# Cassandra Java Driver

Big Data Systems

Dr. Rubi Boim

## Cassandra driver

- Open source —> more than 1…
  Java, Python, Ruby, C#, Nodejs, PHP, C++, Scala and many more

- Just for Java there are more than 5 different vendors

- We will use Datastax’s version
  built in support for AstraDB’s security API

## Datastax Java Driver

- Previously there were 2 versions
  - OSS - Cassandra
  - DSE - Datastax Enterprise

- Today there is a single driver (OSS) that supports Cassandra, Datastax Enterprise and AstraDB

https://github.com/datastax/java-driver

## Installing the driver - option 1

- Using Maven

```
<dependency>
  <groupId>com.datastax.oss</groupId>
  <artifactId>java-driver-core</artifactId>
  <version>${driver.version}</version>
</dependency>

<dependency>
  <groupId>com.datastax.oss</groupId>
  <artifactId>java-driver-query-builder</artifactId>
  <version>${driver.version}</version>
</dependency>

<dependency>
  <groupId>com.datastax.oss</groupId>
  <artifactId>java-driver-mapper-runtime</artifactId>
  <version>${driver.version}</version>
</dependency>
```

## Installing the driver - option 2

- Manually install OSS JARs
  and their dependencies

- For HW2 all the JARs will be provided
  you can use Maven if you prefer

## CqlSession

- The main entry point of the driver
- Holds the state and the connections to the cluster
  with built in connection pool
- Executes queries
- Thread safe

## CqlSession - usage (1)

Connection to AstraDB example

```
CqlSession session = CqlSession.builder()
      .withCloudSecureConnectBundle(Paths.get(pathAstraDBBundleFile))
      .withAuthCredentials(username, password)
      .withKeyspace(keyspace)
      .build();


// finally (when the server terminate, NOT after a single query)
session.close();
```

RTFM https://github.com/datastax/java-driver/tree/4.x/manual/core

## CqlSession - usage (2)

- Create <u>one</u> session per application

```
// Anti-pattern: creating two sessions doubles the number of TCP connections
// opened by the driver
CqlSession session1 = CqlSession.builder().withKeyspace(…).build();
CqlSession session2 = CqlSession.builder().withKeyspace(…).build();
```

# CqlSession - execute

- Executes a **statement** (query)
- Returns a ResultSet
- Sync/Async

`Statement`

- **`SimpleStatement`** (created from String)
- **`BoundStatement`** (created from PreparedStatement)
- **`BatchStatement`**

## CqlSession - execute (example 1)

```
// simple queries
session.execute("INSERT INTO users(user_id,name) VALUES(123,'Rubi')");

// this is alias for
session.execute(SimpleStatement.newInstance(
                "INSERT INTO users(user_id,name) VALUES(123,'Rubi')"));
```

# CqlSession - execute (example 2)

```
// simple queries with placeholders
session.execute(“INSERT INTO users(user_id,name) VALUES(?,?)”,
                123,’Rubi’);
```

## CqlSession - execute (example 3)

```
// simple query with results
ResultSet rs = session.execute(“SELECT * FROM users WHERE user_id=?”, 123);
```

## ResultSet

- An iterable over `Row` objects
- `One()` - Returns the next element, or null if exhausted
- Initialized to a “row before”

## ResultSet - example (1)

```
// simple query with a “single row” result
ResultSet rs = session.execute(“SELECT count(*) FROM users WHERE user_id=?”, 123);

Row row = rs.one();
System.out.println(row.getInt(0));
```

See the table in a few slides for the complete mapping

## ResultSet - example (2)

```
// simple query with a multi rows
ResultSet rs = session.execute(“SELECT * FROM users”);

for (Row row : rs) {
  System.out.println(row.getInt(0) + “ -- “ + row.getString(“name”));
}
```

Java iterator

Column by “location”

Column by name

# ResultSet - example (3)

```
// simple query with a multi rows
ResultSet rs = session.execute(“SELECT * FROM users”);

Row row = rs.one();
while (row != null) {
  System.out.println(row.getInt(0) + “ -- “ + row.getString(“name”));
  row = rs.one();
}
```

