Building loosely coupled and scalable systems using Event-Driven Architecture

Jonas Bonér
Patrik Nordwall
Andreas Källberg

JAYWAY

Why is EDA Important for Scalability?

What building blocks does EDA consists of?

Outline

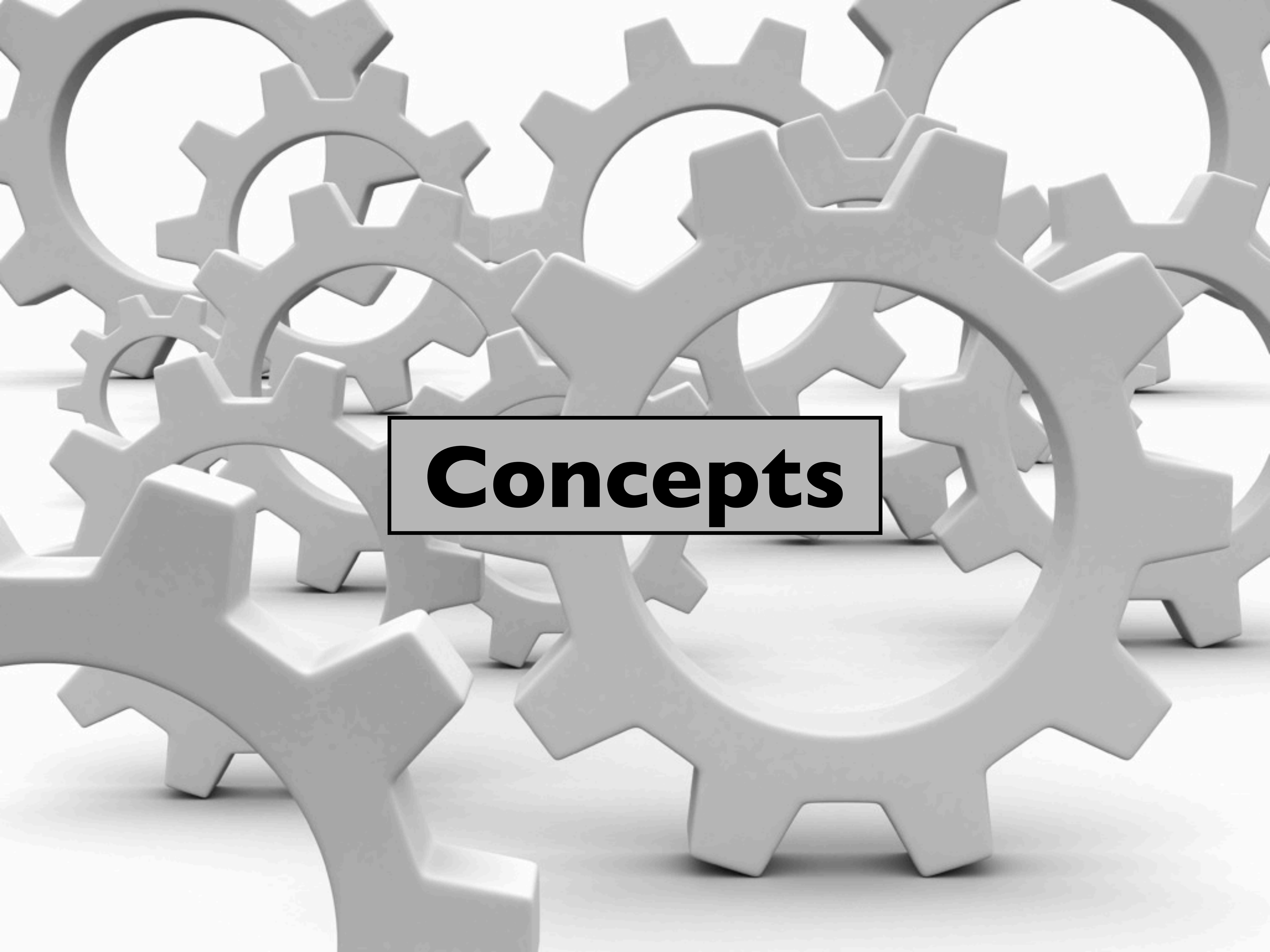
Concepts

Patterns

Challenges

Highly Scalable Web Sites

Concepts



Messaging

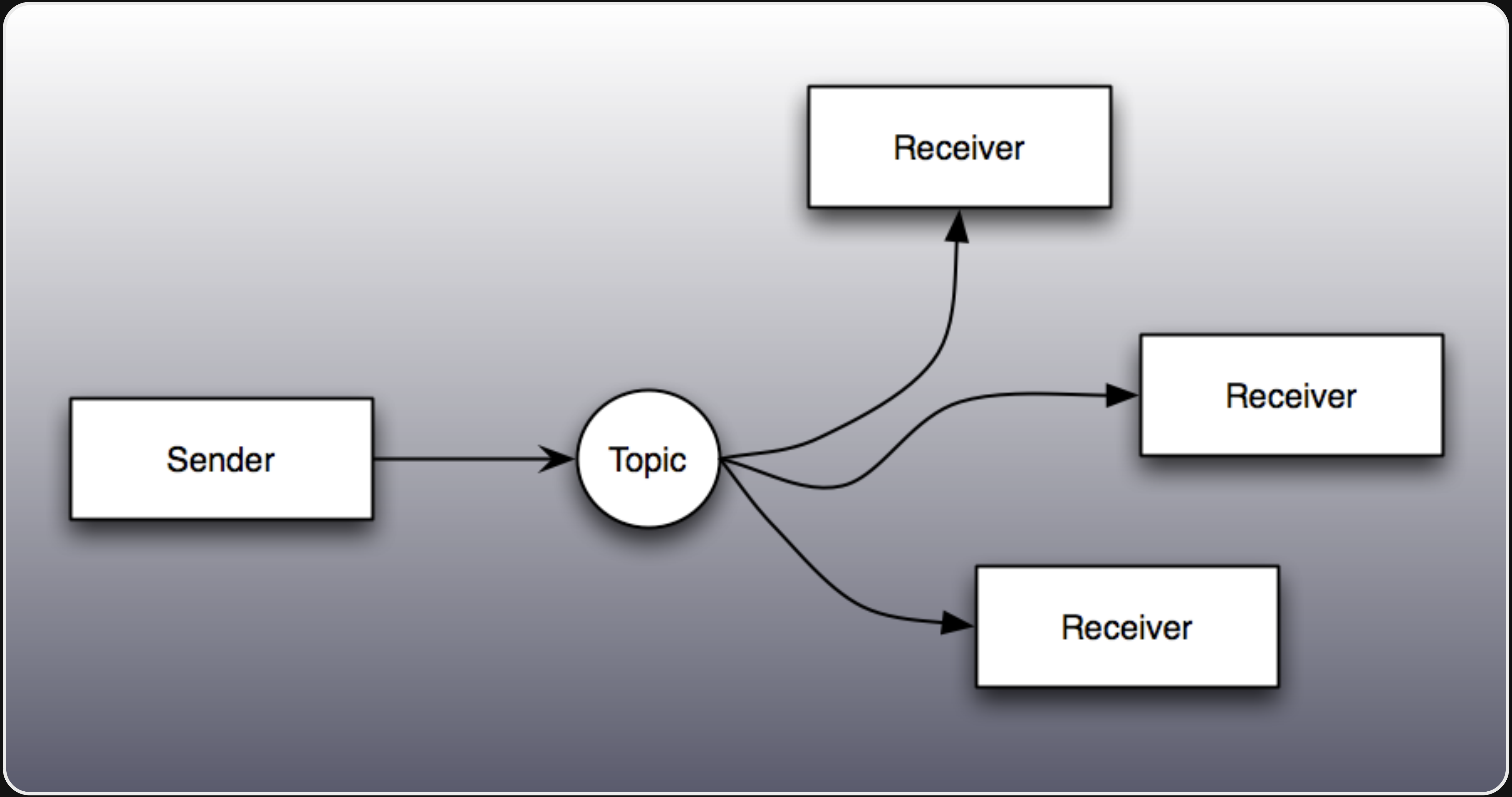
Publish-Subscribe

Point-to-Point

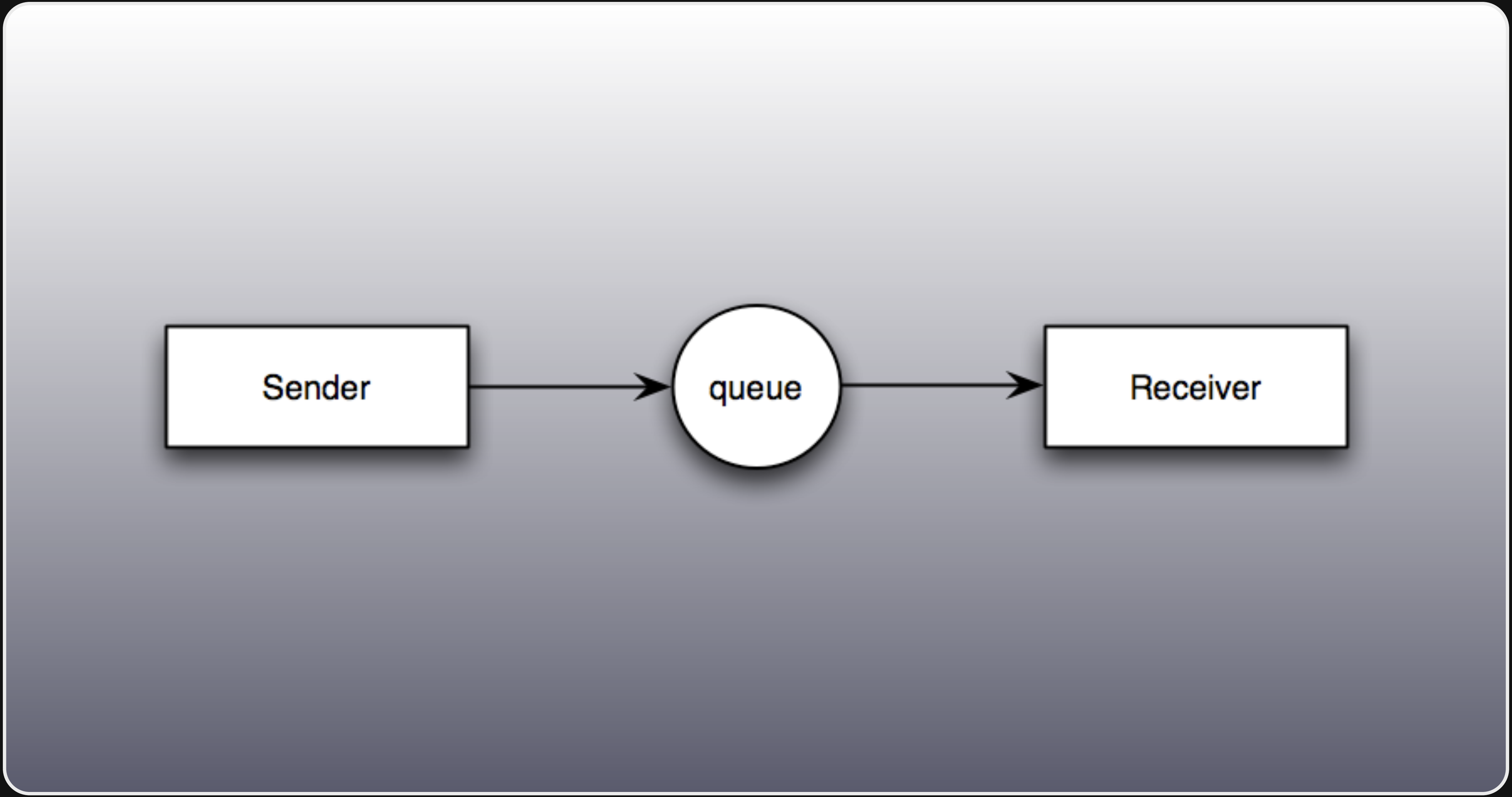
Store-forward

