MICROSERVICES

JÉRÉMY PERROUAULT

DISJONCTEUR

SERVICE RÉSILIENT

DISJONCTEUR

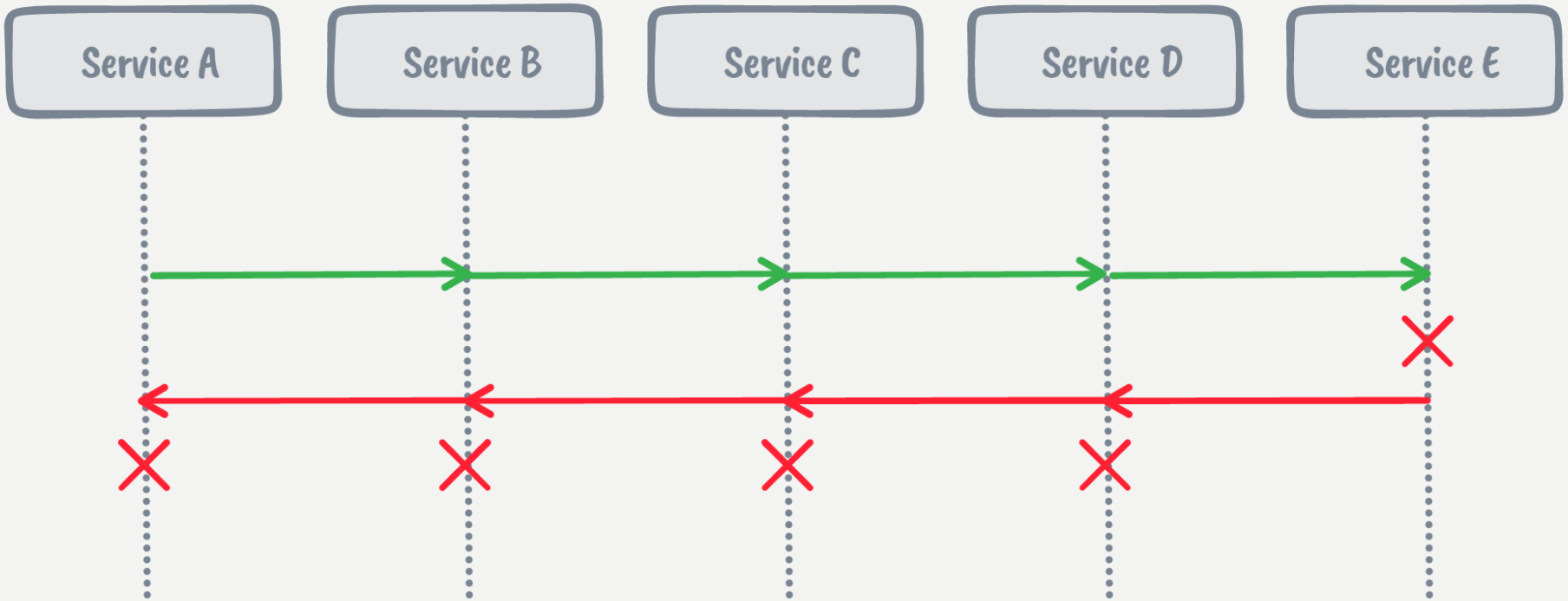
- « Service A » a besoin de « service B » pour composer une réponse
- « Service B » met plus de temps qu'habituellement pour répondre
  - Doit-on continuer à consommer ce service ? Sachant que son délai de réponse alonge celui de « A »
  - Doit-on temporiser sa consommation ? Et mettre en place un mécanisme de contournement ?

- Si « Service B » répond en 250ms au lieu de 10ms habituellement
  - Si on continue de l'appeler, le service risque de saturer encore plus …
  - … Il vaut donc mieux ne pas surcharger les appels, et le laisser reprendre ses capacités
  - Et de répondre avec un traitement alternatif en attendant

