The Role of Reactive and Event-Driven in Microservices

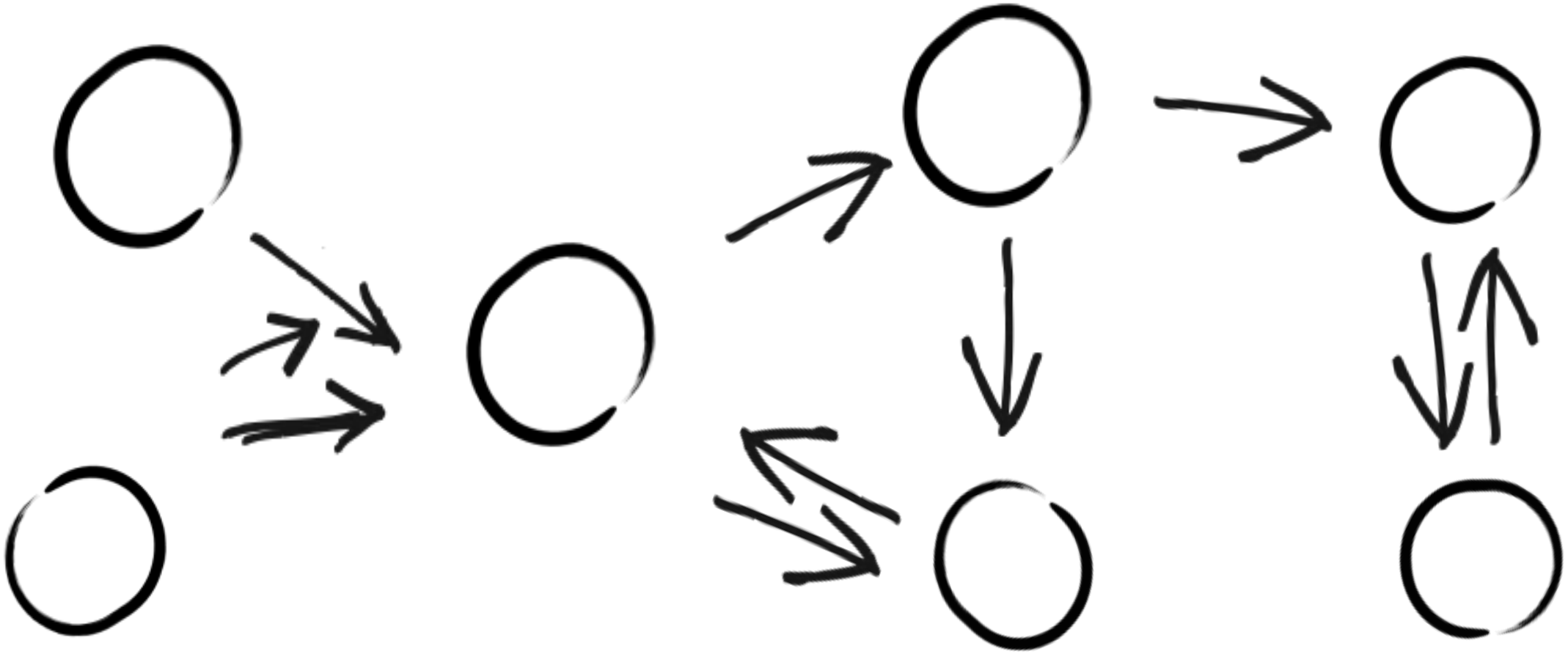
Dr. Clement Escoffier
Senior Principal Software Engineer

