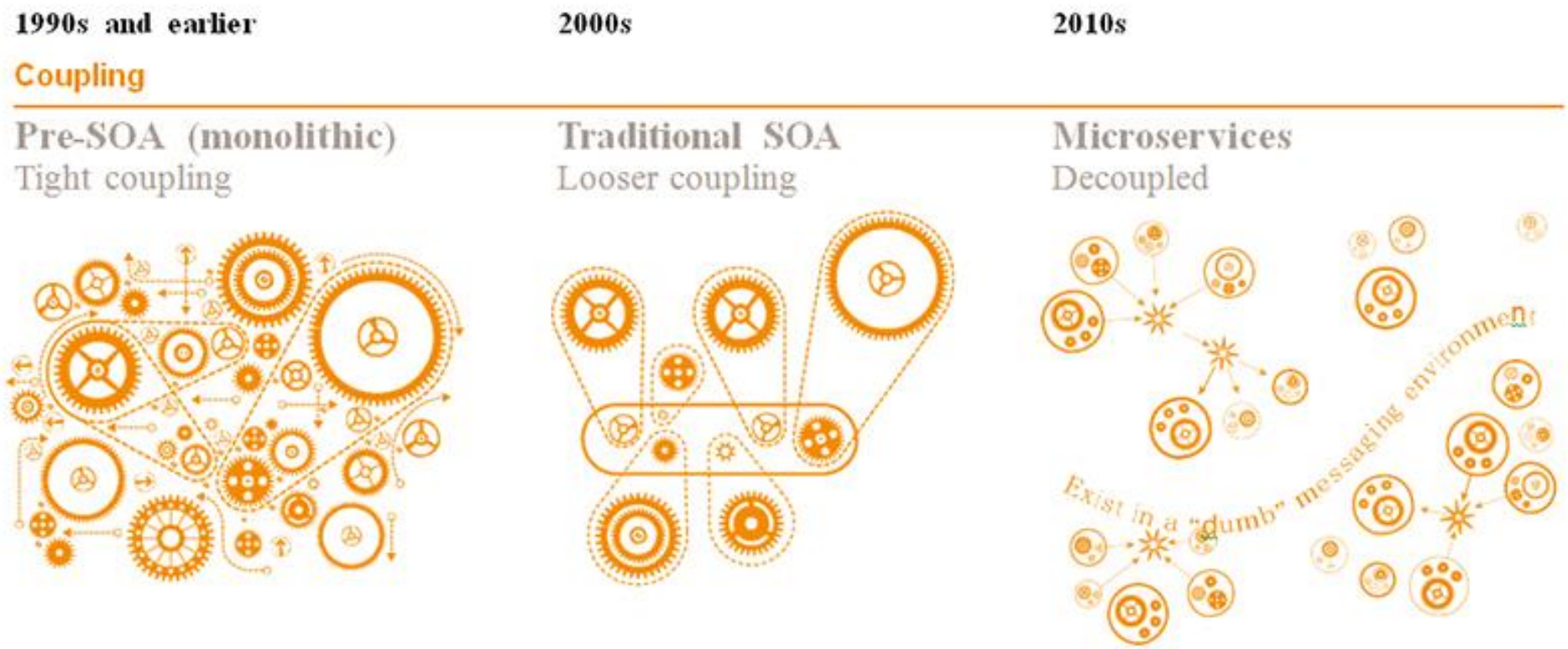
MicroServices

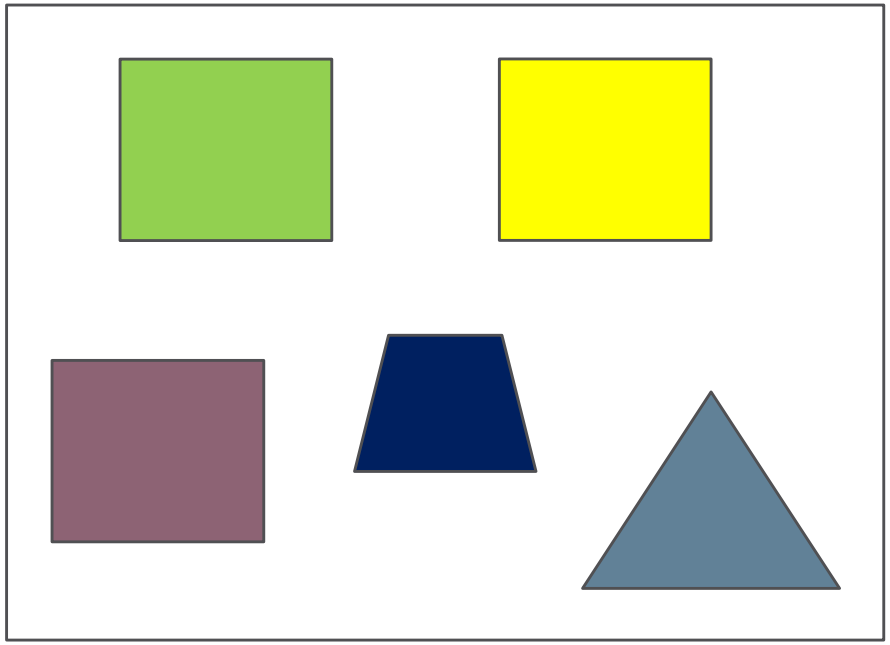
MicroServices evolution



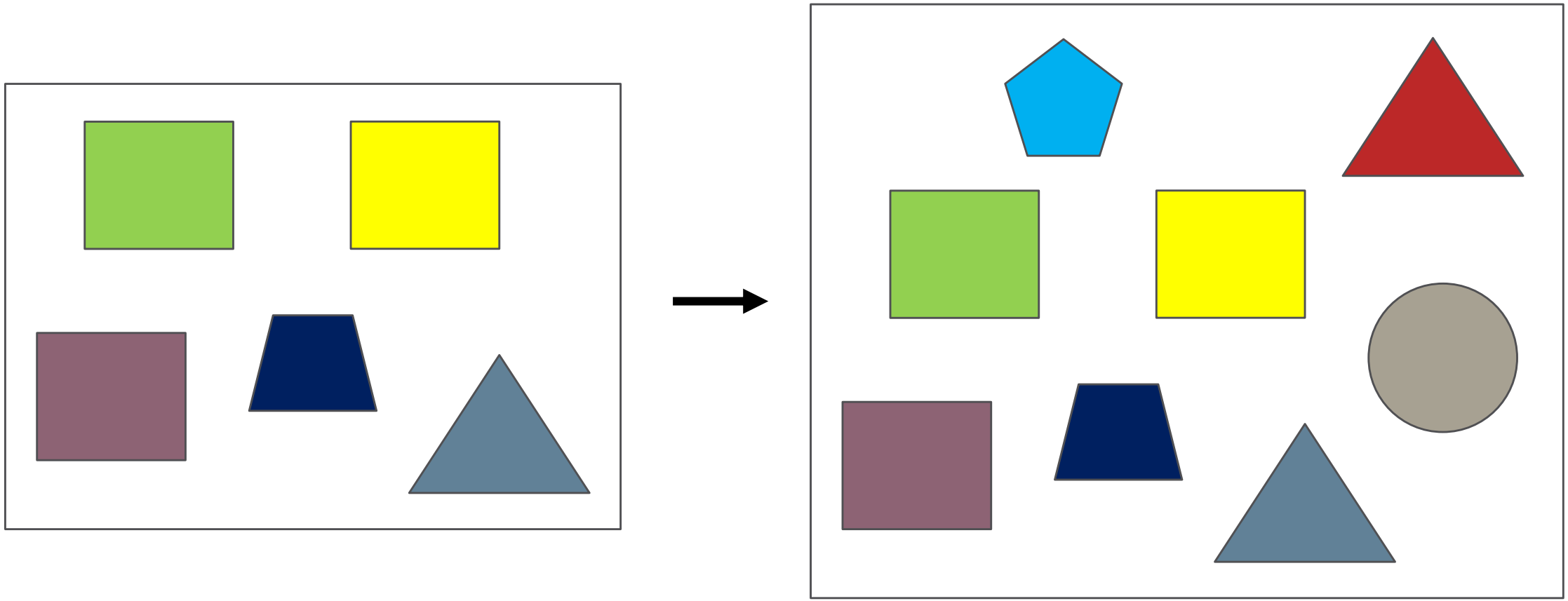
img: *https://open-organization.com/en/2018/01/04/francais-la-vision-technologique-de-philippe-mourant-le-nouveau-cto-de-presans/microservices/*

Monolith

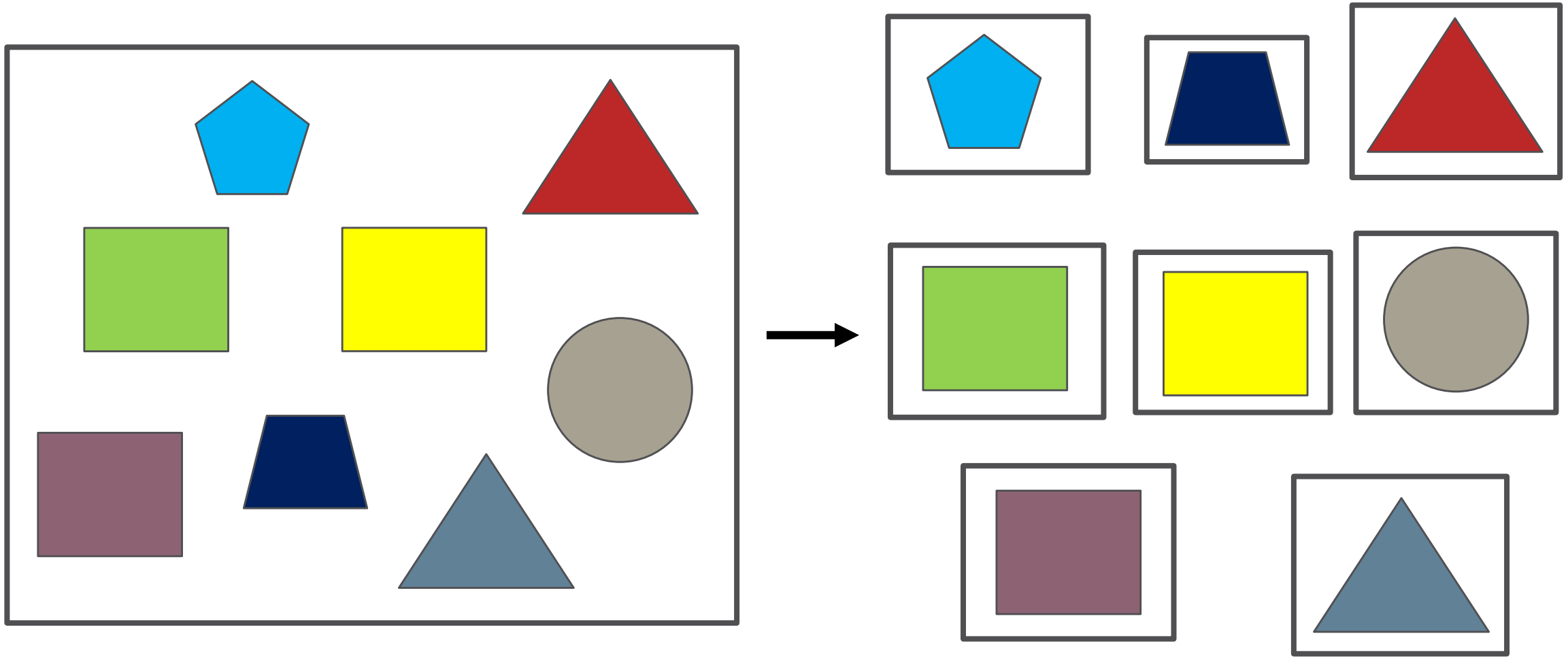
- Добавянето на нови функционалности, с течение на времето, увеличава значимо размера и сложността му.
- Усилието за добавяне на нови функции към монолита също нараства значително.



