KIRILL MERKUSHEV

ПРОЕКТ НА JAVA И REACTOR

А КАК ЖЕ ТЕСТЫ?

SR. JAVA ENGINEER

KIRILL MERKUSHEV

EX. YANDEX

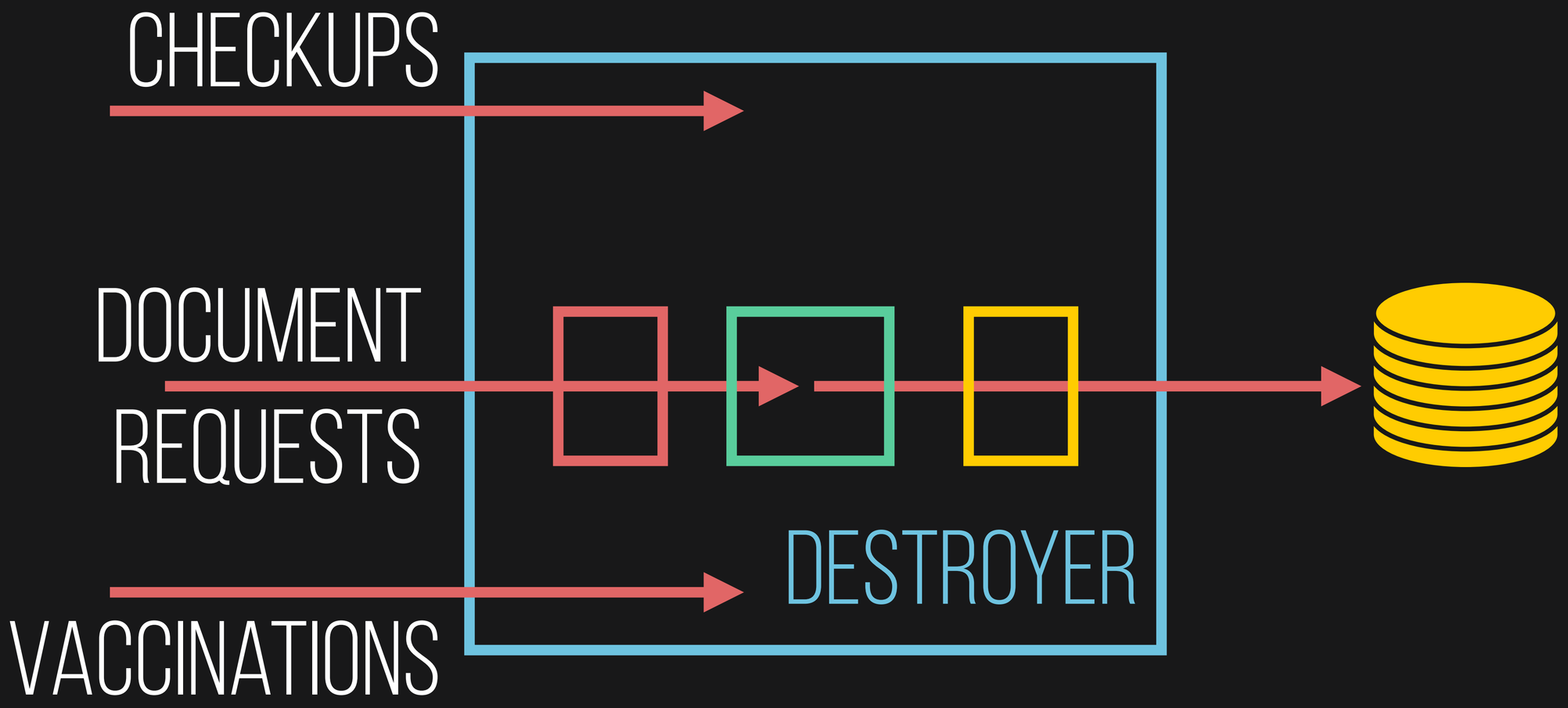
VIVY GMBH BERLIN

AEROKUBE.COM

OPEN SOURCE
JAVA
