mediarithmics

Cats Retry 101

jobs@mediarithmics.com

mediarithmics

Who am I?

Nicolas François
Software Engineer - SRE
nfrancois@mediarithmics.com
@koisell

mediarithmics

Motivations for this talk

mediarithmics

What's retrying?

Retrying is the ability for a system to re-execute an action when the previous execution was unsuccessful.

Policy (max time + max retry) driven

mediarithmics

Cats retry policies

```
case class RetryPolicy[M[_]]
 (decideNextRetry: RetryStatus => M[PolicyDecision])
```

- constantDelay (retry forever, with a fixed delay between retries)
- limitRetries (retry up to N times, with no delay between retries)
- exponentialBackoff (double the delay after each retry)
- fibonacciBackoff (delay(n) = (delay(n - 2) + delay(n - 1))
- fullJitter (randomised exponential backoff)

mediarithmics

Policy combinator: Join

- If either of the policies wants to give up, the combined policy gives up.
- If both policies want to delay and retry, the *longer* of the two delays is chosen.

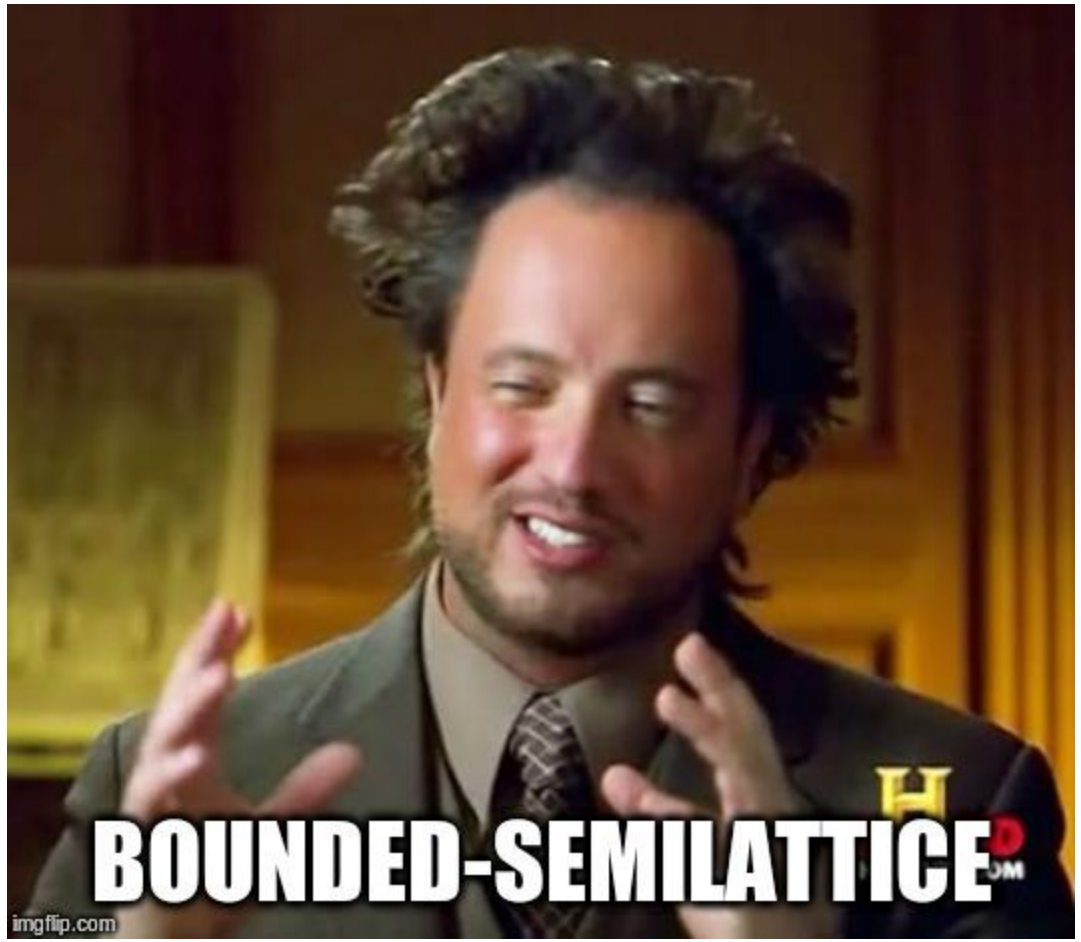
This way of combining policies implies:

- That combining two identical policies result in this same policy.
- That the order you combine policies doesn't affect the resulted policy.
- There is an identity policy: retries with no delay and never gives up

```
val policy = limitRetries[Id](5) join exponentialBackoff[Id](10.milliseconds)
```

mediarithmics

Policy combinator: Join



mediarithmics

Policy combinator: Meet

It is the dual of join and has the following semantics:

- If *both* policies want to give up, the combined policy gives up.
- If both policies want to delay and retry, the *shorter* of the two delays is chosen.

