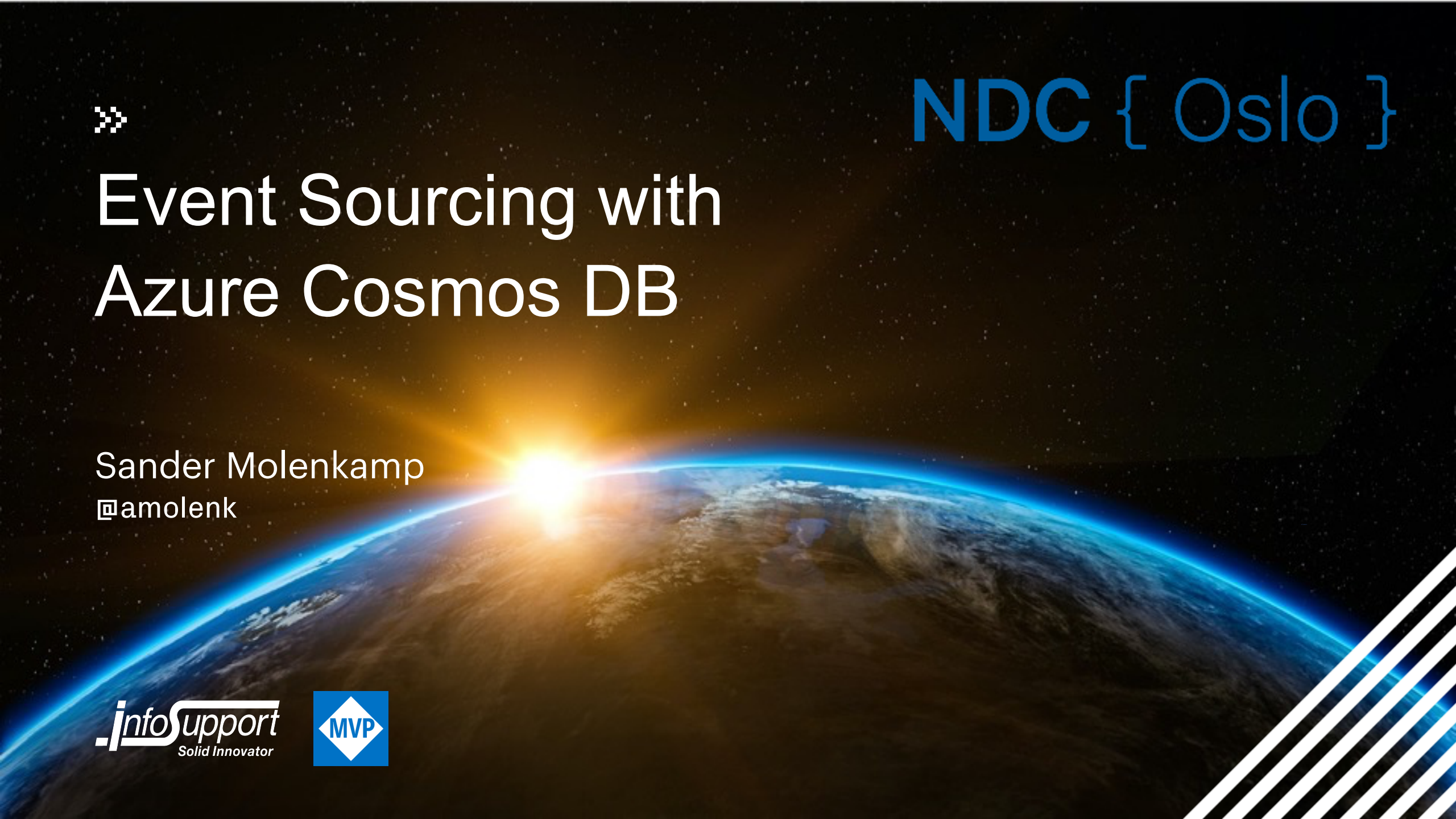
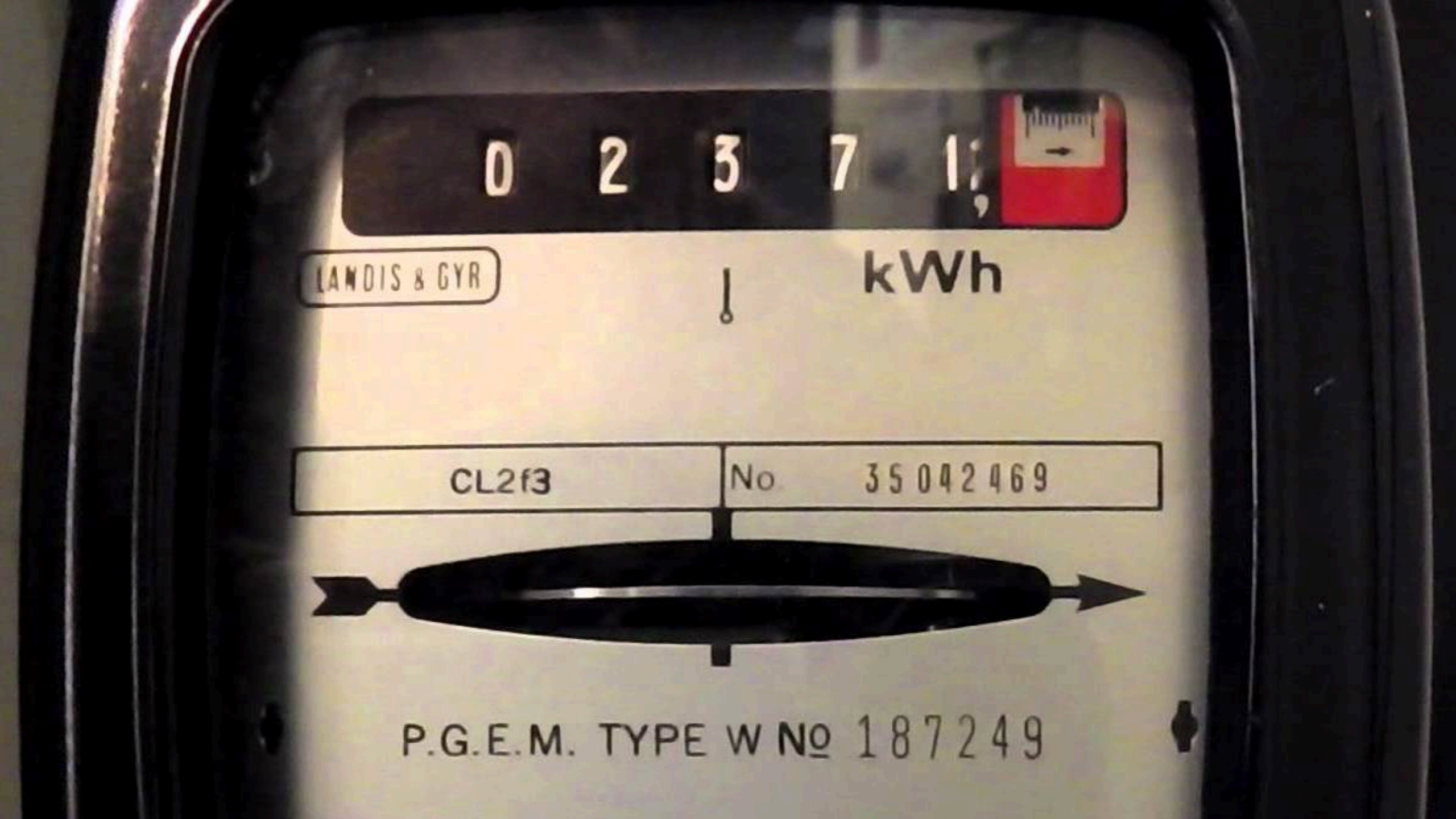
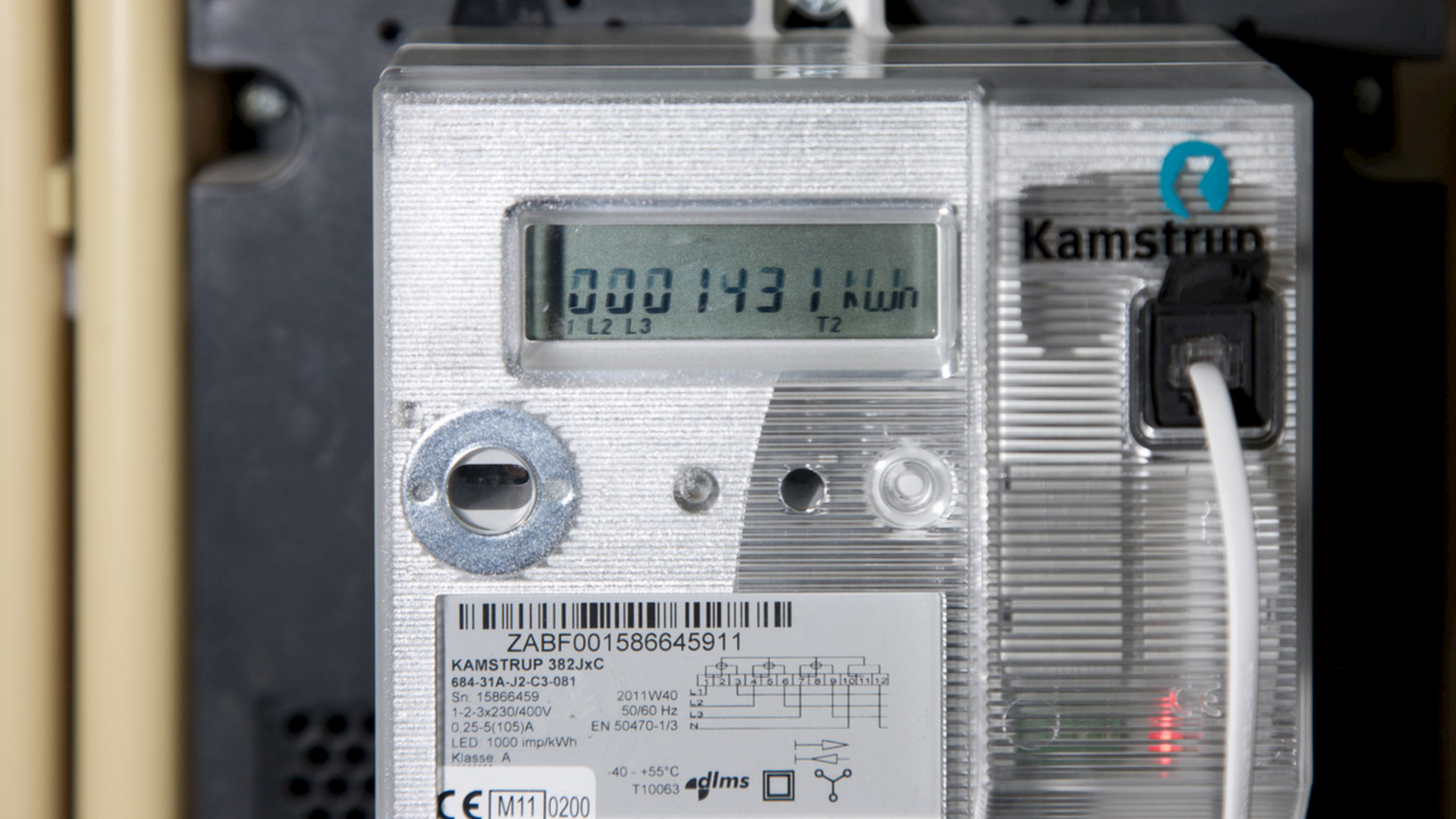
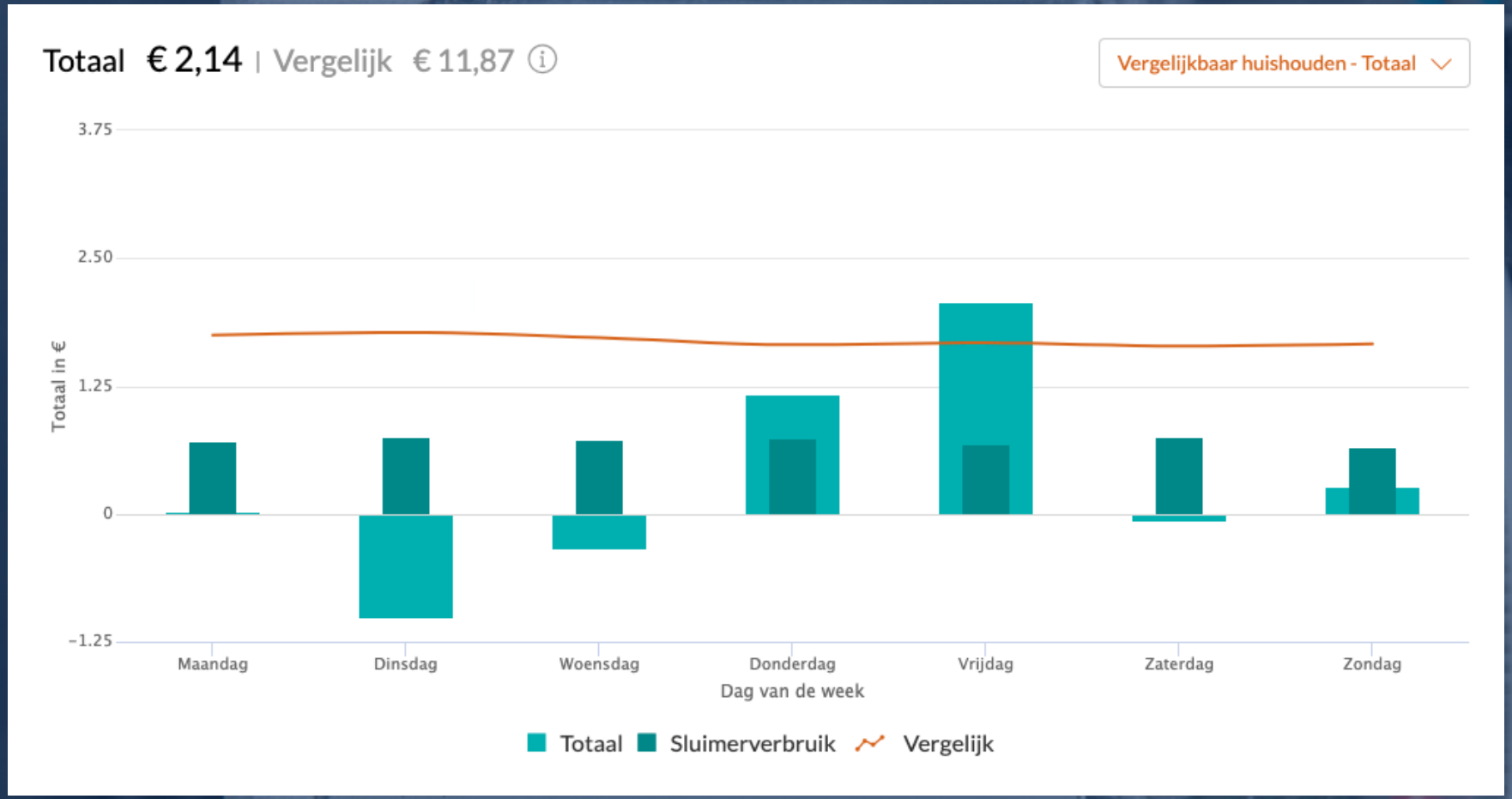

