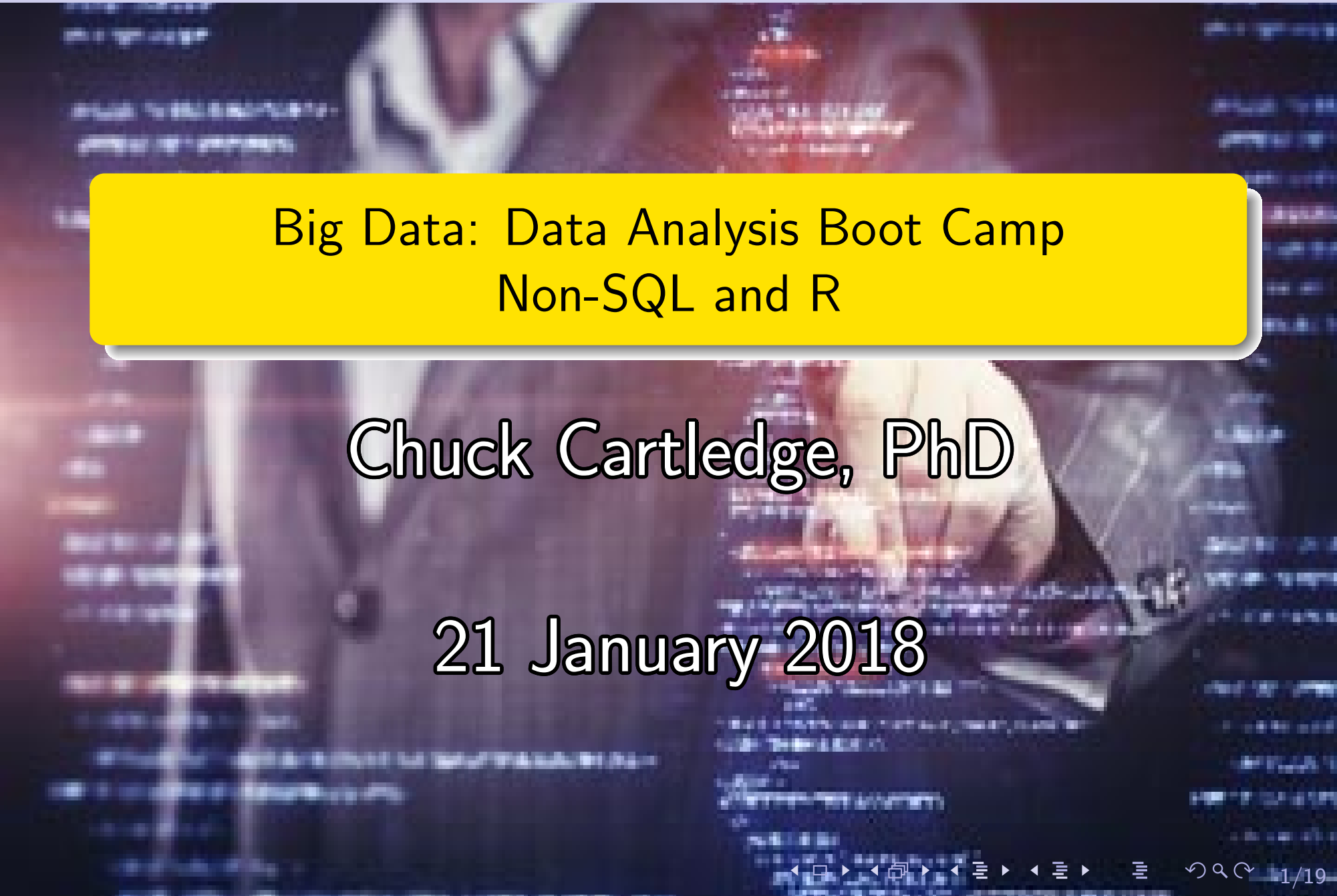


Big Data: Data Analysis Boot Camp
Non-SQL and R

Chuck Cartledge, PhD

21 January 2018

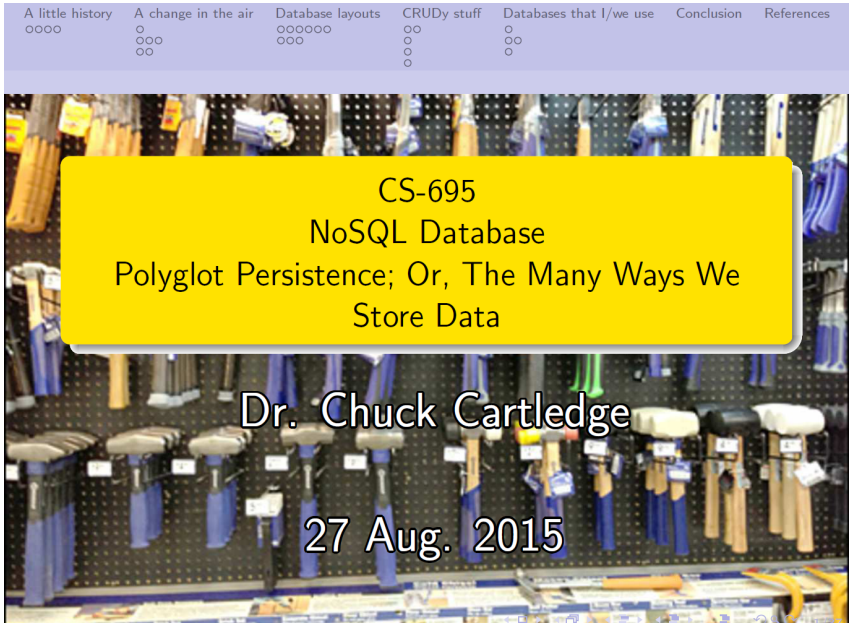
Table of contents (1 of 1)

What are we going to cover?

