

Preliminaries

Reminder: we need to take attendance

- Yasmin Lucero: data scientist
- Dia&Co: a series B start-up in fashion space

Dia & Co



Hour 1

6:00 - 6:10 introduction/preliminaries

6:10 - 6:20 Lecture 1: data science architecture

6:20 - 6:40 Lecture 2: relational databases

6:40 - 7:00 Exercise 1: SQL CRUD

Hour 2

7:00 - 7:10 Break

7:10 - 7:25 Lecture 3: database trade-offs, noSQL, CAP theorem

7:25 - 7:40 Exercise 2: Redis CRUD

7:40 - 7:50 Lecture 4: crucial practicalities: JDBC Drivers, connection strings, cli, IDEs and libraries

7:50 - 8:05 Exercise 3: Mongo CRUD

Hour 3

8:05 - 8:15 Break

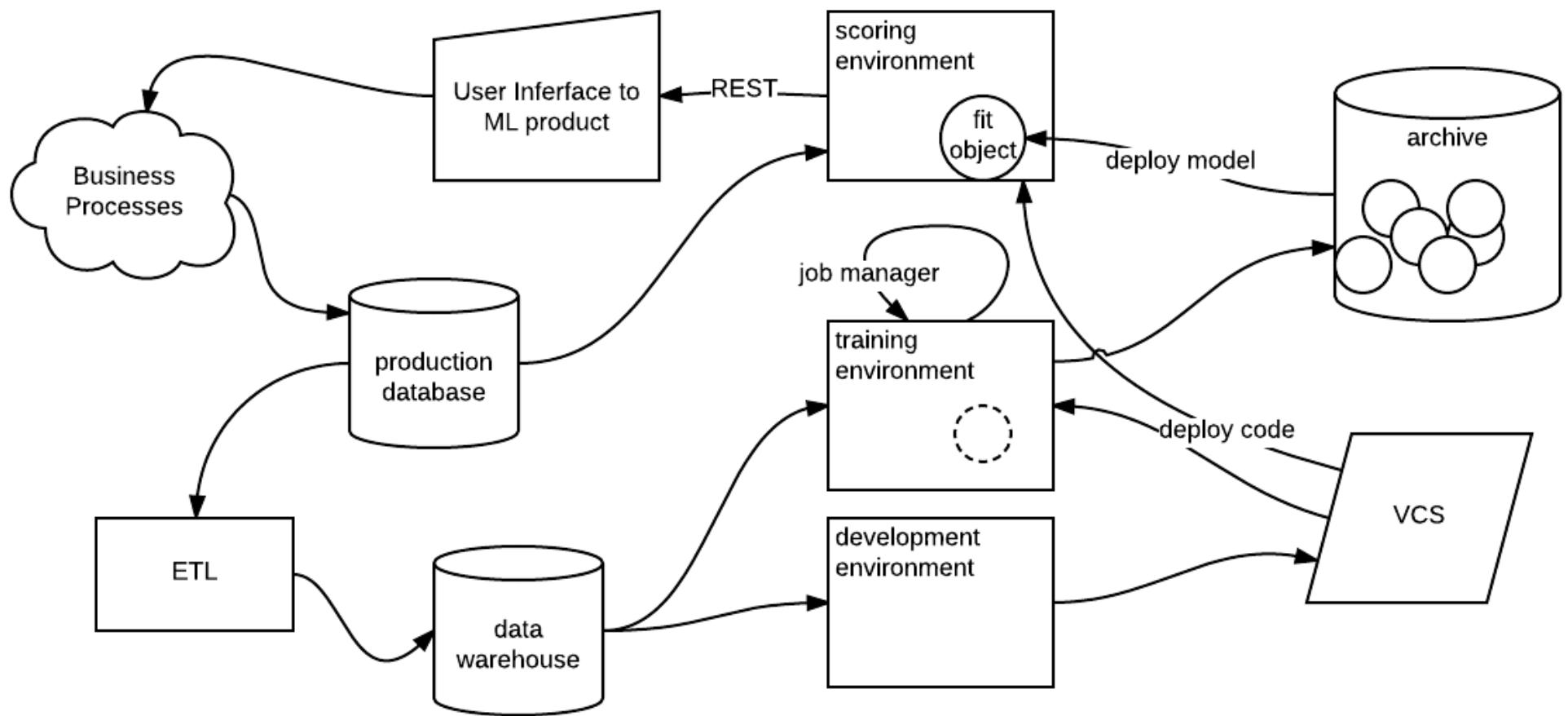
8:15 - 8:45 Lecture 5: Distributed computing, Hadoop, Spark, Hive, Pig

8:45 - 8:50 Wrap up

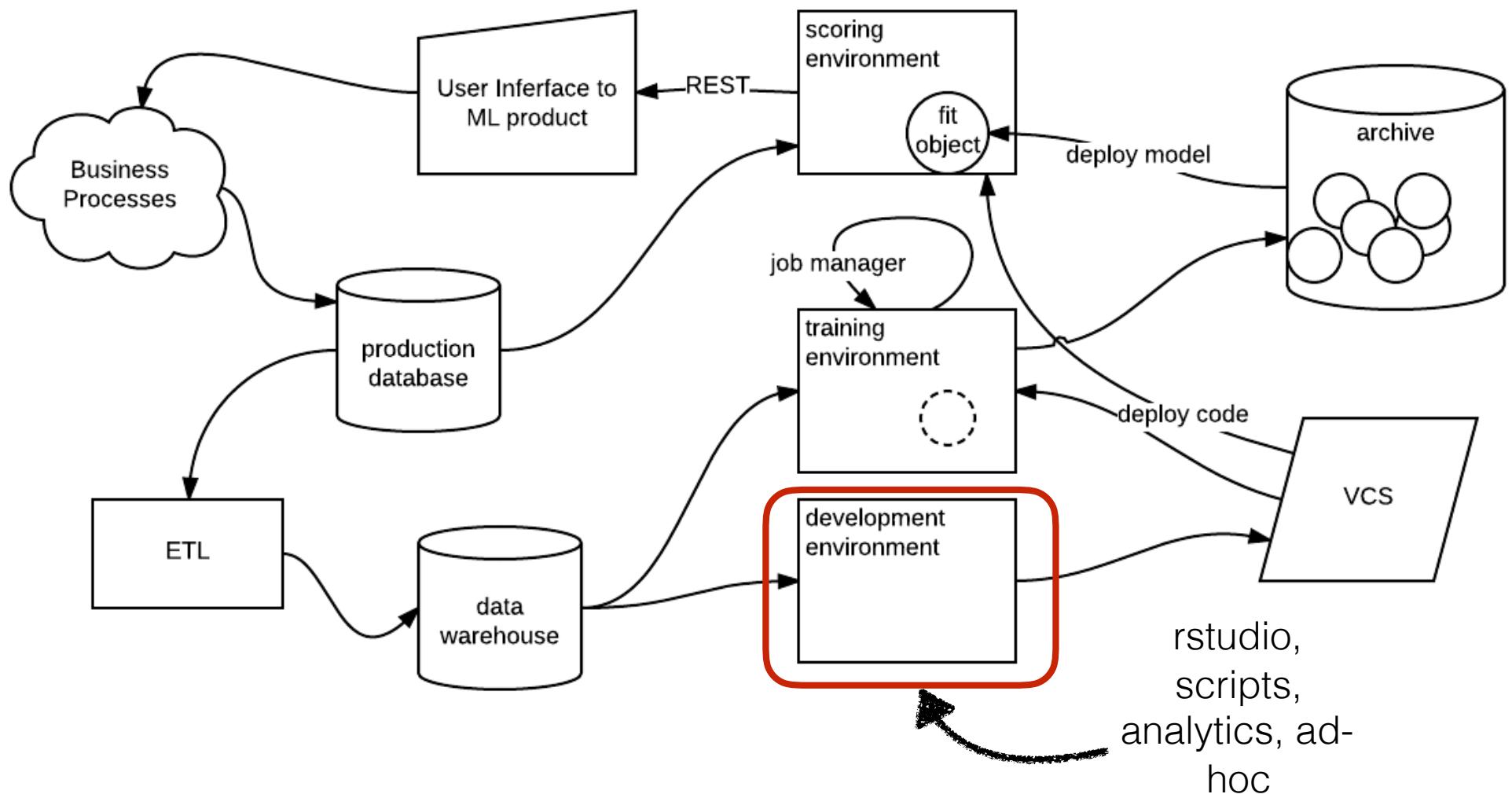
Lecture 1: data science platform architecture

- why should a data scientist care about data stores?

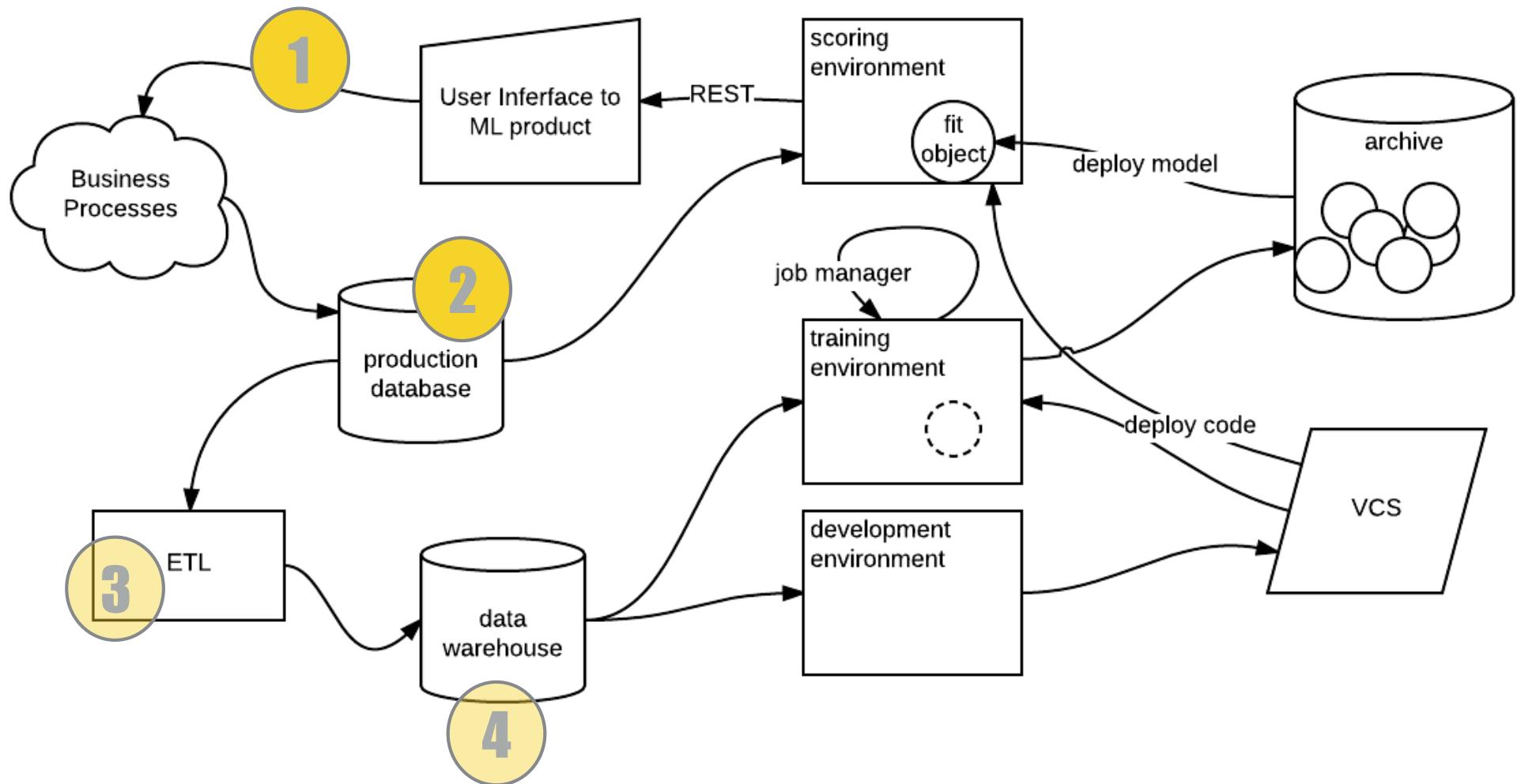
Data science within the production ecosystem



To date, most of your data science life has been within the ‘development environment’



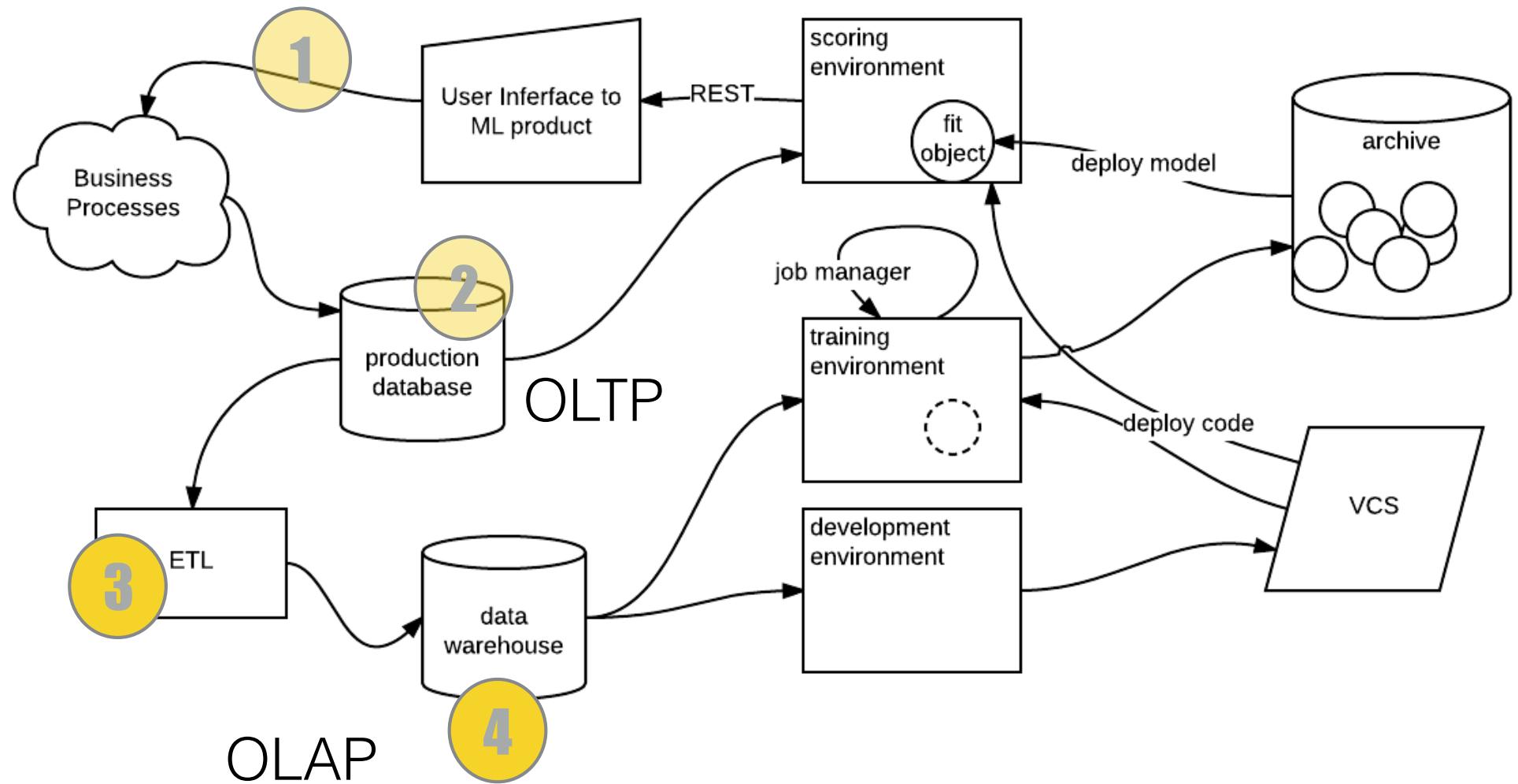
‘event data’ captured in write-optimized data store



