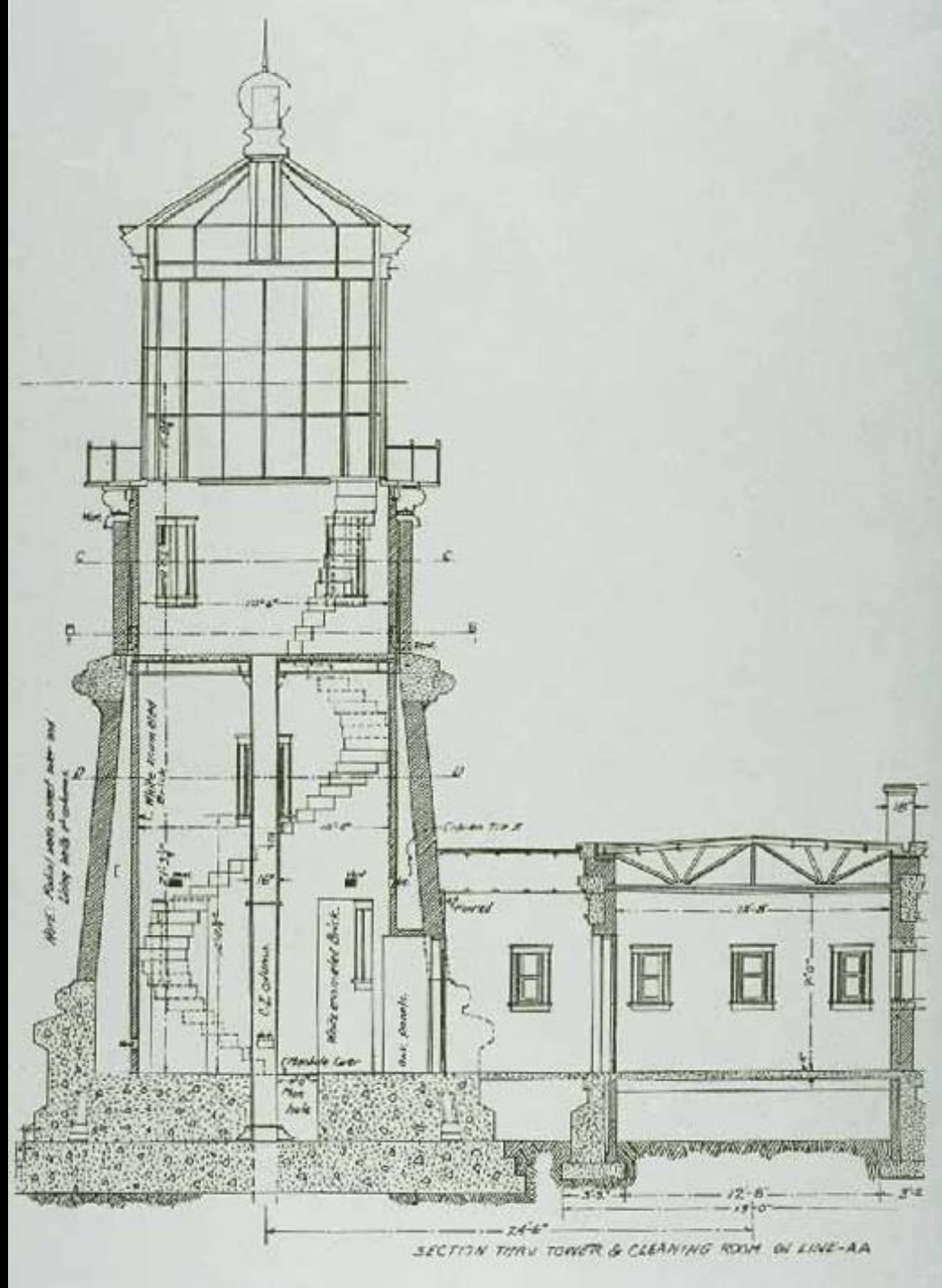
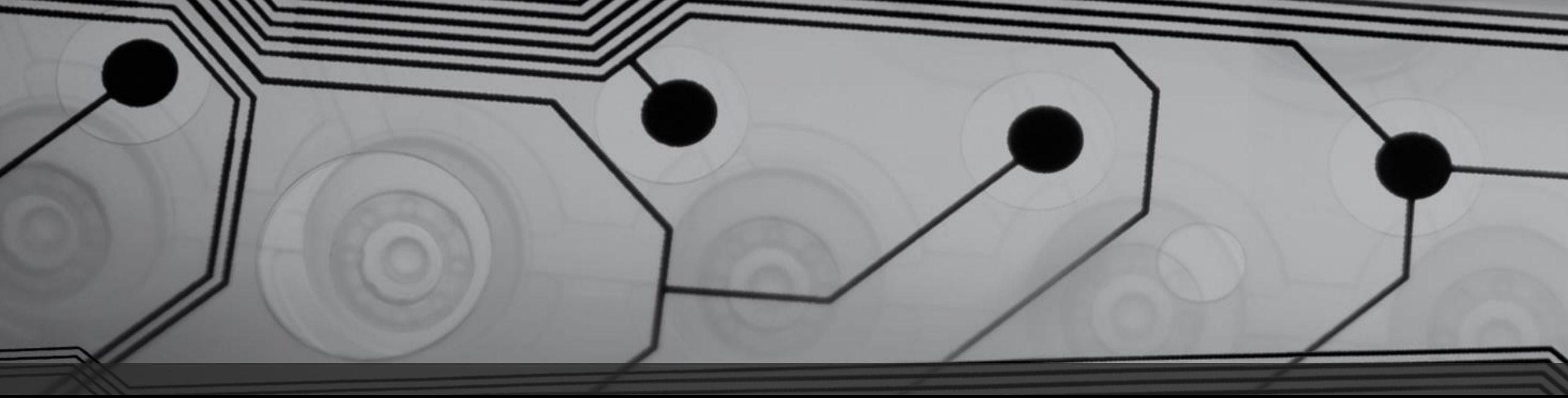
Software Architecture & Design





Messaging

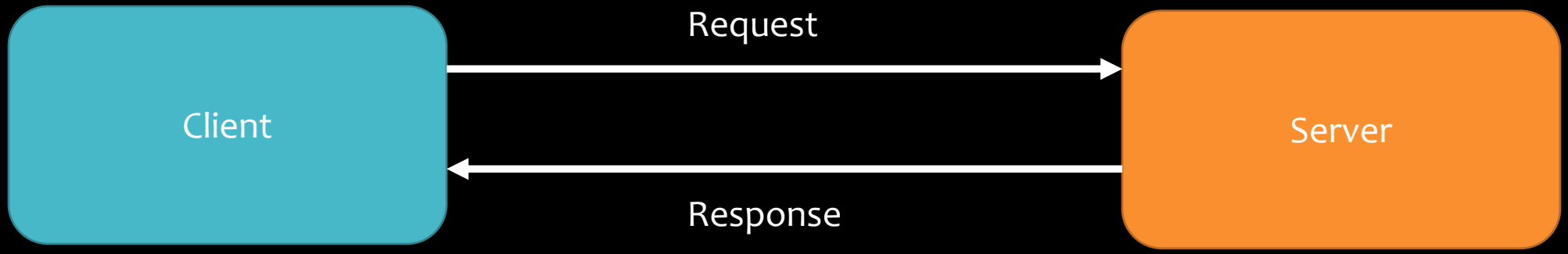
Event-based Architecture

Contents

- Messaging Overview
- Messaging Characteristics
- Notification Models
- Event Channel
- IPC Mechanics
- Middleware Design Patterns

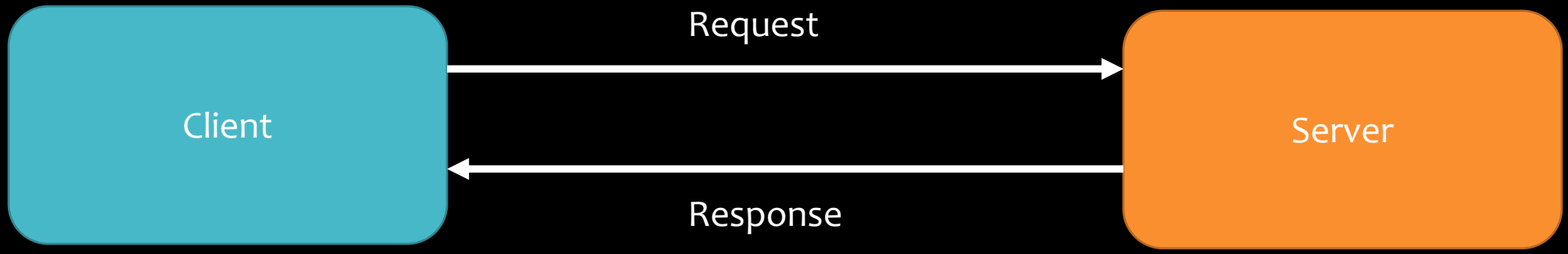
Messaging

How messaging emerged as the dominant paradigm in the Internet-age



Client/Server

1. Client makes a request to the server
2. Server responds to requests

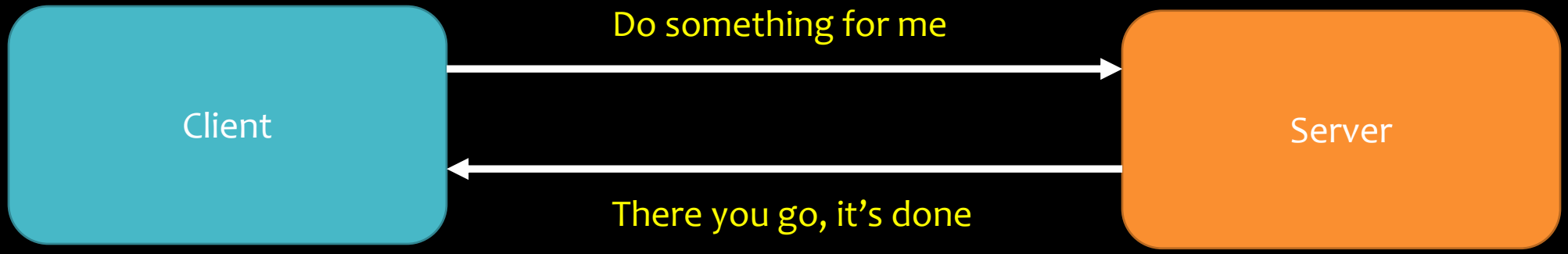


Distinct Roles: Client and Server

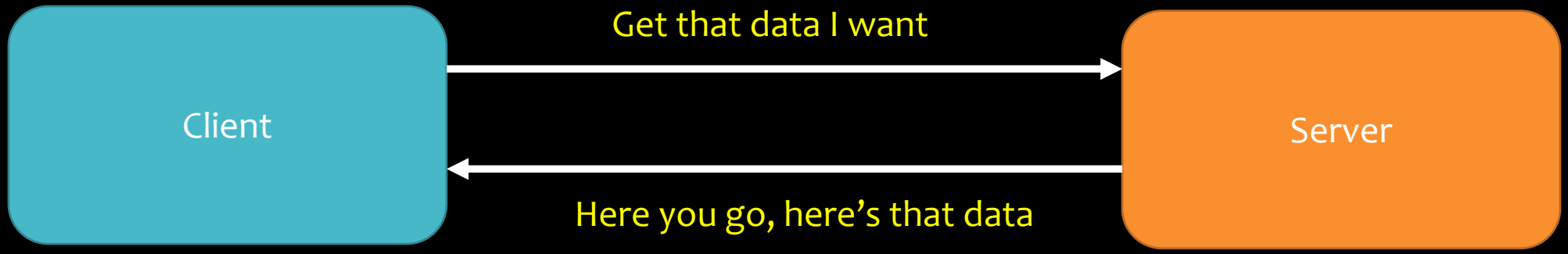
Client initiates interaction

Server is passive

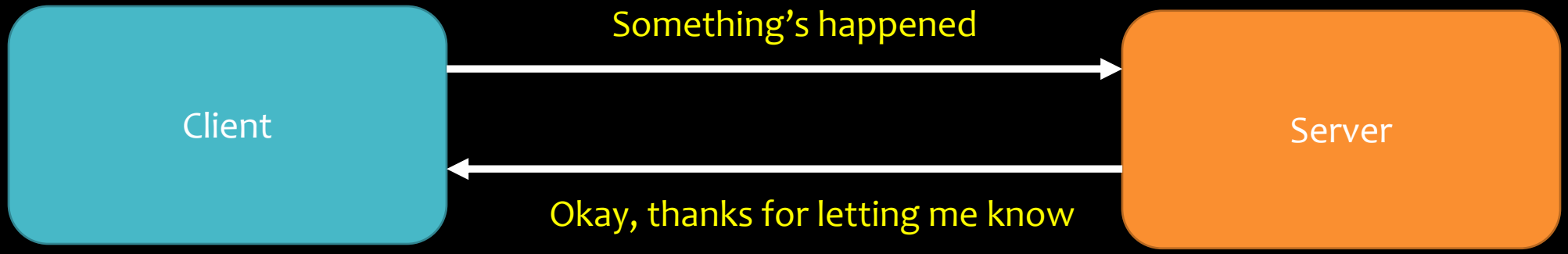
Types of Client/Server Requests



Request an action – i.e. issue a command

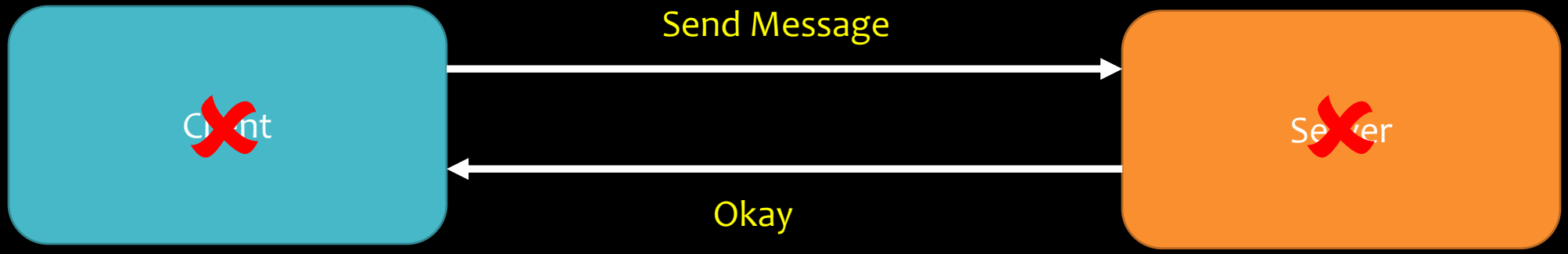


Issue a query – i.e. ask for some data set



Send notification – i.e. let server know of some event

With Messaging...