1. Brief overview of different Non-SQL technologies
2. Revisit our airport service data
3. Ask, and answer some questions about the airports

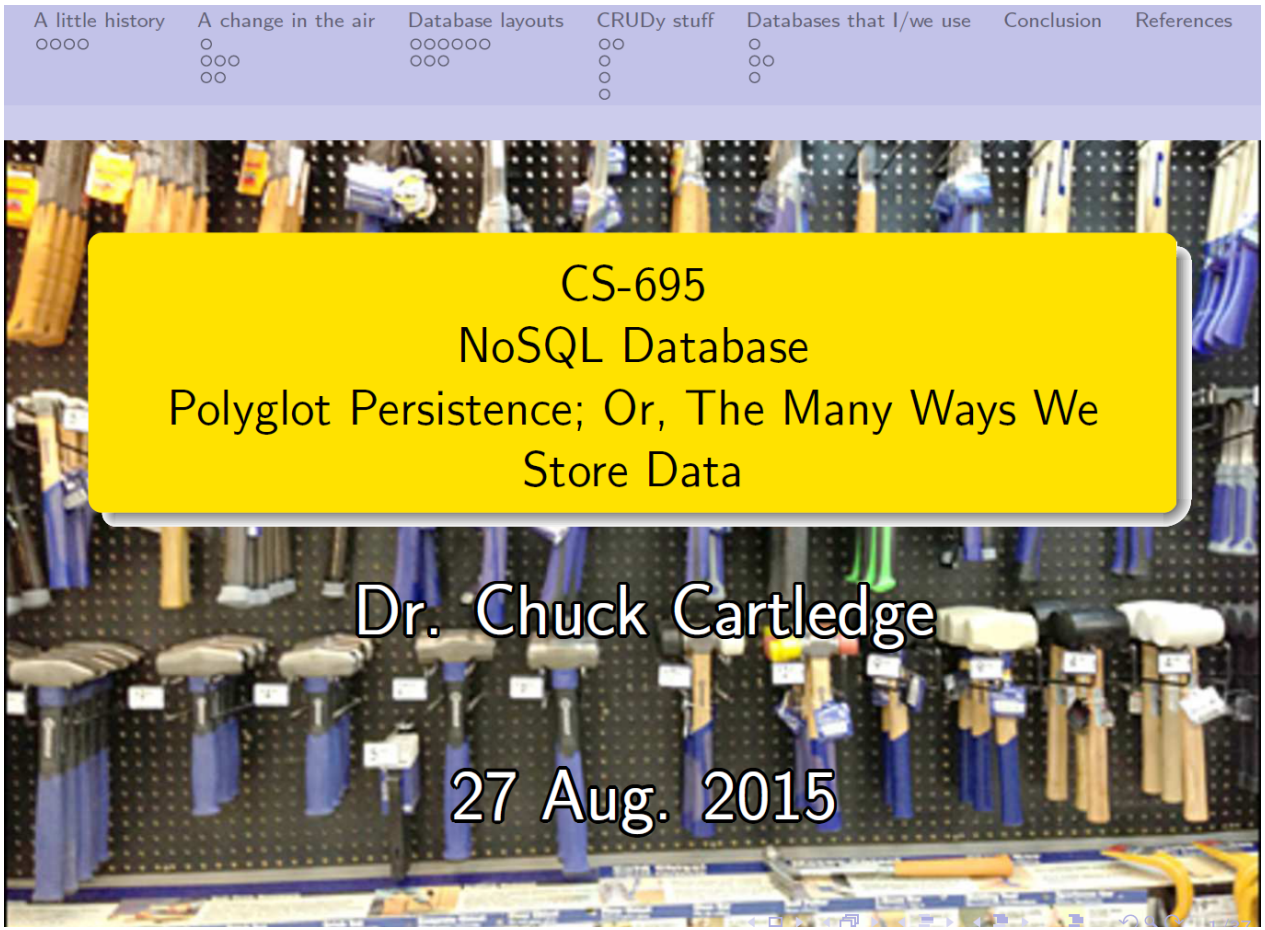
Words from the past.