event log data

@timestamp	_source
15:29:11.468 CET	▶ message: {"EventTime":"2016-01-28 15:29:00","Hostname":"onyxia","Keywords":36028797018963968,"EventType":"INFO","SeverityValue":2,"Severity":"INFO","EventID":82 czasu bezczynności. ","Opcode":"Informacje","EventReceivedTime":1453991341,"SourceModuleName":"in","SourceModuleType":"im_msvisatalog"} @version: 1 host: 192.168.1.40 type: WindowsEventLog EventTime: 2016-01-28 15:29:00 Hostname: onyxia Keywords: 36028797018963970 EventType: IN przekroczenia limitu czasu bezczynności. Opcode: Informacje EventReceivedTime: 1453991341 SourceModuleName: in SourceModuleType: im_msvisatalog @version: 1 host: 192.168.1.40 type: WindowsEventLog EventTime: 2016-01-28 15:26:17 Hostname: onyxia Keywords: 36028797018963970 EventType: IN 15:28:40.444 CET A5F9-F2BDFEA0F156}","Version":0,"Task":0,"OpcodeValue":0,"RecordNumber":3050,"ProcessID":0,"ThreadID":0,"Channel":Application,"Message":Usługa ochrony oprogramowania została zatrzymana. @version: 1 host: 192.168.1.40 type: WindowsEventLog EventTime: 2016-01-28 15:26:17 Hostname: onyxia Keywords: 36028797018963970 EventType: IN A5F9-F2BDFEA0F156} RecordNumber: 3050 Channel: Application Message: Usługa ochrony oprogramowania została zatrzymana. EventReceivedTime: 1453991479 SourceModuleName: in SourceModuleType: im_msvisatalog

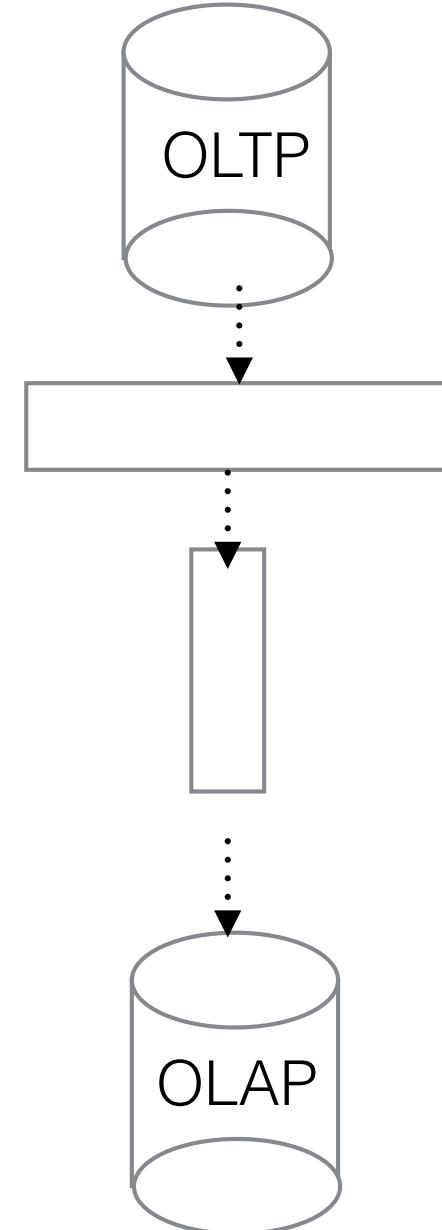
- timestamp
- event type
- user id
- page url
- page data
- user agent string
- ip
- Etc

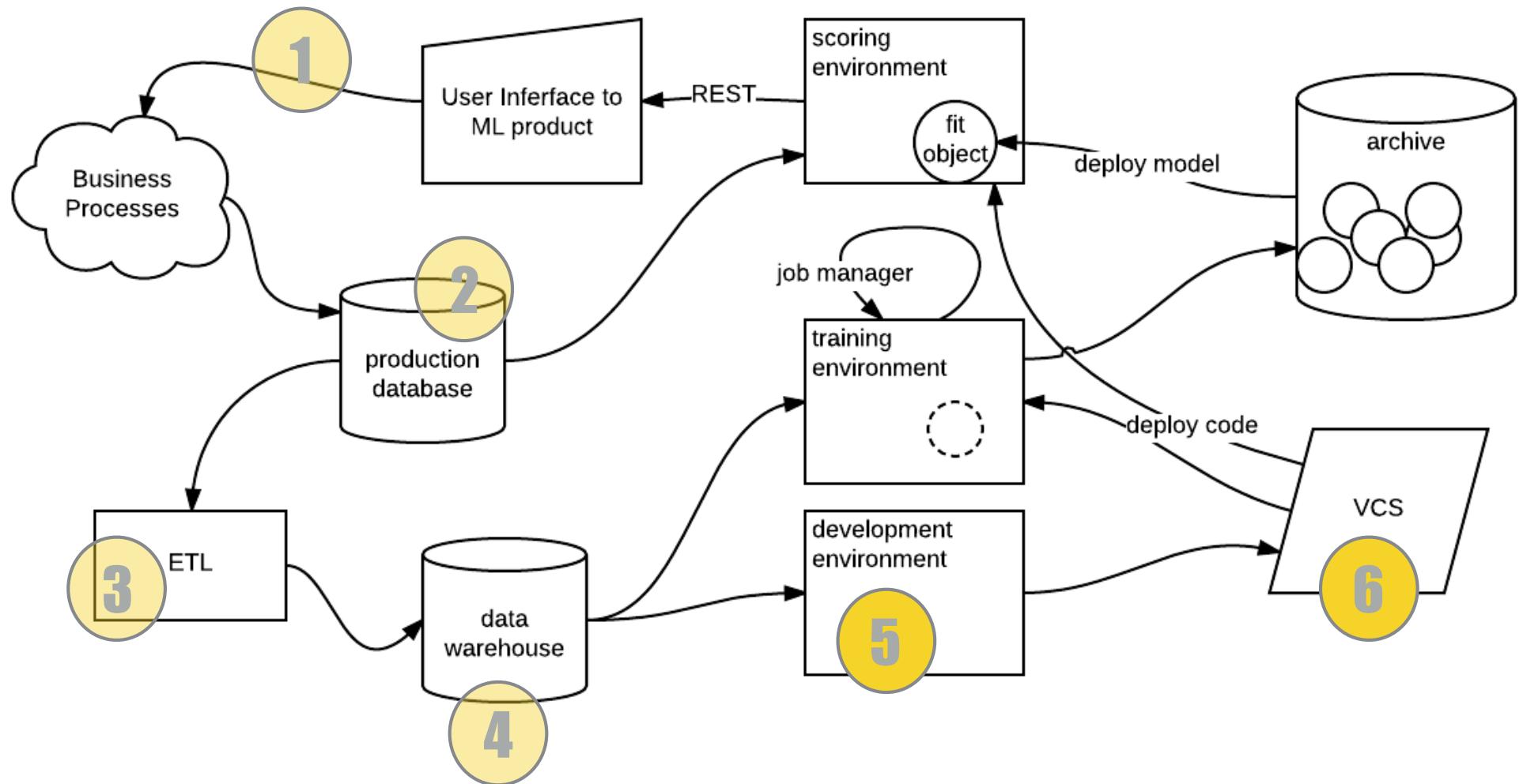


Extract

Transform

Load



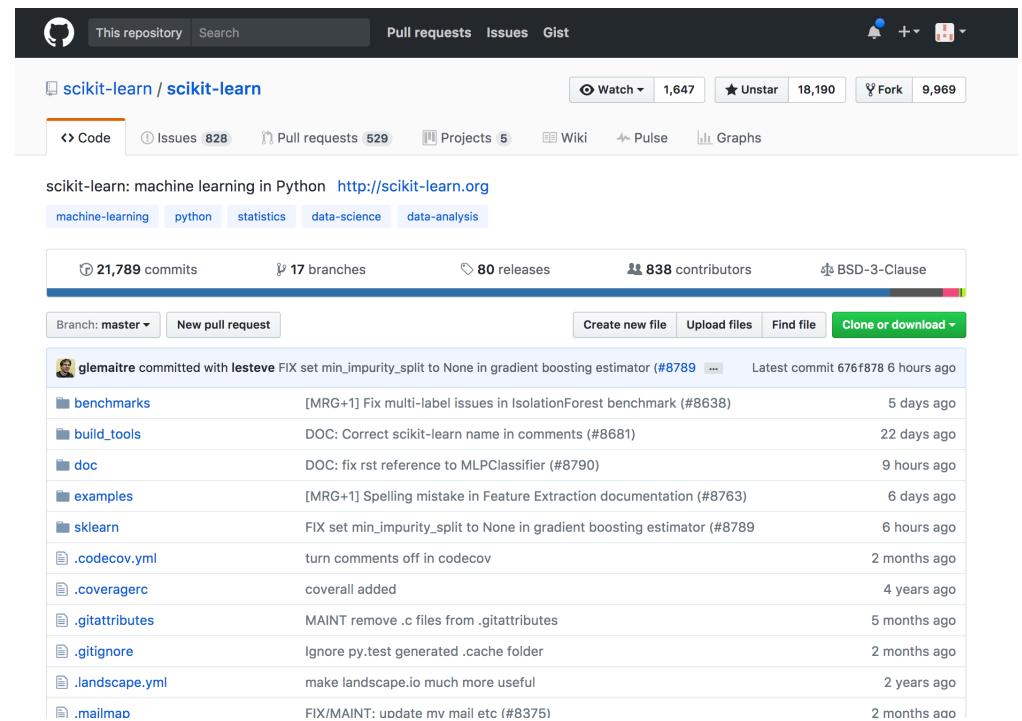


VCS

Version Control System

these days, this
mostly means git used
with GitHub

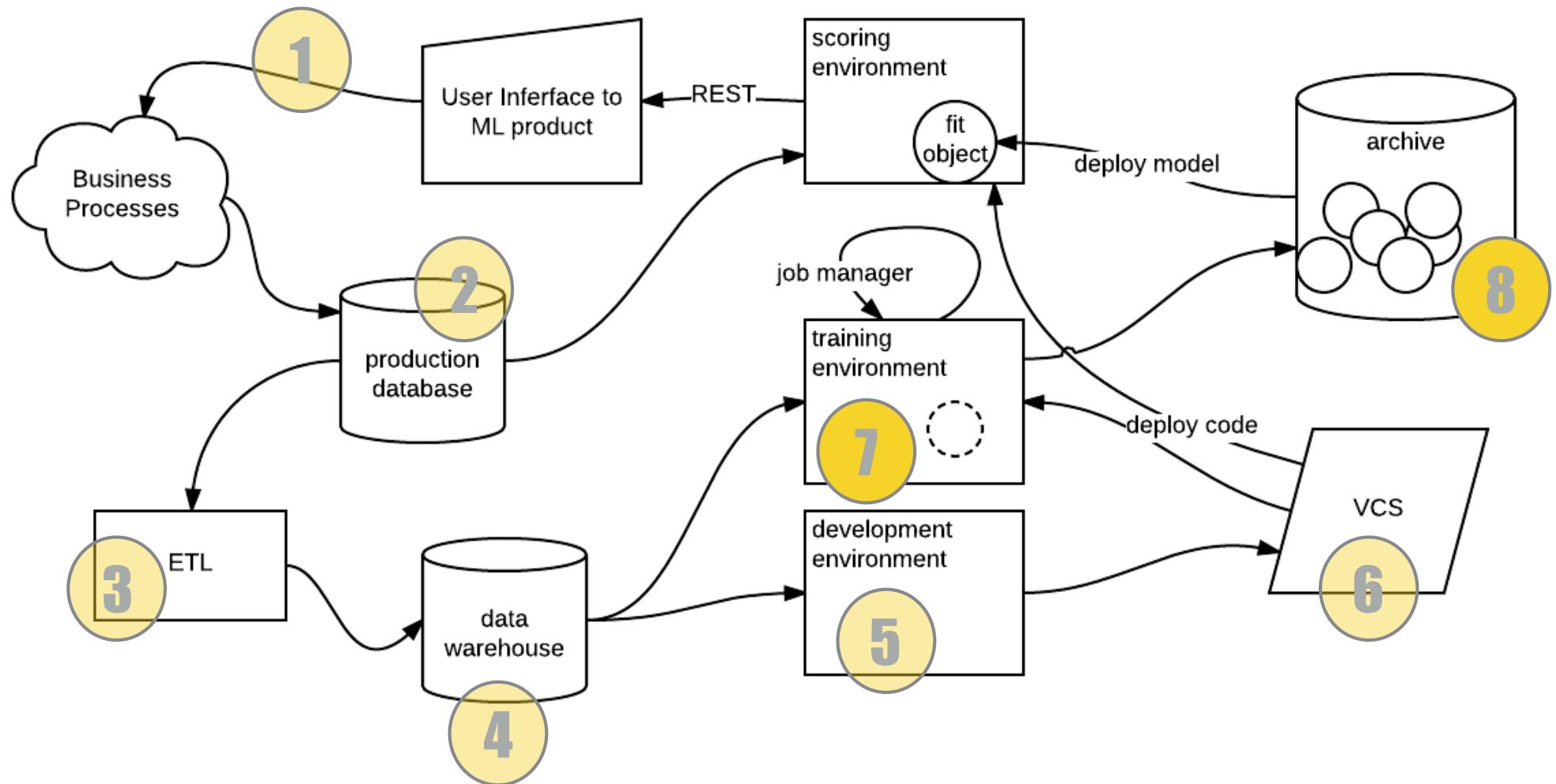
but also
mercurial,
subversion (svn),
others...



