The Cloud Design Patterns you didn't know you needed

Barry Luijbregts
SOFTWARE ARCHITECT & DEVELOPER

@AzureBarry www.azurebarry.com

Find me on Pluralsight

- Introduction to Azure App Services
- Building a Global App with Azure PaaS
- Continuous Integration and Continuous Delivery: The Big Picture

- Cloud Design Patterns for Azure: Design and Implementation
- Cloud Design Patterns for Azure: Availability and Resilience
- Cloud Design Patterns for Azure: Data Management and Performance

- The .NET Ecosystem: The Big Picture
- Microsoft Azure for Developers: What to Use When?
- Microsoft Azure Cognitive Services: The Big Picture

Find me on Pluralsight

- Introduction to Azure App Services
- Building a Global App with Azure PaaS
- Continuous Integration and Continuous Delivery: The Big Picture

- **Cloud Design Patterns for Azure: Design and Implementation**
- **Cloud Design Patterns for Azure: Availability and Resilience**
- **Cloud Design Patterns for Azure: Data Management and Performance**

- The .NET Ecosystem: The Big Picture
- Microsoft Azure for Developers: What to Use When?
- Microsoft Azure Cognitive Services: The Big Picture


kimiakewoo

The cloud takes care of the plumbing, so that you can build things that matter

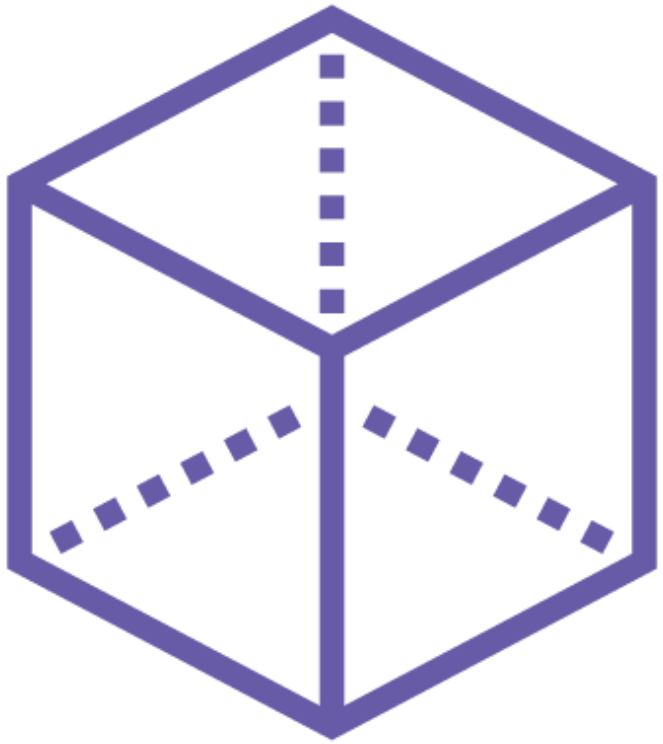
Survey!


Do you use Azure?

Design patterns?

Why Design Patterns Matter

What Are Design Patterns

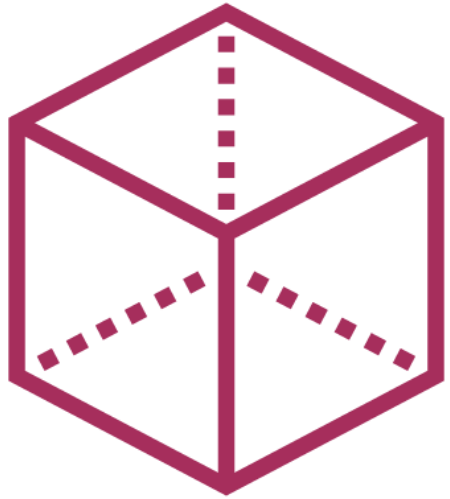


Solutions to common software design problems

A common language for software professionals

Recipes for designing solutions

Why Design Patterns Matter

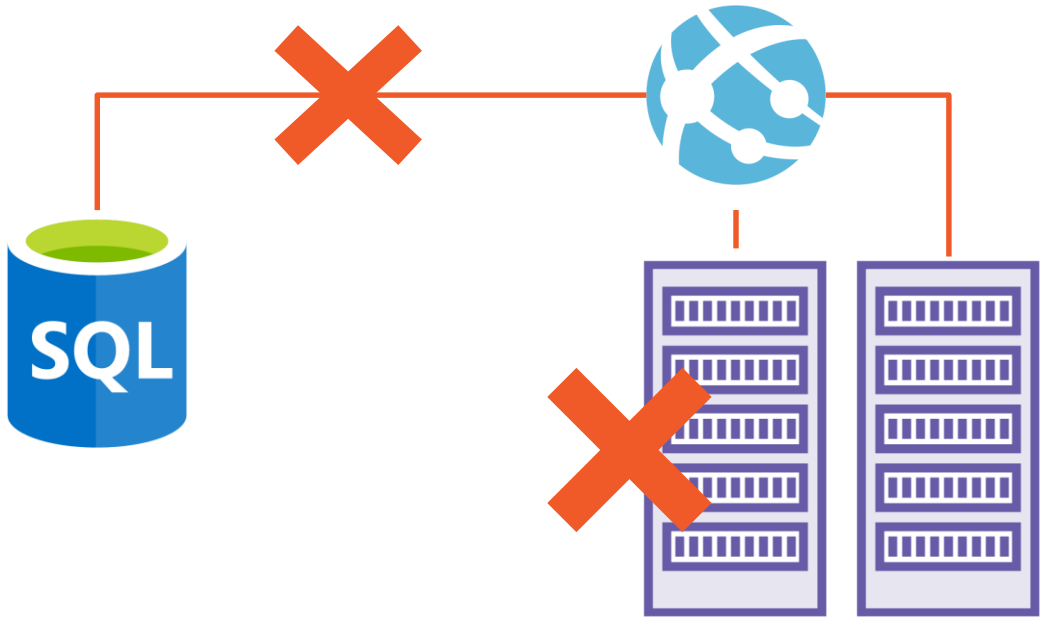


Identify solutions
Consider all caveats

Inspire design thinking

Best practices
Better software

Why Design Patterns for the Cloud?



Cloud Design Patterns

Cloud Design Patterns for Azure

Design and Implementation

- External Config Store
- Federated Identity
- Gatekeeper
- Runtime Reconfiguration
- Valet Key
- ....

Data Management and Performance

- Automatic Scaling
- Cache-aside
- CQRS
- Event Sourcing
- Sharding
- Static Content Hosting
- ....

Availability and Resilience

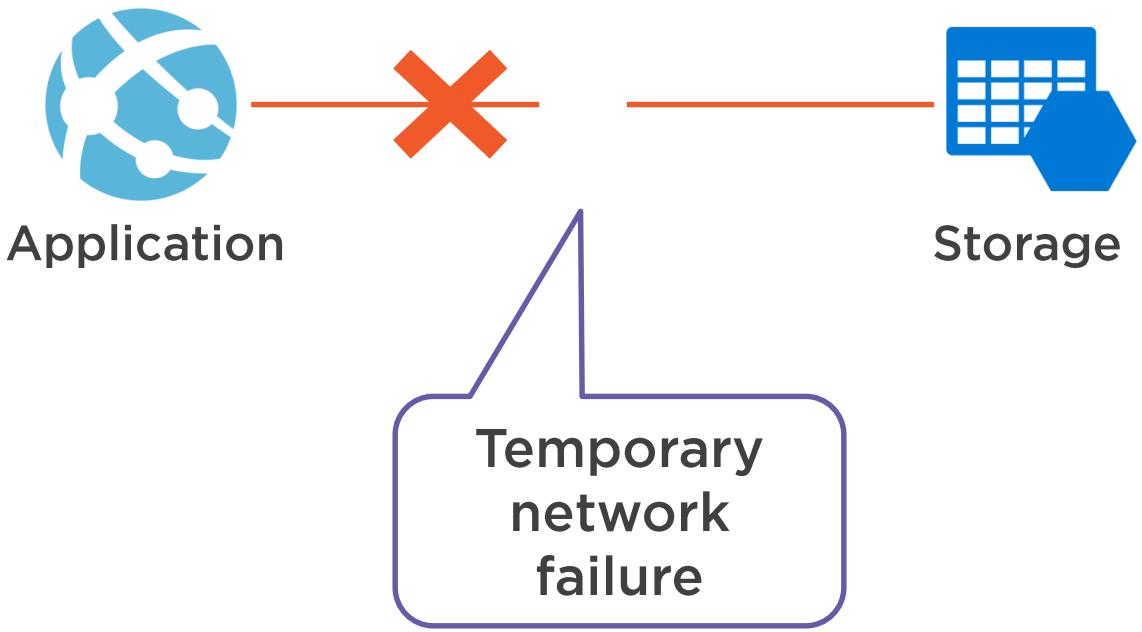
- Circuit Breaker
- Compensating Transaction
- Health Endpoint Monitoring
- Queue-based Load Leveling
- Retry
- Throttling
- ....


YOU CAN DUST IT OFF AND TRY
AGAIN.

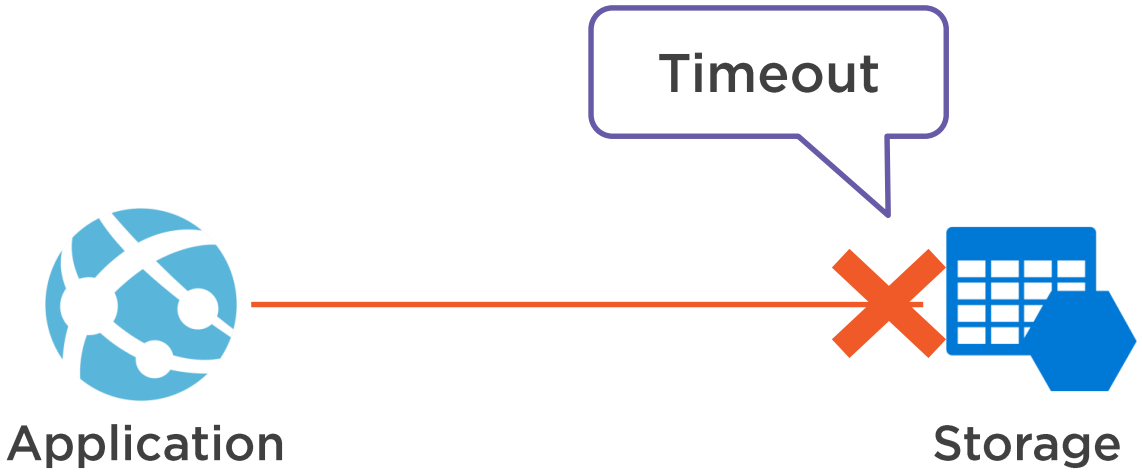
The Retry Pattern

The Problem with Transient Failures

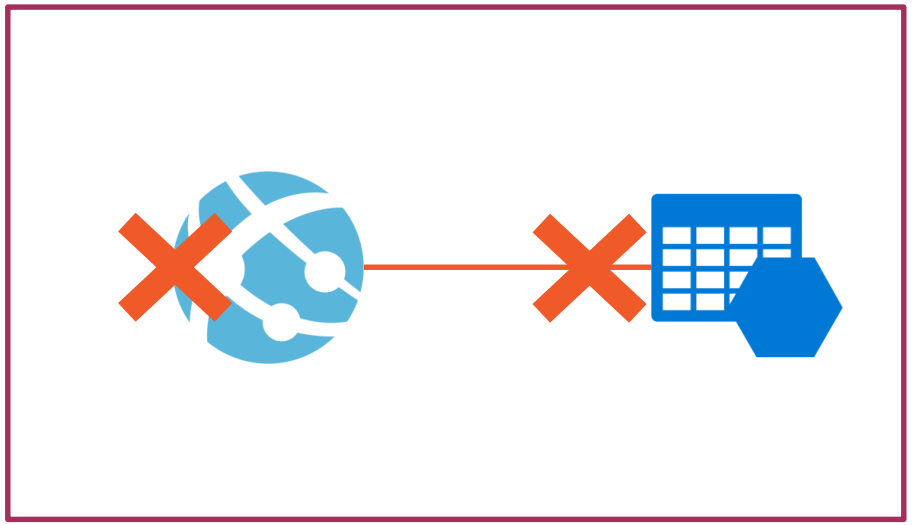
The Problem with Transient Failures



The Problem with Transient Failures



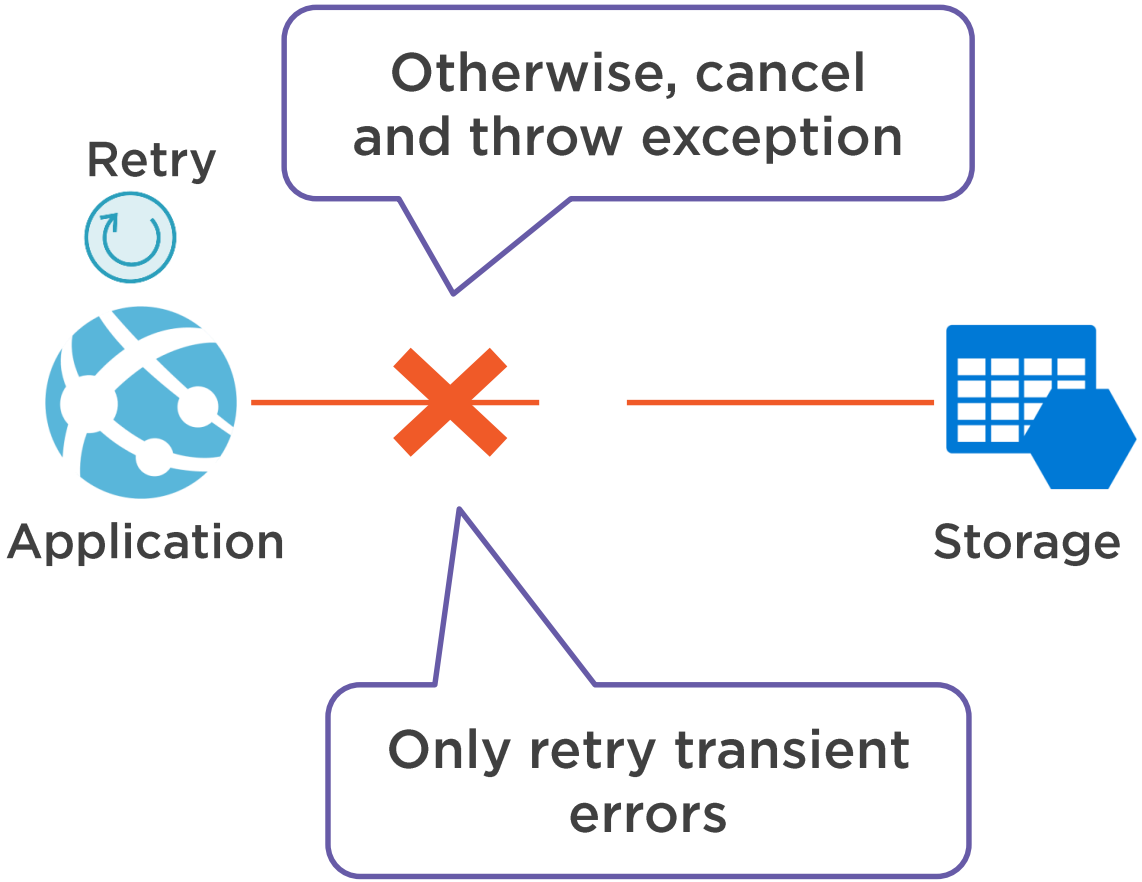
The Problem with Transient Failures



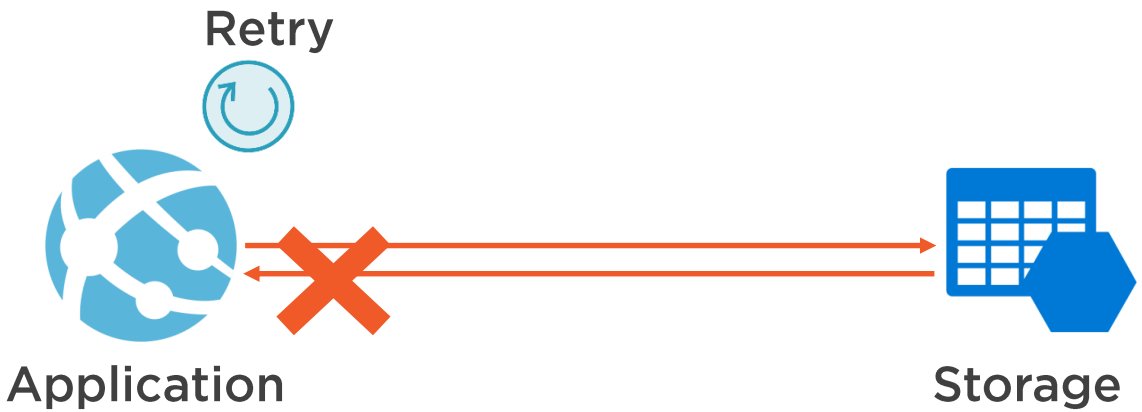
Application stability degrades when services fail

