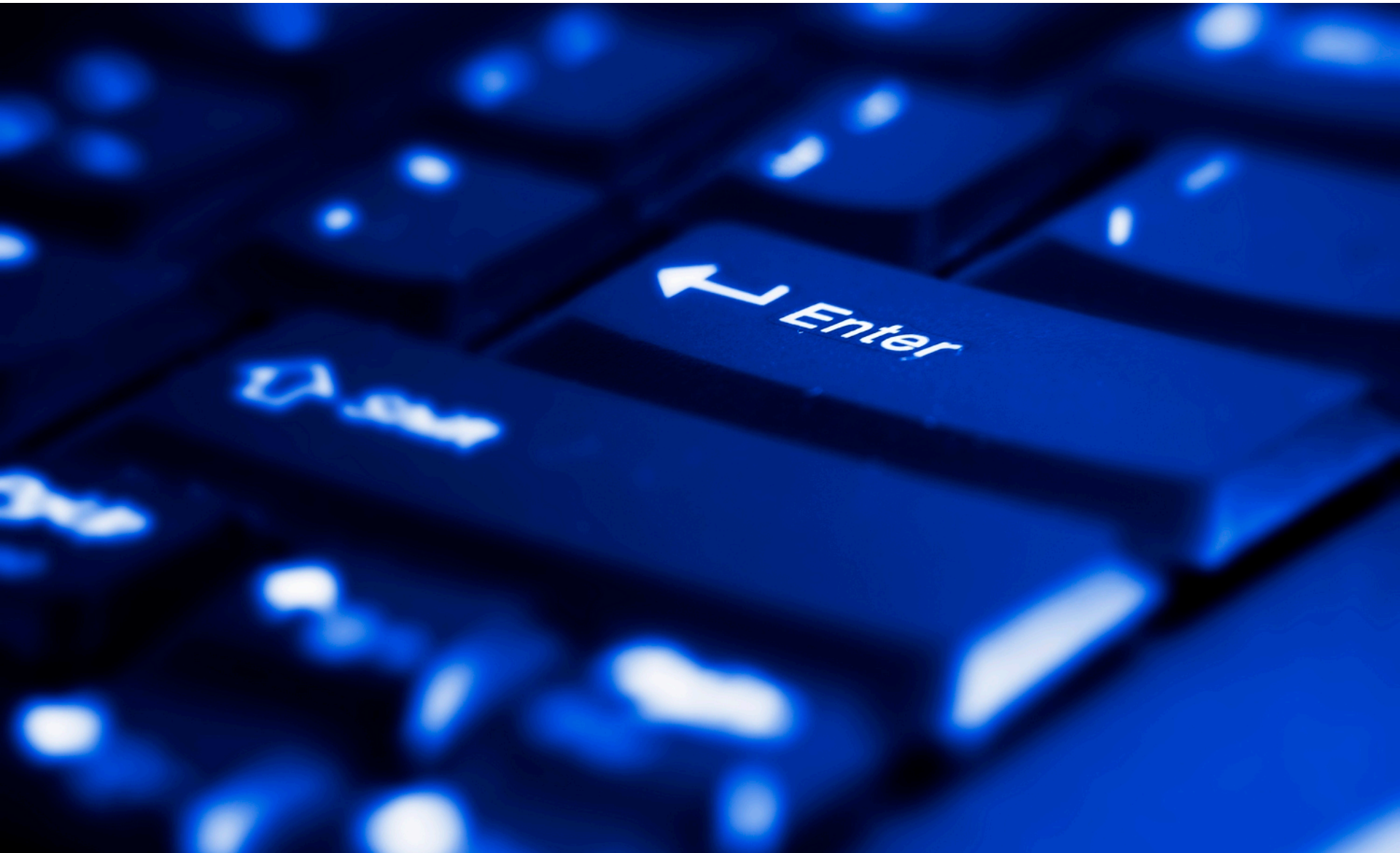
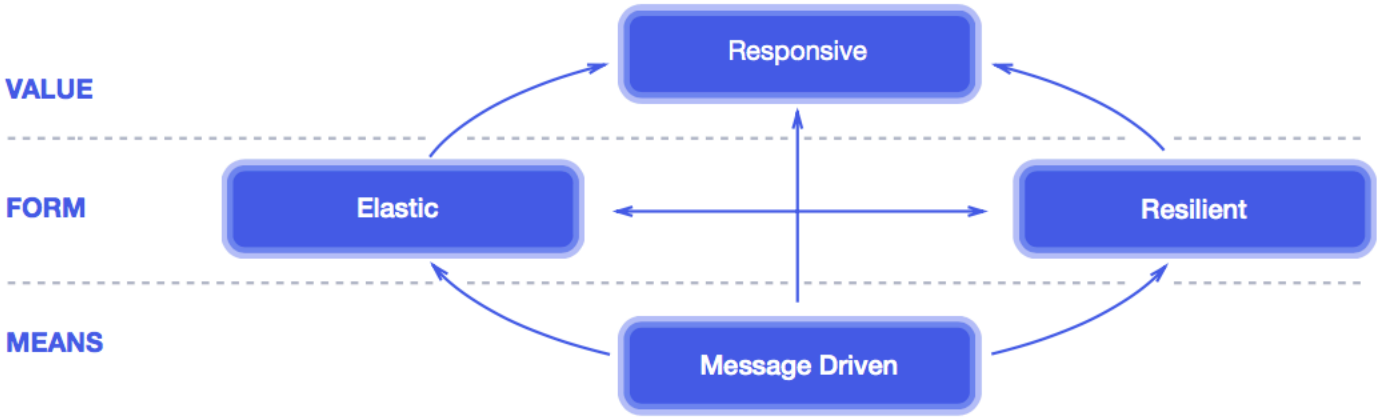
bns it

