

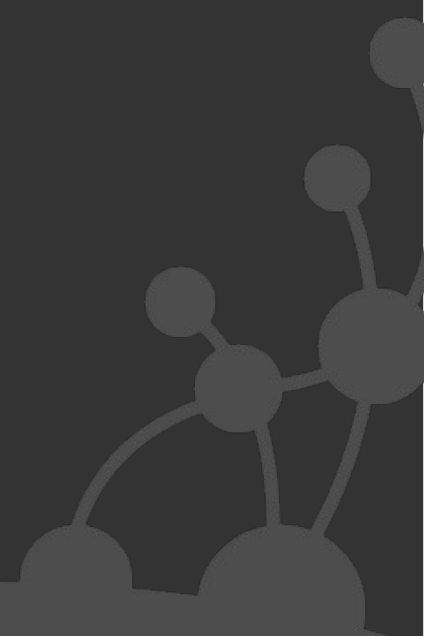
# Neo

some code snippets

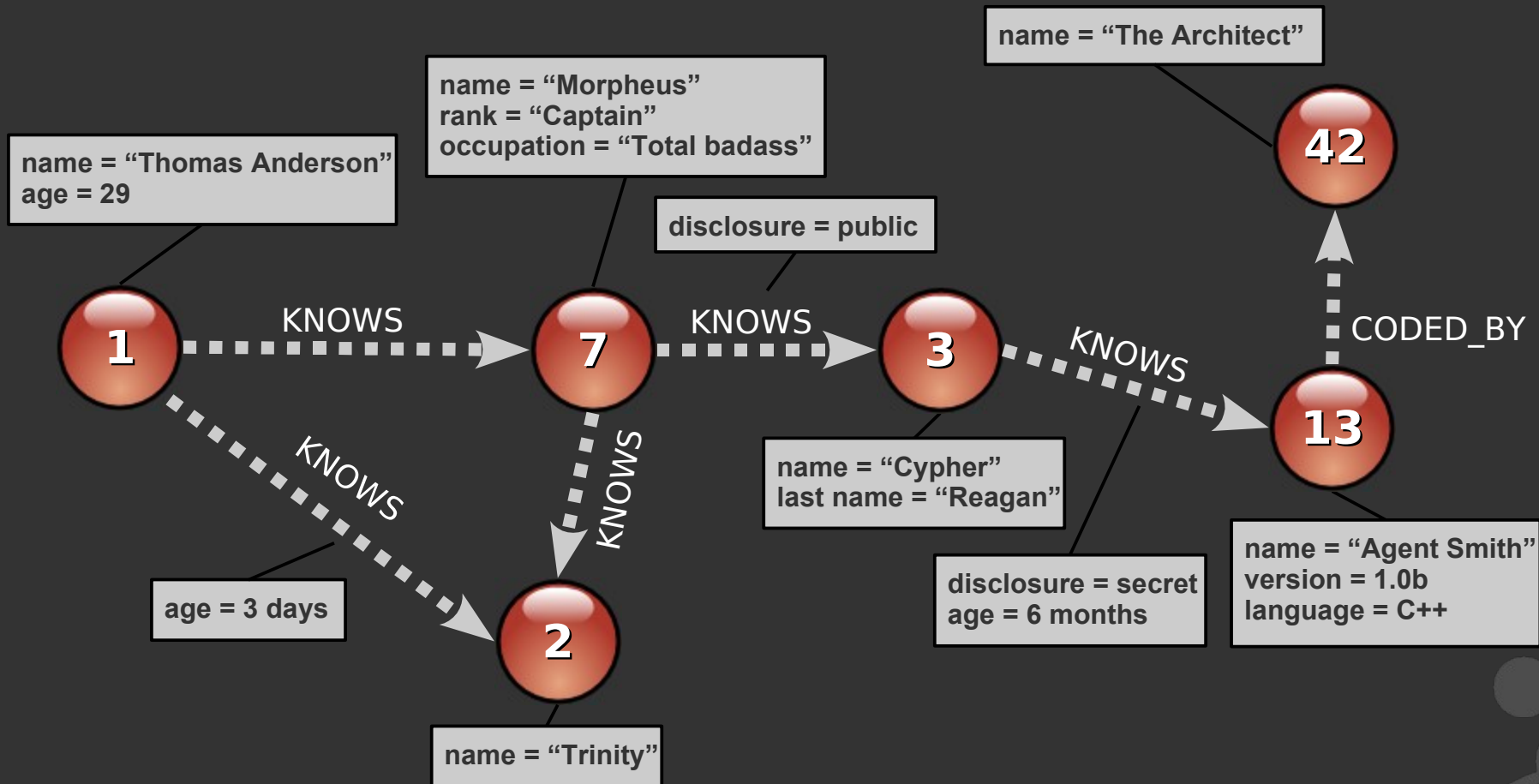
Emil Eifrem

2008-05-08, API v1.0-rc1-SNAPSHOT

# A few brief Neo4j code slides

- The following is a few slides from a live presentation – hopefully the code is self-explanatory
  - But if it isn't, please join the discussion on the mailing list @ <http://lists.neo4j.org>
  - First: how to create a node space
  - Second: how to traverse that node space
- 

# Example: The Matrix social graph



# Code (1): Building a node space

```
NeoService neo = ... // Get factory

// Create Thomas 'Neo' Anderson
