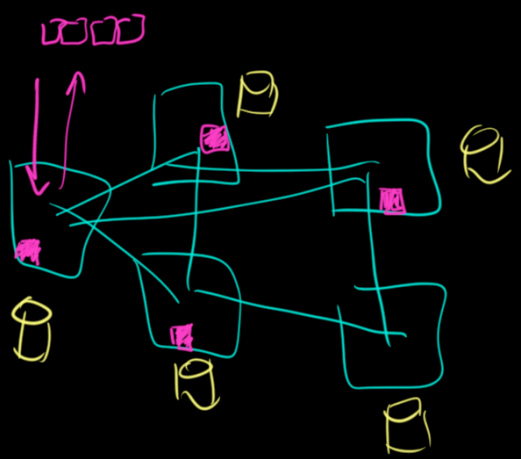
Microservices

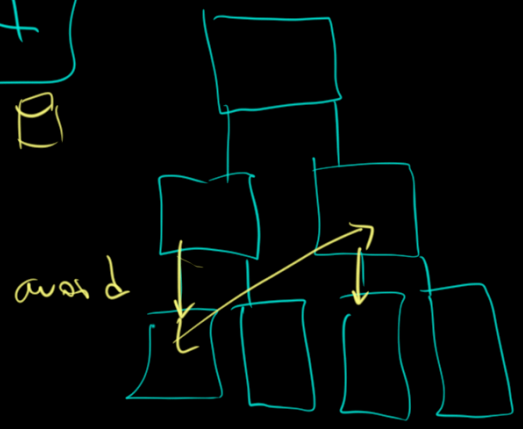
we deploy >1 artifact

Monolith

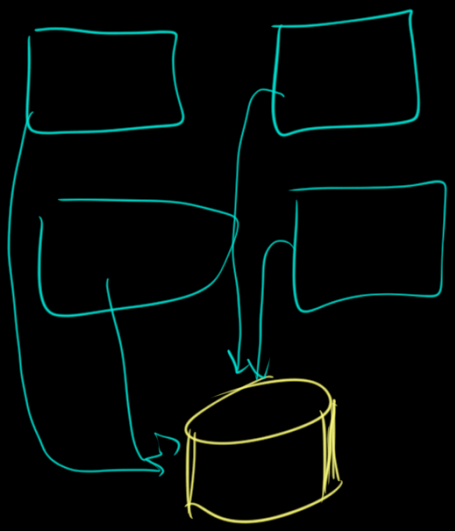
we deploy 1 artifact (JAR)

artifact is
- codebase
- repository
- binary
- file





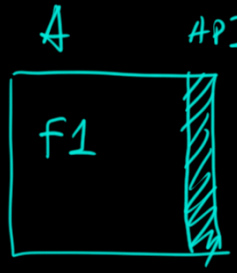
SOA



Each Service
One Function

1990

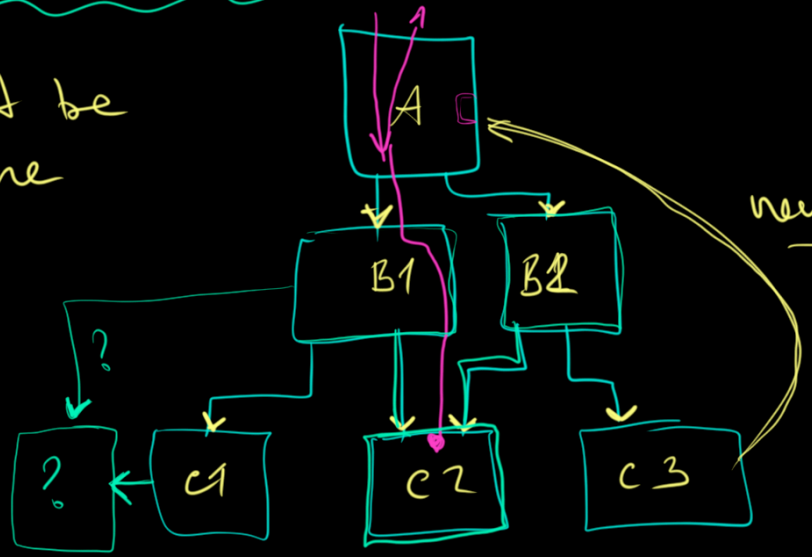
Main Problem



SOA - mostly about
- structure
- responsibility

once it's deployed - you don't know WHO will use it.

