THANK YOU ORGANIZERS!

Ego Slide

- Push buttons
- Type things
- Break stuff
- Wear sweatshirt as cape
- This picture is not stack safe

Everyday Code

```
const processPayment =
  (cardNumber: CardNumber, deliveryAddress: DeliveryAddress,
  email: EmailAddress, orderId: OrderId) ⇒
    Effect.gen(function*(_) {
      // get total order amount
      const totalAmount = yield* _(getTotalAmount(orderId))
      // charge the credit card
      yield* _(chargeCreditCard(cardNumber, totalAmount))
      // create a tracking id
      const trackingId = yield* _(createShippingTrackingCode(deliveryAddress))
      // send the order to shipment
      yield* _(sendOrderToShipping(orderId, trackingId))
      // send a confirmation email
      yield* _(sendConfirmationEmail(email, orderId, trackingId))
    })
```

Failures along the way

```
function chargeCreditCard(cardNumber, totalAmount): Effect<void, InsufficientFundsError | PaymentGateway
function createShippingTrackingCode(deliveryAddress): Effect<TrackingId, NoMoreApiCallQuotaError>
function sendConfirmationEmail(email, orderId, trackingId): Effect<void, SmtpFailureError>

const processPayment: Effect<
    void,
    InsufficientFundsError | PaymentGateway503Error | NoMoreApiCallQuotaError | SmtpFailureError
  > = ...
```

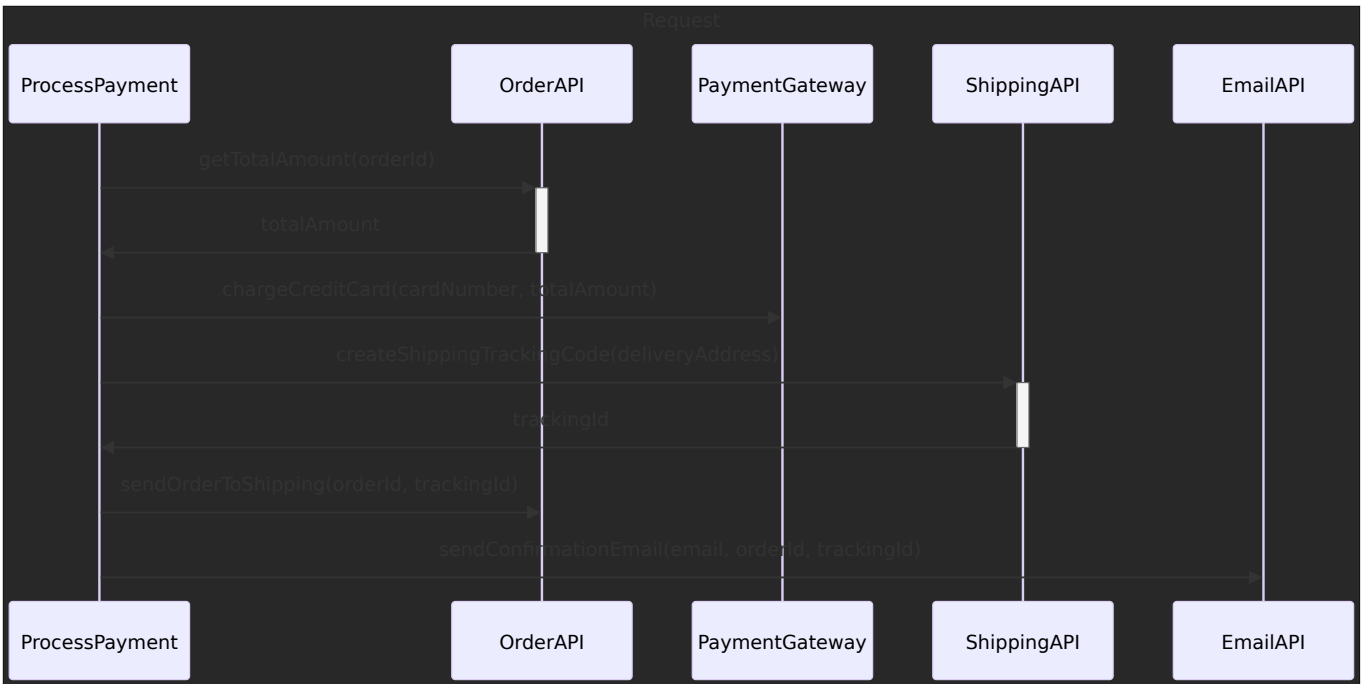
Log on failure

```
const processPayment =
  (cardNumber: CardNumber, deliveryAddress: DeliveryAddress,
  email: EmailAddress, orderId: OrderId) ⇒
    Effect.gen(function*(_) {
      const totalAmount = yield* _(getTotalAmount(orderId))
      yield* _(chargeCreditCard(cardNumber, totalAmount))
      const trackingId = yield* _(createShippingTrackingCode(deliveryAddress))
      // ^- failure raised here!
      yield* _(sendOrderToShipping(orderId, trackingId)) // skipped
      yield* _(sendConfirmationEmail(email, orderId, trackingId)) // skipped
    })
```

