Microsoft

Azure
SQL Data Warehouse

WIFI	MSFTGUEST
EMAIL	KALYANY@MICROSOFT.COM

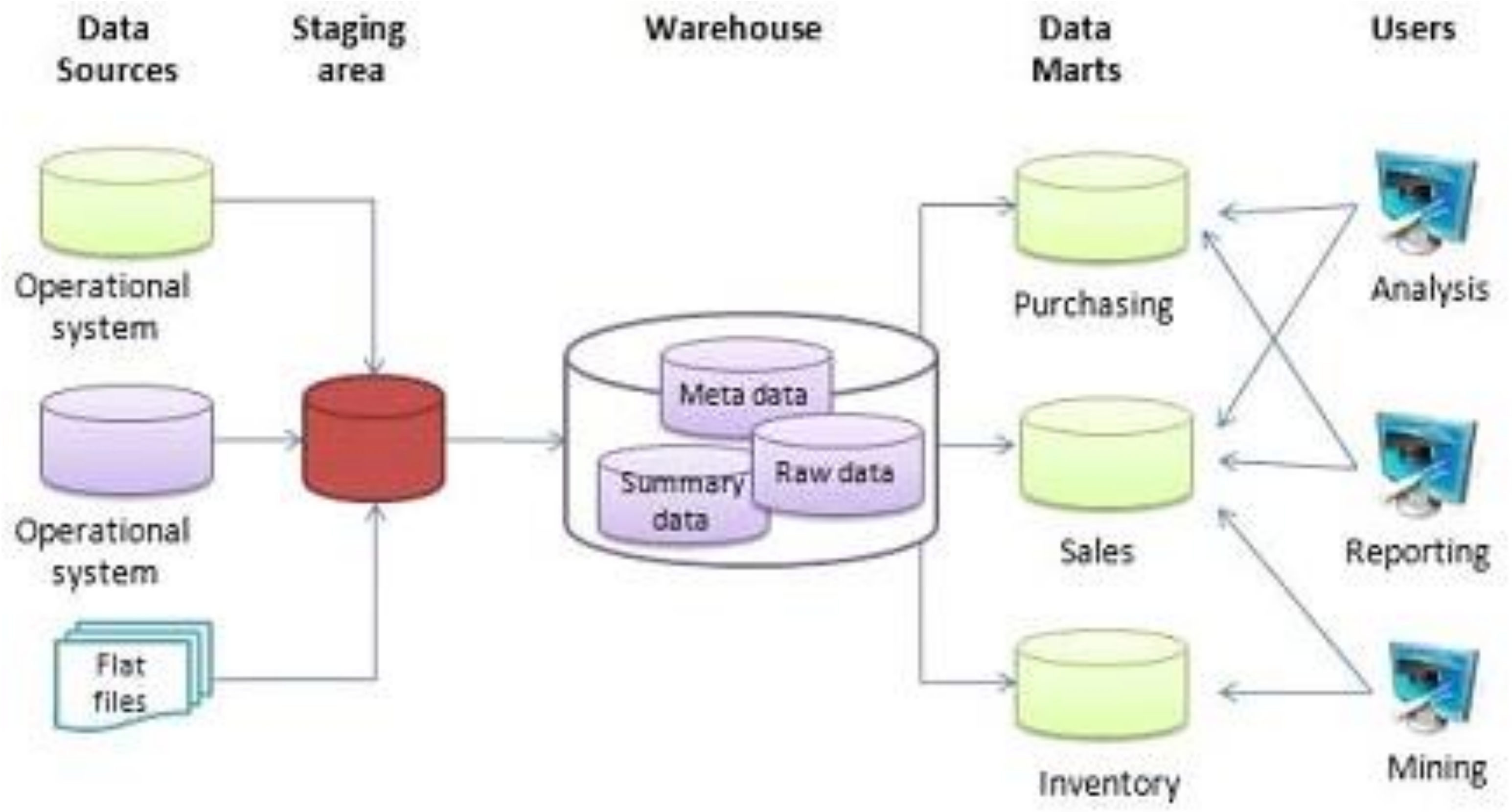
Day 1 Agenda

Title	Speaker	Duration	Start Time	End Time
Azure SQL DW Overview	Casey Karst	90	9:00:00 AM	10:30:00 AM
Break		15	10:30:00 AM	10:45:00 AM
ADF Overview	Charlie Zhu	60	10:45:00 AM	11:45:00 AM
Lunch + Compete	Kal Yella	45	11:45:00 AM	12:30:00 PM
Azure SQL DW New Features and Enhancements	Kal Yella	90	12:30:00 PM	2:00:00 PM
Break		15	2:00:00 PM	2:15:00 PM
SQL DW Loading Scenarios	Casey Karst	30	2:15:00 PM	2:45:00 PM
Data Flow in ADF for code free ETL/ELT	Daniel Perlovsky	75	2:45:00 PM	4:00:00 PM
Loading Lab	All	60	4:00:00 PM	5:00:00 PM