Client & Server Roles change

Client & Server Roles change

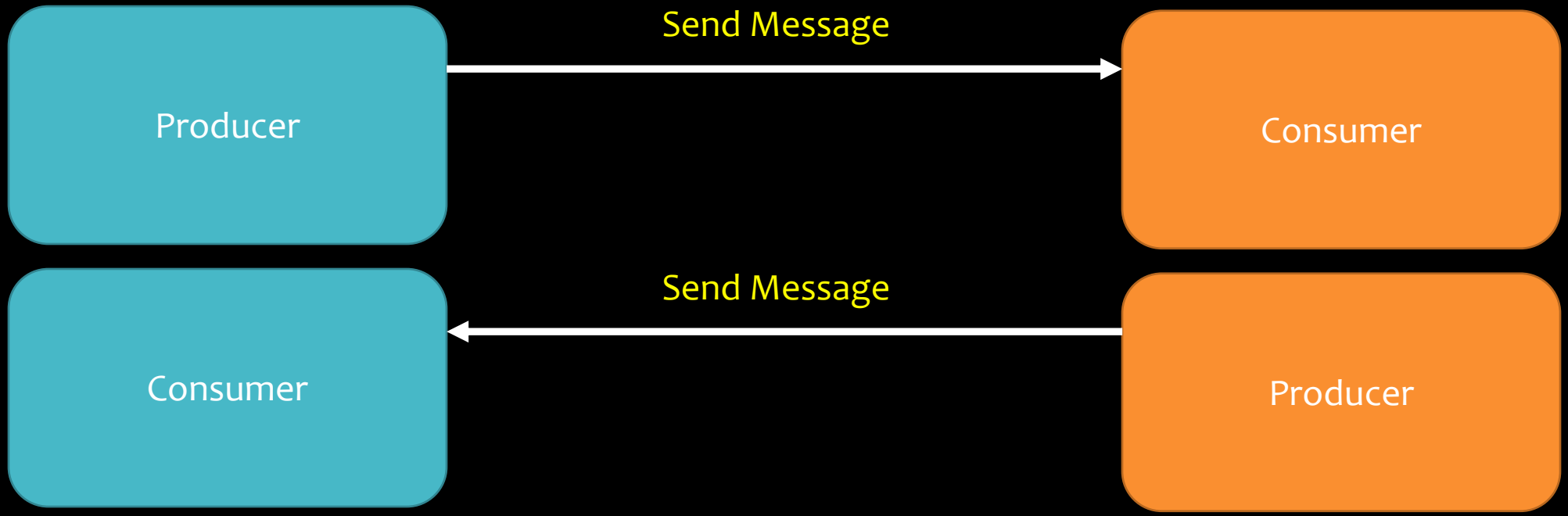
No longer Client/Server but...

Producer/Consumer

Supplier/Consumer

Publisher/Subscriber

Talker/Listener



Roles are reversible and interchangeable
Roles are variable and dynamic

Messaging Characteristics

Concepts & Conventions

Messaging Characteristics

- Message
- Event-based
- Producer/Consumer Roles
- Push/Pull Notification Model
- Decoupled

- Not a request/response
- Often one-way call
- May be sent asynchronously
  - Event dispatching queue
  - Event receiving queue

Messaging Characteristics

- Message
- Event-based
- Producer/Consumer Roles
- Push/Pull Notification Model
- Decoupled

- Event-based Modelling
- Sending ‘notification’ about event that has occurred on the system
- Simple Notification:- raw notification
  - i.e. force other party to invalidate cache & issue new request
  - e.g. user acknowledged alert
- Smart Notification:- contains update details
  - i.e. enough info to update cache
  - e.g. alert id=132, status=ack

Messaging Characteristics

- Message
- Event-based
- Producer/Consumer Roles
- Push/Pull Notification Model
- Decoupled

- Roles are not reserved – any element can be a producer or a consumer
- Any element can and typically are both producer and consumer

Messaging Characteristics

- Message
- Event-based
- Producer/Consumer Roles
- Push/Pull Notification Model
- Decoupled

- Push Model - Events are pushed by the producer/supplier/publisher/talker
- Pull Model – Events are pulled by the consumer/subscriber/listener
- Hybrid: Event Channels allow Publisher push events and Consumer pull events

Messaging Characteristics

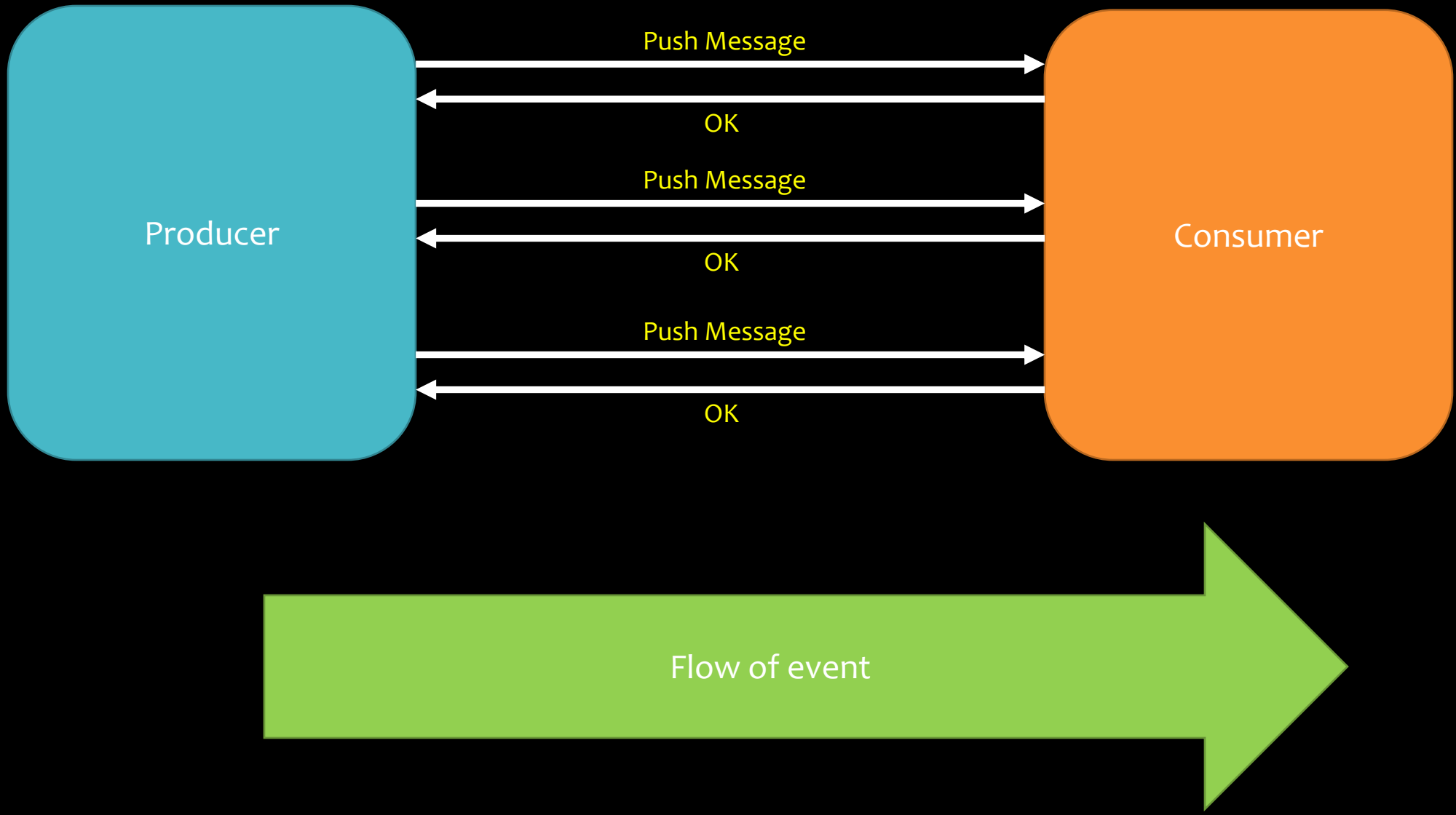
- Message
- Event-based
- Producer/Consumer Roles
- Push/Pull Notification Model
- Decoupled

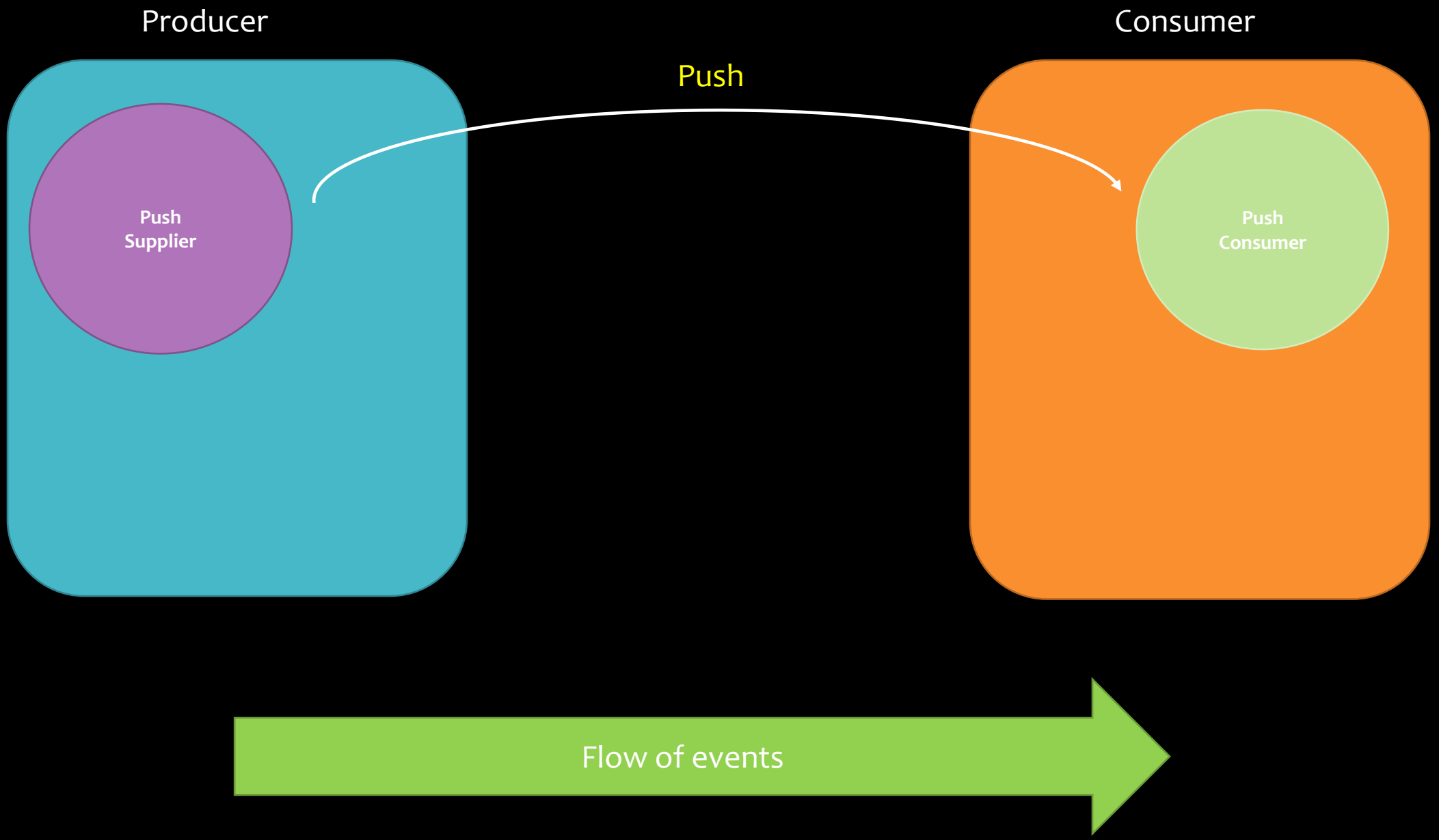
- Producers and Consumers need not know about each other & can be decoupled
- Event Channel acts as a buffer, isolating Producers from Consumers & vice-versa
- Infers asynchronous event flow
- Typically infers queues
  - receiving/dispatching

