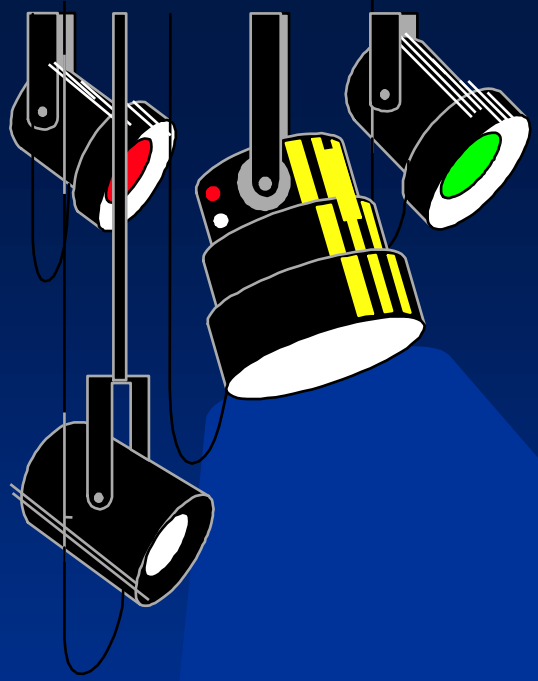


## Outline

- Graph Databases
  - Graph Database Types
  - Graph Storage on Disk
  - Retrieve a graph database
  - Add a new node

- **What is a graph database**
  - A graph database is defined as a specialized, single-purpose platform for creating and manipulating graphs.
  - Graphs contain nodes, edges, and properties, all of which are used to represent and store data in a way that relational databases are not equipped to do.
  - Graph analytics is another commonly used term, and it refers specifically to the process of analyzing data in a graph format using data points as nodes and relationships as edges.
  - Graph analytics requires a database that can support graph formats; this could be a dedicated graph database, or a converged database that supports multiple data models, including graphs.

- **Graph database types**
  - There are two popular models of graph databases: property graphs and RDF graphs.
  - The property graph focuses on analytics and querying.
  - The RDF graph emphasizes data integration.
  - Both types of graphs consist of a collection of points (vertices) and the connections between those points (edges). But there are differences as well.

- **Property graphs**
  - Property graphs are used to model relationships among data, and they enable query and data analytics based on these relationships.
  - A property graph has vertices that can contain detailed information about a subject, and edges that denote the relationship between the vertices. The vertices and edges can have attributes, called properties, with which they are associated.

