Introduction to Database Systems: CS312
The Theory of NoSQL

Oliver Bonham-Carter

25 March 2019

SQL: One Type of Database

Building from our last class ...

- SQL (Structured query language) systems are designed for managing data in relational database management systems (RDBMS)
- SQL has been the standard querying language since the 1980's
- Schemas and Tables
  - Need to know how to store the data from the initial stages of database design
  - Integrity constraints: data types must be understood before populating
  - *Tight* design of data organization ...
- SQLite3: Single read or write at a time:
  - Only one user may use a database at any time

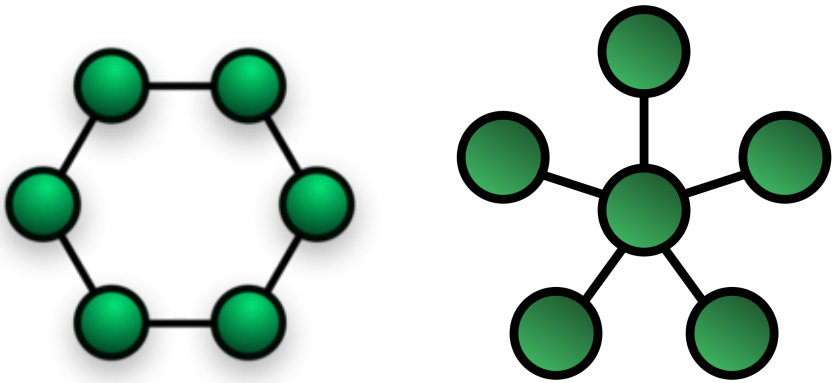
Can One Type of Database Meet Our Needs?

- What if you are working with *Big Data*?
  - Lots of simultaneous reading and writing
  - Different devices in different locations

NoSQL: Another Type of Database

- Different types of NoSQL databases

Token Ring Networks



Wikipedia Says...

A token ring network is a local area network (LAN) in which all computers are connected in a ring (left) or star (right) topology and pass one or more logical tokens from host to host. Only a host that holds a token can send data, and tokens are released when receipt of the data is confirmed.

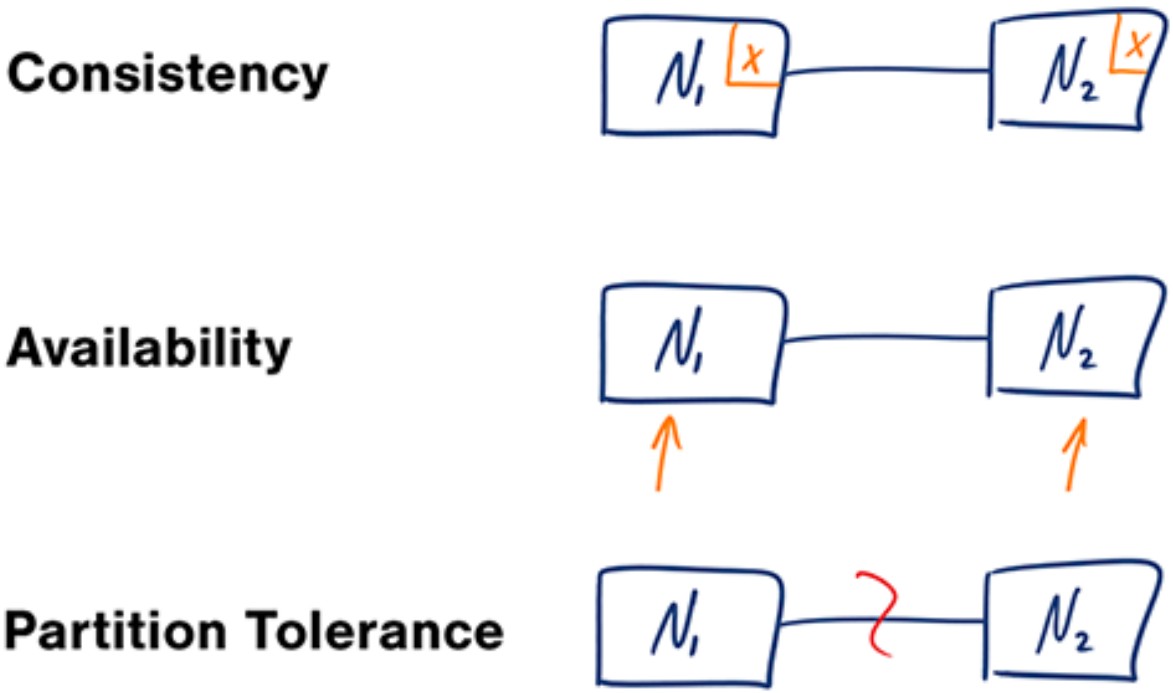
An architecture for NoSQL database connectivity

