Just don’t do it!

Anti-Patterns in event-getriebenen Architekturen

Florian Pfleiderer

Frank Steimle

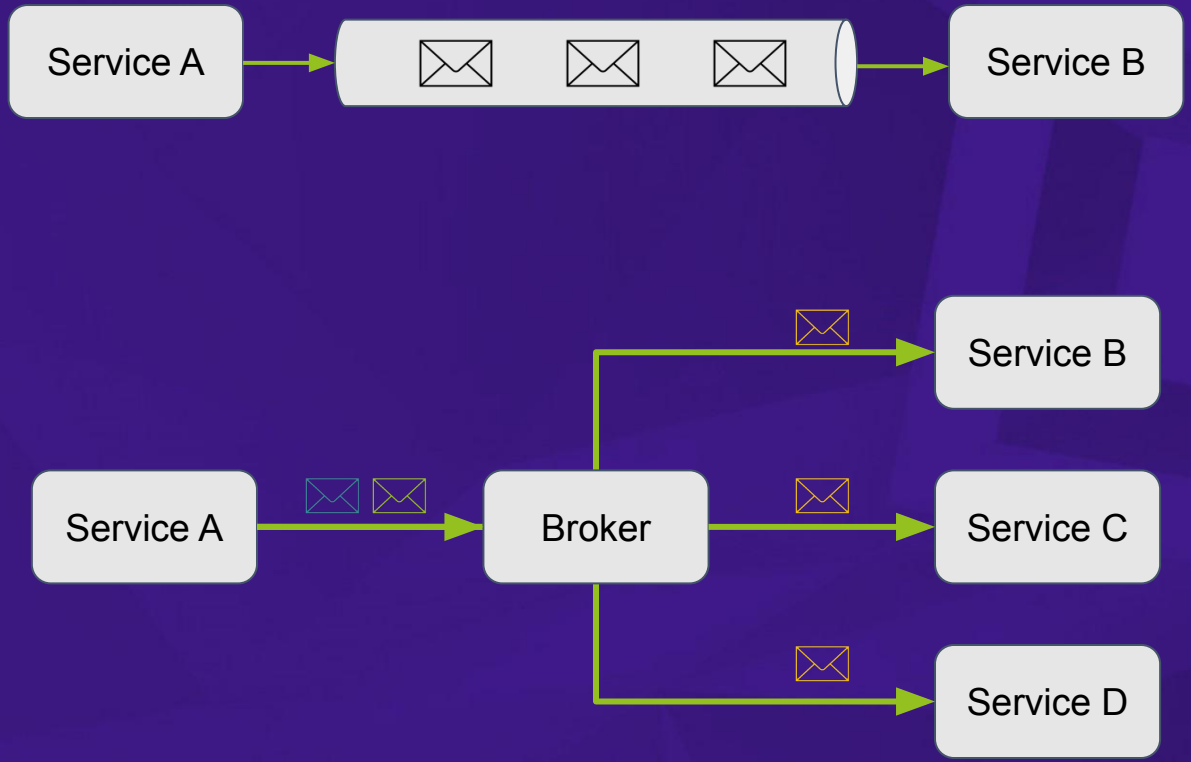
digital frontiers

A Synchronous Example

Details:

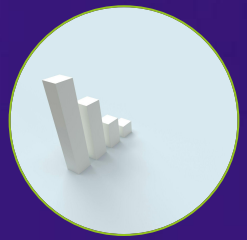
- Characteristics: Fails fast, simple, low latency
- Challenge: How to handle failures?
- Challenge: What happens if Service A does more requests than Service B can handle?
- Challenge: If services belong to different teams: who is responsible?

