Temporal @ Yum

Matthew McDole

Pizza Hut

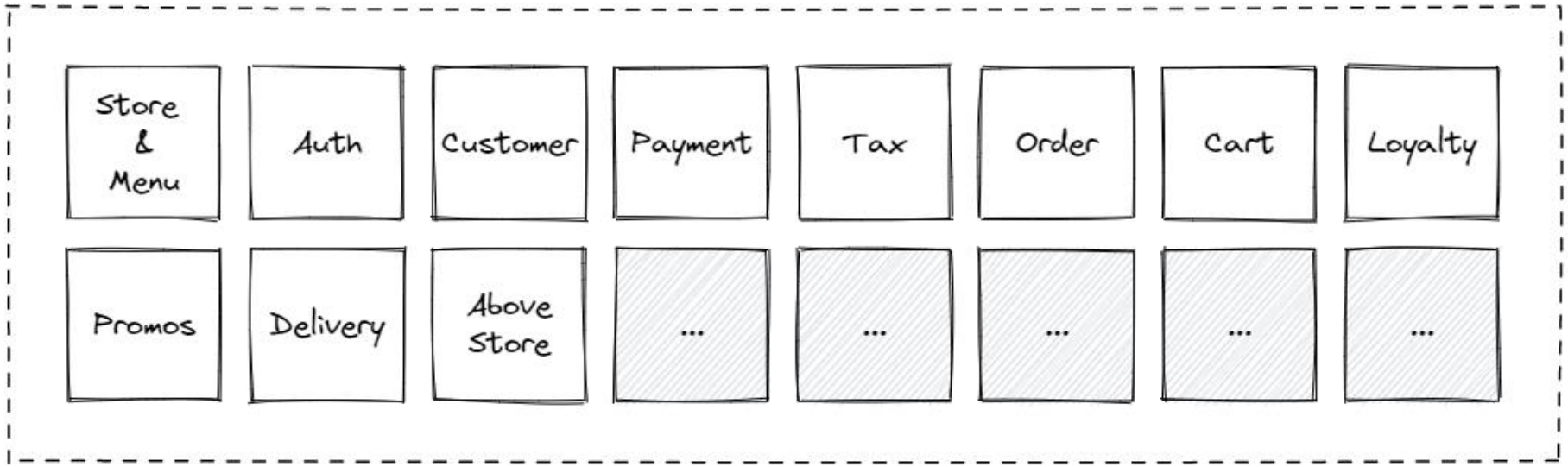
KFC

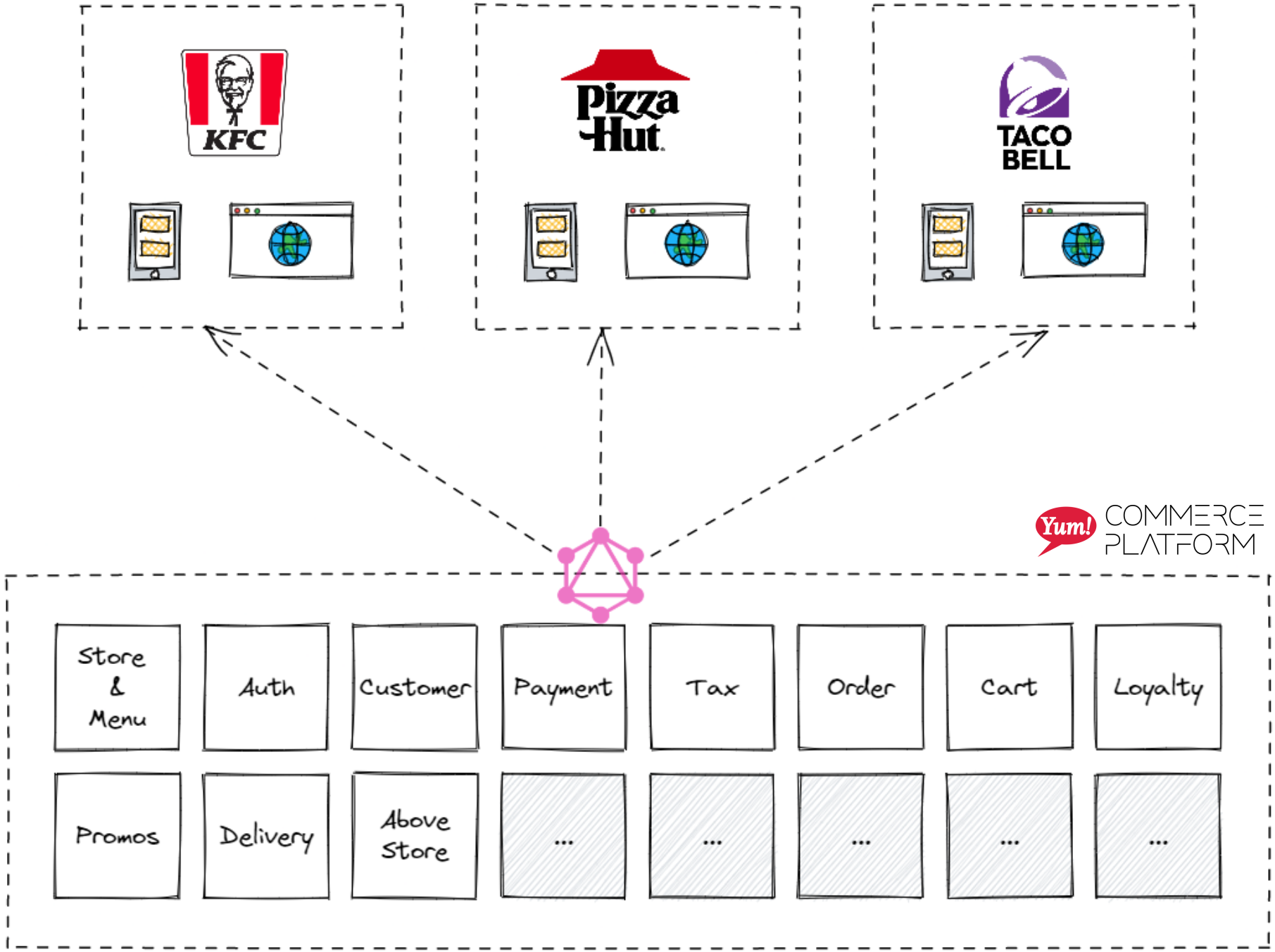
TACO BELL

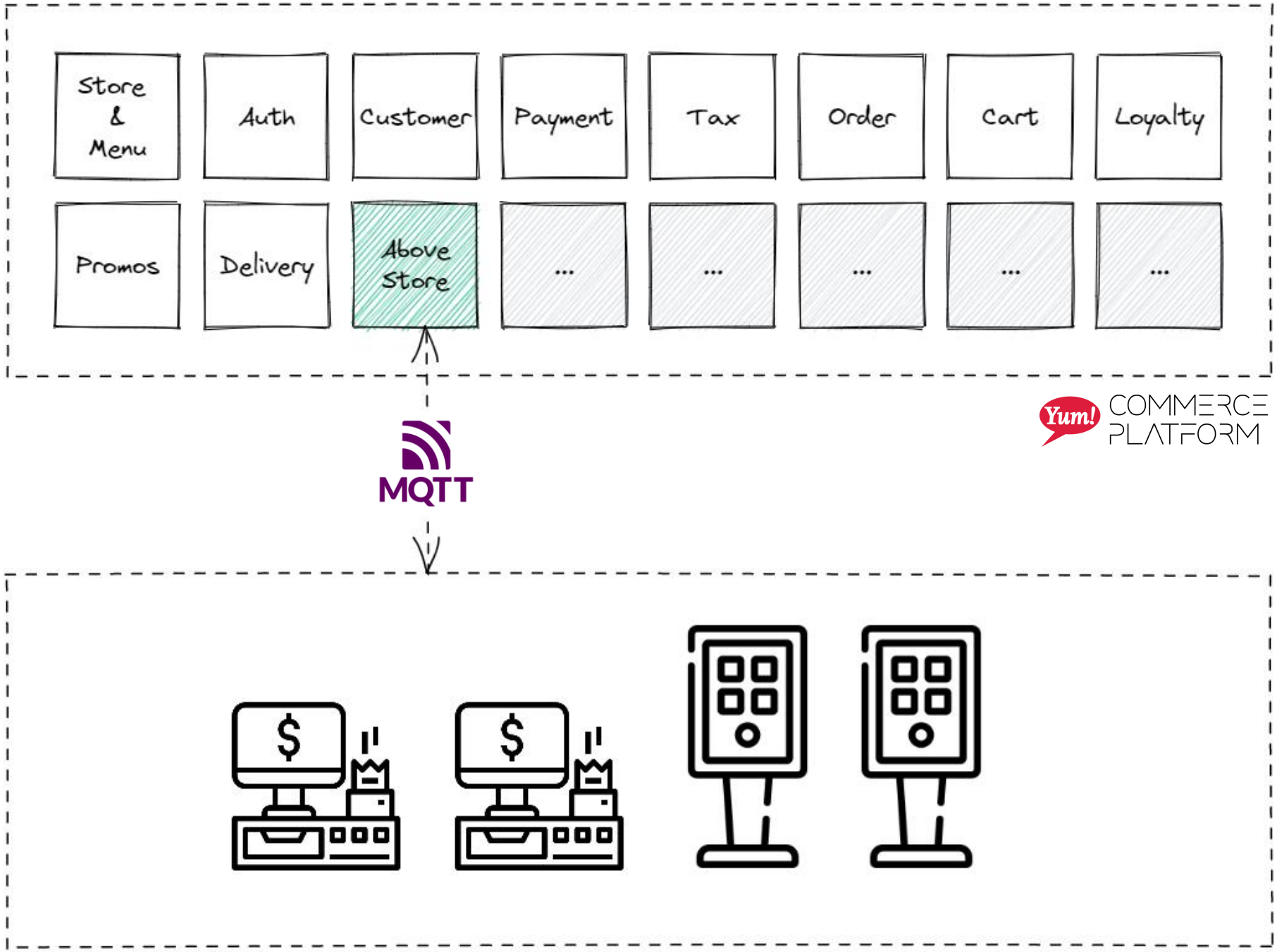
the Habit BURGER GRILL


Yum!
COMMERCE
PLATFORM

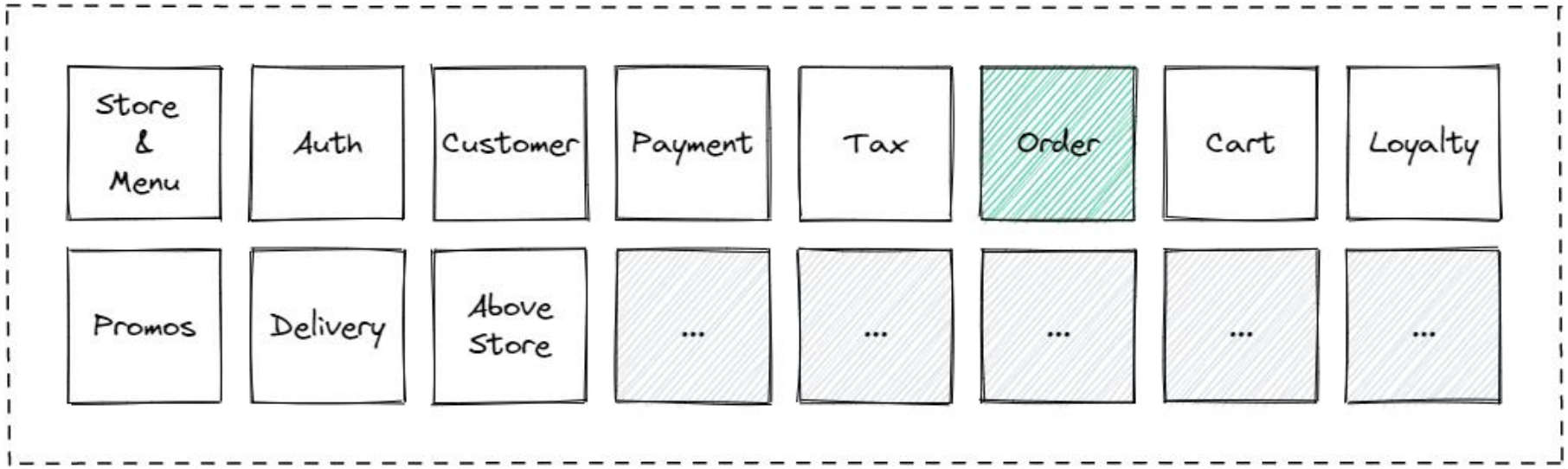
Yum! COMMERCE PLATFORM



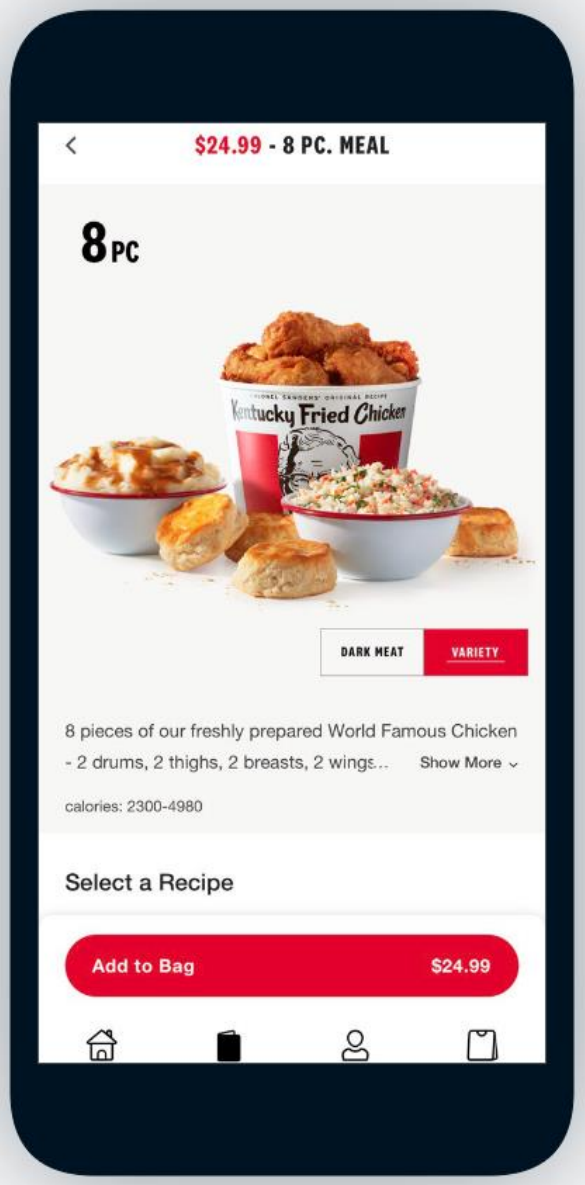

KFC
Pizza Hut
TACO BELL
Yum! COMMERCE PLATFORM
Store & Menu
Auth
Customer
Payment
Tax
Order
Cart
Loyalty
Promos
Delivery
Above Store
...
...
...
...
...

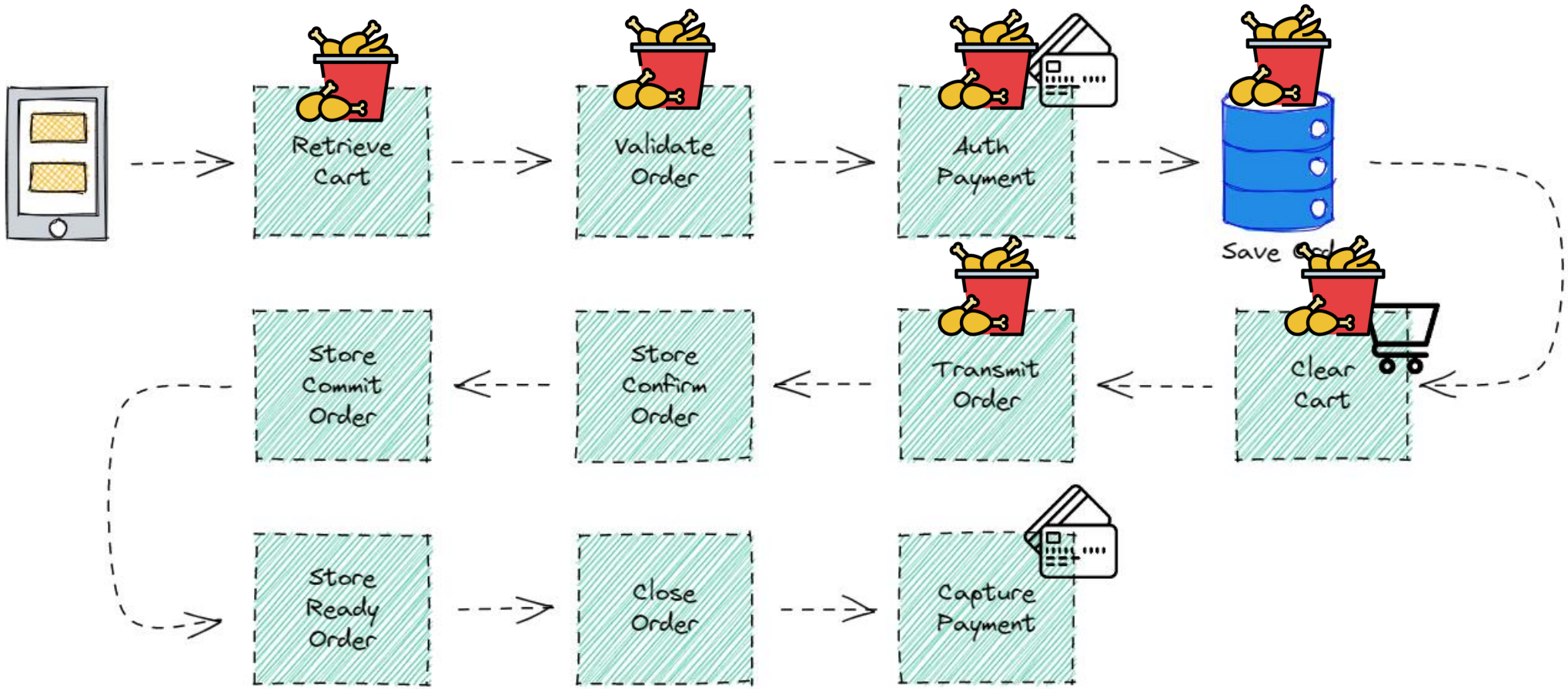

Store & Menu
Auth
Customer
Payment
Tax
Order
Cart
Loyalty
Promos
Delivery
Above Store
...
...
...
...
...
Yum! COMMERCE PLATFORM
MQTT