Sign-Up Link (For All): http://bit.ly/2mG5PaZ

Activation Code	Expiry Date
ACTIVATE4253	Saturday, September 28, 2019

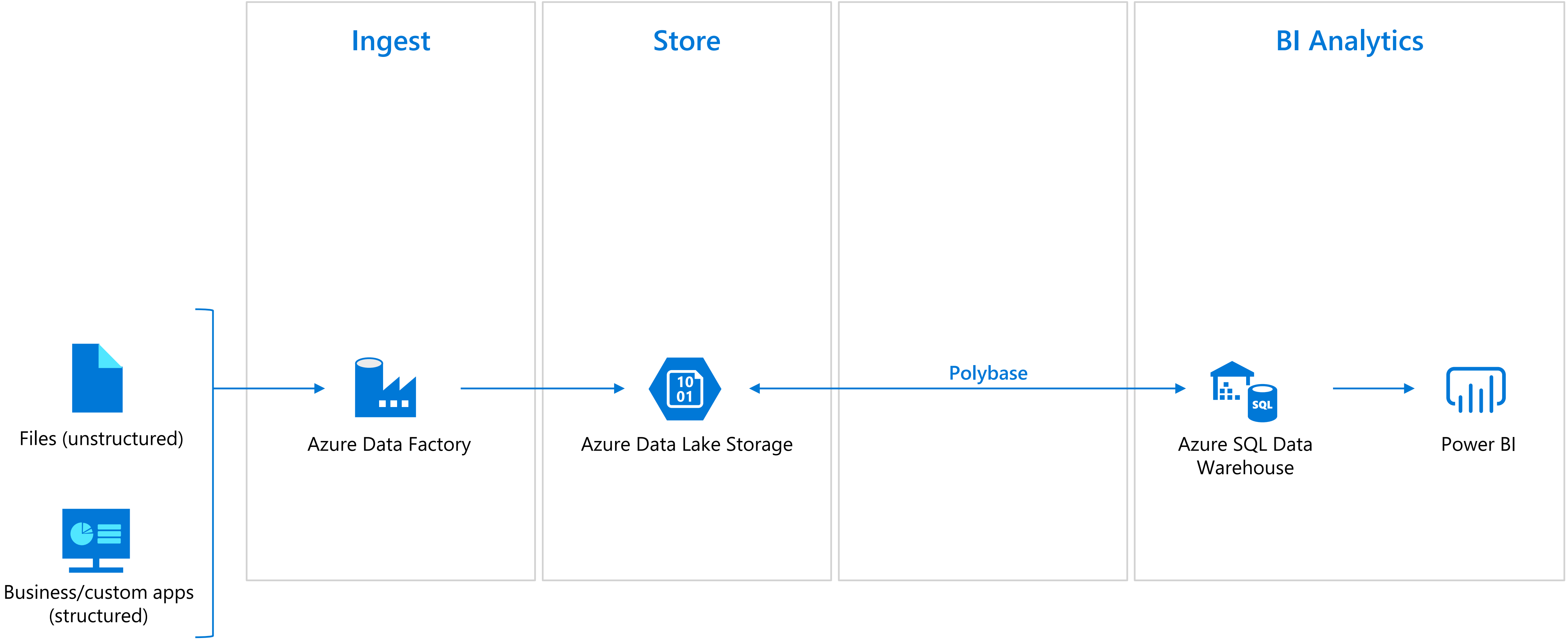
Where we came from



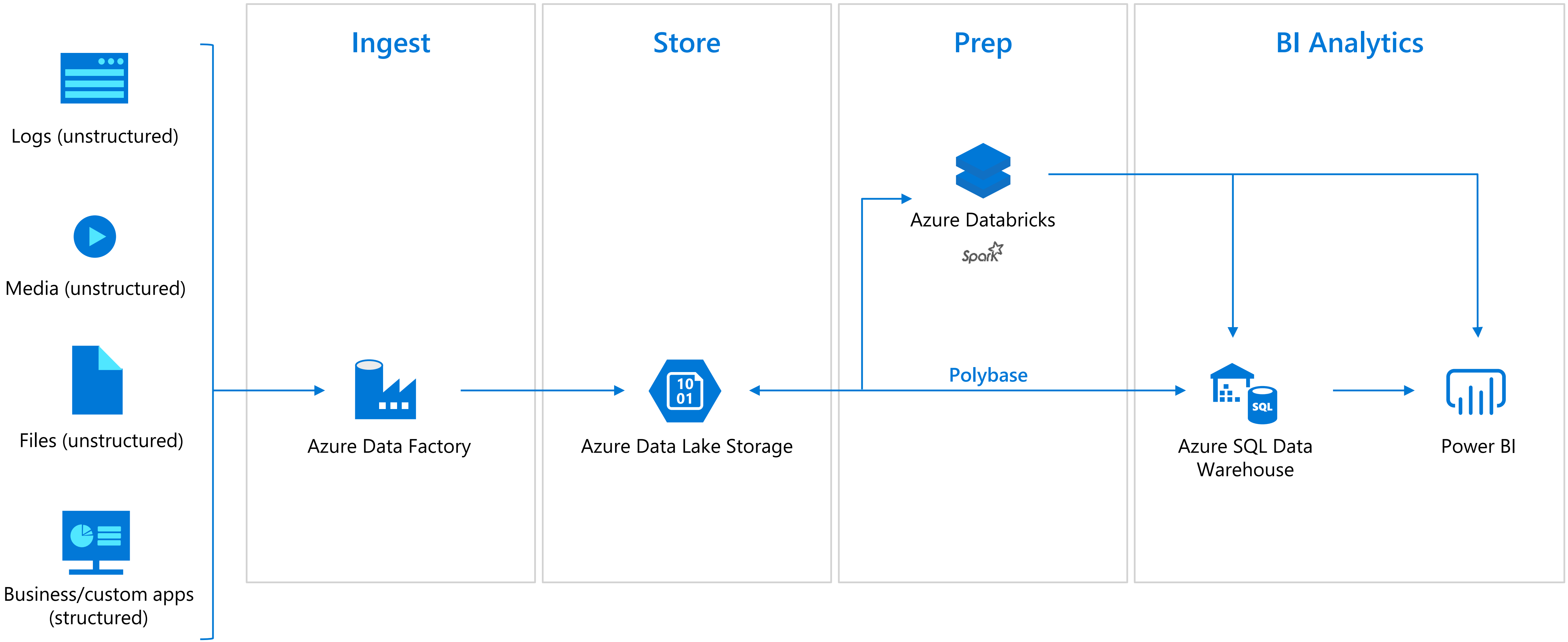
Challenges of the past

- Scalability
- Performance
- Flexibility

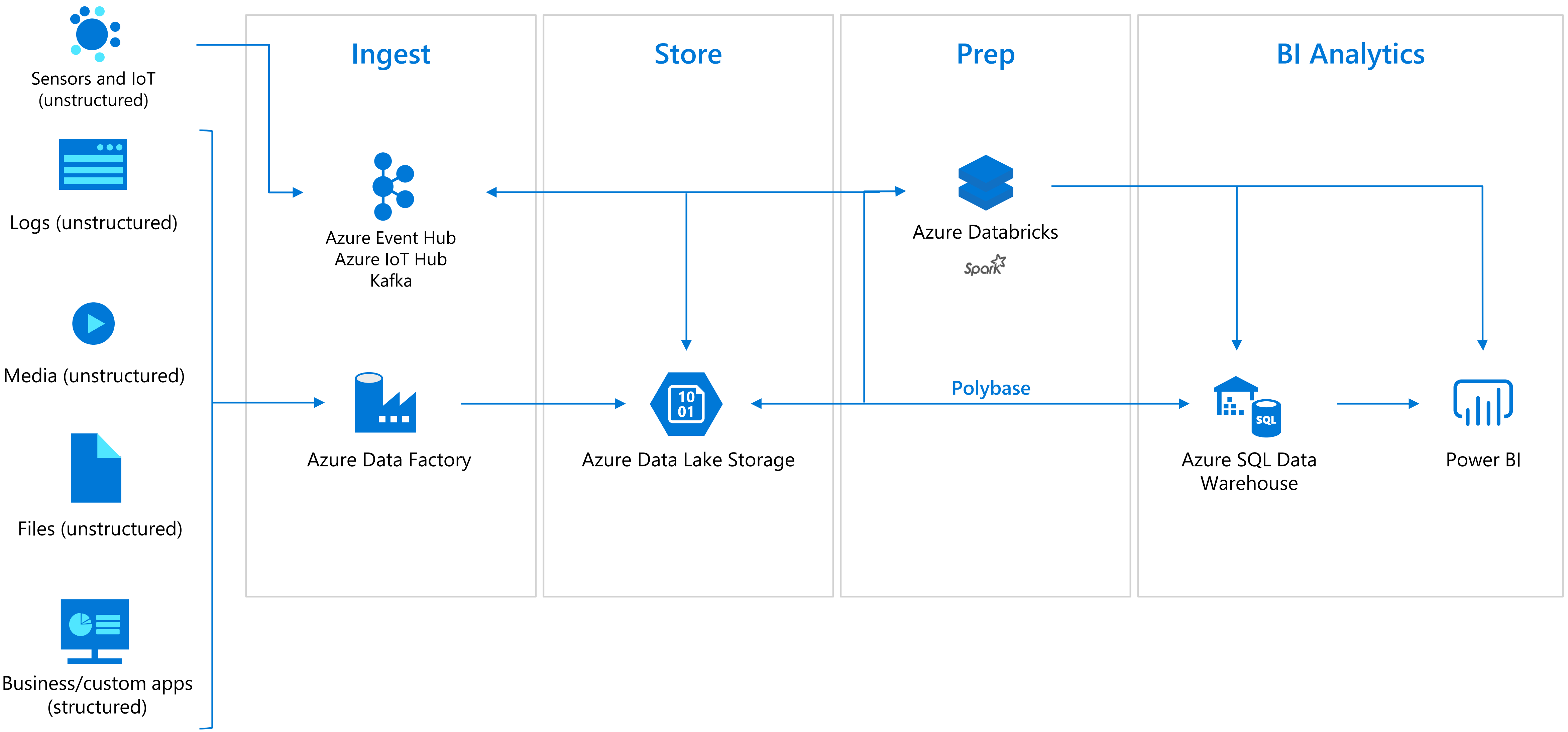
MDW – Traditional ETL



MDW – Traditional ETL + Prep



MDW – Batch and Stream



Azure SQL Data Warehouse

Best in class price-performance

Up to 94% less expensive than competitors

