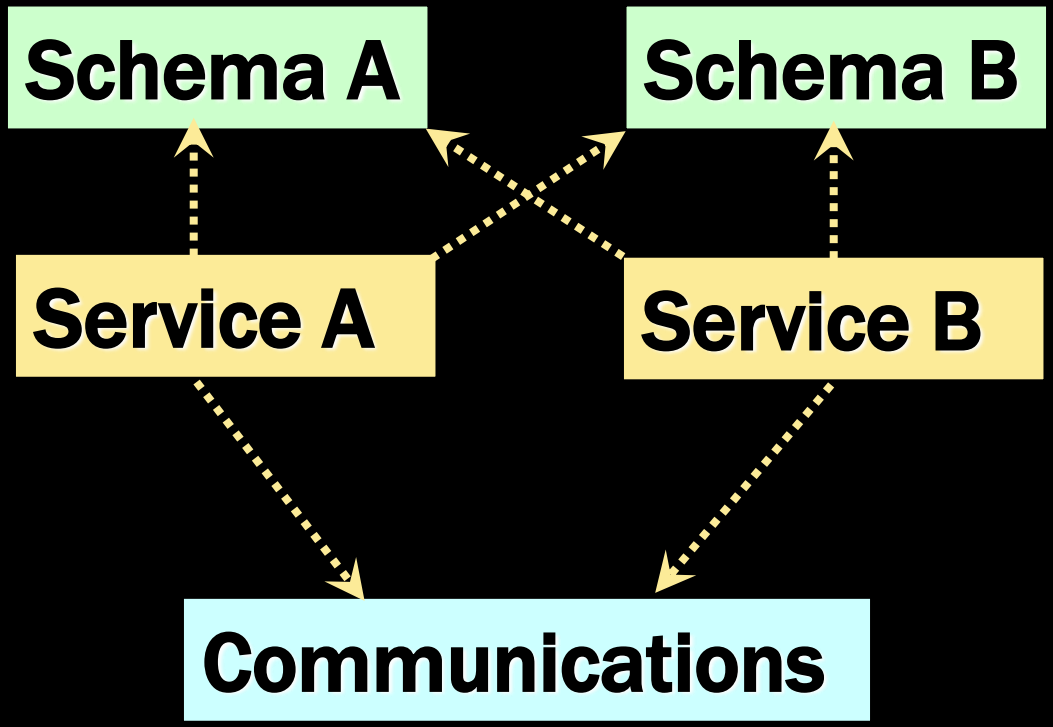
Messaging Patterns

Why Messaging?

- Reduces coupling
  - Use JSON/XML + AMQP for platform coupling
  - Use asynchronous messaging for temporal coupling
- Reduces afferent and efferent coupling while increasing autonomy

Messaging, Coupling, & Autonomy

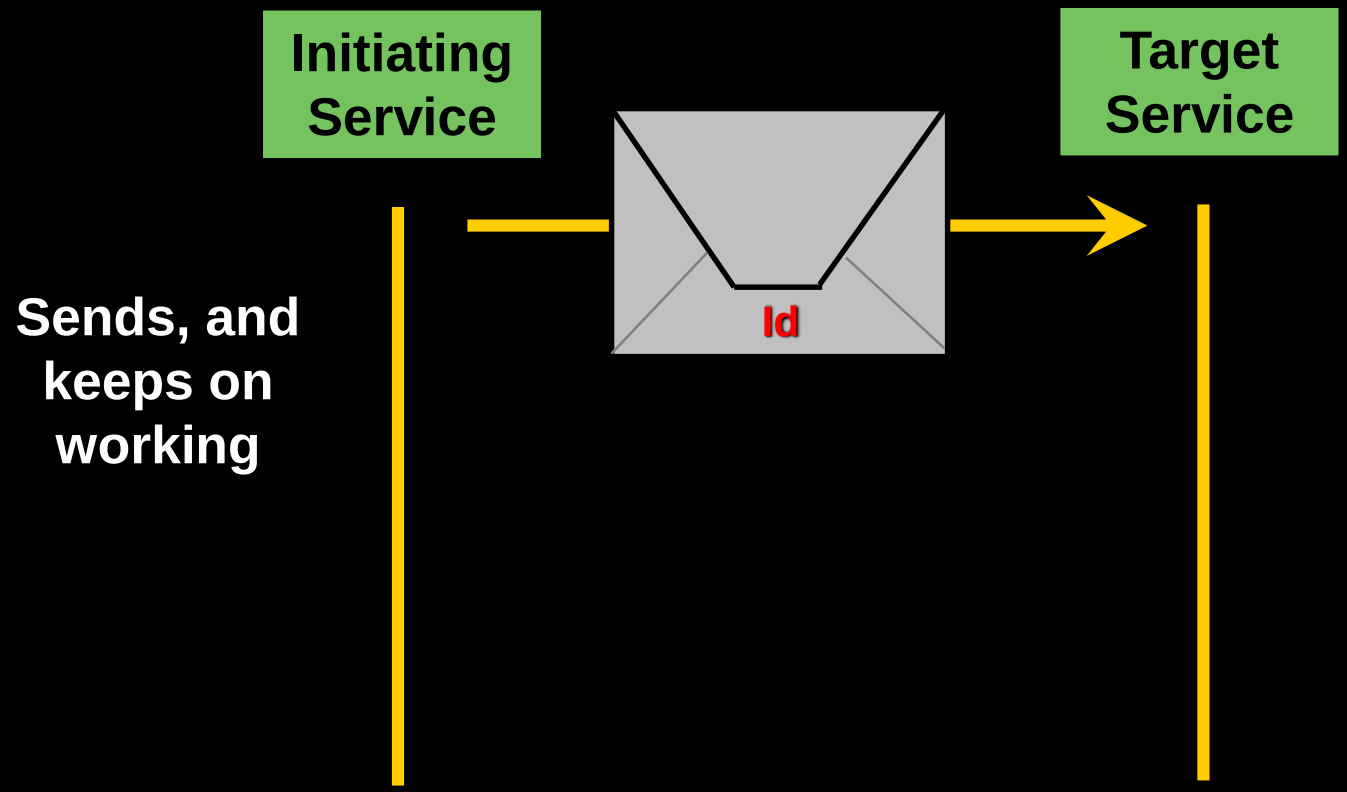


Service A and B don’t directly depend on each other

Asynchronous Messaging

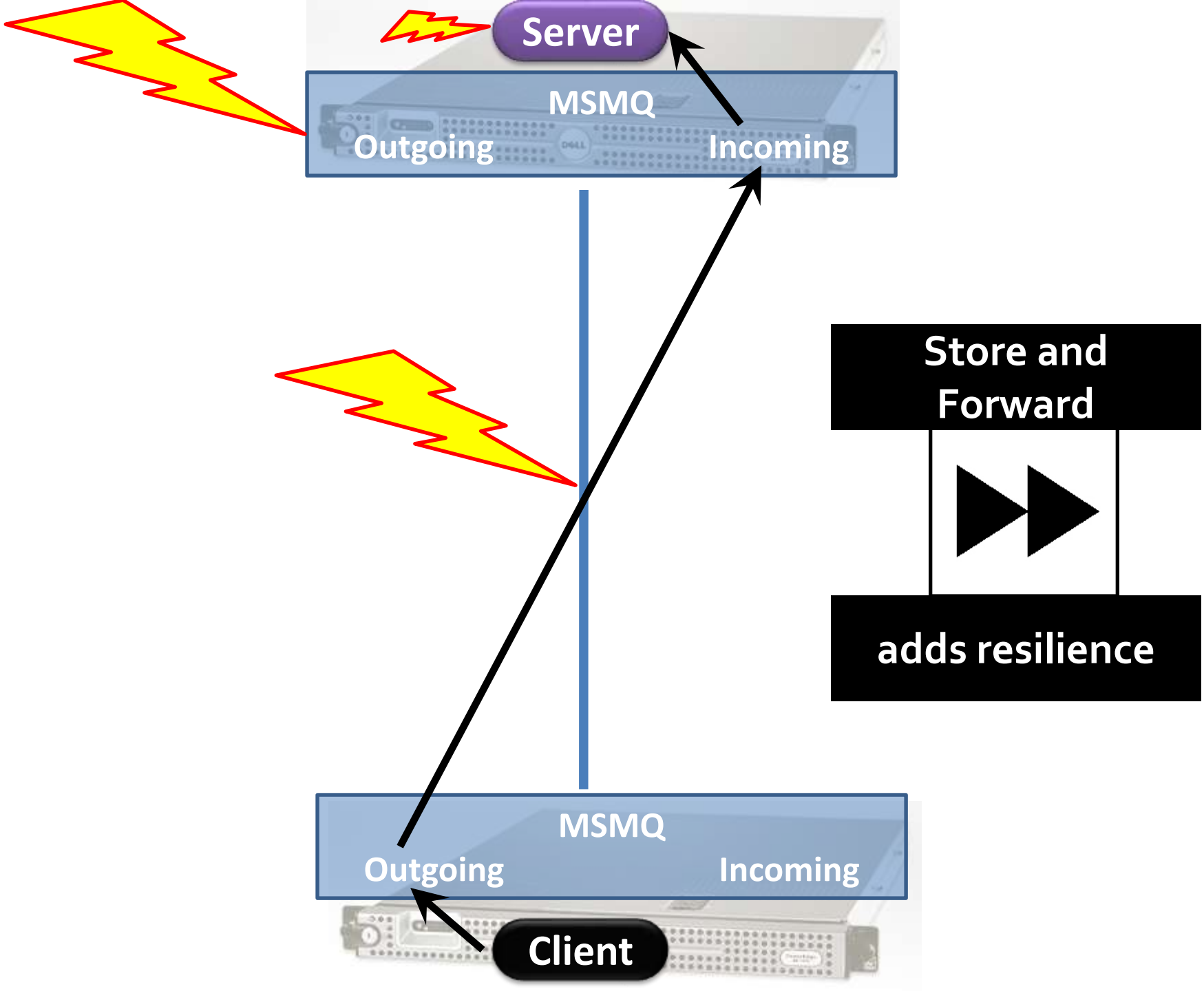
- It's all about one-way, fire & forget messages
- Everything is built on top of it
  - Return Address pattern
  - Correlated Request/Response
  - Publish/Subscribe

One-way, fire & forget messaging

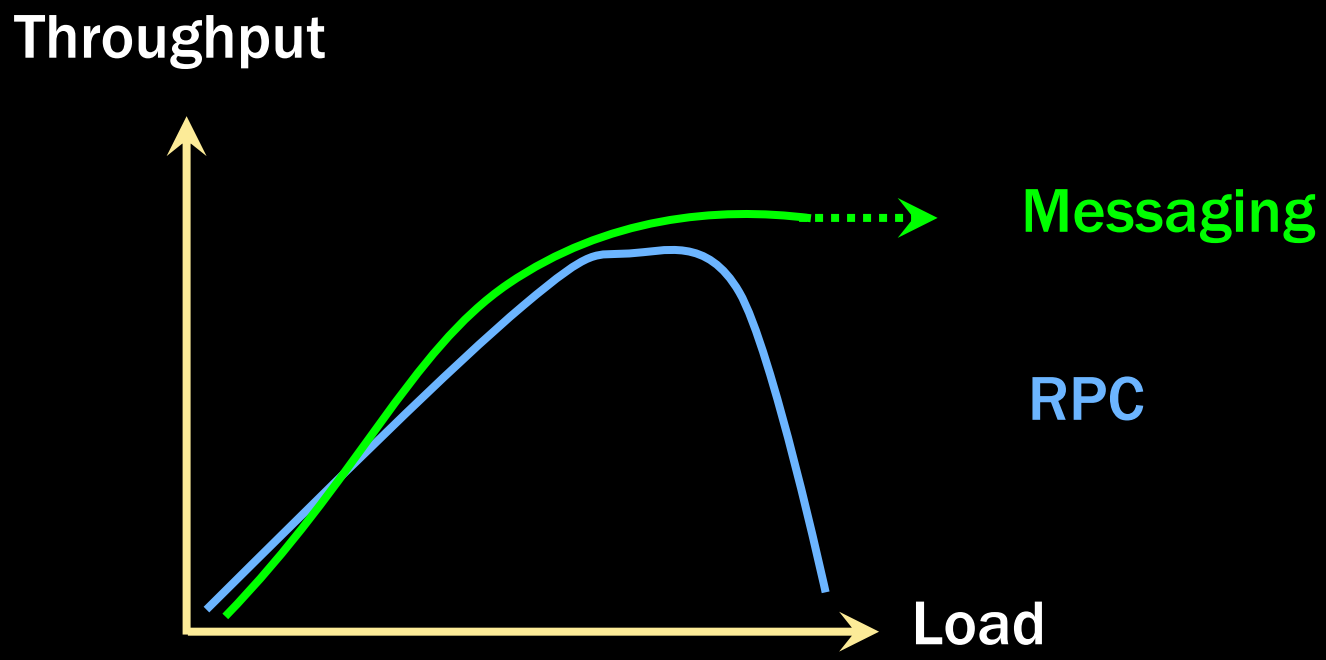


Each message has an Id.

