CS260 – Lecture 10
Yan Gu

Algorithm Engineering
(aka. How to Write Fast Code)

Algorithm Engineering and Graph Processing systems

CS260: Algorithm Engineering Lecture 10

- What is algorithm engineering
- Graphs
- Graph processing systems

Overall Structure in this Course

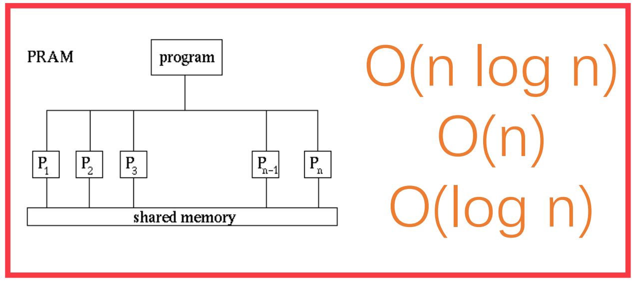
Performance Engineering

Parallelism
I/O efficiency
New Bentley rules
Brief overview of architecture

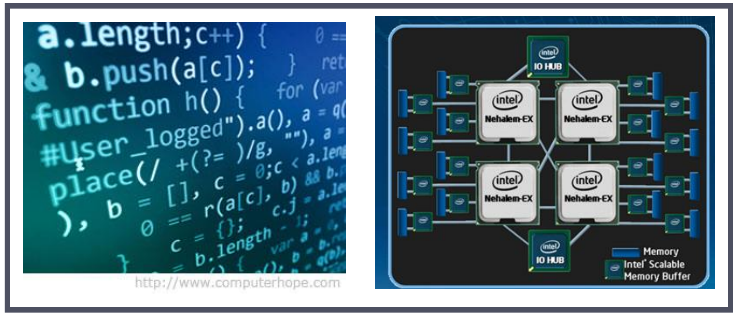
Algorithm Engineering

Sorting / Semisorting
Matrix multiplication
Graph algorithms
Geometric algorithms

What is Algorithm Engineering?



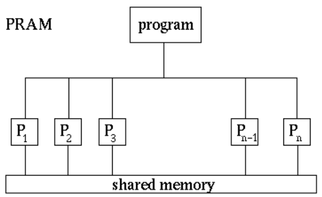
Theory



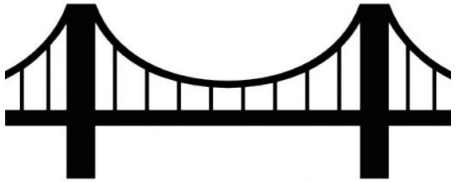
Practice

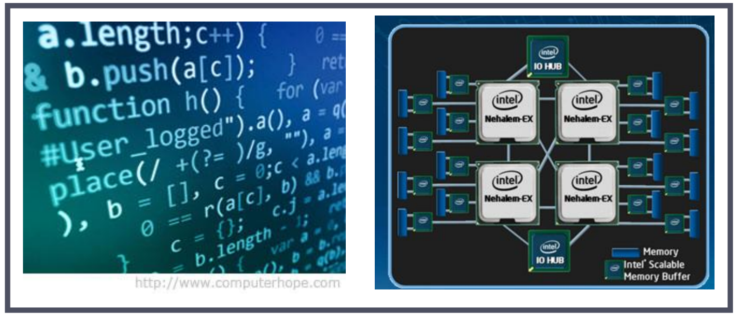
- **For many decades, theory and practice are two separate areas**
- **Theory studies computability (e.g., complexity classes)**
- **Writing faster codes was done the system community**
  - Almost every undergrads know the algorithms with best bounds for classic problems such as SCC, sorting, connectivity, convex hull
  - Research is mostly about specific input instances, detail tuning, on HPCs

What is Algorithm Engineering?



O(n log n)
O(n)
O(log n)

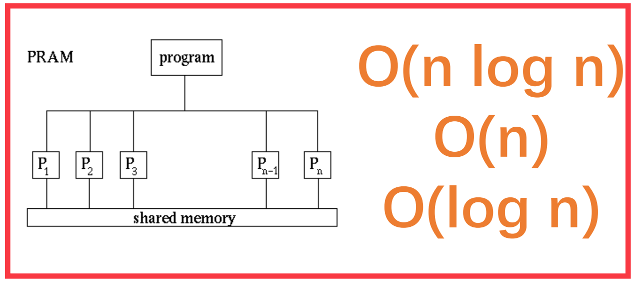


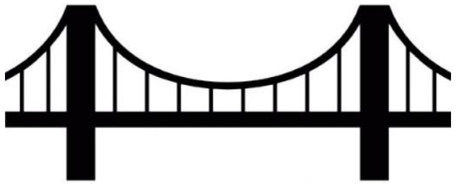


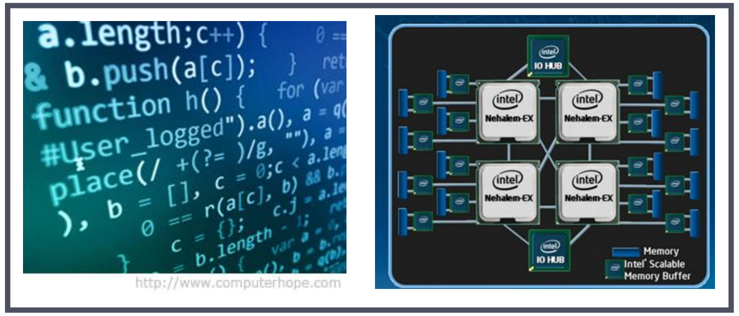
Theory ***Practice***

- **No longer the case in the past decades since computer architecture becomes significantly more sophisticated**
- **Parallelism, I/O efficiency, new hardware such as non-volatile memories**

Bridging Theory and Practice





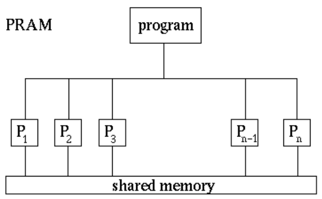


- Good empirical performance
- Confidence that algorithms will perform well in many different settings
- Ability to predict performance (e.g. in real-time applications)
- Important to develop theoretical models to capture properties of technologies

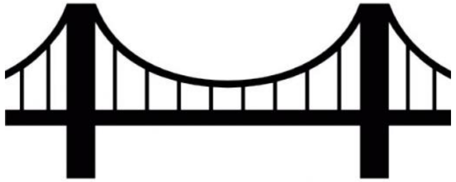
Use theory to inform practice and practice to inform theory.

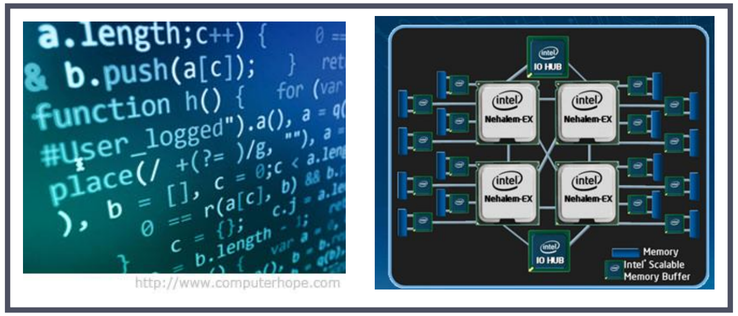
What is Algorithm Engineering?

- Algorithm design
- Algorithm analysis
- Algorithm implementation
- Optimization
- Profiling
- Experimental evaluation



O(n log n)
O(n)
O(log n)



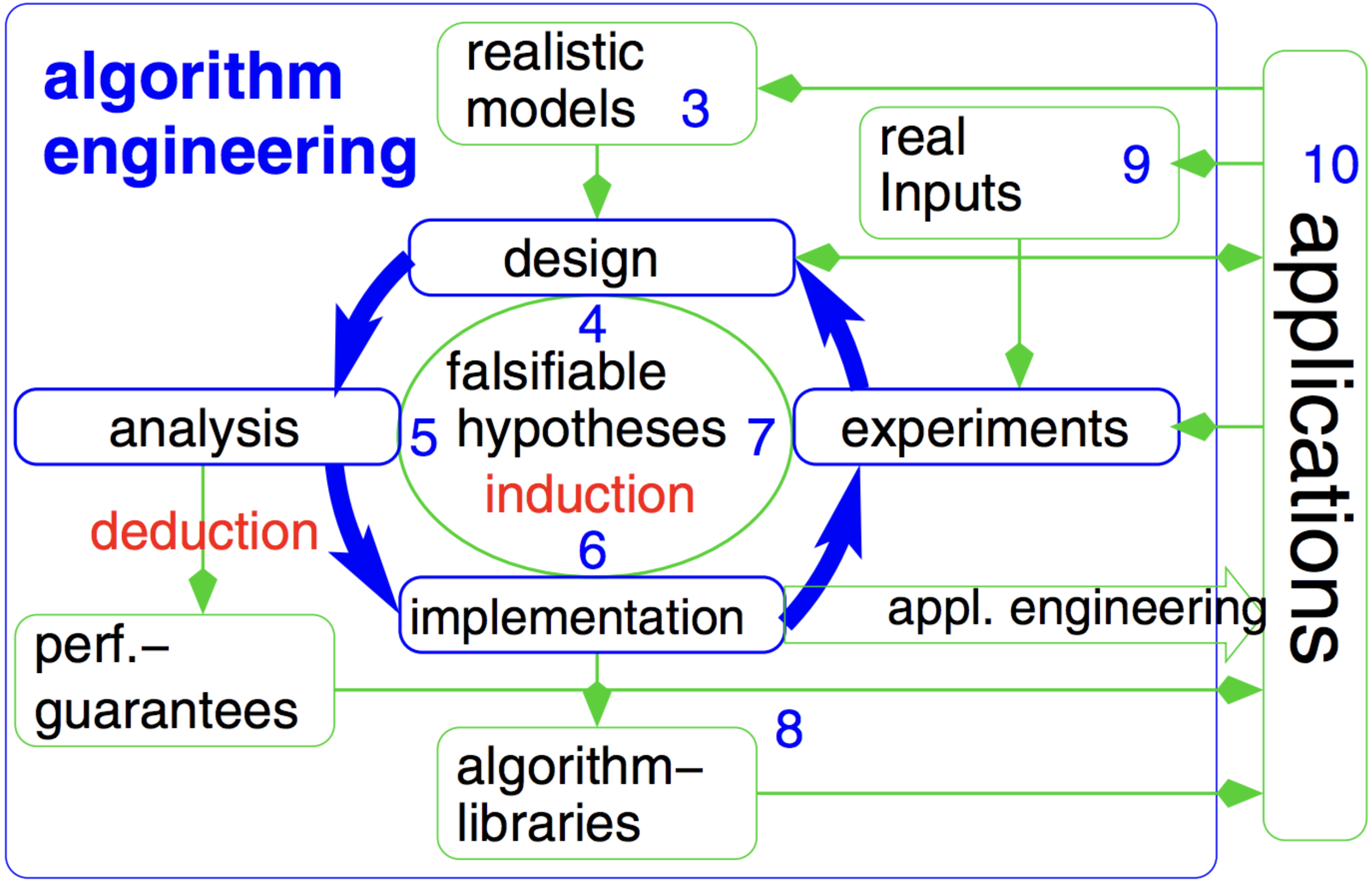


Theory

Practice

Source: MIT 6.886 by Julian Shun

What is Algorithm Engineering?