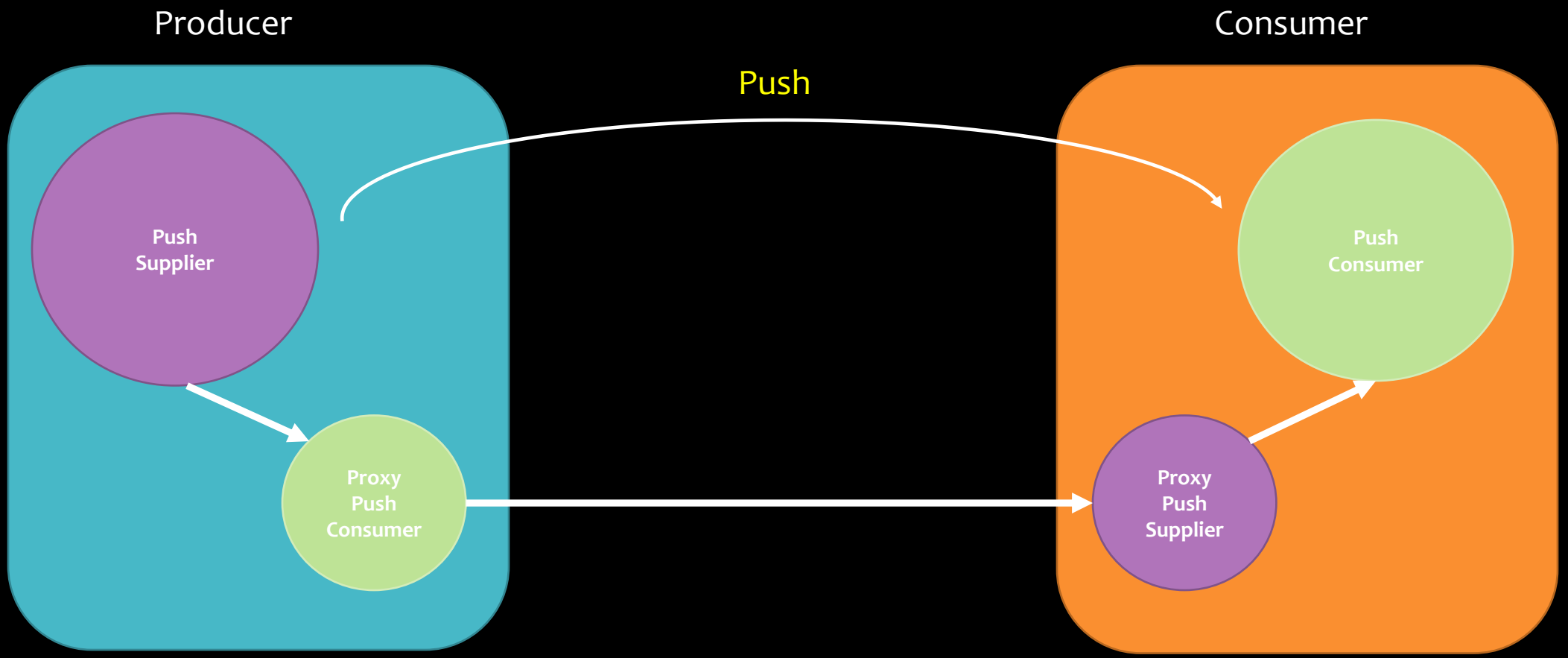
Notification Models

Push-Event & Pull-Event

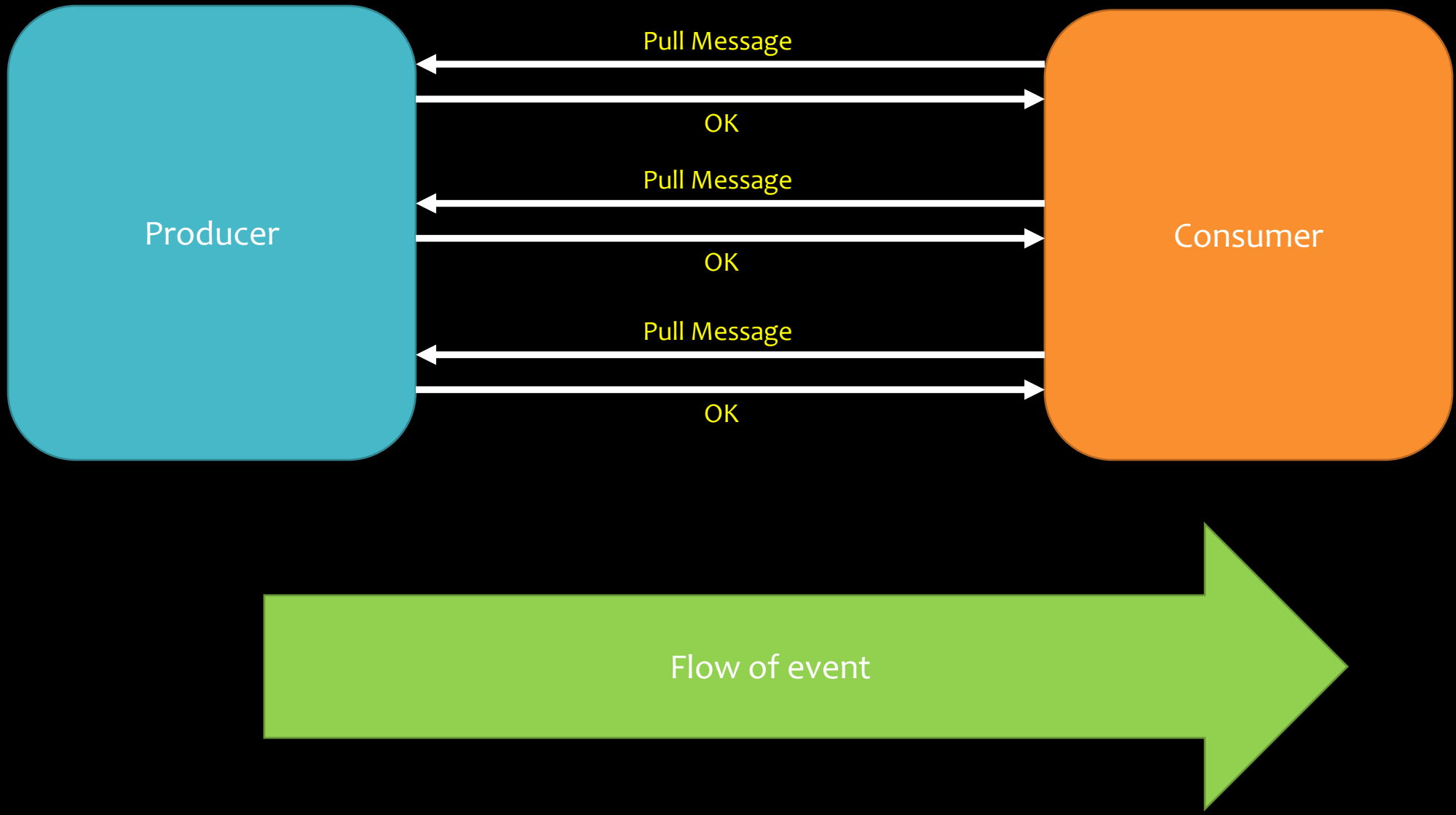
Push Event Model

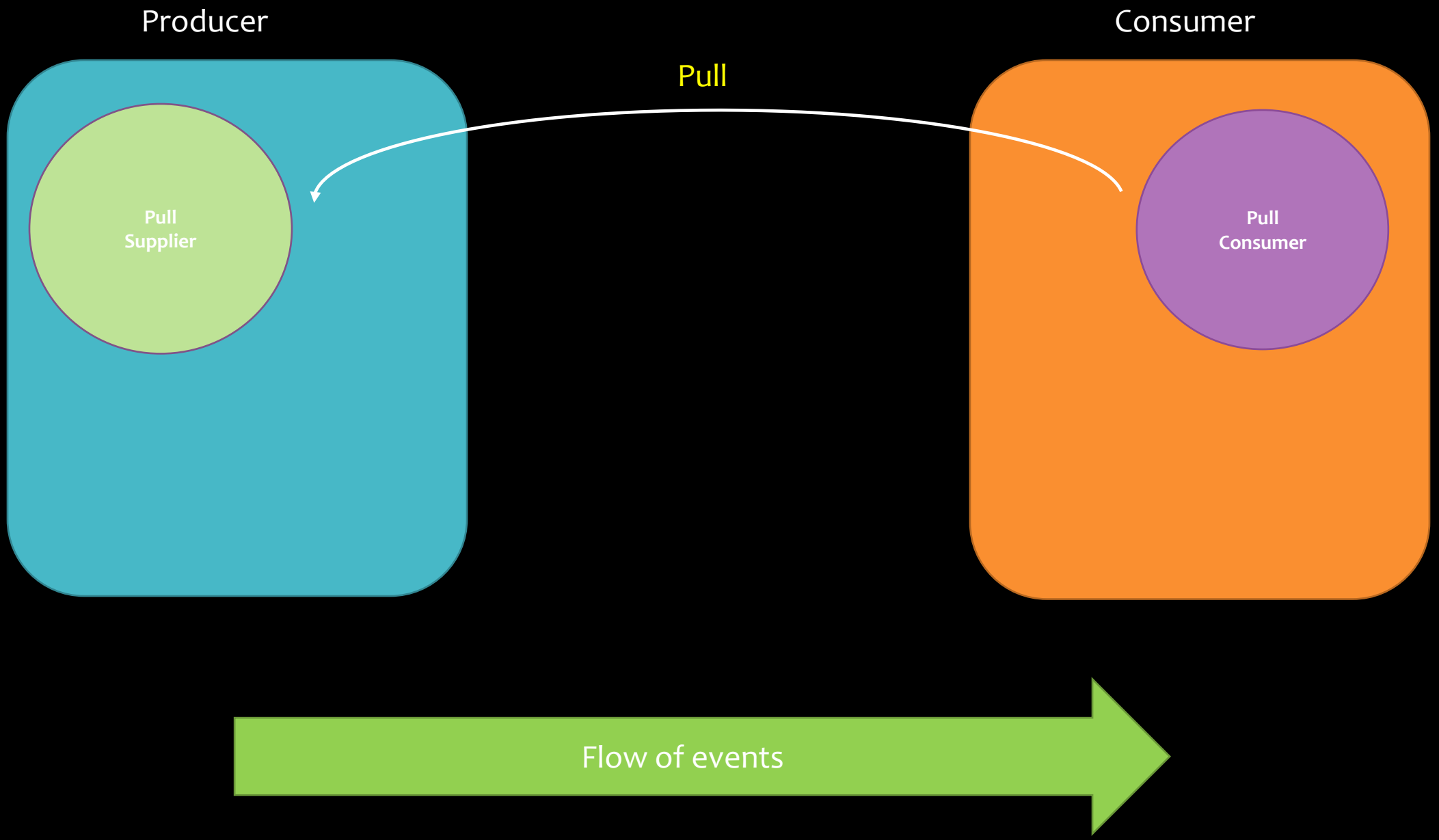


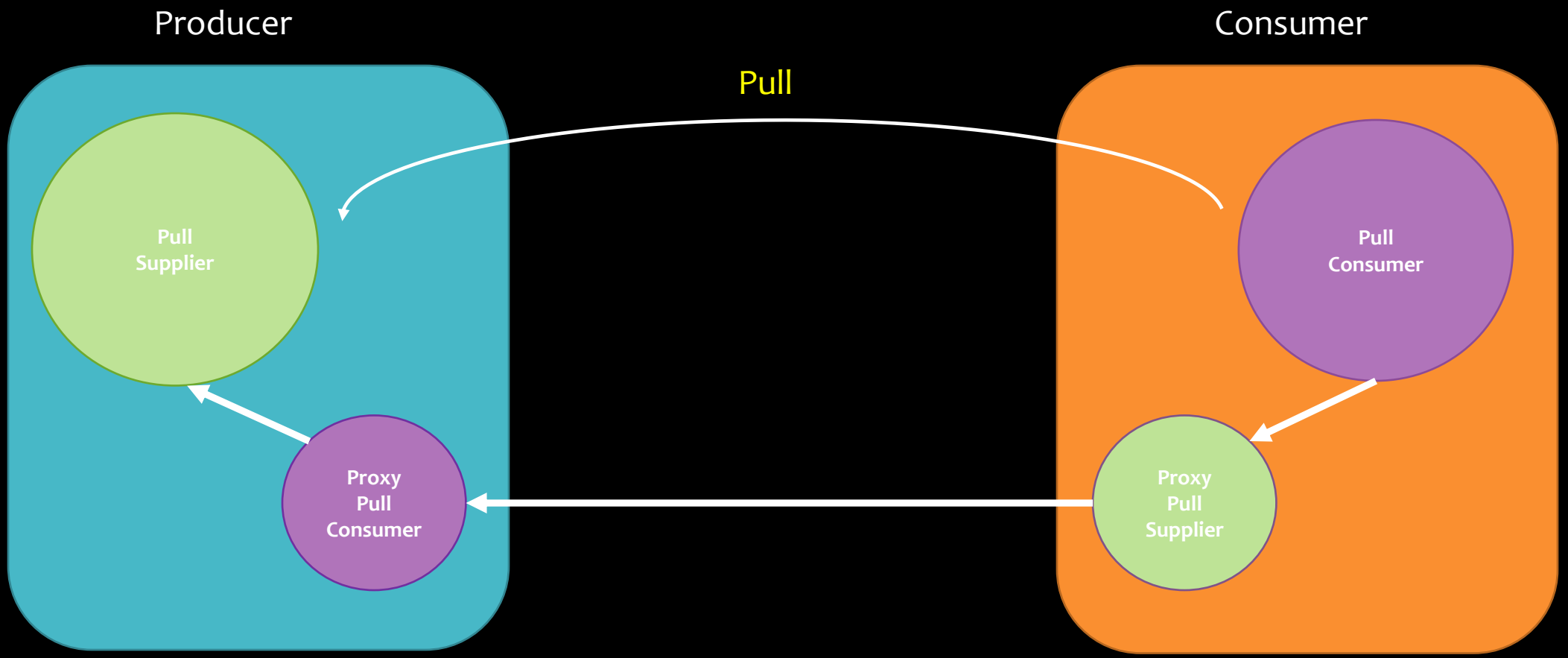




Pull Event Model



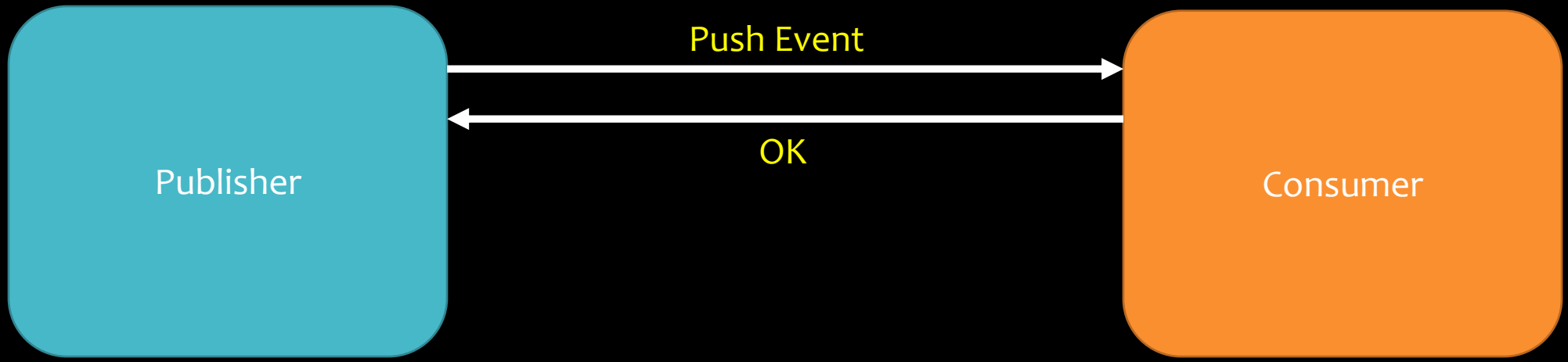


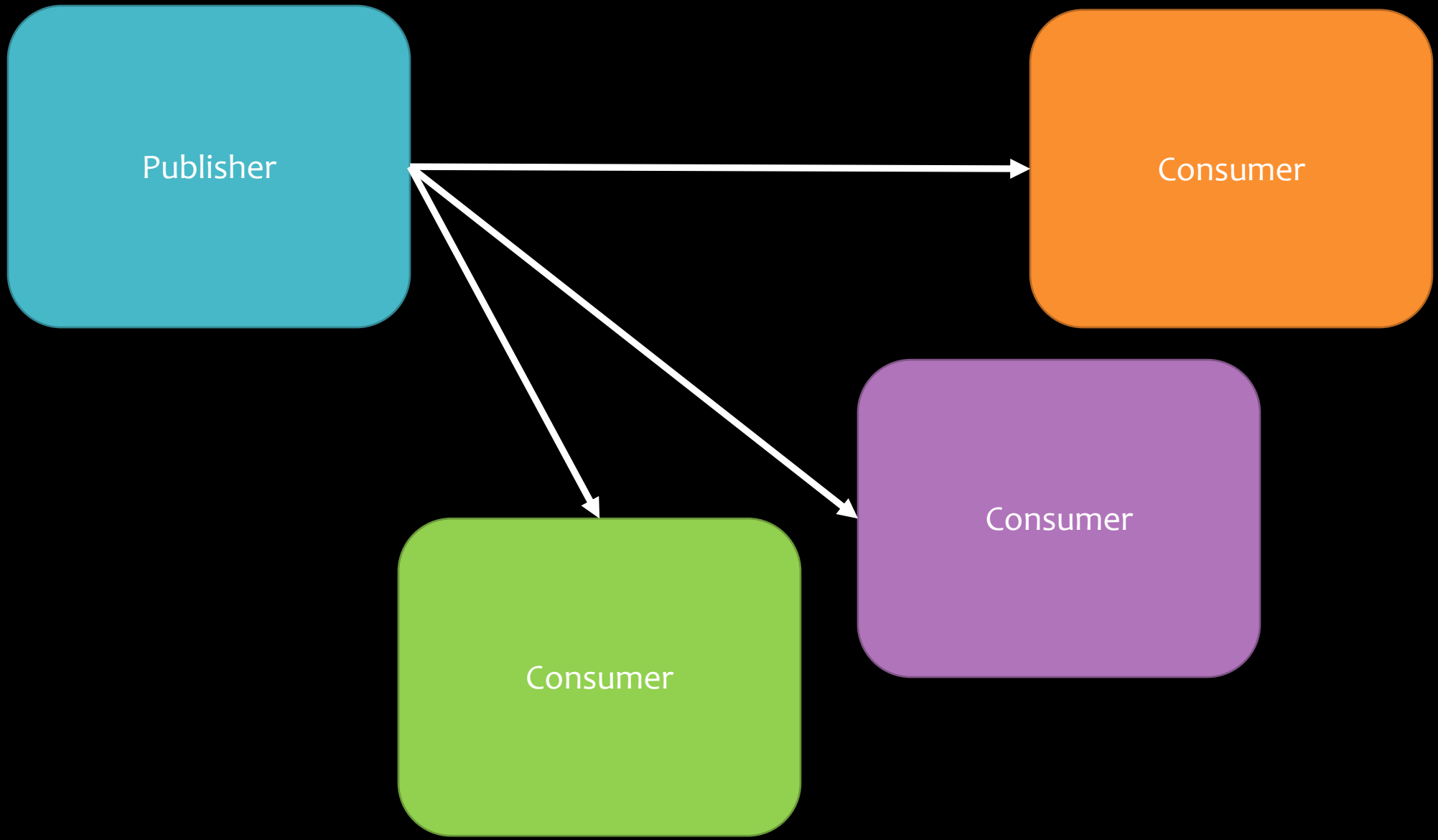


Event Channel

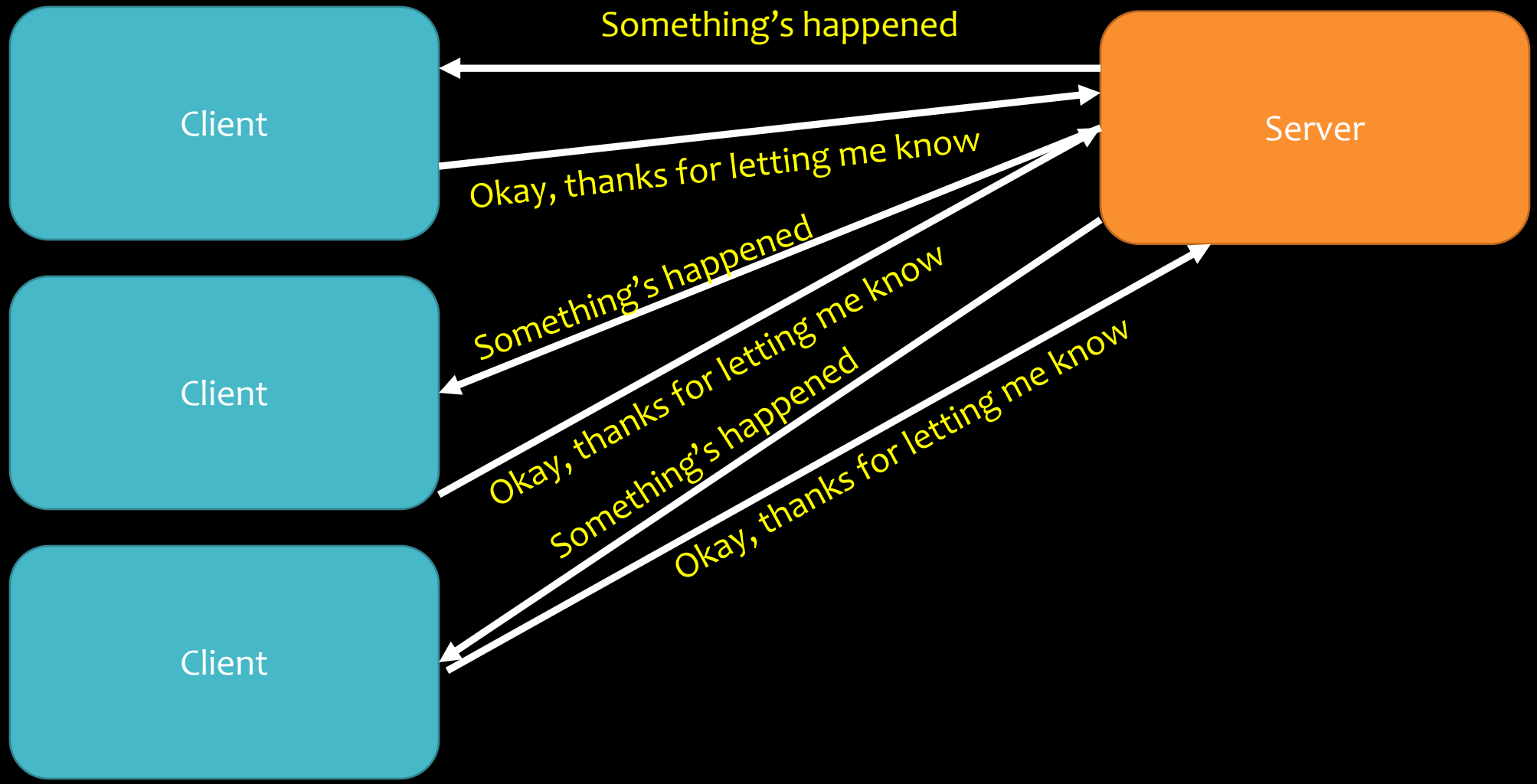
Messaging with Event Services & Brokers

Notifying Consumers
Client → Server





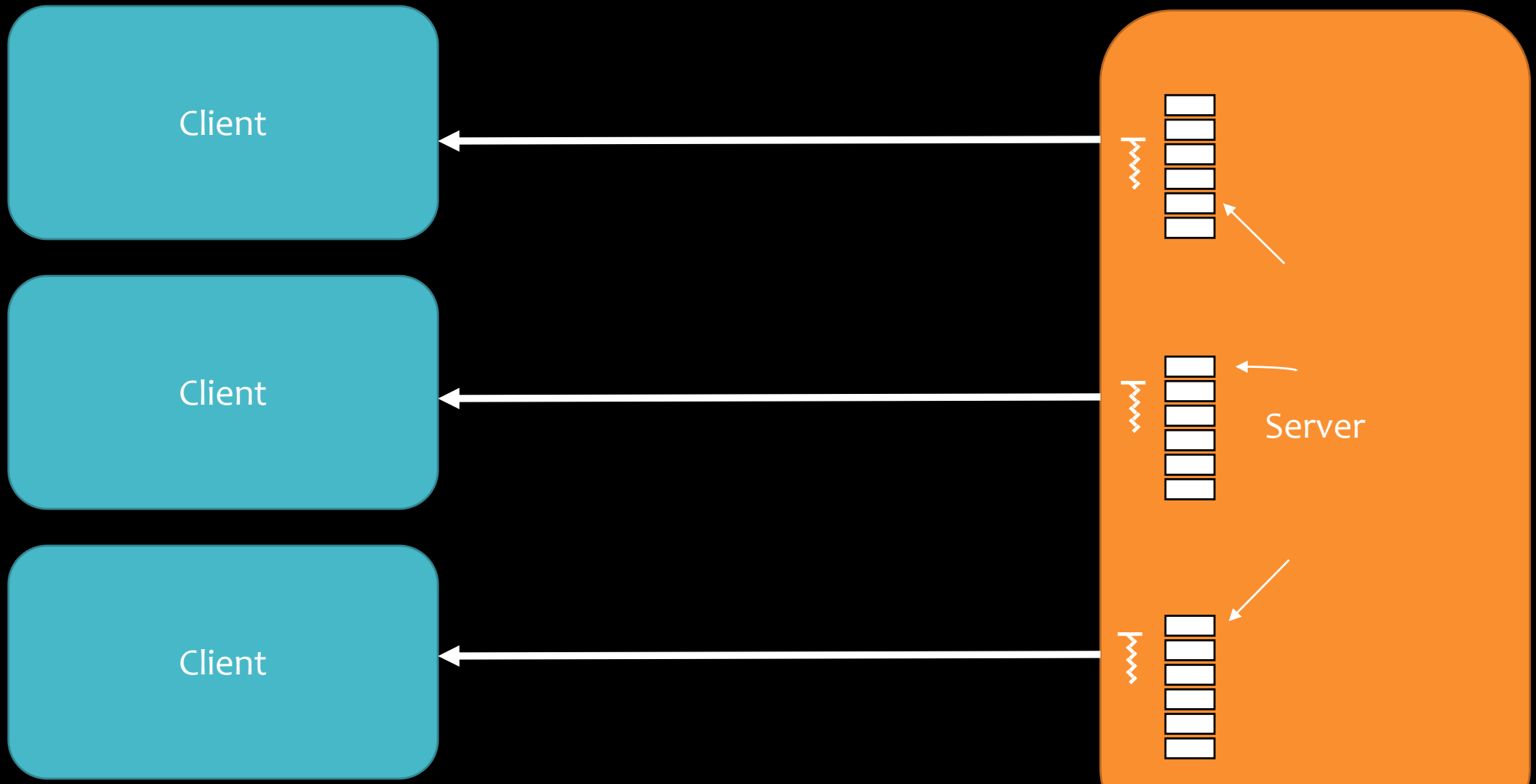
Notifying Consumers
Server → Clients



Messaging Challenges

- Obtain list of consumers
  - i.e. interested parties
- Manage list of consumers
  - Add new subscribers
  - Remove existing subscribers
- Timely operation
  - i.e. cycle through list ensuring each consumer is sent message
  - Last recipient versus first recipient
    - i.e. Time sensitive messages (e.g. Stock Prices)
- Consumers blocked while message is being received
  - Thread used to receive message is thread that reacts to message and does work => Delay sender
  - Messaging Convention: Yield
- Consumers consume messages at different rates
  - Events could be time-sensitive or coordinated
  - How to manage?