Request-Reply

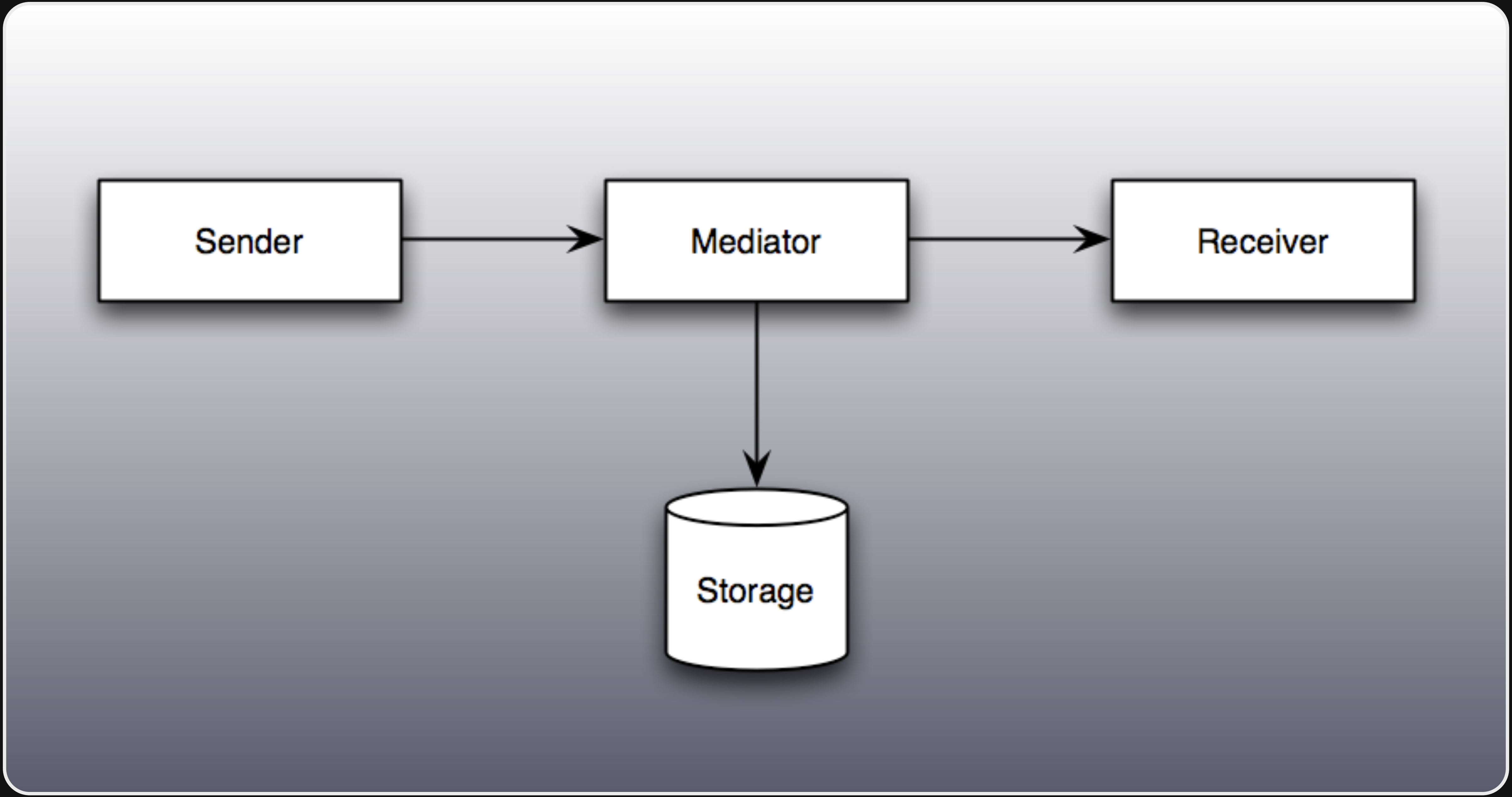
Publish-Subscribe



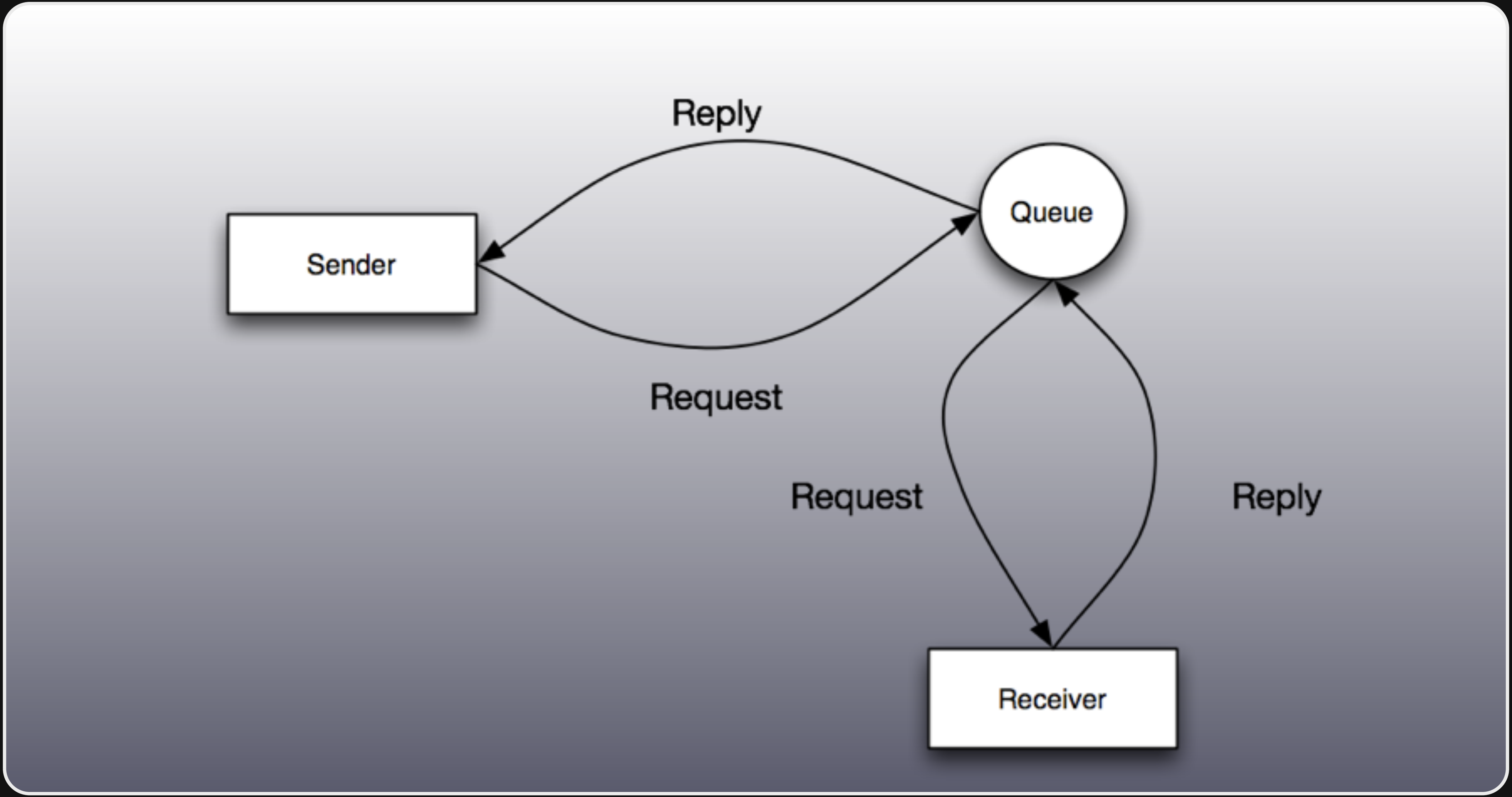
Point-to-Point



Store-Forward



Request-Reply



Standards

AMQP

JMS

Some Products

...

Domain Events

“It's really become clear to me in the last couple of years that we need a new building block and that is the Domain Events”

-- Eric Evans, 2009

Domain Events

"State transitions are an important part of our problem space and should be modeled within our domain."

