- Image segmentation in computer vision
- Scientific simulations
- Many more…

What is a graph?

- Edges can be directed
  - Relationship can go one way or both ways



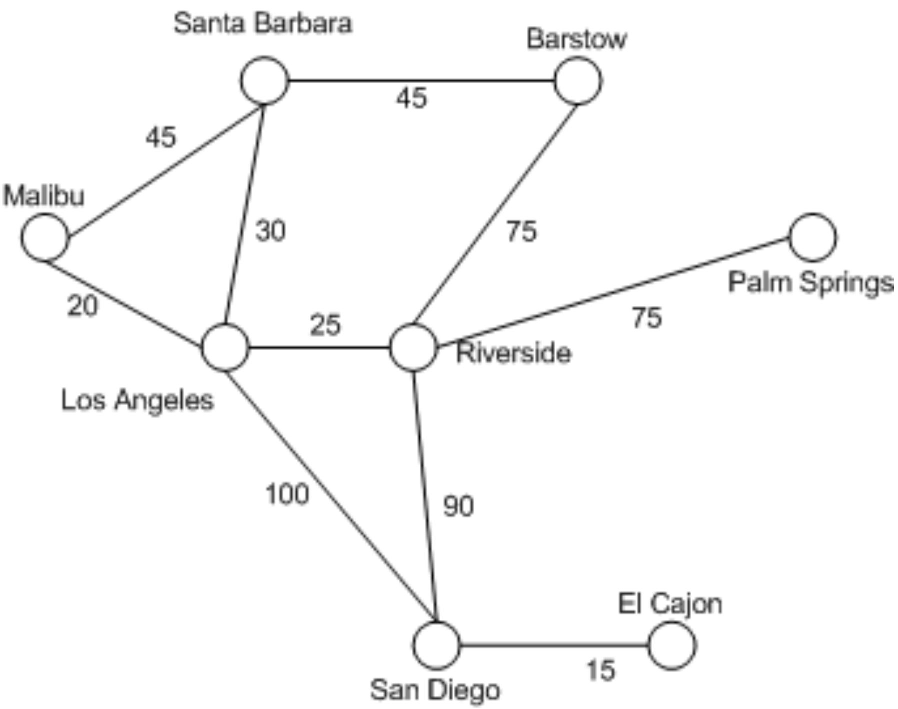
http://www3.nd.edu/~dwang5/courses/spring15/assignments/A1/Assignment1_SocialSensing.html

http://farrall.org/papers/webgraph_as_content.html

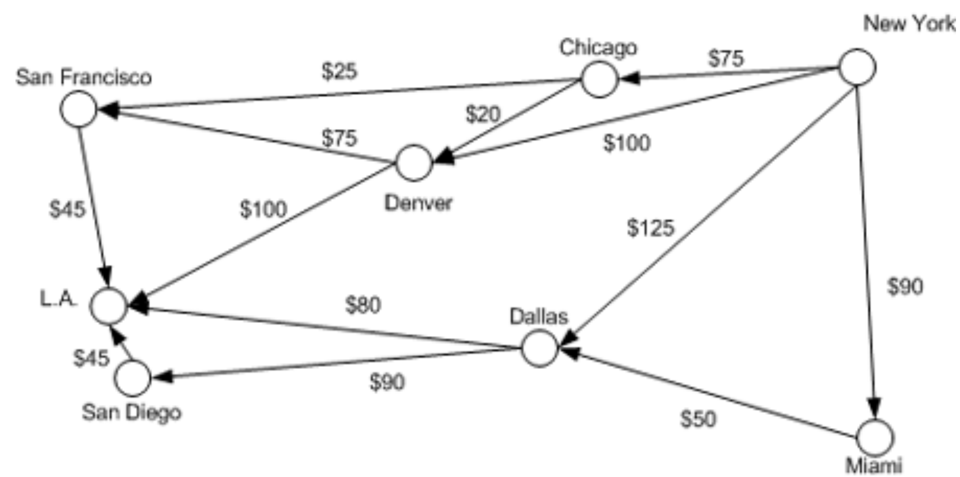
What is a graph?

- Edges can be weighted
  - Denotes "strength", distance, etc.

<u>Distance between cities</u>



<u>Flight costs</u>

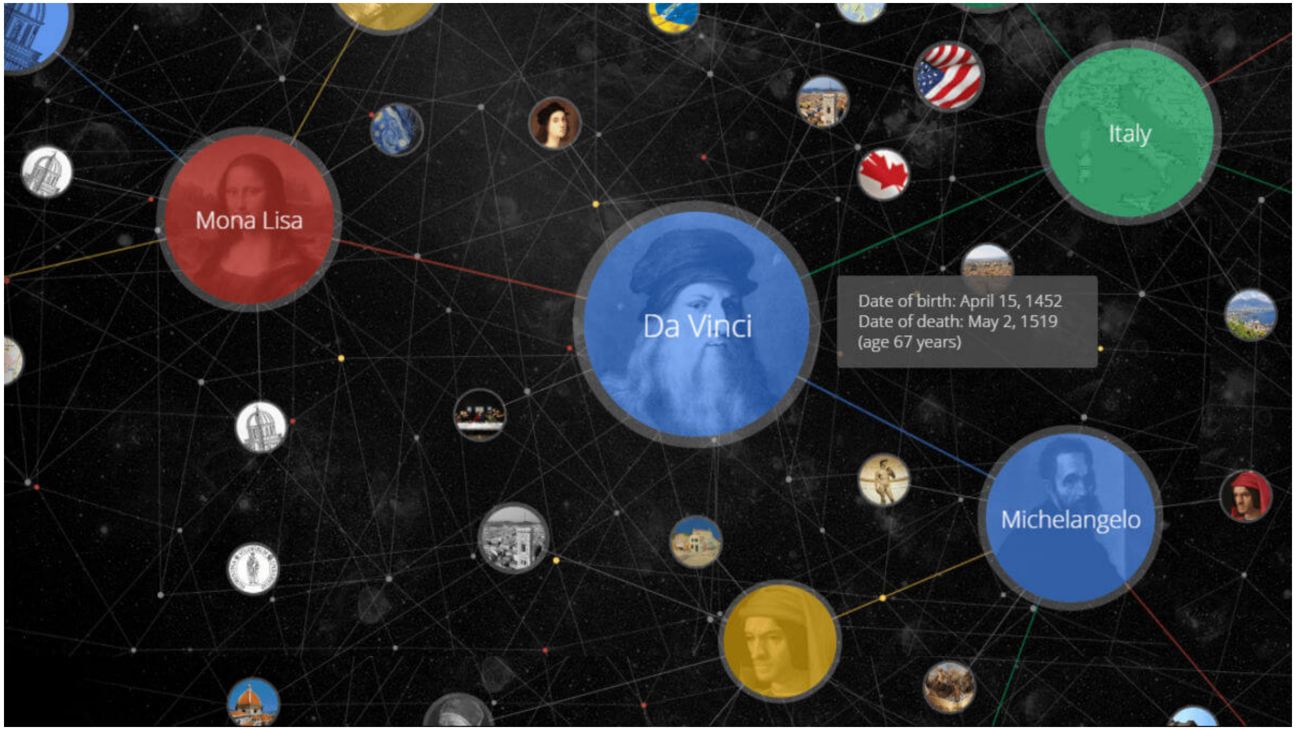


https://msdn.microsoft.com/en-us/library/aa289152(v=vs.71).aspx

What is a graph?

- Vertices and edges can have types and metadata

Google Knowledge Graph



http://searchengineland.com/laymans-visual-guide-googles-knowledge-graph-search-api-241935

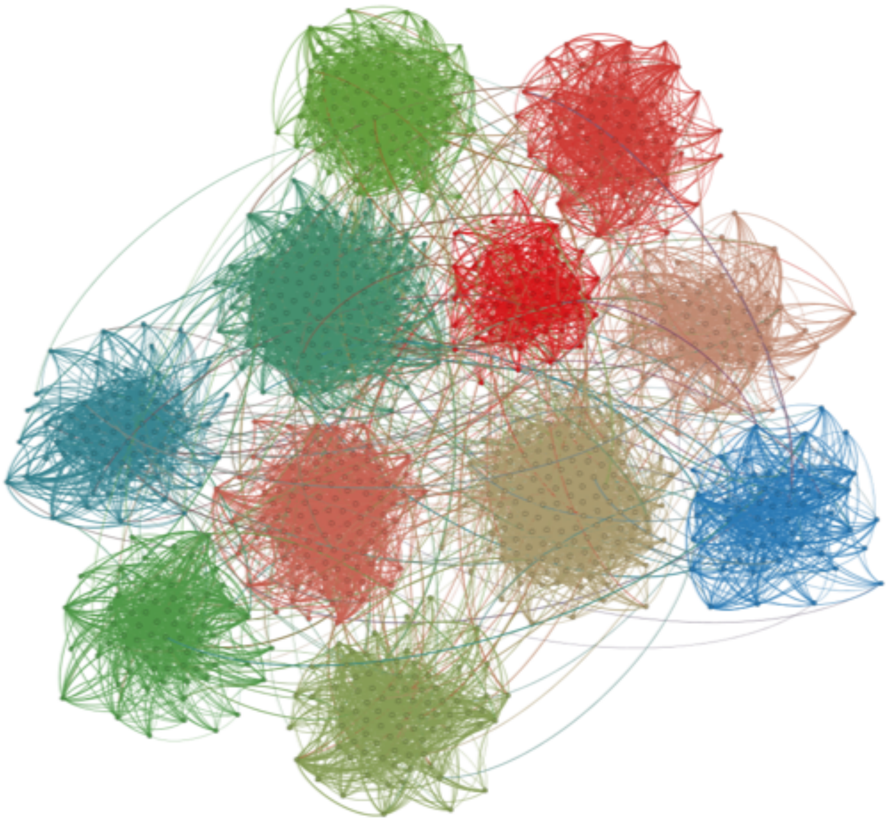
Social network queries

http://www.facebookfever.com/introducing-facebook-new-graph-api-explorer-features/

http://allthingsgraphed.com/2014/10/16/your-linkedin-network/

- Examples:
  - Finding all your friends who went to the same high school as you
  - Finding common friends with someone
  - Social networks recommending people whom you might know

