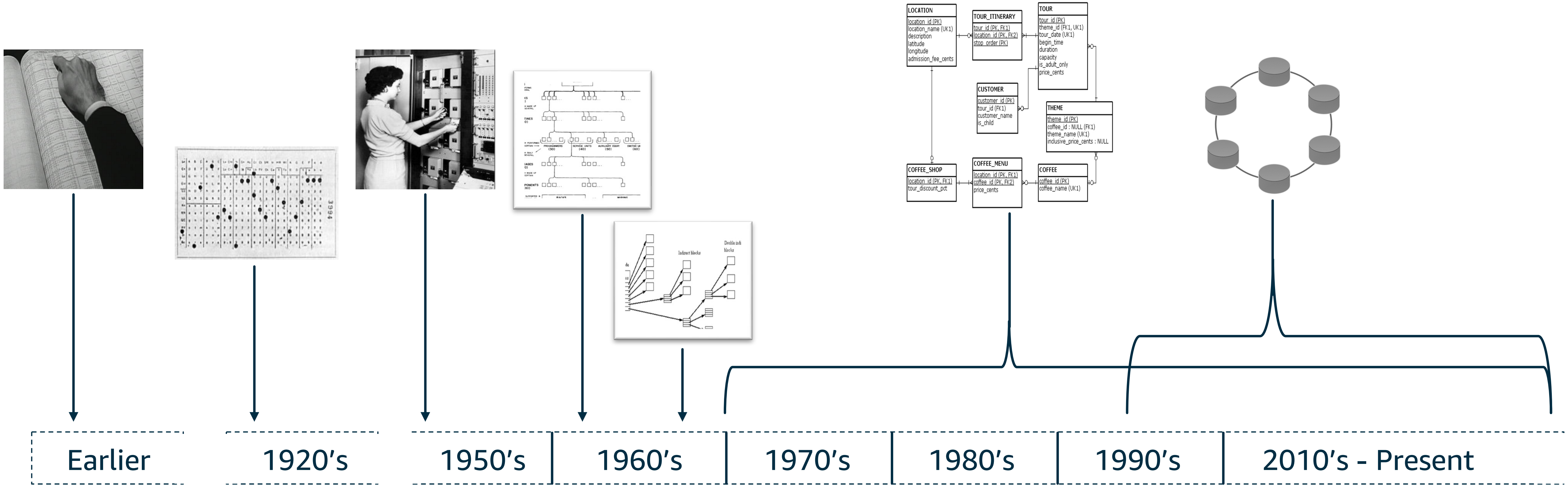
AWS Purpose Built Databases

Overview

Eren Akbaba, AWS
eakbaba@amazon.com
Bilkent University

History of Databases

Timeline of Database Systems



Earlier

1920's

1950's

1960's

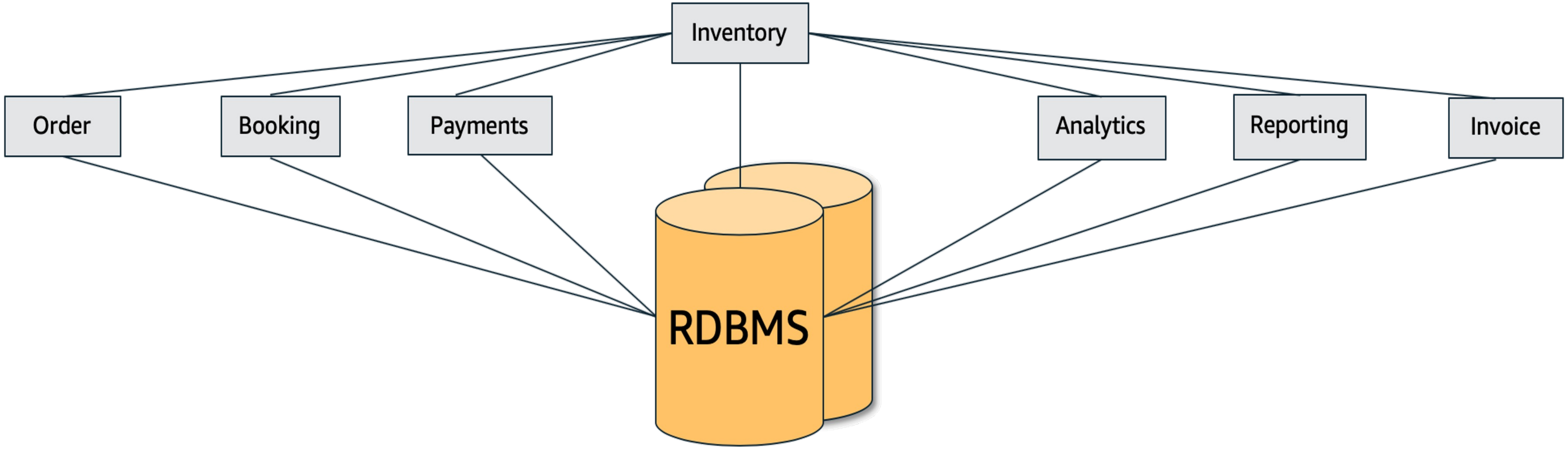
1970's

1980's

1990's

2010's - Present

Typical on-premise / monolithic architecture



Challenges faced with on-premise / monolithic architectures

Managing on-premise database environments - time consuming and complex

Scaling Limitations

- Scaling for peak, forecasting
- Cost of provisioning for peak and licensing
- Horizontally scaling /Sharding requires complex application logic