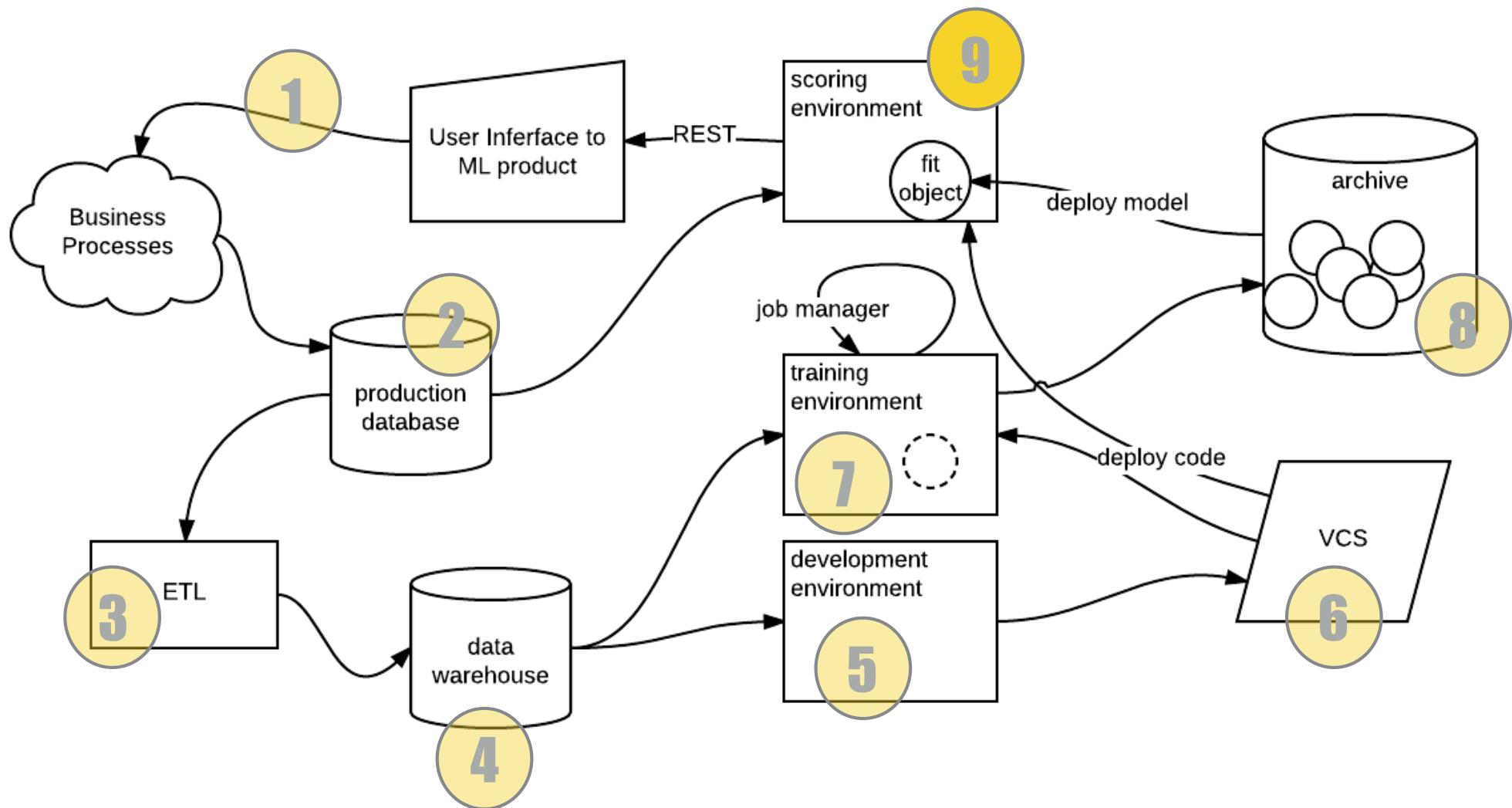
The screenshot shows the GitHub repository page for 'scikit-learn / scikit-learn'. The page includes a navigation bar with 'This repository', 'Search', 'Pull requests', 'Issues', and 'Gist'. Below the navigation bar, there are buttons for 'Code', 'Issues 828', 'Pull requests 529', 'Projects 5', 'Wiki', 'Pulse', and 'Graphs'. The repository name 'scikit-learn / scikit-learn' is displayed, along with statistics: 1,647 watches, 18,190 stars, 9,969 forks, and a BSD-3-Clause license. A commit history table lists 21,789 commits, 17 branches, and 80 releases. The table includes columns for the author, commit message, file, and date. The most recent commit is by 'glemaître' with the message 'FIX set min_impurity_split to None in gradient boosting estimator (#8789)'. The commit was made 6 hours ago.

Author	Commit Message	Date
glemaître	FIX set min_impurity_split to None in gradient boosting estimator (#8789)	6 hours ago
	...	
	Latest commit 676f878 6 hours ago	
benchmarks	[MRG+1] Fix multi-label issues in IsolationForest benchmark (#8638)	5 days ago
build_tools	DOC: Correct scikit-learn name in comments (#8681)	22 days ago
doc	DOC: fix rst reference to MLPClassifier (#8790)	9 hours ago
examples	[MRG+1] Spelling mistake in Feature Extraction documentation (#8763)	6 days ago
sklearn	FIX set min_impurity_split to None in gradient boosting estimator (#8789)	6 hours ago
.codecov.yml	turn comments off in codecov	2 months ago
.coveragerc	coverall added	4 years ago
.gitattributes	MAINT remove .c files from .gitattributes	5 months ago
.gitignore	Ignore py.test generated .cache folder	2 months ago
.landscape.yml	make landscape.io much more useful	2 years ago
.mailmap	FIX/MAINT: update my mail etc (#8375)	2 months ago

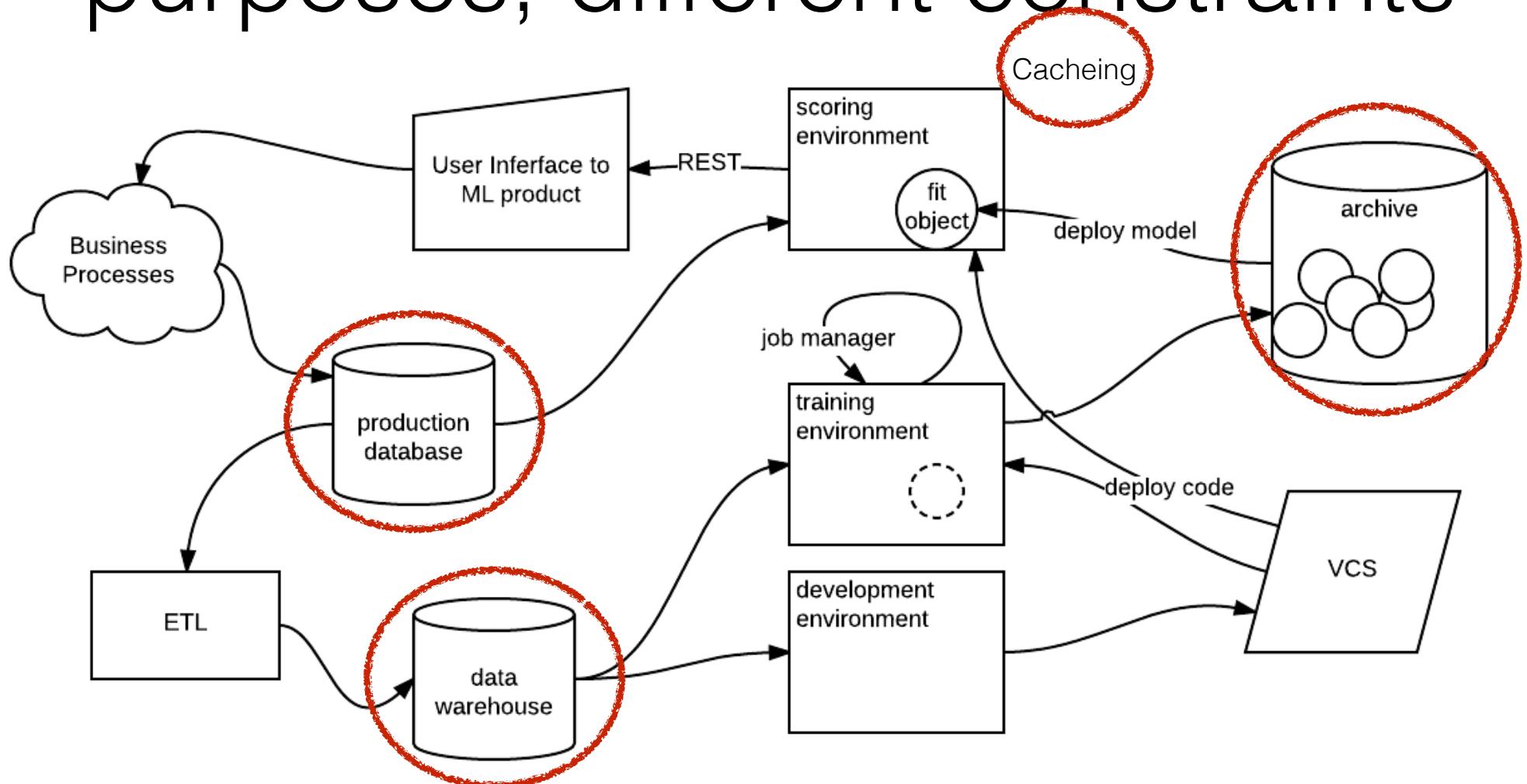
training in production



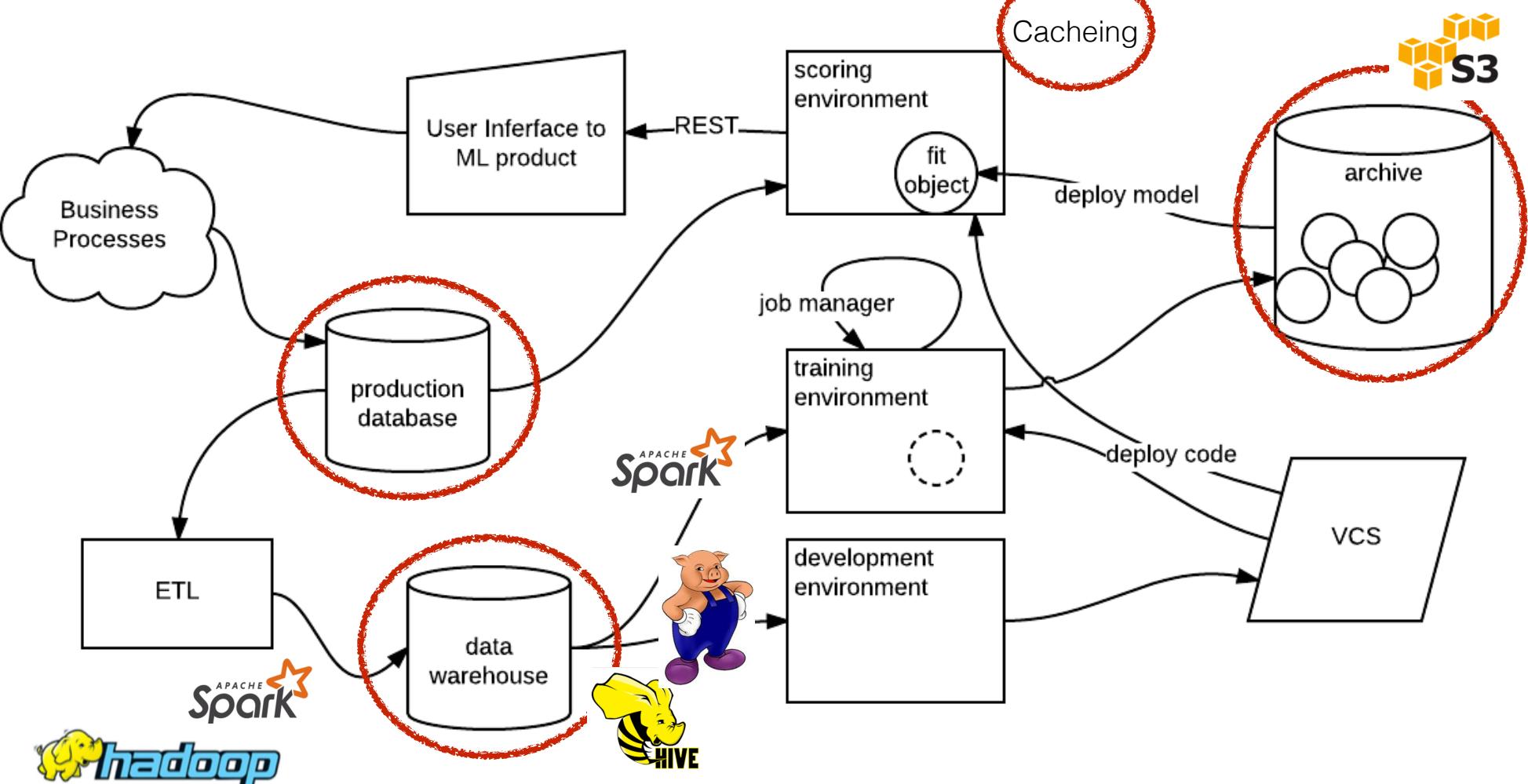
scoring service



several data stores, different purposes, different constraints



also, data management technologies



to sum up

- data scientists ingest data from various data bases
- data scientists write data to various data stores
- data scientists study data that was collected and transformed in ways that matter
- to do your job effectively, you need broad level understanding of these technologies and the problems they solve

Hour 1

6:00 - 6:10 introduction/preliminaries

6:10 - 6:20 Lecture 1: data science architecture

6:20 - 6:40 Lecture 2: relational databases

6:40 - 7:00 Exercise 1: SQL CRUD

Hour 2

7:00 - 7:10 Break

7:10 - 7:25 Lecture 3: database trade-offs, noSQL, CAP theorem

7:25 - 7:40 Exercise 2: Redis CRUD

7:40 - 7:50 Lecture 4: crucial practicalities: JDBC Drivers, connection strings, cli, IDEs and libraries