- **Property graphs**
  - Example

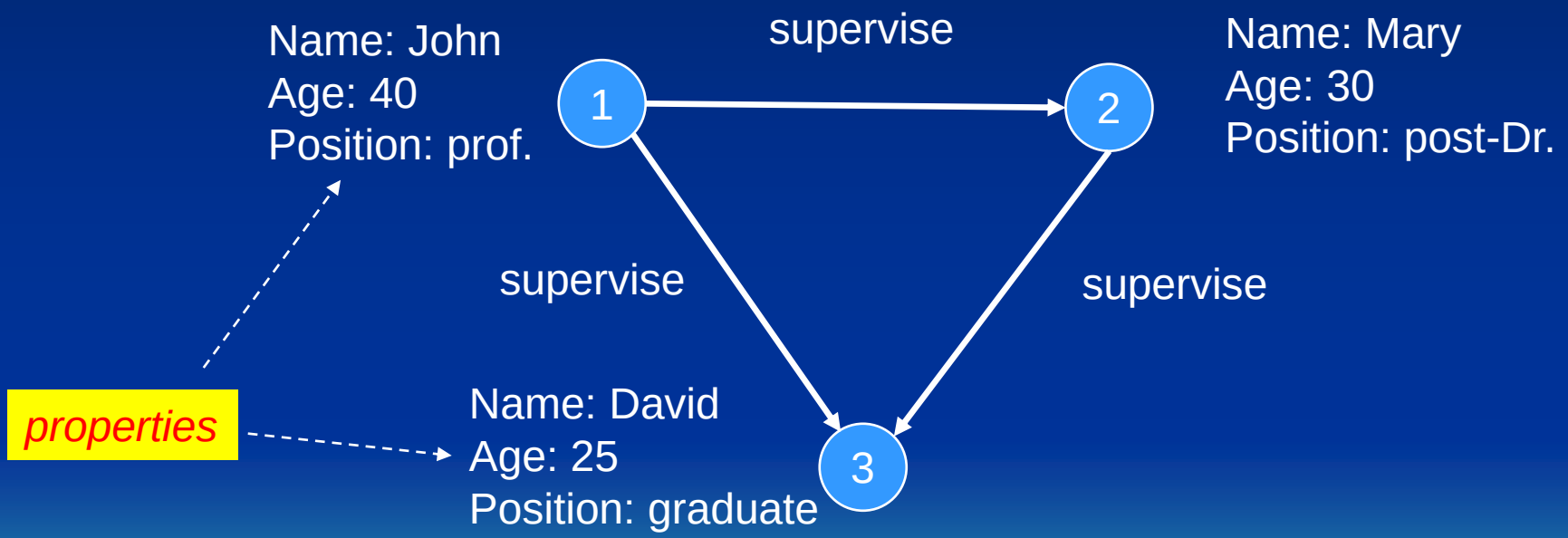



Figure 1

- **RDF**

- RDF graphs (RDF stands for Resource Description Framework) conform to a set of W3C (Worldwide Web Consortium) standards designed to represent statements and are best for representing complex metadata and master data.
- They are often used for linked data, data integration, and knowledge graphs.
- They can represent complex concepts in a domain, or provide rich semantics and inferencing on data.

- **RDF**

- In the RDF model a statement is represented by three elements: two vertices connected by an edge reflecting the subject, predicate and object of a sentence:

  *<subject><predicate><object>.*

  This is known as an *RDF triple*.

- Every vertex and edge is identified by a unique URI, or Unique Resource Identifier.

- The RDF model provides a way to publish data in a standard format with well-defined semantics, enabling information exchange.

- Government statistics agencies, pharmaceutical companies, and healthcare organizations have adopted RDF graphs widely.

- **RDF**
  - Example

In an RDF graph, each edge represents a statement.

