alfalab

Service Discovery

Spring Cloud Internals

About me

Architect @ alfalab

/aatarasoff

/aatarasoff

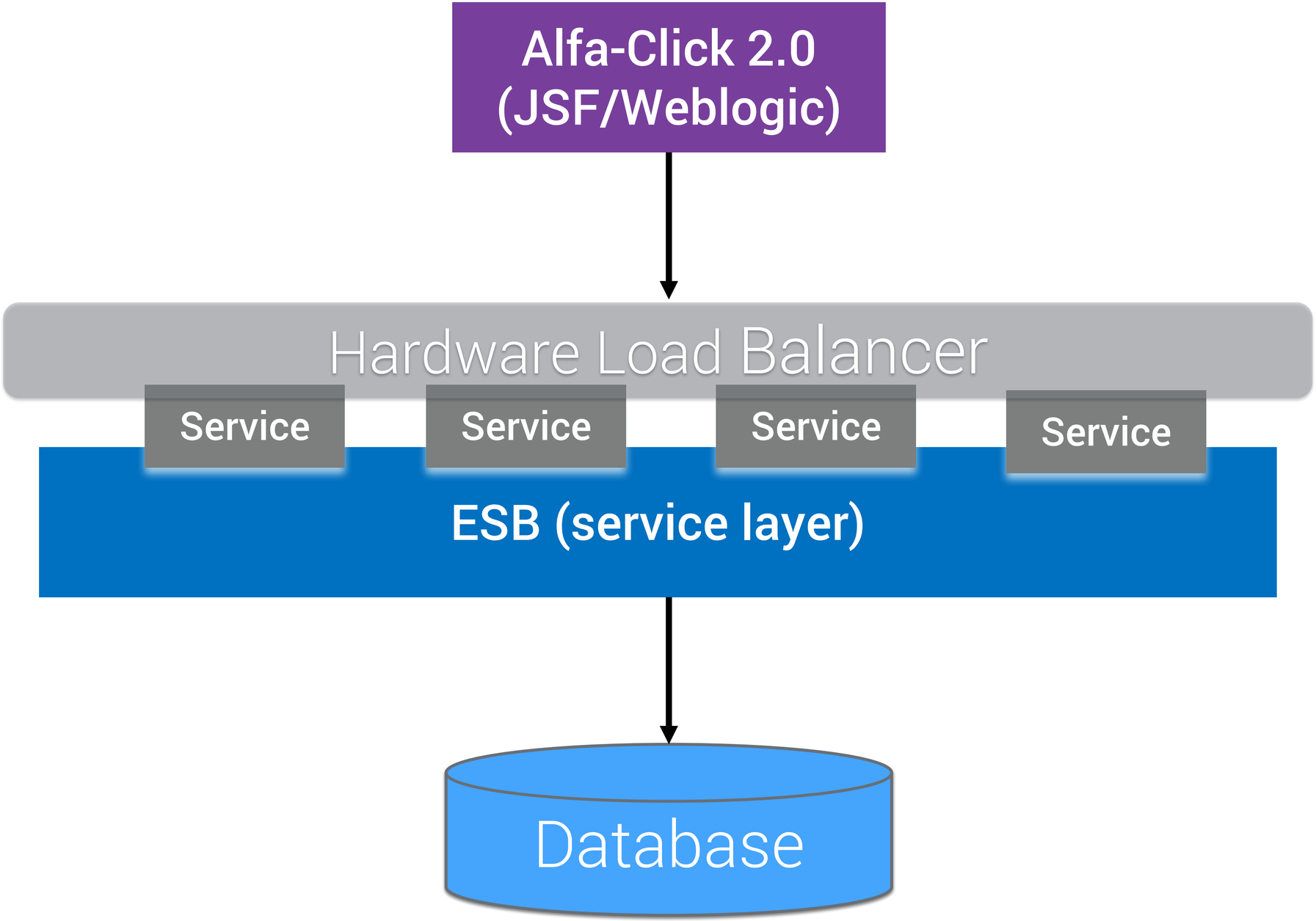
habrahabr.ru/aatarasoff

developerblog.info

Agenda
Today

1. New reality - new problems
2. What service discovery means?
3. Dive into Spring Cloud service discovery pattern implementation
4. Tips and tricks during the session
5. Own experience as a conclusion


What is the problem?


Alfa-Click 2.0
(JSF/Weblogic)
Hardware Load Balancer
Service
Service
Service
Service
ESB (service layer)
Database


Big App
Static resource
Database
Remote EJB

Simple Binding

Small number of resources

Static host and port binding

Rare reconfiguration

```
insert into resources_table values (key, endpoint)
```

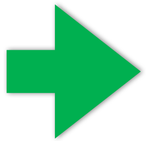
```
final Properties props = new Properties();
final Context context = new InitialContext(props);

// lookup
Foo bean = context.lookup("ejb:myejb//Foo!org.myapp.Foo");
bean.call();
```

```
final Properties props = new Properties();
final Context context = new InitialContext(props);

// lookup
Foo bean = context.lookup("ejb:myejb//Foo!org.myapp.Foo");

//do something
bean.call();
```

Big Frontend App

alfalab

Big Frontend App

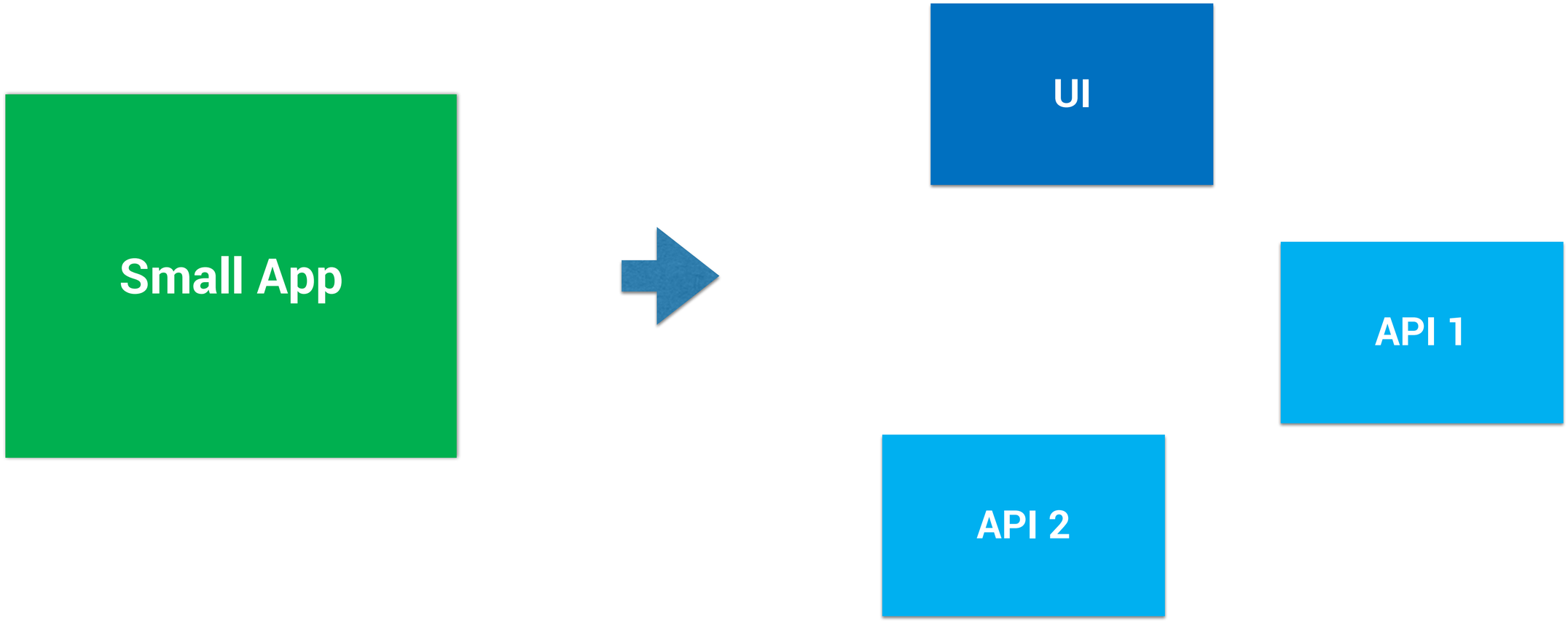


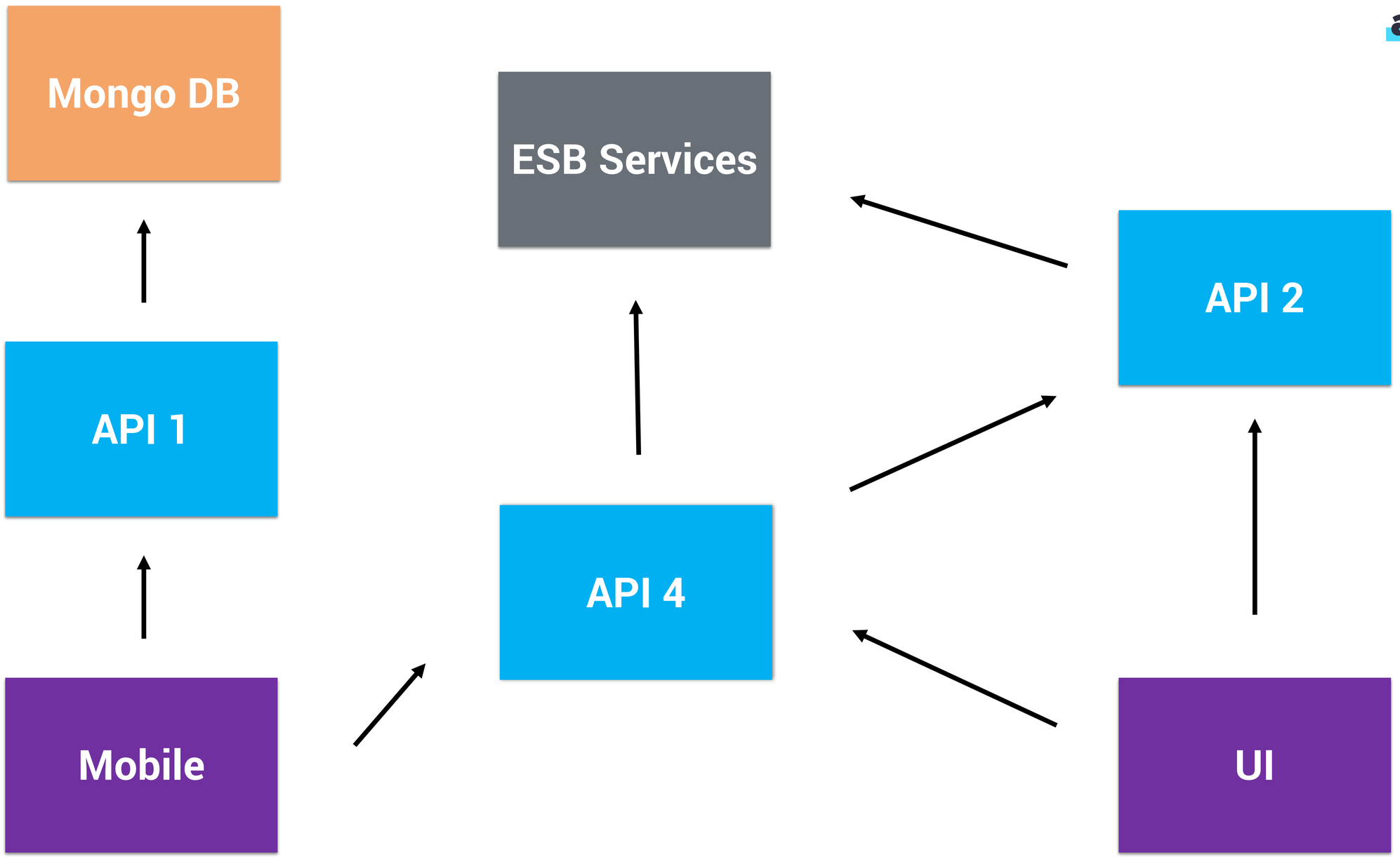
Small App

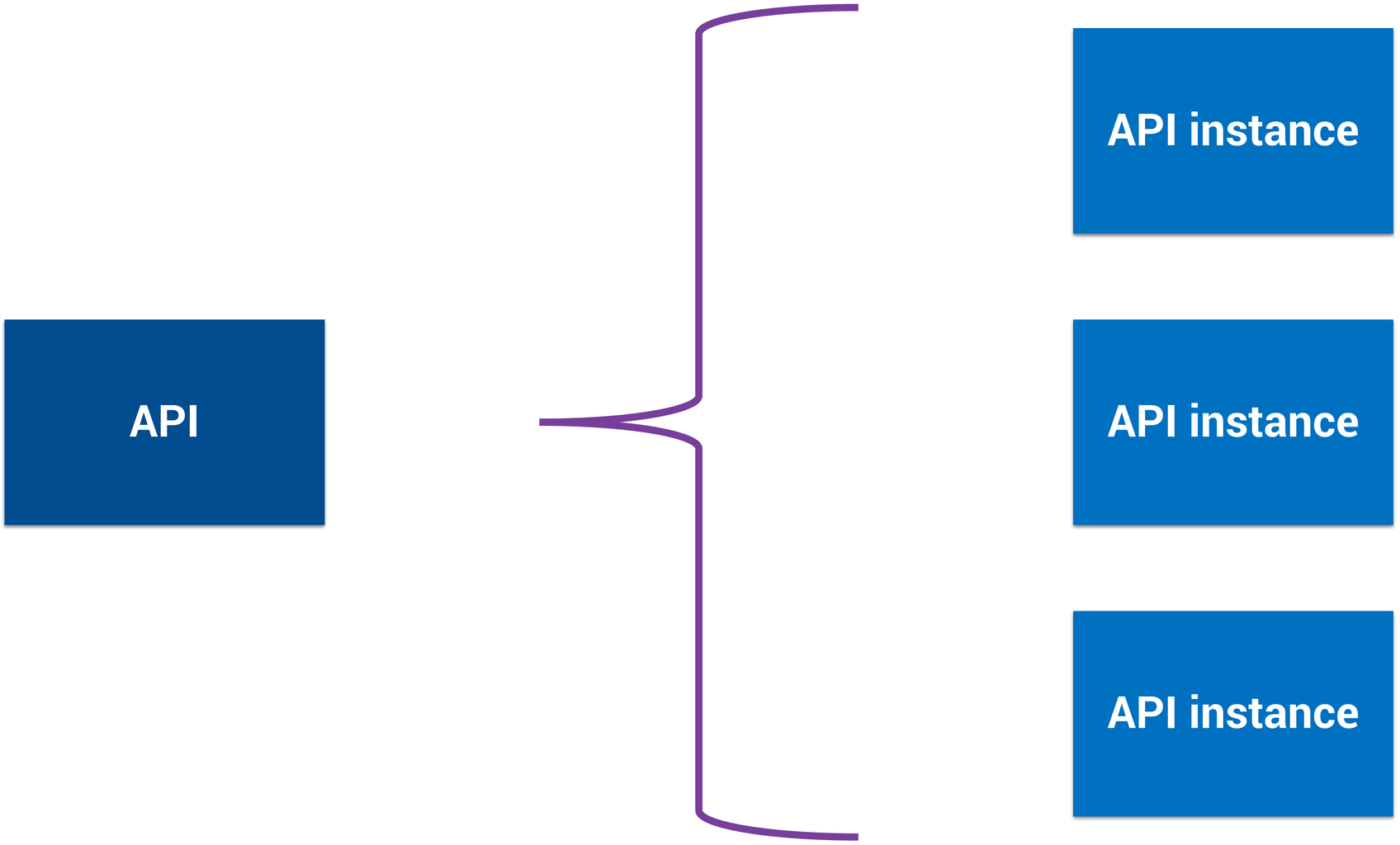
Small App

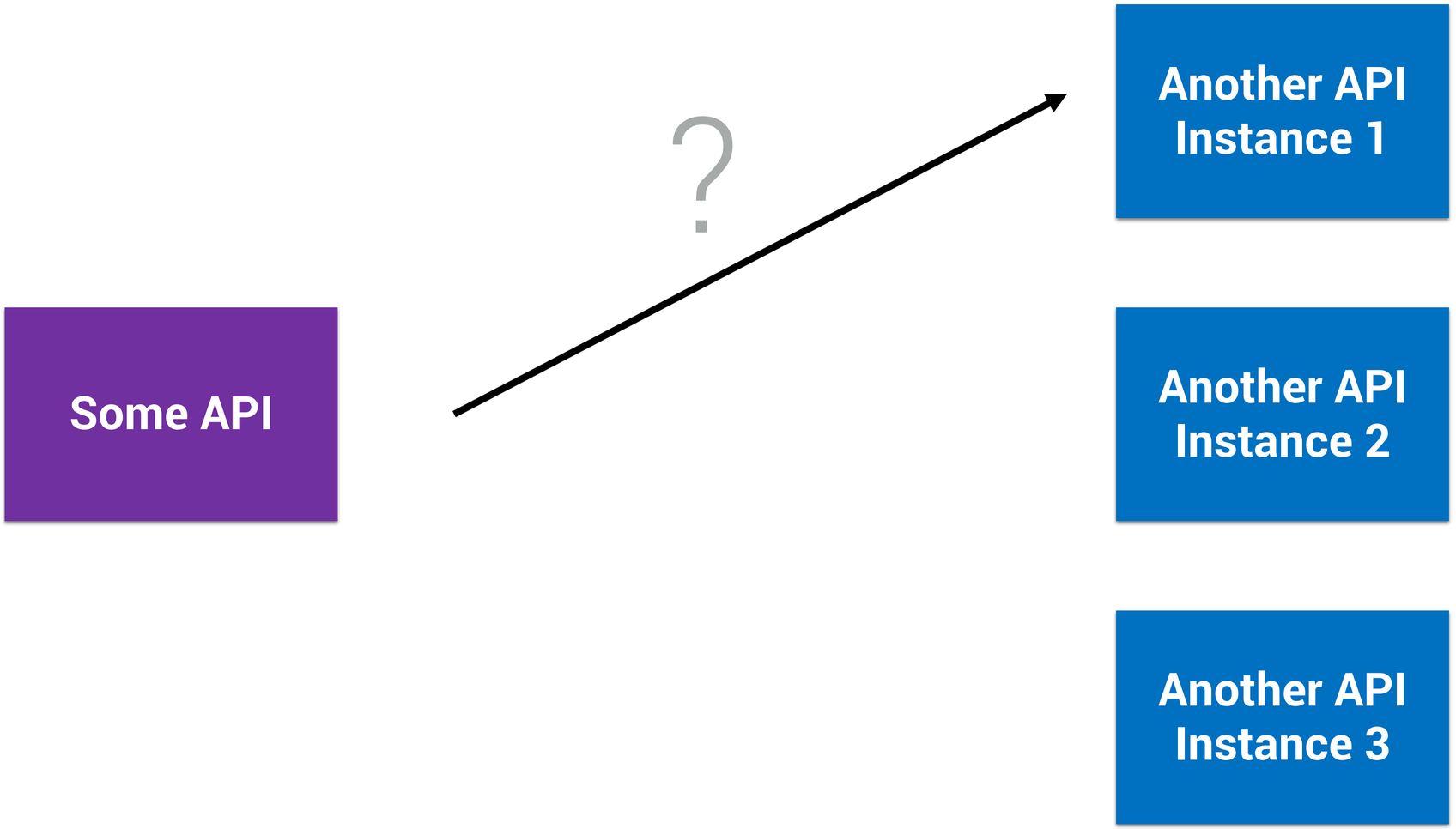
Small App

alfalab




Mongo DB
ESB Services
API 2
API 1
API 4
Mobile
UI


API
API instance
API instance
API instance


Some API
?
Another API Instance 1
Another API Instance 2
Another API Instance 3

Static Configuration

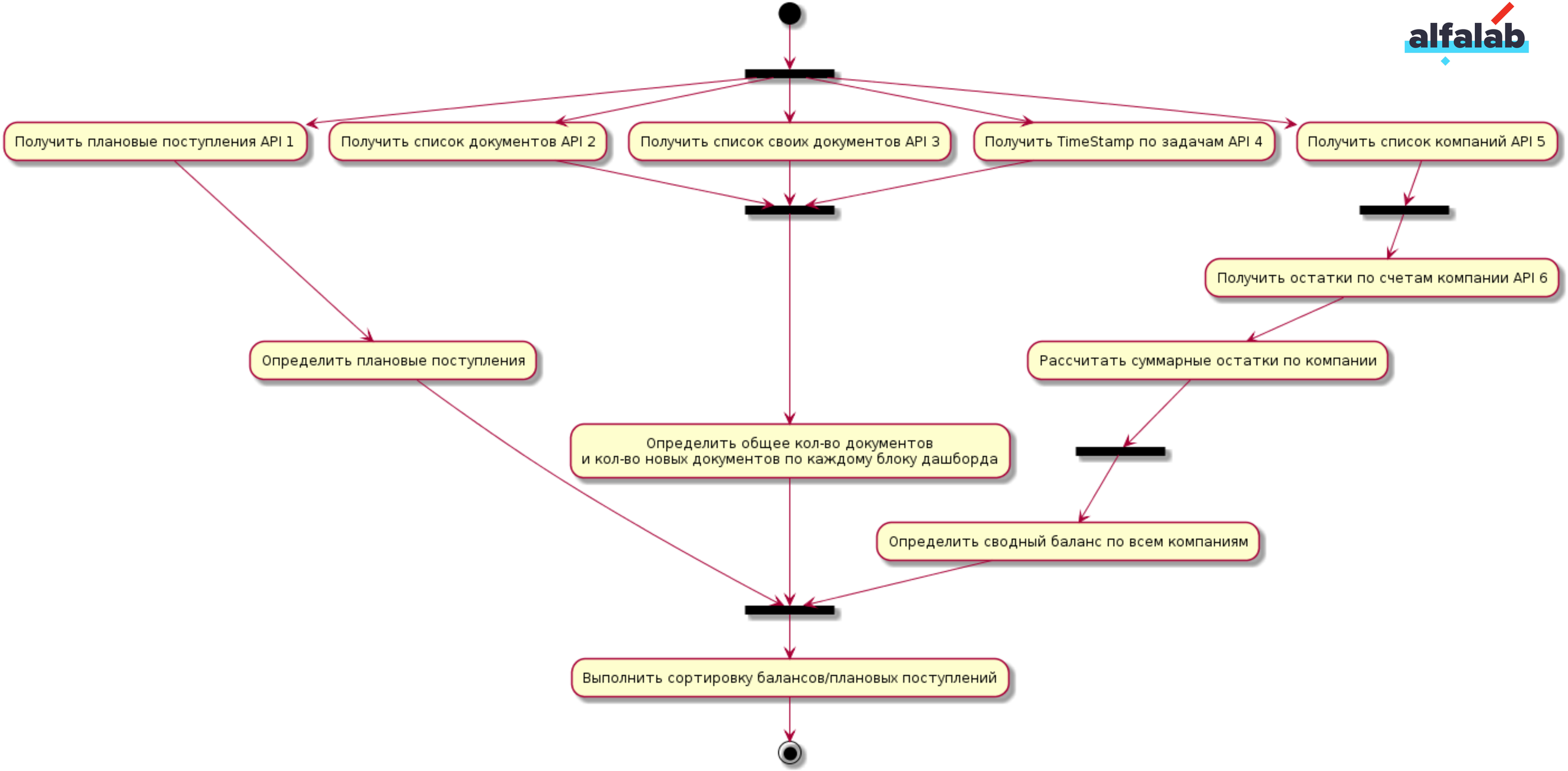
```
- hosts: dc1-rest-api
  roles:
    - { role: api, name: transactions-api, service_port: 9081 }
    - { role: api, name: cards-api, port: 8081 }
    - { role: api, name: customers-api, port: 9091 }

  dc1-rest-api
  <ip_address_1>
  <ip_address_2>
  ...
```

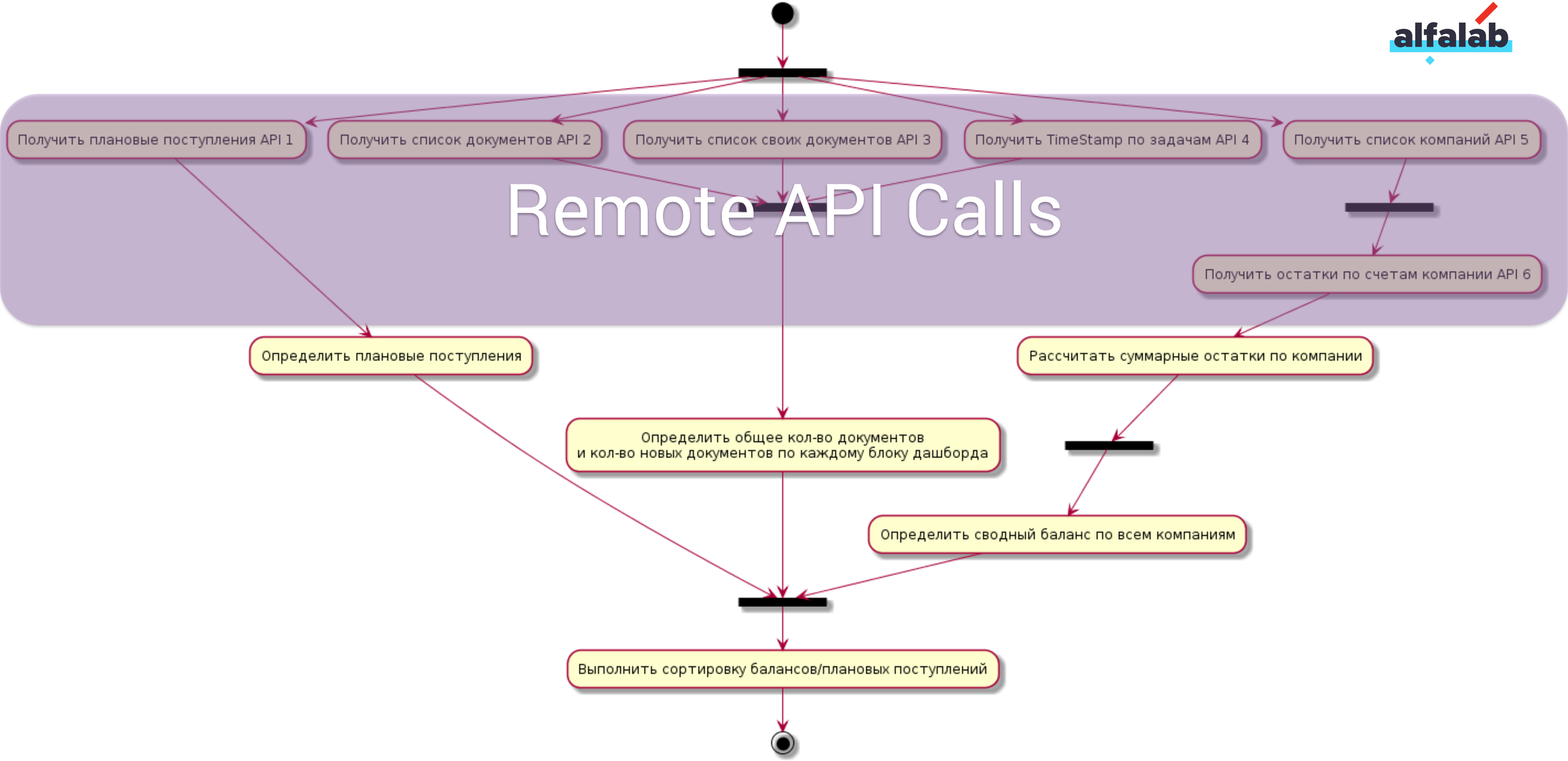
Dynamic Configuration (PAAS)

```
{
  "id": "/retail-online-bank/middle/transactions-api",
  "cpus": 1,
  "mem": 1024,
  "instances": 5,
  "container": {
    "docker": {
      "image": "docker/retail-online-bank-transactions-api:1.17.0",
      "portMappings": [
        {
          "containerPort": 8080,
          "servicePort": 0
        }
      ]
    }
  }
}
```

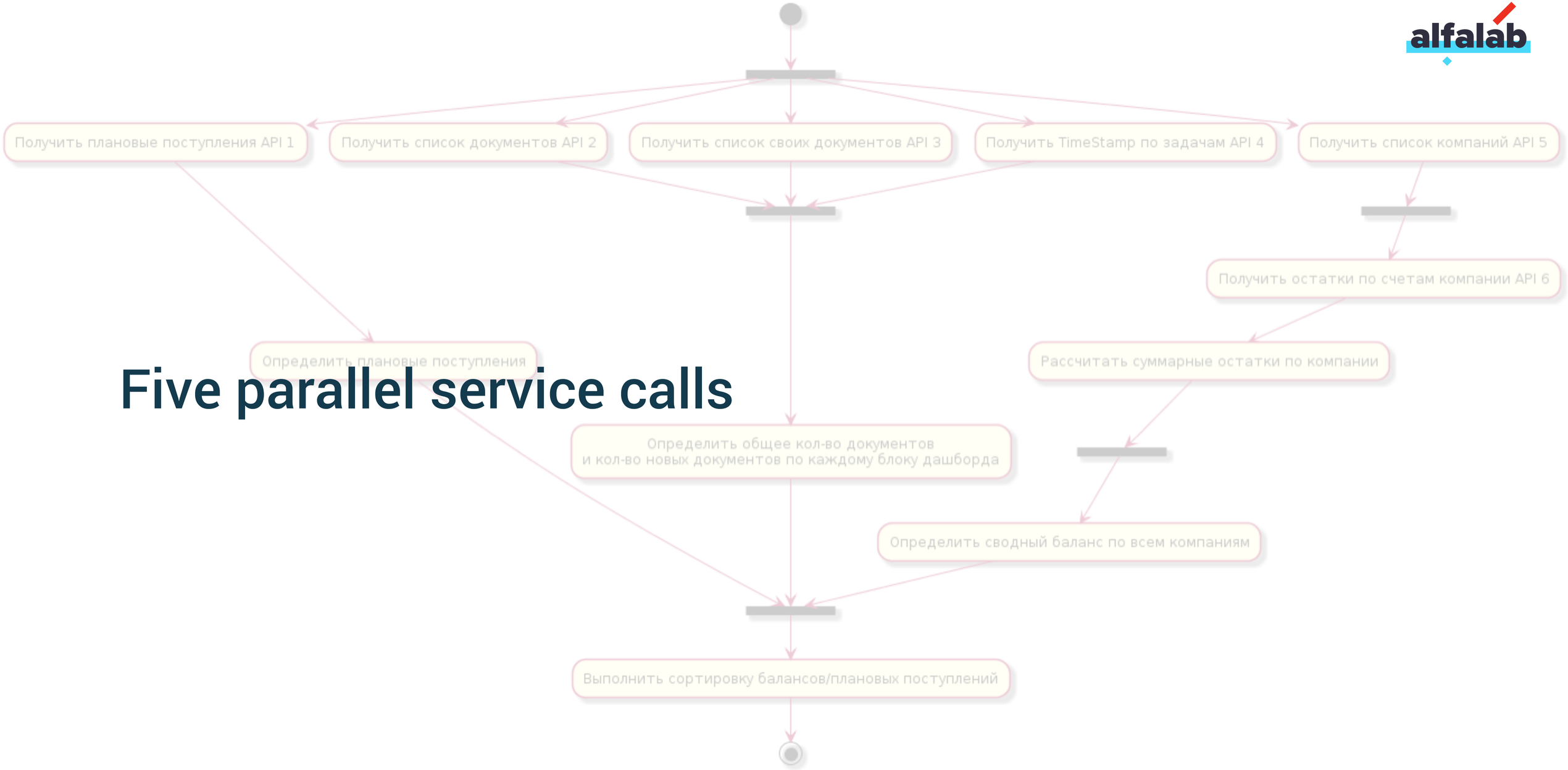
Service Discovery

What instance we can call?

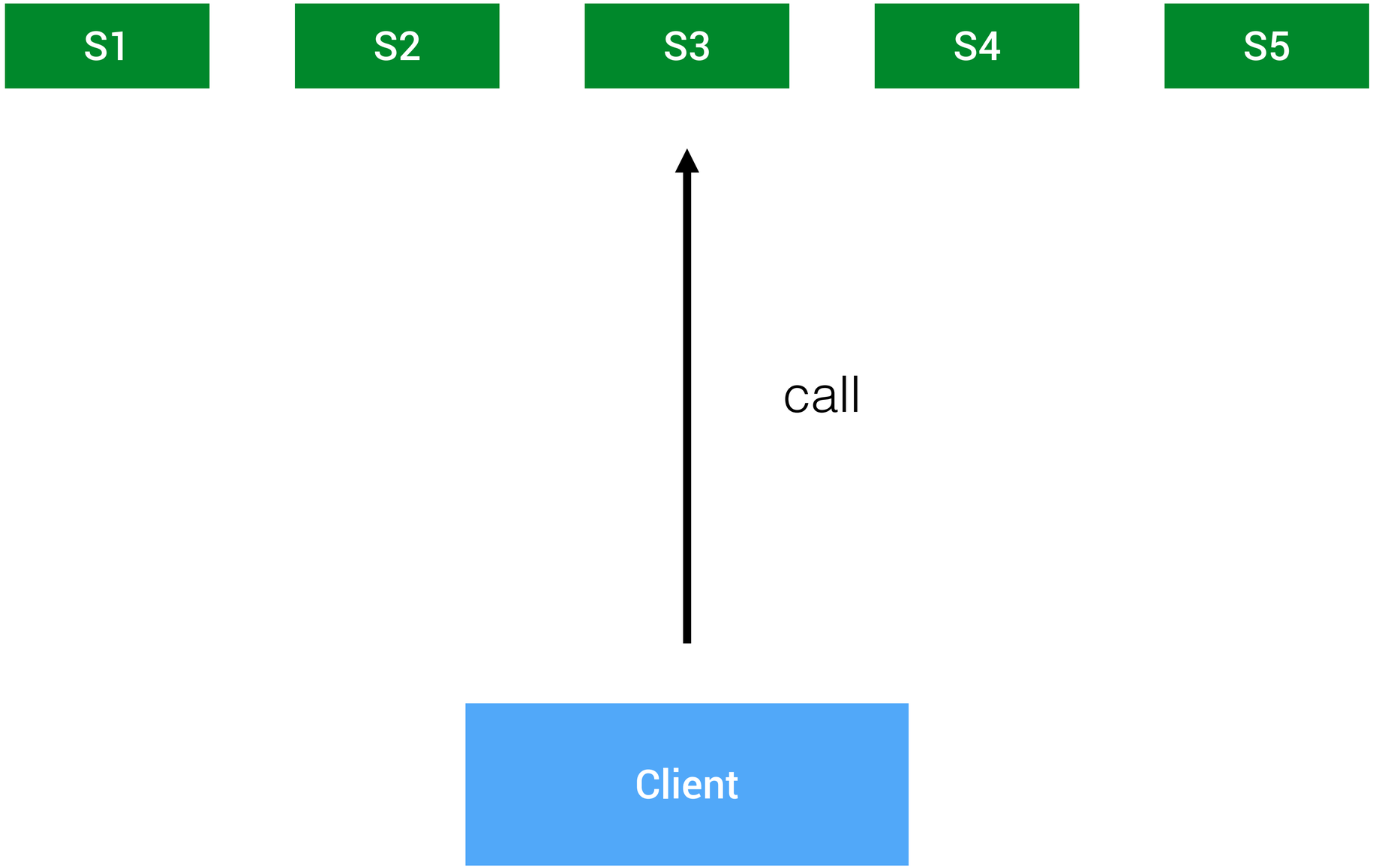

Получить плановые поступления API 1
Получить список документов API 2
Получить список своих документов API 3
Получить TimeStamp по задачам API 4
Получить список компаний API 5
Получить остатки по счетам компании API 6
Определить плановые поступления
Рассчитать суммарные остатки по компании
Определить общее кол-во документов
и кол-во новых документов по каждому блоку дашборда
Определить сводный баланс по всем компаниям
Выполнить сортировку балансов/плановых поступлений

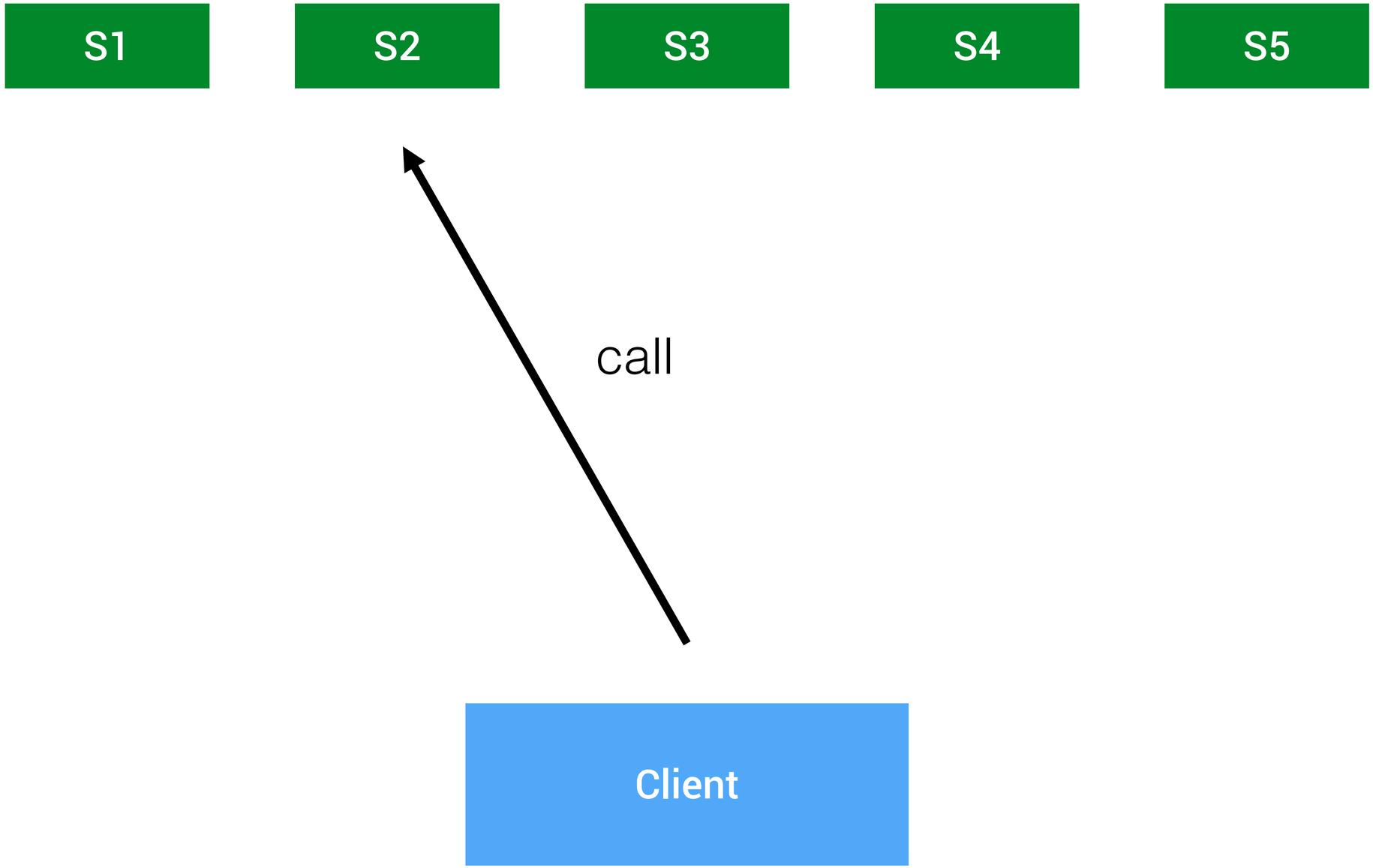

Remote API Calls
Получить плановые поступления API 1
Получить список документов API 2
Получить список своих документов API 3
Получить TimeStamp по задачам API 4
Получить список компаний API 5
Получить остатки по счетам компании API 6
Определить плановые поступления
Рассчитать суммарные остатки по компании
Определить общее кол-во документов
и кол-во новых документов по каждому блоку дашборда
Определить сводный баланс по всем компаниям
Выполнить сортировку балансов/плановых поступлений

