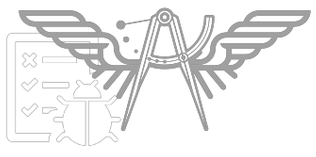
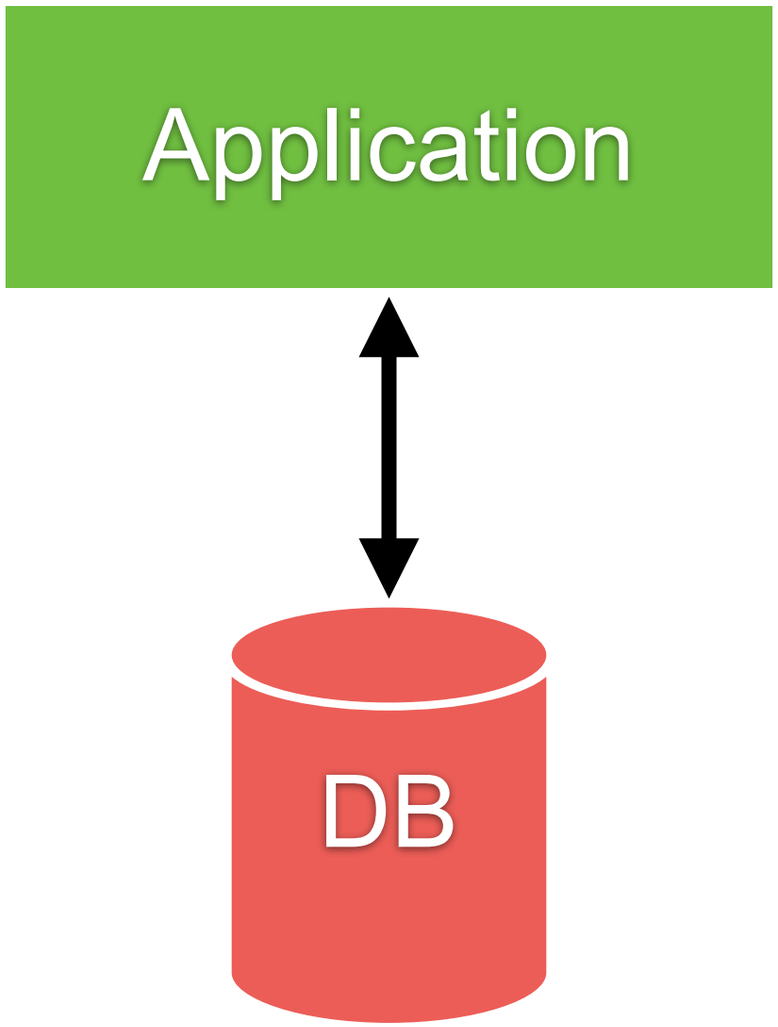
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Data management In Microservices



Data management in Microservices

Monolithic



Microservices

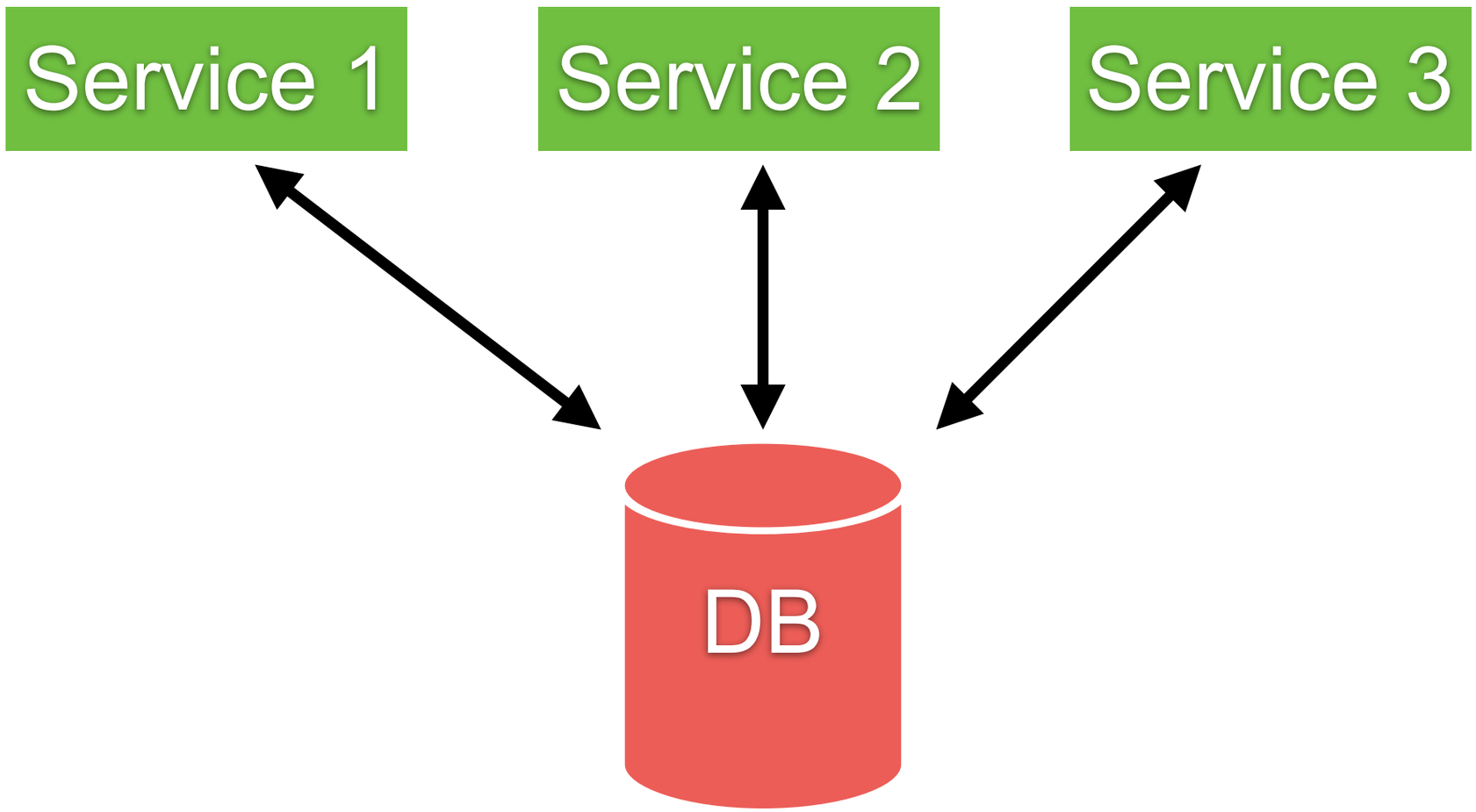
Service 1

Service 2

Service 3

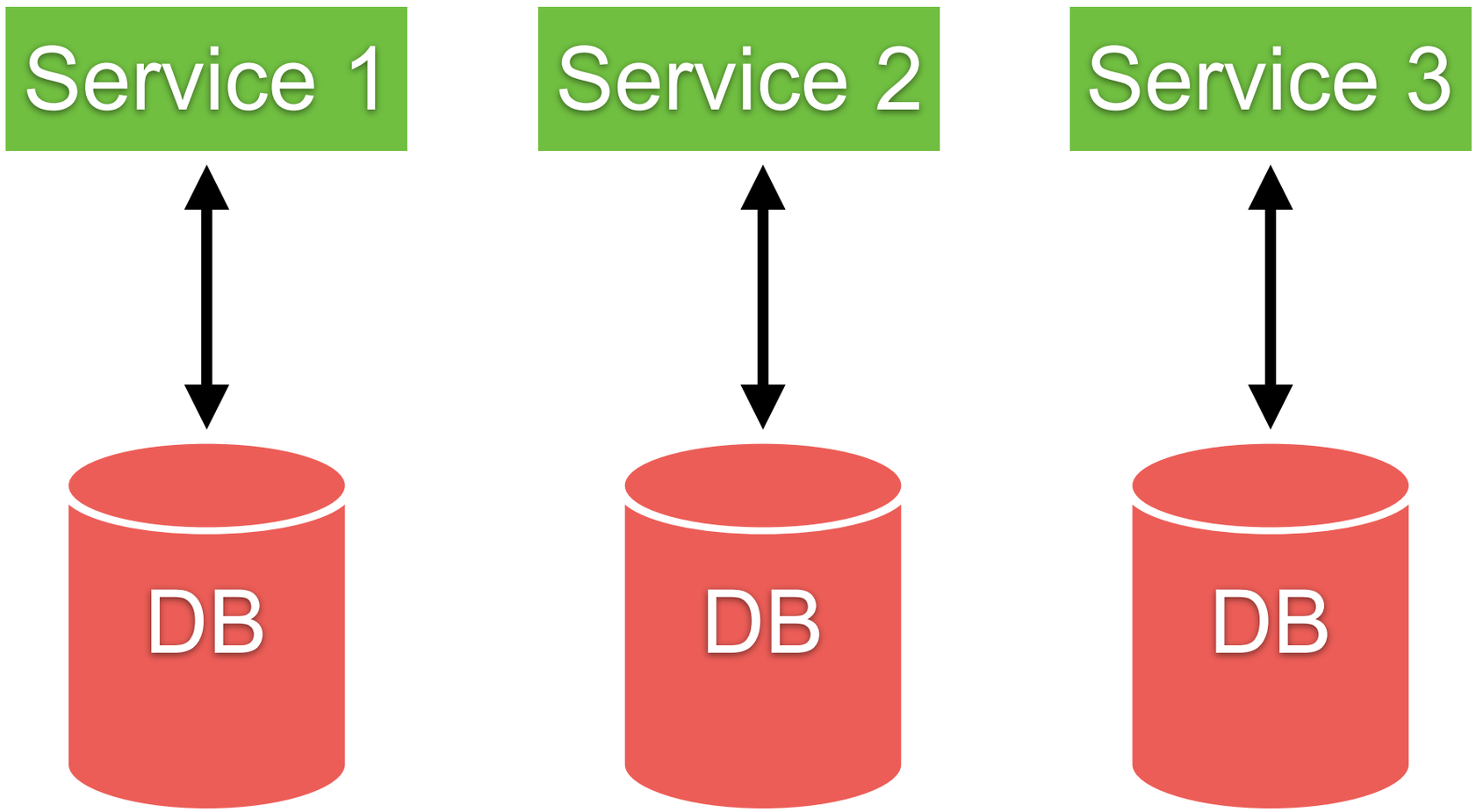
Microservices

1. Shared database



Microservices

2. Database per service



Database per service

High complexity

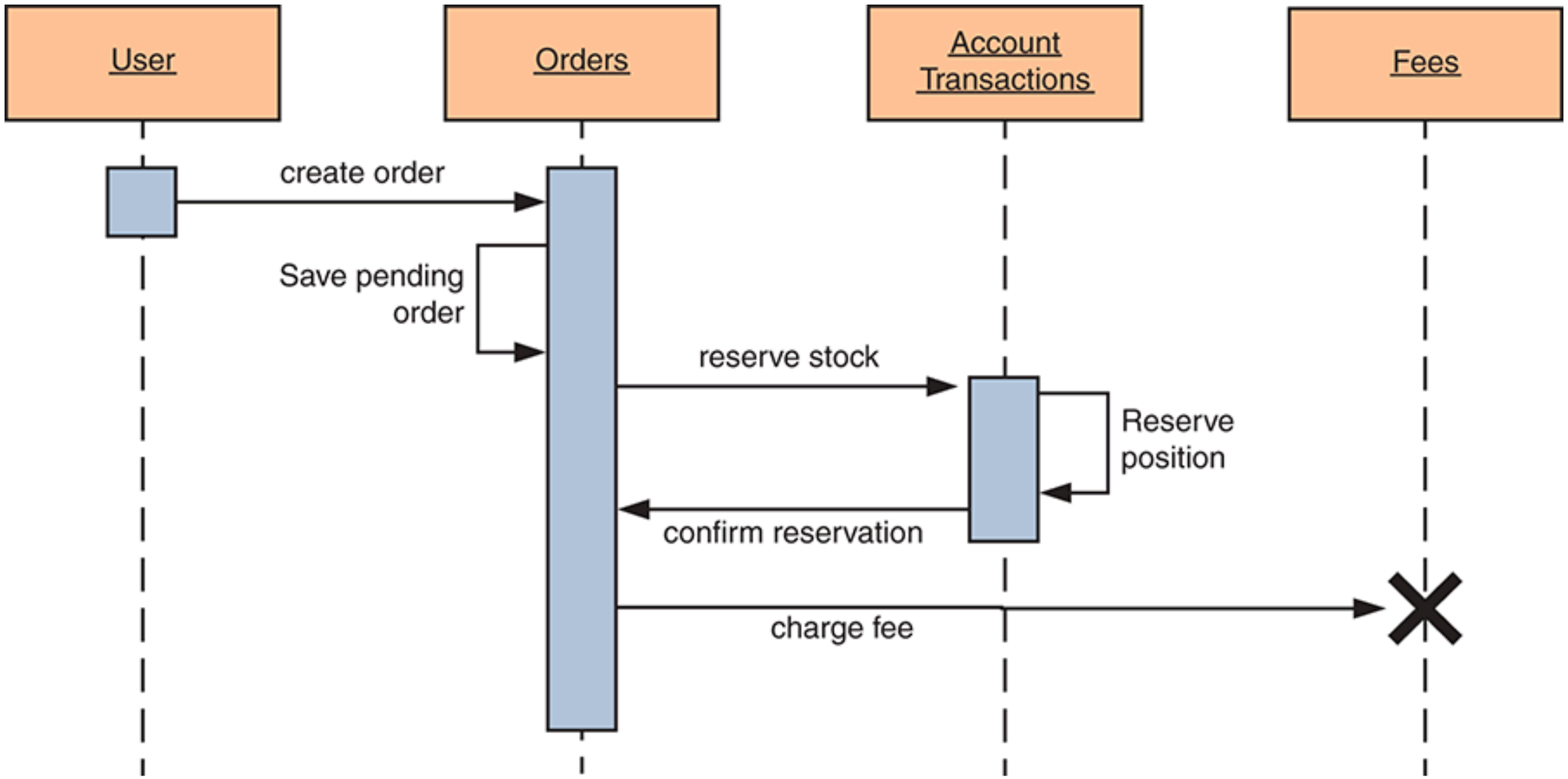
Query data across multiple databases

Challenge “How to join data ?”