NDC { Oslo }
Event Sourcing with Azure Cosmos DB
Sander Molenkamp
@amolenk
infoSupport
Solid Innovator
MVP

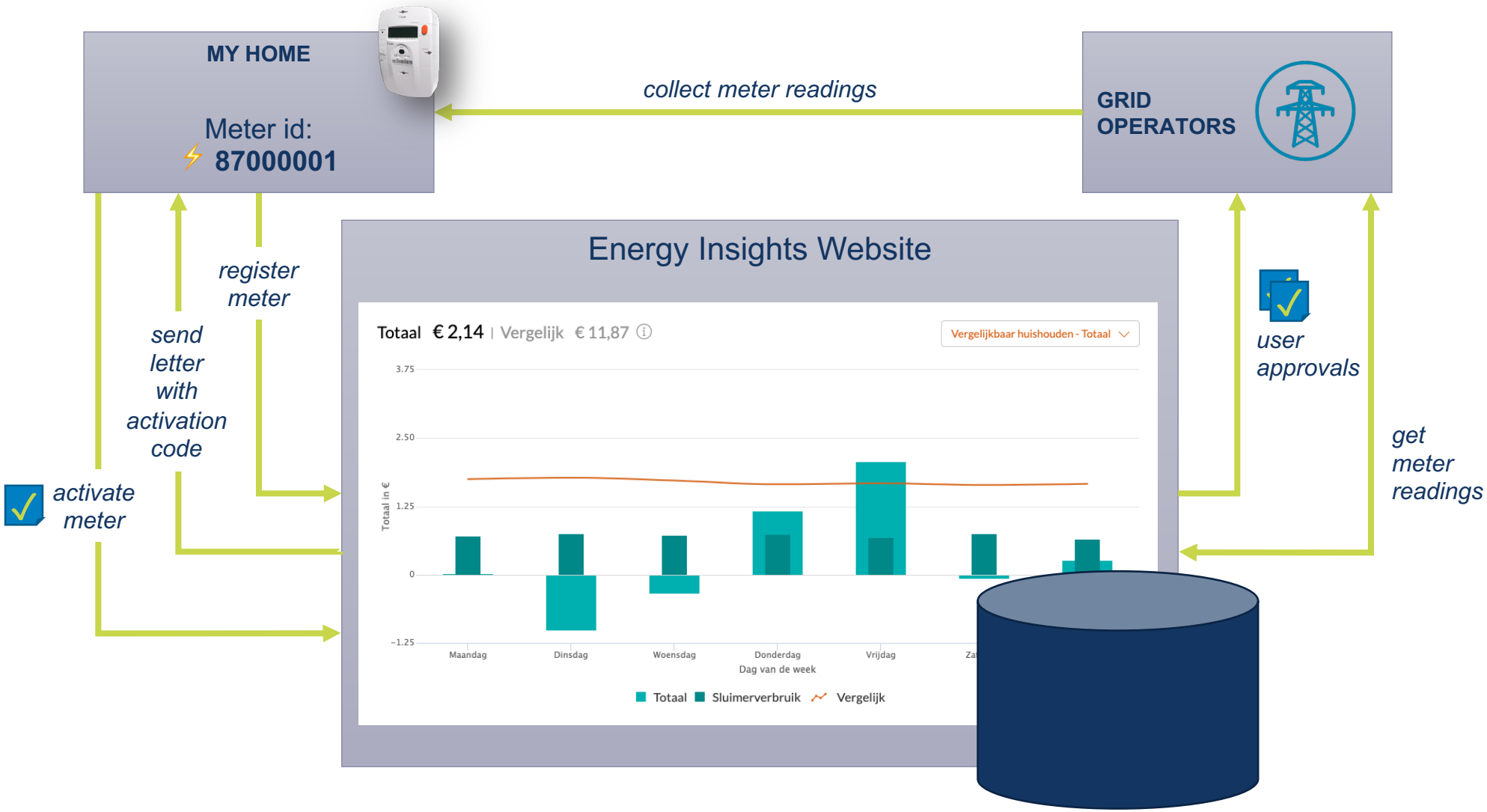

0 2 3 7 1
LANDIS & GYR
kWh
CL2f3
No 35042469
P.G.E.M. TYPE W № 187249


Kamstrup
000143 kWh
L1 L2 L3
T2
ZABF001586645911
KAMSTRUP 382JxC
684-31A-J2-C3-081
Sn: 15866459
1-2-3x230/400V
0.25-5(105)A
LED: 1000 imp/kWh
Klasse A
2011W40
50/60 Hz
EN 50470-1/3
-40 - +55°C
T10063
dlms
M11 0200

