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

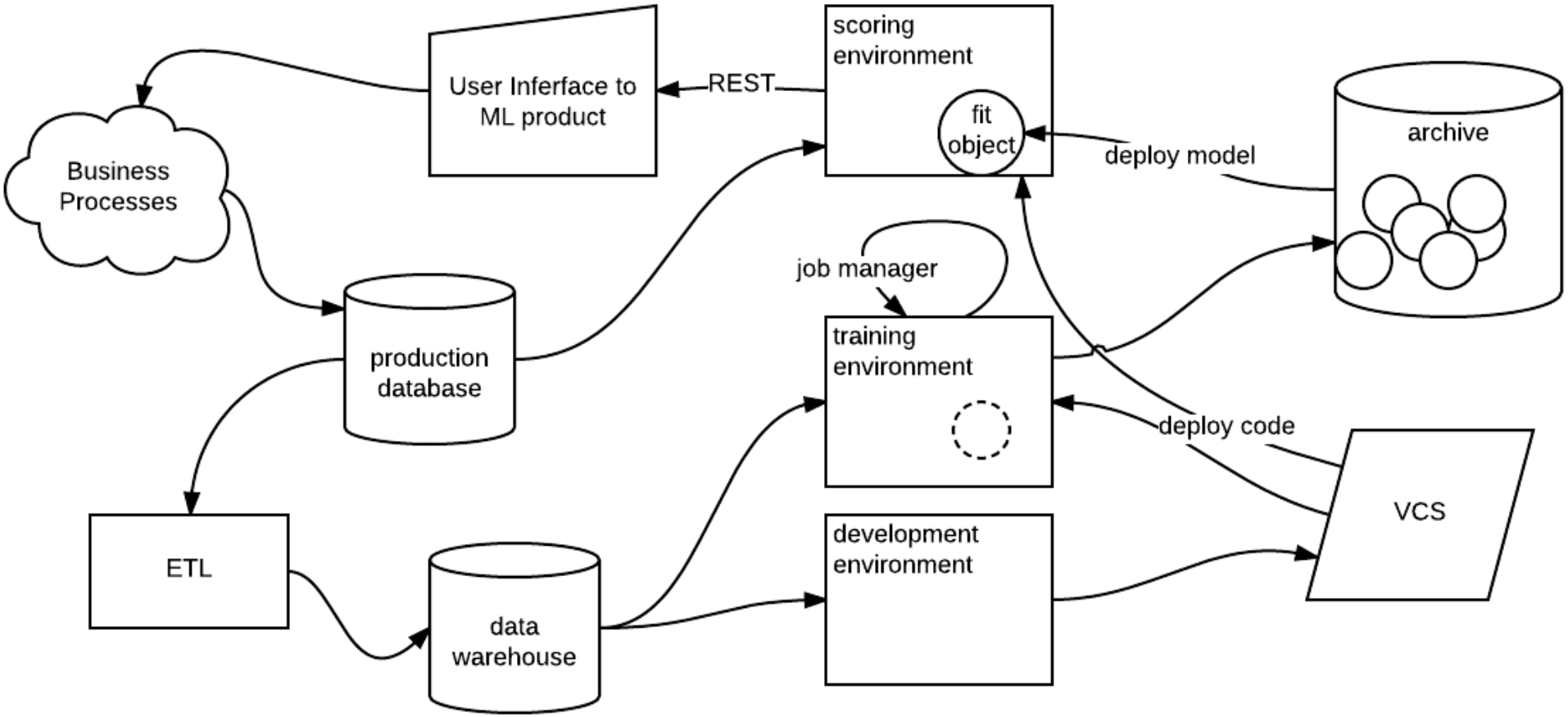
8:15 - 8:45 Lecture 5: Distributed computing, Hadoop, Spark, Hive, Pig

8:45 - 8:50 Wrap up

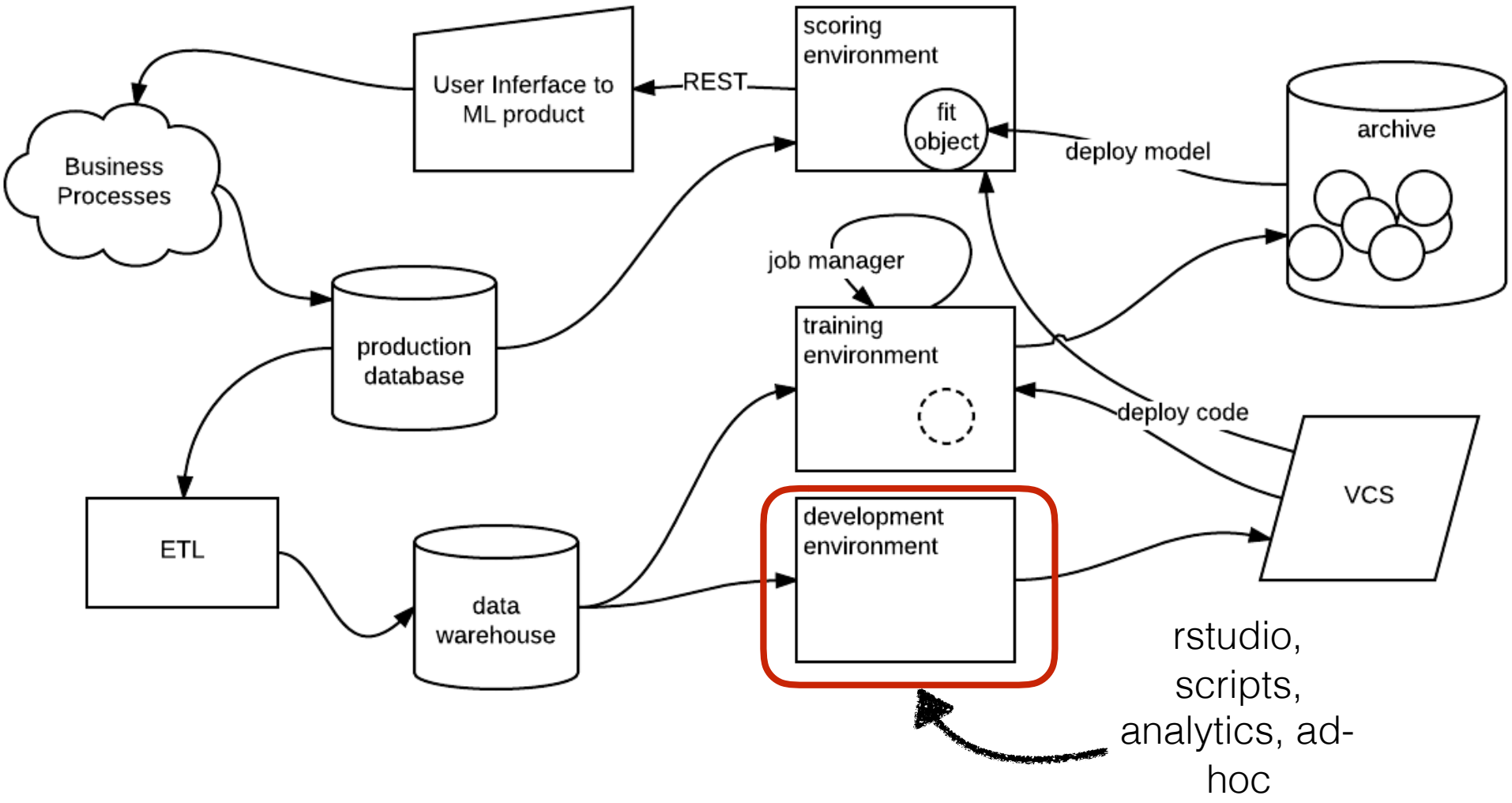
Lecture 1:
data science platform architecture

- why should a data scientist care about data stores?

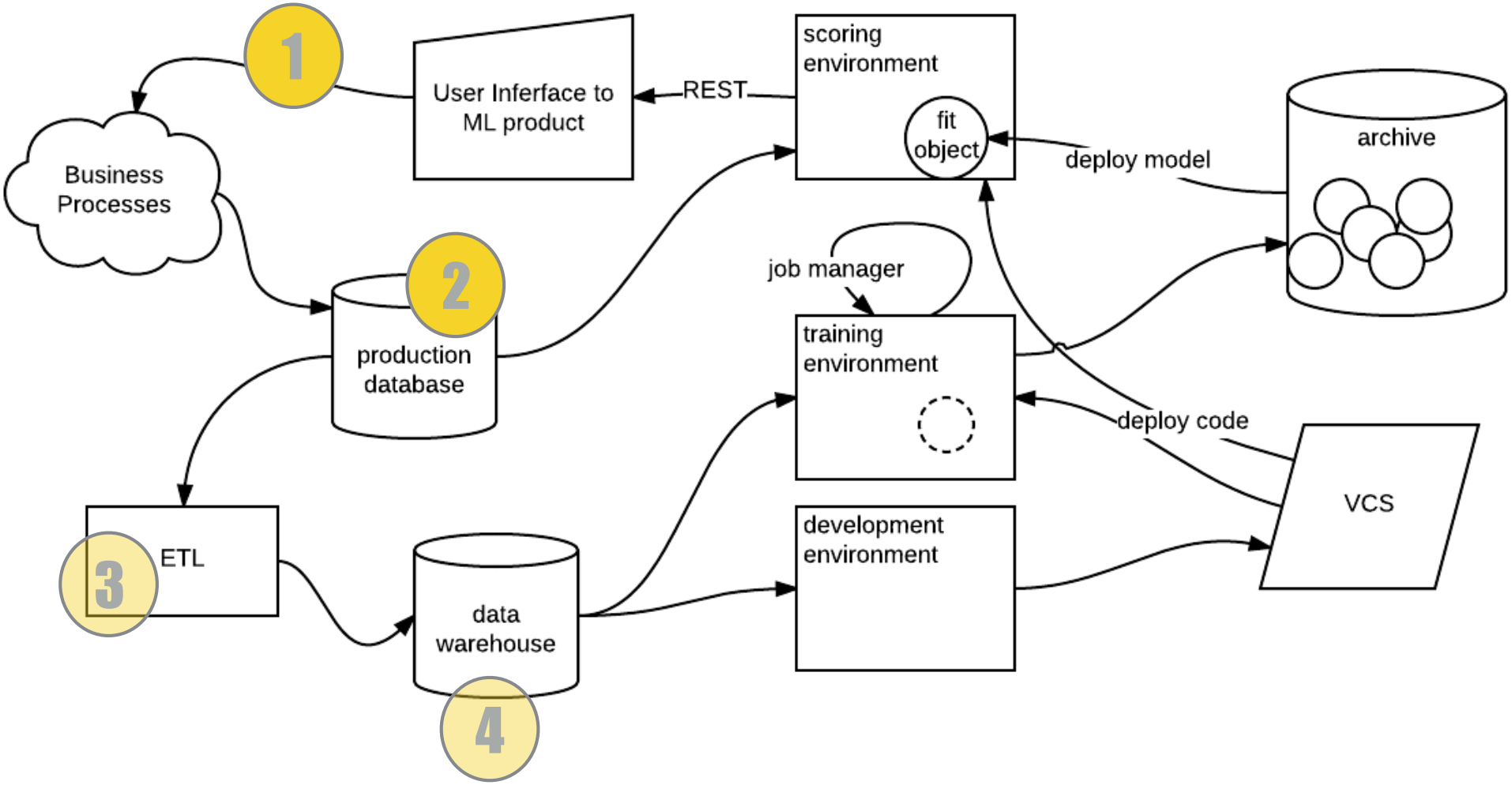
Data science within the production ecosystem



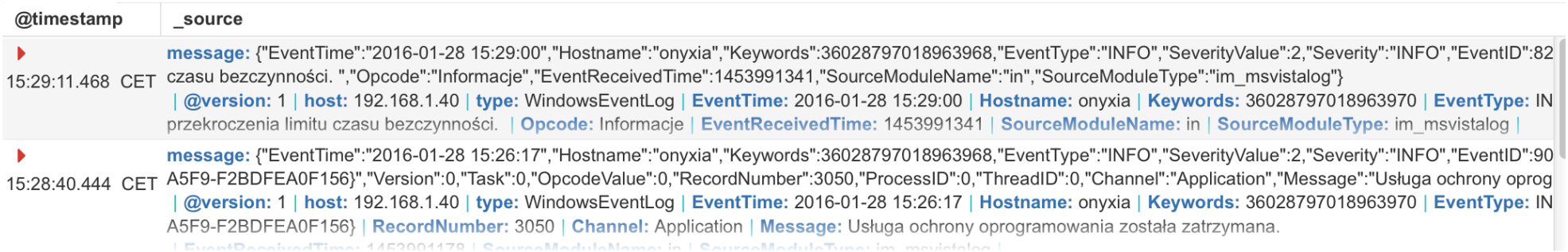
To date, most of your data science life has been within the 'development environment'



