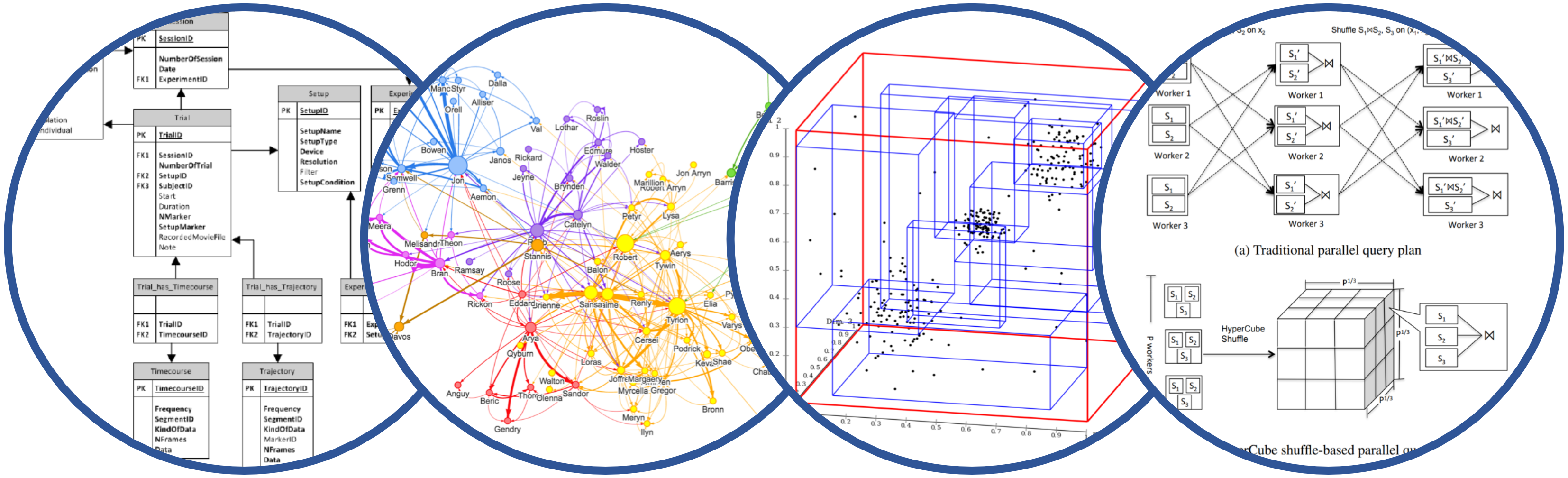


Introduction to Data Management

NoSQL

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Classical Database Application Problems

OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)

Classical Database Application Problems

OLTP (Online Transaction Processing)	**OLAP (Online Analytical Processing)**
Transaction-heavy workloads	Complex query workloads

Classical Database Application Problems

OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Transaction-heavy workloads	Complex query workloads
Many simple lookup or single-join queries	Many joins, aggregations, etc.

Classical Database Application Problems

OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Transaction-heavy workloads	Complex query workloads
Many simple lookup or single-join queries	Many joins, aggregations, etc.
Many small updates and inserts	Little to no updates

Classical Database Application Problems

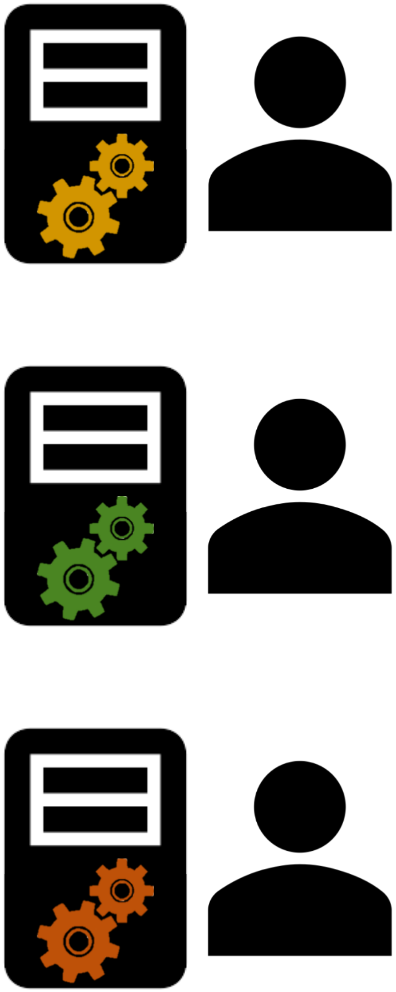
OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Transaction-heavy workloads	Complex query workloads
Many simple lookup or single-join queries	Many joins, aggregations, etc.
Many small updates and inserts	Little to no updates
Managing consistency is critical	Query optimization and processing is critical

Classical Database Application Problems

OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Transaction-heavy workloads	Complex query workloads
Many simple lookup or single-join queries	Many joins, aggregations, etc.
Many small updates and inserts	Little to no updates
Managing consistency is critical	Query optimization and processing is critical
Flights, banking, etc. (many users)	Business intelligence (few users)

Client-Server Applications

Client-Server Applications



Client-Server Applications

Single server runs the entire database

Could be:

- Your own computer
- Cloud-hosted DB

Client-Server Applications

Multiple client applications connect to DB server

Single server runs the entire database

Could be:

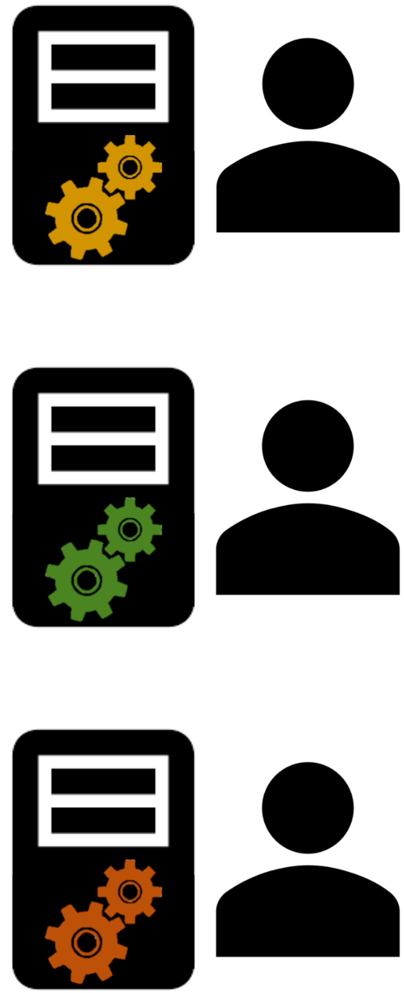
- Your own computer
- Cloud-hosted DB

Client-Server Applications

Multiple client applications connect to DB server

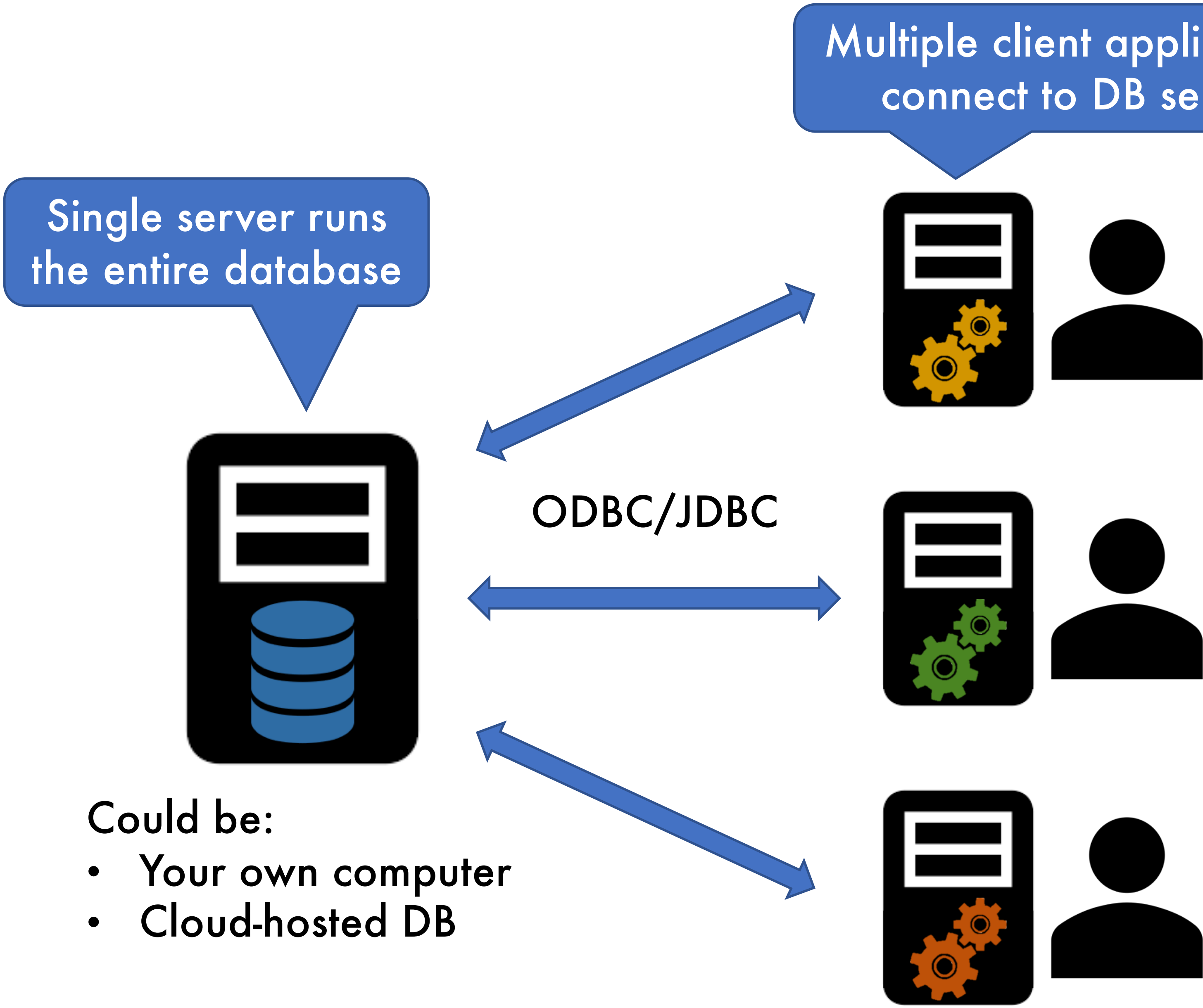
Single server runs the entire database

Could be:
- Your own computer
- Cloud-hosted DB



Could be:
- Query editor
- Java app (lab)
- Analyst app (Tableau)

Client-Server Applications



Could be:
- Query editor
- Java app (lab)
- Analyst app (Tableau)

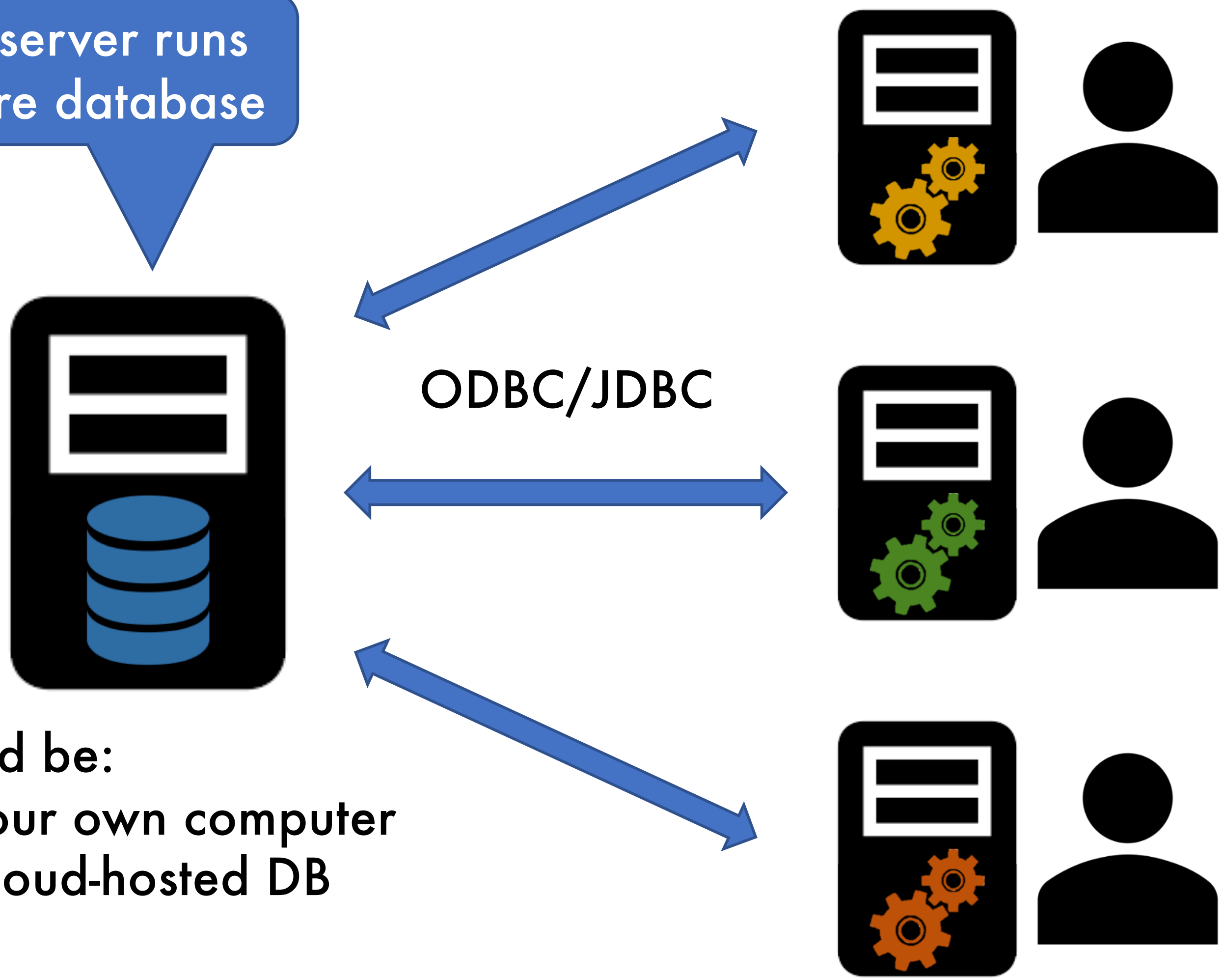
Could be:
- Your own computer
- Cloud-hosted DB

Client-Server Applications

Sufficient for OLAP (simple)
Can't scale connections for OLTP

Single server runs the entire database

Multiple client applications connect to DB server



Could be:
- Your own computer
- Cloud-hosted DB

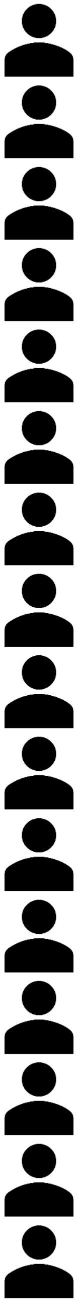
Could be:
- Query editor
- Java app (lab)
- Analyst app (Tableau)

The World Wide Web – Web 2.0

- A new class of problem emerges in the late 90s and early 2000s (and is still a problem today)
- What is Web 2.0?
  - Social web (Facebook, Amazon, Instagram, …)
  - Startup services need to **scale quickly by orders of magnitude** (shared-nothing architecture!)
  - **Exclusively OLTP workloads**