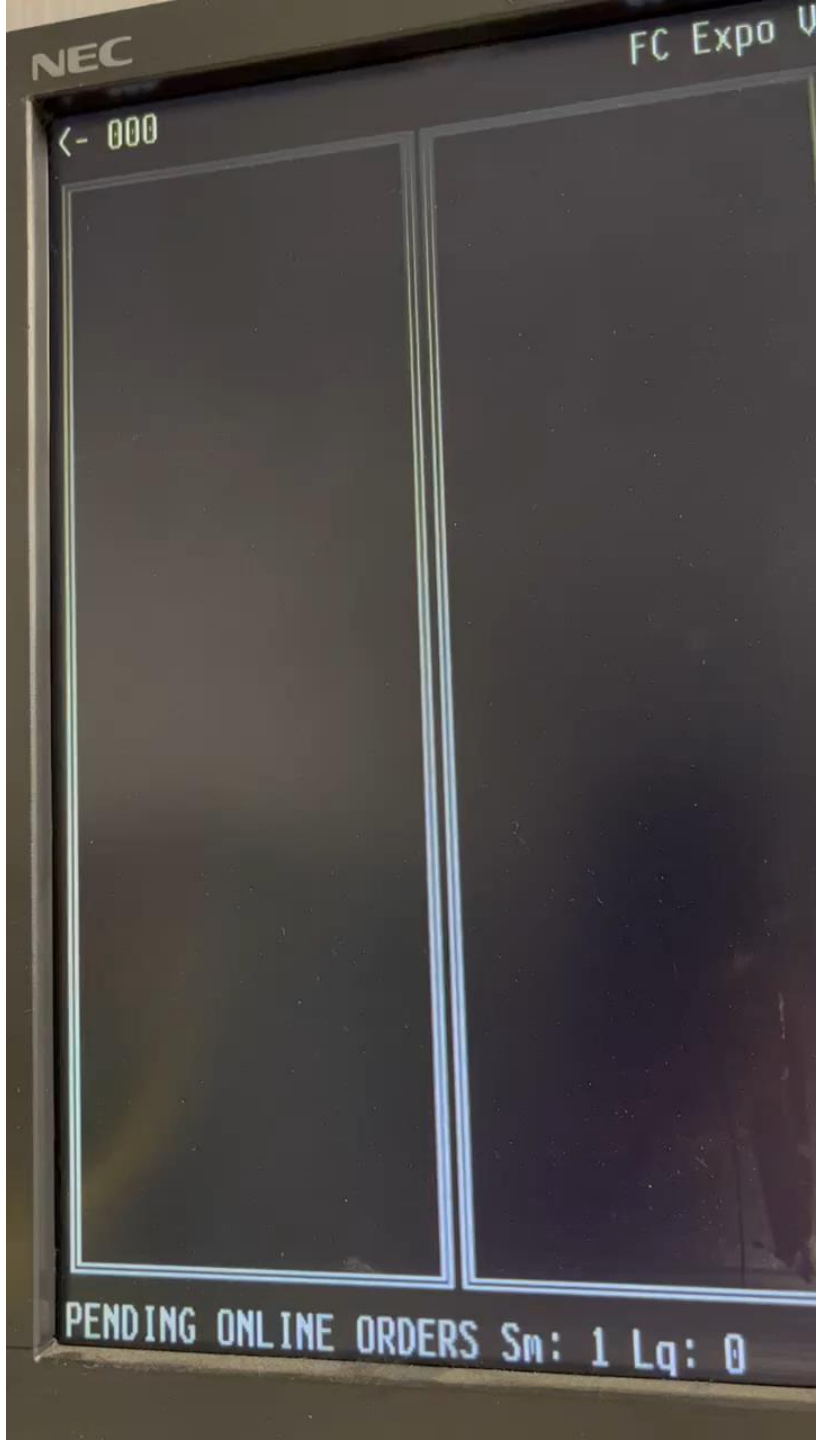
Yum! COMMERCE PLATFORM

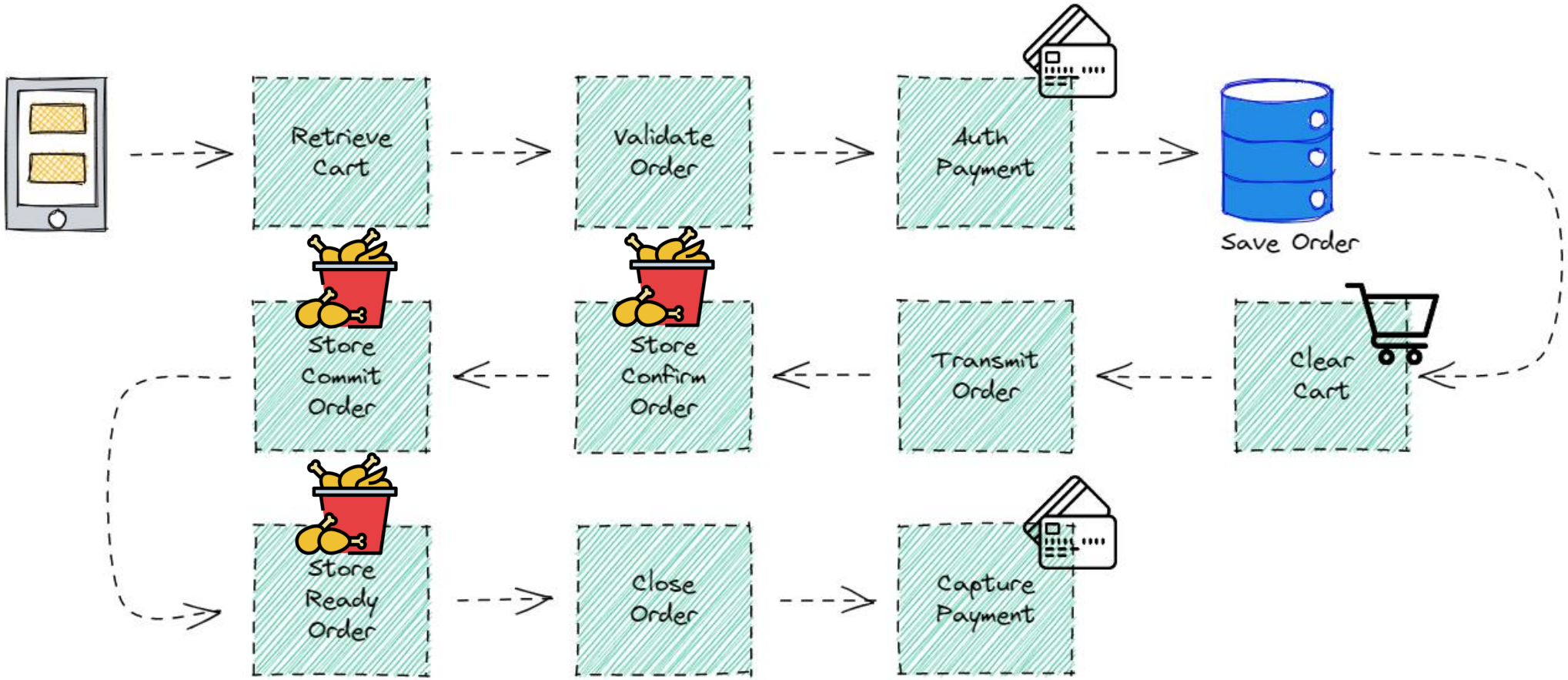


Life of a food Order

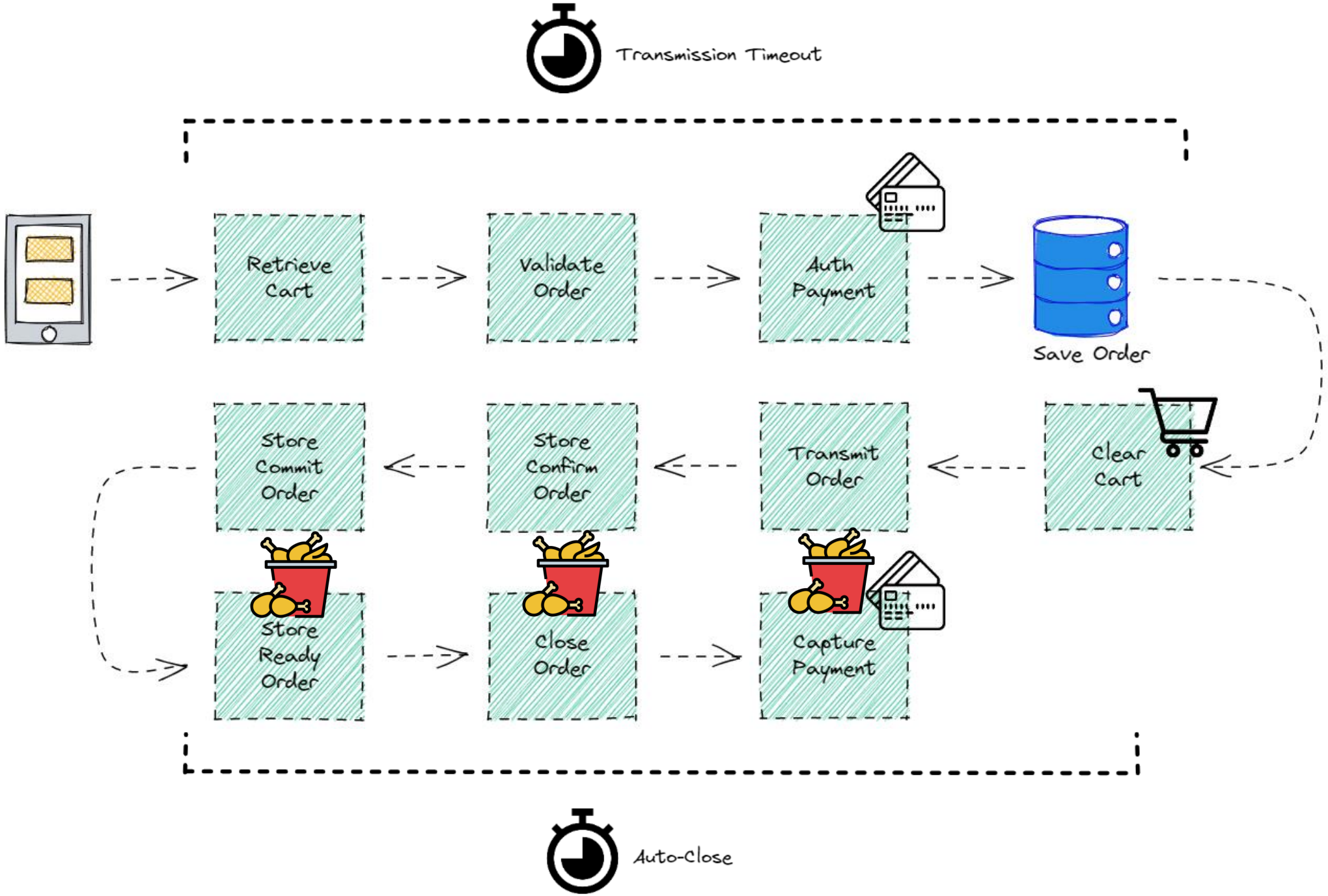

$24.99 - 8 PC. MEAL
8PC
Kentucky Fried Chicken
DARK MEAT
VARIETY
8 pieces of our freshly prepared World Famous Chicken - 2 drums, 2 thighs, 2 breasts, 2 wings...
Show More
calories: 2300-4980
Select a Recipe
Add to Bag
$24.99

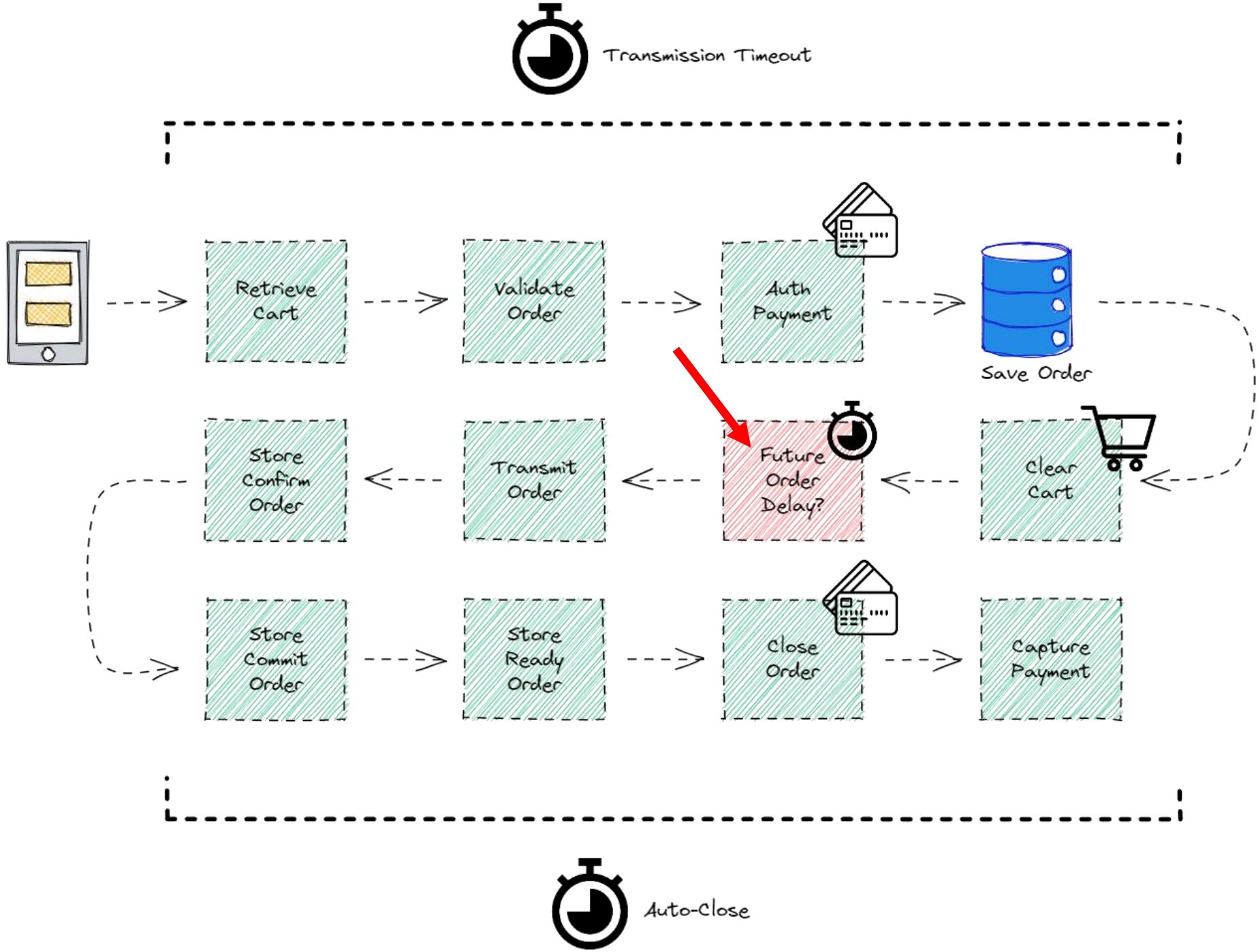

Retrieve Cart
Validate Order
Auth Payment
Save Cart
Clear Cart
Transmit Order
Store Confirm Order
Store Commit Order
Store Ready Order
Close Order
Capture Payment

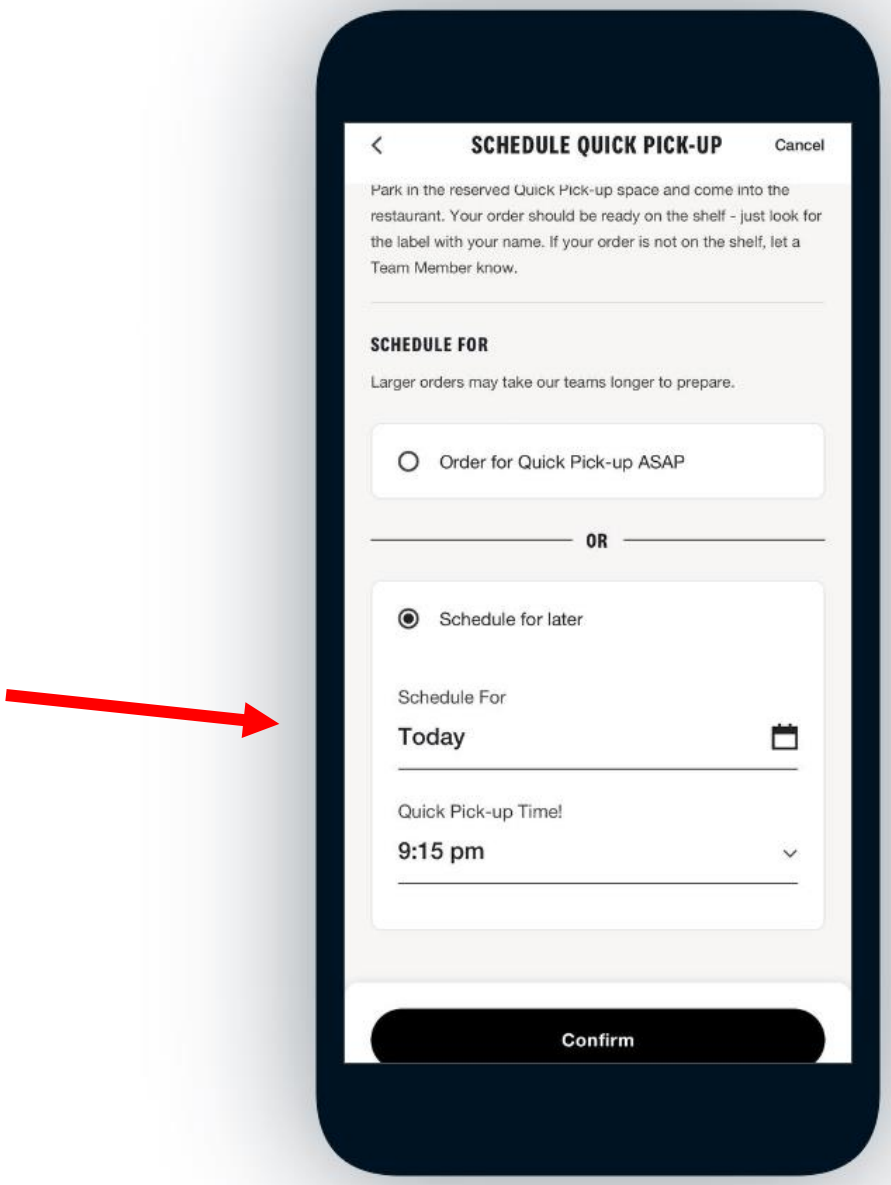

NEC
FC Expo
<- 000
PENDING ONLINE ORDERS Sm: 1 Lq: 0


Retrieve Cart
Validate Order
Auth Payment
Save Order
Clear Cart
Transmit Order
Store Confirm Order
Store Commit Order
Store Ready Order
Close Order
Capture Payment