Reactive architecture

Modern Application Architectures




VALUE
FORM
MEANS
Responsive
Elastic
Resilient
Message Driven

Reactive architecture

A reactive system (architecture)

- architectural style
- allows multiple individual applications to work as a single unit
- reacting to its surroundings
- able to scale up/down
- load balancing

Reactive programming

- Subset of asynchronous programming

- Method invocations can be executed in an asynchronous and non-blocking fashion

Reactive programming

- Reactive programming is generally **low level** *event-driven*

- (API) for reactive programming:
  - Callback-based – invoking callbacks when events pass through the dataflow chain
  - Declarative - using well-established combinators like *map*, *filter*

Events

- **Low-level events** – supported at language level or through design patterns (Observer), used locally inside bounded contexts

- **Domain events** – fat events, part of the domain laguage, passed between bounded contexts

Messages vs Events

- Messages are directed, events are not
- Messages have single destination, events are facts for others to observe
- Messages are needed to communicate across the network and form the basis for communication in distributed systems
- It is common to use messaging for sending events inside messages

Reactive architecture

- Reasons about a system at a higher level
- Message-driven (aka *messaging*)

Reactive architecture

- In a reactive system, especially with reactive programming, both events and messages will be present

- **Message-passing** between allows components to be decoupled in
  - time (concurrency)
  - space (distribution)

Reactive architecture

- Decoupling is a requirement for full isolation
  - base for *resilience* and *elasticity*
- Resilience is about responsiveness *under failure*
- Elasticity is about *responsiveness under load*