Industry-leading security

Defense-in-depth security and 99.9% financially backed availability SLA

Intelligent workload management

Separation of compute and storage

Prioritize resources for the most valuable workloads

Data flexibility

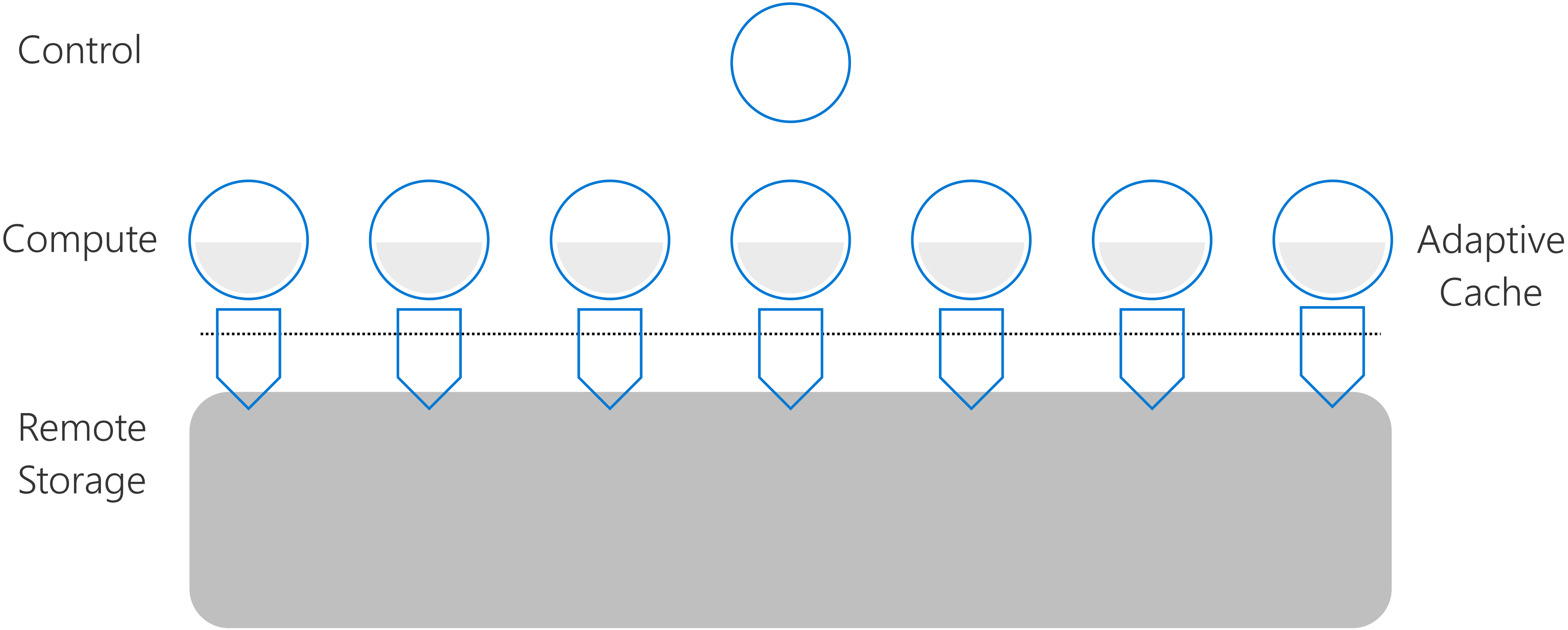
Query directly over the Data Lake

Support for structured and semi-structured data

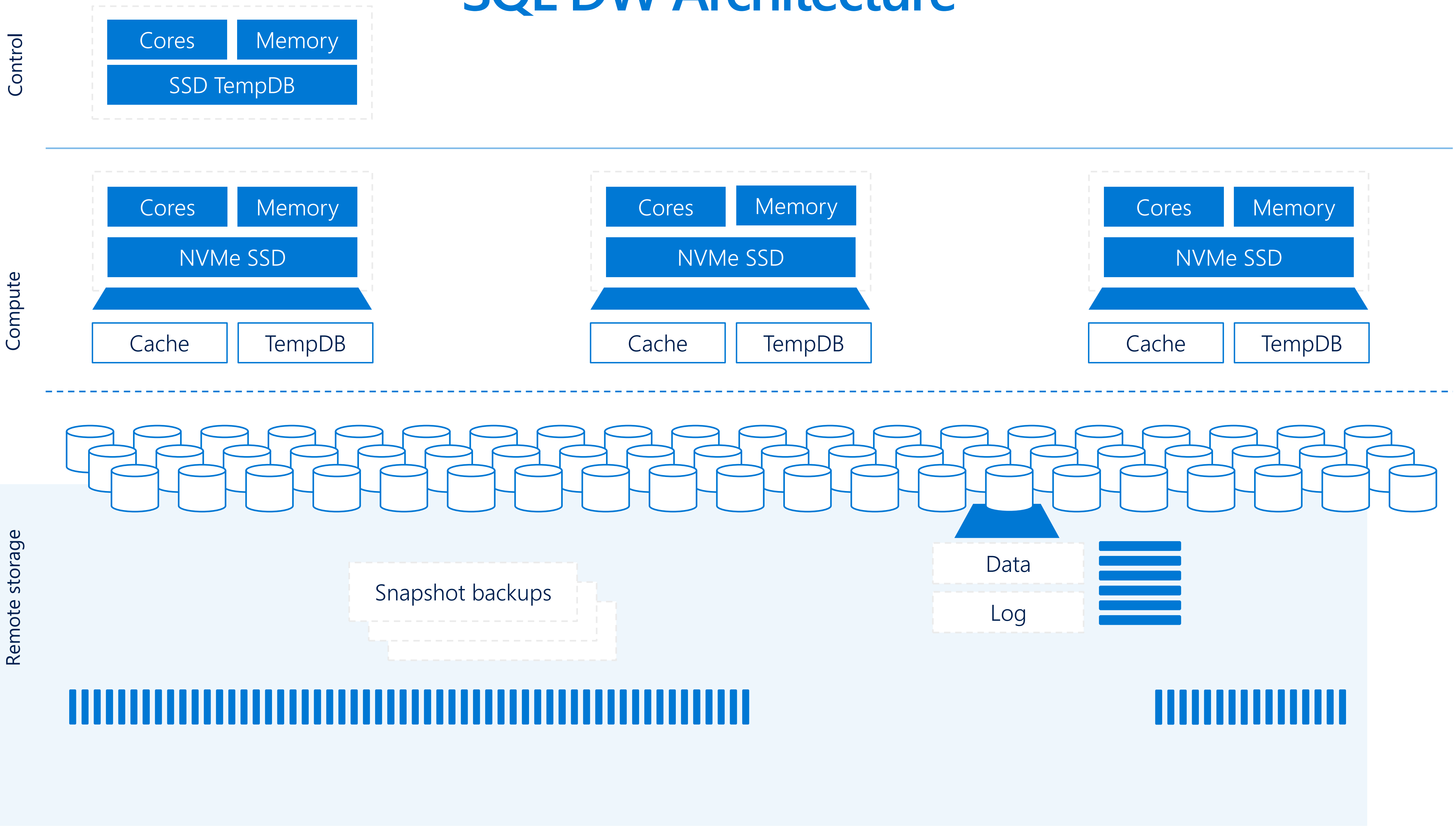
Developer productivity

Enterprise class application lifecycle management

SQL DW Architecture

