Lambda Architecture

Santosh Hari

Nebbia
TECHNOLOGY

Santosh Hari

Azure Consultant @ Nebbia Tech

Azure MVP

President, Orlando .NET UG

Organizer, Orlando Codecamp

santoshhari.wordpress.com

santosh.hari@newsignature.com

@_s_hari

/in/santoshhari

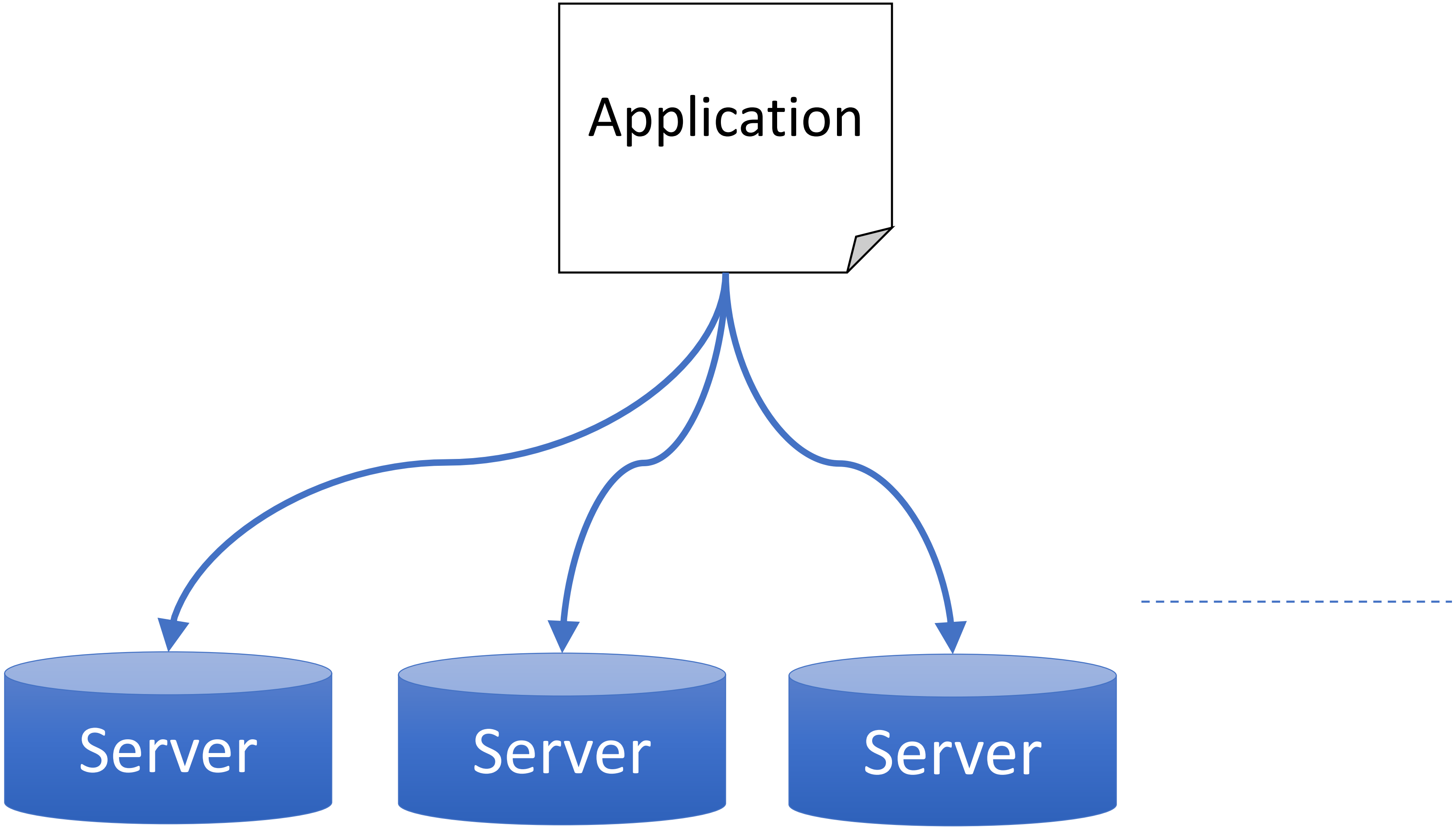
MVP
Microsoft
Most Valuable
Professional

Agenda

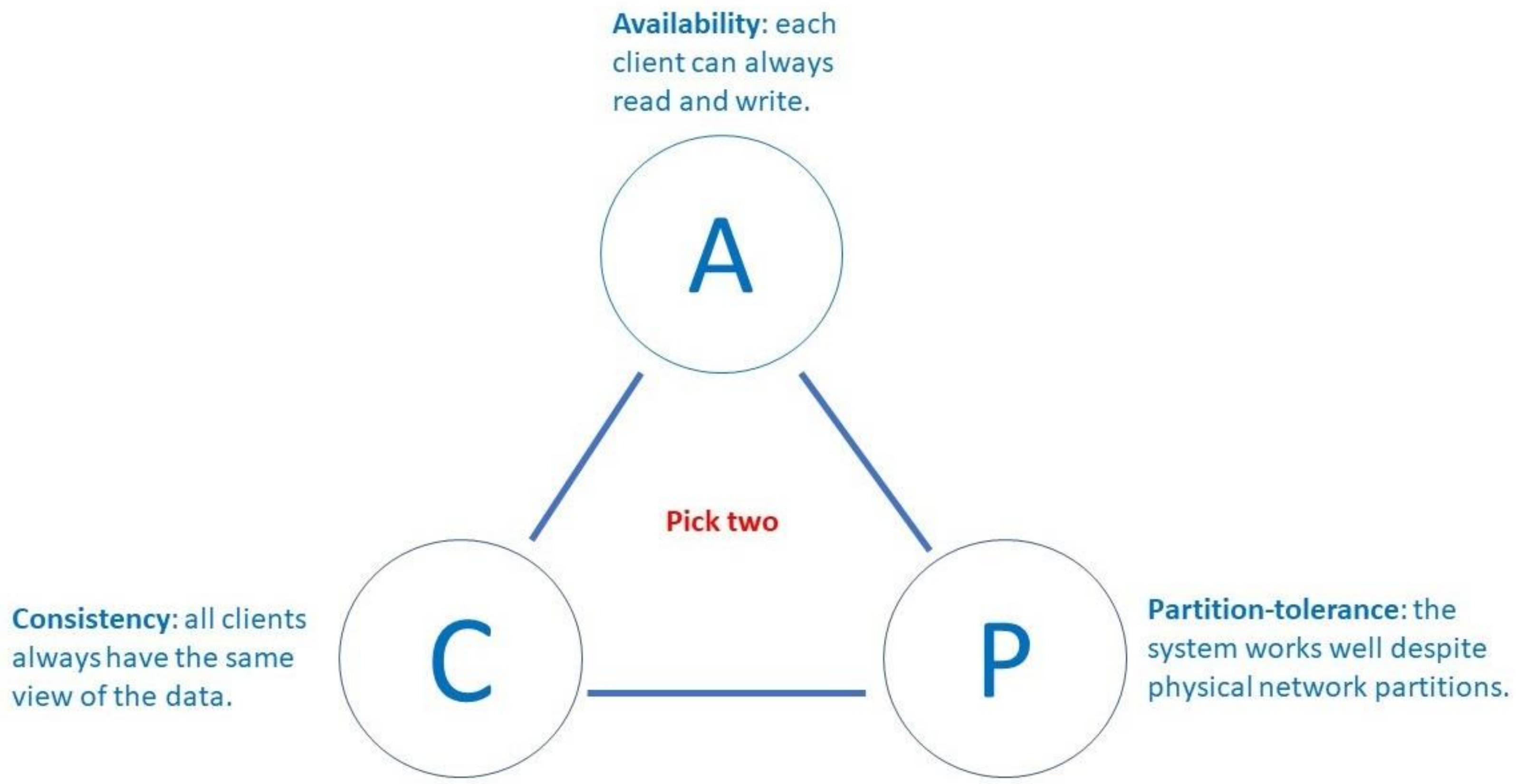
- Intro to Lambda Architecture (LA)
- Intro to Cosmos DB
- Discussion on building blocks
  - Materialized View
  - Event Sourcing
  - CQRS
- Big picture view
- Discussion on how Cosmos DB simplifies LA
- Criticisms of LA and short discussion on Kappa Architecture
- Heavy on concepts, no code or demo

Intro to Lambda Architecture

Modern data system scale horizontal



CAP Theorem



Availability: each client can always read and write.

Consistency: all clients always have the same view of the data.

Partition-tolerance: the system works well despite physical network partitions.

Challenge of data processing in distributed systems

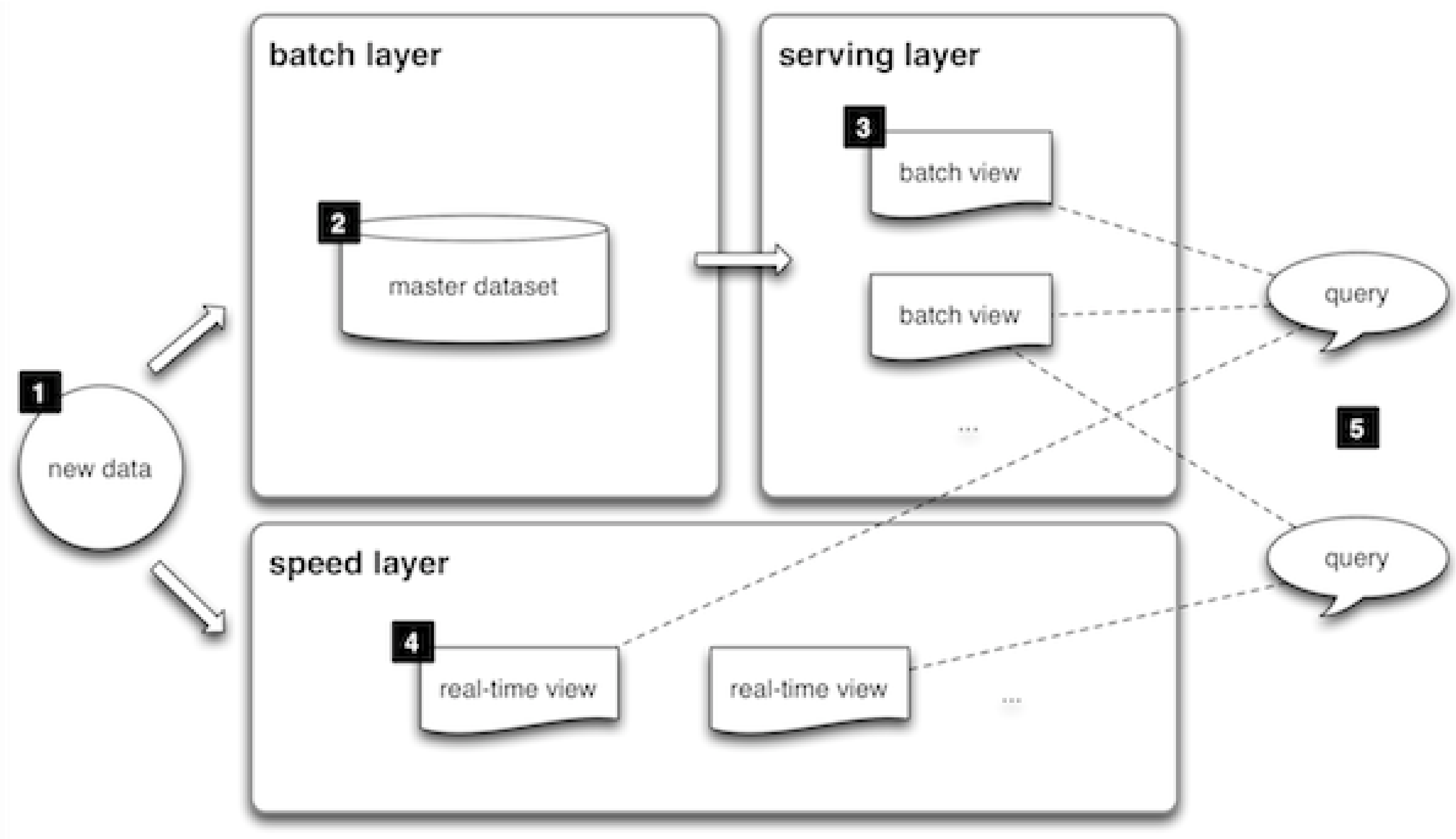
- High latency
  - Unable to read latest writes
  - Keep data in it’s original form

- Low latency
  - Issues querying large amounts of historical data
  - Fault tolerance
  - Not very scalable

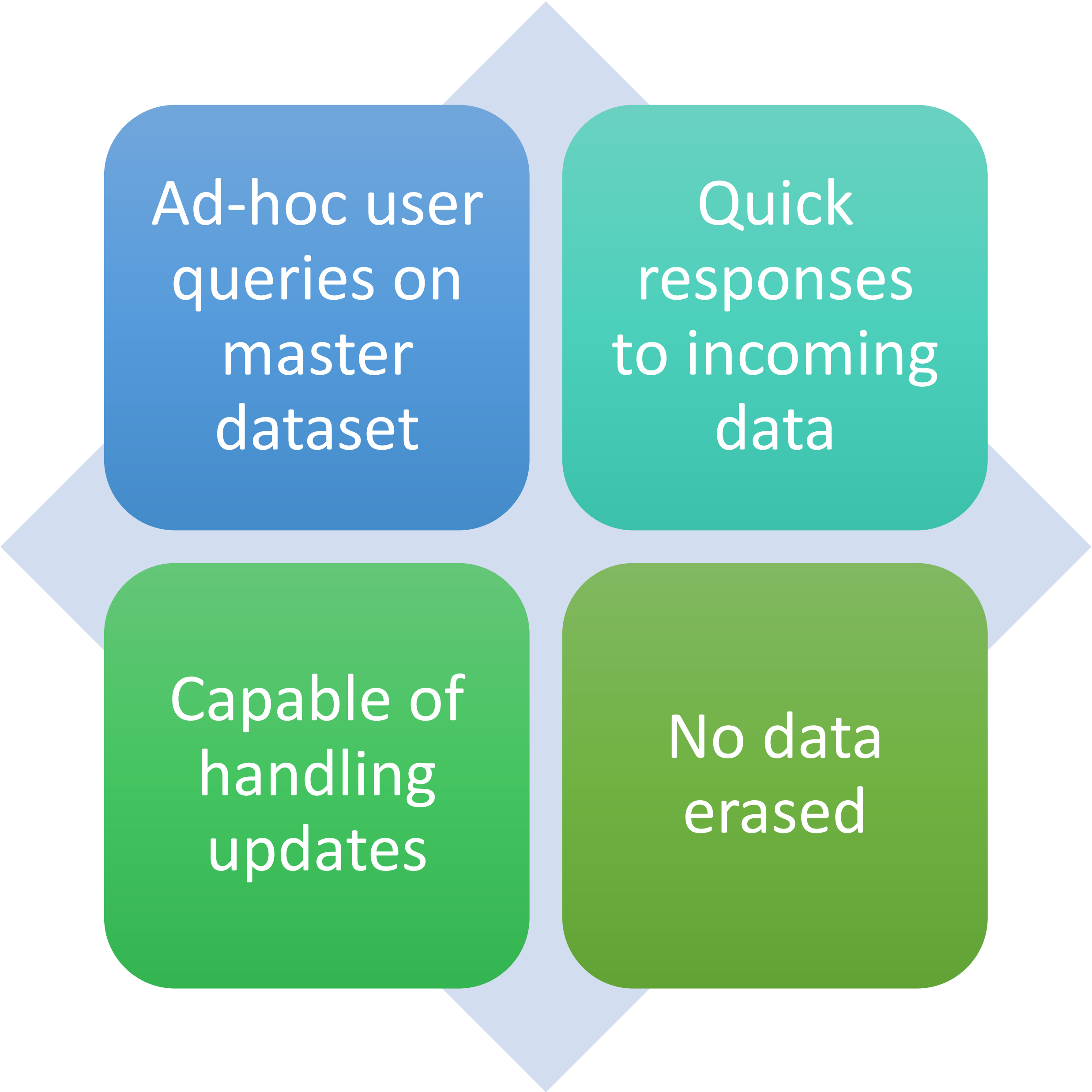
What is Lambda Architecture?