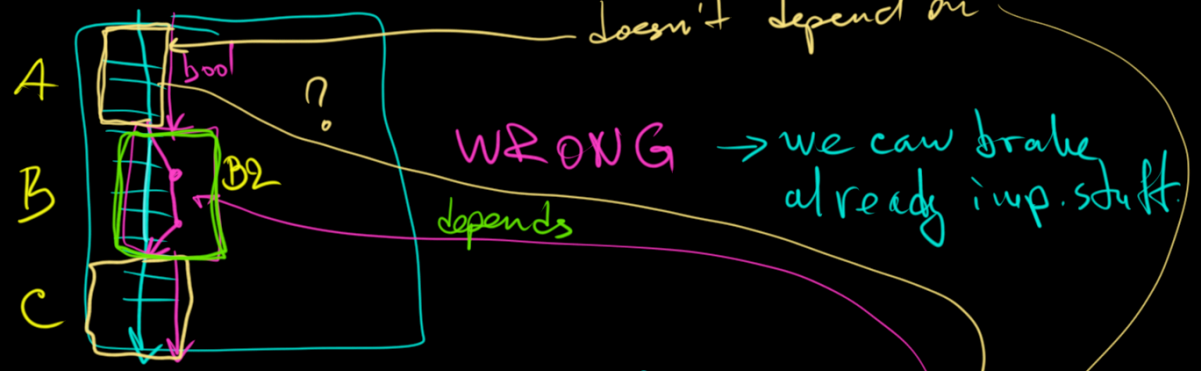
arrows must be only in one direction



never ever do that!

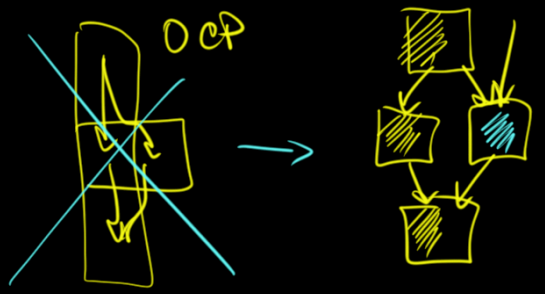
A
B1 B2
C1 C2 C3
A

circular dependency

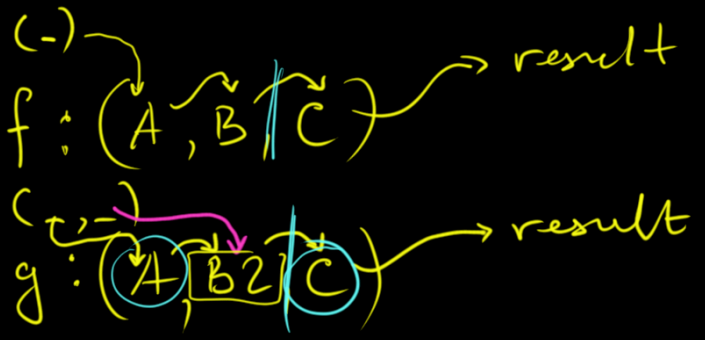


any functionality is $f(A,) \to B$

new functionality $g(A, \text{Bool}) \to B$



we always do decomposition and create new things



- composition
- reusability

A, B, C - is working
we need to test B2 only
contract between B and C
is the same
B2 output will be the same
you don't need to test AB2C

SOLID? OCP

② implement New => compose again

when need do modify - ① factor out common => reuse

f(..... , bool, bool)

