Parallelism

Master 1 International

Andrea G. B. Tettamanzi
Université de Nice Sophia Antipolis
Département Informatique
andrea.tettamanzi@unice.fr

Lecture 7 – Part a

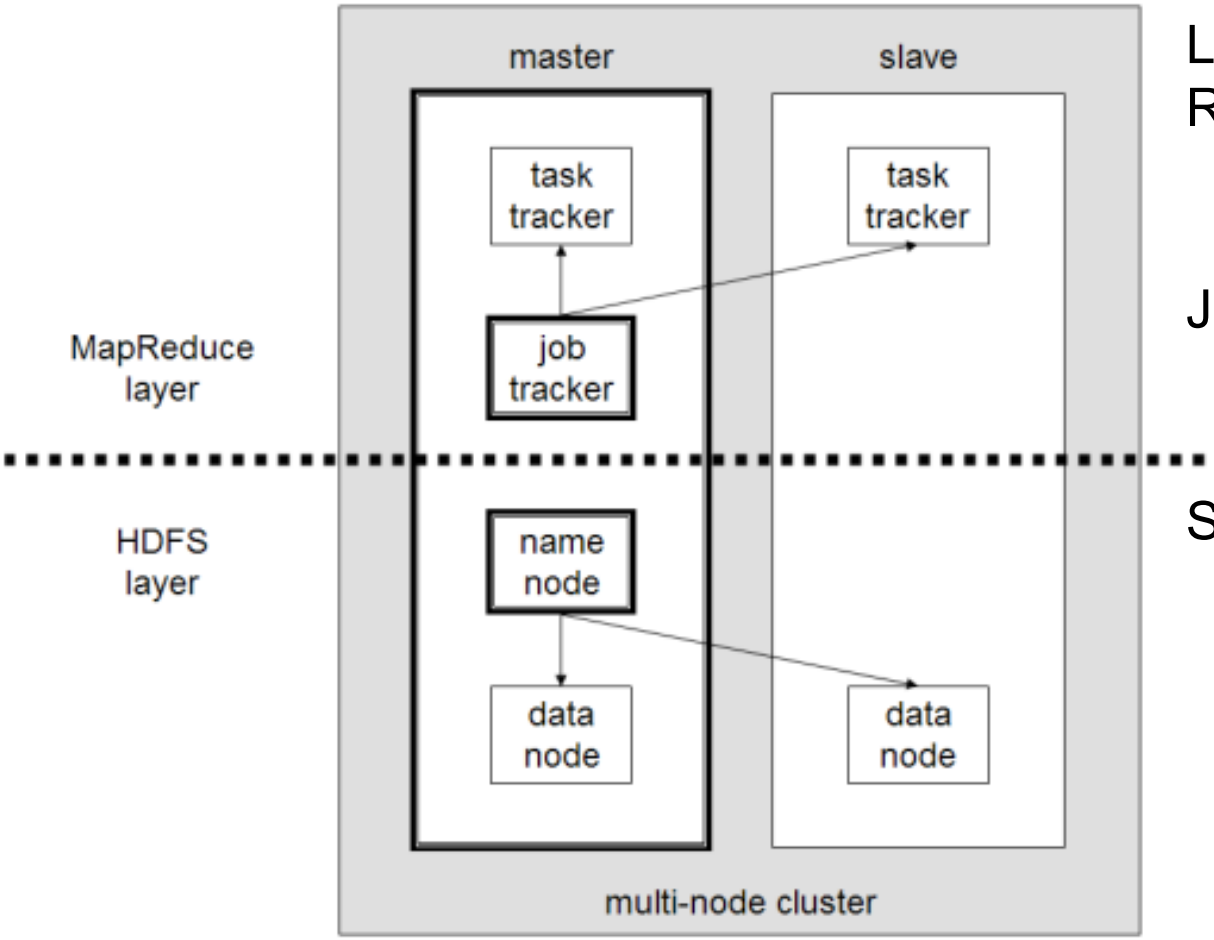
Distributed Computing and Data Base Systems for the Big Data

Plan

- NoSQL Databases
- Hadoop and MapReduce
- Apache Spark
- Document-based DBs: MongoDB
- Graph-based DBs
- RDF, triple stores, and SPARQL (→ MOOC)
- The CAP Theorem