Source: "Algorithm Engineering – An Attempt at a Definition", Peter Sanders

Source: MIT 6.886 by Julian Shun

Algorithm Design & Analysis

	Algorithm 1	Algorithm 2
Complexity	$N \log_2 N$	$1000\ N$

- **Constant factors matter!**
- **Avoid unnecessary computations**
- **Simplicity improves applicability and can lead to better performance**
- **Think about locality and parallelism**
- **Think both about worst-case and real-world inputs**
- **Use theory as a guide to find practical algorithms**
- **Time vs. space tradeoffs**

Source: MIT 6.886 by Julian Shun

Implementation

- **Write clean, modular code**
  - Easier to experiment with different methods, and can save a lot of development time
- **Write correctness checkers**
  - Especially important in numerical and geometric applications due to floating-point arithmetic, possibly leading to different results
- **Save previous versions of your code!**
  - Version control helps with this

Source: MIT 6.886 by Julian Shun

Experimentation

- **Instrument code with timers and use performance profilers (e.g., perf, gprof, valgrind)**
- **Use large variety of inputs (both real-world and synthetic)**
  - Use different sizes
  - Use worst-case inputs to identify correctness or performance issues
- **Reproducibility**
  - Document environmental setup
  - Fix random seeds if needed
- **Run multiple timings to deal with variance**

Source: MIT 6.886 by Julian Shun

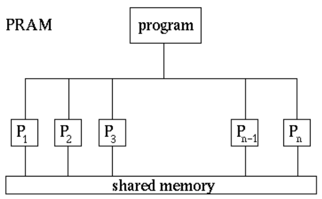
Experimentation II

- **For parallel code, test on varying number of processors to study scalability**
- **Compare with best serial code for problem**
- **For reproducibility, write deterministic code if possible**
  - Or make it easy to turn off non-determinism
- **Use numactl to control NUMA effects on multi-socket machines**

Source: MIT 6.886 by Julian Shun

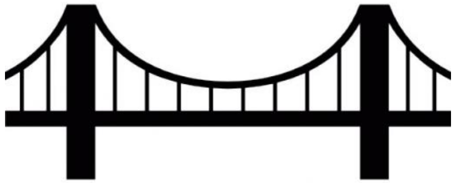
What is Algorithm Engineering?

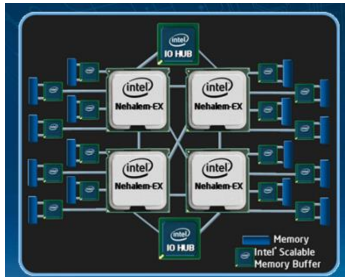
- Algorithm design
- Algorithm analysis
- Algorithm implementation
- Optimization
- Profiling
- Experimental evaluation



O(n log n)
O(n)
O(log n)

Theory





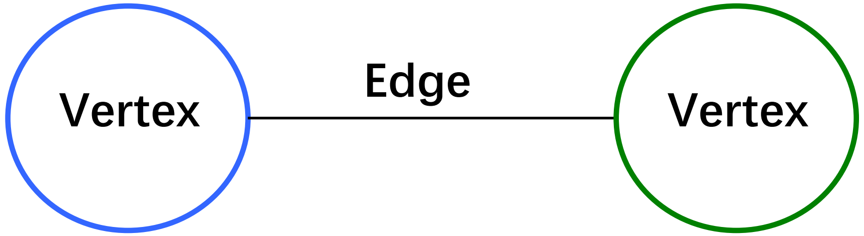
Practice

Source: MIT 6.886 by Julian Shun

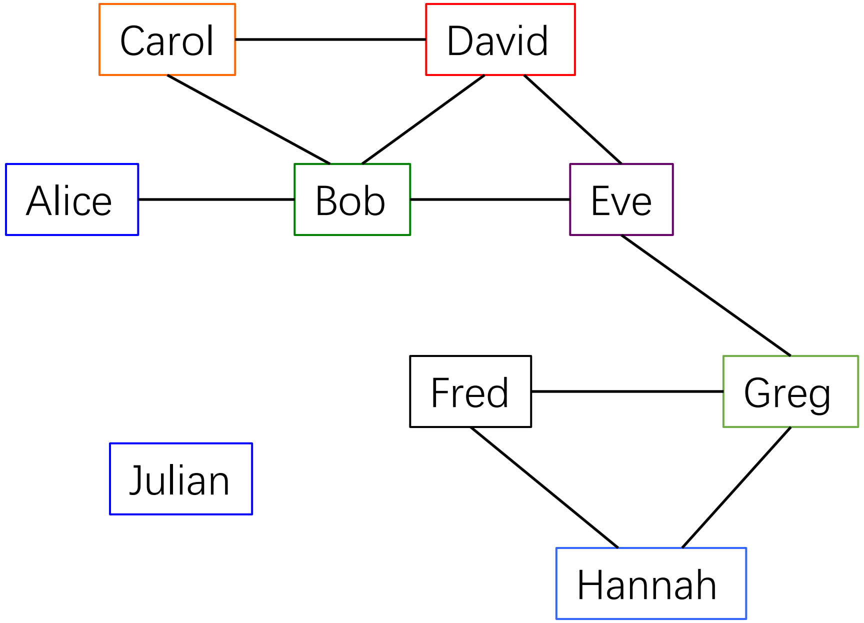
CS260: Algorithm Engineering Lecture 10

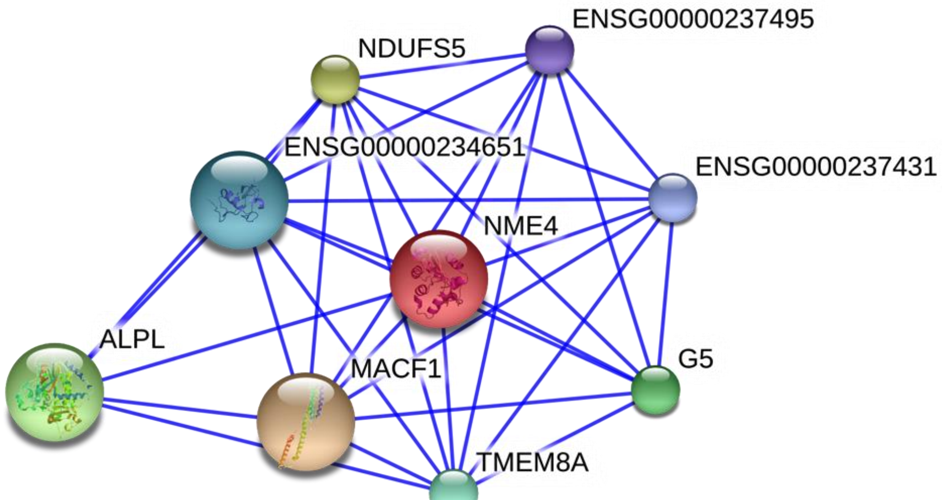
- What is algorithm engineering
- Graphs
- Graph processing systems

What is a graph?



- Vertices model (a set of) objects
- Edges model relationships between objects

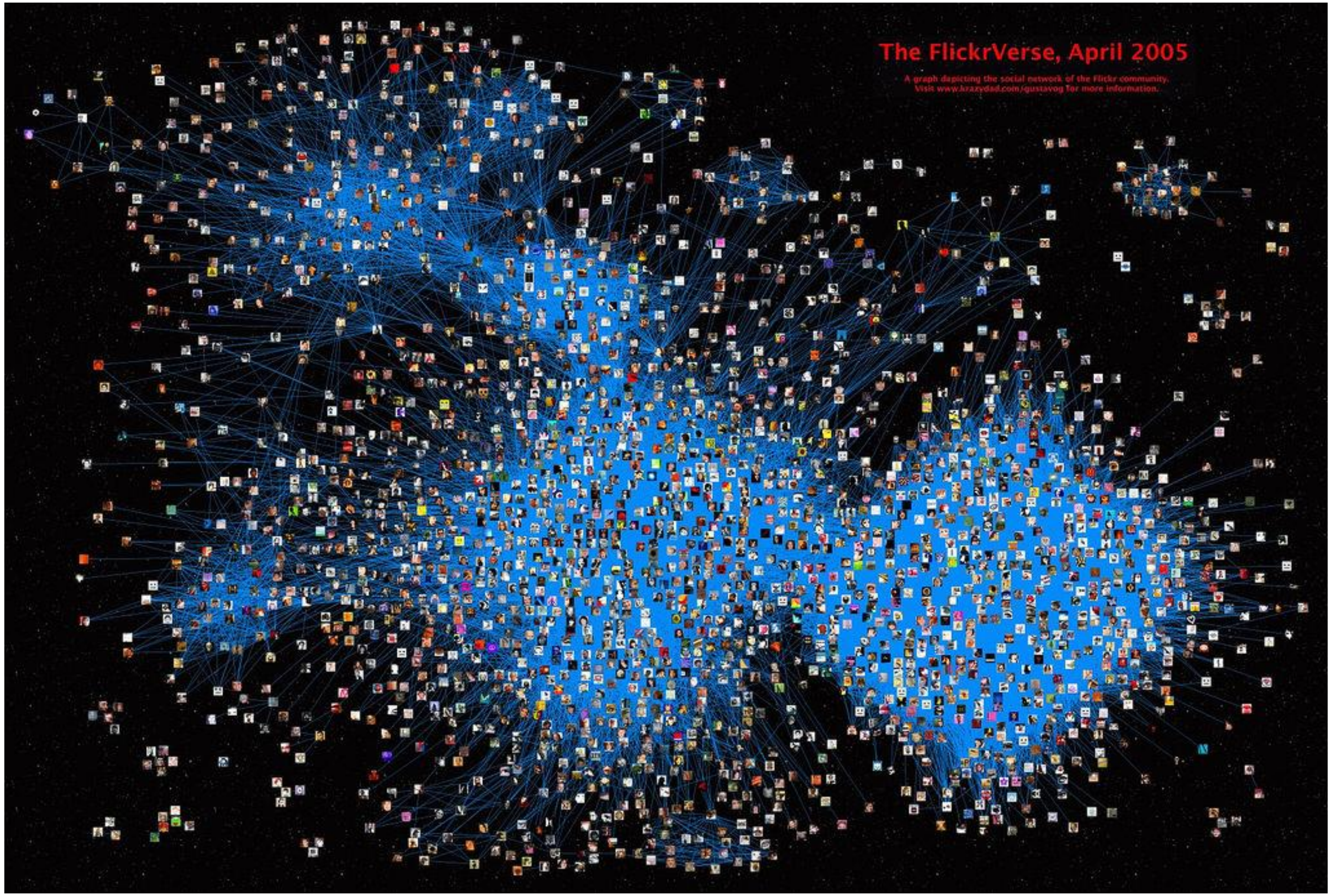




https://commons.wikimedia.org/wiki/File:Protein_Interaction_Network_for_TMEM8A.png