Token Ring Networks

- Early networks established foundational database configurations for wide-spread usage
- All devices on the ring share data and update each other
- Hashing function maps each key to a server (node)

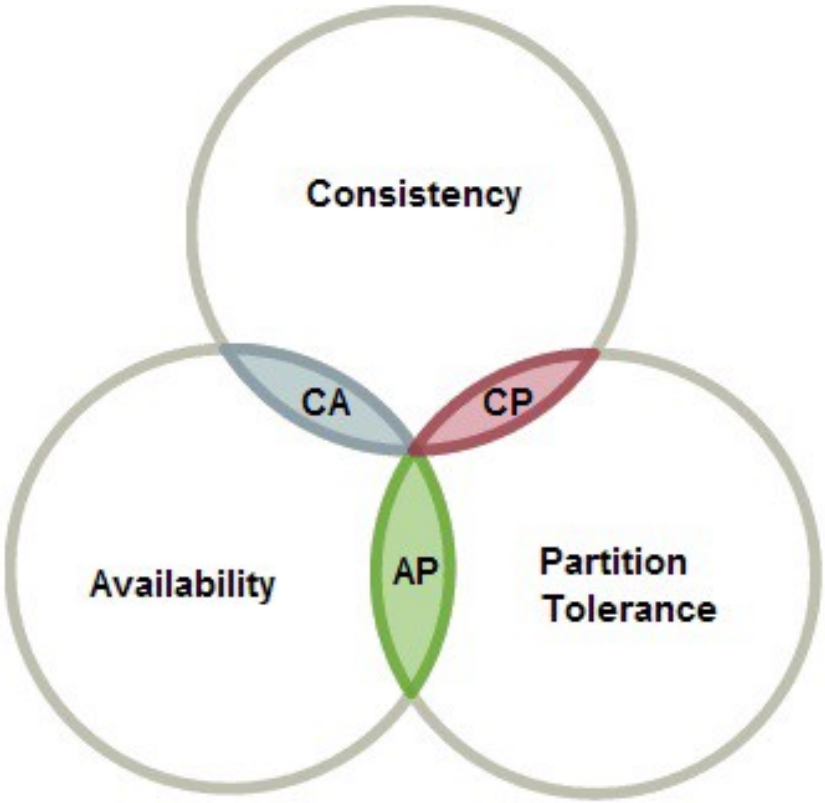
Good Read on CAP Fundamentals

CAP Theorem: Revisited, by Robert Greiner



http://robertgreiner.com/2014/08/cap-theorem-revisited/

Cap Theorem, Formally Stated



Theoretical computer science: the CAP theorem (i.e., Brewer's Theorem), states that it is impossible for a distributed data storage system to simultaneously provide more than two out of the following three guarantees.