EDA Communication Patterns: Point-To-Point Connection & Pub/Sub



Characteristics of Event-Driven Architecture

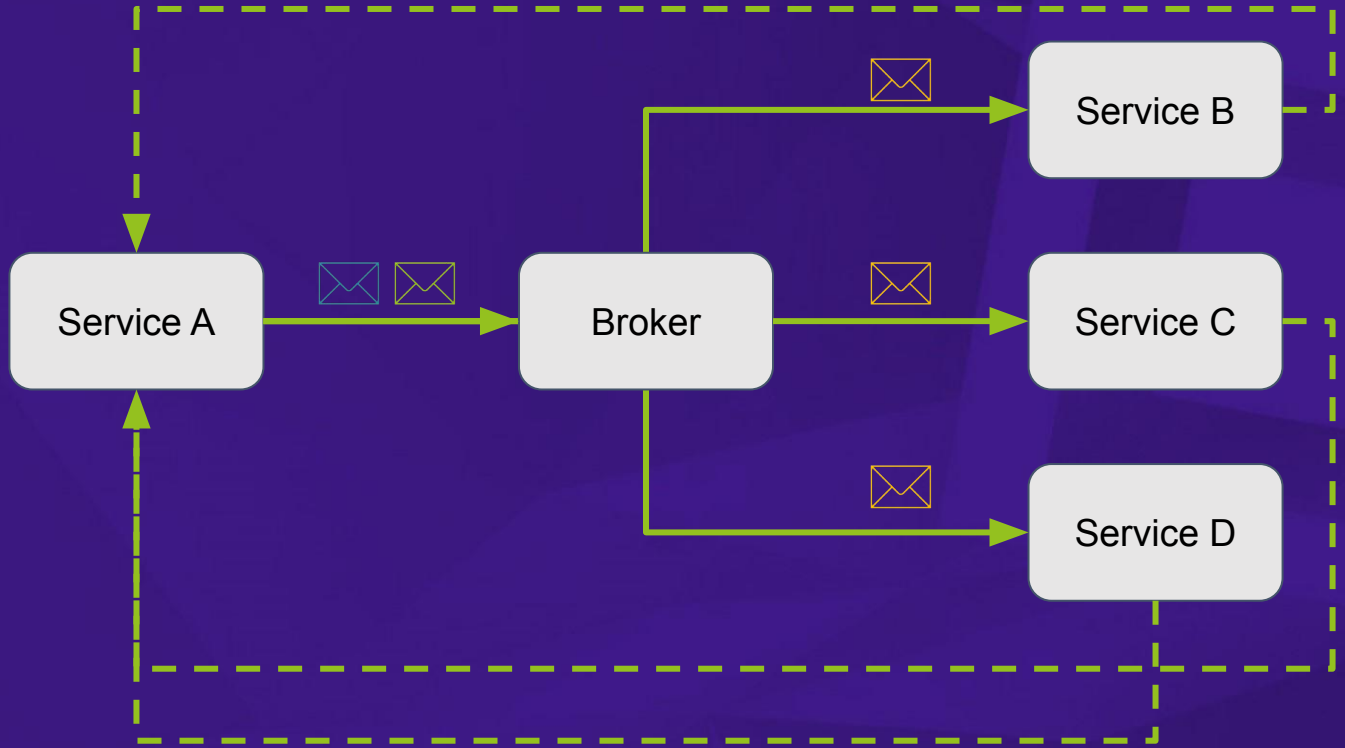
Flexibility &
Scalability



Loose Coupling &
Resiliency

Eventual Consistency &
Developer's Mind Set

Anti-Pattern: Enforce Callbacks



Common Event Types