- Il nous faut des services résilients

DISJONCTEUR

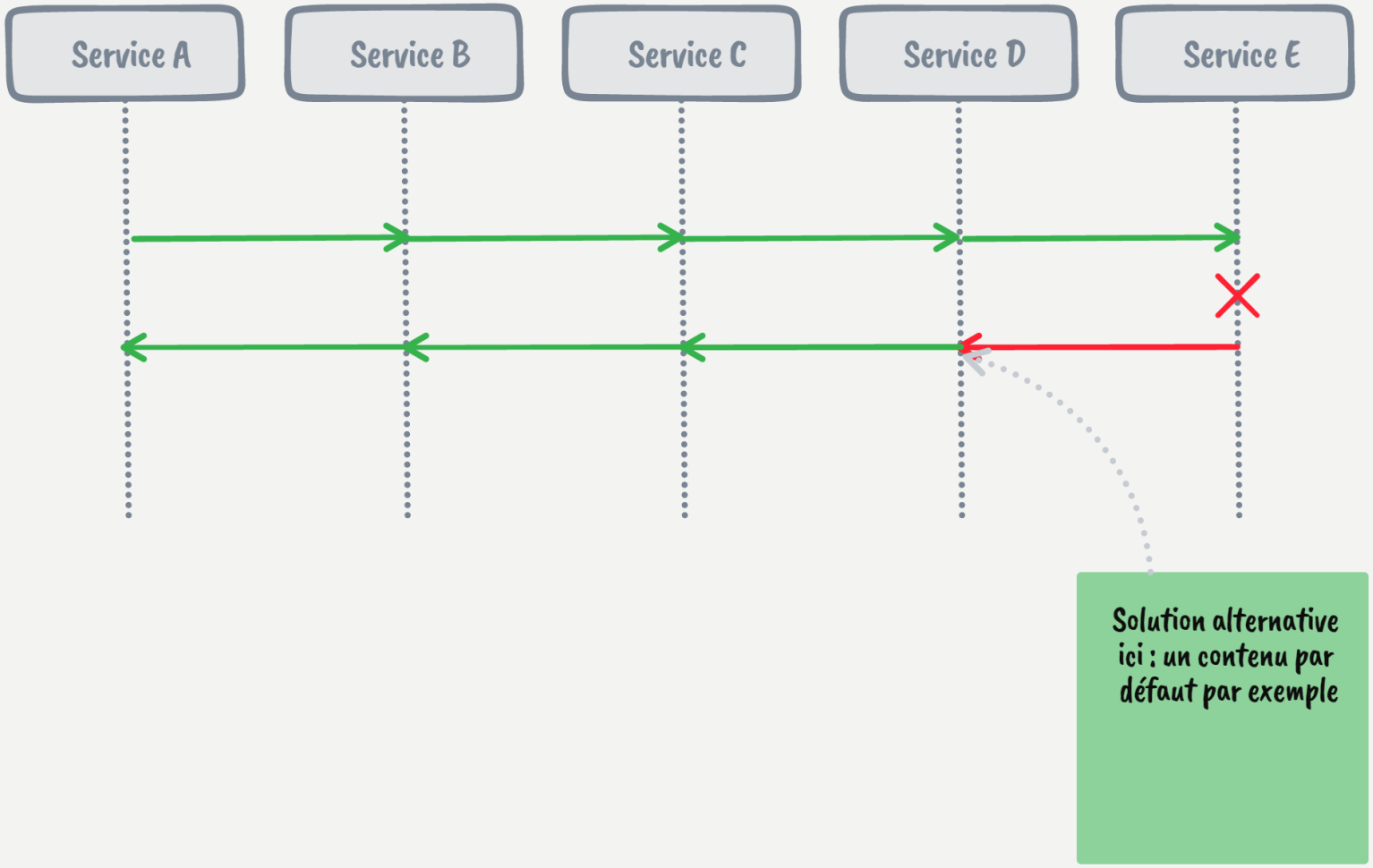
- Reprendre le scénario précédent, avec plus de services …
  - Le service en erreur, finira par mettre en erreur toute la chaine d'appel (effet domino)



DISJONCTEUR

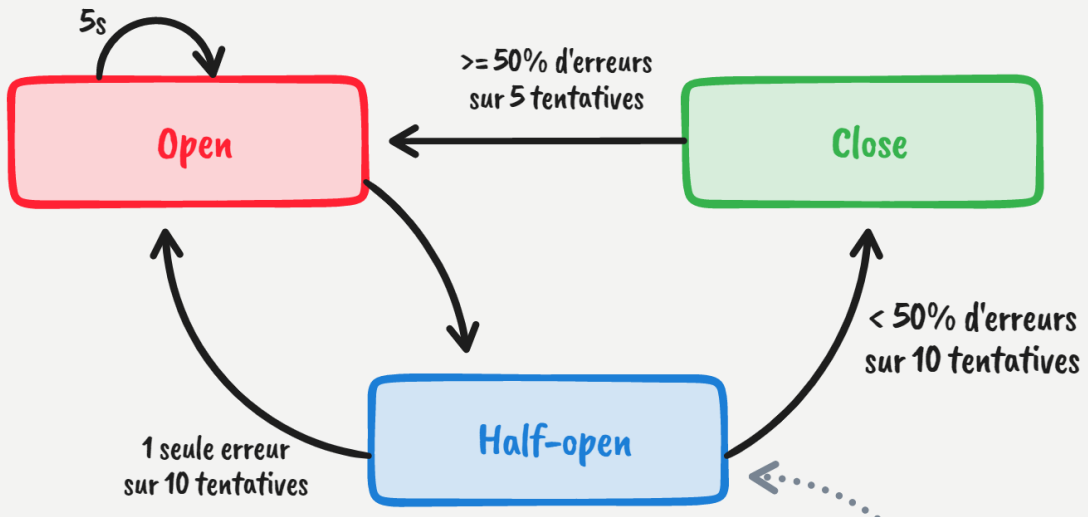
- Utilisation du Pattern « Circuit Breaker »
  - L'idée, c'est de fixer des règles pour lesquels un service est « indisponible »
    - Temps de latence trop long
    - Nombre d'erreurs atteint
  - Si une des règles est détectée, mise en place d'une réponse alternative
    - Cette solution de repli doit être rapide et toujours disponible
    - Cette solution a une durée, c'est la durée pendant laquelle le circuit est « ouvert »