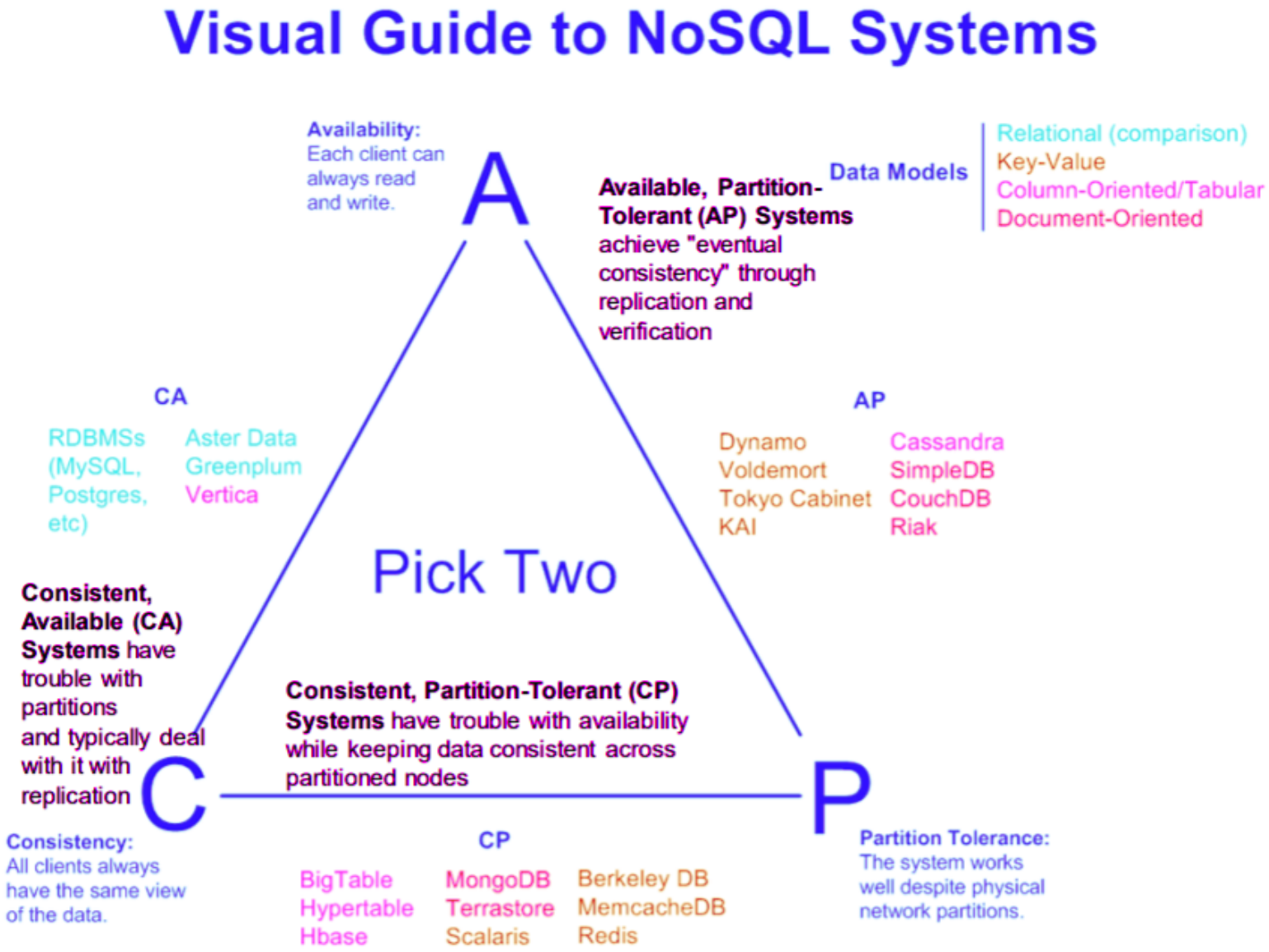
CAP Theorem

Guarantees

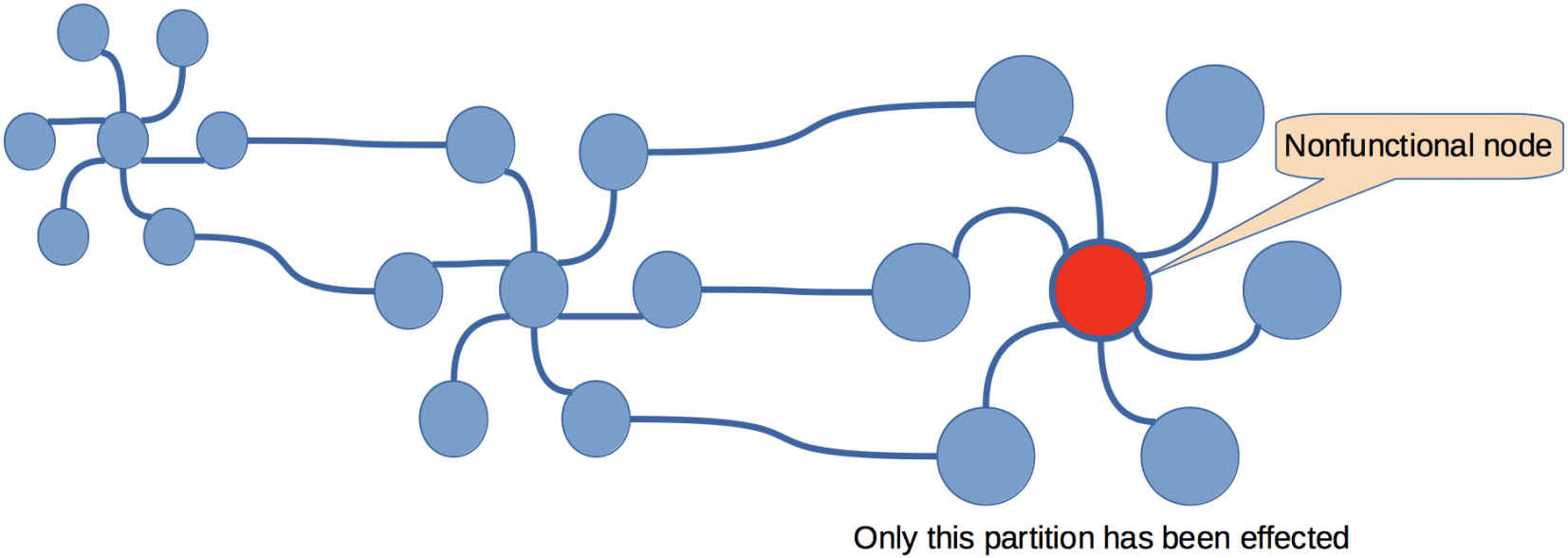
It is not impossible to have more than two out of the three

- **Consistency**:
  - Every `read` receives the most recent `write` or an error
  - All nodes see same data at same time
- **Availability (non-failure)**:
  - Every request receives a (non-error) response – without guarantee of the most recently written data
  - Node failures do not prevent surviving nodes from continuing to operate
- **Partition tolerance (network disconnections)**:
  - The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
  - The system continues to operate despite network partitions

