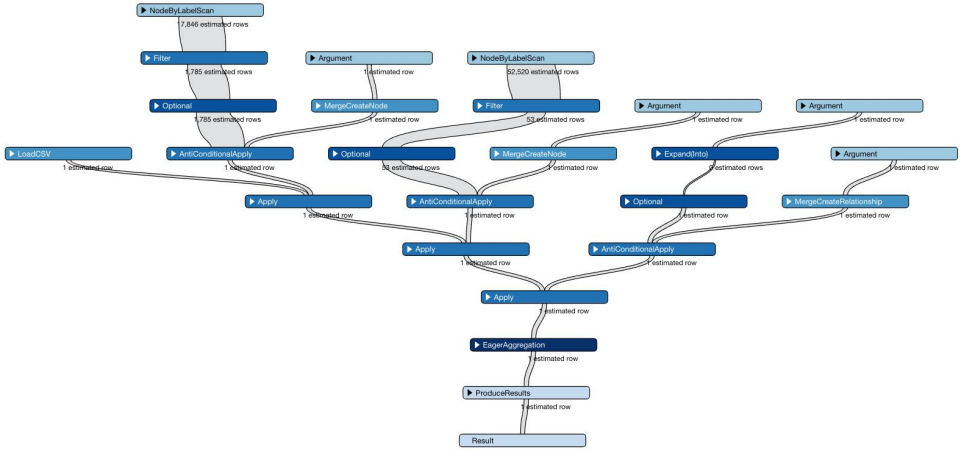
Neo4j Performance

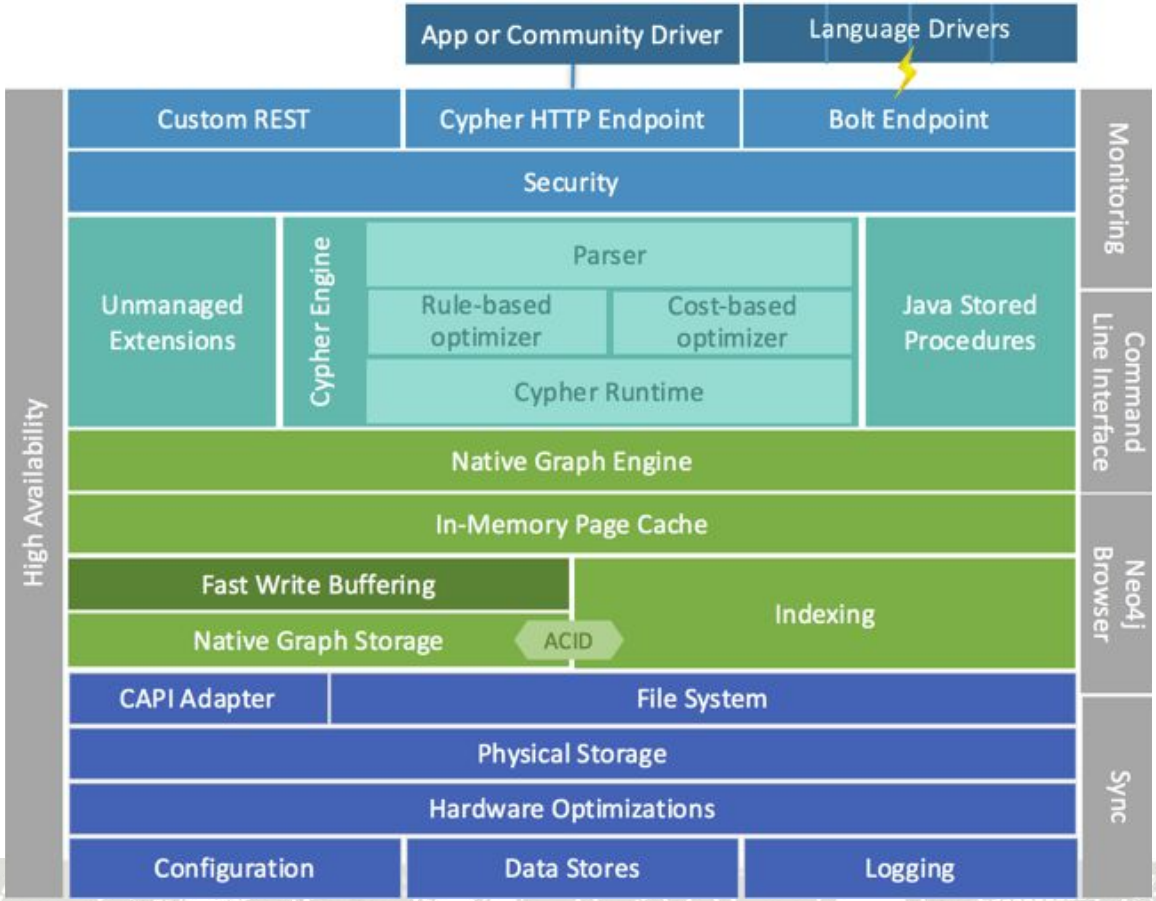
Cphbusiness 18th April, 2017
Craig Taverner

