WHEN SIMPLE API CALL IS NOT ENOUGH

OUTBOX PATTERN

JACEK MILEWSKI

jacek.milewski.k@gmail.com

@jacek_mil

Agenda

1. THE PROBLEM
2. THE OUTBOX PATTERN
3. IMPLEMENTATION
4. TESTING
5. CONCERNS

THE PROBLEM

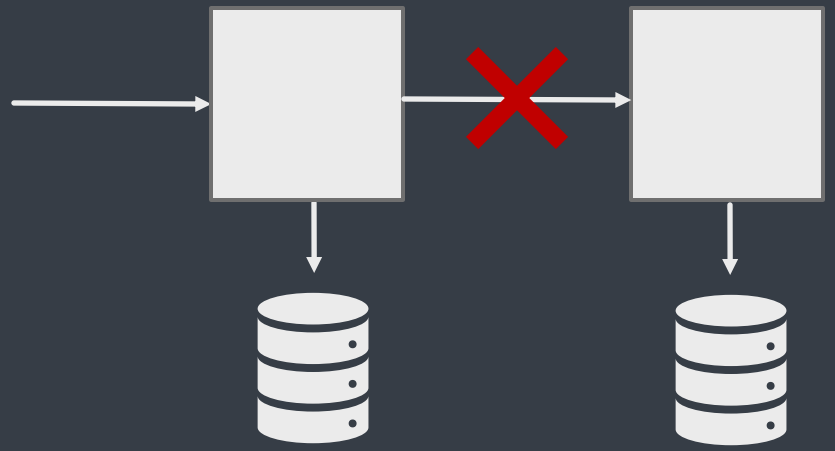
THE PROBLEM

DETAILS

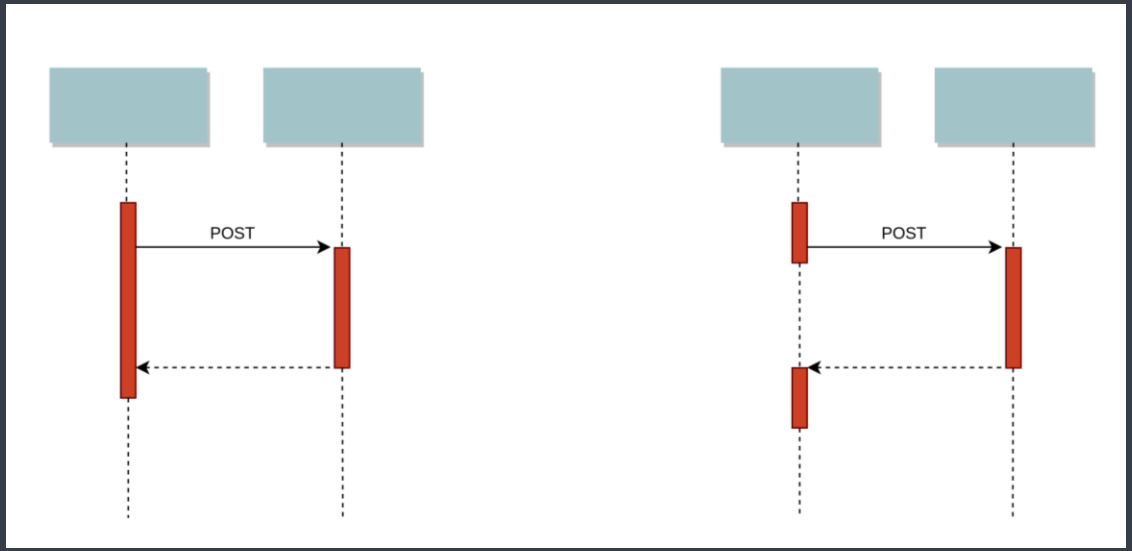
SAVE IT IN DB

AND

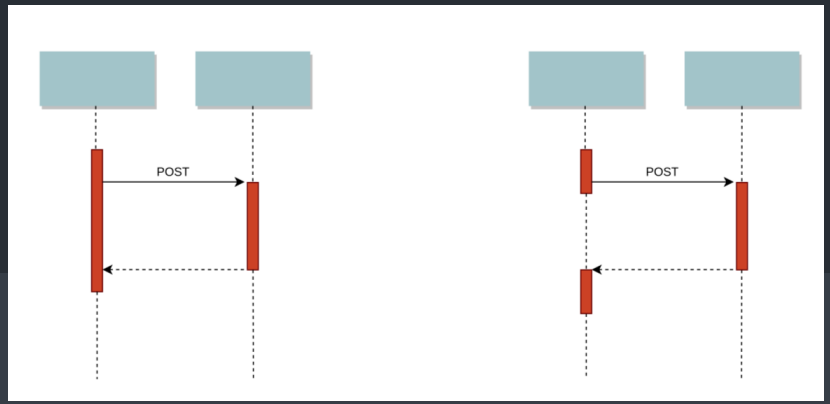
POST A REQUEST TO EXTERNAL API



SYNC? ASYNC?

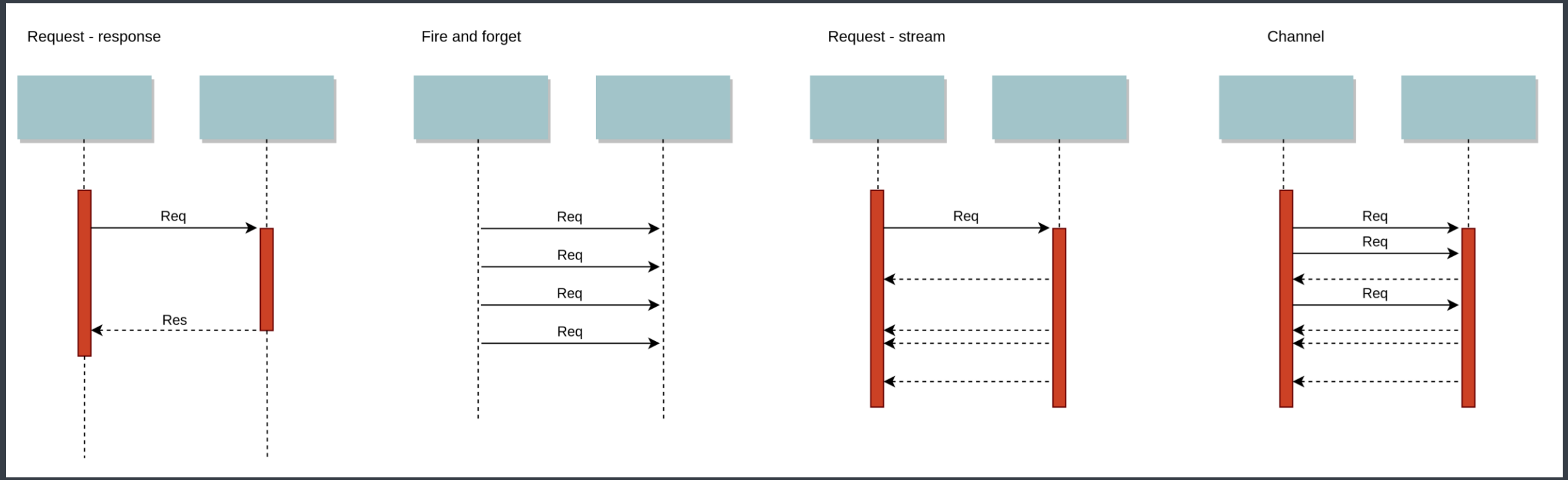


SYNC? ASYNC?



