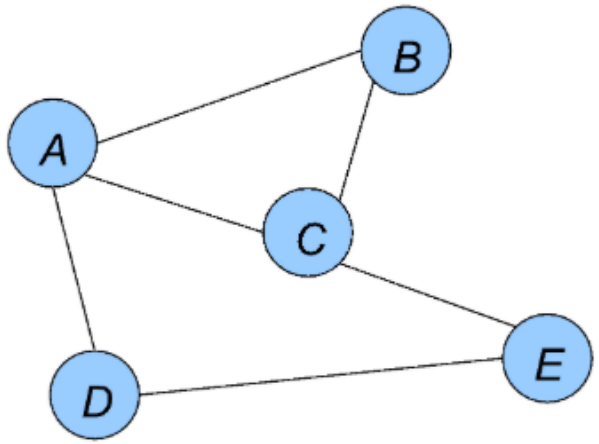
UMD DATA605 - Big Data Systems

12.1: Graph Data Management

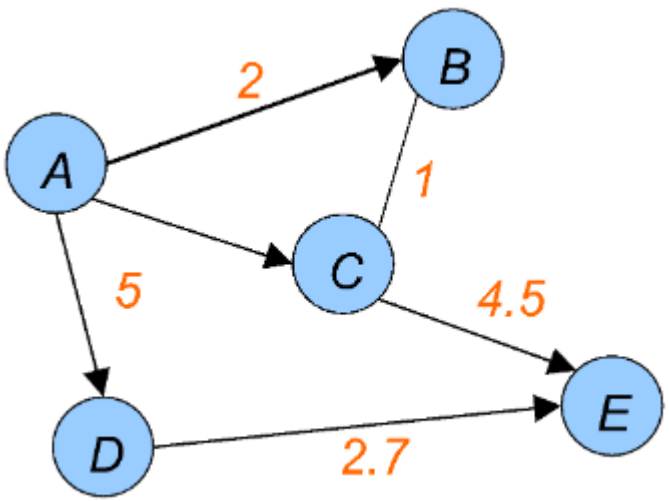
- **Instructor**: Dr. GP Saggese, gsaggese@umd.edu

Graphs: Background

- A **graph** represents entities and their connections
  - Entities are *vertices* (or nodes)
  - Connections are *edges* (or links, arcs, relationships)
- **Applications in many fields**
  - Social networks
  - Biological networks
  - Information networks
  - Infrastructure networks
  - ...



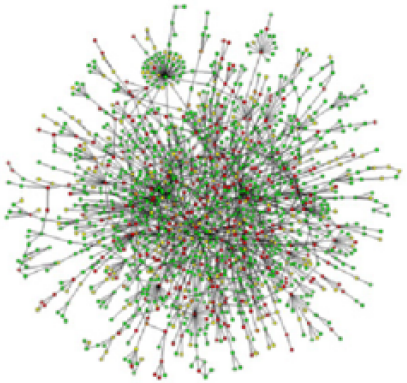
An undirected, unweighted graph



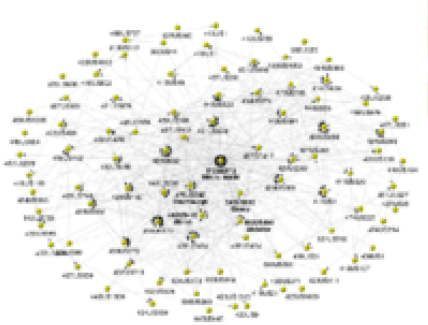
A directed, edge-weighted graph

Graph Data Structures: Motivation

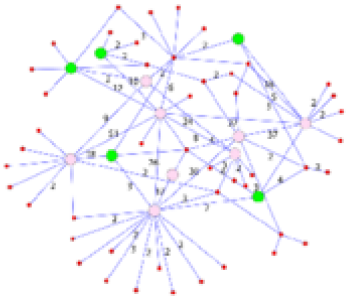
- **Increasing volumes of graph data**
- **Increasing interest in querying and reasoning** about graph data
- **Sectors**
  - Healthcare
  - Finance
  - Logistics
- **Example applications**
  - Fraud detection
  - Recommendation systems
  - Network analysis