- Term coined by Nathan Marz in 2012
- Distributed data processing architecture
- Generic, scalable
- Robust, fault tolerant against system and human failure
- Enables low latency reads and updates
- Scales horizontally

Lambda Architecture Overview



Use Case Scenarios



Ad-hoc user queries on master dataset

Quick responses to incoming data

Capable of handling updates

No data erased

Pros and Cons

CONS

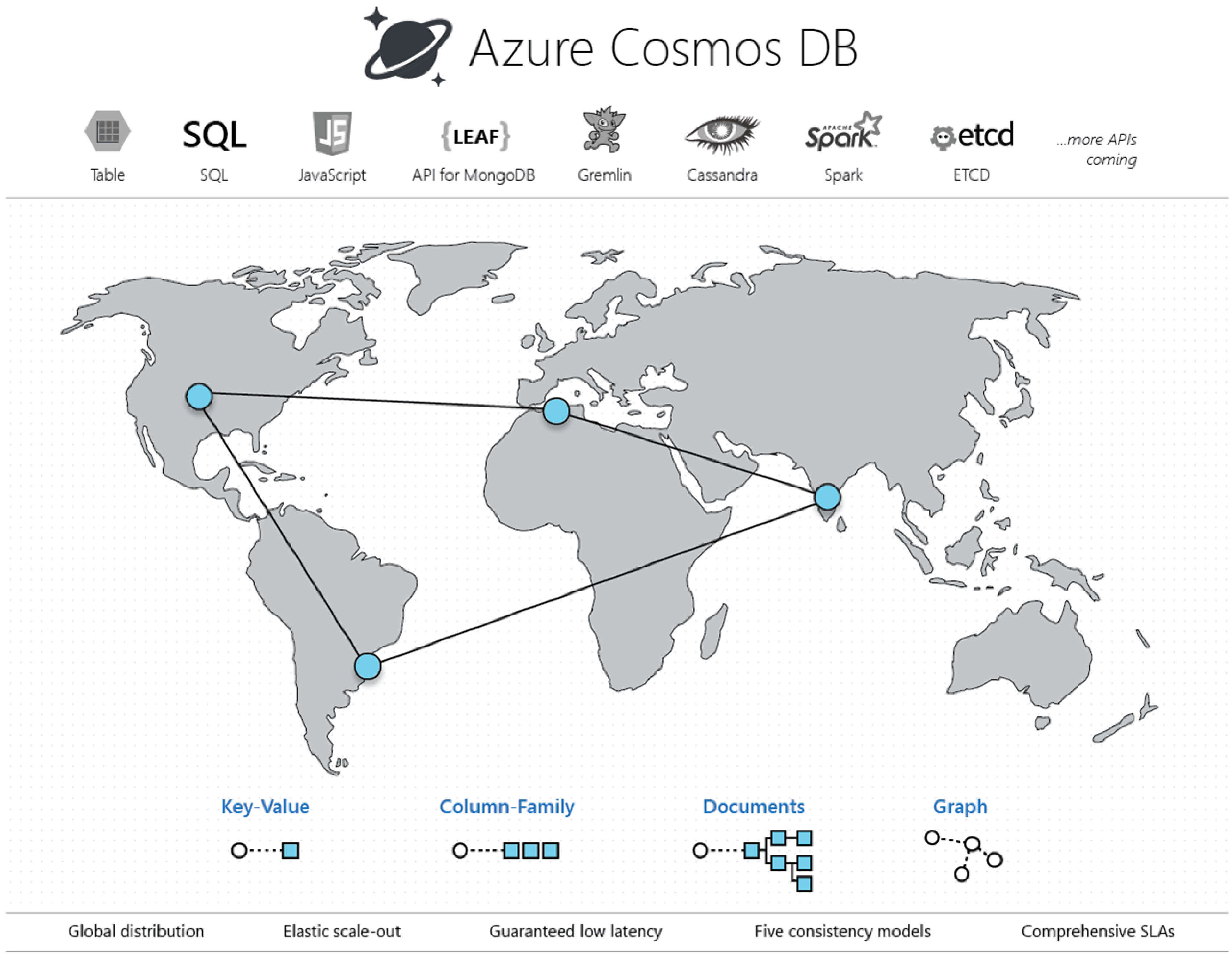
- Complex
- Re-processes every batch cycle
- Difficult to migrate / reorganize

PROS

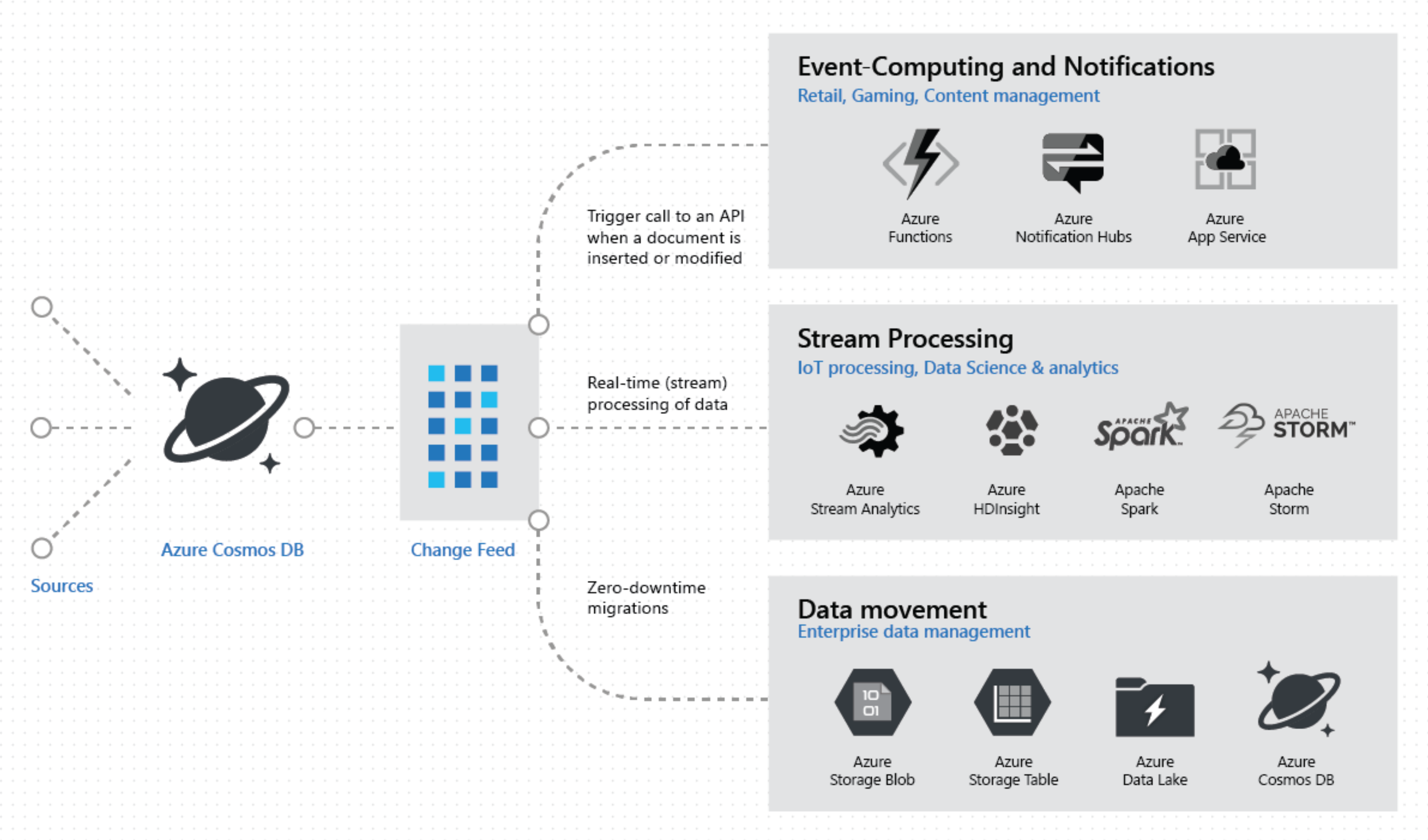
- History
- Less errors
- Speed and reliability.
- Fault tolerant
- Scalable

Cosmos DB

Enter Cosmos DB



Cosmos DB Change Feed



Event-Computing and Notifications

Retail, Gaming, Content management

Stream Processing

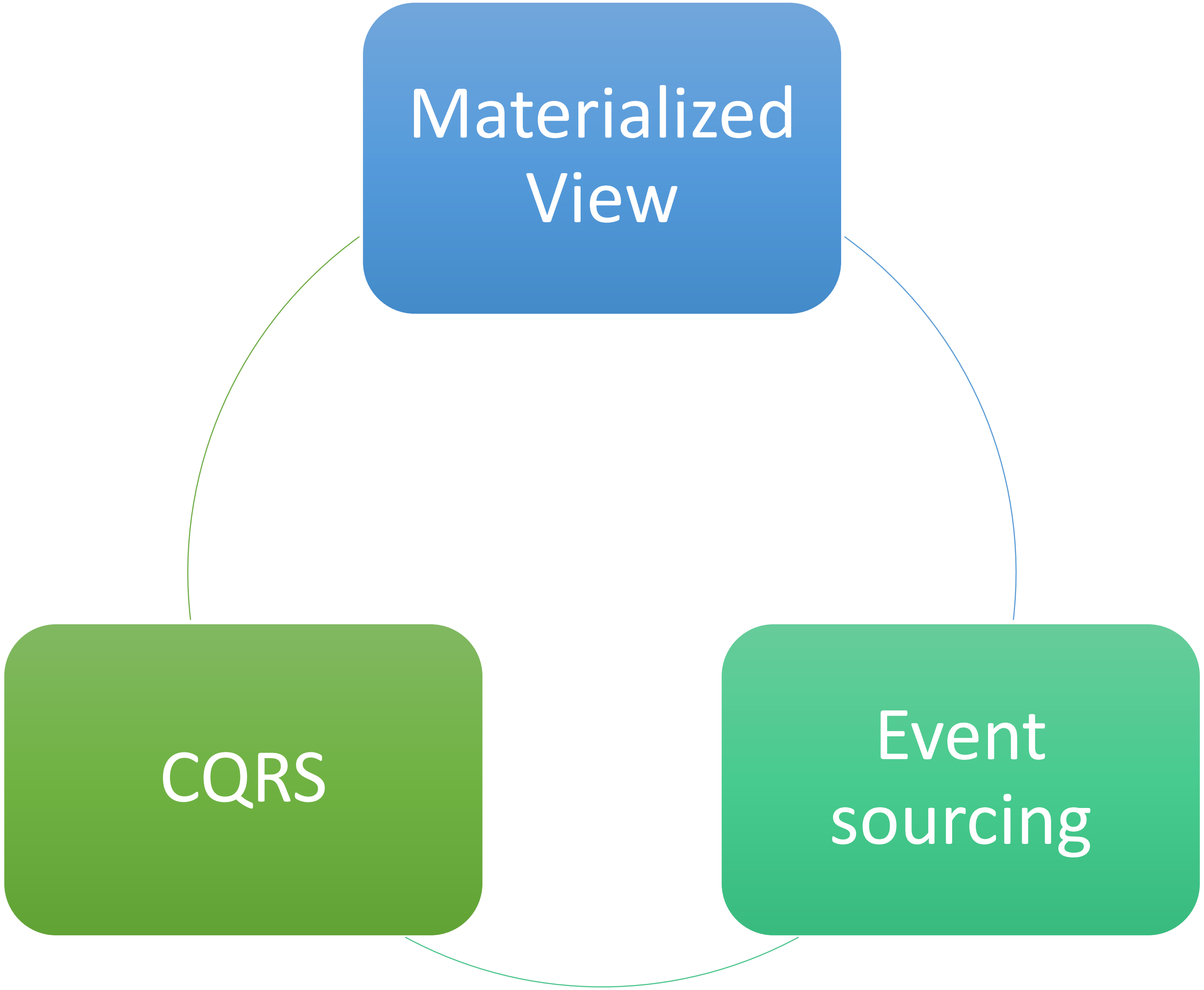
IoT processing, Data Science & analytics