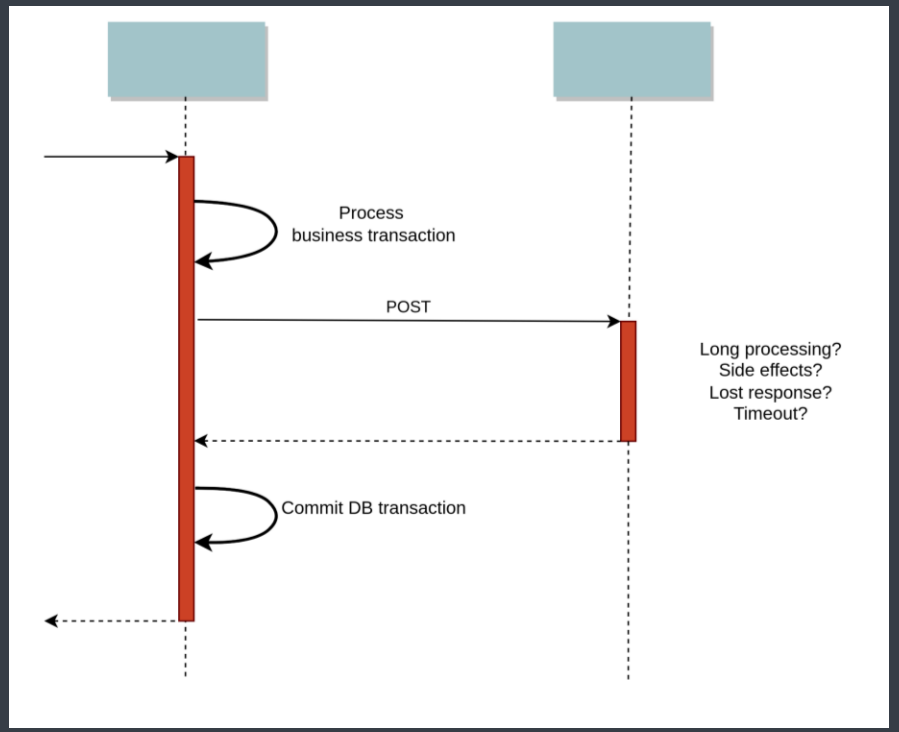
	Sync	async
consistency	Immediate	eventual
Implementation effort	Small	Complex
Processing time	Slower	Same but split to steps
Handling errors / rollbacks	Explicit error responses	Reconciliation needed
Retry mechanism	None	Simpler, depending on a tools

