SQL Day

Distributed Monolith: Challenges and strategies in breaking Databases for Microservices

MAREK MAŚKO

SQL Day

16 edycja konferencji SQLDay

13-15 maja 2024, WROCŁAW + ONLINE

Data Community

partner platynowy

lingaro

partner złoty

DXC TECHNOLOGY

VOLVO

software one

partner srebrny

accenture

Nordcloud an IBM Company

Capgemini

TECHNOLOGY INNOVATION DATA KNOWLEDGE tidk

elitmind think bright

C&F

About the Author

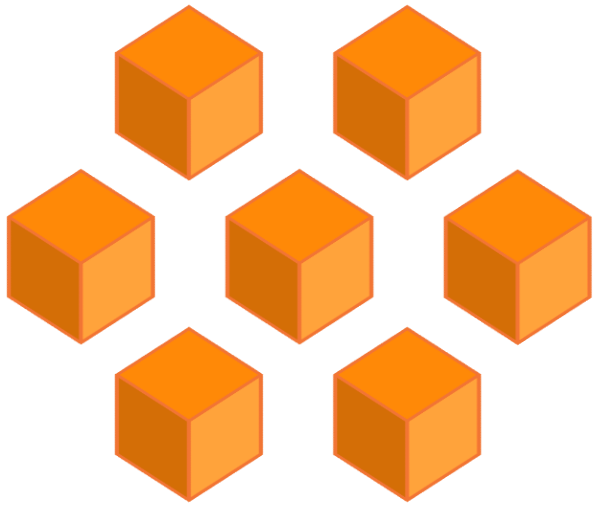
MAREK MAŚKO
Sr Engineering Manager
CAE Flight Services Poland

E-MAIL
MarekMasko@cae.com

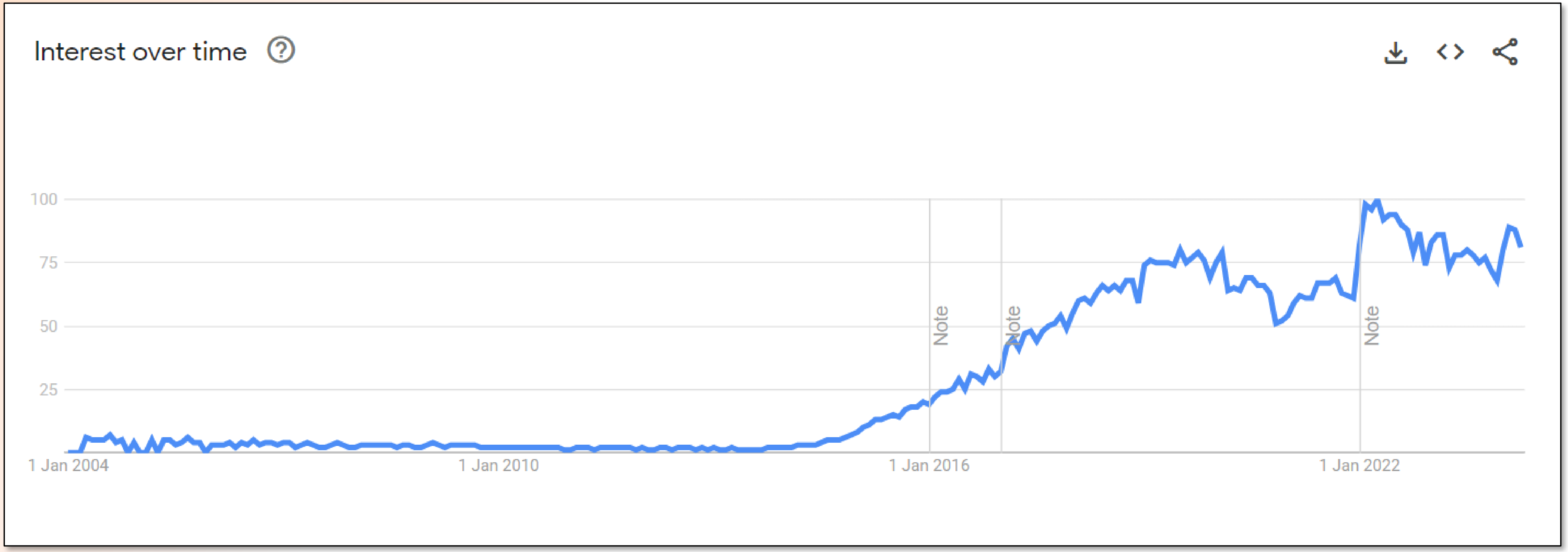
LINKEDIN
linkedin.com/in/MarekMasko/

TWITTER
@MarekMasko

Microservices



An unrelenting trend



Source: trends.google.com

What are microservices?

”

Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

”

Source: aws.amazon.com

What are microservices?

”

Microservices are an architectural approach to building applications where each core function, or service, is built and deployed independently.

Microservice architecture is distributed and loosely coupled, so one component’s failure won’t break the whole app. Independent components work together and communicate with well-defined API contracts.

”

Source: azure.microsoft.com

What are microservices?

”

Microservices (...) refers to an architectural style for developing applications. Microservices allow a large application to be separated into smaller independent parts, with each part having its own realm of responsibility.

To serve a single user request, a microservices-based application can call on many internal microservices to compose its response.

”

Source: cloud.google.com

What are microservices?

”

(...) **microservice** architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

”

Source: martinfowler.com

Small
Independent
Having own responsibility
Loosely coupled

What problems do they solve?

What problems do you want to solve?

Scalability?

Deliver faster and more often?

Too big codebase?

Embrace the new technology?

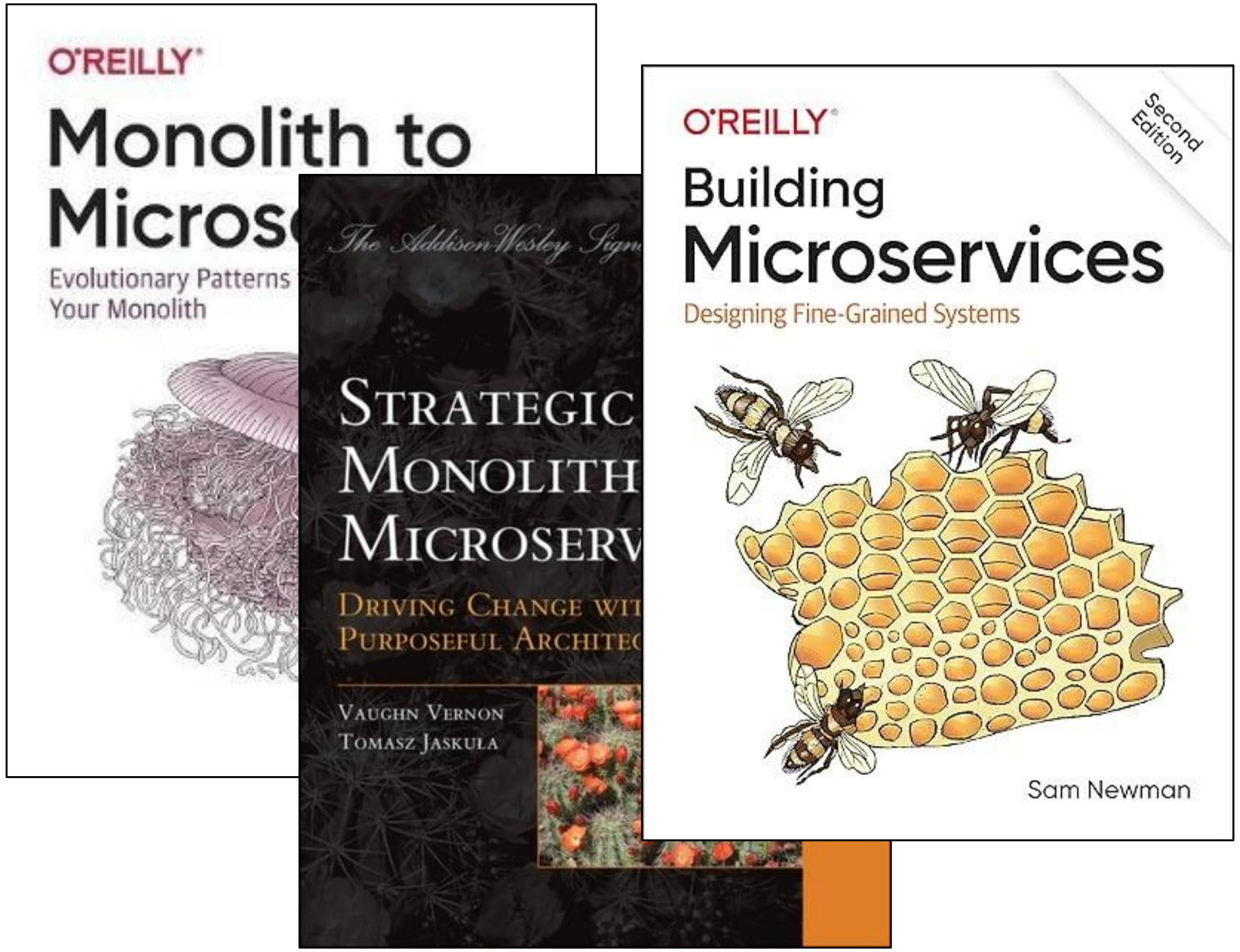
Are microservices a good choice for me?

I don’t know…

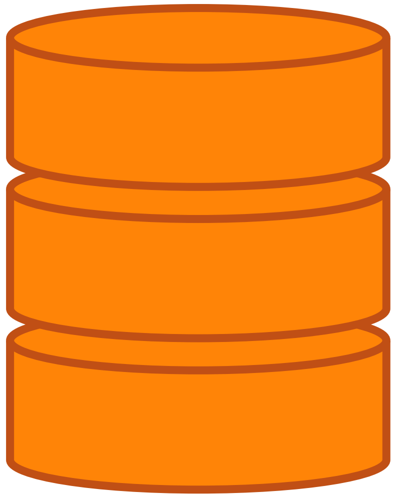
…and we won’t talk about that!

Application migration patterns

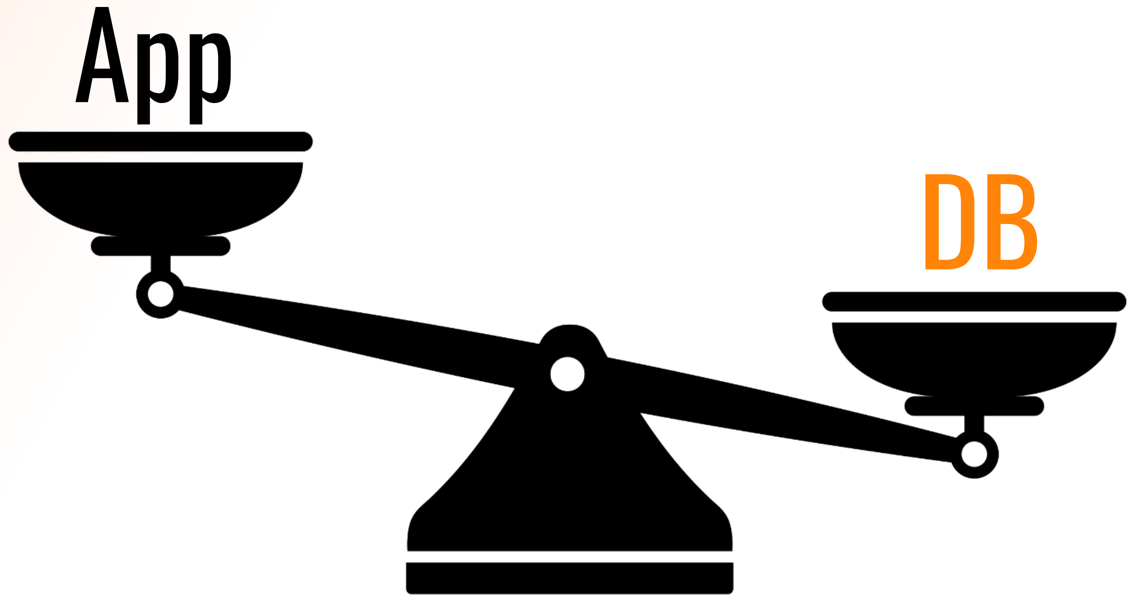
- Strangler Fig Application
- UI Composition
- Branch By Abstraction
- Parallel Run
- Decorating Collaborator
- Change Data Capture
- ...



Where is the database in all of this?



Databases are different...



...databases have state!

Microservice architecture style...

SQL Day

Microservice architecture style…

Microservices

Microservice architecture style...

Microservices ⇨ Stateless apps

Microservice architecture style…

Microservices ⇨ Stateless apps

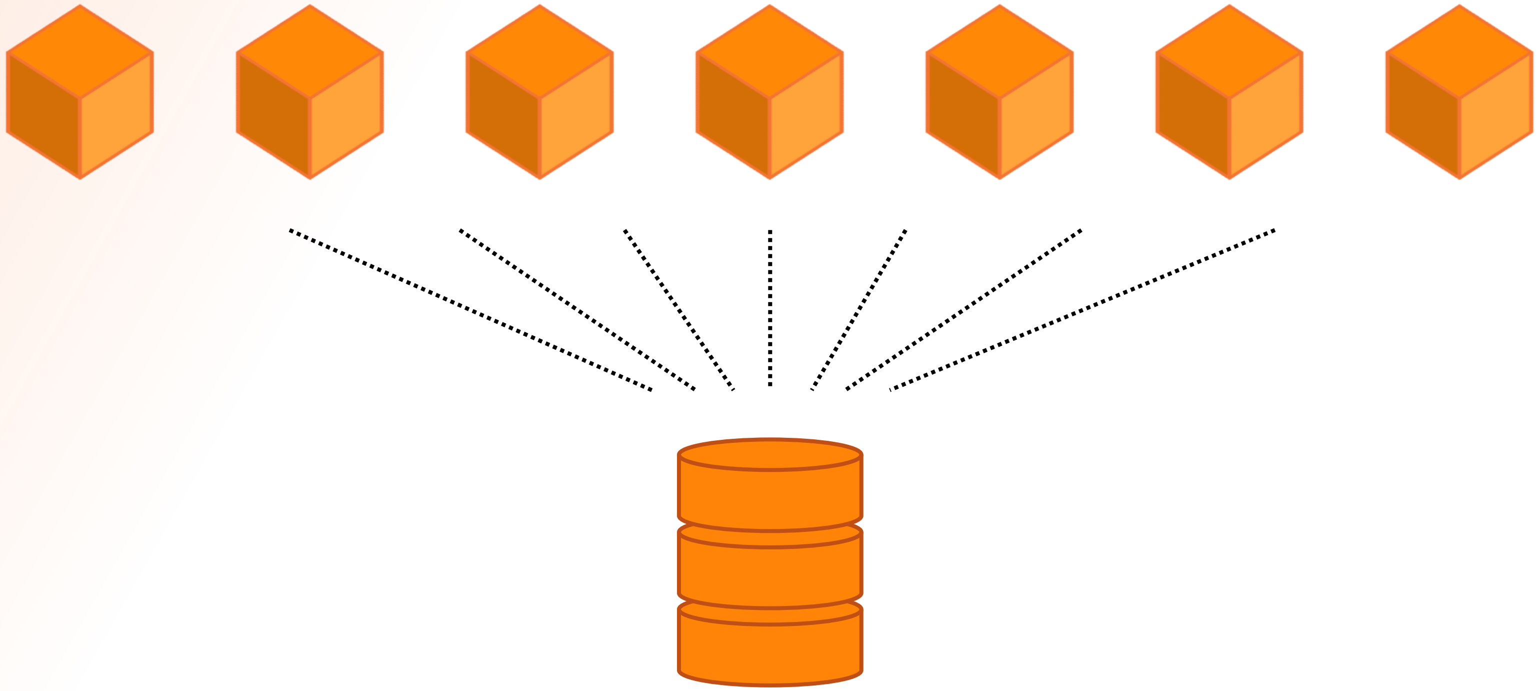
State

Microservice architecture style...

