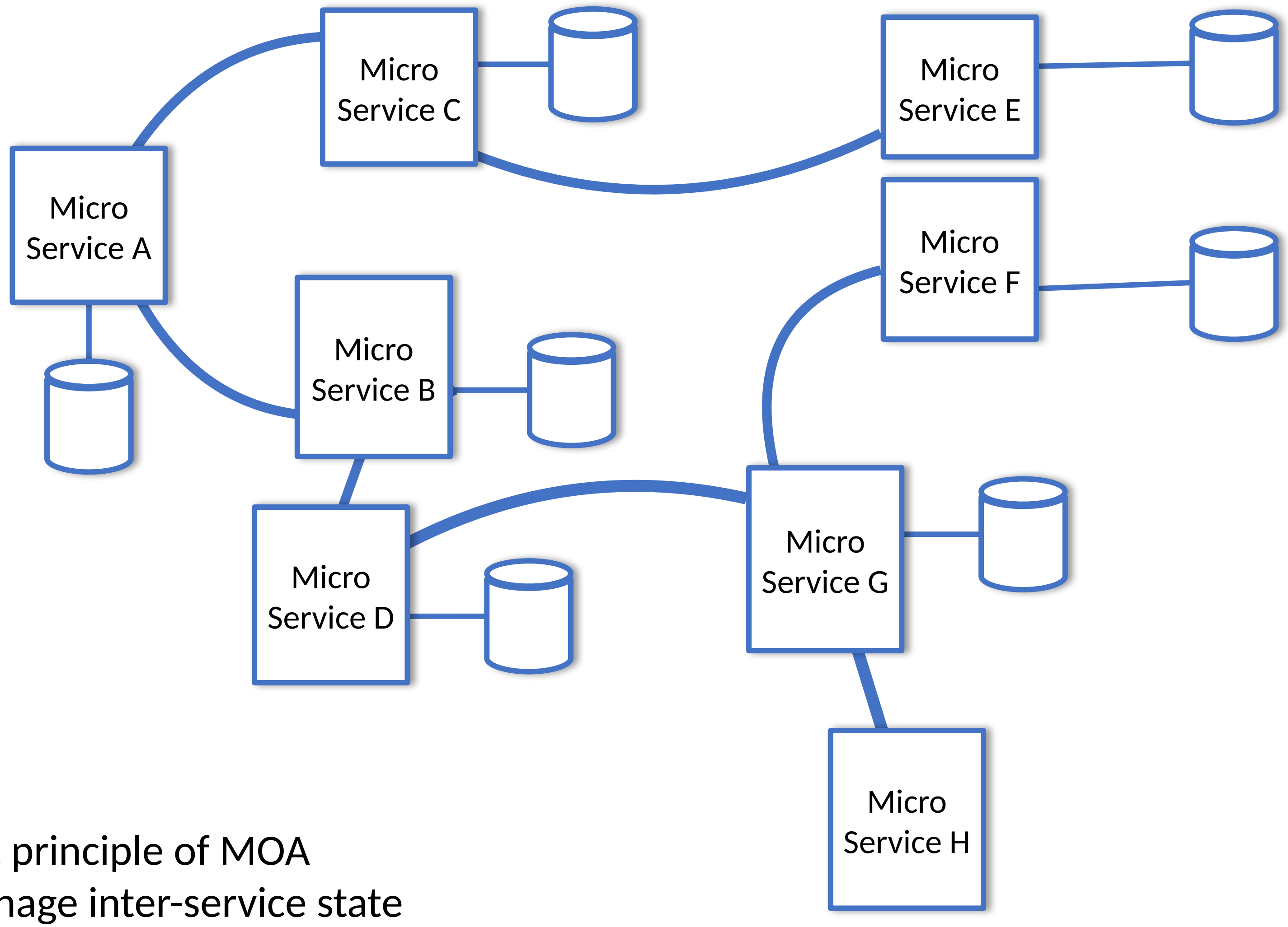
Microservices Architecture

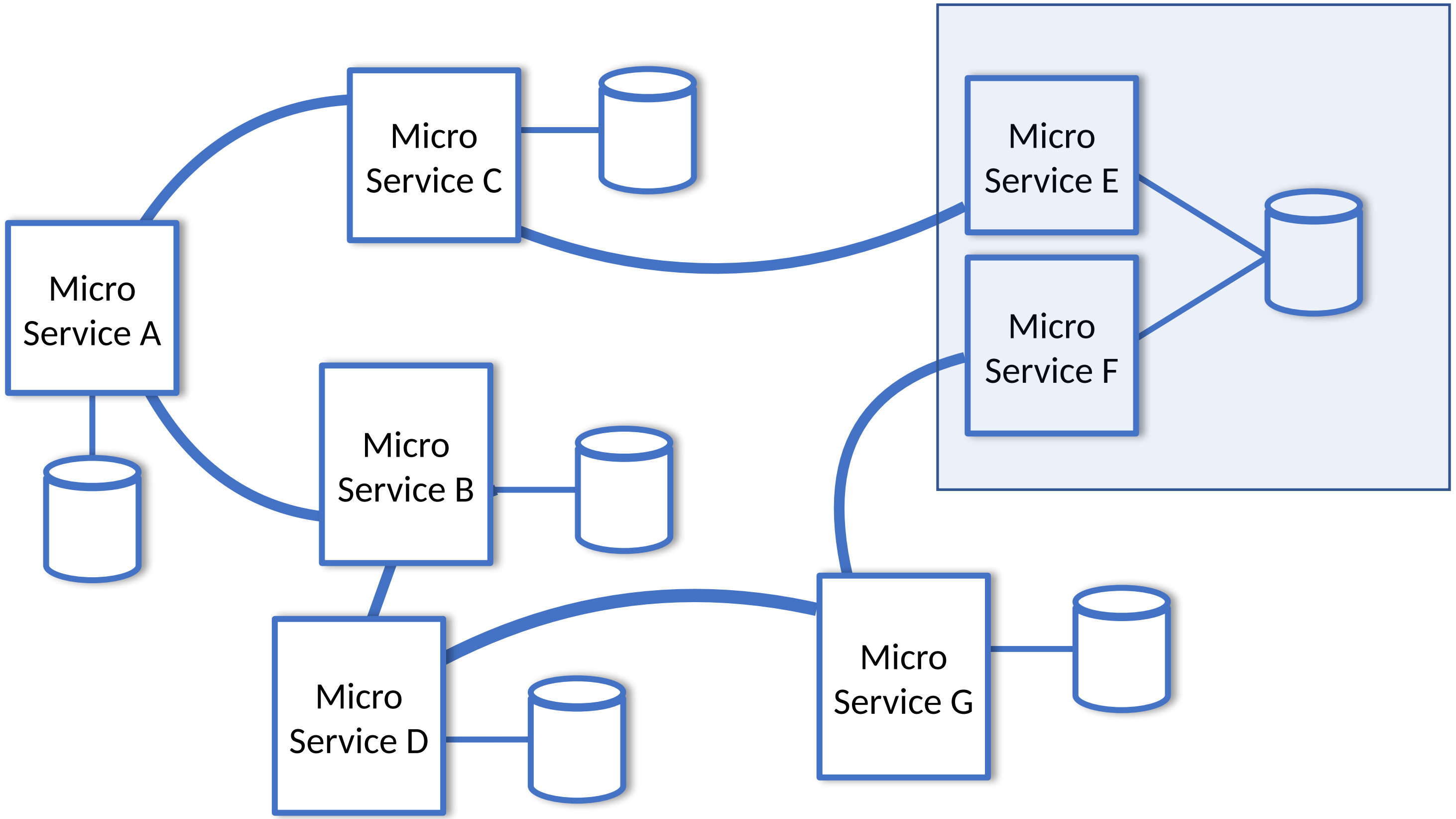
Data

Database Per Service



Good news: Supports a basic principle of MOA
Bad news: Really hard to manage inter-service state

