What’s Old is New Again

NoSQL and NewSQL

Donnie Mattingly

Designing Cheap Applications

Donnie Mattingly

Standings | Leaderboard | Picks | History

2019 Masters Standings Leaderboard **2019 PGA** Standings Leaderboard

2019 Masters Leaderboard

Pos	Name	Total	Today	Penalty	Thru	Tier	Picked By
1	Tiger Woods	-13	-2	-4 🏆	F	A	Brad
T2	Dustin Johnson	-12	-4	0	F	A	Christian, Hamilton, Brice, Jeff, Marc, WD, Brian, Brandon
T2	Xander Schauffele	-12	-4	0	F	C	Christian, Hamilton, Brice, Brad, Jeff, Marmaduke, Rob, Marc, Basil
T2	Brooks Koepka	-12	-2	0	F	B	Brice, Brad, Jeff, Trevor, Basil, Scott
T5	Jason Day	-11	-5	0	F	B	Jhahn
T5	Webb Simpson	-11	-2	0	F	C	Lizza
T5	Francesco Molinari	-11	+2	0	F	B	Rob, Carl, Jhahn
T5	Tony Finau	-11	E	0	F	C	Brad, Jeff, Rob, Basil, Donnie
T9	Jon Rahm	-10	-4	0	F	B	
T9	Patrick Cantlay	-10	-4	0	F	C	
T9	Rickie Fowler	-10	-3	0	F	B	Christian, Hamilton, Lizza, Jeff, Rob, WD, Brian, Scott, Brandon, Donnie, Darien
T12	Bubba Watson	-8	-3	0	F	B	WD
T12	Justin Thomas	-8	-2	0	F	B	Christian, Brice, Lizza, Brad, Marc, Basil
T12	Matt Kuchar	-8	E	0	F	C	Christian, Brad, Marmaduke, Rob, Trevor, WD
T12	Justin Harding	-8	E	0	F	D	
T12	Ian Poulter	-8	+1	0	F	C	Christian

The Goal: cheap as possible

So, don’t use a database

S3 as a Database

```
/tournaments/masters/2019/picks
/tournaments/masters/2019/picks/picks.json
```

S3 as a Database

```
/tournaments/masters/2019/picks
/tournaments/masters/2019/picks/picks.json
/tournaments/masters/2019/picks/individual/donnie.json
```

S3 as a Database

```
/tournaments/masters/2019/picks
/tournaments/masters/2019/picks/picks.json
/tournaments/masters/2019/picks/individual/donnie.json
/tournaments/masters/2019/picks/individual/lizza.json
```

S3 as a Database

```
/tournaments/masters/2019/picks
/tournaments/masters/2019/picks/picks.json
/tournaments/masters/2019/picks/individual/donnie.json
/tournaments/masters/2019/picks/individual/lizza.json
```

Getting Started with AWS Billing & Cost Management

- Manage your costs and usage using AWS Budgets
- Visualize your cost drivers and usage trends via Cost Explorer
- Dive deeper into your costs using the Cost and Usage Reports with Athena integration
- **Learn more:** Check out the AWS What's New webpage

Do you have Reserved Instances (RIs)?

- Access the RI Utilization & Coverage reports—and RI purchase recommendations—via Cost Explorer.

Spend Summary

Cost Explorer

Welcome to the AWS Billing & Cost Management console. Your last month, month-to-date, and month-end forecasted costs appear below.

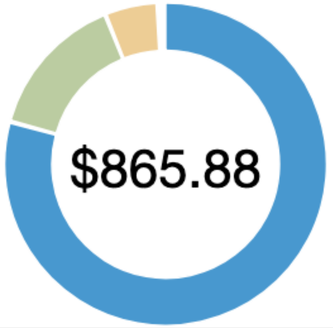
Current month-to-date balance for April 2019

$1,079.29

Month-to-Date Spend by Service

Bill Details

The chart below shows the proportion of costs spent for each service you use.



S3	$865.88
Lambda	$157.18
CloudWatch	$55.06
CodeBuild	$0.68
Other Services	$0.50
Tax	$0.00
Total	**$1,079.29**

The Goal: cheap as possible

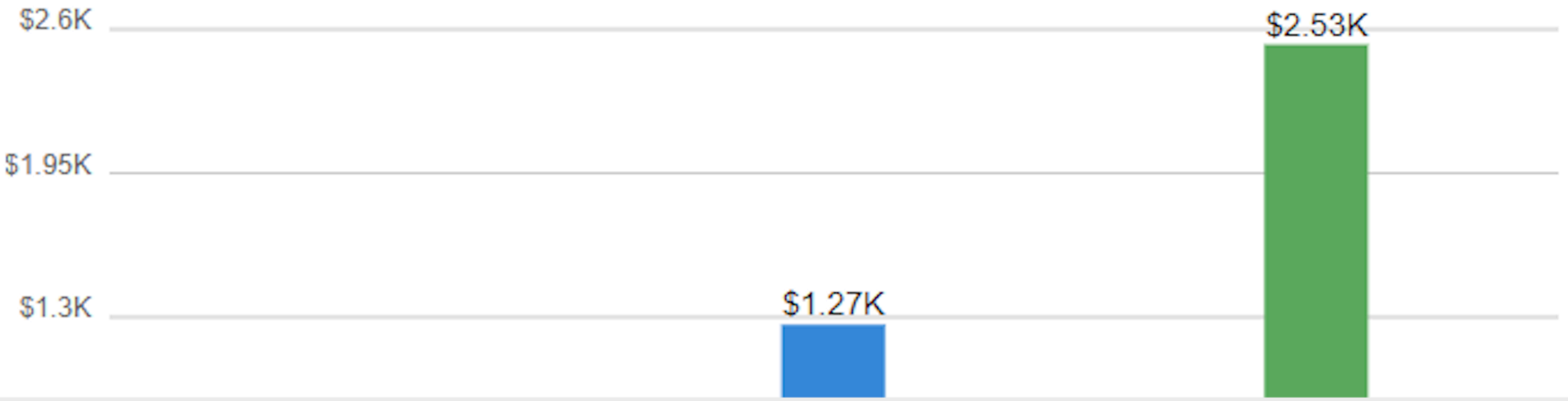
Spend Summary

Cost Explorer

Welcome to the AWS Billing & Cost Management console. Your last month, month-to-date, and month-end forecasted costs appear below.

Current month-to-date balance for April 2019

$1,265.48



The Goal: cheap as possible

Welcome to the AWS Billing & Cost Management c
forecasted costs appear below.

Current month-to-date balance for April 2019

$1,265.48

Some Numbers

- Over less than 20 hours we had
  - 111,759,304 PUT Requests ($563)
  - 754,952,935 GET Requests ($301)
  - $43K / month at that rate
  - 15 users

The Problem: I tried to build a database

Goals

- Background: RDBMS
- CAP and PACELC
- NoSQL
- NewSQL

What we want from Relational Databases

- Based on the Relational Model of Data
- SQL
- Data Integrity
- Transactions
- ACID Compliance

ORACLE®

Why leave the comfortable arms of Relational Databases?

You shouldn’t!

- Stack Overflow runs on 4 Microsoft SQL Servers
- Wikipedia uses MariaDB
- Relational Databases will almost always suit your needs

