Introduction to Databases

Ivan Corneillet

Data Scientist

GENERAL ASSEMBLY

Learning Objectives

After this lesson, you should be able to:

- Understand uses and differences of databases, including:
  - RDBMS and SQL databases
  - NoSQL databases
- Access databases from *pandas*

DS

Announcements and Exit Tickets

Today

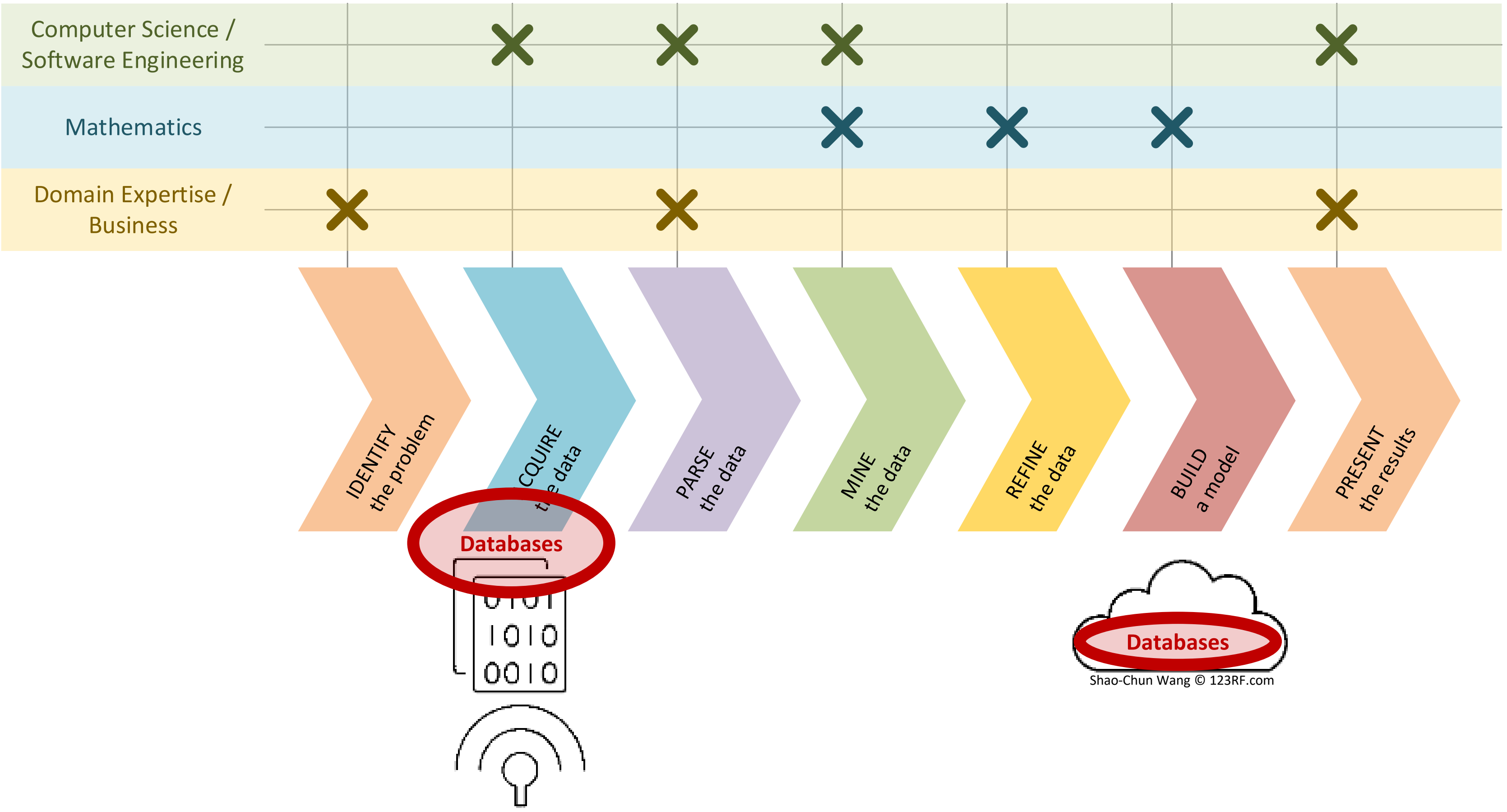
Here’s what’s happening today:

