AWS re:Invent

BDM201

Big Data Architectural Patterns and Best Practices on AWS

Siva Raghupathy, Sr. Manager, Big Data Solutions Architecture, AWS

November 29, 2016

amazon web services

What to Expect from the Session

Big data challenges

Architectural principles

How to simplify big data processing

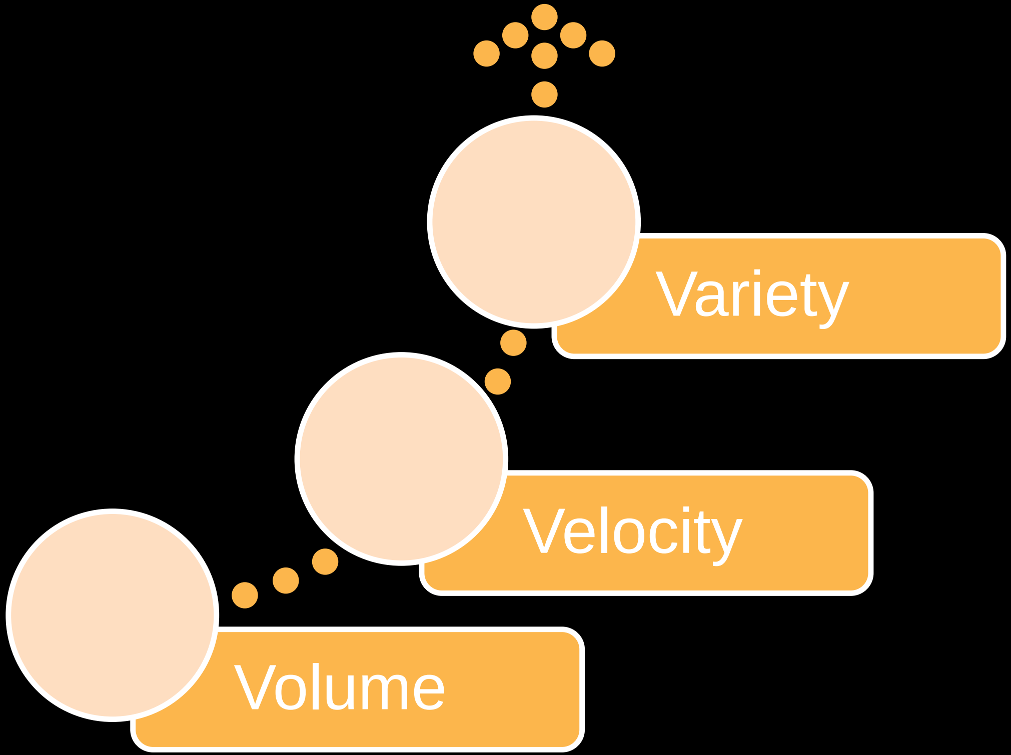
What technologies should you use?

- Why?
- How?

Reference architecture

Design patterns

Ever-Increasing Big Data



Big Data Evolution

Batch processing

Stream processing

Machine learning

Cloud Services Evolution

Virtual machines

Managed services

Serverless

Plethora of Tools

EMR

S3

DynamoDB

SQS

Amazon Redshift

Amazon Glacier

RDS

ElastiCache

Amazon Kinesis

Amazon Kinesis Streams app

Data Pipeline

Amazon Elasticsearch Service

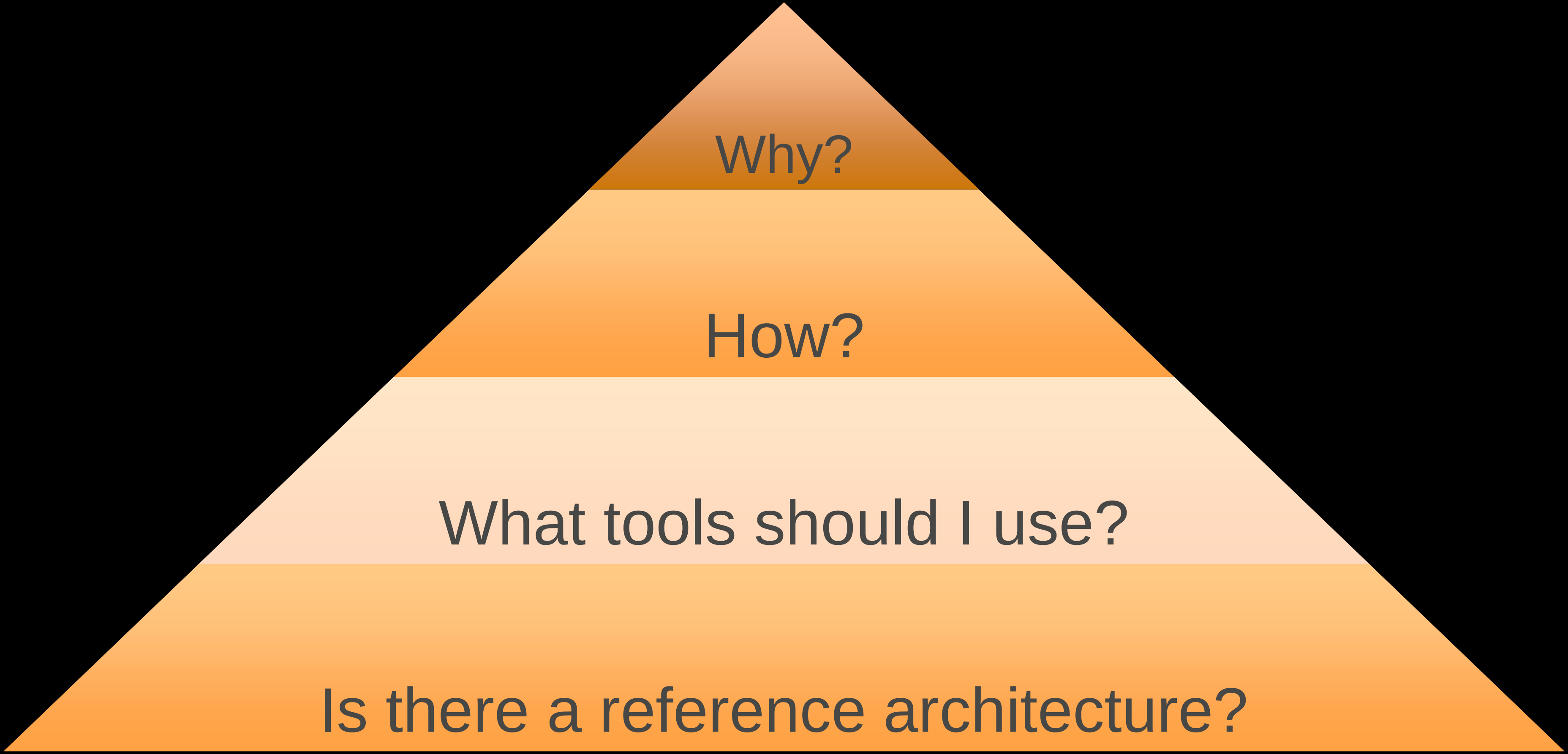
Lambda

Amazon ML

DynamoDB Streams

Amazon Kinesis Analytics

Big Data Challenges



Why?

How?

What tools should I use?

Is there a reference architecture?

Architectural Principles

Build decoupled systems

- Data → Store → Process → Store → Analyze → Answers

Use the right tool for the job

- Data structure, latency, throughput, access patterns

Leverage AWS managed services

- Scalable/elastic, available, reliable, secure, no/low admin

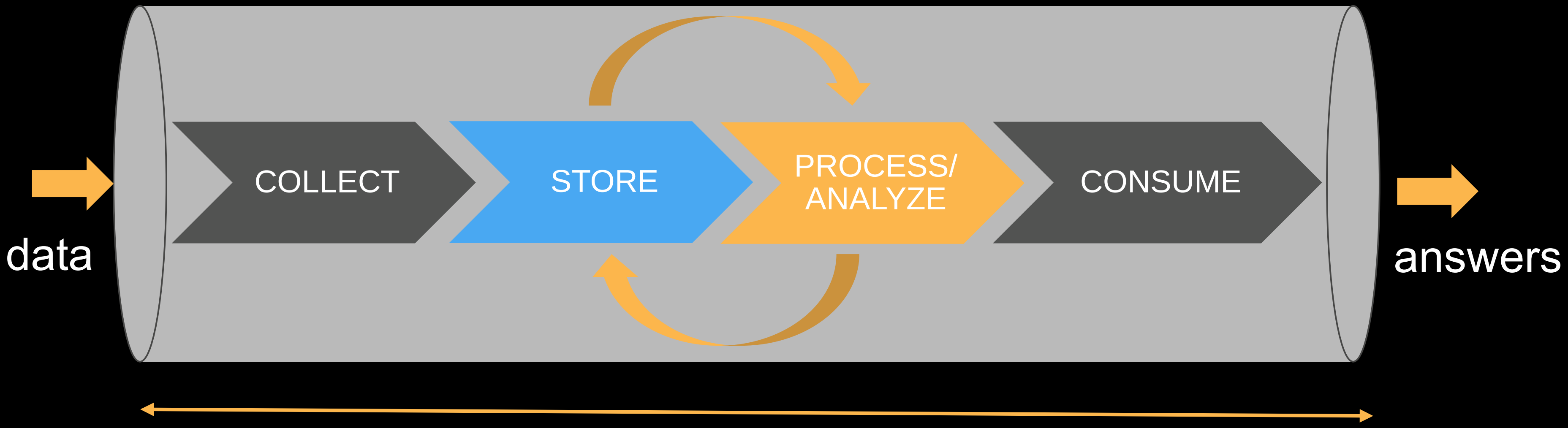
Use log-centric design patterns

- Immutable logs, materialized views

Be cost-conscious

- Big data ≠ big cost

Simplify Big Data Processing

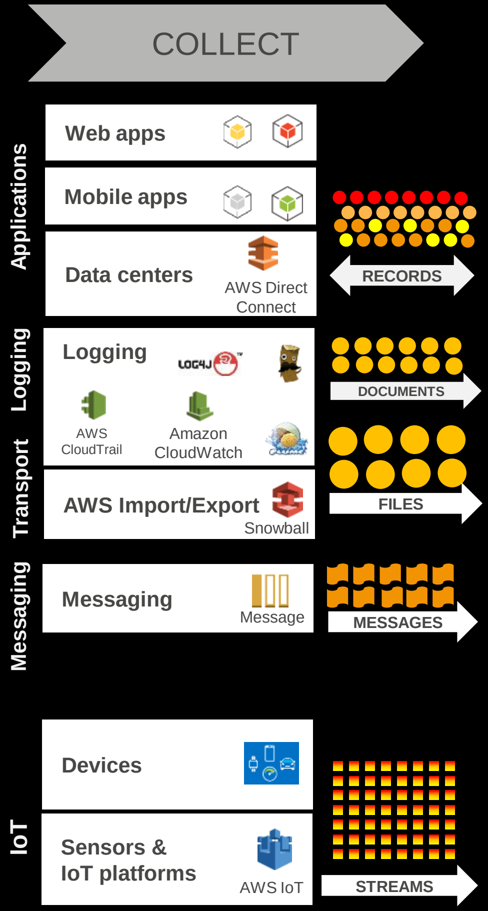


Time to answer (Latency)

Throughput

Cost

COLLECT



Types of Data

In-memory data structures

Database records

Transactions

Search documents

Log files

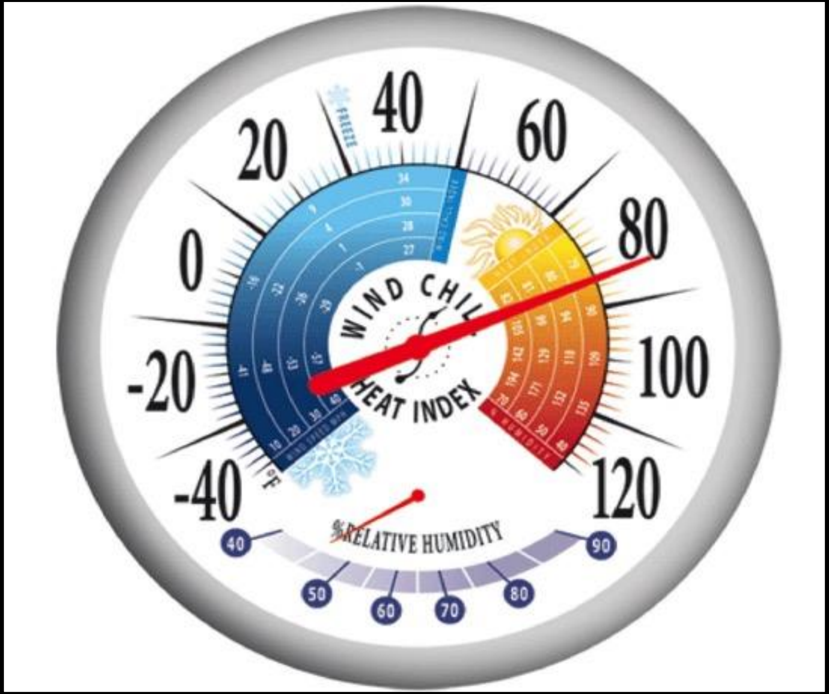
Files

Messages

Data streams

Events

What Is the Temperature of Your Data ?