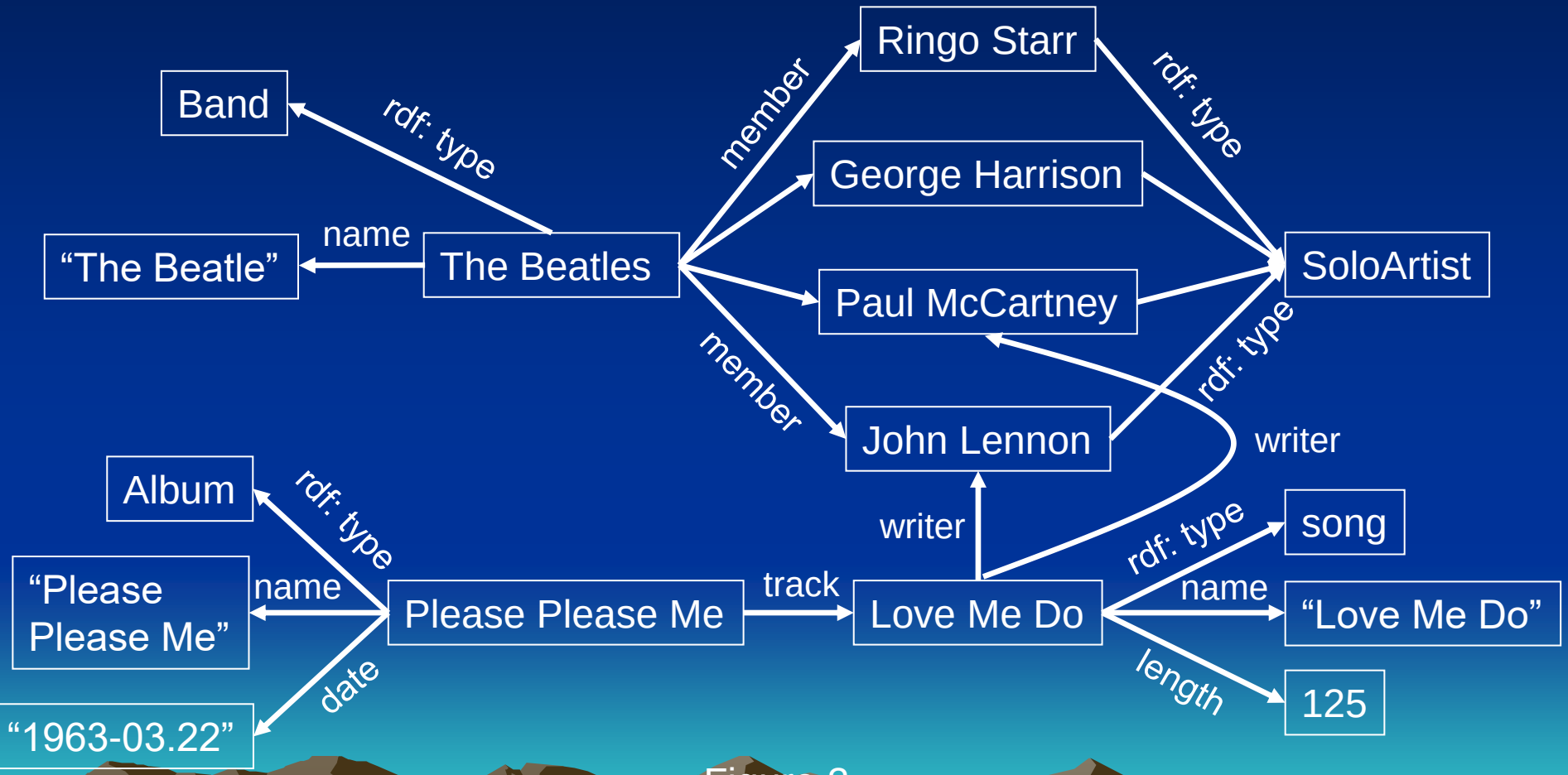



Figure 2

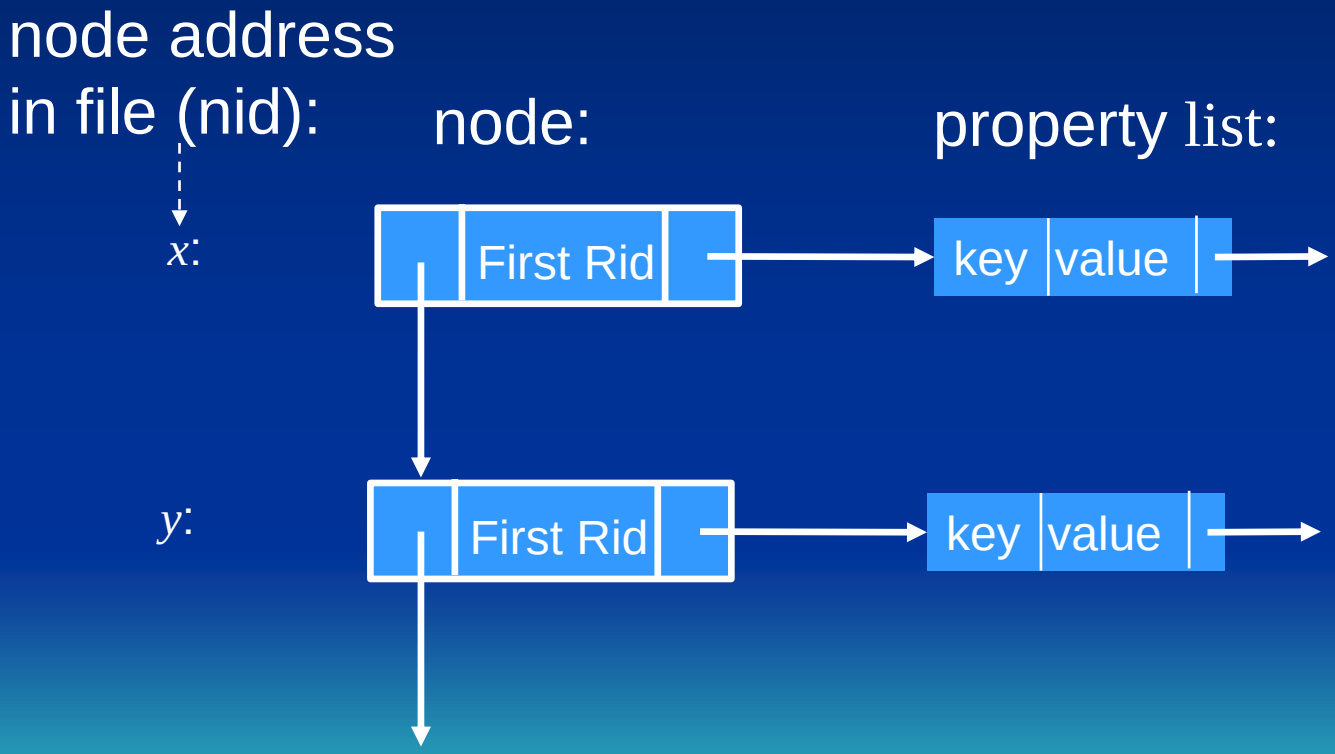
- This graph shows several nodes that represent entities such as the Beatles band and one of their studio albums. Each edge has an identifier that tells us what relationship holds between those nodes. For example, the member edge links bands to its members. The rdf: type edge represents a special kind of relationship.
- In this graph we also have nodes representing datatype values (i.e., “literals”) such as strings, numbers, dates. These edges are sometimes called “attributes” of the node and are often used to represent the characteristics of the nodes.
- This simple, flexible data model has a lot of expressive power to represent complex situations, relationships, and other things of interest, while also being appropriately abstract, i.e., it does not expose very much implementation detail in the same way as, say, the relational data model used with SQL

