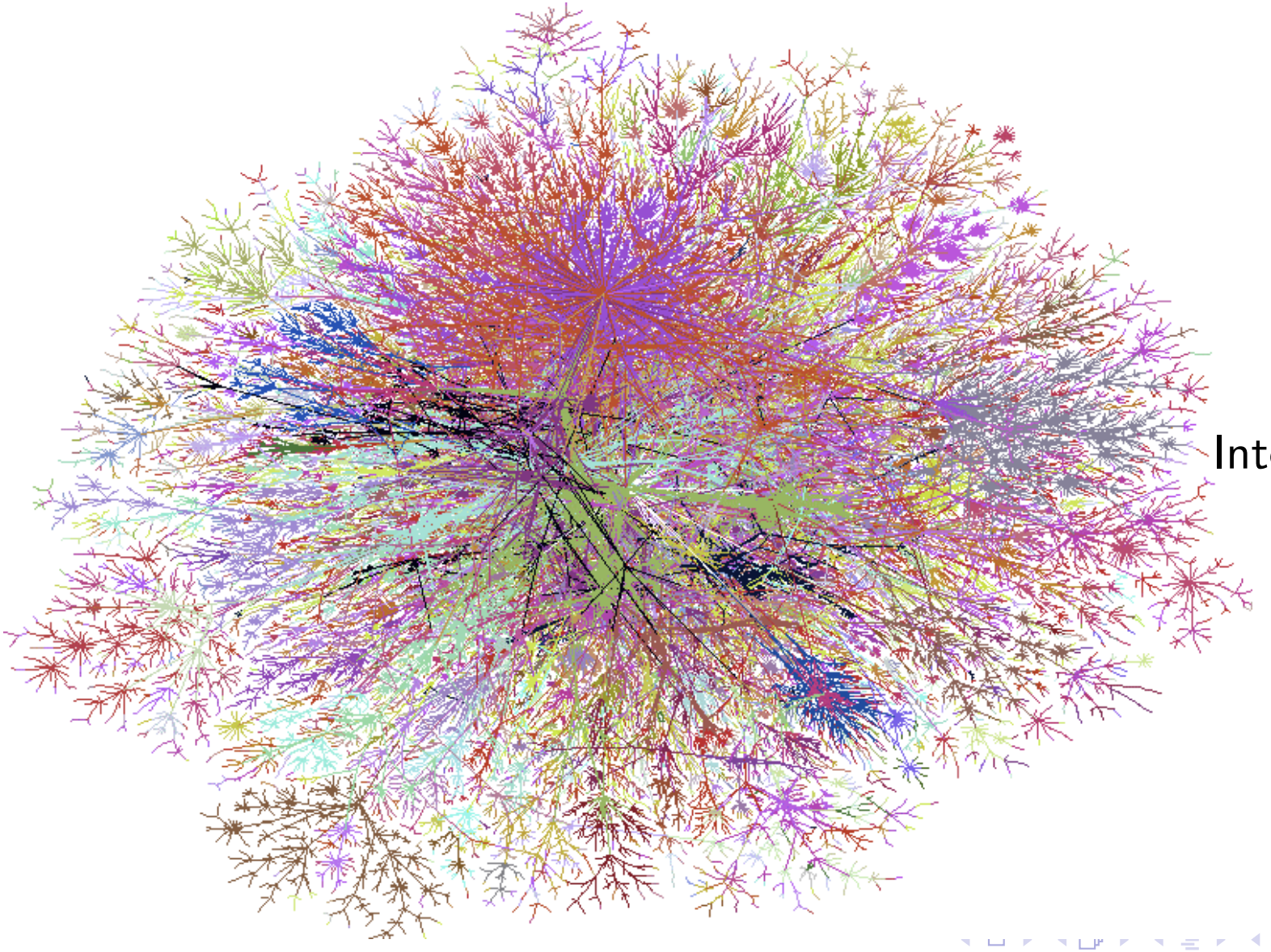
Graph Data Processing

M. Tamer Özsu

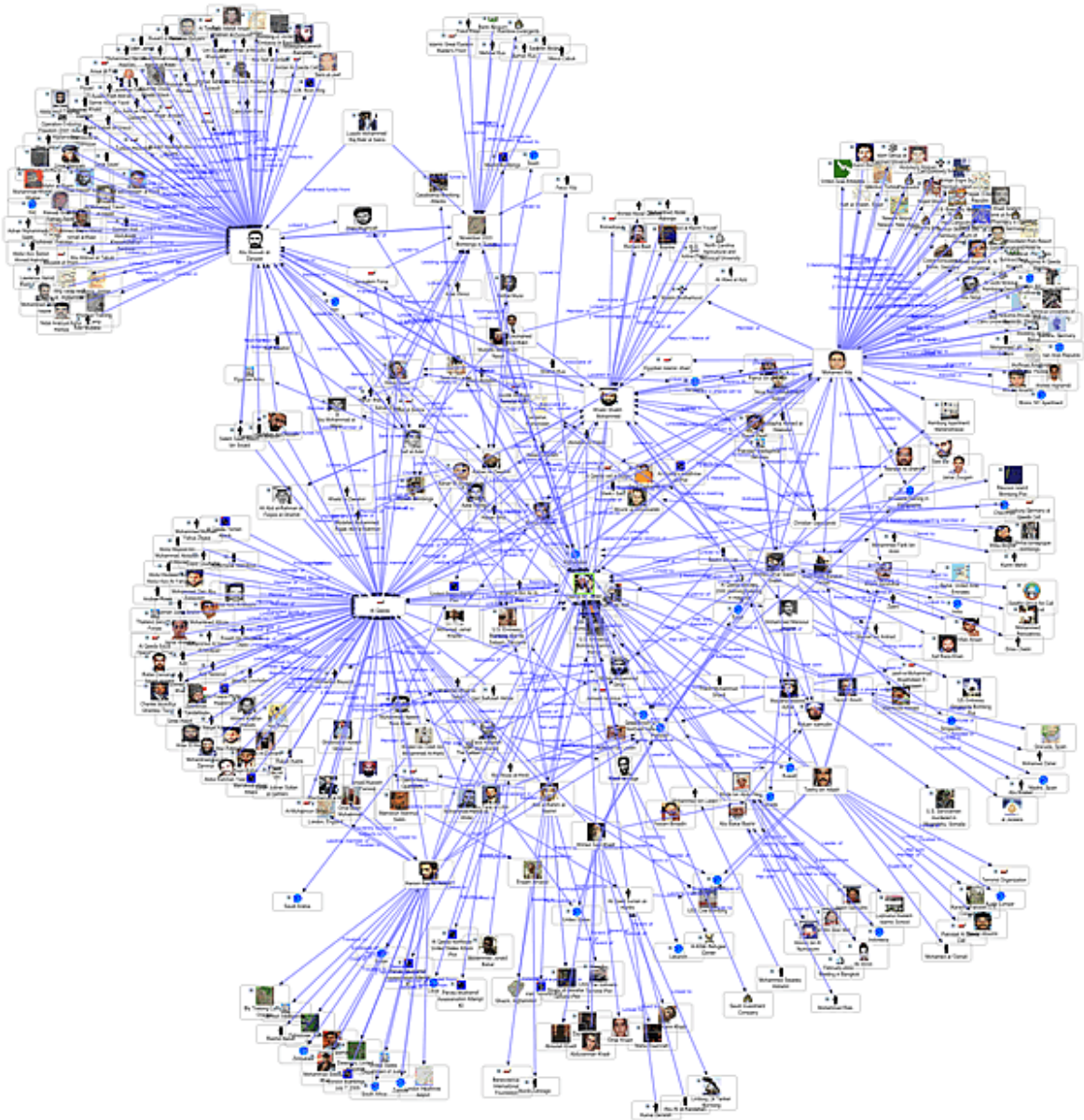
Outline

Graph Data are Very Common



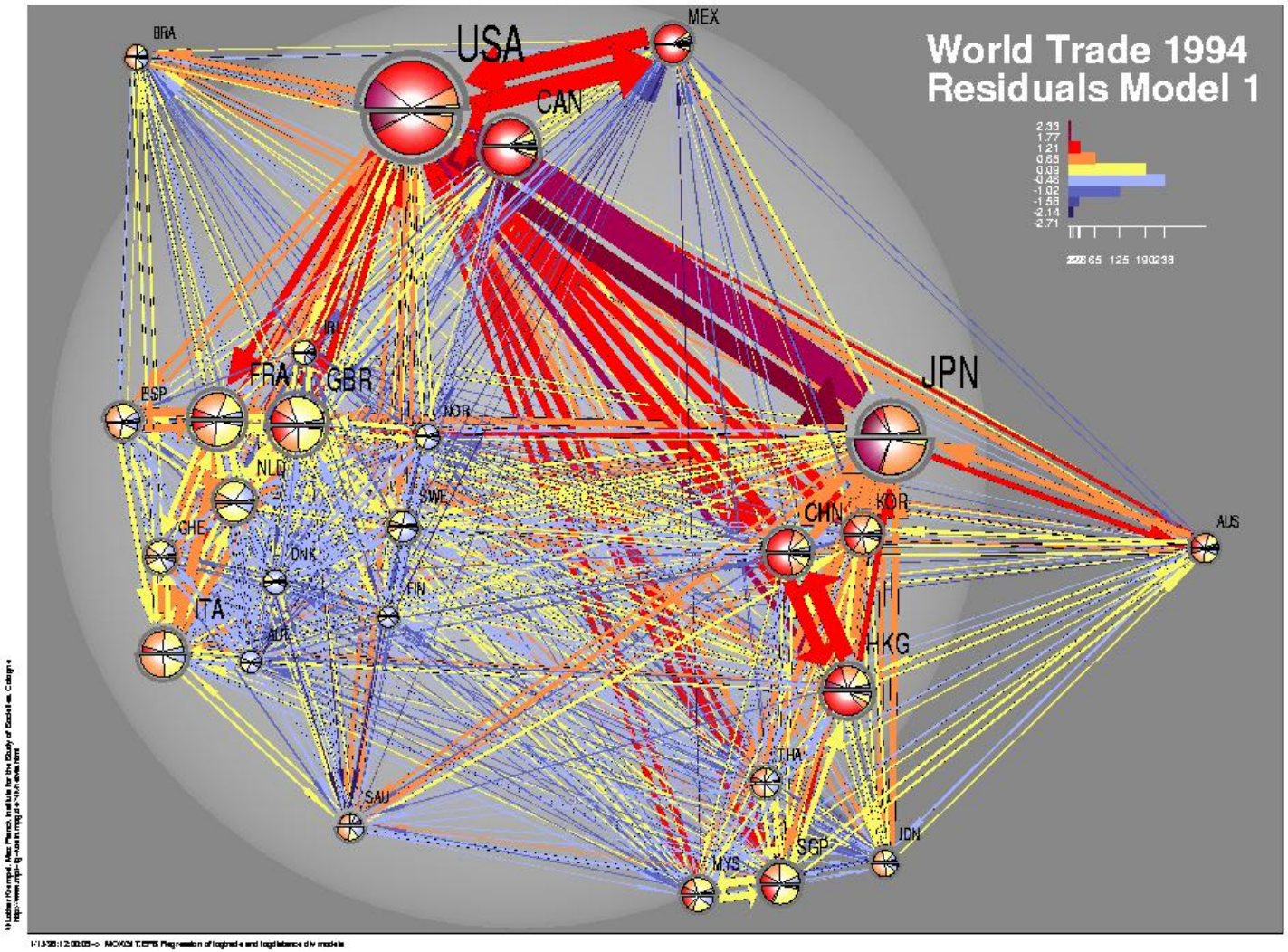
Internet

Graph Data are Very Common



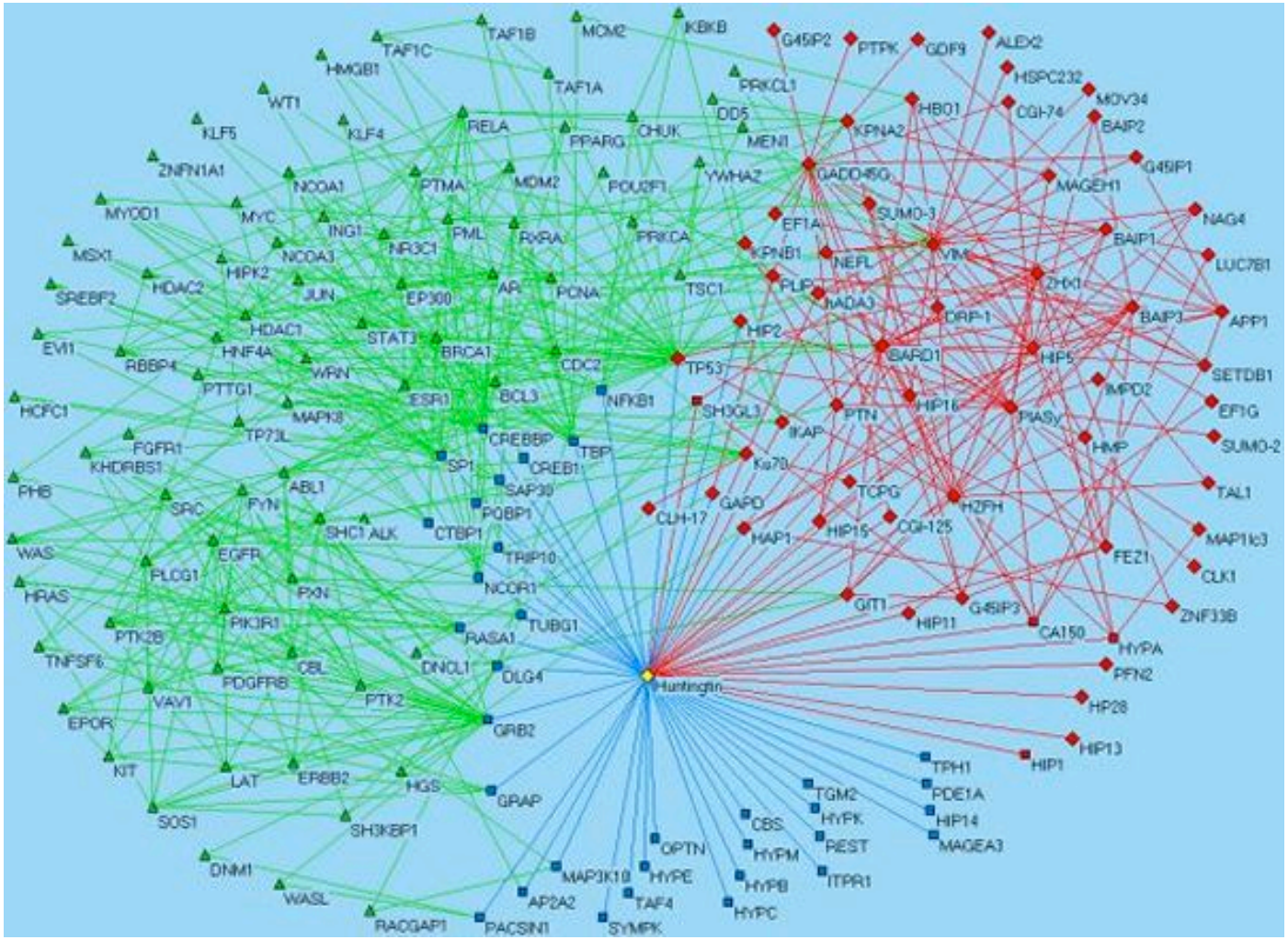
Social networks

Graph Data are Very Common



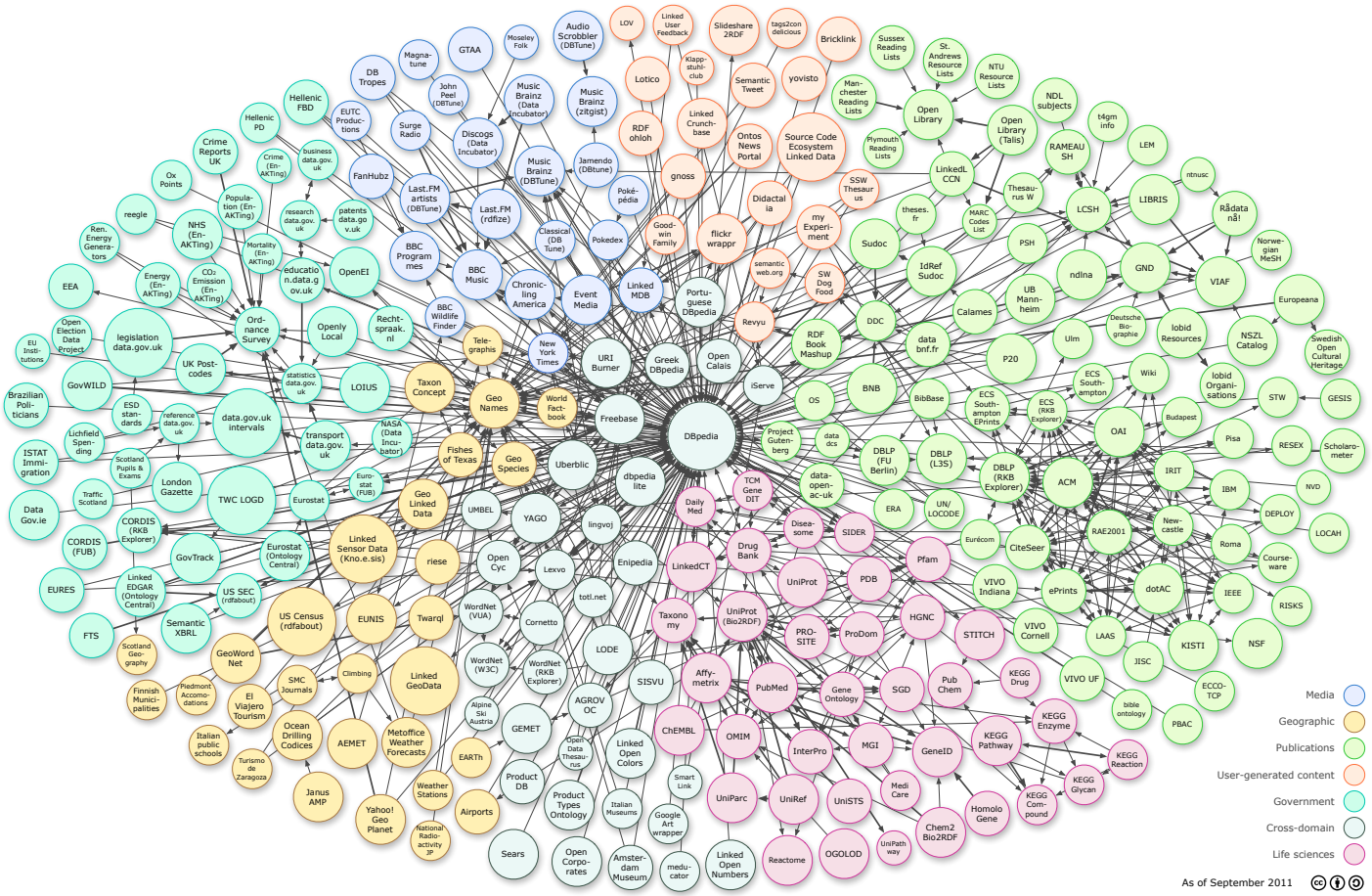
Trade volumes and connections

Graph Data are Very Common



Biological networks

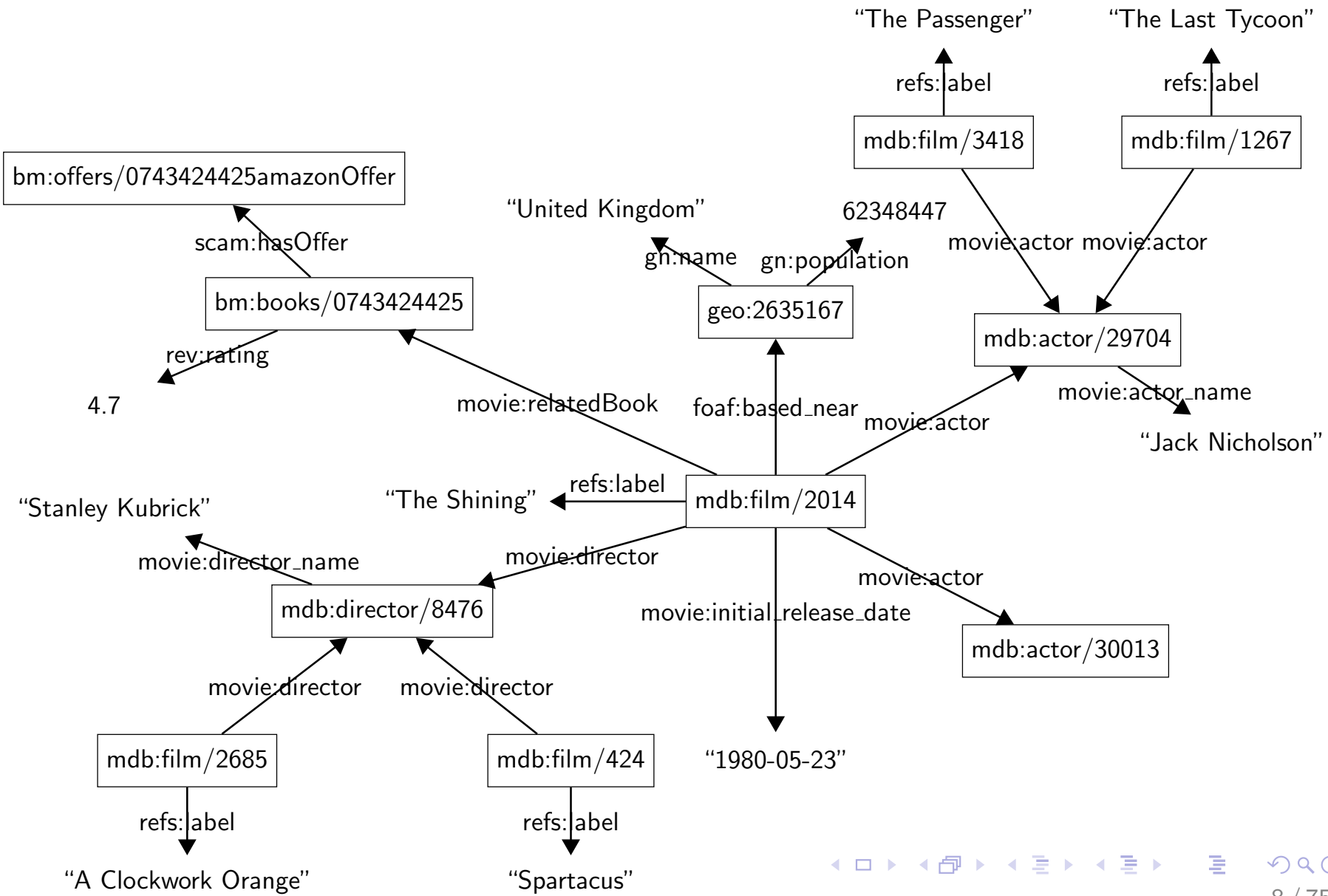
Graph Data are Very Common



Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/

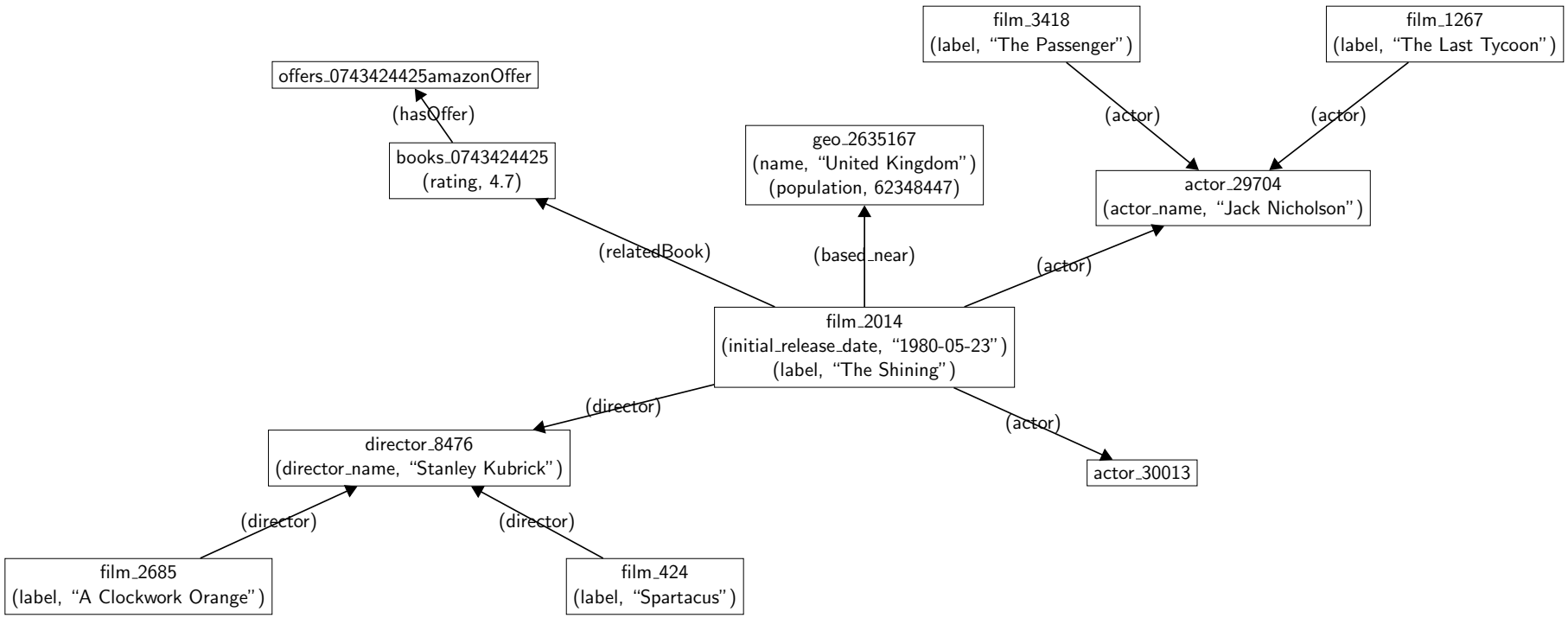
Graph Types

RDF graph



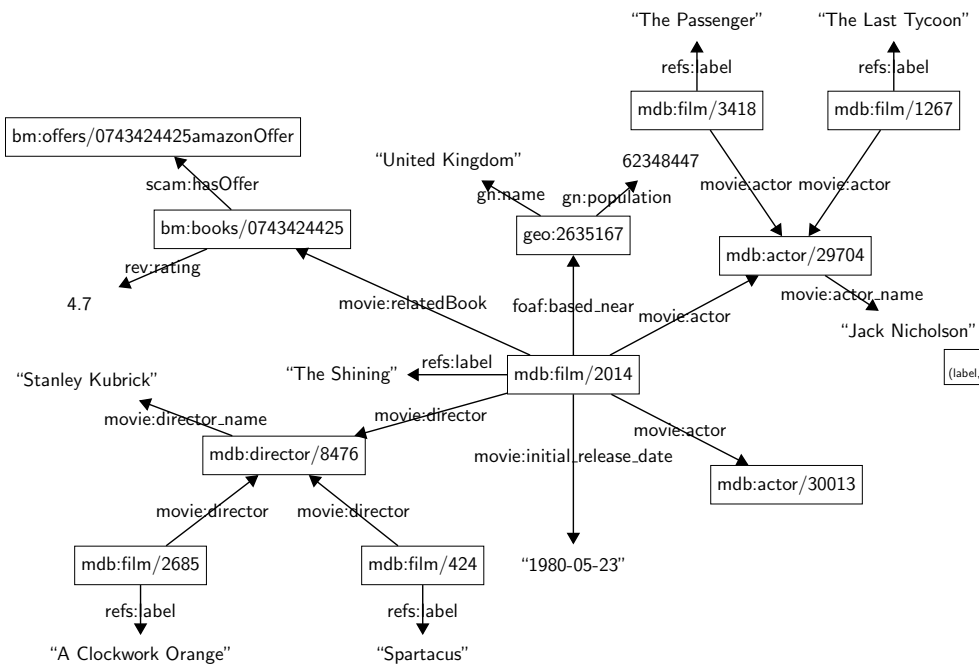
Graph Types

Property graph

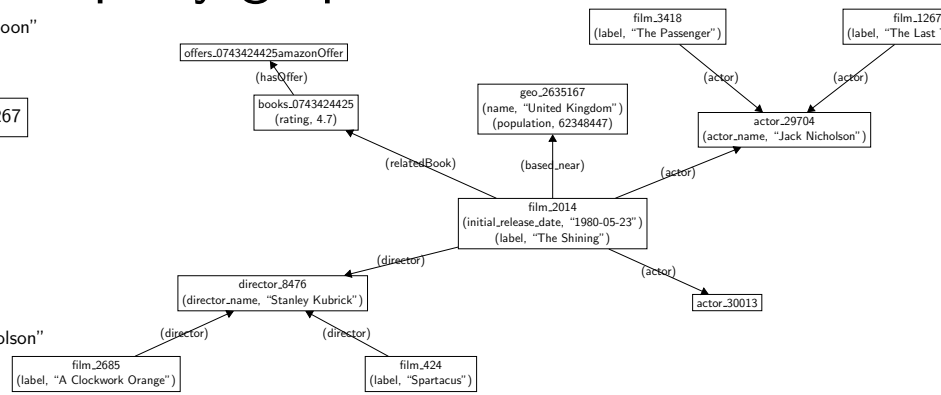


Graph Types

RDF graph



Property graph

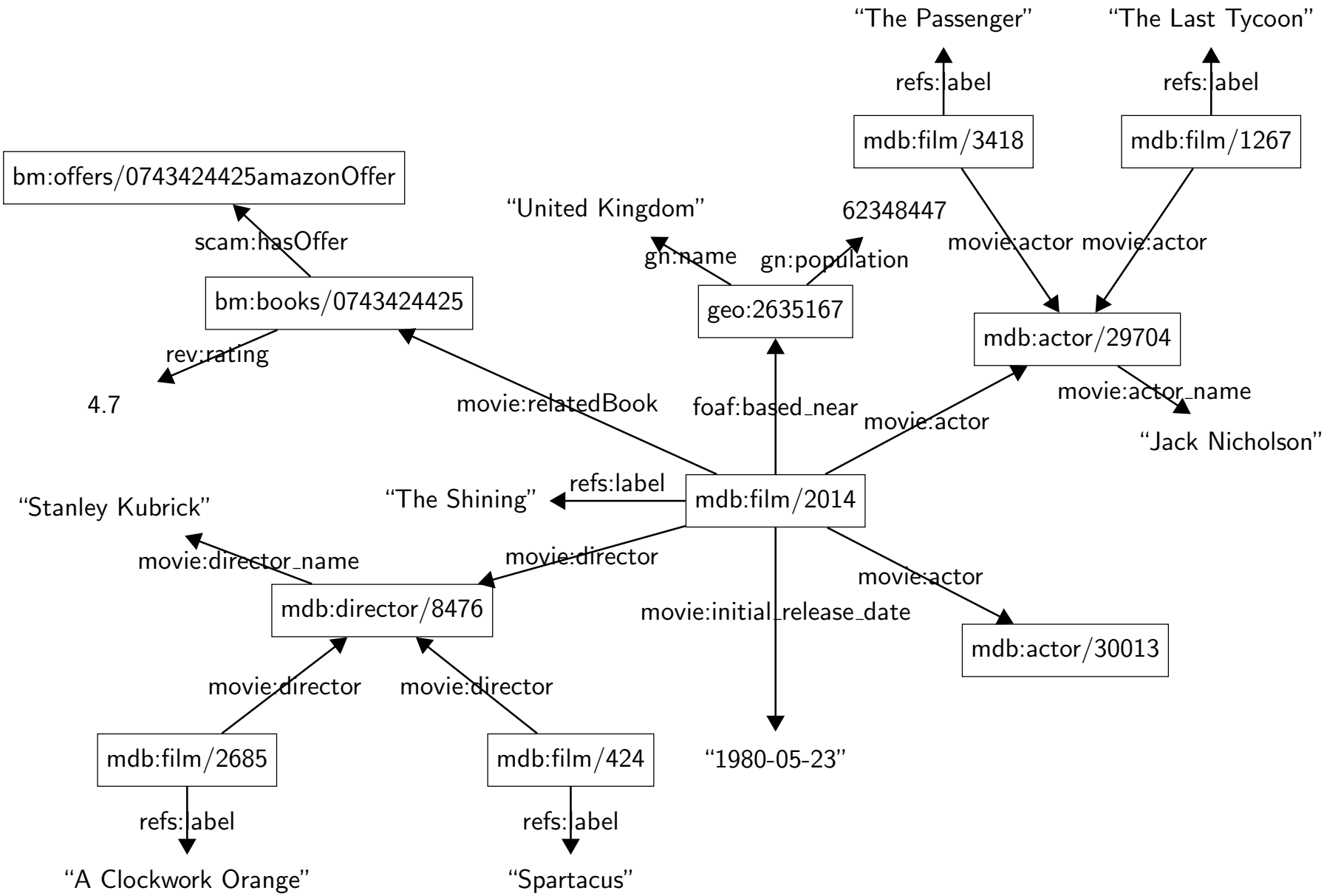


- Workload: SPARQL queries
- Query execution: subgraph matching by homomorphism

- Workload: Online queries and analytic workloads
- Query execution: Much more varied

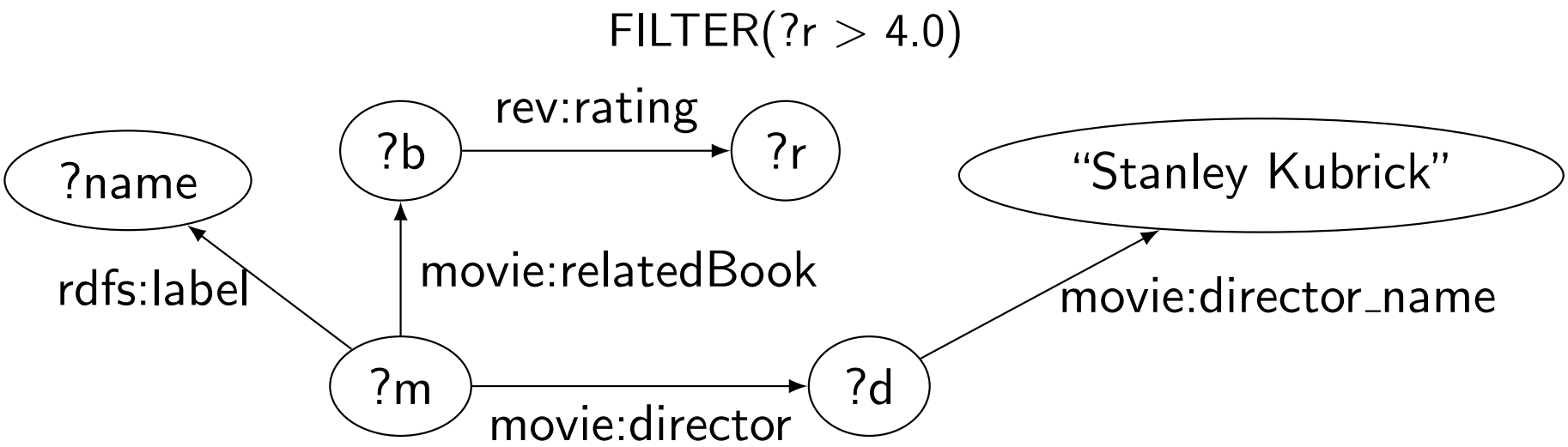
Outline

RDF Graph


"The Passenger"
"The Last Tycoon"
refs:label
refs:label
mdb:film/3418
mdb:film/1267
bm:offers/0743424425amazonOffer
"United Kingdom"
62348447
scam:hasOffer
gn:name
gn:population
movie:actor
movie:actor
