The simplest way to build resilient applications

Giselle van Dongen

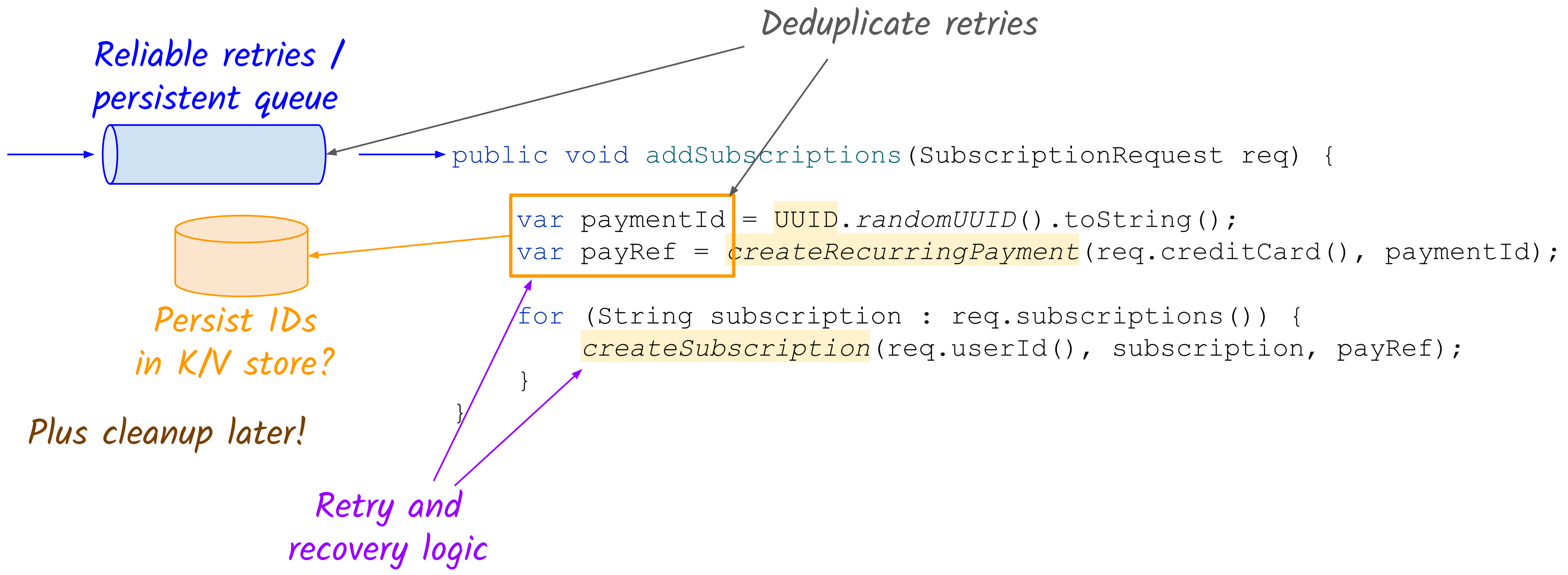
restate

Drop your questions in the Wova Session Q&A!

Let’s start with a simple example…

```
public class User {

    public void addSubscriptions(SubscriptionRequest req) {

        var paymentId = UUID.randomUUID().toString();
        var payRef = createRecurringPayment(req.creditCard(), paymentId);

        for (String subscription : req.subscriptions()) {
            createSubscription(req.userId(), subscription, payRef);
        }
    }

}
```

→ *How can we make this really reliable?*



```
public void addSubscriptions(SubscriptionRequest req) {

    var paymentId = UUID.randomUUID().toString();
    var payRef = createRecurringPayment(req.creditCard(), paymentId);

    for (String subscription : req.subscriptions()) {
        createSubscription(req.userId(), subscription, payRef);
    }
}
```