The Solution to the Problem

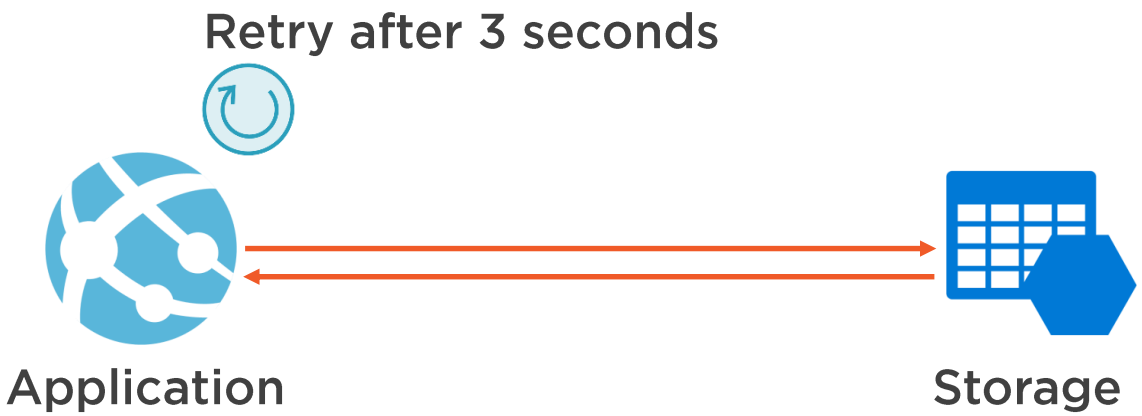
The Retry Pattern



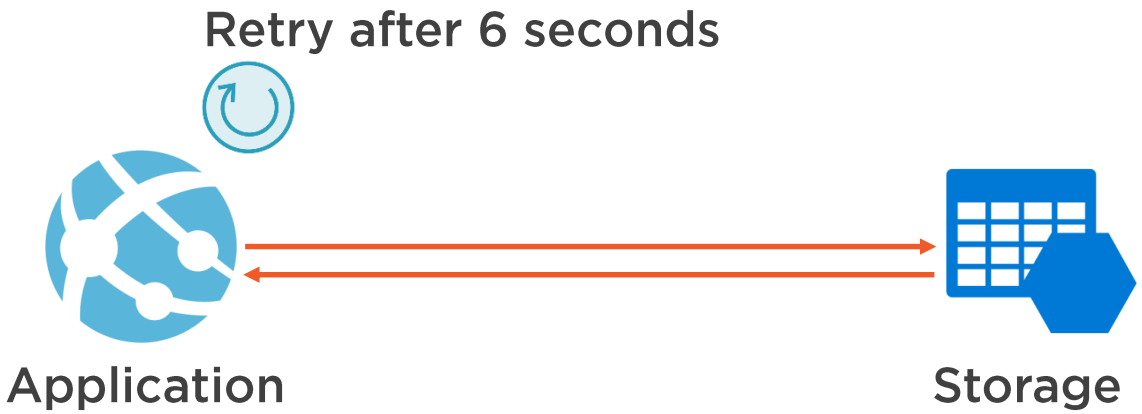
The Retry Pattern



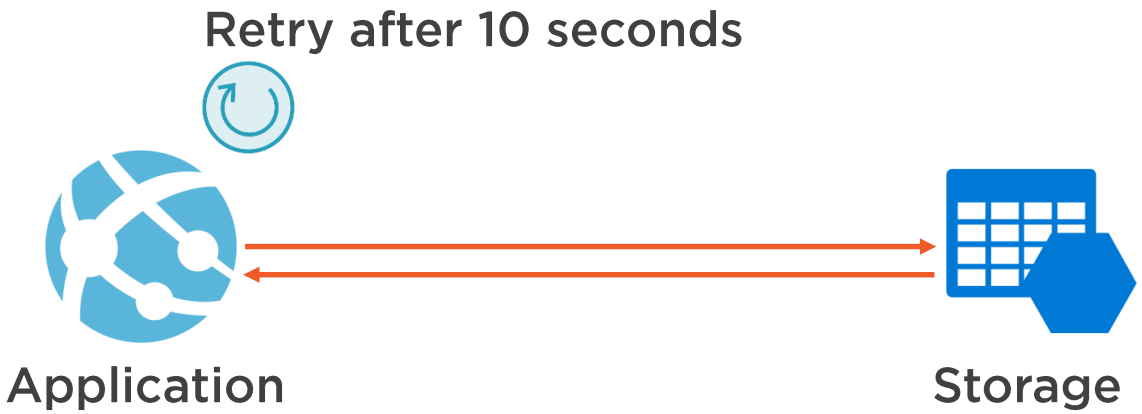
The Retry Pattern



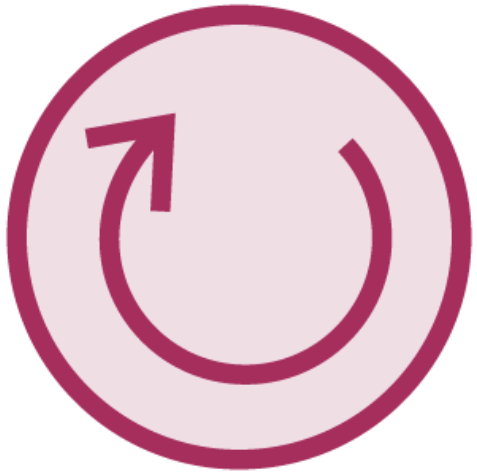
The Retry Pattern



The Retry Pattern



Things to Consider



Only retry transient failures

- That are likely to resolve themselves quickly
- Otherwise, use the circuit breaker pattern

Match the retry policies with the application

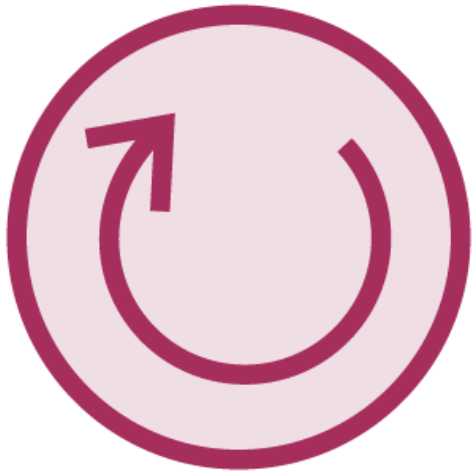
- “please try again”

Don’t cause a chain reaction

- By calling other services that retry

Log all retry attempts

When to Use This Pattern



When calls are likely to succeed after retrying

When Not to Use This Pattern

When calls are likely to fail

Retry Out of The Box

- Azure Redis Cache
- Azure Cosmos DB
- Azure Service Bus
- Azure SQL (Entity Framework)
- Azure Storage

Demo

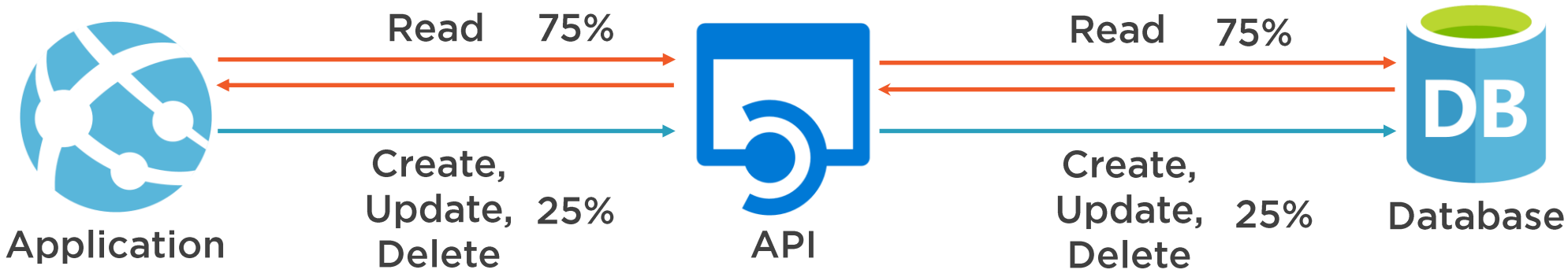
Console Application

Azure Storage

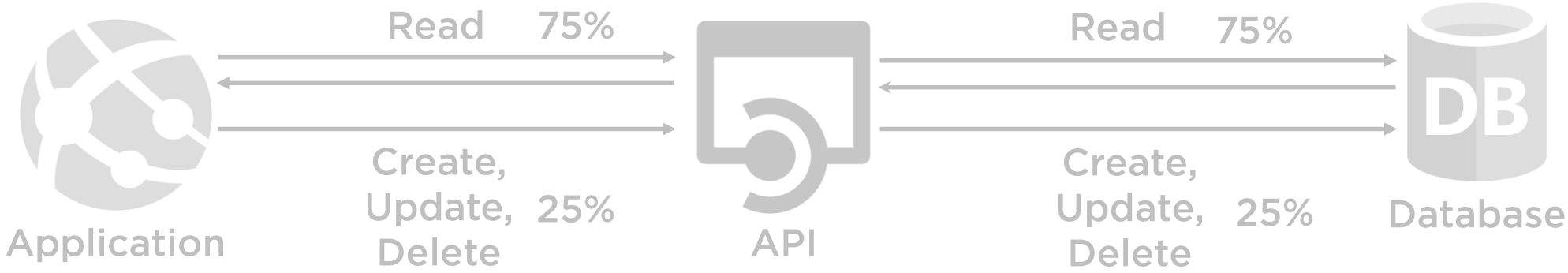
Azure Storage Client Library

The CQRS Pattern

The Problem with Mixing Reads and Writes



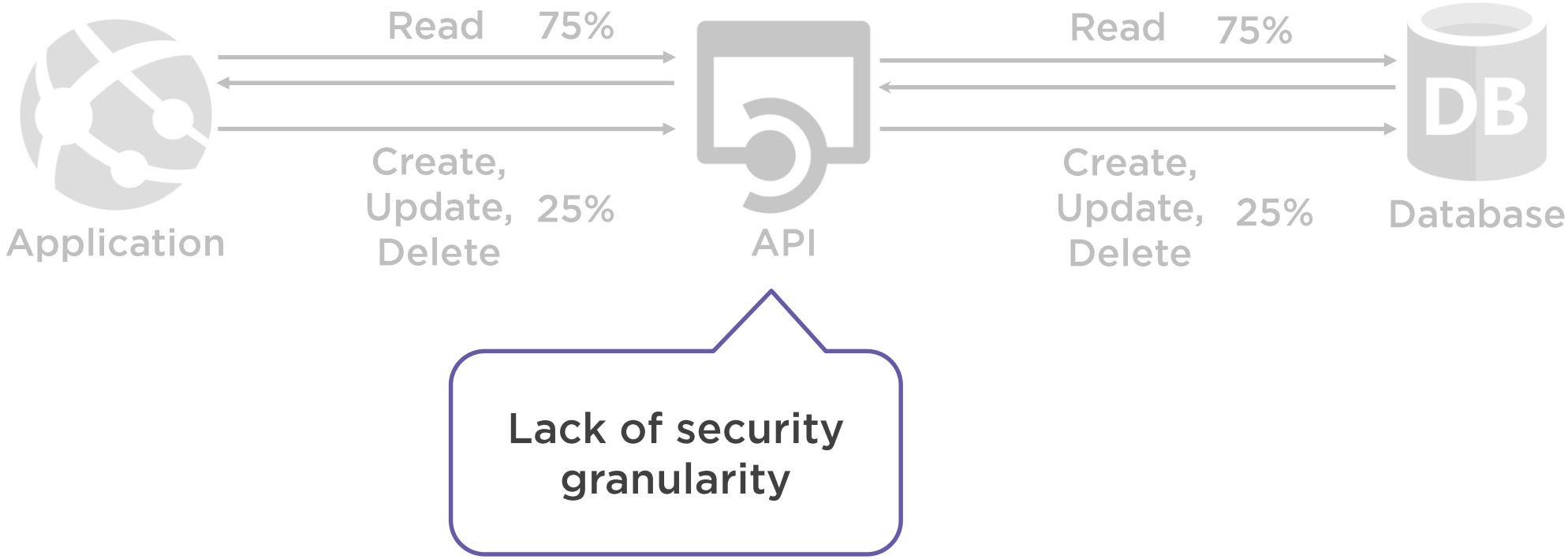
The Problem with Mixing Reads and Writes