bm:books/0743424425
geo:2635167
mdb:actor/29704
rev:rating
4.7
movie:relatedBook
foaf:based_near
movie:actor
movie:actor_name
"Jack Nicholson"
"Stanley Kubrick"
"The Shining"
refs:label
mdb:film/2014
movie:director_name
movie:director
movie:actor
mdb:director/8476
movie:initial_release_date
mdb:actor/30013
movie:director
movie:director
mdb:film/2685
mdb:film/424
"1980-05-23"
refs:label
refs:label
"A Clockwork Orange"
"Spartacus"

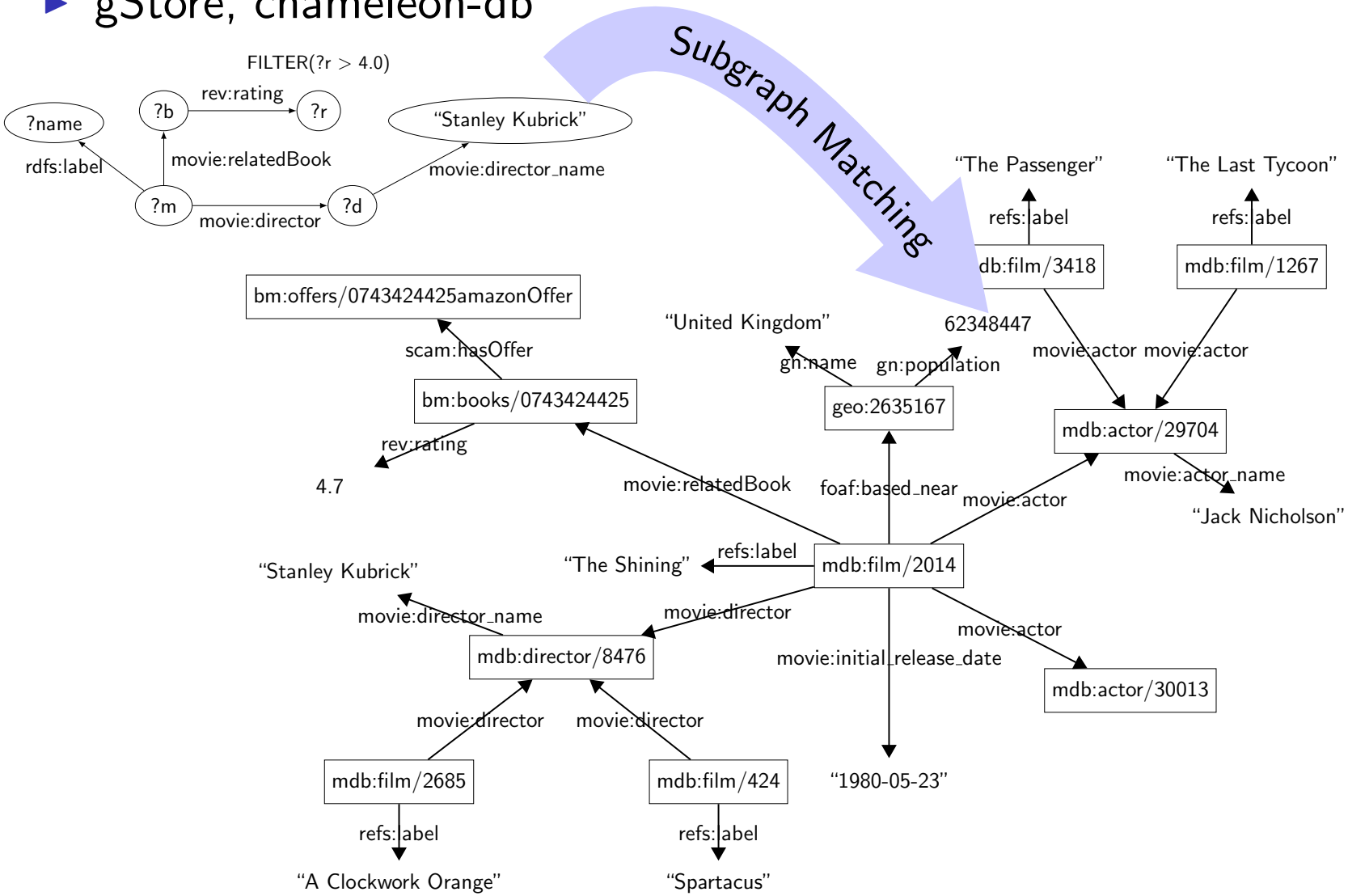
SPARQL Queries

```
SELECT ?name
WHERE {
   ?m rdfs:label ?name. ?m movie:director ?d.
   ?d movie:director_name "Stanley Kubrick".
   ?m movie:relatedBook ?b. ?b rev:rating ?r.
   FILTER(?r > 4.0)
}
```



Graph-based Approach

- Answering SPARQL query ≡ subgraph matching
- gStore, chameleon-db



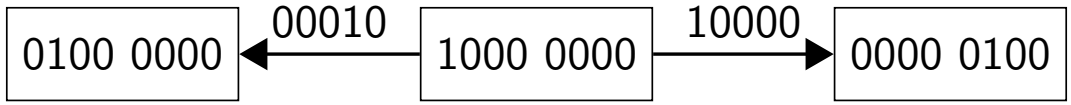
gStore

General Approach:

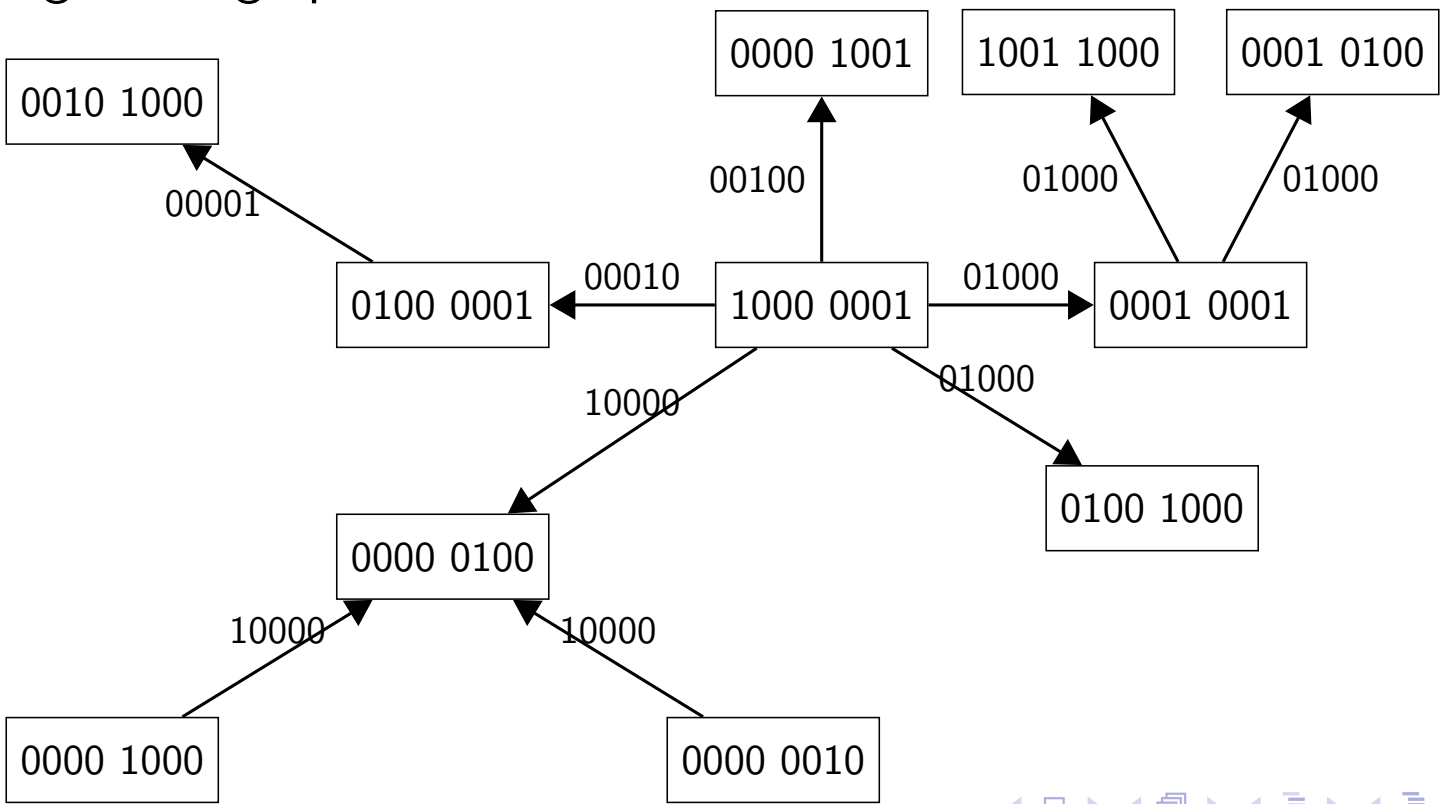
- Work directly on the RDF graph and the SPARQL query graph
- Use a signature-based encoding of each entity and class vertex to speed up matching
- Filter-and-evaluate
  - Use a false positive algorithm to prune nodes and obtain a set of candidates; then do more detailed evaluation on those
- Use an index (VS*-tree) over the data signature graph (has light maintenance load) for efficient pruning