- **Graph storage on disk**
  - Since a graph database is a schema-less database, we use fixed record lengths to persist data and follow offsets in these files to know how to fetch data to answer queries. The following table illustrates the fixed sizes Neo4j uses for the type of Java objects being stored:

| Store File | Record size | Contents |
|---|---|---|
| neostore.nodestore.db | 15 B | Nodes |
| neostore.relationshipstore.db | 34B | Relationships |
| neostore.propertystore.db | 41B | Properties for nodes and relationships |
| neostore.propertystore.db.strings | 128B | Values of string properties |
| neostore.propertystore.db.arrays | 128B | Values of array properties |
| Indexed Property | 1/3 * AVG(x) | Each index entry is approximately 1/3 of the average property value size |

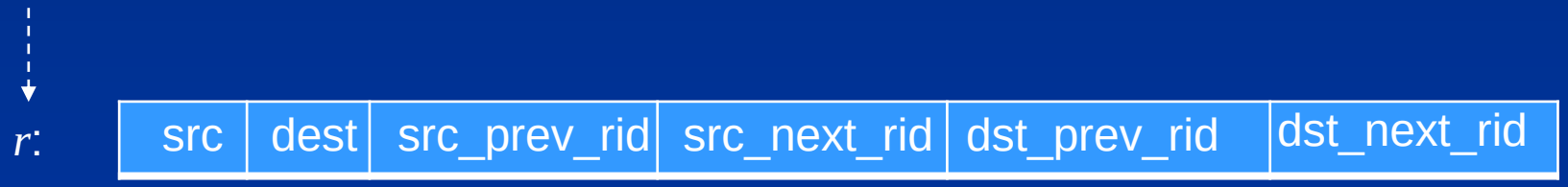
- **Graph storage on disk**
  - It all boils down to linked lists of fixed size records on disk.
  - Properties are stored as a linked list of property records, each holding key+value.
  - Each node/relationship references its first property record.
  - The Nodes also reference the first relation instance in its relationship chain.
  - Each Relationship references its start and end node. It also references the prev/next relationship record for the start/end node respectively.

- **Node storage on disk**
  - In the file for storing nodes, each node storage is of the same length.



- **Relation storage on disk**
  - In the file for storing relationships, each relationship storage is of the same length.

