Synchronizing the Commands and Queries Databases

Vladimir Khorikov

@vkhorikov www.enterprisecraftsmanship.com

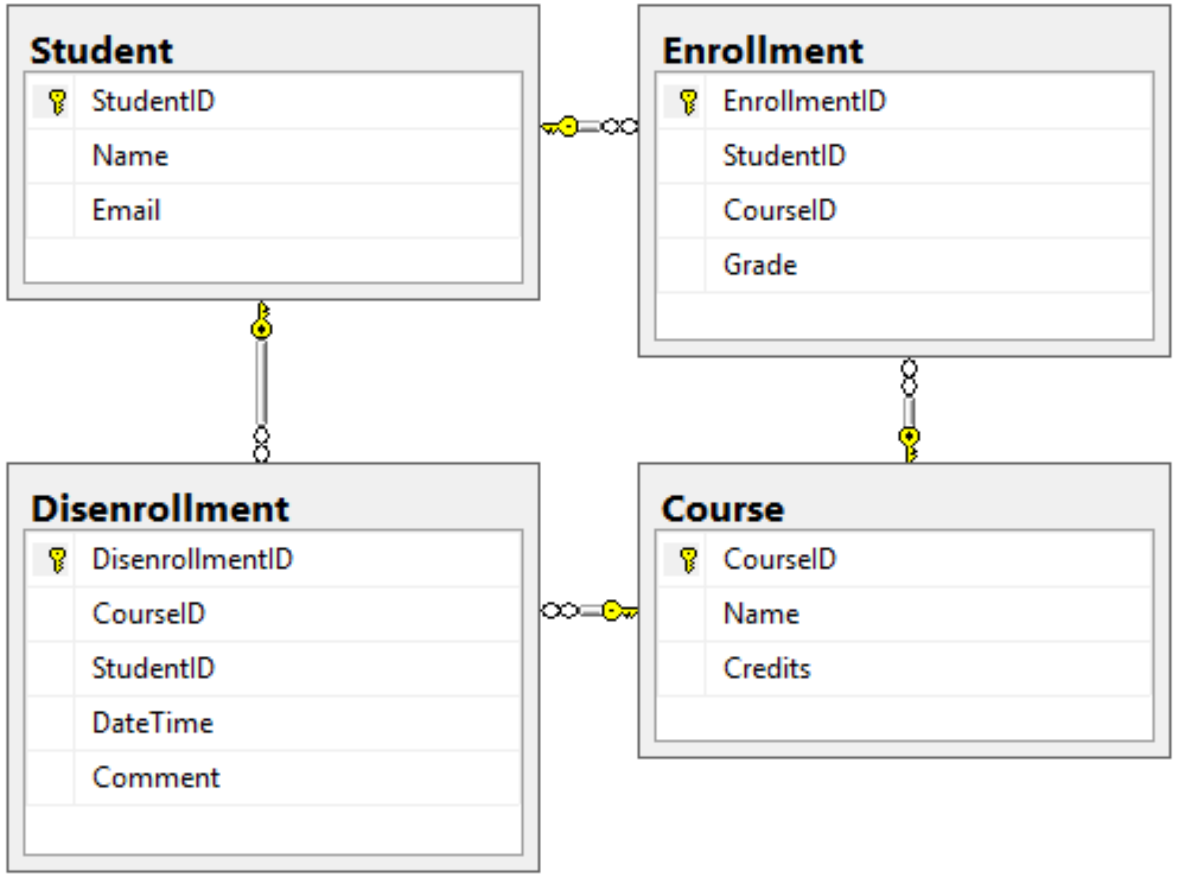
Agenda

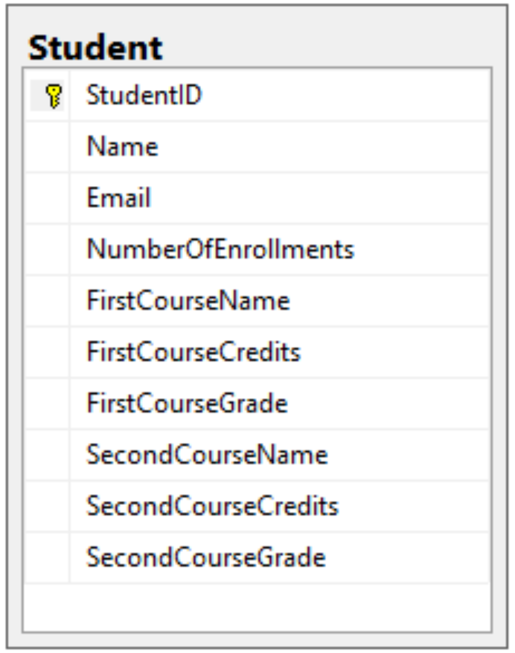
Synchronization

Eventual consistency

The CAP theorem

State-driven Projections

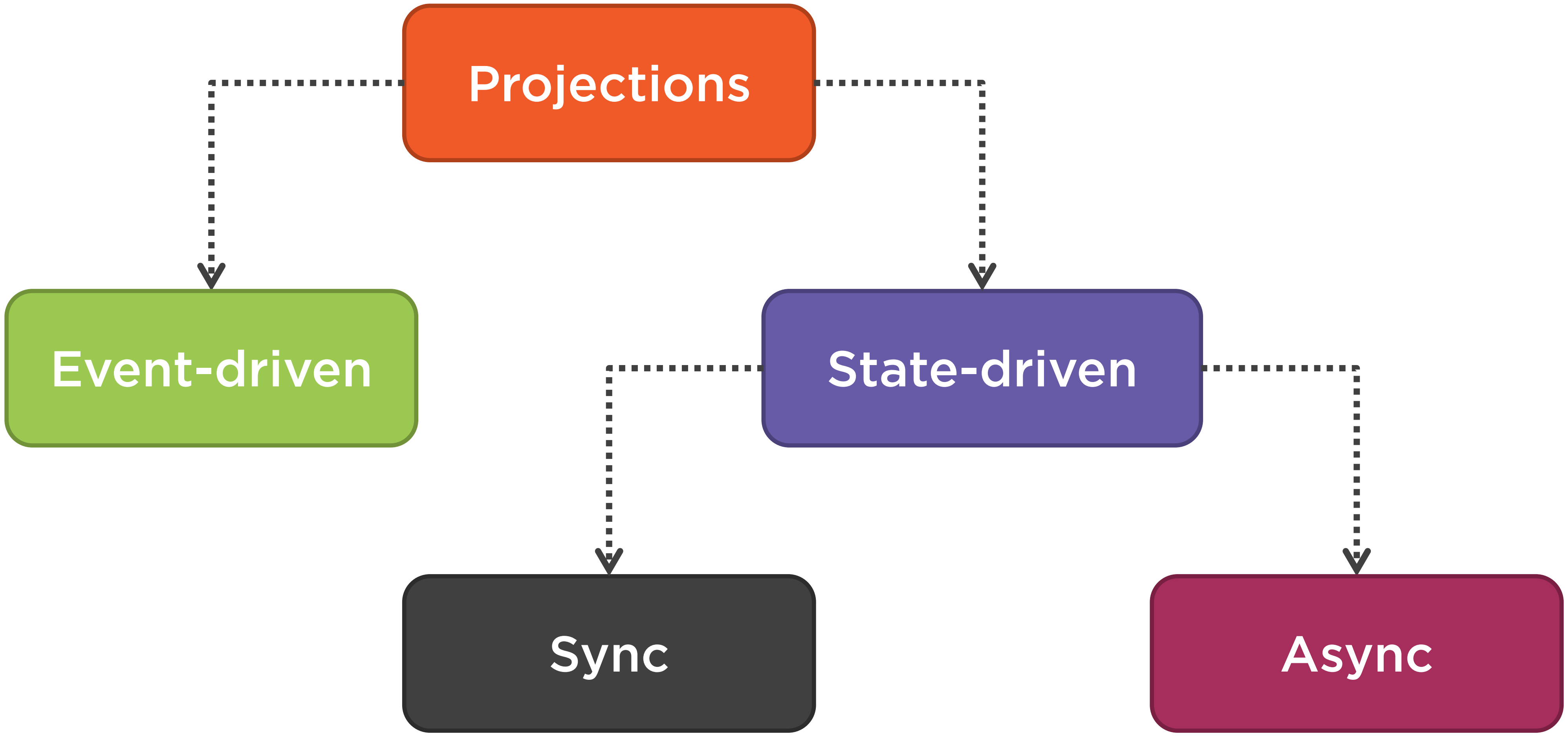




Denormalized

Need a projection

State-driven Projections



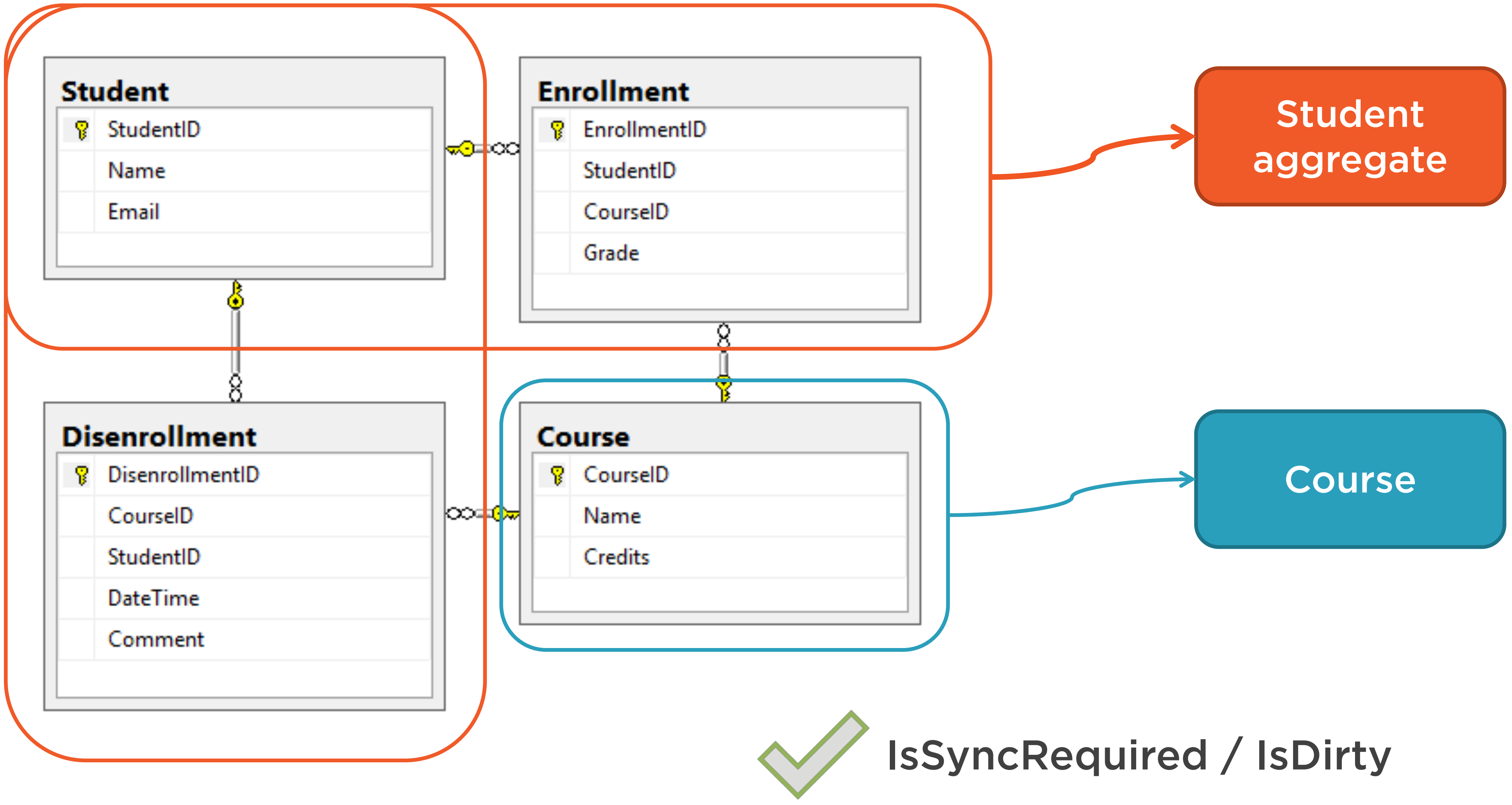
State-driven Projections

State-driven projections

Flags in data tables

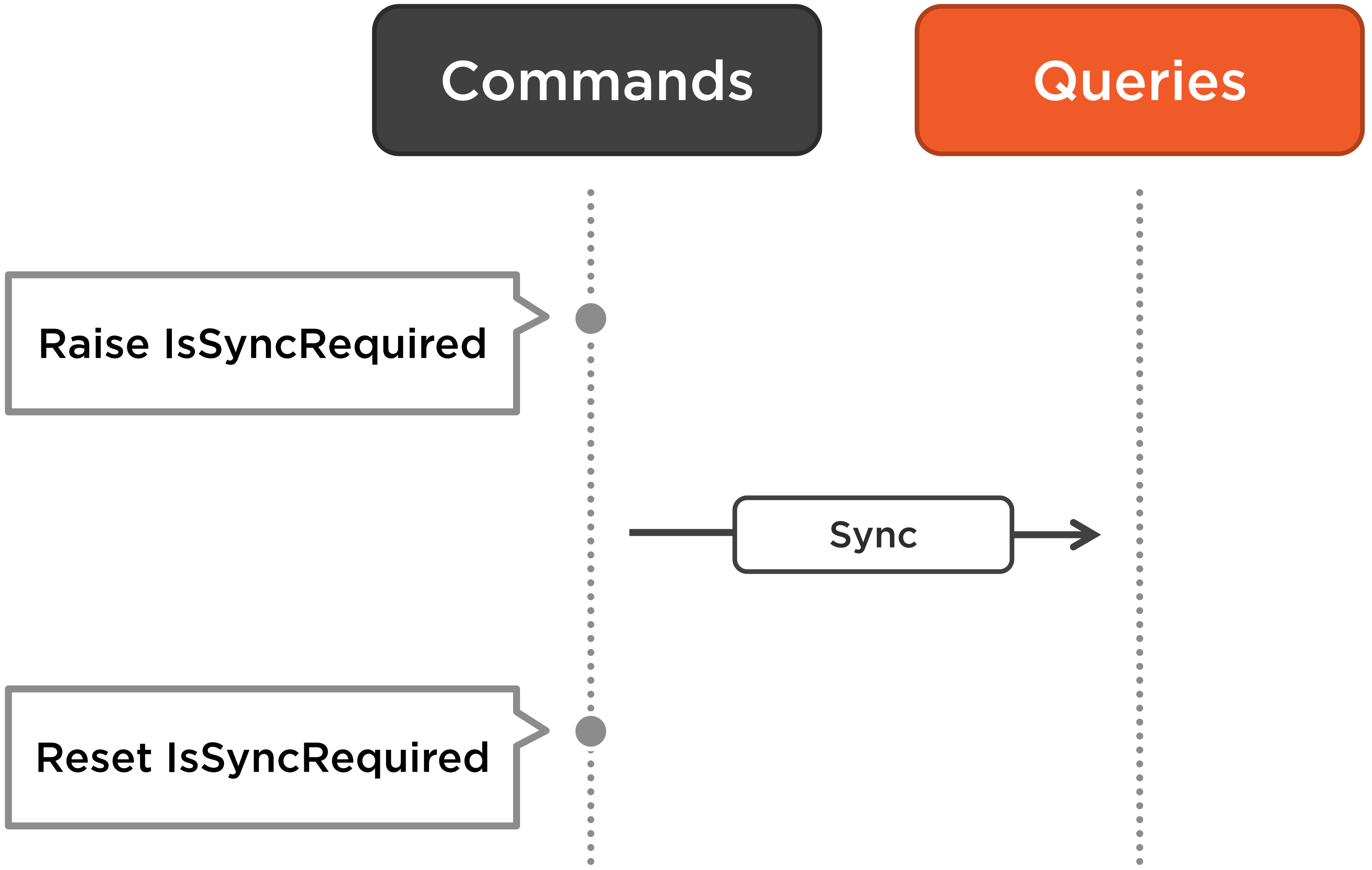
✓ **A flag per each aggregate**

State-driven Projections

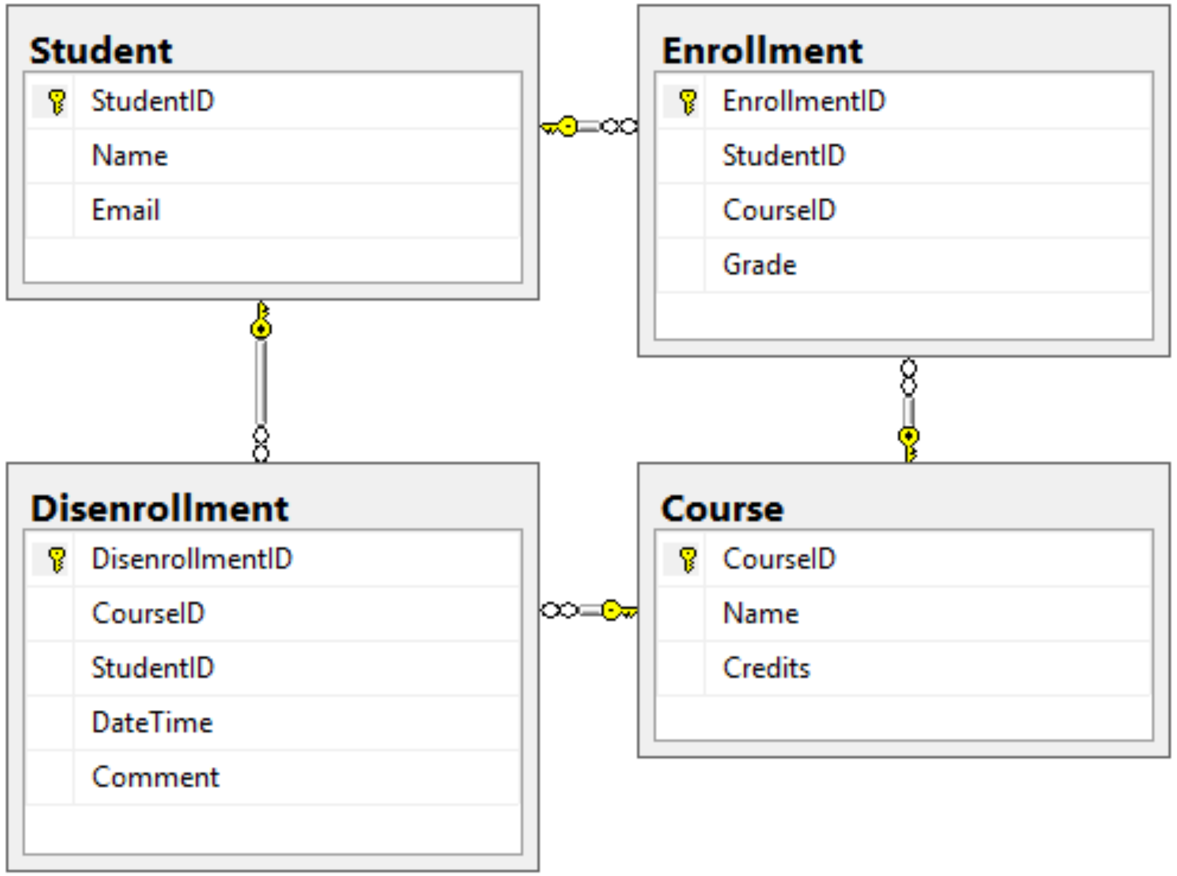


IsSyncRequired / IsDirty

State-driven Projections



State-driven Projections



Synchronization
IsSyncRequired

✓ **Offloads the pressure**

State-driven Projections

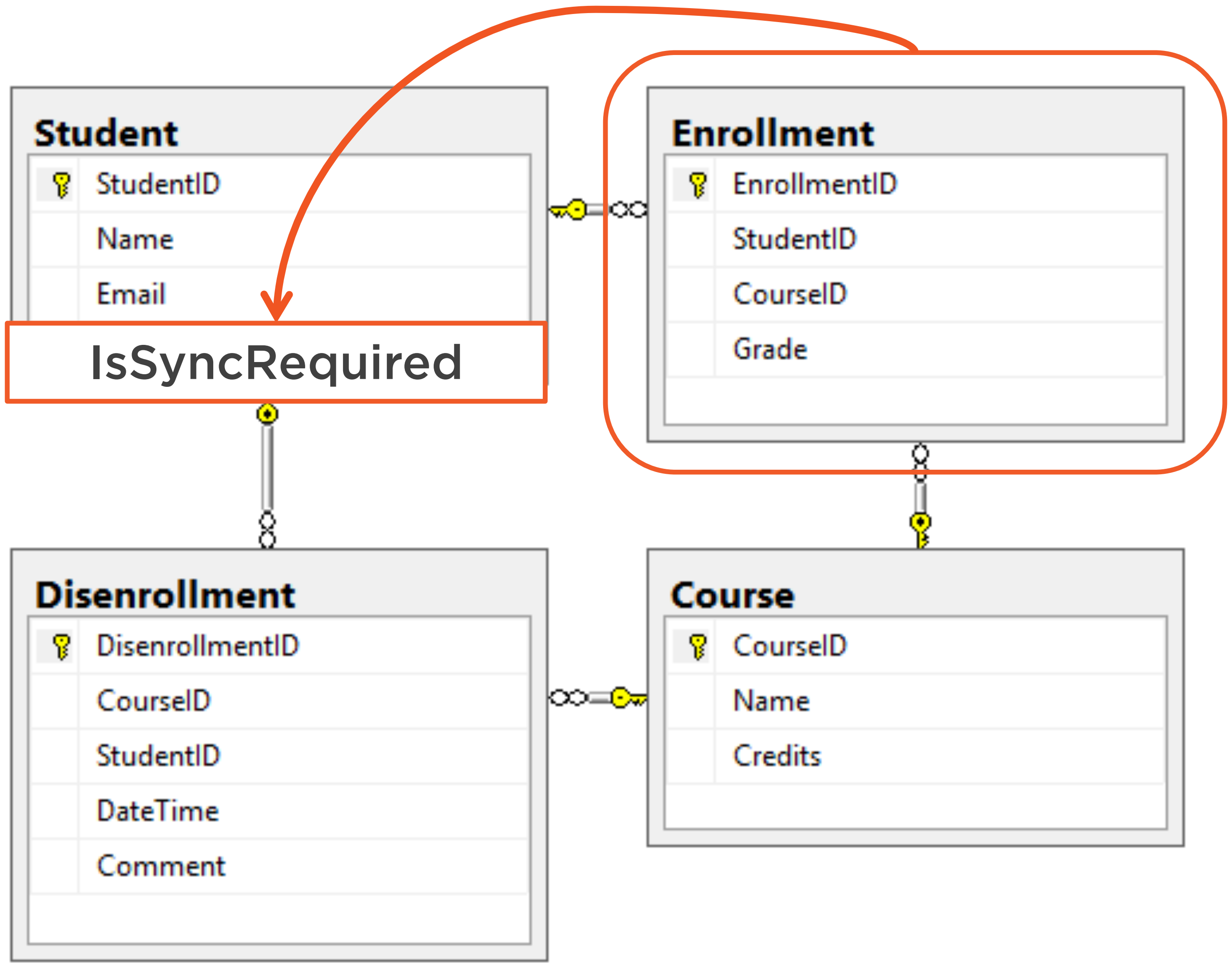