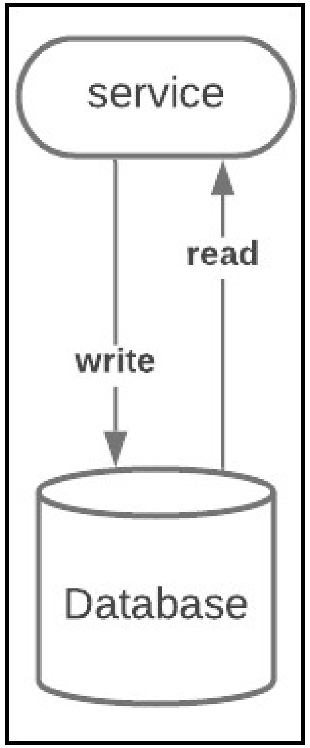
Shared Database per Service



Good news: Makes transactions easier
Bad news: Violates a basic principle of MOA

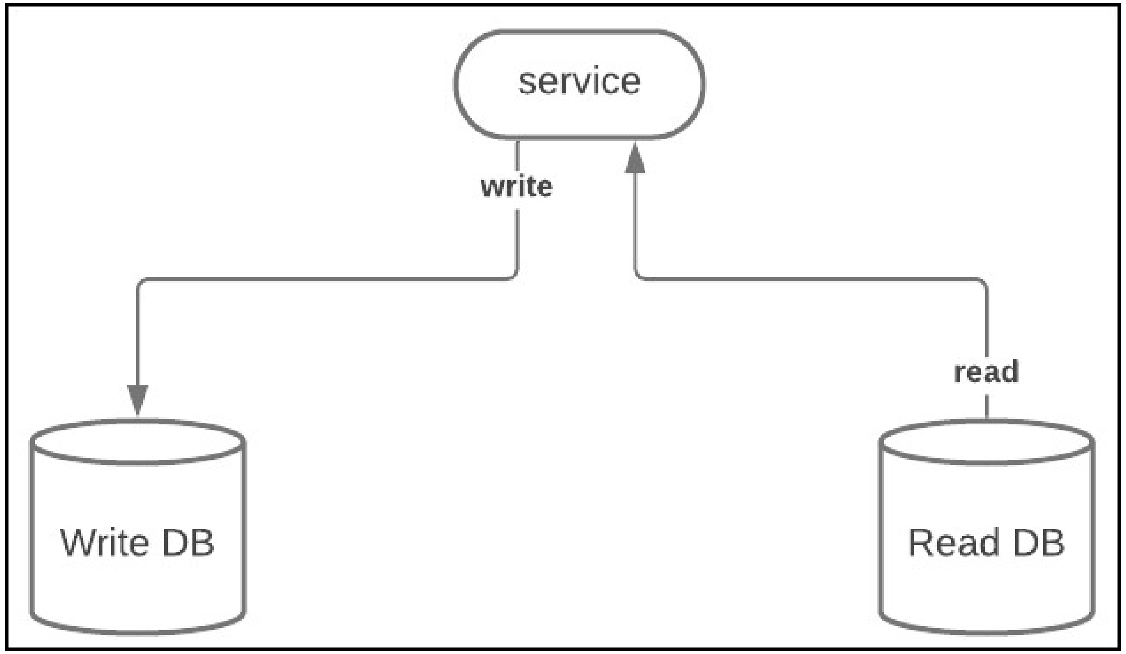
Command-Query Responsibility Segregation (CQRS)