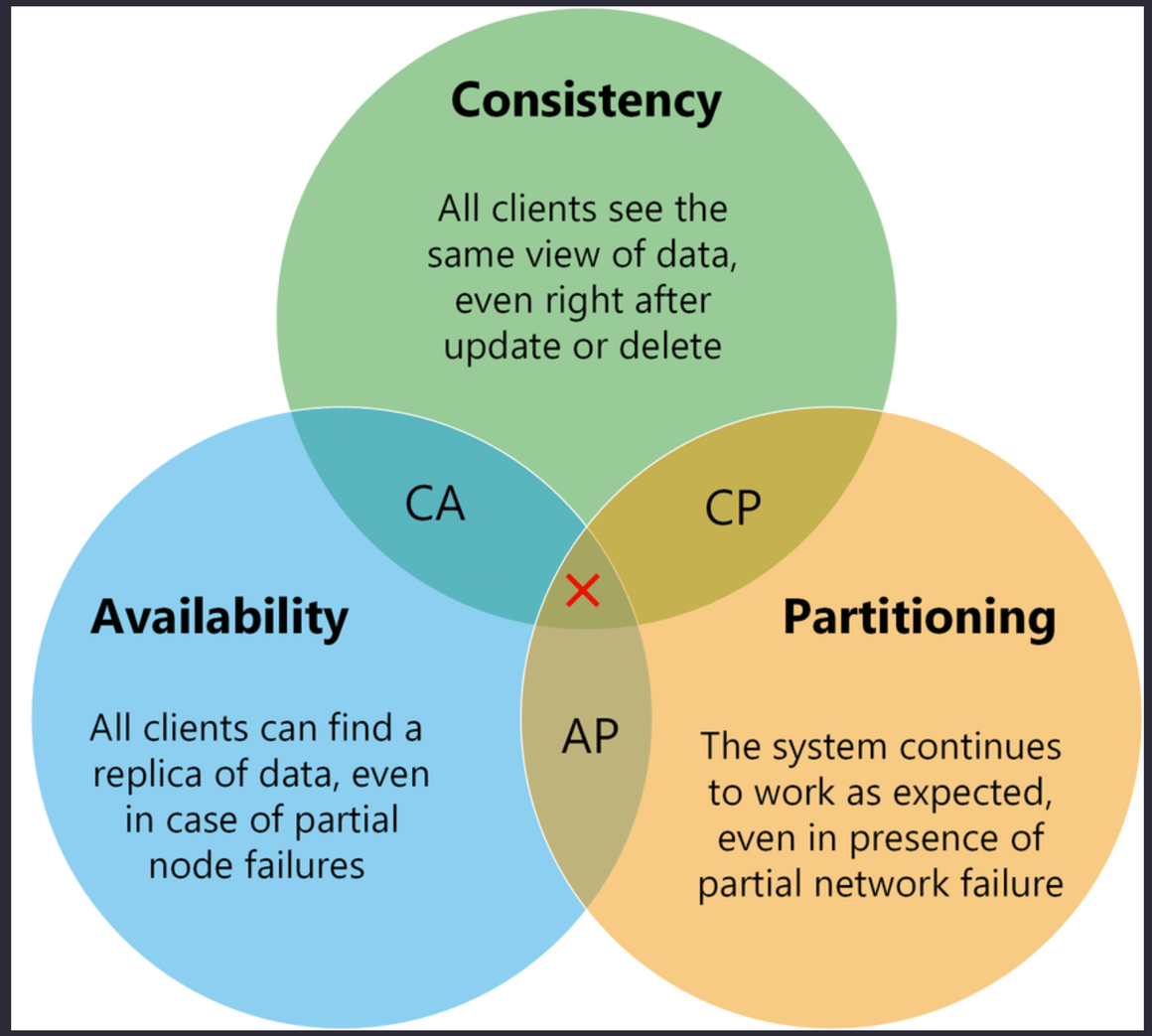
Some Exceptions

- Write heavy workloads
- Truly Big Data
- Data Model is a better fit for NoSQL

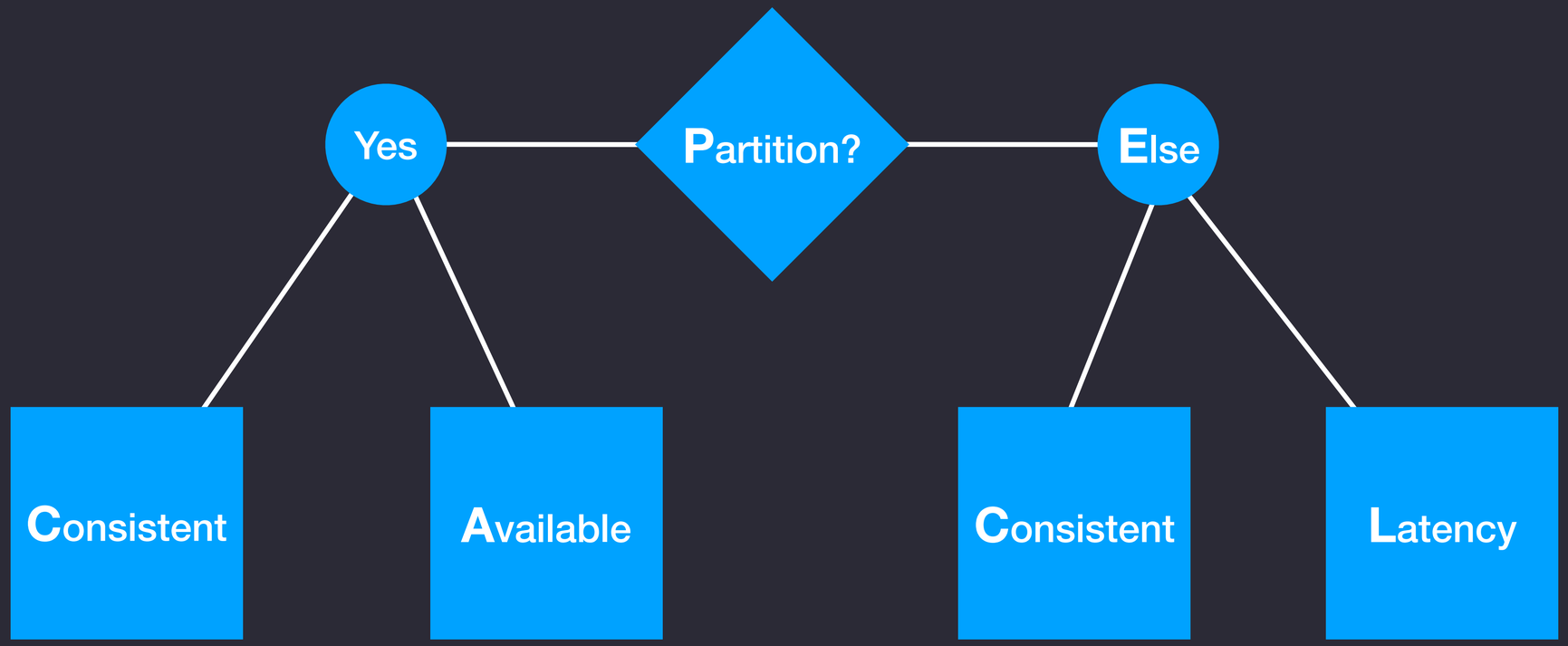
CAP and PACELC

CAP Theorem

- In the event of a network **partition**, a database can either be **available** or **consistent**, but not both.
- **CP** systems:
  - MongoDB, HBase, Memcache
- **AP** systems:
  - Cassandra, Riak, CouchDB
- **CA** systems:
  - Traditional RDBMS



PACELC


Yes
Partition?
Else
Consistent
Available
Consistent
Latency

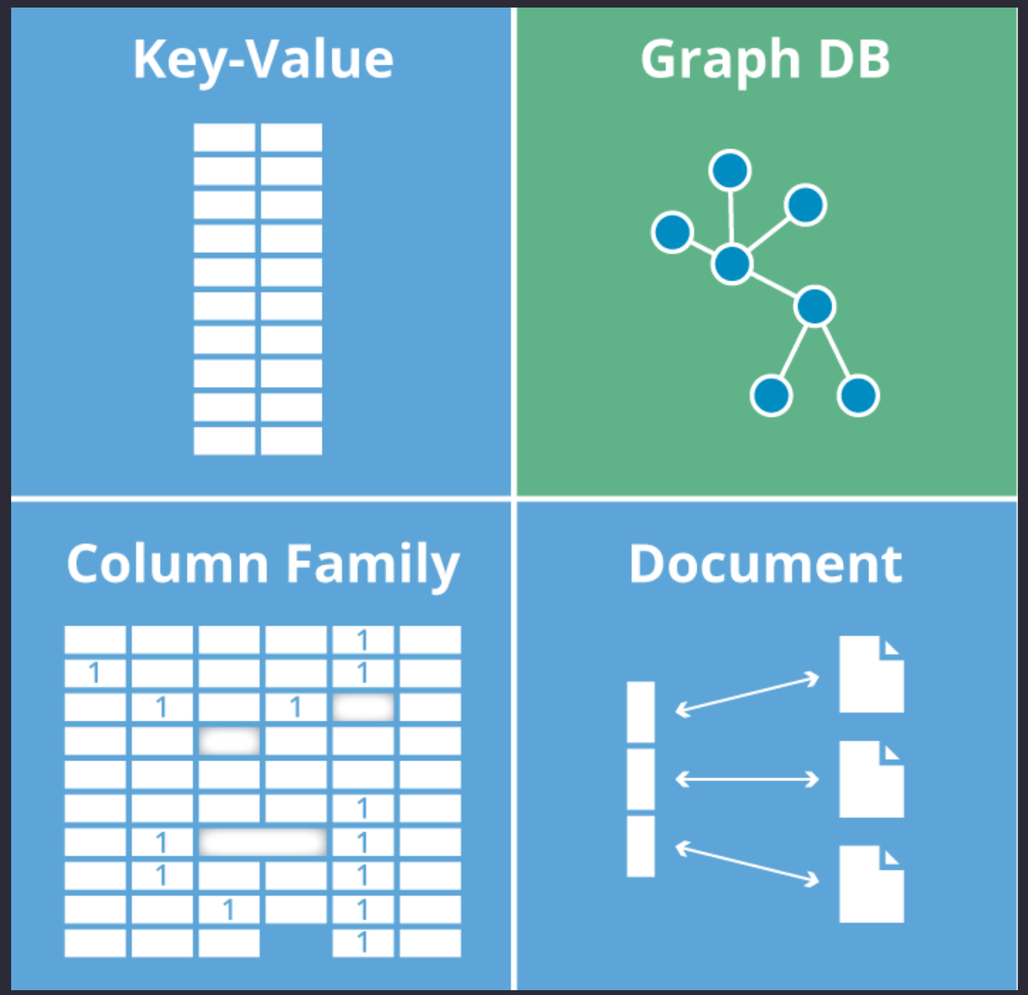
RDMBSs are CA systems, but what if you want a different compromise

