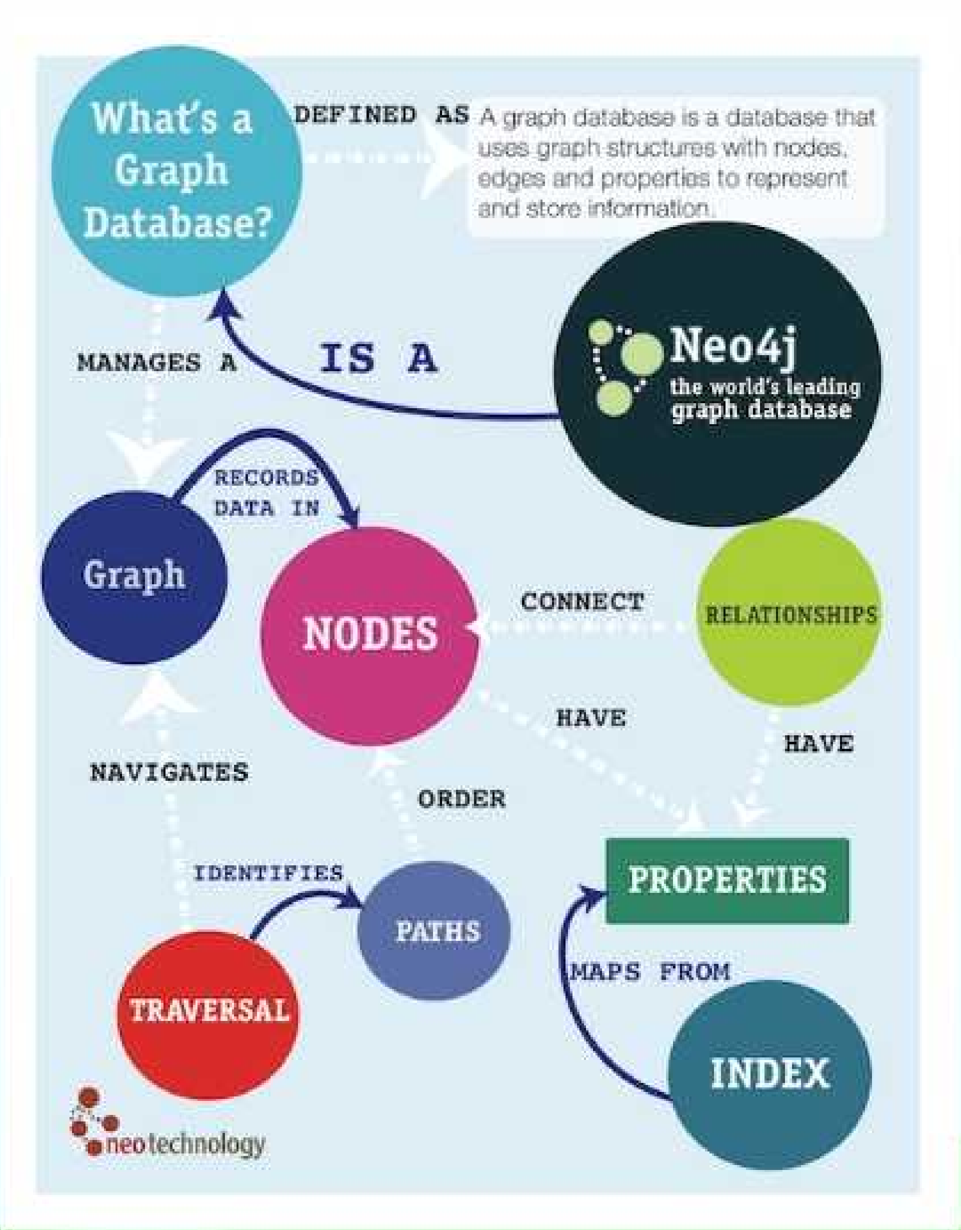
Neo4j

Overview

- Neo4j Introduction
- Neo4j Data Model
- Cypher Language



A graph database is a database that uses graph structures with nodes, edges and properties to represent and store information.

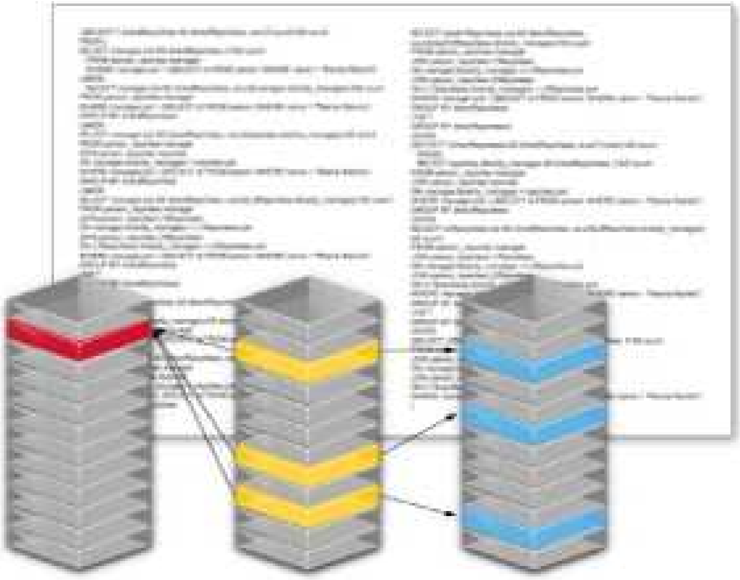
Relational Databases can't handle Relationships

1 **Wrong Model**
They cannot model or store relationships without complexity

2 **Degraded Performance**
Speed plummets as data grows and as the number of joins grows

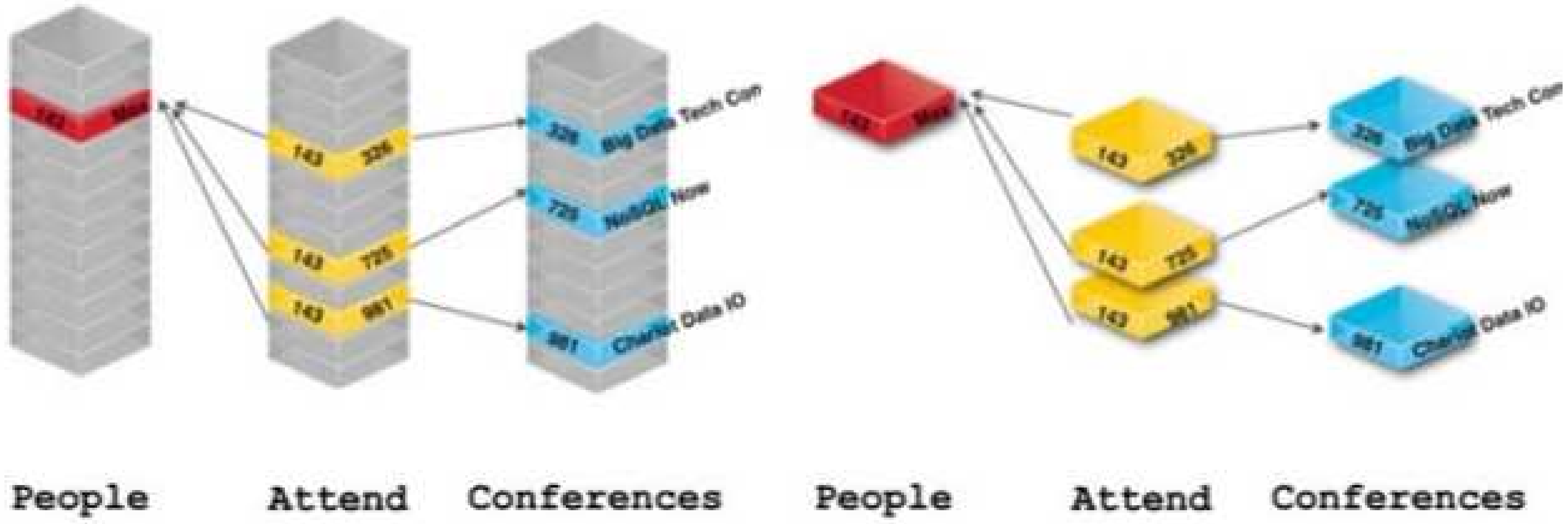
3 **Wrong Language**
SQL was built with Set Theory in mind, not Graph Theory

4 **Not Flexible**
New types of data and relationships require schema redesign



Same Data, Different Layout

No more Tables, no more Foreign Keys, no more Joins



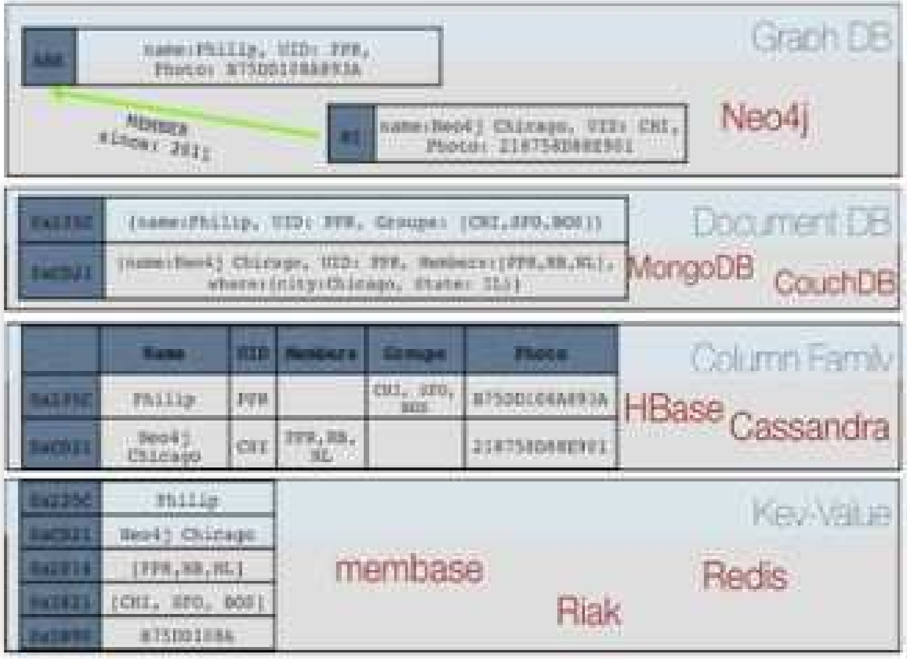
NoSQL Databases can't handle Relationships