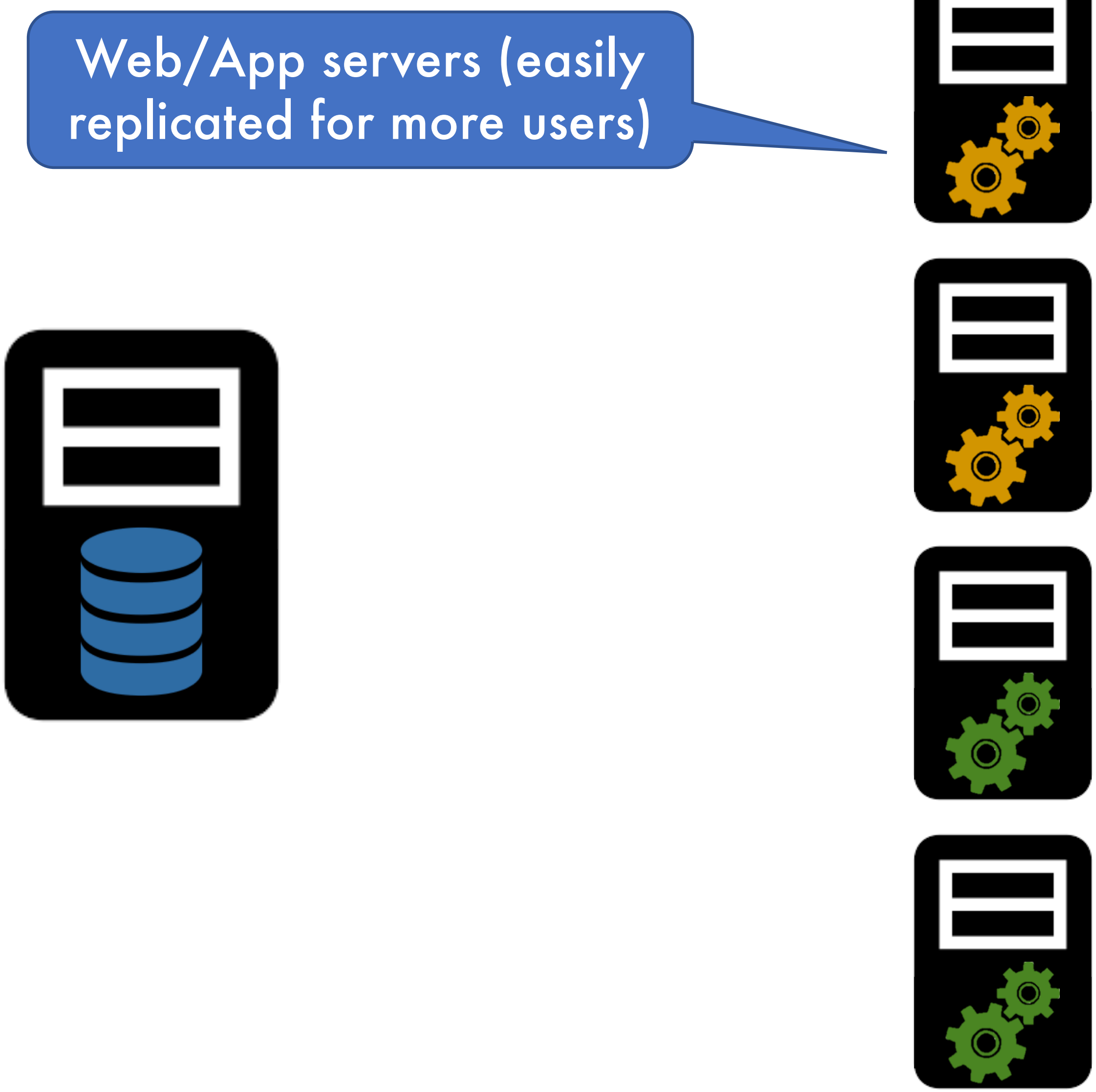
3-Tiered Web Architecture

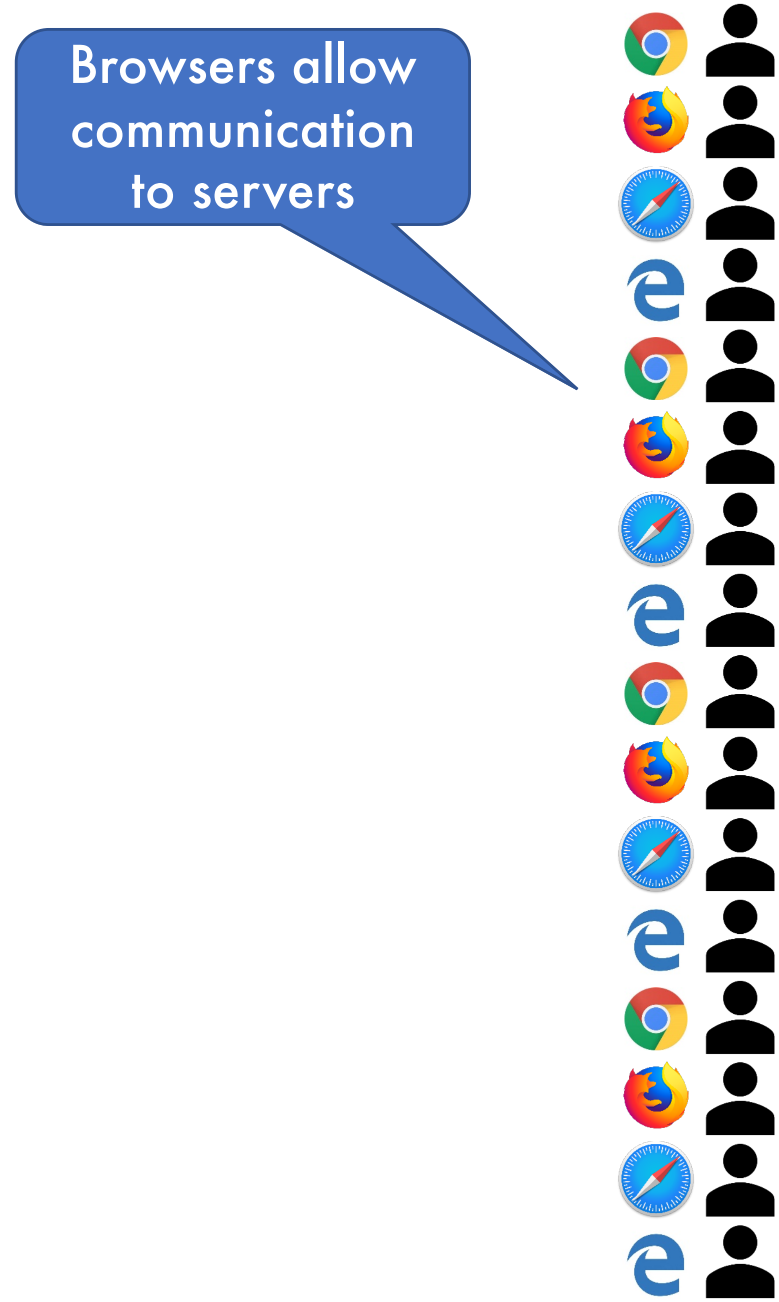
How do we architect an OLTP solution?



3-Tiered Web Architecture

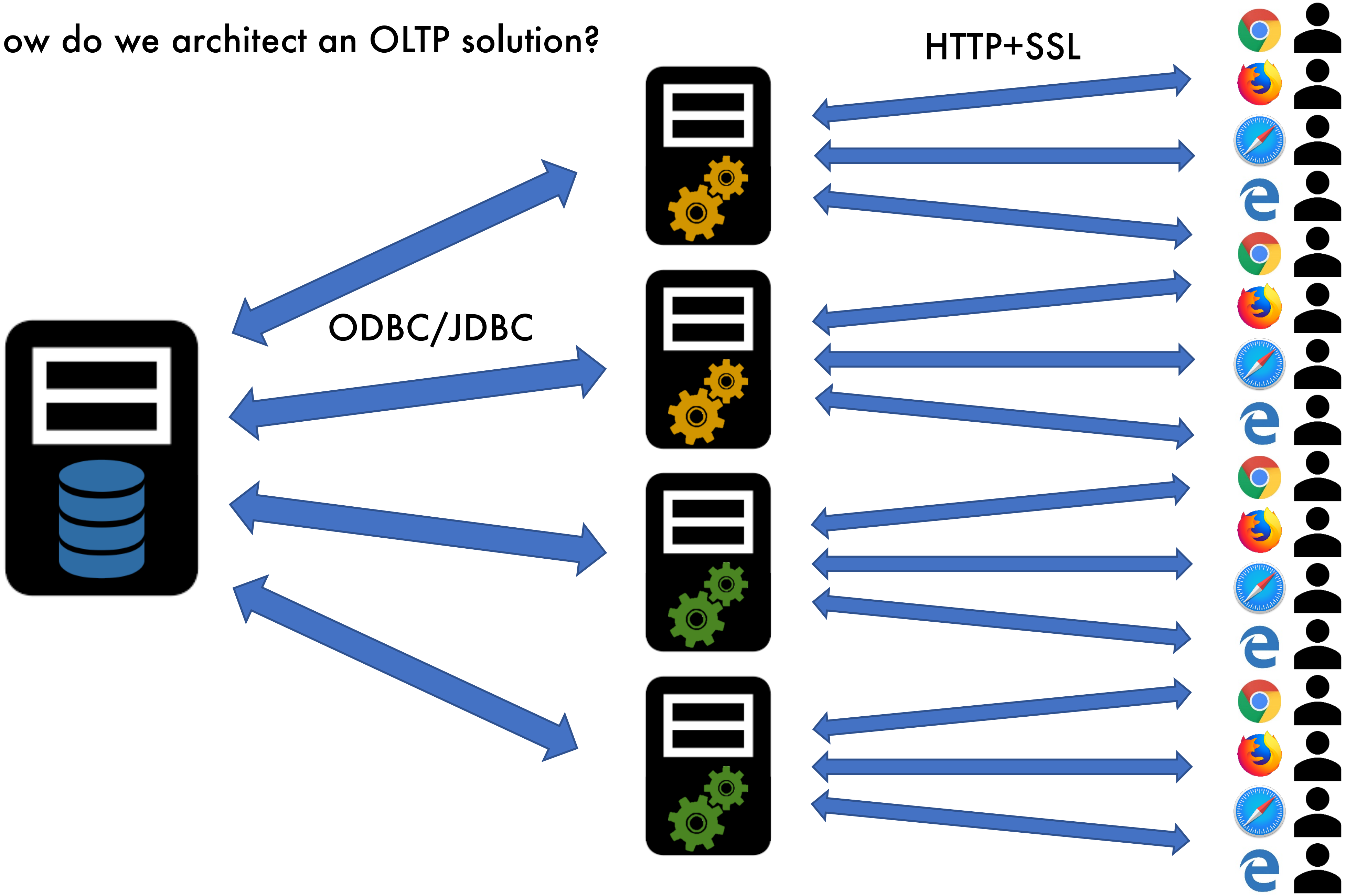
How do we architect an OLTP solution?





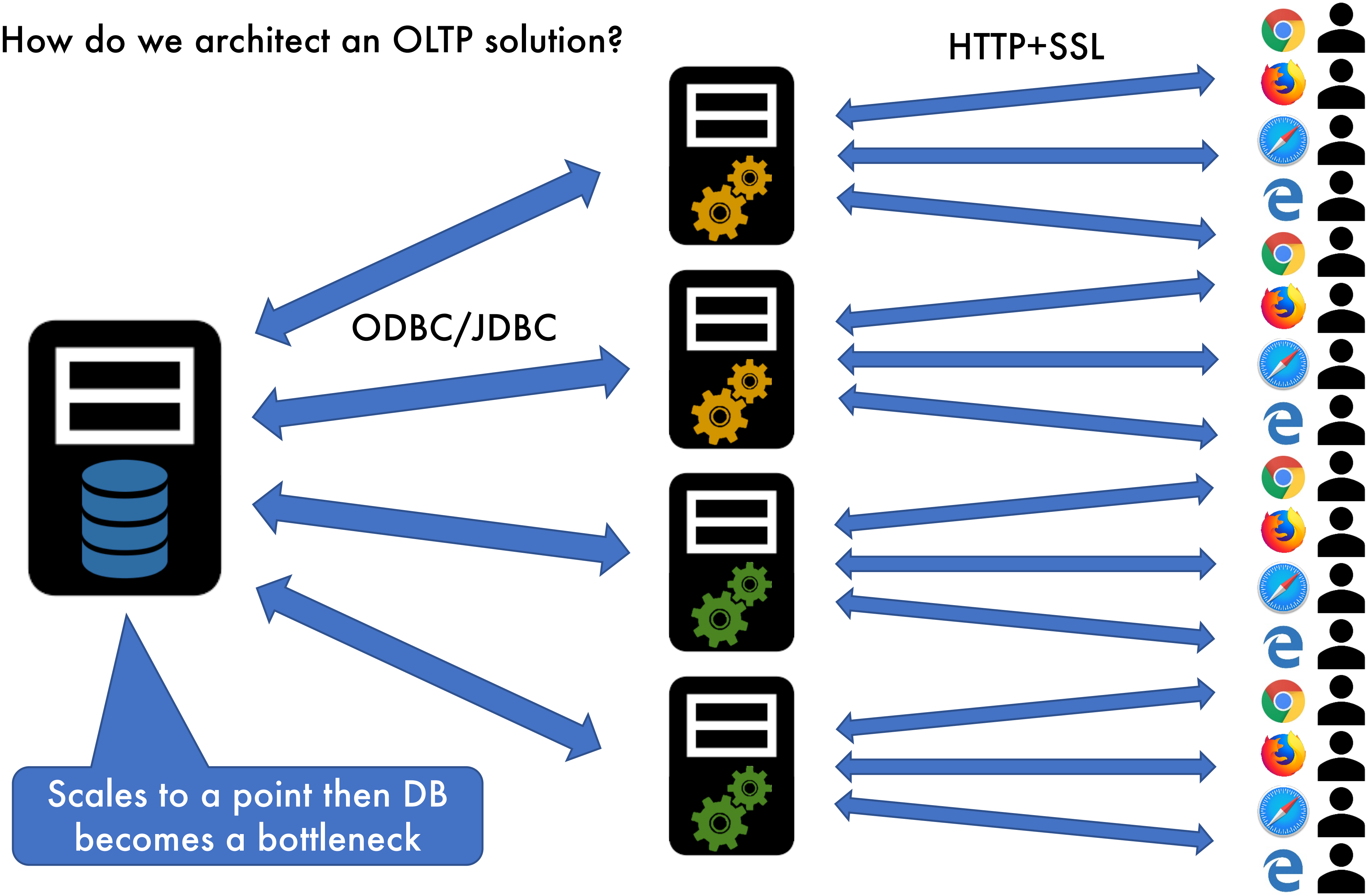
3-Tiered Web Architecture

How do we architect an OLTP solution?