1. Encode Q and G to Get Signature Graphs

Query signature graph Q^*

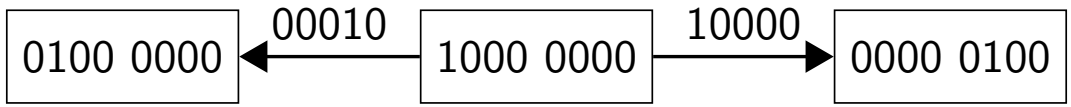


Data signature graph G^*

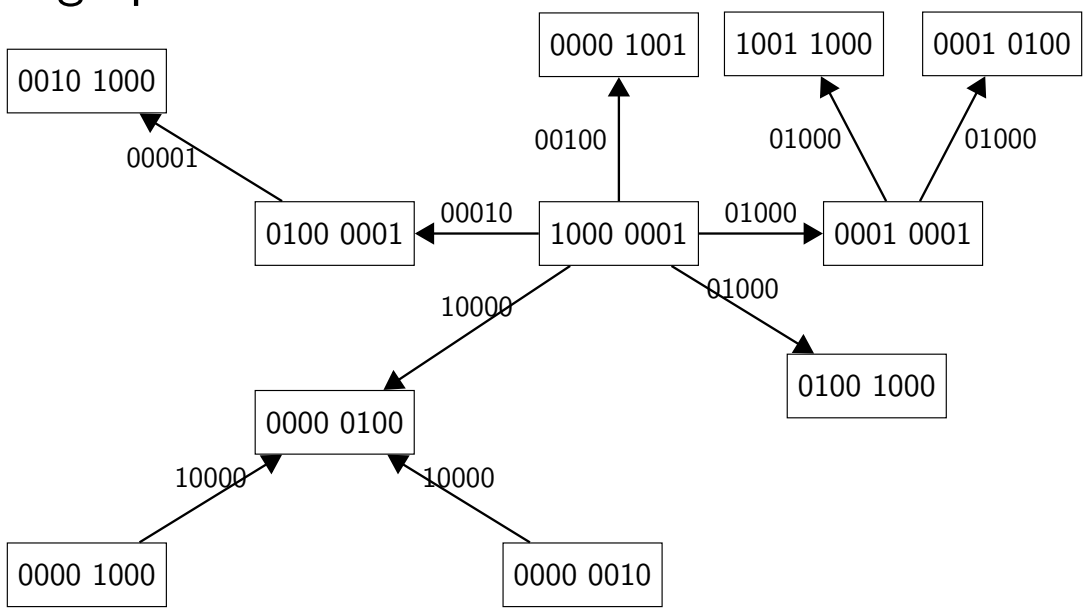


2. Filter-and-Evaluate

Query signature graph Q^*



Data signature graph G^*



Find matches of Q^* over signature graph G^* → Verify each match in RDF graph G

How to Generate Candidate List

- Two step process:
  1. For each node of Q^* get lists of nodes in G^* that *include* that node.
  2. Do a multi-way join to get the candidate list

How to Generate Candidate List

- Two step process:
  1. For each node of Q^* get lists of nodes in G^* that *include* that node.
  2. Do a multi-way join to get the candidate list
- Alternatives:

How to Generate Candidate List

- Two step process:
  1. For each node of Q^* get lists of nodes in G^* that *include* that node.
  2. Do a multi-way join to get the candidate list
- Alternatives:
  - Sequential scan of G^*
    - Both steps are inefficient

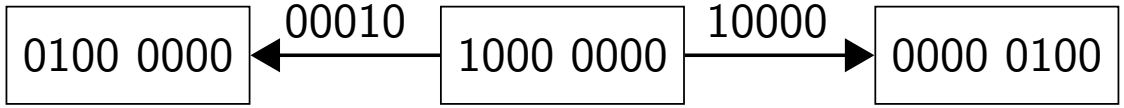
How to Generate Candidate List

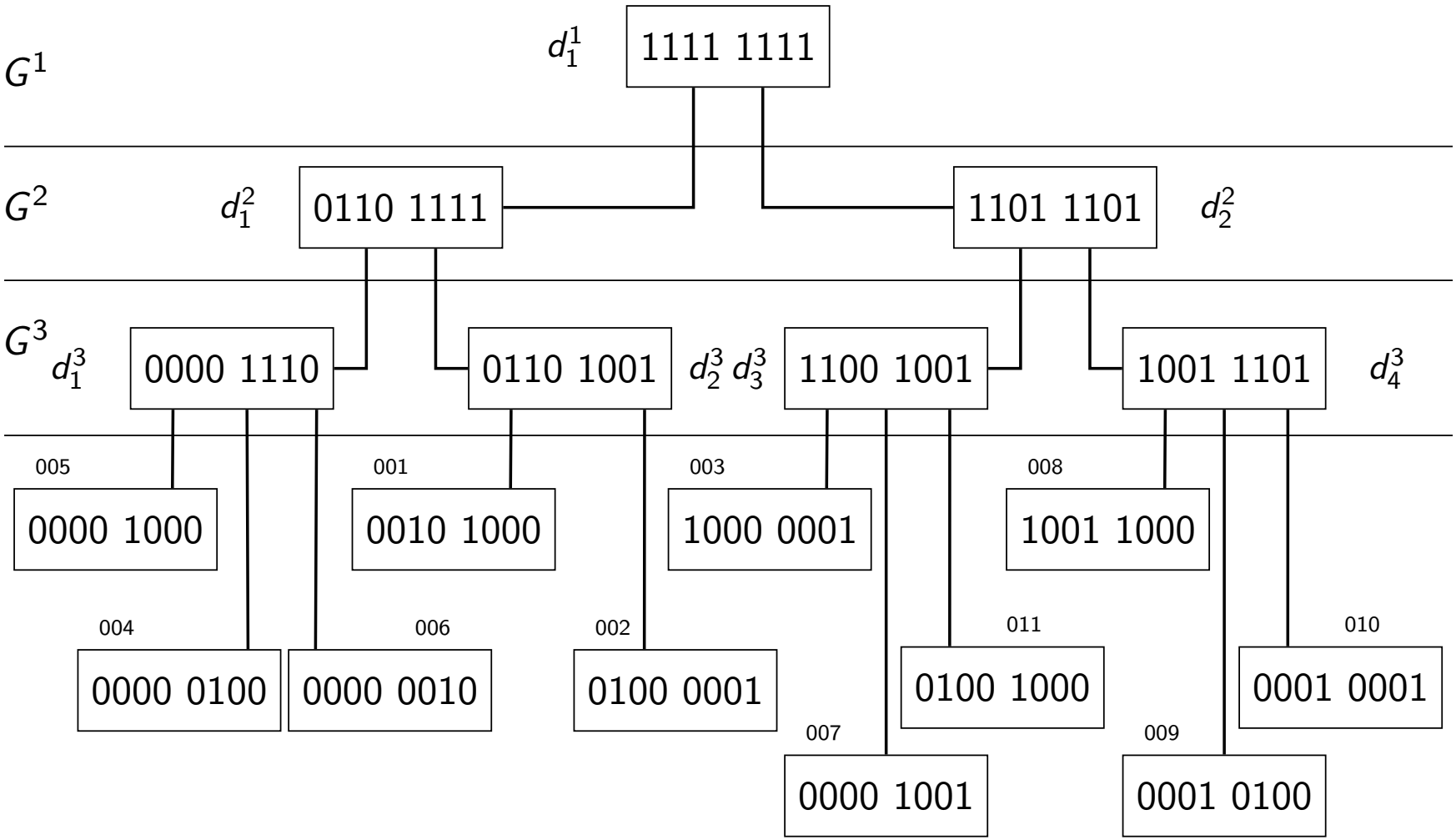
- Two step process:
  1. For each node of Q^* get lists of nodes in G^* that *include* that node.
  2. Do a multi-way join to get the candidate list
- Alternatives:
  - Sequential scan of G^*
    - Both steps are inefficient
  - Use S-trees
    - Height-balanced tree over signatures
    - Run an inclusion query for each node of Q^* and get lists of nodes in G^* that include that node.
      - Given query signature q and a set of data signatures S, find all data signatures $s_i \in S$ where $q \& s_i = q$
    - Does not support second step – expensive

How to Generate Candidate List

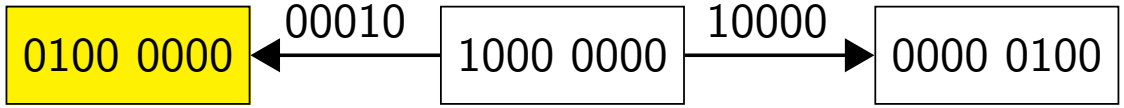
- Two step process:
  1. For each node of Q^* get lists of nodes in G^* that *include* that node.
  2. Do a multi-way join to get the candidate list
- Alternatives:
  - Sequential scan of G^*
    - Both steps are inefficient
  - Use S-trees
    - Height-balanced tree over signatures
    - Run an inclusion query for each node of Q^* and get lists of nodes in G^* that include that node.
      - Given query signature q and a set of data signatures S, find all data signatures $s_i \in S$ where $q \& s_i = q$
    - Does not support second step – expensive
  - VS-tree (and VS*-tree)
    - Multi-resolution summary graph based on S-tree
    - Supports both steps efficiently
    - Grouping by vertices