RDMBSs are single node systems, but what if you want a different compromise

NoSQL

NoSQL

- Not Only SQL
- Not a single technology, but many



NoSQL Goals

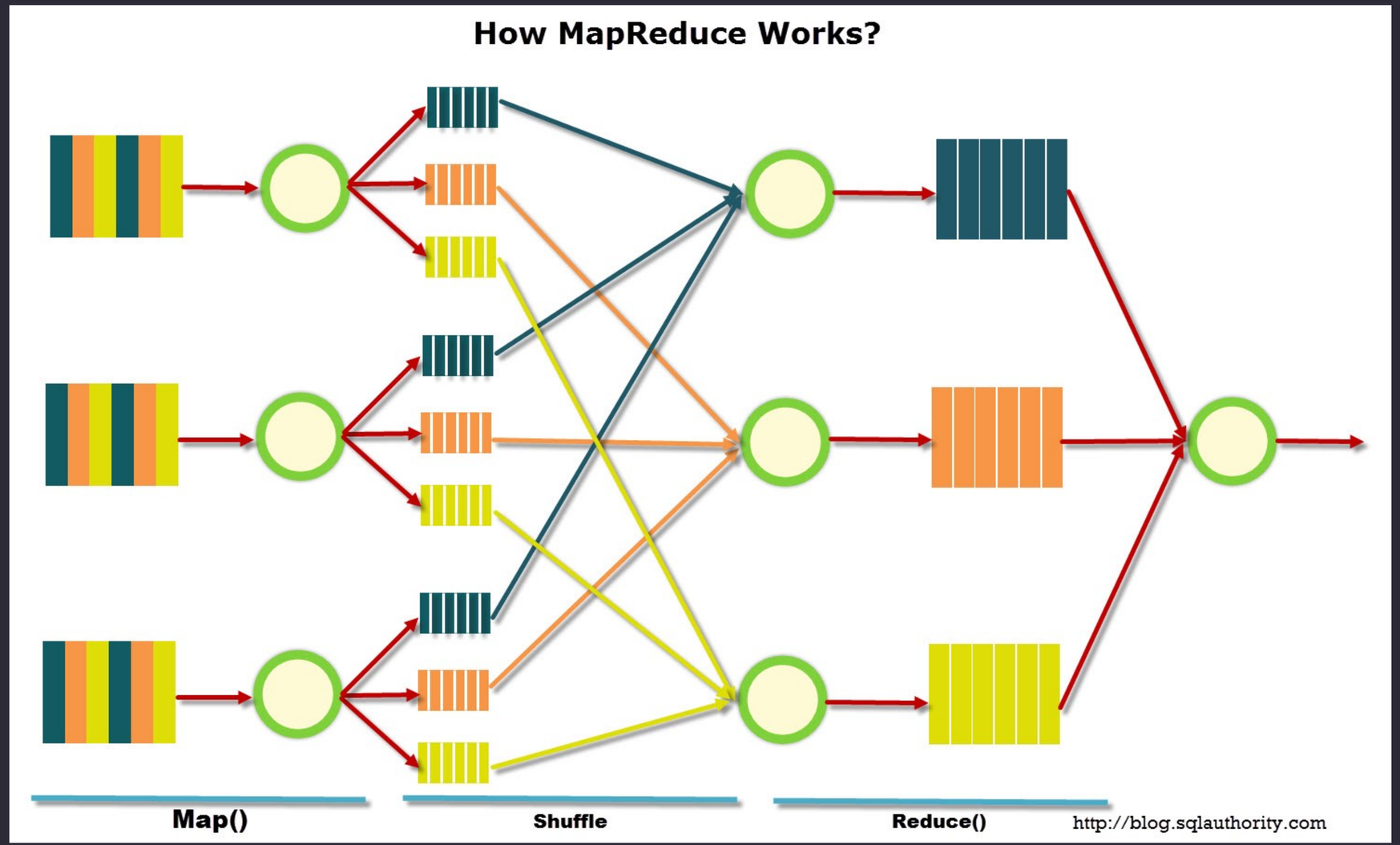
- Brief History of NoSQL Technologies
- Deeper Dive into DynamoDB

MapReduce

MapReduce

- Google solution for processing big data
- Paper published 2004
- Apache Hadoop released as an open source alternative
- Bases on three main steps: Map, Shuffle, and Reduce

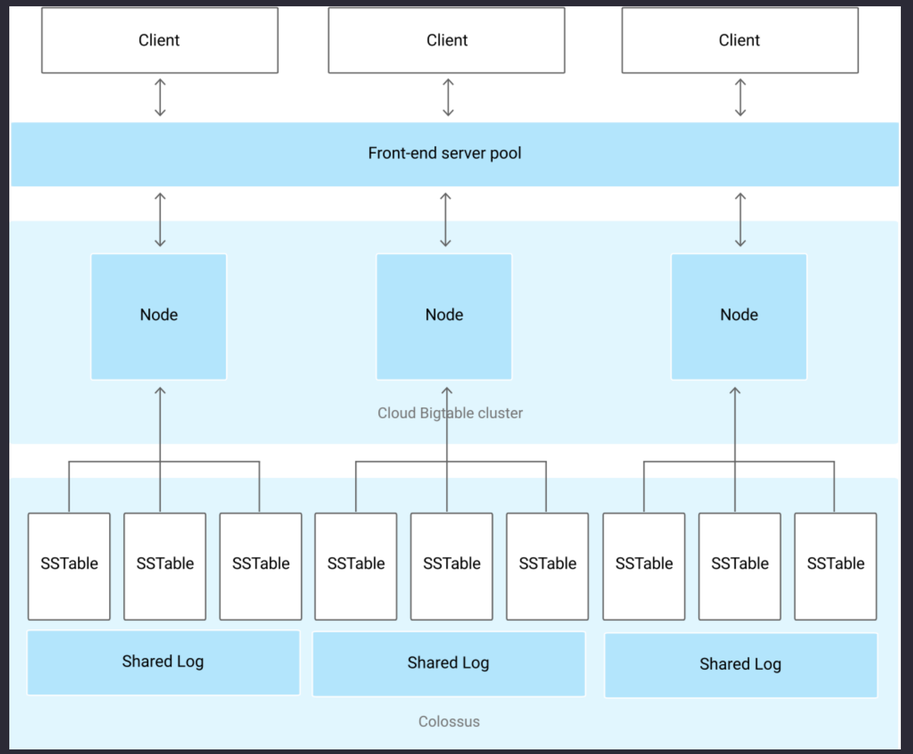
MapReduce


How MapReduce Works?
Map()
Shuffle
Reduce()
http://blog.sqlauthority.com

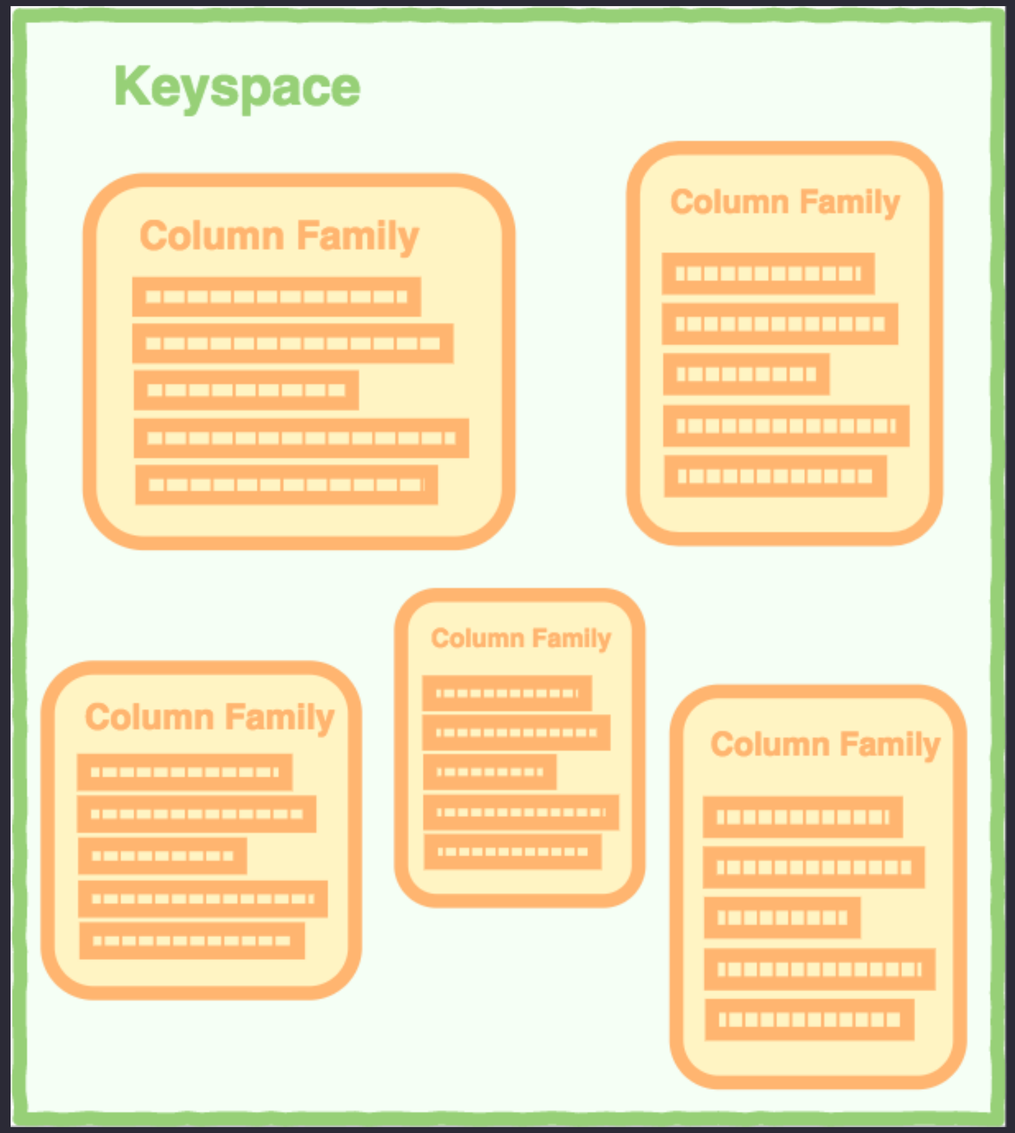
BigTable

BigTable

- Wide-Column Store from Google
- Paper published 2006
- Master node for metadata, child nodes for read and writes
- Multidimensional Sorted Map
- Apache HBase for an open source version