STYLES OF COMMUNICATION

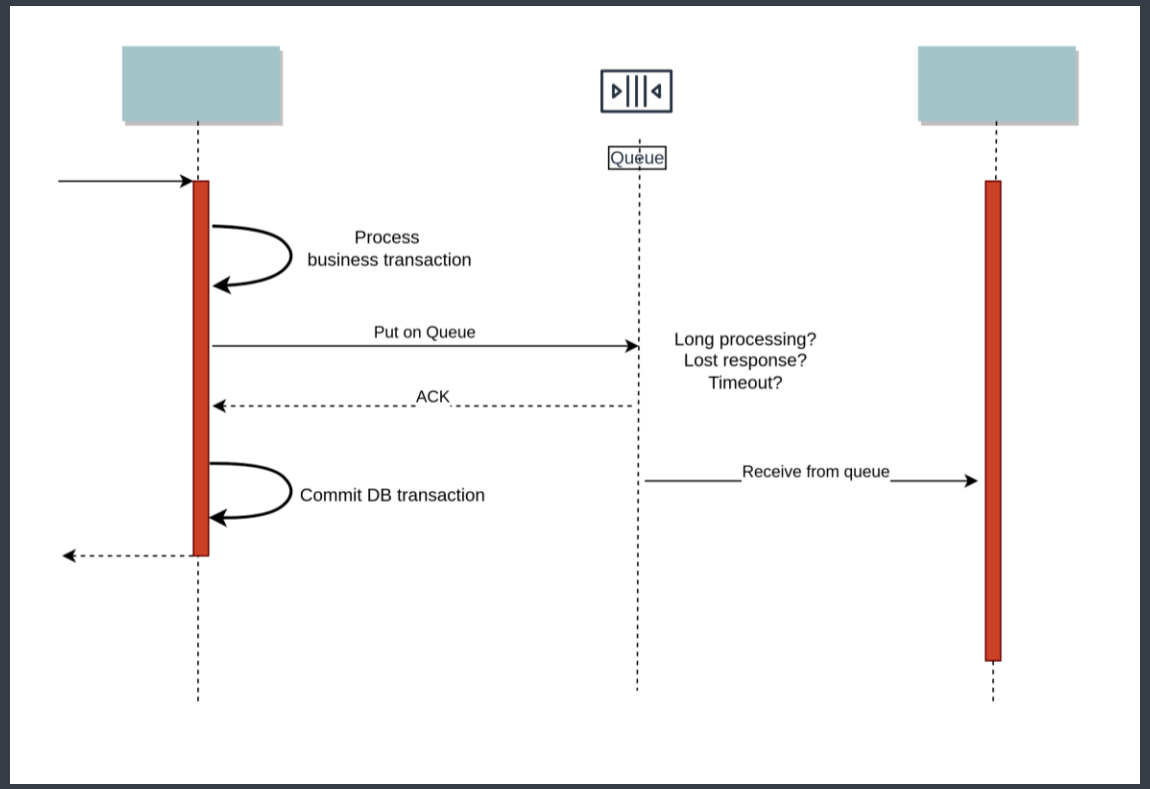


OUTBOX PATTERN

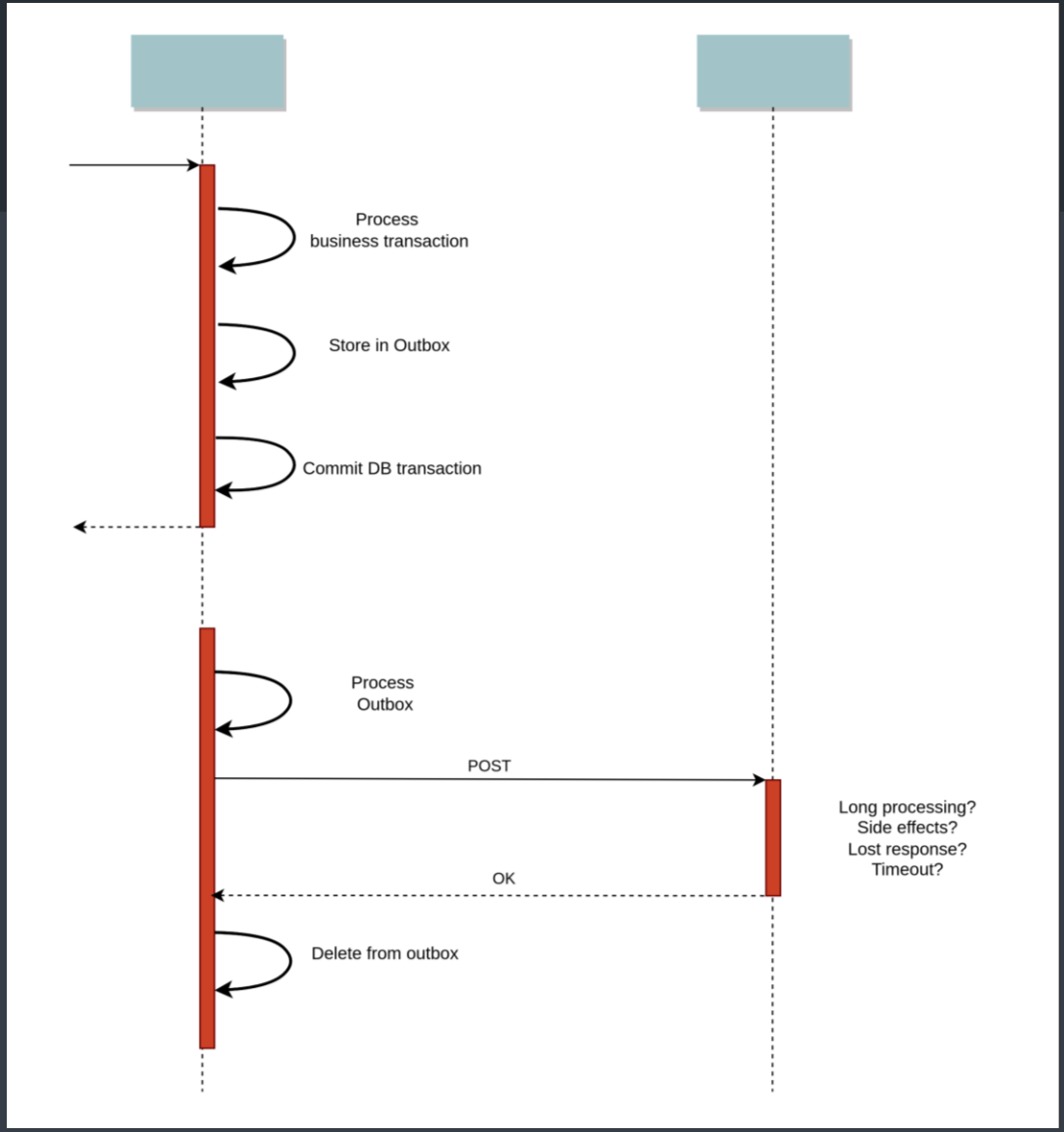
SYNC


Process
business transaction
POST
Long processing?
Side effects?
Lost response?
Timeout?
Commit DB transaction

NAIVE ASYNC


Queue
Process
business transaction
Put on Queue
Long processing?
Lost response?
Timeout?
ACK
Receive from queue
Commit DB transaction