Seems simple, but there's more to it.


Server
MSMQ
Outgoing
Incoming
Store and Forward
adds resilience
MSMQ
Outgoing
Incoming
Client

Performance – RPC vs Messaging



- With RPC, threads are allocated with load
  - With messaging, threads are independent
  - Difference due to synchronous blocking calls
- Memory, DB locks, held longer with RPC

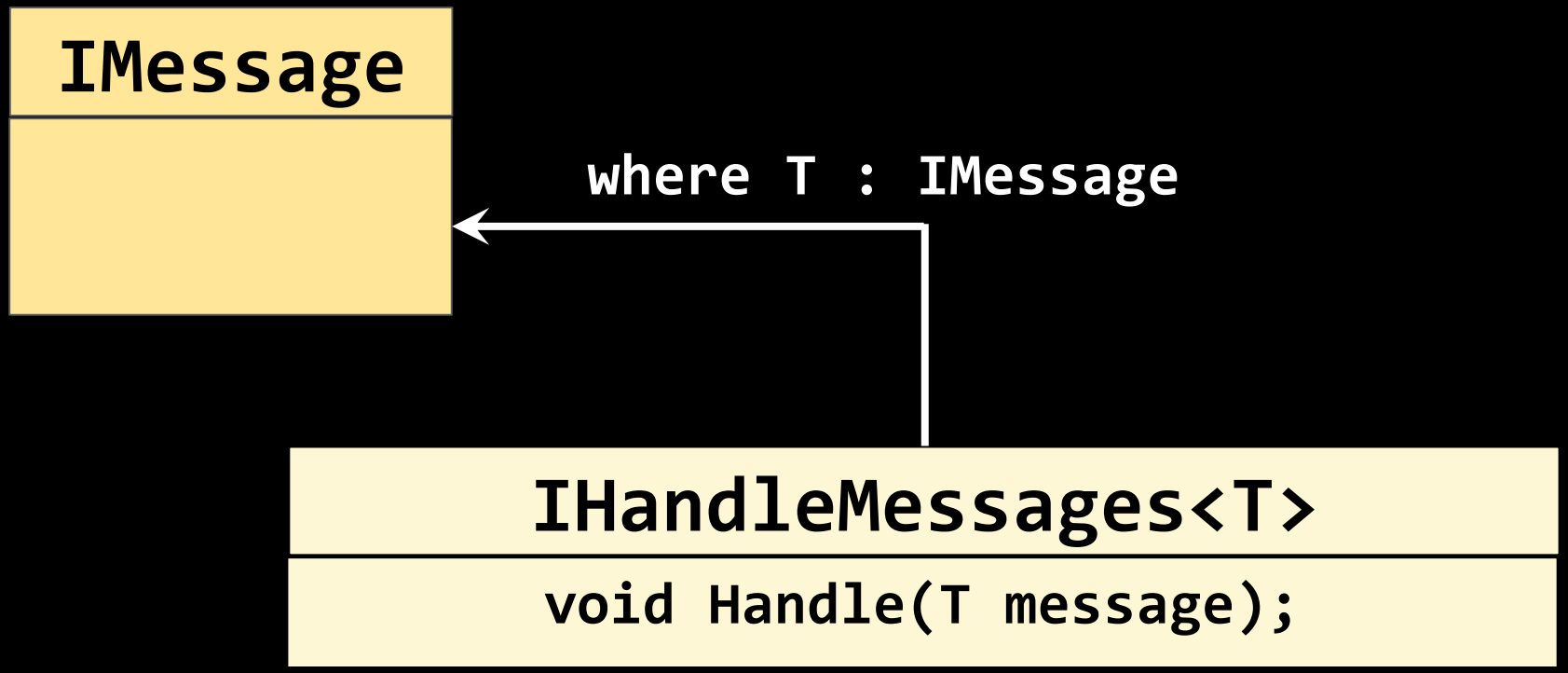
Standard service interfaces

Customer Service

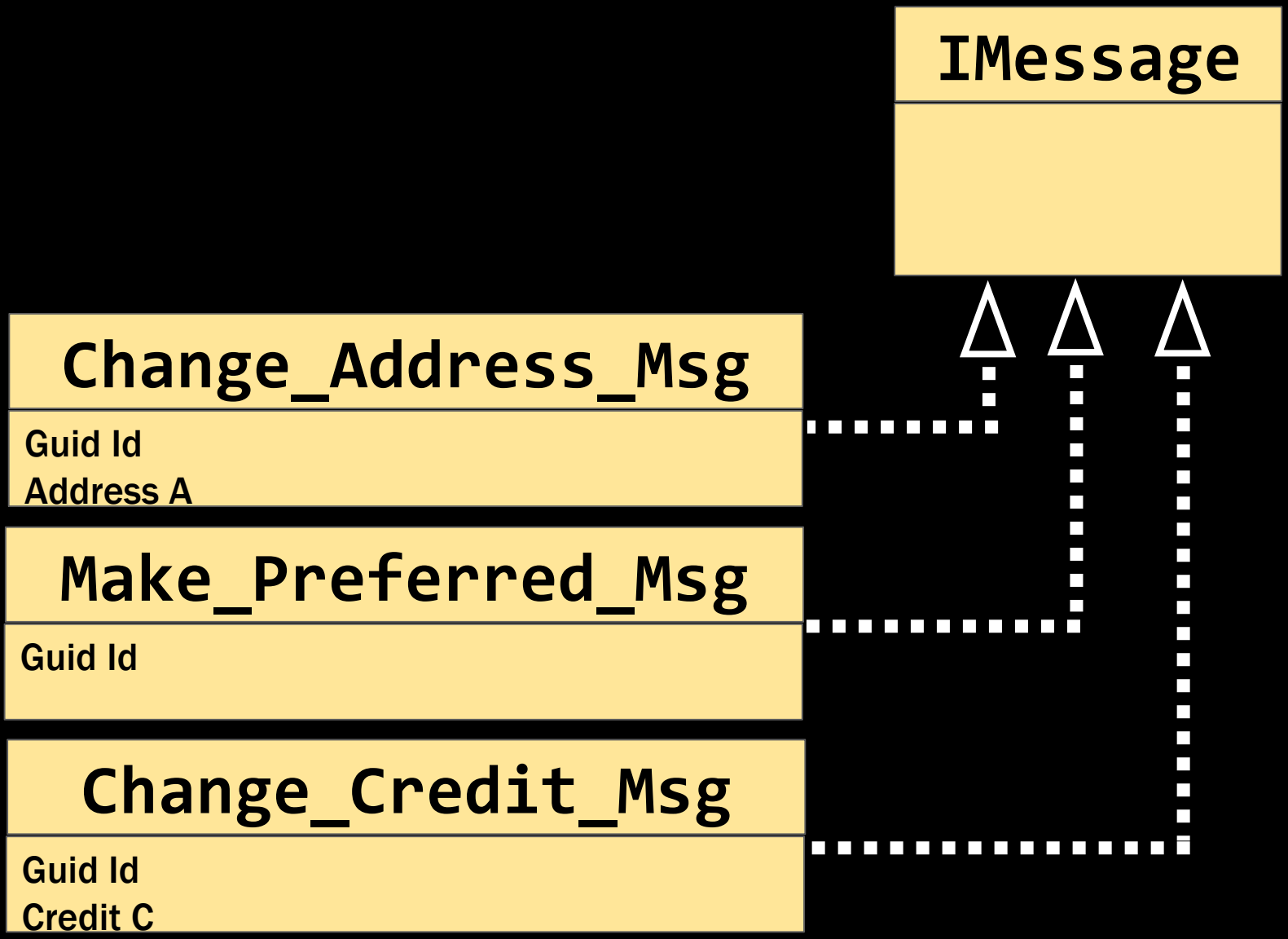
`void Change_Address(Guid id, Address a);`
 	
`void Make_Preferred(Guid id);`
 	
`void Change_Credit(Guid id, Credit c);`

- Problem is that service layers get too large
- Difficult for multiple developers to collaborate
- Difficult to reuse logging, authorization, etc

