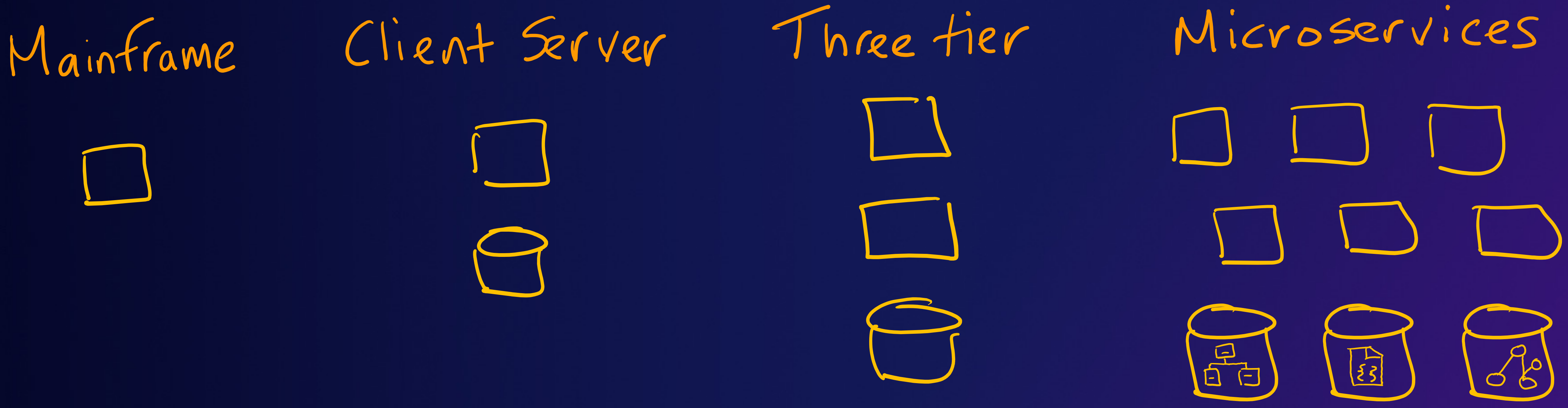
Databases
Part 1: Basics

Aleksandr Bernadskii

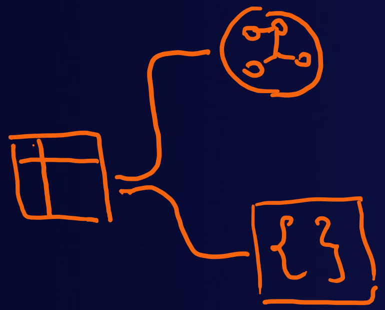
Solutions Architect
Amazon Web Services

There was a time when...



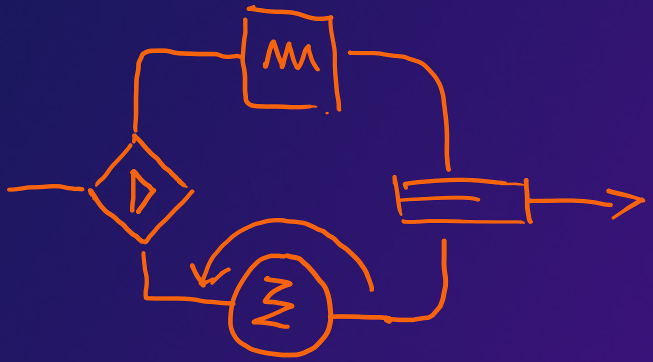
Dimensions to consider

Shape



Size

Compute



Shape of data

Shape	Usecase	Example
Row store	Operate on a single or group of rows	Transactions, Payroll
Column store	Aggregations, big scans	Analytics, DSS
Key-Value	Low latency ingestion and query by Key	Tracking, Caching
Document	Index and store documents for query on attributes	Content Mgmt, Patient data
Graph	Store and query relationships	Recommendations
Time-series	Store and aggregate data sequenced entries	Telemetry, Sensors
Wide-column	Attribute-based data with query and sorting on columns	Features of a car
Blockchain	Ledger with immutable records	Audit records, Record history
Unstructured	Get and put Objects, Documents	Binary files, PDFs