Reactive architecture

… but less control

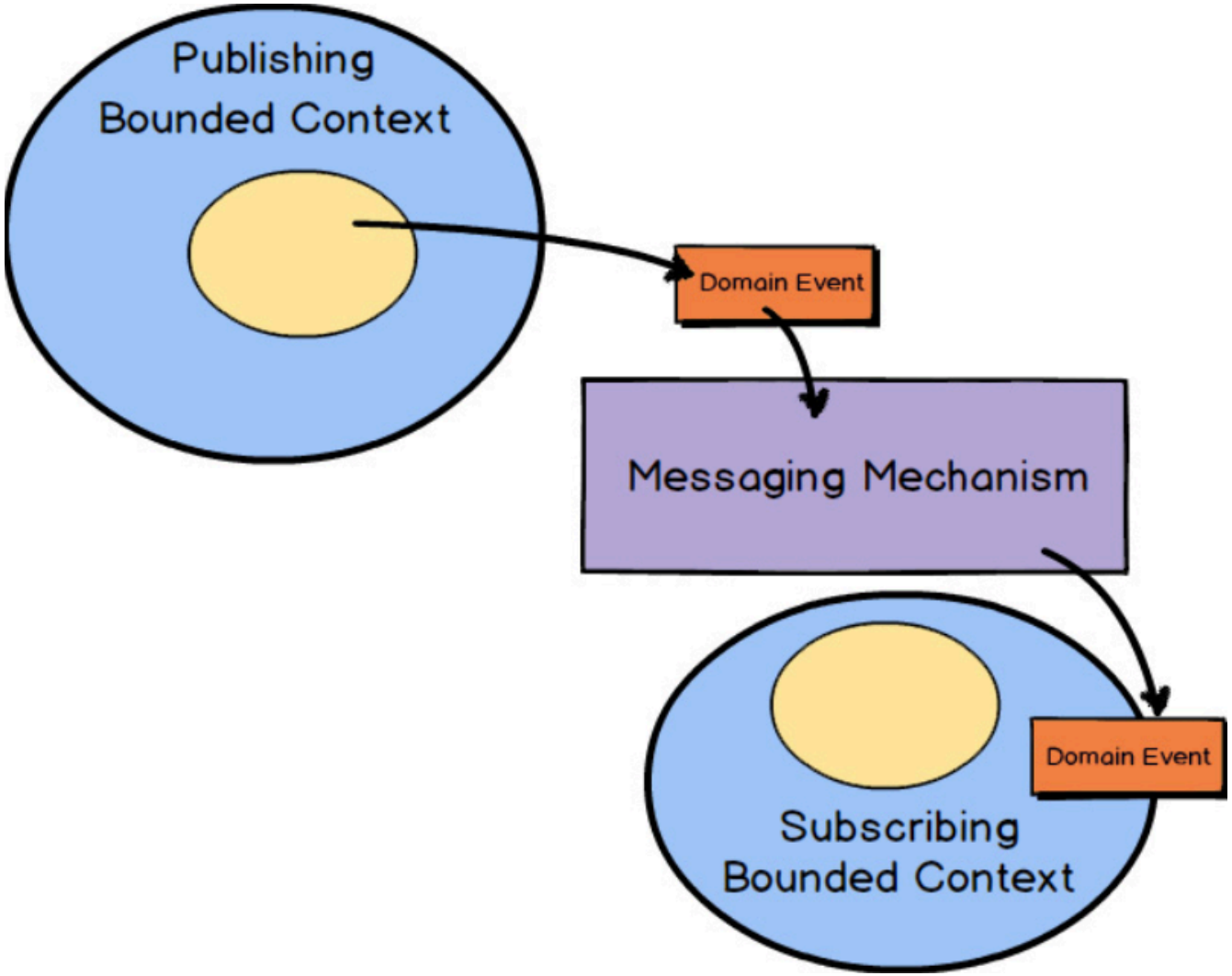
Have to deal with

- partial failures,
- failure detection,
- eventual consistency and so on…

bns it