Availability Challenges

- HA Cluster setup and data replication for high availability
- Security upgrades will require downtime

High Operations

- Engineers/DBAs need to invest significant effort in provisioning fine tuning database configurations , patching and upgrades
- Deal with operational issues with database outages or availability drops

Modern Applications

Modern real-time applications require

Performance, Scale and Availability

E-Commerce

Media streaming

Social media

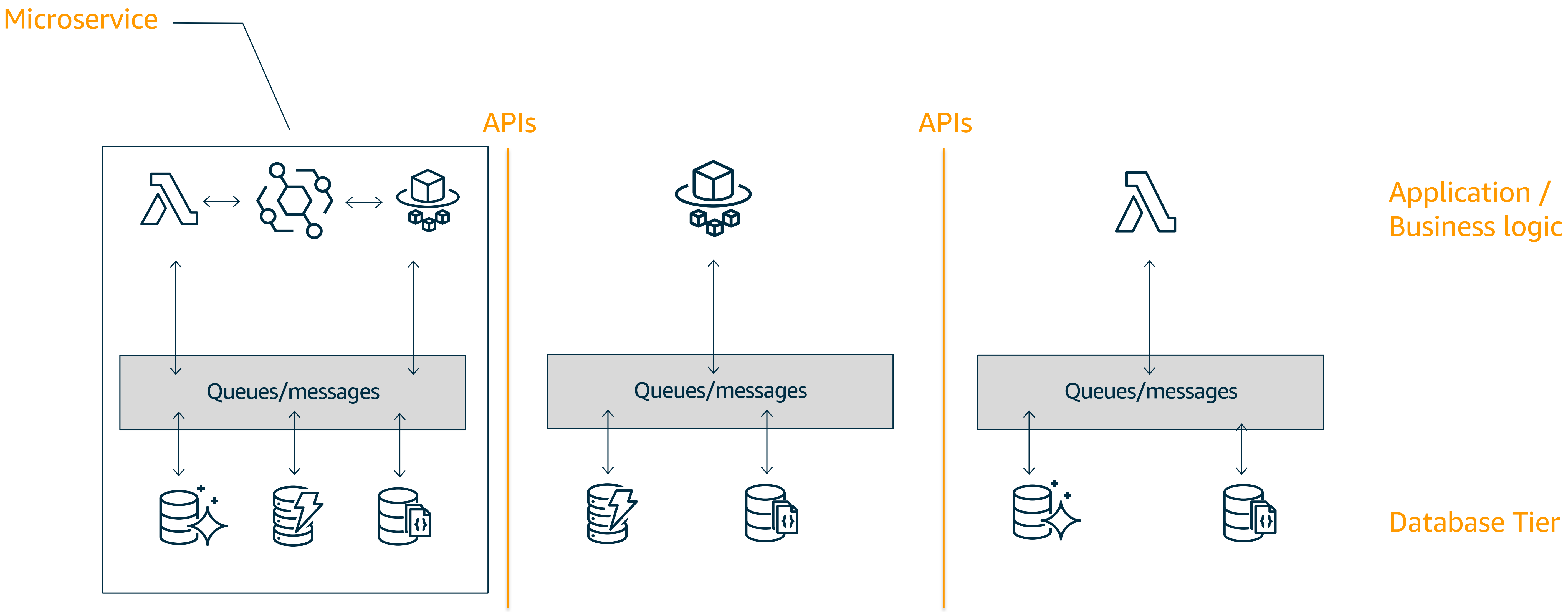
Online gaming

Shared economy

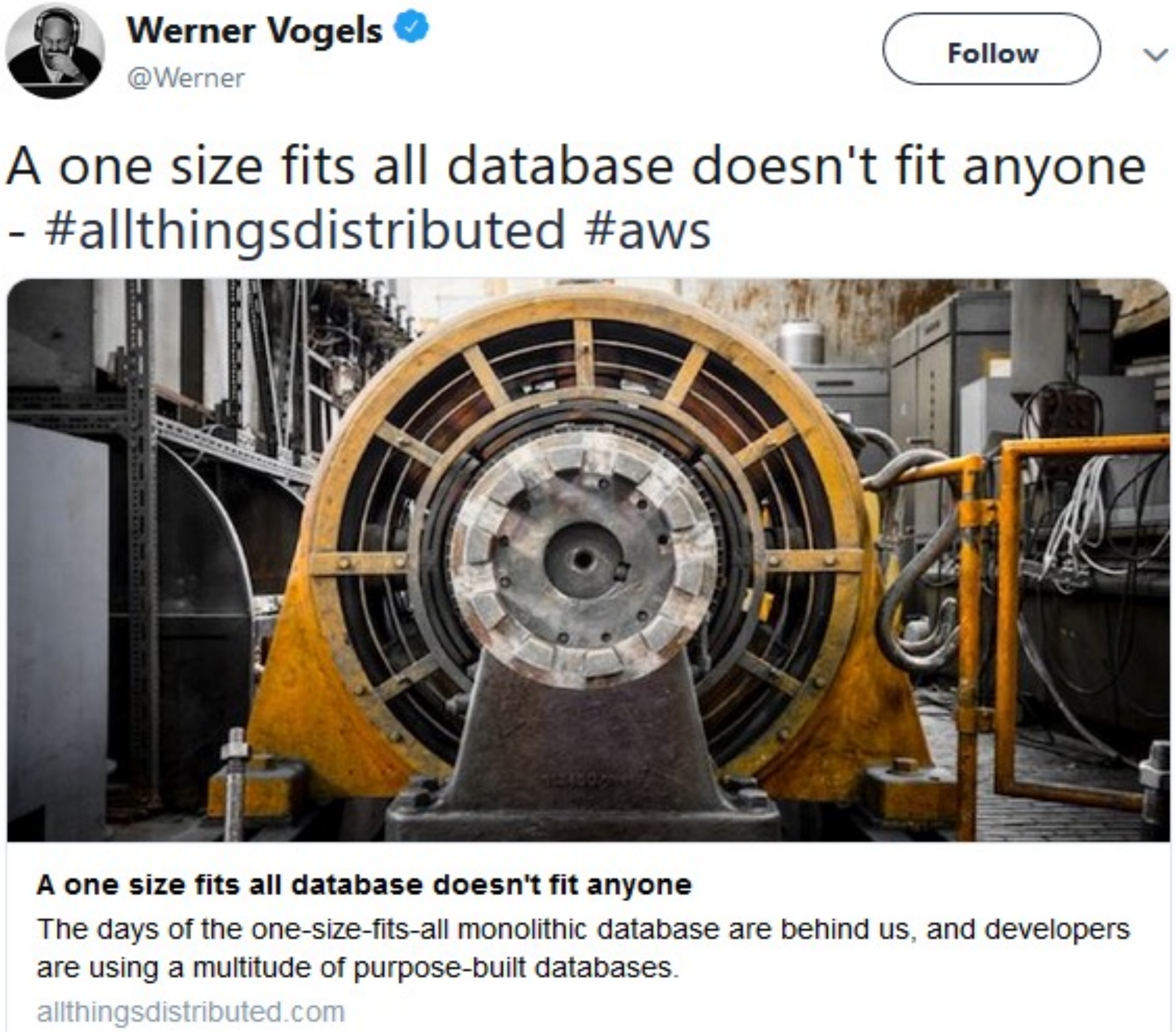
Users	1M+
Data volume	Terabytes—petabytes
Locality	Global
Performance	Microsecond latency
Request rate	Millions per second
Access	Mobile, IoT, devices
Scale	Up-down-out-in
Economics	Pay-as-you-go
Developer access	Open API

To keep up with requirements of modern applications…

customers are shifting to microservice architectures with purpose-built databases.



A one size fits all database doesn’t fit anyone



Werner Vogels
@Werner

A one size fits all database doesn't fit anyone - #allthingsdistributed #aws

A one size fits all database doesn't fit anyone

The days of the one-size-fits-all monolithic database are behind us, and developers are using a multitude of purpose-built databases.

