# Breaking the Monolith

## An example of micro-services for sync and share based on ClawIO



Hugo Gonzalez Labrador CERN IT-ST CS3 SurfSARA 2017

# Outline

- Introduction
- The monolith
- Micro-Services Architecture Requirements
- ClawIO
- Monolith vs MSA
- Conclusion

# Introduction

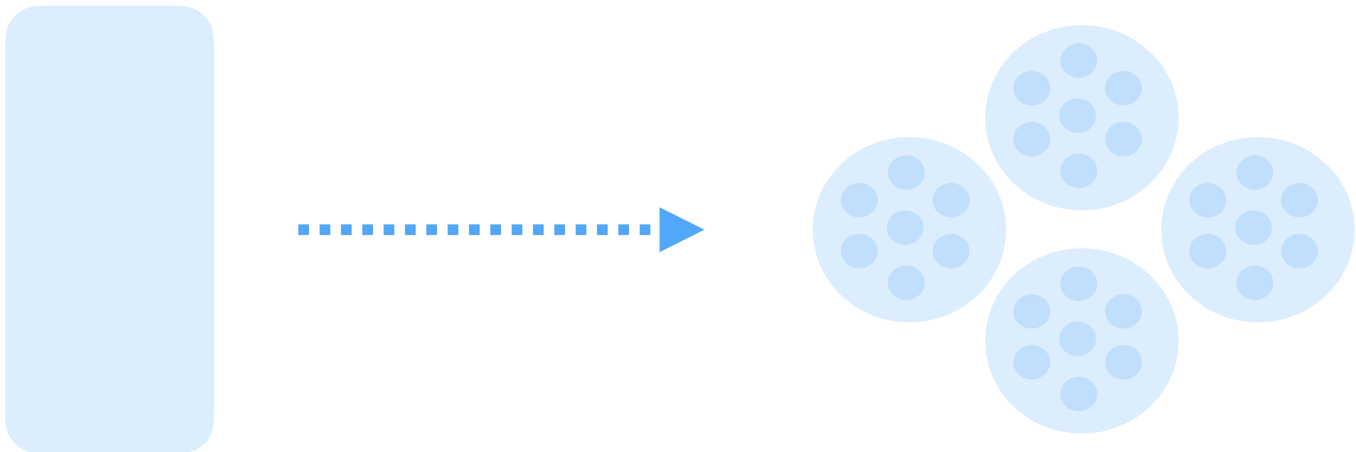
- Micro services is an architecture that structures the application as a set of loosely coupled, collaborating services using lightweight network protocols.