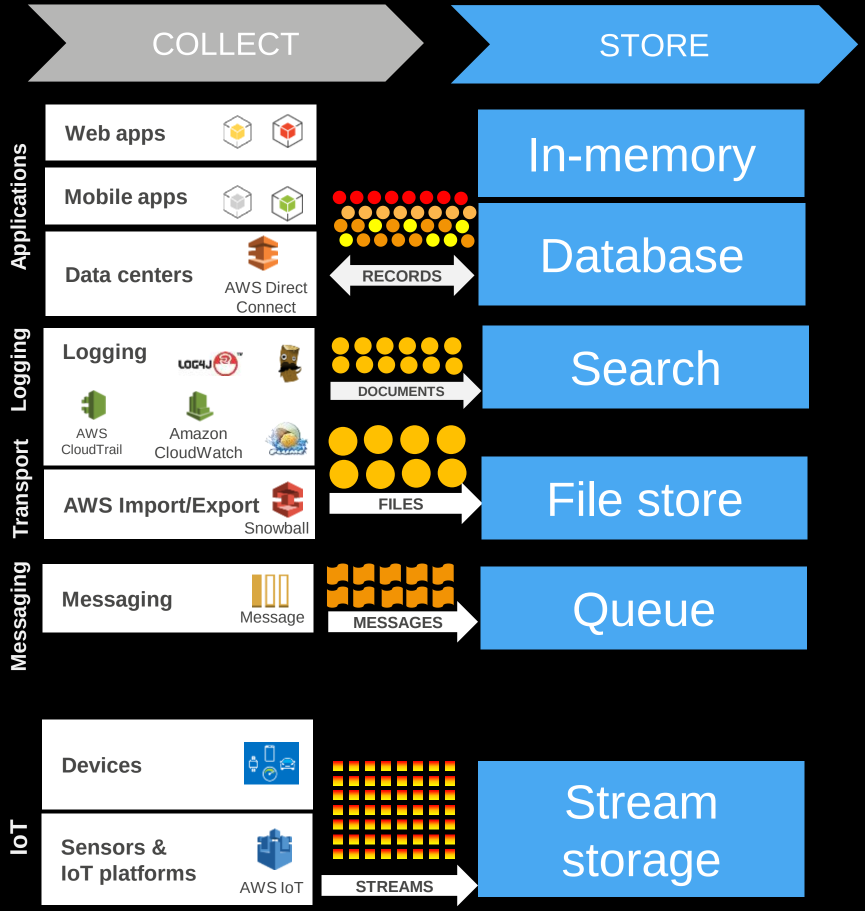


Data Characteristics: Hot, Warm, Cold

	Hot	Warm	Cold
Volume	MB–GB	GB–TB	PB–EB
Item size	B–KB	KB–MB	KB–TB
Latency	ms	ms, sec	min, hrs
Durability	Low–high	High	Very high
Request rate	Very high	High	Low
Cost/GB	$$-$	$-¢¢	¢

Hot data — Warm data — Cold data

Store



COLLECT

STORE

Applications

Web apps

Mobile apps

Data centers

AWS Direct Connect

Logging

Logging

AWS CloudTrail

Amazon CloudWatch

Transport

AWS Import/Export

Snowball

Messaging

Messaging

Message

IoT

Devices

Sensors & IoT platforms

AWS IoT

RECORDS

DOCUMENTS

FILES

MESSAGES

STREAMS

In-memory

Database

Search

File store

Queue

Stream storage

Types of Data Stores

Caches, data structure servers

SQL & NoSQL databases

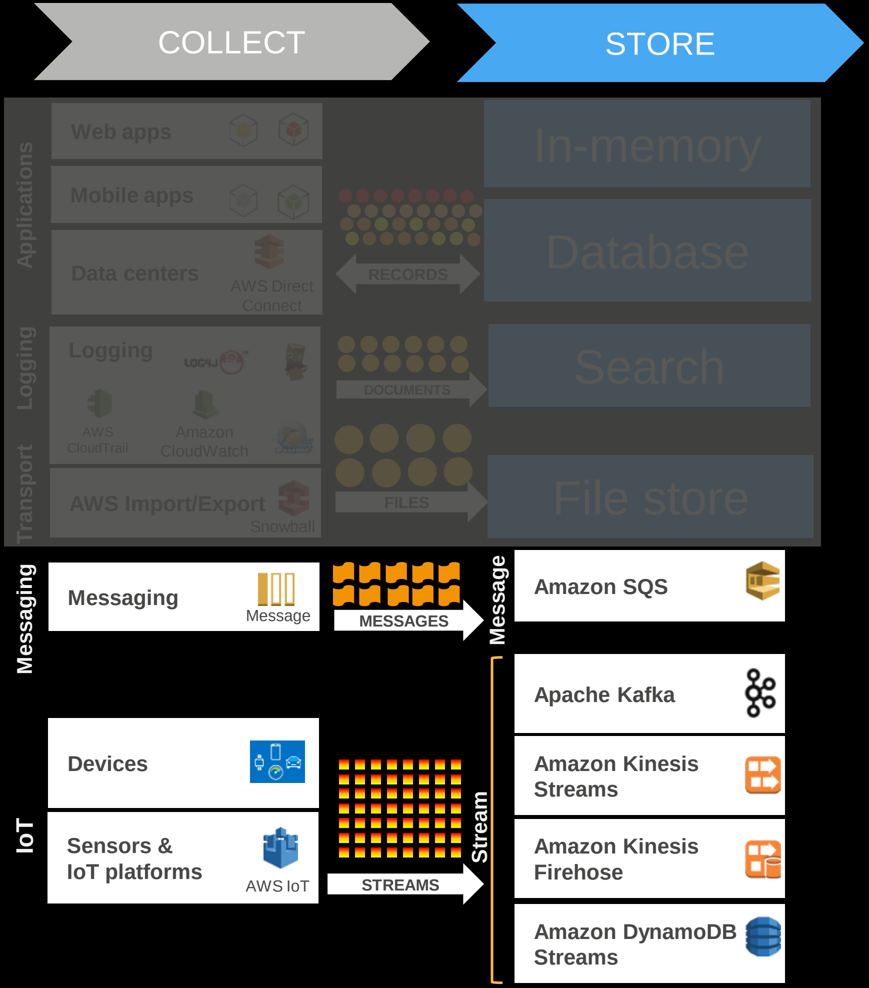
Search engines

File systems

Message queues

Pub/sub message queues

Message & Stream Storage



Amazon SQS

- Managed message queue service

Apache Kafka

- High throughput distributed streaming platform

Amazon Kinesis Streams

- Managed stream storage + processing

Amazon Kinesis Firehose

- Managed data delivery

Amazon DynamoDB

- Managed NoSQL database
- Tables can be stream-enabled

Why Stream Storage?

Decouple producers & consumers

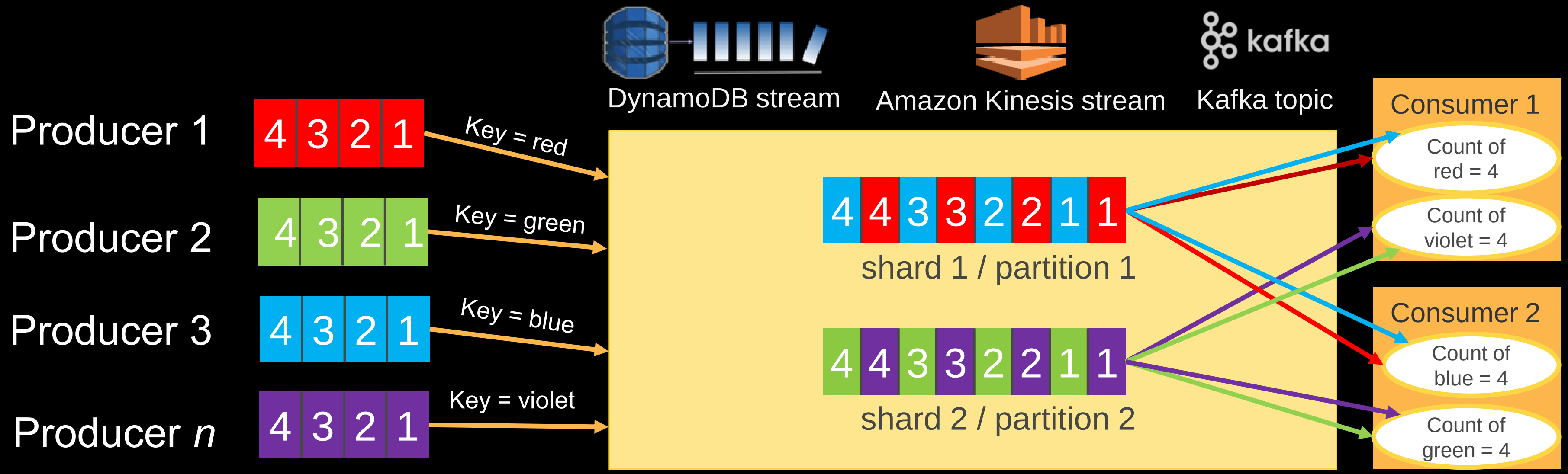
Persistent buffer

Collect multiple streams

Preserve client ordering

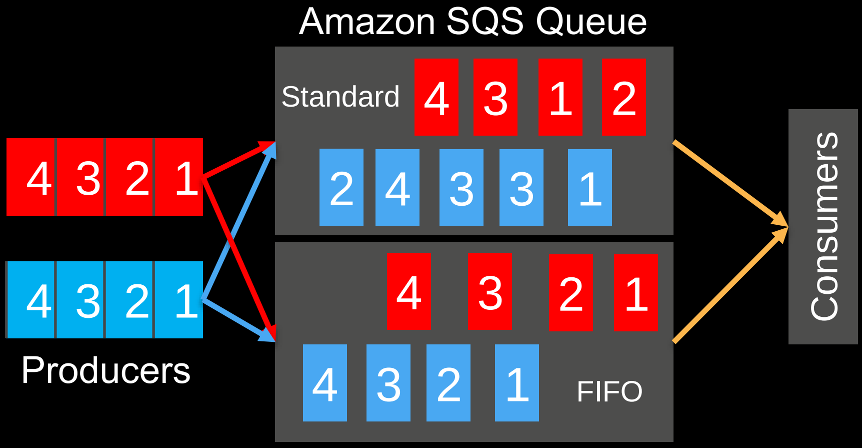
Parallel consumption

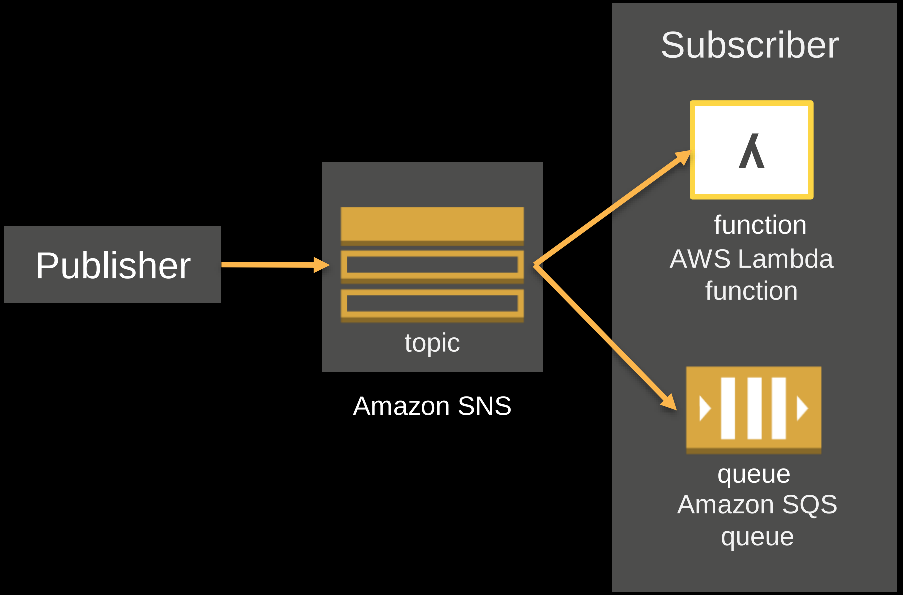
Streaming MapReduce



What About Amazon SQS?

- Decouple producers & consumers
- Persistent buffer
- Collect multiple streams
- **No** client ordering (Standard)
  - FIFO queue preserves client ordering
- **No** streaming MapReduce
- **No** parallel consumption
  - Amazon SNS can publish to multiple SNS subscribers (queues or λ functions)





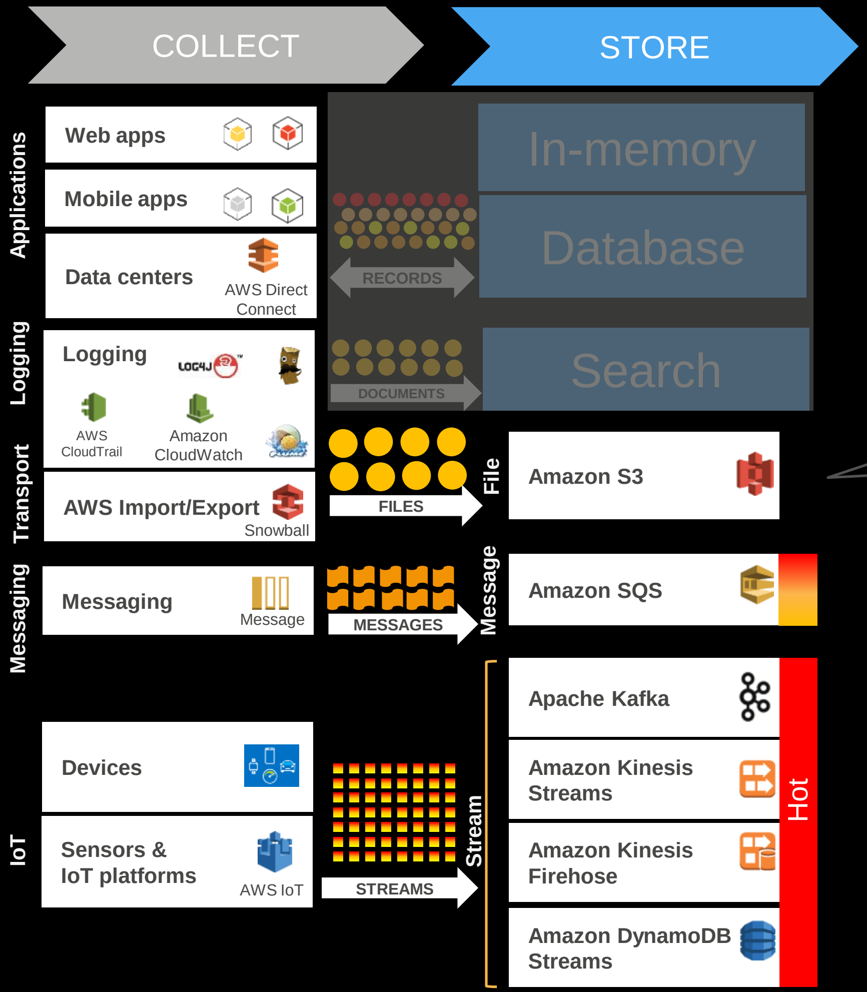
Which Stream/Message Storage Should I Use?