Maintain database consistency

How to maintain data consistency across services ?

Problem



Distributed transaction

Common approach is Two-phase commit(2PC)

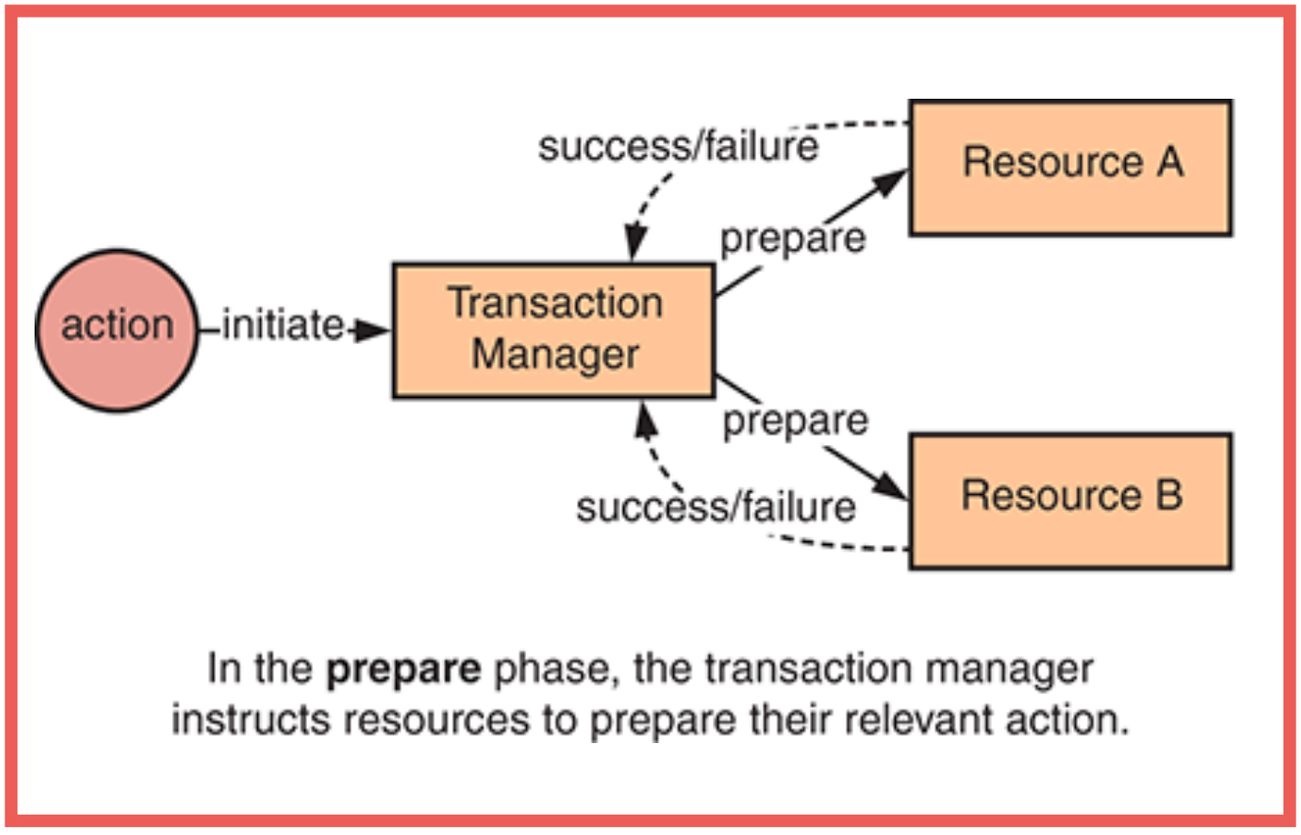
Use transaction manager to split operations across multiple resources in 2 phases

1. Prepare

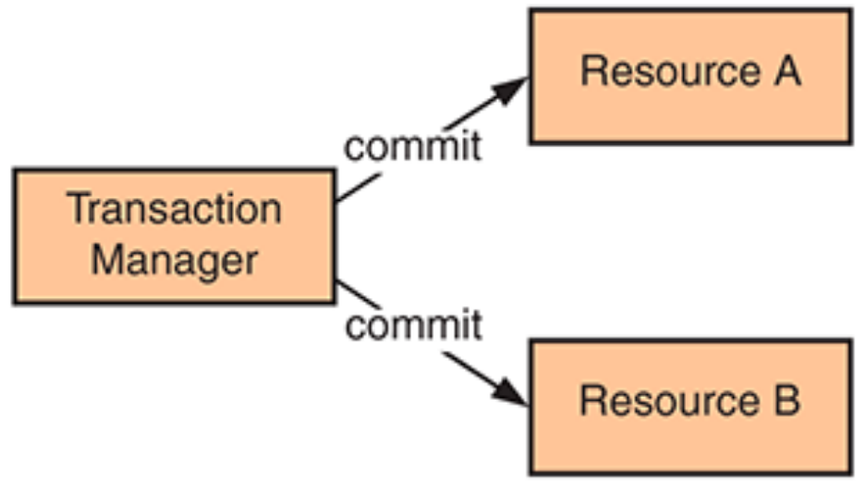
2. Commit

Two-phase commit

Phase 1: prepare



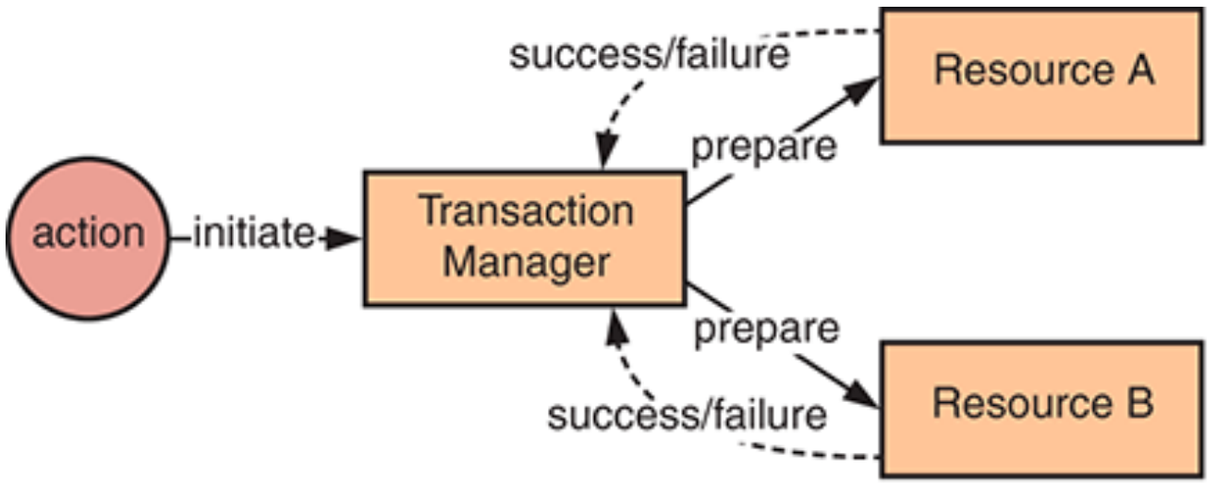
In the **prepare** phase, the transaction manager instructs resources to prepare their relevant action.



In the **commit** phase, the transaction manager instructs resources to commit or abort the prepared action.

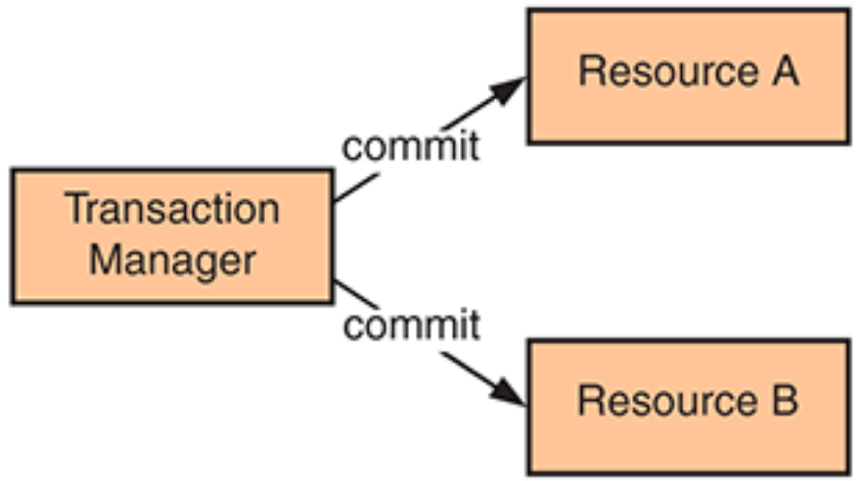
https://en.wikipedia.org/wiki/Two-phase_commit_protocol

Two-phase commit



In the **prepare** phase, the transaction manager instructs resources to prepare their relevant action.

Phase 2: commit



In the **commit** phase, the transaction manager instructs resources to commit or abort the prepared action.

https://en.wikipedia.org/wiki/Two-phase_commit_protocol

Distributed transaction places a lock on resources under transaction to ensure isolation

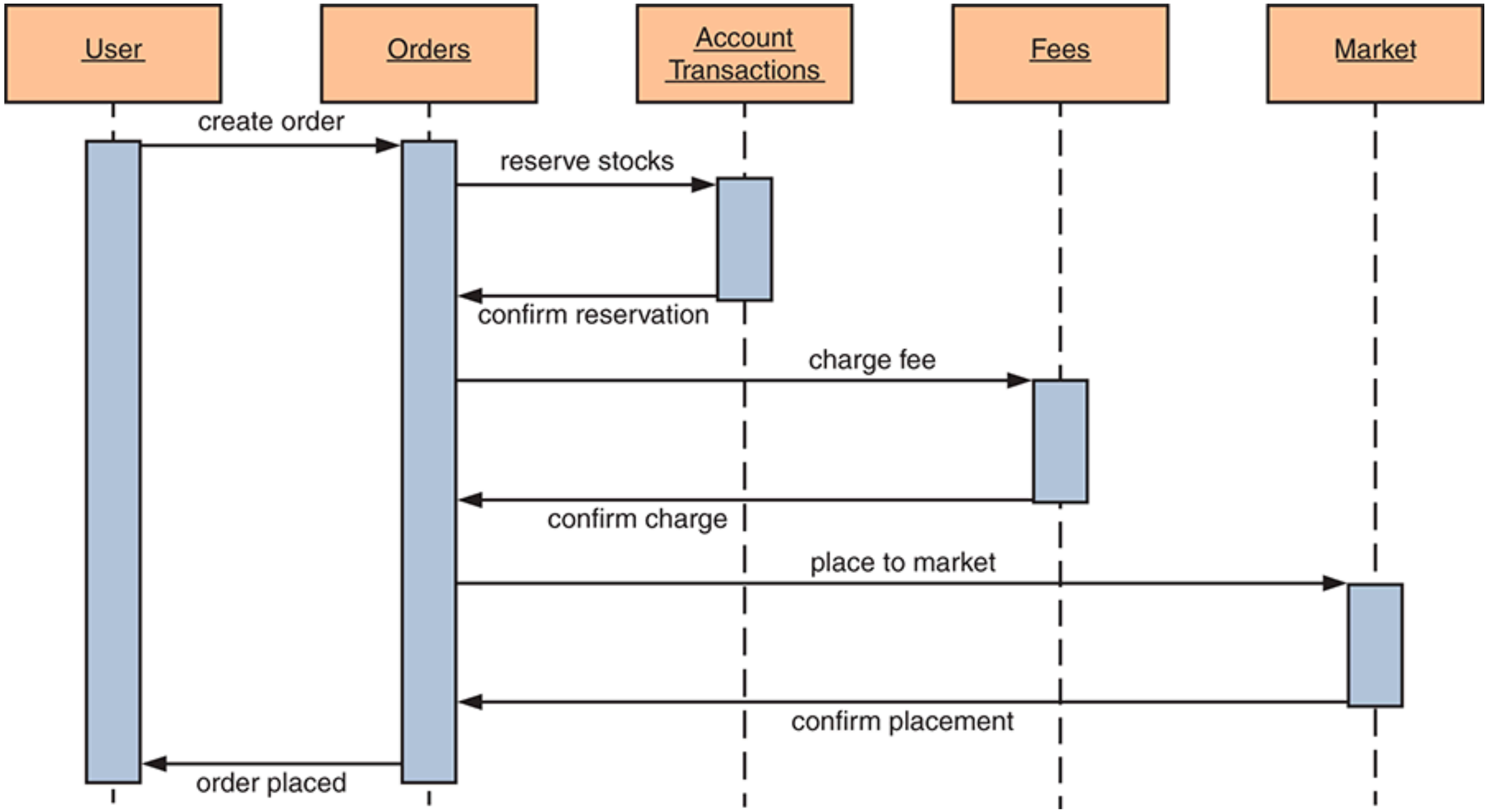
Inappropriate for long-running operations