- QRS pattern separates read and update operations for a data store
  - “Asking a question should not change the answer”
- Addresses issues with reading and writing to a single DB
  - Typically, there are more reads than writes
  - Optimization techniques are different for reads and writes
  - Writes will cause side effects in read behavior



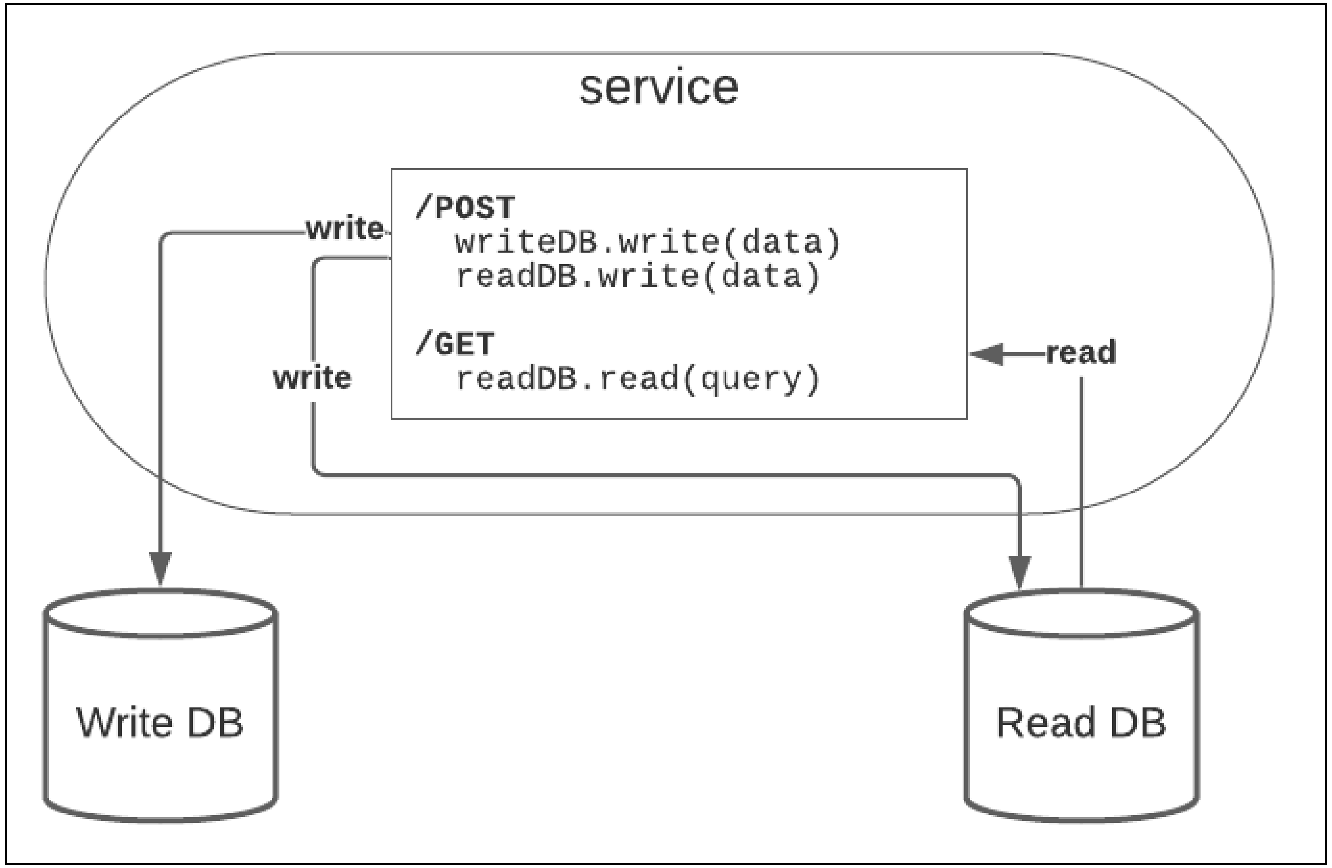
Command-Query Responsibility Segregation (CQRS)

- Separating reads and writes improves performance but
- The problem of synchronization now needs to be addressed