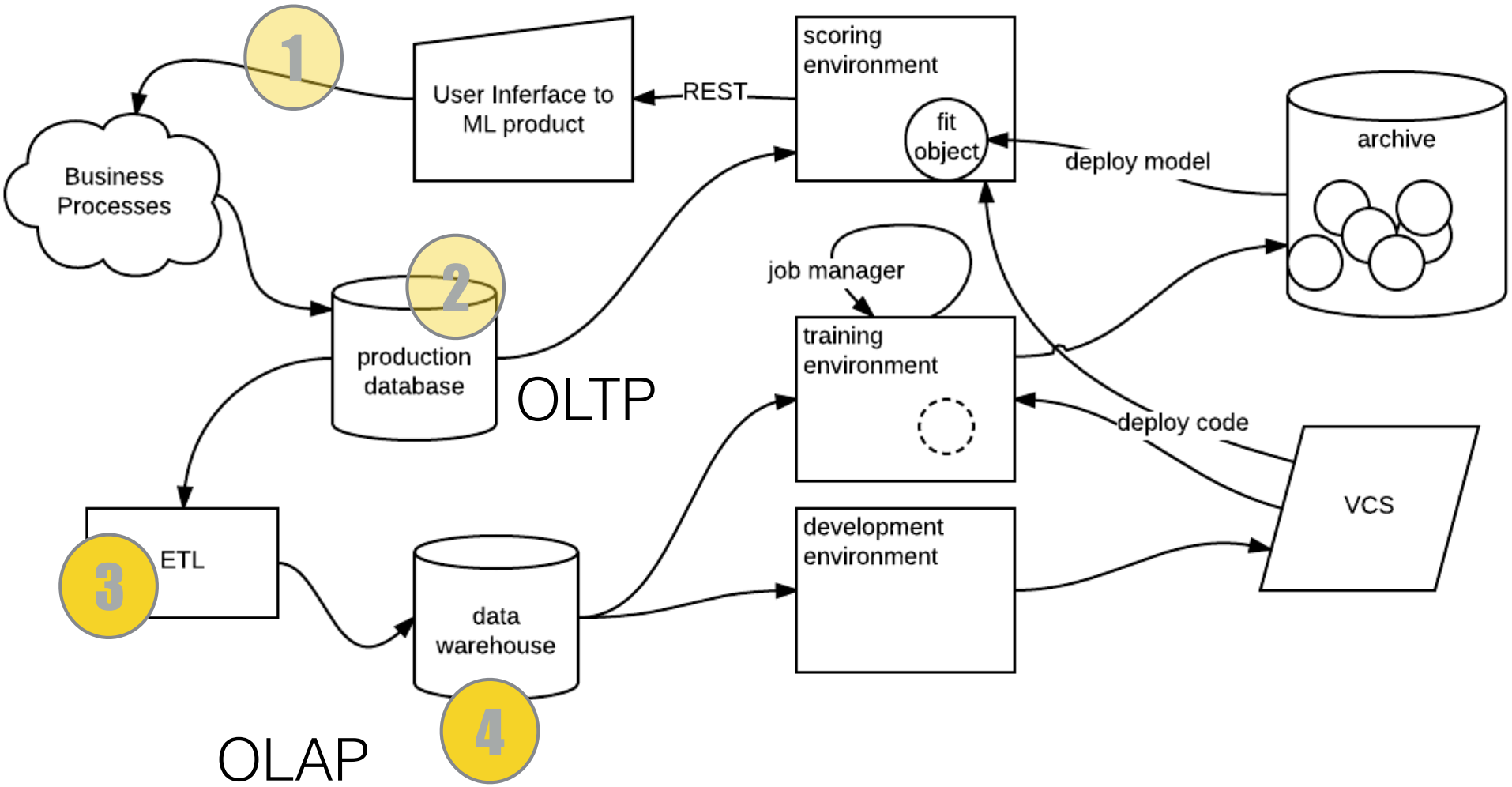
‘event data’ captured in write-optimized data store



event log data



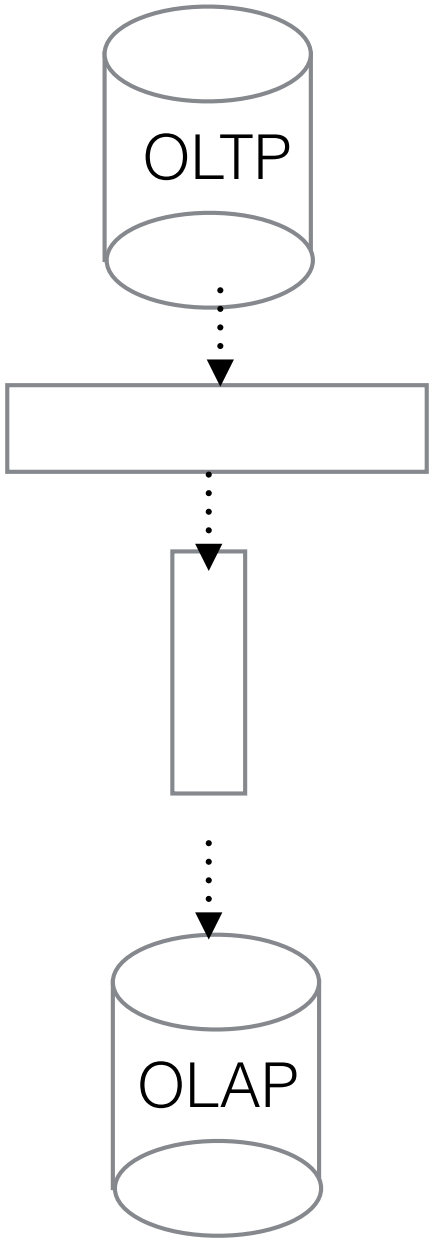
- timestamp
- event type
- user id
- page url
- page data
- user agent string
- ip
- Etc

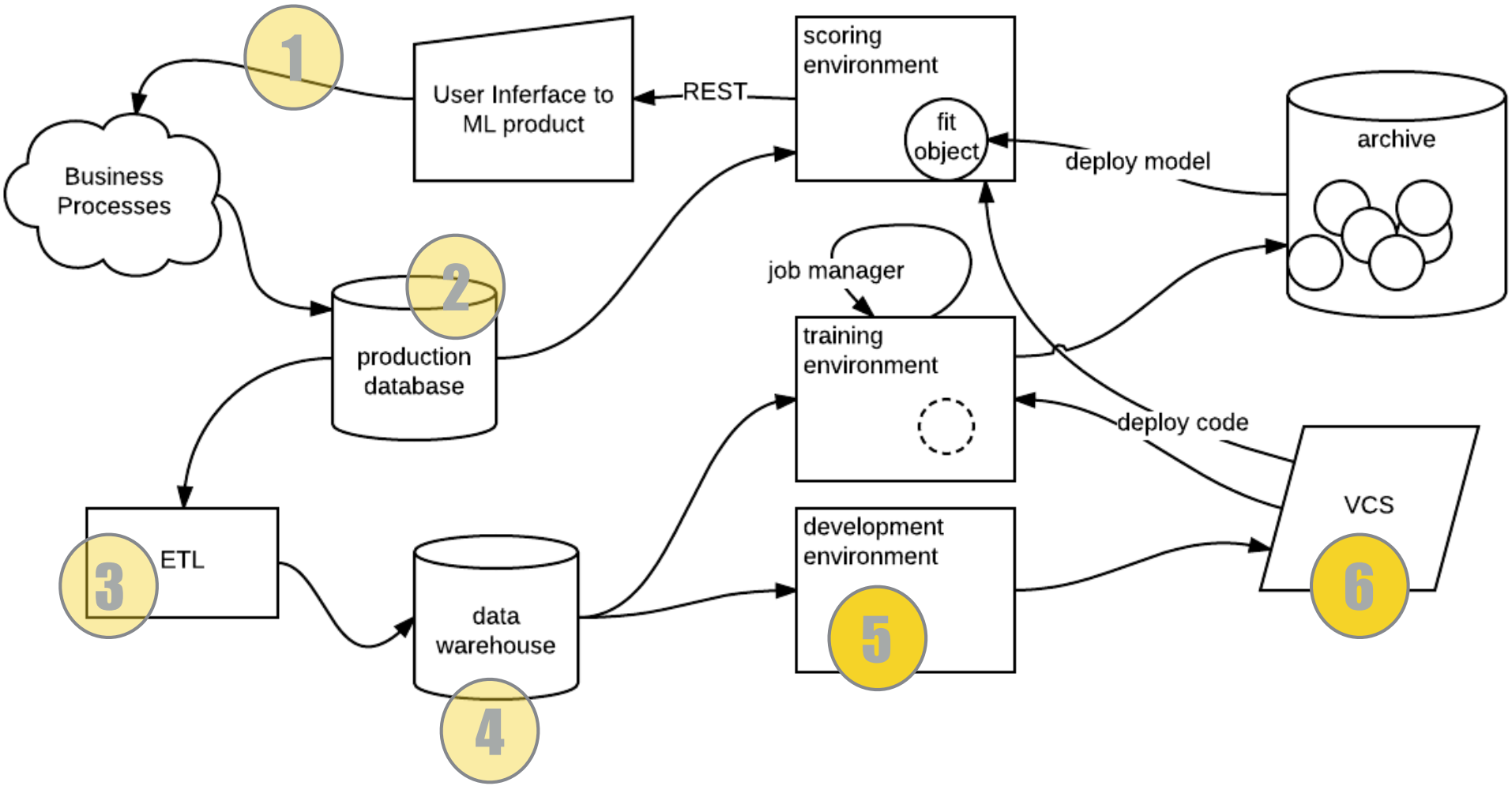

1
User Inferface to
ML product
REST
scoring
environment
fit
object
deploy model
archive
Business
Processes
2
production
database
OLTP
job manager
training
environment
deploy code
VCS
ETL
3
data
warehouse
development
environment
OLAP
4

Extract

Transform

Load

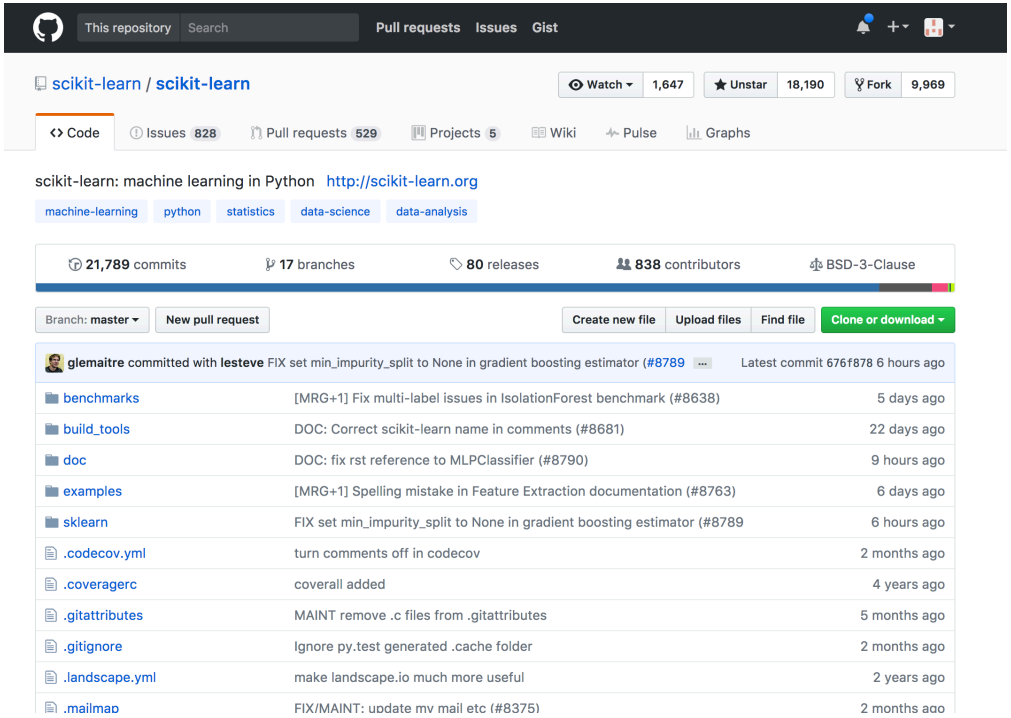



1
User Inferface to
ML product
REST
scoring
environment
fit
object
deploy model
archive
Business
Processes
2
production
database
job manager
training
environment
deploy code
VCS
ETL
3
data
warehouse
development
environment
5
6
4