Increase risk of contention and deadlock

We need …

Don’t need distributed transaction
Reduce complexity of code
Reliable

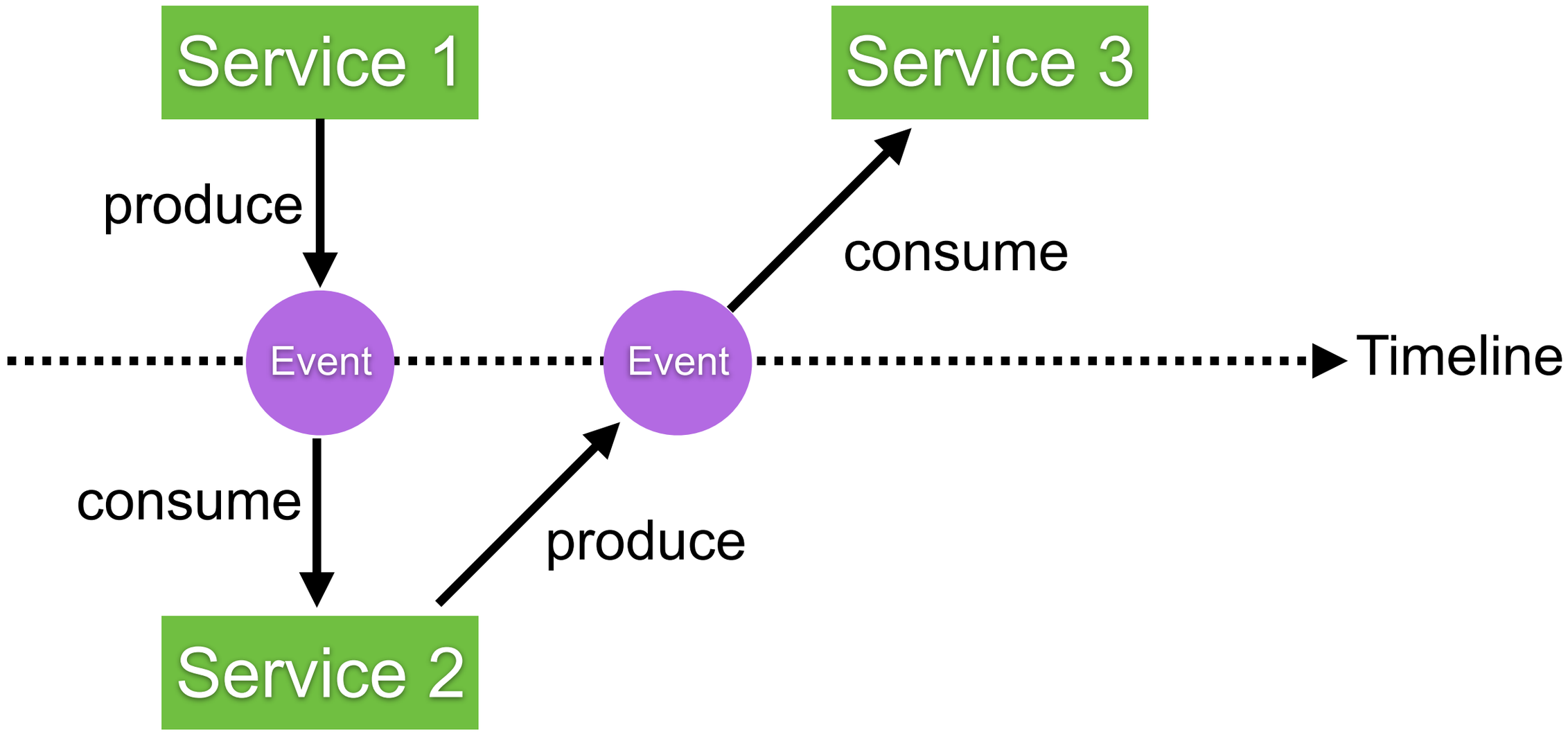
Synchronous process



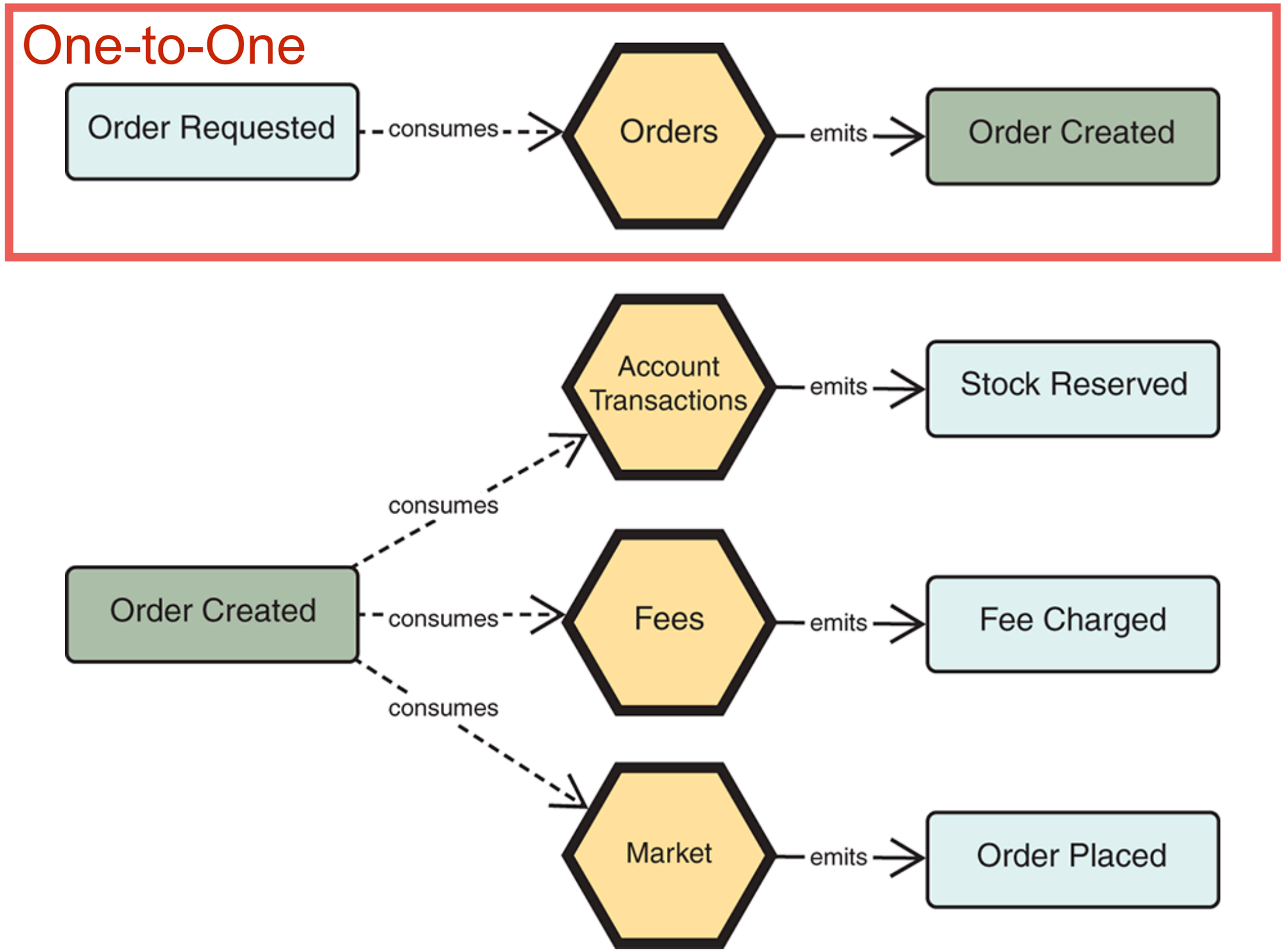
Event-based communication

Choreography pattern

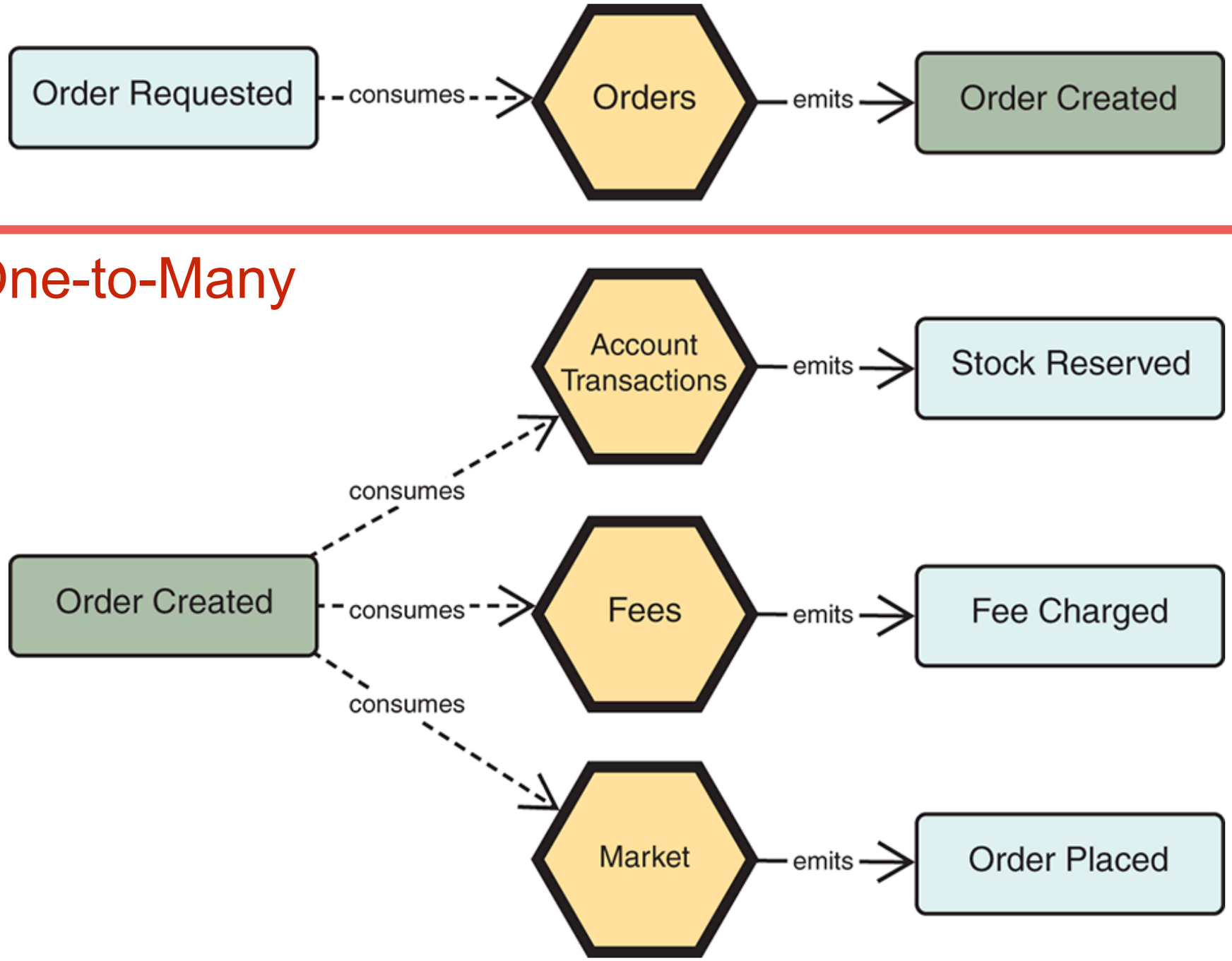
Sequence of Tx and emit **event** or **message** that trigger the next process in Tx



