From SQL to NoSQL

Best Practices for Migrating from RDBMS to DynamoDB

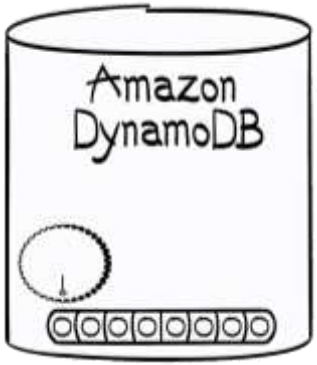
Rick Houlihan, Principal TPM – DBS NoSQL

26 July 2016

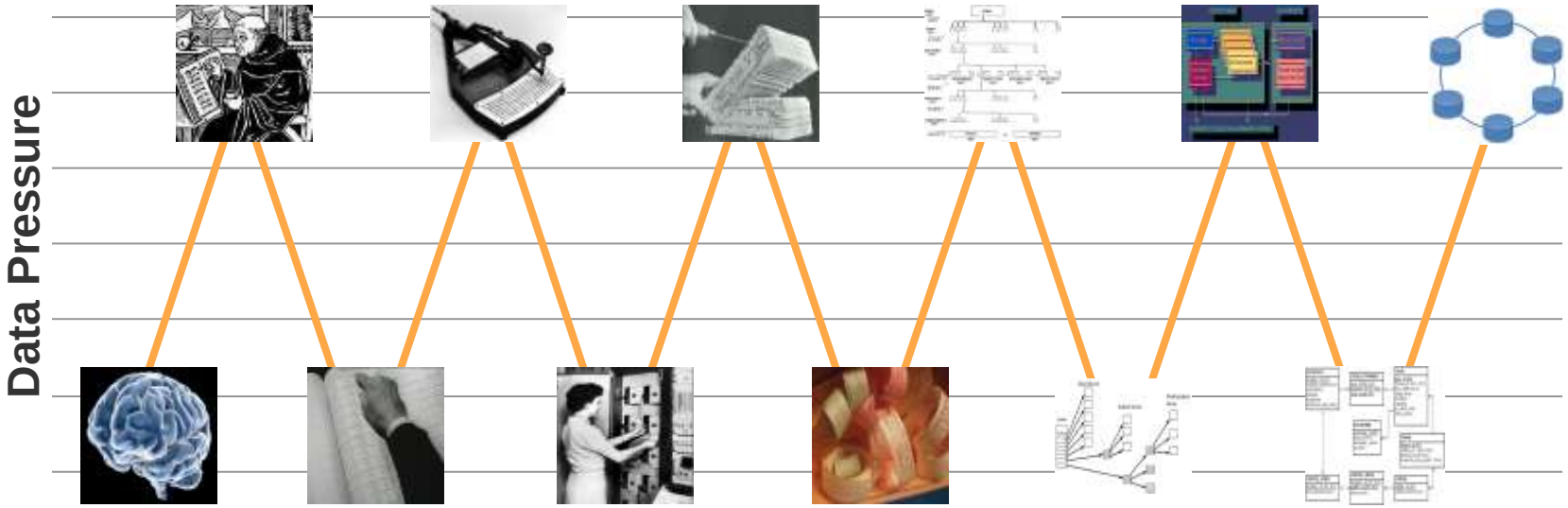
amazon web services | Webinars

Agenda

- Evolution of Data Processing
- Why NoSQL
- Key DynamoDB concepts
- SQL to NoSQL Data Modeling