Agenda

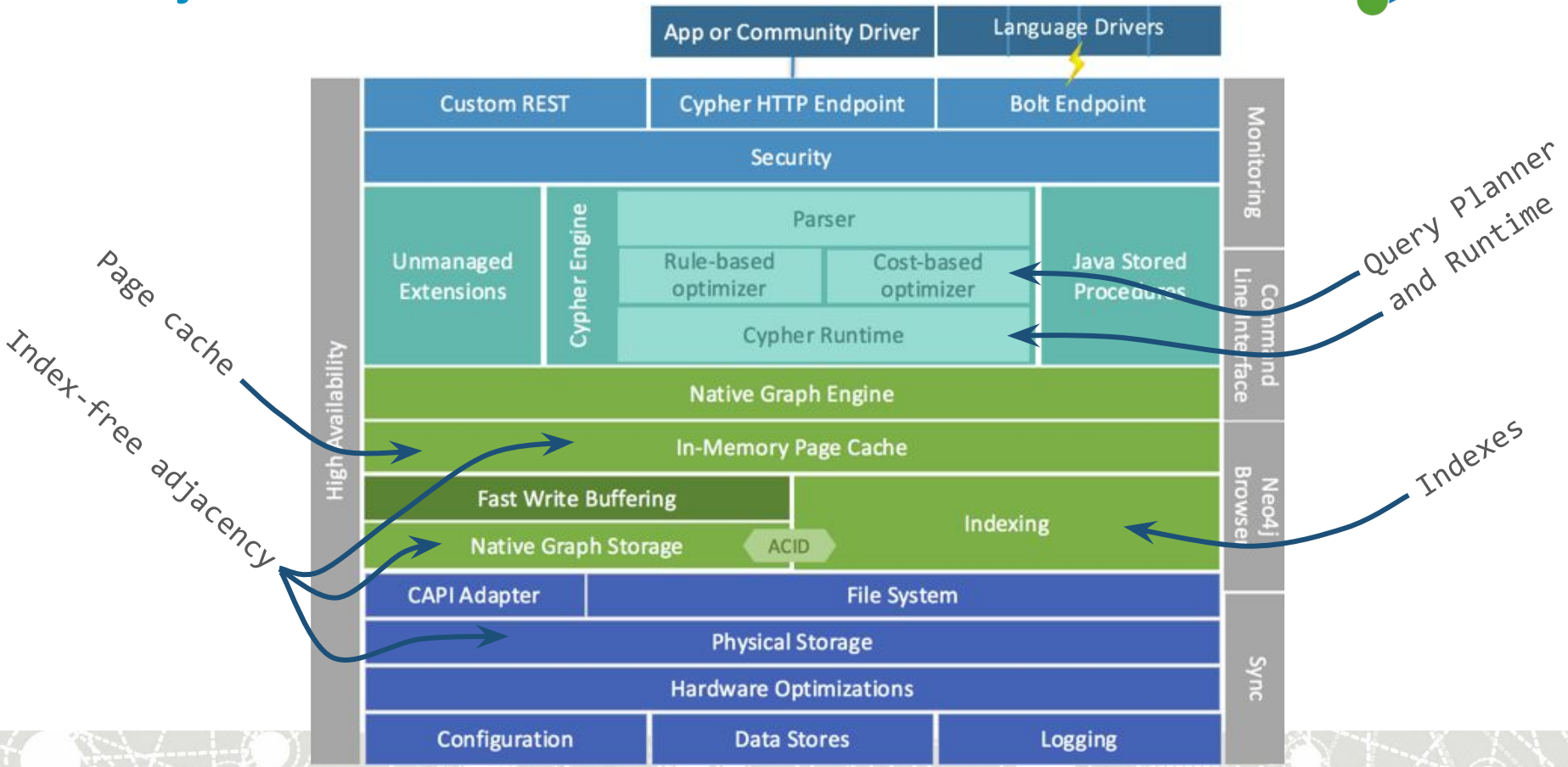
- Why is Neo4j so fast?
  - Index-free adjacency
  - Page cache
  - Indexes
  - Query planner and runtime
  - Efficient data structures
- What can I do?
  - Bulk import
  - Query planning
  - Native code (Procedures and extensions, Core API, Traversal API)
  - Benchmarks (LDBC, and JMH and Cypher)