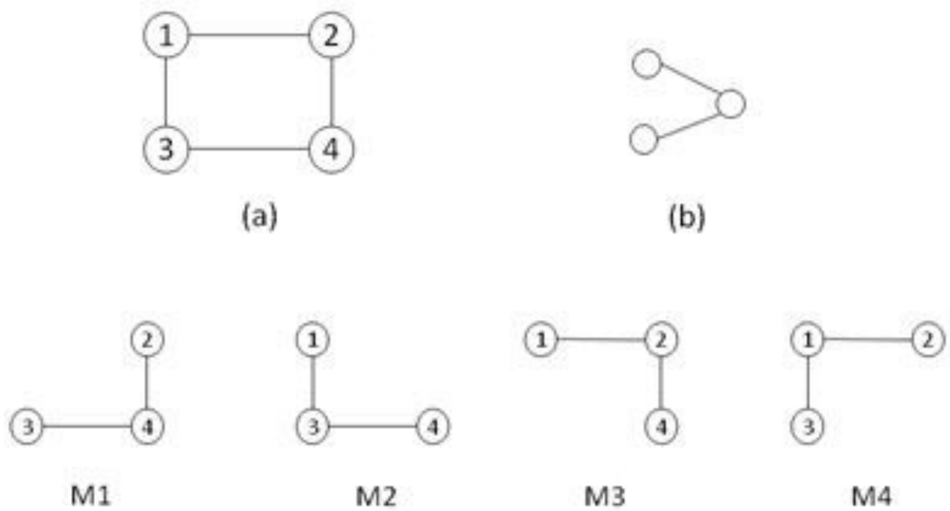
Finding good clusters



- Some applications
  - Finding people with similar interests
  - Detecting fraudulent websites
  - Document clustering
  - Unsupervised learning

- Finding groups of vertices that are “well-connected” internally and “poorly-connected” externally

Subgraph finding/motif discovery



Some applications

- Functions in biological networks
- Node importance in social networks

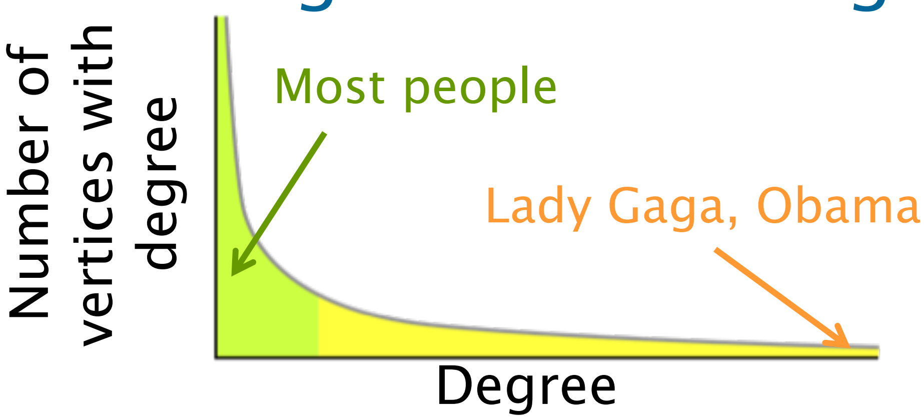
- Finding or counting specific subgraphs inside a graph
- Finding recurrent subgraphs

Properties of real-world graphs

- They can be big

Social network	Web graph	Web graph
41 million vertices	1.4 billion vertices	3.5 billion vertices
1.5 billion edges	6.6 billion edges	128 billion edges
(6.3 GB)	(38 GB)	(540 GB)

- Sparse (m = cn for a small constant c)
- Degrees can be highly skewed



Source: https://en.wikipedia.org/wiki/Power_law

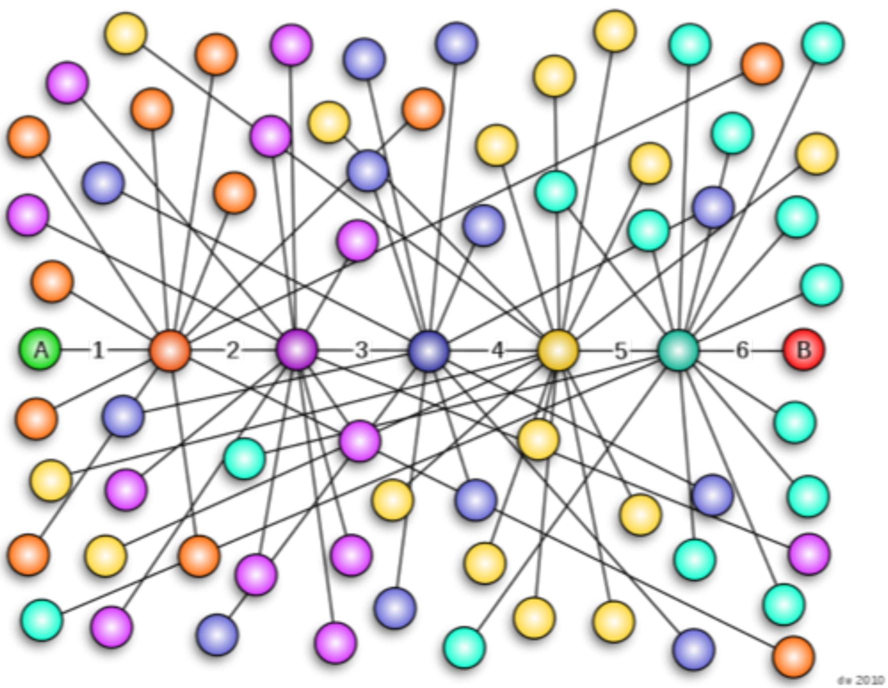
Studies have shown that many real-world graphs have a power law degree distribution

#vertices with deg. d ≈ a×d^{-p}

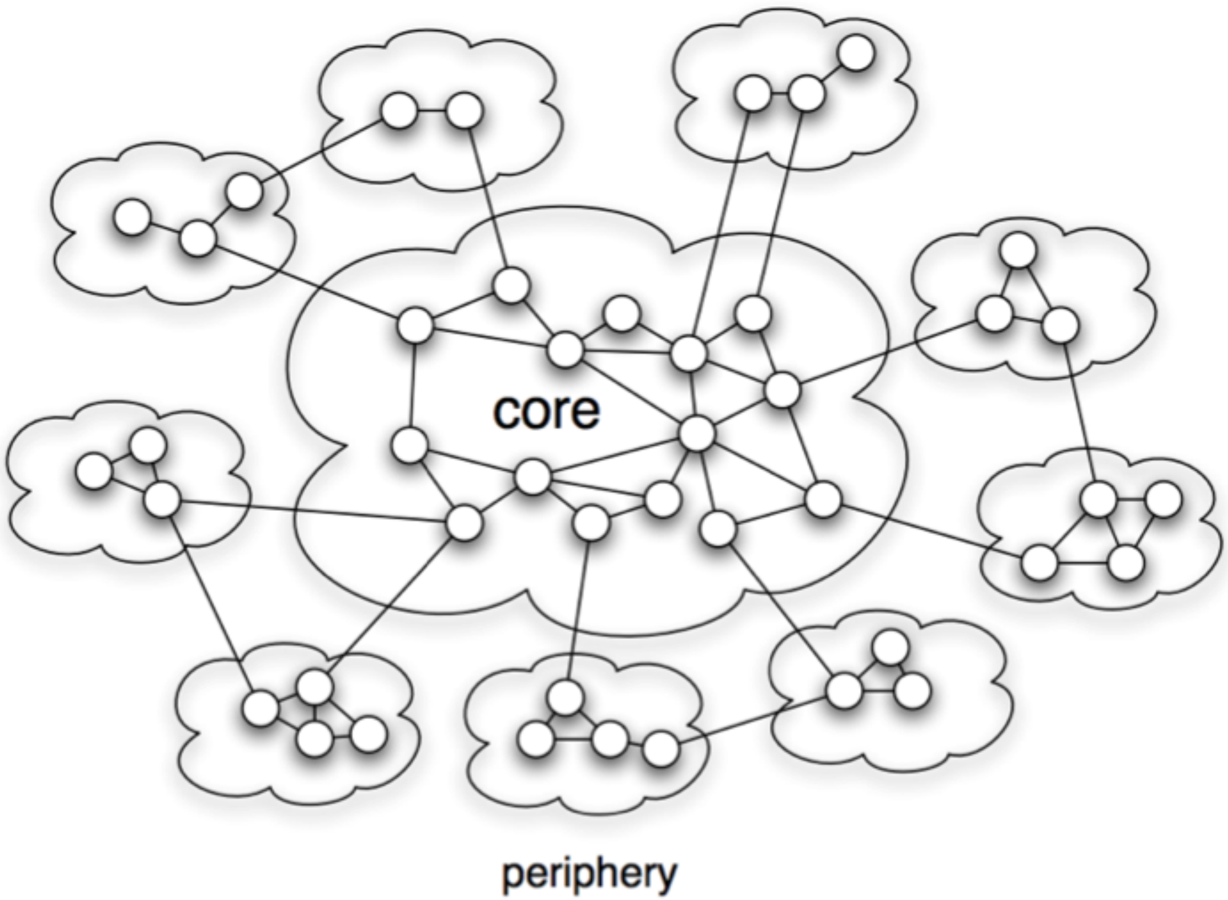
(2 < p < 3)