- Announcements and Exit Tickets
- Review
- RBDMS/SQL Databases
  - Tables and Schema
  - Normalized vs. Denormalized Structures
- NoSQL (Not-Only SQL) Databases
  - NoSQL Classification
  - CAP Theorem
  - ACID vs. BASE
  - Popular NoSQL Databases
  - CRUD and REST
  - Map Reduce
- Codealong with SQLite and CouchDB
- Review
- Exit Tickets

In this class, we will address the last topic of the course: Databases

Research Design and Data Analysis	Research Design	Data Visualization in *pandas*	Statistics	Exploratory Data Analysis in *pandas*
Foundations of Modeling	Linear Regression	Classification Models	Evaluating Model Fit	Presenting Insights from Data Models
Data Science in the Real World	Decision Trees and Random Forests	Time Series Models	Natural Language Processing	Databases

Databases is an essential part to connect Data Science to the real world: ❷ ACQUIRE the data (e.g., from Big Data, IoT) as well as after ❻ **BUILD** a model when large-scale productizing machine learning models (e.g., on the cloud)

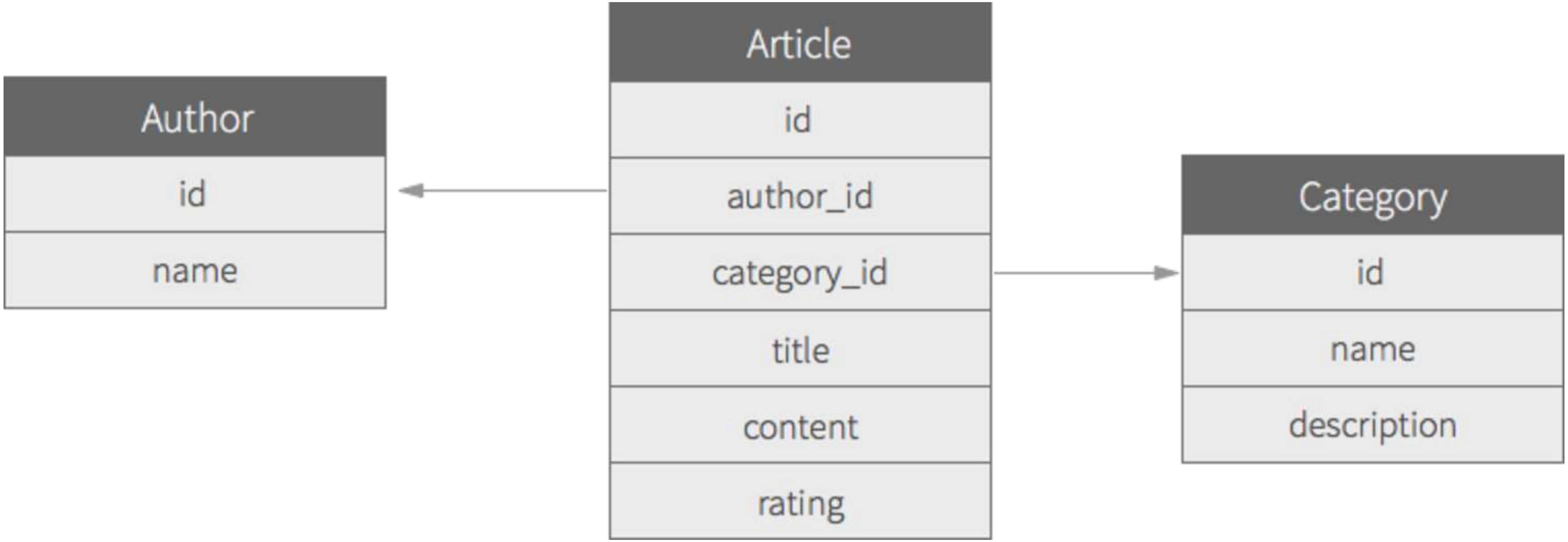


	IDENTIFY the problem	CQUIRE t e data	PARSE the data	MINE the data	REFINE the data	BUILD a model	PRESENT the results
Computer Science / Software Engineering		✕	✕	✕			✕
Mathematics				✕	✕	✕	
Domain Expertise / Business	✕		✕				✕

Shao-Chun Wang © 123RF.com

Relational Database Management Systems (RDBMS) and Structured Query Language (SQL)

A relational database links data entities and concepts. E.g., a database linking books, their authors and categories