Five parallel service calls

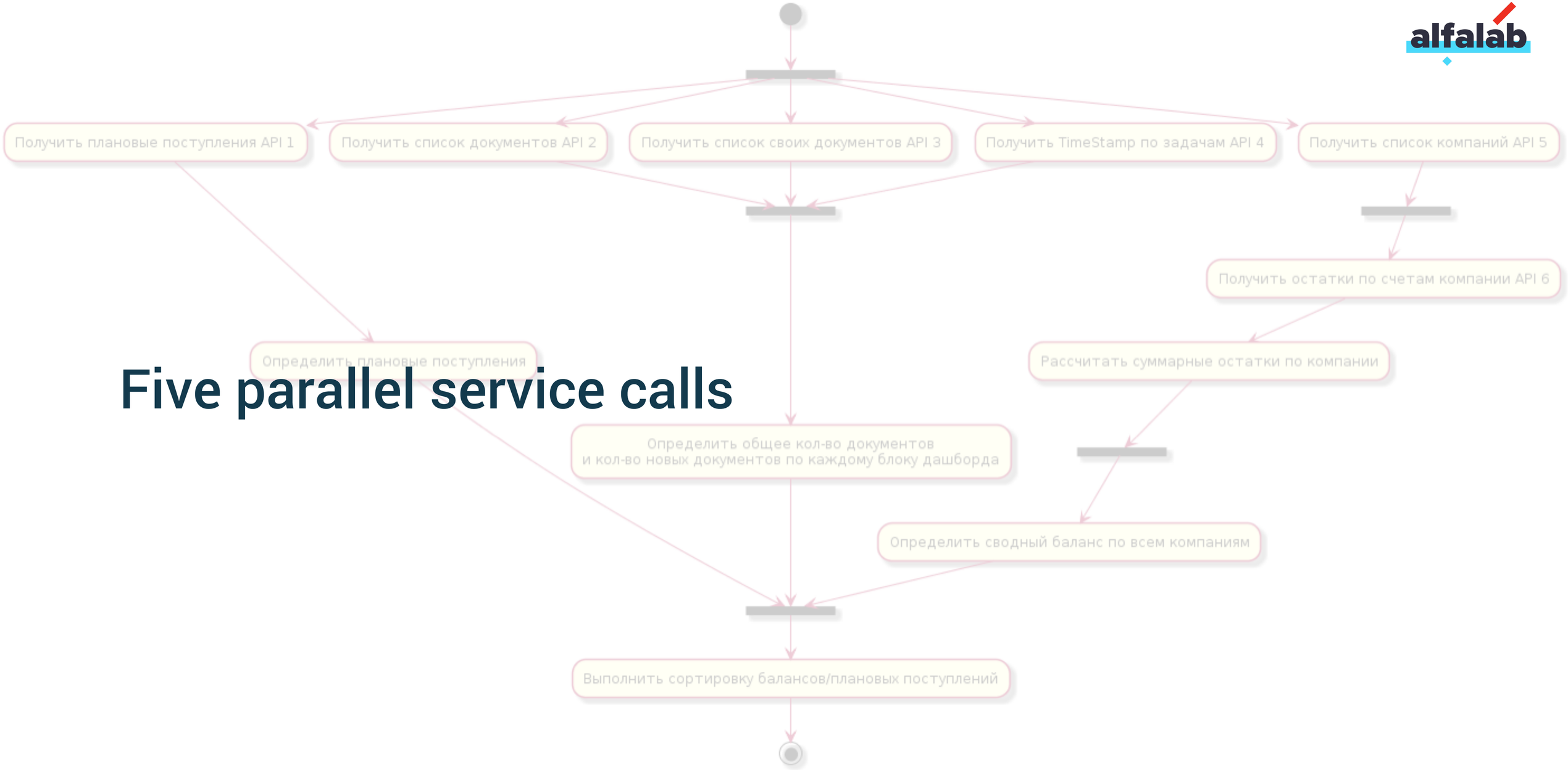


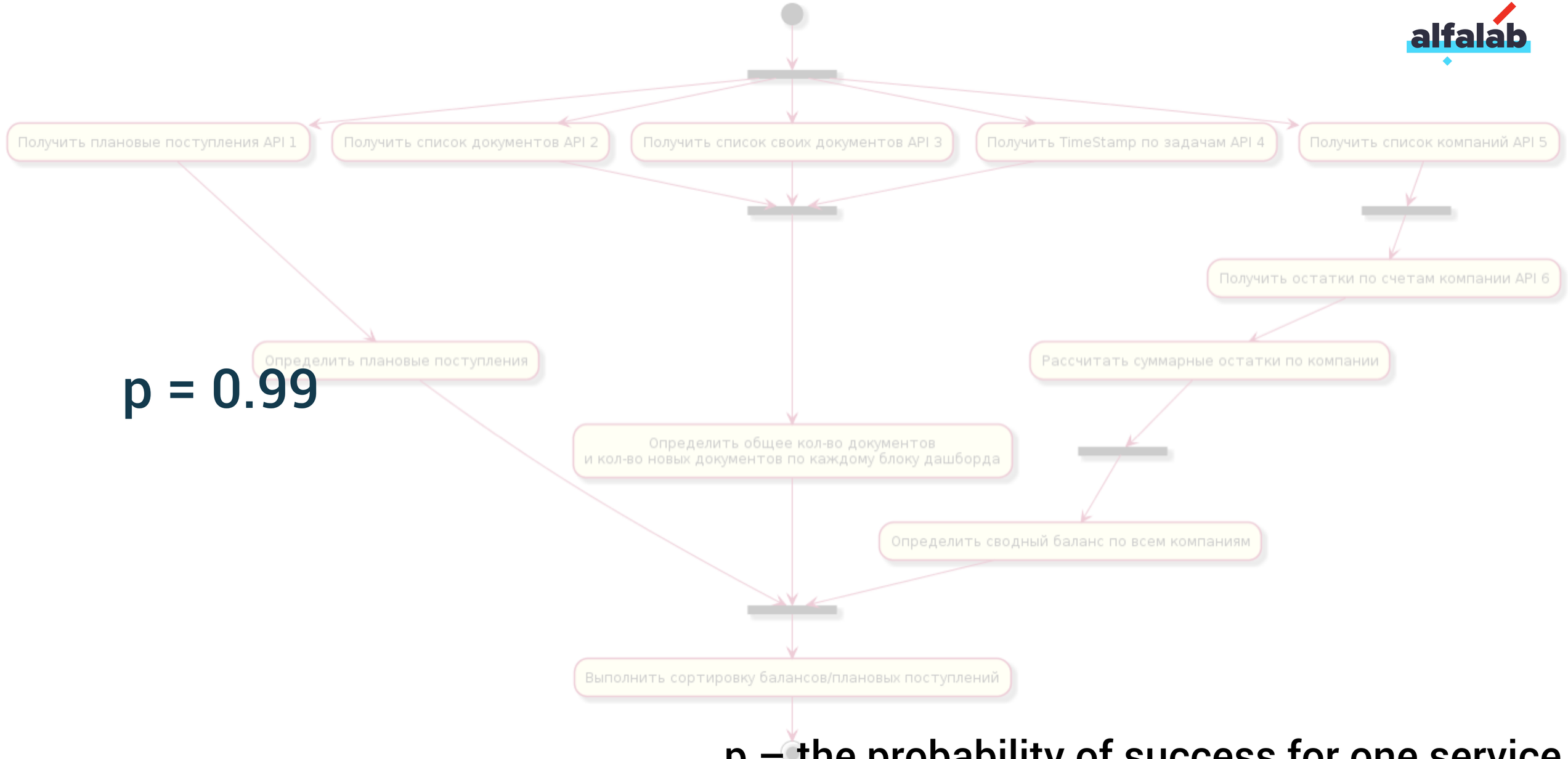
S1 S2 S3 S4 S5


S1
S2
S3
S4
S5
call
Client


S1
S2
S3
S4
S5
call
Client

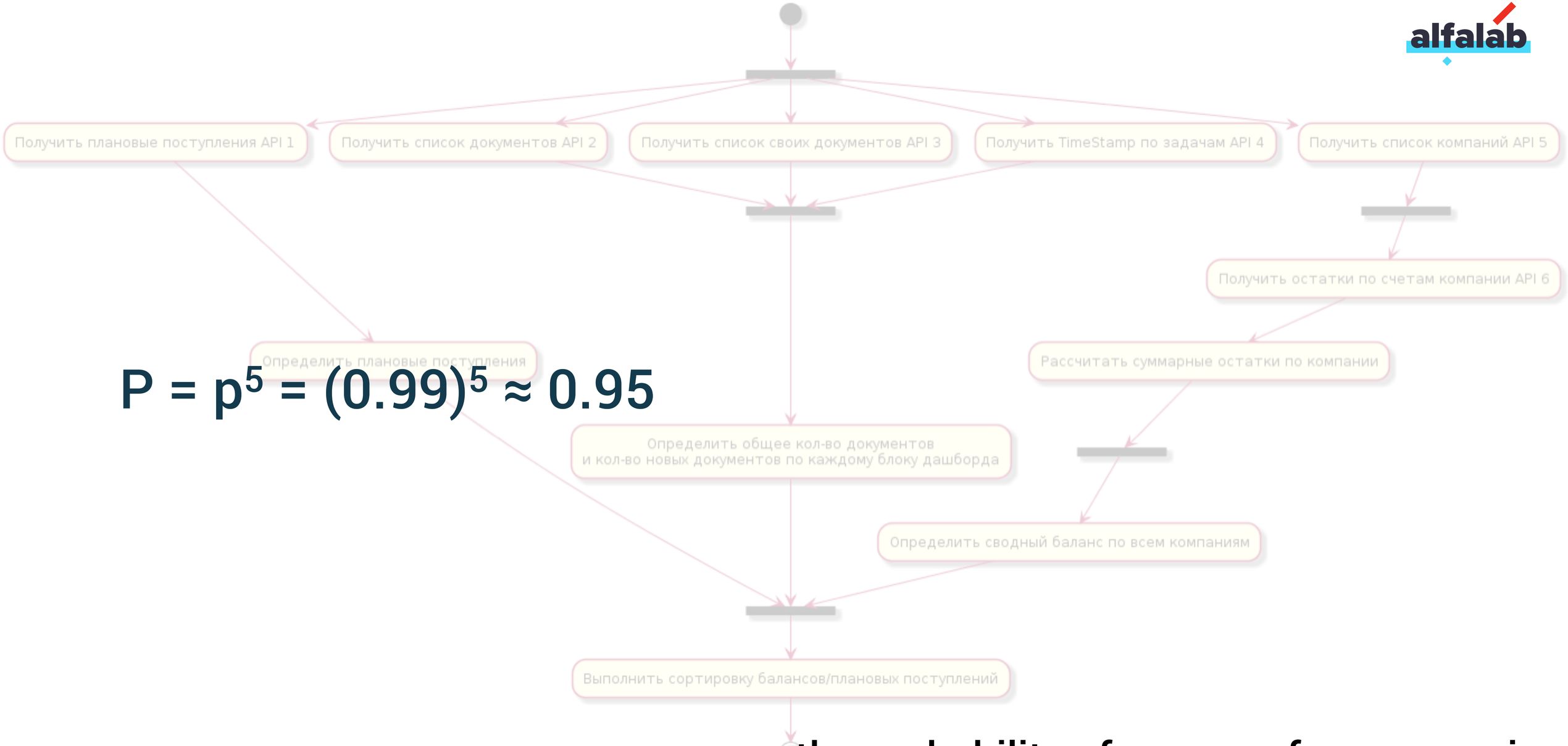
Five parallel service calls





p = 0.99

p – the probability of success for one service



$$P = p^5 = (0.99)^5 \approx 0.95$$

p – the probability of success for one service

alfalab

Service Discovery

What instance we can call?

What instance is better for call?

Service Discovery

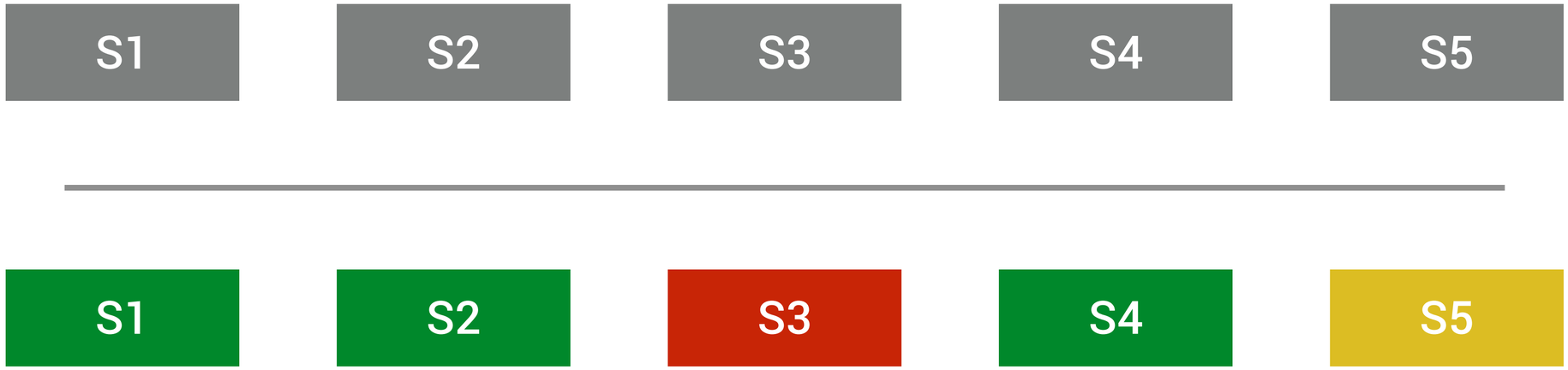
What instance we can call?

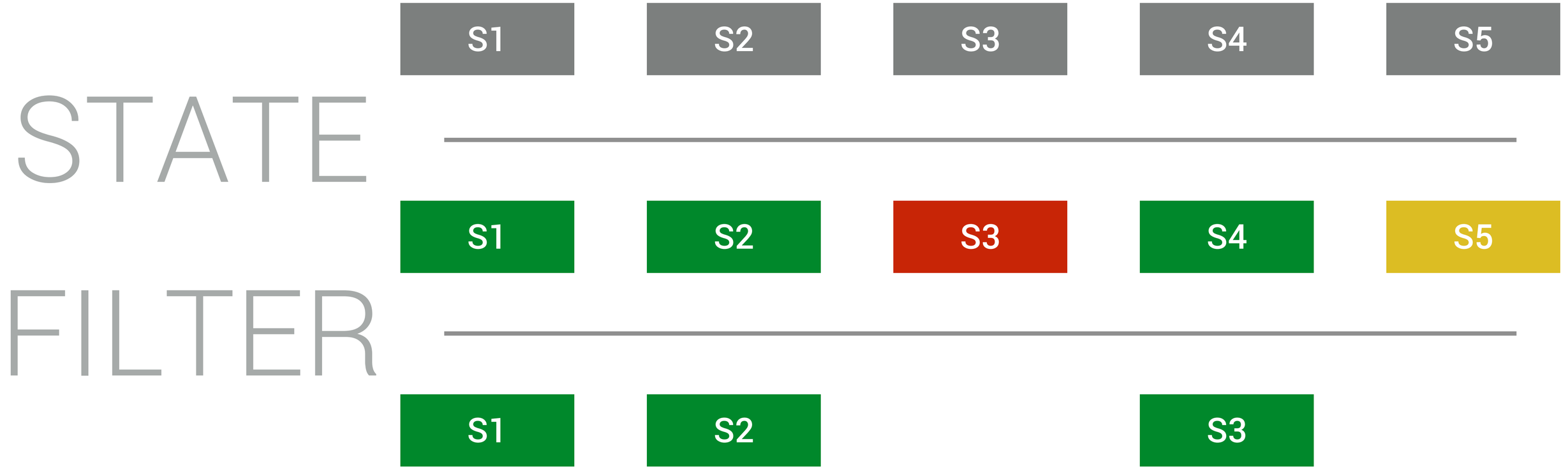
What instance is better for call?

How we can increase probability of success for one call?

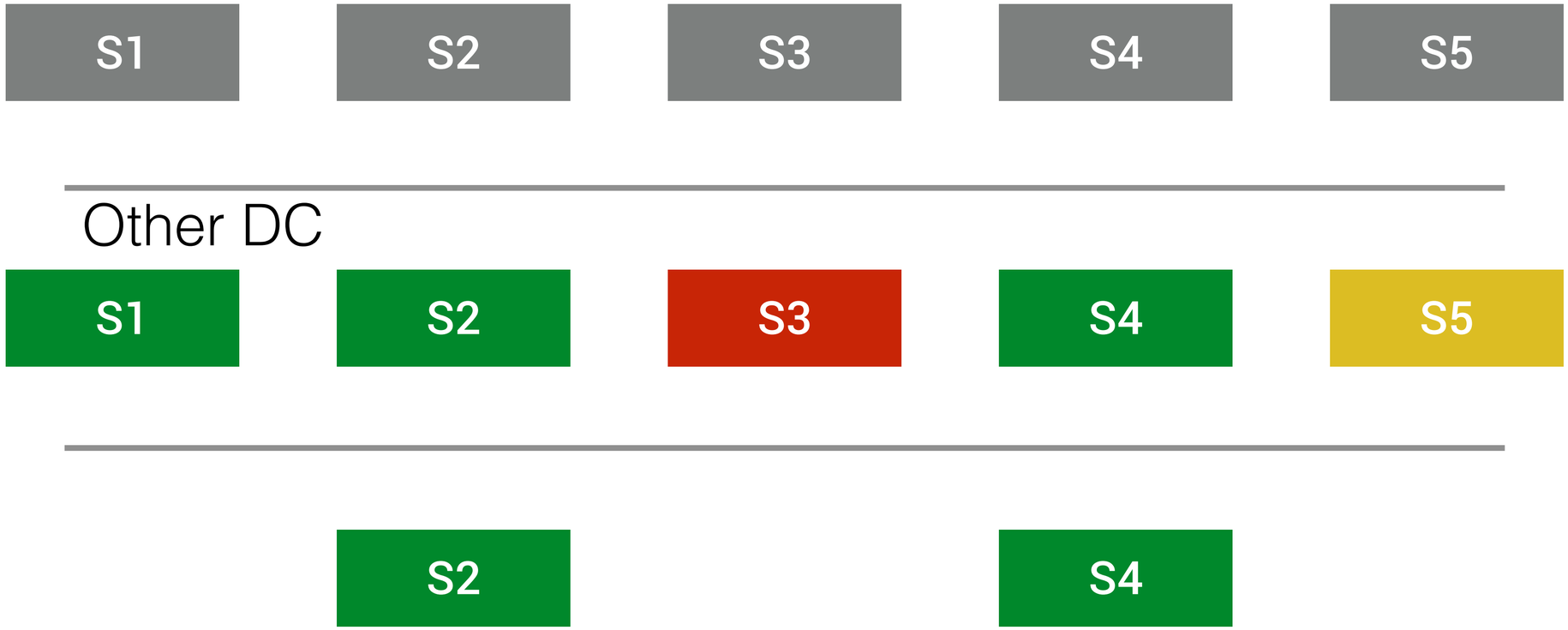
S1 S2 S3 S4 S5

STATE

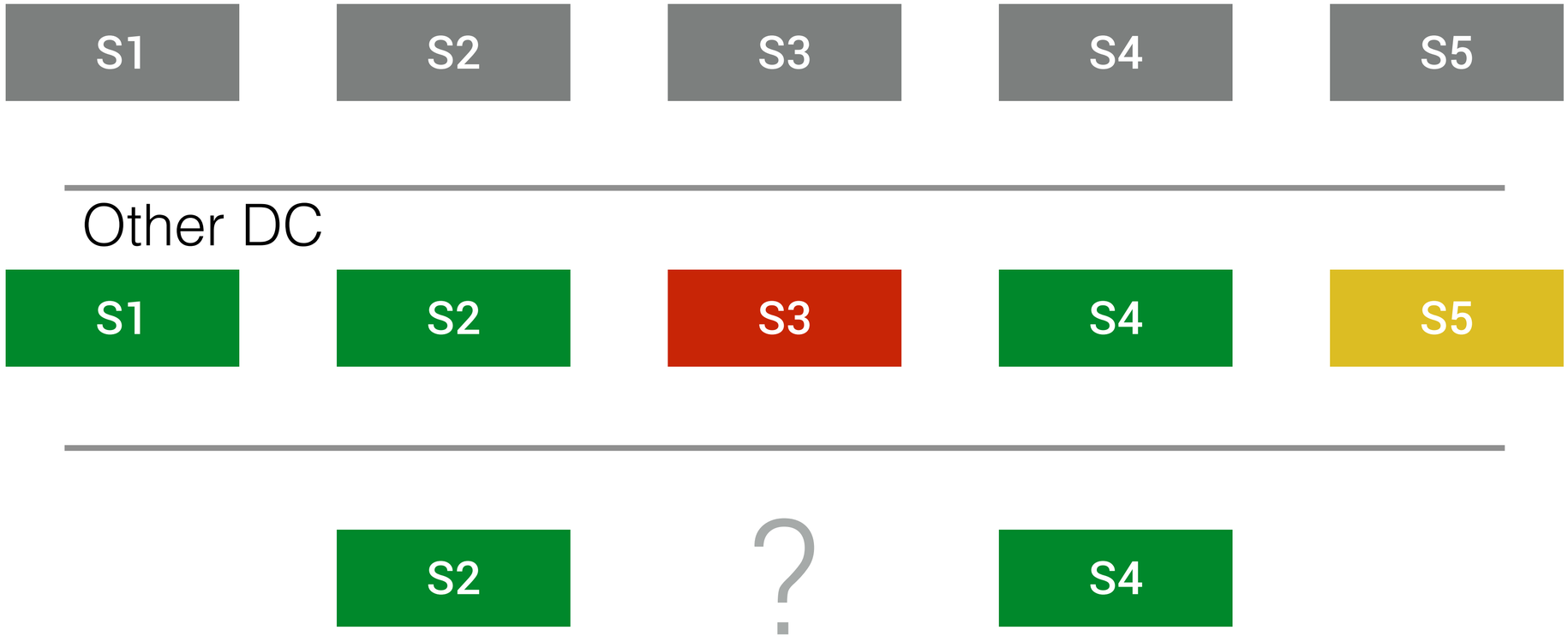


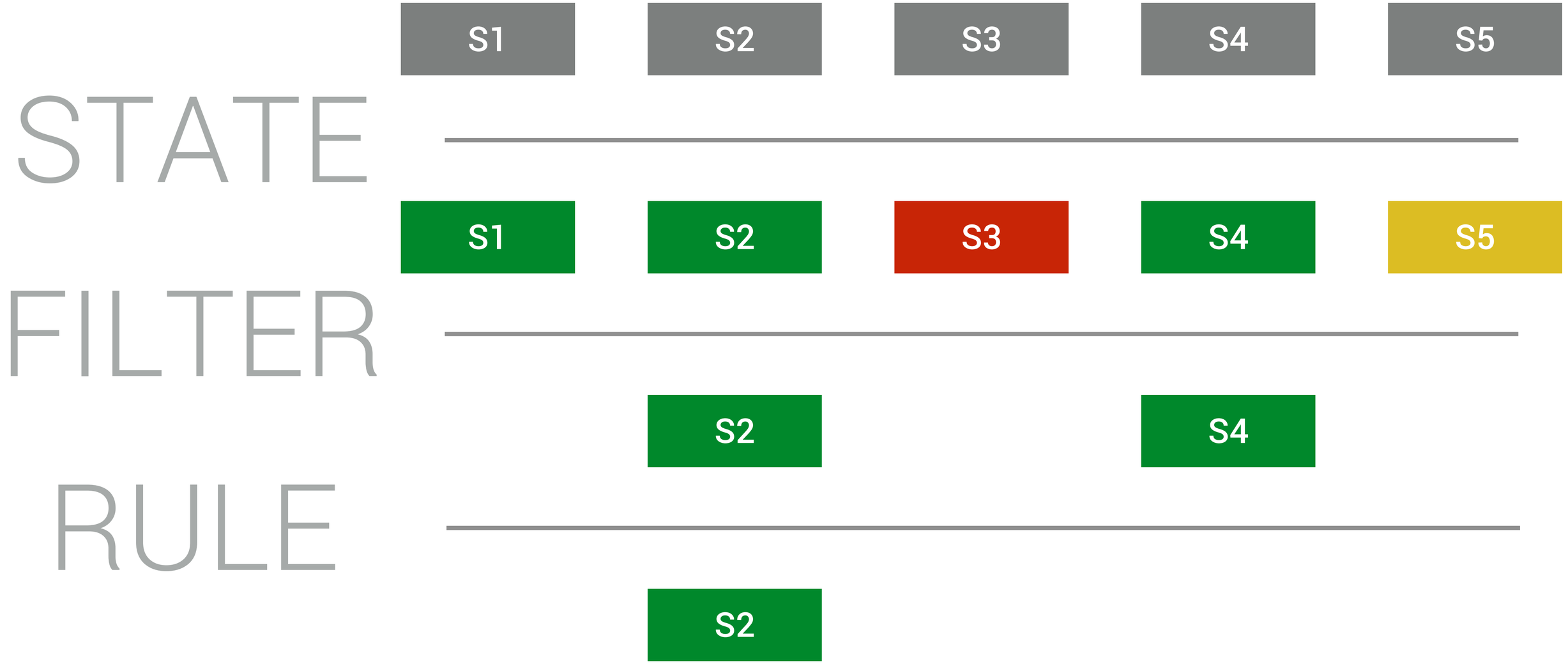

S1
S2
S3
S4
S5
STATE
S1
S2
S3
S4
S5
FILTER
S1
S2
S3

STATE FILTER



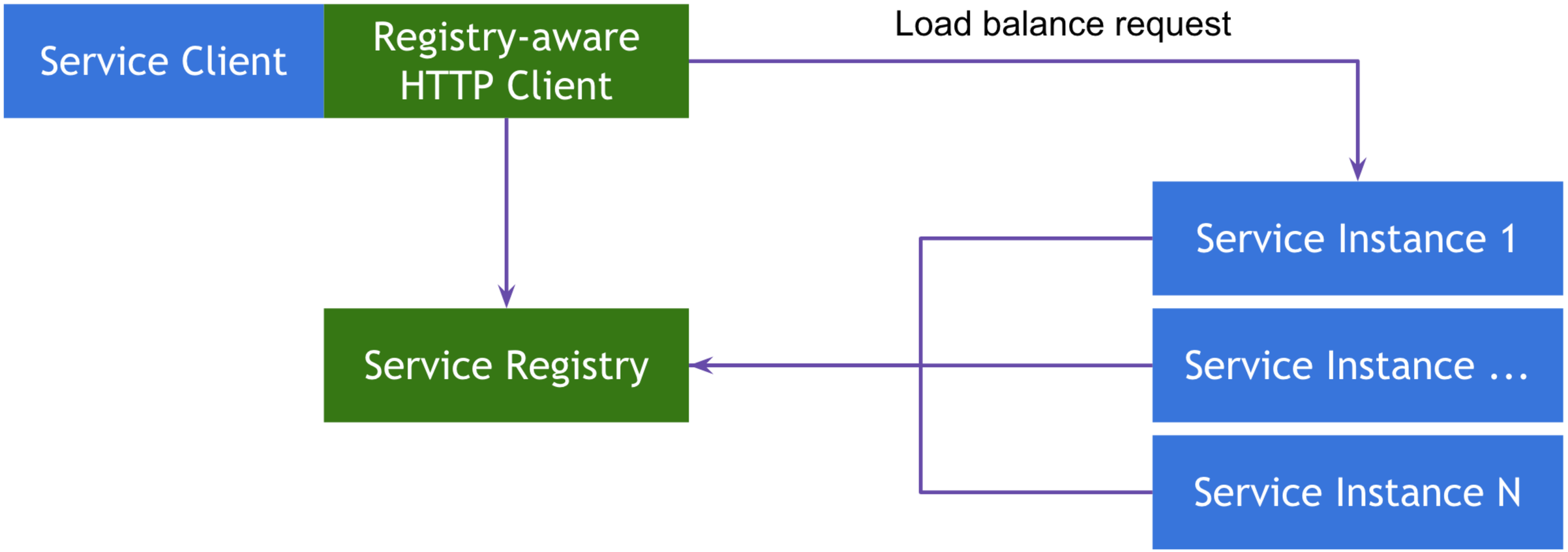
STATE FILTER



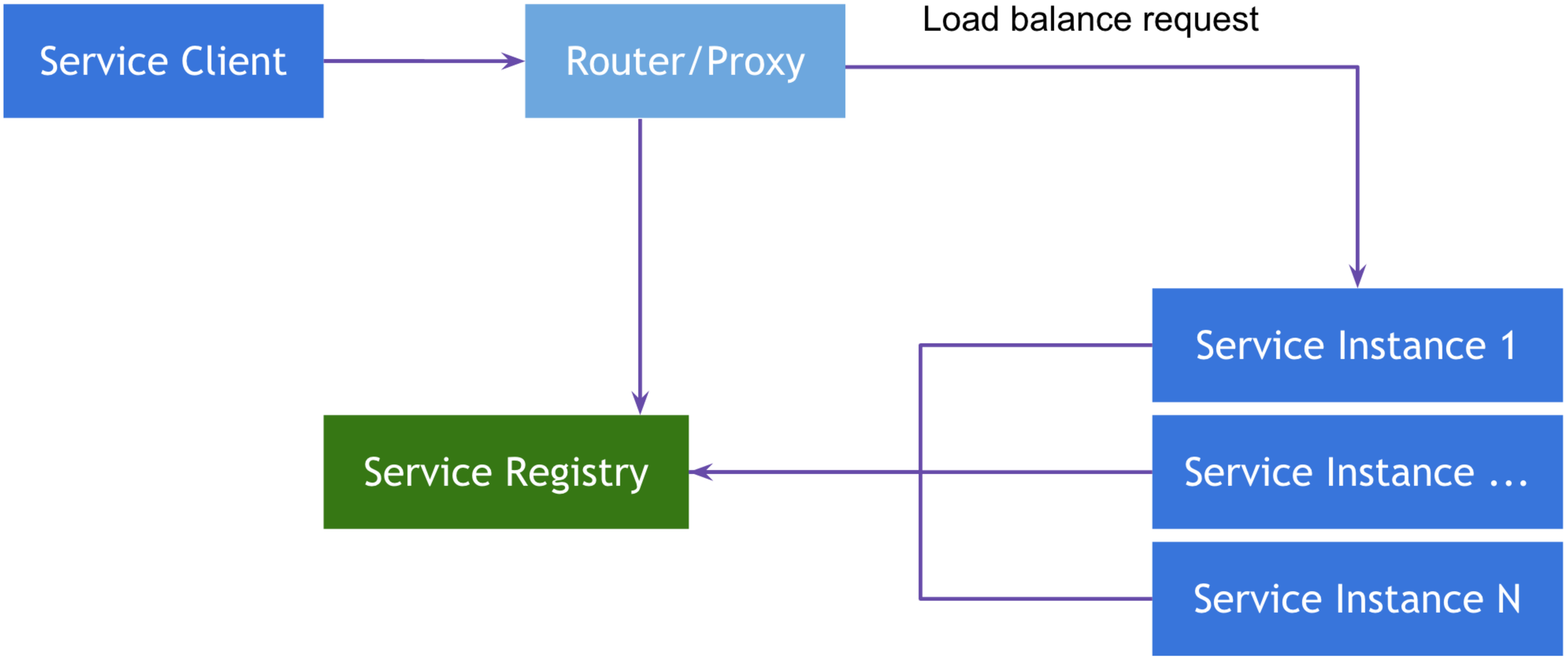

S1
S2
S3
S4
S5
STATE
S1
S2
S3
S4
S5
FILTER
S2
S4
RULE
S2


S1
S2
S3
S4
S5
STATE
S1
S2
S3
S4
S5
FILTER
S2
S4
RULE
S2
call
Client

Client-Side Service Discovery



Server-Side Service Discovery



Server-Side

Dummy client

Language/Library tolerance

Single Responsibility Principle

Hard to implement something complex and smart

One more point of failure

VS

Client-Side

More complex and smart decision implementation

Avoid point of failure

Flexibility

Very good implementation is required

Lock on libraries

Hard to make changes

Spring Cloud

Tools for building common patterns in distributed systems

Part of Spring Boot

Implements Client-Side Service Discovery Pattern

Rollback

alfalab

How we get them all?

Service Registry?

Database

Properties File

Environment Variables

Service Registry?

Database

Properties File

Environment Variables

Service Registry

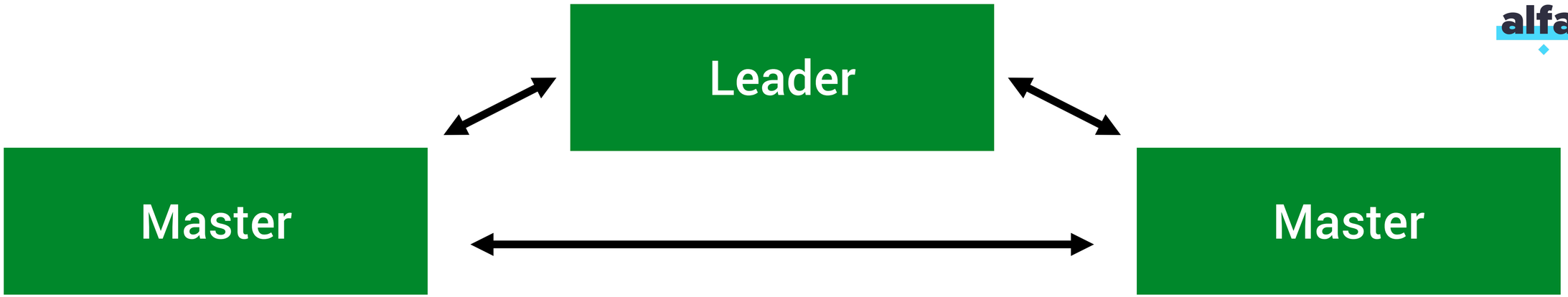
Netflix Eureka

Consul

etcd

Zookeeper

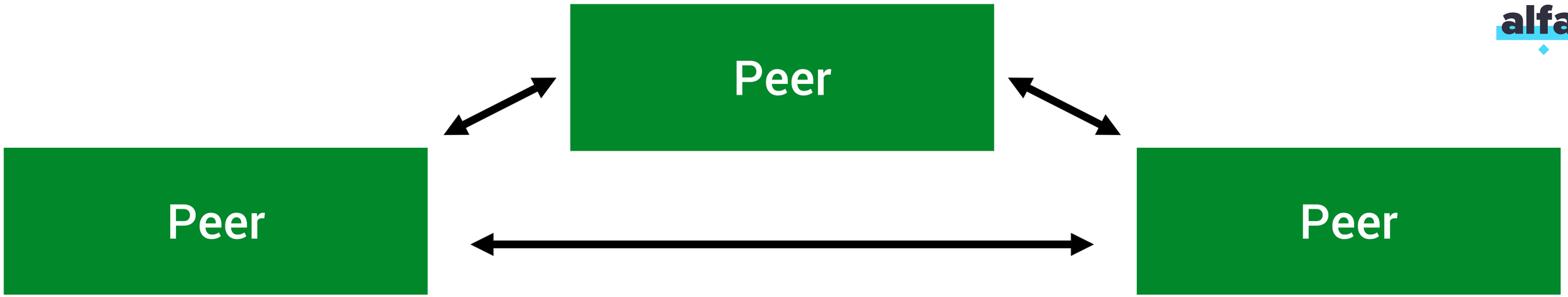
...