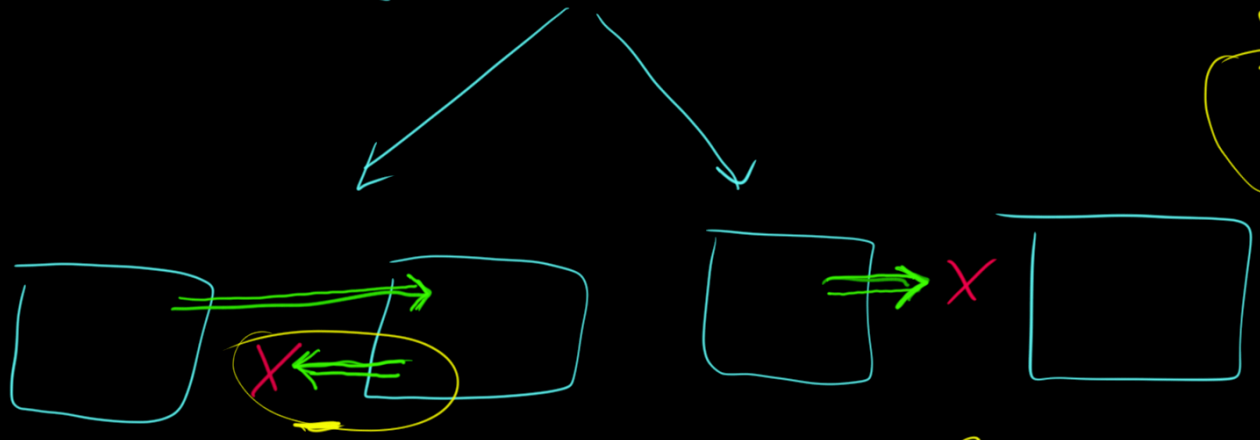
You can easily mix

f(... true, false) ← can compiler help? → NO!

f(... false, true)

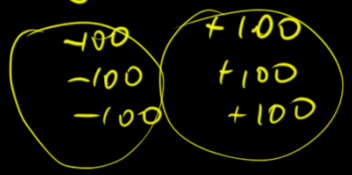
(isSmokingAllowed , isPetAllowed)

we didn't get a response



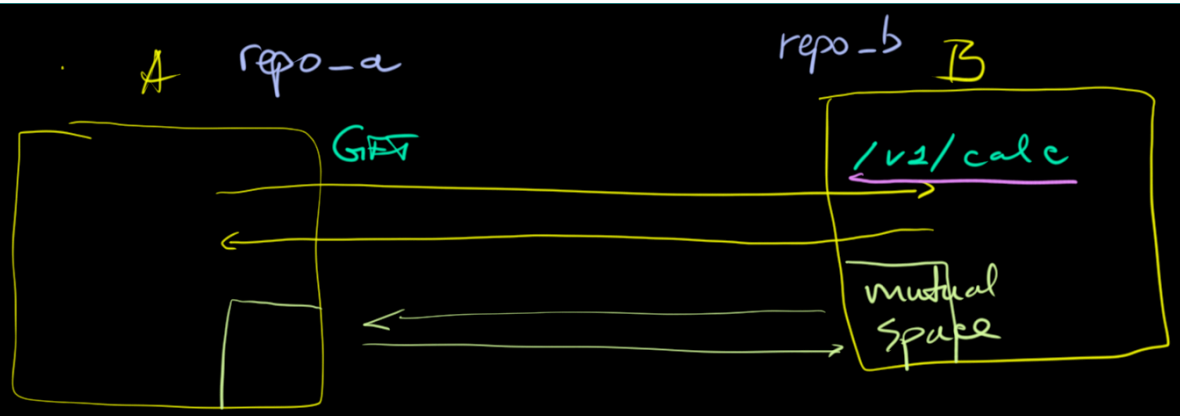
can we distinguish that ? =>

→ If you design your data properly you will



we need to be extremely care with retry strategy

1. we need the microservice structure well documented
   - how to build
   - how to deploy



A ~~depends~~ on B

B ~~depends~~ on A

A → B → A circular dep!

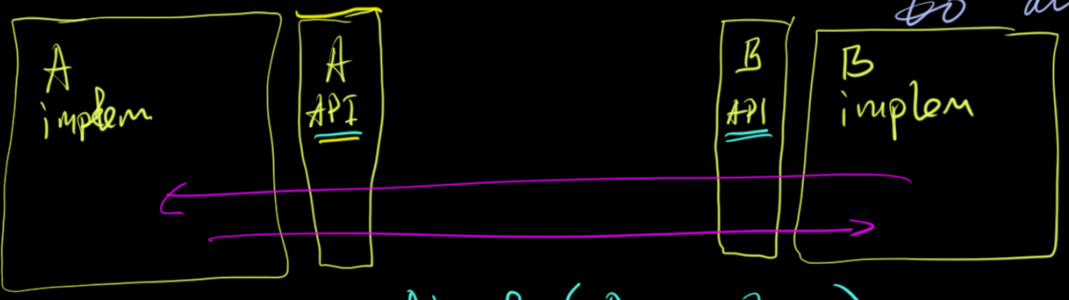
mkReq("/v1/calc")