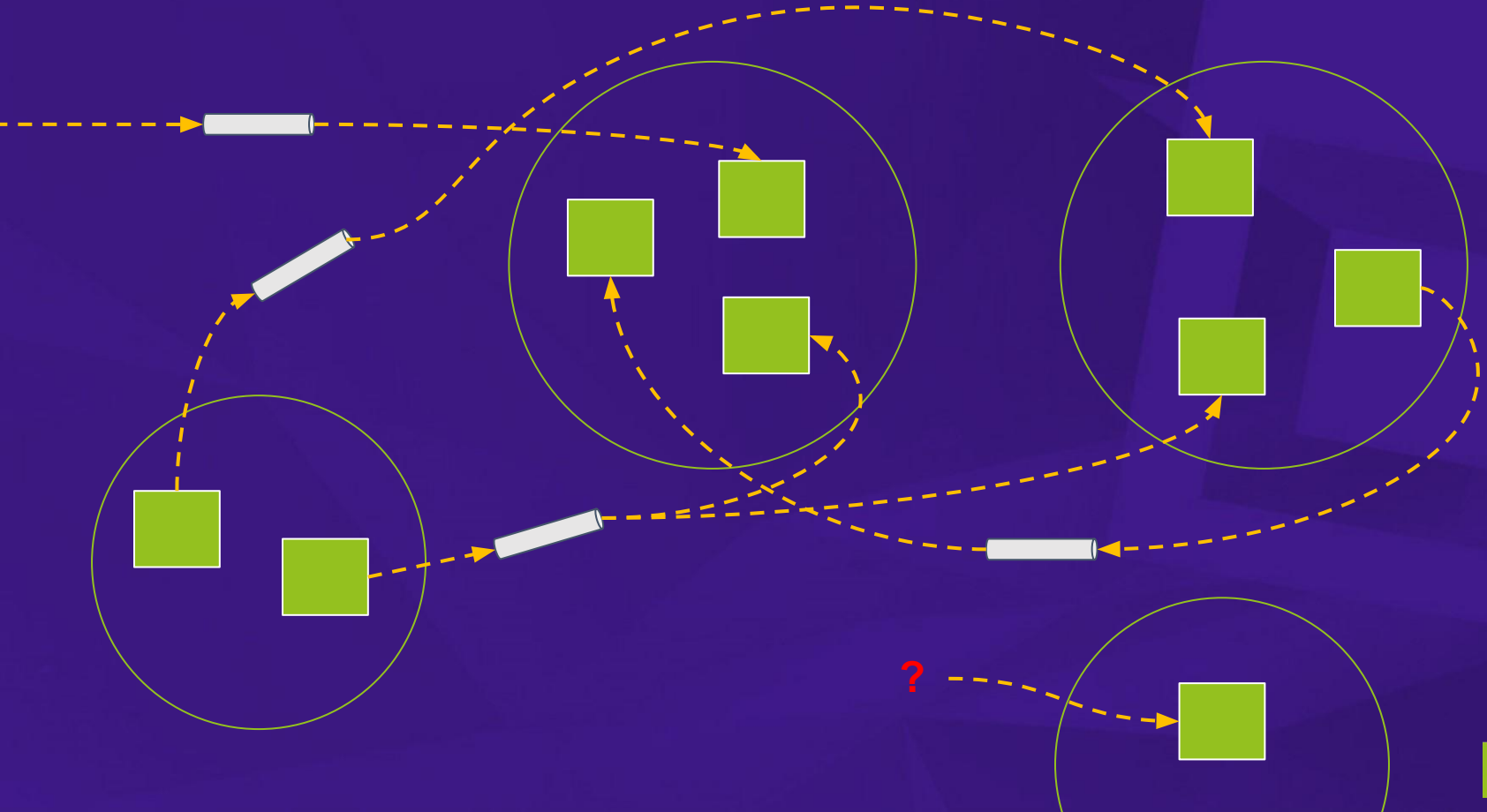
Notification Events

Event Carried State Transfer

Event Sourcing Events

Delta Events

Anti-Pattern: Big Ball Of Mud



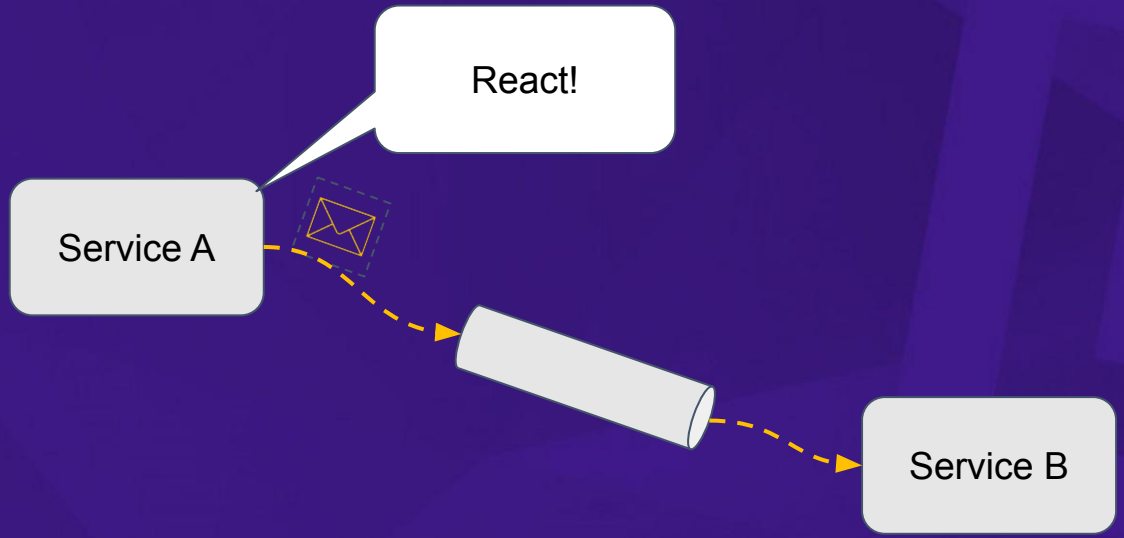
Governance in Event-Driven Architectures

Standards

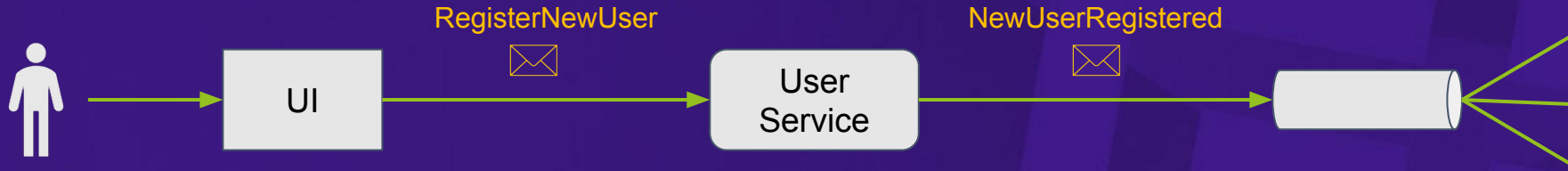
