Балансируем клиентские запросы

вместе со Spring Cloud

@aatarasoff

@aatarasoff

@aatarasoff

</> DEVELOPERBLOG.INFO

Балансируем клиентские запросы

вместе со Spring Cloud

Мнение докладчика может не совпадать с официальной позицией его работодателя, начальника, коллег или других специалистов.

Доклад ***не связан с инженерными решениями в Одноклассниках*** и базируется на проектах и библиотеках с открытым исходным кодом, личном опыте автора по их использованию или созданию.

Все представленные в докладе сведения, примеры, выводы и другую информацию вы можете использовать на свой страх и риск. За все ваши действия ответственность несёте только вы сами.

Все персонажи вымышлены, совпадения случайны.

spring-thrift-starter

Set of cool annotations that helps you building Thrift applications with Spring Boot

Java ★ 96 40

spring-cloud-marathon

Spring Cloud integration with Mesos and Marathon

Java ★ 27 11

spring-one-nio

PoC of integration between Spring Boot and One NIO

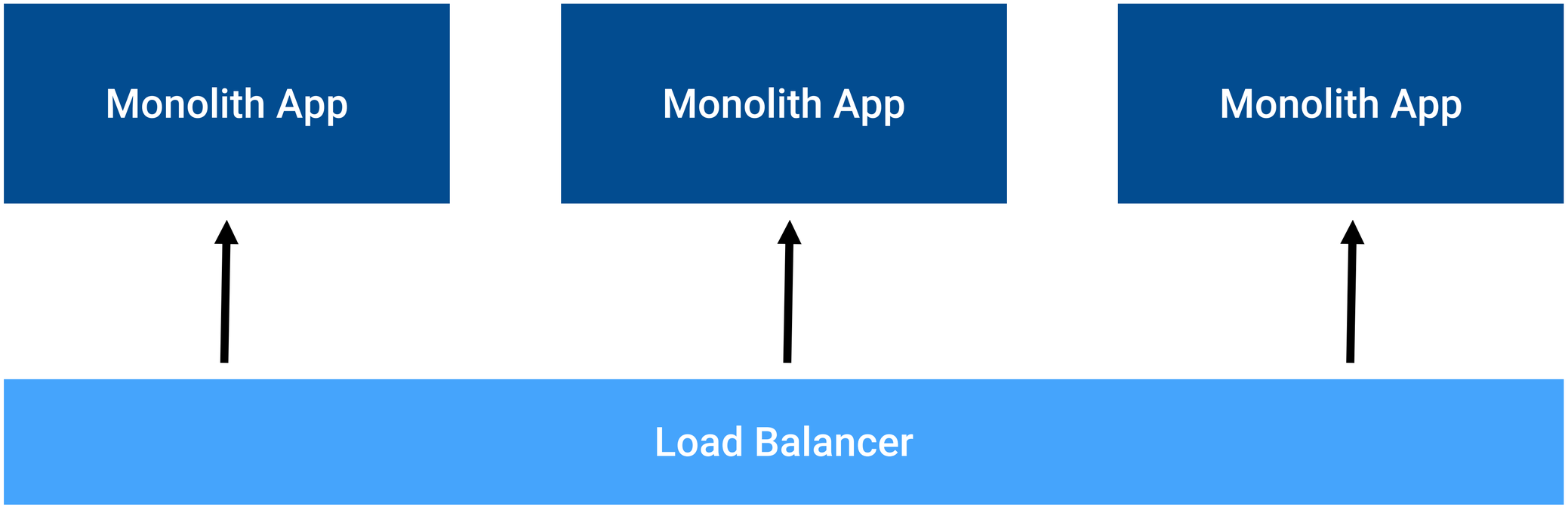
Java ★ 4

План такой

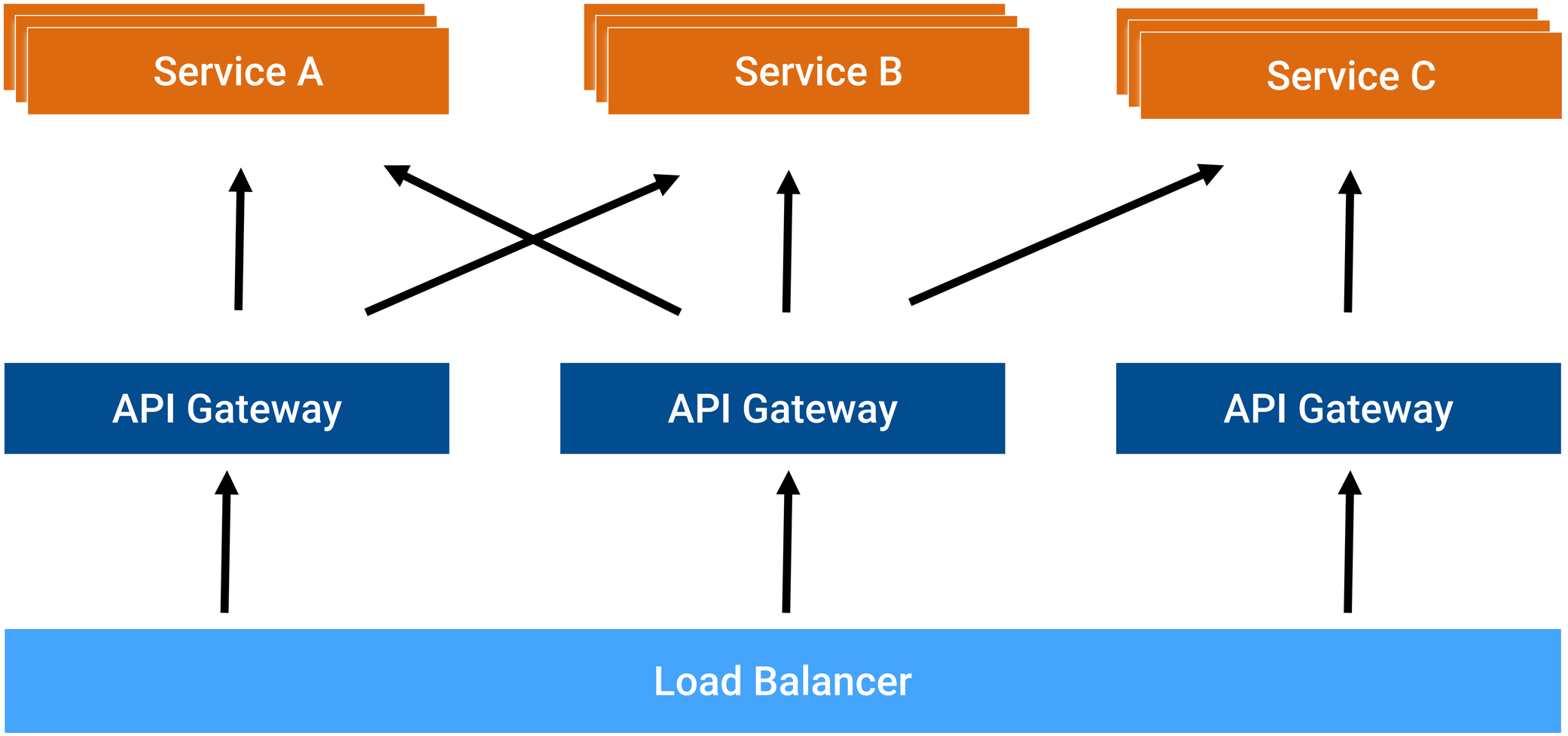
- Совсем чуть-чуть теории
- Много практики с демонстрацией в лабораторных условиях
- Выводы по ходу действа

В чём проблема?

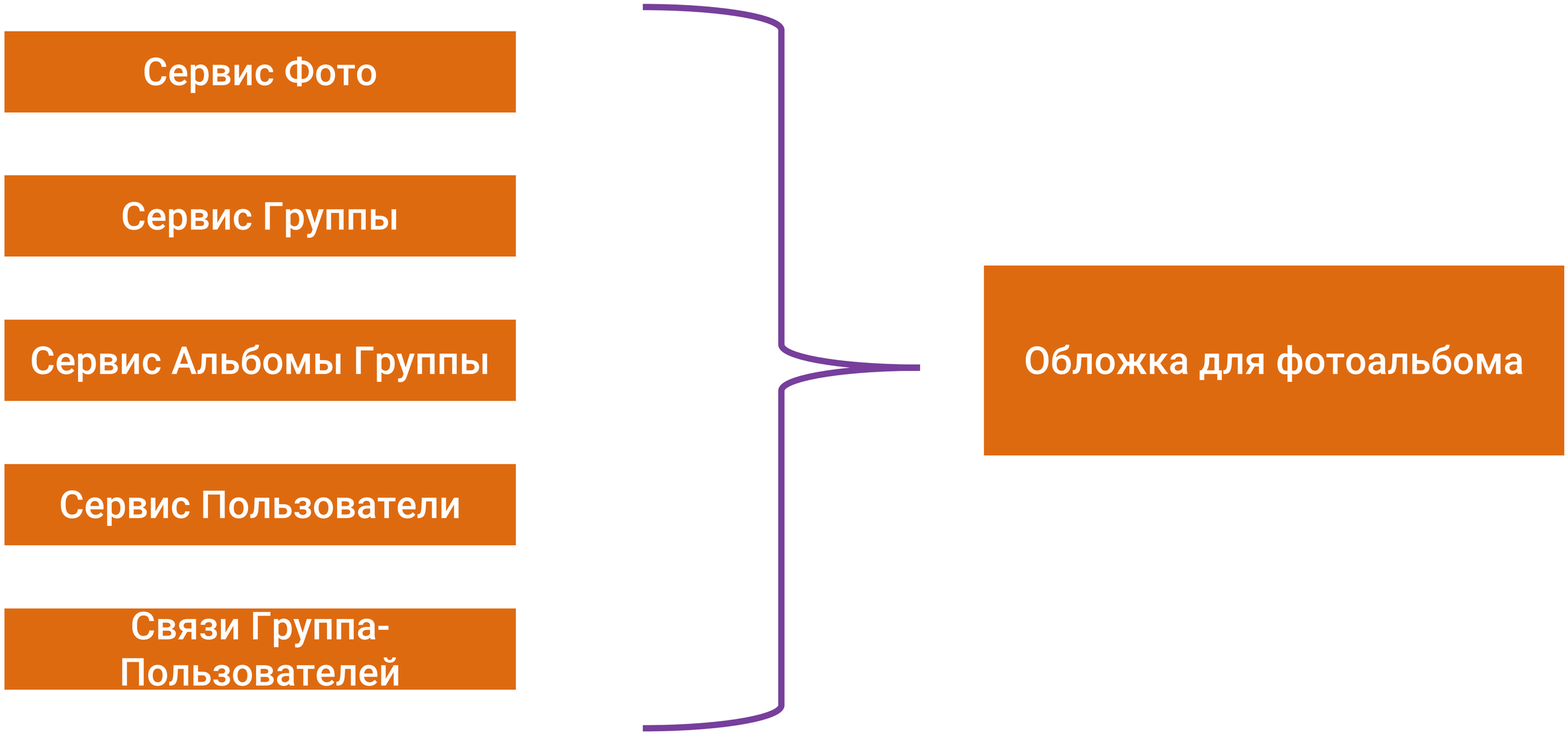
Long time ago



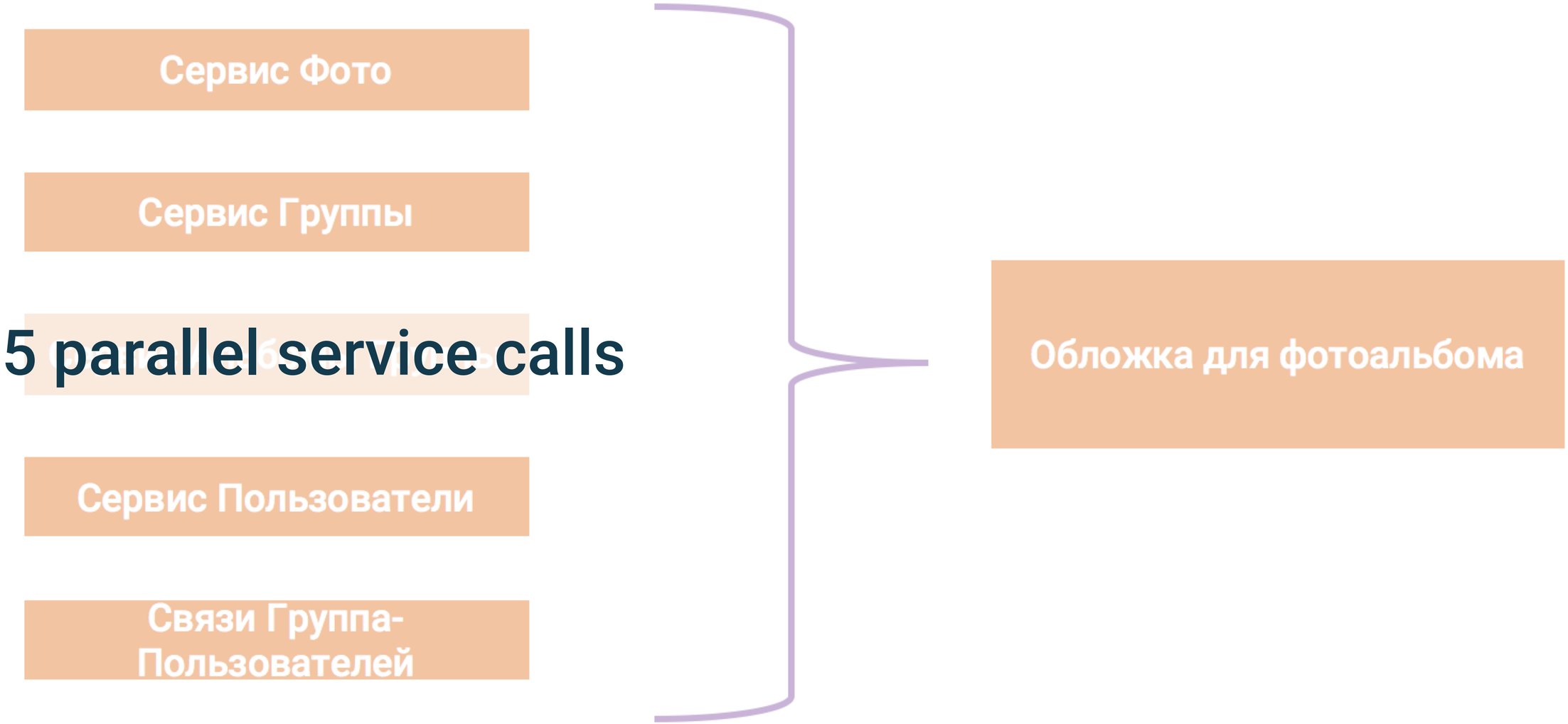
~~Микро~~сервисы



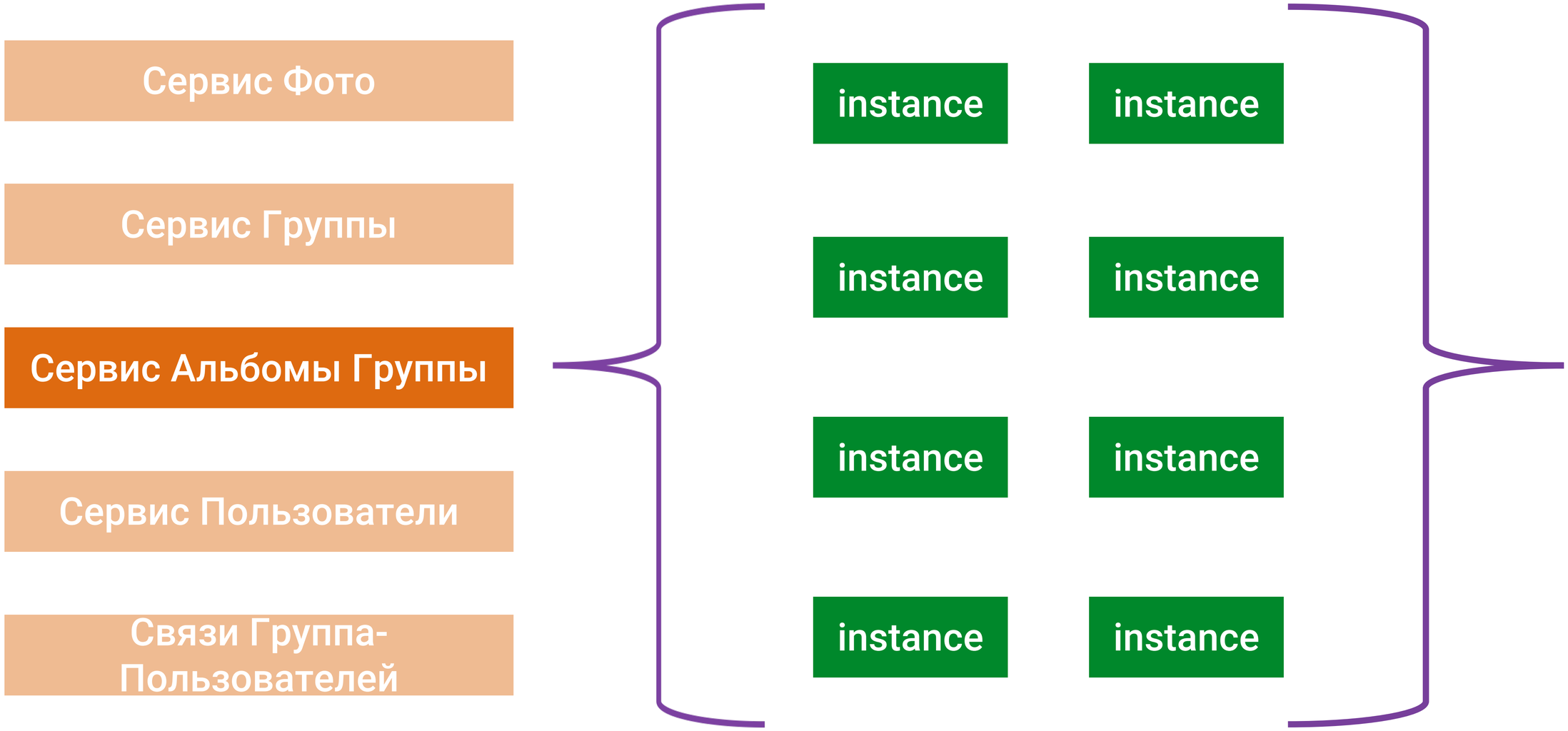
Задача из жизни



Всего то пяток вызовов сделать



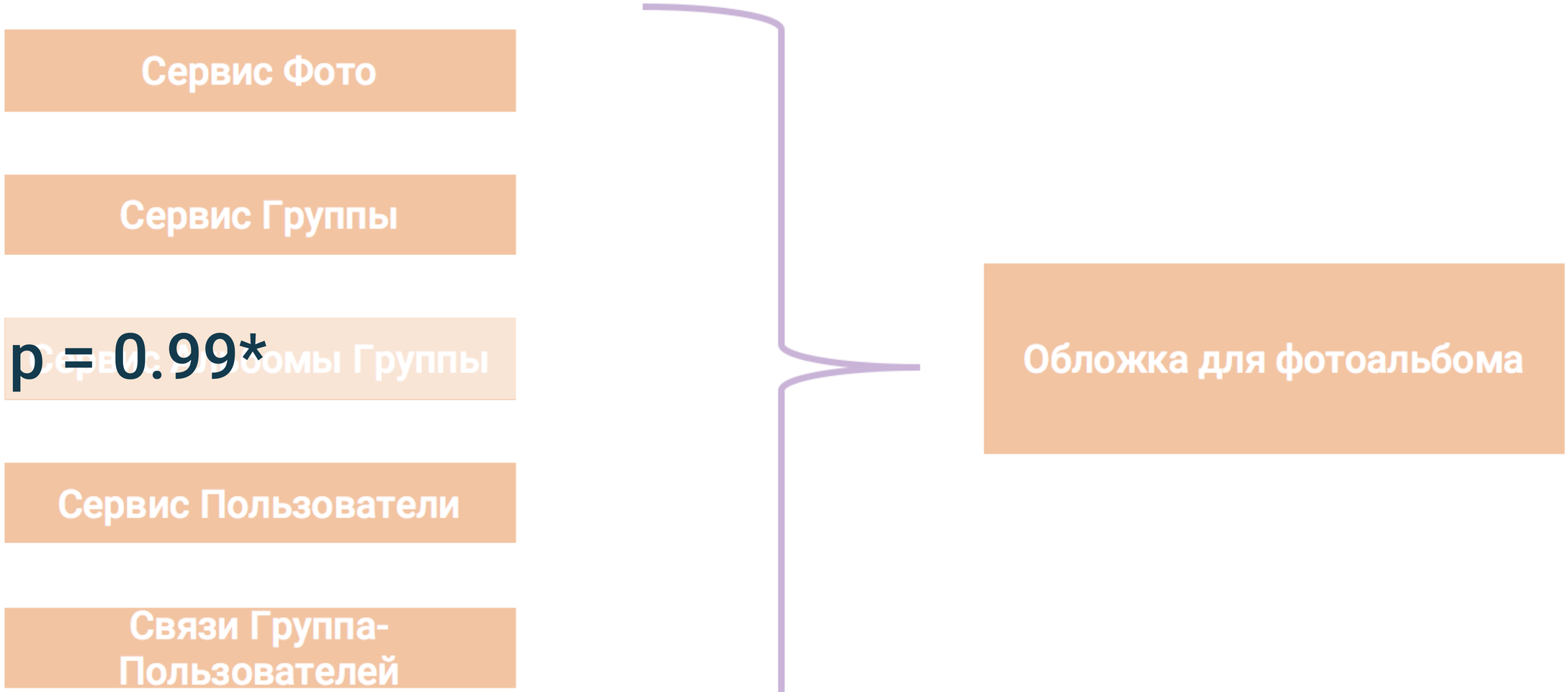
Кого же вызвать?



