Durable Workflows

with Effect Cluster

THANK YOU ORGANIZERS!

Ego Slide

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

Failures along the way

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

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

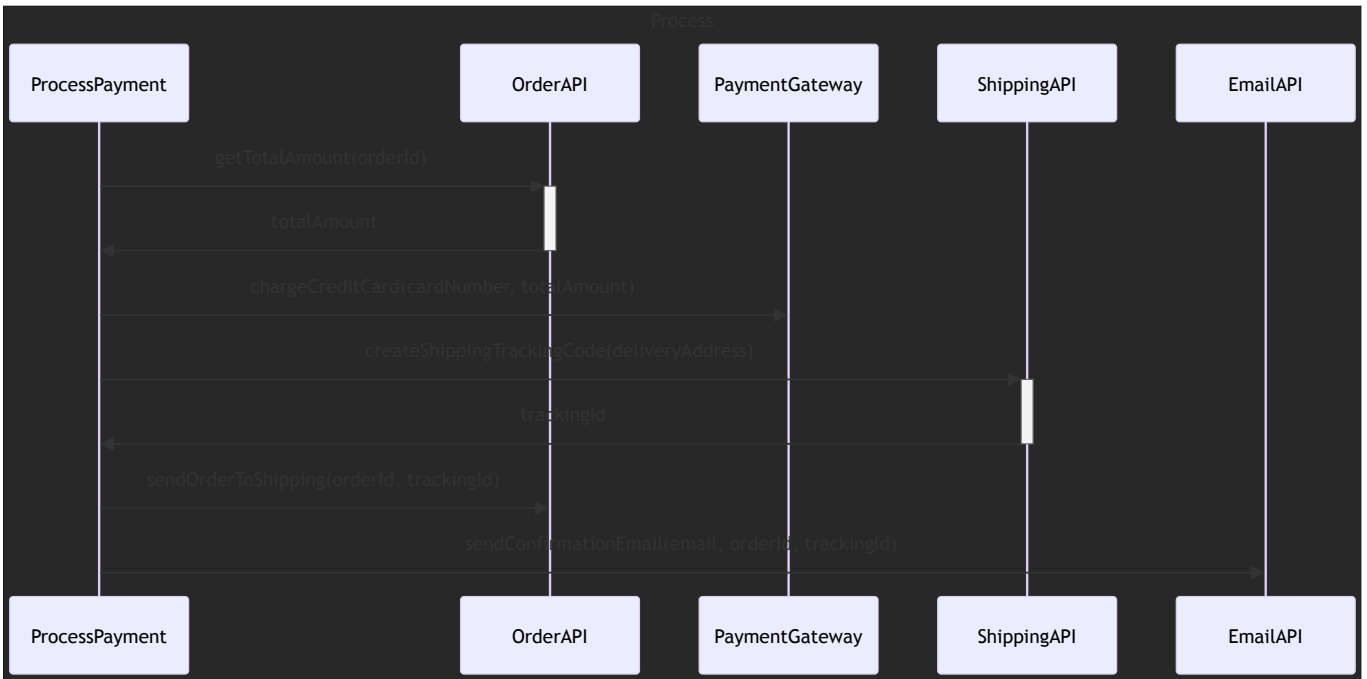
Log on failure

```
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