Events and messaging

Domain events



Designing, implementing and using Domain Events

Designing, implementing and using Domain Events

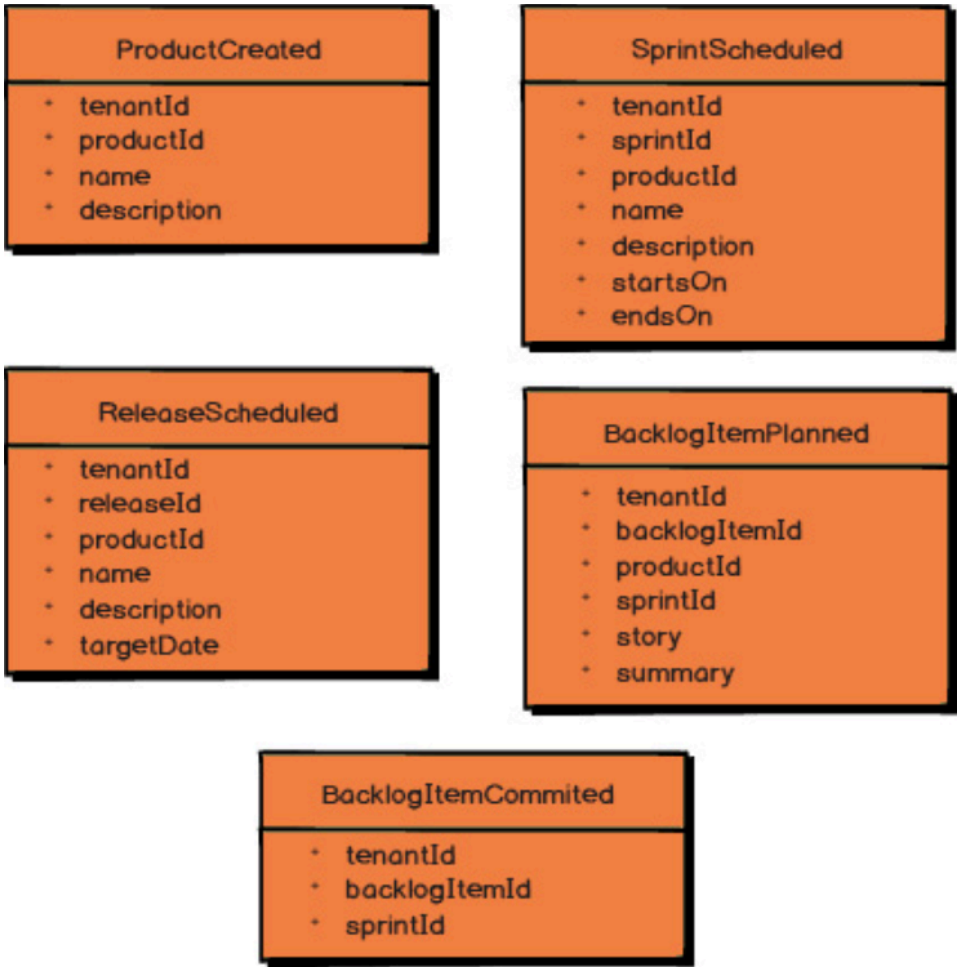
CreateProduct