3-Tiered Web Architecture

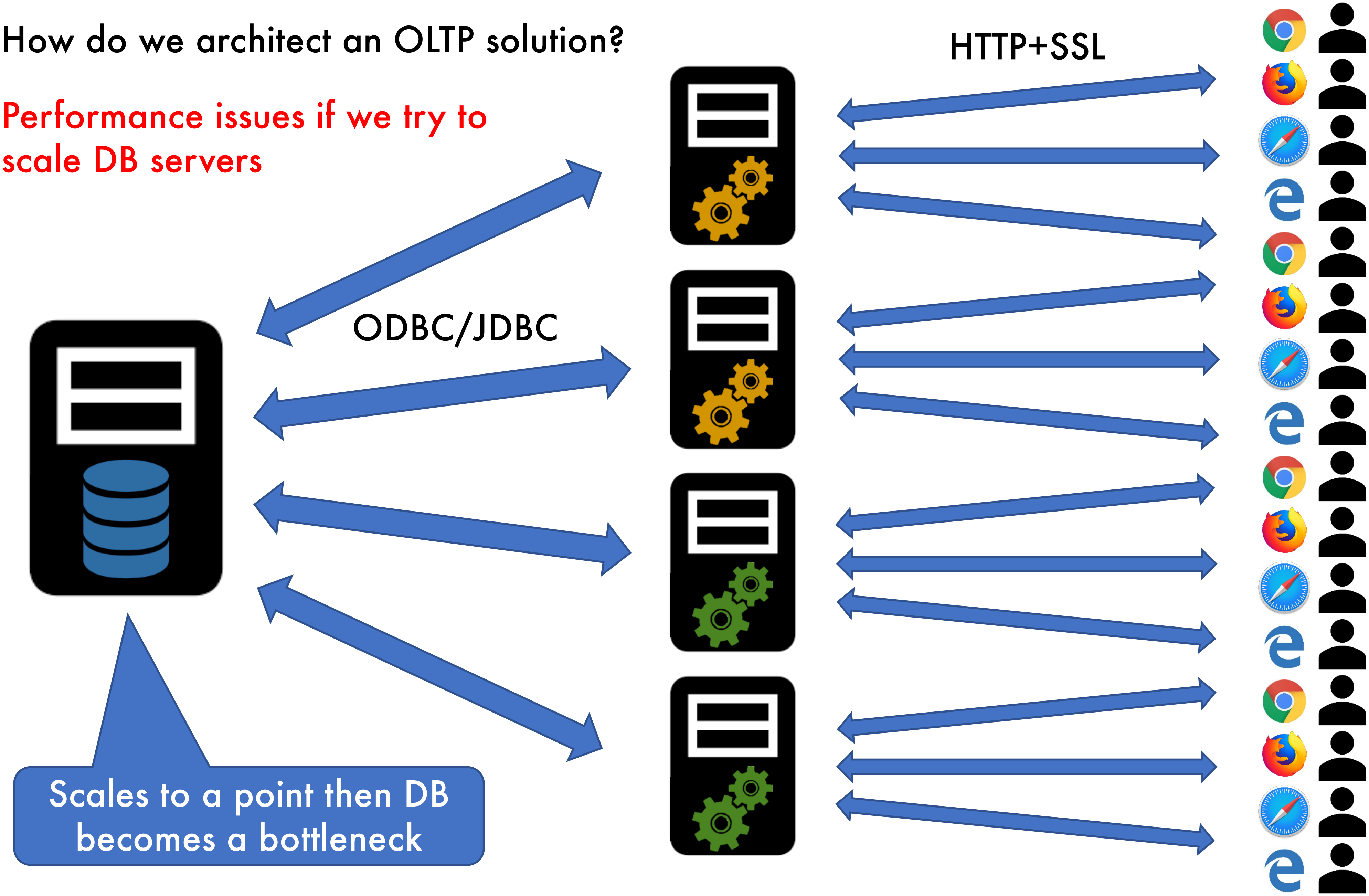
How do we architect an OLTP solution?



3-Tiered Web Architecture

How do we architect an OLTP solution?

Performance issues if we try to scale DB servers



Database Scaling Techniques

- Scale up via:
  - **Partitioning** (sharding)
  - **Replication**

RDBMS Partitioning

- Use multiple machines to distribute data
- Write performance ok
- Read performance suffers!
  - Join across servers may have huge network IO cost

RDBMS Replication

- Create multiple copies of each database partition
- Improves fault tolerance
- Read performance ok
- Write performance suffers!
  - Need to write same value to multiple servers

Distributed RDBMS Consistency Bottleneck

- RDBMS scaling makes consistency hard
  - Partitioning: Need to coordinate server actions
  - Replication: Need to prevent inconsistent versions
  - ACID is hard to maintain

#NoSQL

A hashtag on Twitter for a meetup in San Francisco to discuss systems like Google BigTable, Amazon Dynamo, CouchDB, etc.

Event Details

Introduction

This meetup is about "open source, distributed, non relational databases".
Have you run into limitations with traditional relational databases? Don't mind trading a query language for scalability? Or perhaps you just like shiny new things to try out? Either way this meetup is for you.
Join us in figuring out why these newfangled Dynamo clones and BigTables have become so popular lately. We have gathered presenters from the most interesting projects around to give us all an introduction to the field.

Preliminary schedule

09.45: Doors open
10.00: **Intro session** (Todd Lipcon, Cloudera)
10.40: **Voldemort** (Jay Kreps, Linkedin)
11.20: Short break
11.30: **Cassandra** (Avinash Lakshman, Facebook)
12.10: Free lunch (sponsored by Last.fm)
13.10: **Dynomite** (Cliff Moon, Powerset)
13.50: **HBase** (Ryan Rawson, Stumbleupon)
14.30: Short break
14.40: **Hypertable** (Doug Judd, Zvents)
15.20: **CouchDB** (Chris Anderson, couch.io)
16.00: Short break
16.10: Lightning talks
16.40: Panel discussion
17.00: Relocate to Kate O'Brien's, 579 Howard St. @ 2nd. First round sponsored by Digg

Registration

The event is free but space is limited, please register if you wish to attend.

Location

Magma room, CBS interactive
235 Second Street
San Francisco, CA 94105

thrudb @thrudb · 23 May 2009
sucks i'm not on the west coast and will not be able to attend **#nosql**

Todd Lipcon @tlipcon · 23 May 2009
working on slides for the **#nosql** meetup in June. trying to cover all of dist systems in 40 minutes is not as easy as it sounds.

Chris Anderson @jchris · 13 May 2009
It's official - I'll be relaxifying everyone with CouchDB at **#NoSQL**. Thanks @skr!

NoSQL in a Nutshell

- NoSQL works for Web 2.0 business models
  - **No OLAP anyway**
  - **Availability is more important than consistency for Web 2.0**
  - Facebook:
    - I don't care if I don't see every like in real time
    - I care if I can't send a like
  - Amazon:
    - I don't care if my cart forgot an item
    - I care if I can't put an item into my cart

Let's Drop ACID

- RDBMSs have the ACID consistency model
- NoSQL sys. have the **BASE** consistency model
- **Basically Available**
  - Most failures do not cause a complete system outage

Let’s Drop ACID

- RDBMSs have the ACID consistency model
- NoSQL sys. have the **BASE** consistency model
- **Basically Available**
  - Most failures do not cause a complete system outage
- **Soft state**
  - System is not always write-consistent

Let’s Drop ACID

- RDBMSs have the ACID consistency model
- NoSQL sys. have the **BASE** consistency model
- **Basically Available**
  - Most failures do not cause a complete system outage
- **Soft state**
  - System is not always write-consistent
- **Eventually consistent**
  - Data will eventually converge to agreed values

Why the Sacrifice?

Why can't we have both Consistency and Availability?

NoSQL in a Nutshell

- NoSQL → Looser data model
  - Give up built-in OLAP/analysis functionality
  - Give up built-in ACID consistency

CAP Theorem

- Old name: Brewer's Conjecture
- In a distributed data store, one can only provide two of the following three guarantees:
  - **Consistency**
    - Every read receives the most recent write or an error
  - **Availability**
    - Every request must respond with a non-error
  - **Partition tolerance**
    - Continued operation in presence of dropped or delayed messages

RDBMS vs NoSQL Systems

- Distributed RDBMS
  - Partition tolerance + **Consistency**
- NoSQL Systems
  - Partition tolerance + **Availability**

RDBMS vs NoSQL Systems

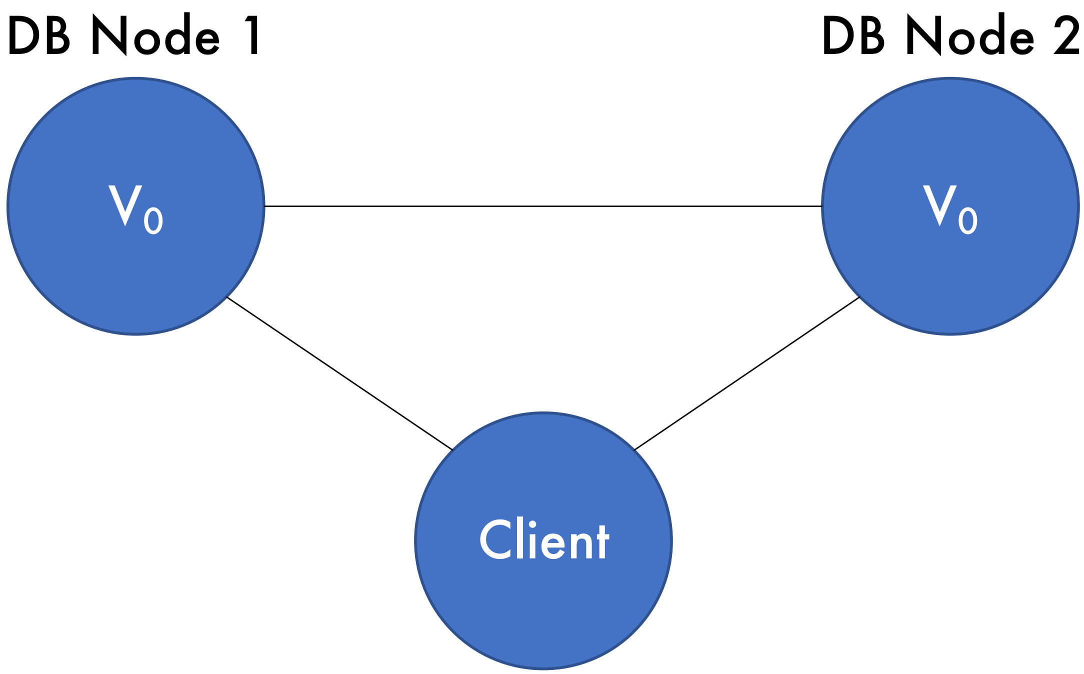
- Distributed RDBMS
  - Partition tolerance + **Consistency**
- NoSQL Systems
  - Partition tolerance + **Availability**

Both must provide partition tolerance by virtue of being distributed systems

RDBMS vs NoSQL Systems

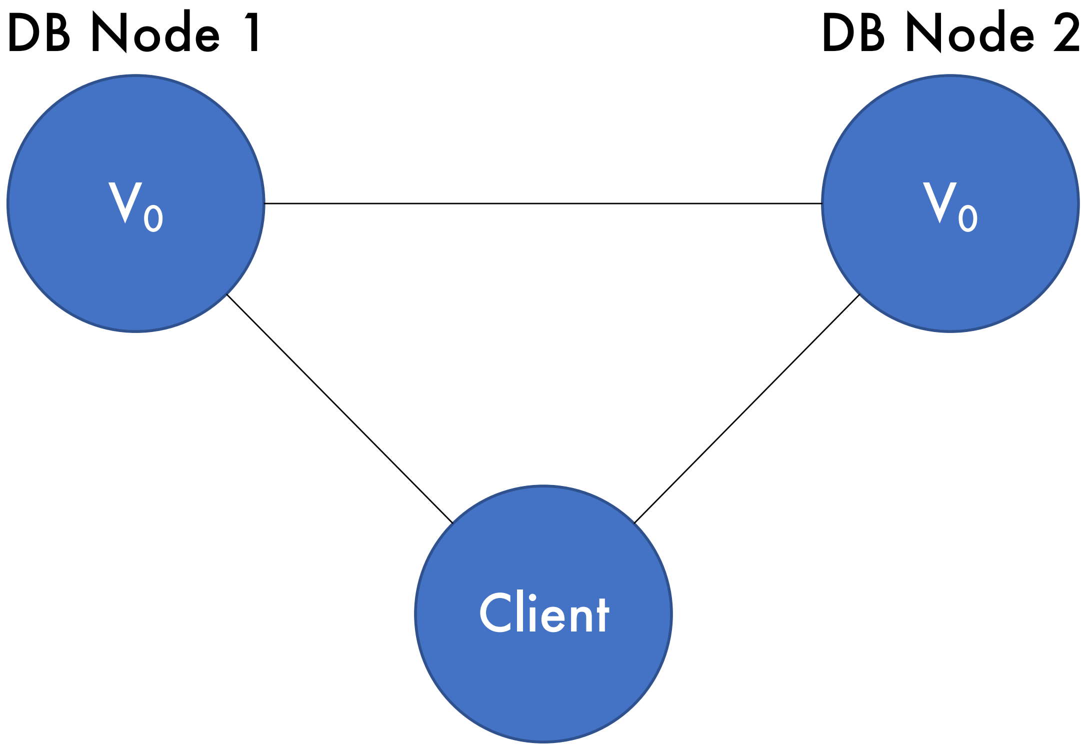
- Let's see how distributed systems act in the presence of network faults

- When we try to maintain **partition tolerance** what do we lose?