Идеальный мир

- Любой запрос всегда будет обработан в ожидаемое время
- Сеть абсолютно надёжна
- Обновление происходит мгновенно
- Придумайте ещё несколько сказочных постулатов

А так в реальной жизни



*** 0.99 не имеет отношения к реальности и взята для упрощения**

Ложь, наглая ложь и статистика

p = 0.99

p = 0.99

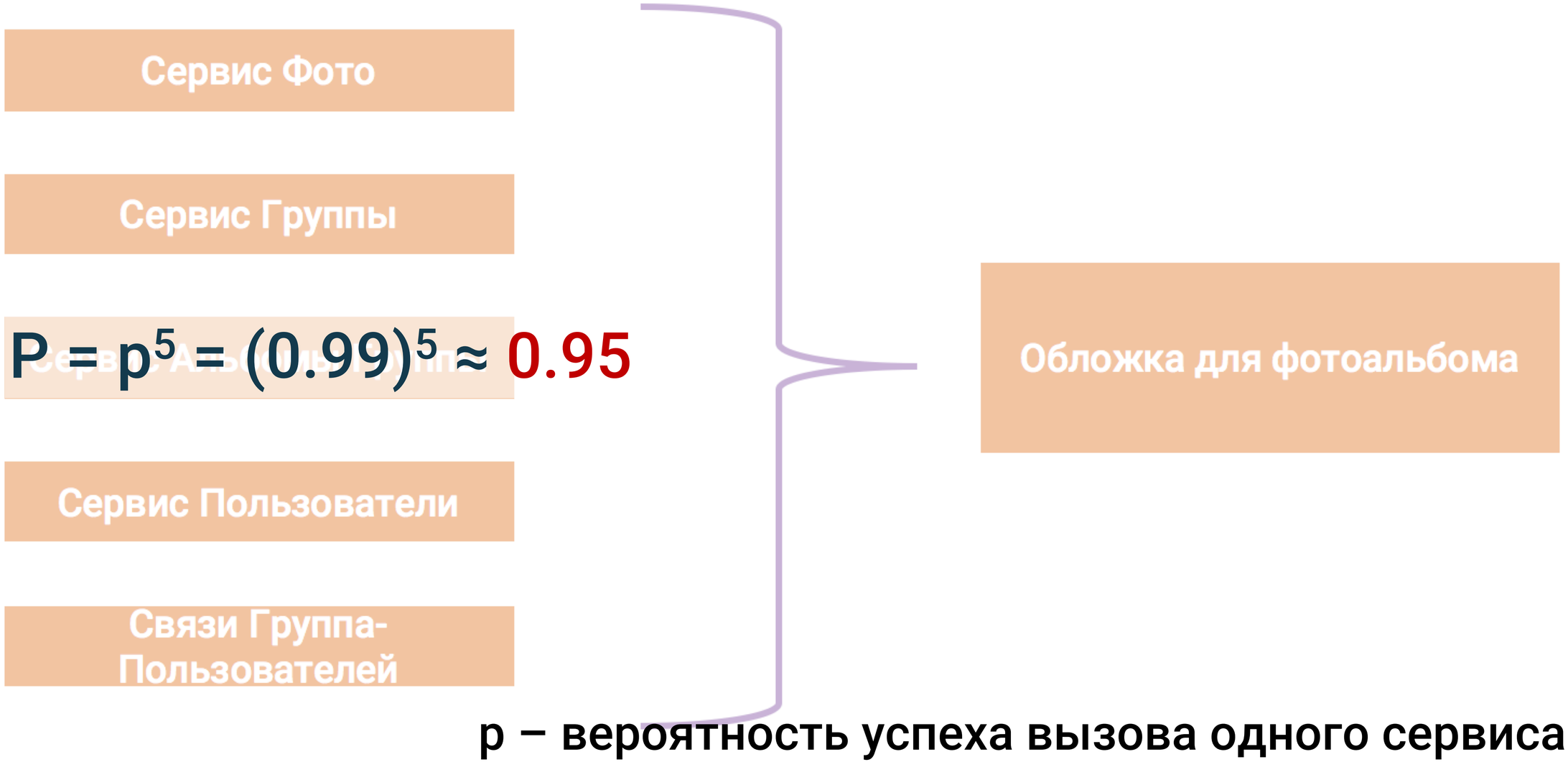
p = 0.99

p = 0.99

p = 0.99

p = 0.99 (?)

Ожидаемая неожиданность



p – вероятность успеха вызова одного сервиса

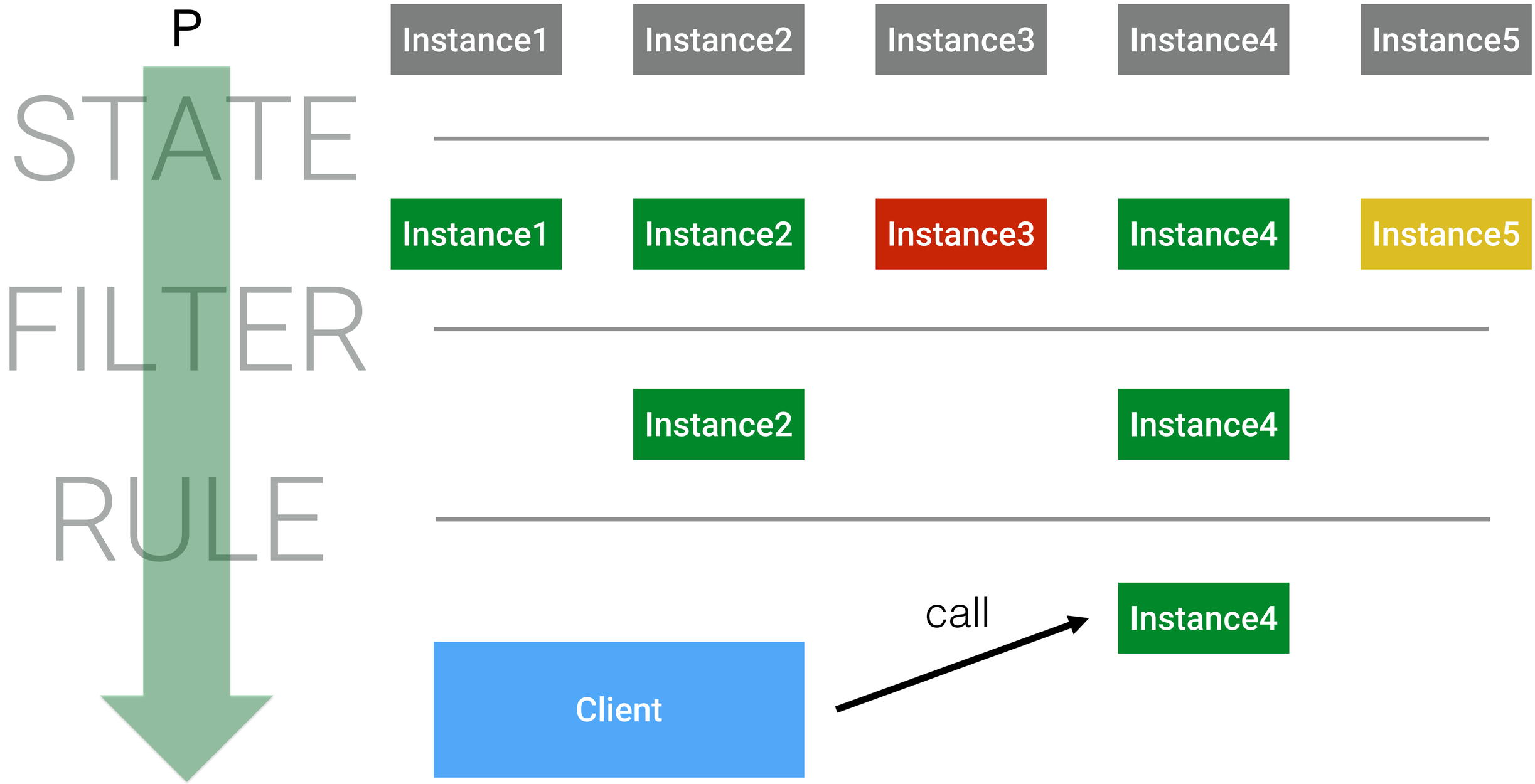
НЕидеальный мир

- Конкретный инстанс, группа инстансов или весь датацентр могут деградировать
- Сеть может мигать, падать и быть недоступна
- Обновление требует времени

Как мы можем увеличить вероятность успеха одного вызова?

Обратная воронка продаж

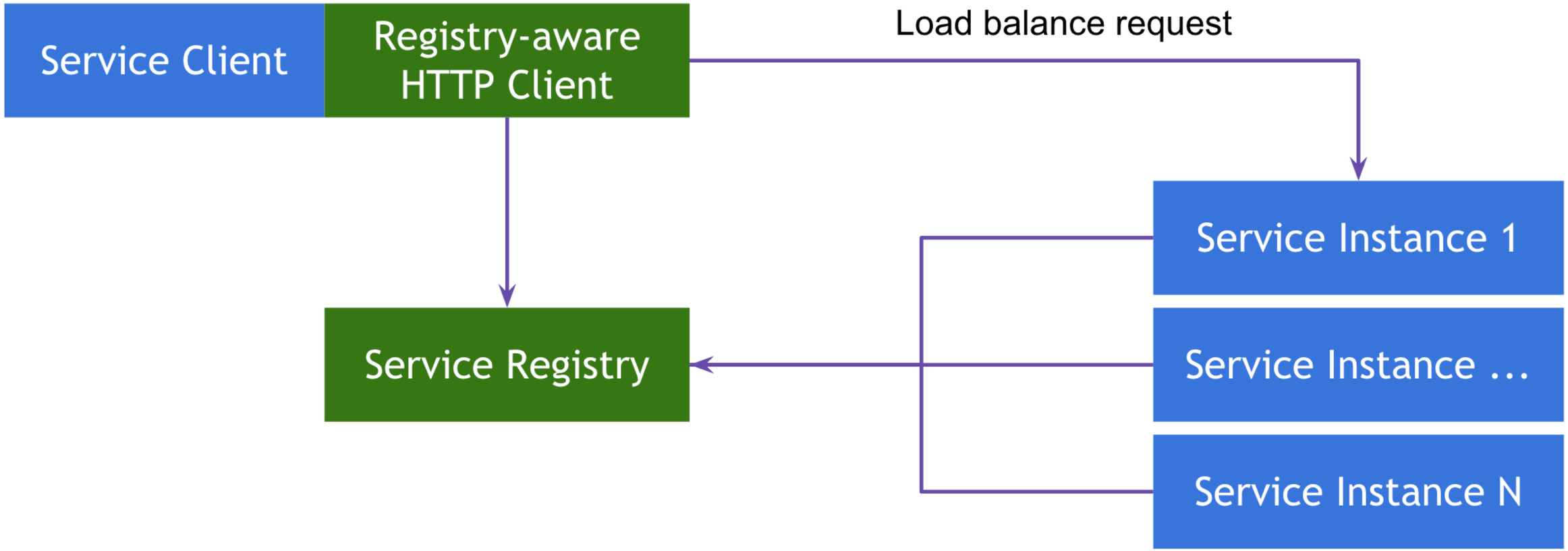
- Каждый инстанс это кандидат, который пытается себя «продать» по набору критериев


P
STATE
FILTER
RULE
Instance1
Instance2
Instance3
Instance4
Instance5
Instance1
Instance2
Instance3
Instance4
Instance5
Instance2
Instance4
call
Instance4
Client

Load Balancer

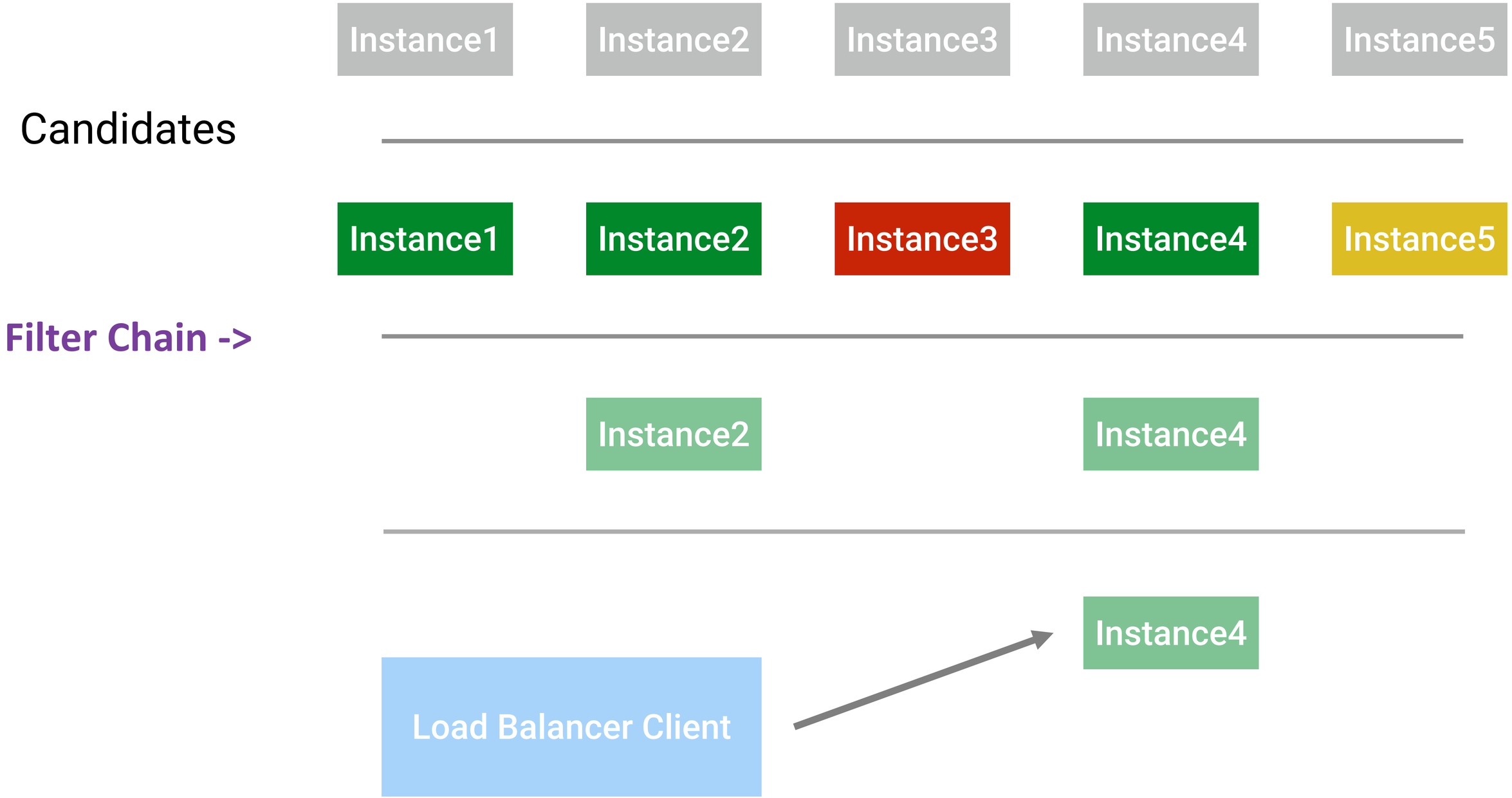
- Основан на Netflix Ribbon
- В принципе может быть использован вне стека Spring Cloud