- tenantId
- productId
- name
- description

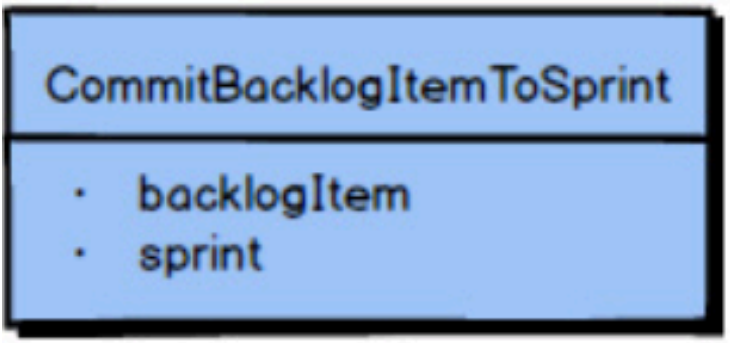
ProductCreated

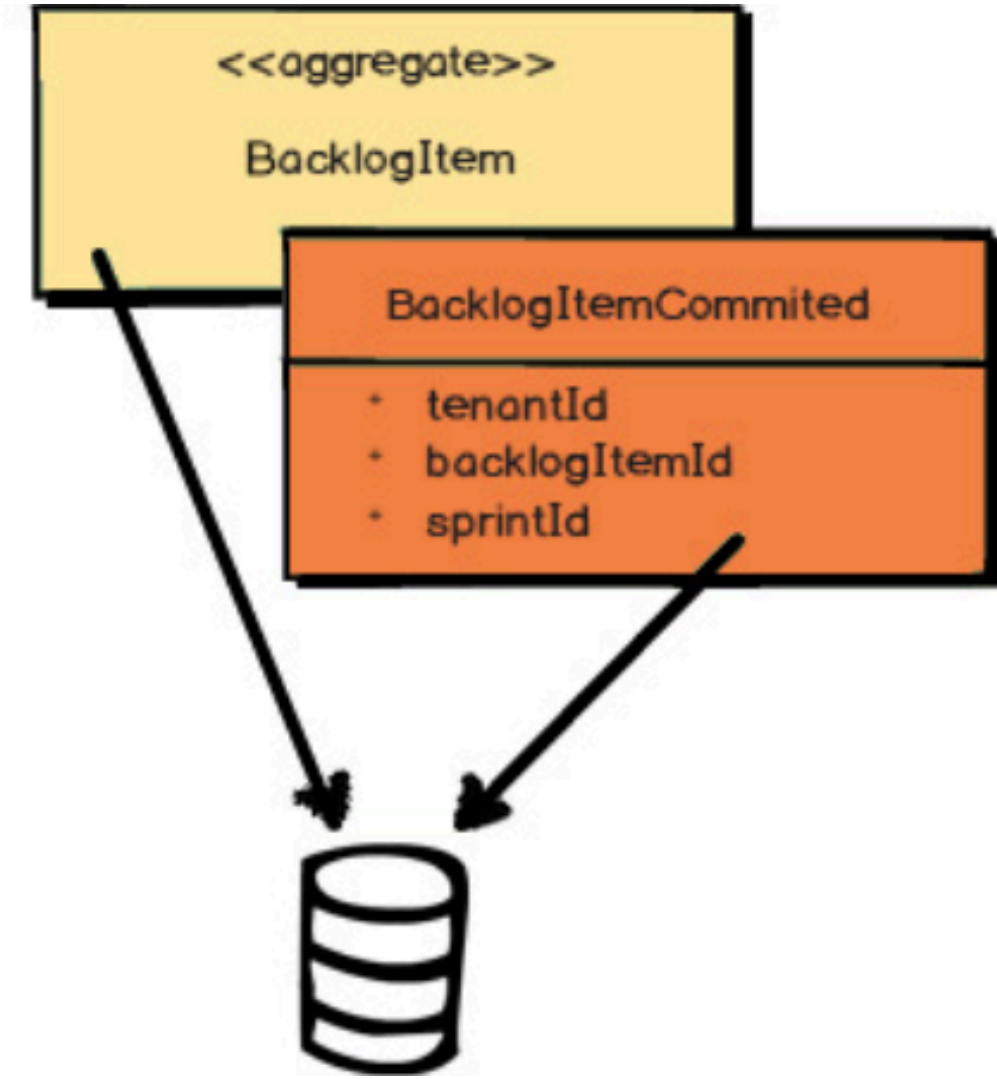
- tenantId
- productId
- name
- description