State-driven projection

Straightforward

Easy to use

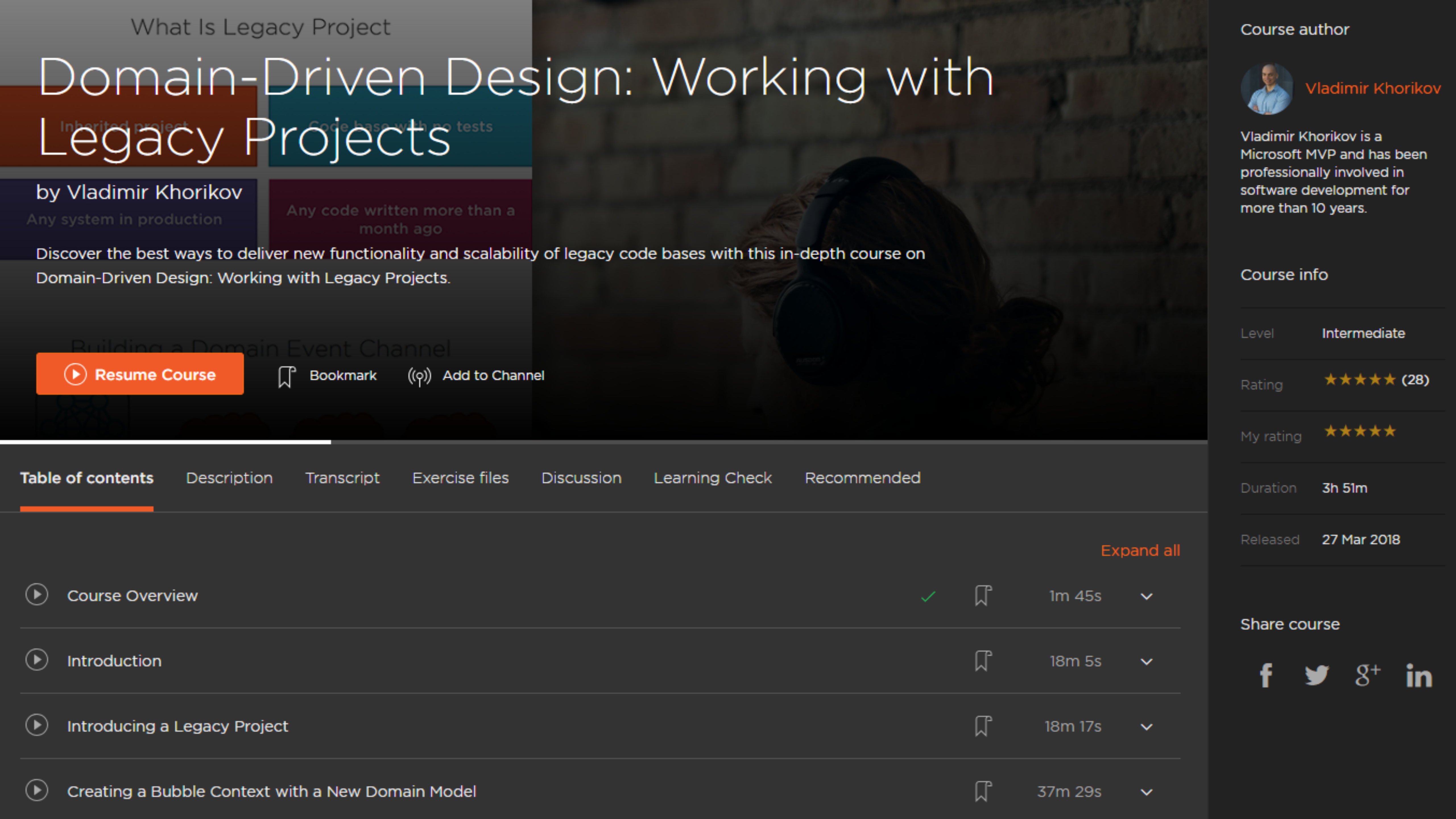
To rebuild the read database, raise the flag for all records

State-driven Projections



Database triggers

- ✓ Monitor all changes
- ✓ Need to implement a soft deletion



Domain-Driven Design: Working with Legacy Projects

by Vladimir Khorikov

Discover the best ways to deliver new functionality and scalability of legacy code bases with this in-depth course on Domain-Driven Design: Working with Legacy Projects.

Resume Course | Bookmark | Add to Channel

Table of contents | Description | Transcript | Exercise files | Discussion | Learning Check | Recommended

Expand all

Course Overview	✓	1m 45s
Introduction		18m 5s
Introducing a Legacy Project		18m 17s
Creating a Bubble Context with a New Domain Model		37m 29s

Course author

Vladimir Khorikov

Vladimir Khorikov is a Microsoft MVP and has been professionally involved in software development for more than 10 years.

Course info

Level	Intermediate
Rating	★★★★★ (28)
My rating	★★★★★
Duration	3h 51m
Released	27 Mar 2018

Share course

State-driven Projections

✓ **Introduce the flags in the domain model**

✓ **Add a flag to Student and Course**

State-driven Projections

```
public class Student : Entity {
    public virtual string Name { get; set; }
    public virtual string Email { get; set; }
    public virtual bool IsSyncRequired { get; private set; }

    public virtual void RemoveEnrollment(Enrollment enrollment, string comment) {
        _enrollments.Remove(enrollment);

        var disenrollment = new Disenrollment(enrollment.Student, enrollment.Course, comment);
        _disenrollments.Add(disenrollment);

        IsSyncRequired = true;
    }
}
```