- Apache Hadoop is a software library
- Framework for the distributed processing of large data sets
- Designed to
  - scale up from single servers to 1000s of machines
  - detect and handle failures at the application layer
- Four modules (components)
  - Hadoop Common: file-system and OS-level abstractions
  - Hadoop distributed file system (HDFS)
  - Hadoop YARN: job scheduling and cluster resource mgmt
  - MapReduce: parallel processing of large data sets
- Related projects: Cassandra, HBase, Spark, and several others

Hadoop Architecture



Location awareness:
Rack (= switch) name

JRE 1.6 or higher

SSH

HDFS

- Distributed, scalable, and portable file-system written in Java
- A Hadoop cluster has nominally a single namenode plus a cluster of datanodes, although redundancy options are available
- Each datanode serves up blocks of data over the network using a block protocol specific to HDFS.
- TCP/IP sockets used for communication
- Clients use RPC to communicate with one another
- Large files stored across multiple machines
- Reliability achieved through replication (cf. Part b of this lecture)

MapReduce (1)

- A programming model for processing parallelizable problems on huge datasets using a large number of computers (nodes)
- Inspired by *map* and *reduce* functions in functional programming
- Originally a proprietary Google technology
- MapReduce libraries have been written in many programming languages
- A popular open-source implementation is part of Apache Hadoop
- Key contributions of the MapReduce framework are
  - Scalability (optimized distributed shuffle operation)
  - Fault-tolerance

MapReduce (2)

Three-Step Parallel and Distributed Computation:

- **"Map" operation:** Each worker node applies the “map(key, value)” function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.
- **"Shuffle" operation:** Worker nodes redistribute data based on the output keys (produced by the “map()” function), such that all data belonging to one key is located on the same worker node.
- **"Reduce" operation:** Worker nodes apply the “reduce(key, list of values)” to each group of output data, per key, in parallel.