Retrying everything

```
const processPayment =
  (cardNumber: CardNumber, deliveryAddress: DeliveryAddress,
  email: EmailAddress, orderId: OrderId) ⇒
    Effect.gen(function*(_) {
      const totalAmount = yield* _(getTotalAmount(orderId))
      yield* _(chargeCreditCard(cardNumber, totalAmount))
      const trackingId = yield* _(createShippingTrackingCode(deliveryAddress))
      // ^- failure raised here!
      yield* _(sendOrderToShipping(orderId, trackingId)) // skipped
      yield* _(sendConfirmationEmail(email, orderId, trackingId)) // skipped
    }).pipe(
      Effect.retry({
        while: error ⇒ isTemporaryError(error)
      })
    )
```

Business Process



Transactions?

```
BEGIN TRANSACTION;
UPDATE card_balances SET balance = balance - 10 WHERE card_number = 42
UPDATE orders SET tracking_id = 'abc' WHERE order_id = 12
COMMIT TRANSACTION;
```

Distributed systems are everywhere

The user is a system as well

Welcome Effect Cluster

All the building blocks you need to deal with distributed workflows with ease!

SAGAS

Hector Garcia-Molina
Kenneth Salem

Department of Computer Science
Princeton University
Princeton, N J 08544

bstract

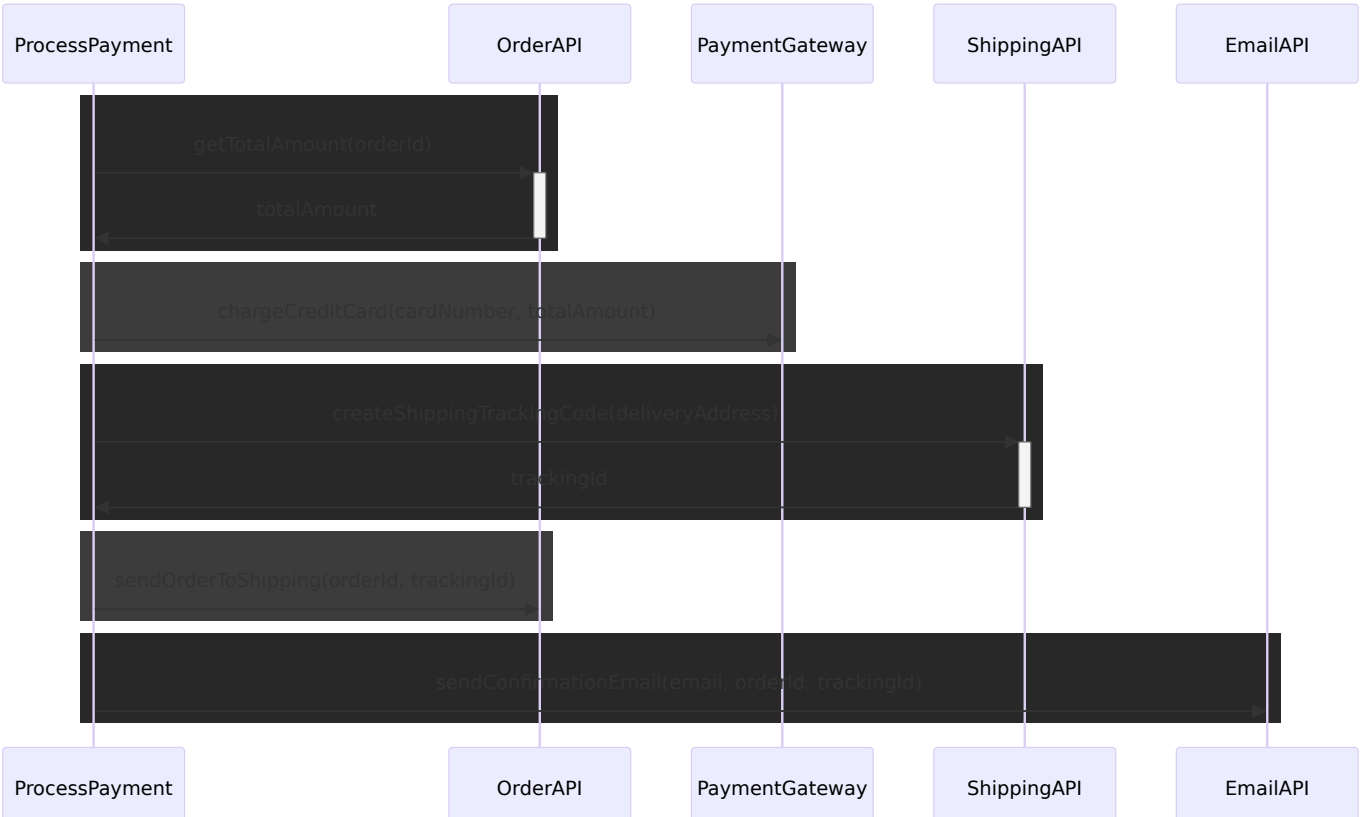
sactions (LLTs) hold on to
or relatively long periods of
elaying the termination of
common transactions To
ms we propose the notion of
aga if it can be written as a
ons that can be interleaved
ons The database manage-
ees that either all the tran-
re successfully completed or
ctions are run to amend a
oth the concept of saga and