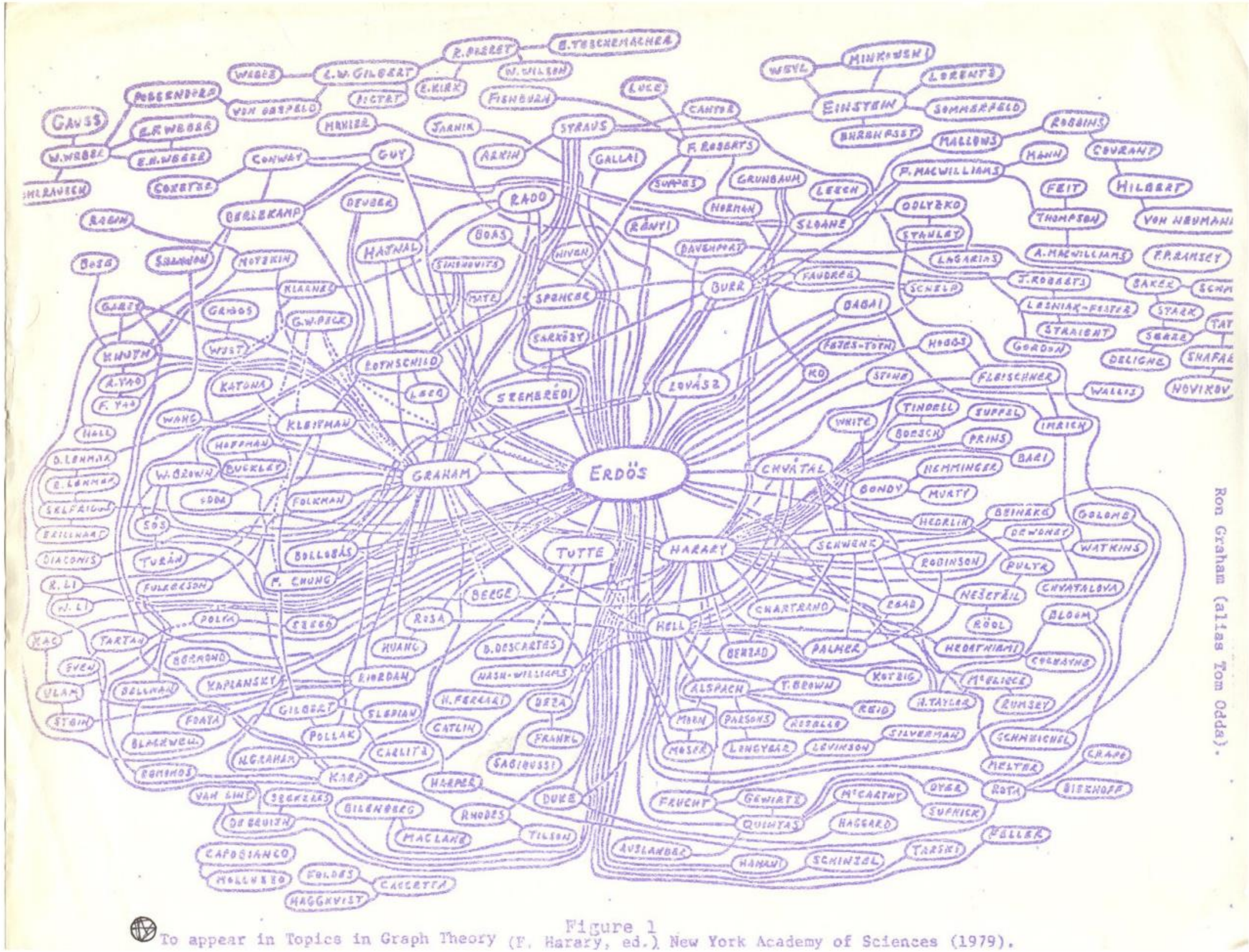
Source: MIT 6.172 by Julian Shun

Social networks



Source: MIT 6.172 by Julian Shun

Collaboration networks

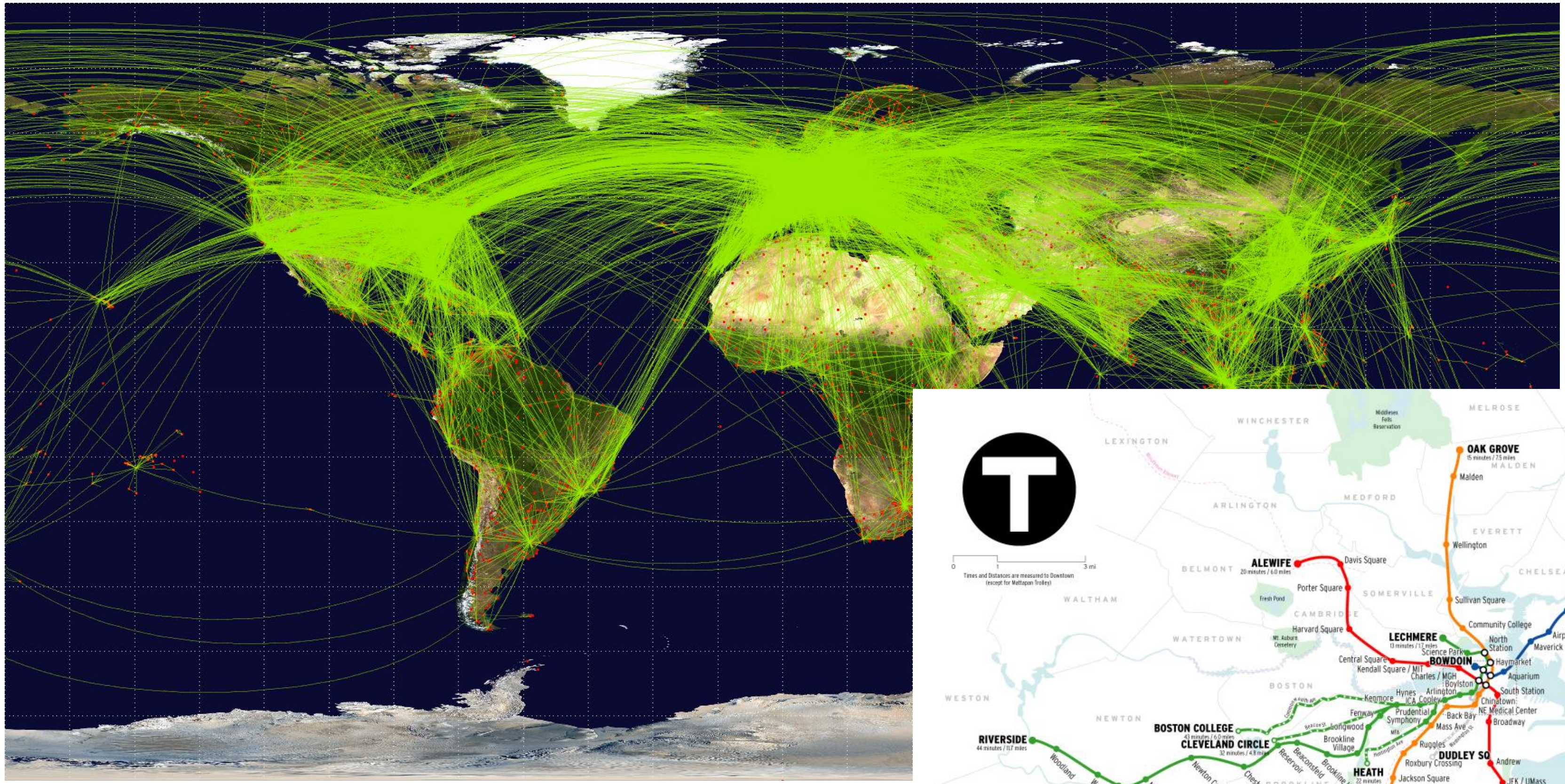


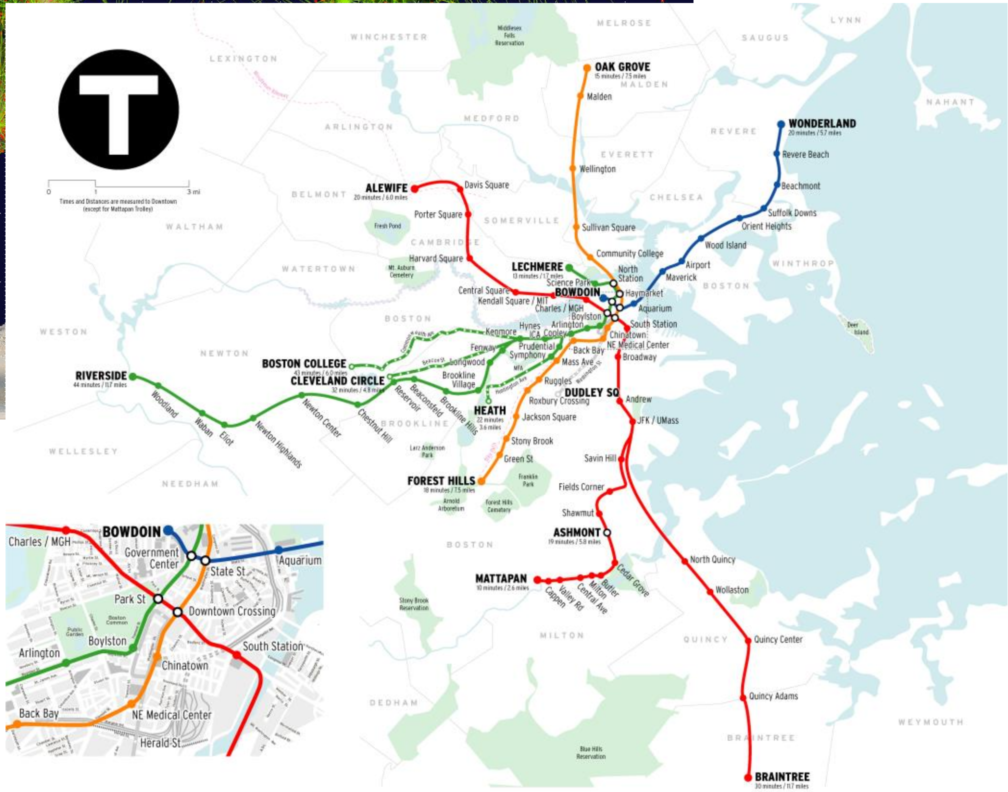
Erdős number:

Number of hops to Erdős via collaboration

Source: MIT 6.172 by Julian Shun

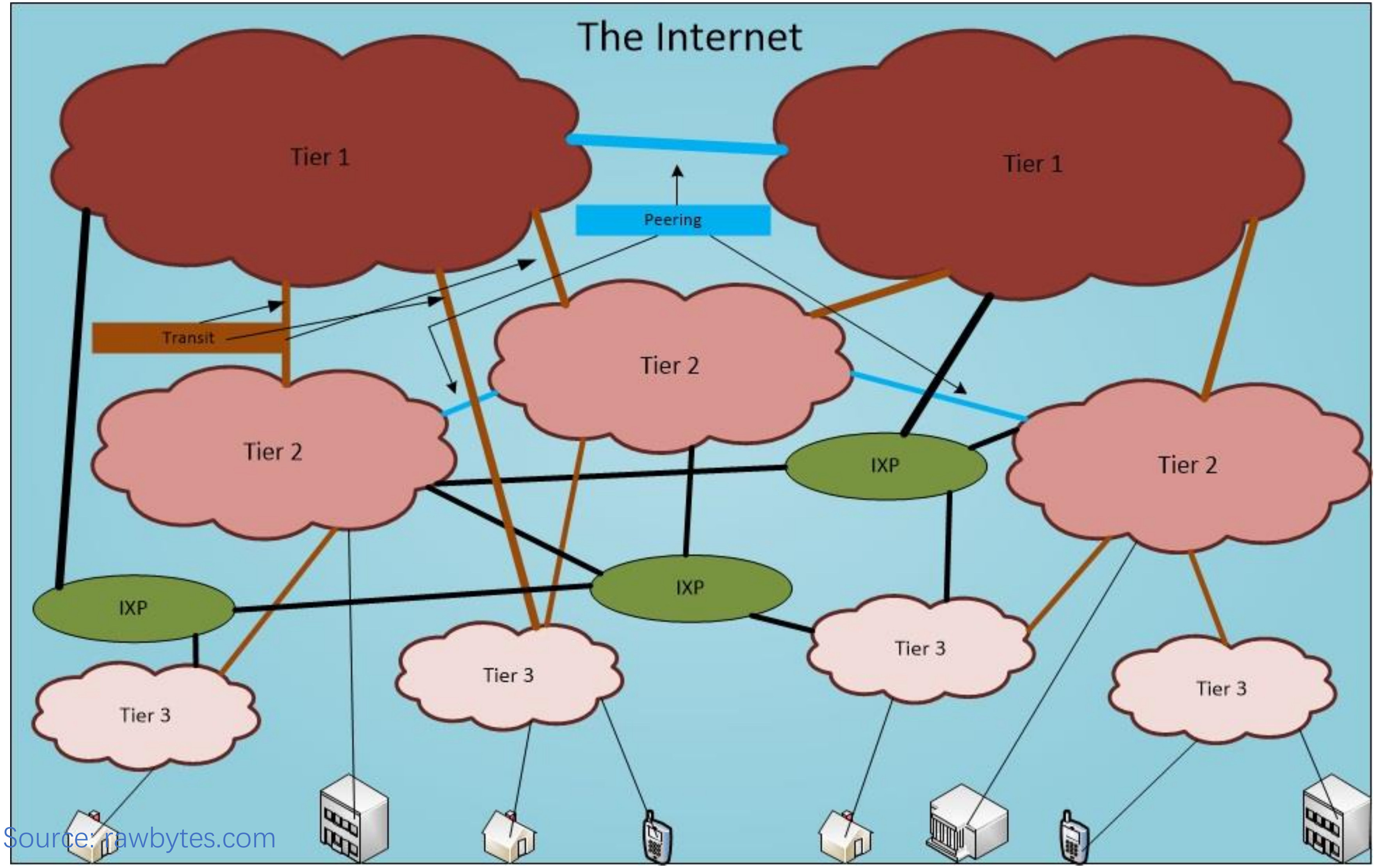
Transportation networks





Source: MIT 6.172 by Julian Shun

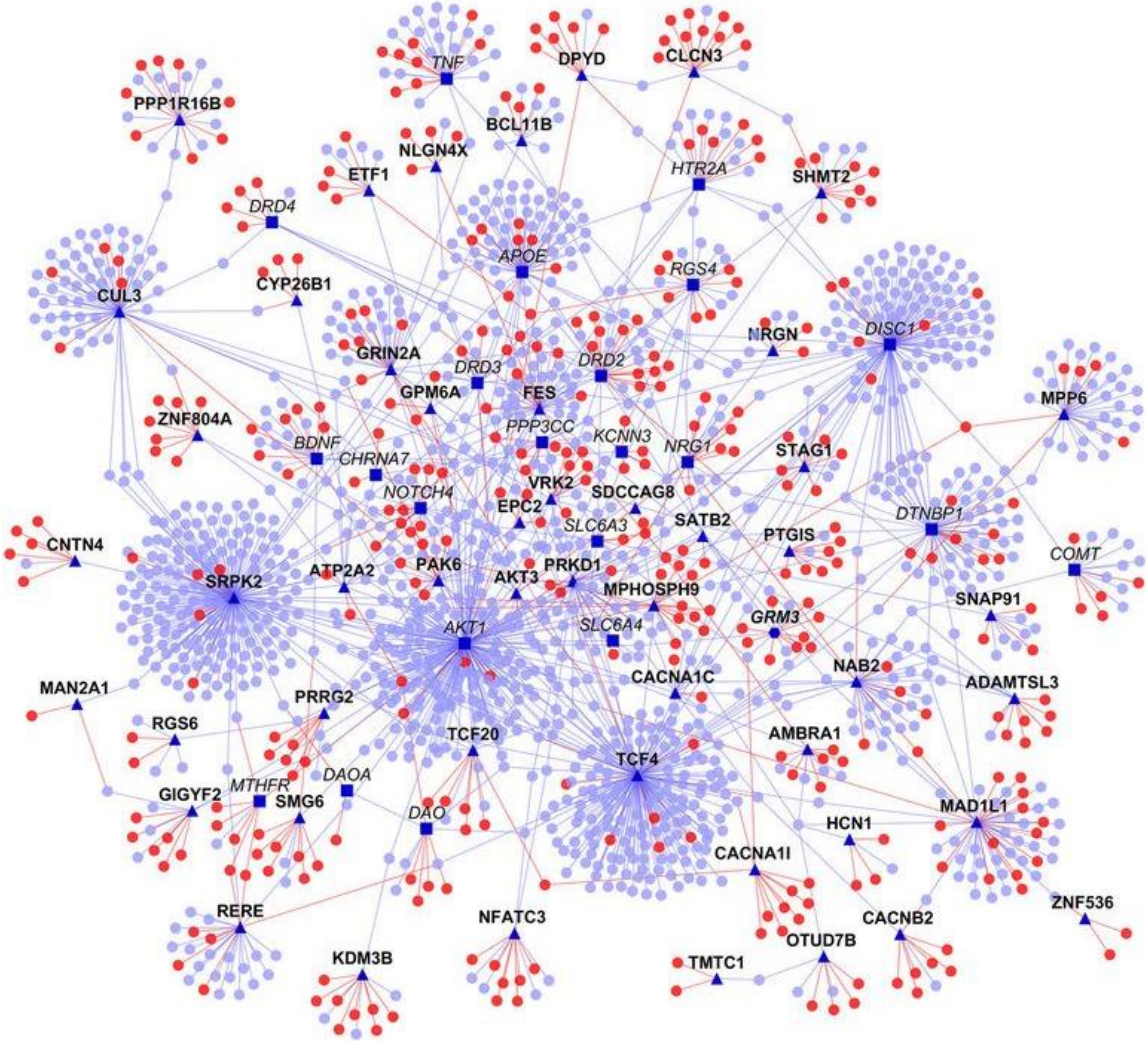
Computer networks



Source: MIT 6.172 by Julian Shun

Biological networks

- Protein–protein interaction (PPI) networks



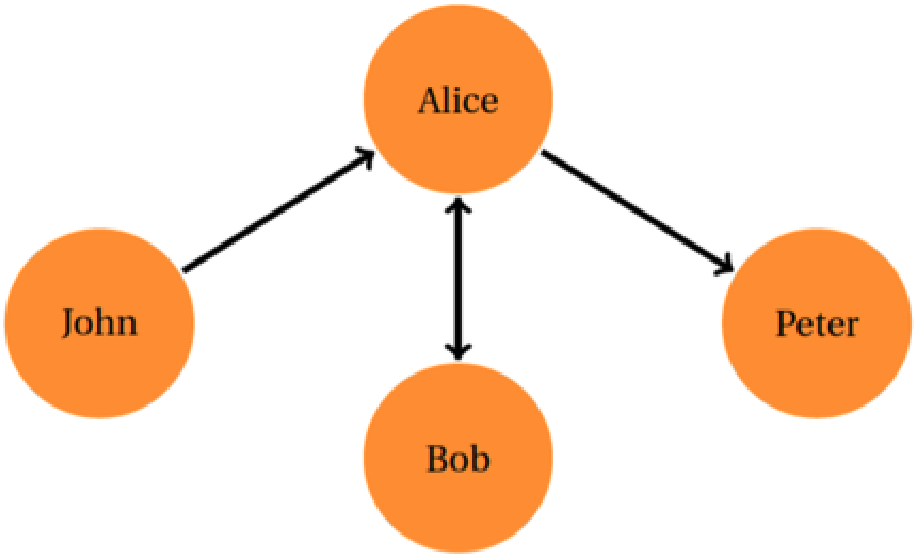
Other Applications

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

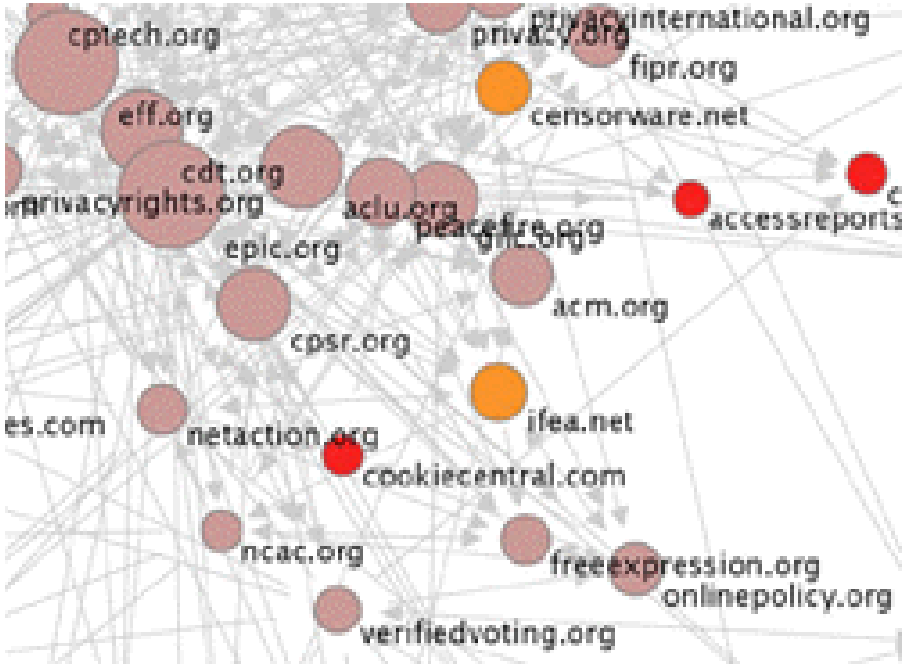
Source: MIT 6.172 by Julian Shun

What is a graph?