Writing resilient systems is hard…

Race conditions

Zombie processes

Half-executed orchestration

Corrupted state

Timeouts

Network partitions

Duplicate requests

Concurrency

Distributed transactions

Scalability

Rf. the IT crowd

Duct-taping it all together

Session K/V stores
for app state

Message queues
for async events

Workflow orchestrators
for execution progress

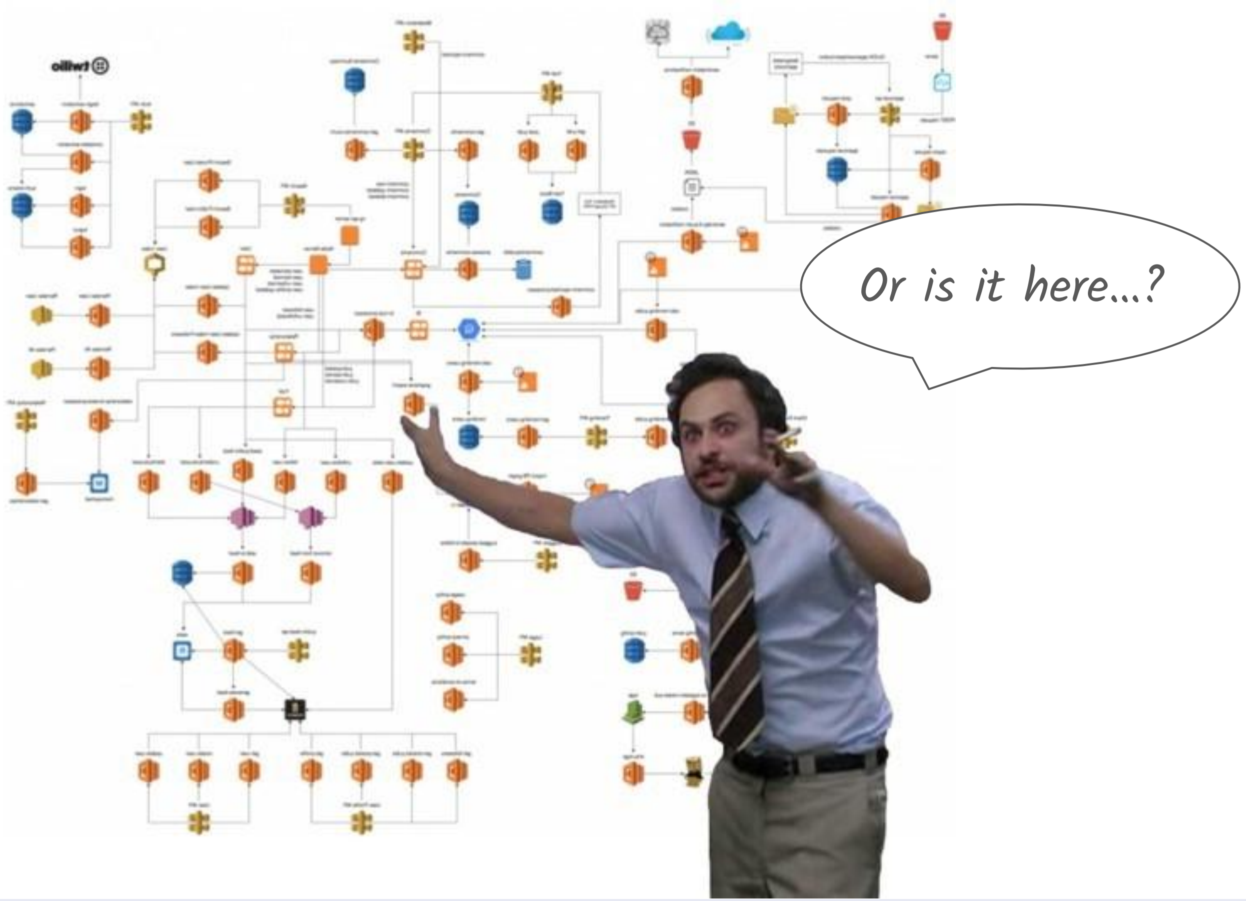
Schedulers
for timers

Manual retry & recovery logic

Don't solve it!

The end...

The end...



Duct-taping it all together

Session K/V stores
for app state

amazon DynamoDB redis

Message queues
for async events

kafka SQS

Workflow orchestrators
for execution progress

AWS StepFunctions CAMUNDA

Schedulers
for timers

QUARTZ + PostgreSQL CRONJOBS

Manual retry & recovery logic

Don't solve it!