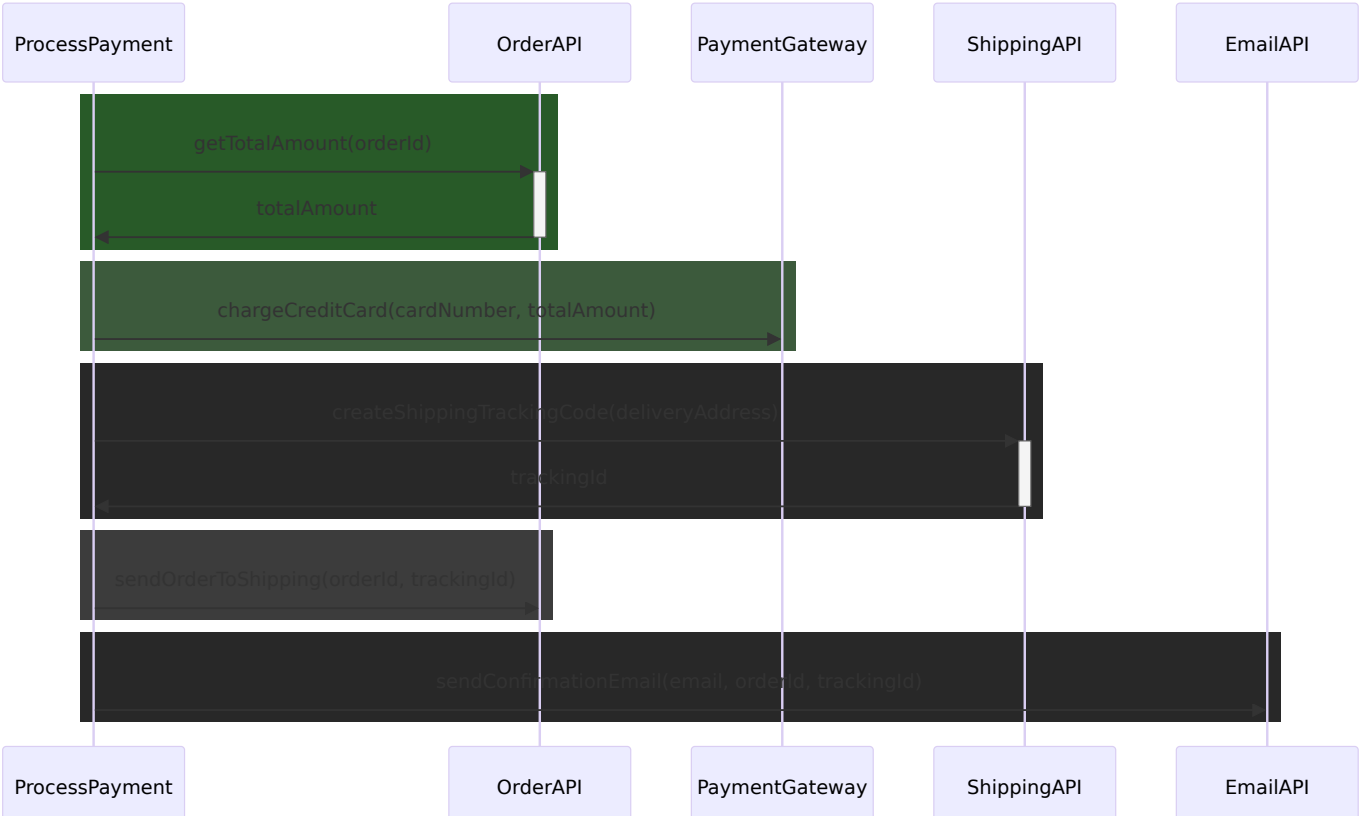
the majority of other tran
it accesses many database
computations, it pauses fo
or a combination of these
LLTs are transactions
account statements at a
process claims at an ins
transactions to collect st
database [Gray81a]