-- Greg Young, 2008

Domain Events

Something that has happened in the past

CustomerRelocated
CargoShipped
InventoryLossageRecorded

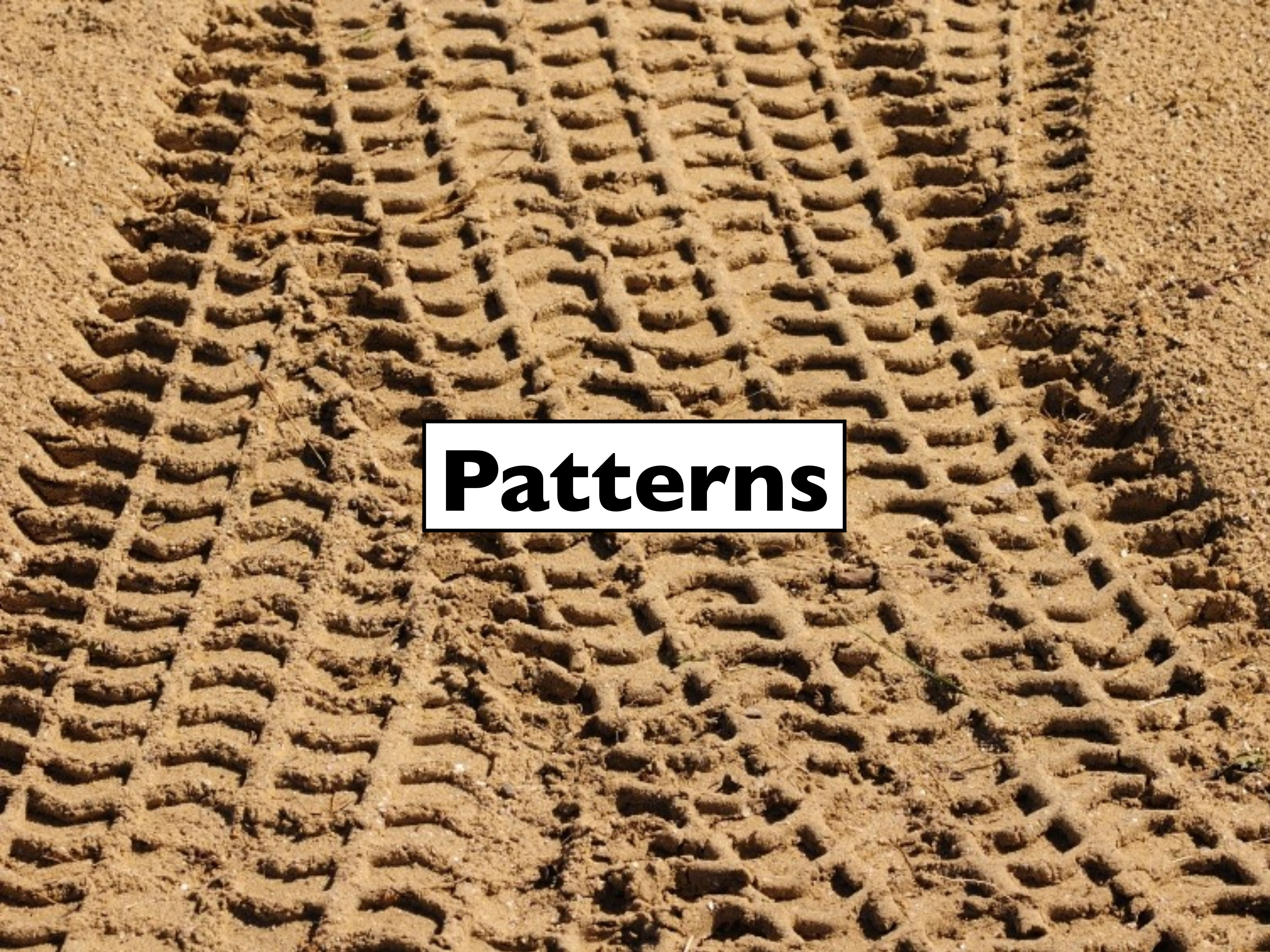
Domain Events

Uniquely identifiable

Self contained

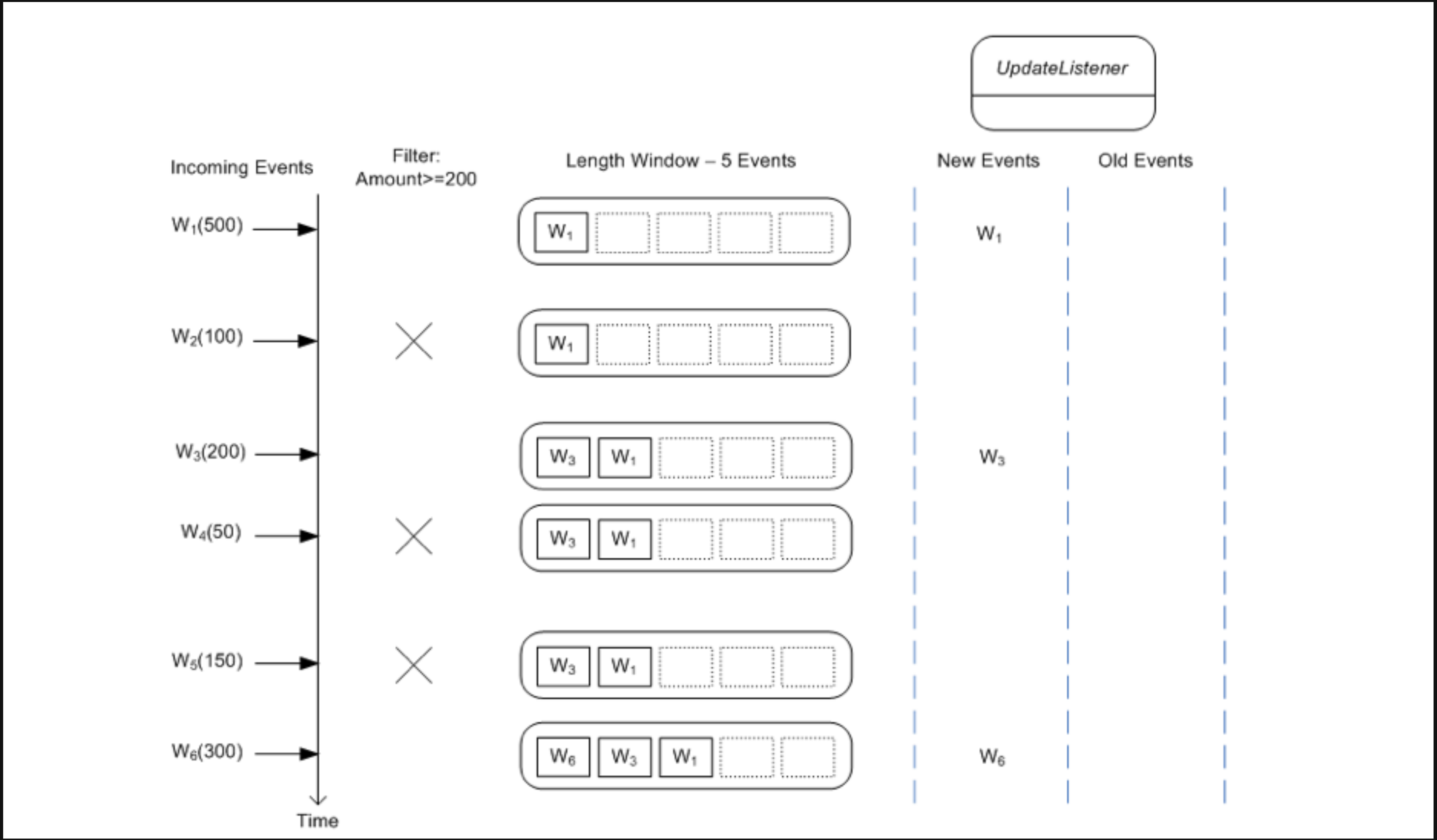
Observable

Time relevant



Patterns

Event Stream Processing



```
select * from Withdrawal
(amount>=200).win:length(5)
```

Actors

- Share NOTHING
- Isolated lightweight processes
- Communicates through messages
- Asynchronous and non-blocking
- No shared state