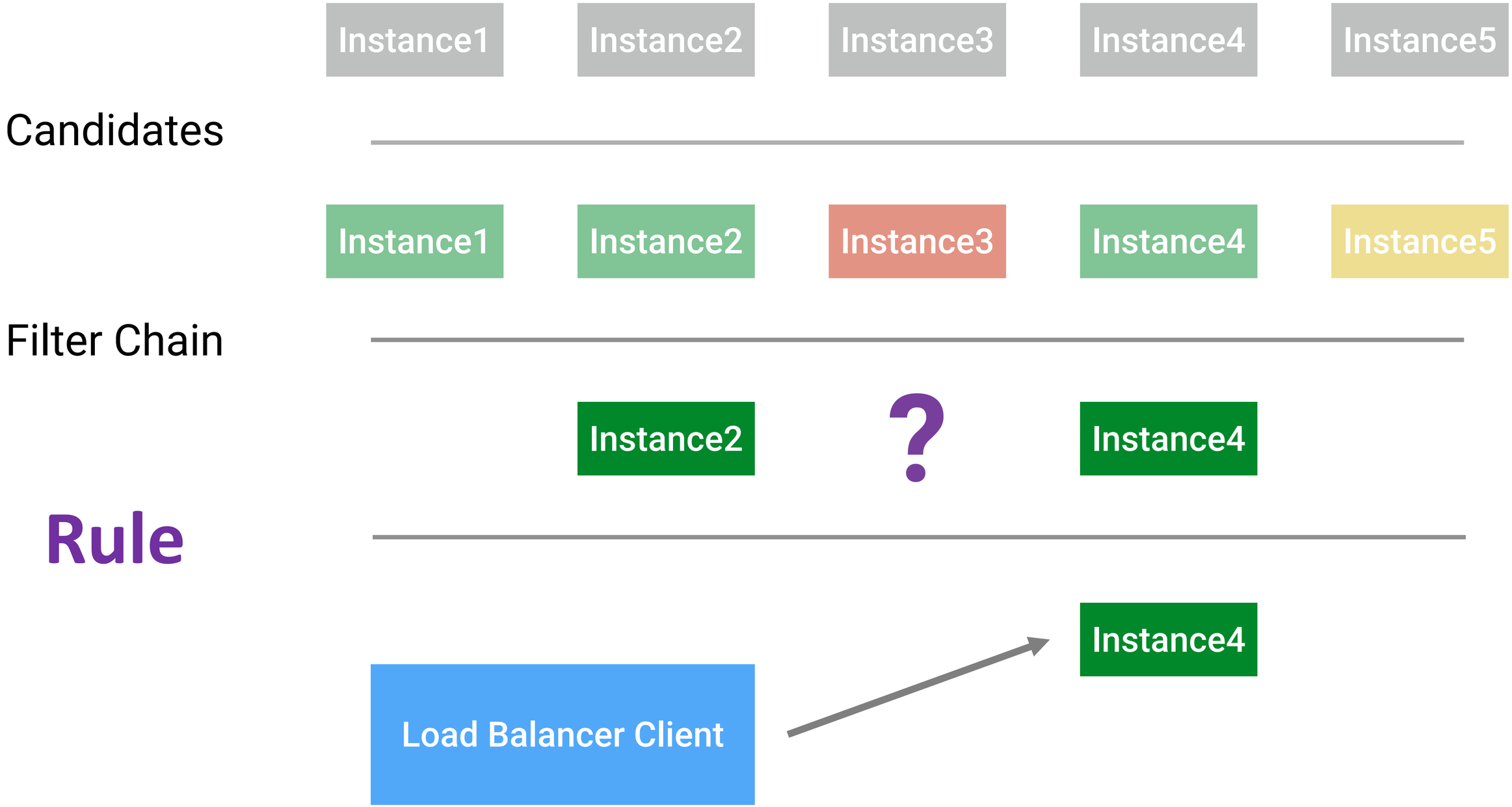
Client-Side Service Discovery



Цепочка фильтров

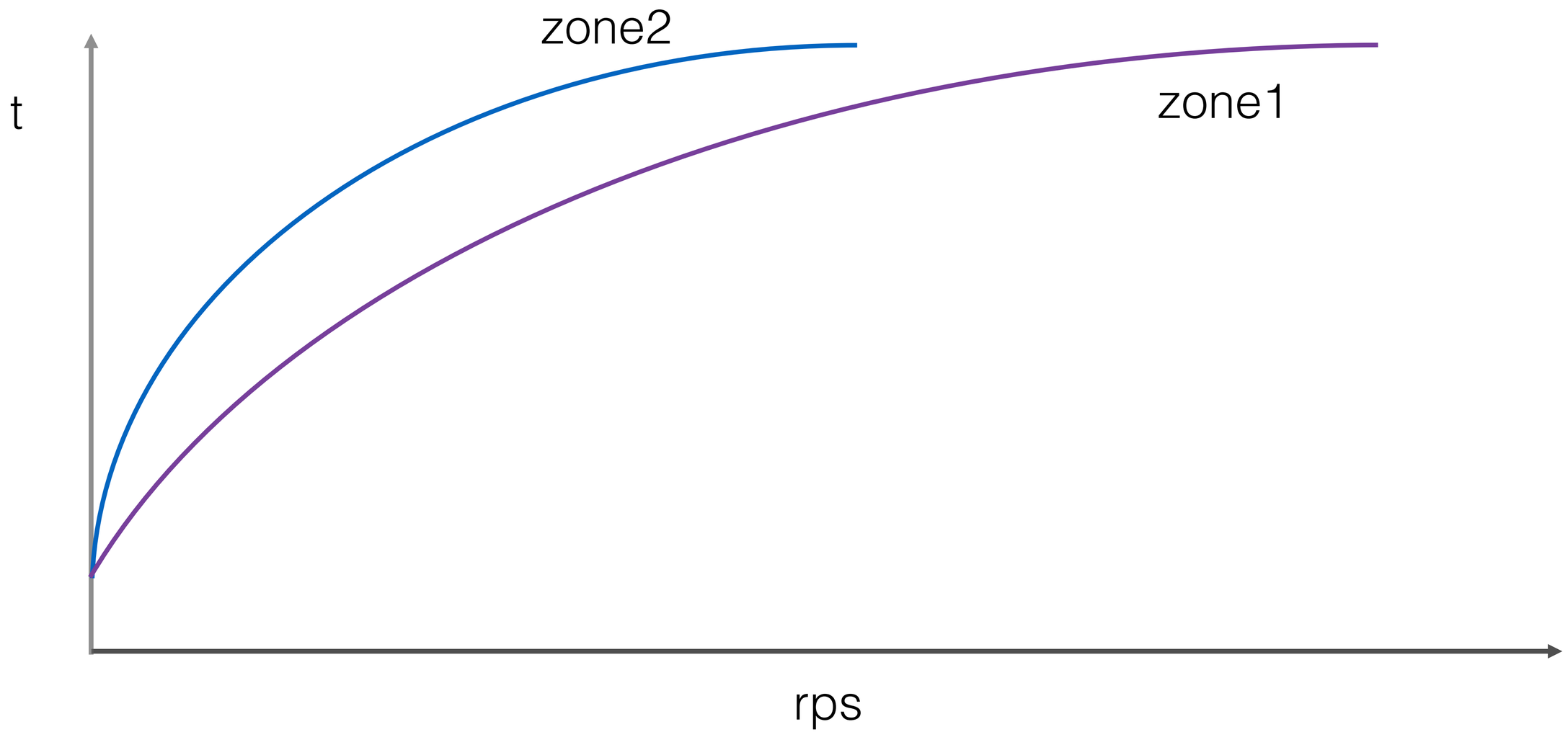
- Помогает увеличить шансы на корректное исполнение запроса
- Отсеивает «плохих» и «дорогих» кандидатов


Instance1
Instance2
Instance3
Instance4
Instance5
Candidates
Instance1
Instance2
Instance3
Instance4
Instance5
Filter Chain ->
Instance2
Instance4
Instance4
Load Balancer Client


Instance1
Instance2
Instance3
Instance4
Instance5
Candidates
Instance1
Instance2
Instance3
Instance4
Instance5
Filter Chain
Instance2
?
Instance4
Rule
Instance4
Load Balancer Client

Сетап для демо

- Настроим демо-стенд
- Разберем какие части есть и для чего они нам нужны

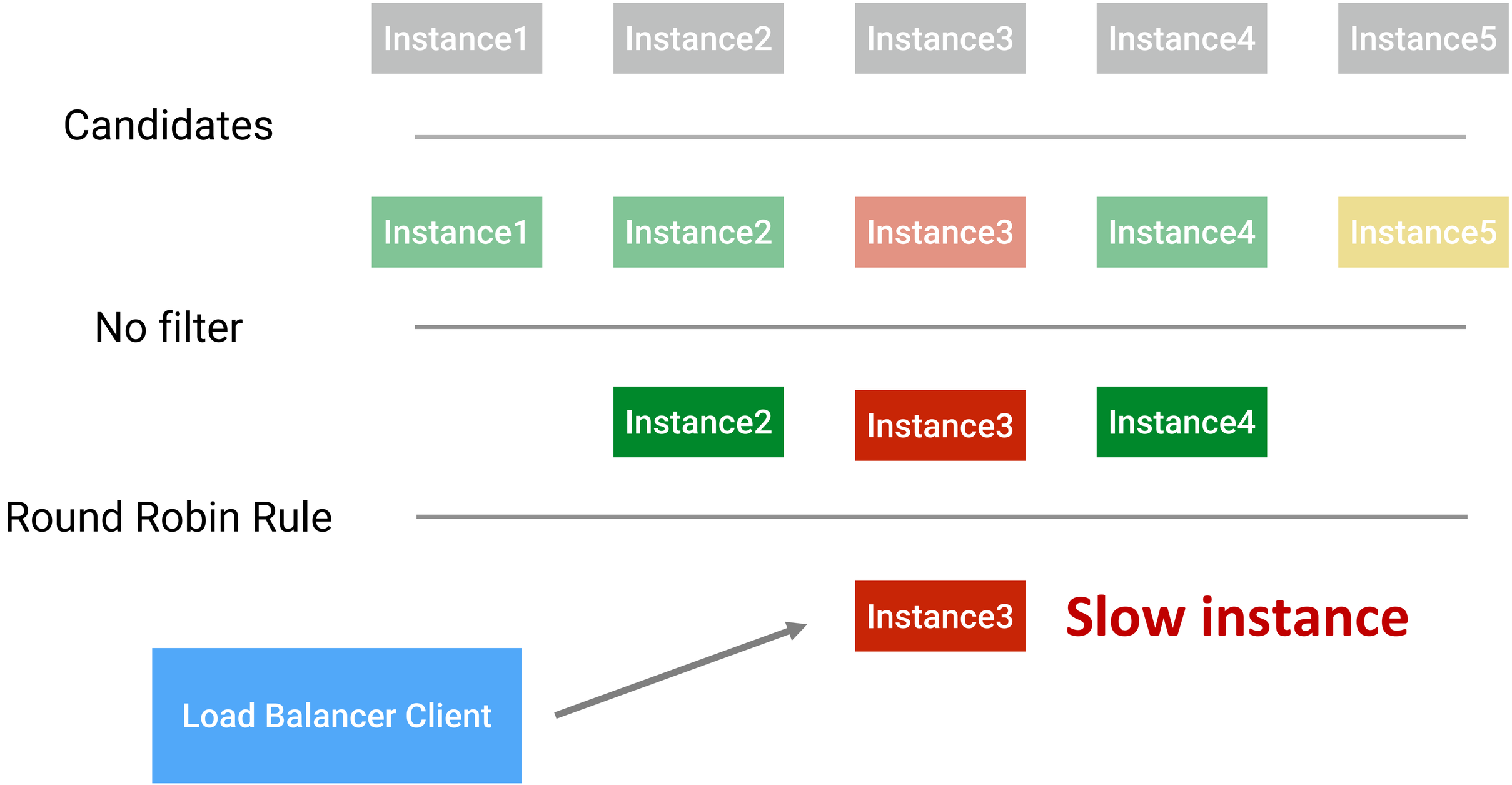

zone2
zone1
t
rps

Вопрос#1

- НАХУА?

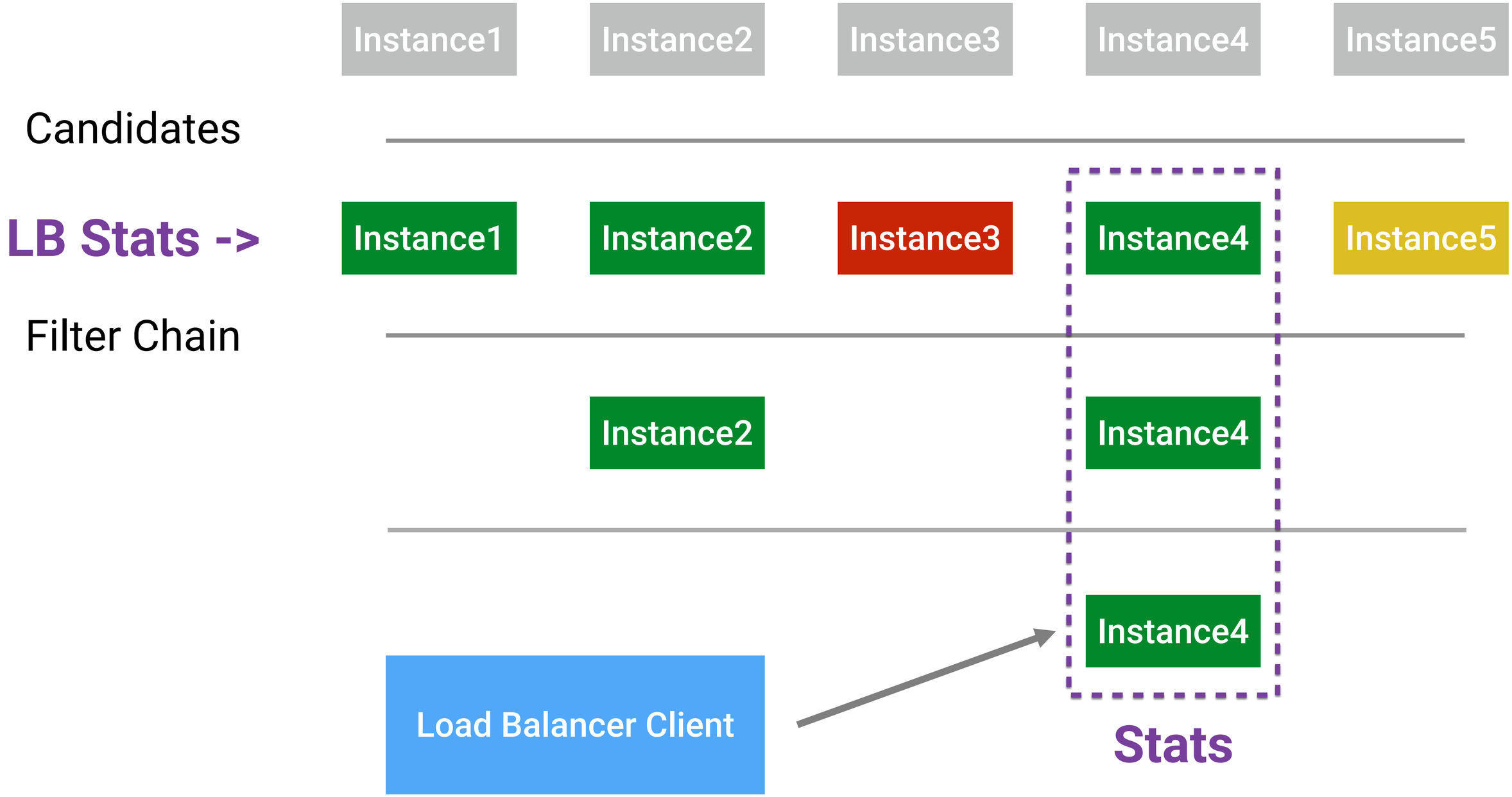
Демо#1

- Тупой и ещё тупее


Instance1
Instance2
Instance3
Instance4
Instance5
Candidates
Instance1
Instance2
Instance3
Instance4
Instance5
No filter
Instance2
Instance3
Instance4
Round Robin Rule
Instance3
Slow instance
Load Balancer Client

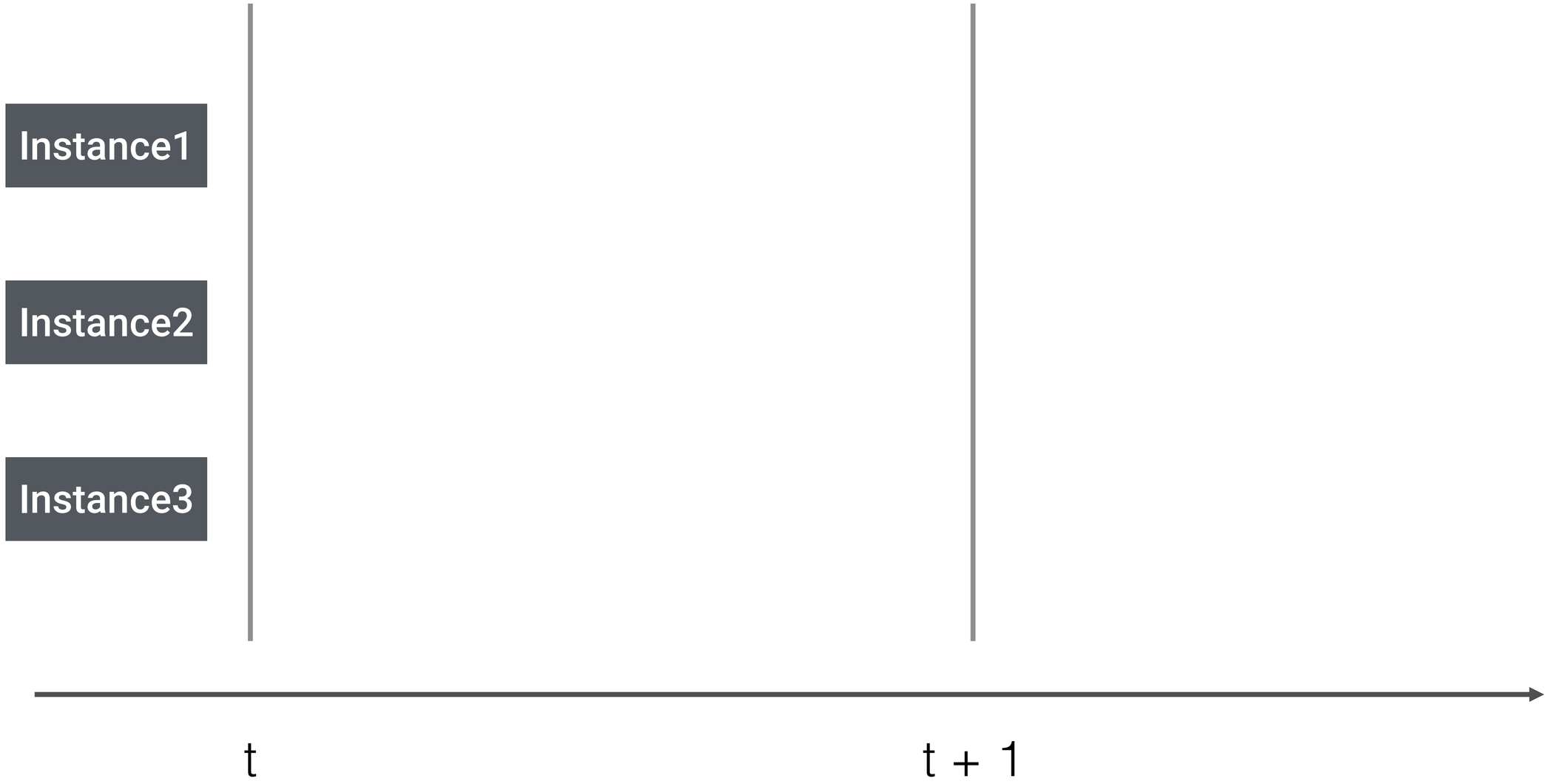
Вопрос#2

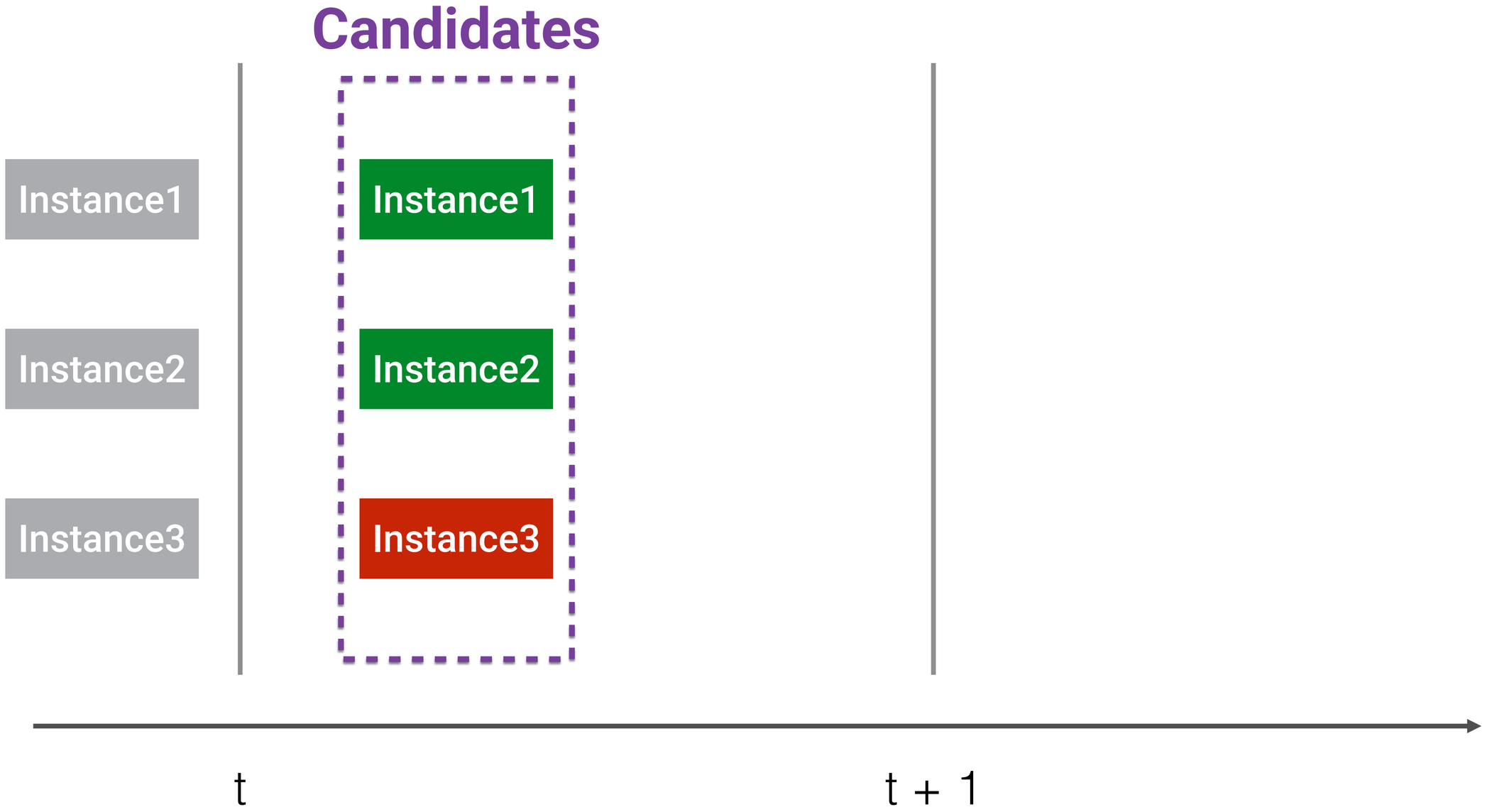
- Зачем вызывать «плохой» инстанс?