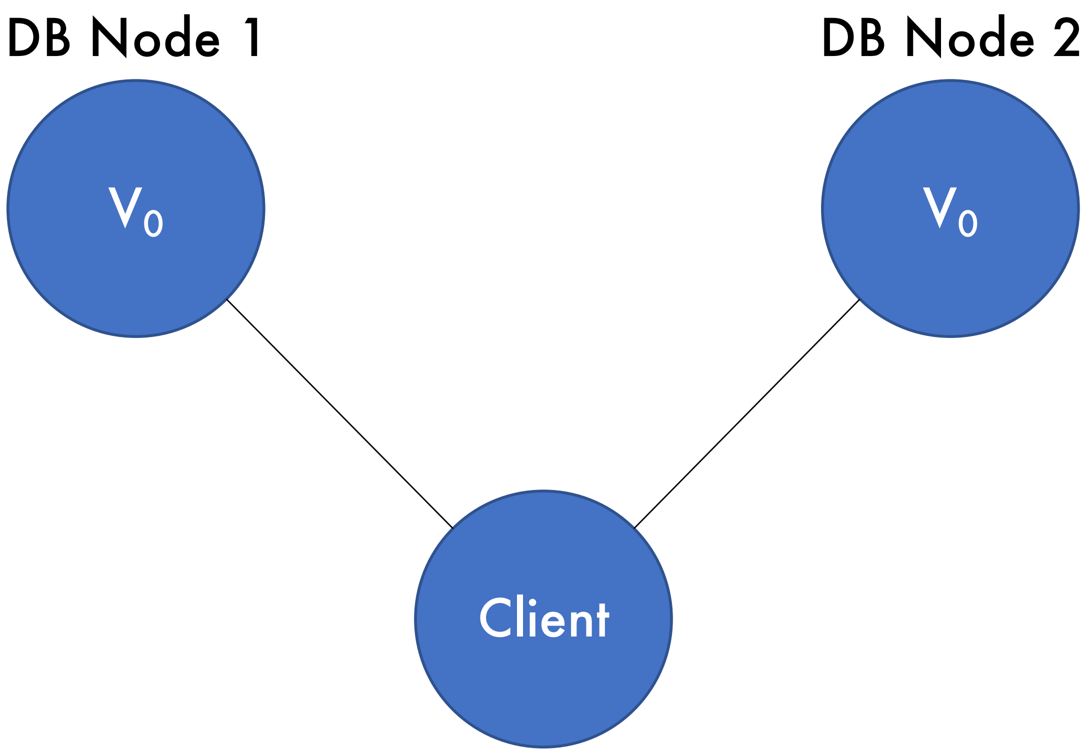
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



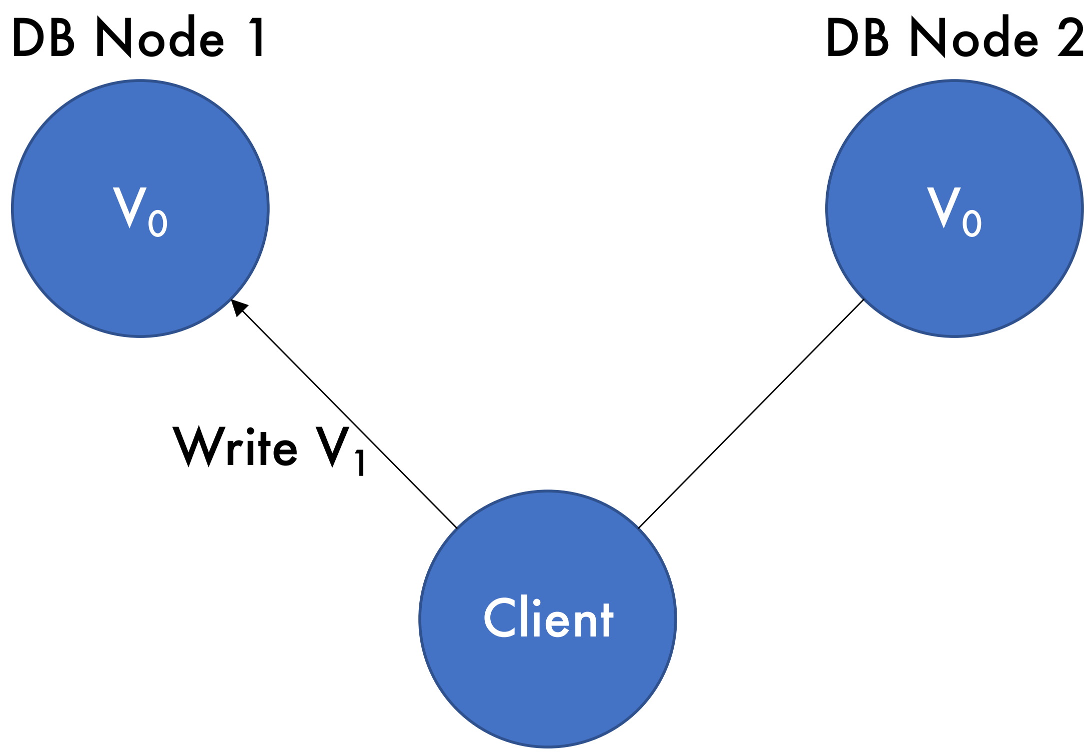
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



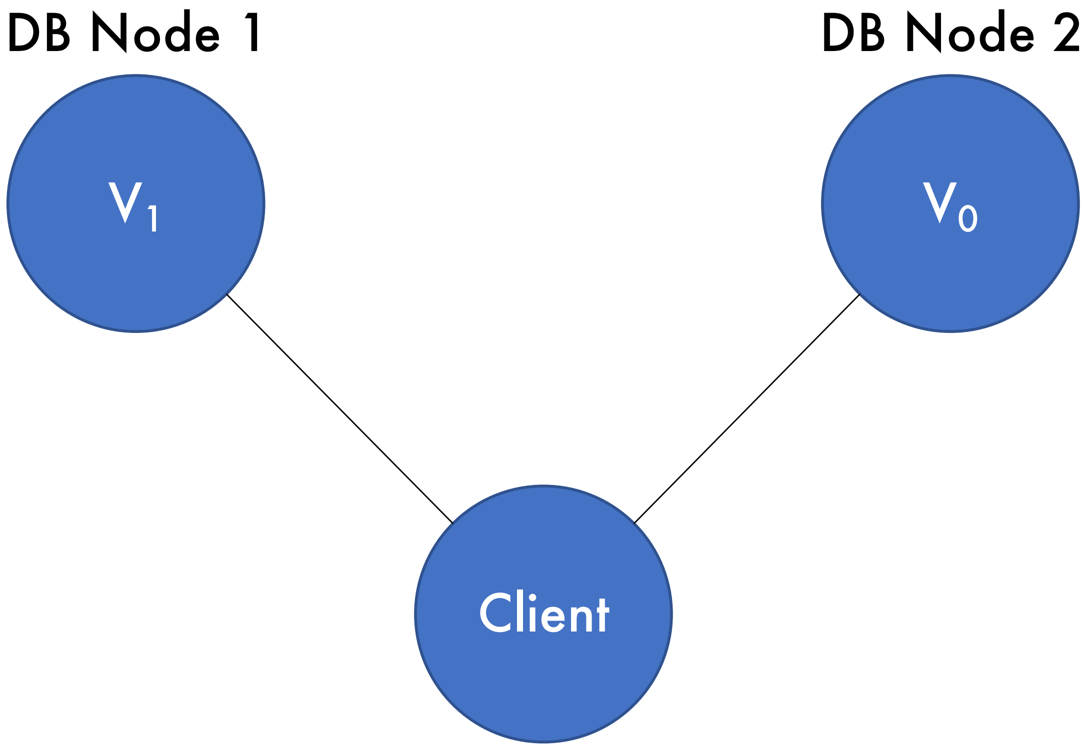
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



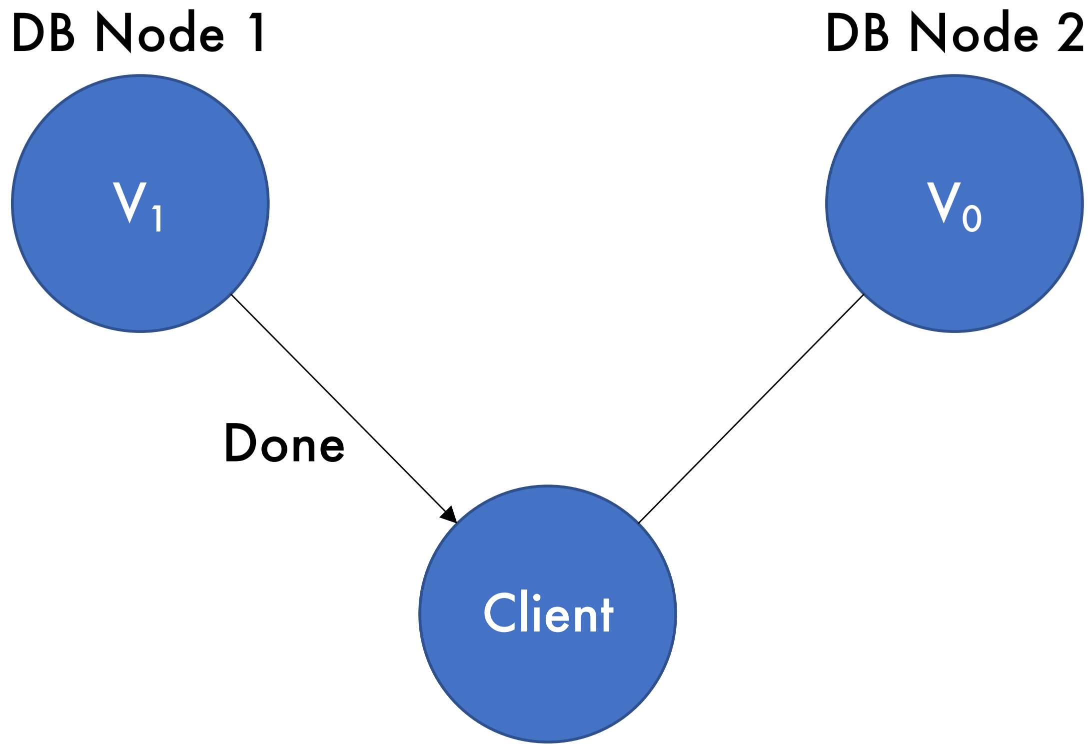
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



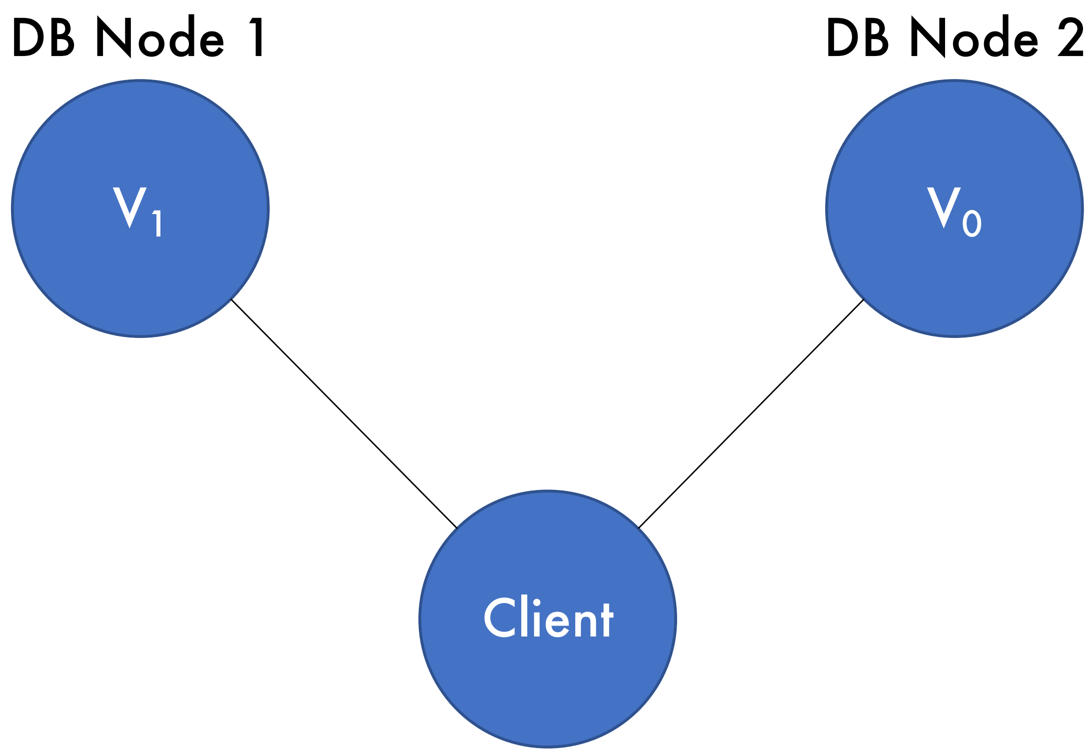
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



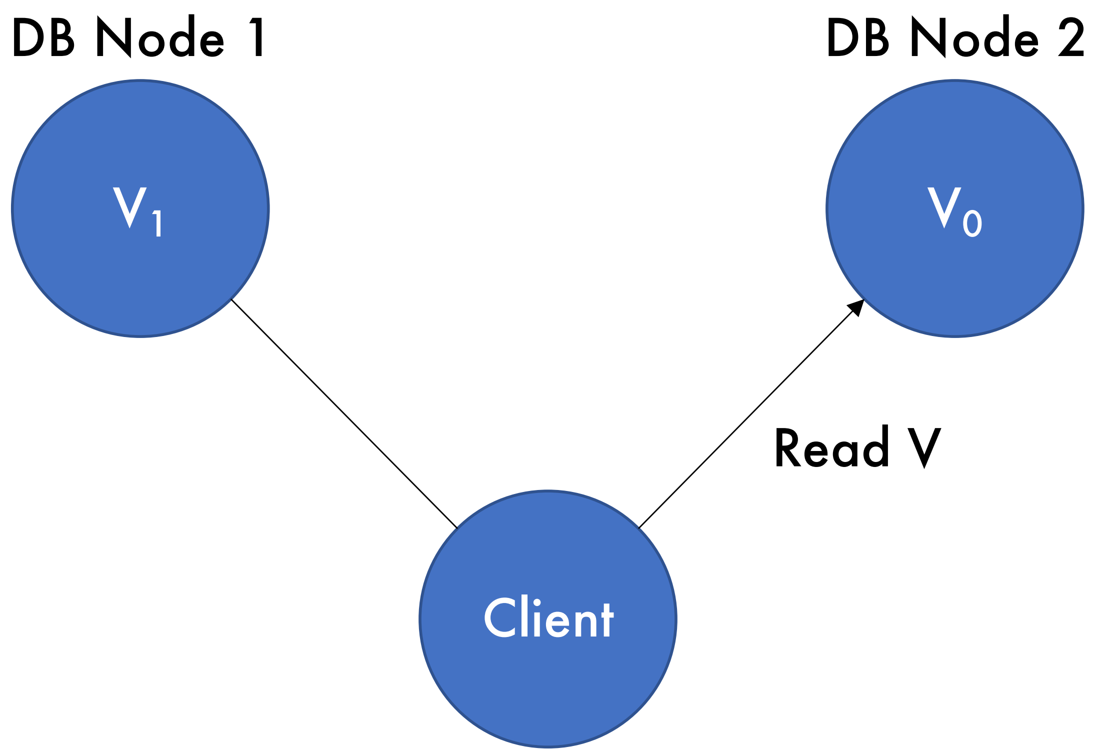
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



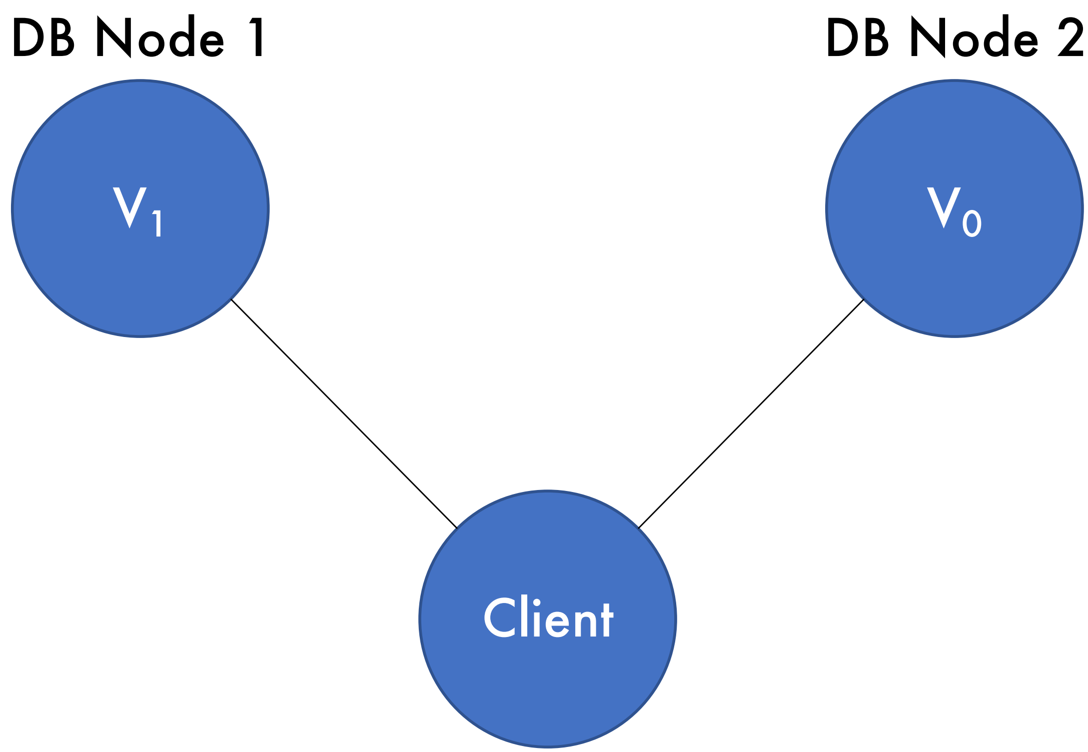
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