allthingsdistributed.com

Purpose Built Databases

Common data models and use cases

Relational

Key-value

Document

In-memory

Graph

Wide-column

Time-series

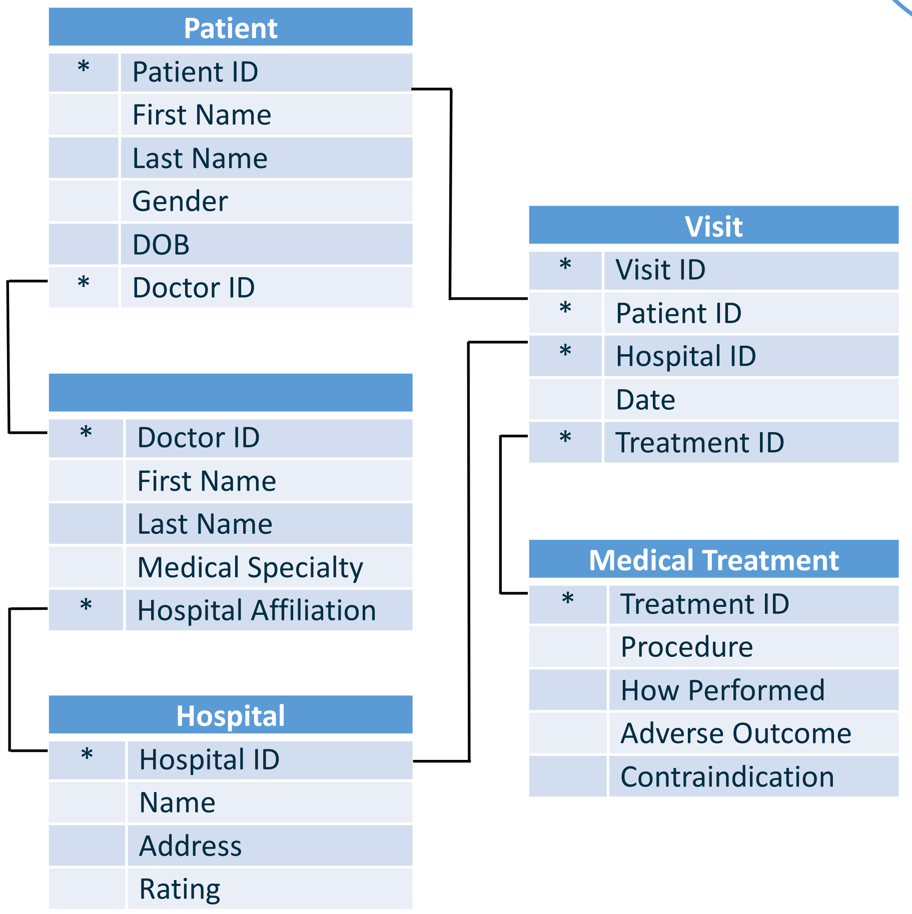
Ledger

Purpose Built Databases: Use cases

Relational Use Case

- Divide data among tables
- Highly structured
- Relationships established via keys
- Data accuracy and consistency enforced by the system

```
SELECT
    d.first_name, d.last_name
FROM
    doctor as d,
    hospital as h
WHERE
    d.hospital = h.hospital_id
    AND h.name =  Mercy
```



	Patient
*	Patient ID
	First Name
	Last Name
	Gender
	DOB
*	Doctor ID

*	Doctor ID
	First Name
	Last Name
	Medical Specialty
*	Hospital Affiliation

	Hospital
*	Hospital ID
	Name
	Address
	Rating

	Visit
*	Visit ID
*	Patient ID
*	Hospital ID
	Date
*	Treatment ID

	Medical Treatment
*	Treatment ID
	Procedure
	How Performed
	Adverse Outcome
	Contraindication

Amazon RDS

Managed relational database service with a choice of popular database engines

Amazon Aurora | MySQL | PostgreSQL | MariaDB | Microsoft SQL Server | ORACLE

Easy to administer

No need to provision infrastructure, install, and maintain DB software

Available & durable

Automatic Multi-AZ data replication; automated backup, snapshots, and failover

Highly scalable

Scale DB compute and storage with a few clicks; minimal downtime for your application

Fast & secure

SSD storage and guaranteed provisioned I/O; data encryption at rest and in transit

Amazon Aurora

MySQL and PostgreSQL-compatible relational database built for the cloud

Performance & scalability

5x throughput of standard MySQL and 3x of standard PostgreSQL; scale-out up to15 read replicas

Availability & durability

Fault-tolerant, self-healing storage; six copies of data across three AZs; continuous backup to S3

Highly secure

Network isolation, encryption at rest/transit

Fully managed

Managed by RDS: no server provisioning, software patching, setup, configuration, or backups

Key-value Use Case

- Simple key value pairs
- Partitioned by keys
- Consistent performance at scale

// Status of Hammer57

```
GET {
  TableName:"Gamers",
  Key: {
    "GamerTag":"Hammer57",
    "Type":"Status” } }
```

// Return all Hammer57

```
QUERY {
  TablaeName:“Gamers”,
  KeyConditionExpression:"GamerTag = :a",
  ExpressionAttributeValues: {
      ":a”:”Hammer57” } }
```

Gamers				
Primary Key		**Attributes**		
Gamer Tag	**Type**			
Hammer57	Rank	Level	Points	Tier
		87	4050	Elite
	Status	Health	Progress	
		90	30	
	Weapon	Class	Damage	Range
		Taser	87%	50
FluffyDuffy	Rank	Level	Points	Tier
		5	1072	Trainee
	Status	Health	Progress	
		37	8	

DynamoDB

Fast and flexible NoSQL database service for any scale

Performance at scale

- Handles millions of requests per second
- Delivers single-digit-millisecond latency
- Automated global replication
- Advanced streaming with Amazon Kinesis Data Streams for Amazon DynamoDB

No servers to manage

- Maintenance free
- Auto scaling
- On-demand capacity mode
- Change data capture for integration with AWS Lambda, Amazon Redshift, and Amazon OpenSearch Service

Enterprise ready

- ACID transactions
- Encryption at rest
- Continuous backups (supporting PITR), and on-demand backup and restore
- Export table data to Amazon S3
- PartiQL (a SQL-compatible query language) support

Wide-column Use Case

Amazon Keyspaces for Apache Cassandra

fully managed Apache Cassandra–compatible database service

Cassandra-compatible

Works with Apache 2.0-licensed Cassandra drivers and developer tools

Fast & scalable

Consistent, single-digit millisecond read and write performance at any scale

Fully managed (Serverless)

No servers to provision, patch, or manage

No software to install, maintain, or operate

Highly available & Secure

Data is replicated 3 times across multiple AWS Availability Zones

Encryption at-rest

Document Use Case

Store, query, & index JSON
data natively

```
for i in response['businesses']:
  col.insert_one(i)
```

```
db.businesses.aggregate([ { $group: { _id: "$price", ratingAvg: { $avg: "$rating"}} } ])
```

```
db.businesses.find({ $and: [{"price" : "$"}, {"rating": { $gt: 4.5}}]})
```

```
db.businesses.createIndex( { review_count: -1 } )
```