Control Access

Discoverability

Anti-Pattern: Everything is an Event



Command vs. Event




```
{
  “type”: “NewUserRegistered”,
  “mail”: “frank@dx.front”,
  “timestamp”: “2024-07-31”
}
```

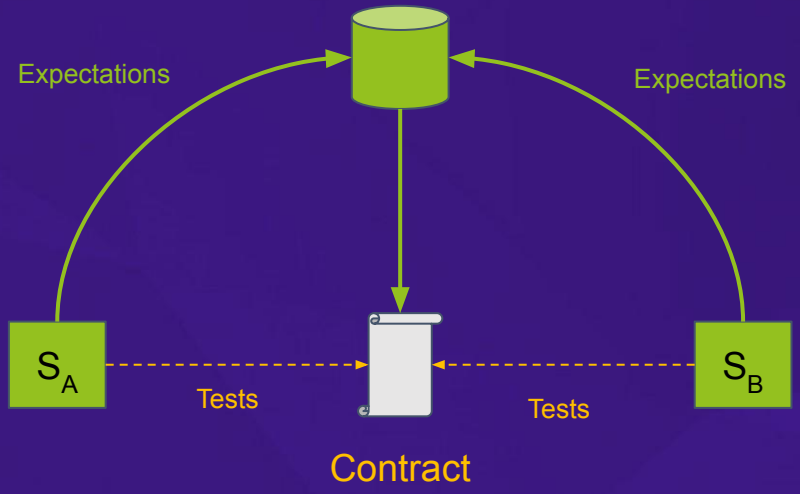
Event

```
{
  “type”: “RegisterNewUser”,
  “mail”: “frank@dx.front”,
}
```