Data movement

Enterprise data management

Relevant Patterns

Patterns to know

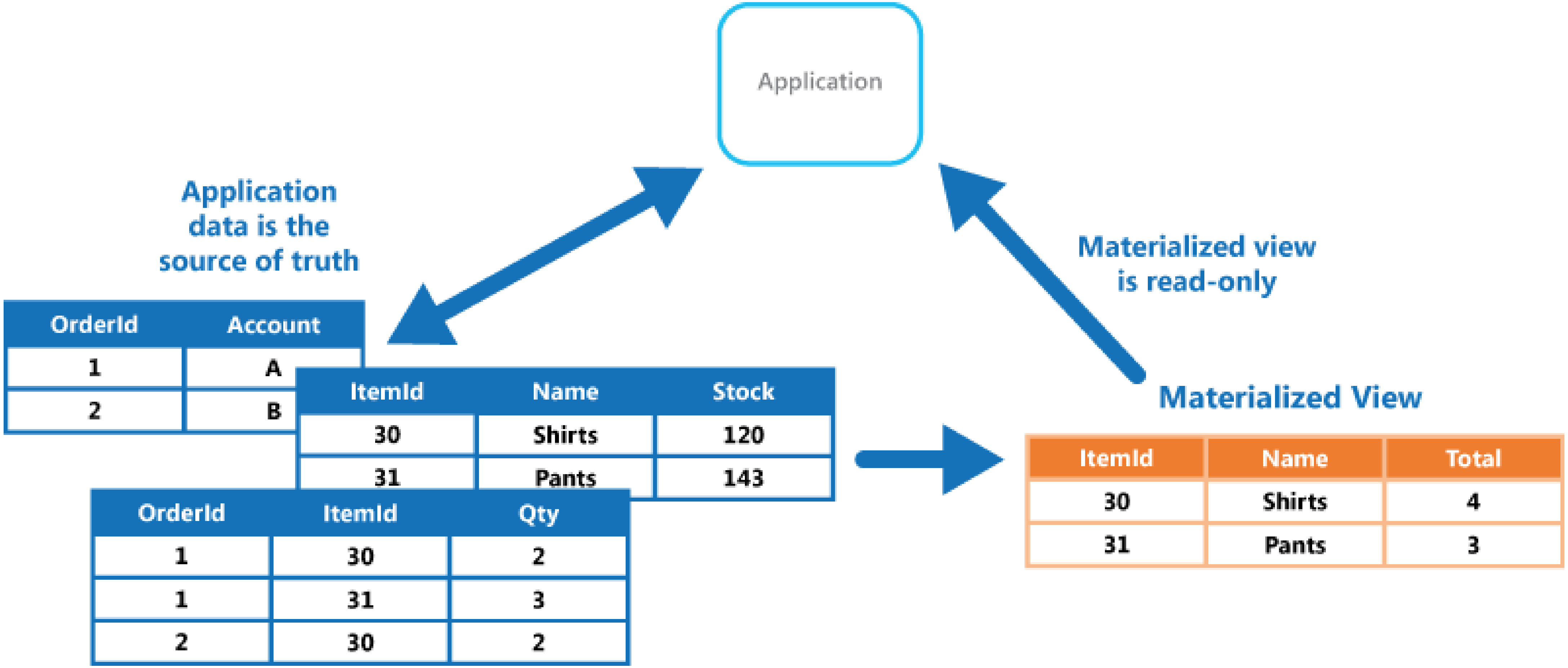


Materialized View

Event sourcing

CQRS

Materialized View



Application data is the source of truth

Materialized view is read-only

OrderId	Account
1	A
2	B

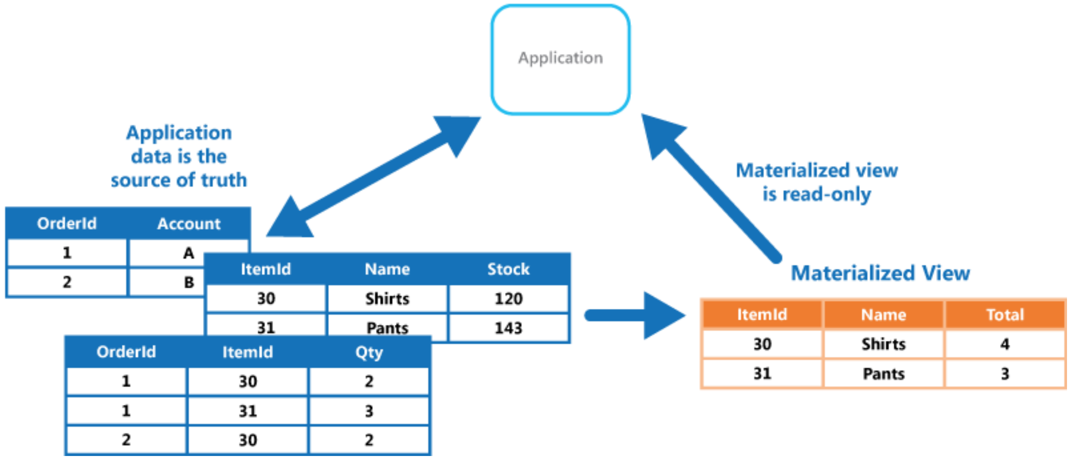
ItemId	Name	Stock
30	Shirts	120
31	Pants	143

OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

Materialized View

ItemId	Name	Total
30	Shirts	4
31	Pants	3

What problems does Materialized View solve


Application
Application data is the source of truth
Materialized view is read-only

OrderId	Account
1	A
2	B

ItemId	Name	Stock
30	Shirts	120
31	Pants	143

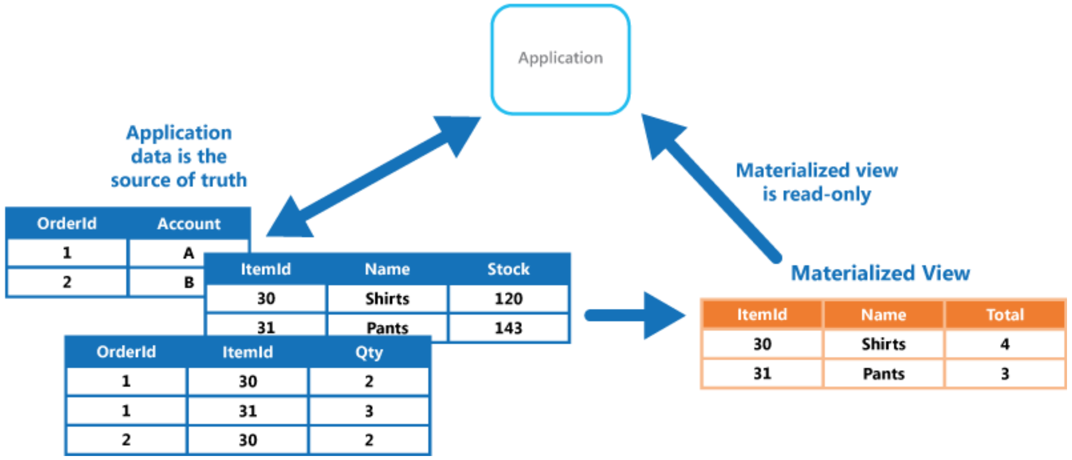
OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

Materialized View

ItemId	Name	Total
30	Shirts	4
31	Pants	3

- Data optimized for storing or reading
- Negative effect on queries

How does Materialized View solve these problems


Application
Application data is the source of truth
Materialized view is read-only
Materialized View

OrderId	Account
1	A
2	B

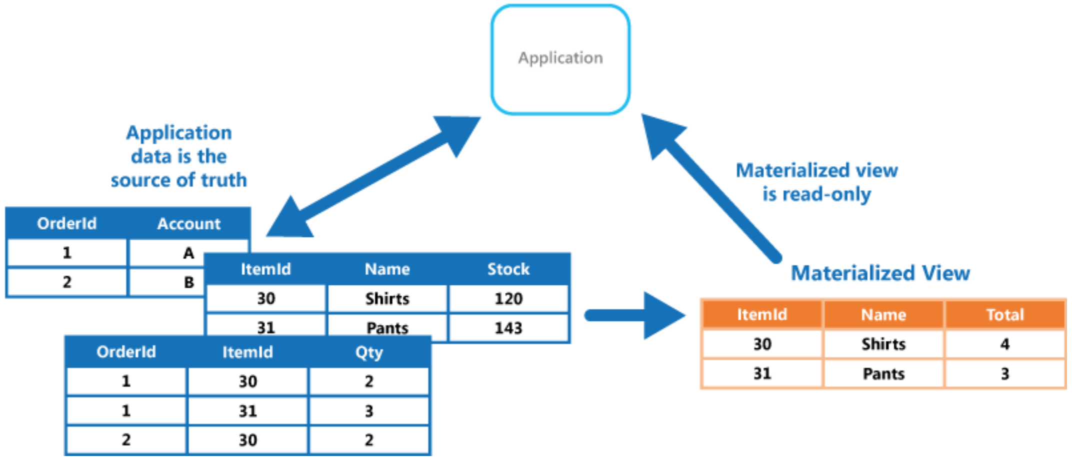
ItemId	Name	Stock
30	Shirts	120
31	Pants	143

OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

ItemId	Name	Total
30	Shirts	4
31	Pants	3

- Contain only read data
- Subset
- Disposable
- Easy re-generation
- Cache

Materialized View considerations


Application
Application data is the source of truth
Materialized view is read-only
Materialized View

OrderId	Account
1	A
2	B

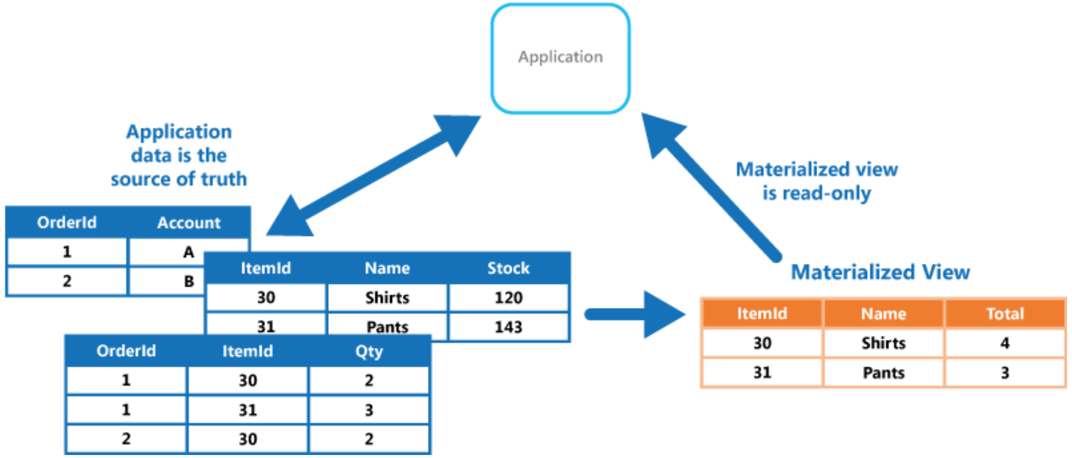
ItemId	Name	Stock
30	Shirts	120
31	Pants	143

OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

ItemId	Name	Total
30	Shirts	4
31	Pants	3

- Handling updates
- **Eventual consistency**
- Storage
- Index and partitioning

Materialized View use cases - good



Application data is the source of truth

OrderId	Account
1	A
2	B

ItemId	Name	Stock
30	Shirts	120
31	Pants	143

OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

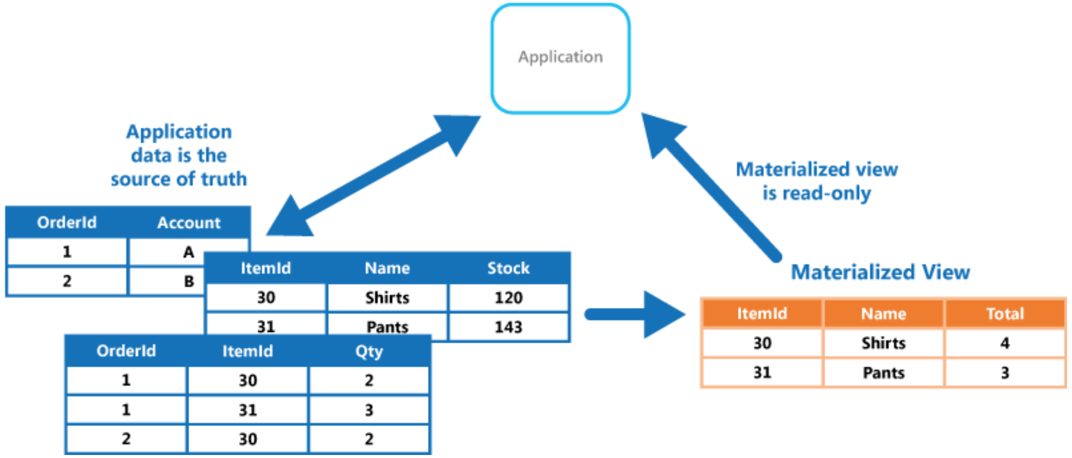
Materialized view is read-only

Materialized View

ItemId	Name	Total
30	Shirts	4
31	Pants	3

- Handles difficult to query data
- Perf
- Local cache
- Separate query from source
- Bridging data sources

Materialized View use cases - bad


Application
Application data is the source of truth
Materialized view is read-only
Materialized View

OrderId	Account
1	A
2	B

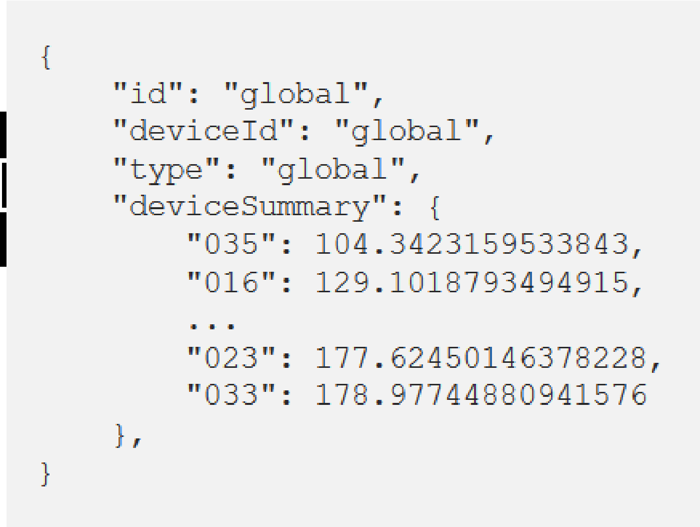
ItemId	Name	Stock
30	Shirts	120
31	Pants	143

OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

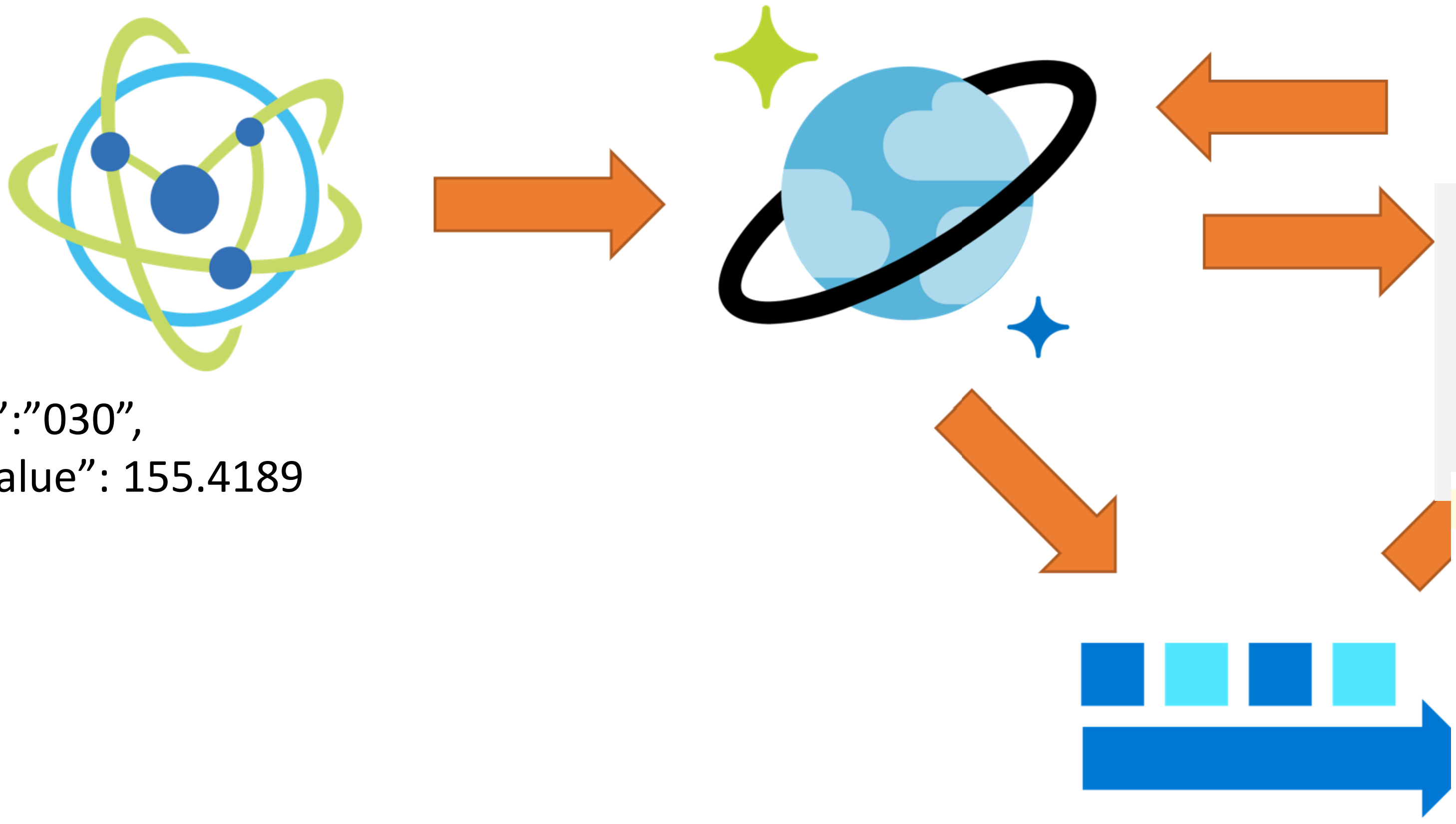
ItemId	Name	Total
30	Shirts	4
31	Pants	3

- Simple to query data
- Rapidly changing data
- Consistency needed

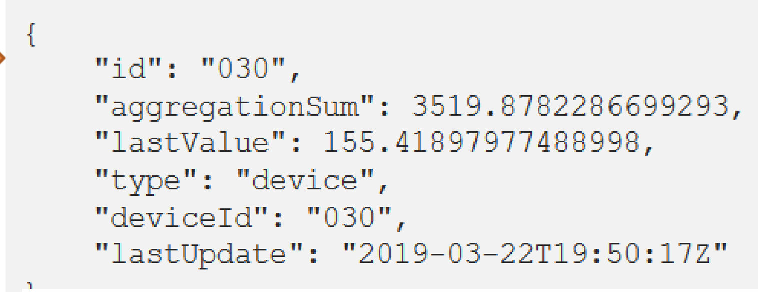
Materialized View with Cosmos D



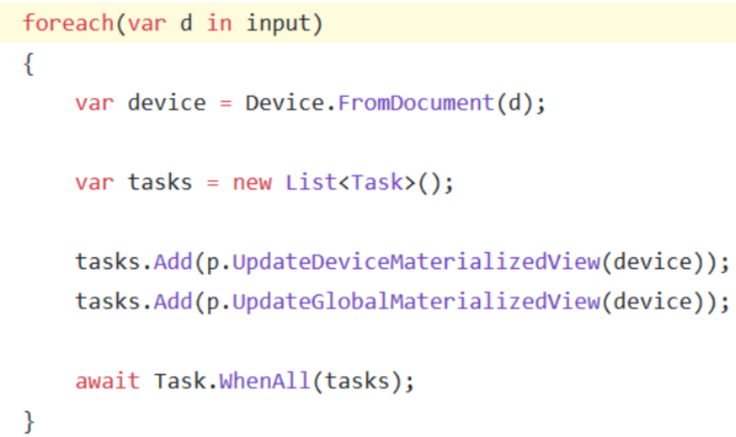
```
{
    "id": "global",
    "deviceId": "global",
    "type": "global",
    "deviceSummary": {
        "035": 104.3423159533843,
        "016": 129.1018793494915,
        ...
        "023": 177.62450146378228,
        "033": 178.97744880941576
    },
}
```



```
{
    “deviceId”:”030”,
    “currentValue”: 155.4189
}
```



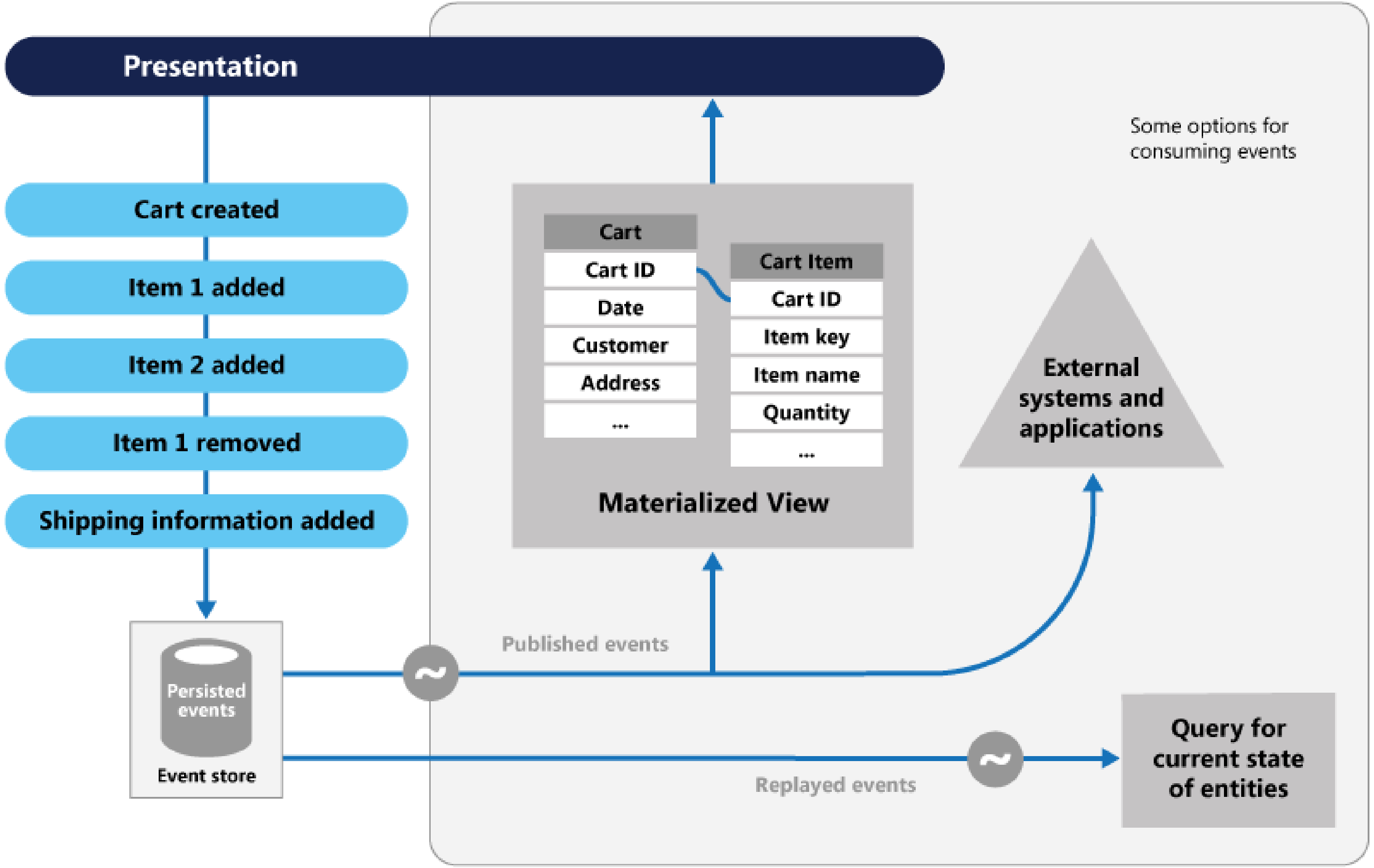
```
{
    "id": "030",
    "aggregationSum": 3519.8782286699293,
    "lastValue": 155.41897977488998,
    "type": "device",
    "deviceId": "030",
    "lastUpdate": "2019-03-22T19:50:17Z"
```



```
foreach(var d in input)
{
    var device = Device.FromDocument(d);

    var tasks = new List<Task>();

    tasks.Add(p.UpdateDeviceMaterializedView(device));
    tasks.Add(p.UpdateGlobalMaterializedView(device));

    await Task.WhenAll(tasks);
}
```

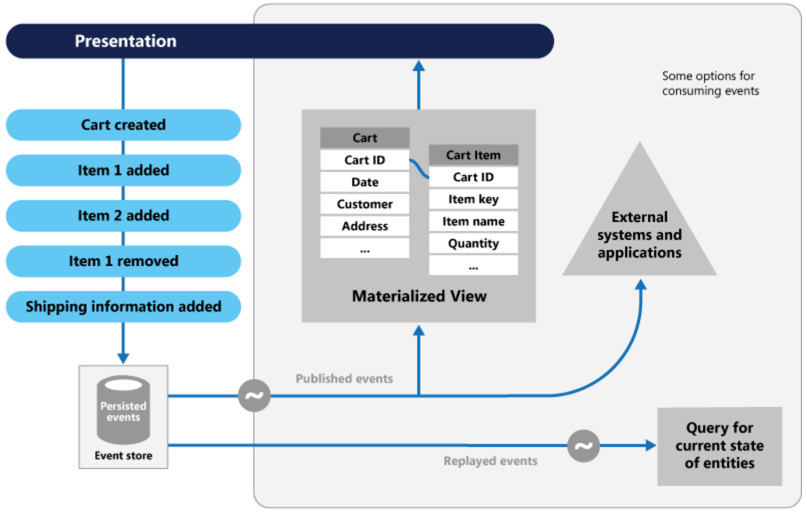
https://medium.com/@mauridb/real-time-materialized-views-with-cosmos-db-90ecea84f650

Event Sourcing



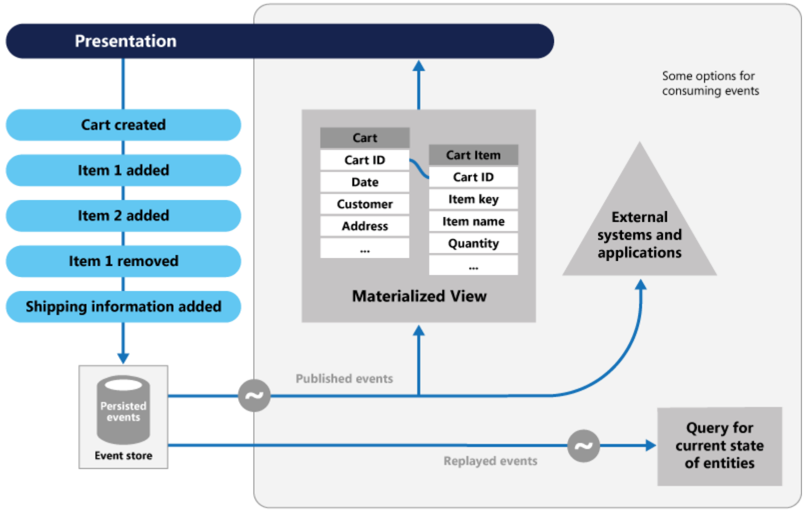
What problems does Event Sourcing solve?

- Current state only
- Locking
- Data loss
- Audit??



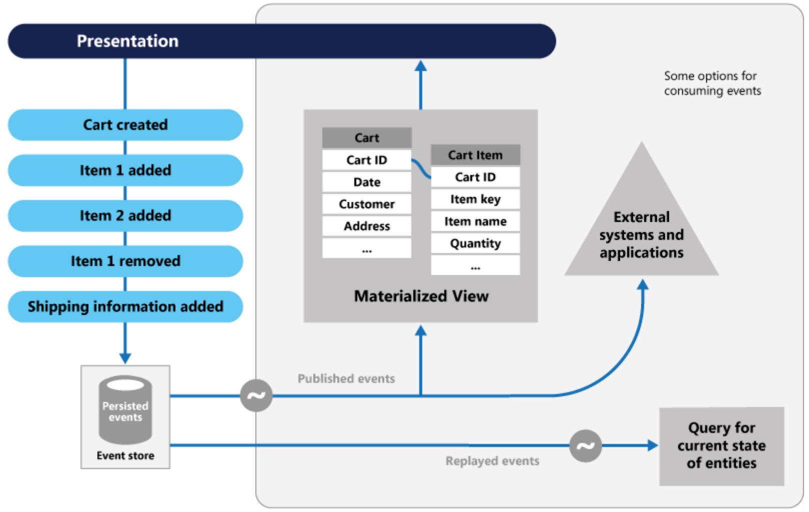
How Event Sourcing solves these problems

- Data-as-events-sequence
- Append-only store
- Store is authoritative
- Store publishes events
- Materialized View
- Replay events



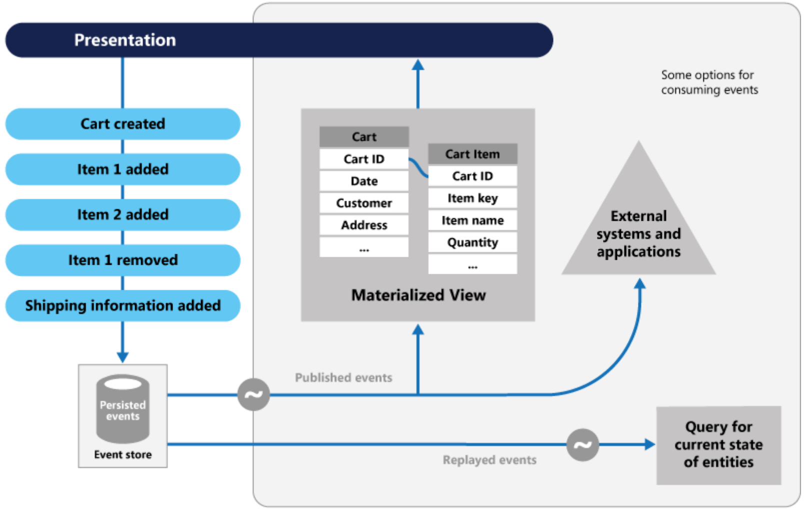
Event Sourcing advantages

- Immutable events
- Simple objects
- Domain experts friendly
- No direct DB updates
- Audit trail
- Decouple events from tasks