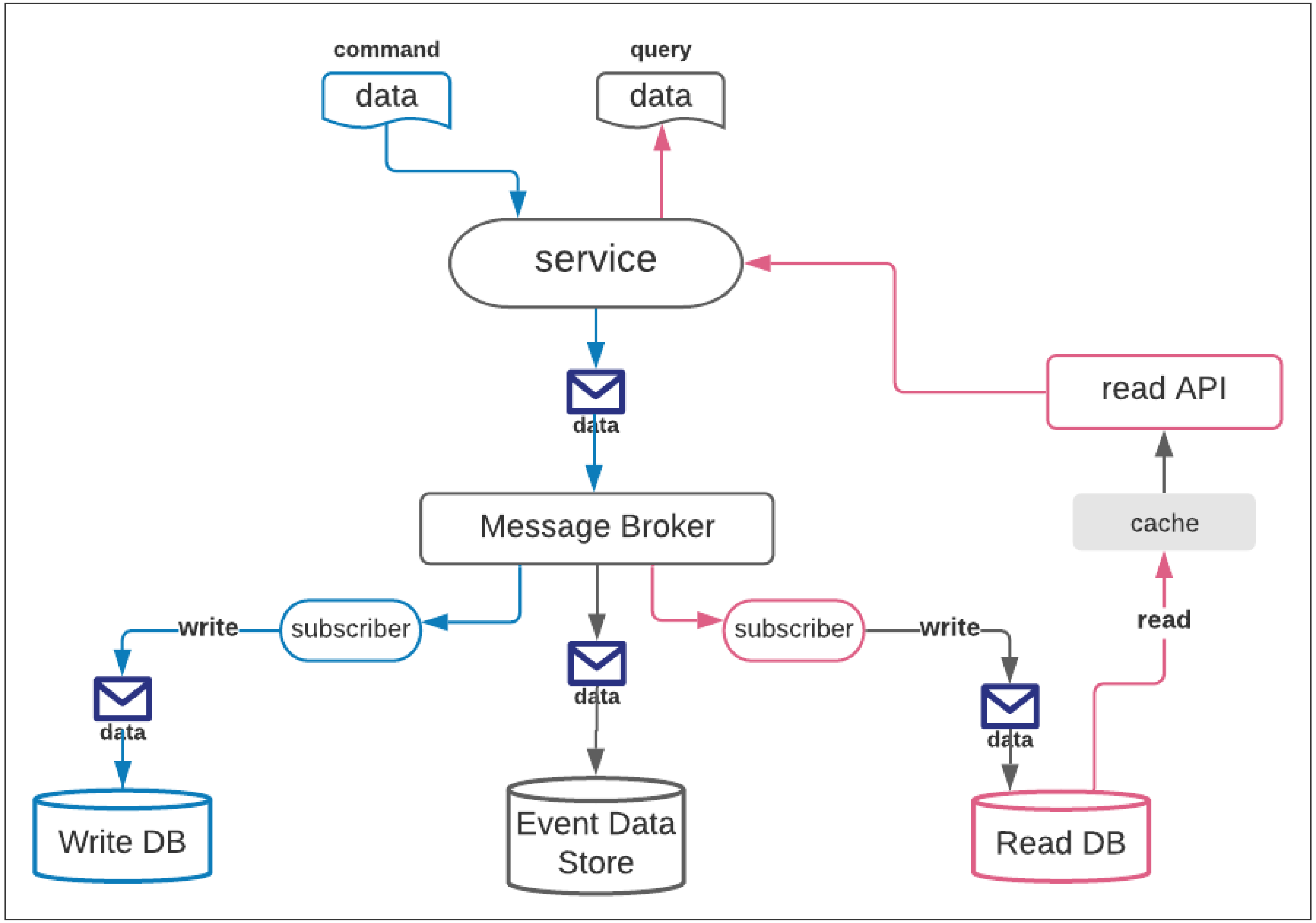


Command-Query Responsibility Segregation (CQRS)