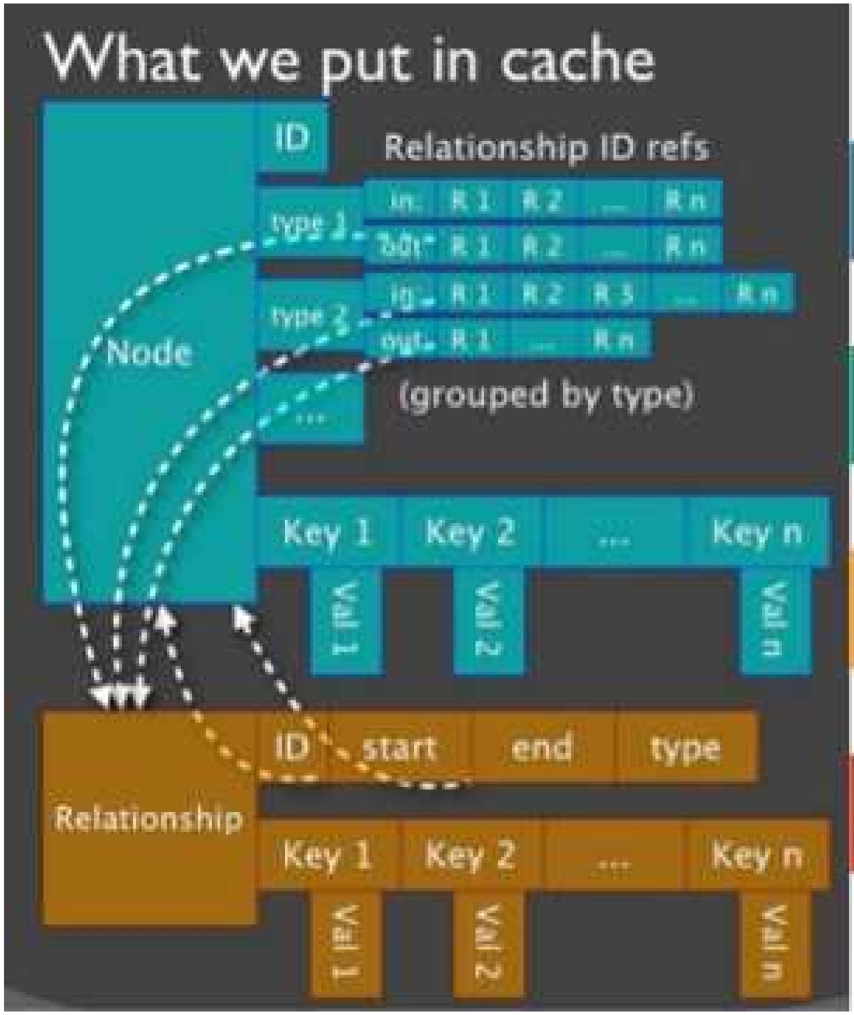
1 **Wrong Model**
They cannot model or store relationships without complexity

2 **Degraded Performance**
Speed plummets as you try to join data together in the application

3 **Wrong Languages**
Lots of wacky "almost sql" languages terrible at "joins"

4 **Not ACID**
Eventually Consistent means Eventually Corrupt





Neo4j Secret Sauce

1 Pointers instead of Lookups

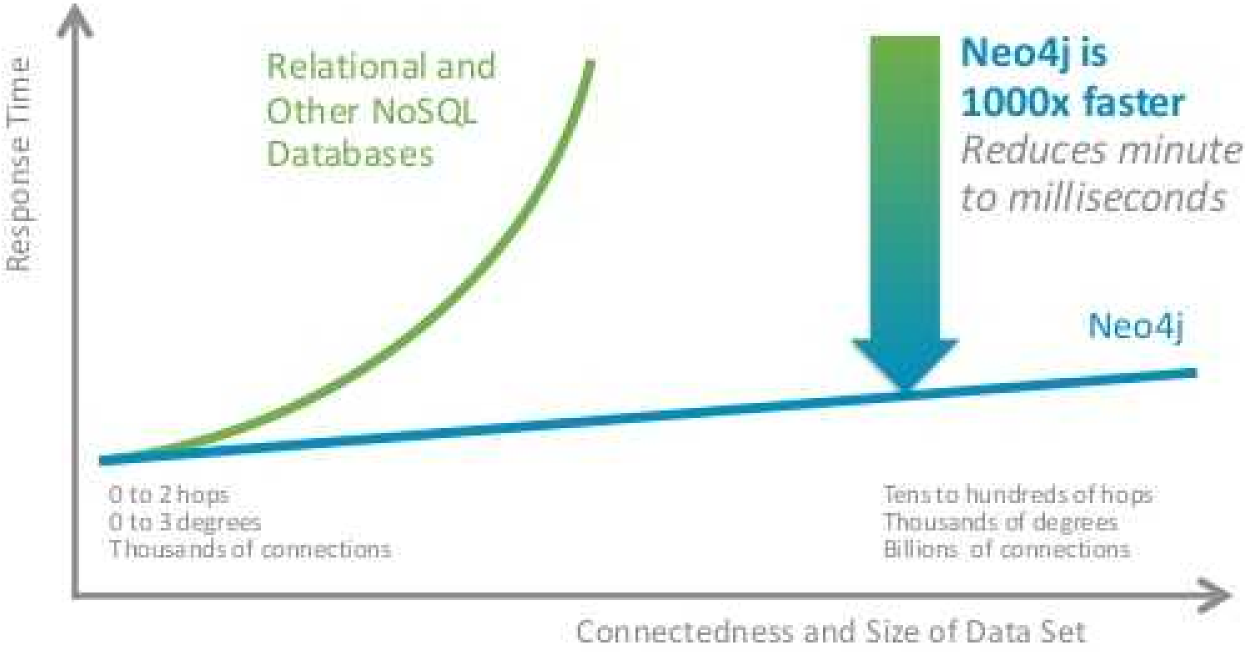
2 Fixed Sized Records

3 "Joins" on Creation

4 Spin Spin Spin through this data structure

Real Time Query Performance

Remains steady as database grows



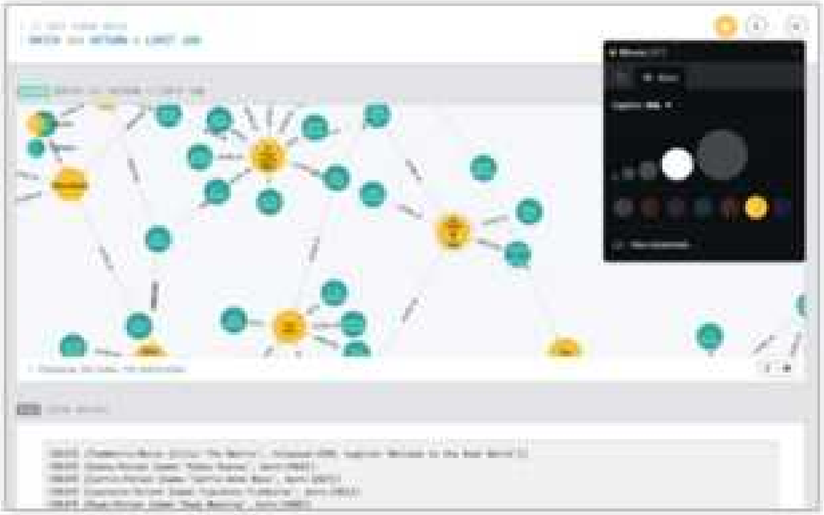
Reimagine your Data as a Graph

1 **Right Model**
Graphs simplify how you think

2 **Better Performance**
Query relationships in real time

3 **Right Language**
Cypher was purpose built for Graphs

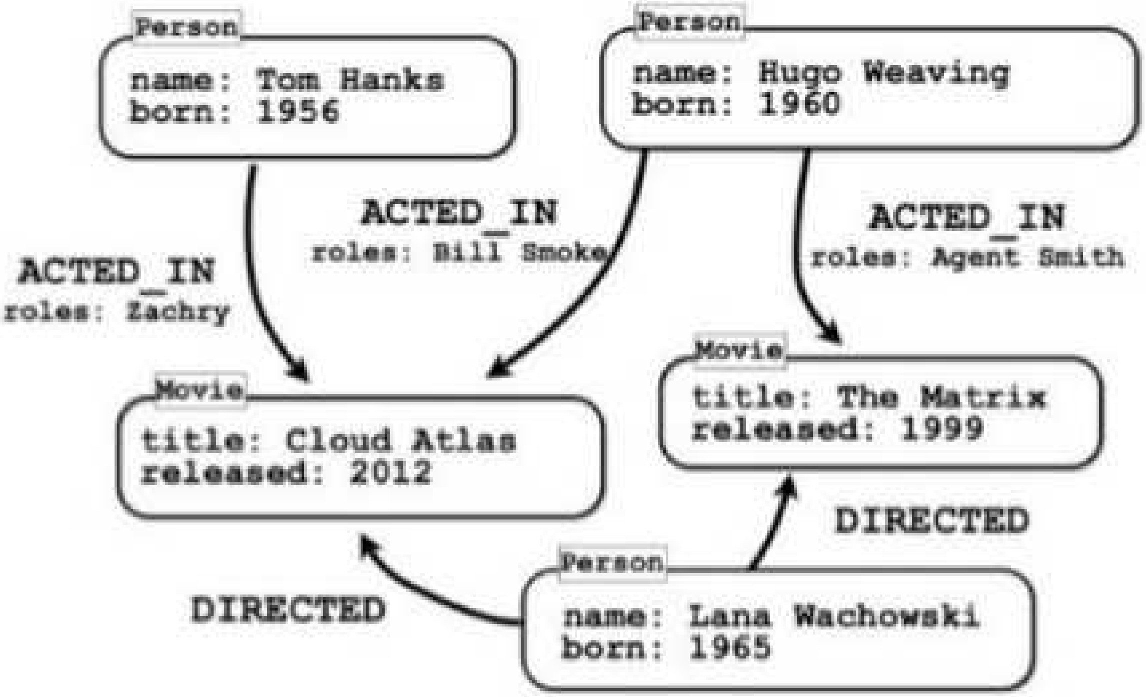
4 **Flexible and Consistent**
Evolve your schema seamlessly while keeping transactions