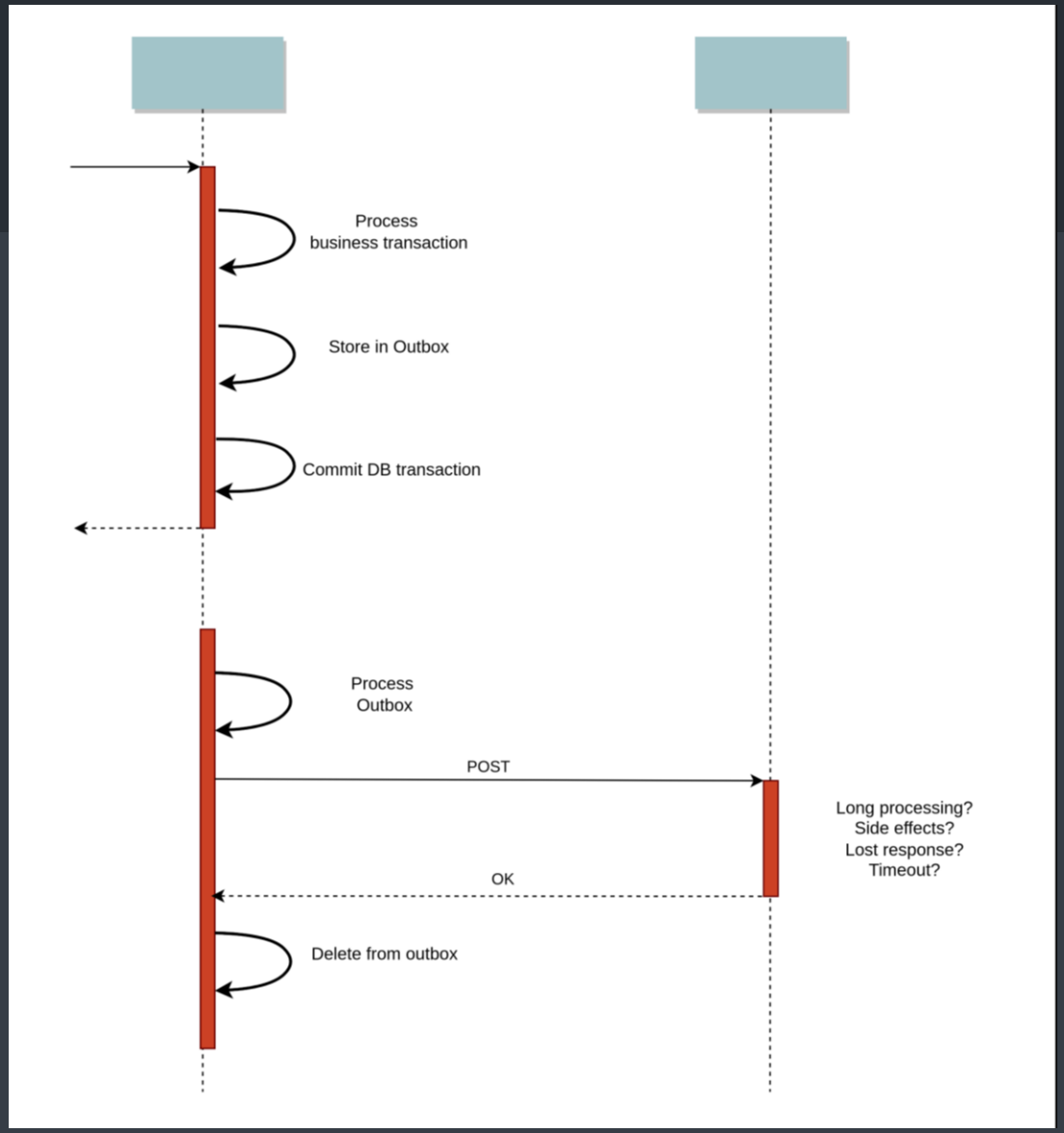
OUTBOX



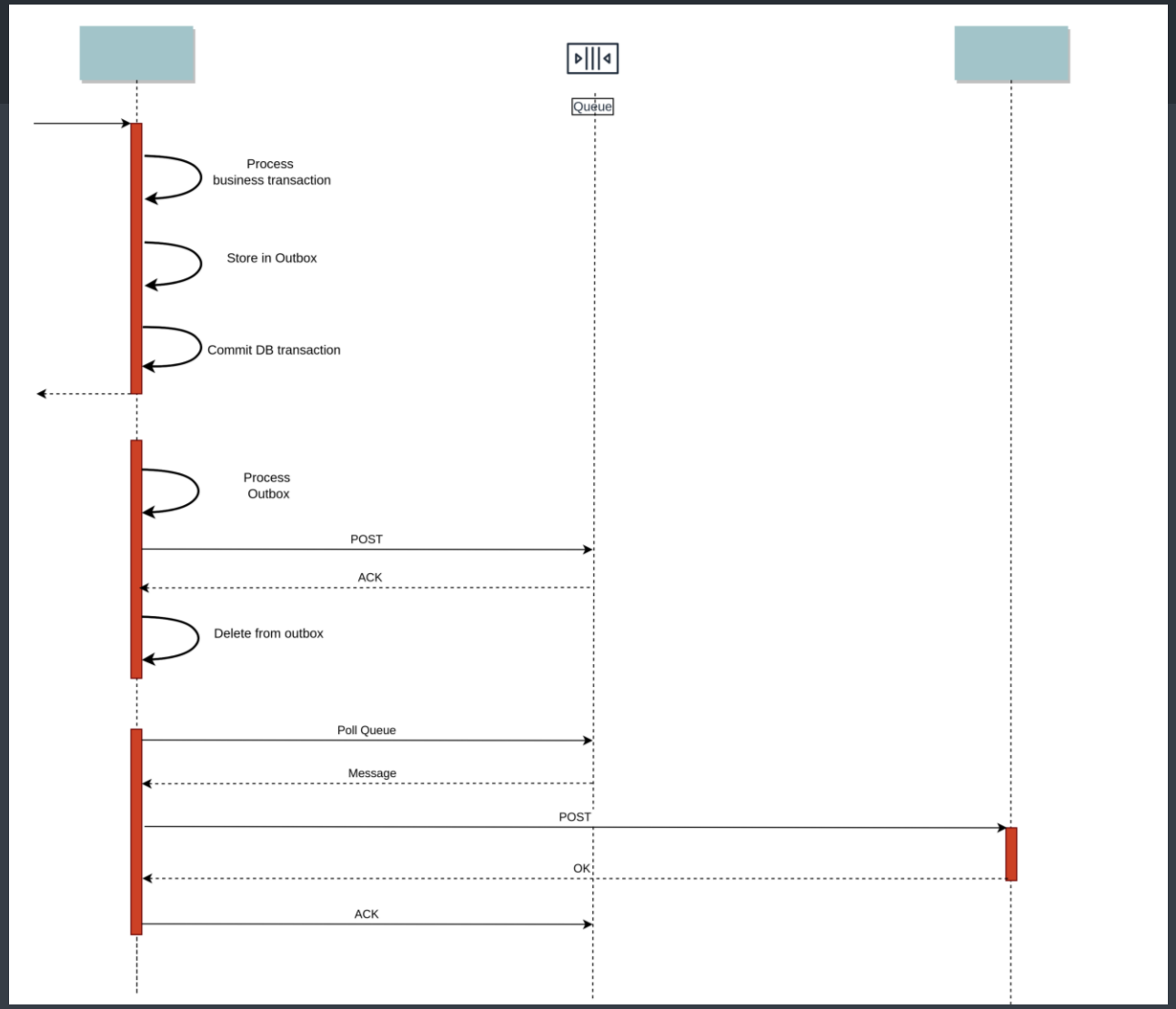
CIRCLE K

OUTBOX

	async
consistency	eventual
Implementation effort	Complex
Processing time	Same but split to steps
Handling errors / rollbacks	Reconciliation needed
Retry mechanism	Simpler, depending on a tools



OUTBOX WITH QUEUE



CIRCLE K

IS THE QUEUE NEEDED?

- Handles retries
  - DLQ Mechanism
- Offloads DB transactions
  - When API requests are slow
  - Reduces the outbox table size
- Adds complexity

WHAT DOES IT SERVE

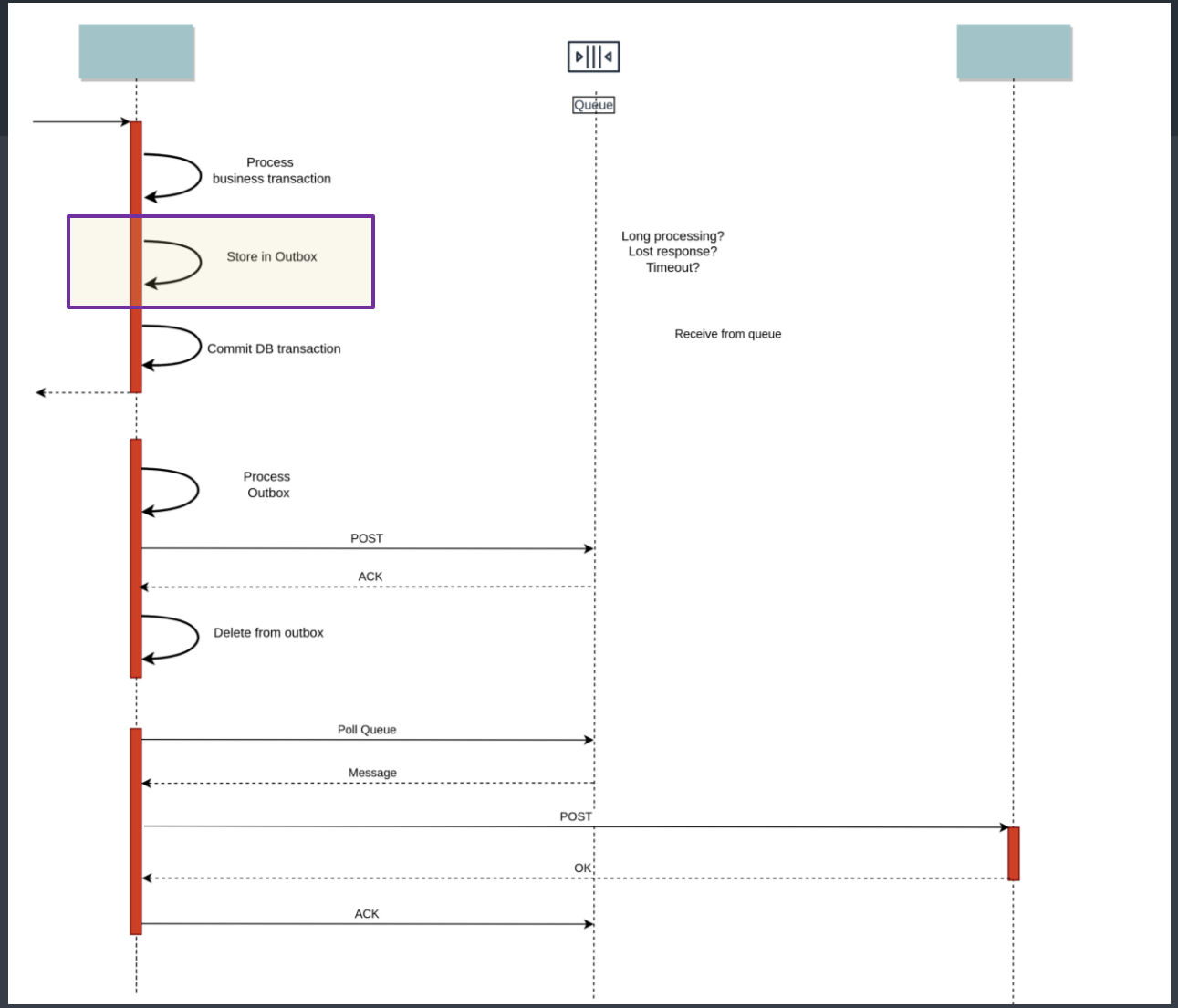
- Async
- Must be consistent, can be eventually consistent
- Request – response
  - With response is handled asynchronously
- Distributed transaction replacement
- At least once delivery