- Different read and write services can use different technologies

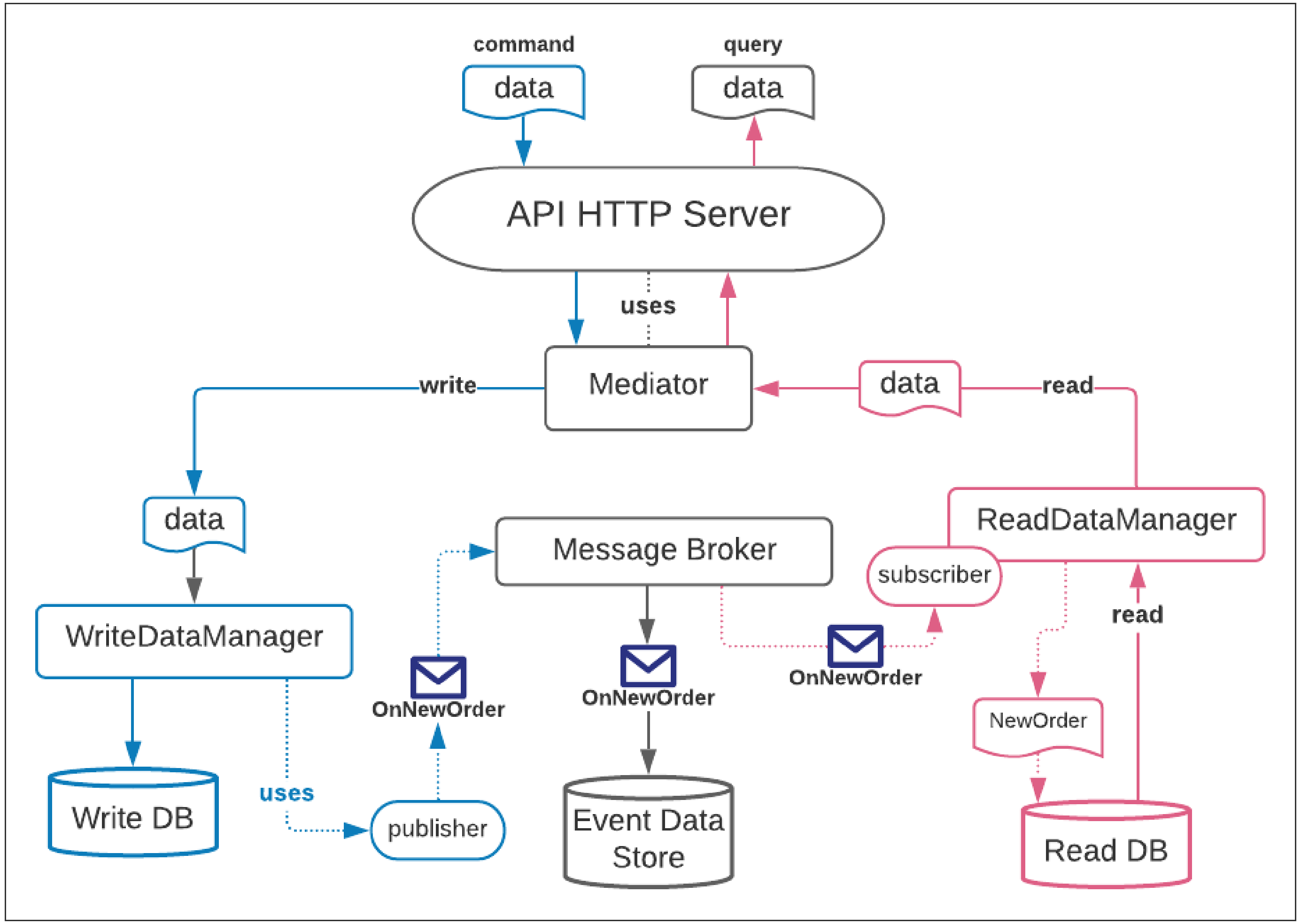


Working with CQRS

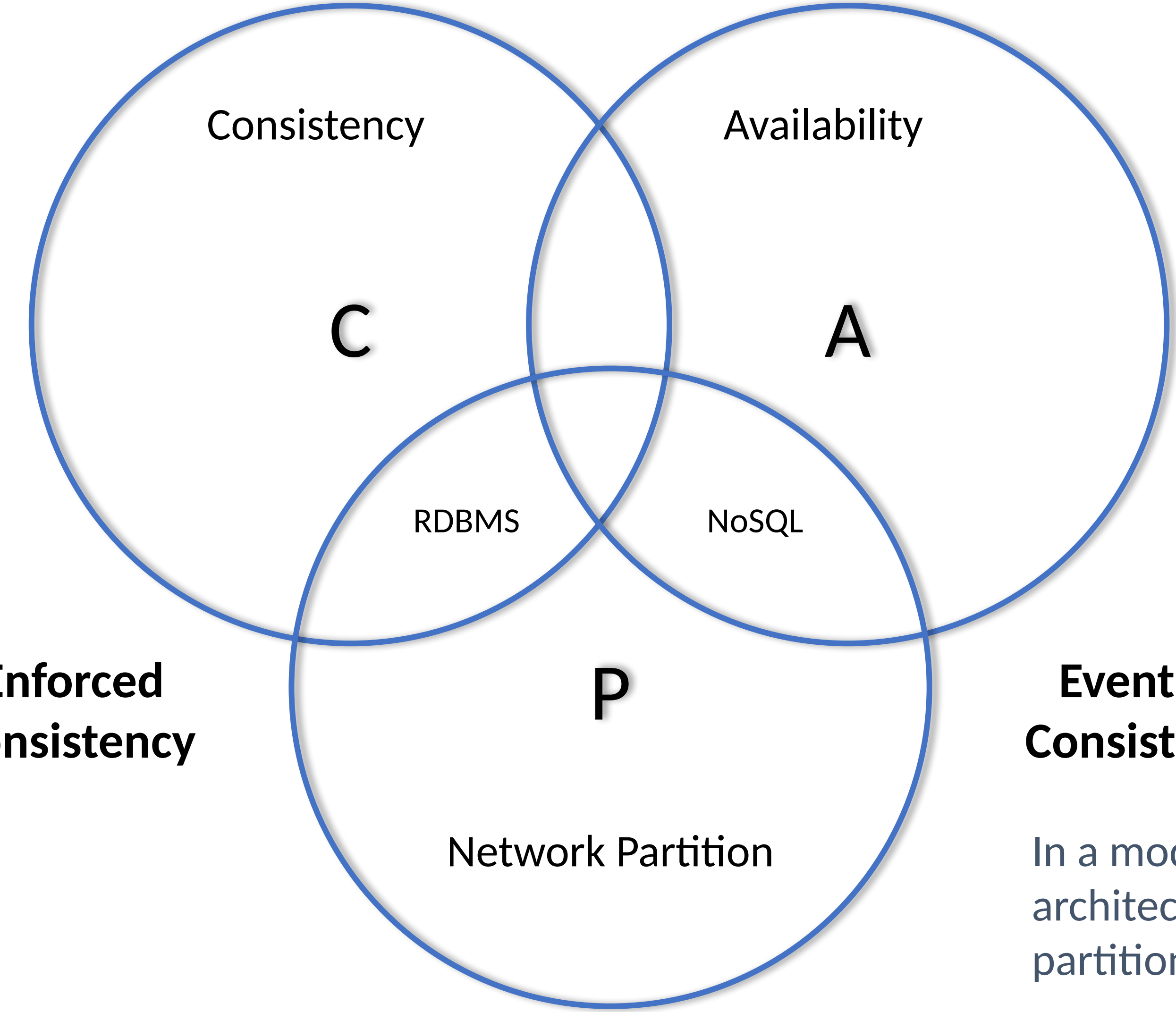
- Using a message driven architecture addresses many of the risks
- 

Using a Mediator

- Using the Mediator pattern reduces the risk of tight coupling



The Cap Theorem



Enforced Consistency

Eventual Consistency

In a modern distributed architecture, the network is always partitioned.

Understanding the CAP Theorem

- When data is distributed among many data sources then:
  - You can have inaccurate date available immediately; or
  - You can have accurate data available eventually
  - You cannot have accurate data available always

CAP Theorem: Consistency Models

		Consistency	Performance	Availability
Strong	See all previous writes.	excellent	poor	poor
Consistent Prefix	See initial sequence of writes.	okay	good	excellent
Bounded Staleness	See all "old" writes	good	okay	poor
Monotonic Read	See increasing subset of writes	okay	good	good
Read Your Writes	See all writes performed by reader	okay	okay	okay
Eventual	See subset of previous writes.	poor	excellent	excellent

Strongest

Wea

CAP Theorem: Consistency Models

- Strong Consistency
  - The data must be consistent at all times
  - All nodes everywhere should contain the same values at all times
  - Implemented by locking down the nodes when being updated
- Consistent Prefix
  - reads never see out of order writes
  - If writes were performed in the order A, B, C, then a client sees either A, A,B, or A,B,C, but never out of order like A,C or B,A,C
- Bounded Staleness
  - All observers have the same data at the same time
  - Writes may only be ahead of reads by a set number of ops or time lag

CAP Theorem: Consistency Models

- Monotonic Read
  - Read operations do not return results that correspond to an earlier state of the data than a preceding read operation
  - For example, if in a session:
    - *write1 precedes write2,*
    - *read1 precedes read2, and*
    - *read1 returns results that reflect write2*
    - *then read2 cannot return results of write1*
- Read Your Writes
  - Guarantees that once a record has been updated, any attempt to read the record will return the updated value.