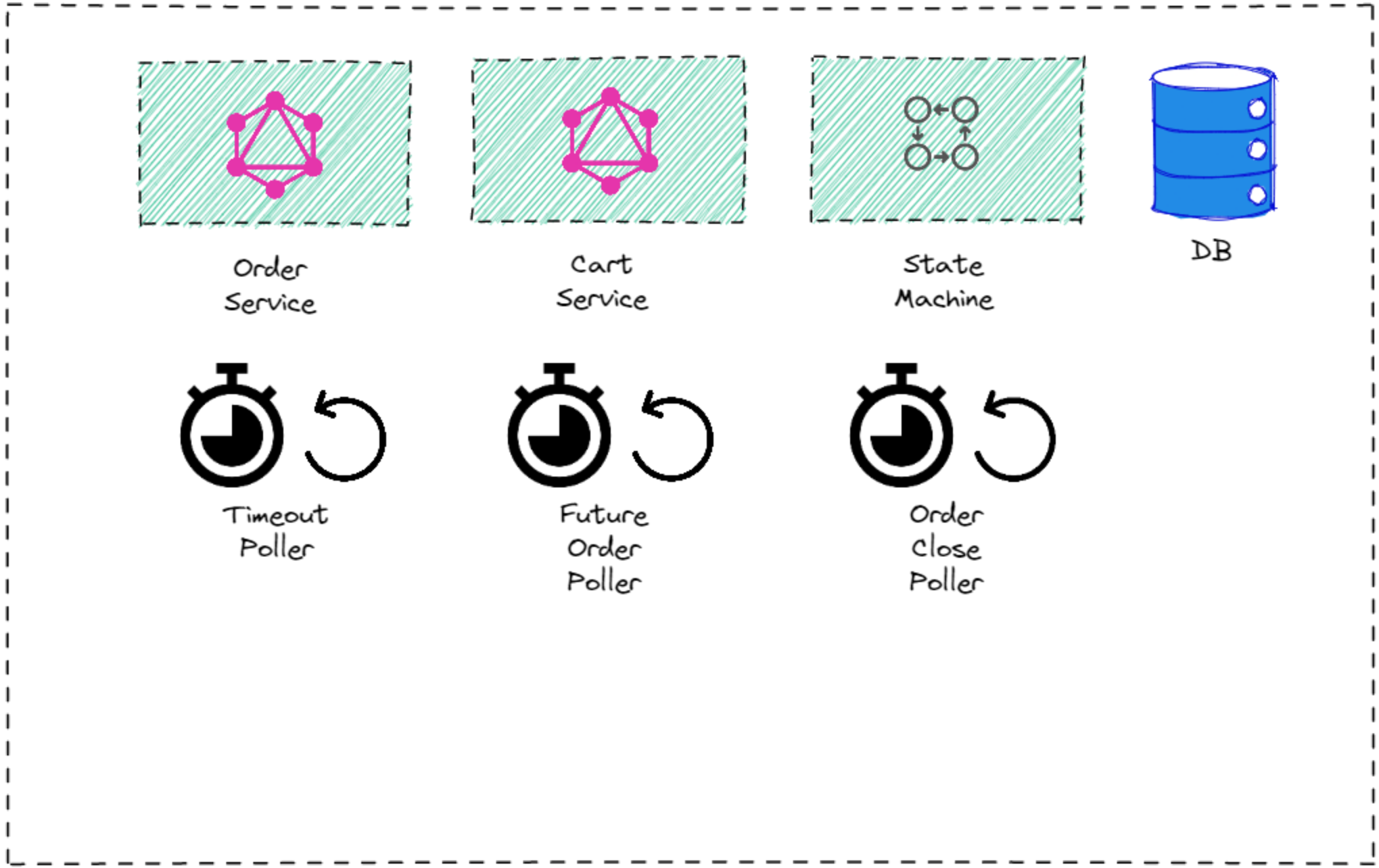

POWER OFF
HENNY PENNY
AHC - 993
SmartHold


Transmission Timeout
Retrieve Cart
Validate Order
Auth Payment
Save Order
Clear Cart
Transmit Order
Store Confirm Order
Store Commit Order
Store Ready Order
Close Order
Capture Payment
Auto-Close

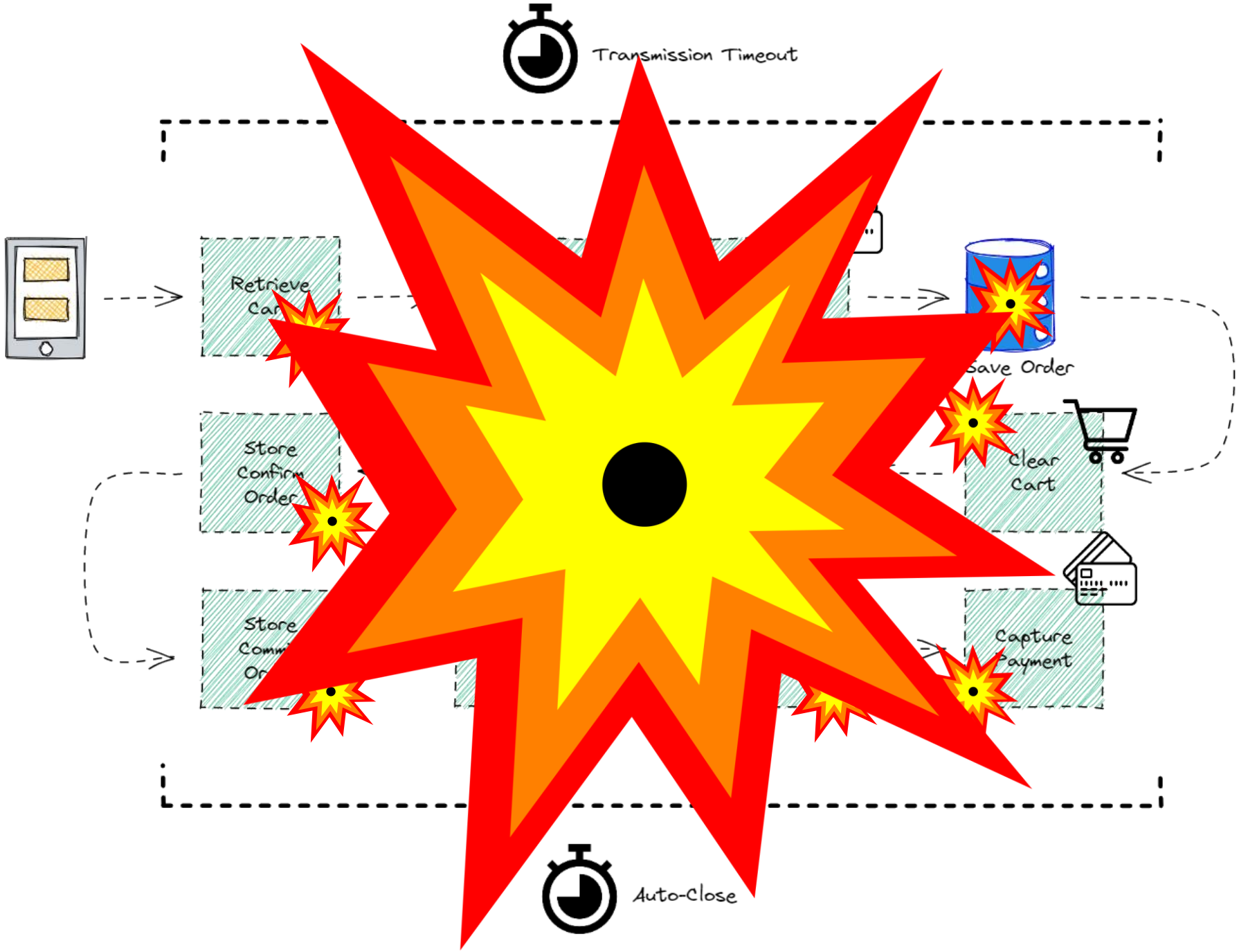

Transmission Timeout
Retrieve Cart
Validate Order
Auth Payment
Save Order
Clear Cart
Future Order Delay?
Transmit Order
Store Confirm Order
Store Commit Order
Store Ready Order
Close Order
Capture Payment
Auto-Close

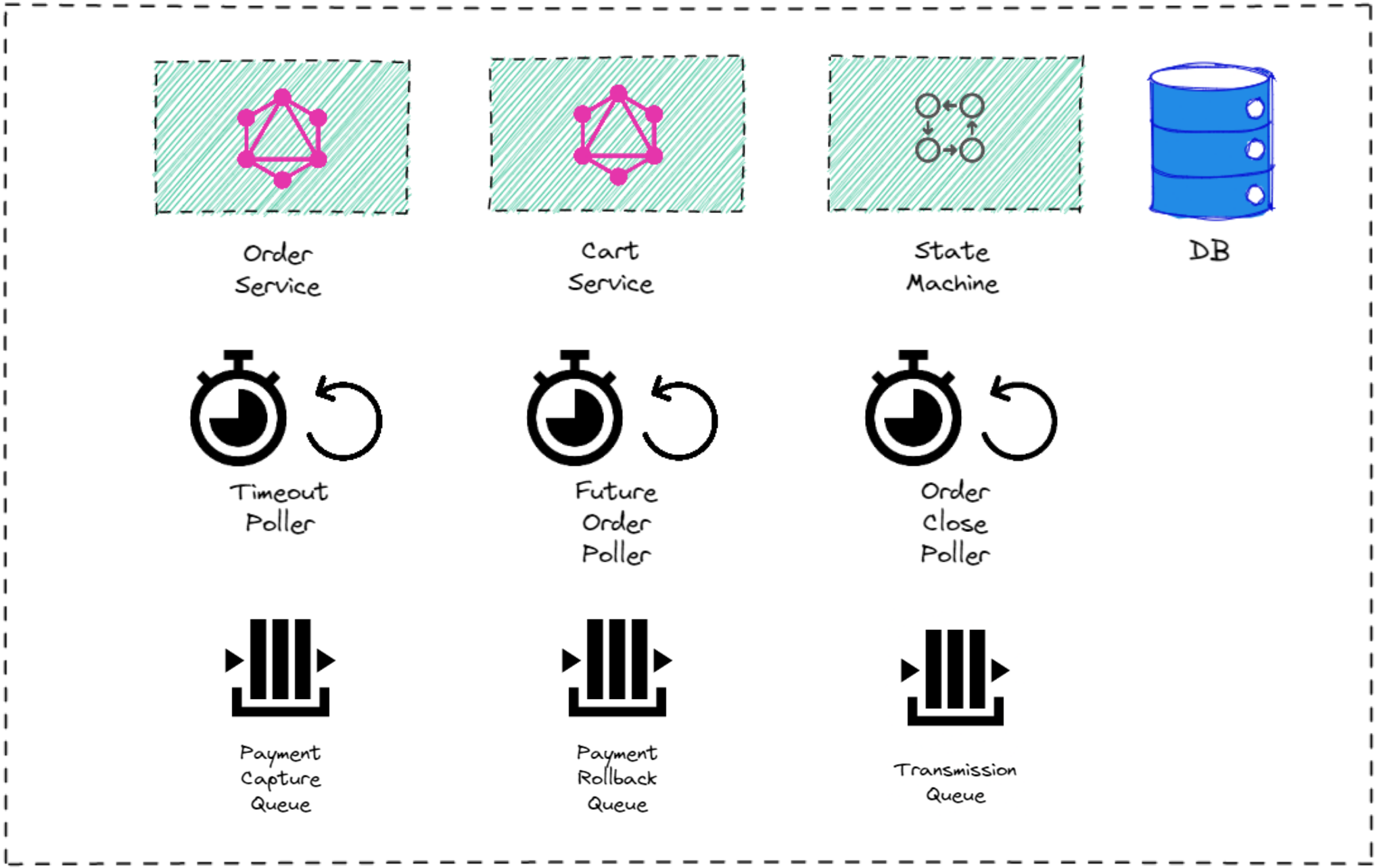

SCHEDULE QUICK PICK-UP
Cancel
Park in the reserved Quick Pick-up space and come into the restaurant. Your order should be ready on the shelf - just look for the label with your name. If your order is not on the shelf, let a Team Member know.
SCHEDULE FOR
Larger orders may take our teams longer to prepare.
Order for Quick Pick-up ASAP
OR
Schedule for later
Schedule For
Today
Quick Pick-up Time!
9:15 pm
Confirm

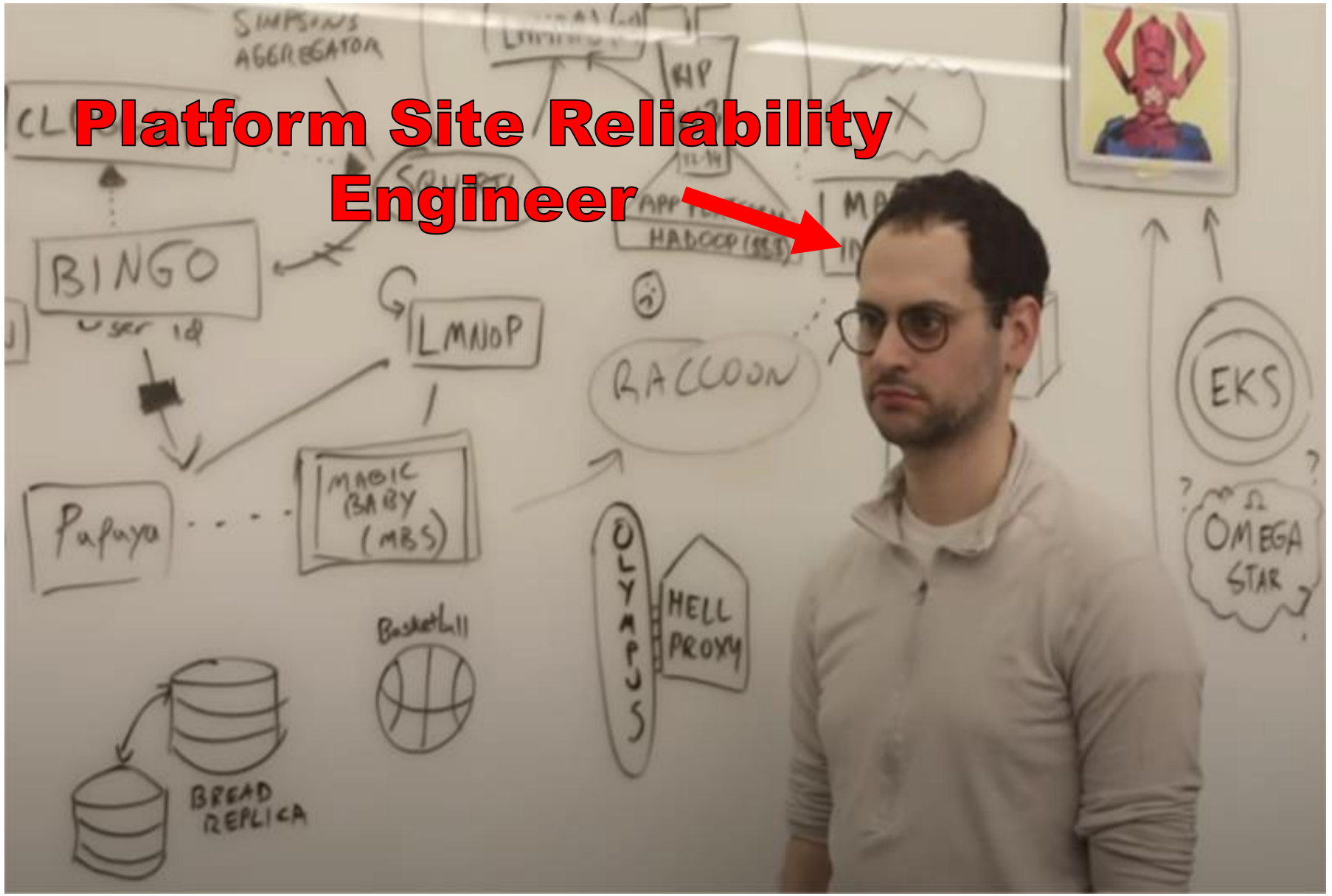

Let’s build it!


Order
Service
Cart
Service
State
Machine
DB
Timeout
Poller
Future
Order
Poller
Order
Close
Poller

What about reliability?


Transmission Timeout
Retrieve
Car
Store
Confirm
Order
Store
Comm
Or
Save Order
Clear
Cart
Capture
Payment
Auto-Close


Order Service
Cart Service
State Machine
DB
Timeout Poller
Future Order Poller
Order Close Poller
Payment Capture Queue
Payment Rollback Queue
Transmission Queue


Platform Site Reliability Engineer
SIMPSONS AGGREGATOR
BINGO
LMNOP
RACCOON
MAGIC BABY (MBS)
Papaya
BREAD REPLICA
OLYMPUS
HELL PROXY
EKS
OMEGA STAR

AWS Step Functions

A large number of use cases span beyond a single request-reply, require tracking of a complex state, respond to asynchronous events, and communicate to external unreliable dependencies.

The usual approach to building such applications is a hodgepodge of stateless services, databases, cron jobs, and queuing systems.

This negatively impacts the developer productivity as most of the code is dedicated to plumbing, obscuring the actual business logic behind a myriad of low-level details. Such systems frequently have availability problems as it is hard to keep all the components healthy.

A large number of use cases span beyond a single request-reply, require tracking of a complex state, respond to asynchronous events, and communicate to external unreliable dependencies.

The usual approach to building such applications is a hodgepodge of stateless services, databases, cron jobs, and queuing systems.

This negatively impacts the developer productivity as most of the code is dedicated to plumbing, obscuring the actual business logic behind a myriad of low-level details. Such systems frequently have availability problems as it is hard to keep all the components healthy.

Cadence

A large number of use cases span beyond a single request-reply, require tracking of a complex state, respond to asynchronous events, and communicate to external unreliable dependencies.

The usual approach to building such applications is a hodgepodge of stateless services, databases, cron jobs, and queuing systems.

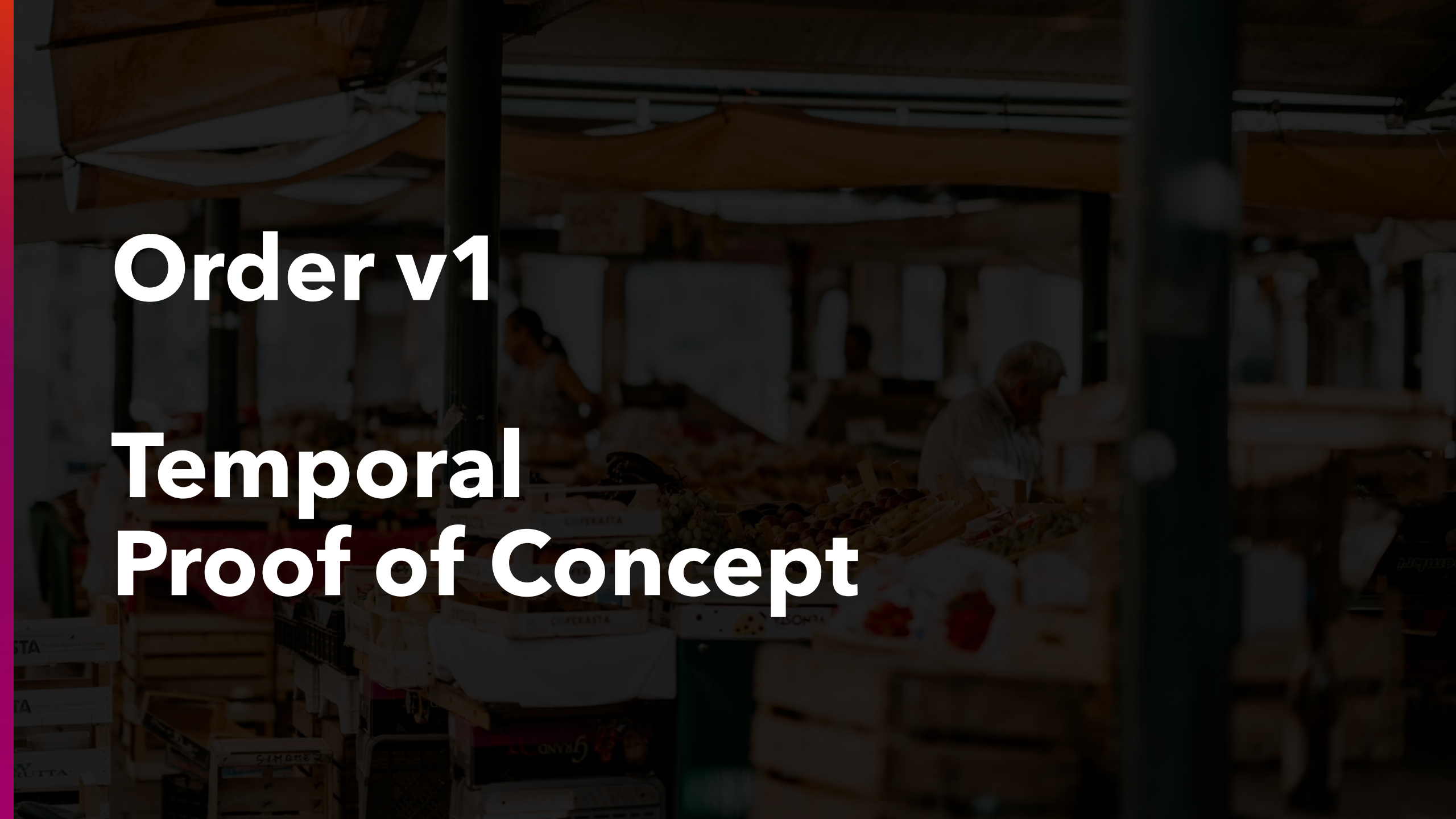
This negatively impacts the developer productivity as most of the code is dedicated to plumbing, obscuring the actual business logic behind a myriad of low-level details. Such systems frequently have availability problems as it is hard to keep all the components healthy.