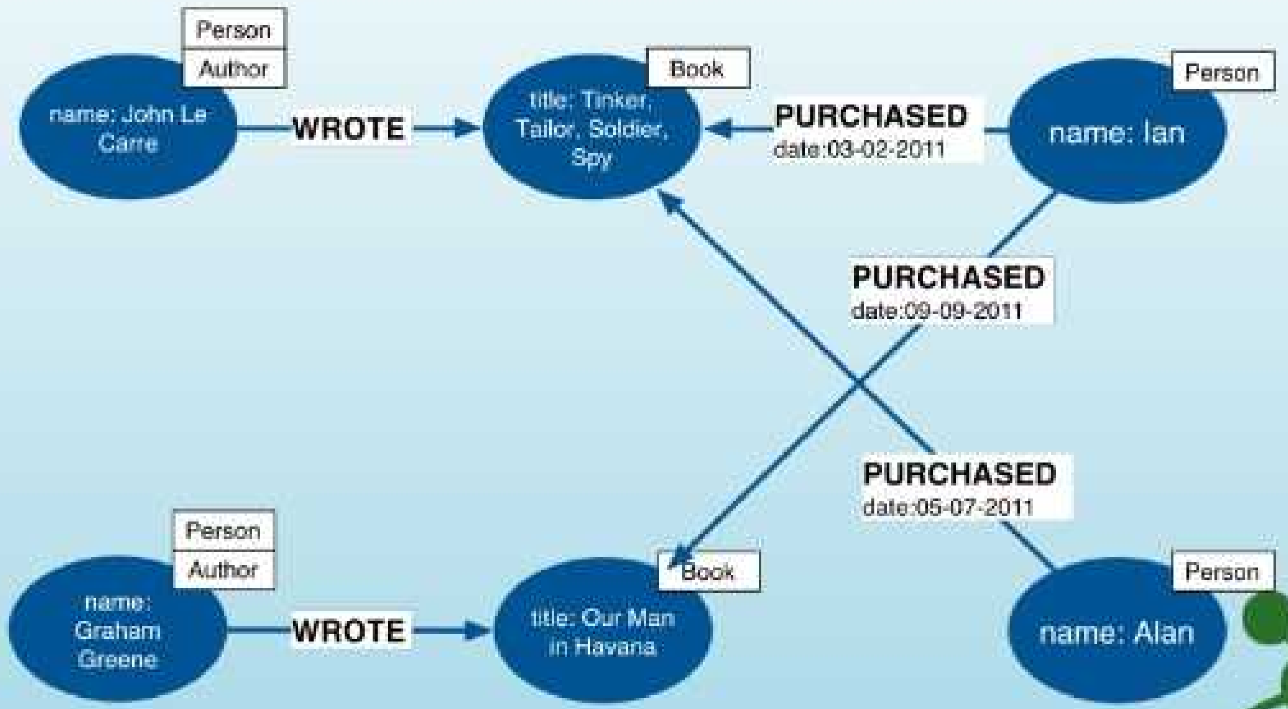
Agile, High Performance and Scalable without Sacrifice