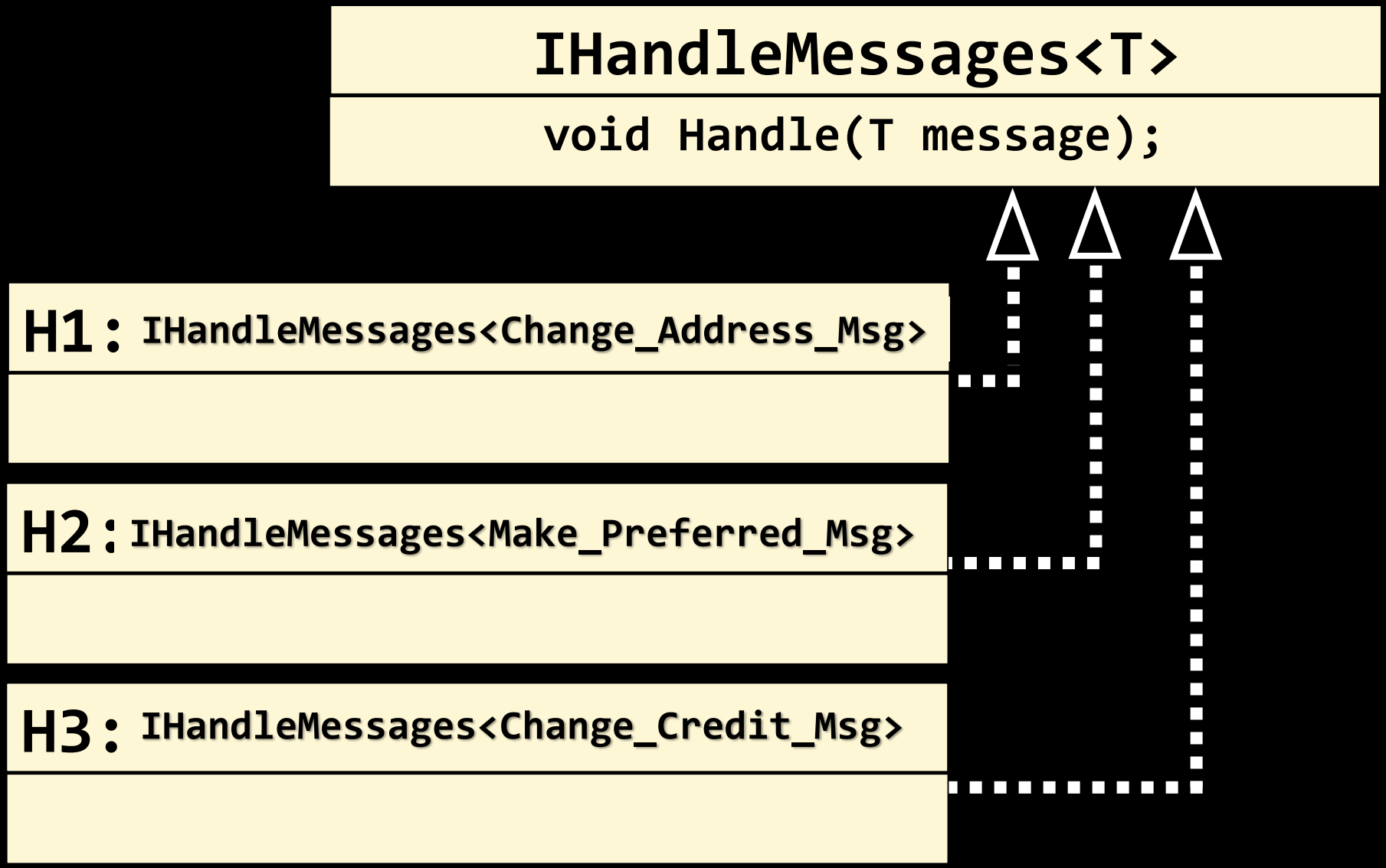
Exploit strongly-typed messages



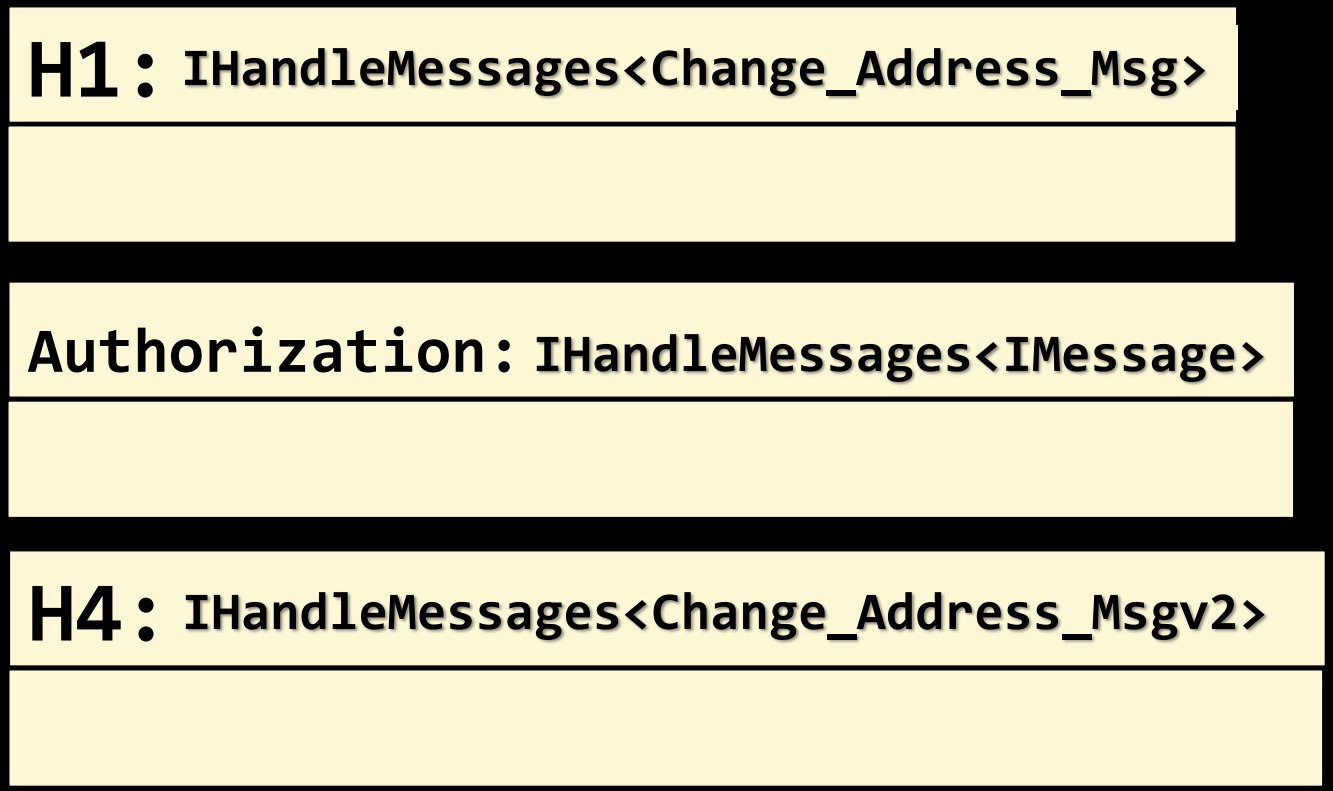
Represent methods as messages



Handling Logic Separated



Multiple handlers per message

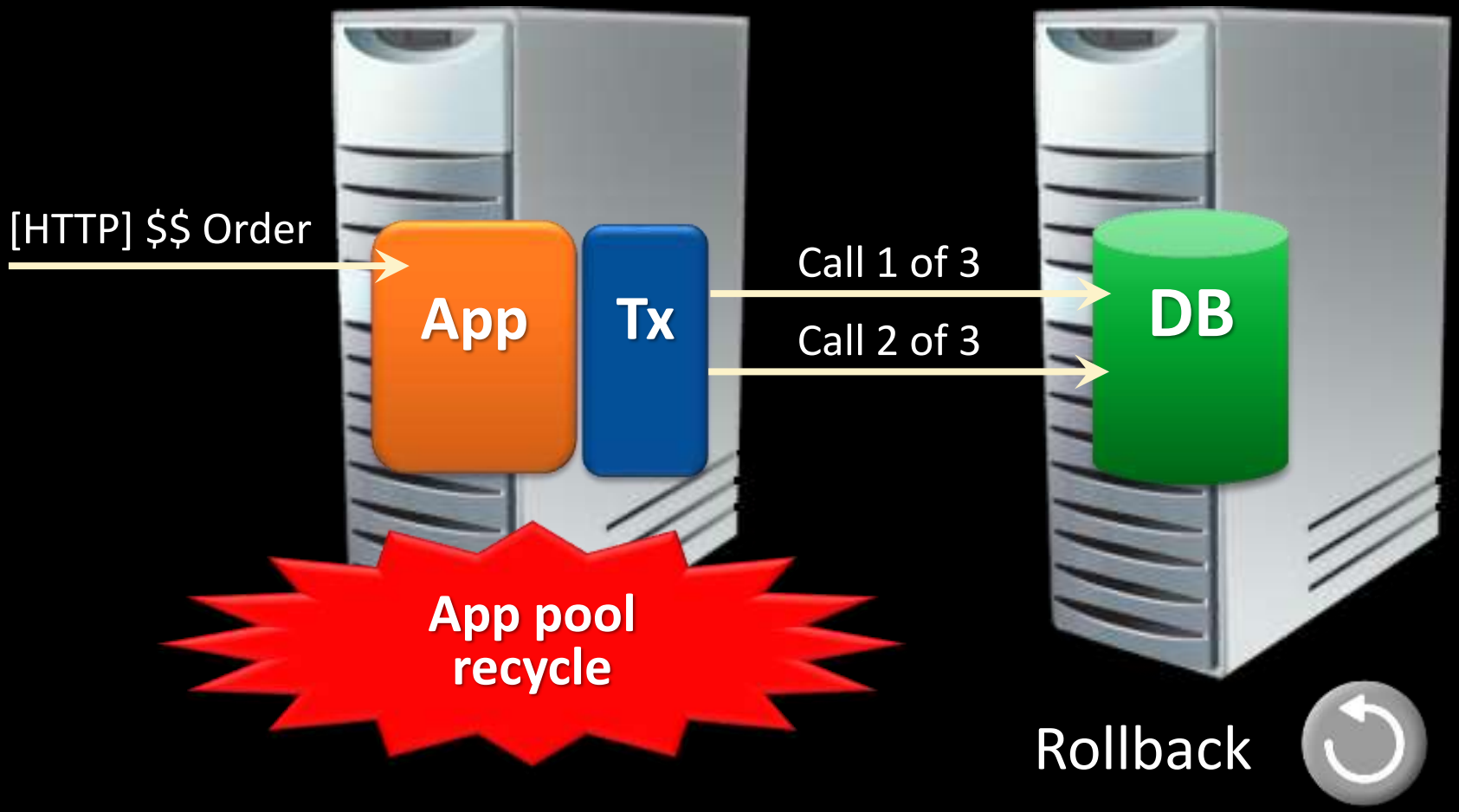


- Dispatch based on type polymorphism
- Allows for pipeline of handler invocation

Fault-tolerance - scenarios

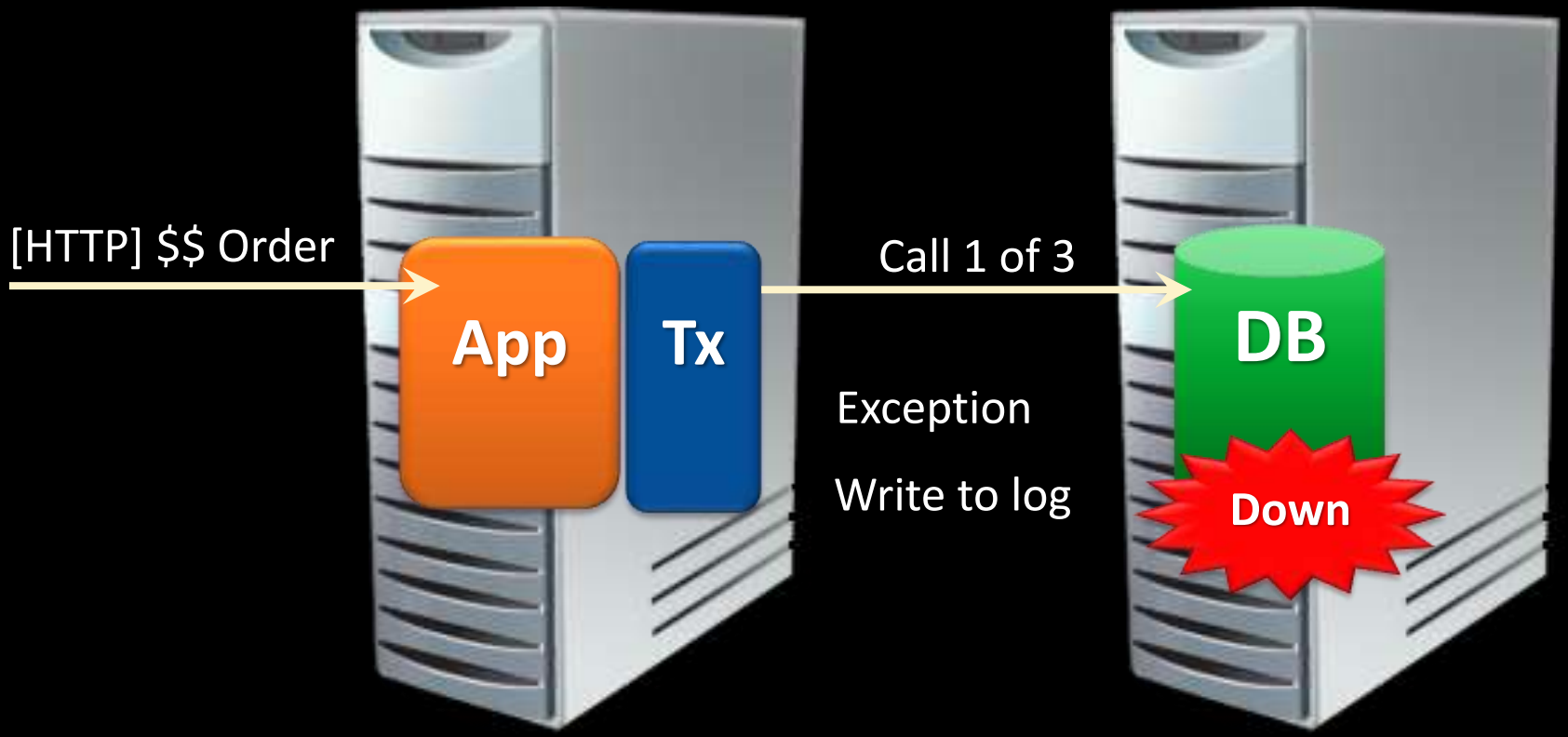
- When servers crash
- When databases are down
- When deadlocks occur in the database

When Servers Crash



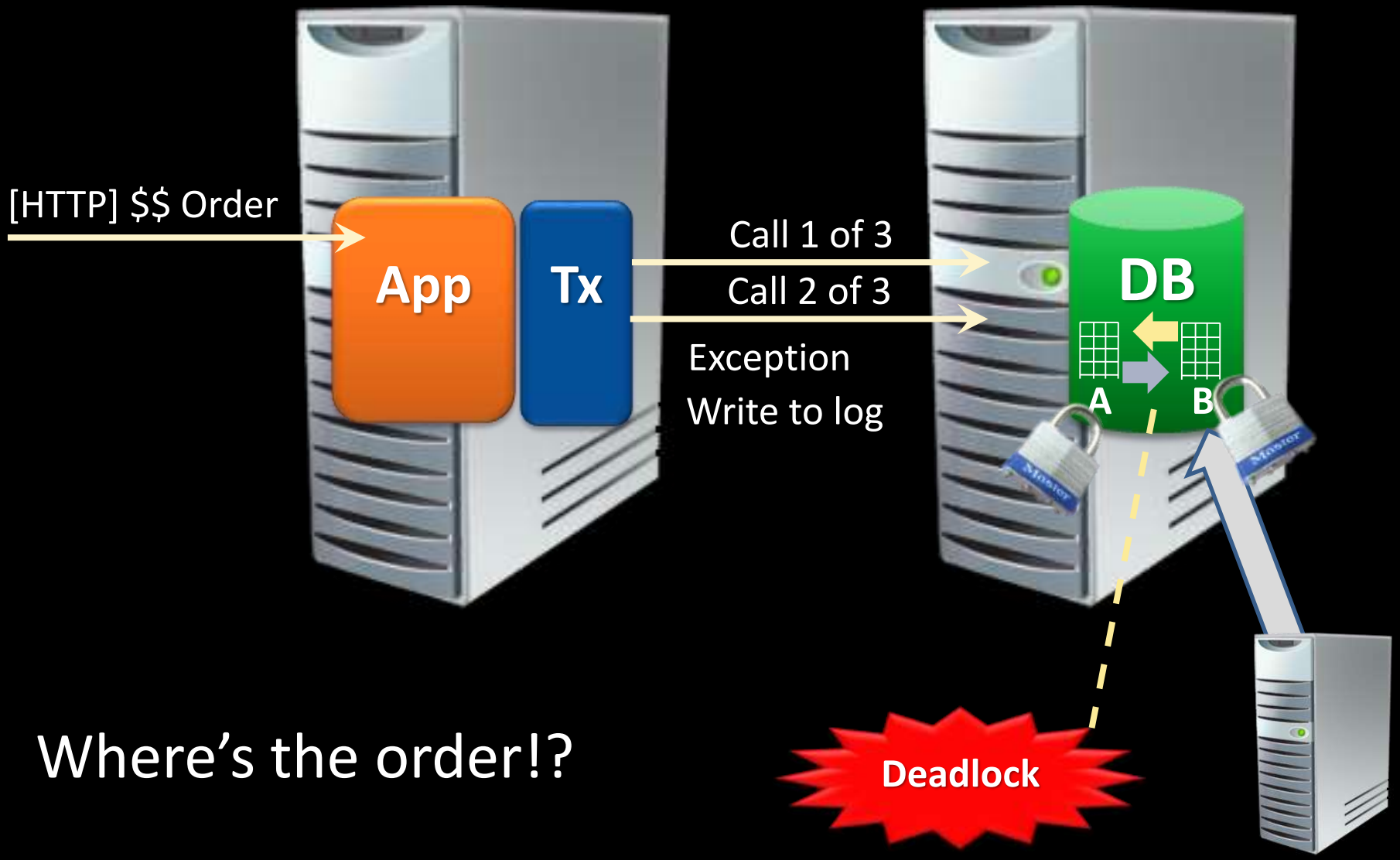
Where's the order!?