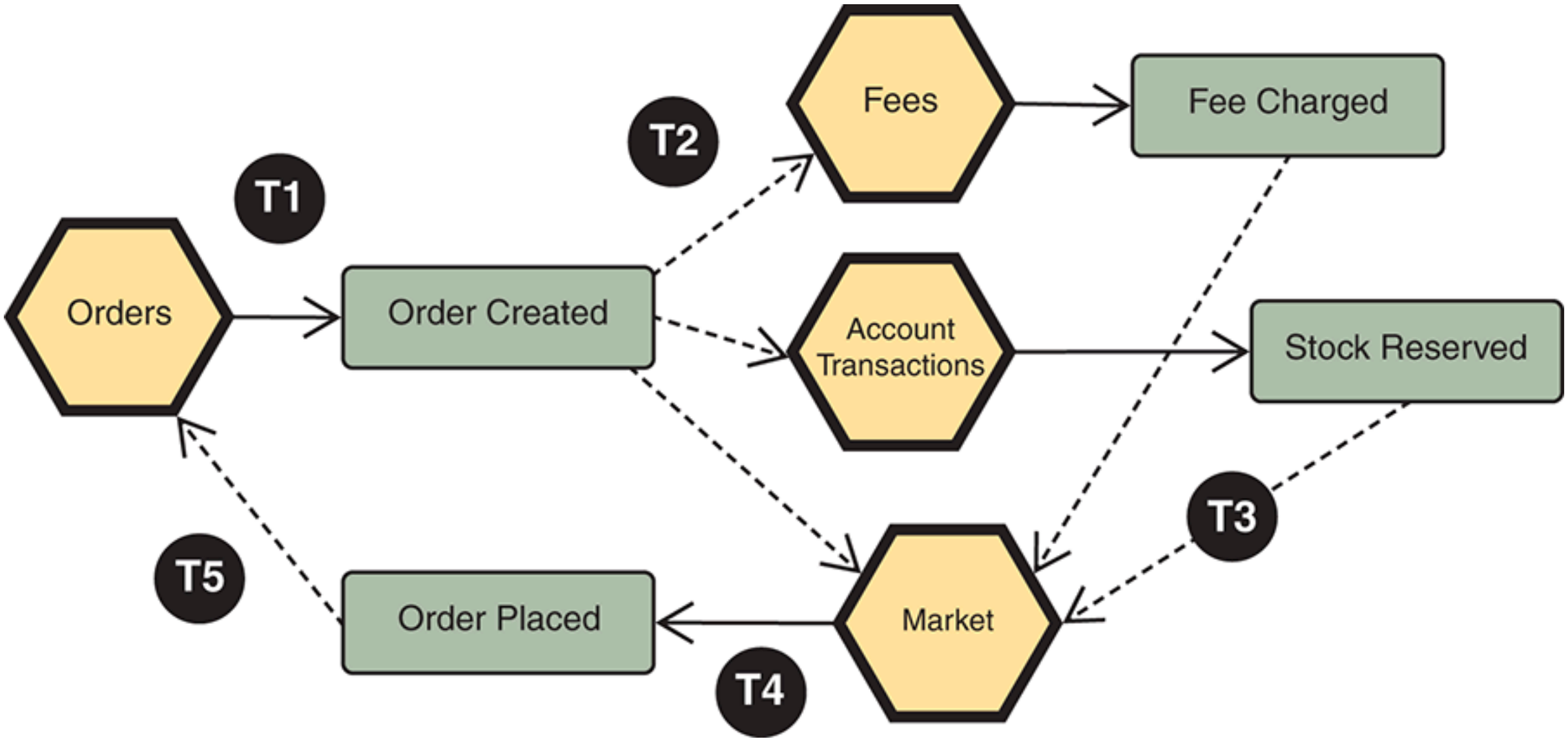
Service consume and emit event



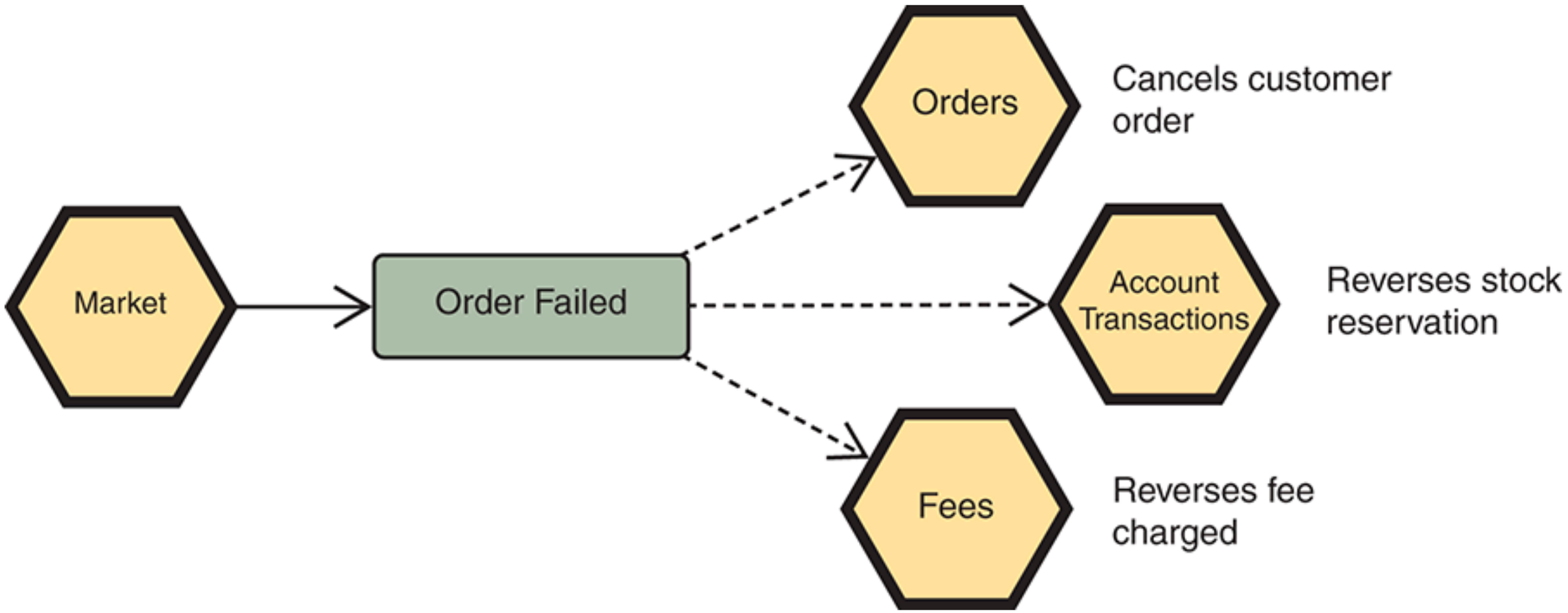
Service consume and emit event



Example (Success)

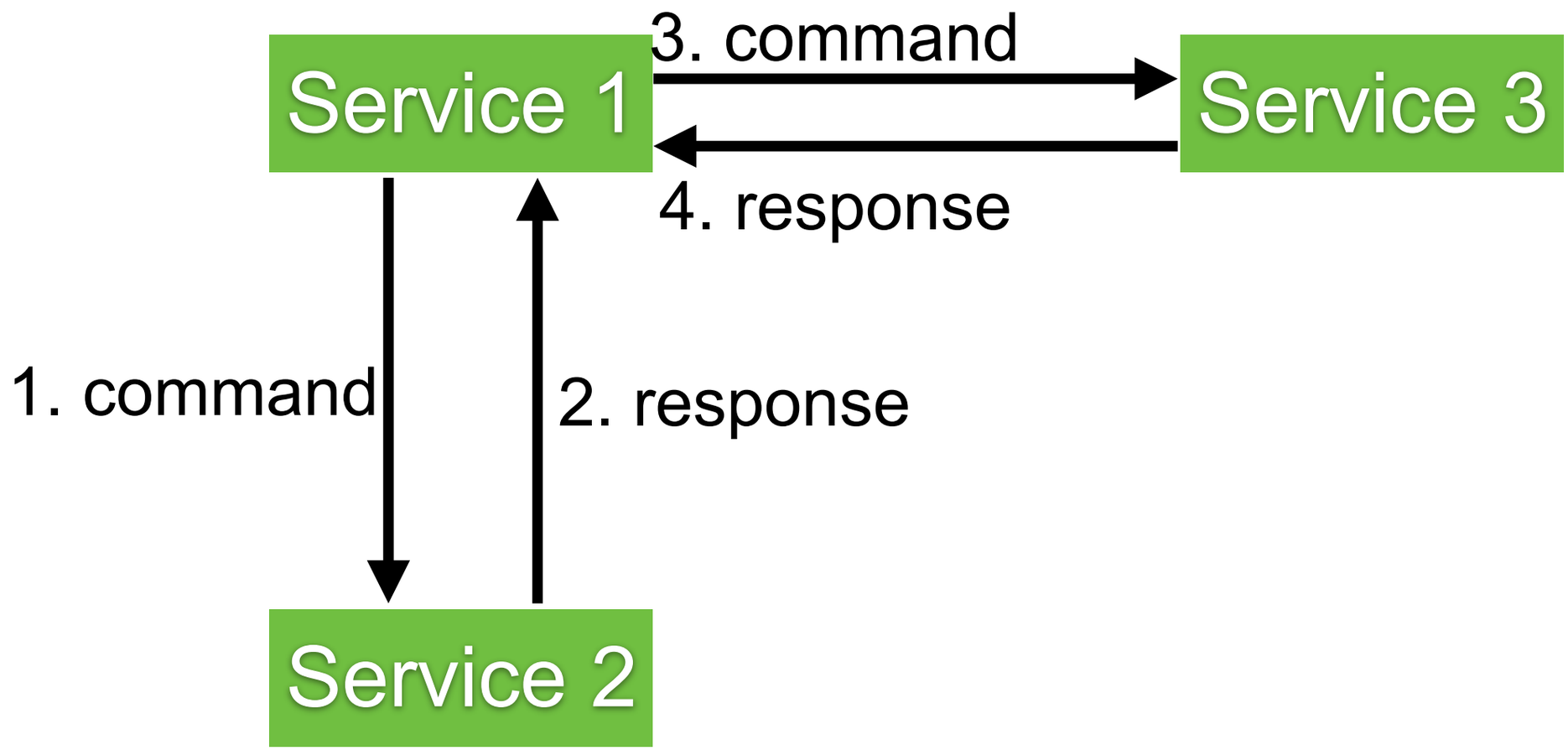


Example (Fail)