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



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



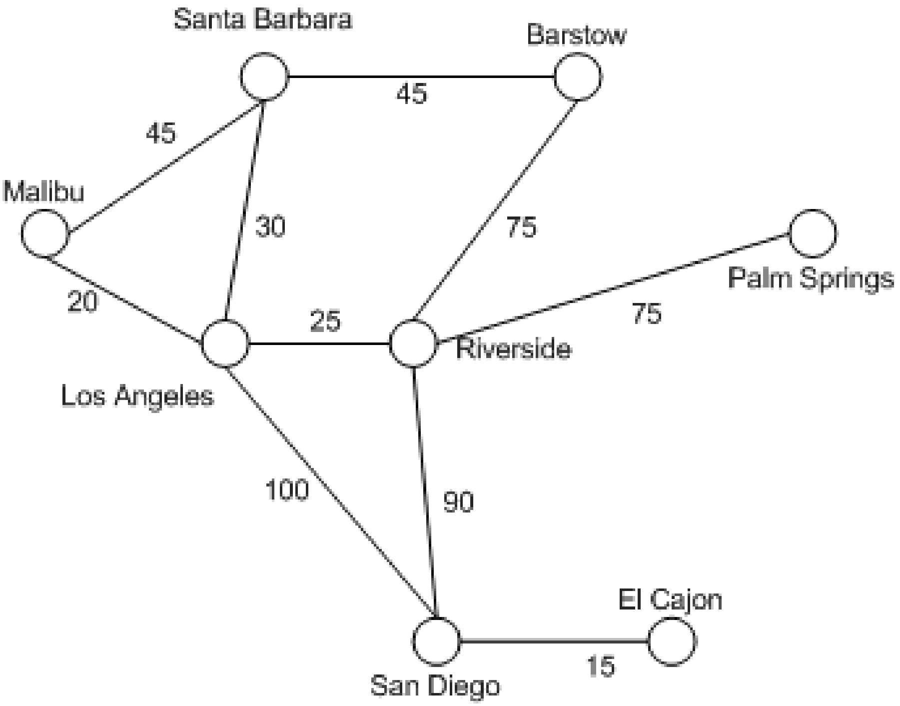
http://farrall.org/papers/webgraph_as_content.html

Source: MIT 6.172 by Julian Shun

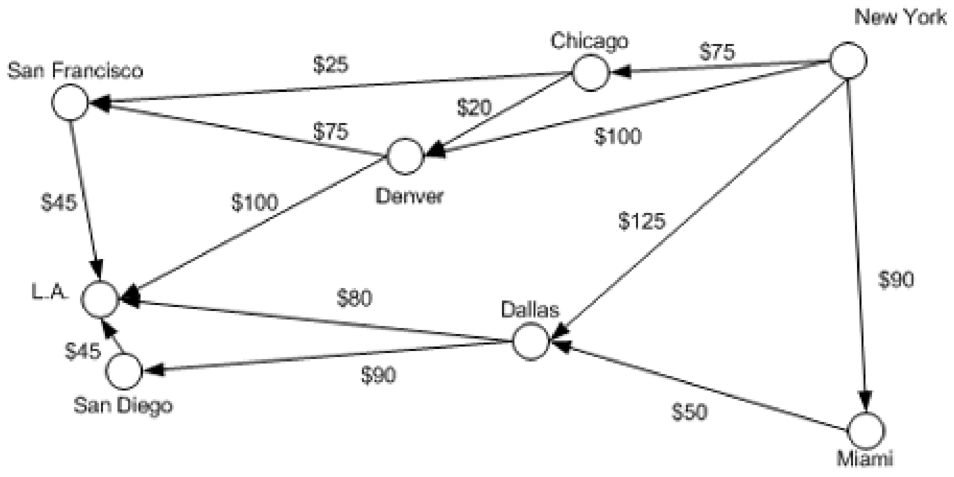
What is a graph?

- **Edges can be weighted / unweighted (unit weighted)**
  - Denotes "strength", distance, etc.

Distance between cities



Flight costs



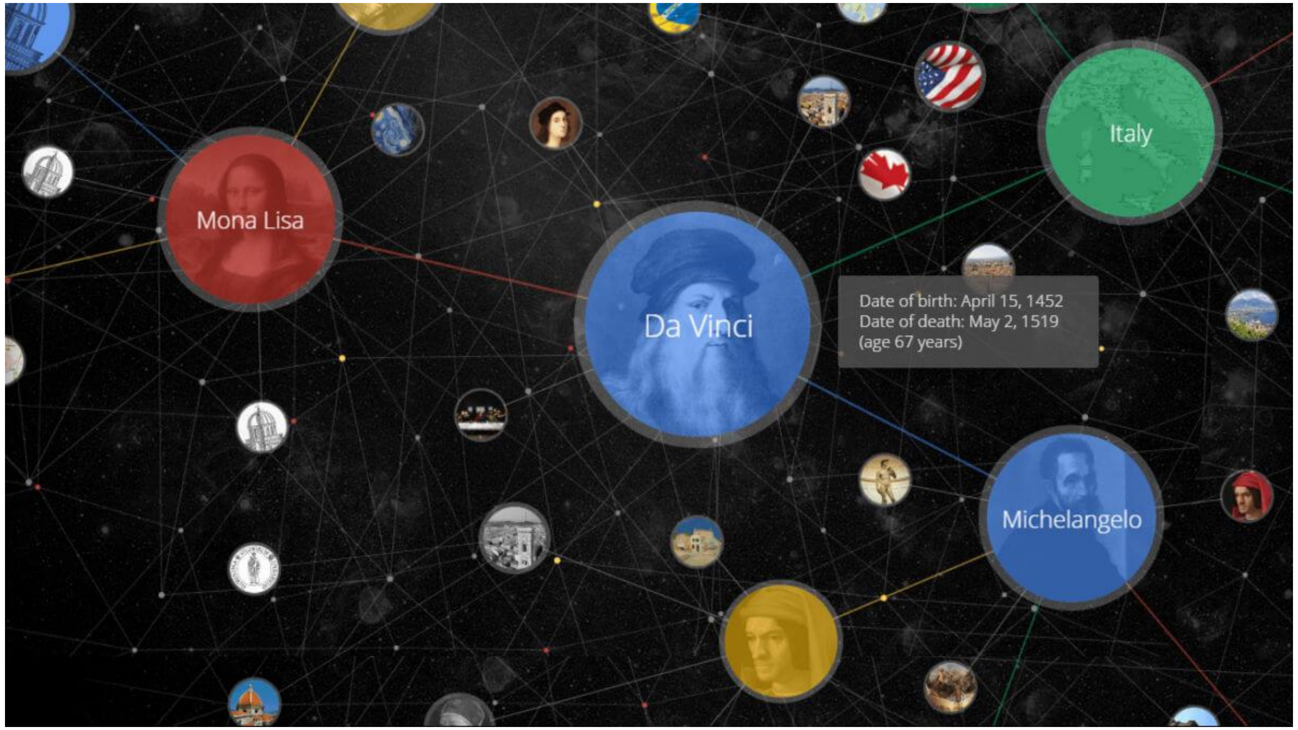
https://msdn.microsoft.com/en-us/library/aa289152(v=vs.71).aspx

Source: MIT 6.172 by Julian Shun

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

Source: MIT 6.172 by Julian Shun

Social network queries

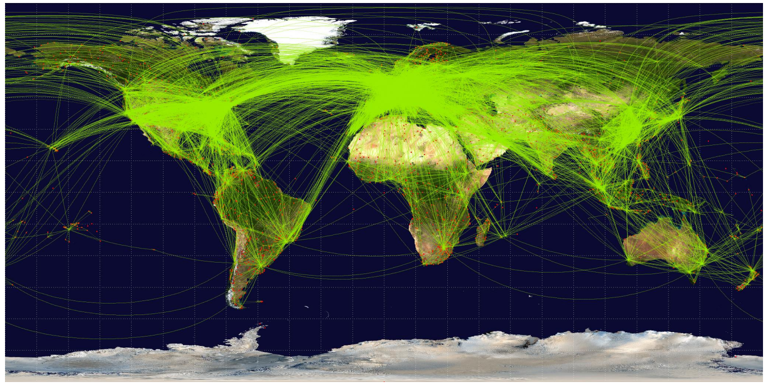
http://www.facebookfever.com/introducing-facebook-new-graph-api-explorer-features/

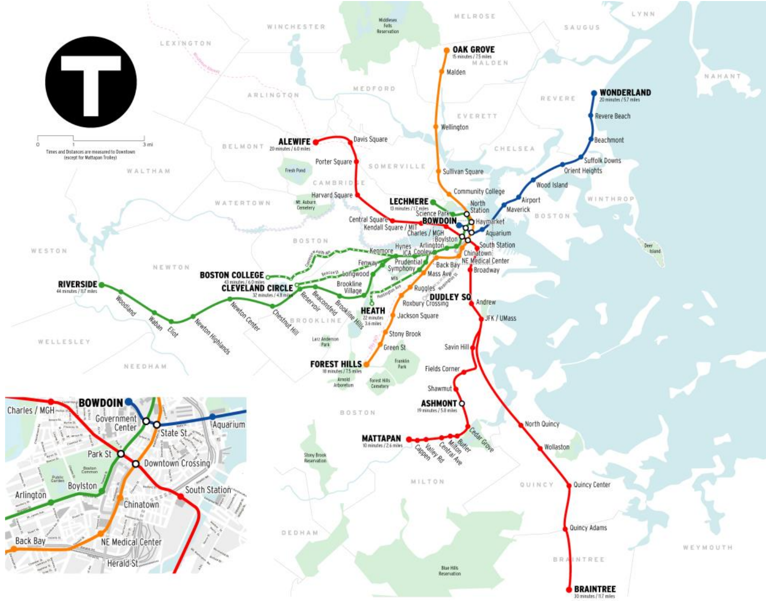
http://allthingsgraphed.com/2014/10/16/your-linkedin-network/

- **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
  - Advertisement recommentations