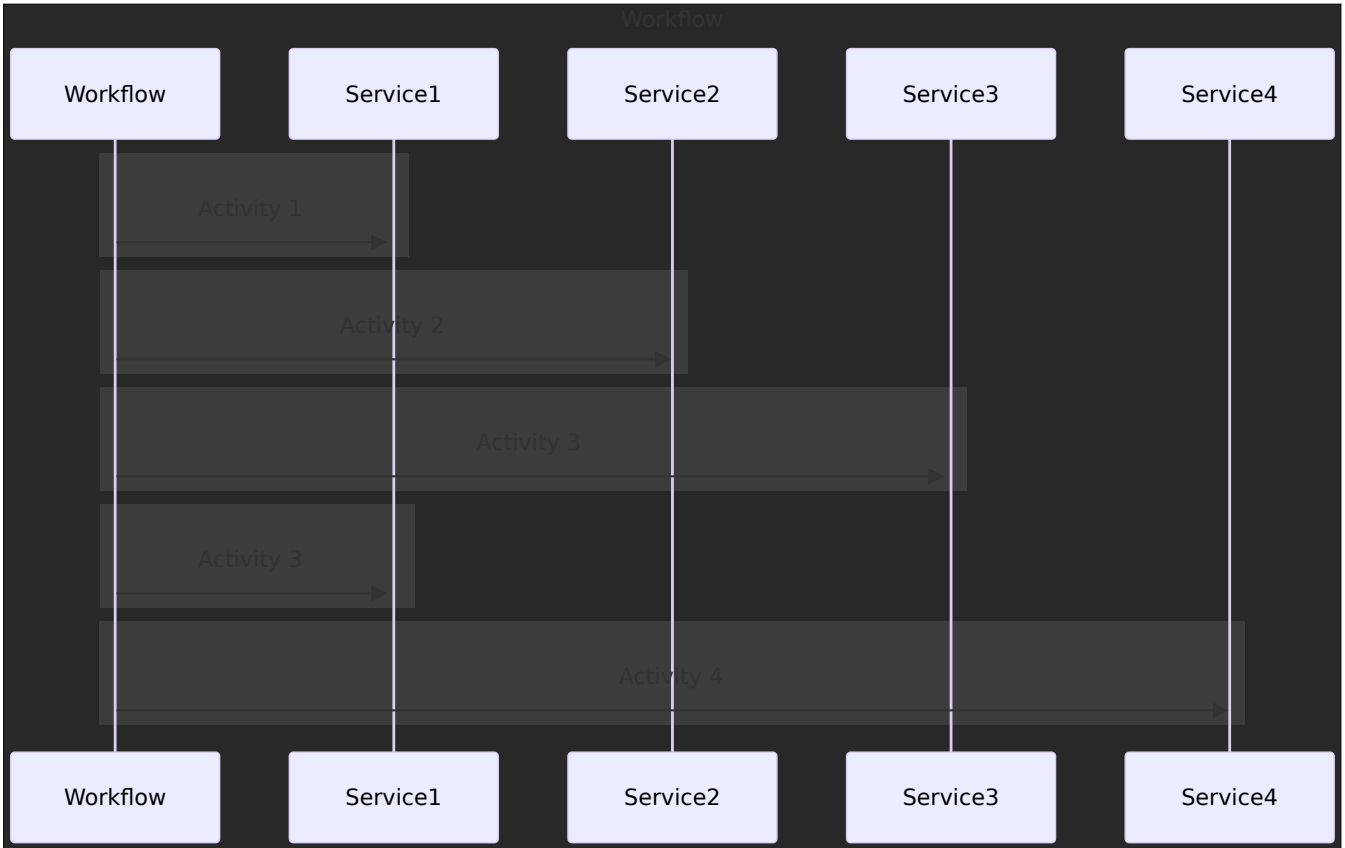
In most cases, LLTs
mance problems Since th
system must execute them
preserving the cor