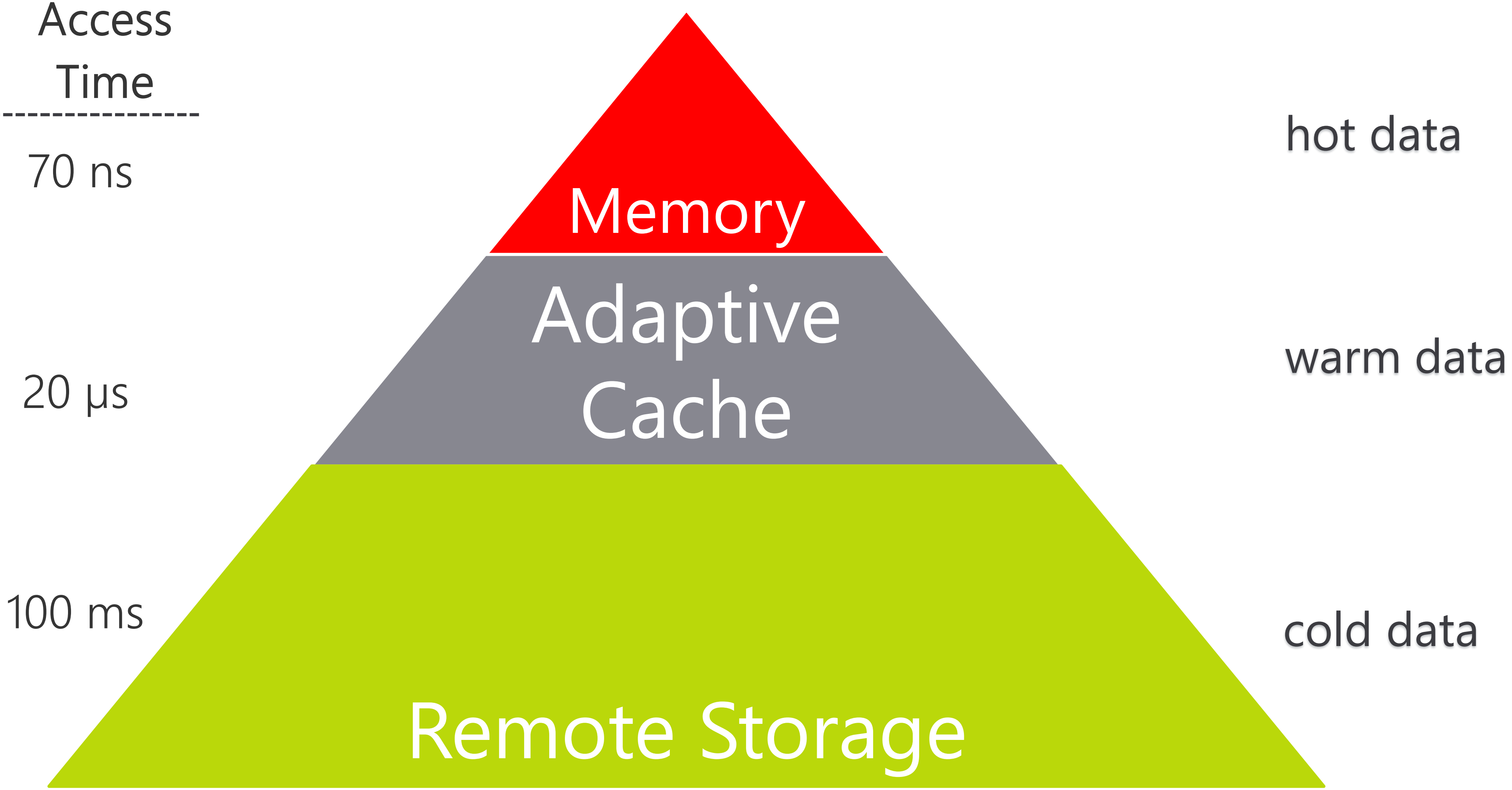


SQL DW Architecture



Automated Tiering Of Storage Layers

Node Capacity	Access Time
400GB+	70 ns
1.5TB	20 µs
∞	100 ms



Memory — hot data

Adaptive Cache — warm data

Remote Storage — cold data

Terms to Remember

Control Node

Connection endpoint of a SQL DW

Generates a Distributed Query Plan

Holds High level Metadata

Compute Node

Executes portion of query

Communicate with other Compute nodes and Control node via Data Movement Service (DMS)