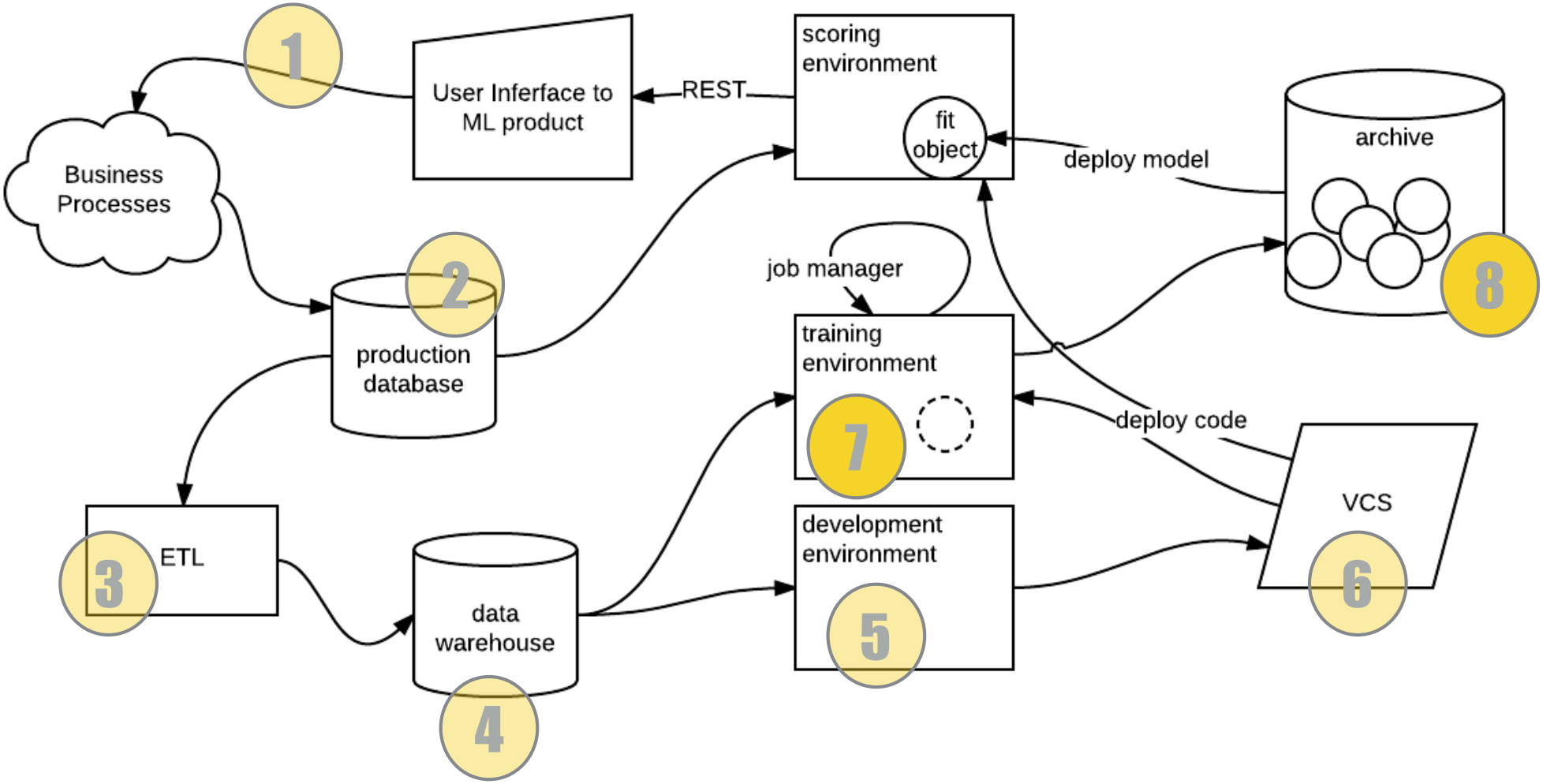
VCS
Version Control System

these days, this mostly means git used with GitHub

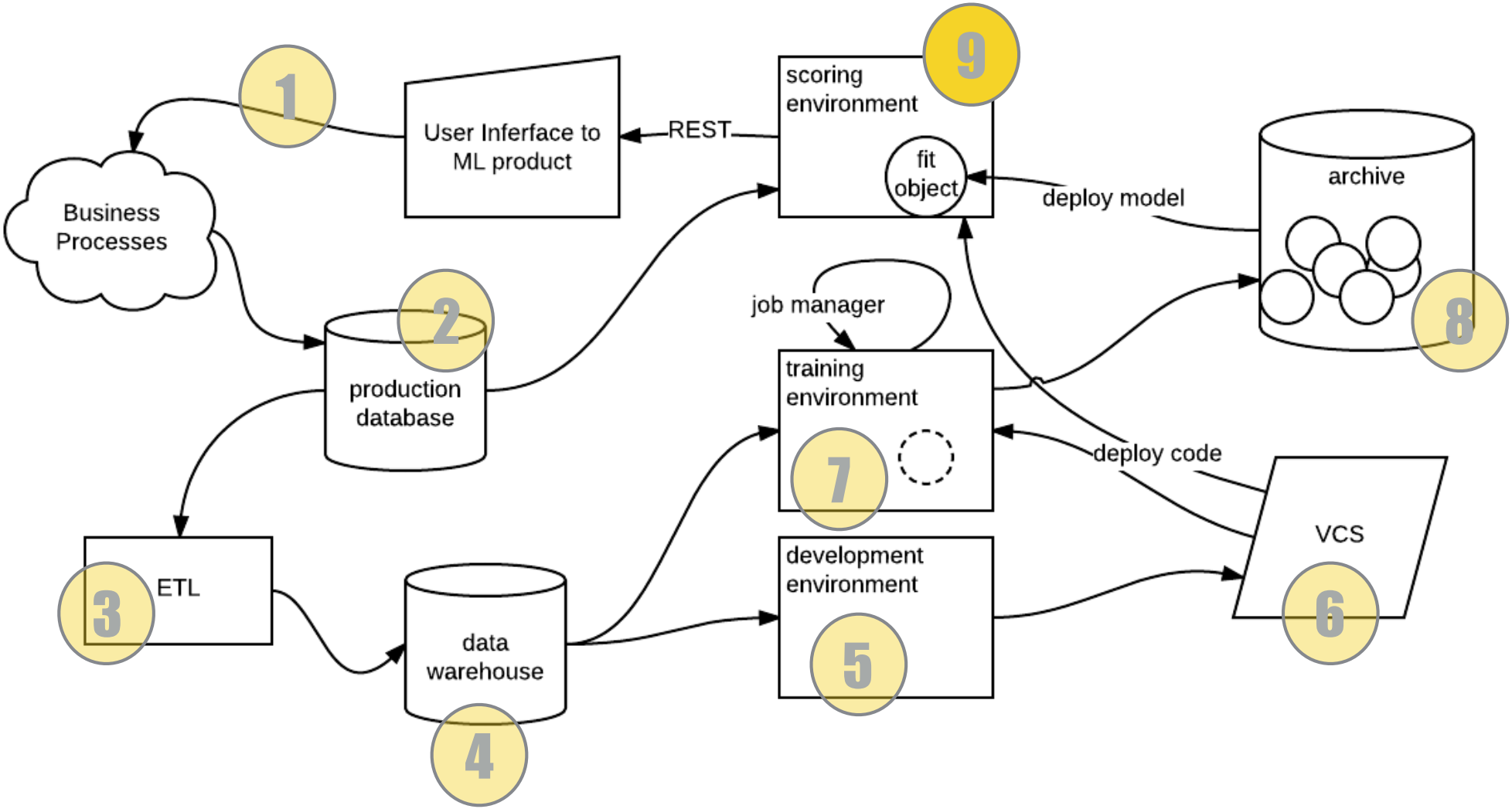
but also mercurial, subversion (svn), others…