- Has been adopted successfully by big companies

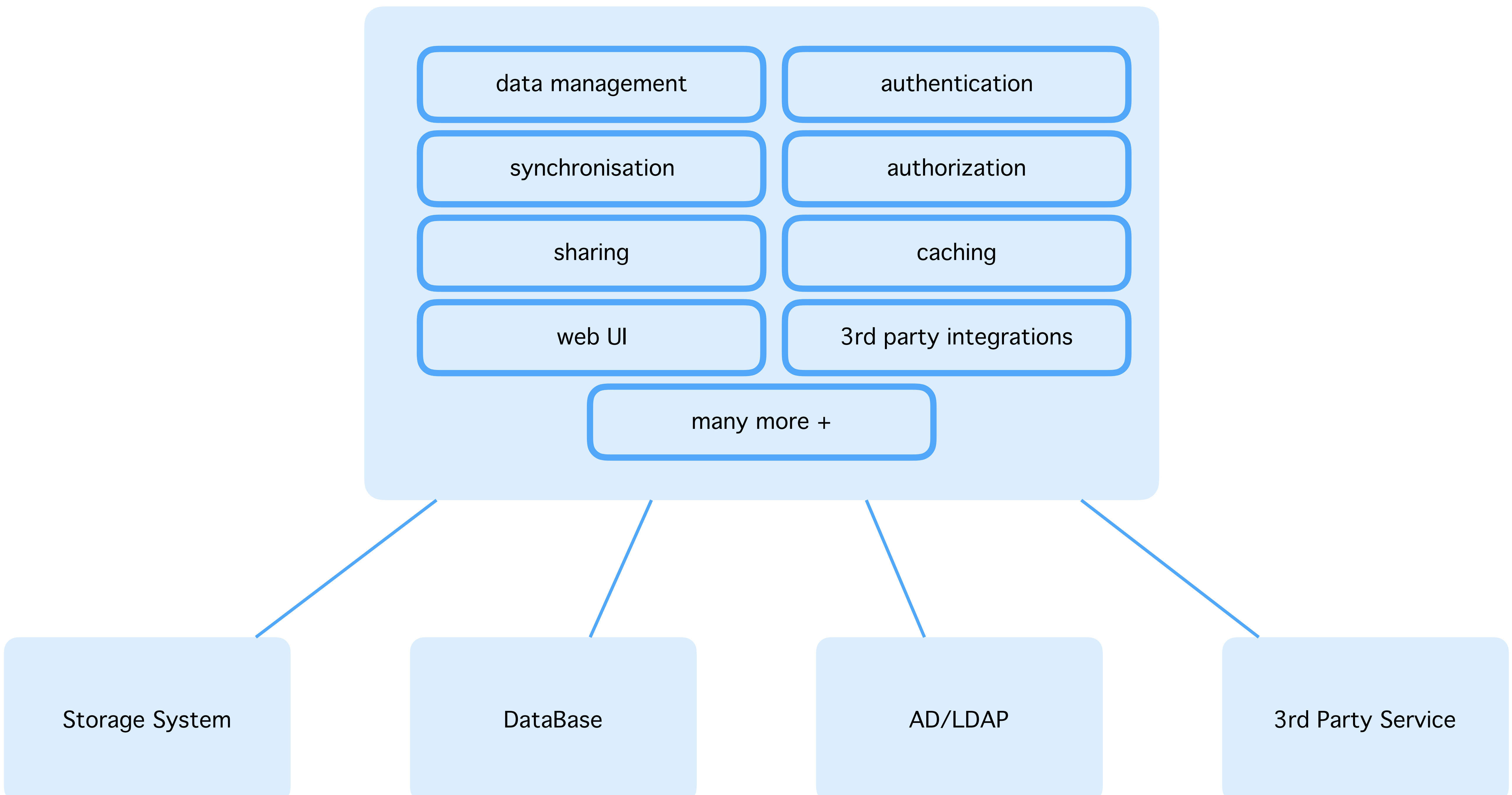
- Lot of hype

# Objective

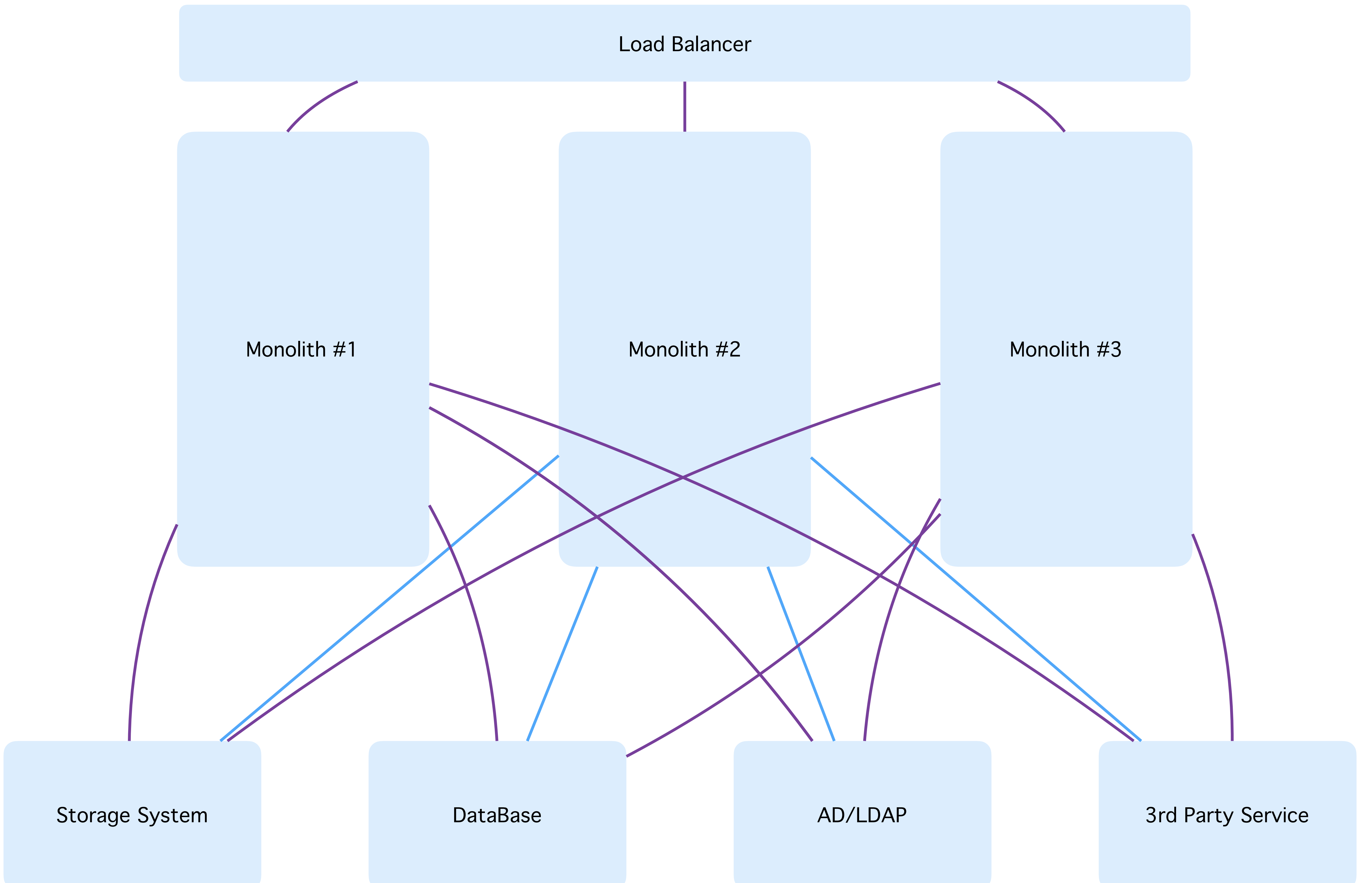
- Many of today open source sync and share platforms run as monolithic applications
- Is it feasible to move to a MSA for sync and share? What are the costs? Are there any benefits?
- Let's build something to see



# The Monolithic Sync and Share Approach



# Scaling the Monolith



# Pros and Cons of the Monolith

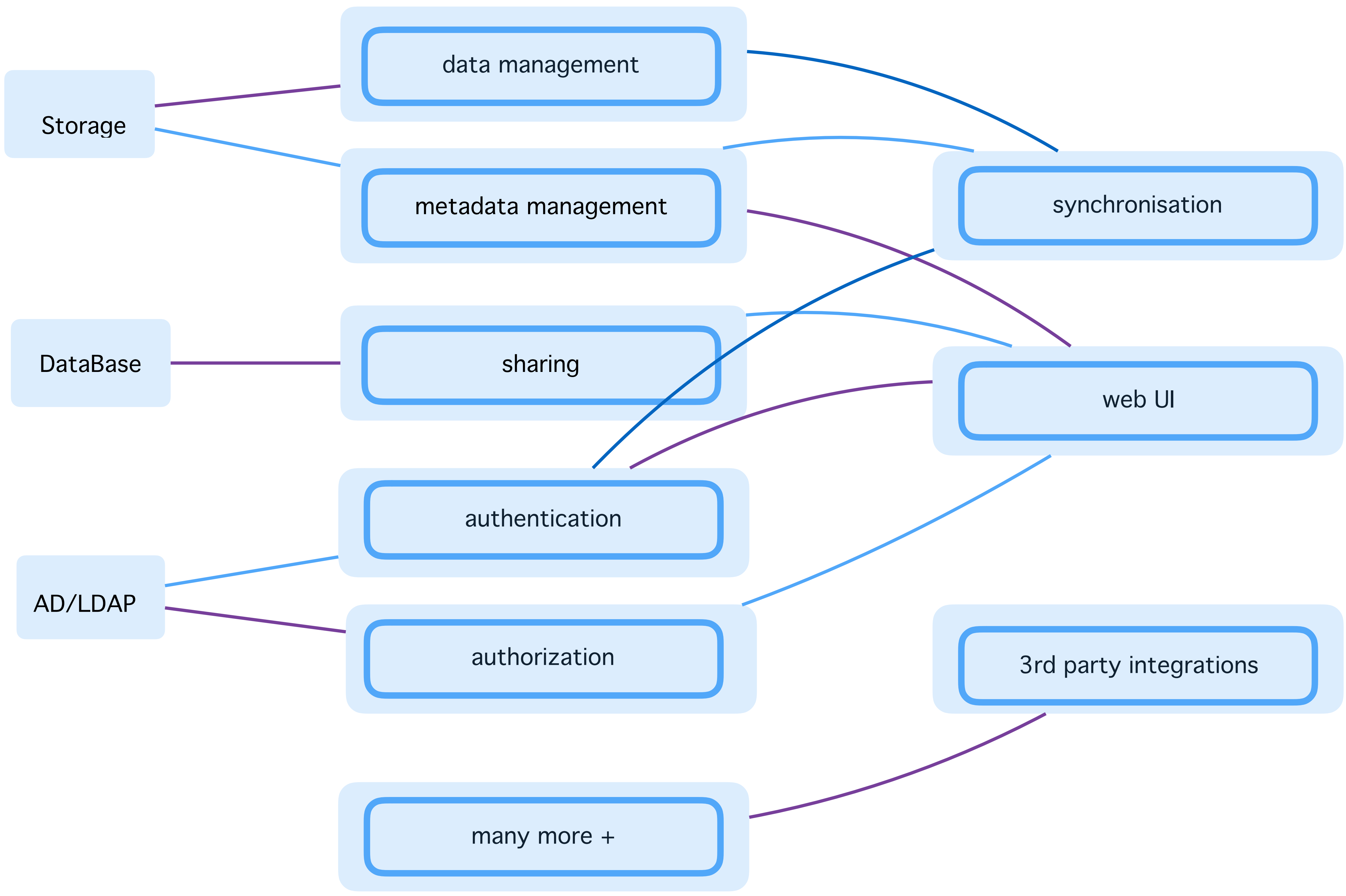
- No network penalty for component interactions
- Simpler deployments (self-contained)
- Cross-boundary overhead
- Complexity (large codebase)
- Different resource requirements (memory, cpu, IO)
- Adhering to agile principles is difficult
- Larger deployment times
- Requires a long commitment to a technology stack
- Expensive to adopt new technologies (rewriting is expensive)