BigTable



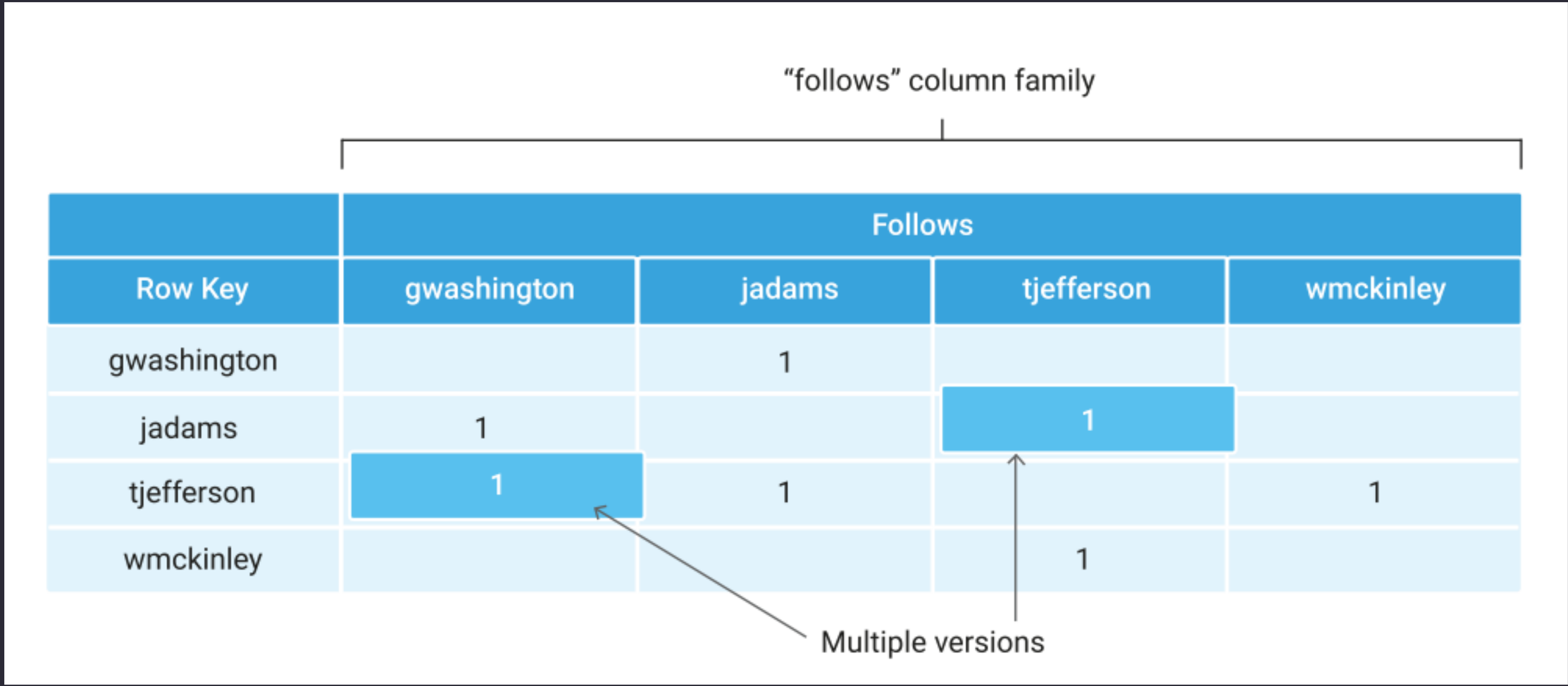
UserProfile

Bob	**emailAddress**	**gender**	**age**
	bob@example.com	male	35
	1465676582	1465676582	1465676582

Britney	**emailAddress**	**gender**
	brit@example.com	female
	1465676432	1465676432

Tori	**emailAddress**	**country**	**hairColor**
	tori@example.com	Sweden	Blue
	1435636158	1435636158	1465633654

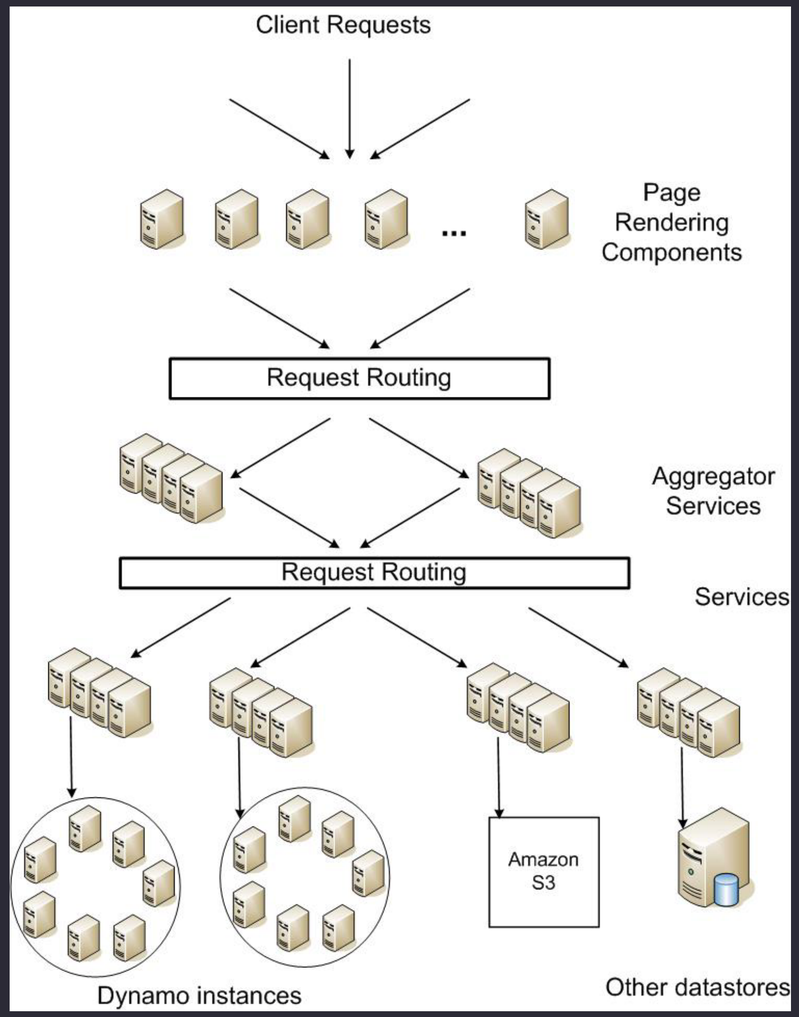
BigTable


"follows" column family
Follows
Row Key
gwashington
jadams
tjefferson
wmckinley
gwashington
1
jadams
1
1
tjefferson
1
1
1
wmckinley
1
Multiple versions

Dynamo

Dynamo

- Key-Value Store from Amazon
- Paper published 2007
- Decentralized
- Writes must never fail
- Gossip based Replication



Dynamo

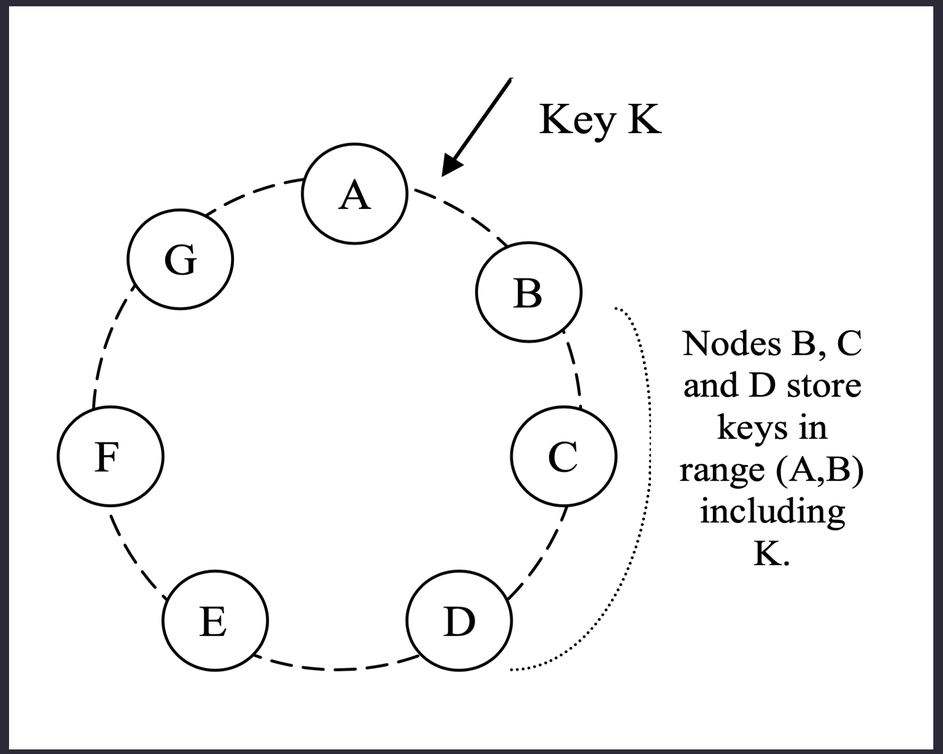
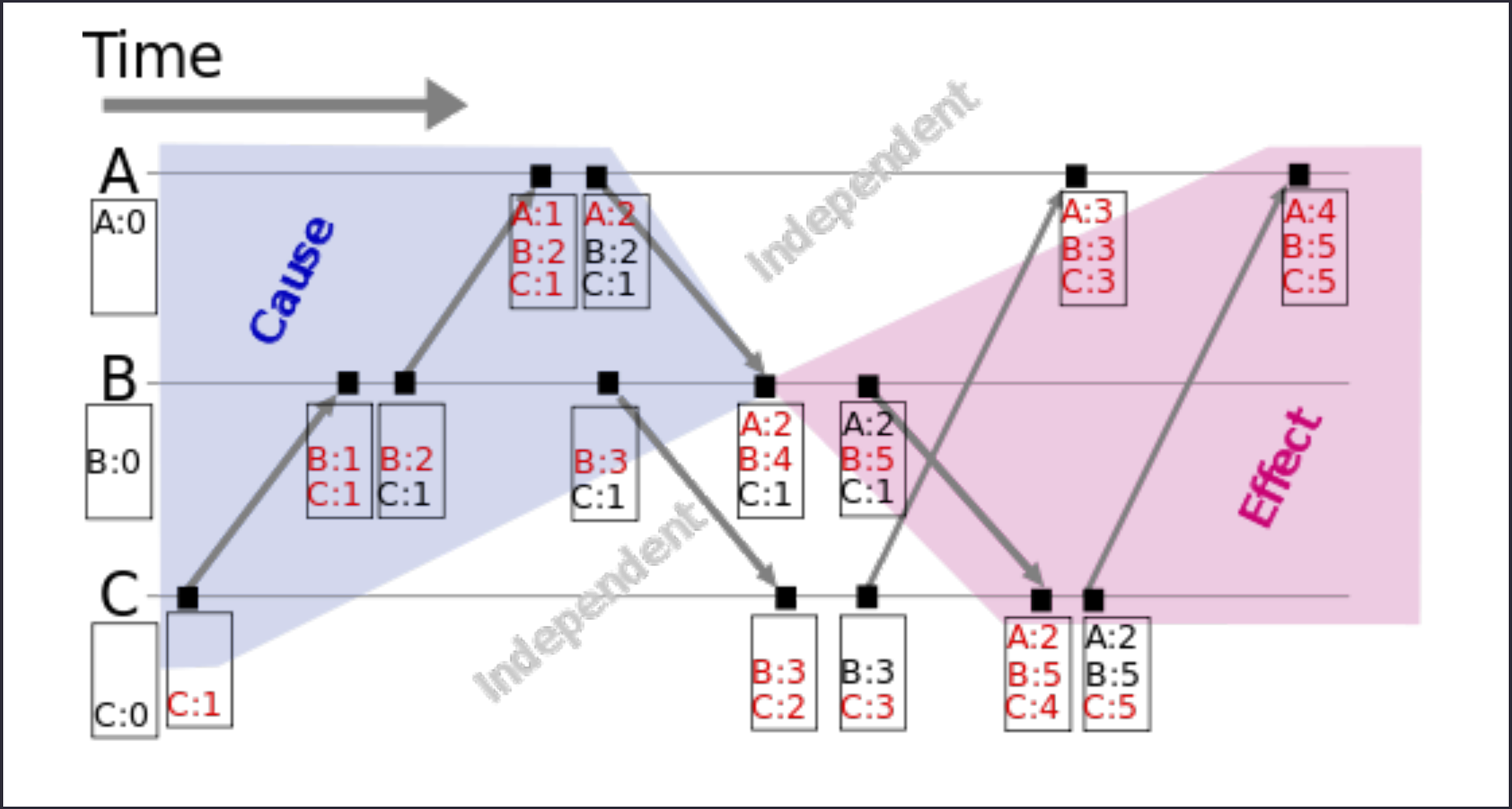