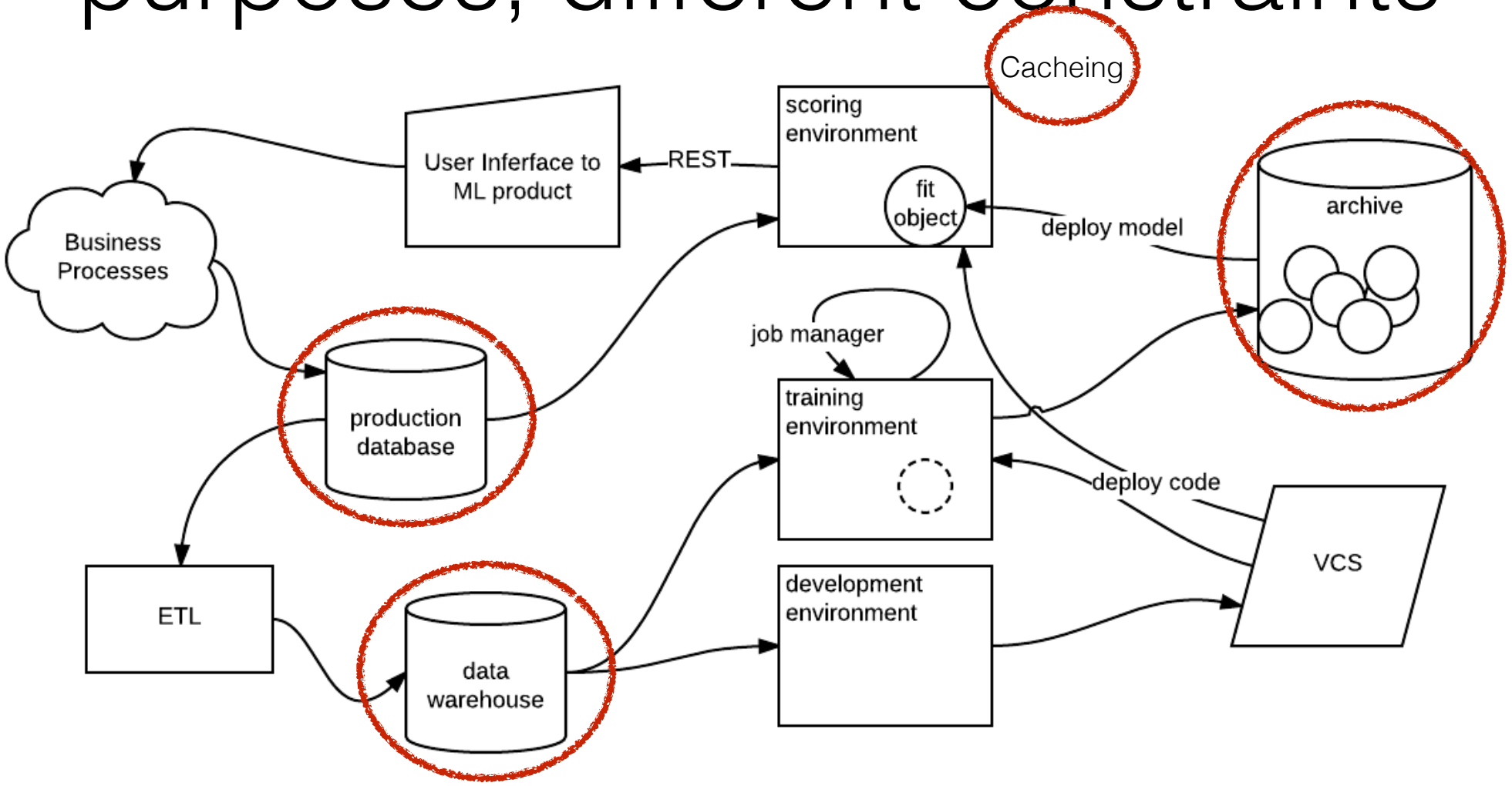
training in production



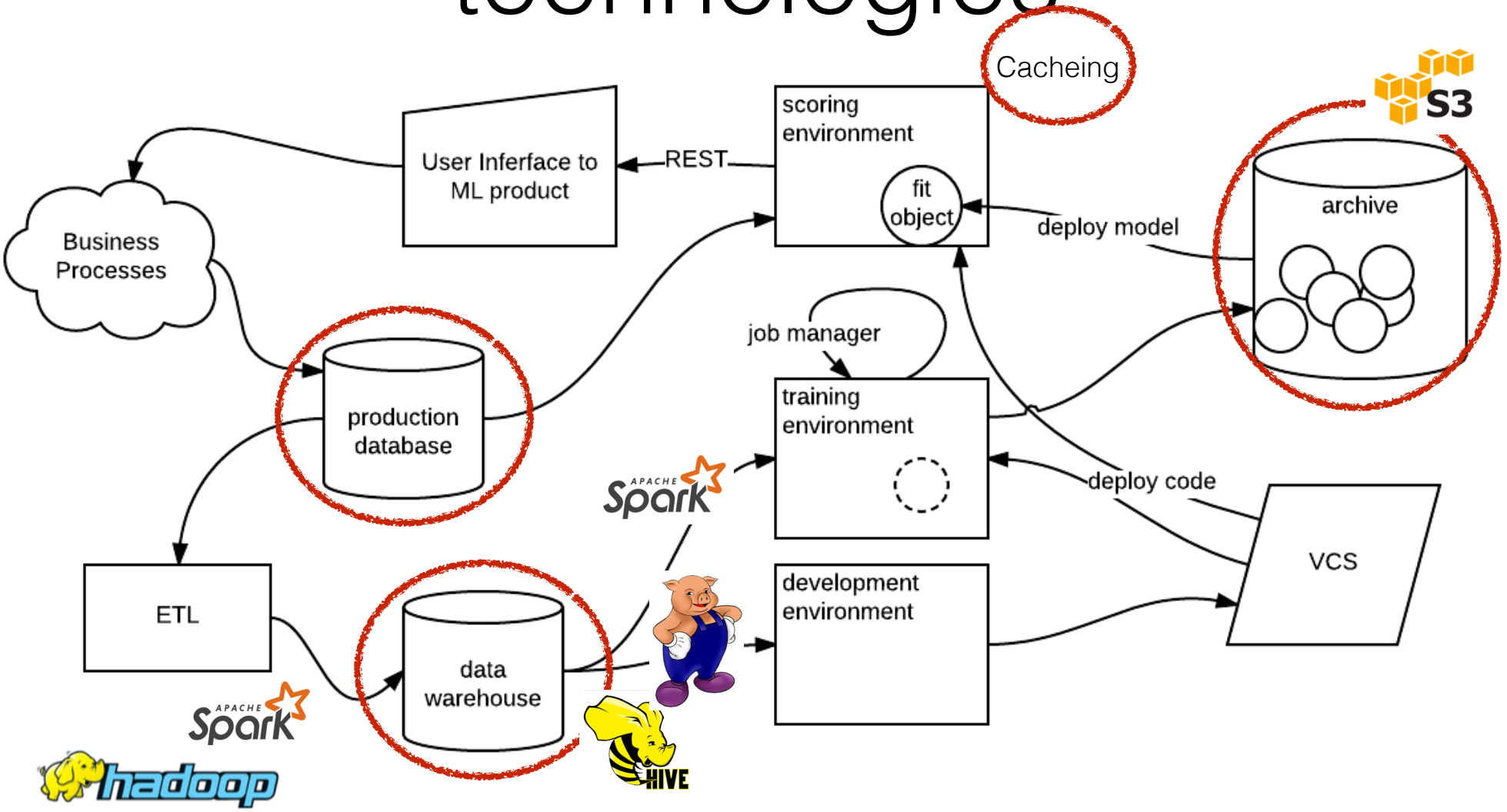
scoring service



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

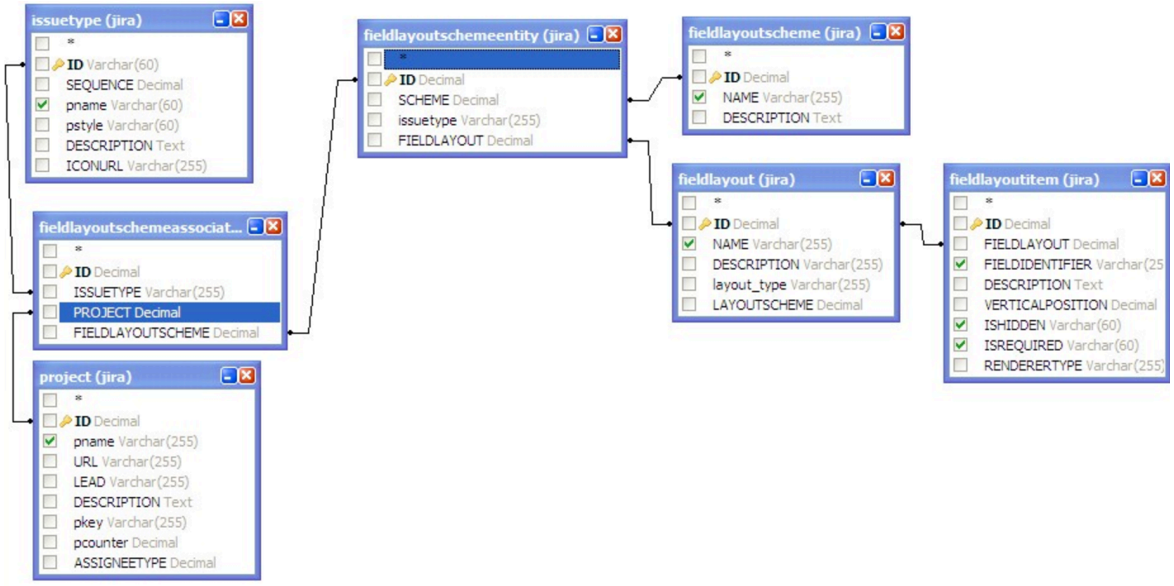
the Relational Model

- 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

- field_name: type
- Id, foreign/ primary

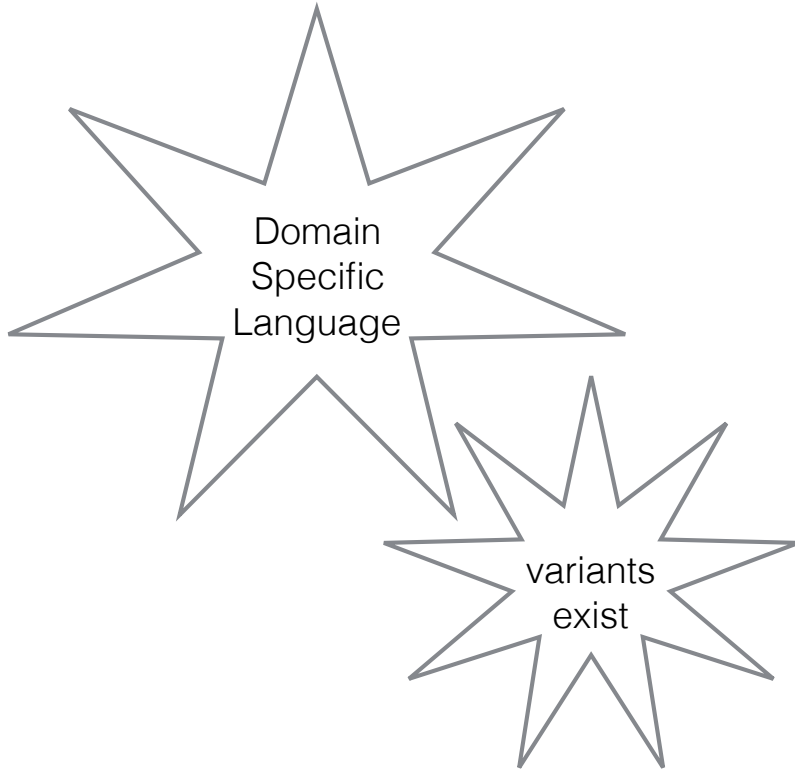
Relational db data model



Structured

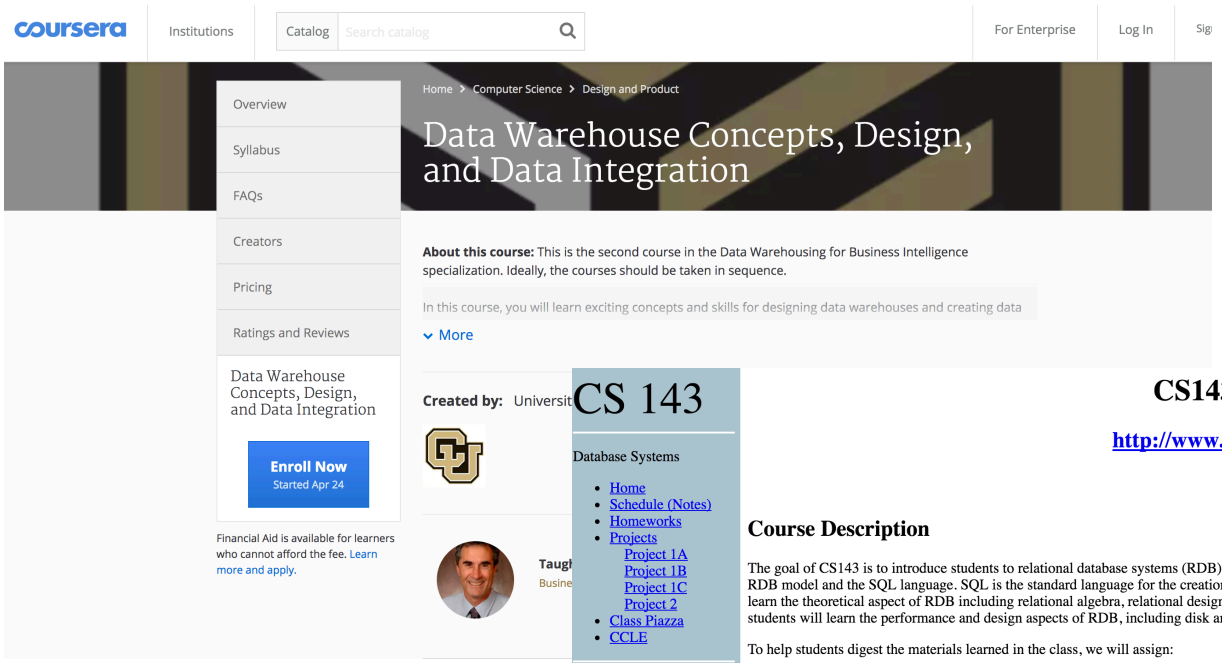
Query

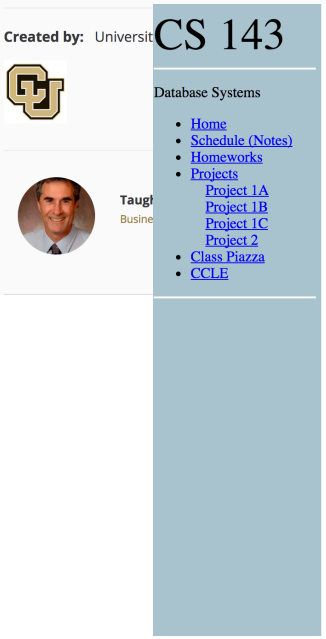
Language



relational databases

not a bad idea to take a databases course





CS143: Database Systems

http://www.cs.ucla.edu/classes/fall14/cs143/

Fall 2014

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.

To help students digest the materials learned in the class, we will assign:

- 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.

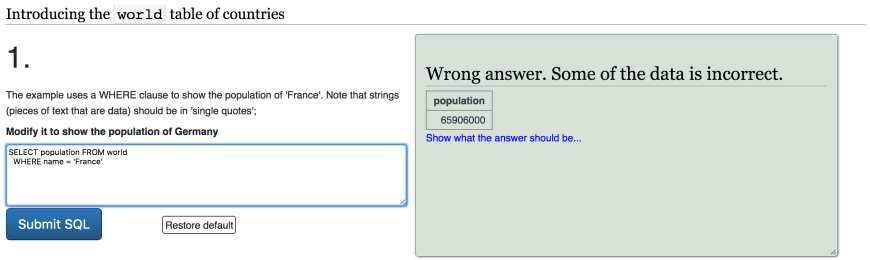
Time and Place

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.