# Anecdote

In August of 2008, Netflix experienced a major database corruption for three days.
That day they realized that had had to move away from vertically scaled single point of failure, towards highly reliable, horizontally scalable, distributed systems in the cloud

https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration

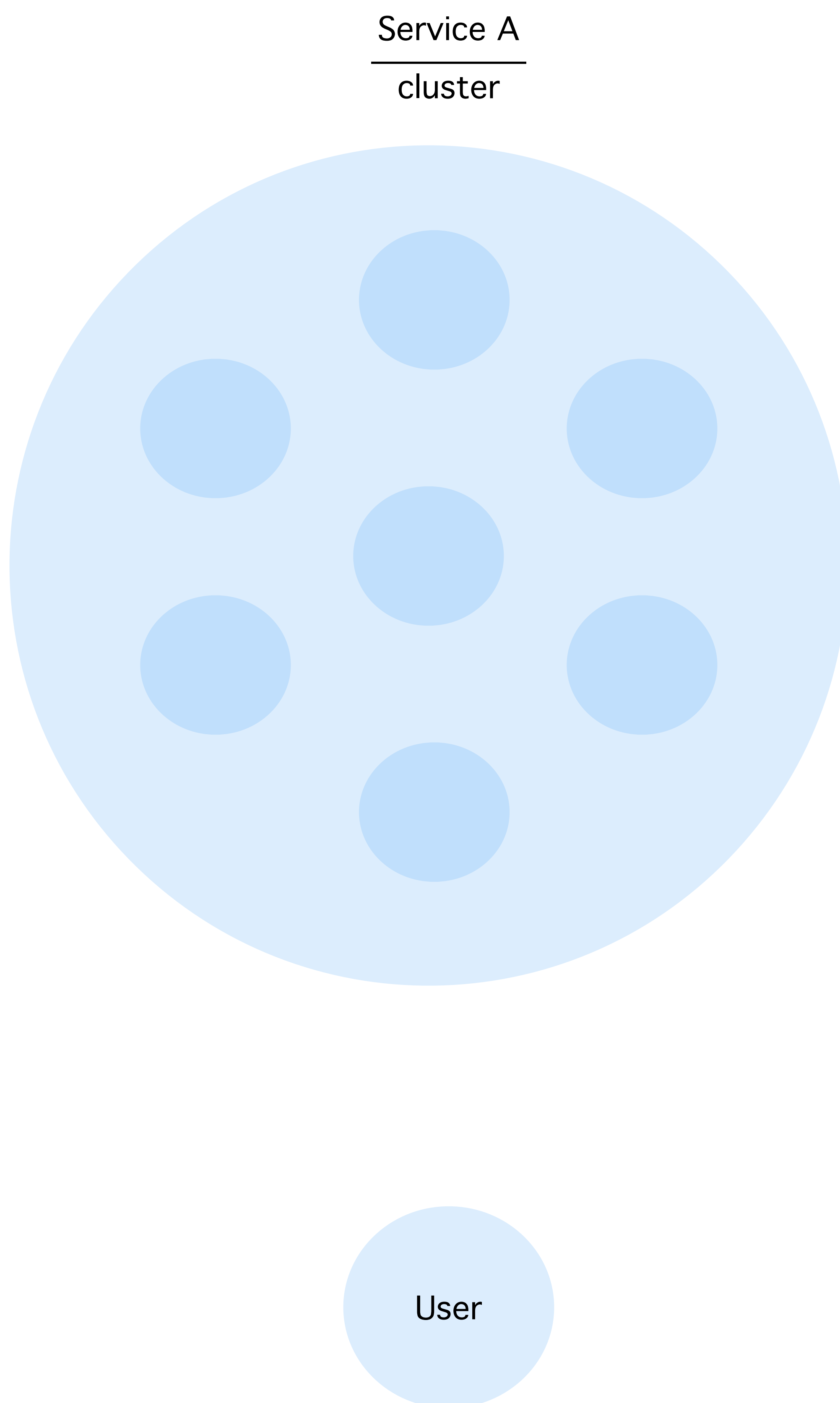
# Micro-Services-Architecture for Sync And Share



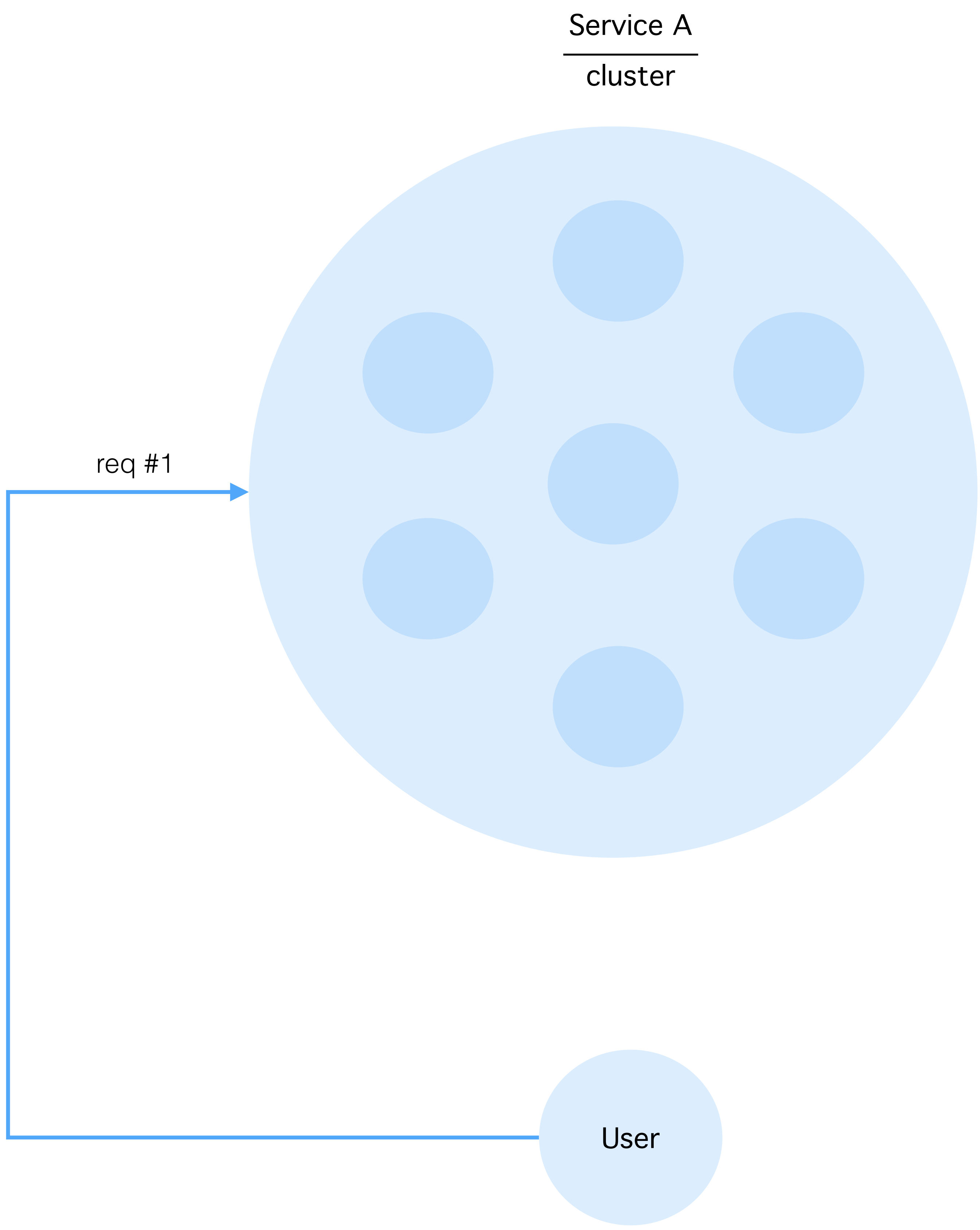
# Some Key Technical Requirements

- Stateless is key for scaling out
- Service Discovery and Registration
- Health Checks
- Distributed Tracing