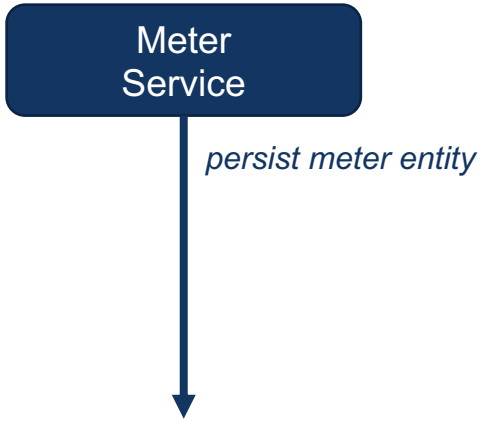
Energy insights



Energy insights



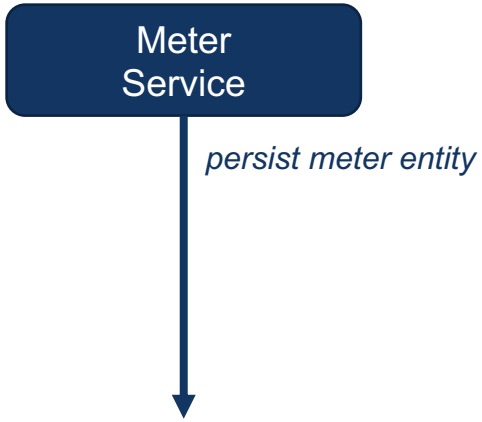
Saving current state



METER table

METER ID	STATUS	…

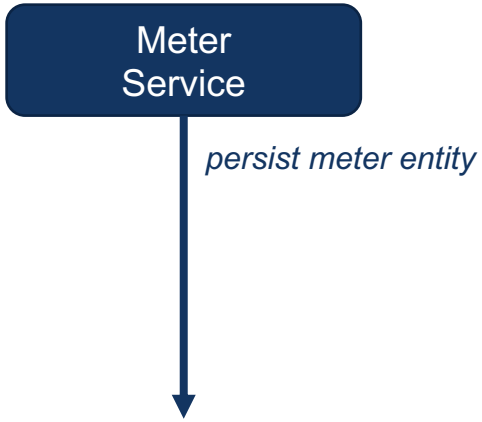
Saving current state



METER table

METER ID	STATUS	…
87000001	Registered	…

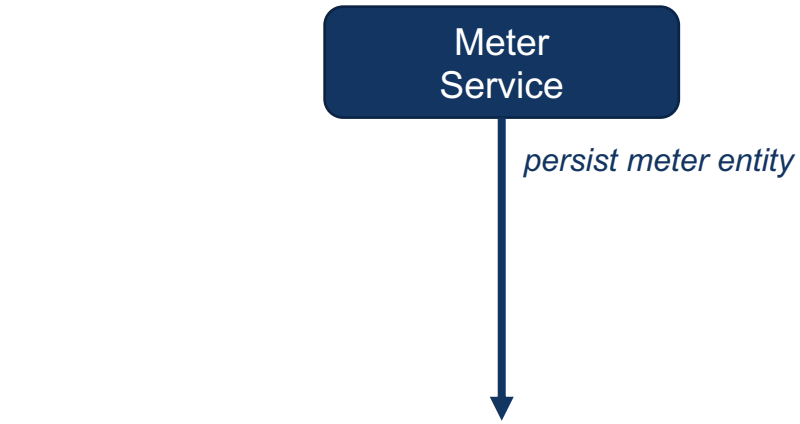
Saving current state



METER table

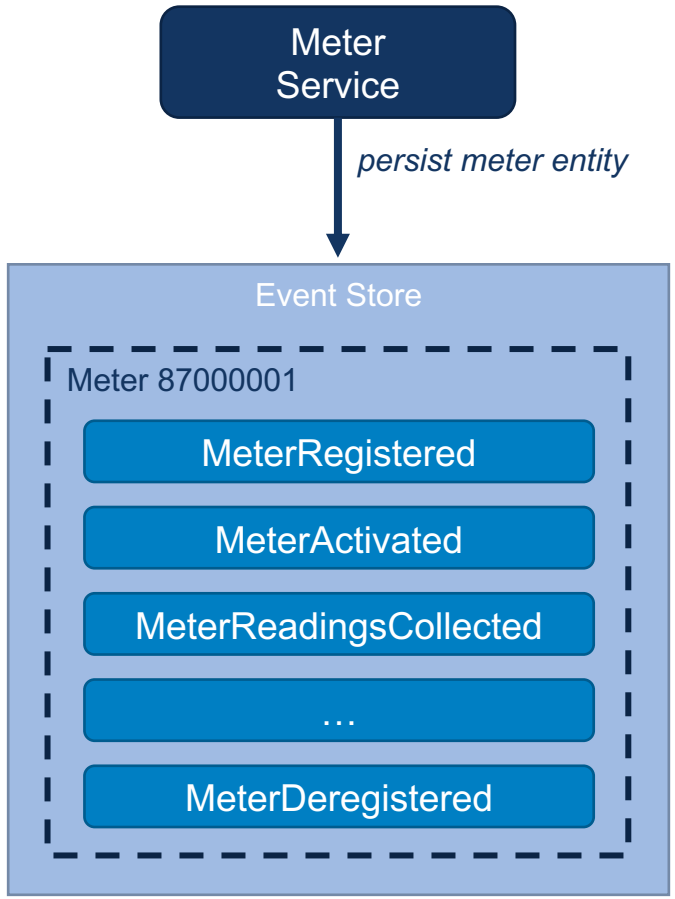
METER ID	STATUS	…
87000001	Activated	…

Storing state as a sequence of events



METER table

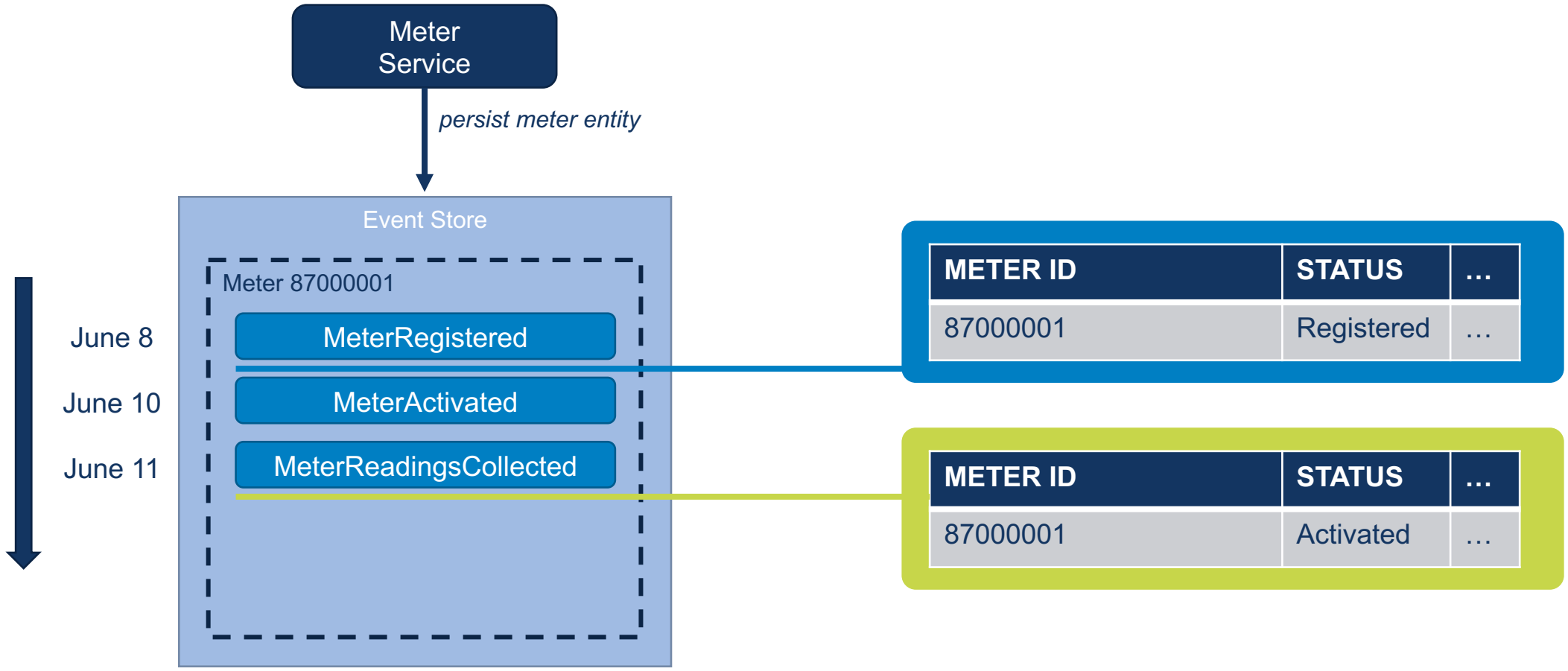
METER ID	STATUS	…
87000001	Activated	…



MeterRegistered event


```
{
    "MeterId": "87000001",
    "PostalCode": "1000 AA",
    "HouseNumber": "25",
    "ActivationCode": "542-484",
    "Timestamp": "2020-04-17T05:37:57.775153Z"
}
```