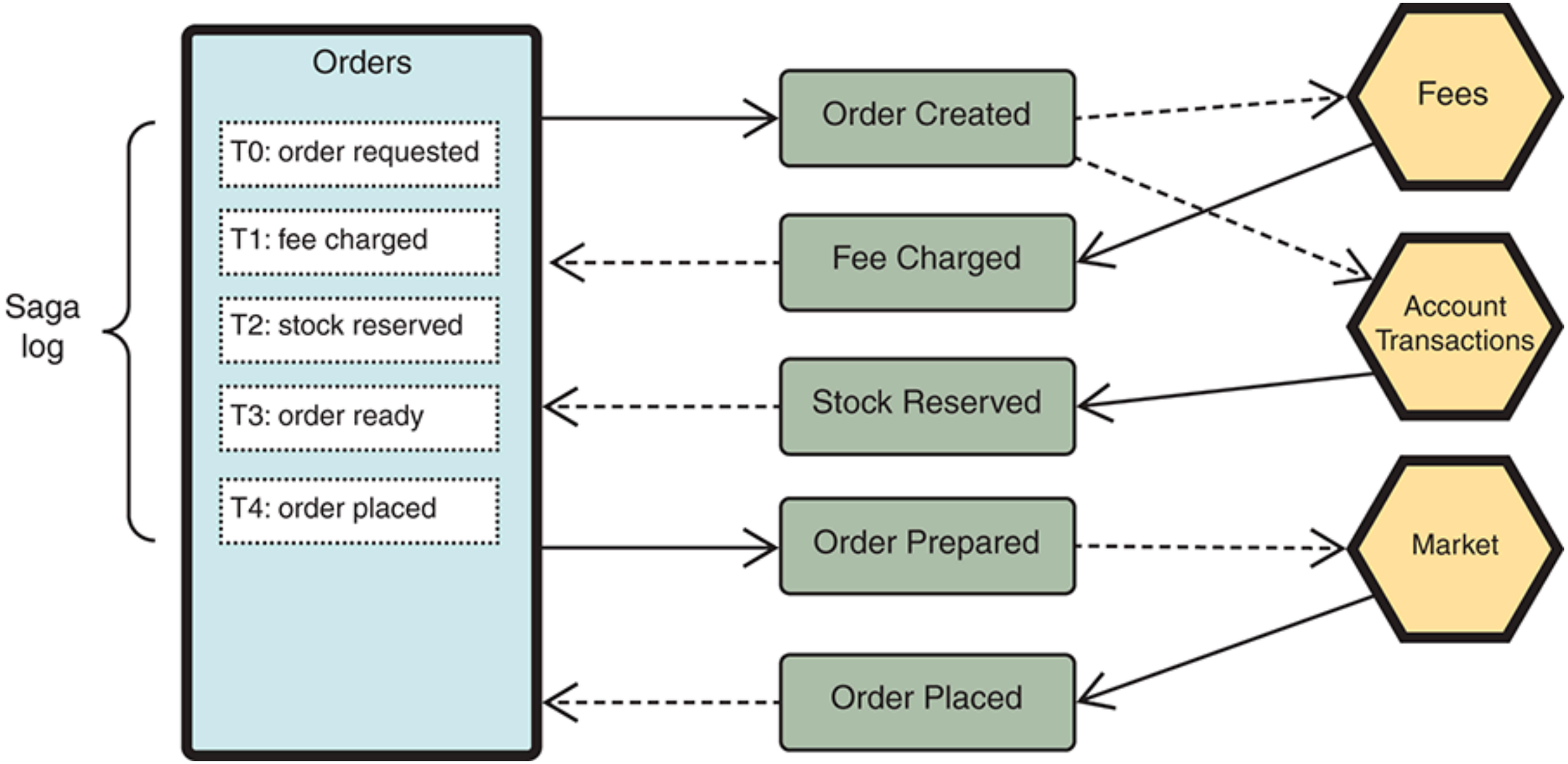


Orchestrate pattern

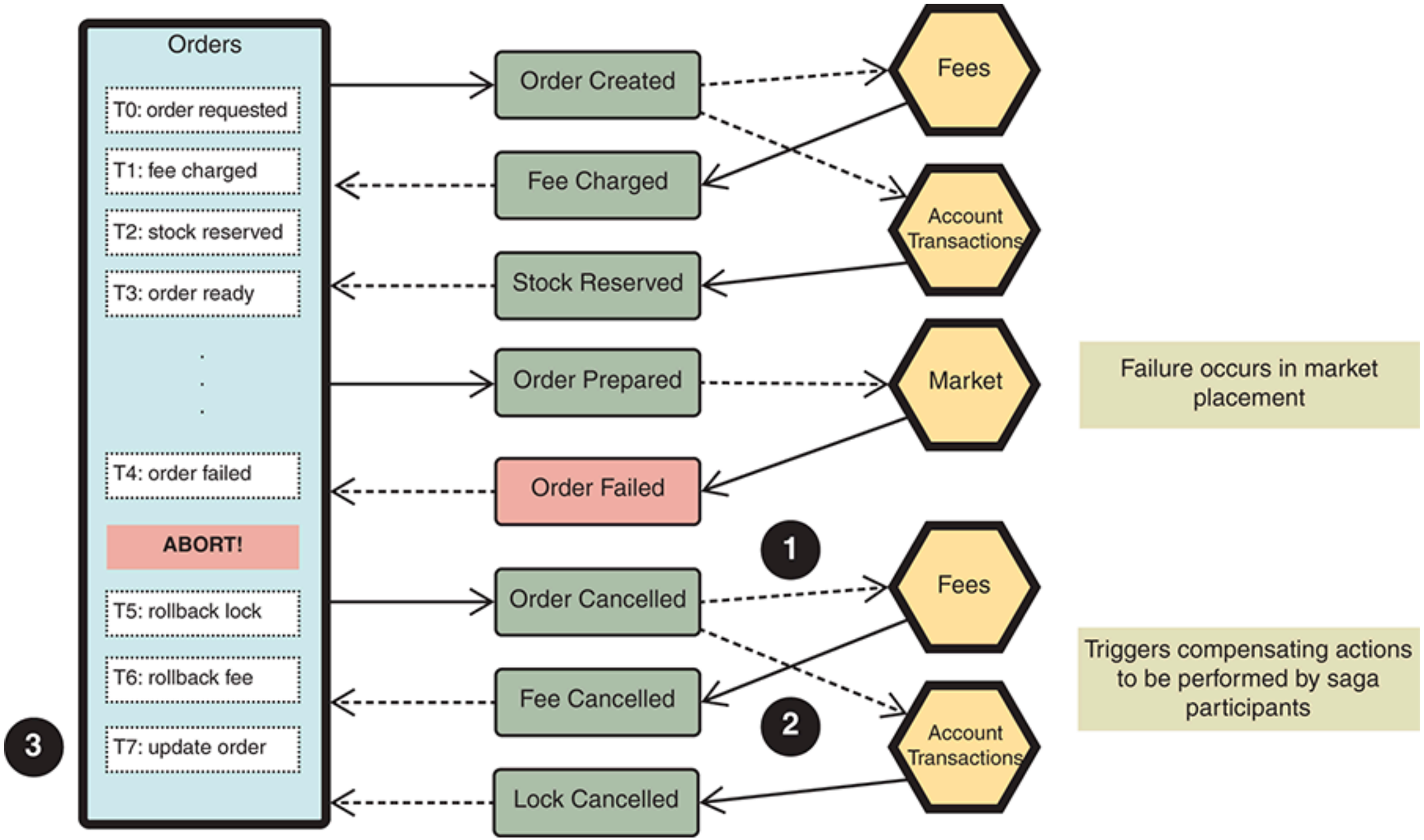
Sequence of Tx and emit **event** or **message** that trigger the next process in Tx



Example (Success)



Example (Fail)



Consistency patterns

Name	Strategy
Compensating action	Perform action that undo prior action(s)
Retry	Retry until success of timeout
Ignore	Do nothing in the event of errors
Restart	Reset to the original state and start again
Tentative operation	Perform a tentative operation and confirm (or cancel) later

“Saga”

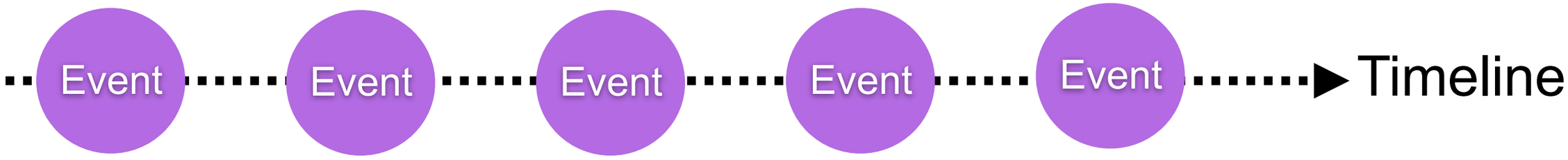
Message vs Event

Message is addressed to someone

Event is something that happen and someone can react to that

Event sourcing

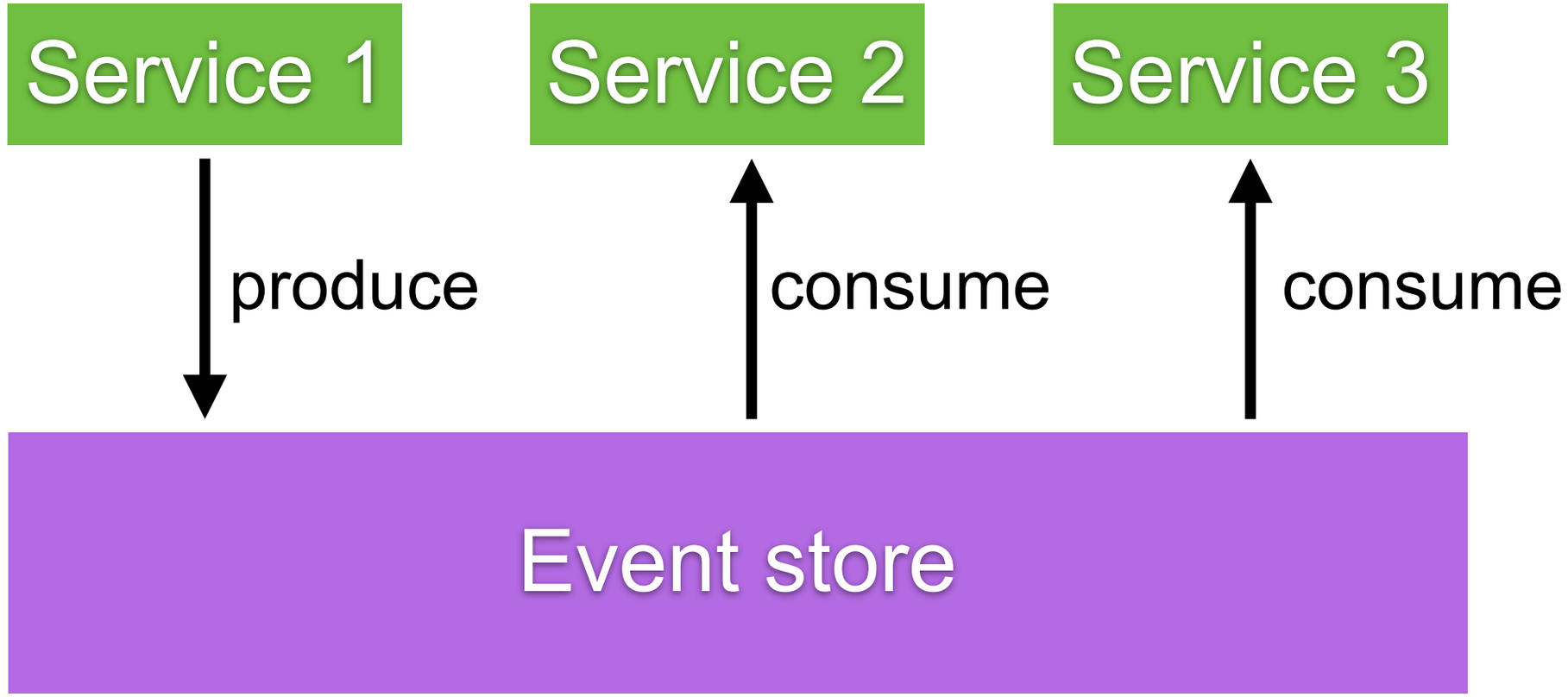
Maintain source of truth of business
Immutable sequence of events



Sequence of event is keep in **Event store**

Event store

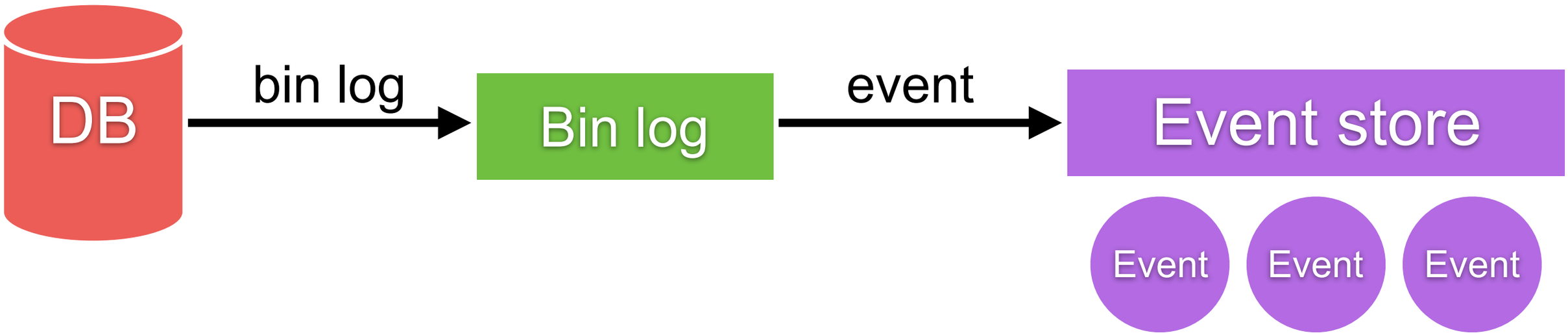
Keep events in order
Broadcast new event



With Event sourcing, we can **decouple services** (query and command)

Event sourcing with database

Working with database



Binlog or Binary log event is information about data modification made to database

Event sourcing

Duplication data (denormalize database)
Complexity (separate query and command)
Difficult to maintain

Event sourcing can’t solve all problems

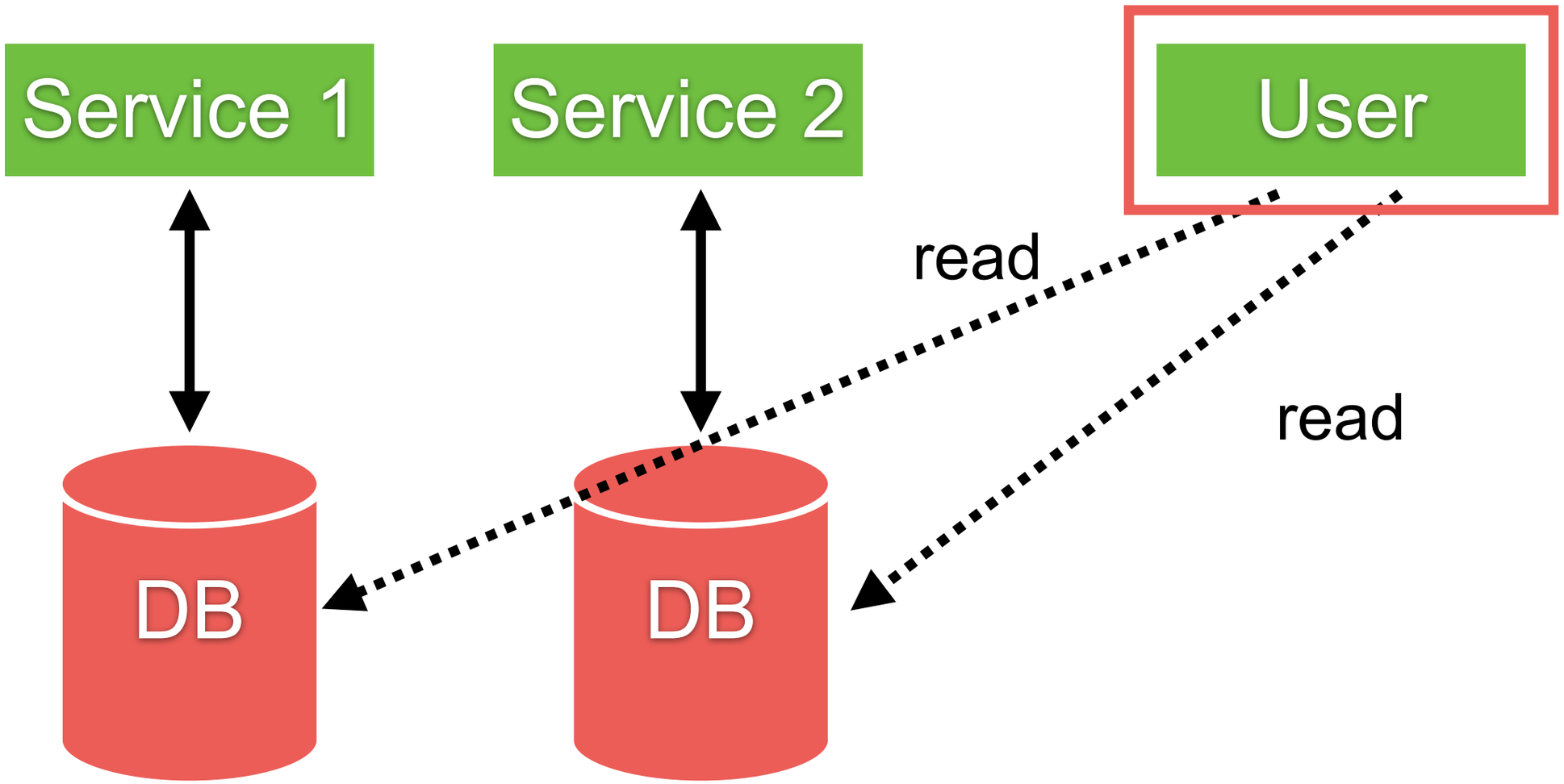
Start with **understand** your **purpose**

Start with understand your **problem** to resolve

How to query data in distributed system ?

