Managing Failures in Distributed Transactions

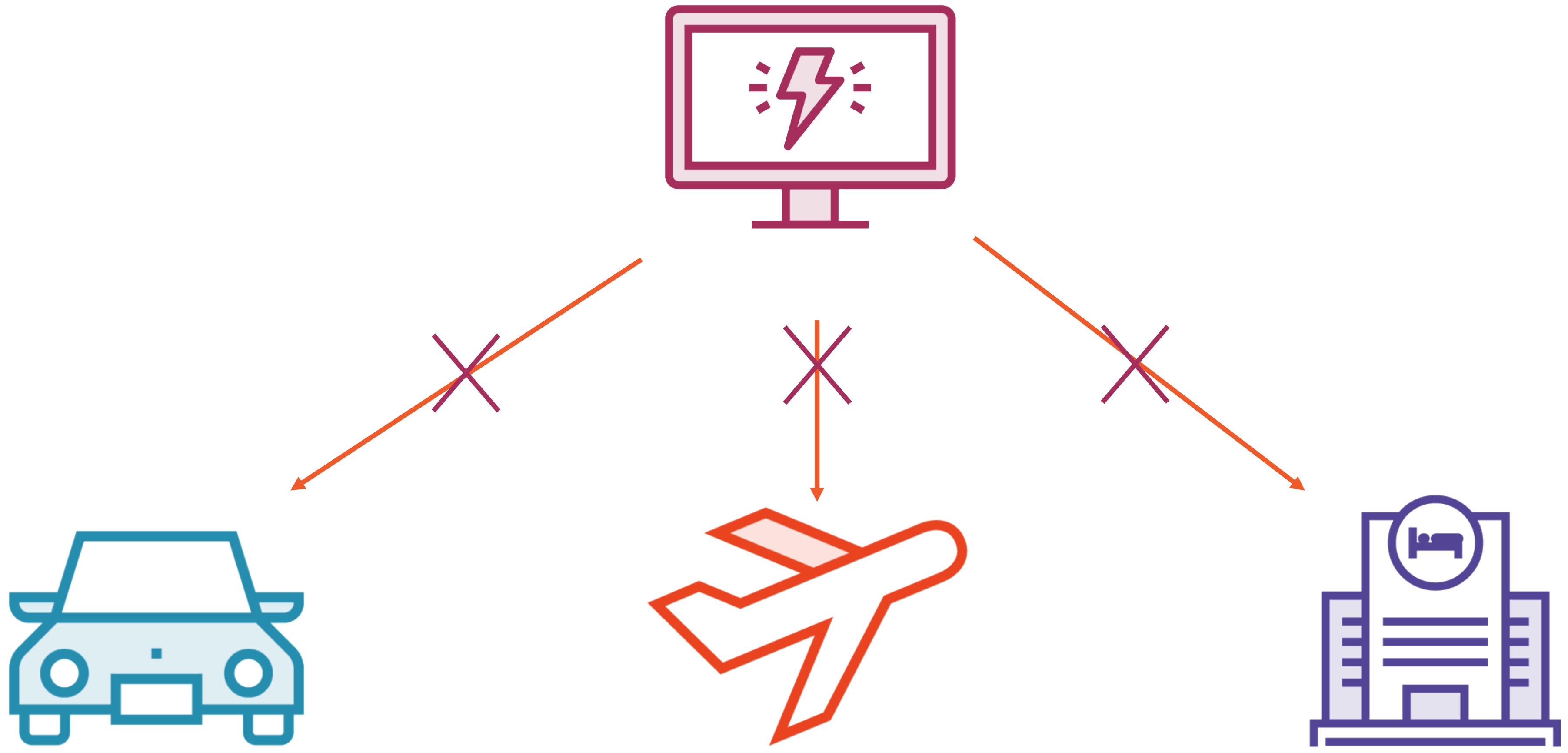
Matthew Alexander
SOFTWARE ENGINEER

@alexandermj

Transaction

An “all or nothing” unit of work.

A Real World Example



Document Translation Platform

Potential to suffer from the same problem

What if we can’t lock a document?

What if no one accepts the request?

What is a translation request is rejected?