	Amazon DynamoDB Streams	Amazon Kinesis Streams	Amazon Kinesis Firehose	Apache Kafka	Amazon SQS (Standard)	Amazon SQS (FIFO) New
AWS managed	Yes	Yes	Yes	No	Yes	Yes
Guaranteed ordering	Yes	Yes	No	Yes	No	Yes
Delivery (deduping)	Exactly-once	At-least-once	At-least-once	At-least-once	At-least-once	Exactly-once
Data retention period	24 hours	7 days	N/A	Configurable	14 days	14 days
Availability	3 AZ	3 AZ	3 AZ	Configurable	3 AZ	3 AZ
Scale / throughput	No limit / ~ table IOPS	No limit / ~ shards	No limit / automatic	No limit / ~ nodes	No limits / automatic	300 TPS / queue
Parallel consumption	Yes	Yes	No	Yes	No	No
Stream MapReduce	Yes	Yes	N/A	Yes	N/A	N/A
Row/object size	400 KB	1 MB	Destination row/object size	Configurable	256 KB	256 KB
Cost	Higher (table cost)	Low	Low	Low (+admin)	Low-medium	Low-medium

Hot

Warm

File Storage



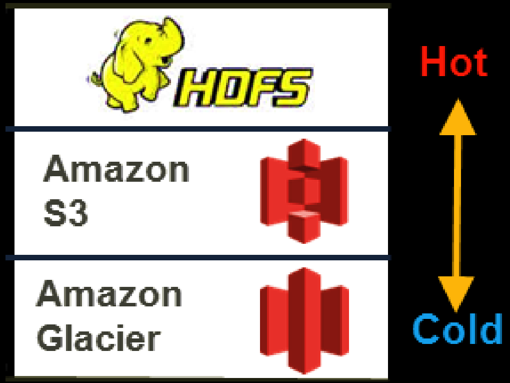
Amazon S3

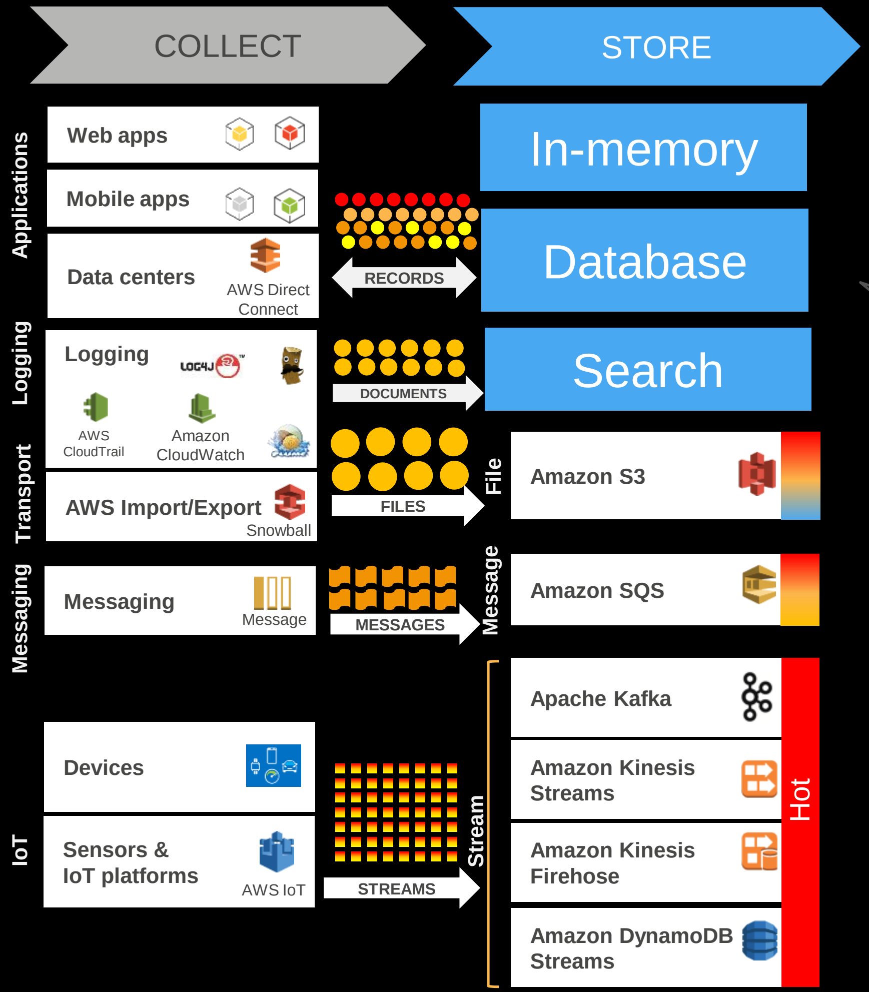
Why Is Amazon S3 Good for Big Data?

- Natively supported by big data frameworks (Spark, Hive, Presto, etc.)
- No need to run compute clusters for storage (unlike HDFS)
- Can run transient Hadoop clusters & Amazon EC2 Spot Instances
- Multiple & heterogeneous analysis clusters can use the same data
- Unlimited number of objects and volume of data
- Very high bandwidth – no aggregate throughput limit
- Designed for 99.99% availability – can tolerate zone failure
- Designed for 99.999999999% durability
- No need to pay for data replication
- Native support for versioning
- Tiered-storage (Standard, IA, Amazon Glacier) via life-cycle policies
- Secure – SSL, client/server-side encryption at rest
- Low cost

What About HDFS & Data Tiering?

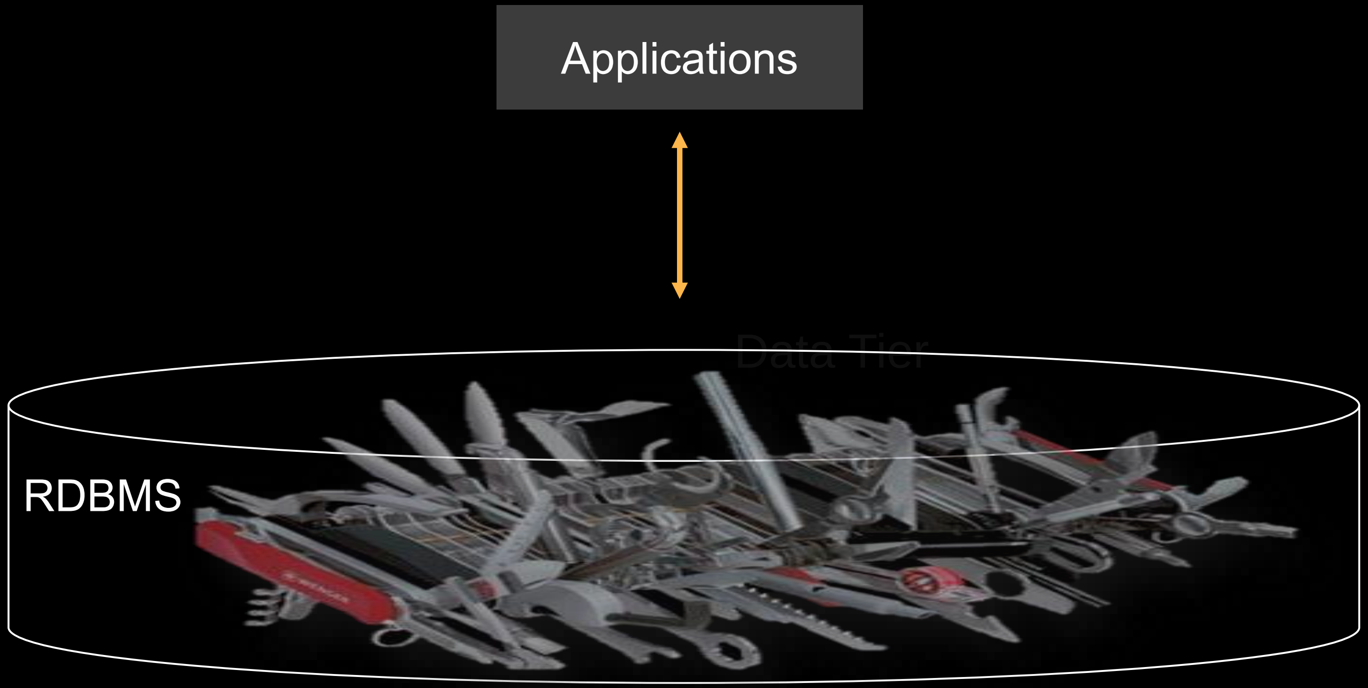
- Use HDFS for very frequently accessed (hot) data
- Use Amazon S3 Standard for frequently accessed data
- Use Amazon S3 Standard – IA for less frequently accessed data
- Use Amazon Glacier for archiving cold data