...long lived transactions?

- Sagas!
- in 1987 they had computers!


ProcessPayment
OrderAPI
PaymentGateway
ShippingAPI
EmailAPI
getTotalAmount(orderId)
totalAmount
chargeCreditCard(cardNumber, totalAmount)
createShippingTrackingCode(deliveryAddress)
trackingId
sendOrderToShipping(orderId, trackingId)
sendConfirmationEmail(email, orderId, trackingId)
ProcessPayment
OrderAPI
PaymentGateway
ShippingAPI
EmailAPI


ProcessPayment
OrderAPI
PaymentGateway
ShippingAPI
EmailAPI
getTotalAmount(orderId)
totalAmount
chargeCreditCard(cardNumber, totalAmount)
createShippingTrackingCode(deliveryAddress)
trackingId
sendOrderToShipping(orderId, trackingId)
sendConfirmationEmail(email, orderId, trackingId)
ProcessPayment
OrderAPI
PaymentGateway
ShippingAPI
EmailAPI


Workflow
Workflow
Service1
Service2
Service3
Service4
Activity 1
Activity 2
Activity 3
Activity 3
Activity 4
Workflow
Service1
Service2
Service3
Service4

Defining an activity

- Work unit of a Workflow
- Can interact with external systems
- Executes an Effect
- Uniquely identified inside Workflow
- Requires schemas for success and failure
- Will be retried upon restarts

```
const getTotalAmount = (id: string) =>
  pipe(
    Http.request.get(`/get-total-amount/${id}`)
    .pipe(
      Http.client.fetchOk(),
      Effect.andThen((response) => response.json),
      Effect.mapError(() => ({
        code: 500,
        message: "API Fetch error"
        })
      )
    ),
    Activity.make(
      "get-amount-due", // identifier
      Schema.number, // success schema
      Schema.struct({ code: Schema.number, message:
    )
  )
```

Defining a workflow

- Is started by a Request
- Is identified by a globally unique id
- Requires schemas for success and failure
- Has a payload of information

```
class ProcessPaymentRequest
  extends Schema.TaggedRequest<ProcessPaymentRequest
  "ProcessPaymentRequest",
  Schema.never, // failure
  Schema.boolean, // success
  {
    orderId: Schema.string,
    cardNumber: Schema.string,
    email: Schema.string,
    deliveryAddress: Schema.string
  }
) {
}
```

Defining a workflow

```
const processPaymentWorkflow = Workflow.make(
  ProcessPaymentRequest,
  (_) ⇒ "ProcessPayment@" + _.orderId,
  ({ cardNumber, deliveryAddress, email, orderId }) ⇒
    Effect.gen(function*(_) {
      const totalAmount = yield* _(getTotalAmount(orderId))
      yield* _(chargeCreditCard(cardNumber, totalAmount))
      const trackingId = yield* _(createShippingTrackingCode(deliveryAddress))
      yield* _(sendOrderToShipping(orderId, trackingId))
      yield* _(sendConfirmationEmail(email, orderId, trackingId))
    })
)
```

- Coordinator of activities
- Durable execution
- Requires deterministic code

WTF is deterministic?