"/v1/calculate"

"/v2/calc"

nobody can force us (help)

→ we need a mutual place to avoid duplication



A impl (A Api, B Api)

B impl (B Api, A Api)

A API
A Implement
) must be different compilation Units

A impl → A API → B API

③ ② ①

B impl → B API → A API

④ ① ②

Mutual Space

1. endpoint path : 'v1/calc' 'v2/getinfo' 'v2/user/set'
2. request type : GET / POST / DELETE ....
3. parameter names
4. parameter types
5. expectable status codes
6. input / output data formats

→ no code duplication
→ TIMES LESS ERROR POSSIBILITY

```
@PostMapping("b")
String[] path() default
```

```
method = {RequestMethod.POST}
```

```
@RequestParam("x") String x
```

```
@ResponseStatus(HttpStatus.ACCEPTED)
```

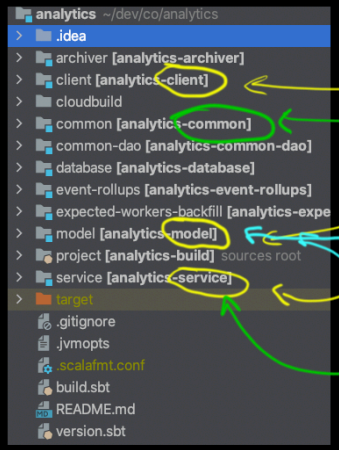
```
@RequestBody Book book
```

```
public class Book {
  private String name;
  private String author;
}
```

```
@PostMapping("person")
public Person handle_pers
```

ANTI PATTERN

in the most projects there is duplication



different compilation units

```
final case class Cpu(time: OffsetDateTime, cpu: Long)

final case class WorkAvailable(numWorkers: Int, secondsRemainingPerWorker: Int, shiftId: ShiftId)
```

```
case request @ GET -> Root / "picking"
      :? DCIdQueryParamMatcher(dcId)
        +& OptionalCursorQueryParamMatcher(maybeCursor)
        +& NumQueryParamMatcher(num)
        +& OptionalWaveTypeIdQueryParameter(maybeWaveTypeId) =>
val result: StuckOrders = stuckOrd
  dcId,
  maybeCursor,
  num,
  InProgressOrderStatePicking,
  Filter.waveTypeIdFilter(maybeWav
)
Ok(result.asJson)
```

```
object ApiParamNames {

  val dcId = "dcId"
  val cursor = "cursor"
  val num = "num"
  val skuId = "skuId"
  val time = "time"
  val timeBucket = "timeBucket"
  val granularity = "granularity"
  val orderType = "orderType"
  val shiftId = "shiftId"
  val waveTypeId = "waveTypeId"
  val moveReasonTypeId = "moveReasonTypeId"

}
```

```
"/blocking-starts" -> new BlockingSt
"/put-wall-recommendations" -> new P
"/pick-assignment-item-hourly-stats"
"/order-progress" -> new OrderProgre
OrderLinesReleased.root -> new OrderLinesReleasedService(config).routesWithCatchAll,
```

```
object OrderLinesReleased {
  val root = "/order-lines-released"
  val `15min` = "15min"
  val `hourly` = "hourly"
}
```