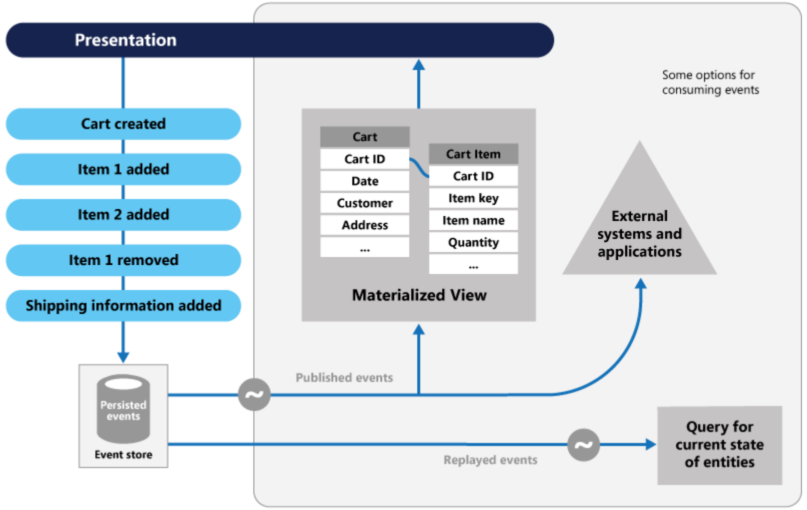
Event Sourcing considerations

- **Eventual Consistency**
- No out-of-band updates to event store
- Event format vital
- Order of events vital
- Current State = sum all
- Use snapshots
- Reduces data conflicts
- Consumers idempotent



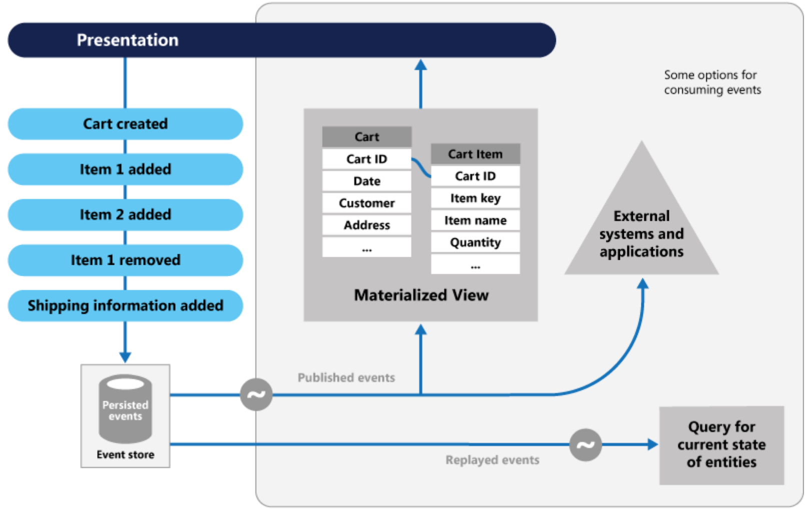
Event Sourcing use cases - good

- Capture intent, purpose, reason
- Avoiding conflicts
- Restore / rollback
- Decouple I/P O/P
- Great with CQRS



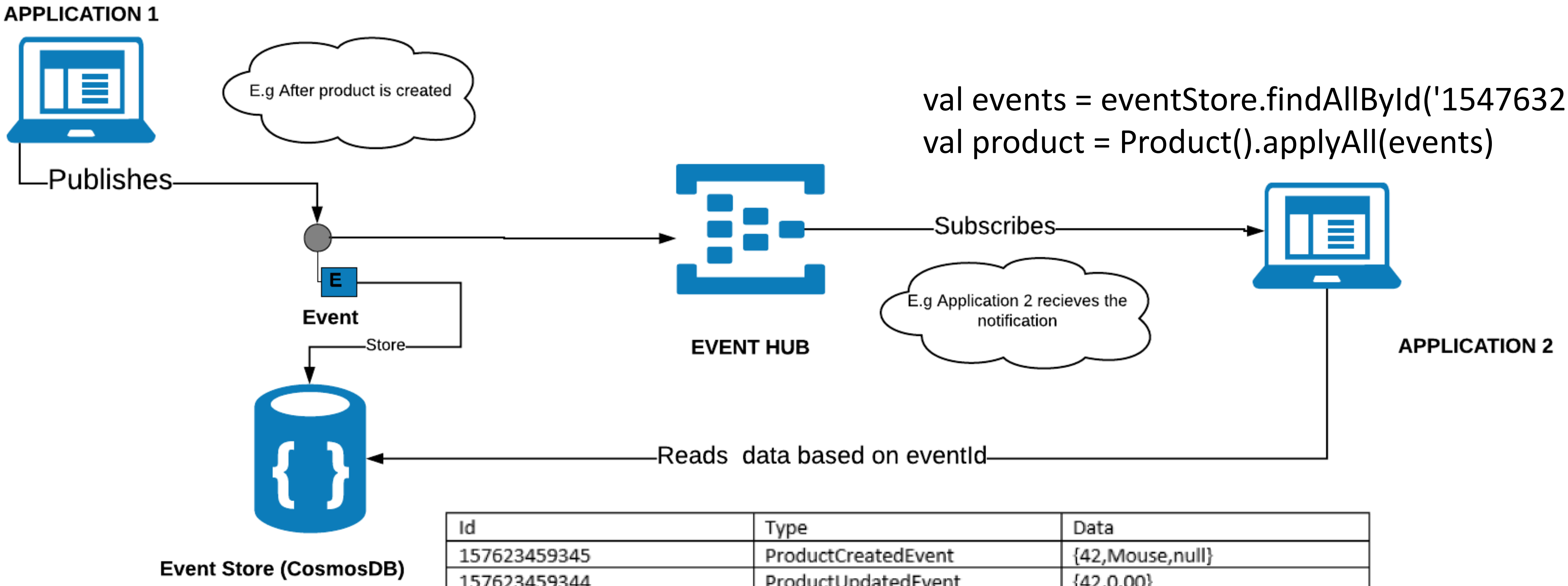
Event Sourcing use cases - bad

- Simple domain
- CRUD
- Consistent systems
- Real-time systems
- Non-audit non-historical systems
- Low conflict systems



Event Sourcing with Cosmos DB

{“MessageId”:1547632386819}

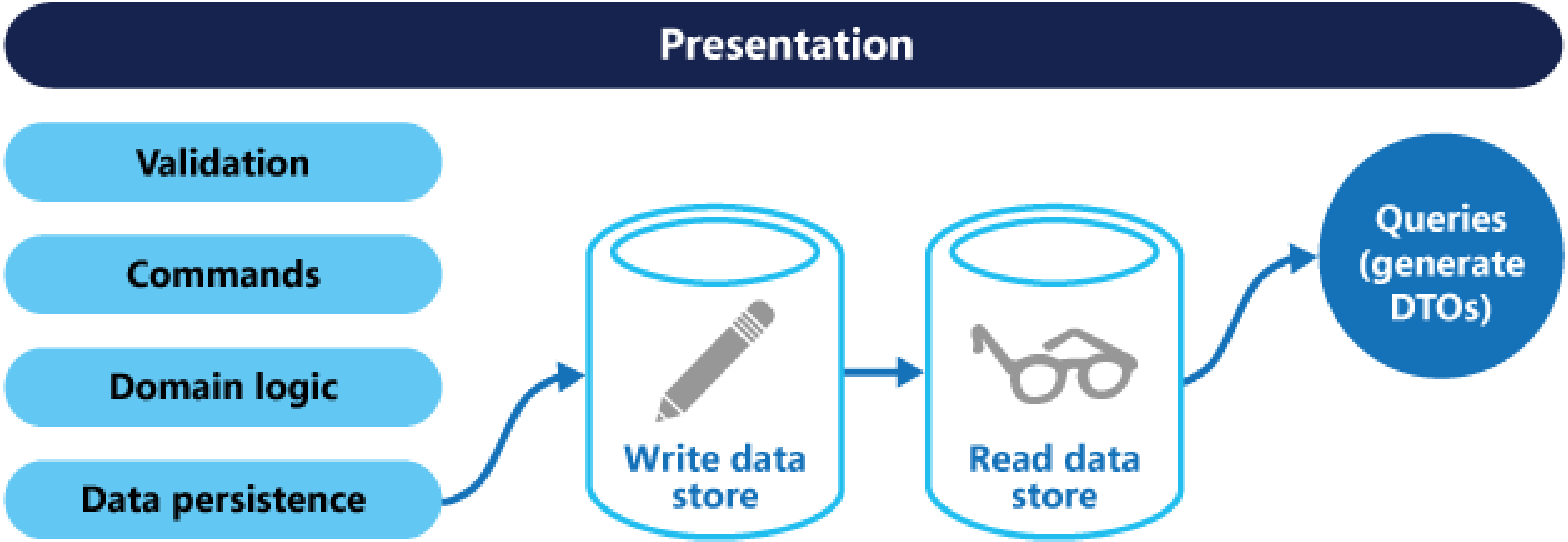

APPLICATION 1
E.g After product is created
Publishes
E
Event
Store
EVENT HUB
Subscribes
E.g Application 2 recieves the notification
APPLICATION 2
Reads data based on eventId
Event Store (CosmosDB)

```
val events = eventStore.findAllById('1547632386819');
val product = Product().applyAll(events)
```

Id	Type	Data
157623459345	ProductCreatedEvent	{42,Mouse,null}
157623459344	ProductUpdatedEvent	{42,0.00}
157623459343	ProductUpdatedEvent	{43,Keyboard,0.99}

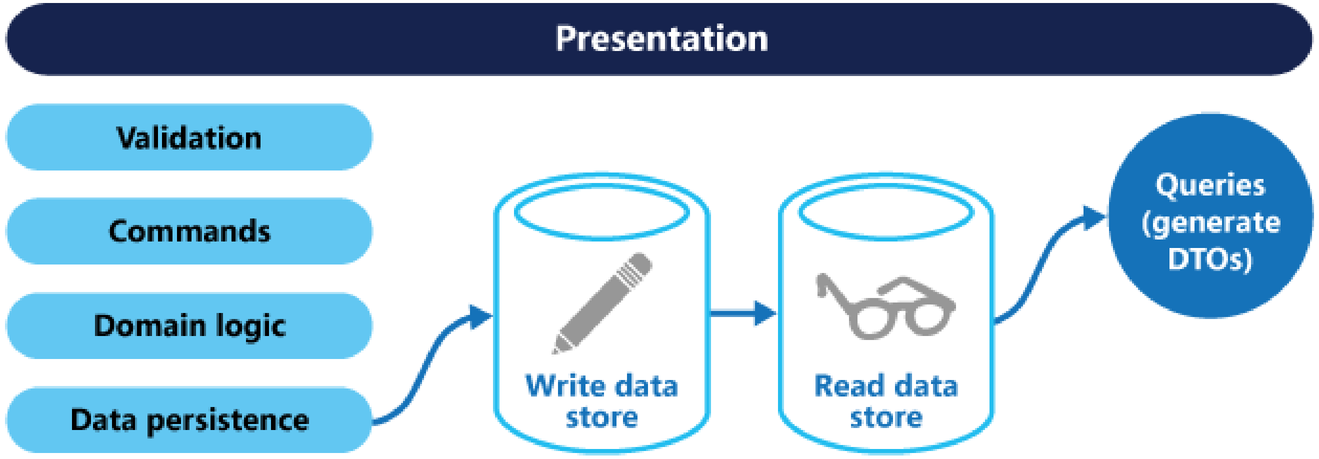
https://sajeetharan.com/2019/02/03/event-sourcing-with-azure-eventhub-and-cosmosdb/

CQRS


Presentation
Validation
Commands
Domain logic
Data persistence
Write data store
Read data store
Queries (generate DTOs)

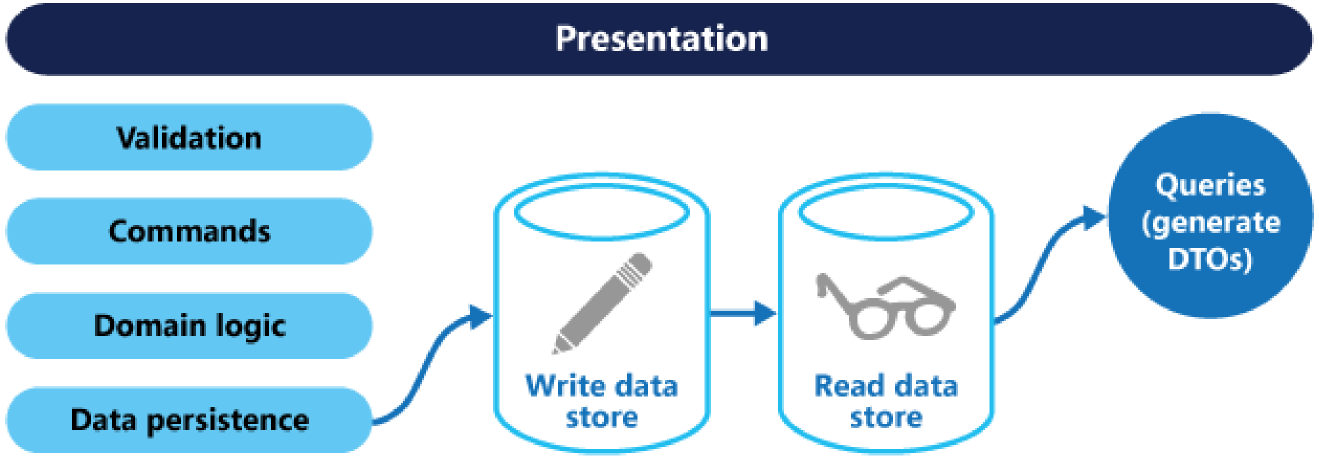
What problems does CQRS solve

- CRUD against same entities
- Scaffolding tools optimize for commands
- Columns with diff update frequencies
- Data contention
- Diff Security /Perms



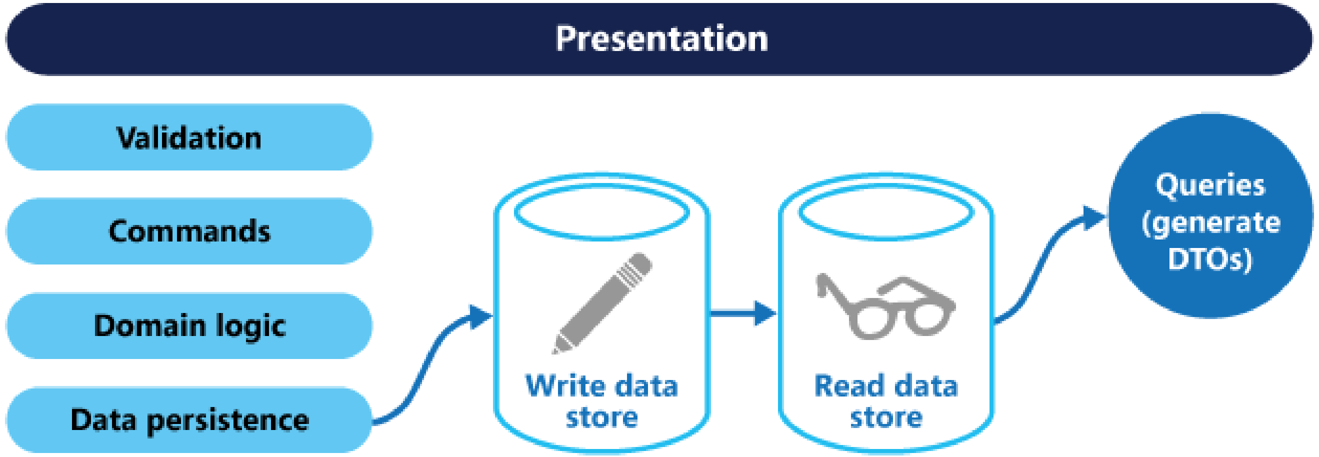
How does CQRS solve these problems?