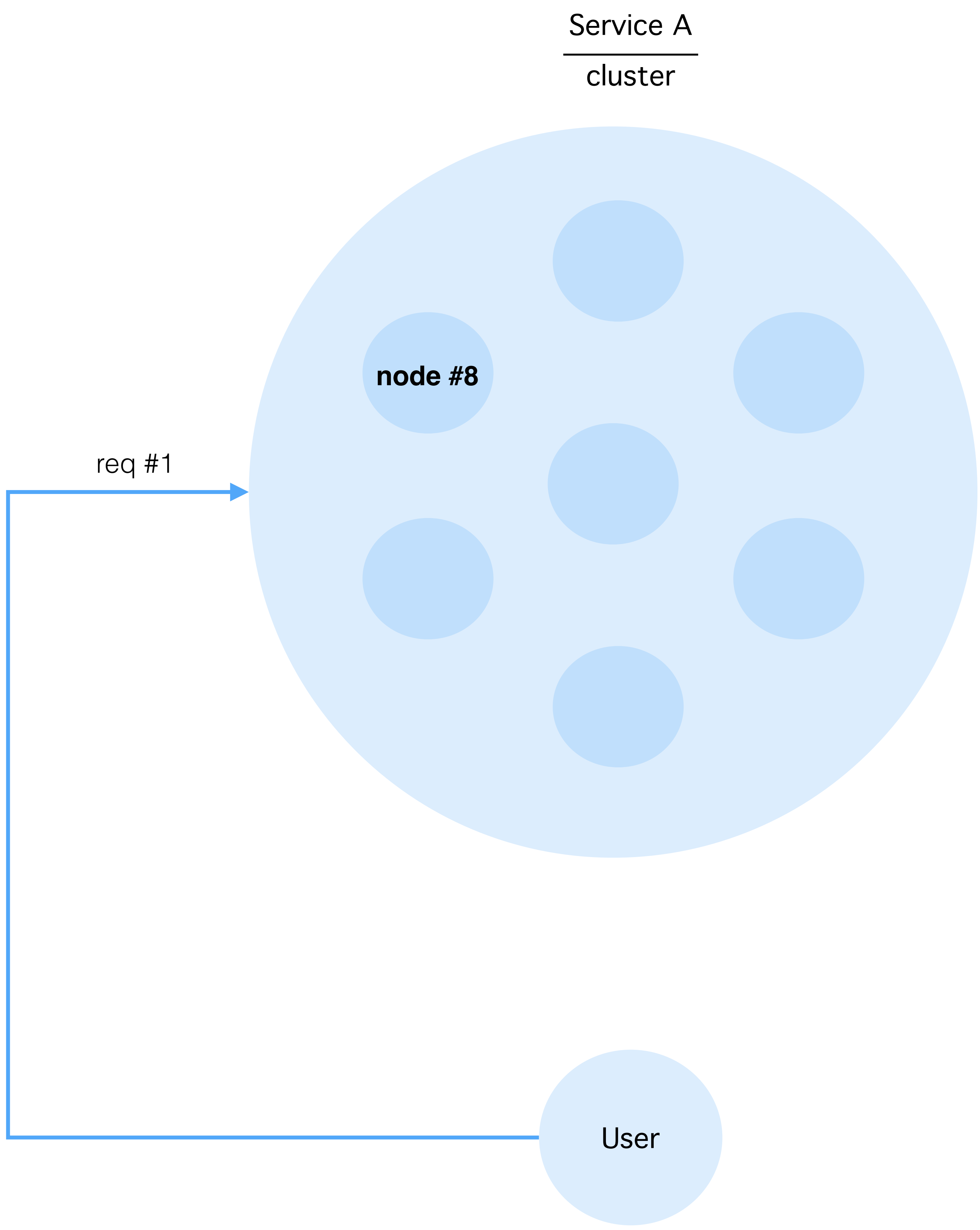
# Tech Req (Stateless => Scaling-Out)



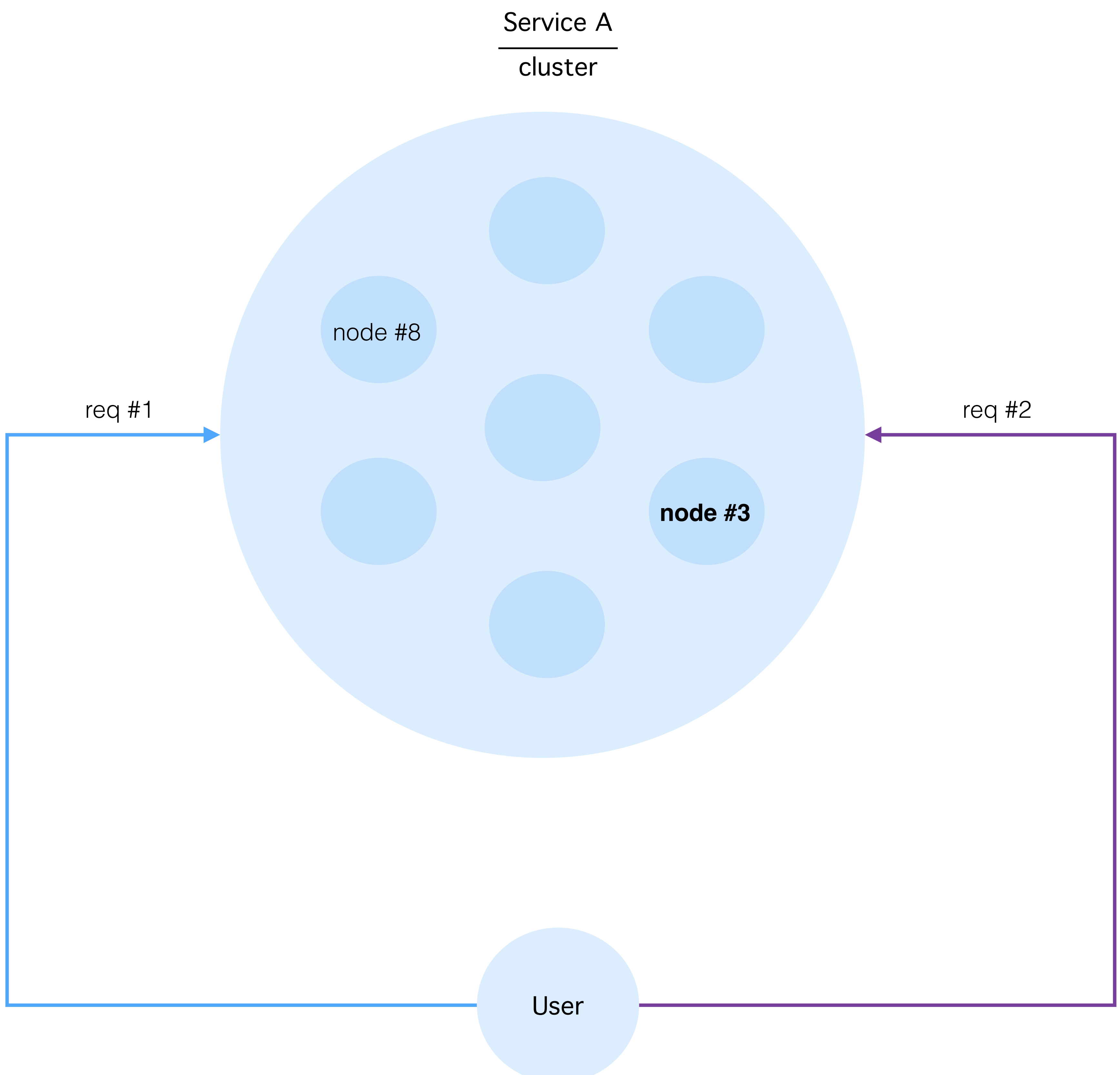
# Tech Req (Stateless => Scaling-Out)