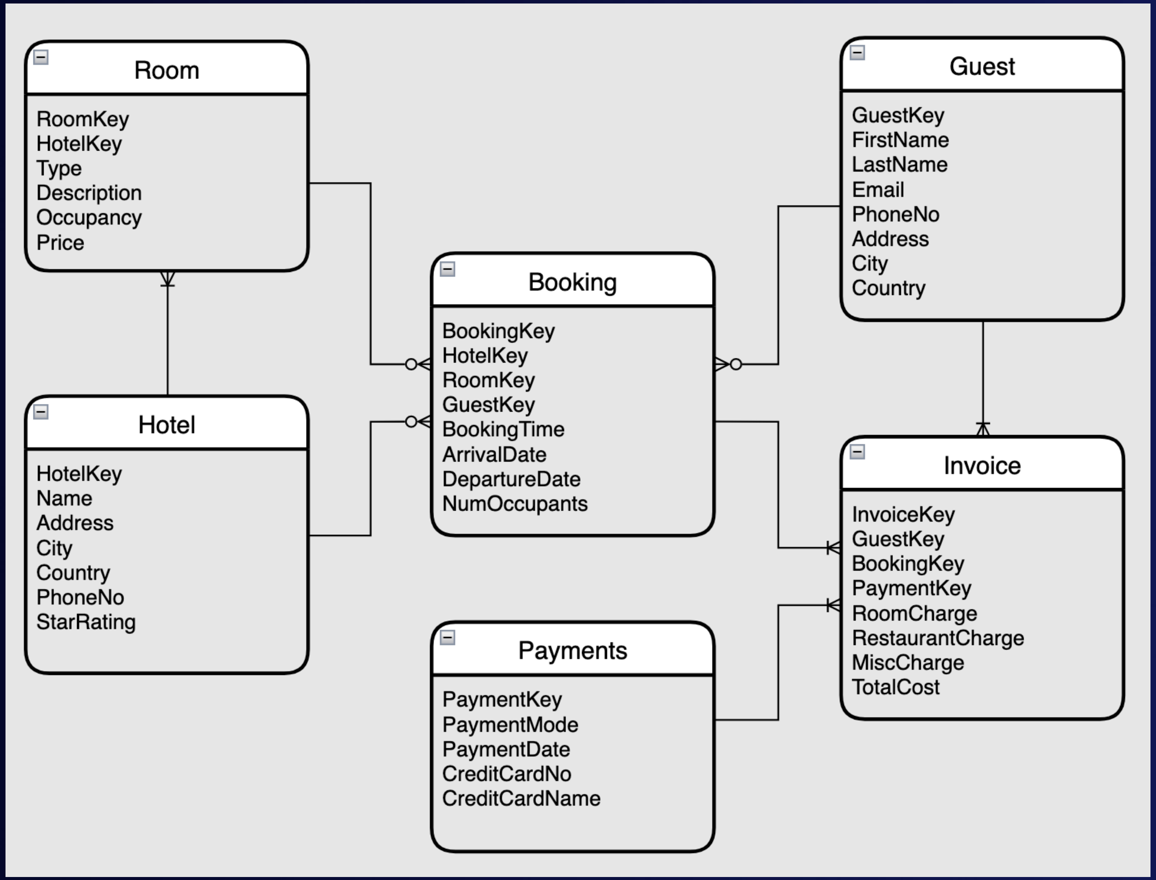
Sizing and scaling

Considerations	Example
Size at limit – bound/unbound	Number of countries/cities – bound Number of sensors - unbounded
Working set size	Sales data history, but only 12 months relevant Session data of active users
Retrieval size and Caching	Get one row/document
Partitioned or monolithic	Sensor data is partitionable Company payrol does not have a natural partition

Compute flexibility

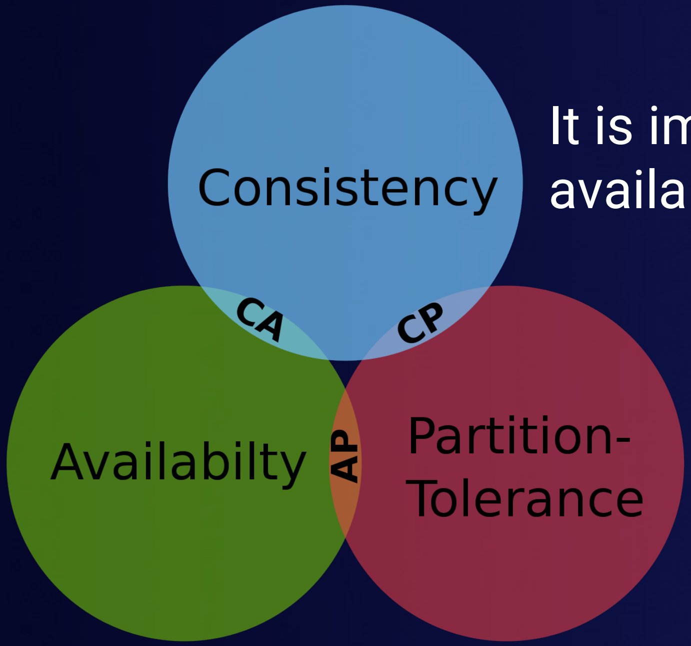
Considerations	Example
Compute functions	Sum of sales for last 12 months Get and Put Transations
Throughput	Milions of users getting hotels availability every minute Users scrolling a blog website
Latency	Get the location of a car every 5 seconds Get min,max of average bonus pay last year
Velocity of change	Inventory counts update frequently GPS data is almost never updated
Rate of ingestion	Device metrics data inserted every second Few new employees added every month

Relational DBs or NoSQL?




```json
{
    "object": {
        "type": "booking",
        "bookingKey": "asdbc-odpkg-otjg7",
        "bookingTime": "2022-05-09 10:00:00 CET",
        "arrivalDate": "2022-05-10 10:00:00 CET",
        "departureDate": "2022-05-12 10:00:00 CET",
        "rooms": [{
            "type": "DoubleRoom",
            "Description": " Double Room with see view",
            "price": 145,
            "hotel": {
                "name": "AnyHotel",
                "address": "AnyAddress",
                "city": "AnyCity",
                "country": "AnyCountry"
            }
        }]
    }
}
```

CAP theorem. ACID and BASE



It is impossible to achieve both consistency and availability in a partition tolerant distributed system

ACID Model

BASE Model

OLTP and OLAP

	OLTP	OLAP
Function	Day to day operation	Decision support
Database Design	Application oriented	Subject oriented
Data	Current, up-to-date detailed, flat relational, isolated	Historical, summarized multi-dimensional, consolidated
Usage	Repetitive	Ad-hoc
Access	Read/Write	Lots of scans
Unit of Work	Short, simple transaction	Complex query
Database Size	Gigabytes	Terabytes
Metric	Transaction throughput	Query throughput, response

