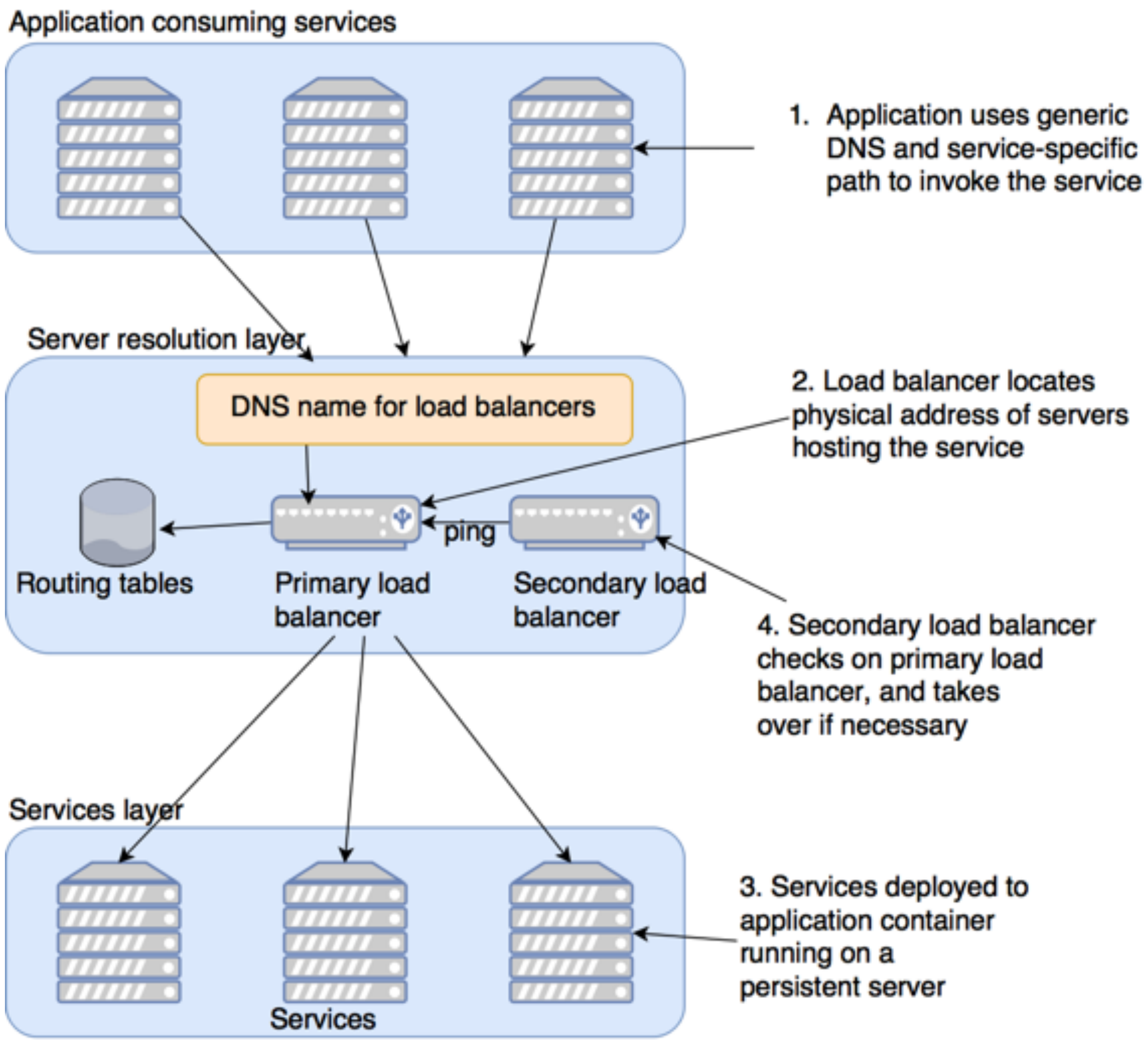
Service Discovery

Chapter Content

- 1. Traditional service location solution model
- 2. Service discovery in the cloud
- 3. Client-side load balancing
- 4. Service discovery using Eureka

- Service discovery is critical to cloud-based microservice for two reasons:
    - It offers the ability to quickly horizontally scale up and down the number of service instances running in an environment
    - It helps increase application resiliency. When an instance fails, it can remove it from the available instance list