Microservices ⇨ Stateless apps

State ⇨ Data services

It’s just a shift of the problem



Distributed monolith

Decomposition of a database

You can put it off initially

But it can’t be delayed forever

Microservice characteristics

- Componentization via services
- Organized around business capabilities
- Products not projects
- Smart endpoints and dumb pipes
- Decentralized governance
- Decentralized data management
- Infrastructure automation
- Design for failure
- Evolutionary design

Source: martinfowler.com

Microservices should own their own data

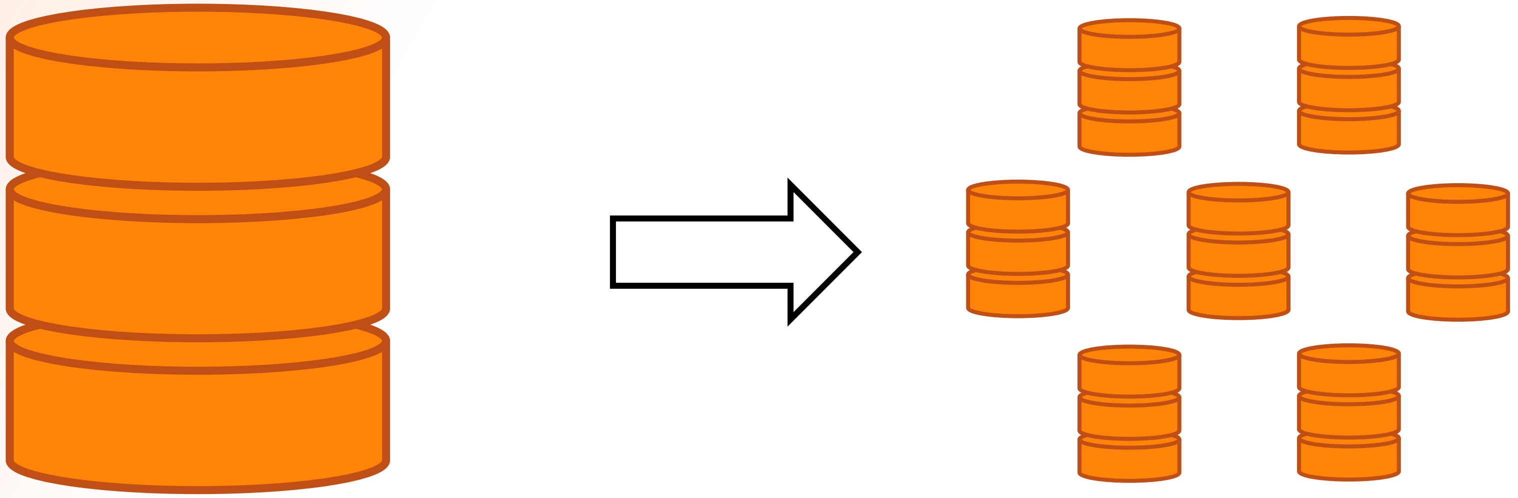
”

One of the things I see people having the hardest time with is the idea that microservices should not share databases.

If one service wants to access data held by another service, then it should go and ask that service for the data it needs.

”

Source: Book „Monolith to Microservices” by Sam Newman



Big bang or smooth migration?

SQL Day

App and DB at the same time

vs

Baby steps…

Zero-downtime migrations

Prerequisites

Treat DB code same as an App code:

- Version control system
- Versioning
- Automated deployment

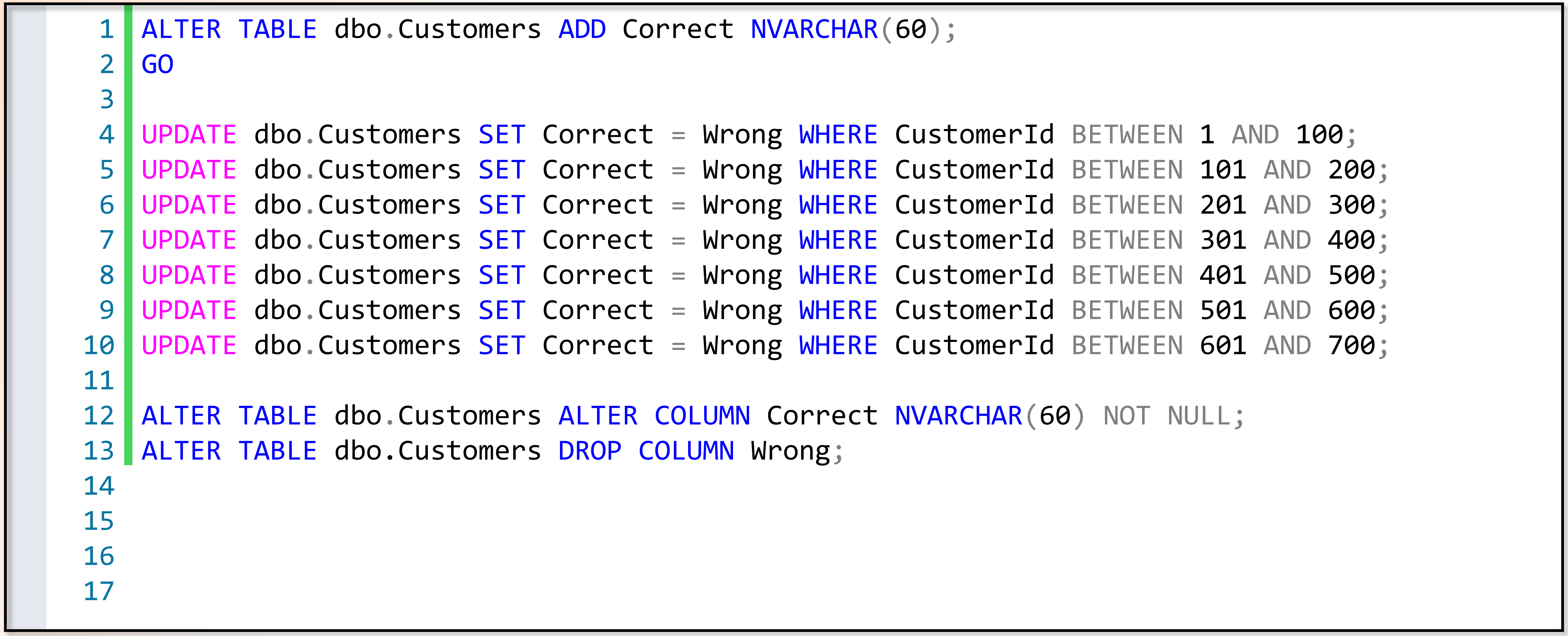
SQL Day

No data loss allowed!

Change column name

```sql
-- ALTER TABLE dbo.customers RENAME COLUMN wrong TO correct;
EXEC sp_rename 'dbo.Customers.Wrong', 'Correct', 'COLUMN';
```

Change column name



```sql
ALTER TABLE dbo.Customers ADD Correct NVARCHAR(60);
GO

UPDATE dbo.Customers SET Correct = Wrong WHERE CustomerId BETWEEN 1 AND 100;
UPDATE dbo.Customers SET Correct = Wrong WHERE CustomerId BETWEEN 101 AND 200;
UPDATE dbo.Customers SET Correct = Wrong WHERE CustomerId BETWEEN 201 AND 300;
UPDATE dbo.Customers SET Correct = Wrong WHERE CustomerId BETWEEN 301 AND 400;
UPDATE dbo.Customers SET Correct = Wrong WHERE CustomerId BETWEEN 401 AND 500;
UPDATE dbo.Customers SET Correct = Wrong WHERE CustomerId BETWEEN 501 AND 600;
UPDATE dbo.Customers SET Correct = Wrong WHERE CustomerId BETWEEN 601 AND 700;

ALTER TABLE dbo.Customers ALTER COLUMN Correct NVARCHAR(60) NOT NULL;
ALTER TABLE dbo.Customers DROP COLUMN Wrong;
```