Some Models are Easy

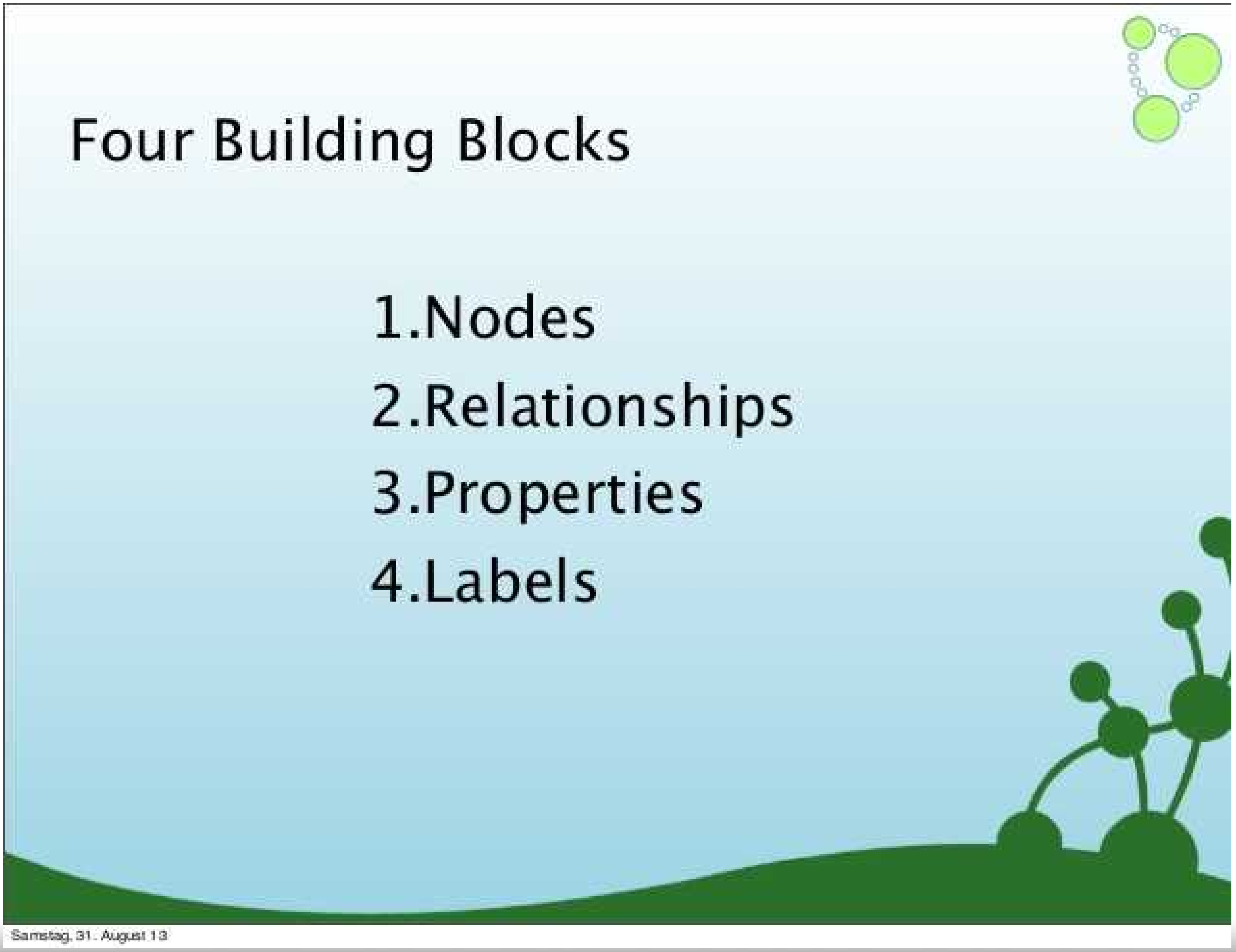
Movie Property Graph



Property Graph Data Model



Samstag, 31. August 13

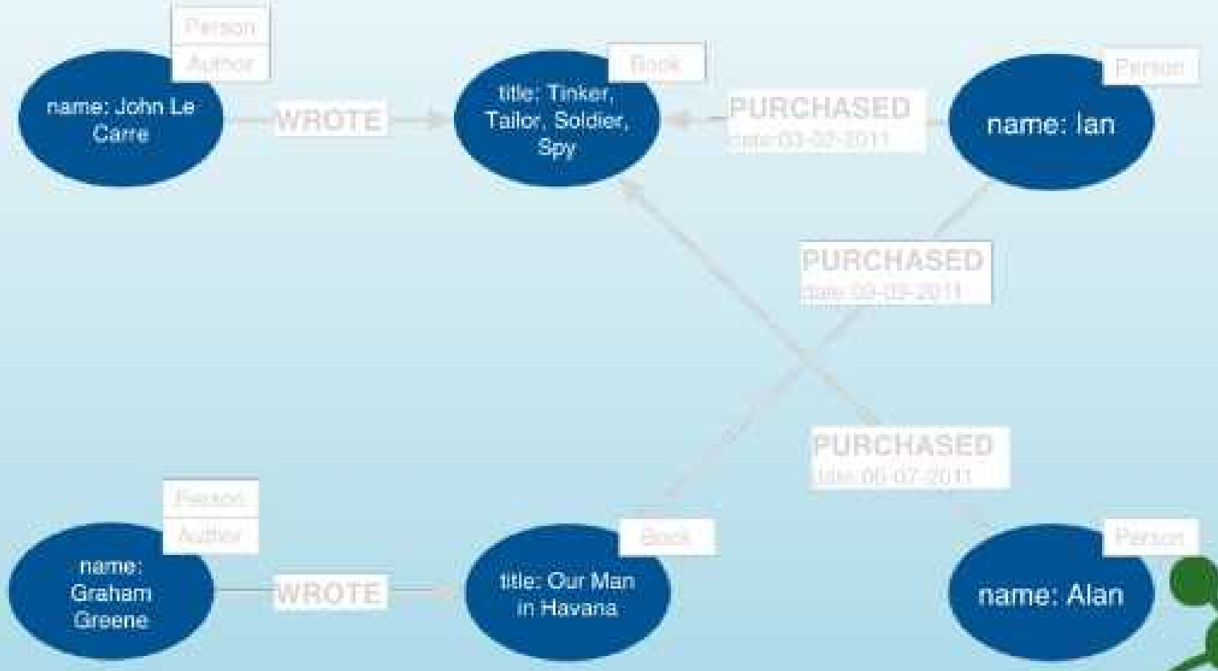


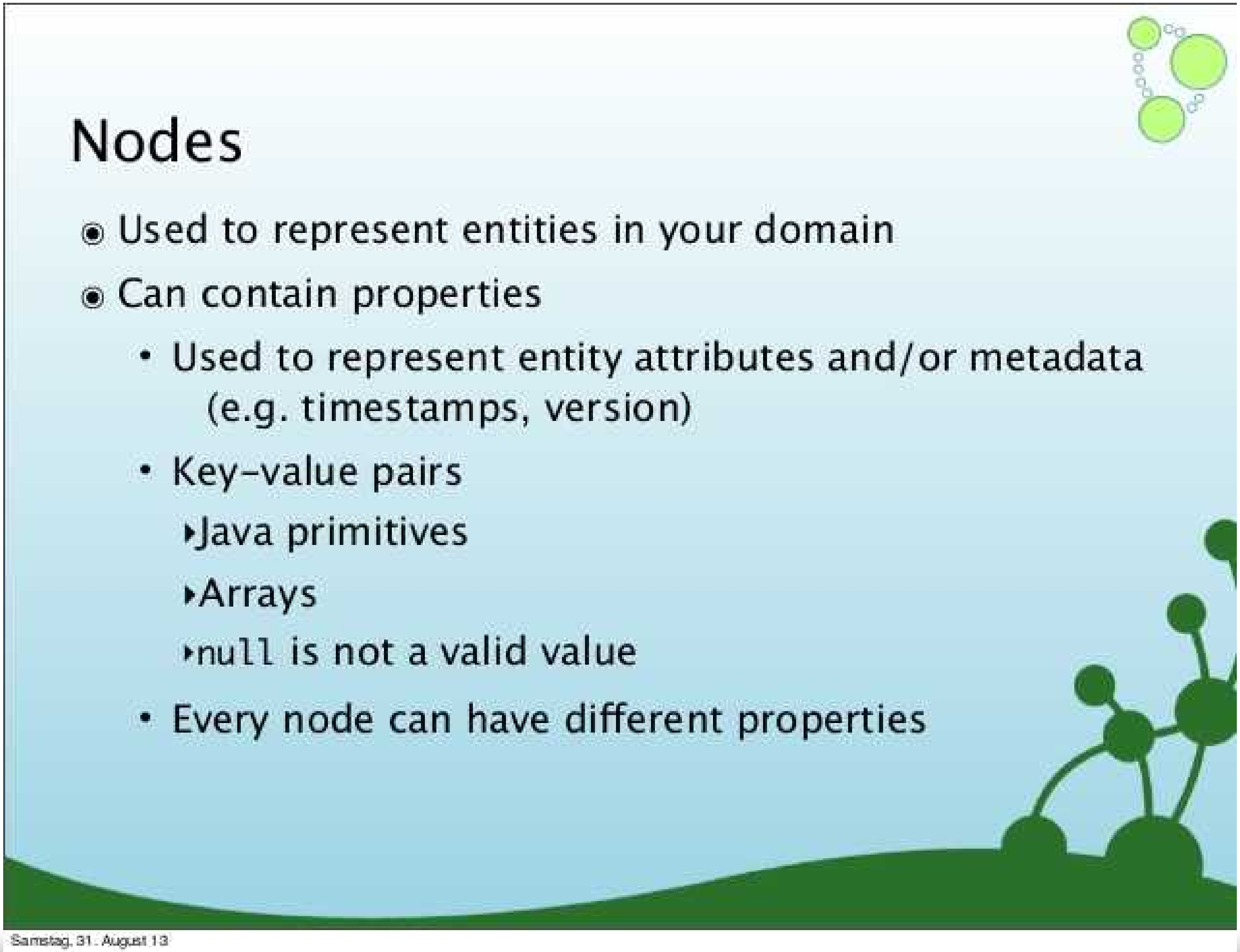
Four Building Blocks

1. Nodes
2. Relationships
3. Properties
4. Labels

Samstag, 31. August 13

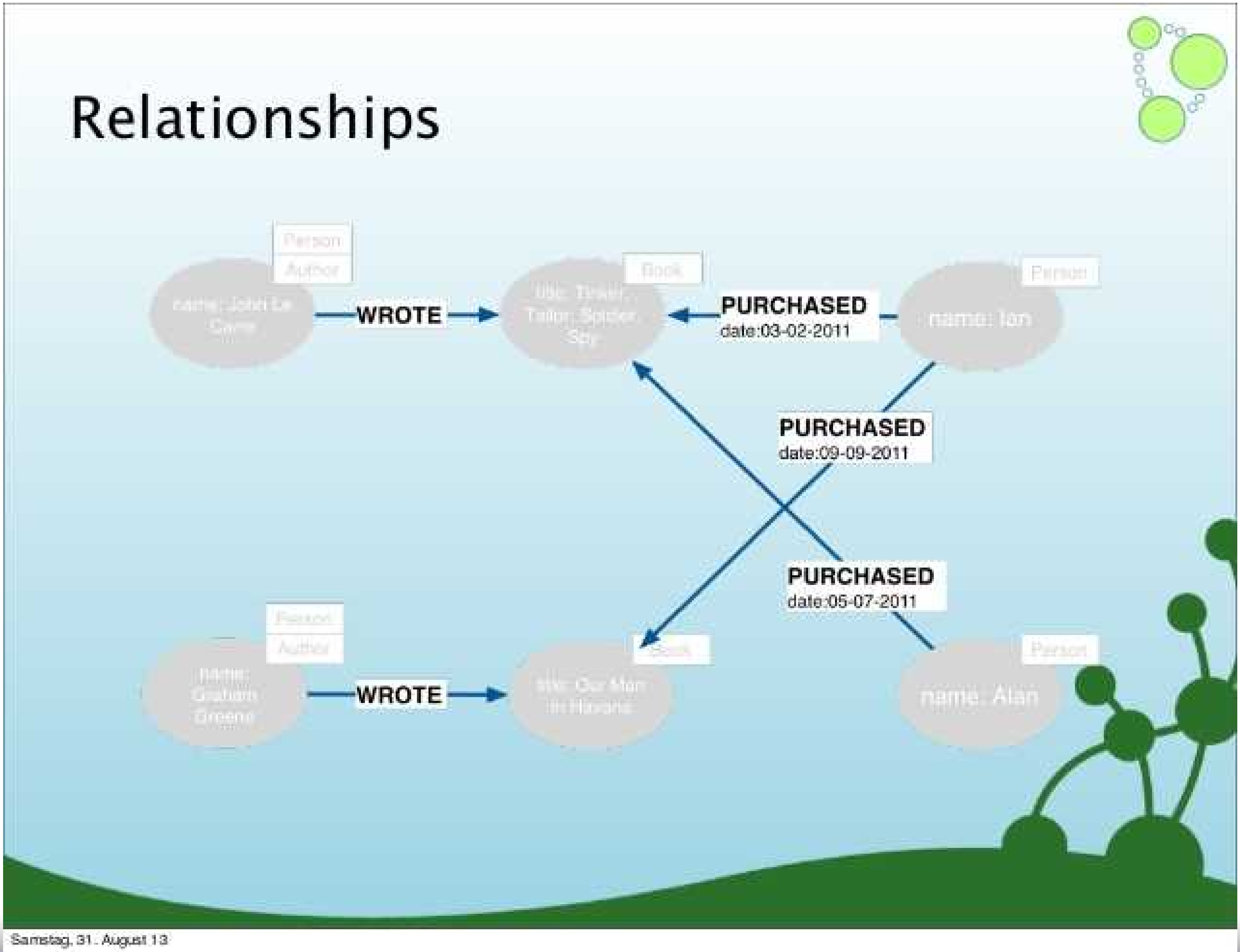
Nodes


Person
Author
name: John Le Carre
WROTE
Book
title: Tinker, Tailor, Soldier, Spy
PURCHASED
date:03-02-2011
Person
name: Ian
PURCHASED
date 09-09-2011
PURCHASED
date:05-07-2011
Person
Author
name: Graham Greene
WROTE
Book
title: Our Man in Havana
Person
name: Alan