✓ Event listeners in NHibernate

✓ Change tracker in Entity Framework

✓ http://bit.ly/ef-vs-nh

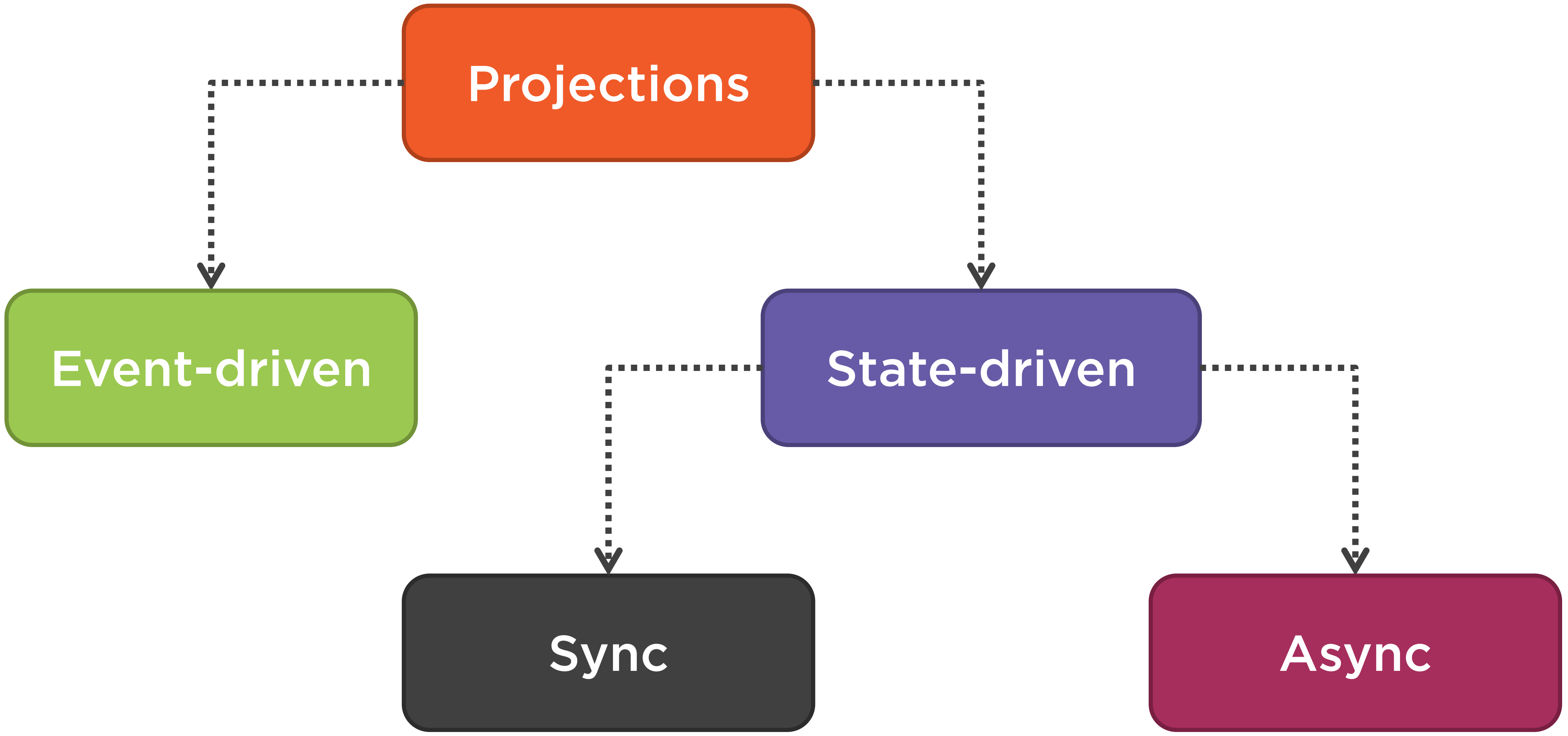
State-driven Projections

Database triggers **vs** **Explicit flags in the domain model**

✓ Choose triggers only if you don't control the source code

✓ Choose the explicit implementation by default

Synchronous State-driven Projections



Synchronous State-driven Projections

Sync

- Runs synchronously

Async

- Runs asynchronously

Asynchronous = Without blocking

✓ **Application doesn't wait for the sync job**

Synchronous State-driven Projections

Synchronous version

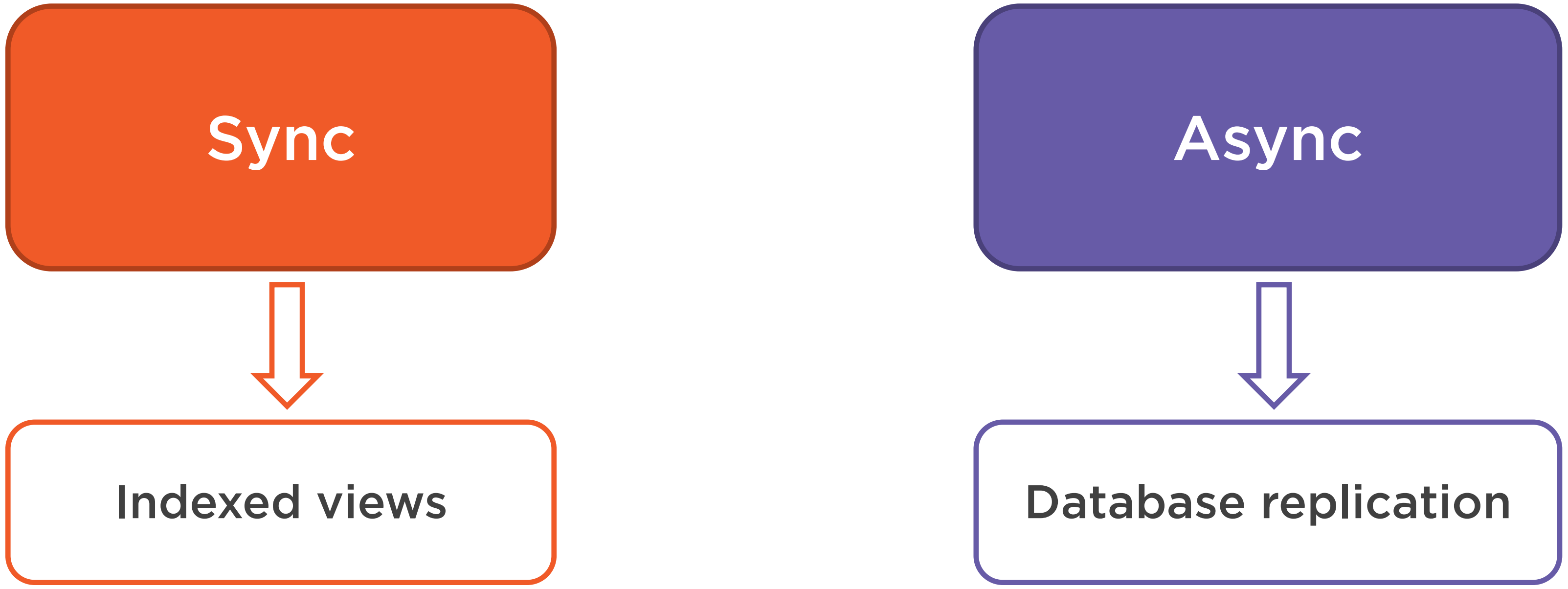
- ✓ Application does the projection
- ✗ Increases the processing time
- ✓ All changes are immediately consistent