Author
id
name

Article
id
author_id
category_id
title
content
rating

Category
id
name
description

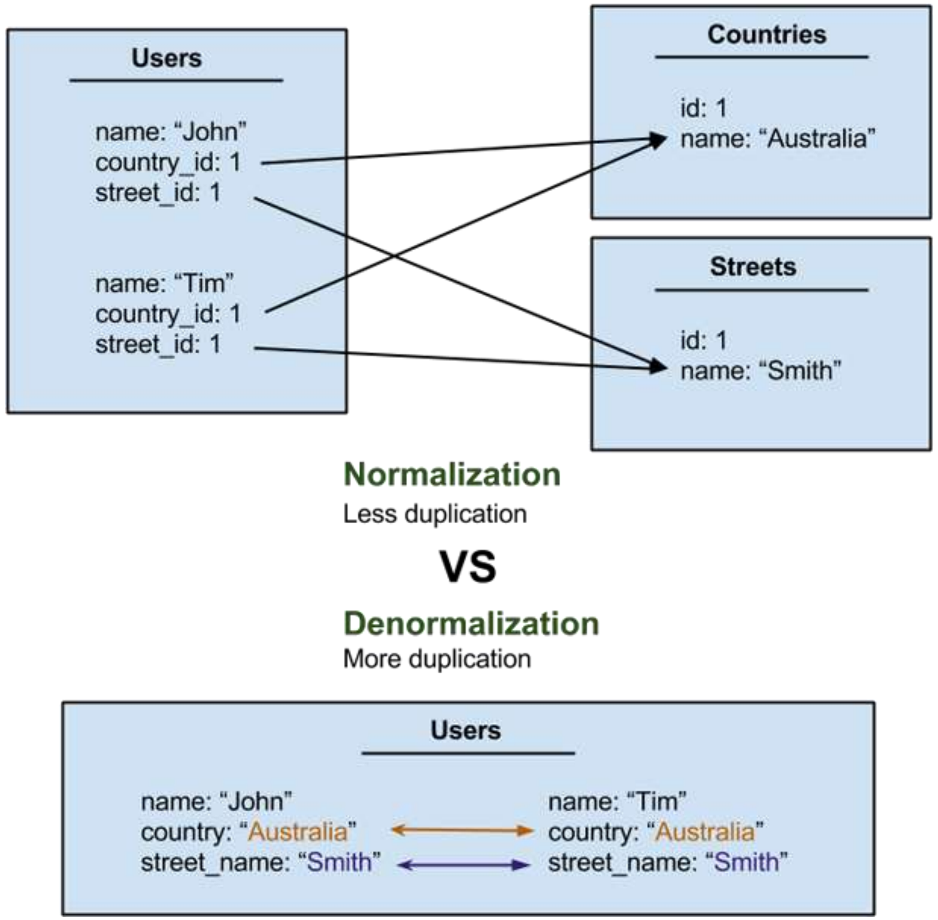
Relational databases are organized into *tables.* Each table has a specific *schema*, a set of rules for what goes in each table

- Each table corresponding to one entity or concept
- A table is made up of rows and columns, similar to a *pandas* dataframe

- Schemas specify which columns are contained in the table and what type of data is in each column (e.g., text, integer, or date). This means you can't add text data to an integer column
- For this reason and many others, databases allow for stronger consistency of the data and are often a better solution for data storage

Once we start organizing our data into tables, we start to separate it into *normalized* and *denormalized* setups

- *Normalized* structures have a single table per entity and use many foreign keys or link tables to connect the entities
- *Denormalized* structures have fewer tables that combine different entities



Normalized vs. Denormalized Data

Normalized

- More tables
- Fewer columns per table
- Fewer rows per table
- Less redundancy
- More joins
- Update efficient

Denormalized

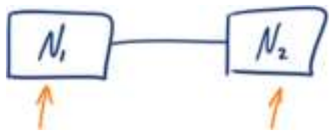
- Fewer tables
- More columns per table
- More rows per table
- Greater redundancy
- Fewer joins
- Read efficient
- Duplicates a lot of information