Duct-taping it all together

Session K/V stores
for app state

amazon DynamoDB
redis

Message queues
for async events

kafka
SQS

Workflow orchestrators
for execution progress

AWS StepFunctions
CAMUNDA

Schedulers
for timers

QUARTZ
+
PostgreSQL
CRONJO

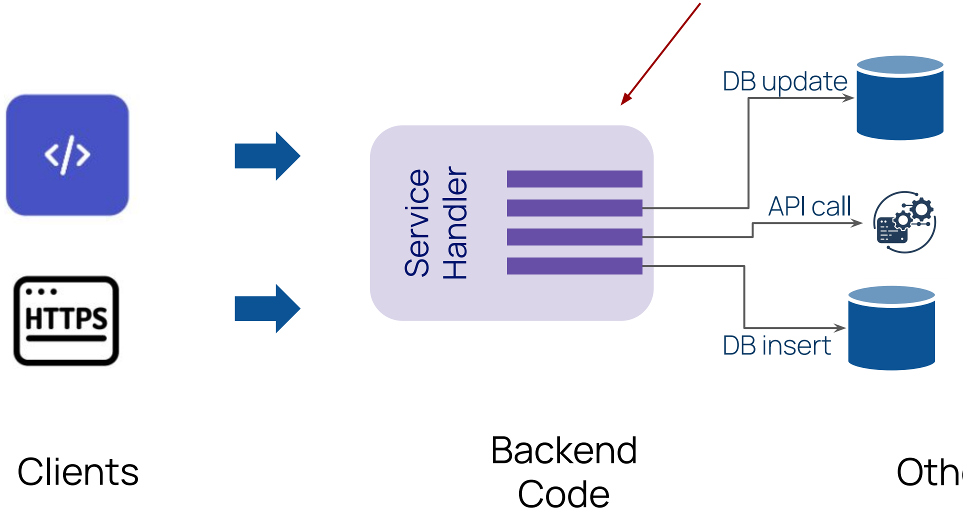
Manual retry &

Don't solve it!

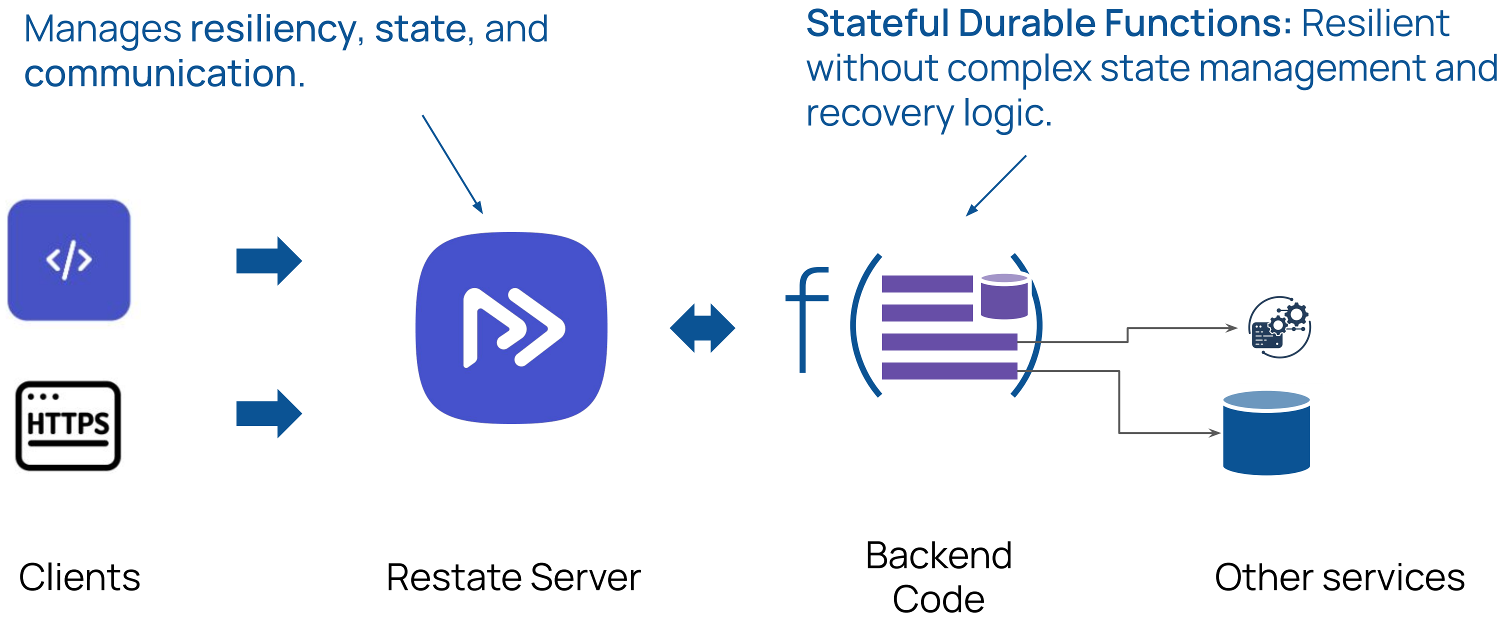
Observability tooling

https://landscape.cncf.io/

On failure, lose state, partial progress.
Resilience is up to the developer.