Distribution

Physical bucket of data

Always 60 distributions (regardless of scale)

Attach to Compute Nodes

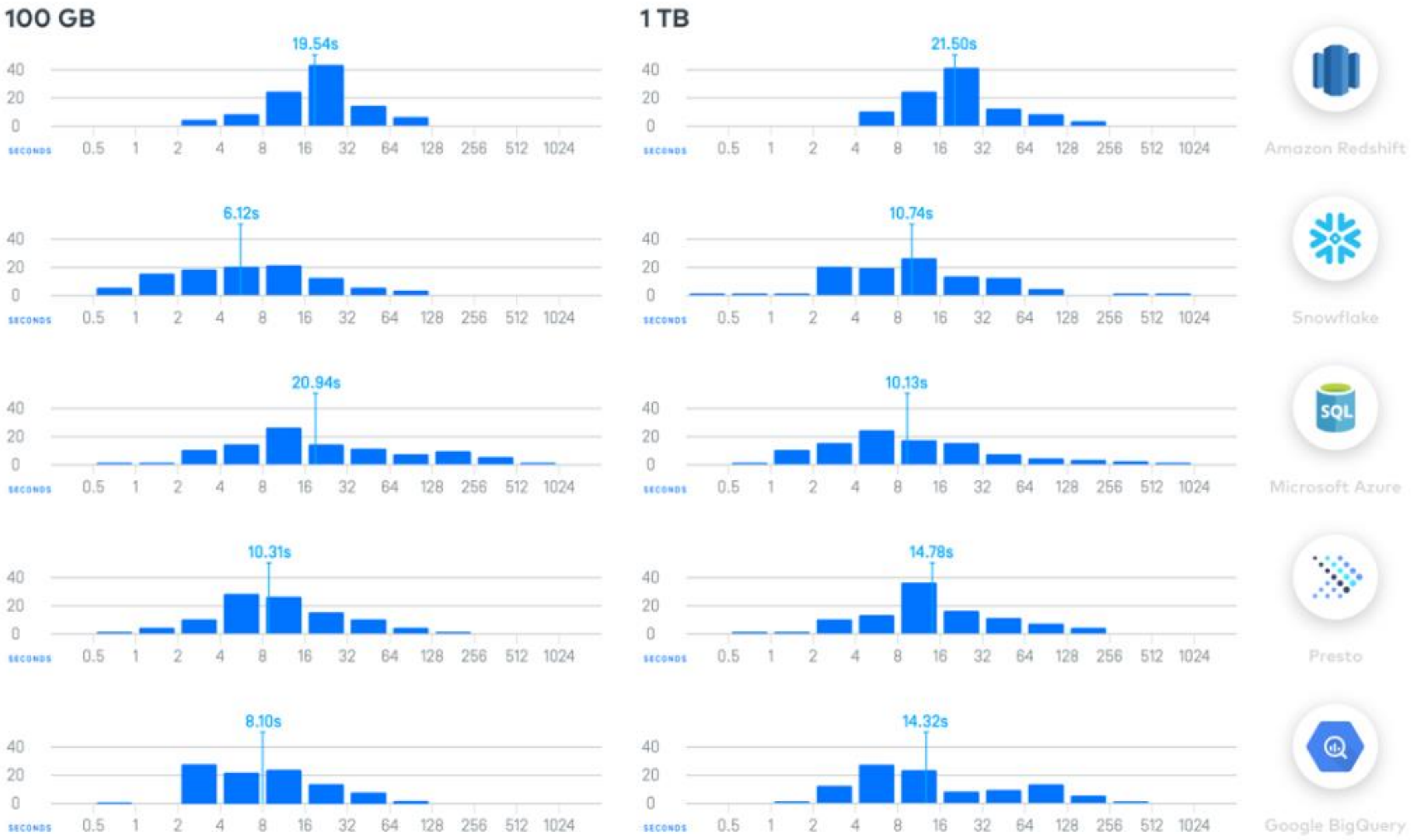
Adaptive Cache

CCI segment cache for faster access of queried data

Scales up as DW scales up

Data Storage and performance optimizations

Fivetran benchmark (TPC-DS) untuned



https://fivetran.com/blog/warehouse-benchmark

Most-comprehensive data optimization & caching features

Category	Feature	SQL Data Warehouse	Amazon Redshift	Snowflake	Google Big Query
Data storage	Columnstore table storage *(Columnstore tables)*	Yes	Yes	Yes	Yes
	Rowstore table storage *(Heap tables)*	Yes	No	No	No
	In-memory table storage *(Replicated tables)*	Yes	No	No	Yes
Clustered Indexes	Ordered columnar indexes	Yes	Yes	Yes	Beta
	Clustered Index	Yes	No	No	No
Non-Clustered Indexes	Non-Clustered Index *(Secondary indexes)*	Yes	No	No	No
Table Partitions	Columnar Rowgroups (micro-partitions)	Yes	No	Yes	No
	Range-based table partitioning	Yes	No	No	Yes
Result Caching	Result-set caching	Yes	Yes	Yes	Yes
Materialized Views	Materialized views	Yes	No	Yes	No

Tables – Indexes

Clustered Columnstore index (Default Primary)

Highest level of data compression

Best overall query performance

Support for ordered Columnstore segments

Clustered index (Primary)

Performant for looking up a single to few rows

Heap (Primary)

Faster loading and landing temporary data

Best for small lookup tables

Nonclustered indexes (Secondary)