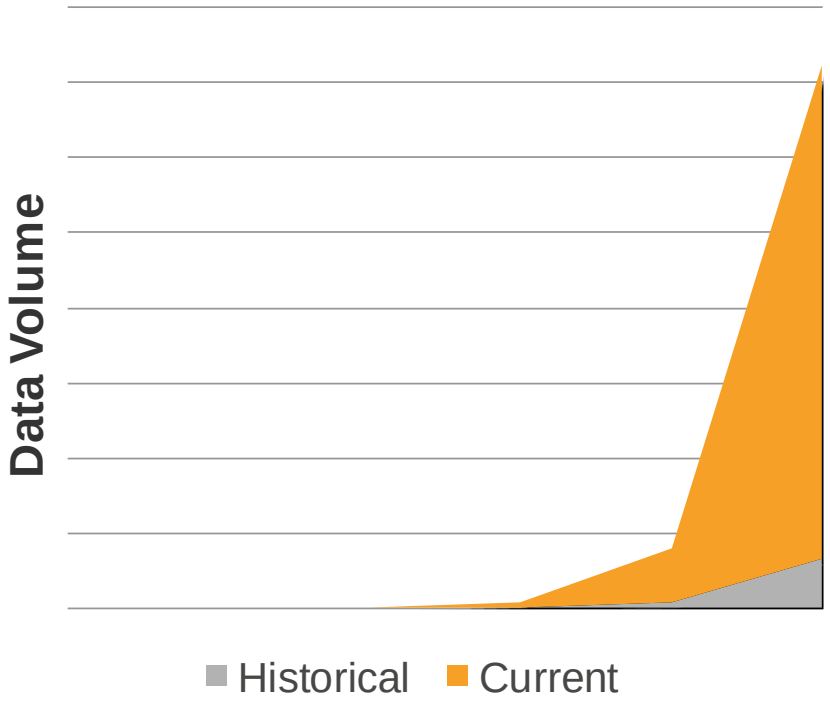


Timeline of Database Technology

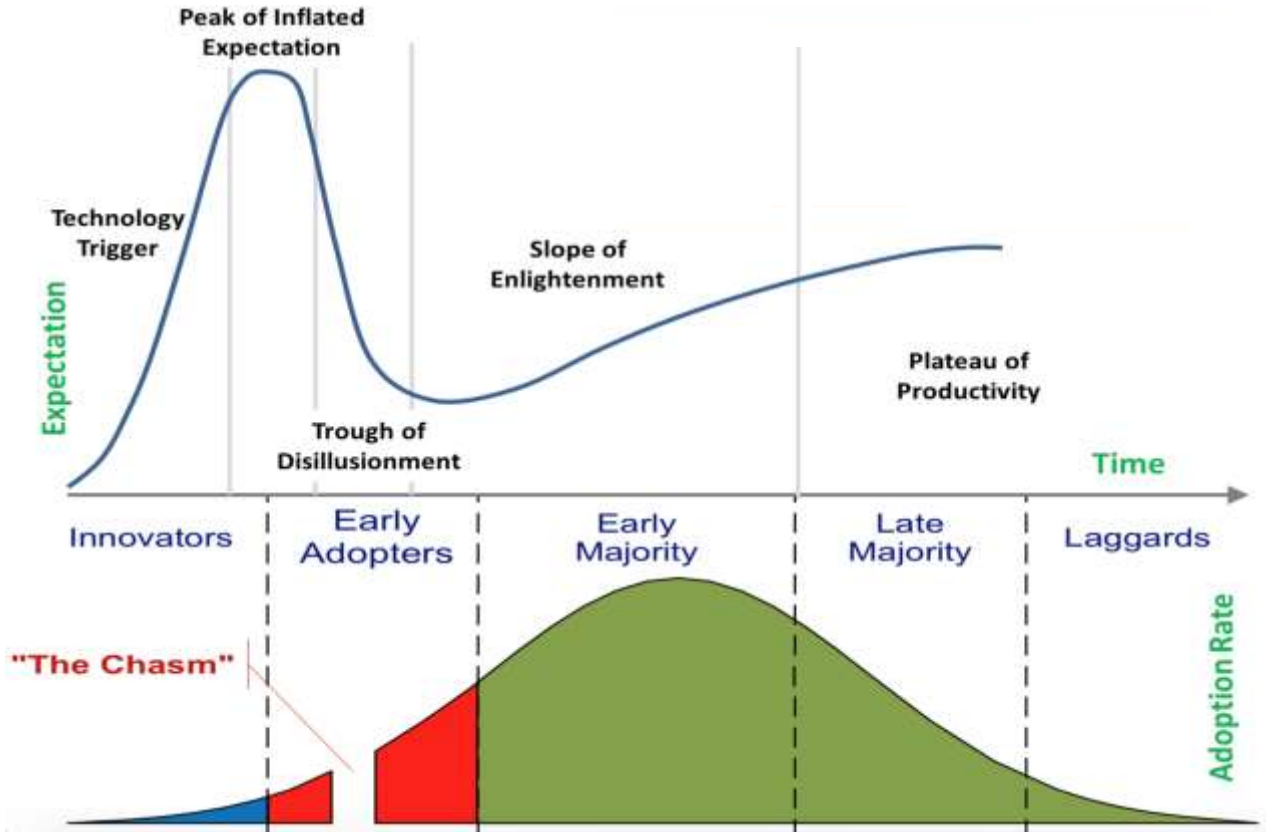


Data Volume Since 2010

- 90% of stored data generated in last 2 years
- 1 Terabyte of data in 2010 equals 6.5 Petabytes today
- Linear correlation between data pressure and technical innovation
- SQL is not built for this



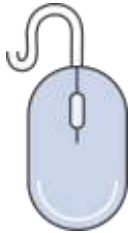
Technology Adoption and the Hype Curve



Why NoSQL?

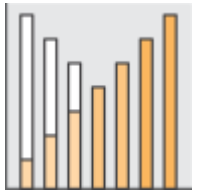
SQL	NoSQL
Optimized for storage	**Optimized for compute**
Normalized/relational	Denormalized/hierarchical
Ad hoc queries	Instantiated views
Scale vertically	Scale horizontally
Good for OLAP	Built for OLTP at scale