Relational Databases (RDB) vs Non-Relational Databases

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

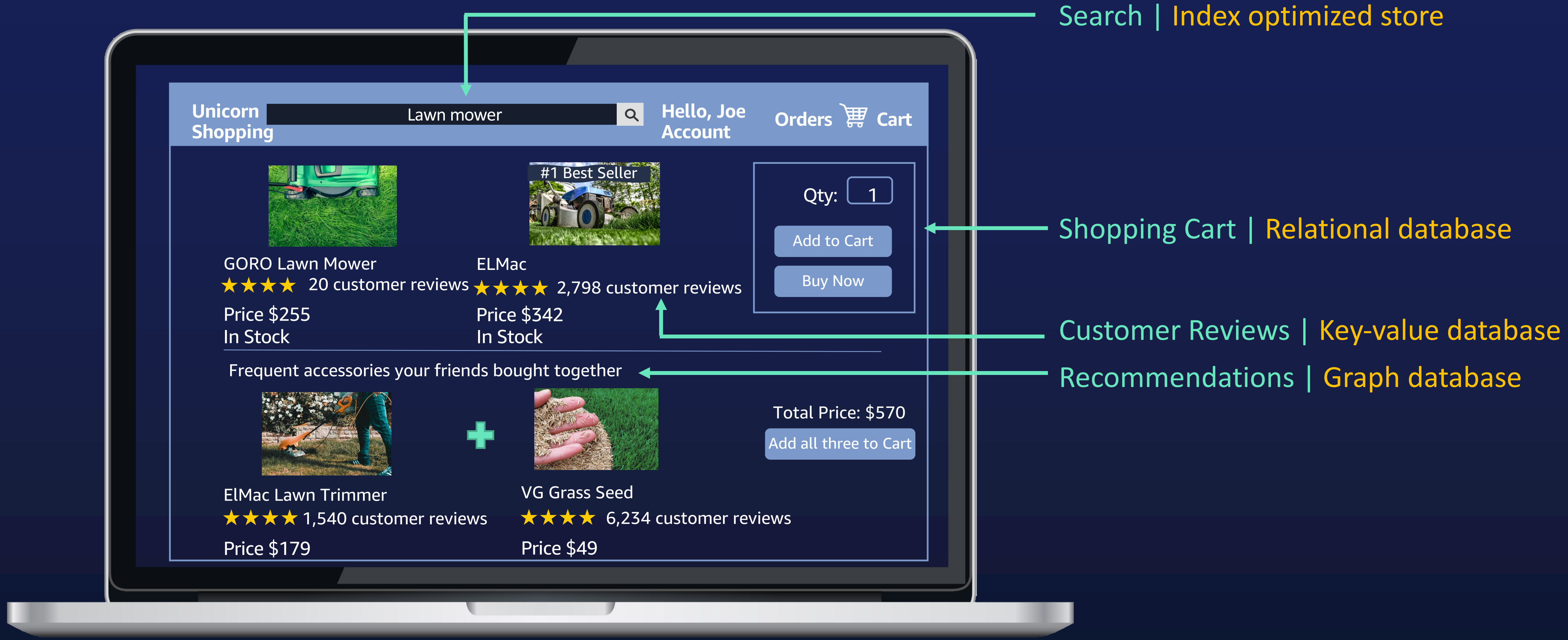
SQL and NoSQL Scalability Pitfalls

- Scaling vertically is not infinitely
- Adding more indexes increases the latency for insert/update
- Transaction consistency overhead
- Sharding may be difficult
- Lock contention increases with the amount of transactions

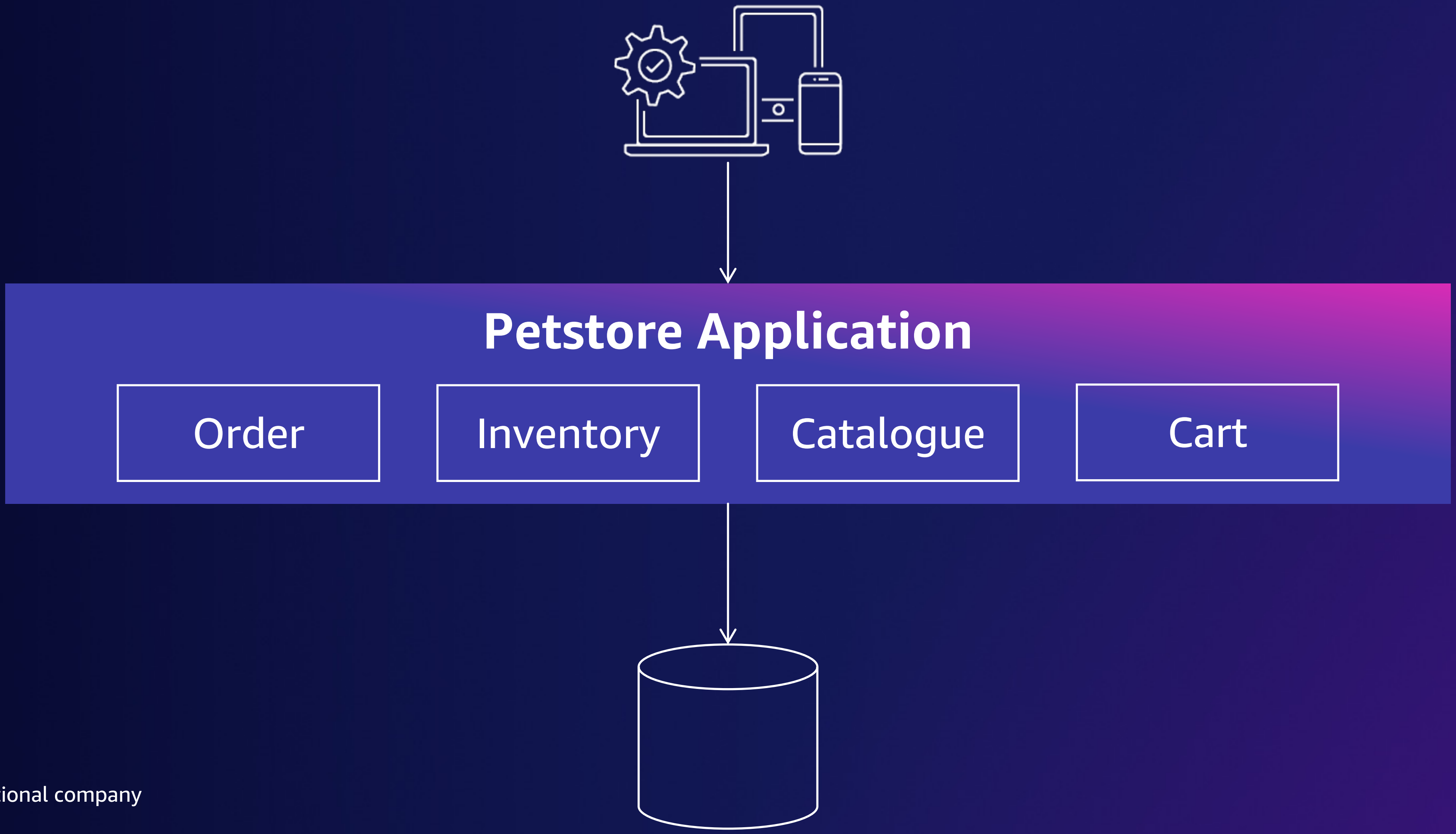
- Not knowing your data access patterns
- Adding too many indexes to satisfy new patterns
- Sharding producing uneven or narrowed distribution of data
- Not keeping related data together
- Scanning whole items/tables instead of specific attributes