NoSQL (Not-Only SQL) Databases

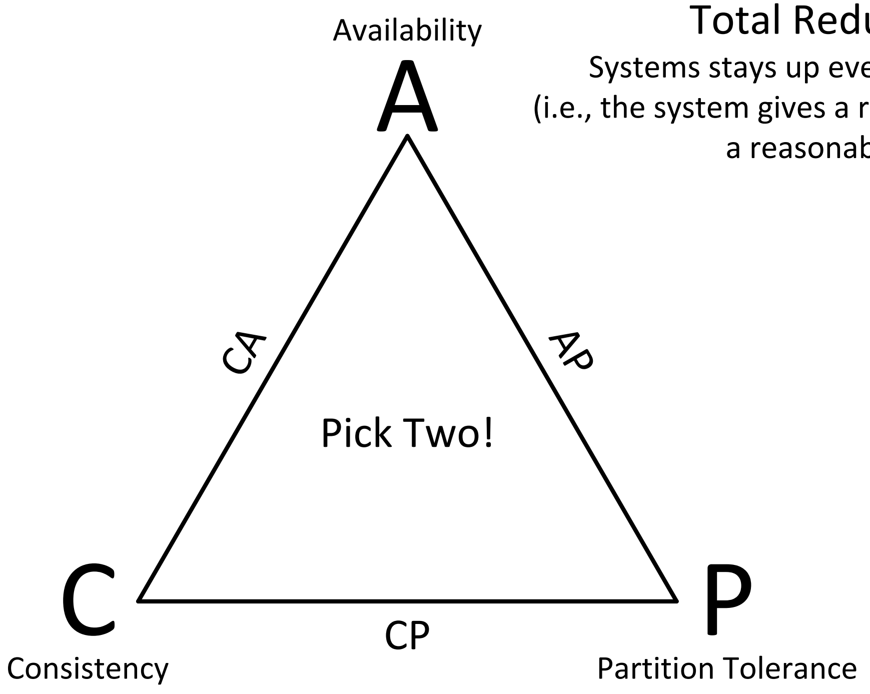
NoSQL databases fall into four primary classifications

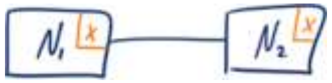
- **Key-value stores** – use a simple data model that pairs a unique key and its associated value in storing data elements. Common uses include storing clickstream data and application logs
- **(Wide)-column stores** (a.k.a., table-style databases) – store data across tables that can have very large numbers of columns. Common uses include Internet search and other large-scale Web applications
- **Document databases** – store data elements in document-like structures that encode information in formats such as JSON. Common uses include content management and monitoring Web and mobile applications
- **Graph databases** – emphasize connections between data elements, storing related "nodes" in graphs to accelerate querying. common uses include recommendation engines and geospatial applications

The CAP theorem states that in the presence of a network partition (P), one has to choose between consistency (C) and availability (A)

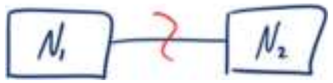


Total Redundancy
Systems stays up even when nodes fail
(i.e., the system gives a reasonable response in
a reasonable time)





ACID Transactions
Everyone always have the same view of the data (regardless of any pending updates or deletes)



Infinite Scaleout
System stays up even when network between nodes fail (i.e., only a total network failure can cause the system to respond incorrectly)

ACID vs. BASE: The pH of Database Transaction Processing

ACID

- **Atomicity** – all operations are performed or none of them are. If one part of the transaction fails, then all fail
- **Consistency** – a transaction must meet all rules defined by the system at all times; there are never any half-completed transactions
- **Isolation** – transactions are independent from each other
- **Durability** – once complete, a transaction cannot be undone

BASE

- **Basically Available** – the system will give and accept queries and give responses even in regards to node failures
- **Soft State** – the data is in a constant state of flux and might be stale
- **Eventual Consistency** – the data will eventually be consistent through all nodes and in all databases, but not every transaction at every moment

AC/AP/CP vs. ACID/BASE

AC

- Small datasets can be both consistent and available but a non-option in distributed systems (networks aren't completely reliable so you must tolerate partitions)

AP
BASE w/ eventual consistency

- System returns the most recent version of the data it has (which could be stale). The system will also accept writes that can be processed later when the partition is resolved
- Choose AP (over C) when you are flexible on when the data in the system synchronizes

CP
ACID w/ eventual availability

- System waits for a response from the partitioned node which could result in a timeout error
- Choose CP (over A) when you require atomic reads and writes

Popular NoSQL databases(*)