Restate makes applications innately resilient


Manages **resiliency, state,** and **communication.**
Stateful Durable Functions: Resilient without complex state management and recovery logic.
HTTPS
Clients
Restate Server
Backend Code
Other services

A hybrid between a workflow orchestrator and a message broker

Workflows
and sagas

Concurrent
async tasks

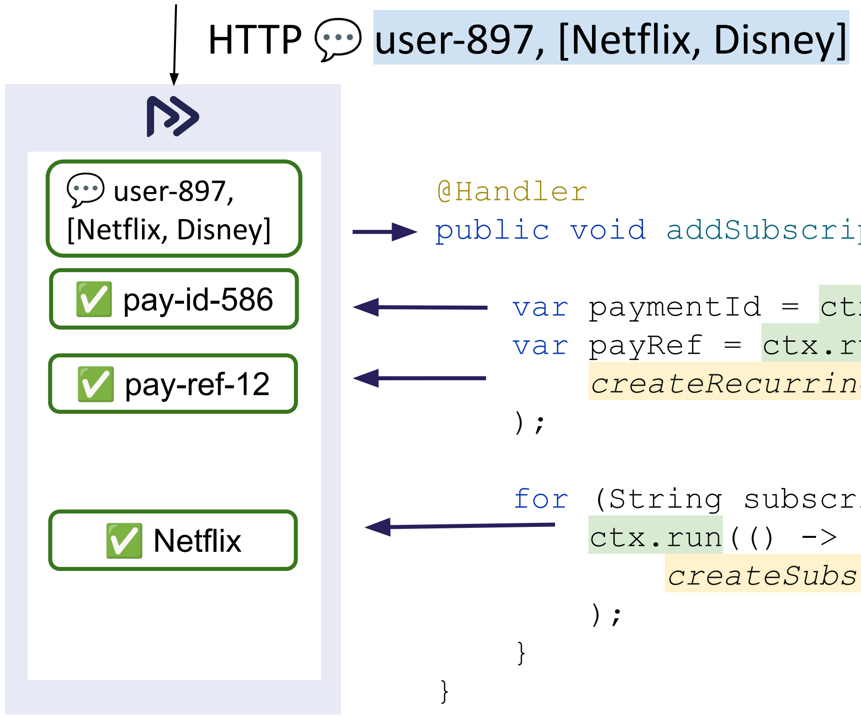
State machines,
agents, actors

Kafka event
processing

Durable Execution

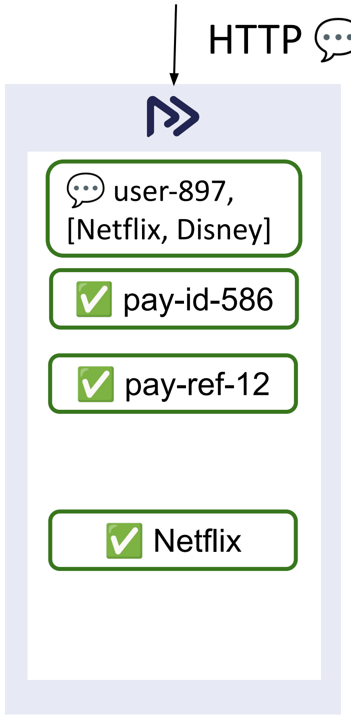
💡 Built-in retries and recovery of progress

```
@Service
public class User {

    @Handler
    public void addSubscriptions(Context ctx, SubscriptionRequest req) {

        var paymentId = ctx.run(() -> UUID.randomUUID().toString());
        var payRef = ctx.run(() -> createRecurringPayment(req.creditCard(), paymentId));

        for (String subscription : req.subscriptions()) {
            ctx.run(() -> createSubscription(req.userId(), subscription, payRef));
        }
    }

}
```