Amazon DynamoDB

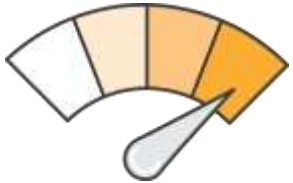


Fully Managed NoSQL

Document or Key-Value



Scales to Any Workload



Fast and Consistent

Access Control

Event Driven Programming

Table

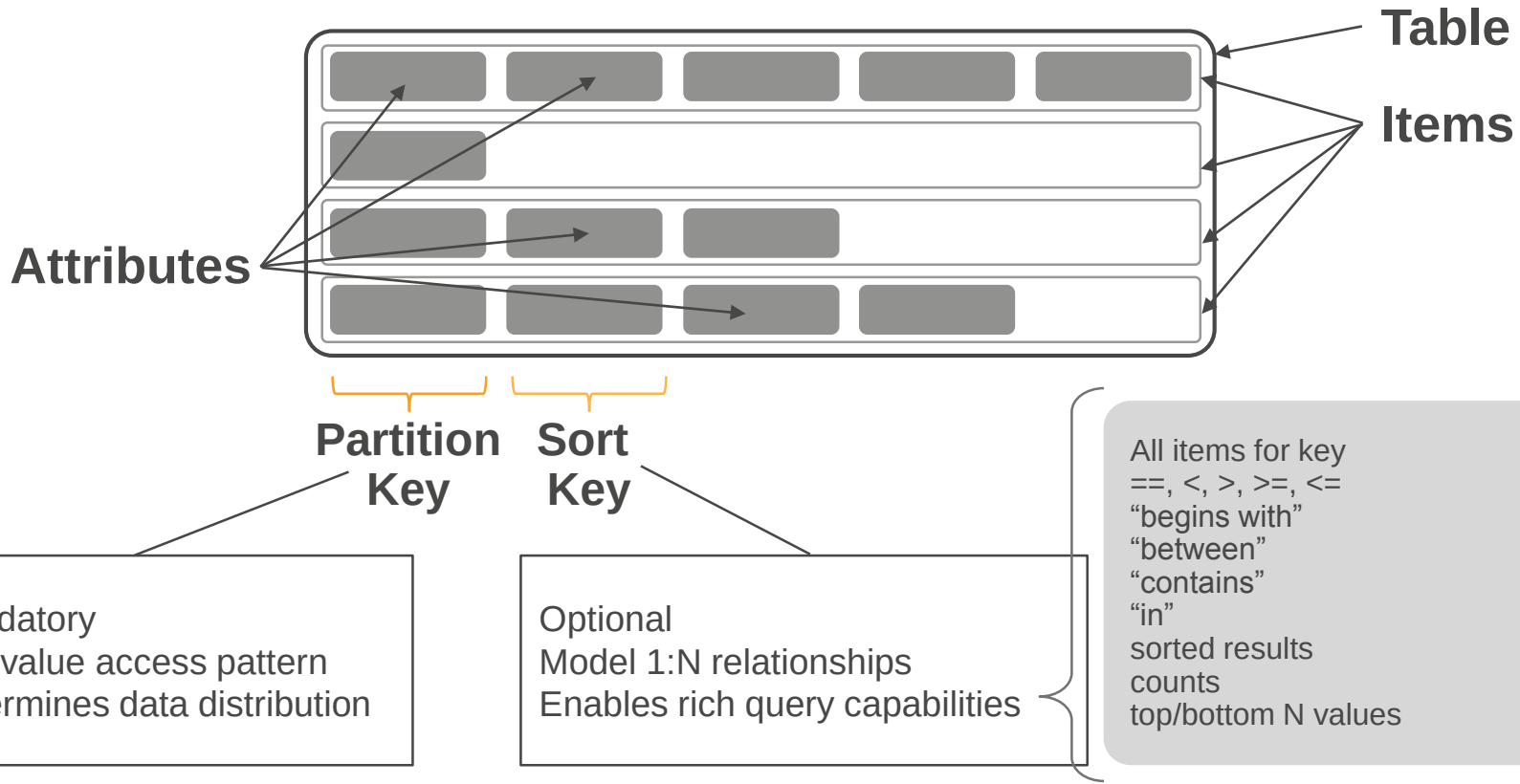


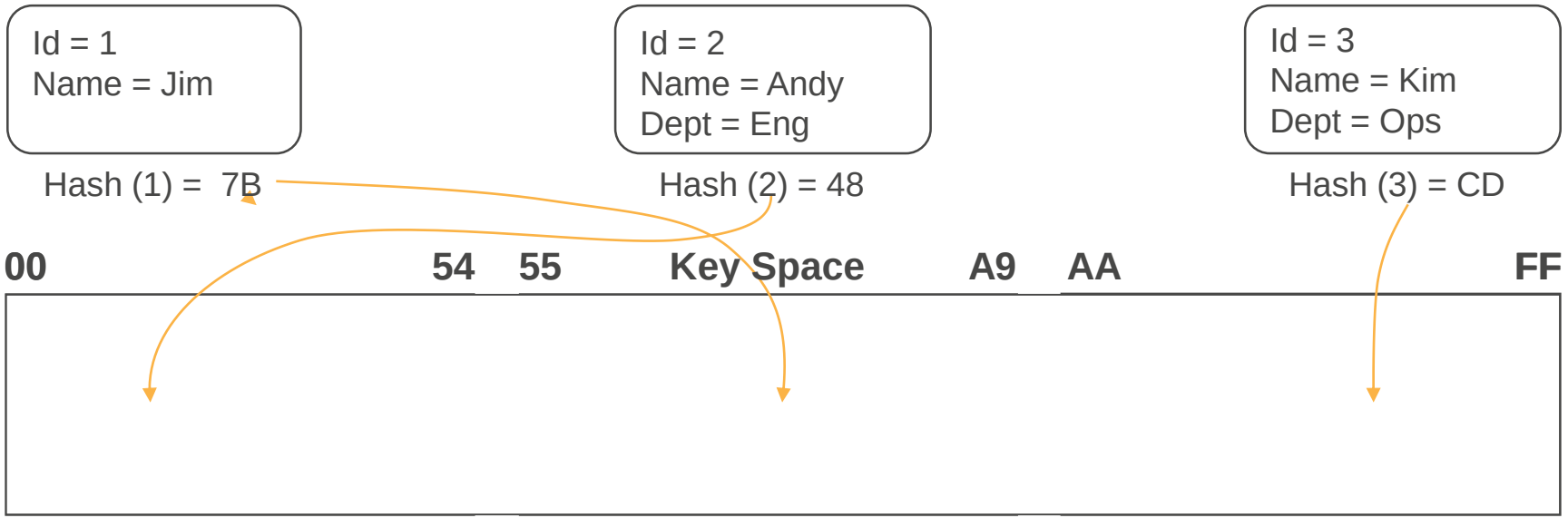
Table
Items
Attributes
Partition Key
Sort Key
Mandatory
Key-value access pattern
Determines data distribution
Optional
Model 1:N relationships
Enables rich query capabilities
All items for key
==, <, >, >=, <=
“begins with”
“between”
“contains”
“in”
sorted results
counts
top/bottom N values

Partition Keys

Partition Key uniquely identifies an item

Partition Key is used for building an unordered hash index

Allows table to be partitioned for scale



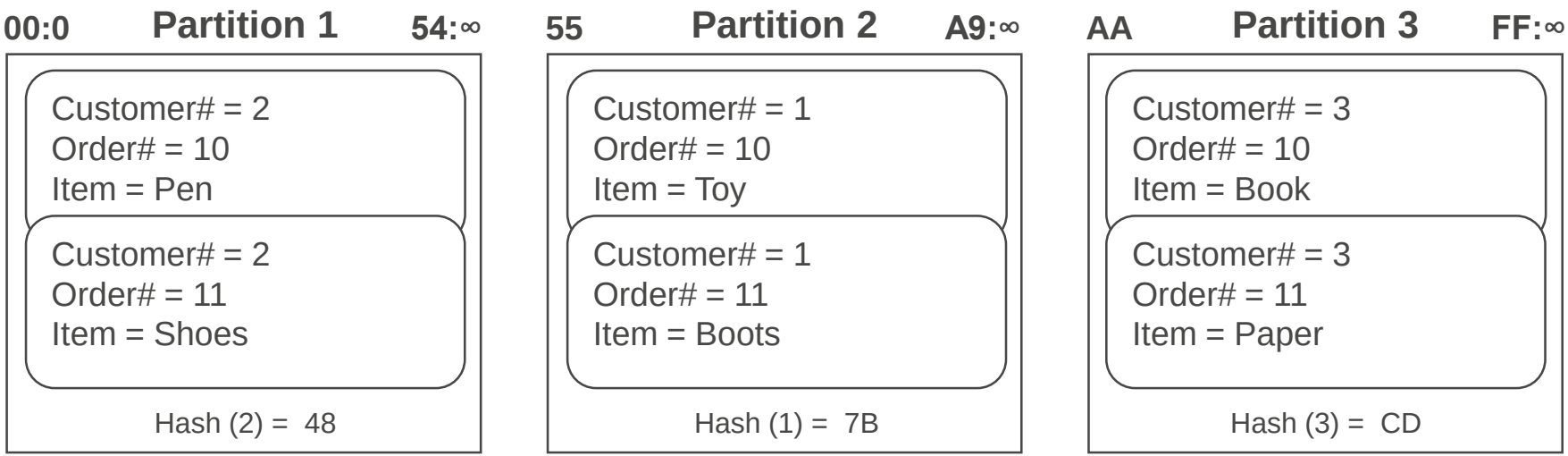
Partition:Sort Key

Partition:Sort Key uses two attributes together to uniquely identify an Item

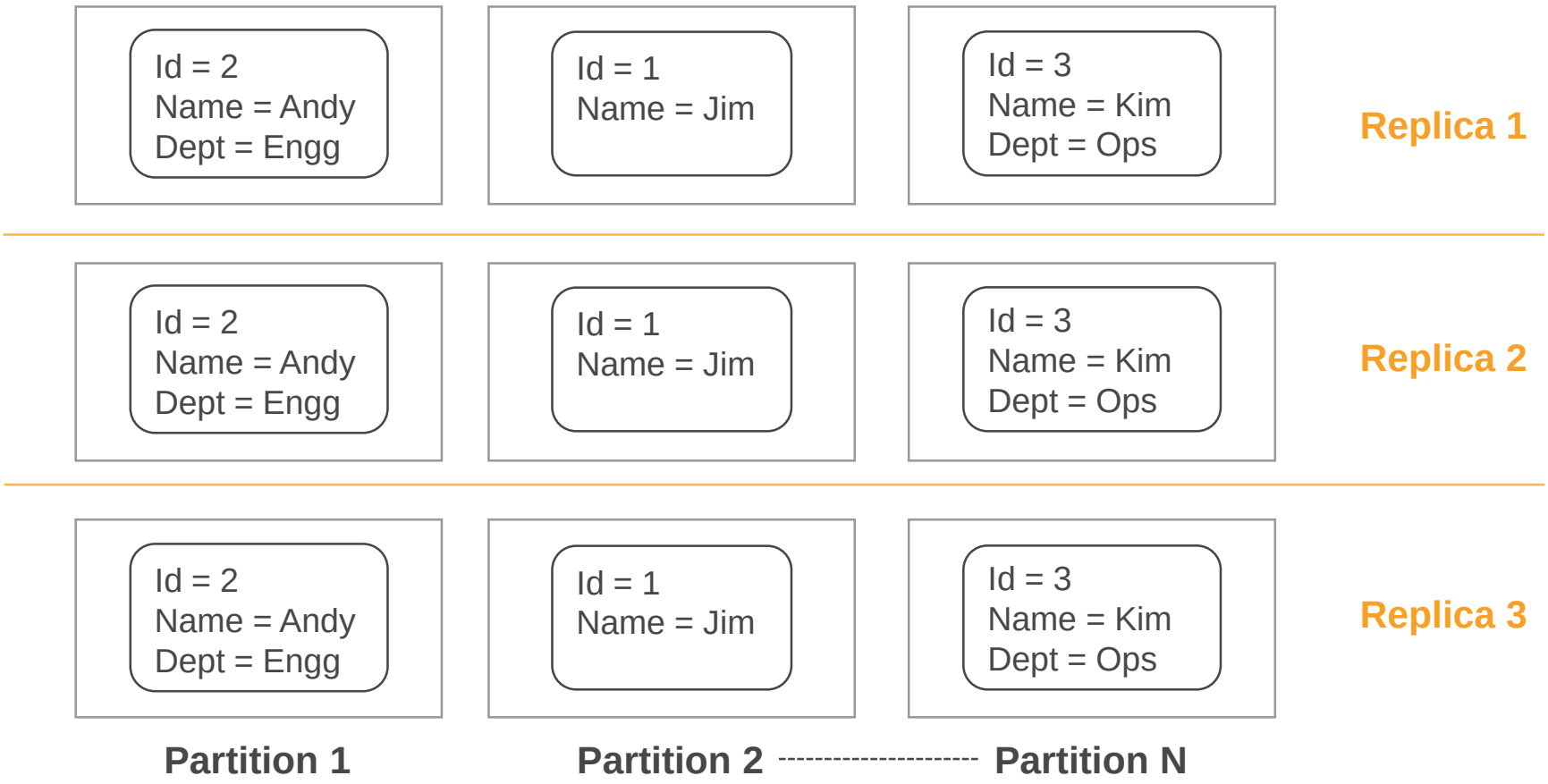
Within unordered hash index, data is arranged by the sort key