GO

PLATFORM FOR NEW AGE OF COMMUNICATION BETWEEN PATIENTS, DOCTORS AND INSURANCES

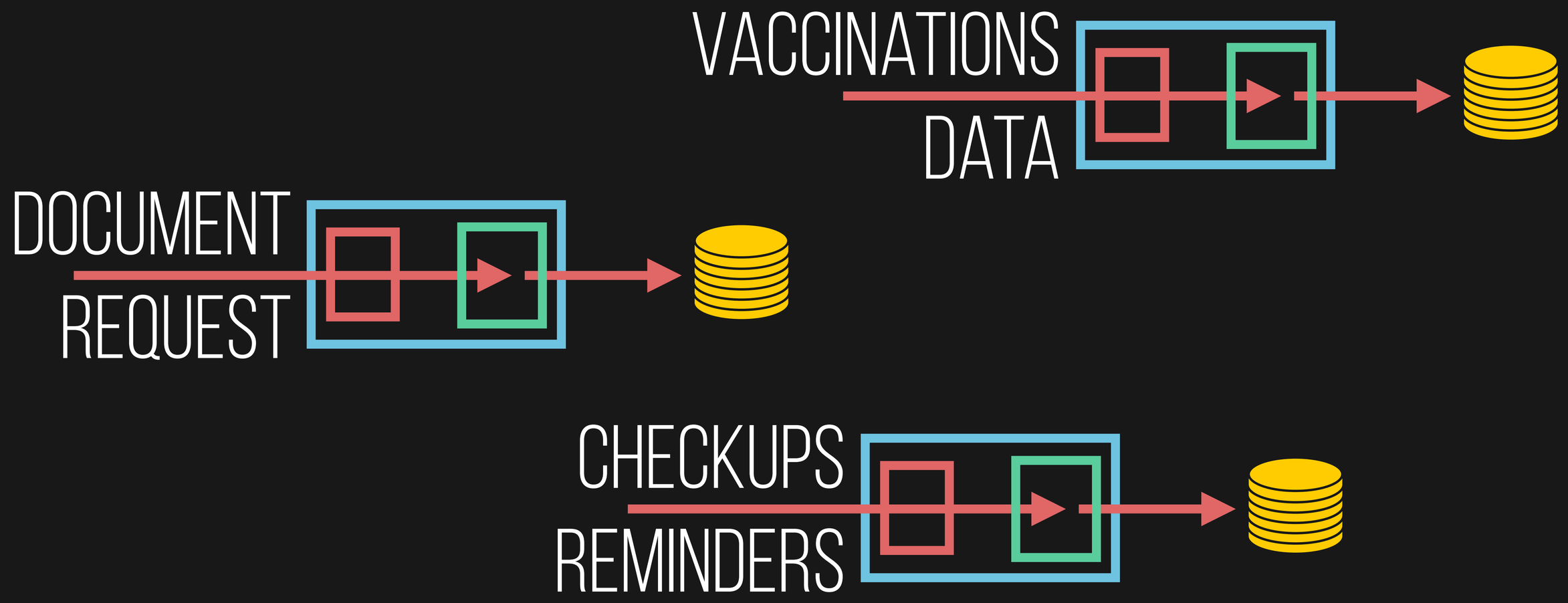
MONOLITH AS MVP



NON-SCALABLE

DEVELOPMENT
TESTING
LOAD

MICROSERVICES?