RDBMS vs NoSQL Systems

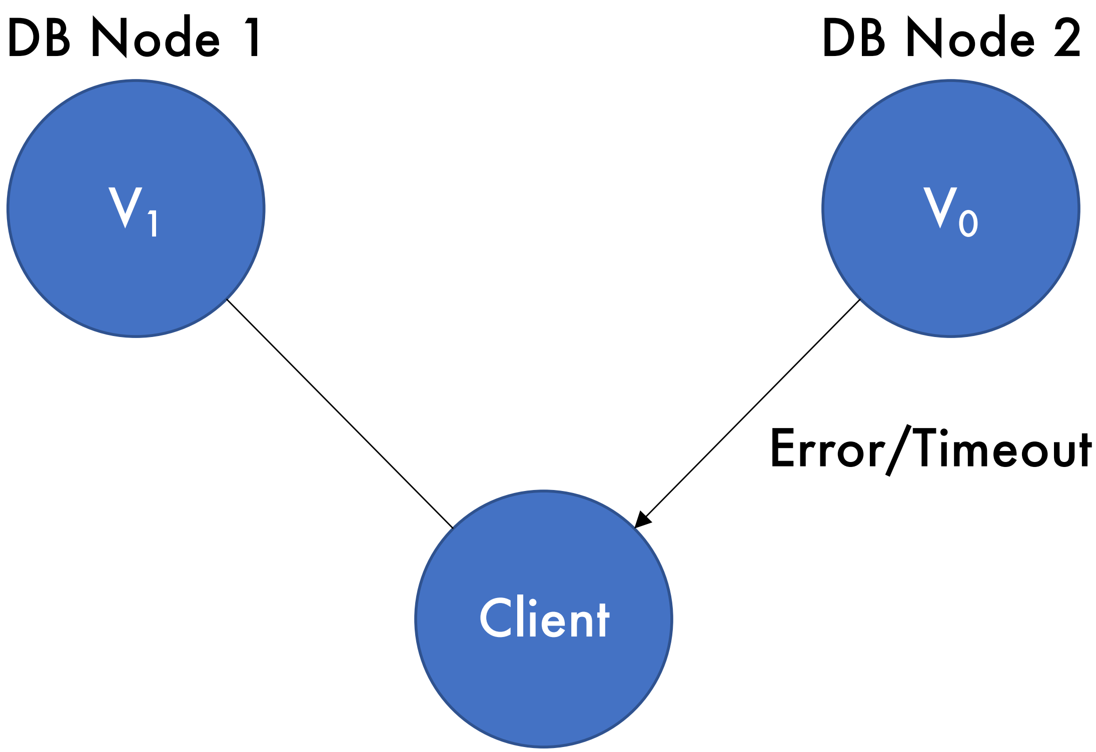
Partition tolerance + **Consistency**



Tries but fails to check consistency of V

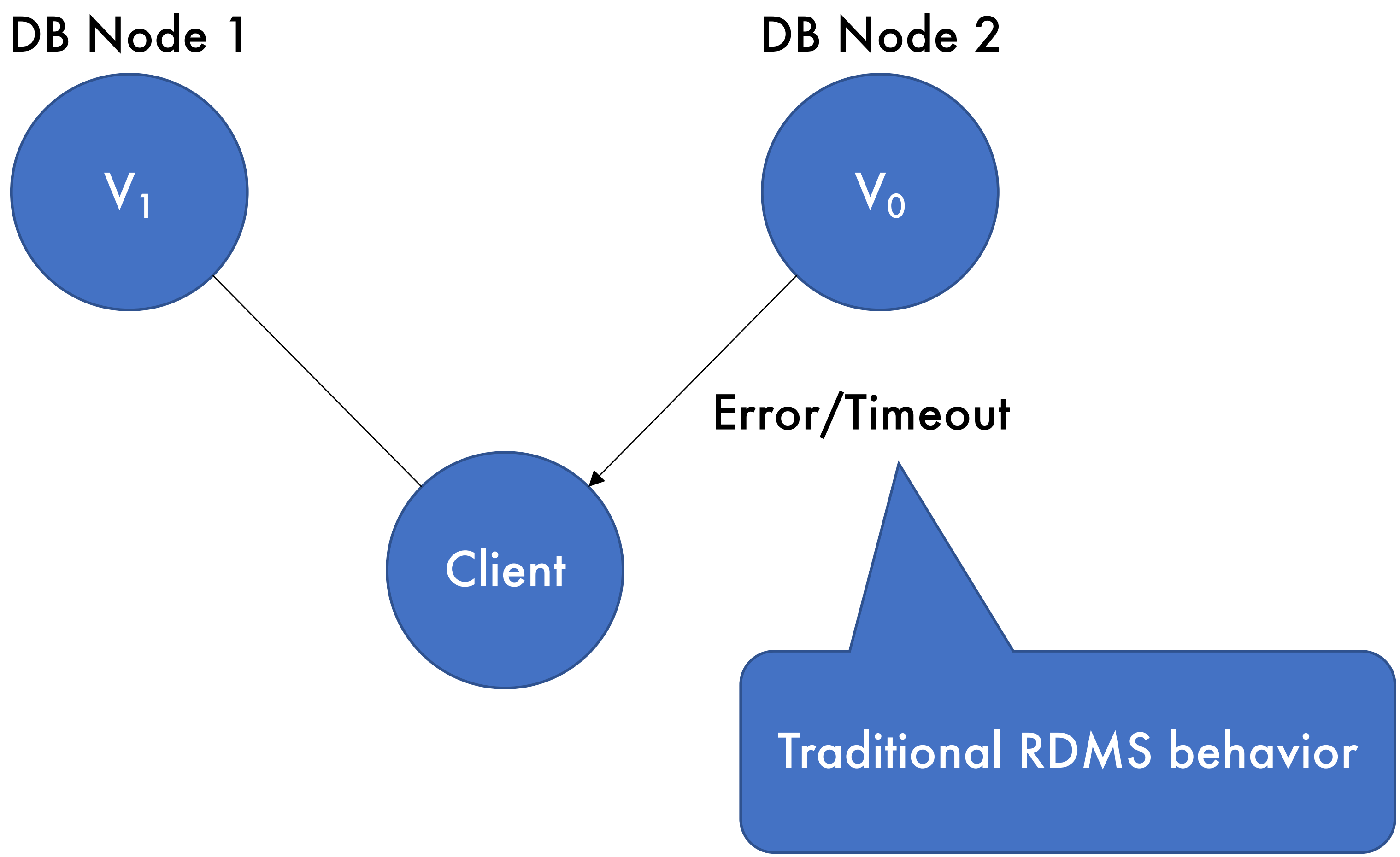
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



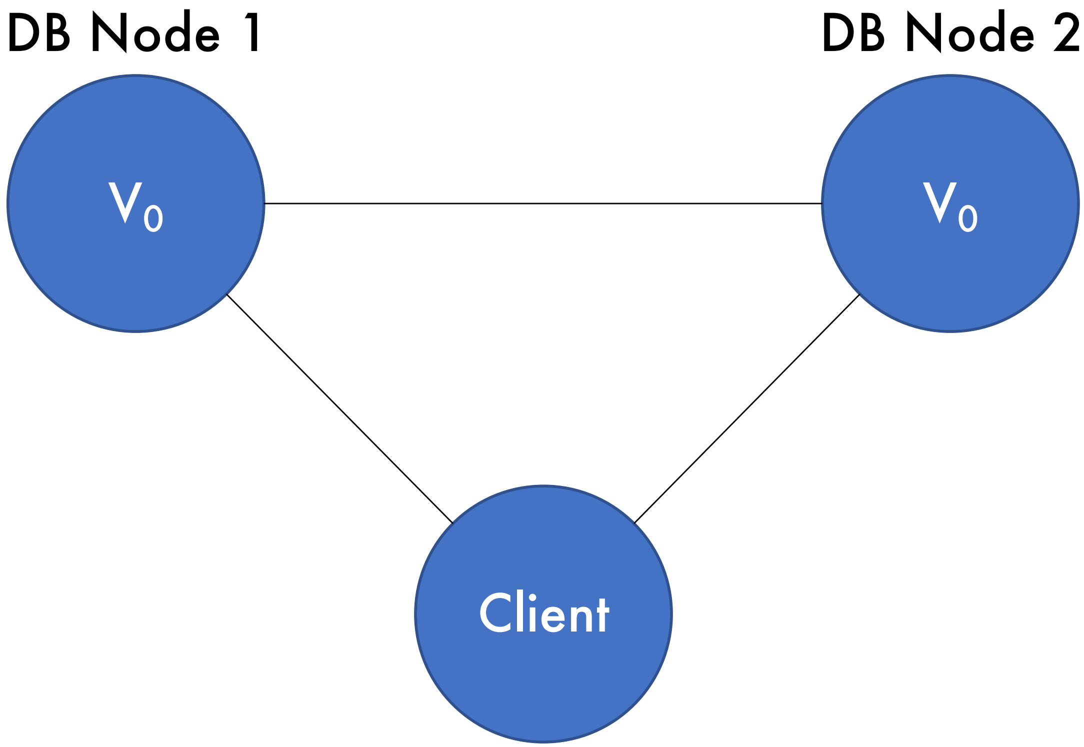
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**



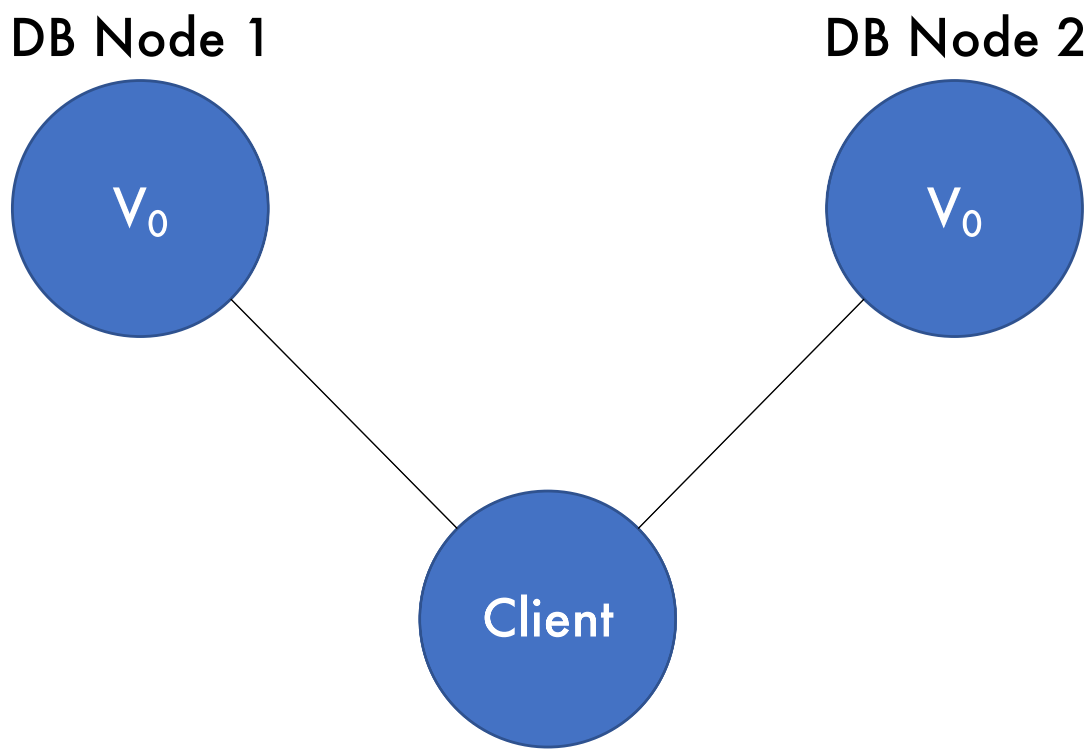
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**



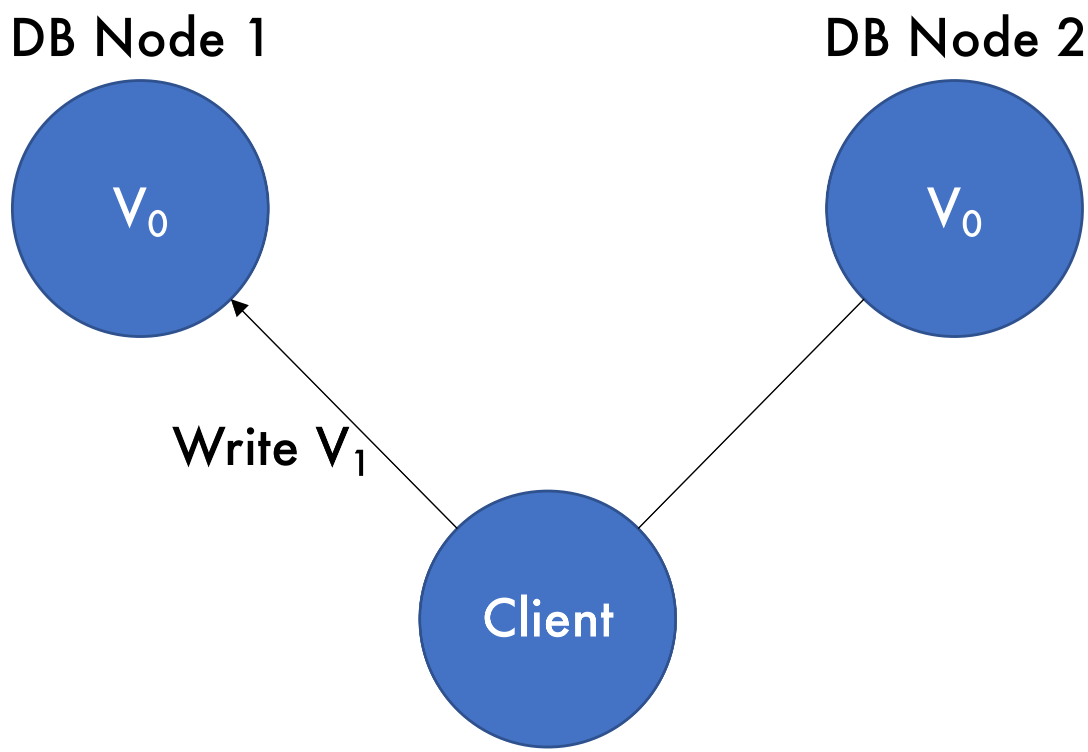
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**