7:50 - 8:05 Exercise 3: Mongo CRUD

Hour 3

8:05 - 8:15 Break

8:15 - 8:45 Lecture 5: Distributed computing, Hadoop, Spark, Hive, Pig

8:45 - 8:50 Wrap up

Lecture 2:

Relational Databases

- RDBMS: relational database management system
- SQL: structured query language
- Schema: *formal* description of data organization
- CRUD: create, retrieve, update, delete

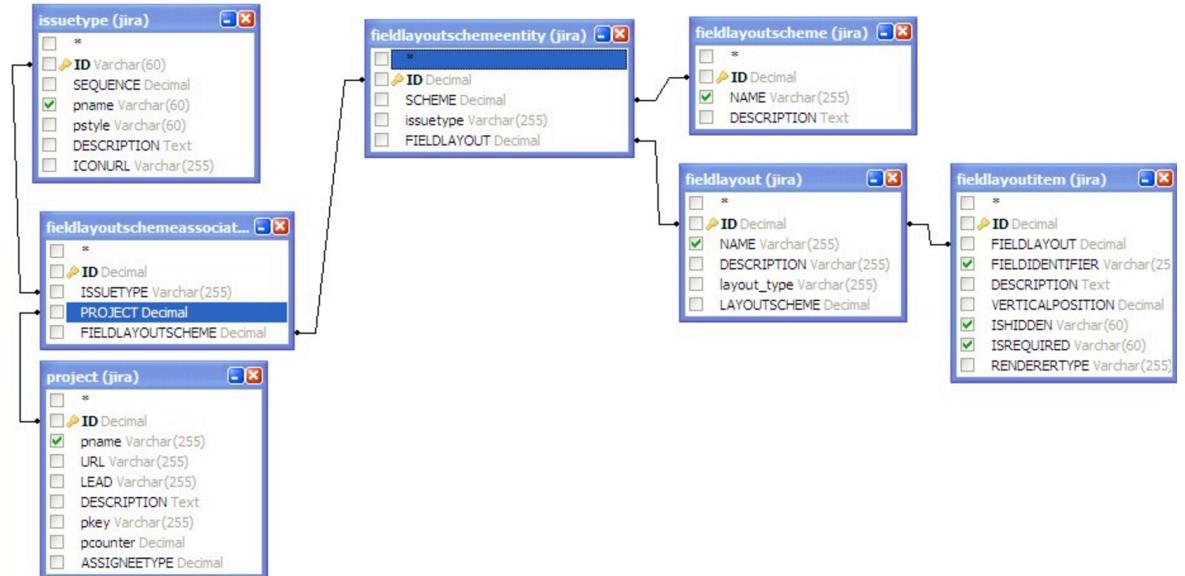
the Relational Model

- familiar because R's data frame is a relational table
- relations are tables of rows and columns
- rows have an id (key)
- columns are fields
- each record (row) is a tuple with a key (id)
- schema defines field and their type

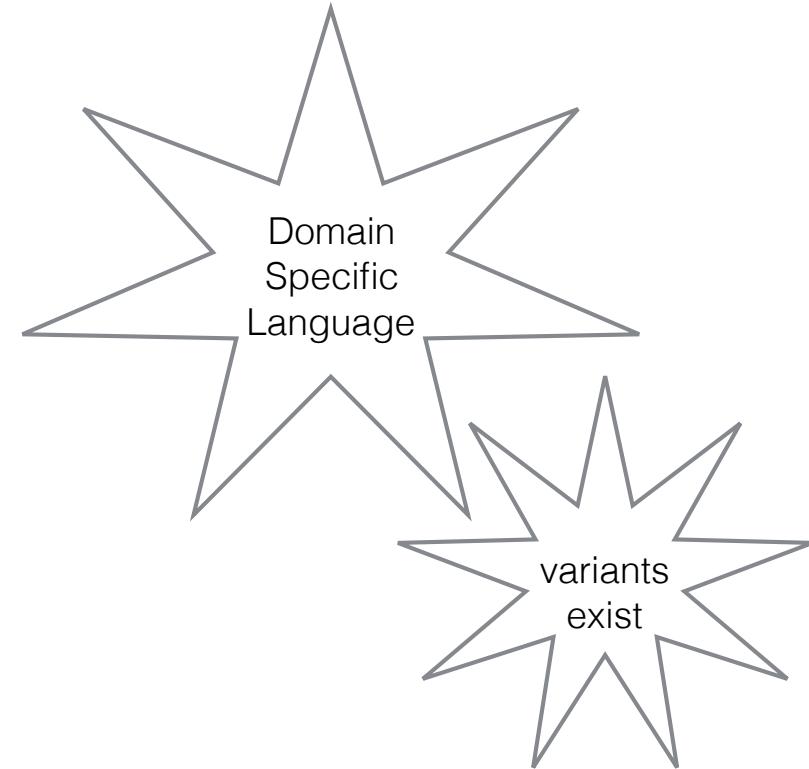
schema: formal description of data organization

Relational db data model

- field_name: type
- Id, foreign/primary



Structured Query Language



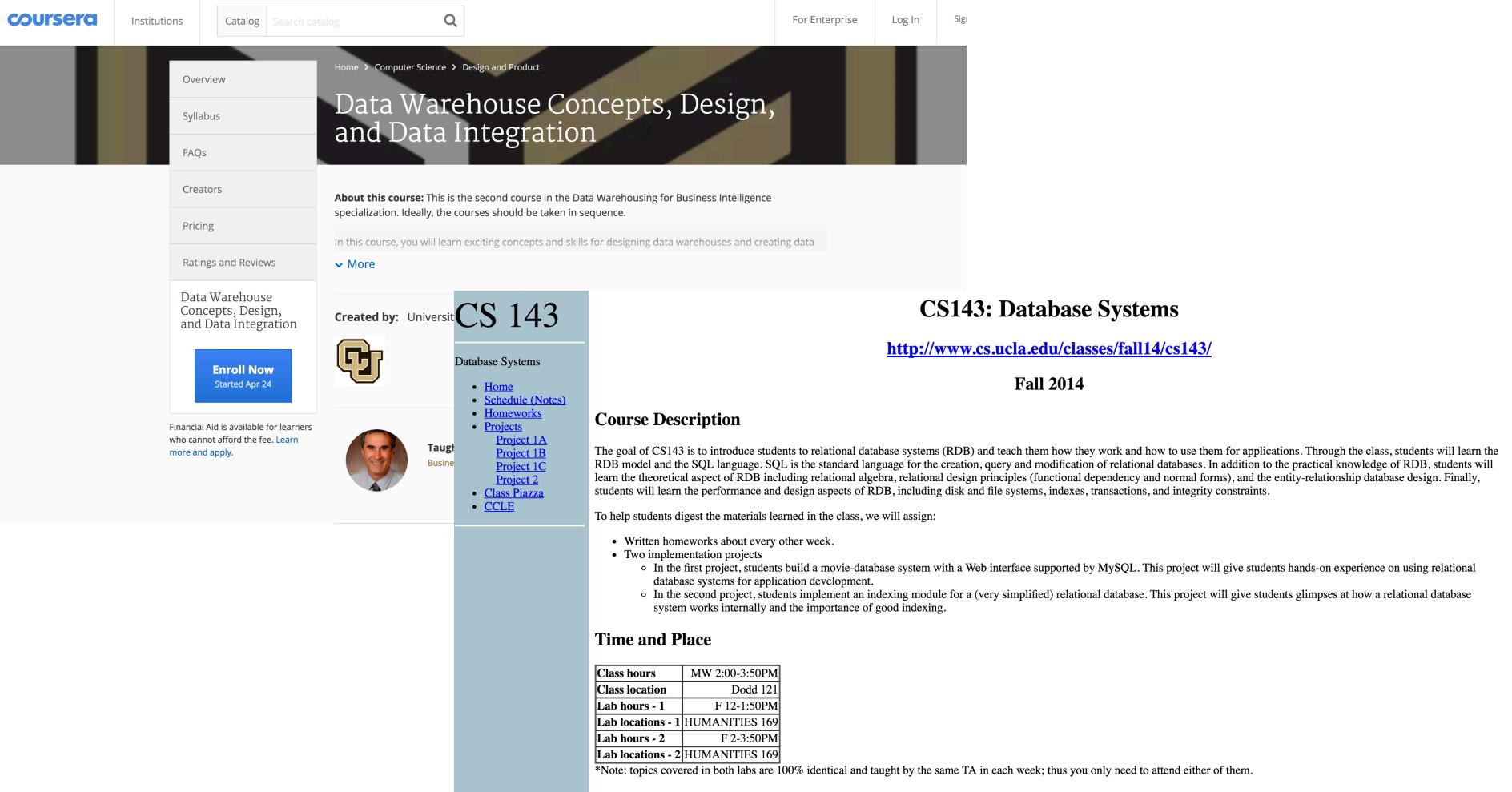
relational databases



ORACLE[®]



not a bad idea to take a databases course



The screenshot shows a Coursera course page for 'Data Warehouse Concepts, Design, and Data Integration' (CS 143). The page includes a sidebar with links for Overview, Syllabus, FAQs, Creators, Pricing, Ratings and Reviews, and a prominent 'Enroll Now' button. The main content area features a large image of a staircase, course details, and a 'Course Description' section. The 'Course Description' section includes a table of contents for the class.

Course Details:

- Created by:** University of California, Los Angeles (UCLA)
- CS 143**
- Database Systems**
- Course Description:** The goal of CS143 is to introduce students to relational database systems (RDB) and teach them how they work and how to use them for applications. Through the class, students will learn the RDB model and the SQL language. SQL is the standard language for the creation, query and modification of relational databases. In addition to the practical knowledge of RDB, students will learn the theoretical aspect of RDB including relational algebra, relational design principles (functional dependency and normal forms), and the entity-relationship database design. Finally, students will learn the performance and design aspects of RDB, including disk and file systems, indexes, transactions, and integrity constraints.
- Assignments:**
 - Written homeworks about every other week.
 - Two implementation projects
 - In the first project, students build a movie-database system with a Web interface supported by MySQL. This project will give students hands-on experience on using relational database systems for application development.
 - In the second project, students implement an indexing module for a (very simplified) relational database. This project will give students glimpses at how a relational database system works internally and the importance of good indexing.
- Time and Place:**

Class hours	MW 2:00-3:50PM
Class location	Dodd 121
Lab hours - 1	F 12-1:50PM
Lab locations - 1	HUMANITIES 169
Lab hours - 2	F 2-3:50PM
Lab locations - 2	HUMANITIES 169
- Note:** topics covered in both labs are 100% identical and taught by the same TA in each week; thus you only need to attend either of them.

SQL learning resources

- SQL Zoo: <https://sqlzoo.net/>
- Learn SQL the Hard Way (\$20): <https://learncodethehardway.org/sql/>
- Head First SQL: <http://www.headfirstlabs.com/books/hfsql/>

Introducing the `world` table of countries

1.

The example uses a `WHERE` clause to show the population of 'France'. Note that strings (pieces of text that are data) should be in 'single quotes'.

Modify it to show the population of Germany

```
SELECT population FROM world  
WHERE name = 'France'
```

Submit SQL

Restore default

Wrong answer. Some of the data is incorrect.

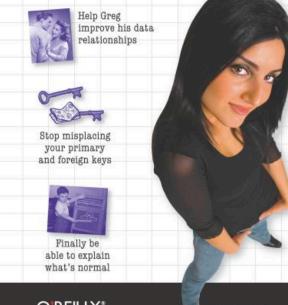
population
65906000

Show what the answer should be...



A Brain-Friendly Guide

Head First
SQL



O'REILLY®

Copyrighted Material

Lynn Beighley

Create

Read

Update

Delete

Create

CREATE TABLE

Read

SELECT * FROM T1

Update

INSERT INTO

Delete

DROP TABLE

Exercise 1: SQL CRUD

Hour 1

6:00 - 6:10 introduction/preliminaries

6:10 - 6:20 Lecture 1: data science architecture

6:20 - 6:40 Lecture 2: relational databases

6:40 - 7:00 Exercise 1: SQL CRUD

Hour 2

7:00 - 7:10 Break

7:10 - 7:25 Lecture 3: database trade-offs, noSQL, CAP theorem

7:25 - 7:40 Exercise 2: Redis CRUD

7:40 - 7:50 Lecture 4: crucial practicalities: JDBC Drivers, connection strings, cli, IDEs and libraries

7:50 - 8:05 Exercise 3: Mongo CRUD

Hour 3

8:05 - 8:15 Break

8:15 - 8:45 Lecture 5: Distributed computing, Hadoop, Spark, Hive, Pig

8:45 - 8:50 Wrap up

Lecture 3: Data Store Diversity noSQL, redis and mongo

Q: why so many?