A large number of use cases span beyond a single request-reply, require tracking of a complex state, respond to asynchronous events, and communicate to external unreliable dependencies.

The usual approach to building such applications is a hodgepodge of stateless services, databases, cron jobs, and queuing systems.

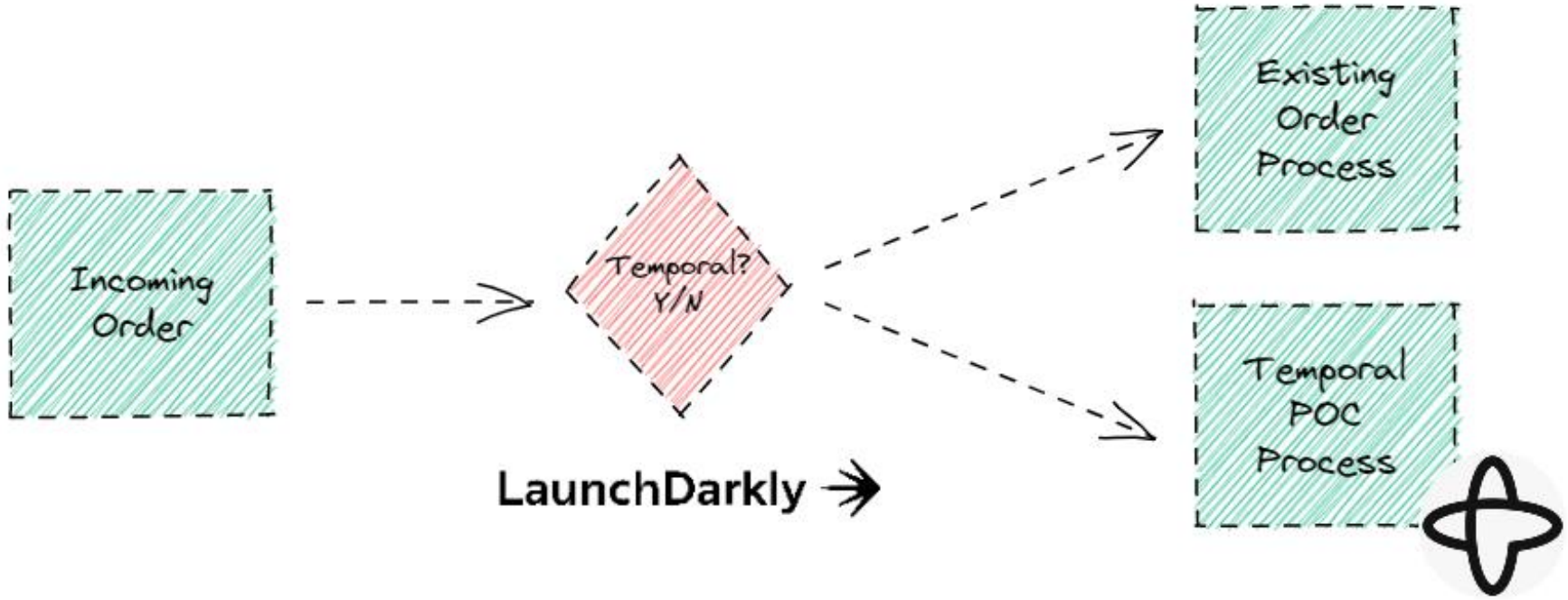
This negatively impacts the developer productivity as most of the code is dedicated to plumbing, obscuring the actual business logic behind a myriad of low-level details. Such systems frequently have availability problems as it is hard to keep all the components healthy.

Cadence



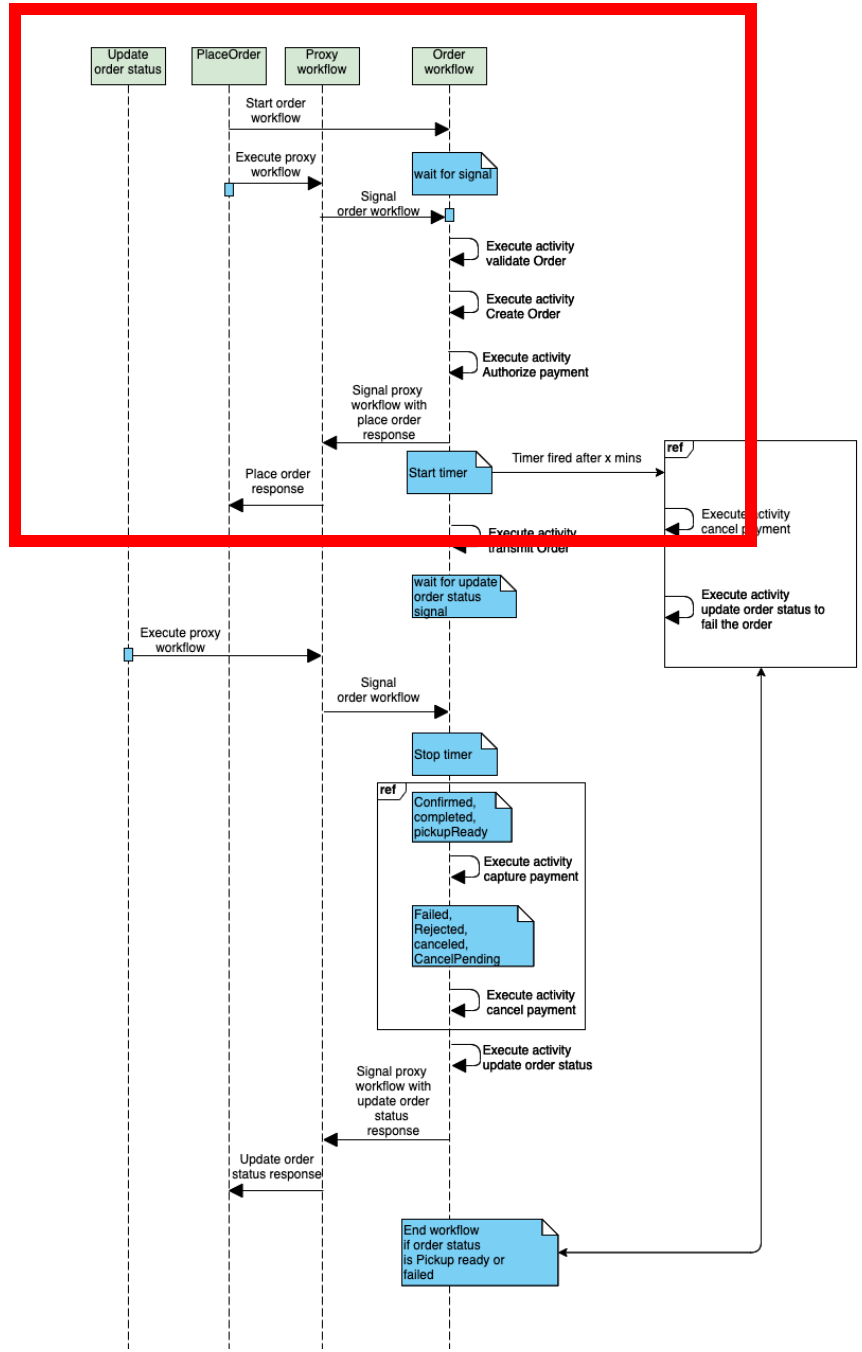
Order v1

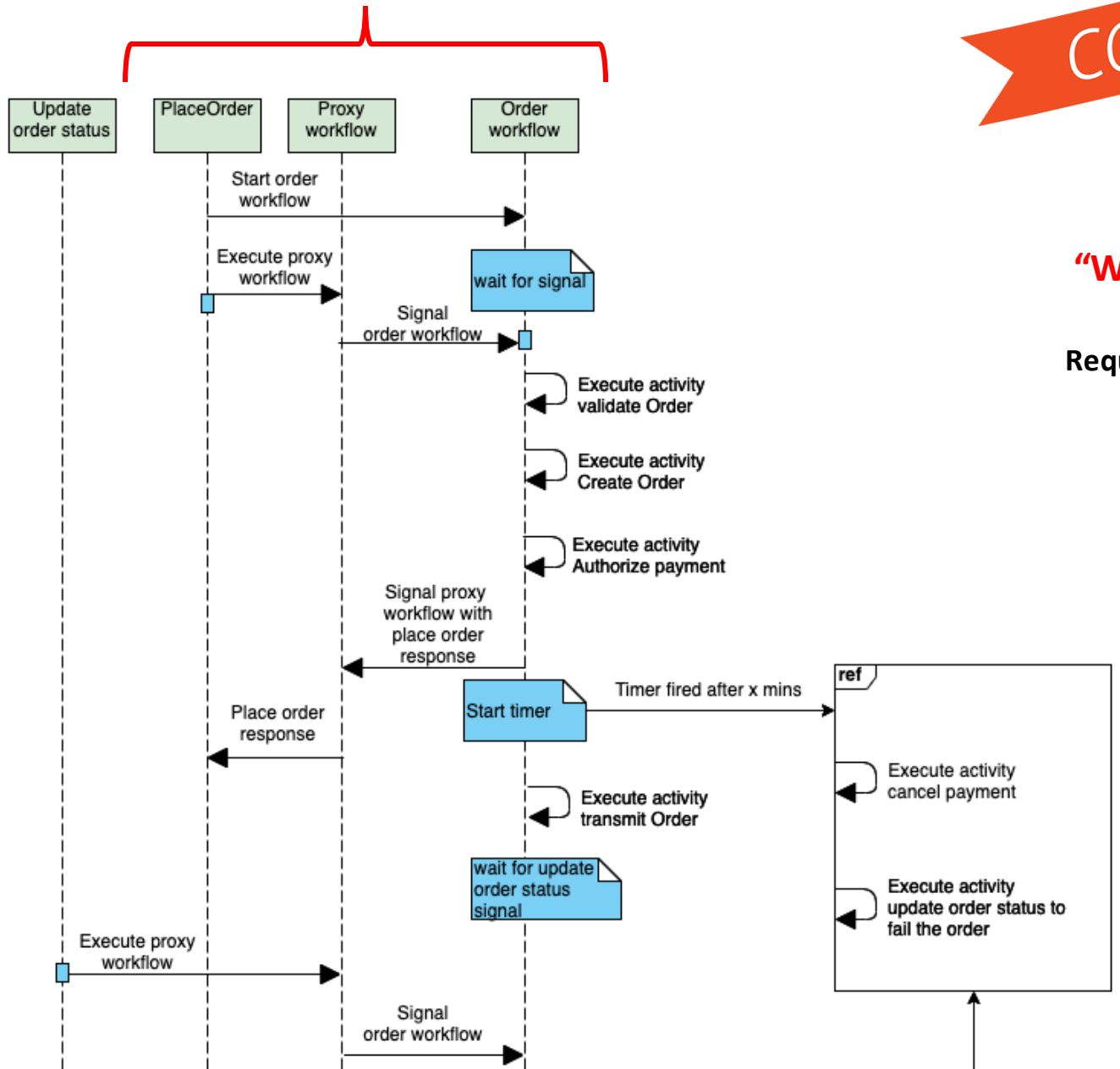
Temporal Proof of Concept


Incoming Order
Temporal? Y/N
LaunchDarkly
Existing Order Process
Temporal POC Process

PoC Lesson #1

Supporting Synchronous Flows


Update order status
PlaceOrder
Proxy workflow
Order workflow
Start order workflow
Execute proxy workflow
wait for signal
Signal order workflow
Execute activity validate Order
Execute activity Create Order
Execute activity Authorize payment
Signal proxy workflow with place order response
Start timer
Timer fired after x mins
ref
Place order response
Execute activity cancel payment
Execute activity transmit Order
wait for update order status signal
Execute activity update order status to fail the order
Execute proxy workflow
Signal order workflow
Stop timer
ref
Confirmed, completed, pickupReady
Execute activity capture payment
Failed, Rejected, canceled, CancelPending
Execute activity cancel payment
Execute activity update order status
Signal proxy workflow with update order status response
Update order status response
End workflow if order status is Pickup ready or failed



"Workflow Update"

Request / Response in one Synchronous call

PoC Lesson #2

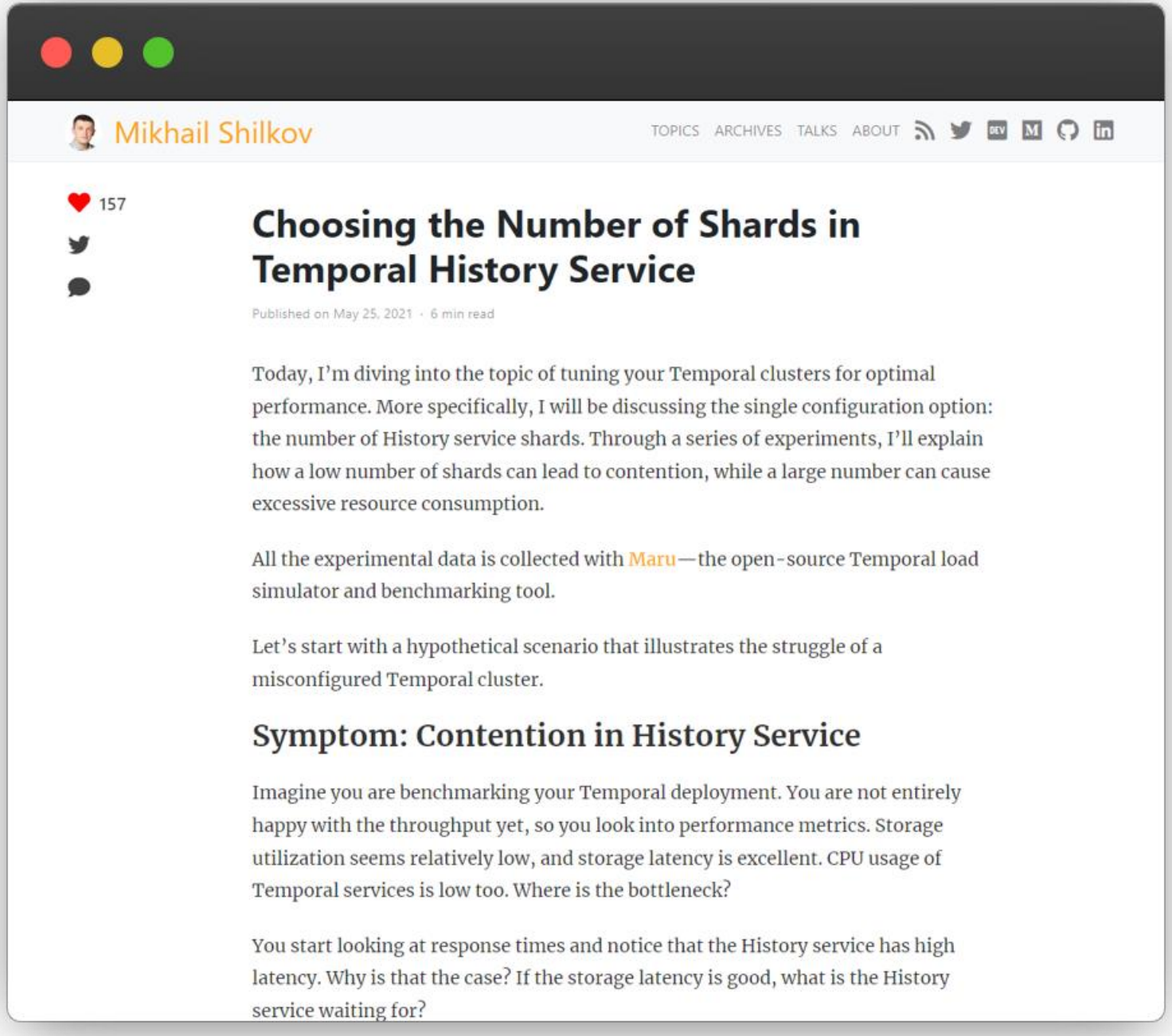
Tuning History Shard Count

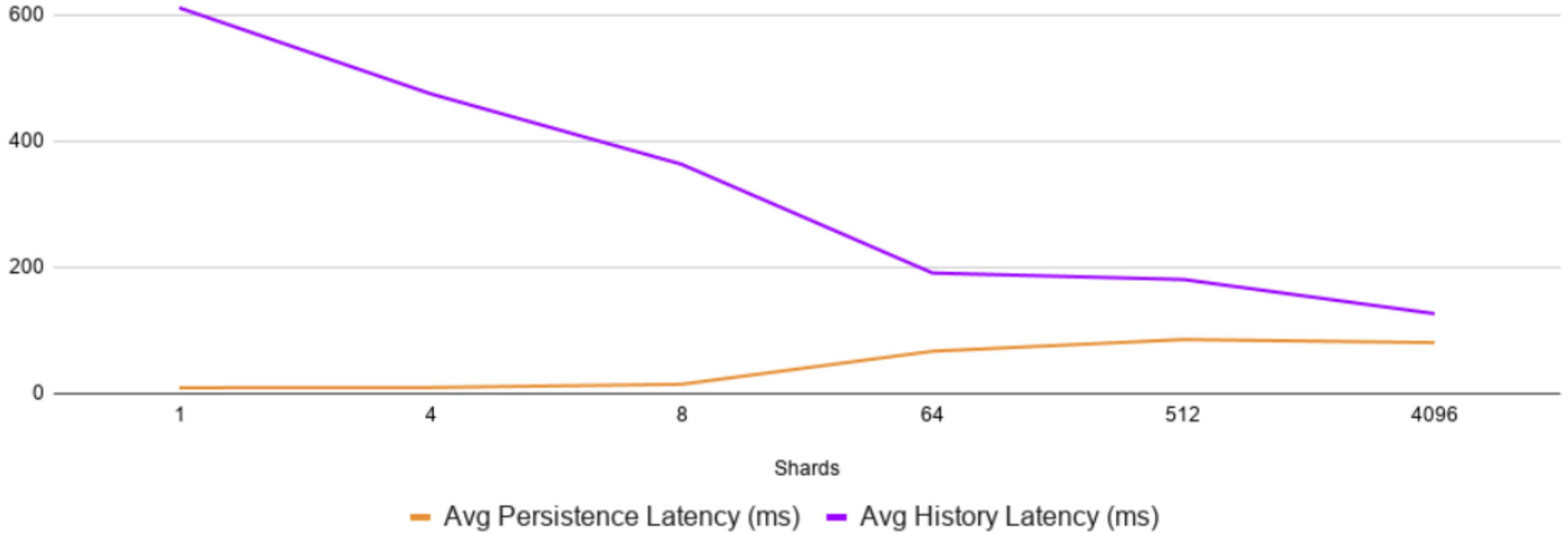
256

History Shards

4096

History Shards


Mikhail Shilkov
TOPICS ARCHIVES TALKS ABOUT
157
Choosing the Number of Shards in Temporal History Service
Published on May 25, 2021 · 6 min read
Today, I'm diving into the topic of tuning your Temporal clusters for optimal performance. More specifically, I will be discussing the single configuration option: the number of History service shards. Through a series of experiments, I'll explain how a low number of shards can lead to contention, while a large number can cause excessive resource consumption.
All the experimental data is collected with Maru—the open-source Temporal load simulator and benchmarking tool.
Let's start with a hypothetical scenario that illustrates the struggle of a misconfigured Temporal cluster.
Symptom: Contention in History Service
Imagine you are benchmarking your Temporal deployment. You are not entirely happy with the throughput yet, so you look into performance metrics. Storage utilization seems relatively low, and storage latency is excellent. CPU usage of Temporal services is low too. Where is the bottleneck?
You start looking at response times and notice that the History service has high latency. Why is that the case? If the storage latency is good, what is the History service waiting for?