Remember

Rehearse your migrations

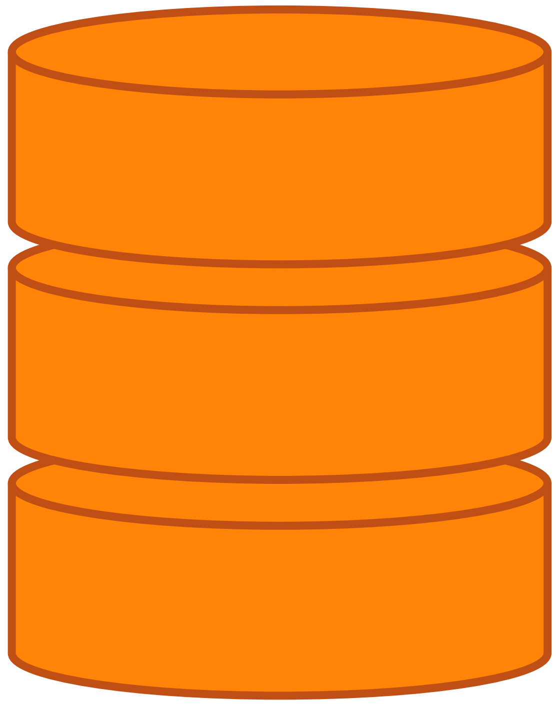
and

Check your data between migrations steps

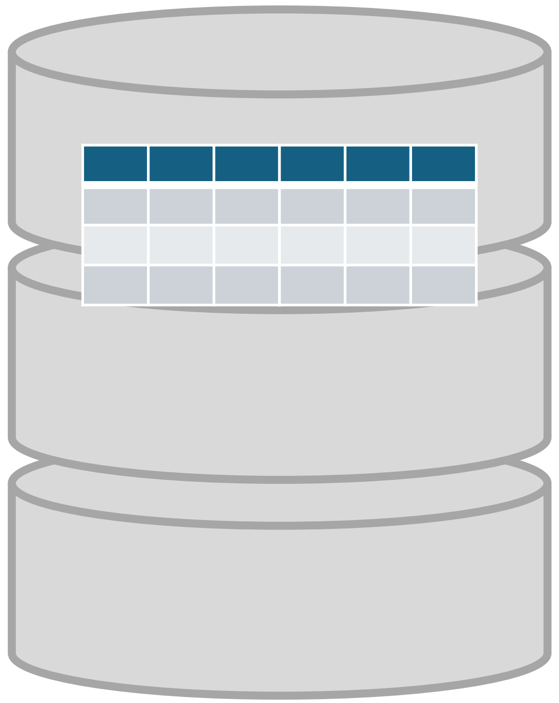
A few things to consider

Logical vs Physical decomposition

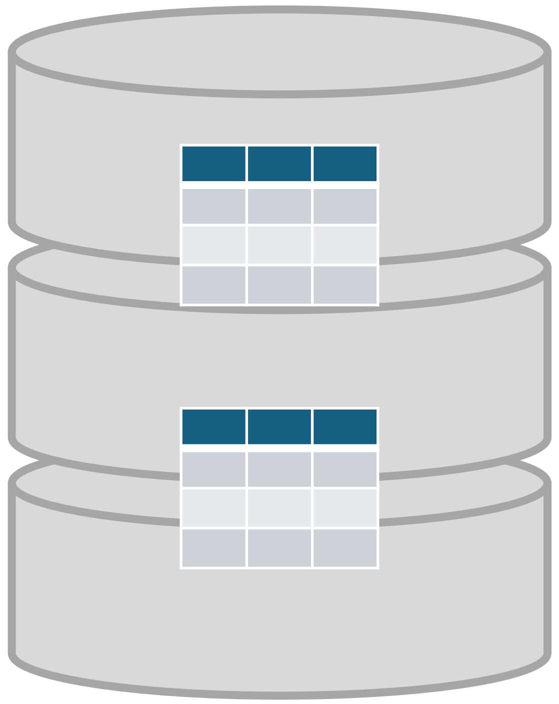
Logical vs Physical decomposition



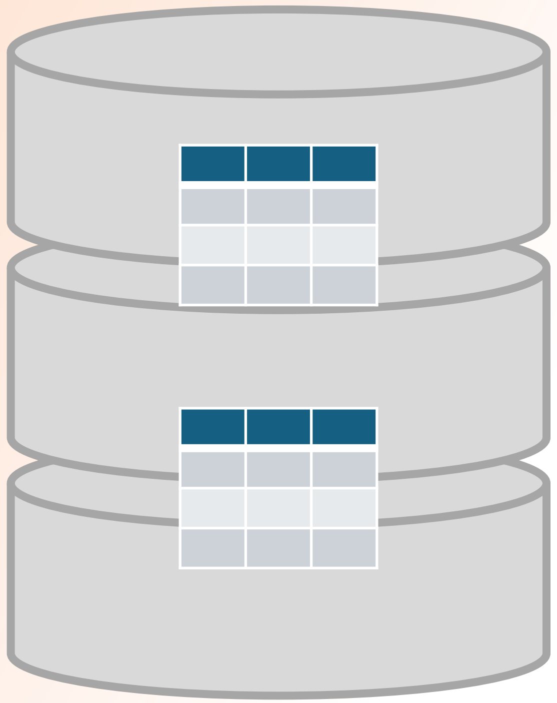
Logical vs Physical decomposition

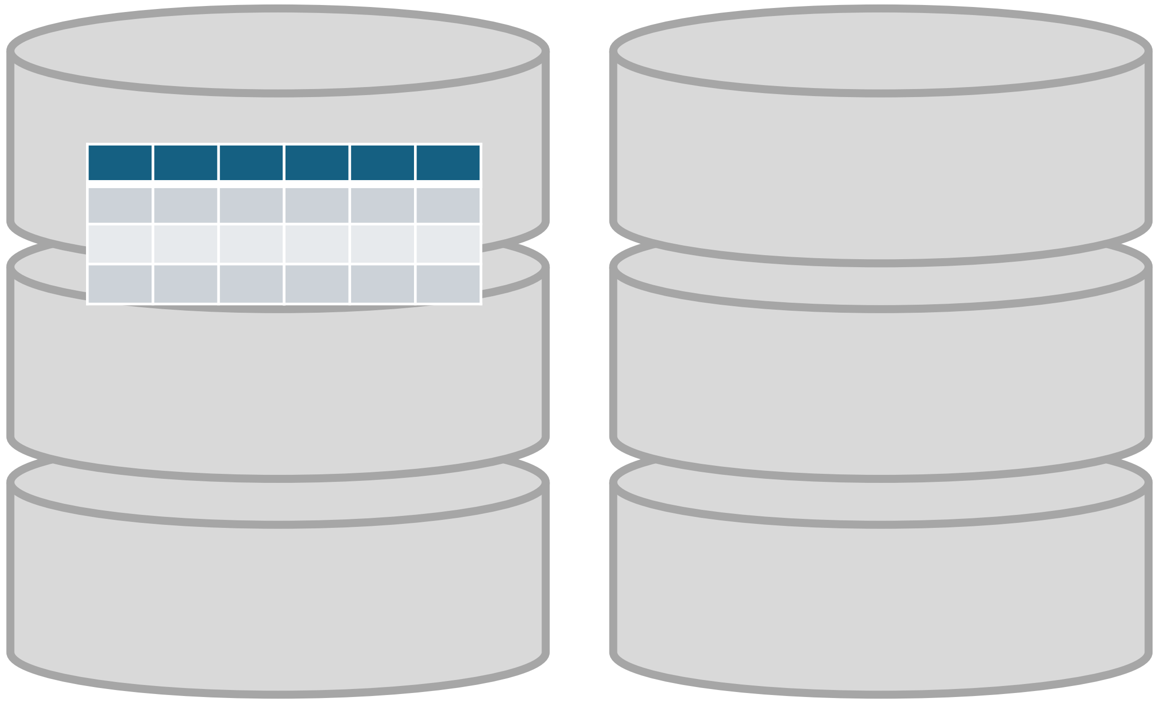


Logical vs Physical decomposition

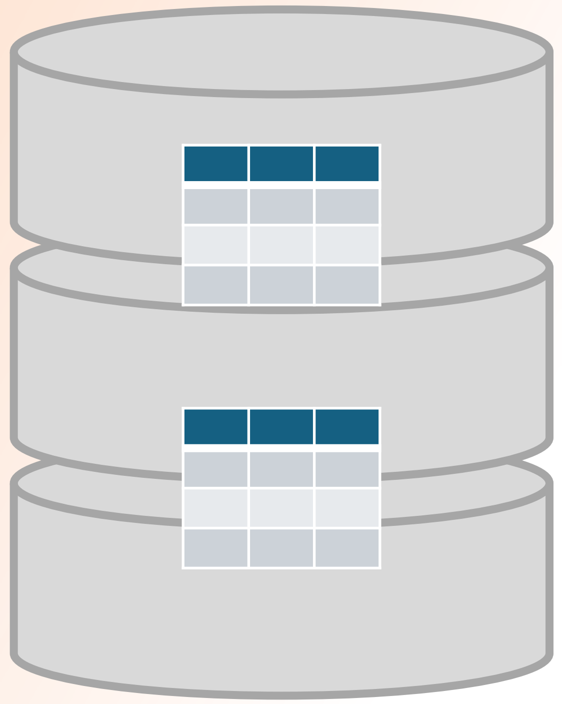


Logical vs Physical decomposition

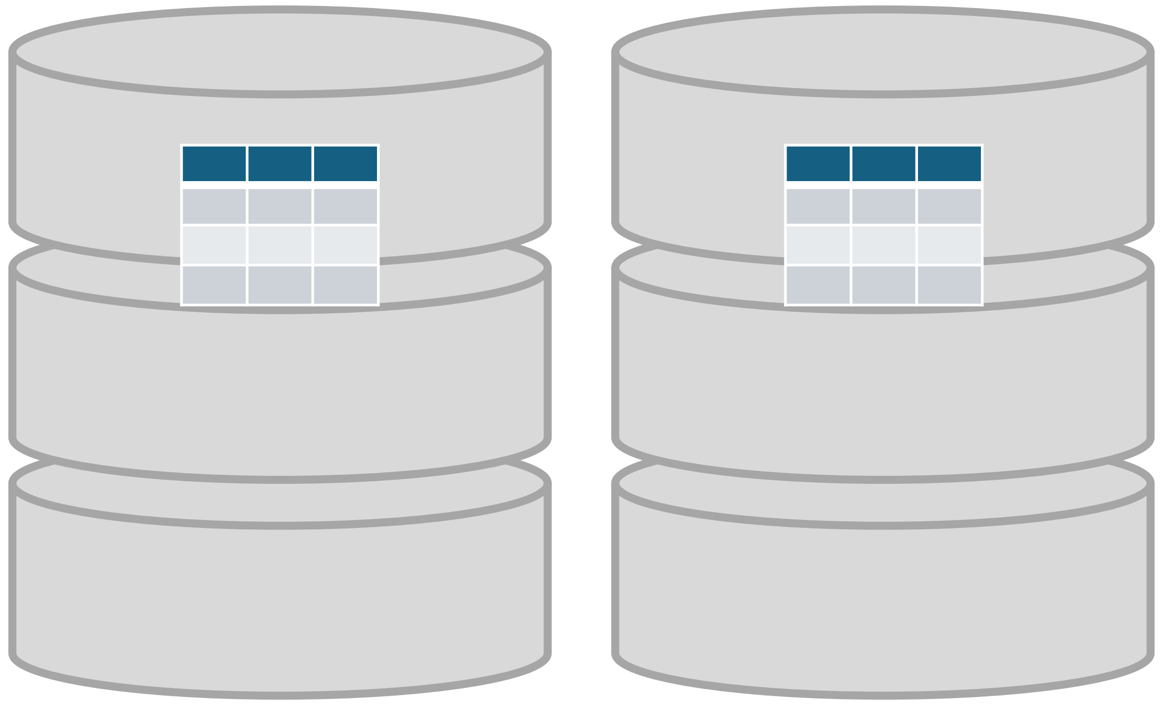




Logical vs Physical decomposition

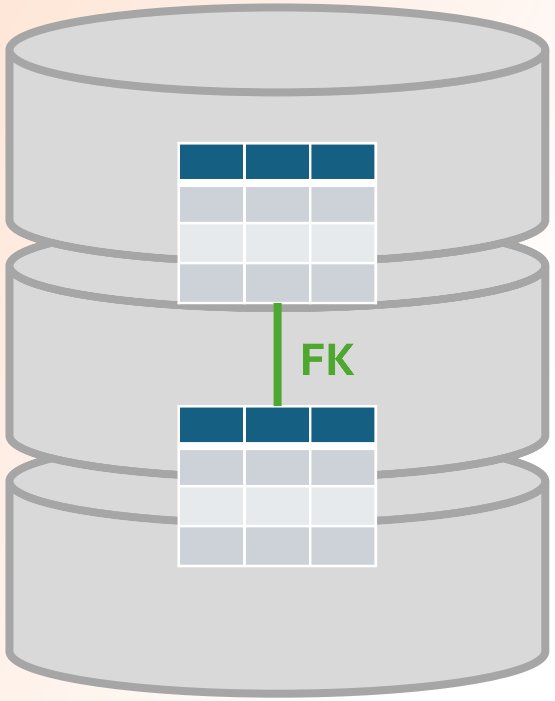


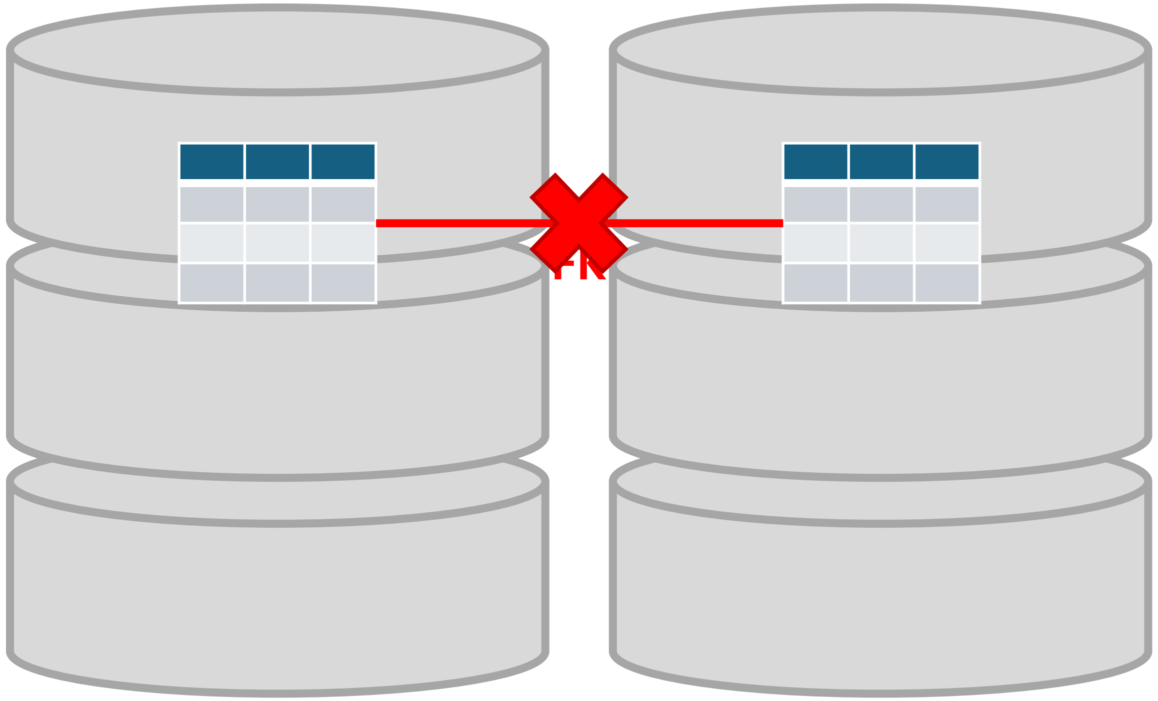
Logical decomposition



Physical decomposition

Referential Integrity

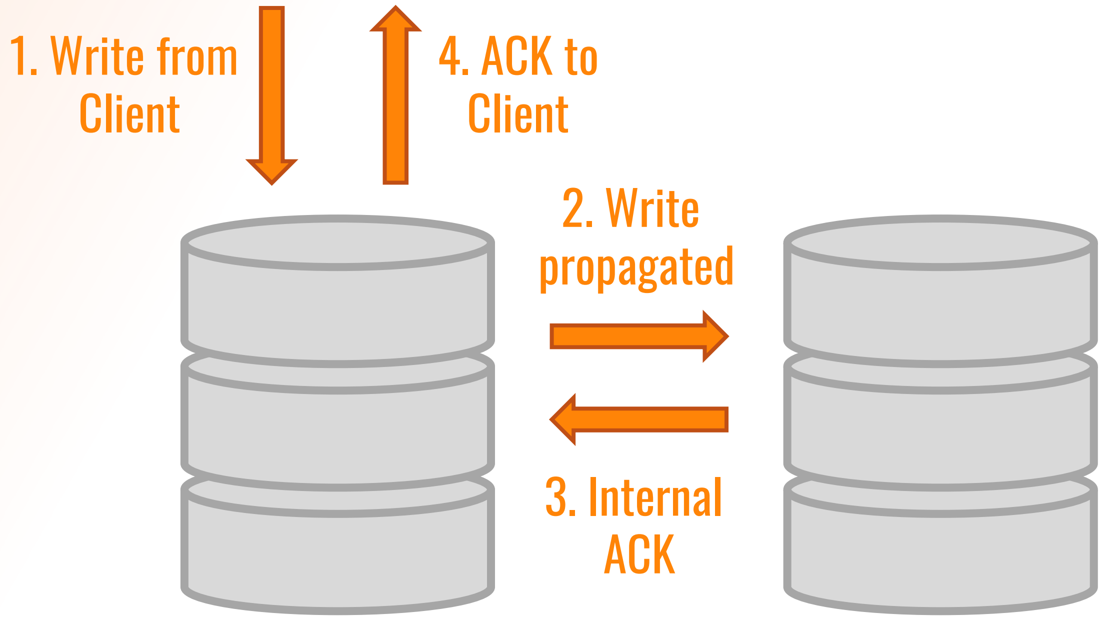




Move FK relationship to the App code

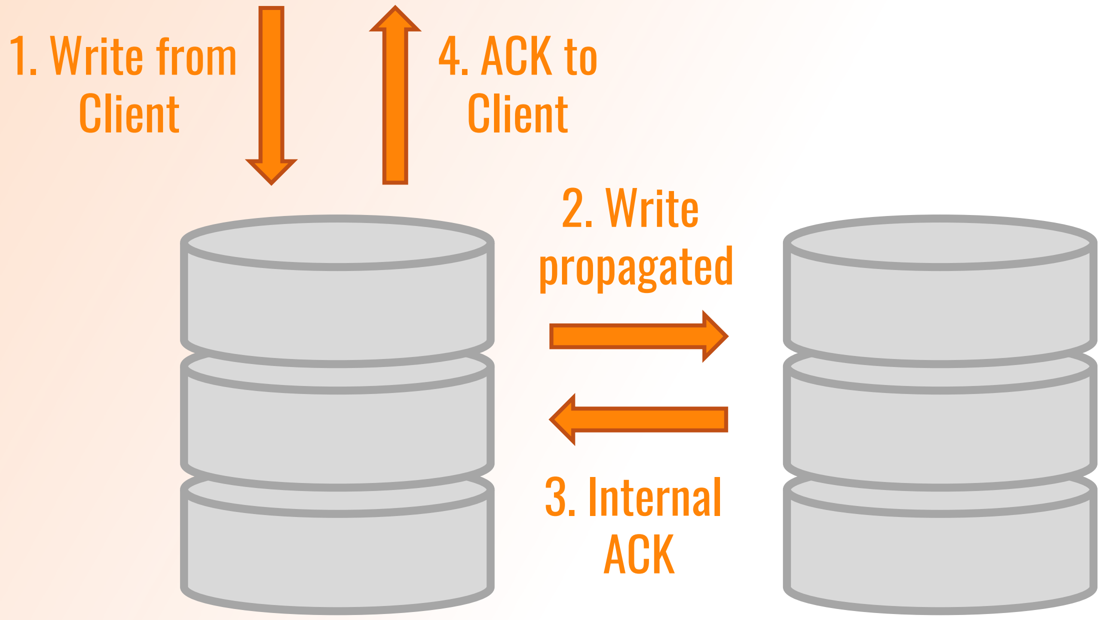
Move from Strong Consistency to Eventual Consistency

Move from Strong Consistency to Eventual Consistency

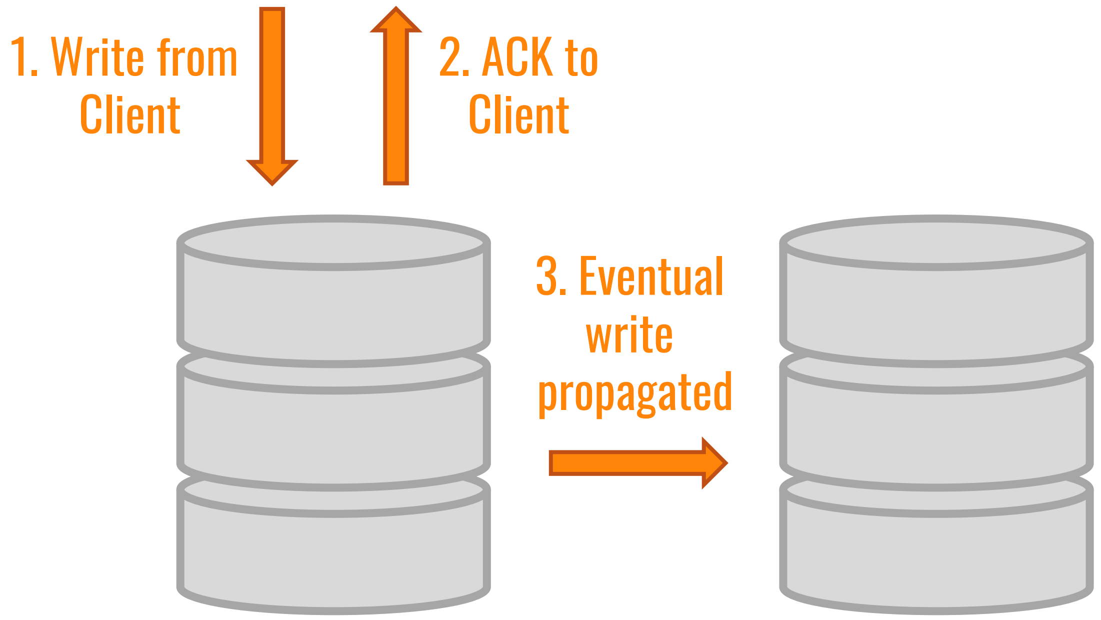


Strong Consistency

Move from Strong Consistency to Eventual Consistency

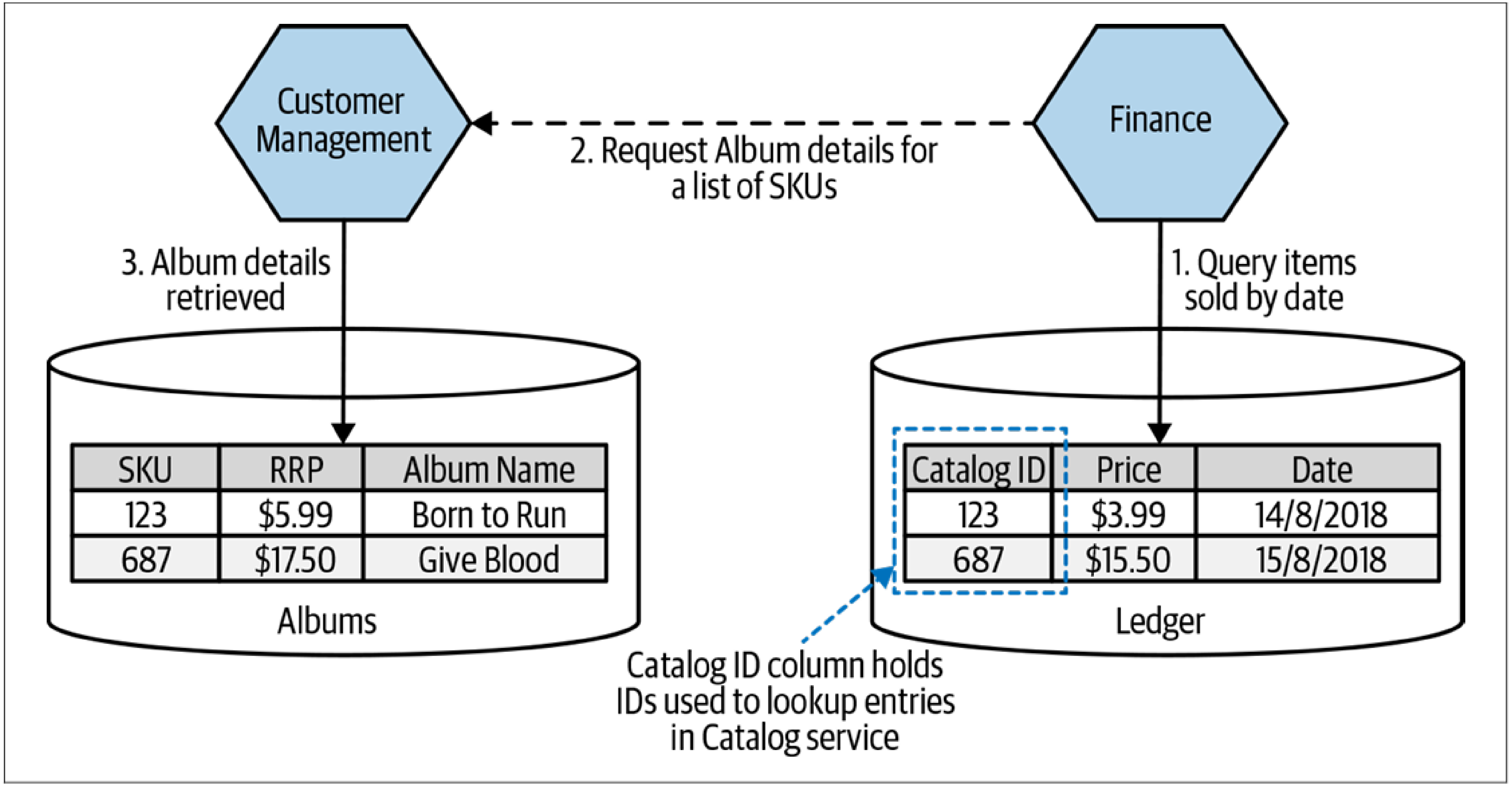


Strong Consistency



Eventual Consistency

Data consistency – Deleting data



Customer Management
Finance
2. Request Album details for a list of SKUs
3. Album details retrieved
1. Query items sold by date
Albums
Ledger
Catalog ID column holds IDs used to lookup entries in Catalog service