Bring up the attached Polyglot Persistence presentation.
We'll be looking at pages 1 – 22.



Attached file.

Same image.



Attached file.

Finding a "friend of a friend"

A common question is: who is a friend of a friend?
It comes up in all sorts of relationship type questions. Not only interpersonal; but also organizational, system analysis, law, etc.
Easily answered in some languages, harder in others.

Searching Friends
SQL/MySQL vs. Gremlin/Neo4j

What are the names of Rand Fitzpatrick's friends?[45]

```
mysql> SELECT friend.inV, b._value FROM friend, metadata as a,
       metadata as b WHERE a._key='name' AND
       a._value='Rand Fitzpatrick' AND a.vertex=friend.outV AND
       b.vertex=friend.inV AND b._key='name';
97 rows in set (0.32 sec -- 320.0 ms)

gremlin> g:key('name','Rand Fitzpatrick')/outE[@label='friend']/inV/@name
97 results returned (0.00258 sec -- 25.88 ms)
```

[45] When in cache (through repeated, identical querying), SQL/MySQL evaluates in ∼0.005 seconds (5ms) and Gremlin/Neo4j evaluates in ∼0.0002 seconds (0.2ms).

Image from [1].

Same image.

Searching Friends
SQL/MySQL vs. Gremlin/Neo4j

What are the names of Rand Fitzpatrick's friends?[45]

```
mysql> SELECT friend.inV, b._value FROM friend, metadata as a,
       metadata as b WHERE a._key='name' AND
       a._value='Rand Fitzpatrick' AND a.vertex=friend.outV AND
       b.vertex=friend.inV AND b._key='name';
97 rows in set (0.32 sec -- 320.0 ms)

gremlin> g:key('name','Rand Fitzpatrick')/outE[@label='friend']/inV/@name
97 results returned (0.00258 sec -- 25.88 ms)
```

[45] When in cache (through repeated, identical querying), SQL/MySQL evaluates in ∼0.005 seconds (5ms) and Gremlin/Neo4j evaluates in ∼0.0002 seconds (0.2ms).

Image from [1].