Node mrAnderson = neo.createNode();
mrAnderson.setProperty( "name", "Thomas Anderson" );
mrAnderson.setProperty( "age", 29 );

// Create Morpheus
Node morpheus = neo.createNode();
morpheus.setProperty( "name", "Morpheus" );
morpheus.setProperty( "rank", "Captain" );
morpheus.setProperty( "occupation", "Total bad ass" );

// Create a relationship representing that they know each other
mrAnderson.createRelationshipTo( morpheus, RelTypes.KNOWS );
// ...create Trinity, Cypher, Agent Smith, Architect similarly
```

# Code (1): Building a node space

```
NeoService neo = ... // Get factory
Transaction tx = neo.beginTransaction();

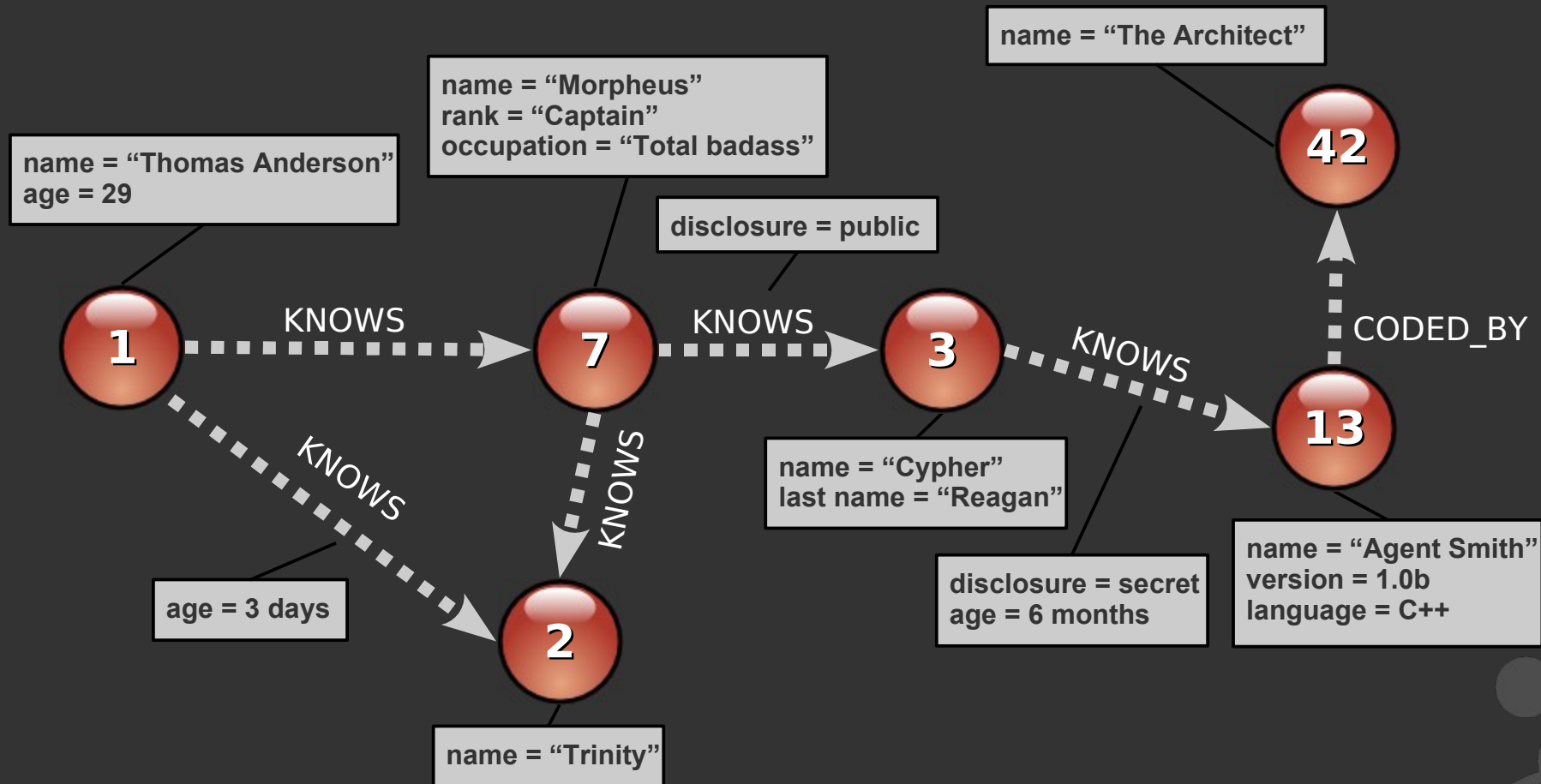
// Create Thomas 'Neo' Anderson
Node mrAnderson = neo.createNode();
mrAnderson.setProperty( "name", "Thomas Anderson" );
mrAnderson.setProperty( "age", 29 );

// Create Morpheus
Node morpheus = neo.createNode();
morpheus.setProperty( "name", "Morpheus" );
morpheus.setProperty( "rank", "Captain" );
morpheus.setProperty( "occupation", "Total bad ass" );

// Create a relationship representing that they know each other
mrAnderson.createRelationshipTo( morpheus, RelTypes.KNOWS );
// ...create Trinity, Cypher, Agent Smith, Architect similarly

tx.commit(); // Pseudo code, obviously wrap it in try-finally
```