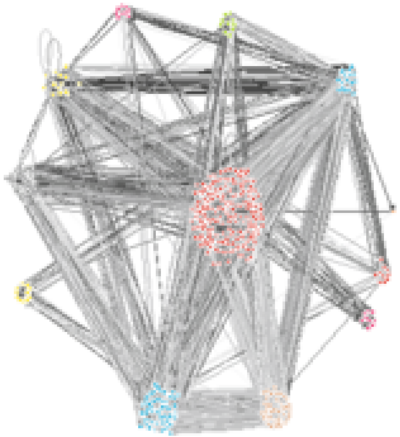
Protein-protein interaction network



Stock trading network



Supreme court citation network

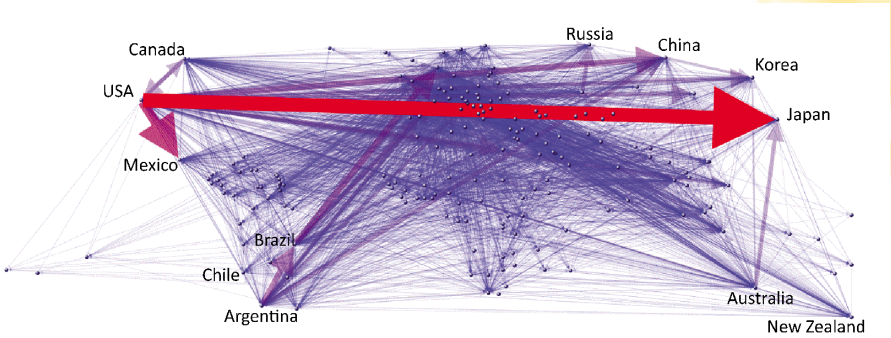


Social networks

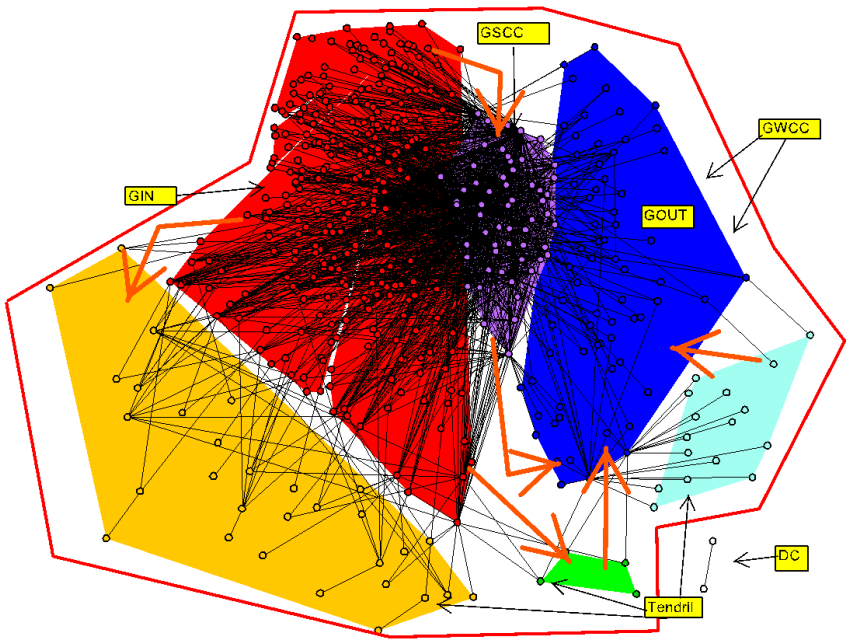
SCIENCE ACADEMY

Graph Data Structures: Motivation

- **Traditional tools** (e.g., relational DBs, NoSQL DBs) struggle with:
  - Storing and querying graph data
    - Dedicated solutions: Neo4j
  - Processing graph-structured queries
    - Dedicated solutions: Google Pregel, Apache Giraph, Spark GraphX



Global virtual trade network



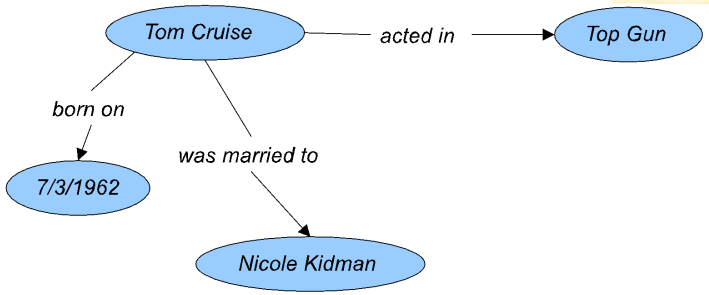
Federal funds networks

Knowledge Graphs

- **Representation of knowledge in the form of graphs**
  - Capture entities, relationships, properties
  - Provide structured view of real-world information
- Represent using RDF or Property Graph models
  - E.g., Google Knowledge Graph, DBpedia, Wikidata
- **Applications**
  - Enable machine understanding of complex domains
  - Support semantic search, recommendation, analytics
  - Used in industries for data integration, knowledge discovery, AI applications
- **Ontologies**
  - Provide formal representation of knowledge
  - Promote interoperability across knowledge bases