A: trade-offs



Data Store Typology

1. relational
2. columnar
3. key-value store
4. document oriented
5. graph

The
Pragmatic
Programmers

Seven Databases in Seven Weeks

A Guide to Modern Databases
and the NoSQL Movement

Eric Redmond
and Jim R. Wilson

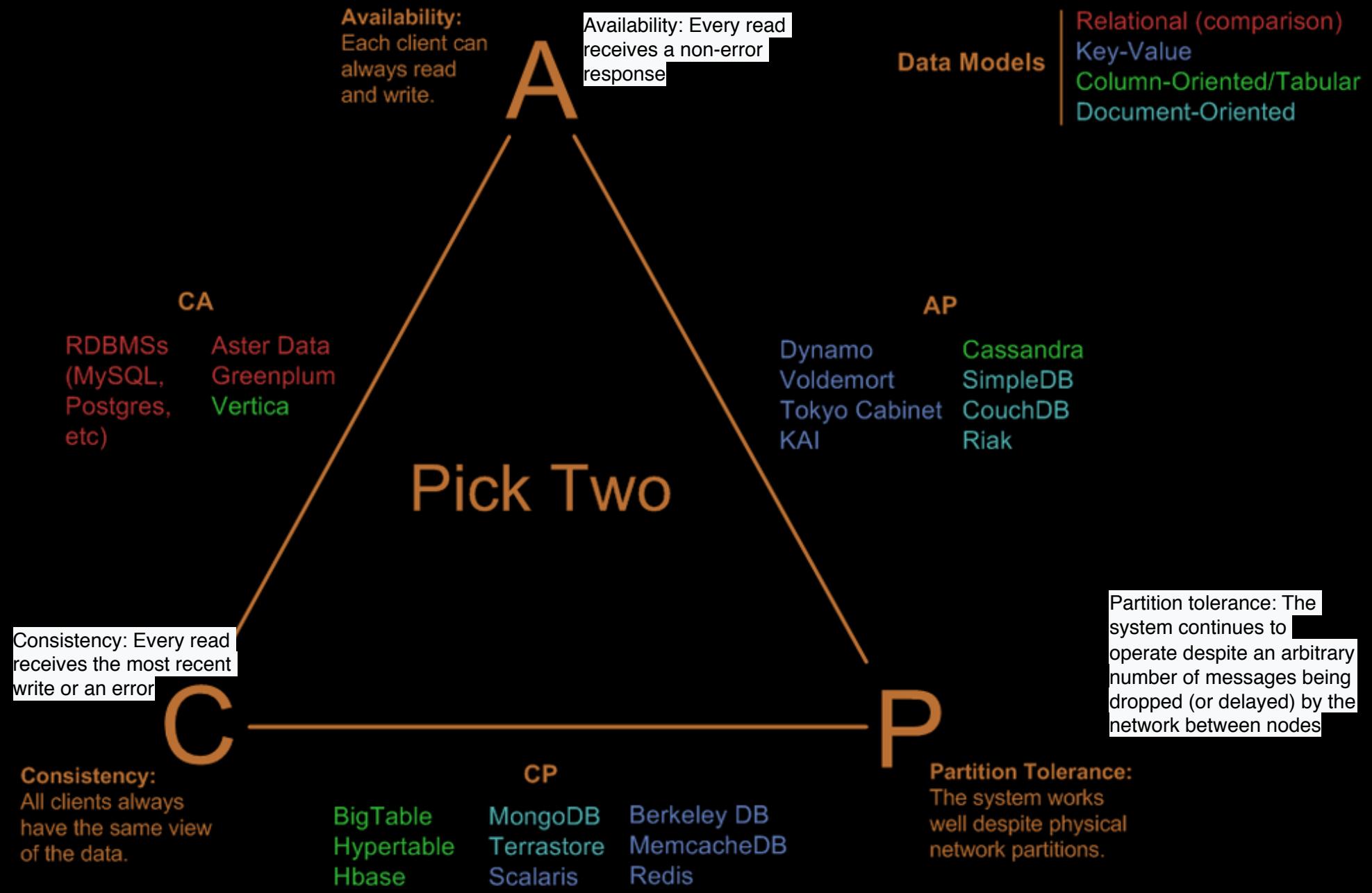
Series editor: *Bruce A. Tate*
Development editor: *Jacquelyn Carter*



different trade-offs are made

	top pro	top con
★ relational	flexible queries	scale, complexity
columnar	massive scale	hard to query, \$\$
★ key-value	fast	scale
★ document-oriented	schema-less	schema-less
graph	optimized for graph queries	graph queries is a narrow use

CAP theorem

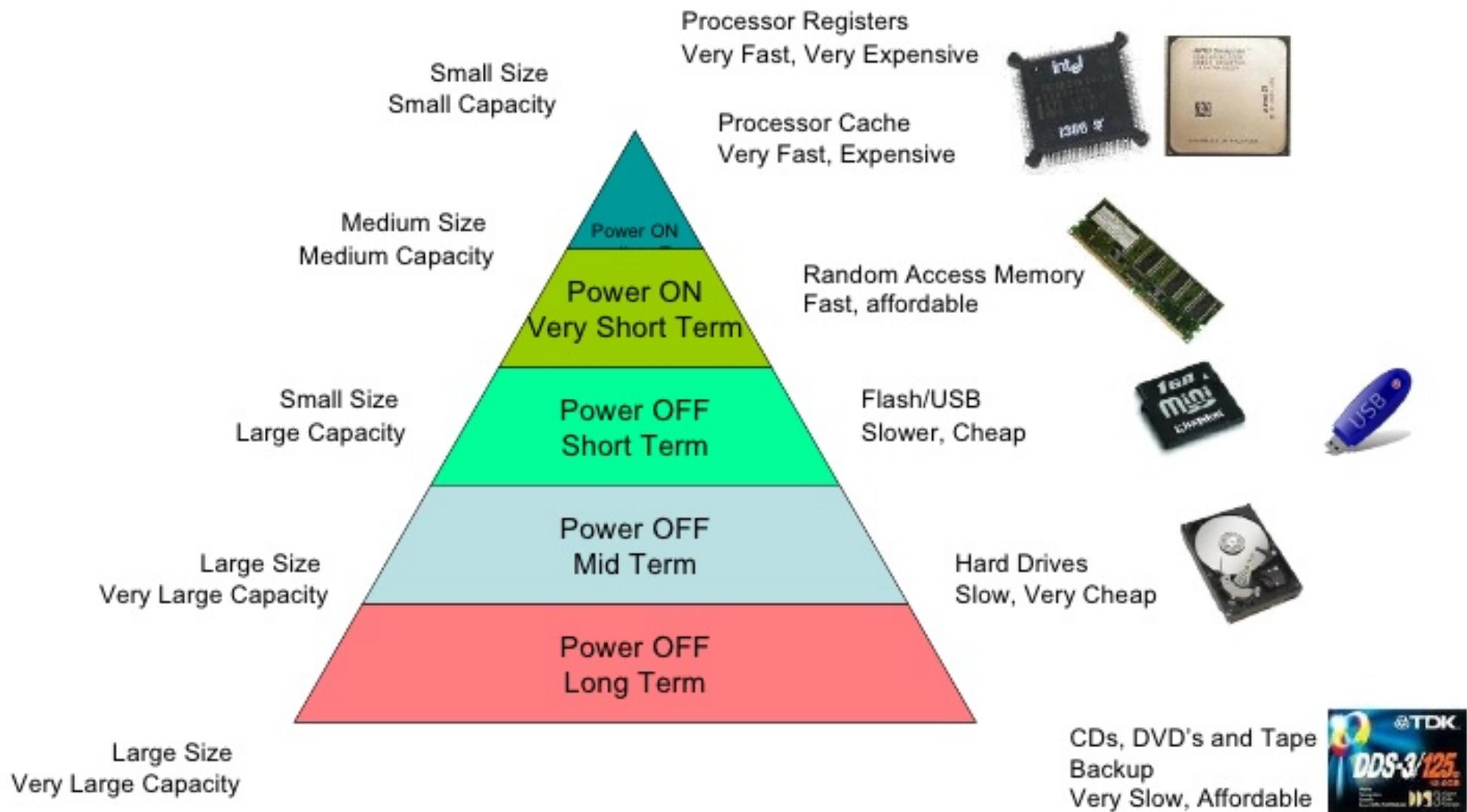


Redis: a key-value store

- In-memory store (fast access, but uses memory)
- simple, popular
- keys are strings
- values can be one of 5 types:
 - strings — a sequence of binary safe bytes up to 512 MB.
 - hashes — a collection of key value pairs.
 - lists — an in-insertion-order collection of strings.
 - sets — a collection of unique strings with no ordering.
 - sorted sets — a collection of unique strings ordered by user defined scoring.
- keys are organized into “databases”, which are just key namespaces

What is this “in-memory” thing I keep hearing about?

Computer Memory Hierarchy



top redis applications

1. caching

- repeated calls for same computation

2. queue

- requests come in fast, handled by slow process

3. counting

- Keep track of on the fly counts, used to detect bots/DOS attacks

4. pub/sub

- generic message management: one process sends messages with no knowledge of who receives them. Receivers subscribe to get messages.

Exercise 2: Redis CRUD

Lecture 4: Crucial Practicalities

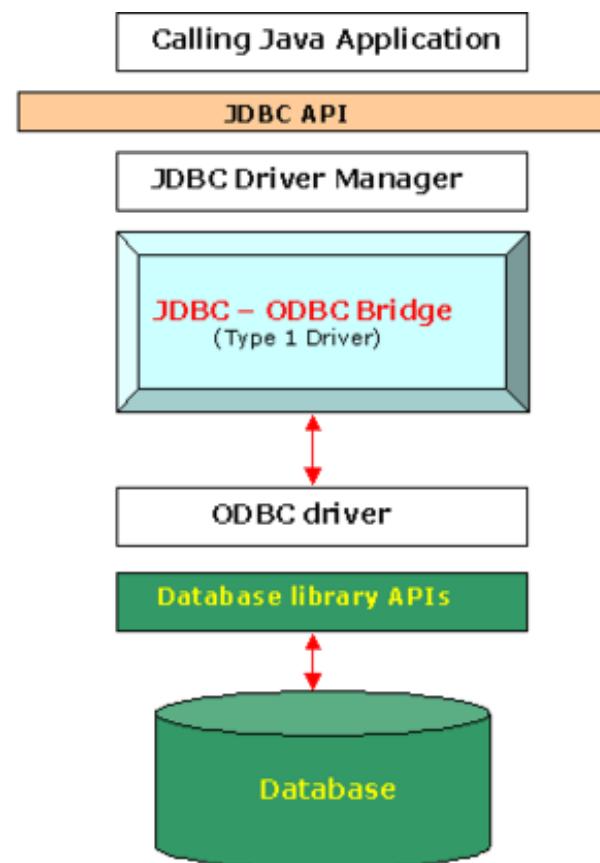
drivers, connections and interfaces

Connecting to the DB

- native client, cli, shell
- Apache thrift server
- http interface
- drivers (jdbc/odbc)
 - connection string: protocol, host, port
 - Database clients/IDE applications
 - connection libraries
 - connection objects

drivers

- API layer for talking to databases
- jdbc: java database connectivity
- odbc: open database connectivity
- example class:
org.postgresql.Driver



connection strings

<protocol>:// <host>:<port>/<schema>;<property>=<value>

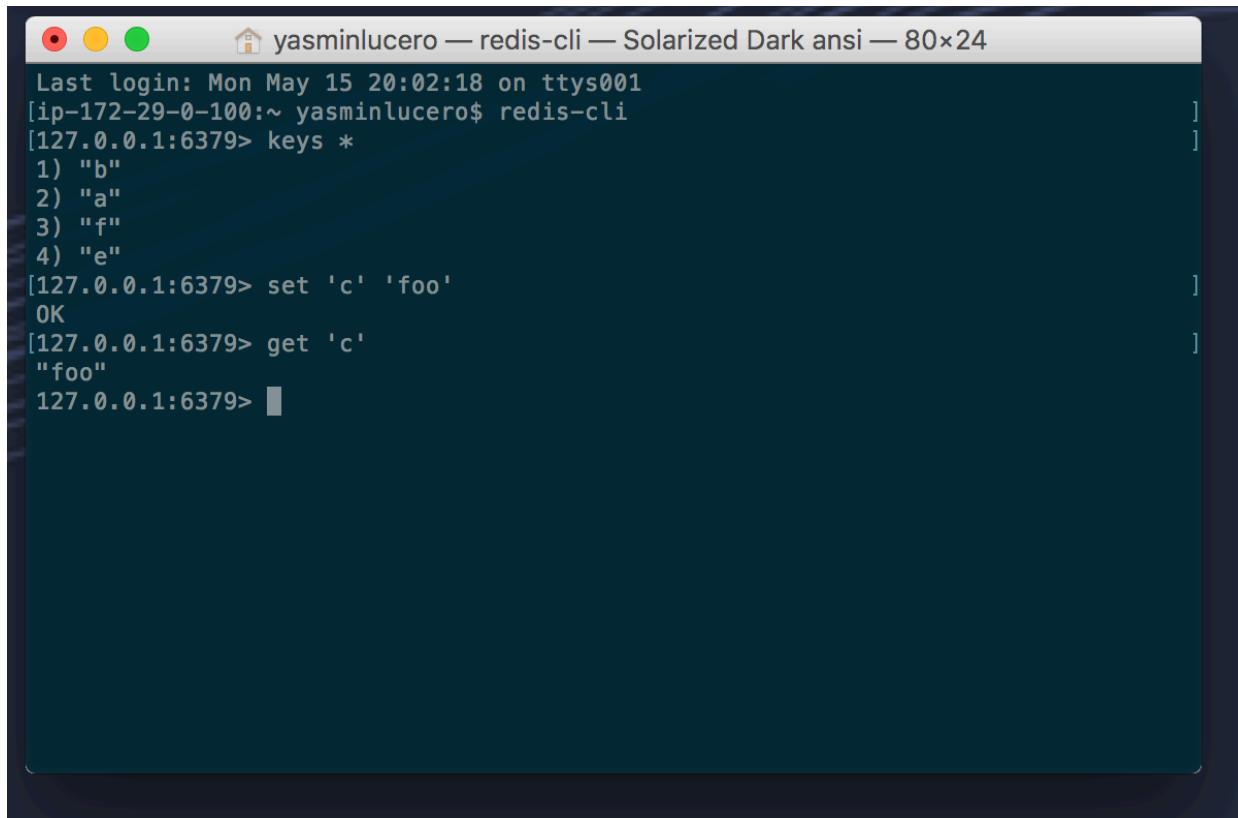
jdbc:postgresql://localhost:5432/myDB

jdbc:postgresql://www.foo.com:5432/myDB

jdbc:postgresql://www.foo.com:5432/myDB;username='meme';password='1234'

mongodb://<username>:<password>@<host>:<port>/<database><?options>

CLI: command line interface



A screenshot of a terminal window titled "yasminlucero — redis-cli — Solarized Dark ansi — 80x24". The window shows a Redis CLI session. The user has run "keys *" and received a list of keys: "b", "a", "f", and "e". Then, the user has run "set 'c' 'foo'" and received "OK". Finally, the user has run "get 'c'" and received the value "foo". The terminal has a dark background with light-colored text and a light gray header bar.

```
Last login: Mon May 15 20:02:18 on ttys001
[ip-172-29-0-100:~ yasminlucero$ redis-cli
[127.0.0.1:6379> keys *
1) "b"
2) "a"
3) "f"
4) "e"
[127.0.0.1:6379> set 'c' 'foo'
OK
[127.0.0.1:6379> get 'c'
"foo"
127.0.0.1:6379>
```

Thick Clients



sQuirreL



 **DataGrip**

Demo my data grip set up

client libraries (R-centric)

- DBI
- RJDBC
- RODBC
- RPostgreSQL
- rredis
- Rmongo

```
1 library(DBI)
2 library(RPostgreSQL)
3
4 drv <- dbDriver("PostgreSQL")
5 |
6 load("db_config.Rdata")
7 con <- dbConnect(drv,
8                     user = db_config$user,
9                     password = db_config$password,
10                    host = db_config$host,
11                    port = db_config$port,
12                    dbname = db_config$dbname)
13
14 data <- dbGetQuery(con, 'select * from boxes limit 5')
15
16 dbDisconnect(con)
```

connection objects in R

<https://cran.r-project.org/doc/manuals/r-release/R-data.html#Connections>

7 Connections

Connections are used in R in the sense of Chambers (1998) and Ripley (2001), a set of functions to replace the use of file names by a flexible interface to file-like objects.