Source: MIT 6.172 by Julian Shun

Transportation network queries



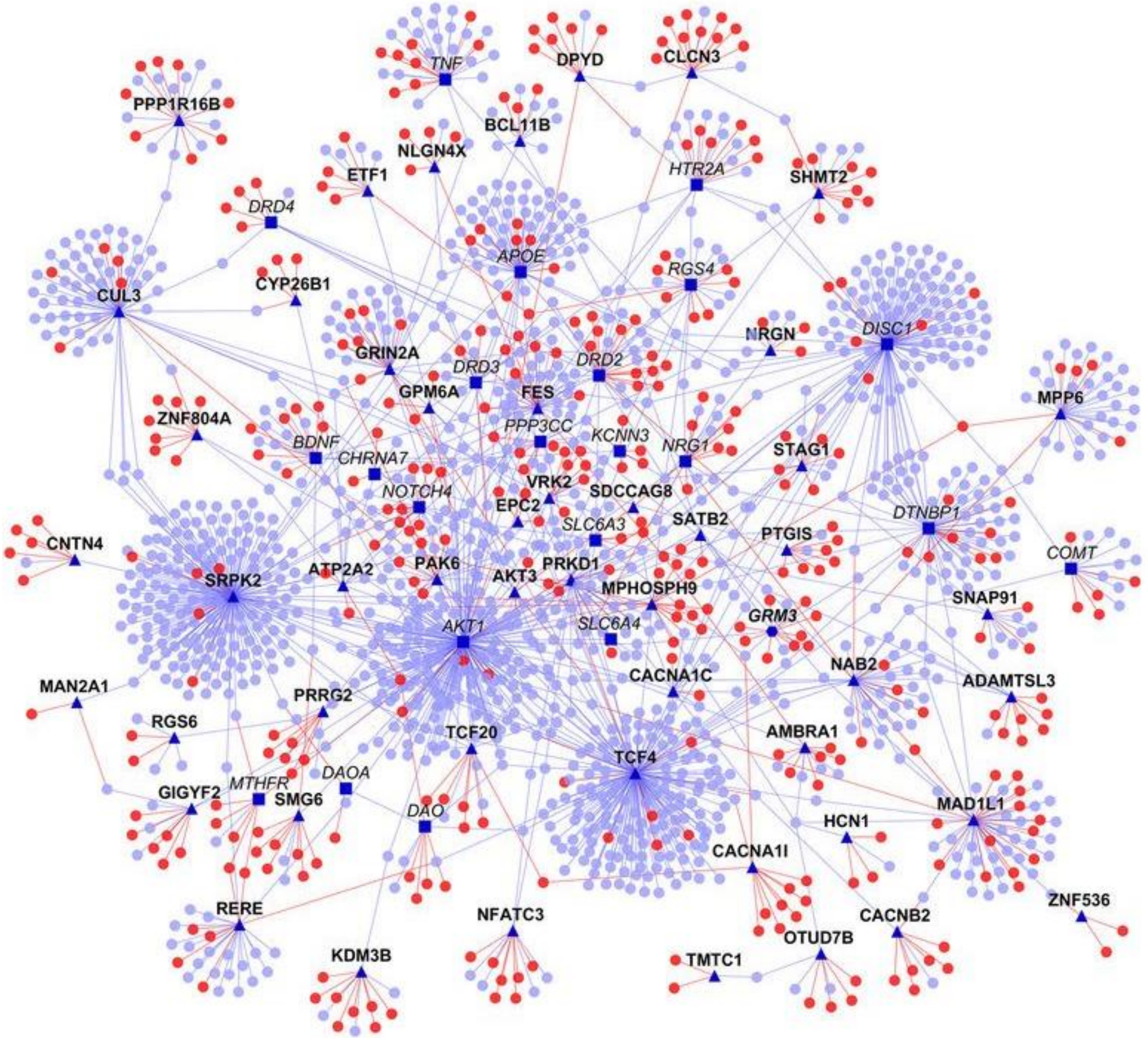


- **Examples:**
  - Find the cheapest way traveling from one city to the other
  - Decide where to build a hub/add a flight to make more profit
  - Find the shortest way to visit a set of locations (e.g., postman)

Source: MIT 6.172 by Julian Shun

Biological network queries

- Example:
  - Find patterns in biological networks
  - Find similarity between different species



Source: UCR CS 260 (214) by Yihan Sun

Graph Problems

	Reachability based	Distance based	Other
Undirected			
Directed			

Graph Problems

	Reachability based	Distance based	Other
Undirected	Breadth-first search (BFS) Connectivity Biconnectivity Spanning forest Low-diameter decomposition (LDD)	Minimum spanning forest / tree (undirected) Single-source shortest-paths (SSSP) All-pair shortest-paths (APSP) Betweenness centrality (BC) Spanner / Hopset	Maximal independent set (MIS) Matching Graph coloring Coreness Isomorphism
Directed	Strongly Connected Components (SCC)		Page rank

- Planar graphs (graphs that can be drawn on a plain)
- Dynamic graphs (can change over time)

Real-world graph sizes in 2019

Graph	Num. Vertices	Num. Undirected Edges
soc-LiveJournal	4.8M	85M
com-Orkut	3M	234M
Twitter	41M	2.4B
Facebook (2011) [1]	721M	68.4B
Hyperlink2014 [2]	1.7B	124B
Hyperlink2012 [2]	3.5B	225B
Facebook (2018)	> 2B	> 300B
Yahoo!	272B	5.9T
Google (2018)	> 100B	6T
Brain Connectome	100B (neurons)	100T (connections)

●: Publicly available graphs

●: Private graph datasets

[1] The Anatomy of the Facebook Social Graph, Ugander et al. 2011
[2] http://webdatacommons.org/hyperlinkgraph/

Source: CMU 15-853 by Laxman Dhulipala

Graph Representation

Graph Representations

- Vertices labeled from 0 to n−1

$n = \#$ of vertices
$m = \#$ of edges

$O(n^2)$

	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)

$O(m)$

Edge list

- Space?

Source: MIT 6.172 by Julian Shun

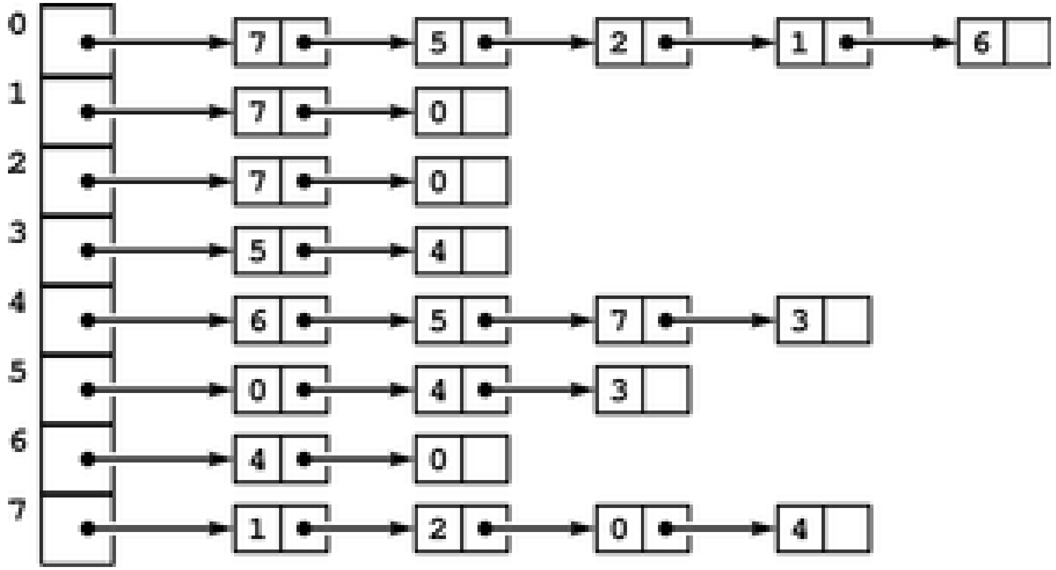
Graph Representations

n = # of vertices
m = # of edges

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

Space requirement?

O(n+m)



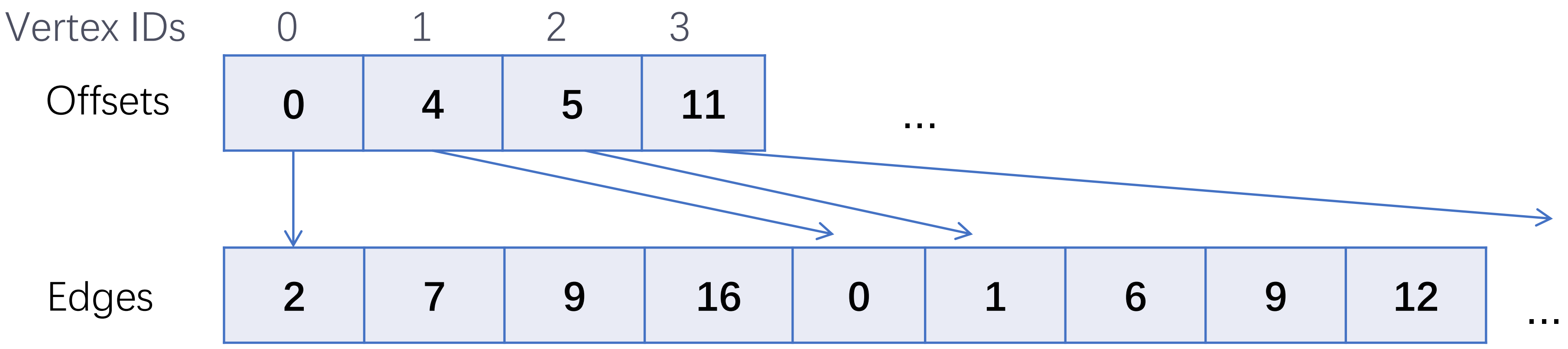
- Can substitute linked lists with arrays for better cache performance
  - Tradeoff: more expensive to update graph

Source: MIT 6.172 by Julian Shun

Graph Representations

$n = \#$ of vertices
$m = \#$ of edges

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



- How do we compute the offset array?
- Space?
  - O(n+m)

Source: MIT 6.172 by Julian Shun

CS260: Algorithm Engineering Lecture 10

- What is algorithm engineering
- Graphs
- Graph processing systems

Graph Processing Frameworks

- **Provides high level primitives for graph algorithms**
- **Reduce programming effort of writing efficient parallel graph programs**