DISJONCTEUR



DISJONCTEUR

Le circuit est ouvert après la 6ème tentatives, ayant eu 3 tentatives infructueuses. Cet état sera actif pendant 5s avant de basculer en état "semi-ouvert", puis en état "fermé".



Les erreurs peuvent être des temps de latence > durée fixée, ou un statut HTTP 5xx ou 408

Dans cet état, les requêtes sont de nouveaux exécutées. Si une seule erreur est détectée, retour dans l'état "ouvert".

DISJONCTEUR

- En complément, il est possible d'utiliser le Pattern « Bulkhead »
  - Permet de limiter les accès concurrentiels au service
- Les deux sont complémentaires
  - « Circuit Breaker » se paramètre sur le service procède aux appels
  - « Bulkhead » se paramètre sur le service appelé

- Lorsque le Pattern « Bulkhead » est implémenté, un nombre d'appels simulatanés trop important peut faire augmenter le taux d'échec, et le « Circuit Breaker » entrera plus rapidement en état « ouvert »

DISJONCTEUR

- Utilisation d'une technologie existante …
  - Hystrix (Netflix)
  - Resilience4j (JAVA)
  - Polly (.NET)
  - …

DISJONCTEUR

- Mettre en place un coupe circuit sur le service concerné

SPRING BOOT

EXEMPLE D'IMPLÉMENTATIONS

SPRING BOOT

- Ajouter la dépendance *spring-cloud-starter-circuitbreaker-resilicence4j*

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
</dependency>
```

- Injecter **CircuitBreakerFactory** à l'endroit nécessaire

SPRING BOOT

- Utiliser cette Factory pour paramétrer un fallback

```
String result = this.circuitBreakerFactory.create("nomService").run(
  () ->  restTemplate.getForObject("...", String.class),
  t -> "défaut"
);
```

- [Optionnel] Adapter les paramètres par défaut dans le fichier *application.properties*

```
resilience4j.circuitbreaker.instances.nomService.failure-rate-threshold = 50
resilience4j.circuitbreaker.instances.nomService.minimum-number-of-calls = 3
resilience4j.circuitbreaker.instances.nomService.permitted-number-of-calls-in-half-open-state = 10
resilience4j.circuitbreaker.instances.nomService.wait-duration-in-open-state = 5s
resilience4j.circuitbreaker.instances.nomService.sliding-window-size = 5
resilience4j.circuitbreaker.instances.nomService.sliding-window-type = count-based
resilience4j.circuitbreaker.instances.nomService.slow-call-duration-threshold = 1s
resilience4j.circuitbreaker.instances.nomService.slow-call-rate-threshold = 50
```

.NET6

EXEMPLE D'IMPLÉMENTATIONS

.NET6

- Ajouter le package *Microsoft.Extensions.Http.Polly*

```
<PackageReference Include="Microsoft.Extensions.Http.Polly" Version="6.0.10" />
```

.NET6

- Utiliser HttpClientFactory & HttpClient

```
builder.Services.AddHttpClient("nom-service", client => {
    client.BaseAddress = new Uri("lb://nom-service/");
})
.AddRandomLoadBalancer()
.AddTransientHttpErrorPolicy(policy => policy.CircuitBreakerAsync(3, TimeSpan.FromSeconds(5)));
```

```
var httpClient = _httpClientFactory.CreateClient("nom-service");

var fallbackForAnyException = Policy<string>
  .Handle<Exception>()
  .FallbackAsync(async (ct) => "défaut");

return await fallbackForAnyException.ExecuteAsync(async () => {
  return await httpClient.GetStringAsync("...");
});
```