Table 1: Summary of techniques used in *Dynamo* and their advantages.

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Vector Clocks

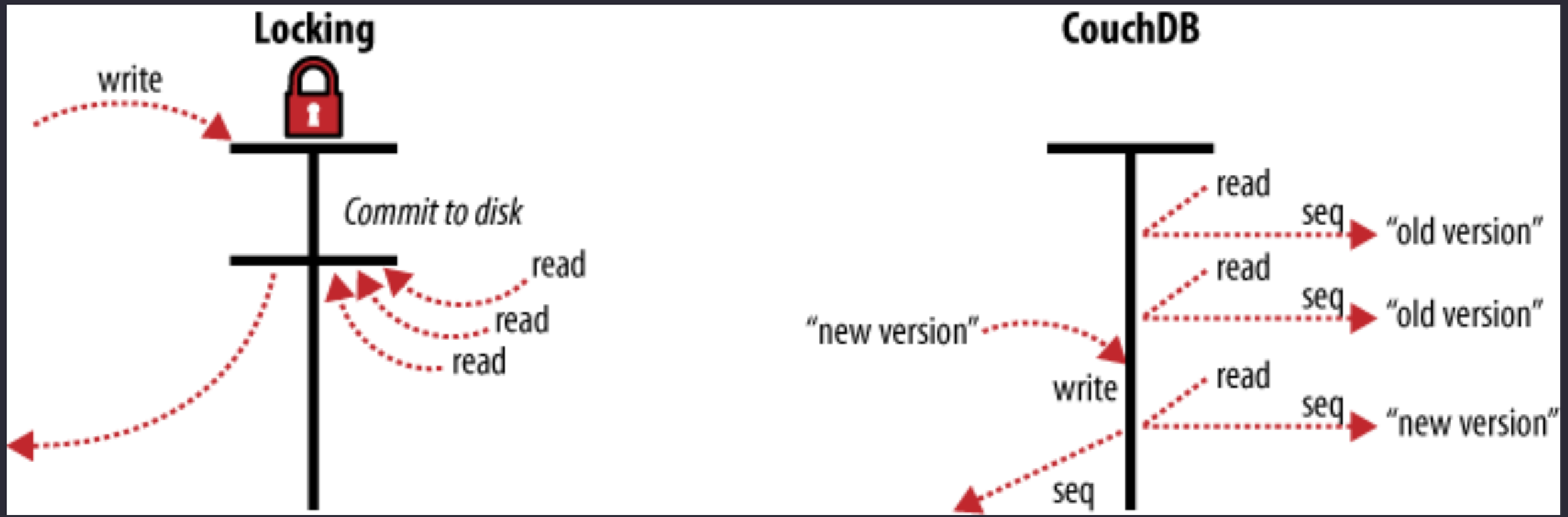


CouchDB

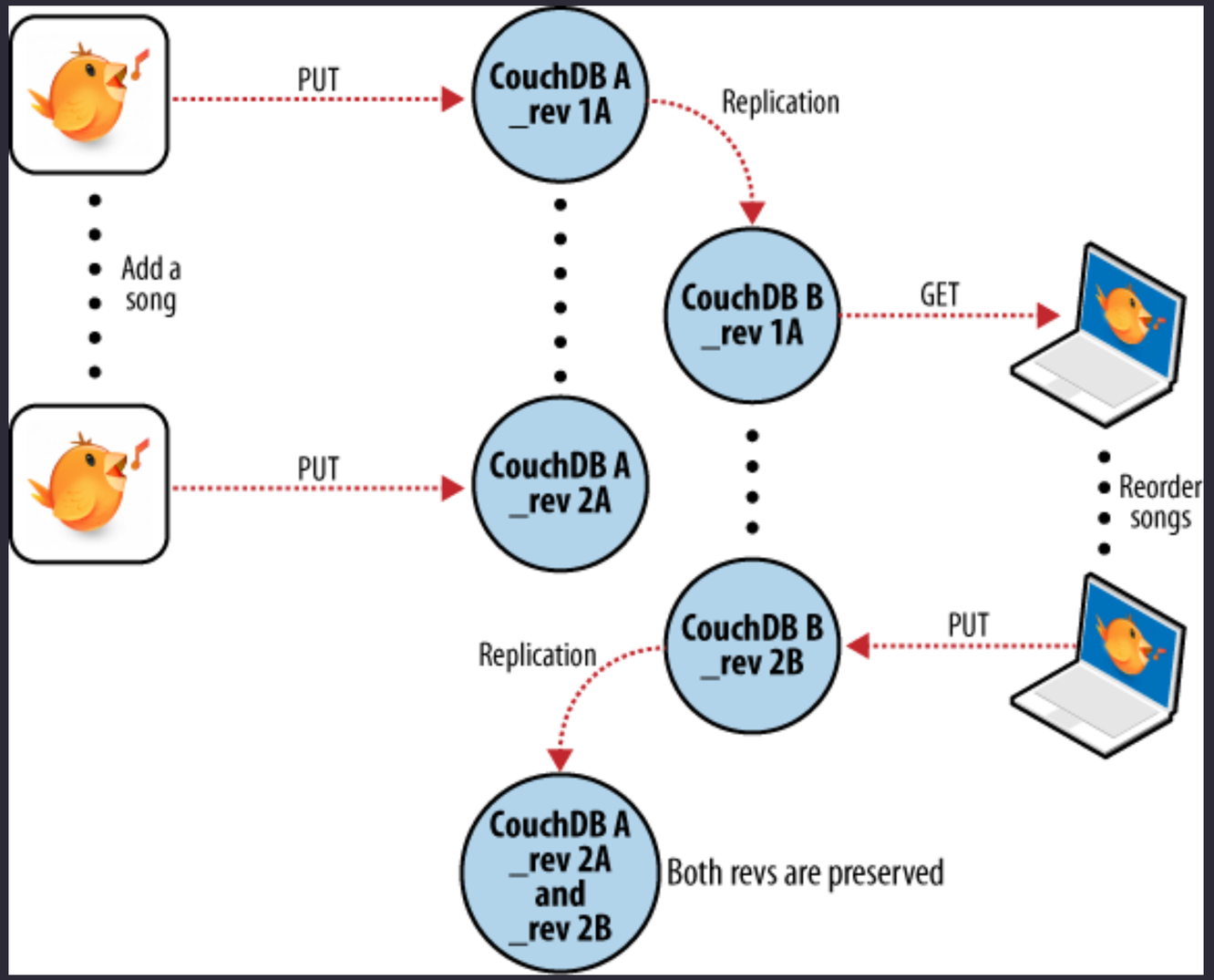
CouchDB

- Document Database written in Erlang
- Released 2005
- Multimaster
- Lock - free
- MapReduce Based Views