  … hence, nothing to synchronize.
- Each actor has a mailbox (message queue)

Actors

Easier to reason about

Raised abstraction level

Easier to avoid

Race conditions

Deadlocks

Starvation

Live locks

Actors

Transparent remoting

- Client-managed
- Server-managed

Pub-Sub

- Redis
- ZeroMQ

Guaranteed delivery

Persistent mailbox

- File-based
- Network-based

Command and Query Responsibility Segregation

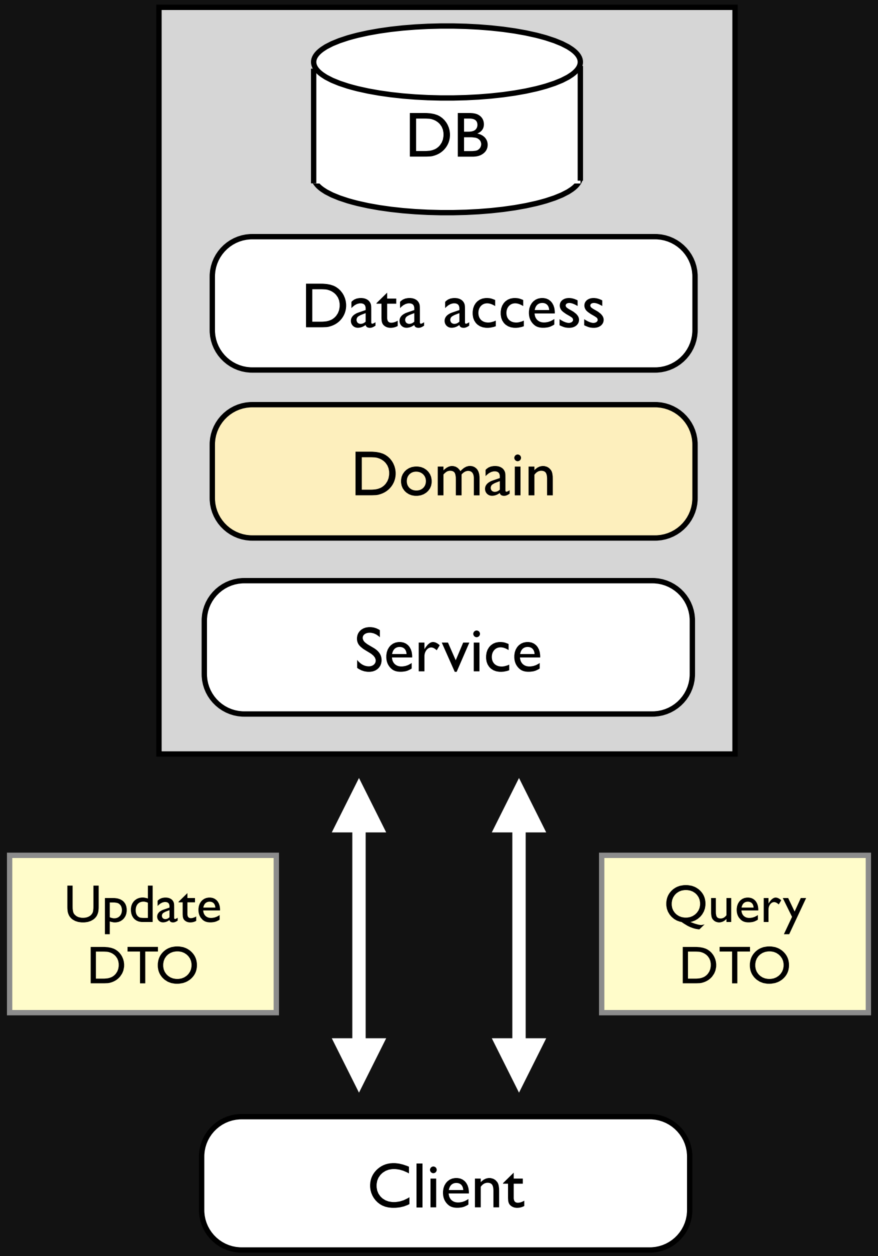
“A single model cannot be appropriate for reporting, searching and transactional behavior.”