Microservices

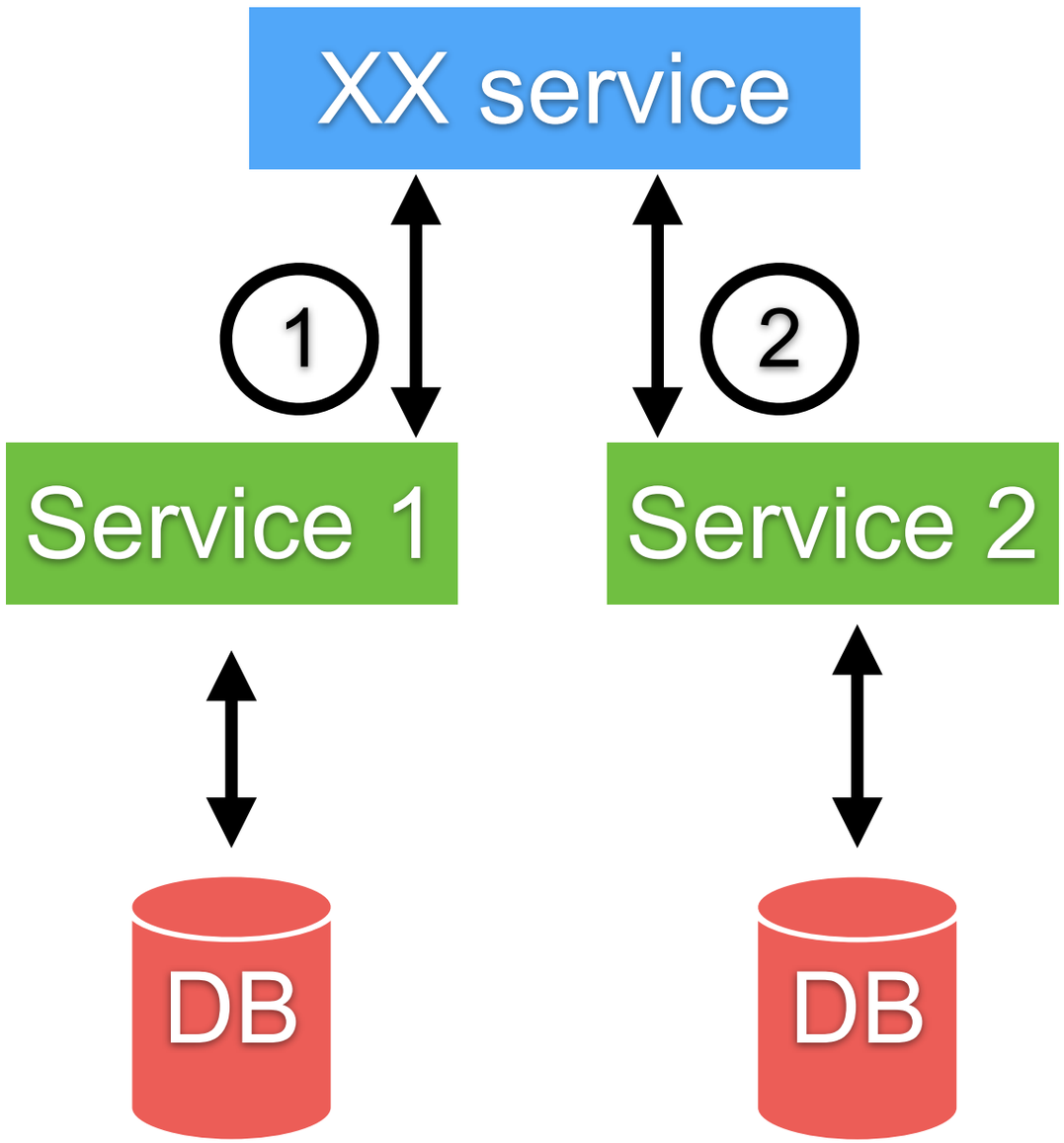
Separate database per service



API composition

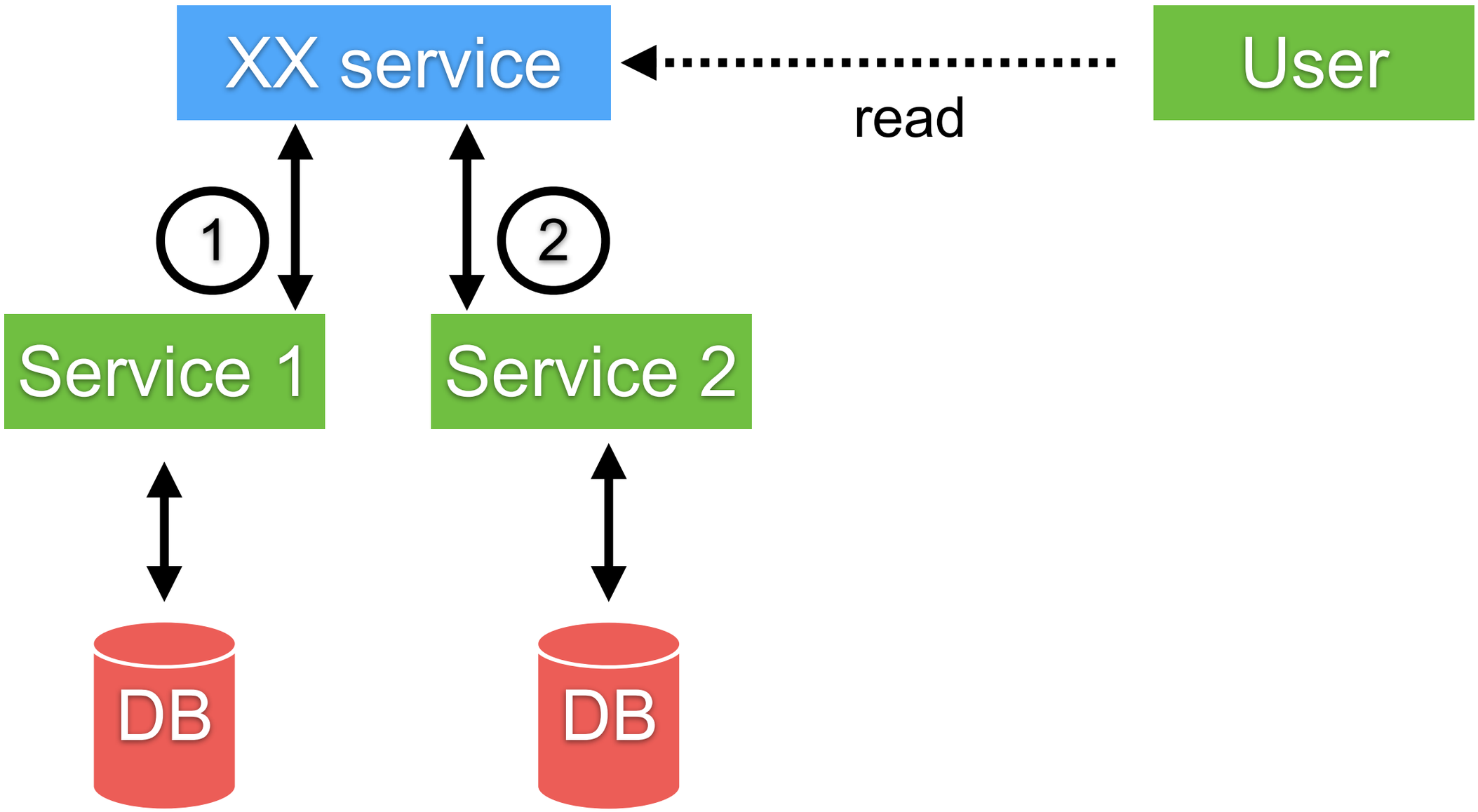
API composition

Simple solution for many use cases



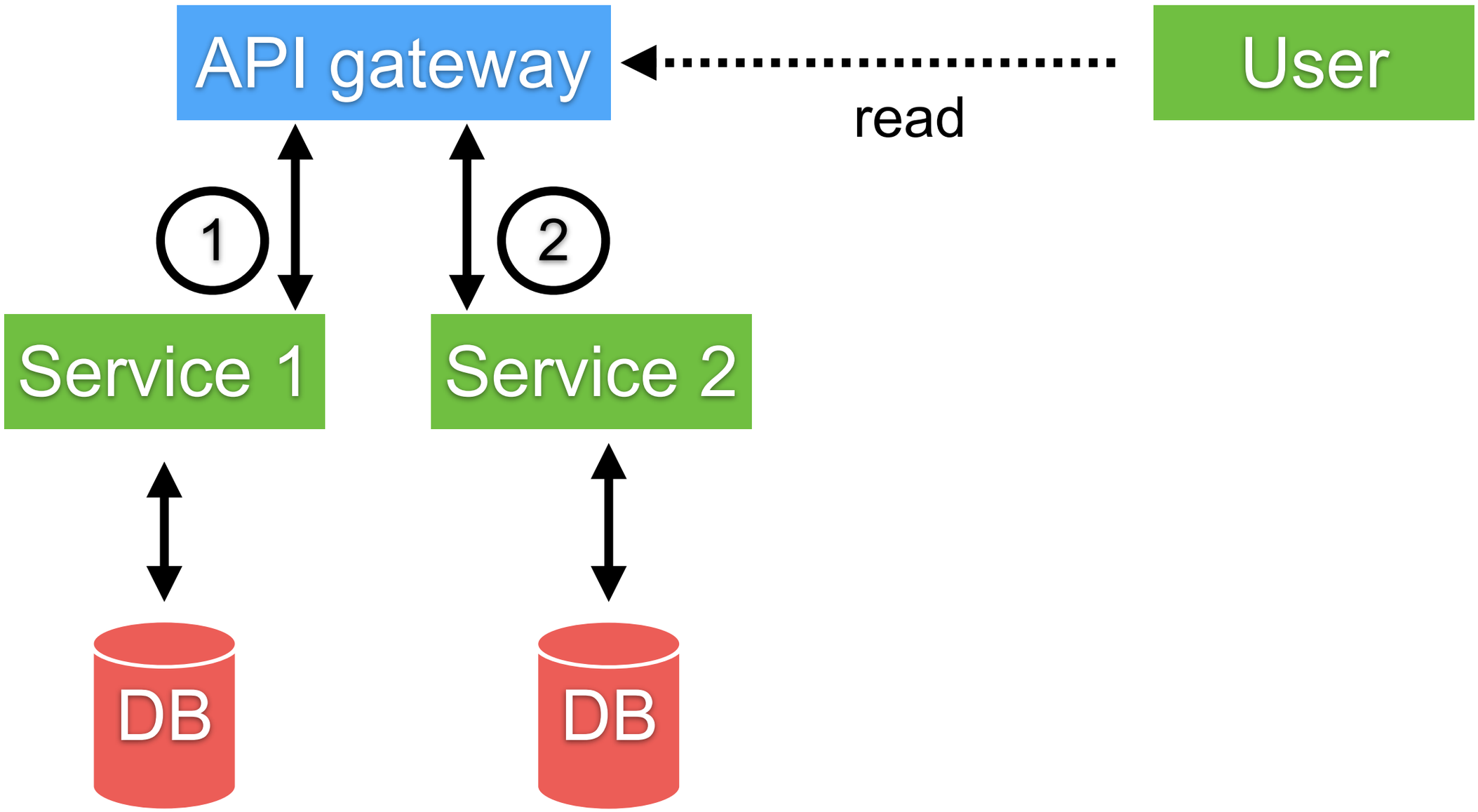
API composition

Simple solution for many use cases



API composition

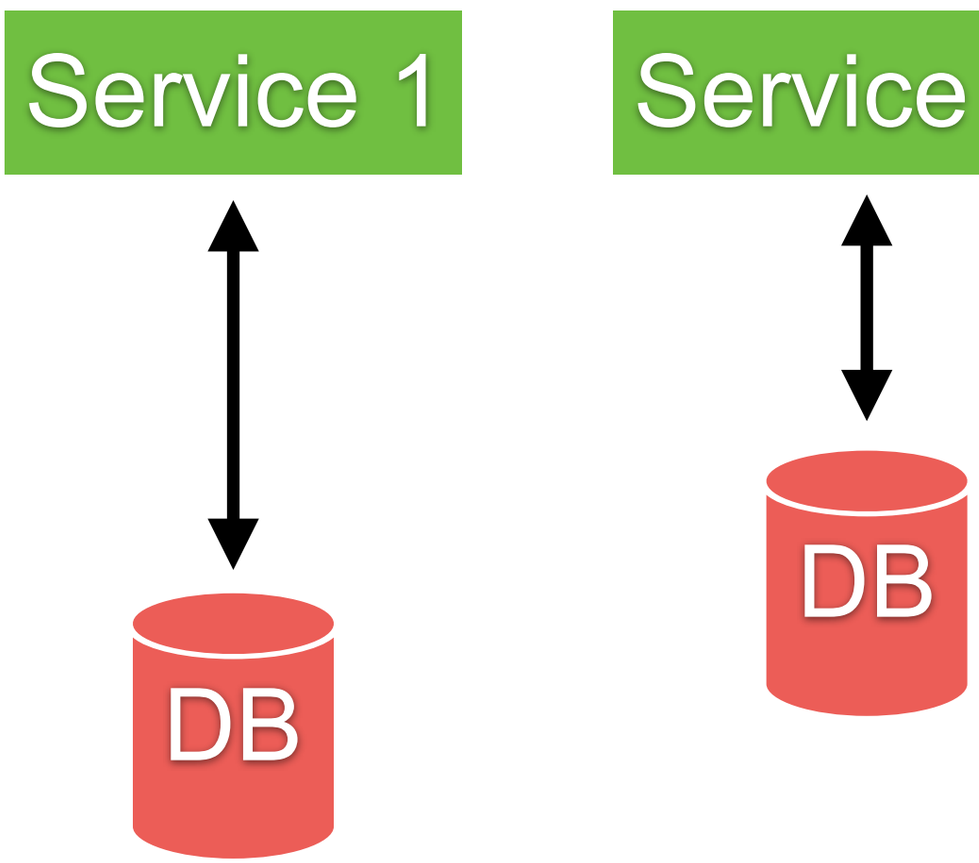
Simple solution for many use cases



Storing copy of data

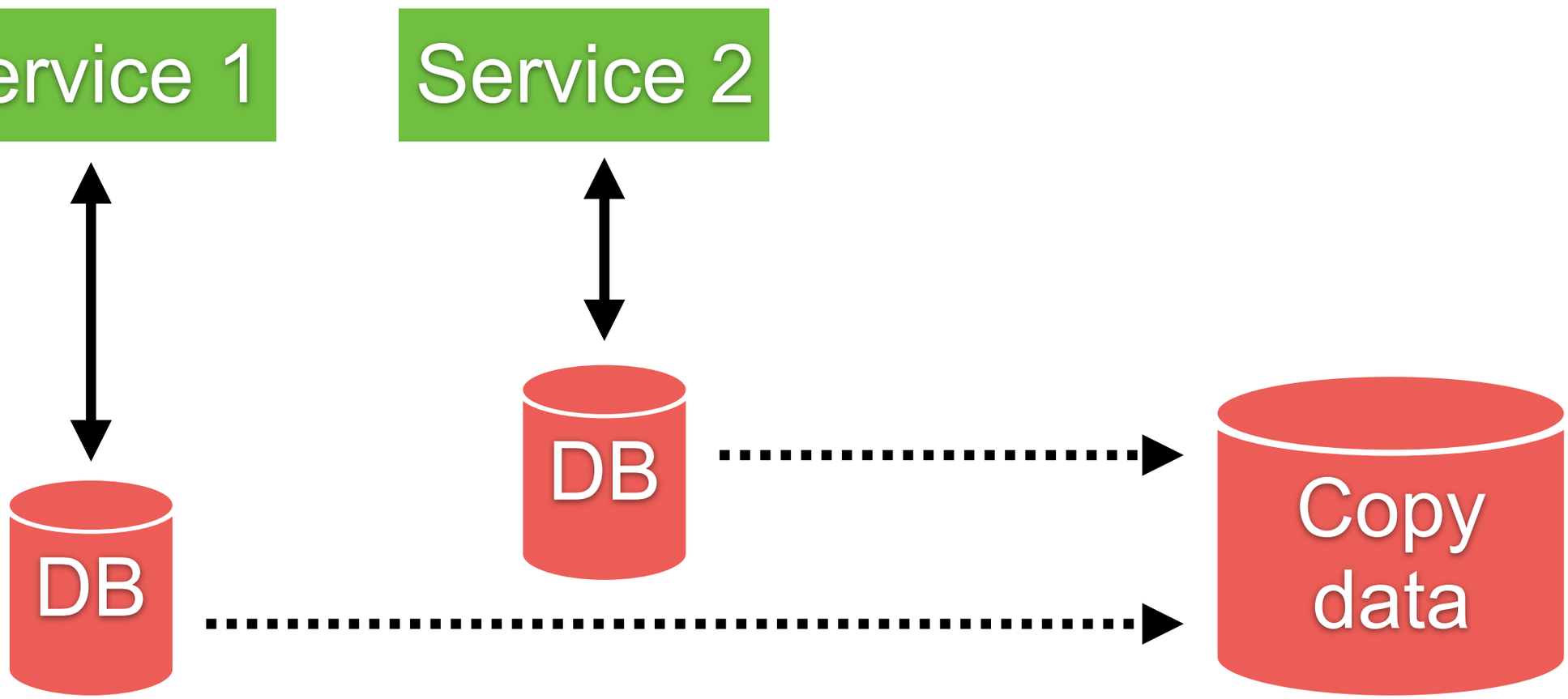