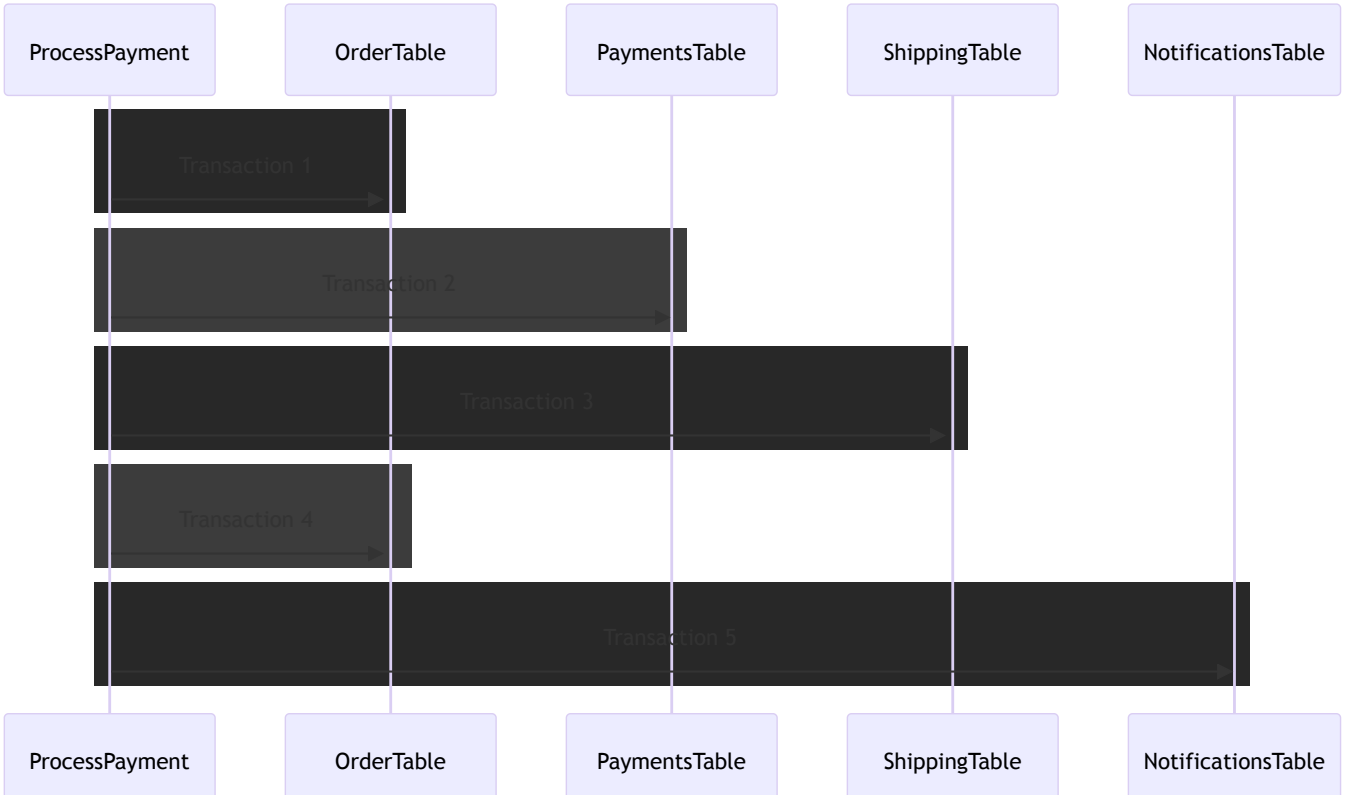
Distributed Workflows are everywhere

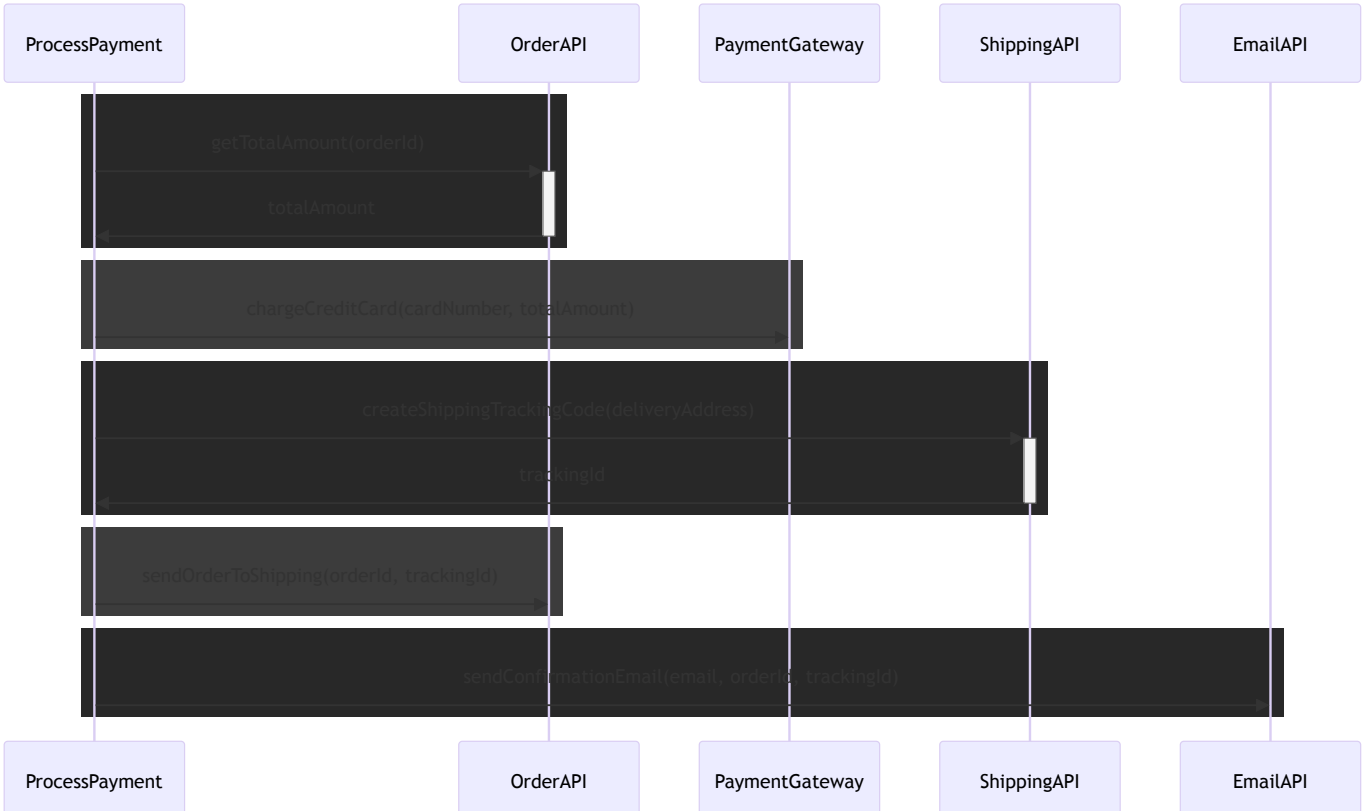
Welcome Effect Cluster

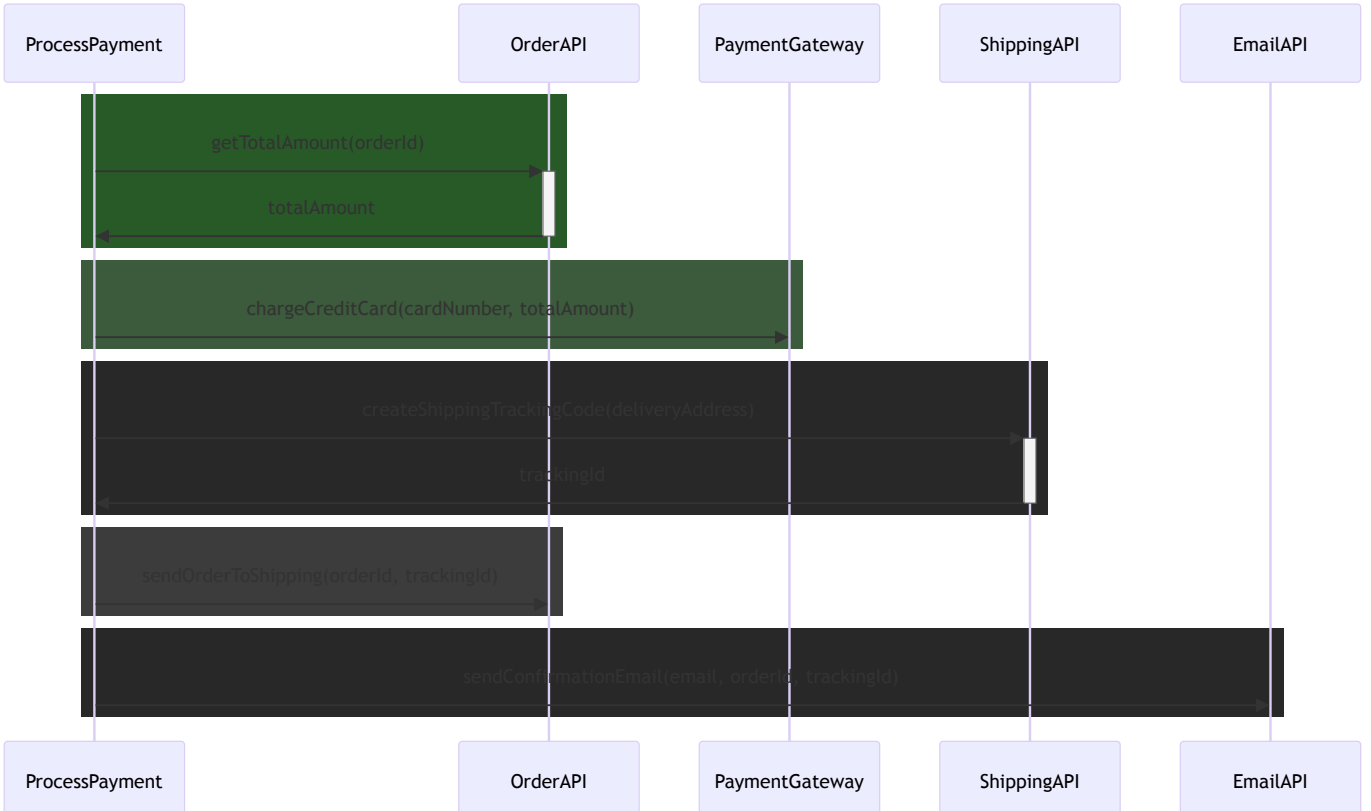
All the building blocks you need to deal with distributed workflows with ease!

…long lived transactions?

- Sagas!
- in 1987 they had computers!


ProcessPayment
OrderTable
PaymentsTable
ShippingTable
NotificationsTable
Transaction 1
Transaction 2
Transaction 3
Transaction 4
Transaction 5
ProcessPayment
OrderTable
PaymentsTable
ShippingTable
NotificationsTable


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

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

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

Defining a workflow

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