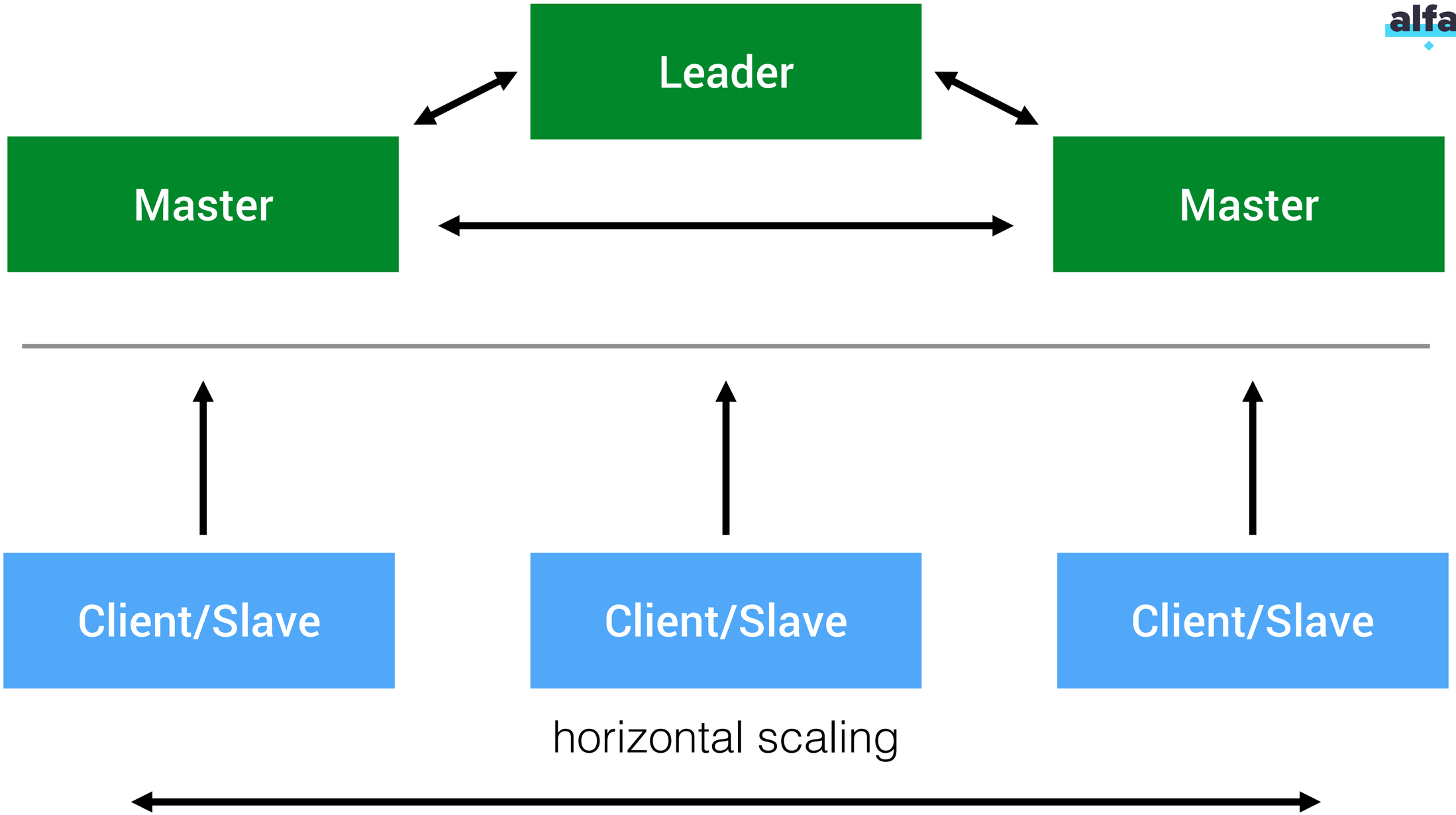
Leader-Election

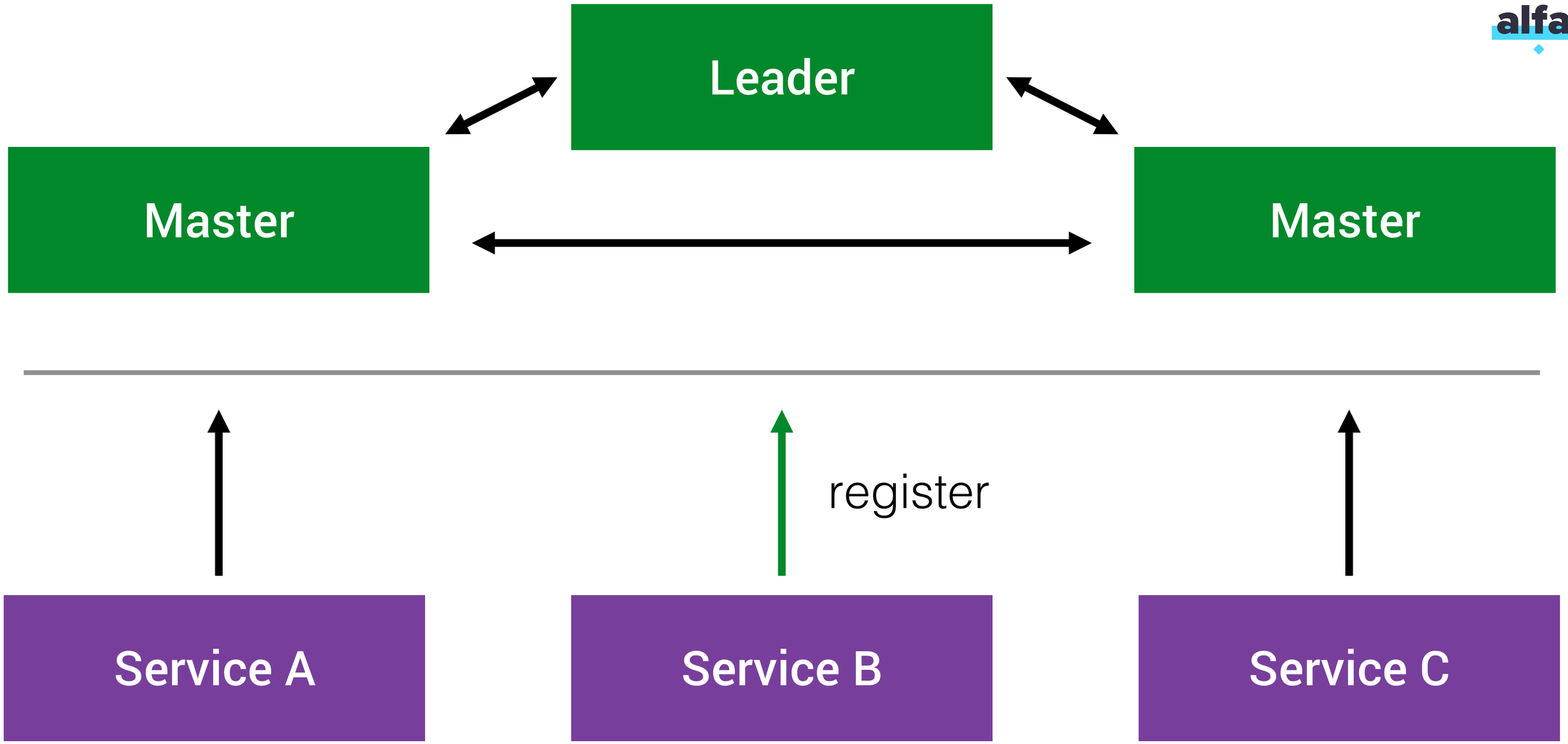
Quorum (n + 1)/2

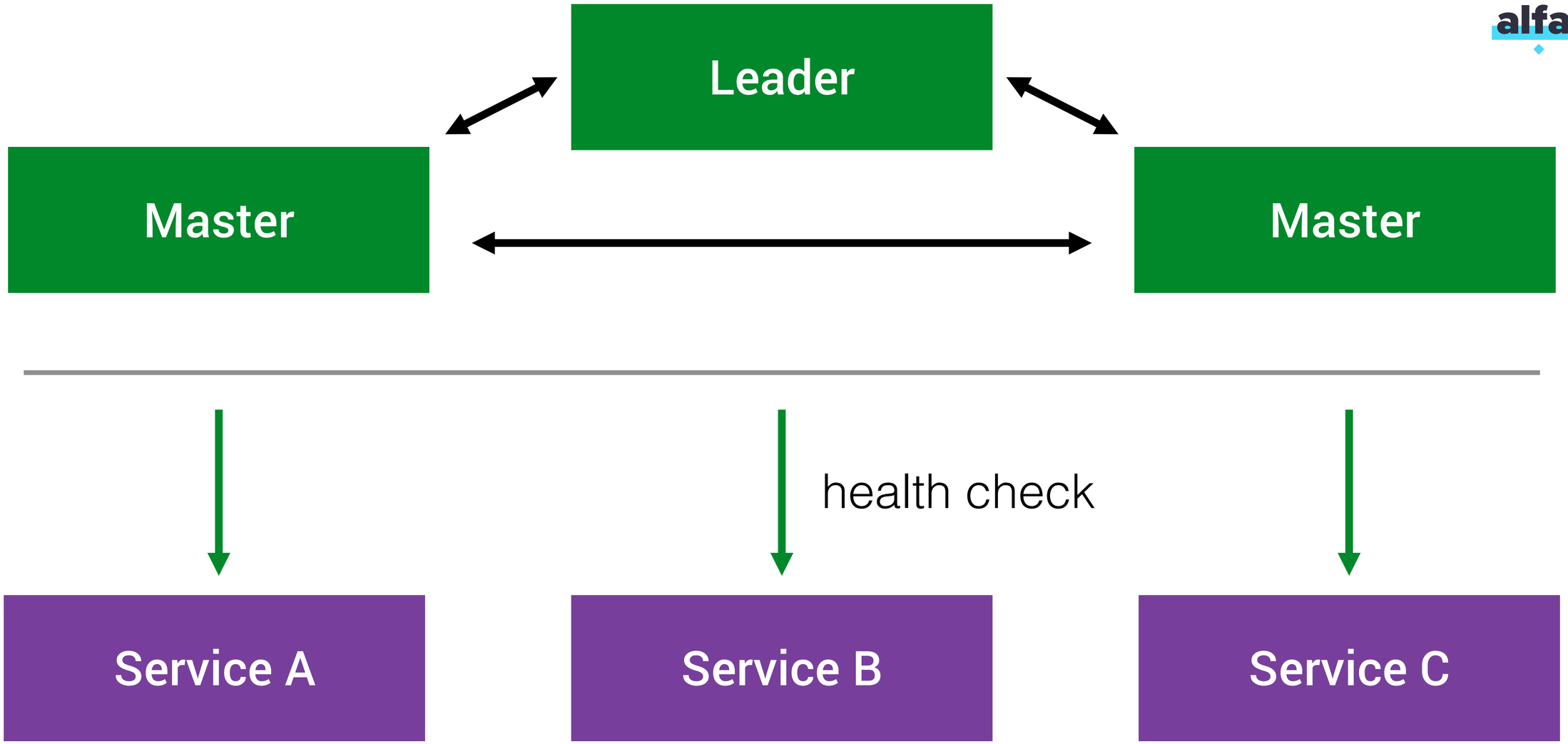
Distributed Locks

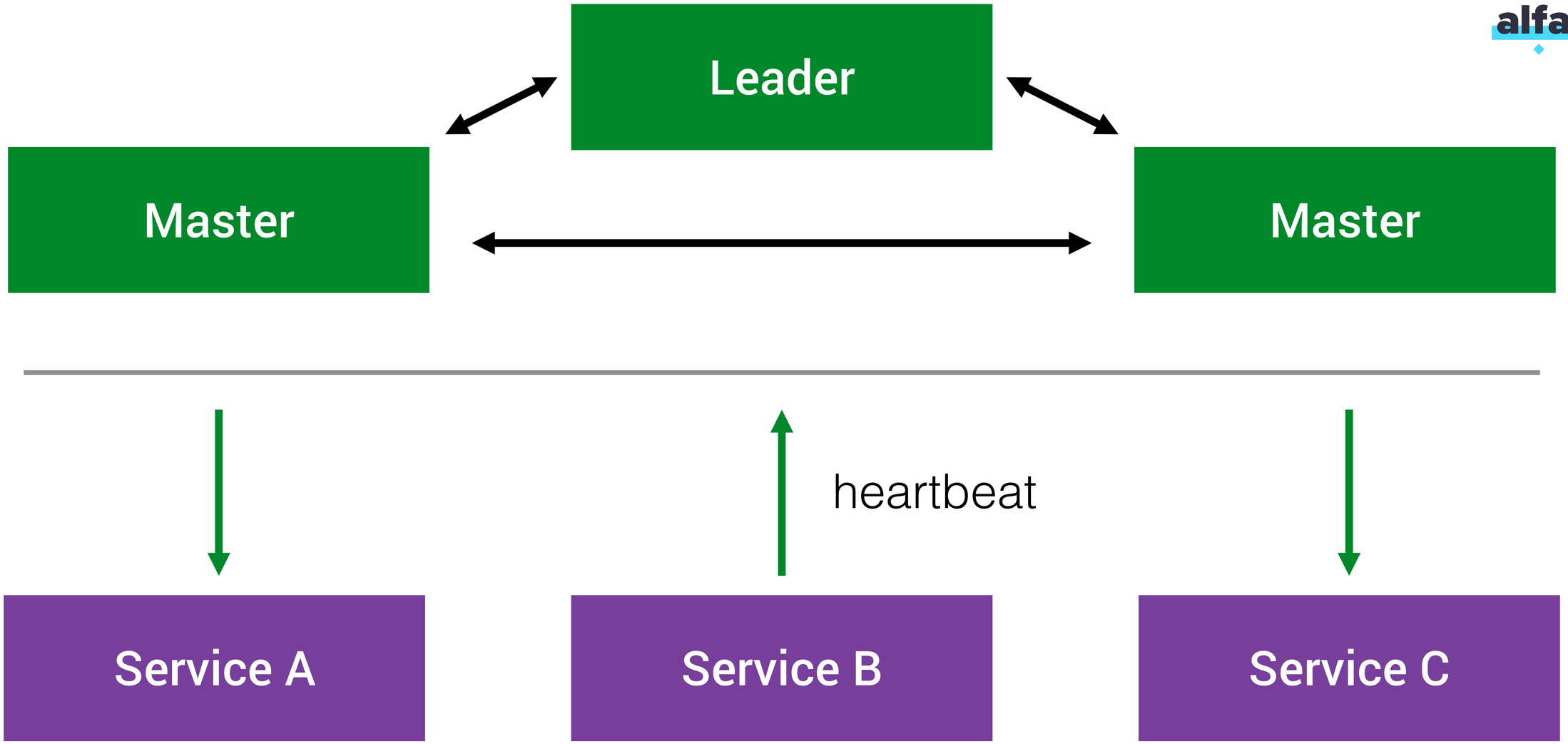


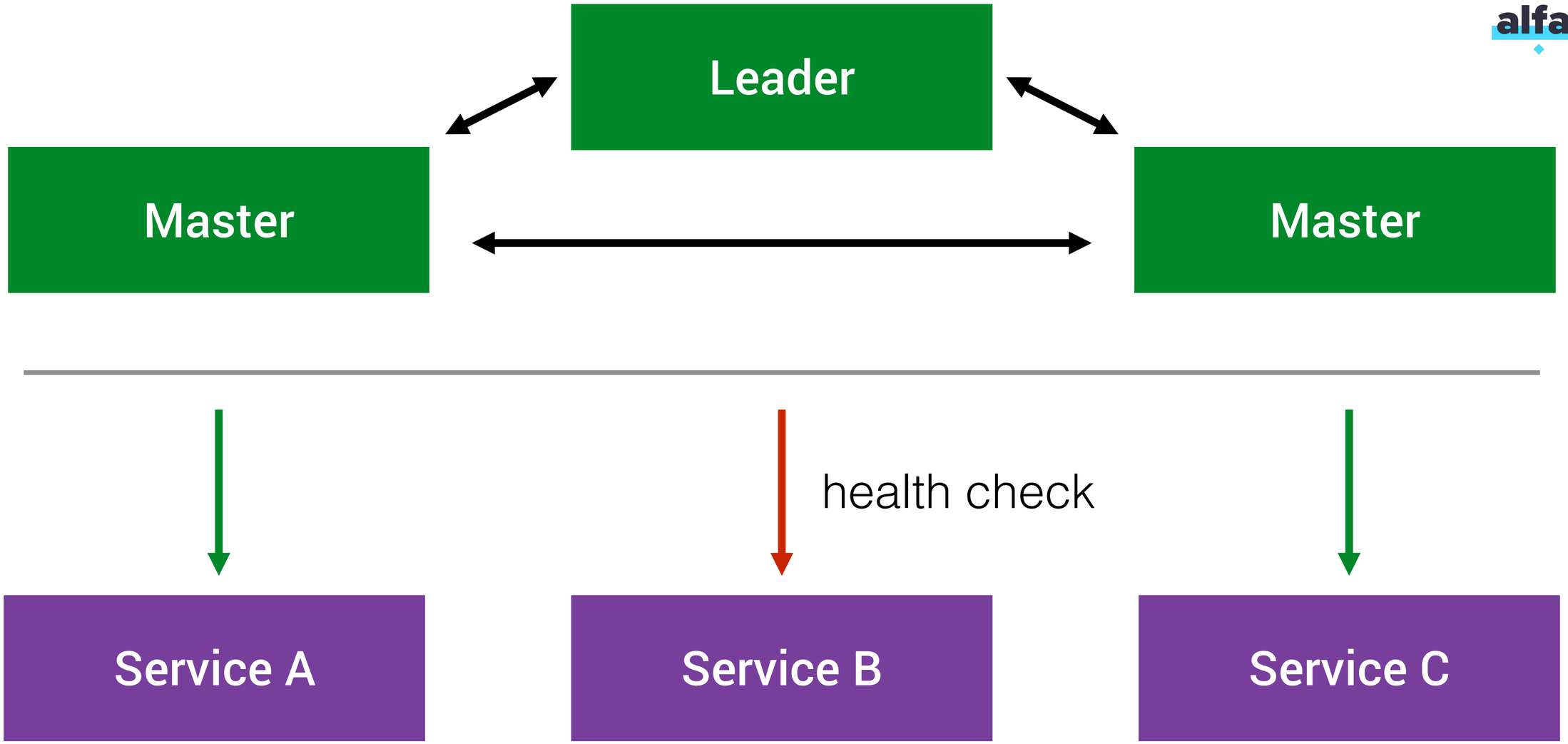
Eureka is not the same

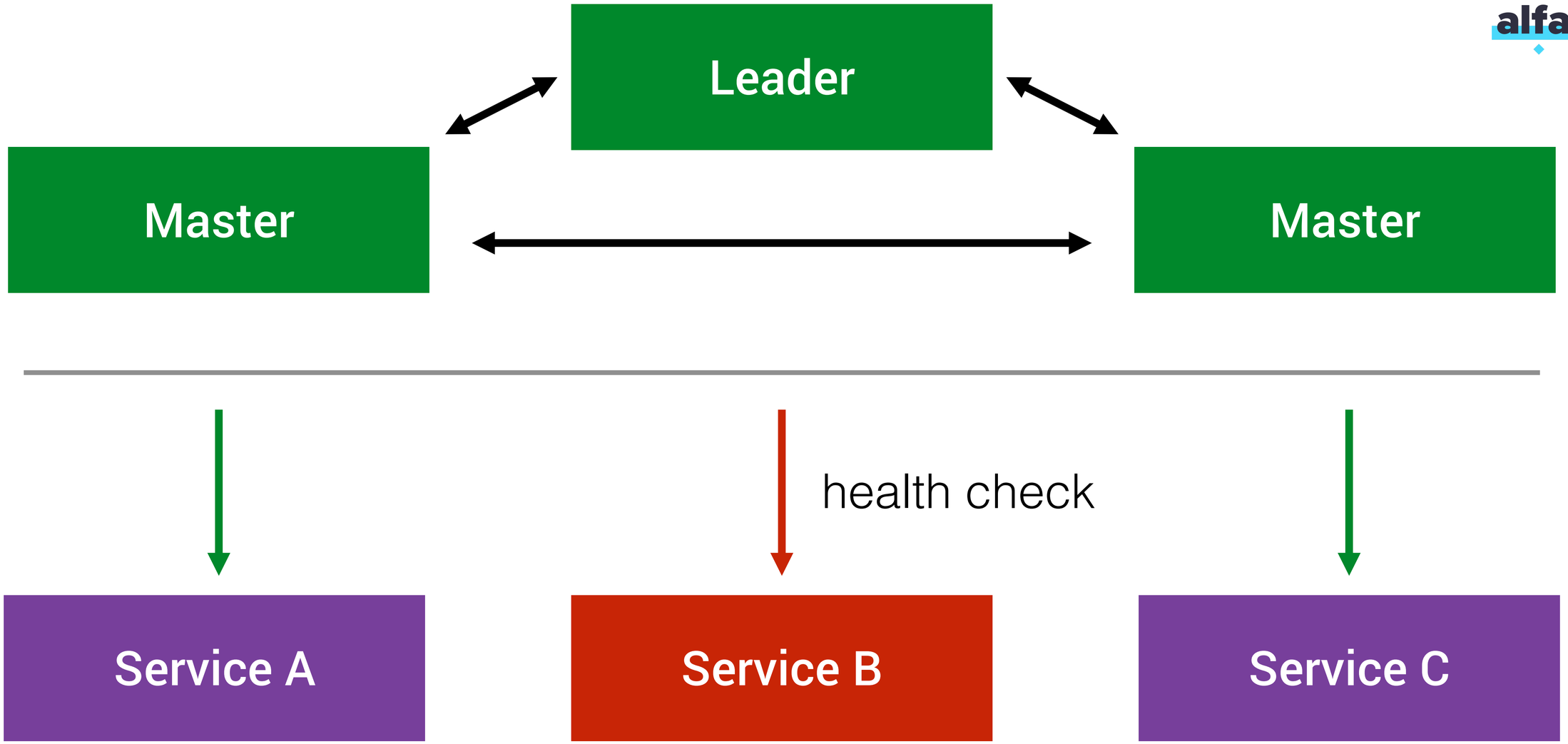

Leader
Master
Master
Client/Slave
Client/Slave
Client/Slave
horizontal scaling

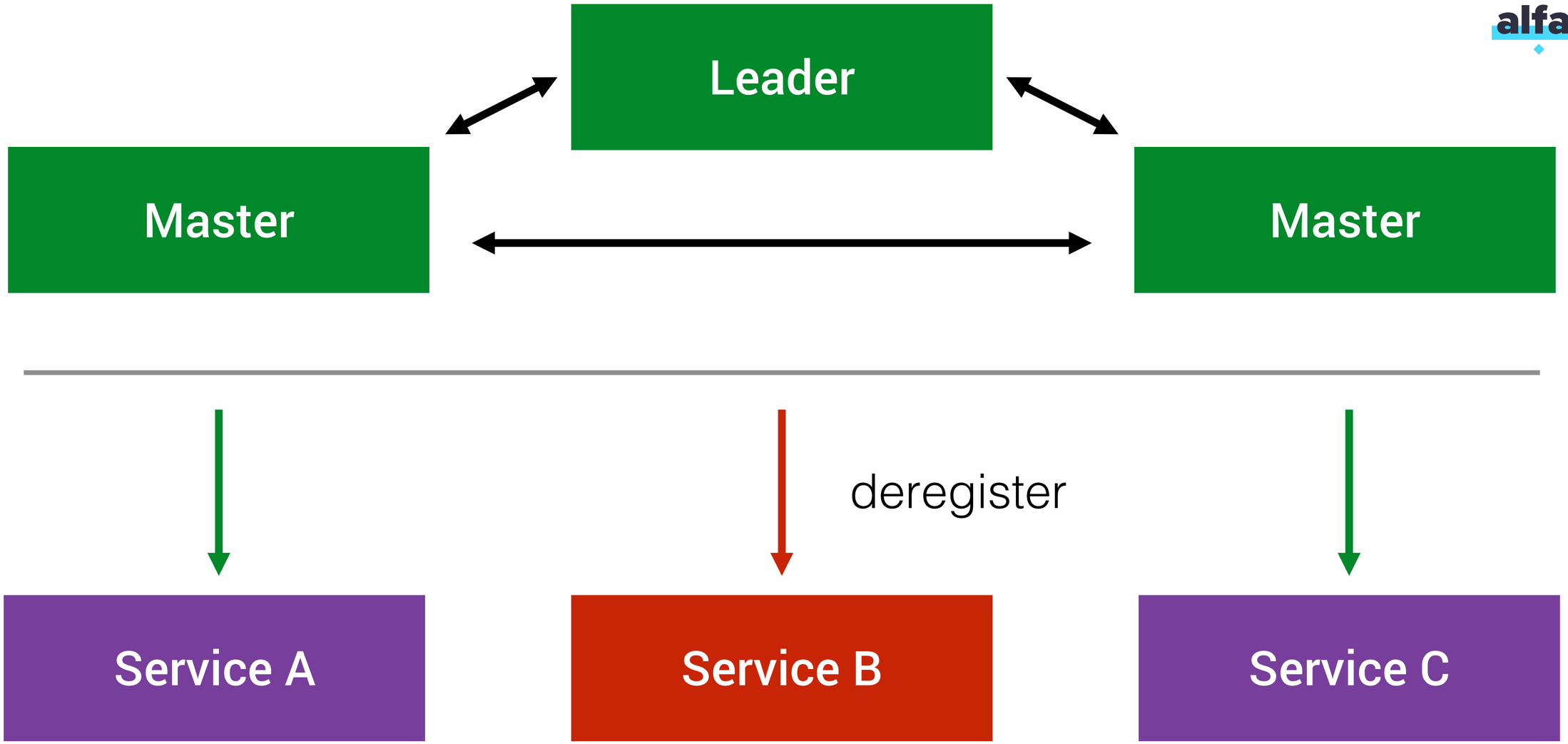

Leader
Master
Master
register
Service A
Service B
Service C

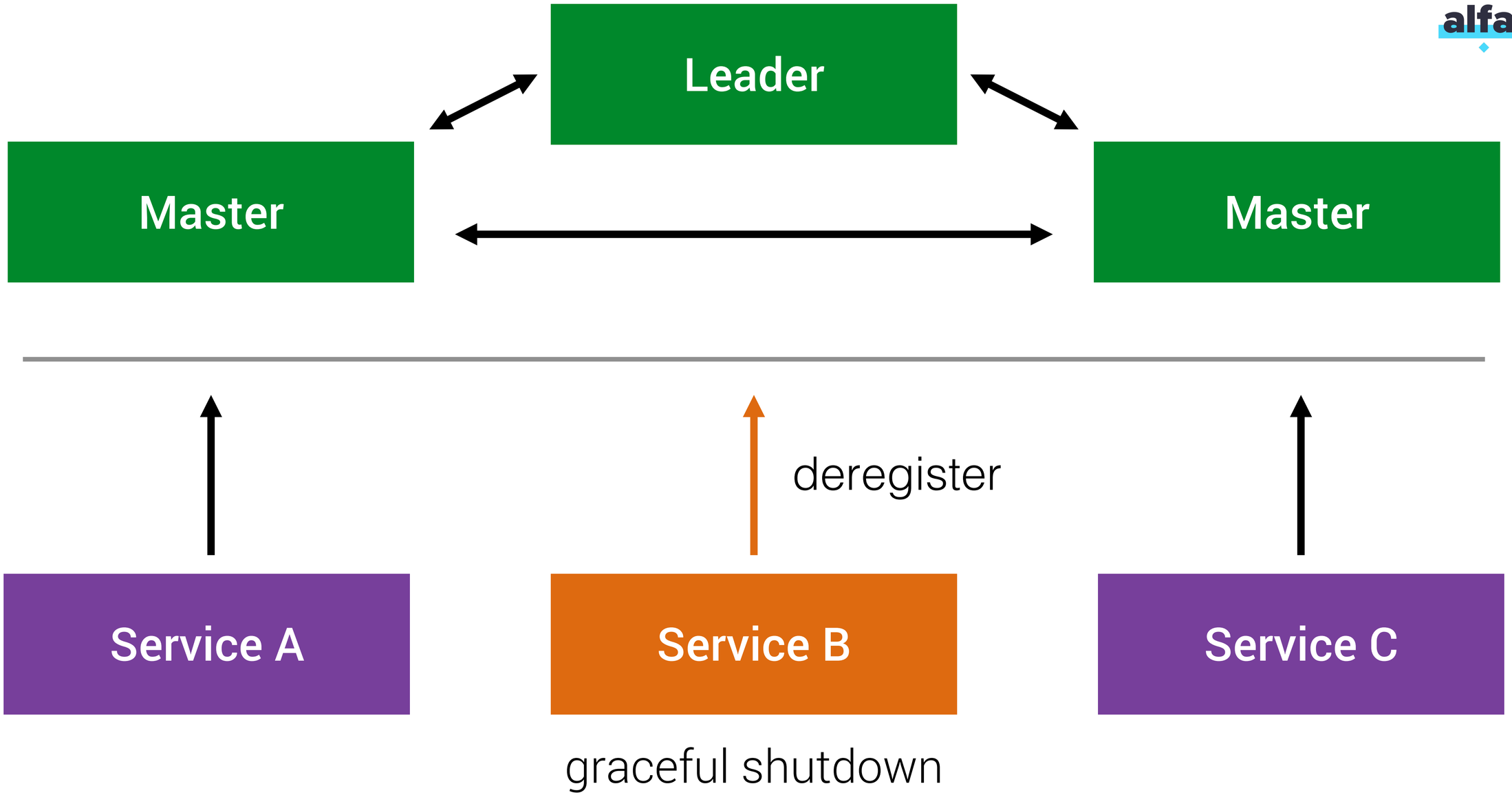

Leader
Master
Master
health check
Service A
Service B
Service C

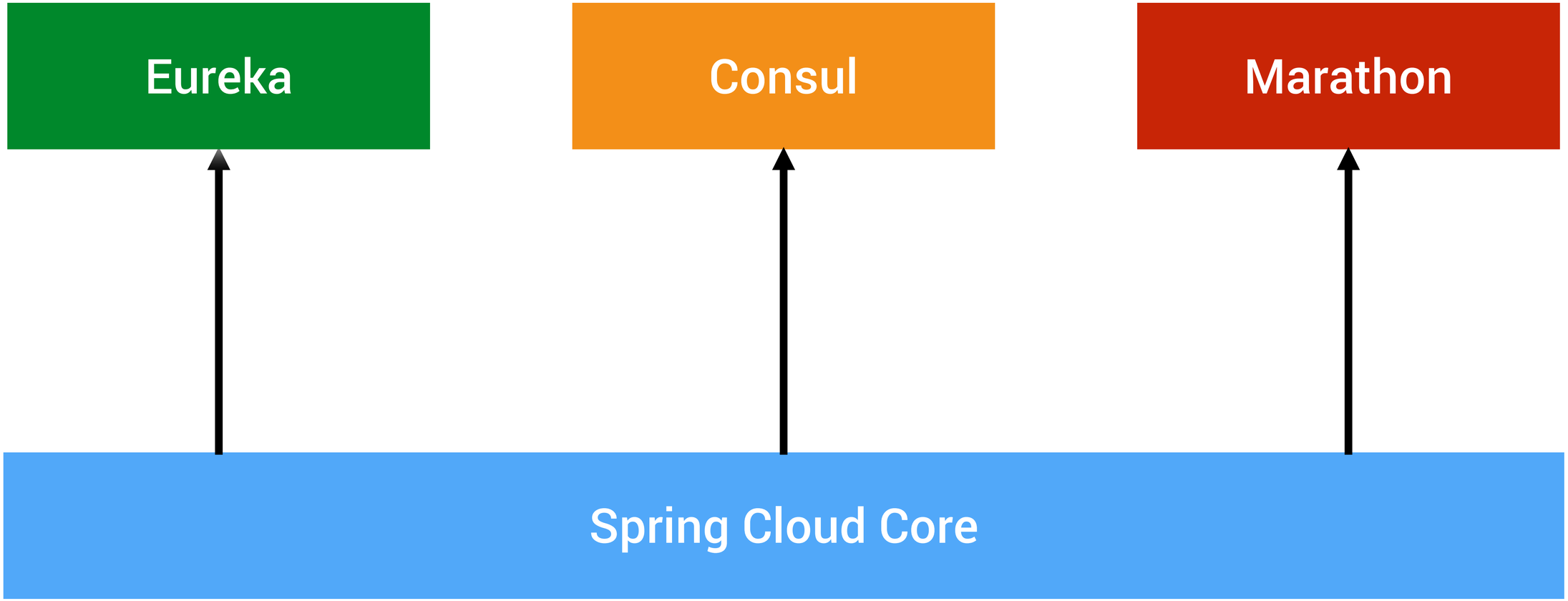

Leader
Master
Master
heartbeat
Service A
Service B
Service C

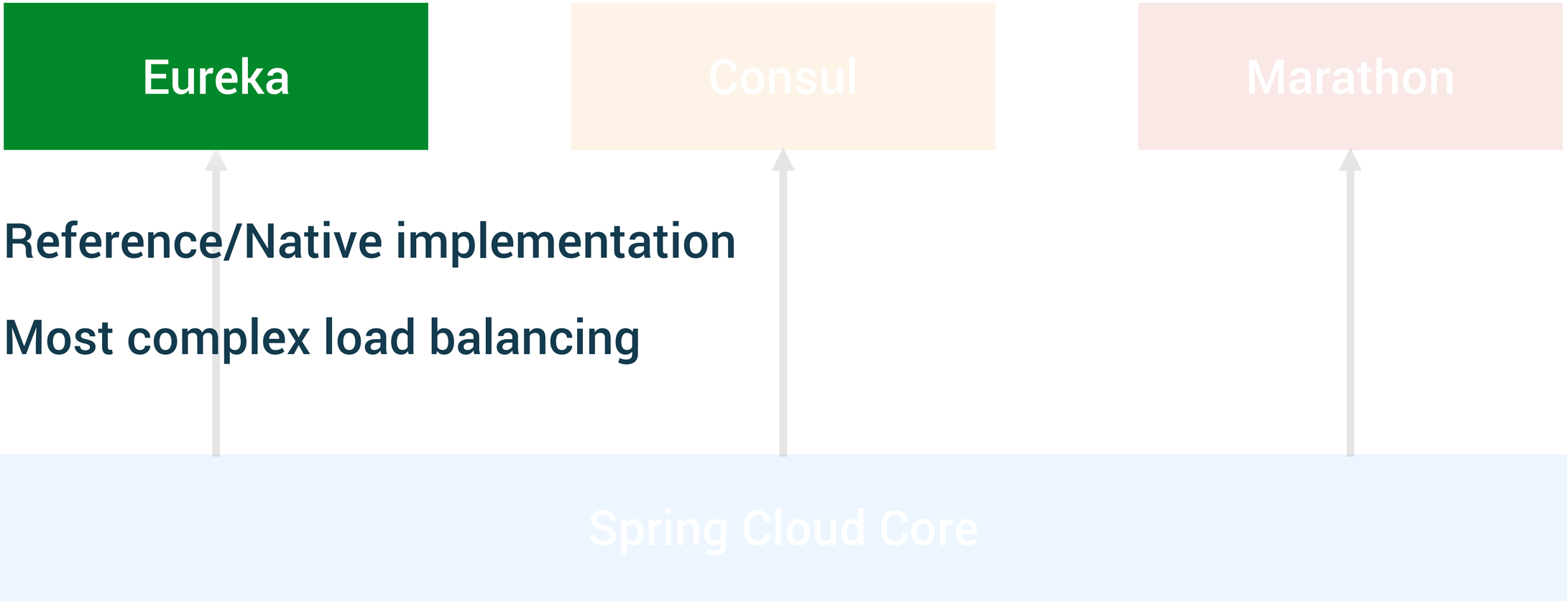

Leader
Master
Master
health check
Service A
Service B
Service C

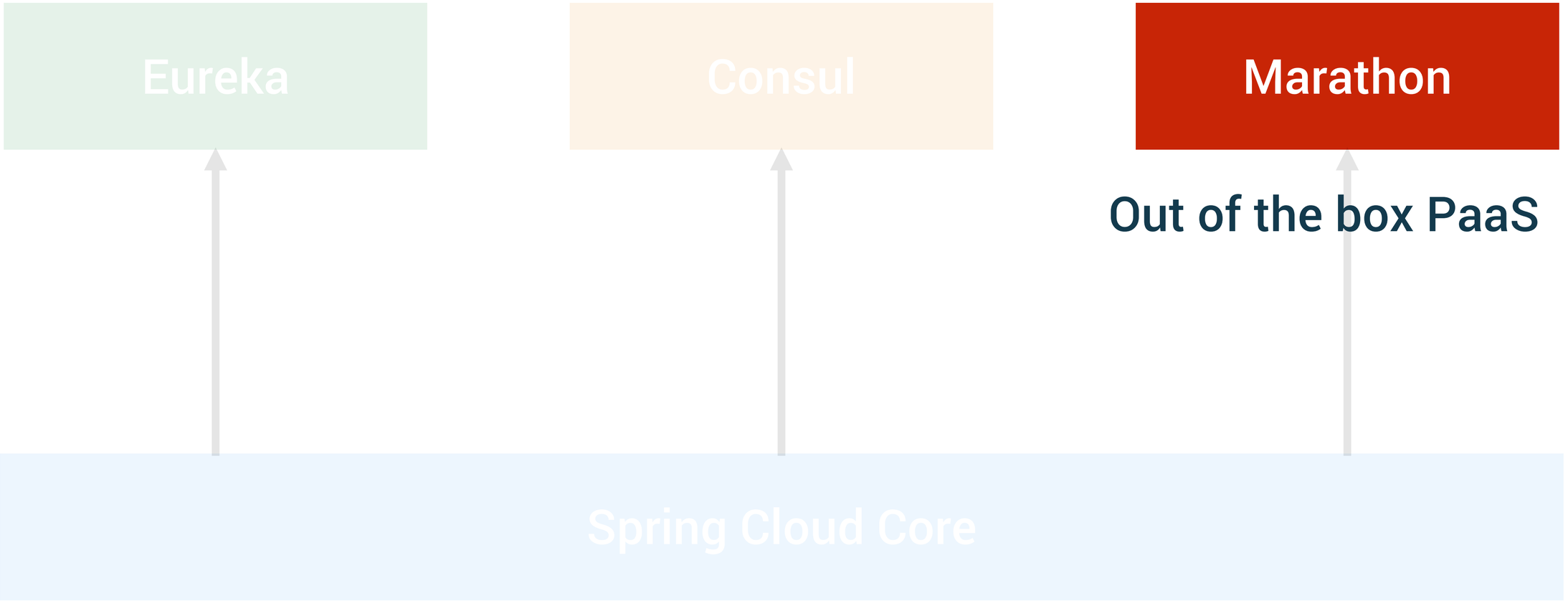

Leader
Master
Master
health check
Service A
Service B
Service C


Leader
Master
Master
deregister
Service A
Service B
Service C


Leader
Master
Master
deregister
Service A
Service B
Service C
graceful shutdown


Eureka
Consul
Marathon
Spring Cloud Core


Eureka
Consul
Marathon
Reference/Native implementation
Most complex load balancing
Spring Cloud Core


Eureka
Consul
Marathon
Out of the box PaaS
Spring Cloud Core

a2a-transactions-api

Deploying (7 of 7 instances)

3 Healthy (43%) 4 Unhealthy (57%) 0 Unknown

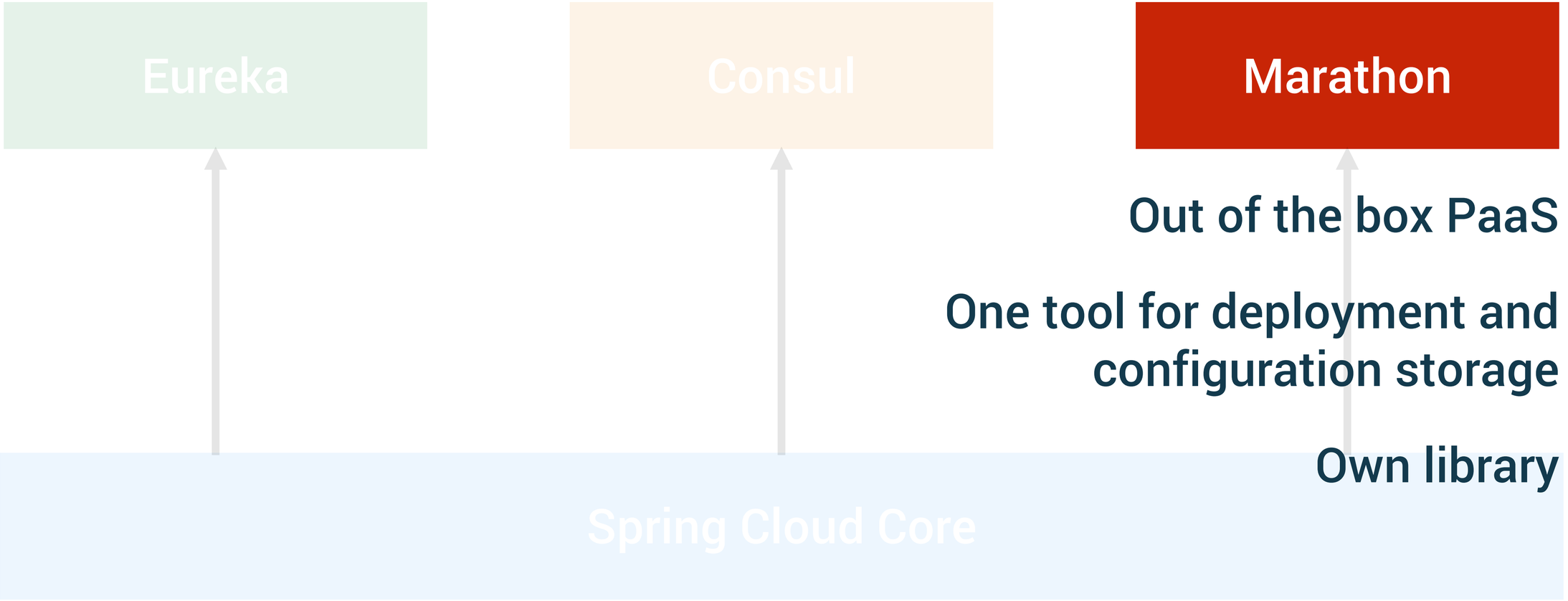
Scale Application | Restart | ⚙

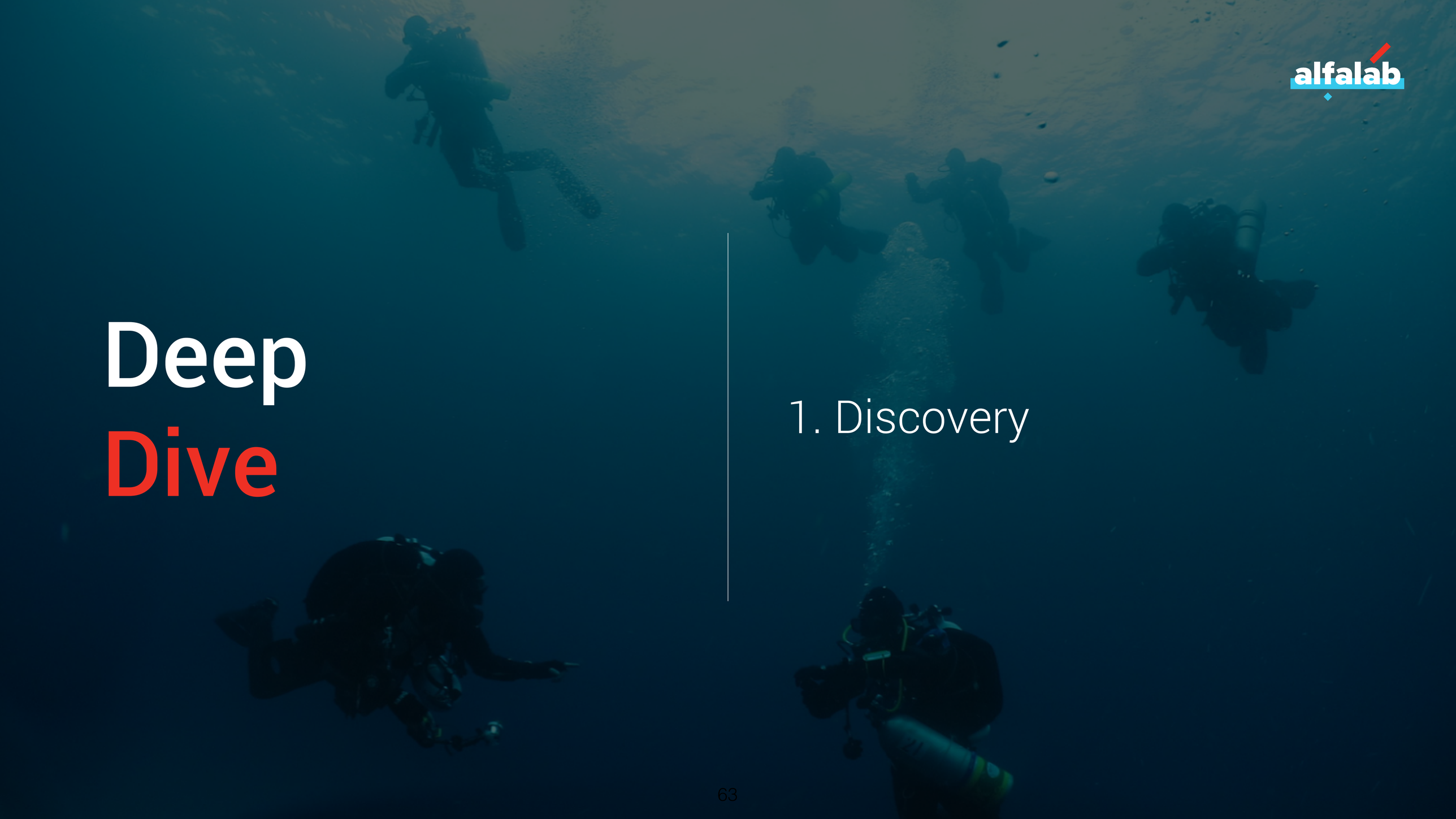
Instances | Configuration | Debug

↻ Refresh

ID	Health	Status	Error Log	Output Log	Version	Updated
retail-online-bank_middle_a2a-transactions-api.7f7bcbd9-892a-11e6-900f-e248e8be00cd abmdev:20757	Healthy	Started	stderr	stdout	17 days ago	03.10.2016, 8:30:33
retail-online-bank_middle_a2a-transactions-api.6a0d09b5-8cb4-11e6-900f-e248e8be00cd abmdev:20204	Healthy	Started	stderr	stdout	a few seconds ago	07.10.2016, 20:35:21
retail-online-bank_middle_a2a-transactions-api.6eafb6ca-8cb4-11e6-900f-e248e8be00cd abmdev:20099	Unhealthy	Started	stderr	stdout	a few seconds ago	07.10.2016, 20:35:28
retail-online-bank_middle_a2a-transactions-api.6eafb6c9-8cb4-11e6-900f-e248e8be00cd abmdev:20697	Unhealthy	Started	stderr	stdout	a few seconds ago	07.10.2016, 20:35:30
retail-online-bank_middle_a2a-transactions-api.6eafb6c8-8cb4-11e6-900f-e248e8be00cd abmdev:20688	Healthy	Started	stderr	stdout	a few seconds ago	07.10.2016, 20:35:31
retail-online-bank_middle_a2a-transactions-api.6eaf8fb7-8cb4-11e6-900f-e248e8be00cd abmdev:20175	Unhealthy	Started	stderr	stdout	a few seconds ago	07.10.2016, 20:35:32
retail-online-bank_middle_a2a-transactions-api.6eaf8fb6-8cb4-11e6-900f-e248e8be00cd abmdev:20494	Unhealthy	Started	stderr	stdout	a few seconds ago	07.10.2016, 20:35:33

```
{
  "host": "server1",
  "ports": [
    20688
  ],
  "ipAddresses": [
    {
      "ipAddress": "172.17.0.21",
      "protocol": "IPv4"
    }
  ],
  "appId": "/retail-online-bank/middle/a2a-transactions-api",
  "healthCheckResults": [
    {
      "alive": true
    }
  ]
}
```


Eureka
Consul
Marathon
Out of the box PaaS
One tool for deployment and configuration storage
Own library
Spring Cloud Core



Deep Dive

1. Discovery

```
...
@EnableDiscoveryClient
...
public class Application {
  @Autowired
  private DiscoveryClient discoveryClient;

  public List<String> services() {
    return discoveryClient.getServices();
  }
}
```

```
...
@EnableDiscoveryClient
...
public class Application {
  @Autowired
  private DiscoveryClient discoveryClient;

  public List<String> services() {
    return discoveryClient.getServices();
  }
}
```

```
public interface DiscoveryClient {

 public ServiceInstance getLocalServiceInstance();

 public List<ServiceInstance> getInstances(String serviceId);

 public List<String> getServices();

}
```

```
public interface DiscoveryClient {

 public ServiceInstance getLocalServiceInstance();

 public List<ServiceInstance> getInstances(String serviceId);

 public List<String> getServices();

}
```

```
public interface DiscoveryClient {

 public ServiceInstance getLocalServiceInstance();

 public List<ServiceInstance> getInstances(String serviceId);

 public List<String> getServices();

}
```

Eureka
Implementation

```
eureka:
 client:
  serviceUrl:
   defaultZone: ${ZONE1_EUREKA_URL}
```

serviceId -> virtualHostName

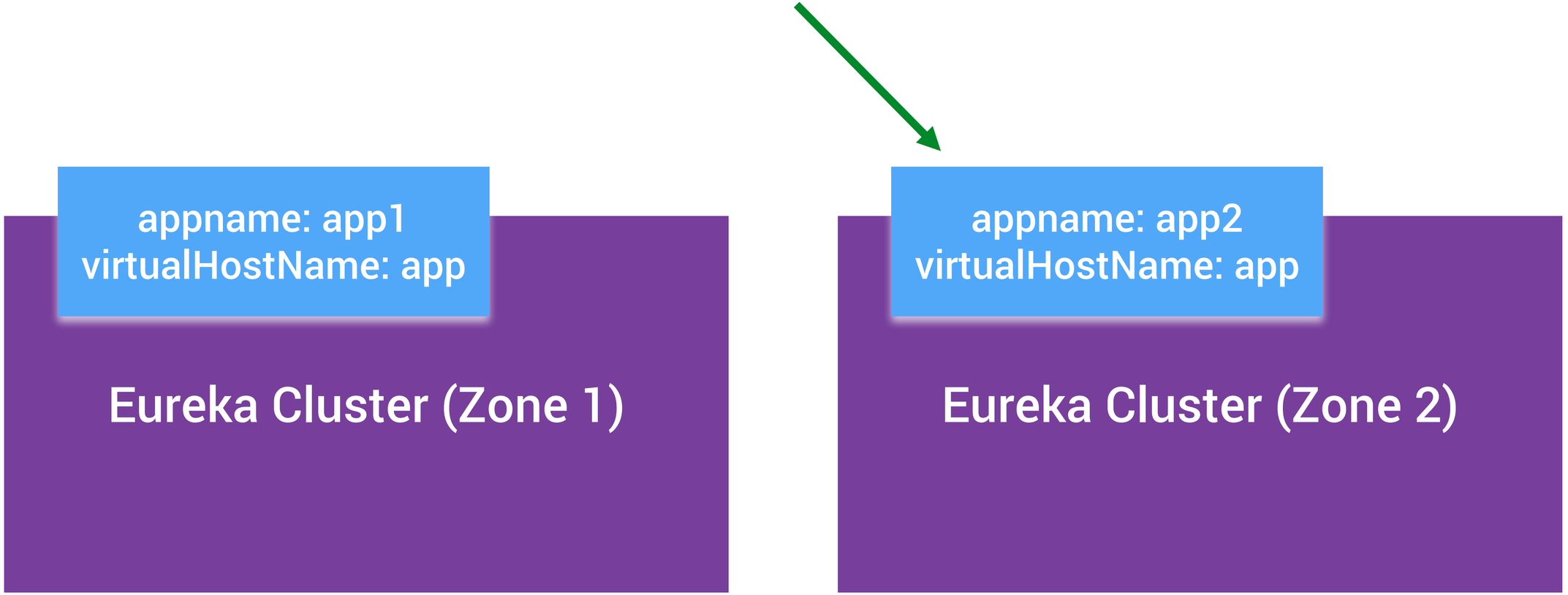
appname: app1
virtualHostName: app

Eureka Cluster (Zone 1)

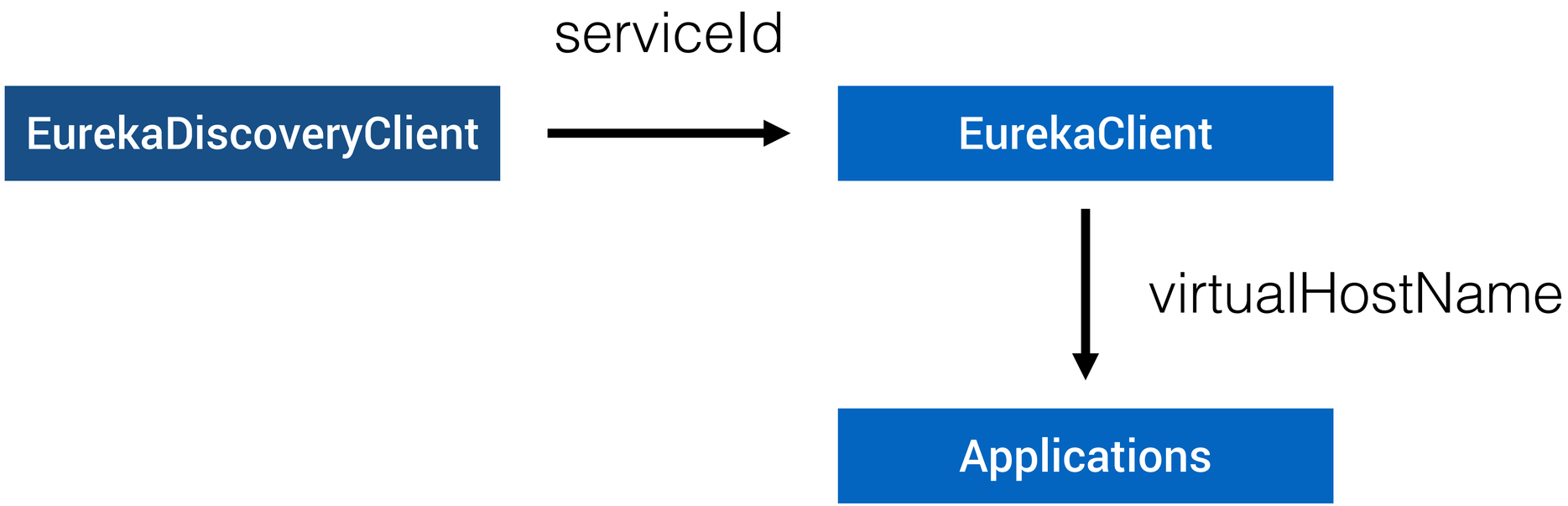
appname: app2
virtualHostName: app

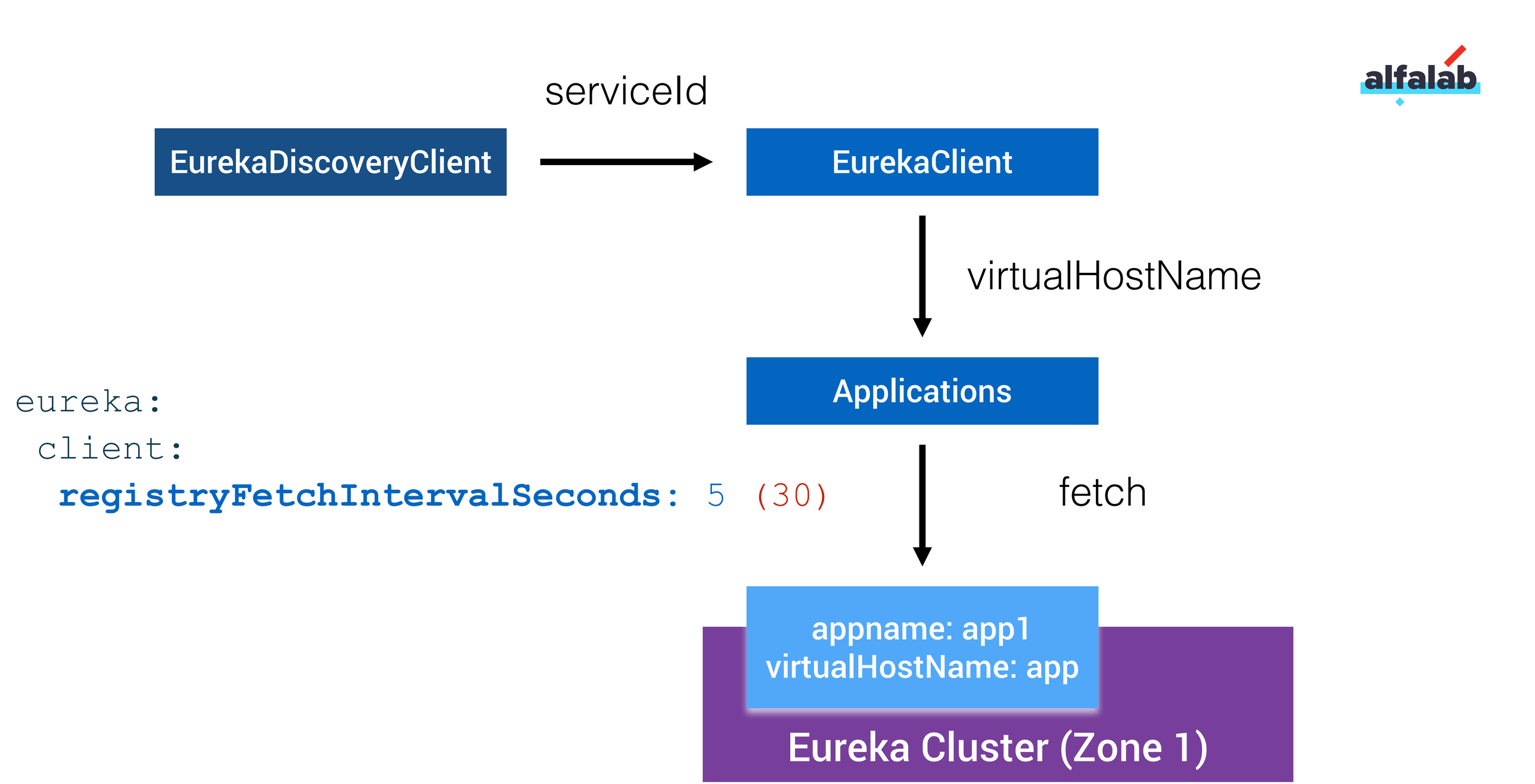
Eureka Cluster (Zone 2)

```
eureka:
 client:
  serviceUrl:
   defaultZone: ${ZONE1_EUREKA_URL}
   zone2: ${ZONE2_EUREKA_URL}
  availabilityZones: zone1,zone2
```