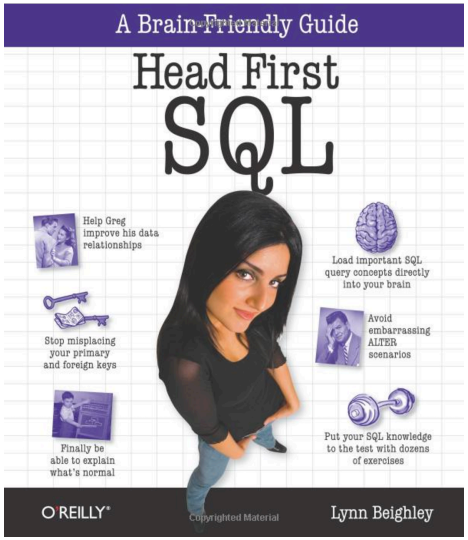
SQL learning resources

- SQL Zoo: https://sqlzoo.net/



- Learn SQL the Hard Way ($20): https://learncodethehardway.org/sql/

- Head First SQL: http://www.headfirstlabs.com/books/hfsql/



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

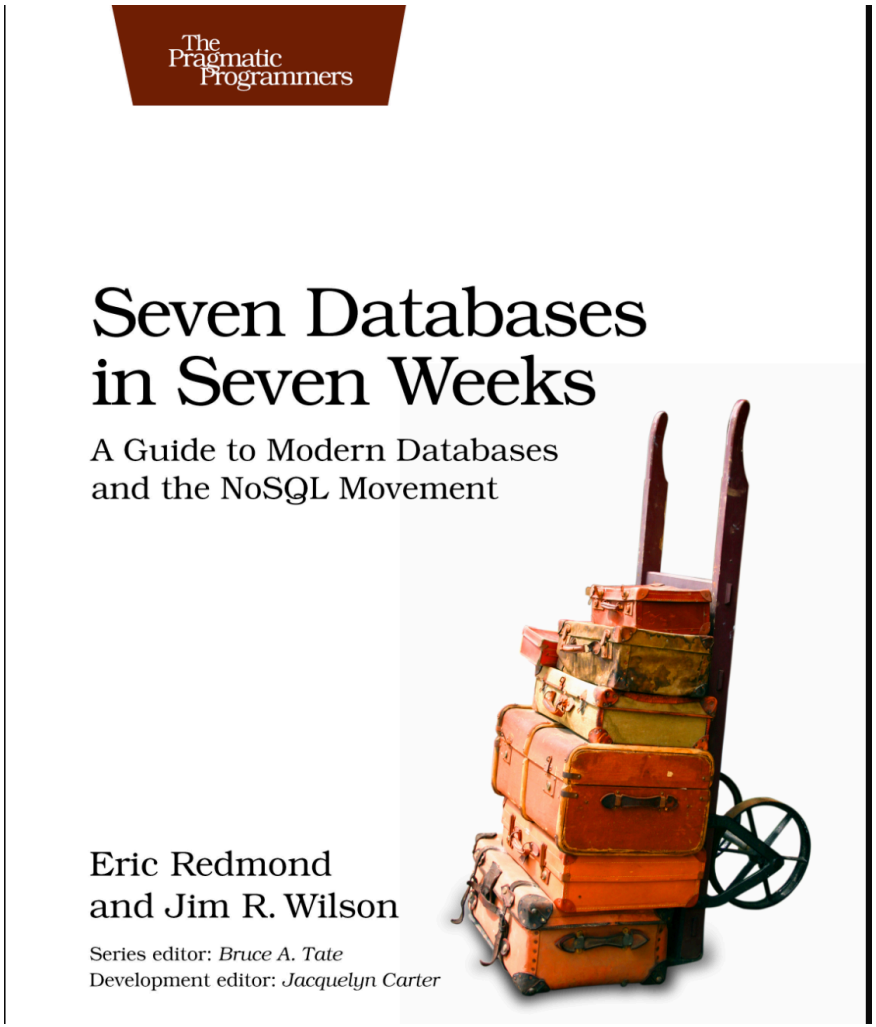
8:45 - 8:50 Wrap up

Lecture 3:
Data Store Diversity
noSQL, redis and mongo

Q: why so many?
A: trade-offs

Data Store Typology

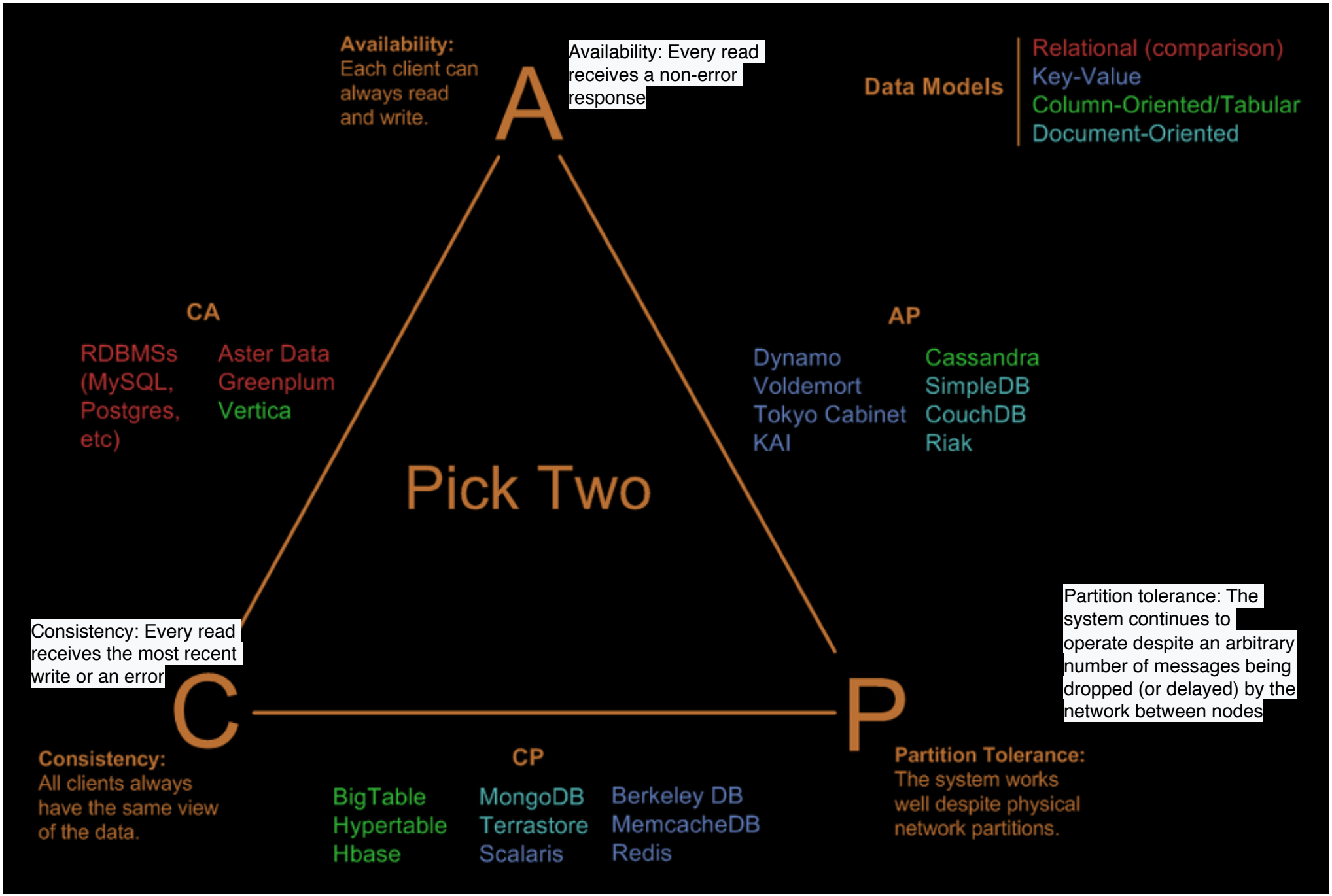
1. relational
2. columnar
3. key-value store
4. document oriented
5. graph



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

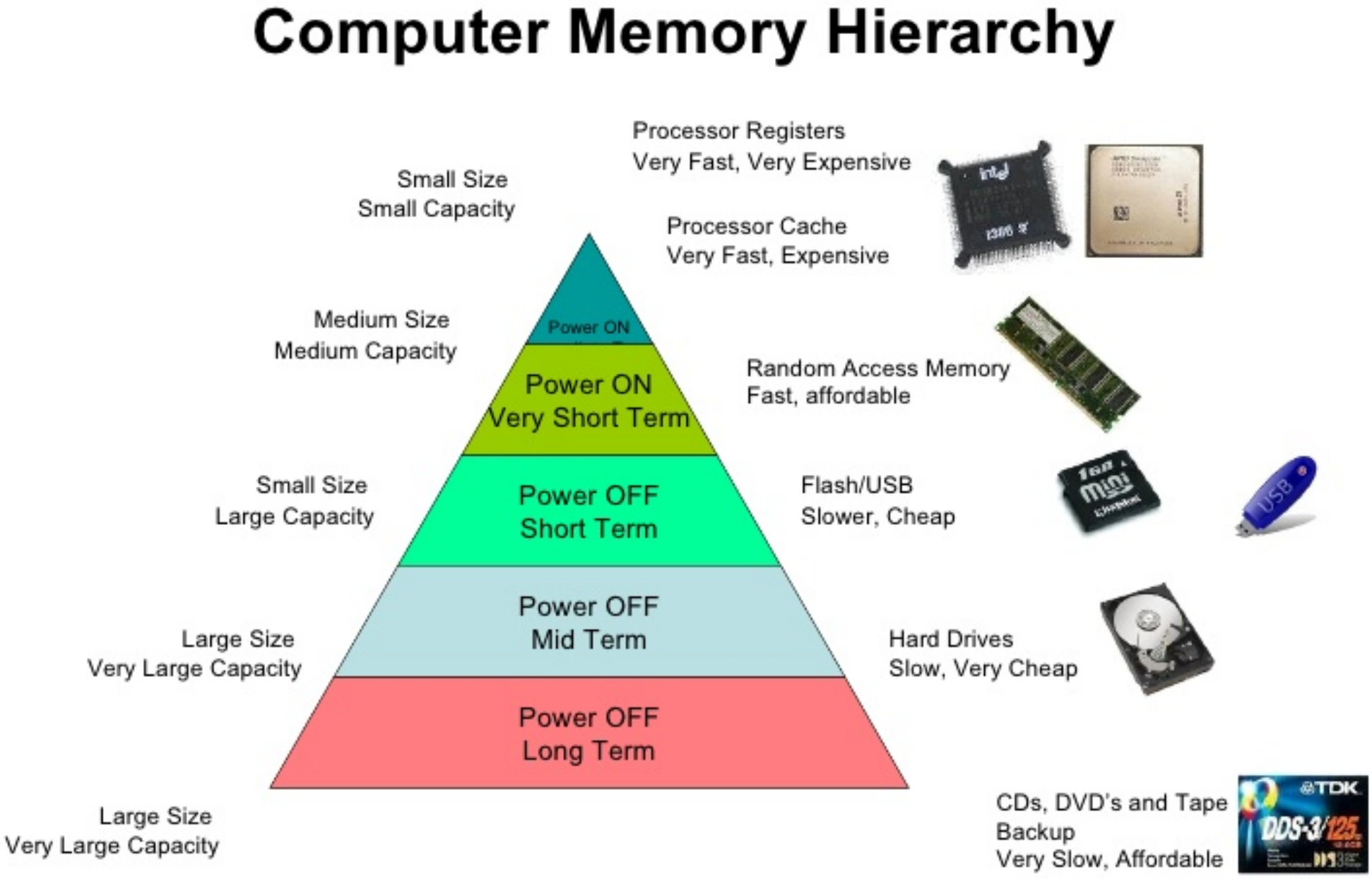
CAP theorem


Availability:
Each client can always read and write.
A
Availability: Every read receives a non-error response
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
Consistency: Every read receives the most recent write or an error
C
P
Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

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?



top redis applications

1. cacheing
    - 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

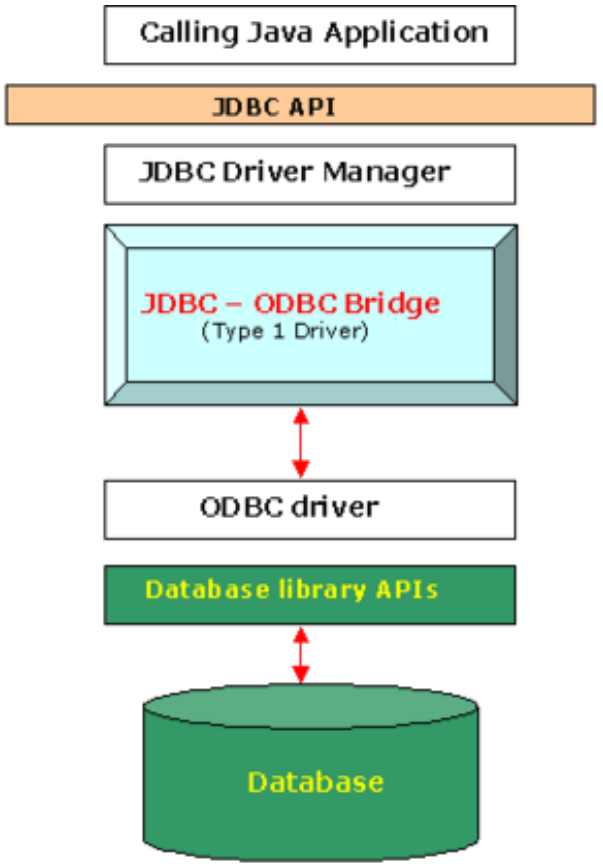
Lecture 4: Crucial Practicalities drivers, connections and interfaces

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

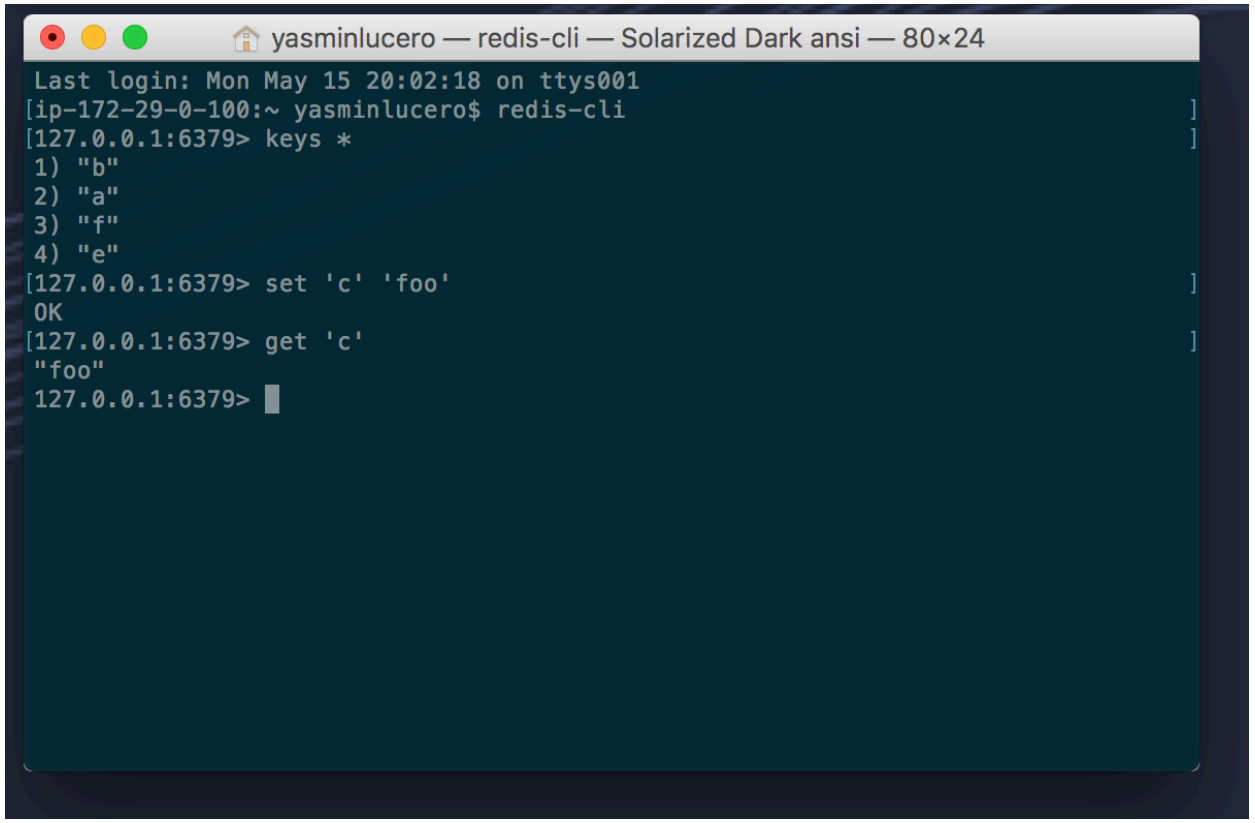
<protocol>:// <host>:<port>/<schema>;<property>=<value>

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



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.