- [Types of connections](#):
- [Output to connections](#):
- [Input from connections](#):
- [Listing and manipulating connections](#):
- [Binary connections](#):

connections {base} Functions to Manipulate Connections (Files, URLs, ...)

Next: [Output to connections](#), Previous: [Connections](#), Up: [Connections](#) [[Con](#) [Description](#)

7.1 Types of connections

Functions to create, open and close connections, i.e., “generalized files”, such as possibly compressed files, URLs, pipes, etc.

The most familiar type of connection will be a file, and file connections are c opened for reading or writing or appending, in text or binary mode. In fact, fi and writing.

Note that by default a connection is not opened when it is created. The rule is already open, and close a connection after use if it opened it. In brief, leave tl methods to explicitly open and close connections.

Files compressed via the algorithm used by `gzip` can be used as connections

Usage

```
file(description = "", open = "", blocking = TRUE,
      encoding = getOption("encoding"), raw = FALSE,
      method = getOption("url.method", "default"))

url(description, open = "", blocking = TRUE,
      encoding = getOption("encoding"),
      method = getOption("url.method", "default"))

gzfile(description, open = "", encoding = getOption("encoding"),
      compression = 6)

bzfile(description, open = "", encoding = getOption("encoding"),
      compression = 9)

xzfile(description, open = "", encoding = getOption("encoding"),
      compression = 6)

unz(description, filename, open = "", encoding = getOption("encoding"))

pipe(description, open = "", encoding = getOption("encoding"))

fifo(description, open = "", blocking = FALSE,
      encoding = getOption("encoding"))

socketConnection(host = "localhost", port, server = FALSE,
                 blocking = FALSE, open = "a+",
                 encoding = getOption("encoding"),
                 timeout = getOption("timeout"))

open(con, ...)
```

Exercise 3:

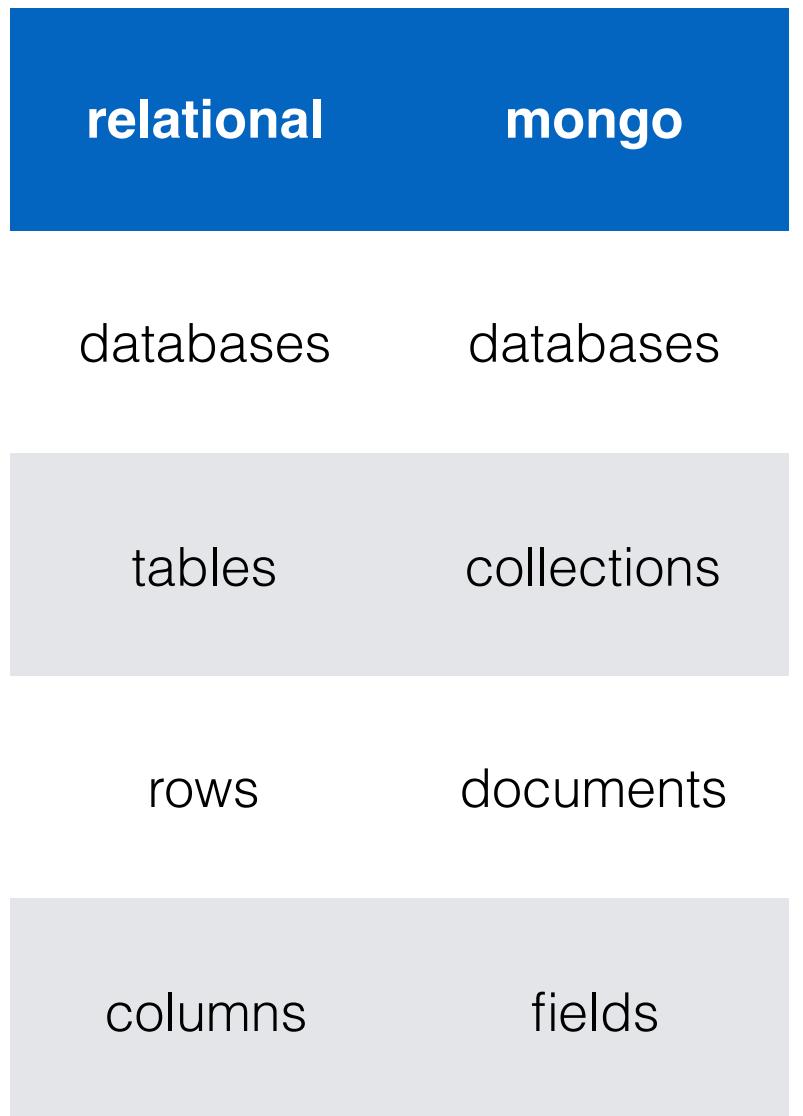
Mongo CRUD

a document-oriented db

Mongo DB

- mongo is for ‘humongous’
- mongo has its own query language based on JSON
- Liked by web devs because it speaks ‘web’, i.e. REST API
- hierarchy: databases > collections > documents > fields
- collections can be indexed
- cursors

“flexible schema”



- relational columns are defined at table level while mongo fields are defined at the document level

normalized

user document

```
{  
  _id: <ObjectId1>,  
  username: "123xyz"  
}
```

contact document

```
{  
  _id: <ObjectId2>,  
  user_id: <ObjectId1>,  
  phone: "123-456-7890",  
  email: "xyz@example.com"  
}
```

access document

```
{  
  _id: <ObjectId3>,  
  user_id: <ObjectId1>,  
  level: 5,  
  group: "dev"  
}
```