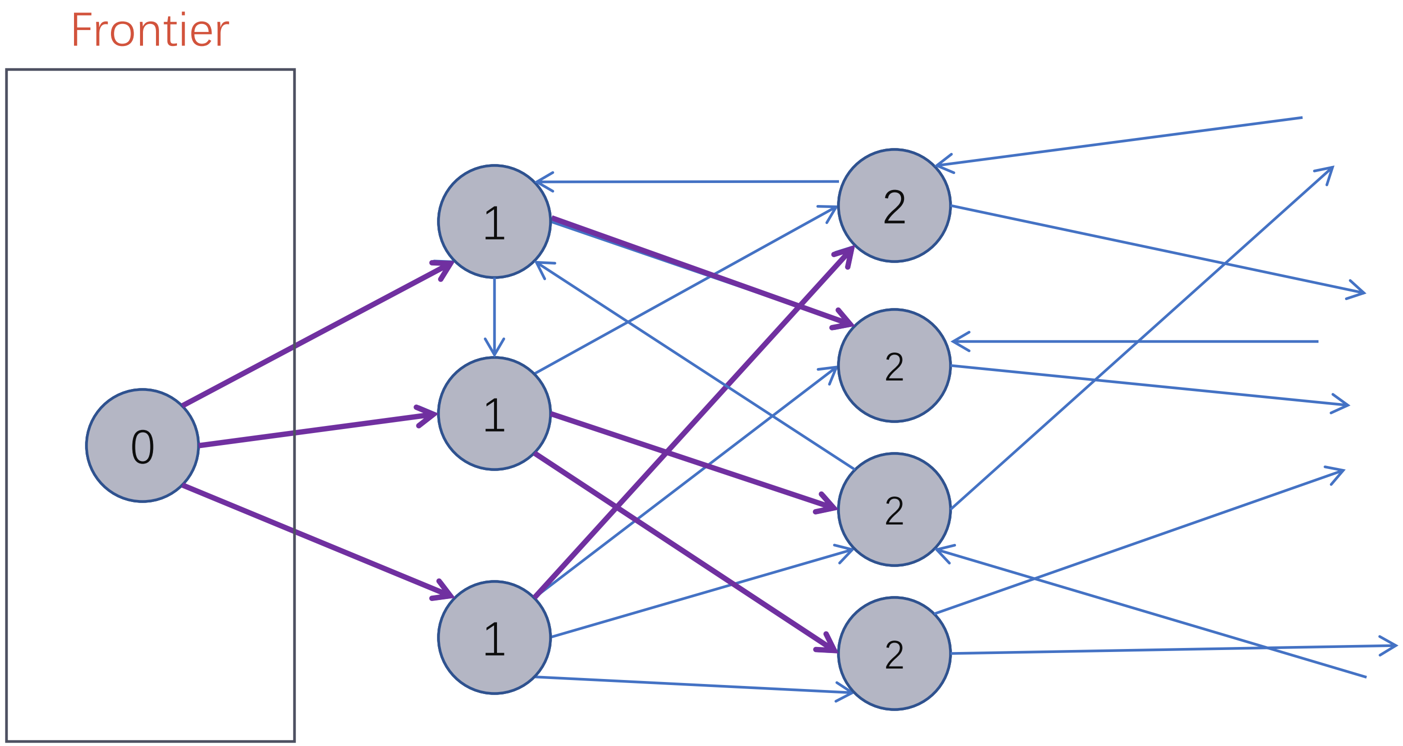
Graph processing frameworks/libraries

Pregel, Giraph, GPS, GraphLab, PowerGraph, PRISM, Pegasus, Knowledge Discovery Toolbox, CombBLAS, GraphChi, GraphX, Galois, X-Stream, Gunrock, GraphMat, Ringo, TurboGraph, TurboGraph++, FlashGraph, Grace, PathGraph, Polymer, GPSA, 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, cuStinger, Distinger, Hornet, 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, GraphIt, GraPu, GraphJet, ImmortalGraph, LA3, CellIQ, AsyncStripe, Cgraph, GraphD, GraphH, ASAP, RStream, and many others…

Four papers about graph processing systems in CS 260

- **Ligra: a lightweight graph processing framework for shared memory**
  - Frontier-based algorithms similar to BFS
- **Julienne: A Framework for Parallel Graph Algorithms using Work-efficient Bucketing, by Zhongqi Wang**
  - Distance-based algorithms such as SSSP, k-core
- **Aspen: Low-Latency Graph Streaming Using Compressed Purely-Functional Trees, by Xiaojun Dong**
  - Graph processing systems for dynamic graphs
- **Sage: Semi-Asymmetric Parallel Graph Algorithms for NVRAMs, by Kristian Tram**
  - Graph processing systems optimized for non-volatile main memories

Parallel BFS Algorithm



Source: MIT 6.172 by Julian Shun

Ligra: based on shared-memory multicore machines

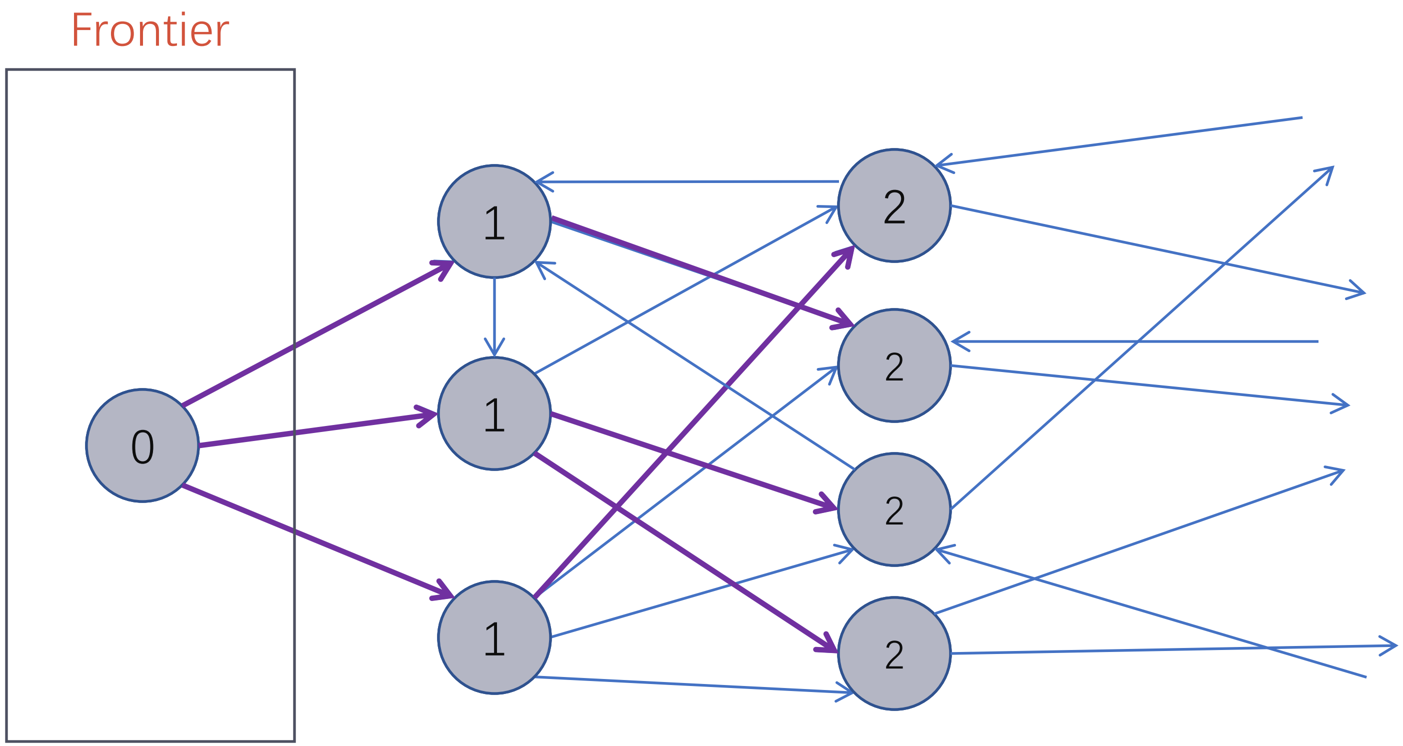
- **Motivating example: breadth-first search**

```
parents = {-1, ..., -1}
// d = dst: vertex to "update" (just encountered)
// s = src: vertex on frontier with edge to d

procedure UPDATE(s, d)
    return compare-and-swap(parents[d], -1, s);

procedure COND(i)
    return parents[i] == -1;

procedure BFS(G, r)
    parents[r] = r;
    frontier = {r};
    while (size(frontier) != 0) do:
        frontier = EDGEMAP(G, frontier, UPDATE, COND);
```

Semantics of EDGEMAP:
Foreach vertex i in frontier, call UPDATE for all neighboring vertices j for which COND(j) is true. Add j to returned set if UPDATE(i, j) returns true

Source: Stanford CS 149 by Kayvon Fatahalian

Parallel BFS Algorithm



- **Can process each frontier in parallel**
  - Parallelize over both the vertices and their outgoing edges

Implementing EDGEMAP

- Assume the frontier is small

graph

set of vertices (previous frontier)

condition check on neighbor vertex

update function on neighbor vertex

```
procedure EDGEMAP_FORWARD(G, U, F, C):
   result = {}
   parallel foreach v in U do:
      parallel foreach v2 in out_neighbors(v) do:
         if (C(v2) == 1 and F(v,v2) == 1) then
            add v2 to result
   remove duplicates from result
   return result
```

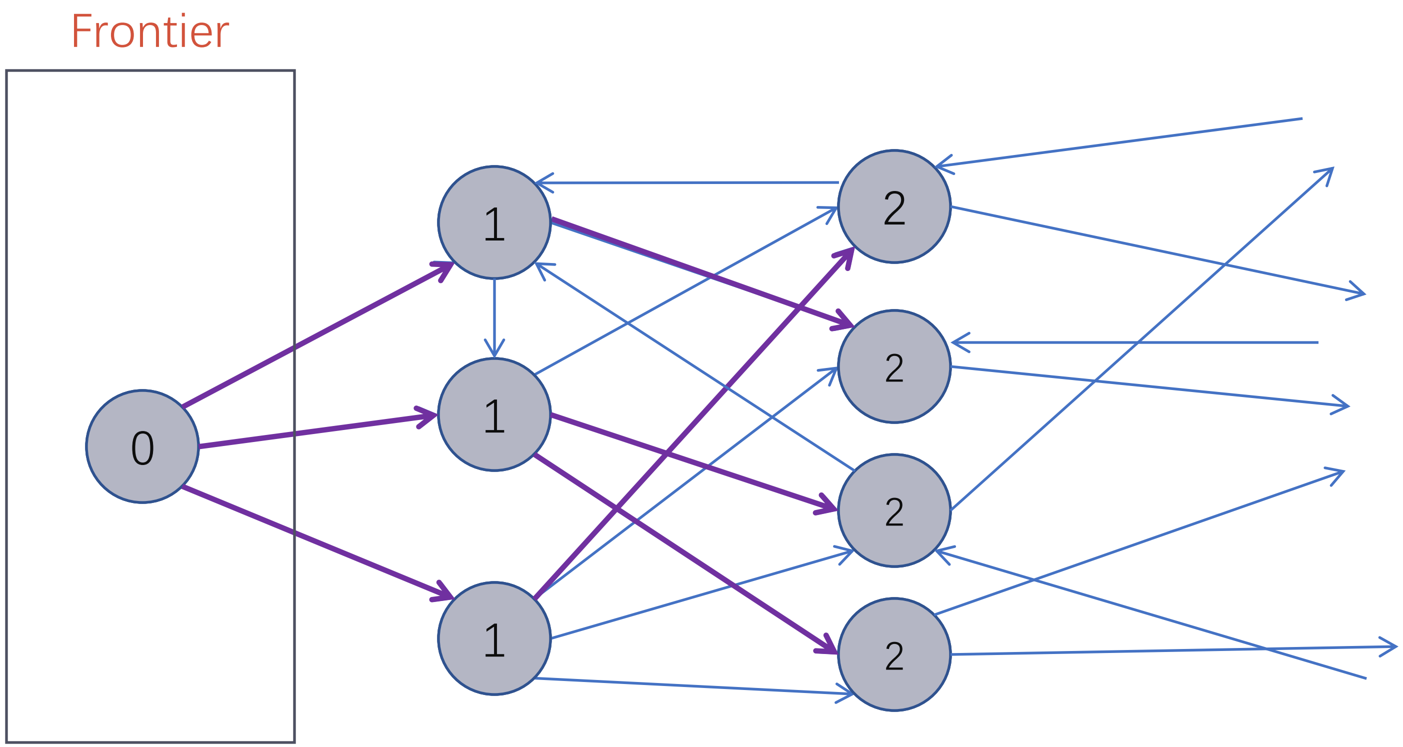
Work:
O(|U| + sum of outgoing edges from U)