In-memory, Database, Search

Anti-Pattern

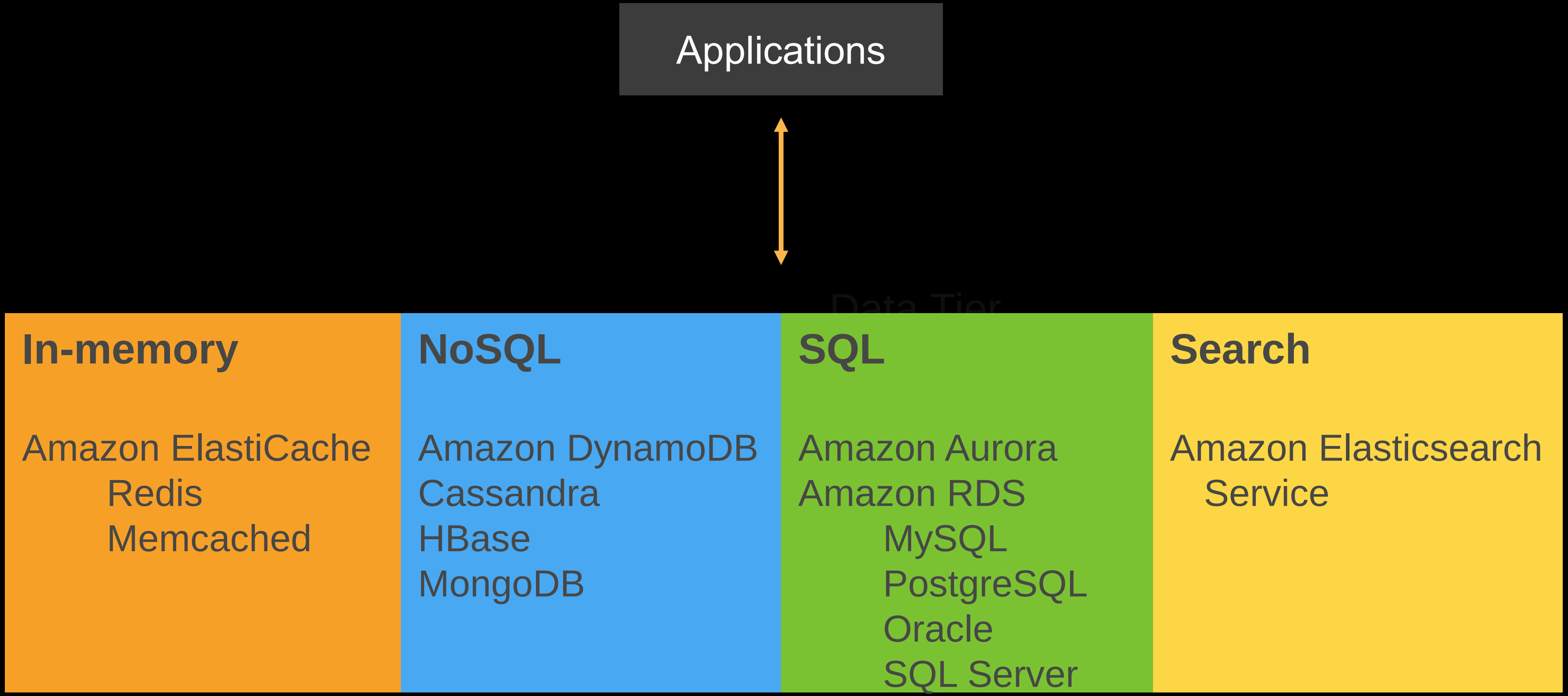


Applications

Data Tier

RDBMS

Best Practice: Use the Right Tool for the Job



Applications

Data Tier

In-memory

- Amazon ElastiCache
  - Redis
  - Memcached

NoSQL

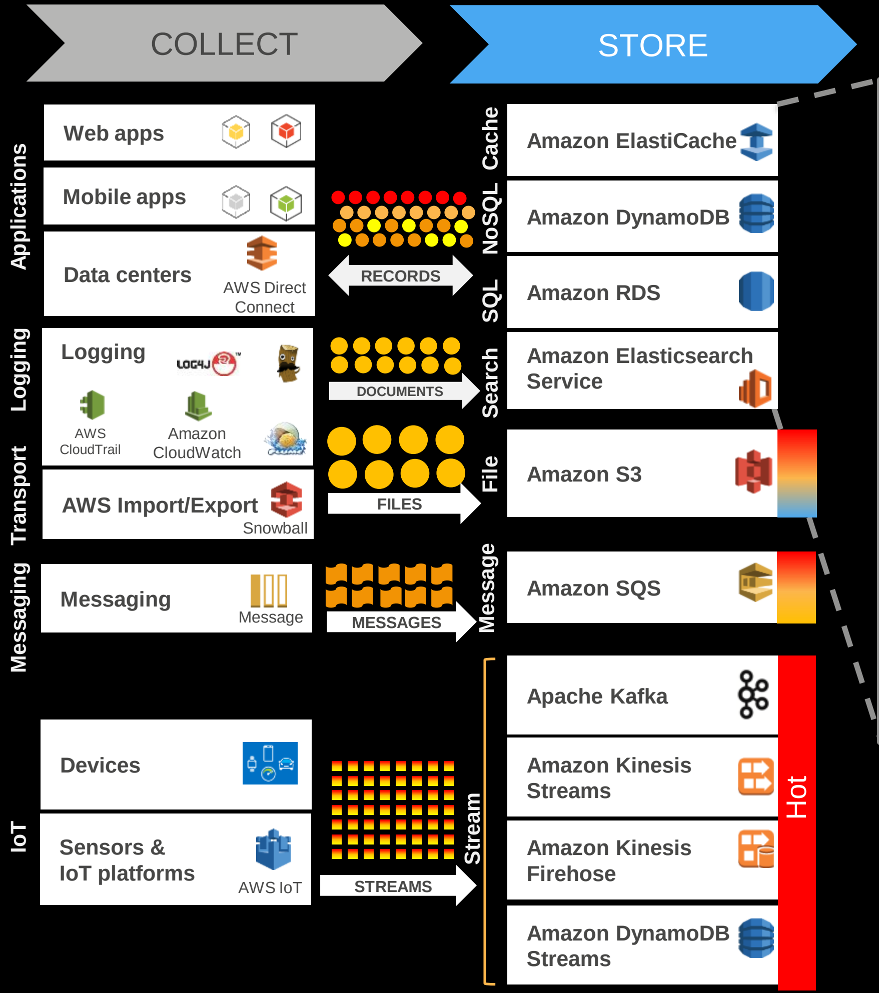
- Amazon DynamoDB
- Cassandra
- HBase
- MongoDB

SQL

- Amazon Aurora
- Amazon RDS
  - MySQL
  - PostgreSQL
  - Oracle
  - SQL Server

Search

- Amazon Elasticsearch Service



Amazon ElastiCache

- Managed Memcached or Redis service

Amazon DynamoDB

- Managed NoSQL database service

Amazon RDS

- Managed relational database service

Amazon Elasticsearch Service

- Managed Elasticsearch service

What Data Store Should I Use?

Data structure → Fixed schema, JSON, key-value

Access patterns → Store data in the format you will access it

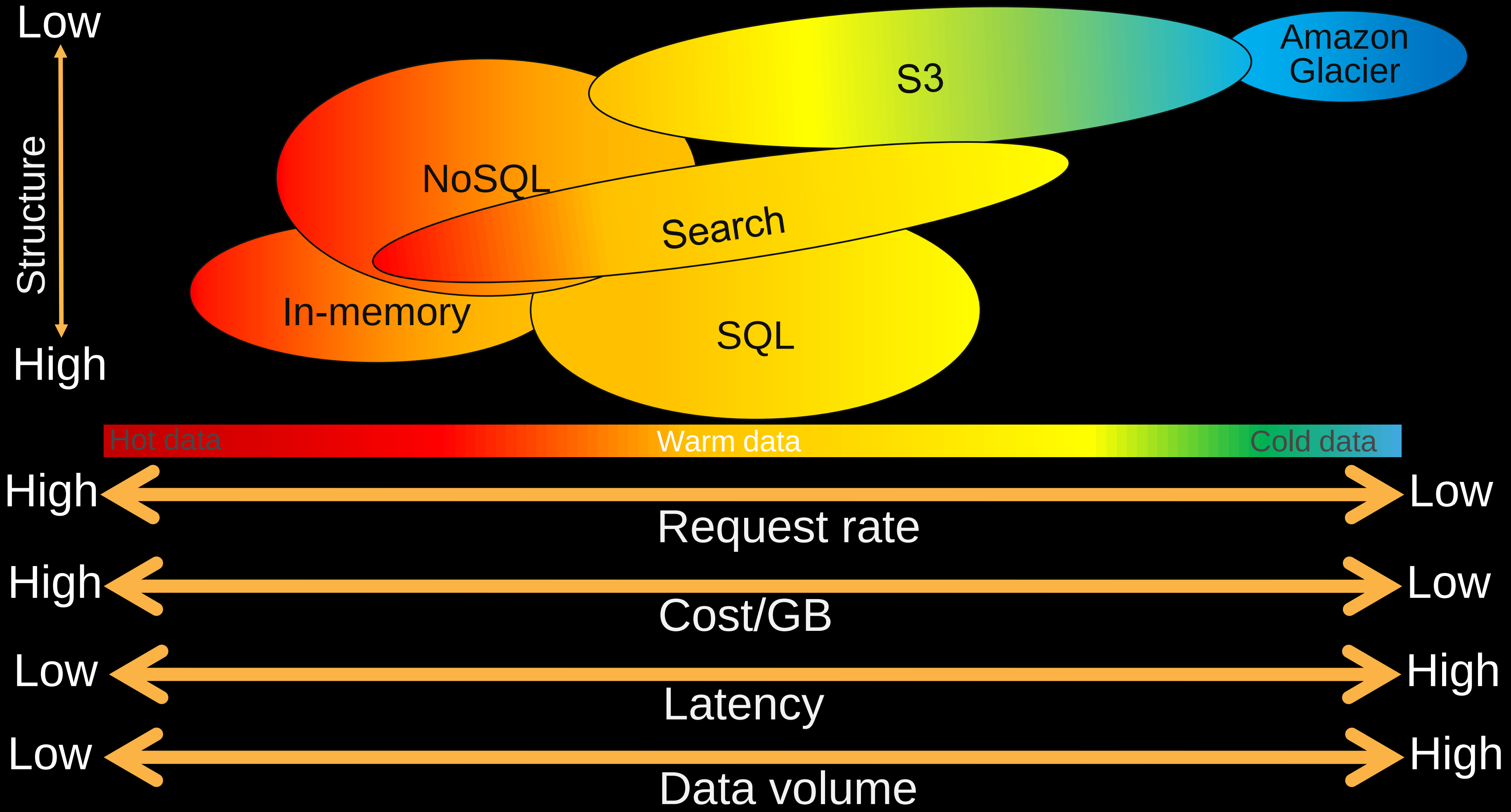
Data characteristics → Hot, warm, cold

Cost → Right cost

Data Structure and Access Patterns

Access Patterns	What to use?
Put/Get (key, value)	In-memory, NoSQL
Simple relationships → 1:N, M:N	NoSQL
Multi-table joins, transaction, SQL	SQL
Faceting, search	Search

Data Structure	What to use?
Fixed schema	SQL, NoSQL
Schema-free (JSON)	NoSQL, Search
(Key, value)	In-memory, NoSQL



Low
Structure
High
NoSQL
S3
Amazon Glacier
Search
In-memory
SQL
Hot data
Warm data
Cold data
High
Request rate
Low
High
Cost/GB
Low
Low
Latency
High
Low
Data volume
High

Which Data Store Should I Use?

	Amazon ElastiCache	Amazon DynamoDB	Amazon RDS/Aurora	Amazon ES	Amazon S3	Amazon Glacier
Average latency	ms	ms	ms, sec	ms,sec	ms,sec,min (~ size)	hrs
Typical data stored	GB	GB–TBs (no limit)	GB–TB (64 TB max)	GB–TB	MB–PB (no limit)	GB–PB (no limit)
Typical item size	B-KB	KB (400 KB max)	KB (64 KB max)	B-KB (2 GB max)	KB-TB (5 TB max)	GB (40 TB max)
Request Rate	High – very high	Very high (no limit)	High	High	Low – high (no limit)	Very low
Storage cost GB/month	$$	¢¢	¢¢	¢¢	¢	¢4/10
Durability	Low - moderate	Very high	Very high	High	Very high	Very high
Availability	High 2 AZ	Very high 3 AZ	Very high 3 AZ	High 2 AZ	Very high 3 AZ	Very high 3 AZ

Hot data Warm data Cold data

Cost-Conscious Design

Example: Should I use Amazon S3 or Amazon DynamoDB?

“I’m currently scoping out a project. The design calls for **many small files**, perhaps up to a **billion during peak**. The **total size** would be on the order of **1.5 TB per month**…”

Request rate (Writes/sec)	Object size (Bytes)	Total size (GB/month)	Objects per month
300	2048	1483	777,600,000

Cost-Conscious Design

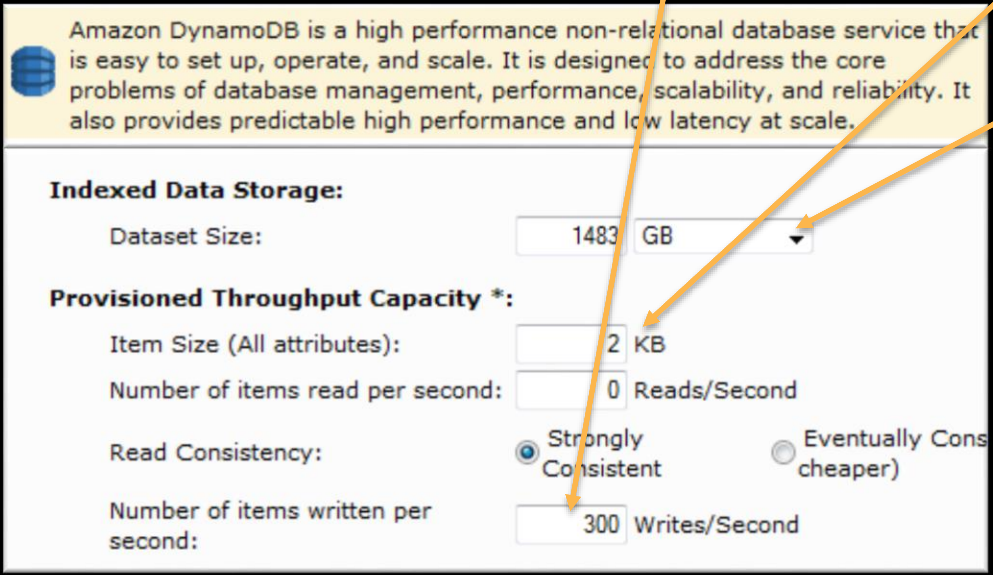
Example: Should I use Amazon S3 or Amazon DynamoDB?

amazon web services | Simple Monthly Calculator

https://calculator.s3.amazonaws.com/index.html

Amazon S3 or DynamoDB?

Request rate (Writes/sec)	Object size (Bytes)	Total size (GB/month)	Objects per month
300	2,048	1,483	777,600,000



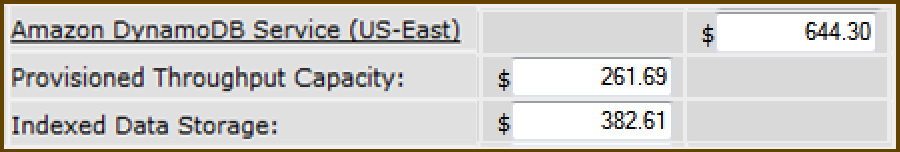
Amazon DynamoDB is a high performance non-relational database service that is easy to set up, operate, and scale. It is designed to address the core problems of database management, performance, scalability, and reliability. It also provides predictable high performance and low latency at scale.

Indexed Data Storage:

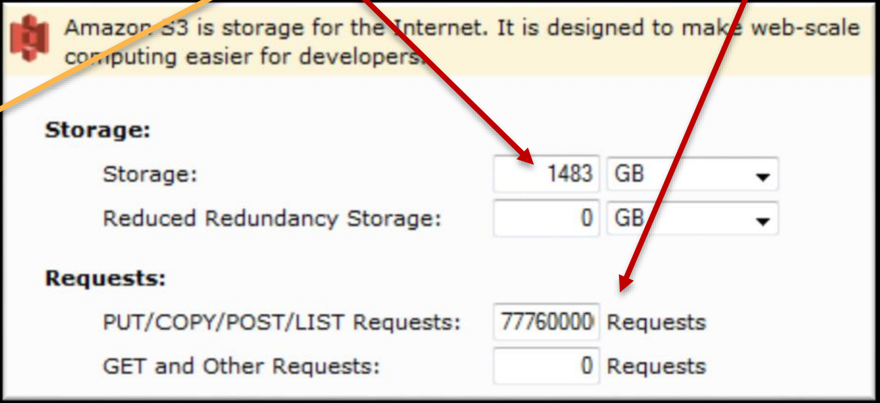
Dataset Size:	1483	GB

Provisioned Throughput Capacity *:

Item Size (All attributes):	2	KB
Number of items read per second:	0	Reads/Second
Read Consistency:	● Strongly Consistent	○ Eventually Cons cheaper)
Number of items written per second:	300	Writes/Second



Amazon DynamoDB Service (US-East)		$ 644.30
Provisioned Throughput Capacity:	$ 261.69	
Indexed Data Storage:	$ 382.61	



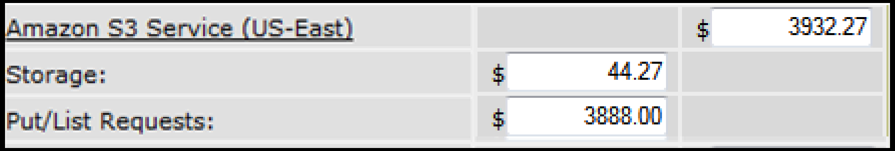
Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers.