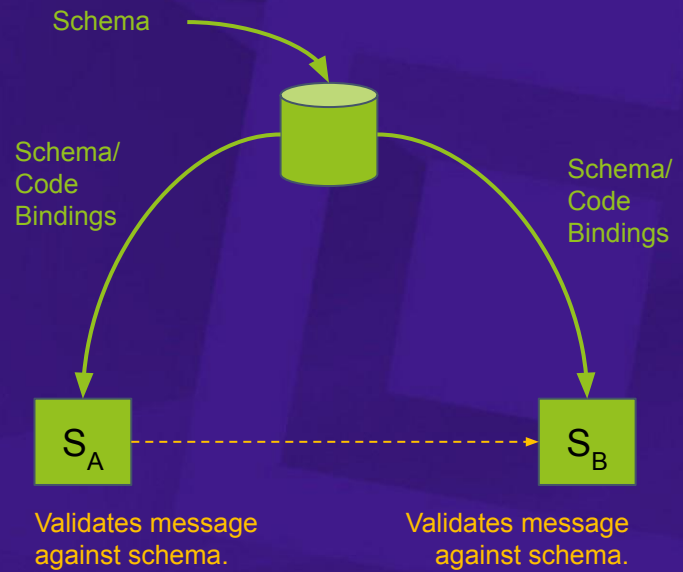
Command

Anti-Pattern: Implicit Coupling

Contract Tests & Schema Registry

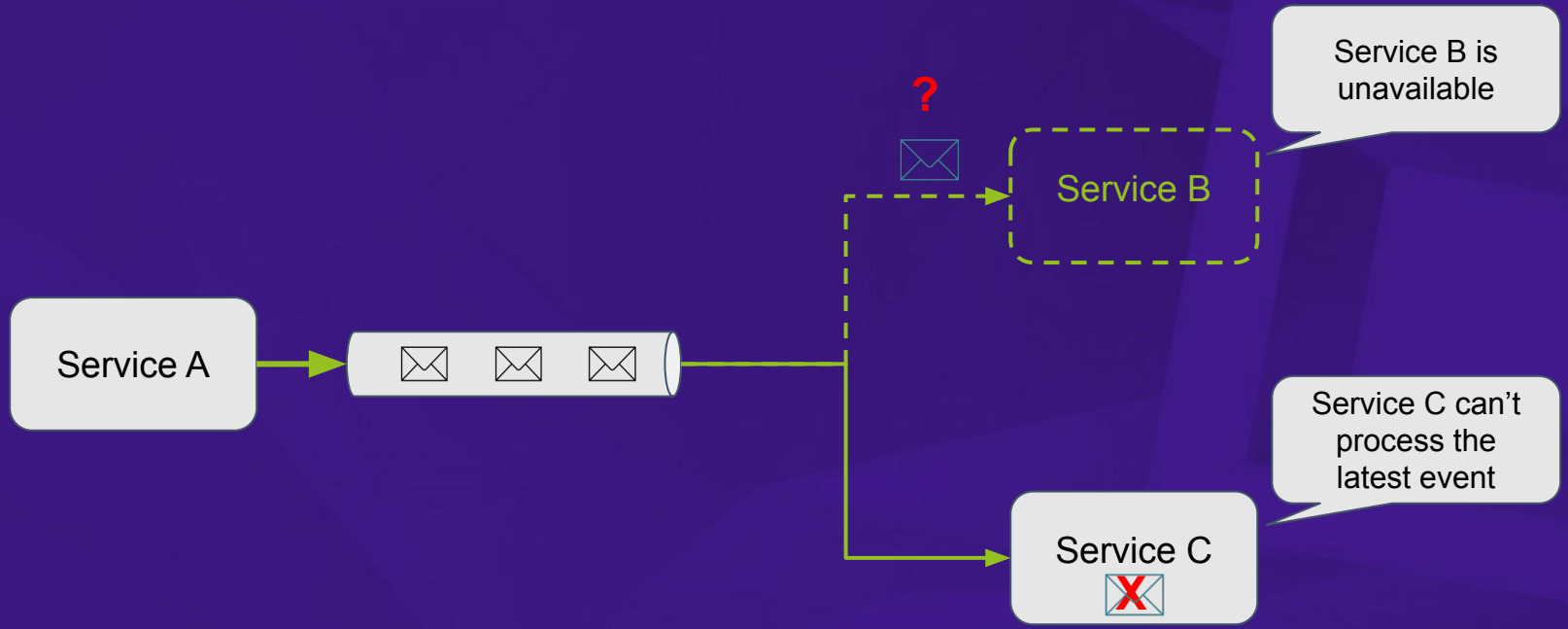


Contract Tests

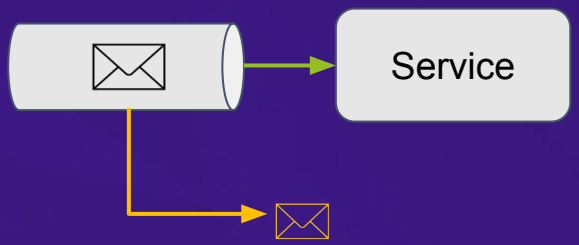


Schema Registry

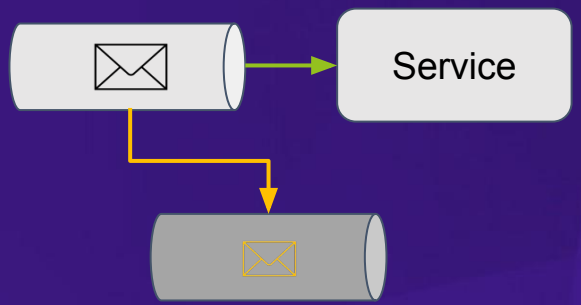
Anti-Pattern: False trust in resiliency



Handling Failing Event Delivery & Idempotency



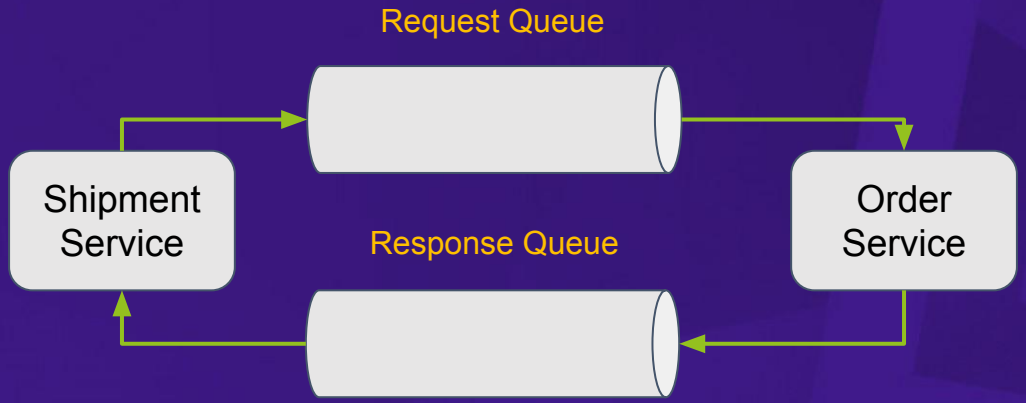
Drop Events



Dead Letter Queue

Idempotency

Anti-Pattern: Everything Async



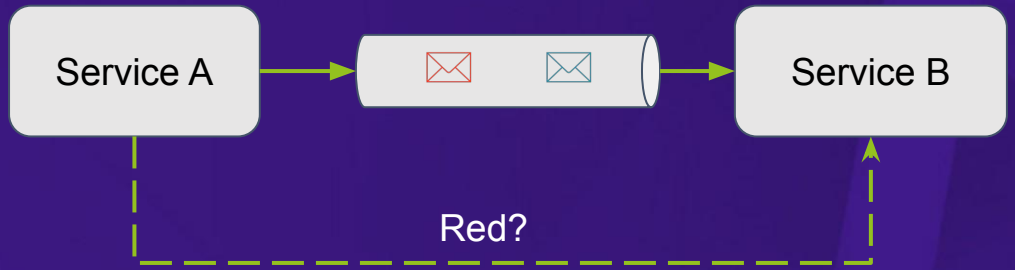
Establish Common Architectural Vision

Language