MapReduce: Example

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += pc
  emit (word, sum)
```

Apache Spark

- The Spark project was started by Matel Zaharia at UC Berkeley to provide programming tools for big data volumes that are easy to use and versatile as those for single machines
- Main motivation: overcome limitations in the MapReduce model
- It has language-integrated APIs in Python, Java, R, and Scala
- Supports streaming, batch, and interactive computations
- Open source (donated to the Apache Software Foundation)
- Implicit data parallelism and fault-tolerance
- Spark requires:
  - cluster manager (e.g., native Spark or Apache YARN),
  - distributed storage system (e.g., HDFS, Cassandra, …)

NoSQL Databases

- Storage and retrieval of data modeled in means other than the tabular relations used in relational databases
- Non-relational databases have existed since the 1960s
- However, real NoSQL (“not only SQL”) databases appeared as a solution for the needs of Web 2.0 companies such as Google, Facebook, and Amazon.
- Increasingly used in big data and real-time web applications
- Motivations include:
  - simplicity of design
  - simpler horizontal scaling to clusters of machines
  - finer control over availability.
- Data structures: key-value, wide column, graph, document

Document-Oriented Databases

- Document Stores
- Documents encapsulate and encode data (or information) in some standard formats or encodings:
  - XML
  - JSON
  - Etc.
- Documents contain both the data and the metadata (think of an XML document!)
- One of the most popular document-oriented databases is nowadays MongoDB

MongoDB

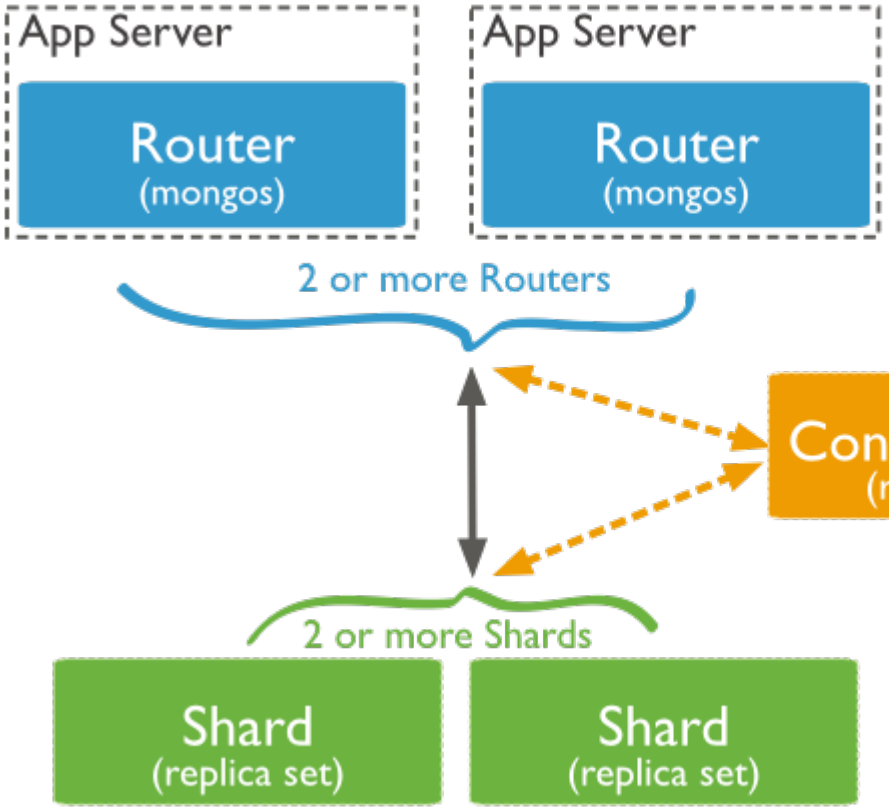
- MongoDB is a scalable, high-performance, open-source, schema-free, document-oriented DB based on BSON (= binary JSON)
- Documents are organized in collections (~ DB tables)
- Performance: no joins, no complex transactions
- Scalability: sharding (= horizontal data distribution)
- Rich JavaScript-based query syntax
  - Allows deep nested queries:
    db.order.find({ shipping : { carrier : “DHL” } })
  - Documents are just JSON objects that Mongo stores in binary
  - Queries return a cursor (= a result set iterator): more efficient
  - Cursor methods: hasNext(), forEach()

MongoDB's Features