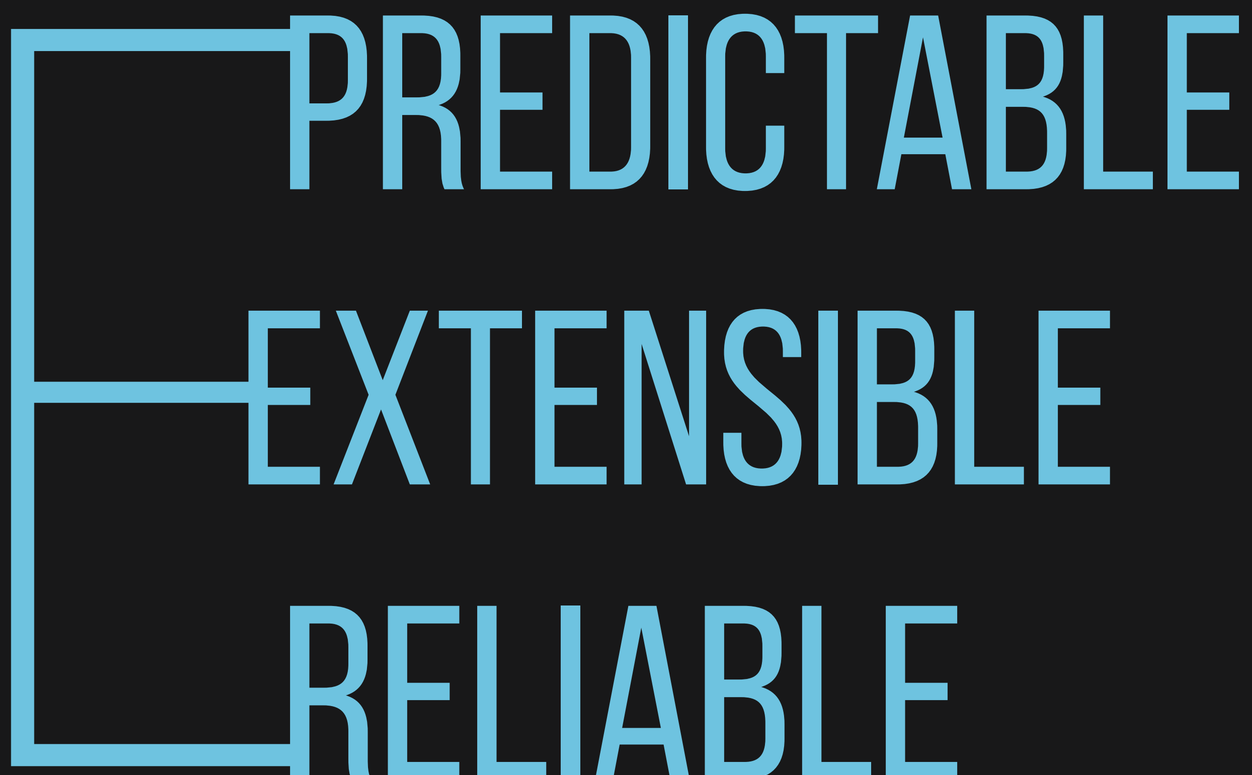
HOW TO BUILD
COMMUNICATION?