Thread-per-Consumer Model



Thread-per-Consumer Model

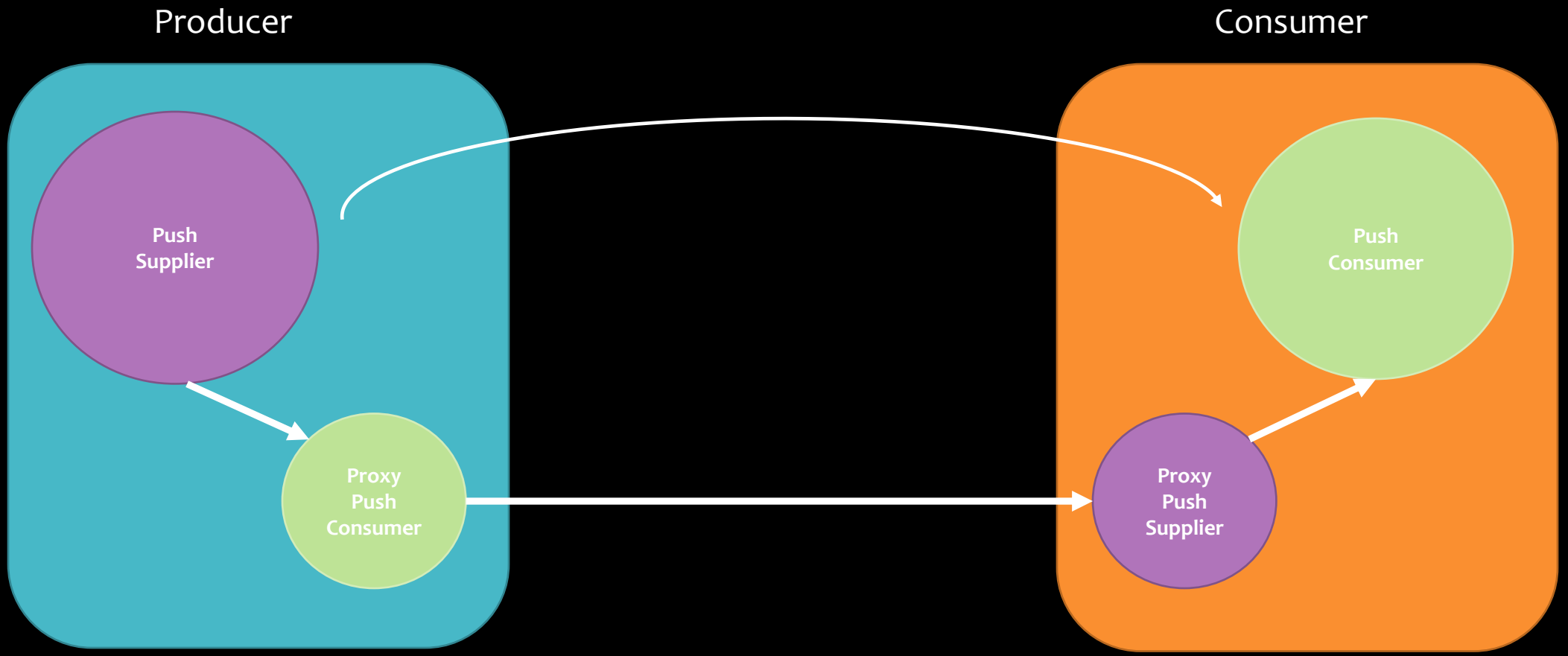
Advantages

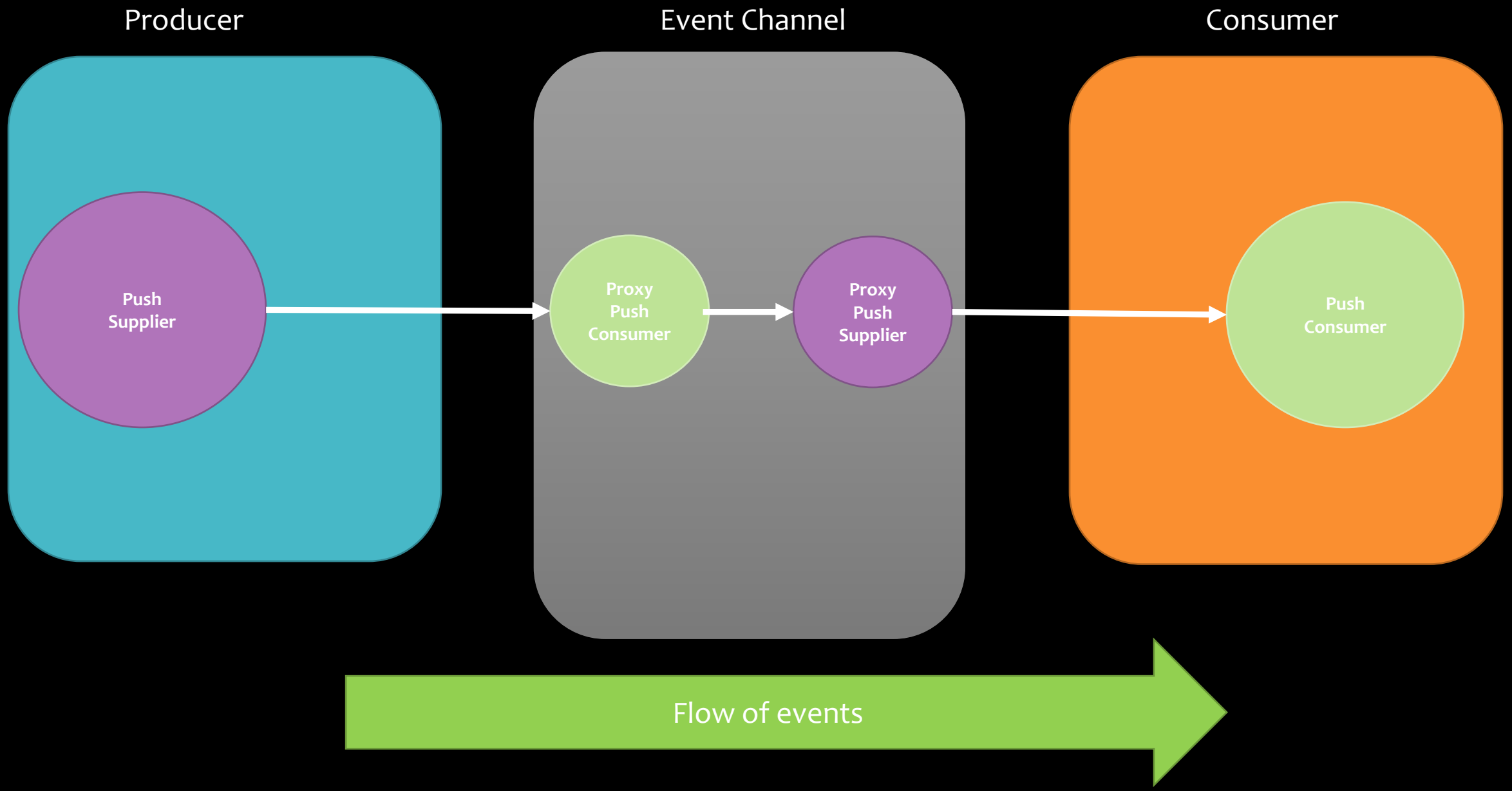
- All Consumers sent event at the same time*
- Differing Consumer consumption rates handled with dedicated queues

Disadvantages

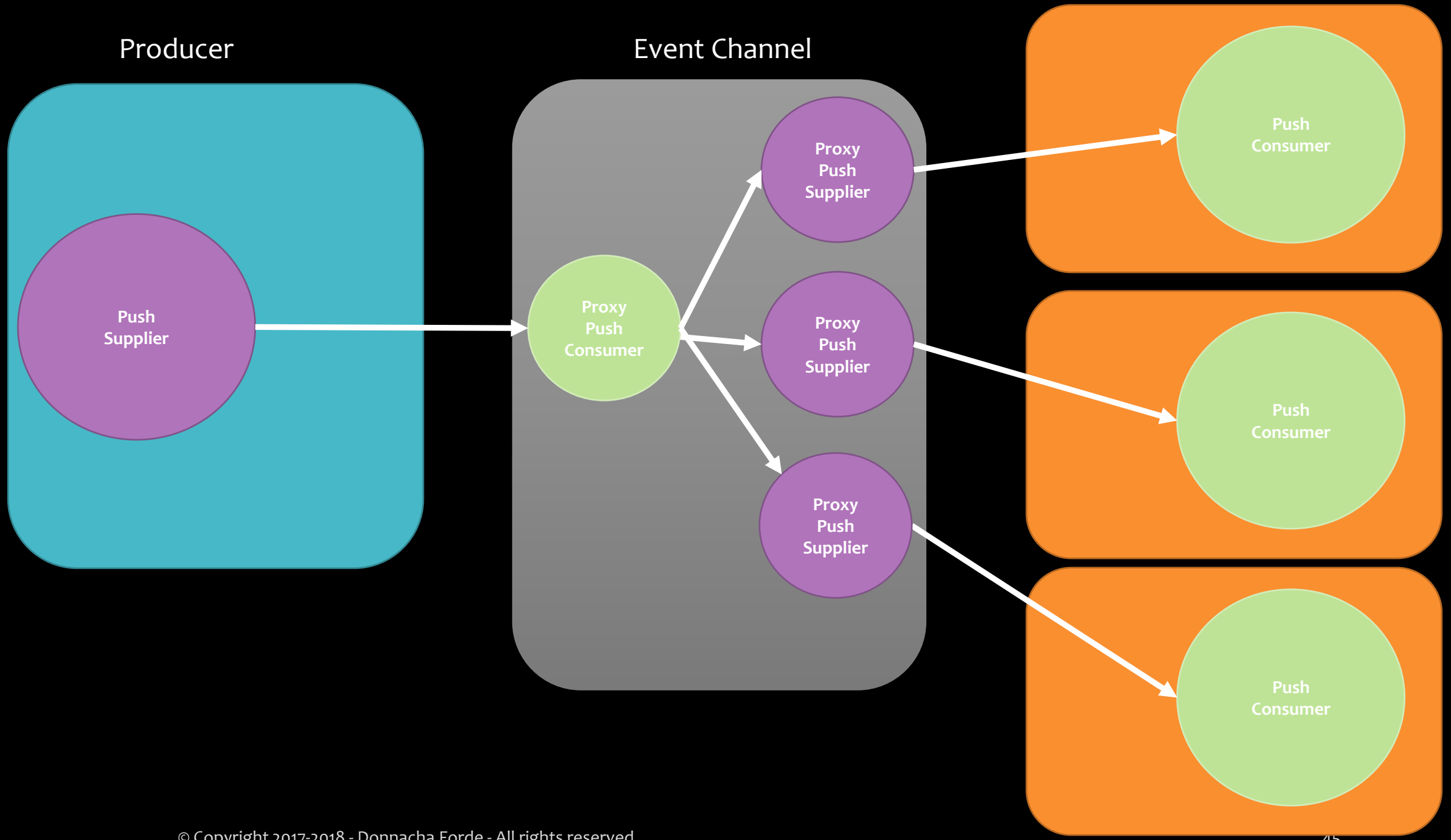
- Complexity – More complex programming model
  - Concurrency control for adding/removing items to/from queue
  - Adding/Removing Consumers
- Producer memory occupied
  - Risk of queue backlog
- Error Handling – Replay requests
- Delayed Consumption - Historic events