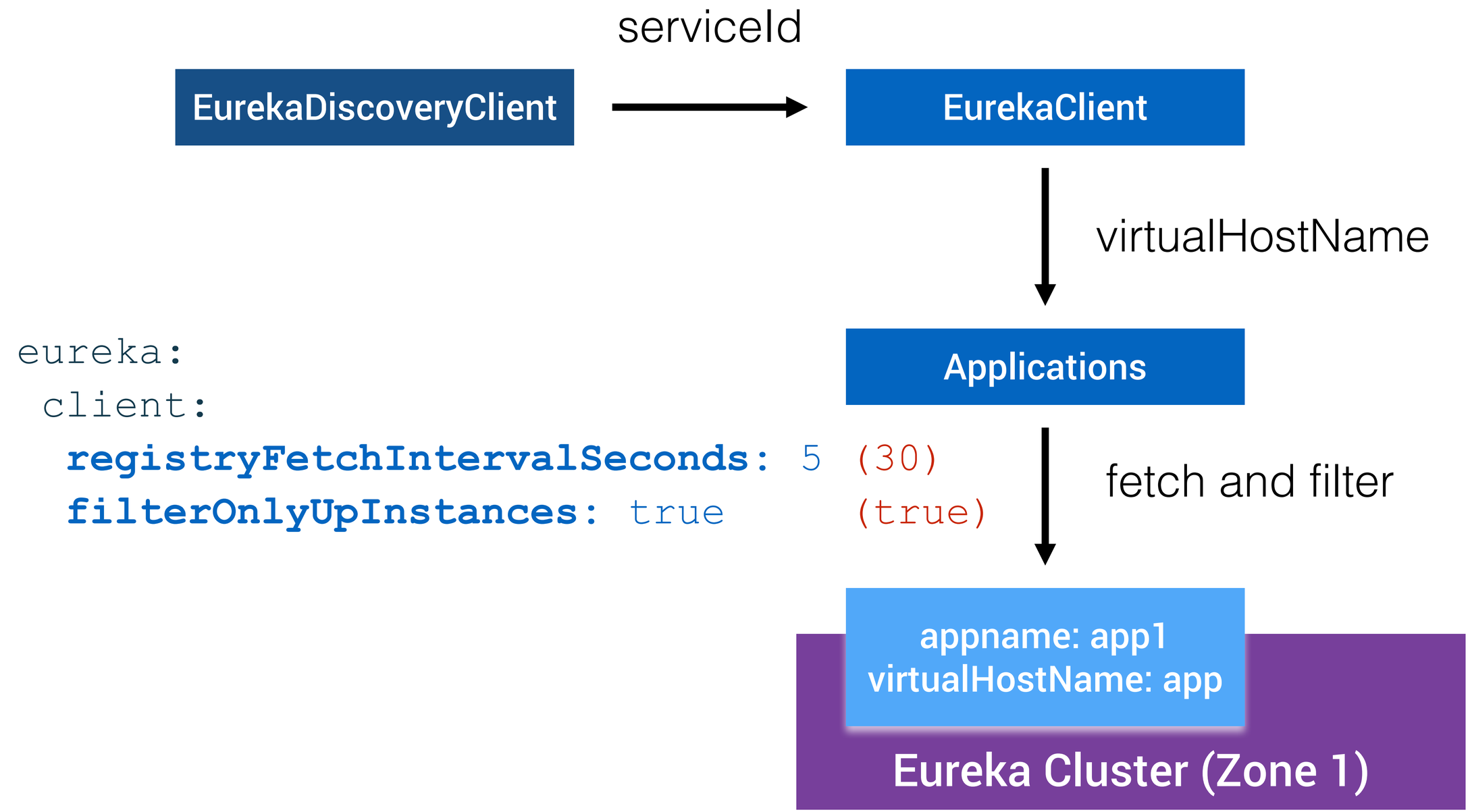



serviceId
EurekaDiscoveryClient
EurekaClient


serviceId
EurekaDiscoveryClient
EurekaClient
virtualHostName
Applications




```
eureka:
 client:
  registryFetchIntervalSeconds: 5 (30)
```




```
eureka:
 client:
  registryFetchIntervalSeconds: 5  (30)
  filterOnlyUpInstances: true      (true)
```

Marathon Implementation

```
spring:
 cloud:
  marathon:
   host: ${MARATHON_HOST}
   port: ${MARATHON_PORT}
```

serviceId -> appId

taskId: {random}
appId: app1

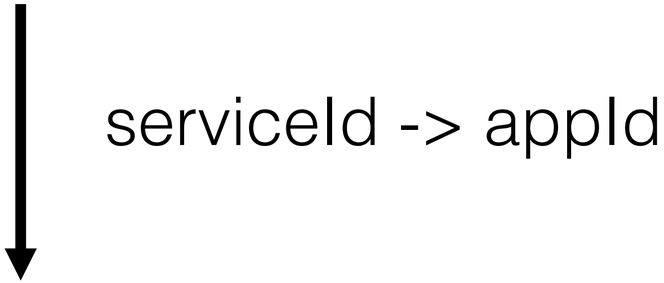
Mesos Cluster (DC 1)

taskId: {random}
appId: app1

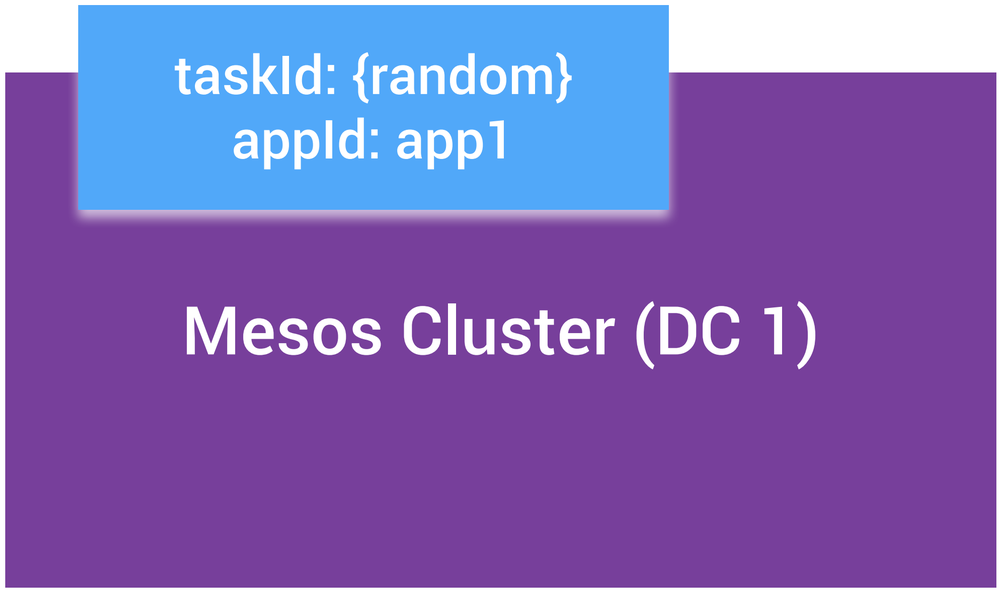
Mesos Cluster (DC 2)

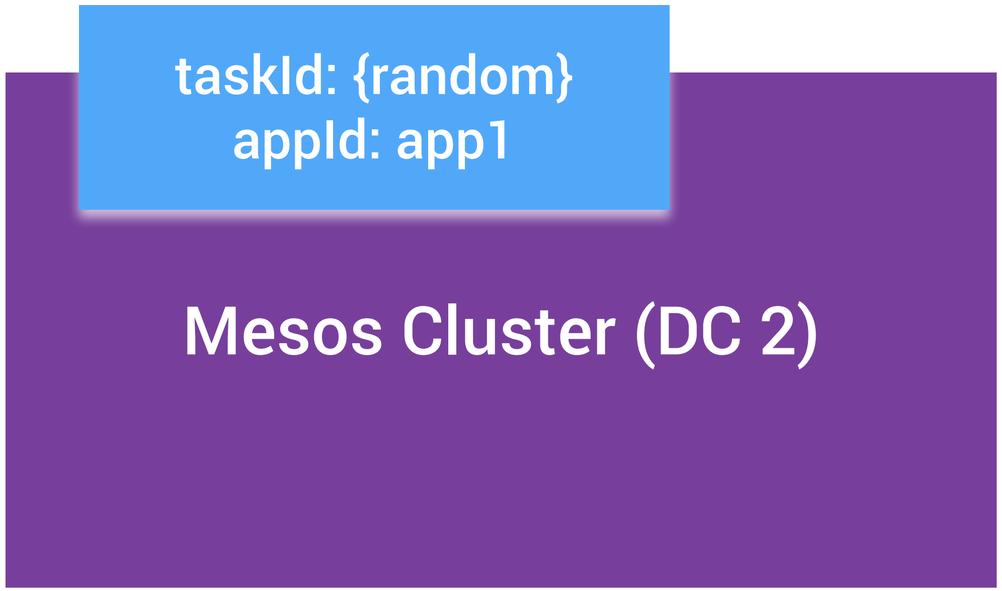
```
spring:
 cloud:
  marathon:
   host: ${MARATHON_HOST}
   port: ${MARATHON_PORT}
```

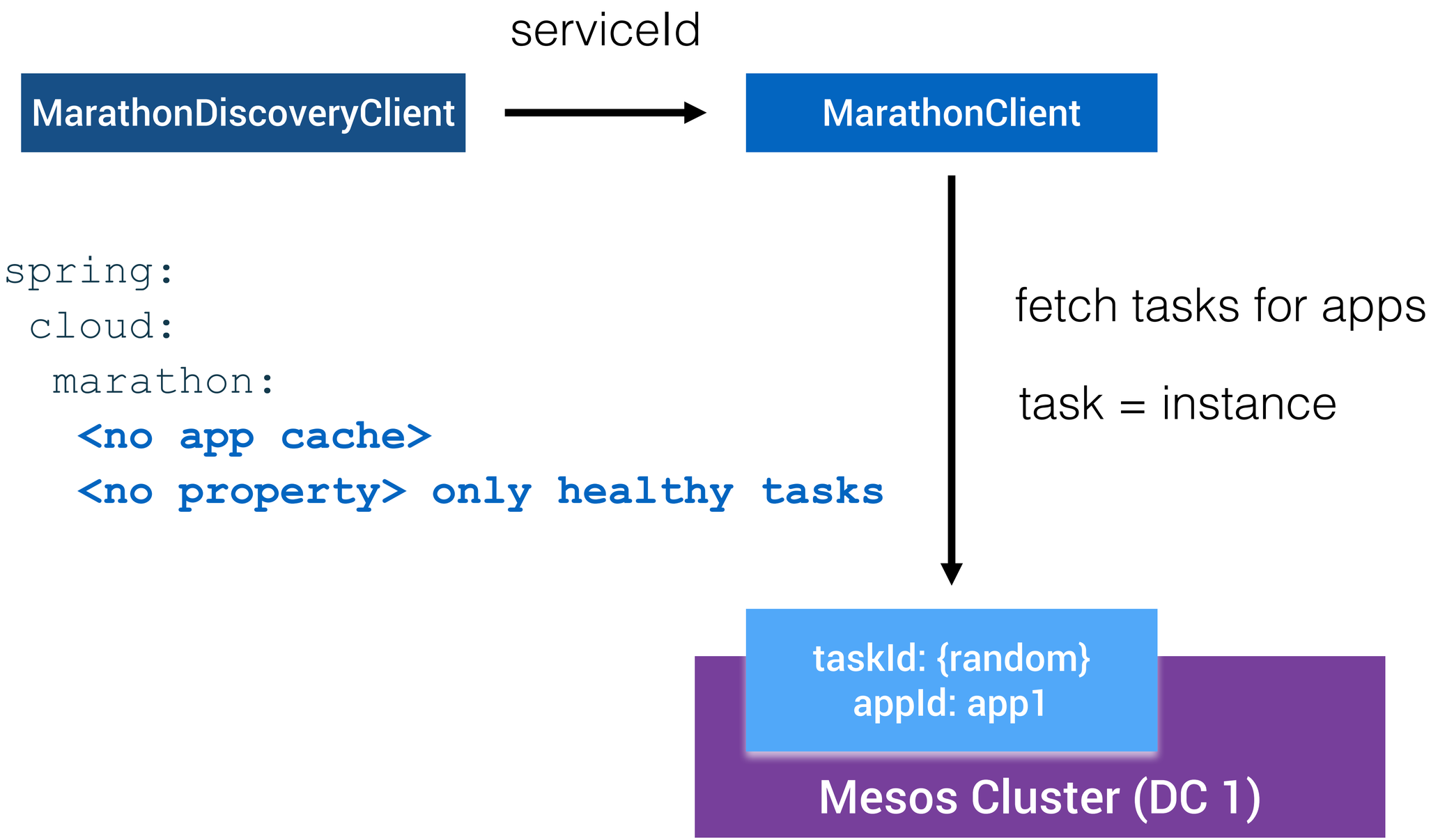
<- - - - there is no failover to another dc



serviceId -> appId






serviceId
MarathonDiscoveryClient
MarathonClient
spring:
cloud:
marathon:
<no app cache>
<no property> only healthy tasks
fetch tasks for apps
task = instance
taskId: {random}
appId: app1
Mesos Cluster (DC 1)


alfalab
Is it success?

```
@Autowired
private LoadBalancerClient loadBalancer;

@RequestMapping("/instance")
public ServiceInstance instance() {
    return loadBalancer.choose(serviceId);
}
```

```
@Autowired
private LoadBalancerClient loadBalancer;

@RequestMapping("/instance")
public ServiceInstance instance() {
    return loadBalancer.choose(serviceId);
}
```

?

```
@Autowired
private LoadBalancerClient loadBalancer;

@RequestMapping("/instance")
public ServiceInstance instance() {
    return loadBalancer.choose(serviceId);
}
```

1. ServiceInstance **2. null** **3. Error**

```
@Autowired
private LoadBalancerClient loadBalancer;

@RequestMapping("/instance")
public ServiceInstance instance() {
    return loadBalancer.choose(serviceId);
}
```

1. ServiceInstance **2. null** **3. Error**

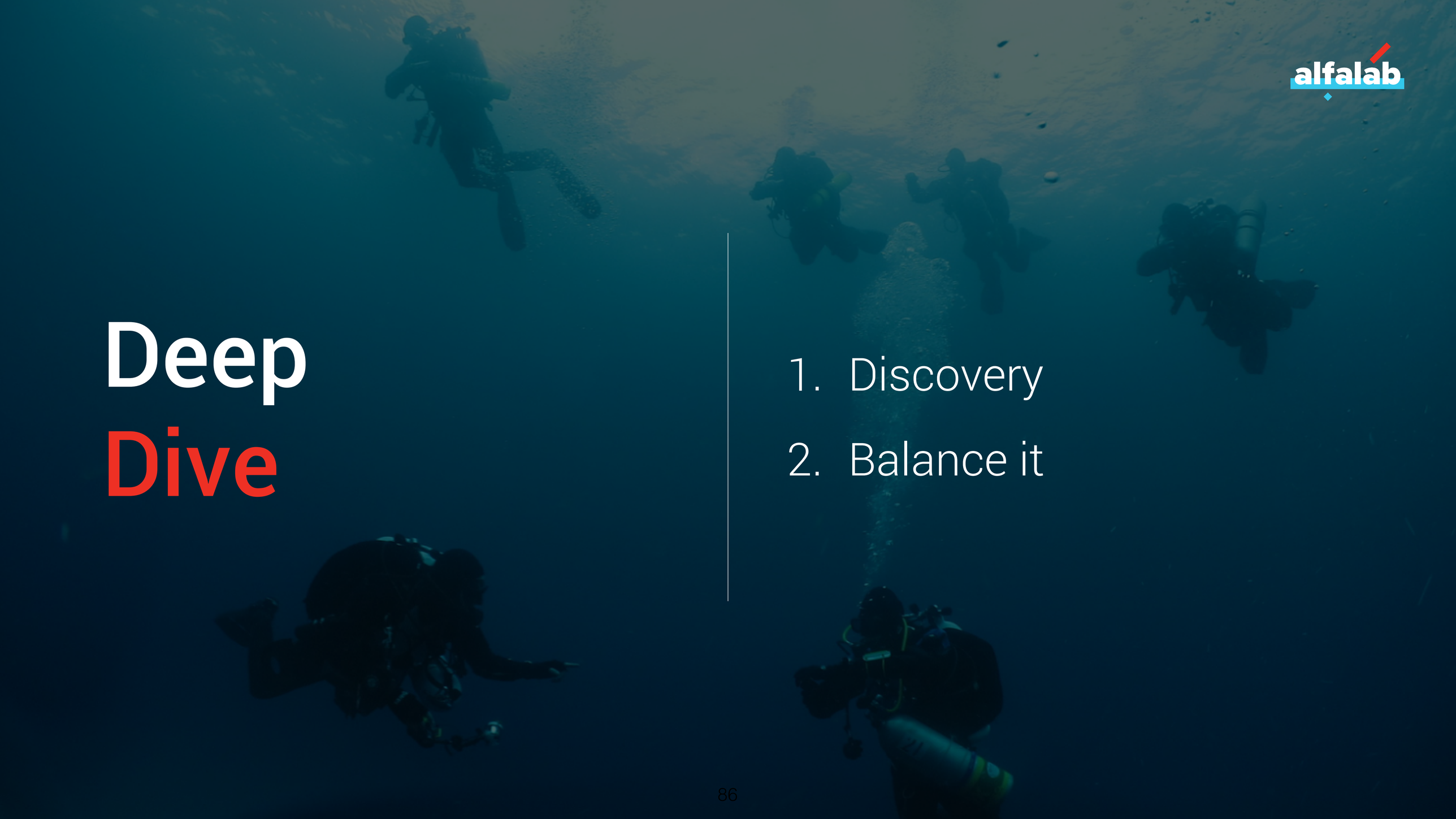
Where it is used?

Hypermedia Links

Dynamic Routes for Edge Server

DiscoveryClient health check in actuator is based on it

Less than you expect, right?



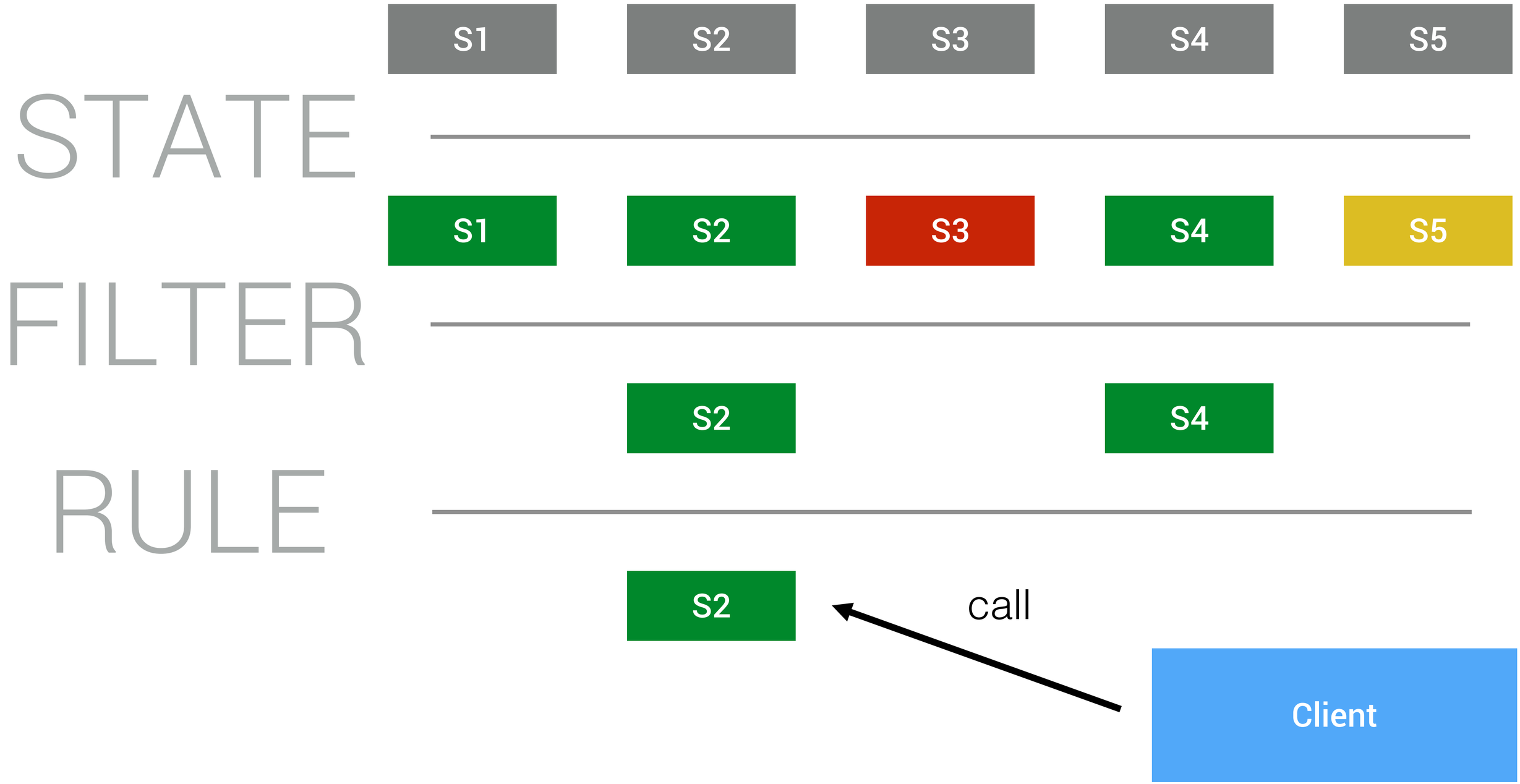
Deep Dive

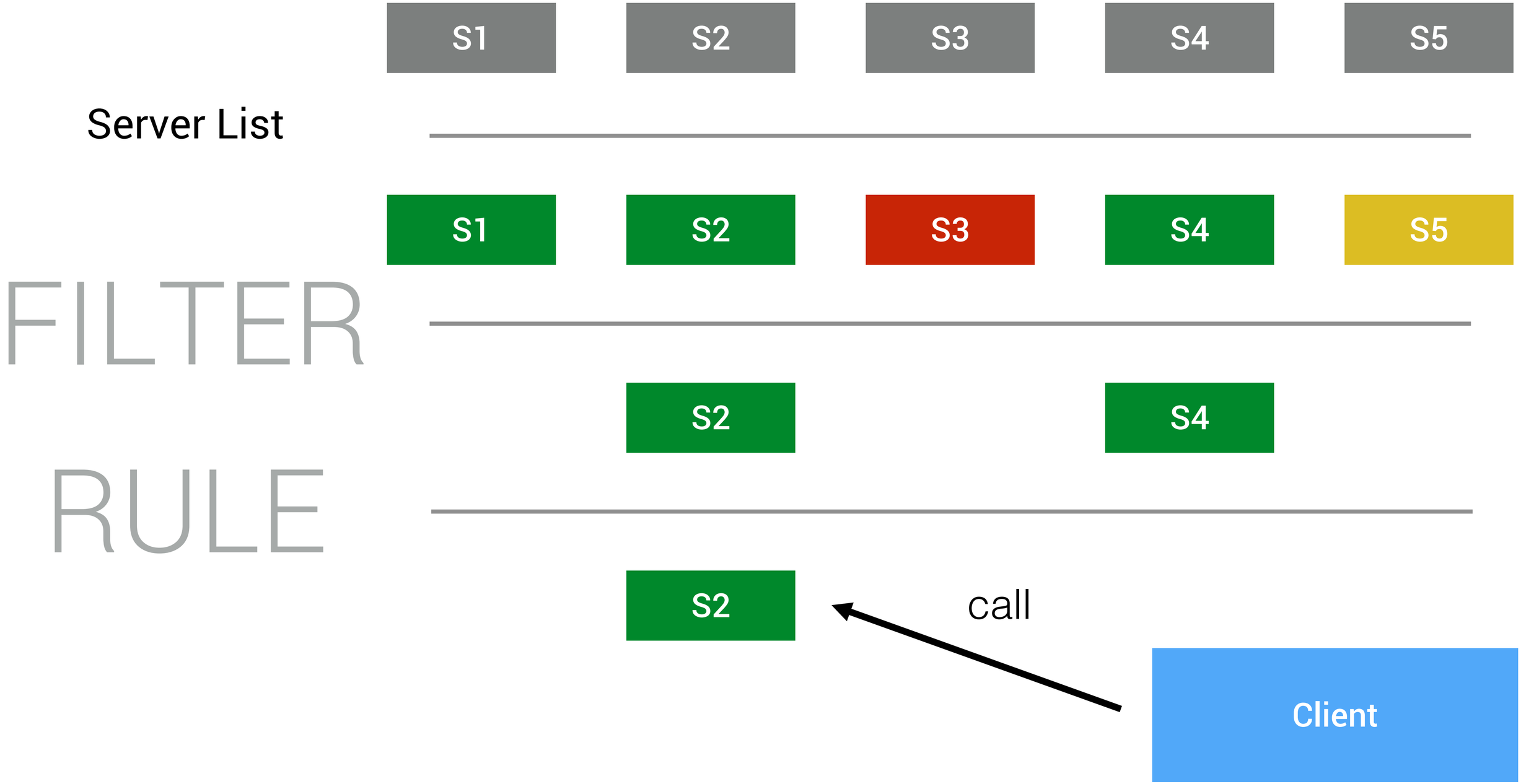
1. Discovery
2. Balance it

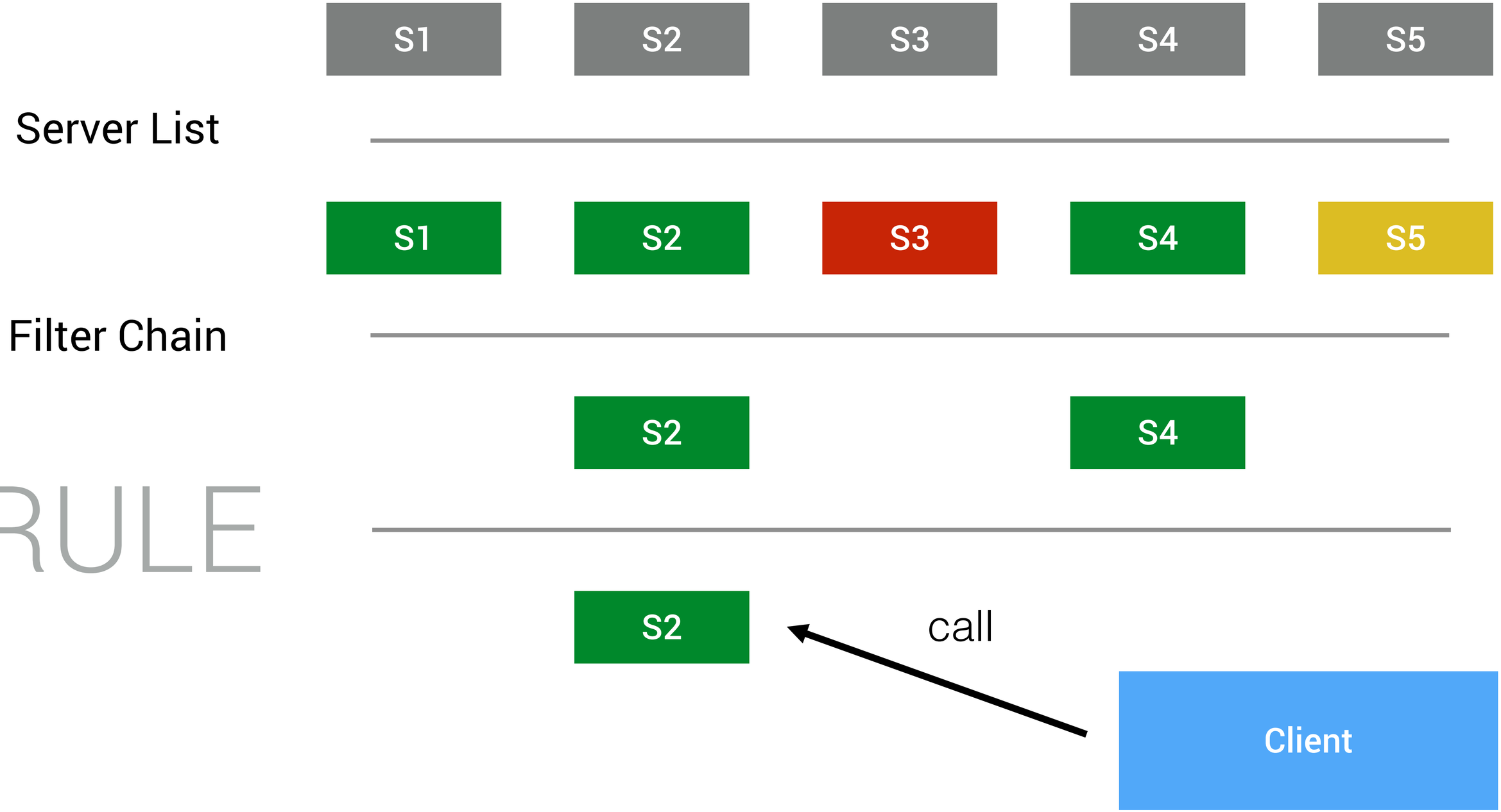
Load Balancer
Client

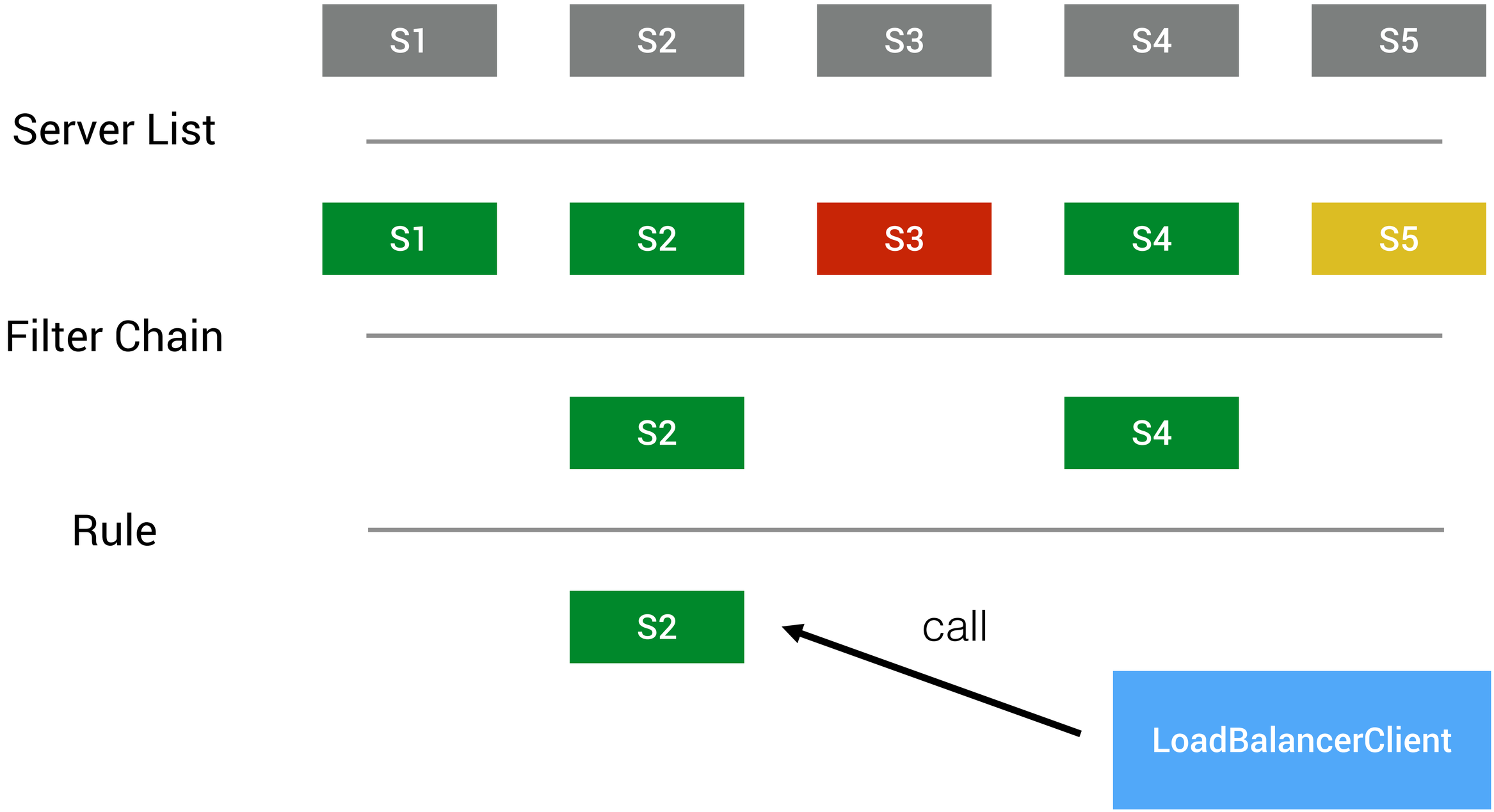
Based on Netflix Ribbon

Ribbon may be used with or without service registry


S1
S2
S3
S4
S5
STATE
S1
S2
S3
S4
S5
FILTER
S2
S4
RULE
S2
call
Client


S1
S2
S3
S4
S5
Server List
S1
S2
S3
S4
S5
FILTER
S2
S4
RULE
S2
call
Client


S1
S2
S3
S4
S5
Server List
S1
S2
S3
S4
S5
Filter Chain
S2
S4
RULE
S2
call
Client


S1
S2
S3
S4
S5
Server List
S1
S2
S3
S4
S5
Filter Chain
S2
S4
Rule
S2
call
LoadBalancerClient

Ribbon
Server List

Static

Configuration Based

Discovery Based

Spring Cloud
Ribbon Configure

Bean Lifecycle

Spring native configuration

ConfigurationBasedServerList

alfalab

```
test-service:                                          #service name
  ribbon:                                              #namespace
    listOfServers: host:8080,anotherHost:8081
```

ConfigurationBasedServerList

```
test-service:                                          #service name
  ribbon:                                              #namespace
    listOfServers: host:8080,anotherHost:8081
```

```
public List<Server> getListOfServers() {
  return derive(
    clientConfig.get(CommonClientConfigKey.ListOfServers)
  );
}
```

```
@Autowired
private LoadBalancerClient loadBalancer;

@RequestMapping("/url")
public String realUrl() throws IOException {
    return loadBalancer.reconstructURI(
                    instance,
                    new URI("http://"+ serviceId +"/me")
                ).toString();
}
```

it will be replaced with real host and port

```
@Autowired
private LoadBalancerClient loadBalancer;

@RequestMapping("/url")
public String realUrl() throws IOException {
    return loadBalancer.reconstructURI(
                    instance,
                    new URI("http://"+ serviceId +"/me")
                ).toString();
}
```

```
> curl service:8080/url
http://host:8080/me
```

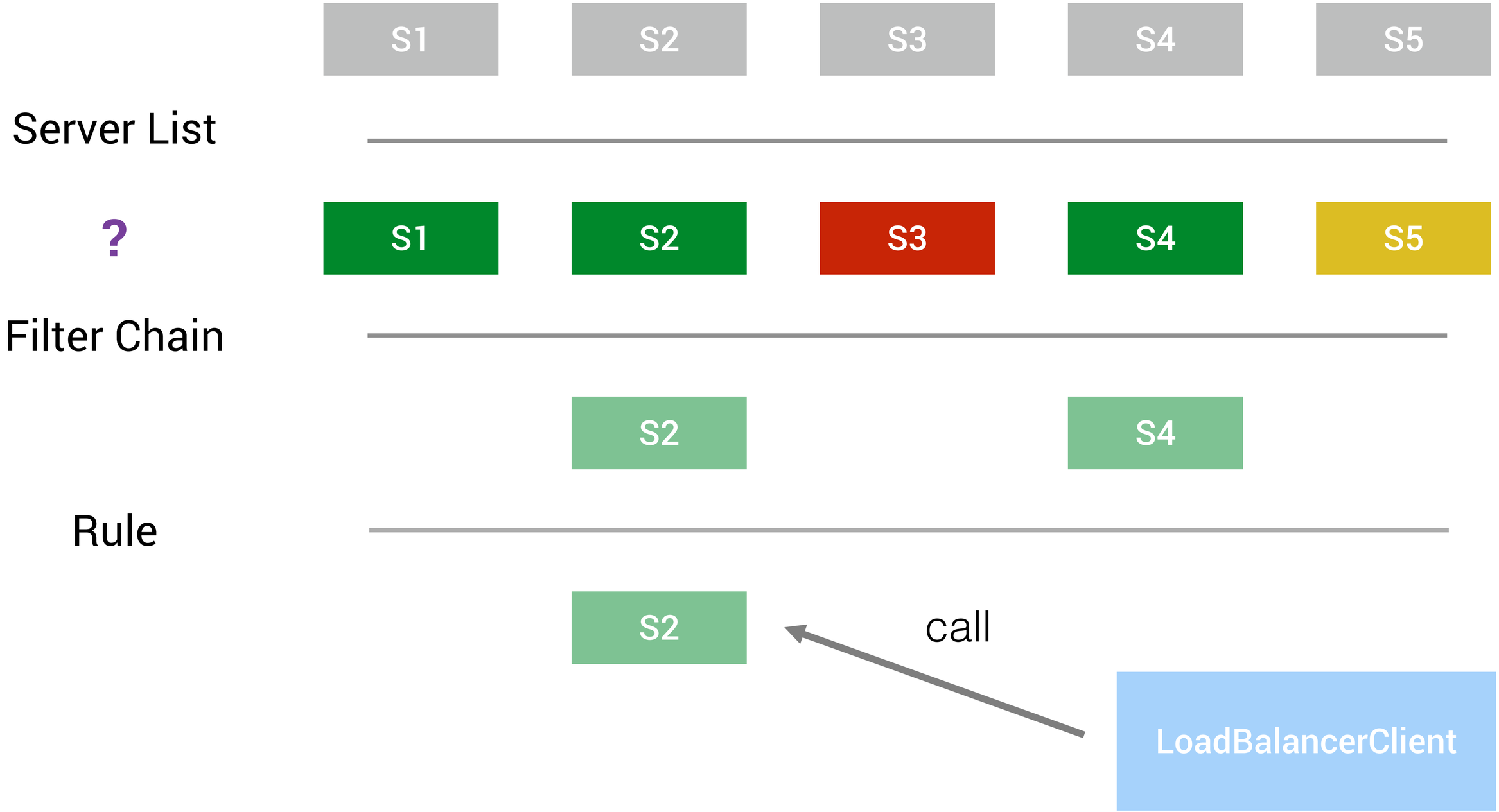
```
@Autowired
private LoadBalancerClient loadBalancer;

@RequestMapping("/url")
public String realUrl() throws IOException {
    return loadBalancer.reconstructURI(
                    instance,
                    new URI("http://"+ serviceId +"/me")
                ).toString();
}
```

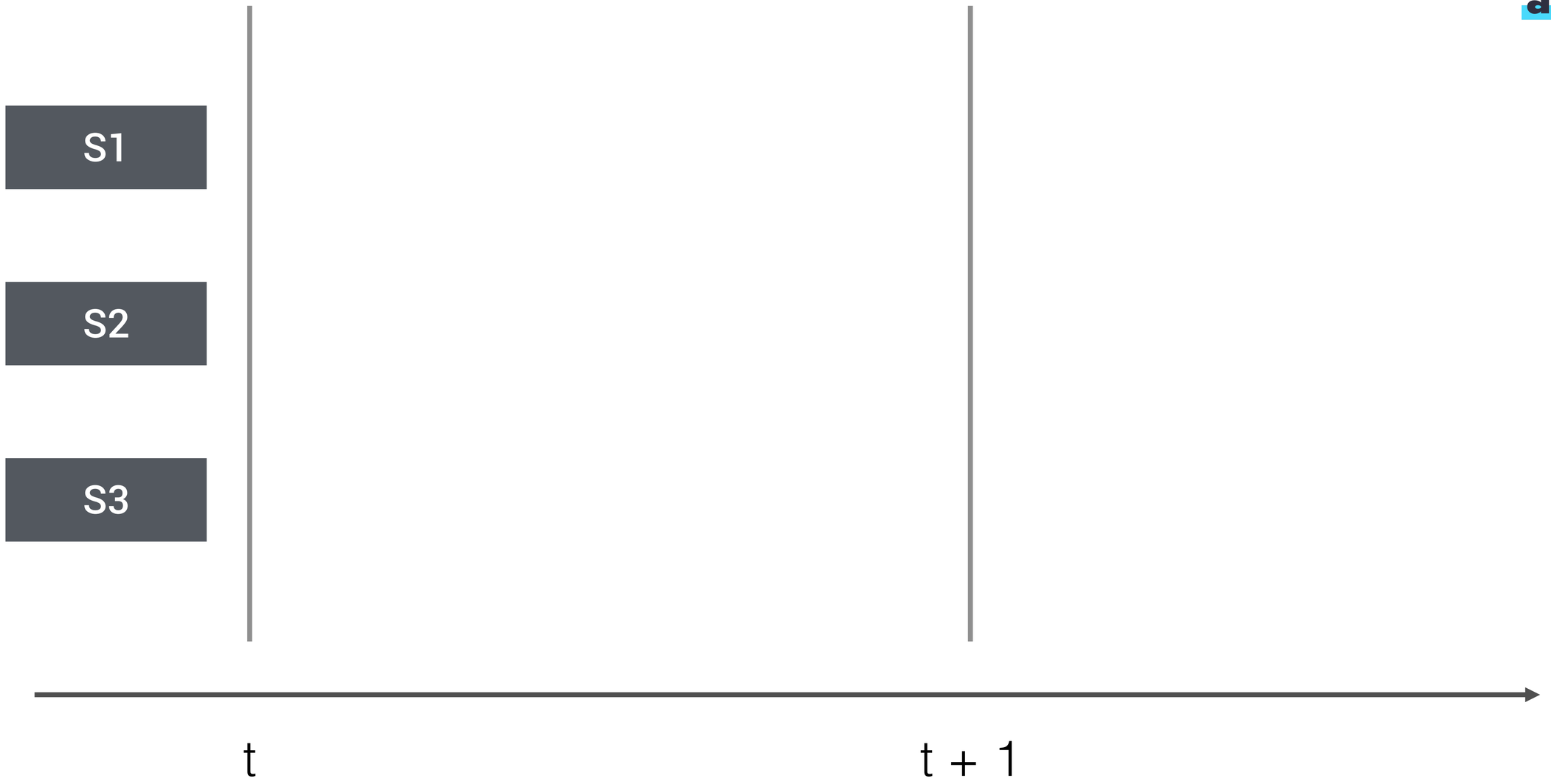
```
> curl service:8080/url
http://anotherHost:8080/me
```