No limit on the number of items (∞) per partition key

- Except if you have local secondary indexes



Partitions are three-way replicated

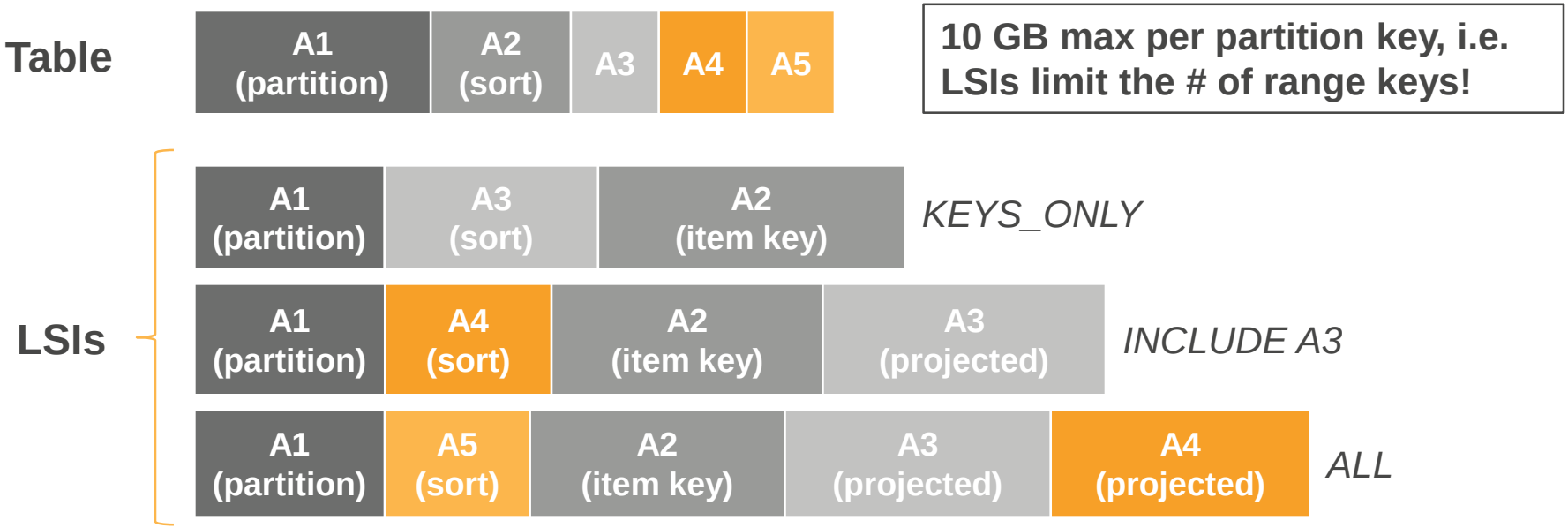


Indexes

Local secondary index (LSI)

Alternate sort key attribute

Index is local to a partition key



Global secondary index (GSI)