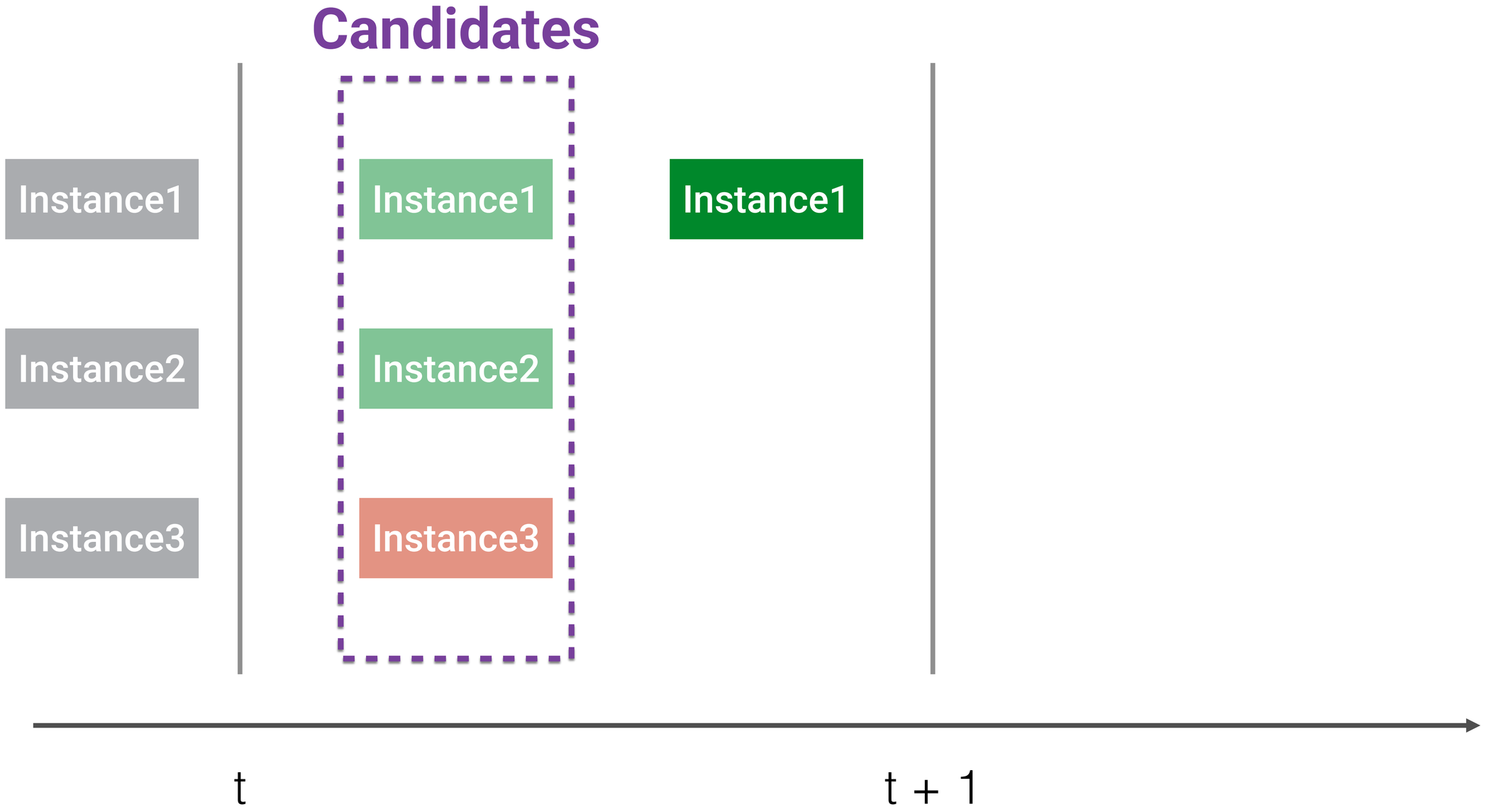

Instance1
Instance2
Instance3
Instance4
Instance5
Candidates
LB Stats ->
Instance1
Instance2
Instance3
Instance4
Instance5
Filter Chain
Instance2
Instance4
Instance4
Load Balancer Client
Stats

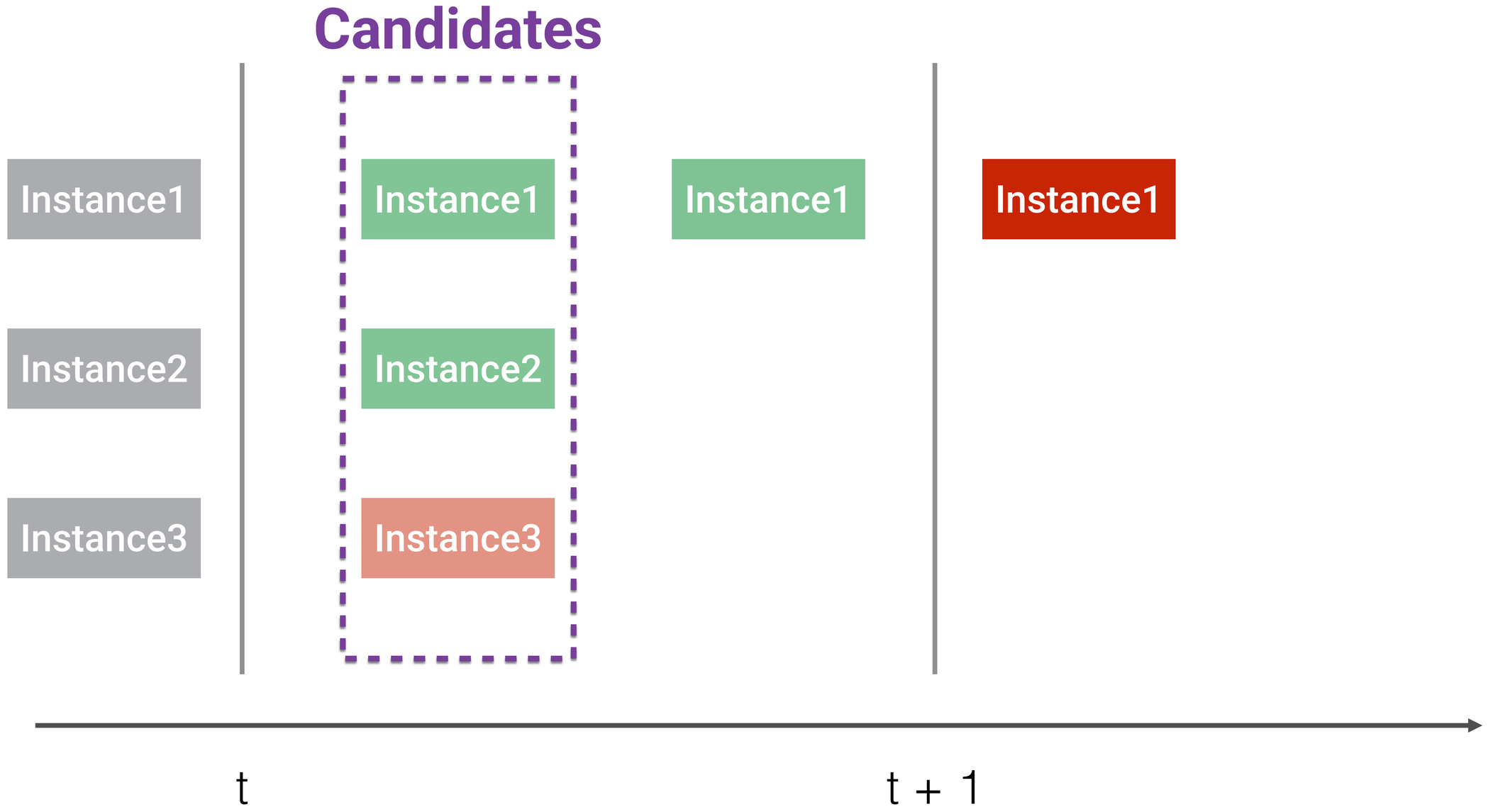
Демо#2

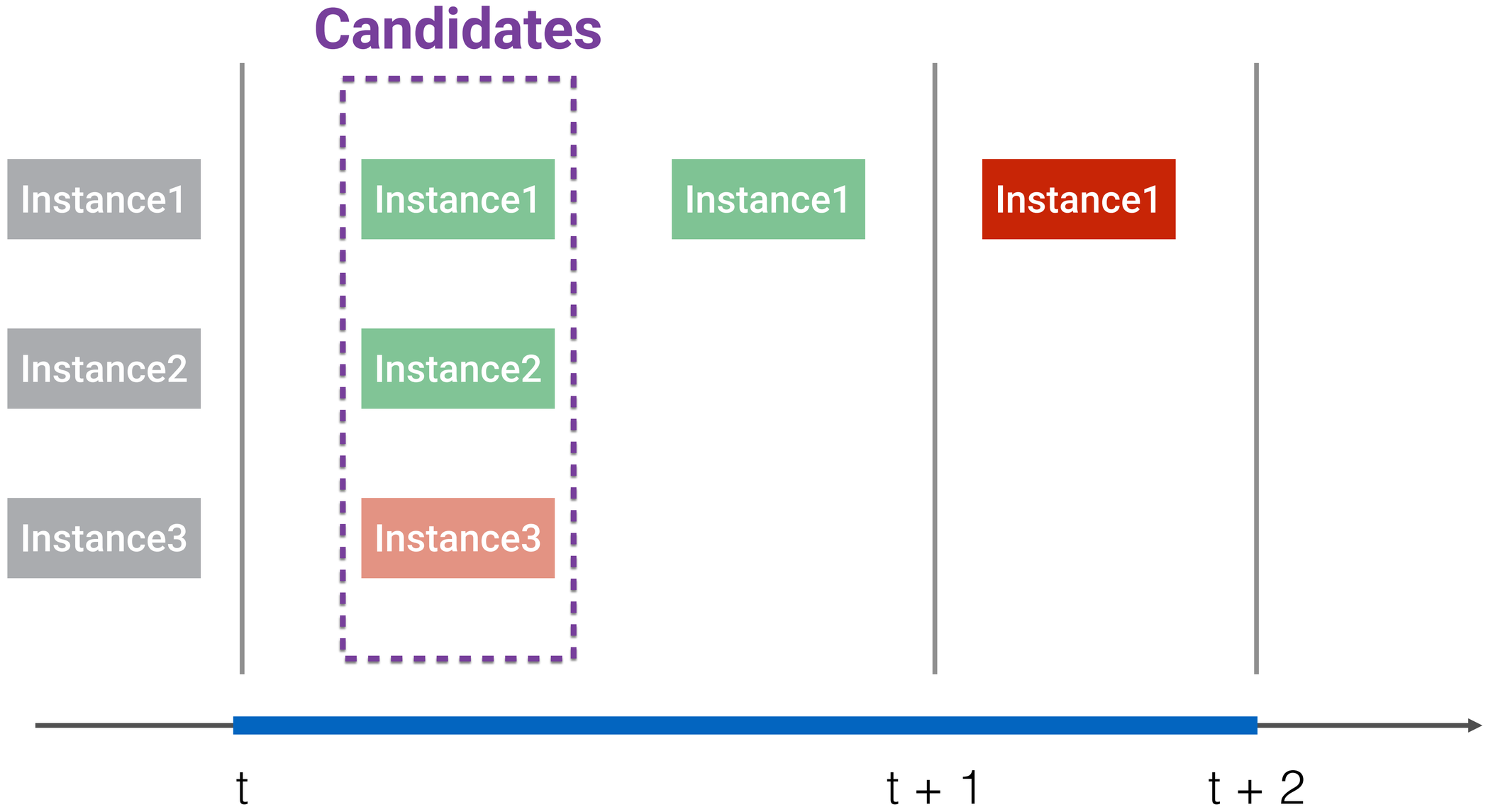
- Минимизация ущерба


Instance1
Instance2
Instance3
t
t + 1


Candidates
Instance1
Instance2
Instance3
Instance1
Instance2
Instance3
t
t + 1


Candidates
Instance1
Instance2
Instance3
Instance1
Instance2
Instance3
Instance1
t
t + 1


Candidates
Instance1
Instance2
Instance3
Instance1
Instance2
Instance3
Instance1
Instance1
t
t + 1

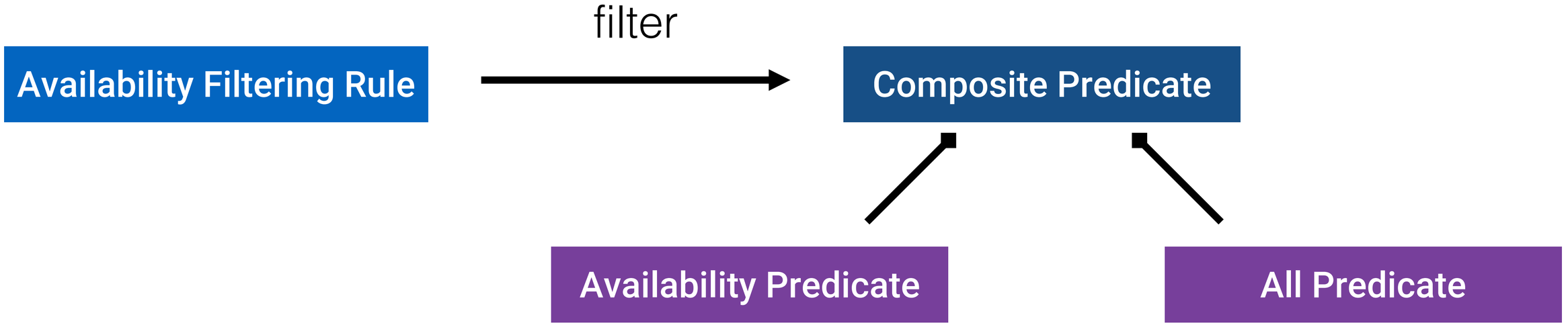

Candidates
Instance1
Instance2
Instance3
Instance1
Instance2
Instance3
Instance1
Instance1
t
t + 1
t + 2

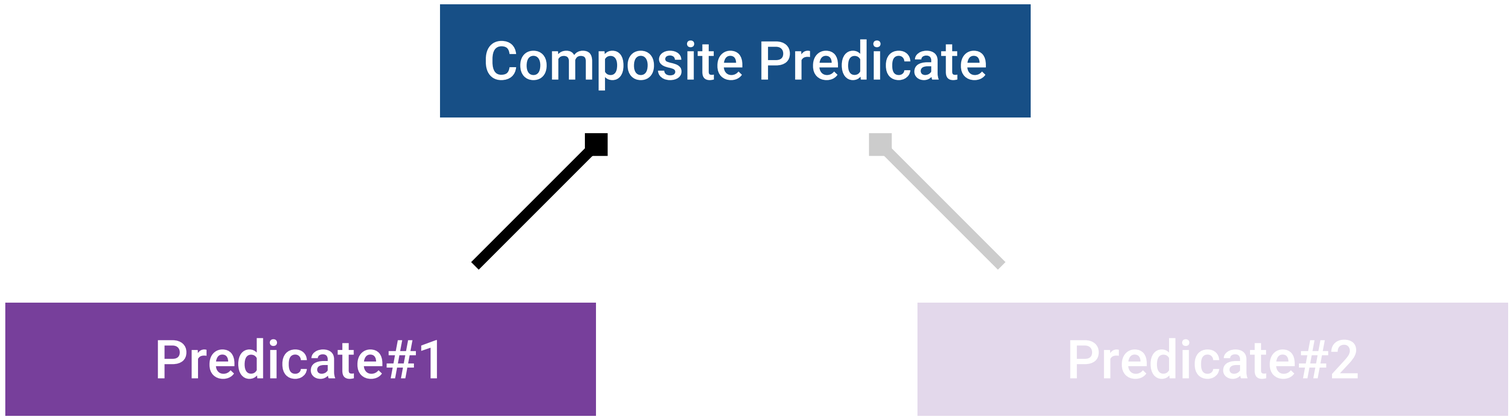
Вопрос#3

- Можно ли реагировать на изменения оперативнее?

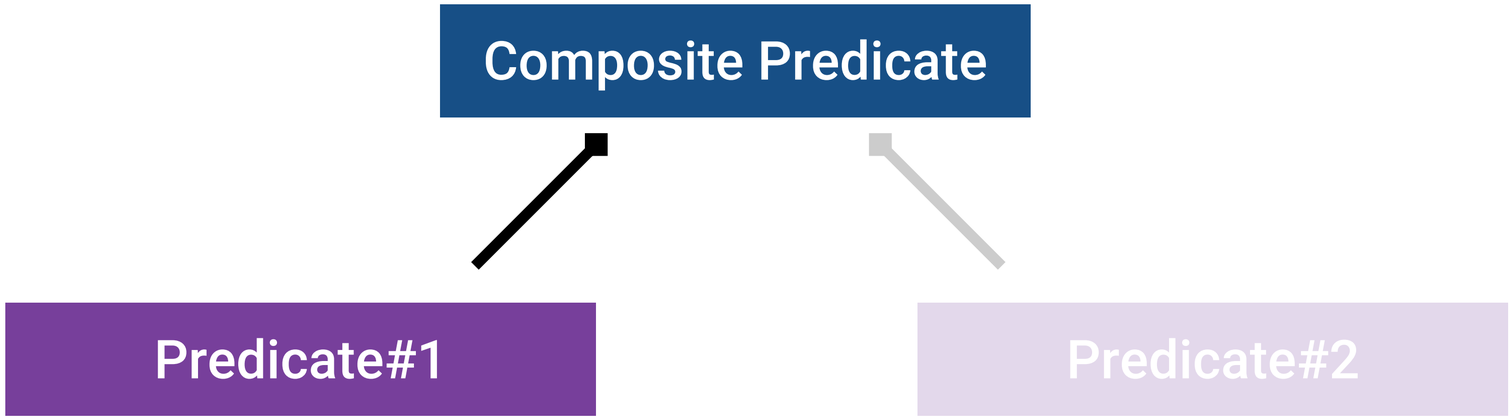
Демо#3

- Сила в ограничении

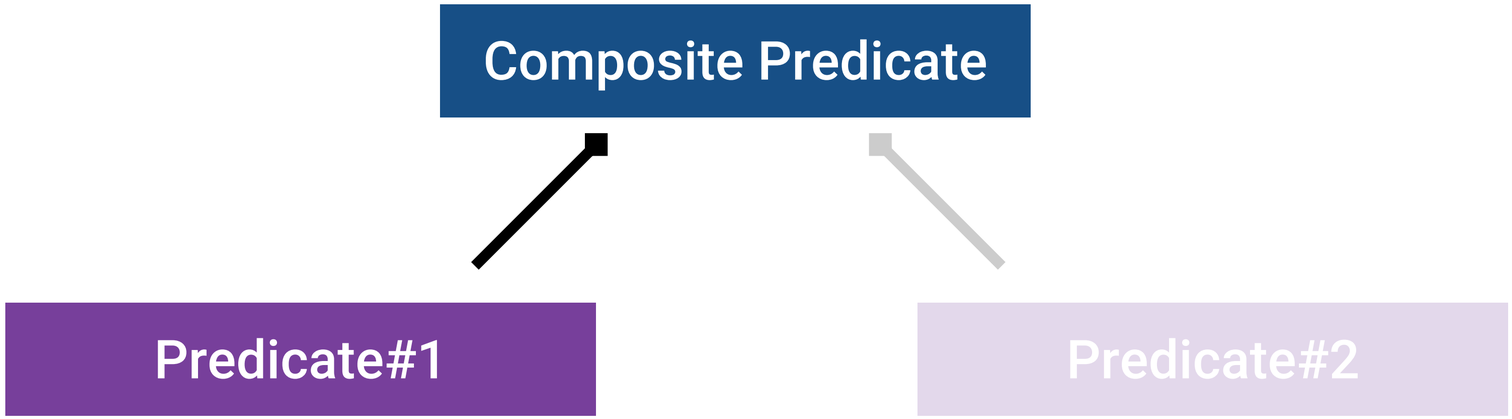

filter
Availability Filtering Rule
Composite Predicate
Availability Predicate
All Predicate