Synchronous State-driven Projections

Synchronous projections don't scale

Synchronous State-driven Projections

Synchronous projections



Event-driven Projections

Event-driven projections

Domain events drive the changes

Subscribe to domain events

~~Synchronization~~

~~IsSyncRequired~~

Event-driven Projections

Event-driven projections

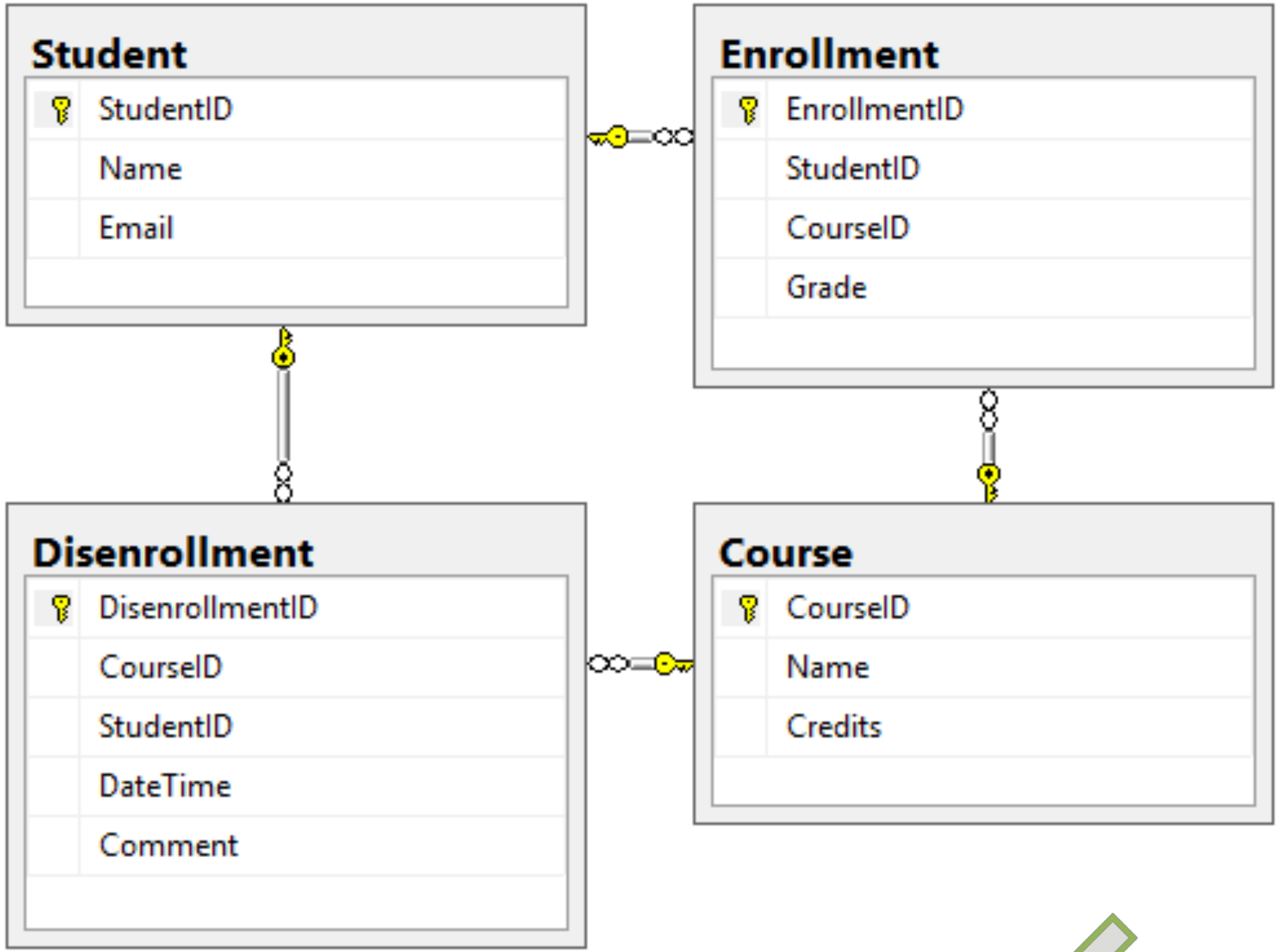
Domain events drive the changes

✓ Scales really well

✓ Can use a message bus

✗ Cannot rebuild the read database

Event-driven Projections



State

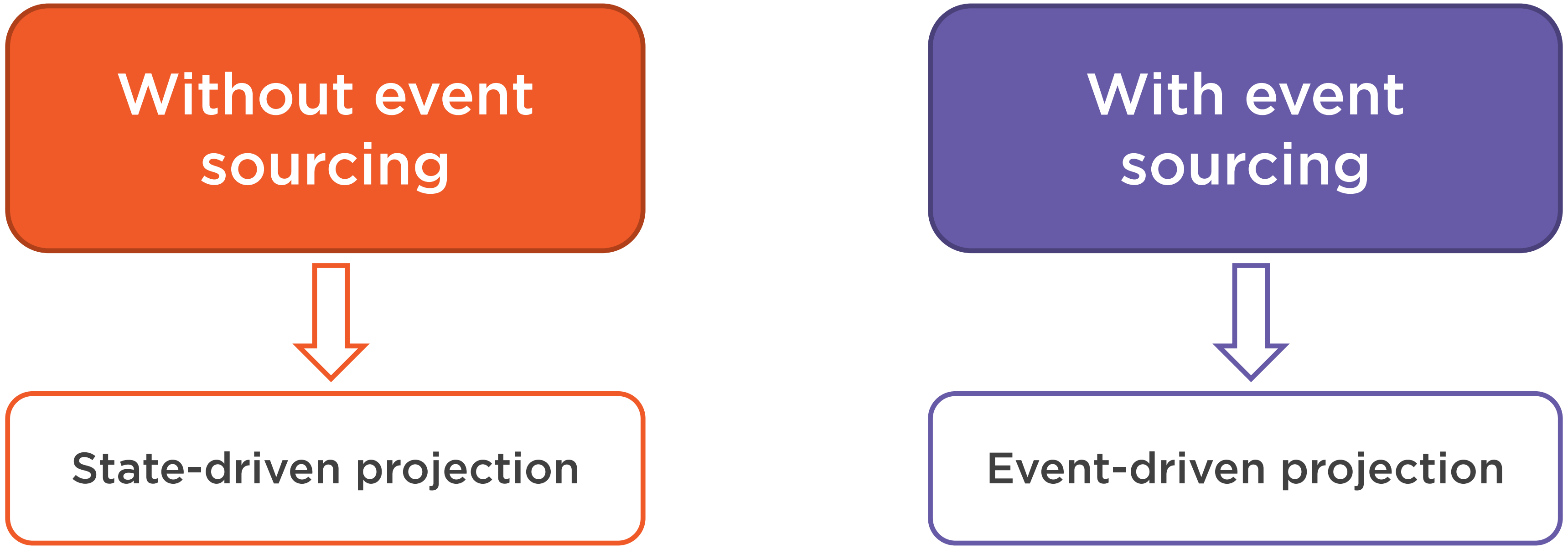
✗ Don't store domain events

✗ Impossible to derive events from state

✓ Transition to event sourcing

Event-driven Projections

How should you choose the projection type?



Align the projection strategy with the persistence mechanism

Consistency

Having two databases instead of one introduces latency

✗ May end up with duplicate records

✓ You will still gain a lot of benefits even with a single database

Consistency

Ways to mitigate the potential confusion

✓ Uniqueness constraints

✓ Commands database is always immediately consistent

Consistency

How should you query the database during a command execution?

~~Run a query from a command handler?~~

Queries database might not be up to date with the commands database

Consistency

✓ **Query the commands database**

Consistency

Reading read the commands database

vs.

Reading read the queries database

⇩

Part of the command processing flow