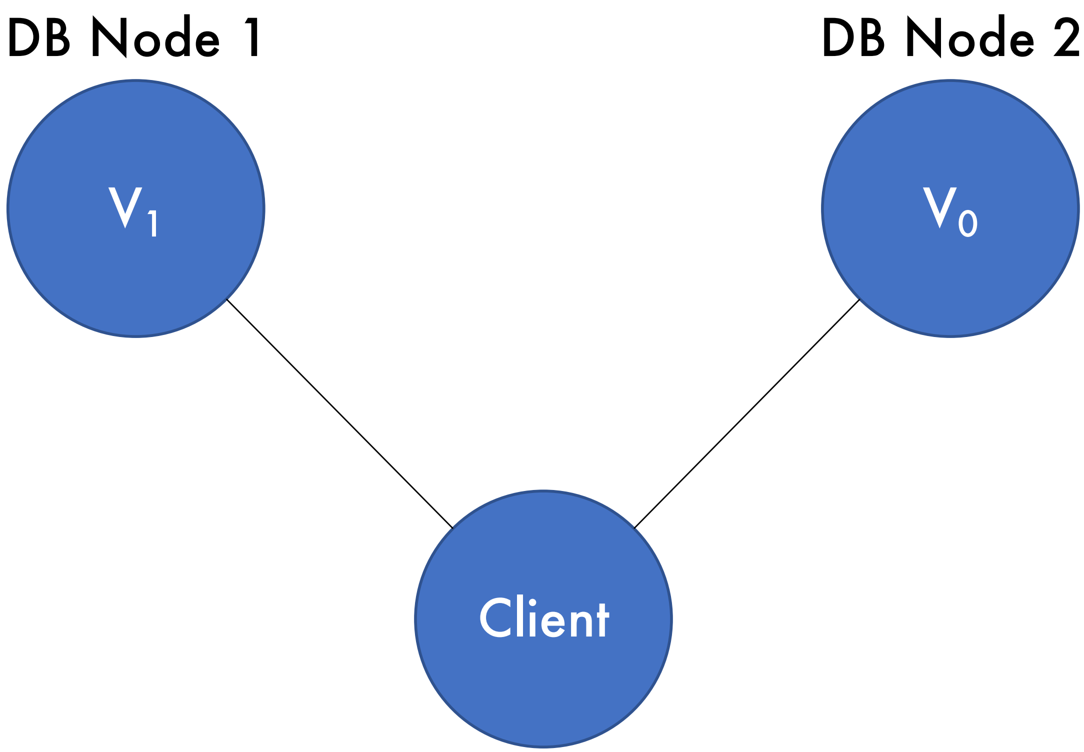
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**



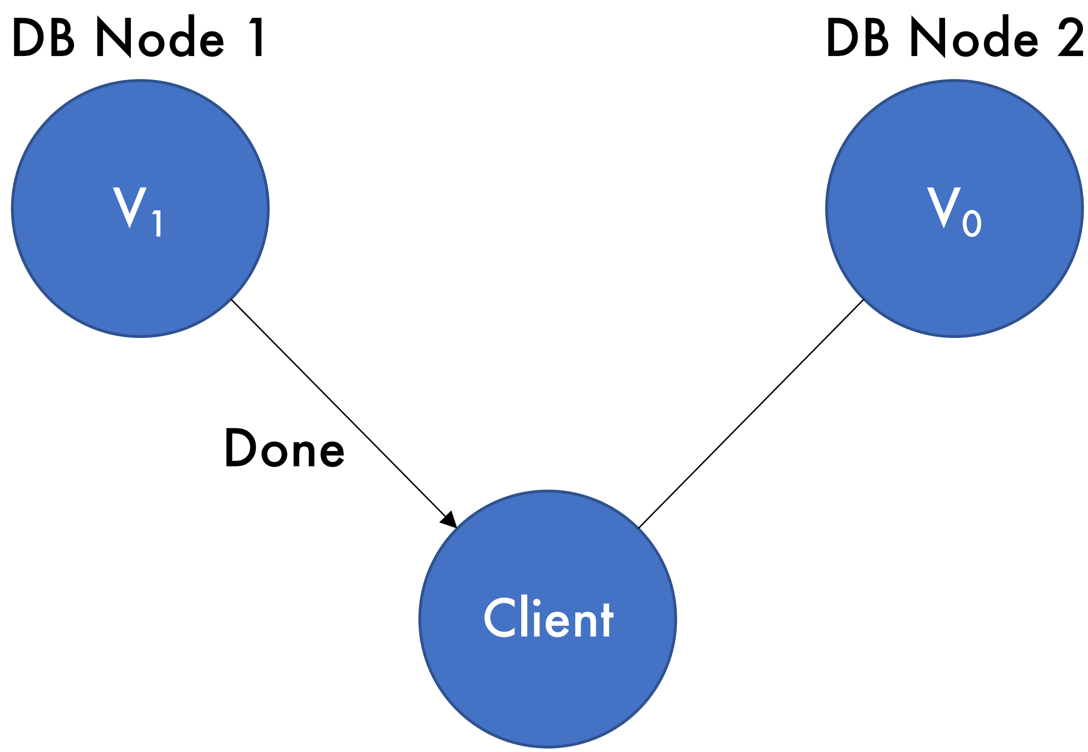
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**



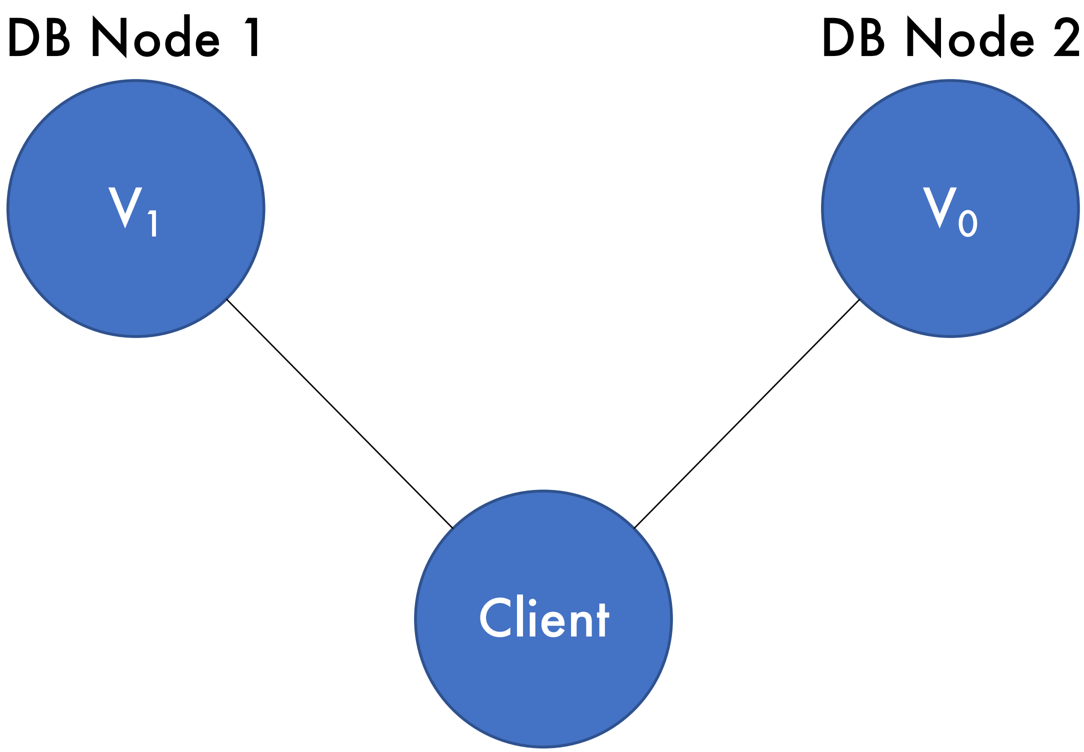
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**