Replay events to get current state



Event sourcing

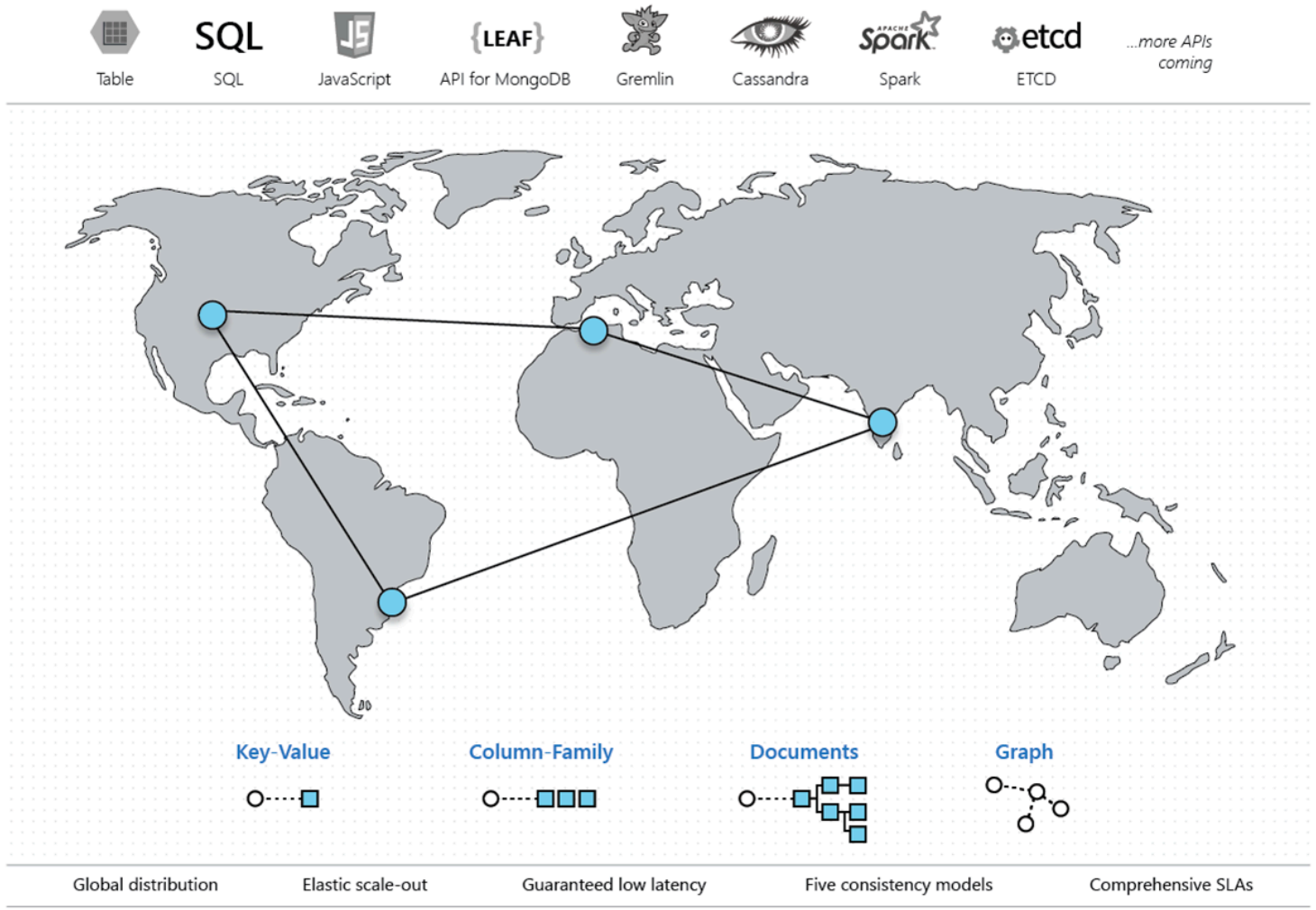
- Immutable stream of events
  - Just like an accountant you never change past events
- 100% accurate audit logging
- Easy temporal queries
  - Reconstructing historical state is easy by replaying a part of a stream

Let’s build an Event Store!

Building your own event store is a hands-on way to increase your understanding of the core concepts.

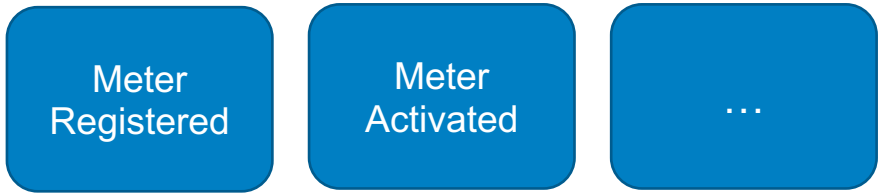
```
public interface IEventStore
{
  Task<EventStream> LoadStreamAsync(string streamId);

  Task AppendToStreamAsync(
    string streamId,
    int expectedVersion,
    IEnumerable<IEvent> events);
}
```

Azure Cosmos DB



Event sourcing using Cosmos DB

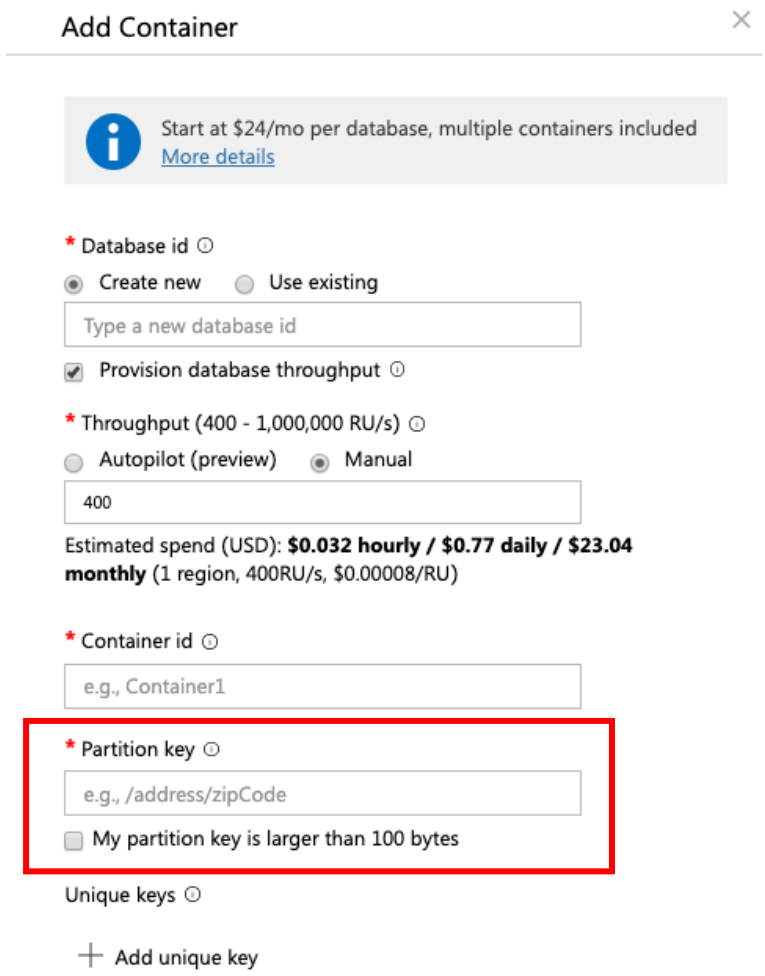
- We will use the SQL API
  - Each JSON item in the database represents a single event



  - C# / SDK v3

- Each JSON item contains additional metadata
  - Stream ID
  - Stream version
  - Event type/name

Choosing a partition key



AppendToStream(streamId, expectedVersion, events)

TRANSACTION SCOPE

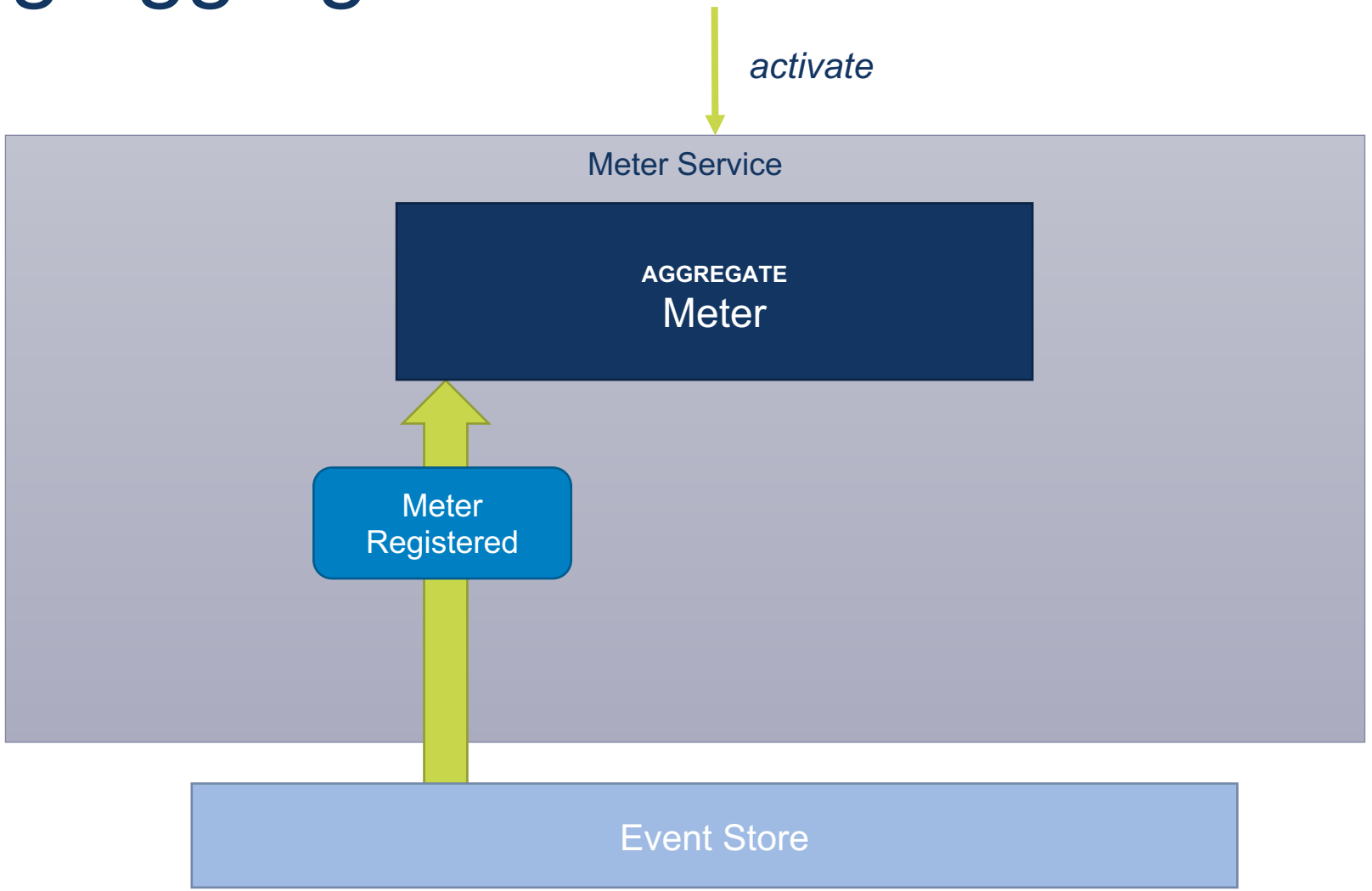
1. Get current stream version
2. Verify that version matched expected version
3. If so, write events to store