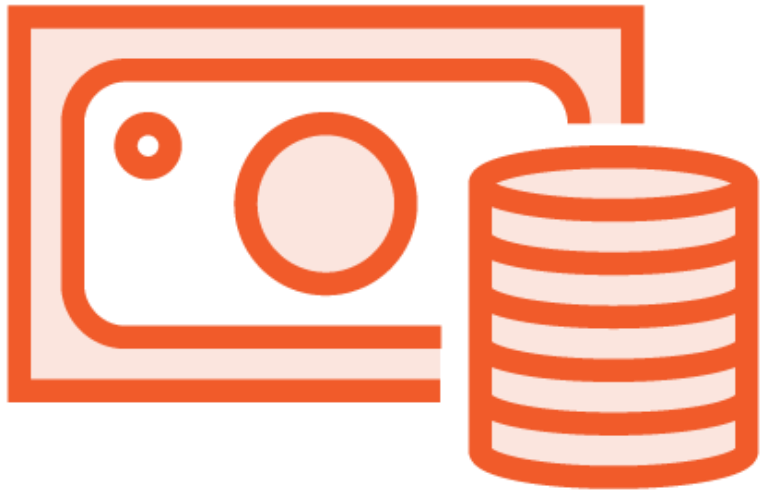
Scaling

- **is inefficient**
- **expensive**

The Problem with Mixing Reads and Writes



The Problem with Mixing Reads and Writes

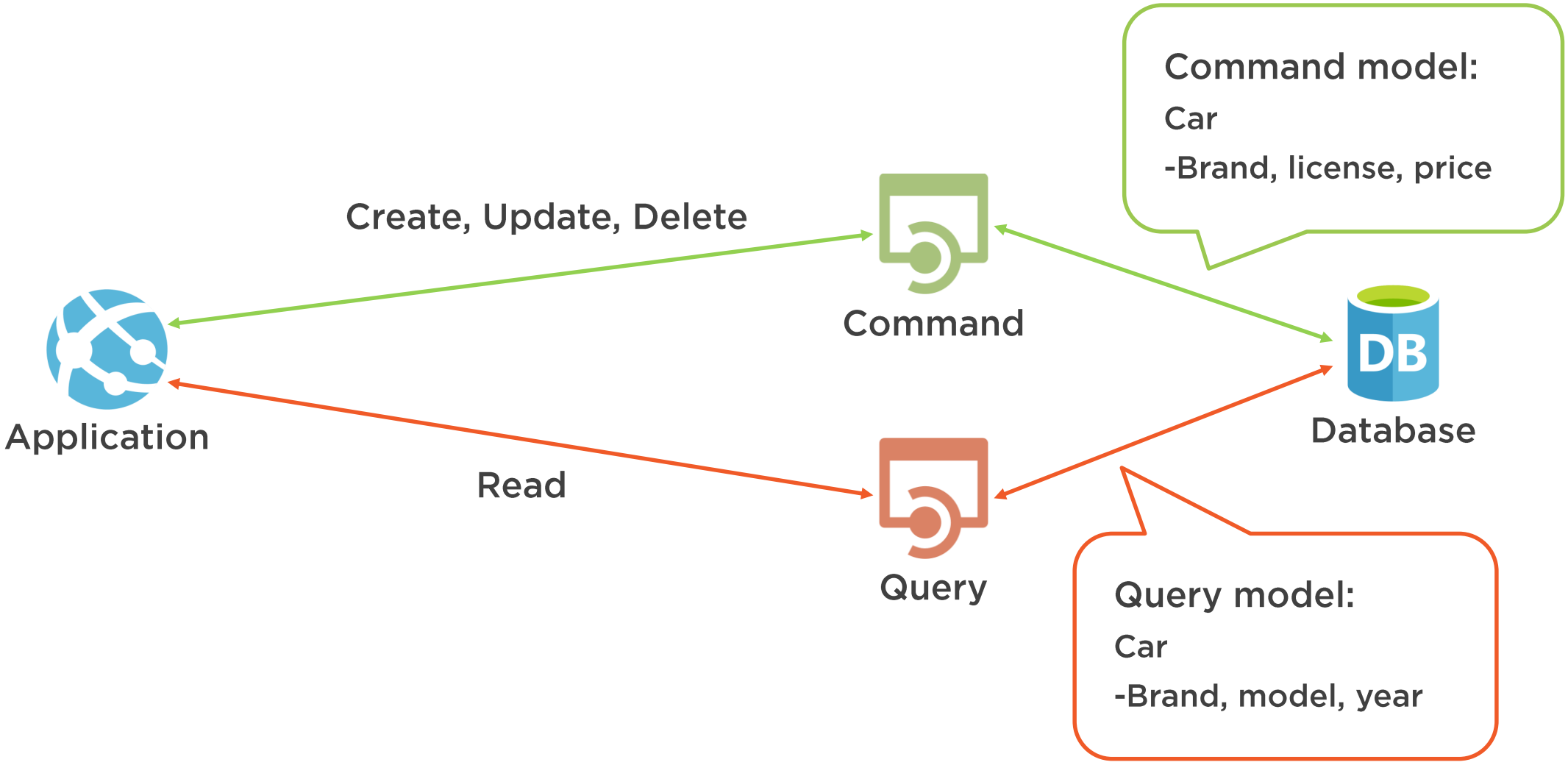


Services and data sources are expensive to scale

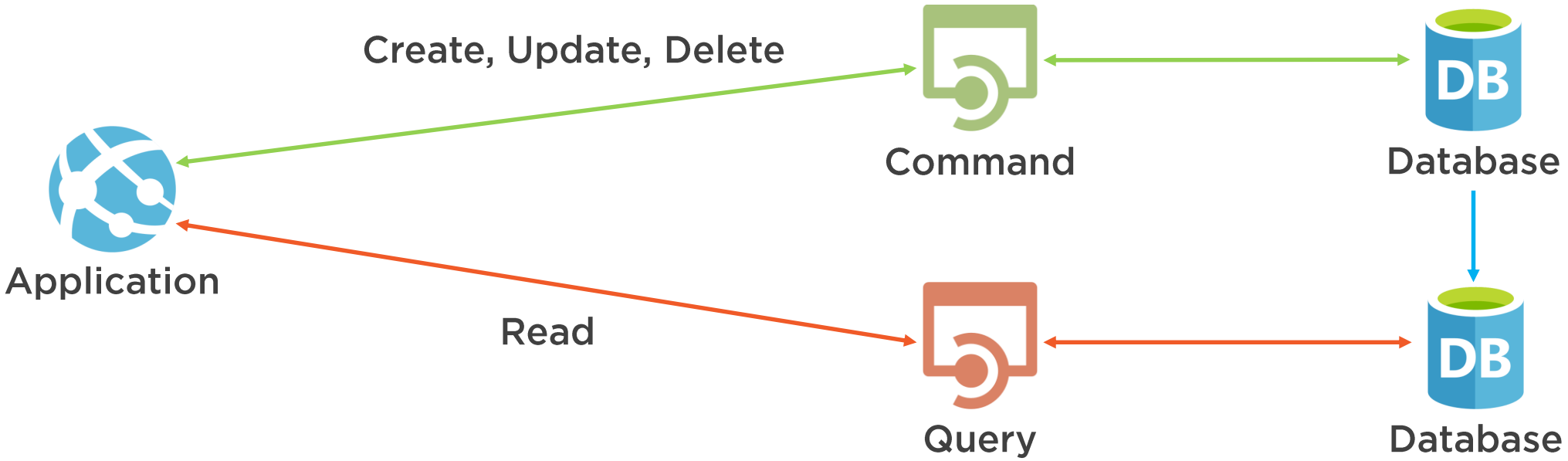
Security isn't granular

The Solution to the Problem

Command and Query Responsibility Segregation (CQRS) Pattern



Command and Query Responsibility Segregation (CQRS) Pattern



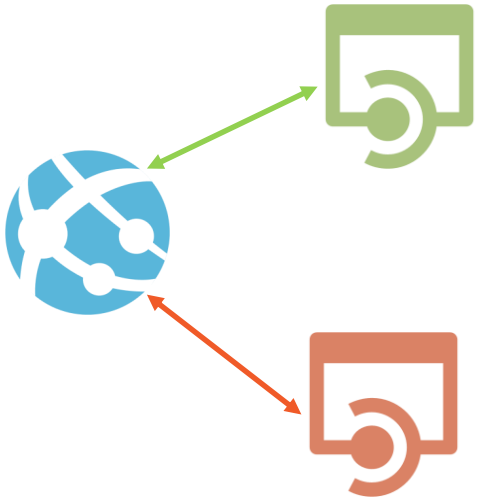
Command and Query Responsibility Segregation (CQRS) Pattern

CQRS

!= Eventual Consistency

!= Event Sourcing

Things to Consider



Separating data sources introduces

- The need for communication
- Eventual consistency

Separating command from query introduces

- Additional production components
  - Performant
  - Available
  - Secure

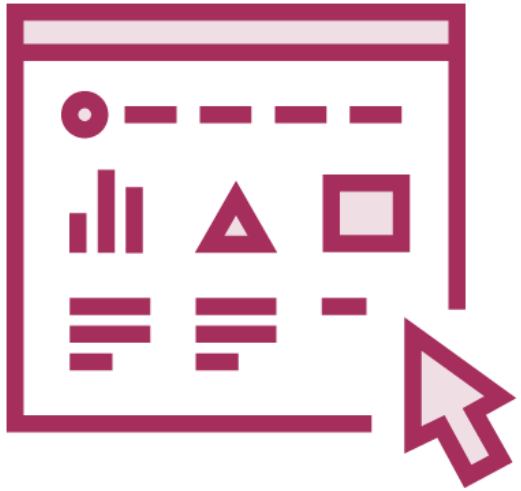
CQRS is complex

- Use it where it makes sense
- In parts of your system

When to Use This Pattern

Increase application performance

Improve security



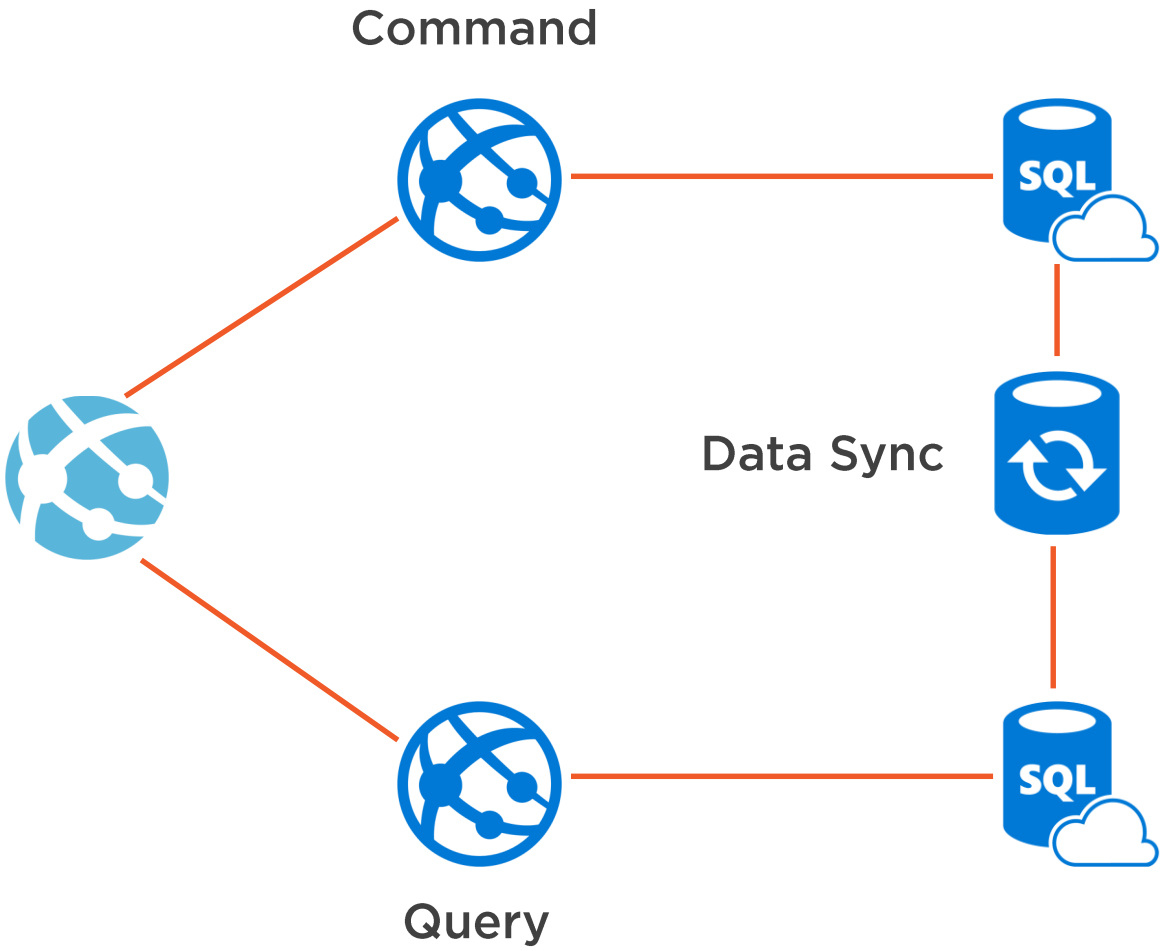
Separate read and write models

When Not to Use This Pattern

Simple domain

Simple business logic

Demo

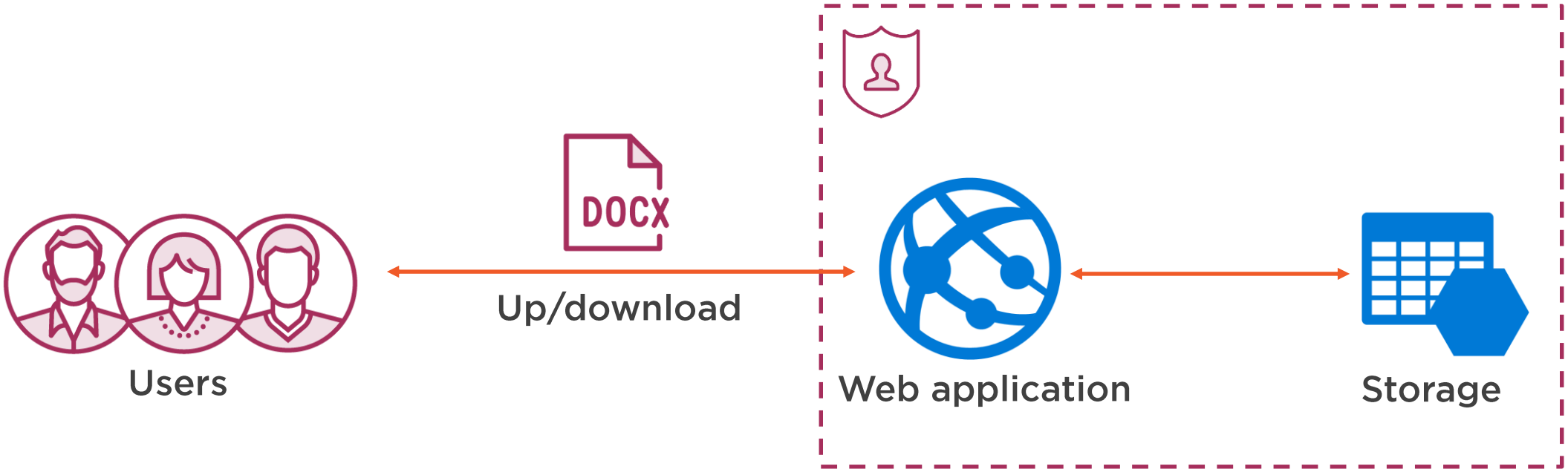


Who used CQRS?