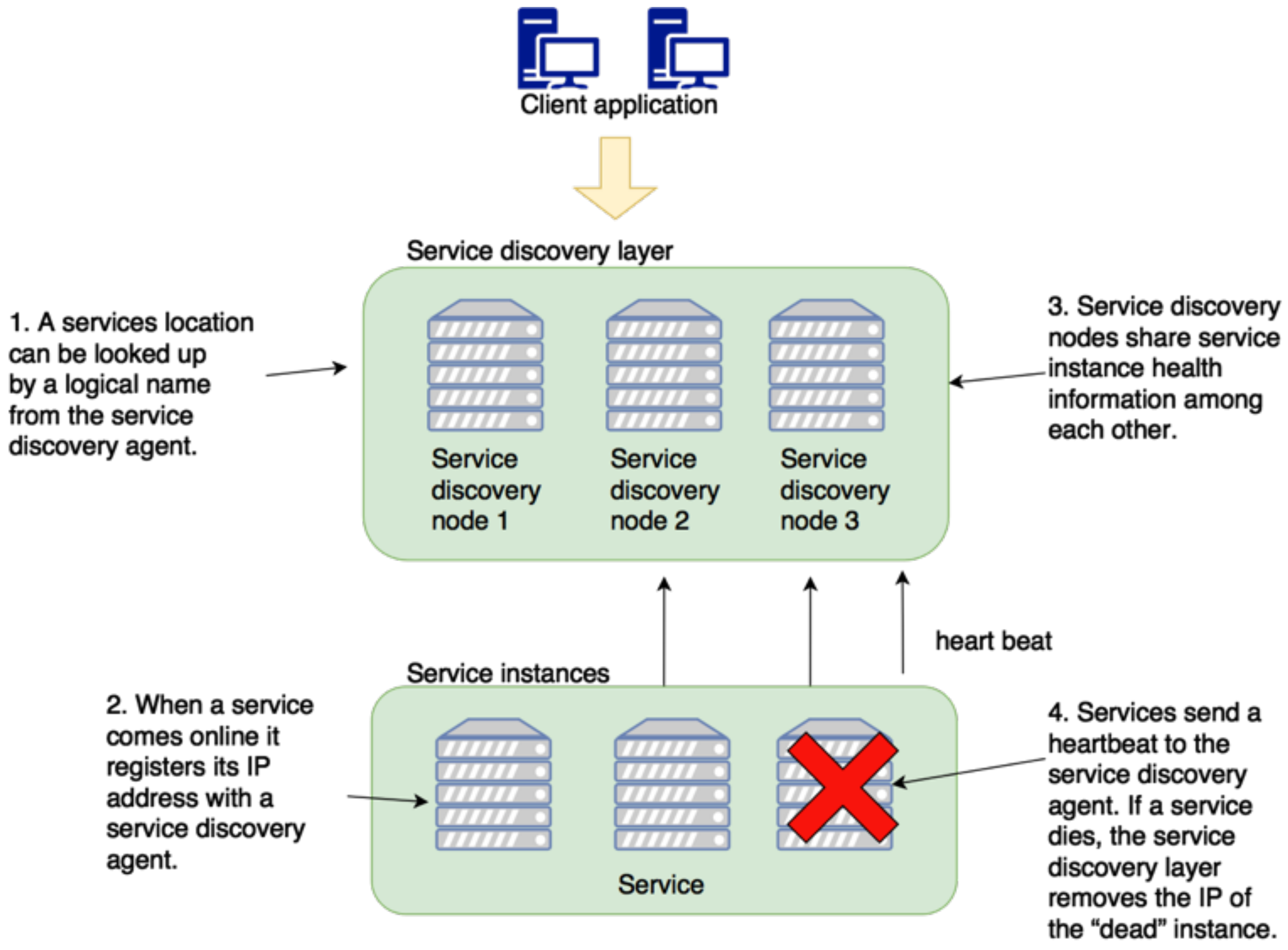
1.Traditional service location solution model


Application consuming services
1. Application uses generic DNS and service-specific path to invoke the service
Server resolution layer
DNS name for load balancers
2. Load balancer locates physical address of servers hosting the service
ping
Routing tables
Primary load balancer
Secondary load balancer
4. Secondary load balancer checks on primary load balancer, and takes over if necessary
Services layer
3. Services deployed to application container running on a persistent server
Services

- The above architecture works well on-premise, but not for cloud-based microservice applications due to:
    - **Single-point failure** - load balancer is a single point of failure for the entire infrastructure
    - **Limited horizontal scalability** - Hardware constraint
    - **Statically managed** - not designed for rapid registration and de-registration of services
    - **Complex** - service consumer requests have to have their requests mapped to the physical services. This is usually done manually.