# Tech Req (Stateless => Scaling-Out)



# Tech Req (Stateless => Scaling-Out)

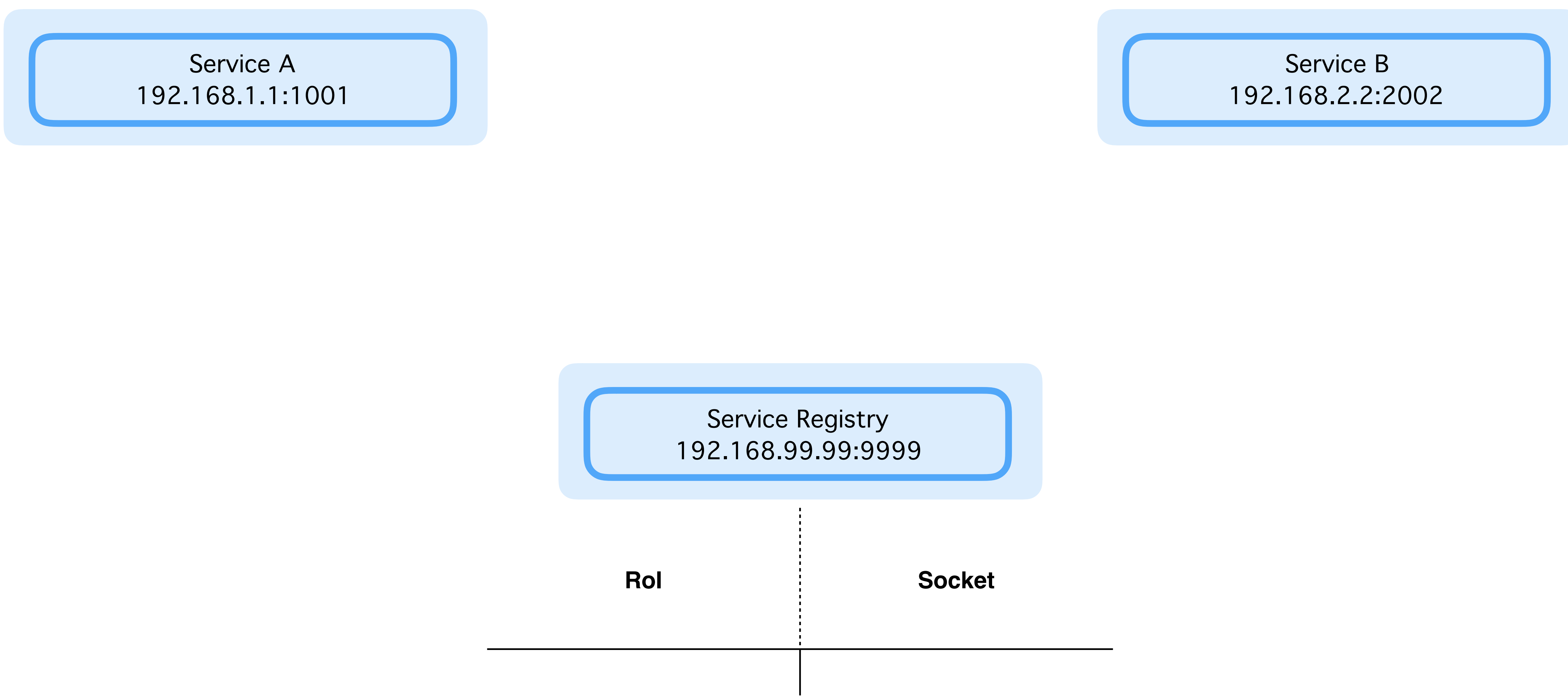


# Tech Req (Service Discovery and Registration)

Service A wants to communicate with Service B,
but what socket is using Service B?

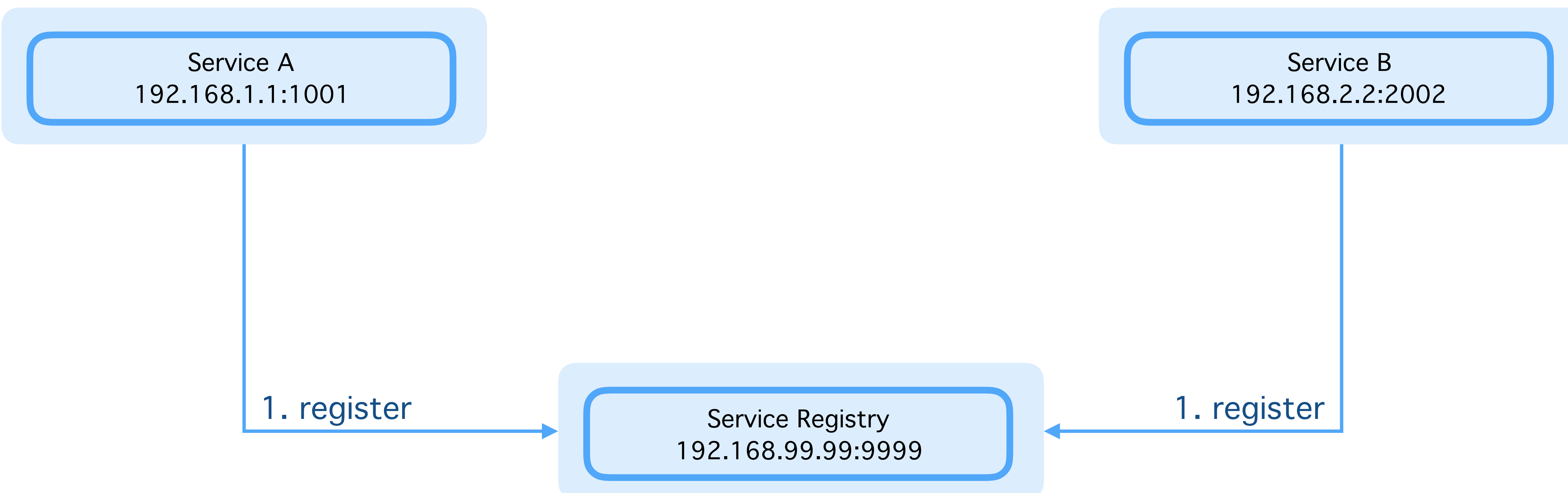
# Tech Req (Service Discovery and Registration)

Service A wants to communicate with Service B,
but what socket is using Service B?



# Tech Req (Service Discovery and Registration)

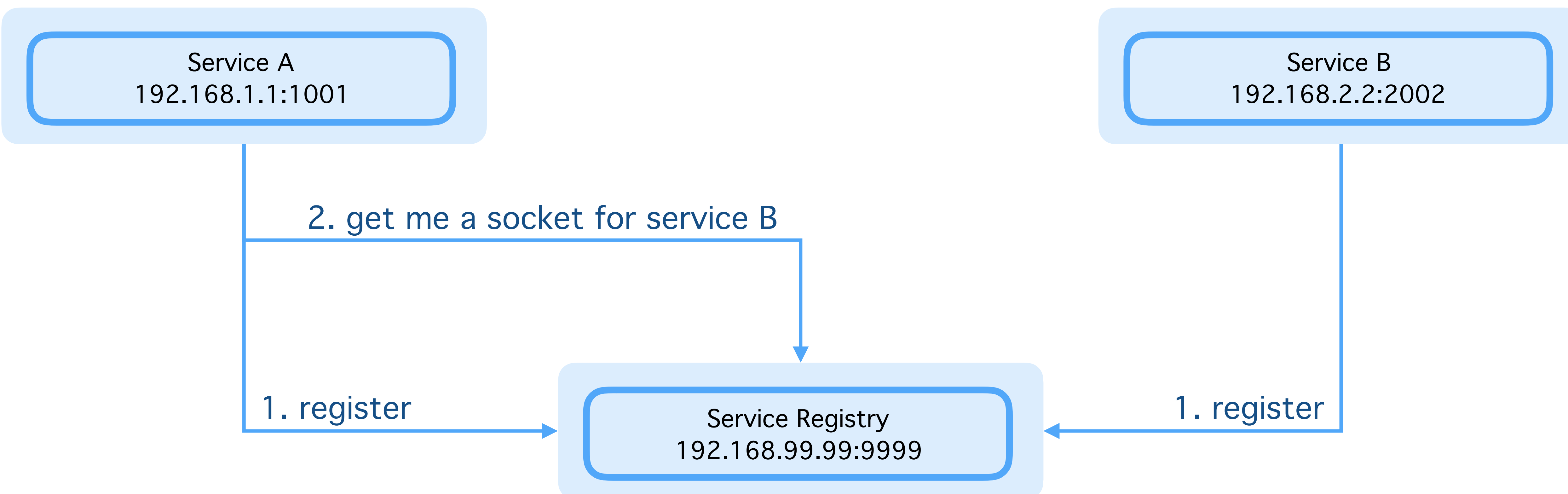
Service A wants to communicate with Service B,
but what socket is using Service B?




| **Rol** | **Socket** |
|---|---|
| service A | 192.168.1.1:1001 |
| service B | 192.168.2.2:2002 |