```
class ProcessPaymentRequest extends
  Schema.TaggedRequest<ProcessPaymentRequest>()(
  `ProcessPaymentRequest`, // tag
  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))
    })
)
```

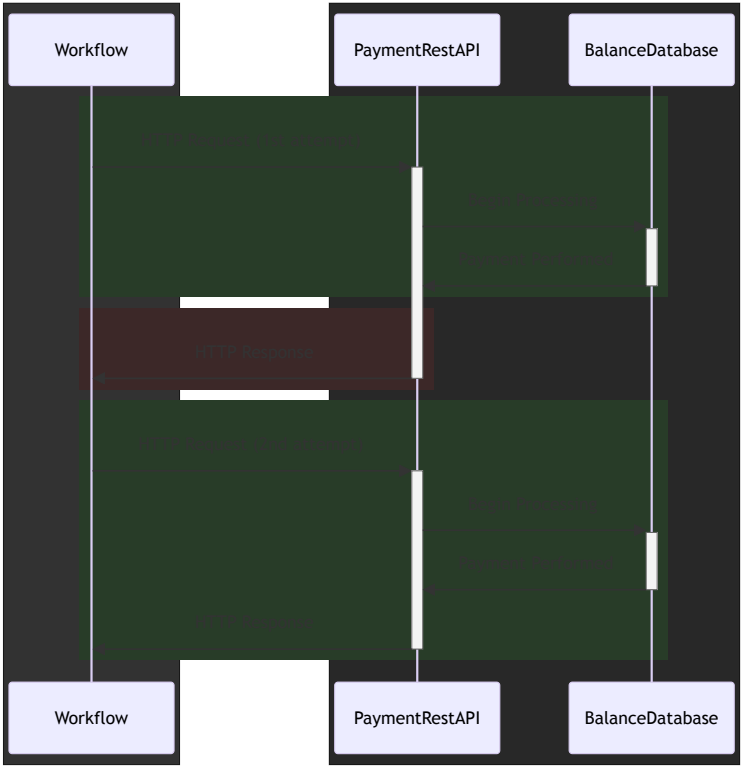
Running a Workflow