SKU	RRP	Album Name
123	$5.99	Born to Run
687	$17.50	Give Blood

Catalog ID	Price	Date
123	$3.99	14/8/2018
687	$15.50	15/8/2018

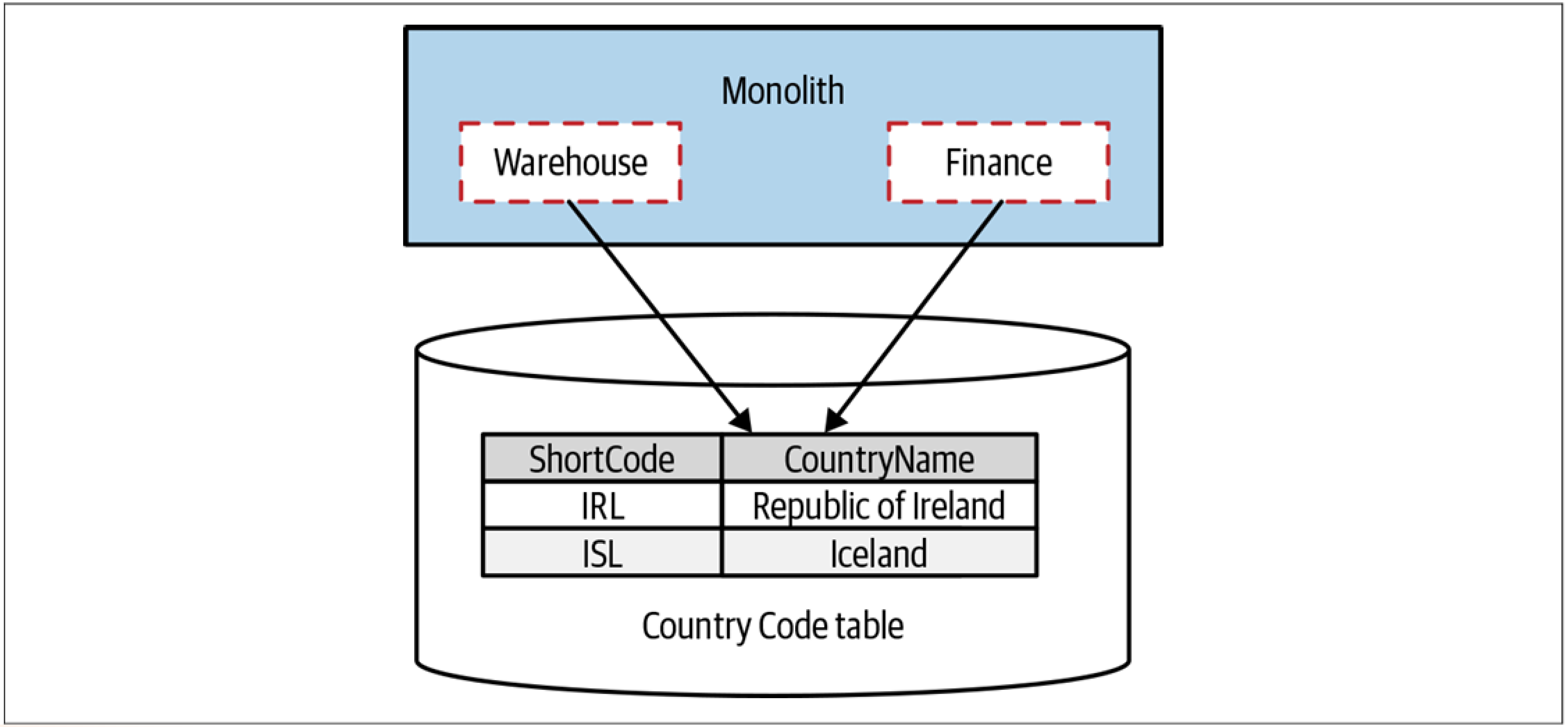
Data consistency – Deleting data

- Check before deletion
  - Stop new references being created – locking
  - Reverse dependencies
- Handle deletion gracefully
  - „Information not available”
  - Not found vs Gone
  - Subscribe to deletion event and copy data locally
- Don’t allow deletion
  - Soft delete

What about recovering the DB to an earlier state?!

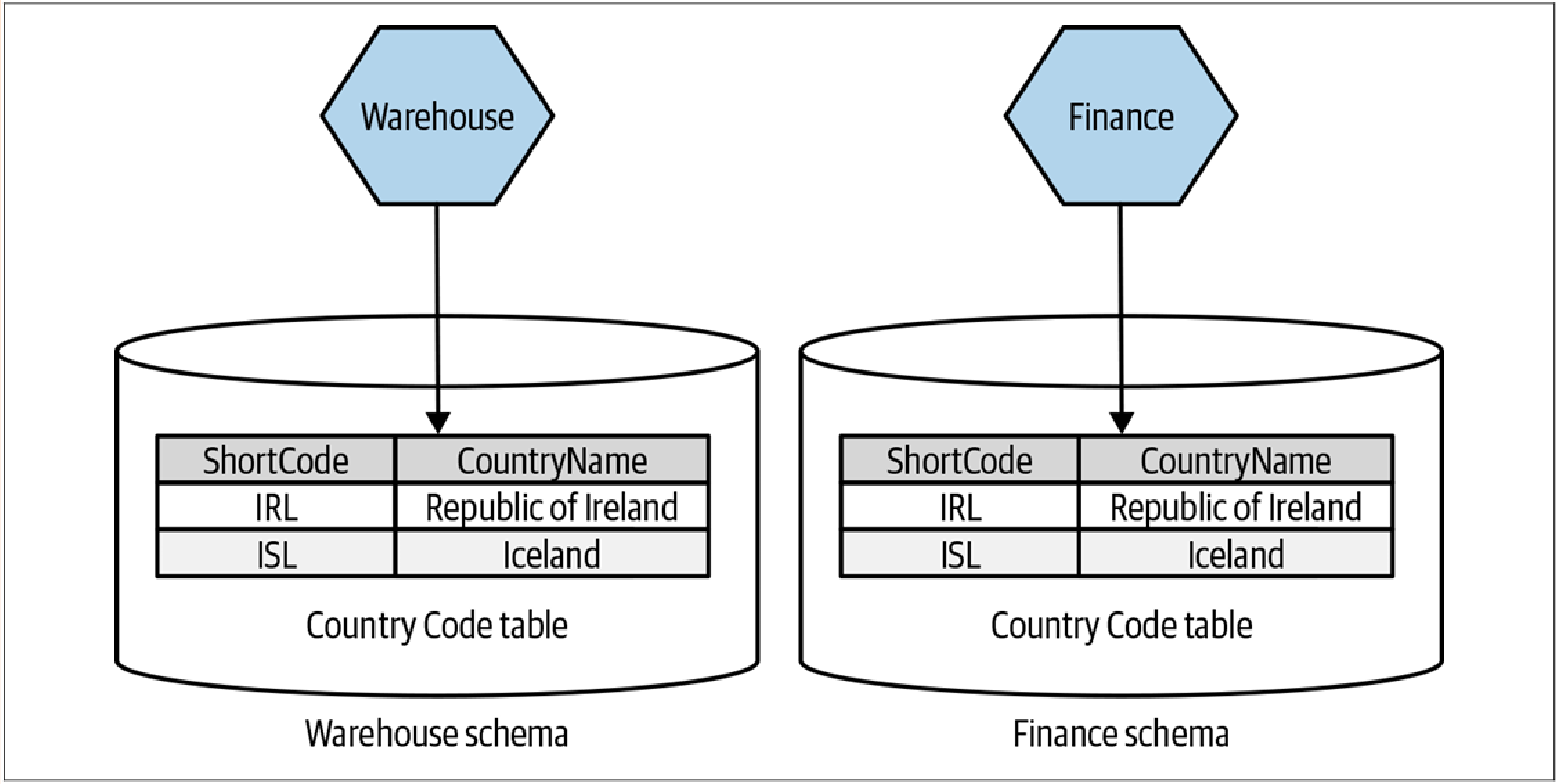
We created an issue that doesn’t exists in the monolithic system

Shared static data



ShortCode	CountryName
IRL	Republic of Ireland
ISL	Iceland

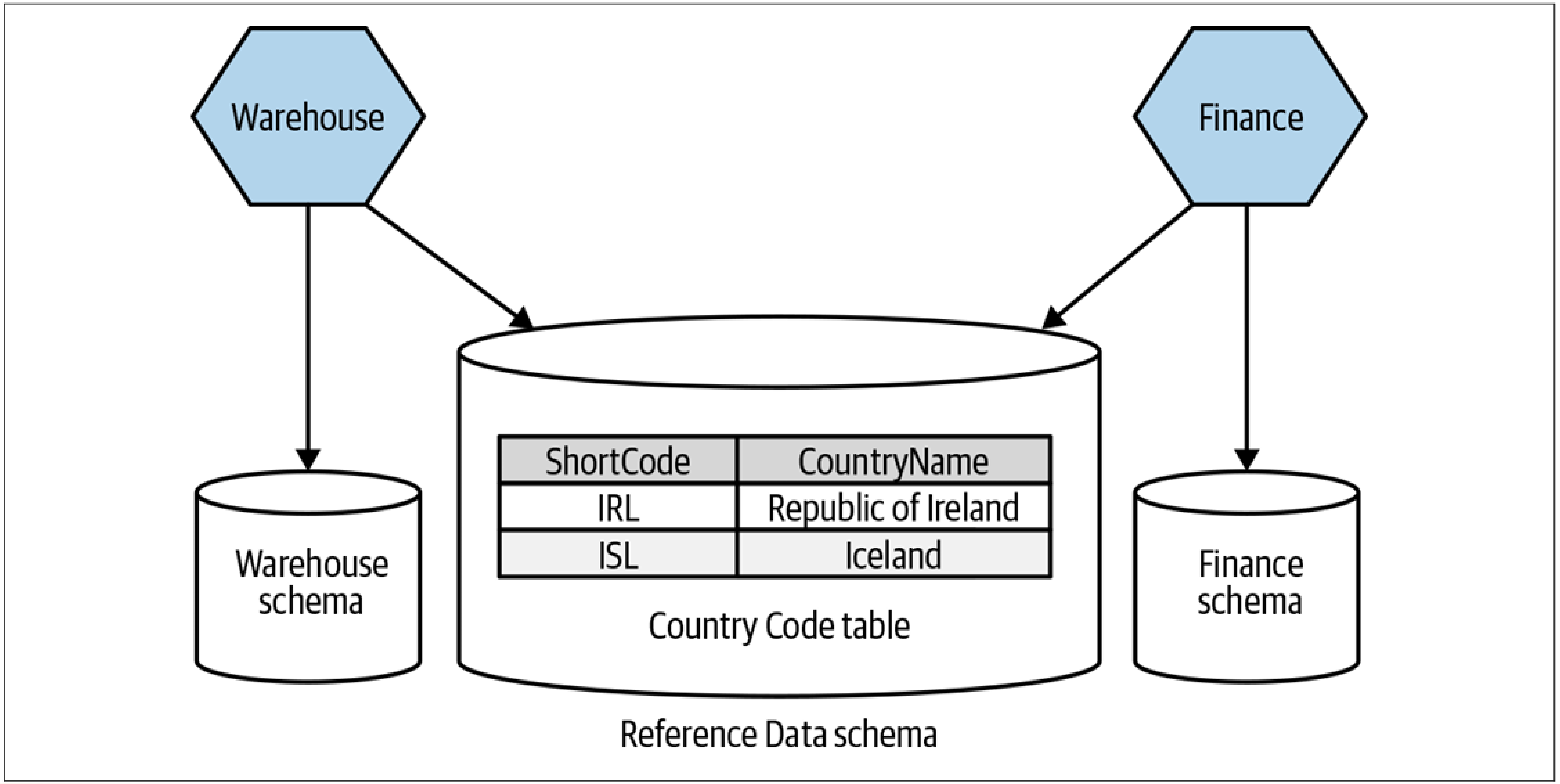
Pattern: Duplicate static reference data



ShortCode	CountryName
IRL	Republic of Ireland
ISL	Iceland

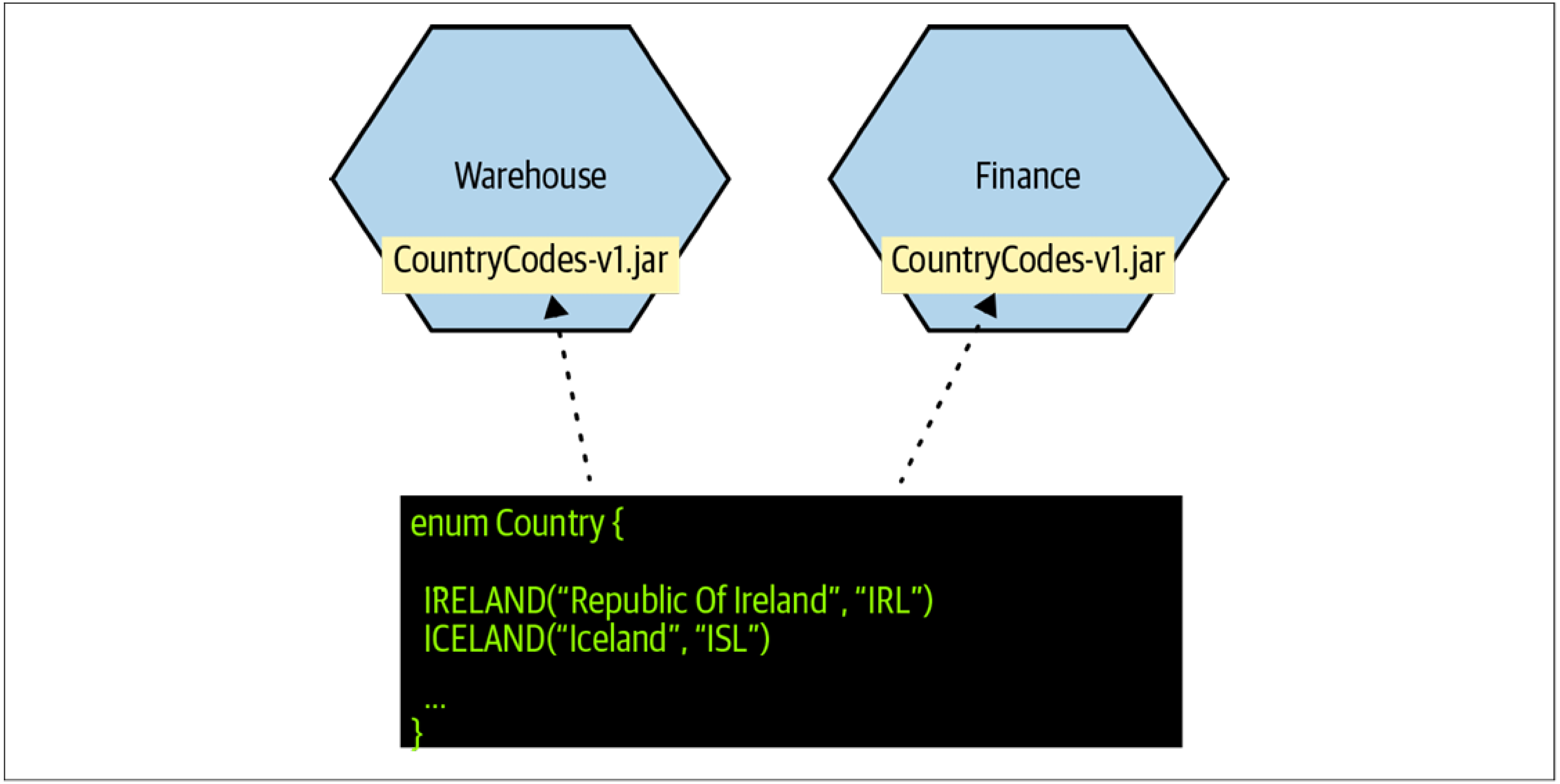
ShortCode	CountryName
IRL	Republic of Ireland
ISL	Iceland