Just like join, meet is also associative, commutative and idempotent, which implies:

- That combining two identical policies result in this same policy.
- That the order you combine policies doesn't affect the resulted policy.

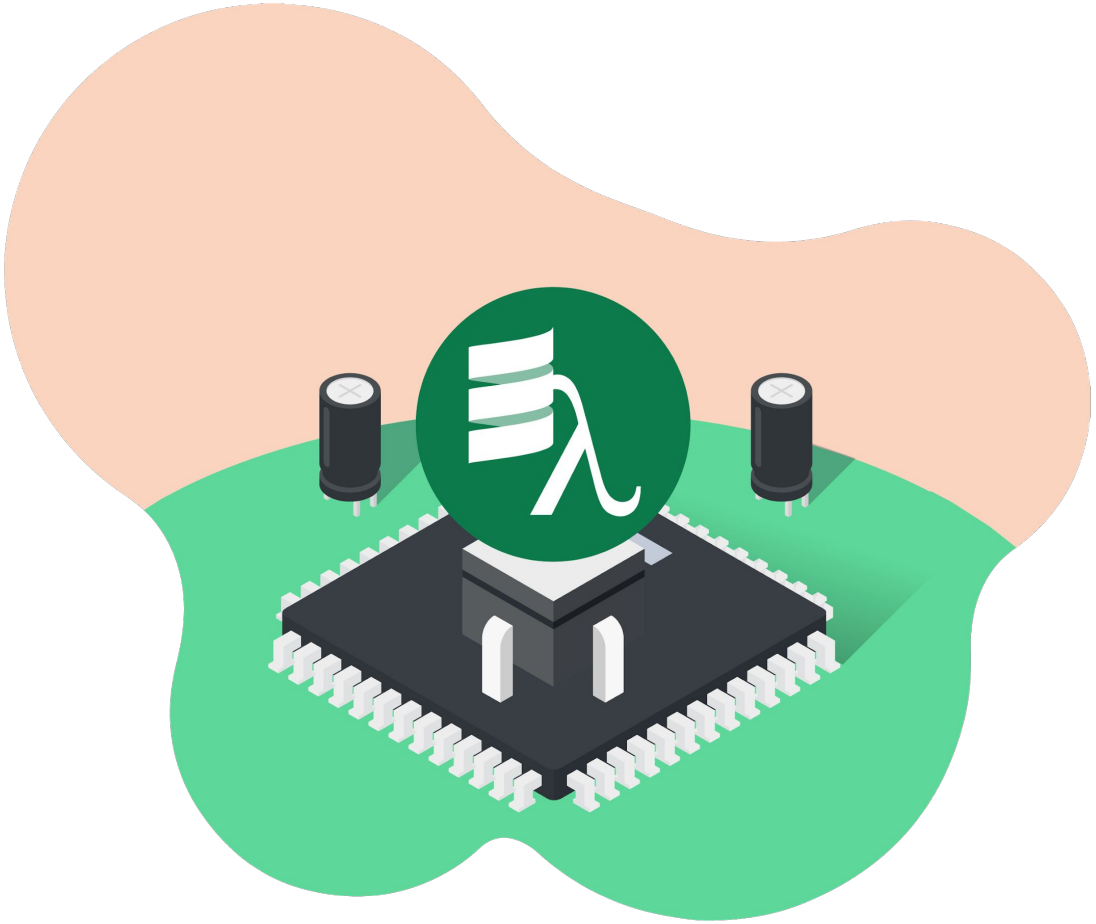
```
val policy = limitRetries[Id](5) meet exponentialBackoff[Id](10.milliseconds)
```

mediarithmics

Sleep typeclass

```
trait Sleep[M[_]] {
  def sleep(delay: FiniteDuration): M[Unit]
}
```