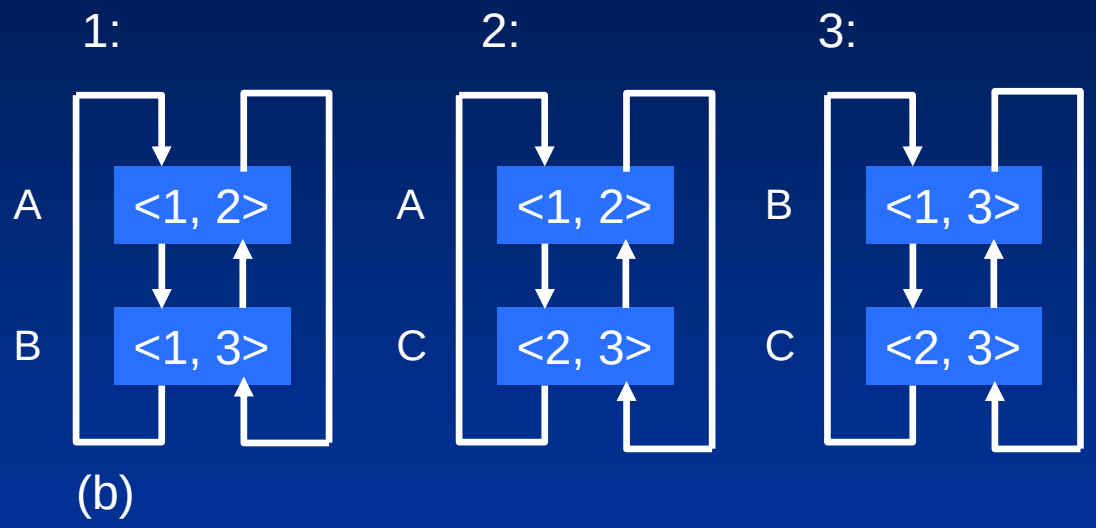
relationship address
in file (rid):



- **Relation storage on disk**
  - Example

| | src | dest |
|---|---|---|
| A | 1 | 2 |
| B | 1 | 3 |
| C | 2 | 3 |

(a)




(b)

Node:

| | First-Rid |
|---|---|
| 1: | A |
| 2: | A |
| 3: | B |

(c)

Relation:

| | src | dest | src_prev_rid | src_next_rid | dst_prev_rid | dst_next_rid |
|---|---|---|---|---|---|---|
| A | 1 | 2 | B | B | C | C |
| B | 1 | 3 | A | A | C | C |
| C | 2 | 3 | A | A | B | B |

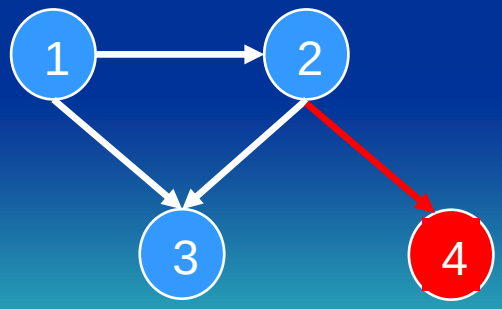
(d)