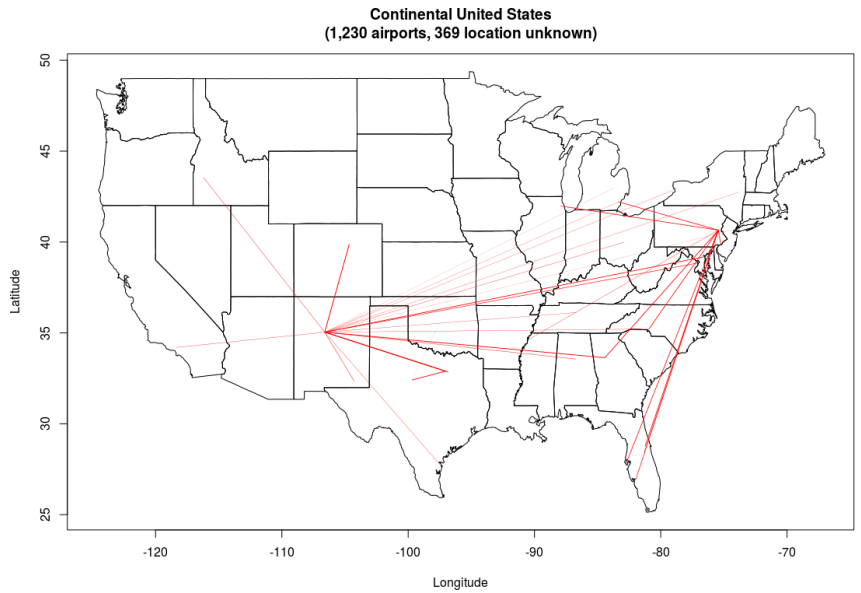
Revisit our airport data.

We're going to look at the airport data in a different way.

Airports become nodes (or vertices)

Service become edges (or arcs)

Load the attached file.



Overview of the program

1. By default the database is reset each time main() is executed

```
resetDB <- TRUE
if (resetDB == TRUE)
{
    d <- initDatabase(graph, limit=NULL)
}
```

2. main() prints the time to find different paths through the database

```
print(system.time(findPath(graph, "CVG","MIA")))
print(system.time(findPath(graph, "MIA","IAH")))
print(system.time(findPath(graph, "MYK","DTR")))
```

3. main() finds the "friend of a friend" from a selected airport

```
print(system.time(findFriendOfFriend(graph, "MIA")))
```

A few database initialization details

1. Need to ensure that Airport nodes are unique

```
addConstraint(graph, "Airport", "name")
```

2. Create the airport location file and load a subset into the database

```
createTextFile(airportLocationFile, airportLocationURL, overwrite=TRUE)
temp <- read.csv(file=airportLocationFile, header=FALSE)
temp <- temp[,c("V5", "V7", "V8", "V9")]
colnames(temp) <- c("name", "lat", "lon", "el")
temp <- temp[!duplicated(temp$name),]
write.table(temp, file=tempFile, append=FALSE, sep=",", col.names=TRUE,
    row.names=FALSE, qmethod=NULL)
command <- sprintf("USING PERIODIC COMMIT 10000 load csv with headers from
    'file://\%s' as line merge (d:Airport {name:line.name, lat:line.lat,
    lon:line.lon, el:line.el})", tempFile)
dumpObject(system.time(cypher(graph, command)), comment="Creating airport
    info nodes")
```

A few database initialization details

1. Resulting in:

```
[1] "Creating airport info nodes --- Dumping the object: system.time(
      cypher(graph, command)) (of type: double, class: proc_time)"
   user  system elapsed
  0.008   0.004   0.675
```

2. The origin and destination data is loaded and cleaned

```
unzip(flightDataZipFileName, files=flightDataFileName, exdir=tempDir)
unzipFileName <- file.path(tempDir, flightDataFileName)
print(sprintf("Copying \%s to local file system directory: \%s",
      unzipFileName, tempDir))
temp <- read.csv(file=unzipFileName, header=TRUE, sep=",")
temp <- temp[-8]
df <- data.frame(temp$ORIGIN,temp$DEST)
colnames(df) <- c("ORIGIN", "DEST")
originNumberOfRows <- nrow(df)
df <- df[!duplicated(df),]
```