Dr. Julien Ponge
Principal Software Engineer

Red Hat

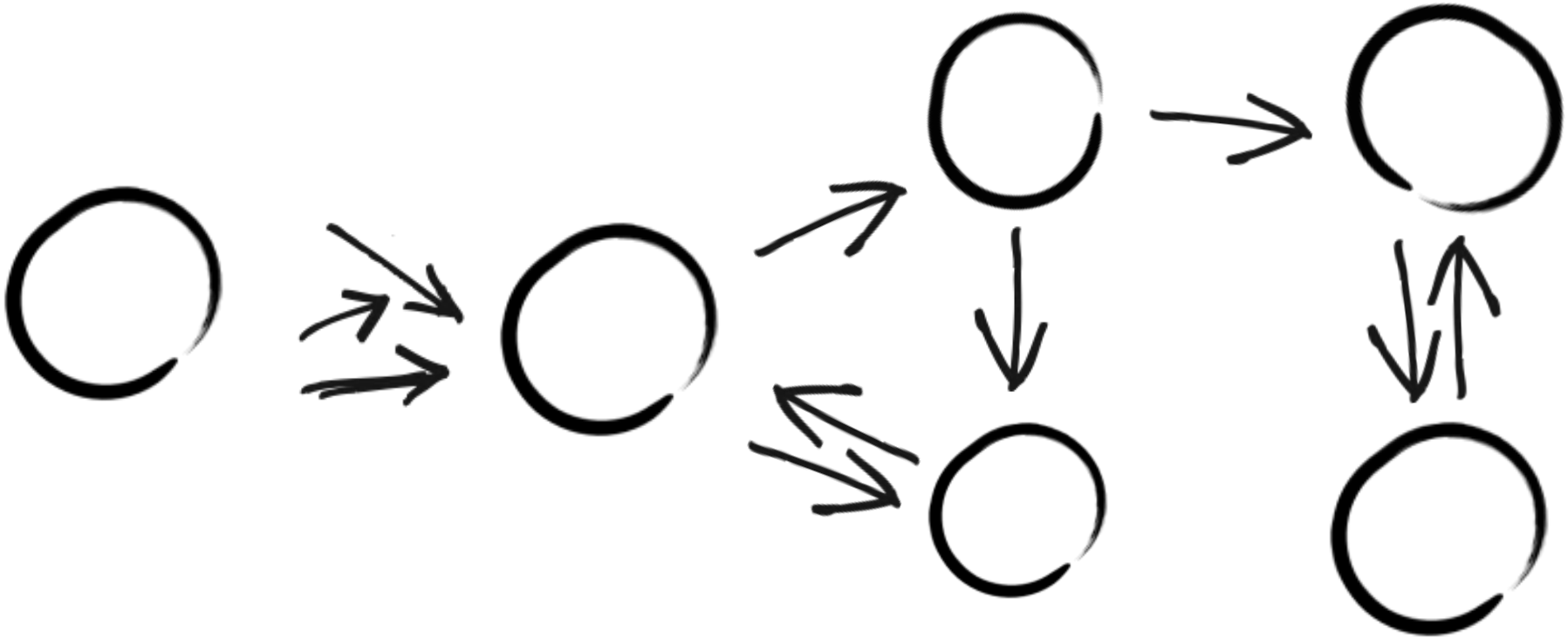
Microservices are all about distributed systems...