- Capped collections: fixed-sized, ldt-operation, auto-LRU age-out
  - Fixed insertion order
  - Super fast, ideal for logging and caching
- Sharding: a method for storing data across multiple machines
  - Horizontal vs. Vertical Scaling:
    - Vertical: add more CPUs and RAM/disks
    - Horizontal: divide the dataset and distribute over multiple servers (shards).
  - Each shard is an independent DB
  - Collectively, the shards make up a single logical DB
  - MongoDB supports sharding through the configuration of a sharded cluster

MongoDB Sharded Cluster

Query routers



Cluster's metadata

Shard keys are used to assign documents to shards

Range-based partitioning vs. hash-based partitioning

Apache Cassandra

- An open-source distributed DB management system
  - handles large amounts of data across many servers
  - provides high availability with no single point of failure
  - asynchronous masterless replication
- Initially developed at Facebook by Avinash Lakshman
- Based on DHTs
- Borrows many elements from Amazon's Dynamo
- Hybrid data model: key-value, column-oriented, row-partitioned
- Cassanda Query Language (CQL)
  - Syntax similar to SQL
  - Alternative to RPC-based interface

Graph-Oriented Databases

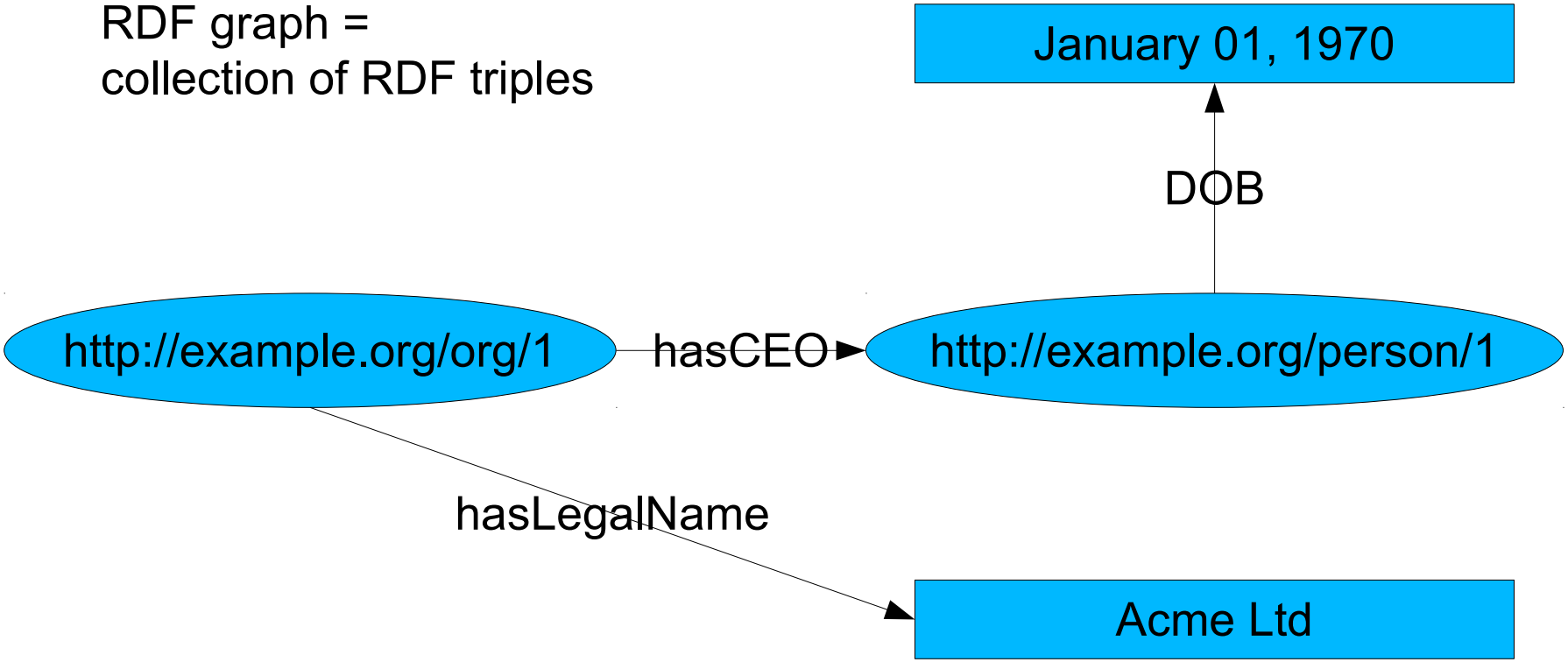
- A graph database is database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data
- Most graph databases store their data in a key-value store or document-oriented database
- The two most popular graph databases are OpenLink Virtuoso and AllegroGraph
- One particularly influential graph model used by these databases is RDF (the resource description framework)

RDF

- The data model of the Semantic Web
  - Published as a W3C recommendation in 1999
  - Initially a model for metadata
  - Now used to represents all sorts of data and knowledge
- Resource Description Framework
  - Resource: anything that can have an URI, a node of the graph
  - Decription: attributes, features, and relations of the resources
  - Framework: model, languages and their syntax and semantics
- Based on triples of the form (subj pred obj)
  - Subj and obj are resources (= nodes); obj can be a literal
  - Pred is an arc label

RDF Data Model

RDF graph =
collection of RDF triples



RDF Syntax

- RDF/XML