Given a set of input, the output of the function must be always the same and predictable, without triggering any side effects that may later affect the computation.

Deterministic

- Math & Logic ops
- Seed based random

Non-Deterministic

- Math.random()
- new Date()
- R/W Global Shared State (also DB)

Determinism & Workflows

```
const processPaymentWorkflow = Workflow.make(
  ProcessPaymentRequest,
  (_) => "ProcessPayment@" + _.orderId,
  ({ cardNumber, deliveryAddress, email, orderId }) =>
    Effect.gen(function*(_) {
      const totalAmount = yield* _(Effect.succeed(42.1) /* getTotalAmount(orderId) */)
      yield* _(Effect.unit /* chargeCreditCard(cardNumber, totalAmount) */)
      const trackingId = yield* _(createShippingTrackingCode(deliveryAddress))
      yield* _(sendOrderToShipping(orderId, trackingId))
      yield* _(sendConfirmationEmail(email, orderId, trackingId))
    })
)
```

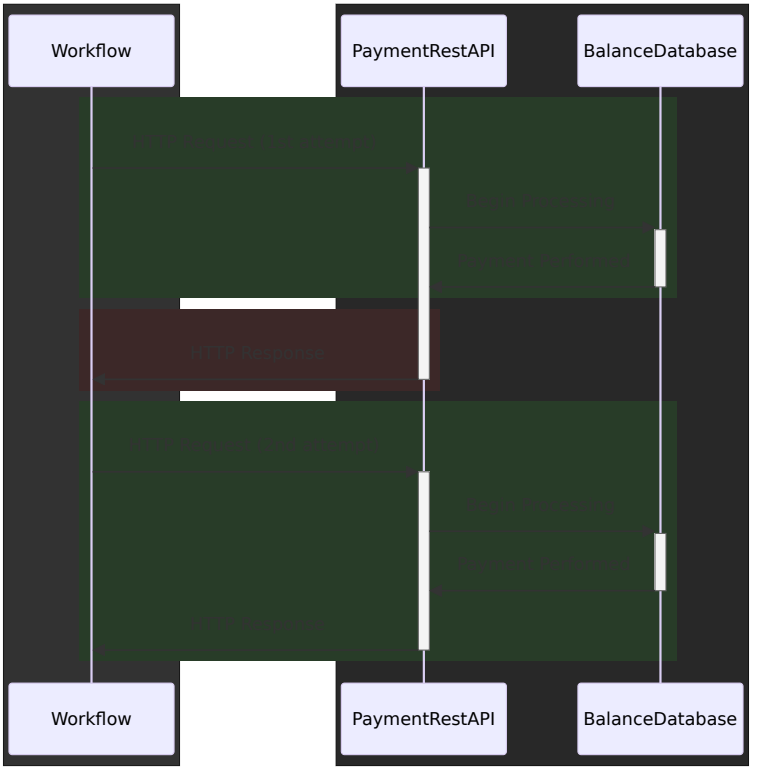
Running a Workflow

```
const main = Effect.gen(function*(_) {
  const workflows = Workflow.union(processPaymentWorkflow, requestRefundWorkflow)
  const engine = yield* _(WorkflowEngine.makeScoped(workflows))
  yield* _(
    engine.sendDiscard(
      new ProcessPaymentRequest({
        orderId: "order-1",
        cardNumber: "my-card",
        deliveryAddress: "My address, 5, Italy",
        email: "my@email.com"
      })
    )
  )
})
runMain(
  pipe(
    main,
    Effect.provide(DurableExecutionJournalPostgres.DurableExecutionJournalPostgres)
  )
)
```

Activity is a Black Box

The double-payment problem

- Any transport may fail
- Not getting a response does not mean it has not been processed