Retrying



mediarithmics

Retrying

```
def retrying[A](policy: RetryPolicy[Id],
                wasSuccessful: A => Boolean,
                onFailure: (A, RetryDetails) => Unit)
                (action: => A): A



def retryingM[A](policy: RetryPolicy[M],
                wasSuccessful: A => Boolean,
                onFailure: (A, RetryDetails) => M[Unit])
                (action: => M[A]): M[A]

// sleep[M] is implicit here
```

mediarithmics

retryingOnErrors

```
def retryingOnSomeErrors[A, E, M: Monad](policy: RetryPolicy[Id],
                                         isWorthRetrying: A => Boolean,
                                     onFailure: (E, RetryDetails) => M[Unit])
                                         (action: => M[A]): M[A]


def retryingOnAllErrors[A, E, M: Monad](policy: RetryPolicy[Id],
                                     onFailure: (E, RetryDetails) => M[Unit])
                                         (action: => M[A]): M[A]


// sleep[M] and MonadError[M, A, E] are implicit here
```

mediarithmics

Syntactic sugar

```
import retry._
import cats.effect.IO


def noop[IO[_], A]: (A, RetryDetails) => IO[Unit] = retry.noop[IO, A]
val policy: RetryPolicy[IO] = RetryPolicies.limitRetries[IO](2)

val httpClient = util.FlakyHttpClient()

val flakyRequest: IO[String] = IO(httpClient.getCatGif())

 retryingOnAllErrors(policy, noop)(flakyRequest)
```

mediarithmics

Syntactic sugar

```
import retry._
import cats.effect.IO
import retry.syntax.all._

implicit def noop[IO[_], A]: (A, RetryDetails) => IO[Unit] = retry.noop[IO, A]
implicit val policy: RetryPolicy[IO] = RetryPolicies.limitRetries[IO](2)

val httpClient = util.FlakyHttpClient()

val flakyRequest: IO[String] = IO(httpClient.getCatGif())

 flakyRequest.retryingOnAllErrors
```

mediarithmics

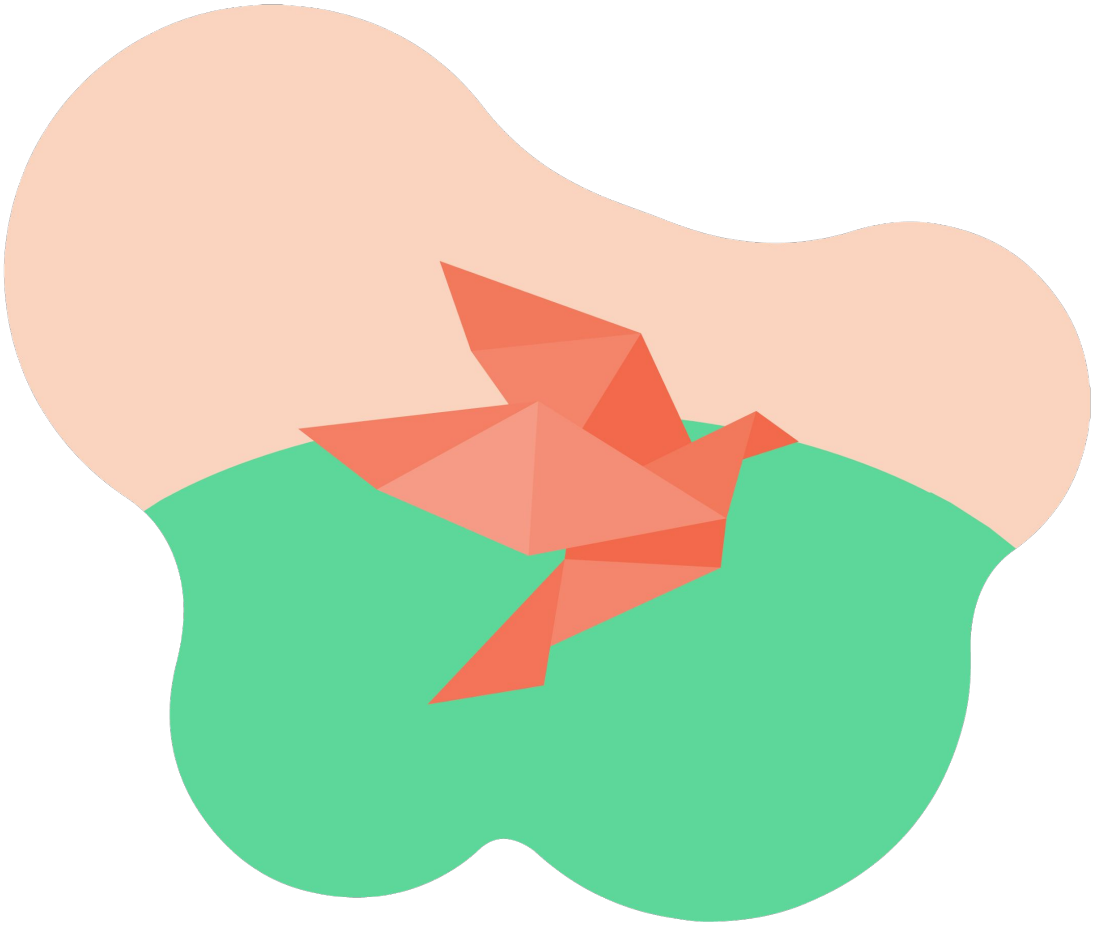
Composing

```
val retry1 = flakyRequest1.retryingOnAllErrors
val retry2 = flakyRequest2.retryingOnAllErrors

val globalRetry = (retry1.flatMap(retry2)).retryingOnAllErrors


// It’s that simple
```