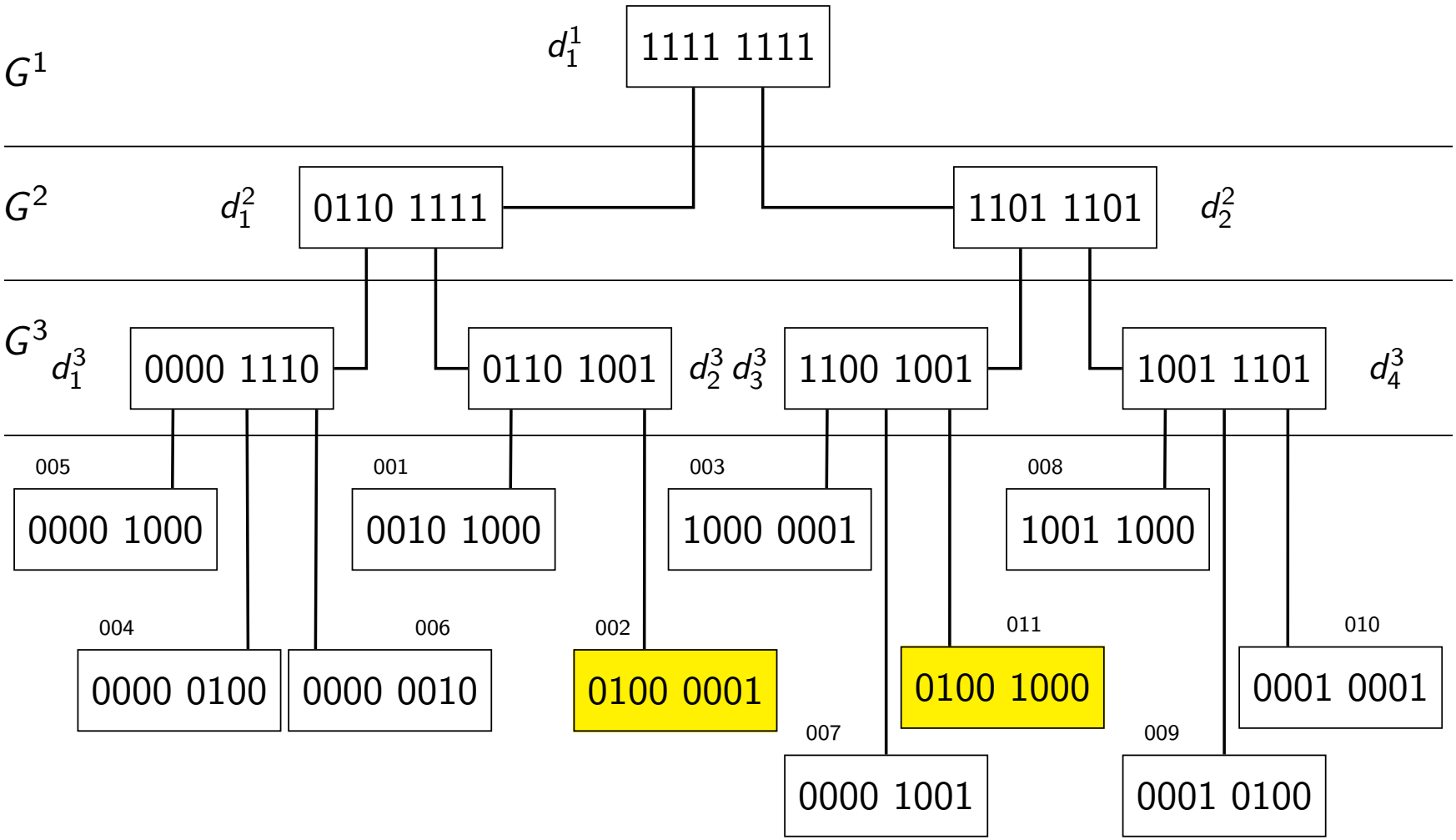
S-tree Solution



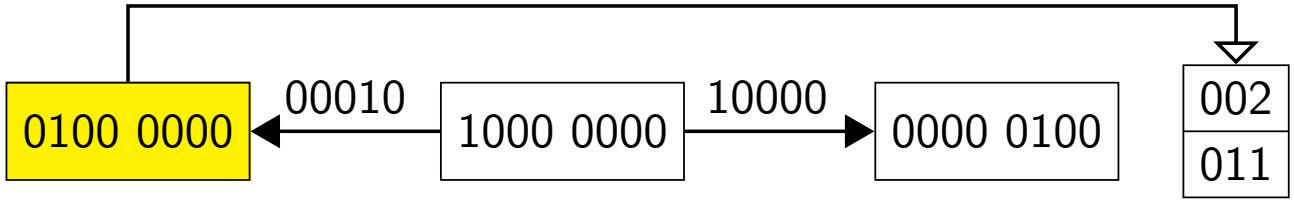


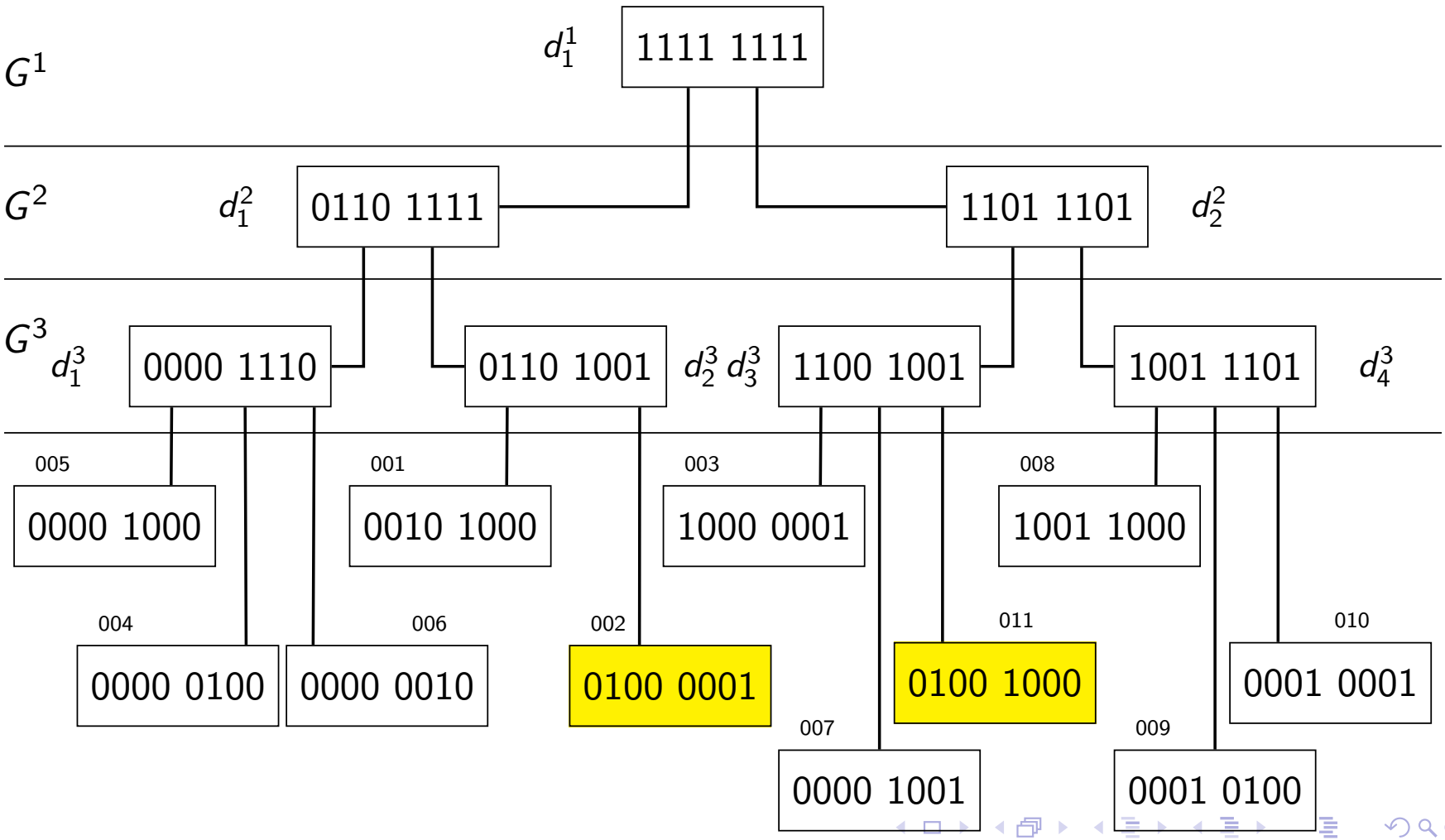
S-tree Solution



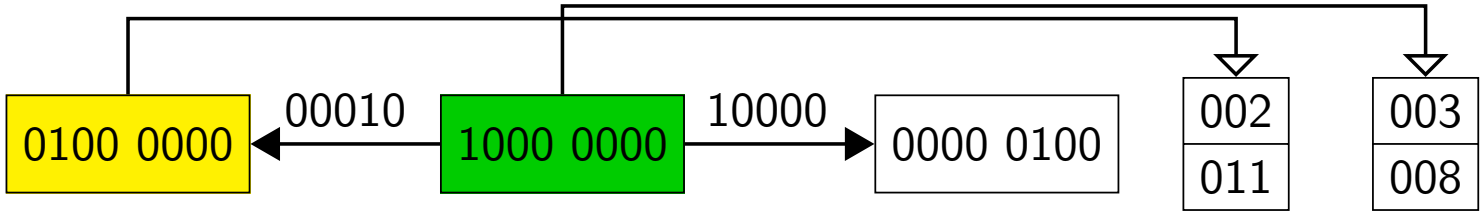


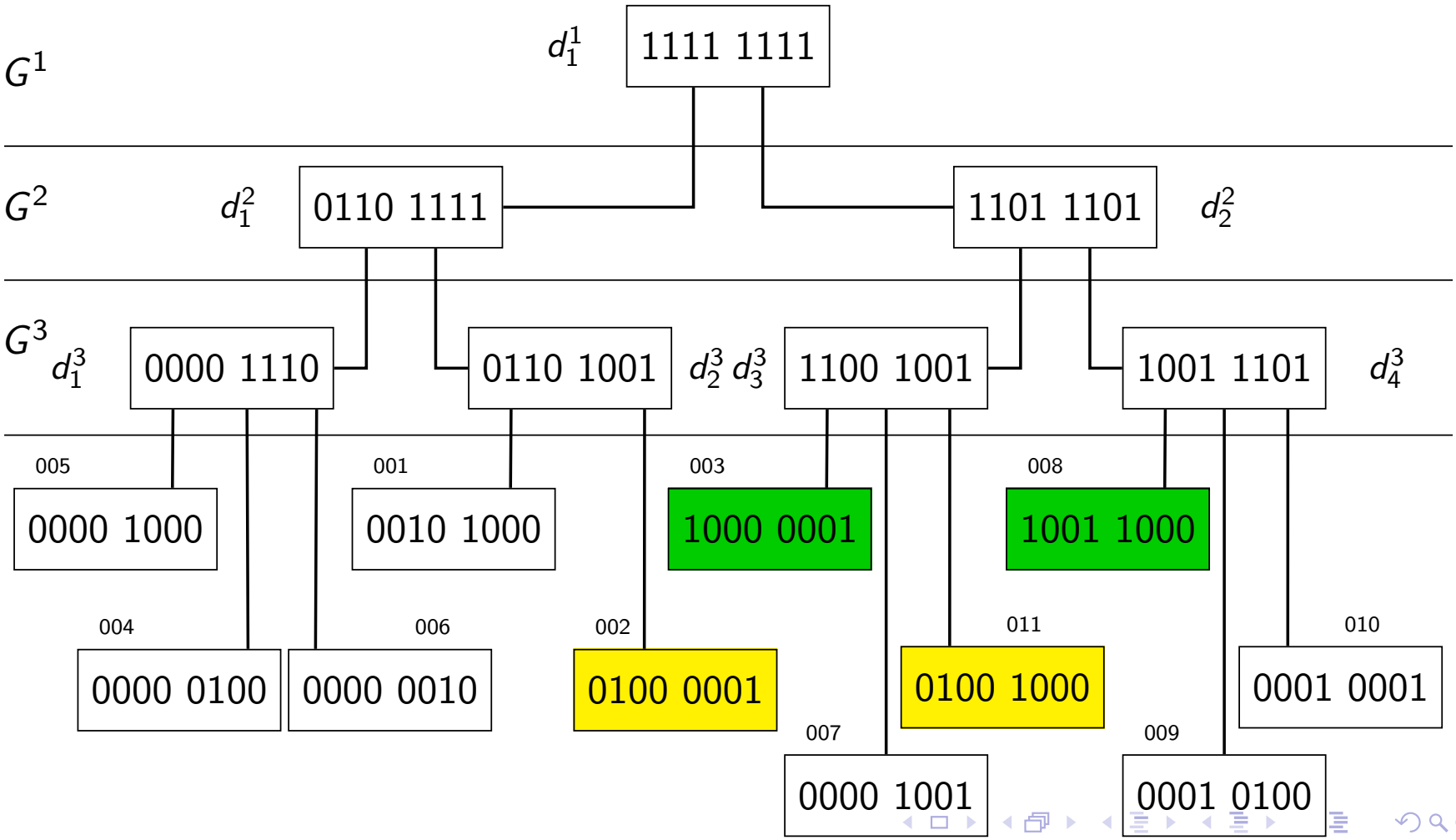
S-tree Solution



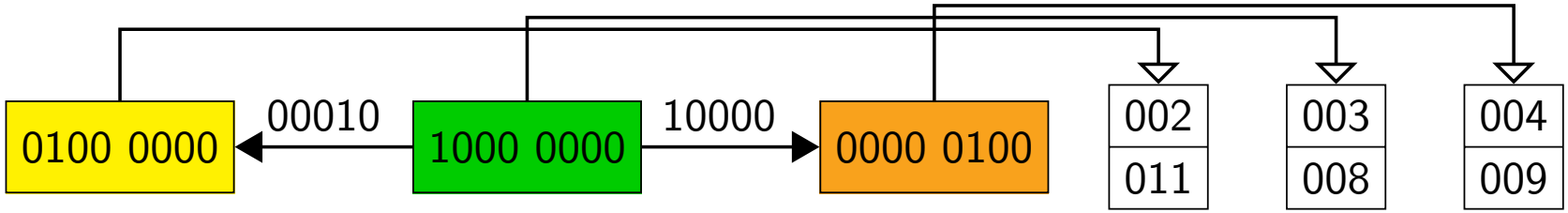


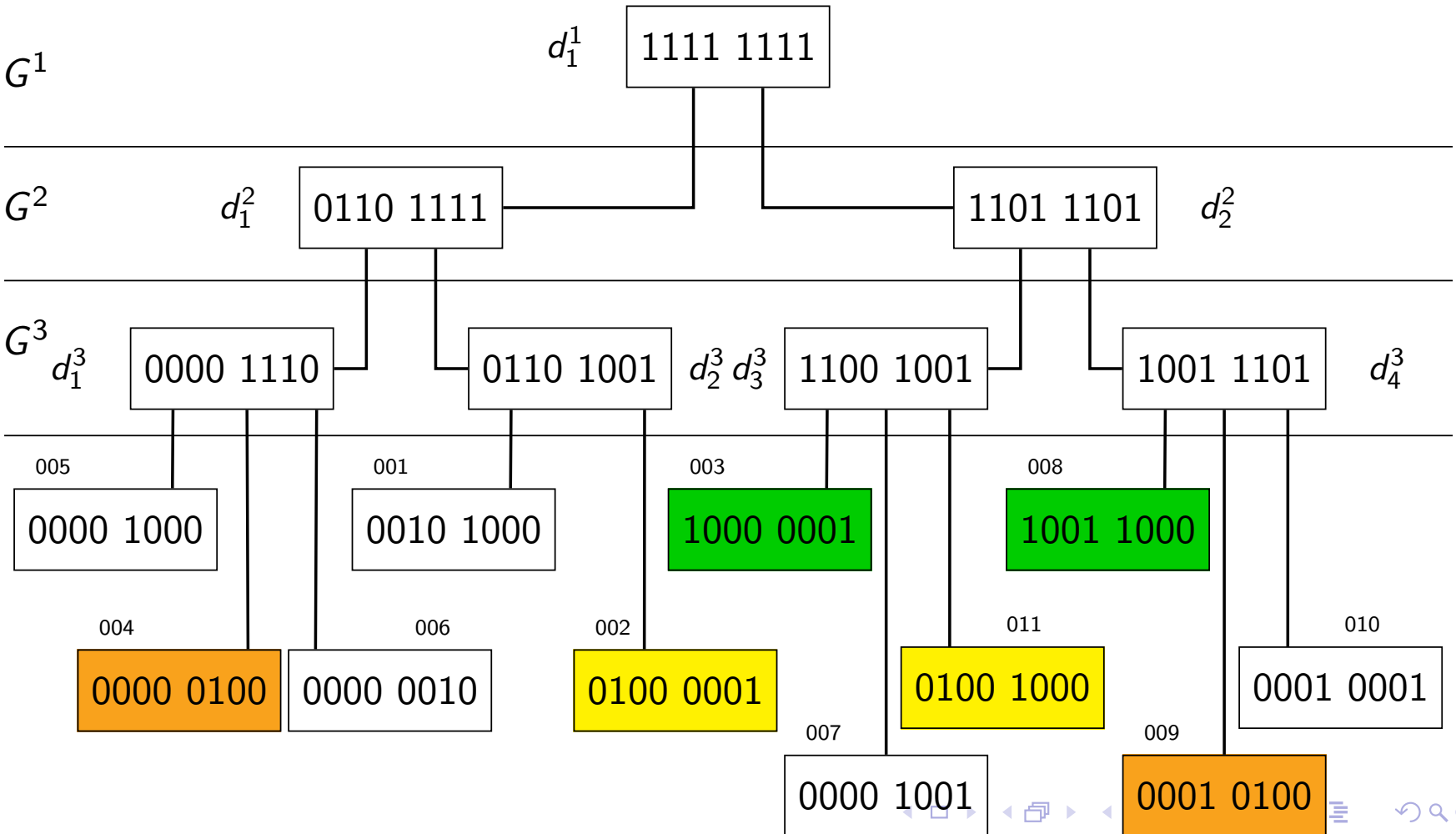
S-tree Solution



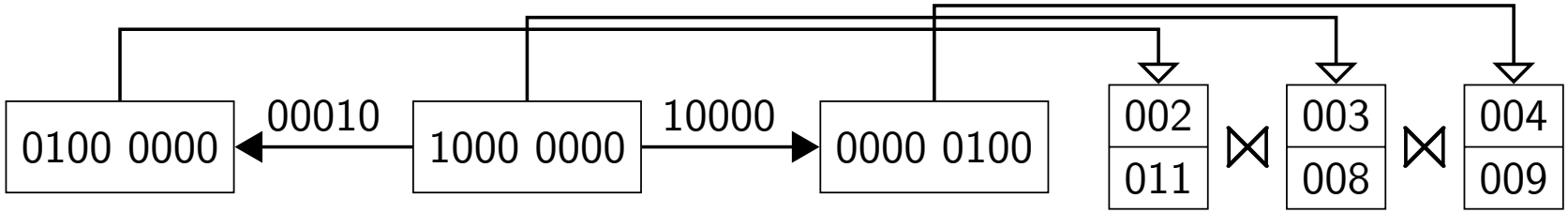


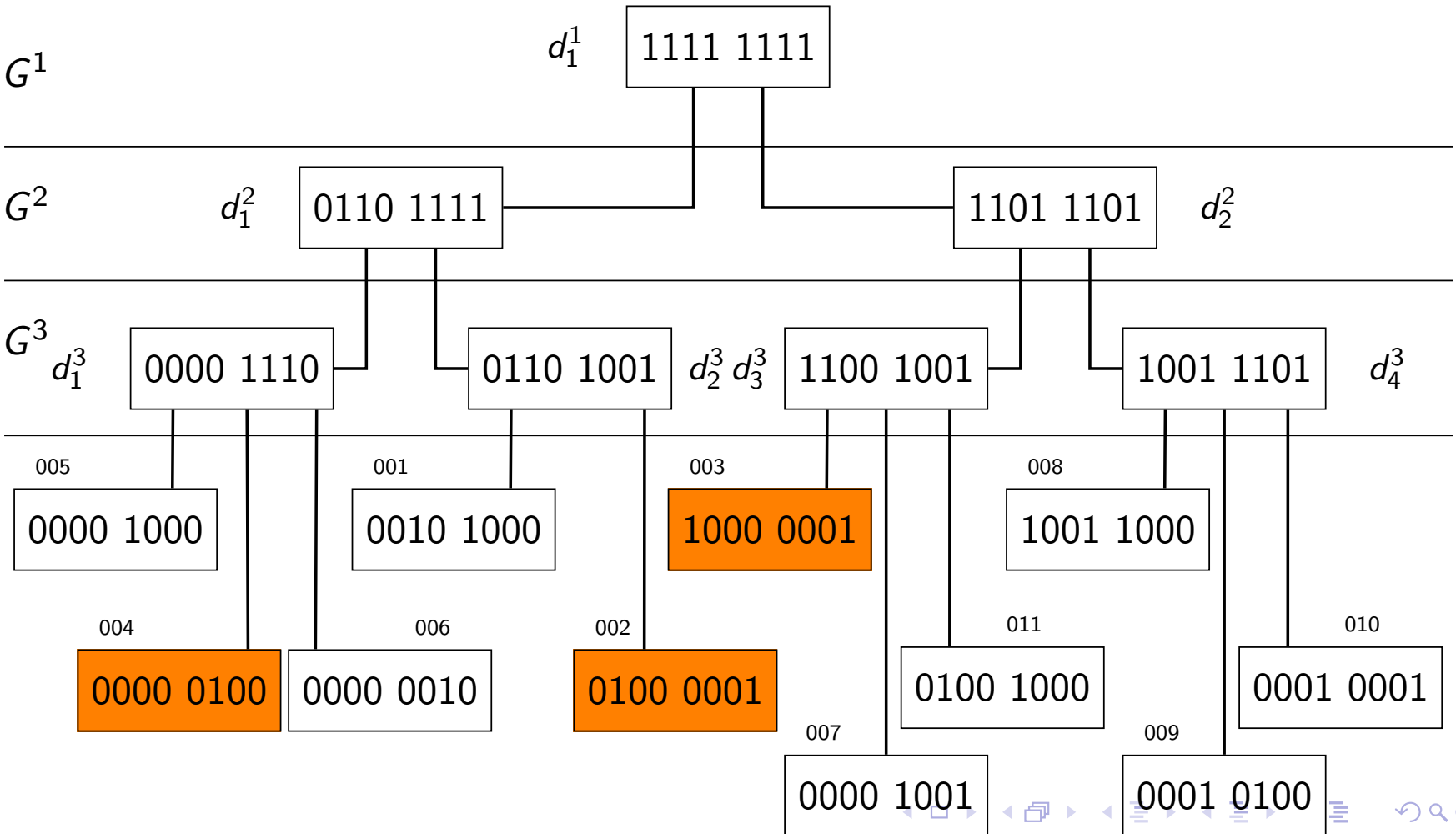
S-tree Solution



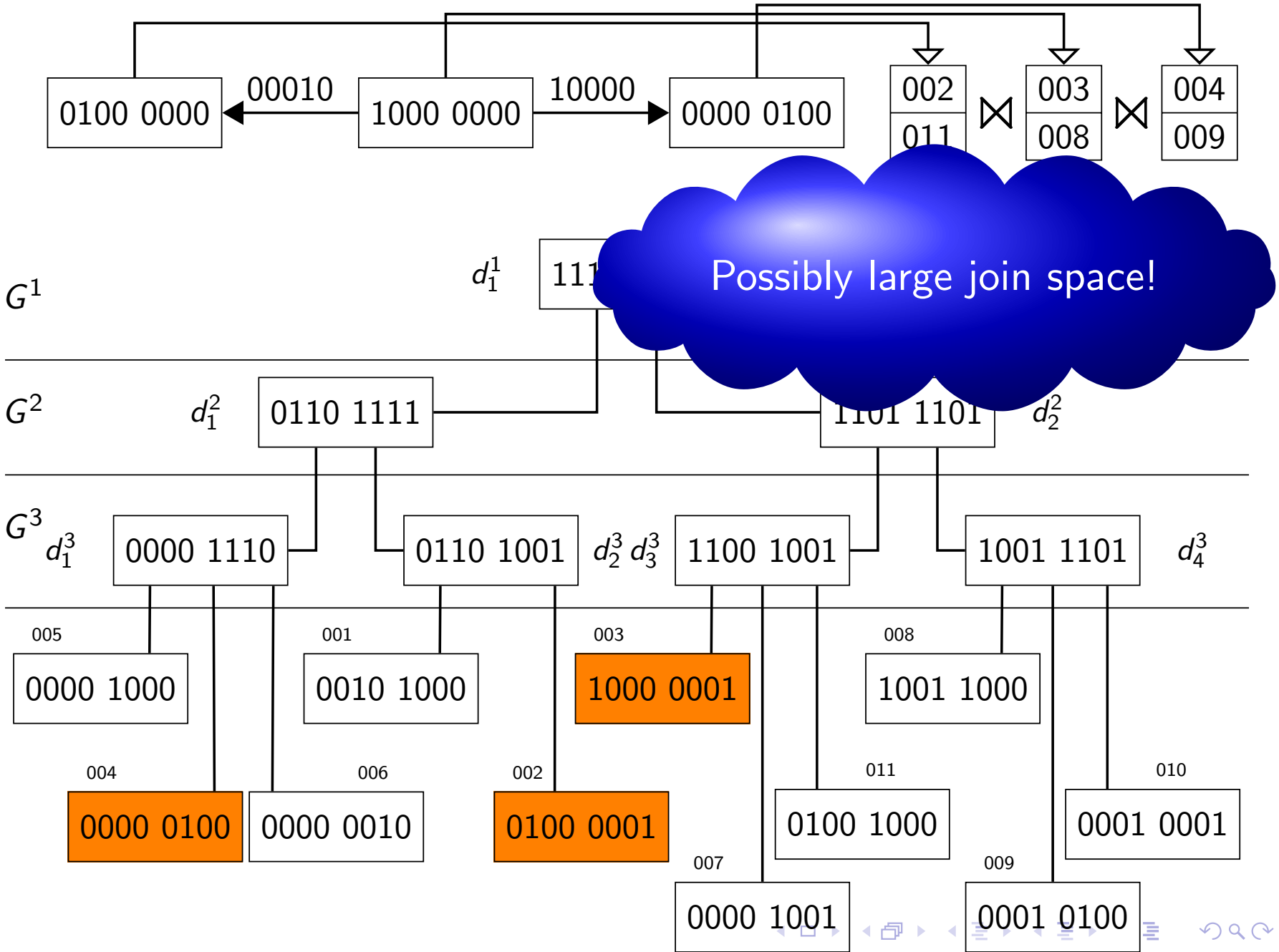


S-tree Solution



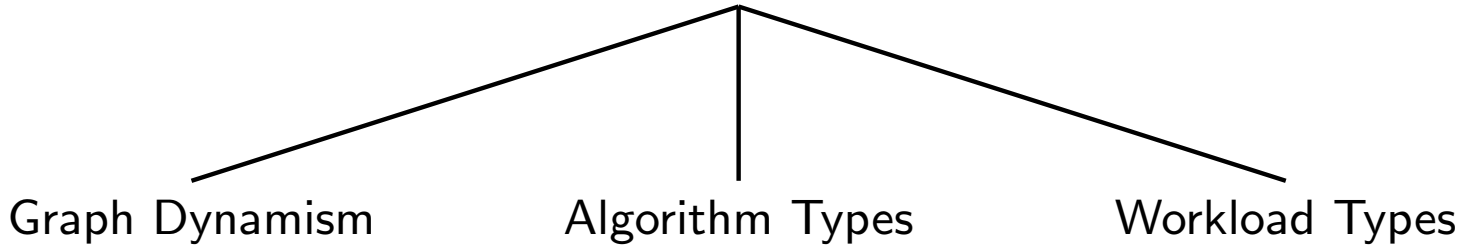


S-tree Solution

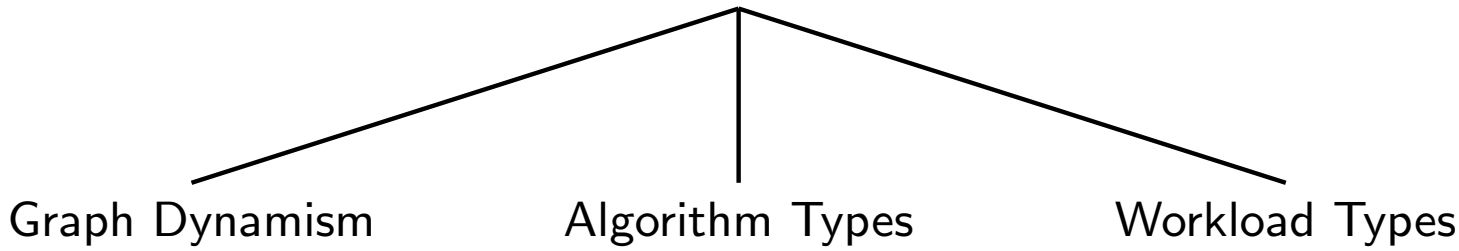


Outline

Classification

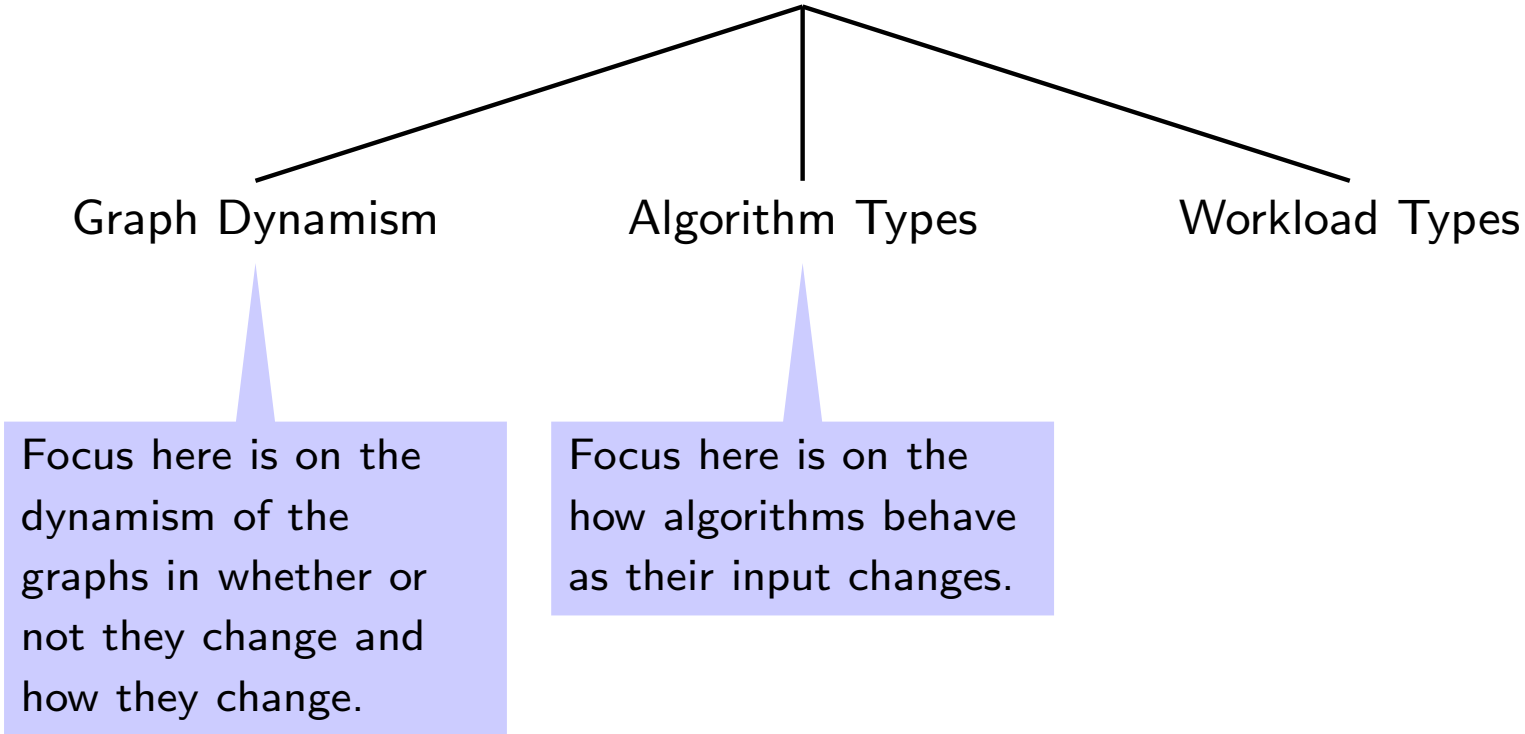


Classification

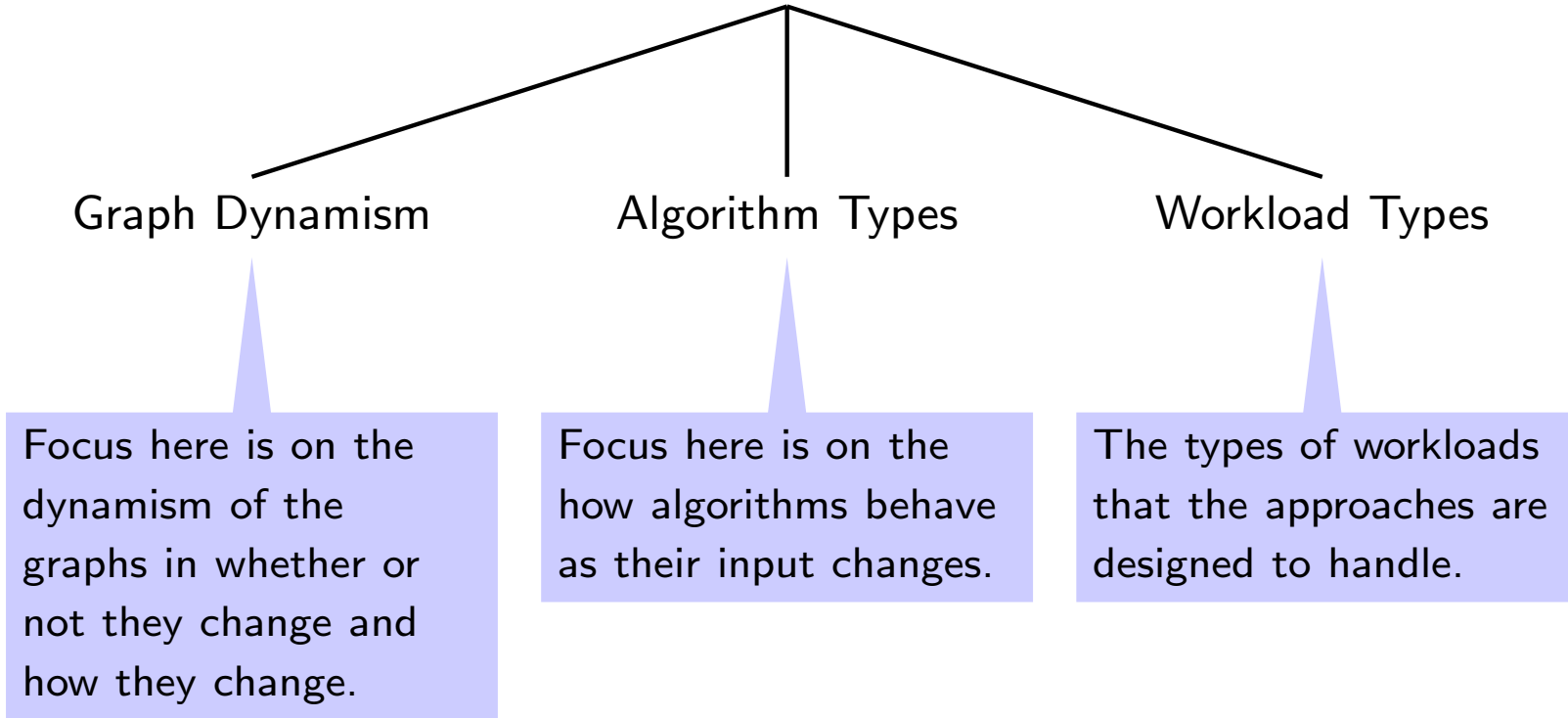


Focus here is on the dynamism of the graphs in whether or not they change and how they change.