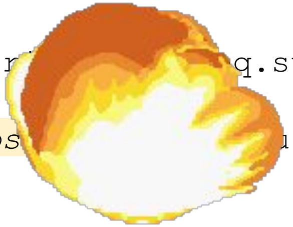

HTTP 💬 user-897, [Netflix, Disney]
💬 user-897, [Netflix, Disney]
✅ pay-id-586
✅ pay-ref-12
✅ Netflix

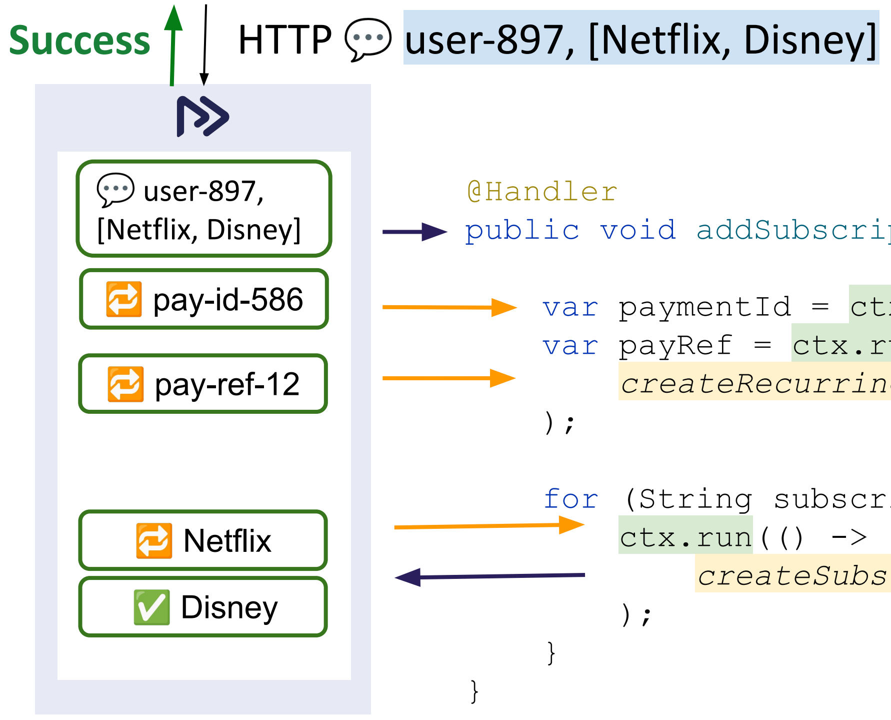
```
@Handler
public void addSubscriptions(Context ctx, SubscriptionRequest req) {

    var paymentId = ctx.run(() -> UUID.randomUUID().toString());
    var payRef = ctx.run(() ->
        createRecurringPayment(req.creditCard(), paymentId)
    );

    for (String subscription : req.subscriptions()) {
        ctx.run(() ->
            createSubscription(req.userId(), subscription, payRef)
        );
    }
}
```

HTTP 💬 user-897, [Netflix, Disney]



```
@Handler
public void addSubscriptions(Context ctx, SubscriptionRequest req) {

    var paymentId = ctx.run(() -> UUID.randomUUID().toString());
    var payRef = ctx.run(() ->
        createRecurringPayment(req.creditCard(), paymentId)
    );

    for (String subscr[illegible]q.subscriptions()) {
        ctx.run(() ->
            createSubs[illegible]userId(), subscription, payRef)
        );
    }
}
```




Success
HTTP 💬 user-897, [Netflix, Disney]
💬 user-897, [Netflix, Disney]
🔁 pay-id-586
🔁 pay-ref-12
🔁 Netflix
✅ Disney

```java
@Handler
public void addSubscriptions(Context ctx, SubscriptionRequest req) {

    var paymentId = ctx.run(() -> UUID.randomUUID().toString());
    var payRef = ctx.run(() ->
        createRecurringPayment(req.creditCard(), paymentId)
    );

    for (String subscription : req.subscriptions()) {
        ctx.run(() ->
            createSubscription(req.userId(), subscription, payRef)
        );
    }
}
```

DEMO

Writing resilient systems is hard...