Event Channel

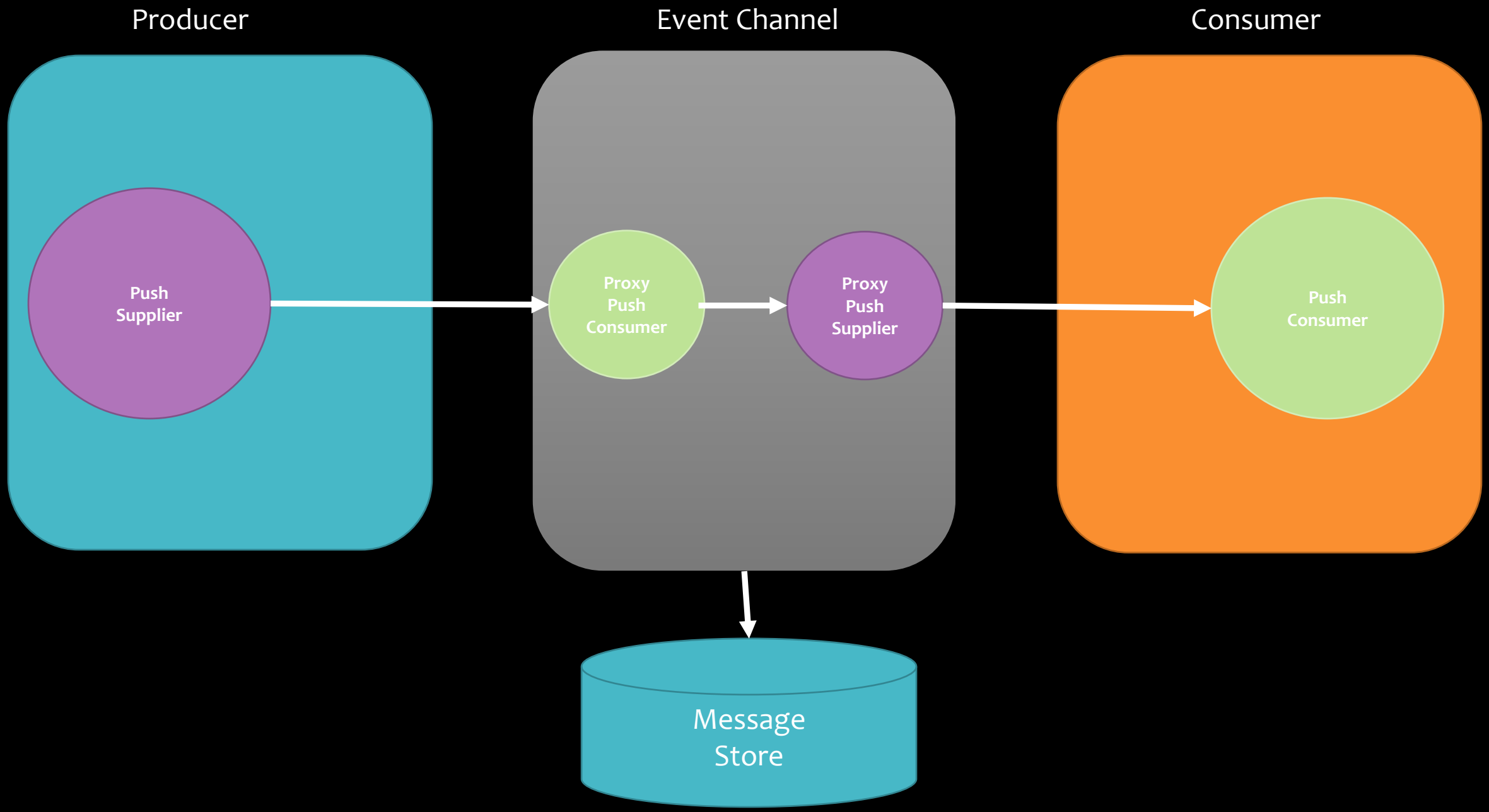




Event Channel Multicast Mechanism



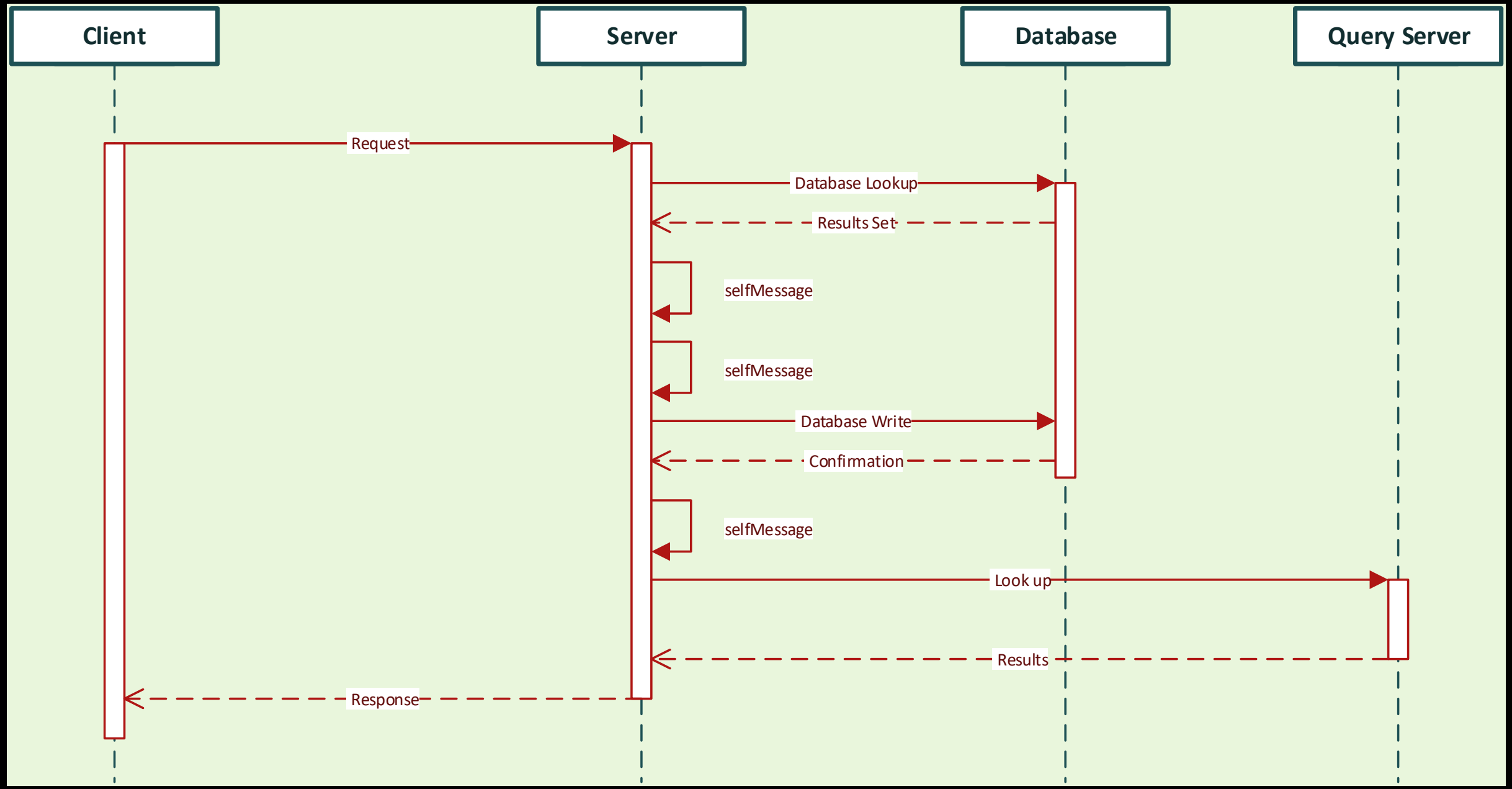
Event Channel + Message Store = Guaranteed Delivery



IPC Mechanics

Synchronous versus Asynchronous IPC

Synchronous Client/Server Call



Synchronous Client/Server Call

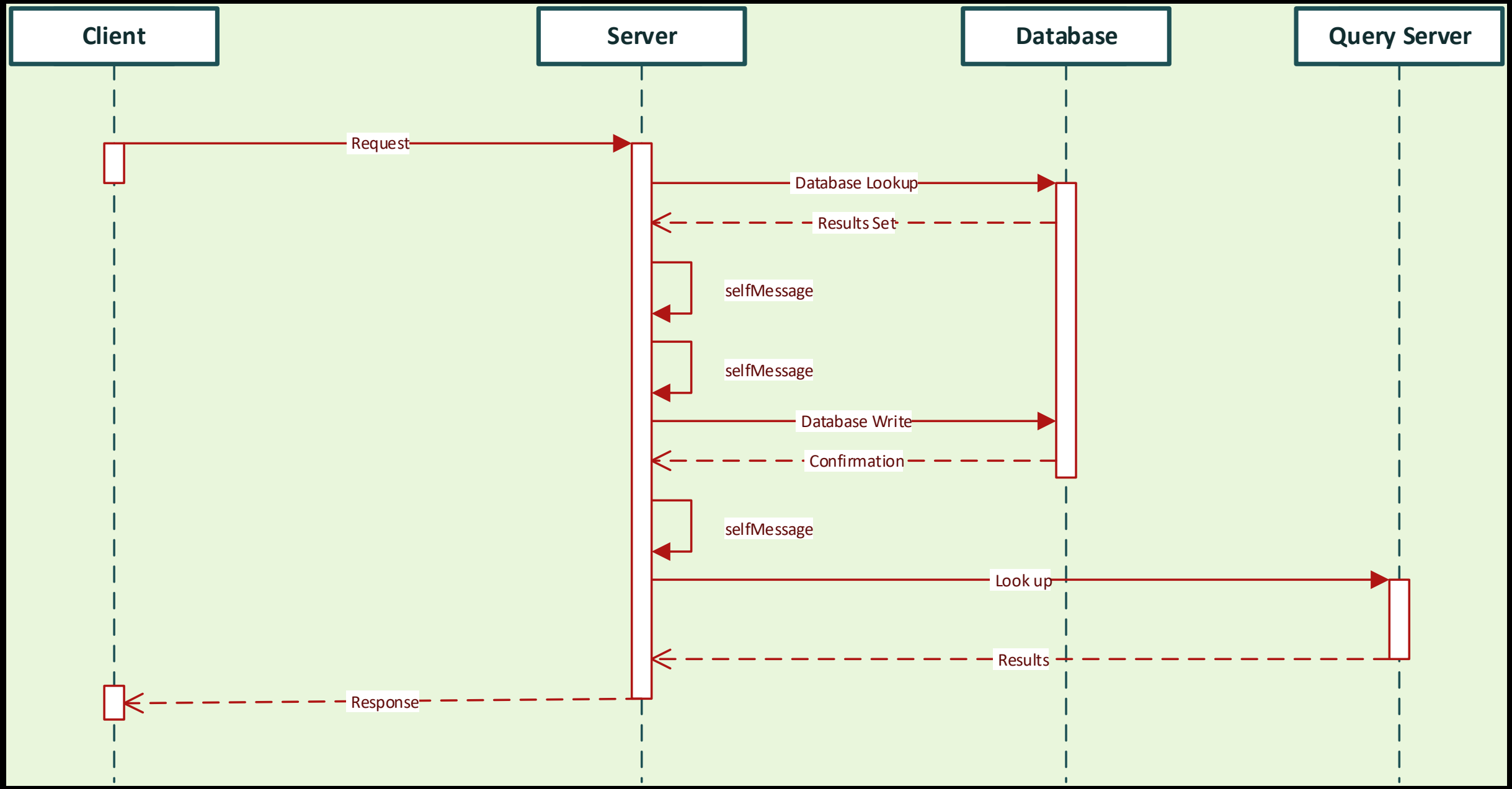
Advantages

- Simpler Programming Model
  - Synchronized access
  - Avoid any threading or concurrency control issues
- Familiar Programming Model
  - IPC Request/Response similar to local function call (i.e. RPC)

Disadvantages

- Blocking
- Surrender Control
  - i.e. Unable to do any work on blocked call
- Possibly poorer UX
  - e.g. GUI hung while waiting for a response

Asynchronous Client/Server Call



Asynchronous Client/Server Call

Advantages

- Non-Blocking
  - Caller (caller thread) is free to do other work
  - More efficient use of thread
- More responsive UI

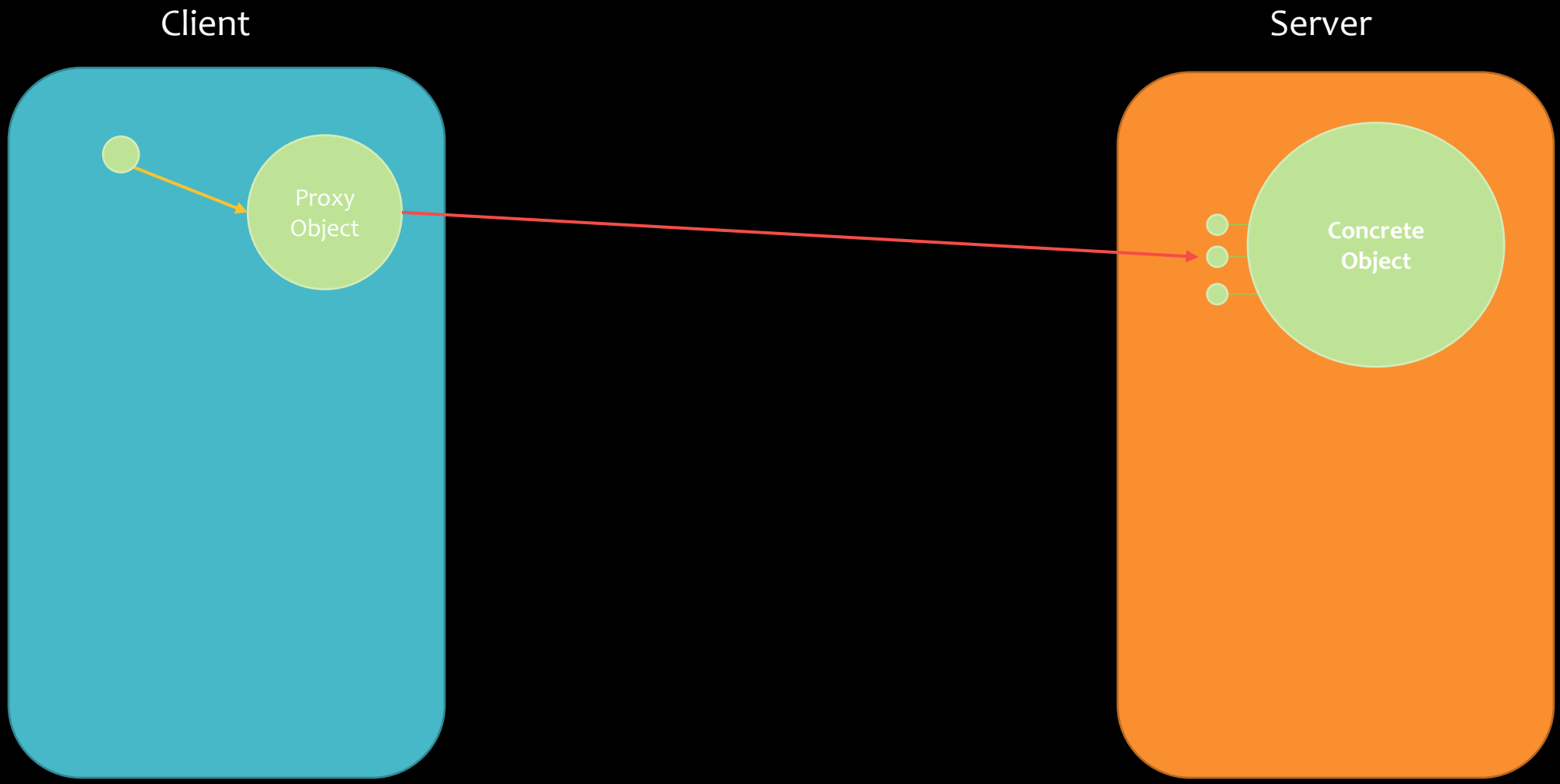
Disadvantages

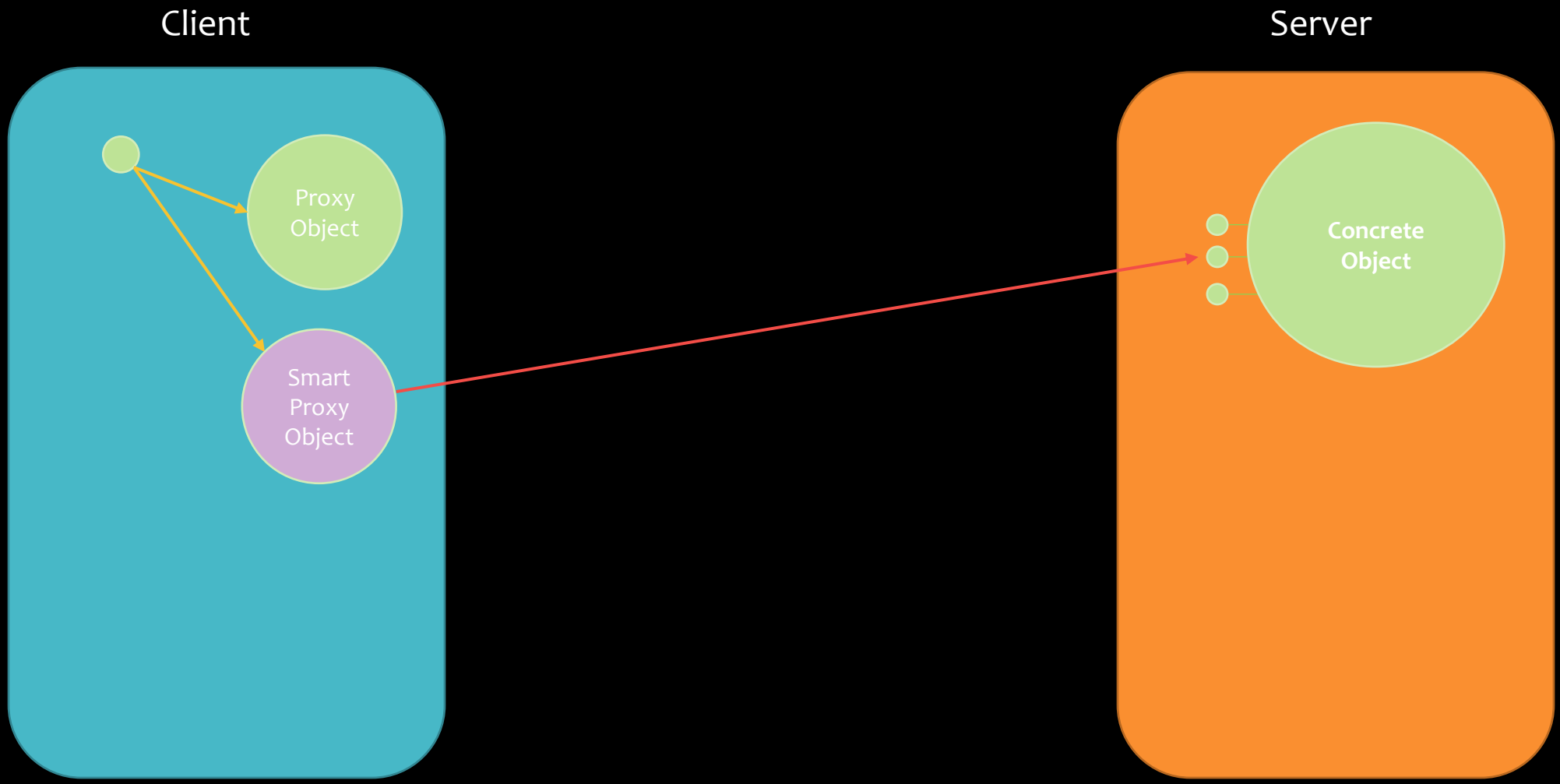
- More complex programming model
  - Need to employ thread concurrency controls
- Need to co-ordinate processing of response
  - i.e. Need to 'check back' to see a response has been received and is ready to be processed
- Correlation
  - May need to match specific response with corresponding request