-- Greg Young, 2008

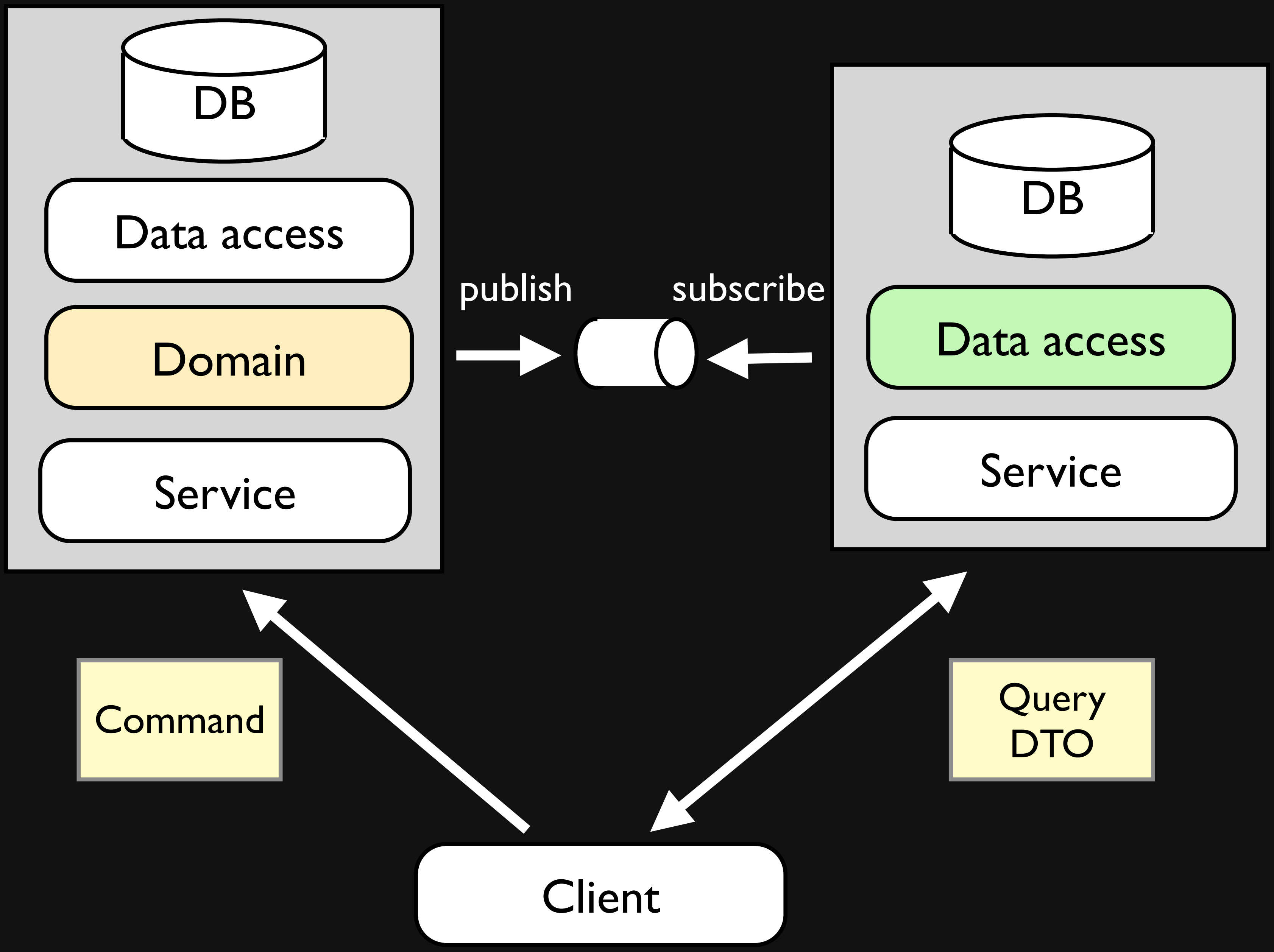
CQRS

- Aggregate roots receive Commands and publish Events
- All state changes are represented by Domain Events
- Reporting module is updated as a result of the published Events
- All Queries go directly to the Reporting, the Domain is not involved

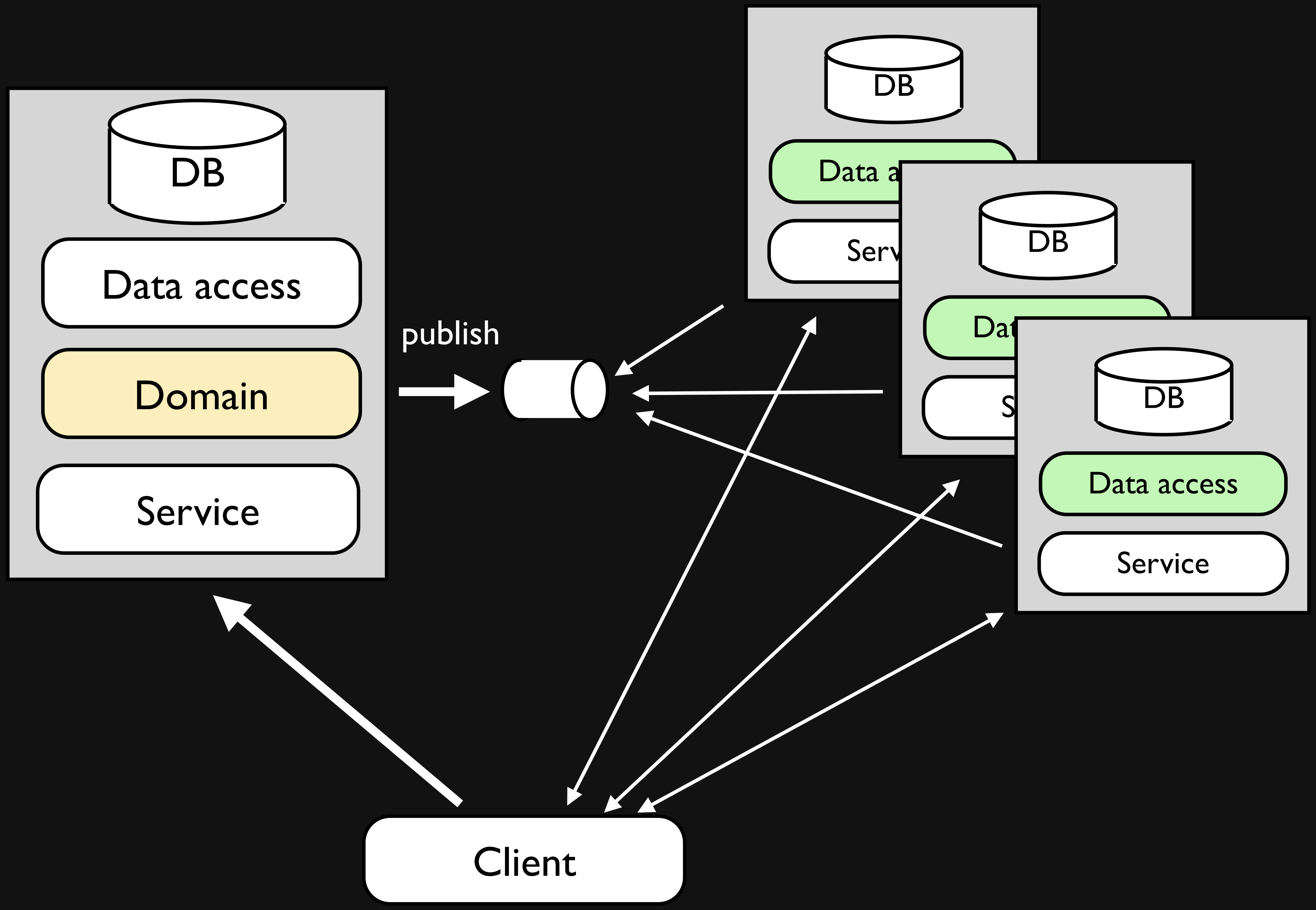
The traditional way...



The CQRS way...


DB
Data access
Domain
Service
publish
subscribe
DB
Data access
Service
Command
Query
DTO
Client

The CQRS way...