Graph Data Models: RDF

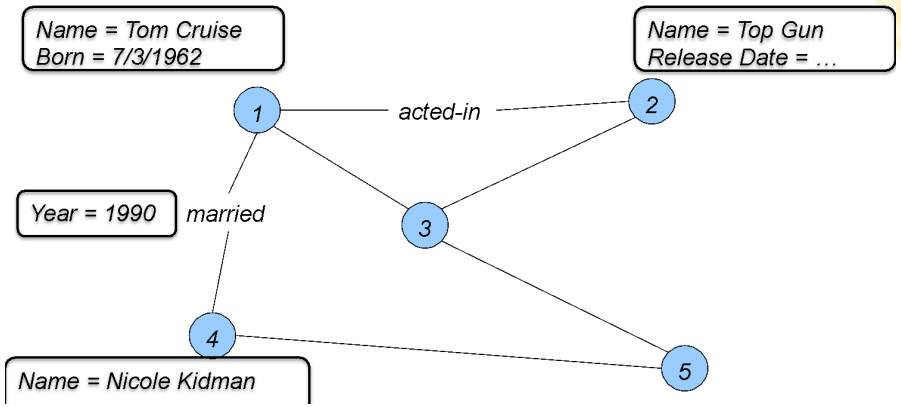
- **Resource Description Framework**
  - RDF uses triples: subject-predicate-object
  - Connects "subject" and "object" through "predicate"
  - E.g., "TomCruise-acted-TopGun"
- **Used to represent knowledge bases**
  - Queried through SPARQL



- **Pros**
  - Standardization
    - W3C standard to model data
    - Subject and object can be URIs in semantic web
  - Interoperability
    - Merge RDF data stores
  - Extensibility
    - Add new nodes and relationships
    - Support ontologies

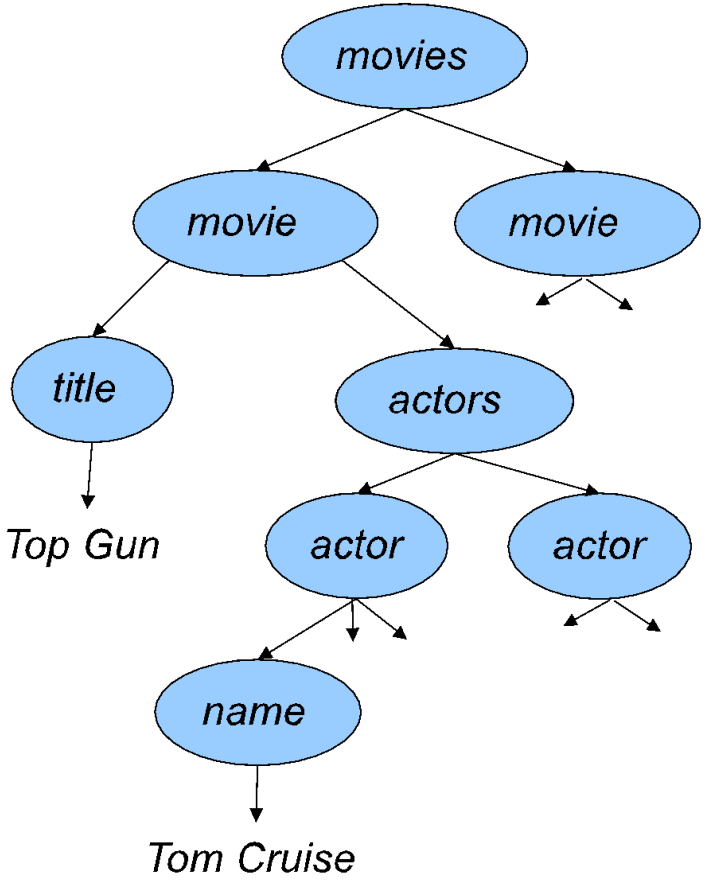
Graph Data Models: Property Graph

- Directed graph with nodes and edges having key-values *properties*
  - Similar expressive power to RDFs
- **No universal standard**
  - Less "schema"
  - Harder to interoperate
- **Examples of query languages**:
  - Cypher for Neo4j
  - Gremlin for Apache TinkerPop



Graph Data Models: XML

- Common data model for flexible data representation
- Directed labeled tree
- Popular for non-tabular data exchange




```
<movies>
  <movie>
  <title>Top Gun</title>
  <actors>
    <actor>
     <name>Tom Cruise</name>
     <born>7/3/1962</born>
    </actor>
   <actor>
     ...
   </actor>
  </actors>
  </movie>
```

Storing Graph Data

- **File systems**
  - Very simple
  - No support for transactions, ACID compliance
  - Minimal functionality (e.g., must build the analysis/querying on top)
- **Relational databases**
  - Mature technology
  - All the good stuff (SQL, transactions, ACID compliance, toolchains)
  - Minimal functionality for graph data
- **NoSQL key-value stores**
  - Can handle very large datasets efficiently in a distributed fashion
  - Minimal native functionality for graph data
- **Graph databases**
  - Efficiently support complex queries/tasks (e.g., graph traversals)
  - Not as mature as RDBMSs
  - Often lack declarative language similar to SQL
    - You may need to write custom programs

Graph Databases

- Many specialized graph database systems
  - E.g., Neo4j, Titan, OrientDB, AllegroGraph
- Key distinctions from relational databases
  - Manage and query graph-structured data
  - Store graph structure explicitly with pointers
    - Avoid joins, simplify graph traversals
    - Natural to write *queries* and *graph algorithms* (reachability, shortest paths)
  - Support graph query languages: SPARQL, Cypher, Gremlin
  - Rudimentary declarative interfaces
  - Applications often require programmatic interfaces
  - Provide programmatic API for arbitrary graph algorithms