SAGAs

A Brief History Around SAGAs

Introduced by Hector Garcia Molina and Kenneth Salem in 1987

Concerned about long lived transaction causing resource exhaustion

Break apart long lived transaction into interleaving smaller transactions

Types of SAGAs

Choreographed

Orchestrated

Choreographed SAGAs

Centered on events going into global event store

Publishers

Consumers

Orchestrated SAGAs

Centered on commands rather than events

Publishers

Consumers

Remaining Problems In Current Architecture

Certain assumptions are naive

Missing ability to roll back in the face of failure

Designing a State Machine

State Machine

Visual abstraction

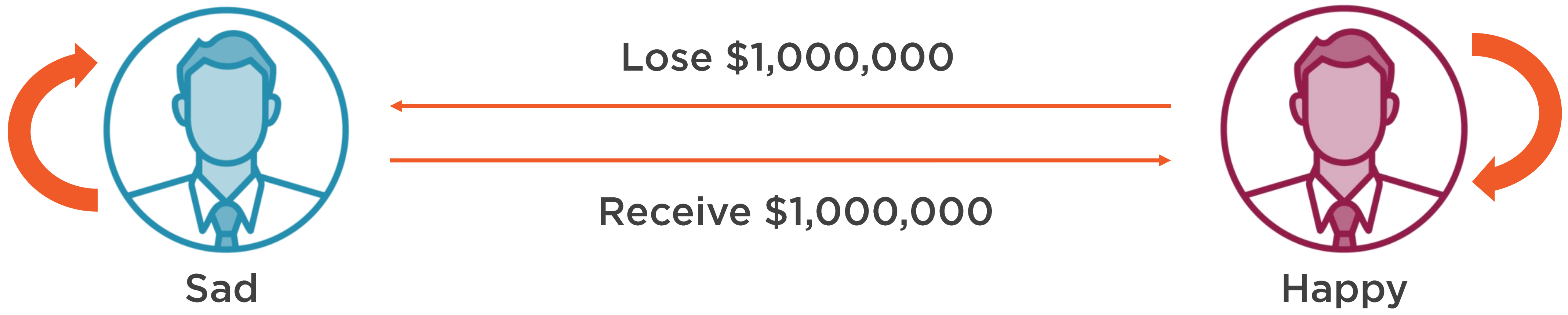
States

Conditions

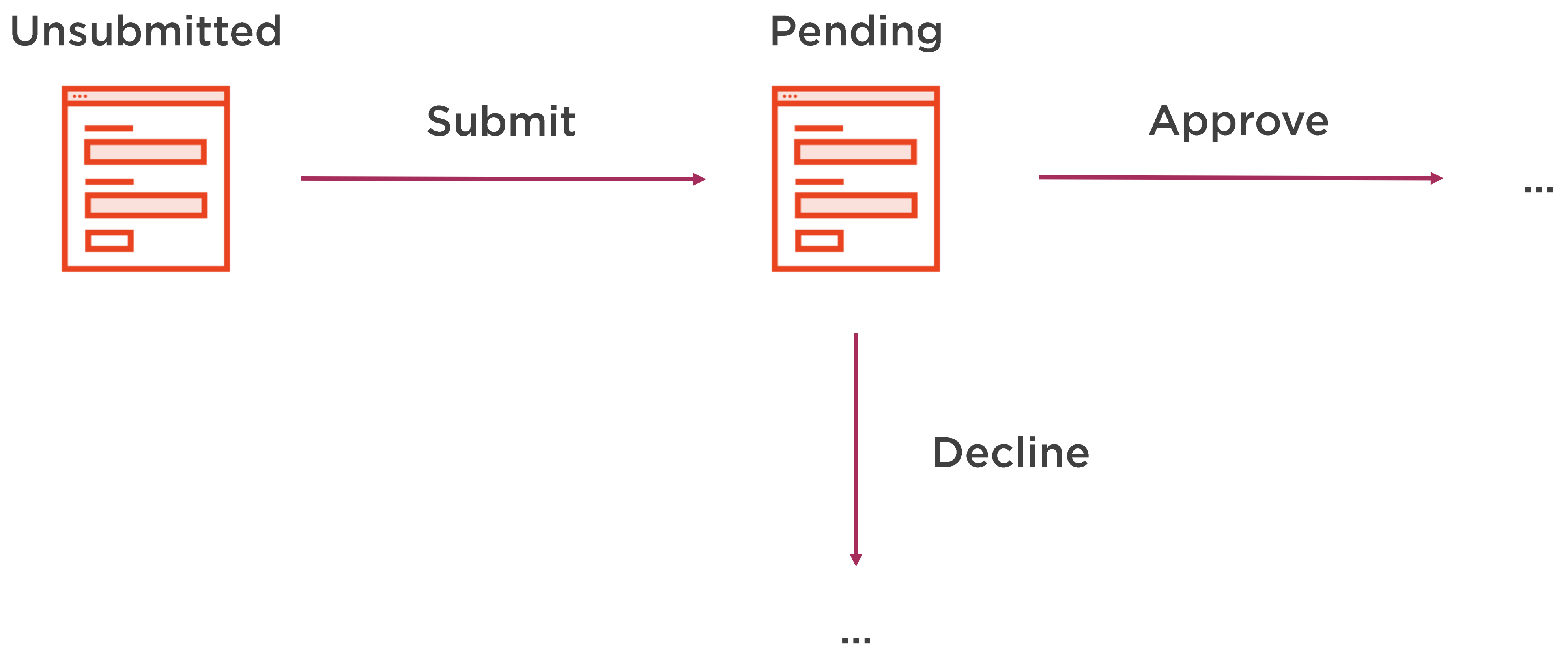