Amazon DocumentDB

Scale enterprise workloads with ease using a fully managed native JSON document database

Fast and scalable

- Decoupled compute and storage support independent scaling
- Scale to millions of reads using read replicas for instance-based clusters
- Scale to over a million of writes and reads and 1 PB of storage with Elastic Clusters

Fully managed

- Improve productivity and lower total cost of ownership by removing undifferentiated database management tasks

Enterprise ready

- High availability and durability
- Global Clusters provide local reads and disaster recovery from Region-wide outages
- Built-in security best practices with encryption-in-transit and encryption-at-rest, Amazon VPC, and AWS KMS

MongoDB compatible

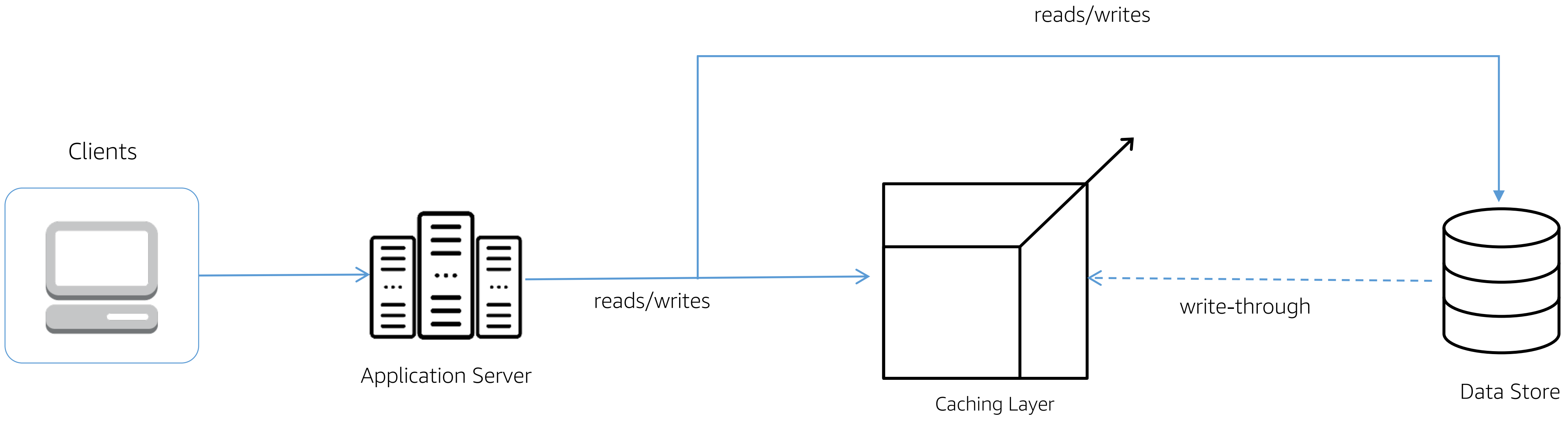
- Supports hundreds of APIs, operators, and data types
- MongoDB APIs, drivers, and tools can be used with little to no change

In-memory Use Case

Latency: Extremely low, microsecond to millisecond responses

Request Rate: Thousands to millions of reads and/or writes per second

Data Volume: Will scale up to over 100TB



Amazon ElastiCache

Fully managed, Redis or Memcached compatible, low-latency, in-memory data store

Extreme performance

In-memory data store and cache for sub-millisecond response times

Fully managed

AWS manages all hardware and software setup, configuration, and monitoring

Easily scalable

Read scaling with replicas

Write and memory scaling with sharding

Nondisruptive scaling

Amazon MemoryDB

Redis-compatible, durable, in-memory database service for ultra-fast performance

Redis compatible

Fully compatible with open source Redis Cluster

Extreme performance

In-memory database for microsecond reads

Secure and reliable

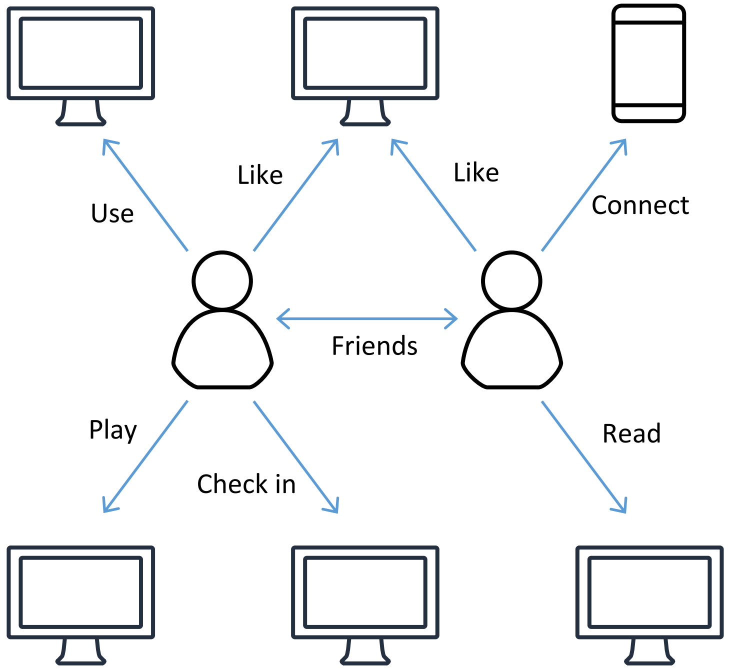
Network isolation, encryption at rest/transit, multi AZ, and automatic failover