can be reused in server and in the client as well

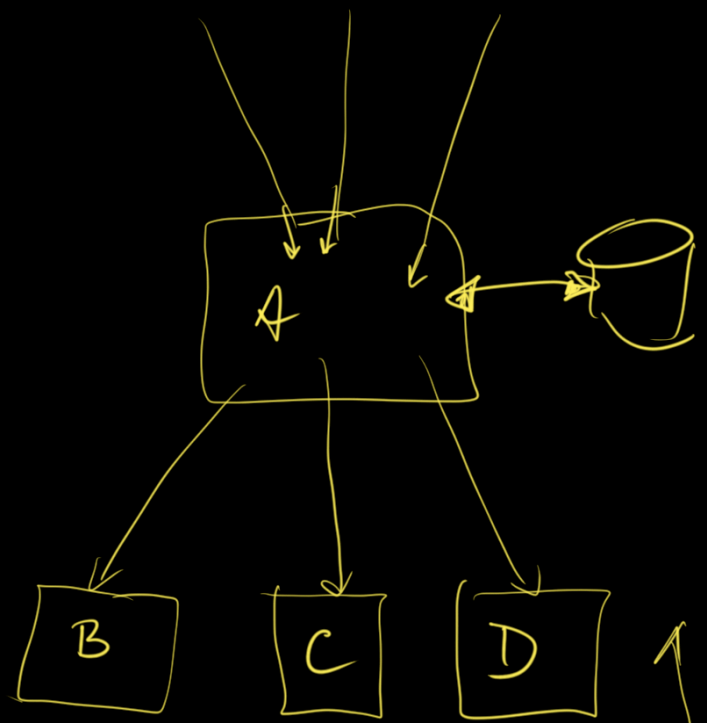
```
Method.GET, {
  import OrderLinesReleased._
  (Uri.unsafeFromString(root) / `15min`)
    .withQueryParam(p.dcId, dcId)
    .withQueryParam(p.time, time)
}
```

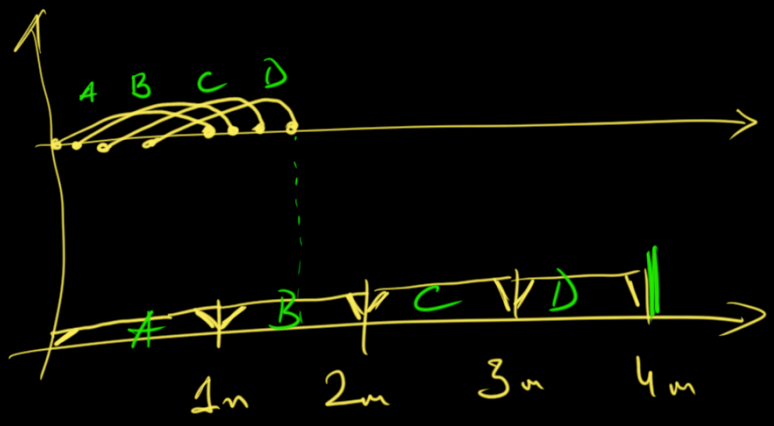
```
main {
  connectDatasource
  run();
}
```

? should you check the B, D, C availability during startup

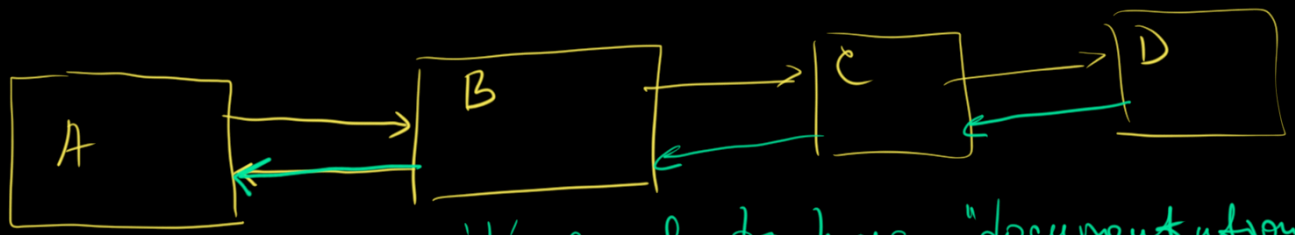
Yes

No



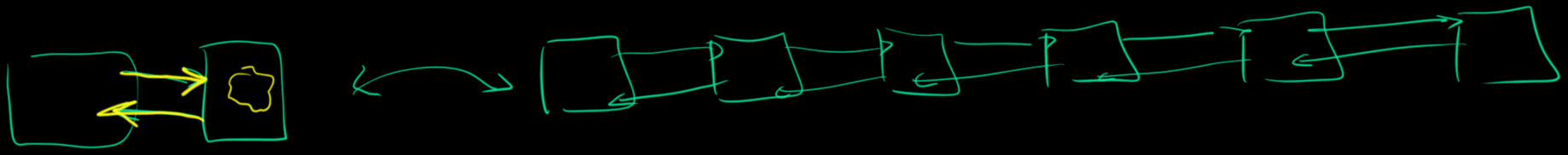


Retry, Cancellation, Timeout



timeout: 30s

- it's good to have "documentation" about average response time
- if we don't have connection diagram we can say nothing about timeout