Designing, implementing and using Domain Events

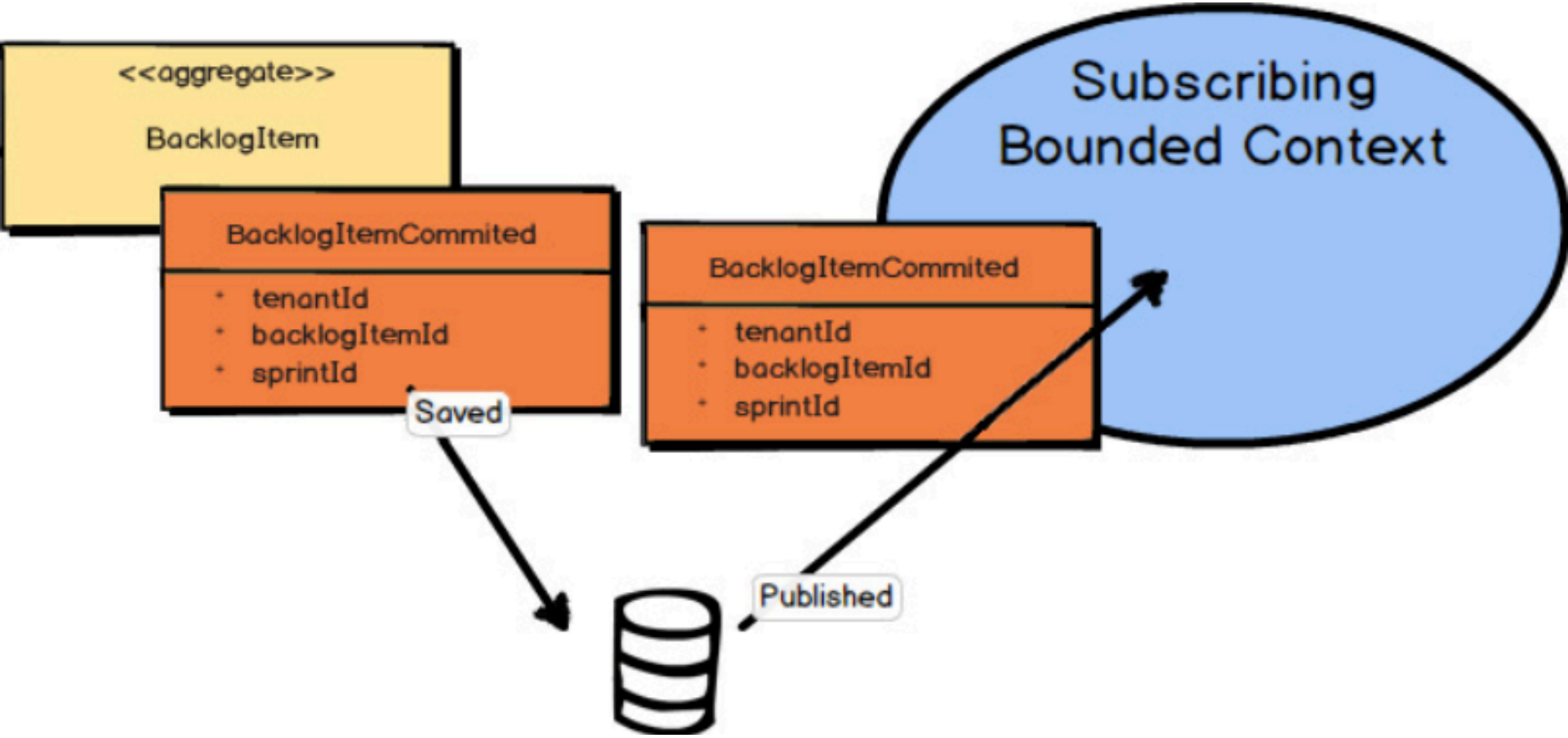


Designing, implementing and using Domain Events





Designing, implementing and using Domain Events



```java
public void reopen() {
    if (!this.isClosed()) {
        throw new IllegalStateException("The discussion is not closed.");
    }

    this.apply(new DiscussionReopened(this.tenant(), this.forumId(),
                this.discussionId(), this.exclusiveOwner()));
}
```