# Tech Req (Service Discovery and Registration)

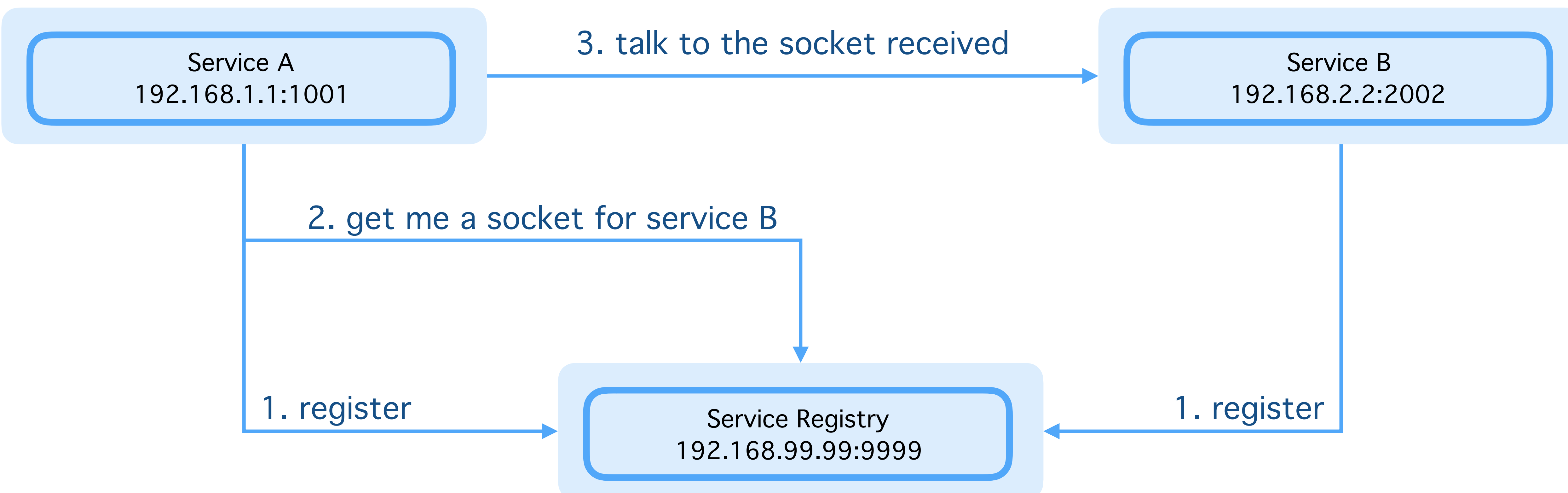
Service A wants to communicate with Service B,
but what socket is using Service B?




| Rol | Socket |
|---|---|
| service A | 192.168.1.1:1001 |
| service B | 192.168.2.2:2002 |

# Tech Req (Service Discovery and Registration)

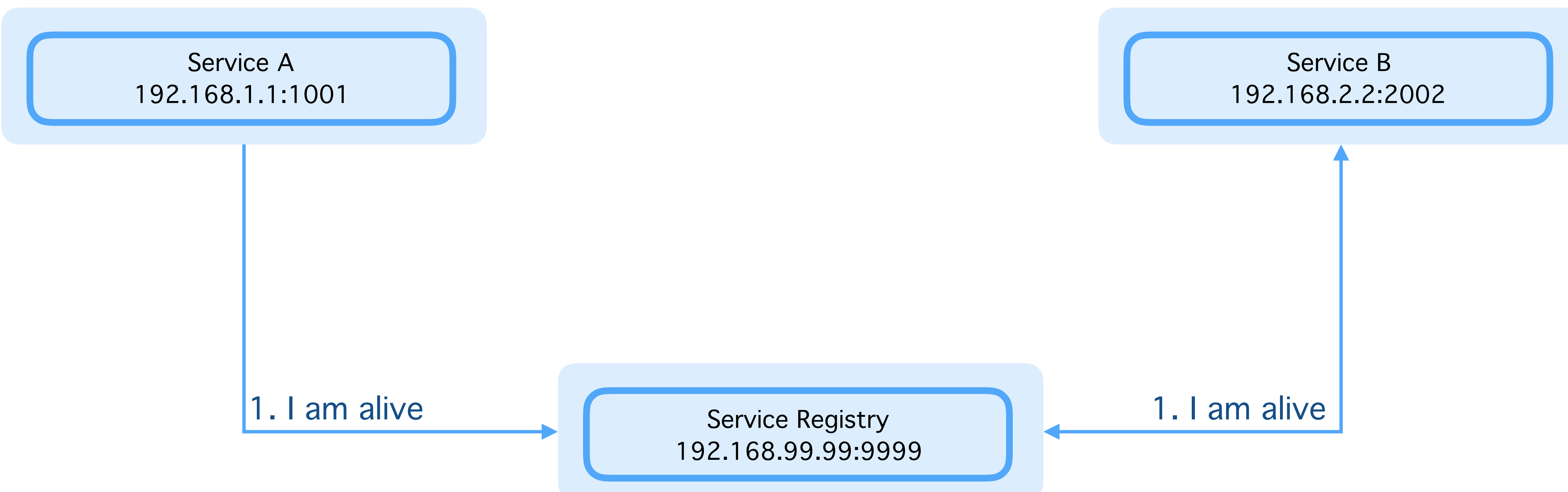
Service A wants to communicate with Service B,
but what socket is using Service B?




| **Rol** | **Socket** |
| --- | --- |
| service A | 192.168.1.1:1001 |
| service B | 192.168.2.2:2002 |

# Tech Req (Health Check)

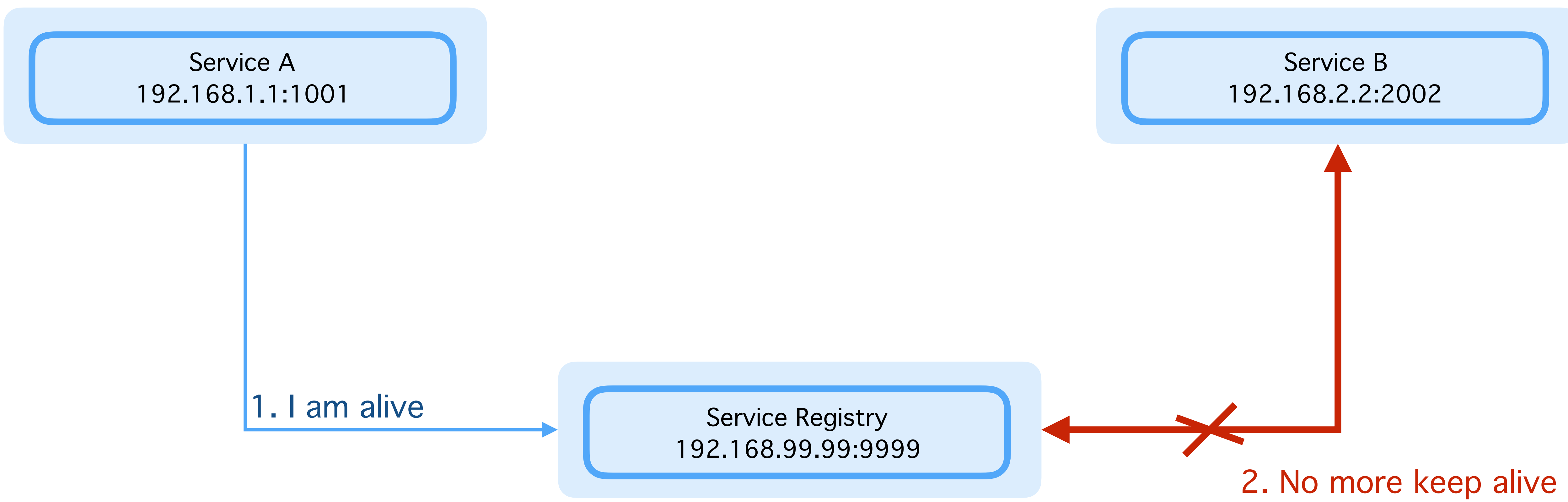
Service A wants to communicate with Service B,
but what socket is using Service B?




| Rol | Socket | TTL |
|---|---|---|
| service A | 192.168.1.1:1001 | 19 |
| service B | 192.168.2.2:2002 | 5 |

# Tech Req (Health Check)

Service A wants to communicate with Service B,
but what socket is using Service B?