- Use Stored Procedure for transaction

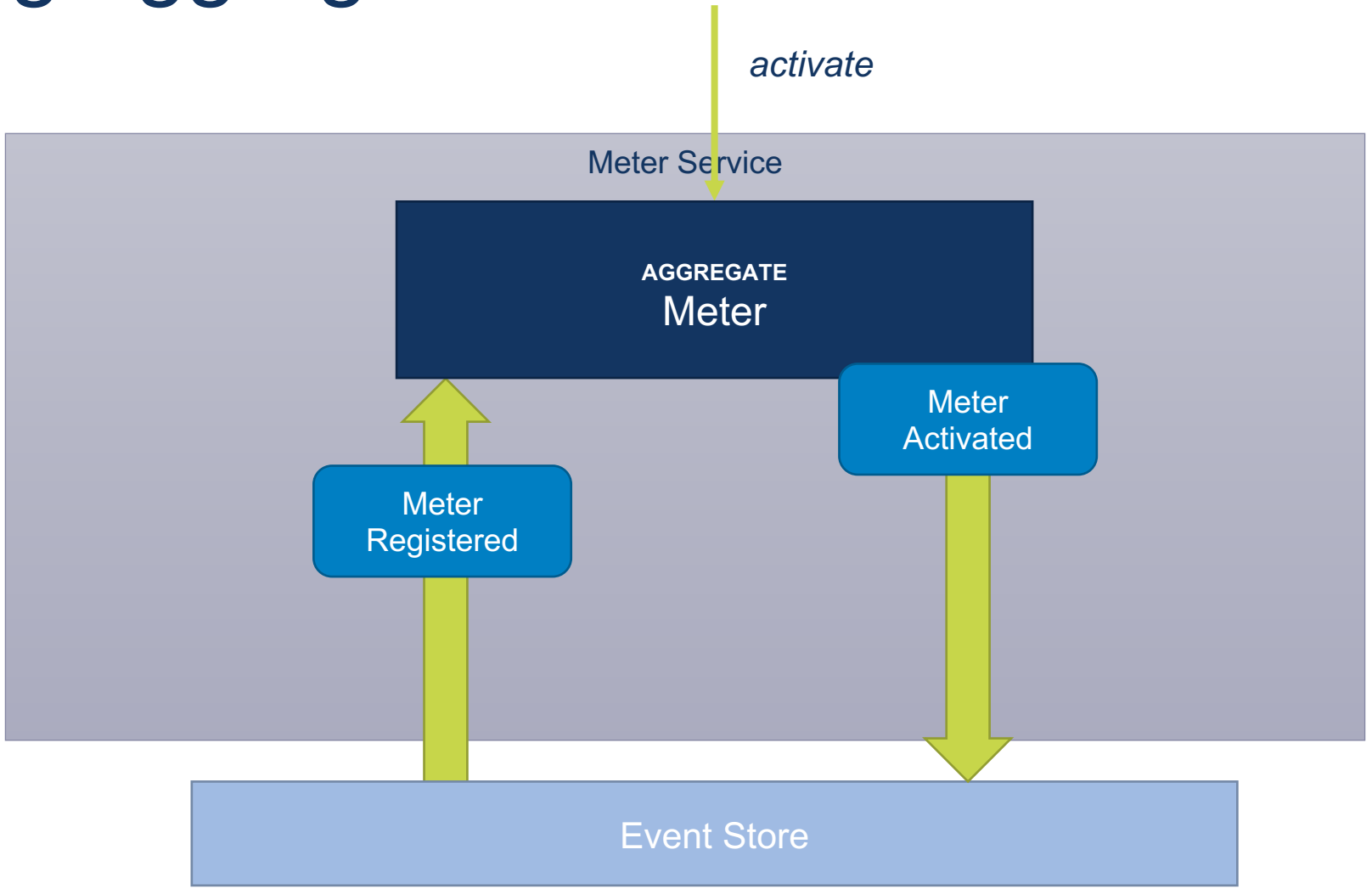
Writing & reading event streams

COSMOS DB - DEMO

Using aggregates



Using aggregates



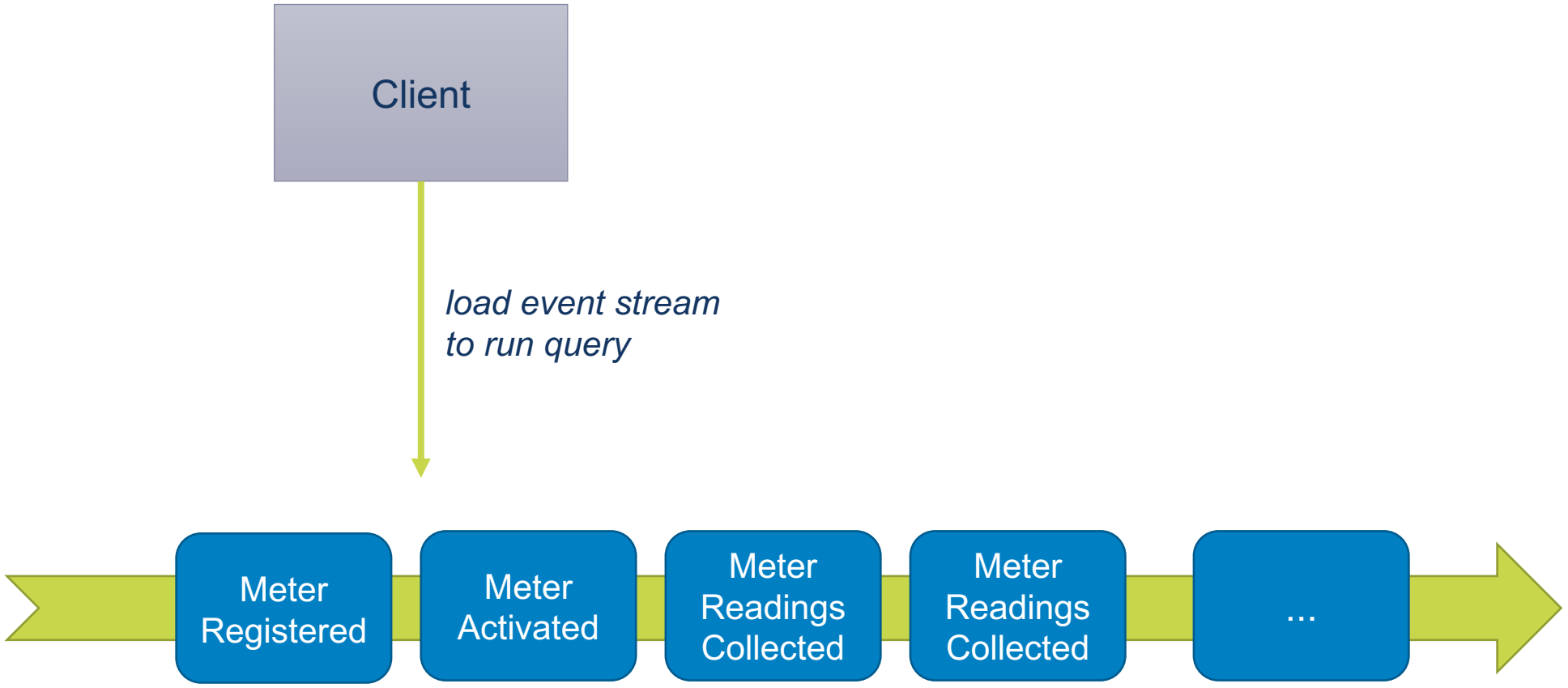
Adding business logic

COSMOS DB - DEMO

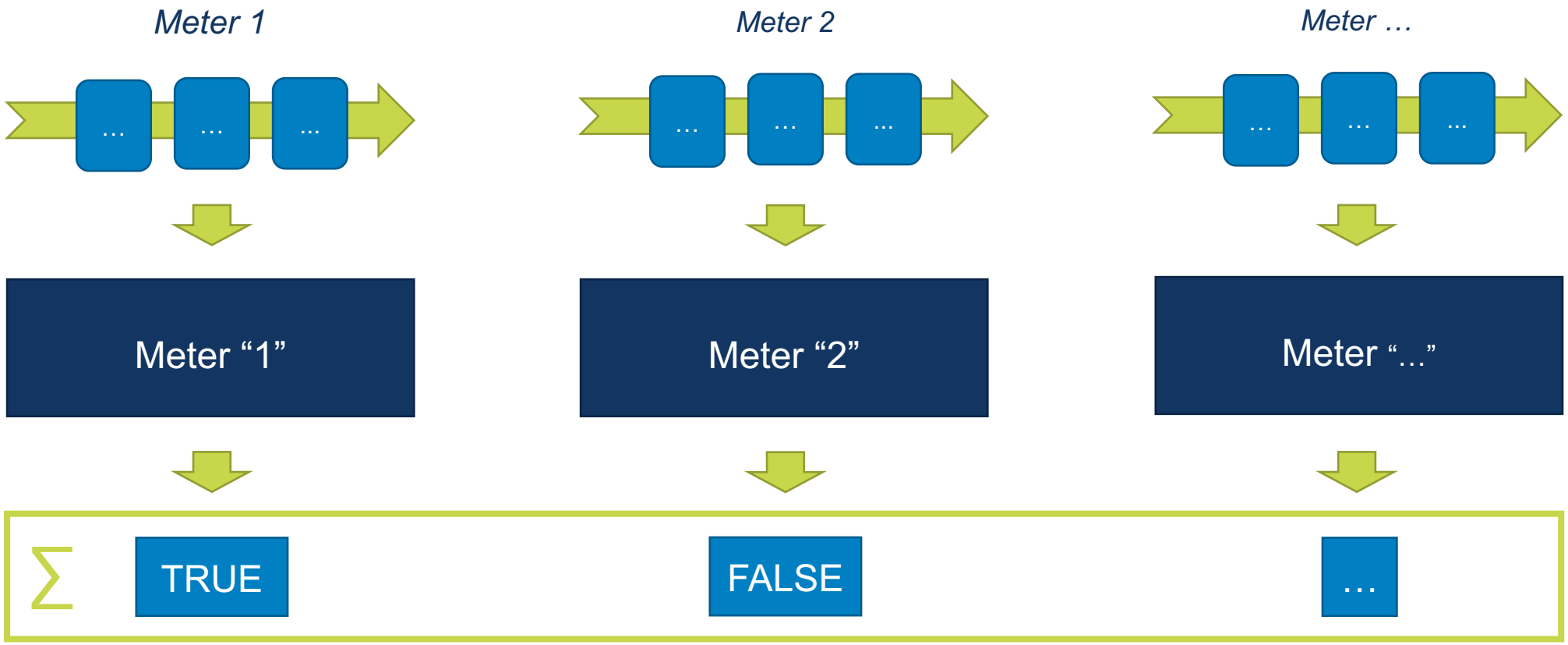
Querying event streams

COSMOS DB

Querying event streams

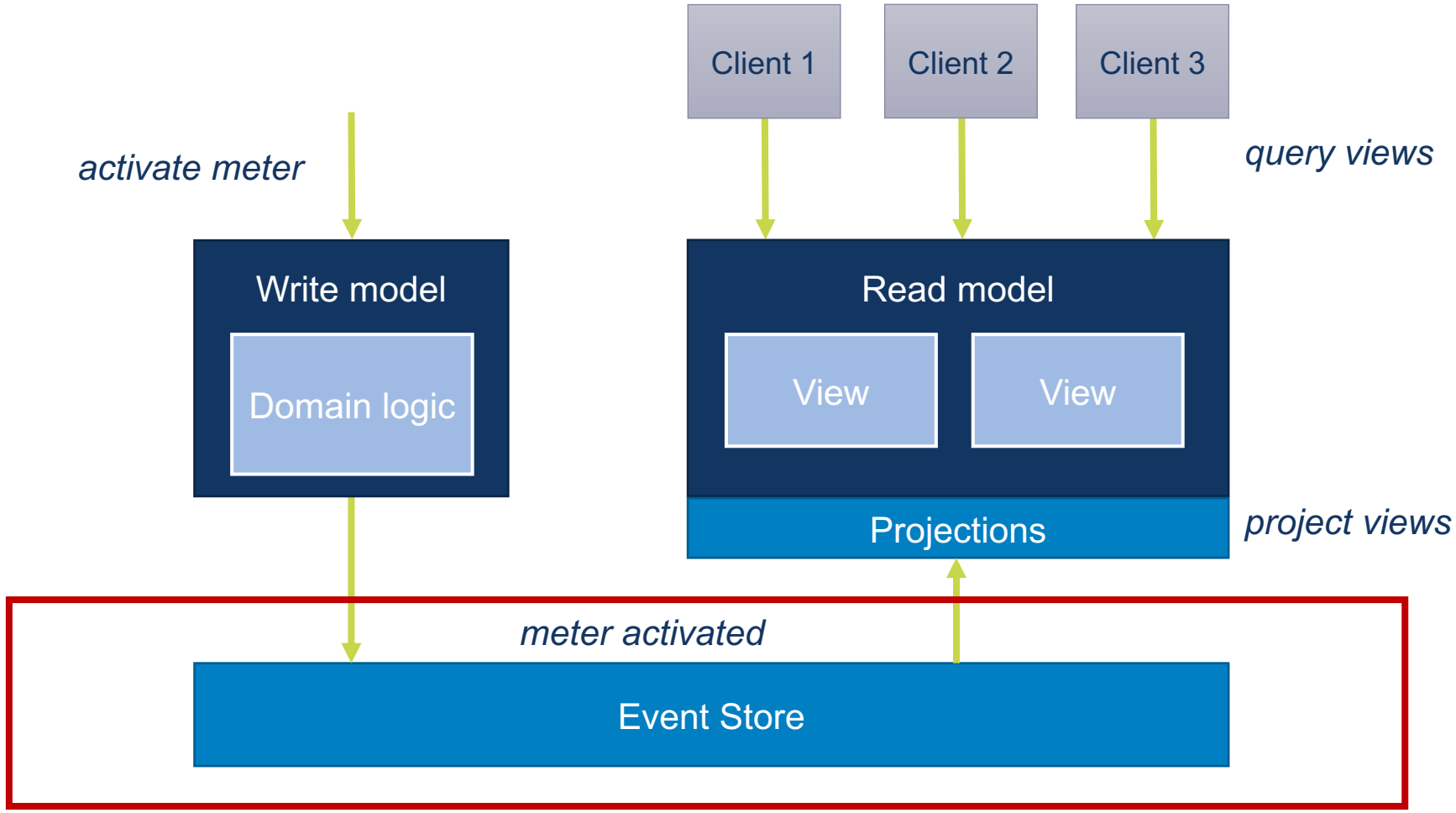


Multi-stream queries



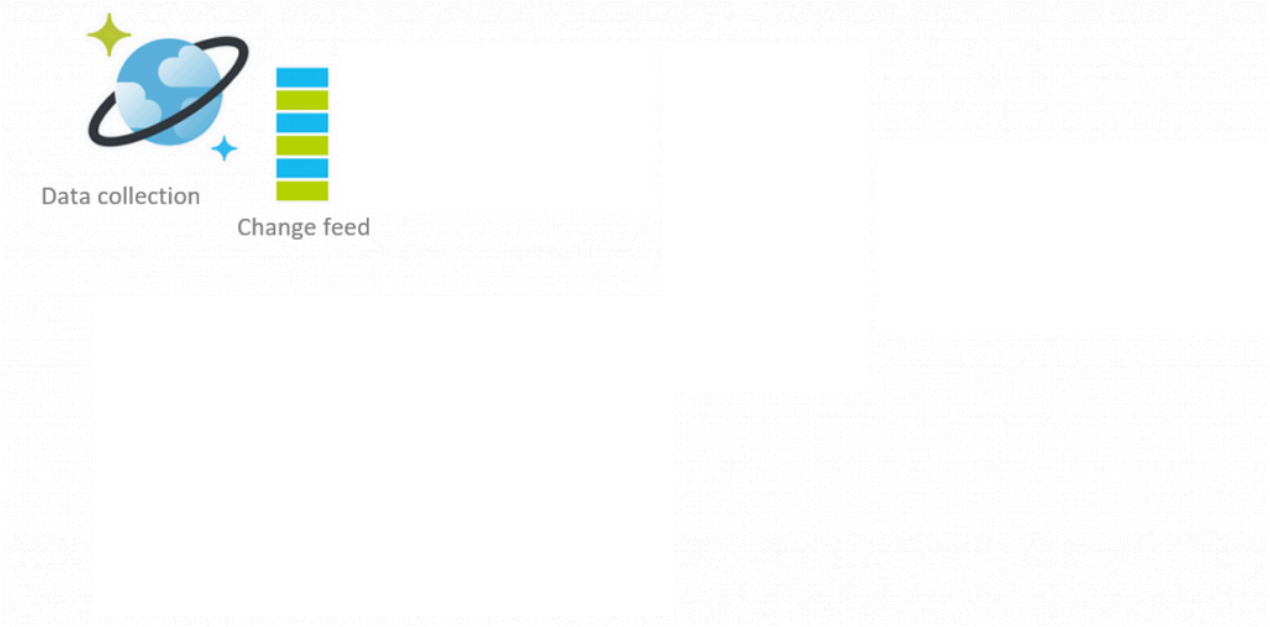
Command Query Responsibility Segregation

Command Query Responsibility Segregation

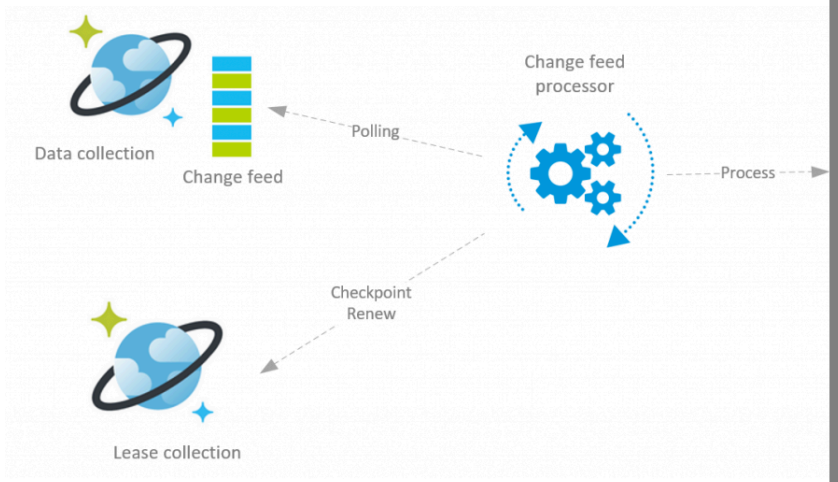