- Segregate Reads and Commands
- Diff data models
- Separate R/W stores



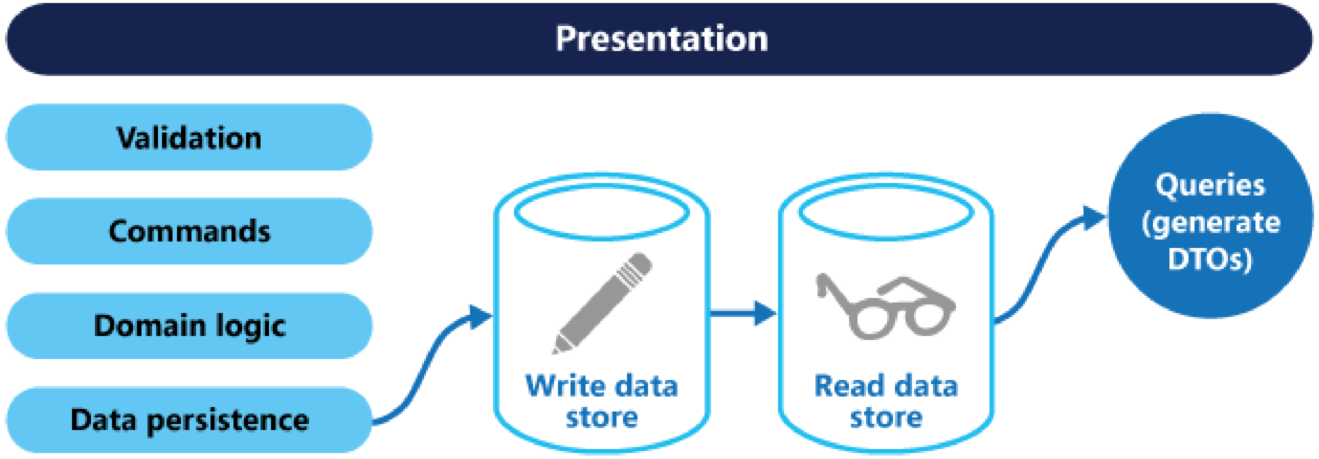
CQRS Considerations

- No Scaffolding
- Increase perf+security
- More complex
- Model change mgmt.
- Limited scope
- Task not data
- **<u>Eventual Consistency</u>**



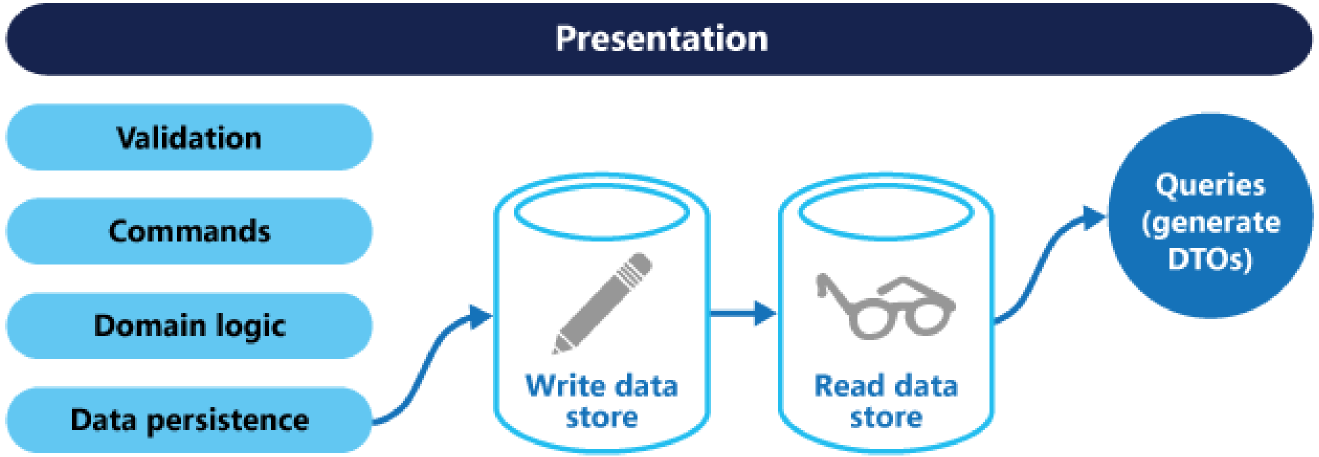
CQRS Use Cases - Good

- Collaborative domain
- DDD
- R to W ratios high
- Separation of concerns
- Different read models
- Evolving BL



CQRS Use Cases - Bad

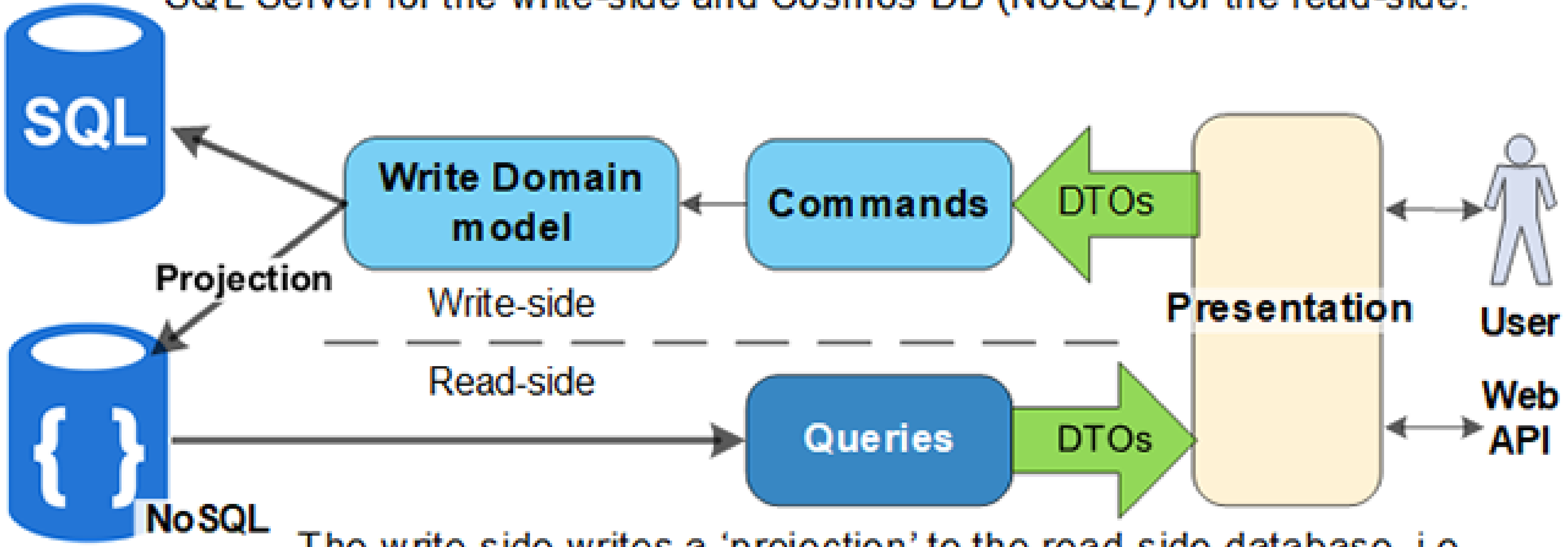
- Simple domains
- Less Data
- Whole systems



CQRS with Cosmos DB

CQRS database pattern, with separate write and read databases.

SQL Server for the write-side and Cosmos DB (NoSQL) for the read-side.



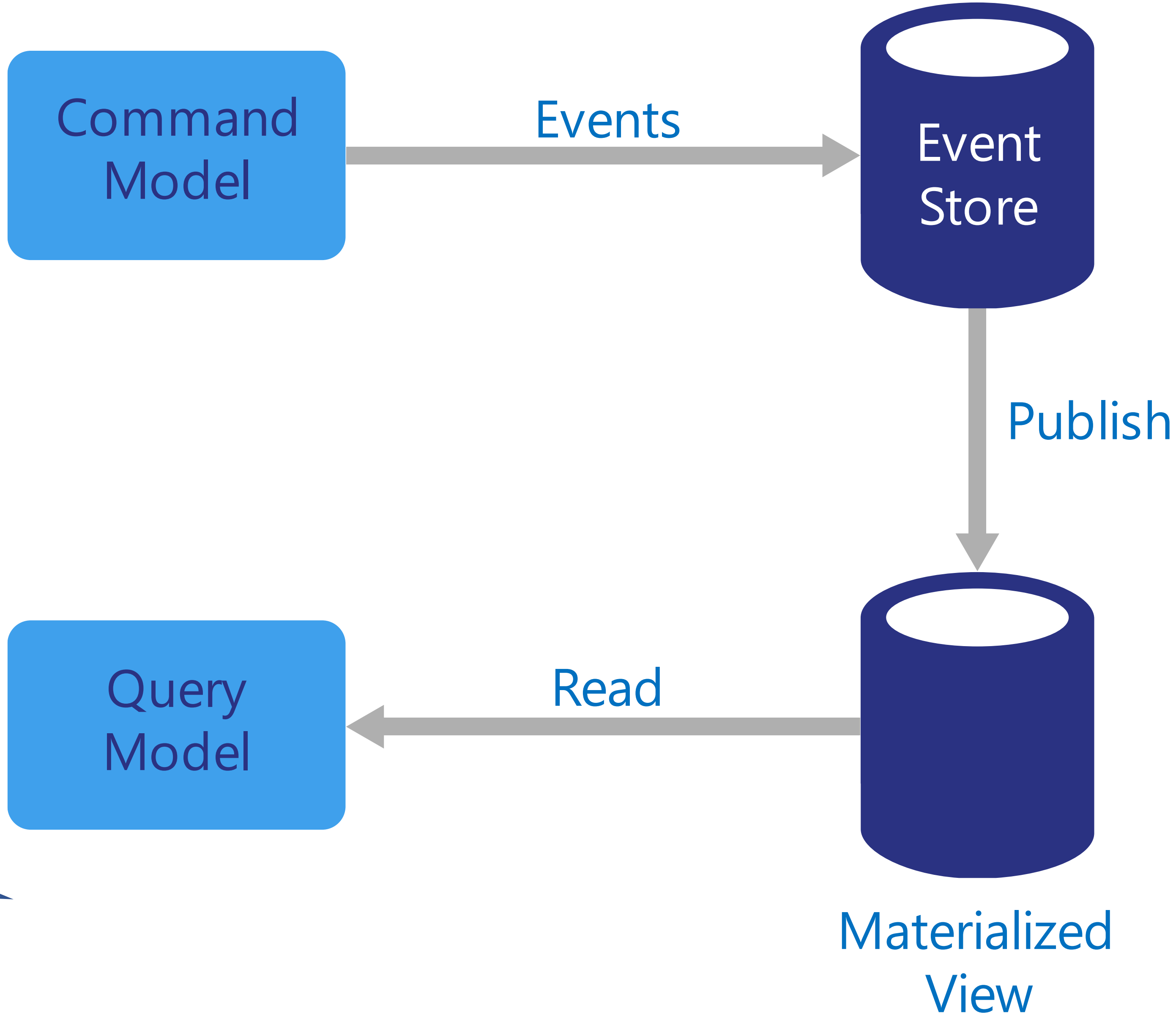
The write-side writes a ‘projection’ to the read-side database, i.e. the data is written in a form that is ready to display to the user.

https://www.thereformedprogrammer.net/building-a-robust-cqrs-database-with-ef-core-and-cosmos-db/