Small world phenomenon

- Also known as "six degrees of separation"
- Experiment by Stanley Milgram (1967)
  - Forward letter to a "target"
  - Could only mail letter to acquaintance you know on a first-name basis
  - 1/3 of letters eventually arrived, in a median of 6 steps



Core-periphery structure



- High-status nodes linked in a dense "core"
- Low-status nodes are on sparse "periphery"

COURSE INFORMATION

Course Information

- Graduate-level class
  - Undergraduates who have taken 6.046 and 6.172 are welcome
- Lectures: Wednesday and Friday 2:30-4pm
- Instructor: Julian Shun
- TA: Sherry (Mengjiao) Yang
- Units: 3-0-9
- We will use Piazza for communication

Course Website

https://people.csail.mit.edu/jshun/6886-s18/

Schedule (tentative)

Date	Topic	Required Reading	Optional Reading
Wednesday 2/7	Course Introduction and Graph Algorithms	Review CLRS Chapters 22-24 Chapter 1-2 of Networks, Crowds, and Markets	Algorithm Engineering - An Attempt at a Definition
Friday 2/9	Parallel Algorithms	Parallel Algorithms CLRS Chapter 27	Thinking in Parallel: Some Basic Data-Parallel Algorithms and Techniques Scheduling Multithreaded Computations by Work Stealing Thread Scheduling for Multiprogrammed Multiprocessors Provably Efficient Scheduling for Languages with Fine-Grained Parallelism Prefix Sums and Their Applications Book: Introduction to Parallel Algorithms by Joseph JaJa
Wednesday 2/14	Real-World and Synthetic Graphs	The Graph Structure in the Web - Analyzed on Different Aggregation Levels* Kronecker Graphs: An Approach to Modeling Networks* Chapters 13, 18, and 20 of Networks, Crowds, and Markets	Graph structure in the Web Power laws and the AS-level internet topology R-MAT: A Recursive Model for Graph Mining Statistical mechanics of complex networks Collective dynamics of 'small-world' networks The Small-World Phenomenon: An Algorithmic Perspective Four Degrees of Separation
Friday 2/16	Parallel Graph Traversal	Direction-Optimizing Breadth-First Search* A Faster Algorithm for Betweenness Centrality The More the Merrier: Efficient Multi-Source Graph Traversal*	A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers) Internally Deterministic Parallel Algorithms Can Be Fast SlimSell: A Vectorizable Graph Representation for Breadth-First Search Better Approximation of Betweenness Centrality ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation Fast approximation of betweenness centrality through sampling Scalable Betweenness Centrality Maximization via Sampling Articulation Points Guided Redundancy Elimination for Betweenness Centrality Betweenness Centrality: Algorithms and Implementations

Grading

Grading Breakdown	
Paper Questions and Reviews	20%
Paper Presentations	25%
Research Project	50%
Class Participation	5%

Paper Presentations

- This is a research-oriented course
- Cover content from 2–3 research papers each lecture
- 20-minute student presentation per paper
  - Discuss motivation for the problem solved
  - Key technical ideas
  - Theoretical/experimental results
  - Related work
  - Strengths/weaknesses
  - Directions for future work
  - Include several questions for discussion
- Sign up for presentation slots this week in Google doc
- Would be helpful to sign up even if listening

Paper Questions

- There will be a question per paper posted on Learning Modules
  - Submit answers on Learning Modules by 12pm on the day of the lecture

Paper Reviews

- Submit one paper review each week on a paper that will be covered that week
  - Cover motivation, key ideas, results, novelty, strengths/weaknesses, your ideas for improving the techniques or evaluation, any open problems or directions for further work
  - Submit on Learning Modules by Tuesday 11:59pm each week (before we cover the papers)
  - Reviews will be made viewable to class (anonymously)
  - Read them before the lecture to help prepare for the discussions

Research Project

- Open-ended research project related to graphs to be done in groups of 2-3
- Some ideas
  - Implementation of non-trivial algorithm
  - Analyzing/optimizing performance of existing algorithm
  - Designing new theoretically and/or practically efficient algorithms
  - Applying graph algorithms in larger applications
  - Coming up with new graph problems
  - Improving or designing new graph frameworks
  - Survey of an area
  - Any topic may involve parallelism, cache-efficiency, I/O-efficiency, and memory-efficiency
- Can be related to any research you are doing
- Can possibly be a starting point for a publication

Project Timeline

Assignment	Due Date
Pre-proposal meeting	3/14
Proposal	3/16
Mid-term report	4/13
Poster Session	5/14 or 5/16
Final Report	5/17

- Pre-proposal meeting
  - 15-minute meeting to run idea by instructors
- Talk to instructors if you need computing resources for the project
  - We may have some AWS credits

GRAPH REPRESENTATIONS

Graph Representations

- Vertices labeled from 0 to n−1

	0	1	2	3	4
0	0	1	0	0	0
1	1	0	0	1	1
2	0	0	0	1	0
3	0	1	1	0	0
4	0	1	0	0	0

Adjacency matrix
("1" if edge exists,
"0" otherwise)

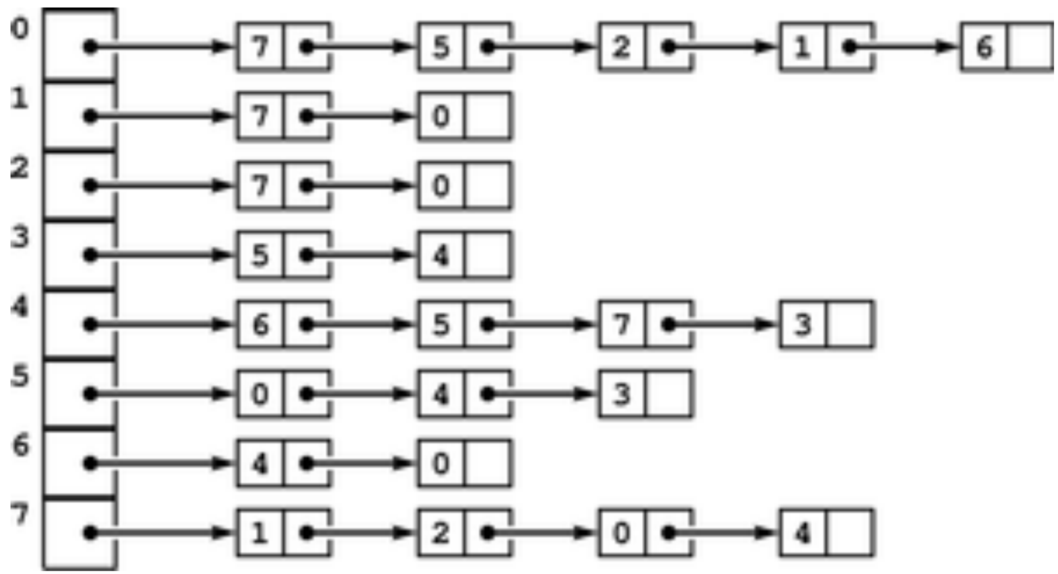
(0,1)
(1,0)
(1,3)
(1,4)
(2,3)
(3,1)
(3,2)
(4,1)

Edge list

- $O(n^2)$ space for adjacency matrix
- $O(m)$ space for edge list

Graph Representations

- Adjacency list
  - Array of pointers (one per vertex)
  - Each vertex has an unordered list of its edges



- Space requirement is O(n+m)
- Can substitute linked lists with arrays for better cache performance
  - Tradeoff: more expensive to update graph

Graph Representations

- Compressed sparse row (CSR)
  - Two arrays: Offsets and Edges
  - Offsets[i] stores the offset of where vertex i's edges start in Edges

Vertex IDs	0	1	2	3	
Offsets	0	4	5	11	…

Edges	2	7	9	16	0	1	6	9	12	…

- How do we know the degree of a vertex?
- Space usage is O(n+m)
- Can also store values on the edges with an additional array or interleaved with Edges

Tradeoffs in Graph Representations

- What is the cost of different operations?