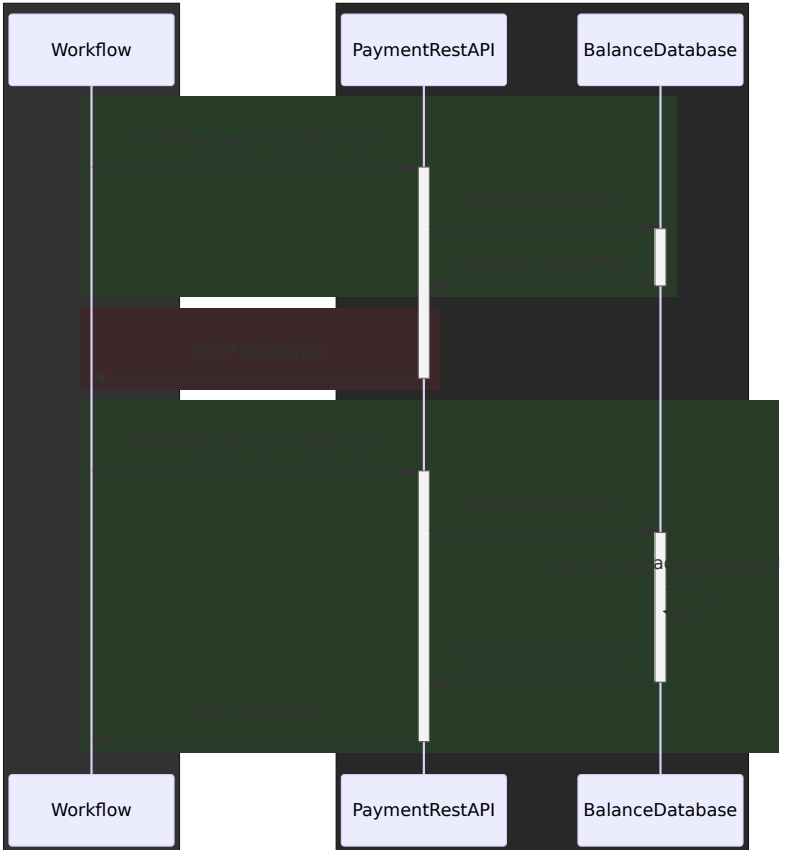


Idempotency

Multiple invocation of the same function, result on a state change as they were executed only the first time.

Idempotence using IDs

```
const chargeCreditCard =
 (cardNumber: string, amountDue: number) ⇒
  pipe(
    Activity.persistenceId,
    Effect.flatMap(
      (persistenceId) ⇒ callPaymentGateway(
        persistenceId,
        cardNumber,
        amountDue
      )
    ),
    Activity.make(
      "charge-credit-card",
      Schema.void,
      Schema.never
    )
  )
```



```ts
const createShippingTrackingCode = (deliveryAddress: string) =>
  pipe(
    ShipmentApiConfig,
    Effect.flatMap((apiConfig) => callShipmentApi(apiConfig.apiSecret, apiConfig.apiSecret, deliveryAddress)),
    Activity.make("create-tracking-code", Schema.string, Schema.never)
  )
```

cluster-pg — node /opt/homebrew/bin/pnpm example examples/simple-workflow.ts — 144×22

```
mattiamanzati@MacBook-Pro-di-Mattia-2 cluster-pg % pnpm example examples/simple-workflow.ts
```

Yielding workflow execution

```
const getTotalAmount = (id: string) ⇒
  pipe(
    Http.request.get(`/get-total-amount/${id}`)
    .pipe(
      Http.client.fetchOk(),
      Effect.andThen((response) ⇒ response.json),
      Effect.retry(
        Schedule.exponential(1000).pipe(
          Schedule.compose(Schedule.recurs(32)),
        ),
      ),
      Effect.catchAllCause(() ⇒ pipe(
        Effect.logError("Something is wrong with the OrderAPI right now"),
        Effect.zipRight(Workflow.yieldExecution)
      ))
    ),
    Activity.make("get-amount-due", Schema.number, Schema.never)
  )
```

Fixing workflows

```
const processPaymentWorkflow = Workflow.make(
  ProcessPaymentRequest,
  (_) ⇒ "ProcessPayment@" + _.orderId,
  ({ cardNumber, deliveryAddress, email, orderId }) ⇒
    Effect.gen(function*(_) {
      const totalAmount = yield* _(getTotalAmount(orderId))
      yield* _(chargeCreditCard(cardNumber, totalAmount))
      const trackingId = yield* _(createShippingTrackingCode(deliveryAddress))
      yield* _(sendOrderToShipping(orderId, trackingId))
      yield* _(sendConfirmationEmail(email, orderId, trackingId))
    })
)
```