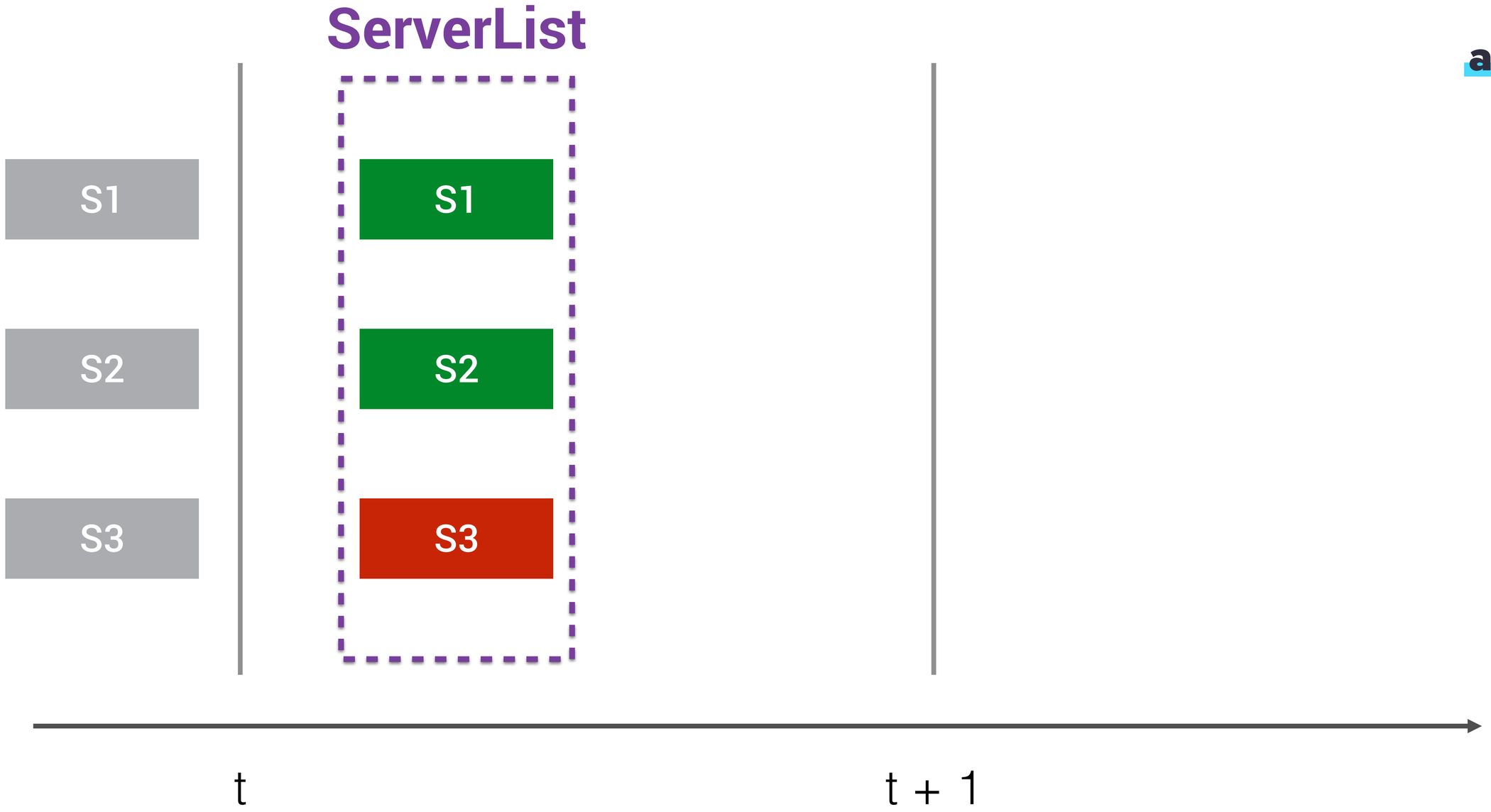
Copy-past Implementation

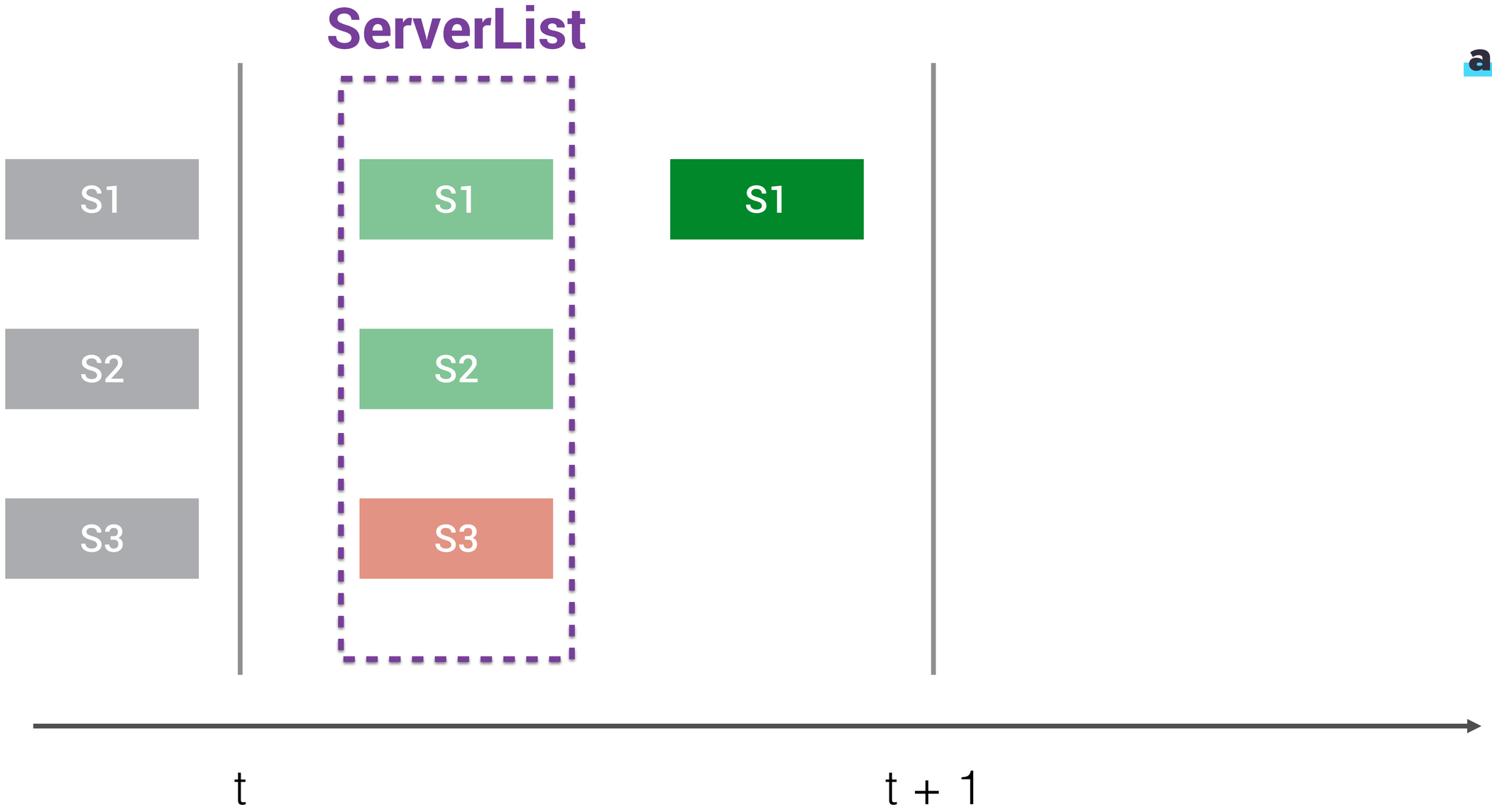
Same as DiscoveryClient

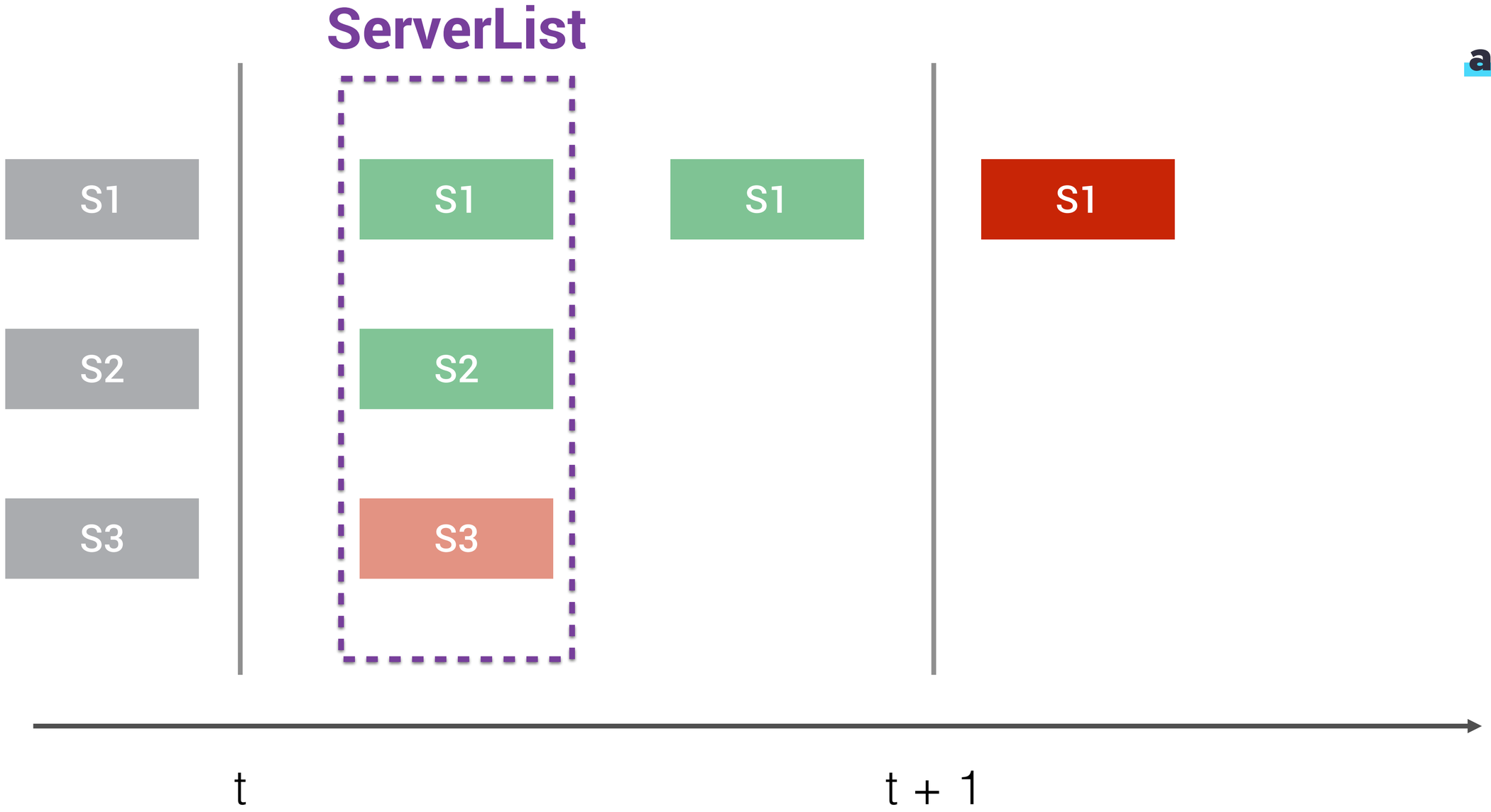
Is our instance alive?

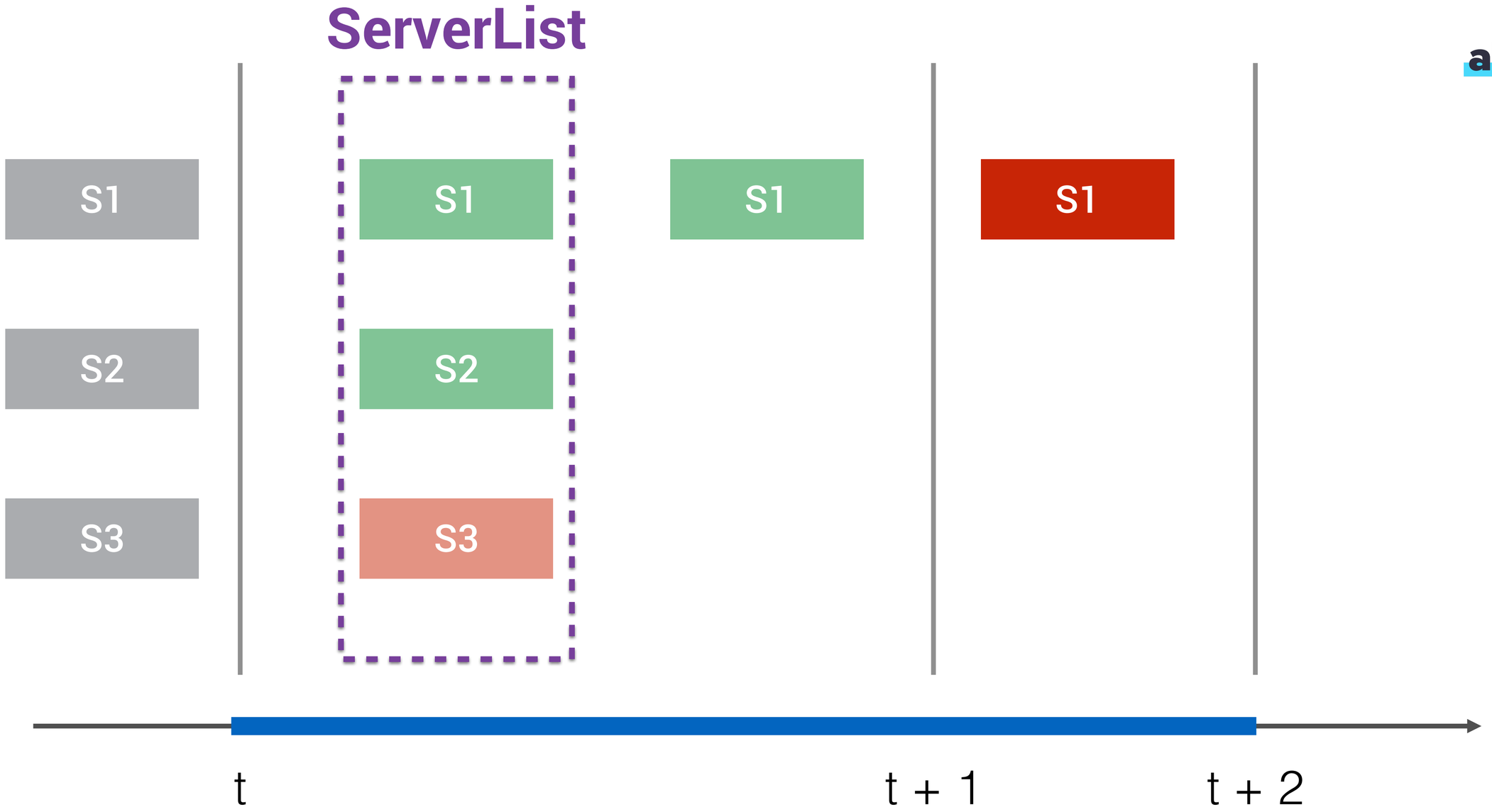

S1
S2
S3
S4
S5
Server List
?
S1
S2
S3
S4
S5
Filter Chain
S2
S4
Rule
S2
call
LoadBalancerClient

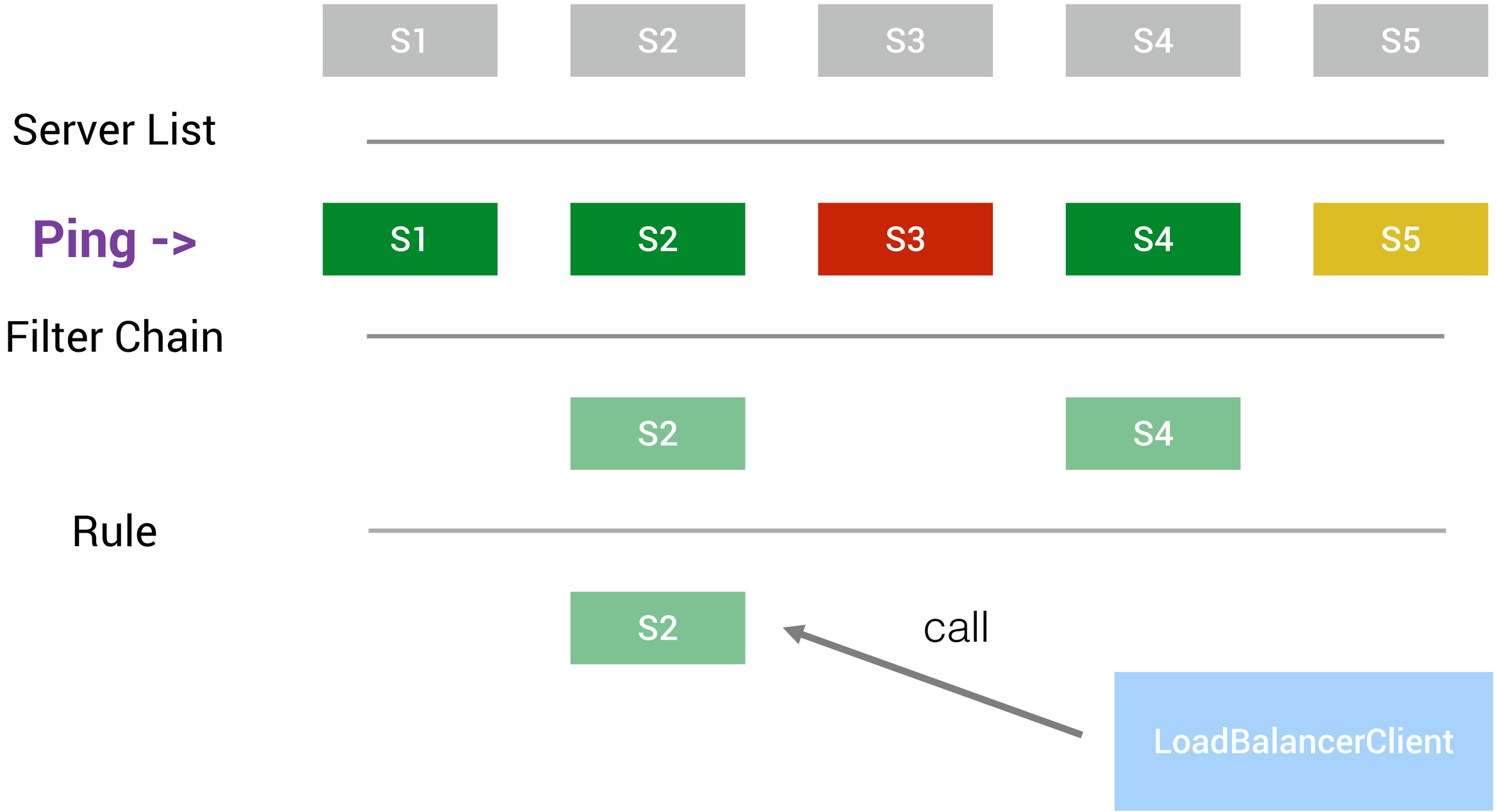

S1
S2
S3
t
t + 1

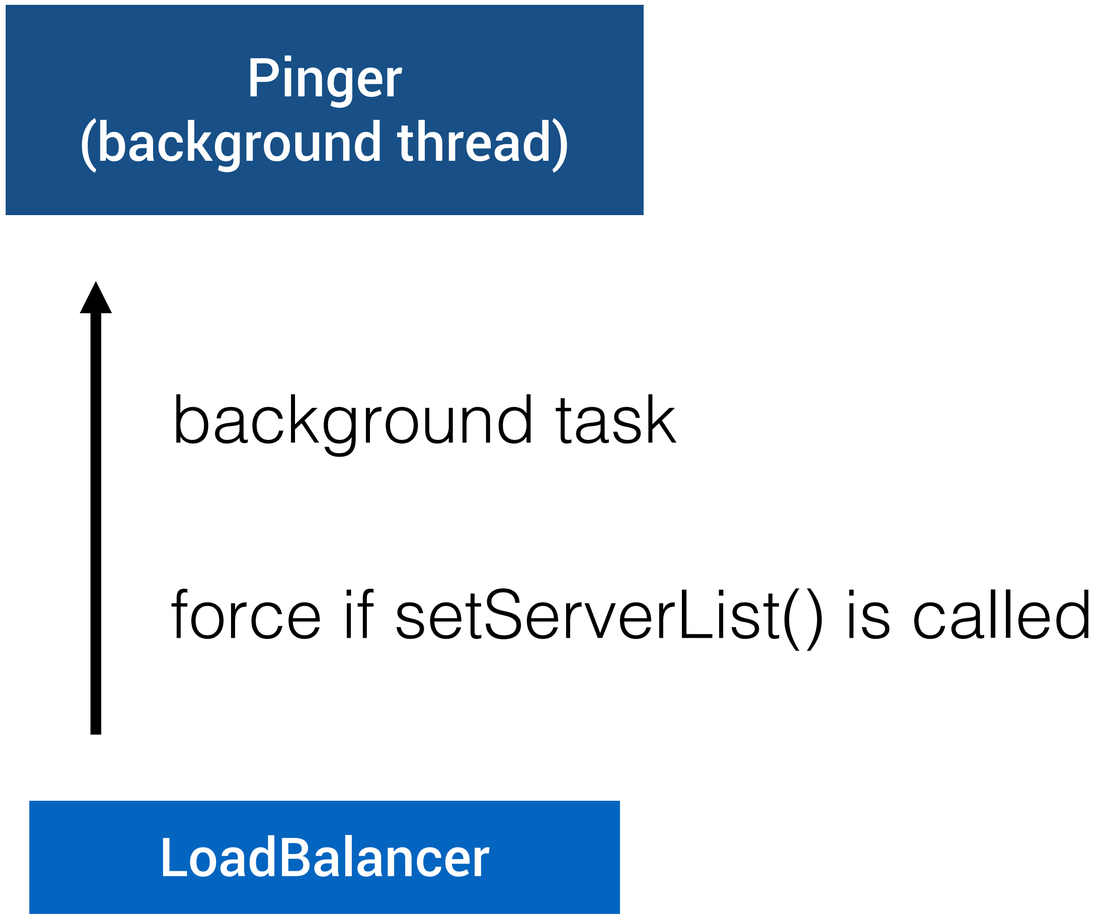

ServerList
S1
S2
S3
S1
S2
S3
t
t + 1

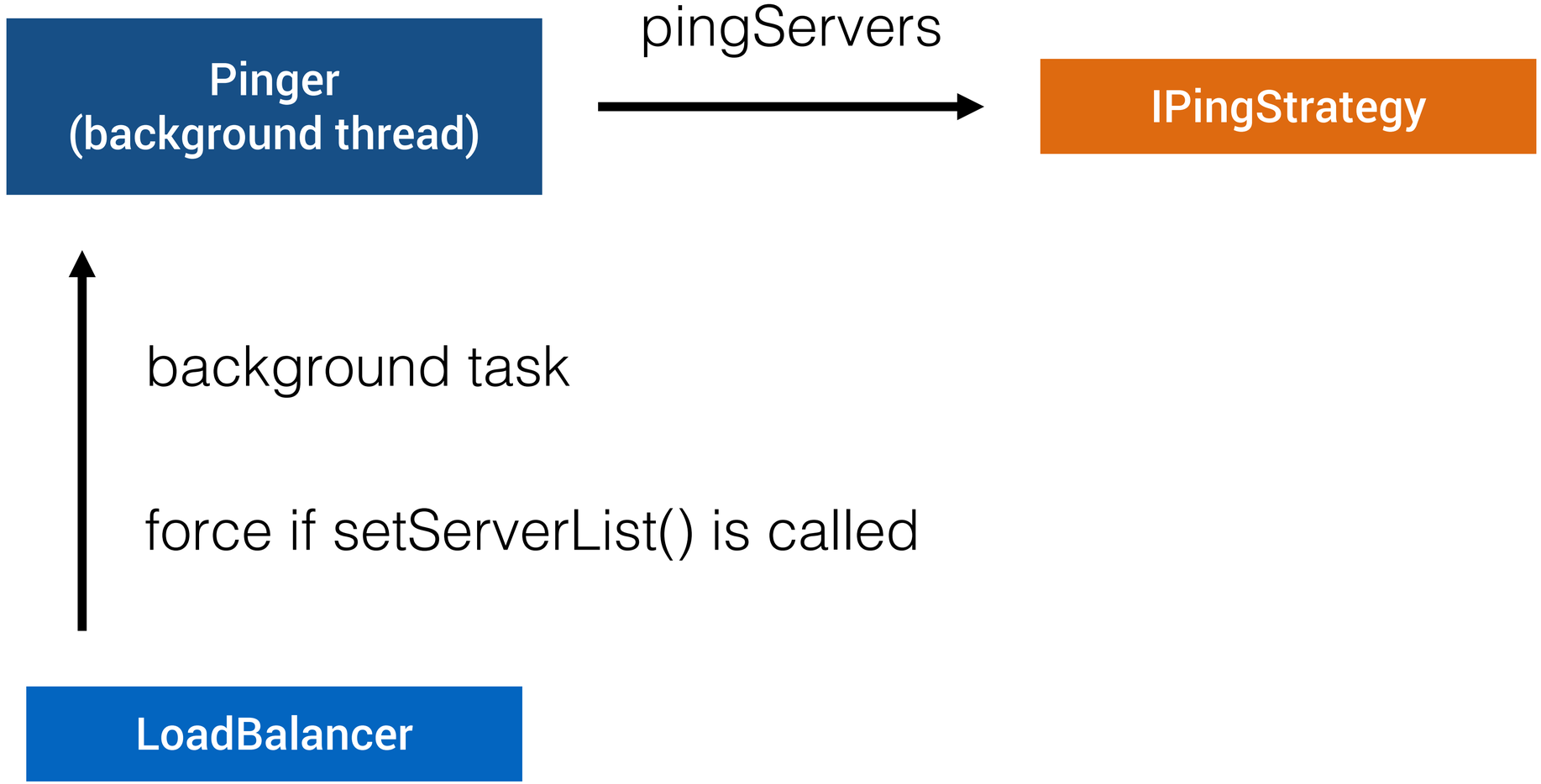

ServerList
S1
S2
S3
S1
S2
S3
S1
t
t + 1

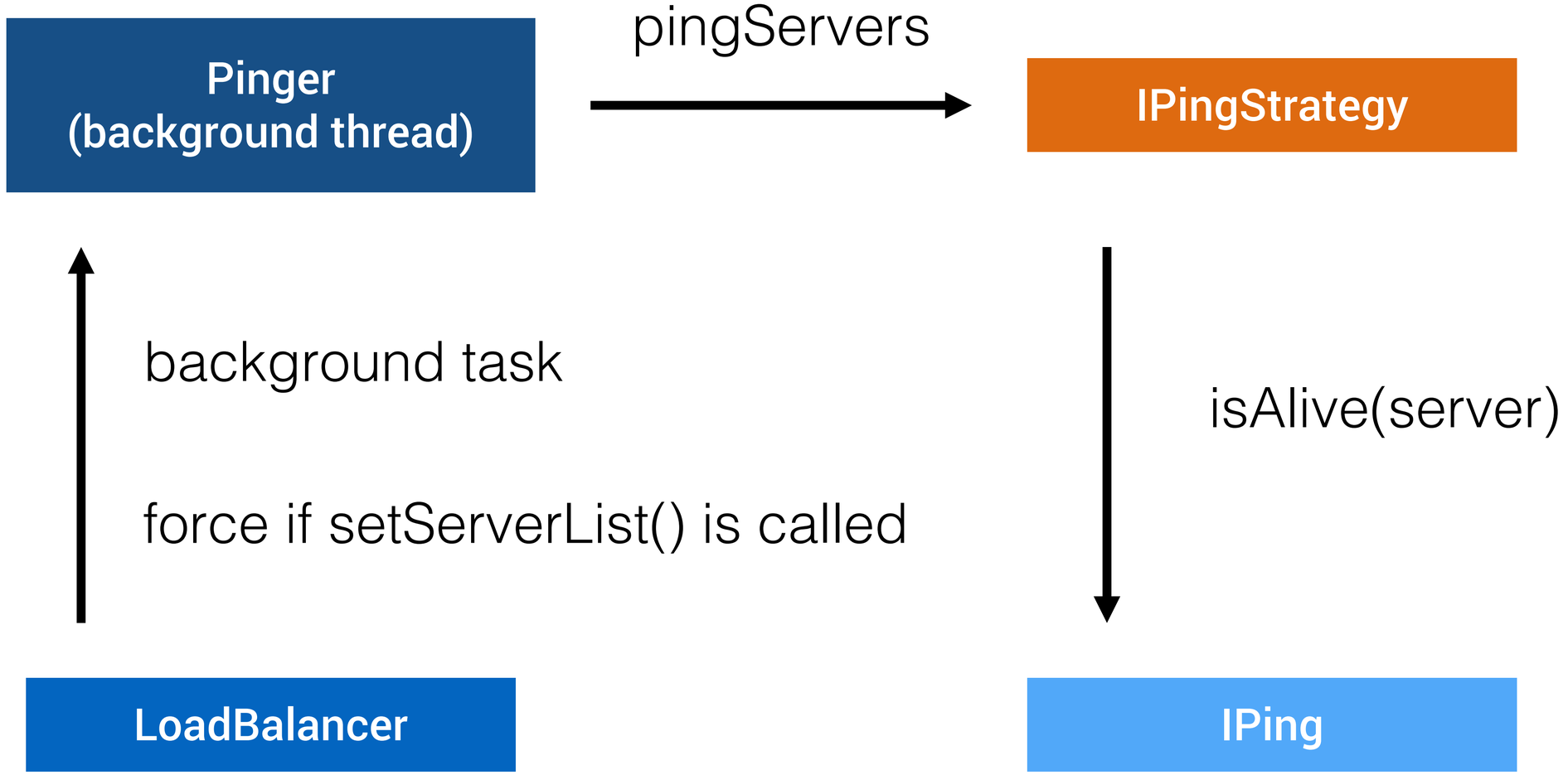

ServerList
S1
S2
S3
S1
S2
S3
S1
S1
t
t + 1


ServerList
S1
S2
S3
S1
S2
S3
S1
S1
t
t + 1
t + 2


S1
S2
S3
S4
S5
Server List
Ping ->
S1
S2
S3
S4
S5
Filter Chain
S2
S4
Rule
S2
call
LoadBalancerClient

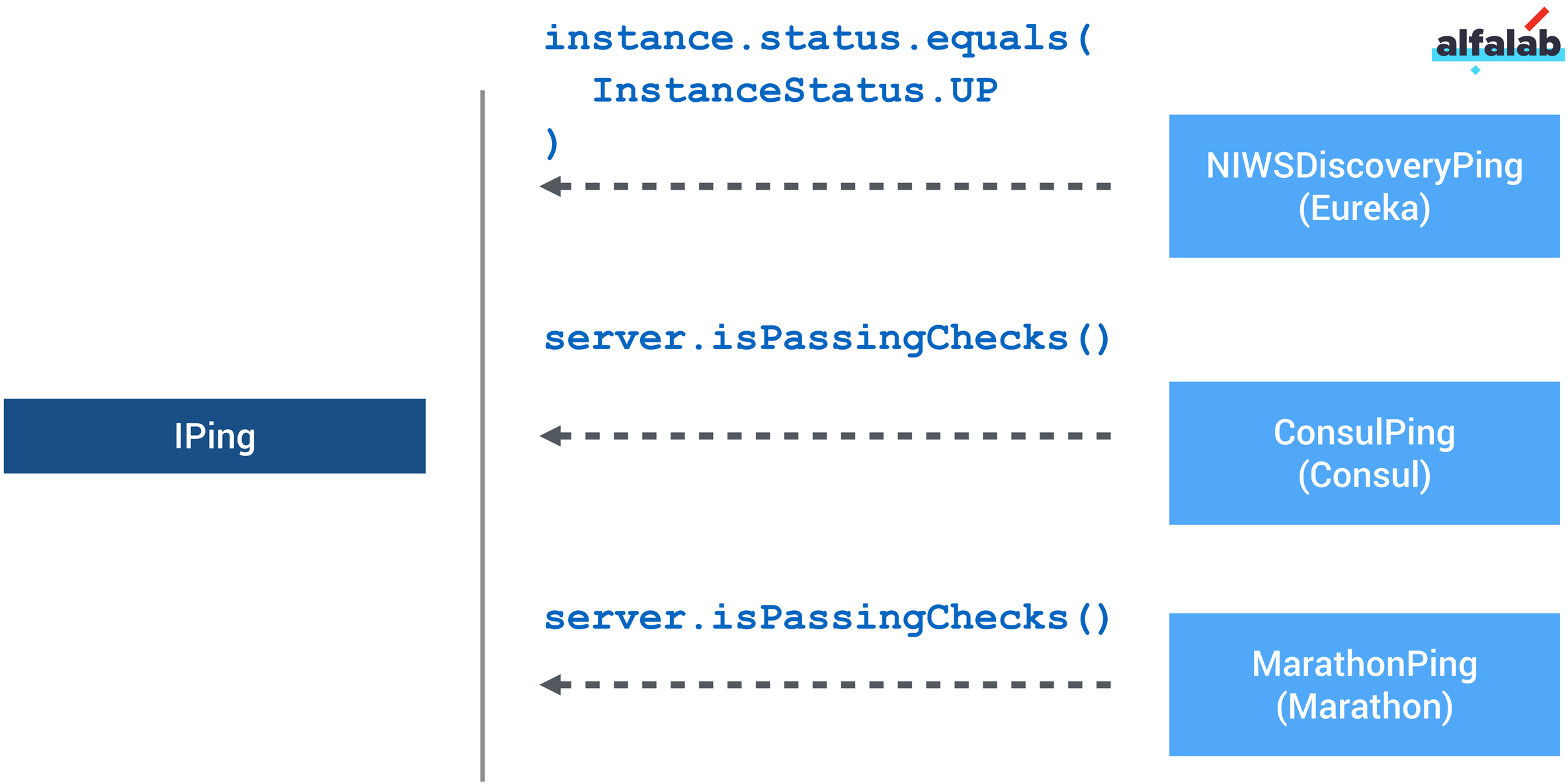

Pinger
(background thread)
background task
force if setServerList() is called
LoadBalancer


Pinger
(background thread)
pingServers
IPingStrategy
background task
force if setServerList() is called
LoadBalancer


Pinger
(background thread)
pingServers
IPingStrategy
background task
force if setServerList() is called
isAlive(server)
LoadBalancer
IPing


```
int numCandidates = servers.length;
boolean[] results = new boolean[numCandidates];

for (int i = 0; i < numCandidates; i++) {
    results[i] = false; /* Default answer is DEAD. */
    if (ping != null) {
        results[i] = ping.isAlive(servers[i]);
    }
}
return results;
```

And what if ping will take a long time?


IPing
instance.status.equals(
InstanceStatus.UP
)
NIWSDiscoveryPing
(Eureka)
server.isPassingChecks()
ConsulPing
(Consul)
server.isPassingChecks()
MarathonPing
(Marathon)
All are fake pings

```
public class MarathonPing implements IPing {
    @Override
    public boolean isAlive(Server server) {
        return server.isHealthChecksPassing();
    }
}
```

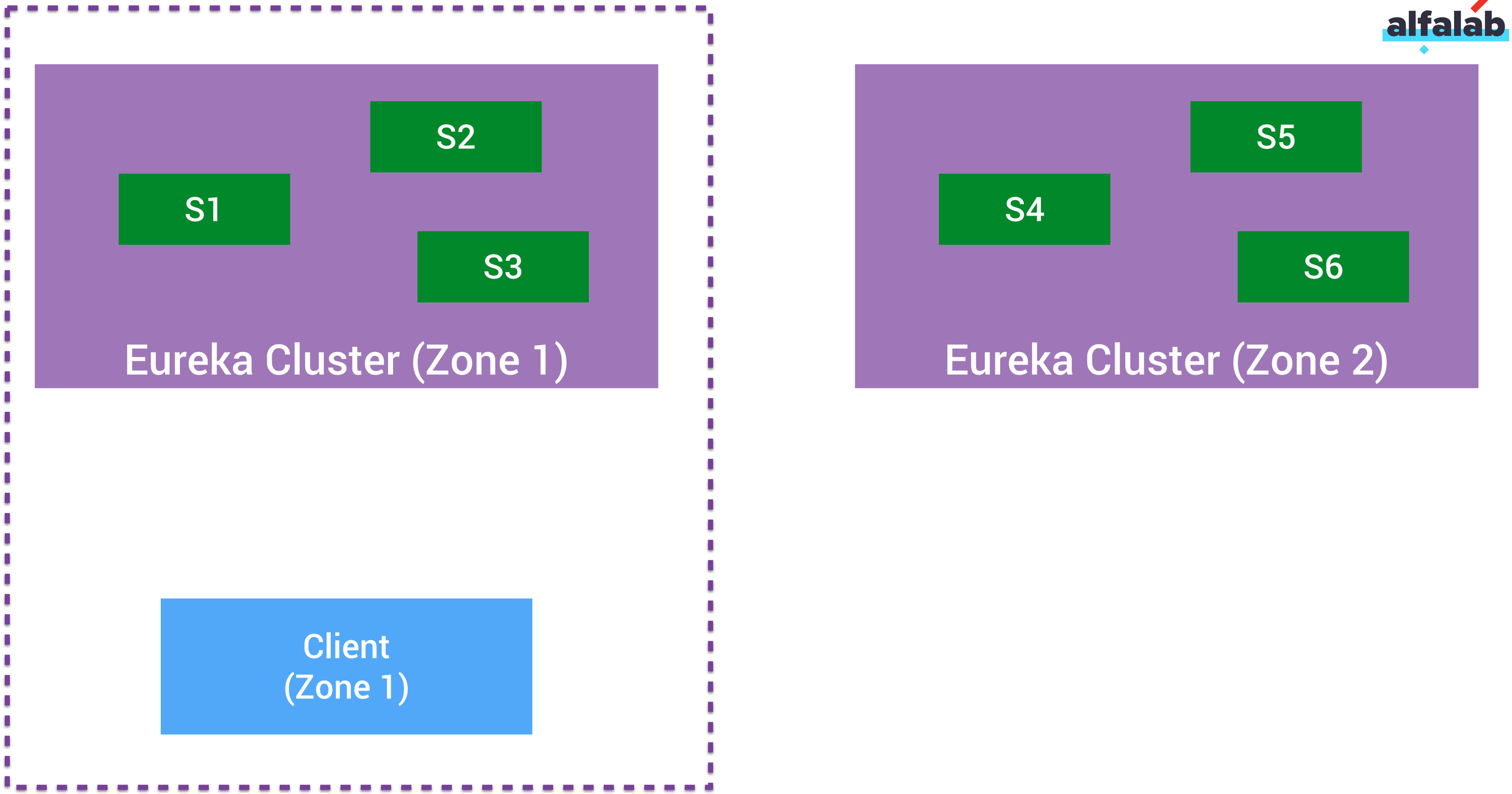
```
public class MarathonPing implements IPing {
    @Override
    public boolean isAlive(Server server) {
        return server.isHealthChecksPassing();
    }
}

public class MarathonServer extends Server {
    public boolean isHealthChecksPassing() {
    return healthChecks.parallelStream()
            .allMatch(HealthCheckResult::isAlive);
    }
}
```


```
public boolean isAlive(Server server) {
  URL url = constructUrl(server);
  HttpResponse response = httpClient.execute(createReq(url));
  return response.getStatusLine().getStatusCode() == 200;
}
```

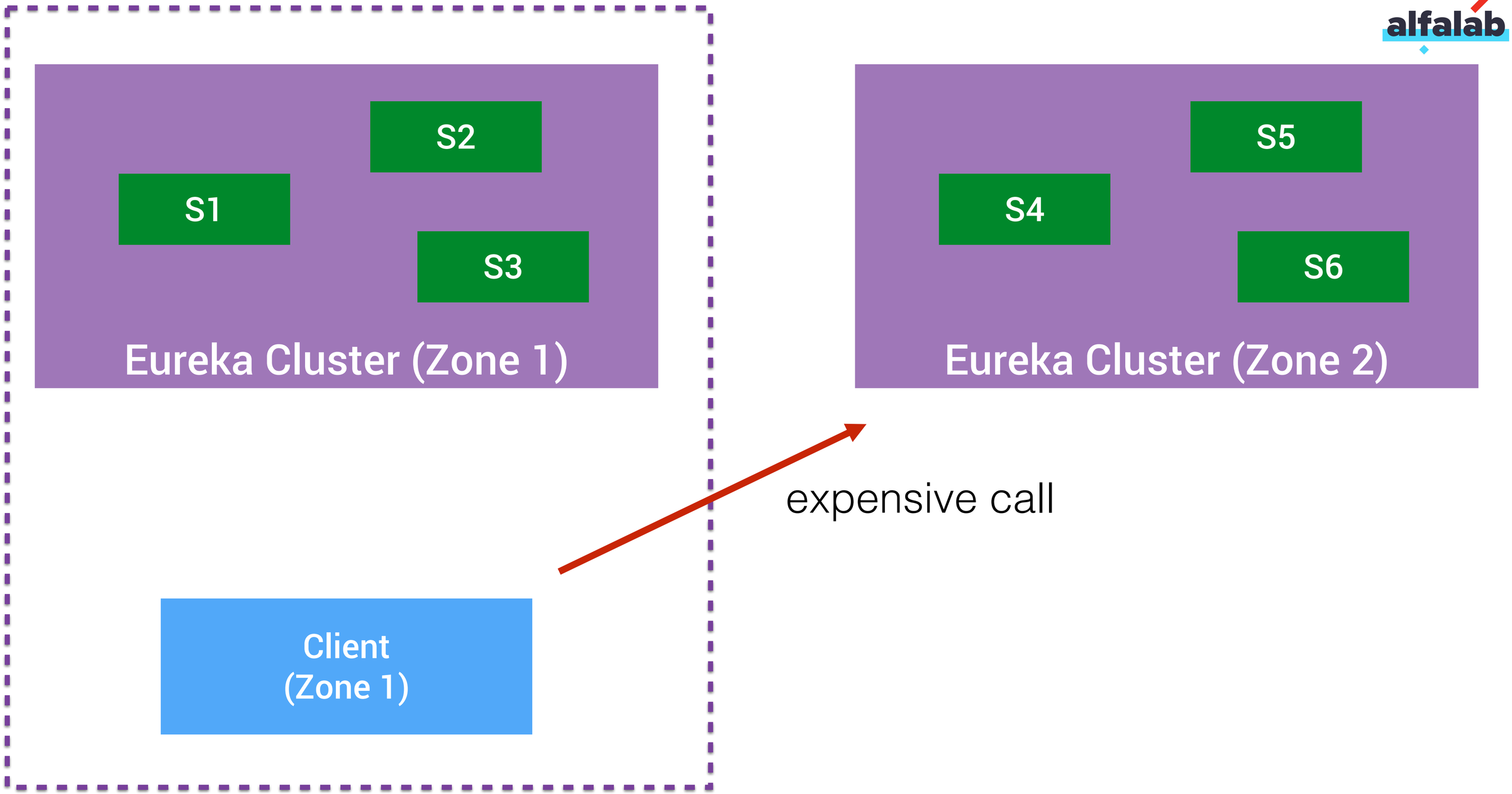
You may implement your own strategy at your own risk