```
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

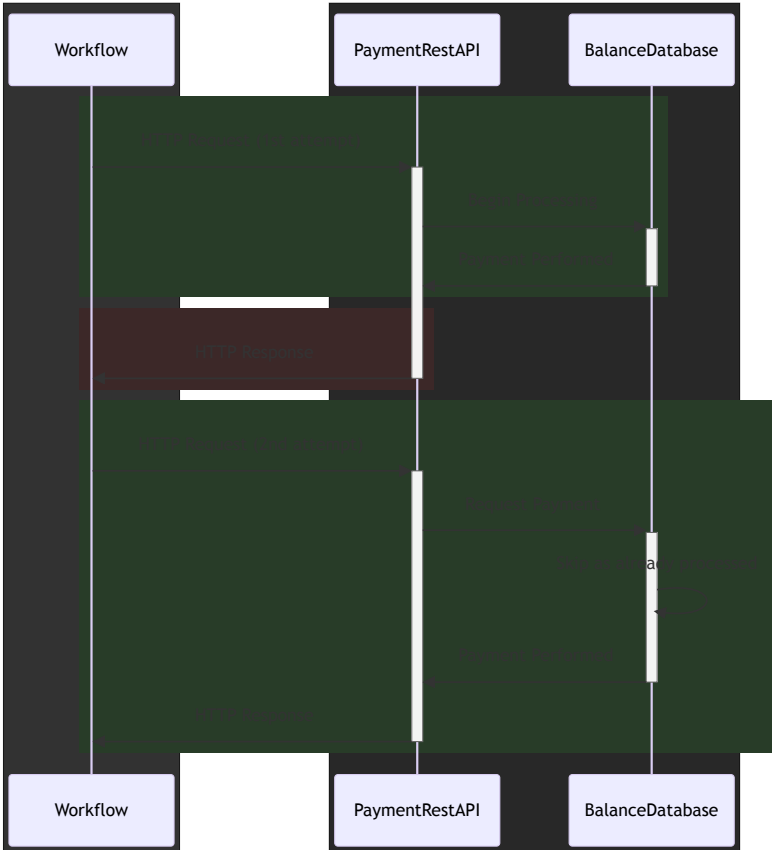


Idempotency

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
    )
  )
```



Yielding workflow execution

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

Fixing workflows

```
const processPaymentWorkflow = Workflow.make(
  ProcessPaymentRequest,
  (_) => `ProcessPayment@${_.orderId}`,
  ({ cardNumber, deliveryAddress, email, orderId }) =>
    Effect.gen(function*($) {
      const version = yield* $(getCurrentVersion(2))
      // ^- for already running WFs, will resolve with previous value (1)
      // ...
      if(version >= 2){
        yield* $(checkTrustedCardNumber(cardNumber))
      }
      yield* $(chargeCreditCard(cardNumber, totalAmount))
      // ...
    })
)

const getCurrentVersion = (definitionVersion: number) => pipe(
  Effect.succeed(definitionVersion),
  Activity.make("get-current-version", Schema.number, Schema.void)
)
```

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

Effect Cluster Sharding and Location Transparency

Safely distribute work and refer to entity without knowing their location

```ts
import * as RecipientType from "@effect/cluster/RecipientType"
import * as Schema from "@effect/schema/Schema"

export class Increment extends Schema.TaggedClass<Increment>()("Increment", {
  messageId: Schema.string
}) {}

export class Decrement extends Schema.TaggedClass<Decrement>()("Decrement", {
  messageId: Schema.string
}) {}

export class GetCurrent extends
  Schema.TaggedRequest<GetCurrent>()("GetCurrent", Schema.never, Schema.number, {
  messageId: Schema.string
}) {}

export const CounterMsg = Schema.union(Increment, Decrement, GetCurrent)
export type CounterMsg = Schema.Schema.To<typeof CounterMsg>

export const CounterEntity = RecipientType.makeEntityType("Counter", CounterMsg, (_) => _.messageId)
```

```
Sharding.registerEntity(
  CounterEntity
)(
  RecipientBehaviour.fromFunctionEffectStateful(
    () ⇒ Effect.succeed(0),
    (_, message, stateRef) ⇒ {
      switch (message._tag) {
        case "Increment":
          return pipe(Ref.update(stateRef, (count) ⇒ count + 1), Effect.as(MessageState.Processed(Opt
        case "Decrement":
          return pipe(Ref.update(stateRef, (count) ⇒ count - 1), Effect.as(MessageState.Processed(Opt
        case "GetCurrent":
          return pipe(
            Ref.get(stateRef),
            Effect.exit,
            Effect.map((result) ⇒ MessageState.Processed(Option.some(result)))
          )
      }
    }
  )
)
```

Today's Recap

Effect cluster provides all the basic building blocks you'll need to build distributed workflows and systems

`@effect/cluster`

- Sharding
- Shard Manager
- Messaging utilities

`@effect/cluster-node`

- Cluster specific utilities for NodeJS

`@effect/cluster-workflow`

- Activity
- Workflow
- WorkflowEngine

`@effect/cluster-pg`

- Postgres based persistency for Workflow and Sharding

Happy Effect-ing!

Thanks for your time!