Next: Output to connections, Previous: Connections, Up: Connections [Con

7.1 Types of connections

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

Unix programmers are used to dealing with special files `stdin`, `stdout` and s
also refer to input from and output to a GUI console. (Even with the standard

connections {base}

Functions to Manipulate Connections (Files, URLs, ...)

Description

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

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 it’s 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

relational	mongo
databases	databases
tables	collections
rows	documents
columns	fields

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

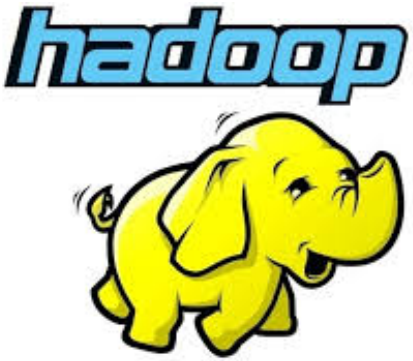
8:15 - 8:45 Lecture 5: Distributed computing, Hadoop, Spark, Hive, Pig

8:45 - 8:50 Wrap up

Lecture 5:
distributed data computation
Hadoop, sharding, Spark

distributed means many machines, it means cluster

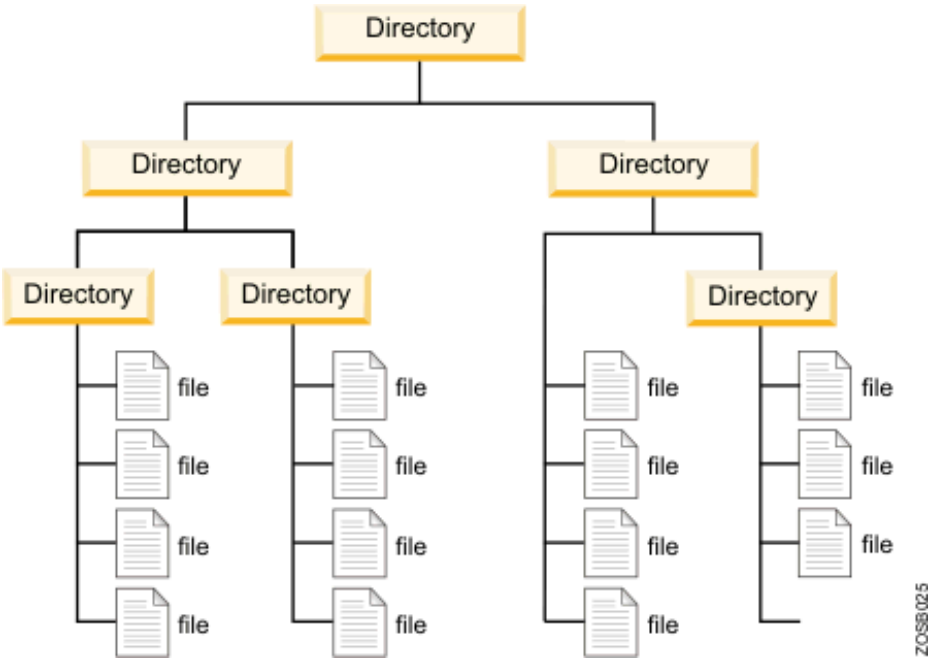
- distributed compute must be parallel, but not all parallel computation is distributed
- network IO
- replication
- manager nodes
- you only do it if you have to
- big data requires distribution



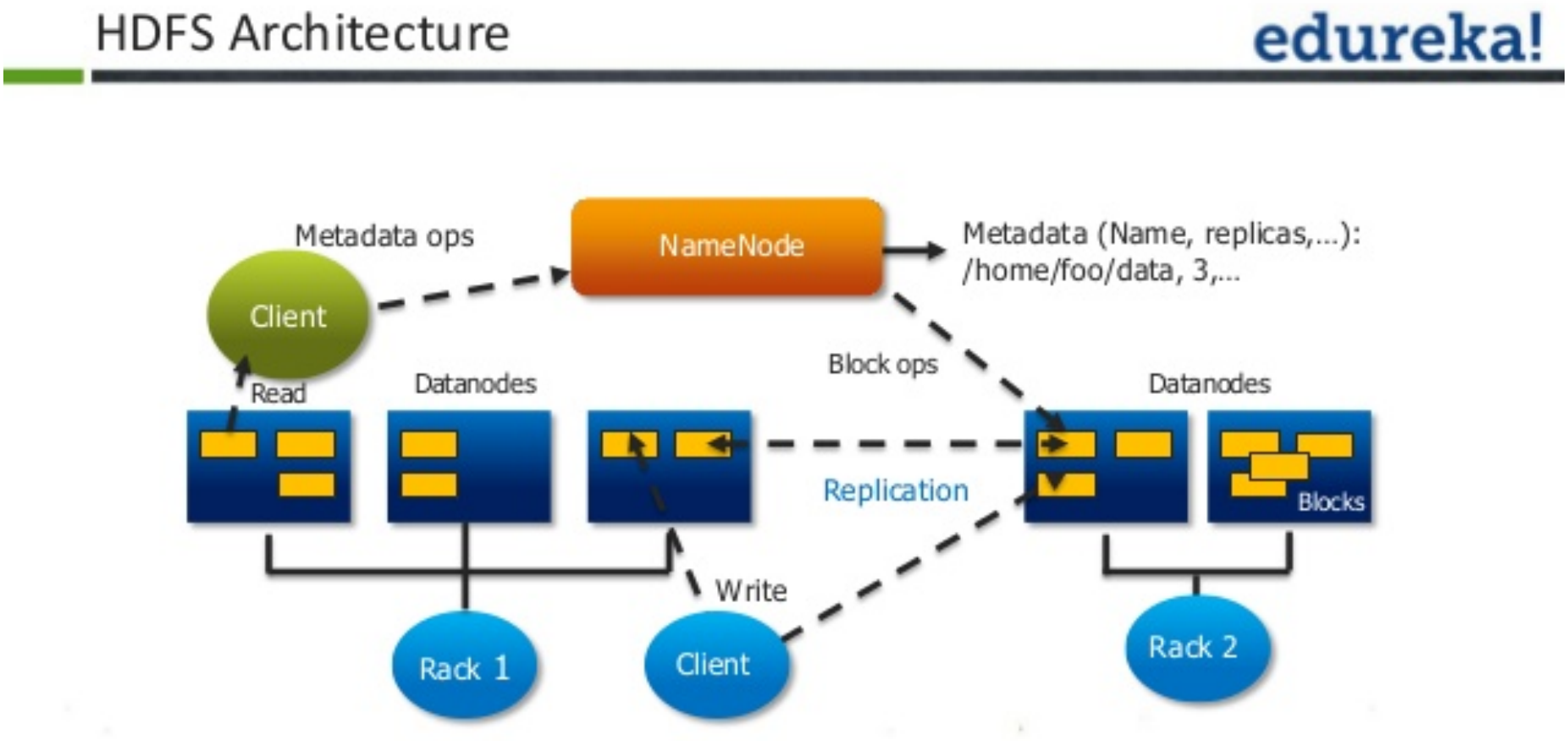
Hadoop is actually two distinct things:
HDFS and MapReduce