IMPLEMENTATION HOW-TO

OUTBOX EVENTS PROCESSING

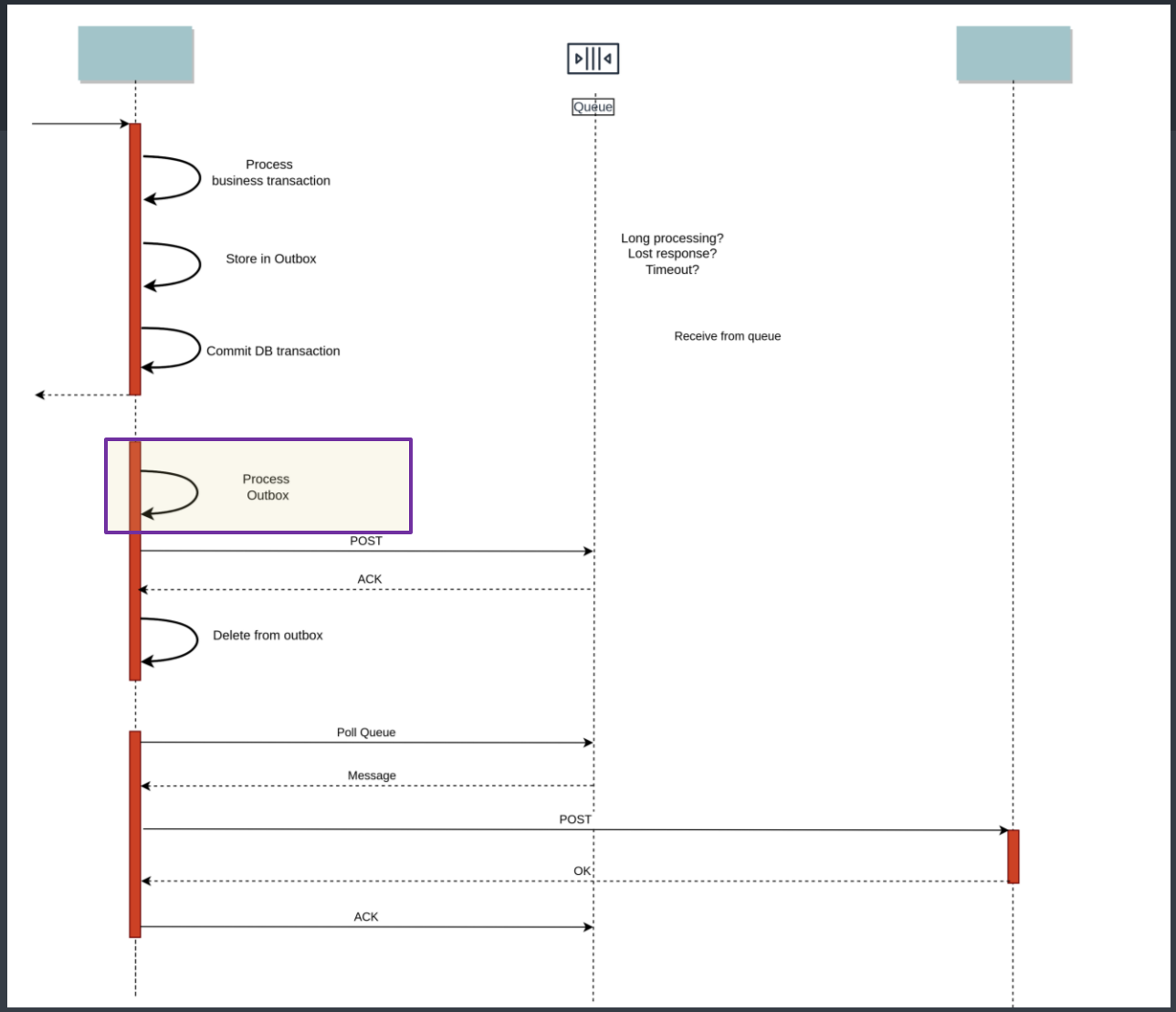
SAVING TO OUTBOX

- WITHIN BUSINESS TRANSACTION
- INCLUDE ALL DETAILS NEEDED AT LATER STEPS
- ORDER
  - UNORDERED
  - GLOBAL ORDER
  - LOCAL ORDER