Storage:

Storage:	1483	GB
Reduced Redundancy Storage:	0	GB

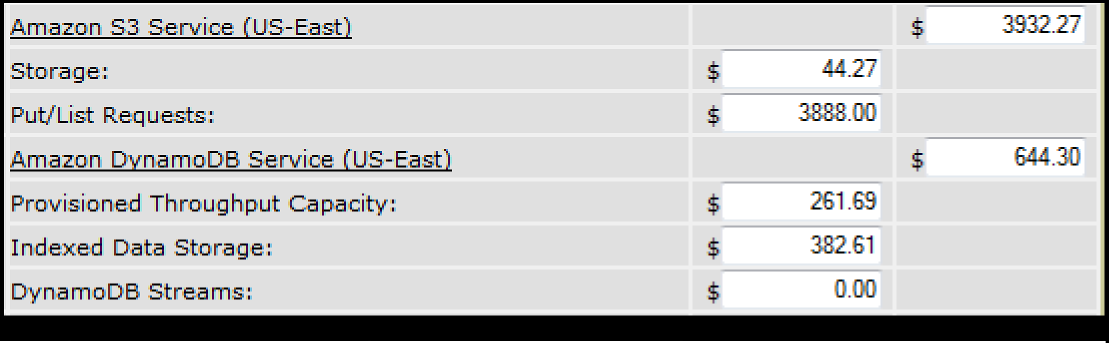
Requests:

PUT/COPY/POST/LIST Requests:	77760000	Requests
GET and Other Requests:	0	Requests

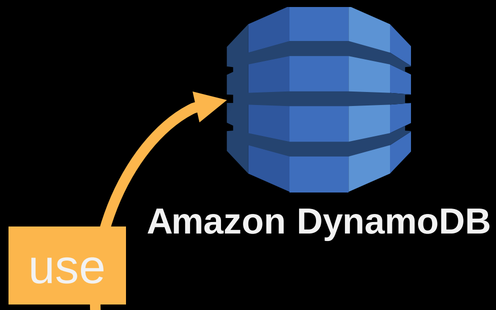


Amazon S3 Service (US-East)		$ 3932.27
Storage:	$ 44.27	
Put/List Requests:	$ 3888.00	

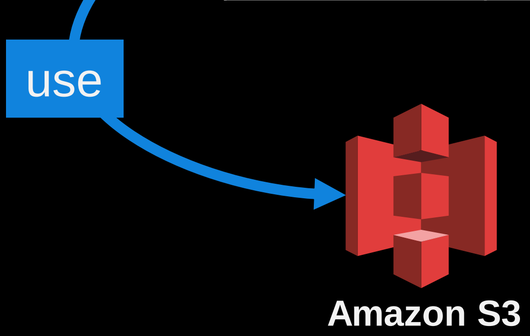
amazon web services SIMPLE MONTHLY CALCULATOR

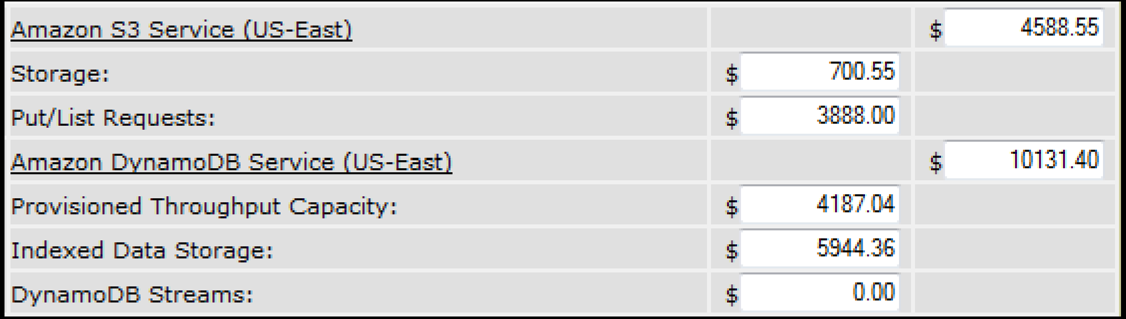


Amazon S3 Service (US-East)		$ 3932.27
Storage:	$ 44.27	
Put/List Requests:	$ 3888.00	
Amazon DynamoDB Service (US-East)		$ 644.30
Provisioned Throughput Capacity:	$ 261.69	
Indexed Data Storage:	$ 382.61	
DynamoDB Streams:	$ 0.00	



	Request rate (Writes/sec)	Object size (Bytes)	Total size (GB/month)	Objects per month
Scenario 1	300	2,048	1,483	777,600,000
Scenario 2	300	32,768	23,730	777,600,000





Amazon S3 Service (US-East)		$ 4588.55
Storage:	$ 700.55	
Put/List Requests:	$ 3888.00	
Amazon DynamoDB Service (US-East)		$ 10131.40
Provisioned Throughput Capacity:	$ 4187.04	
Indexed Data Storage:	$ 5944.36	
DynamoDB Streams:	$ 0.00	

PROCESS / ANALYZE

Analytics Types & Frameworks

Batch

- Takes minutes to hours
- Example: Daily/weekly/monthly reports
- Amazon EMR (MapReduce, Hive, Pig, Spark)

Interactive

- Takes seconds
- Example: Self-service dashboards
- Amazon Redshift, Amazon EMR (Presto, Spark)

Message

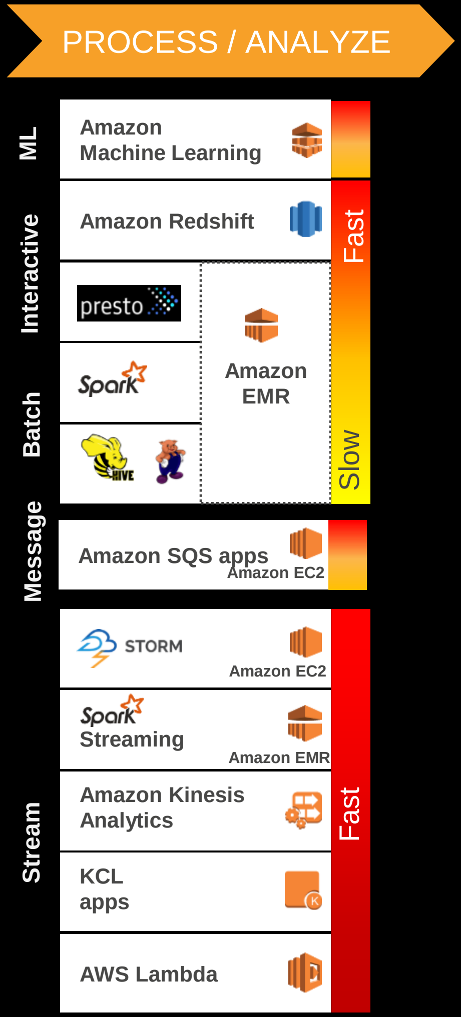
- Takes milliseconds to seconds
- Example: Message processing
- Amazon SQS applications on Amazon EC2

Stream

- Takes milliseconds to seconds
- Example: Fraud alerts, 1 minute metrics
- Amazon EMR (Spark Streaming), Amazon Kinesis Analytics, KCL, Storm, AWS Lambda

Machine Learning

- Takes milliseconds to minutes
- Example: Fraud detection, forecast demand
- Amazon ML, Amazon EMR (Spark ML)



PROCESS / ANALYZE

ML: Amazon Machine Learning

Interactive: Amazon Redshift

Interactive / Batch: presto, Spark, HIVE — Amazon EMR

Fast / Slow

Message: Amazon SQS apps, Amazon EC2

Stream: STORM (Amazon EC2), Spark Streaming (Amazon EMR), Amazon Kinesis Analytics, KCL apps, AWS Lambda

Fast

Which Stream & Message Processing Technology Should I Use?

	Amazon EMR (Spark Streaming)	Apache Storm	KCL Application	Amazon Kinesis Analytics	AWS Lambda	Amazon SQS Application
AWS managed	Yes (Amazon EMR)	No (Do it yourself)	No (EC2 + Auto Scaling)	Yes	Yes	No (EC2 + Auto Scaling)
Serverless	No	No	No	Yes	Yes	No
Scale / throughput	No limits / ~ nodes	No limits / ~ nodes	No limits / ~ nodes	Up to 8 KPU / automatic	No limits / automatic	No limits / ~ nodes
Availability	Single AZ	Configurable	Multi-AZ	Multi-AZ	Multi-AZ	Multi-AZ
Programming languages	Java, Python, Scala	Almost any language via Thrift	Java, others via MultiLangDaemon	ANSI SQL with extensions	Node.js, Java, Python	AWS SDK languages (Java, .NET, Python, …)
Uses	Multistage processing	Multistage processing	Single stage processing	Multistage processing	Simple event-based triggers	Simple event based triggers
Reliability	KCL and Spark checkpoints	Framework managed	Managed by KCL	Managed by Amazon Kinesis Analytics	Managed by AWS Lambda	Managed by SQS Visibility Timeout

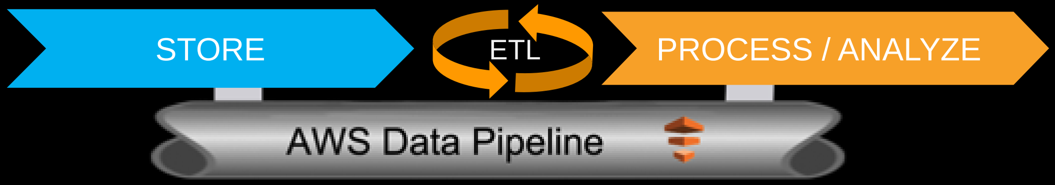
Fast

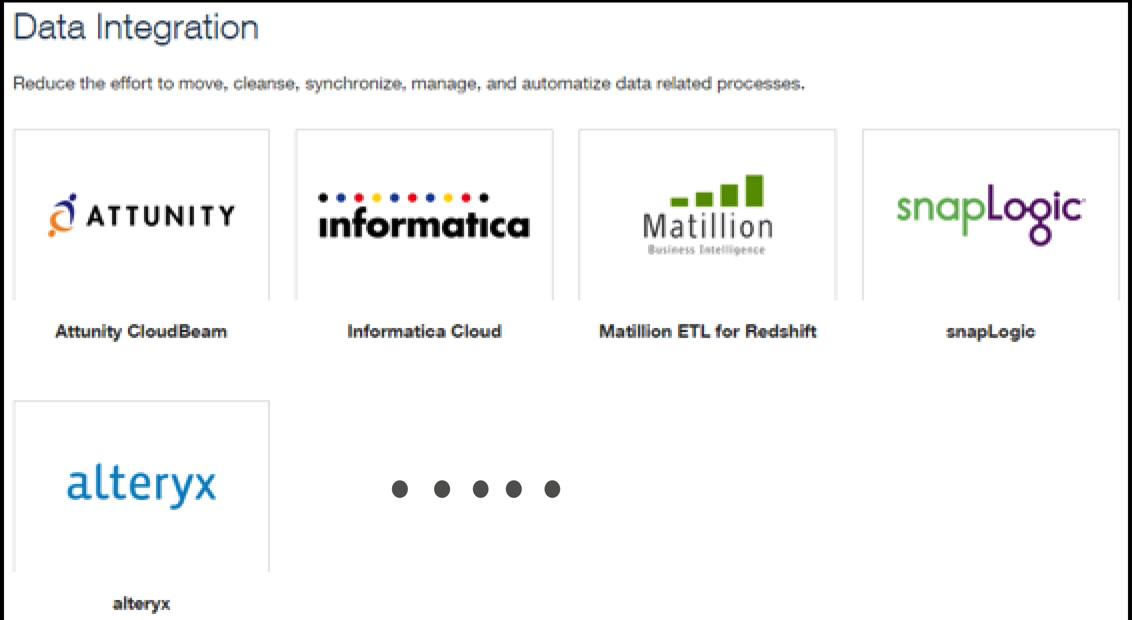
Which Analysis Tool Should I Use?

	Amazon Redshift	Amazon EMR		
		Presto	Spark	Hive
Use case	Optimized for data warehousing	Interactive query	General purpose (iterative ML, RT, ..)	Batch
Scale/throughput	~Nodes	~ Nodes		
Storage	Local storage	Amazon S3, HDFS		
Optimization	Columnar storage, data compression, and zone maps	Framework dependent		
Metadata	Amazon Redshift managed	Hive Meta-store		
BI tools supports	Yes (JDBC/ODBC)	Yes (JDBC/ODBC & Custom)		
Access controls	Users, groups, and access controls	Integration with LDAP		
UDF support	Yes (Scalar)	Yes		

Fast Slow

What About ETL?





Data Integration

Reduce the effort to move, cleanse, synchronize, manage, and automatize data related processes.