Modern cloud-based applications

Loosely coupled micro-services and purpose-built data stores

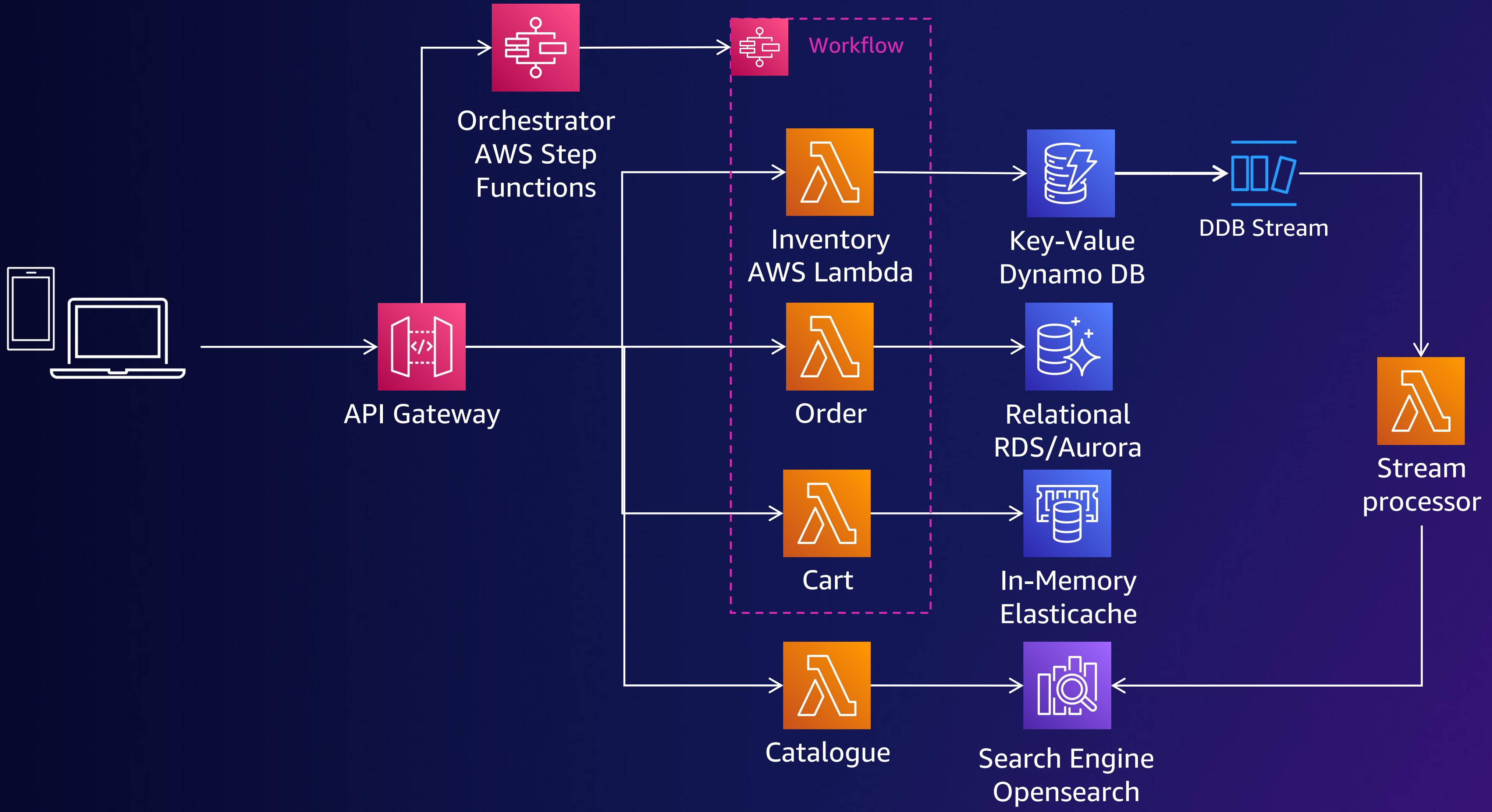


AWSome pets - Current architecture



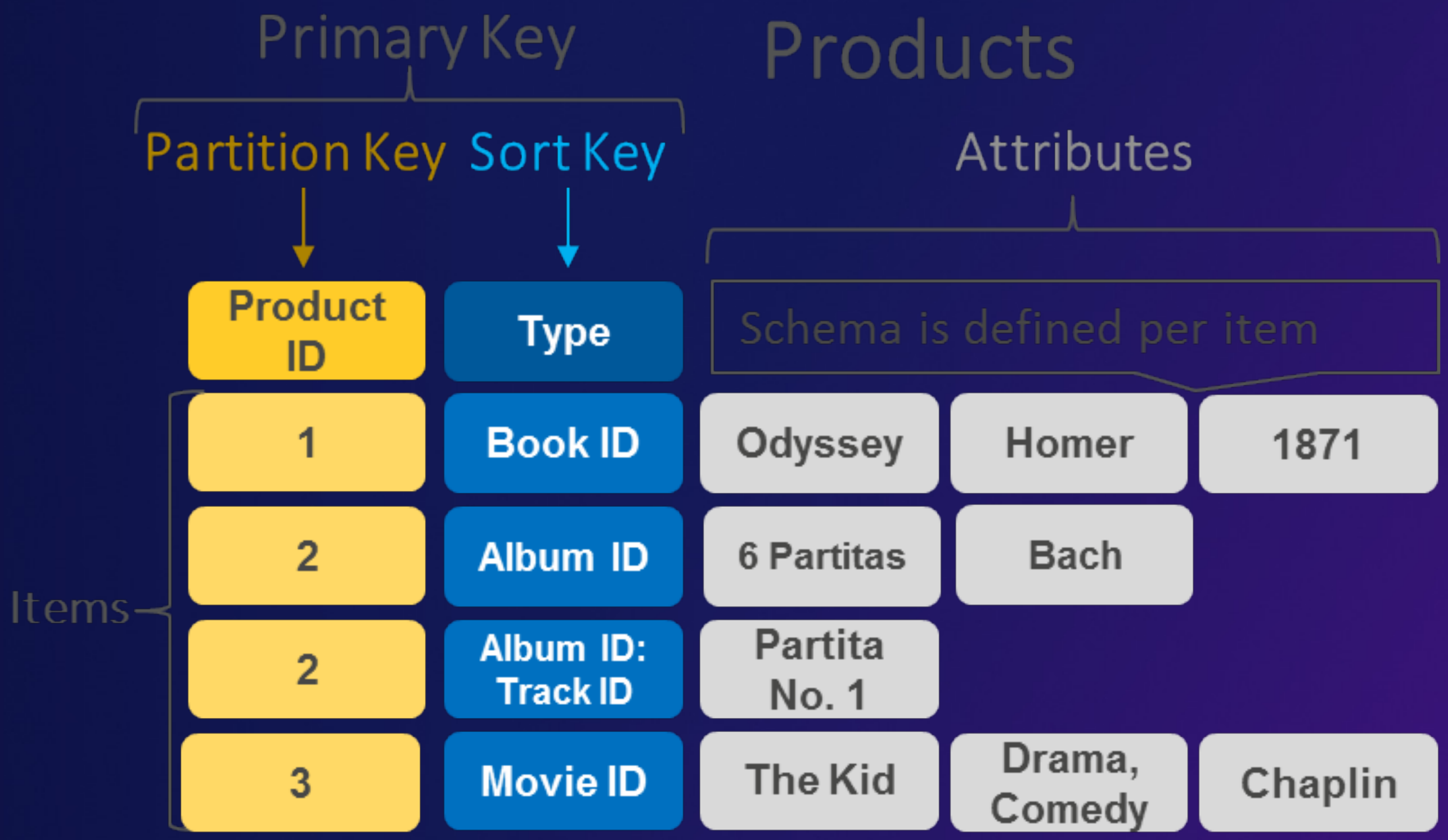
* AWSome pets is a fictional company

Target architecture

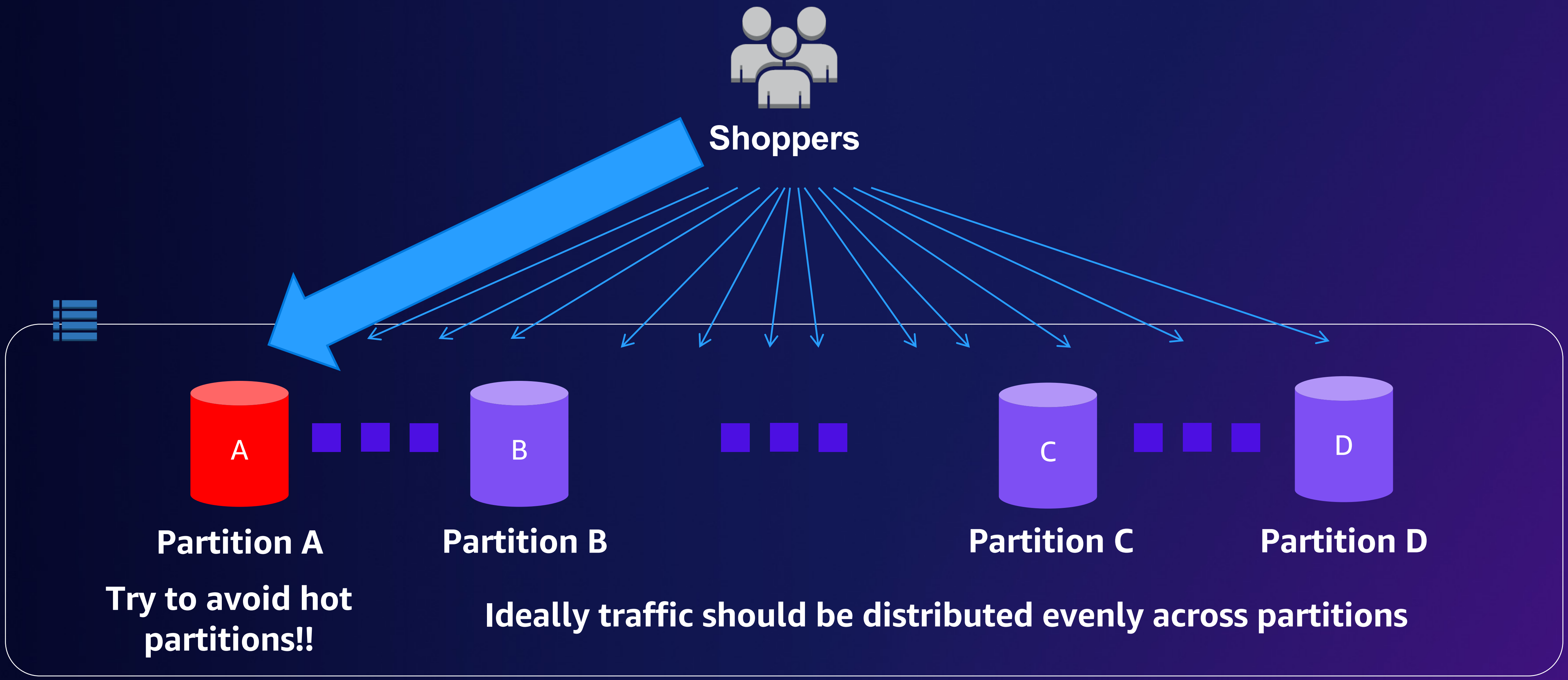


Key–value database

- Simple key–value pairs
- Partitioned by keys
- Resilient to failure
- High-throughput, low-latency reads and writes
- Consistent performance at scale



Good Partitioning



Why document databases?

JSON is a flexible and natural format

JSON is the de facto API output

Store, query, & index JSON data natively

JSON all the way

When should you use a document database?