	Adjacency matrix	Edge list	Adjacency list (linked list)	Compressed sparse row
Storage cost / scanning whole graph	$O(n^2)$	$O(m)$	$O(m+n)$	$O(m+n)$
Add edge	$O(1)$	$O(1)$	$O(1)$	$O(m+n)$
Delete edge from vertex v	$O(1)$	$O(m)$	$O(deg(v))$	$O(m+n)$
Finding all neighbors of a vertex v	$O(n)$	$O(m)$	$O(deg(v))$	$O(deg(v))$
Finding if w is a neighbor of v	$O(1)$	$O(m)$	$O(deg(v))$	$O(deg(v))$

- There are variants/combinations of these representations

BREADTH-FIRST SEARCH

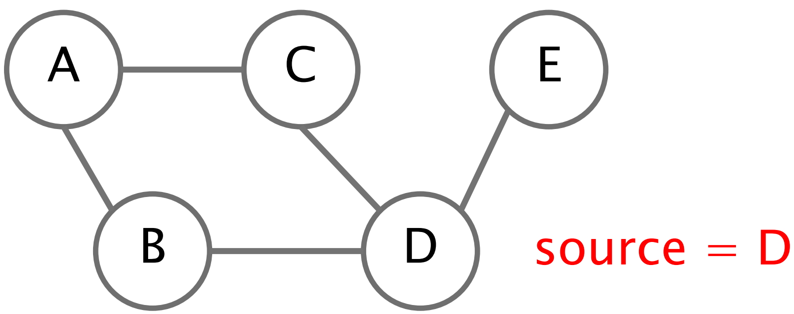
Breadth-First Search (BFS)

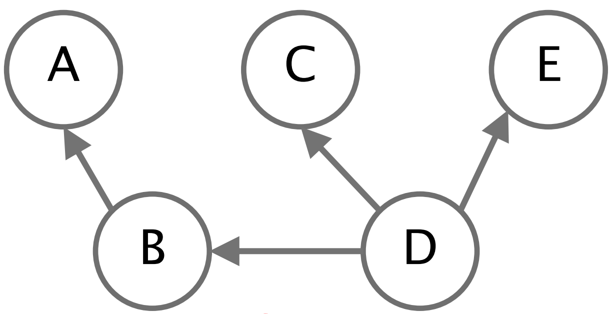
- Given a source vertex *s*, visit the vertices in order of distance from *s*
- Possible outputs:
  - Vertices in the order they were visited
    - D, B, C, E, A
  - The distance from each vertex to *s*

A	B	C	D	E
2	1	1	0	1

  - A BFS tree, where each vertex has a parent to a neighbor in the previous level

Applications
Betweenness centrality
Eccentricity estimation
Maximum flow
Web crawlers
Network broadcasting
Cycle detection
…





BFS tree

Sequential BFS Algorithm

```
Breadth-First-Search(Graph, root):

    for each node n in Graph:
        n.distance = INFINITY
        n.parent = NIL
```

Source: https://en.wikipedia.org/wiki/Breadth-first_search

- BFS requires O(n+m) work on n vertices and m edges

Sequential BFS Algorithm

- Assume graph is given in compressed sparse row format
  - Two arrays: Offsets and Edges
  - n vertices and m edges (assume Offsets[n] = m)

```
int* parent =
 (int*) malloc(sizeof(int)*n);
int* queue =
 (int*) malloc(sizeof(int)*n);

for(int i=0; i<n; i++) {
   parent[i] = -1;
}

queue[0] = source;
parent[source] = source;

int q_front = 0, q_back = 1;
```

```
//while queue not empty
while(q_front != q_back) {
   int current = queue[q_front++]; //dequeue
   int degree =
        Offsets[current+1]-Offsets[current];
   for(int i=0;i<degree; i++) {
        int ngh = Edges[Offsets[current]+i];
        //check if neighbor has been visited
        if(parent[ngh] == -1) {
            parent[ngh] = current;
            //enqueue neighbor
            queue[q_back++] = ngh;
        }
   }
}
```

Total of m random accesses

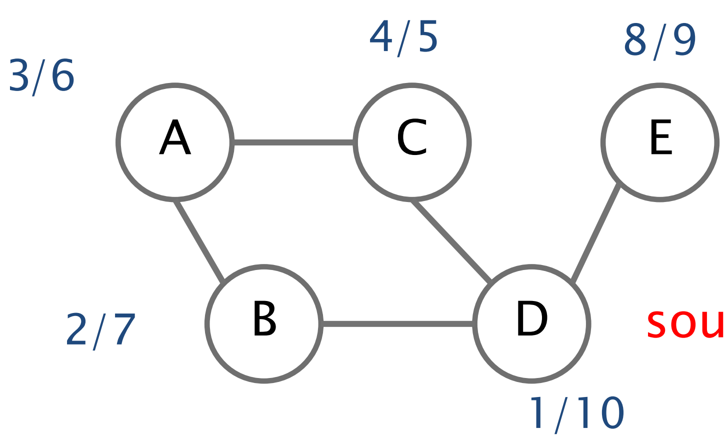
- What is the most expensive part of the code?
  - Random accesses cost more than sequential accesses

DEPTH-FIRST SEARCH

Depth-First Search (DFS)

- Explores edges out of the most recently discovered vertex
- Possible outputs:
  - Depth-first forest
  - Vertices in the order they were first visited (preordering)
  - Vertices in the order they were last visited (postordering)
  - Reverse postordering

Applications
Topological sort
Solving mazes
Biconnected components
Strongly connected components
Cycle detection
…



source = D

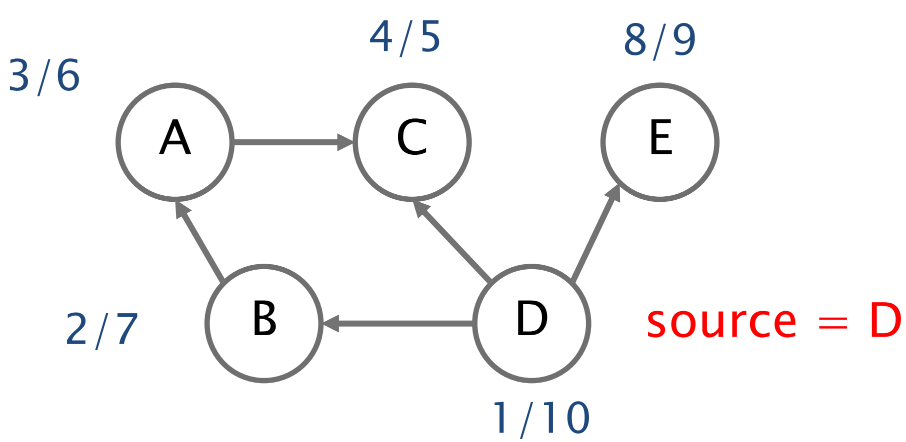
Preorder: D, B, A, C, E
Postorder: C, A, B, E, D
Reverse postorder: D, E, B, A, C

DFS requires O(n+m) work on n vertices and m edges

TOPOLOGICAL SORT

Topological Sort

- Given a directed acyclic graph, output the vertices in an order such that all predecessors of a vertex appear before it
  - Application: scheduling tasks with dependencies (e.g. parallel computing, Makefile)
- Solution: output vertices in reverse postorder in DFS



Reverse postorder: D, E, B, A, C

SHORTEST PATHS

Single-Source Shortest Paths

- Given a weighted graph and a source vertex, output the distance from the source vertex to every vertex

- Non-negative weights
  - Dijkstra's algorithm
  - O(m + n log n) work using Fibonnaci heap

- General weights
  - Bellman-Ford algorithm
  - O(mn) work

Dijkstra's Algorithm

```
function Dijkstra(Graph, source):
    dist[source] ← 0                                    // Initialization

    create vertex set Q

```

- O((m+n)log n) work using normal heap
- O(m + nlog n) work using Fibonacci heap
  - Extract-min takes O(log n) work but decreasing priority only takes O(1) work (amortized)

Bellman-Ford Algorithm

```
Bellman-Ford(G, source):
    ShortestPaths = {∞, ∞, …, ∞}    //size n; stores shortest path distances
    ShortestPaths[source] = 0
    for i=1 to n-1:
        for each vertex v in G:
            for each w in neighbors(v):
                if(ShortestPaths[v] + weight(v,w) < ShortestPaths[w]):
                    ShortestPaths[w] = ShortestPaths[v] + weight(v,w)
        if no shortest paths changed:
            return ShortestPaths
    report "negative cycle"
```

- At most n rounds, each doing O(n+m) work
- Total work = O(mn)

Parallelism

Parallelism

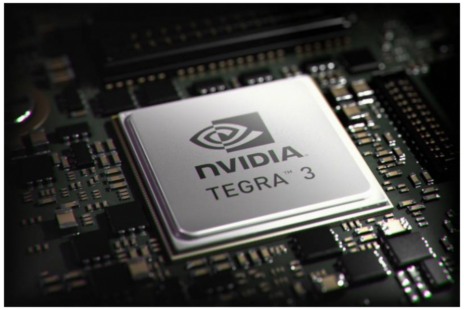
Graphs are becoming very large!