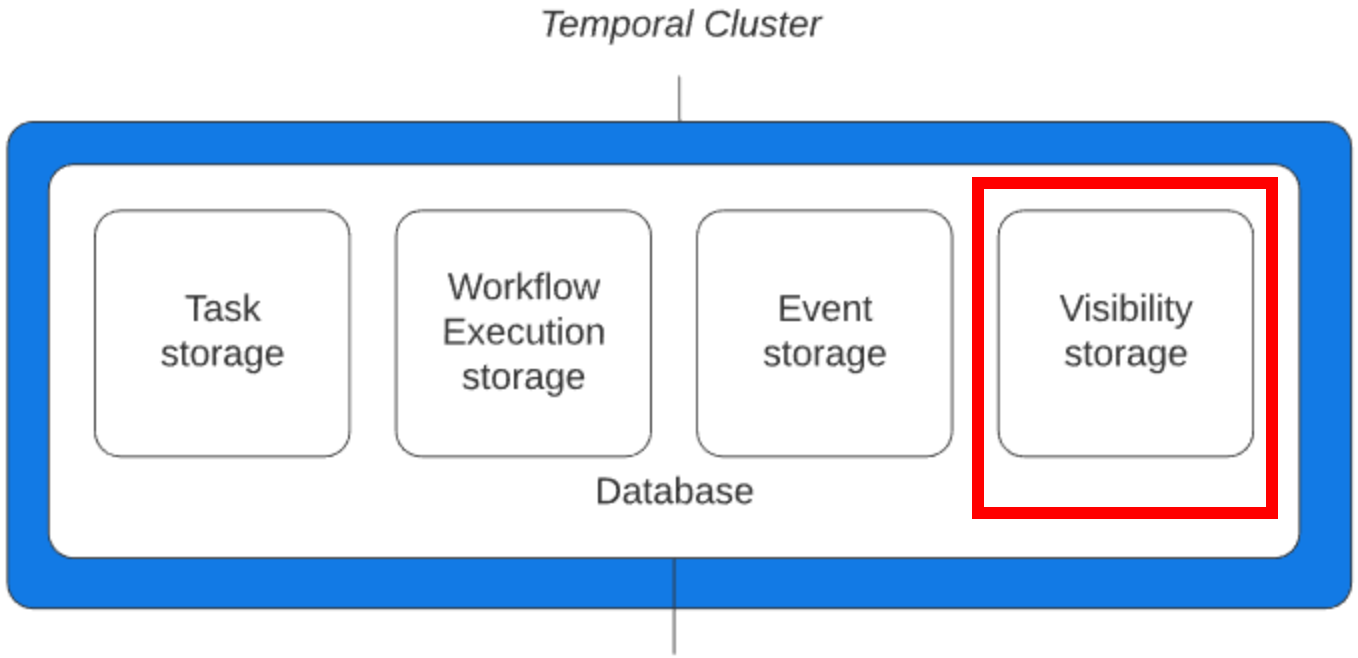


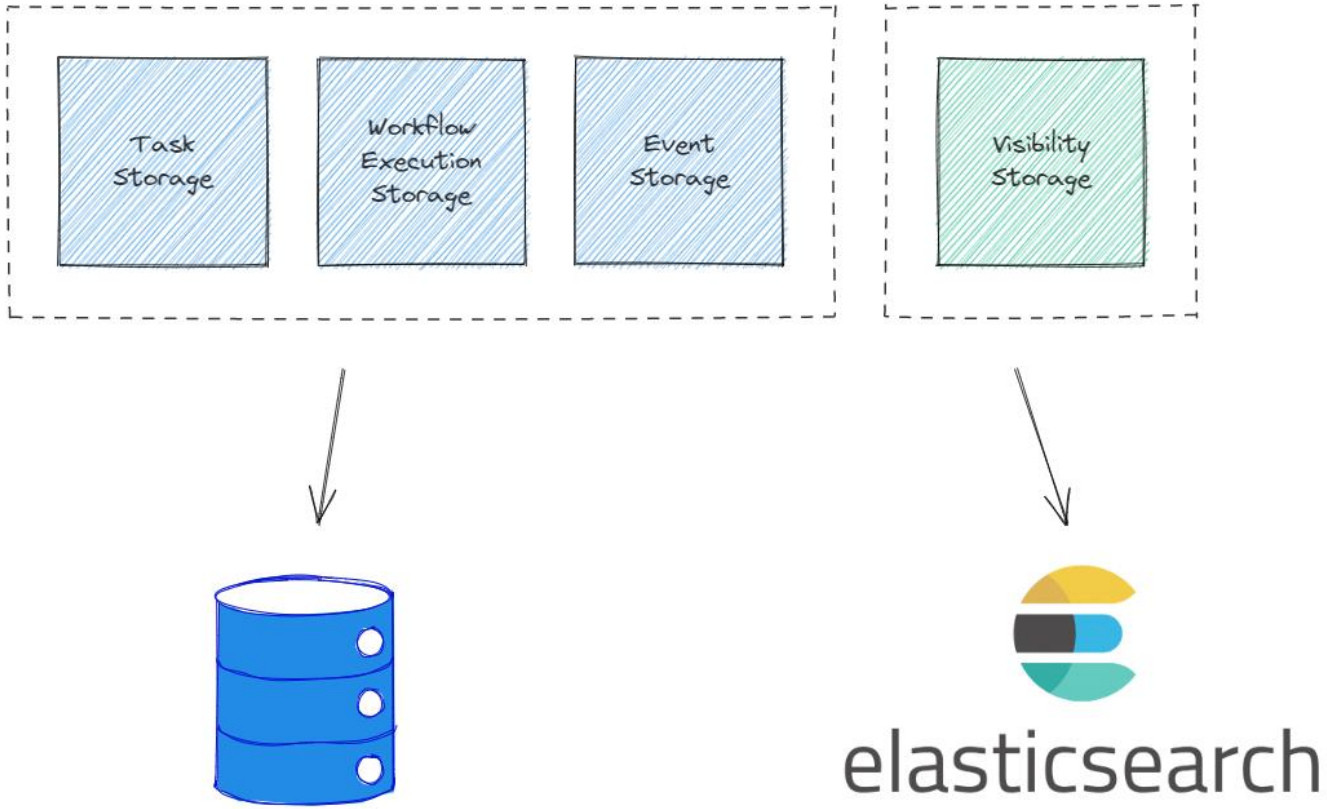
Database and service latencies as functions of the number of shards

Credit: Mikhail Shilkov

PoC Lesson #3

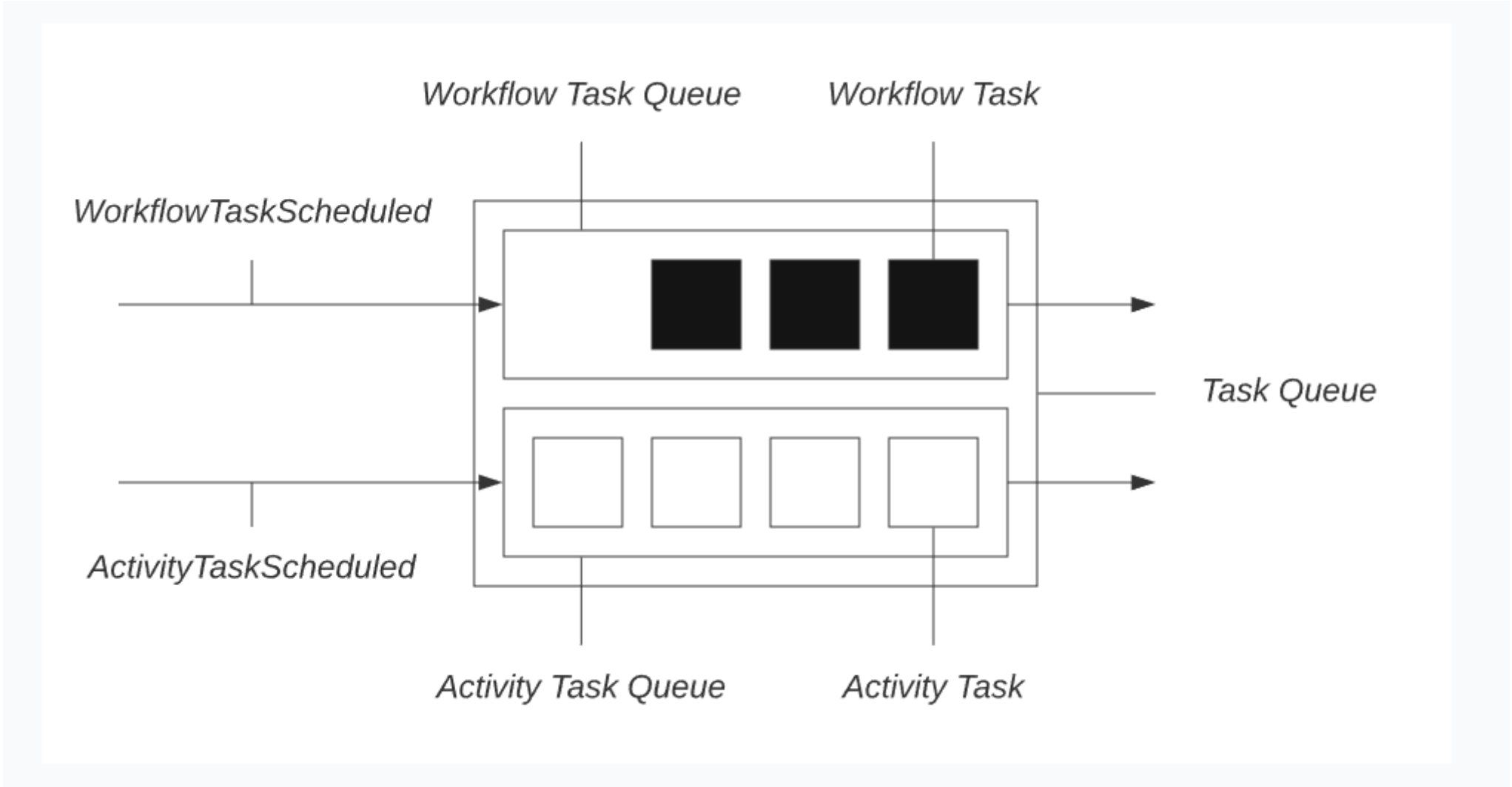
Split Visibility Database


Temporal Cluster
Task storage
Workflow Execution storage
Event storage
Visibility storage
Database
Cassandra, MySQL, and PostgreSQL are supported.


Task Storage
Workflow Execution Storage
Event Storage
Visibility Storage
elasticsearch

PoC Lesson #4

Tuning Task Queue Partitions


Workflow Task Queue
Workflow Task
WorkflowTaskScheduled
Task Queue
ActivityTaskScheduled
Activity Task Queue
Activity Task

~100

Tasks / Sec
Per Partition

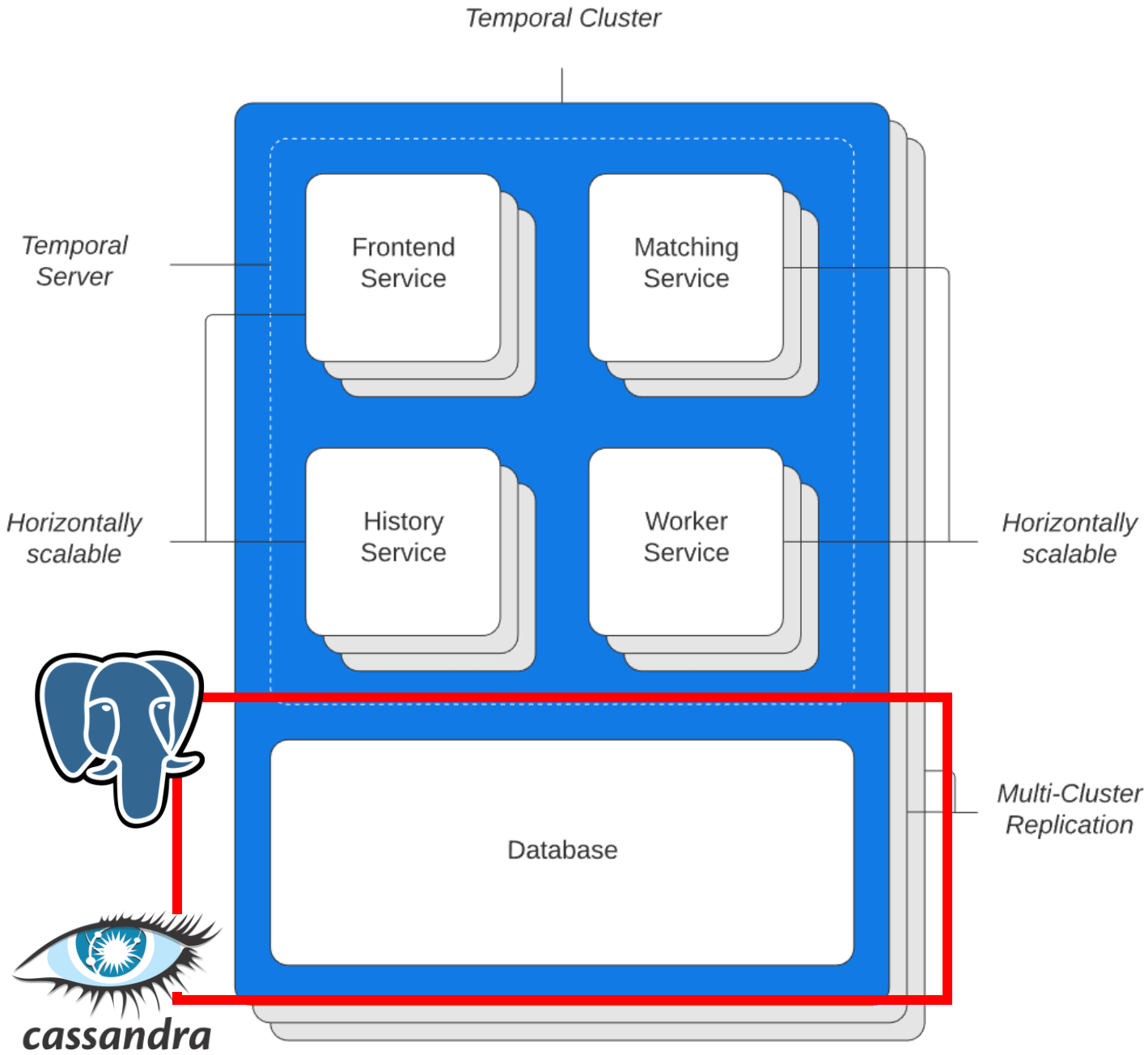
1000

Tasks / Sec

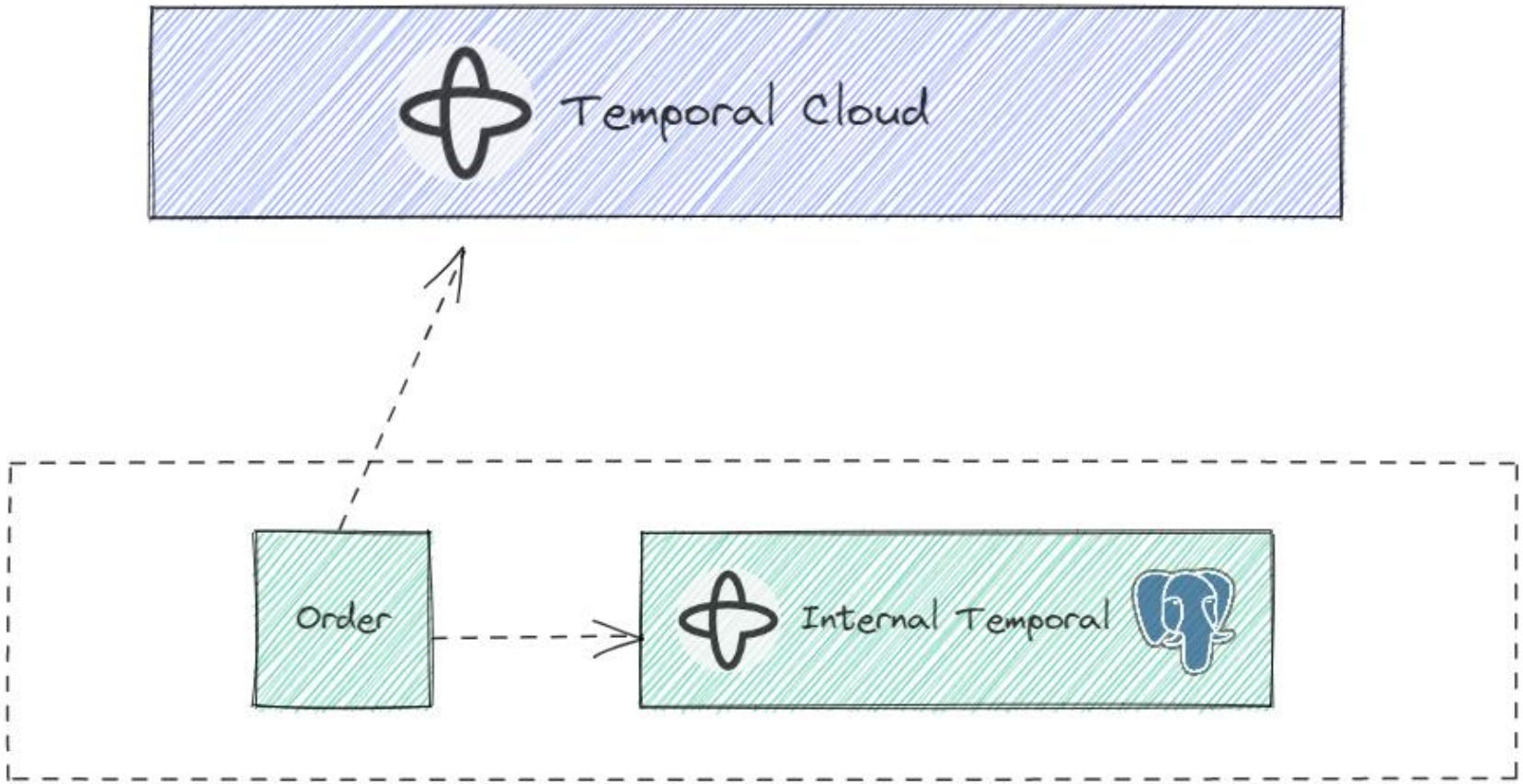
~10 Partitions

PoC Lesson #5

Trade-offs between persistence methods


Temporal Cluster
Temporal Server
Frontend Service
Matching Service
History Service
Worker Service
Horizontally scalable
Horizontally scalable
Database
Multi-Cluster Replication
cassandra