Enable ordering of multiple columns in a table

Allows multiple nonclustered on a single table

Can be created on any of the above primary indexes

More performant lookup queries

```
-- Create table with index
CREATE TABLE orderTable
(
    OrderId  INT NOT NULL,
    Date     DATE NOT NULL,
    Name     VARCHAR(2),
    Country  VARCHAR(2)
)
WITH
(
    CLUSTERED COLUMNSTORE INDEX |
    HEAP |
    CLUSTERED INDEX (OrderId)
);


-- Add non-clustered index to table
CREATE INDEX NameIndex ON orderTable (Name);
```

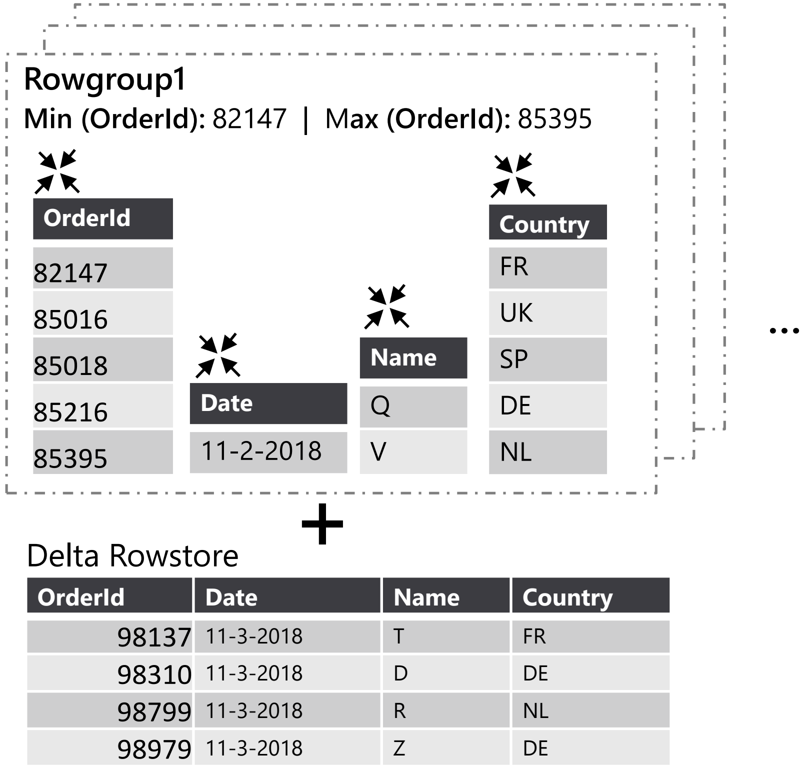
Tables – DW Indexes Illustrated

Logical table structure

OrderId	Date	Name	Country
85016	11-2-2018	V	UK
85018	11-2-2018	Q	SP
85216	11-2-2018	Q	DE
85395	11-2-2018	V	NL
82147	11-2-2018	Q	FR
86881	11-2-2018	D	UK
93080	11-3-2018	R	UK
94156	11-3-2018	S	FR
96250	11-3-2018	Q	NL
98799	11-3-2018	R	NL
98015	11-3-2018	T	UK
98310	11-3-2018	D	DE
98979	11-3-2018	Z	DE
98137	11-3-2018	T	FR
…	…	…	…

Clustered columnstore index

(OrderId)



Rowgroup1

Min (OrderId): 82147 | Max (OrderId): 85395

OrderId
82147
85016
85018
85216
85395

Date
11-2-2018

Name
Q
V

Country
FR
UK
SP
DE
NL

…

+

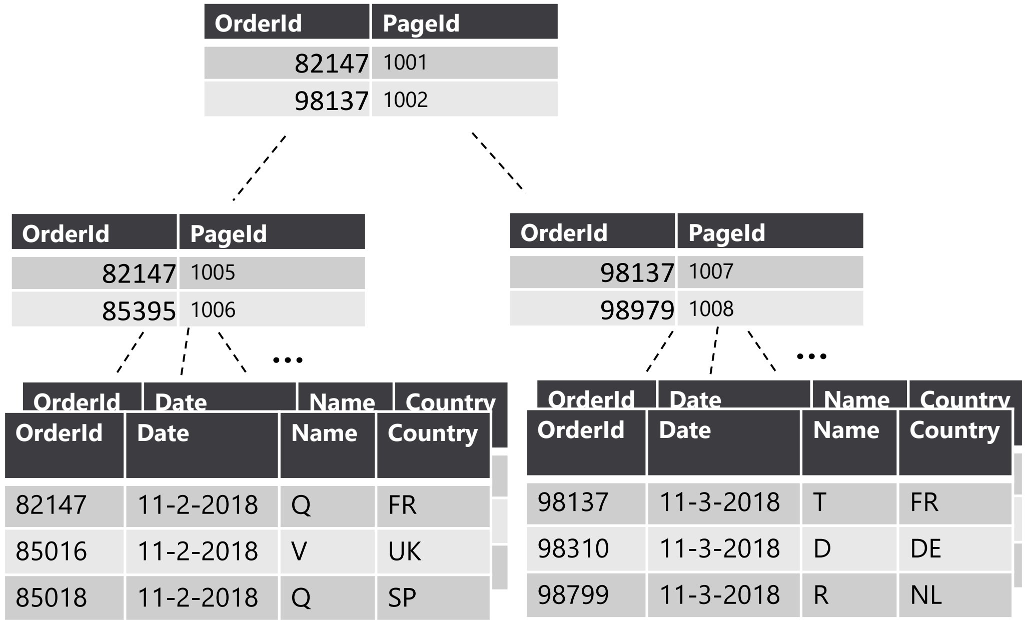
Delta Rowstore

OrderId	Date	Name	Country
98137	11-3-2018	T	FR
98310	11-3-2018	D	DE
98799	11-3-2018	R	NL
98979	11-3-2018	Z	DE

- Data stored in compressed columnstore segments after being sliced into groups of rows (rowgroups/micro-partitions) for maximum compression
- Rows are stored in the delta rowstore until the number of rows is large enough to be compressed into a columnstore

Clustered/Non-clustered rowstore index

(OrderId)