JSON data

Flexible schema for fast iteration

Ad hoc query capabilities

Flexible indexing

Operational and analytics workloads

When are other databases more appropriate?

Database to enforce referential integrity	Known, static access patterns for primary key lookups	Large binary data	Ultralow latency, ephemeral data	Highly connected social dataset	Log analytics, full-text searches
Relational	Key value	Object Store	In memory	Graph	Search

Graph databases

Graph databases excel at answering questions like:

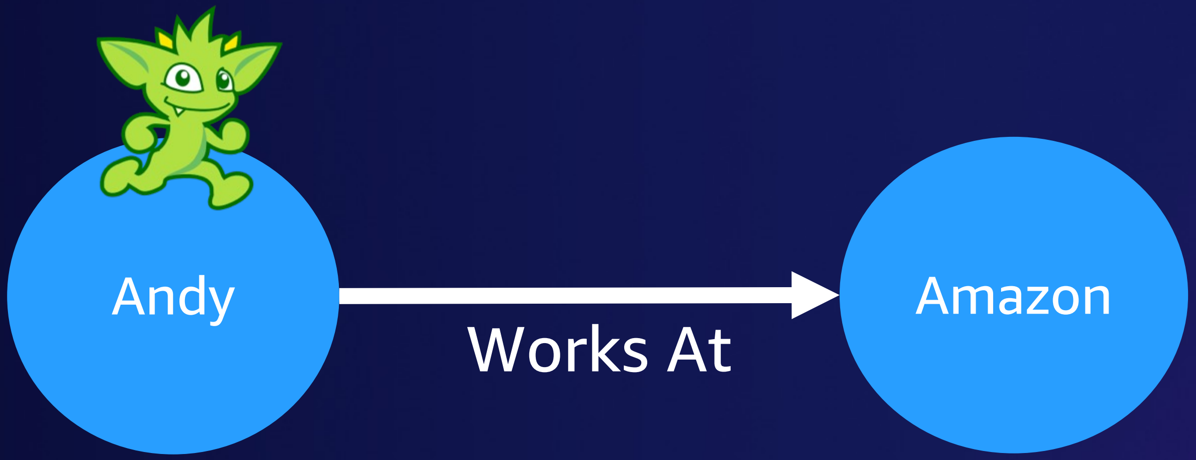
- What does this person want to buy?
- How are these two people connected?
- Why did X impact Y?

These questions:

- Navigate (variably) connected structure
- Filter or compute a result on the basis of the *strength*, *weight,* or *quality* of relationships
- Require traversing an unknown number of connections

Query languages- Designed to move through data

- Graphs query languages are optimized to use connections to move through a network



- Relational queries work by combining sets of data.
-

Person
Andy

X

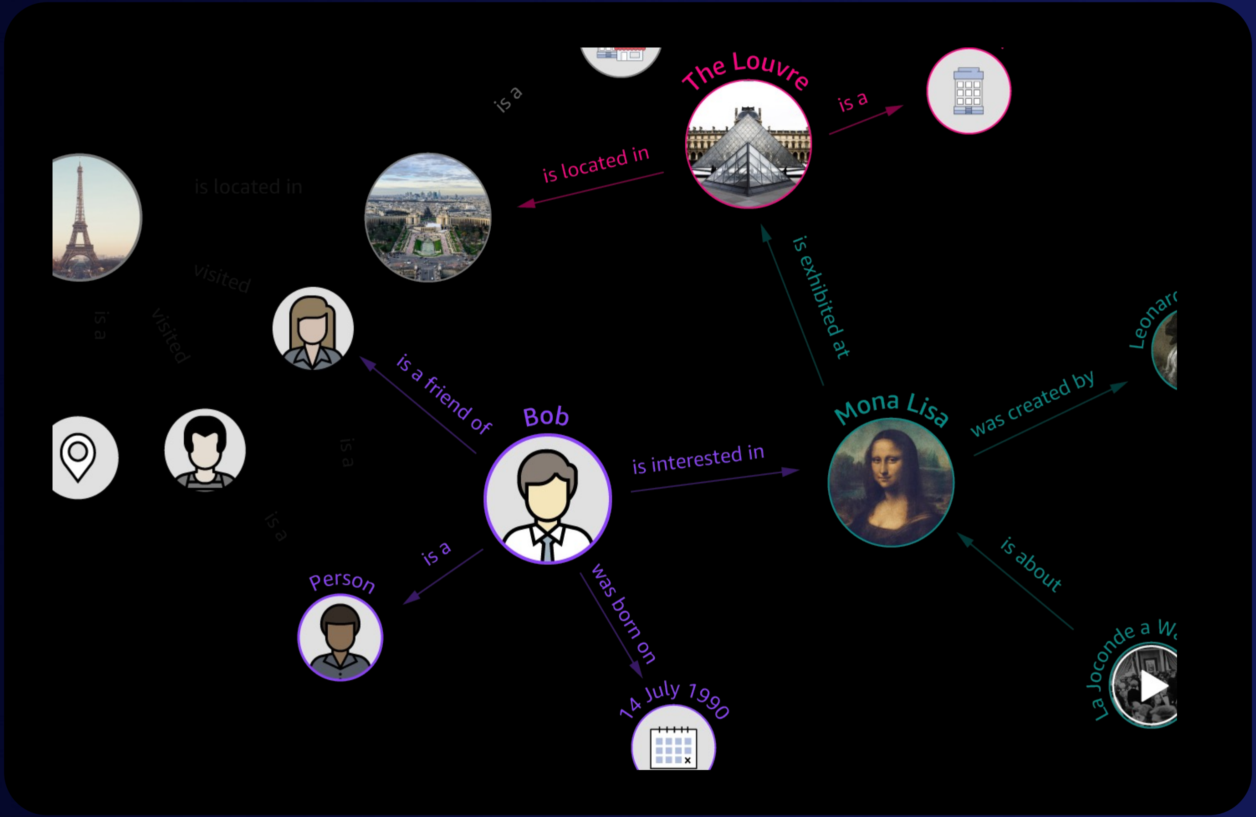
Company
Amazon

=

Person	Company
Andy	Amazon

What is a Knowledge Graph?