Results don't cross the application boundaries

Consistency

```
public sealed class StudentRepository
{
    public Student GetById(long id)
    {
        return _unitOfWork.Get<Student>(id);
    }
}

public sealed class CourseRepository
{
    public Course GetByName(string name)
    {
        return _unitOfWork.Query<Course>()
            .SingleOrDefault(x => x.Name == name);
    }
}
```

✓ **Serve the commands, not queries**

Consistency

You are not able to efficiently query the current state with Event Sourcing

✓ Have to query the read database

Eventual Consistency

Train users not to expect data to be immediately consistent

Wouldn't the software become less usable without immediate consistency?

The concept of immediate consistency is counterintuitive.

Eventual Consistency

Driver's license

- Are changes in the real world immediately consistent?
- ✓ The real world is inherently asynchronous and eventually consistent
- ✓ Users quickly learn the concept of eventual consistency

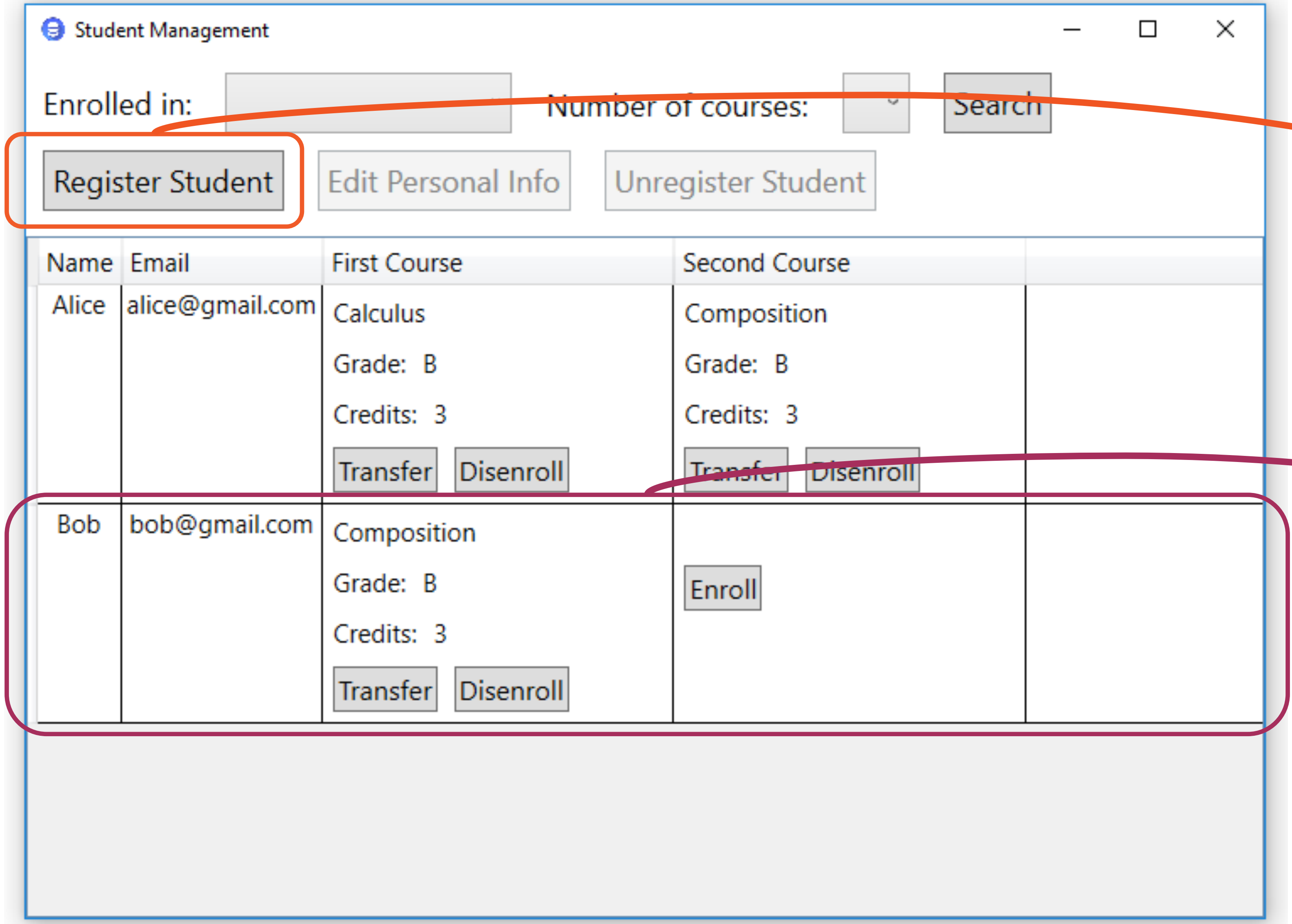
Eventual Consistency

A consistency model which guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

Eventual Consistency

Display helpful messages and set proper expectations

Eventual Consistency



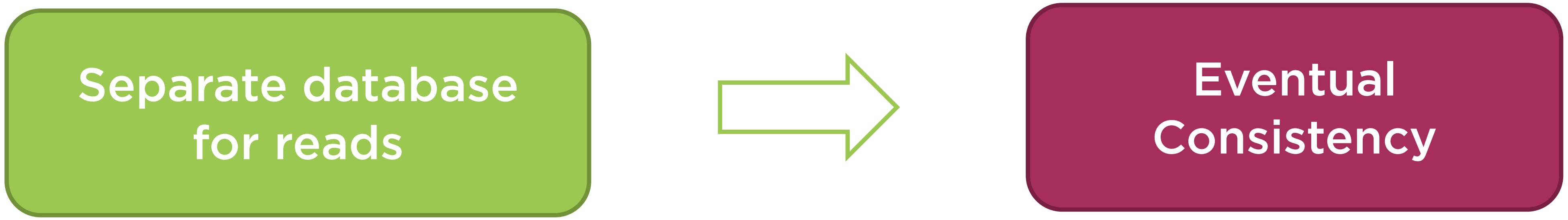
Name	Email	First Course	Second Course
Alice	alice@gmail.com	Calculus Grade: B Credits: 3 Transfer Disenroll	Composition Grade: B Credits: 3 Transfer Disenroll
Bob	bob@gmail.com	Composition Grade: B Credits: 3 Transfer Disenroll	Enroll