WE HAVE SLA

COMMUNICATION

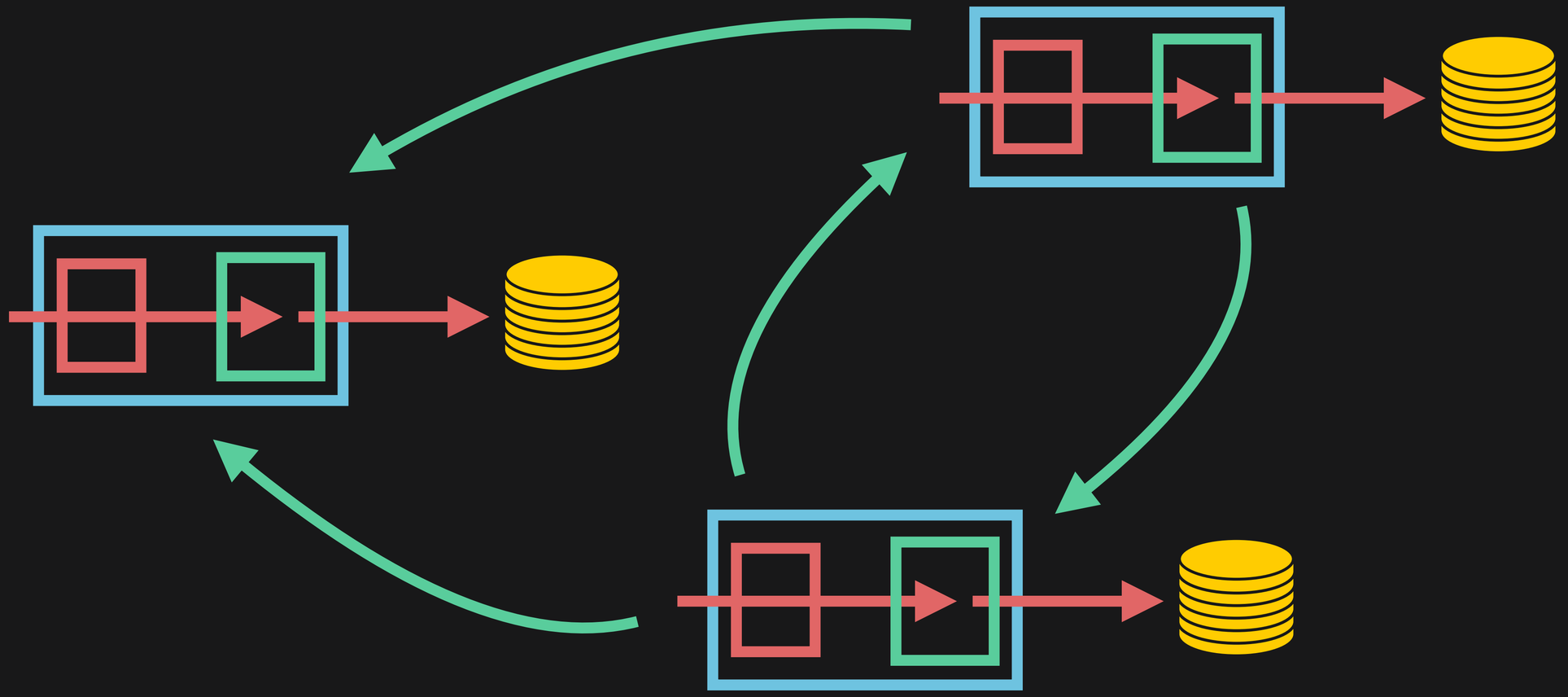
HIGH LEVEL

LOW LEVEL

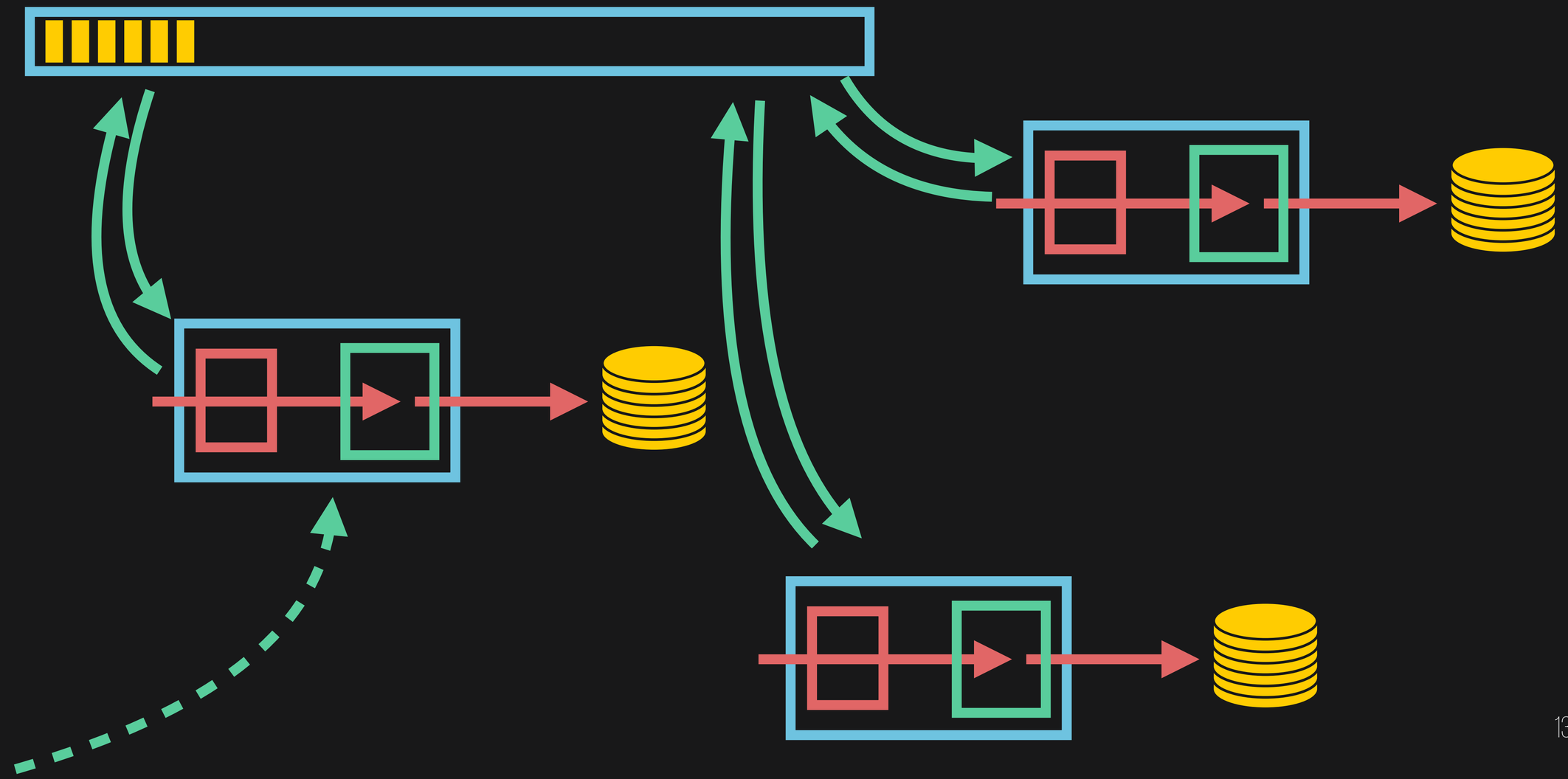

PREDICTABLE
EXTENSIBLE
RELIABLE

ARCHITECTURE

DIRECT COMMUNICATION?



EVENT SOURCING



IMPLEMENTATION

REACTOR

REACTOR WAY

```
service.getUser(id)
   .filter(permissionsService::allowed)
   .flatMap(user → service
                     .fetchDocumentRequests(user.getId())
                     .zipWith(logService::reportAction)
   )
   .doOnError(logService::reportFail)
   .onErrorReturn(
      RateLimitException.class,
      status(TOO_MANY_REQUESTS).build()
   )
   .onErrorReturn(status(INTERNAL_SERVER_ERROR).build());
```

HOW TO TEST?

HOW TO DEBUG?

YOU WON'T WANT TO USE REACTOR

YOU WON'T WANT TO USE REACTOR UNTIL...