Pattern: Dedicated reference data scheme



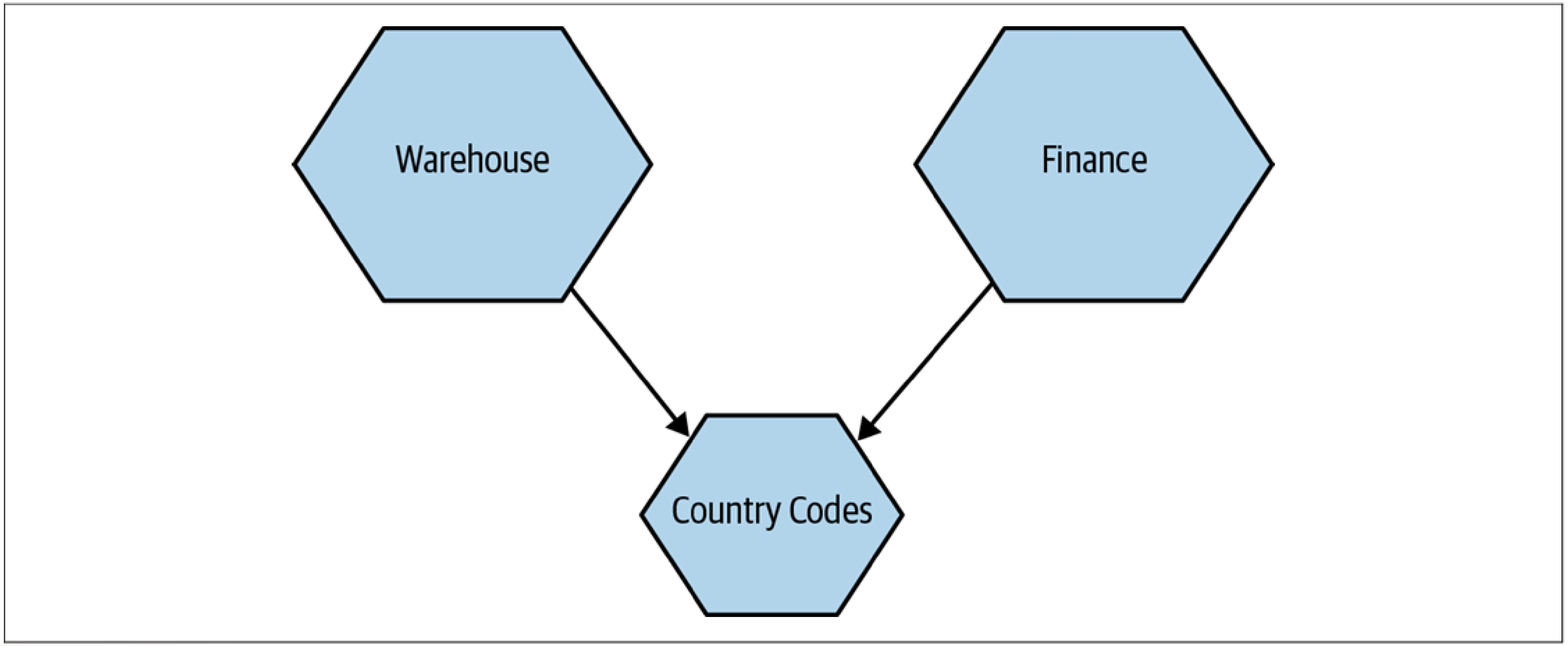
ShortCode	CountryName
IRL	Republic of Ireland
ISL	Iceland

Pattern: Static reference data library

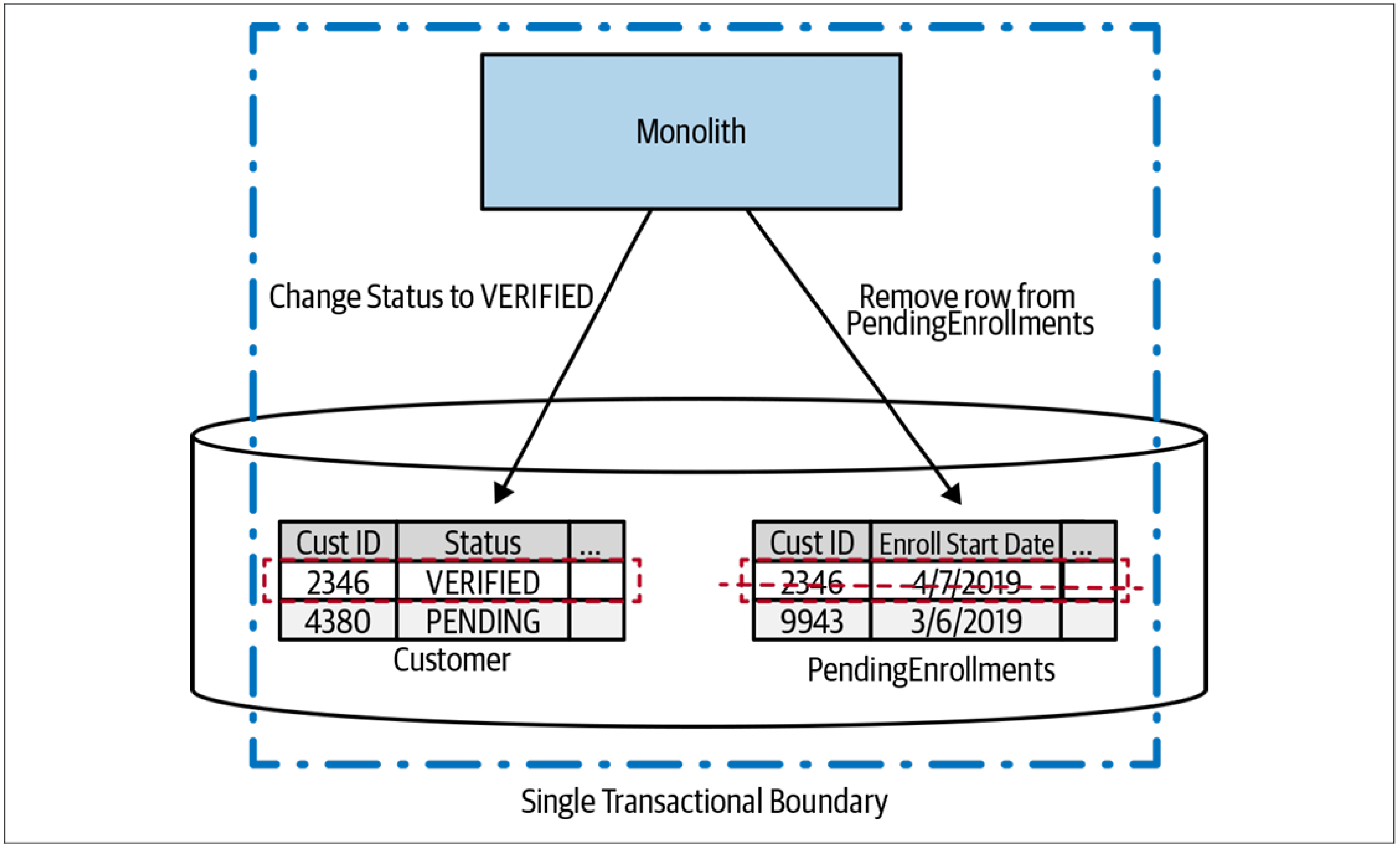


```
enum Country {

 IRELAND("Republic Of Ireland", "IRL")
 ICELAND("Iceland", "ISL")

 ...
}
```

Pattern: Static reference data service



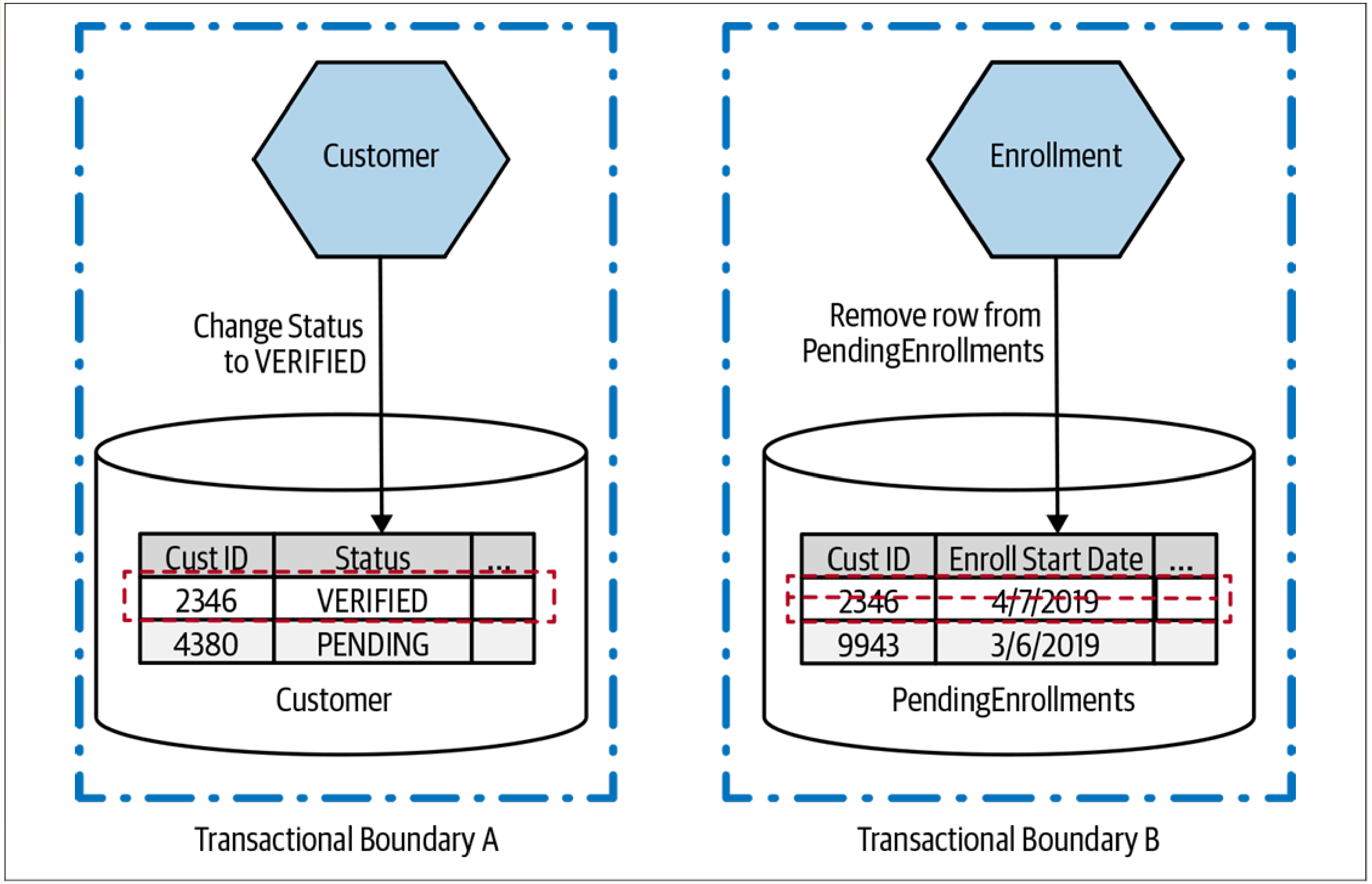
Transactions


Monolith
Change Status to VERIFIED
Remove row from PendingEnrollments
Customer
PendingEnrollments
Single Transactional Boundary

Cust ID	Status	...
2346	VERIFIED	
4380	PENDING	

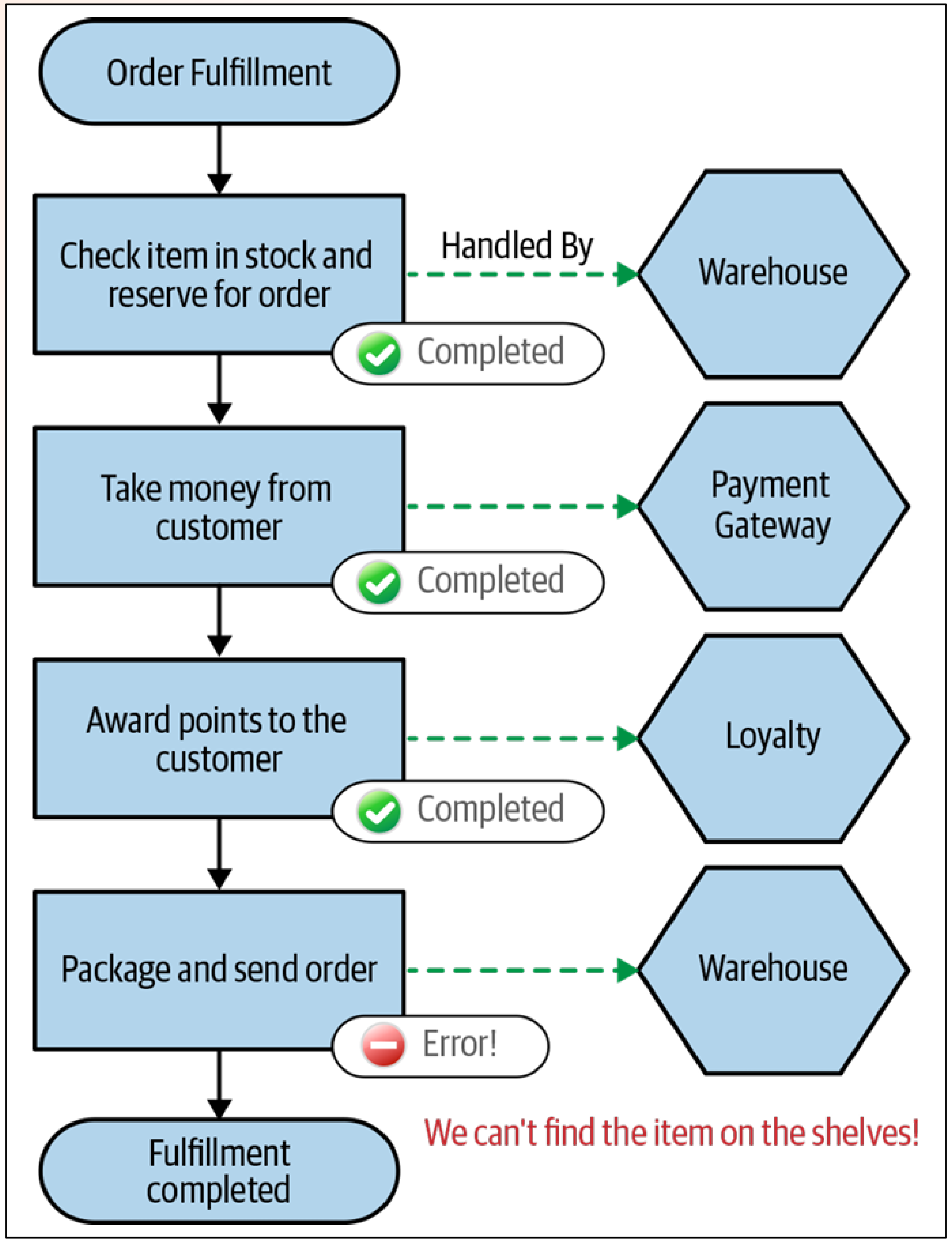
Cust ID	Enroll Start Date	...
~~2346~~	~~4/7/2019~~	
9943	3/6/2019	

Transactions



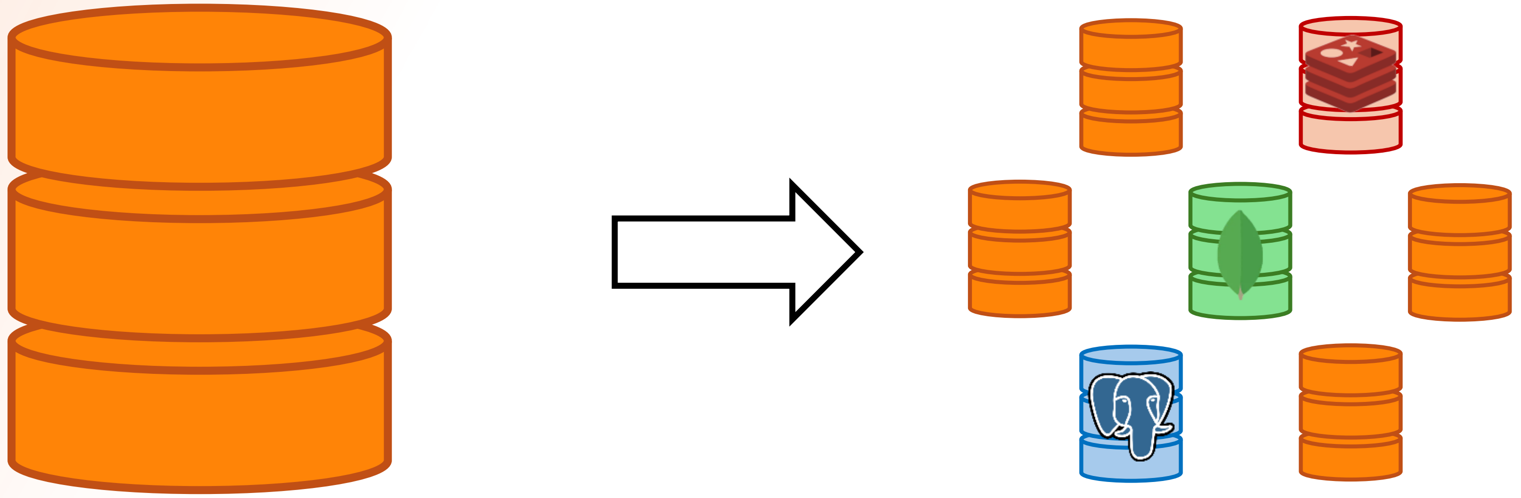
Lack of atomicity is a serious issue for systems that previously relied on this property