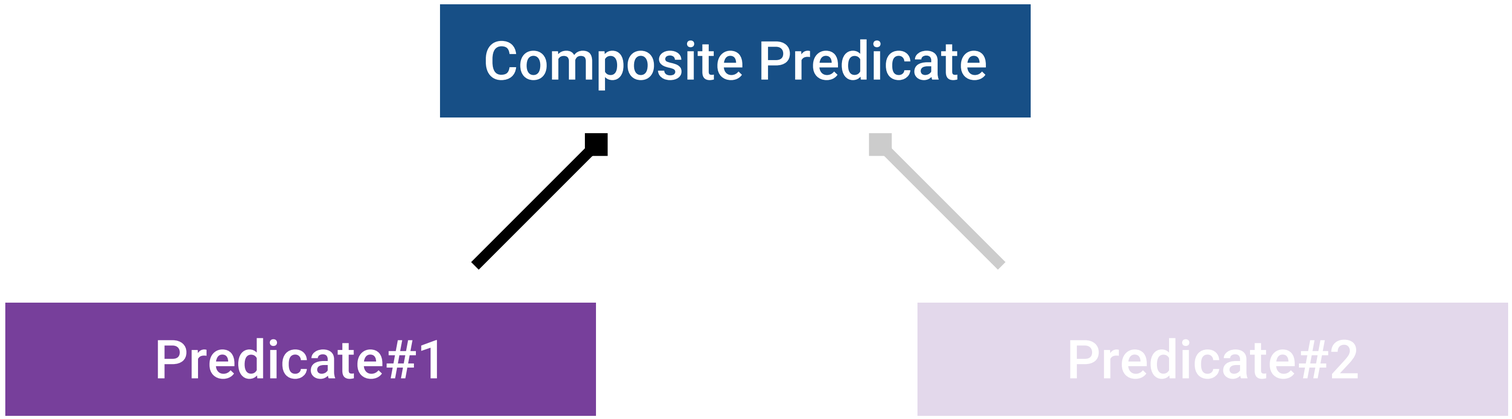

```
CompositePredicateBuilder
   int minimalFilteredServers = 1
   float minimalFilteredPercentage = 0


predicate.filter(servers)
```



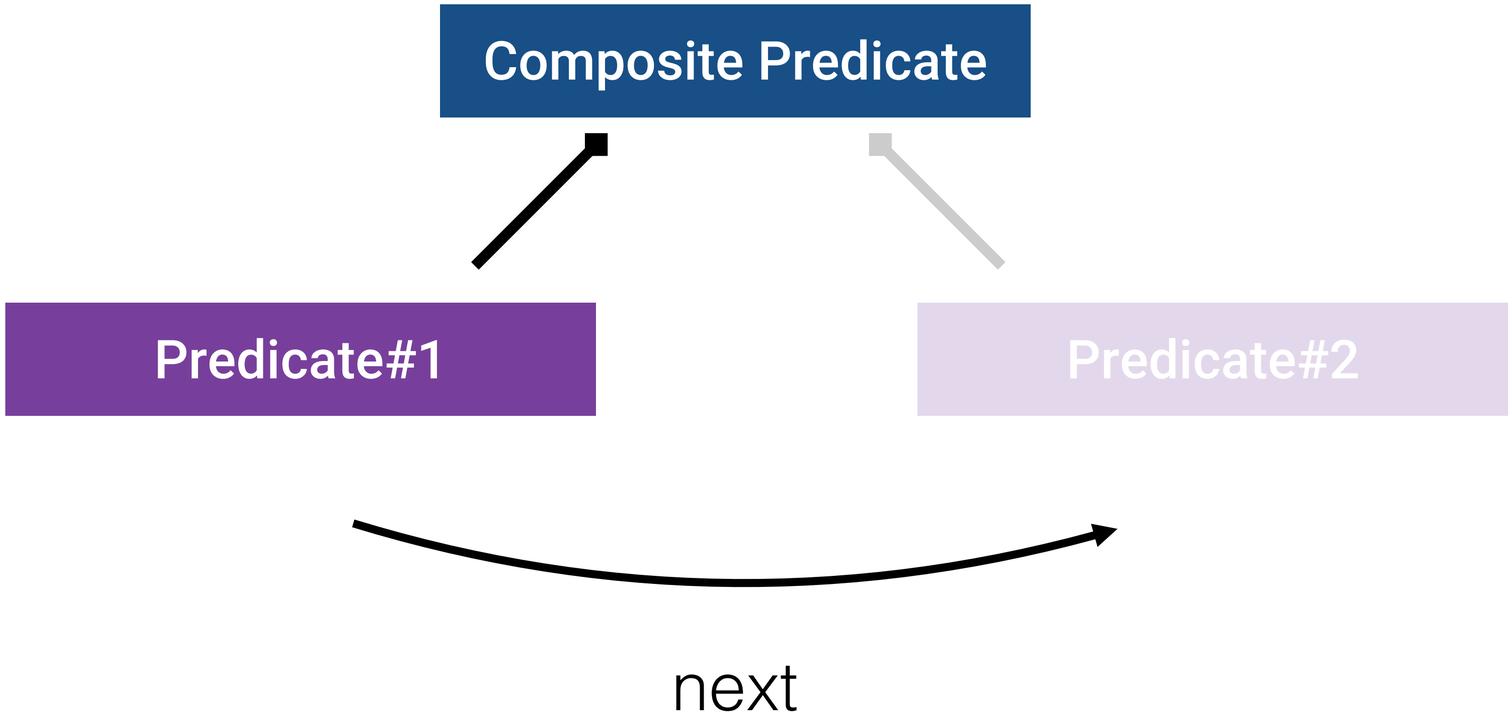

```
CompositePredicateBuilder
   int minimalFilteredServers = 1
   float minimalFilteredPercentage = 0


int count = predicate.filter(servers).size() //2
```




```
CompositePredicateBuilder
   int minimalFilteredServers = 1
   float minimalFilteredPercentage = 0

int count = predicate.filter(servers).size() //2

count >= minimalFilteredServers
count >= minimalFilteredPercentage * count
```




```
CompositePredicateBuilder
   int minimalFilteredServers = 1
   float minimalFilteredPercentage = 0

int count = predicate.filter(servers).size() //0

count >= minimalFilteredServers
count >= minimalFilteredPercentage * count
```

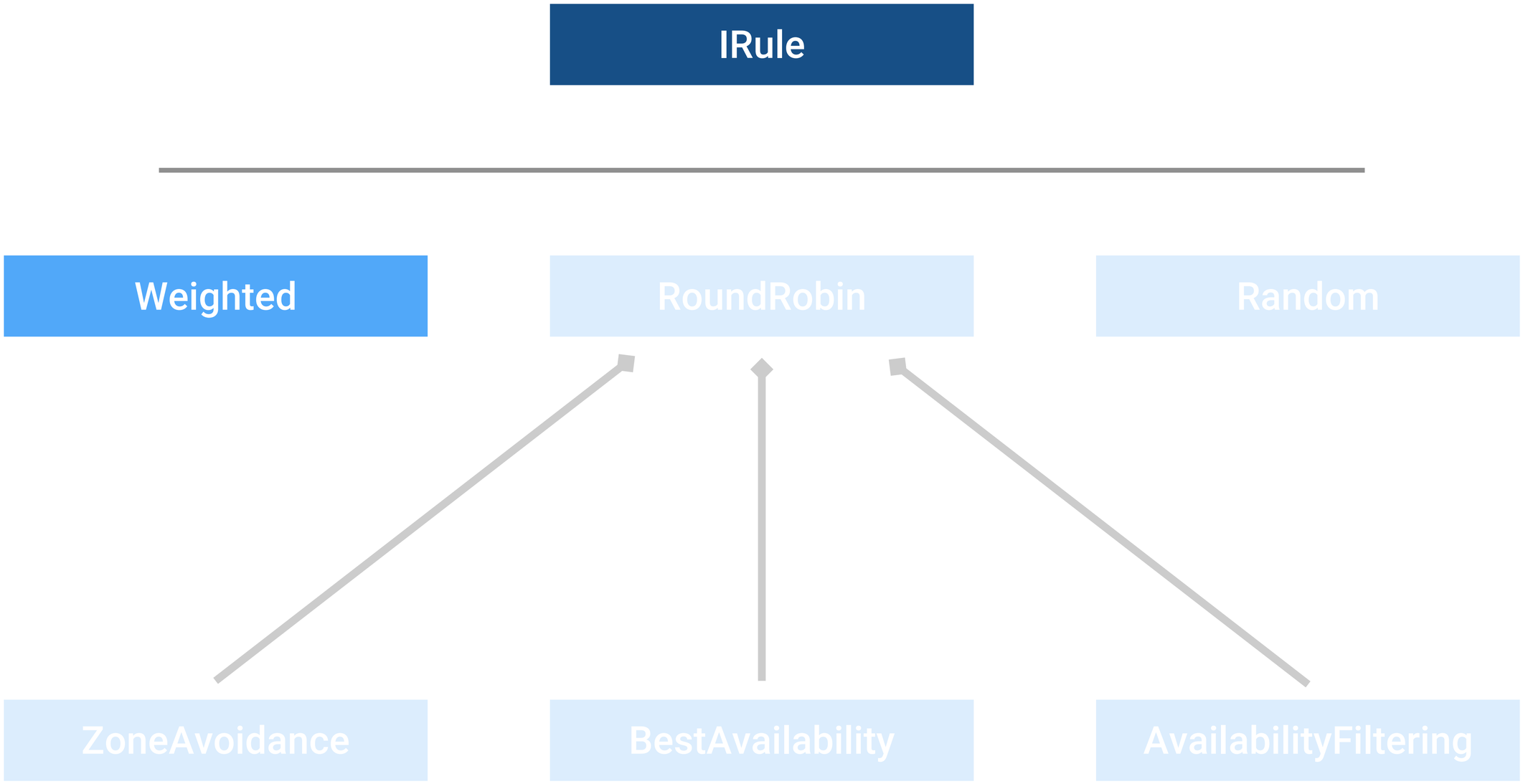

Composite Predicate
Predicate#1
Predicate#2
next

Вопрос#4

- Можно ли приспосабливаться на ходу?

Демо#4

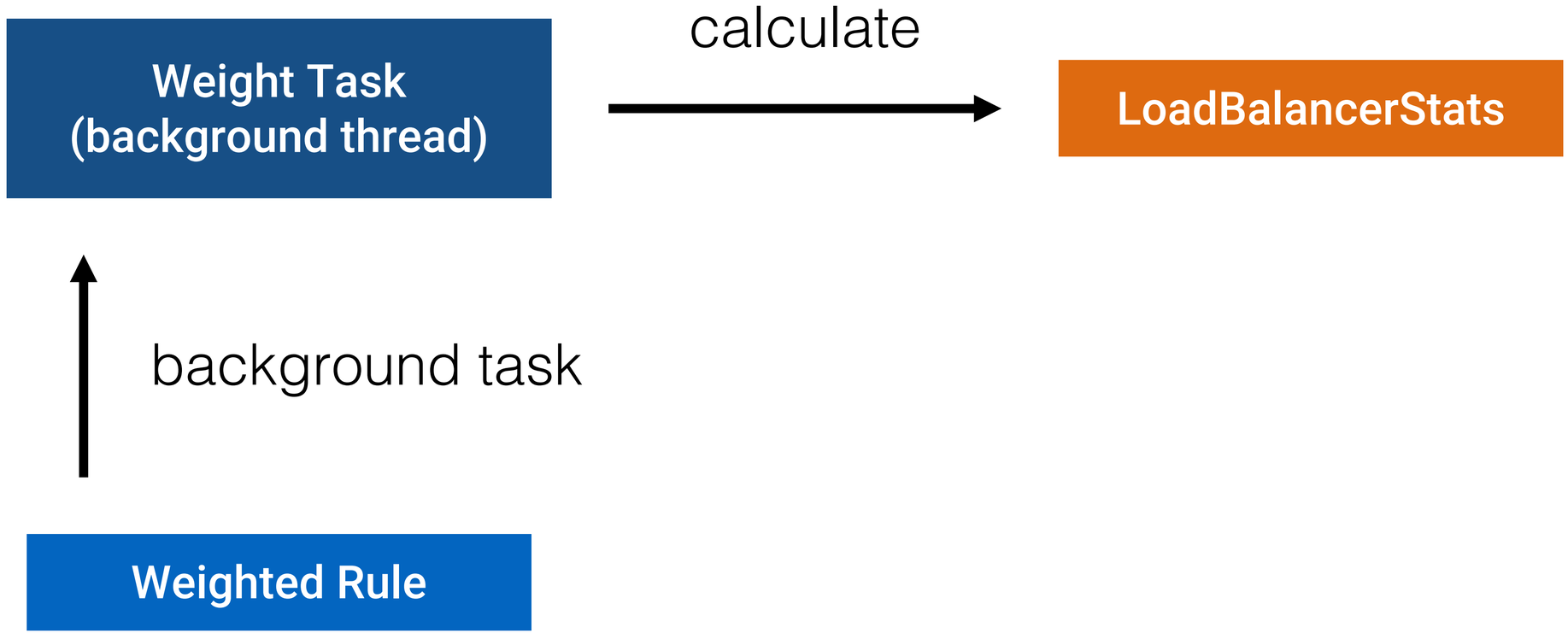
- Искусство приспосабливаться


IRule
Weighted
RoundRobin
Random
ZoneAvoidance
BestAvailability
AvailabilityFiltering

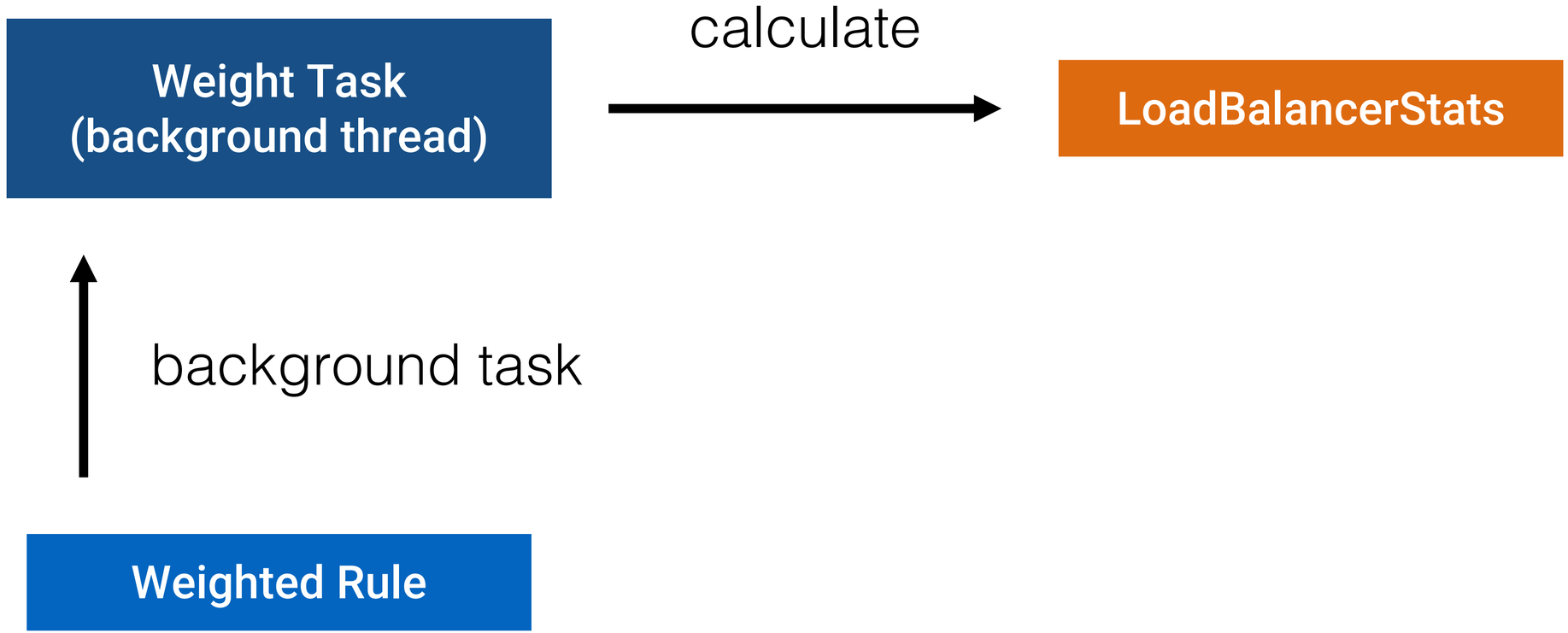
Типичный конфиг с весами

```
upstream cool-app {
    server host1 weight=3;
    server host2 weight=1;
    server host3 weight=1;
}
```

Как правило вес предопределен

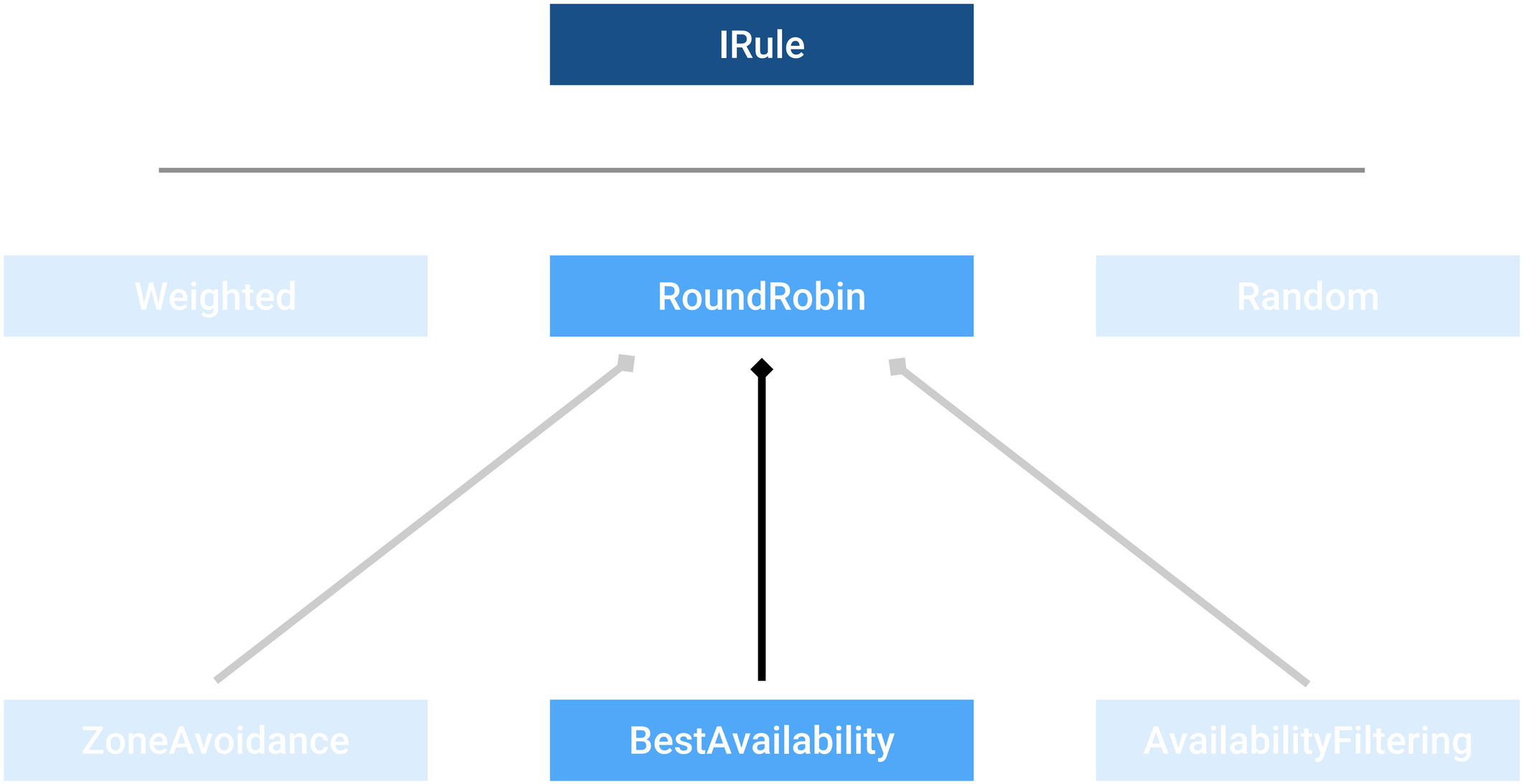


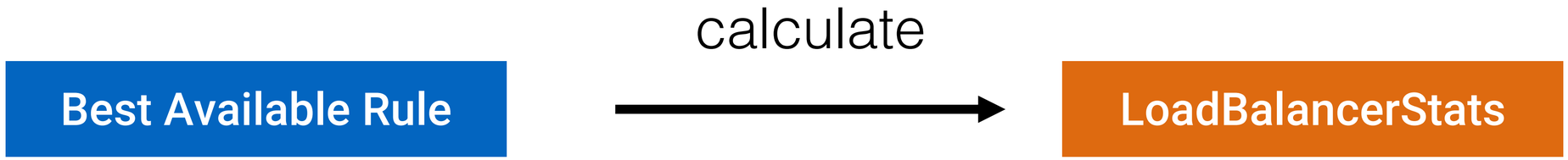

```
serverWeight = summ(avgServerResponseTime)- avgServerResponseTime
```