OrderId	PageId
82147	1001
98137	1002

OrderId	PageId
82147	1005
85395	1006

OrderId	PageId
98137	1007
98979	1008

…

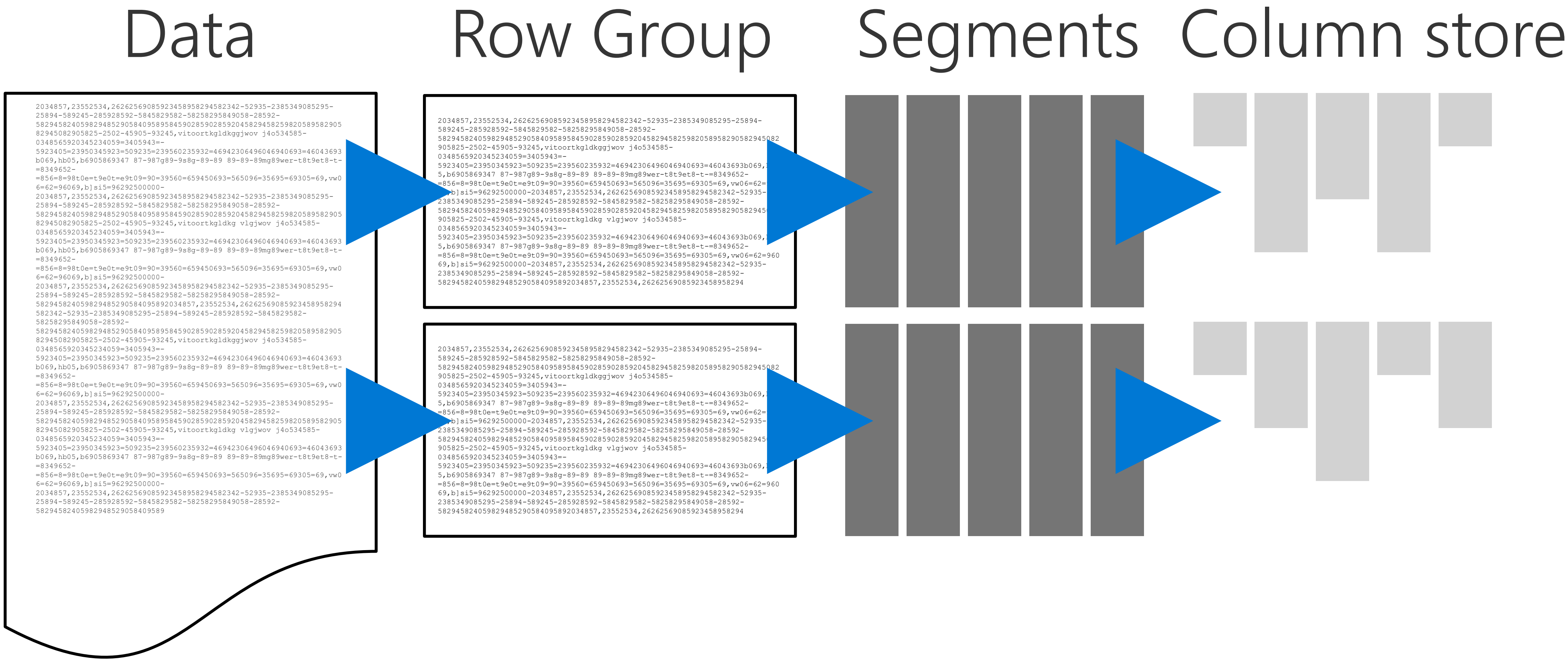
OrderId	Date	Name	Country
82147	11-2-2018	Q	FR
85016	11-2-2018	V	UK
85018	11-2-2018	Q	SP

…

OrderId	Date	Name	Country
98137	11-3-2018	T	FR
98310	11-3-2018	D	DE
98799	11-3-2018	R	NL

- Data is stored in a B-tree index structure for performant lookup queries for particular rows.
- Clustered rowstore index: The leaf nodes in the structure store the data values in a row (as pictured above)
- Non-clustered (secondary) rowstore index: The leaf nodes store pointers to the data values, not the values themselves

Column store taxonomy



Data

Row Group

Segments

Column store

CCI Best Practices

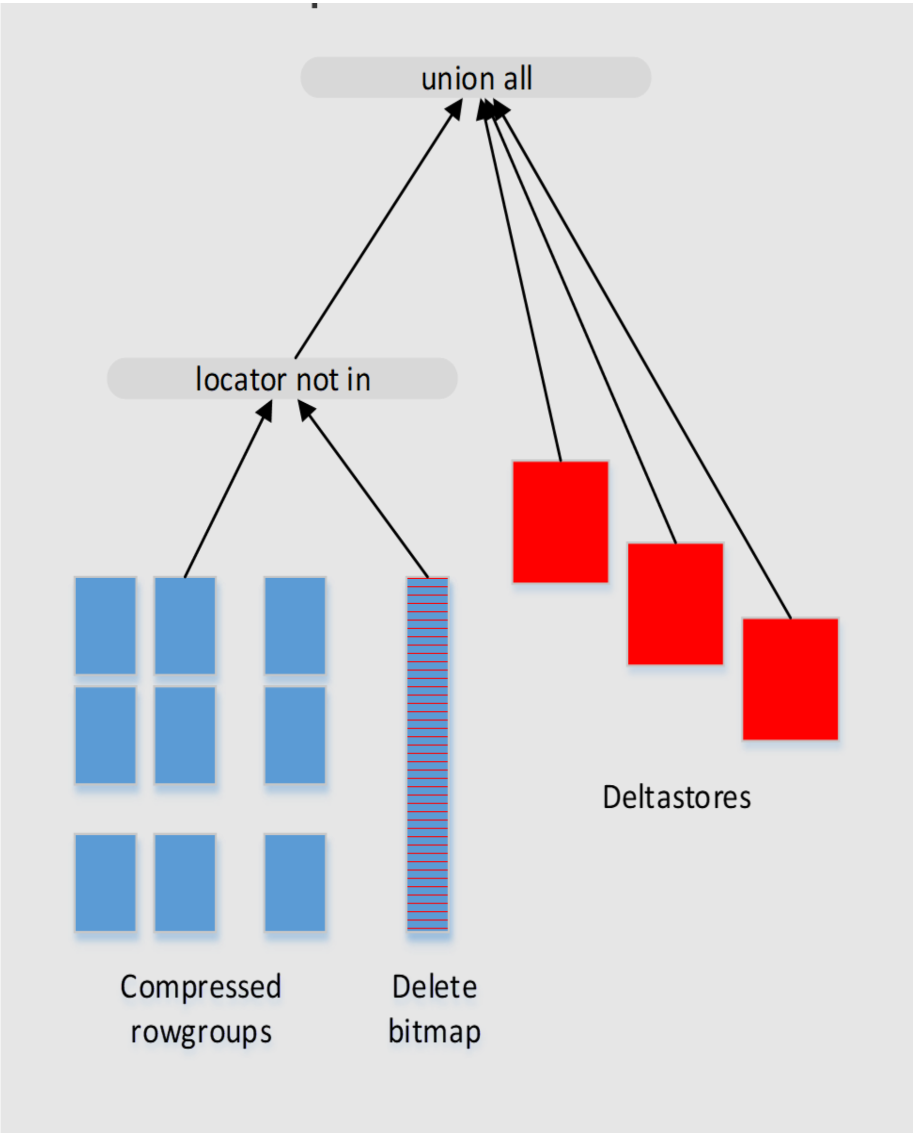
Be aware of the default

Not efficient for

- transient data (frequent updates/deletes)
- Singleton loads or micro-batches
- Small tables (<100 Million rows)

>100K rows per rowgroup

Highest possible RC for loading



Tables – Distributions

Round-robin distributed

Distributes table rows evenly across all distributions at random.