Multiversion Concurrency Control



Multiversion Concurrency Control



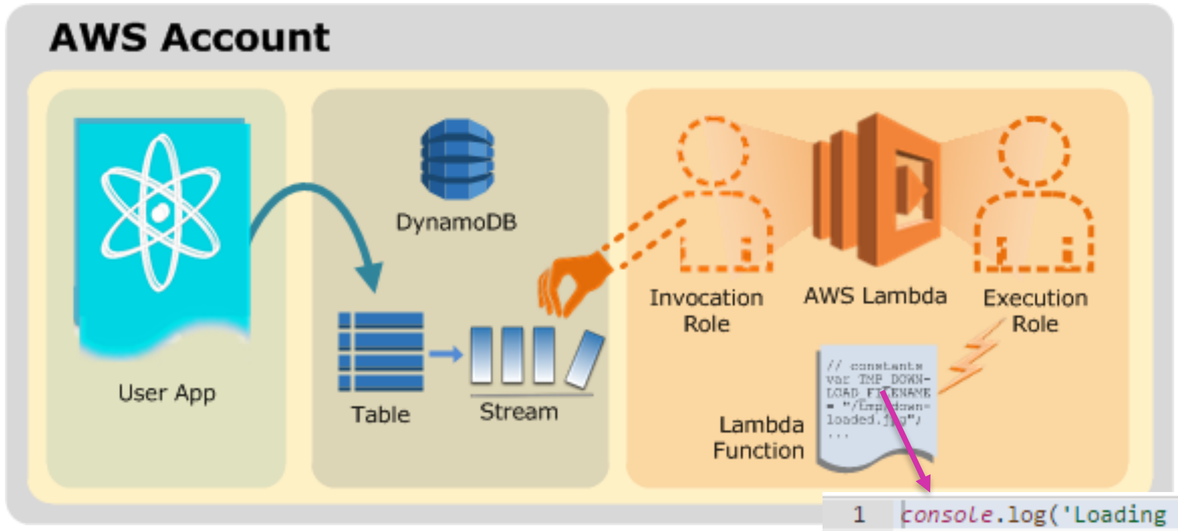
DynamoDB

DynamoDB

- Key-Value and Document Database by Amazon
- Technically a ‘Wide Column Store’ similar to BigTable
- Evolved from Dynamo

DynamoDB Streams & Lambda

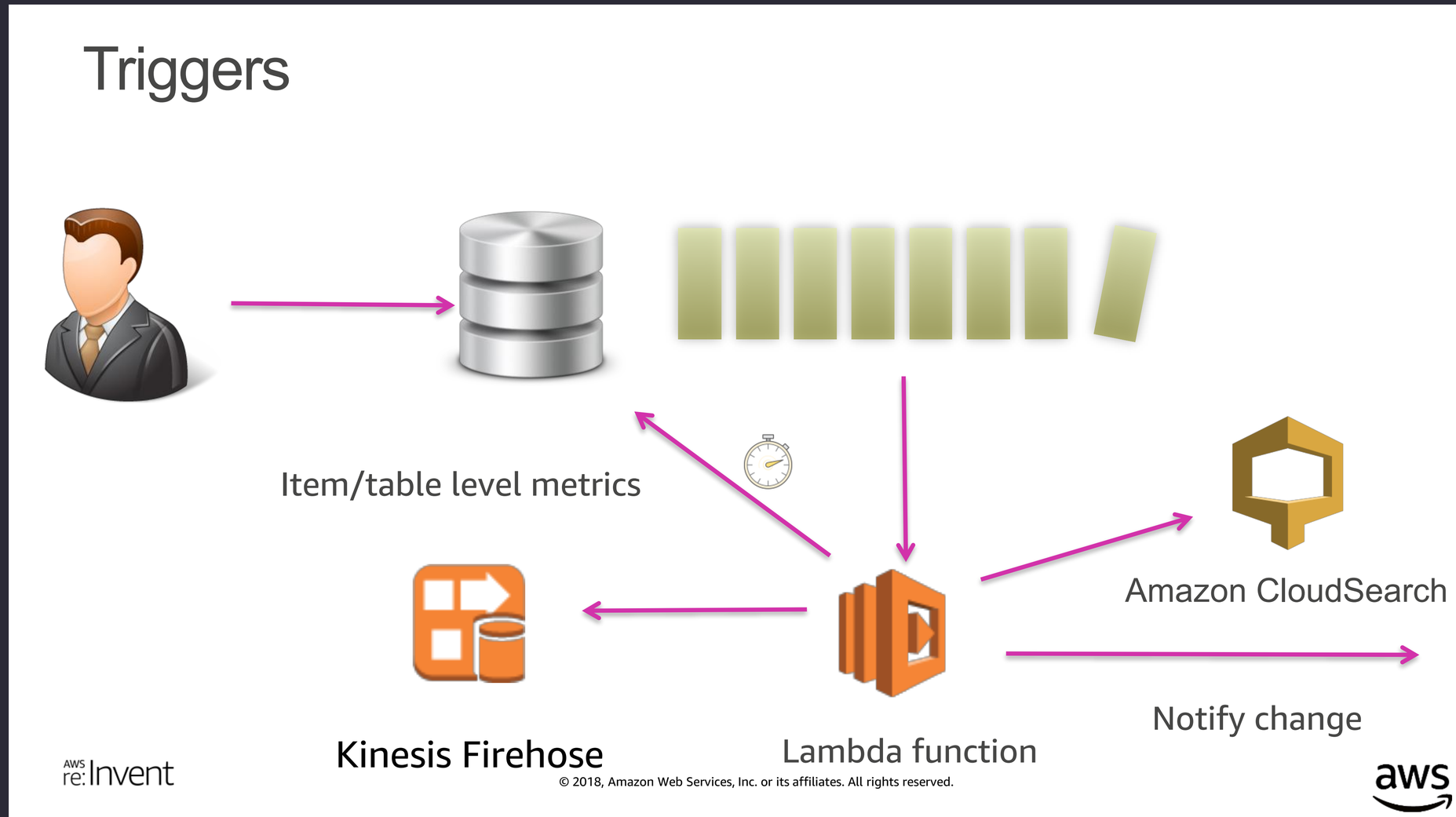
DynamoDB Streams and AWS Lambda




```
1  console.log('Loading event');
2  exports.handler = function(event, context) {
3    console.log("Event: %j", event);
4    for(i = 0; i < event.Records.length; ++i) {
5      record = event.Records[i];
6      console.log(record.EventID);
7      console.log(record.EventName);
8      console.log("DynamoDB Record: %j", record.Dynamodb);
9    }
10   context.done(null, "Hello World");  // SUCCESS with message
11 }
```

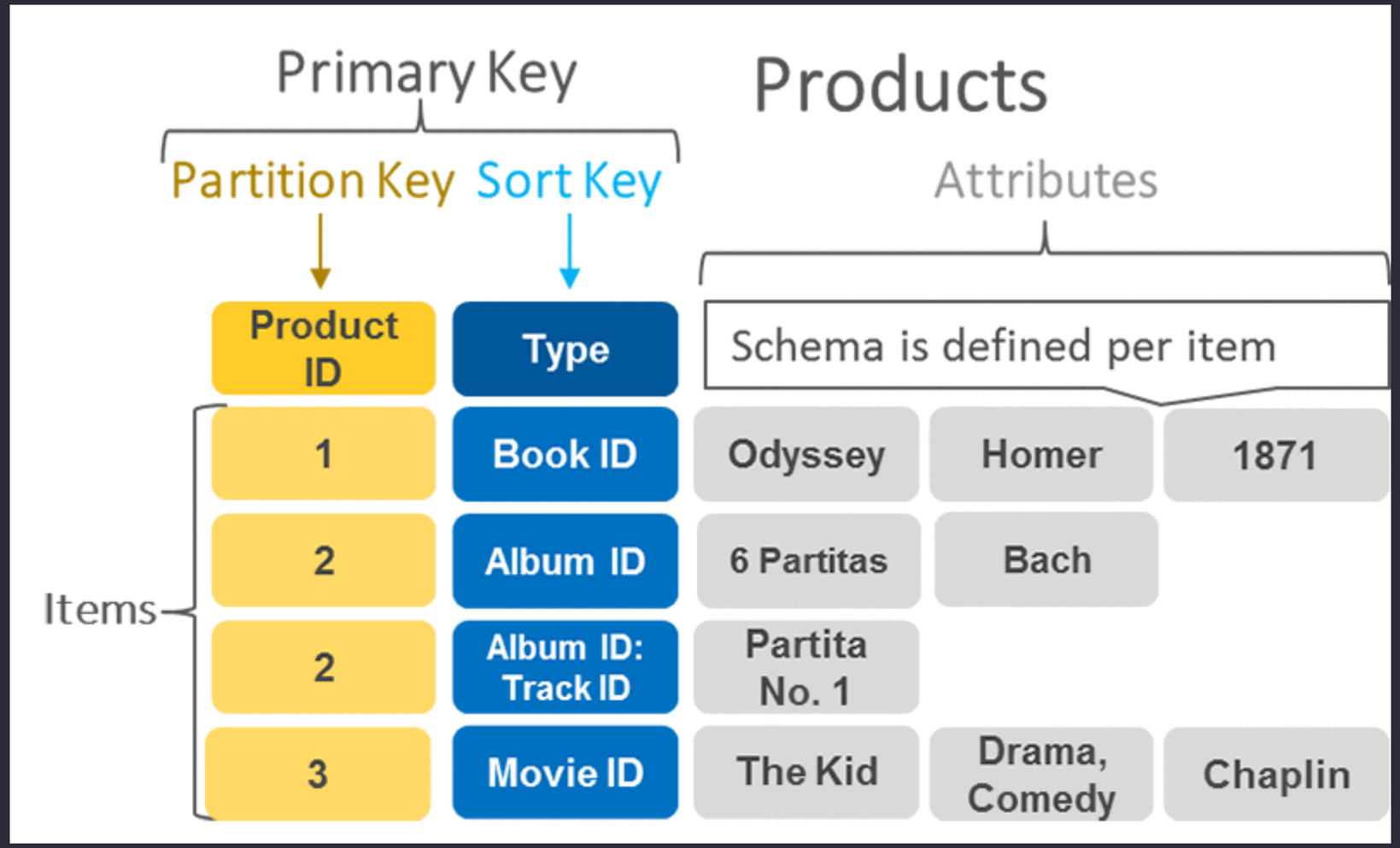
```
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff INSERT
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff DynamoDB Record:{ "NewImage": { "name": { "S": "sivar" }, "hk": { "S": "3" } }, "SizeBytes": 15, "StreamViewType": "NEW_AND_OLD_IMAG
2015-03-21T07:44:58.883Z 2ca3769a-cf9e-11e4-b270-ad4d24b312ff Message: "Hello World"
```