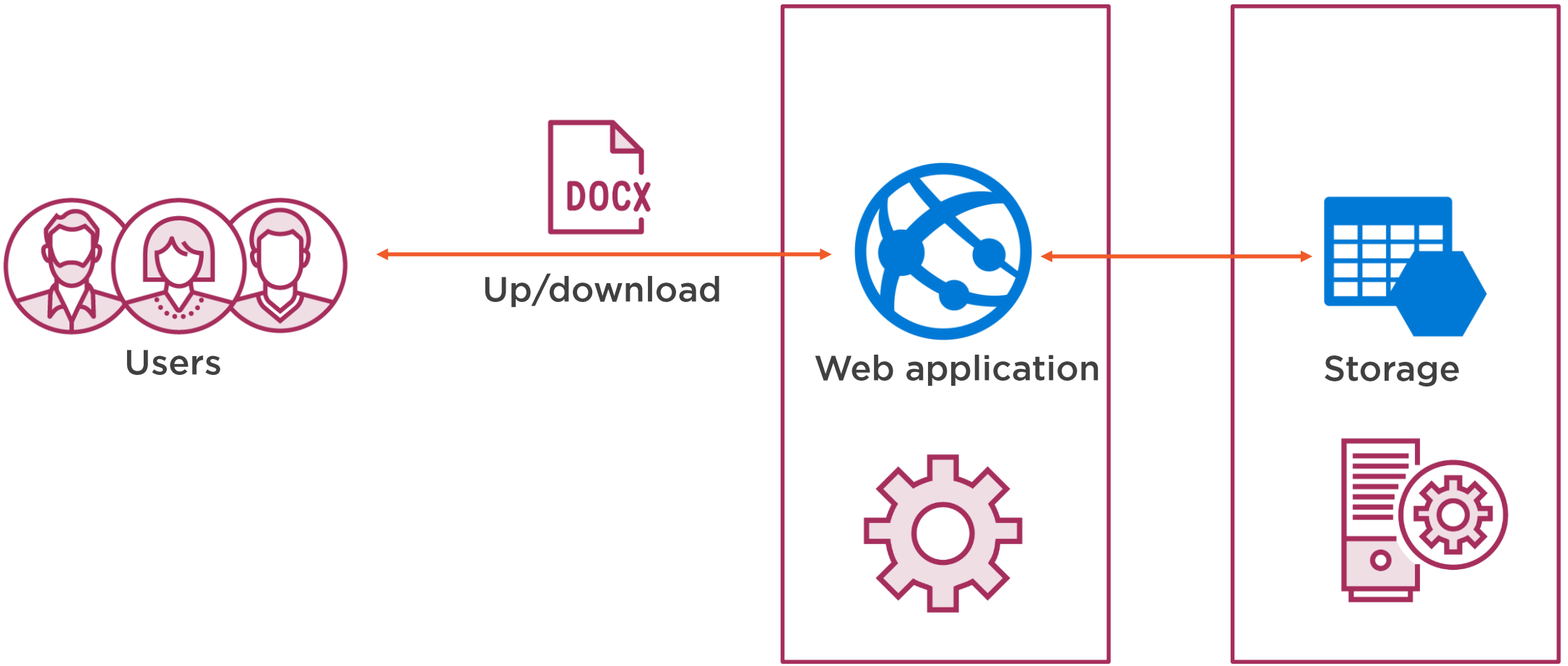

VALET

The Valet Key Pattern

The Problem with Data Transfer



The Problem with Data Transfer

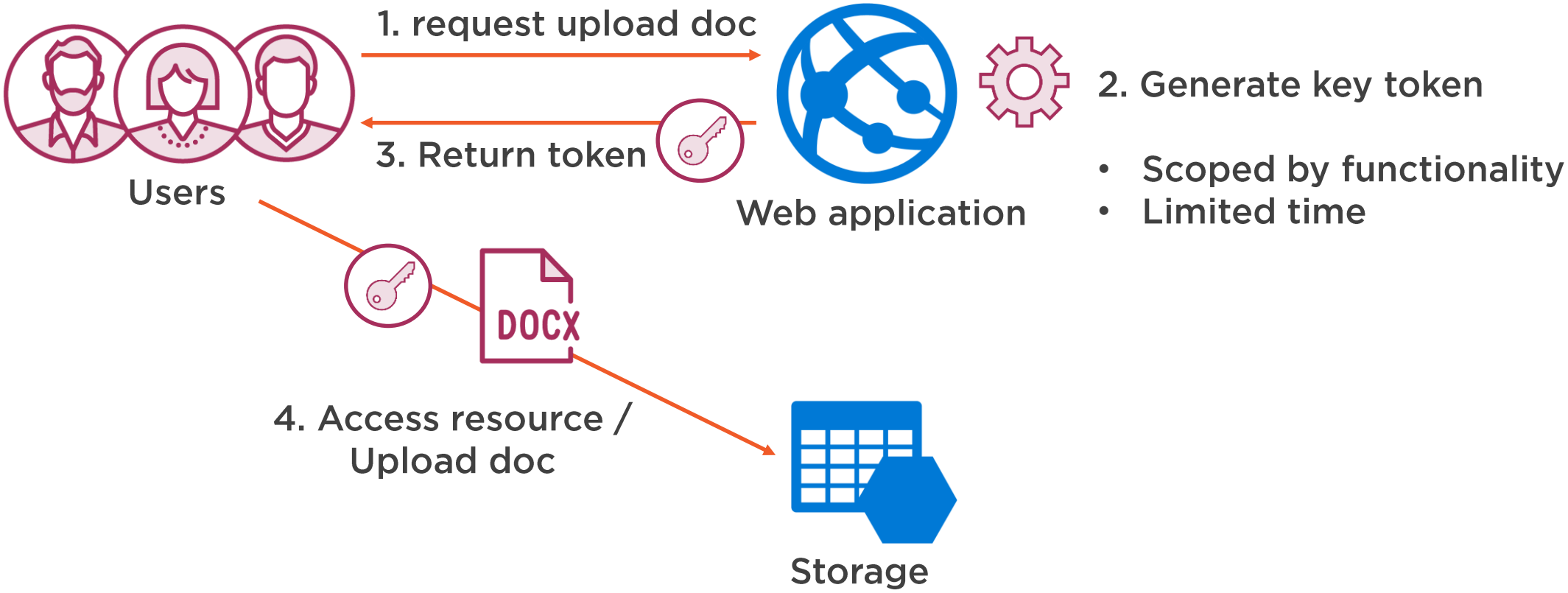


The Problem with Data Transfer

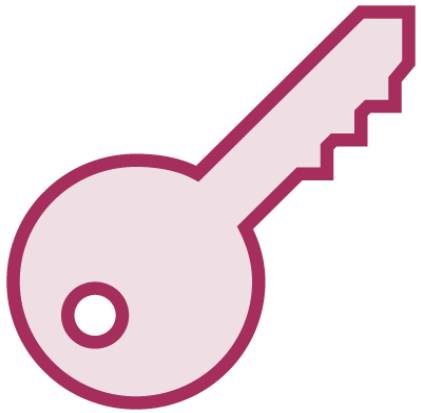
System doesn’t scale at maximum potential

The Solution to the Problem

The Valet Key Pattern



Things to Consider



Restrict the token

- By functional scope
- By time

Transfer the token securely

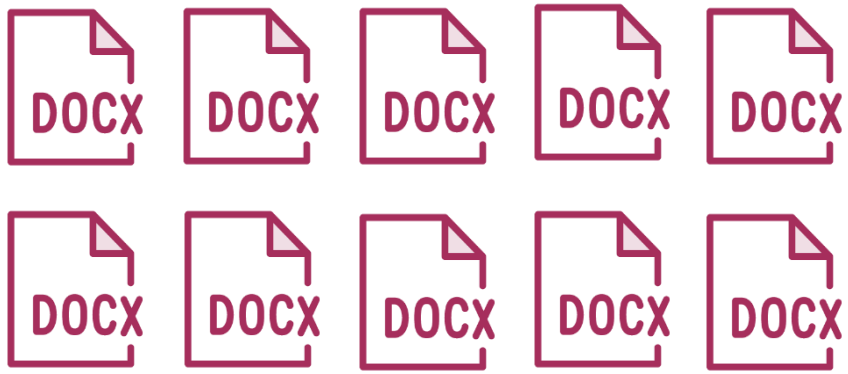
Restrict the users behavior

- By number of actions
- By metrics like amount of data

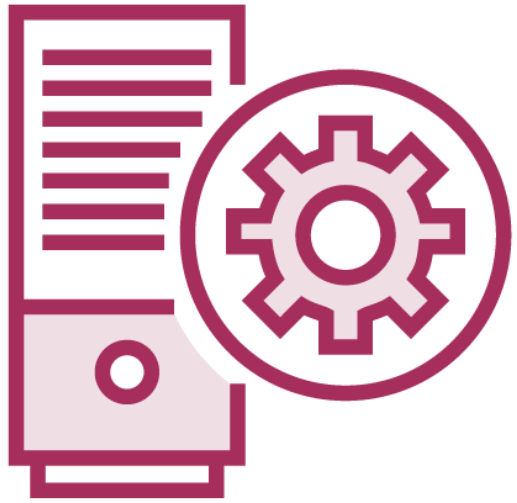
Validate all uploaded data

Audit all operations on the resource

When to Use This Pattern

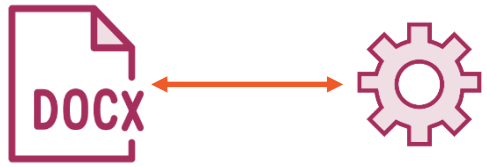


Many actions to and from external resources, like storage



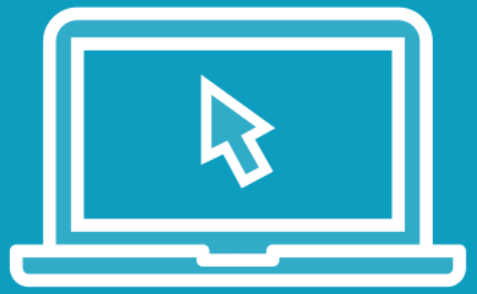
The application has limited resources

When Not to Use This Pattern



Application needs to transform data

Demo

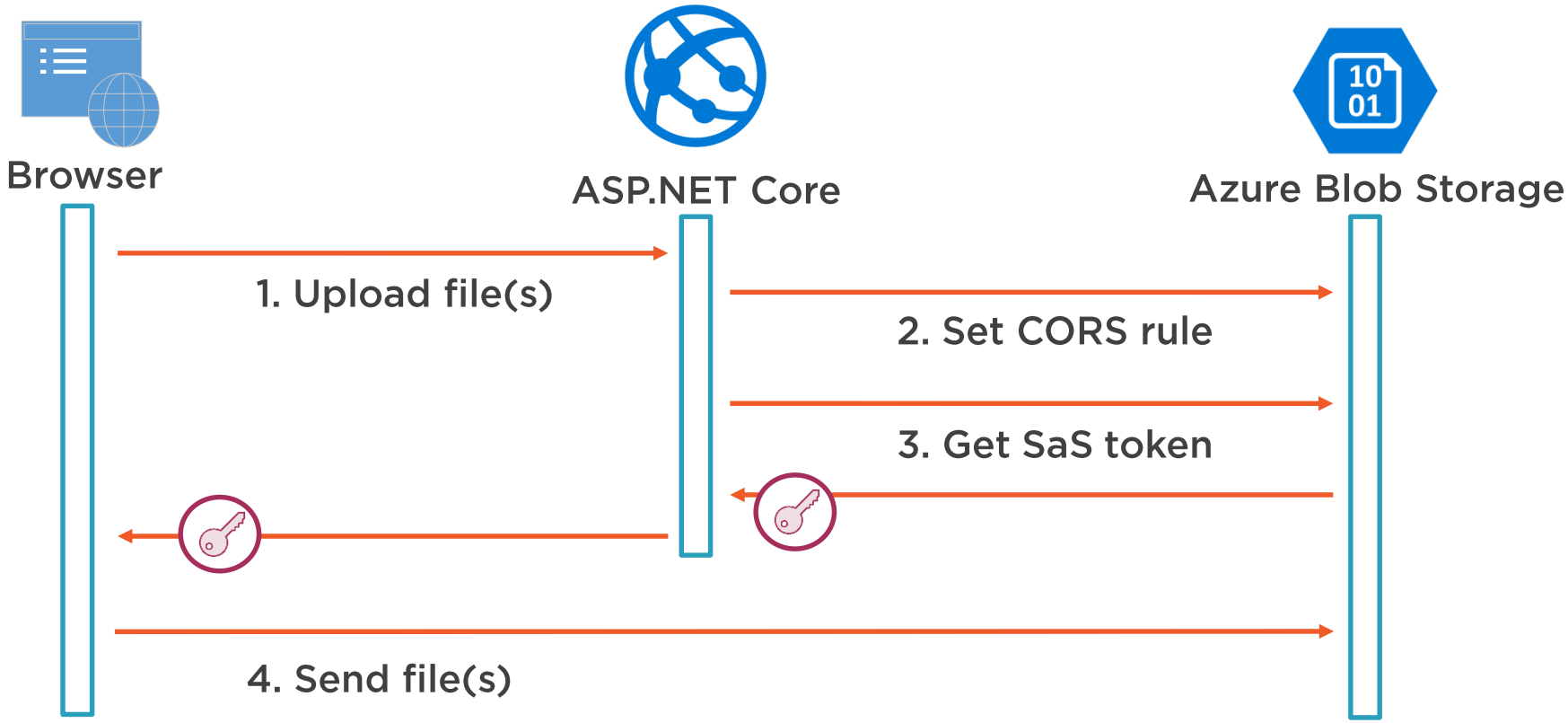


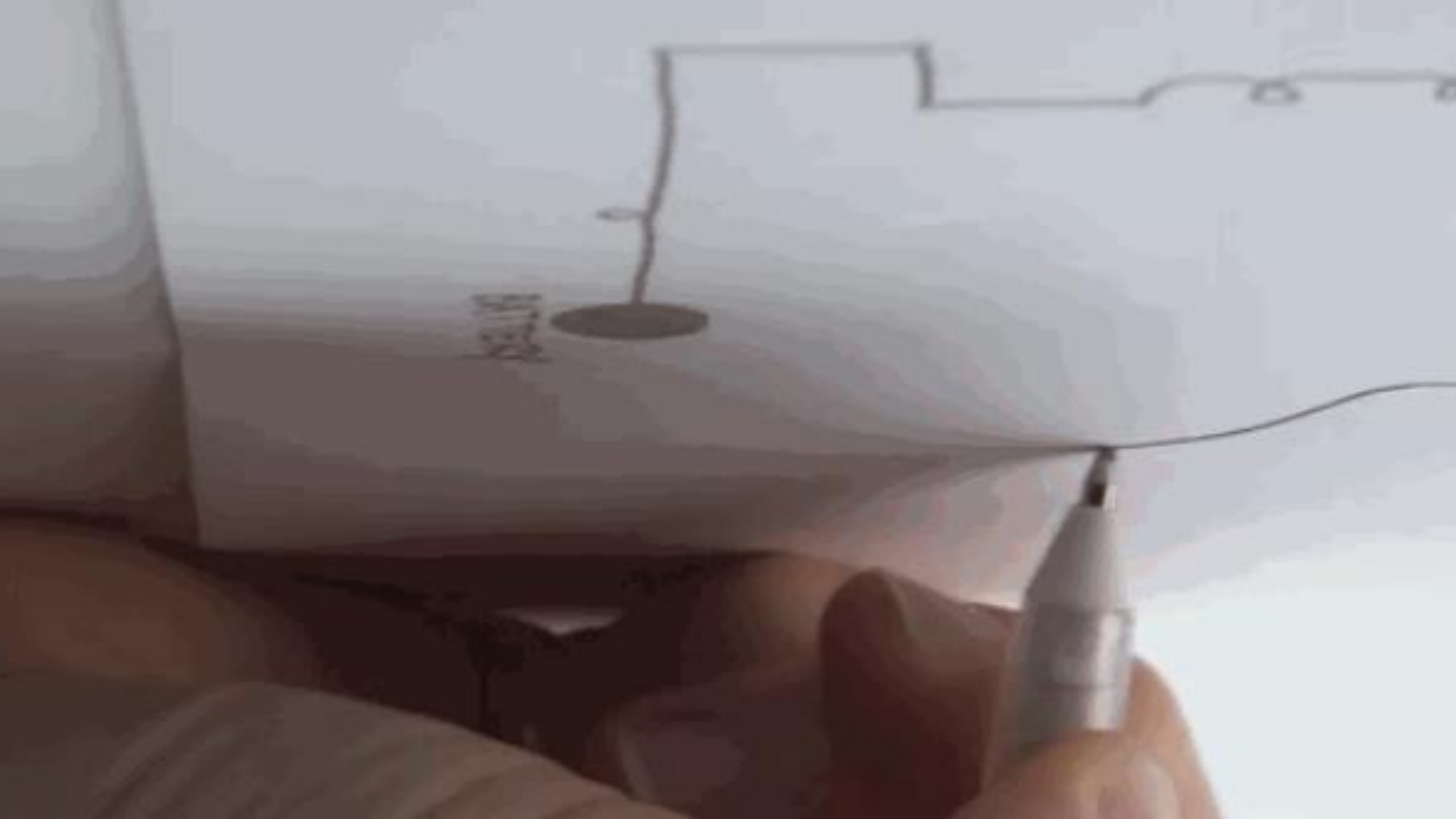
Azure Storage

Shared Access Signature (SAS)