Easily scales to massive workloads

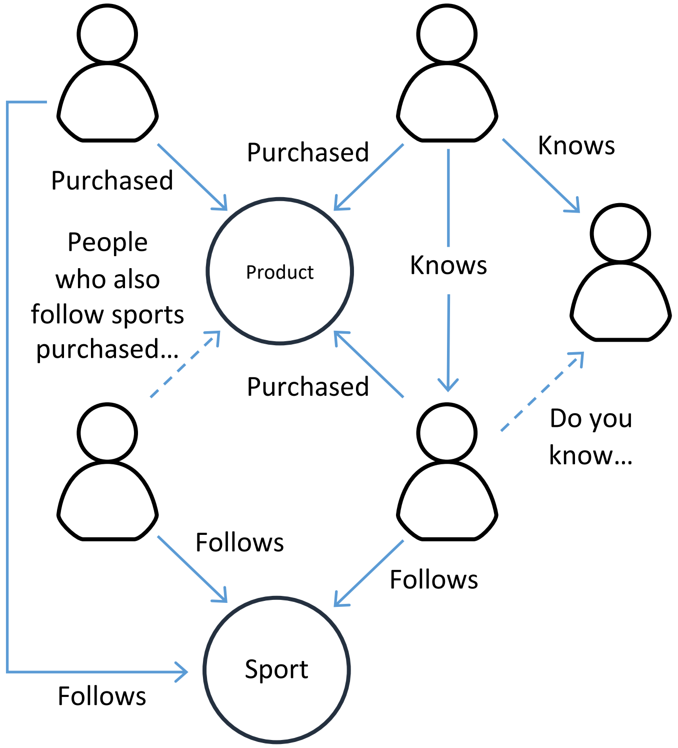
Scale writes and reads with sharding and replicas