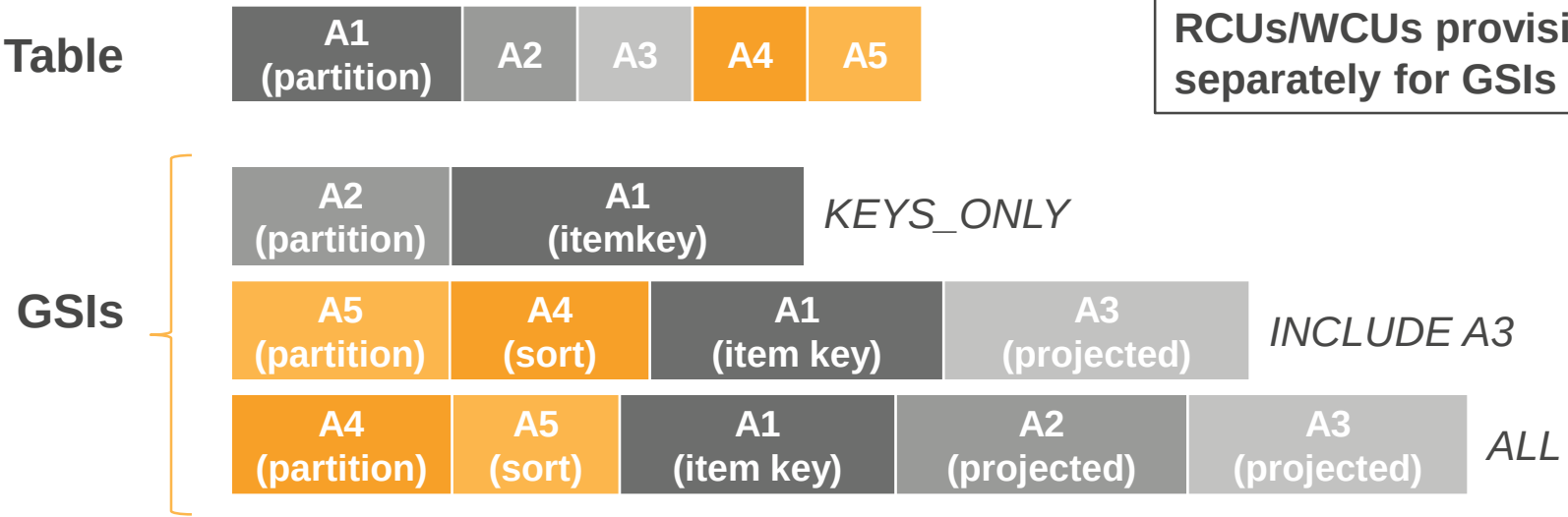
Online indexing

Alternate partition and/or sort key

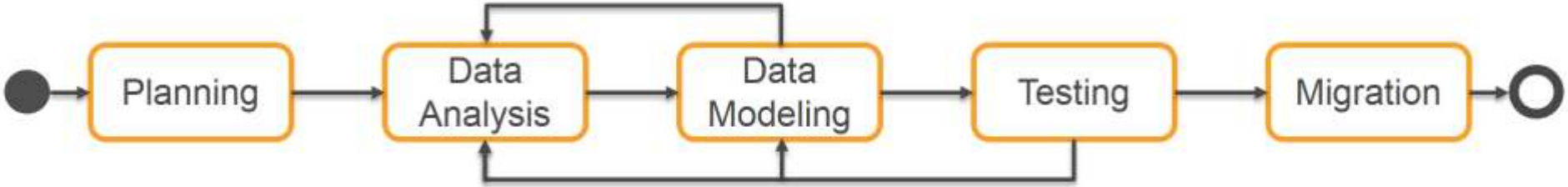
Index is across all partition keys

Use composite sort keys for compound indexes

RCUs/WCUs provisioned separately for GSIs



Migrating to NoSQL



Data Analysis Phase

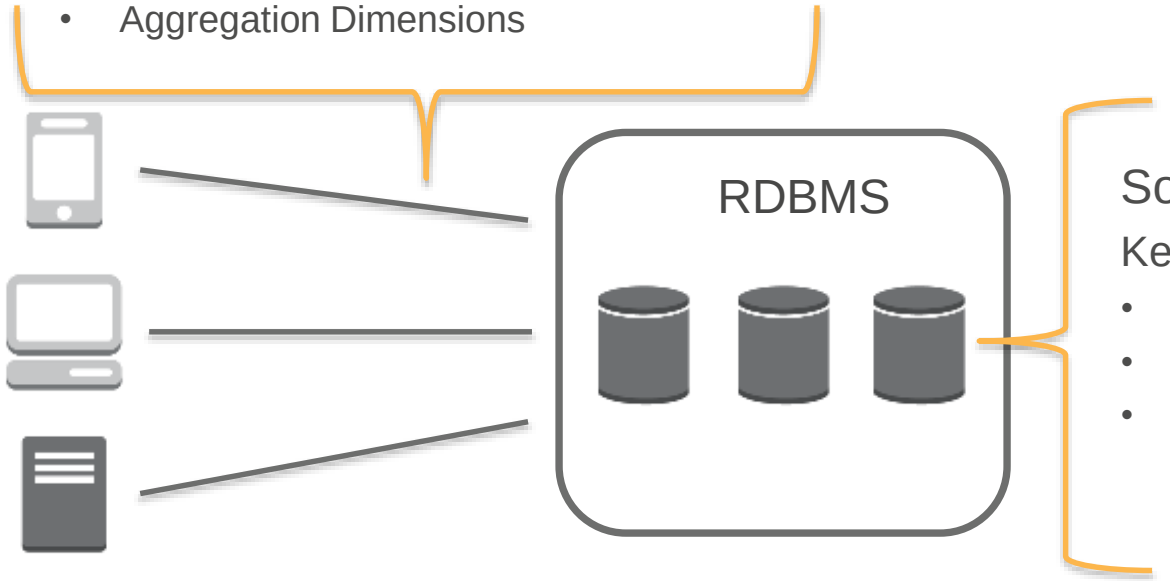
Analysis of both the RDBMS schema as well as application access patterns is key to understanding the performance of workloads on a NoSQL database platform.

Data Analysis Phase

Access Pattern of the Application

Examples:

- Write/Read Workloads
- Relational Structures
- Aggregation Dimensions



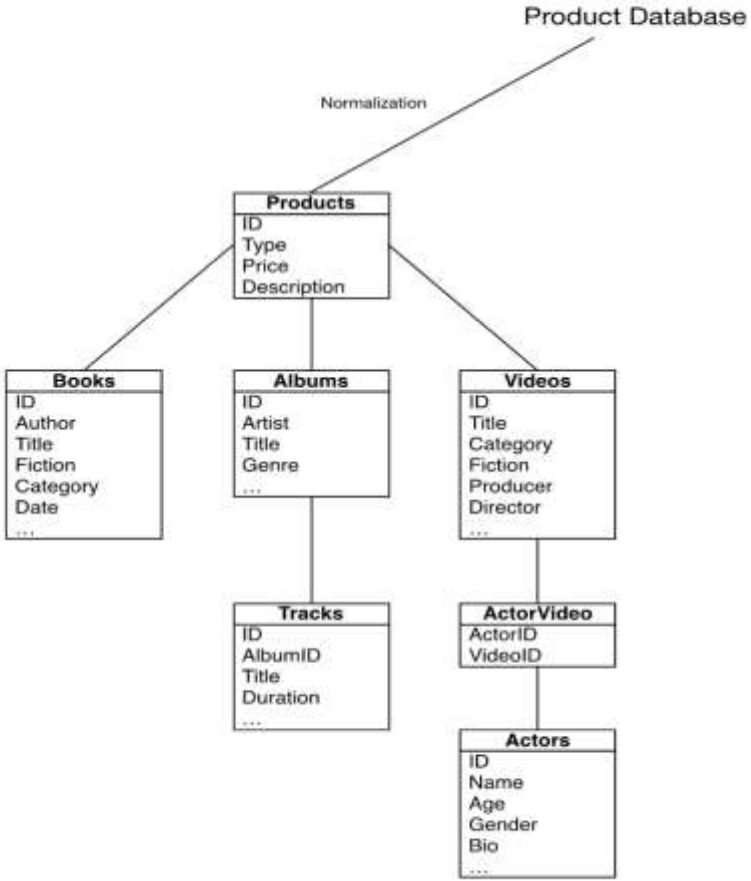
Source Data Analysis

Key data attributes

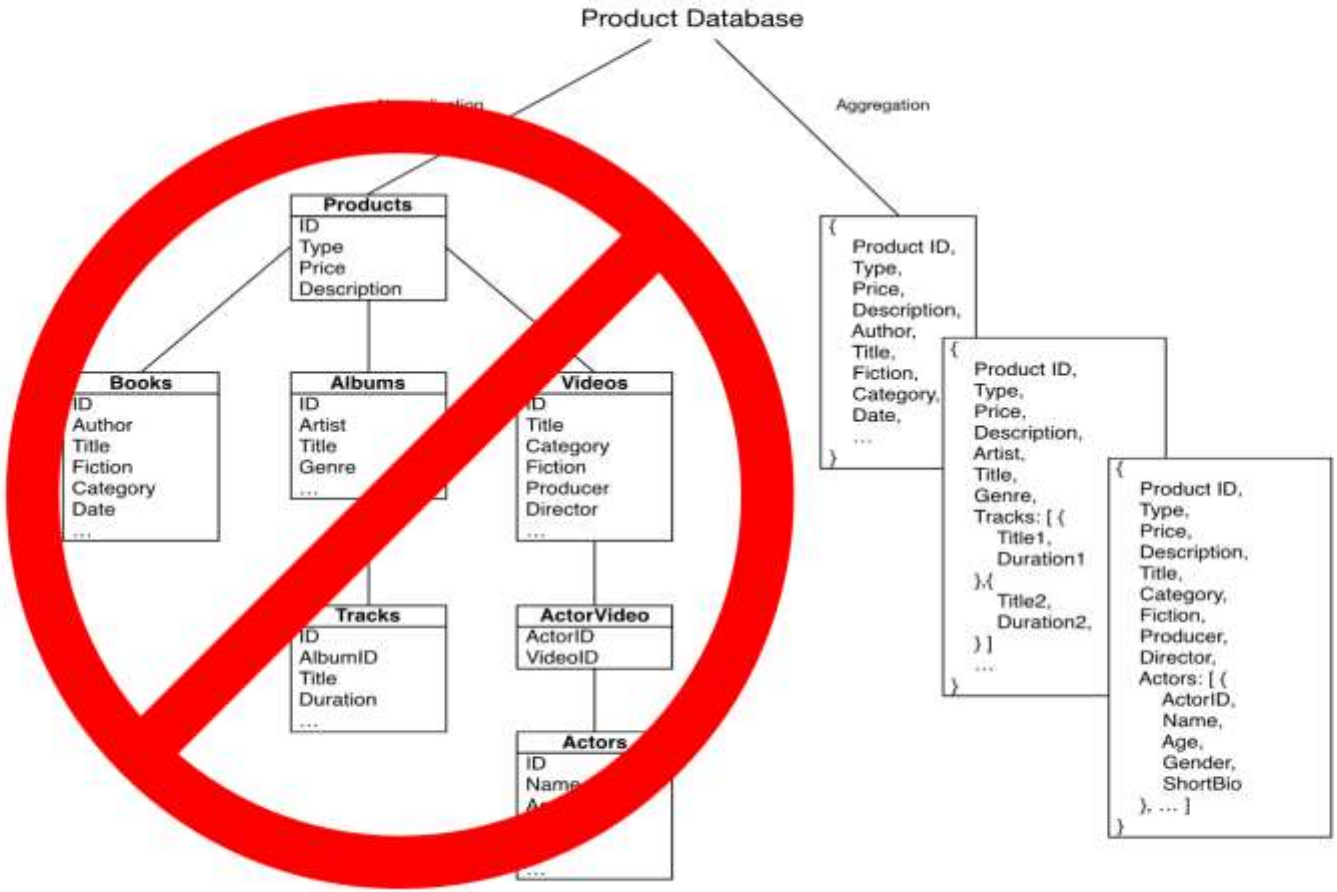
- Read/Write Velocity
- Data Partitioning
- Key Cardinality