Attunity CloudBeam

Informatica Cloud

Matillion ETL for Redshift

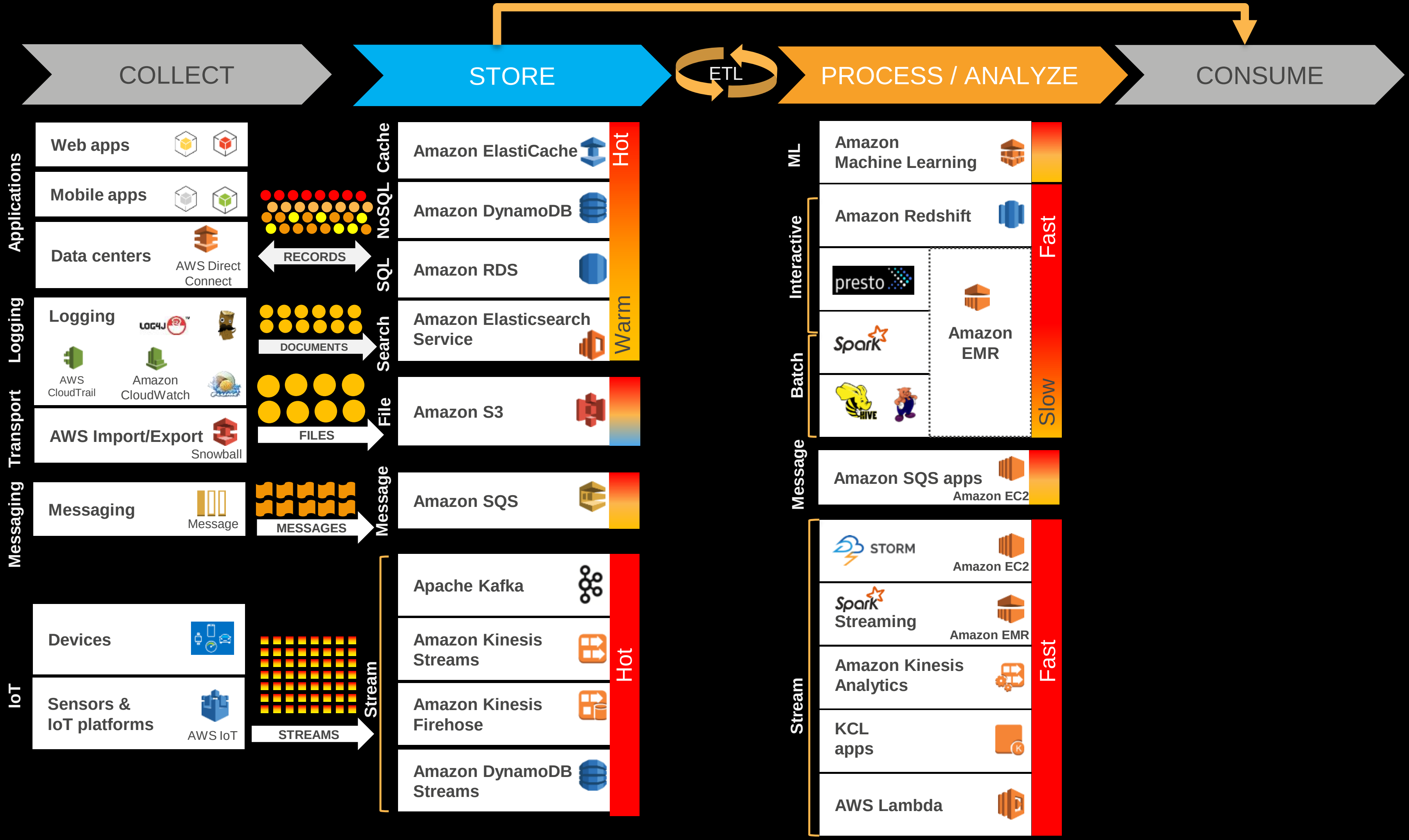
snapLogic

alteryx

• • • • •

https://aws.amazon.com/big-data/partner-solutions/

CONSUME



COLLECT

STORE

ETL

PROCESS / ANALYZE

CONSUME

Applications
- Web apps
- Mobile apps
- Data centers — AWS Direct Connect

Logging
- Logging — LOG4J
- AWS CloudTrail
- Amazon CloudWatch

Transport
- AWS Import/Export — Snowball

Messaging
- Messaging — Message

IoT
- Devices
- Sensors & IoT platforms — AWS IoT

RECORDS

DOCUMENTS

FILES

MESSAGES

STREAMS

Cache
- Amazon ElastiCache

NoSQL
- Amazon DynamoDB

SQL
- Amazon RDS

Search
- Amazon Elasticsearch Service

Hot

Warm

File
- Amazon S3

Message
- Amazon SQS

Stream
- Apache Kafka
- Amazon Kinesis Streams
- Amazon Kinesis Firehose
- Amazon DynamoDB Streams

Hot

ML
- Amazon Machine Learning

Interactive
- Amazon Redshift
- presto

Batch
- Spark
- HIVE

Amazon EMR

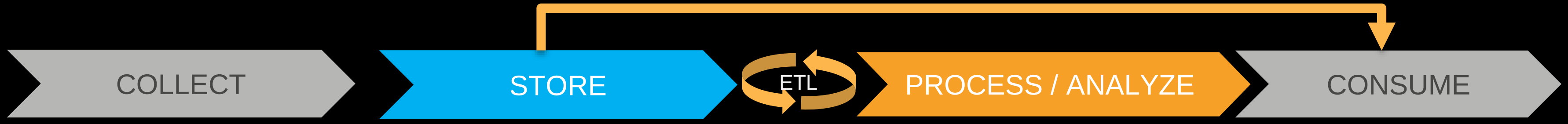
Fast

Slow

Message
- Amazon SQS apps — Amazon EC2

Stream
- STORM — Amazon EC2
- Spark Streaming — Amazon EMR
- Amazon Kinesis Analytics
- KCL apps
- AWS Lambda

Fast

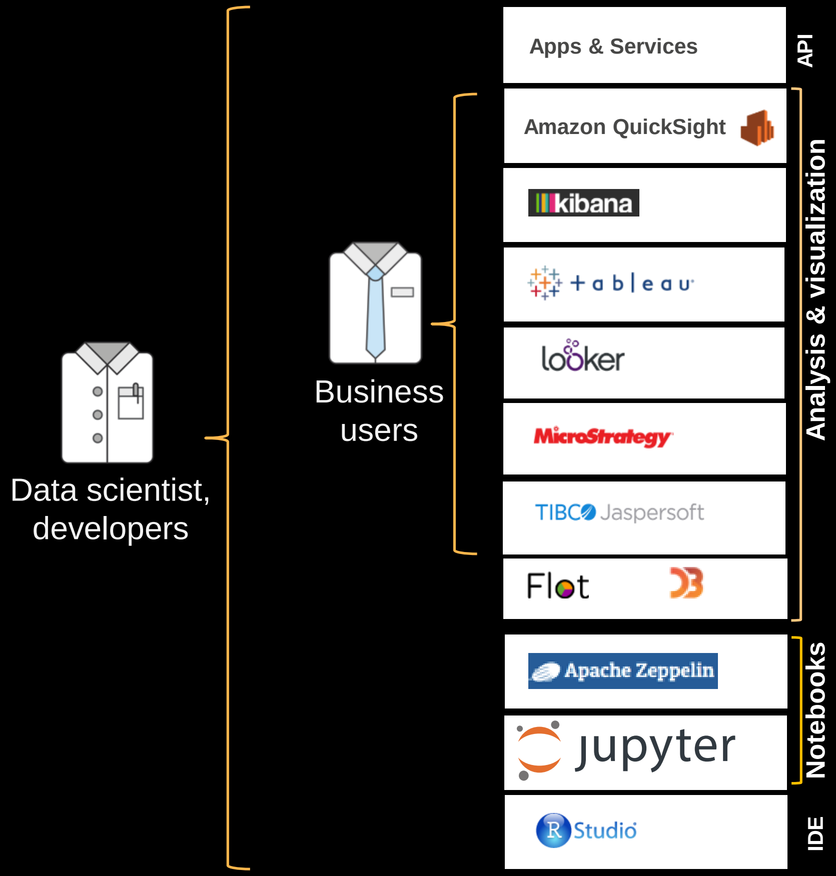


Applications & API

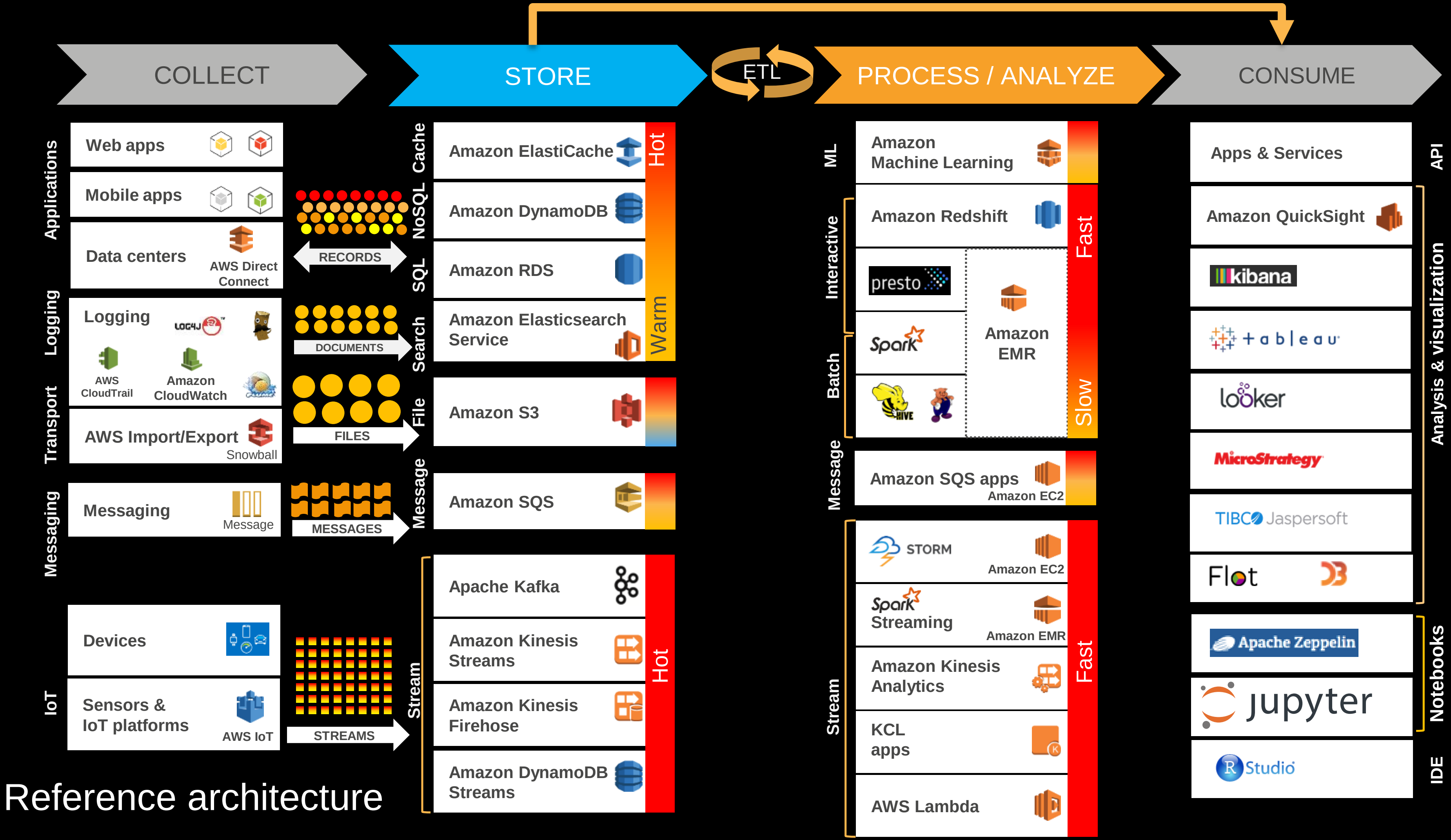
Analysis and visualization

Notebooks

IDE



Putting It All Together



COLLECT

STORE

ETL

PROCESS / ANALYZE

CONSUME

COLLECT

Applications
- Web apps
- Mobile apps
- Data centers — AWS Direct Connect

Logging
- Logging — AWS CloudTrail, Amazon CloudWatch

Transport
- AWS Import/Export Snowball

Messaging
- Messaging — Message

IoT
- Devices
- Sensors & IoT platforms — AWS IoT

RECORDS

DOCUMENTS

FILES

MESSAGES

STREAMS

STORE

- Cache: Amazon ElastiCache — Hot
- NoSQL: Amazon DynamoDB
- SQL: Amazon RDS
- Search: Amazon Elasticsearch Service — Warm
- File: Amazon S3
- Message: Amazon SQS
- Stream: Apache Kafka; Amazon Kinesis Streams; Amazon Kinesis Firehose; Amazon DynamoDB Streams — Hot

PROCESS / ANALYZE

- ML: Amazon Machine Learning
- Interactive: Amazon Redshift; presto — Fast
- Batch: Spark; HIVE; Amazon EMR — Slow
- Message: Amazon SQS apps — Amazon EC2
- Stream: STORM — Amazon EC2; Spark Streaming — Amazon EMR; Amazon Kinesis Analytics; KCL apps; AWS Lambda — Fast

CONSUME

- API: Apps & Services
- Analysis & visualization: Amazon QuickSight; kibana; tableau; looker; MicroStrategy; TIBCO Jaspersoft; Flot; D3
- Notebooks: Apache Zeppelin; jupyter
- IDE: RStudio