CASSANDRA DATABASE?
I WASN'T PLANNING ON SLEEPING ANYWAYS!
emegenerator.net


Temporal Cloud
Order
Internal Temporal

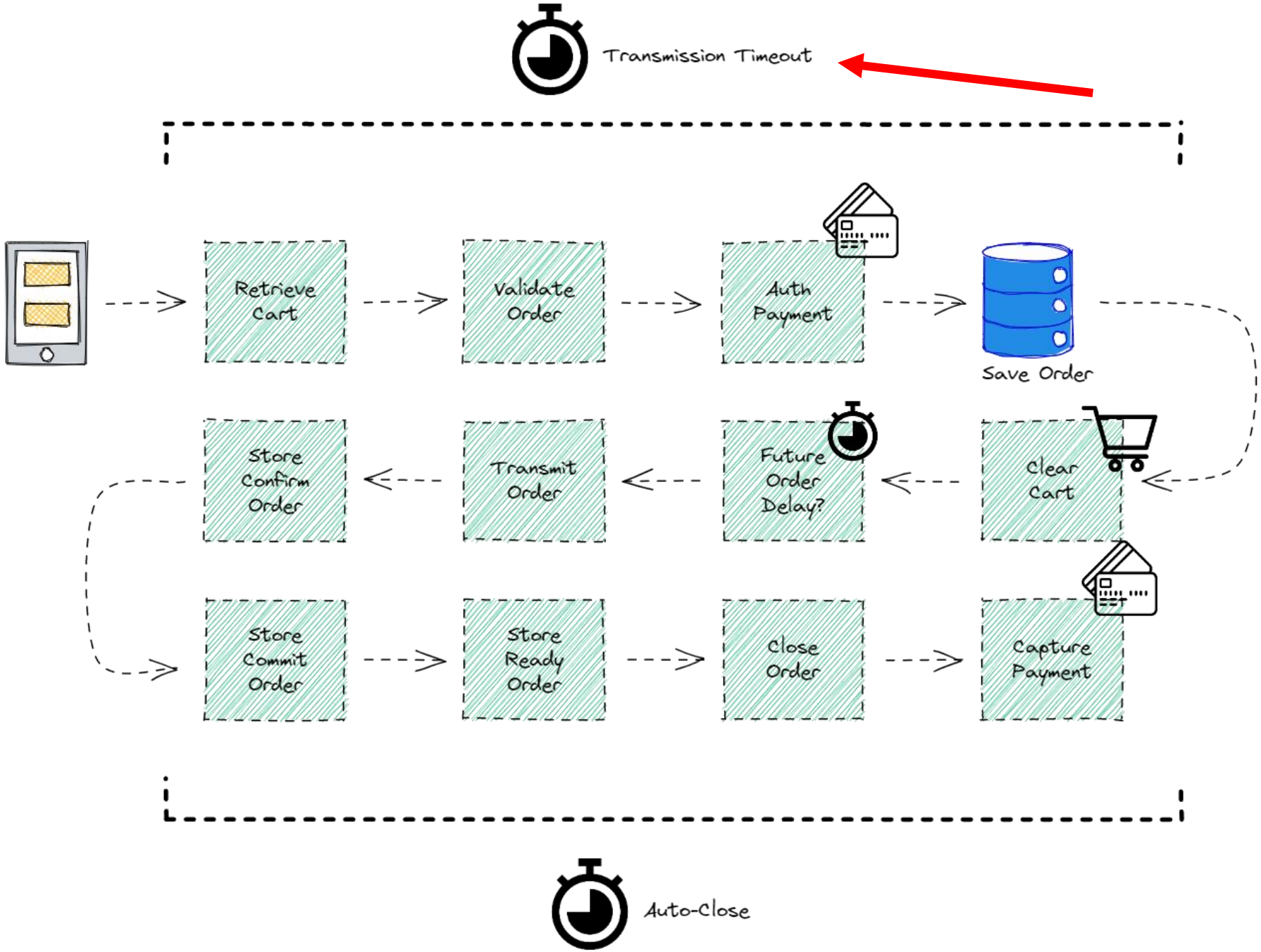
Order v2

Project Goals

1. Simplify both the code and infrastructure of our service
2. Increase our order flow reliability
3. Evaluate operation idempotency
4. Easier production and outage support

Project Goals

Simplify both **code** and **infrastructure** of our service

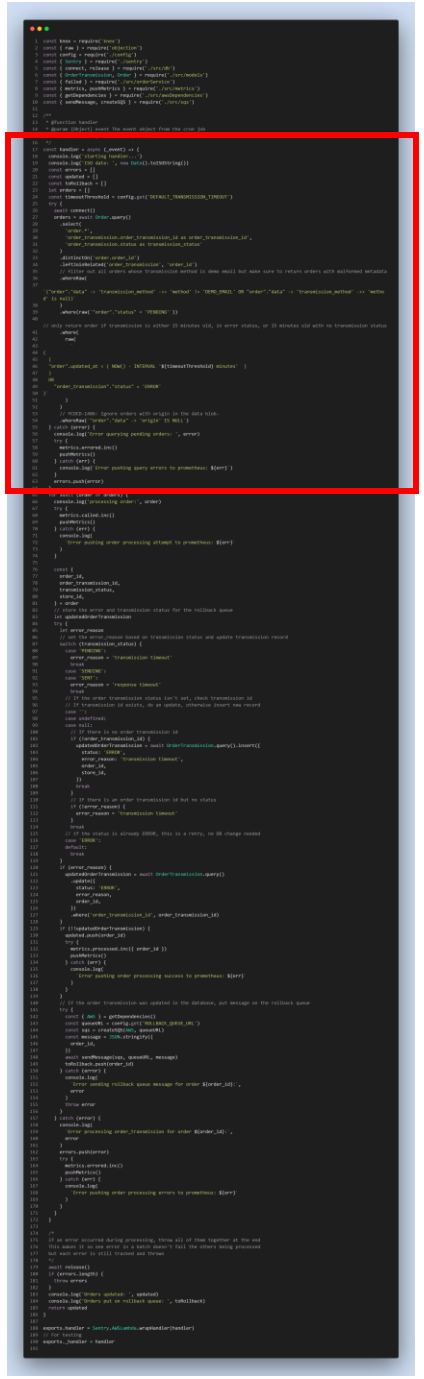

Transmission Timeout
Retrieve Cart
Validate Order
Auth Payment
Save Order
Clear Cart
Future Order Delay?
Transmit Order
Store Confirm Order
Store Commit Order
Store Ready Order
Close Order
Capture Payment
Auto-Close

Timeout Poller

Timeout Code

Timeout Poller Config

```yaml
enabled: true
image: yum-domain.com/yc-commerce/order-timeout-poller:develop
imagePullPolicy: Always
chart_features:
  dd_tracing_enabled: "true"
imagePullSecrets:
  - docker-registry
extraEnv:
  LOG_JSON_TO_STDOUT: true
  DD_SERVICE: commerceorder-order-timeout-poller
cronjob:
  schedule: "15 * * * *"
  name: commerceordertimeoutpoller-app
serviceAccount:
  enabled: true
future
```



Timeout Poller

Query orders that have exceeded their transmission timeout

```
const handler = async (_event) => {
  console.log('starting handler...')
  console.log('ISO date: ', new Date().toISOString())
  const errors = []
  const updated = []
  const toRollback = []
  let orders = []
  const timeoutThreshold = config.get('DEFAULT_TRANSMISSION_TIMEOUT')
  try {
    await connect()
    orders = await Order.query()
      .select(
        'order.*',
        'order_transmission.order_transmission_id as order_transmission_id',
        'order_transmission.status as transmission_status'
      )
      .distinctOn('order.order_id')
      .leftJoinRelated('order_transmission', 'order_id')
      // filter out all orders whose transmission method is demo email but make sure to return orders with malformed metadata
      .whereRaw(
`("order"."data" -> 'transmission_method' ->> 'method' != 'DEMO_EMAIL' OR "order"."data" -> 'transmission_method' ->> 'method' is null)`
      )
      .where(raw(`"order"."status" = 'PENDING'`))

// only return order if transmission is either 15 minutes old, in error status, or 15 minutes old with no transmission status
      .where(
        raw(
          `
(
  (
  "order".updated_at < ( NOW() - INTERVAL '${timeoutThreshold} minutes'  )
  )
  OR
    "order_transmission"."status" = 'ERROR'
)`
        )
      )
      // YCOCD-1486: Ignore orders with origin in the data blob.
      .whereRaw(`"order"."data" -> 'origin' IS NULL`)
  } catch (error) {
    console.log('Error querying pending orders: ', error)
    try {
      metrics.errored.inc()
      pushMetrics()
    } catch (err) {
      console.log(`Error pushing query errors to prometheus: ${err}`)
    }
    errors.push(error)
  }
```

Timeout Poller

Query orders that have exceeded their transmission timeout

State management for various explicit order states

```
82    // store the error and transmission status for the rollback queue
83    let updatedOrderTransmission
84    try {
85      let error_reason
86      // set the error_reason based on transmission status and update transmission re
87      switch (transmission_status) {
88        case 'PENDING':
89          error_reason = 'transmission timeout'
90          break
91        case 'SENDING':
92        case 'SENT':
93          error_reason = 'response timeout'
94          break
95        // If the order transmission status isn't set, check transmission id
96        // If transmission id exists, do an update, otherwise insert new record
97        case '':
98        case undefined:
99        case null:
100         // If there is no order transmission id
101         if (!order_transmission_id) {
102           updatedOrderTransmission = await OrderTransmission.query().insert({
103             status: 'ERROR',
104             error_reason: 'transmission timeout',
105             order_id,
106             store_id,
107           })
108           break
109         }
110         // If there is an order transmission id but no status
111         if (!error_reason) {
112           error_reason = 'transmission timeout'
113         }
114         break
115       // if the status is already ERROR, this is a retry, no DB change needed
116       case 'ERROR':
117       default:
118         break
119     }
120     if (error_reason) {
121       updatedOrderTransmission = await OrderTransmission.query()
122         .update({
123           status: 'ERROR',
124           error_reason,
125           order_id,
126         })
127         .where('order_transmission_id', order_transmission_id)
128     }
129     if (!!updatedOrderTransmission) {
130       updated.push(order_id)
131       try {
132         metrics.processed.inc({ order_id })
133         pushMetrics()
134       } catch (err) {
135         console.log(
136           `Error pushing order processing success to prometheus: ${err}`
137         )
138       }
139     }
```

Timeout Poller

Query orders that have exceeded their transmission timeout

State management for various explicit order states

Initiate a **rollback** / compensating actions

```
      // If the order transmission was updated in the database, put me
      try {
        const { AWS } = getDependencies()
        const queueURL = config.get('ROLLBACK_QUEUE_URL')
        const sqs = createSQS(AWS, queueURL)
        const message = JSON.stringify({
          order_id,
        })
        await sendMessage(sqs, queueURL, message)
        toRollback.push(order_id)
      } catch (error) {
        console.log(
          `Error sending rollback queue message for order ${order_id}`
          error
        )
        throw error
      }
    } catch (error) {
      console.log(
        `Error processing order_transmission for order ${order_id}:`,
        error
      )
      errors.push(error)
      try {
        metrics.errored.inc()
        pushMetrics()
      } catch (err) {
        console.log(
          `Error pushing order processing errors to prometheus: ${err}
        )
      }
    }
  }

  /*
  If an error occurred during processing, throw all of them together a
  This makes it so one error in a batch doesn't fail the others being
  but each error is still tracked and thrown
  */
  await release()
  if (errors.length) {
    throw errors
  }
  console.log('Orders updated: ', updated)
  console.log('Orders put on rollback queue: ', toRollback)
  return updated
}
```

Rollback Queue

```
const config = require('../config')
const { createSQS, changeMessageVisibility } = require('./helpers')
const Order = require('../models/order')
const logger = require('../lib/logger')
const runStateMachine = require('../lib/stateMachine')
const poll = require('./poll')

/**
 * @function handler
 * @param {Object} event The event object from SQS
 * Handle the message from the SQS queue
 */
async function handler({ sqs, queueUrl }, event){
  let payload

  try {
    payload = JSON.parse(event.Body)
  }
  catch (err){
    logger.error('Unable to parse message body', {
      action: 'rollback_order',
      error: err
    })
    throw err
  }

  if (!('order_id' in payload)){
    throw new Error(`Rollback message must include the order_id`)
  }

  const { order_id } = payload
  logger.info(`Rolling back ${order_id}`, {
    action: 'rollback_order',
    type: ['change']
  })

  const { client_id, customer_id } = payload

  try {
    const order = await Order.query().findById(order_id)

    await runStateMachine({
      order,
      event: 'UPDATE_STATUS_FAILED',
      context: {
        origin: 'rollbackQueue',
        order_status: 'FAILED',
        order_id,
        claims: { client_id, customer_id },
        isAdmin: true
      }
    })

    return true
  }
  catch (err){
    logger.error('Failed to update status for rollback', {
      action: 'rollback_order',
      error: err
    })

    await changeMessageVisibility(sqs, {
      queueUrl,
      handle: event.ReceiptHandle,
      retryCount: event.Attributes.ApproximateReceiveCount,
      retryPrimaryLimit: config.get('ROLLBACK_RETRY_PRIMARY_LIMIT'),
      retryPrimaryDelay: config.get('ROLLBACK_RETRY_PRIMARY_DELAY'),
      retrySecondaryDelay: config.get('ROLLBACK_RETRY_SECONDARY_DELAY')
    })

    return false
  }
}

module.exports = function rollbackQueue({
  queueUrl = config.get('ROLLBACK_SQS'),
  sqs = createSQS(queueUrl)
} = {}){
  const args = { sqs, queueUrl }

  logger.info(`Listening to rollback queue`, {
    action: 'listening_rollback_queue',
    type: ['info'],
    queueUrl
  })

  return poll({
    queueUrl,
    sqs
  }, (message) => handler(args, message))
}
```

Execute “failed order” state machine step

```
  await runStateMachine({
    order,
    event: 'UPDATE_STATUS_FAILED',
    context: {
      origin: 'rollbackQueue',
      order_status: 'FAILED',
      order_id,
      claims: { client_id, customer_id },
      isAdmin: true
    }
  })

  return true
}
```

Rollback Queue

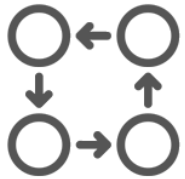
```
const config = require('../config')
const { createSQS, changeMessageVisibility } = require('./helpers')
const Order = require('../models/order')
const logger = require('../lib/logger')
const runStateMachine = require('../lib/stateMachine')
const poll = require('./poll')

/**
 * @function handler
 * @param {Object} event The event object from SQS
 * Handle the message from the SQS queue
 */
async function handler({ sqs, queueUrl }, event){
  let payload

  try {
    payload = JSON.parse(event.Body)
  }
  catch (err){
    logger.error('Unable to parse message body', {
      action: 'rollback_order',
      error: err
    })
    throw err
  }

  if (!('order_id' in payload)){
    throw new Error(`Rollback message must include the order_id`)
  }

  const { order_id } = payload
  logger.info(`Rolling back ${order_id}`, {
    action: 'rollback_order',
    type: ['change']
  })

  const { client_id, customer_id } = payload

  try {
    const order = await Order.query().findById(order_id)

    await runStateMachine({
      order,
      event: 'UPDATE_STATUS_FAILED',
      context: {
        origin: 'rollbackQueue',
        order_status: 'FAILED',
        order_id,
        claims: { client_id, customer_id },
        isAdmin: true
      }
    })

    return true
  }
  catch (err){
    logger.error('Failed to update status for rollback', {
      action: 'rollback_order',
      error: err
    })

    await changeMessageVisibility(sqs, {
      queueUrl,
      handle: event.ReceiptHandle,
      retryCount: event.Attributes.ApproximateReceiveCount,
      retryPrimaryLimit: config.get('ROLLBACK_RETRY_PRIMARY_LIMIT'),
      retryPrimaryDelay: config.get('ROLLBACK_RETRY_PRIMARY_DELAY'),
      retrySecondaryDelay: config.get('ROLLBACK_RETRY_SECONDARY_DELAY')
    })

    return false
  }
}

module.exports = function rollbackQueue({
  queueUrl = config.get('ROLLBACK_SQS'),
  sqs = createSQS(queueUrl)
} = {}){
  const args = { sqs, queueUrl }

  logger.info(`Listening to rollback queue`, {
    action: 'listening_rollback_queue',
    type: ['info'],
    queueUrl
  })

  return poll({
    queueUrl,
    sqs
  }, (message) => handler(args, message))
}
```

Execute “failed order” **state machine step**

Retry rollback should it fail

```
  catch (err){
    logger.error('Failed to update status for rollback', {
      action: 'rollback_order',
      error: err
    })

    await changeMessageVisibility(sqs, {
      queueUrl,
      handle: event.ReceiptHandle,
      retryCount: event.Attributes.ApproximateReceiveCount,
      retryPrimaryLimit: config.get('ROLLBACK_RETRY_PRIMARY_LIMIT'),
      retryPrimaryDelay: config.get('ROLLBACK_RETRY_PRIMARY_DELAY'),
      retrySecondaryDelay: config.get('ROLLBACK_RETRY_SECONDARY_DELAY')
    })

    return false
  }
}
```

State Machine



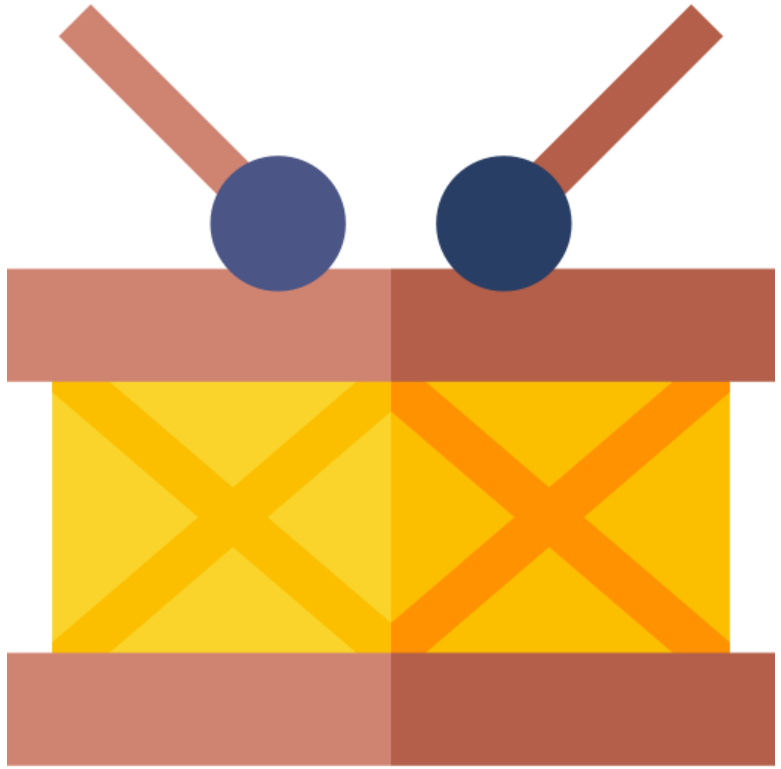
```
const machineConfig = {
  id: 'orderservice',
  initial: 'processing',
  states: {
    processing: {
      type: 'parallel',
      states: {
        order: {
          initial: 'creating_order',
          states: {

            // ... A million other states

            pending: {
              on: {
                UPDATE_STATUS_FAILED: {
                  target: '#orderservice.processing.payment.canceling'
                },
                UPDATE_STATUS_CONFIRMED: [
                  {
                    cond: 'isNonDeliveryOrder',
                    target: '#orderservice.processing.payment.capturing'
                  },
                  {
                    cond: 'isDeliveryOrder',
                    target: 'updating_status'
                  }
                ],
```

```
canceling: {
   invoke: {
     id: 'cancelPayment',
     src: cancelPayment,
     onDone: [
       {
         cond: 'statusAlreadyUpdated',
         target: '#orderservice.processing.order.publish_event'
       },
       {
         target: '#orderservice.processing.order.updating_status'
       }
     ],
     onError: {
       target: '#orderservice.done',
       actions: 'assignResponseSendError'
     }
   }
 },
```

```
async function cancel({
  paymentAuthorizationId,
  authorizedAmount,
  currency_code,
  orderId,
}) {
  // Get token from Cognito
  const token = await getToken()

  console.log(
    `Canceling: (${currency_code})${authorizedAmount}
 for paymentAuthId #${paymentAuthorizationId}, order #${orderId}`
  )

  const response = await fetch(
    `${PAYMENTS_API_URL}/authorizations/${paymentAuthorizationId}
/cancels`,
    {
      method: 'POST',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        cancel_amount_request: {
          amount: authorizedAmount,
          currency_code,
        },
      }),
    }
  )

}
```