Understanding the who, what, when, and where



Benefits

1. Link disparate data sources

Link disparate and heterogeneous data sources together to discover hidden connections

2. Improved search results

Increase productivity by making data easily accessible through improved search relevance

3. Augment ML/AI

Improve the efficiency and effectiveness of machine learning models by providing context and augmentation with related content

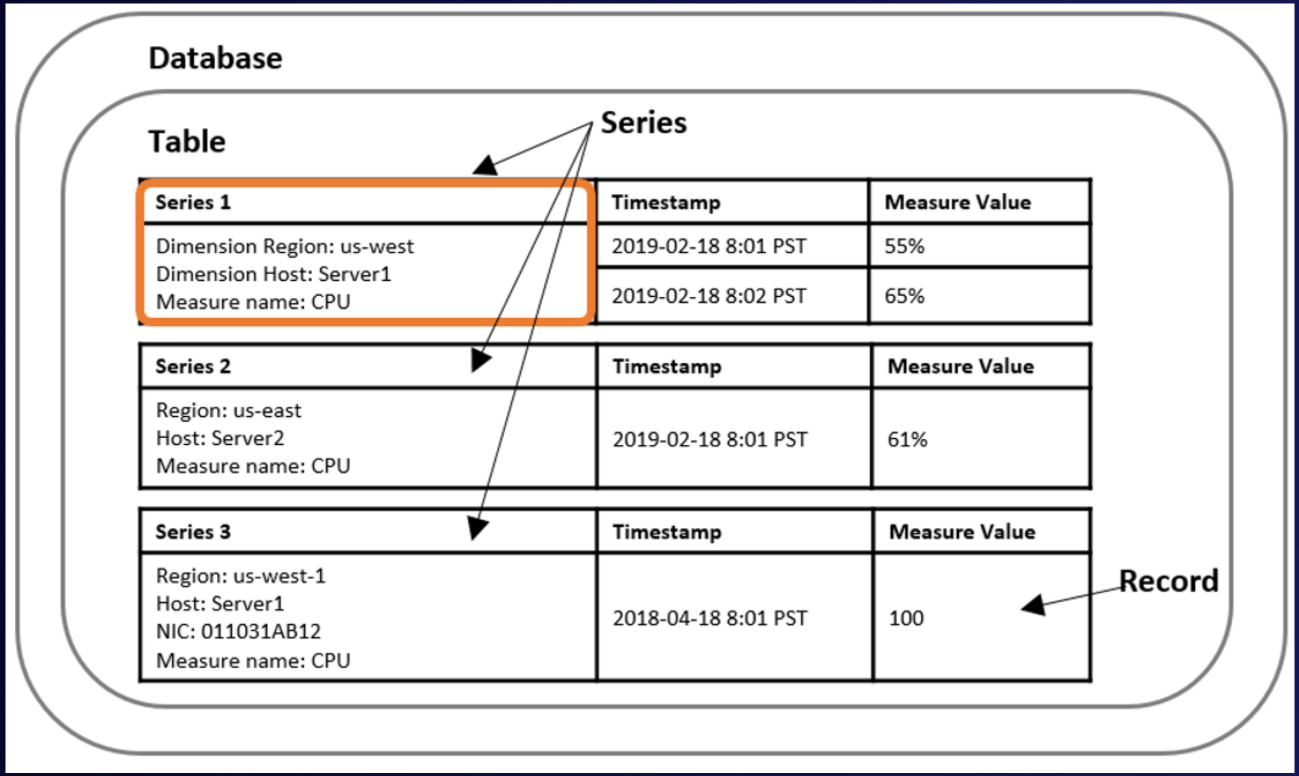
https://aws.amazon.com/blogs/apn/exploring-knowledge-graphs-on-amazon-neptune-using-metaphactory/

Characteristics of time-series data

A sequence of data points recorded over a time interval

- Data is append-only
- Data typically arrives in time-order
- Queries are over a time interval
- Specialized math is required to find trends and patterns
- Older data is typically aggregated over time or discarded

Timeseries databases. Concepts



- **Series** - A sequence of data points recorded over a time interval
- **Record** - A single data point in the time series.
- **Dimension** - An attribute that describes the meta-data of the time series.
- **Measure** - An attribute that describes the data of the series
- **Timestamp** - Every record has a timestamp, which indicates when the **Measure** was collected.
- **Table** - A table is a container for (related) time series with timestamp, dimensions and measures.
- **Rentention policy** - A value you can set at the table level to dictate how long time series data will be retained.
- **Database** - A top level container for tables and retention policies.

AWS Purpose built databases

Relational	Key-value	Document	In-memory	Graph	Time-series	Ledger	Wide column
Referential integrity, ACID transactions, schema-on-write	High throughput, low-latency reads and writes, endless scale	Store documents; quickly access querying on any attribute	Query by key with microsecond latency	Quickly and easily create and navigate relationships between data	Collect, store, and process data sequenced by time	Complete, immutable and verifiable history of all changes to application data	Scalable, highly available, and managed Apache Cassandra-compatible service
Aurora RDS	DynamoDB	DocumentDB	ElastiCache	Neptune	Timestream	QLDB	Amazon Keyspaces
Lift and shift, ERP, CRM, finance	Real-time bidding, shopping cart, social, product catalog, customer preferences	Content management, personalization, mobile	Leaderboards, real-time analytics, caching	Fraud detection, social networking, recommendation engine	IoT applications, event tracking	Systems of record, supply chain, health care, registrations, financial	Build low-latency applications, leverage open source, migrate Cassandra to the cloud

Thank you!