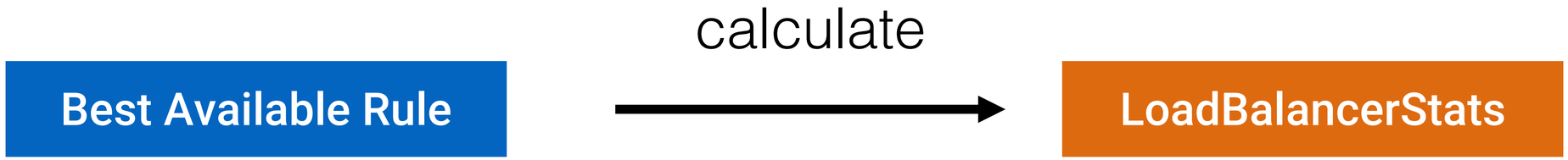

```
serverWeight = summ(avgServerResponseTime)- avgServerResponseTime
```

Больше время ответа – меньше вес


IRule
Weighted
RoundRobin
Random
ZoneAvoidance
BestAvailability
AvailabilityFiltering




```
chosen -> activeConnections == min(activeConnections)
```

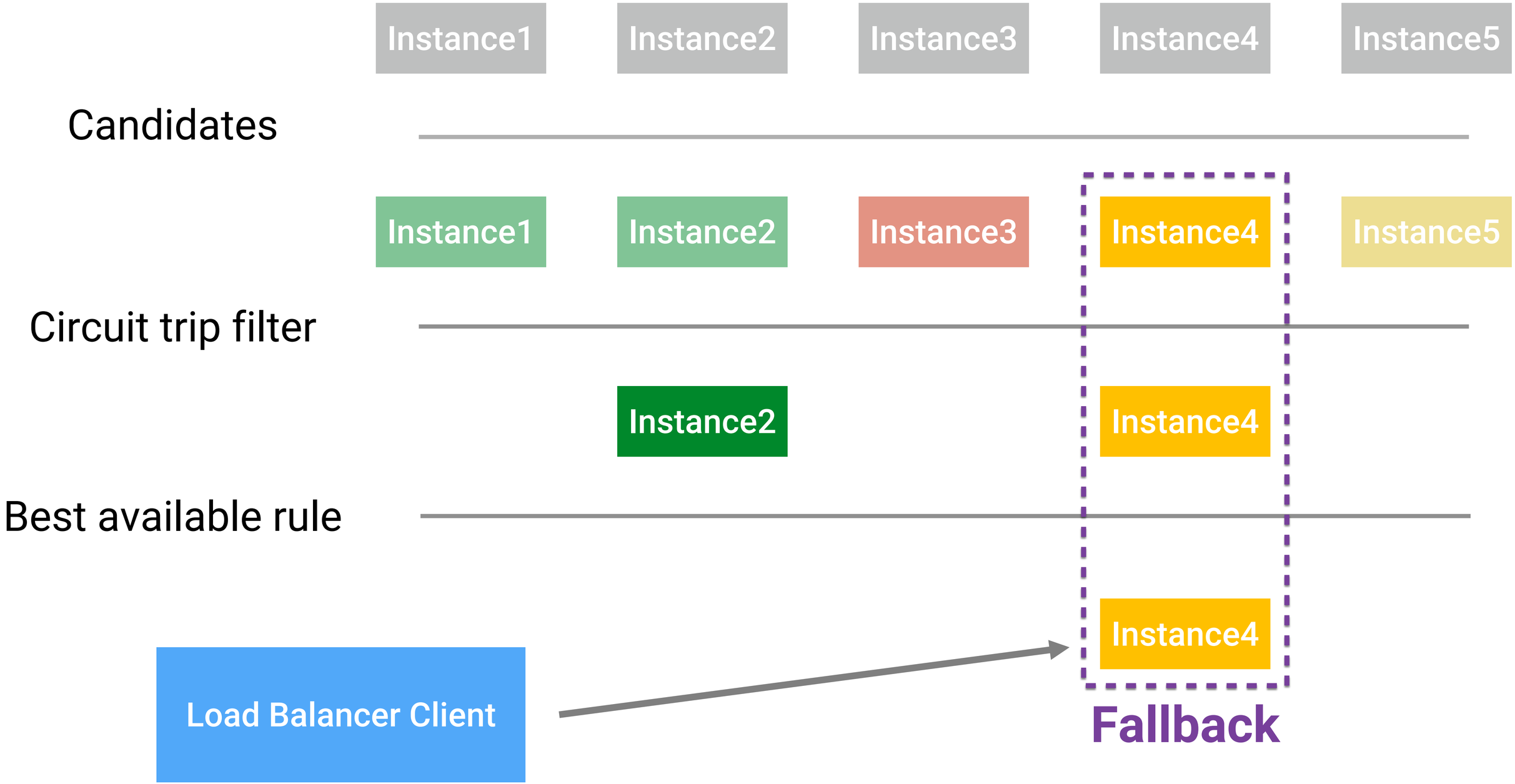



```
chosen -> activeConnections == min(activeConnections)
```

Меньше активных соединений -> больше шансов на успех

Graceful degradation

- Балансировка не поможет, если есть недостаток в ресурсах
- В какой-то момент нужно уметь деградировать с минимальным эффектом на пользователей


Instance1
Instance2
Instance3
Instance4
Instance5
Candidates
Instance1
Instance2
Instance3
Instance4
Instance5
Circuit trip filter
Instance2
Instance4
Best available rule
Instance4
Load Balancer Client
Fallback

Вопрос#5

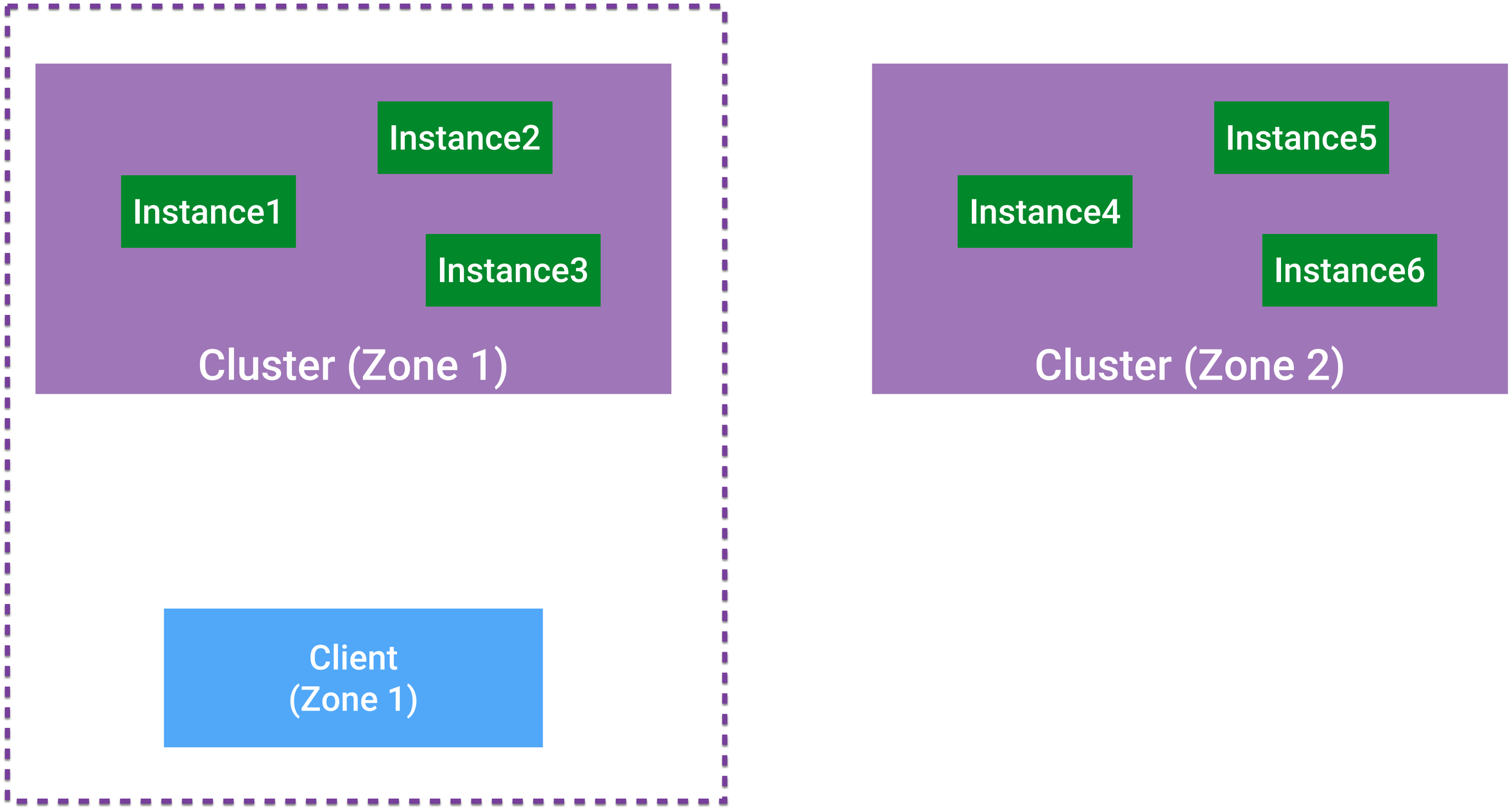
- Можем ли мы локализовать запросы в рамках одной зоны?

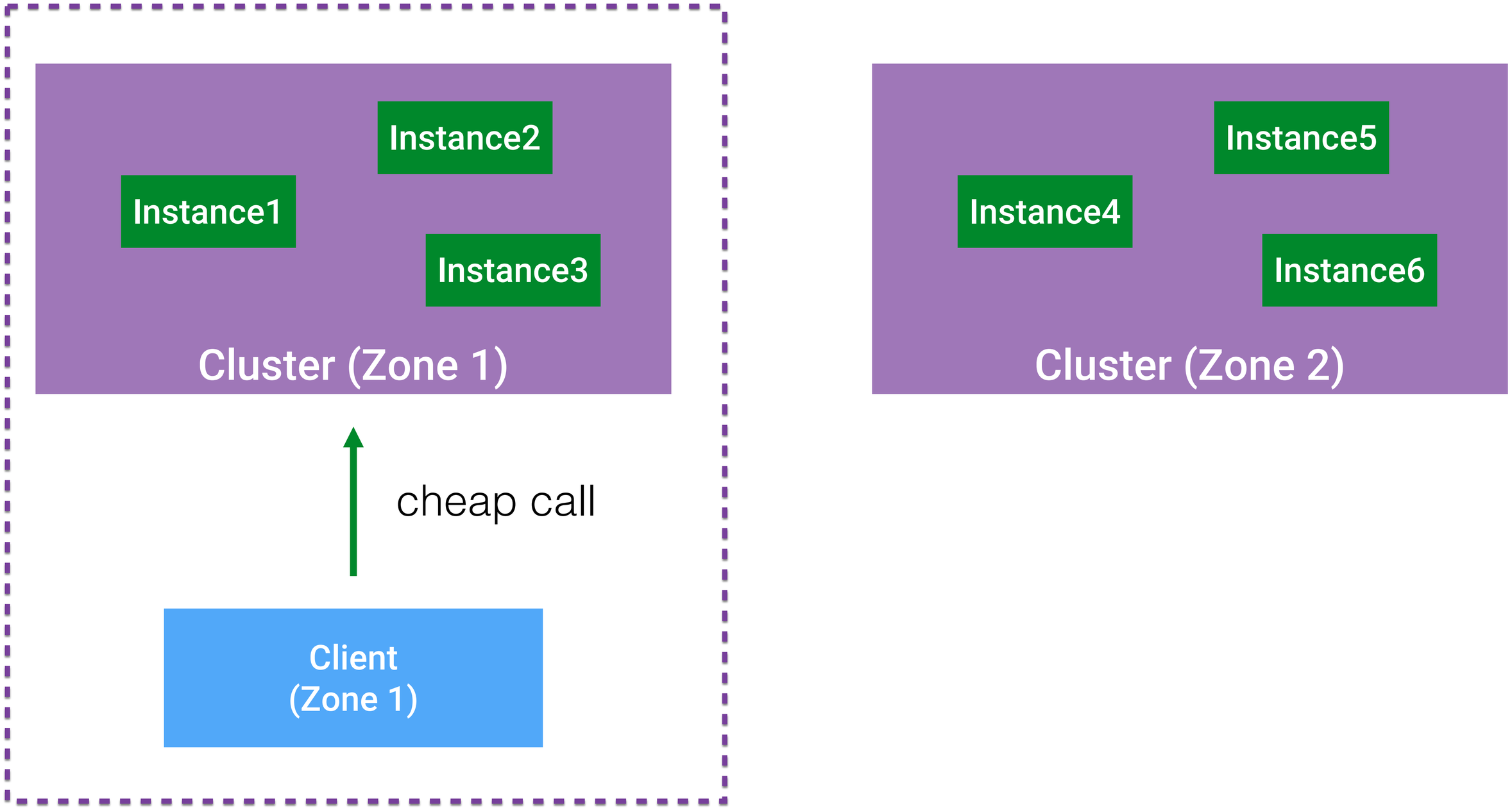
Демо#5

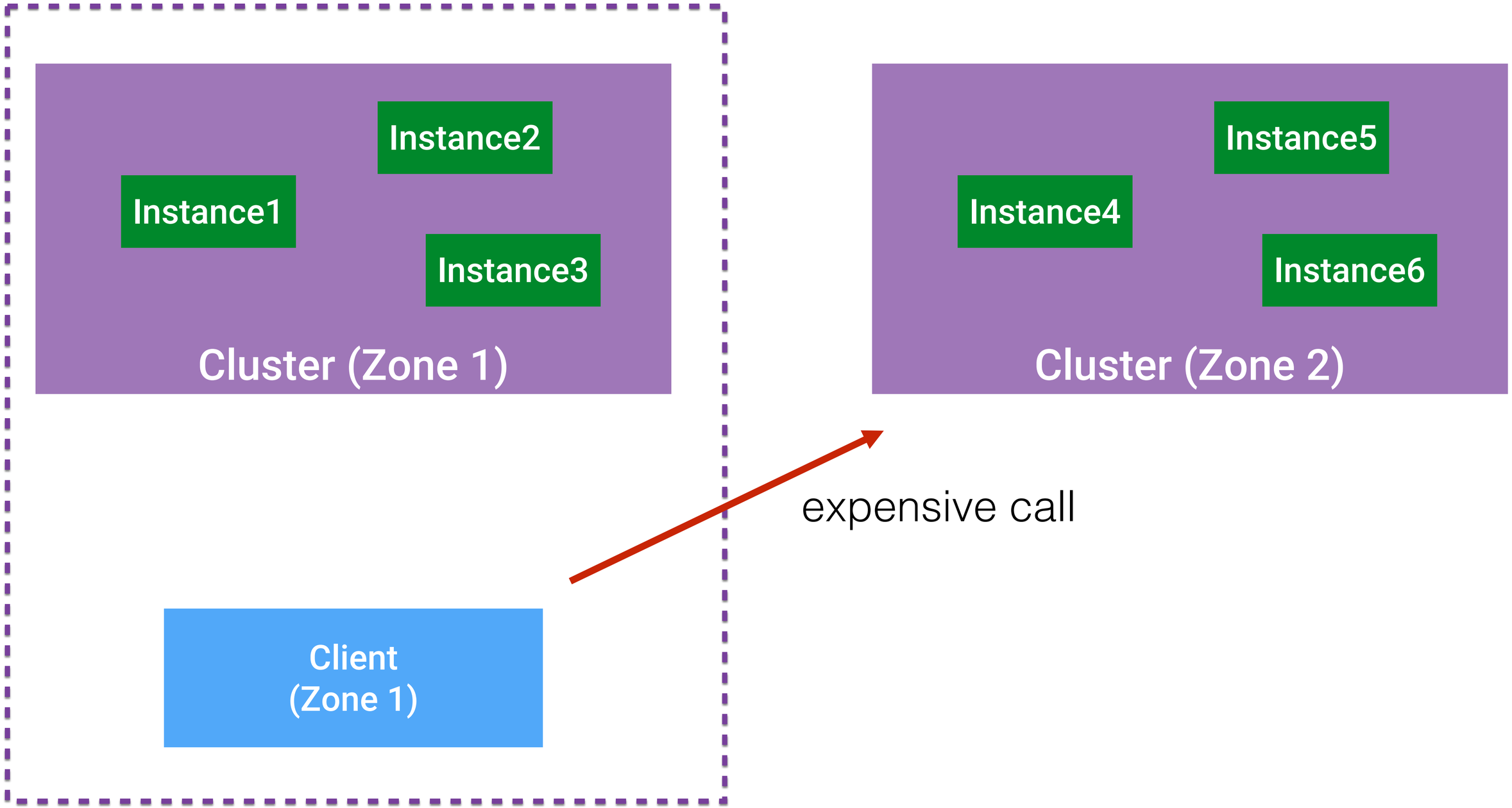
- Добро пожаловать на зоны

Availability Zones

- Изолированная локация
- Можете думать, что это аналог датацентра


Instance2
Instance1
Instance3
Cluster (Zone 1)
Instance5
Instance4
Instance6
Cluster (Zone 2)
Client
(Zone 1)


Instance2
Instance1
Instance3
Cluster (Zone 1)
Instance5
Instance4
Instance6
Cluster (Zone 2)
cheap call
Client
(Zone 1)


Instance2
Instance1
Instance3
Cluster (Zone 1)
Instance5
Instance4
Instance6
Cluster (Zone 2)
expensive call
Client
(Zone 1)