2. Service discovery in the cloud

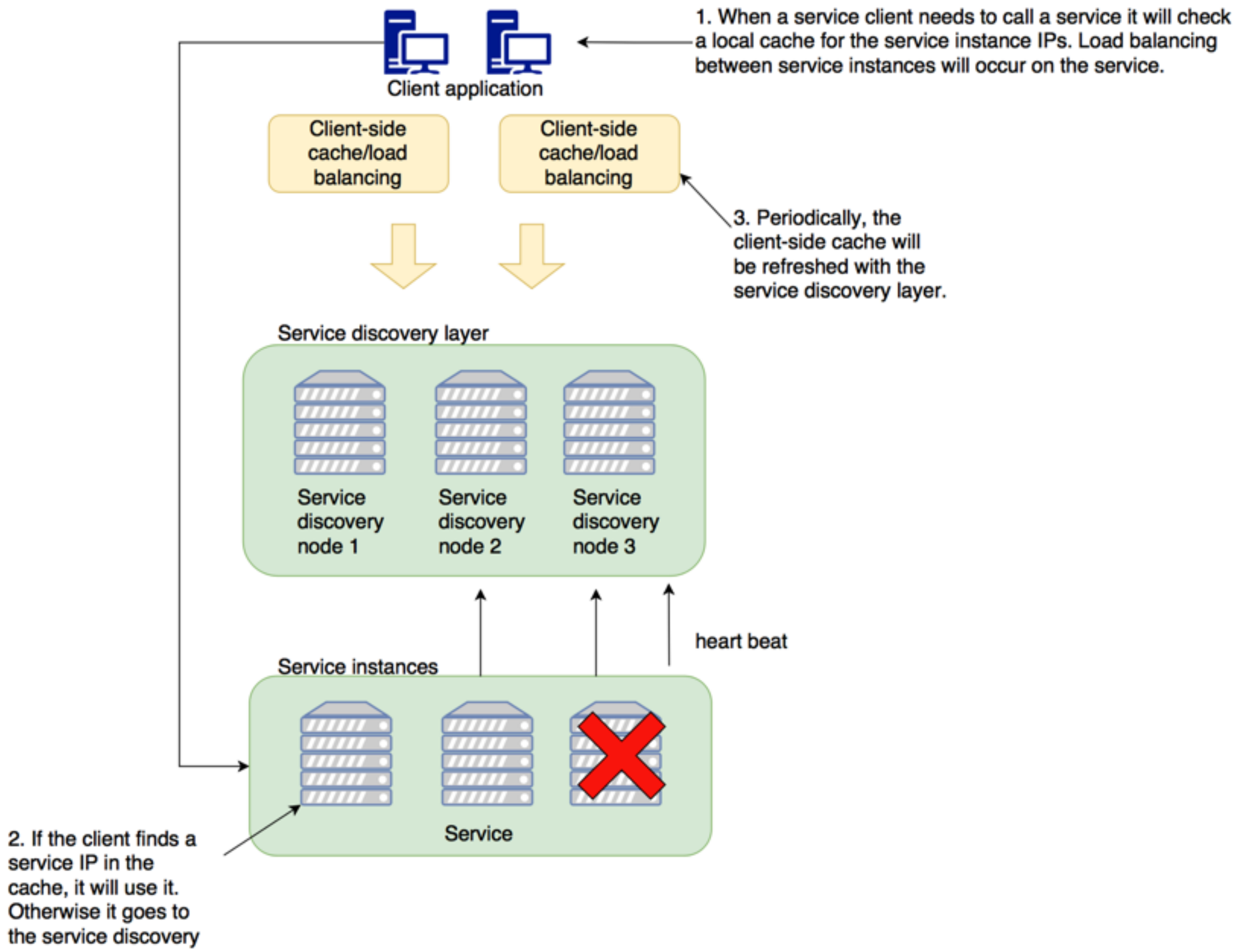
- Four concepts around service discovery architecture:
    - Service registration
    - Client lookup of service address
    - Information sharing
    - Health monitoring


Client application
Service discovery layer
1. A services location can be looked up by a logical name from the service discovery agent.
Service discovery node 1
Service discovery node 2
Service discovery node 3
3. Service discovery nodes share service instance health information among each other.
heart beat
Service instances
2. When a service comes online it registers its IP address with a service discovery agent.
Service
4. Services send a heartbeat to the service discovery agent. If a service dies, the service discovery layer removes the IP of the "dead" instance.