Graph Database Use Case

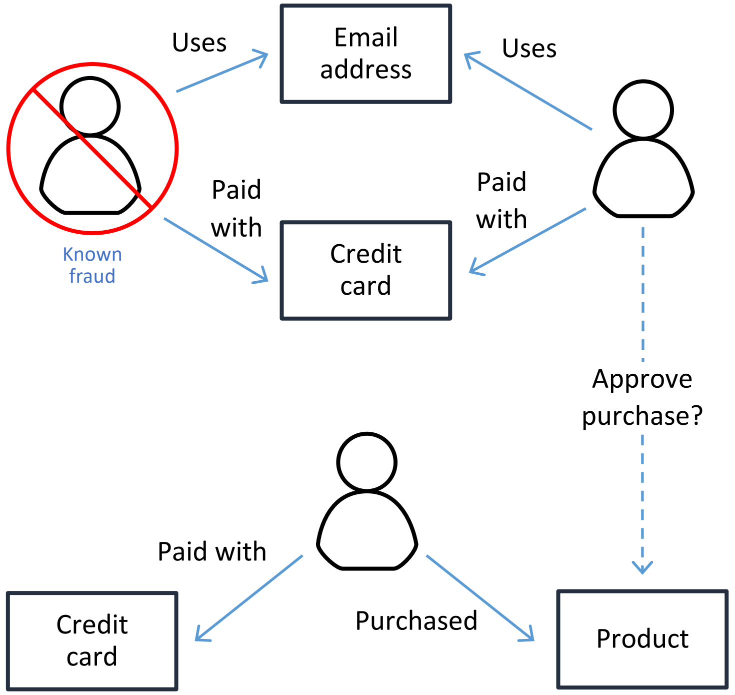
Social news feed



Recommendations



Retail fraud detection



Amazon Neptune

Fast, reliable graph database built for the cloud

Open

Supports Apache TinkerPop & W3C RDF graph models

Fast

Query billions of relationships with millisecond latency

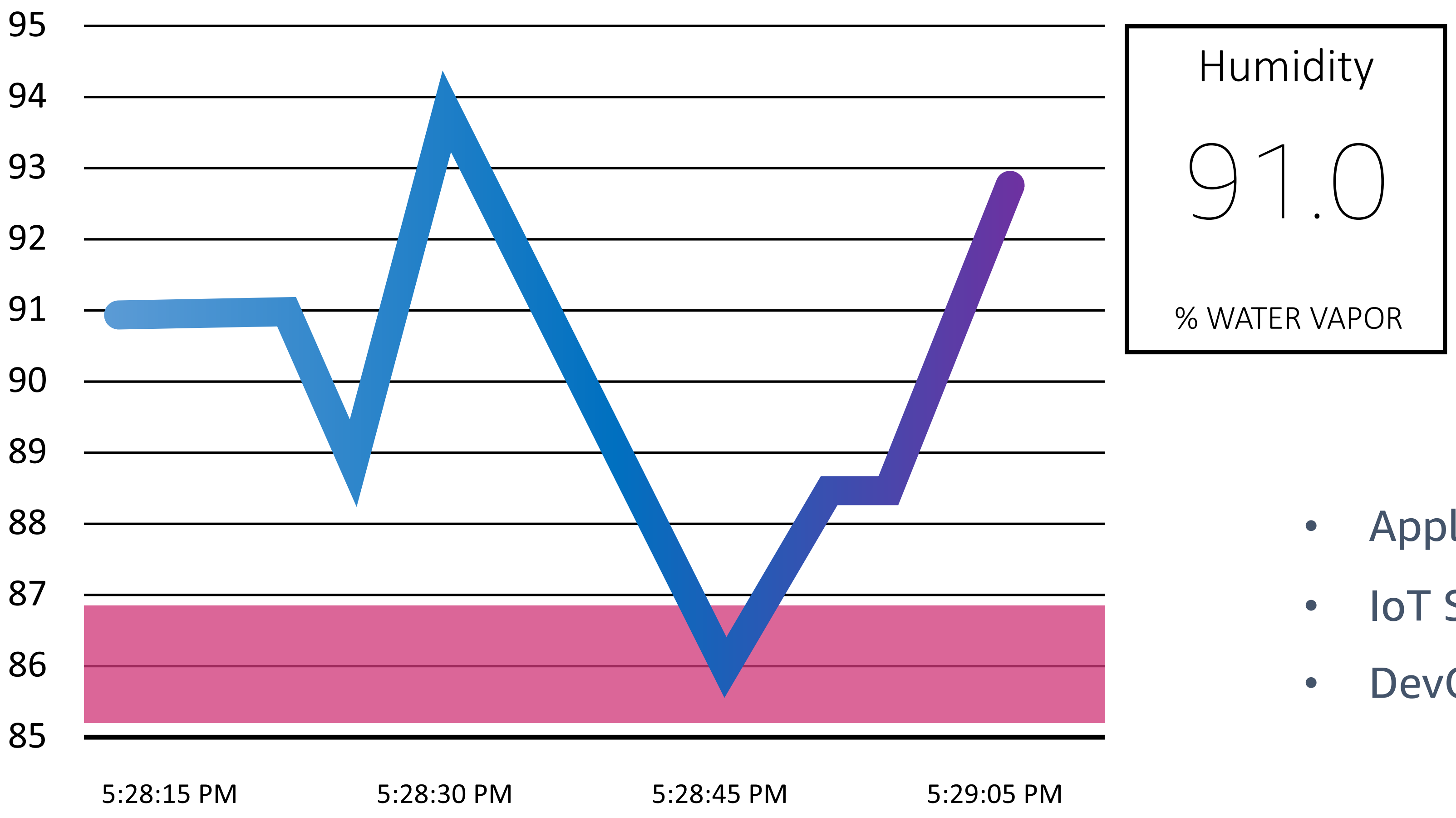
Reliable

6 replicas of your data across 3 AZs with full backup and restore

Easy

Build powerful queries easily with Gremlin and SPARQL

Time-series



- Application events
- IoT Sensor Readings
- DevOps data

Amazon Timestream

serverless time series database service for IoT and operational applications

1,000x faster at 1/10th the cost of relational databases

Collect fast moving time-series data from multiple sources at the rate of millions of inserts per second

Trillions of daily events

Capable of processing trillions of events daily; the adaptive query processing engine maintains steady, predictable performance