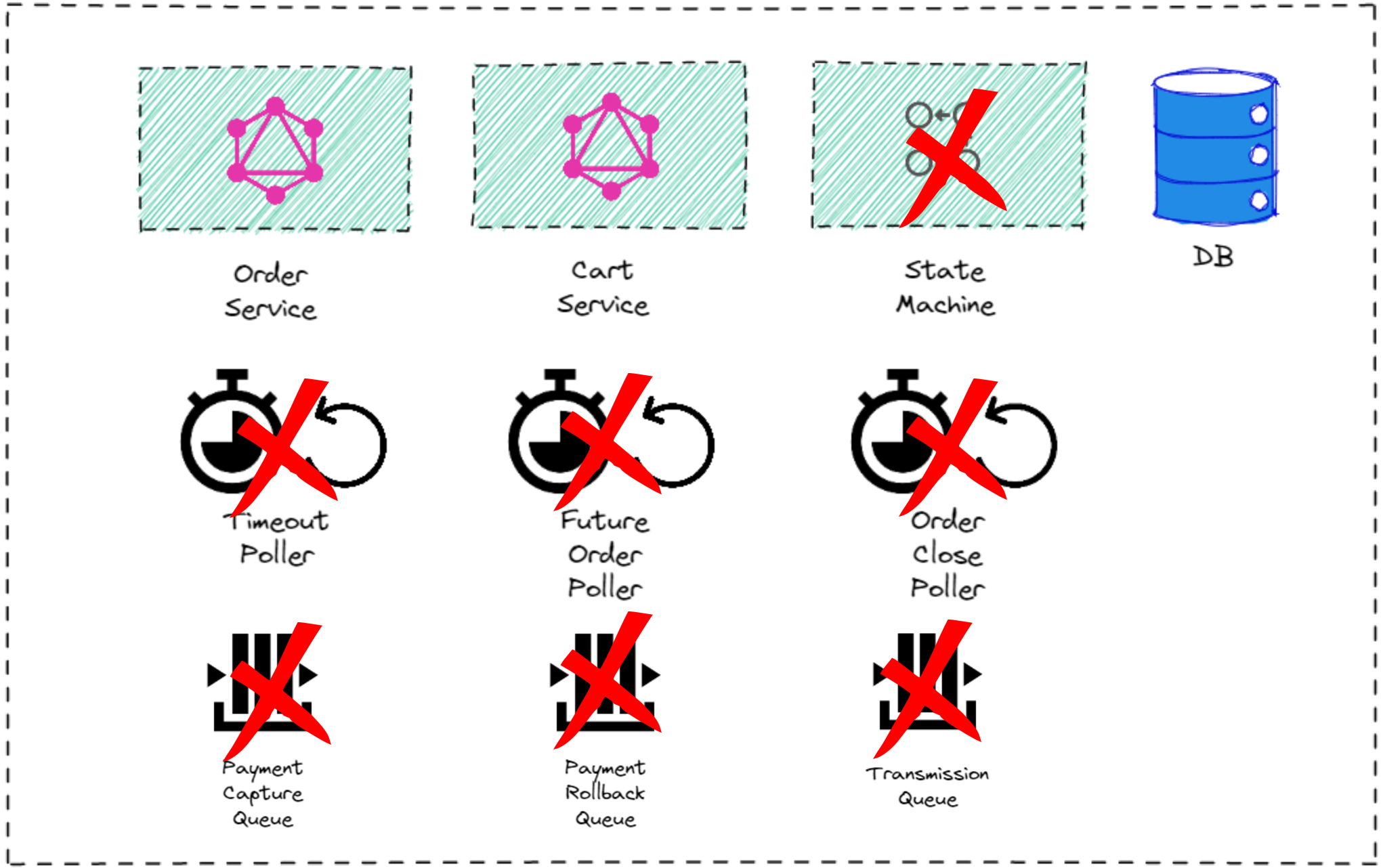
Payment Cancellation

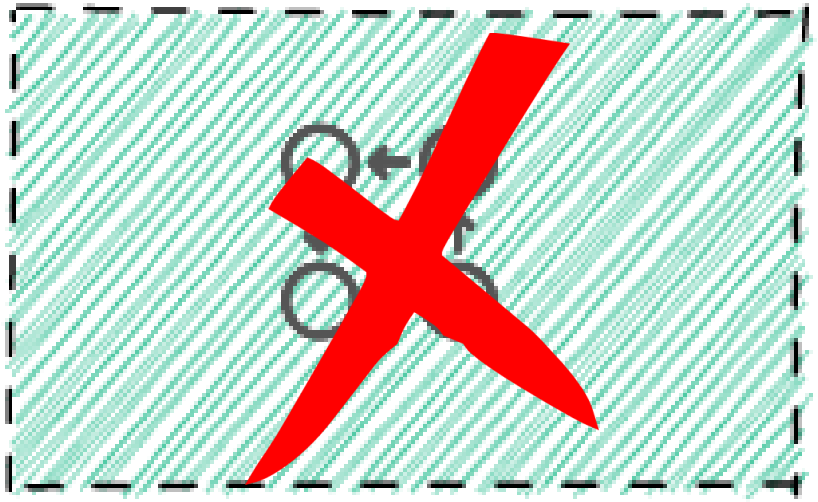
Temporal Transmission Timeout



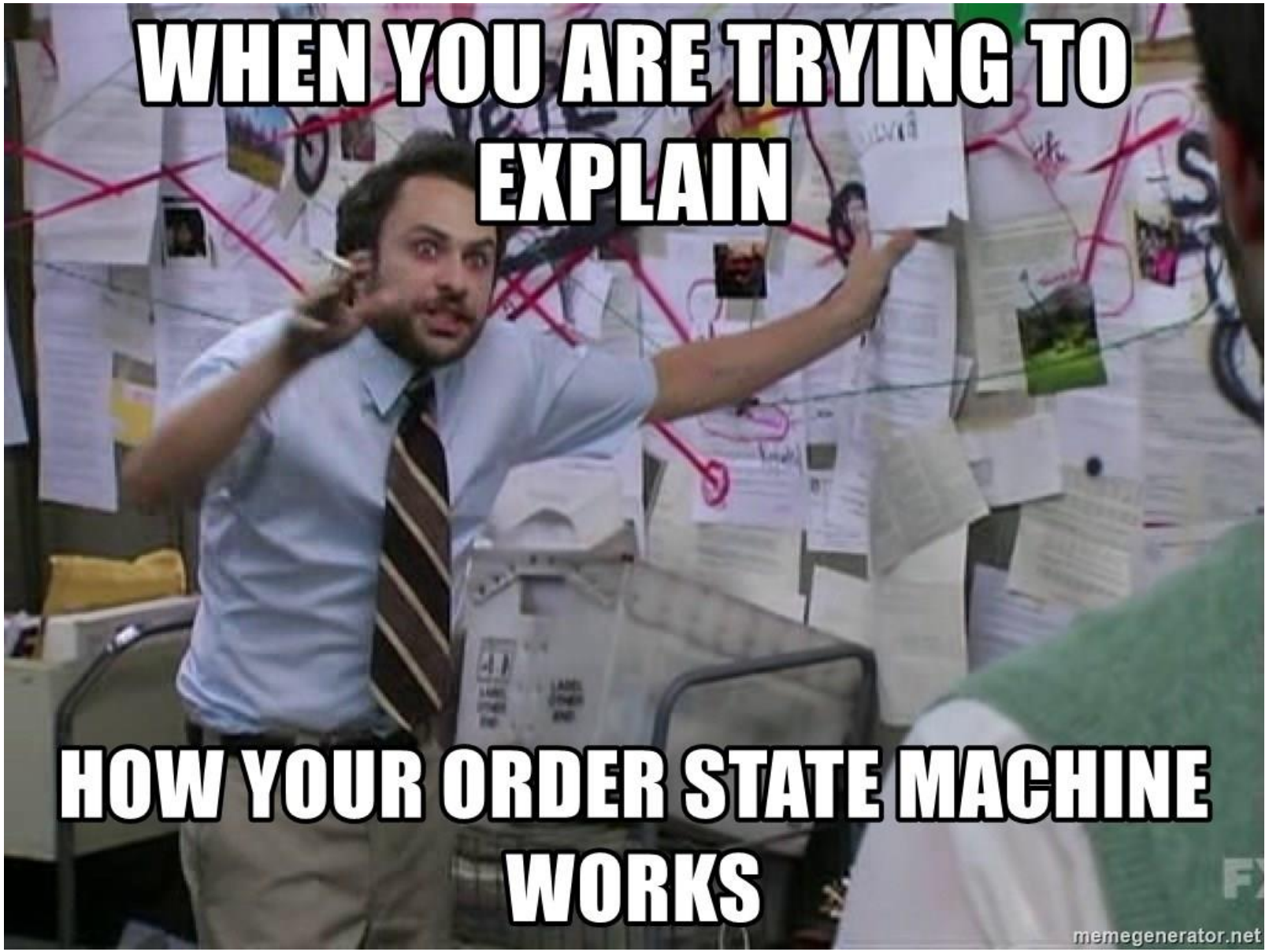
Temporal Transmission Timeout

```go
timeoutCtx, cancelTransmissionTimeout := workflow.WithCancel(ctx)
selector := workflow.NewSelector(timeoutCtx)

transmissionTimeout := workflow.NewTimer(timeoutCtx, transmissionTimeoutDuration)

selector.AddFuture(transmissionTimeout, func (f workflow.Future)  {

    activityOptions := workflow.ActivityOptions{
        RetryPolicy: &temporal.RetryPolicy{
            MaximumAttempts:    3,
        },
    }
    retryCtx := workflow.WithActivityOptions(ctx, activityOptions)

    var voidPaymentResult string
    if err := workflow.ExecuteActivity(retryCtx, activities.PaymentVoidActivity, authorizationId).Get(ctx,
        &voidPaymentResult); err != nil {
        return err
    }

    var failOrderResult string
    if err := workflow.ExecuteActivity(retryCtx, activities.FailOrderActivity, orderId).Get
(ctx, &failOrderResult); err != nil {
        return nil, err
    }
})

selector.AddReceive(updCh, func(_ workflow.ReceiveChannel, _ bool) {
    cancelTransmissionTimeout()
})

selector.Select(ctx)
```


Order Service
Cart Service
State Machine
DB
Timeout Poller
Future Order Poller
Order Close Poller
Payment Capture Queue
Payment Rollback Queue
Transmission Queue

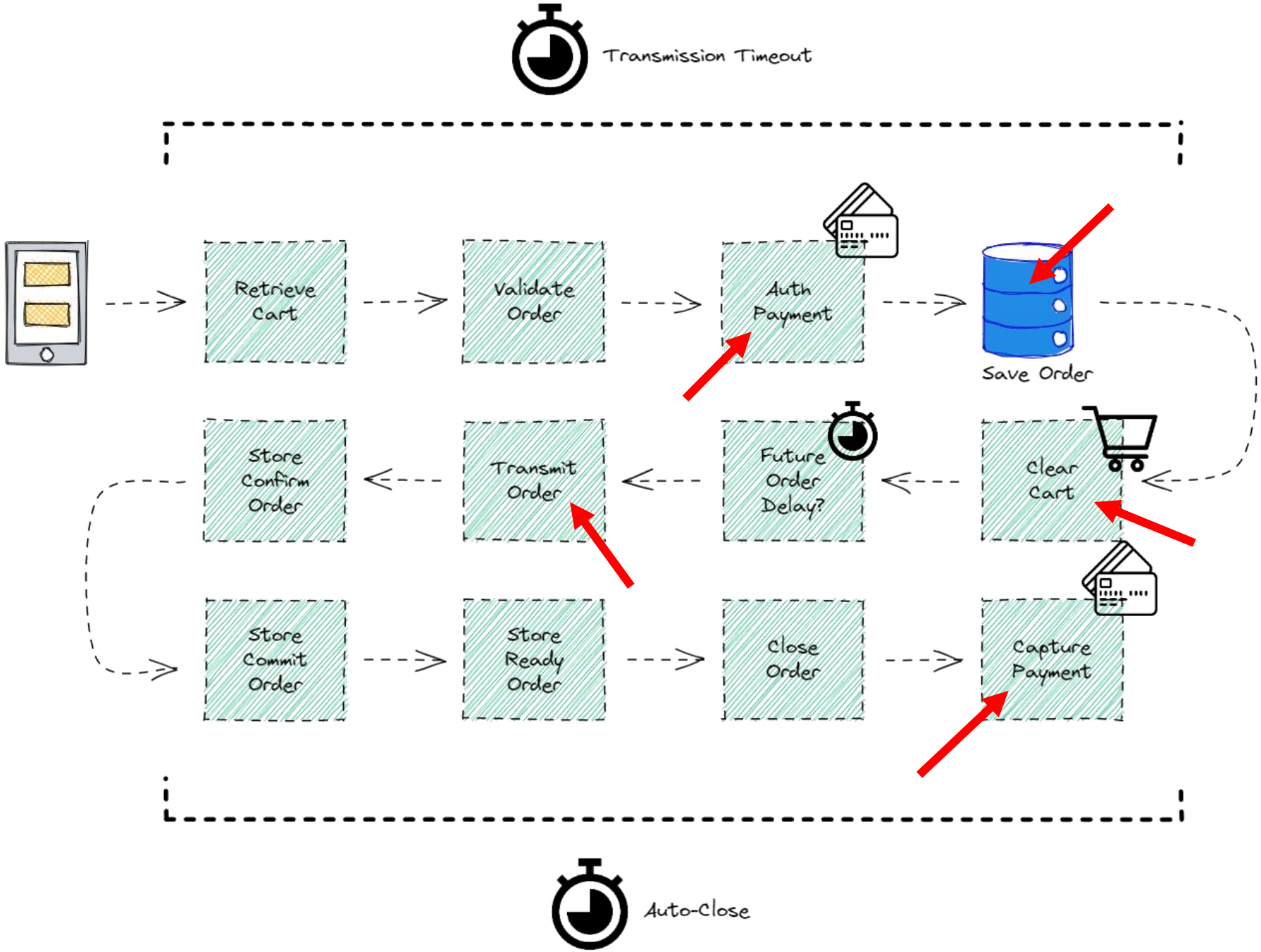


State
Machine


WHEN YOU ARE TRYING TO EXPLAIN
HOW YOUR ORDER STATE MACHINE WORKS
memegenerator.net

Project Goals

Increase our order flow **reliability**


Transmission Timeout
Retrieve Cart
Validate Order
Auth Payment
Save Order
Clear Cart
Future Order Delay?
Transmit Order
Store Confirm Order
Store Commit Order
Store Ready Order
Close Order
Capture Payment
Auto-Close

Default values for Retry Policy

```
Initial Interval     = 1 second
Backoff Coefficient  = 2.0
Maximum Interval     = 100 × Initial Interval
Maximum Attempts     = ∞
Non-Retryable Errors = []
```

```go
func (o *OrderWorkflowV2) authorizePayment(ctx workflow.Context, params authorizePayment) (model.PaymentAuth, error) {
    var pa model.PaymentAuth

    input := adapters.PaymentAuthorizeInput{
        OrderID:        params.OrderID,
        StoreNumber:    params.StoreNumber,
        OrgID:          params.OrgID,
        PaymentMethods: params.PaymentMethods,
    }

    authorizeCreditCardActivityOptions := workflow.LocalActivityOptions{
        StartToCloseTimeout: defaultStartToCloseTimeOut,
        RetryPolicy: &temporal.RetryPolicy{
            BackoffCoefficient: 2.0,
            MaximumAttempts: 3,
        },
    }

    actx := workflow.WithLocalActivityOptions(ctx, authorizeCreditCardActivityOptions)
    if err := workflow.ExecuteActivity(actx, activities.PaymentAuthorizeCreditCardActivityName, input).Get(ctx,
        &pa); err != nil {
        return pa, err
    }
    return pa, nil
}

```

Project Goals

Evaluate operation
idempotency

*“An idempotent operation is one that has no **additional effect** if it is called more than once with the **same input** parameters.”*

Credit: Wikipedia Idempotency Article

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 2 January 2022

J. Jena
PayPal, Inc.
S. Dalal
1 July 2021

The Idempotency-Key HTTP Header Field
draft-ietf-httpapi-idempotency-key-header-00

Abstract

The HTTP Idempotency-Key request header field can be used to carry idempotency key in order to make non-idempotent HTTP methods such as "POST" or "PATCH" fault-tolerant.