# Traversal: Find Mr Anderson's friends



# What do we want to do?

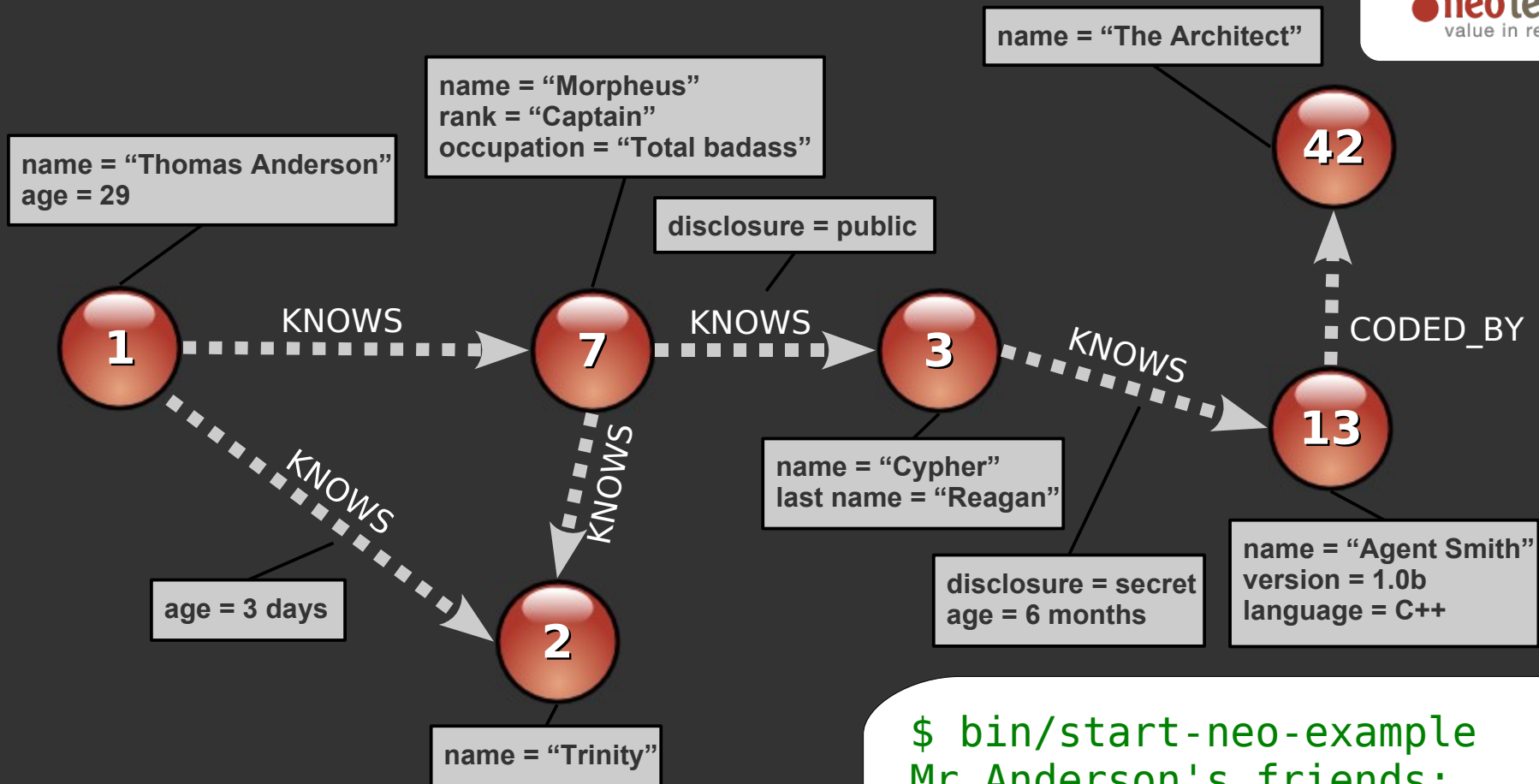
- ◎ We want to find all Mr Anderson's transitive friends
- ◎ So conceptually, we want to **traverse**, starting from the Mr Anderson node...
  - ... **breadth first** (closest friends first)
  - ... until the **end of the network** (ALL friends)
  - ... returning **all nodes** we visit, **except the first** one (only Mr Anderson's friends, not Mr Anderson himself)
  - ... but only traverse relationships of the **KNOWS** type in the **OUTGOING** direction