PLAN

WHAT WE HAVE

HOW TO FOLLOW

USE REACTOR SAFE

REACTIVE MANIFESTO

WWW.REACTIVEMANIFESTO.ORG

RESPONSIVE

RESILIENT

ELASTIC

MESSAGE DRIVEN

REACTIVE MANIFESTO

NON-BLOCKING
WITH BACK-PRESSURE

RESPONSIVE
RESILIENT
ELASTIC
MESSAGE DRIVEN

REACTIVE MANIFESTO

TO CONSUME RESOURCES

ONLY WHEN ACTIVE

NON-BLOCKING

WITH BACK-PRESSURE

RESPONSIVE

RESILIENT

ELASTIC

MESSAGE DRIVEN

REACTIVE STREAMS

PUBLISHER

| SUBSCRIPTION

SUBSCRIBER

REACTIVE STREAMS

REACTOR

REACTIVE
STREAMS
IMPLEMENTATION **V8**

REACTOR

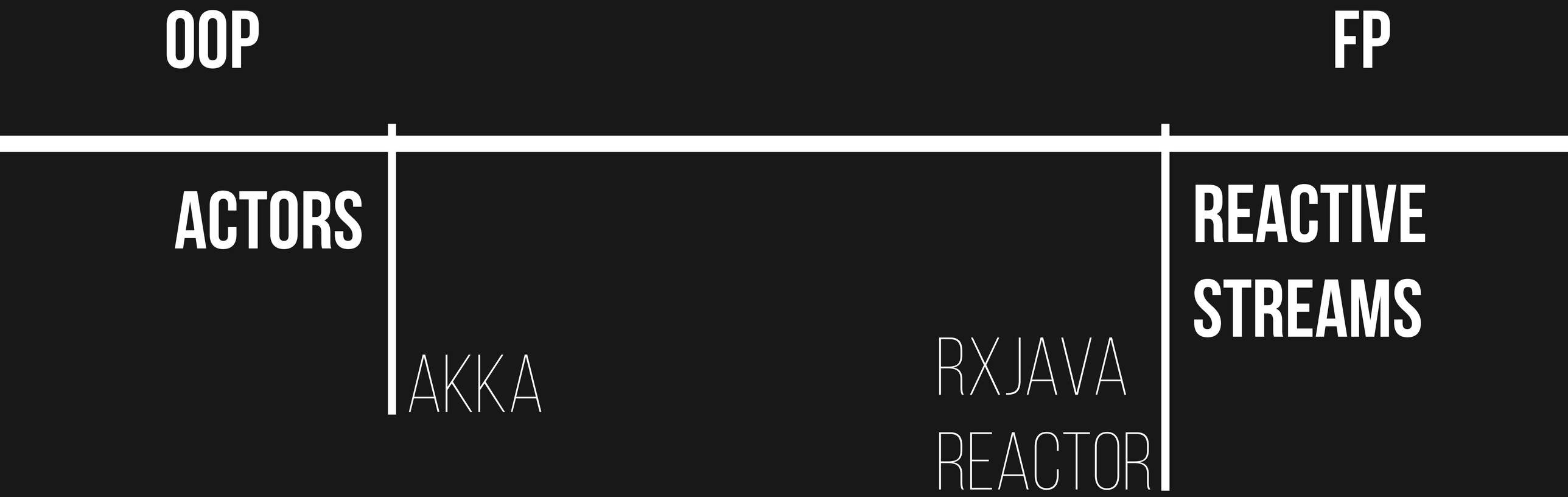
FLUX: 0..N

MONO: 0..1

~ JDK8

STREAM: 0..N

OPTIONAL: 0..1


OOP
FP
ACTORS
AKKA
RXJAVA
REACTOR
REACTIVE
STREAMS

WE HAVE

```
service.getUser(id)
   .filter(permissionsService::allowed)
   .flatMap(user → service
                     .fetchDocumentRequests(user.getId())
                     .zipWith(logService::reportAction)
   )
   .doOnError(logService::reportFail)
   .onErrorReturn(
      RateLimitException.class,
      status(TOO_MANY_REQUESTS).build()
   )
   .onErrorReturn(status(INTERNAL_SERVER_ERROR).build());
```

WE HAVE

FLOW

```
service.getUser(id)
   .filter(permissionsService::allowed)
   .flatMap(user → service
                     .fetchDocumentRequests(user.getId())
                     .zipWith(logService::reportAction)
   )
   .doOnError(logService::reportFail)
   .onErrorReturn(
      RateLimitException.class,
      status(TOO_MANY_REQUESTS).build()
   )
   .onErrorReturn(status(INTERNAL_SERVER_ERROR).build());
```

WE HAVE

MERGE

```
service.getUser(id)
   .filter(permissionsService::allowed)
   .flatMap(user → service
                     .fetchDocumentRequests(user.getId())
                     .zipWith(logService::reportAction)
   )
   .doOnError(logService::reportFail)
   .onErrorReturn(
      RateLimitException.class,
      status(TOO_MANY_REQUESTS).build()
   )
   .onErrorReturn(status(INTERNAL_SERVER_ERROR).build());
```

WE HAVE

ERROR HANDLING

```
service.getUser(id)
   .filter(permissionsService::allowed)
   .flatMap(user → service
                       .fetchDocumentRequests(user.getId())
                       .zipWith(logService::reportAction)
   )
   .doOnError(logService::reportFail)
   .onErrorReturn(
      RateLimitException.class,
      status(TOO_MANY_REQUESTS).build()
   )
   .onErrorReturn(status(INTERNAL_SERVER_ERROR).build());
```

CONCURRENT

HAVE BACK-PRESSURE

DECLARATIVE

HAVE TO GET USED

HOW TO DEBUG?

CONCURRENT

HAVE BACK-PRESSURE

HOW TO TEST?

DECLARATIVE

WHERE LAYERS?