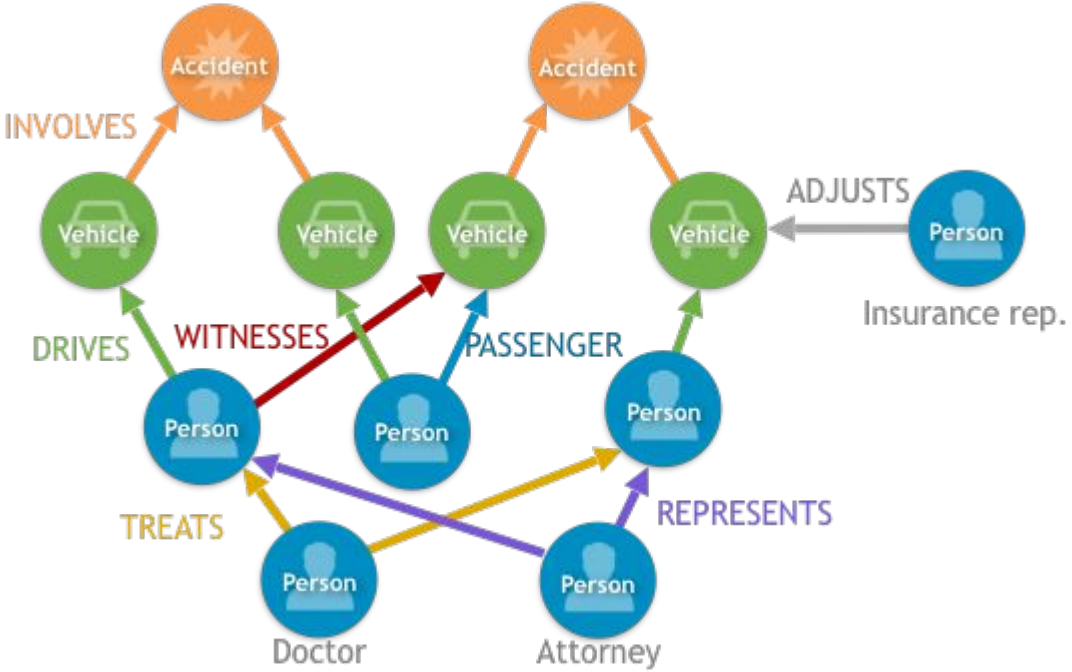
Neo4j Architecture



Neo4j Architecture

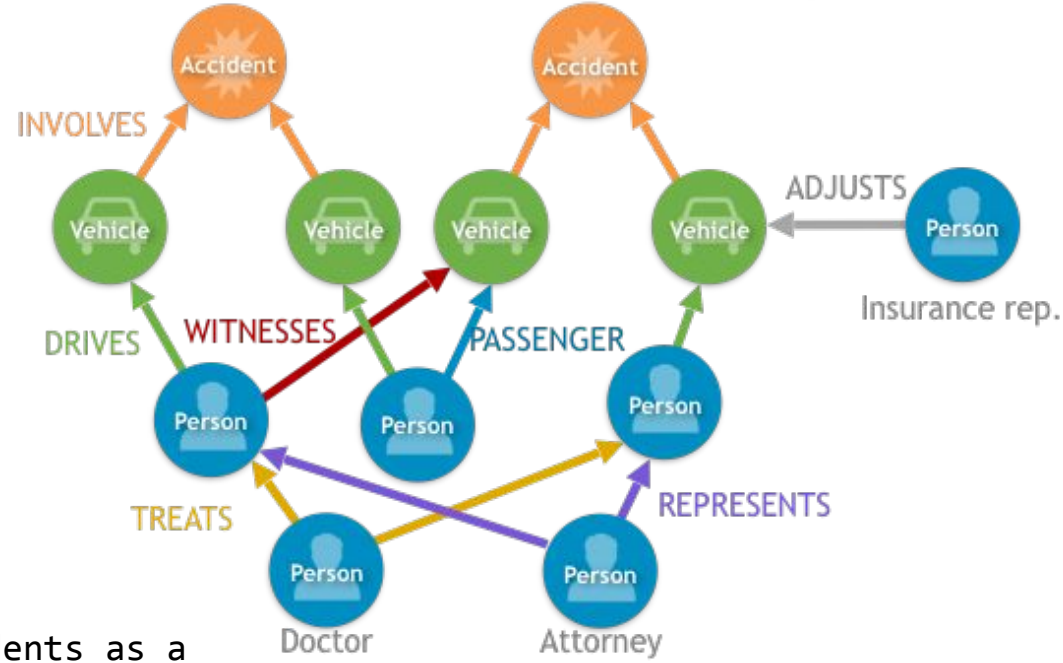


Index-free Adjacency



Index-free Adjacency: RDBMS vs Graph

```
MATCH (d:Doctor)
      -[:TREATS]->(p:Person)
      -[:DRIVES]->(v:Vehicle)
      -[:INVOLVES]->(a:Accident)
 WHERE a.year = '2016'
RETURN d.name, count(a) as count
 ORDER BY count DESC LIMIT 5
```

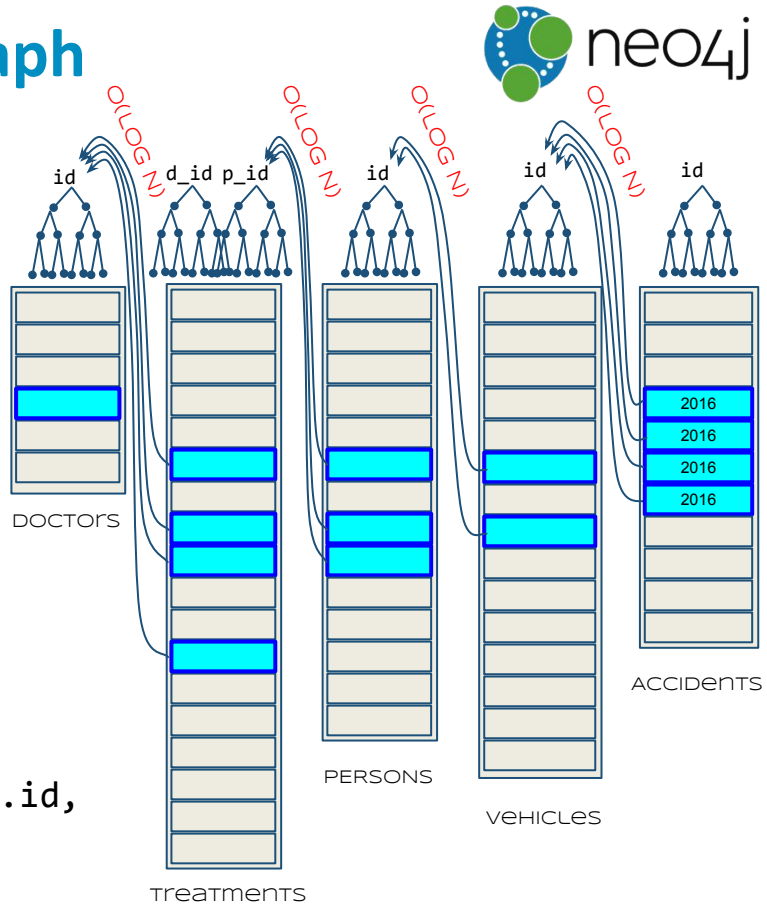



```
SELECT d.name, count(a) as count
FROM Doctors as d, Treatments as t
  Persons as p, Vehicles as v, Accidents as a
INNER JOIN ON t.doctor_id = d.id, t.patient_id = p.id,
  p.id = v.owner_id, v.id = a.vehicle_id
WHERE a.year = '2016' ORDER BY count DESC LIMIT 5
```

Index-free Adjacency: RDBMS vs Graph

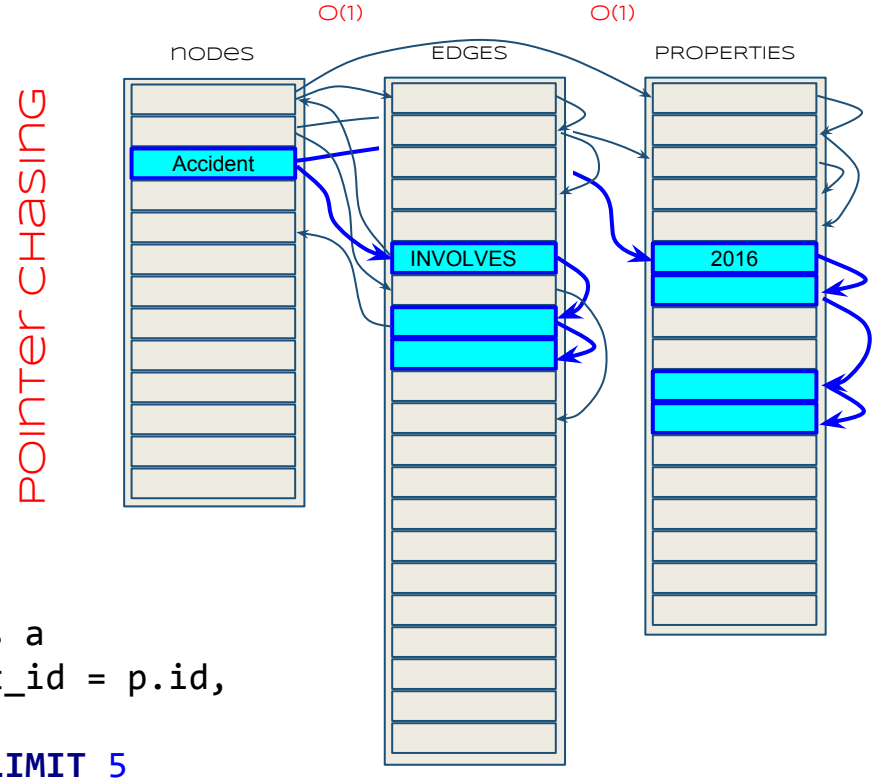
```
MATCH (d:Doctor)
      -[:TREATS]->(p:Person)
      -[:DRIVES]->(v:Vehicle)
      -[:INVOLVES]->(a:Accident)
 WHERE a.year = '2016'
RETURN d.name, count(a) as count
 ORDER BY count DESC LIMIT 5
```

```
SELECT d.name, count(a) as count
FROM Doctors as d, Treatments as t
  Persons as p, Vehicles as v, Accidents as a
INNER JOIN ON t.doctor_id = d.id, t.patient_id = p.id,
  p.id = v.owner_id, v.id = a.vehicle_id
WHERE a.year = '2016' ORDER BY count DESC LIMIT 5
```