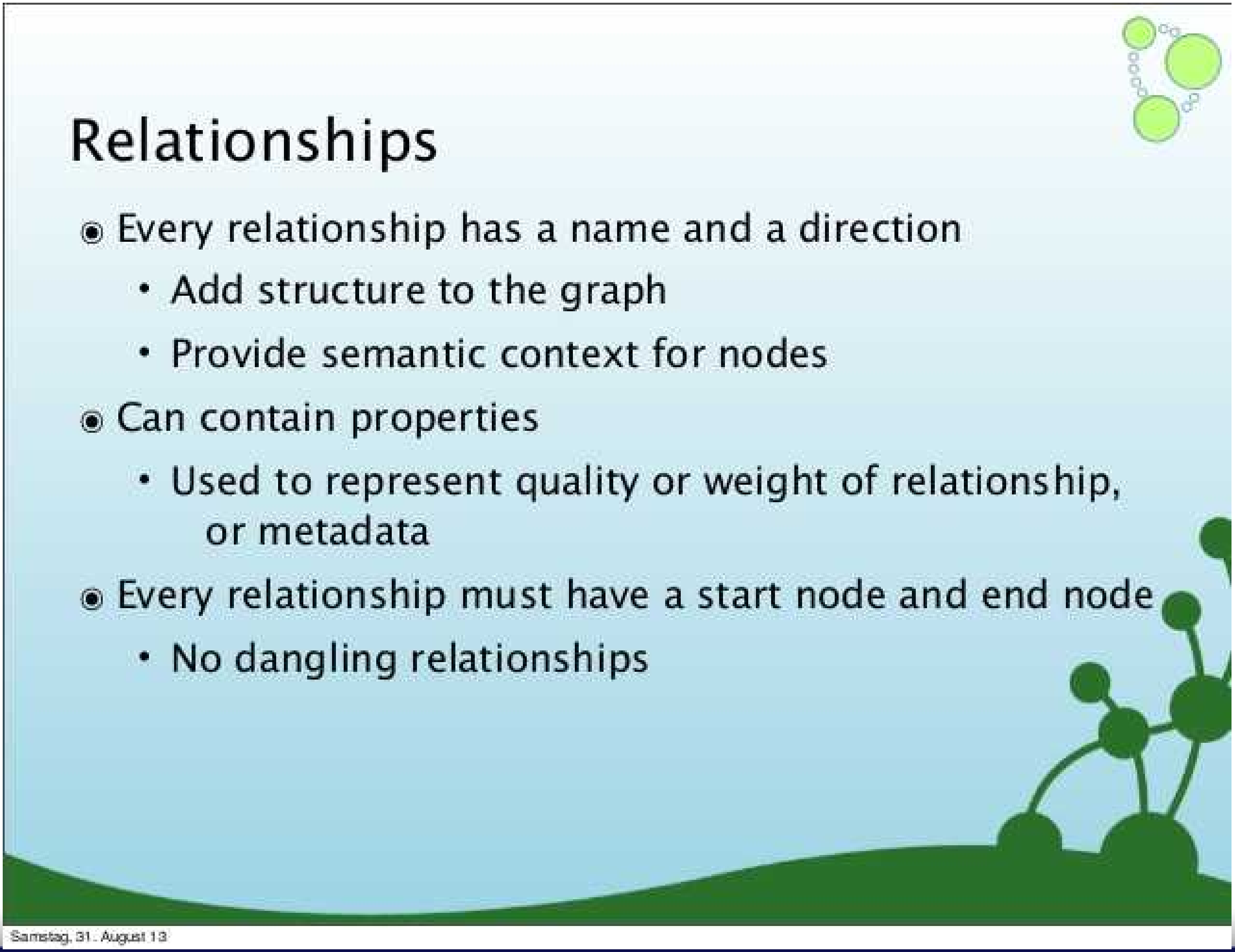
Nodes

- Used to represent entities in your domain
- Can contain properties
  - Used to represent entity attributes and/or metadata (e.g. timestamps, version)
  - Key-value pairs
    - Java primitives
    - Arrays
    - `null` is not a valid value
  - Every node can have different properties



Relationships

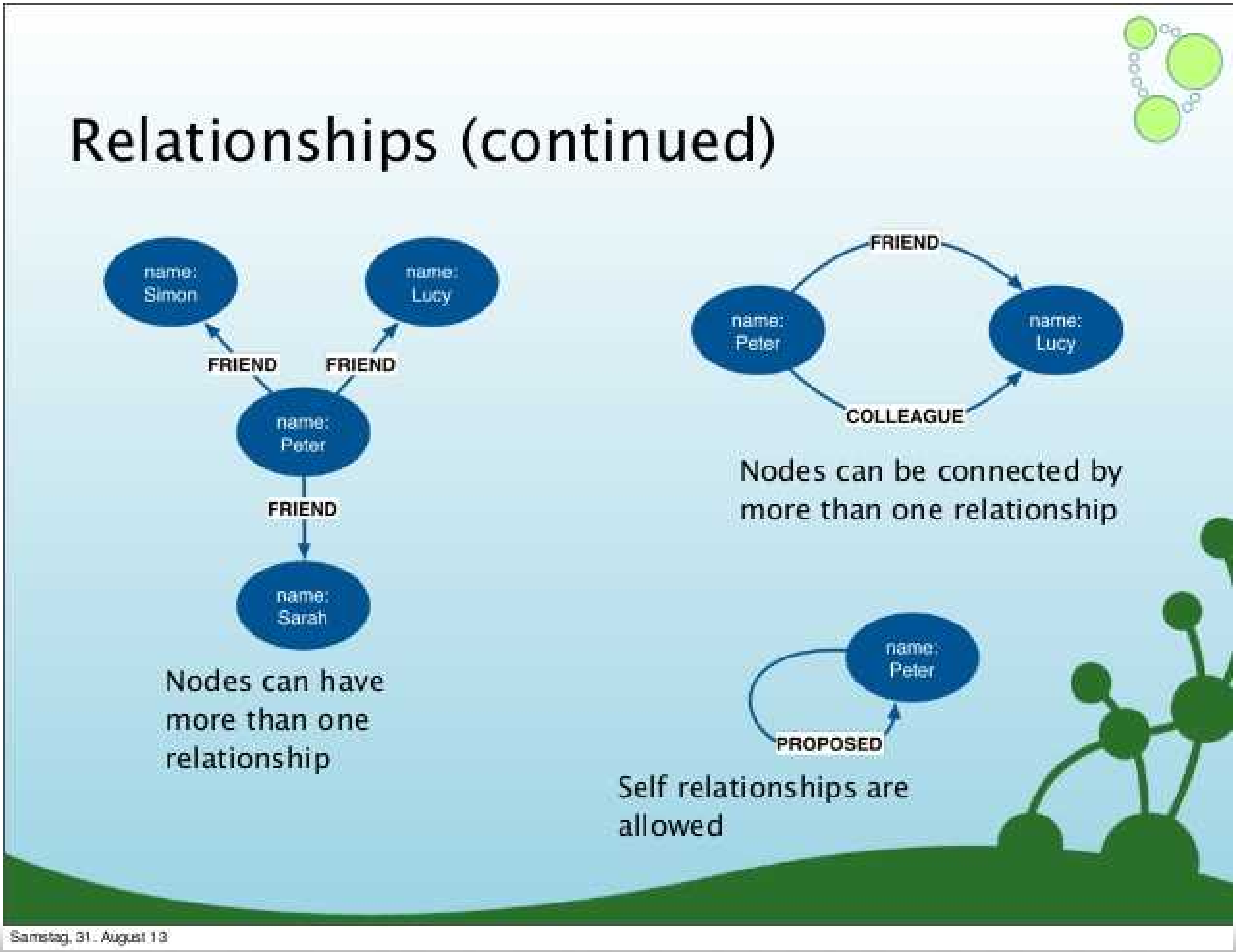
Samstag, 31. August 13



Relationships

- Every relationship has a name and a direction
  - Add structure to the graph
  - Provide semantic context for nodes
- Can contain properties
  - Used to represent quality or weight of relationship, or metadata
- Every relationship must have a start node and end node
  - No dangling relationships

Samstag, 31. August 13



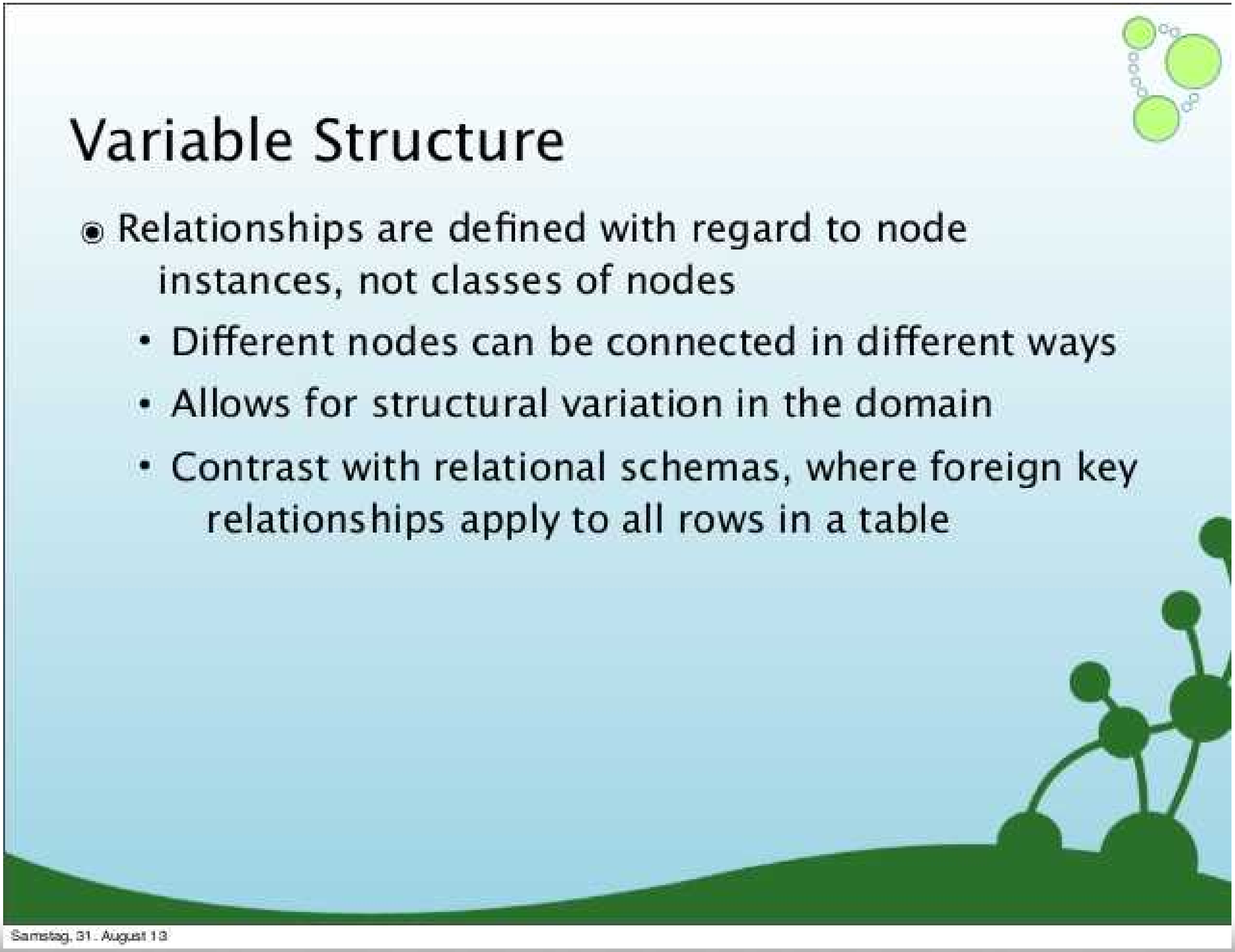
Relationships (continued)