```
<rdf:RDF xmlns:voc="http://example.org/vocabulary">
  <voc:RegisteredOrganization rdf:about="http://example.org/org/1">
    <voc:hasLegalName> "Acme Ltd" </voc:hasLegalName>
    <voc:hasCEO rdf:resource="http://example.org.person/1"/>
  </voc:RegisteredOrganization>
  <voc:Person rdf:about="http://example.org.person/1">
    <voc:DOB>January 1, 1970</voc:DOB>
  </voc:Person>
</rdf:RDF>
```

- Turtle

```
@prefix voc: <http://example.org/vocabulary/>
<http://example.org/org/1> rdf:type voc:RegisterdOrganization ;
  voc:hasLegalName "Acme Ltd" ;
  voc:hasCEO <http://example.org.person/1> .
<http://example.org.person/1> rdf:type voc:Person ;
  voc:DOB "January 1, 1970" .
```

SPARQL

- SPARQL = **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
- SPARQL is the query language for RDF
  - Based on the RDF data model (triples/graph)
  - Main idea: pattern matching
  - Declarative: describe subgraphs of the queried RDF graph
  - Graph patterns (= RDF graphs with variables
- SPARQL is a Protocol
  - Transmission of SPARQL queries and results
  - SPARQL Endpoint: a Web service implementing the protocol
- W3C standard since January 2008

SPARQL Query Types

- SELECT ?x ?y … WHERE *graph-pattern*(?x, ?y, …)
  - Return a table of all *x*, *y, …* matching the description of the graph contained in the graph pattern.
- CONSTRUCT {?s ?p ?o} WHERE *graph-pattern*(?s, ?p, ?o)
  - Find all *x*, *y, …* satisfying the given conditions and replace them in the given triple templates to create a new RDF graph from them
- DESCRIBE ?x WHERE *graph-pattern*(?x)
  - Find all declarations providing information about the given resource(s) matching the given graph conditions
- ASK WHERE *graph-pattern*(?x, ?y, …)
  - Check whether there exist *x*, *y, …* matching the description