Pick Only Two

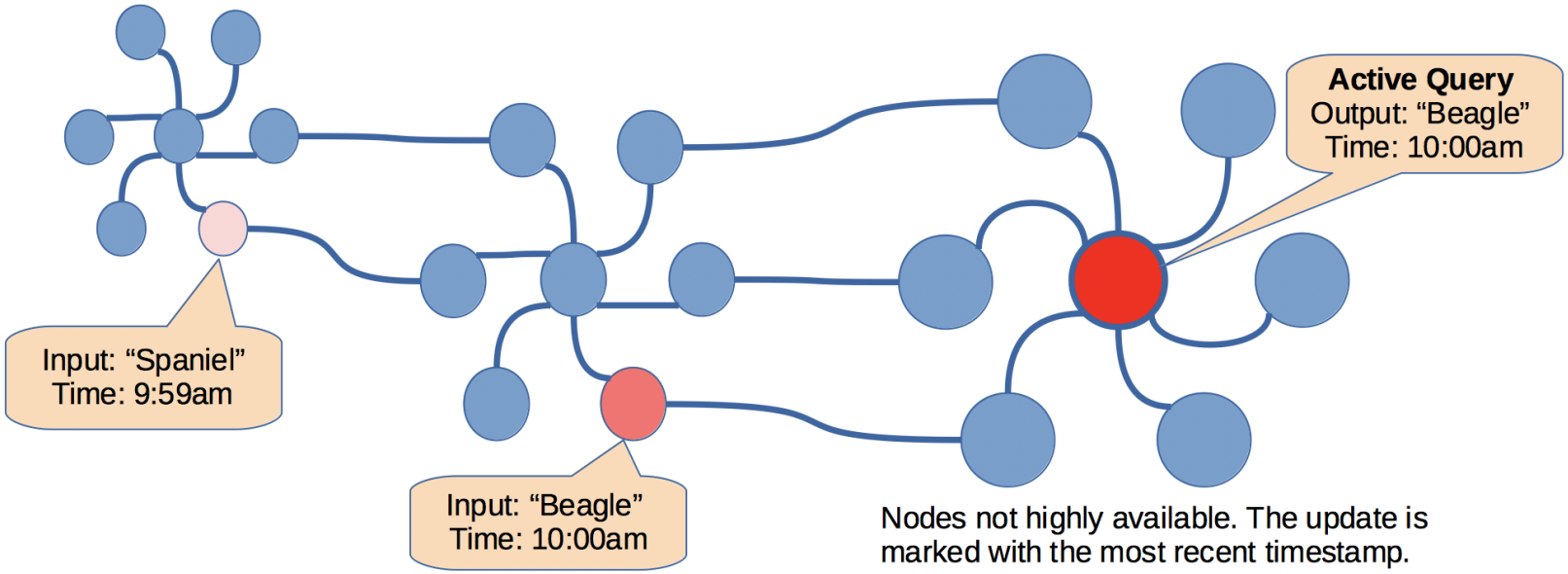


Partition Tolerance



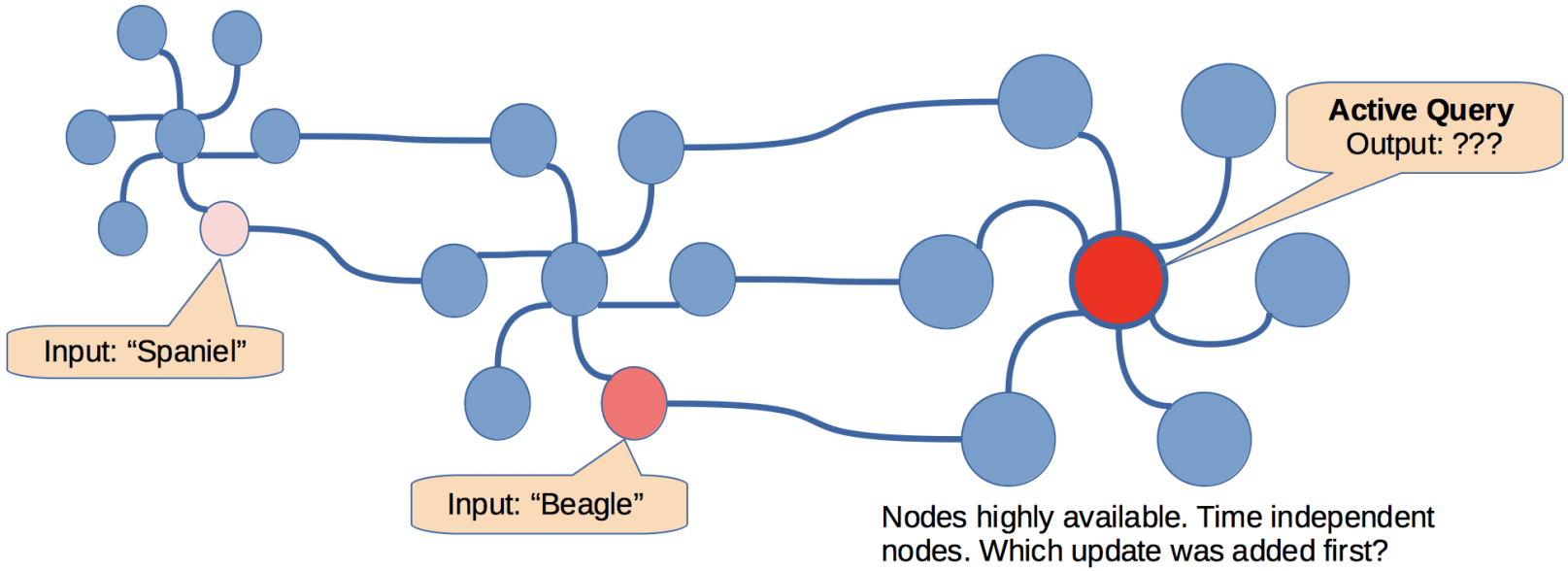
- The system continues to run, regardless of the number of messages being delayed by the network between nodes.
- Partition-tolerant networks sustain network failures with no result of a failure of the entire network