Nodes can have more than one relationship

Nodes can be connected by more than one relationship

Self relationships are allowed

Samstag, 31. August 13

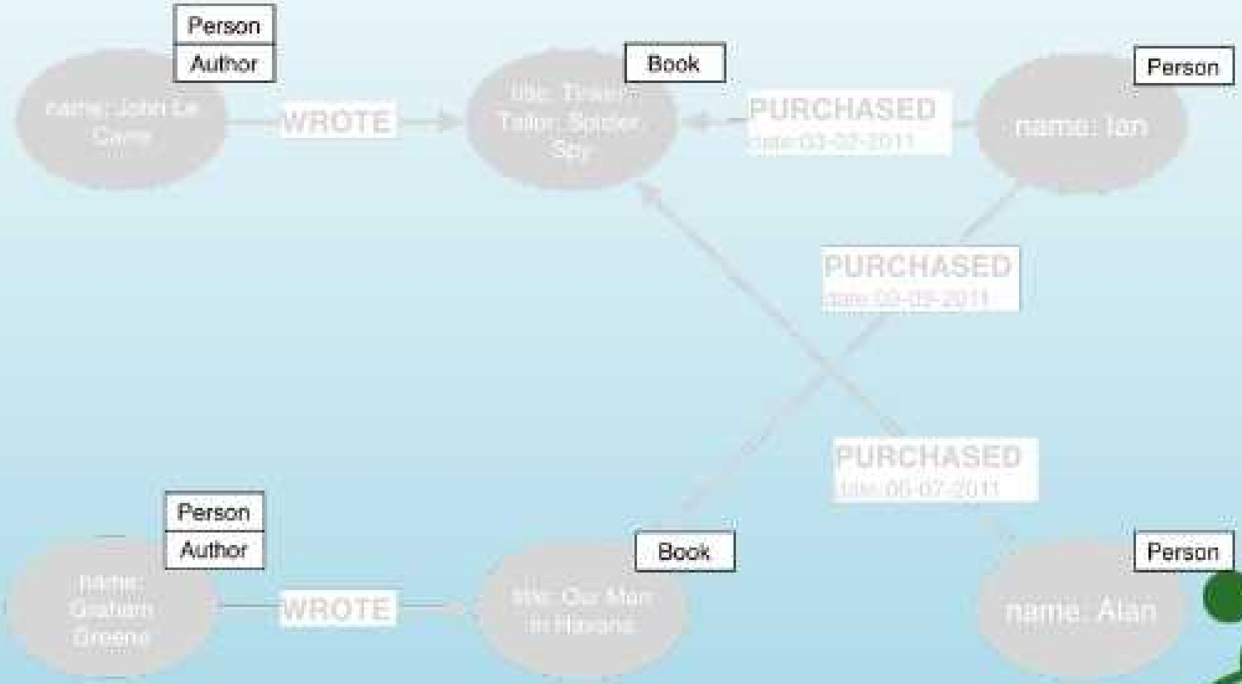


Variable Structure

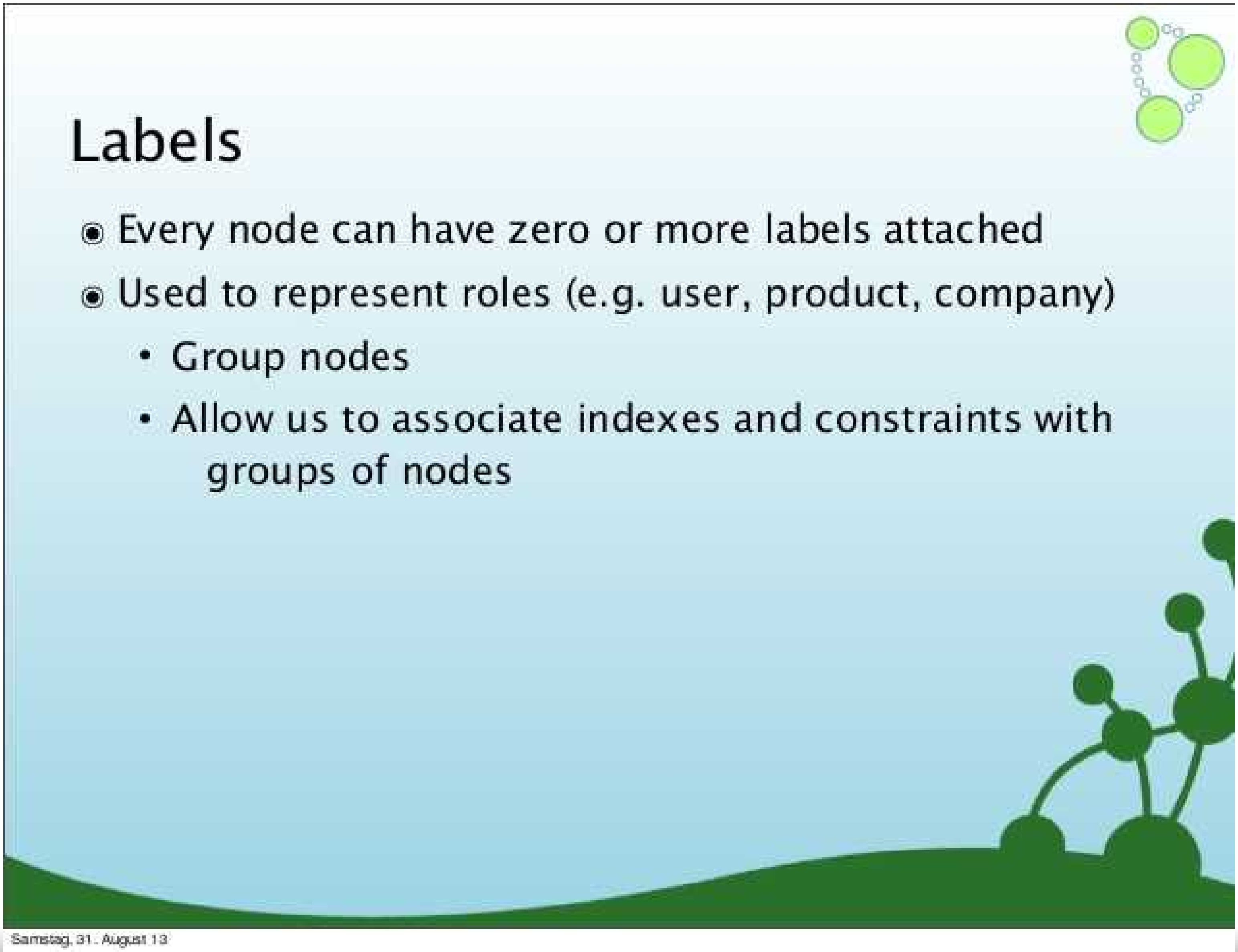
- Relationships are defined with regard to node instances, not classes of nodes
  - Different nodes can be connected in different ways
  - Allows for structural variation in the domain
  - Contrast with relational schemas, where foreign key relationships apply to all rows in a table

Samstag, 31. August 13

Labels

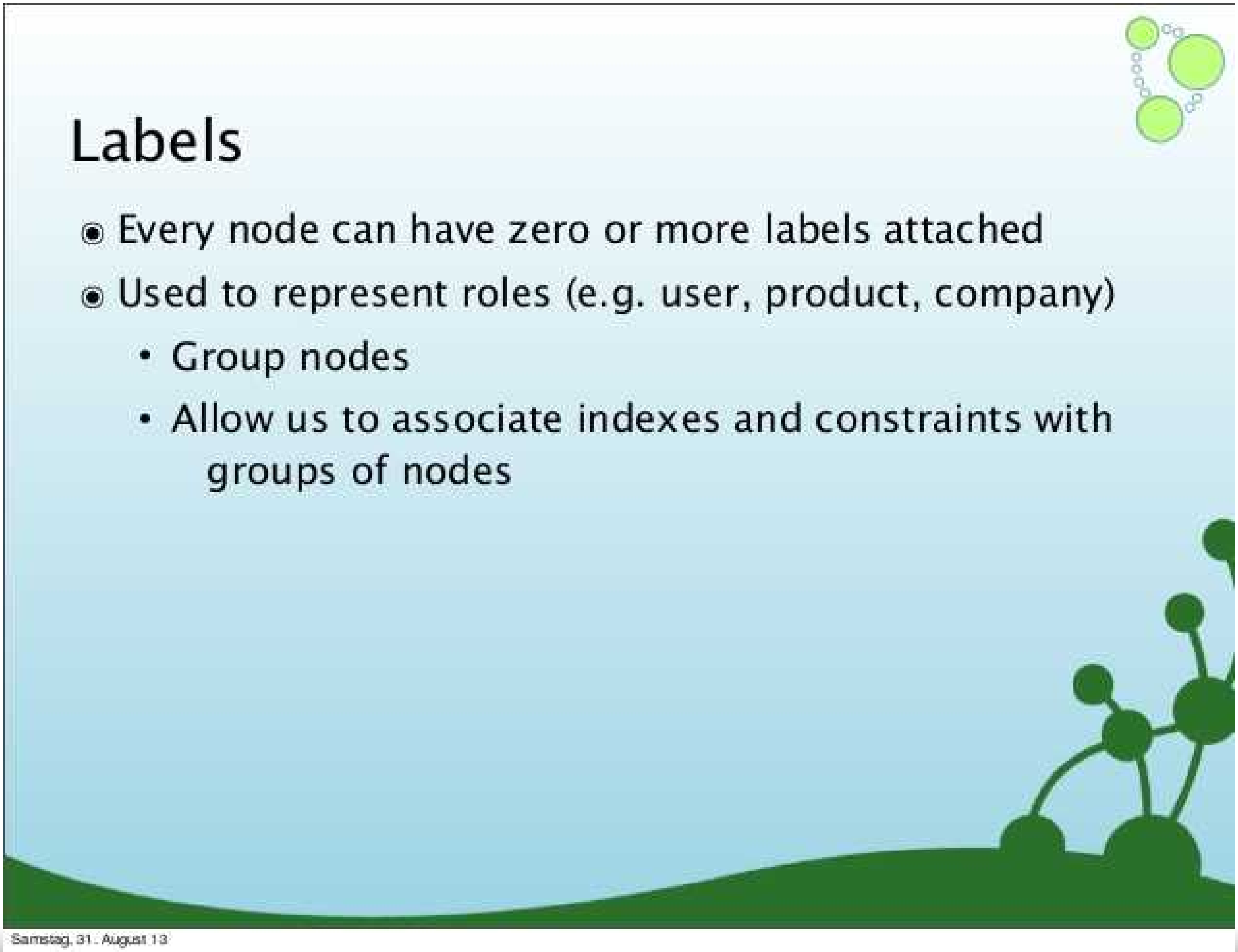


Samstag, 31. August 13



Labels

- Every node can have zero or more labels attached
- Used to represent roles (e.g. user, product, company)
  - Group nodes
  - Allow us to associate indexes and constraints with groups of nodes