SPARQL: Example

- SELECT ?companyName
  WHERE {
    ?company voc:hasLegalName ?companyName .
    ?company voc:hasCEO ?ceo .
    ?ceo voc:DOB “January 1, 1970” .
  }

- Returns:

  companyName
  …
  “ACME Ltd”
  …

Semantic Web and Linked Data

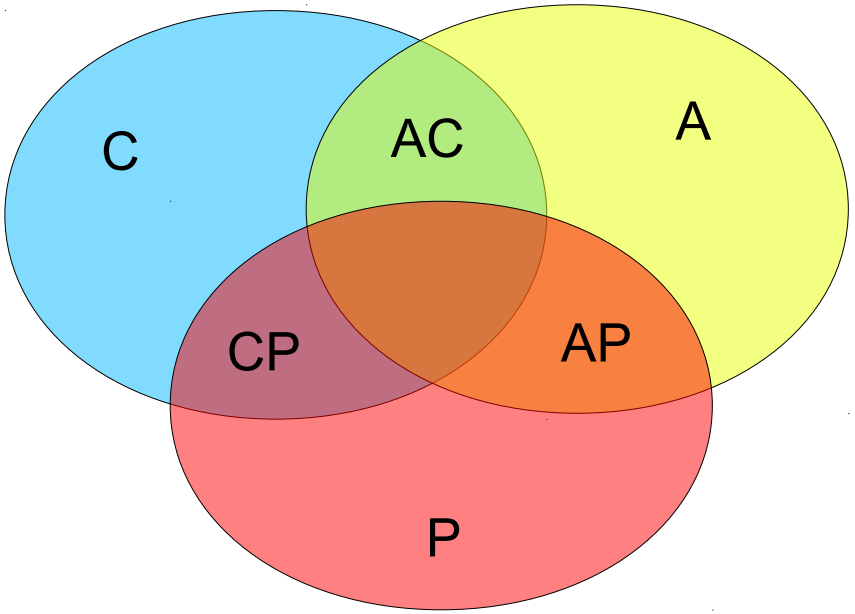
- To learn more about RDF, SPARQL, and other Semantic Web related technologies...
- If you can understand French:
  - MOOC “Web sémantique et Web de données”
  - www.fun-mooc.fr/courses/inria/41002S02/session02/about
- Otherwise:
  - MOOC “Knowledge Engineering with Semantic Web Technologies”, 2015 edition
  - https://open.hpi.de/courses/semanticweb2015

The CAP Theorem

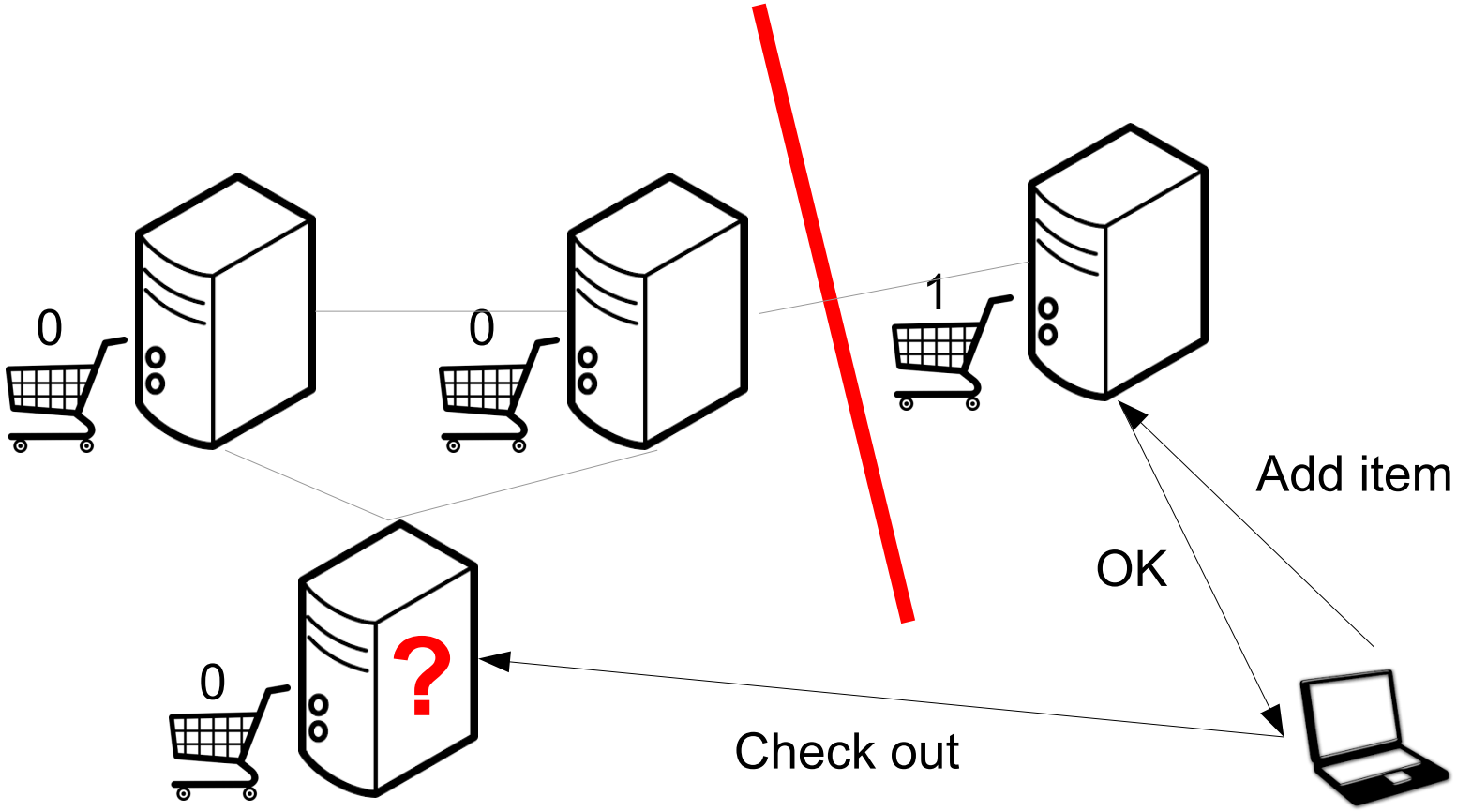
- Probably the must cited distributed systems theorem these days
- Stated by Eric Brewer, 2000, proved by Gilbert and Lynch, 2002
- Relates the following 3 properties
  - C: Consistency
    - Cf. Part b of this lecture
  - A: Availability
    - Every client's request is served (receives a response) unless a client fails (despite a strict subset of server nodes failing)
  - P: Partition-tolerance
    - System functions properly even if the network is allowed to lose arbitrarily many messages

CAP Theorem

C, A, P: pick two!



CAP Theorem: an Illustration



The CAP Theorem in Practice

- In practical distributed systems
  - Partitions may occur
  - This is not under your control (as a system designer)
- Designer's choice
  - You choose whether you want your system in C or A when/if (temporary) partitions occur
  - Note: You may choose neither of C or A, but this is not a very smart option
- Bottom line
  - Practical distributed systems are either in CP (e.g.: MongoDB, RDBMs) or in AP (e.g.: Cassandra)

Thank you for your attention!