Data Modeling

SQL vs. NoSQL Access Pattern



SQL vs. NoSQL Access Pattern



1:1 relationships or key-values

Use a table or GSI with an alternate partition key

Use GetItem or BatchGetItem API

Example: Given an SSN or license number, get attributes

Users Table	
Partition key	Attributes
SSN = 123-45-6789	Email = johndoe@nowhere.com, License = TDL25478134
SSN = 987-65-4321	Email = maryfowler@somewhere.com, License = TDL78309234

Users-License-GSI	
Partition key	Attributes
License = TDL78309234	Email = maryfowler@somewhere.com, SSN = 987-65-4321
License = TDL25478134	Email = johndoe@nowhere.com, SSN = 123-45-6789

1:N relationships or parent-children

Use a table or GSI with partition and sort key

Use Query API

Example: Given a device, find all readings between epoch X, Y

Device-measurements		
Partition Key	Sort key	Attributes
DeviceId = 1	epoch = 5513A97C	Temperature = 30, pressure = 90
DeviceId = 1	epoch = 5513A9DB	Temperature = 30, pressure = 90

N:M relationships

Use a table and GSI with partition and sort key elements switched

Use Query API

Example: Given a user, find all games. Or given a game, find all users.

User-Games-Table	
Partition Key	Sort key
UserId = bob	GameId = Game1
UserId = fred	GameId = Game2
UserId = bob	GameId = Game3

Game-Users-GSI	
Partition Key	Sort key
GameId = Game1	UserId = bob
GameId = Game2	UserId = fred
GameId = Game3	UserId = bob

Hierarchical Data

Tiered relational data structures

It’s all about aggregations…

Social Network

Document Management

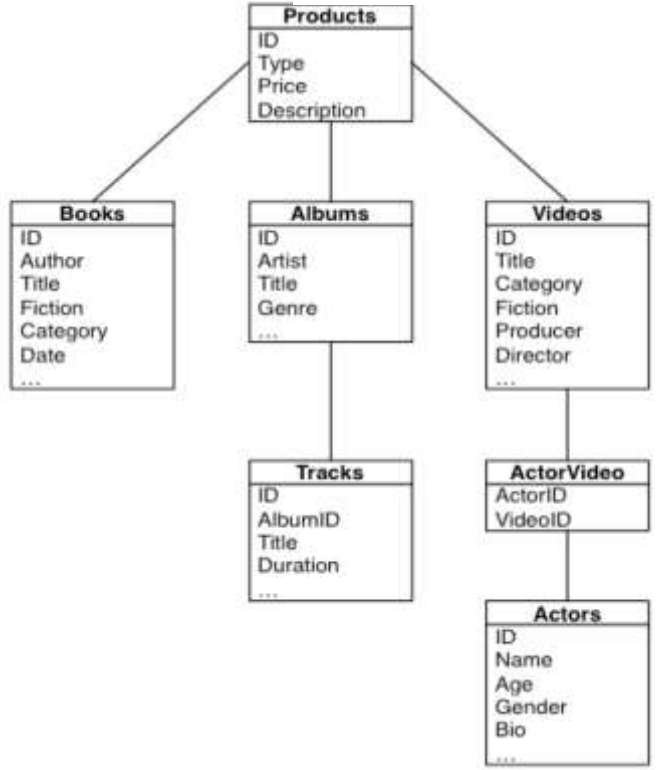
Process Control

IT Monitoring

Data Trees

How OLTP Apps Use Data

- Mostly Hierarchical Structures
- Entity Driven Workflows
- Data Spread Across Tables
- Requires Complex Queries
- Primary driver for ACID



Hierarchical Structures as Item Collections…

Use composite sort key to define a Hierarchy
Highly selective result sets with sort queries
Index anything, scales to any size

	Primary Key		Attributes						
	ProductID	type							
Items	1	bookID	title	author	genre	publisher	datePublished	ISBN	
			Ringworld	Larry Niven	Science Fiction	Ballantine	Oct-70	0-345-02046-4	
	2	albumID	title	artist	genre	label	studio	relesed	producer
			Dark Side of the Moon	Pink Floyd	Progressive Rock	Harvest	Abbey Road	3/1/73	Pink Floyd
		albumID:trackID	title	length	music	vocals			
			Speak to Me	1:30	Mason	Instrumental			
		albumID:trackID	title	length	music	vocals			
			Breathe	2:43	Waters, Gilmour, Wright	Gilmour			
		albumID:trackID	title	length	music	vocals			
			On the Run	3:30	Gilmour, Waters	Instrumental			
	3	movieID	title	genre	writer	producer			
			Idiocracy	Scifi Comedy	Mike Judge	20th Century Fox			
		movieID:actorID	name	character	image				
			Luke Wilson	Joe Bowers	img2.jpg				
		movieID:actorID	name	character	image				
			Maya Rudolph	Rita	img3.jpg				
		movieID:actorID	name	character	image				
			Dax Shepard	Frito Pendejo	img1.jpg				

… or as Documents (JSON)

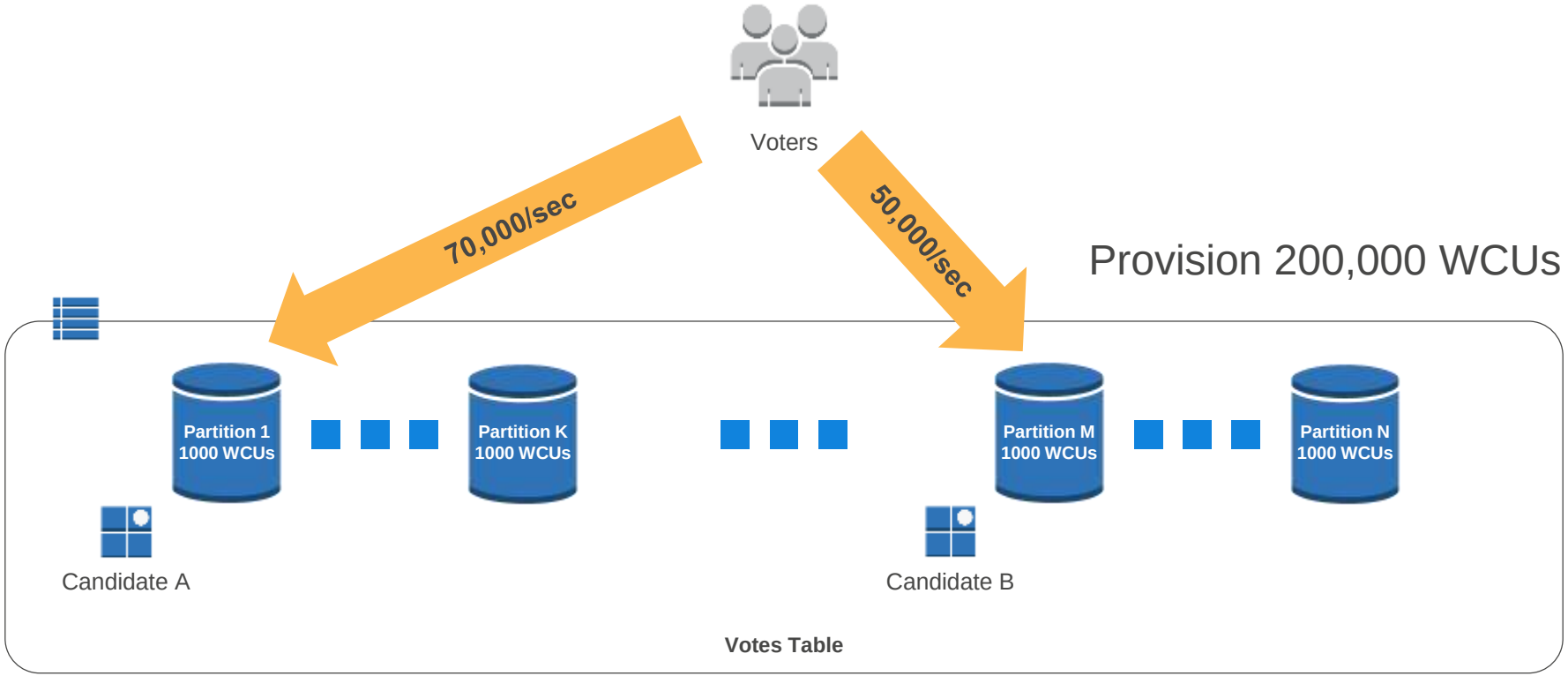
JSON data types (M, L, BOOL, NULL)