Storing copy data

Copy or caching data to other database



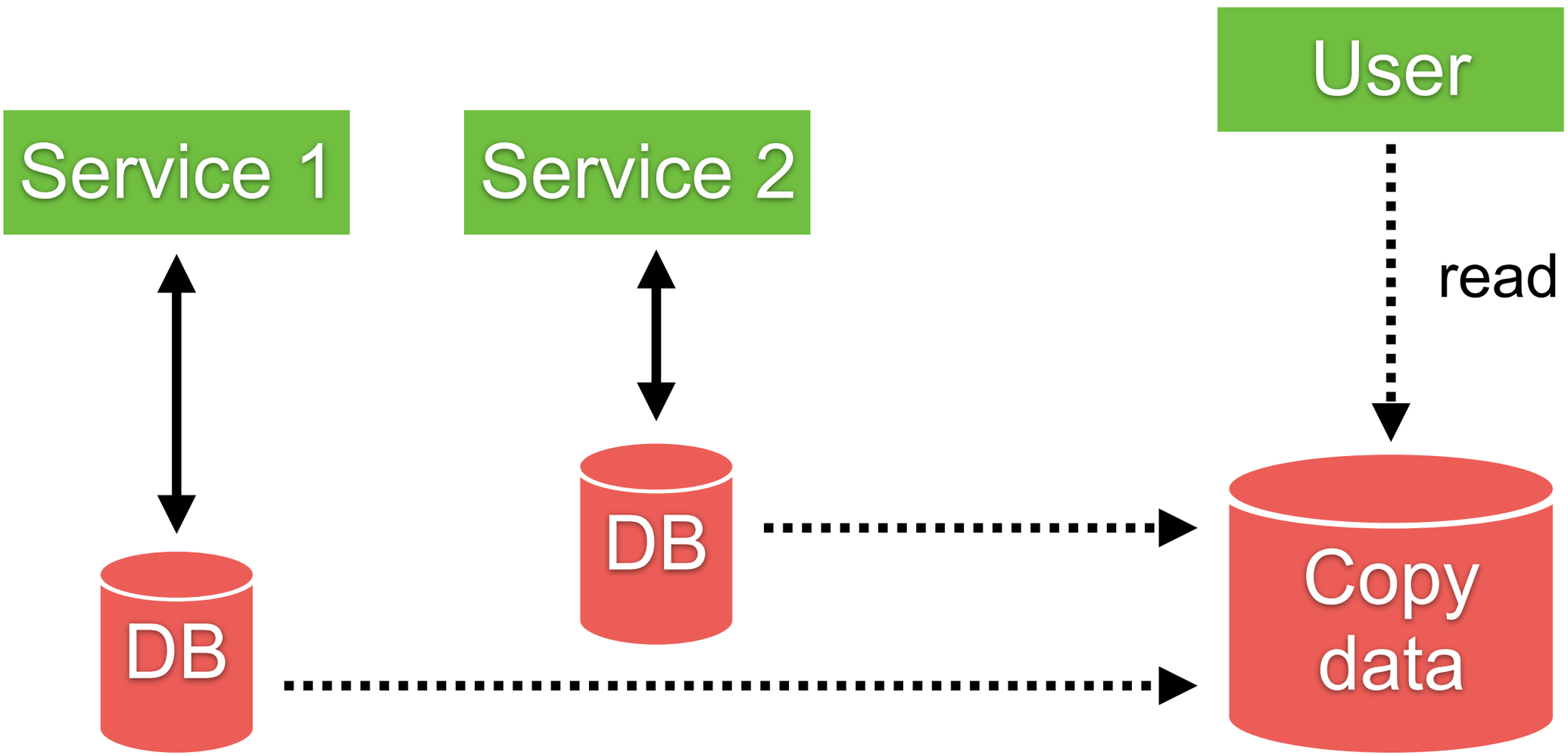
Storing copy data

Copy or caching data to other database



Storing copy data

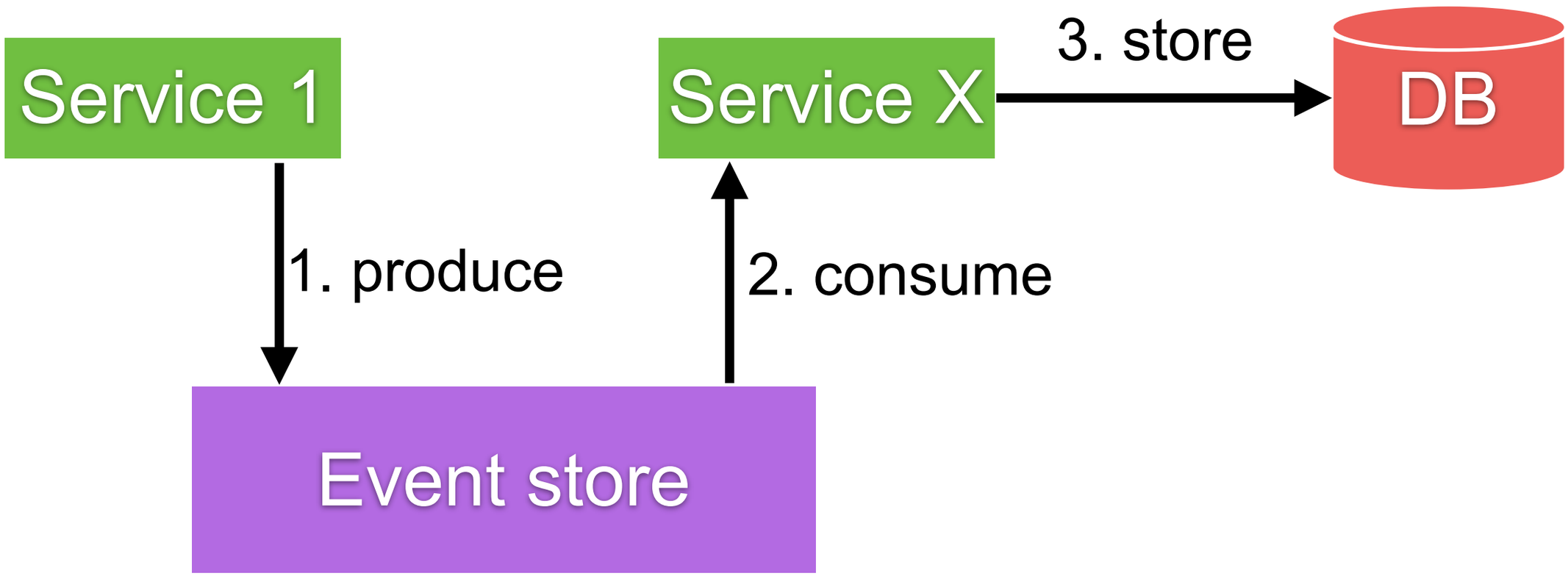
Copy or caching data to other database



Using event to store copy data

Storing copy data

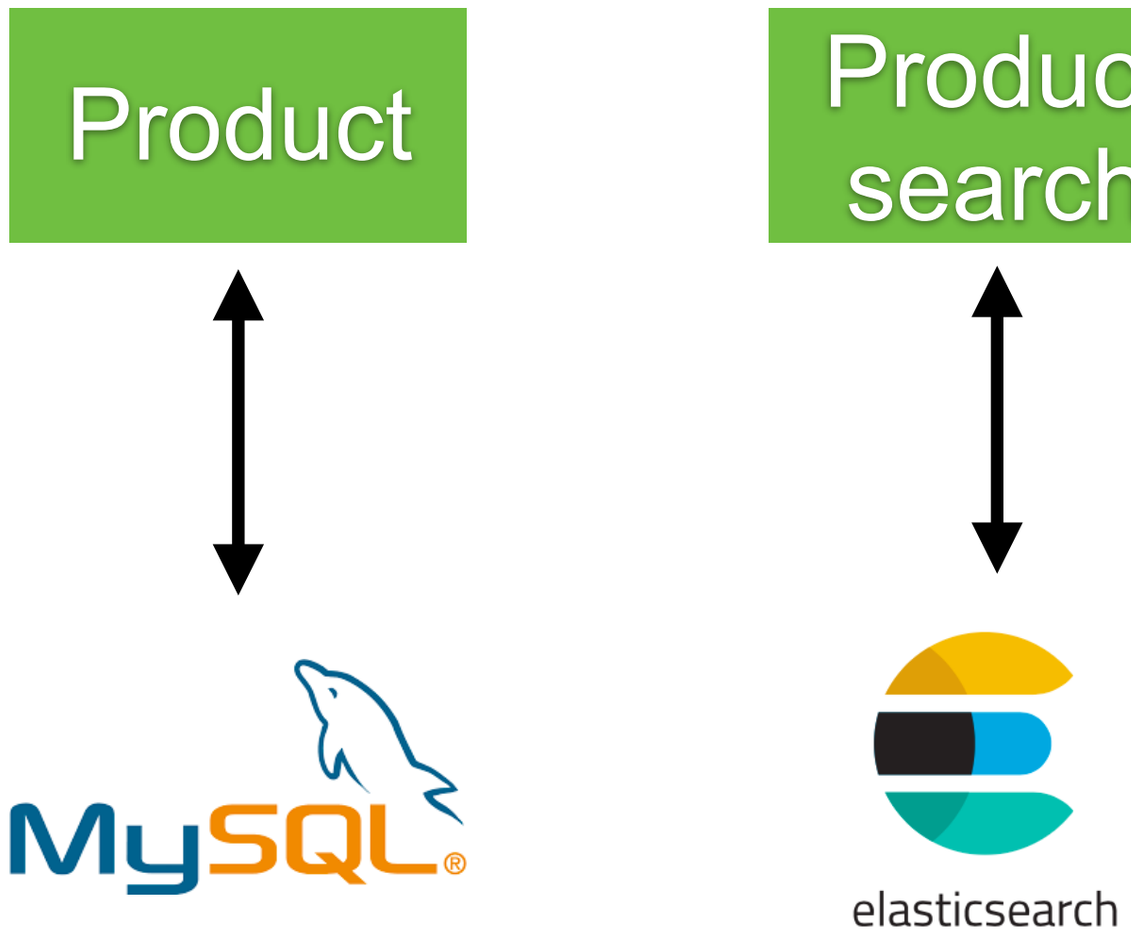
Publish event to store copy data



Separate query and command