"Student registration is submitted"

Display the new record locally

✓ Two-way communication

Eventual Consistency



Starbucks doesn't use two-phase commit

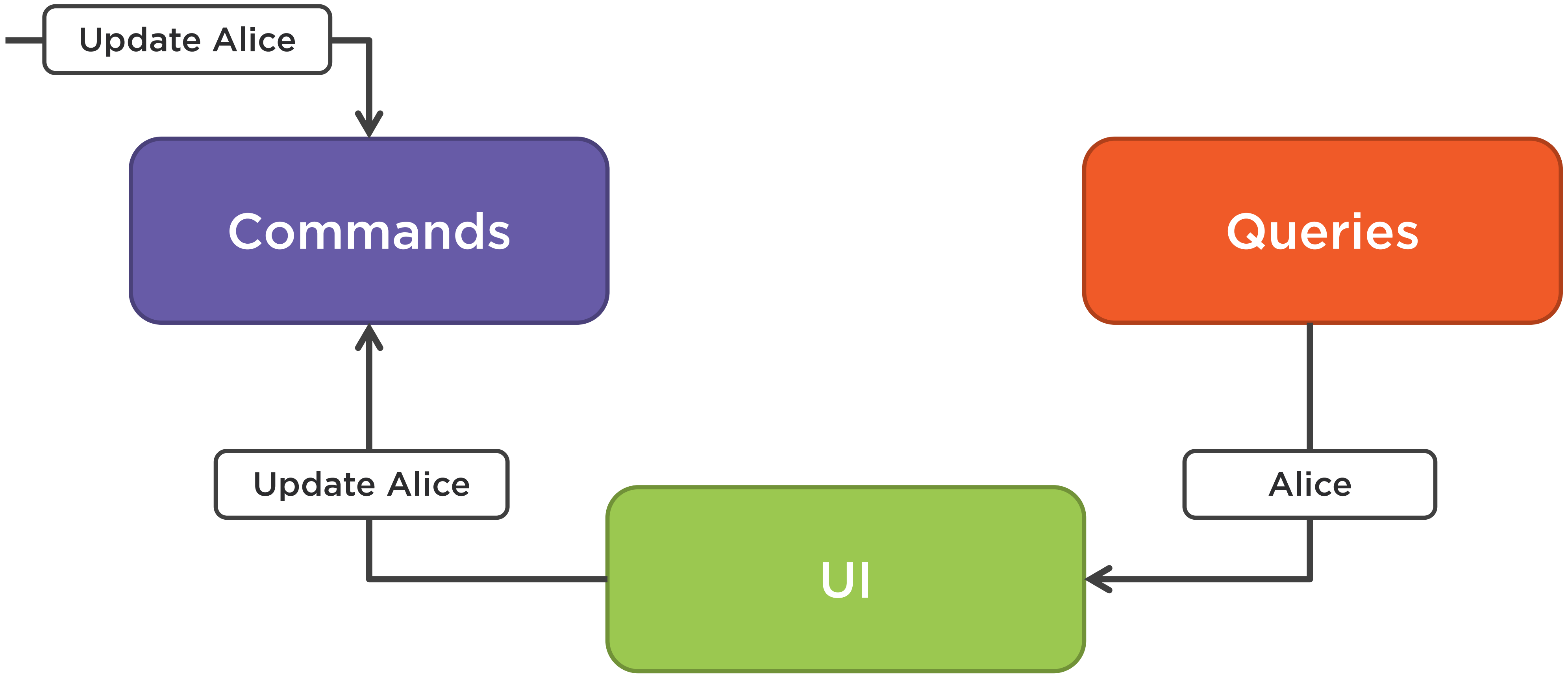
http://bit.ly/starbucks-cons

Eventual consistency is problematic when the cost of making a decision based on the stale data is high.