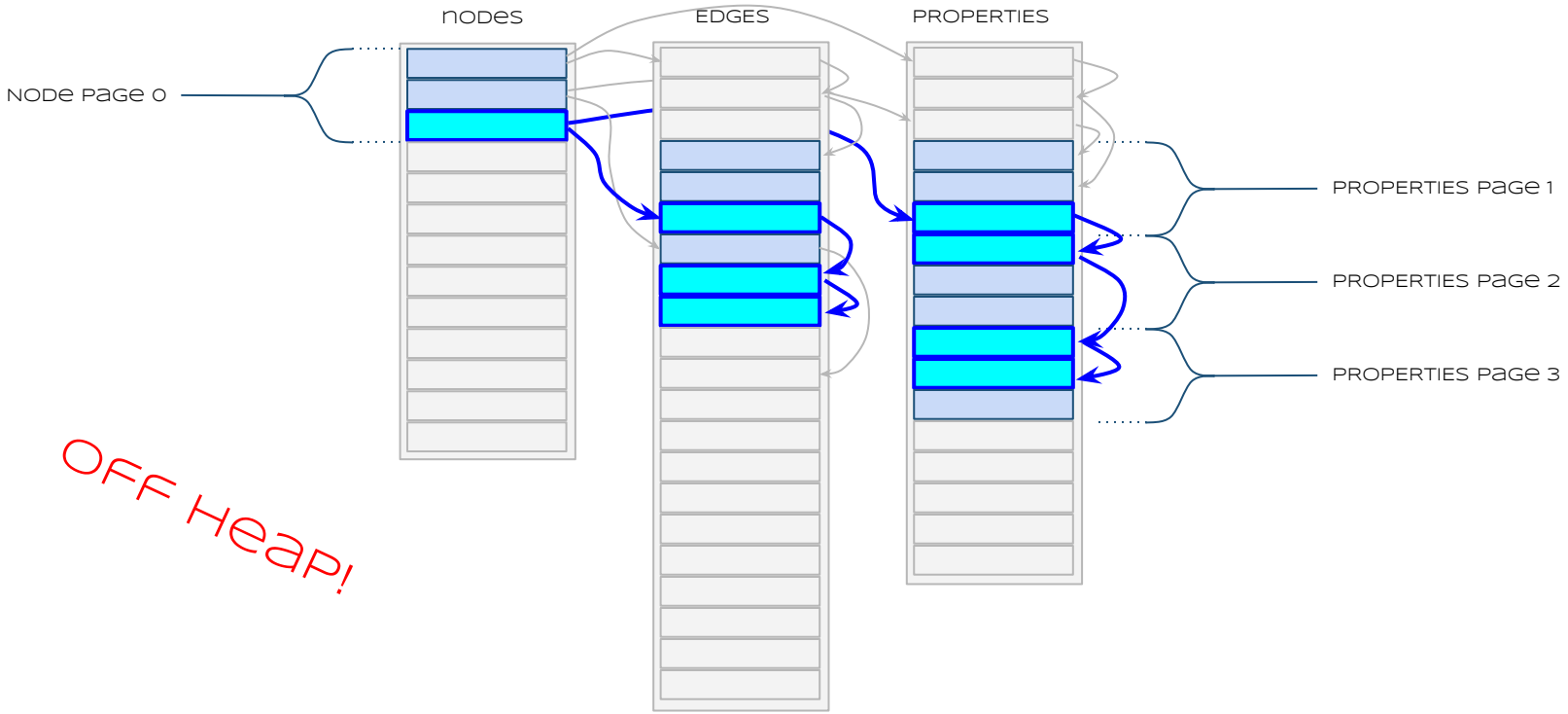
Index-free Adjacency: RDBMS vs Graph

```
MATCH (d:Doctor)
      -[:TREATS]->(p:Person)
      -[:DRIVES]->(v:Vehicle)
      -[:INVOLVES]->(a:Accident)
 WHERE a.year = '2016'
RETURN d.name, count(a) as count
 ORDER BY count DESC LIMIT 5
```




```
SELECT d.name, count(a) as count
FROM Doctors as d, Treatments as t
  Persons as p, Vehicles as v, Accidents as a
INNER JOIN ON t.doctor_id = d.id, t.patient_id = p.id,
  p.id = v.owner_id, v.id = a.vehicle_id
WHERE a.year = '2016' ORDER BY count DESC LIMIT 5
```

Page cache


NODES
EDGES
PROPERTIES
NODE PAGE 0
PROPERTIES PAGE 1
PROPERTIES PAGE 2
PROPERTIES PAGE 3
OFF HEAP!