Direct upload

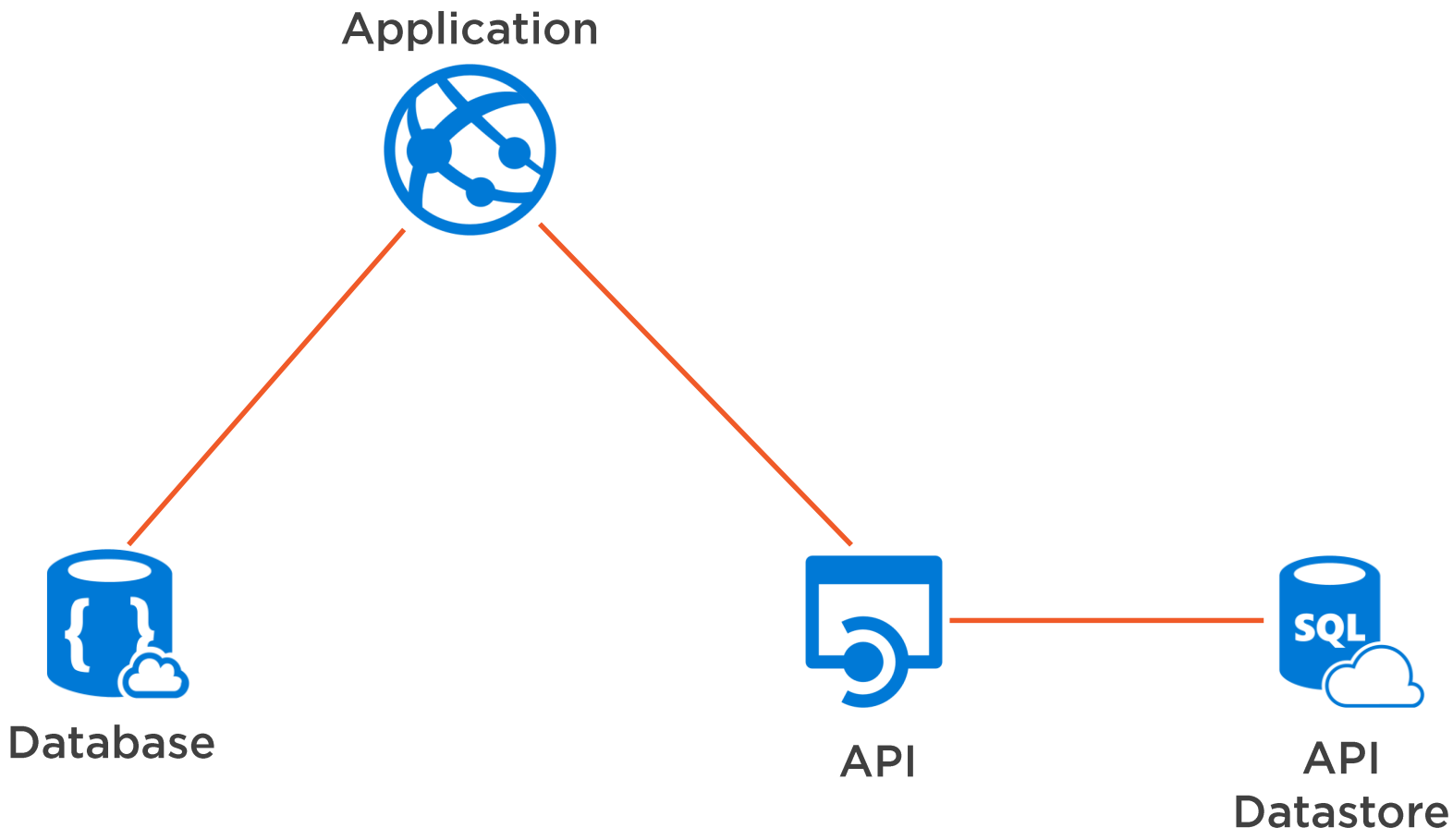
The Valet Key Pattern Demo



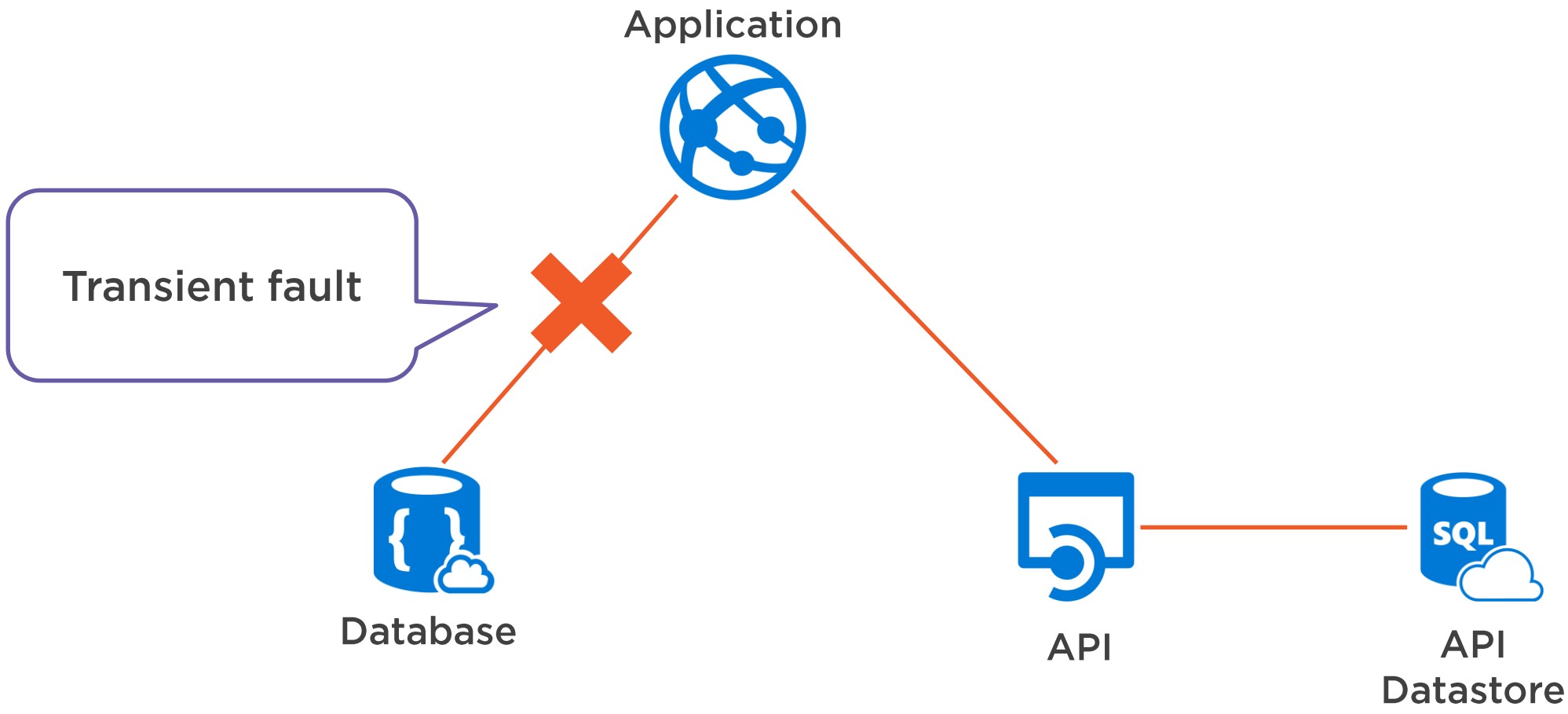


The Circuit Breaker Pattern

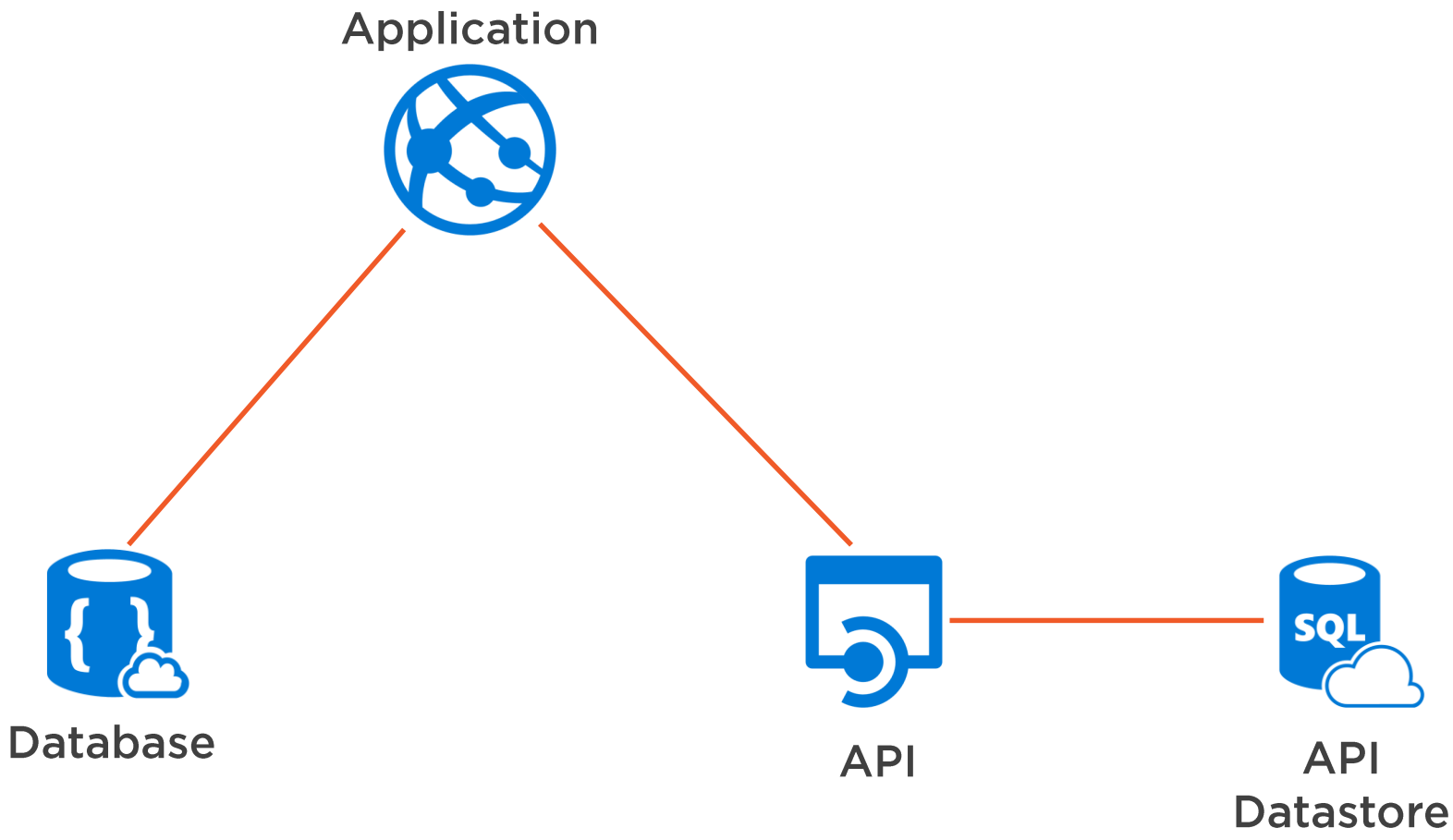
The Problem with Non-transient Failures



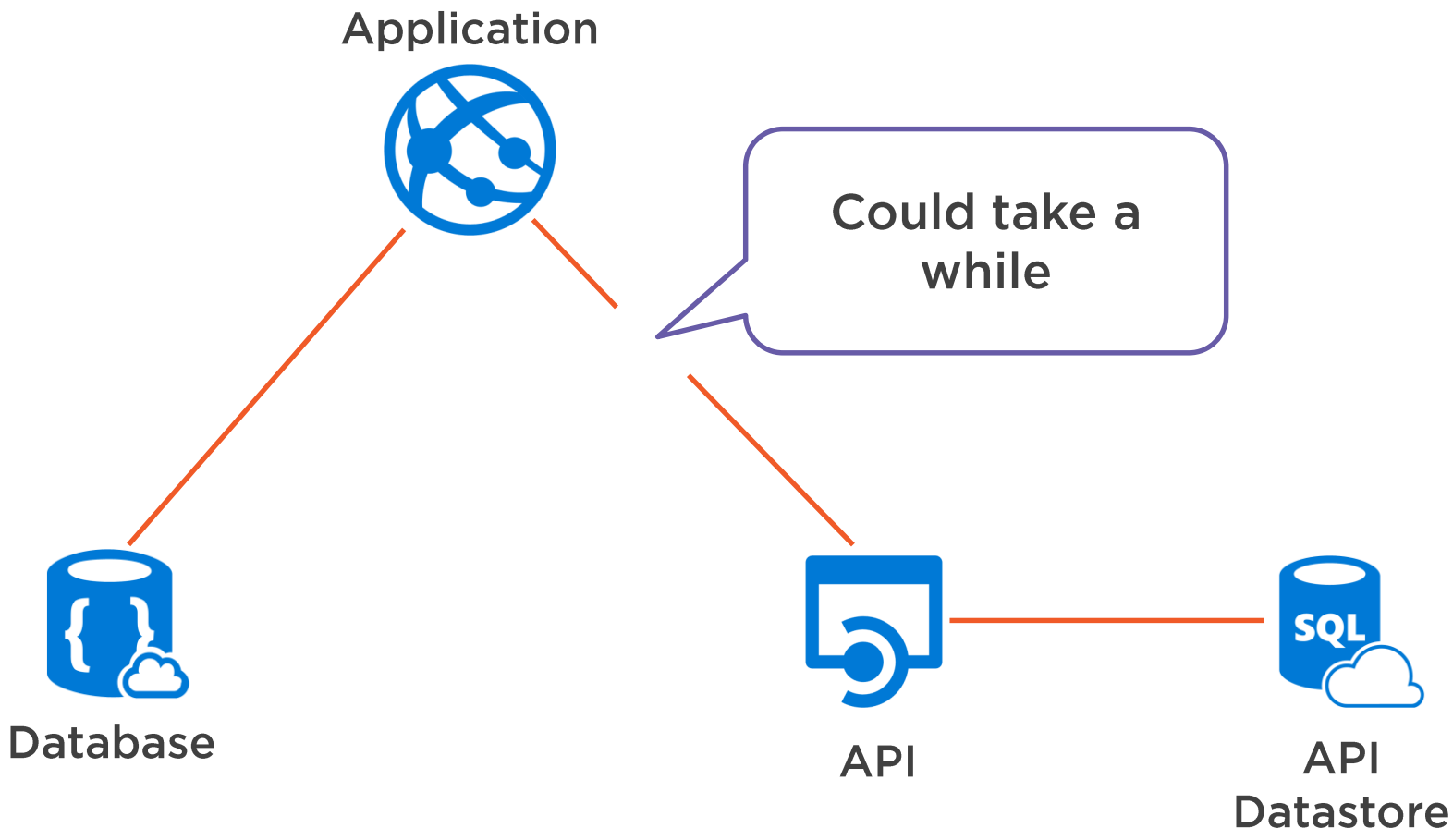
The Problem with Non-transient Failures



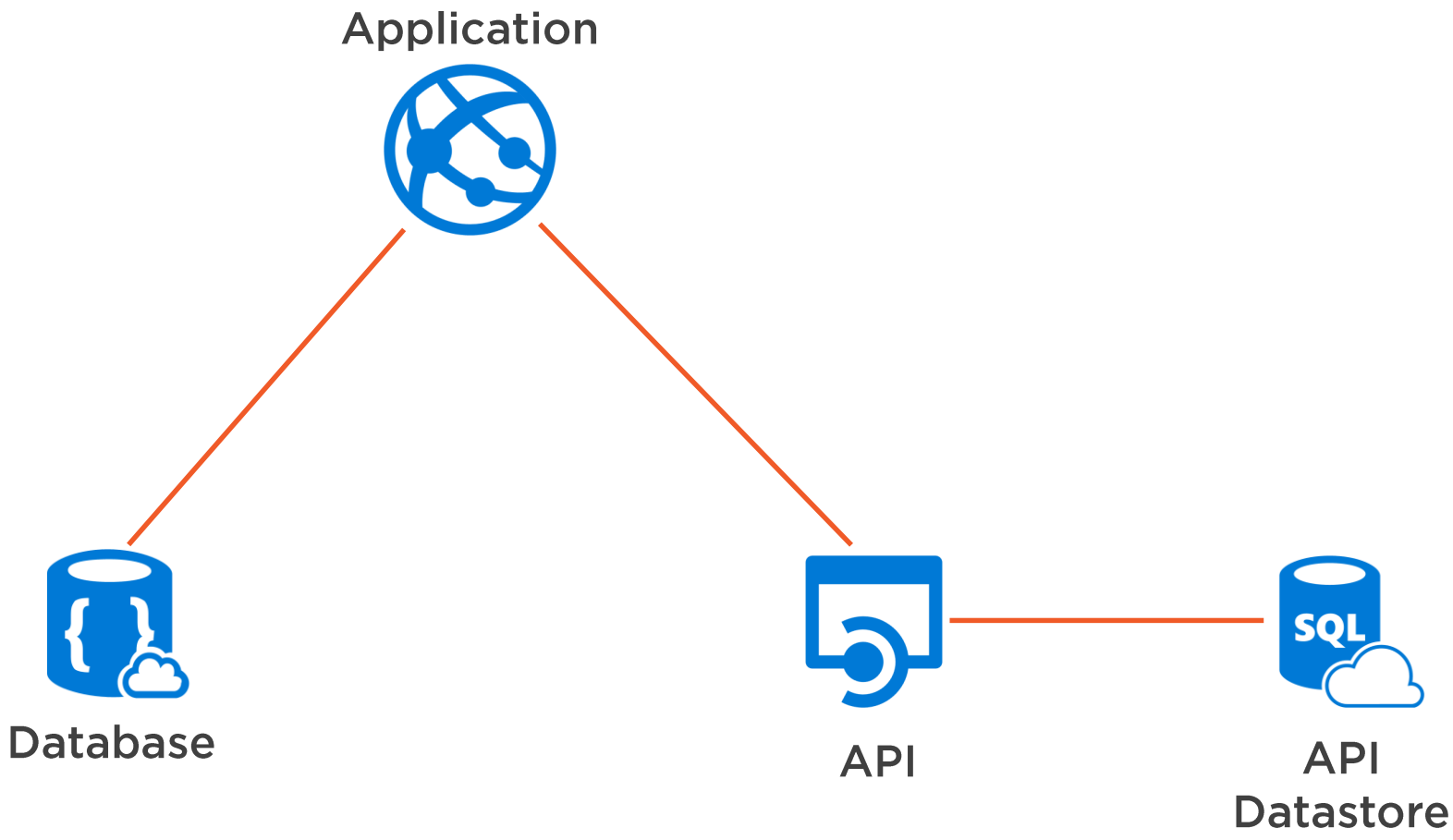
The Problem with Non-transient Failures



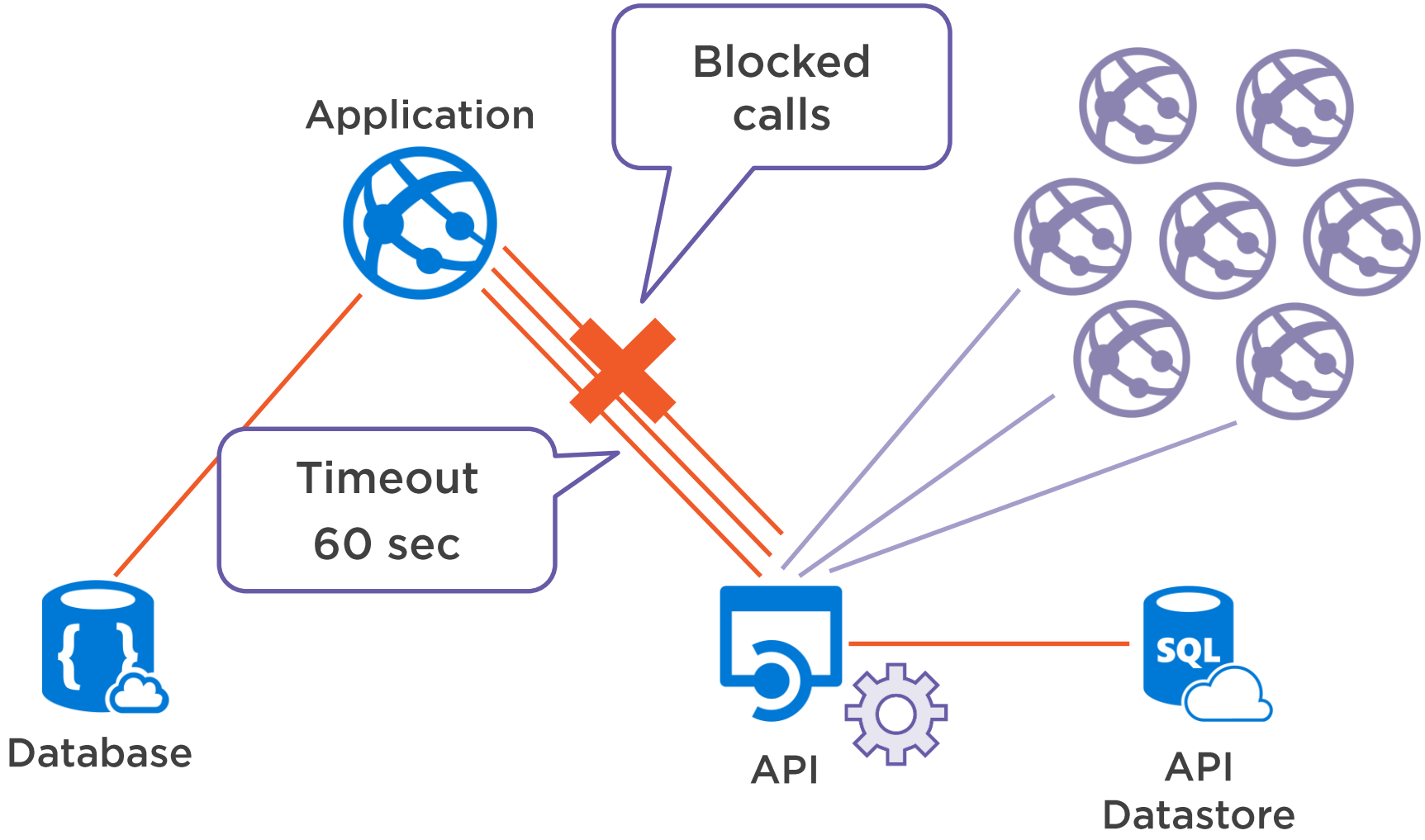
The Problem with Non-transient Failures



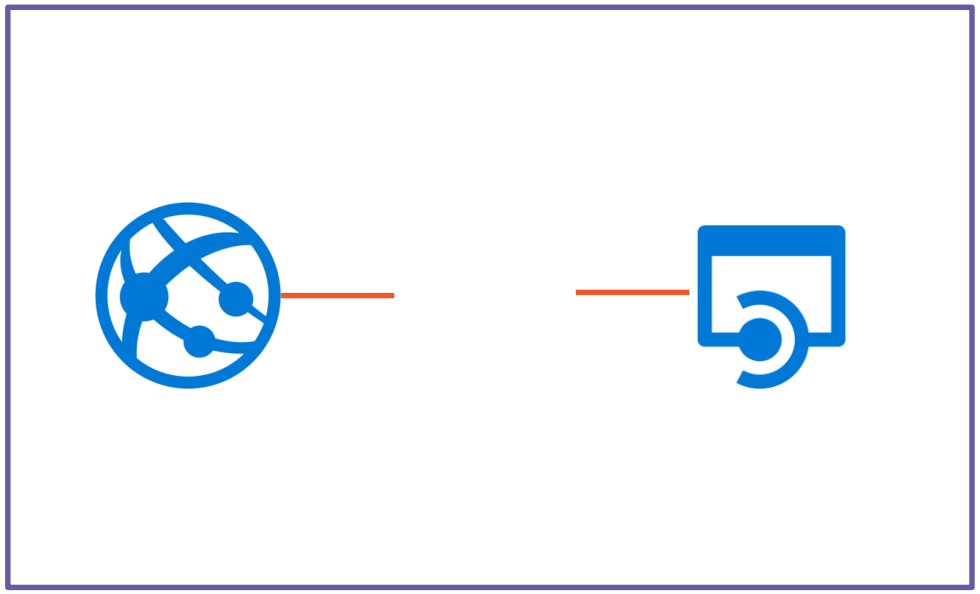
The Problem with Non-transient Failures



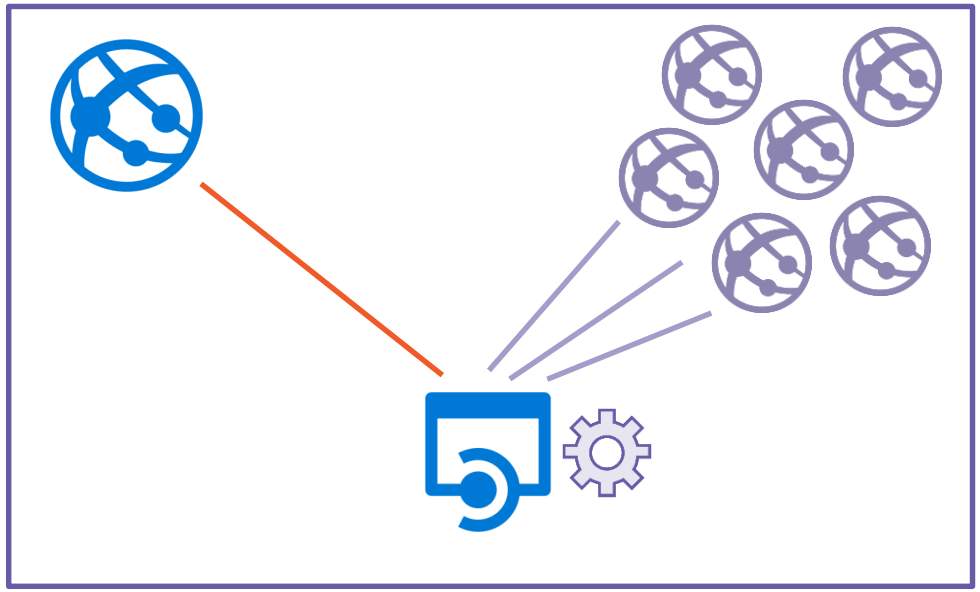
The Problem with Non-transient Failures



The Problem with Non-transient Failures