41 million vertices
1.5 billion edges
(6.3 GB)

1.4 billion vertices
6.6 billion edges
(38 GB)

3.5 billion vertices
128 billion edges
(540 GB)

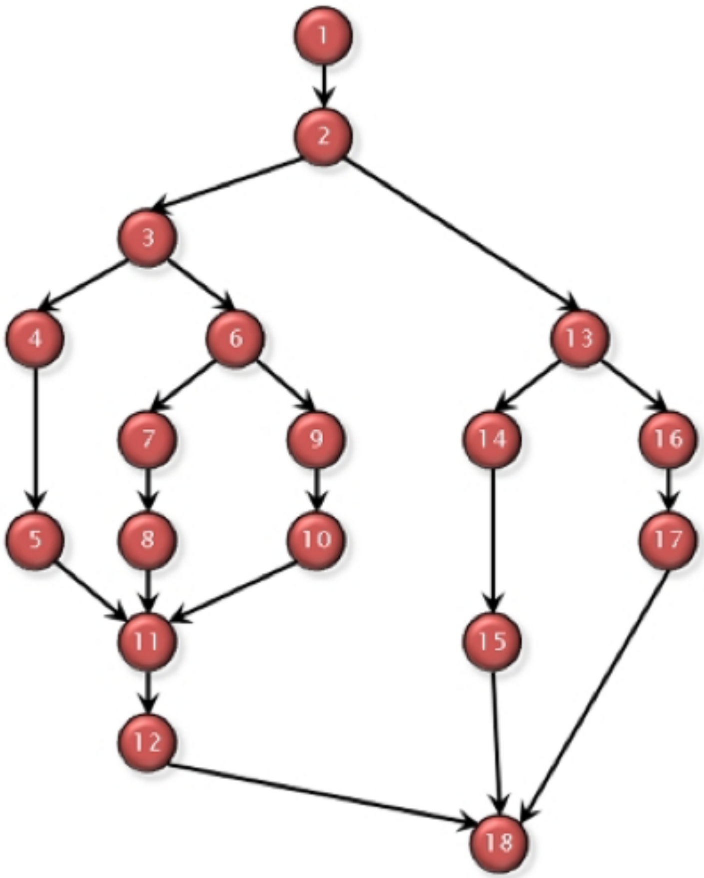
Parallel machines are everywhere!



Can rent machines on AWS with 72 cores (144 hyper-threads) and 4TB of RAM

Parallelism Models

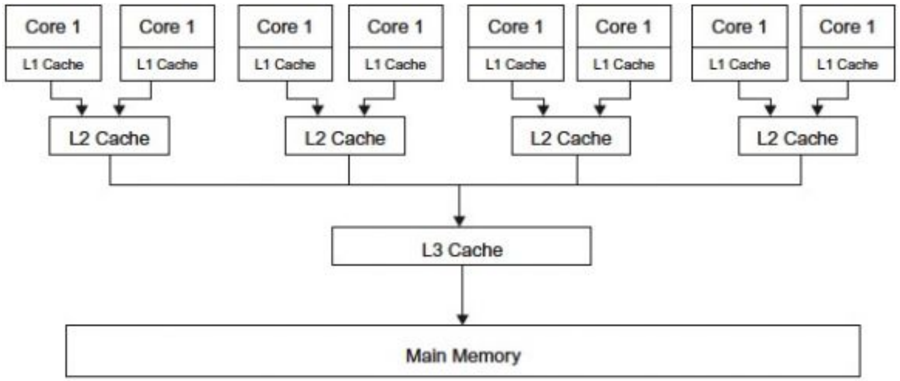
Computation graph



- Work = number of vertices in graph (number of operations)
- Depth = longest directed path in graph (dependence length)
- Parallelism = Work / Depth

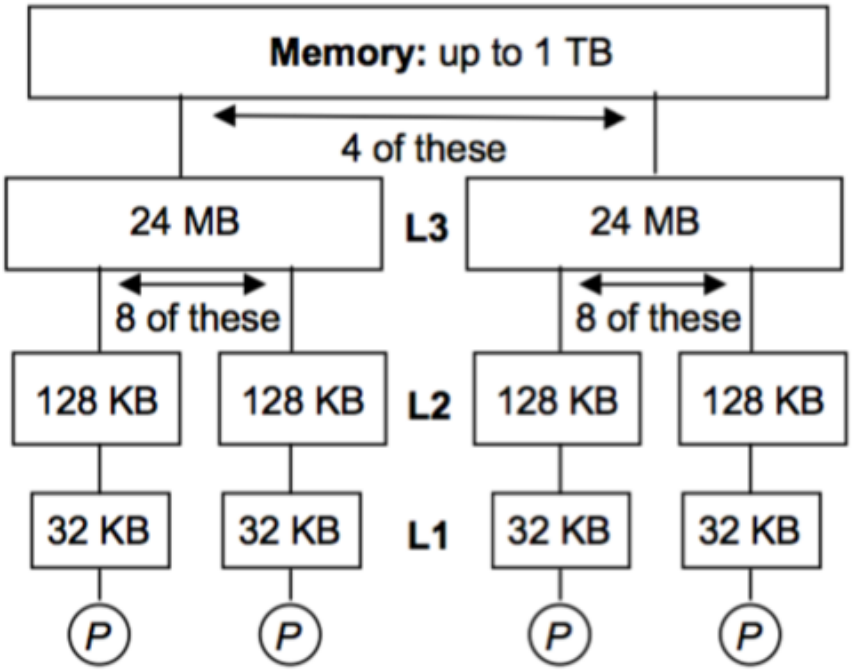
Goal 1: work-efficient and low (polylogarithmic) depth algorithms

Goal 2: simple, practical, and cache-friendly



CACHING AND NON-UNIFORM MEMORY ACCESS

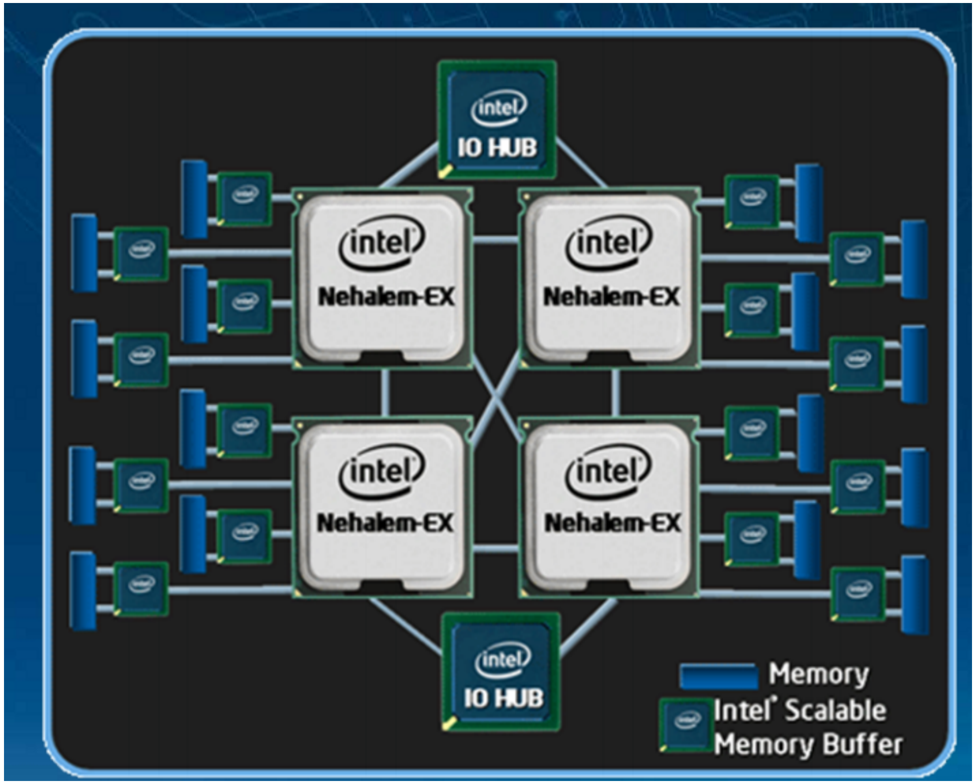
Cache Hierarchies



Design cache-efficient and cache-oblivious graph algorithms to improve locality

Memory level	Approx latency
L1 Cache	1-2ns
L2 Cache	3-5ns
L3 cache	12-40ns
DRAM	60-100ns

Non-uniform Memory Access (NUMA)



Design NUMA-aware graph algorithms to improve locality

- Accessing remote memory is more expensive than accessing local memory of a socket
  - Latency depends on the number of hops

I/O Efficiency

I/O Efficiency

- Need to read input from disk at least once
- Need to read many more times if graph doesn't fit in memory

Memory	Latency	Throughput
DRAM	60-100 ns	Tens of GB/s
SSD	Tens of µs	500 MB-2 GB/s (seq), 50-200 MB/s (rand)
HDD	Tens of ms	200 MB/s (seq), 1 MB/s (rand)

Source: https://www.pcgamer.com/hard-drive-vs-ssd-performance/2/

I/O Efficiency

- For graphs larger than main memory, disk-based computing can be competitive with distributed clusters
- GraphChi: Large-Scale Graph Computation on Just a PC (OSDI 2012)

Application & Graph	Iter.	Comparative result	GraphChi (Mac Mini)	Ref
Pagerank & domain	3	GraphLab[30] on AMD server (8 CPUs) **87 s**	**132 s**	-
Pagerank & twitter-2010	5	Spark [45] with 50 nodes (100 CPUs): **486.6 s**	**790 s**	[38]
Pagerank & V=105M, E=3.7B	100	Stanford GPS, 30 EC2 nodes (60 virt. cores), **144 min**	approx. **581 min**	[37]
Pagerank & V=1.0B, E=18.5B	1	Piccolo, 100 EC2 instances (200 cores) **70 s**	approx. **26 min**	[36]
Webgraph-BP & yahoo-web	1	Pegasus (Hadoop) on 100 machines: **22 min**	**27 min**	[22]
ALS & netflix-mm, D=20	10	GraphLab on AMD server: **4.7 min**	**9.8 min** (in-mem) **40 min** (edge-repl.)	[30]
Triangle-count & twitter-2010	-	Hadoop, 1636 nodes: **423 min**	**60 min**	[39]
Pagerank & twitter-2010	1	PowerGraph, 64 x 8 cores: **3.6 s**	**158 s**	[20]
Triange-count & twitter- 2010	-	PowerGraph, 64 x 8 cores: **1.5 min**	**60 min**	[20]