Sagas

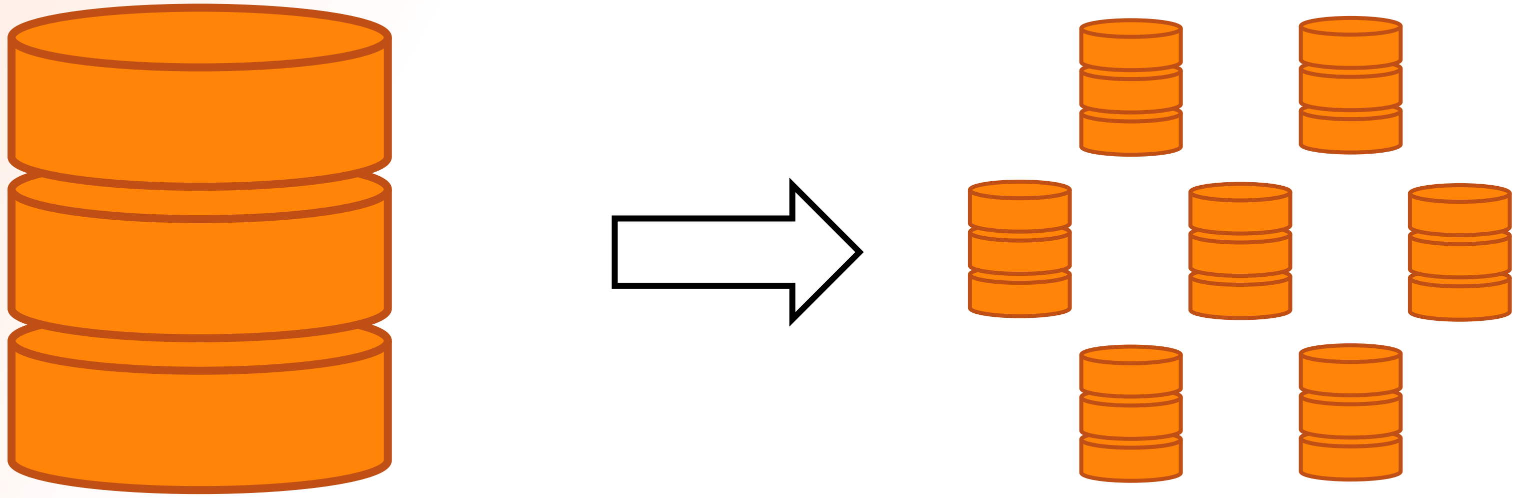

Order Fulfillment
Check item in stock and reserve for order
Handled By
Warehouse
Completed
Take money from customer
Payment Gateway
Completed
Award points to the customer
Loyalty
Completed
Package and send order
Warehouse
Error!
We can't find the item on the shelves!
Fulfillment completed

Change the database type?

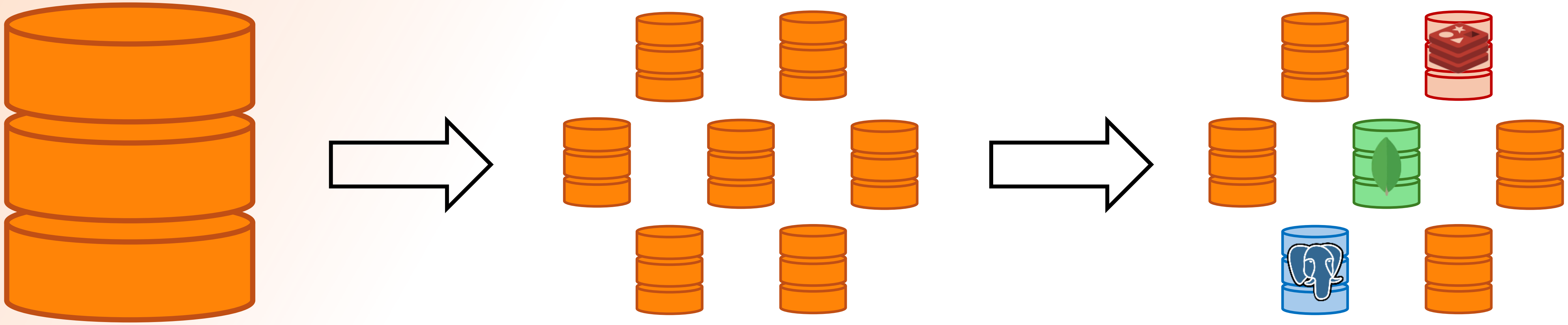
Change the database type?



Change the database type?

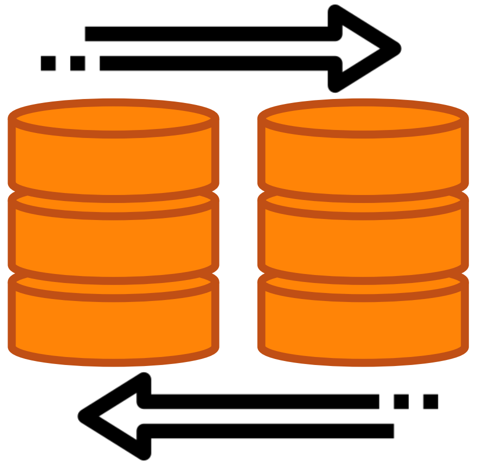


Change the database type?

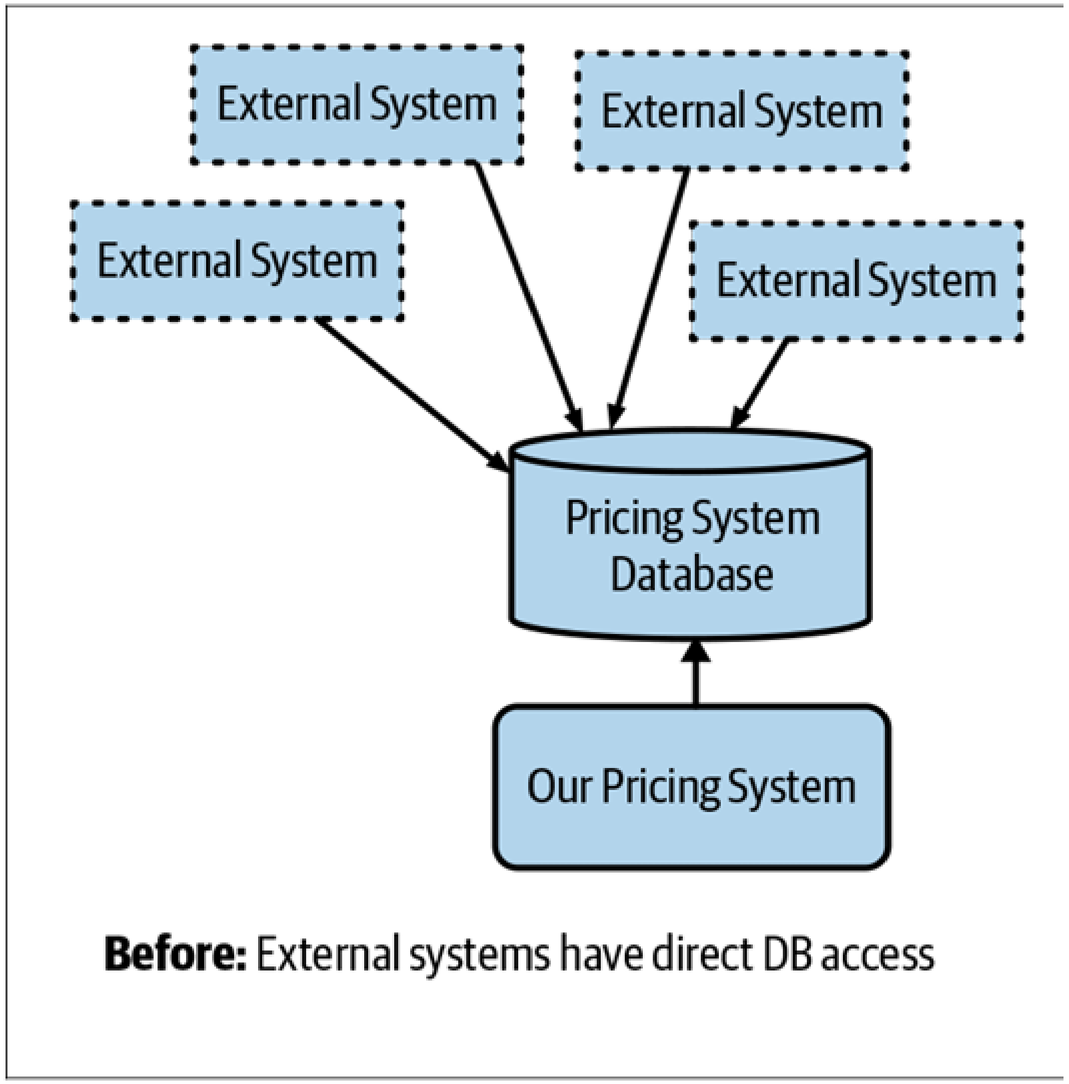


Migration Patterns

Expose Data

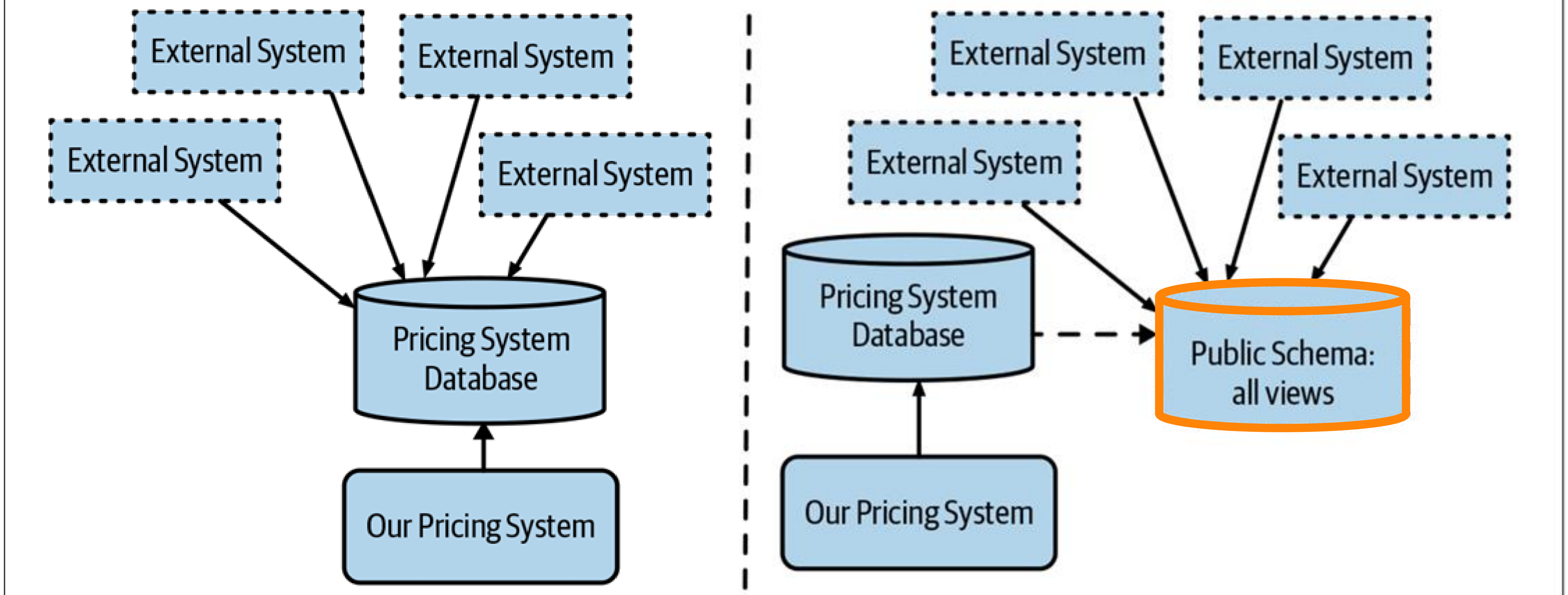


Database views



Before: External systems have direct DB access

Database views



Before: External systems have direct DB access

After: External systems are redirected to read data from views, allowing the schema for the pricing system to change

Database views

Considerations:

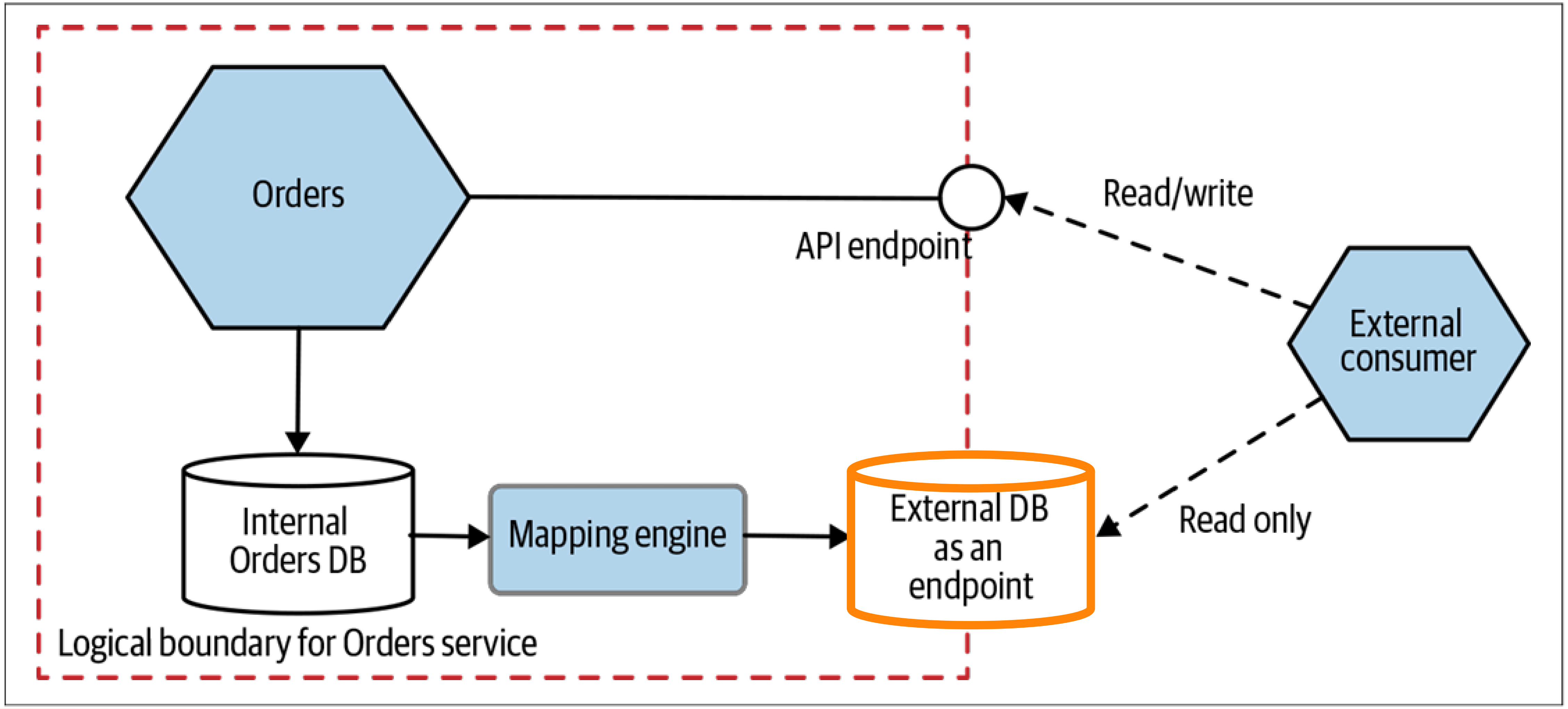
- Easiest strategy to implement
- Largest support from DBMS vendors
- Possible performance issues
- Strong consistency
- One database must be reachable by the other
- Updatable depending on DBMS support

Database materialized views

Considerations:

- Better performance
- Strong or eventual consistency
- One database must be reachable by the other
- Updatable depending on DBMS support