When Databases Are Down



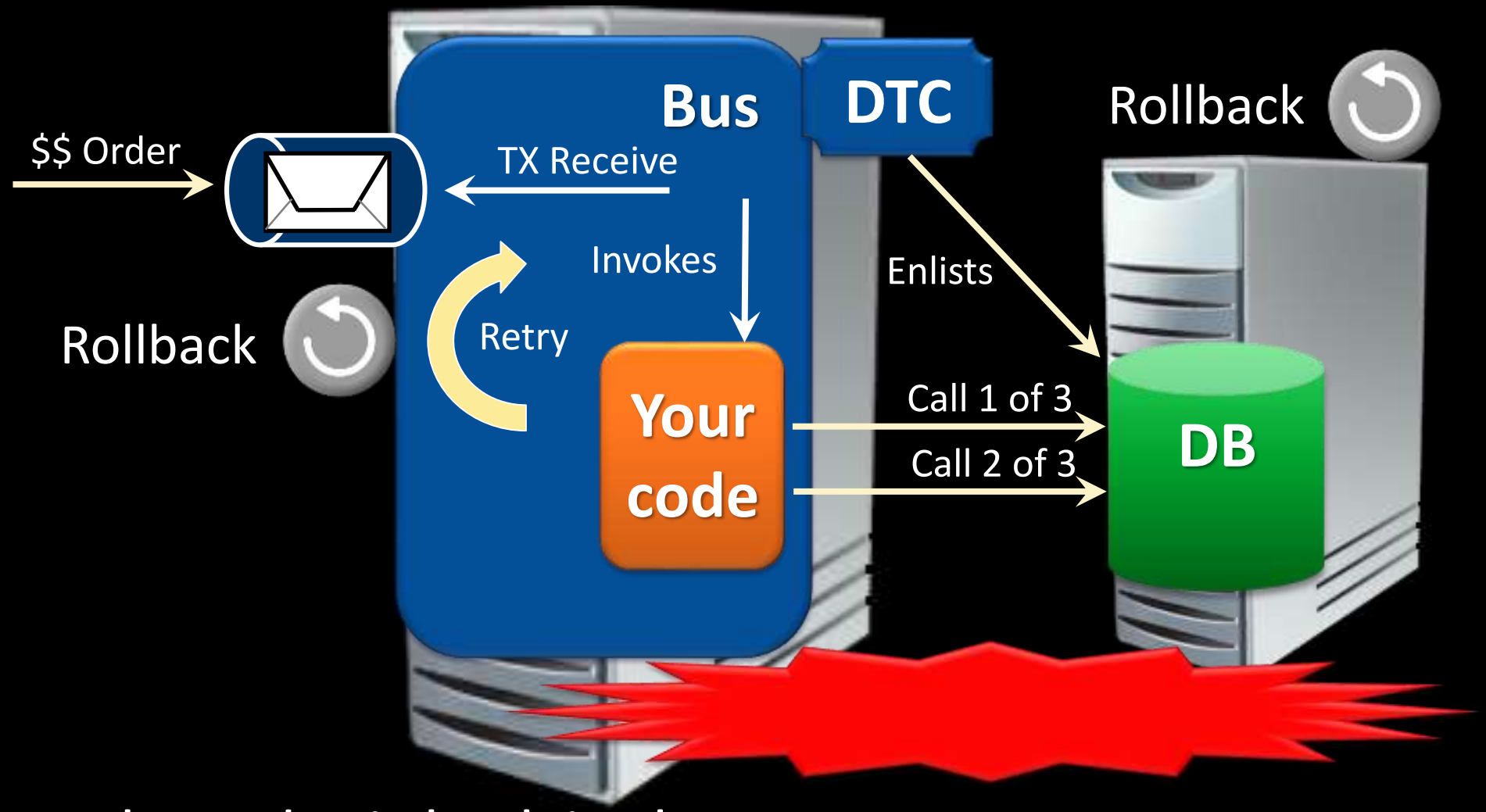
Where's the order!?

When Deadlocks Happen



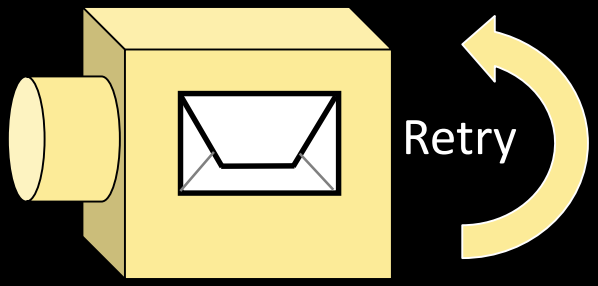
Where's the order!?

How Does Messaging Help?



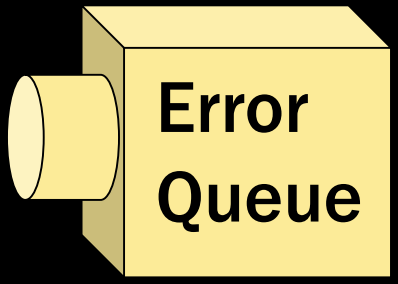
The order is back in the queue

After all retries exhausted



Append exception info to headers*

Moved failed message

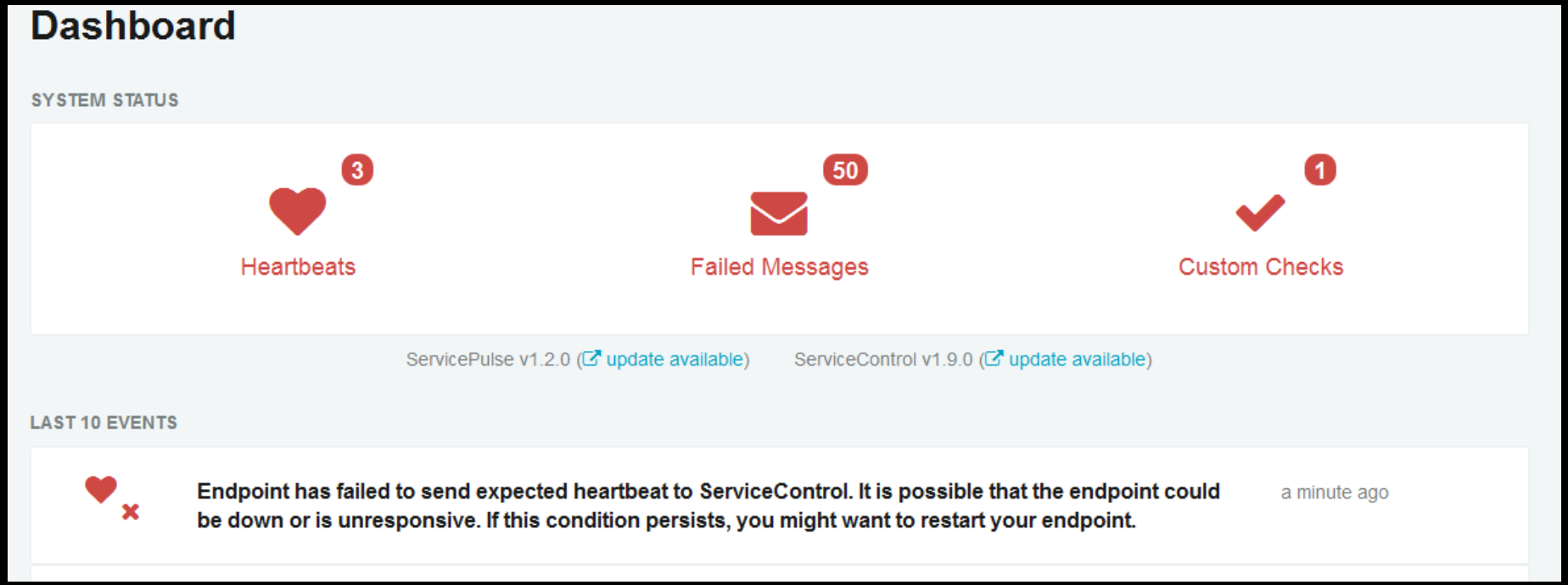


Notify admin

a.k.a “poison letter queue”

* NServiceBus feature – not done by all queues natively

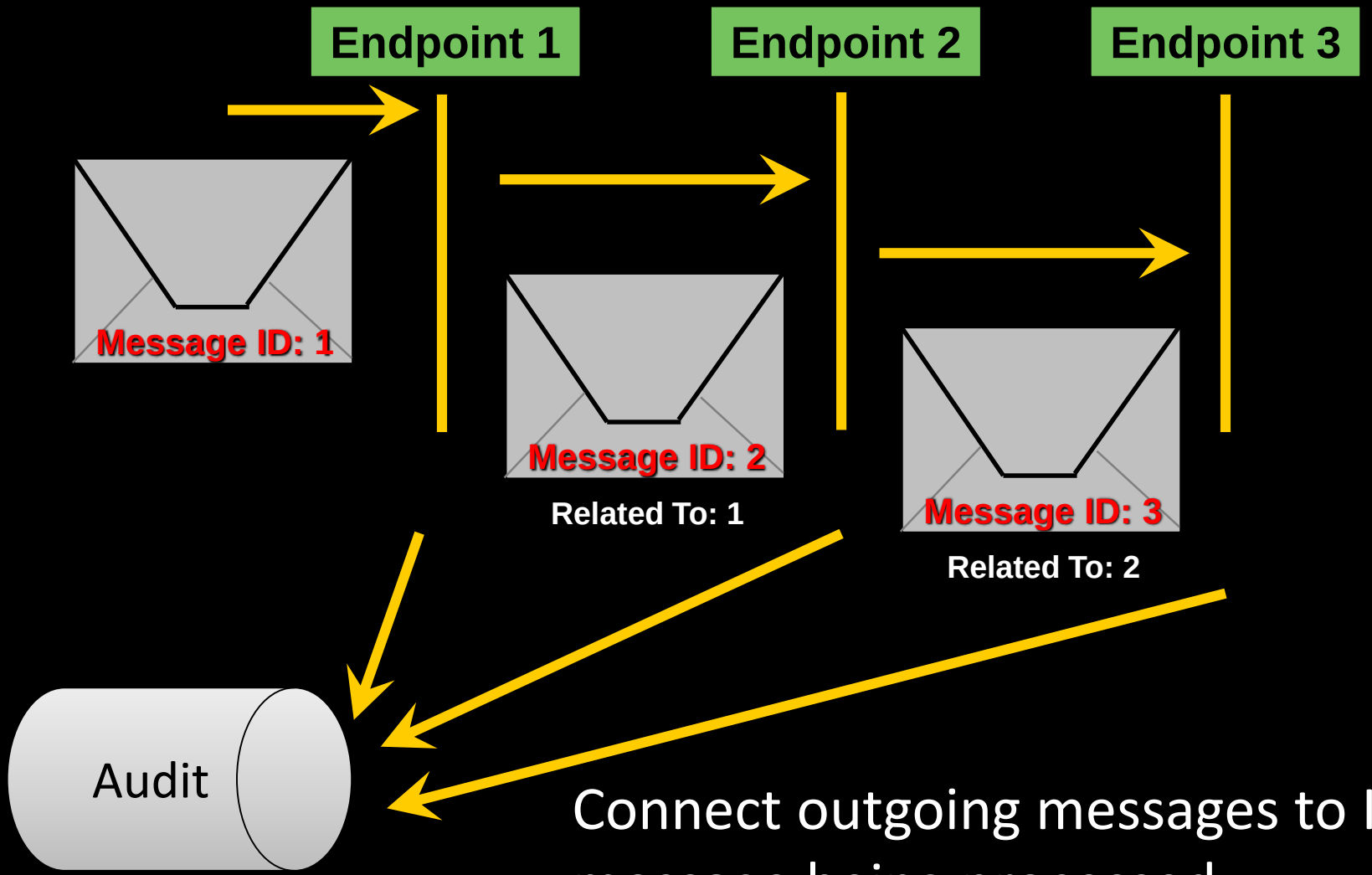
Monitoring


Dashboard
SYSTEM STATUS
3
Heartbeats
50
Failed Messages
1
Custom Checks
ServicePulse v1.2.0 (update available)
ServiceControl v1.9.0 (update available)
LAST 10 EVENTS
Endpoint has failed to send expected heartbeat to ServiceControl. It is possible that the endpoint could be down or is unresponsive. If this condition persists, you might want to restart your endpoint.
a minute ago