- Lots of follow-up work on disk-based computing that we will study
- External-memory algorithms to minimize I/O's

ALGORITHMS

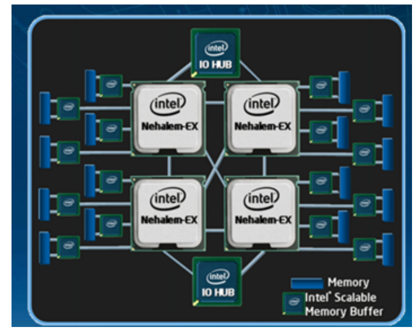
Graph Algorithms

- We will study algorithms for particular problems
  - Parallelism, cache-efficiency, I/O-efficiency, streaming

Breadth-first search	Betweenness centrality	SSSP
PageRank	Triangle Computations	Graphlet counting
Frequent subgraph finding	Dense subgraph discovery	Graph coloring
Connected components	Clustering	Partitioning
K-core decomposition	Truss decomposition	Nuclei decomposition
Minimum spanning forest	Spanning forest	Eccentricity estimation
Maximal matching	Set cover	Collaborative filtering
Strongly connected components	Biconnected components	Maximum flow
Local clustering	Belief propagation	Maximal independent set

Efficient Graph Processing

- Use parallelism



- Design efficient algorithms

Breadth-first search	Single-source shortest paths
Betweenness centrality	Eccentricity estimation
Connected components	PageRank
…	…

- Write/optimize code for each application
- Build a general framework

GRAPH PROCESSING FRAMEWORKS

Graph Processing Frameworks

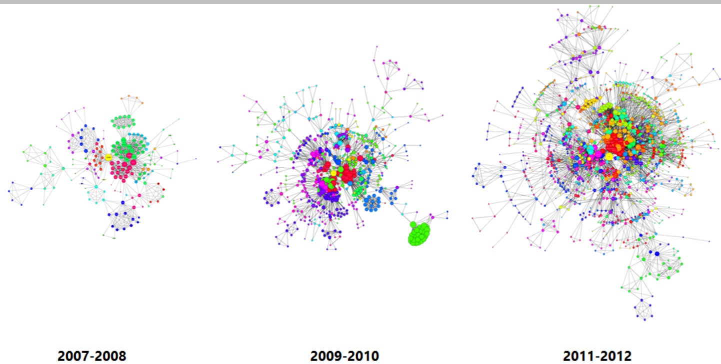
- Reduce programming effort of writing efficient parallel graph programs

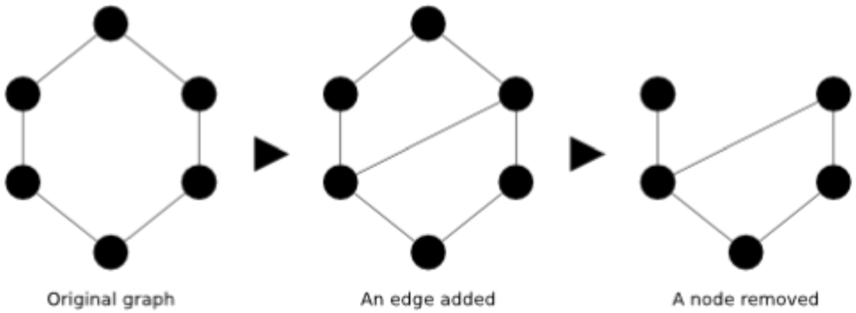
Graph processing frameworks/libraries

Pregel, Giraph, GPS, GraphLab, PowerGraph, PRISM, Pegasus, Knowledge Discovery Toolbox, CombBLAS, GraphChi, GraphX, Galois, X-Stream, Gunrock, GraphMat, Ringo, TurboGraph, FlashGraph, Grace, PathGraph, Polymer, GoFFish, Blogel, LightGraph, MapGraph, PowerLyra, PowerSwitch, Imitator, XDGP, Signal/Collect, PrefEdge, EmptyHeaded, Gemini, Wukong, Parallel BGL, KLA, Grappa, Chronos, Green-Marl, GraphHP, P++, LLAMA, Venus, Cyclops, Medusa, NScale, Neo4J, Trinity, GBase, HyperGraphDB, Horton, GSPARQL, Titan, ZipG, Cagra, Milk, Ligra, Ligra+, Julienne, GraphPad, Mosaic, BigSparse, Graphene, Mizan, Green-Marl, PGX, PGX.D, Wukong+S, Stinger, GraphIn, Tornado, Bagel, KickStarter, Naiad, Kineograph, GraphMap, Presto, Cube, Giraph++, Photon, TuX2, GRAPE, GraM, Congra, MTGL, GridGraph, NXgraph, Chaos, Mmap, Clip, Floe, GraphGrind, DualSim, ScaleMine, Arabesque, GraMi, SAHAD, Facebook TAO, Weaver, G-SQL, G-SPARQL, gStore, Horton+, S2RDF, Quegel, EAGRE, Shape, RDF-3X, CuSha, Garaph, Totem, GTS, Frog, GBTL-CUDA, Graphulo, Zorro, Coral, GraphTau, Wonderland, GraphP, and many others…

Dynamic Graphs

Dynamic Graphs

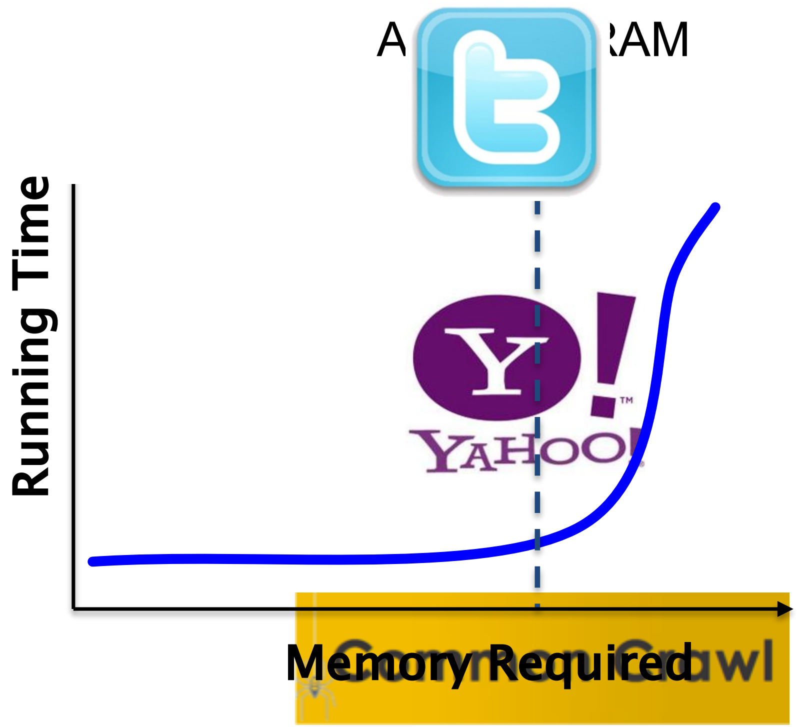


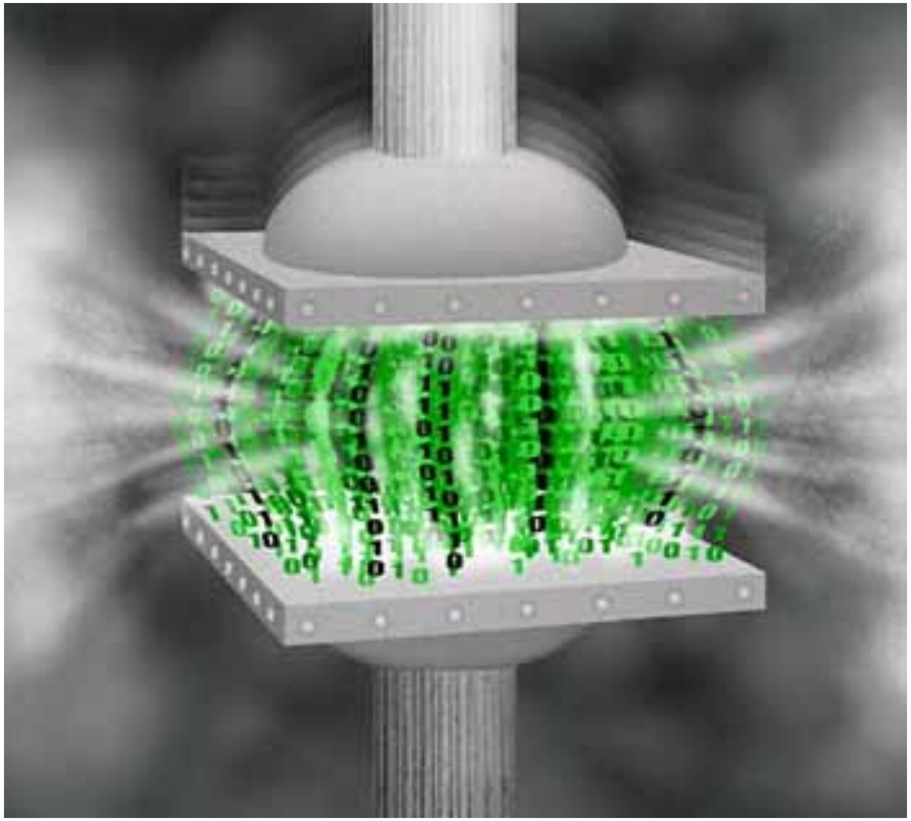


- Many graphs are changing over time
  - Adding/deleting connections on social networks
  - Traffic conditions changing
  - Communication networks (email, IMs)
  - World Wide Web
  - Content sharing (Youtube, Flickr, Pinterest)
- Need graph data structures that allow for efficient updates (in parallel)
- Need (parallel) algorithms that respond to changes without re-computing from scratch