A few database initialization details

1. Chunks of data are loaded

```
for (i in 1:(length(chunks) - 1)) {
    write(x=c('"src","dest"', paste0(df\$ORIGIN[chunks[i]:(chunks[i+1] -
        1)], ",", df\$DEST)[chunks[i]:(chunks[i+1] - 1)]), file=tempFile)
    command <- sprintf("USING PERIODIC COMMIT 10000 load csv with headers
        from 'file://\%s' as line merge (s:Airport {name:line.src}) merge
         (d:Airport {name:line.dest})", tempFile)
    dumpObject(system.time(cypher(graph, command)), comment=sprintf("
        Creating nodes (pass \%.0f of \%.0f)", i , (length(chunks) - 1)))
    command <- sprintf("USING PERIODIC COMMIT 10000 load csv with headers
        from 'file://\%s' as line match (s:Airport {name:line.src}), (d:
        Airport {name:line.dest}) create (s) -[r:Services {edge:1} ]-> (d
        )", tempFile)
    dumpObject(system.time(cypher(graph, command)), comment="Creating
        relationships") }
```

2. Resulting in:

```
[1] "Creating relationships -- Dumping the object: system.time(cypher(
    graph, command)) (of type: double, class: proc_time)"
   user  system elapsed
  0.016   0.000   0.419
```

The database is now ready for use.

Ways to modify the Airport program.

A CYPHER[1] statement or R[2] can be used to query or modify the database, and R can be used for the numeric heavy lifting.

- Use distance between airports as a metric to find the "diameter" of the graph.
- Find the connectiveness (degree) distribution of the airports.
- Use an airport's connections (degreeness) to identify the "most important" airport (may not be the one with the highest degree).
- Find the path between "interesting" airports, and then remove an airport along the path. Is there another path from the source to the destination?
- Update the missing location information.

---

[1] `https://neo4j.com/docs/developer-manual/current/cypher/`

[2] ls("package:RNeo4j")

Good and not so good

Strengths:

- A graph database — typeless, schemaless, unstructured relationships
- Large capacity (˜34.4 billion nodes, and relationships)
- ReSTful interfaces — means lots of different language support

Weaknesses:

- Graph terminology is not consistent — node vs. vertex, arc vs. edge, etc.
- Sharding is not supported
- Licensing may be an issue for production applications

Good for, and not so good for

Good fit;

- Anything that can be represented as a "social graph"
- Any "link rich" domain
- Routing, dispatch, and location based services (getting from A to B)
- Recommendation engines ("also bought" statements)

Not so good fit:

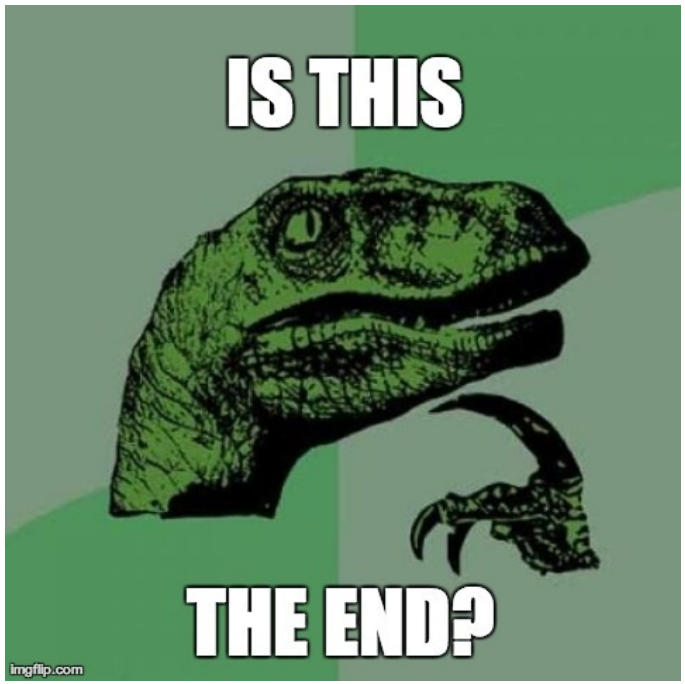
- When updating "all" items in a DB (requires total graph traversal)

Q & A time.

Q: How many marketing people does it take to change a light bulb?
A: I'll have to get back to you on that.

What have we covered?