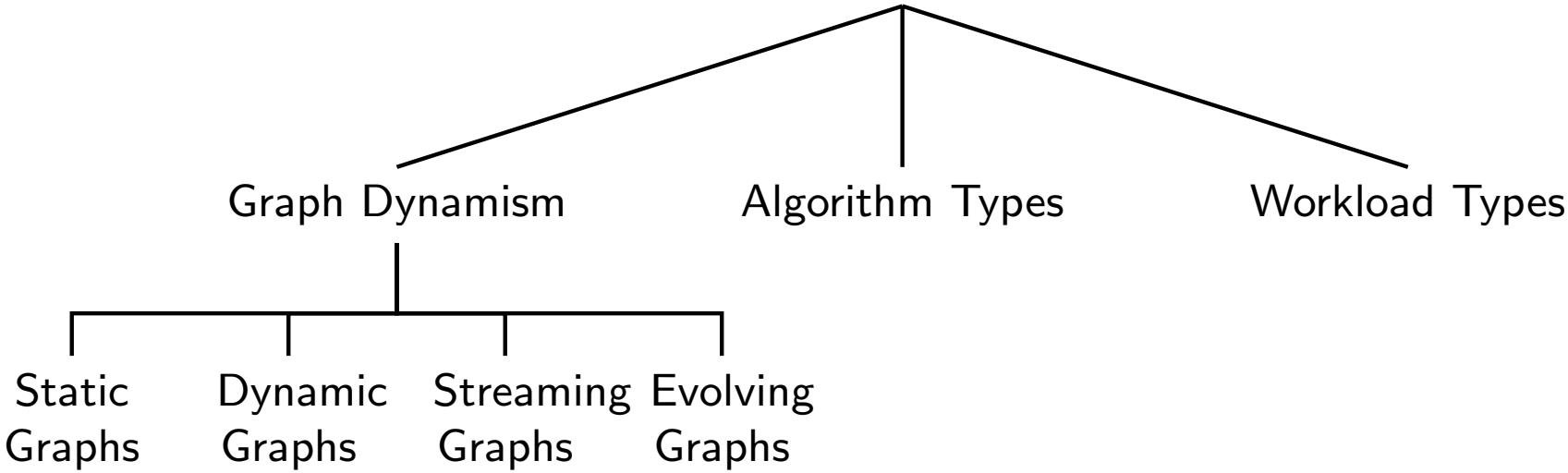
Classification



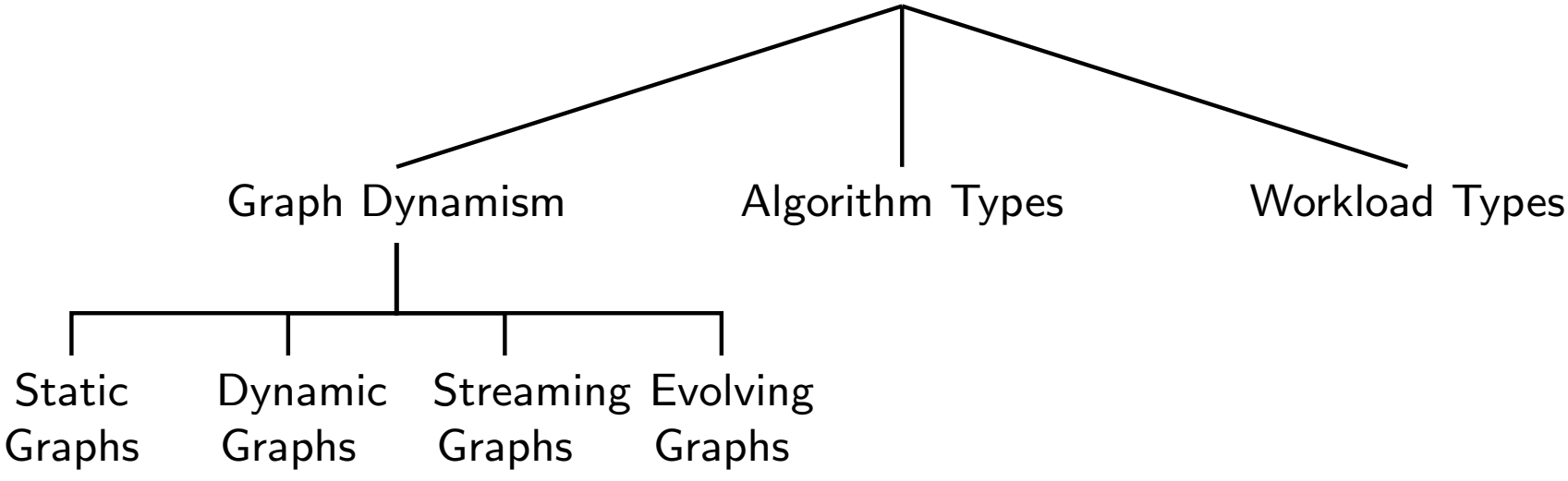
Classification



Classification

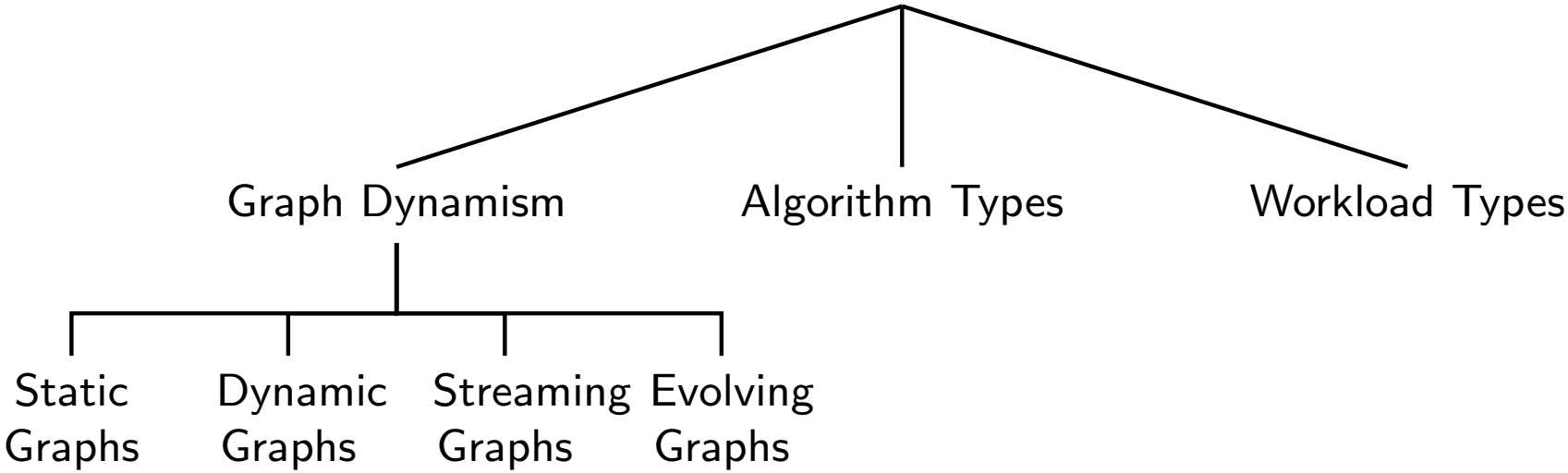


Classification



Graphs do not change or we are not interested in their changes – only a *snapshot* is considered.

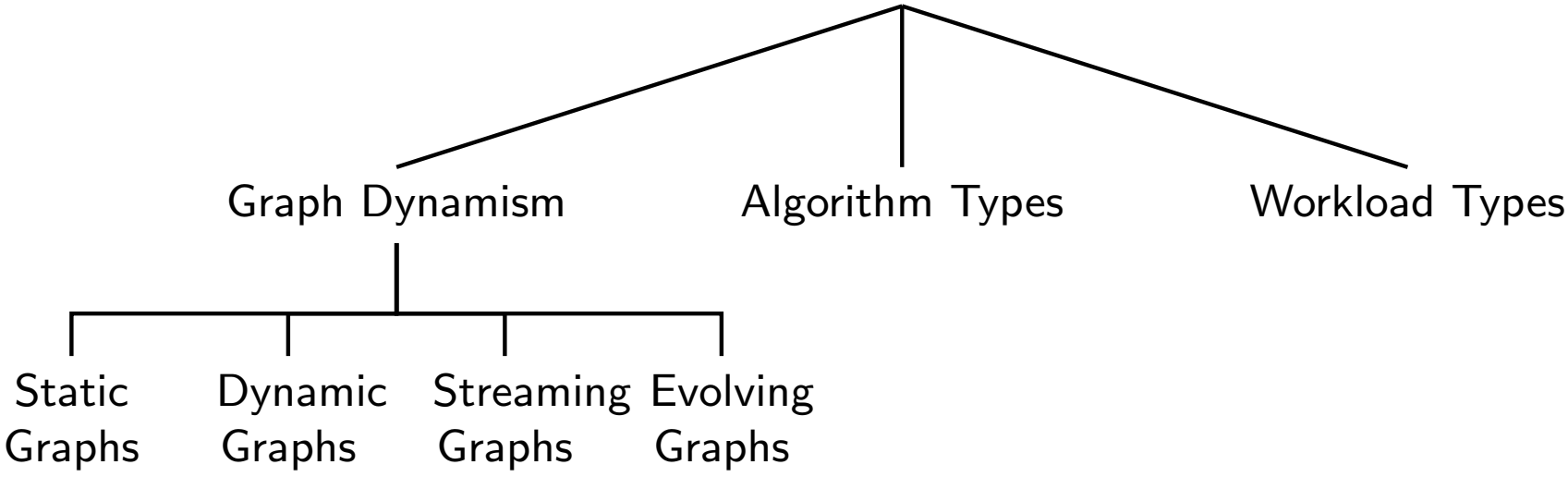
Classification



Graphs do not change or we are not interested in their changes – only a *snapshot* is considered.

Graphs change and we are interested in their changes.

Classification

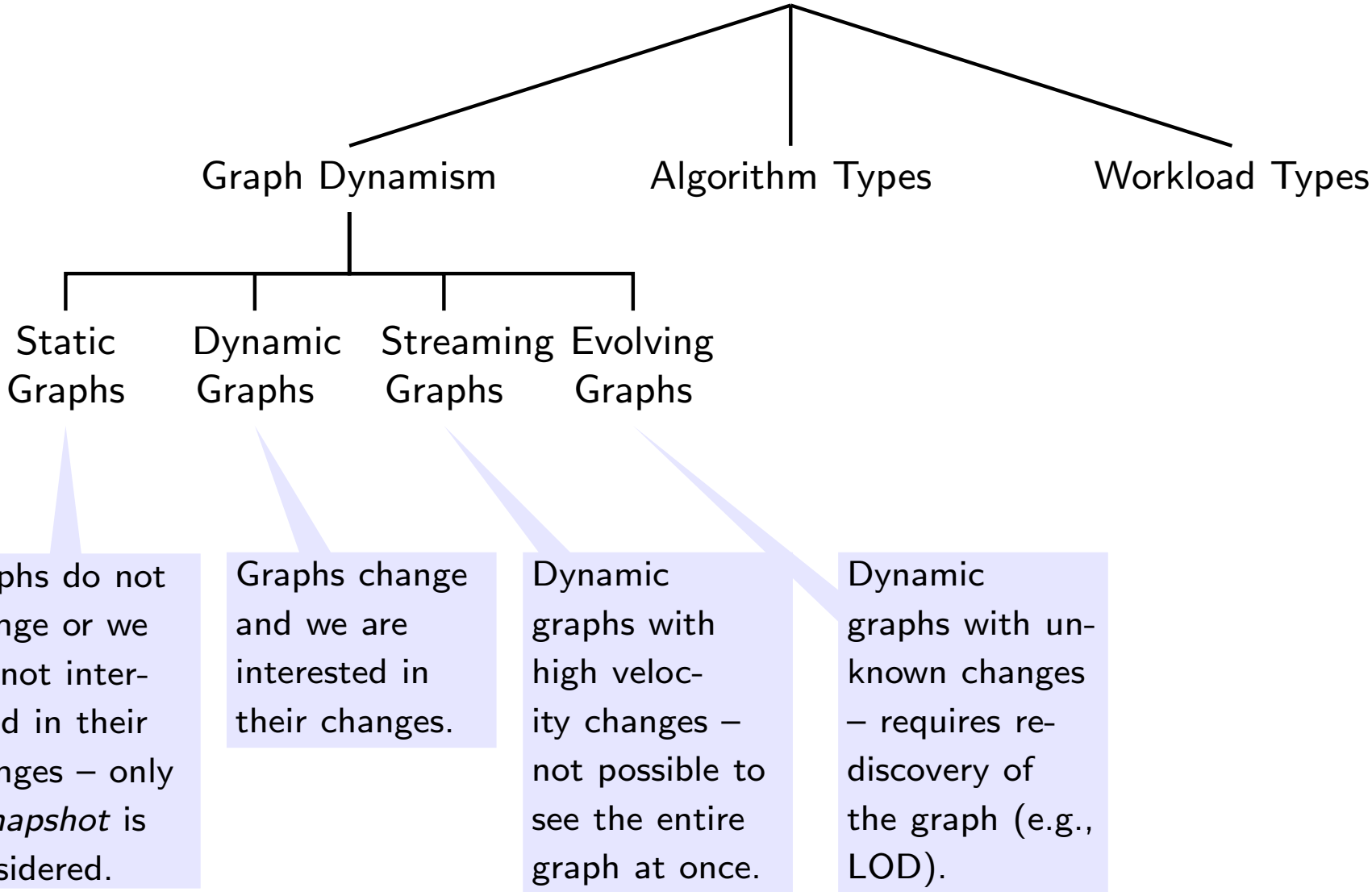


Graphs do not change or we are not interested in their changes – only a *snapshot* is considered.

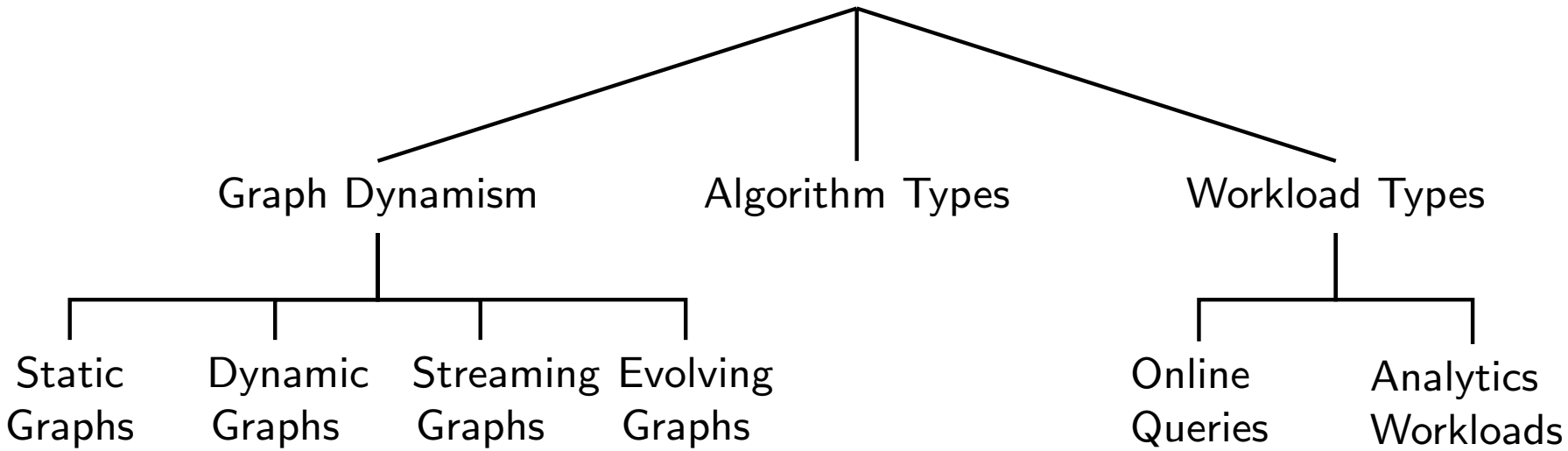
Graphs change and we are interested in their changes.

Dynamic graphs with high velocity changes – not possible to see the entire graph at once.

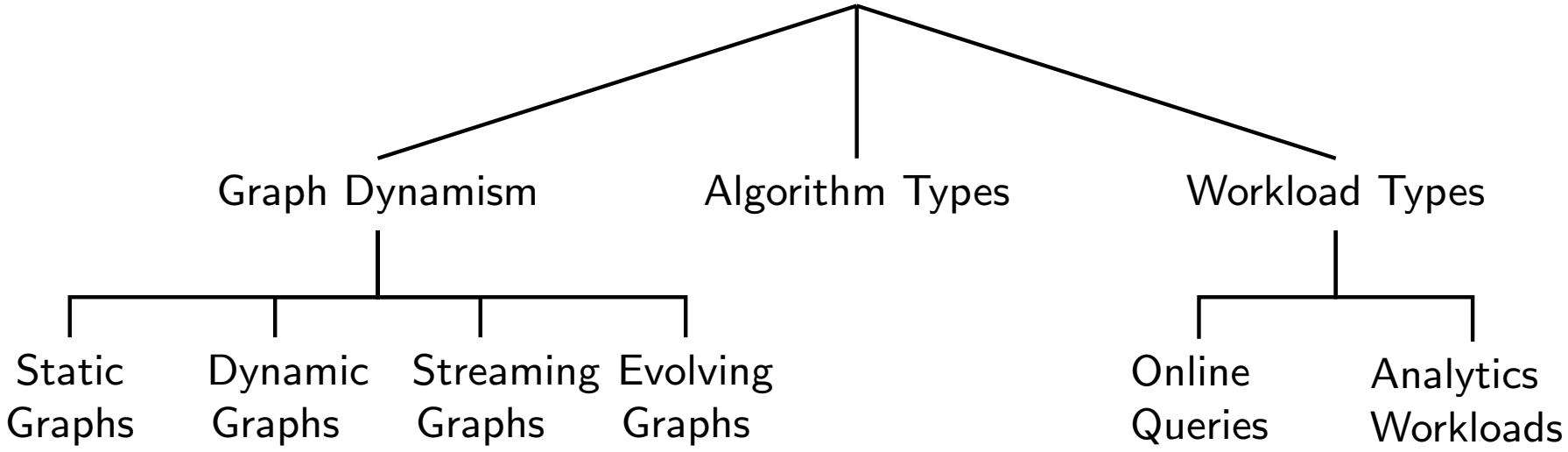
Classification



Classification

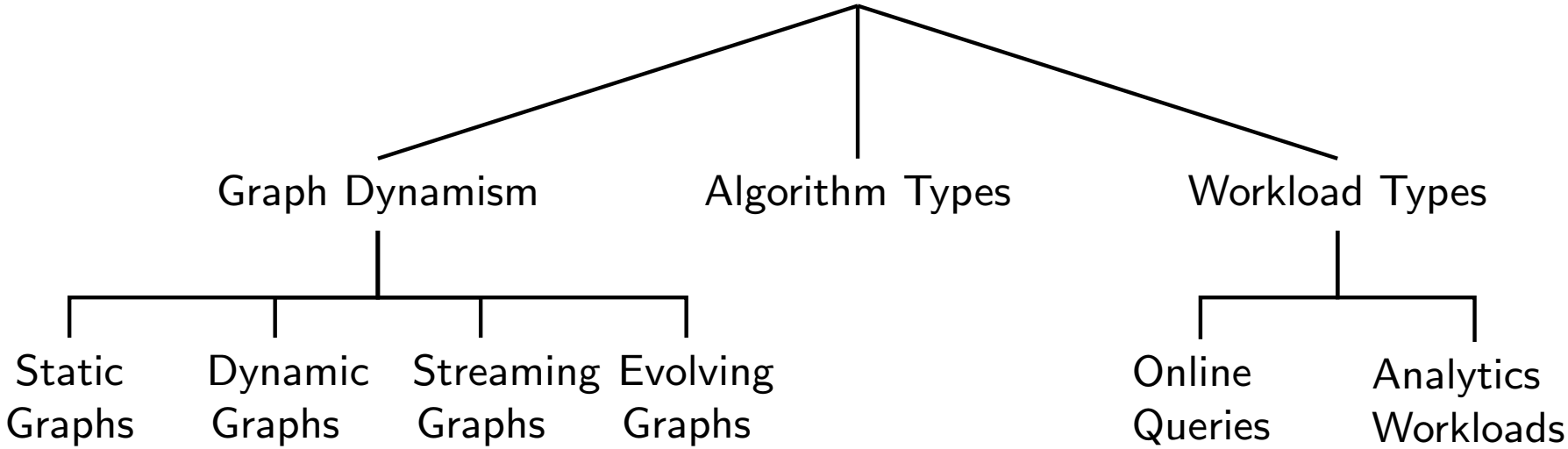


Classification



Computation accesses a portion of the graph and the results are computed for a subset of vertices; e.g., point-to-point shortest path, subgraph matching, reachability, SPARQL.