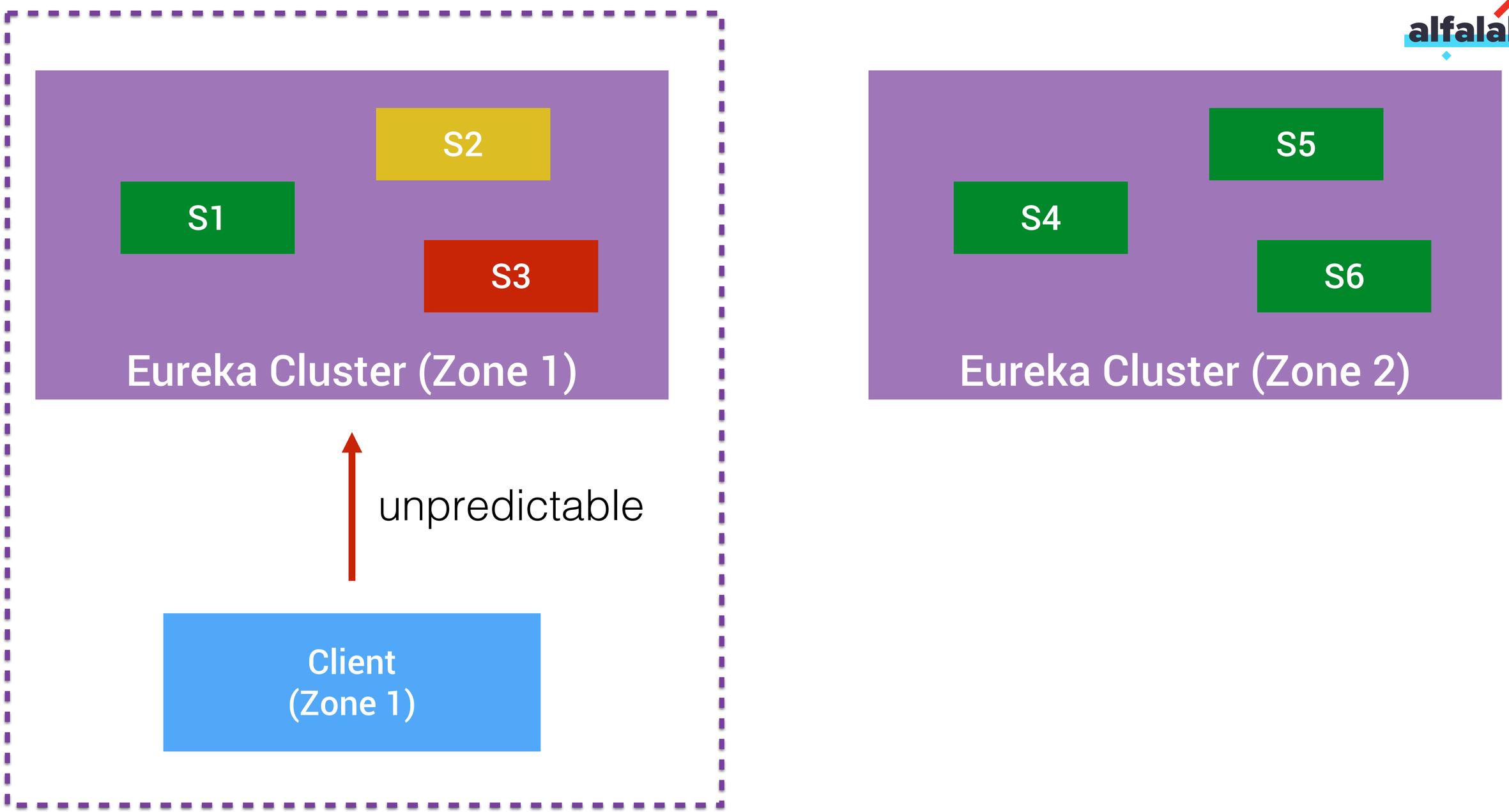
alfalab

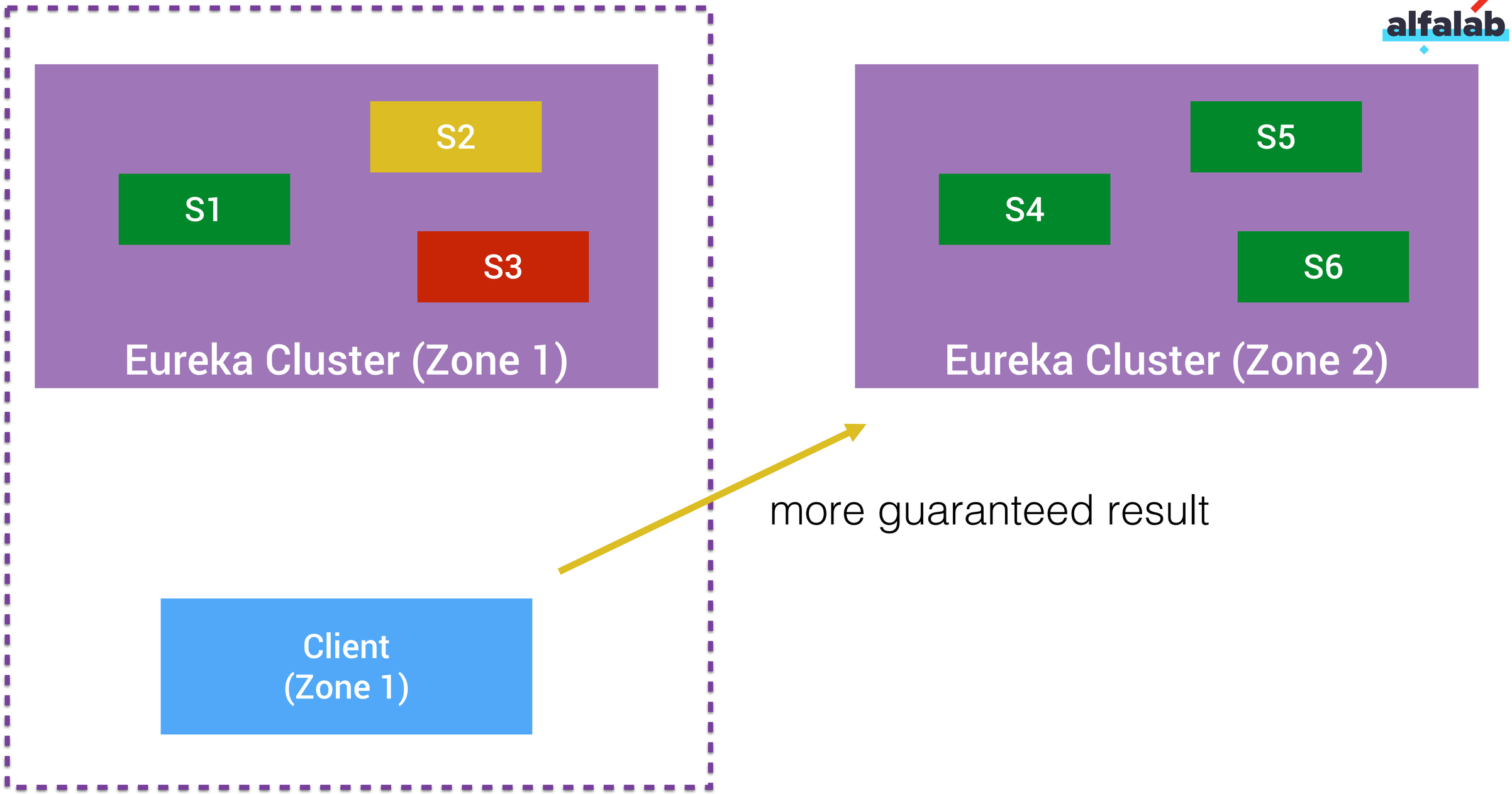

S1
S2
S3
S4
S5
Server List
S1
S2
S3
S4
S5
Filter Chain ->
S2
S4
Rule
S2
call
LoadBalancerClient


S2
S1
S3
Eureka Cluster (Zone 1)
S5
S4
S6
Eureka Cluster (Zone 2)
Client
(Zone 1)


S2
S1
S3
Eureka Cluster (Zone 1)
S5
S4
S6
Eureka Cluster (Zone 2)
cheap call
Client
(Zone 1)


S2
S1
S3
Eureka Cluster (Zone 1)
S5
S4
S6
Eureka Cluster (Zone 2)
expensive call
Client
(Zone 1)


S2
S1
S3
Eureka Cluster (Zone 1)
S5
S4
S6
Eureka Cluster (Zone 2)
unpredictable
Client
(Zone 1)


S2
S1
S3
Eureka Cluster (Zone 1)
S5
S4
S6
Eureka Cluster (Zone 2)
more guaranteed result
Client
(Zone 1)

super.getFilteredServers()

alfalab

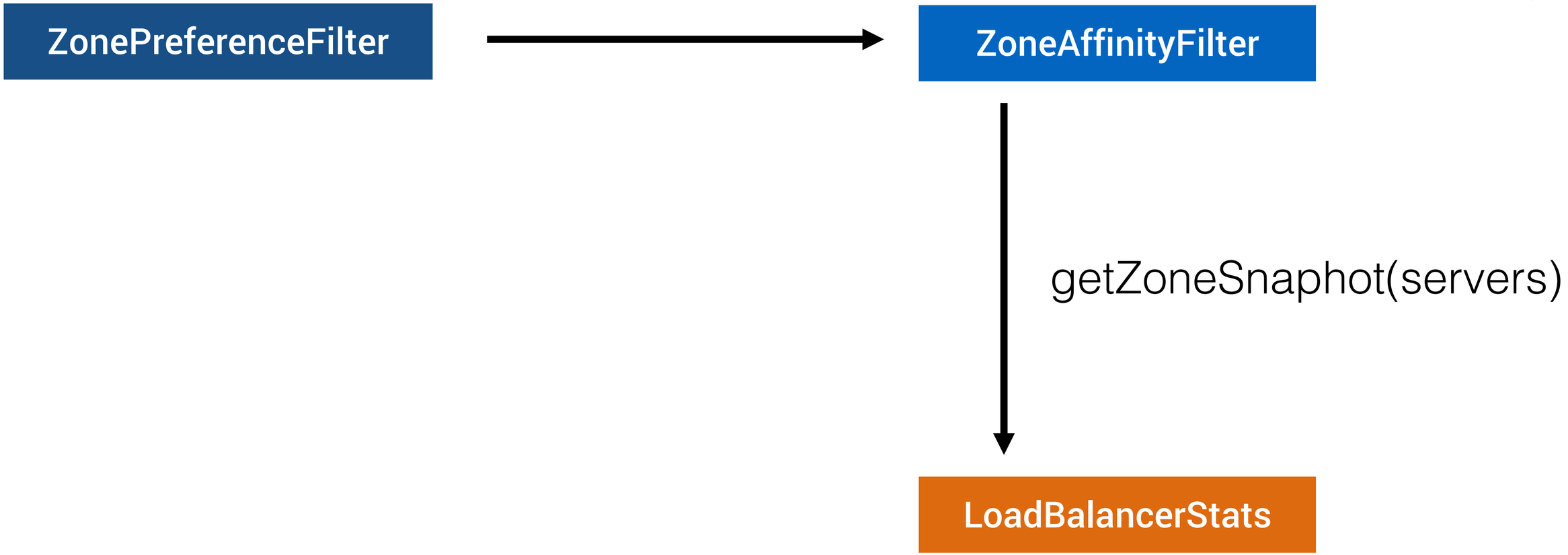
ZonePreferenceFilter → Filter

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


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

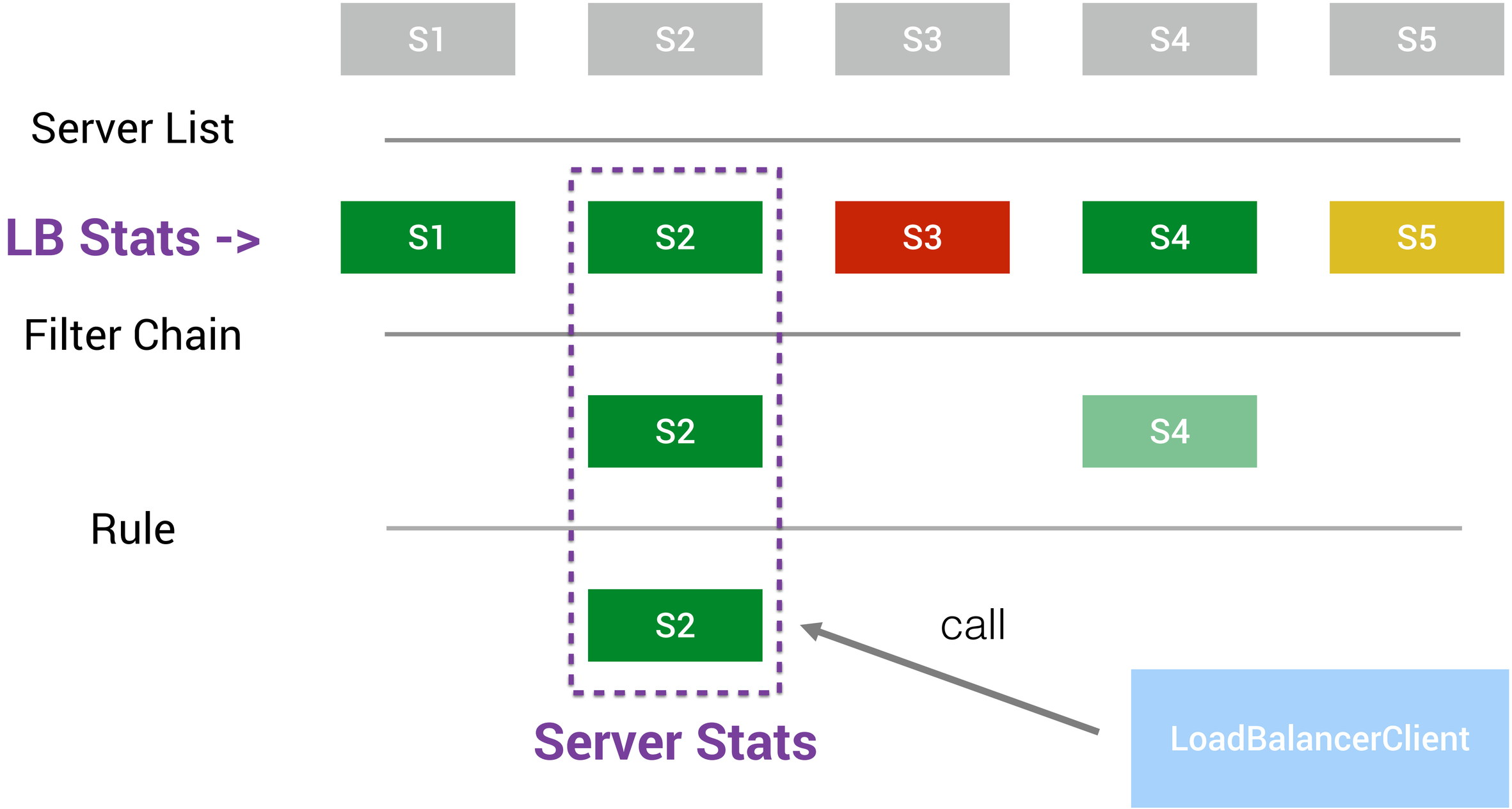
default (local) zone

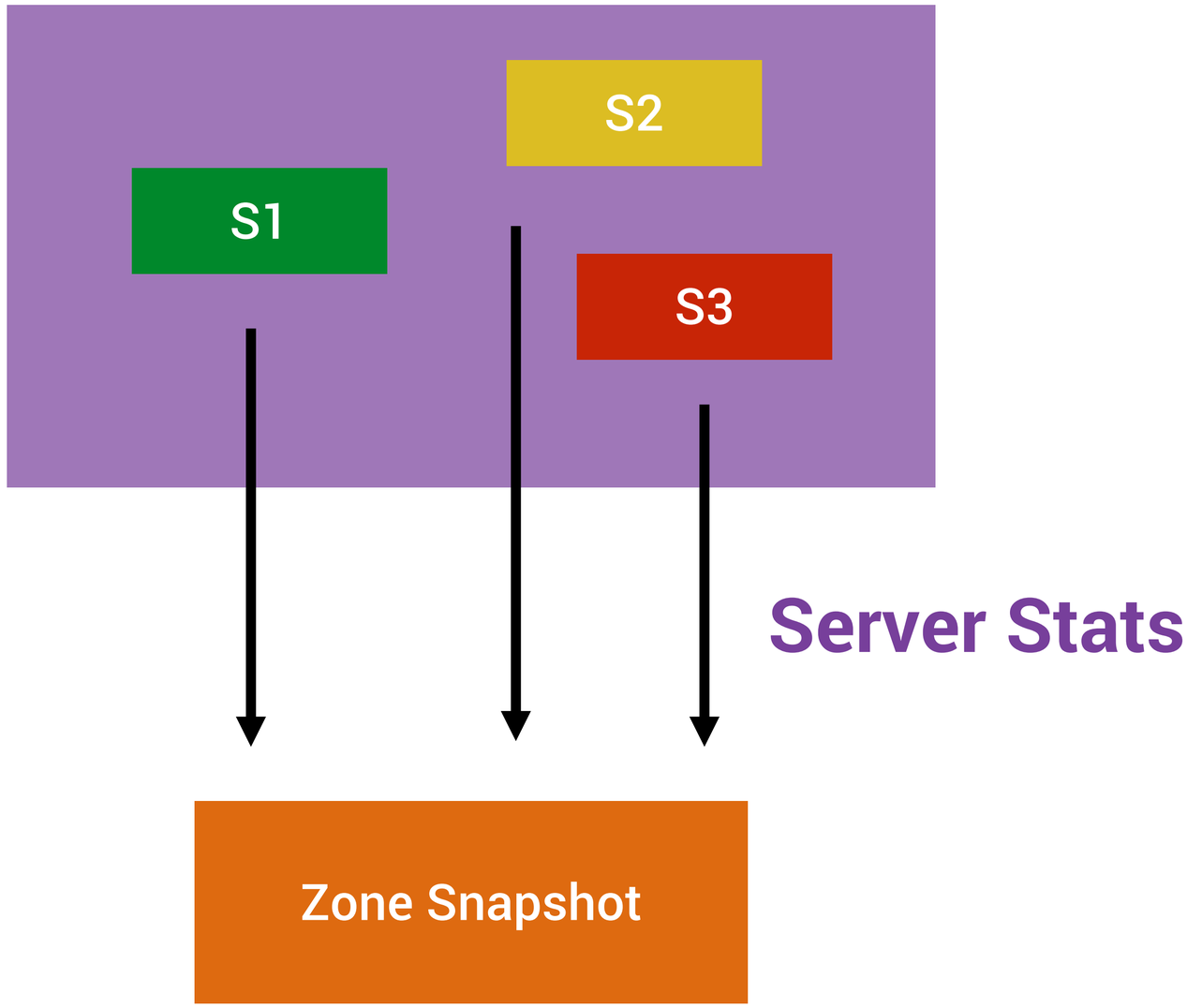
DynamicServerListLoadBalancer

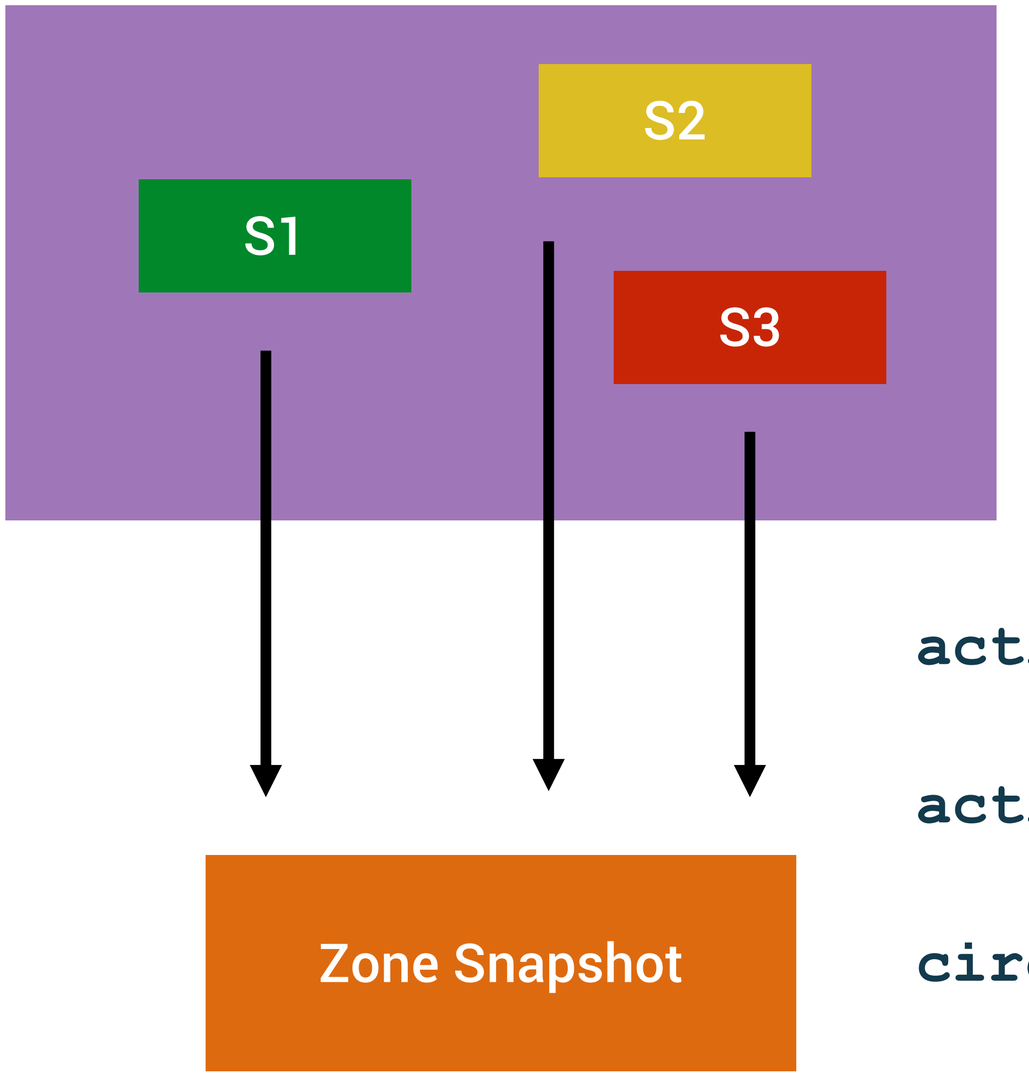
super.getFilteredServers(servers)




S1
S2
S3
S4
S5
Server List
LB Stats ->
S1
S2
S3
S4
S5
Filter Chain
S2
S4
Rule
S2
call
LoadBalancerClient


S1
S2
S3
S4
S5
Server List
LB Stats ->
S1
S2
S3
S4
S5
Filter Chain
S2
S4
Rule
S2
call
Server Stats
LoadBalancerClient


S2
S1
S3
Server Stats
Zone Snapshot

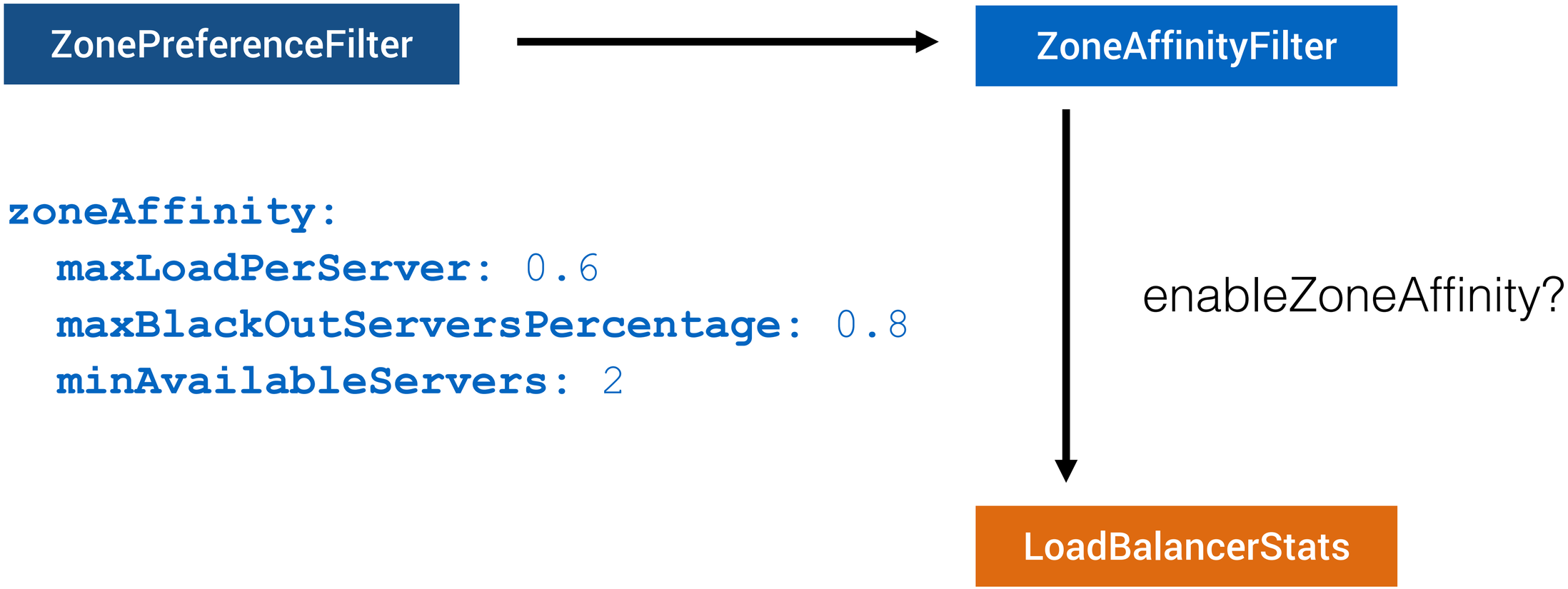


`activeConnections`

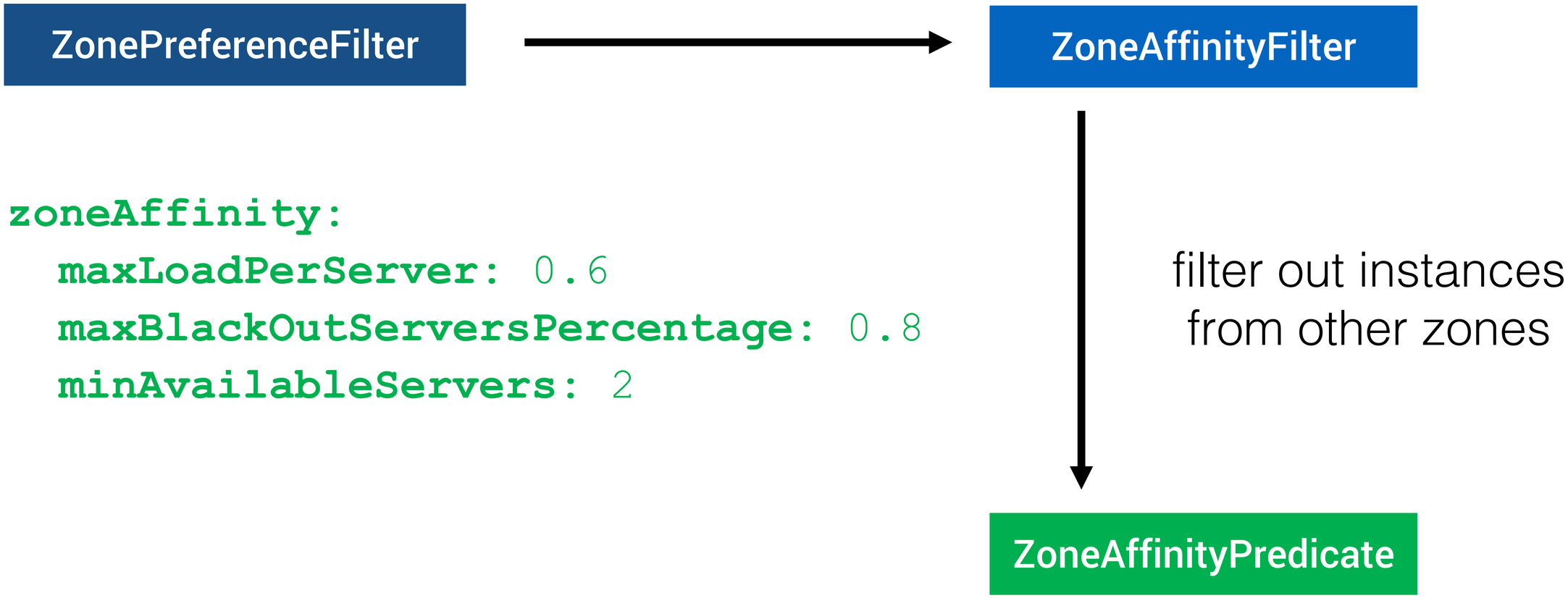
`activeServers`

`circuitBreakerTrippedCount`

super.getFilteredServers(servers)

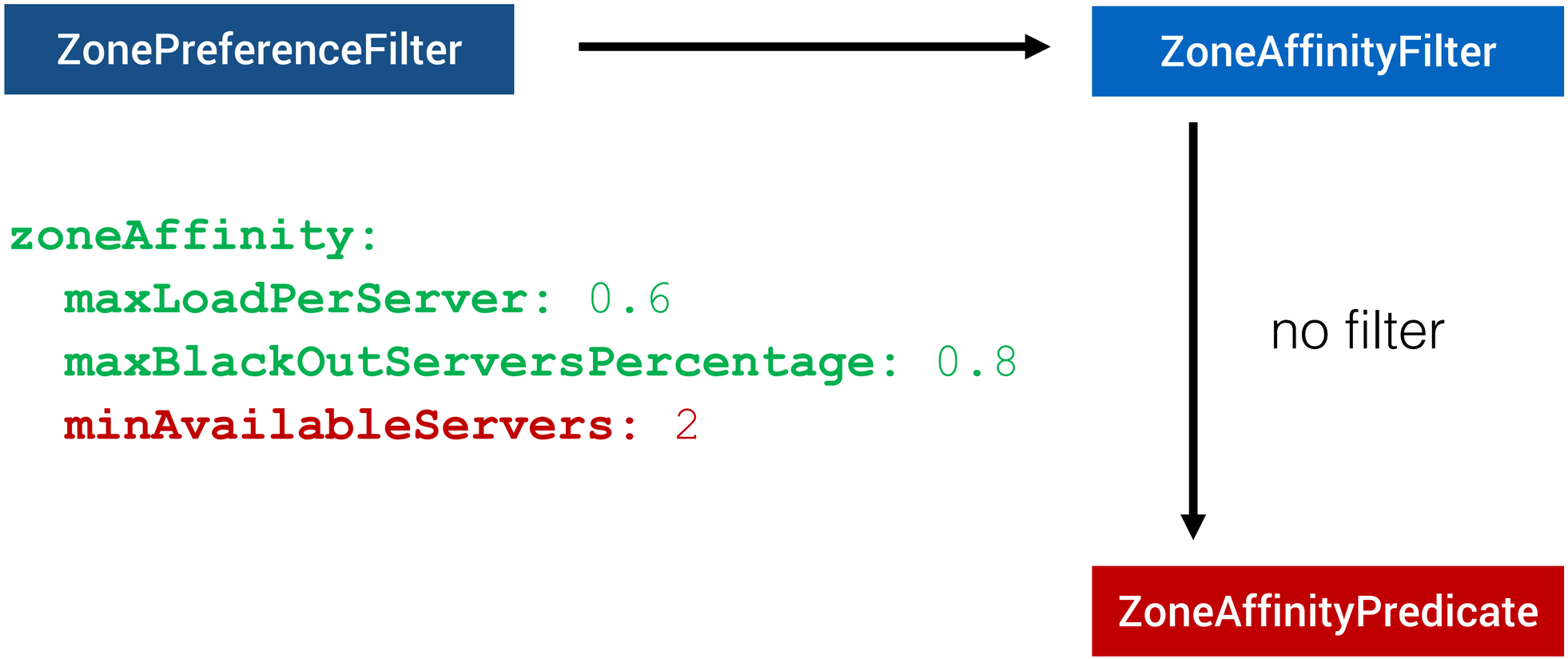


super.getFilteredServers(servers)

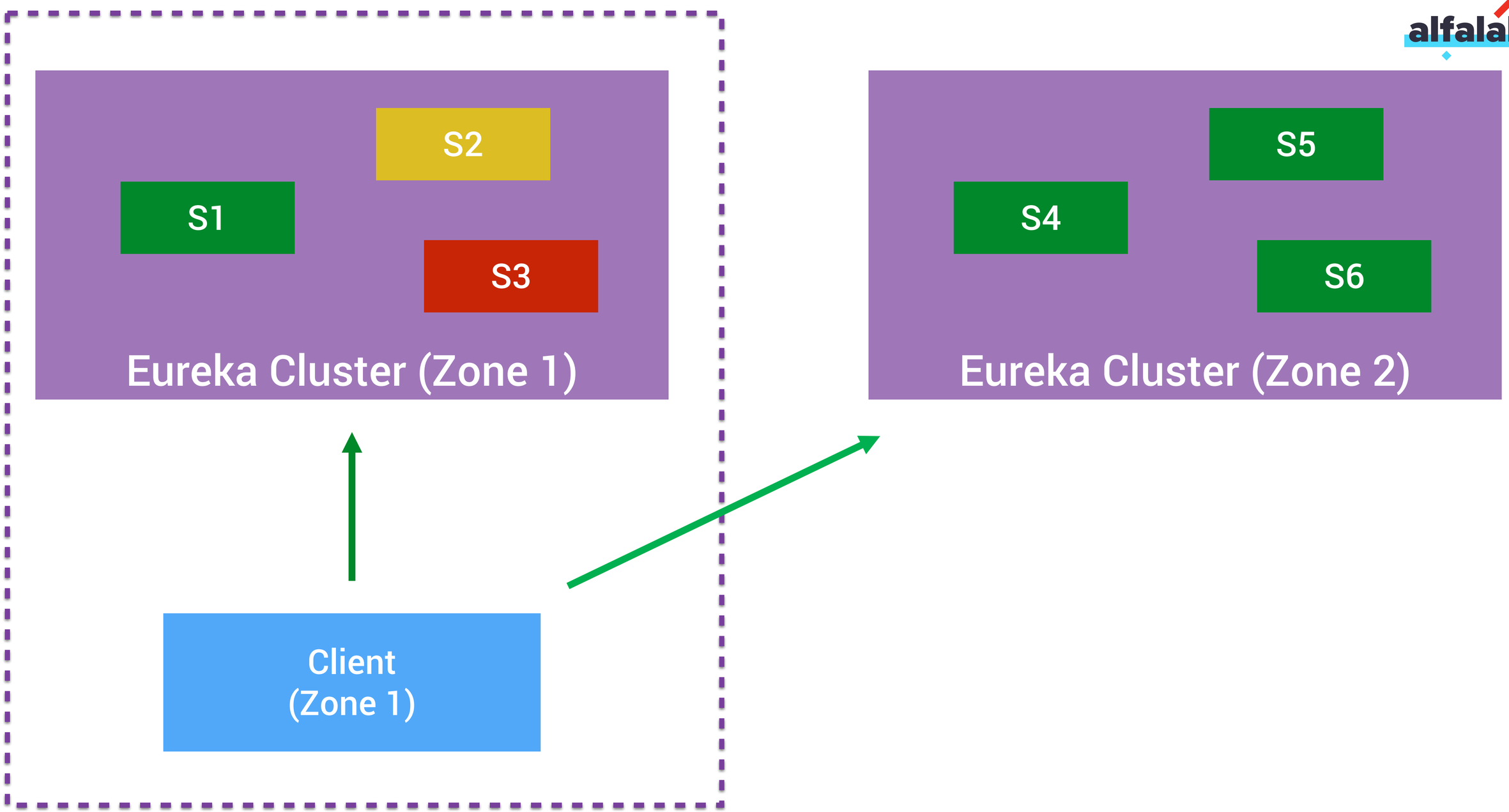



S2
S1
S3
Eureka Cluster (Zone 1)
S5
S4
S6
Eureka Cluster (Zone 2)
cheap call
Client
(Zone 1)

super.getFilteredServers(servers)



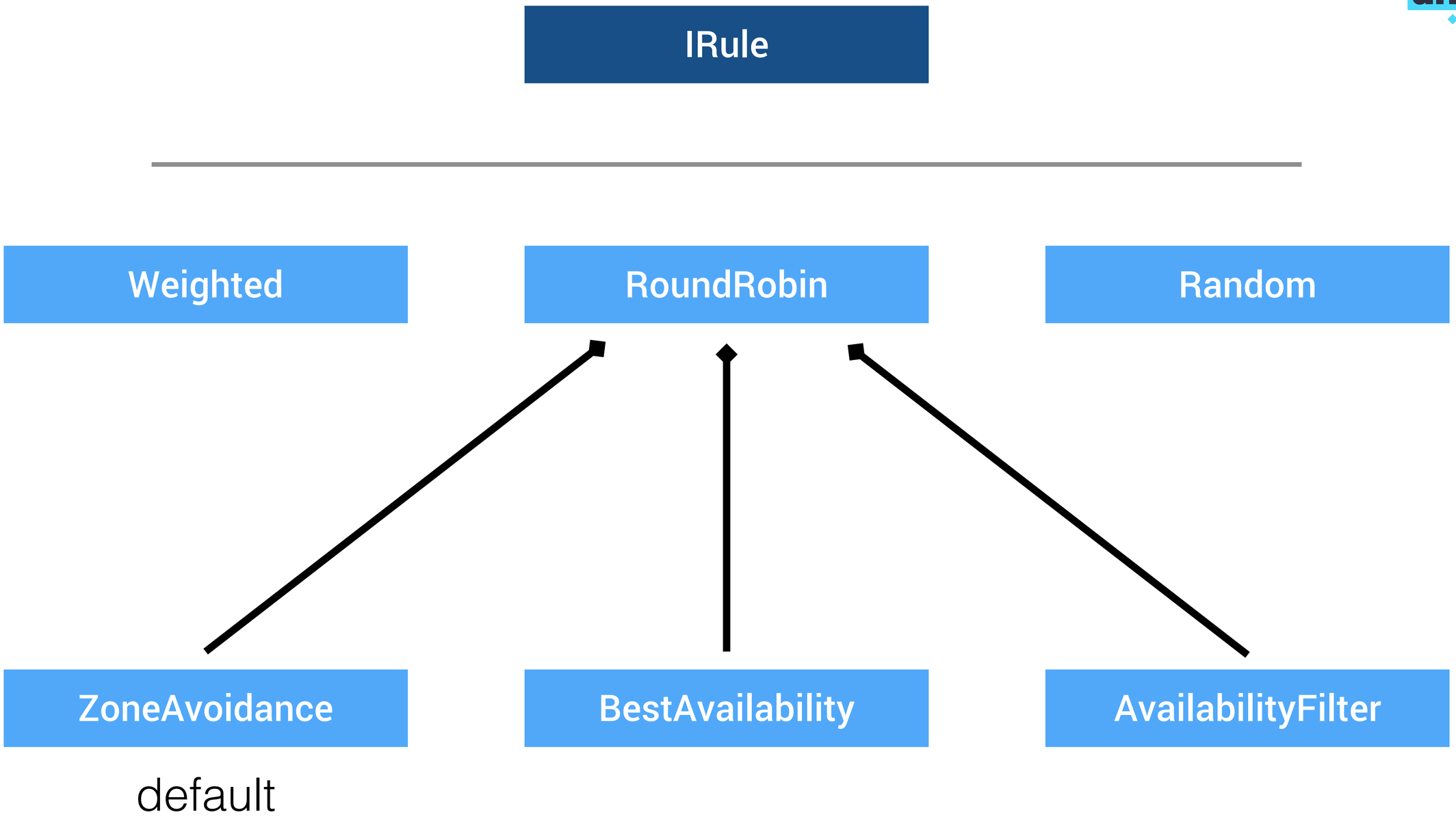

```
zoneAffinity:
  maxLoadPerServer: 0.6
  maxBlackOutServersPercentage: 0.8
  minAvailableServers: 2
```

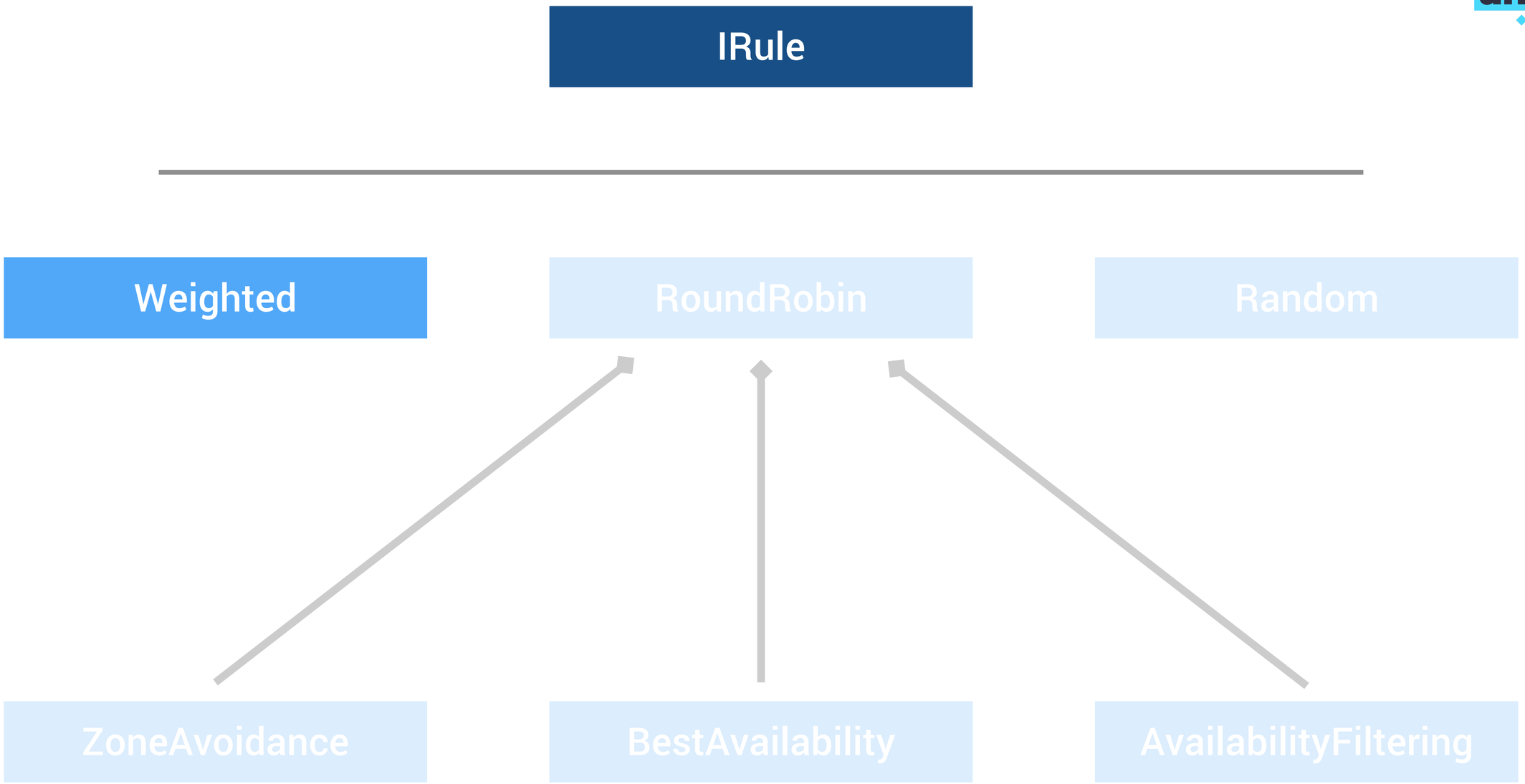

S2
S1
S3
Eureka Cluster (Zone 1)
S5
S4
S6
Eureka Cluster (Zone 2)
Client
(Zone 1)


Use Eureka. Use zones

Define your rules

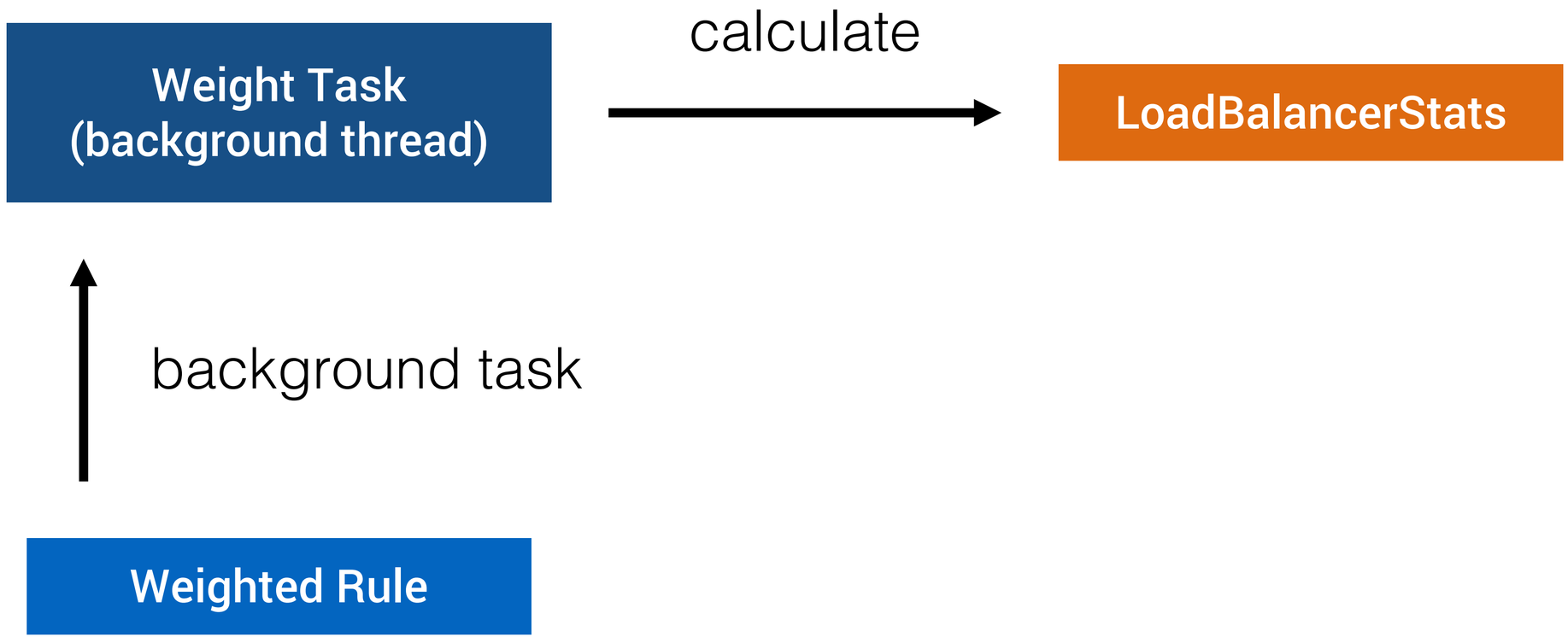

S1
S2
S3
S4
S5
Server List
S1
S2
S3
S4
S5
Filter Chain
S2
?
S4
Rule
S2
call
LoadBalancerClient