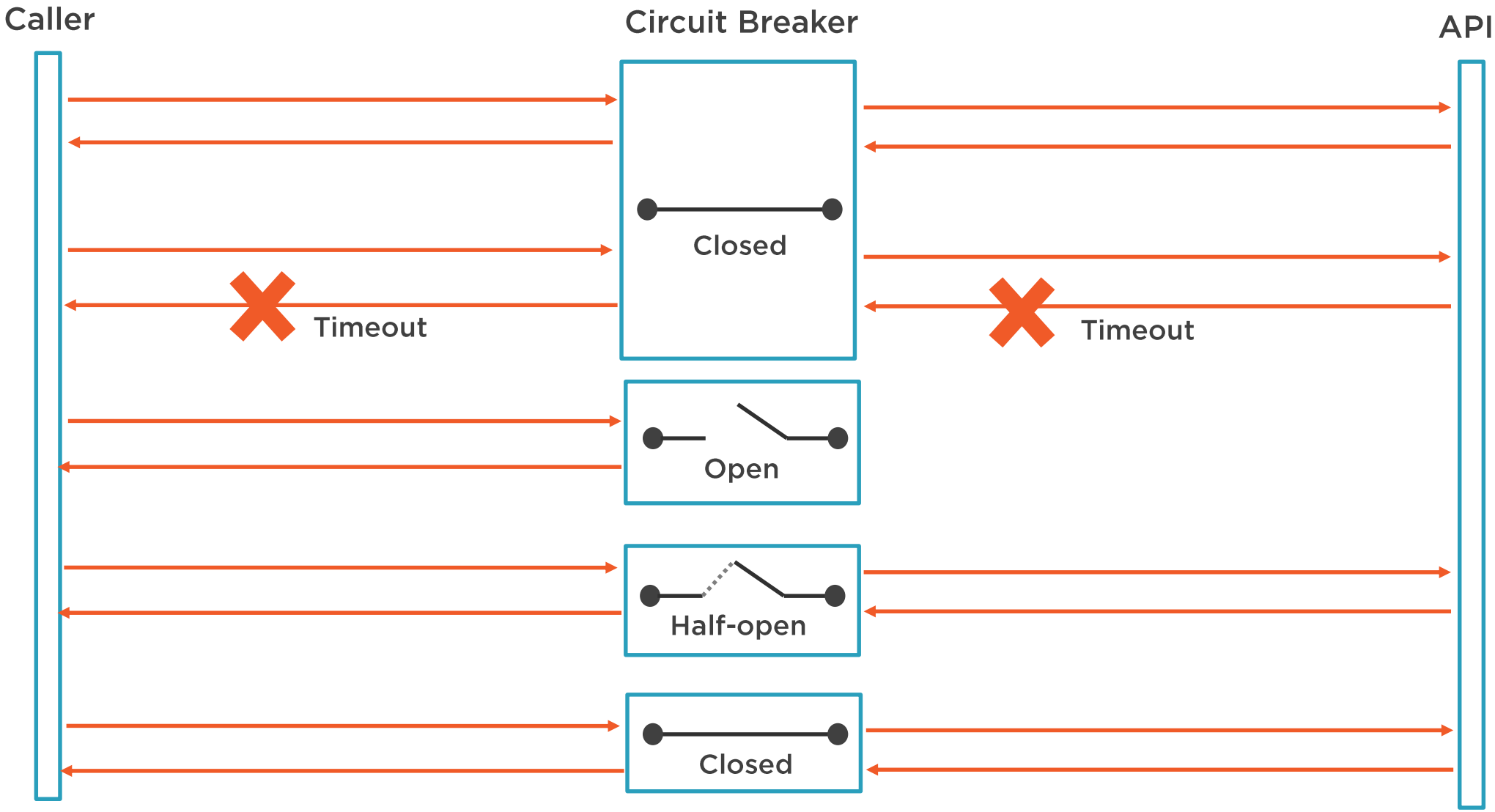
Retrying wastes resources



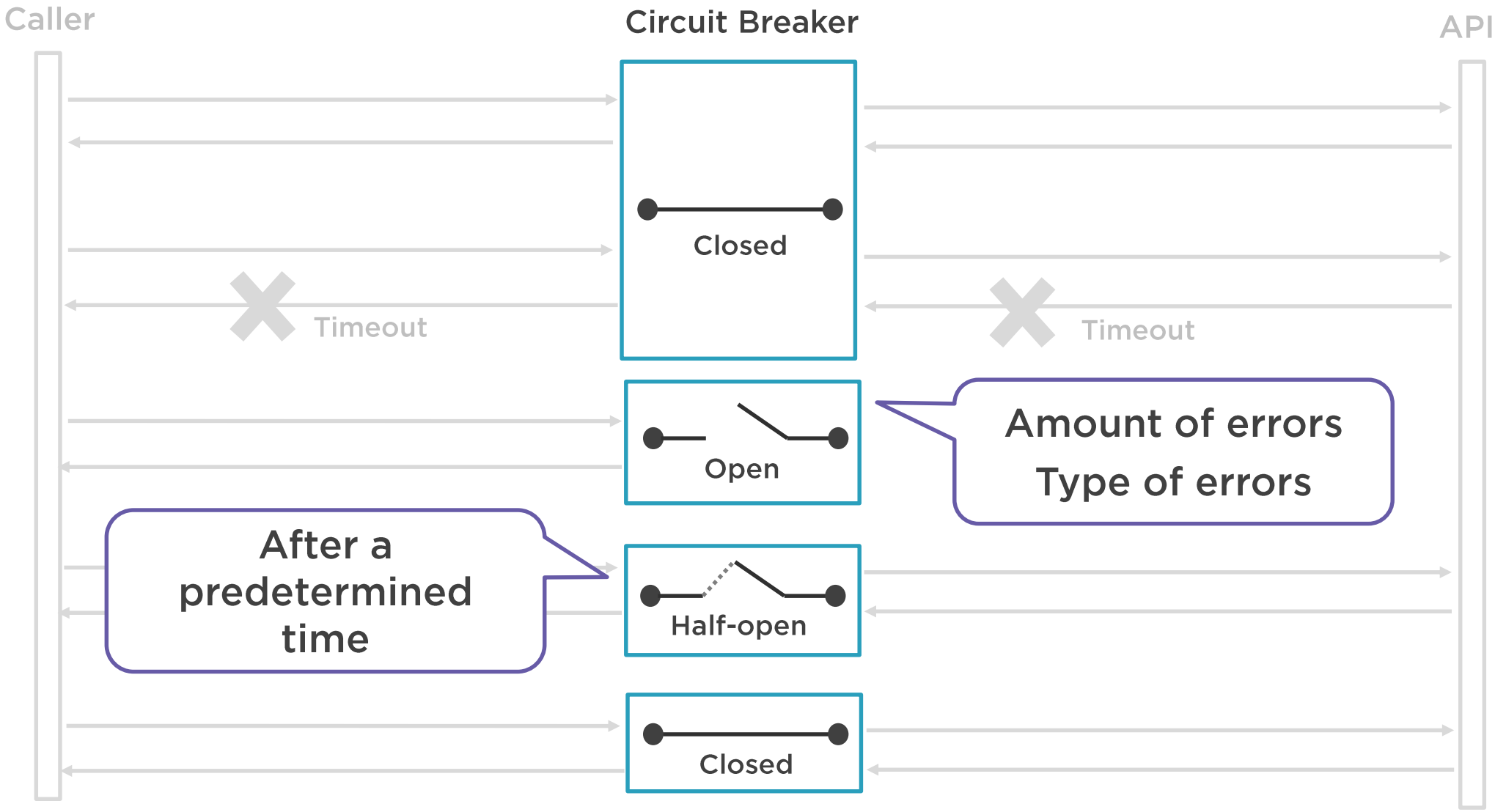
Waiting on a timeout wastes resources

The Solution to the Problem

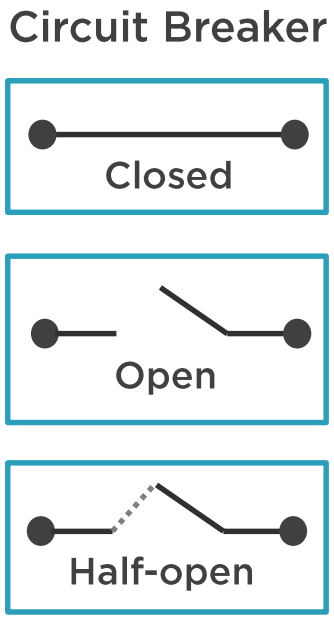
The Circuit Breaker Pattern



The Circuit Breaker Pattern



Things to Consider



Calling code

- Needs to be able to deal with exceptions
- Exceptions need to be well-defined

The circuit breaker can be smart

- Examine failures & change strategy

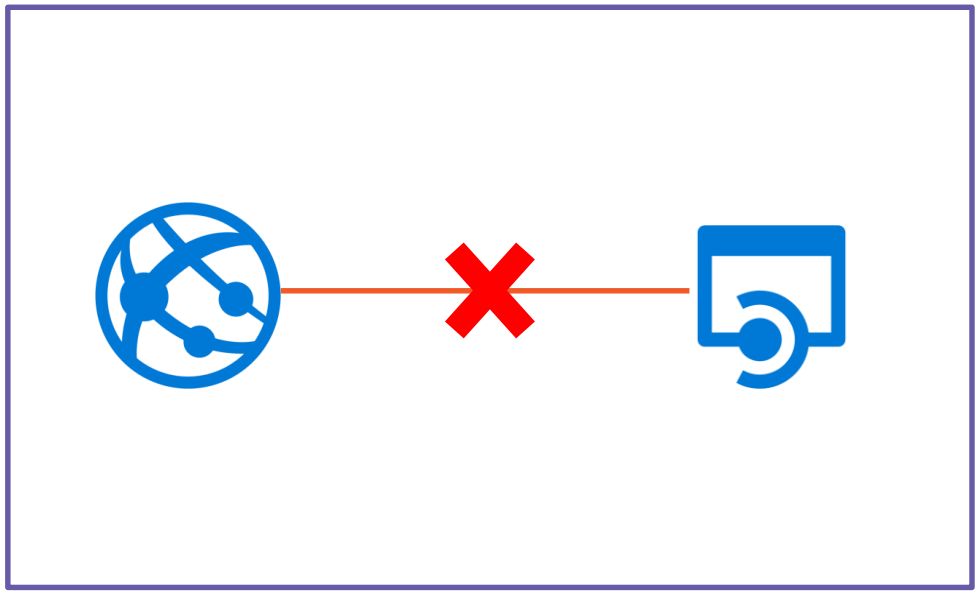
Health logging

- (Failed) requests should be logged

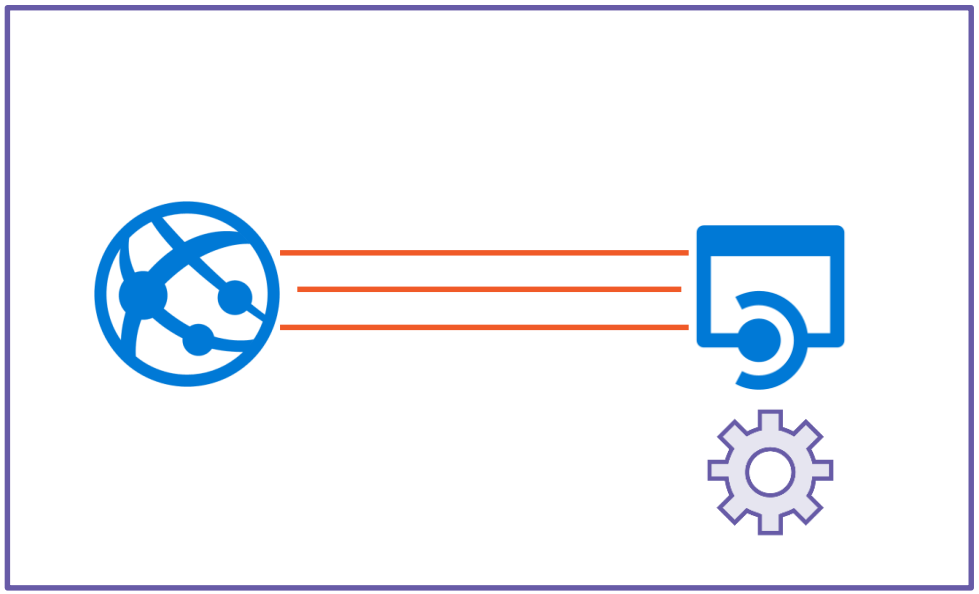
Ability to manually override state

Deal with concurrent requests

When to Use This Pattern



Prevent calls that are likely to fail