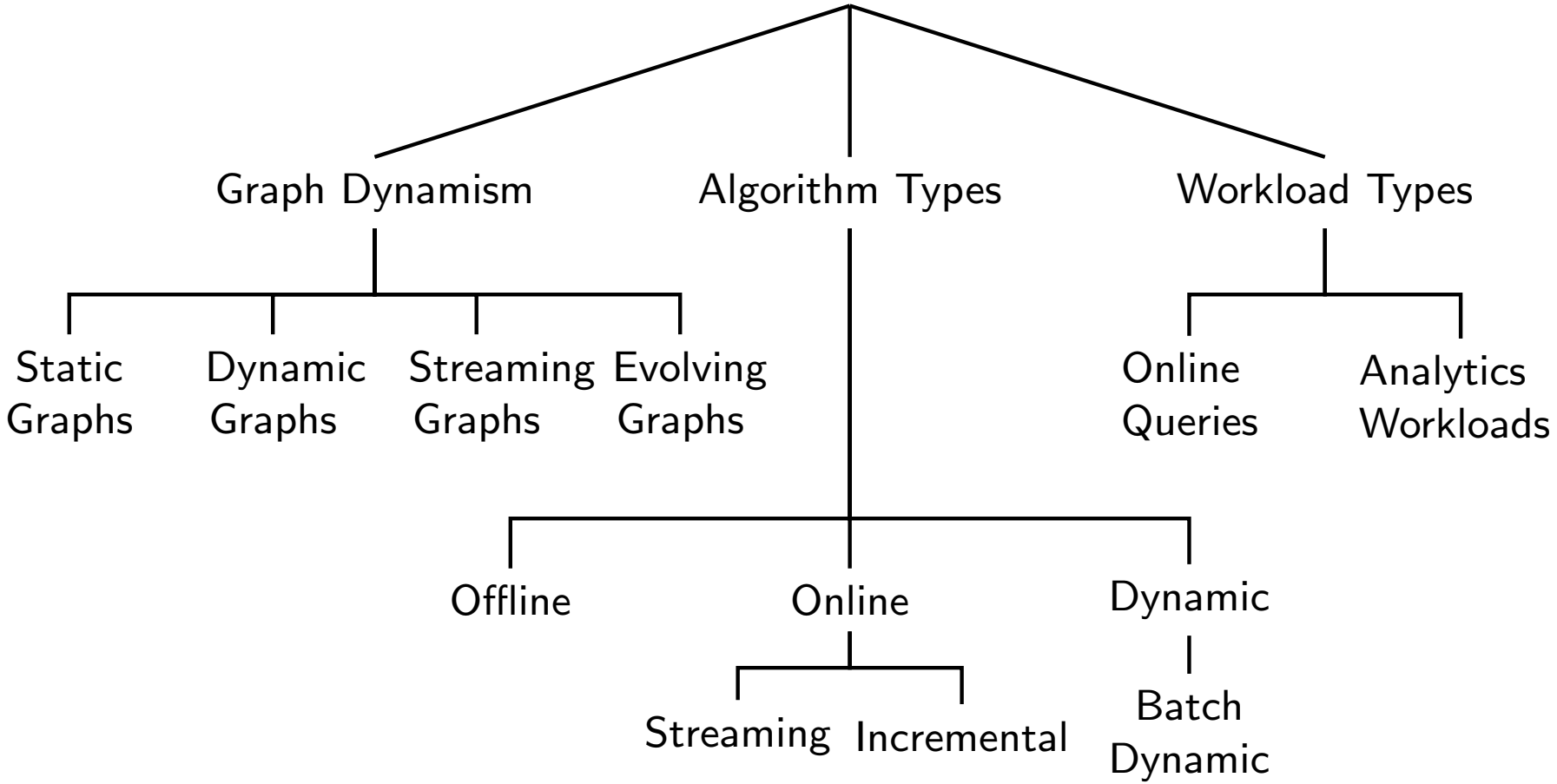
Classification



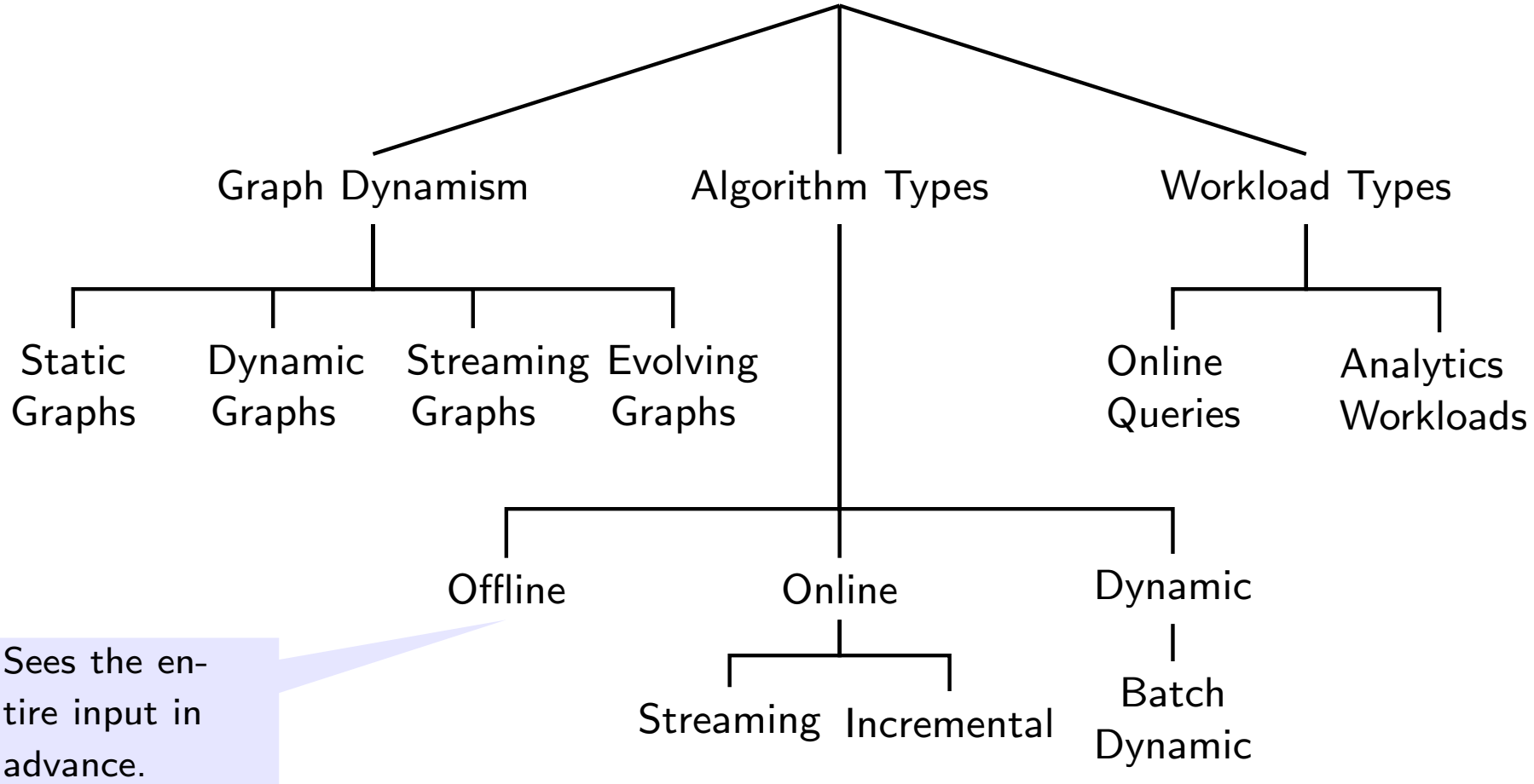
Computation accesses a portion of the graph and the results are computed for a subset of vertices; e.g., point-to-point shortest path, subgraph matching, reachability, SPARQL.

Computation accesses the entire graph and may require multiple iterations; e.g., PageRank, clustering, graph colouring, all pairs shortest path.