- Talked about different types of No-SQL database technologies and what they are good for
- "Played" with the airport service data as a graph database
- Asked and answered some questioned geared towards graph database technology

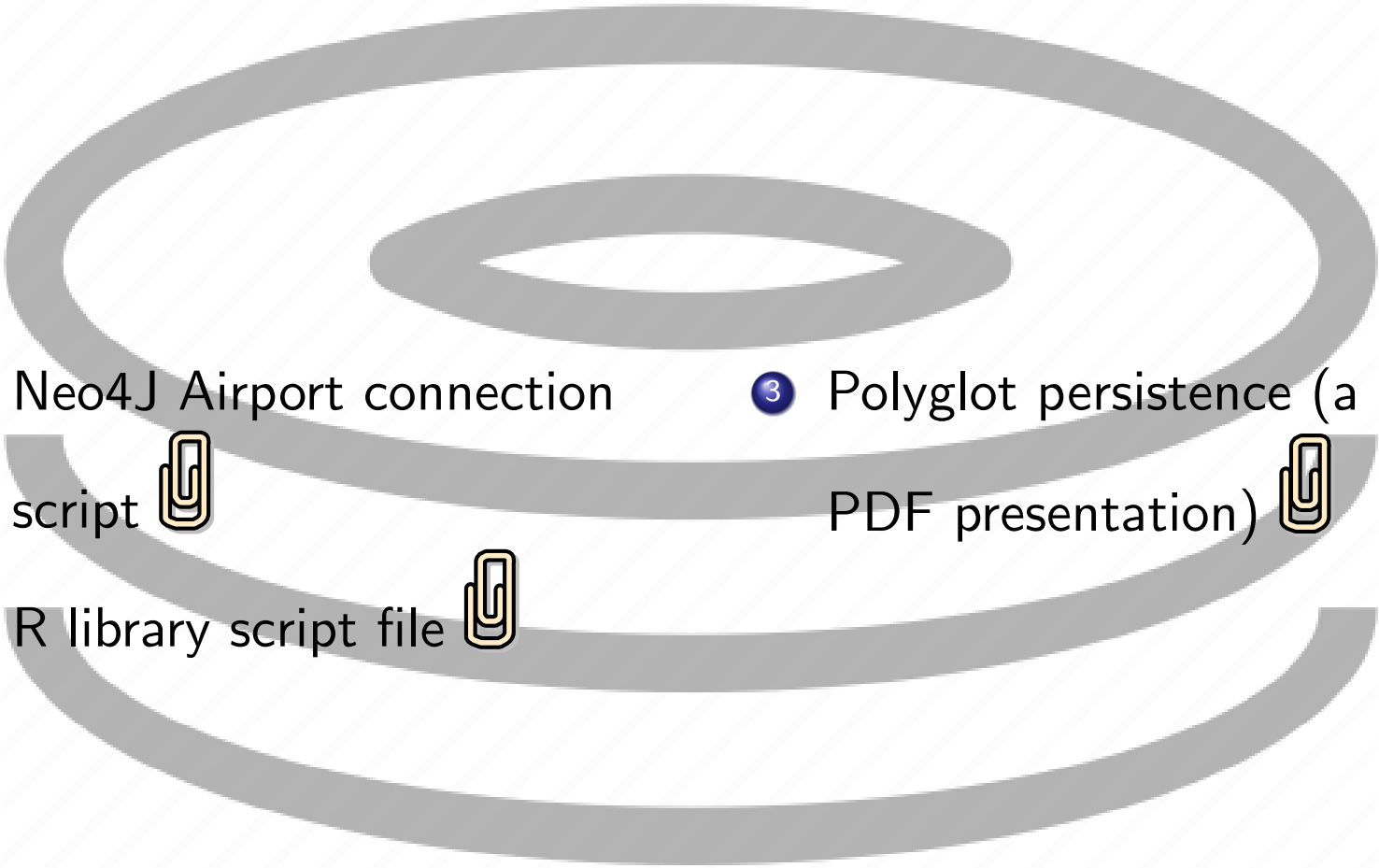


Next: Looking at crime data

References (1 of 1)

[1] Marko A. Rodriguez, Problem-Solving using Graph Traversals, https://www.slideshare.net/slidarko/problemsolving-using-graph-traversals-searching-scoring-ranking-and-recommendation/88-Searching_Friends_SQLMySQL_vs_GremlinNeo4jWhat, 2010.

Files of interest



1. Neo4J Airport connection script
2. R library script file
3. Polyglot persistence (a PDF presentation)