CAP Theorem: Consistency Models

- Eventual Consistency
  - Ensures high availability to all users
  - If no new updates are made to a given data item, eventually all accesses to that item will return the last updated value

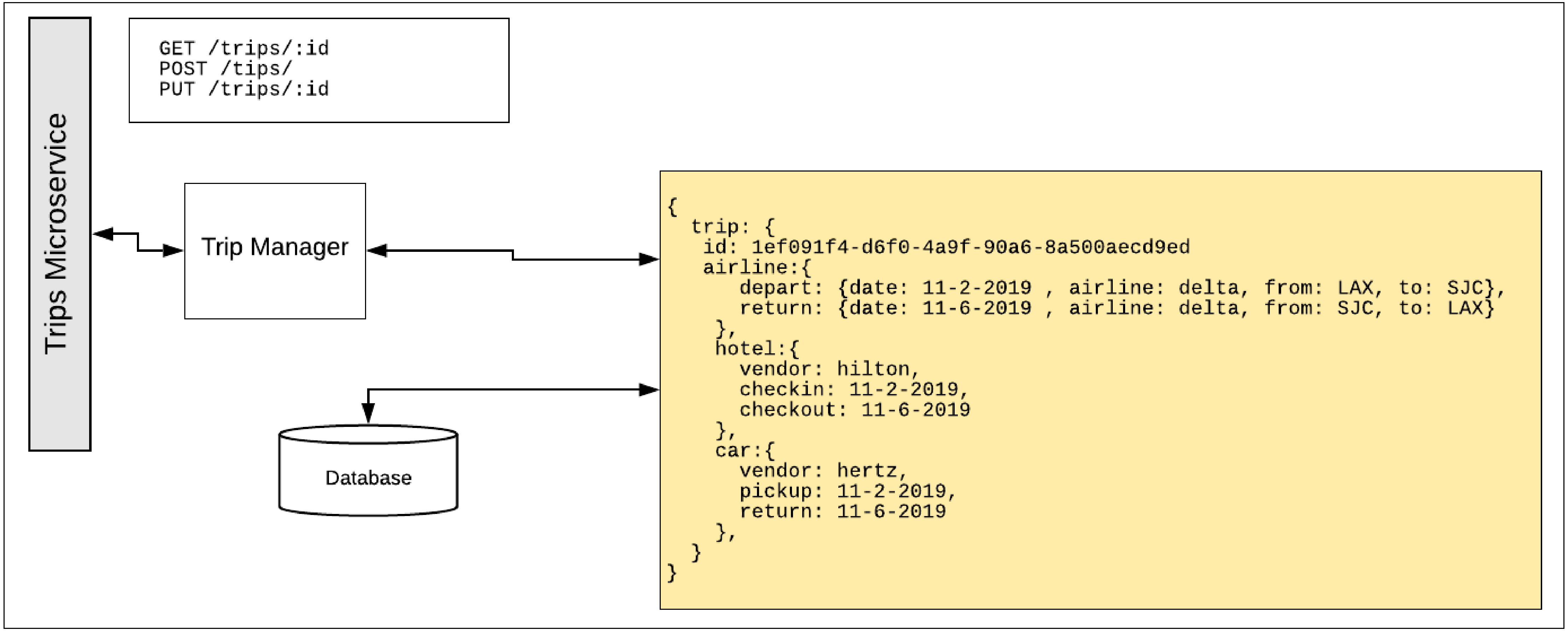
Data Management: Transactions

- Transaction design patterns
  - Same Bounded Context
  - State Store
  - 2 Phase Commit Protocol (2PC)
  - Routing Slip