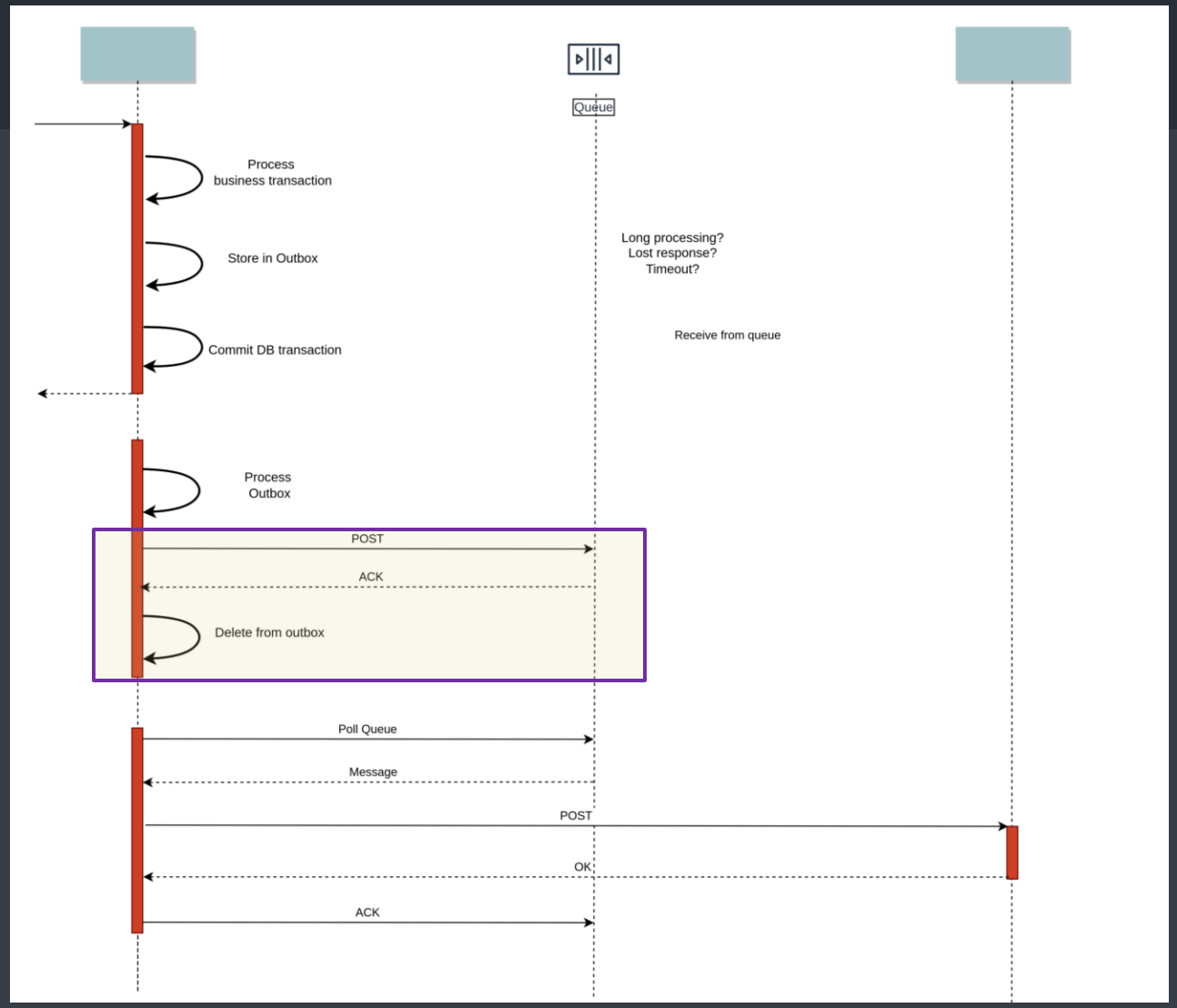
READING OUTBOX

- PERIODICALLY
  - SHEDLOCK, QUARTZ, CRON
  - READER ON SINGLE INSTANCE IN MULTI INSTANCE ENVIRONMENT
  - CHOOSE READING INTERVAL
- AFTER TRANSACTION COMMIT
  - REDUCING DELAY
  - BY A SEPARATE THREAD



SAVING TO QUEUE

- EACH OUTBOX EVENT PROCESSED IN A SEPARATE TRANSACTION
- MIND THE ORDERING
- OFFLOAD DB FROM HEAVY PROCESSING
- AFTER ACK:
  - REMOVE FROM OUTBOX
  - OR UPDATE ITS STATUS