It’s all about distributed systems

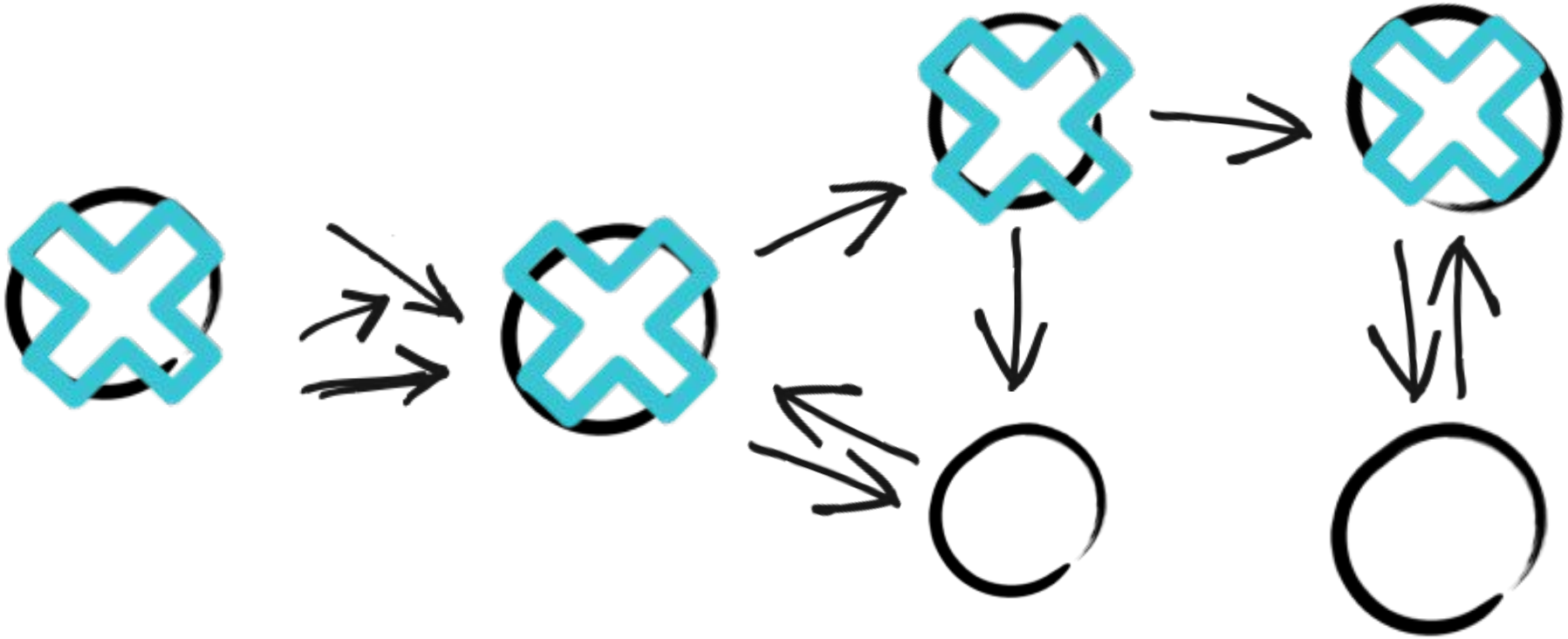


Source:
Insert source data here
Insert source data here
Insert source data here

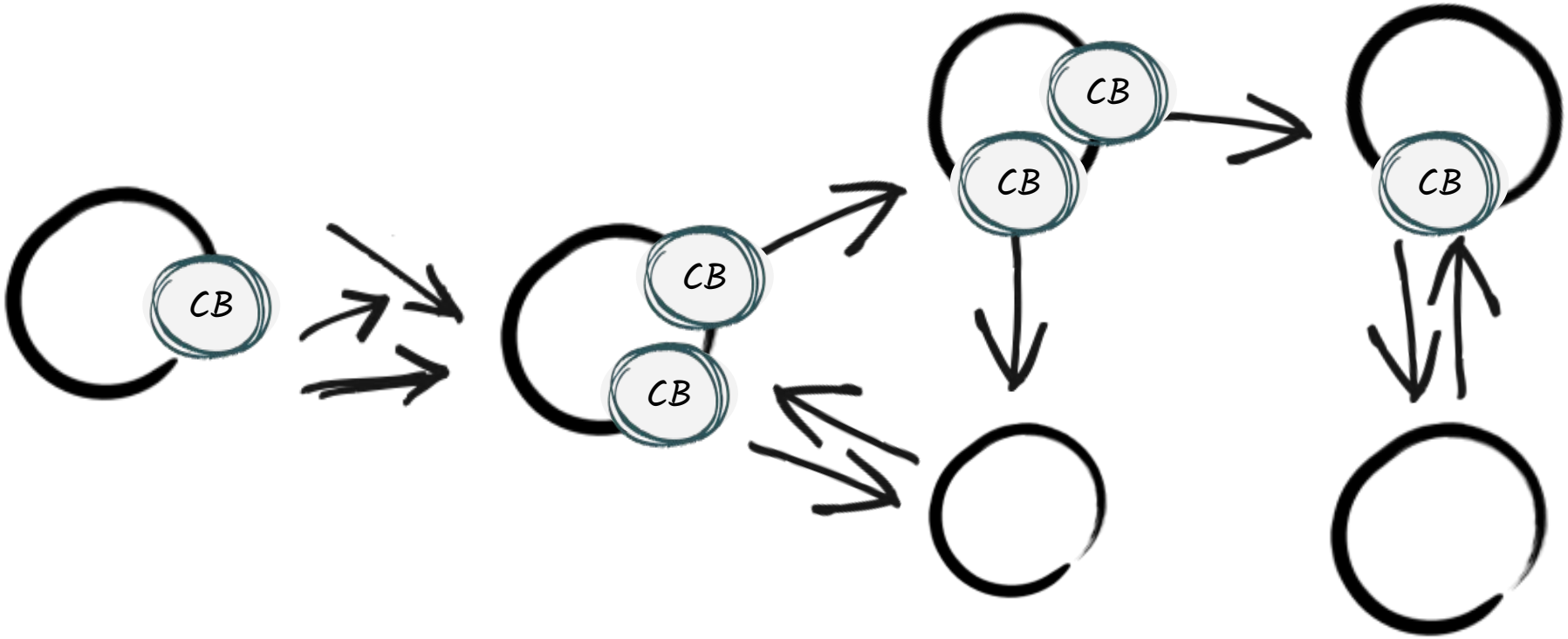
HTTP => STRONG COUPLING



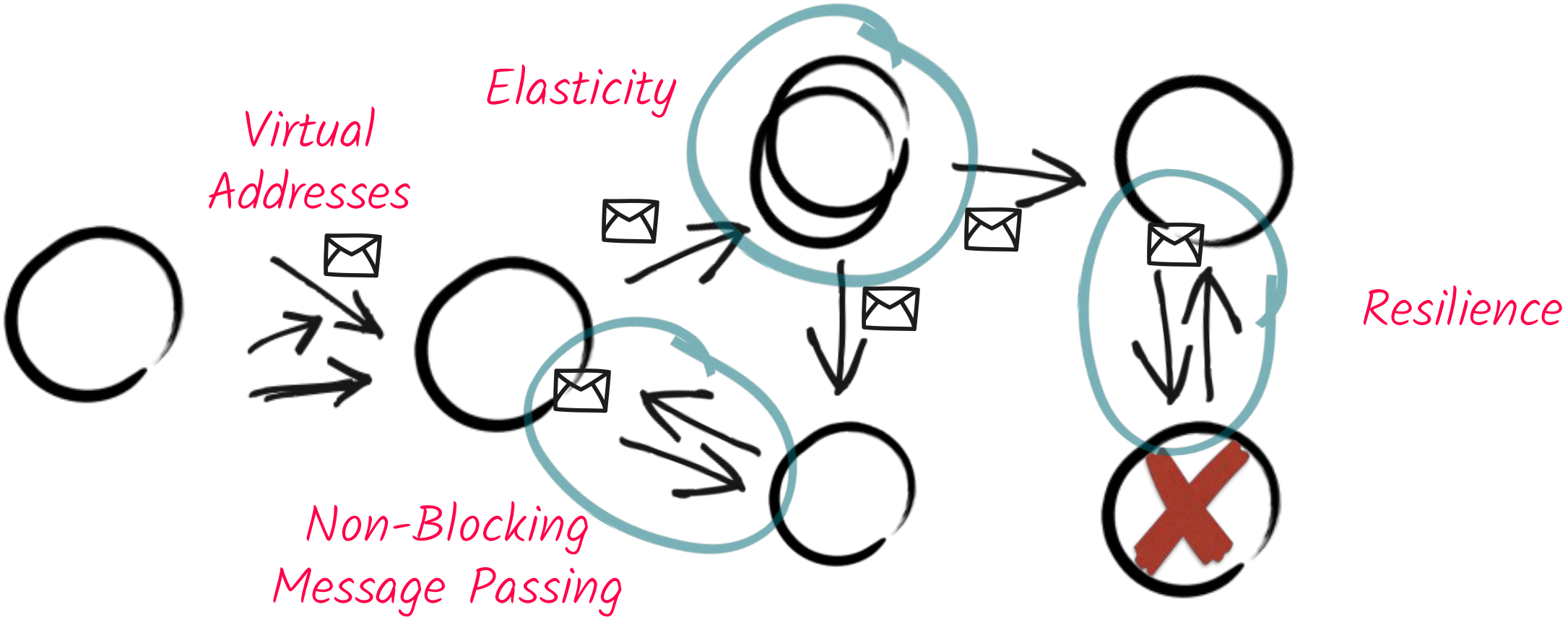
HTTP => UPTIME COUPLING



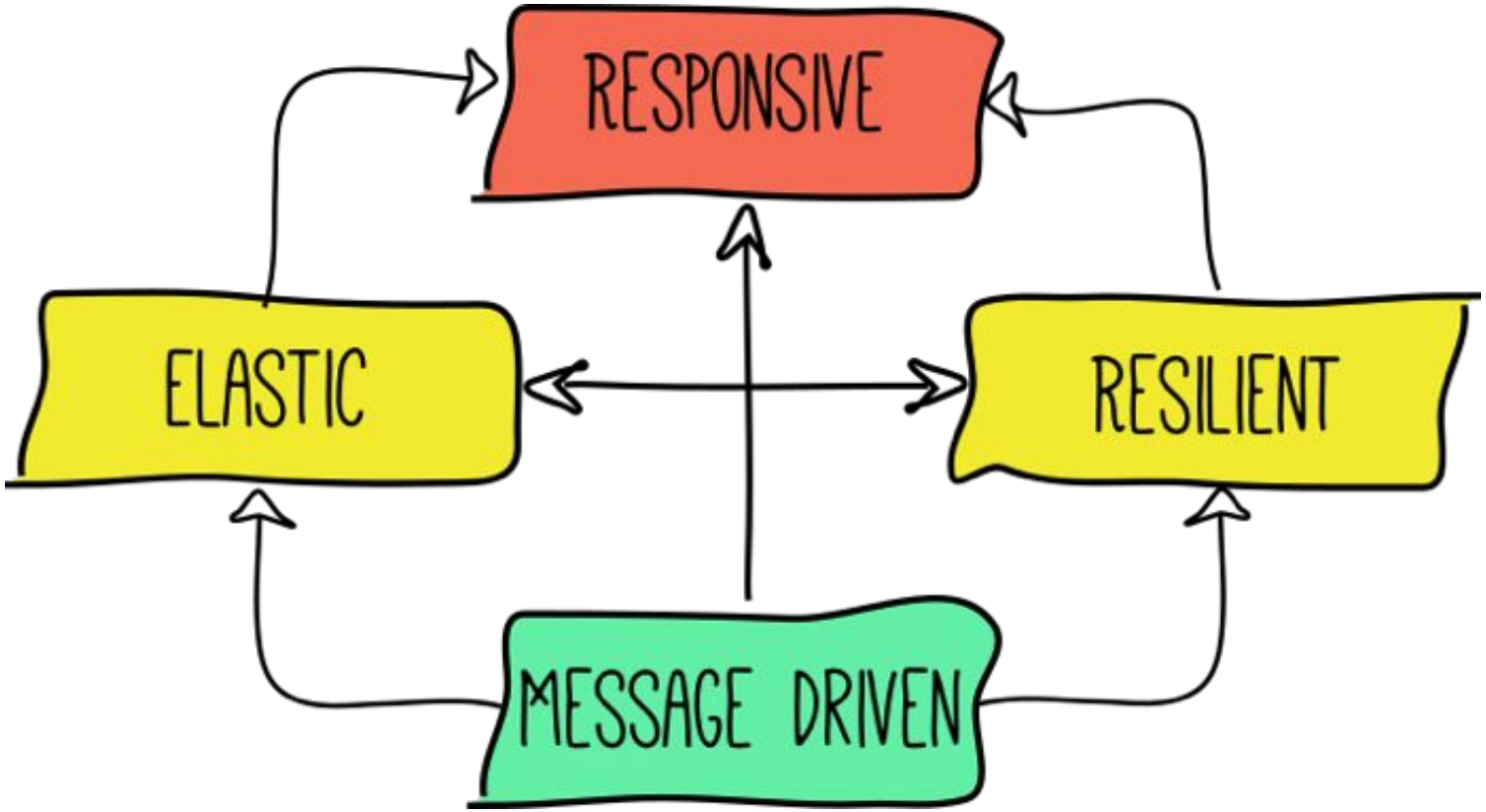
THE RISE OF CIRCUIT BREAKERS



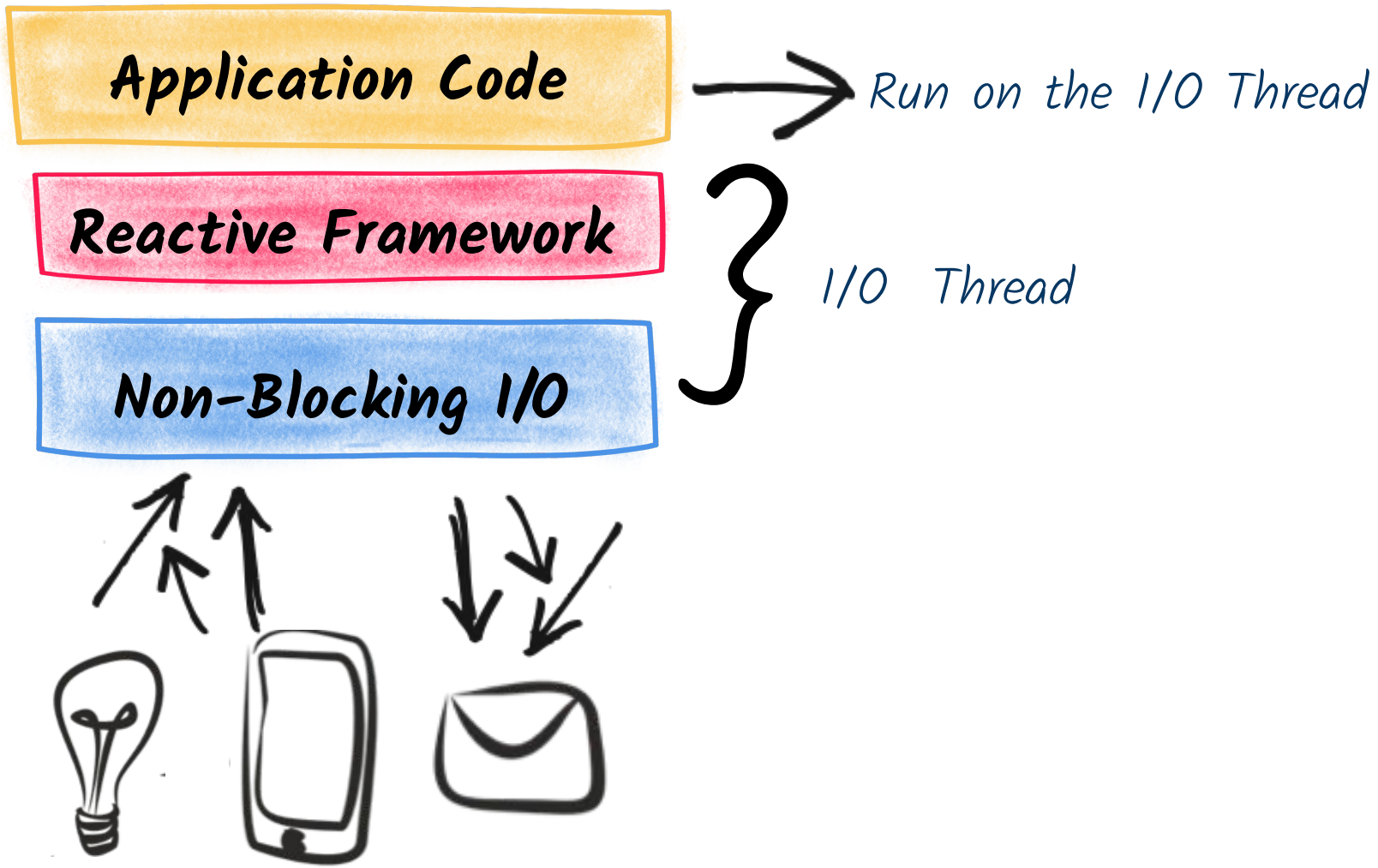
THE BENEFITS OF MESSAGING



REACTIVE => RESPONSIVE



REACTIVE: A DIFFERENT CONCURRENCY MODEL



Writing asynchronous code is ~~challenging~~ HARD!

It’s all about expressing **continuation**

CALLBACKS

- \+ Simple to understand