Indexes

- Equality
  - `MATCH (user:User) WHERE user.email = 'joe@soap.com'`
- Range
  - `MATCH (user:User) WHERE user.age > 23`
- Existence
  - `MATCH (user:User) WHERE exists(user.marker)`
- Prefix
  - `MATCH (user:User) WHERE user.name STARTS WITH 'joe'`
- Composite
  - `MATCH (user:User) WHERE user.firstname = 'joe' AND user.lastname = 'soap'`

When are indexes used

- Rules
  - Cypher query behaves the same with and without indexes
    - Same results
    - Only performance is impacted
- Literal predicates
  - `MATCH (user:User) WHERE user.email = 'joe@soap.com'`
- Nested Index Loop Join
  - ```
    MATCH (user:User) WHERE user.firstname = 'joe'
    MATCH (other:User) WHERE other.age = user.age
    ```
- USING Index Hint
  - ```
    MATCH (user:User) WHERE user.email = 'joe@soap.com'
        USING INDEX user:User(email)
    ```

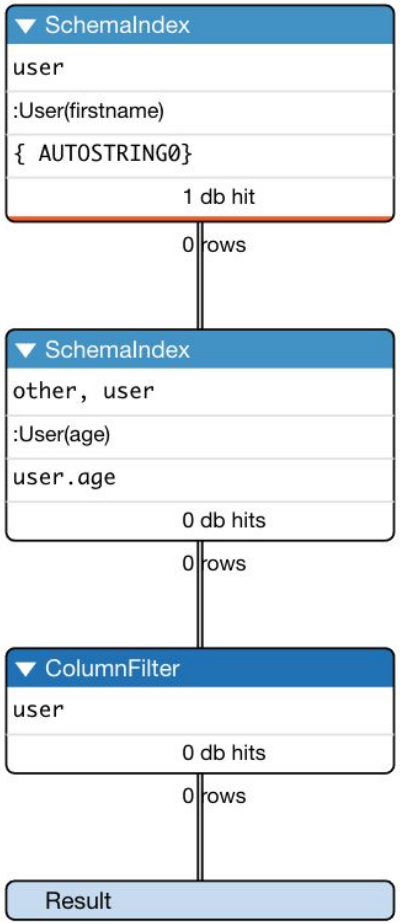
Query Planner and Runtime

- RULE
  - Plan based entirely on the query using rules
- COST
  - Plan based on statistics from actual database
- INTERPRETED
  - Plan converted into operator objects that stream results objects
- COMPILED
  - Plan converted into optimized code with minimum overhead

RULE Query Planner

- RULE
  - Plan based entirely on the query using rules

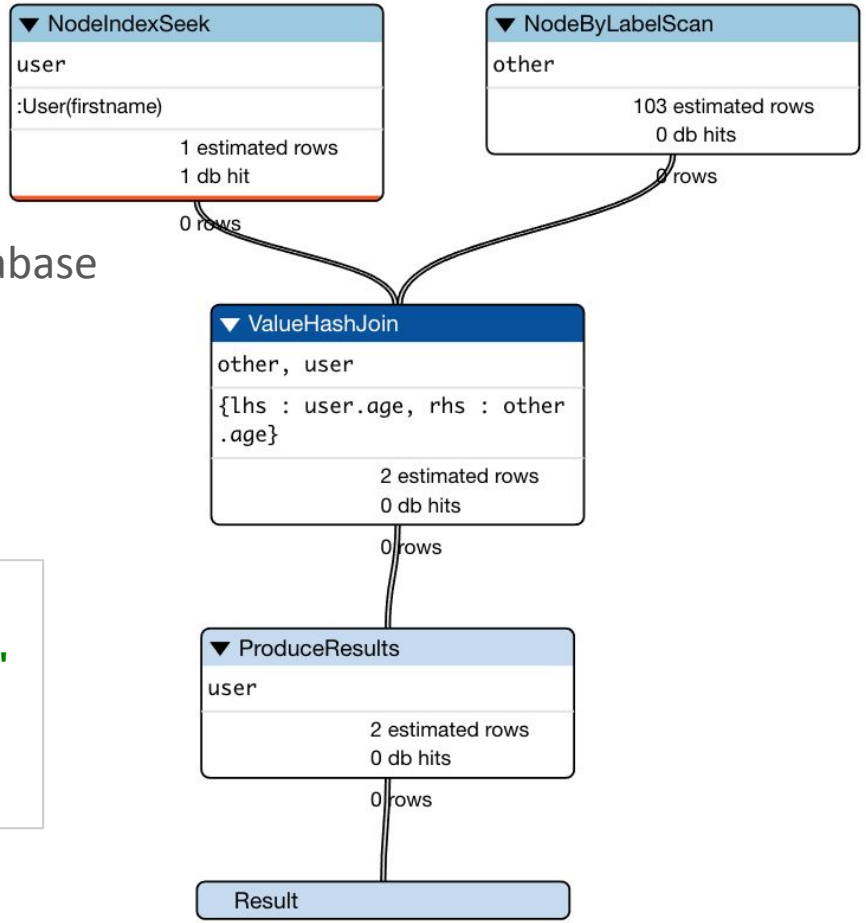
```
CYPHER planner=RULE PROFILE
MATCH (user:User) WHERE user.firstname = 'joe'
MATCH (other:User) WHERE other.age = user.age
RETURN user, other
```



COST Query Planner

- COST
  - Plan based on statistics from actual database
  - IDP algorithm
  - Since Neo4j 2.1 and 2.2
  - Write support since Neo4j 2.3

```
CYPHER planner=RULE PROFILE
MATCH (user:User) WHERE user.firstname = 'joe'
MATCH (other:User) WHERE other.age = user.age
RETURN user, other
```



Runtimes: Interpreted and Compiled

- INTERPRETED
  - Plan converted into operator objects that stream results objects

```
CYPHER runtime=interpreted MATCH (user:User) WHERE user.firstname = 'joe' RETURN user
```