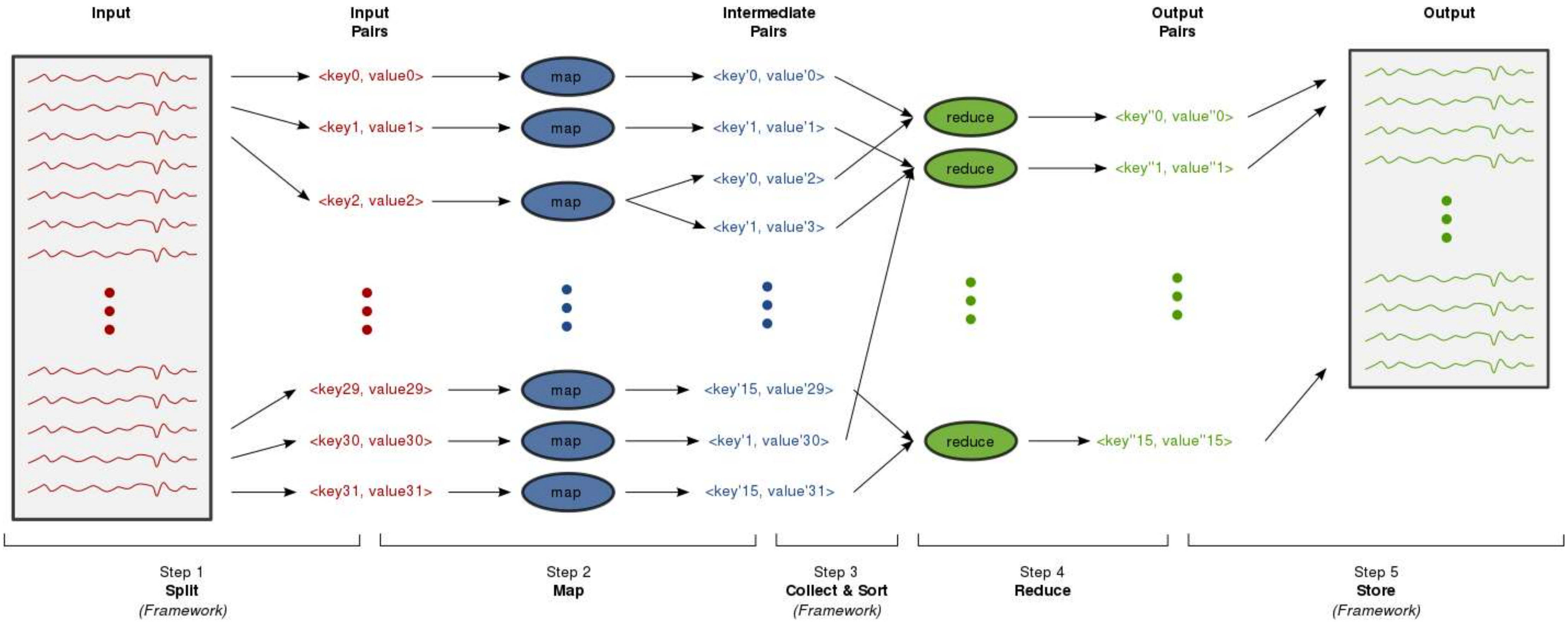
(*) Many systems allows you to tune both the write and the read quorums can be either CP or AP, depending on your needs.

Accumulo	Wide-column store	CP/ACID
Cassandra	Wide-column store	AP/BASE
CouchDB	Document database	AP/BASE
DynamoDB	Key-value store	AP/BASE
HBase	Wide-column store	CP/ACID
MongoDB	Document database	CP/ACID
Neo4j	Graph database	CP/ACID
Redis	Key-value store	CP/ACID
Riak	Key-value store	AP/BASE
SimpleDB	Wide-column store	AP/BASE

CRUD and REST

CRUD[*] [*] **the four basic functions of persistent storage**	**HTTP**[**] [**] **Methods for RESTful services**	
Create	POST	Create or add new entries
Read	GET	Read, retrieve, search, or view existing entries
Update	POST	Update or edit existing entries
Delete	DELETE	Delete/deactivate/remove existing entries

Map Reduce


Input
Input Pairs
Intermediate Pairs
Output Pairs
Output
<key0, value0>
<key1, value1>
<key2, value2>
<key29, value29>
<key30, value30>
<key31, value31>
map
<key'0, value'0>
<key'1, value'1>
<key'0, value'2>
<key'1, value'3>
<key'15, value'29>
<key'1, value'30>
<key'15, value'31>
reduce
<key''0, value''0>
<key''1, value''1>
<key''15, value''15>
Step 1
Split
(Framework)
Step 2
Map
Step 3
Collect & Sort
(Framework)
Step 4
Reduce
Step 5
Store
(Framework)

Exit Ticket

Don’t forget to fill out your exit ticket here