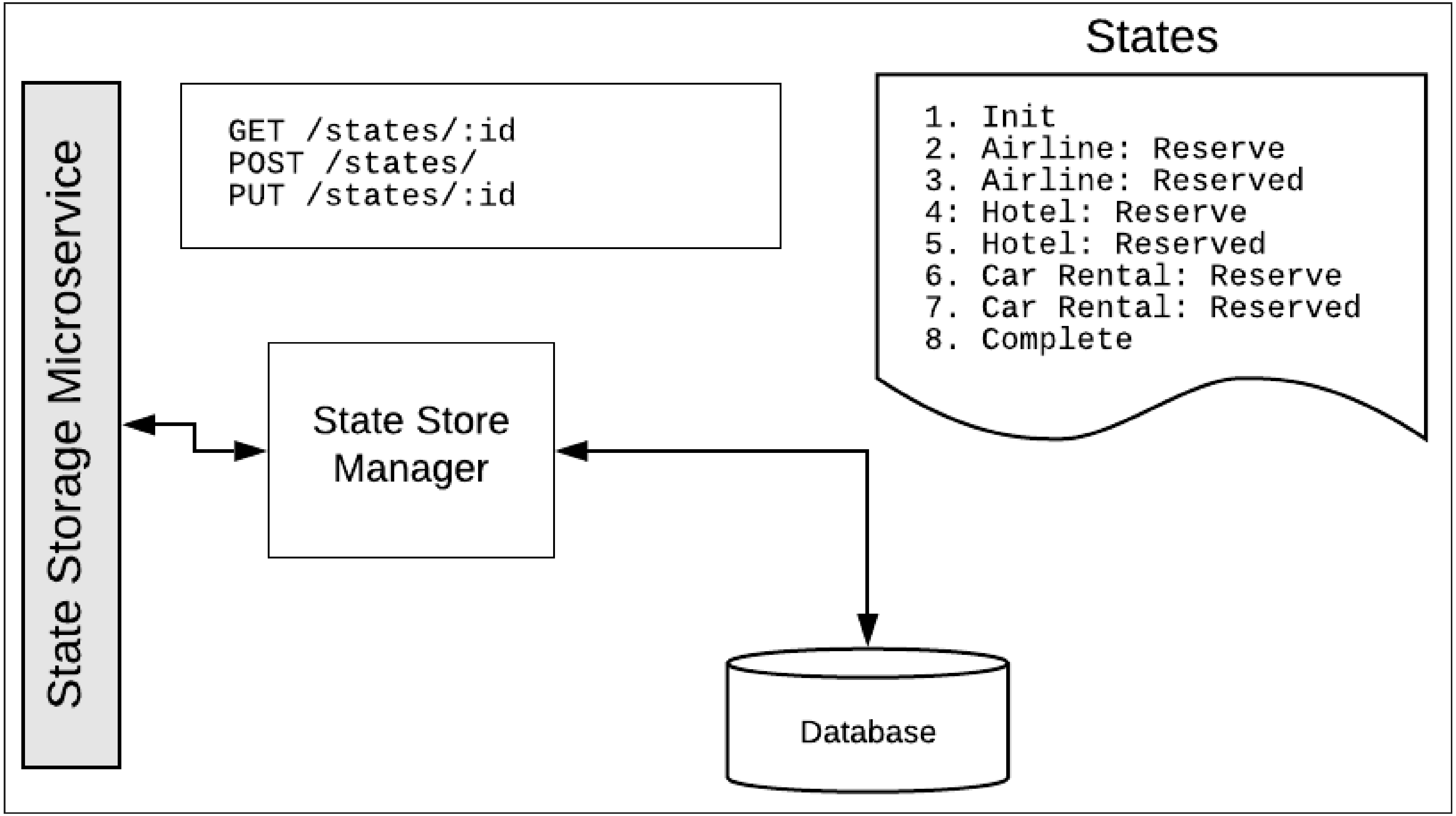
Implementing a transactions that occurs across several microservices residing in a variety of datacenters is always a challenge. The easiest thing to do is to avoid inter-service transactions completely.

Sometimes circumstances in the real world require supporting transactions. So, we need to do what the need demands. However, be advised supporting inter-service transactions is rarely an easy undertaking.