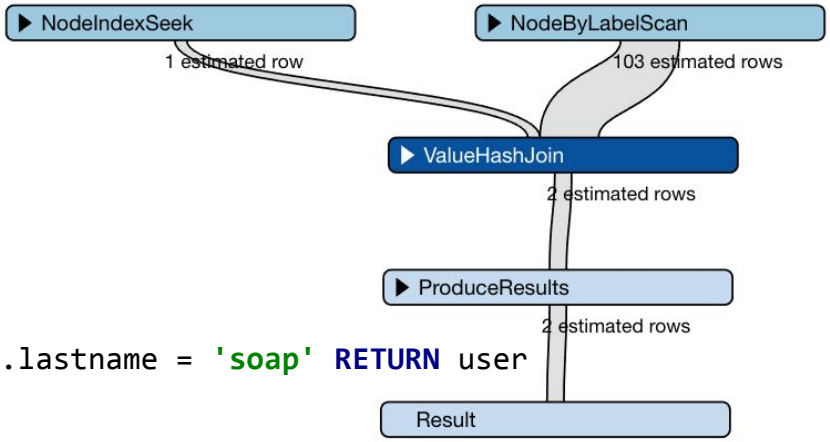
- COMPILED
  - Plan converted into optimized code with minimum overhea
  - Experimental support in Neo4j 3.0
  - Supports some parts of Cypher since Neo4j 3.2
  - Some queries 10x faster

```
CYPHER runtime=compiled MATCH (user:User) WHERE user.firstname = 'joe' RETURN user
```

Efficient data structures

- Object pooling
  - Transaction pool
- Cursors
  - Create on demand
- Caches
  - When necessary
  - Removed object cache
  - Added count store

Query Plan Cache



- Plan caching

```
MATCH (user:User) WHERE user.firstname = 'joe' AND user.lastname = 'soap' RETURN user
```

- Parameterization

```
MATCH (user:User) WHERE user.firstname = 'joe' AND user.lastname = $soap RETURN user
```

- Auto-parameterization

```
MATCH (user:User) WHERE user.firstname = $auto_0 AND user.lastname = $auto_1 RETURN user
```

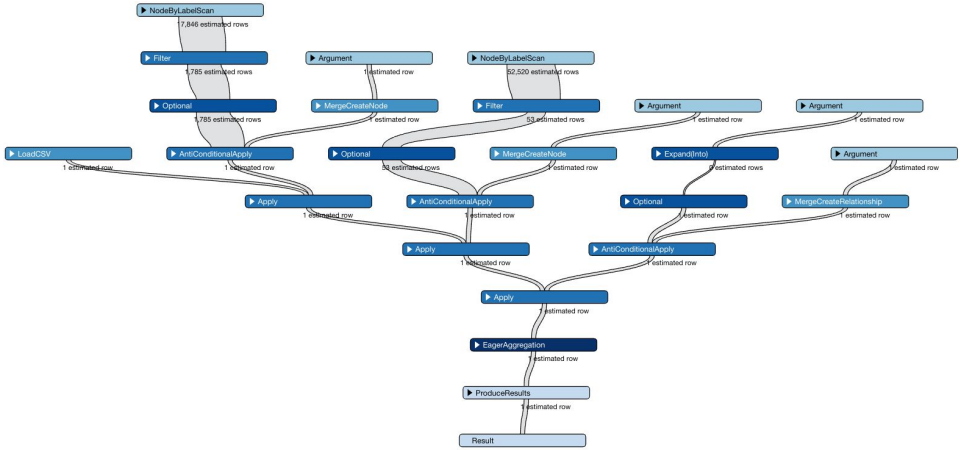
Neo4j Performance Tuning

Cphbusiness 18th April, 2017

Craig Taverner

Agenda

- Why is Neo4j so fast?
  - Index-free adjacency
  - Page cache
  - Indexes
  - Query planner and runtime
  - Efficient data structures
- What can I do?
  - Bulk import
  - Query planning
  - Native code (Procedures and extensions, Core API, Traversal API)
  - Benchmarks (LDBC, and JMH and Cypher)



Bulk Import

- Online
  - CREATE / MERGE — many small transactions - overhead
  - LOAD CSV — one massive transaction - memory
  - PERIODIC COMMIT — several big transactions - risk of partial load
  - Core API — Java code
- Offline
  - Import-tool - *'lock'* the database and write directly to stores

Bulk Import

- Online
  - CREATE / MERGE
  - LOAD CSV
  - PERIODIC COMMIT
  - Core API

```
MERGE (a:Author {name:book.author})
```

```
LOAD CSV WITH HEADERS FROM 'file:///books.csv' as book
```

```
USING PERIODIC COMMIT 1000
```

```
try ( Transaction tx = graph.beginTx() ) {...}
```