RDBMS vs NoSQL Systems

Partition tolerance + **Availability**

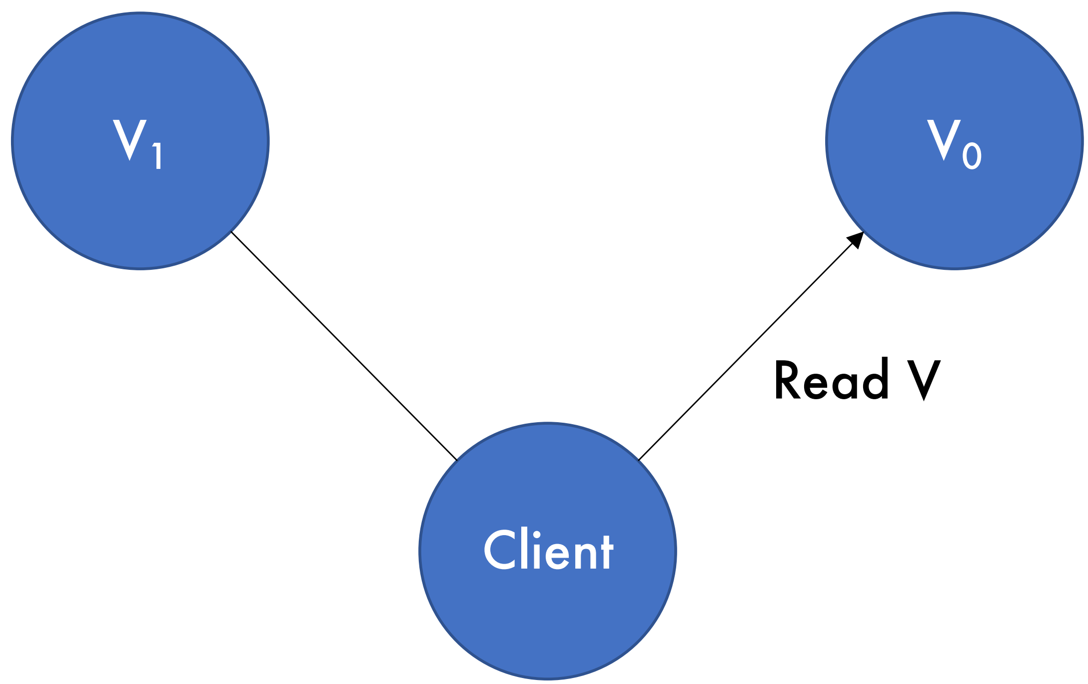


RDBMS vs NoSQL Systems

Partition tolerance + **Availability**

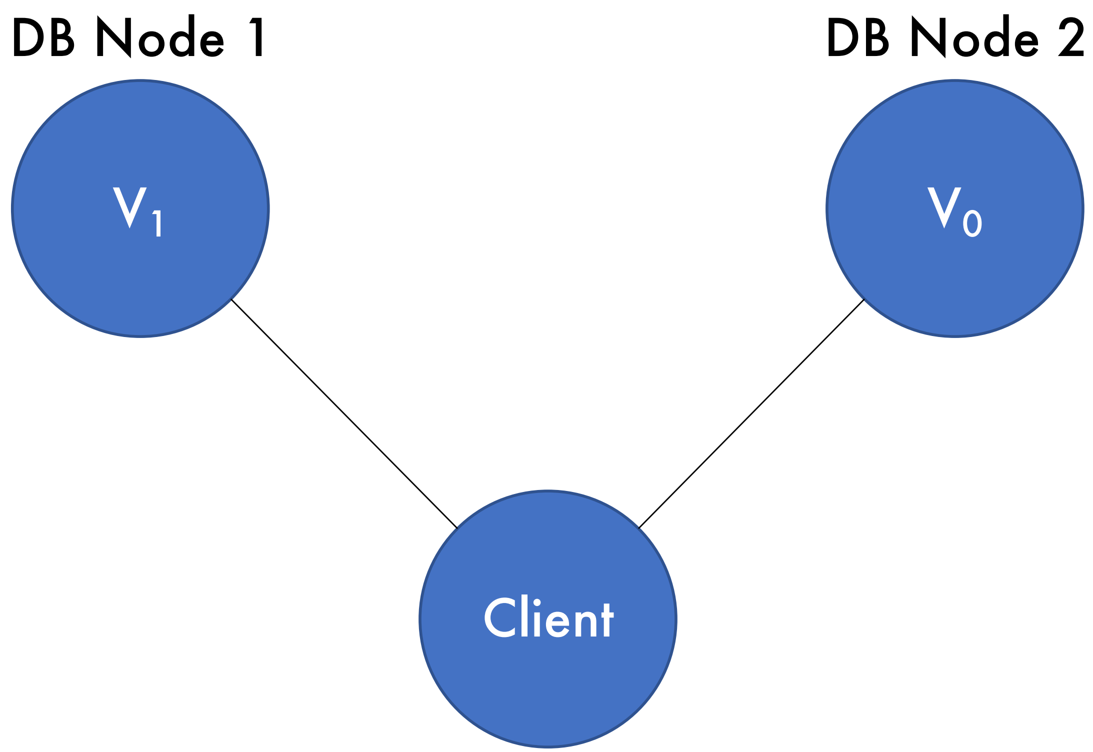
DB Node 1

DB Node 2



RDBMS vs NoSQL Systems

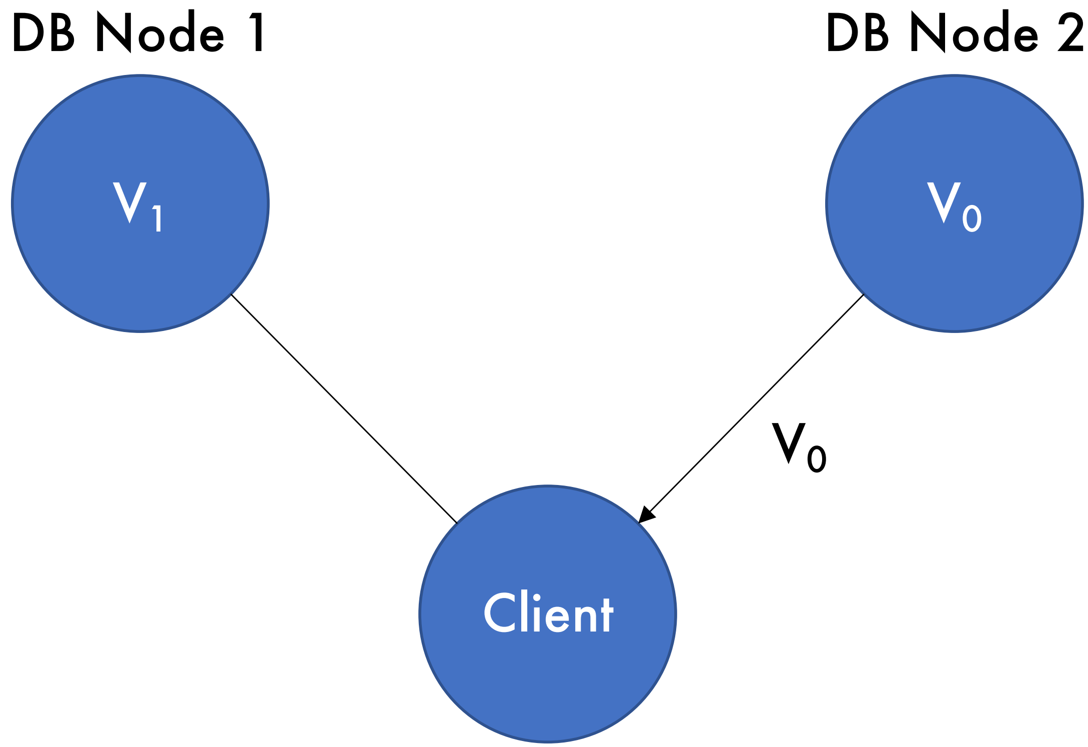
Partition tolerance + **Availability**



Tries but fails to check consistency of V

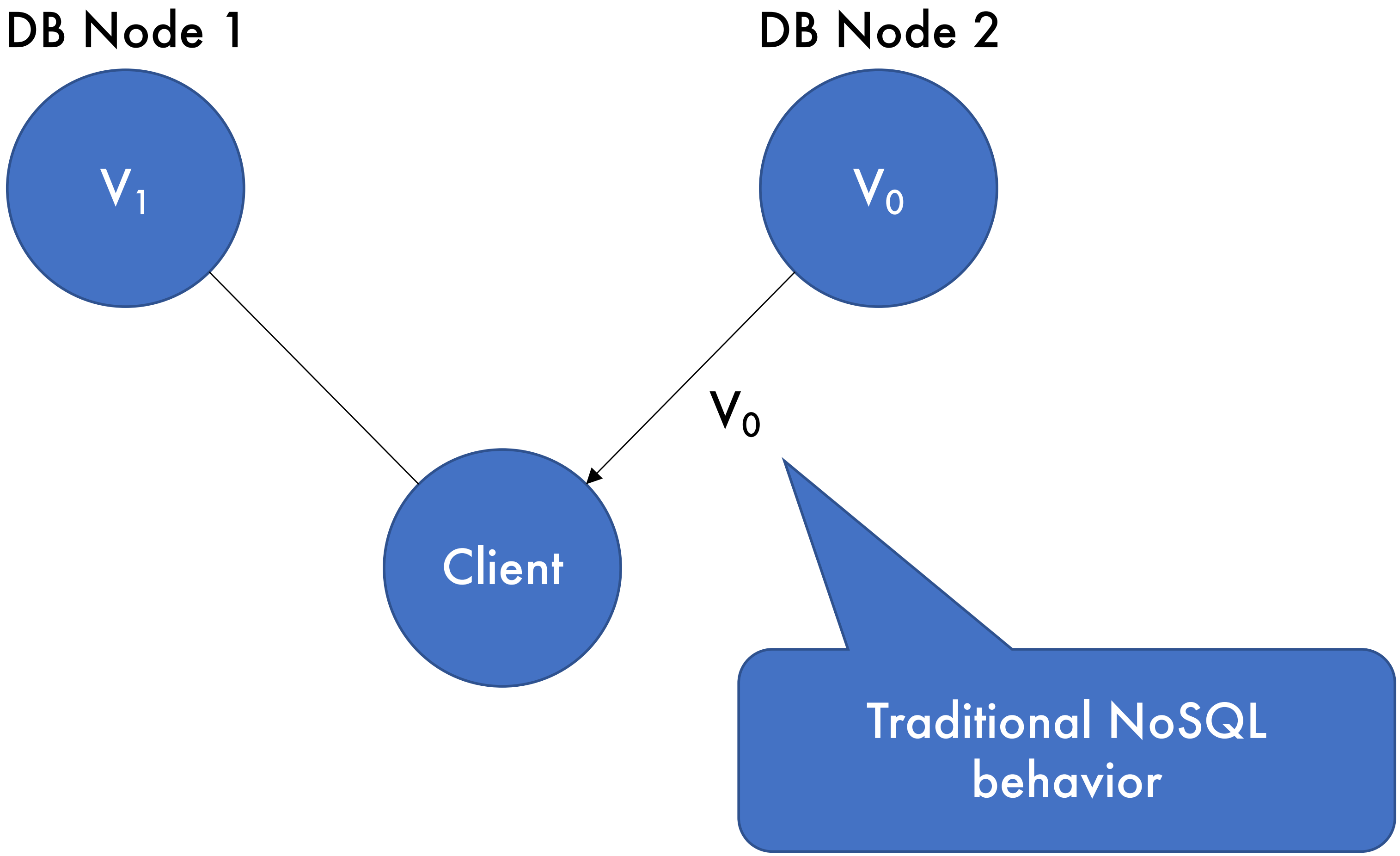
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**



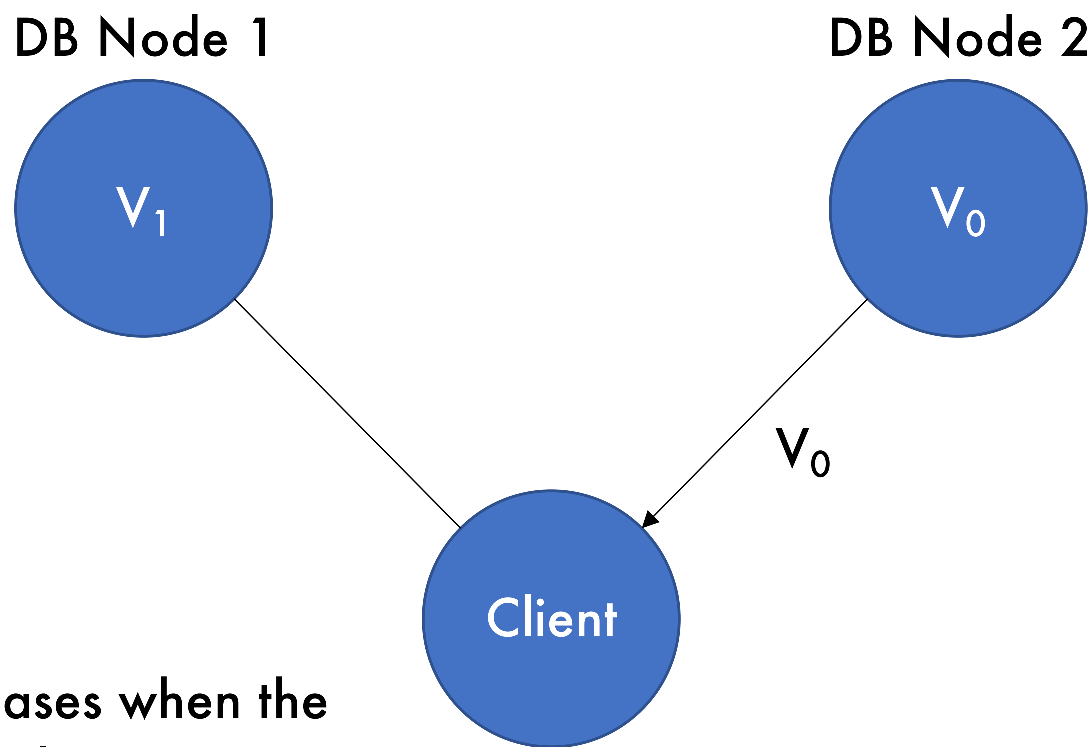
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**



RDBMS vs NoSQL Systems

Partition tolerance + **Availability**



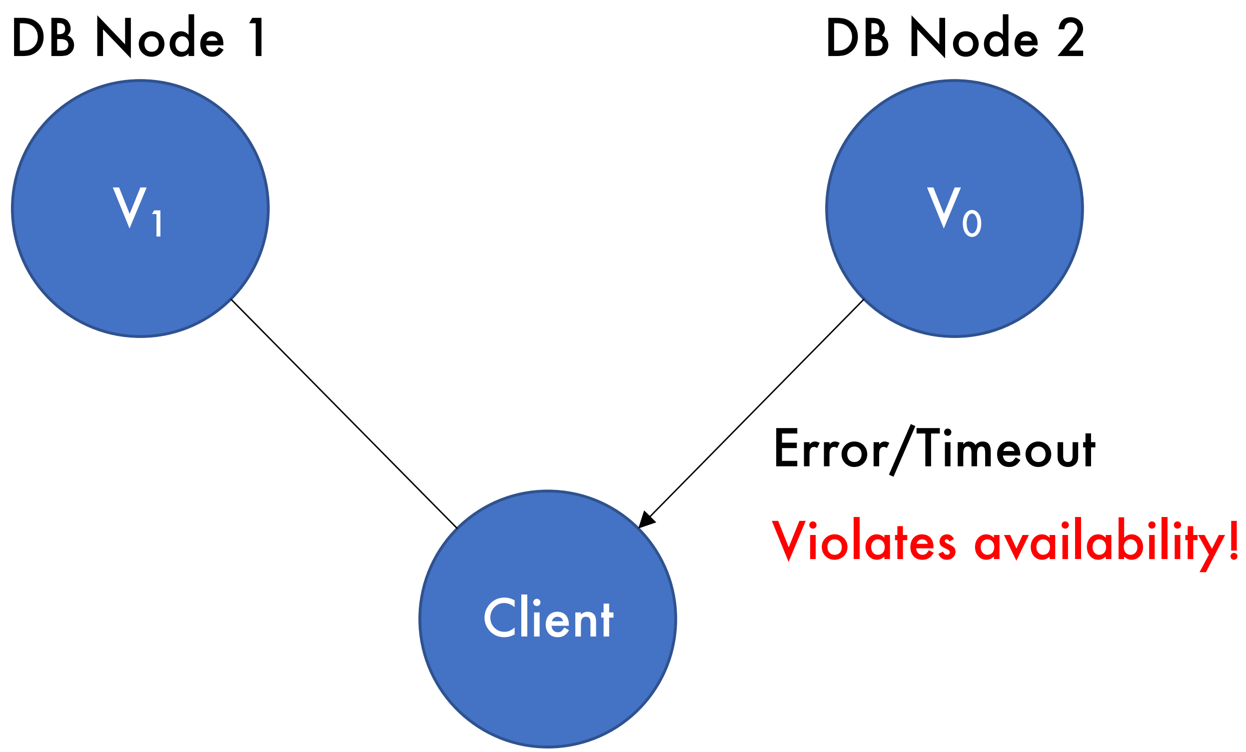
IMPORTANT:
These are only cases when the network infrastructure goes down completely. Usually, nodes should be able to check on other nodes.

Proof of CAP Theorem

- 2002 original paper (S. Gilbert & N. Lynch)
- More digestible blog post (M. Whittaker)
- Proof by contradiction: Assume we had a system that guaranteed availability, consistency, and partition tolerance...

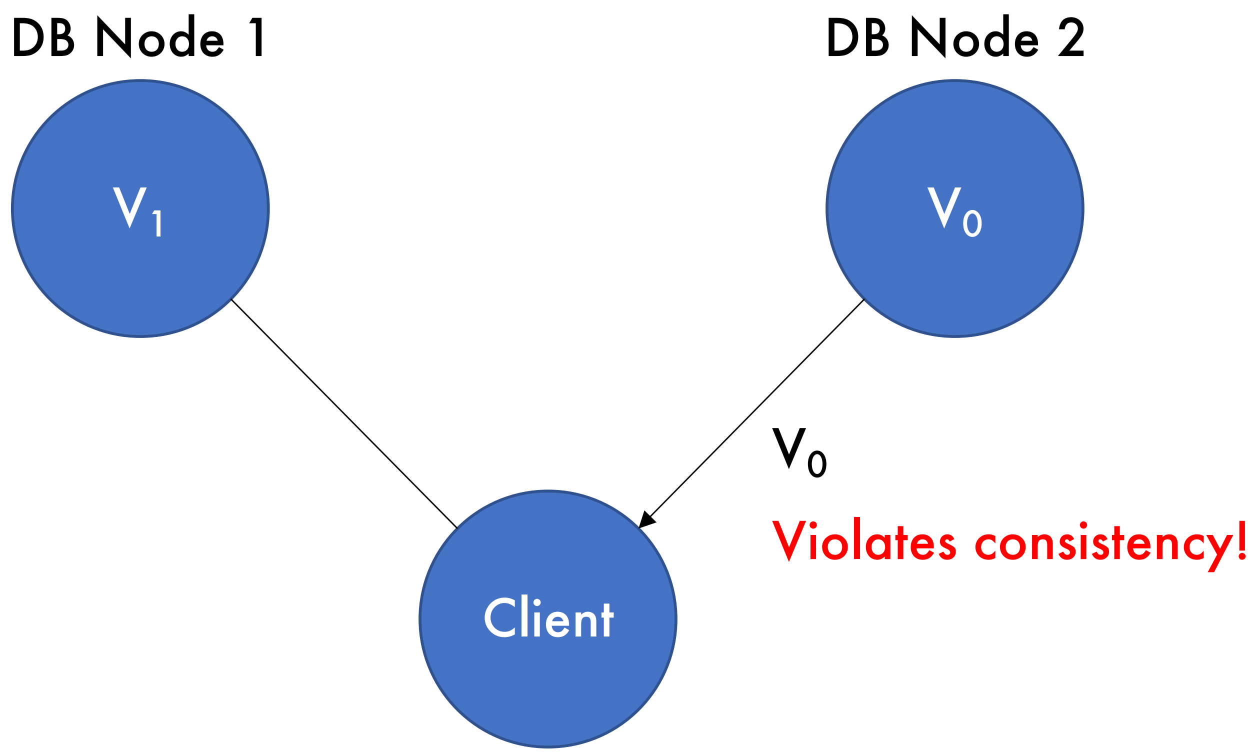
Proof of CAP Theorem

Partition tolerance + Consistency
+ **Availability**?



Proof of CAP Theorem

Partition tolerance + Availability
+ **Consistency?**



On a Practical Note

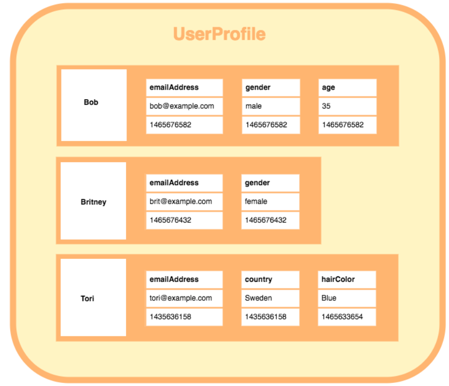
- RDBMSs are *intended* to be highly consistent
  - Boost availability by sacrificing some consistency
- NoSQL sys. are *intended* to be highly available
  - Boost consistency by sacrificing some availability
- Most applications OK with some compromise
  - "Return most of data most of the time"
  - DBMS choice has many factors
    - Consistency/Availability requirements
    - Scalability
    - Usability
    - OLAP/analysis requirements
    - …

NoSQL Data Models

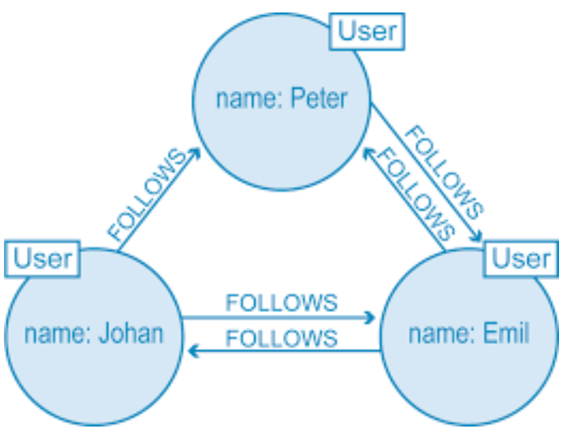
Key-Value Database

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Wide-Column Store (Extensible Record Store)



Graph Database



Document Store

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40></age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