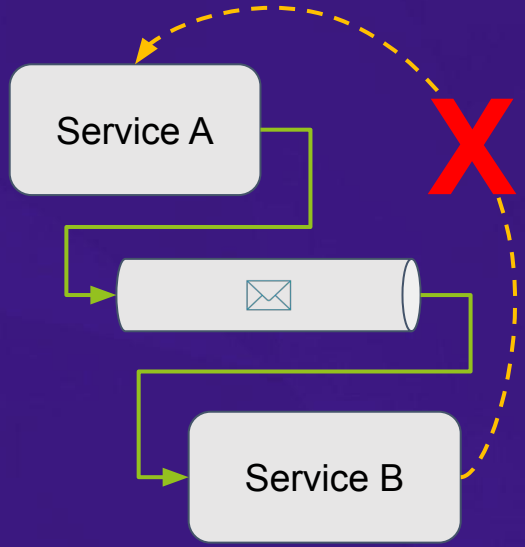
Goals

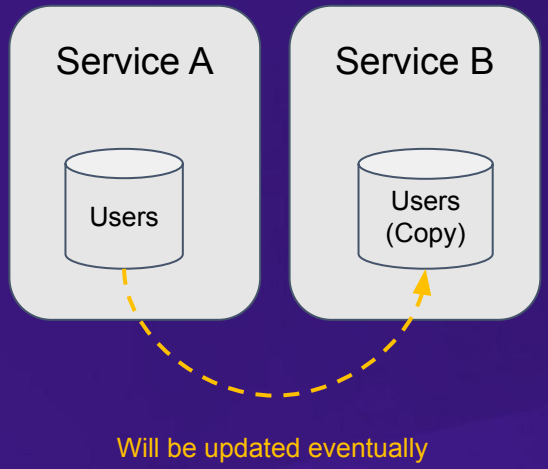
Standards

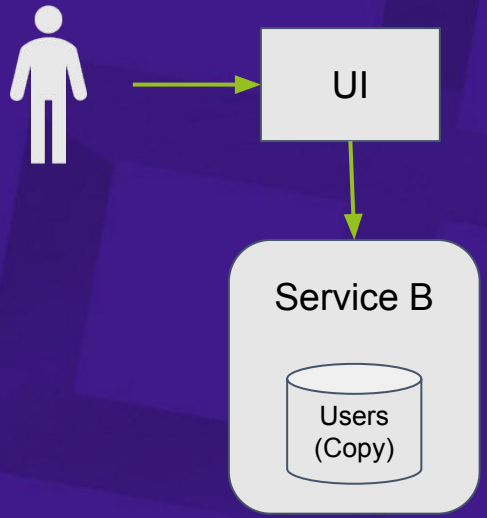
Anti-Pattern: Read your own writes



Eventual Consistency







Prefer availability over consistency

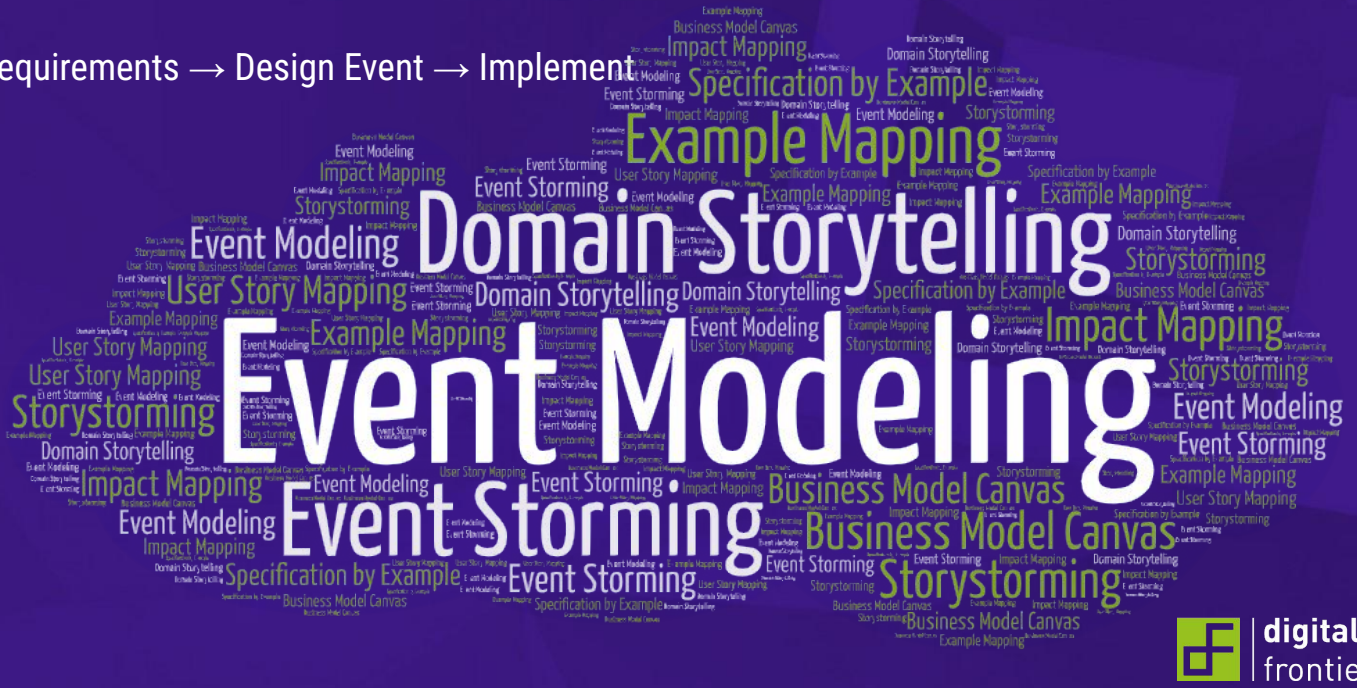
Where do you really need strict consistency?

Anti-Pattern: Implementation First

- Typical journey:
  - Every team starts designing and publishing events
  - Infrastructural topics like topic design and standards for event metadata arise
  - How to handle event schema evolution?
  - Do we have a shared understanding of event-driven and its implications?
  - Why does event X not contain information Y?

Events first!

- Create common language
- Discuss and understand overall process
- Identify key events
- Establish: Discuss Requirements → Design Event → Implement



Conclusion

- EDAs have many advantages: flexibility, scalability, loose coupling, availability, robustness
- Introducing and Evolving EDAs can be hard, due to
  - wrong event design
  - not leveraging event data
  - not existing governance concepts, i.e. discoverability
  - not defining what happens on event delivery failures
  - not ensuring ability to evolve cent schemas
  - lack of common understanding of the architectural style
  - prioritizing implementation over understanding
- Biggest anti-pattern: lack of communication!

Just don’t do it!

Anti-Patterns in Event-getriebenen Architekturen

Florian Pfleiderer

florian.pfleiderer@digitalfrontiers.de
https://www.linkedin.com/in/pfleidfn/

https://blog.digitalfrontiers.de

https://github.com/dxfrontiers

@dxfrontiers

Frank Steimle

frank.steimle@digitalfrontiers.de
https://www.linkedin.com/in/frank-steimle/