- Offline
  - Import-tool - *'lock'* the database and write directly to stores

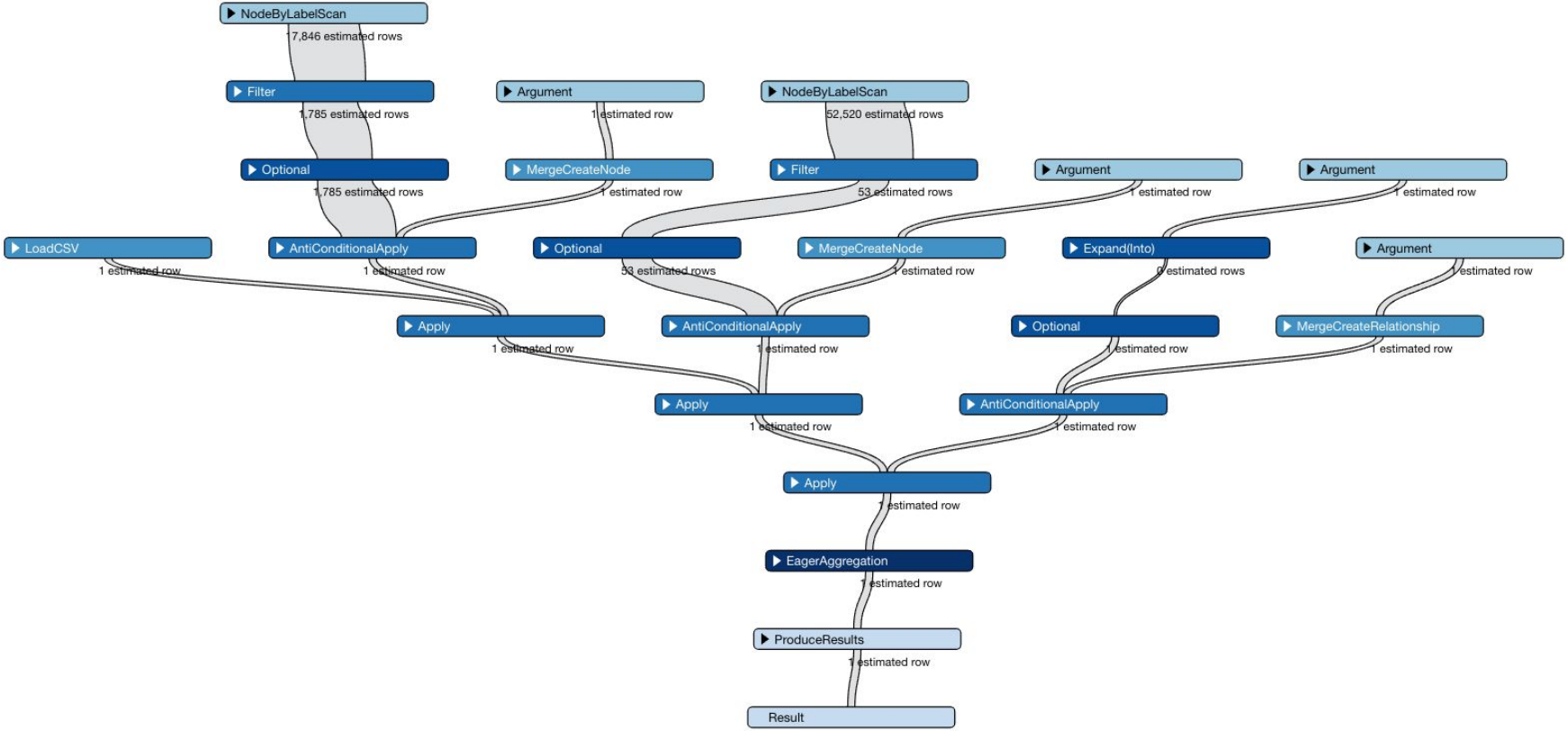
LOAD CSV

```
pg_id,title,author,language
1,The Declaration of Independence of the United States of America,"Jefferson, Thomas",en
100,The Complete Works of William Shakespeare,"Shakespeare, William",en
10000,The Magna Carta,Anonymous,en
10001,Apocolocyntosis,"Seneca, Lucius Annaeus",en
10002,The House on the Borderland,"Hodgson, William Hope",en
10003,"My First Years as a Frenchwoman, 1876-1879","Waddington, Mary King",en
10004,The Warriors,"Lindsay, Anna Robertson Brown",en
10005,"A Voyage to the Moon With Some Account of the Manners and Customs, Science and Philosophy, of the People of
Morosofia, and Other Lunarians","Tucker, George",en
...
```

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM 'file:///books.csv' as book
  MERGE (a:Author {name:book.author})
  MERGE (b:Book {pg_id:book.pg_id, title:book.title, language:book.language})
  MERGE (b)-[:AUTHORED_BY]->(a)
  WITH book
  RETURN count(book)
```

30 minutes to load 52K books!

LOAD CSV


NodeByLabelScan
7,846 estimated rows
Filter
1,785 estimated rows
Argument
1 estimated row
NodeByLabelScan
52,520 estimated rows
Optional
1,785 estimated rows
MergeCreateNode
1 estimated row
Filter
53 estimated rows
Argument
1 estimated row
Argument
1 estimated row
LoadCSV
1 estimated row
AntiConditionalApply
1 estimated row
Optional
53 estimated rows
MergeCreateNode
1 estimated row
Expand(Into)
0 estimated rows
Argument
1 estimated row
Apply
1 estimated row
AntiConditionalApply
1 estimated row
Optional
1 estimated row
MergeCreateRelationship
1 estimated row
Apply
1 estimated row
AntiConditionalApply
1 estimated row
Apply
1 estimated row
EagerAggregation
1 estimated row
ProduceResults
1 estimated row
Result

bin/neo4j-import

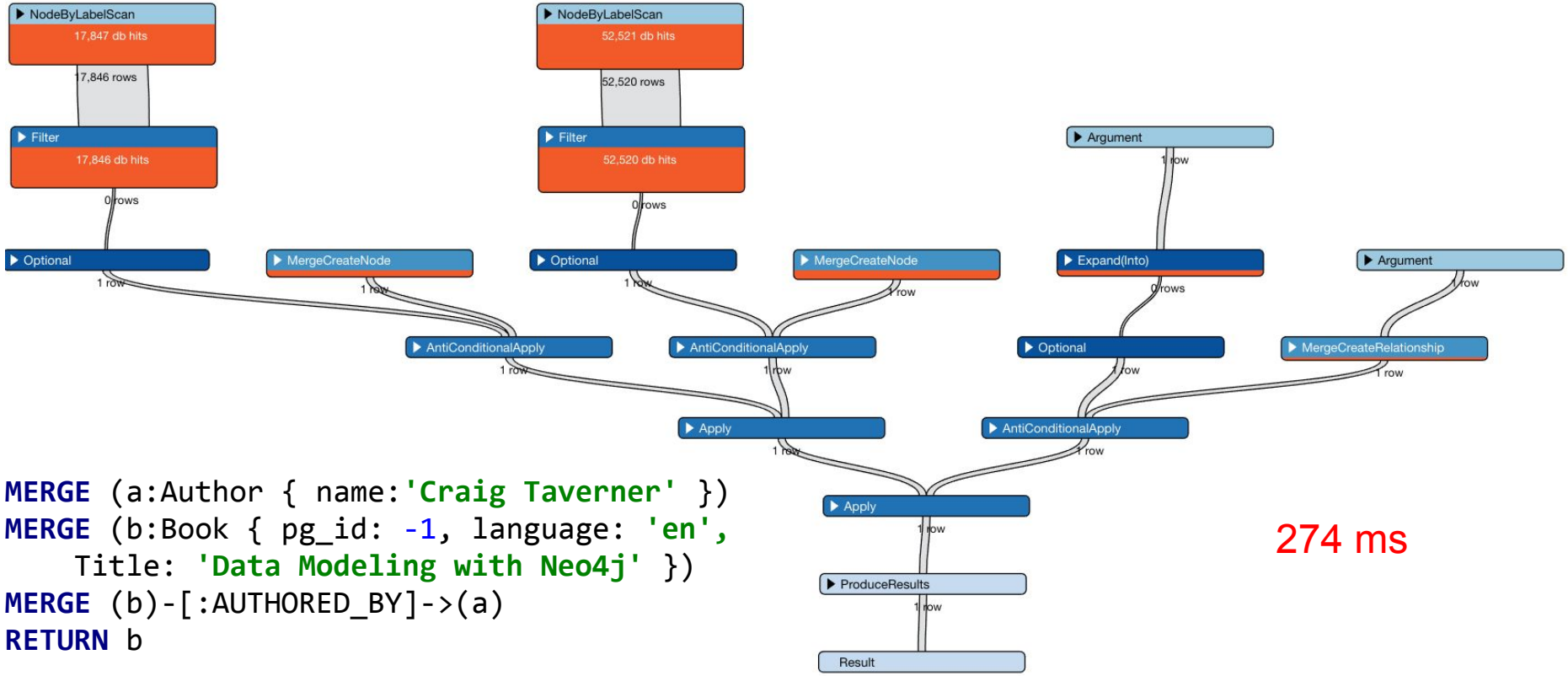
- Create CSV files for nodes and relationships
  - Books `pg_id:ID(Book),title,language`
  - Authors `author:ID(Author)`
  - Book Authors `:START_ID(Book),:END_ID(Author)`
  - Cities `city_id:ID(City),name,latitude,longitude`
  - City mentions `pg_id:START_ID(Book),city_id:END_ID(City)`
- Import into blank database