Triggers



Data Modeling in NoSQL

Primary and Sort Keys



Composite Key Modeling

Multi-value Sorts and Filters

Bob

Partition key

Sort key

Secondary index

Opponent	Date	GameId	Status	Host
Alice	2014-10-02	d9bl3	DONE	David
Carol	2014-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2014-09-30	72f49	PENDING	Alice
Bob	2014-10-03	b932s	PENDING	Carol
Bob	2014-10-03	ef9ca	IN_PROGRESS	David

Approach 1: Query Filter

```
SELECT * FROM Game
WHERE Opponent='Bob'
ORDER BY Date DESC
FILTER ON Status='PENDING'
```

Bob

Secondary Index

Opponent	Date	GameId	Status	Host
Alice	2014-10-02	d9bl3	DONE	David
Carol	2014-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2014-09-30	72f49	PENDING	Alice
Bob	2014-10-03	b932s	PENDING	Carol
Bob	2014-10-03	ef9ca	IN_PROGRESS	David

(filtered out)

Approach 2: Composite Key

Status
DONE
IN_PROGRESS
IN_PROGRESS
PENDING
PENDING

+

Date
2014-10-02
2014-10-08
2014-10-03
2014-10-03
2014-09-30

=

StatusDate
DONE_2014-10-02
IN_PROGRESS_2014-10-08
IN_PROGRESS_2014-10-03
PENDING_2014-09-30
PENDING_2014-10-03

Approach 2: Composite Key

Partition key

Sort key

Opponent	StatusDate	GameId	Host
Alice	DONE_2014-10-02	d9bl3	David
Carol	IN_PROGRESS_2014-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2014-10-03	ef9ca	David
Bob	PENDING_2014-09-30	72f49	Alice
Bob	PENDING_2014-10-03	b932s	Carol

Approach 2: Composite Key

```
SELECT * FROM Game
WHERE Opponent='Bob'
    AND StatusDate BEGINS_WITH 'PENDING'
```

Bob

Secondary index

Opponent	StatusDate	GameId	Host
Alice	DONE_2014-10-02	d9bl3	David
Carol	IN_PROGRESS_2014-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2014-10-03	ef9ca	David
Bob	PENDING_2014-09-30	72f49	Alice
Bob	PENDING_2014-10-03	b932s	Carol

Version History In DynamoDB

Maintaining Version History

ItemID (PK)	Version (SK)	CurVer	Attrs
1	v0	2	…
	v1	…	
	v2	…	
	v3	…	

(Many more item partitions)

Transaction

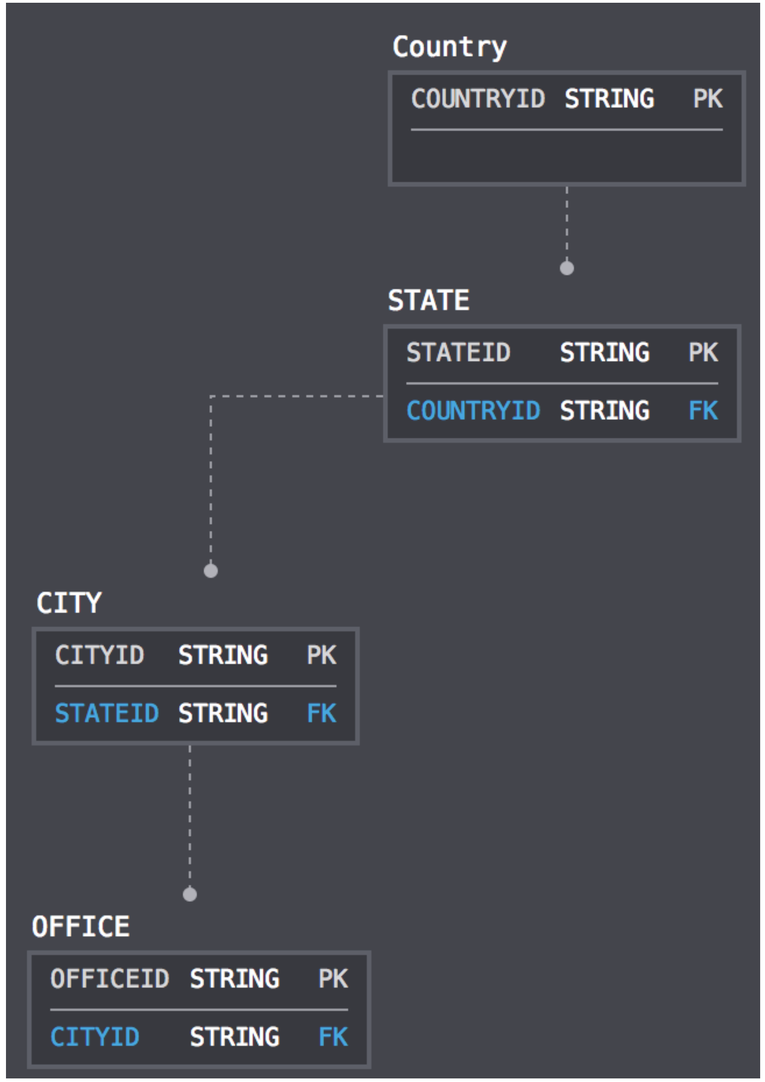
```
COPY Item.v0 -> Item1.v3 IF Item.v3 == NULL
UPDATE Item1.v3 SET Attr1 += 1
UPDATE Item1.v3 SET Attr2 = …
UPDATE Item1.v3 SET Attr3 = …
COPY Item1.v3 -> Item1.v0 SET CurVer = 3
```

Overwrite v0 Item to Commit changes

Item versions

Modeling Data Hierarchies

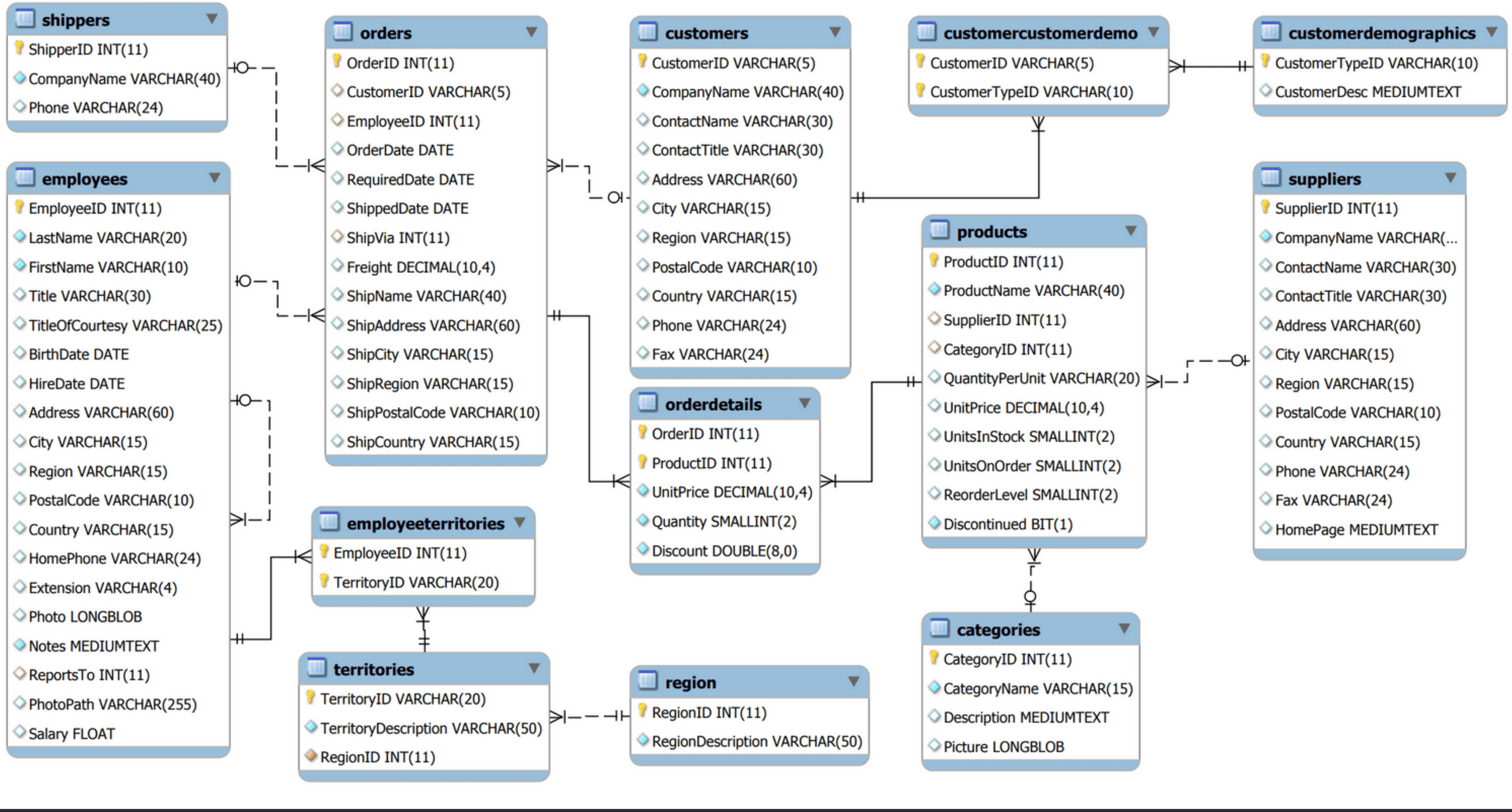
Hierarchical Data Structures as Items



- Use composite sort key to define a hierarchy
- Highly selective queries with sort conditions
- Reduce query complexity

Primary Key		Attributes		
COUNTRY	LOCATION			
USA	NY#NYC#JFK11	Address	EmployeeCount	BuildingManager
	NY#NYC#JFK14	Address	EmployeeCount	BuildingManager
	WA#SEA#BLACKFOOT	Address	EmployeeCount	BuildingManager
	WA#SEA#KUMO	Address	EmployeeCount	BuildingManager
	WA#SEA#MAYDAY	Address	EmployeeCount	BuildingManager