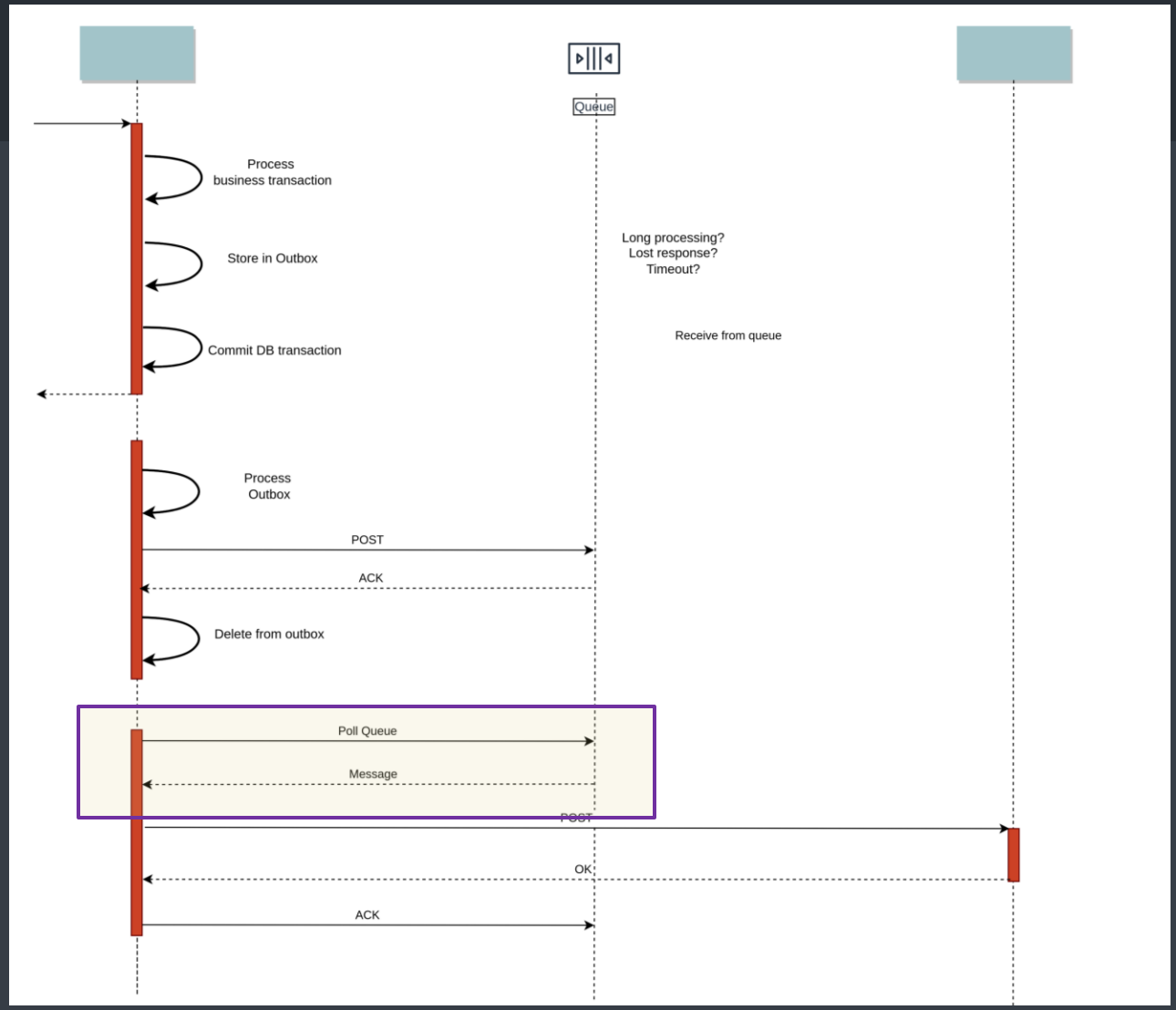


QUEUE CONFIG

- FIFO
- CONTENT BASED DEDUPLICATION
- VISIBILITY TIMEOUT: 60S
- MAX RECEIVE COUNT: 10

READING QUEUE

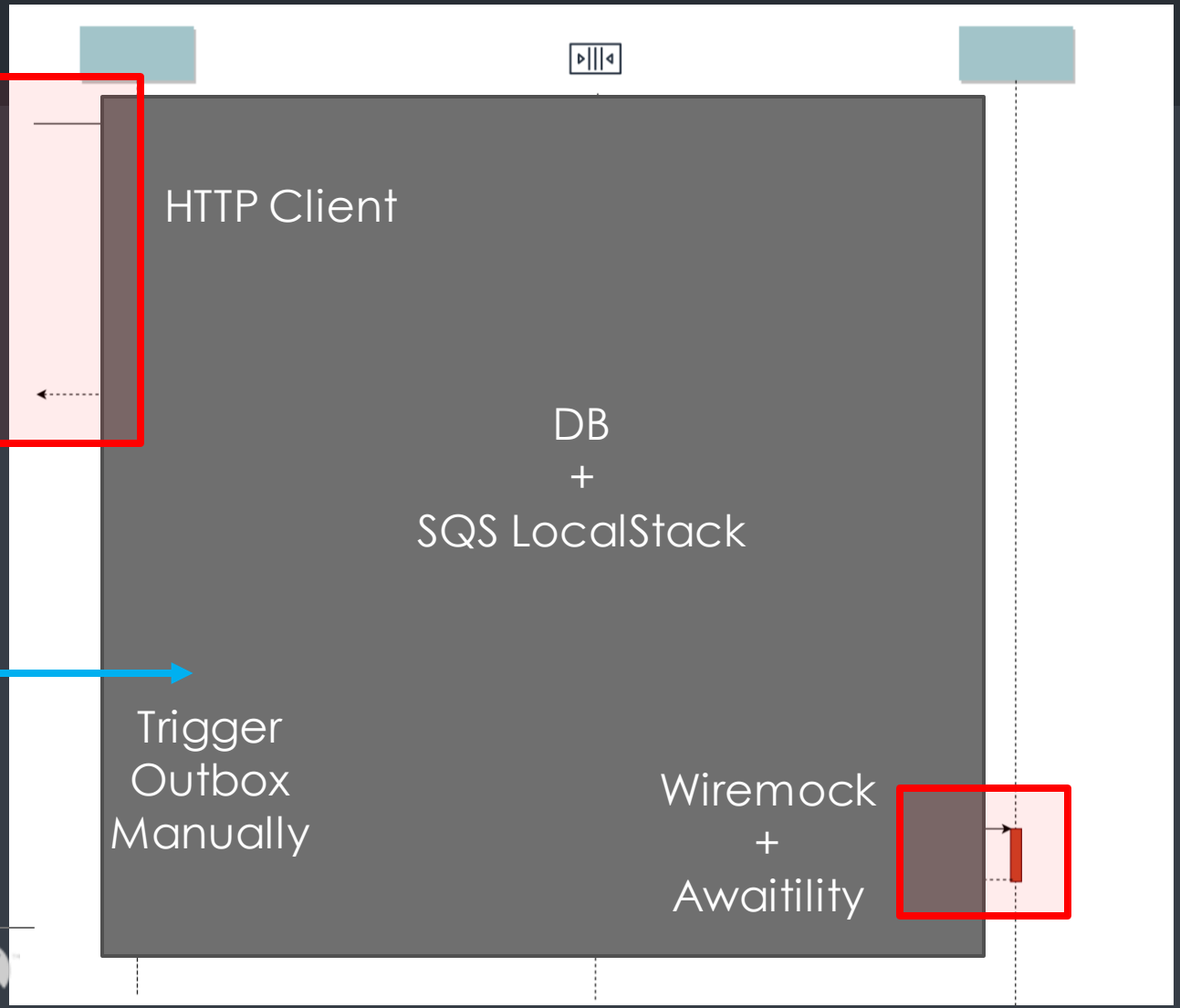
- ORDER
  - MULTIPLE THREADS MIGHT BREAK THE ORDER
- RETRY MECHANISM ON FAILED PROCESSING
  - MESSAGE COMES BACK TO QUEUE WITH VISIBILITY TIMEOUT
  - WHEN FAILED AFTER MAX RETRIES, GOES TO DLQ
- DEAD LETTER QUEUE
  - DLQ EVENTS ARE MONITORED
  - DLQ SUBSCRIBER NOTIFIES CUSTOMER CARE ON MANUAL INTERVENTION REQUIRED
- WHEN API RESPONDS WITH OUR ERROR (4XX)
  - MESSAGE LANDS IN DLQ DIRECTLY