Advancing…

# ResultSet - CQL to Java mapping (1)

| CQL3 data type | Getter name | Java type | See also |
|---|---|---|---|
| ascii | getString | java.lang.String | |
| bigint | getLong | long | |
| blob | getByteBuffer | java.nio.ByteBuffer | |
| boolean | getBoolean | boolean | |
| counter | getLong | long | |
| date | getLocalDate | java.time.LocalDate | Temporal types |
| decimal | getBigDecimal | java.math.BigDecimal | |
| double | getDouble | double | |
| duration | getCqlDuration | CqlDuration | Temporal types |
| float | getFloat | float | |
| inet | getInetAddress | java.net.InetAddress | |
| int | getInt | int | |
| list | getList | java.util.List | |
| map | getMap | java.util.Map<K, V> | |
| set | getSet | java.util.Set | |

# ResultSet - CQL to Java mapping (2)

| CQL3 data type | Getter name | Java type | See also |
|---|---|---|---|
| smallint | getShort | short | |
| text | getString | java.lang.String | |
| time | getLocalTime | java.time.LocalTime | Temporal types |
| timestamp | getInstant | java.time.Instant | Temporal types |
| timeuuid | getUuid | java.util.UUID | |
| tinyint | getByte | byte | |
| tuple | getTupleValue | TupleValue | Tuples |
| user-defined types | getUDTValue | UDTValue | User-defined types |
| uuid | getUuid | java.util.UUID | |
| varchar | getString | java.lang.String | |

## Prepared statements

Prepare a query string once, reuse with different values

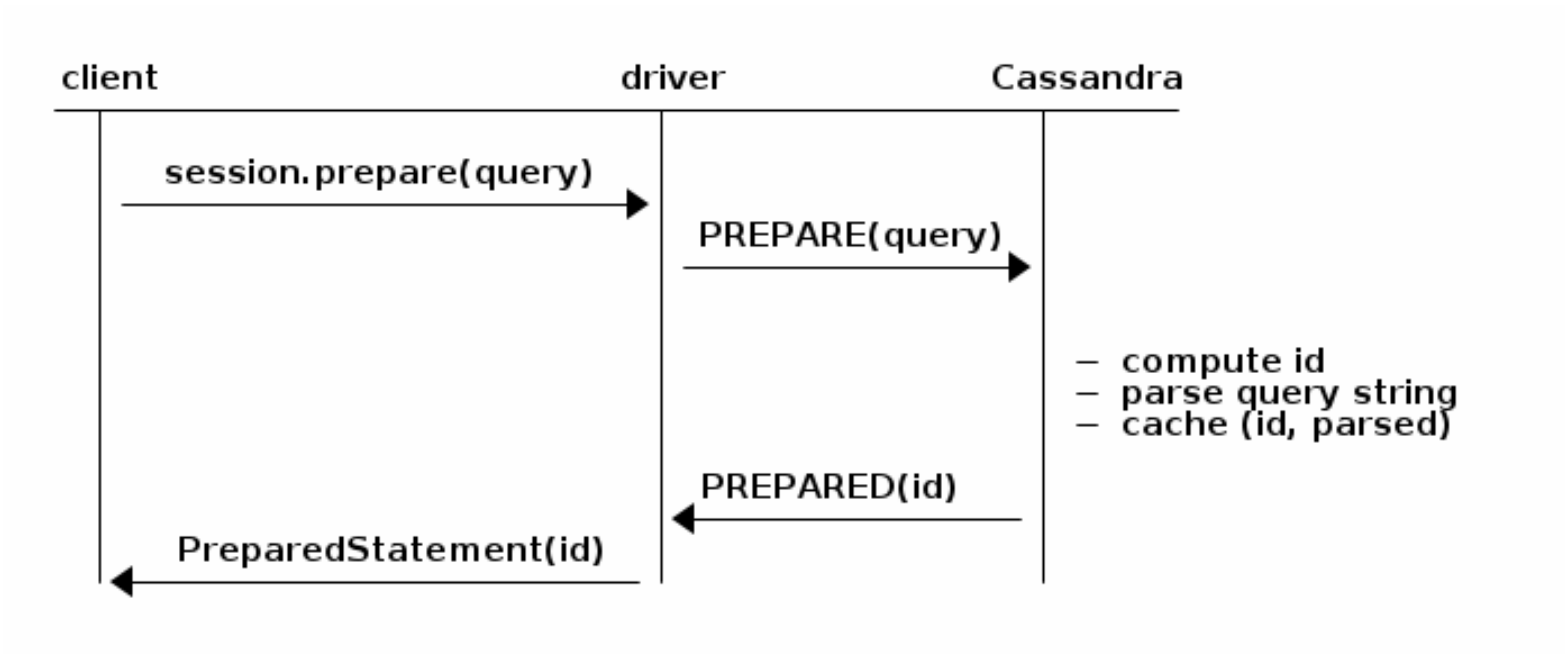
- More efficient than simple statements for queries that are used often