- \+ Reflect the event-driven nature of the code
- \- Hard to compose
- \- Lead to callback-hell

```
vertx.createHttpServer()
    .requestHandler(req ->  // Async reaction
        req.response().end("Reactive Greetings")
    )
    .listen(8080, ar -> { // Async operation
        // Continuation...
    });
```

FUTURES & PROMISES

+ More composable than callbacks
+ Built-in support in many languages
- Don't reflect the event-driven nature
- Limited to single write (multiple read)

```
CompositeFuture.all(
 fetchTemperature(3000),
 fetchTemperature(3001),
 fetchTemperature(3002))
 .flatMap(this::sendToSnapshot)
 .onSuccess(data -> request.response()
   .putHeader("Content-Type", "application/json")
   .end(data.encode()))
 .onFailure(err -> {
   logger.error("Something went wrong", err);
   request.response().setStatusCode(500).end();
 });
```

REACTIVE PROGRAMMING

+ Use data stream as primary construct
+ Laziness
+ Back Pressure (Reactive Streams)
- Not everything is a stream
- Functional - hard to grasp
- Too many operators - *Nomad* Hell

```
client.rxGetConnection()  // Single(async op)
 .flatMapPublisher(conn ->
   conn
     .rxQueryStream("SELECT * from PRODUCTS")
     .flatMapPublisher(SQLRowStream::toFlowable)
     .doAfterTerminate(conn::close)
 ) // Flowable of Rows