```
bin/neo4j-import --into ../../data/databases/graph.db \
    --nodes:Book books.csv \
    --nodes:Author authors.csv \
    --relationships:AUTHORED_BY book_authors.csv \
    --nodes:City cities.csv \
    --relationships:MENTIONS books_mentions_city.csv
```

IMPORT DONE
in 6s 445ms

```
MATCH (b:Book)-[:AUTHORED_BY]->(a:Author)
WHERE b.language = 'zh'
RETURN a,b
```

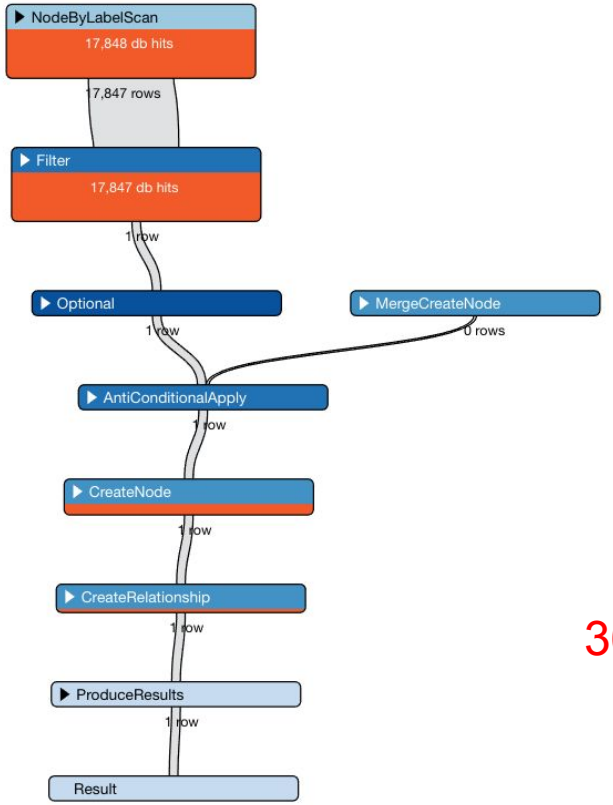
Query Planning




```
MERGE (a:Author { name:'Craig Taverner' })
MERGE (b:Book { pg_id: -1, language: 'en',
    Title: 'Data Modeling with Neo4j' })
MERGE (b)-[:AUTHORED_BY]->(a)
RETURN b
```

274 ms

Query Planning




```
MERGE (a:Author { name:'Craig Taverner' })
CREATE (b:Book { pg_id: -1, language: 'en',
    Title: 'Data Modeling with Neo4j' })
CREATE (b)-[:AUTHORED_BY]->(a)
RETURN b
```

36 ms

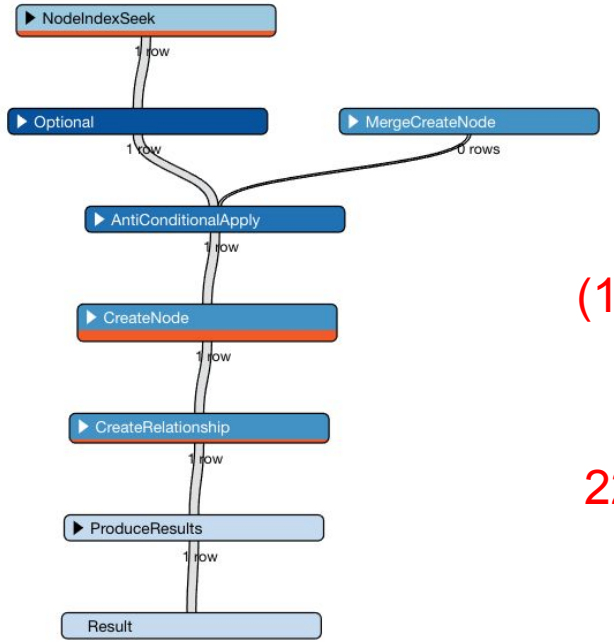
Query Planning

```
CREATE INDEX ON :Author(name)
```

(118 ms)

```
MERGE (a:Author { name:'Craig Taverner' })
CREATE (b:Book { pg_id: -1, language: 'en',
    Title: 'Data Modeling with Neo4j' })
CREATE (b)-[:AUTHORED_BY]->(a)
RETURN b
```

22 ms



Native Code

- User Defined Procedures and Functions
  - `MATCH (me:User {name:'me'}) CALL my.findMyFavorites(me)`
- Unmanaged Extensions and Plugins
  - Extend REST API (older approach, before procedures)
- Core API
  - `graph.findNode( Label.label( "User" ), "name", "me" );`
- Traversal API
  - `gds.traversalDescription().depthFirst()....traverse(startNode);`

Building a user defined procedure

code walkthrough demo

Benchmarks

- LDBC
  - Linked Data Benchmark Council
- Microbenchmarks
  - Extensive coverage of all Core API and other components
- Cypher benchmarks
  - Comparative benchmarks of wide range of Cypher queries and datasets
- Marketing
  - Benchmarks focusing on specific hot topics