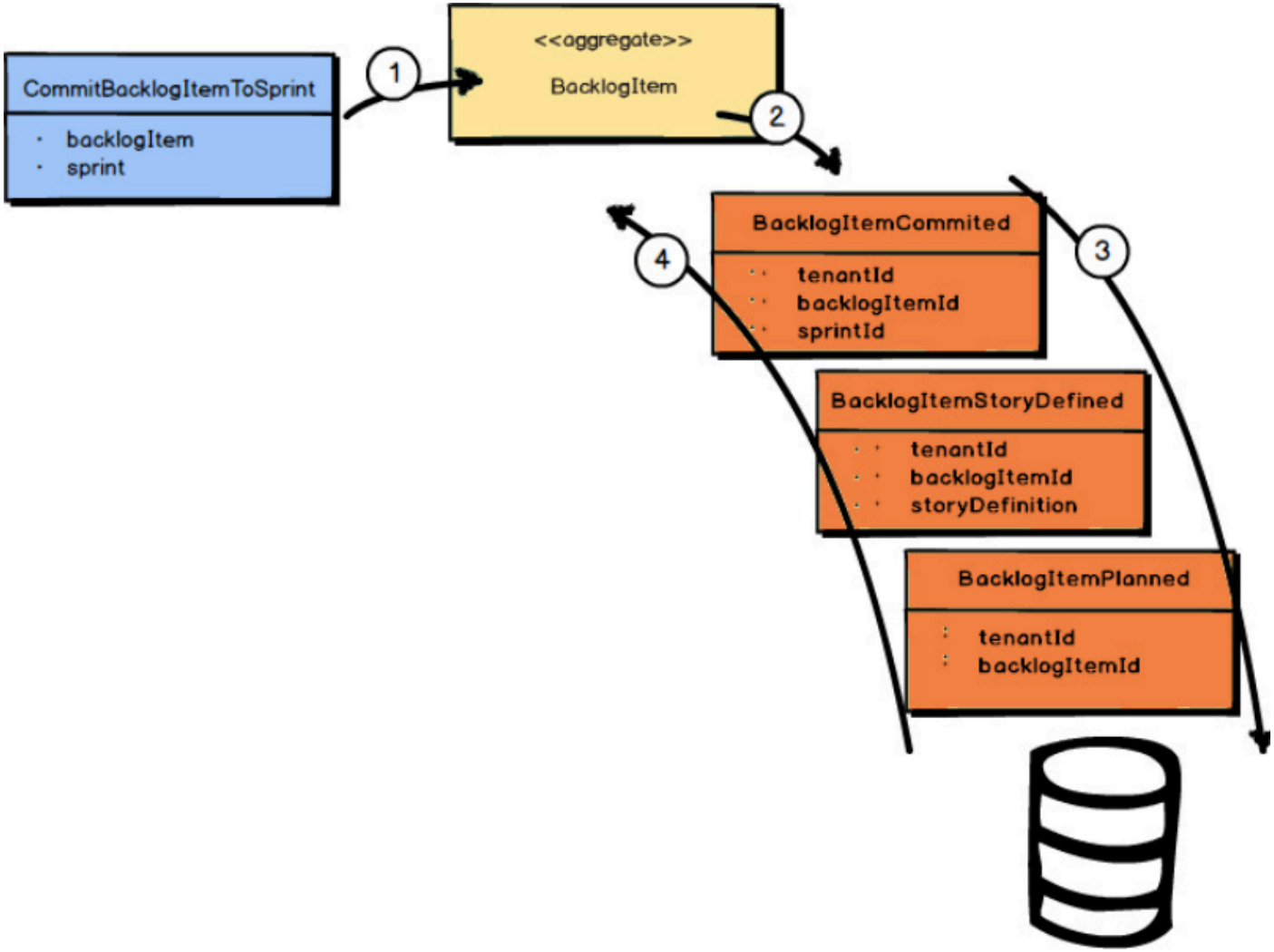
```java
public void close() {
    if (this.isClosed()) {
        throw new IllegalStateException("This discussion is already cl
    }

    this.apply(new DiscussionClosed(this.tenant(), this.forumId(),
                this.discussionId(), this.exclusiveOwner()));
}
```

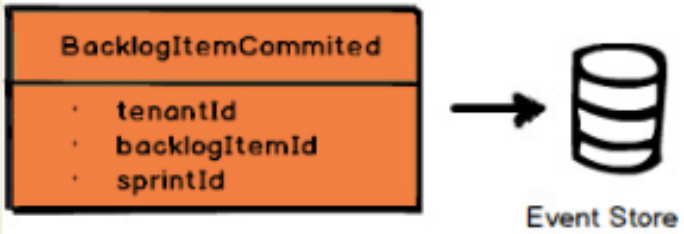
bns it

Event sourcing

Designing, implementing and using Domain Events



Designing, implementing and using Domain Events



Stream Id	Stream Version	Event Type	Event Content
backlogItem123	1	BacklogItemPlanned	{ ... }
backlogItem123	2	BacklogItemStoryDefined	{ ... }
backlogItem123	3	BacklogItemCommitted	{ ... }
...	N	...	{ ... }
...	N	...	{ ... }
...	N	...	{ ... }