IRule
Weighted
RoundRobin
Random
ZoneAvoidance
BestAvailability
AvailabilityFilter
default

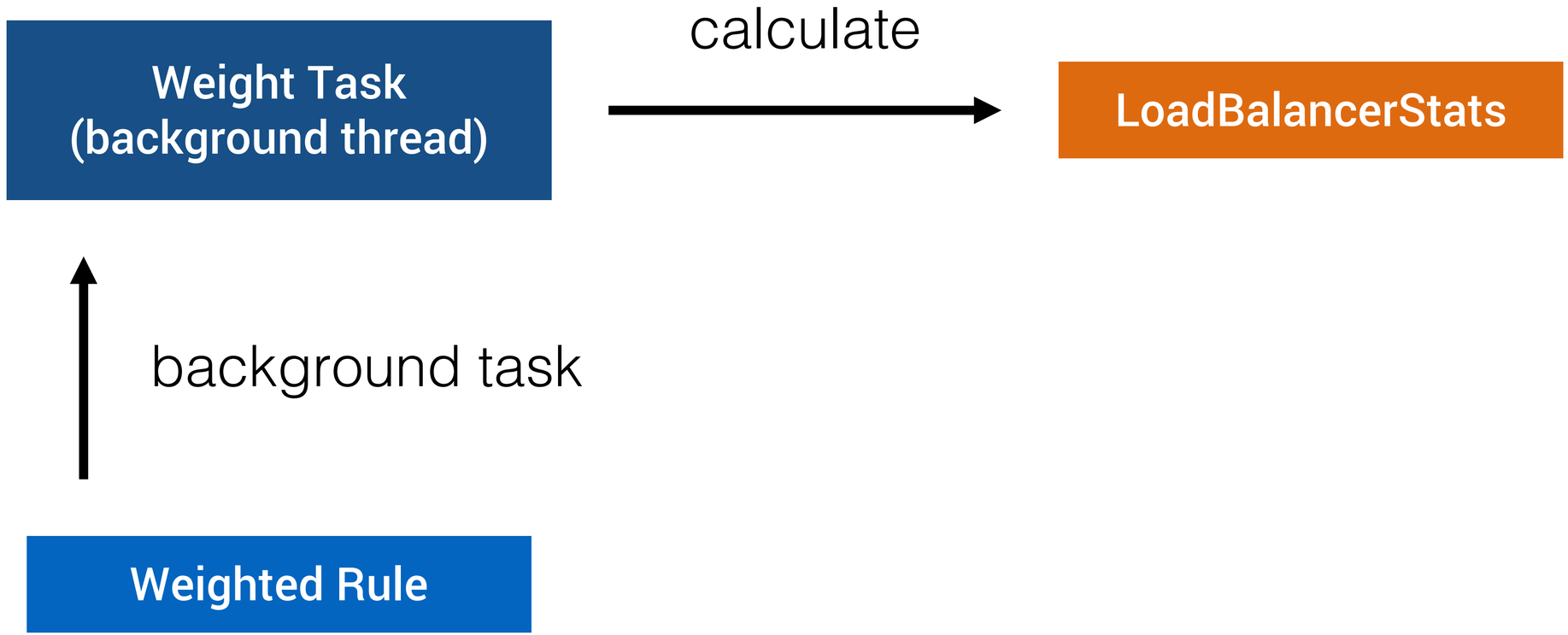

IRule
Weighted
RoundRobin
Random
ZoneAvoidance
BestAvailability
AvailabilityFiltering

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

Weighted typically is predefined




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




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

More response time -> less weight

alfalab

Slow Instance

100 ms

Fast Instance

10 ms

	Slow Instance	Fast Instance
5 000 requests	100 ms	10 ms

	Slow Instance	Fast Instance
5 000 requests	100 ms	10 ms

Round Robin Rule

```
Slow: 2500 and Fast: 2500
Average time: 70
```

	Slow Instance	Fast Instance
5 000 requests	100 ms	10 ms

Round Robin Rule

```
Slow: 2500 and Fast: 2500
Average time: 70
```

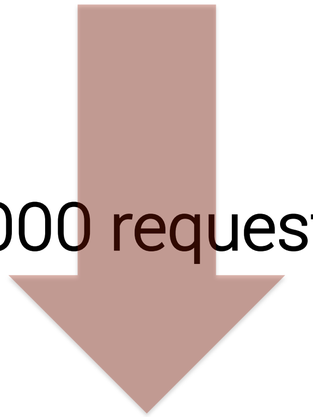
	Slow Instance	Fast Instance
5 000 requests	100 ms	10 ms

Round Robin Rule

```
Slow: 2500 and Fast: 2500
Average time: 70
```

Weighted Rule

```
Slow: 634 and Fast: 4366
Average time: 27
```



1 000 requests

Slow Instance

100 ms

Fast Instance

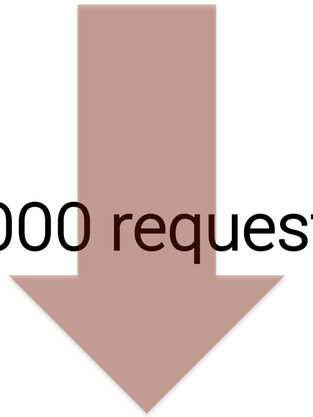
10 ms

Round Robin Rule

```
Slow: 500 and Fast: 500
Average time: 70
```

Weighted Rule

```
?
?
```



1 000 requests

Slow Instance

100 ms

Fast Instance

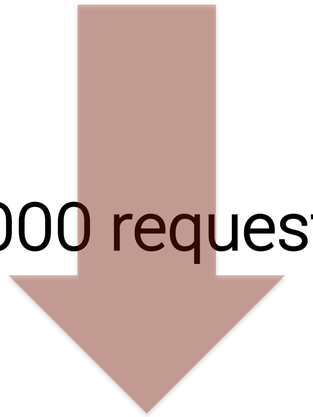
10 ms

Round Robin Rule

```
Slow: 500 and Fast: 500
Average time: 70
```

Weighted Rule

```
Slow: 500 and Fast: 500
Average time: 72
```



1 000 requests

Slow Instance

100 ms

Fast Instance

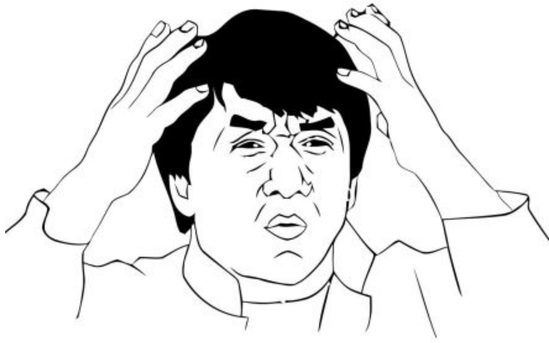
10 ms

Round Robin Rule

```
Slow: 500 and Fast: 500
Average time: 70
```

Weighted Rule

```
Slow: 500 and Fast: 500
Average time: 72
```



Let's explain

```
test-service:                                                #service name
  ribbon:                                                    #namespace
    ServerWeightTaskTimerInterval: 30000                     #ms
```

Let's explain

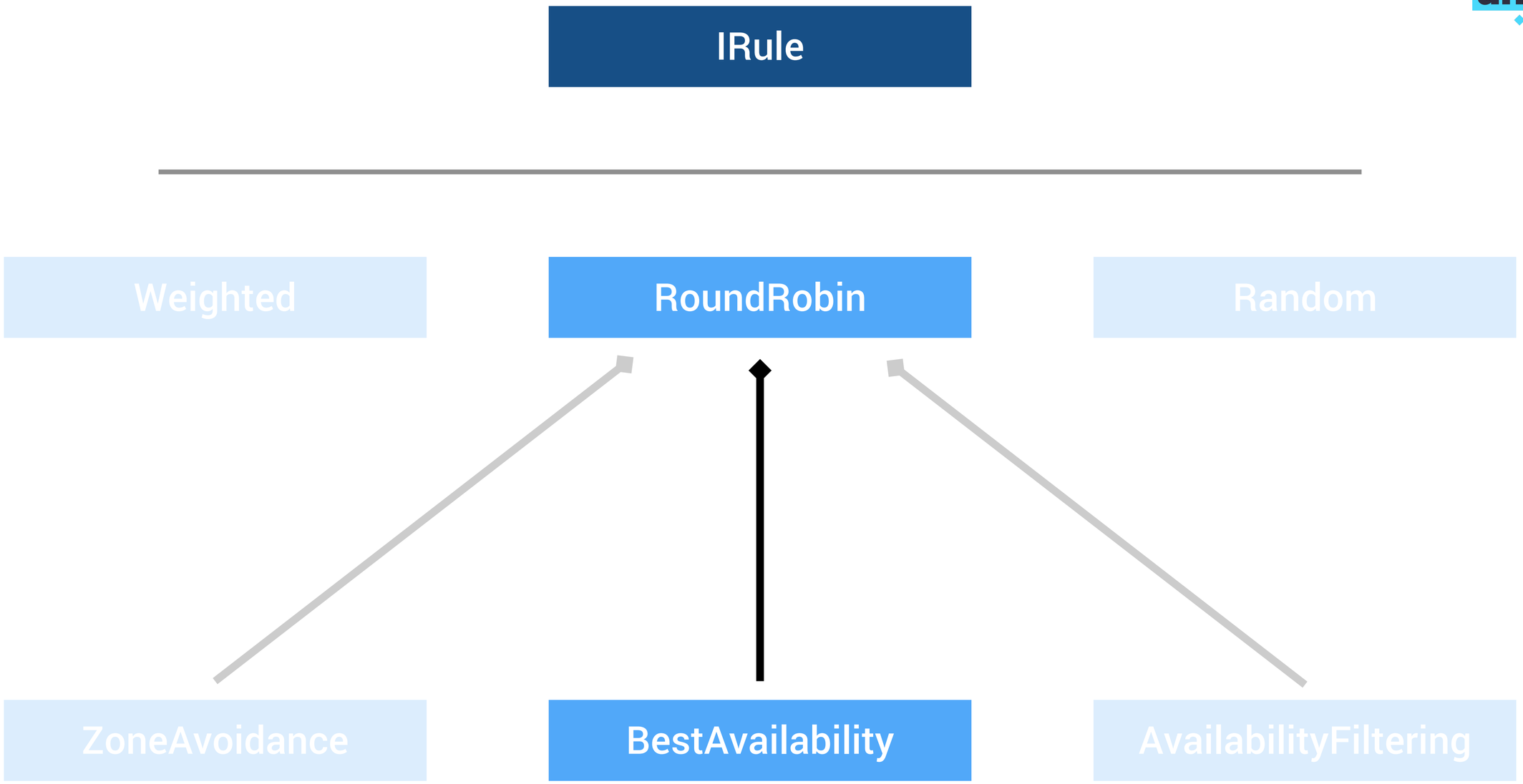
```
test-service:                                                #service name
  ribbon:                                                    #namespace
    ServerWeightTaskTimerInterval: 30000                     #ms

Weigths [0.0, 0.0]
```

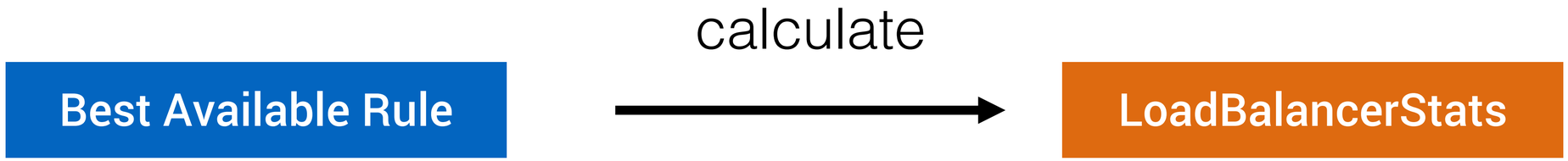
Let's explain

```
test-service:                                                #service name
  ribbon:                                                    #namespace
    ServerWeightTaskTimerInterval: 1000                      #ms
```

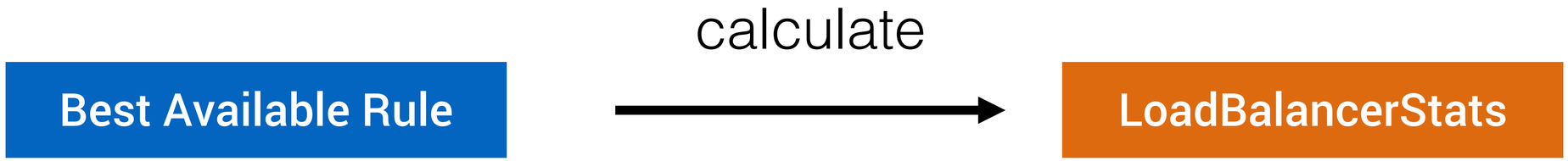
Let's explain

```
test-service:                                          #service name
  ribbon:                                              #namespace
    ServerWeightTaskTimerInterval: 1000                #ms

Weigths [12.2851230167854 62, 120.30885719400065]
```


IRule
Weighted
RoundRobin
Random
ZoneAvoidance
BestAvailability
AvailabilityFiltering




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




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

Less active connections -> more chance to success

	Slow Instance	Fast Instance
1 000 requests	100 ms	10 ms

Round Robin Rule

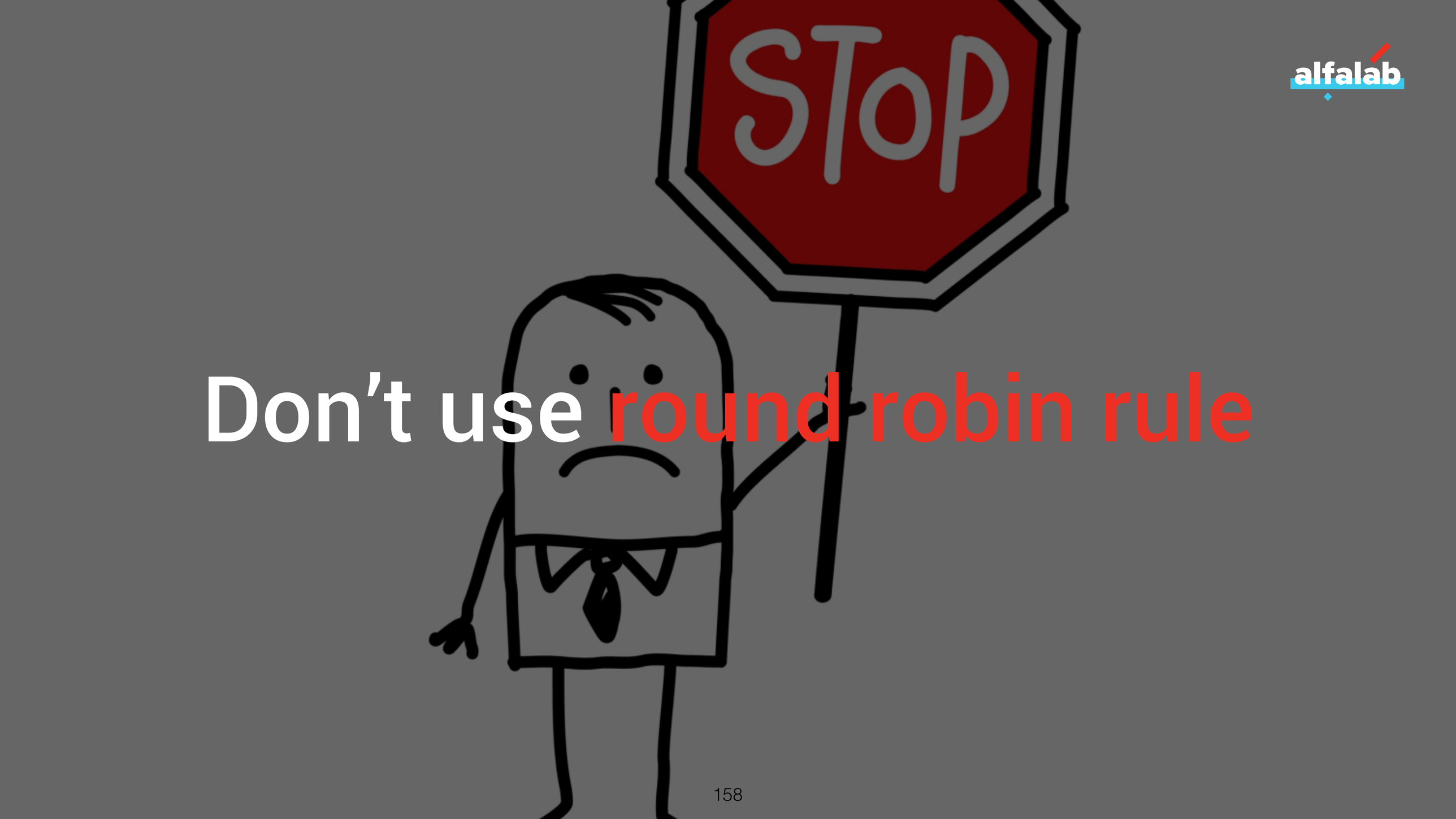
```
Slow: 500 and Fast: 500
Average time: 70
```

Weighted Rule

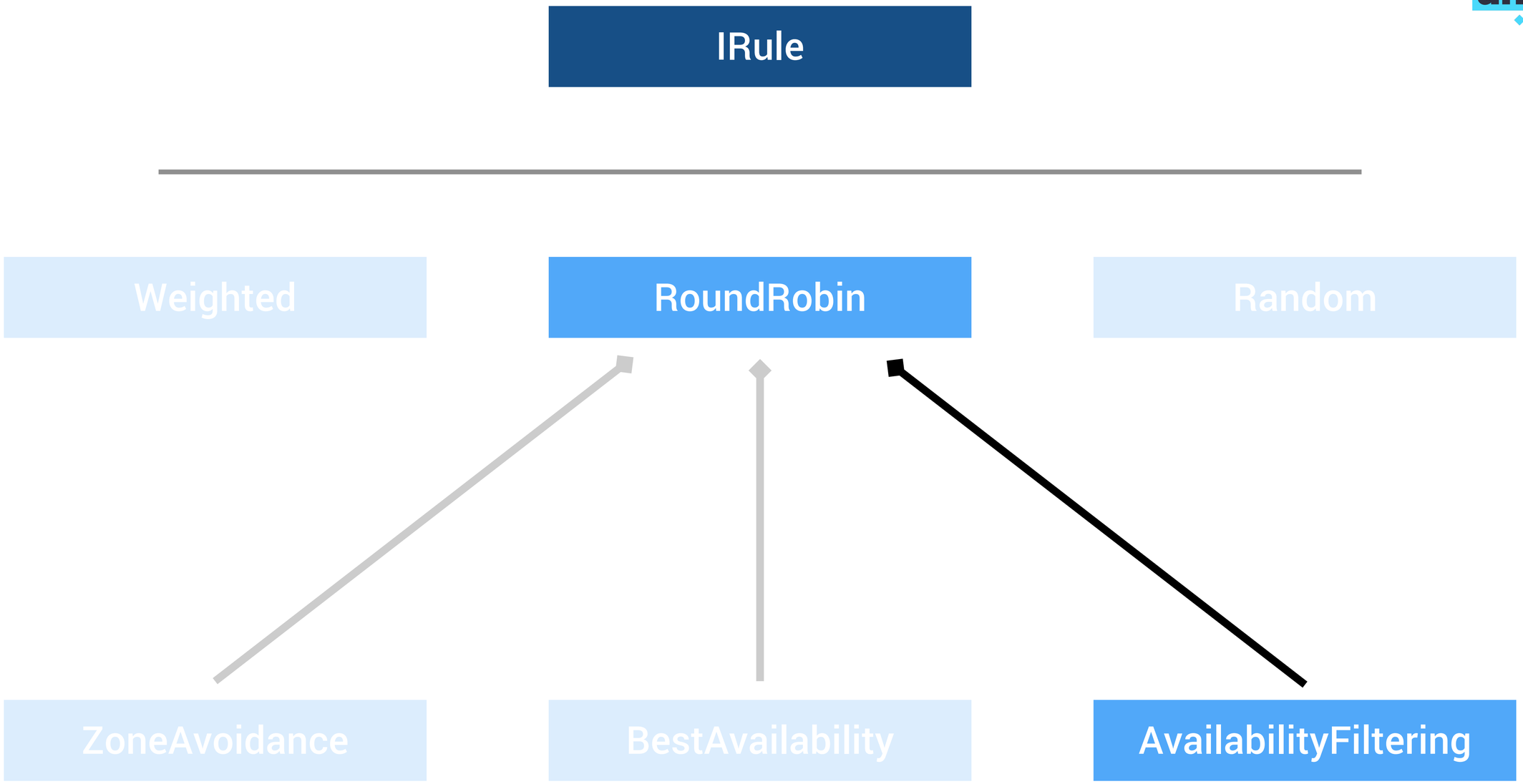
```
Slow: 500 and Fast: 500
Average time: 72
```

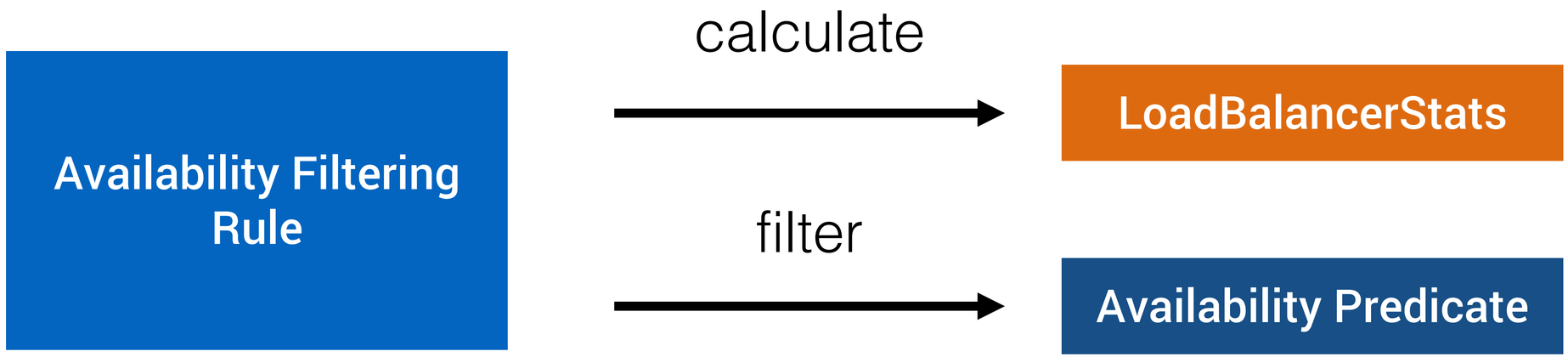
Best Available Rule

```
Slow: 152 and Fast: 847
Average time: 38
```



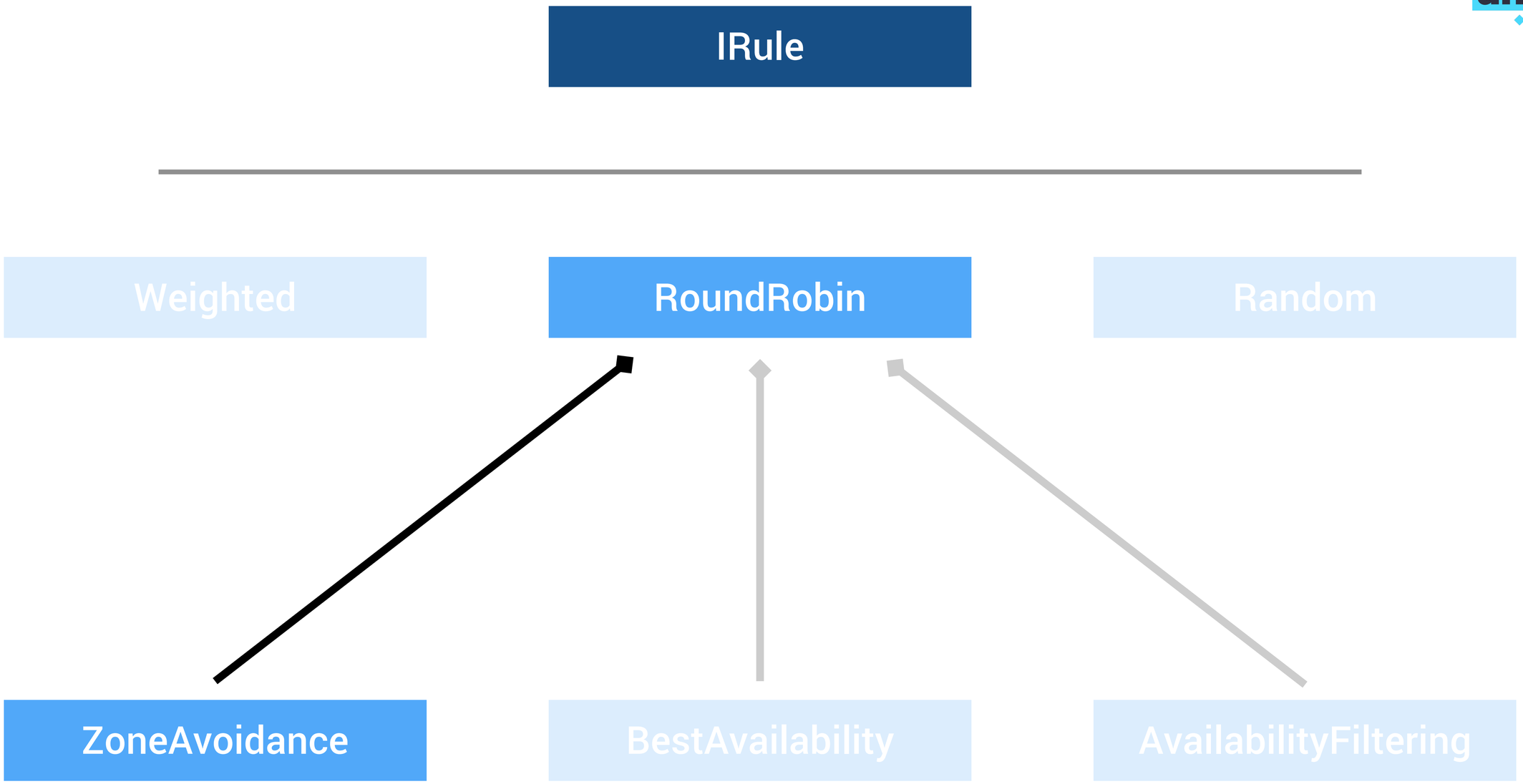
Don’t use round robin rule

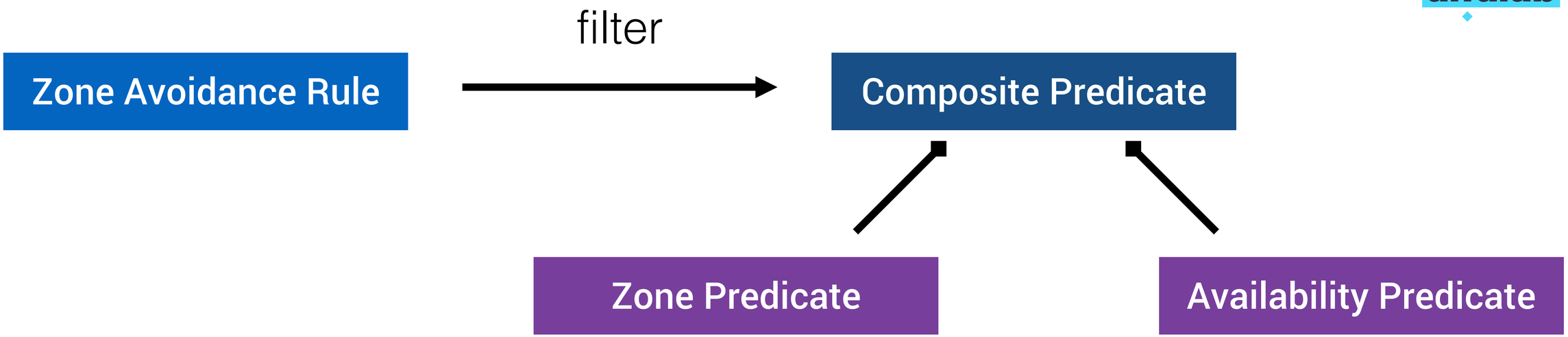

IRule
Weighted
RoundRobin
Random
ZoneAvoidance
BestAvailability
AvailabilityFiltering

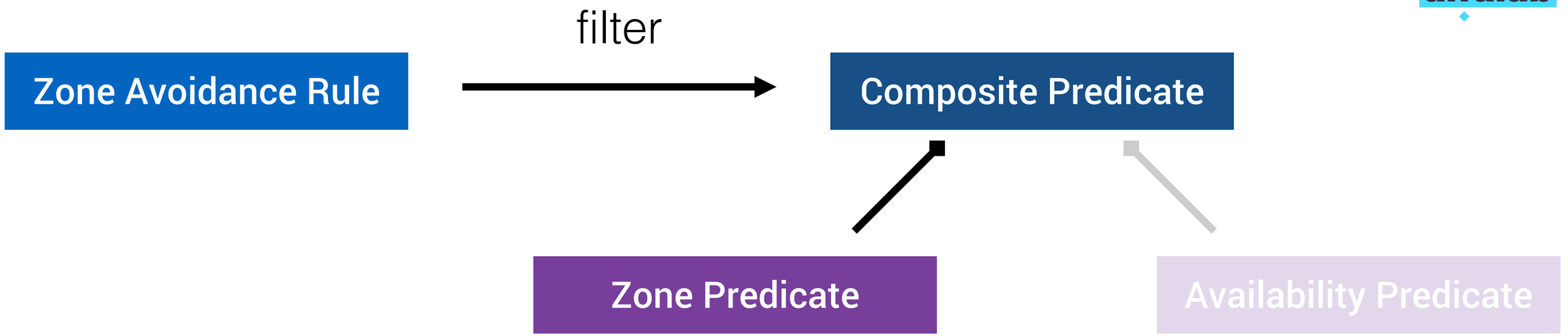



```
filtered -> activeConnections < maxActiveConnections

filtered -> !isCircuitBreakerTripped
```

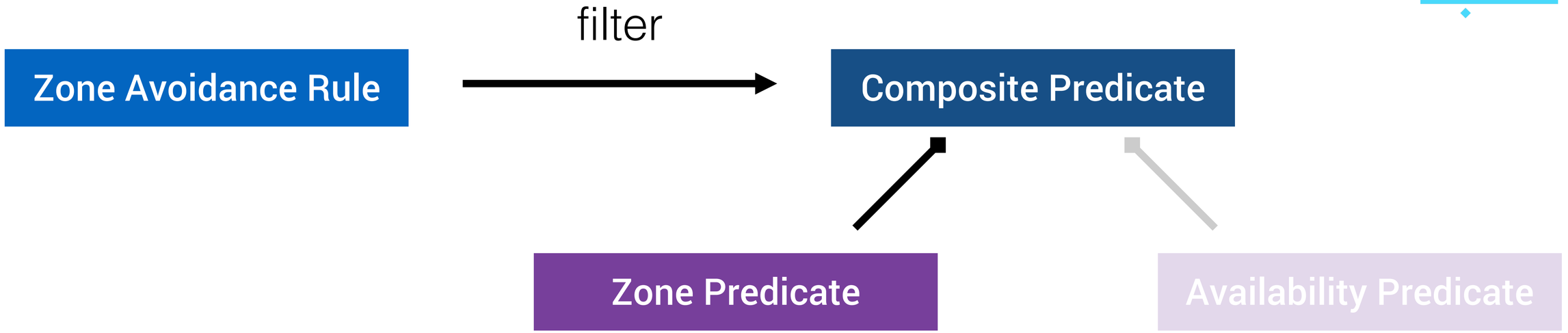
Like a round robin


IRule
Weighted
RoundRobin
Random
ZoneAvoidance
BestAvailability
AvailabilityFiltering

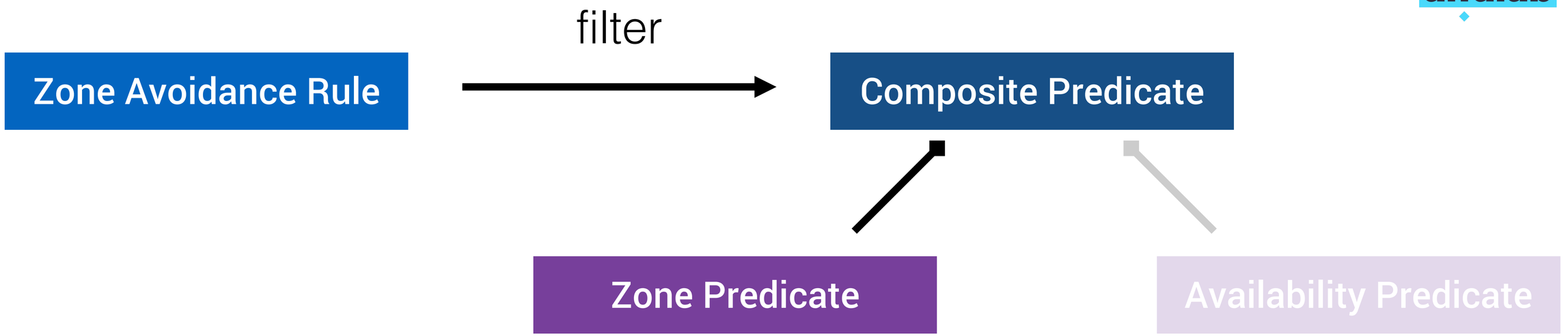

filter
Zone Avoidance Rule
Composite Predicate
Zone Predicate
Availability Predicate



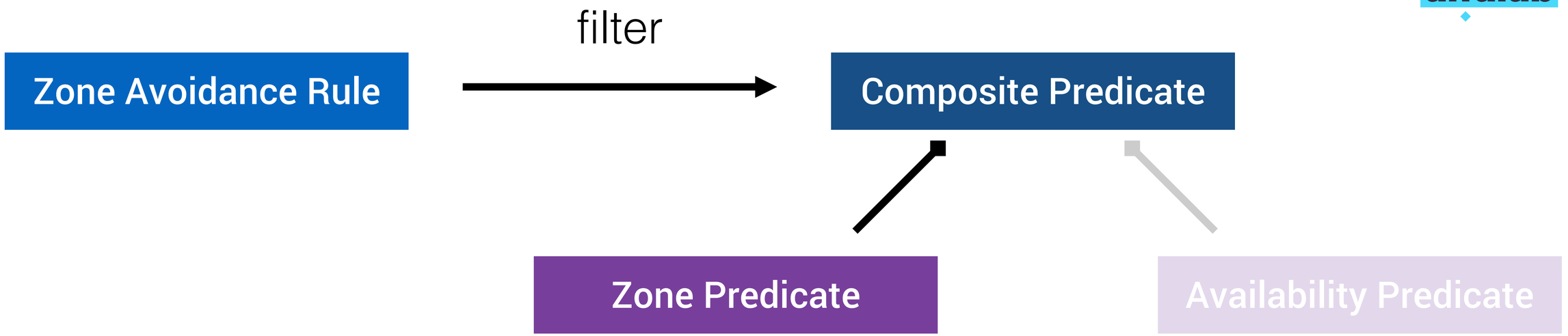

```
CompositePredicateBuilder
   int minimalFilteredServers = 1
   float minimalFilteredPercentage = 0


predicate.filter(servers)
```



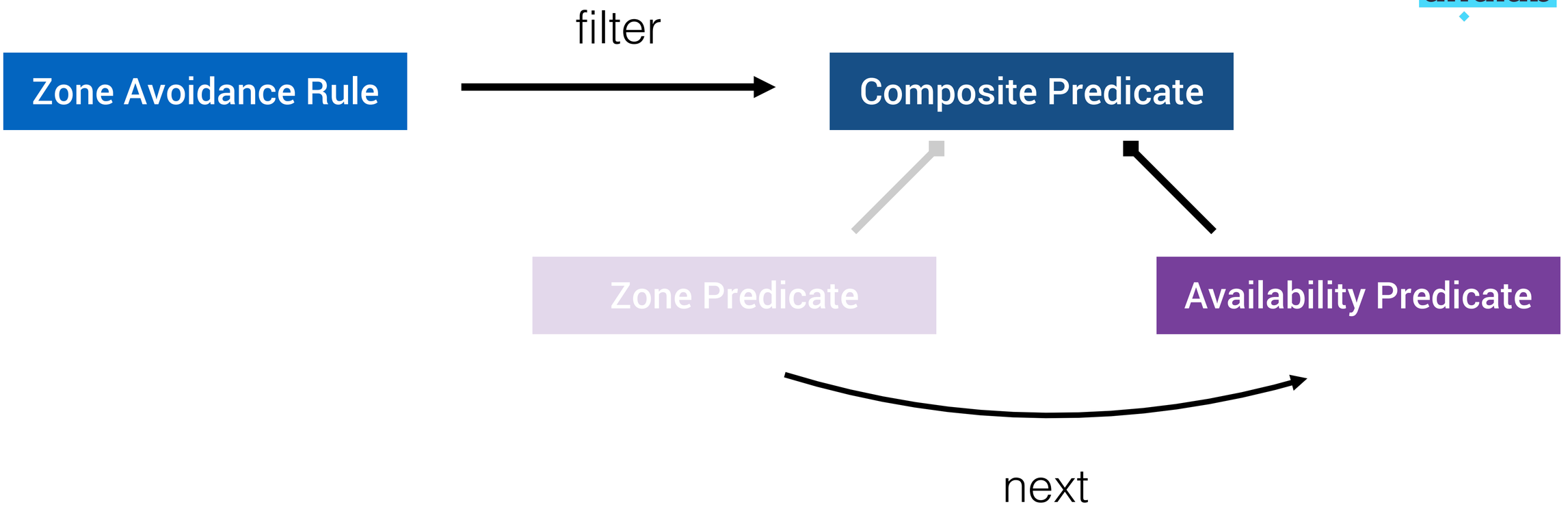

```
CompositePredicateBuilder
   int minimalFilteredServers = 1
   float minimalFilteredPercentage = 0


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

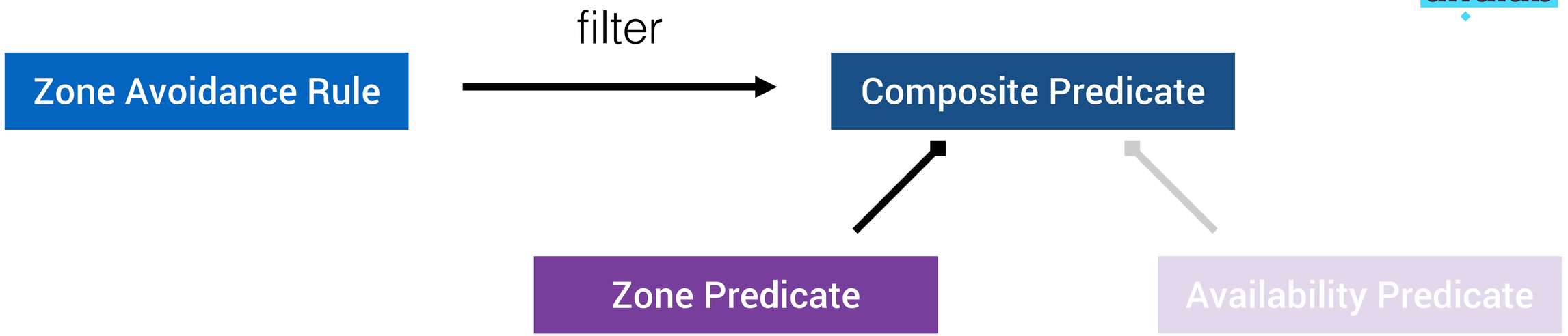



```
CompositePredicateBuilder
   int minimalFilteredServers = 1
   float minimalFilteredPercentage = 0