- Requires query to be “saved” on the servers

  thus - do not use with infrequent queries

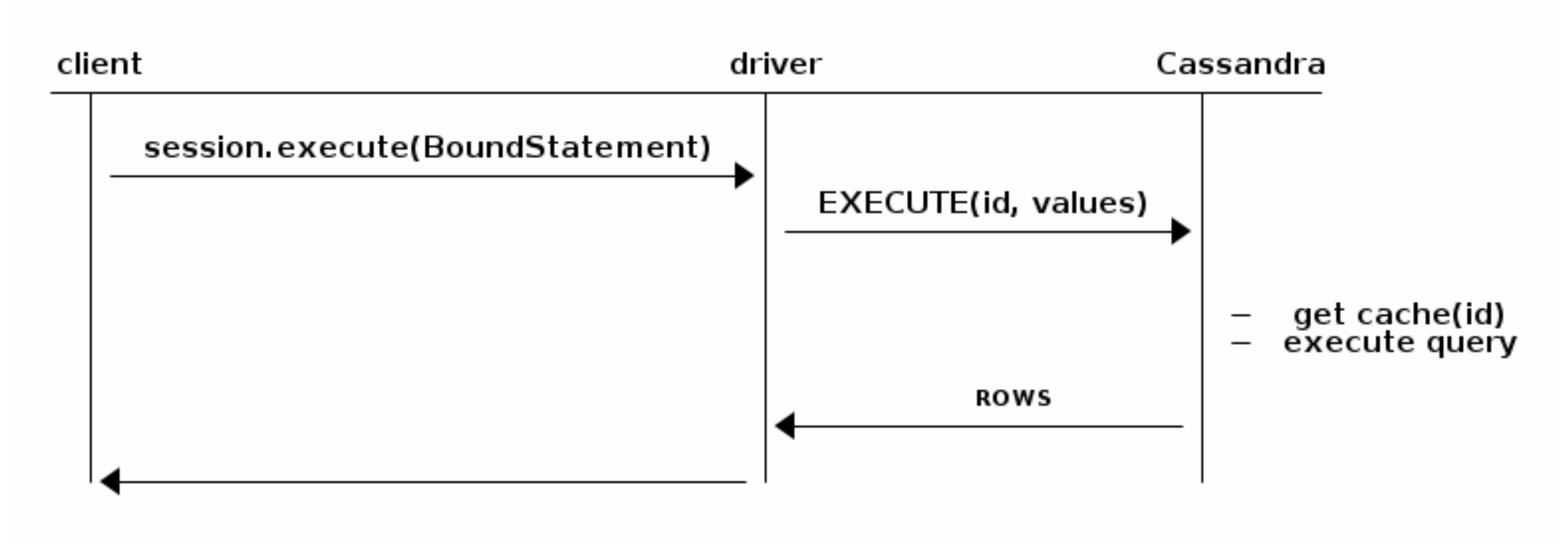
## Prepared statements - logic (1)

- Cassandra stores the cached query on server



## Prepared statements - logic (1)

- For queries, the cache id and raw value are sent



## Prepared statements - advantages

- Saving parsing overhead
- Result set metadata is cached on the driver
  saves bandwidth / resources
- Better CQL data types check
  on driver, not on server
- Calculates the partition key on the driver
- More optimizations

## Prepared statements - example (1)

```
// query using prepared statement (insert)
PreparedStatement pstmt   =
          session.prepare(“INSERT INTO users(user_id, name, age) VALUES(?,?,?);

BoundStatement bstmt = pstmt.bind()
              .setLong(0, 123)
              .setString(1, “Rubi Boim)
              .setInt(2, 21);

session.execute(bstmt);
```

## Prepared statements - example (2)

```
// query using prepared statement (select)
PreparedStatement pstmt   =
          session.prepare(“SELECT * FROM users WHERE user_id=?”);

BoundStatement bstmt = pstmt.bind()
              .setLong(0, 123);

ResultSet rs = session.execute(bstmt);
```