HAVE TO GET USED

HOW TO FOLLOW

LAYERS

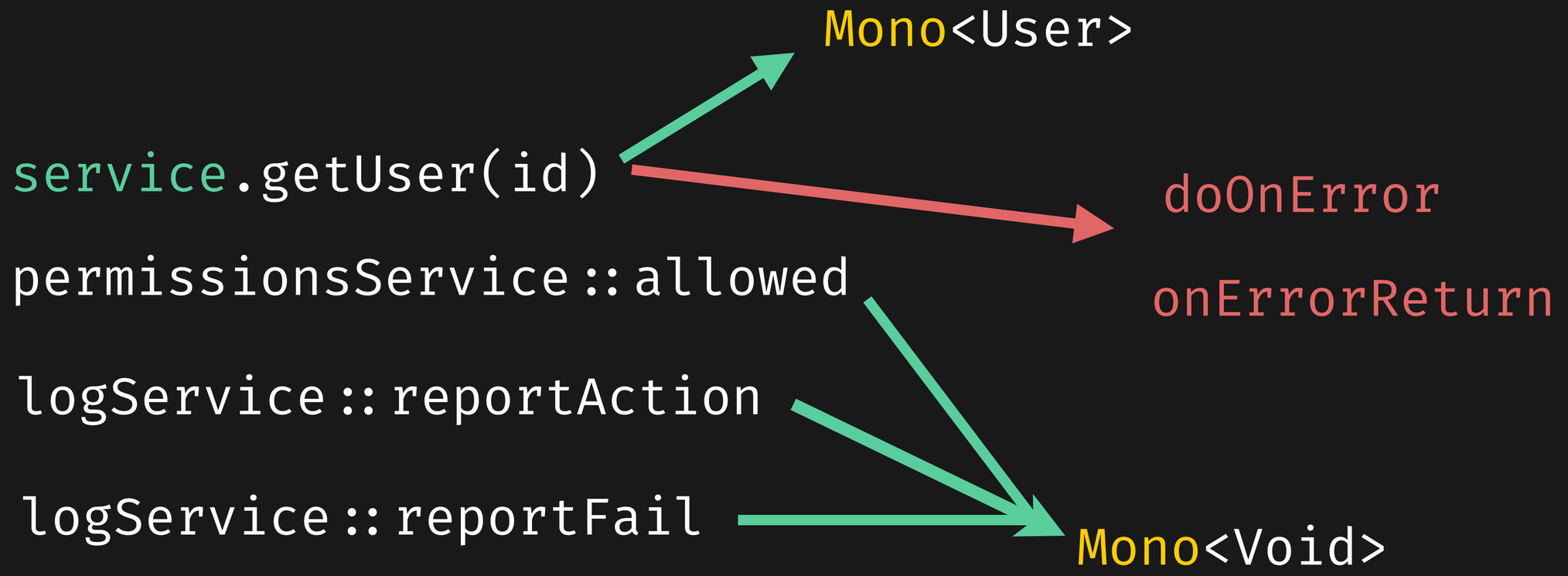
```
service.getUser(id)

permissionsService :: allowed

logService :: reportAction

logService :: reportFail
```

LAYERS


Mono<User>
service.getUser(id)
doOnError
permissionsService::allowed
onErrorReturn
logService::reportAction
logService::reportFail
Mono<Void>

```
service.getUser(id)
   .filter(permissionsService::allowed)
   .flatMap(user → service
                     .fetchDocumentRequests(user.getId())
                     .zipWith(logService::reportAction)
   )
   .doOnError(logService::reportFail)
   .onErrorReturn(
      RateLimitException.class,
      status(TOO_MANY_REQUESTS).build()
   )
   .onErrorReturn(status(INTERNAL_SERVER_ERROR).build());
```

~~onErrorReturn~~

UNHANDLED EXCEPTION

```
at reactor.util.Loggers$Slf4JLogger.error(Loggers.java:295)
    at reactor.core.publisher.SignalLogger.lambda$onErrorCall$5(SignalLogger.java:296)
    at reactor.core.publisher.FluxPeekFuseable$PeekFuseableSubscriber.onError(FluxPeekFuseable.java:219)
    at reactor.core.publisher.FluxMapFuseable$MapFuseableSubscriber.onError(FluxMapFuseable.java:134)
    at reactor.core.publisher.MonoCollectList$MonoBufferAllSubscriber.onError(MonoCollectList.java:107)
    at reactor.core.publisher.FluxMergeSequential$MergeSequentialMain.drain(FluxMergeSequential.java:341)
    at reactor.core.publisher.FluxMergeSequential$MergeSequentialMain.onError(FluxMergeSequential.java:242)
    at reactor.core.publisher.FluxFlattenIterable$FlattenIterableSubscriber.drainAsync(FluxFlattenIterable.java:305)
```

CHECKPOINTS

```
.checkpoint("allow-service")
```

```
at reactor.util.Loggers$Slf4JLogger.error(Loggers.java:295)
    at reactor.core.publisher.SignalLogger.lambda$onErrorCall$5(SignalLogger.java:296)
    at reactor.core.publisher.FluxPeekFuseable$PeekFuseableSubscriber.onError(FluxPeekFuseable.java:219)
    at reactor.core.publisher.FluxMapFuseable$MapFuseableSubscriber.onError(FluxMapFuseable.java:134)
    at reactor.core.publisher.MonoCollectList$MonoBufferAllSubscriber.onError(MonoCollectList.java:107)
    at reactor.core.publisher.FluxMergeSequential$MergeSequentialMain.drain(FluxMergeSequential.java:341)
    at reactor.core.publisher.FluxMergeSequential$MergeSequentialMain.onError(FluxMergeSequential.java:242)
    at reactor.core.publisher.FluxFlattenIterable$FlattenIterableSubscriber.drainAsync(FluxFlattenIterable.java:305)
```

```
Assembly site of producer
  [reactor.core.publisher.MonoIgnoreThen] is
  identified by light checkpoint [allow-
  service]."description" : "allow-service"
```