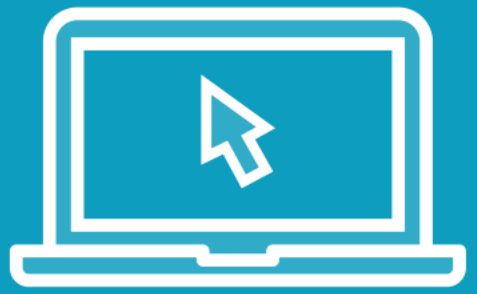
Prevent recovering system dependencies from being overloaded

Implementing the Circuit Breaker Pattern

Write your own

https://github.com/App-vNext/Polly

Demo



Console application

Simple Circuit Breaker Pattern


Who uses
Circuit Breaker?


EVENTS

The Event Sourcing Pattern

The Problem with Storing the Current State of the Data

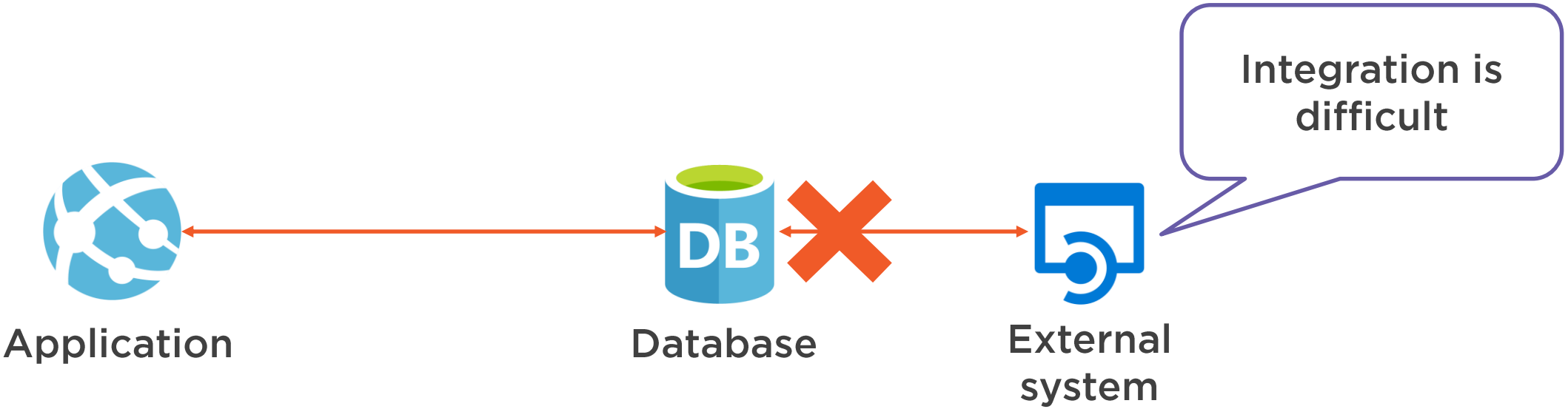
Id	Article	Amount
1	E-learning	1
2	Software	1
3	Books	3

The Problem with Storing the Current State of the Data

Data conflicts

Id	Article	Amount
1	E-learning	1
2	Software	1
3	Books	3

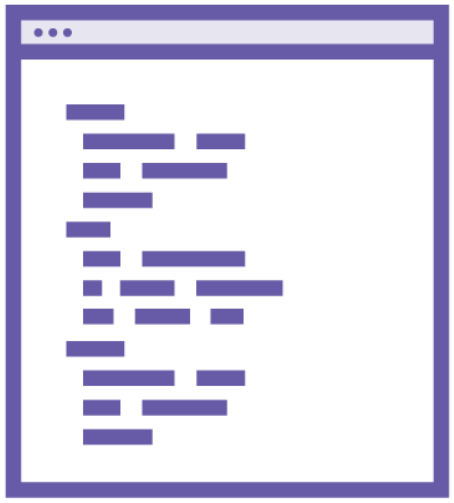
The Problem with Storing the Current State of the Data