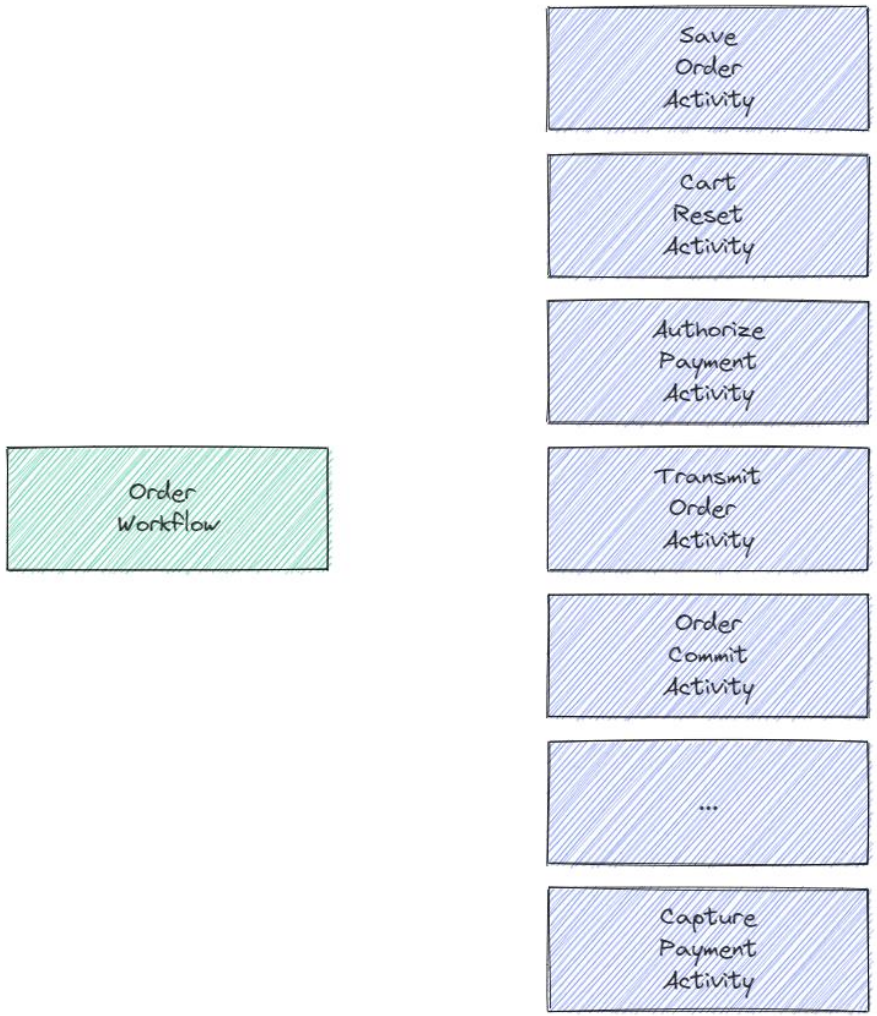
Status of This Memo

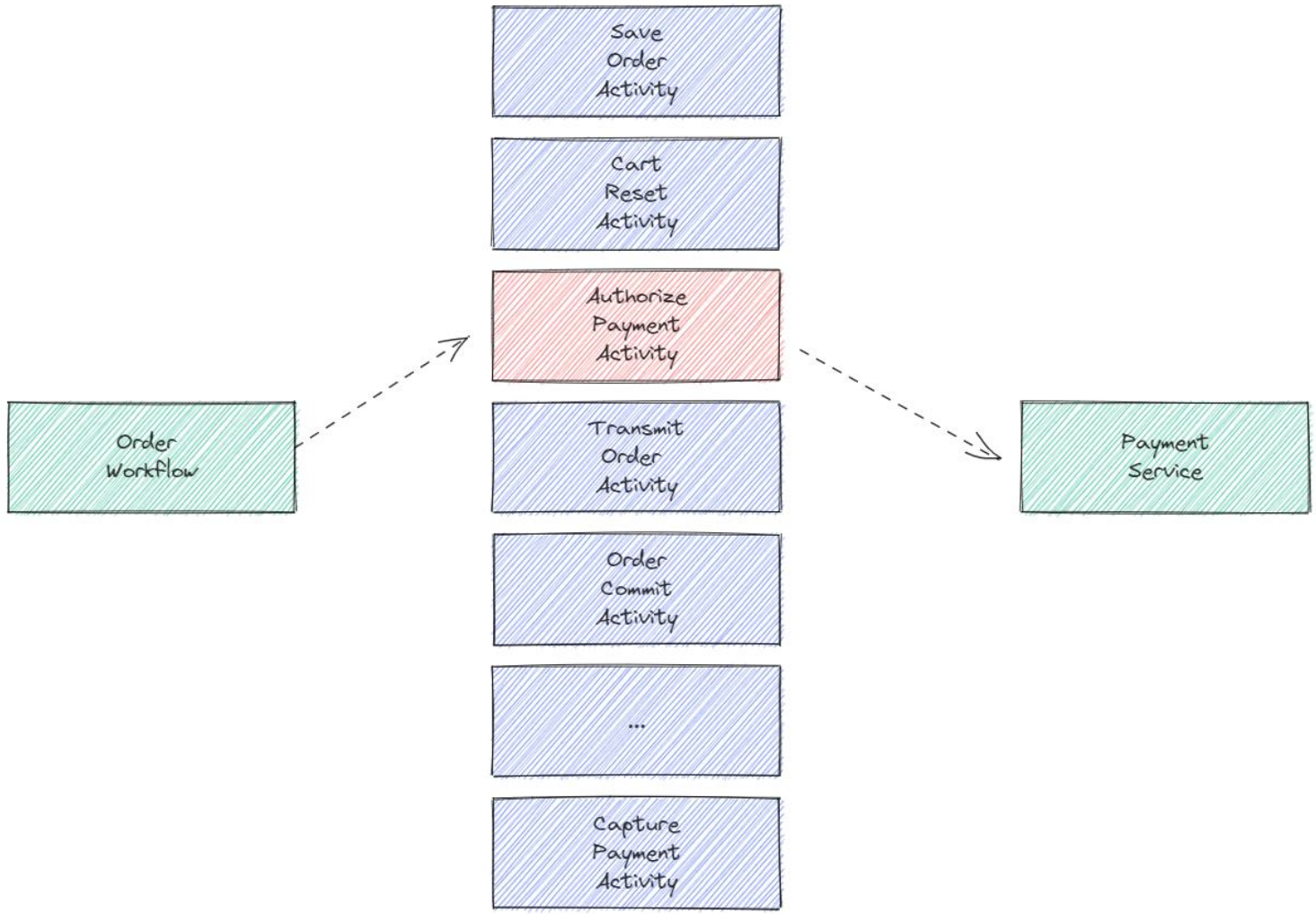
This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

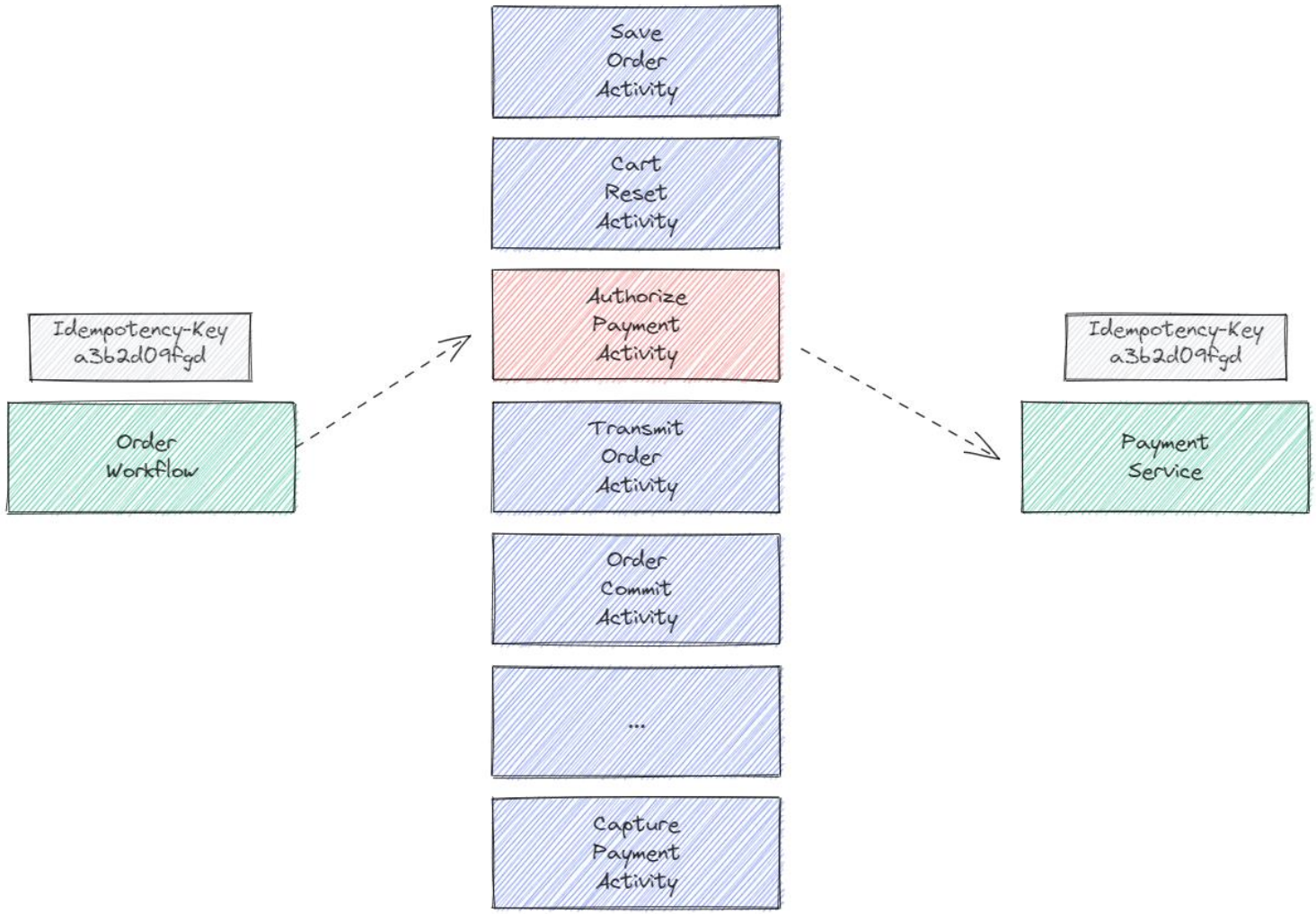
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 January 2022.

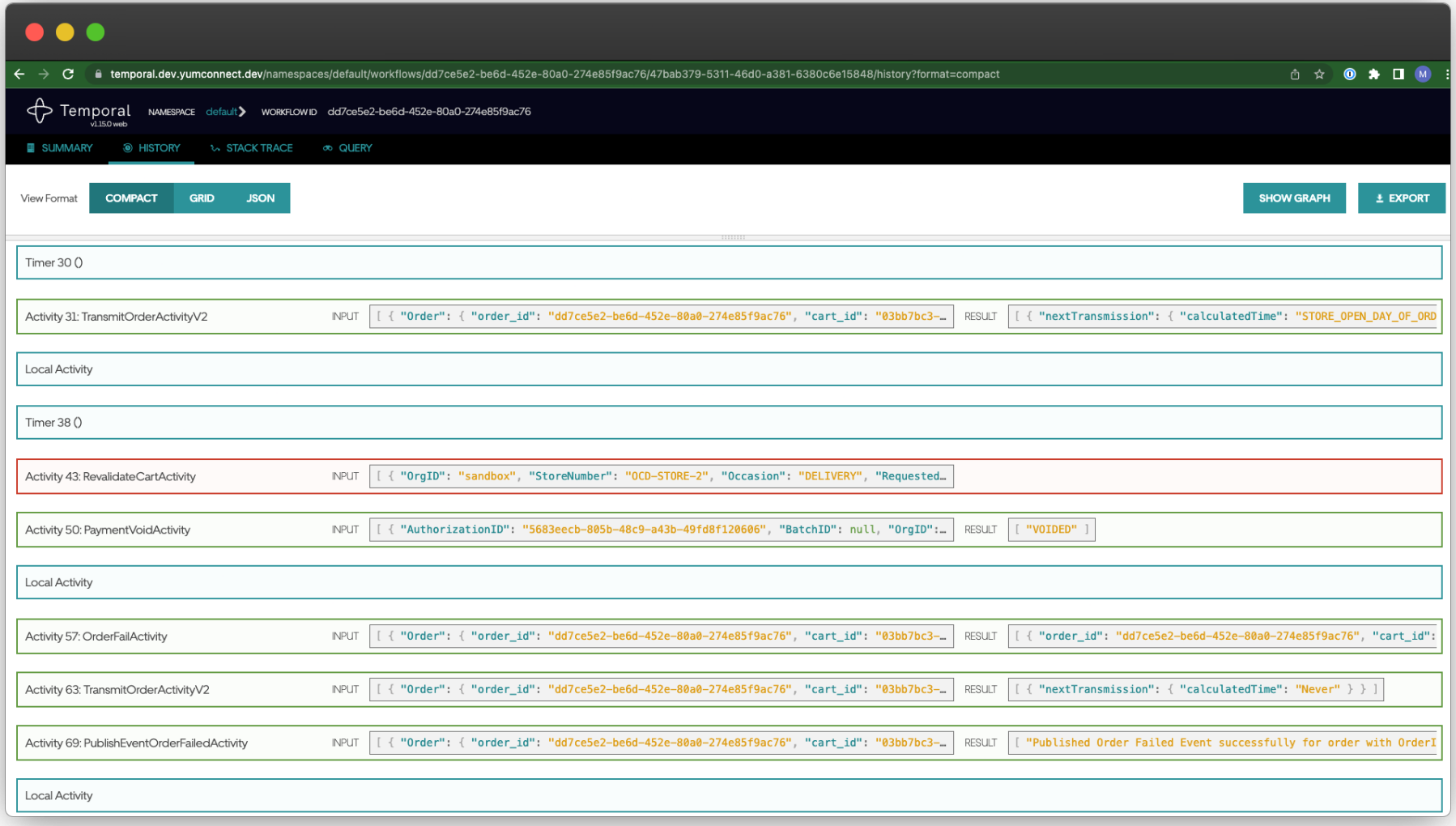

Order
Workflow
Save
Order
Activity
Cart
Reset
Activity
Authorize
Payment
Activity
Transmit
Order
Activity
Order
Commit
Activity
...
Capture
Payment
Activity


Save Order Activity
Cart Reset Activity
Authorize Payment Activity
Transmit Order Activity
Order Commit Activity
...
Capture Payment Activity
Order Workflow
Payment Service

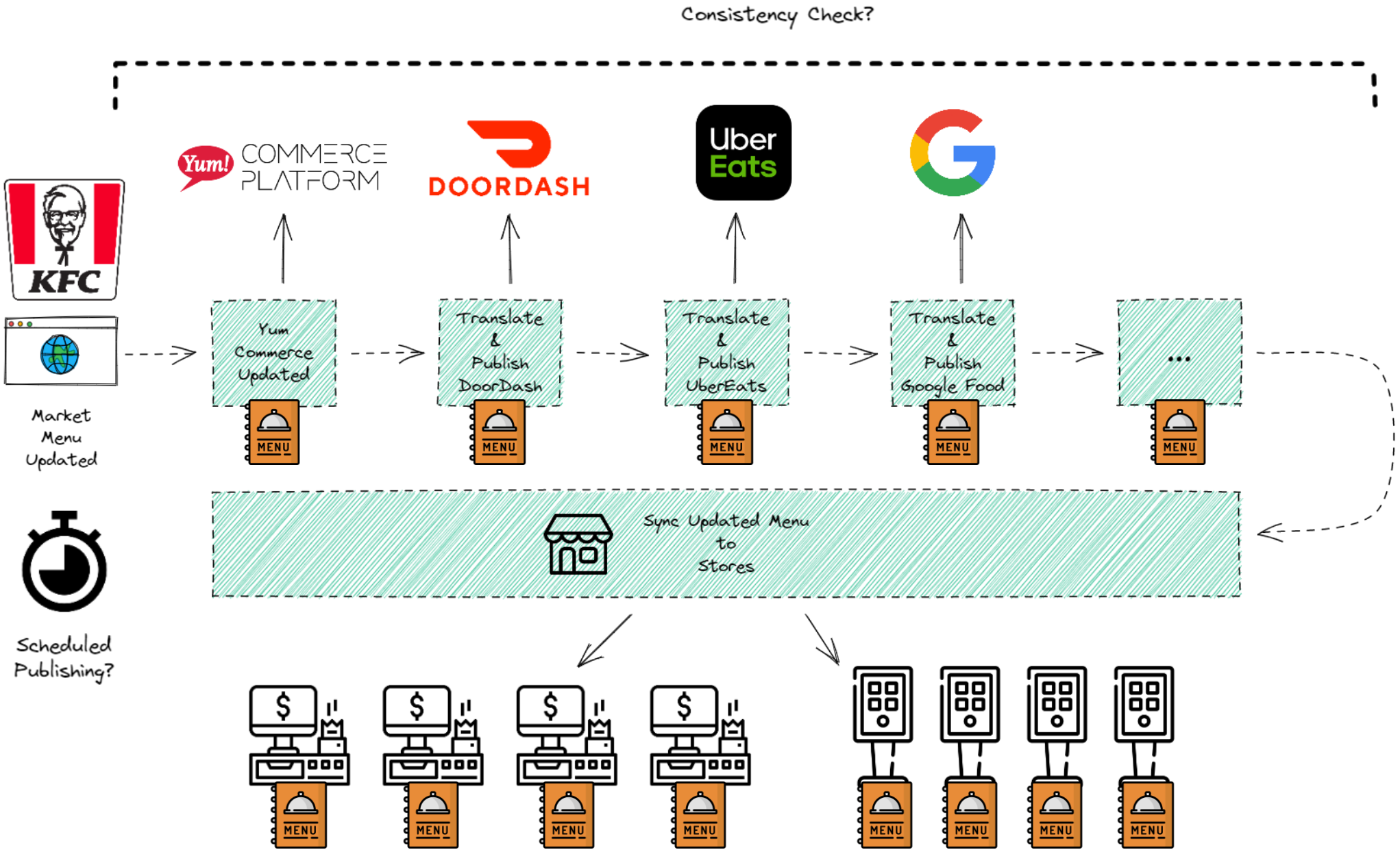

Save Order Activity
Cart Reset Activity
Authorize Payment Activity
Transmit Order Activity
Order Commit Activity
...
Capture Payment Activity
Idempotency-Key a3b2d09fgd
Order Workflow
Idempotency-Key a3b2d09fgd
Payment Service

Project Goals

Simplify **production support**


temporal.dev.yumconnect.dev/namespaces/default/workflows/dd7ce5e2-be6d-452e-80a0-274e85f9ac76/47bab379-5311-46d0-a381-6380c6e15848/history?format=compact
Temporal
v1.15.0 web
NAMESPACE default
WORKFLOW ID dd7ce5e2-be6d-452e-80a0-274e85f9ac76
SUMMARY
HISTORY
STACK TRACE
QUERY
View Format
COMPACT
GRID
JSON
SHOW GRAPH
EXPORT
Timer 30 ()
Activity 31: TransmitOrderActivityV2
INPUT
[{ "Order": { "order_id": "dd7ce5e2-be6d-452e-80a0-274e85f9ac76", "cart_id": "03bb7bc3-…
RESULT
[{ "nextTransmission": { "calculatedTime": "STORE_OPEN_DAY_OF_ORD
Local Activity
Timer 38 ()
Activity 43: RevalidateCartActivity
INPUT
[{ "OrgID": "sandbox", "StoreNumber": "OCD-STORE-2", "Occasion": "DELIVERY", "Requested…
Activity 50: PaymentVoidActivity
INPUT
[{ "AuthorizationID": "5683eecb-805b-48c9-a43b-49fd8f120606", "BatchID": null, "OrgID":…
RESULT
["VOIDED"]
Local Activity
Activity 57: OrderFailActivity
INPUT
[{ "Order": { "order_id": "dd7ce5e2-be6d-452e-80a0-274e85f9ac76", "cart_id": "03bb7bc3-…
RESULT
[{ "order_id": "dd7ce5e2-be6d-452e-80a0-274e85f9ac76", "cart_id":
Activity 63: TransmitOrderActivityV2
INPUT
[{ "Order": { "order_id": "dd7ce5e2-be6d-452e-80a0-274e85f9ac76", "cart_id": "03bb7bc3-…
RESULT
[{ "nextTransmission": { "calculatedTime": "Never" } }]
Activity 69: PublishEventOrderFailedActivity
INPUT
[{ "Order": { "order_id": "dd7ce5e2-be6d-452e-80a0-274e85f9ac76", "cart_id": "03bb7bc3-…
RESULT
["Published Order Failed Event successfully for order with OrderI
Local Activity

What is next?


Consistency Check?
Yum! COMMERCE PLATFORM
DOORDASH
Uber Eats
KFC
Yum Commerce Updated
Translate & Publish DoorDash
Translate & Publish UberEats
Translate & Publish Google Food
...
MENU
Market Menu Updated
Sync Updated Menu to Stores
Scheduled Publishing?

Summary

- Deleted - Massive state machine
- Deleted - Tons of explicitly maintained queues
- Deleted - A bunch of pollers
- Deleted - Got rid of a bunch of deployables to support our flow
- Improved - Robustness with easier retry behavior
- Improved - Performance exceeded our long-term goals
- Improved - Simplified the mental overhead of our code significantly

Q&A