DEBUG - LOGS

```
.log("flow")
```

```
2018-11-21 17:59:23.285  INFO   --- [    Test worker] flow
  : | request(unbounded)
2018-11-21 17:59:23.275  INFO   --- [    Test worker] flow
  : | onSubscribe([Fuseable] FluxOnAssembly.OnAssemblySubscriber)
2018-11-21 17:59:23.291 ERROR   --- [    Test worker] flow
  : | onError(java.lang.IllegalStateException)
```

USE REACTOR SAFE

HOOKS + TESTS

WARN: PERFORMANCE IMPACT

```
Hooks.onOperatorDebug();
```

```
Error has been observed by the following operator(s):
  |_  Mono.create(PublicEmergencyDataRepository.java:101)
  |_  MonoCreate$DefaultMonoSink.error(DefaultAsyncHandler.java:14)
  |_  Mono.onErrorMap(PublicEmergencyDataRepository.java:102)
  |_  Mono.then(PublicEmergencyDataRepository.java:103)
  |_  MonoCreate$DefaultMonoSink.error(DefaultAsyncHandler.java:14)
  |_  Mono.onErrorResume(PublicEmergencyDataService.java:70)
  |_  Mono.then(PublicEmergencyDataService.java:71)
  |_  Mono.then(UserEventLogConsumer.java:63)
  |_  Mono.defer(UserEventLogConsumer.java:29)
```

UNITTEST

```
void expectTwoUsers(Flux<User> flux) {
    StepVerifier.create(flux)
        .expectNextMatches(
          user → user.getUsername().equals("swhite")
        )
        .expectNextMatches(
          user → user.getUsername().equals("jpinkman")
        )
        .expectComplete();
}
```

COLD/HOT

Mono.defer	Mono.just
Mono.fromCallable	Mono.error
Mono.fromRunnable	
Flux.~	

COLD/HOT

DELAY

HEAVY

```
.switchIfEmpty(Mono.defer(() → service.random(id))
                .delayUntil(__ → accessLogService.unsuccess(id, headers))
                .map(data → ResponseEntity.ok()
                                .body(new ByteArrayResource(data))
                )
        )
```

COLD/HOT

IMMEDIATELY EVALUATED

```
Mono.just(Files.readString(path))
```

RETRY

```
.retryWhen(it ->
  it.delayElements(Duration.ofMillis(100)).take(2)
)
.defaultIfEmpty(Collections.emptyList())
```

RETRY

```
.retryWhen(Retry
        .anyOf(ServerException.class)
        .retryMax(5)
        .exponentialBackoff(Duration.ofMillis(100), Duration.ofMillis(500))
)
.onErrorMap(RetryExhaustedException.class, Throwable::getCause)
```

REACTOR-EXTRA

SCHEDULERS

```
Mono.fromCallable(() → heavyIO(next))
        .subscribeOn(Schedulers.parallel())
        .log("heavy-io")
```

SCHEDULERS

```
Mono.fromCallable(() → heavyIO(next))
        .subscribeOn(Schedulers.parallel())
        .log("heavy-io")
```

BLOCKS

FLUX.INTERVAL
.DELAY…
.TIMEOUT…

DRAWBACKS

BENEFITS ONLY IF FULLY NON BLOCKING ALREADY REACTIVE

FULLY NON BLOCKING
ALREADY REACTIVE

OR

```
Mono.fromCallable
Mono.fromRunnable
.subscribeOn(
   Schedulers.elastic
)
Mono::block

Flux…
```

FULLY NON BLOCKING
ALREADY REACTIVE

OR

```
Mono.fromCallable
Mono.fromRunnable
.subscribeOn(
   Schedulers.elastic
)
Mono::block

Flux…
```

OR DEAL WITH LEAKS,
RACE CONDITIONS,
DEADLOCKS

TAKEAWAYS

TOP 3

CHECKPOINTS, LOGS

TAKEAWAYS

TOP 3

CHECKPOINTS, LOGS

UNIT TESTS, HOOKS

TAKEAWAYS

TOP 3

CHECKPOINTS, LOGS

UNIT TESTS, HOOKS

RETRIES

I AM OPEN TO TALK!

GITHUB.COM/LANWEN

T.ME/LANWEN

KIRILL MERKUSHEV
@DELNARIEL