Figure 3

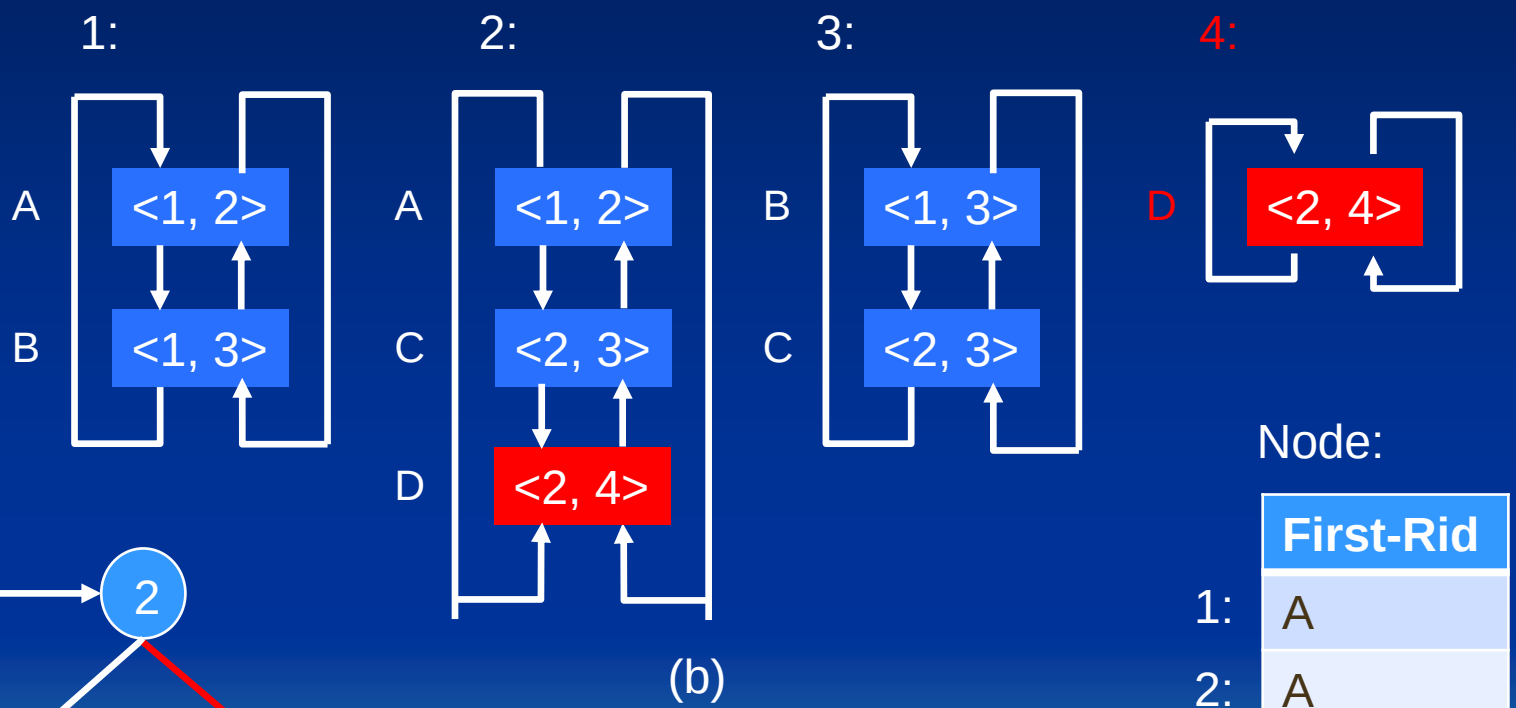
- Figure 3 (b) is a conceptual model. It lists all of the edges associated with each node and organizes them as doubly linked lists. that an edge is associated with a node if the node is either the source or the destination of that edge. For example, edge A <1,2> is associated with both node 1 and node 2.
- Figure 3 (c) is the actual nodes storage. Each record stores the first relationship ID (first_rid) of a node. For example, the first relationship for both node 1 and 2 is A.
- Figure 3 (d) is the actual relationships storage. Each record stores the source (src) And destination (dest) of a relationship. In addition, it also stores the previous and next relationship IDs for both the source and destination nodes (src_prev_rid, src_next_rid, dst_prev_rid, dst_next_rid). This essentially preserves the conceptual doubly linked lists for both the source and destination but only stores a given edge once.

- **Retrieve a graph database**
  - Say you want to retrieve all the outbound relationships of node 2. See the blue arrows in Figure 3 as an illustration. You first go to the nodes storage and see that the first_rid of node 2 is A.
  - Then you go to the relationships storage to look up A. A isn't an outbound relationship of 2 because 2 is the dst for A. But that's fine. You follow the dst_next_rid in A (because 2 is the destination in relationship A). That points to C. C is a outbound relationship for 2 (2 is the source in C) . You continue to go to the src_next_rid, which loops back to A. Then you know you've exhausted all of node 2's relationship.

- **Add a new node**
  - Now let's say you want to add a new node 4 and make 2 follow 4.
  - See Figure 4 for an overview of the change. Again, let's walk through that.




(a)




(b)

Node:

| | First-Rid |
|---|---|
| 1: | A |
| 2: | A |
| 3: | B |
| 4: | D |

(c)

Figure 4

Relation:

| | src | dest | src_prev_rid | src_next_rid | dst_prev_rid | dst_next_rid |
|---|---|---|---|---|---|---|
| A | 1 | 2 | B | B | ~~C~~ D | C |
| B | 1 | 3 | A | A | C | C |
| C | 2 | 3 | A | ~~A~~ D | B | B |
| D | 2 | 4 | C | A | D | D |

- A new record is added to the nodes storage to store the first_rid of node 4, which is D. In the relationships storage, a new record D is inserted. Its src and dst are 2 and 4, respectively.
- Its dst_prev_rid and dst_next_rid are trivial because its dst node 4 has has only one relationship, which is D itself. The slightly tricky part is to update the doubly linked list of relationships for node 2. I've highlighted them in red. It should be  fairly straightforward to see how the red part for node 2 in Figure 4 (b) is reflected in the changed relation.