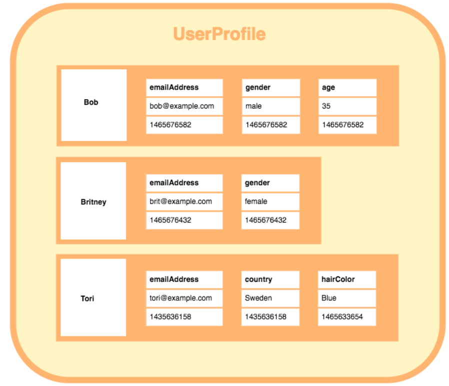
```
{  "empinfo" :
      {
        "employees" :  [
        {
           "name" : "James Kirk",
           "age" : 40,
        },
        {
           "name" : "Jean-Luc Picard",
           "age" : 45,
        },
        {
           "name" : "Wesley Crusher",
           "age" : 27,
        }
                      ]
      }
}
```

NoSQL Data Models

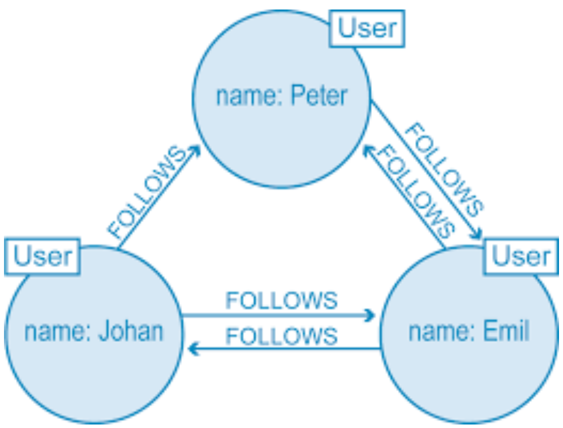
Key-Value Database

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Wide-Column Store (Extensible Record Store)



Graph Database



Document Store

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

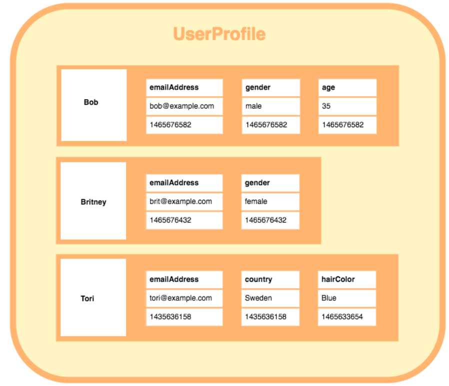
```
{  "empinfo" :
      {
         "employees" :  [
         {
            "name" : "James Kirk",
            "age" : 40,
         },
         {
            "name" : "Jean-Luc Picard",
            "age" : 45,
         },
         {
            "name" : "Wesley Crusher",
            "age" : 27,
         }
                        ]
      }
}
```

NoSQL Data Models

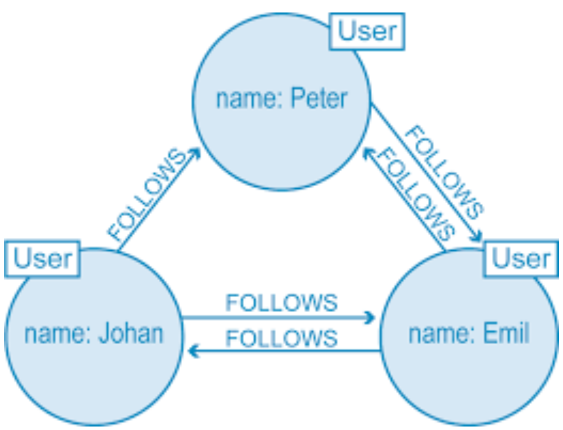
Key-Value Database

- Key to value pairs
- “A hash table”

Wide-Column Store (Extensible Record Store)



Graph Database



Document Store

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40></age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```
{  “empinfo” :
      {
        “employees” :  [
        {
           “name” : “James Kirk”,
           “age” : 40,
        },
        {
           “name” : “Jean-Luc Picard”,
           “age” : 45,
        },
        {
           “name” : “Wesley Crusher”,
           “age” : 27,
        }
                    ]
      }
   }
```

NoSQL Data Models

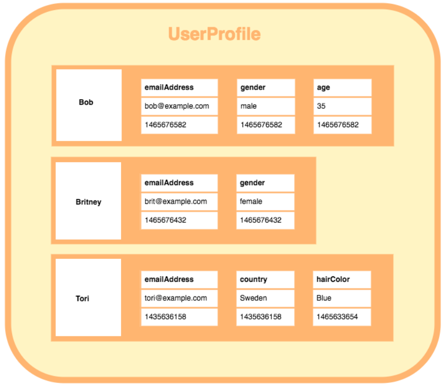
Key-Value Database

amazon DynamoDB

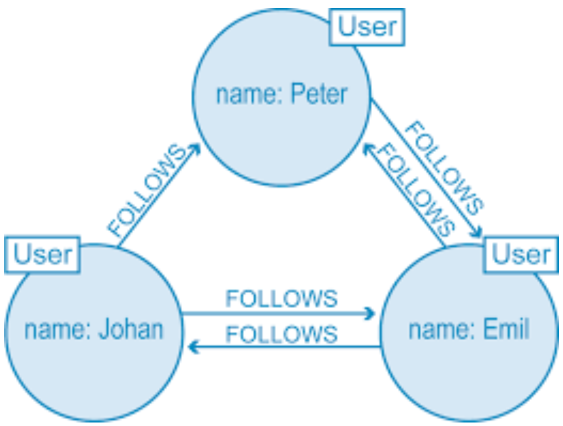
RocksDB

redis

Wide-Column Store (Extensible Record Store)



Graph Database



Document Store

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
         "employees" :  [
         {
            "name" : "James Kirk",
            "age" : 40,
         },
         {
            "name" : "Jean-Luc Picard",
            "age" : 45,
         },
         {
            "name" : "Wesley Crusher",
            "age" : 27,
         }
                        ]
      }
}
```

NoSQL Data Models

Key-Value Database

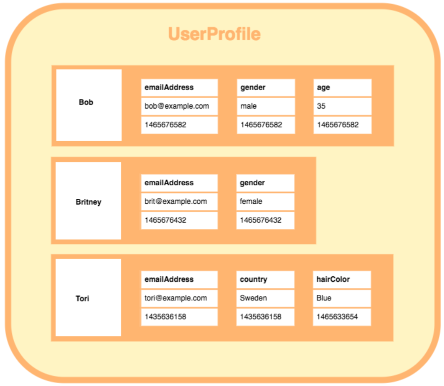
amazon DynamoDB

RocksDB

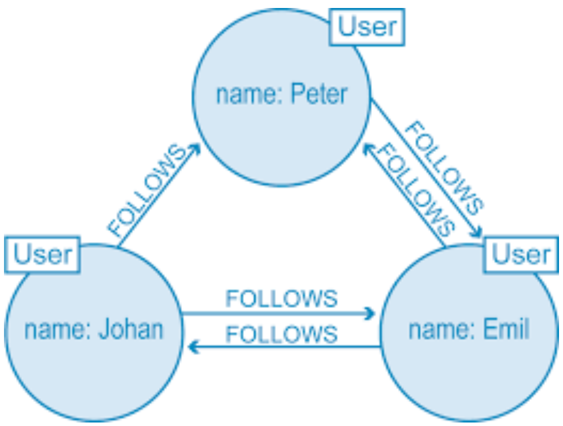
redis

Persistent KV store

Wide-Column Store (Extensible Record Store)



Gr ase



Document Store

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
         "employees" :  [
         {
            "name" : "James Kirk",
            "age" : 40,
         },
         {
            "name" : "Jean-Luc Picard",
            "age" : 45,
         },
         {
            "name" : "Wesley Crusher",
            "age" : 27,
         }
                    ]
      }
   }
```

NoSQL Data Models

Key-Value Database

amazon DynamoDB

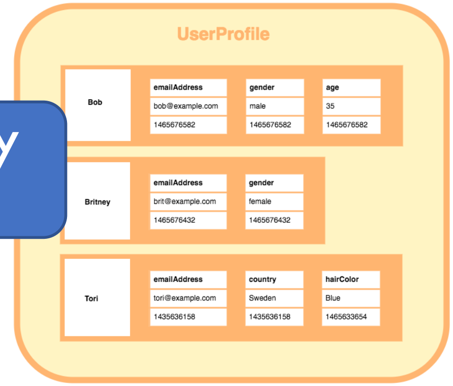
RocksDB

redis

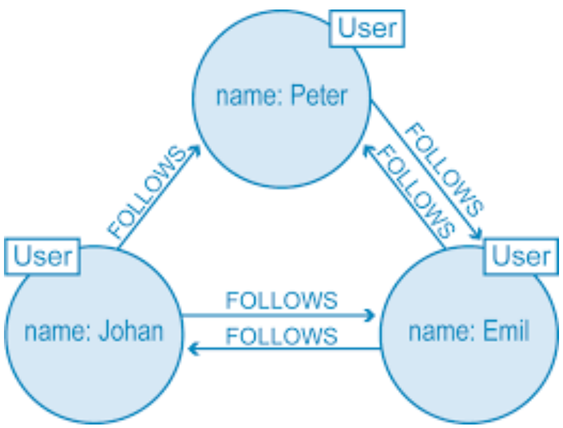
In-memory KV store

Persistent KV store

Wide-Column Store (Extensible Record Store)



Gr ase



Document Store

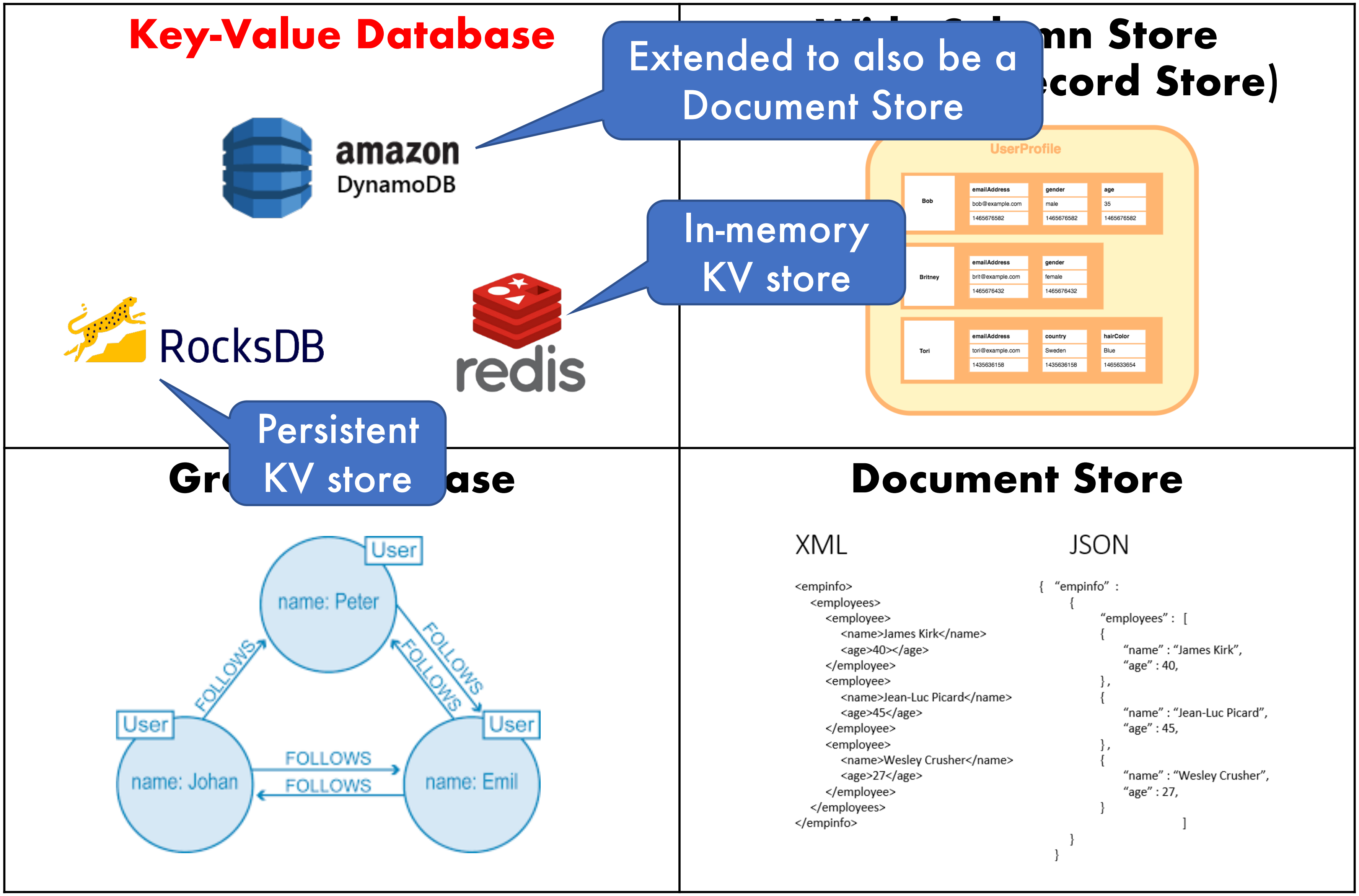
XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  “empinfo” :
     {
        “employees” :  [
        {
           “name” : “James Kirk”,
           “age” : 40,
        },
        {
           “name” : “Jean-Luc Picard”,
           “age” : 45,
        },
        {
           “name” : “Wesley Crusher”,
           “age” : 27,
        }
                    ]
     }
}
```

NoSQL Data Models



Key-Value Store

- Data model:
  - (key, value) pairs
  - Key → string/integer/..., unique for the entire data
  - Value → anything

Key-Value Store

- Data model:
  - (key, value) pairs
  - Key → string/integer/..., unique for the entire data
  - Value → anything
- Basic Operations:
  - get(key)
  - put(key, value)

Key-Value Store

- Data model:
  - (key, value) pairs
  - Key → string/integer/…, unique for the entire data
  - Value → anything
- Basic Operations:
  - get(key)
  - put(key, value)
- Distribution/Partitioning:
  - Access via hash function
  - No replication: Key k stored at server h(k)%N
  - 3-way replication: Key k stored at servers $h_1(k)\%N$, $h_2(k)\%N$, $h_3(k)\%N$

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

Key	**Value**

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

Key	Value
FID	Single flight record

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

Key	Value
FID	Single flight record
Date	All flight records on that day

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

Key	Value
FID	Single flight record
Date	All flight records on that day
(origin, destination)	All flight records between the cities

DynamoDB API

- Create, Read, Update, Delete (CRUD) actions
  - Create → **PutItem**
  - Read → **GetItem**
  - Update → UpdateItem (Document store functionality)
  - Delete → DeleteItem
- Read consistency options
  - Eventually consistent (default, may be stale data)
  - Strongly consistent (gets most recent written data)
- As of December 2018, ACID is “supported”
  - **TransactWriteItems**
  - **TransactGetItems**

Other NoSQL Data Models

- There are many other data models

NoSQL Data Models

Key-Value Database	Wide-Column Store (Extensible Record Store)
• Key to value pairs • "A hash table"	• Row + column key to value pairs • "A multidimensional hash table"
Graph Database	**Document Store**
• Entities and relationships • "Unstructured graph"	• Key to document pairs • "Semi-structured file collection"

NoSQL Data Models

Key-Value Database	Wide-Column Store (Extensible Record Store)
amazon DynamoDB RocksDB redis	Google Bigtable APACHE HBASE cassandra
Graph Database	**Document Store**
neo4j Apache TinkerPop Amazon Neptune	AsterixDB™ CouchDB mongoDB