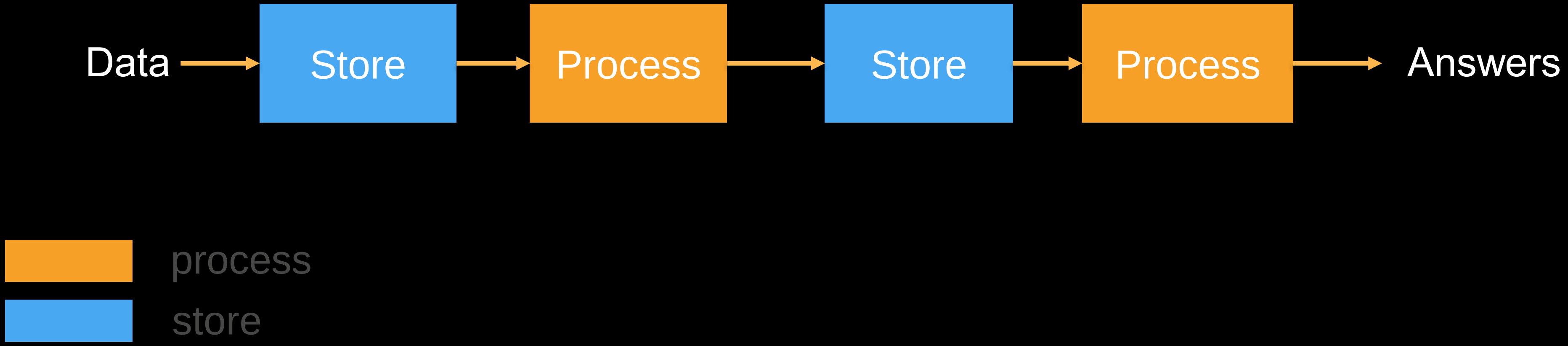
Reference architecture

Design Patterns

Primitive: Decoupled Data Bus

Storage decoupled from processing

Multiple stages



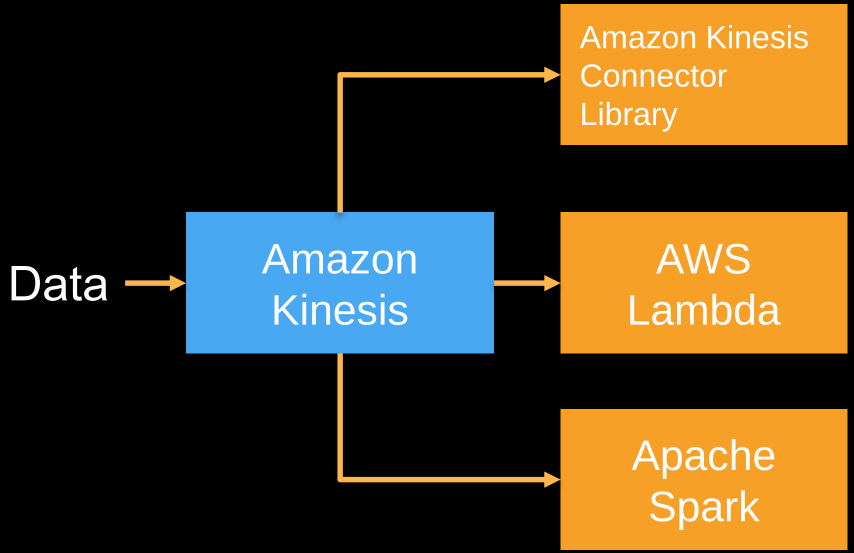
Data → Store → Process → Store → Process → Answers

🟧 process

🟦 store

Primitive: Pub/Sub

Parallel stream consumption/processing

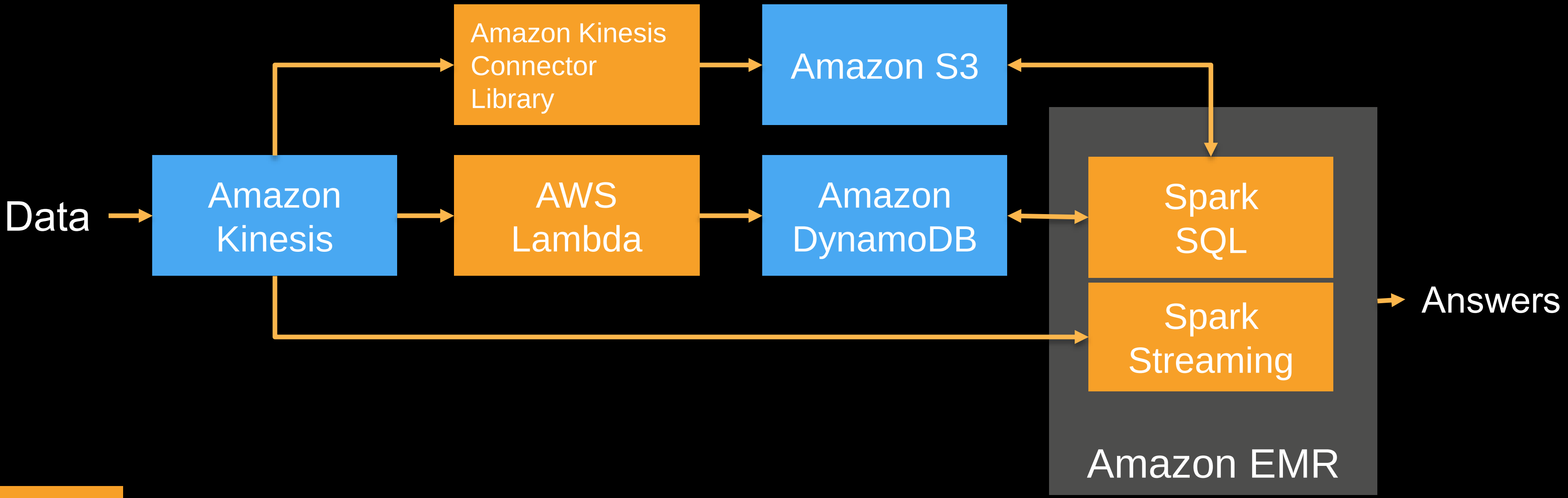


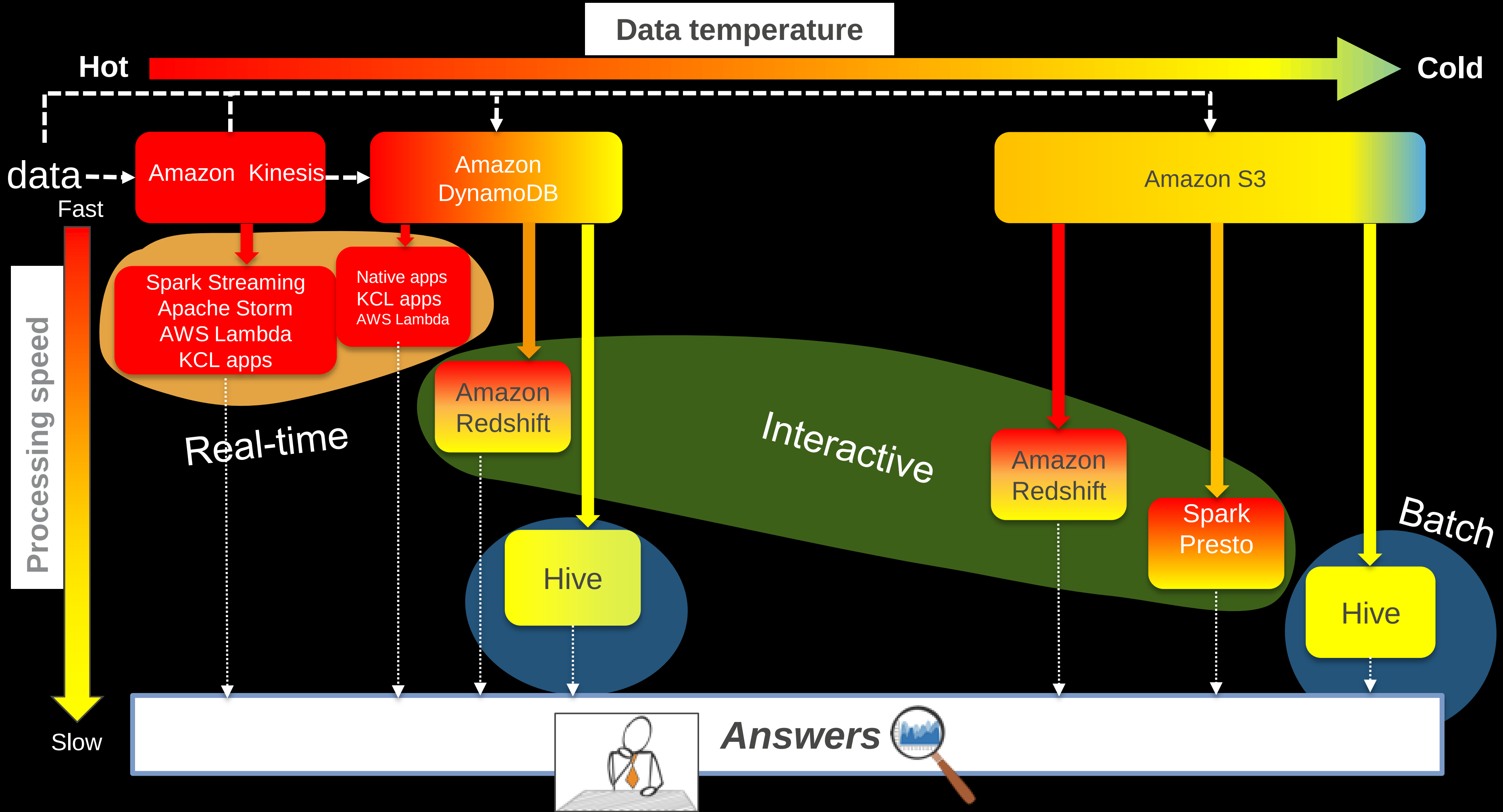
🟧 process

🟦 store

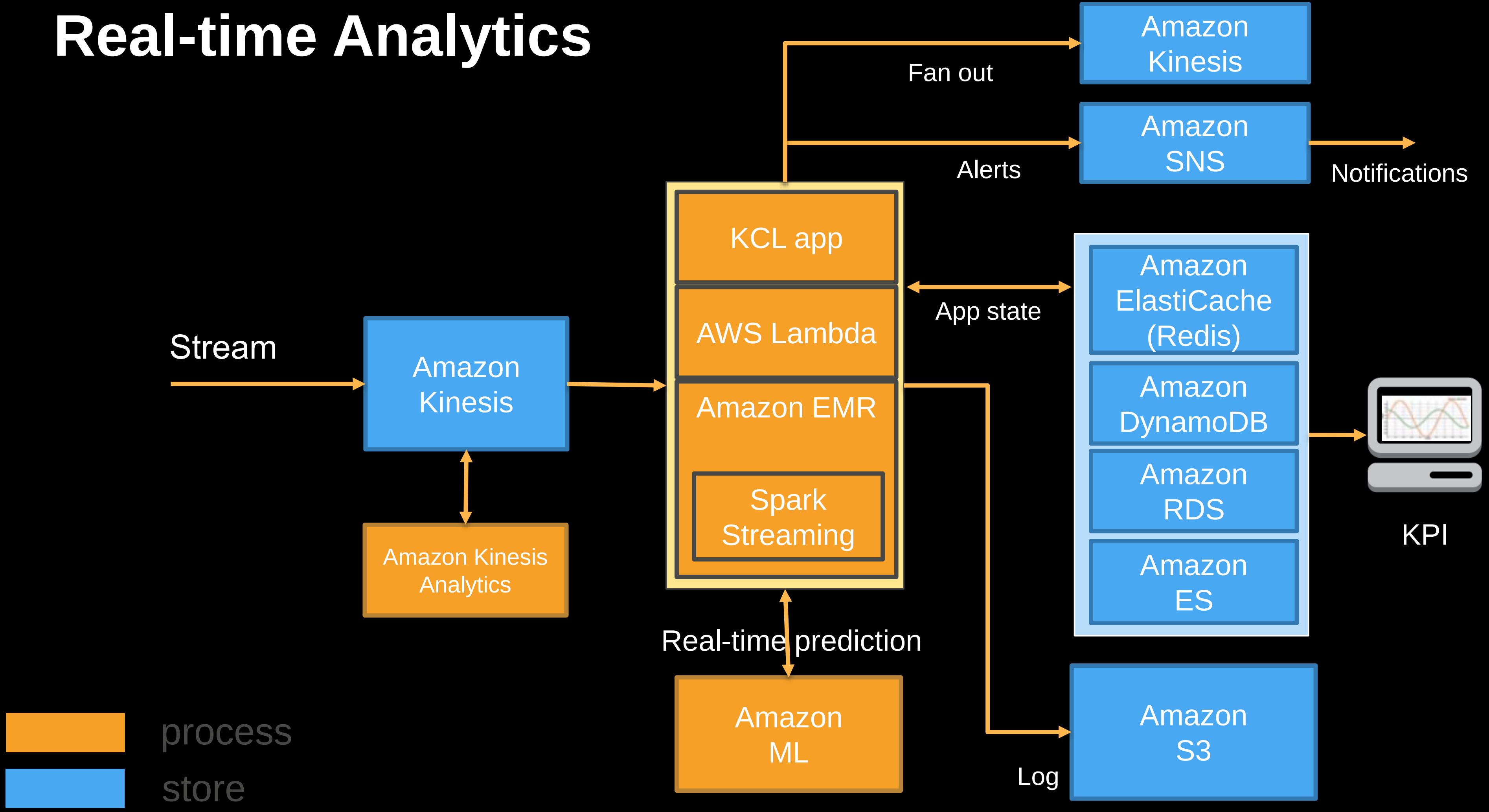
Primitive: Materialized Views

Analysis framework reads from or writes to multiple data stores




Data temperature
Hot
Cold
data
Fast
Slow
Processing speed
Amazon Kinesis
Amazon DynamoDB
Amazon S3
Spark Streaming
Apache Storm
AWS Lambda
KCL apps
Native apps
KCL apps
AWS Lambda
Real-time
Amazon Redshift
Interactive
Amazon Redshift
Spark
Presto
Hive
Batch
Hive
Answers

Real-time Analytics



Stream

Amazon Kinesis

Amazon Kinesis Analytics

KCL app

AWS Lambda

Amazon EMR

Spark Streaming

Real-time prediction

Amazon ML

Fan out

Amazon Kinesis

Alerts

Amazon SNS

Notifications

App state

Amazon ElastiCache (Redis)