a file system

- ls
- cp
- mv
- touch
- rm



HDFS: Hadoop distributed file system

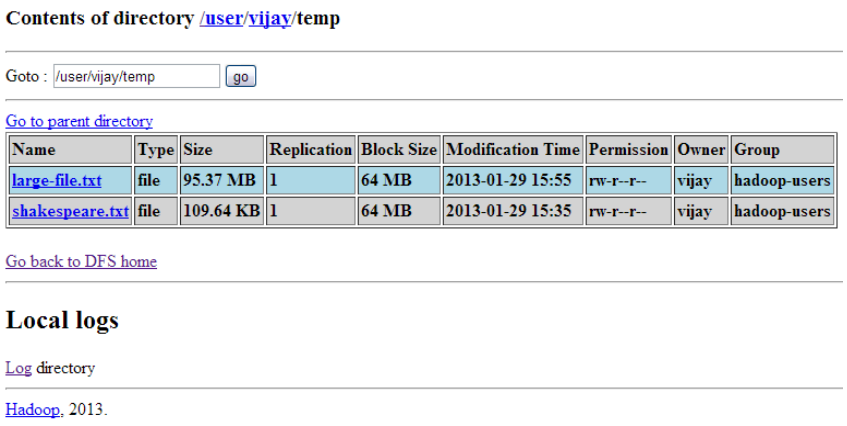


problems solved by HDFS

- 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 : /user/vijay/temp go

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

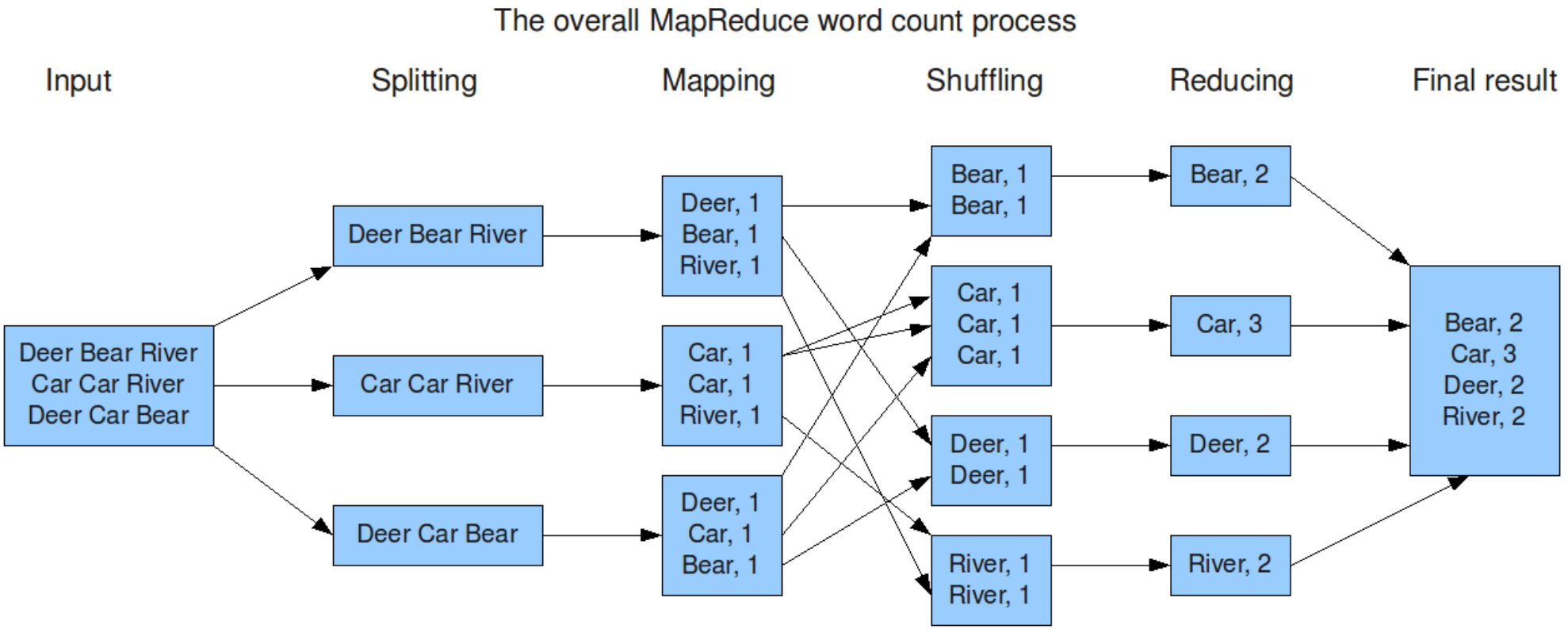
Go back to DFS home

Local logs

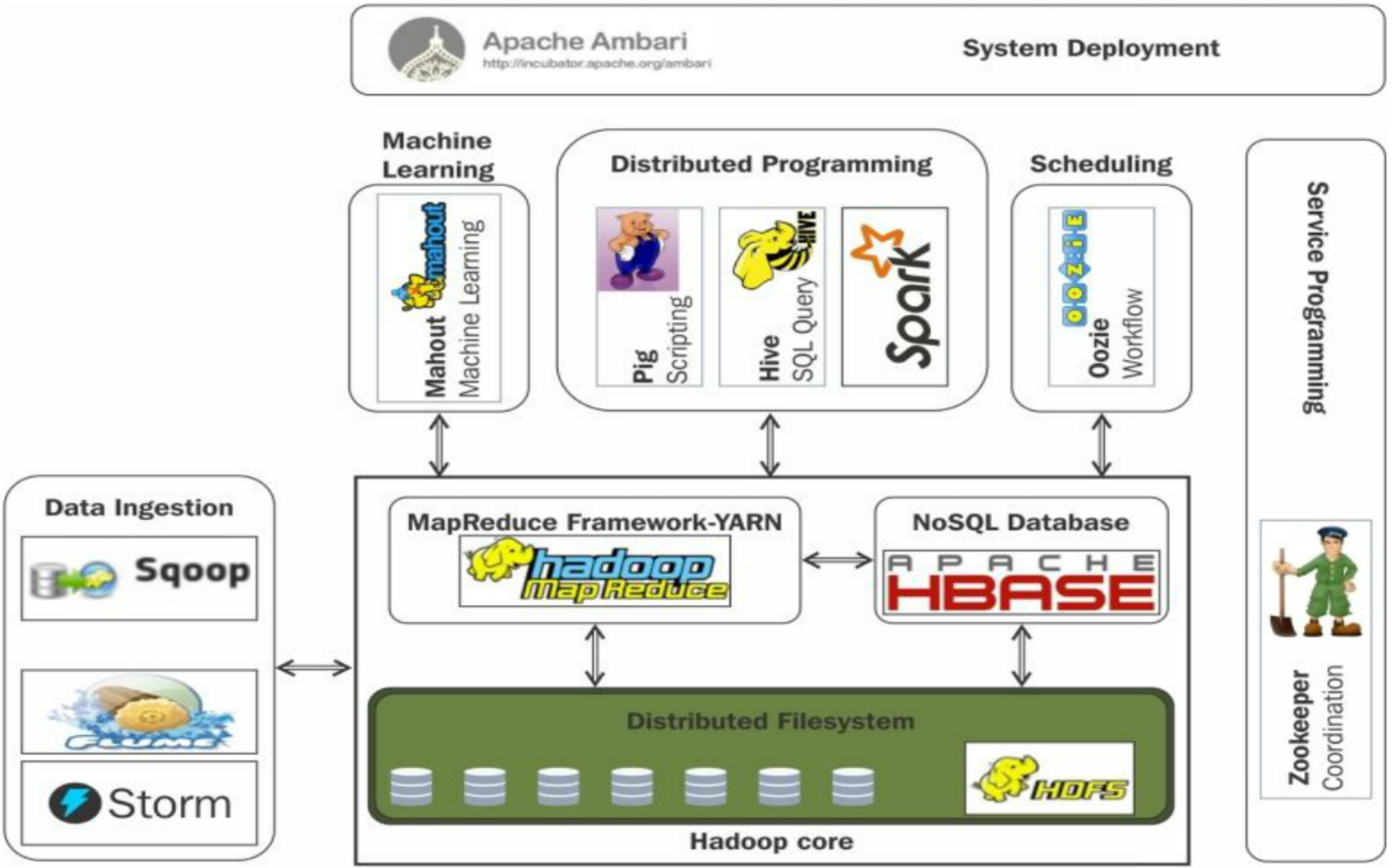
Log directory

Hadoop, 2013.

MapReduce: split-apply-combine idiom



Hadoop Eco System

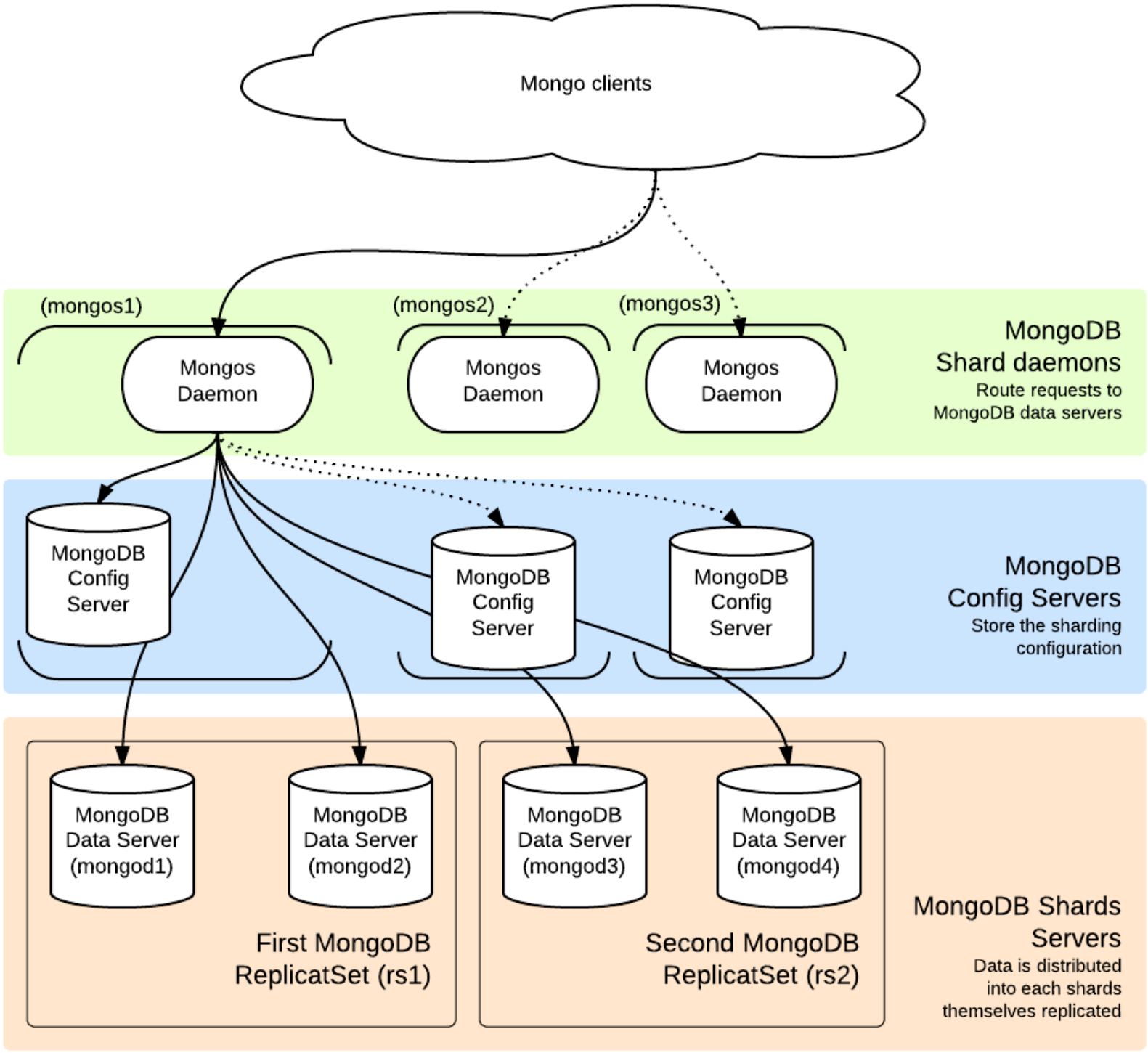


distributed data bases

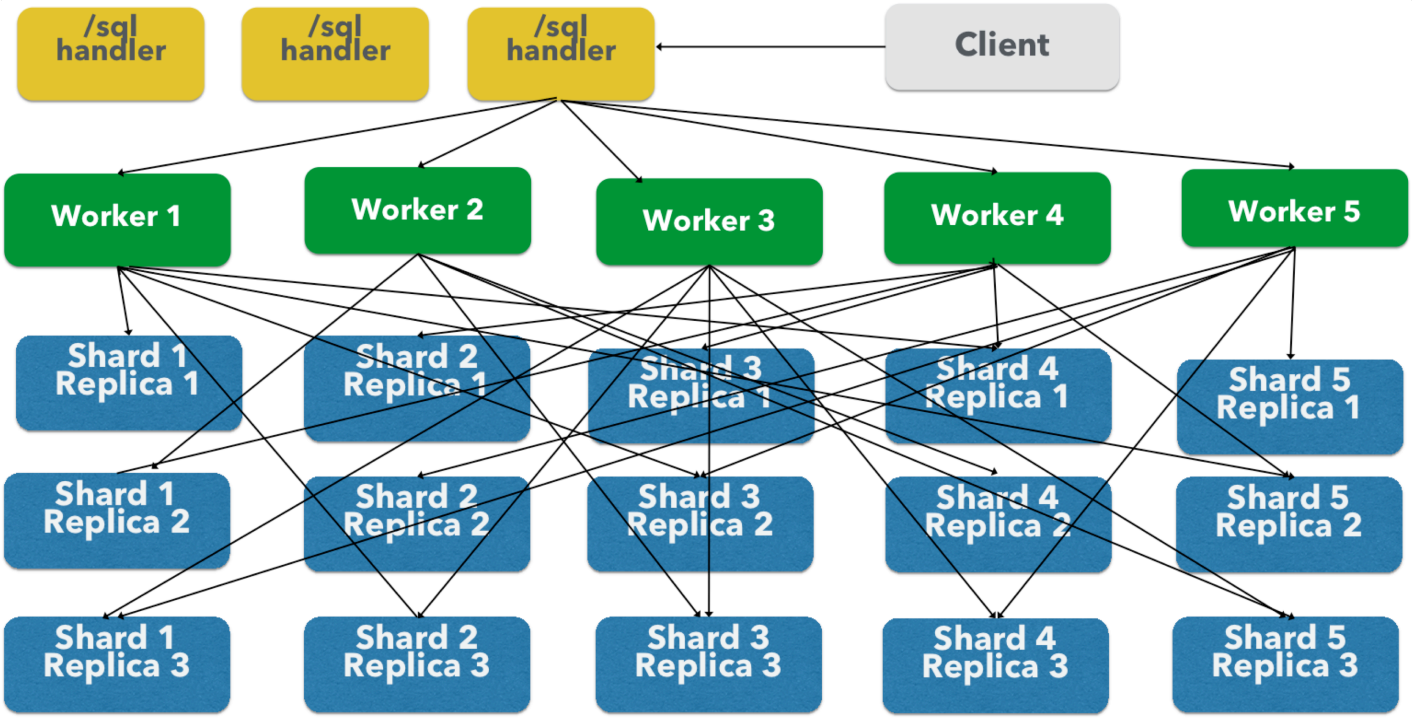
A **database shard** is a horizontal partition of data in a database or search engine. Each individual partition is referred to as a **shard** or **database shard**. Each shard is held on a separate database server instance, to spread load.

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)


Mongo clients
(mongos1)
(mongos2)
(mongos3)
Mongos Daemon
Mongos Daemon
Mongos Daemon
MongoDB Shard daemons
Route requests to MongoDB data servers
MongoDB Config Server
MongoDB Config Server
MongoDB Config Server
MongoDB Config Servers
Store the sharding configuration
MongoDB Data Server (mongod1)
MongoDB Data Server (mongod2)
MongoDB Data Server (mongod3)
MongoDB Data Server (mongod4)
First MongoDB ReplicatSet (rs1)
Second MongoDB ReplicatSet (rs2)
MongoDB Shards Servers
Data is distributed into each shards themselves replicated

Distributed SQL DB



distributed Postgres

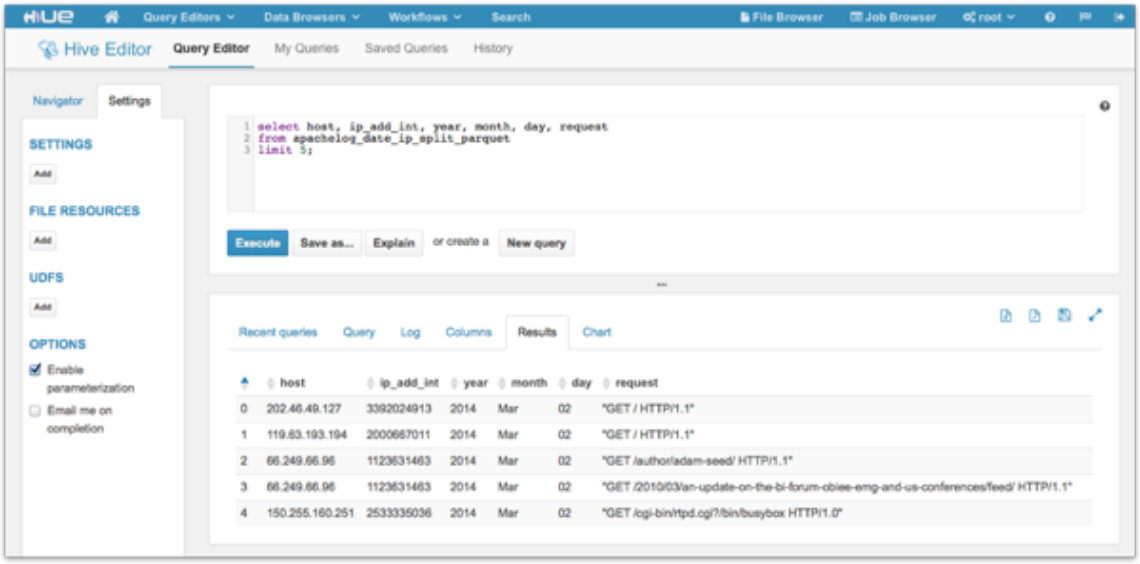
Program	License	Maturity	Replication Method	Sync	Connection Pooling	Load Balancing	Query Partitioning
PgCluster	BSD	Not production ready	Master-Master	Synchronous	No	Yes	No
pgpool-I	BSD	Stable	Statement-Based Middleware	Synchronous	Yes	Yes	No
Pgpool-II	BSD	Recent release	Statement-Based Middleware	Synchronous	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