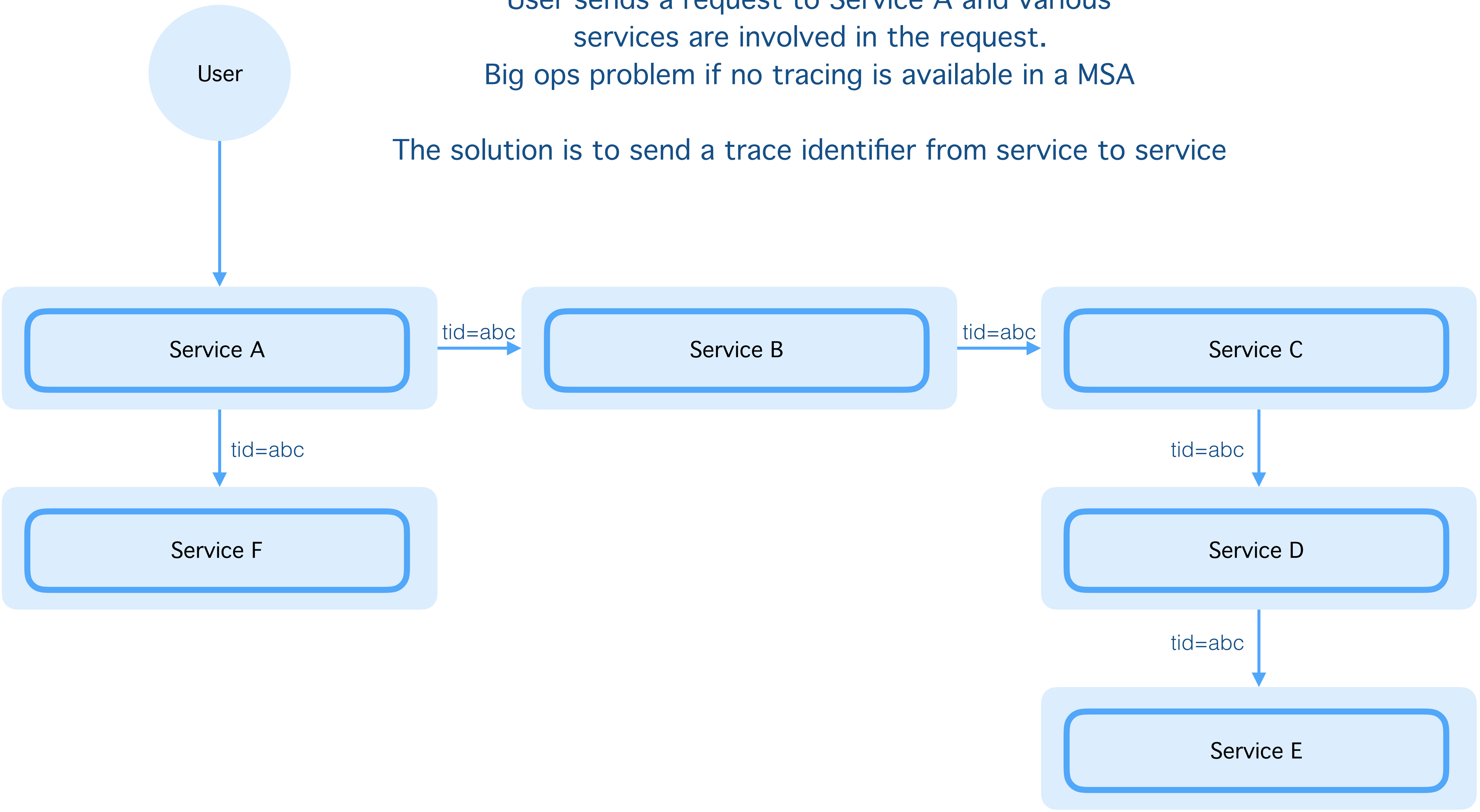



| Rol | Socket | TTL |
|---|---|---|
| service A | 192.168.1.1:1001 | 19 |
| [illegible] | [illegible] | [illegible] |

# Tech Req (Distributed tracing)

User sends a request to Service A and various services are involved in the request.
Big ops problem if no tracing is available in a MSA

The solution is to send a trace identifier from service to service



# Pros and Cons of the MSA

- Independent and loosely coupled services
- Easy to add new features
- Better resource allocation (memory, cpu, IO)
- Small codebase
- Cheap to adopt new technologies and mix them
- Fit into agile pipelines (code, test, deploy)
- Go hand by hand with platforms like Docker and Kubernetes
- Ensure better long-term system stability
- Network penalty
- Requires instrumentation around (service discovery, tracing ...)

# Building a testing sync MSA with ClawIO

- Swiss-Army-Knife for my ideas
- Presented last year at CS3 Zurich as a benchmarking platform for OC sync performance
- Configurable and modular server daemon (MSA and monolith modes)
- Prototyped Web Application and CLI
- Written in Go, one binary, no dependencies
- Very easy to use (clawiod -conf my.conf)
- OpenSource (github.com/clawio/clawiod)

# ClawIO Web Services APIS

data
- POST /data/upload
- POST /data/download

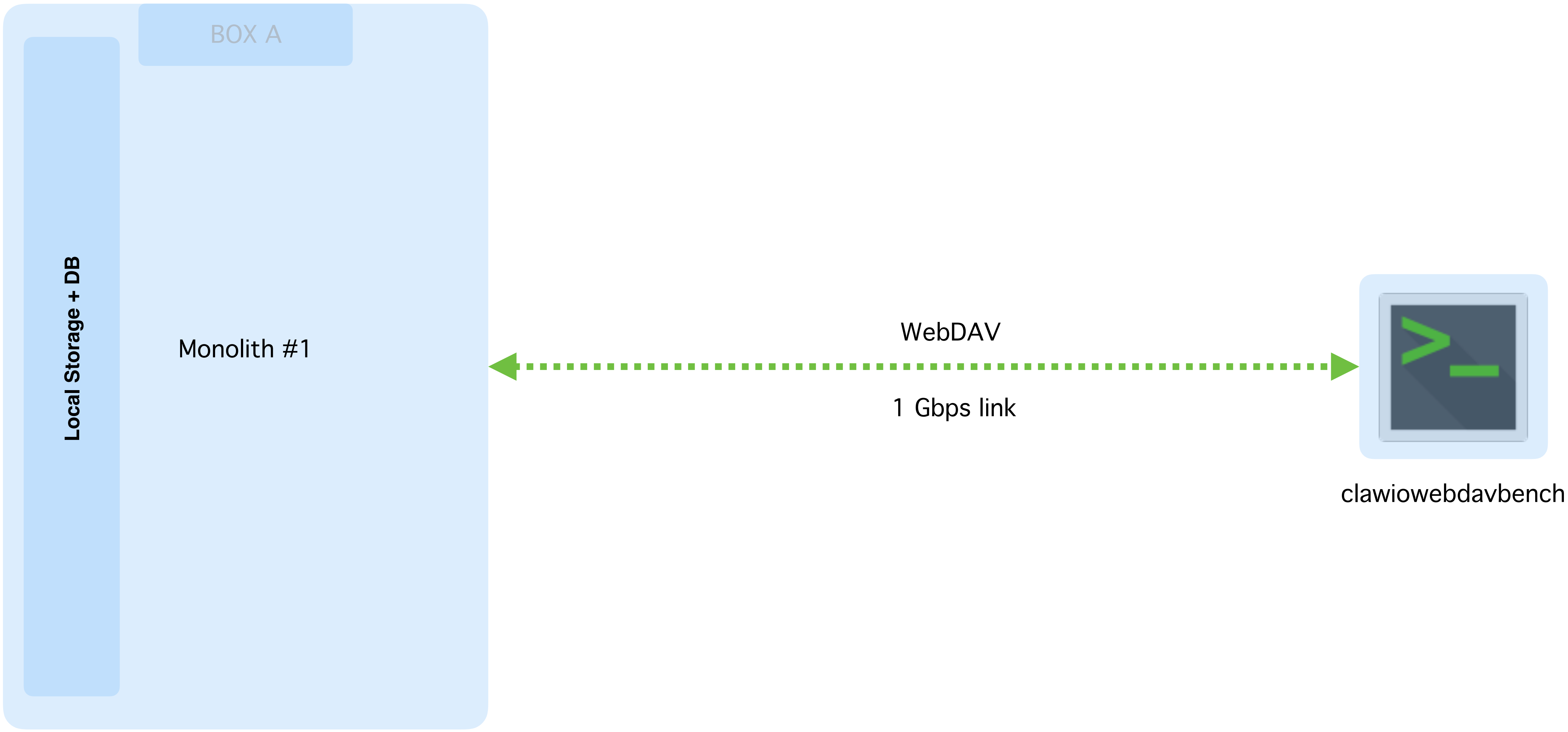
metadata
- POST /meta/examine
- POST /meta/list
- POST /meta/createfolder
- POST /meta/move
- POST /meta/rm