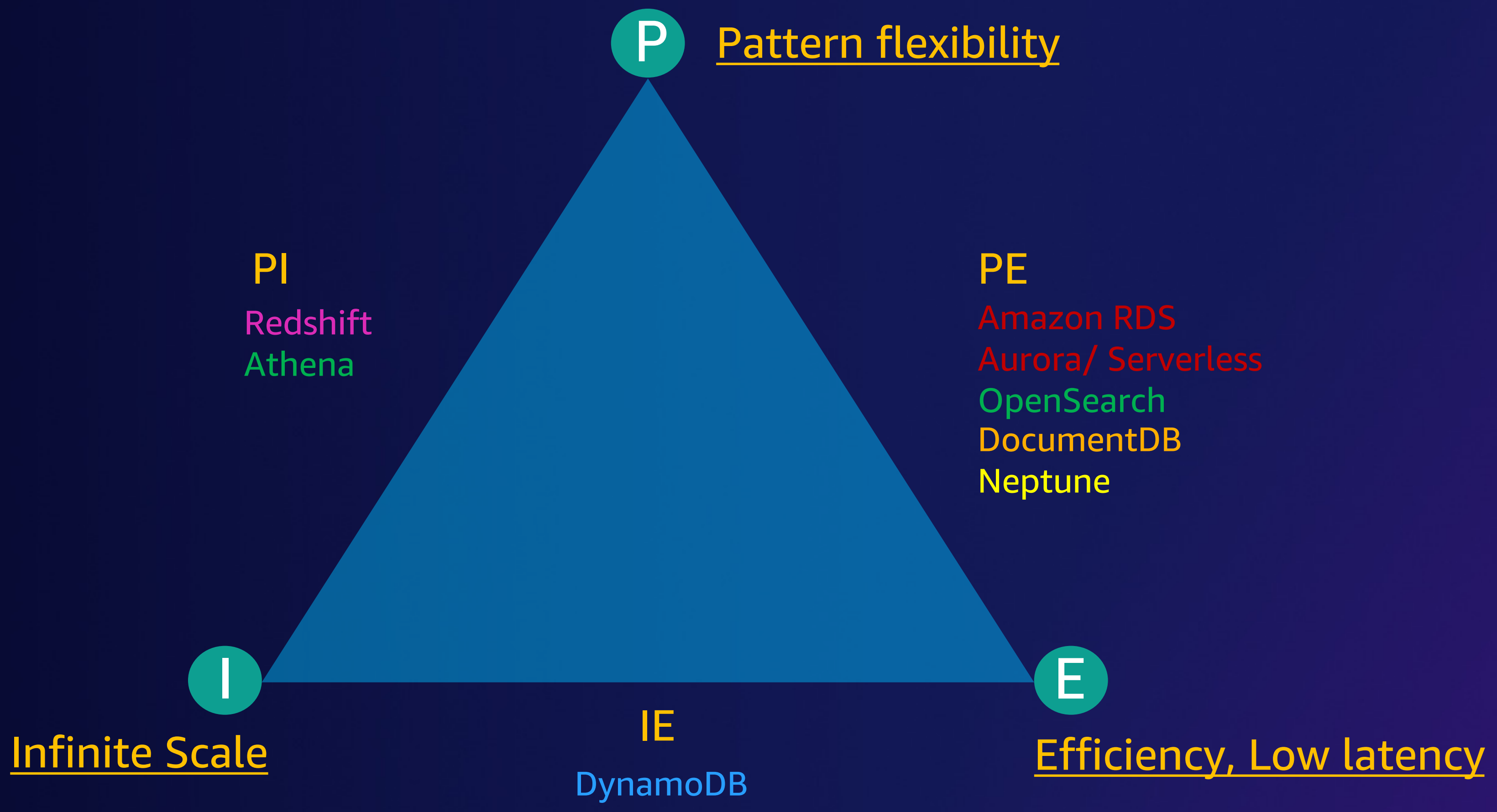
Decision Matrix

Service	Shape	Size	Workload	Performance	Durability	Expertise	Legacy	Business Needs	Ops. Load	Platform Integration
Amazon Aurora MySQL	Structured, semistruct.	Mid TB Range	K/V lookups, transactional, light analytics	High throughput, low latency	High	Relational, MySQL, SQL Server	User defined code, COTS	Agility, cost opt.	Low to moderate	Serverless, IAM, Lambda, Auto Scaling, Amazon S3
Amazon Aurora PostgreSQL	Structured, semistruct.	Mid TB Range	Transactional, light analytics	High throughput, low latency	High	Relational, PostgreSQL, Oracle	User defined code, COTS	Agility, cost opt.	Moderate	In the works
Amazon RDS DB Engines	Structured, semistruct.	Low TB Range	Transactional, light analytics	Mid-to-high throughput, low latency	User controlled	Engine specific	Engine req., COTS	Right sizing	Moderate	Log streaming
Amazon DynamoDB	Semistruct.	High TB Range	K/V lookups, NoSQL, OLTP, document store	Ultra-high throughput, low latency	High	NoSQL	No active development	Zero downtime, Ultra-high scale	Low	Serverless, IAM, Lambda, DAX, Auto Scaling, Kinesis Streams
Amazon Neptune	Graph-structured	Mid TB Range	Graph, highly connected data, transact.	High throughput, low latency	High	Graph, Gremlin, SPARQL		Agility, cost opt.	Low	IAM, Amazon S3

Decision Matrix (cont.)

Service	Shape	Size	Workload	Performance	Durability	Expertise	Legacy	Business Needs	Ops. Load	Platform Integration
Amazon ElastiCache	Semistruct., Unstructured	Low TB Range	In-memory caching, K/V lookups, NoSQL	High throughput, ultra-low latency	Low (In-memory, auto failover to read replica	Caching, NoSQL		Response latency, DB cost opt.	Low	Scalable clusters
Amazon Redshift	Structured, semistruct.	PB Range	Optimized analytics	Mid-to-high latency	High	DW, data science	User defined code, COTS	Cost opt.	Moderate	IAM, Amazon S3
Amazon Athena	*Structured, semistruct.*	*TB Range*	*Flexible analytics*	*High latency*	*High (Amazon S3)*	*Data lakes, data science*		*Flexibility*	*Low*	*IAM, Amazon S3*
Amazon EMR	*Semistruct.*	*PB Range*	*Flexible analytics*	*Low-to-high latency*	*User-controlled*	*Data lakes, data science, DW*	*Tooling & versioning*	*Cost opt.*	*High*	*IAM, Amazon S3, EC2 Spot*

PIE Theorem for data stores



Data Models:
Relational
Columnar
Wide column
Document
Graph
Unstructured