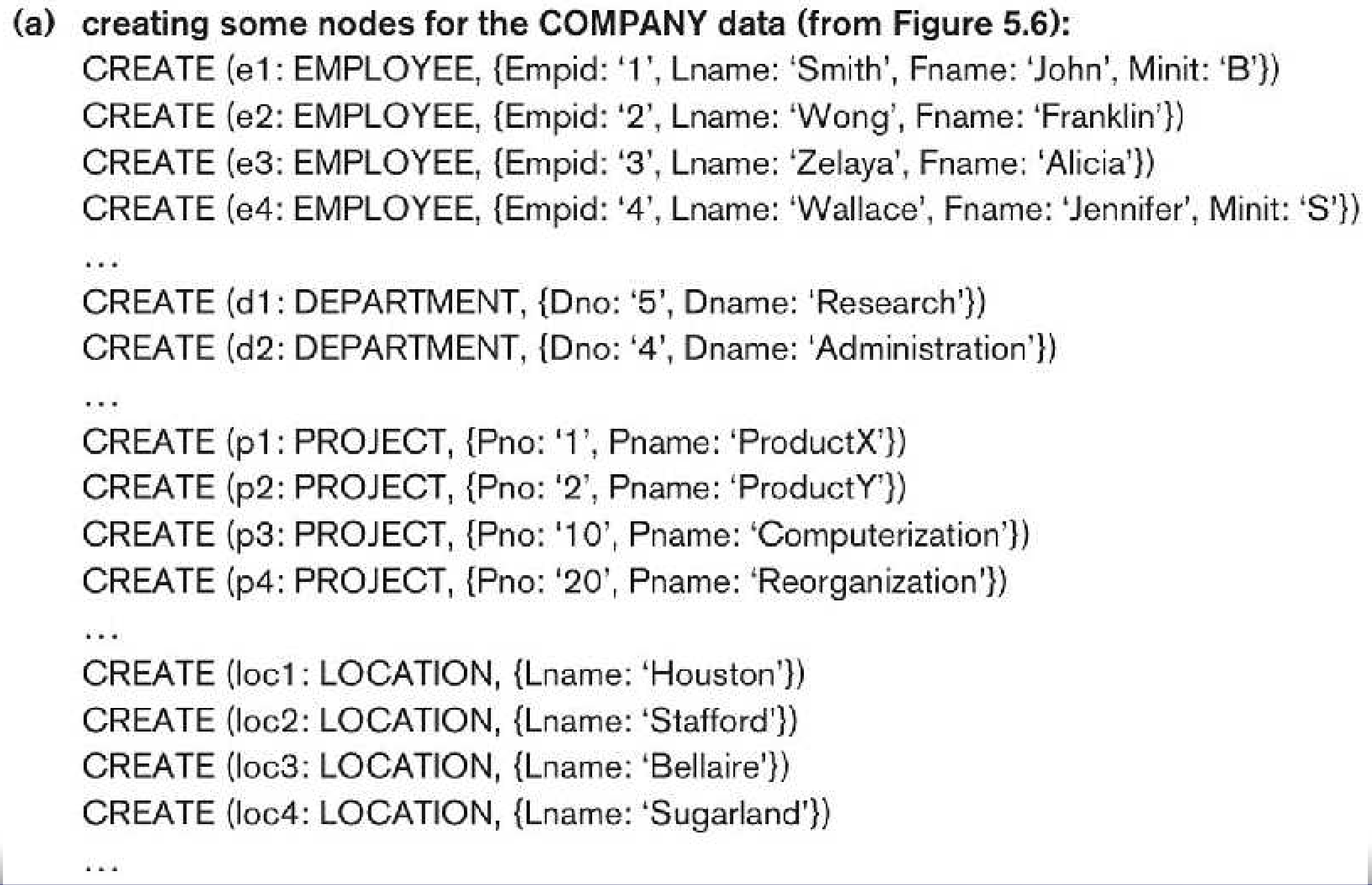


Labels

- ◉ Every node can have zero or more labels attached
- ◉ Used to represent roles (e.g. user, product, company)
  - Group nodes
  - Allow us to associate indexes and constraints with groups of nodes

Samstag, 31. August 13

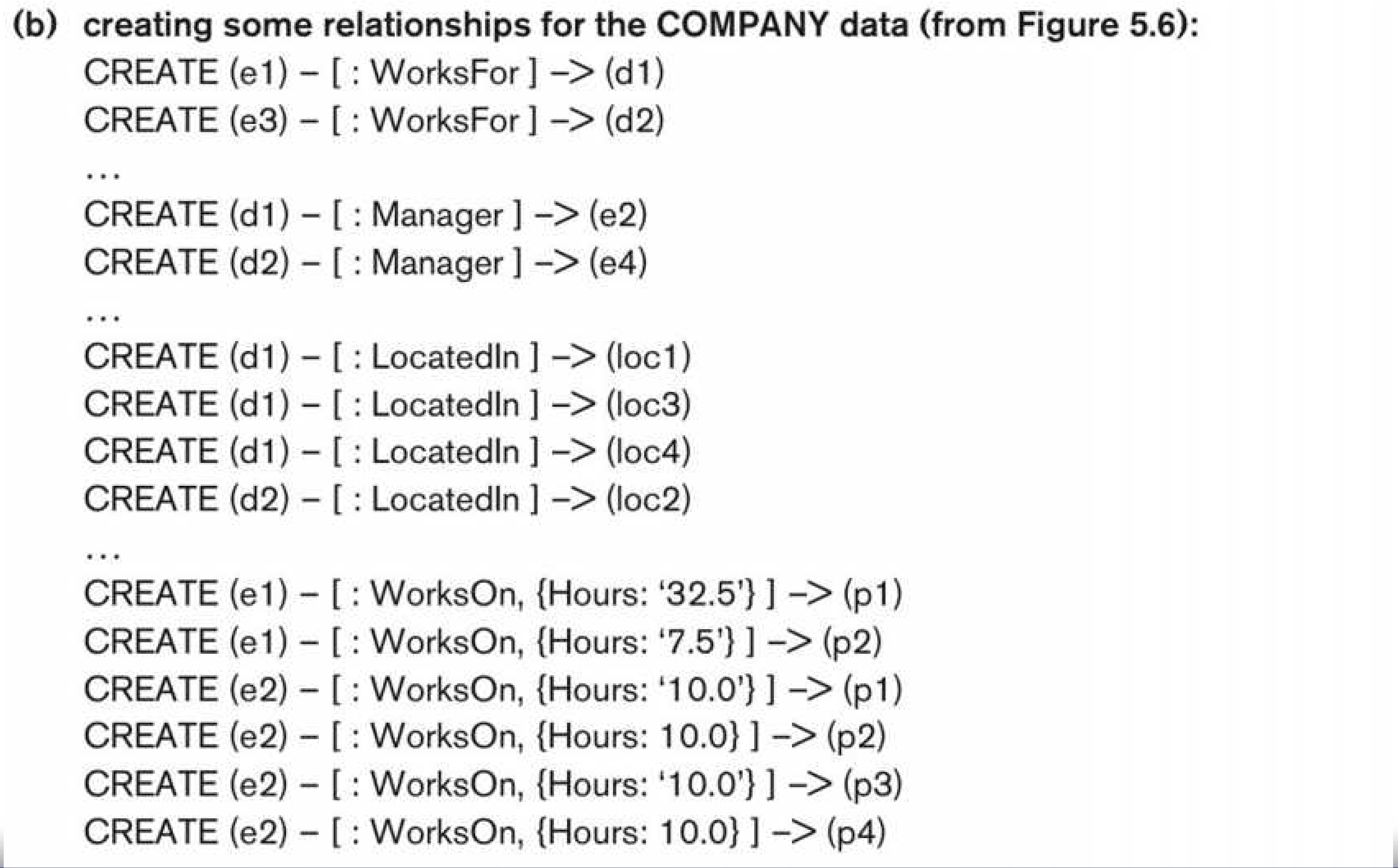
Examples in Neo4j using the Cypher language. Creating some nodes.