Index root attributes

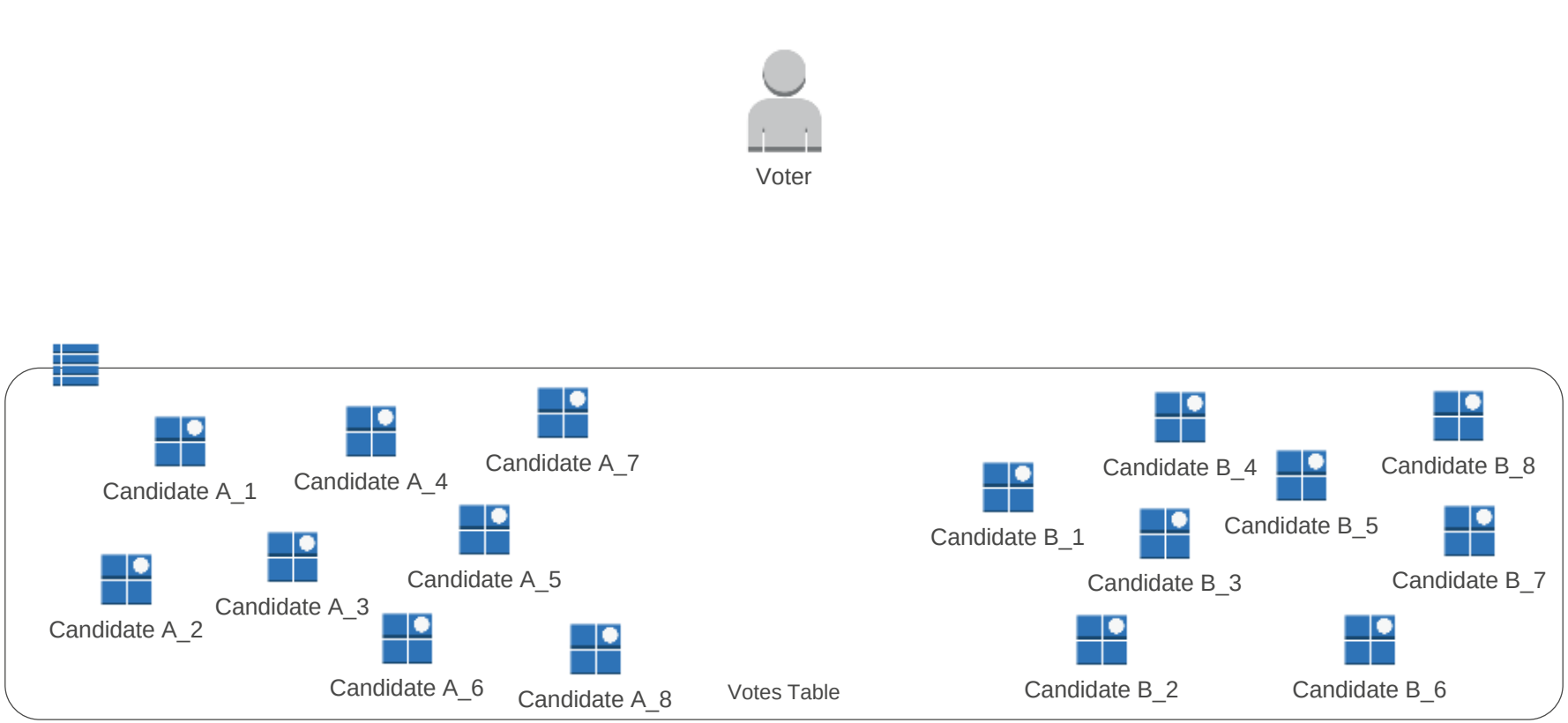
Fully Atomic Updates

	Primary Key	Attributes							
	ProductID								
Items	1	id	title	author	genre	publisher	datePublished	ISBN	
		bookID	Ringworld	Larry Niven	Science Fiction	Ballantine	Oct-70	0-345-02046-4	
	2	id	title	artist	genre	Attributes			
		albumID	Dark Side of the Moon	Pink Floyd	Progressive Rock	{ label:"Harvest", studio: "Abbey Road", published: "3/1/73", producer: "Pink Floyd", tracks: [{title: "Speak to Me", length: "1:30", music: "Mason", vocals: "Instrumental"},{title: ”Breathe", length: ”2:43", music: ”Waters, Gilmour, Wright", vocals: ”Gilmour"},{title: ”On the Run", length: “3:30", music: ”Gilmour, Waters", vocals: "Instrumental"}]}			
	3	id	title	genre	writer	Attributes			
		movieID	Idiocracy	Scifi Comedy	Mike Judge	{ producer: "20th Century Fox", actors: [{ name: "Luke Wilson", dob: "9/21/71", character: "Joe Bowers", image: "img2.jpg"},{ name: "Maya Rudolph", dob: "7/27/72", character: "Rita", image: "img1.jpg"},{ name: "Dax Shepard", dob: "1/2/75", character: "Frito Pendejo", image: "img3.jpg"}]			