Monolith



Monolith vs MicroServices

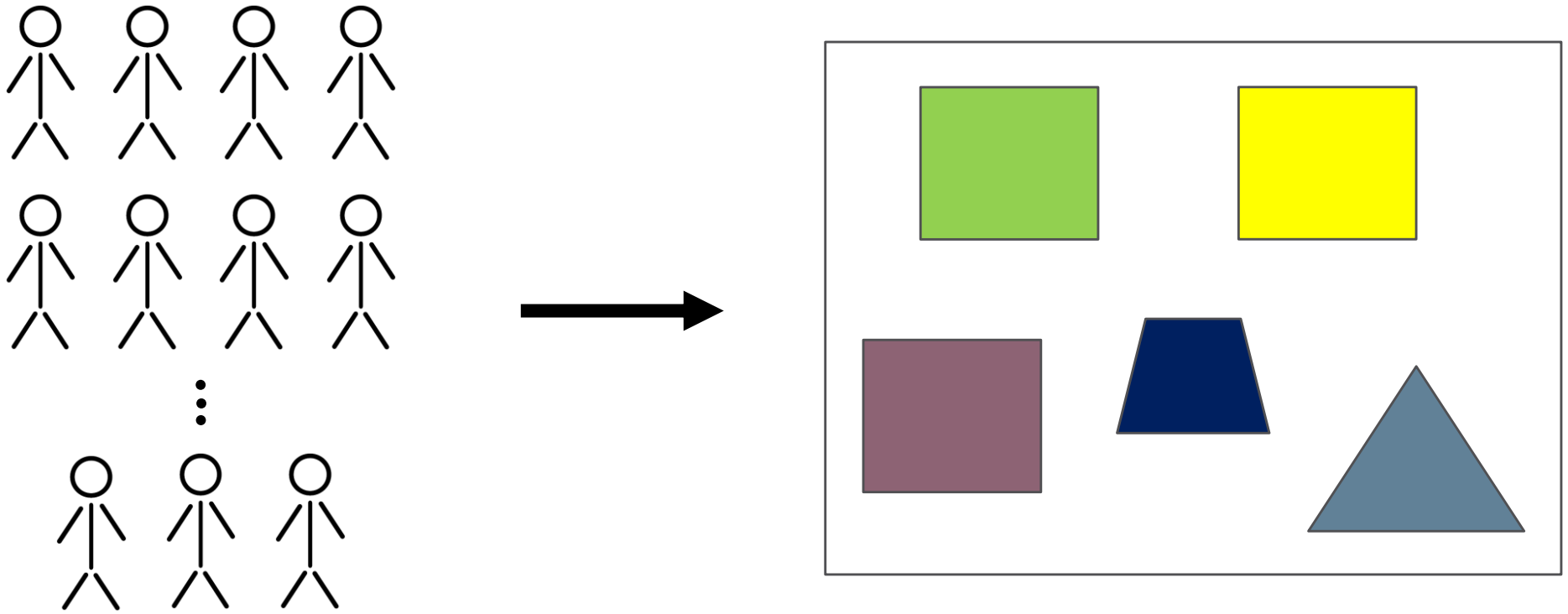


MicroServices

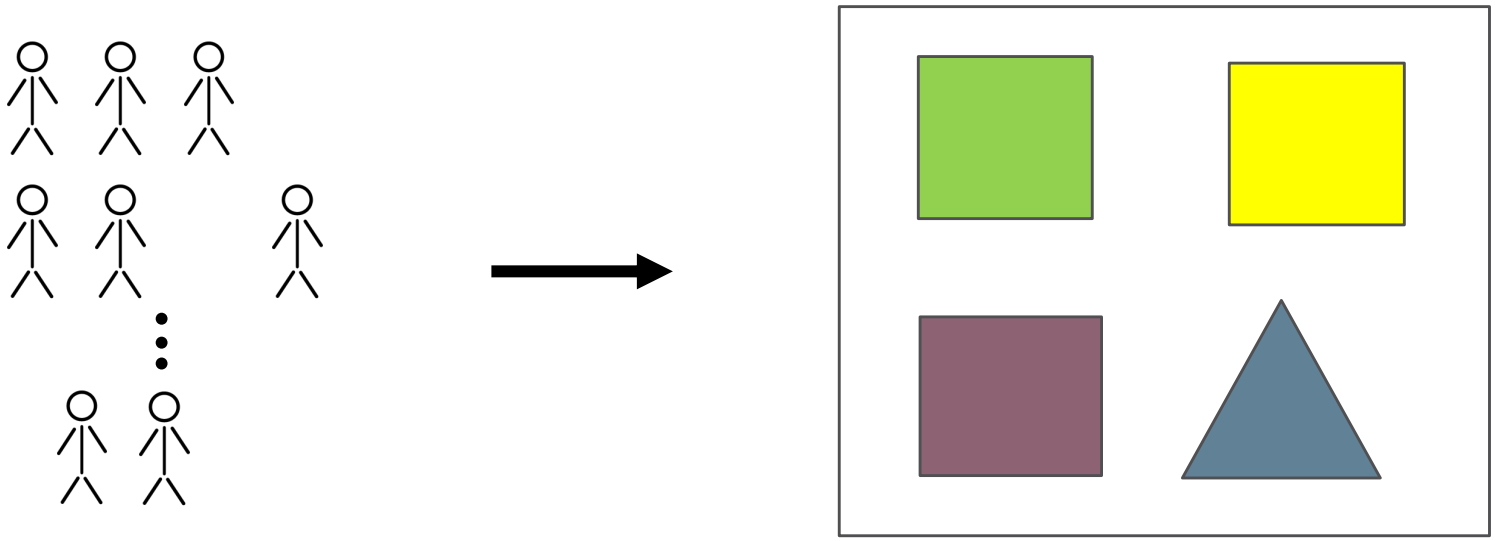
Типични свойства на архитектурата на микроуслуги са:

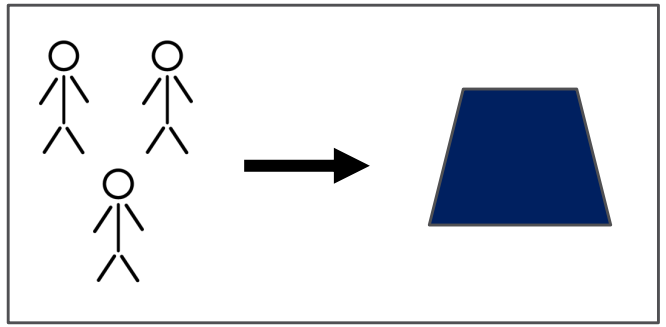
- те са (в контекста им) *малки*.
- могат да се „*деплойнат*“ независимо
- не са зависими от дадена технология
- комуникират стриктно чрез своите API-та