(a) creating some nodes for the COMPANY data (from Figure 5.6):

```
CREATE (e1: EMPLOYEE, {Empid: ‘1’, Lname: ‘Smith’, Fname: ‘John’, Minit: ‘B’})
CREATE (e2: EMPLOYEE, {Empid: ‘2’, Lname: ‘Wong’, Fname: ‘Franklin’})
CREATE (e3: EMPLOYEE, {Empid: ‘3’, Lname: ‘Zelaya’, Fname: ‘Alicia’})
CREATE (e4: EMPLOYEE, {Empid: ‘4’, Lname: ‘Wallace’, Fname: ‘Jennifer’, Minit: ‘S’})
...
CREATE (d1: DEPARTMENT, {Dno: ‘5’, Dname: ‘Research’})
CREATE (d2: DEPARTMENT, {Dno: ‘4’, Dname: ‘Administration’})
...
CREATE (p1: PROJECT, {Pno: ‘1’, Pname: ‘ProductX’})
CREATE (p2: PROJECT, {Pno: ‘2’, Pname: ‘ProductY’})
CREATE (p3: PROJECT, {Pno: ‘10’, Pname: ‘Computerization’})
CREATE (p4: PROJECT, {Pno: ‘20’, Pname: ‘Reorganization’})
...
CREATE (loc1: LOCATION, {Lname: ‘Houston’})
CREATE (loc2: LOCATION, {Lname: ‘Stafford’})
CREATE (loc3: LOCATION, {Lname: ‘Bellaire’})
CREATE (loc4: LOCATION, {Lname: ‘Sugarland’})
...
```

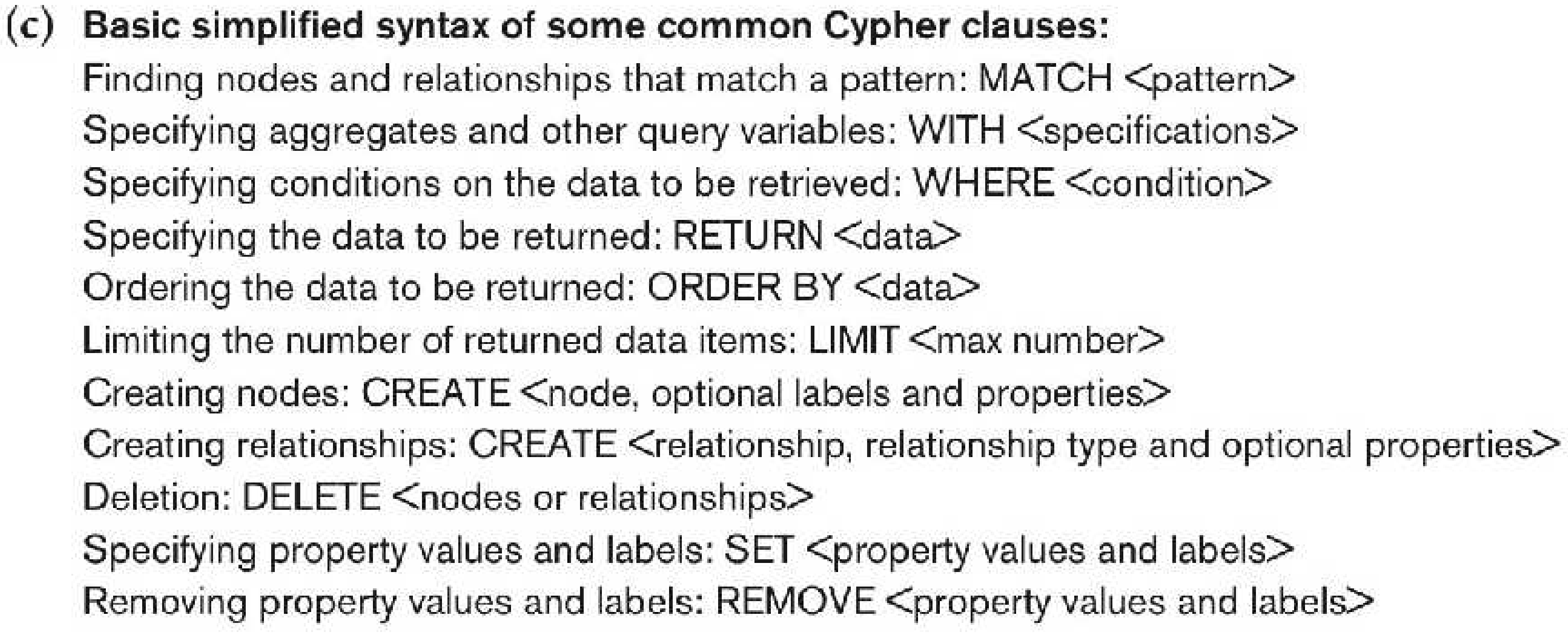
Examples in Neo4j using the Cypher language.
Creating some relationships.



(b) creating some relationships for the COMPANY data (from Figure 5.6):

```
CREATE (e1) – [ : WorksFor ] –> (d1)
CREATE (e3) – [ : WorksFor ] –> (d2)
…
CREATE (d1) – [ : Manager ] –> (e2)
CREATE (d2) – [ : Manager ] –> (e4)
…
CREATE (d1) – [ : LocatedIn ] –> (loc1)
CREATE (d1) – [ : LocatedIn ] –> (loc3)
CREATE (d1) – [ : LocatedIn ] –> (loc4)
CREATE (d2) – [ : LocatedIn ] –> (loc2)
…
CREATE (e1) – [ : WorksOn, {Hours: ‘32.5’} ] –> (p1)
CREATE (e1) – [ : WorksOn, {Hours: ‘7.5’} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: ‘10.0’} ] –> (p1)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: ‘10.0’} ] –> (p3)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p4)
```

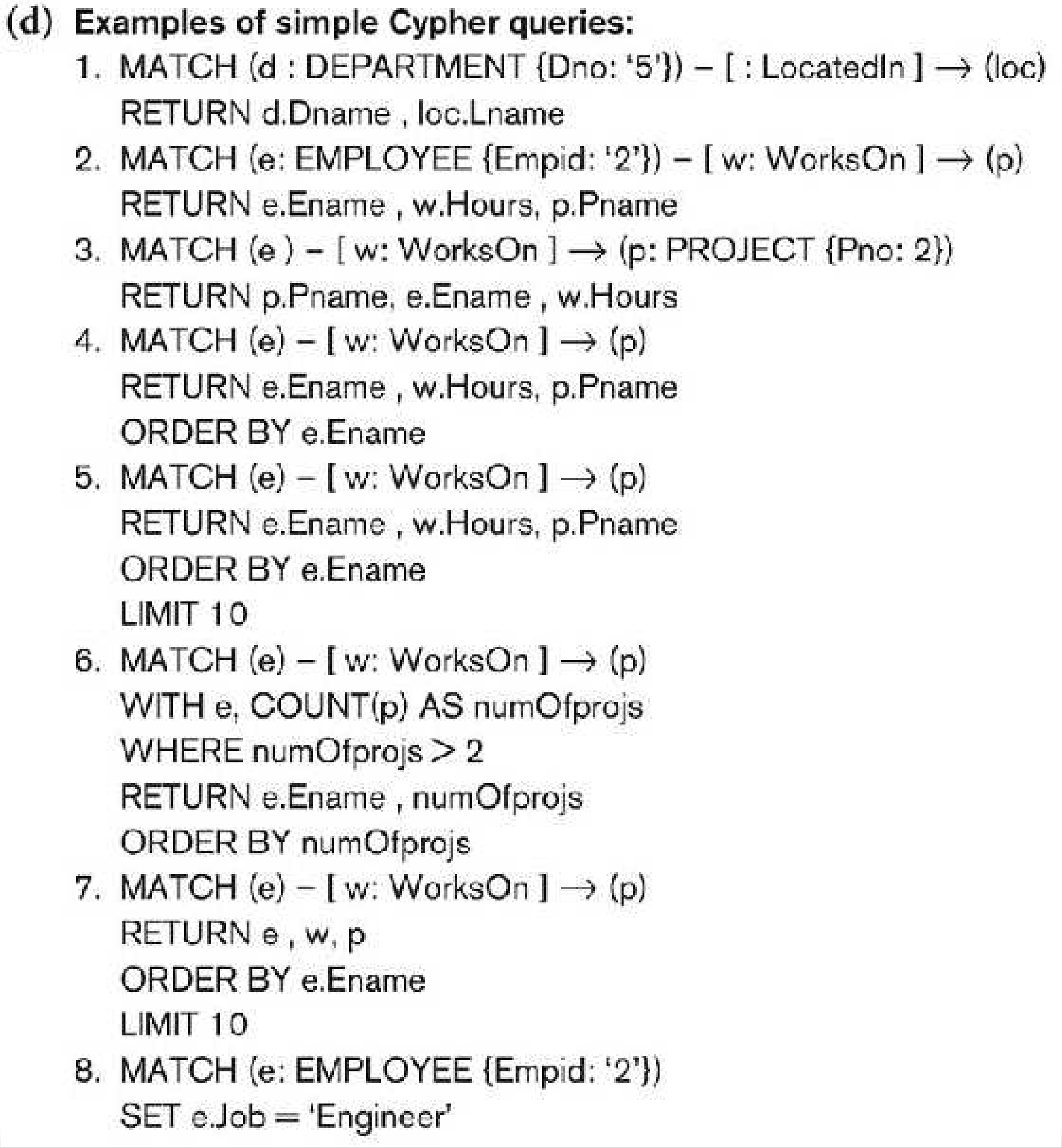
Examples in Neo4j using the Cypher language.
Basic syntax of Cypher queries.



(c) **Basic simplified syntax of some common Cypher clauses:**

Finding nodes and relationships that match a pattern: MATCH <pattern>
Specifying aggregates and other query variables: WITH <specifications>
Specifying conditions on the data to be retrieved: WHERE <condition>
Specifying the data to be returned: RETURN <data>
Ordering the data to be returned: ORDER BY <data>
Limiting the number of returned data items: LIMIT <max number>
Creating nodes: CREATE <node, optional labels and properties>
Creating relationships: CREATE <relationship, relationship type and optional properties>
Deletion: DELETE <nodes or relationships>
Specifying property values and labels: SET <property values and labels>
Removing property values and labels: REMOVE <property values and labels>

Examples in Neo4j using the Cypher language. Examples of Cypher queries.



(d) Examples of simple Cypher queries:

```
1. MATCH (d : DEPARTMENT {Dno: '5'}) – [ : LocatedIn ] → (loc)
   RETURN d.Dname , loc.Lname
2. MATCH (e: EMPLOYEE {Empid: '2'}) – [ w: WorksOn ] → (p)
   RETURN e.Ename , w.Hours, p.Pname
3. MATCH (e ) – [ w: WorksOn ] → (p: PROJECT {Pno: 2})
   RETURN p.Pname, e.Ename , w.Hours
4. MATCH (e) – [ w: WorksOn ] → (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
5. MATCH (e) – [ w: WorksOn ] → (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
   LIMIT 10
6. MATCH (e) – [ w: WorksOn ] → (p)
   WITH e, COUNT(p) AS numOfprojs
   WHERE numOfprojs > 2
   RETURN e.Ename , numOfprojs
   ORDER BY numOfprojs
7. MATCH (e) – [ w: WorksOn ] → (p)
   RETURN e , w, p
   ORDER BY e.Ename
   LIMIT 10
8. MATCH (e: EMPLOYEE {Empid: '2'})
   SET e.Job = 'Engineer'
```

Thank you