auth
- POST /auth/token
- POST /auth/ping

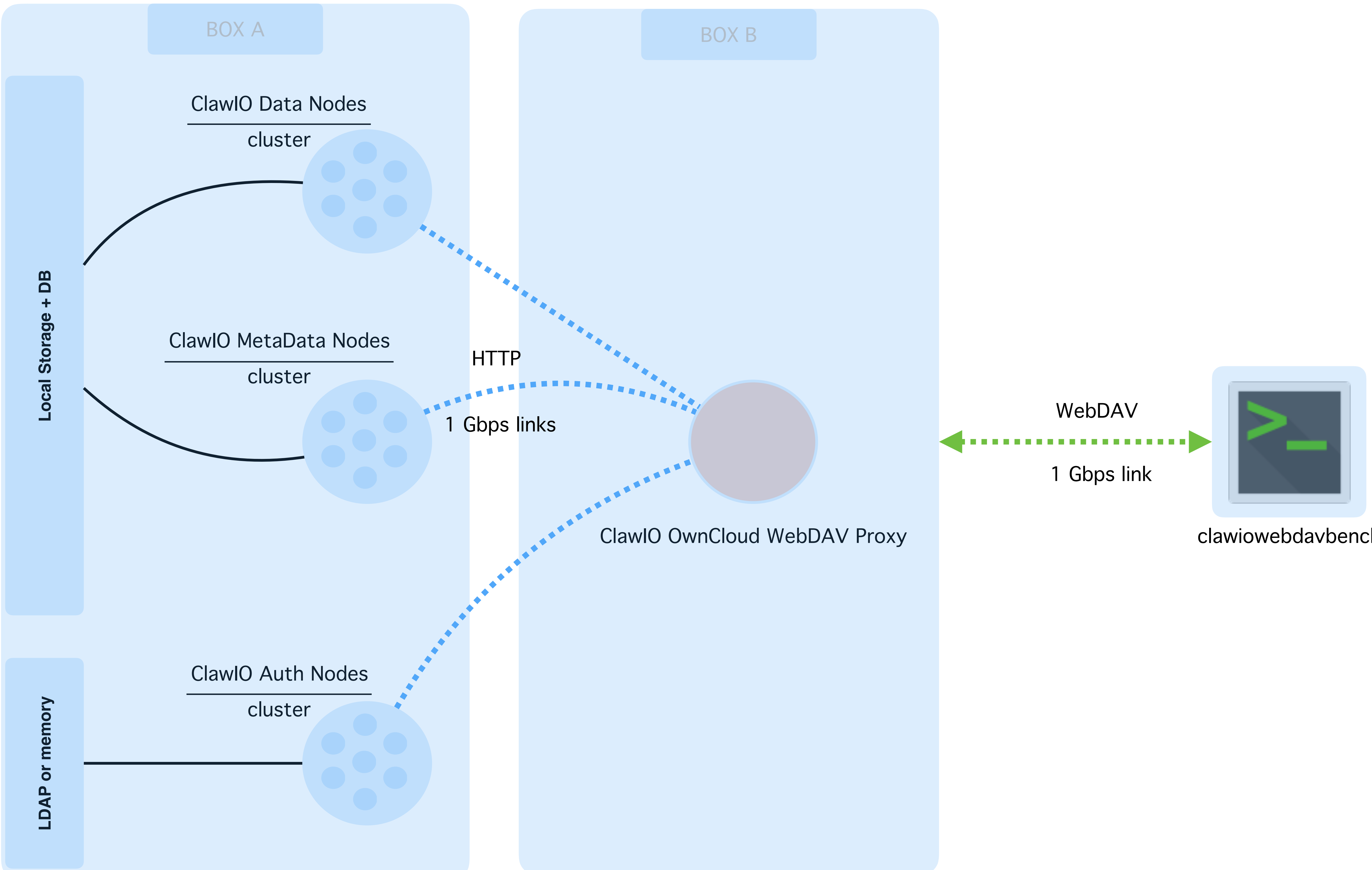
oc webdav
- GET /ocwebdav/
- PUT /ocwebdav/
- PROPFIND /ocwebdav/
- ...

- RPC oriented
- Dropbox API v2 style
- Very simple
- Very lightweight

# TestBed for ClawIO in Monolith Mode



# Testbed for ClawIO in Micro-Services-Architecture Mode



# WebDAV PROPFIND Test

5000 requests per test with different number of concurrent clients
The payload is a few KB, to observe better network influence

| CONCURRENCY | MONOLITH | MSA x 1 | MSA x 2 | MSA x 3 |
| --- | --- | --- | --- | --- |
| 1 | 114 Hz | | | |
| 100 | 685 Hz | | | |
| 200 | 793 Hz | | | |
| 400 | 916 Hz | | | |

# WebDAV PROPFIND Test

5000 requests per test with different number of concurrent clients
The payload is a few KB, to observe better network influence

| CONCURRENCY | MONOLITH | MSA x 1 | MSA x 2 | MSA x 3 |
|---|---|---|---|---|
| 1 | 114 Hz | 98 Hz | | |
| 100 | 685 Hz | 524 Hz | | |
| 200 | 793 Hz | 637 Hz | | |
| 400 | 916 Hz | 772 Hz | | |

# WebDAV PROPFIND Test

5000 requests per test with different number of concurrent clients
The payload is a few KB, to observe better network influence

| CONCURRENCY | MONOLITH | MSA x 1 | MSA x 2 | MSA x 3 |
|---|---|---|---|---|
| 1 | 114 Hz | 98 Hz | 98 Hz | |
| 100 | 685 Hz | 524 Hz | 714 Hz | |
| 200 | 793 Hz | 637 Hz | 799 Hz | |
| 400 | 916 Hz | 772 Hz | 1 KHz | |

# WebDAV PROPFIND Test

5000 requests per test with different number of concurrent clients
The payload is a few KB, to observe better network influence

| CONCURRENCY | MONOLITH | MSA x 1 | MSA x 2 | MSA x 3 |
|---|---|---|---|---|
| 1 | 114 Hz | 98 Hz | 98 Hz | 78 Hz |
| 100 | 685 Hz | 524 Hz | 714 Hz | 893 Hz |
| 200 | 793 Hz | 637 Hz | 799 Hz | 956 Hz |
| 400 | 916 Hz | 772 Hz | 1 KHz | 1,05 KHz |

# Some Preliminary Conclusions

- MSA for sync and share could be used with a distributed storage to benefit from parallel access from data and metadata nodes

- A MSA could allow to efficiently use your data center resources fitting services to hardware

- MSA should play well with a containerized infrastructure

```
gonzalhu@data-center-01:/root  $ docker scale oc-data-node=30
```

Thank you