Id	Article	Amount
1	E-learning	1
2	Software	1
3	Books	3

The Problem with Storing the Current State of the Data

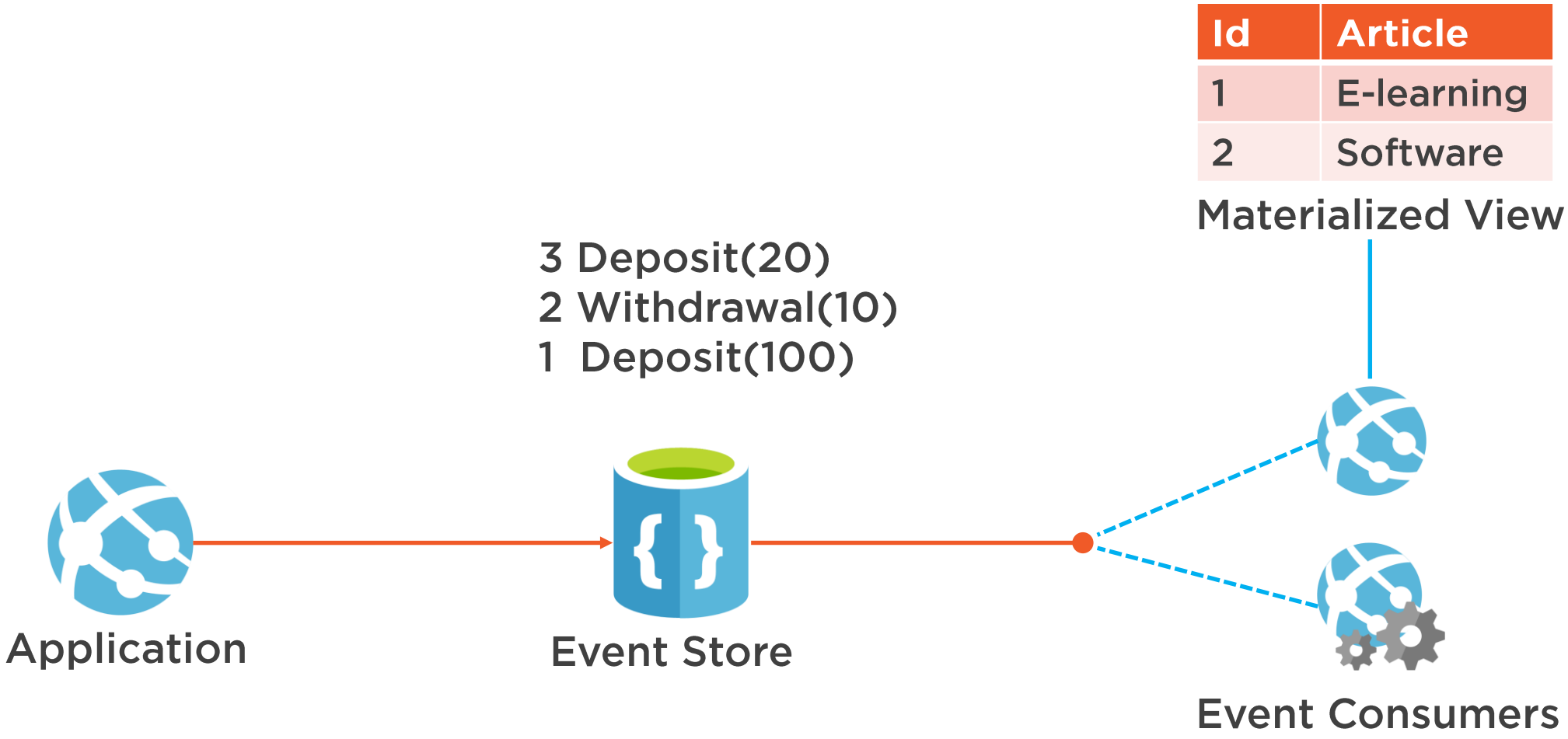
Data conflicts



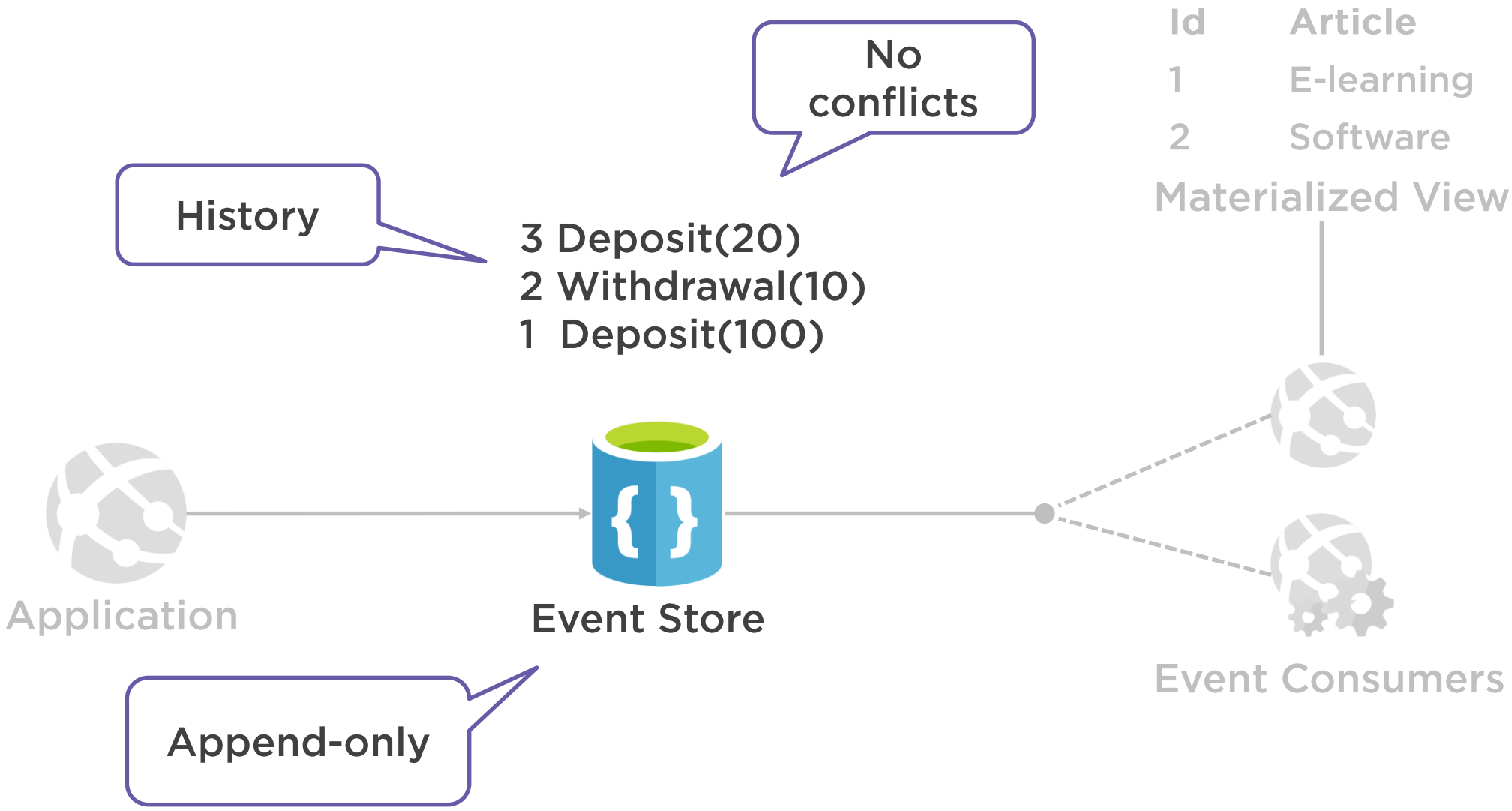
Shaping data is difficult

The Solution to the Problem

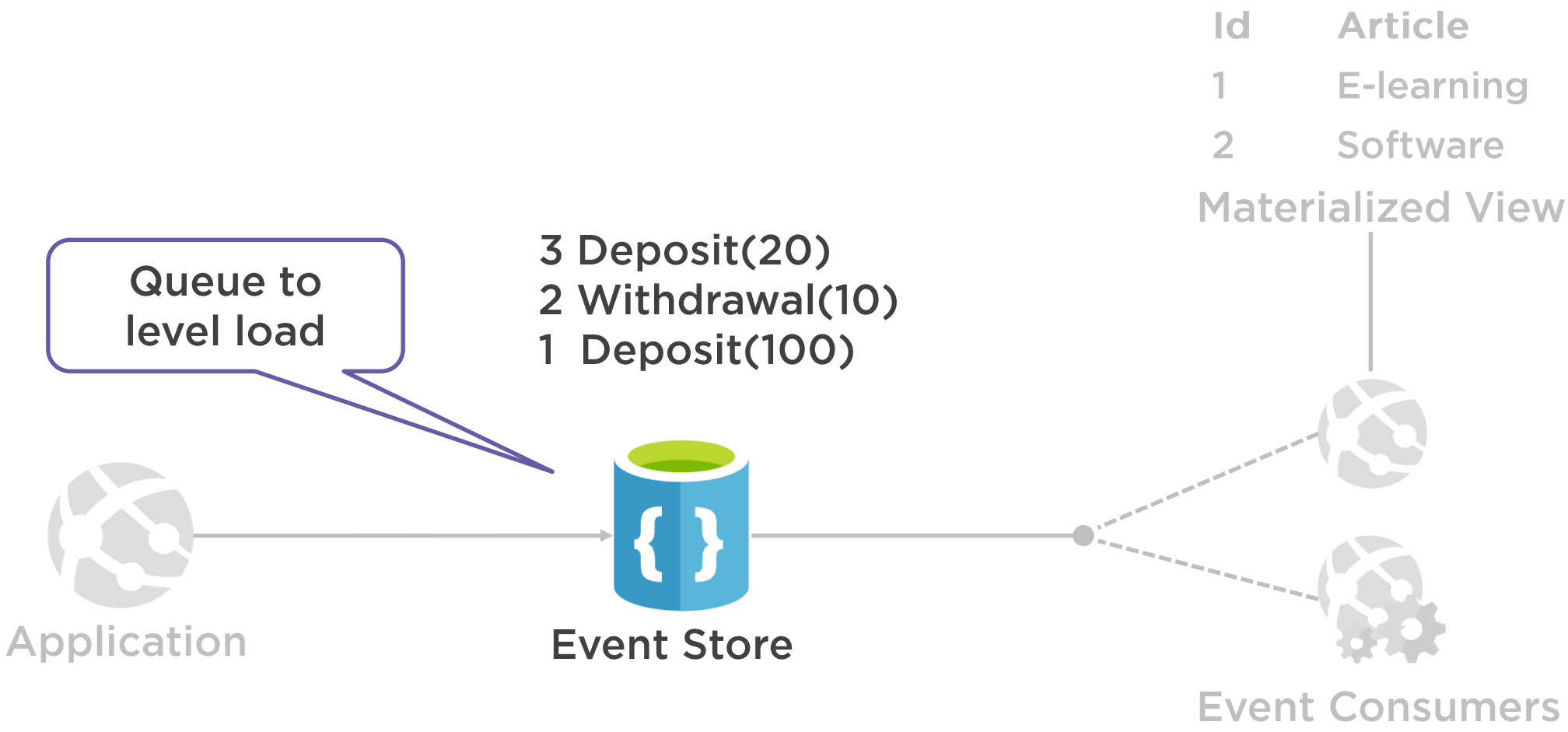
The Event Sourcing Pattern



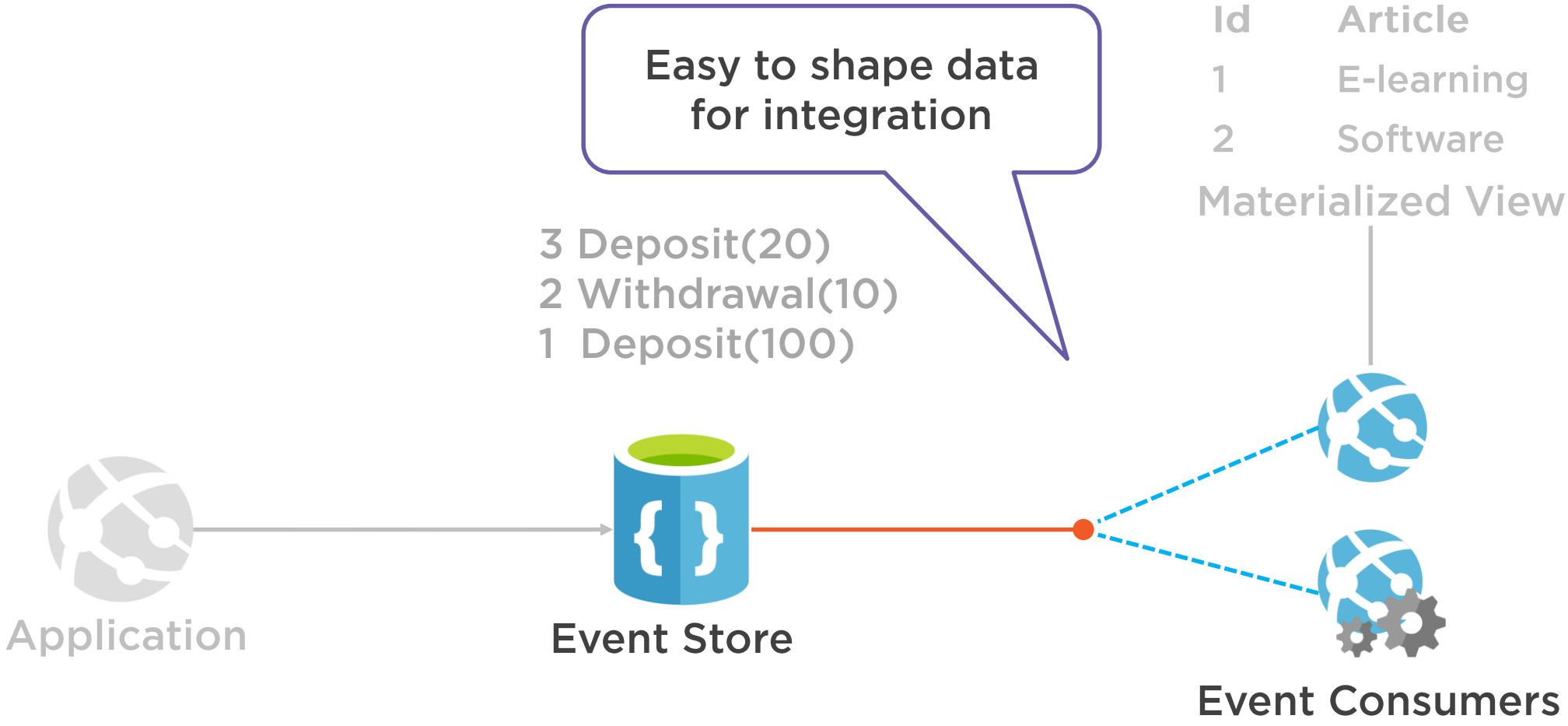
The Event Sourcing Pattern



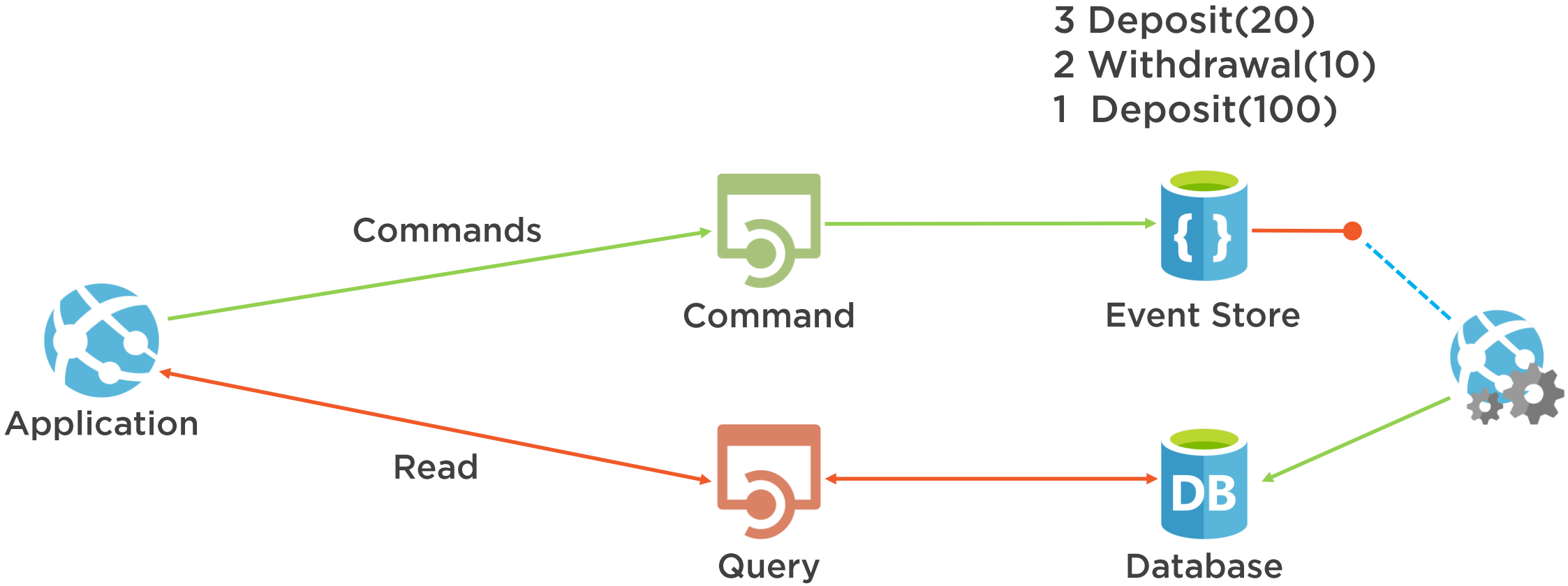
The Event Sourcing Pattern



The Event Sourcing Pattern



Event Sourcing and CQRS



Things to Consider

3 Deposit(20)
2 Withdrawal(10)
1 Deposit(100)

The event store is a production system

- Backup / restore

Get the current state of the data

- By replaying events

You should annotate events with

- A sequence and a version

Compensate events by adding events

Lag in publishing and processing events

- Will cause eventual consistency

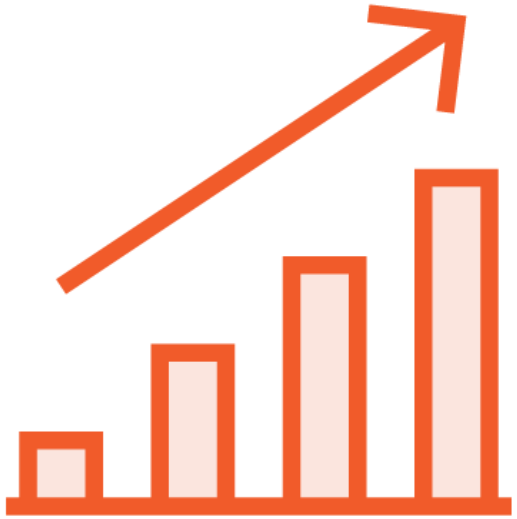
Event sourcing is complex to implement

When to Use This Pattern

3 Deposit(20)

2 Withdrawal(10)

1 Deposit(100)



3 Deposit(20)

2 Withdrawal(10)

1 Deposit(100)

Capture purpose in data

Improve performance by decoupling input and processing

Avoid or minimize data conflicts

Complete data history

When Not to Use This Pattern

When your system has a simple or small domain

When your system implements strong transactional consistency

Demo

Console application

Event Sourcing pattern

Food for Thought

Where to Find the Demo Files

https://github.com/bmaluijb/CloudDesignPatterns

Find me on Pluralsight

- Introduction to Azure App Services
- Building a Global App with Azure PaaS
- Continuous Integration and Continuous Delivery: The Big Picture

- Cloud Design Patterns for Azure: Design and Implementation
- Cloud Design Patterns for Azure: Availability and Resilience
- Cloud Design Patterns for Azure: Data Management and Performance

- The .NET Ecosystem: The Big Picture
- Microsoft Azure for Developers: What to Use When?
- Microsoft Azure Cognitive Services: The Big Picture