Team Organization



Team Organization

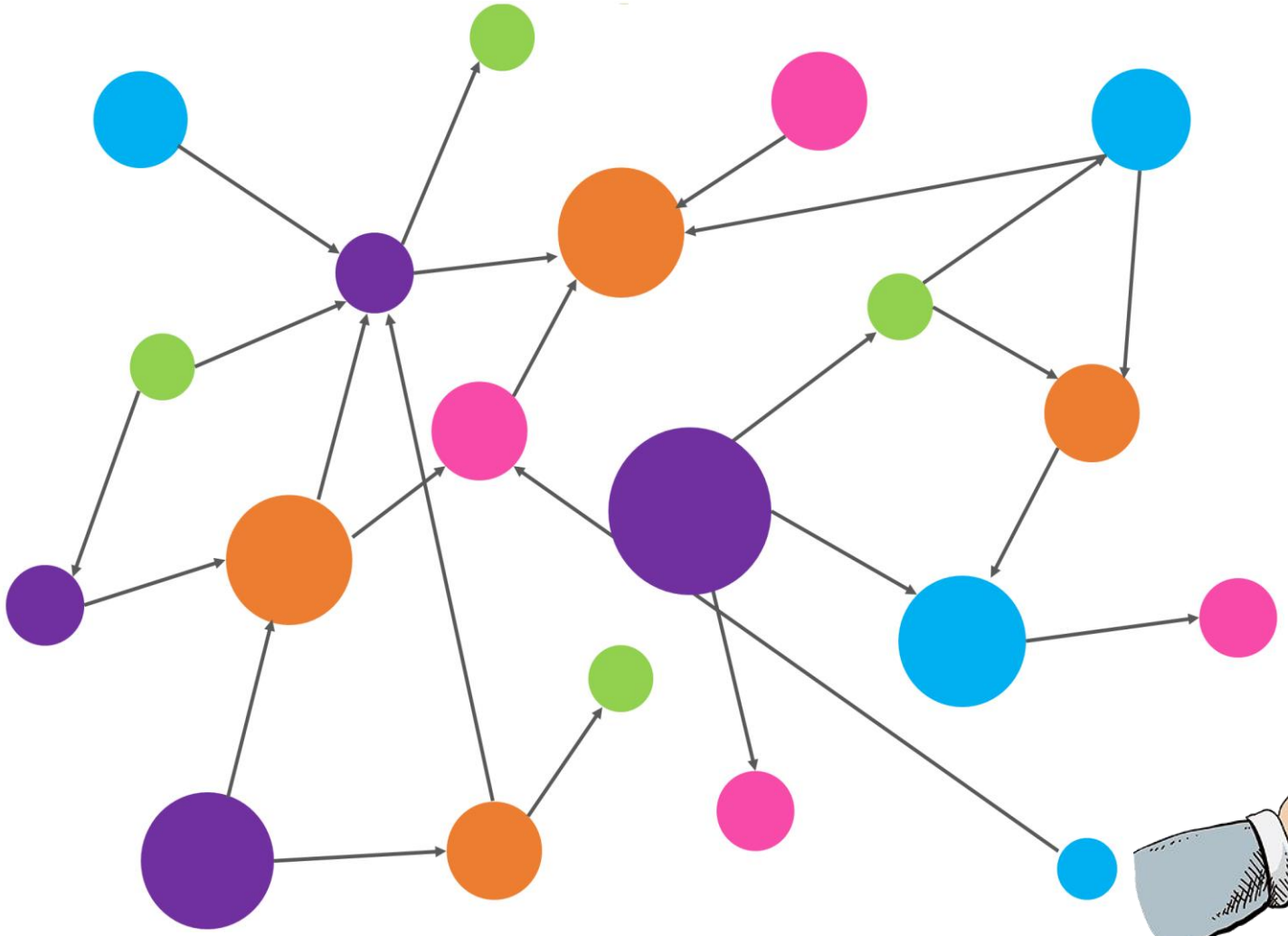


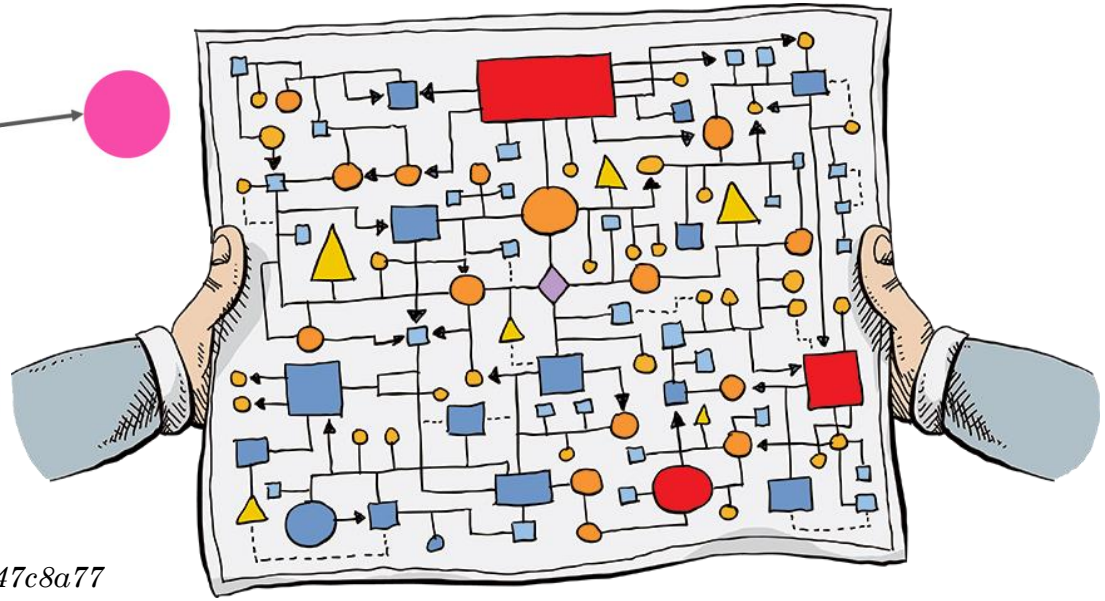


Team Organization

- ✓ Екипа се развива по-лесно – по-тесен обхват (*narrow scope*)
- ✓ Намалява риска за bottleneck
- ✓ По-лесно за нови разработчици да се присъединят
- ✓ Всеки екип може да взема изолирани архитектурни решения, които не влияят на цялата система (*isolated impact*)
- ✓ Екипите могат да управляват работата си по различен начин
- ✓ Децентрализирано вземане на решения

MicroServices - Difficulties



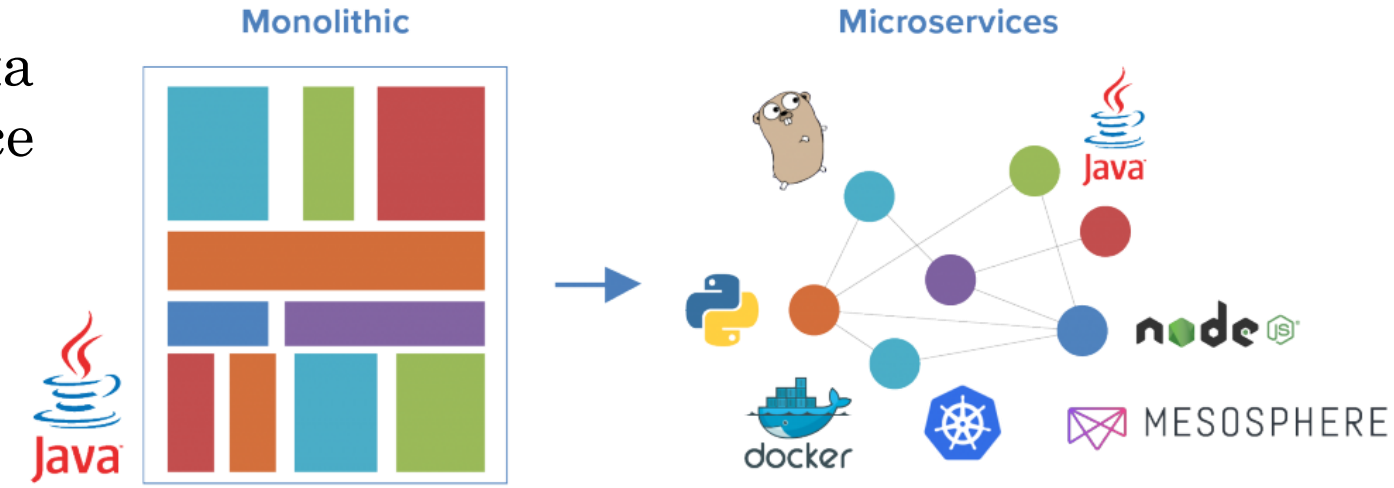


https://medium.com/@marcus.cavalcanti/lessons-learned-about-run-microservices-b360347c8a77

https://medium.com/@vladikk.com/tackling-complexity-in-microservices-6253c2aad007

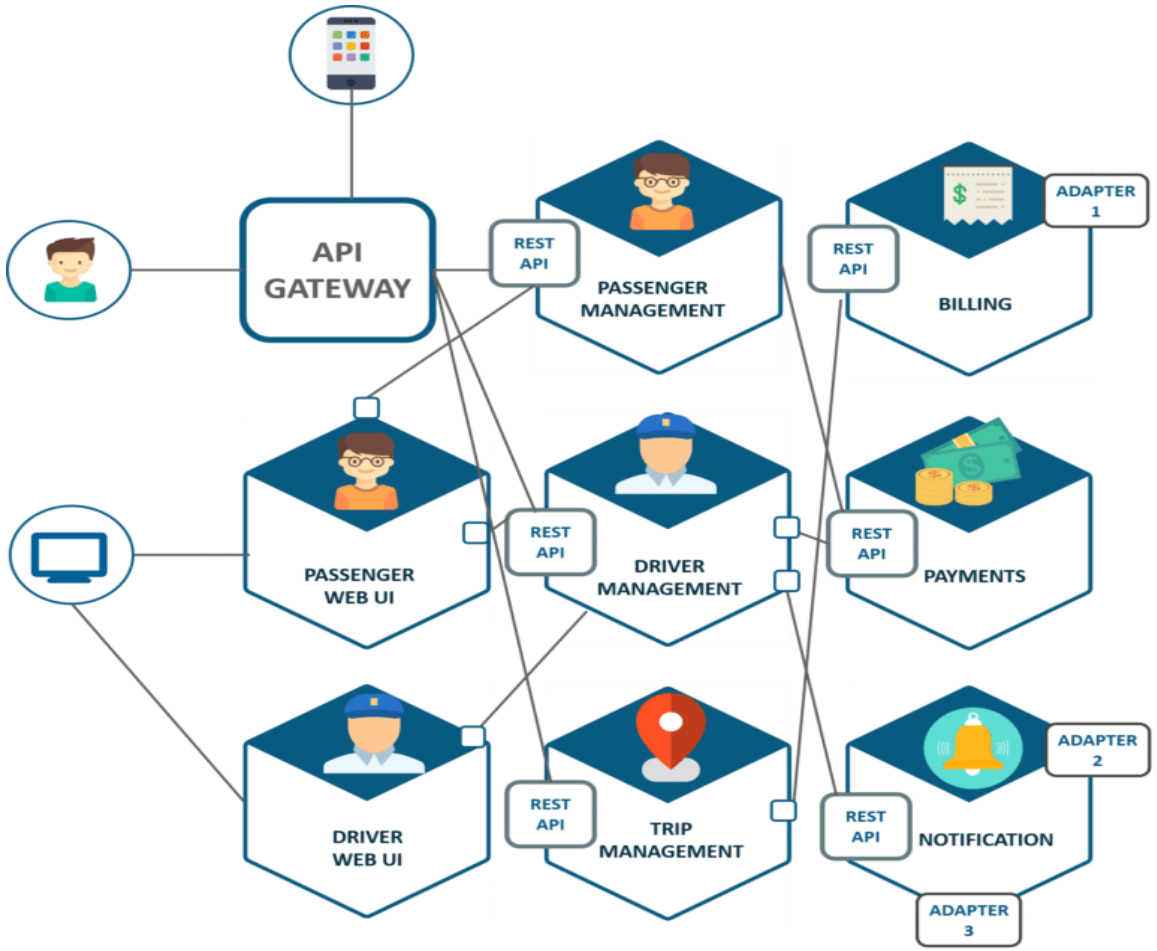
MicroServices - Difficulties

- Всяка комуникация е мрежова комуникация
- Мрежата се третира като ненадежна (*design for failure*)
- Разпределението на логика в независимите системи е *трудно*
- По-труден анализ на логове в сравнение с монолита
- Независимо съхранение на данни (*Independent data storage*)
- Границите (*boundaries*) на микроуслугите трябва да се избират внимателно



https://blog.netsil.com/microservices-devops-and-operational-complexity-be98cb01b660