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

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

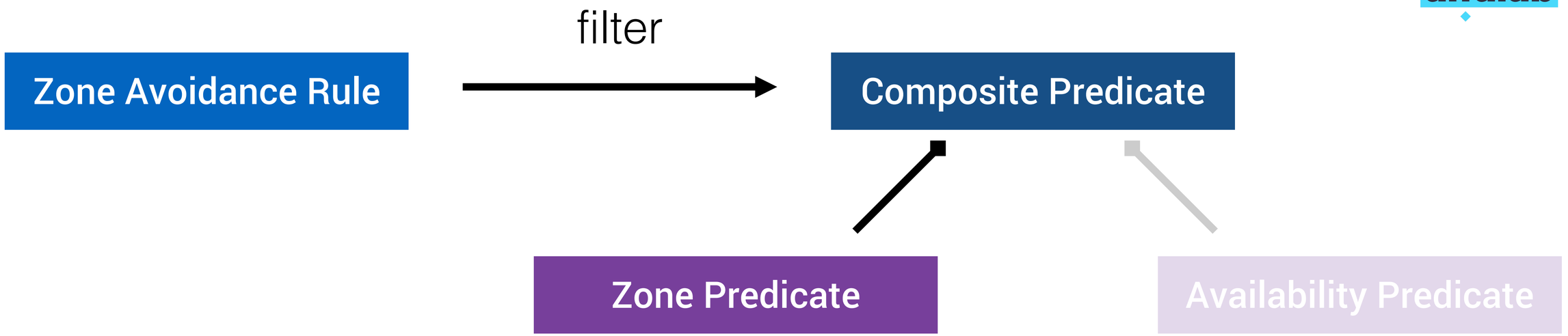



```
CompositePredicateBuilder
   int minimalFilteredServers = 1
   float minimalFilteredPercentage = 0

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

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

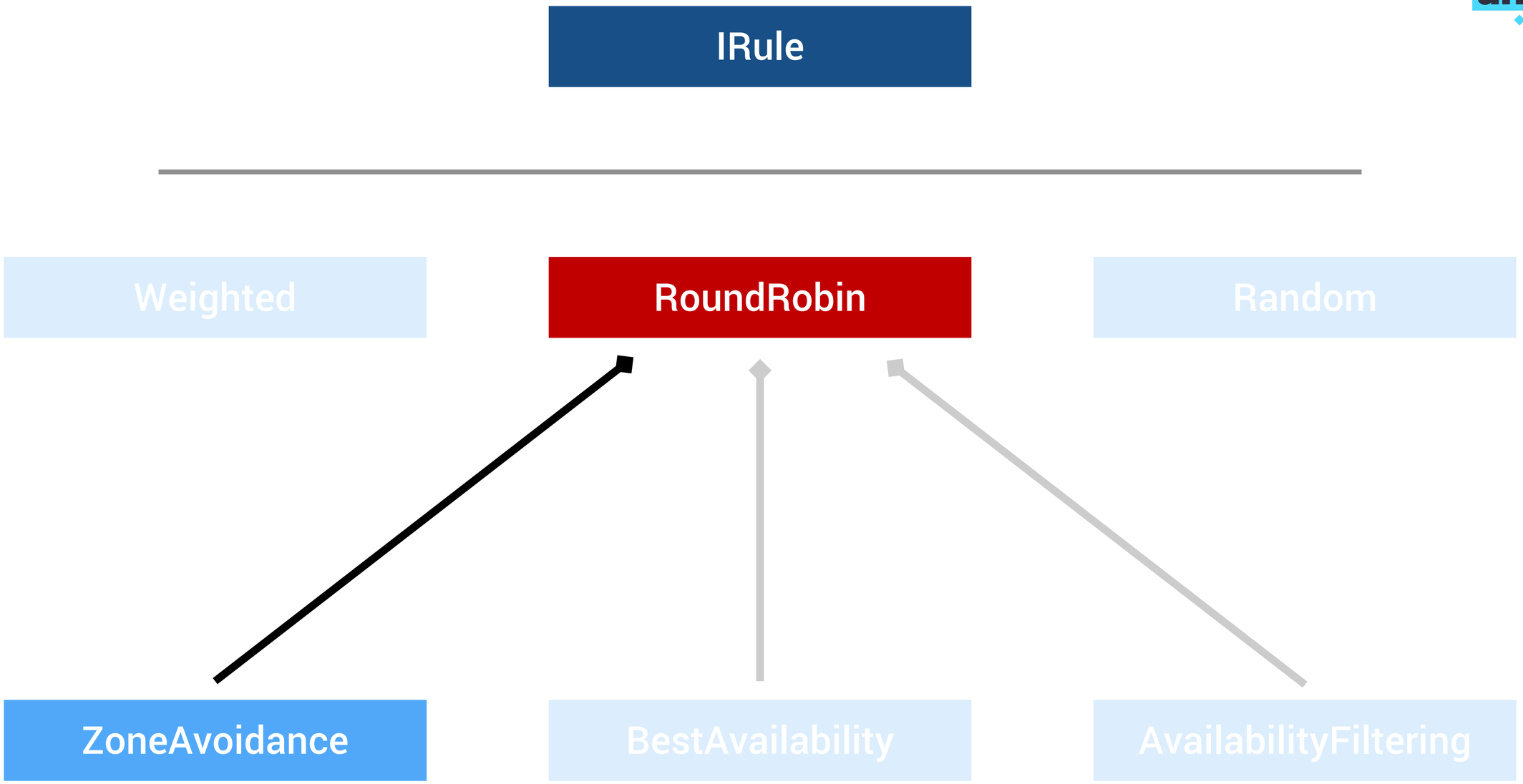

filter
Zone Avoidance Rule
Composite Predicate
Zone Predicate
Availability Predicate
next



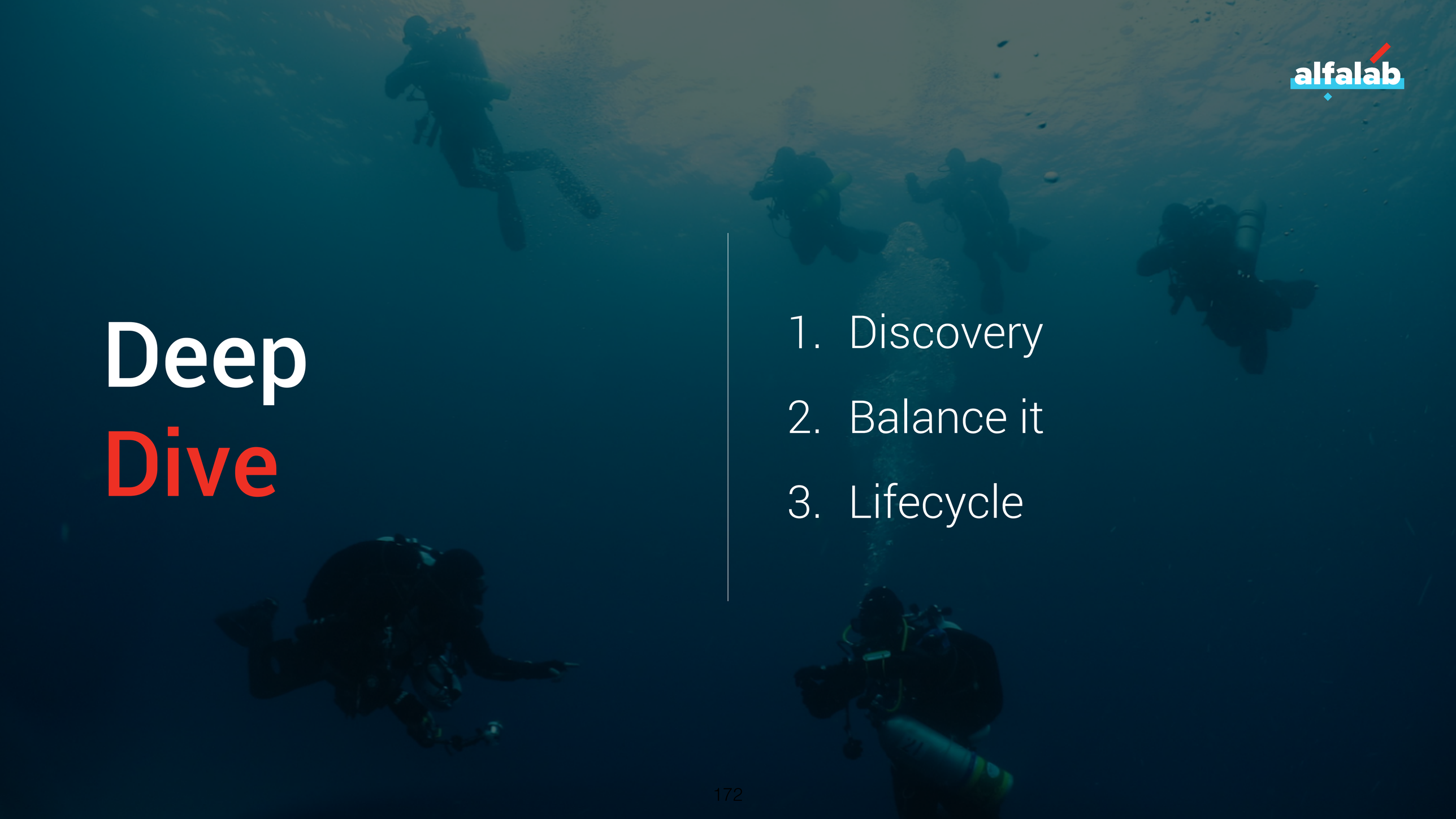

```
triggeringLoadPerServerThreshold: 0.2
avoidZoneWithBlackoutPercetage: 0.99999d

worstZone -> loadPerServer > max(loadPerServer) &&
             loadPerServer > triggeringLoadPerServerThreshold
zones.remove(worstZone)
```




```
triggeringLoadPerServerThreshold: 0.2
avoidZoneWithBlackoutPercetage: 0.99999d

zoneToAvoid -> circuitBreakerTrippedCount / instanceCount >=
                                        avoidZoneWithBlackoutPercetage
zones.remove(zoneToAvoid)
```

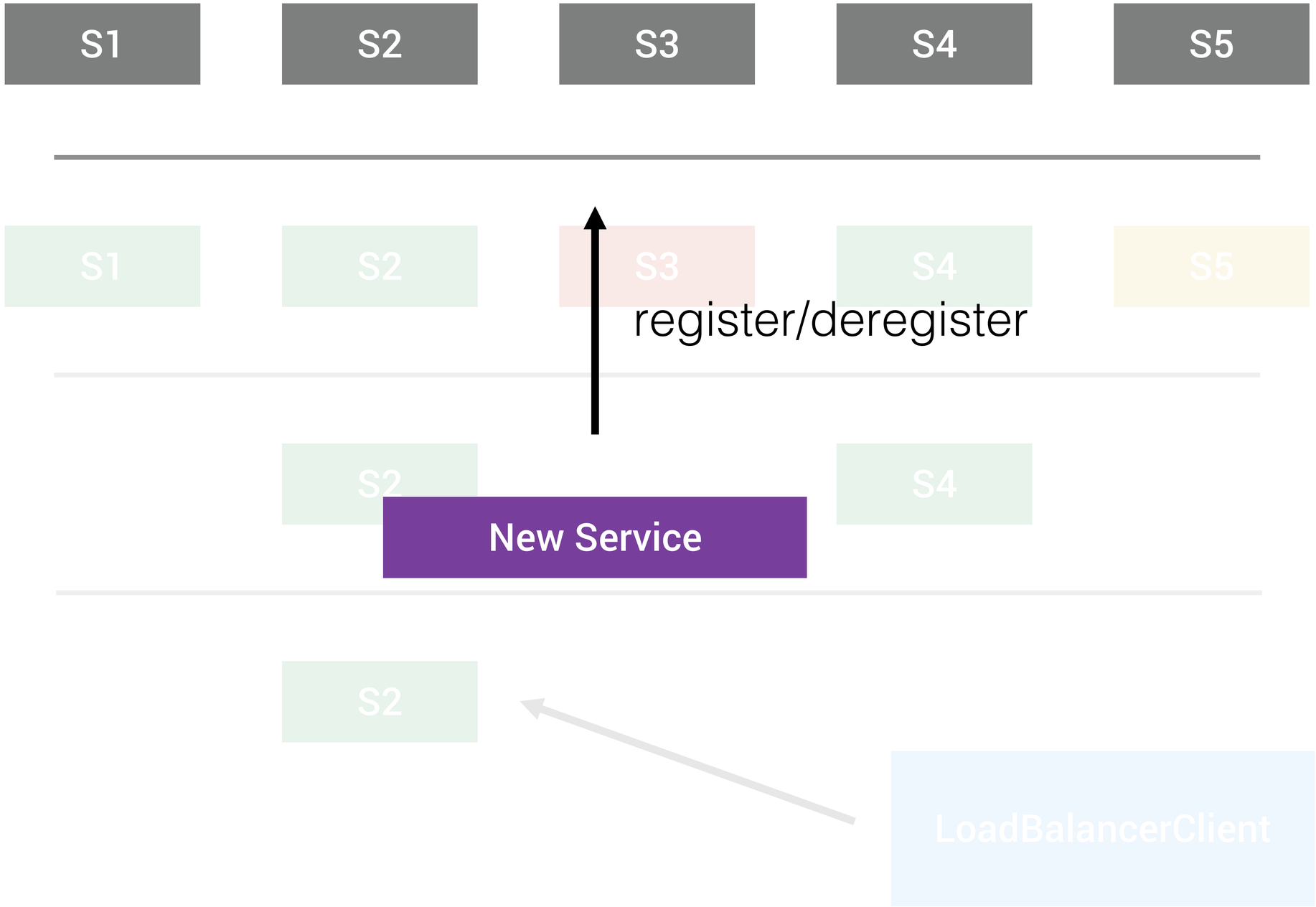

IRule
Weighted
RoundRobin
Random
ZoneAvoidance
BestAvailability
AvailabilityFiltering


alfalab
If you don’t use zones
then don’t use default rule



Deep Dive

1. Discovery
2. Balance it
3. Lifecycle


S1
S2
S3
S4
S5
S1
S2
S3
S4
S5
register/deregister
S2
S4
New Service
S2
LoadBalancerClient

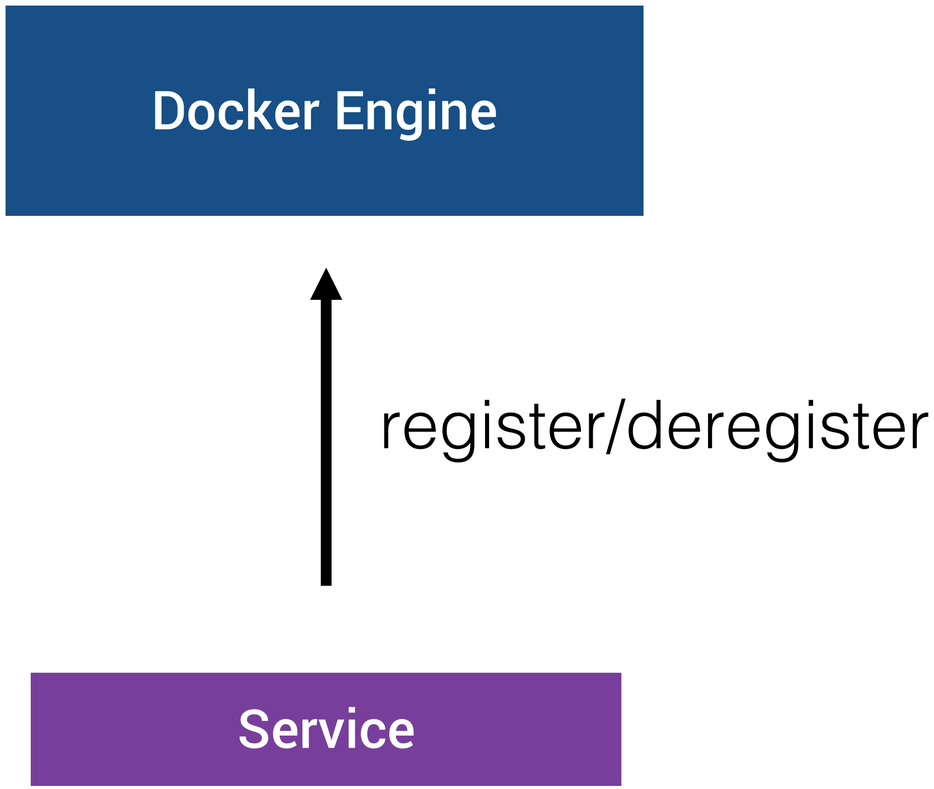
How to do it?

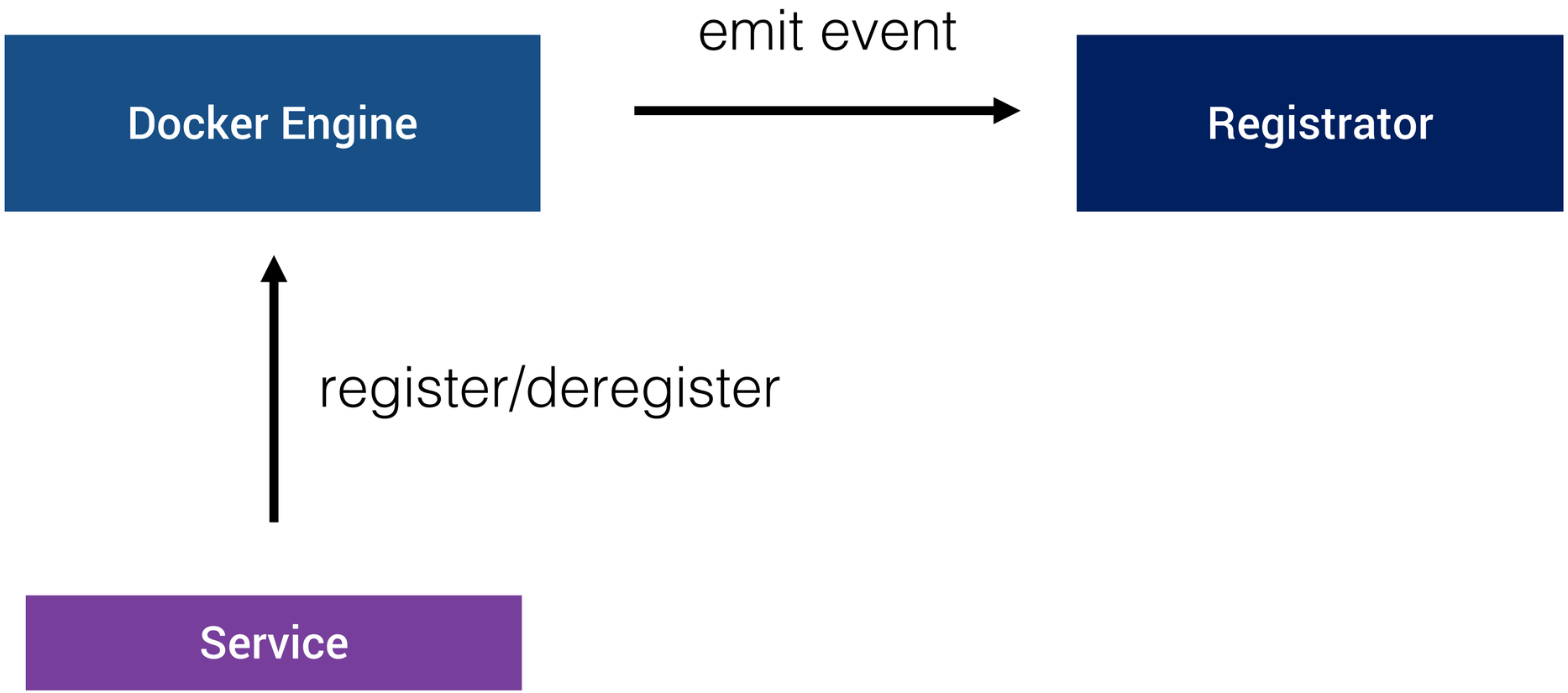
3rd party registration pattern

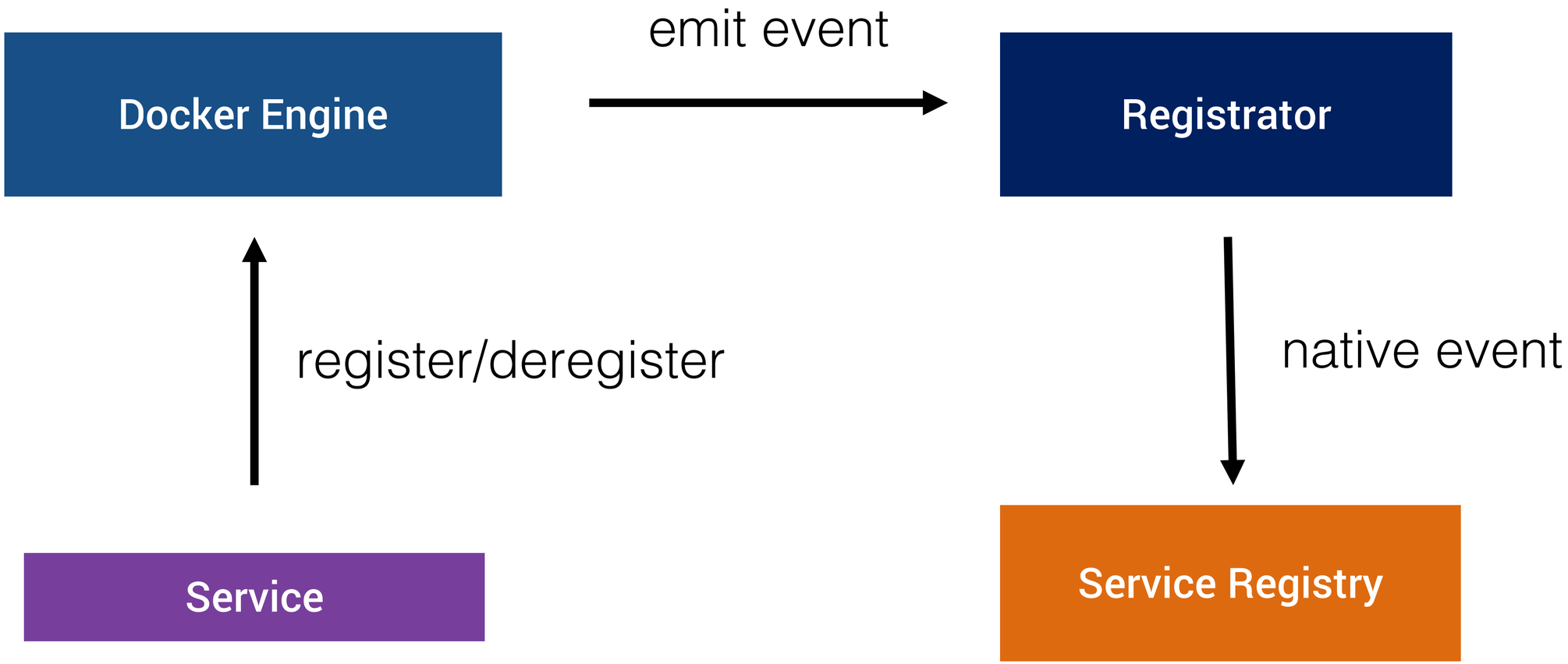
- dynamic
- static

Self-registration pattern (Lifecycle)

Docker Engine


Docker Engine
register/deregister
Service

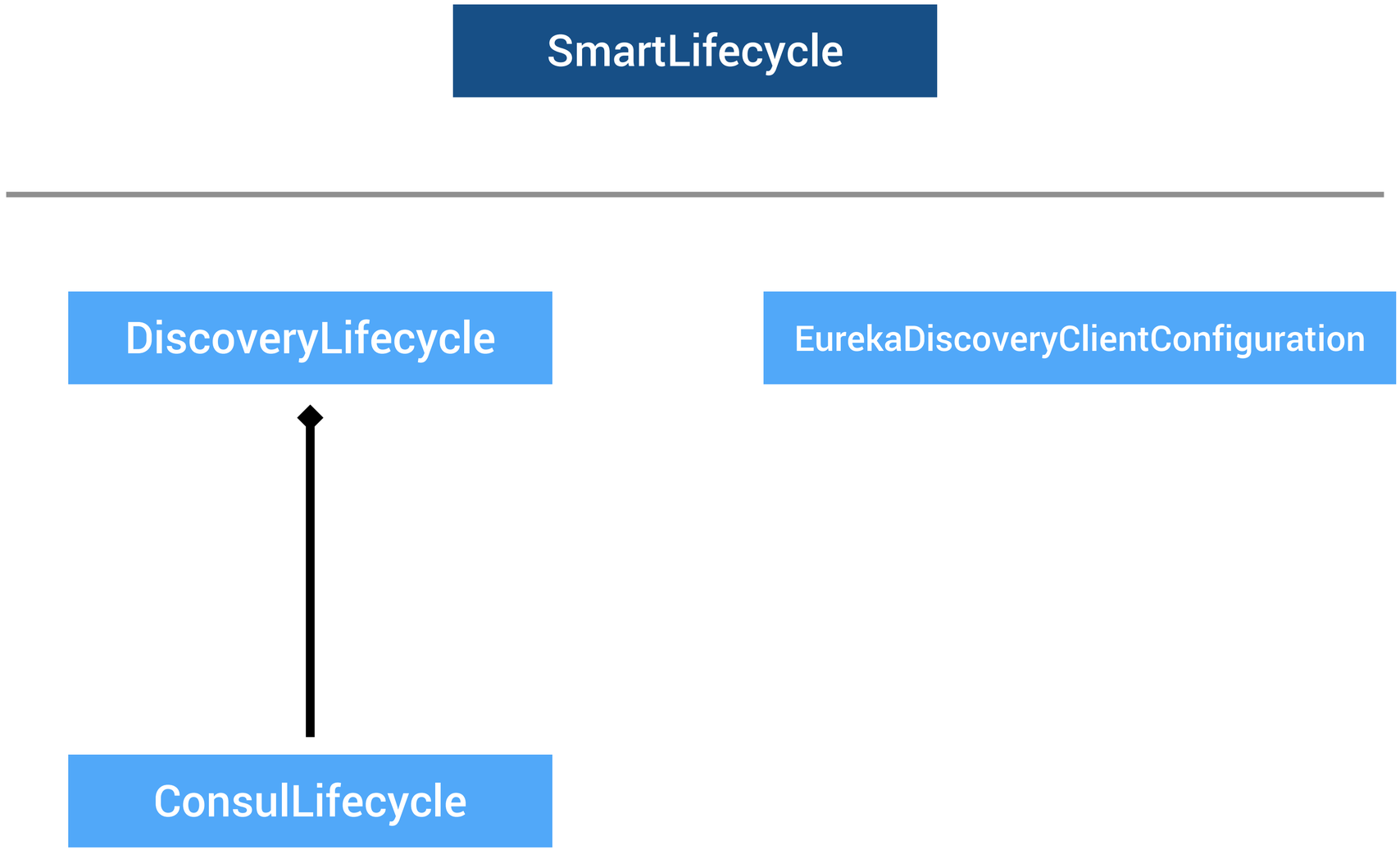

Docker Engine
emit event
Registrator
register/deregister
Service

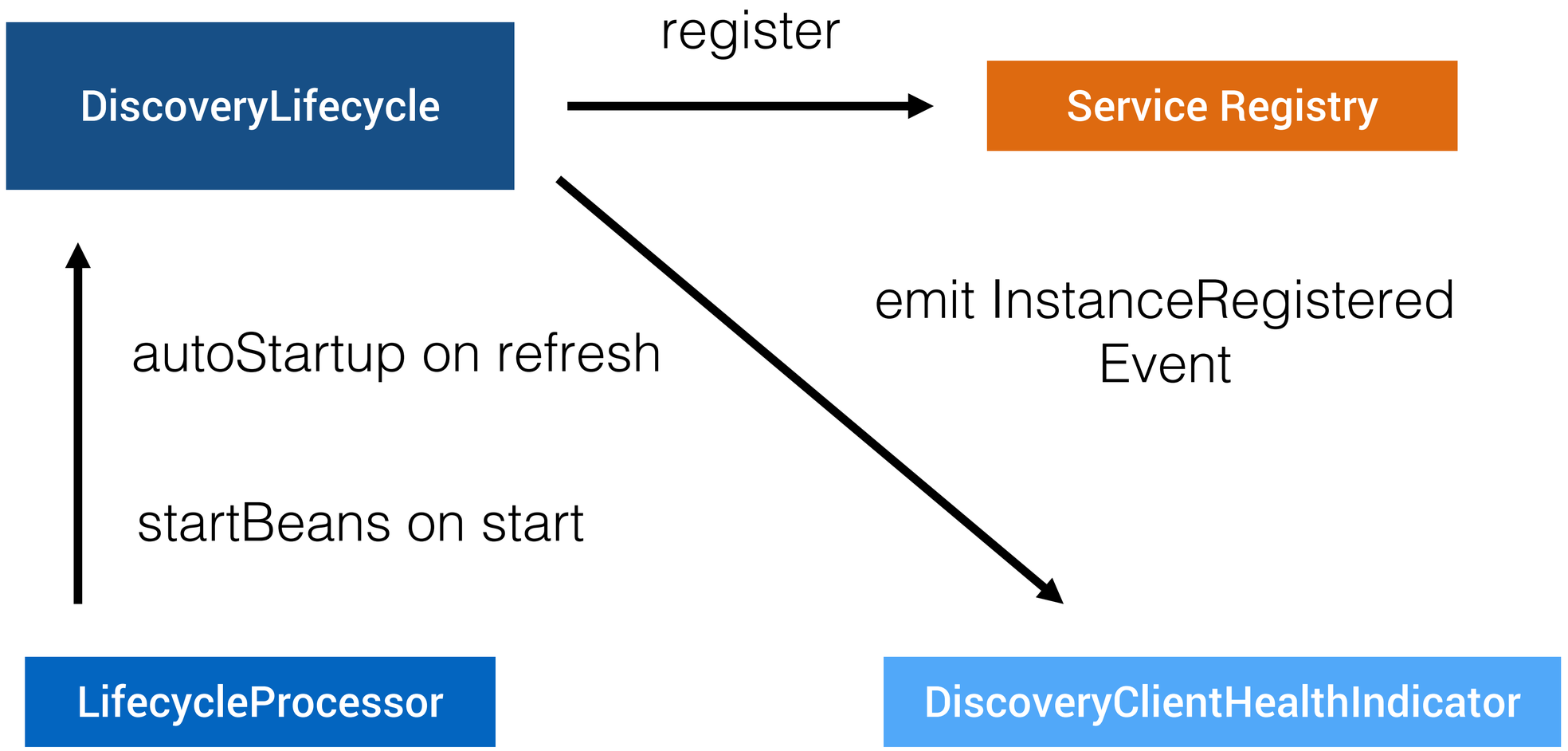

Docker Engine
emit event
Registrator
register/deregister
native event
Service
Service Registry

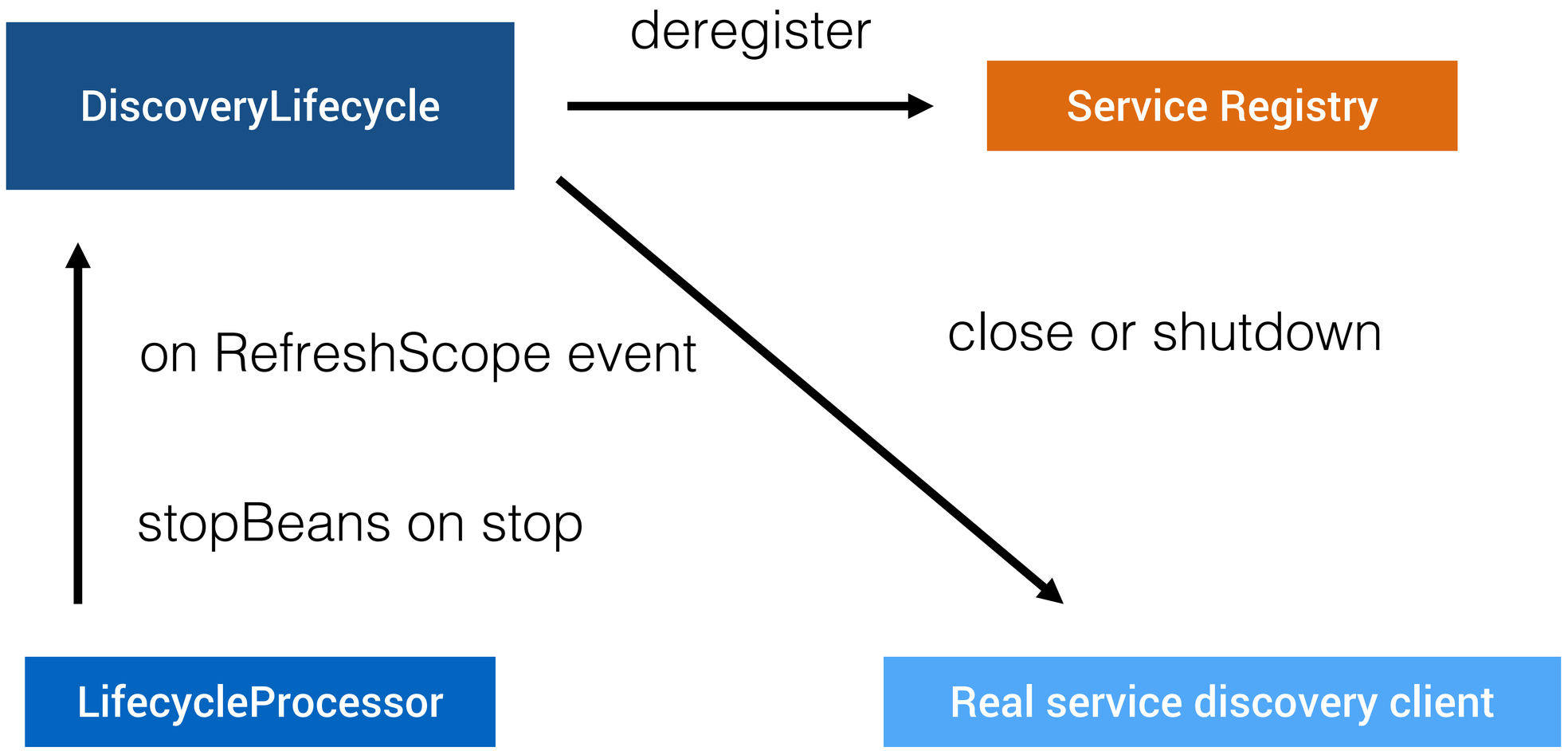
What is lifecycle?

Allow automatically register and deregister service in service registry

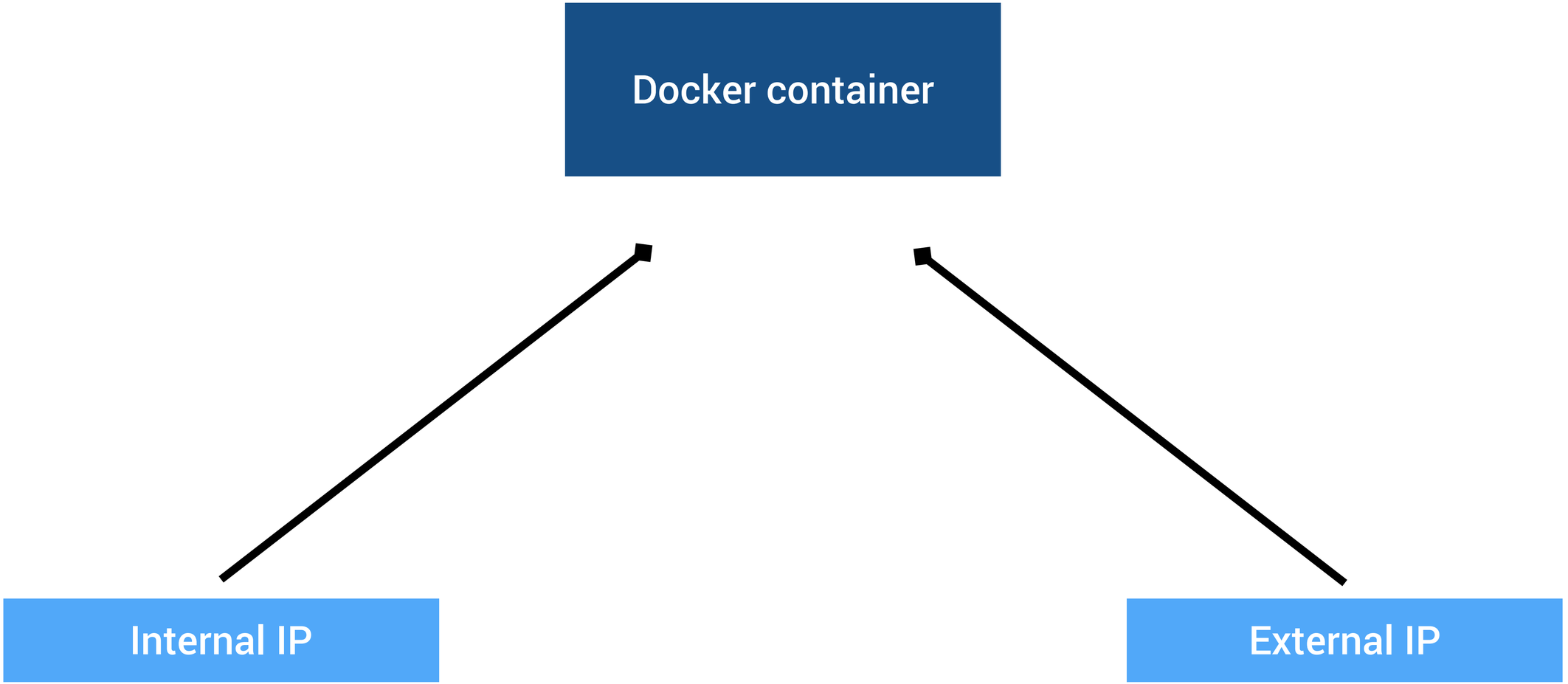
Implements Spring Lifecycle interface


SmartLifecycle
DiscoveryLifecycle
EurekaDiscoveryClientConfiguration
ConsulLifecycle


DiscoveryLifecycle
register
Service Registry
emit InstanceRegistered Event
autoStartup on refresh
startBeans on start
LifecycleProcessor
DiscoveryClientHealthIndicator


DiscoveryLifecycle
deregister
Service Registry
on RefreshScope event
close or shutdown
stopBeans on stop
LifecycleProcessor
Real service discovery client

Something about docker


Docker container
Internal IP
External IP

> ifconfig

```
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
en1: flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
en2: flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
p2p0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
awdl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1484
bridge0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
utun0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1500
```

> ifconfig

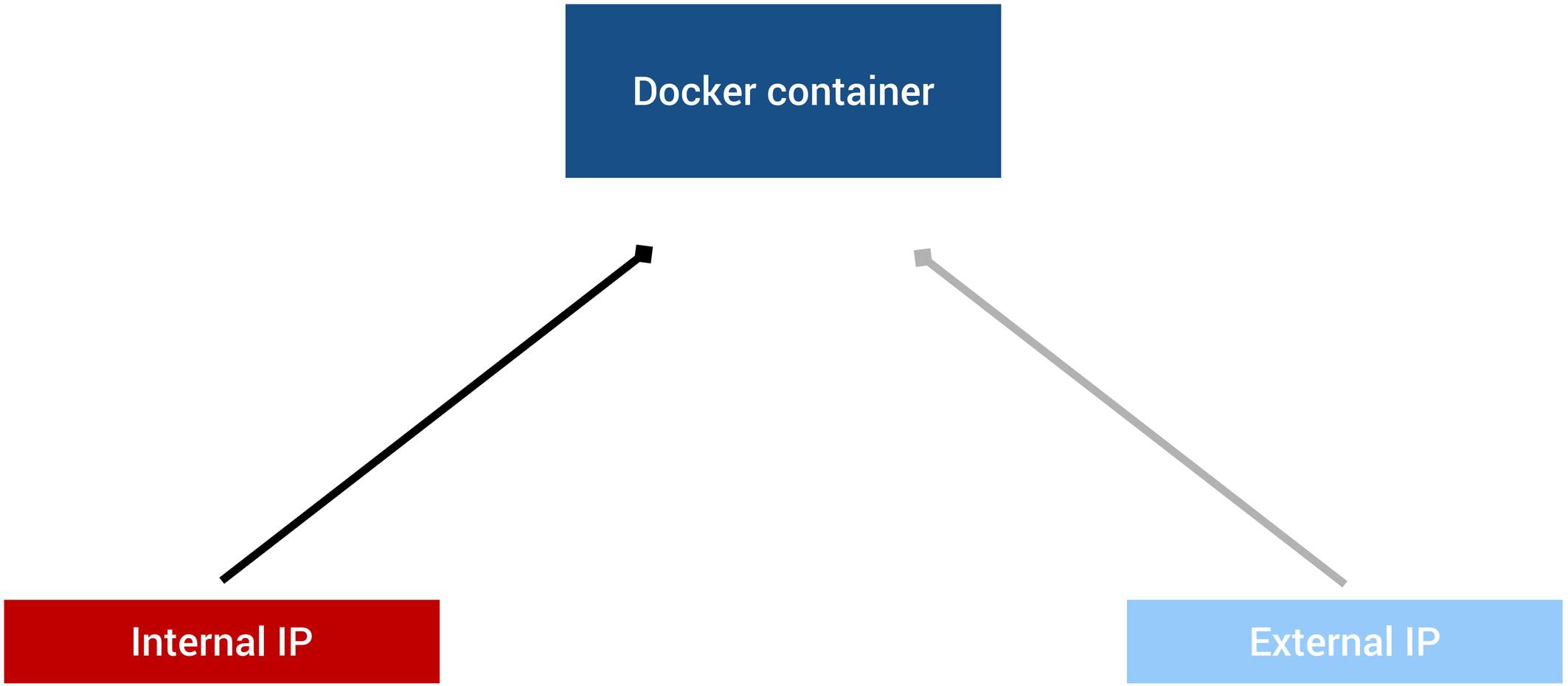
```
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
en1: flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
en2: flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
p2p0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
awdl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1484
bridge0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
utun0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1500
```

?

> ifconfig

```
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
en1: flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
en2: flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
p2p0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
awdl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1484
bridge0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
utun0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1500
```

Low index

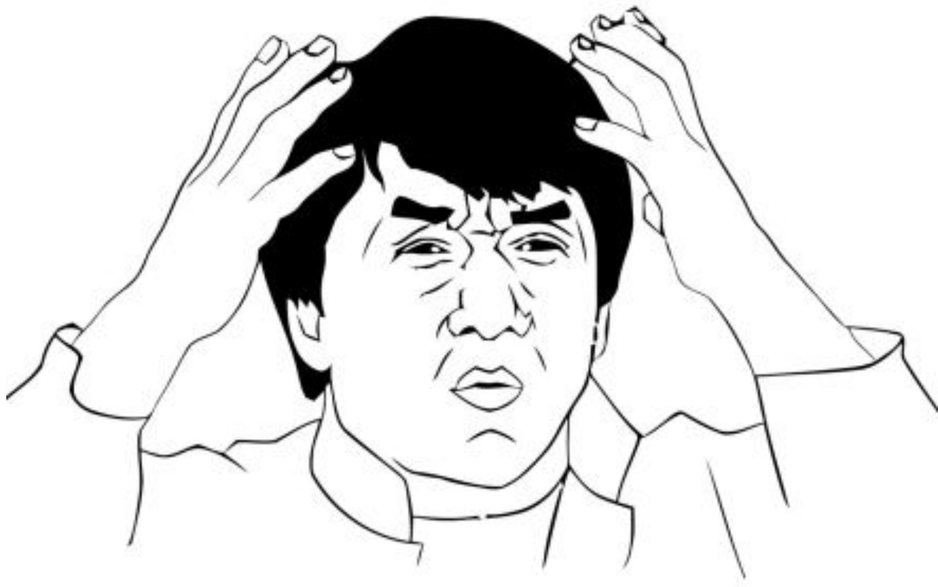

Docker container
Internal IP
External IP

```
spring:
  cloud:
    consul:
      discovery:
        lifecycle:
          enabled: false
```

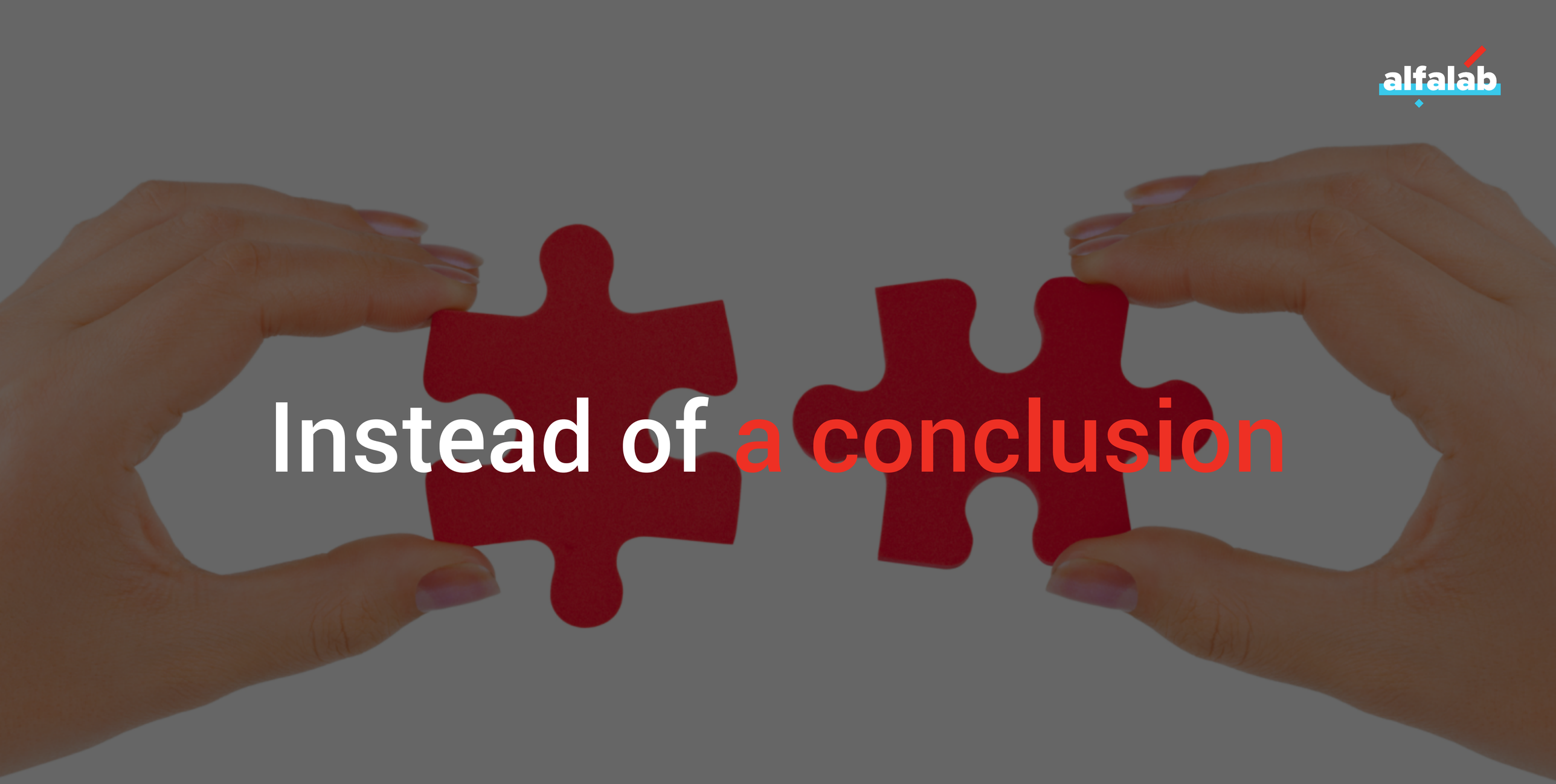
Registrator

```
spring:
   cloud:
      inetutils:
        ignored-interfaces:
          - vbox*
          - bridge*
          - lo*
```

```
spring:
  cloud:
    inetutils:
      ignored-interfaces:
        - vbox*
        - bridge*
        - lo*
```



```
spring:
  cloud:
    inetutils:
      ignored-interfaces:
        - vbox*
        - bridge*
        - lo*
    consul:
      discovery:
        preferIpAddress: true (false)
```



Instead of a conclusion

Own experience

Own experience

We use both patterns: server and client side

- Consul + Registrator + Nginx
- Marathon + MarathonLB

- Eureka

Own experience

Use client side pattern for API and Edge server

Use server side for NodeJS front apps and non-standard API (Python)

Own experience

Spring Cloud is production-ready and very customizable

Minimum number of settings are needed to be changed, but for better results there are some of them that should be changed

There are some issues that make you sad

alfalab

Service Discovery Patterns
http://microservices.io/patterns/server-side-discovery.html
http://microservices.io/patterns/client-side-discovery.html

Spring Cloud
http://projects.spring.io/spring-cloud

Spring Cloud Marathon (Proof of Concept)
https://github.com/aatarasoff/spring-cloud-marathon

Questions and Answers

Architect @ alfalab

/aatarasoff

/aatarasoff

habrahabr.ru/aatarasoff

developerblog.info