total time = connection time + sending RQ + processing + getting response

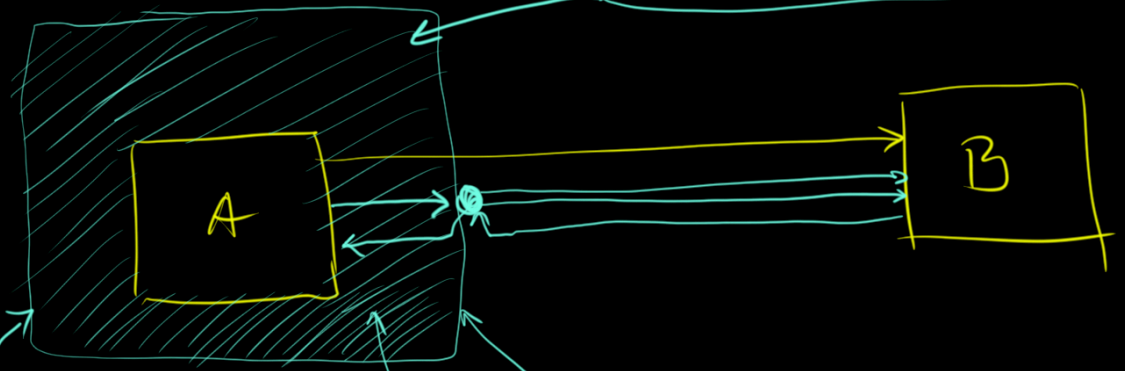
Σ time(sending + receiving) > processing VERY BAD

sending time + receiving time < processing time

Retry , Service Mesh

K8 Kubernetes

contains config.



1. timeout
2. retry strategy

this outer service is responsible for all retry / cancel strategy

you can run any service in "side car"

it can be configured w/o restarting service A

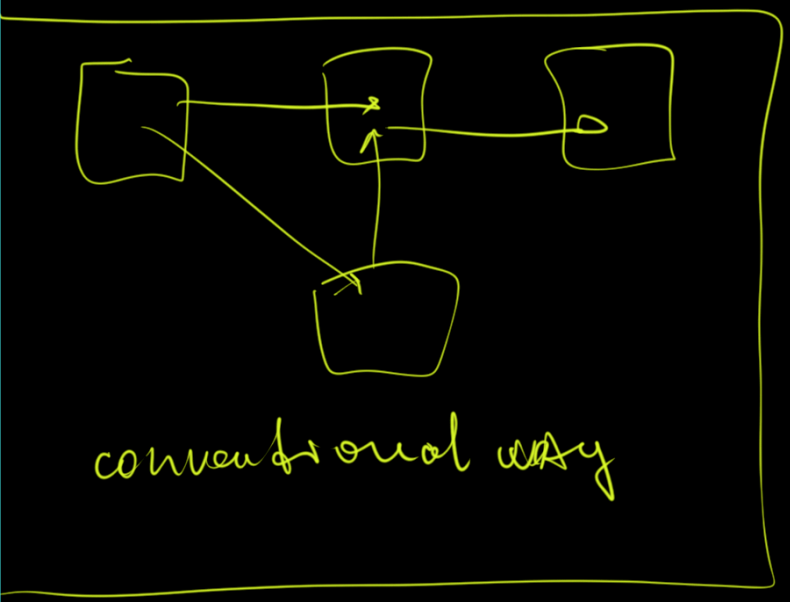
request

retry (request)