DB
Data access
Domain
Service
publish
DB
Data a
Serv
DB
Dat
S
DB
Data access
Service
Client

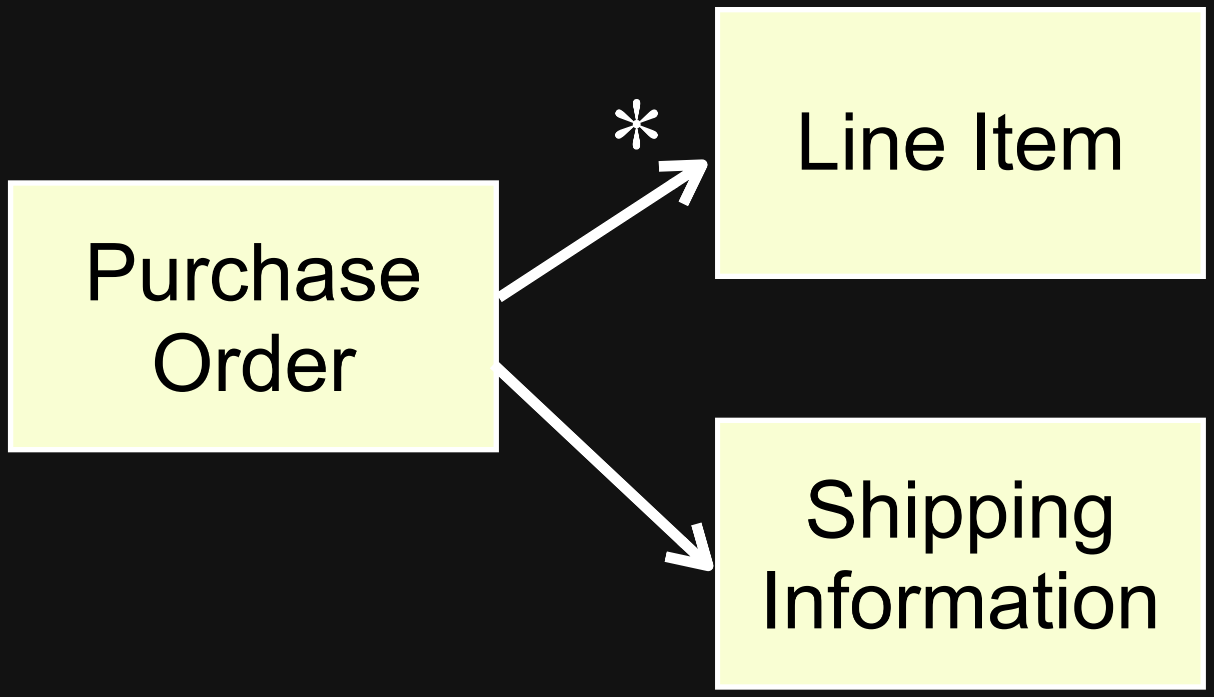
CQRS Benefits

- Separation of concern
- Fully encapsulated domain that only exposes behavior
- Queries do not use the domain model
- Easy integration with external systems
- Performance and scalability
- Testability

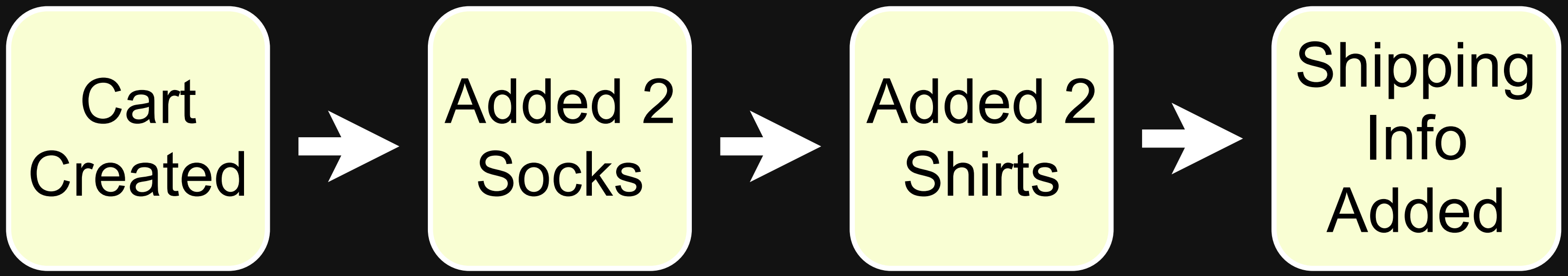
Event Sourcing

- Every state change is materialized in an Event
- All events are stored in an Event Log
- System can be reset and Event Log replayed
- Many different Listeners can be added

Storing Structure



Event Sourcing - Storing Deltas



Cart Created → Added 2 Socks → Added 2 Shirts → Shipping Info Added

Aggregates are tracking events as to what has changed within them

Current state is constructed by replaying all events

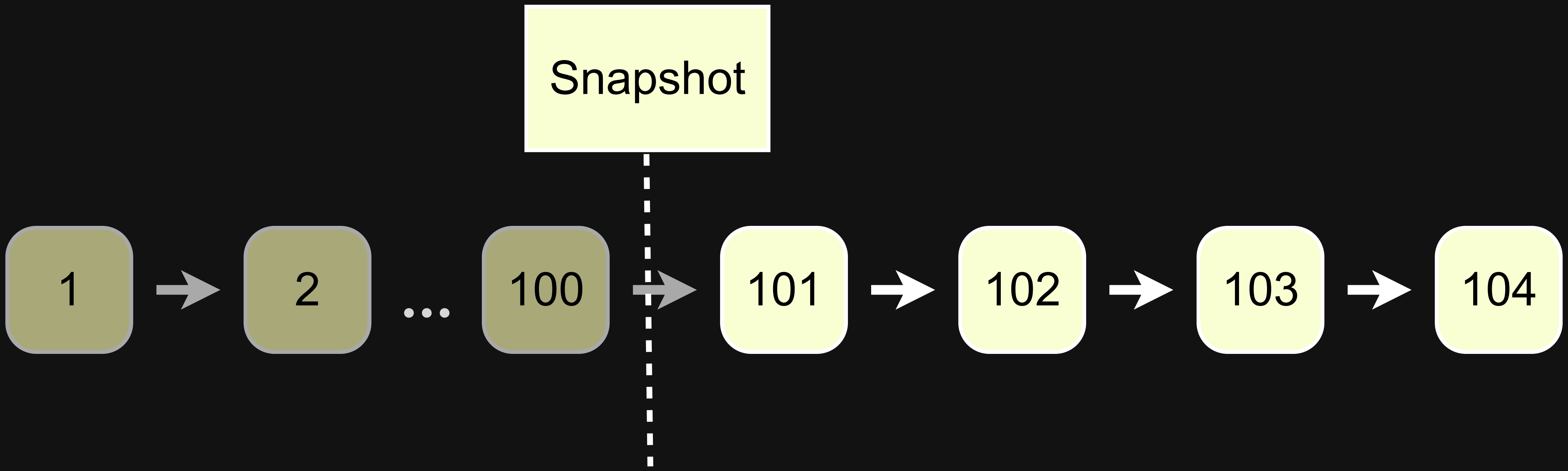
Data is not persisted in a structure but as a series of transactions

No ORM is needed

Event Sourcing - Replaying Events

1 → 2 → 3 → 4 → 5 → 6 → 7

Event Sourcing - Rolling Snapshot



Event Sourcing - Benefits

- No object-relational impedance mismatch
- Bullet-proof auditing and historical tracing
- Support future ways of looking at data
- Performance and scalability
- Testability
- Reconstruct production scenarios