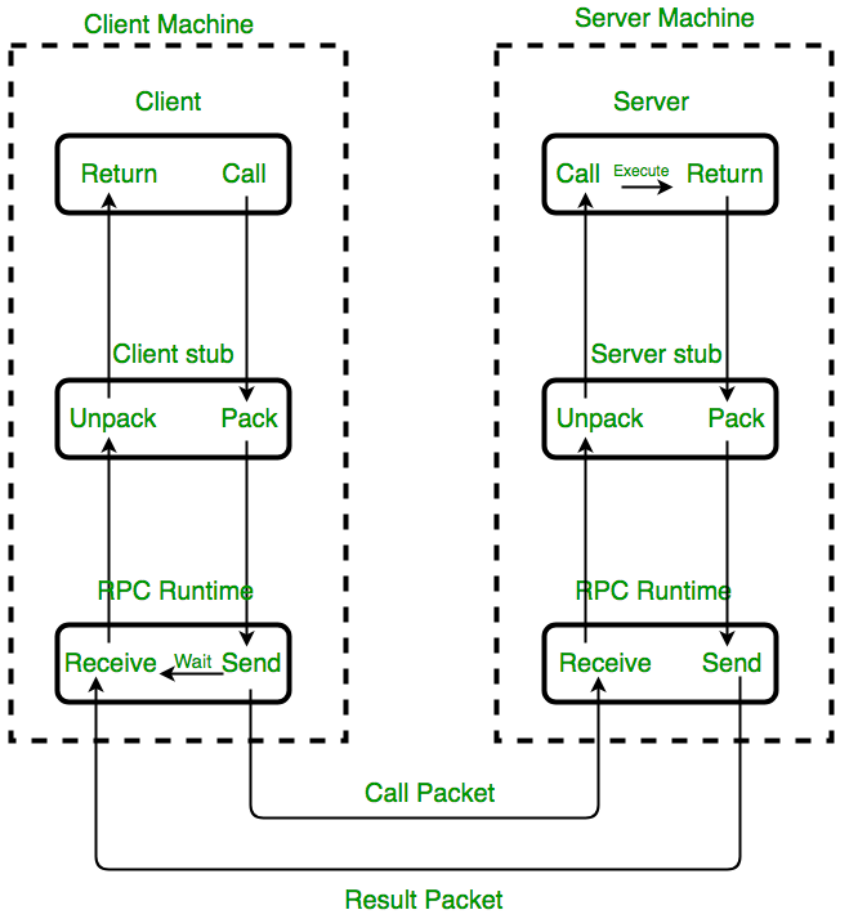
MicroServices - Communication



src: *https://dzone.com/articles/microservice-architecture-learn-build-and-deploy-a*

Synchronous Communication

Remote Procedure Call – *RPC*

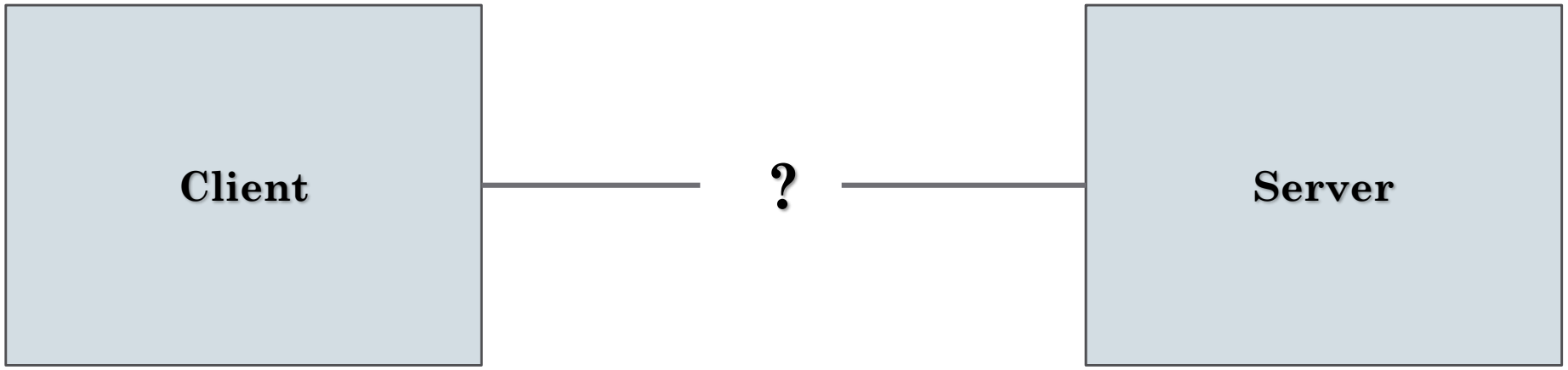


Implementation of RPC mechanism

https://www.geeksforgeeks.org/remote-procedure-call-rpc-in-operating-system/

Synchronous Communication

Service Discovery in RPC



Synchronous Communication

Service Discovery in RPC

- Има различни начини зависи от нуждите ни.
- Един от най-широко използвани и прости е DNS.
- Много различни други опции като *Apache ZooKeeper* (+ scalability, - complexity), *Hashicorp Consul* (*multi-cloud* deployment)

Synchronous Communication

State in RPC

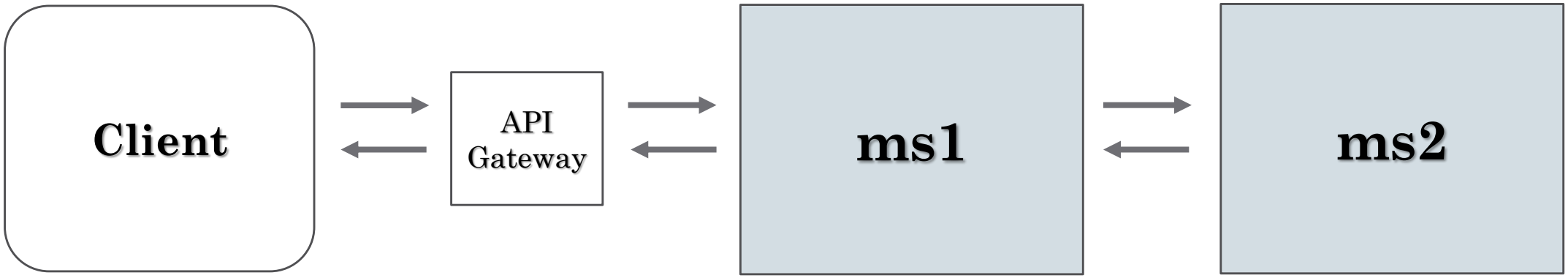
RPC могат да се разделят на две категории относно тяхното състояние (*state*):

- *Stateless* : в този случай просто се извършват някои изчисления и се връща резултат, без значение от състоянието на микроуслугата.

- *Stateful* : в този случай изчисленията се извършват с контекста на предишни транзакции и текущата транзакция може да бъде повлияна от случилото се по време на предишни транзакции.

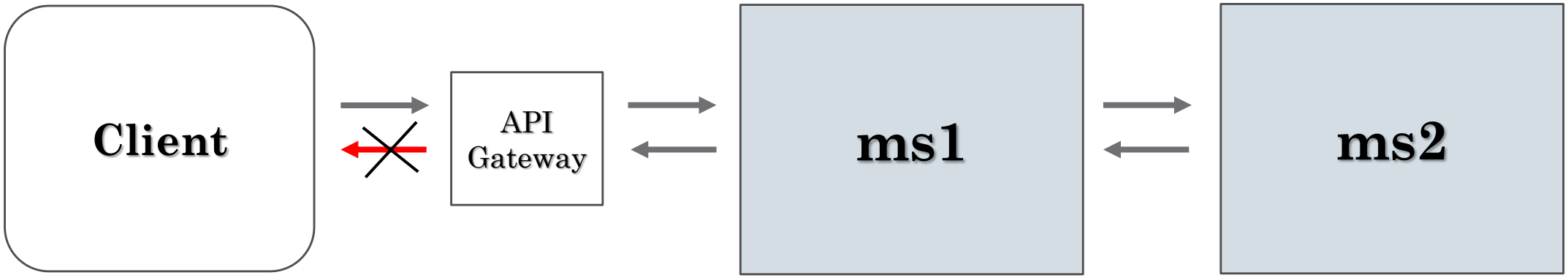
Stateful RPC

Idempotence **:** свойство на система, която не се променя, ако извършите една и съща операция многократно.



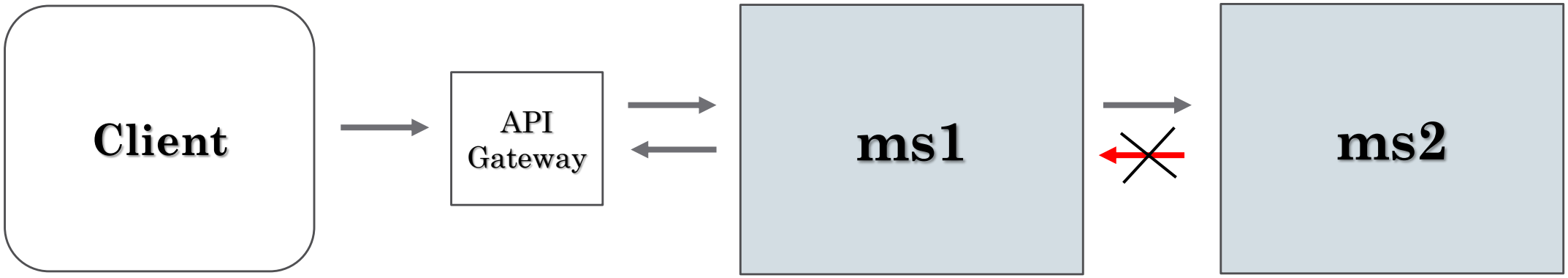
Stateful RPC

Idempotence **:** свойство на система, която не се променя, ако извършите една и съща операция многократно.



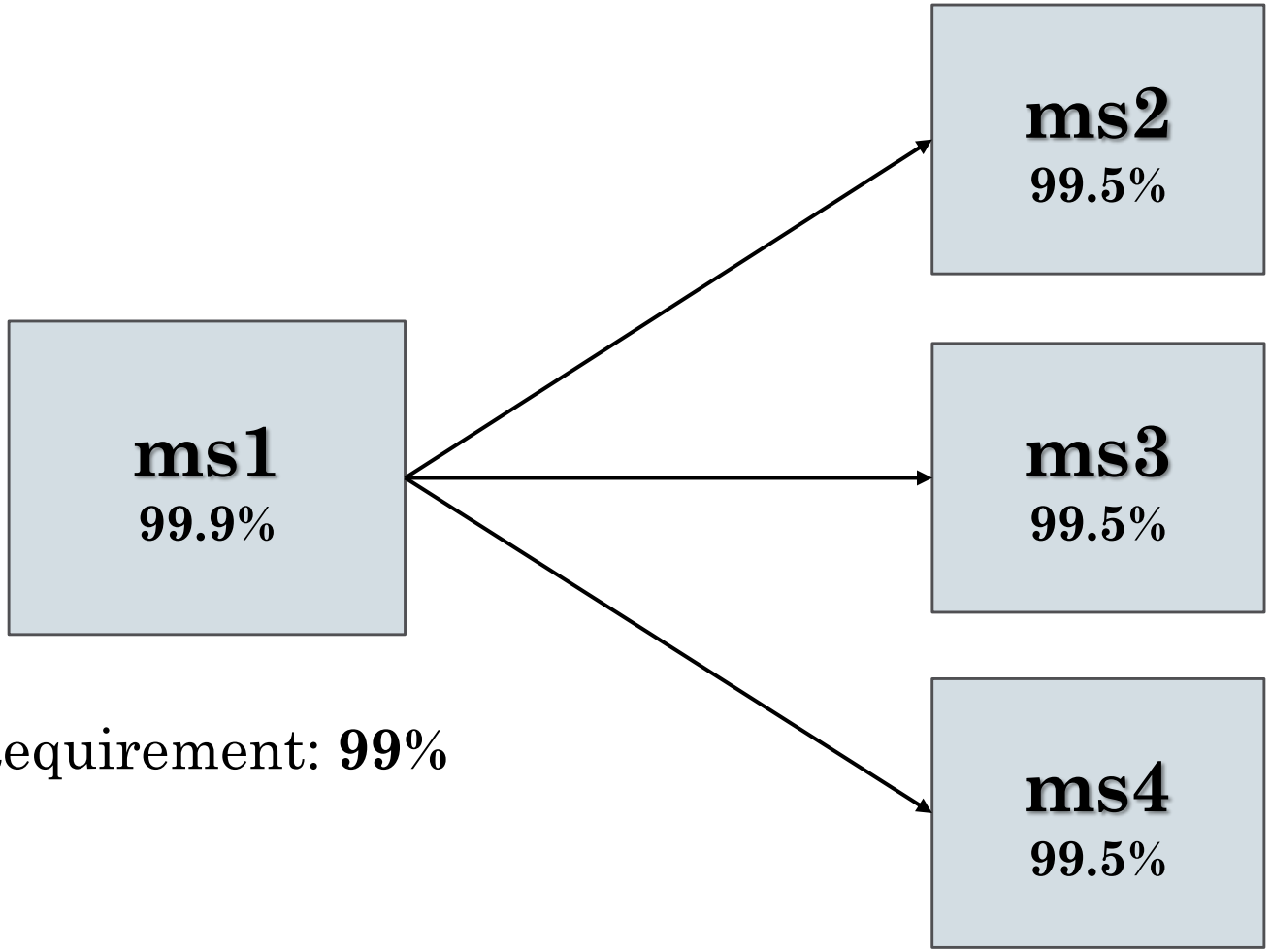
Stateful RPC

Idempotence **:** свойство на система, която не се променя, ако извършите една и съща операция многократно.



Synchronous Communication

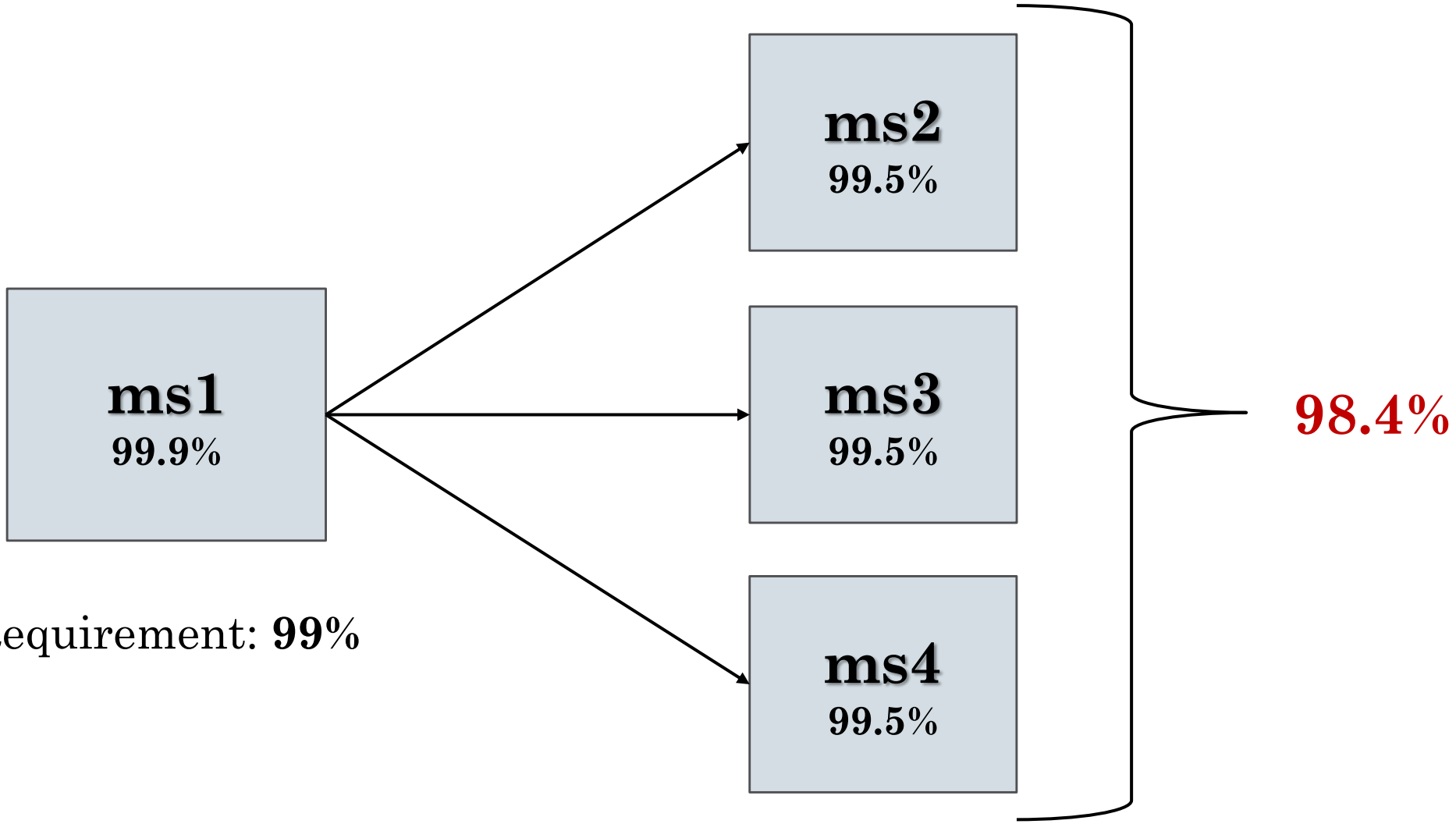
Availability



Requirement: **99%**

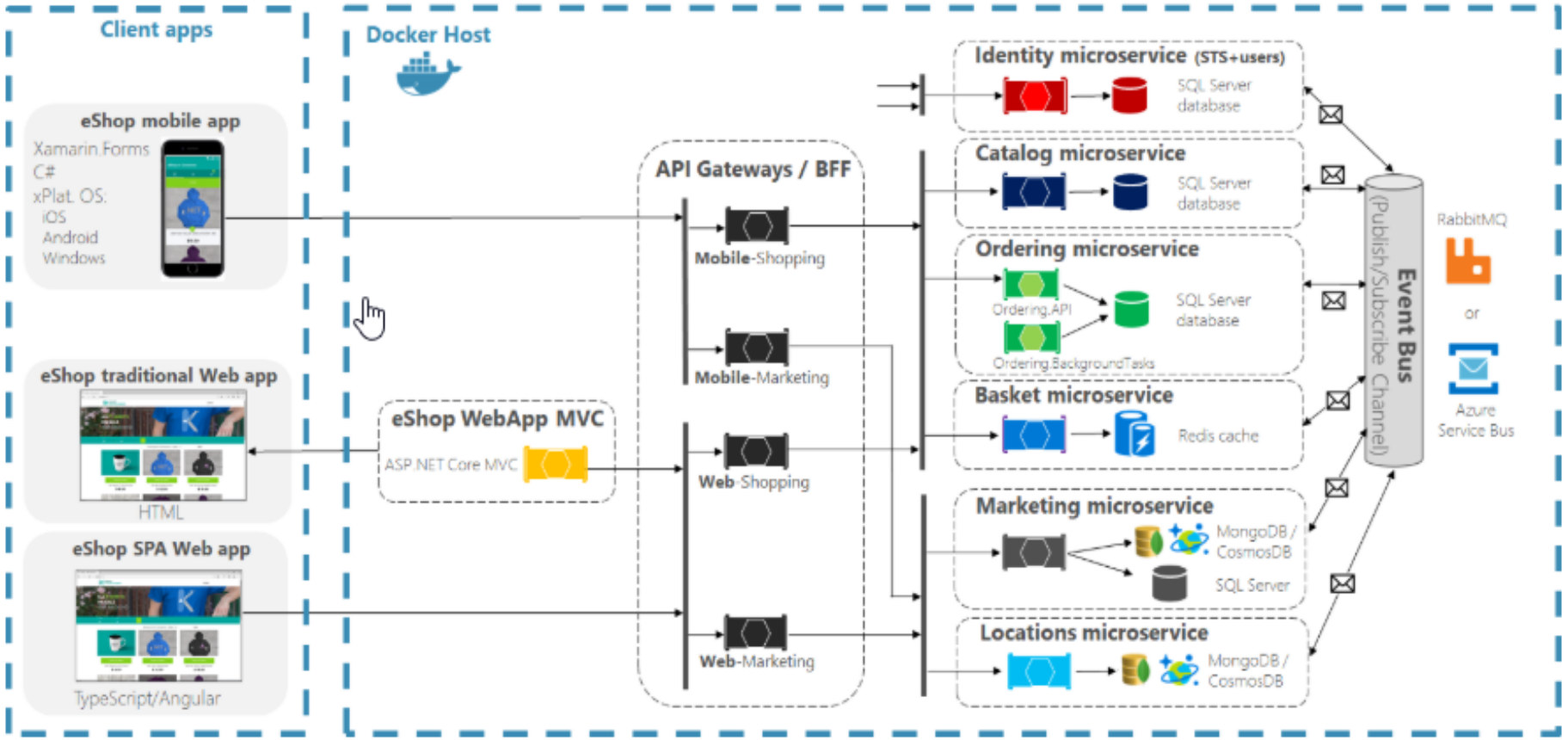
Synchronous Communication

Availability



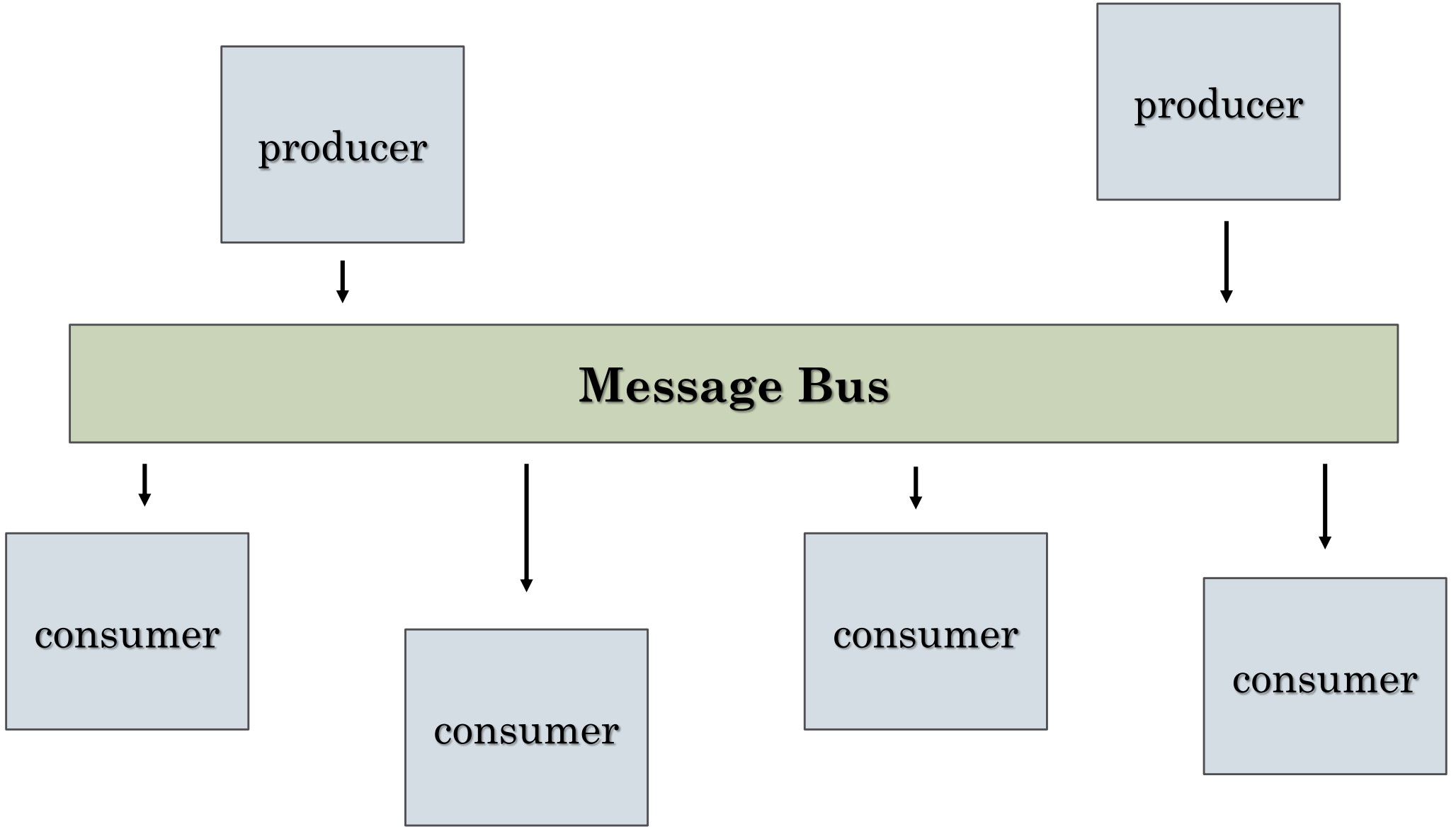
Requirement: **99%**

Asynchronous Communication



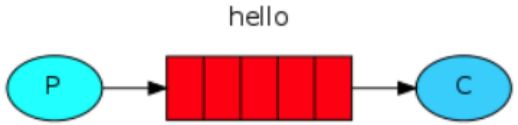
src: *https://www.asptricks.net/2019/05/distributed-microservice-architecture.html*

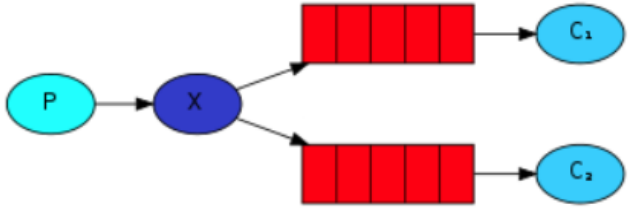
Asynchronous Communication

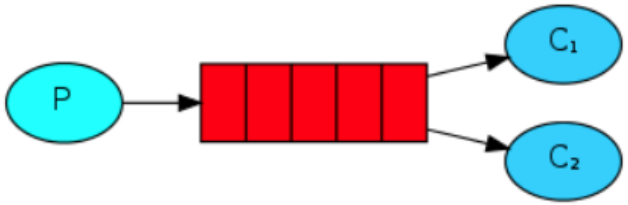


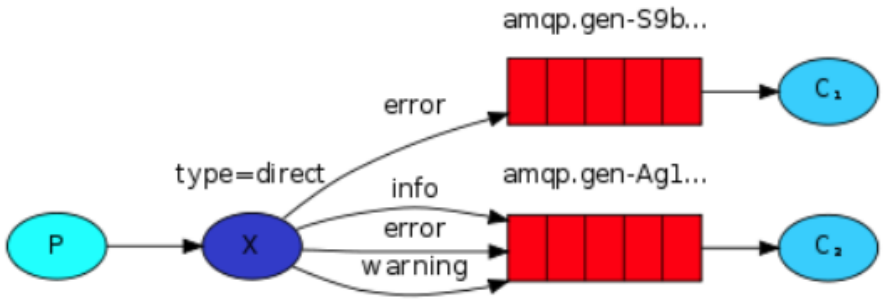
Asynchronous Communication

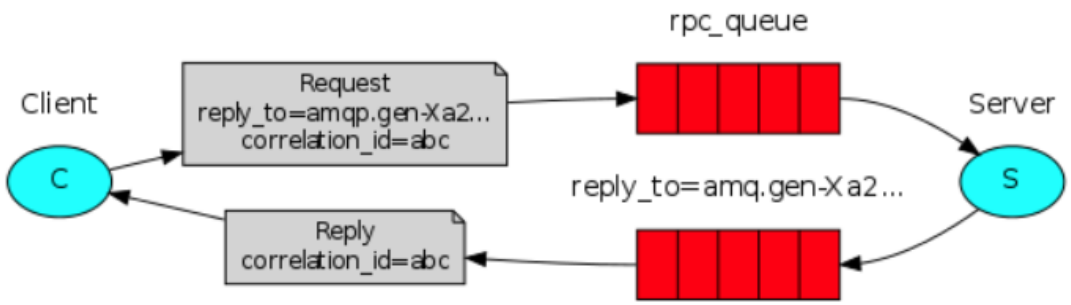
Queues in a Message Bus







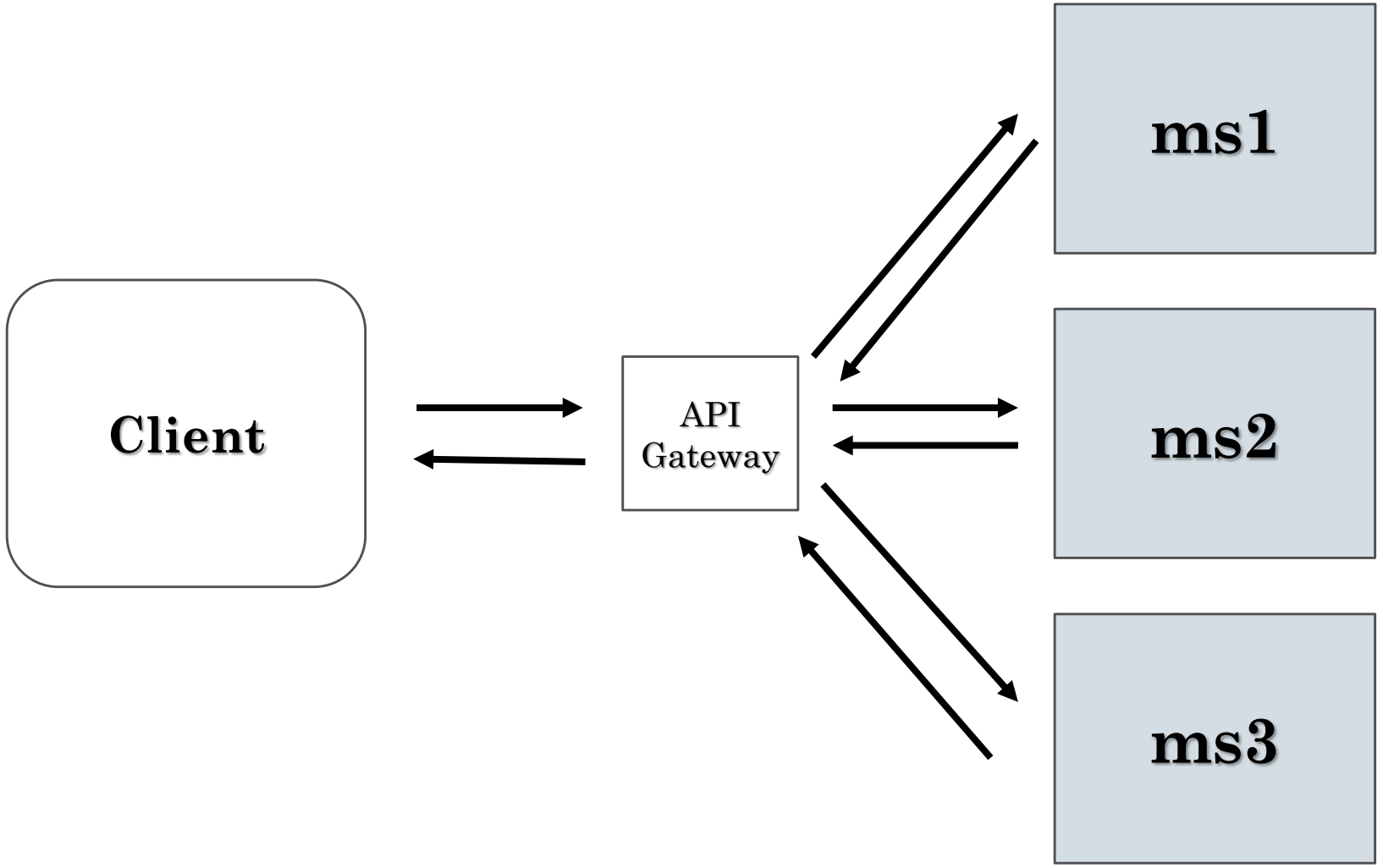




src: *https://www.rabbitmq.com/tutorials/*

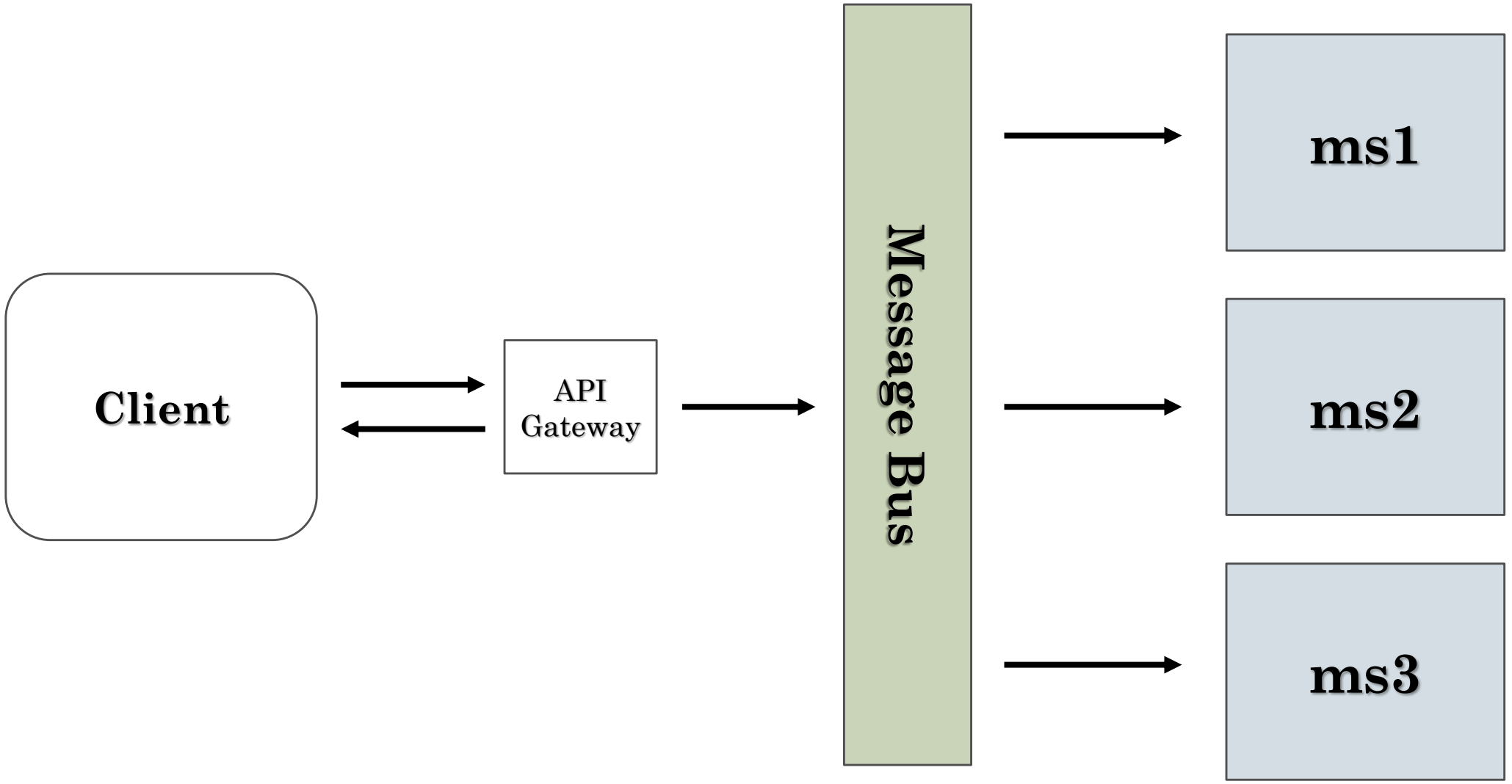
Asynchronous Communication

Message Bus **vs** ***RPC***



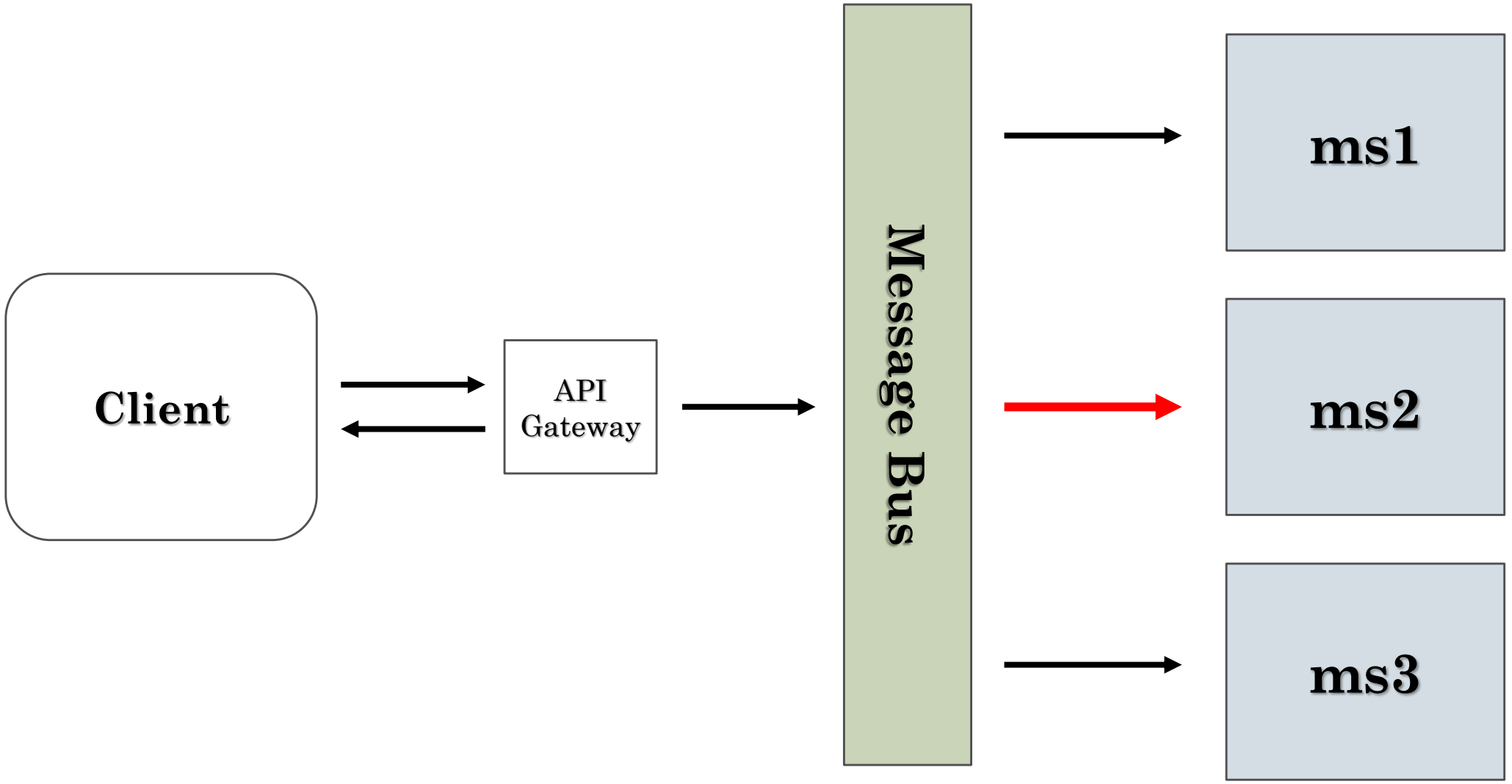
Asynchronous Communication

Message Bus **vs** ***RPC***



Asynchronous Communication

Message Bus fault tolerance



Asynchronous Communication

The *Saga* Pattern

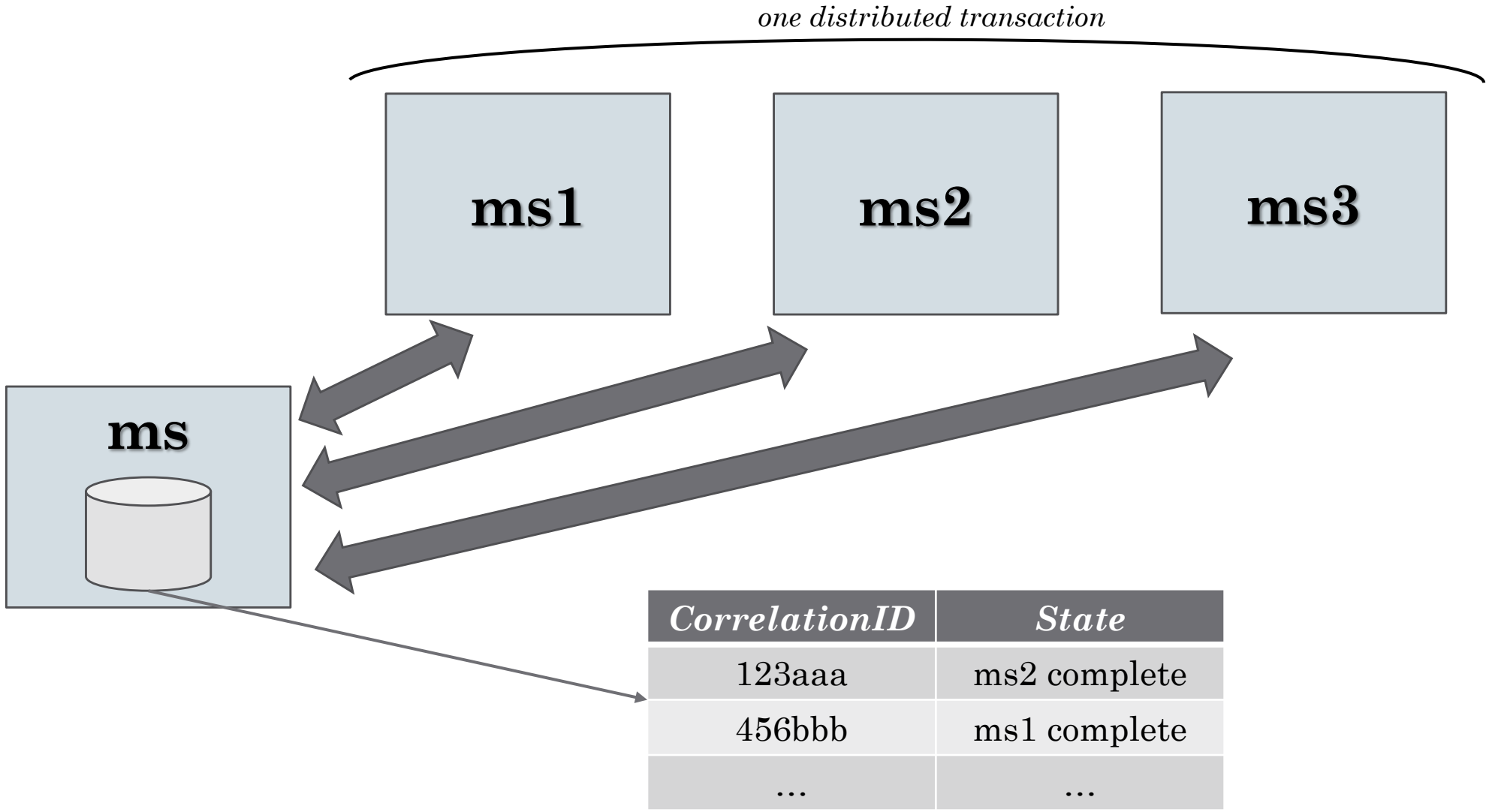
- Всяка транзакция, която обхваща множество услуги се изпълнява като ***Сага*** (*Saga*). Сагата е поредица от локални (*local*) транзакции.

- Всяка локална транзакция актуализира базата данни и публикува съобщение за да задейства следващата локална транзакция в сагата.

- Ако една транзакция се провали, сагата изпълнява серия от компенсиращи (*compensations*) транзакции, които отменят промените, направени от предходните транзакции.

Asynchronous Communication

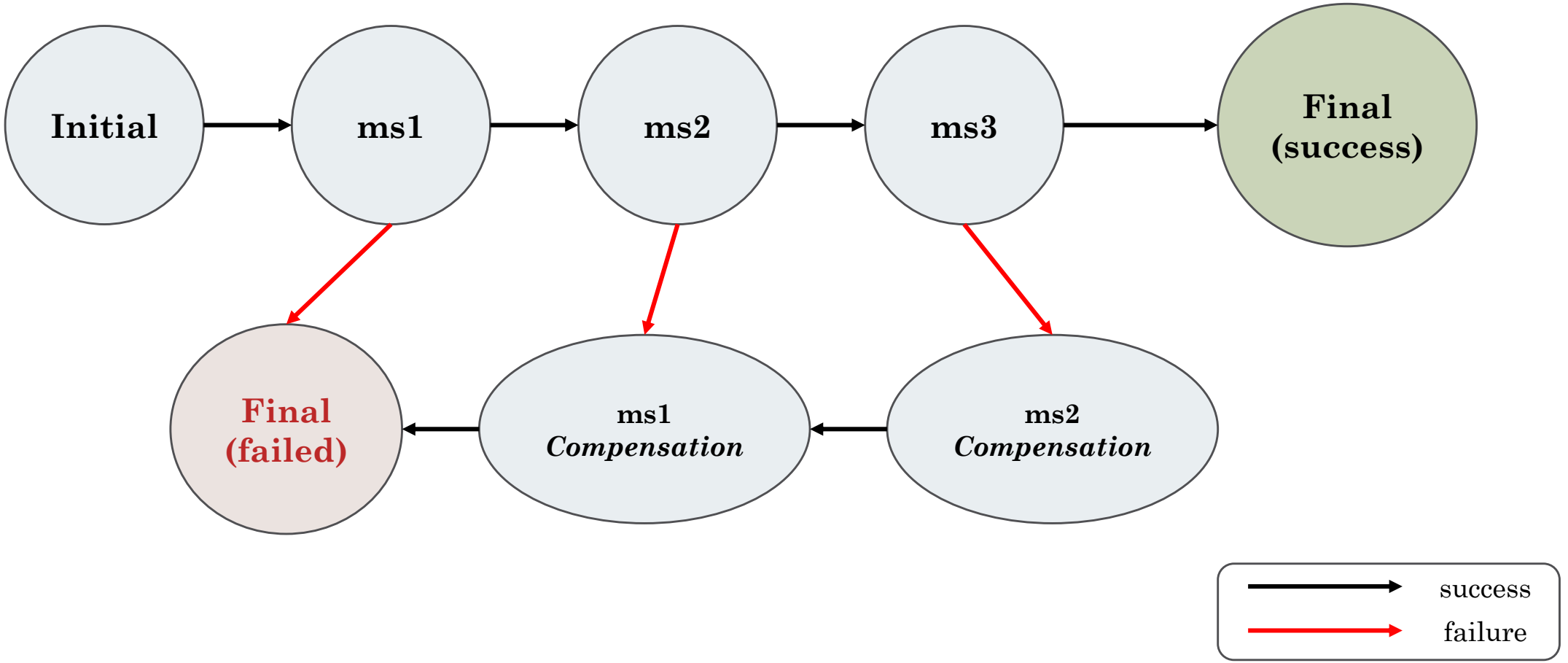
Saga pattern example



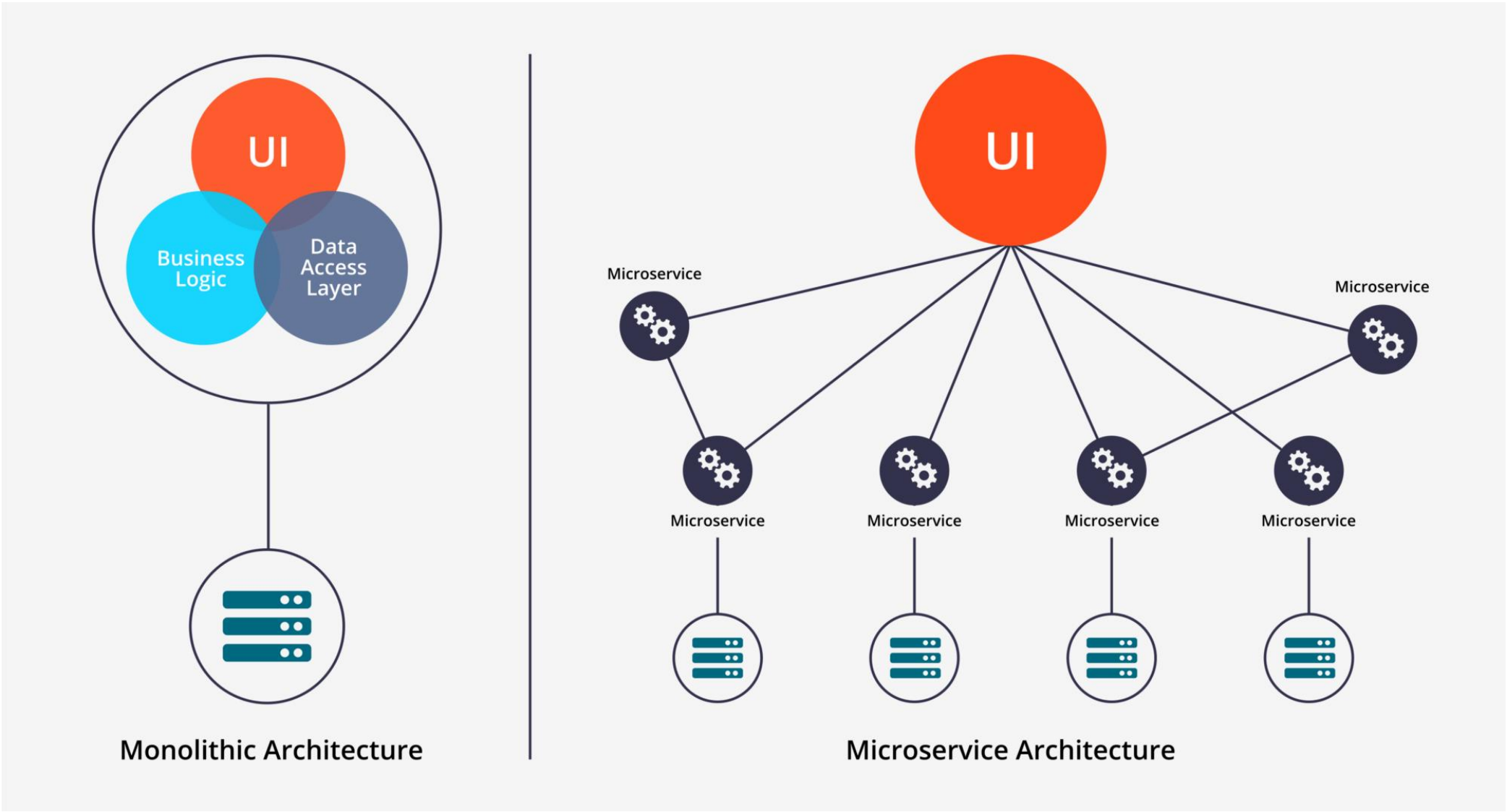
CorrelationID	*State*
123aaa	ms2 complete
456bbb	ms1 complete
…	…

Asynchronous Communication

State Machine in Saga



Conclusion



src: *https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4*