COMPRESSION AND REORDERING

Large Graphs



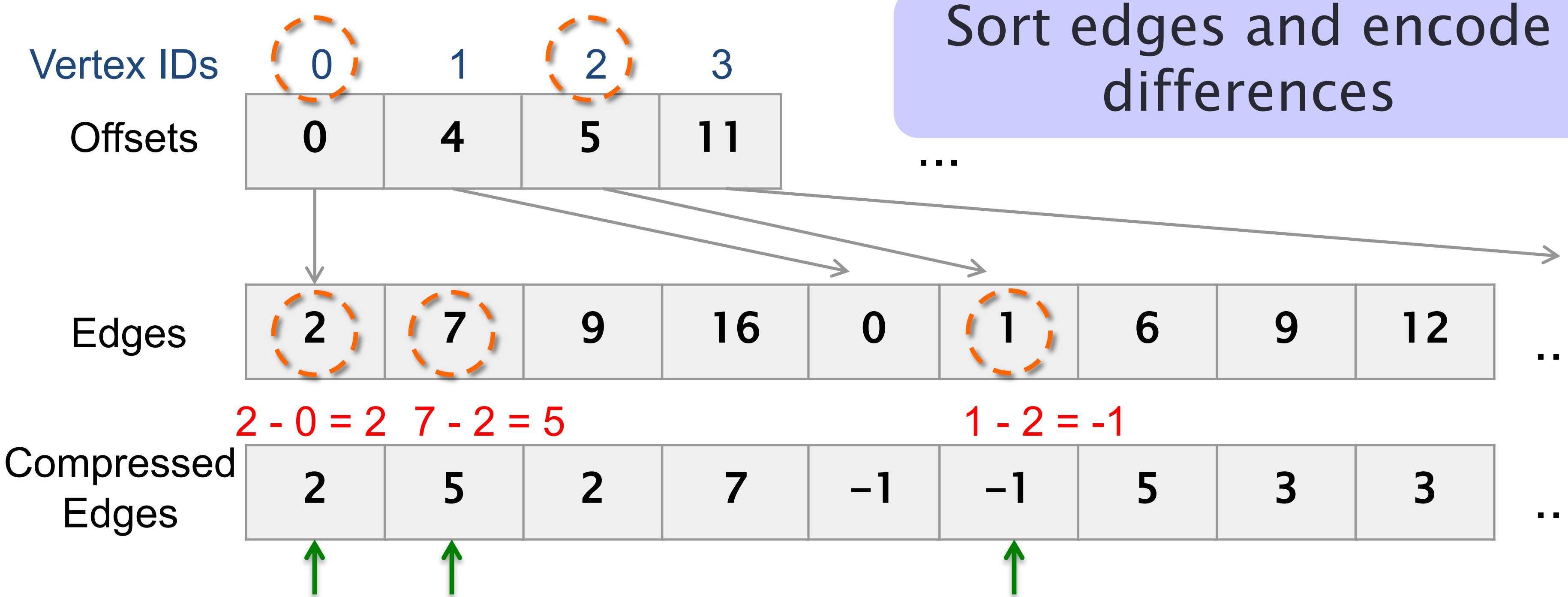


- What if you cannot fit a graph on your machine?
- Cost of machines increases with memory size

Graph Compression

Graph Compression on CSR

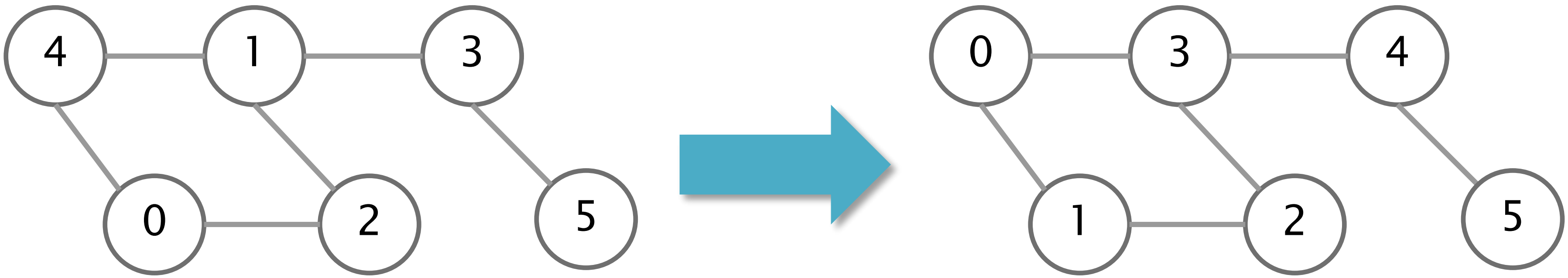
Sort edges and encode differences



- For each vertex v:
  - First edge: difference is Edges[Offsets[v]]−v
  - i'th edge (i>1): difference is Edges[Offsets[v]+i]−Edges[Offsets[v]+i−1]
- Want to use fewer than 32 or 64 bits per value
- Compression can improve parallel running time

Graph Reordering

- Reassign IDs to vertices to improve locality
  - Goal: Make vertex IDs close to their neighbors' IDs and neighbors' IDs close to each other



Sum of differences = 21

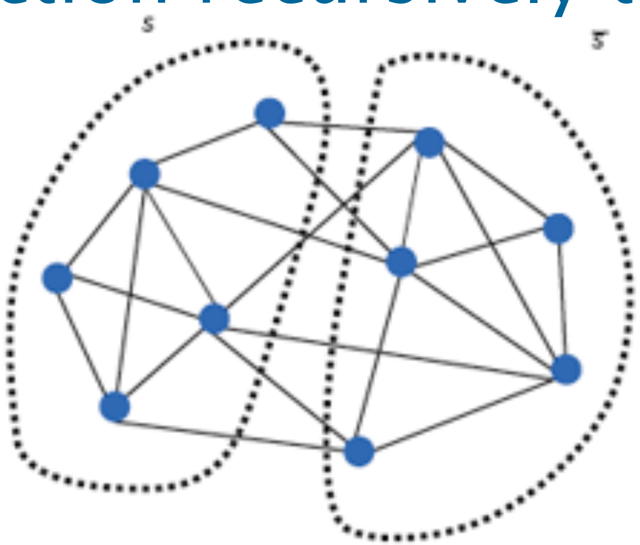
Sum of differences = 19

- Can improve compression rate due to smaller "differences"
- Can improve performance due to higher cache hit rate
- Various methods: BFS, DFS, METIS, degree, etc.

PARTITIONING/CLUSTERING

Graph Partitioning/Clustering

- Partition graph so that parts have similar size and there are few crossing edges
- Conductance = (# crossing edges)/(size of smaller partition)
- Minimizing conductance is NP-hard
- Many approximation methods
- Apply bisection recursively to get more partitions



Applications
Parallel computing
Community detection
VLSI circuit design
Image segmentation
…

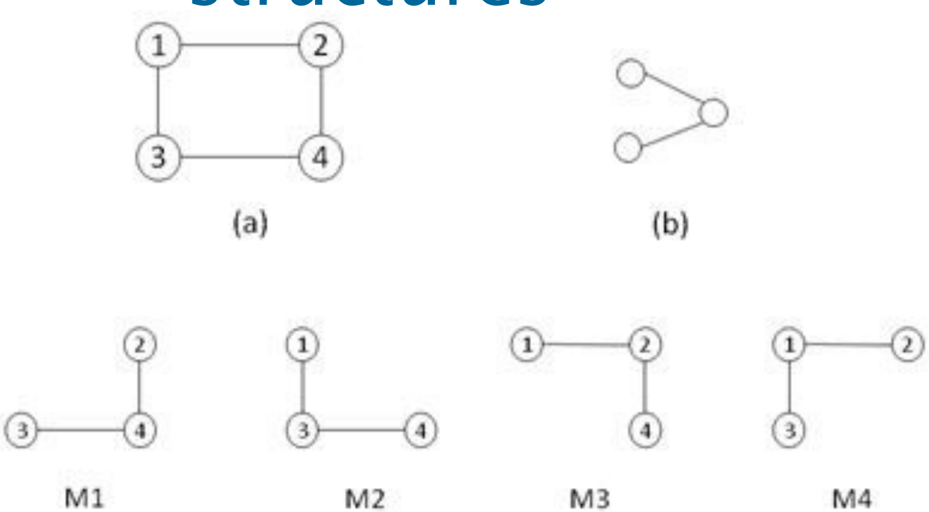
Source: https://cacm.acm.org/magazines/2008/10/515-geometry-flows-and-graph-partitioning-algorithms/fulltext