Database as an interface



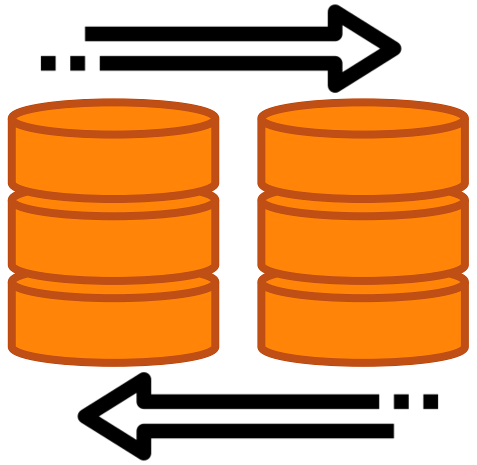
Database as an interface

Considerations:

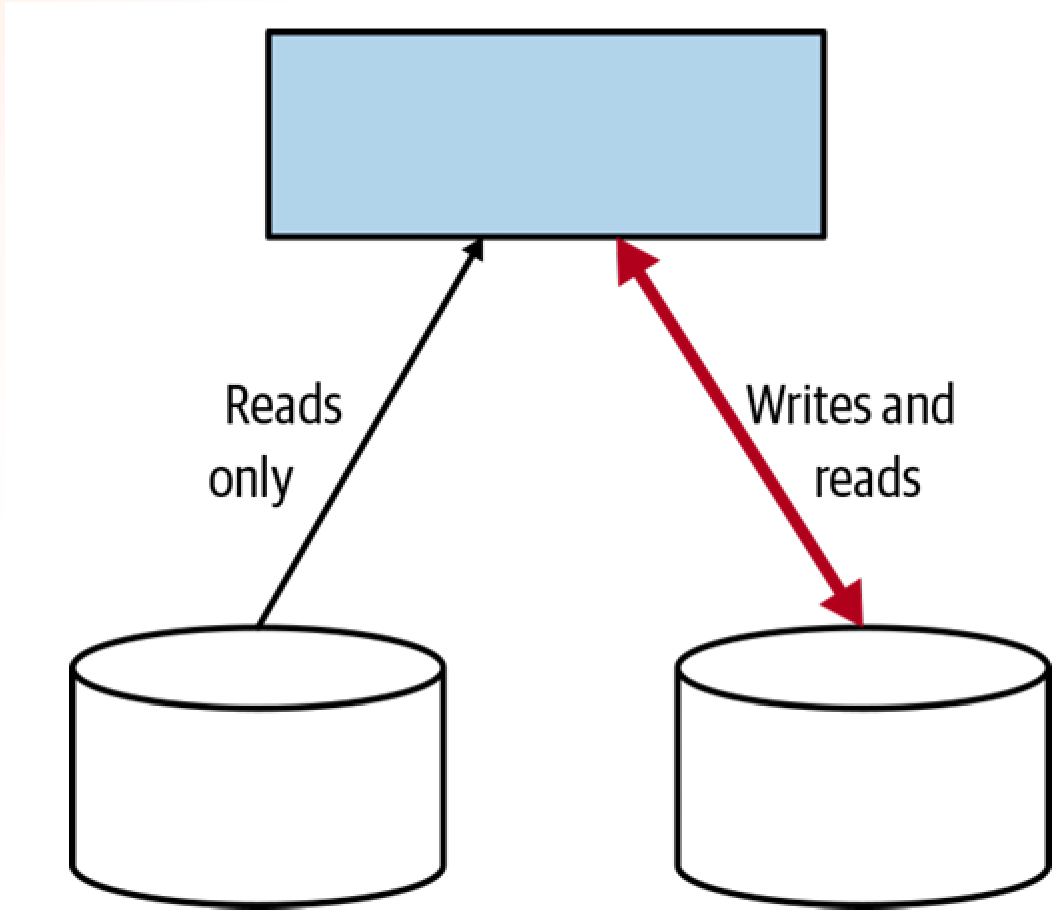
- Requires mapping engine
- Strong or eventual consistency
- Read-only integration
- Can be implemented in many ways

Migration Patterns

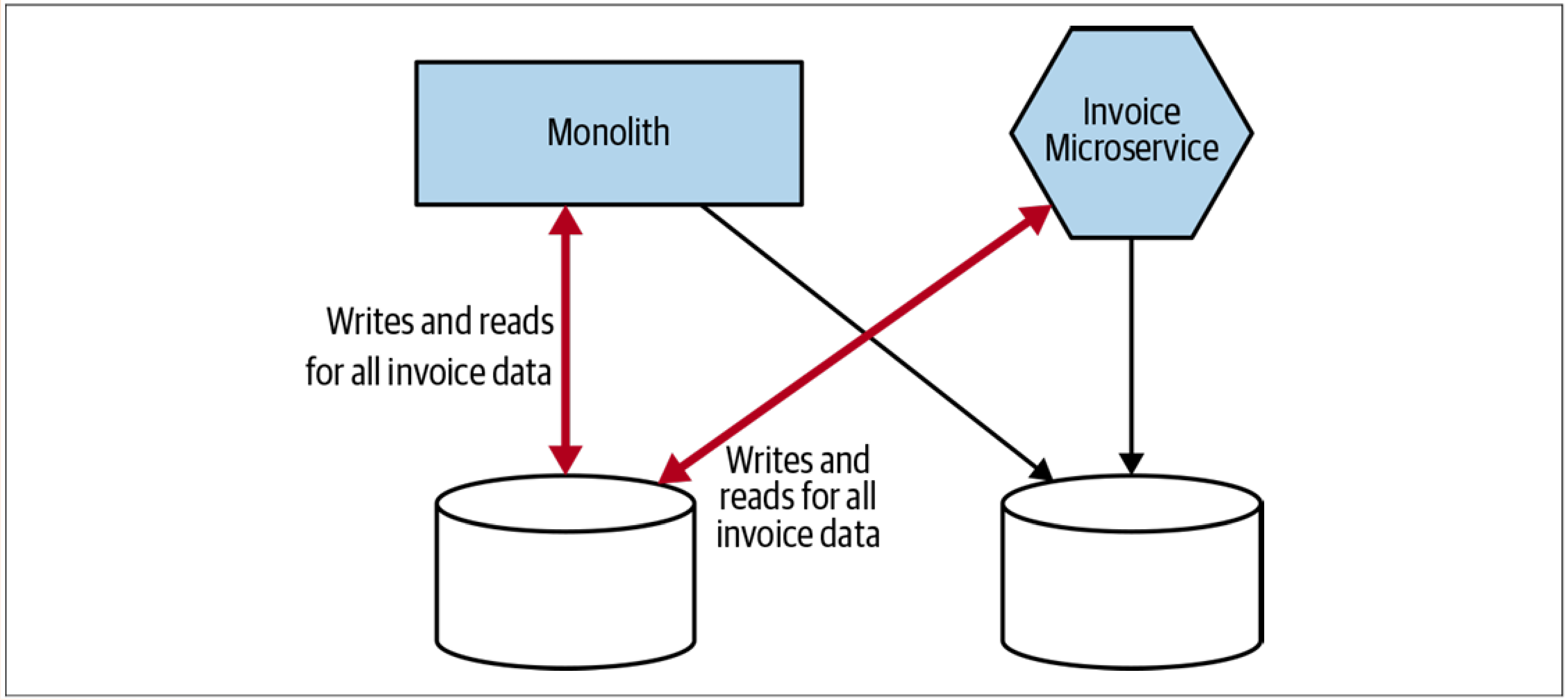
Synchronize Data



Synchronize data in the app



Synchronize data in the app

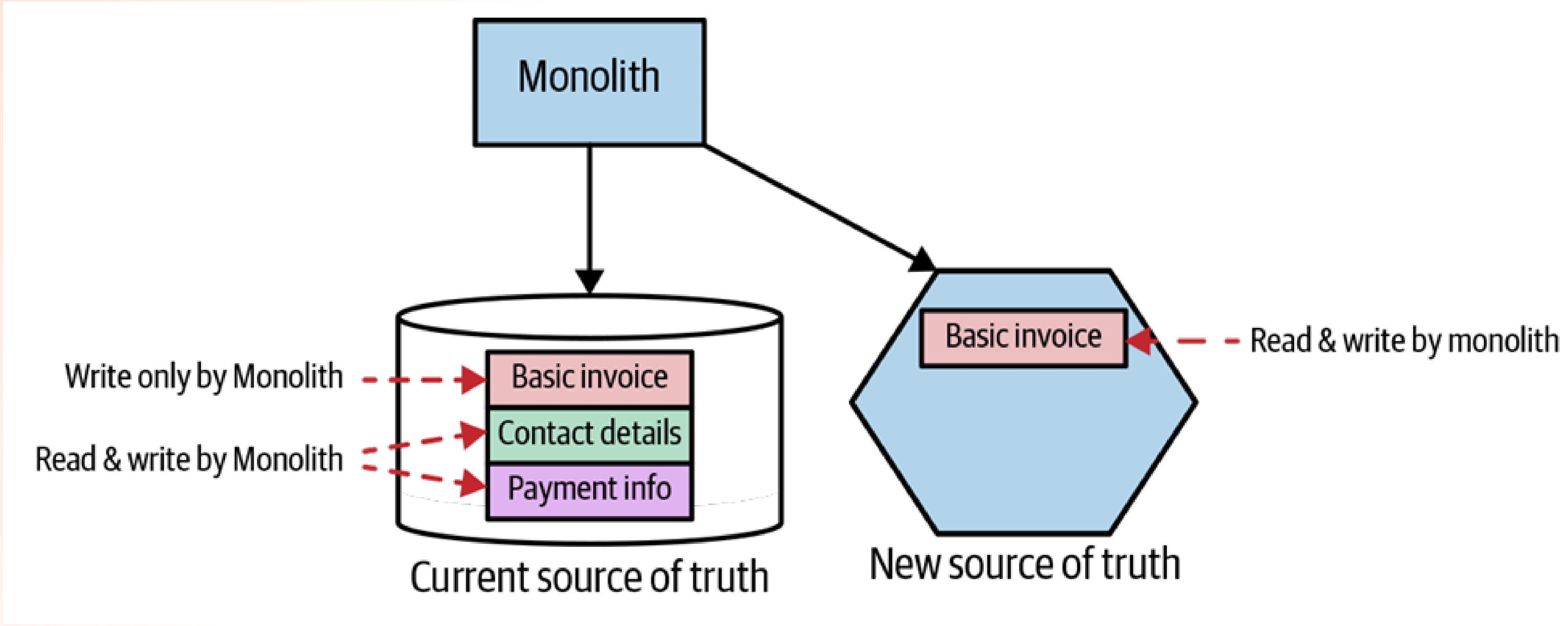

Monolith
Invoice Microservice
Writes and reads for all invoice data
Writes and reads for all invoice data

Synchronize data in the app

Considerations:

- Double write logic in the app
- Strong consistency
- High cohesion and low coupling

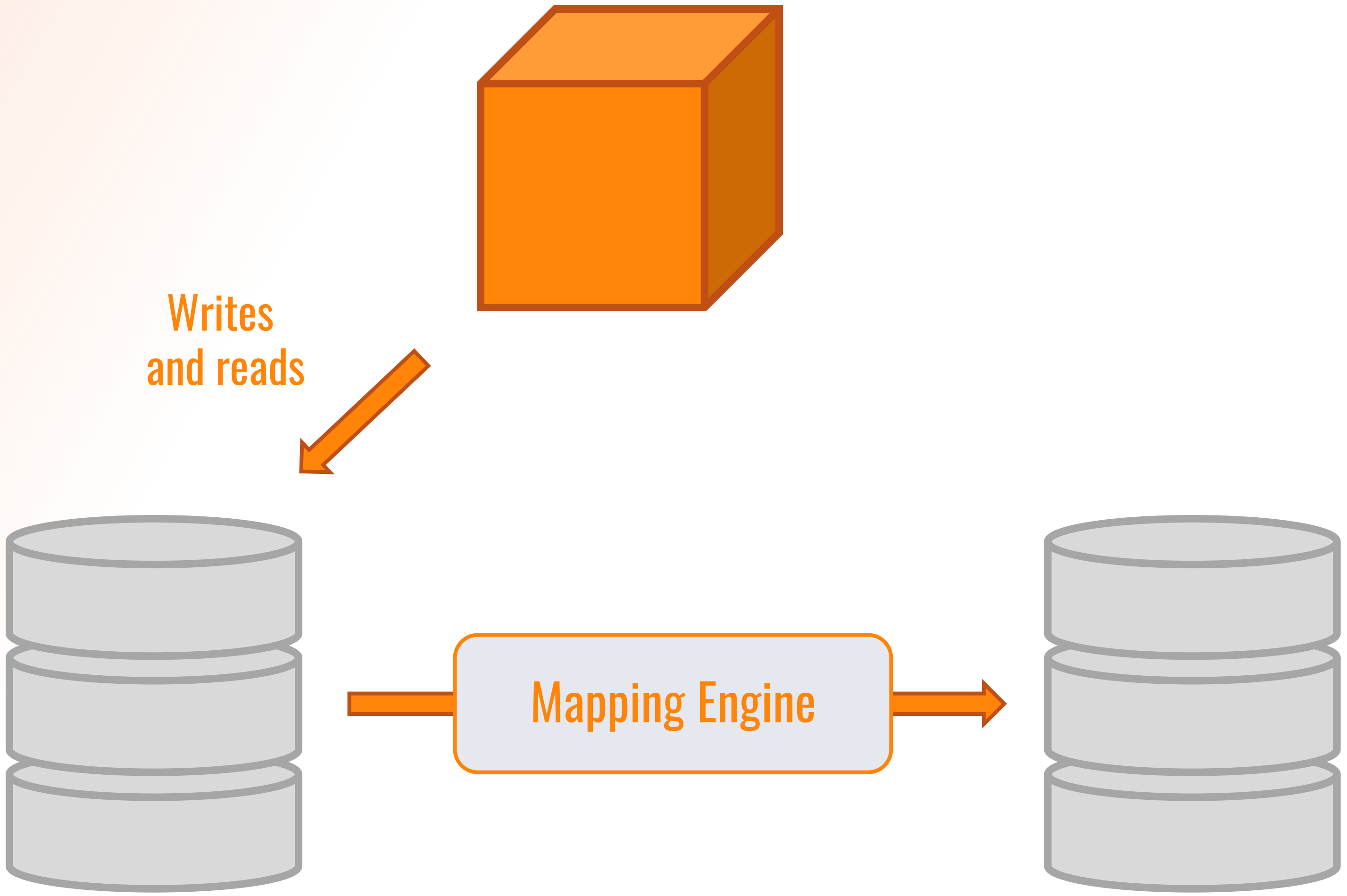
Tracer Write



Write - inconsistency

- Write to one source
- Send writes to both sources
- Seed writes to either source

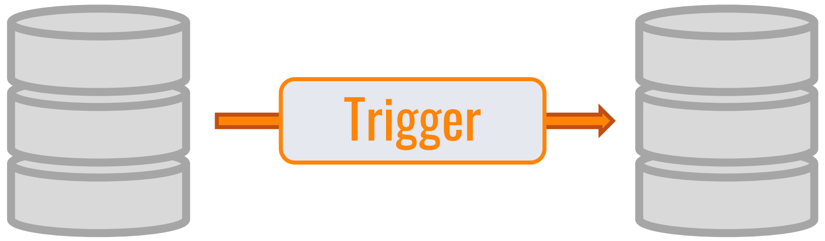
How to keep this in sync?



Database trigger

Considerations:

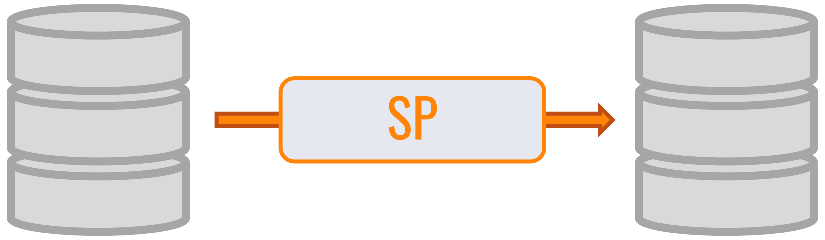
- Depends on the DBMS support
- Strong consistency
- One database must be reachable by the other



Transactional code

Considerations:

- Any code: usually stored procedures or distributed transactions
- Strong consistency
- Possible cohesion/coupling issues
- Possible performance issues
- Updatable depending on how it is implemented



ETL - Extract Transform Load

Considerations:

- Many available tools
- Requires external trigger (usually time-based)
- Can aggregate from multiple data sources
- Eventual consistency
- Read-only integration

Replication

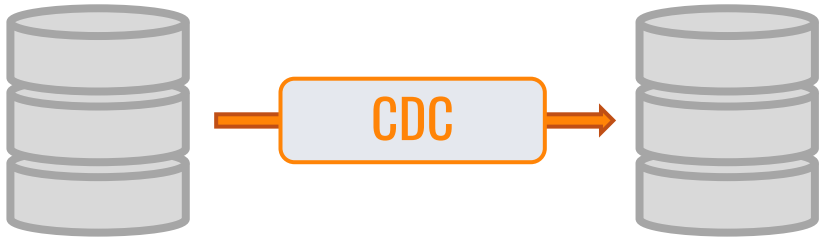
Considerations:

- A few available options
- The same source and target schemas
- Strong or Eventual consistency
- Updatable depending on replication type

Replication

Change Data Capture

It’s a process of identifying and capturing changes made to data in a database and then delivering those changes in real time to a downstream process or system.