## Code (2): Traversing a node space

```
// Instantiate a traverser that returns Mr Anderson's friends
Traverser friendsTraverser = mrAnderson.traverse(
    Traverser.Order.BREADTH_FIRST,
    StopEvaluator.END_OF_NETWORK,
    ReturnableEvaluator.ALL_BUT_START_NODE,
    RelTypes.KNOWS,
    Direction.OUTGOING );

// Traverse the node space and print out the result
System.out.println( "Mr Anderson's friends:" );
for ( Node friend : friendsTraverser )
{
    System.out.printf( "At depth %d => %s%n",
        friendsTraverser.currentPosition().getDepth(),
        friend.getProperty( "name" ) );
}
```





```

friendsTraverser = mrAnderson.traverse(
  Traverser.Order.BREADTH_FIRST,
  StopEvaluator.END_OF_NETWORK,
  ReturnableEvaluator.ALL_BUT_START_NODE,
  RelTypes.KNOWS,
  Direction.OUTGOING );
  
```

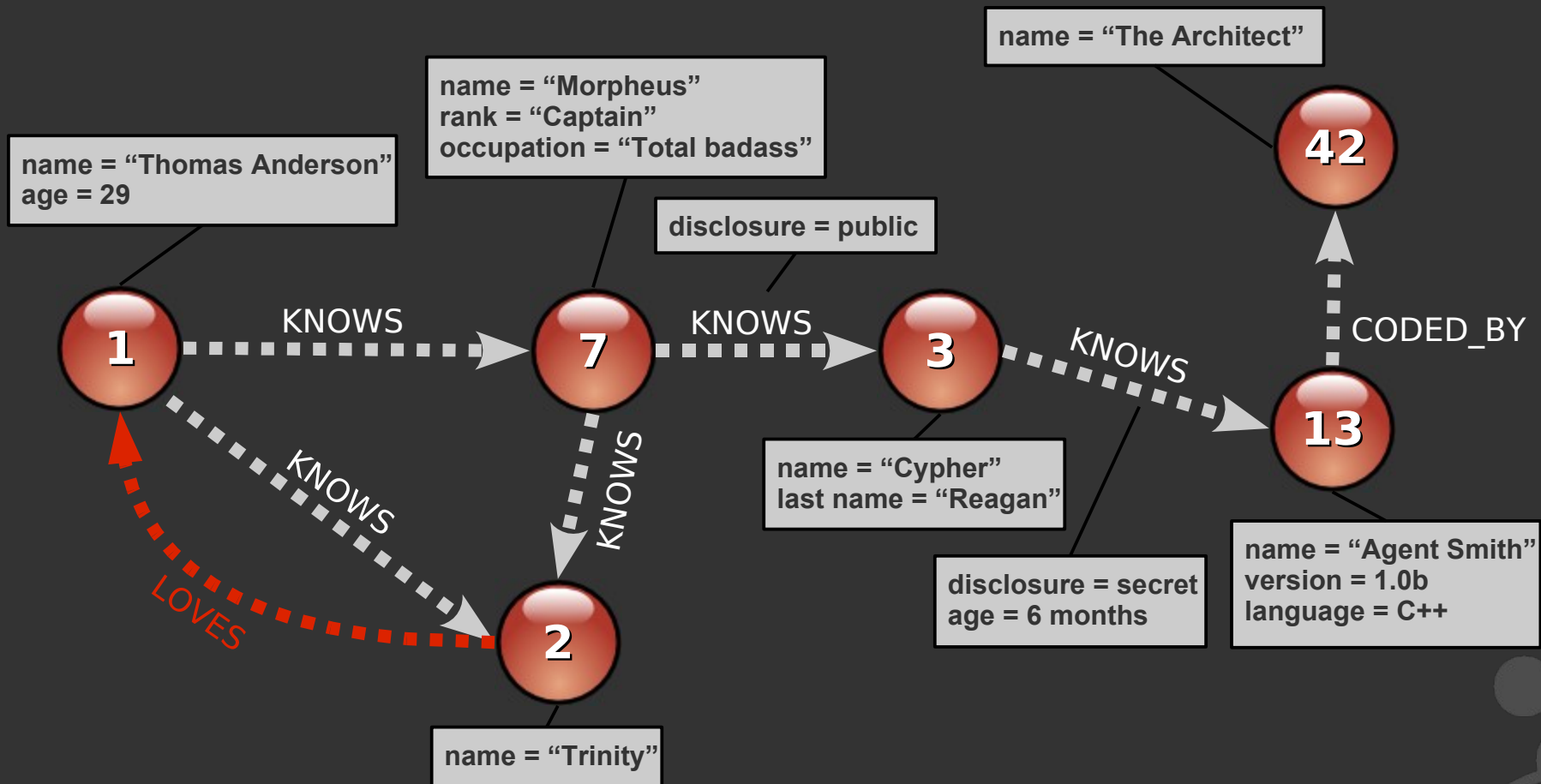
```

$ bin/start-neo-example
Mr Anderson's friends:
  
```

```

At depth 1 => Morpheus
At depth 1 => Trinity
At depth 2 => Cypher
At depth 3 => Agent Smith
$
  