↑

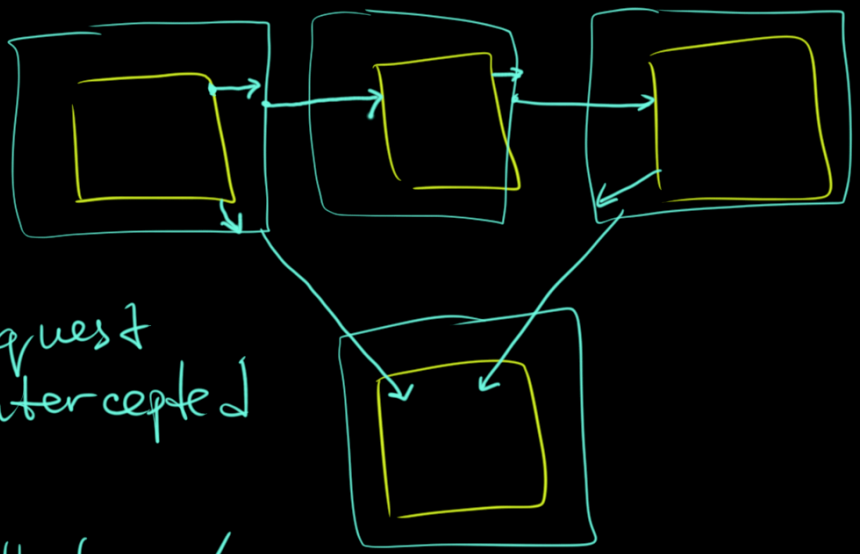
functionality of k8

~~Service Mesh~~



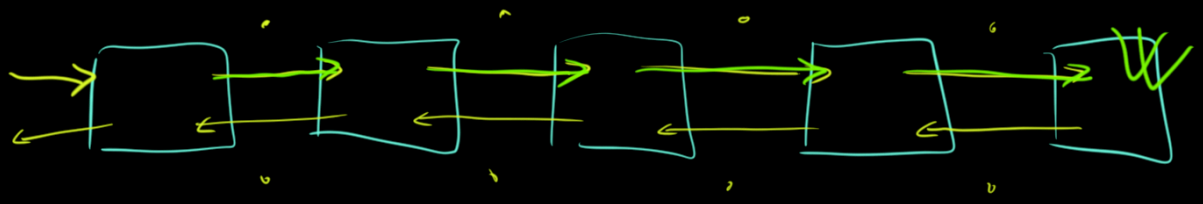
conventional way

Service Mesh



all request are intercepted and cancellation/retry policy is implemented on the Mesh level

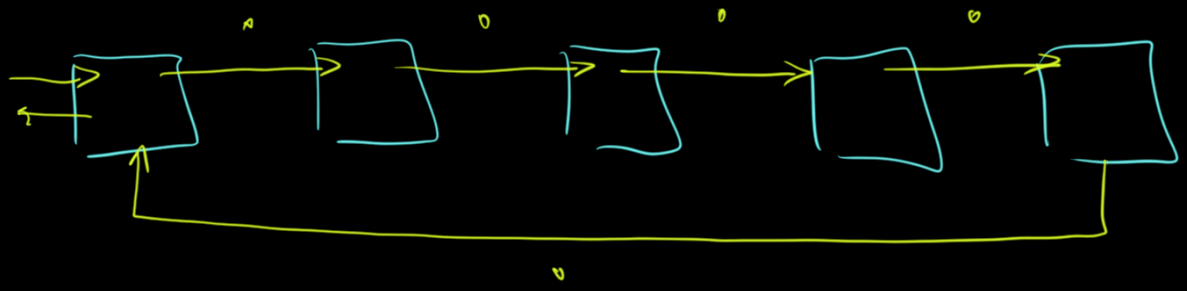
Request - Response Model



8 hops to get A response

the most probably our data is ready after 4 hops

Messaging



Your data is ready after 5 hops

5 < 8

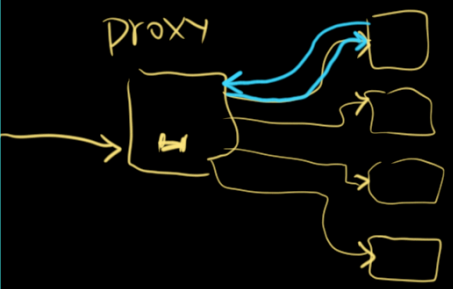
1 ms 100 rq/s

200 → 2 ms

500 → 5 ms

1000 → 10

isn't enough



- because communication overhead
- communicatiw costs too big