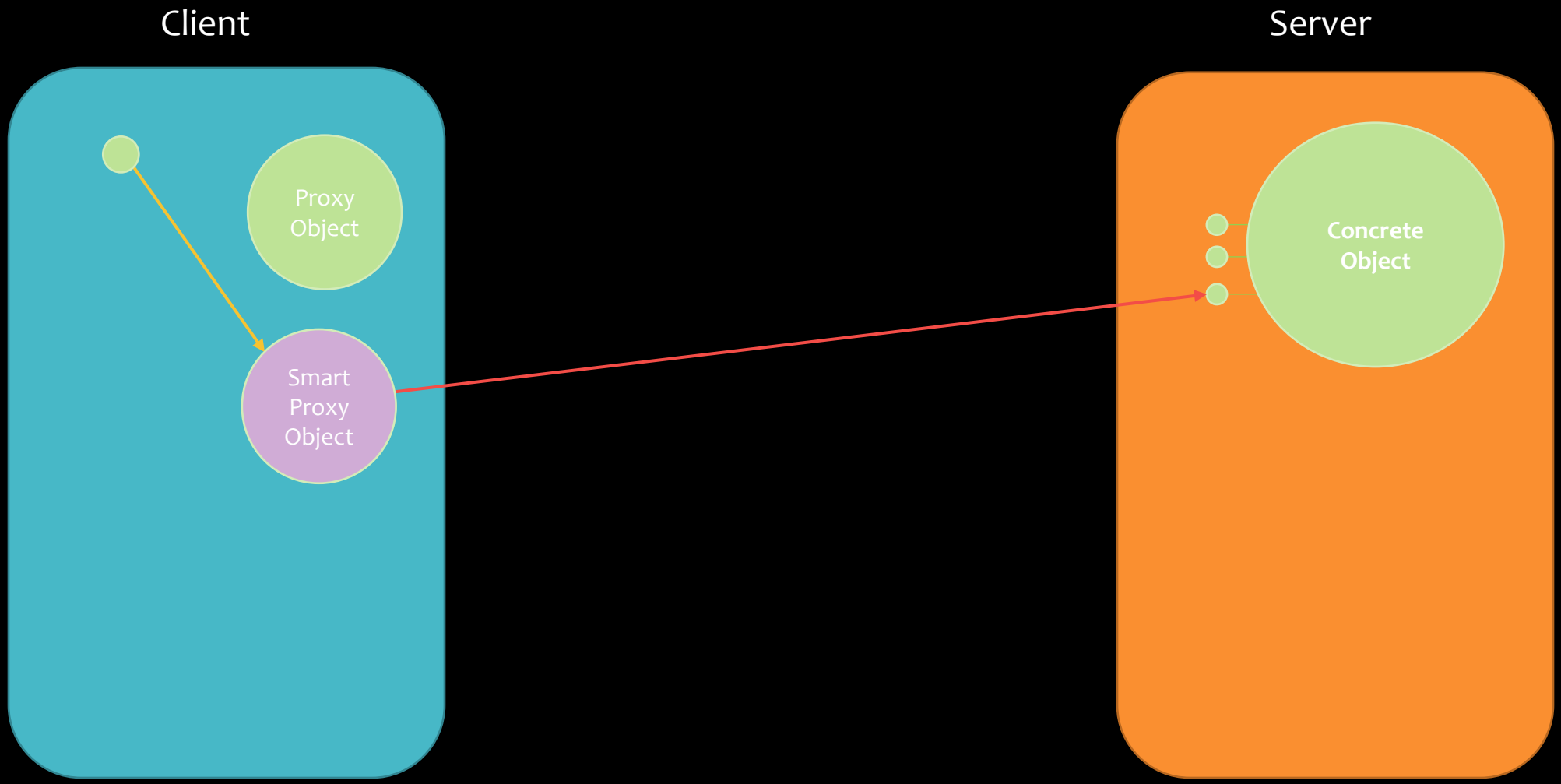
Middleware Design Patterns

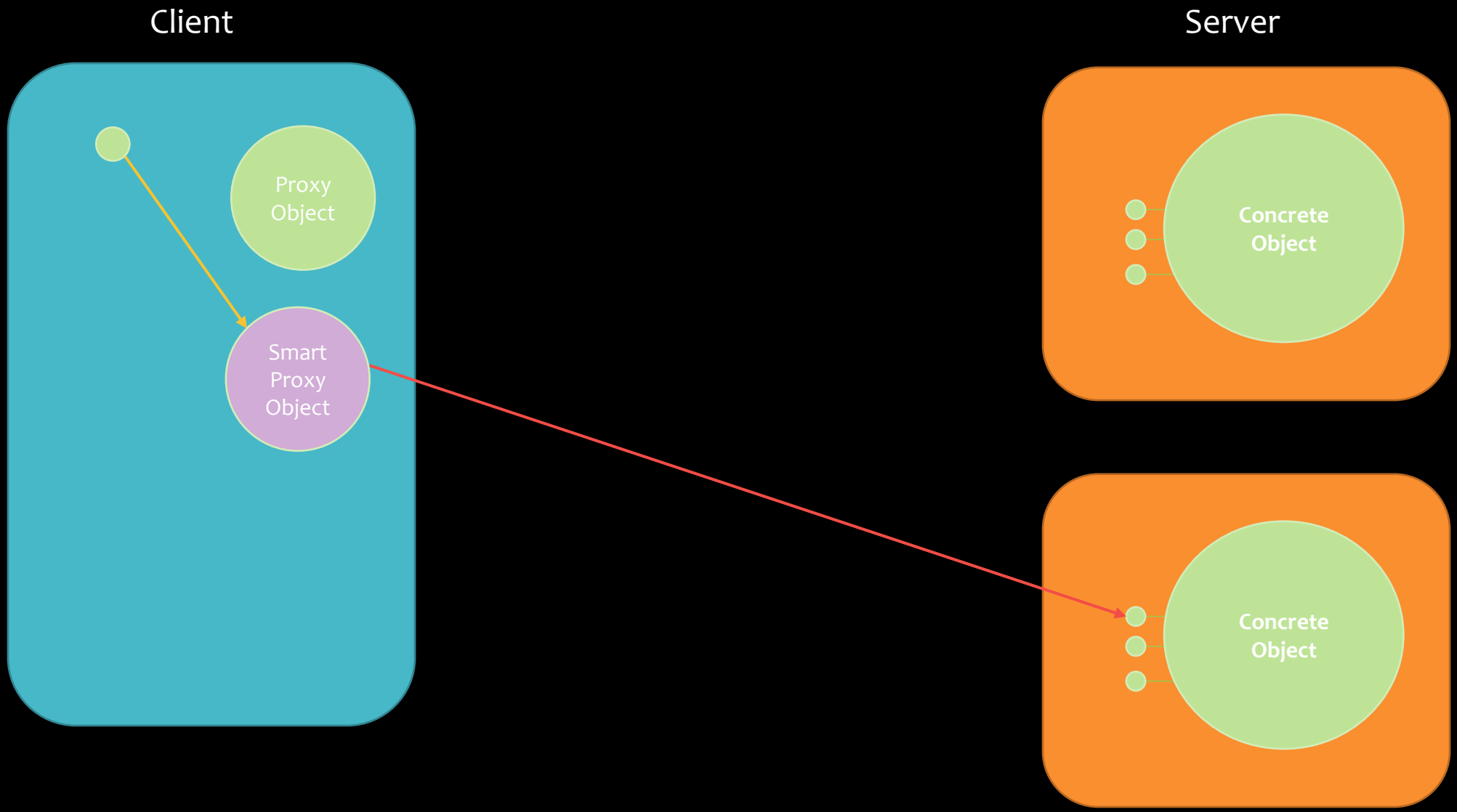
Smart Proxy

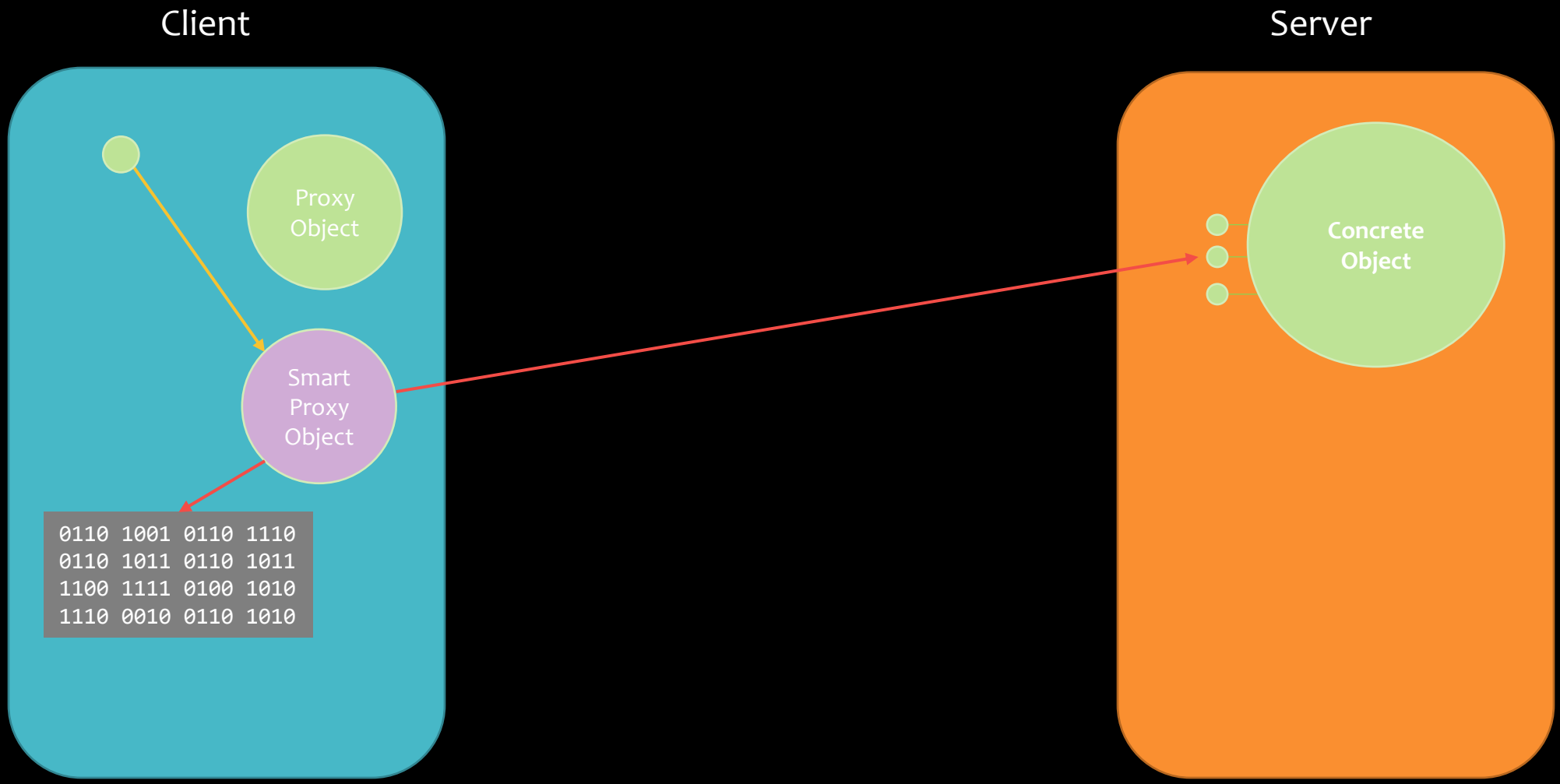
Smart Proxy











Smart Proxy Requirements

- Encapsulation – The caller should not be aware that a smart-proxy is in effect
- Smart-Proxy must implement all interfaces of original proxy
  - May need to delegate to original proxy
- Creation of Smart Proxy should be transparent
  - Need to rely on a Creation Pattern to ensure Smart Proxy is created in place of the original proxy