Cosmos DB change feed

- Subscribe to changes in a Cosmos DB container
- Change feed items come in the order of their modification time
- If the data is not deleted, it will remain in the change feed



Handle changes

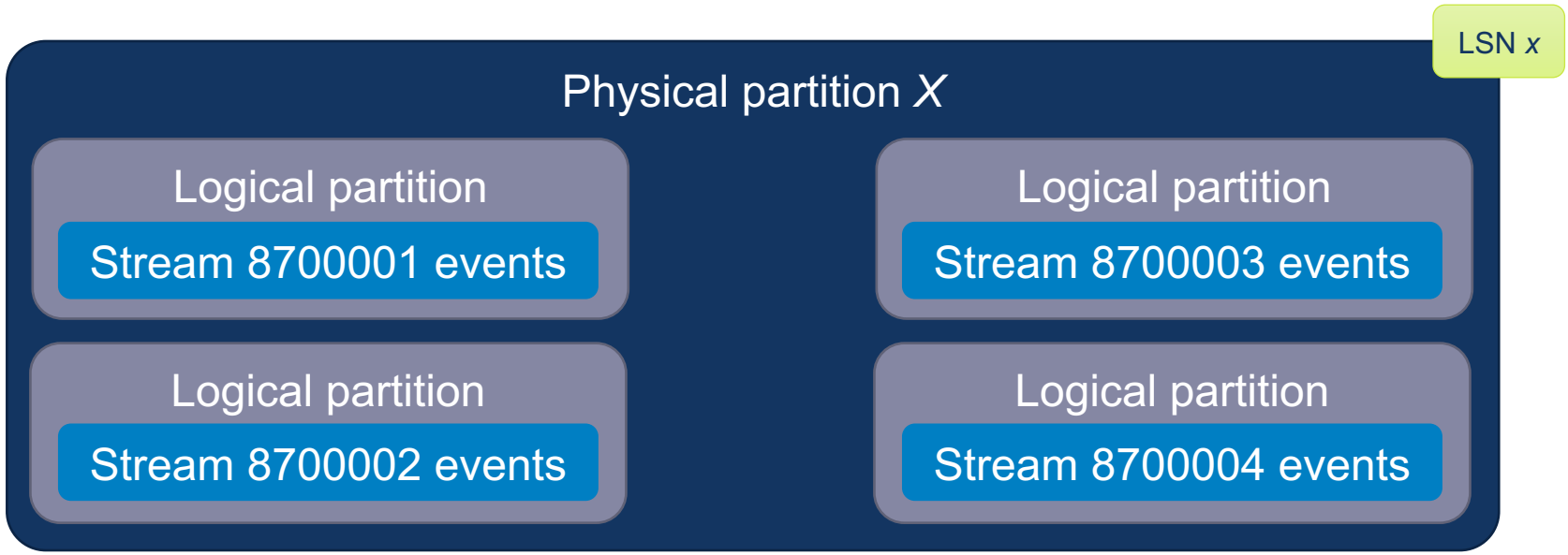


1. Deserialize events
2. For each event:
    1. Get subscribed projections
    2. For each subscribed projection:
        1. Load view
        2. Skip projection if view LSN is newer than event LSN
        3. Run projection
        4. Save view (use Etag for optimistic concurrency control)