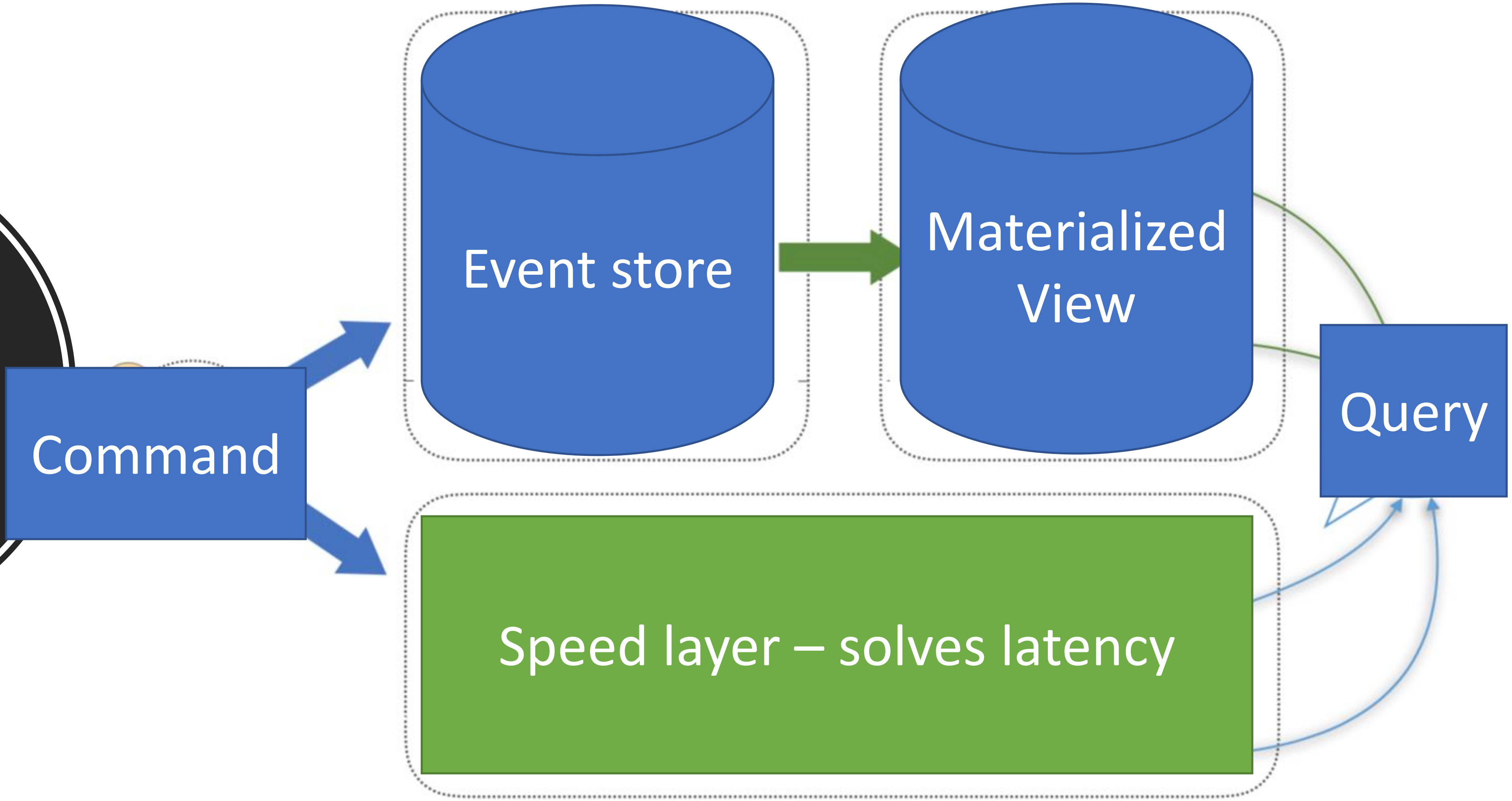
How it all ties together

Event sourcing + CQRS + Materialized views

Still Eventually Consistent

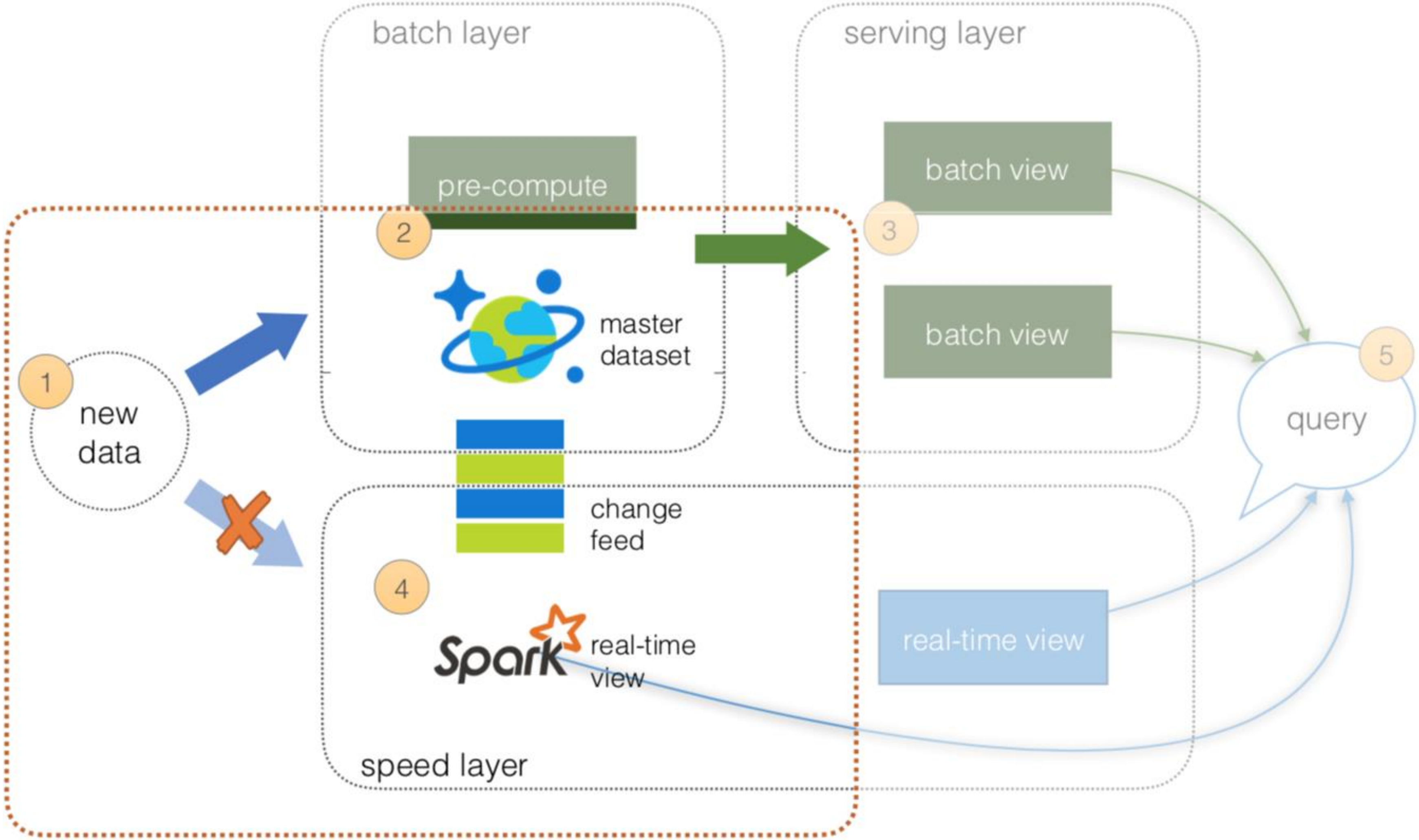


Lambda Architecture

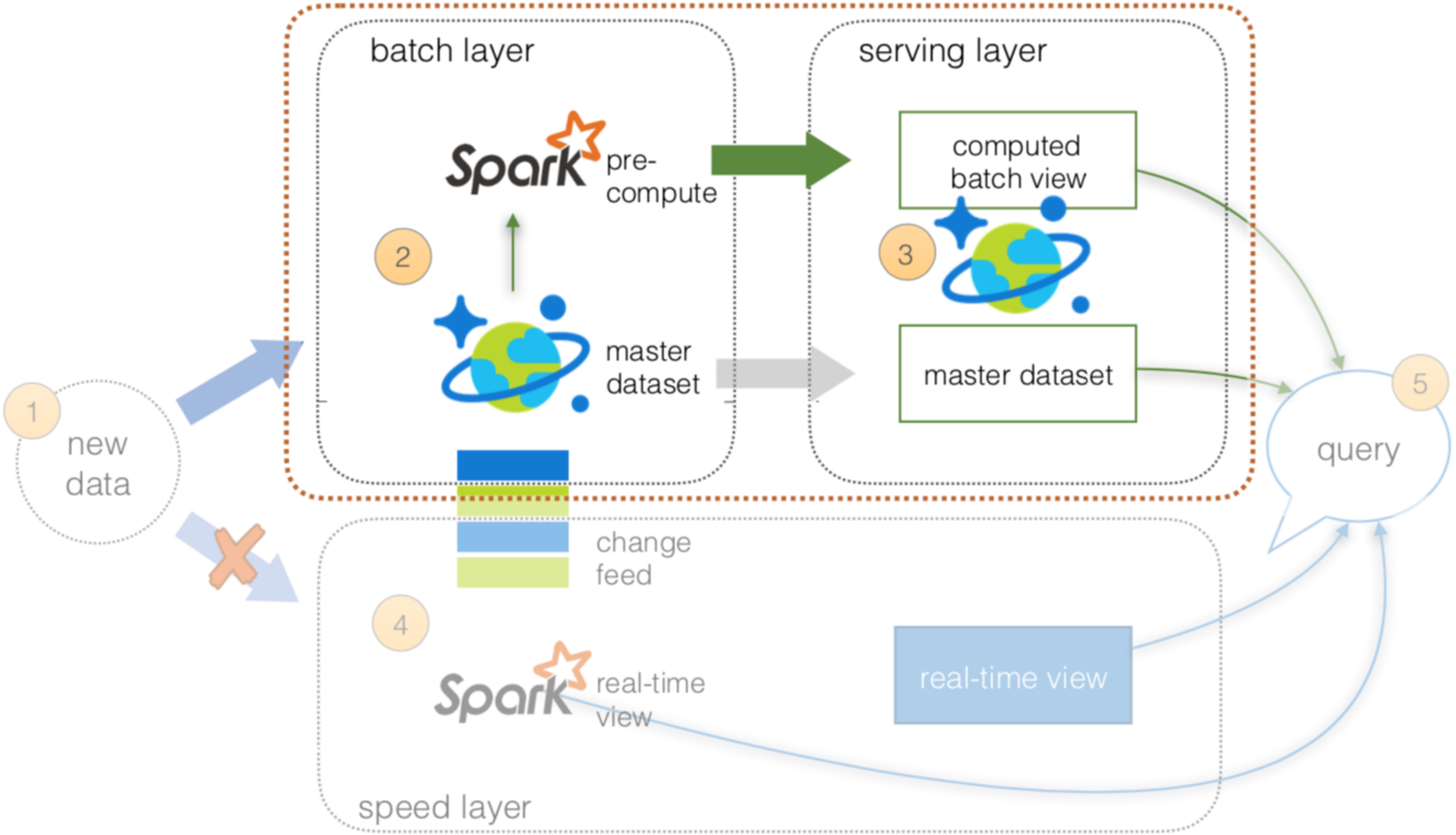


How Cosmos DB simplifies things

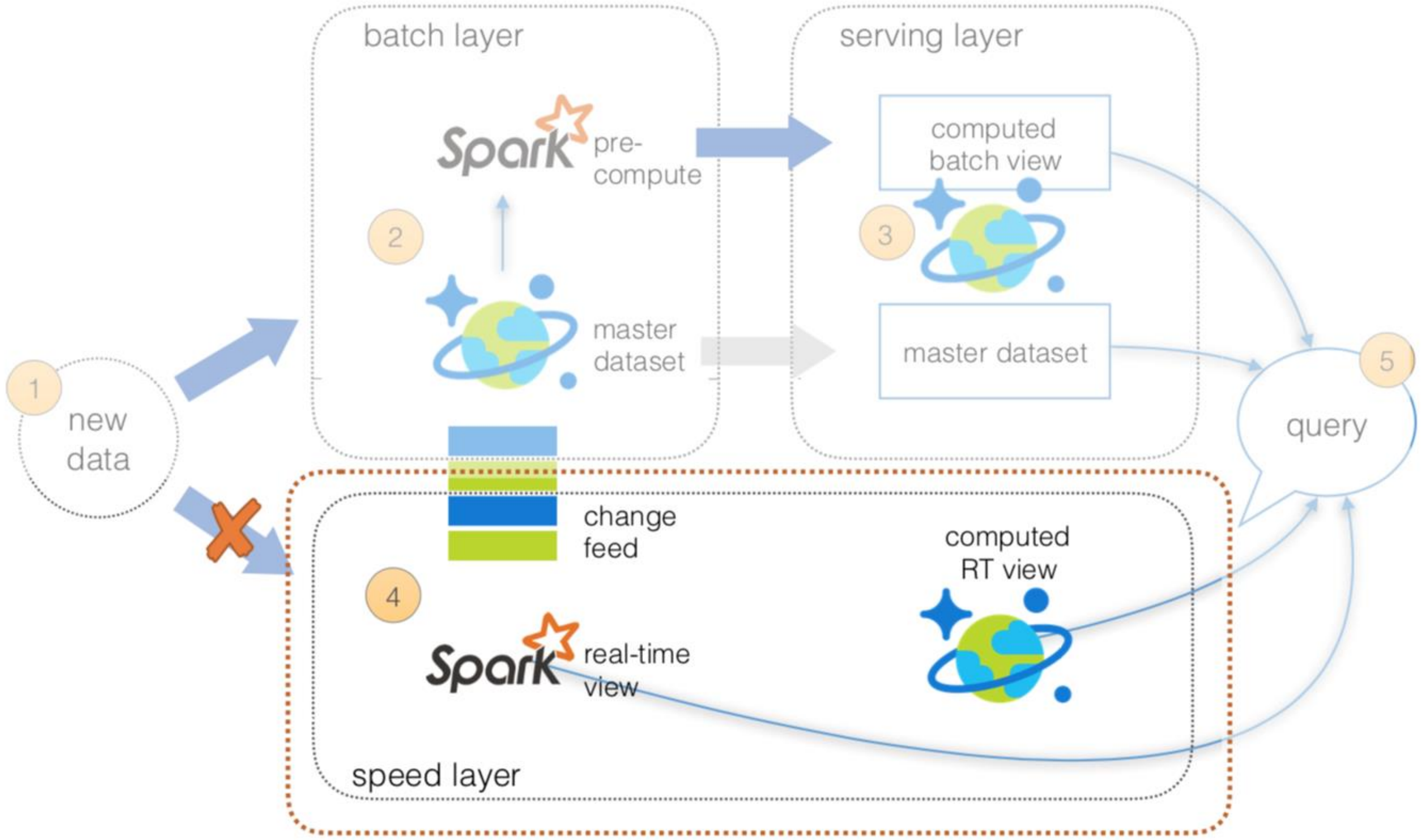
Simplify Lambda Architecture with Cosmos DB