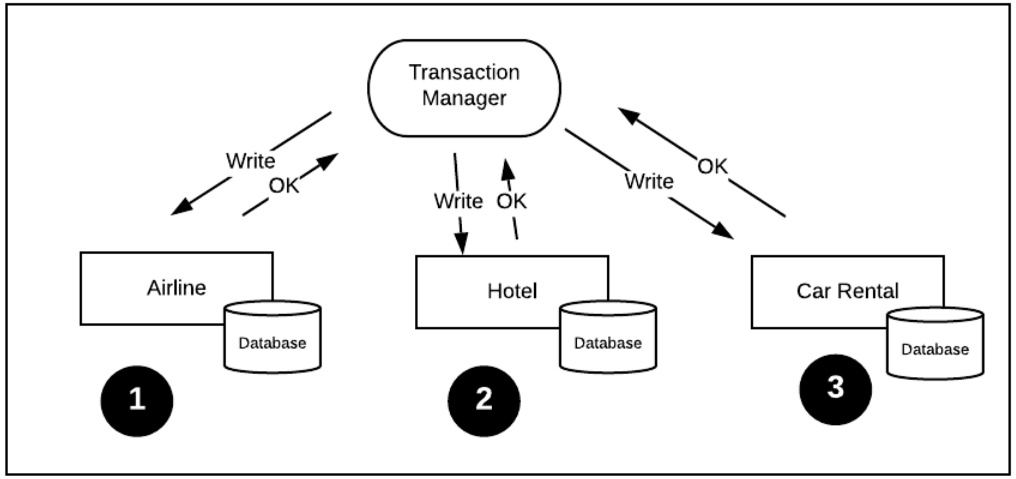
Transactions: Same Bounded Context

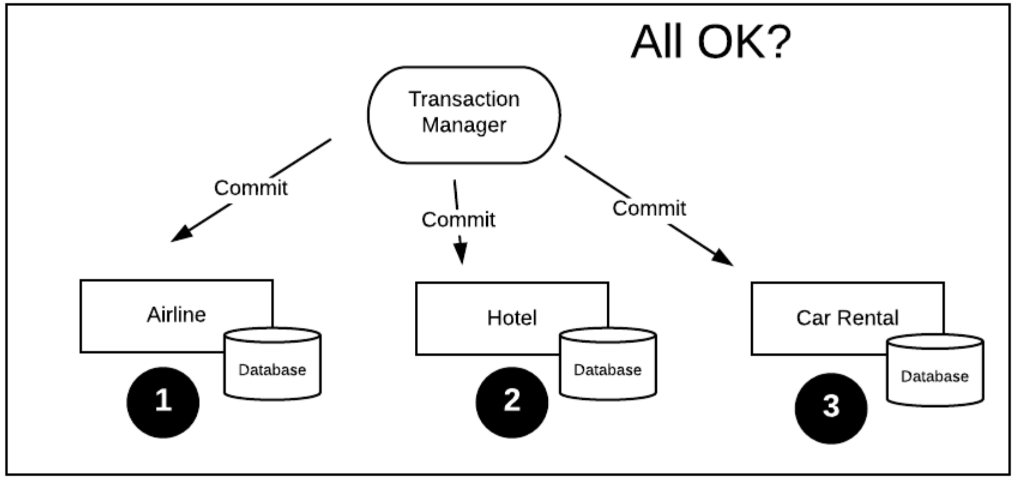


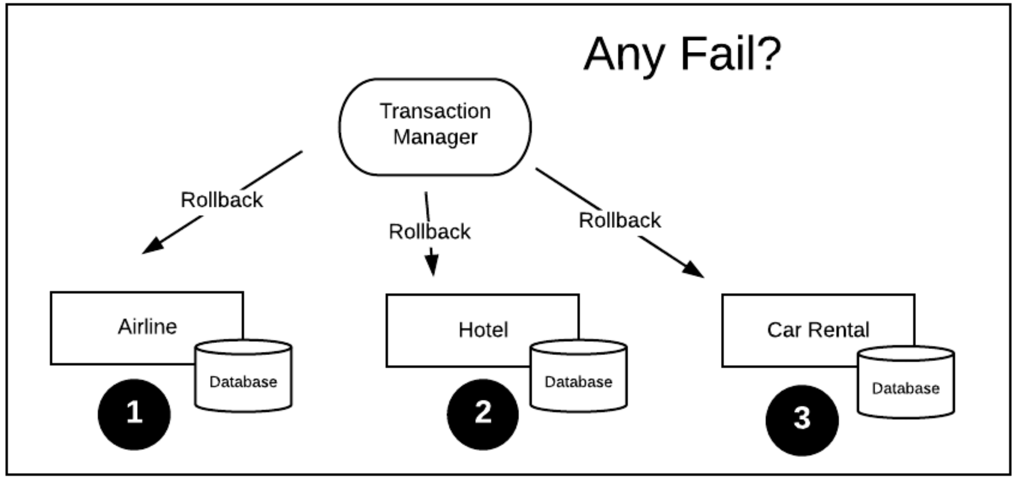
Transactions: State Store



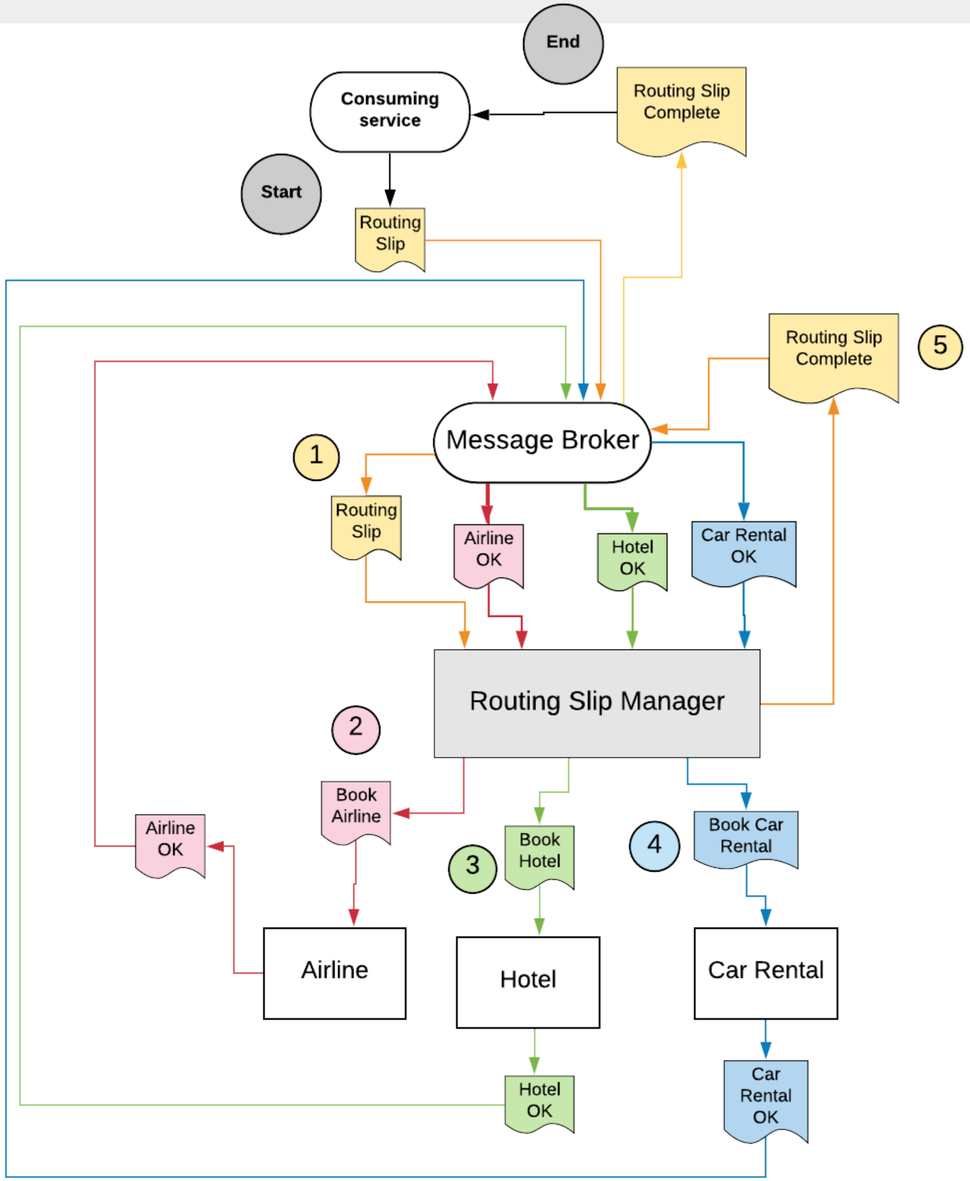
Transactions: Two Phase Commit (2PC)







Transactions: Routing Slip



End of Module