Separate data for read and write

For example MySQL to write, Elasticsearch to search

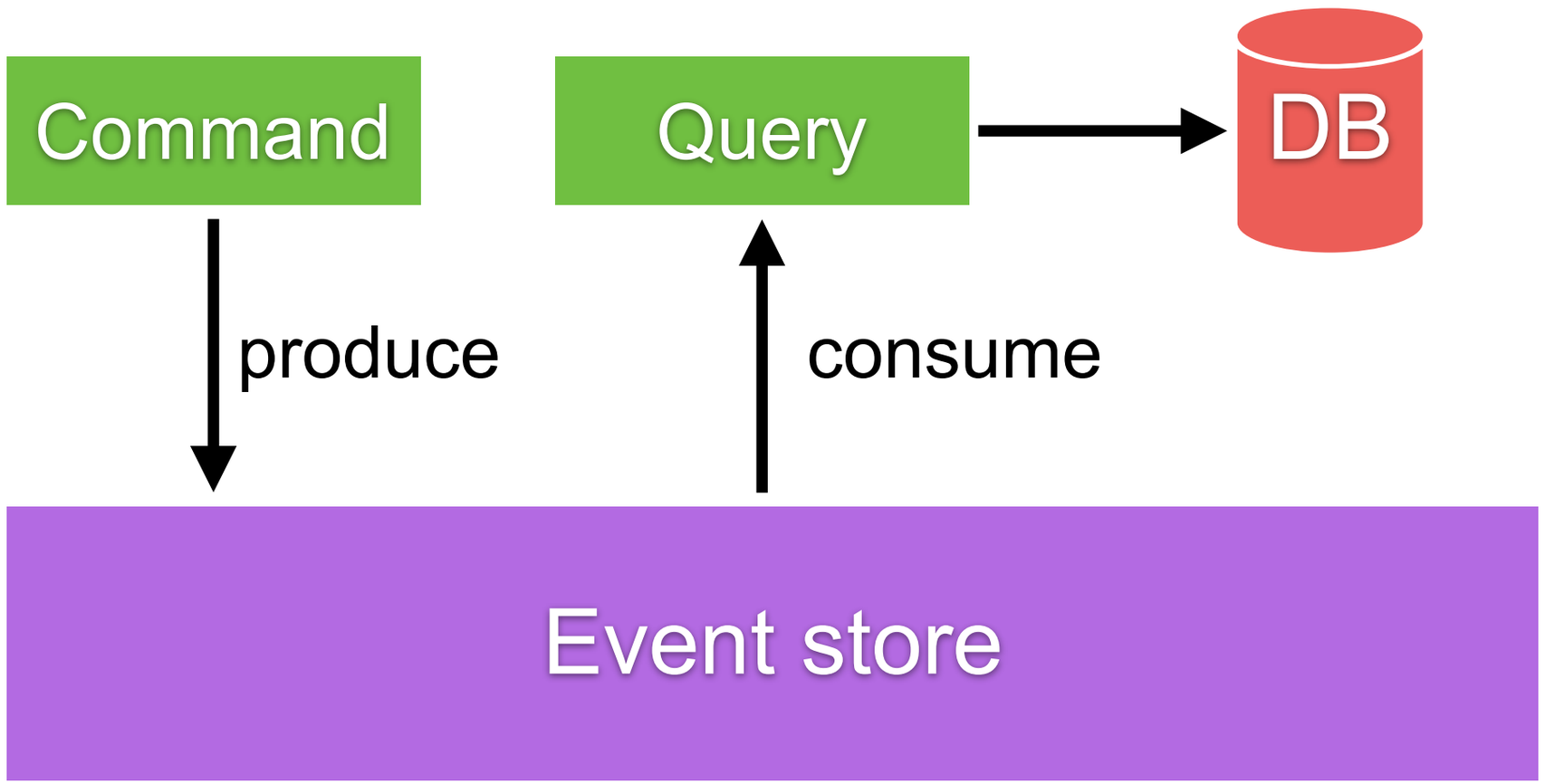


CQRS

Command **Q**uery **R**esponsibility **S**egregation

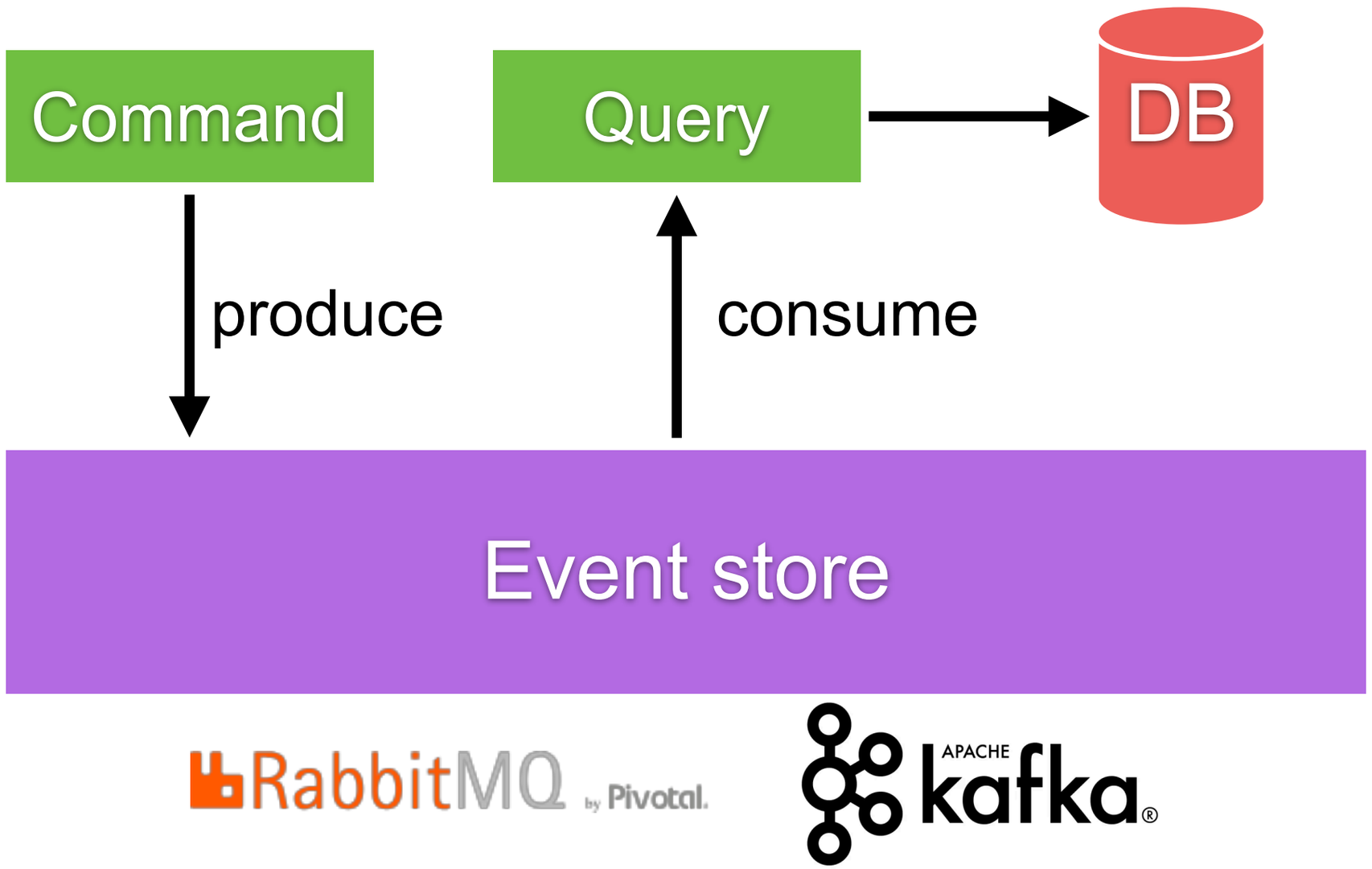
CQRS

Separate command and query services



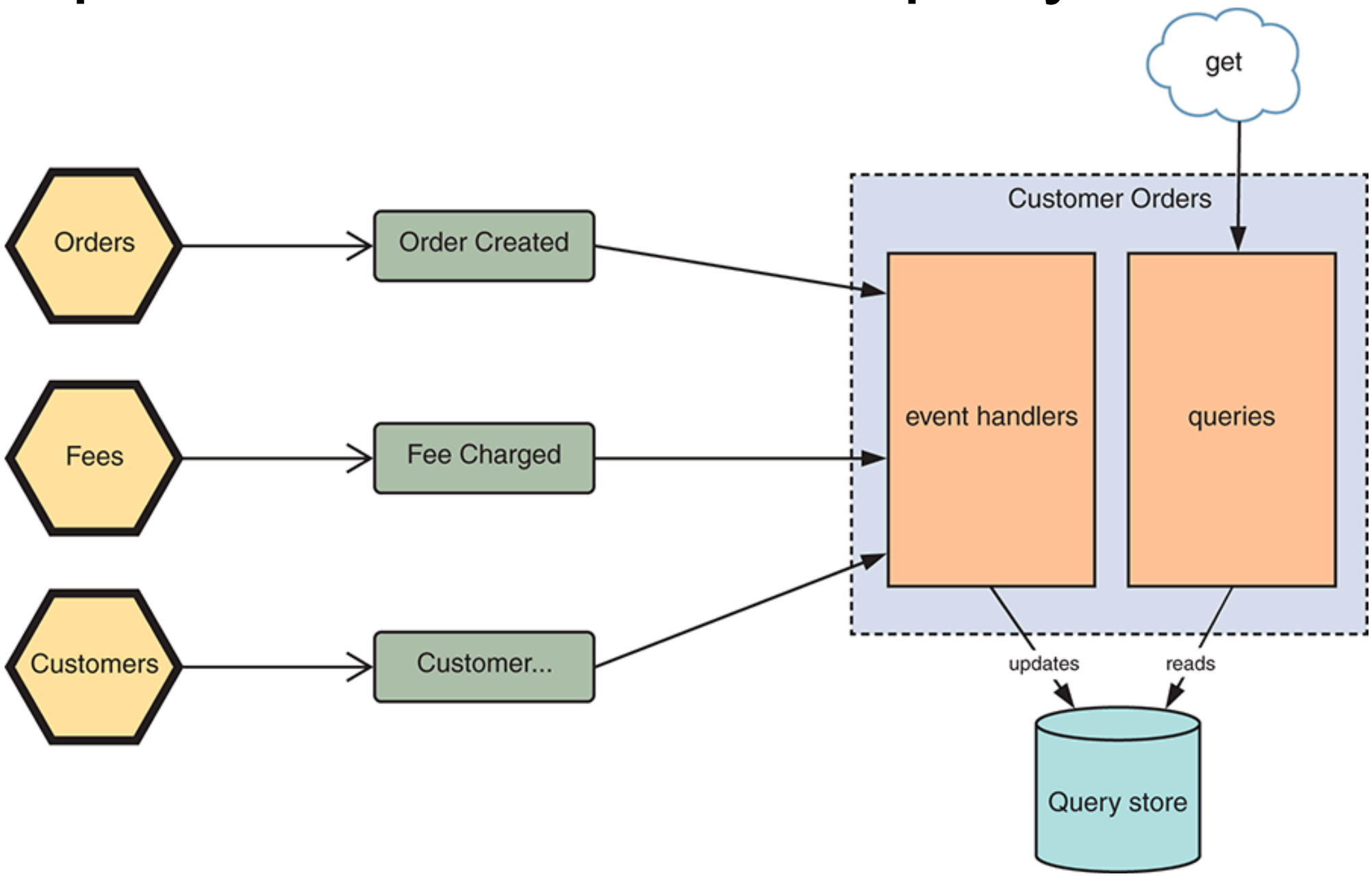
CQRS

Separate command and query services



CQRS

Separate command and query services



You don’t need to use only CQRS with you application