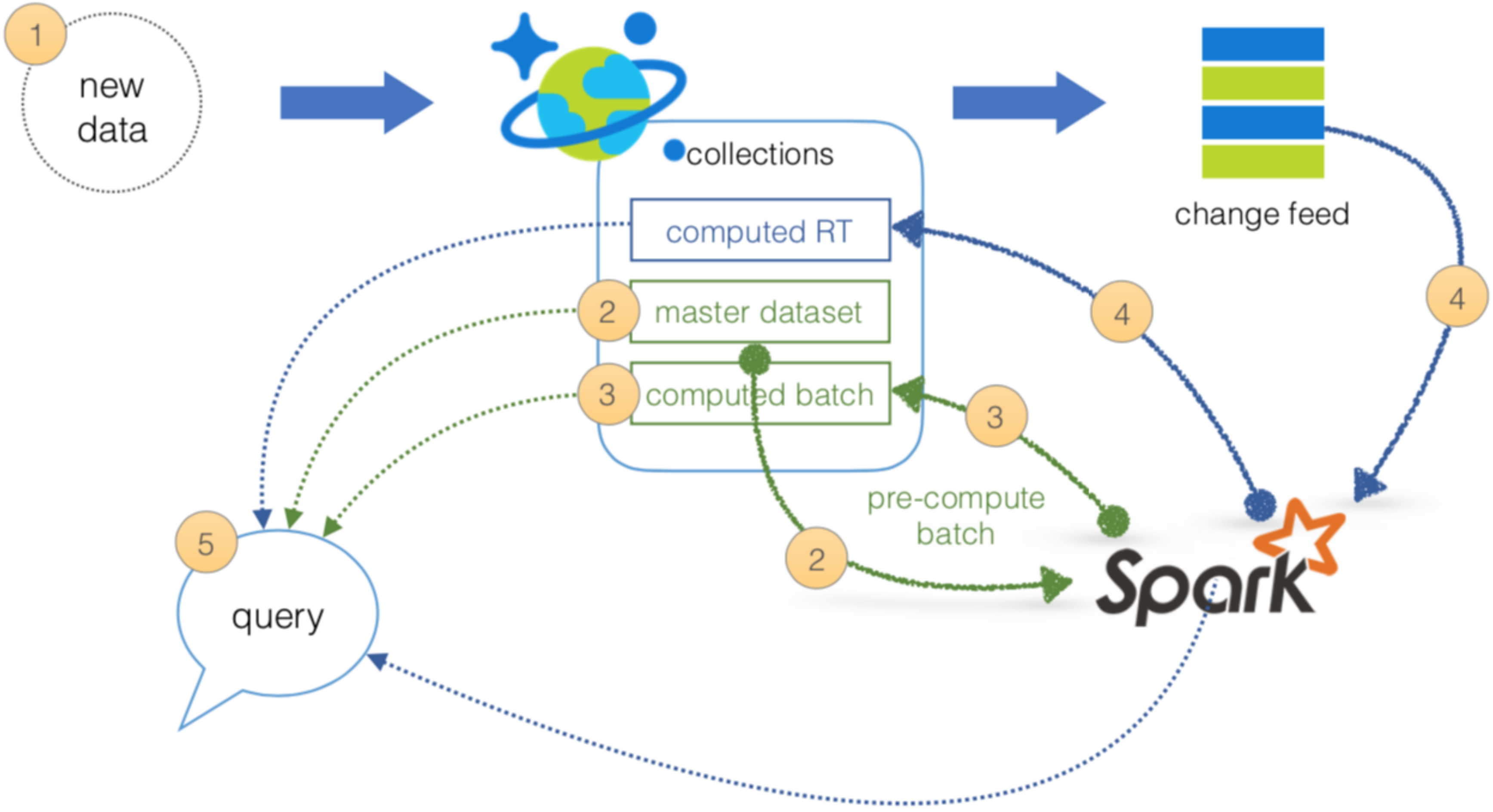
Lambda Architecture Cosmos DB – Batch and Serving Layers



Lambda Architecture Cosmos DB – Speed Layer

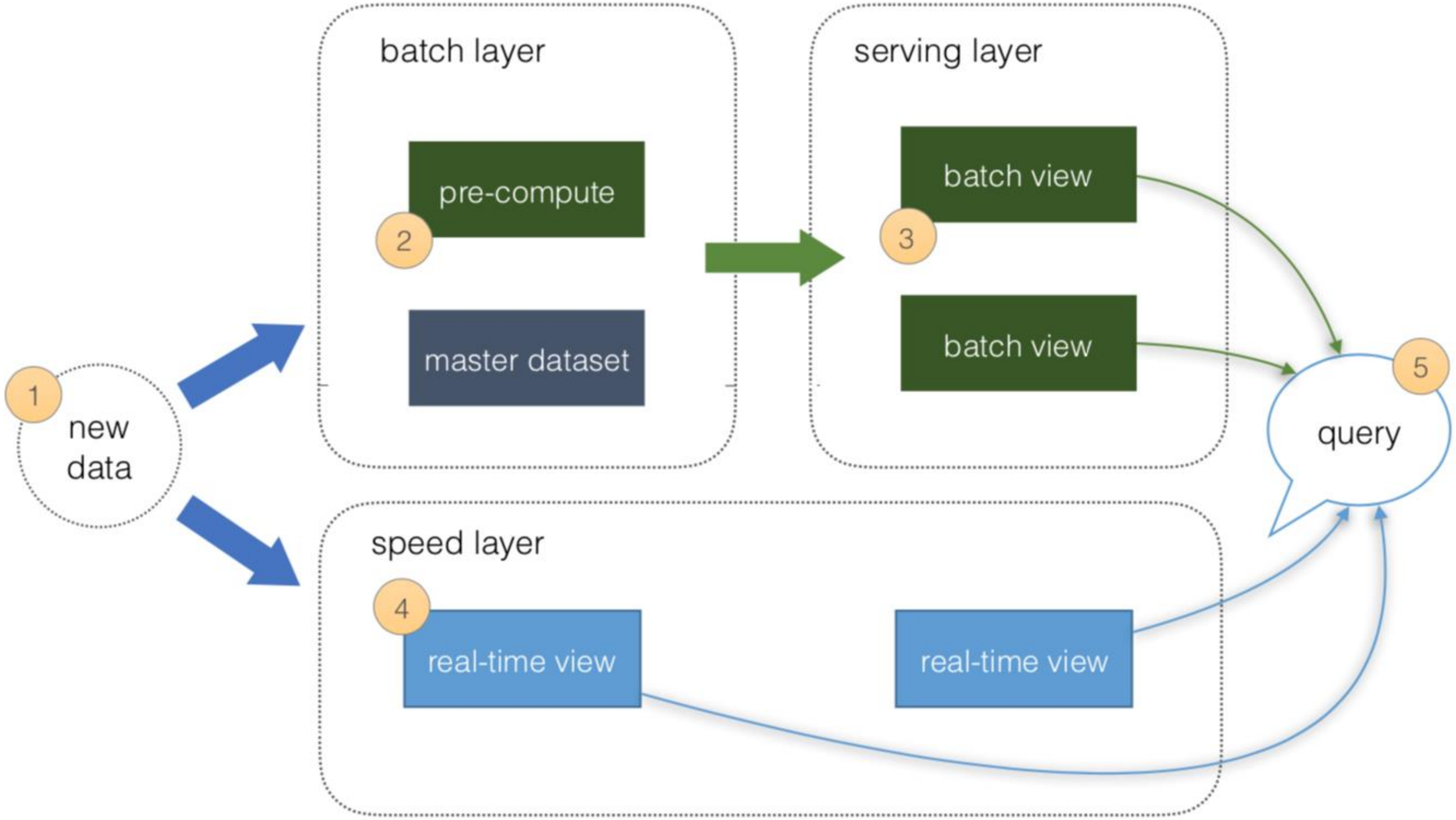


Lambda Architecture Cosmos DB - Re-architected

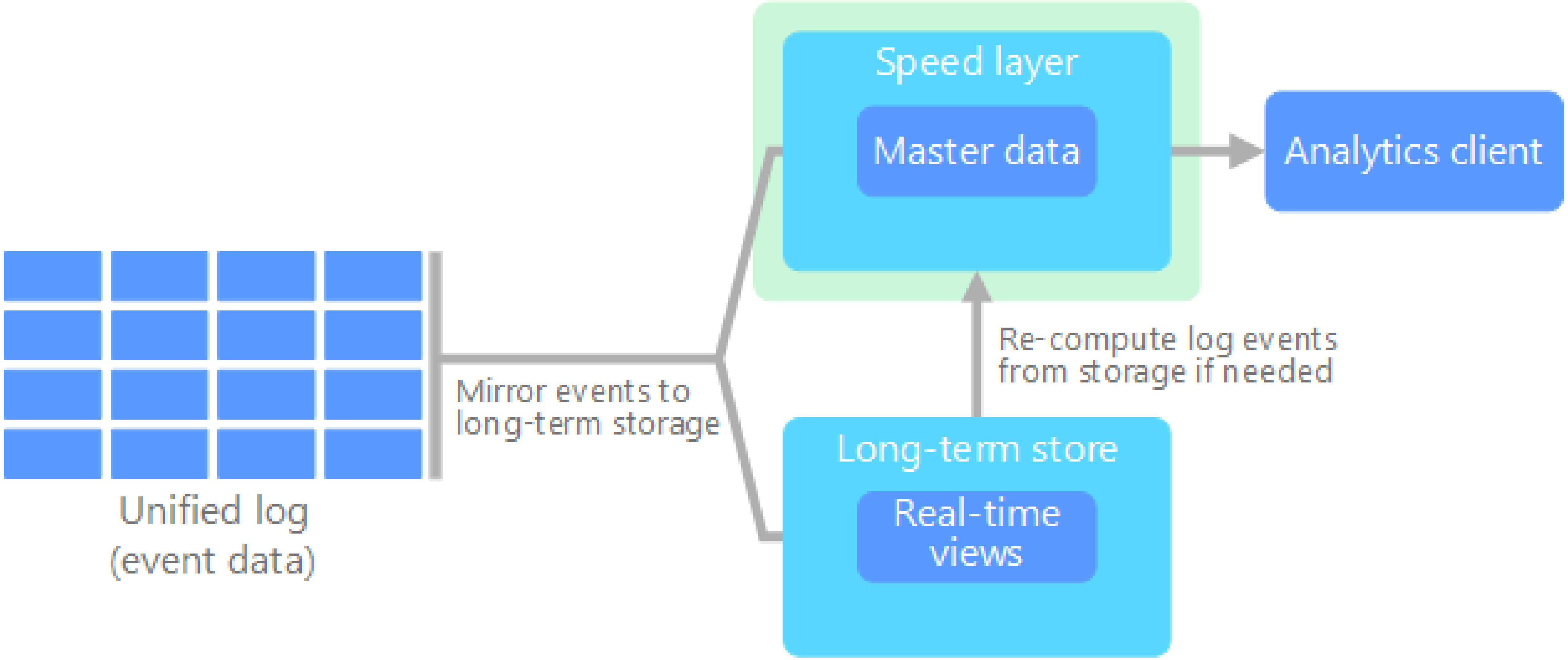


Lambda Architecture - Criticism

Lambda Architecture - downside



Kappa Architecture



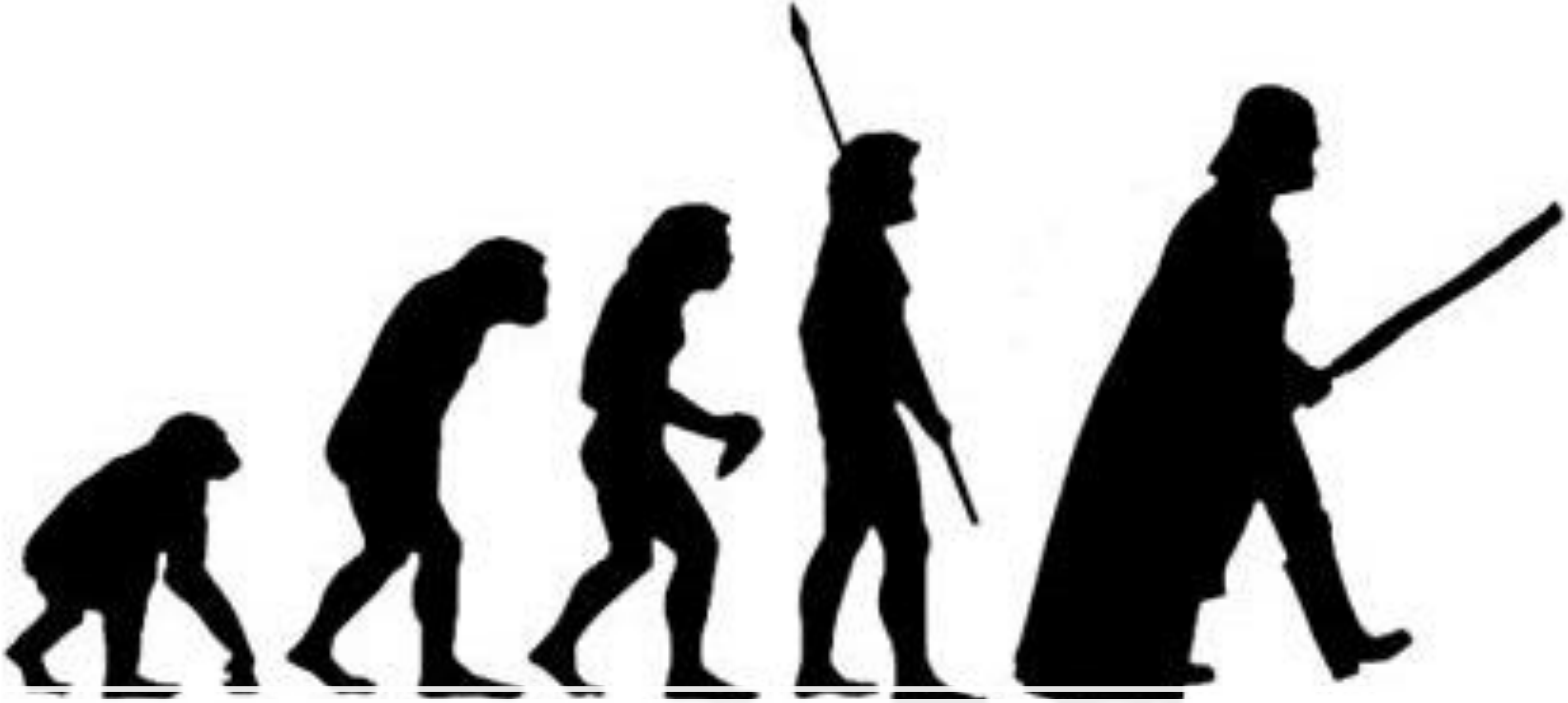
Pros and Cons

CONS

- Re-processing costly
- Speed layer always busy
- Less robust

PROS

- No batch layer, less complex
- Re-processing infrequent
- Runs on fixed memory
- Horizontally scalable
- Fewer resources



Materialized View

Event Sourcing

CQRS

Kappa/Lambda Architecture

????????

Evolution

Santosh Hari

Azure Consultant @ Nebbia Tech

Azure MVP

President, Orlando .NET UG

Organizer, Orlando Codecamp

santoshhari.wordpress.com

✉ santosh.hari@newsignature.com

@_s_hari

in /in/santoshhari