Zone Preference Filter

```
List servers = super.getFilteredServers();
List local = servers.filter(server ->
                    server.zone == zoneFromConfig
                );


if (!local.isEmpty()) {
    return local;
}
return servers;
```

default (local) zone

Load Balancer

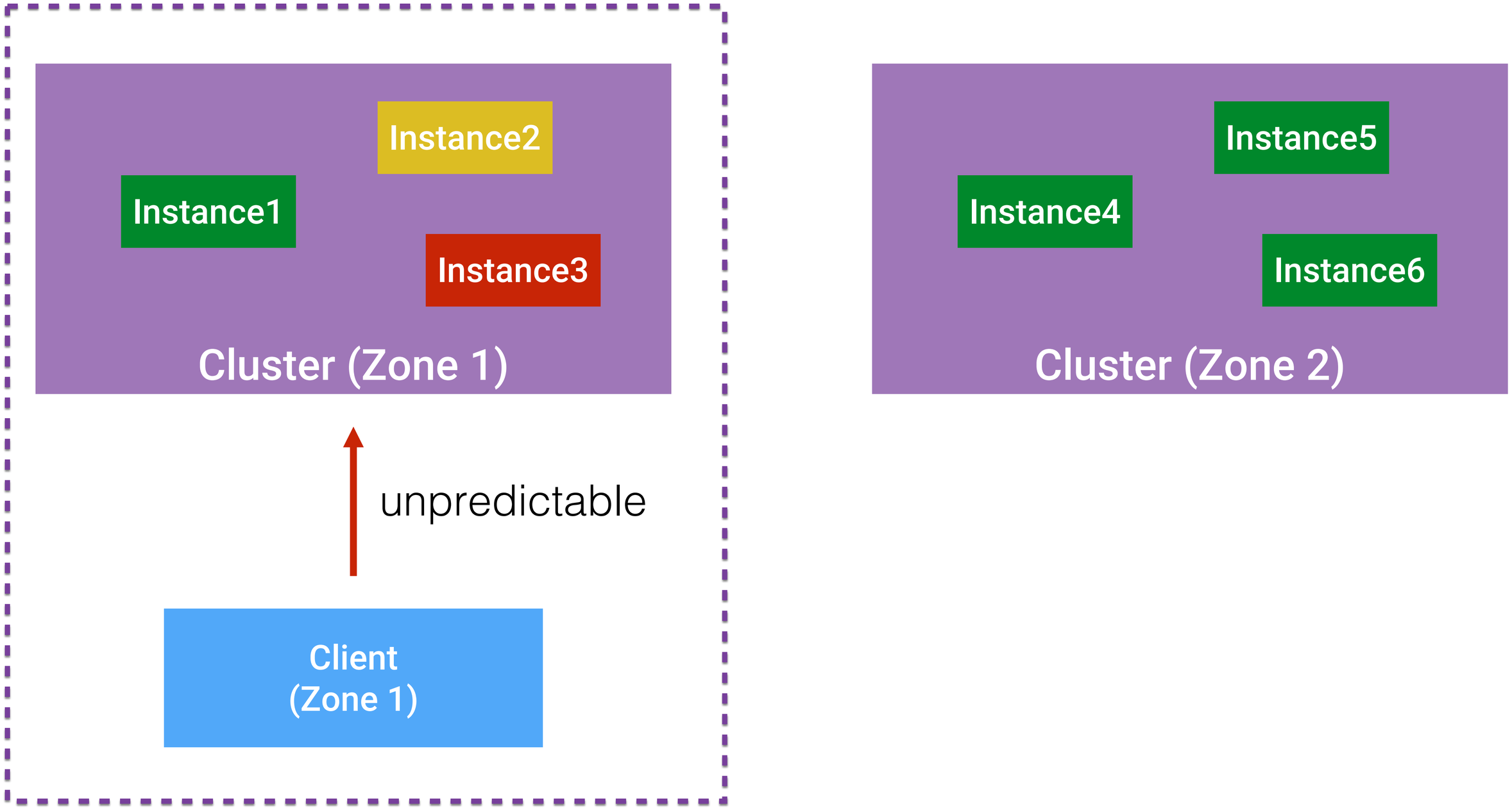
Zone Preference Filter

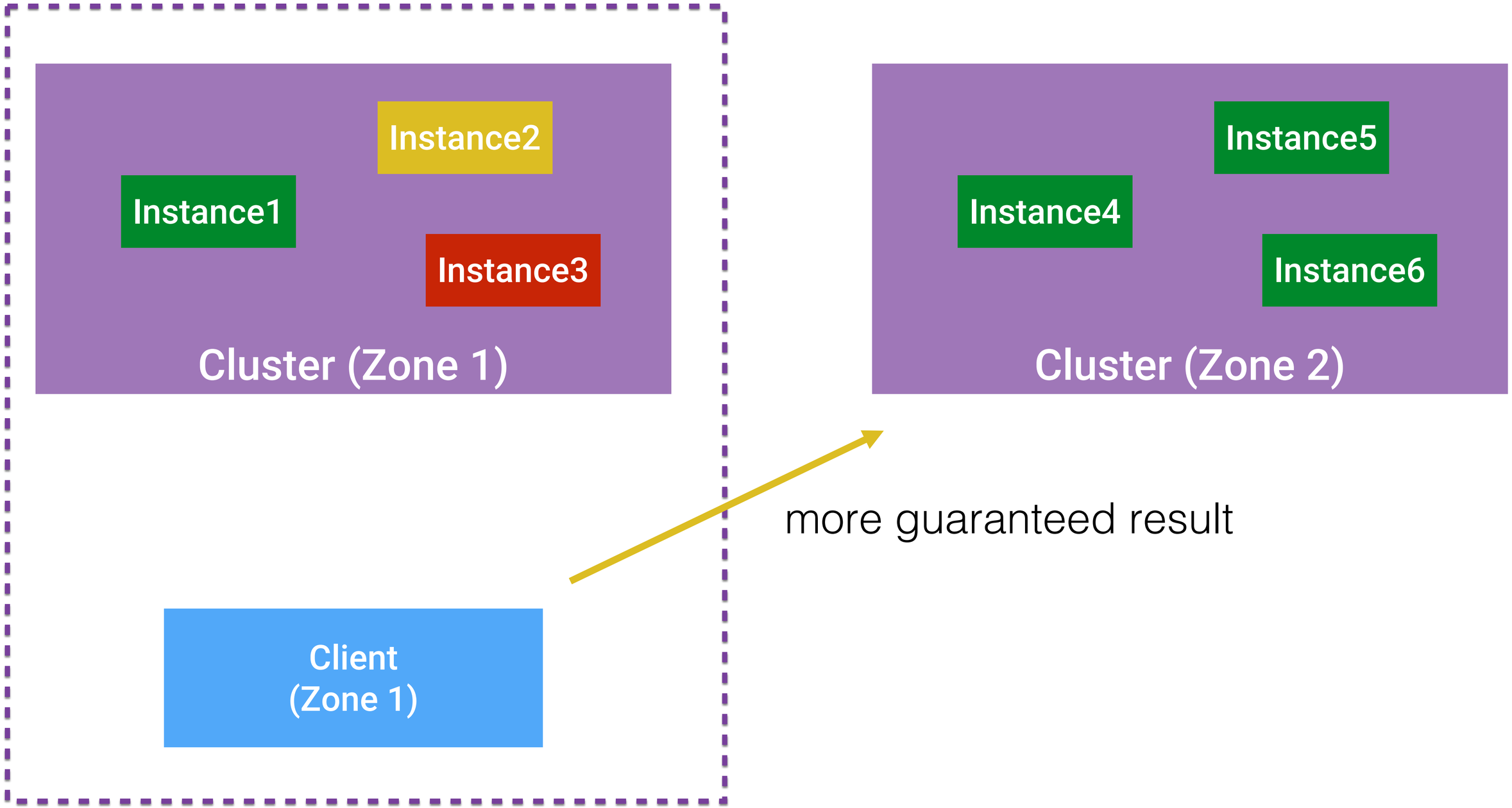
```
List servers = super.getFilteredServers();
List local = servers.filter(server ->
                    server.zone == zoneFromConfig
                );

if (!local.isEmpty()) {
    return local;
}
return servers;
```

Load Balancer

Вопрос#6

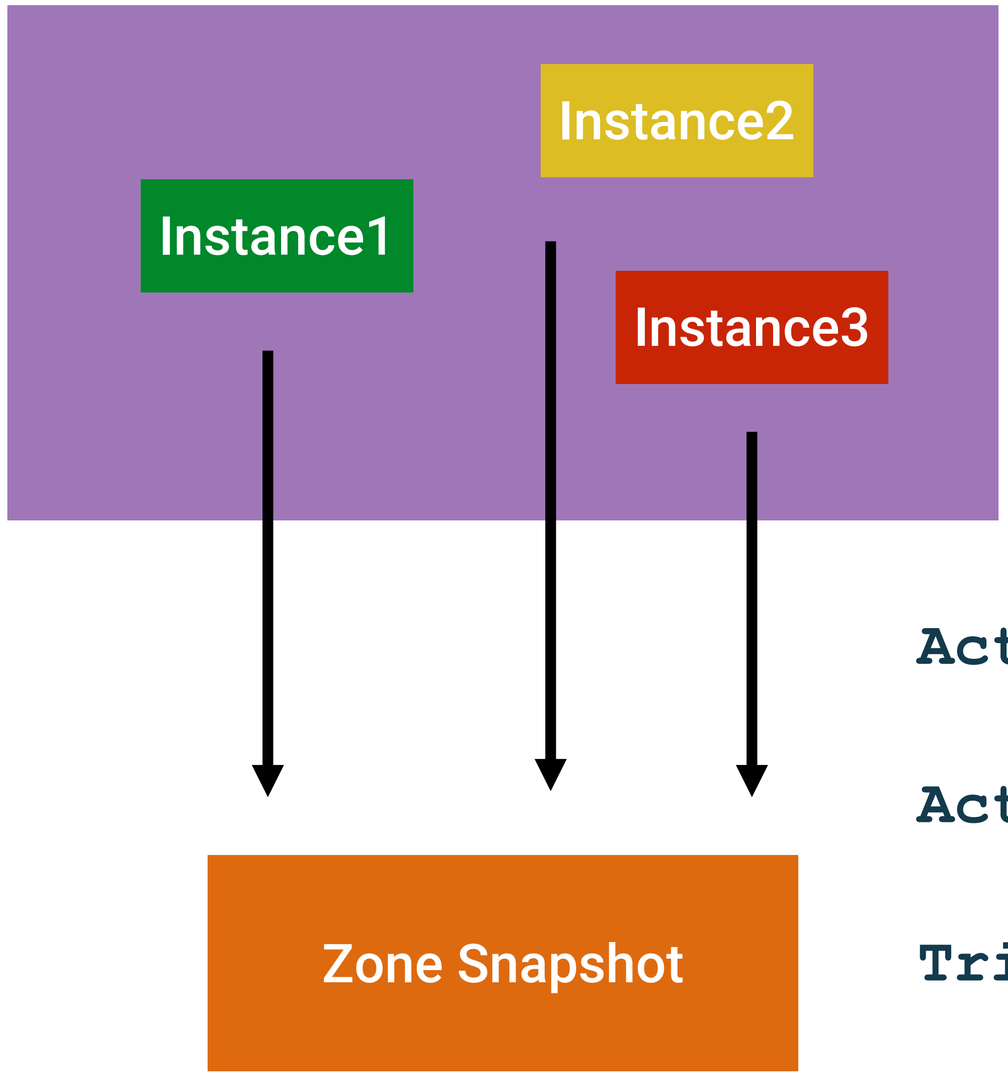
- А может не стоит локализовывать вызовы в одной зоне?


Instance2
Instance1
Instance3
Cluster (Zone 1)
Instance5
Instance4
Instance6
Cluster (Zone 2)
unpredictable
Client
(Zone 1)


Instance2
Instance1
Instance3
Cluster (Zone 1)
Instance5
Instance4
Instance6
Cluster (Zone 2)
more guaranteed result
Client
(Zone 1)

Демо#6

- Подстилая соломку

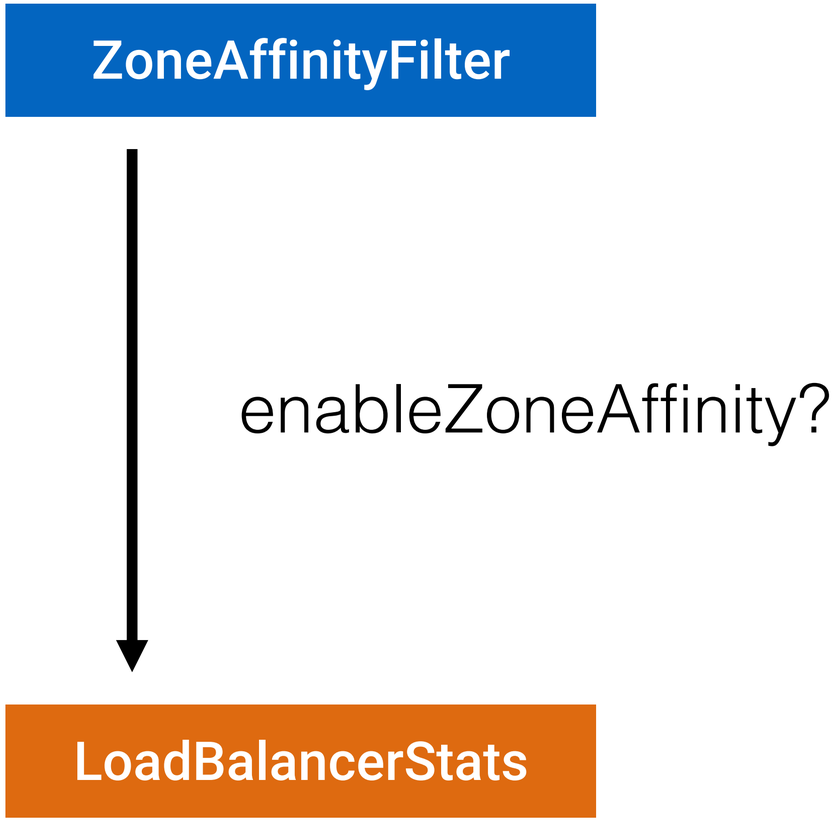


Active connections

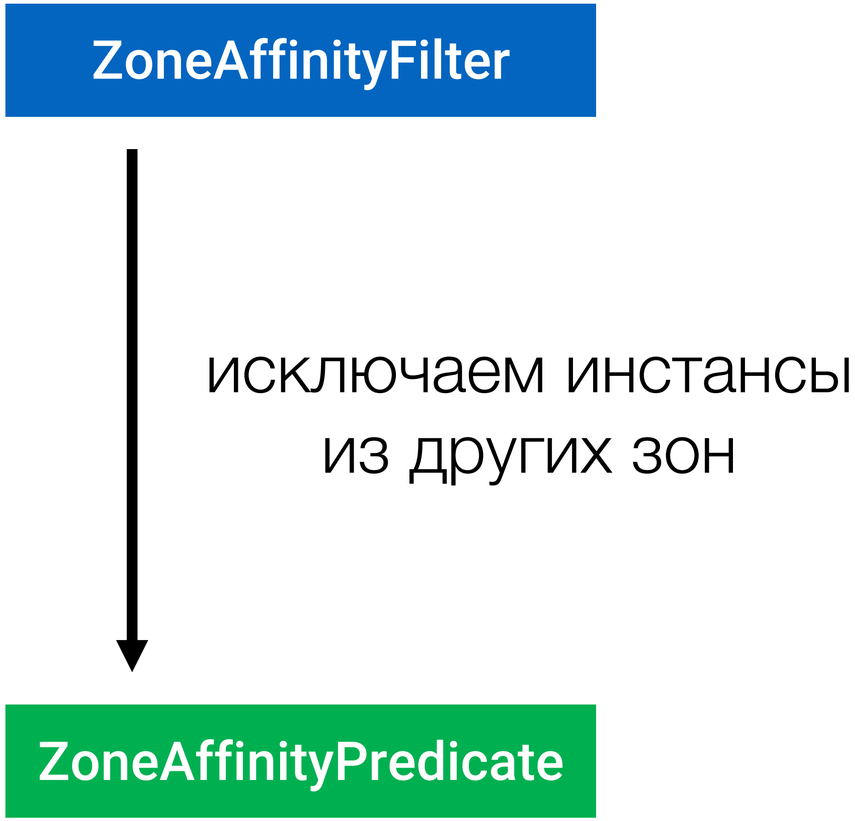
Active servers

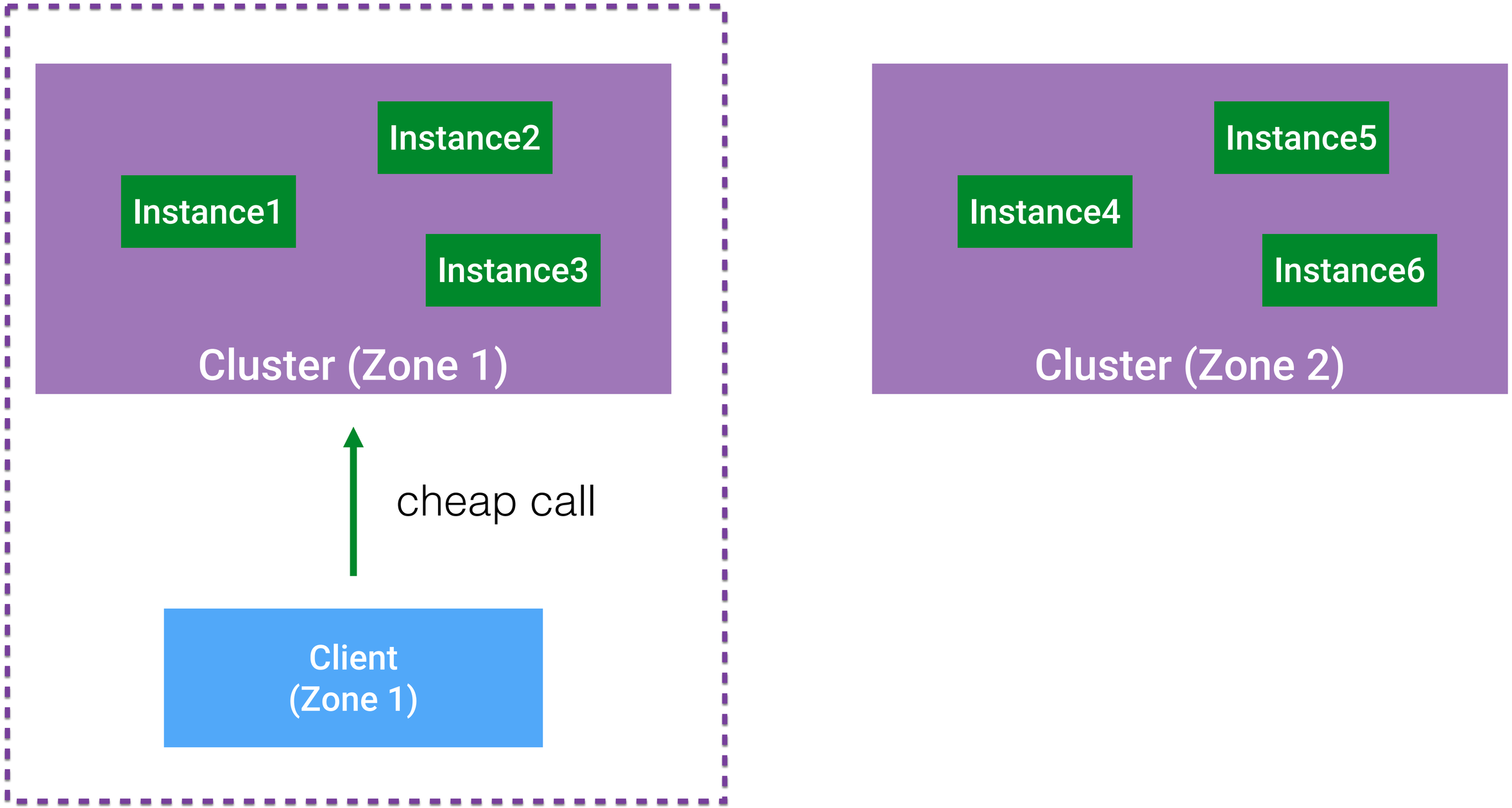
Tripped in Circuit Breaker

```
zoneAffinity:
  maxLoadPerServer: 0.6
  maxBlackOutServersPercentage: 0.8
  minAvailableServers: 2
```