Amazon DynamoDB

Amazon RDS

Amazon ES

KPI

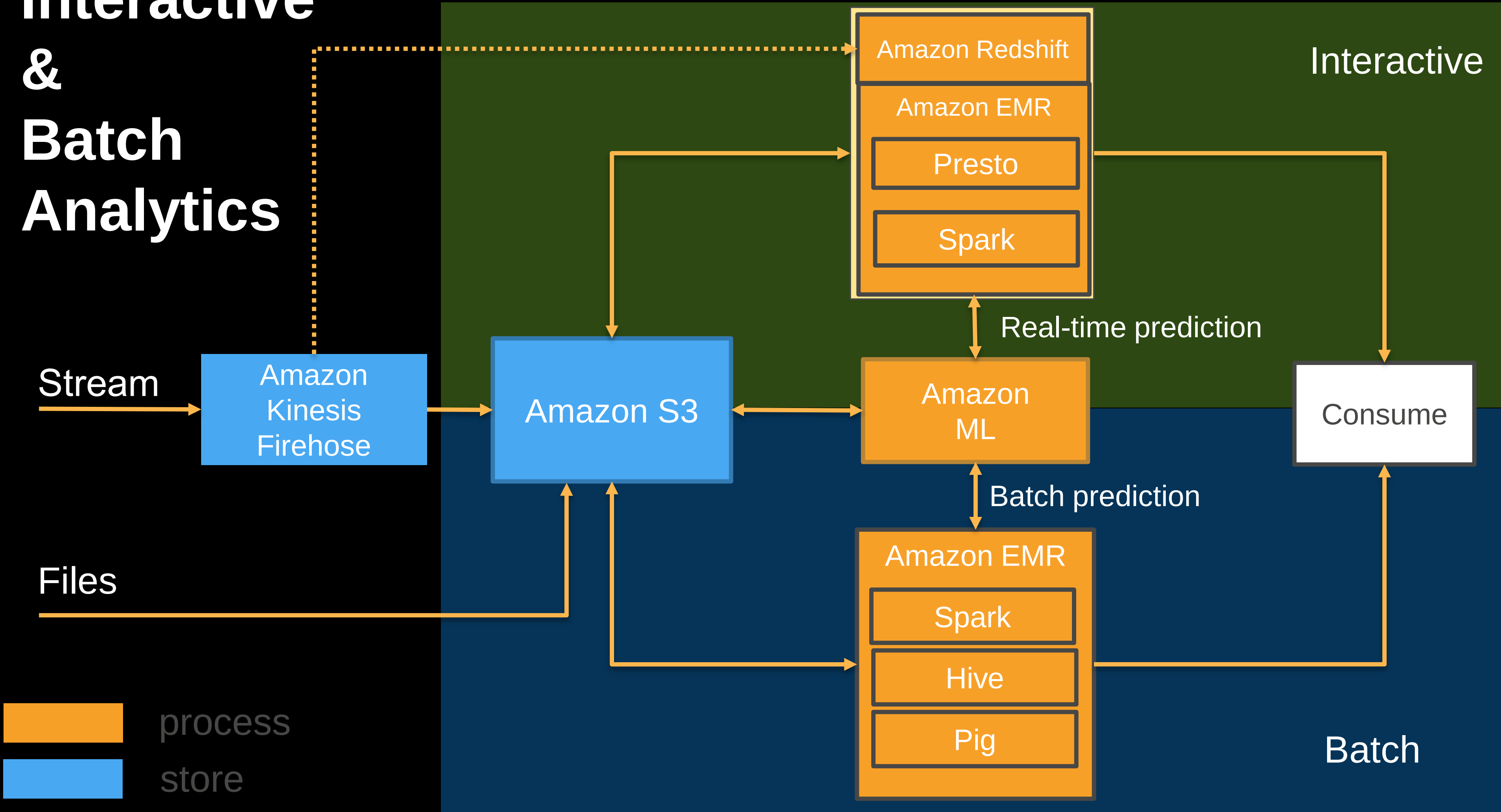
Log

Amazon S3

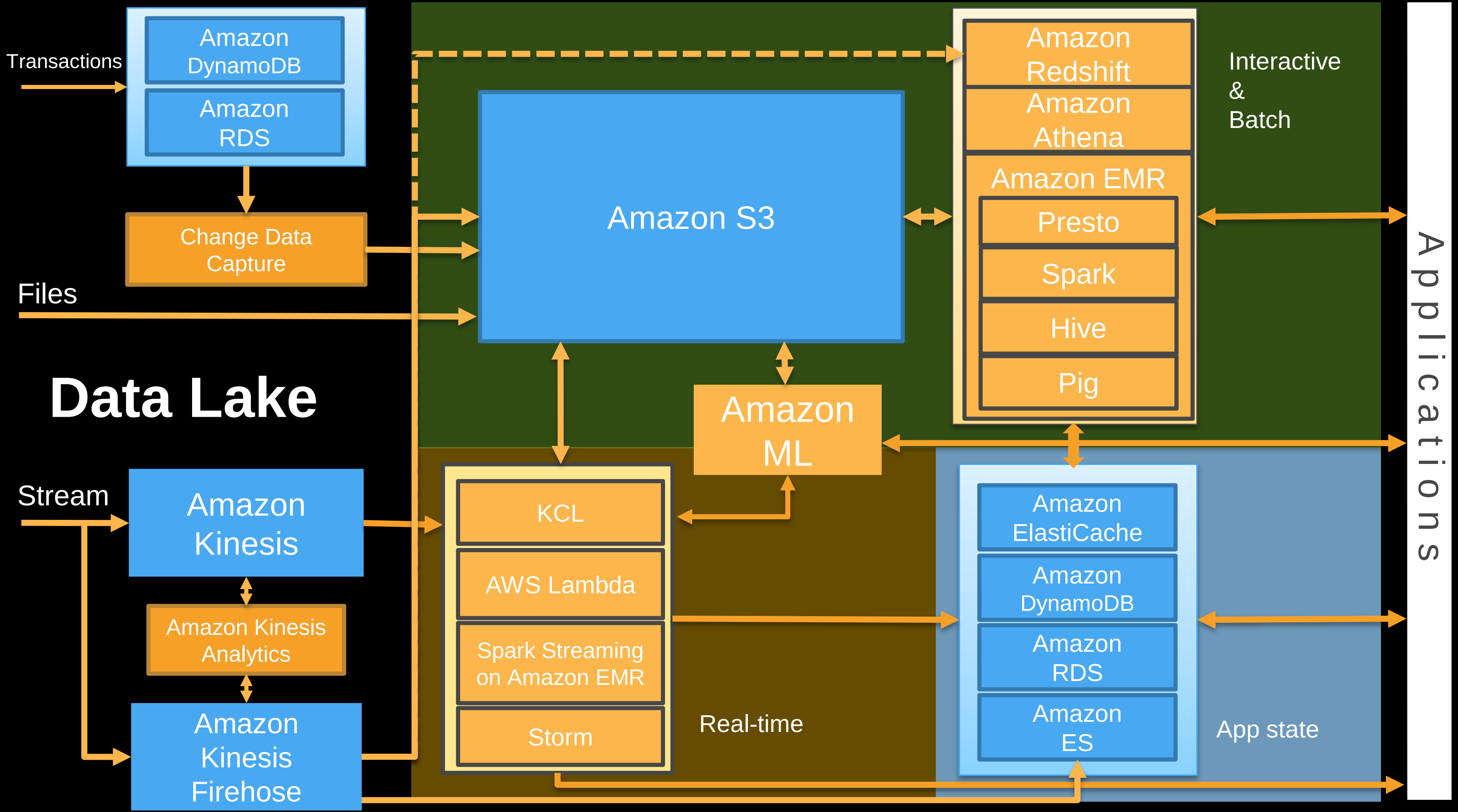
🟧 process

🟦 store

Interactive & Batch Analytics



Data Lake



Transactions

Amazon DynamoDB

Amazon RDS

Change Data Capture

Files

Stream

Amazon Kinesis

Amazon Kinesis Analytics

Amazon Kinesis Firehose

Amazon S3

Amazon Redshift

Amazon Athena

Amazon EMR

Presto

Spark

Hive

Pig

Interactive & Batch

Amazon ML

KCL

AWS Lambda

Spark Streaming on Amazon EMR

Storm

Real-time

Amazon ElastiCache

Amazon DynamoDB

Amazon RDS

Amazon ES

App state

Applications

Summary

Build decoupled systems

- Data → Store → Process → Store → Analyze → Answers

Use the right tool for the job

- Data structure, latency, throughput, access patterns

Leverage AWS managed services

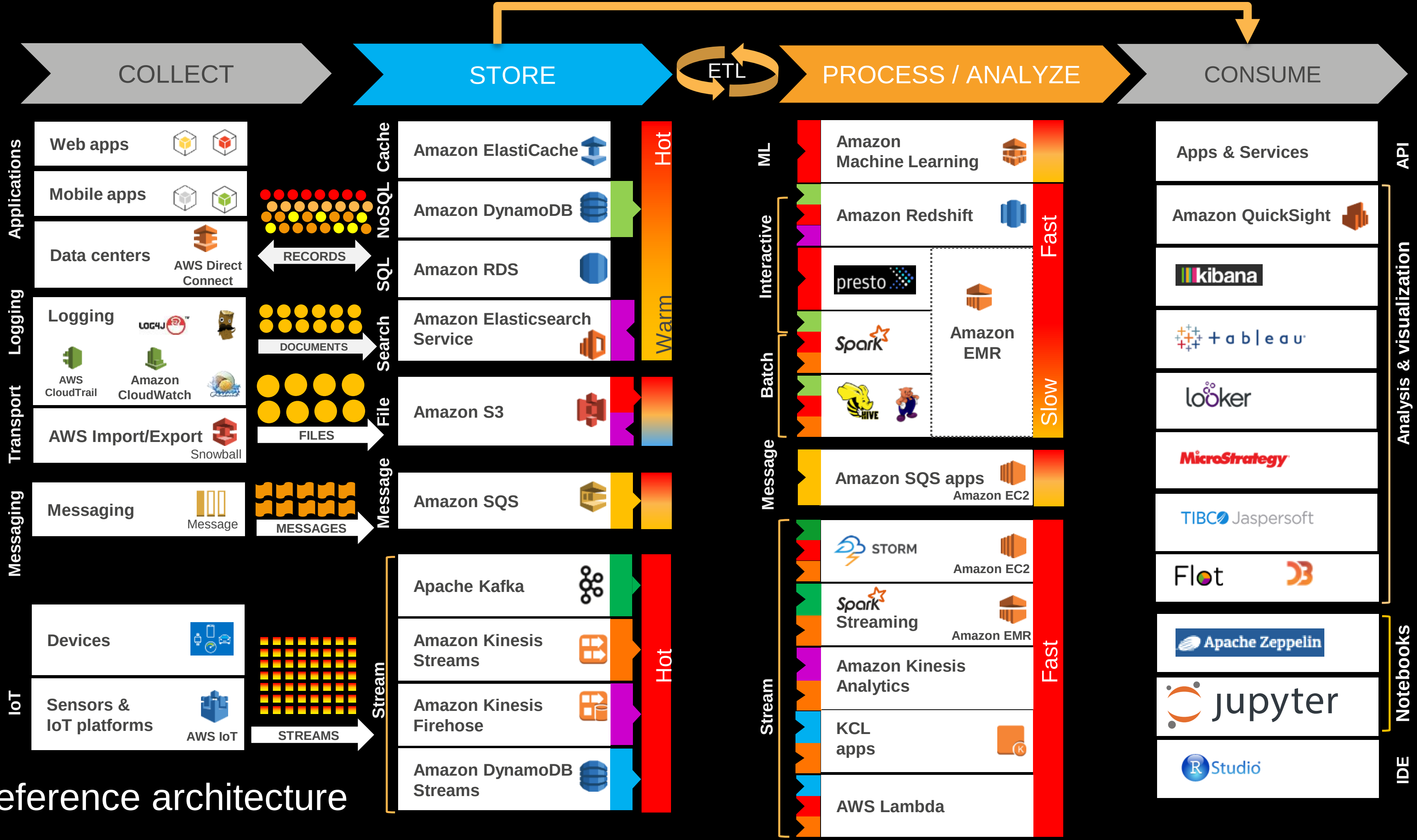
- Scalable/elastic, available, reliable, secure, no/low admin

Use log-centric design patterns

- Immutable log, batch, interactive & real-time views

Be cost conscious

- Big data ≠ big cost



COLLECT

Applications
- Web apps
- Mobile apps
- Data centers: AWS Direct Connect

Logging
- Logging: LOG4J, AWS CloudTrail, Amazon CloudWatch

Transport
- AWS Import/Export Snowball

Messaging
- Messaging: Message

IoT
- Devices
- Sensors & IoT platforms: AWS IoT

RECORDS, DOCUMENTS, FILES, MESSAGES, STREAMS

STORE

Cache: Amazon ElastiCache

NoSQL: Amazon DynamoDB

SQL: Amazon RDS

Search: Amazon Elasticsearch Service

Hot / Warm

File: Amazon S3

Message: Amazon SQS

Stream
- Apache Kafka
- Amazon Kinesis Streams
- Amazon Kinesis Firehose
- Amazon DynamoDB Streams

Hot

ETL

PROCESS / ANALYZE

ML: Amazon Machine Learning

Interactive
- Amazon Redshift
- presto

Batch
- Spark
- HIVE

Amazon EMR

Fast / Slow

Message: Amazon SQS apps, Amazon EC2

Stream
- STORM, Amazon EC2
- Spark Streaming, Amazon EMR
- Amazon Kinesis Analytics
- KCL apps
- AWS Lambda

Fast

CONSUME

API: Apps & Services

Analysis & visualization
- Amazon QuickSight
- kibana
- tableau
- looker
- MicroStrategy
- TIBCO Jaspersoft
- Flot, D3

Notebooks
- Apache Zeppelin
- jupyter

IDE: RStudio

Reference architecture

AWS re:Invent

Thank you!

amazon web services

Remember to complete your evaluations!