embedded

```
{  
  _id: <ObjectId1>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```

Embedded sub-document

Embedded sub-document

Hour 1

6:00 - 6:10 introduction/preliminaries

6:10 - 6:20 Lecture 1: data science architecture

6:20 - 6:40 Lecture 2: relational databases

6:40 - 7:00 Exercise 1: SQL CRUD

Hour 2

7:00 - 7:10 Break

7:10 - 7:25 Lecture 3: database trade-offs, noSQL, CAP theorem

7:25 - 7:40 Exercise 2: Redis CRUD

7:40 - 7:50 Lecture 4: crucial practicalities: JDBC Drivers, connection strings, cli, IDEs and libraries

7:50 - 8:05 Exercise 3: Mongo CRUD

Hour 3

8:05 - 8:15 Break

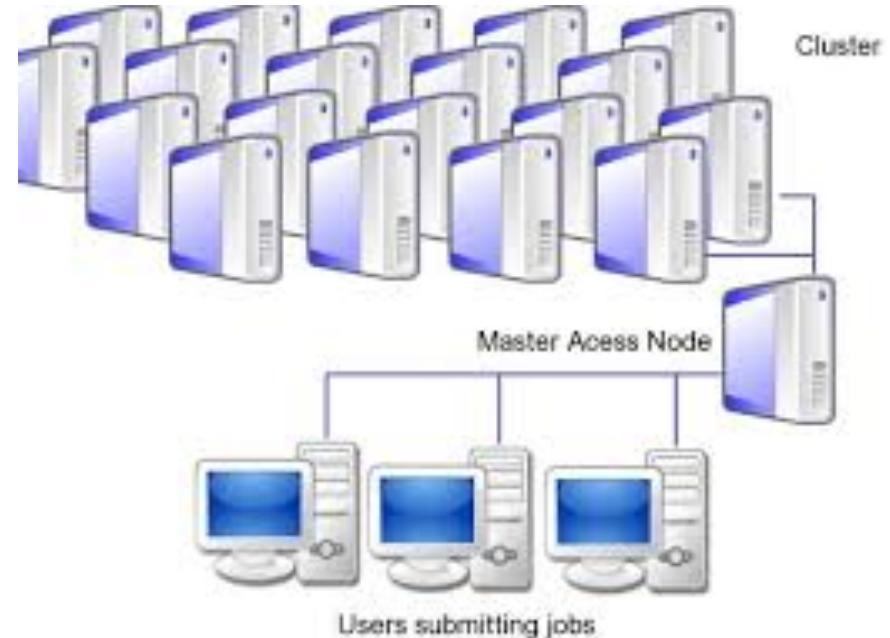
8:15 - 8:45 Lecture 5: Distributed computing, Hadoop, Spark, Hive, Pig

8:45 - 8:50 Wrap up

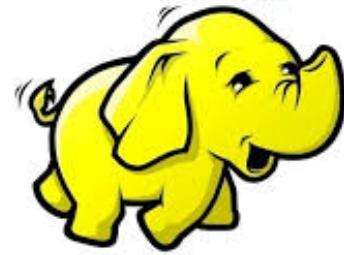
Lecture 5: distributed data computation Hadoop, sharding, Spark

distributed means many machines, it means cluster

- distributed compute must be parallel, but not all parallel computation is distributed
- network IO
- replication
- manager nodes
- you only do it if you have to
- big data requires distribution



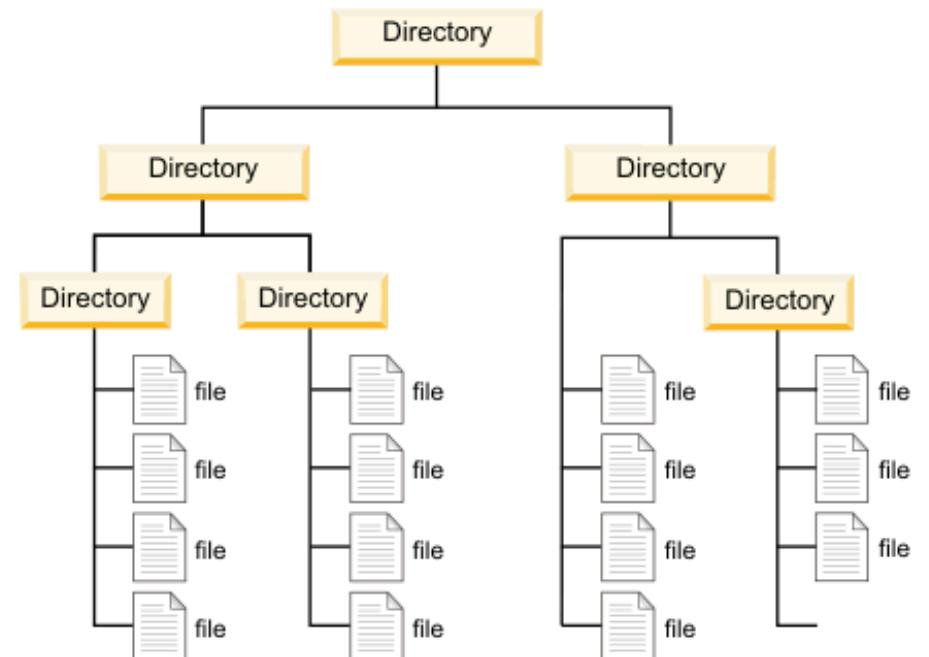
hadoop



Hadoop is actually two distinct things:
HDFS and MapReduce

a file system

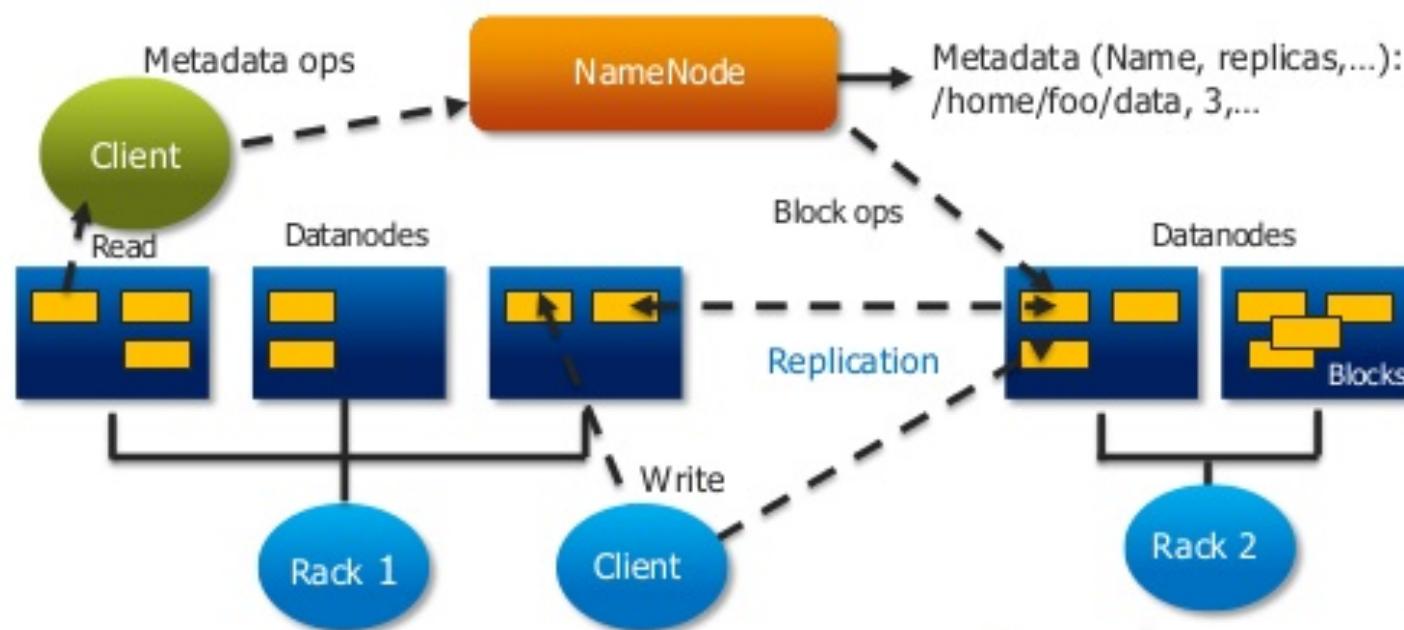
- ls
- cp
- mv
- touch
- rm



HDFS: Hadoop distributed file system

HDFS Architecture

edureka!



problems solved by HDFS

- massive scale
- cheap hardware (commodity hardware)
- fault tolerant (see above cheap hardware)
- distribution abstracted away

interface to HDFS

- hadoop cli
- web interface
- programmatically (java client)
- webhdfs (REST API)
- file system mount (fuse, NFS)

Contents of directory [/user/vijay/temp](#)

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
large-file.txt	file	95.37 MB	1	64 MB	2013-01-29 15:55	rw-r--r--	vijay	hadoop-users
shakespeare.txt	file	109.64 KB	1	64 MB	2013-01-29 15:35	rw-r--r--	vijay	hadoop-users

[Go back to DFS home](#)

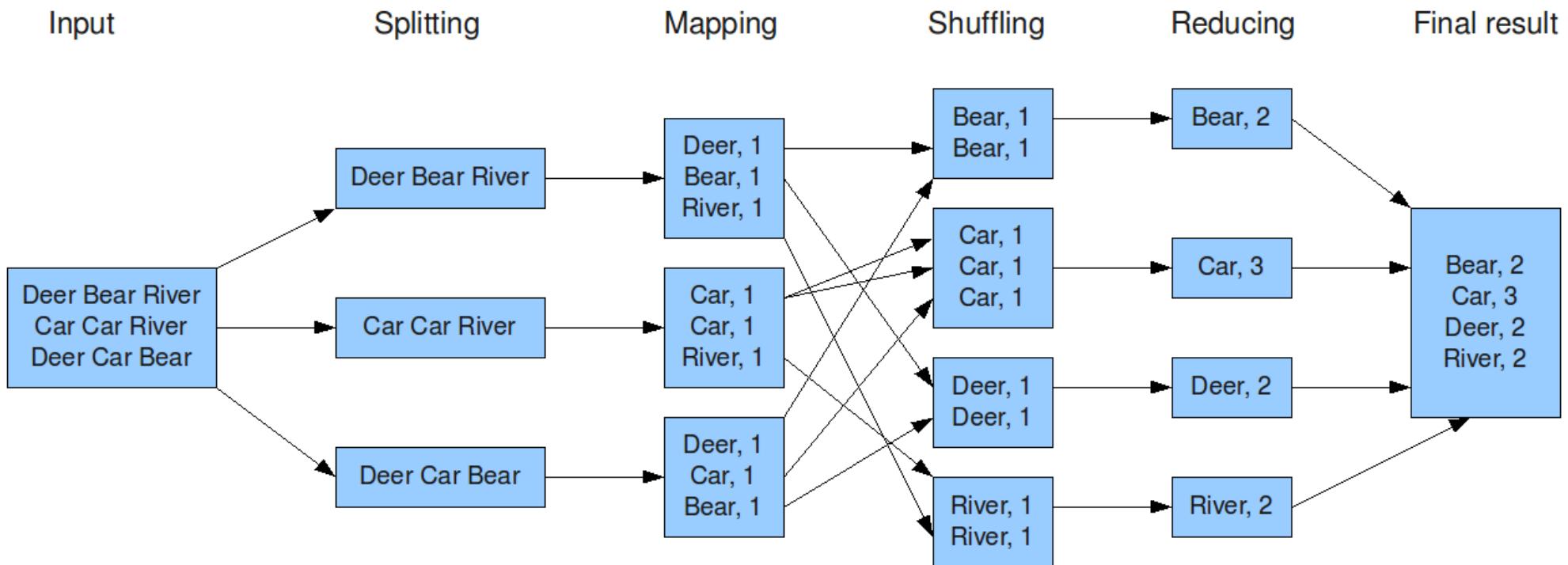
Local logs

[Log directory](#)

[Hadoop](#), 2013.

MapReduce: split-apply-combine idiom

The overall MapReduce word count process



Hadoop Eco System



Apache Ambari
<http://incubator.apache.org/ambari>

System Deployment

Machine Learning



Mahout
Machine Learning

Distributed Programming



Pig
Scripting



Hive
SQL Query



Spark

Scheduling



Oozie
Workflow

Service Programming

Data Ingestion



Sqoop



Storm

MapReduce Framework-YARN



NoSQL Database



Distributed Filesystem



Hadoop core

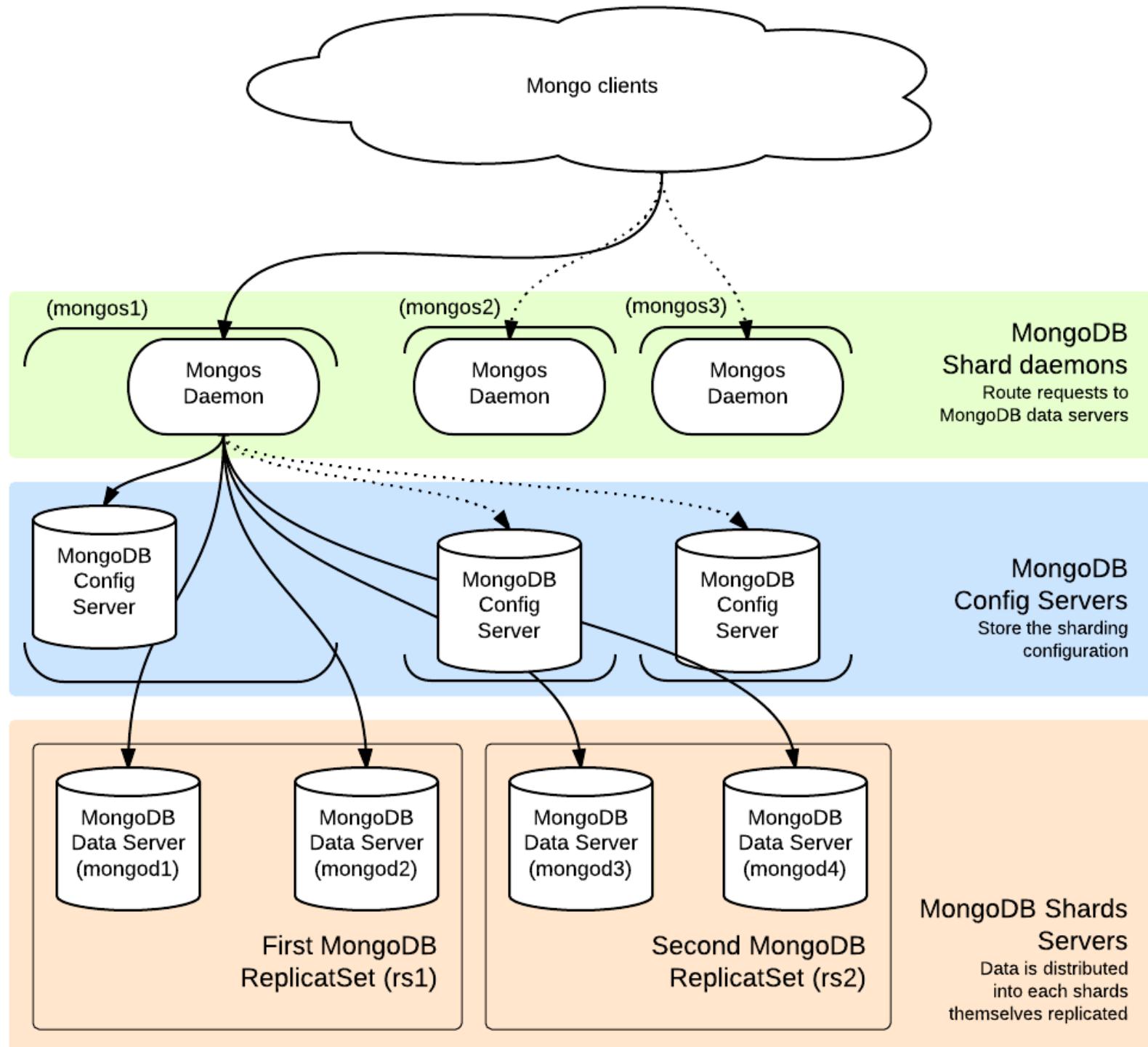
distributed data bases

A **database shard** is a [horizontal partition](#) of data in a [database](#) or [search engine](#). Each individual partition is referred to as a **shard** or **database shard**. Each shard is held on a separate [database server](#) instance, to spread load.

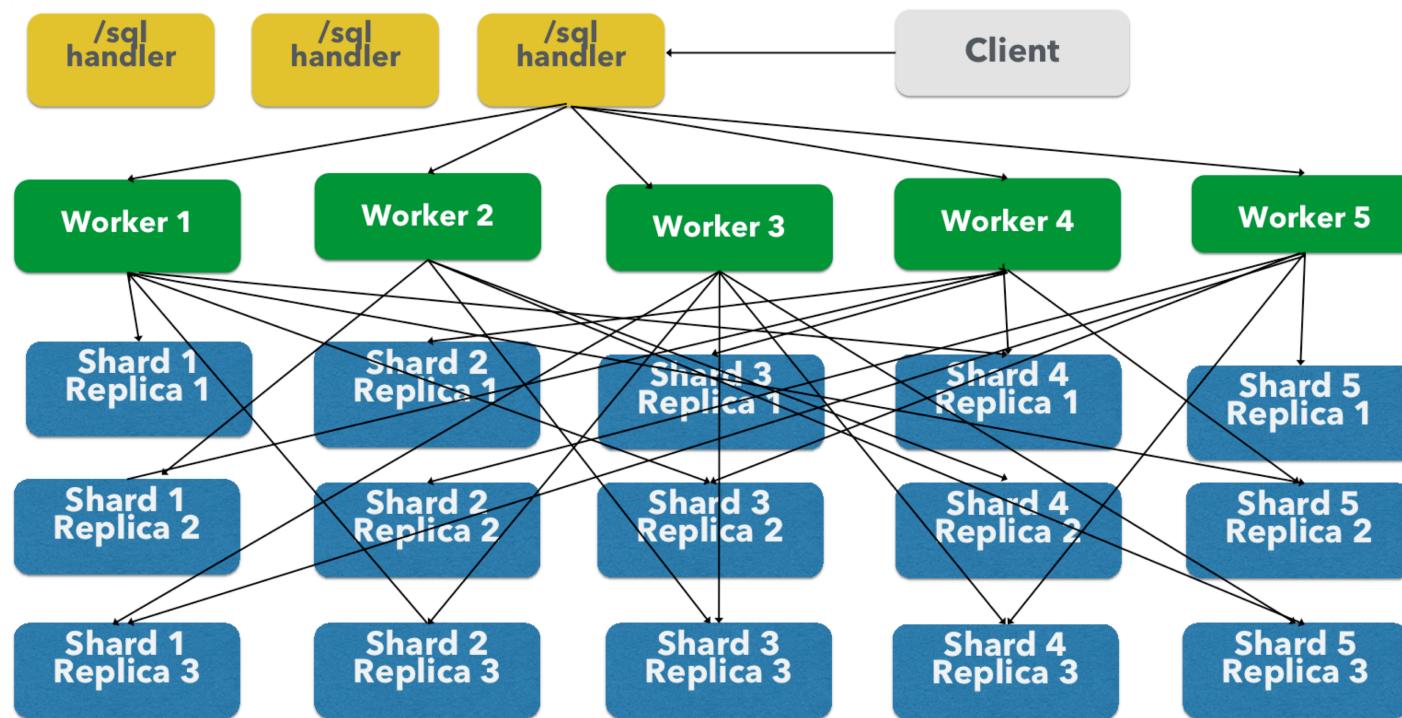
Some data within a database remains present in all shards,^{[\[notes 1\]](#)} but some appears only in a single shard. Each shard (or server) acts as the *single* source for this subset of data.^{[\[1\]](#)}

– Wikipedia

[https://en.wikipedia.org/wiki/Shard_\(database_architecture\)](https://en.wikipedia.org/wiki/Shard_(database_architecture))



Distributed SQL DB



distributed Postgres

Program	License	Maturity	Replication Method	Sync	Connection Pooling	Load Balancing	Query Partitioning
PgCluster	BSD	Not production ready	Master-Master	Synchronous	No	Yes	No
pgpool-I	BSD	Stable	Statement-Based Middleware	Synchronous	Yes	Yes	No
Pgpool-II	BSD	Recent release	Statement-Based Middleware	Synchronous	Yes	Yes	Yes
slony	BSD	Stable	Master-Slave	Asynchronous	No	No	No
Bucardo	BSD	Stable	Master-Master, Master-Slave	Asynchronous	No	No	No
Londiste	BSD	Stable	Master-Slave	Asynchronous	No	No	No
Mammoth	BSD	No longer maintained	Master-Slave	Asynchronous	No	No	No
rubyrep	MIT	No longer maintained	Master-Master, Master-Slave	Asynchronous	No	No	No
Bi-Directional Replication 	PostgreSQL (BSD)	Recent release	Master-Master (no triggers needed)	Asynchronous	No	No	No
pg_shard 	LGPL	Recent release	Statement-based Middleware (as an extension)	Synchronous	No	Yes	Yes
pglogical 	PostgreSQL	Recent release	Master-Slave	Asynchronous	No	No	No
Postgres-XL 	PostgreSQL	Recent release	MPP Postgres, scalable writes & reads	Synchronous	Yes	Yes	Yes
Citus 	AGPL	Recent release	MPP Postgres, scalable writes & reads	Asynchronous or Synchronous	Yes	Yes	Yes

distributed compute



Hive: SQL on HDFS

you write sql:

hive submits a
MapReduce job:

HUE

Query Editors Data Browsers Workflows Search

Hive Editor Query Editor My Queries Saved Queries History

Navigator Settings

SETTINGS

Add

FILE RESOURCES

Add

UDFS

Add

OPTIONS

Enable parameterization

Email me on completion

1 select host, ip_add_int, year, month, day, request
2 from apache_log_date_ip_split_parquet
3 limit 5;

Execute Save as... Explain or create a New query

Recent queries Query Log Columns Results Chart

host ip_add_int year month day request

0 202.46.49.127 3392024913 2014 Mar 02 "GET / HTTP/1.1"
1 119.63.193.194 2000687011 2014 Mar 02 "GET / HTTP/1.1"
2 66.249.66.96 1123631463 2014 Mar 02 "GET /author/adam-seed/ HTTP/1.1"
3 66.249.66.96 1123631463 2014 Mar 02 "GET /2010/03/an-update-on-the-bl-forum-oblere-emr-and-us-conferencesfeed/ HTTP/1.1"
4 150.255.160.251 2533335036 2014 Mar 02 "GET /cgi-bin/htpd.cgi?/bin/busybox HTTP/1.0"

10 Hadoop MapReduce Administration

Job: 10_MR01
Started: Sun May 10 17:17:20 CEST 2015
Version: 0.21.0-965206
Completed: Sun May 10 17:20:25 CEST 2015 by hadoop01 from slave01/slave01-0.21
Master: 20150510-1711

Cluster Summary (Heap Size is 106.12 MB/1.56 GB)

Queues	Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Allocated Map Tasks	Allocated Reduce Tasks	Reserved Map Tasks	Reserved Reduce Tasks	Map Slot Capacity	Reduce Slot Capacity	Avg. Slots/Node	Allocated Mappers	Allocated Reducers
1	0	1	1	1	0	1	0	0	0	0	1.00	0	0

Filter (Global, Priority, User, Name)
Example: You can filter by priority, user or name, but not both at the same time.

Running Jobs

JobID	Priority	User	Name	Map % Complete	Map Total	Reduce % Complete	Reduce Total	Reduce % Complete	Reduce Total	Reducers Completed	All % Scheduling Information
Job_20150510-1711-0001	HIGH	hadoop	CustomJob	100%	10	100%	10	100%	10	0	0%

Retired Jobs

Local Logs

Log directory: jobTracker history
Hadoop: 28 H.



Impala: Also SQL on HDFS

You write SQL, get your answers back super fast

The screenshot shows the MySQL Workbench interface. The top navigation bar includes 'File', 'Edit', 'Open Editors', 'Data Browser', 'Workflows', 'Search', 'File Browser', 'Edit Browser', 'OC host', and 'Help'. Below the navigation is a toolbar with 'Import', 'Query Editor' (selected), 'My Queries', 'Shared Queries', and 'History'. The main area is divided into two panes. The left pane, titled 'Navigator' and 'Settings', shows a 'DATABASE' dropdown set to 'default' and a 'Tables name...' input field. Below are lists of tables: 'sys_apache_log', 'sys_apache_log_ip', 'sys_apache_log_ip_port', 'sys_apache_log_ip_port_1', 'sys_apache_log_ip_port_2', 'sys_apache_log_ip_port_3', 'sys_apache_log_ip_port_4', and 'sys_apache_log_ip_port_5'. The right pane contains a query editor with the following complex query:

```
SELECT * FROM sys_apache_log_ip_port_1 WHERE ip_port_1_ip = '127.0.0.1' AND ip_port_1_port = 80 AND ip_port_1_protocol = 'HTTP/1.1' AND ip_port_1_time = '2014-04-01 10:00:00' AND ip_port_1_user_agent = 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1651.102 Safari/537.36' AND ip_port_1_referer = 'http://www.google.com/';
```

Below the query editor are tabs: 'Execute' (selected), 'Save', 'Save as...', 'Explain', 'Print/execute', and 'New query'. The results pane at the bottom shows a table with columns: 'Recent queries', 'Query', 'List', 'Columns', 'Results' (selected), and 'Chart'. The results table contains the following data:

Recent queries	Query	List	Columns	Results	Chart
1	1 AND 1=ip_port_1_ip year month day request				1 country_name
2	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	Croatia
3	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
4	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
5	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
6	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
7	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
8	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
9	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
10	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
11	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
12	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
13	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
14	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
15	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
16	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
17	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
18	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
19	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
20	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
21	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
22	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
23	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
24	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
25	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
26	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
27	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
28	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
29	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
30	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
31	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
32	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
33	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
34	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
35	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
36	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
37	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
38	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
39	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
40	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
41	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
42	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
43	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
44	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
45	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
46	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
47	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
48	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
49	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
50	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
51	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
52	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
53	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
54	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
55	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
56	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
57	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
58	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
59	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
60	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
61	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
62	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
63	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
64	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
65	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
66	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
67	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
68	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
69	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
70	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
71	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
72	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
73	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
74	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
75	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
76	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
77	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
78	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
79	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
80	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
81	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
82	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
83	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
84	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
85	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
86	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
87	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
88	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
89	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
90	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
91	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
92	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
93	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
94	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
95	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
96	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
97	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
98	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
99	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
100	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
101	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
102	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
103	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
104	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
105	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
106	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
107	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
108	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
109	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
110	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
111	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
112	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
113	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
114	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
115	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
116	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
117	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
118	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
119	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
120	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
121	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
122	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
123	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
124	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
125	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
126	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
127	00:00:00.000-00:00:00.000	2014	Apr	00:00:00.000-00:00:00.000	United States
128	00:00:00.000-00				

The catch:

1. Impala not free
 2. Impala creates an in-memory data index (memory is not free either)



Apache Pig

You write a pig script:

- Sample Pig script:

```
stock = LOAD '/user/fred/stock' AS (id, item);
orders= LOAD '/user/fred/orders' AS (id, cost);
grpds = GROUP orders BY id;
totals = FOREACH grpds GENERATE group,
SUM(orders.cost) AS t;
result = JOIN stock BY id, totals BY group;
DUMP result;
```

Pig submits a MapReduce job:

10 Hadoop MapReduce Administration

State: 0.00%
Started: Sun May 10 17:17:39 PDT 2015
Version: 0.21.0-96526
Completed: Sun May 10 17:02:25 PDT 2015 by hadoop01 from slave01@slave01
Master: 203.65.142.111

Cluster Summary (Heap Size is 106.12 MB/1.56 GB)

Queues	Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Allocated Map Tasks	Allocated Reduce Tasks	Reserved Map Tasks	Reserved Reduce Tasks	Map Slot Capacity	Reduce Slot Capacity	Avg. Slots/Node	Allocated Nodes	Allocated Slots
1	0	1	1	0	0	0	0	0	0	0	0.00	0	0

Filter (Global, Priority, User, Name)
Example: You can filter by priority, user, name or node. Just enter a part of it.

Running Jobs

JobId	Priority	User	Name	Map % Complete	Map Total	Reduce % Complete	Reduce Total	Reduce Total	Reducers Completed	All % Scheduling Information
job_201505101702_0001	HIGH	hadoop	CustomerCounts	100%	10	100%	10	10	1	100%

Retired Jobs

None

Local Logs

Log directory: jobTracker history
Hadoop: 24 H.



- Spark is a cluster compute framework
- provides an API build around the RDD
- spark can interact with Hadoop clusters, but can also operate with other kinds of clusters

RDD: resilient distributed dataset

- distributed: more than one machine
- resilient: fault tolerant

RDD: resilient distributed dataset

- immutable collection of elements that can be operated on in parallel
- implemented in Scala with the Akka actor model
- transformations executed lazily
- caching of intermediate steps

https://cs.stanford.edu/~matei/papers/2012/nsdi_spark.pdf

Spark SQL and DataFrames

- spark sql is the implementation of the relational model for RDDs
- DataFrames are a super familiar data structure
- built on top of RDDs
- interacting with a spark DataFrame looks a lot like a `dplyr` workflow
- Also a SQL interface to Spark DataFrames

example script in Scala using spark DataFrame API

```
// Creates a DataFrame based on a table named "people"  
// stored in a MySQL database.  
val url =  
  "jdbc:mysql://yourIP:yourPort/test?user=yourUsername;password=yourPassword"  
val df = sqlContext  
  .read  
  .format("jdbc")  
  .option("url", url)  
  .option("dbtable", "people")  
  .load()  
  
// Looks the schema of this DataFrame.  
df.printSchema()  
  
// Counts people by age  
val countsByAge = df.groupBy("age").count()  
countsByAge.show()  
  
// Saves countsByAge to S3 in the JSON format.  
countsByAge.write.format("json").save("s3a://...")
```

spark interfaces

- scala API
- python API (pySpark)
- R API: sparkR and sparklyR (rstudio)
- REPL (interactive command line interface)
- Notebooks: jupyter, DataBricks, other

spark is at the peak of its hype cycle

Gartner Hype Cycle

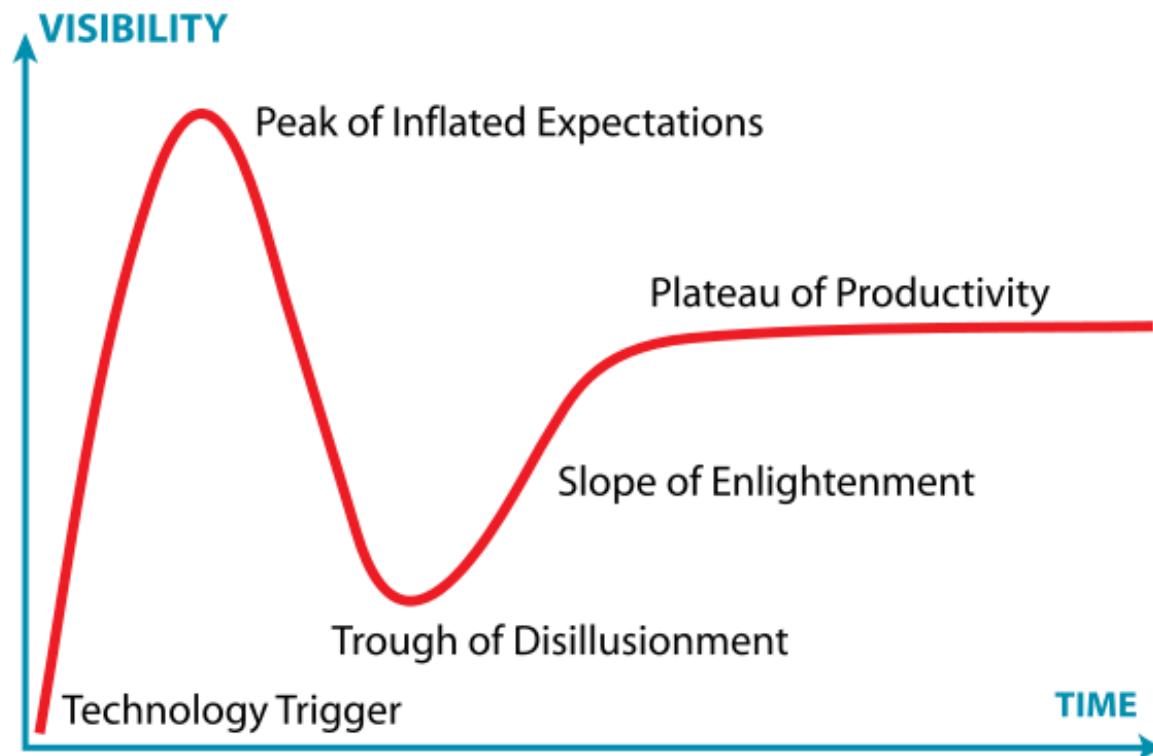
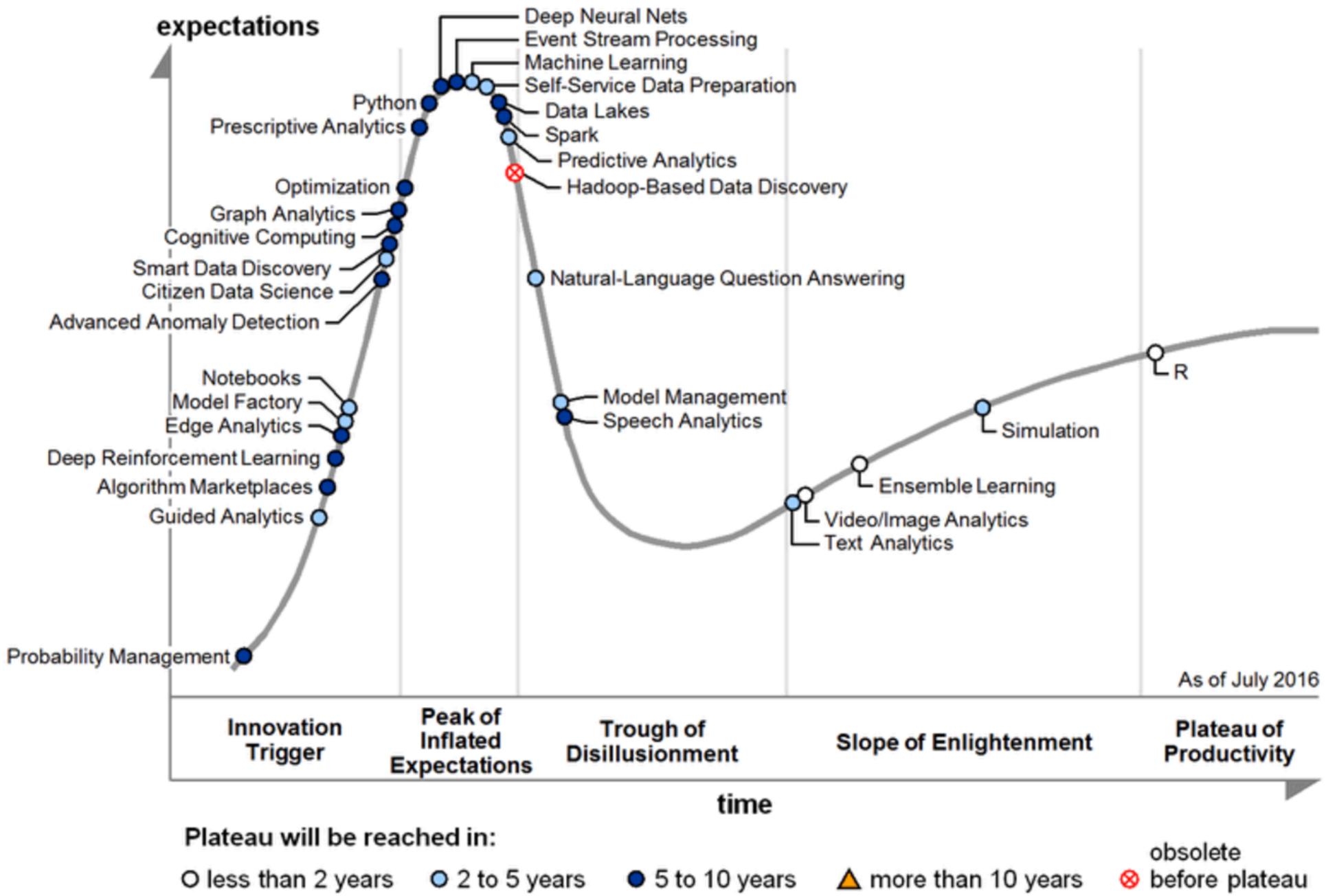


Figure 1. Hype Cycle for Data Science, 2016



Source: Gartner (July 2016)

“The difference between theory and practice is much bigger in theory than in practice.”

–Walter Savitch

to sum up

- Distribution makes easy things hard
- Distribution also makes impossibly large things possible
- Frameworks and tools now exist to manage the complexity
- Not all the problems are solved yet, the tech is continuing to develop rapidly

wrap up

- databases, and their issues, are your problem
- there is no one data solution that meets all needs
- trade-offs are the name of the game
- people say a lot of things: count on nothing that you have not verified for yourself
- distribution is both great and awful
- DataFrame has emerged as an important abstraction