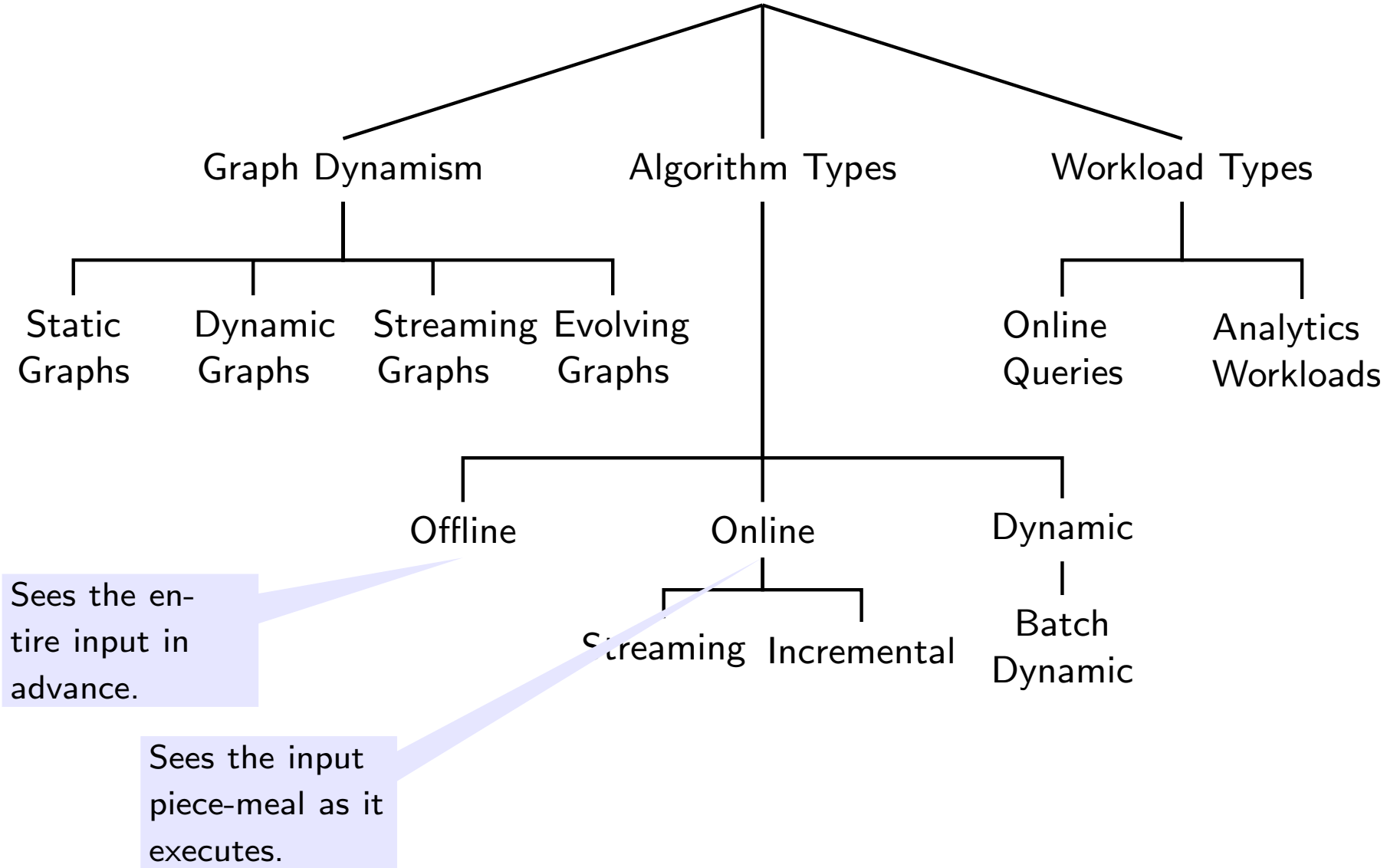
Classification



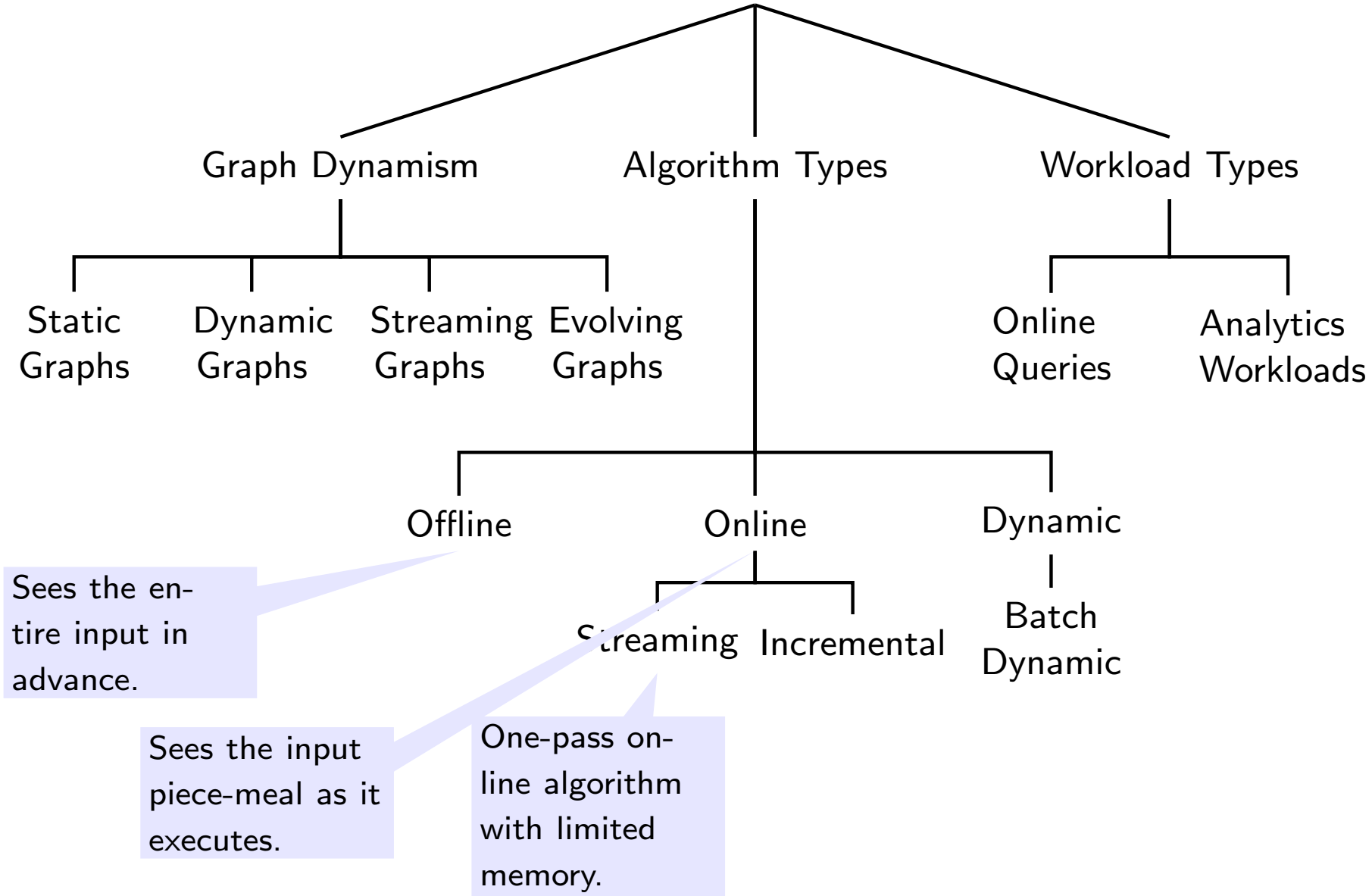
Classification



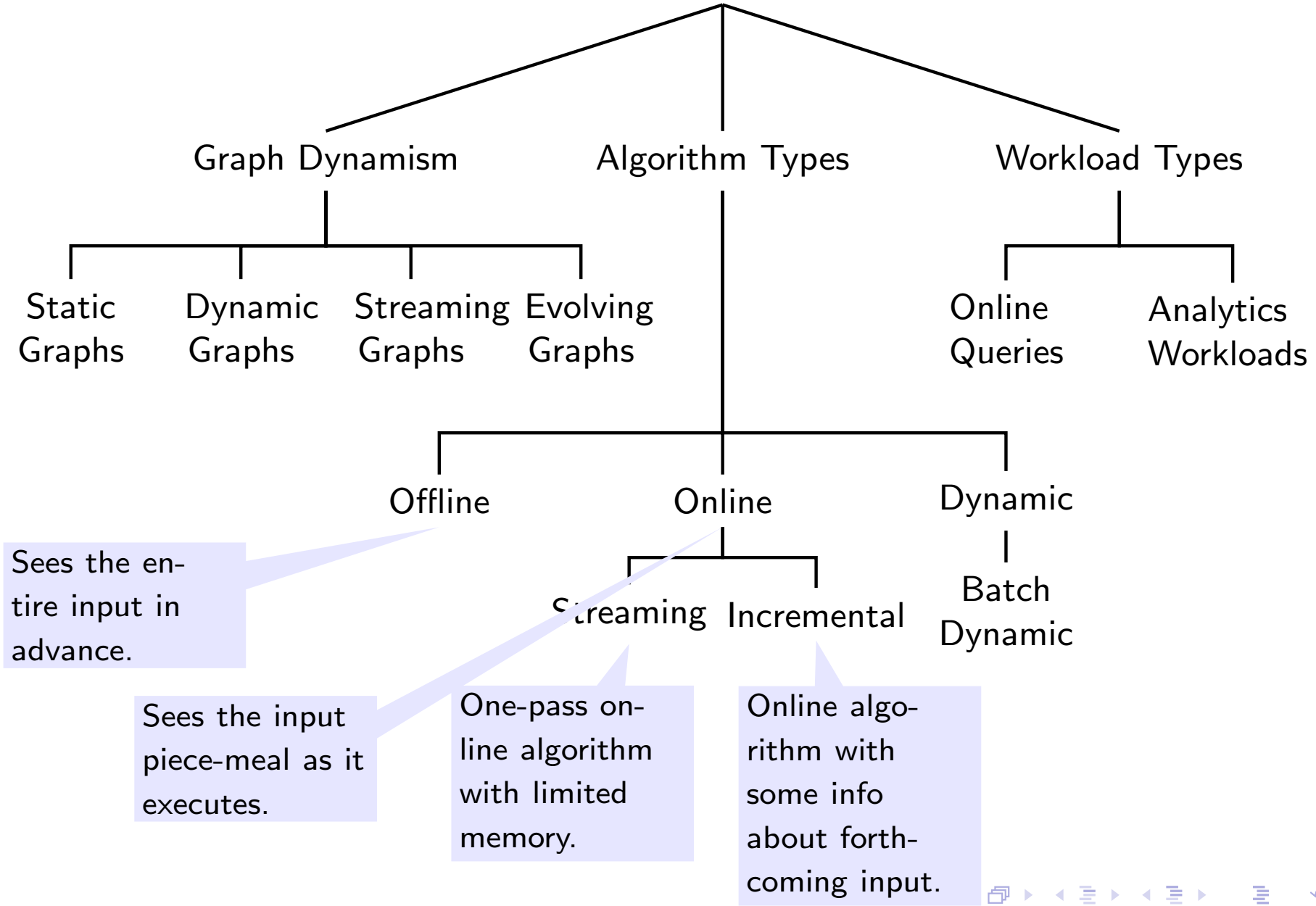
Classification



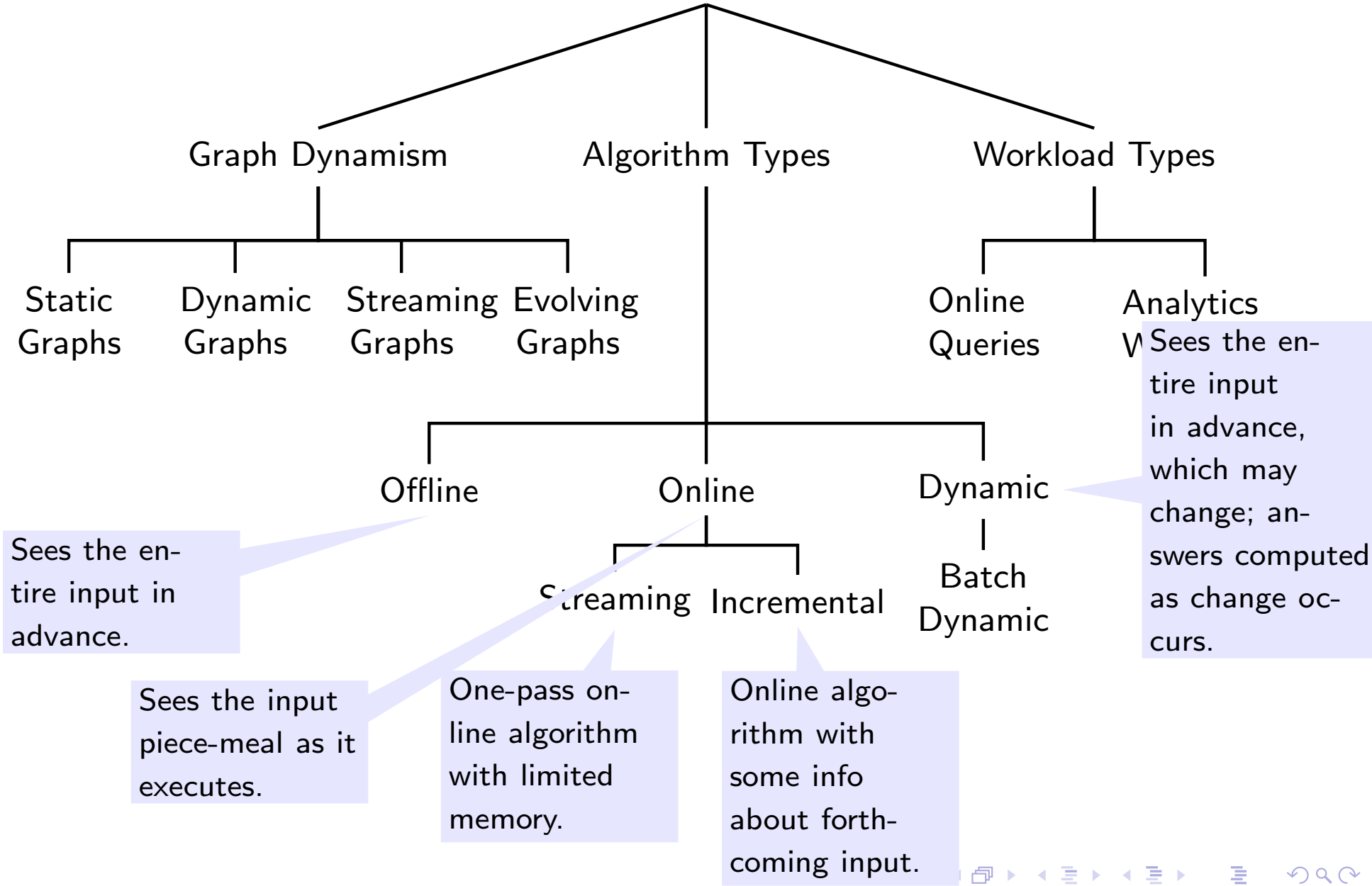
Classification



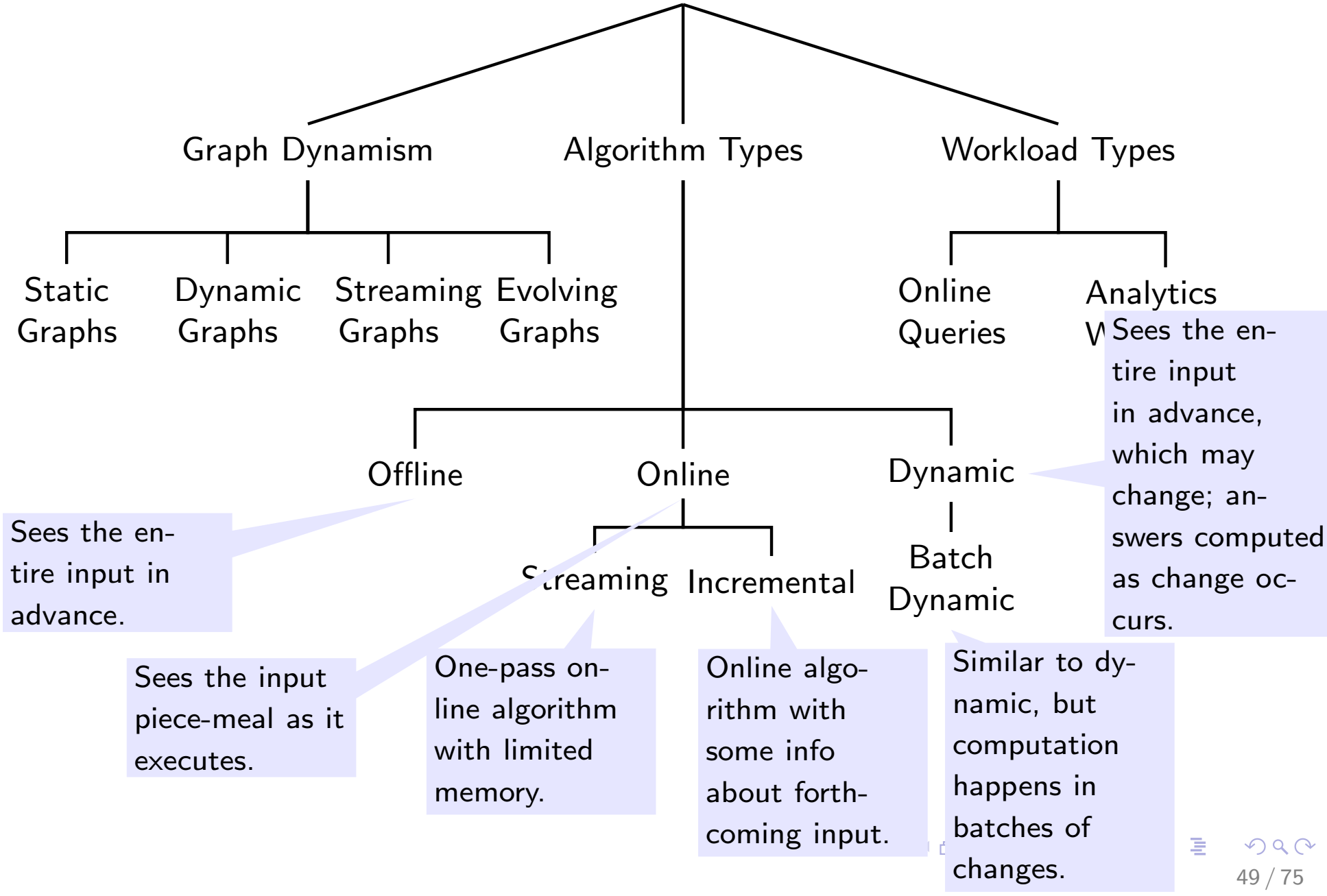
Classification



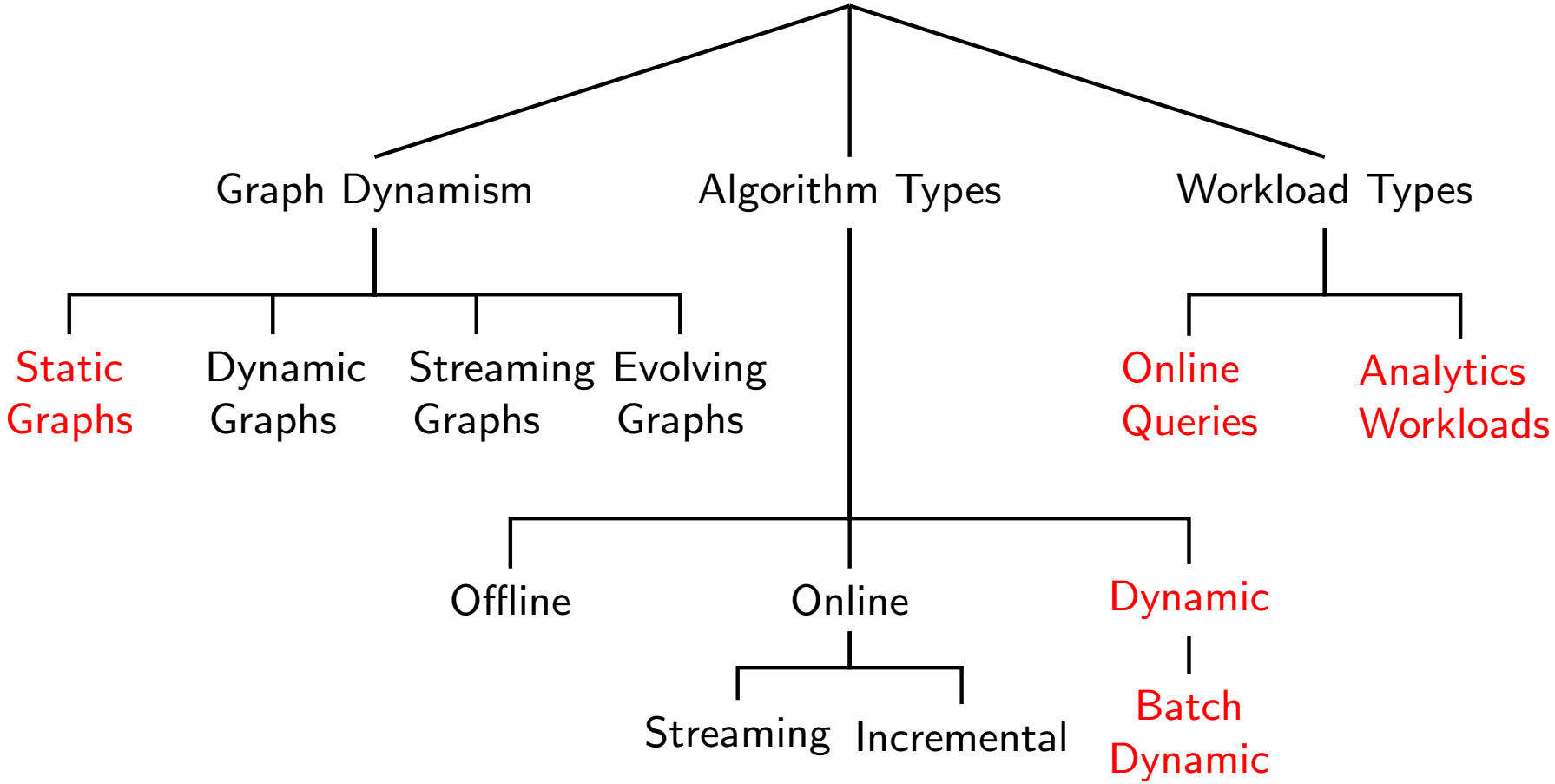
Classification



Classification



Example Design Points – Not all alternatives make sense



Dynamic (or batch-dynamic) algorithms do not make sense for static graphs.

Graph Workloads

Offline graph analytics

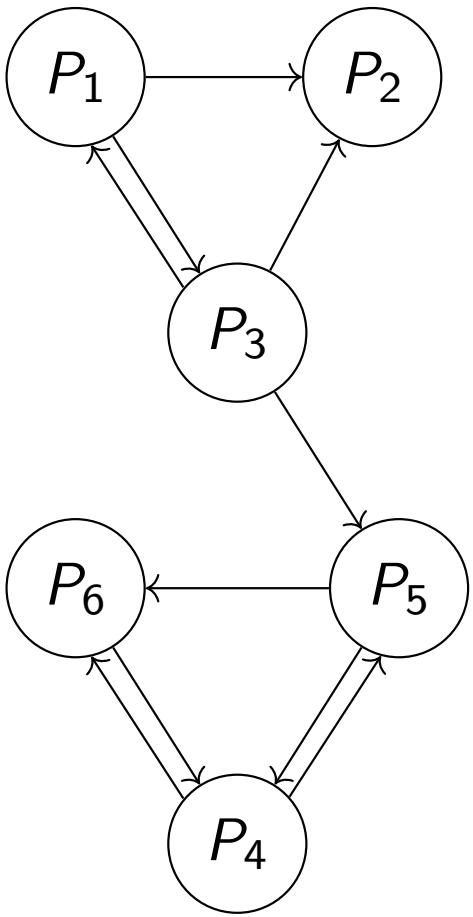
- PageRank
- Clustering
- Strongly connected components
- Diameter finding
- Graph colouring
- All pairs shortest path
- Graph pattern mining
- Machine learning algorithms (Belief propagation, Gaussian non-negative matrix factorization)

Online graph querying

- Reachability
- Single source shortest-path
- Subgraph matching
- SPARQL queries

PageRank Computation

A web page is important if it is pointed to by other important pages.



$$r(P_i) = \sum_{P_j \in B_{P_i}} \frac{r(P_j)}{|F_{P_j}|}$$

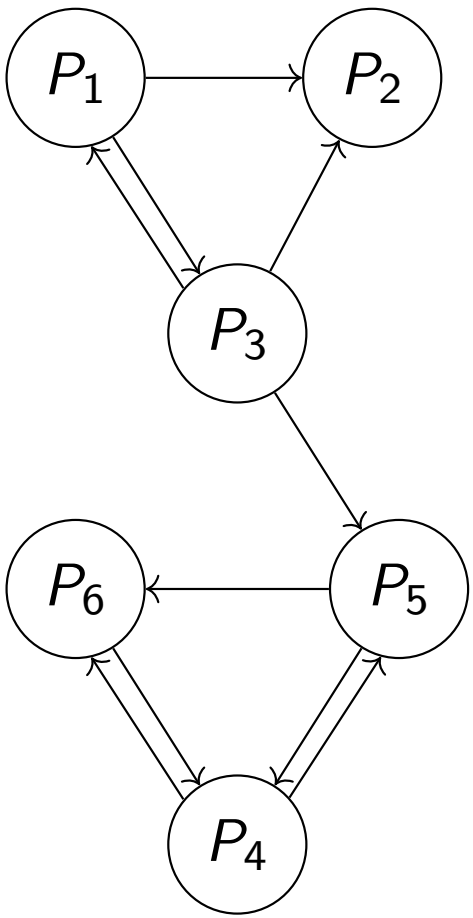
$$r(P_2) = \frac{r(P_1)}{2} + \frac{r(P_3)}{3}$$

$$r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|F_{P_j}|}$$

B_{P_i}: in-neighbours of P_i
F_{P_i}: out-neighbours of P_i

PageRank Computation

A web page is important if it is pointed to by other important pages.



$$r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|F_{P_j}|}$$

Iteration 0	Iteration 1	Iteration 2	Rank at Iter. 2
$r_0(P_1) = 1/6$	$r_1(P_1) = 1/18$	$r_2(P_1) = 1/36$	5
$r_0(P_2) = 1/6$	$r_1(P_2) = 5/36$	$r_2(P_2) = 1/18$	4
$r_0(P_3) = 1/6$	$r_1(P_3) = 1/12$	$r_2(P_3) = 1/36$	5
$r_0(P_4) = 1/6$	$r_1(P_4) = 1/4$	$r_2(P_4) = 17/72$	1
$r_0(P_5) = 1/6$	$r_1(P_5) = 5/36$	$r_2(P_5) = 11/72$	3
$r_0(P_6) = 1/6$	$r_1(P_6) = 1/6$	$r_2(P_6) = 14/72$	2

Iterative processing.

Some Alternative Computational Models for Offline Analytics

- MapReduce
  - `map` and `reduce` functions
  - Not suitable for iterative processing due to data movement at each stage
  - Need to save in storage system intermediate results of each iteration

Some Alternative Computational Models for Offline Analytics

- MapReduce
  - `map` and `reduce` functions
  - Not suitable for iterative processing due to data movement at each stage
  - Need to save in storage system intermediate results of each iteration
- Vertex-centric paradigm
  - Specify (a) the computation to be performed at each vertex, and (b) its communication with neighbour vertices
  - Designed specifically for interactive graph processing
  - Synchronous (e.g., Pregel, Giraph)

  - Asynchronous (e.g., GraphLab)

Pregel-like Graph Processing Systems

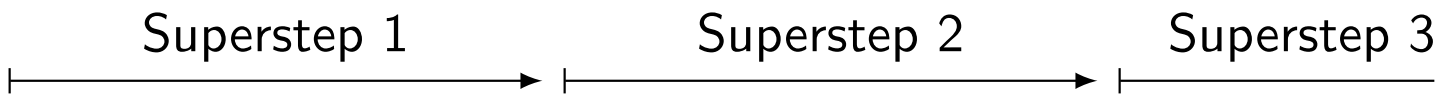
Pregel-like systems are **BSP**, **vertex-centric** programs.

Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.

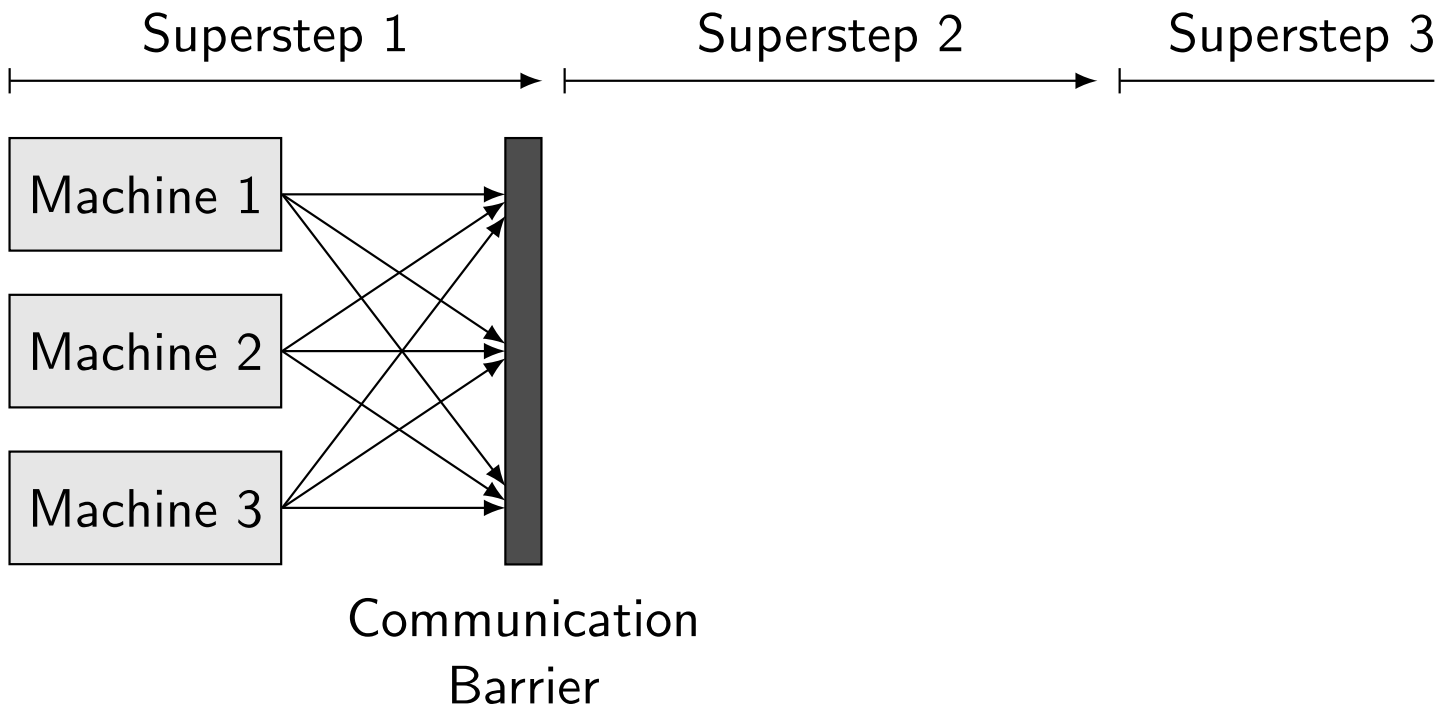
Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.



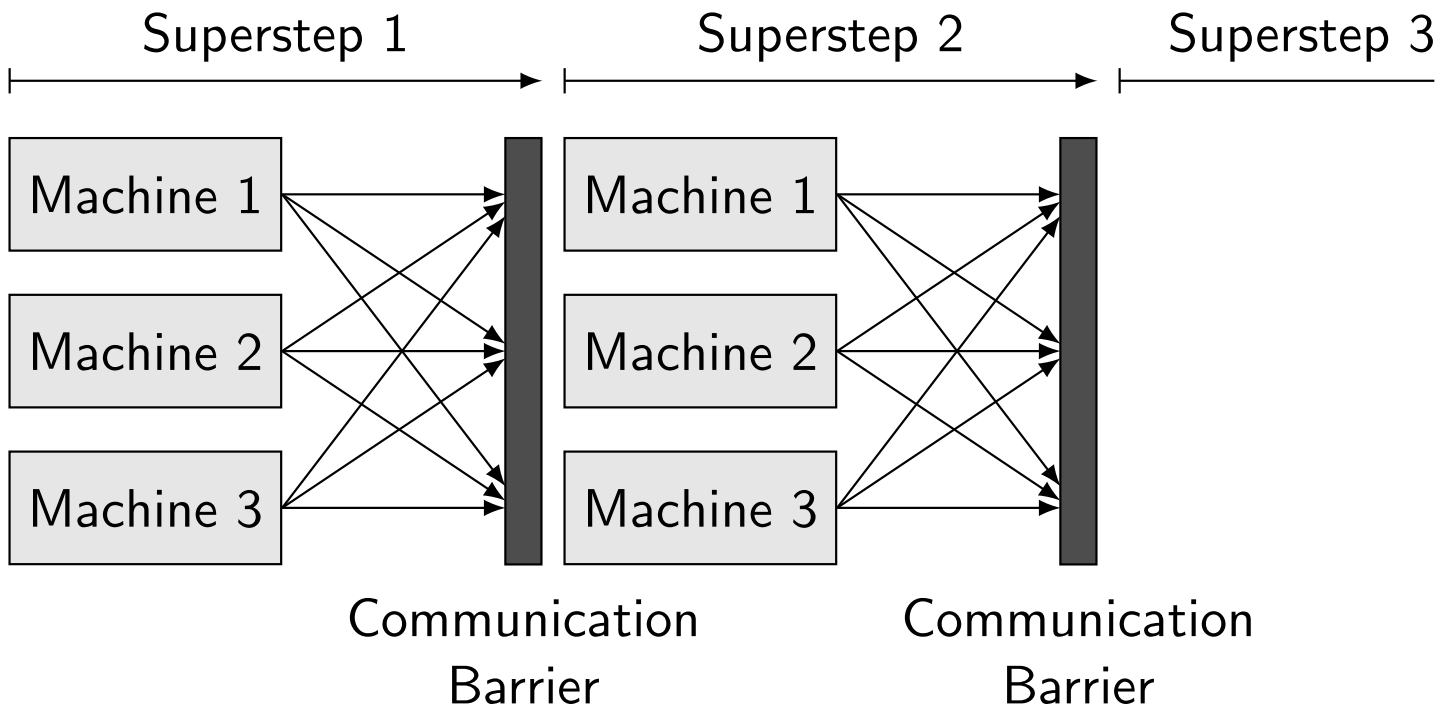
Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.



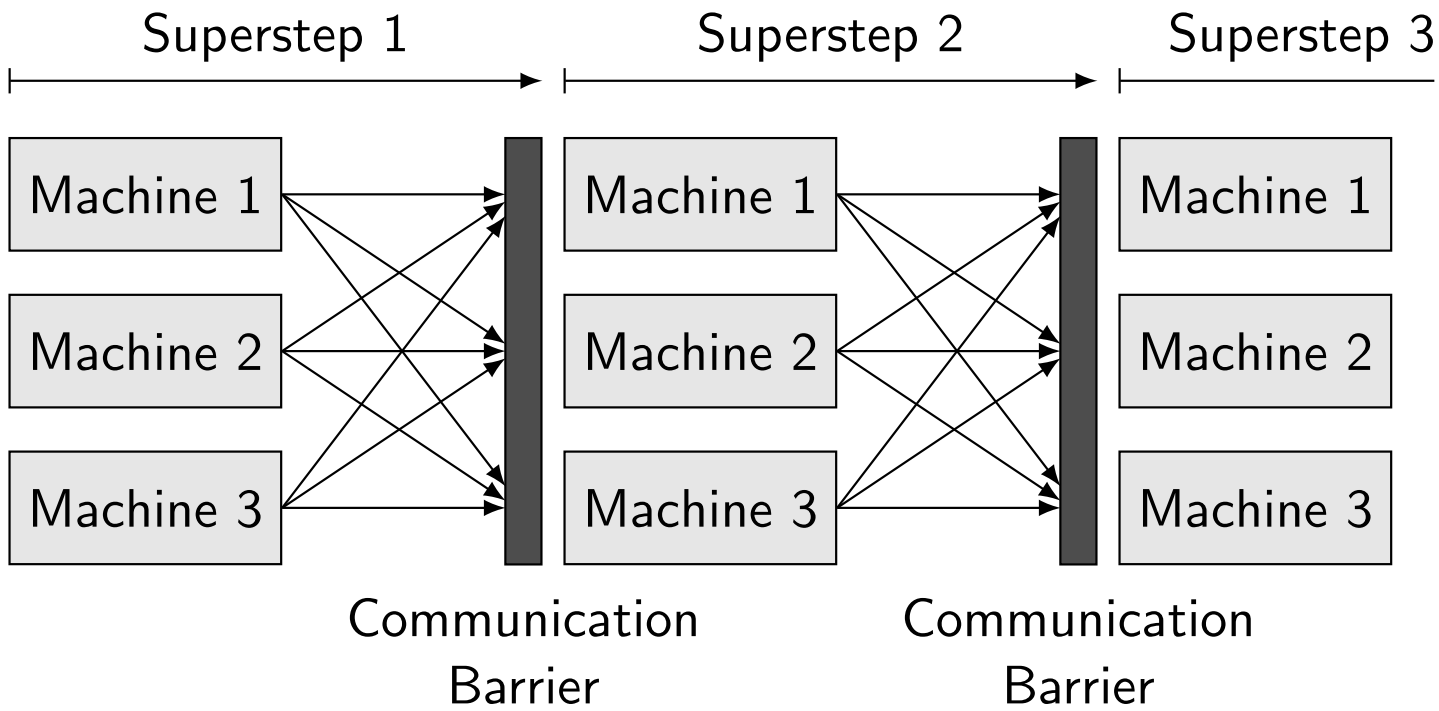
Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.



Pregel-like Graph Processing Systems

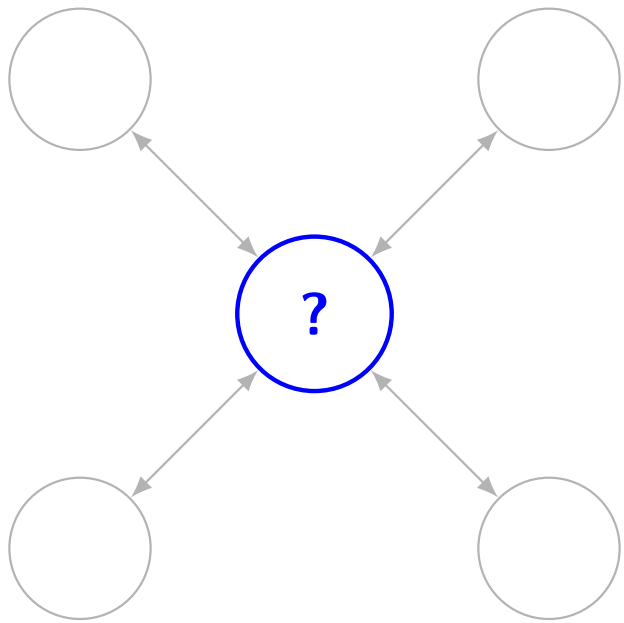
Pregel-like systems are **BSP**, **vertex-centric** programs.



Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.