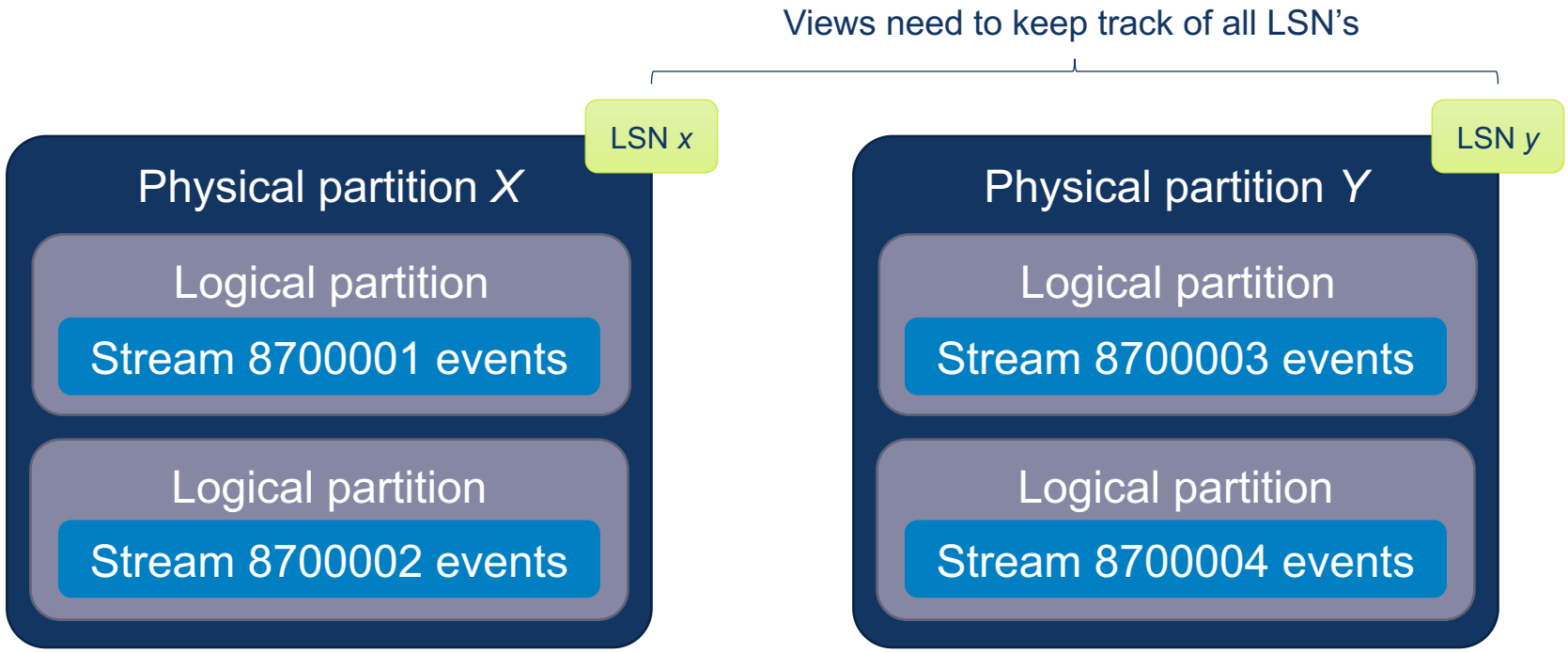
CQRS

COSMOS DB - DEMO

Cross-partition projection caveat



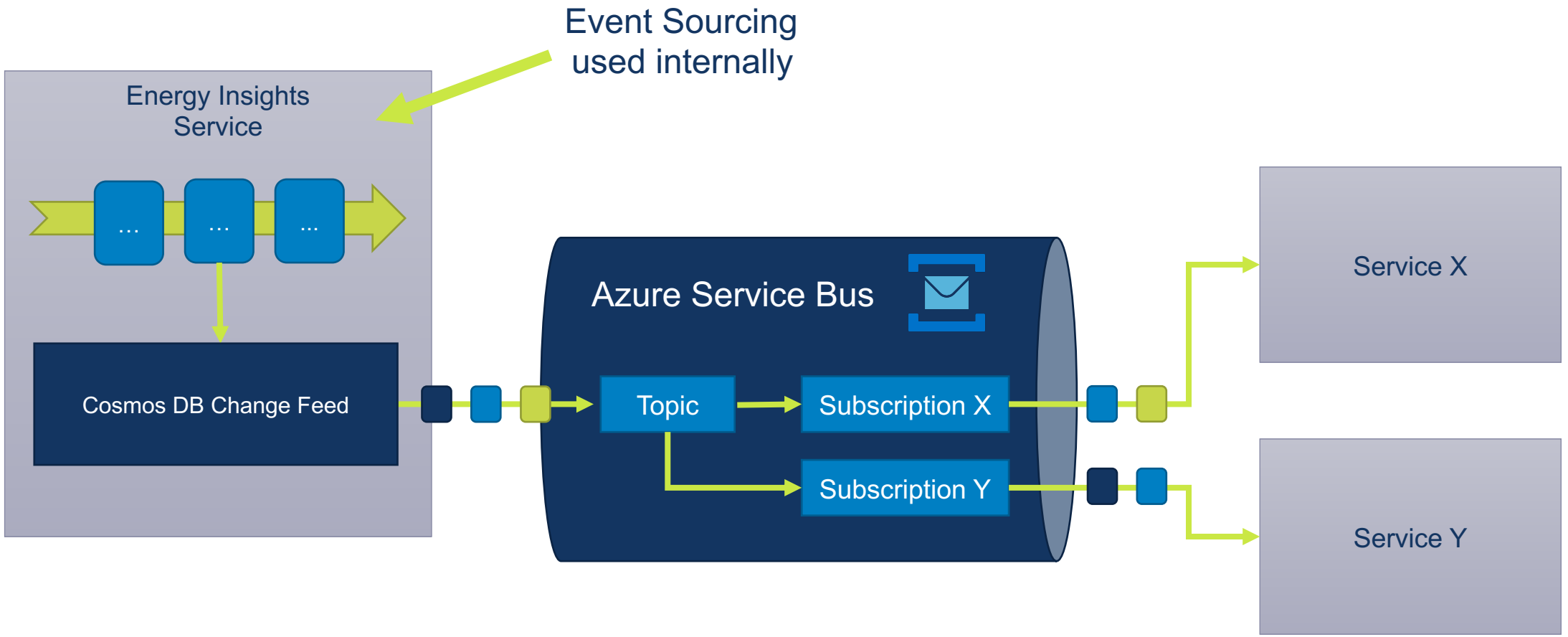
Cross-partition projection caveat



V3 change feed support doesn't expose that information, possible solutions include:

- Use the V2 Change Feed Processor Library which does expose the PartitionId
- Wait for future releases to reinstate Context: https://github.com/Azure/azure-cosmos-dotnet-v3/issues/1122

Integration with Azure Service Bus



Event Driven Architecture to connect services

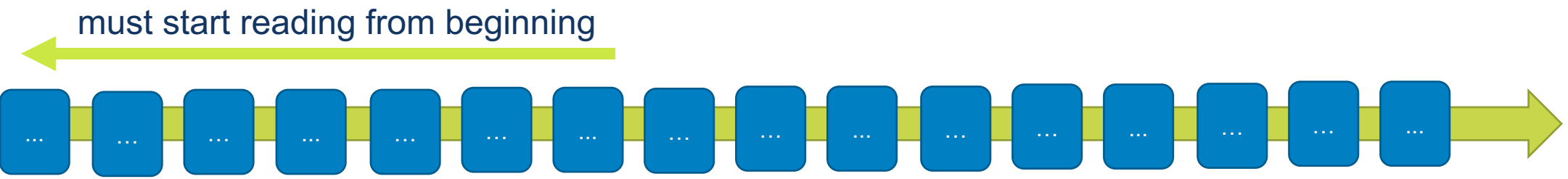
Write-side Performance

Performance

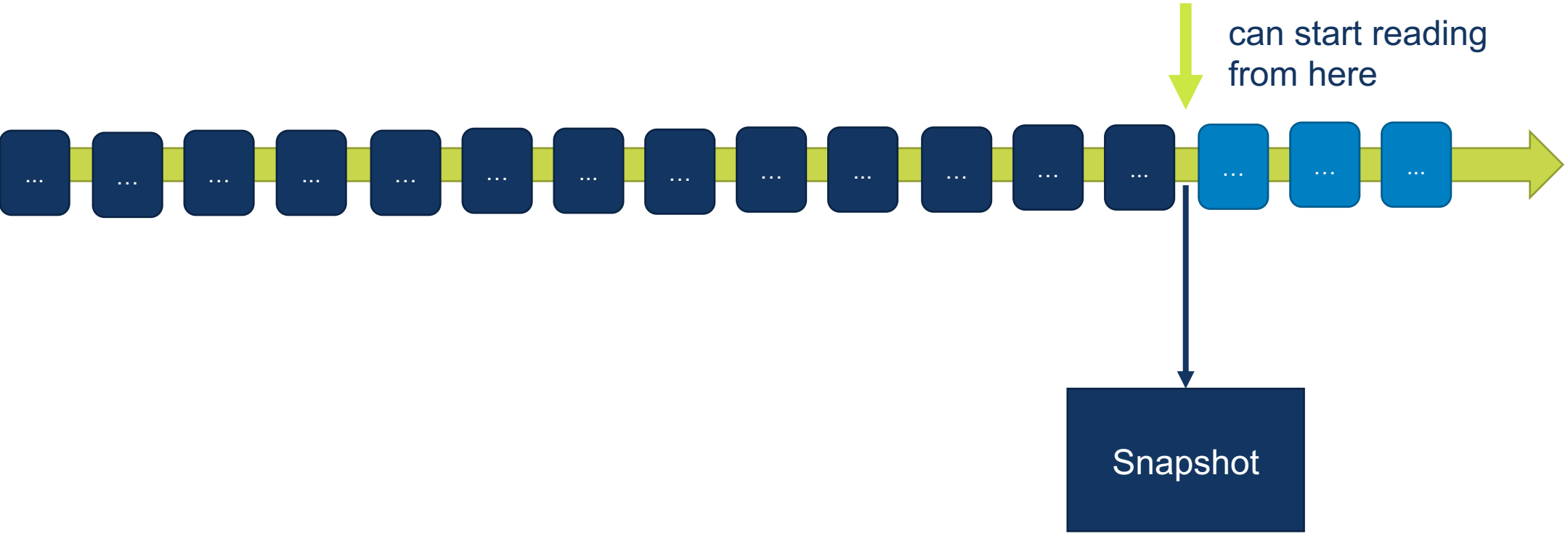
- Partition streams
  - This happens in the real world too, e.g. closing a financial year