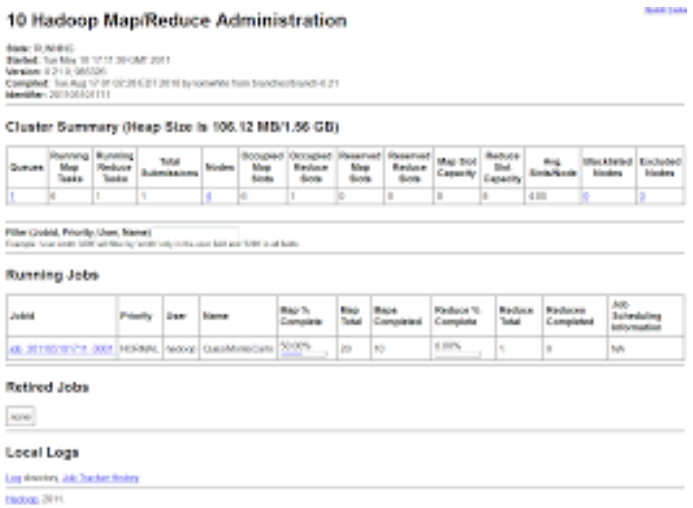
distributed compute

Hive:
SQL on HDFS

you write sql:

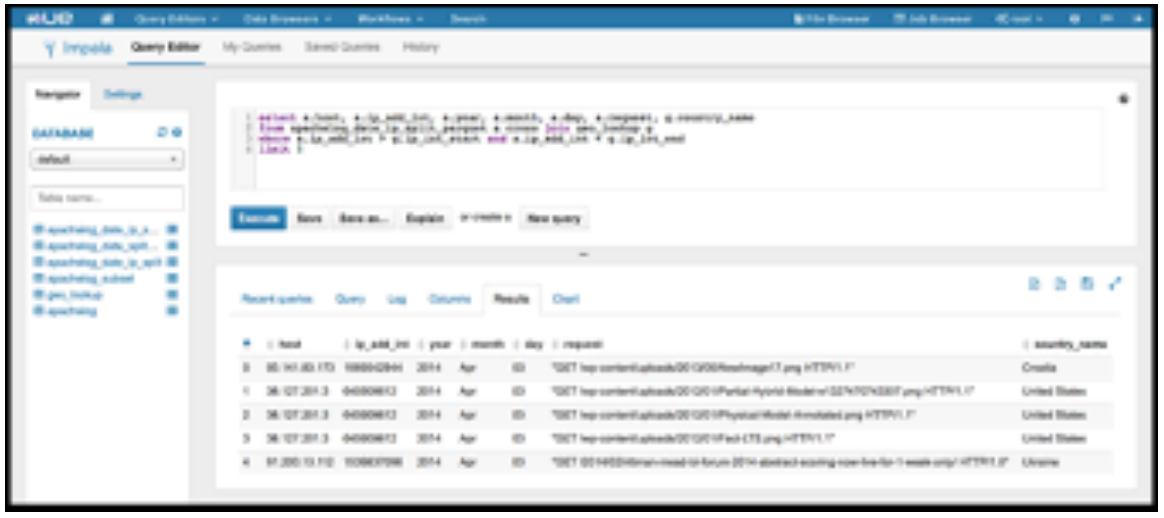
hive submits a MapReduce job:





Impala: Also SQL on HDFS

You write SQL, get your answers back super fast



The catch:

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

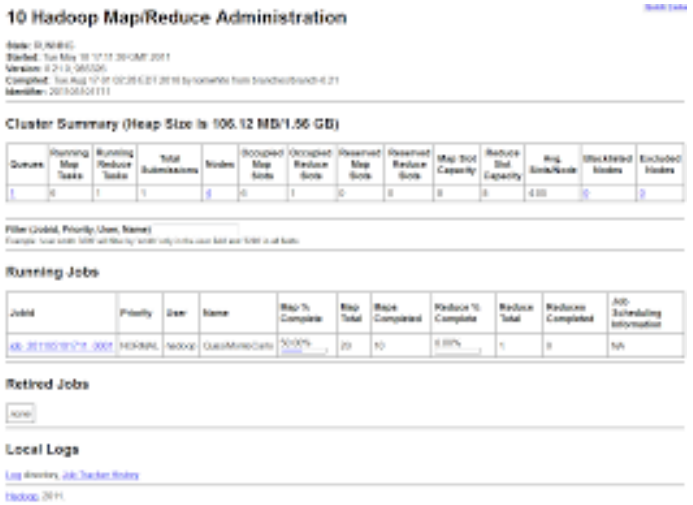
Apache Pig

You write a pig script:

Pig submits a MapReduce job:

- **Sample Pig script:**

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



- 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
- cacheing 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

```scala
// 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

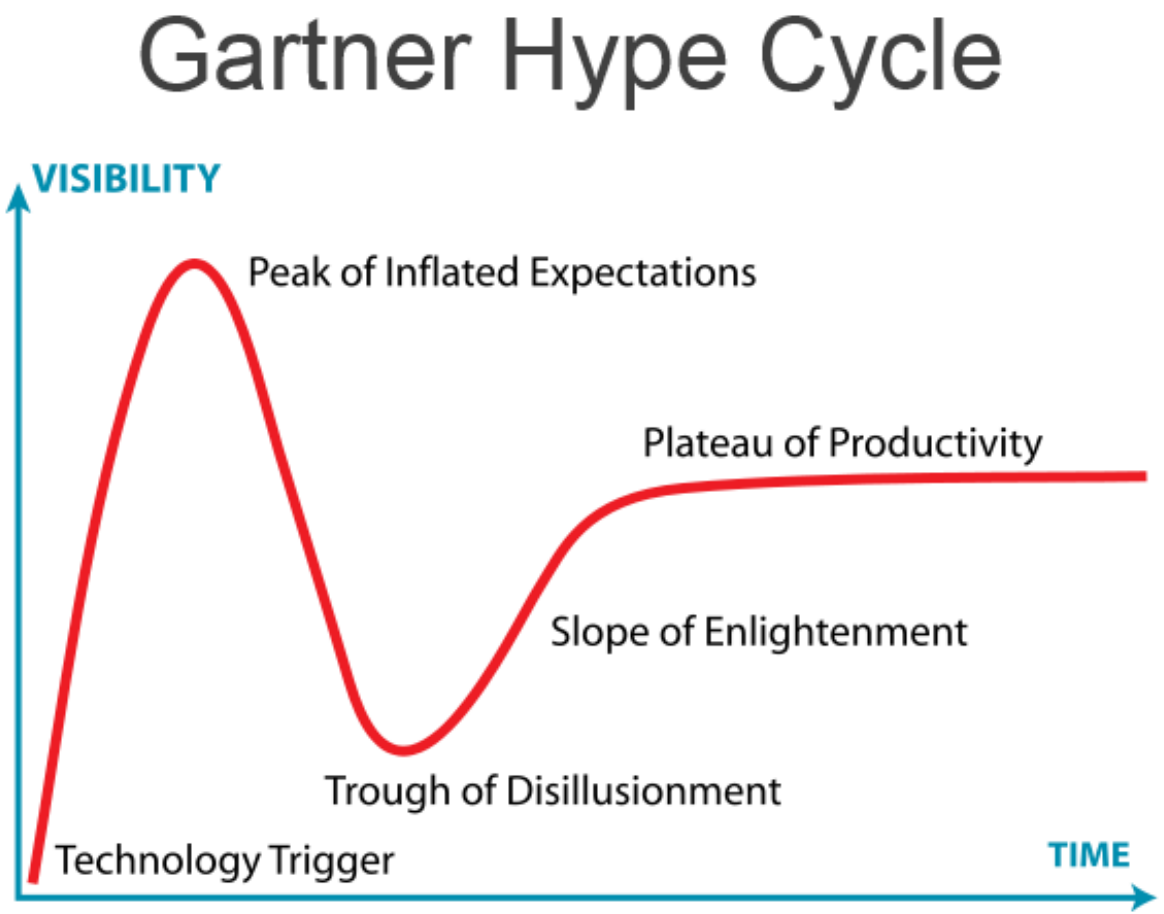
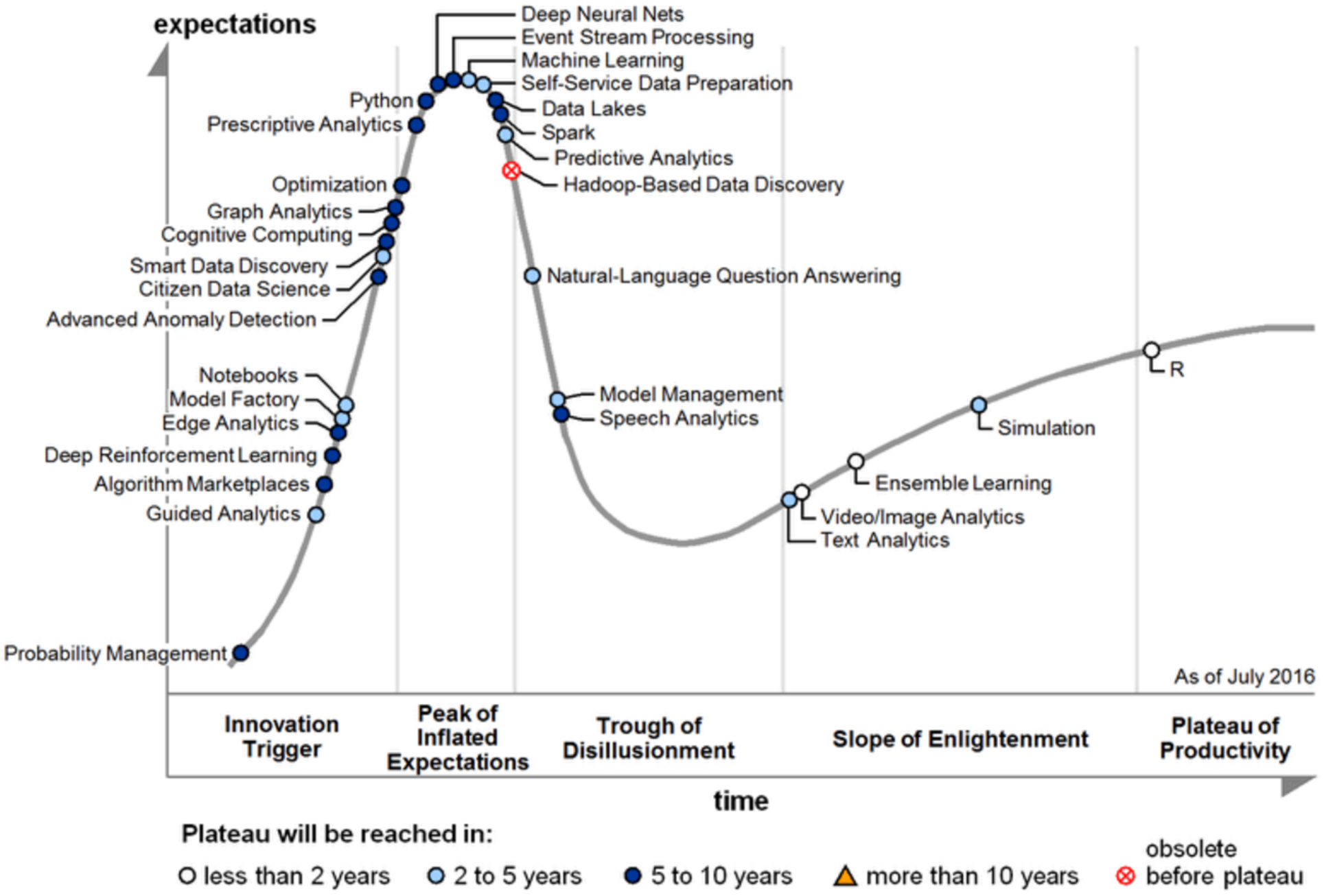


Figure 1. Hype Cycle for Data Science, 2016



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