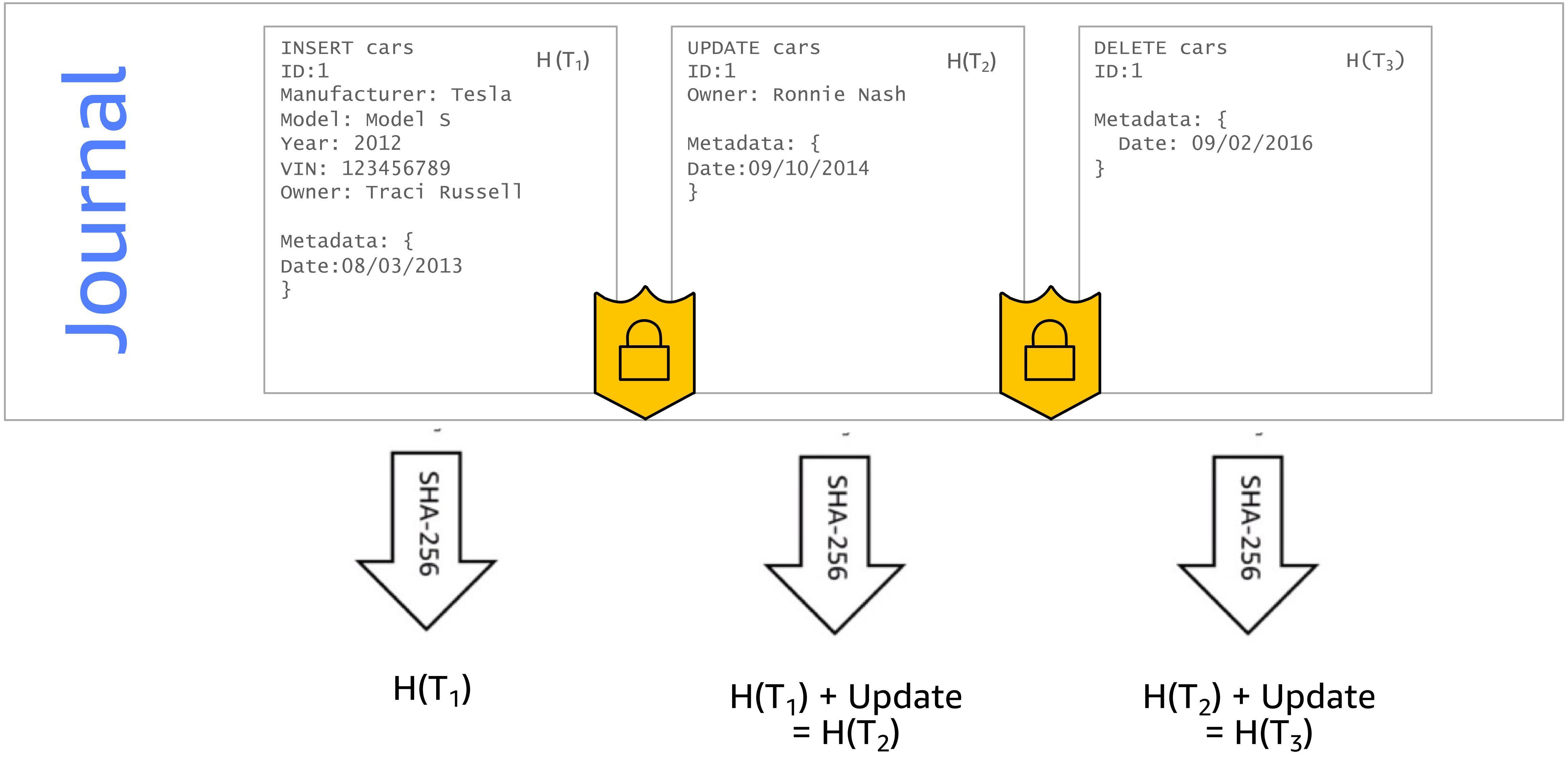
Analytics optimized for time series data

Built-in analytics for interpolation, smoothing, and approximation to identify trends, patterns, and anomalies

Serverless

No servers to manage; time-consuming tasks such as hardware provisioning, software patching, setup, & configuration done for you

Ledger Use Case



```
INSERT cars
ID:1
Manufacturer: Tesla
Model: Model S
Year: 2012
VIN: 123456789
Owner: Traci Russell

Metadata: {
Date:08/03/2013
}
```

$H(T_1)$

```
UPDATE cars
ID:1
Owner: Ronnie Nash

Metadata: {
Date:09/10/2014
}
```

$H(T_2)$

```
DELETE cars
ID:1

Metadata: {
  Date: 09/02/2016
}
```

$H(T_3)$

$H(T_1)$

$H(T_1)$ + Update = $H(T_2)$

$H(T_2)$ + Update = $H(T_3)$

Amazon QLDB

fully managed ledger database

Immutable

Maintains a sequenced record of all changes to your data, which cannot be deleted or modified; you have the ability to query and analyze the full history

Cryptographically verifiable

Uses cryptography to generate a secure output file of your data’s history

Highly scalable

Executes 2–3X as many transactions as ledgers in common blockchain frameworks

Easy to use

Easy to use, letting you use familiar database capabilities like SQL APIs for querying the data

Best Practices with purpose built databases

- Structure of your data
  - Data types
  - Flat vs Json
  - Volume and size of data over time
- Data Modeling
  - Choosing the key for uniqueness and join across data stores
- Access Pattern
  - Key vs Range lookup vs Full search
  - Ad-Hoc/Ops Queries
  - Access pattern changes over time?

- Data consistency and durability
  - Strong vs Eventual
  - Source of truth
  - Data validation / reconciliation
- Performance
  - Concurrency
  - Transaction throughput/response time (p50/p99)
- Data Integration
  - Federated Query
  - ETL
  - Data Lake

Summary

Summary

	Relational	Key-value	Document	In-memory	Graph	Time-series	Ledger	Wide Column
Data Models	Referential integrity, ACID transactions, schema-on-write	High throughput, low-latency reads and writes, endless scale	Store documents and 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
Common Use Cases	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
	Amazon Aurora Amazon Relational Database Service (RDS)	Amazon DynamoDB	Amazon DocumentDB	Amazon ElastiCache Amazon MemoryDB	Amazon Neptune	Amazon Timestream	Amazon QLDB	Amazon Keyspaces (for Apache Cassandra)

Airbnb uses different databases based on the purpose

User search history: **Amazon DynamoDB**

- Massive data volume
- Need quick lookups for personalized search

Session state: **Amazon ElastiCache**

- In-memory store for submillisecond site rendering

Relational data: **Amazon RDS**

- Referential integrity
- Primary transactional database