- Use a cache
  - No complex cache invalidation because streams are immutable
- Use snapshots

Snapshots



Snapshots



GDPR

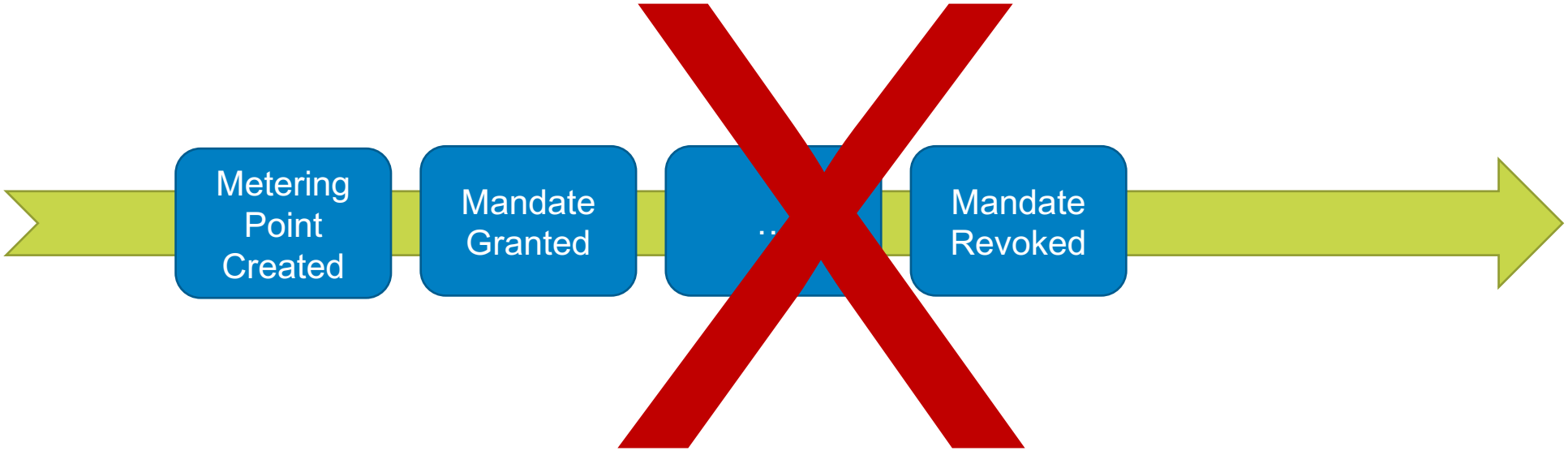
Immutability vs GDPR

“… the data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay…”

Article 17 – “right to erasure”

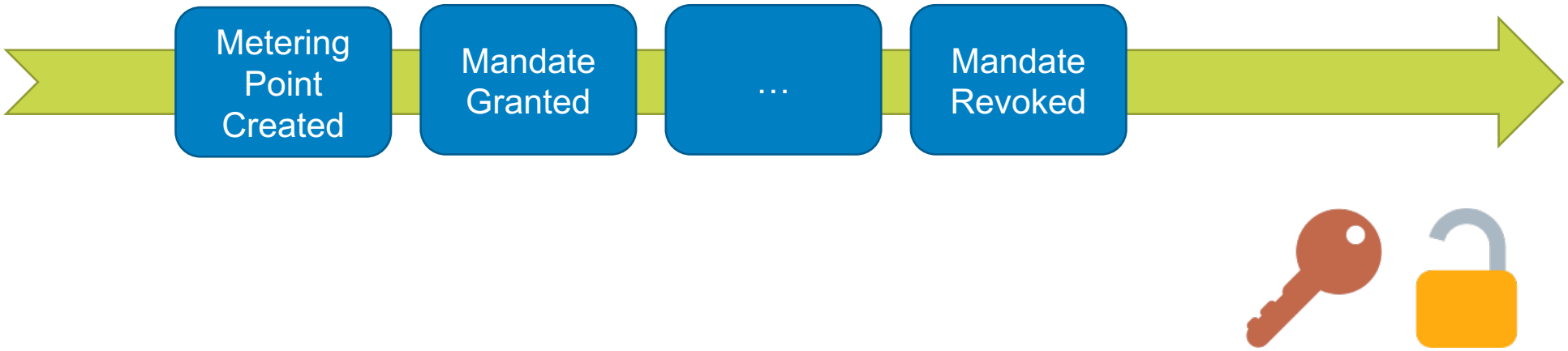
Delete entire stream

Removing an entire stream is a “safe” operation.



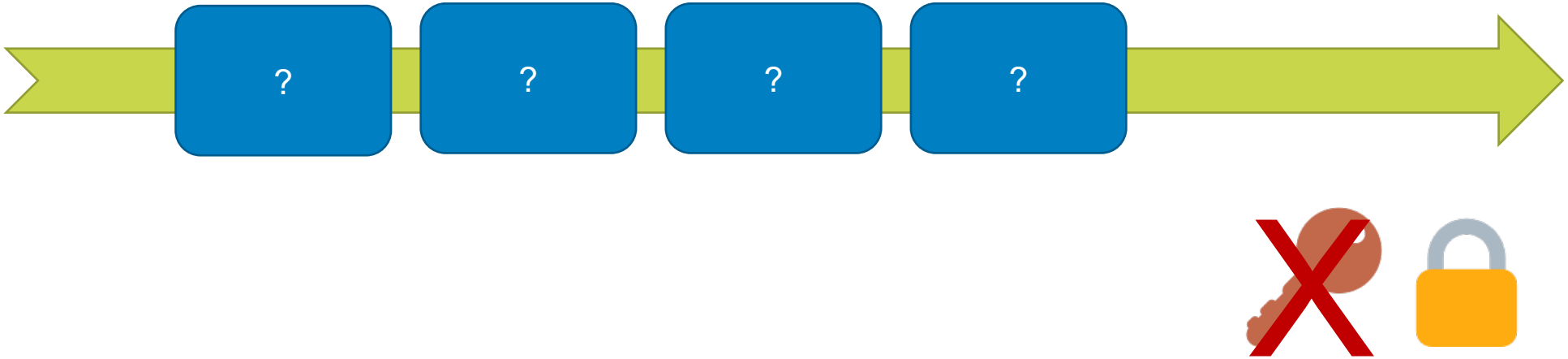
Crypto shedding

- Encrypt each stream using a unique key.



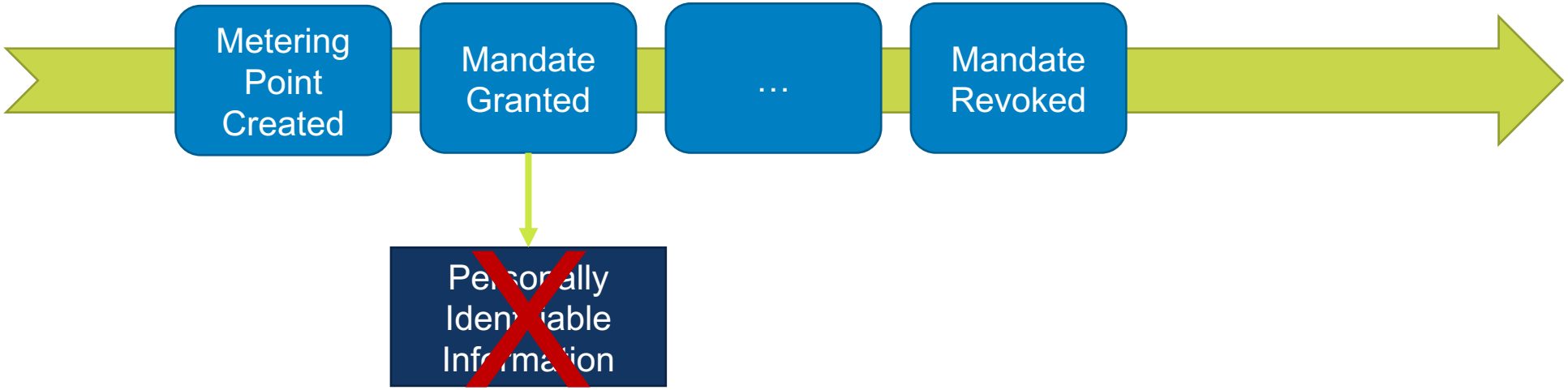
Crypto shedding

- Encrypt each stream using a unique key.
- Throw away the key to make the PII unreadable.



External storage

Store the PII outside of the event.



- All usernames/e-mail addresses/whatever must be unique!

All usernames/e-mail addresses/whatever must be unique!

- Don’t feel obligated to use ES everywhere in your application!
- Use the unique field as the event stream identifier
  - We use the unique meter id as the stream id
- Keep an index that you can reference during command processing
  - Only use on command side, do not mix with read model to avoid unnecessary dependencies
- What’s the real business scenario?
  - How often does this really happen? Can you just do a compensating action?

Thank you!

Sander Molenkamp

@amolenk

All demo source code available at:

https://github.com/amolenk/CosmosEventSourcing

infoSupport

Solid Innovator

dotnetFlix

www.youtube.com

/dotnetflix