```

# Evolving the domain: Friends in love?



# What do we want to do?

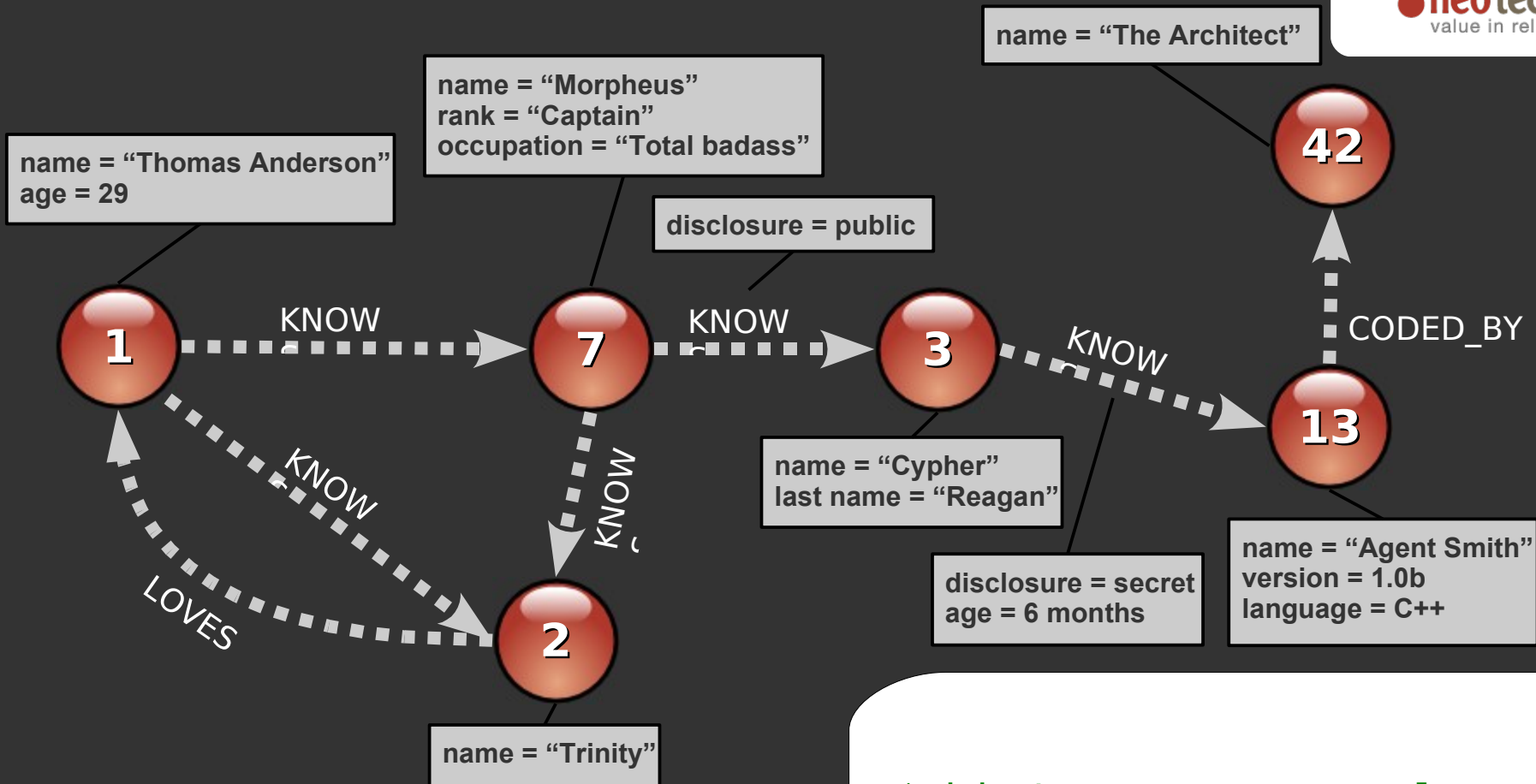
- ◎ We've now extended the domain with completely new functionality
  - Note how we don't have any predefined schemas – we could even create the new reltype dynamically without restarting our app
- ◎ Conceptually, we want to find everyone amongst Mr Anderson's friends who has a crush on someone
- ◎ So we still want to *traverse* all Mr Anderson's friends (like last time)
- ◎ But this time we only want to *return* the nodes that has an **OUTGOING** relationship of the **LOVES** type

# Code (3a): Custom traverser

```
// Create a traverser that returns all "friends in love"
Traverser loveTraverser = mrAnderson.traverse(
    Traverser.Order.BREADTH_FIRST,
    StopEvaluator.END_OF_NETWORK,
    new ReturnableEvaluator()
    {
        public boolean isReturnableNode( TraversalPosition pos )
        {
            return pos.currentNode().hasRelationship(
                RelTypes.LOVES, Direction.OUTGOING );
        }
    },
    RelTypes.KNOWS,
    Direction.OUTGOING );
```

# Code (3a): Custom traverser

```
// Traverse the node space and print out the result  
System.out.println( "Who's in love?" );  
for ( Node person : loveTraverser )  
{  
    System.out.printf( "At depth %d => %s%n",  
        loveTraverser.currentPosition().getDepth(),  
        person.getProperty( "name" ) );  
}
```



```

new ReturnableEvaluator()
{
  public boolean isReturnableNode(
    TraversalPosition pos)
  {
    return pos.currentNode().
      hasRelationship( RelTypes.LOVES,
        Direction.OUTGOING );
  }
},

```

```

$ bin/start-neo-example
Who's in love?

```

```

At depth 1 => Trinity
$

```

# Summary

## ◎ API details

- <http://api.neo4j.org>

## ◎ Feedback

- <http://lists.neo4j.org>

## ◎ Download

- <http://neo4j.org/download>

## ◎ Business

- <http://neotechnology.com>



---

[www.neo4j.org](http://www.neo4j.org)