Change Data Capture - Debezium

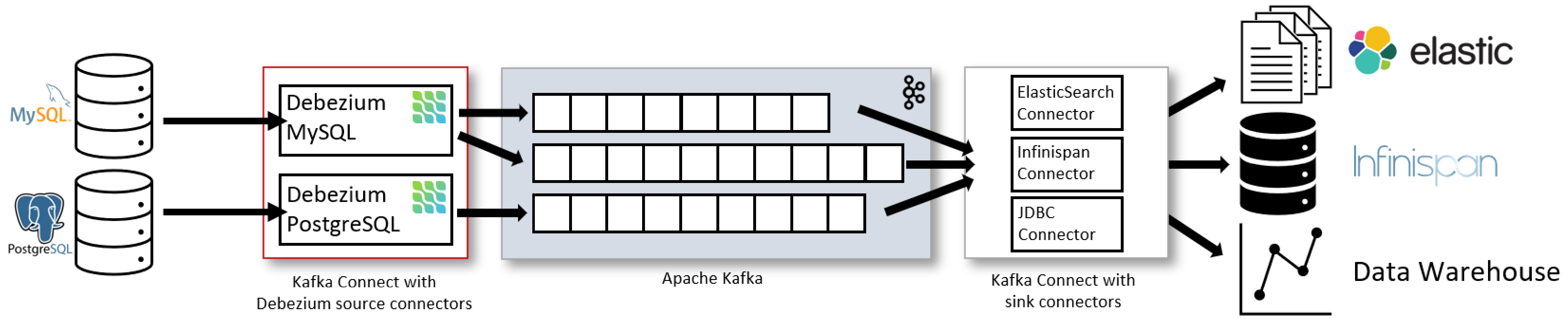


Image source: debezium.io

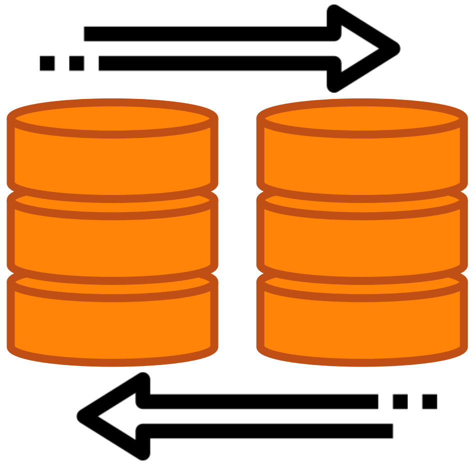
Change Data Capture

Considerations:

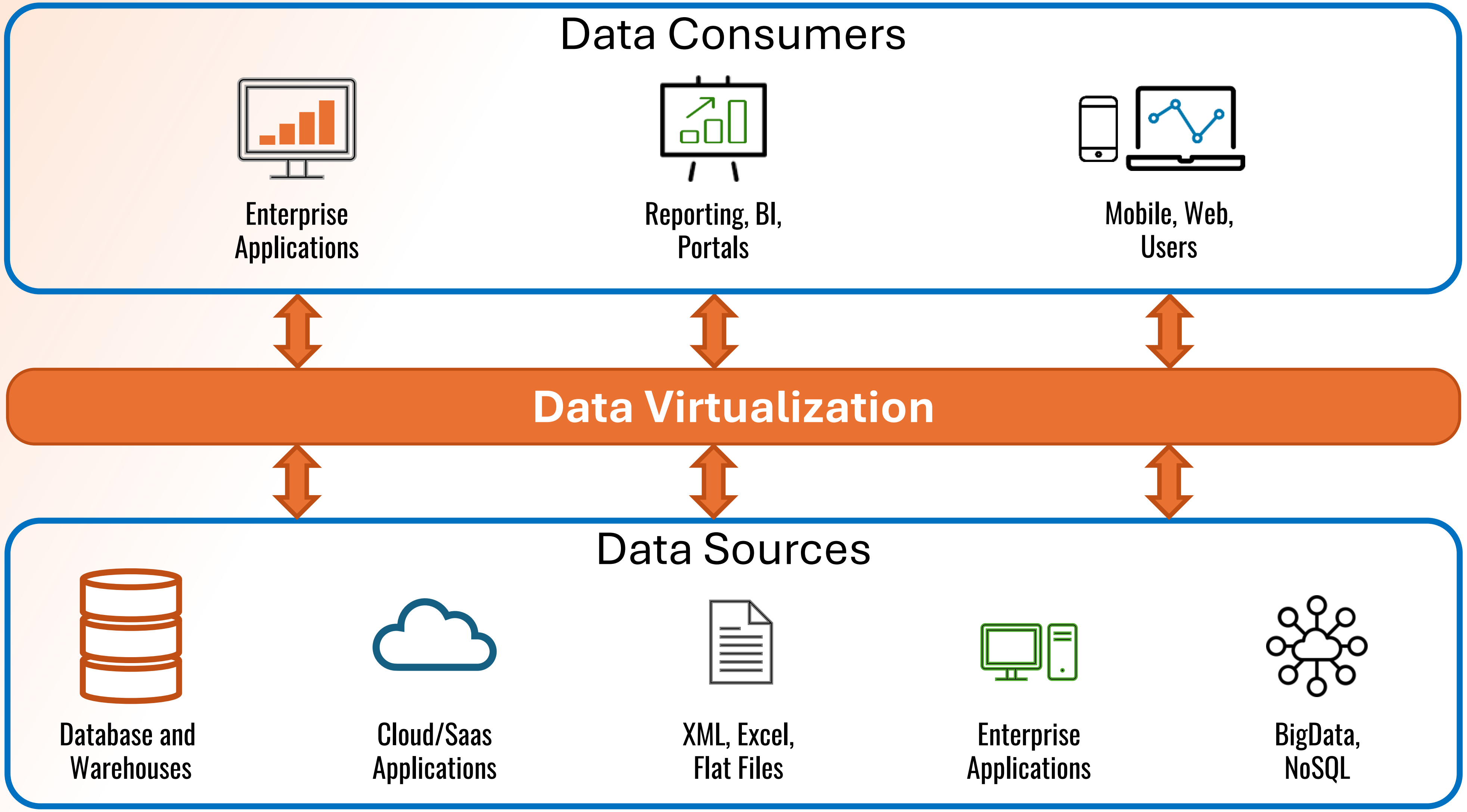
- Read data source is updated through a stream of events
- Eventual consistency
- Usually combined with a message bus – Kafka?
- High scalability

Migration Patterns

Virtualize Data



Data virtualization



Data virtualization

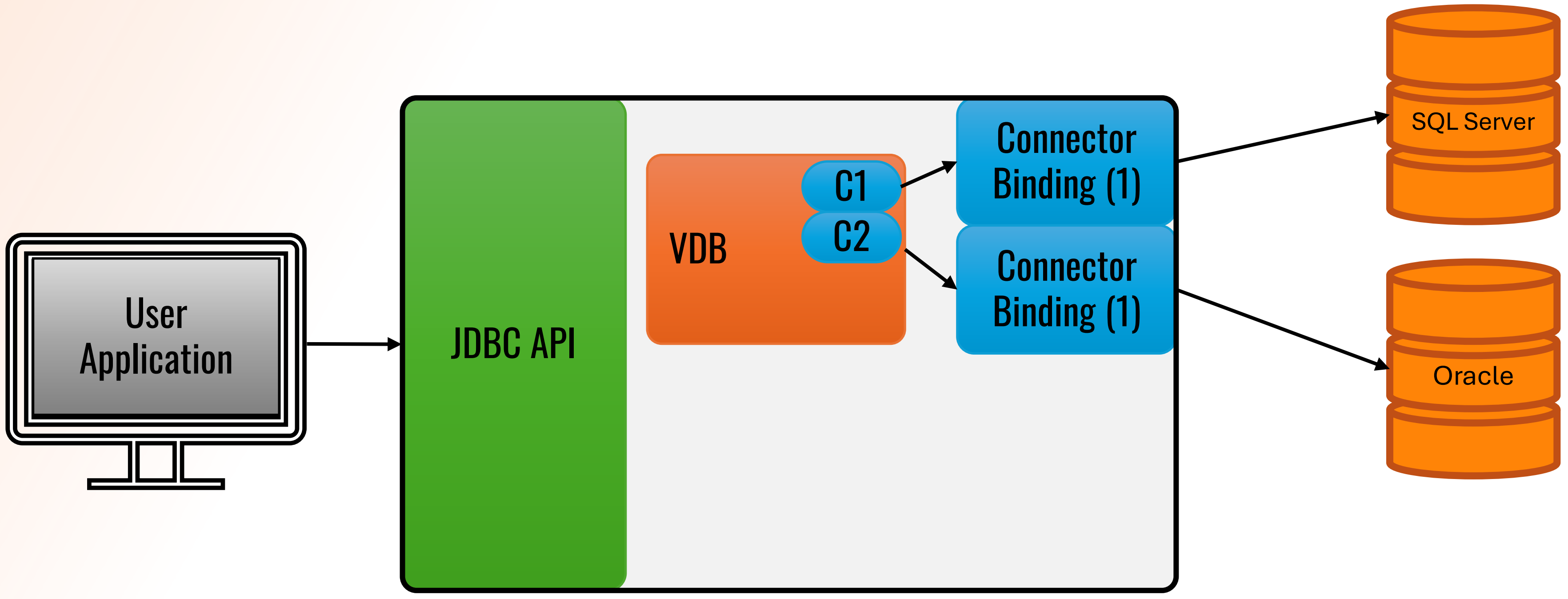


Image source: teiid.io

Data virtualization

Many tools available:

- Teiid.io
- Denodo
- ...
- SQL Server - Polybase

Data virtualization

Considerations:

- Real-time access option
- Strong or eventual consistency
- Can aggregate from multiple data sources
- Updatable depending on data virtualization platform
- Additional layer can introduce latency

Summary

SQL Day

”

Code is easy;
State is hard.

”

-Edson Yanaga

Data integrity

Data consistency

Transaction boundaries

Data mapping engine

Be prepared…

…for issues that didn’t exist in the monolithic system!

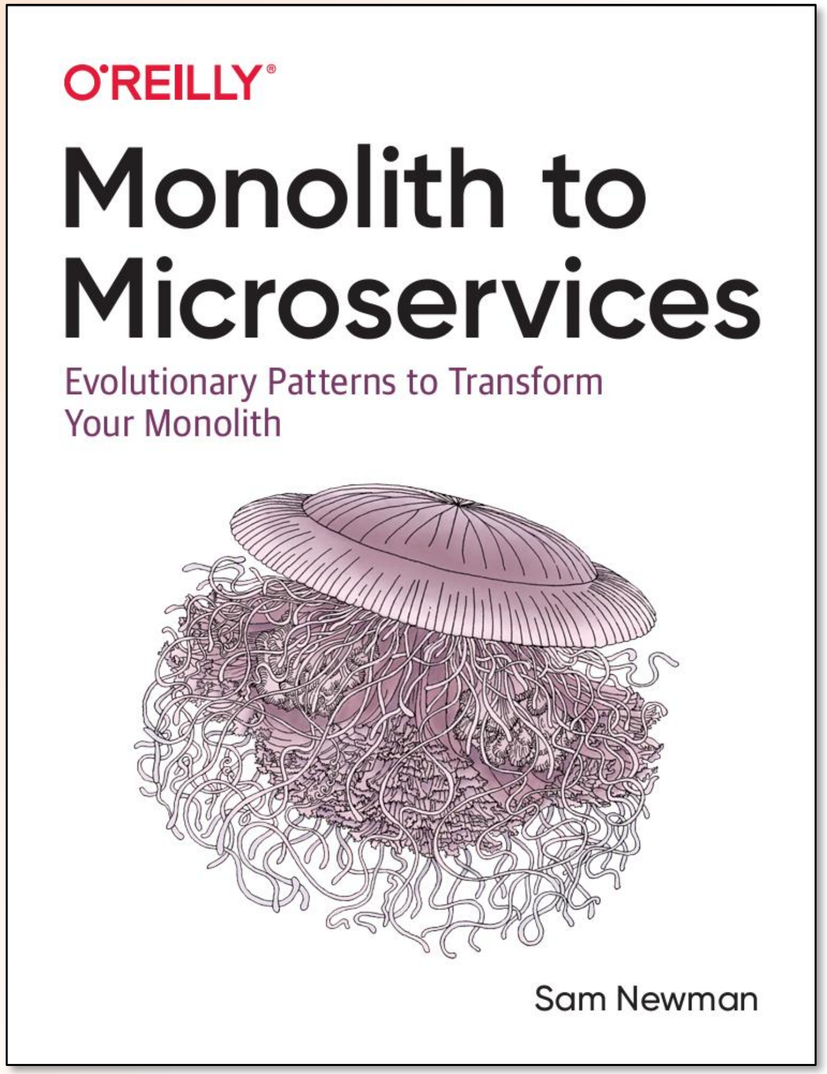
Remember

Rehearse your migrations

and

Check your data between migrations steps

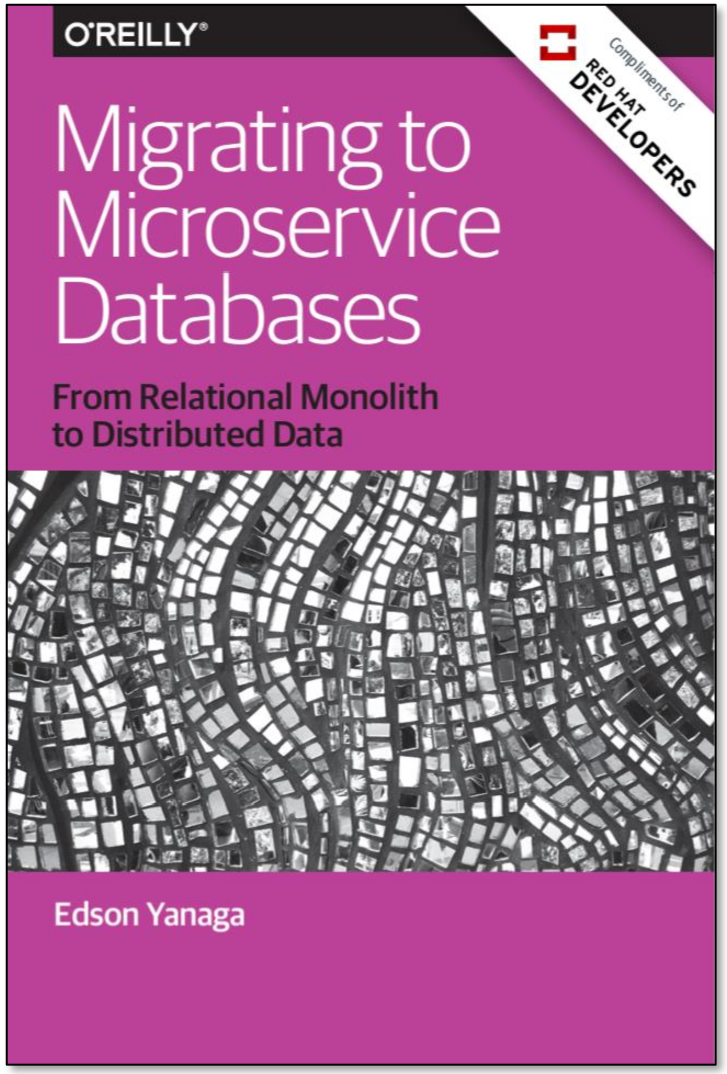
Resources


O'REILLY

Monolith to Microservices

Evolutionary Patterns to Transform Your Monolith

Sam Newman


O'REILLY

Compliments of RED HAT DEVELOPERS

Migrating to Microservice Databases

From Relational Monolith to Distributed Data

Edson Yanaga

Data
Community

SQL Day

16 edycja konferencji SQLDay

13-15 maja 2024, WROCŁAW + ONLINE

Data Community

partner platynowy

lingaro

partner złoty

DXC TECHNOLOGY

VOLVO

software one

partner srebrny

accenture

Nordcloud an IBM Company

Capgemini

TECHNOLOGY INNOVATION DATA KNOWLEDGE tidk

elitmind think bright

C&F