Auditing / Journaling

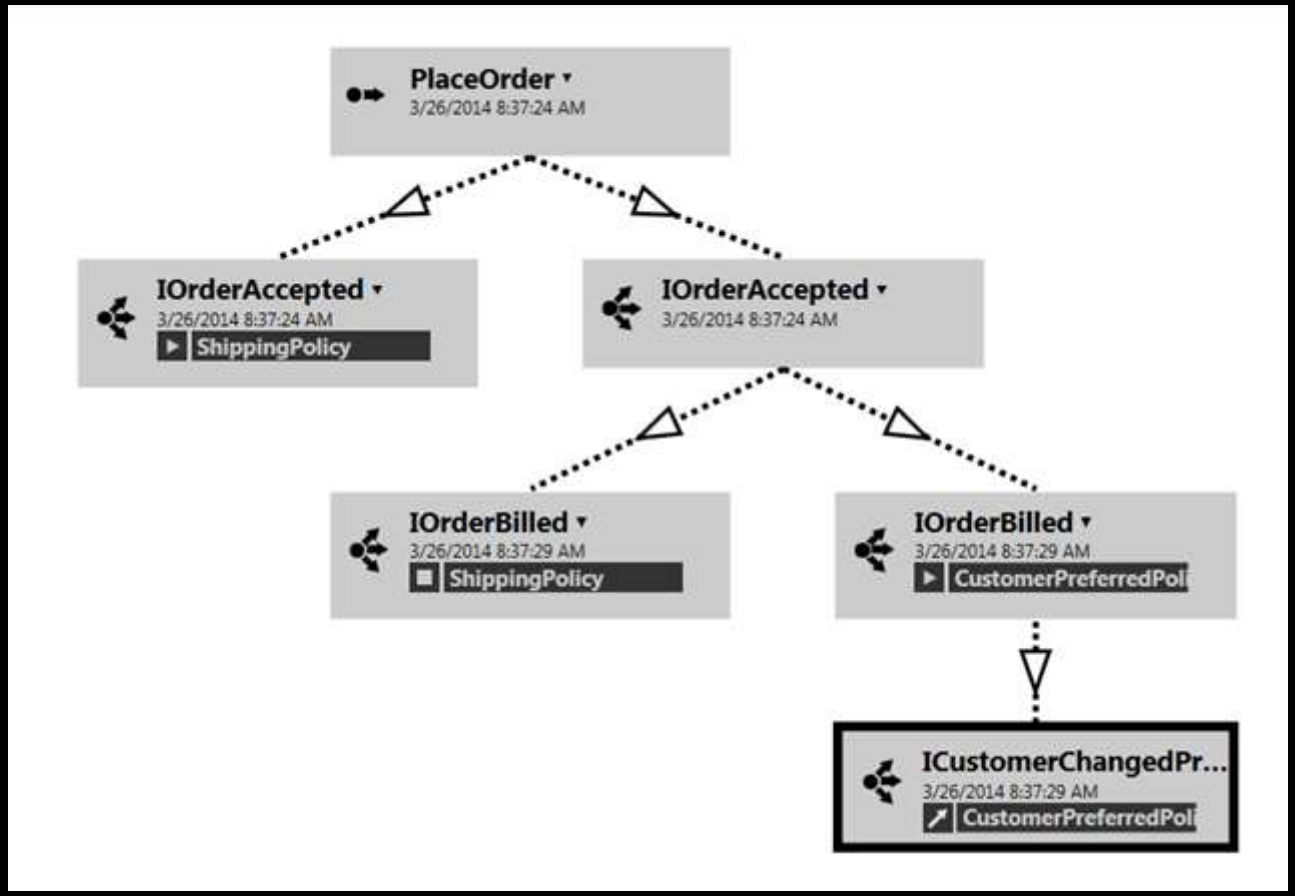
- Sends a copy of the message to another queue when it is processed
  - Supported out-of-the-box by most queues
  - Extract to longer-term storage
    - So the queue doesn't "explode"
- A central log of everything that happened
- Can be difficult to interpret by itself