Transitions

Inputs

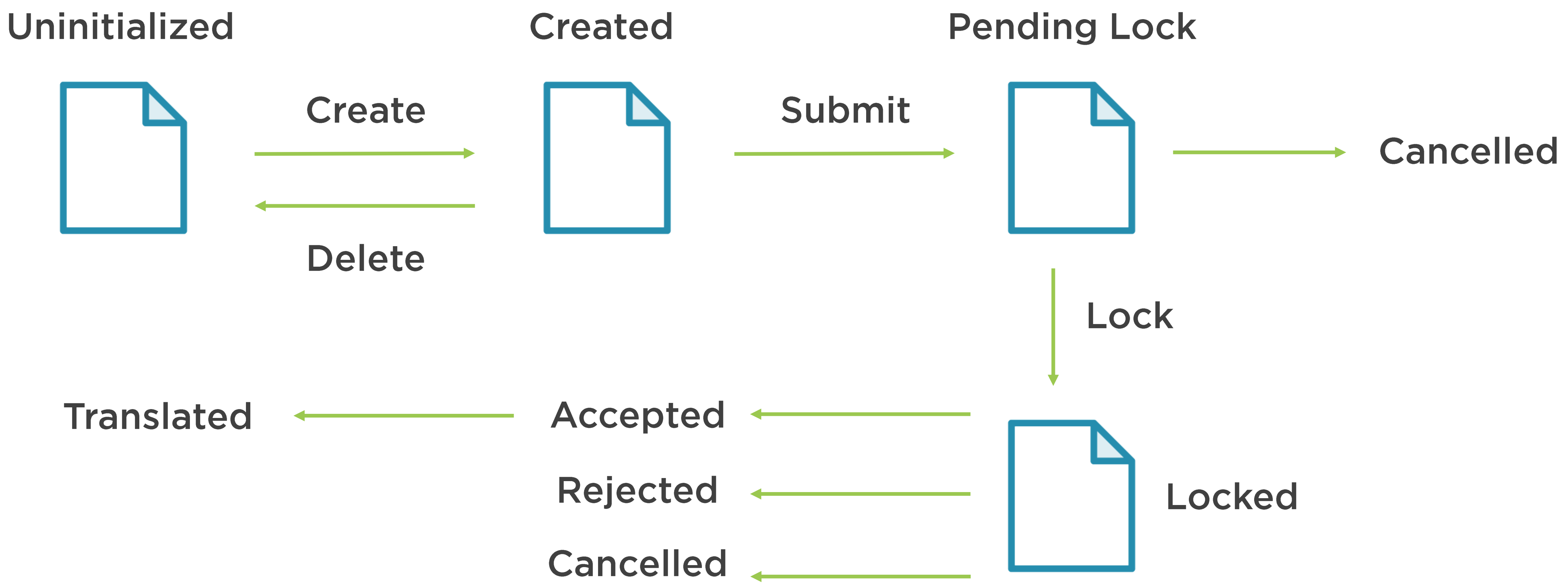
A Simple State Machine



A Real World Example

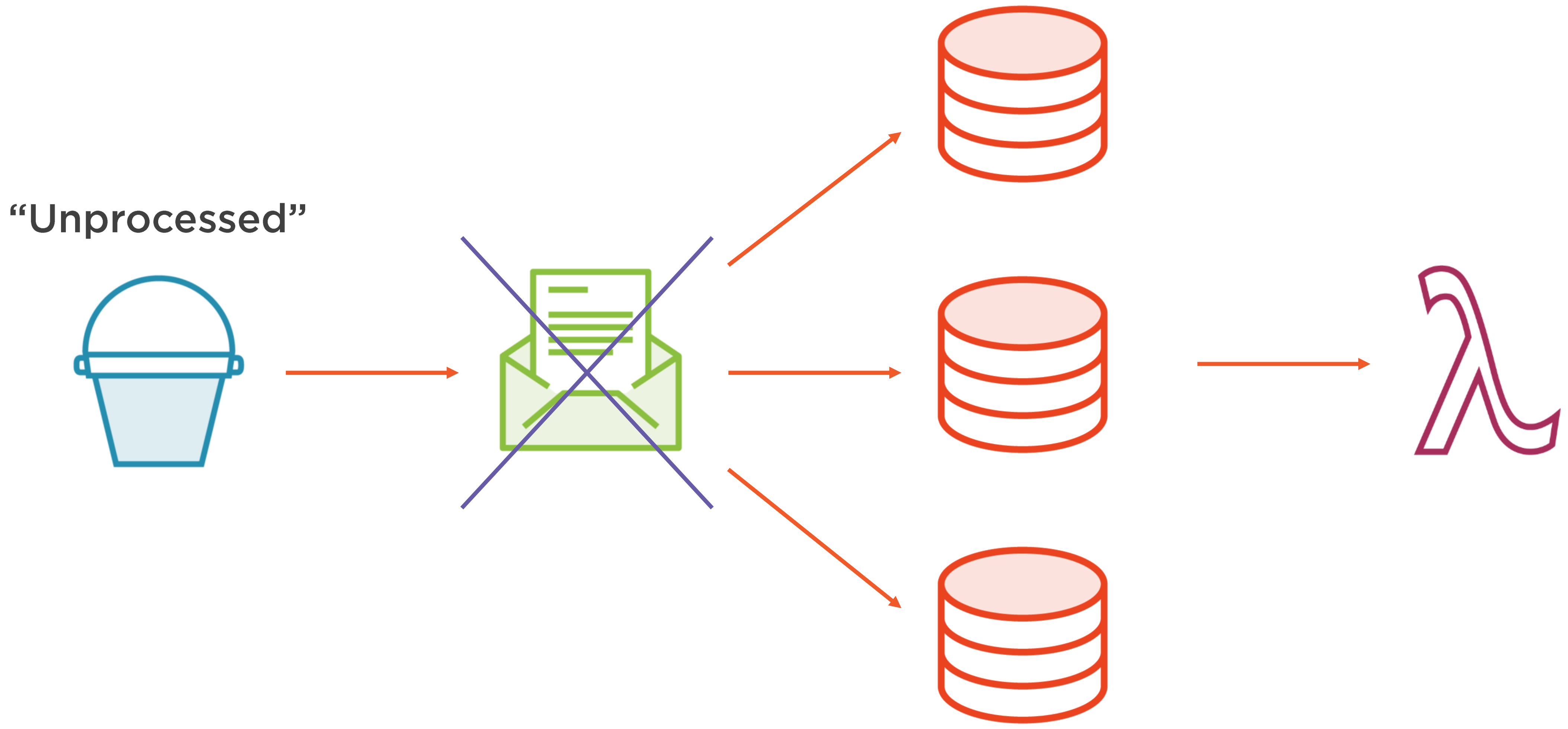


Document State Machine



Accounting for Missed Notifications

External Document Platform



Polling For Missed Notifications

Can be efficient

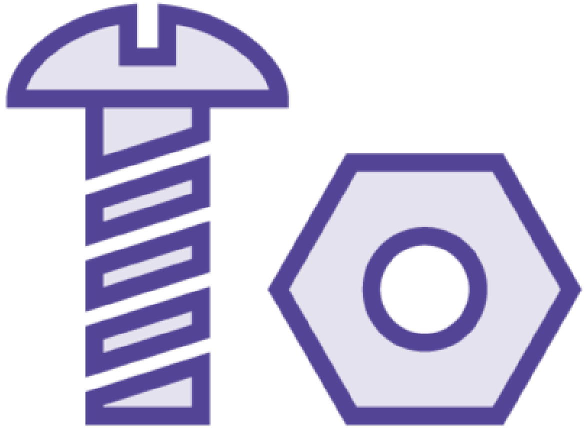
Stateful

Partitioned for elasticity, responsiveness, and resilience

Polling For Missed Notifications

Modeling Rejected Translation Requests

Translation Submission SAGA



Submit a request to have a document translated

Lock the document to prevent against modification

Notify translators of the pending request

Unlock the document on completion or failure

Notify the customer of the document's new status

Acting on Failure Touch Points

If a submission request fails to get persisted to our database then we have nothing to rollback

If we fail to lock a document, we will need to update the pending submission request with the appropriate notification status

If we fail to notify translators for whatever reason, rollback all changes

Implementing Rejection Handlers

Conclusion

The Reactive Manifesto

Resilient	Responsive
Elastic	Message Driven

Summary

Design decisions between microservices and monoliths

Two RESTful web services and a message bus

Durable messaging

Elastic capabilities

Idempotence, correlation identifiers, and commutative messaging

Implemented robust distributed transaction framework

Docker and Docker Compose