Modeling Relational Data


shippers
ShipperID INT(11)
CompanyName VARCHAR(40)
Phone VARCHAR(24)
orders
OrderID INT(11)
CustomerID VARCHAR(5)
EmployeeID INT(11)
OrderDate DATE
RequiredDate DATE
ShippedDate DATE
ShipVia INT(11)
Freight DECIMAL(10,4)
ShipName VARCHAR(40)
ShipAddress VARCHAR(60)
ShipCity VARCHAR(15)
ShipRegion VARCHAR(15)
ShipPostalCode VARCHAR(10)
ShipCountry VARCHAR(15)
customers
CustomerID VARCHAR(5)
CompanyName VARCHAR(40)
ContactName VARCHAR(30)
ContactTitle VARCHAR(30)
Address VARCHAR(60)
City VARCHAR(15)
Region VARCHAR(15)
PostalCode VARCHAR(10)
Country VARCHAR(15)
Phone VARCHAR(24)
Fax VARCHAR(24)
customercustomerdemo
CustomerID VARCHAR(5)
CustomerTypeID VARCHAR(10)
customerdemographics
CustomerTypeID VARCHAR(10)
CustomerDesc MEDIUMTEXT
employees
EmployeeID INT(11)
LastName VARCHAR(20)
FirstName VARCHAR(10)
Title VARCHAR(30)
TitleOfCourtesy VARCHAR(25)
BirthDate DATE
HireDate DATE
Address VARCHAR(60)
City VARCHAR(15)
Region VARCHAR(15)
PostalCode VARCHAR(10)
Country VARCHAR(15)
HomePhone VARCHAR(24)
Extension VARCHAR(4)
Photo LONGBLOB
Notes MEDIUMTEXT
ReportsTo INT(11)
PhotoPath VARCHAR(255)
Salary FLOAT
orderdetails
OrderID INT(11)
ProductID INT(11)
UnitPrice DECIMAL(10,4)
Quantity SMALLINT(2)
Discount DOUBLE(8,0)
products
ProductID INT(11)
ProductName VARCHAR(40)
SupplierID INT(11)
CategoryID INT(11)
QuantityPerUnit VARCHAR(20)
UnitPrice DECIMAL(10,4)
UnitsInStock SMALLINT(2)
UnitsOnOrder SMALLINT(2)
ReorderLevel SMALLINT(2)
Discontinued BIT(1)
suppliers
SupplierID INT(11)
CompanyName VARCHAR(...
ContactName VARCHAR(30)
ContactTitle VARCHAR(30)
Address VARCHAR(60)
City VARCHAR(15)
Region VARCHAR(15)
PostalCode VARCHAR(10)
Country VARCHAR(15)
Phone VARCHAR(24)
Fax VARCHAR(24)
HomePage MEDIUMTEXT
employeeterritories
EmployeeID INT(11)
TerritoryID VARCHAR(20)
territories
TerritoryID VARCHAR(20)
TerritoryDescription VARCHAR(50)
RegionID INT(11)
region
RegionID INT(11)
RegionDescription VARCHAR(50)
categories
CategoryID INT(11)
CategoryName VARCHAR(15)
Description MEDIUMTEXT
Picture LONGBLOB

Modeling Relational Data

- Attribute names have no relationship to attribute values.
- Index Overloading

What access patterns?

- Get employee by employee ID
- Get direct reports for an employee
- Get discontinued products
- List all orders of a given product
- Get the most recent 25 orders
- Get shippers by name
- Get customers by contact name
- List all products included in an order
- Get suppliers by country and region

Create your table

pk	sk	data	hire_date	sales_rep
PRODUCT_ID	USA	Product Name		
ORDER_ID	PRODUCT_ID	CATEGORY_ID		SALES_REP_ID
EMPLOYEE_ID	MANAGER_ID	Beth Smith	07/06/05	

Create a row for each entity

	pk	sk (GSI PK)	data (GSI SK)	*additional attributes...*
Order	orderID	"ORDER"	orderDate	
Product	productID	"PRODUCT"	discontinued	
Employee	employeeID	reportsTo	hireDate	
Category	categoryID	categoryName	description	
Shipper	shipperID	companyName	phone	
Supplier	supplierID	"SUPPLIER"	country#region#city#address	
Customer	customerID	contactName	country#region#city#address	

Create many to many relationships

	pk	sk (GS1 PK)	data	...*additional attributes*...
"entity" order record	orderID	"ORDER"	customerID	etc
NEW -- "OrderDetails" record for product 1	orderID	productID_1	unitPrice	quantity, discount
NEW -- "OrderDetails" record for product 2	orderID	productID_2	unitPrice	quantity, discount
"entity" product 1 record	productID_1	"PRODUCT"	discontinued	etc
"entity" product 2 record	productID_2	"PRODUCT"	discontinued	etc

All of that NoSQL data modeling is hard. What if I could get the benefits of NoSQL without having to design complex NoSQL data models?

NewSQL

CA systems at Scale

CockroachDB

CockroachDB

- World-Scale SQL Database
- SQL (talks Postgres on the wire)
- High Availability, High Consistency

CockroachDB Architecture

Layer	Order	Purpose
SQL	1	Translate client SQL queries to KV operations.
Transactional	2	Allow atomic changes to multiple KV entries.
Distribution	3	Present replicated KV ranges as a single entity.
Replication	4	Consistently and synchronously replicate KV ranges across many nodes.
Storage	5	Write and read KV data on disk.

CockroachDB Architecture

Data mapping between SQL and KV

Key	Values
`/system/databases/mydb/id`	51
`/system/tables/customer/id`	42
`/system/desc/51/42/address`	69
`/system/desc/51/42/url`	66

Key	Values
`/51/42/Apple/69`	`1 Infinite Loop, Cupertino, CA`
`/51/42/Apple/66`	`http://apple.com/`

Cloud Spanner

The Promise

	CLOUD SPANNER	TRADITIONAL RELATIONAL	TRADITIONAL NON-RELATIONAL
Schema	✓ **Yes**	✓ Yes	✗ No
SQL	✓ **Yes**	✓ Yes	✗ No
Consistency	✓ **Strong**	✓ Strong	✗ Eventual
Availability	✓ **High**	✗ Failover	✓ High
Scalability	✓ **Horizontal**	✗ Vertical	✓ Horizontal
Replication	✓ **Automatic**	↻ Configurable	↻ Configurable

But How?

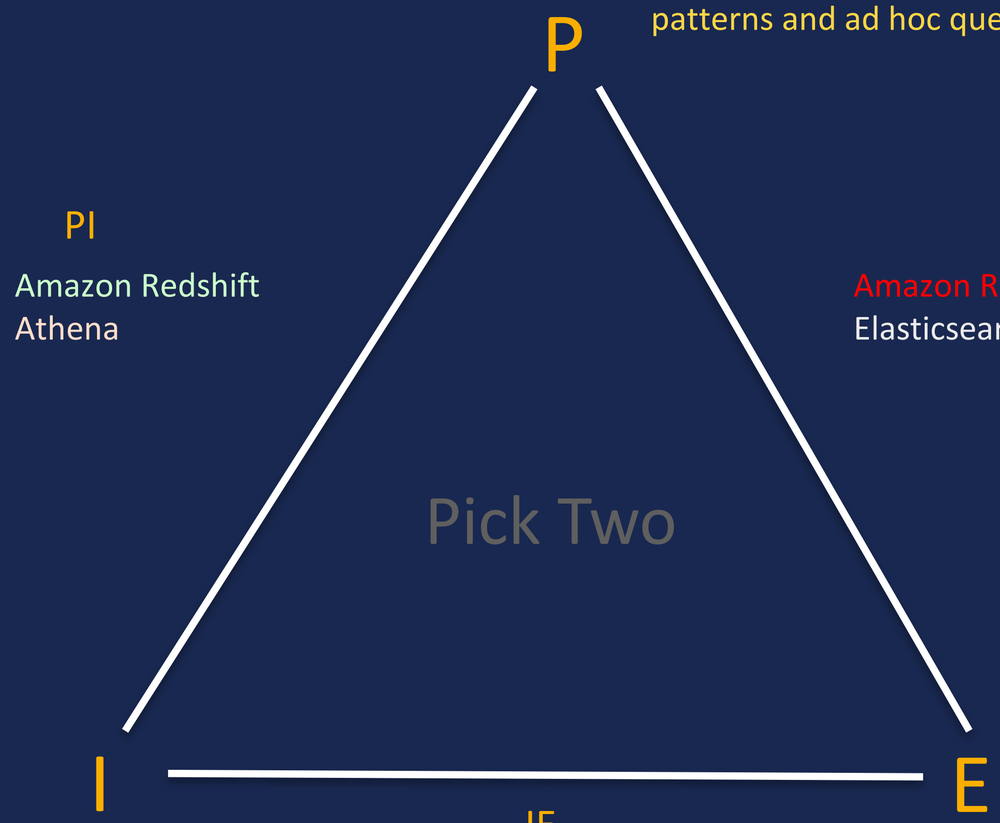
How Spanner ‘Breaks’ CAP

- Spanner is a distributed CA system
- Uses TrueTime for MVCC
- Google’s network prevents the need to optimize for Partitions

How Does CockroachDB do it?

- While Spanner always waits after writes, CockroachDB sometimes waits before reads.
- Hybrid Logical Clocks

The Iron Triangle of Purpose (The PIE Theorem)



Pattern Flexibility:
The database supports random access patterns and ad hoc queries

Data Models:
Relational
Wide Column
Document
Graph
Columnar
Unstructured

PI

Amazon Redshift
Athena

PE

Amazon RDS
Elasticsearch
Aurora Serverless
Neptune

Infinite Scale:
The database can gracefully increase size and throughput without practical limits

IE

Amazon DynamoDB

Efficiency:
The database will deliver required query latency for the workload at all times

Questions?