Hash distributed

Distributes table rows across the Compute nodes by using a deterministic hash function to assign each row to one distribution.

Replicated

Full copy of table accessible on each Compute node.

```
CREATE TABLE dbo.OrderTable
(
    OrderId   INT NOT NULL,
    Date      DATE NOT NULL,
    Name      VARCHAR(2),
    Country   VARCHAR(2)
)
WITH
(
  CLUSTERED COLUMNSTORE INDEX,
  DISTRIBUTION = HASH([OrderId]) |
                 ROUND ROBIN |
                 REPLICATED
);
```

Tables – Partitions

Overview

Table partitions divide data into smaller groups

In most cases, partitions are created on a date column

Supported on all table types

RANGE RIGHT – Used for time partitions

RANGE LEFT – Used for number partitions

Benefits

Improves efficiency and performance of loading and querying by limiting the scope to subset of data.

Offers significant query performance enhancements where filtering on the partition key can eliminate unnecessary scans and eliminate IO.

```sql
CREATE TABLE partitionedOrderTable
(
    OrderId  INT NOT NULL,
    Date     DATE NOT NULL,
    Name     VARCHAR(2),
    Country  VARCHAR(2)
)
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = HASH([OrderId]),
    PARTITION (
    [Date] RANGE RIGHT FOR VALUES (
    '2000-01-01', '2001-01-01', '2002-01-01’,
    '2003-01-01', '2004-01-01', '2005-01-01'
    )
  )
);
```

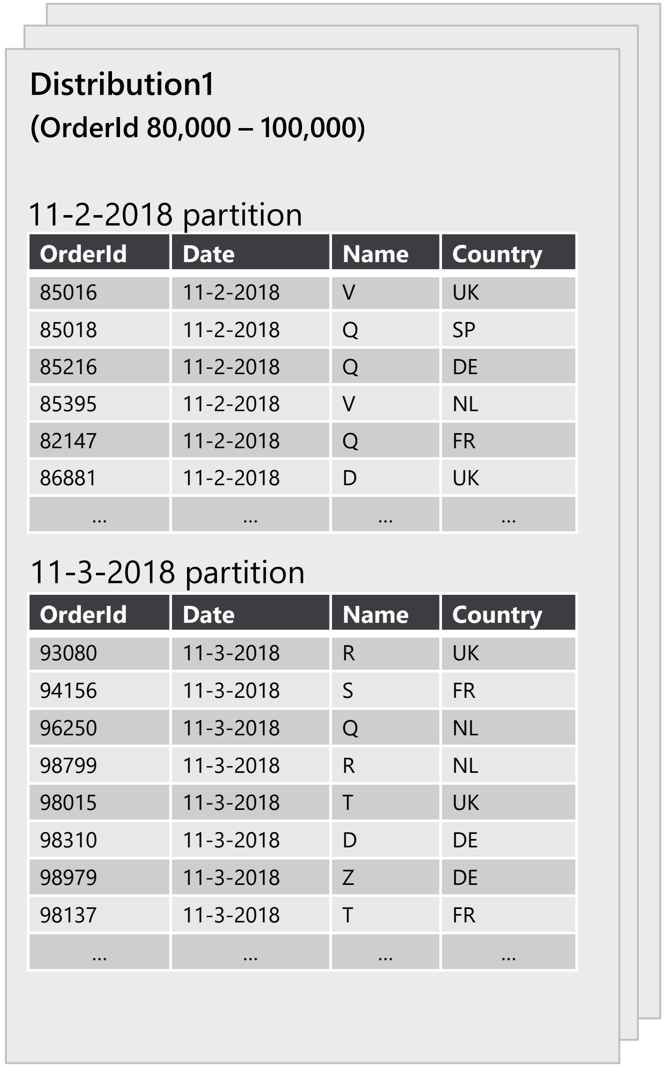
Tables – DW Distributions & Partitions Illustrated

Logical table structure

OrderId	Date	Name	Country
85016	11-2-2018	V	UK
85018	11-2-2018	Q	SP
85216	11-2-2018	Q	DE
85395	11-2-2018	V	NL
82147	11-2-2018	Q	FR
86881	11-2-2018	D	UK
93080	11-3-2018	R	UK
94156	11-3-2018	S	FR
96250	11-3-2018	Q	NL
98799	11-3-2018	R	NL
98015	11-3-2018	T	UK
98310	11-3-2018	D	DE
98979	11-3-2018	Z	DE
98137	11-3-2018	T	FR
…	…	…	…

Physical data distribution

(Hash distribution (OrderId), Date partitions)



Distribution1

(OrderId 80,000 – 100,000)

11-2-2018 partition

OrderId	Date	Name	Country
85016	11-2-2018	V	UK
85018	11-2-2018	Q	SP
85216	11-2-2018	Q	DE
85395	11-2-2018	V	NL
82147	11-2-2018	Q	FR
86881	11-2-2018	D	UK
…	…	…	…

11-3-2018 partition

OrderId	Date	Name	Country
93080	11-3-2018	R	UK
94156	11-3-2018	S	FR
96250	11-3-2018	Q	NL
98799	11-3-2018	R	NL
98015	11-3-2018	T	UK
98310	11-3-2018	D	DE
98979	11-3-2018	Z	DE
98137	11-3-2018	T	FR
…	…	…	…

…

x 60 distributions (shards)

- Each shard is partitioned with the same date partitions
- A minimum of 1 million rows per distribution and partition is needed for optimal compression and performance of clustered Columnstore tables

Demo

Common table distribution methods

Table Category	Recommended Distribution Option
Fact	Use hash-distribution with clustered columnstore index. Performance improves because hashing enables the platform to localize certain operations within the node itself during query execution. Operations that benefit: COUNT(DISTINCT(<hashed_key>)) OVER PARTITION BY <hashed_key> JOIN ON <hashed_key>
Dimension	Use replicated for smaller tables. If tables are too large to store on each Compute node, use hash-distributed.
Staging	Use round-robin for the staging table. The load with CTAS is faster. Once the data is in the staging table, use INSERT...SELECT to move the data to production tables.

Questions?