Simple CQRS Sample

http://github.com/patriknw/sculptor-simplecqrs/

Challenges

Clustering of Brokers

ActiveMQ

- Master-Slave
- Store and Forward Network of Brokers

RabbitMQ

- Cluster of Erlang nodes

ZeroMQ

- Brokerless - point-to-point or pub-sub

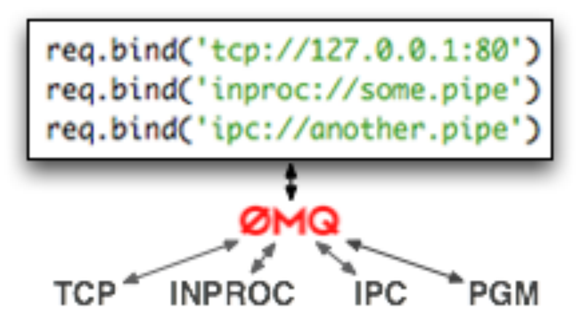
ZeroMQ

Network protocol - thin layer above TCP/IP

Transports

- INPROC
- IPC
- MULTICAST
- TCP

Text

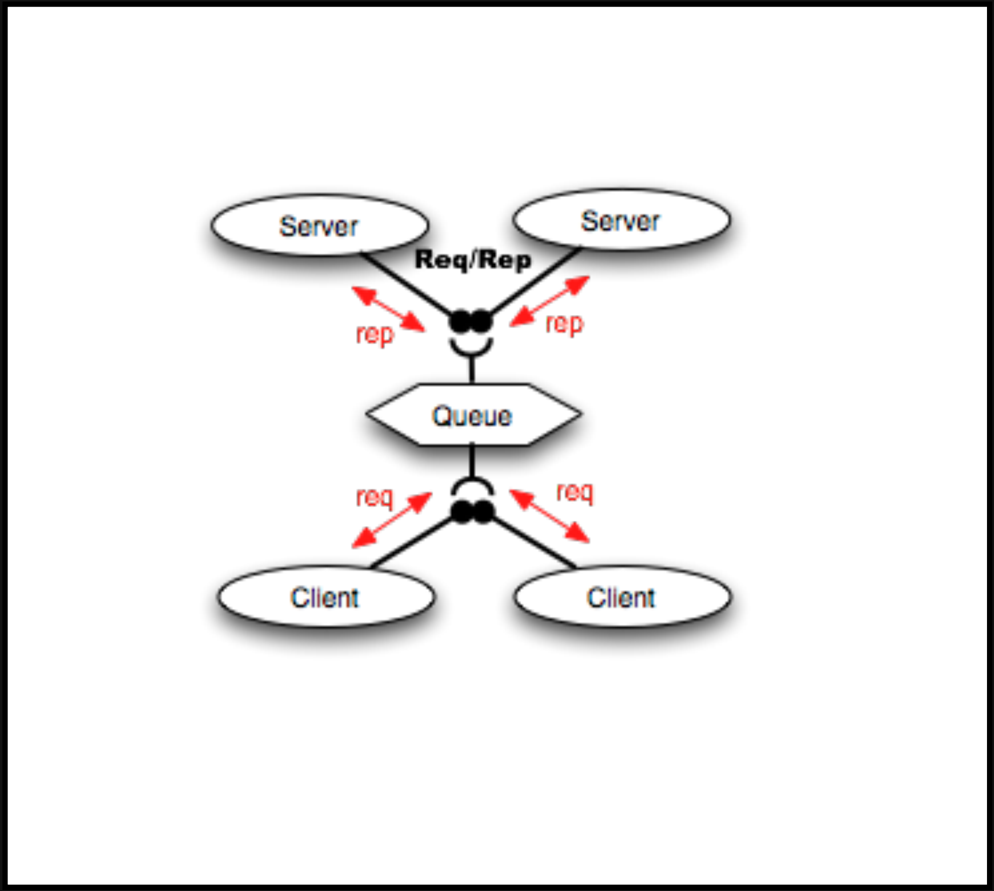


```
req.bind('tcp://127.0.0.1:80')
req.bind('inproc://some.pipe')
req.bind('ipc://another.pipe')
```

ZeroMQ

Forwarding devices

- QUEUE
- FORWARDER
- STREAMER



ZeroMQ

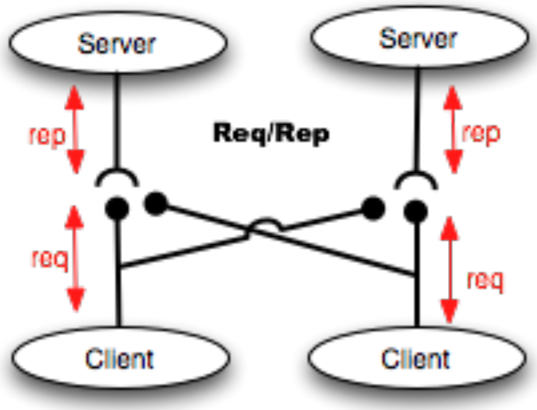
Patterns

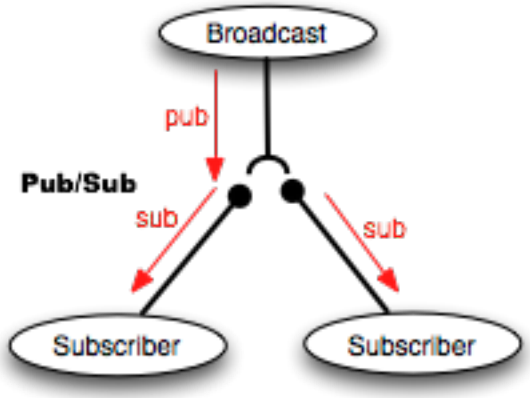
REQUEST/REPLY (load-balanced)

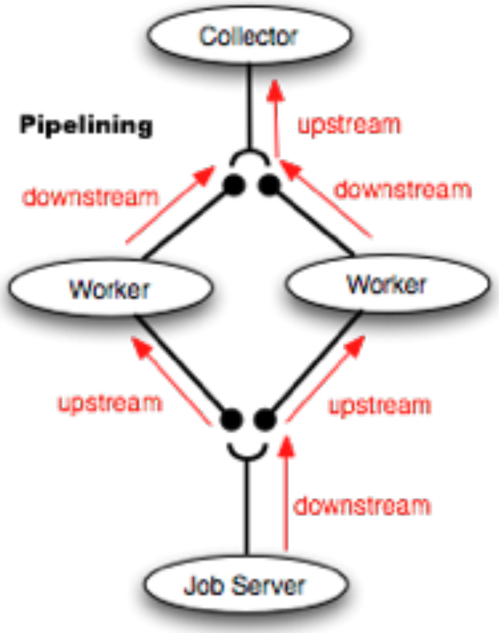
PUB/SUB

UPSTREAM/DOWNSTREAM (pipelining)

PAIR (exclusive)







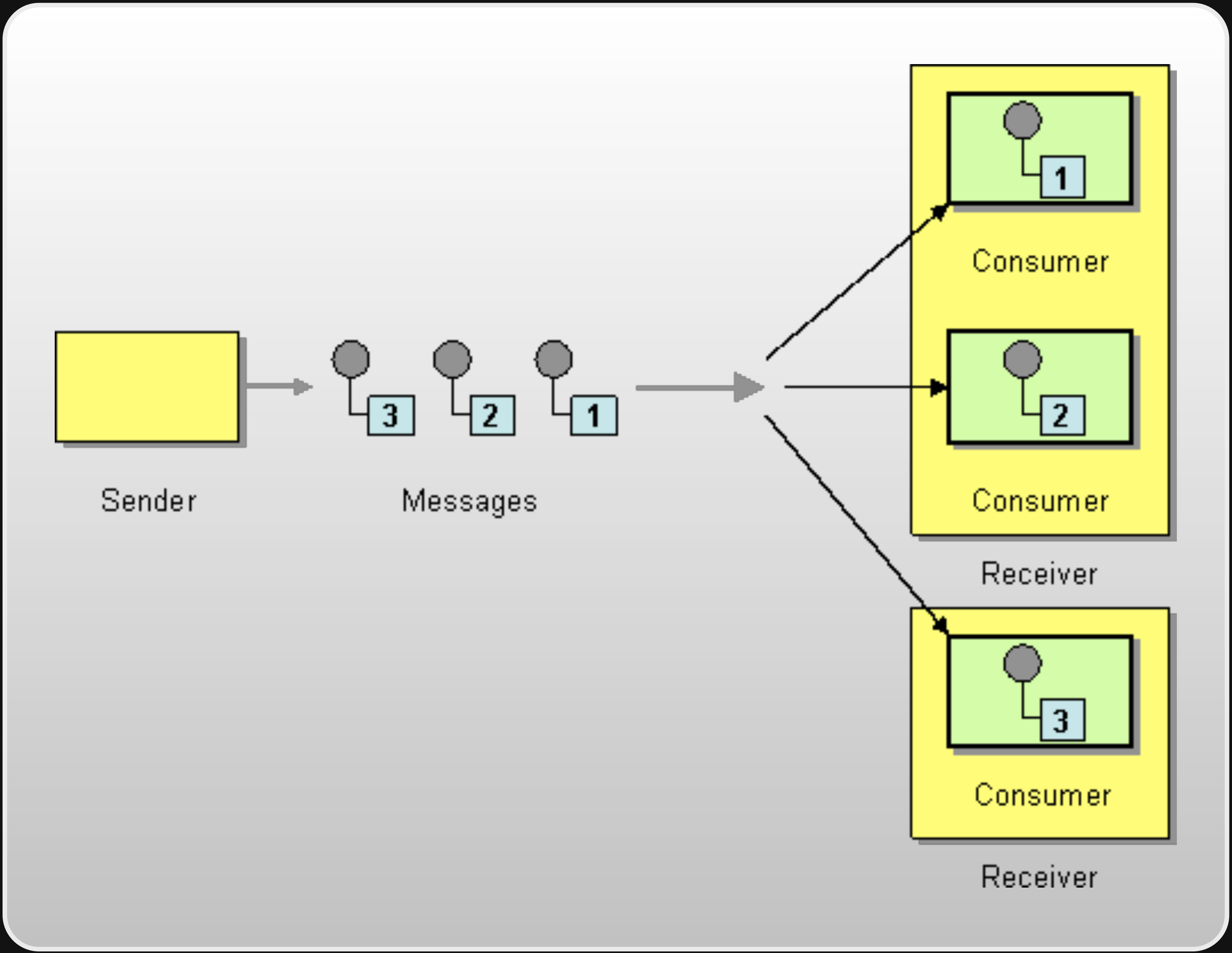
Wire Formats

Java serialization (binary, schema, runtime)
Protobuf (binary, schema, compiled)
Avro (binary, schema, compiled & runtime)
Thrift (binary, schema, compiled)
Text
MsgPack (binary, schema, compiled)
Protostuff (binary, schema, compiled)
Kryo (binary, schema-less, runtime)
BERT (binary, schema-less, runtime)
Hessian (binary, schema-less, compiled)
XML (text, schema)
JSON (text, schema-less)