Race conditions

Zombie processes

Half-executed orchestration

Corrupted state

Timeouts

Network partitions

Duplicate requests

Concurrency

Distributed transactions

Scalability

restate

Rf. the IT crowd

Writing resilient systems is a lot easier…

~~Race conditions~~

~~Zombie processes~~

~~Half-executed orchestration~~

~~Corrupted state~~

~~Timeouts~~

~~Network partitions~~

~~Duplicate requests~~

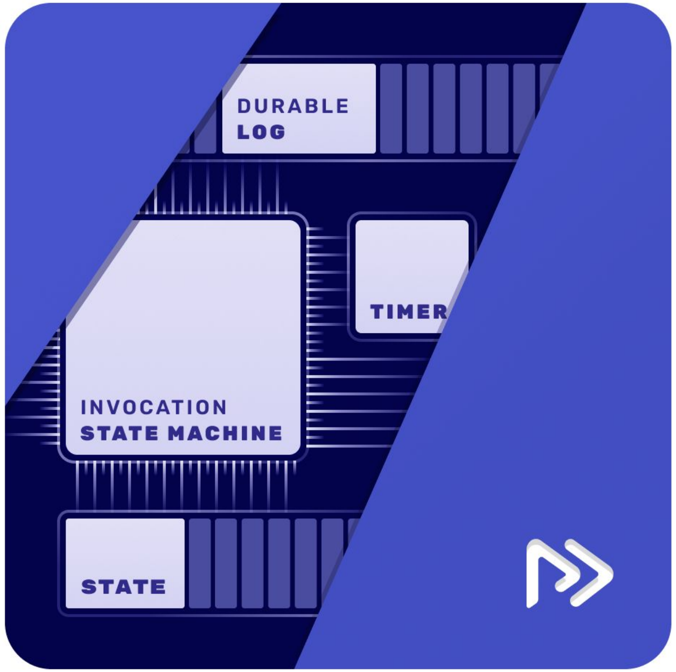
~~Concurrency~~

~~Distributed transactions~~

~~Scalability~~

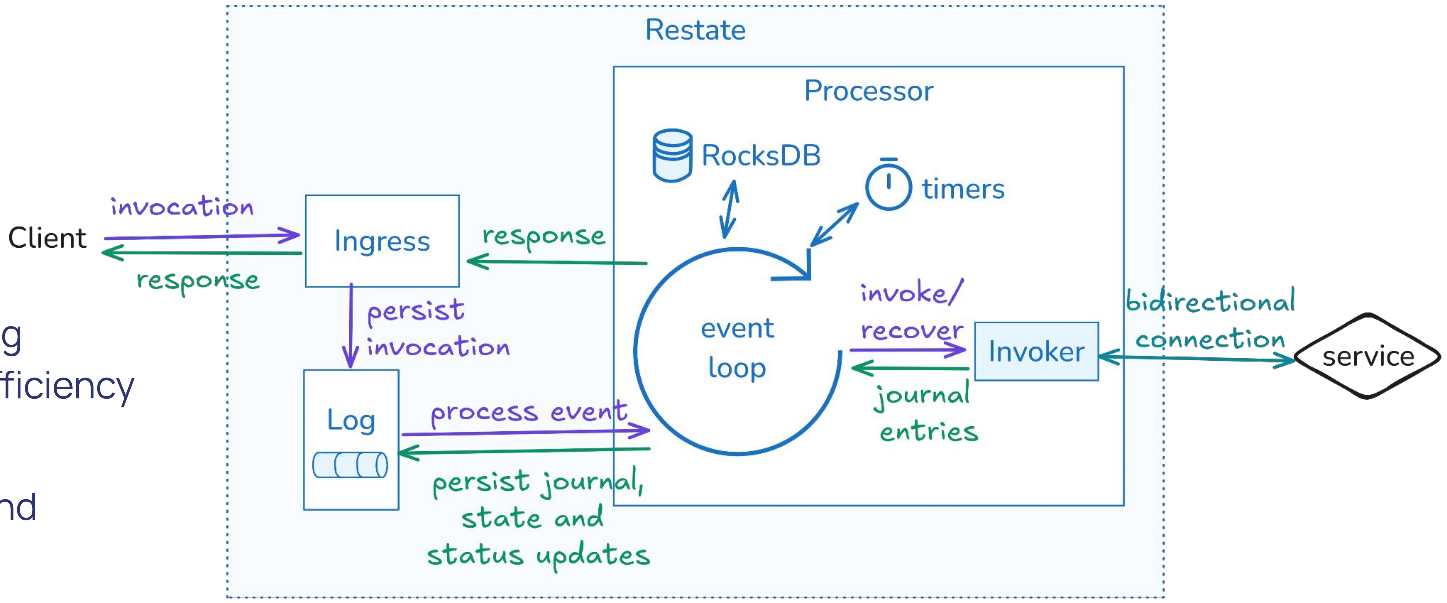
Rf. the IT crowd

- Single binary, written in Rust
- No need for database, queues, ...
- Distributed setup with snapshots to S3
- Cloud-native failover support
- Open Source



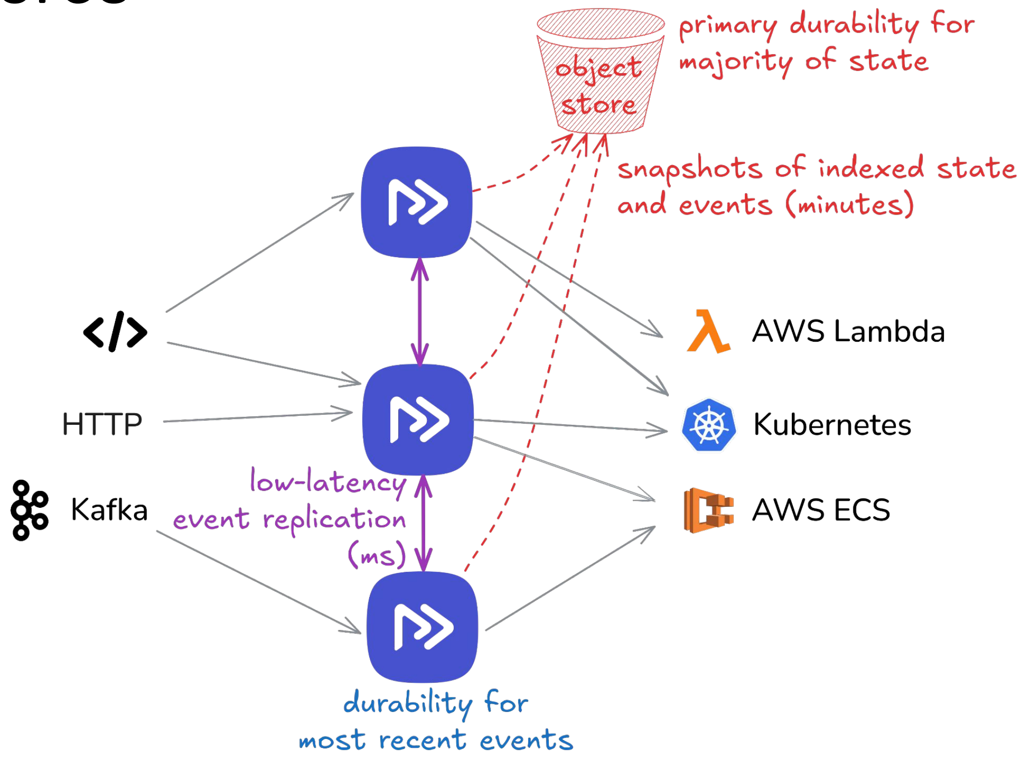
A transactional stateful stream processing engine

- Stream-processing architecture for efficiency and low latency
- Co-located logs and processors



Replication + object stores

- Restate clusters replicate events between nodes for low latency
- The majority of the data is stored on object store


object store
primary durability for majority of state
snapshots of indexed state and events (minutes)
</>
HTTP
Kafka
low-latency event replication (ms)
durability for most recent events
AWS Lambda
Kubernetes
AWS ECS

Currently Supported Languages

TypeScript / JavaScript

Java

Kotlin

Python

Go

Rust

What our community builds with Restate

Workflows/ SAGAs	AI inference	Event-driven Services	Ledgers
Service Orchestration	Payment Processing	Workflow interpreters	AI Agents
Webhook ingestion	Distributed Transactions	Control Planes	Stateful Event Processing

A computer lets you make more mistakes faster than any invention in human history, with the possible exceptions of handguns and tequila.
- Mitch Ratcliffe -

https://restate.dev

@restatedev

@restatedev.bsky.social

/restatedev

Drop your questions in the Wova Session Q&A!