Span: polylogarithmic

```
parents = {-1, ..., -1}
// d = dst: vertex to "update" (just encountered)
// s = src: vertex on frontier with edge to d

procedure UPDATE(s, d)
    return compare-and-swap(parents[d], -1, s);

procedure COND(i)
    return parents[i] == -1;

procedure BFS(G, r)
    parents[r] = r;
    frontier = {r};
    while (size(frontier) != 0) do:
        frontier = EDGEMAP(G, frontier, UPDATE, COND);
```

Visiting every edge on frontier can be wasteful

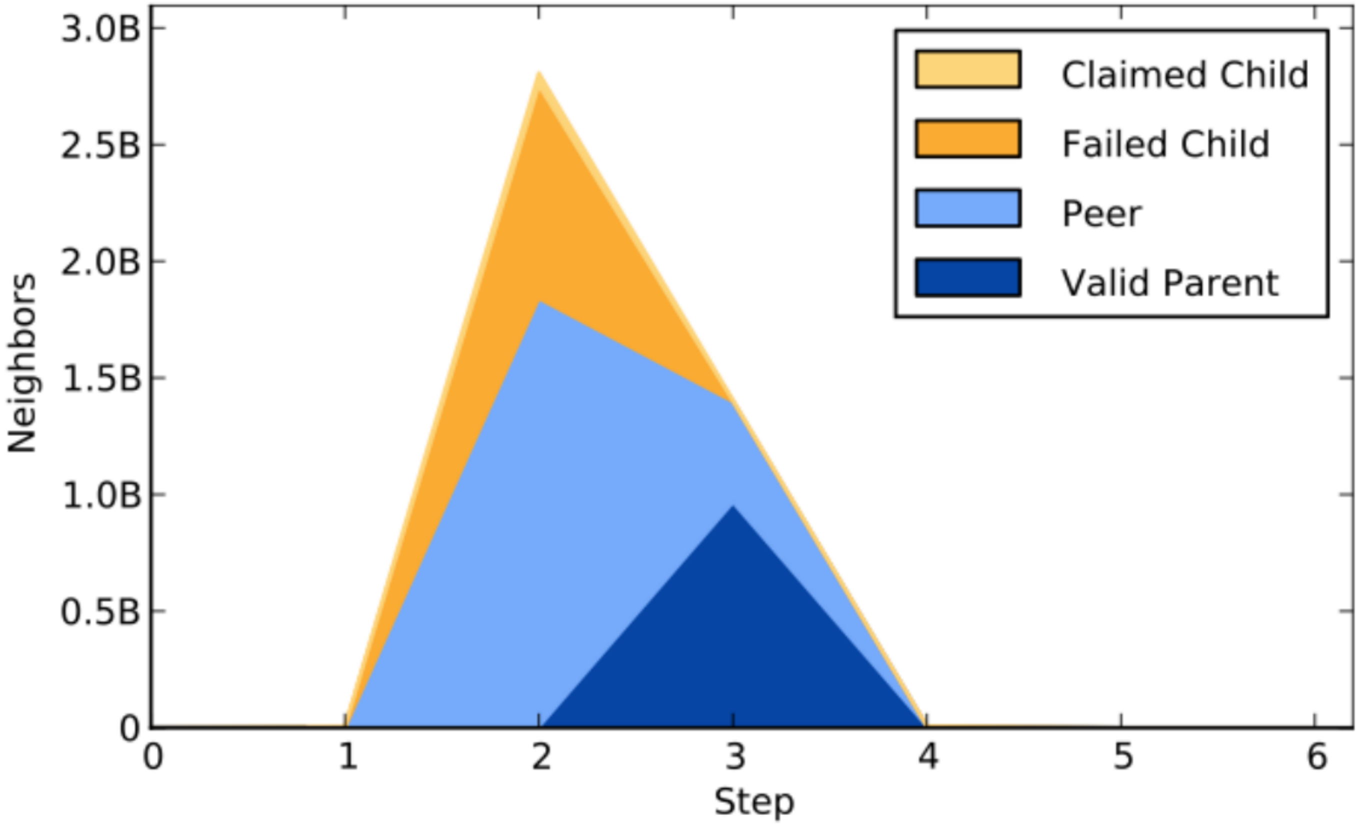
- **Each step of BFS, every edge on frontier is visited**
  - Frontier can grow quickly for social graphs (few steps to visit all nodes)
  - Most edge visits are wasteful! (they don't lead to a successful "update")



Visiting every edge on frontier can be wasteful

- **Each step of BFS, every edge on frontier is visited**
  - Frontier can grow quickly for social graphs (few steps to visit all nodes)
  - Most edge visits are wasteful! (they don't lead to a successful "update")

- **claimed child: edge points to unvisited vertex (useful work)**
- **failed child: edge points to vertex found in this step via another edge**
- **peer: edge points to a vertex that was added to frontier in same step as current vertex**
- **valid parent: edge points to vertex found in previous step**



 [Credit: Beamer et al. SC12]

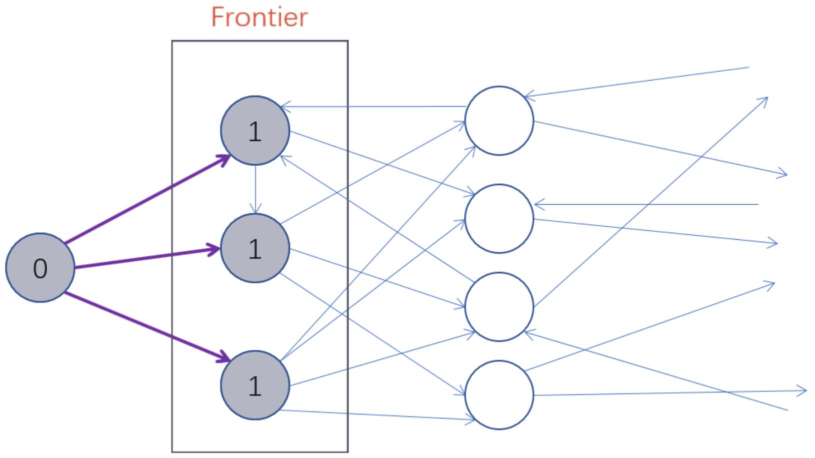
Implementing EDGEMAP for large frontier size

- **Assume the frontier is large**

```
procedure EDGEMAP_FORWARD(G, U, F, C):
   result = {}
   parallel foreach v in U do:
      foreach v2 in out_neighbors(v) do:
         if (C(v2) == 1 and F(v,v2) == 1) then
            add v2 to result
   remove duplicates from result
   return result
```

```
procedure EDGEMAP_BACKWARD(G, U, F, C):
  result = {}
  parallel foreach v in V do:
    if (C(v) == 1)
      foreach v2 in in_neighbors(v) do:
        if (v2∈U and F(v2,v) == 1) then
          add v to result and break
  pack the result and return
```

Work for a round:
Still can be as large as O(|E|),
but usually less than that since once the loop can quit once one of the in-neighbors is visited



Page rank in Ligra

```
r_cur = {1/|V|, ... 1/|V|};
r_next = {0,...,0};
diff = {}

procedure PRUPDATE(s, d):
   atomicIncrement(&r_next[d], r_cur[s] / vertex_degree(s));

procedure PRLOCALCOMPUTE(i):
   r_next[i] = alpha * r_next[i] + (1 - alpha) / |V|;
   diff[i] = |r_next[i] - r_cur[i]|;
   r_cur[i] = 0;
   return 1;

procedure COND(i):
   return 1;

procedure PAGERANK(G, alpha, eps):
   frontier = {0, ... , |V|-1}
   error = HUGE;
   while (error > eps) do:
      frontier = EDGEMAP(G, frontier, PRUPDATE, COND);
      frontier = VERTEXMAP(frontier, PRLOCALCOMPUTE);
      error = sum of per-vertex diffs // this is a parallel reduce
      swap(r_cur, r_next);
   return err
```

Ligra summary

- System abstracts graph operations over vertices and edges
  - Frontier-based graph traversal algorithms

- These basic operations permit a surprisingly wide space of graph algorithms:
  - Betweenness centrality
  - Connected components
  - Single-source shortest paths (Bellman-Ford)
  - graph radii estimation

Ligra: a Lightweight Framework for Graph Processing for Shared Memory [Shun and Blelloch 2013]

Ligra

- **Simple library with many useful examples**

Examples

Implementation files are provided in the apps/ directory:

- **BFS.C** (breadth-first search)
- **BFS-Bitvector.C** (breadth-first search with a bitvector to mark visited vertices)
- **BC.C** (betweenness centrality)
- **Radii.C** (graph eccentricity estimation)
- **Components.C** (connected components)
- **BellmanFord.C** (Bellman-Ford shortest paths)
- **PageRank.C**
- **PageRankDelta.C**
- **BFSCC.C** (connected components based on BFS)
- **KCore.C** (computes k-cores of the graph)

Eccentricity Estimation

Code for eccentricity estimation is available in the apps/eccentricity/ directory:

- **kBFS-Ecc.C** (2 passes of multiple BFS's)
- **kBFS-1Phase-Ecc.C** (1 pass of multiple BFS's)
- **FM-Ecc.C** (estimation using Flajolet-Martin counters; an implementation of a variant of HADI from *TKDD '11*)
- **LogLog-Ecc.C** (estimation using LogLog counters; an implementation of a variant of HyperANF from *WWW '11*)
- **RV.C** (parallel implementation of the algorithm by Roditty and Vassilevska Williams from *STOC '13*)

Elements of good graph processing system design (and other domain-specific systems)

#1: good systems identify the most important cases, and provide most benefit in these situations

- **Structure of code mimics the natural structure of problems in the domain**
  - Graph processing algorithms are designed in terms of per-vertex operations

- **Efficient expression: common operations are easy and intuitive to express**

- **Efficient implementation: the most important optimizations in the domain are performed by the system for the programmer**

#2: good systems are usually simple systems

- **They have a small number of key primitives and operations**
    - Ligra: only two operations! (vertexmap and edgemap)

- **Allows compiler/runtime to focus on optimizing these primitives Provide parallel implementations, utilize appropriate hardware**

- **Common question that good architects ask: “do we really need that?” (can this concept be reduced to a primitive we already have?)**
    - Better theoretical bounds / performance

#3: good primitives compose

- **Composition of primitives allows for wide application scope, even if scope remains limited to a domain**
  - Ligra supports a wide variety of graph algorithms
- **Composition often allows optimization to generalizable**
  - If system can optimize A and optimize B, then it can optimize programs that combine A and B
- **Sign of a good design**
  - System ultimately is used for applications original designers never anticipated

Wednesday's lecture

- **Julienne: A Framework for Parallel Graph Algorithms using Work-efficient Bucketing, by Zhongqi Wang**
  - Distance-based algorithms such as SSSP, k-core
- **Aspen: Low-Latency Graph Streaming Using Compressed Purely-Functional Trees, by Xiaojun Dong**
  - Graph processing systems for dynamic graphs
- **Sage: Semi-Asymmetric Parallel Graph Algorithms for NVRAMs, by Kristian Tram**
  - Graph processing systems optimized for non-volatile main memories