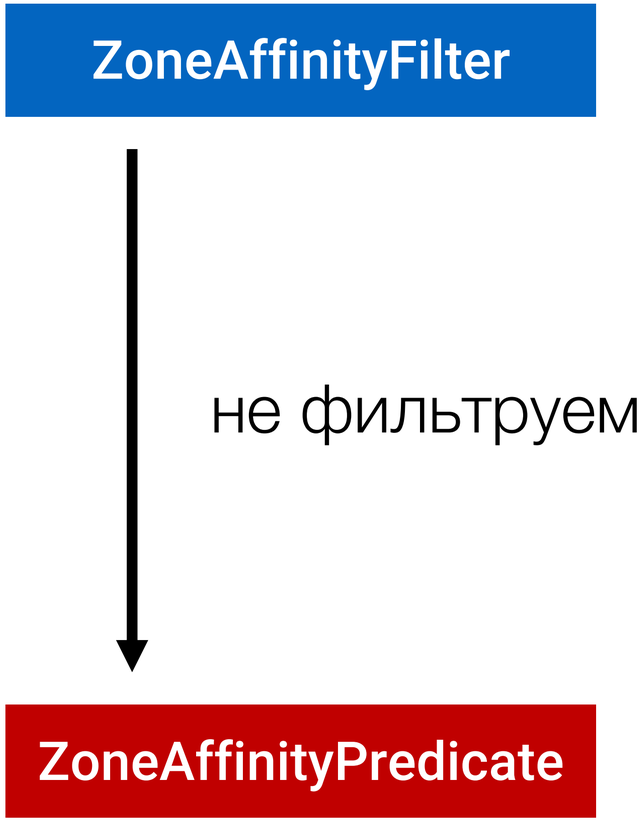


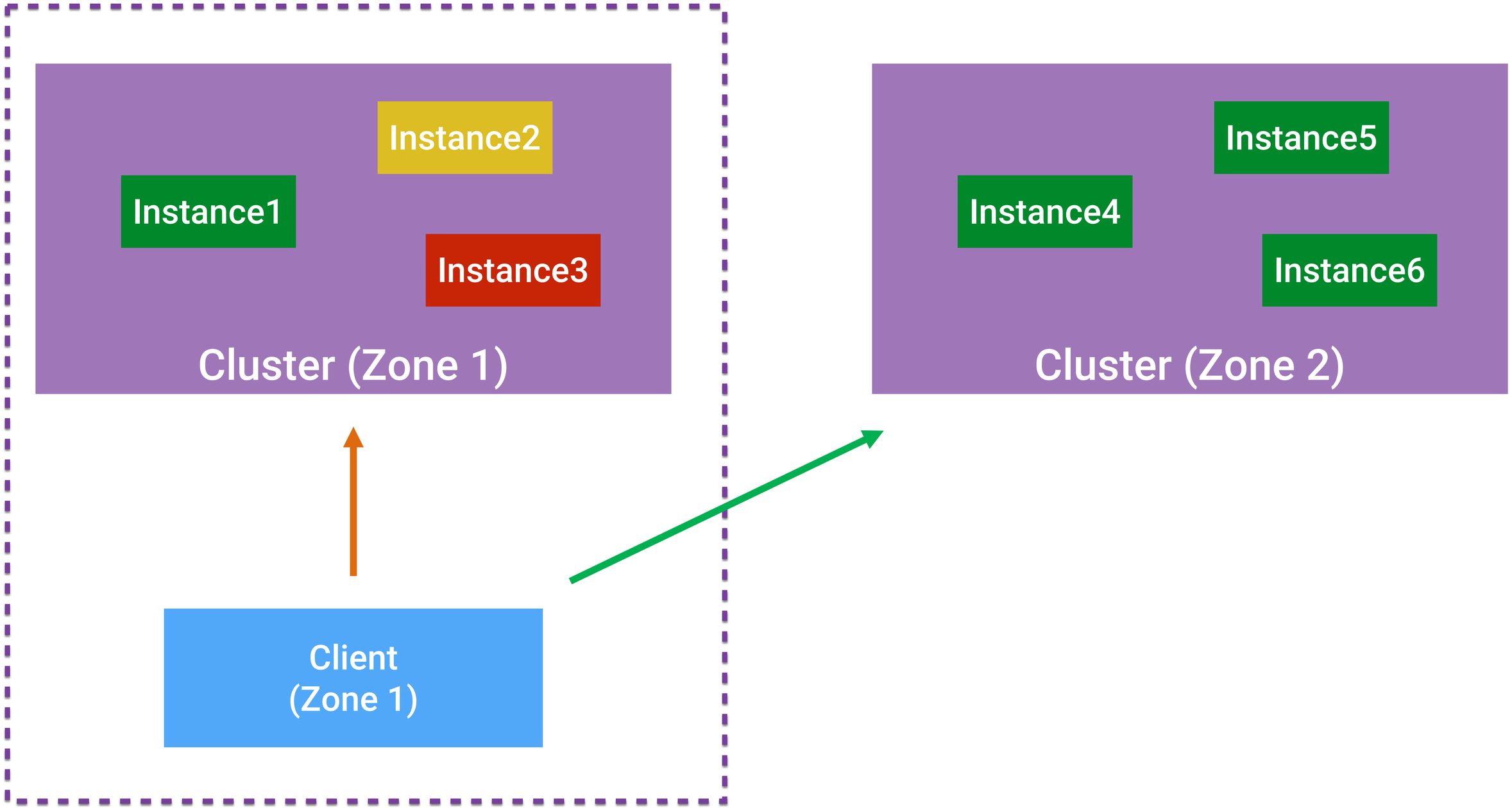
```
zoneAffinity:
  maxLoadPerServer: 0.6
  maxBlackOutServersPercentage: 0.8
  minAvailableServers: 2
```




Instance2
Instance1
Instance3
Cluster (Zone 1)
Instance5
Instance4
Instance6
Cluster (Zone 2)
cheap call
Client
(Zone 1)

```
zoneAffinity:
  maxLoadPerServer: 0.6
  maxBlackOutServersPercentage: 0.8
  minAvailableServers: 2
```




Instance2
Instance1
Instance3
Cluster (Zone 1)
Instance5
Instance4
Instance6
Cluster (Zone 2)
Client
(Zone 1)

Вопрос#7

- А можно ли лучше?

Демо#7

- Ковыряемся и хакаем

Выводы#1

- Основная цель – это предсказуемая и стабильная работа всей системы в целом
- Серебрянной пули нет. Ваш ворклоад другой и конфигурация тоже должна быть другой

Выводы#2

- От нас многое прячут в недрах конфигурации, и если покопаться, то можно найти интересные возможности, которые следует использовать с умом
- Изучение базовых решений (Netflix Ribbon. и т.д.) даст возможность эффективнее решать задачи

Выводы#3

- Балансировка не поможет, если есть недостаток в ресурсах
- В какой-то момент нужно уметь деградировать с минимальным эффектом на пользователей
- Используйте Hystrix
- Включайте голову

Ссылки на библиотеки

В главной роли

https://github.com/Netflix/ribbon

Библотеки, участвовавшие в демо

https://github.com/spring-cloud/spring-cloud-netflix

https://github.com/aatarasoff/spring-cloud-marathon

https://github.com/aatarasoff/spring-one-nio

Напосмотреть

Service Discovery: больше, чем кажется

https://www.youtube.com/watch?v=6uVgR9WPjYM

Попробовать Demo

https://github.com/aatarasoff/spring-cloud-load-balancing-sandbox

QA

@aatarasoff

@aatarasoff

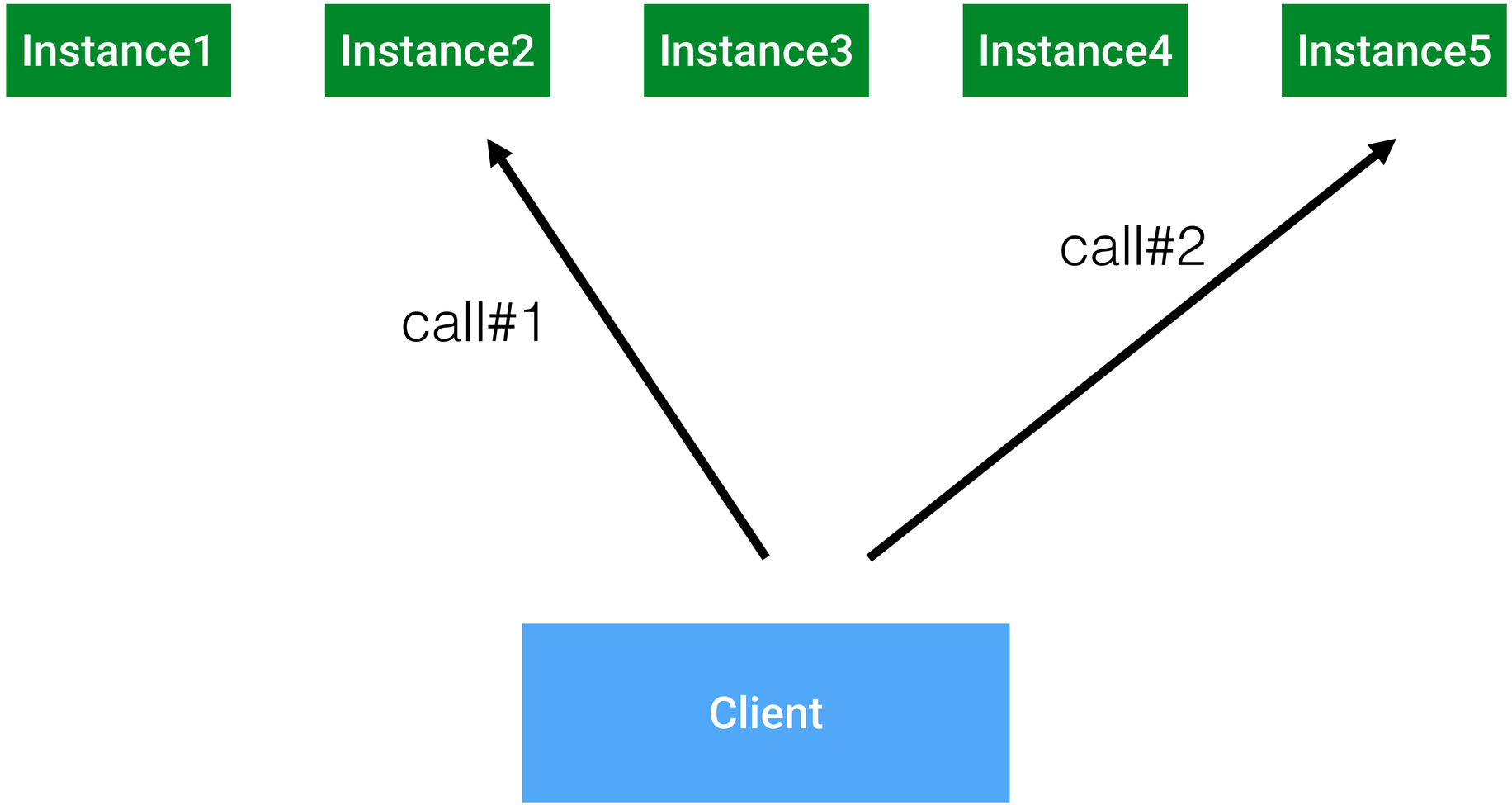
@aatarasoff

</> DEVELOPERBLOG.INFO

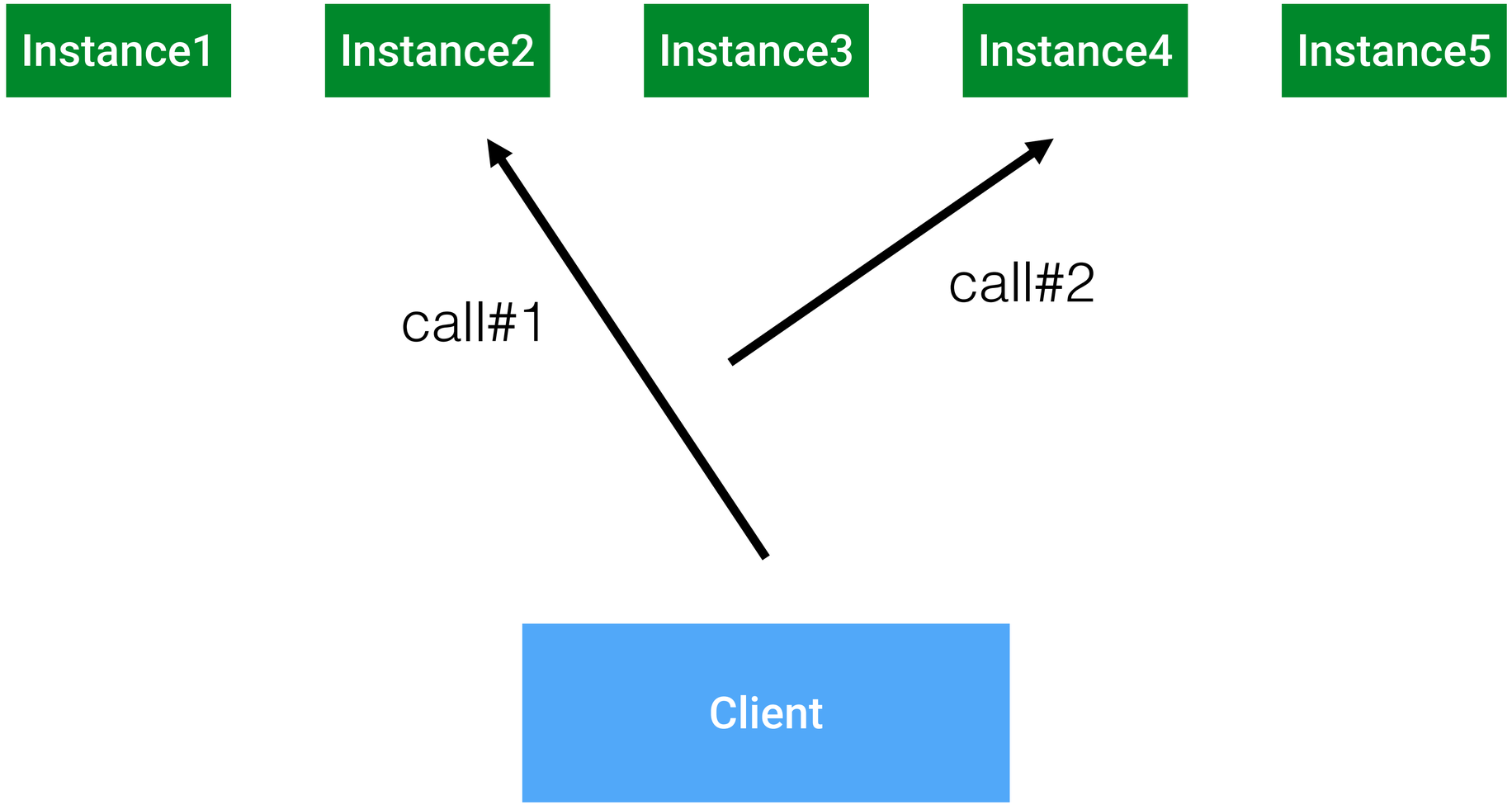
Speculative Retry

- Альтернативный вариант

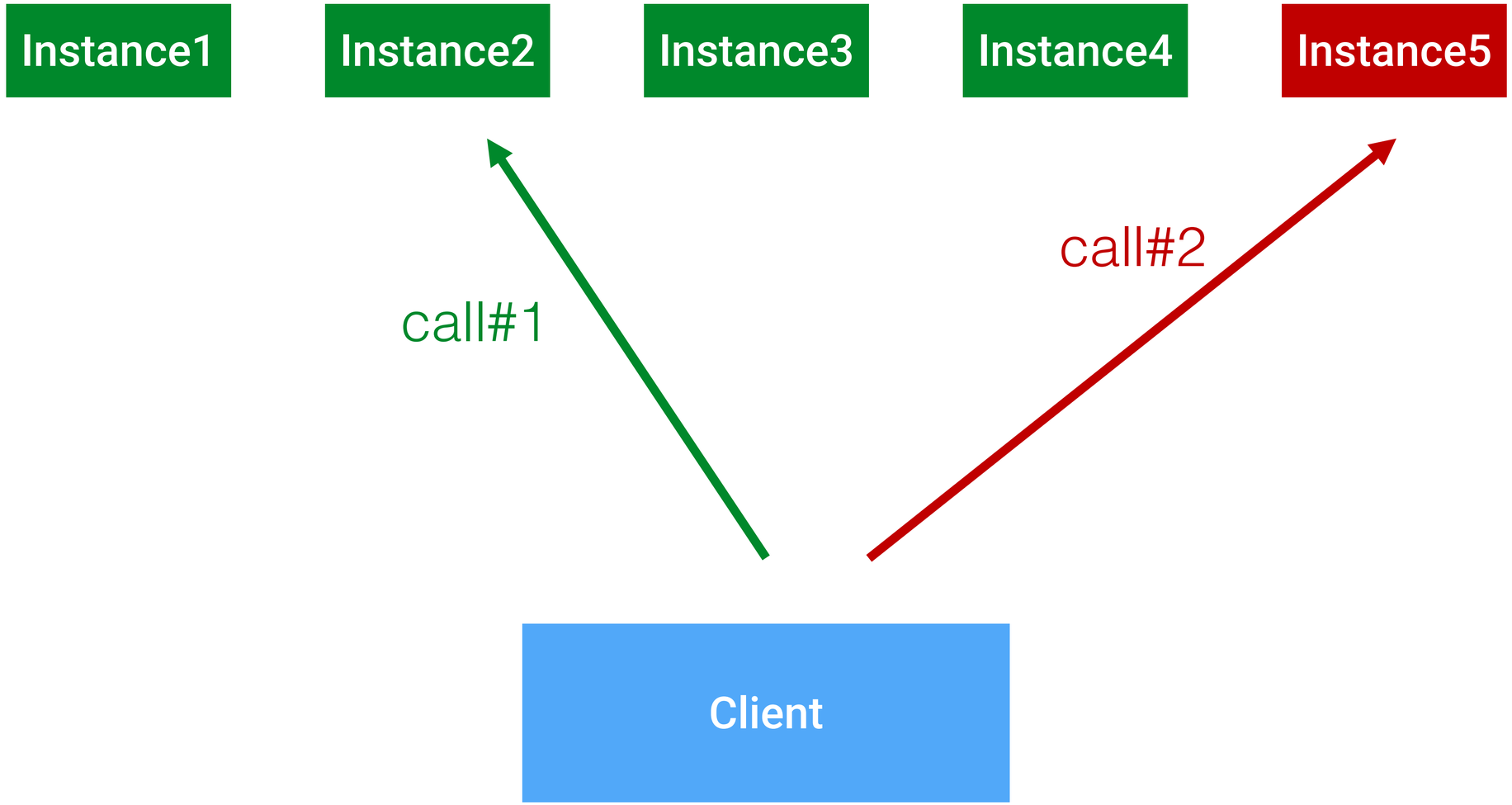
Speculative retry



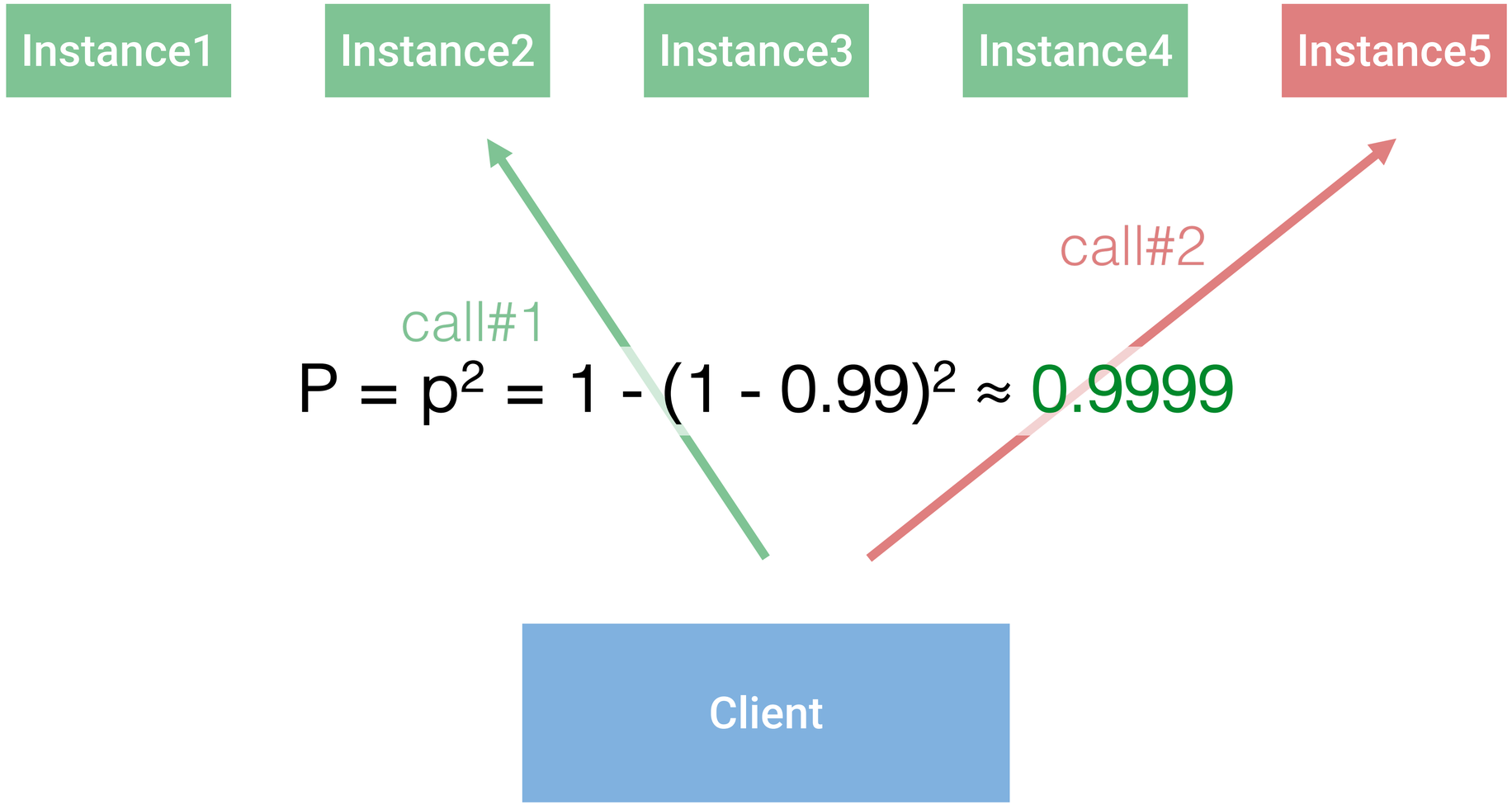
Split Strategy



Merge strategy



Speculative retry



Speculative Retry

- Надёжен в эксплуатации
- Все операции должны быть идемпотентными
- Количество вызовов x2

Интерфейсы Ribbon-a

```
//ILoadBalancer
public Server chooseServer(Object key);

//IRule
public Server choose(Object key);
```

Speculative и Ribbon

- Все абстракции рассчитаны на работу с одним инстансом
- Теоретически имплементация возможна, но надо лезть в кишки