High Consistency



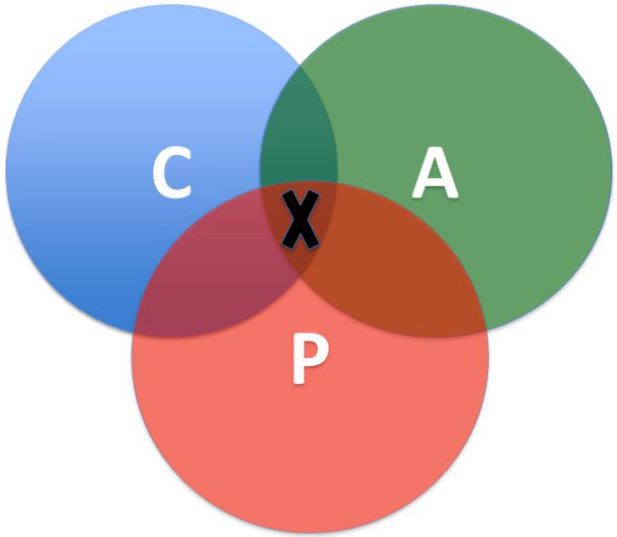
- A `read` operation returns the value of the most recent `write` operation causing all nodes to return the same data
- Time to update: A system can be in an inconsistent state during a transaction
- The entire transaction is reverted during errors in the update processes

High Availability



- Always operational: Every request always gets a response on success/failure
- Databases are time-independent since the nodes have to always be available online (at all times)

Conclusions I



- DB Systems are higher in performance, lower latency, and near 100 percent up-time in data centers all over the world.
- Complexity makes it necessary to compromise

Conclusions II

The CAP Theorem

- Networks may fail and (1) if you cannot limit the number of faults, and (2) the requests can be directed to any server, and (3) you insist on serving *every* request you receive, then you cannot possibly be consistent

Understanding this theorem:

- You must always give something up: consistency, availability or tolerance to failure and reconfiguration
- We get to choose what to do when a partition (network failure) occurs. According to the CAP theorem, we have two options:
  - Consistency and Availability.

Two Choices

Choosing one over the other ...

- **Consistency over availability**: the system will return an error or a time-out if particular information cannot be guaranteed to be current due network partitioning.
- **Availability over consistency**: the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network partitioning.

To summarize ...

- In the *absence of network failure* (i.e., the distributed system is running normally and as expected) **both** availability and consistency can be satisfied

Getting started with Mongo

Setup a data directory (if you have not already done so)

```
mkdir ~/mongodbData
```

Start the Mongo server with data directory as a parameter
Note: Control-C to exit.

```
mongod --dbpath ~/mongodbData/
```

With new another terminal, start the Mongo client

```
mongo
```

Find databases or collections, from Mongo's client

```
show dbs
show collections
```

Begin a new database, from Mongo's client

```
use myDB
```

Getting started with Mongo

Enter data into your first collection (i.e., a *table*)

```
db.people.insert({name:'James Bond', shoes:'brown'})
db.people.insert({name:'S. Holmes', shoes:'black'})
db.people.insert({name:'Wonder Woman', shoes:'boots'})
db.people.insert({name:'Batman', shoes:'black'})
db.people.insert({name:'Flash', shoes:'slippers'})
```

A general query of the collection (people)

```
db.people.find()
db.people.find().pretty()
```

- Where is the schema!?

Getting started with Mongo

A specific query the collection (people)

```
db.people.find({shoes:'brown'})
db.people.find({shoes:'black'})
db.people.find({},{"name":1,"shoes":'brown',"_id":0})
db.people.find({},{"name":1,"shoes":'black',"_id":0})
db.people.find({},{"name":1,"shoes":'boots',"_id":0})

db.people.find({shoes:'black'}).pretty()
db.people.find({shoes:'boots'}).pretty()

db.people.find({name:'Wonder Woman'}).pretty()
db.people.find({name:'Batman'}).pretty()
db.people.find({"name":"S. Holmes"},{"shoes":'boots',"_id":0}).pretty()
```

Drop the collection (people): Destroy the data, remove collection

```
db.people.drop()
```