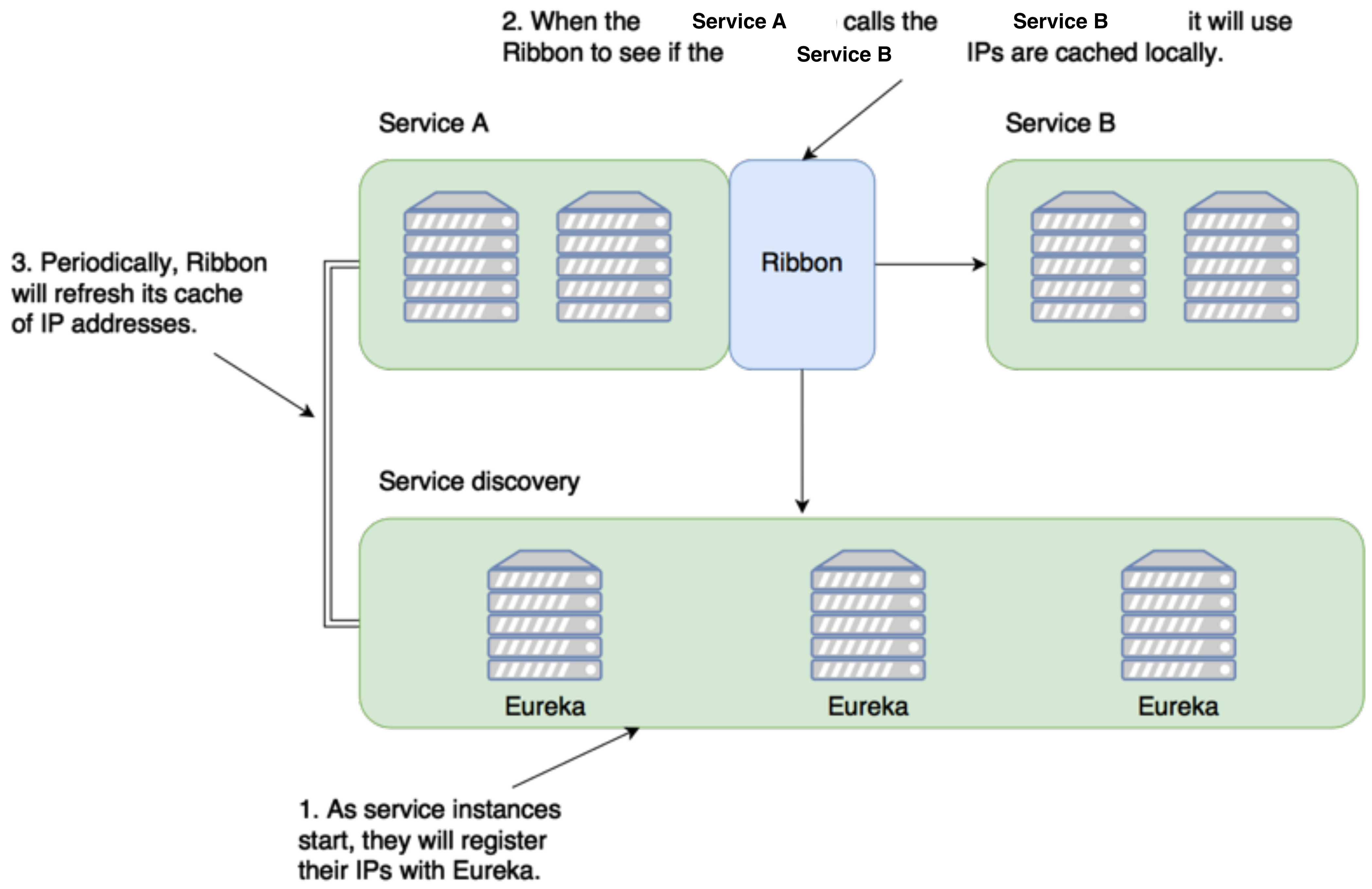
- Discovery agent instances are usually unique and don’t have a **load balancer** that sits in front of them
- As service instances start up, they’ll register their physical location, path, and port that they can be accessed by with one or more service discovery instances
- While each instance of service will have a **unique IP address** and port, each service instance that comes up will register under the **same service ID**
- A service ID is a key that uniquely identifies a group of the same service instances
- A service will usually **only** register with **one** service discovery service instance. Most service discovery implementations use a peer-to-peer model of data propagation where the data around each service instance is communicated to all the other nodes in the cluster
- Each service instance will push to or have pulled from its status by discovery agent. Any services failing to return a good health check will be removed from the pool of available service instances

3. Client-side load balancing

- There are multiple ways to “discover” a service:
    - A client can rely solely on the service discovery engine to resolve service locations each time a service is called. With this approach, the service discovery engine will be invoked every time a call to a registered microservice instance is made.This approach is brittle as it completely depends on the service discovery engine
    - A more robust approach is to use **client-side load balancing**


1. When a service client needs to call a service it will check a local cache for the service instance IPs. Load balancing between service instances will occur on the service.
Client application
Client-side cache/load balancing
Client-side cache/load balancing
3. Periodically, the client-side cache will be refreshed with the service discovery layer.
Service discovery layer
Service discovery node 1
Service discovery node 2
Service discovery node 3
heart beat
Service instances
Service
2. If the client finds a service IP in the cache, it will use it. Otherwise it goes to the service discovery

4. Service discovery using Eureka


2. When the Service A calls the Service B it will use Ribbon to see if the Service B IPs are cached locally.
Service A
Service B
Ribbon
3. Periodically, Ribbon will refresh its cache of IP addresses.
Service discovery
Eureka
Eureka
Eureka
1. As service instances start, they will register their IPs with Eureka.