All or Nothing

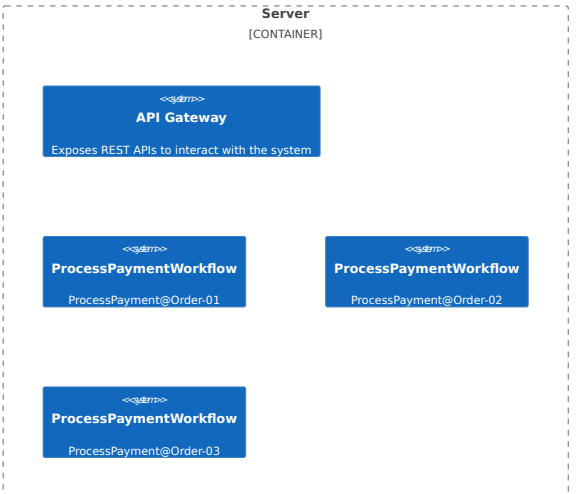
Compensating actions

```
const processPaymentWorkflow = Workflow.make(
  ProcessPaymentRequest,
  (_) ⇒ "ProcessPayment@" + _.orderId,
  ({ cardNumber, deliveryAddress, email, orderId }) ⇒
    pipe(
      getTotalAmount(orderId),
      Effect.flatMap(totalAmount ⇒ pipe(
        chargeCreditCard(cardNumber, totalAmount),
        Effect.flatMap(() ⇒ createShippingTrackingCode(deliveryAddress)),
        Effect.tap(trackingId ⇒ sendOrderToShipping(orderId, trackingId)),
        Effect.tap(trackingId ⇒ sendConfirmationEmail(email, orderId, trackingId)),
        Effect.catchTag("OutOfStockError", () ⇒ refundCreditCard(cardNumber, totalAmount))
      ))
    )
)
```

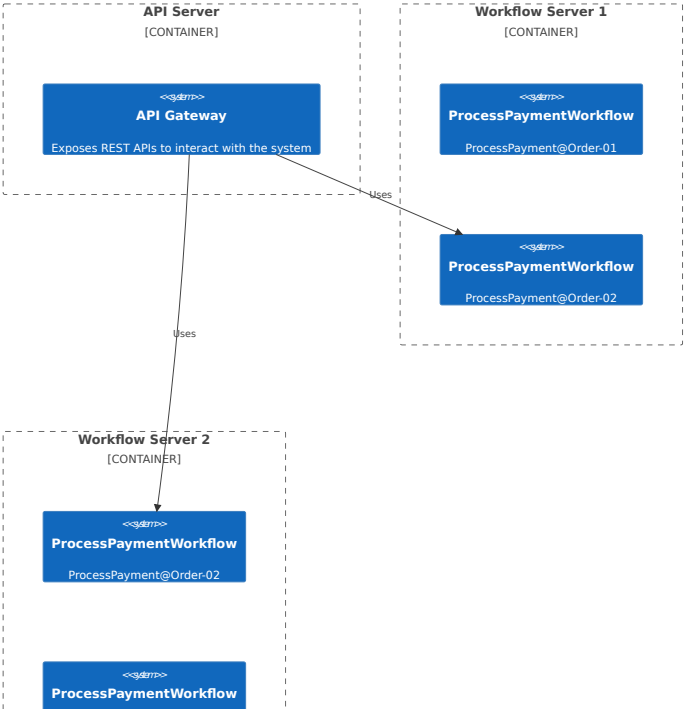
Effect Cluster Workflows

Building durable and reliable Effects for your applications

Scaling the system



Ensuring only one execution of each workflow instance



The restaurant problem

- Table 1: Waiter Mattia
- Table 2: Waiter Mattia
- Table 3: Waiter Mattia
- Table 4: Waiter Mattia
- Table 5: Waiter Mattia
- Table 6: Waiter Mattia

Tables sharding

- Table 1: Waiter Mattia
- Table 2: Waiter Mattia
- Table 3: Waiter Mattia
- Table 4: Waiter Michael
- Table 5: Waiter Michael
- Table 6: Waiter Michael

Effect Cluster Sharding and Location Transparency

Safely distribute work and refer to entity without knowing their location