Leveraging message headers

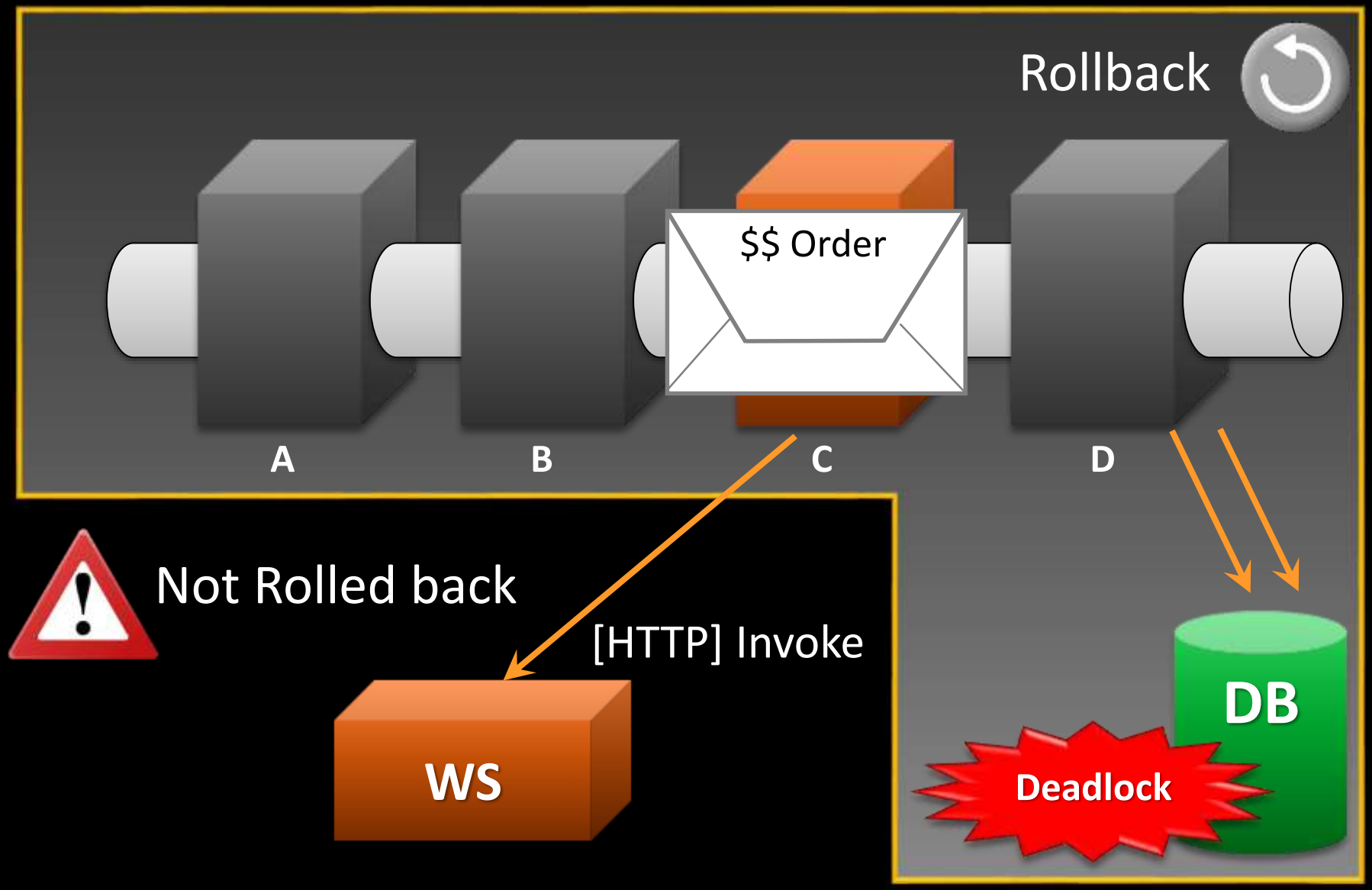


Connect outgoing messages to ID of message being processed

Visualizing the audit store

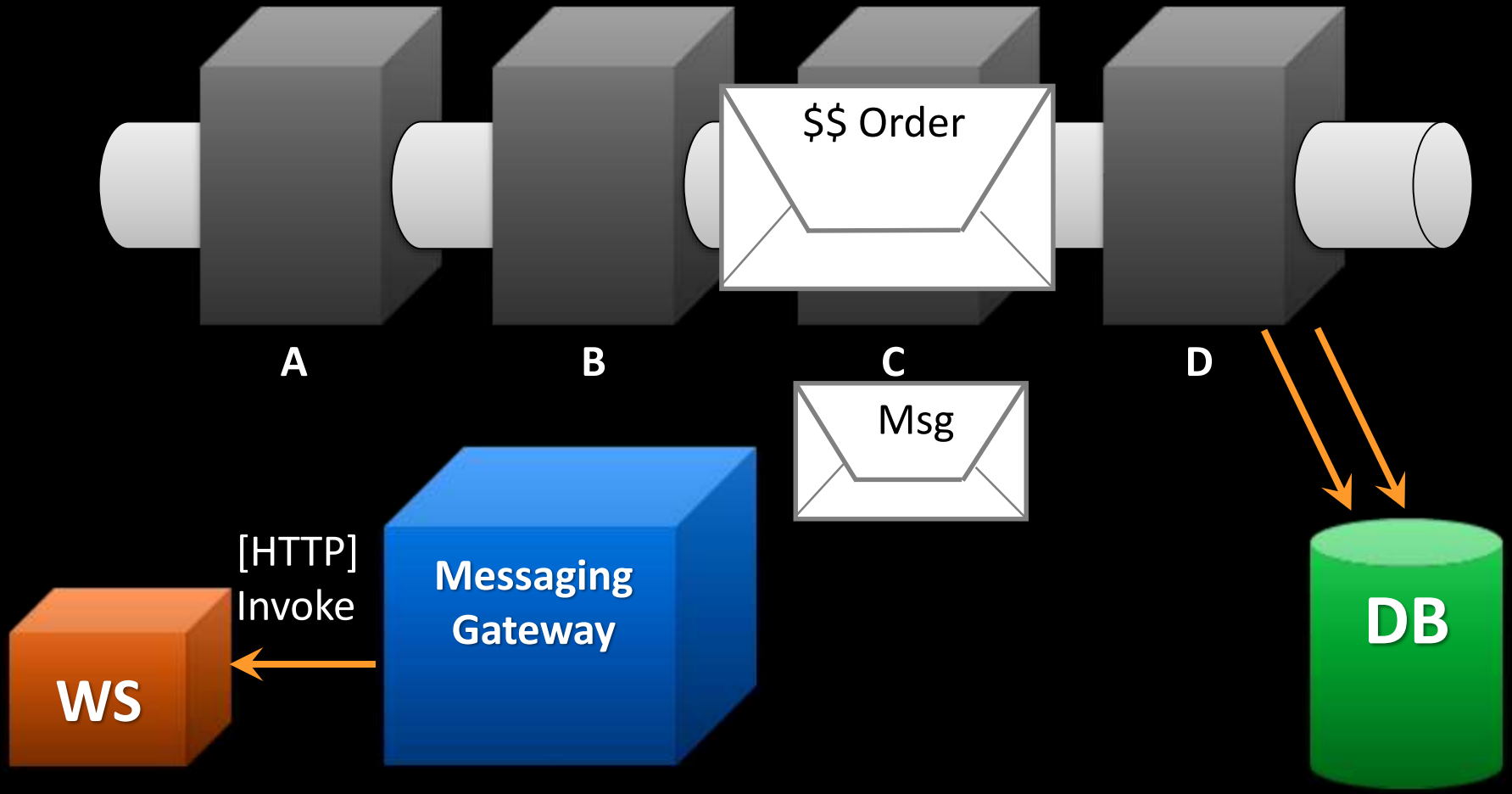


Calling Web Services

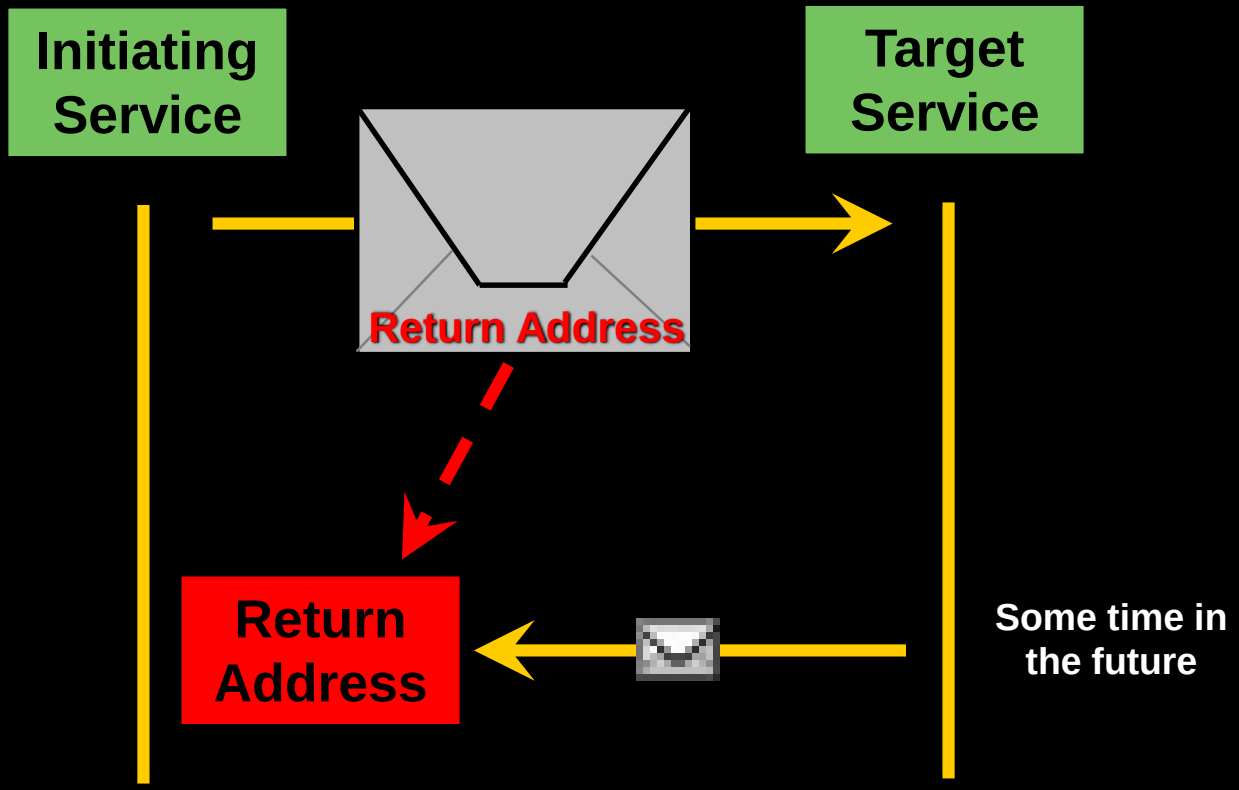


Web Services with Messaging

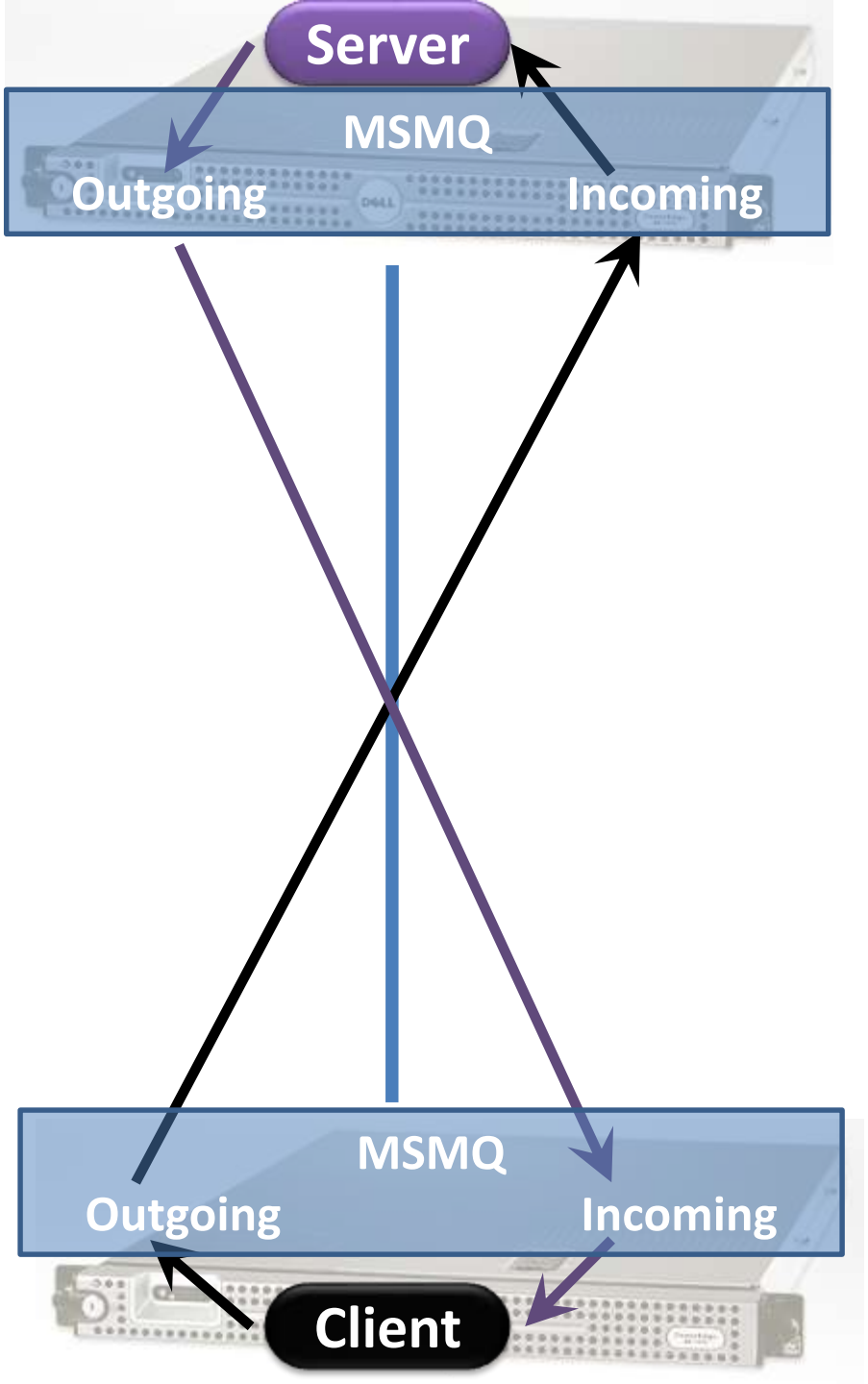
The message won't be sent if there's a failure



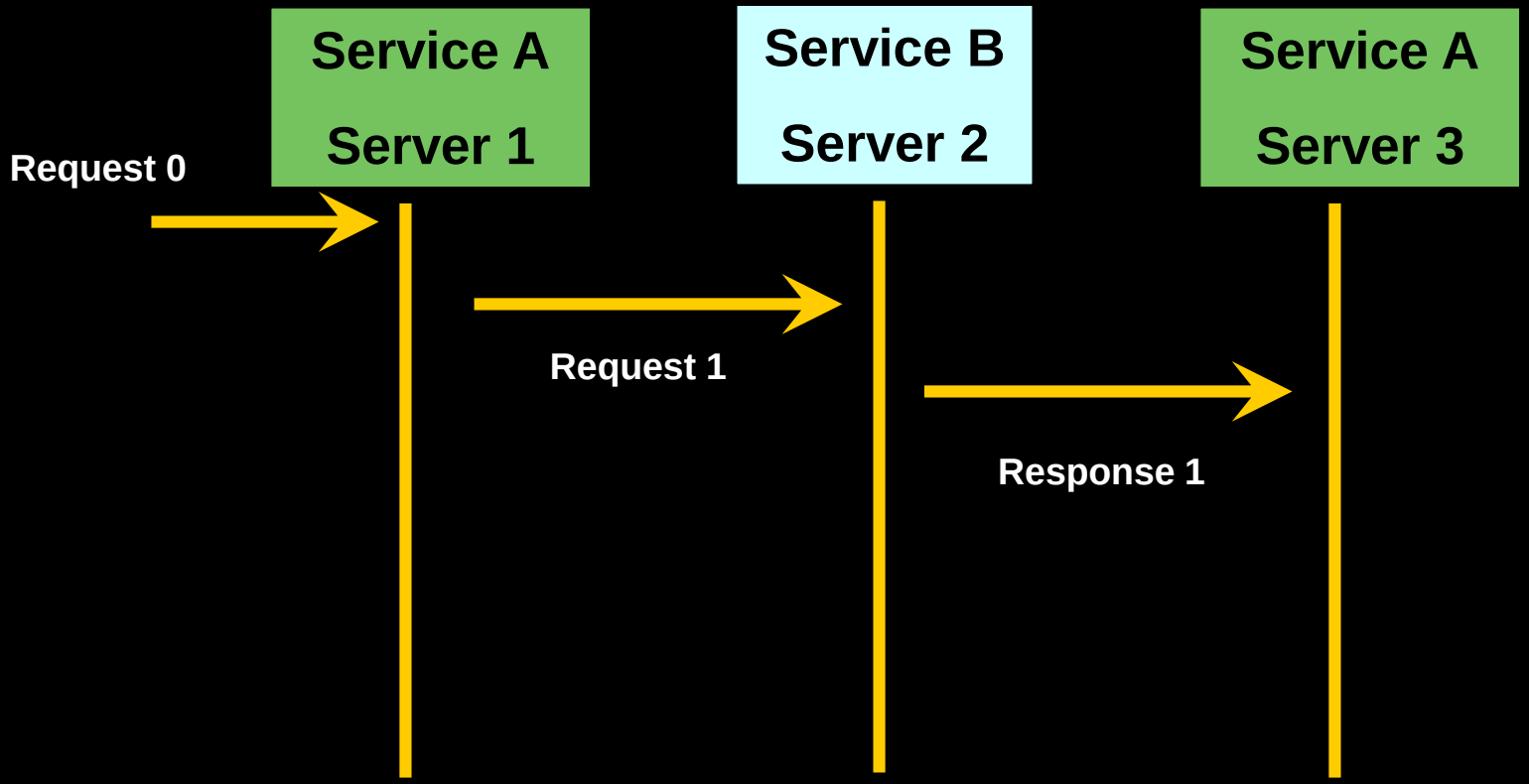
Return Address Pattern



2 Channels: one for requests, one for responses


Server
MSMQ
Outgoing
Incoming
MSMQ
Outgoing
Incoming
Client

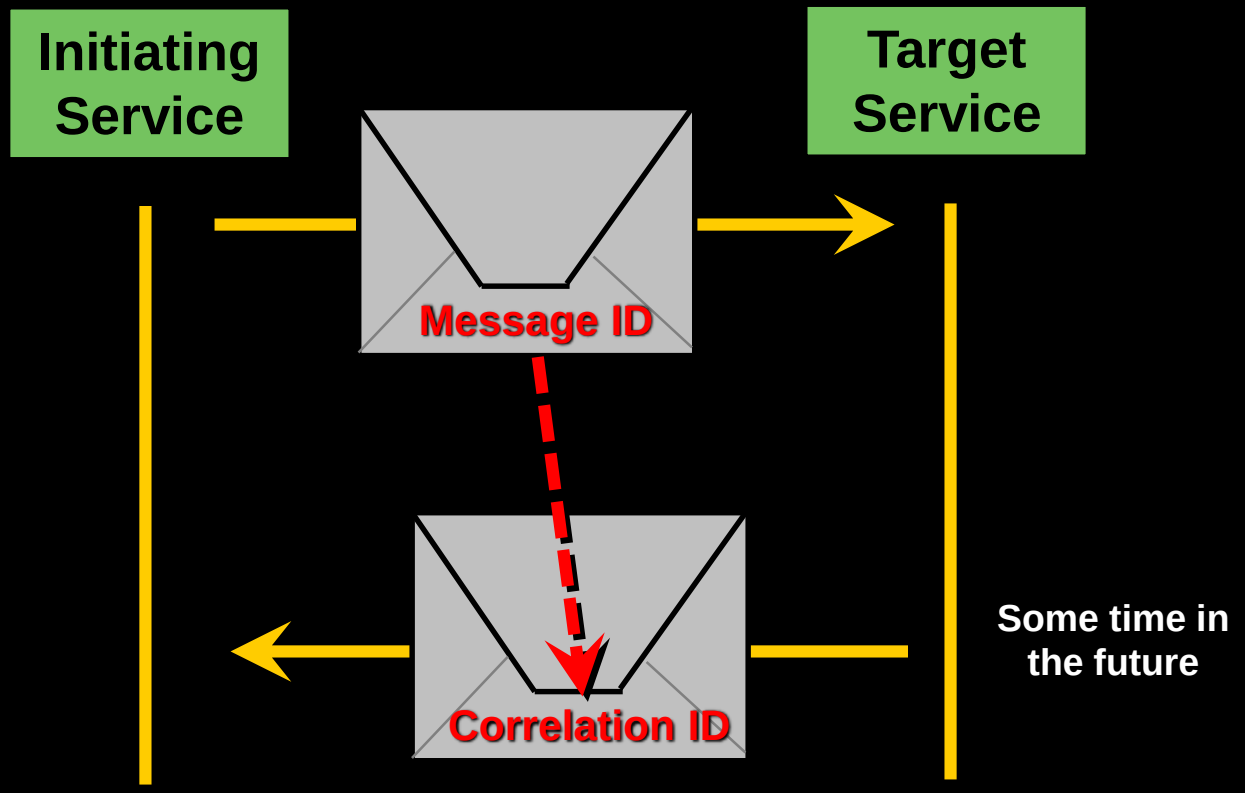
Other uses of Return Address



Enables distributing load between servers in the same service, creating a message handling pipeline