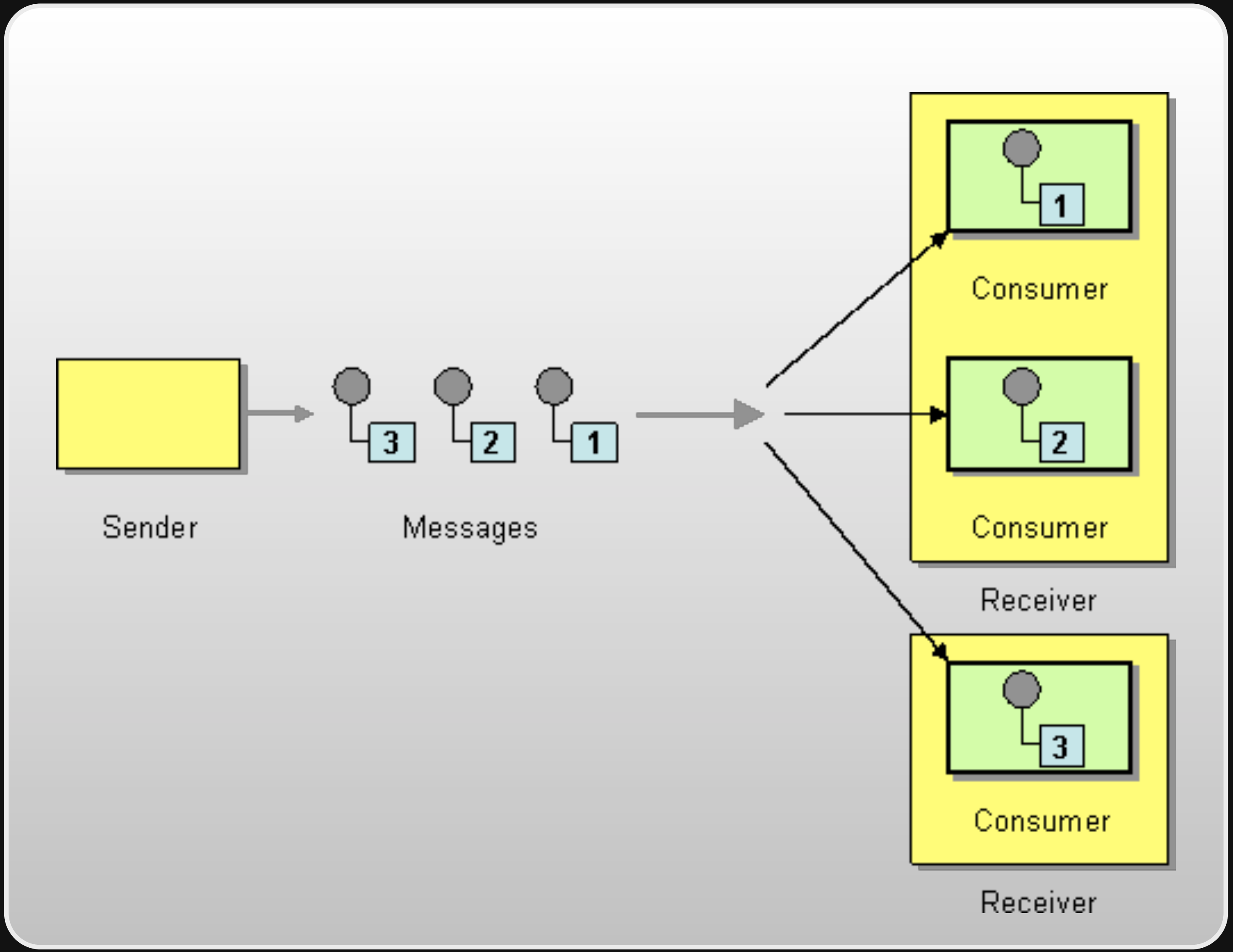
Guaranteed Delivery

Do I really need it?

Persistence increases reliability at the expense of performance



Competing Consumers



Pattern for solving:

- Load balancing
- Concurrency
- Failover

Only works with
Point-to-Point Channel

Challenge

- ordering
- duplicates (idempotent receiver)

Duplicate Messages

What do I need?

- Once-and-only-once
- At-least-once
- At-most-once

QOS

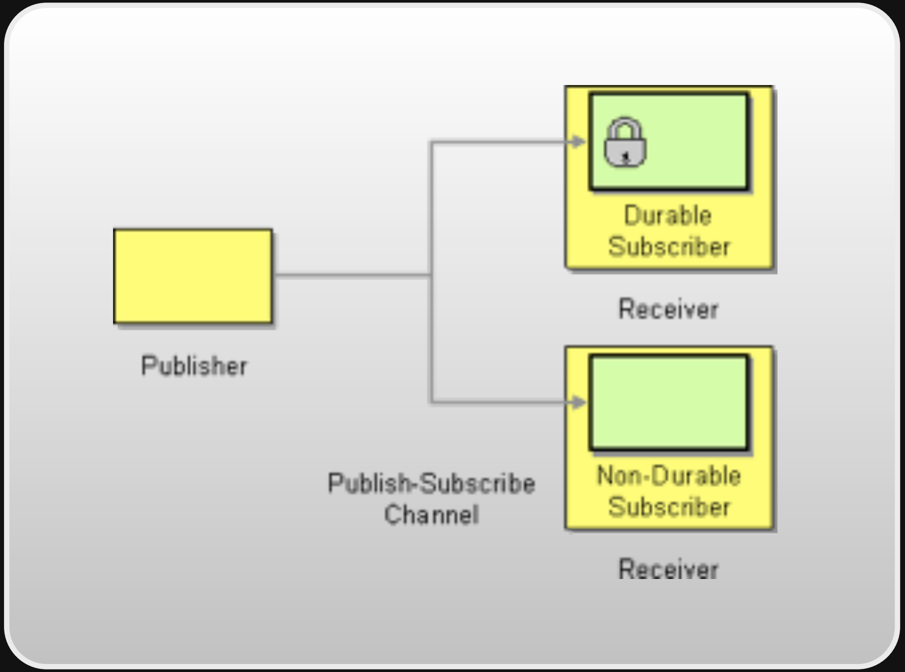
keep history of processed ids

Unique message identifier

Business semantics

How to get back on track?

Point-to-point: no problem, just make the queue persistent



Pub/sub: well, not so straight forward

Problem: only active subscribers

Solution: durable subscriber

Problem: failover and load balancing

Producer Flow Control

What to do when producers flood you with messages?

Running low on broker resources, slow consumers

Graceful degradation

- caller run (in process only)
- block
- abort
- discard

Behind the Scenes of Highly Scalable Web Sites

caching is important, but also...

Minimize latency

Flickr: Do The Essential Work Up-Front And Queue The Rest

Amazon: ~99% of content is static

Reddit: Precompute everything and cache it

Changes - pull or push

Facebook: Pull on Demand

Digg: Push on Change

Twitter: Push tweets

Truly event-driven web clients

Request-response doesn't fit collaborative systems

WebSockets enable real event-driven web

Why is EDA Important for Scalability?

- Scale out and up
- Load balance
- Parallel execution
- Non-blocking
- Loosely coupled components can scale more independent of each other

?

thanks

for listening