Graph Partitioning/Clustering

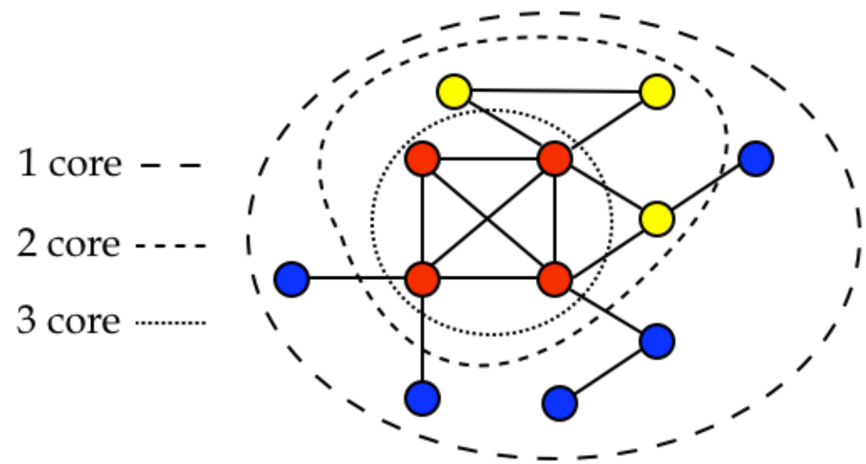
- Will study different algorithms
  - Global vs. local algorithms
- Variants on optimization metric
- Apply algorithms to find communities in real networks

Finding Graph Structure

- Triangles, 4-cliques, cycles, wedges, etc.
  - # incident subgraphs is a measure of importance
- Frequent subgraph mining
  - Extract all subgraphs whose counts are above threshold
- Decomposing graphs into cores and other structures



Example subgraphs



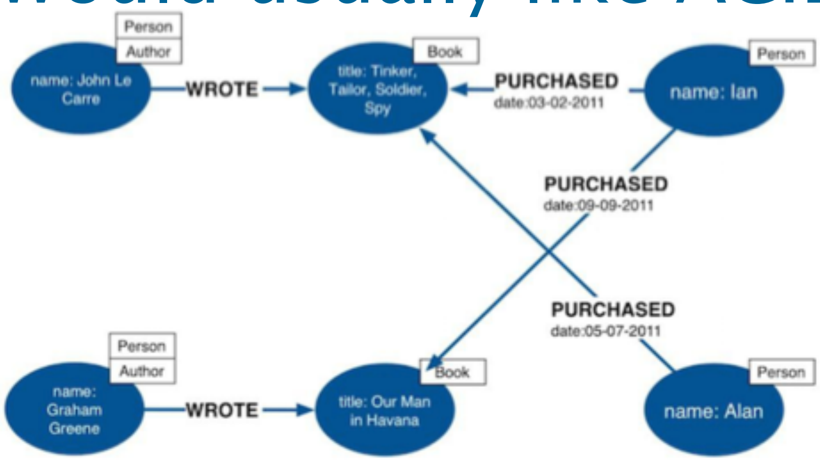
Core decomposition

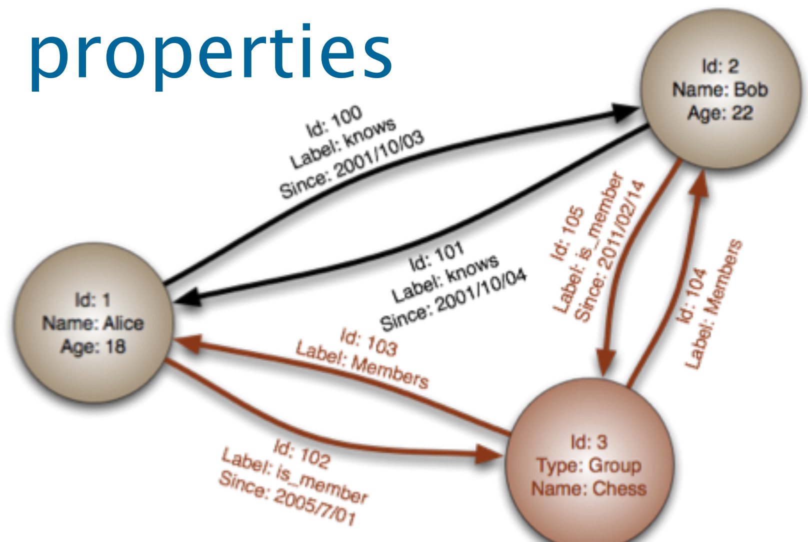
Source: https://chaoslikehome.wordpress.com/tag/topology/

GRAPH STORES

Graph Stores

- A database that allows for efficient semantic queries on graphs
- Useful for queries on graphs with lots of metadata
  - Example: On Facebook, find all people who are currently students, study at MIT, and have at least 100 friends who study elsewhere
- Allows efficient updates
- Would usually like ACID properties





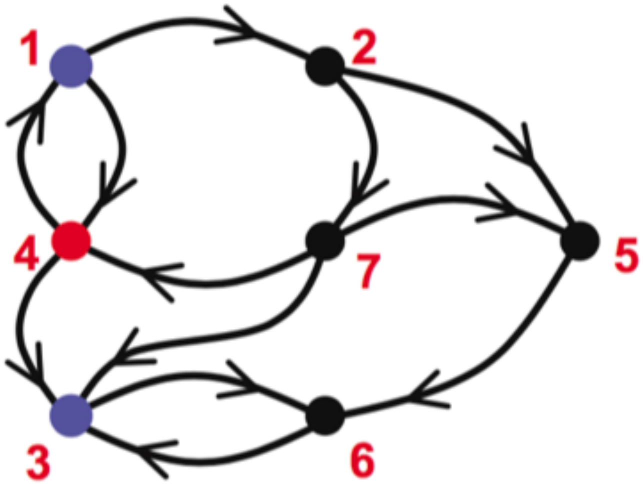
GPUs

GPUs

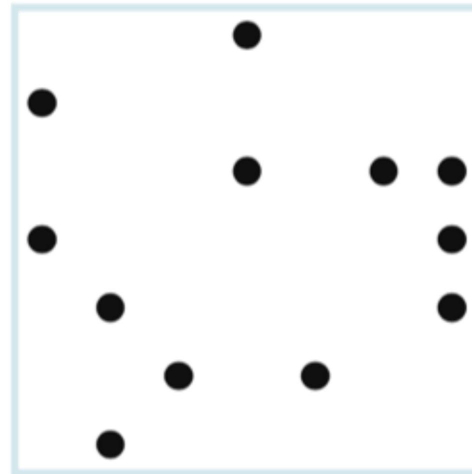
- Pros: More cores, more memory bandwidth
- Cons: Less memory, harder to program, each core is slower, data transfer time
- GPU (and GPU+CPU) graph processing an active area of research

LINEAR ALGEBRA AND GRAPH

Matrix-Graph Duality



$G = (V, E)$

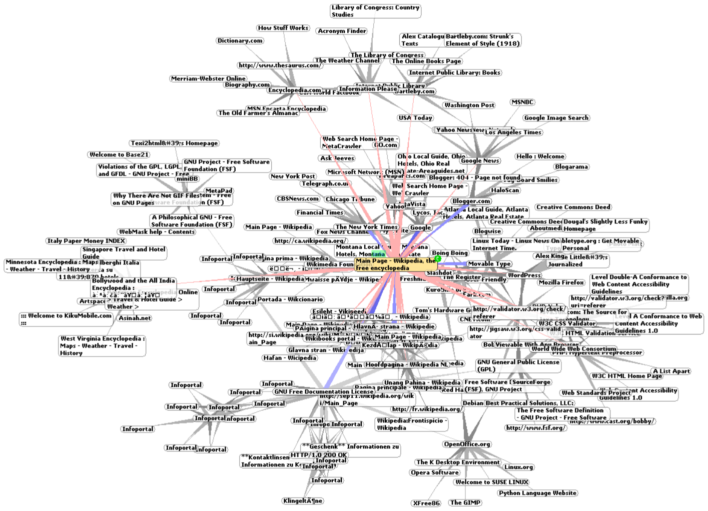


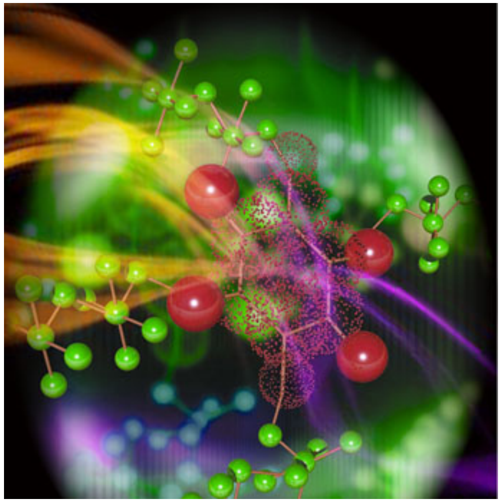
A^T

Source: Graph Algorithms in the Language of Linear Algebra (SIAM)

- Graph algorithms as matrix-vector multiply
  - Traditionally use (+,*) semiring
  - (or, and) for breadth-first search
  - (+, min) for single-source shortest paths
- One step of a breadth-first search
- CSR, reordering, compression, partitioning

Summary





- Lots of exciting research going on in graph analytics!
- Take this course to learn about latest results and try out research in graph analytics