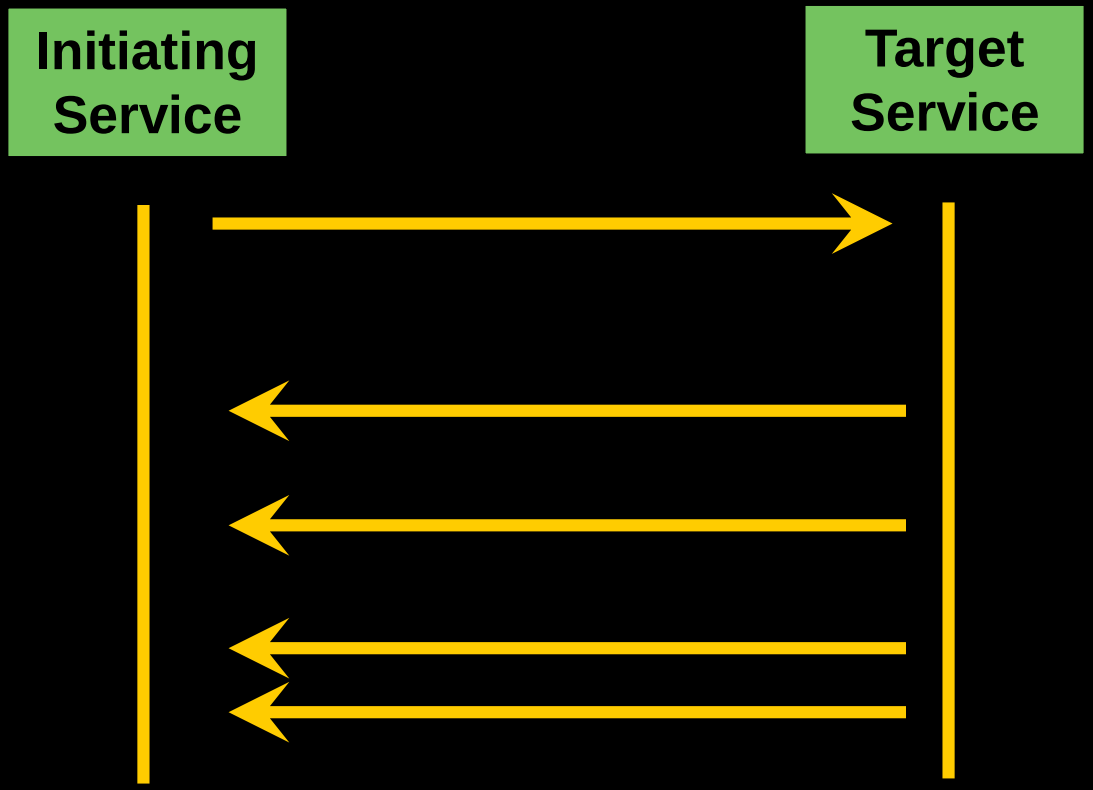
Correlated Request/Response

Based on Return Address



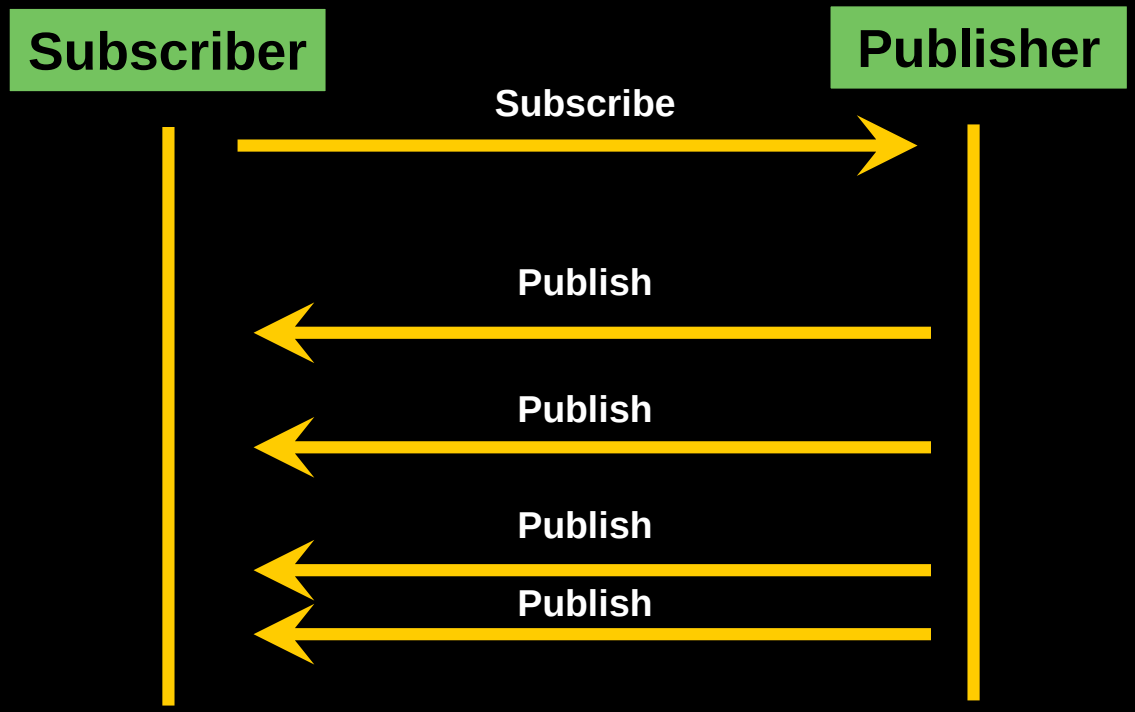
In the header of the response message, there is a correlation id equal to the request message id