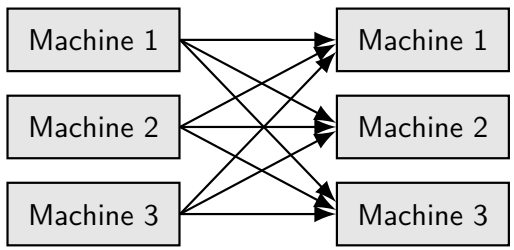
- "Think like a vertex":



GraphLab (Asynchronous)

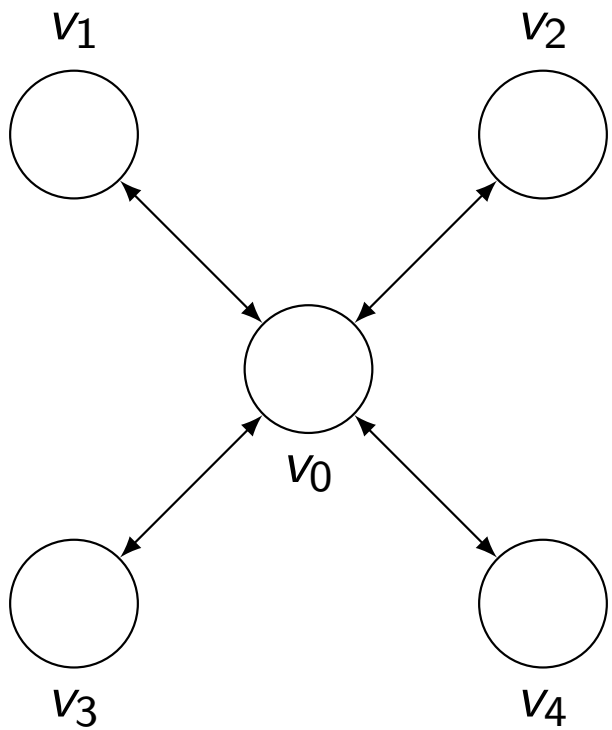
GraphLab features *asynchronous* execution:

- No communication barriers. ✓
- Uses the *most recent* vertex values. ✓



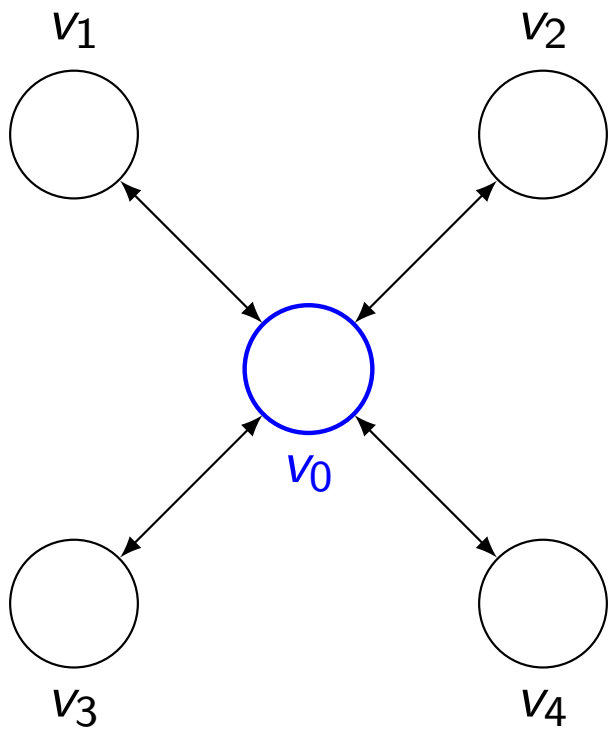
GraphLab (Asynchronous)

Implemented via distributed locking:



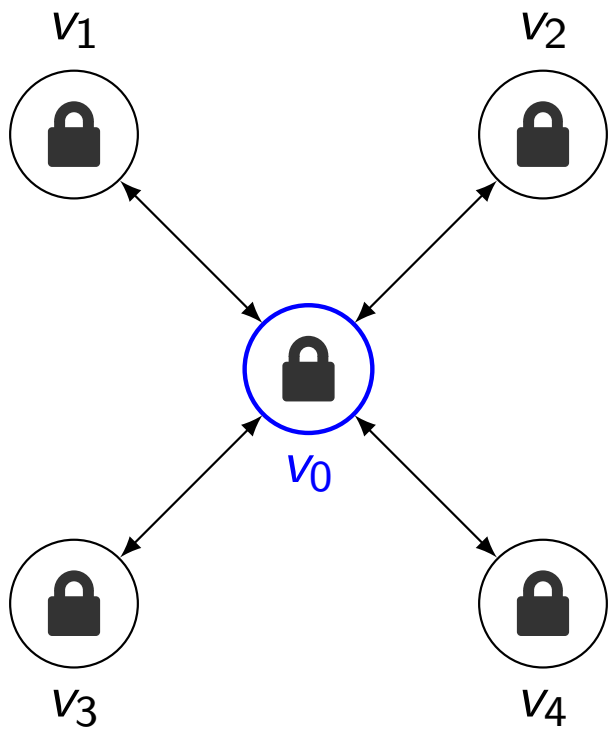
GraphLab (Asynchronous)

Implemented via distributed locking:



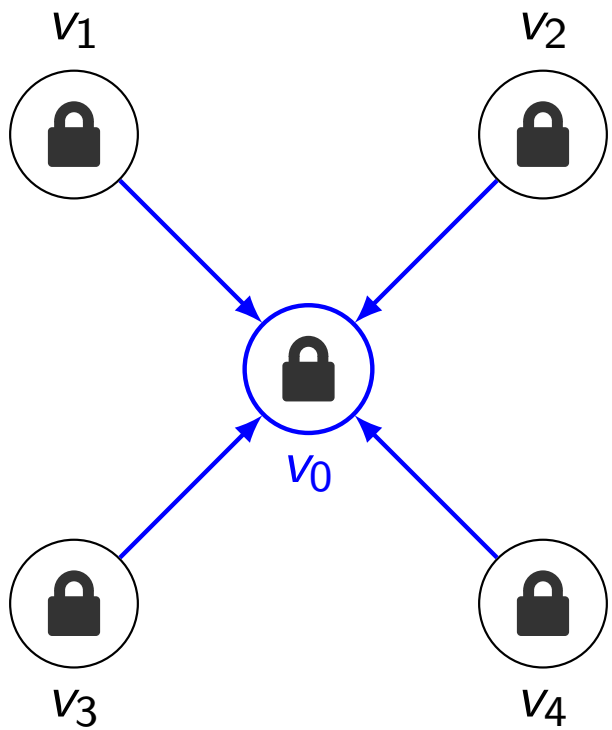
GraphLab (Asynchronous)

Implemented via distributed locking:



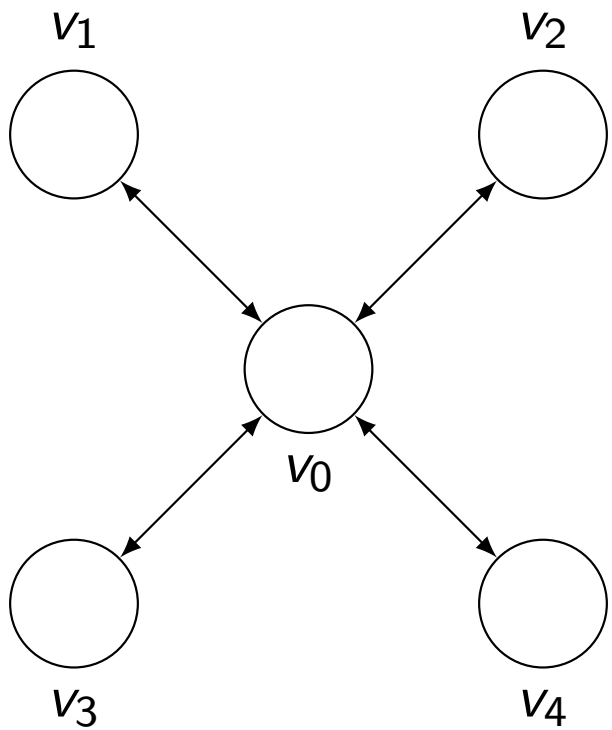
GraphLab (Asynchronous)

Implemented via distributed locking:

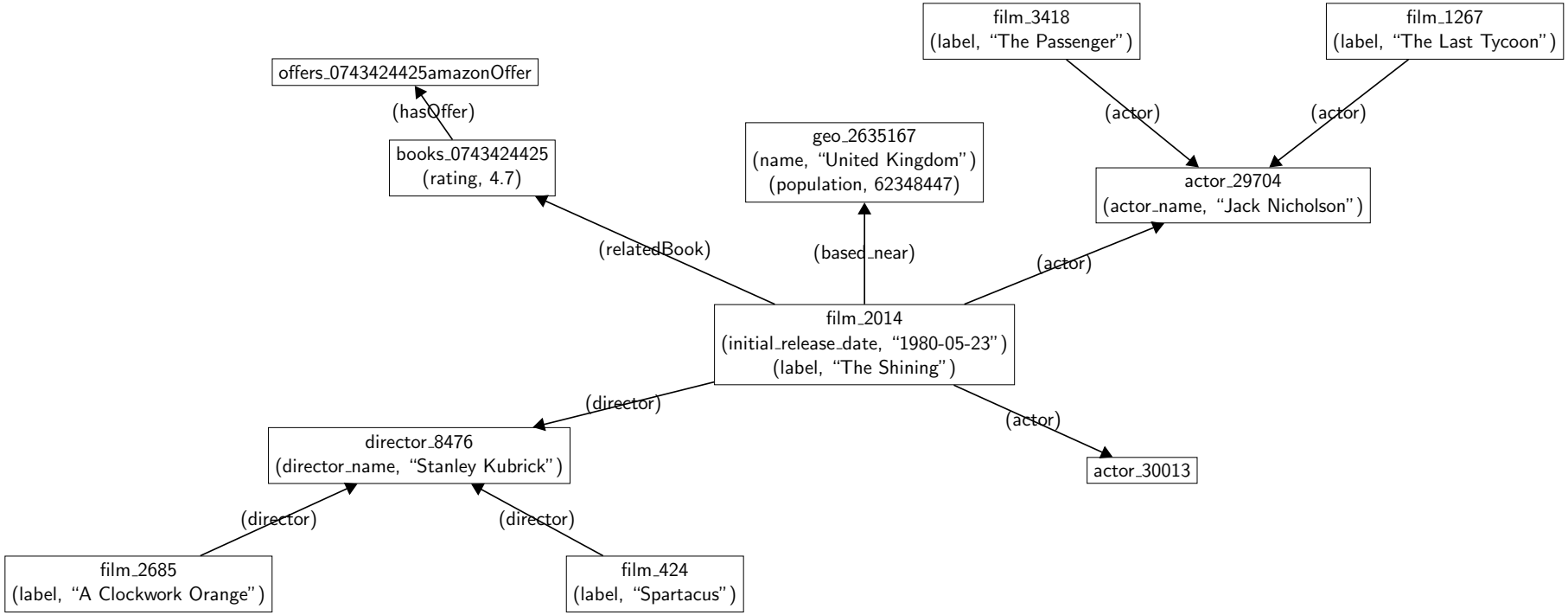


GraphLab (Asynchronous)

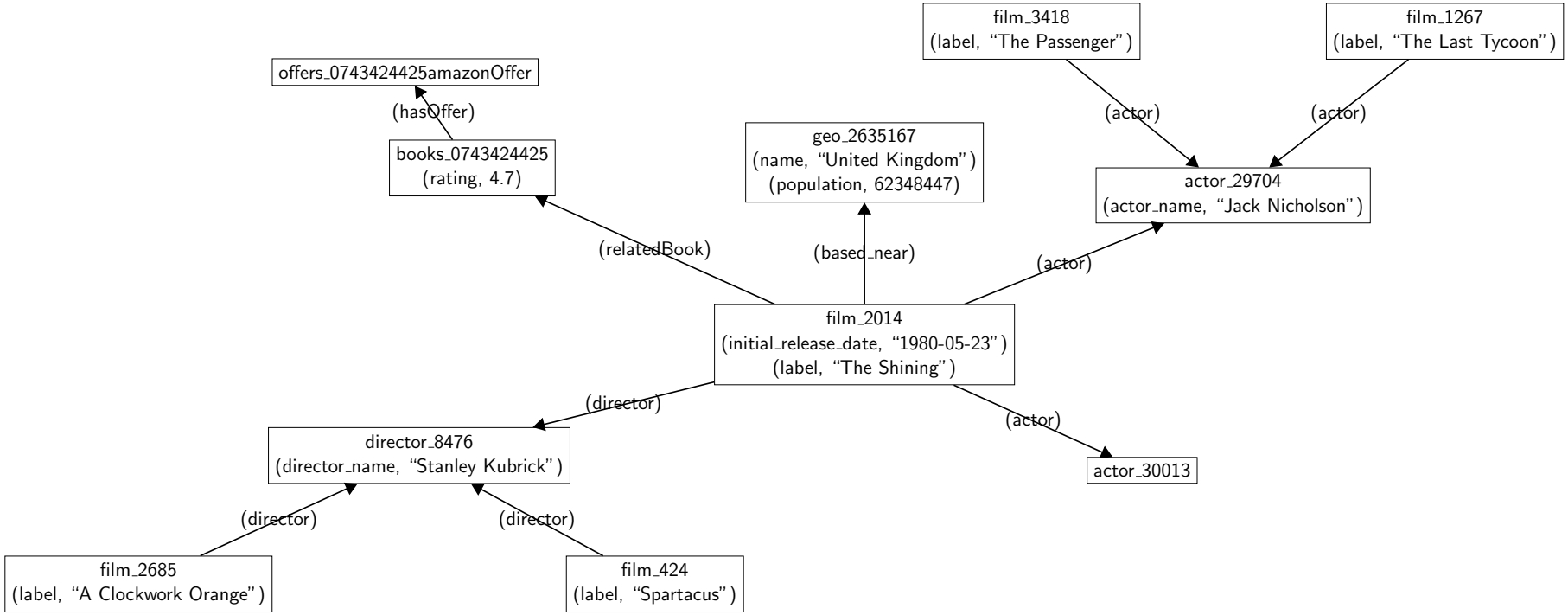
Implemented via distributed locking:



Reachability Queries

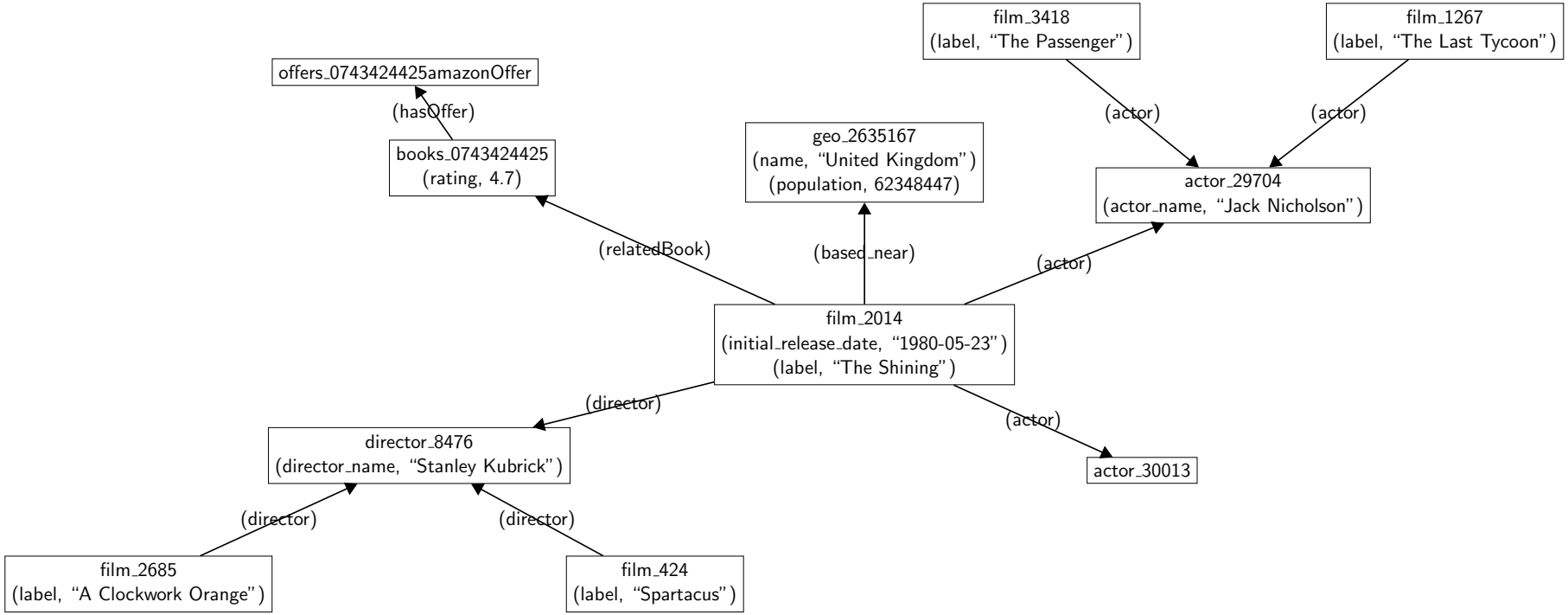


Reachability Queries



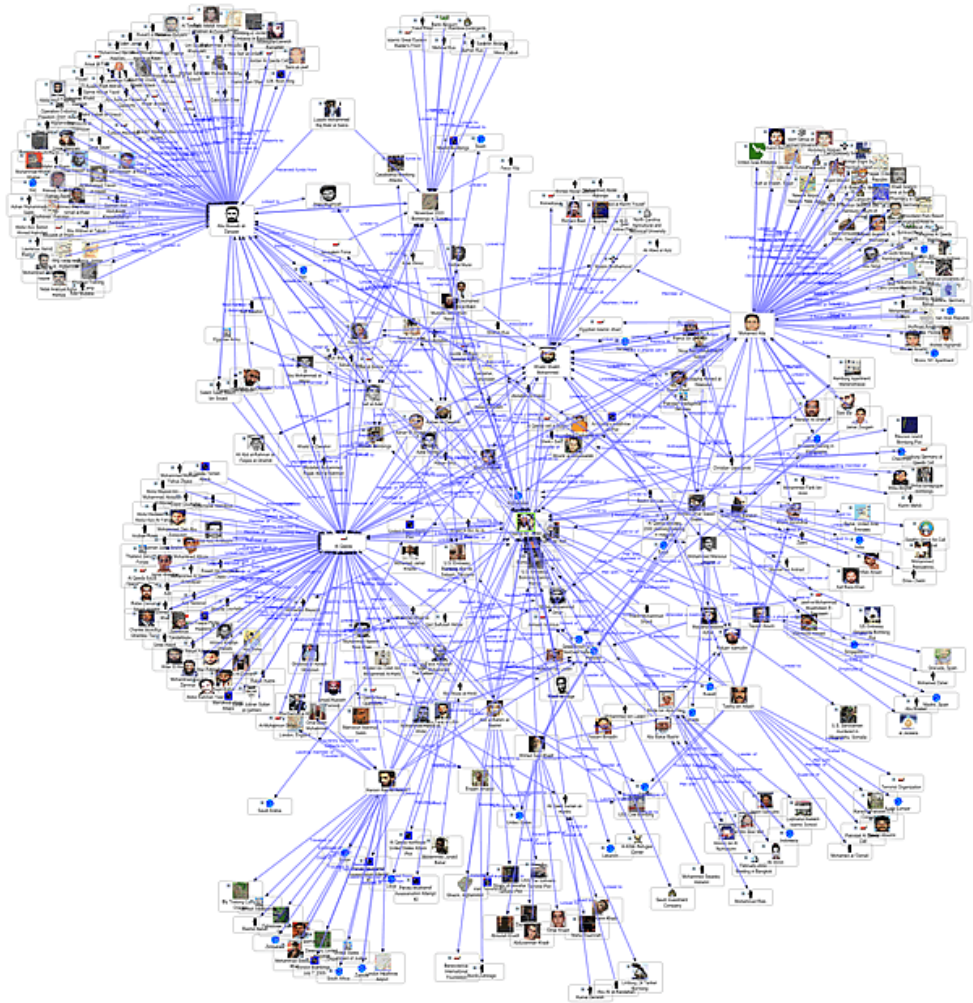
Can you reach film_1267 from film_2014?

Reachability Queries



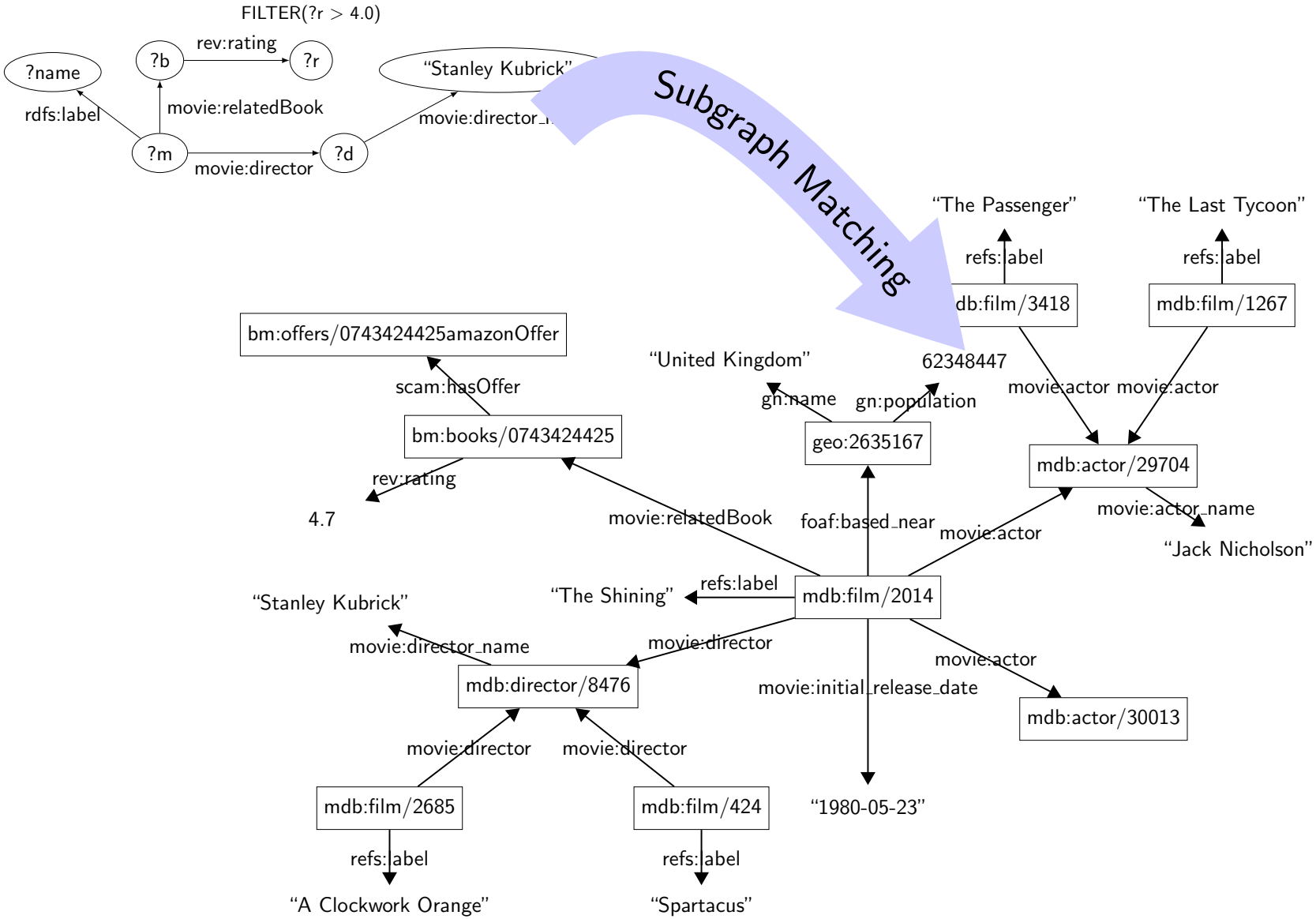
Is there a book whose rating is > 4.0 associated with a film that was directed by Stanley Kubrick?

Reachability Queries



Think of Facebook graph and finding friends of friends.

Subgraph Matching



For more information I

Graph-based SPARQL processing

- Lei Zou, Jinghui Mo, Lei Chen, M. Tamer Özsu, and Dongyan Zhao. gStore: answering SPARQL queries via subgraph matching. *Proc. VLDB Endowment*, 4(8):482–493, 2011
- Lei Zou, M. Tamer Özsu, Lei Chen, Xuchuan Shen, Ruizhe Huang, and Dongyan Zhao. gStore: A graph-based SPARQL query engine. *VLDB J.*, 23(4):565–590, 2014
- Güneş Aluç, M. Tamer Özsu, Khuzaima Daudjee, and Olaf Hartig. chameleon-db: a workload-aware robust RDF data management system. Technical Report CS-2013-10, University of Waterloo, 2013. Available at `https://cs.uwaterloo.ca/sites/ca.computer-science/files/uploads/files/CS-2013-10.pdf`

For more information II

More on RDF graph management

- Olaf Hartig and M. Tamer Özsu. Linked data query processing. In *Proc. 30th Int. Conf. on Data Engineering*, pages 1286–1289, 2014. Tutorial description
- More organized slides from my talks are available at `http://www.slideshare.net/MTamerOzsu/web-data-management-with-rdf`

General graph processing

- Arijit Khan and Sameh Elnikety. Systems for big-graphs. *Proc. VLDB Endowment*, 7(13):1709–1710, 2014
- Slides available at `http://people.inf.ethz.ch/khana/Papers/2014_VLDB_GraphSystemsTutorial.pptx`