Eventual Consistency

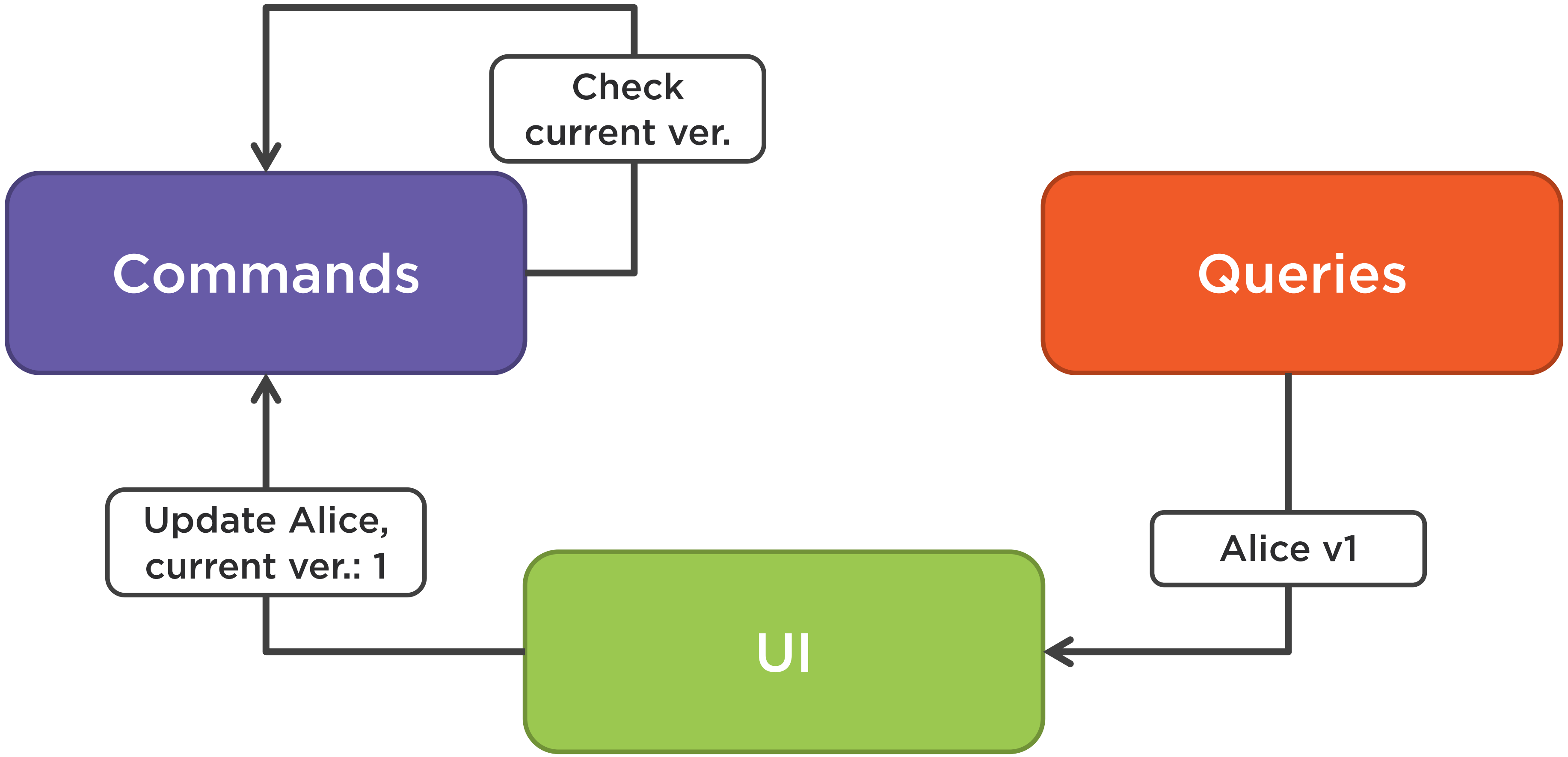
✓ **Introduce versioning**

Eventual Consistency



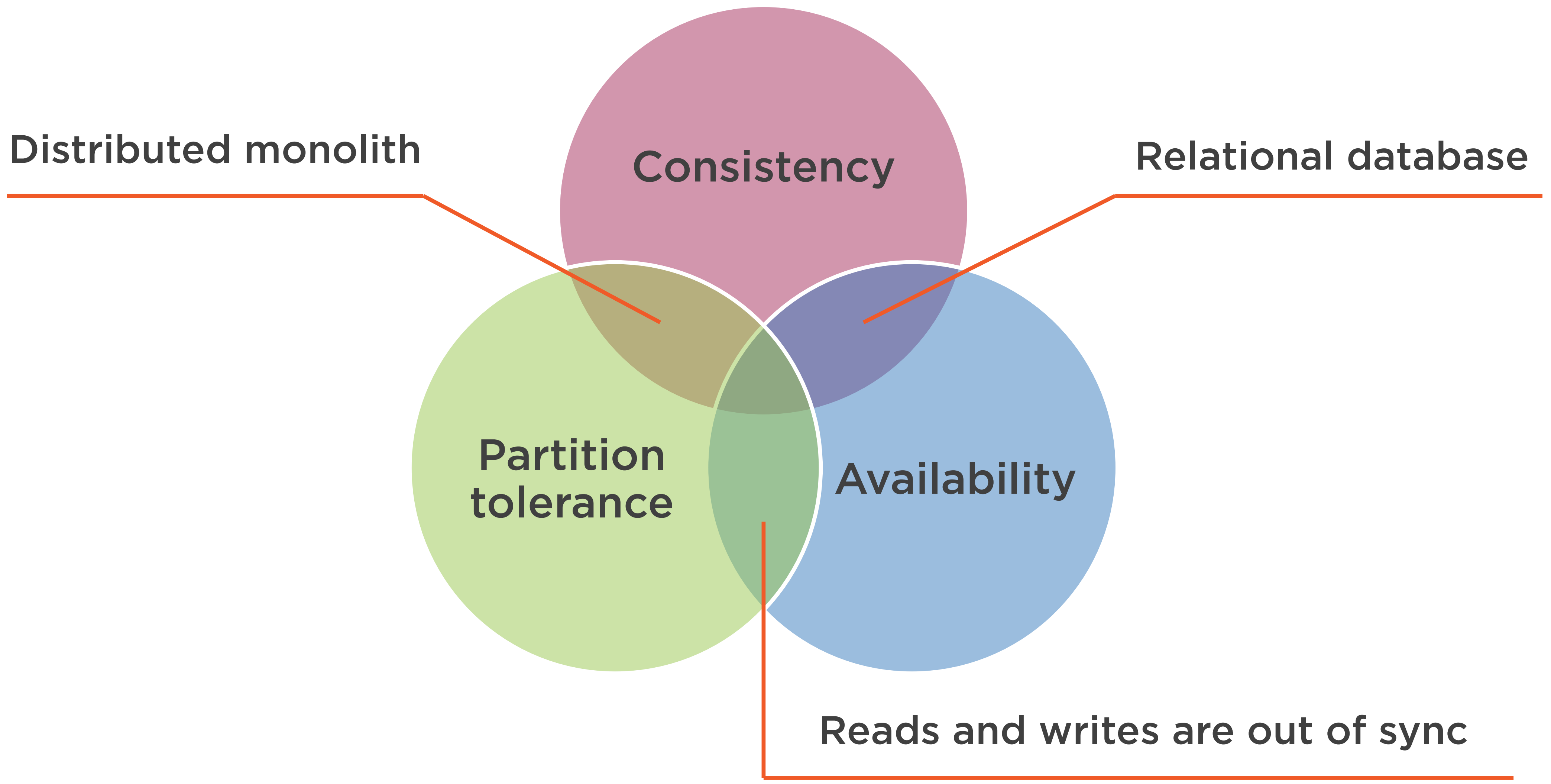
✓ Make the version number part of all the communications

Eventual Consistency



✓ **Make the version number part of all the communications**

CQRS and the CAP Theorem

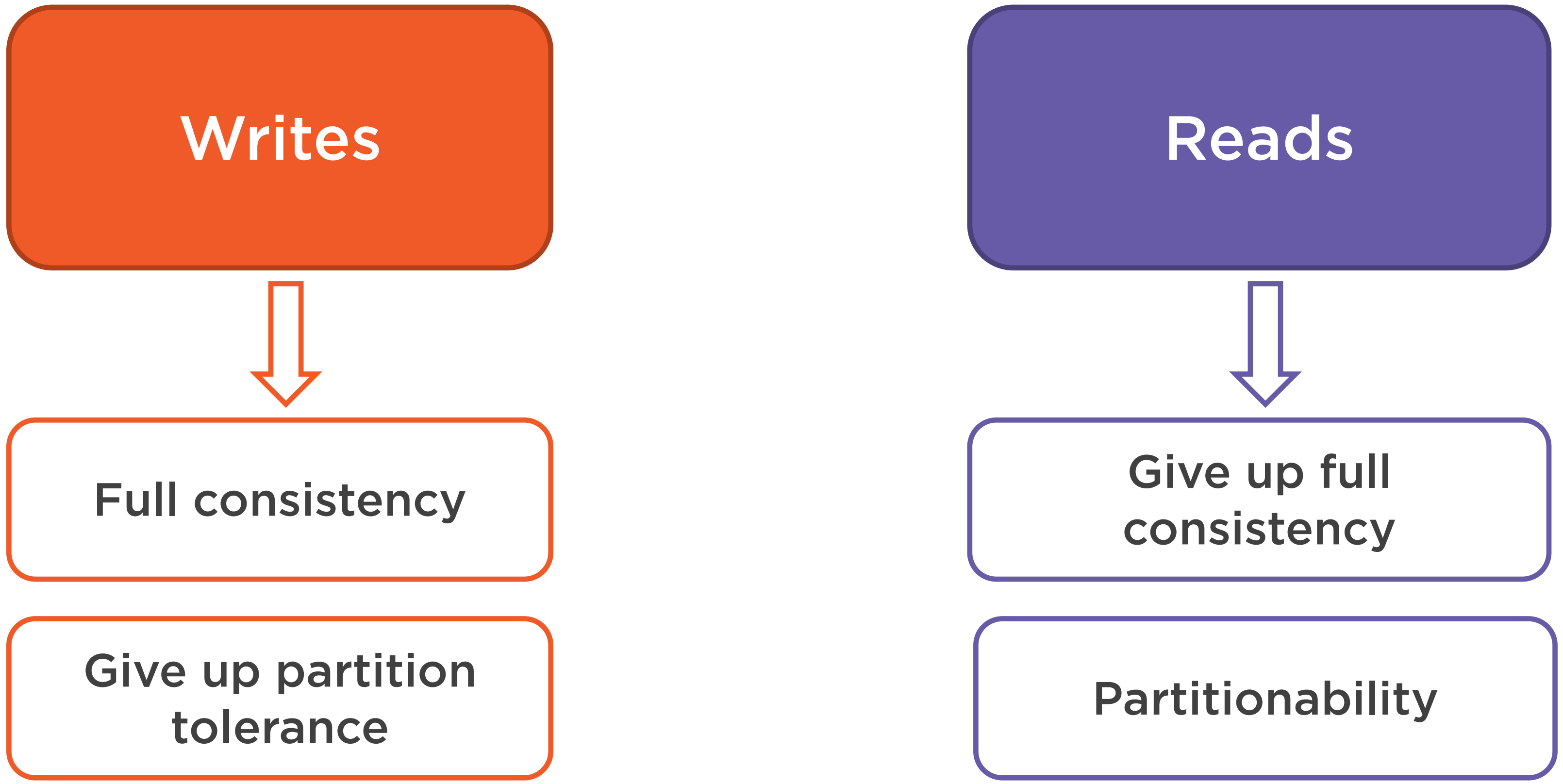


CQRS and the CAP Theorem

Finding a proper balance is hard

CQRS allows you to make different choices for reads and writes

CQRS and the CAP Theorem



- ✓ Consistent changes are important
- ✓ Scalability is important

Summary

Synchronization between commands and queries

State-driven projection

- Introducing an IsSyncRequired flag in aggregates
- Database triggers or explicit in the model update
- Choose the explicit route by default
- Synchronous and asynchronous

Event-driven projection

- Using domain events to build the queries database

Without Event Sourcing: use state-driven projections

With Event Sourcing: use event-driven projections

Summary

Immediate vs. eventual consistency

- Immediate consistency is contrary to the real world experience
- People pick up eventual consistency quickly
- Implement data versioning and the optimistic concurrency control

CAP theorem

- CQRS is about making different choices with regards to the balance within CAP
- Choose consistency and availability at the expense of partitioning for writes
- Choose availability and partitioning at the expense of consistency for reads

In the Next Module

CQRS Best Practices and Misconceptions