Smart Proxy

Advantages

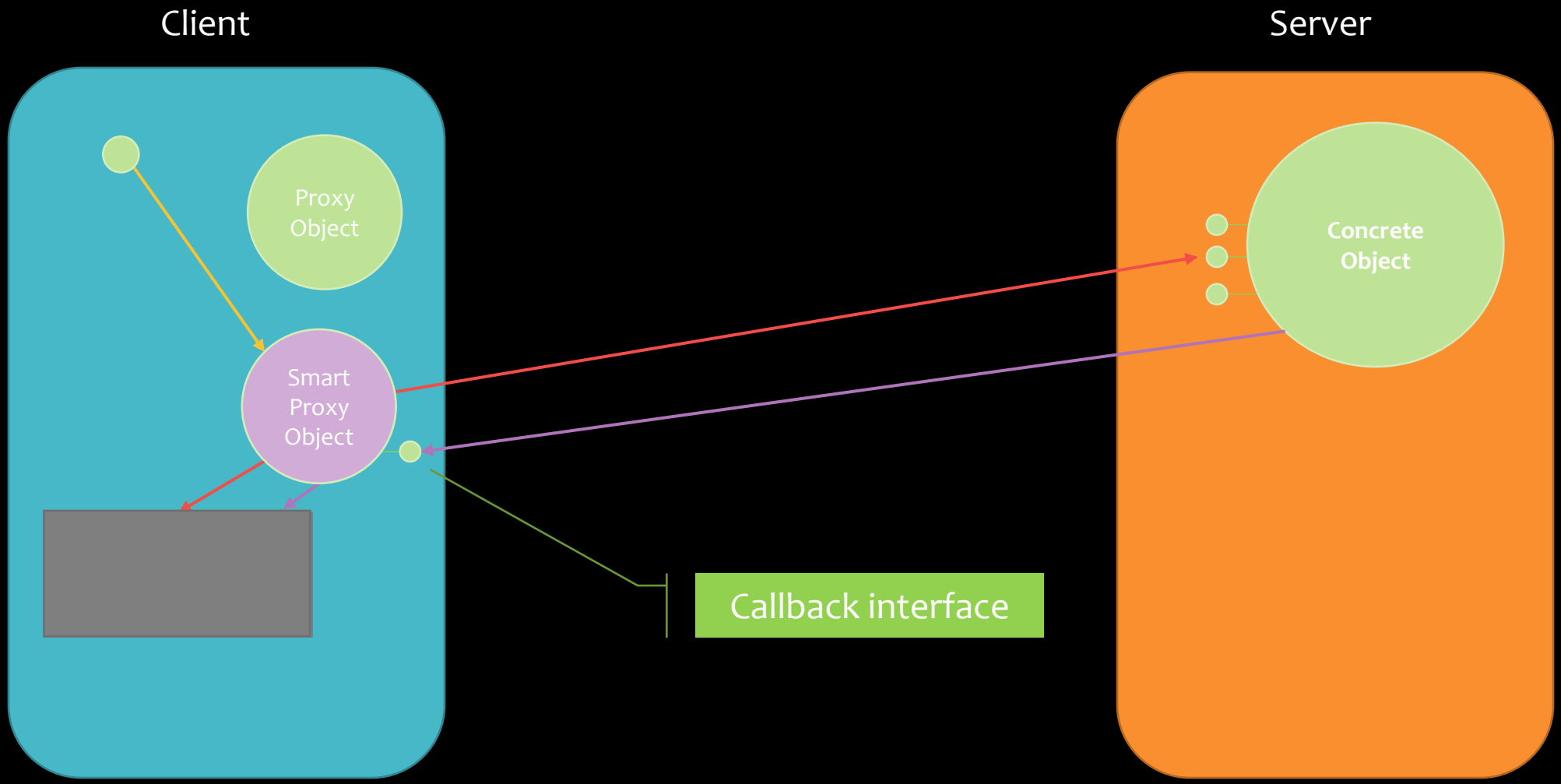
- Deploy server-oriented logic
- Save unnecessary round-trip calls
- Buffer/Cache expensive results

Disadvantages

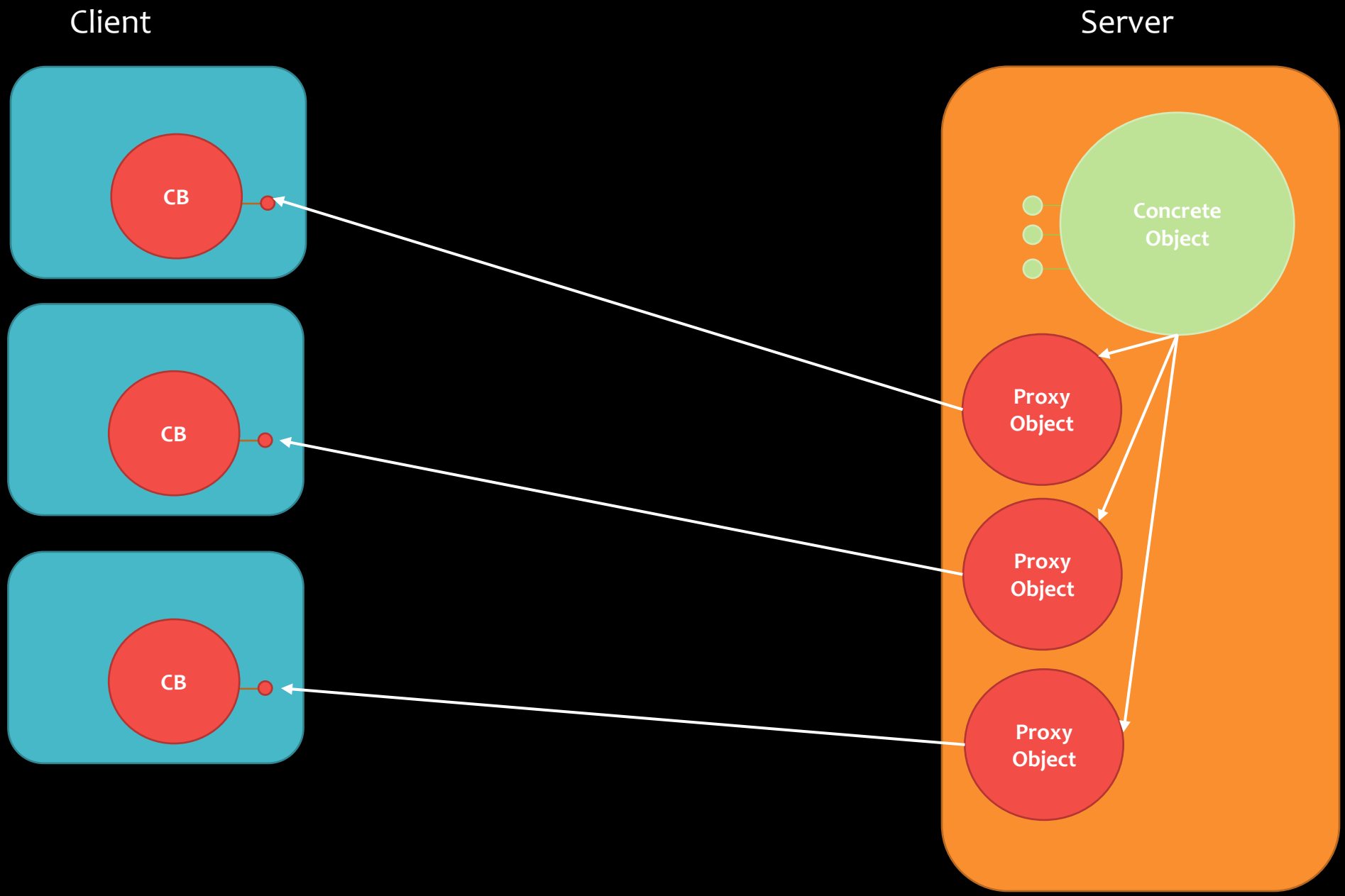
- Custom coding likely required
- Need to make available to client-side developers
- May be better to load-balance on server/cloud
- Need to invalidate cache!

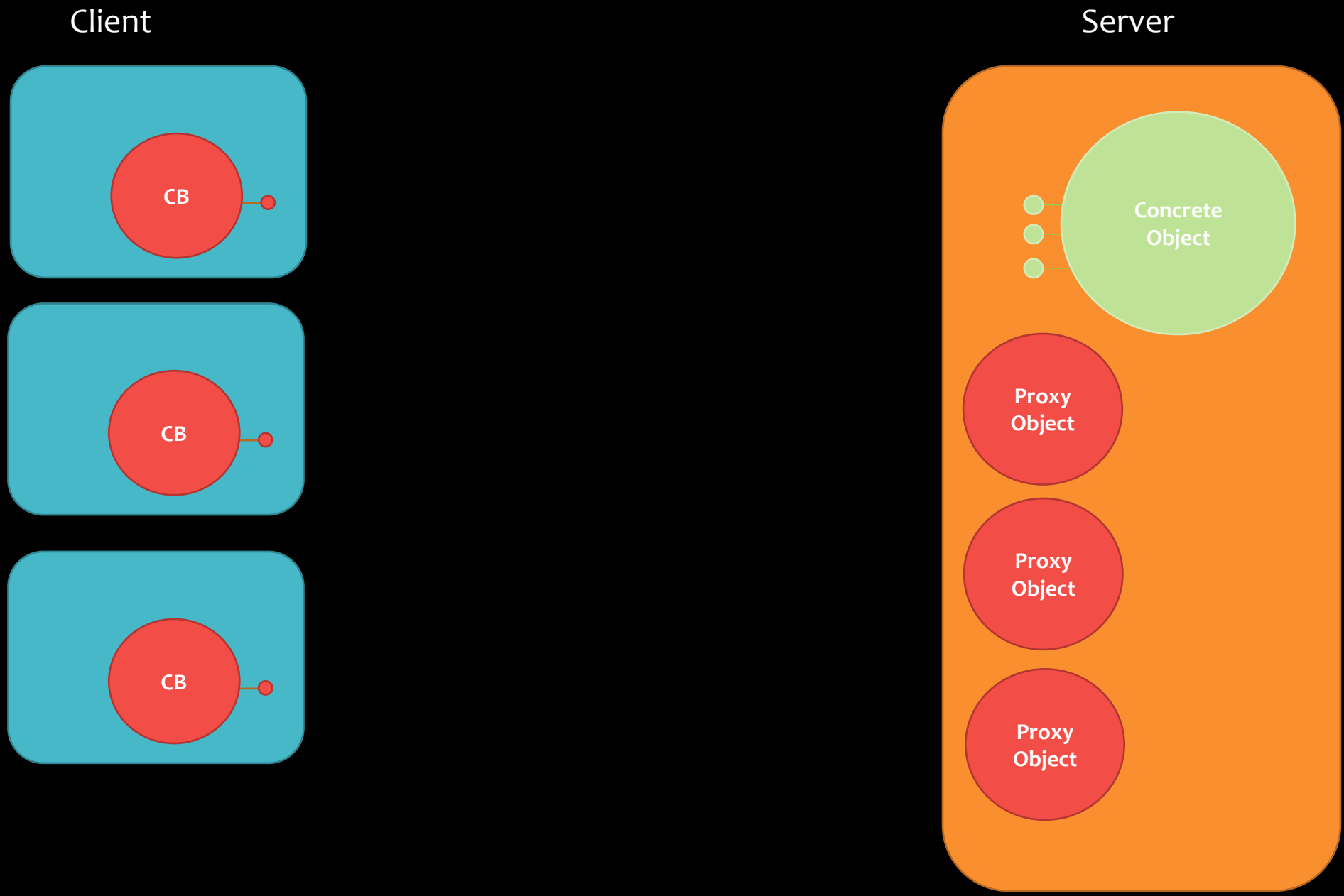
Smart Proxy + Callback

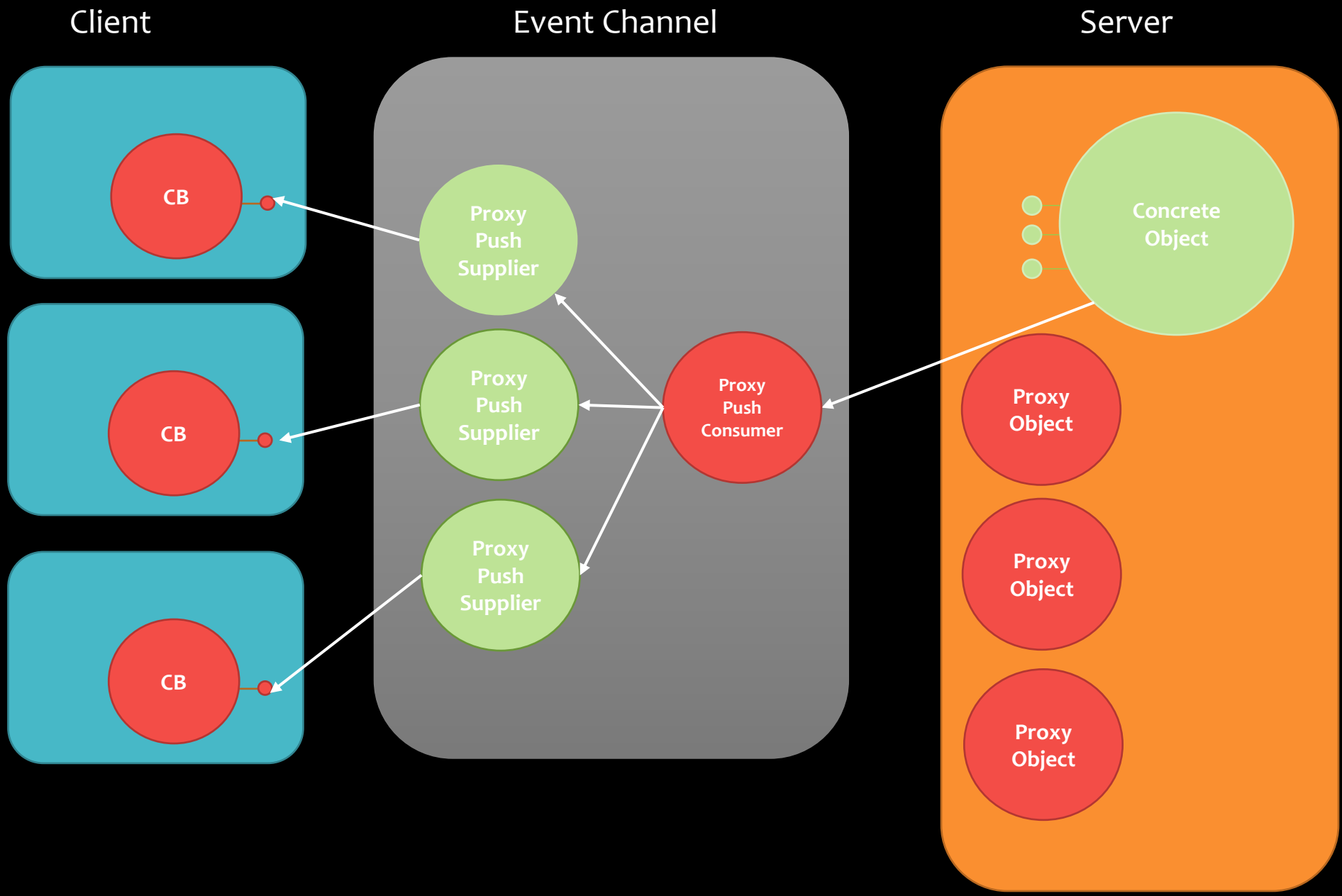
- Implement a Callback mechanism to allow Server to communicate with Client
  - e.g. Invalidate Cache



Callback at Scale







Code Demo

MQTT Publisher & Subscriber

Q&A

Discussion Time

Recommended Reading

- 'Patterns of Enterprise Application Architecture' *by Martin Fowler*
- CORBA Event Service
  - http://www.omg.org/spec/EVNT/
  - Chapter on CORBA Event Service in 'Instant CORBA' *by Orfali et al*
- MQTT Example Code
  - https://github.com/donnachaforde/example-mqtt
- HiveMQTT
  - https://www.hivemq.com/
- Eclipse Paho Java Client
  - https://www.eclipse.org/paho/clients/java/

Thank You