When to retry:
A bird’s eye view

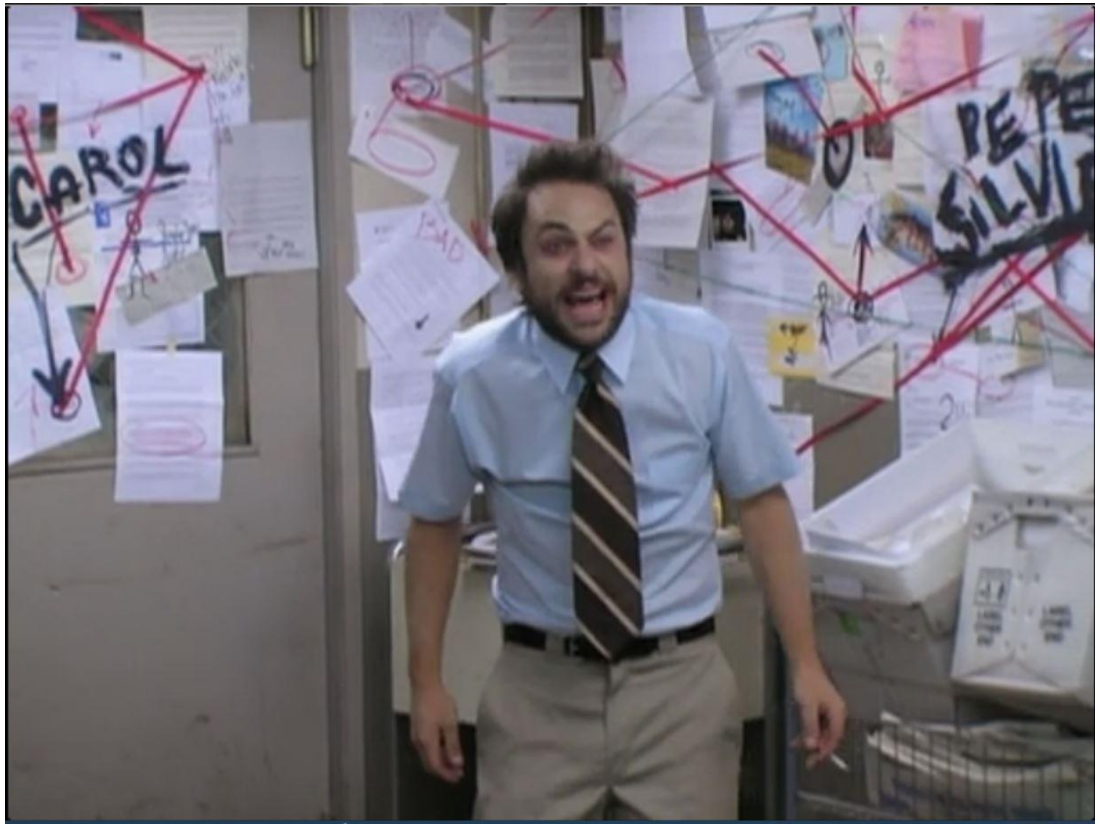


mediarithmics

Service mesh

A service mesh is a dedicated infrastructure layer for making service-to-service communication safe, fast, and reliable.

It tends to control high level circuit breaker, message encryption, service discovery, retrying, ...



mediarithmics

Thank you !

https://github.com/MEDIARITHMICS/talks

jobs@mediarithmics.com