IMPLEMENTATION HOW-TO

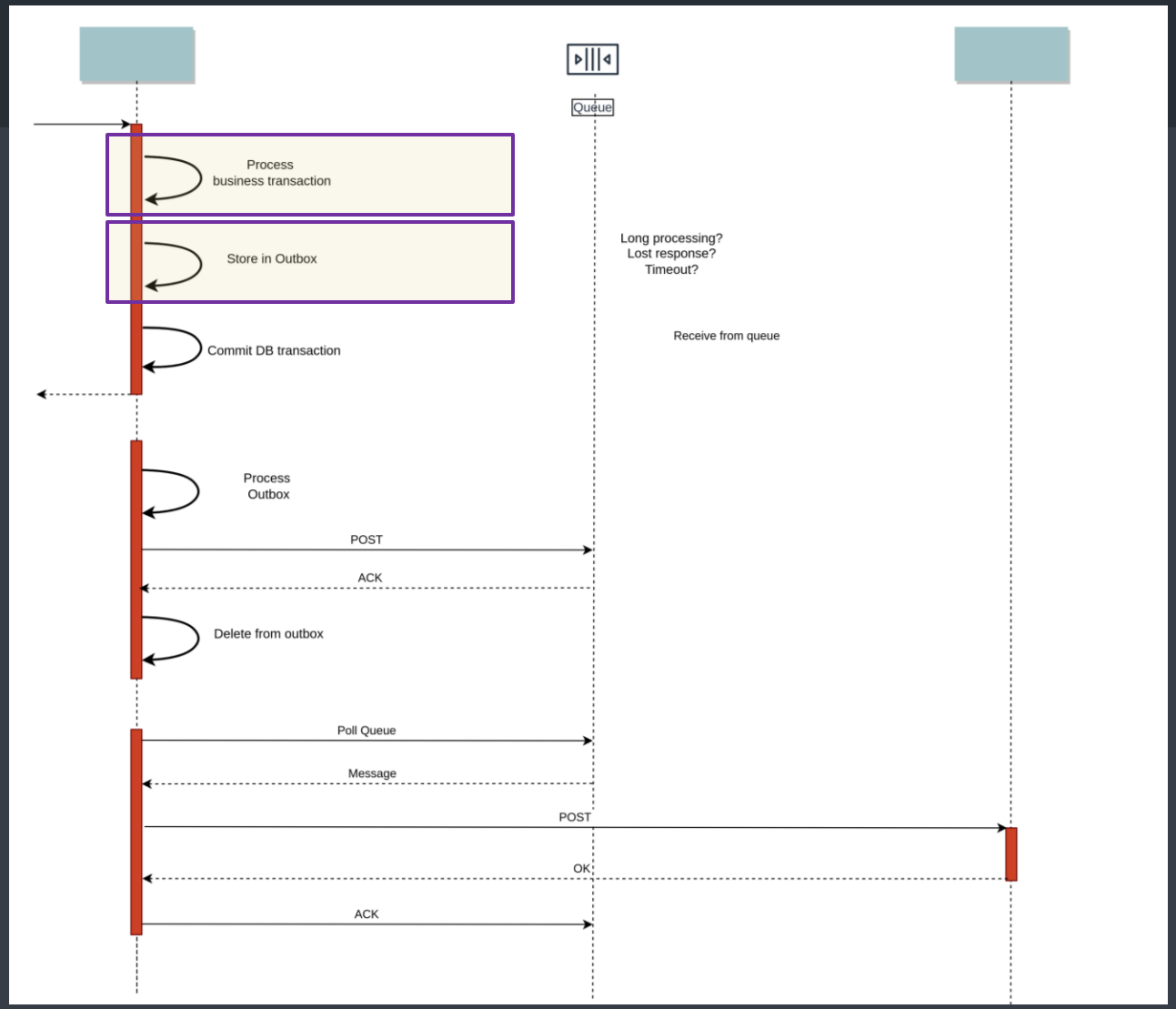
TESTING

INTEGRATION TESTS



UNIT TESTING

- TESTING BUSINESS AND PROCES LOGIC
- NOT VERY SPECIFIC TO OUTBOX

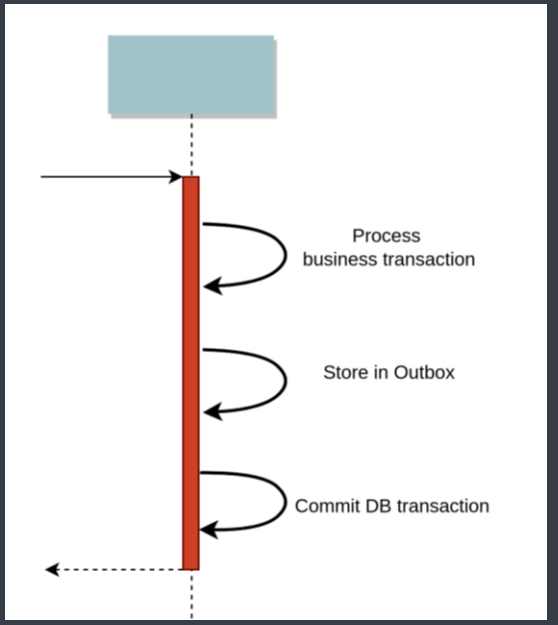


CROSS CUTTING CONCERNS

OUTBOX DATABASE

USE SAME DB AS YOUR PRODUCTION DB

- SQL
  - EVENT IS STORED AS A JSONB TYPE IN PSQL
- NOSQL
  - TRANSACTIONS ARE DIFFERENT BUT THE IDEA OF CONSISTENCY IS THE SAME



VERSIONING EVENTS

OUTBOX AND QUEUE EVENT IS A CONTRACT

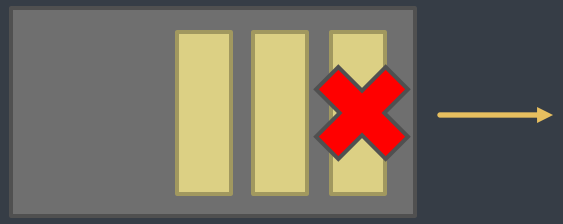
- OLD APP VERSION MIGHT NOT BE ABLE TO PROCESS NEW VERSION
- WHEN CHANGING THE CONTRACT
  - DO BACKWARD COMPATIBLE CHANGES
  - VERSIONING IN EVENTS

```
{
  name: "Jacek"
}
```

```
{
  name: "Jacek",
  status: "BLOCK"
}
```

POISONING MESSAGE

Broken Message at the head of the queue



- Ensure it does not block the whole processing, but just itself
  - Ignore error and continue
- Might happen when:
  - Breaking event syntax compatibility: Postpone it
  - Bug: Drop it to DLQ

OBSERVABILITY

- LOGGING ALL PROCES STEPS
- MONITORING THE METRICS
- TRACING ALL STEPS
  - TRACEID IS PERSISTED THROUGH THE WHOLE PROCESSING

RETRY MECHANISM

At least once delivery

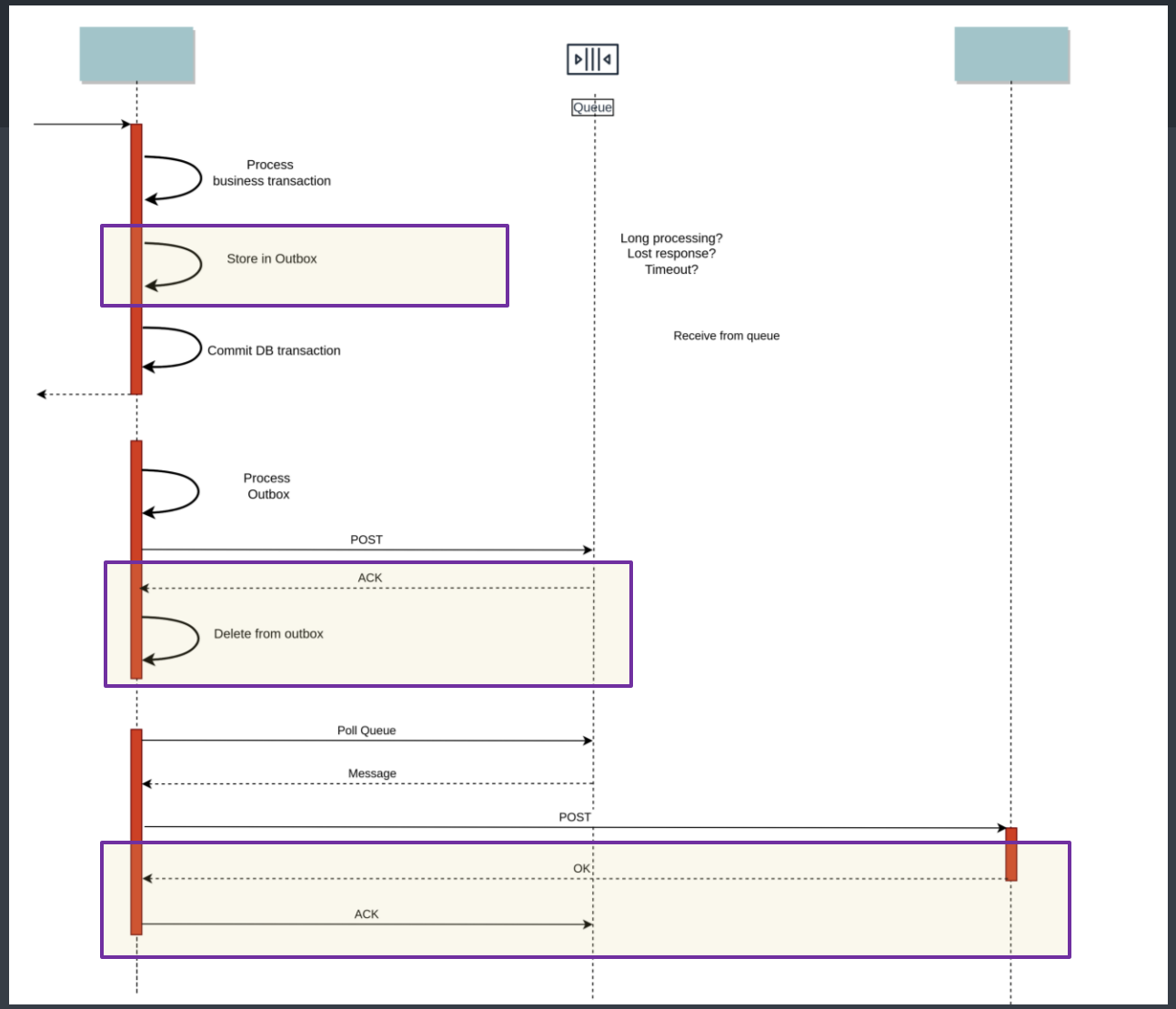
In two flavors:

- Retry reading from outbox
- retry with visibility timeout on SQS

AT LEAST ONCE DELIVERY

DELIVERY GUARANTEED ON EACH STEP

- EVENT IS DELETED ONLY AFTER ACK IS RECEIVED
- DUPLICATION HAPPENS WHEN:
  - ACK WAS LOST
  - DELETE FAILED
  - ON SPRINT DEMO



ORDER

Ordering needs special treatment in async communication

- How strict order is required?
  - Unordered
  - Global order
  - Local order
  - Order of delivery or a presentation order

- Events need to have sequence or timestamp
  - Sourced from central source of truth, e.g. Database
  - Reading from outbox and SQS honors the order
  - One thread at a time can read outbox and process events in an ordering boundaries

OUTBOX PATTERN:

WHEN SIMPLE API CALL IS NOT ENOUGH

THANK YOU!

JACEK MILEWSKI

jacek.milewski.k@gmail.com

@jacek_mil