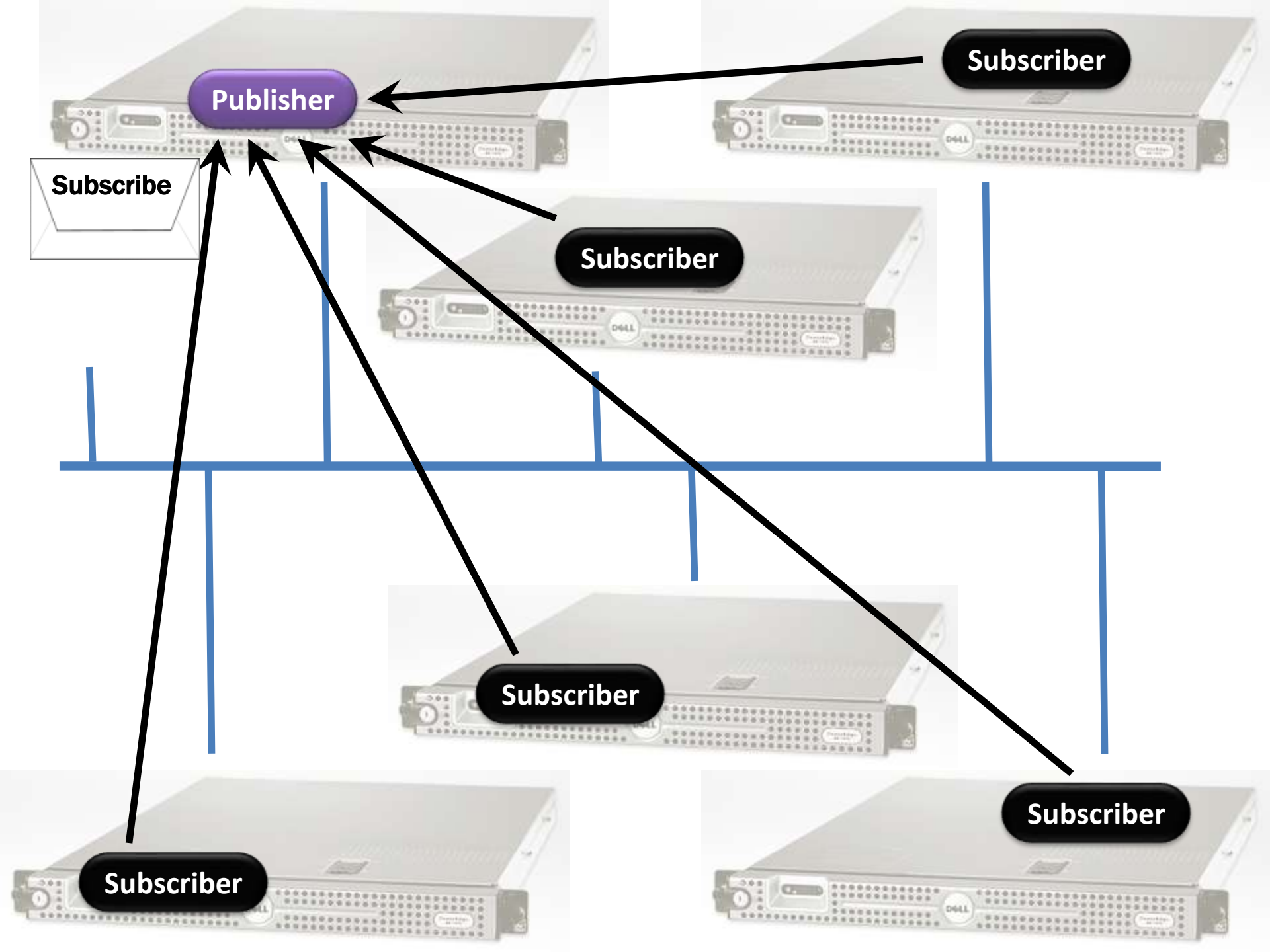
Request / Multi Response

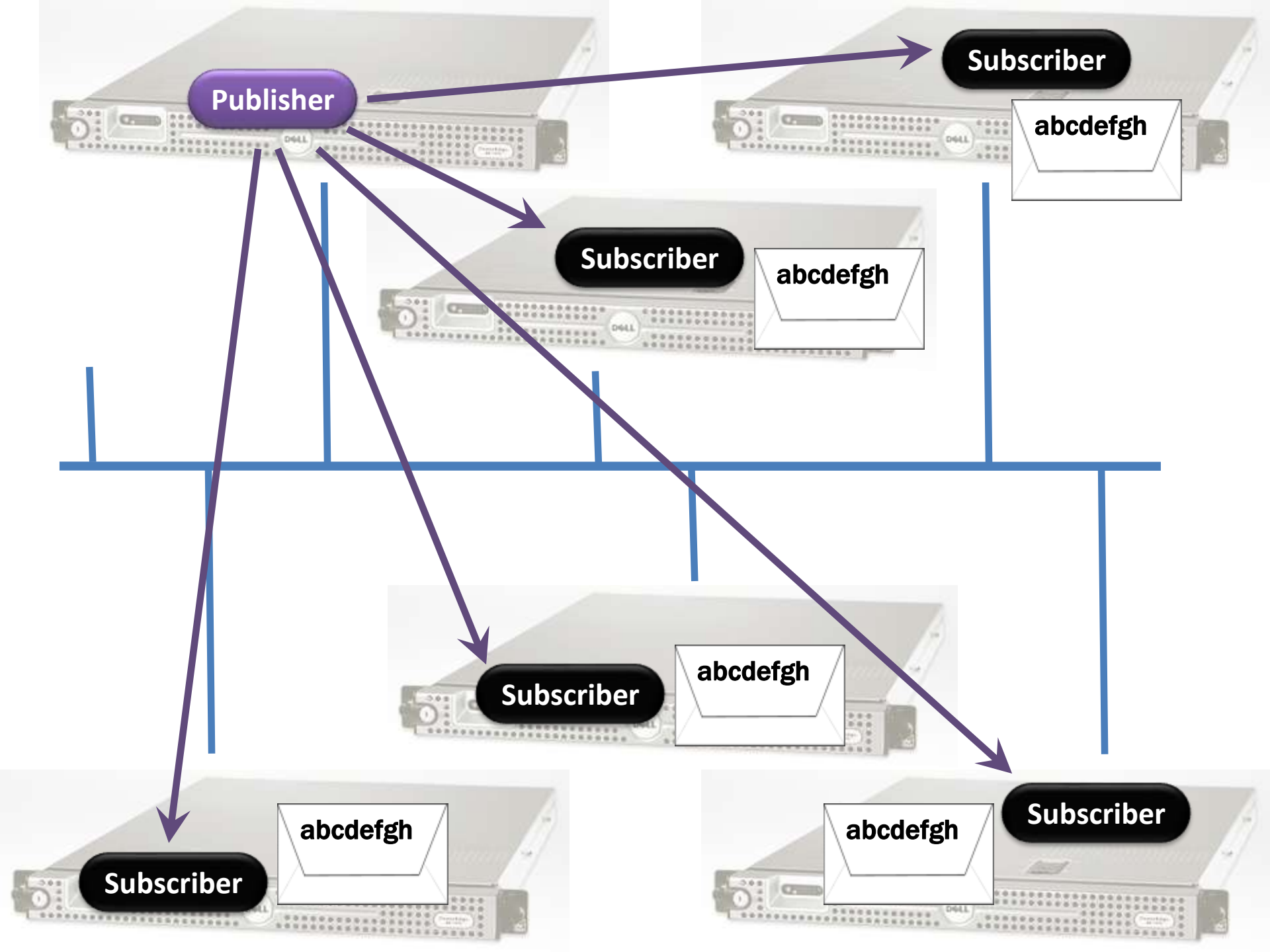


Responses can be of different types

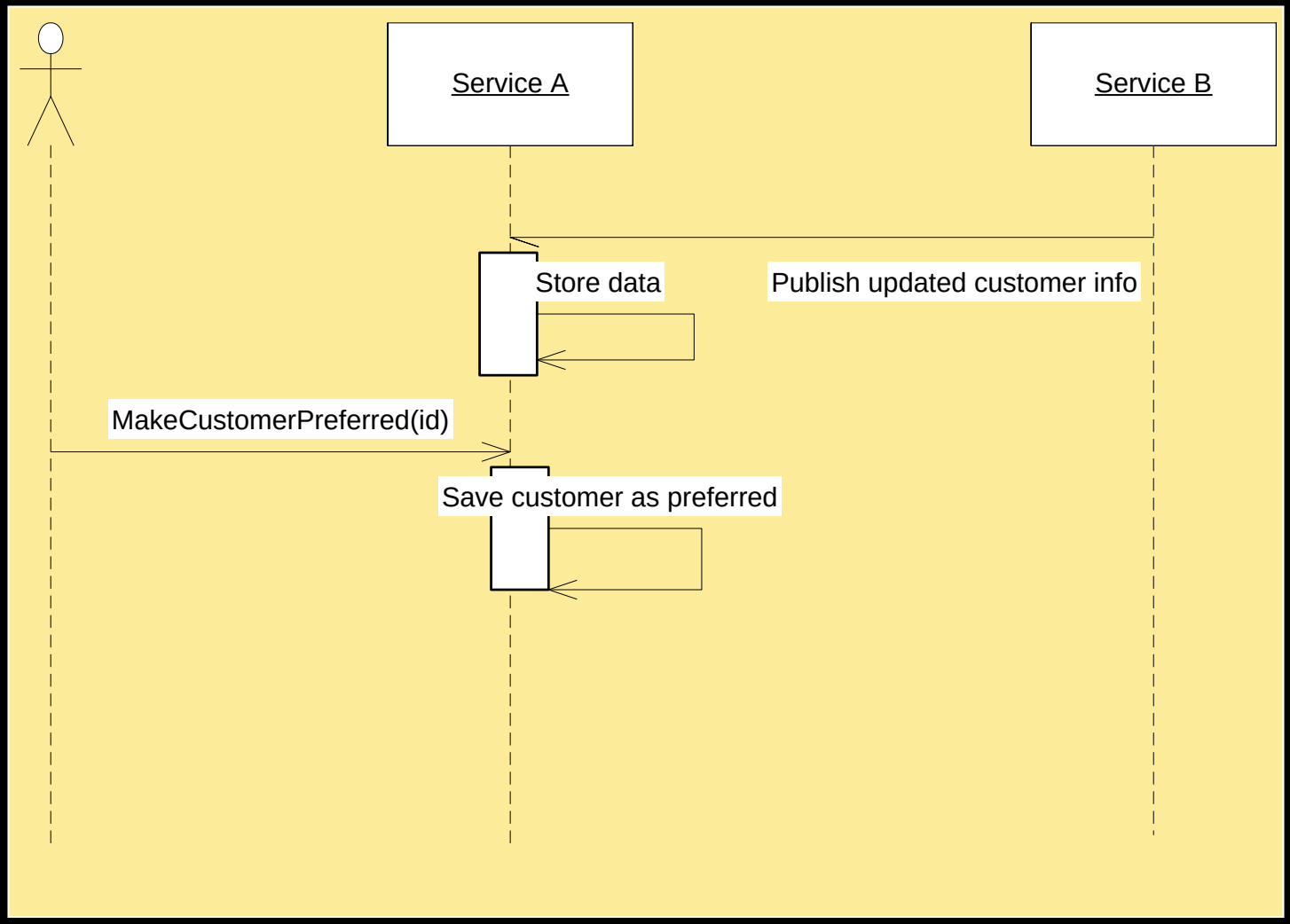
Subscribe / Publish

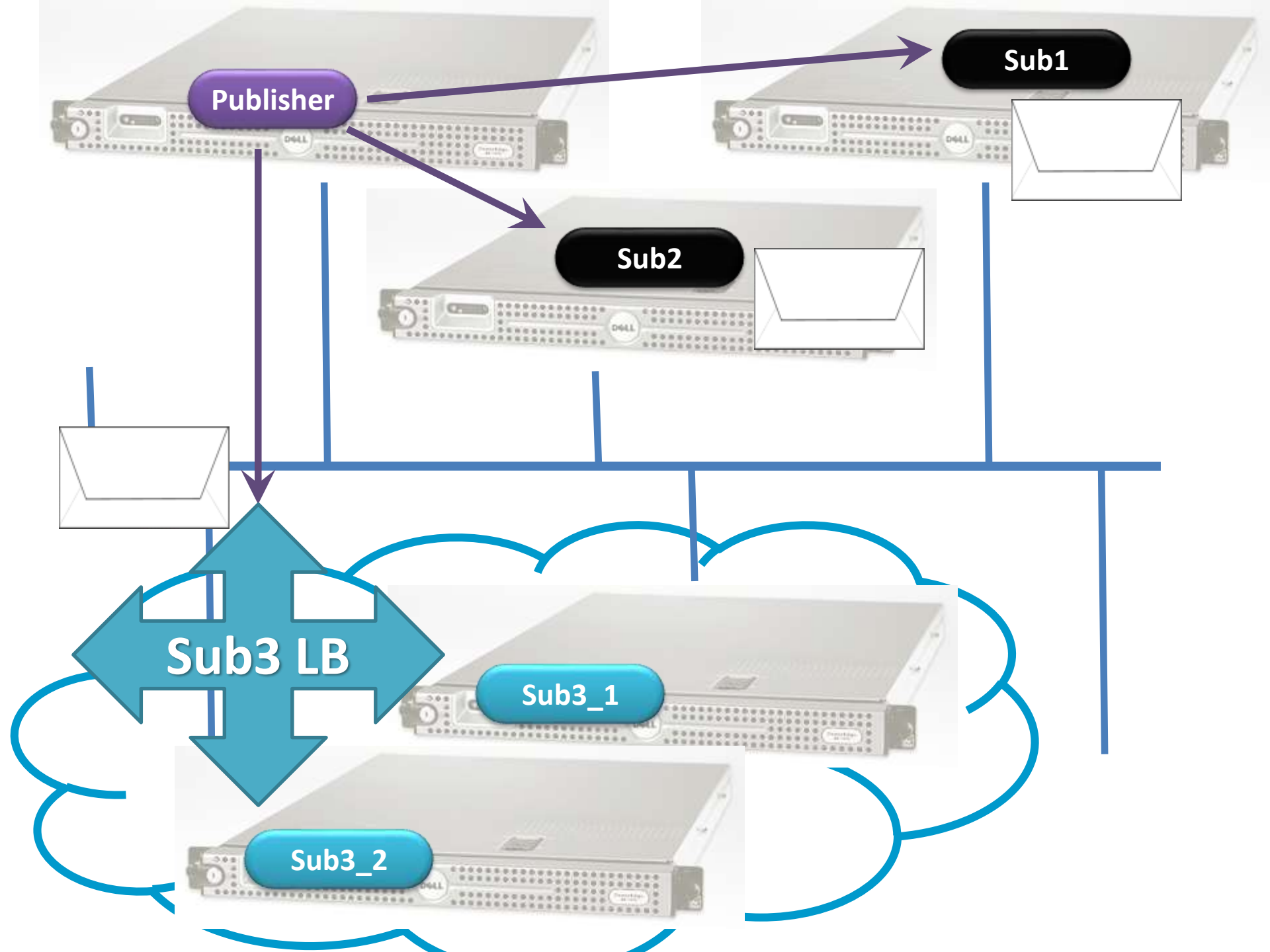



Publisher
Subscribe
Subscriber
Subscriber
Subscriber
Subscriber
Subscriber
Subscriber


Publisher
Subscriber
abcdefgh
Subscriber
abcdefgh
Subscriber
abcdefgh
Subscriber
abcdefgh
Subscriber
abcdefgh
Subscriber

Don’t forget consistency boundaries




Publisher
Sub1
Sub2
Sub3 LB
Sub3_1
Sub3_2

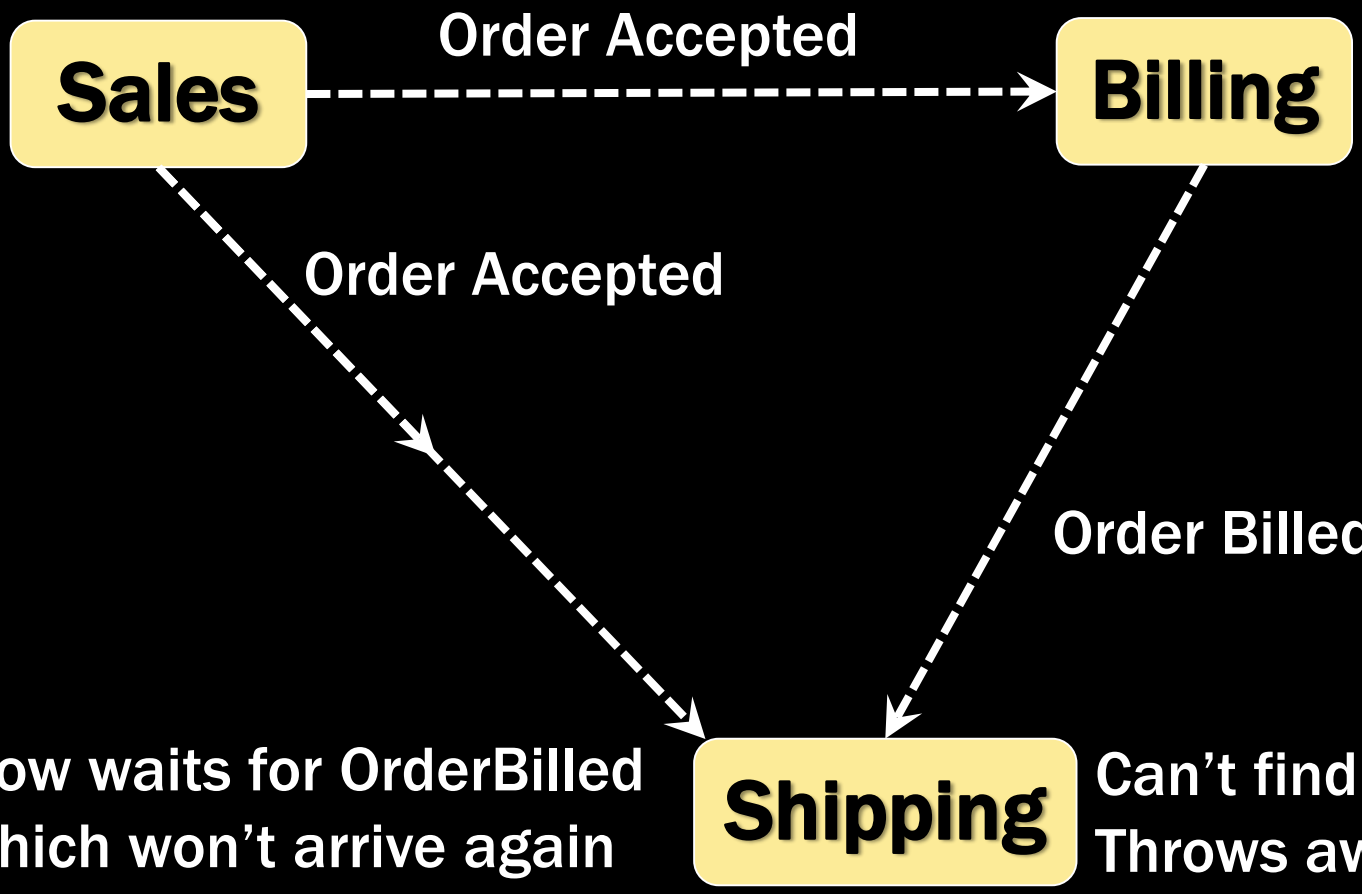
Topic hierarchies & polymorphism

- Subscribe to “Products”, “Products.InStock”, “Products.InStock.PricedToClear”
- Multiple-inheritance even more interesting
  - Publishing an event A which inherits B, C, and D
  - Can subscribe to any or all A, B, C, or D
  - Must use interfaces (not classes)
    - Might not be supported by standard serializers

Events: in-process vs. distributed

- In-memory, synchronous invocation
  - Publisher can know when all subscribers up to date
- Distributed, asynchronous invocation
  - Publisher (and other subscribers) can't know

Out-of-order events



Summary

- Building blocks are simple
  - IMessage
  - IHandleMessages
  - Send, Reply, and Publish
- Identifying boundaries is most important

Easing corporate adoption

- People are afraid of change
- Meet them where they are
- Consider using database tables under a message-driven API
- Diffuses admin/backup/monitoring objections
- Message-driven code is a good first step

Questions?