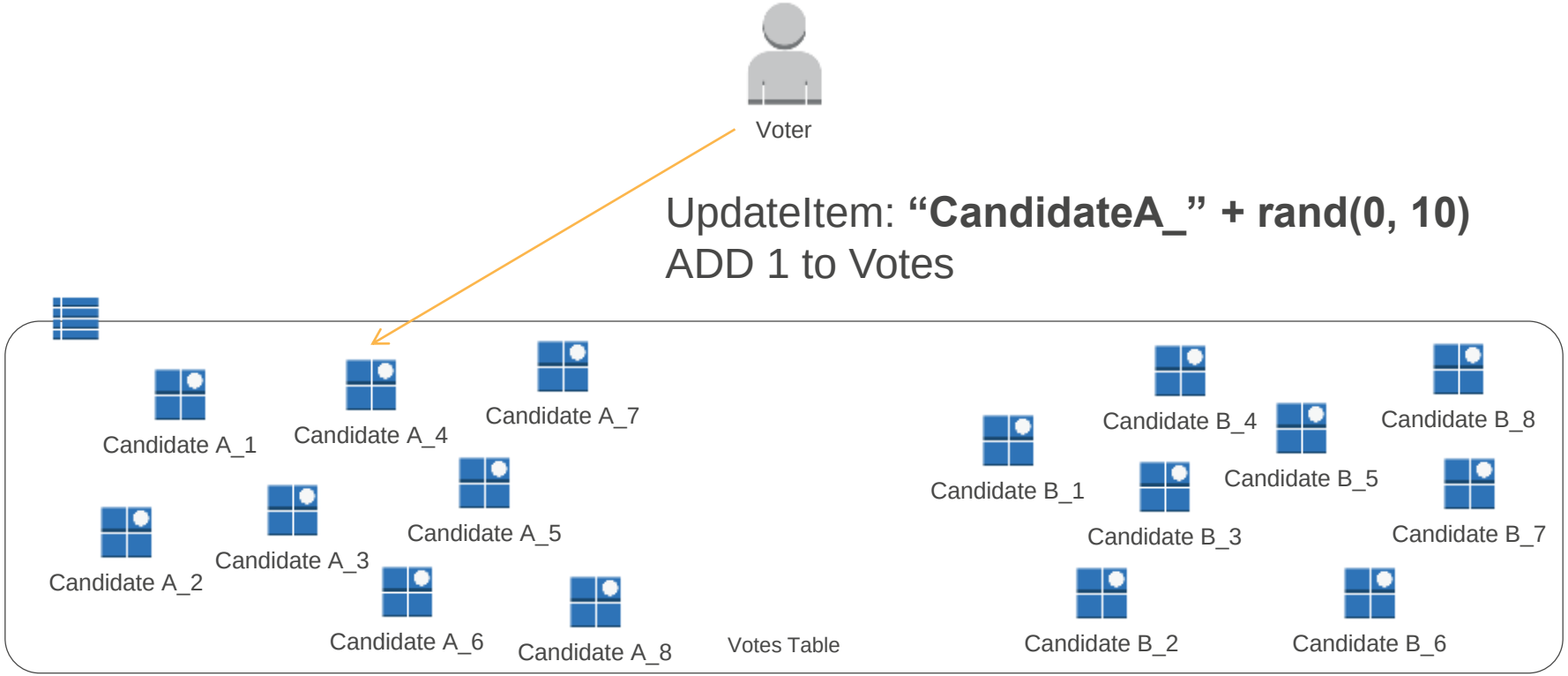
Scaling bottlenecks



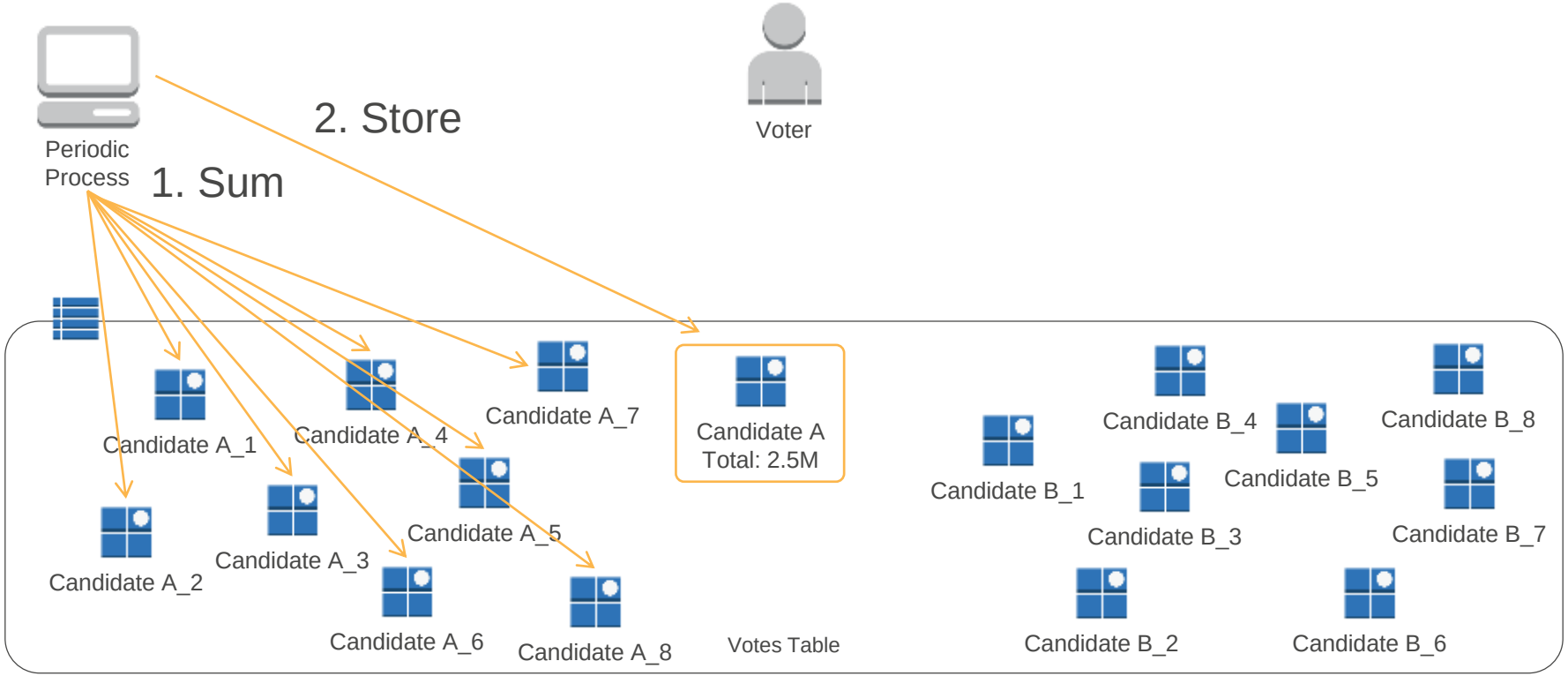
Write sharding



Write sharding



Shard aggregation



Write Sharded Indexing

Use for GSI's with high volume aggregations
Common when low cardinality attributes must be indexed
Scales to any size workload

	Primary Key		Attributes						
	ProductID	type							
Items	1	bookID	title	author	genre	publisher	datePublished	ISBN	
			Ringworld	Larry Niven	Science Fiction:1	Ballantine:50	Oct-70	0-345-02046-4	
	2	albumID	title	artist	genre	label	studio	relesed	producer
			Dark Side of the Moon	Pink Floyd	Progressive Rock:99	Harvest	Abbey Road	3/1/73	Pink Floyd:2
		albumID:trackID	title	length	music	vocals			
			Speak to Me	1:30	Mason	Instrumental			
		albumID:trackID	title	length	music	vocals			
			Breathe	2:43	Waters, Gilmour, Wright	Gilmour			
		albumID:trackID	title	length	music	vocals			
			On the Run	3:30	Gilmour, Waters	Instrumental			
	3	movieID	title	genre	writer	producer			
			Idiocracy	Scifi Comedy:42	Mike Judge	20th Century Fox:17			
		movieID:actorID	name	character	image				
			Luke Wilson	Joe Bowers	img2.jpg				
		movieID:actorID	name	character	image				
			Maya Rudolph	Rita	img3.jpg				
		movieID:actorID	name	character	image				
			Dax Shepard	Frito Pendejo	img1.jpg				

Conclusion

Keys to Success

- Select a workload that is a good fit for NoSQL
- Understand the source data and access patterns
- Test thoroughly and often
- Plan on an iterative migration process

Thank you!