 .map(Product::new) // Flowable of Products
 .subscribe(System.out::println);
```

VIRTUAL THREAD AND COROUTINES

+ Write async code in a synchronous fashion
  - Code rewriting (Quasar, Kotlin, JS)
  - Runtime support (Loom)
+ Easy to reason about
- Requires runtime support
- Not event-driven
- No real stream and Back-Pressure support (see channels, blocking queues, etc)
- Integration with the reactive and non-blocking ecosystem can be challenging

```
Router router = Router.router(vertx);
router.route().handler(rc -> {
    Thread.startVirtualThread(() -> { rc.next(); });
});
router.get("/").handler(rc -> {
  HttpResponse<Buffer> response = client.getAbs(...)
          .sendAndAwait();
  rc.response().end(response.bodyAsString());
});
```

ROADMAP

Objective: integration in Quarkus (https://quarkus.io)

INTUITIVE EVENT-DRIVEN REACTIVE PROGRAMMING

- Based on the idea of reactive programming
  - More event-driven
  - API navigability
  - Not everything is a stream
- SmallRye Mutiny: https://smallrye.io/smallrye-mutiny/

STRUCTURED AND MANAGED CONCURRENCY

- Based on the idea of coroutine
  - Better integration with reactive ecosystem (Eclipse Vert.x)
  - Provide higher-level abstraction to compose actions

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat