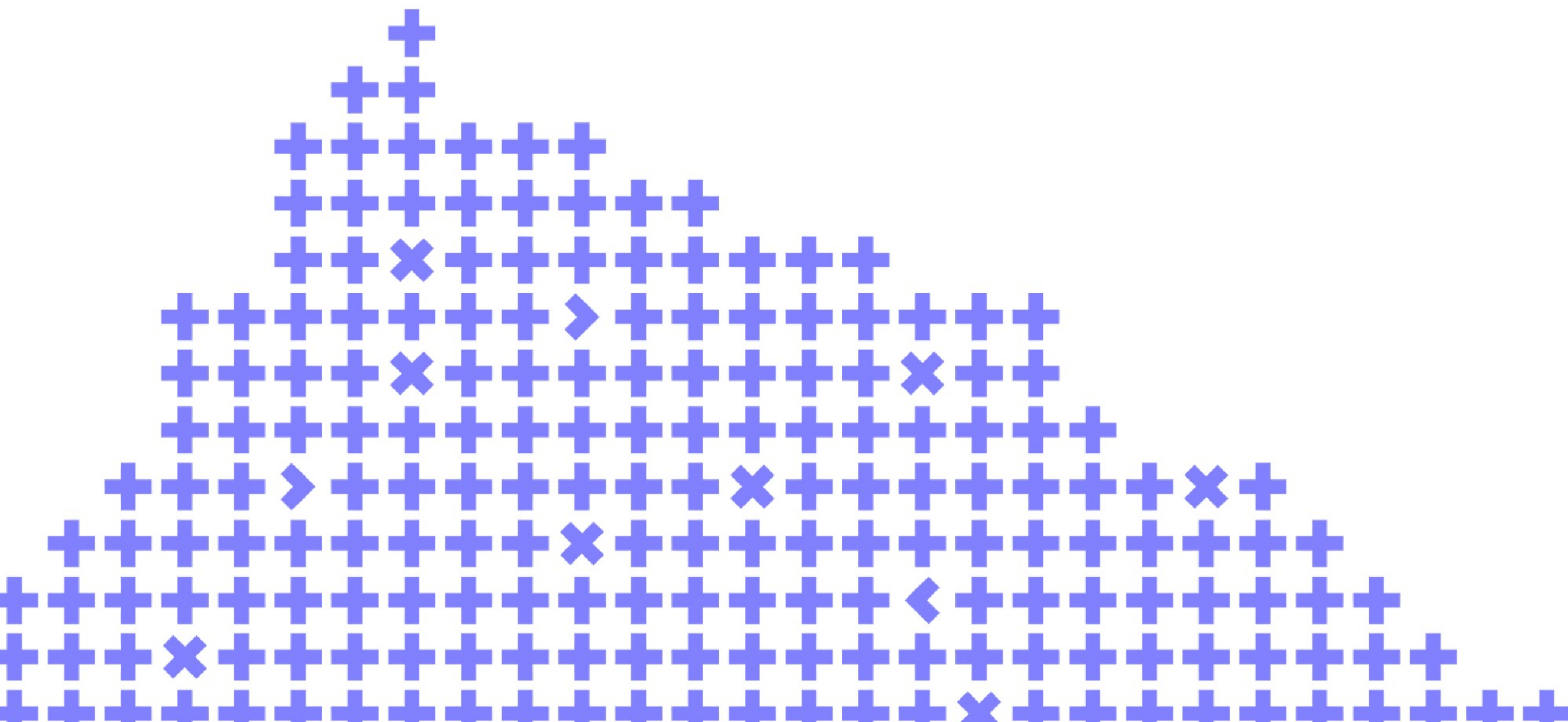


# Microservices on C++

## or Why We Made Our Own Framework

Antony Polukhin



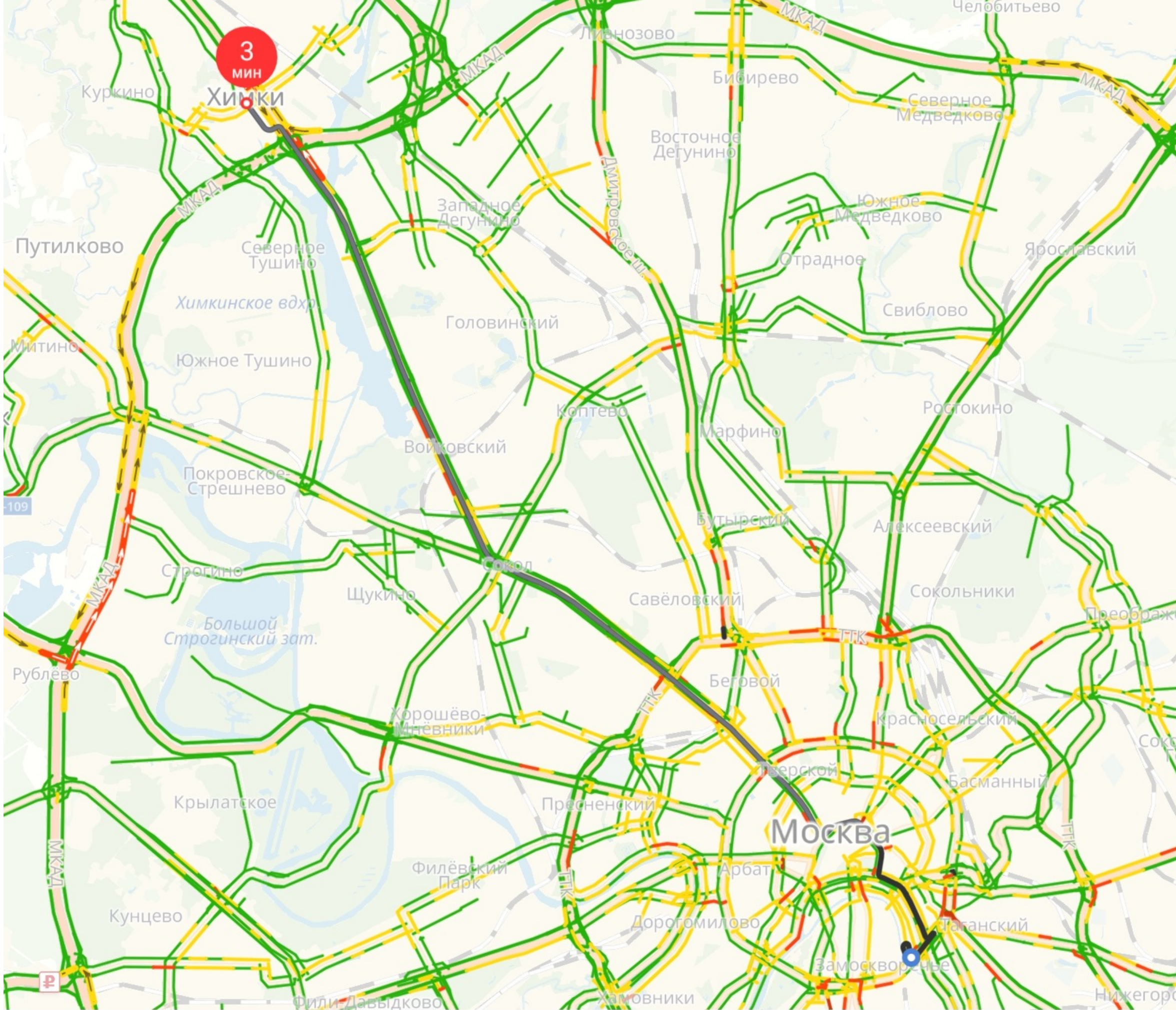
Co-organizer

**Yandex**



# Table of Contents

- The Task
- The Pains
- Fixing the Pains
- The Result



monolith



microservices



ЭКОНОМ  
4₽



КОМФОРТ  
8₽



КОМФОРТ+  
9₽



БИЗНЕС  
34₽



МИНИВЭН  
15₽



ДЕТСКИЙ  
2₽

Комментарий, пожелания

Способ оплаты  
Команда Яндекс.Такси



# The Task

The Task — make everything  
better

# The Task

# The Task

We need

# The Task

We need:

- Efficiency

# The Task

We need:

- Efficiency
- Simplicity of development



# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have



# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Monolith architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- **Monolith architecture**



# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Monolith architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Monolith architecture

# The Task

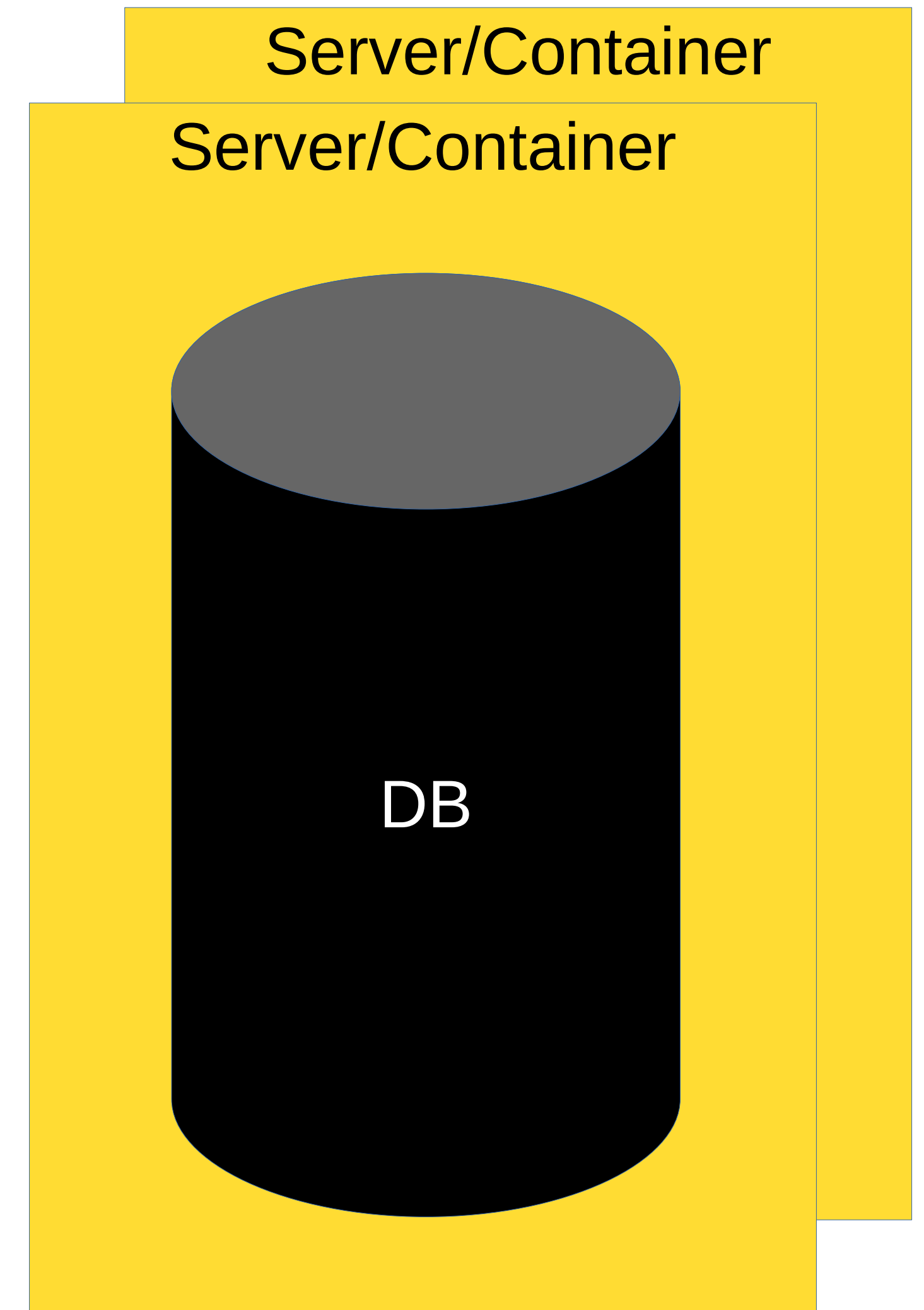
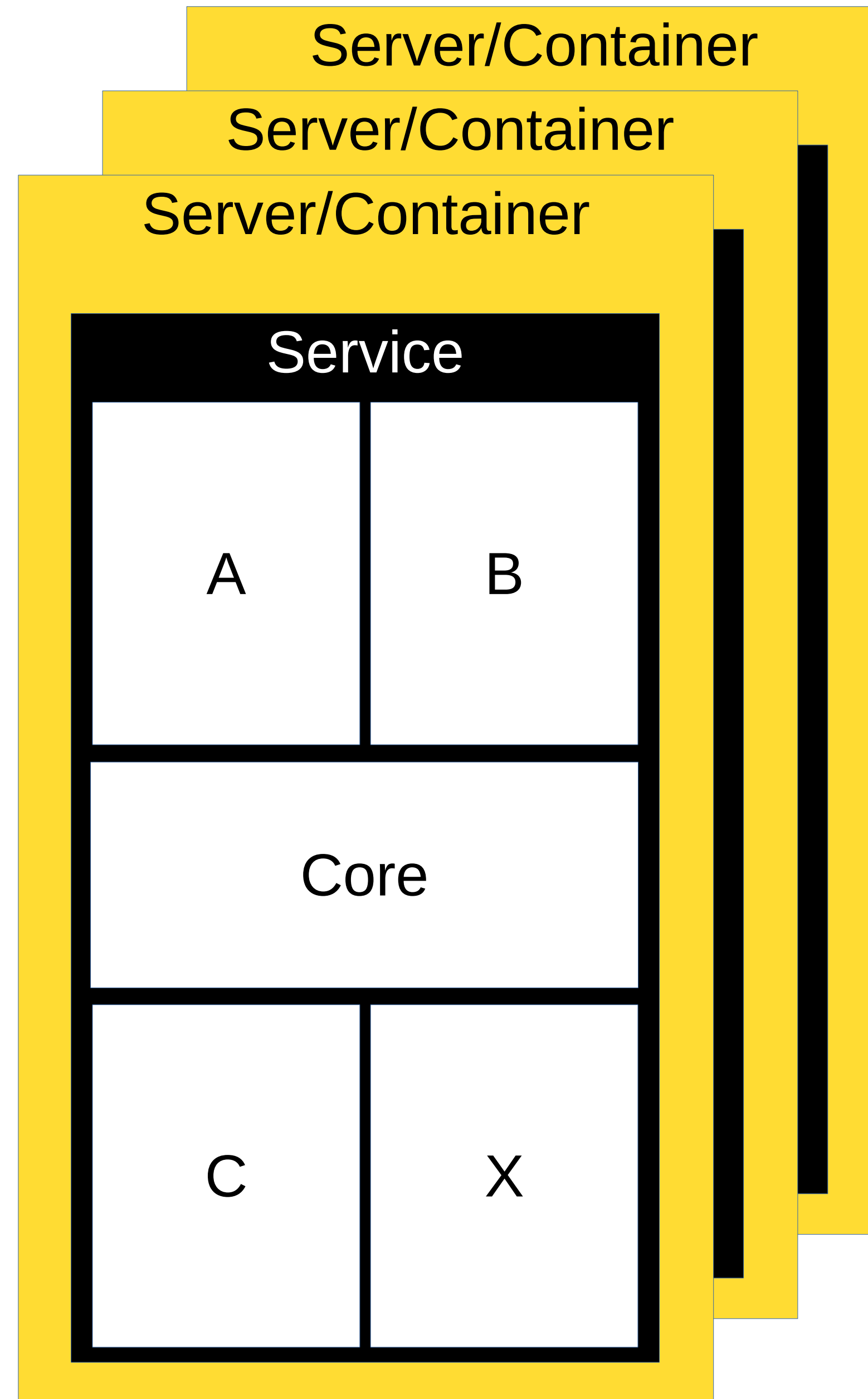
We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

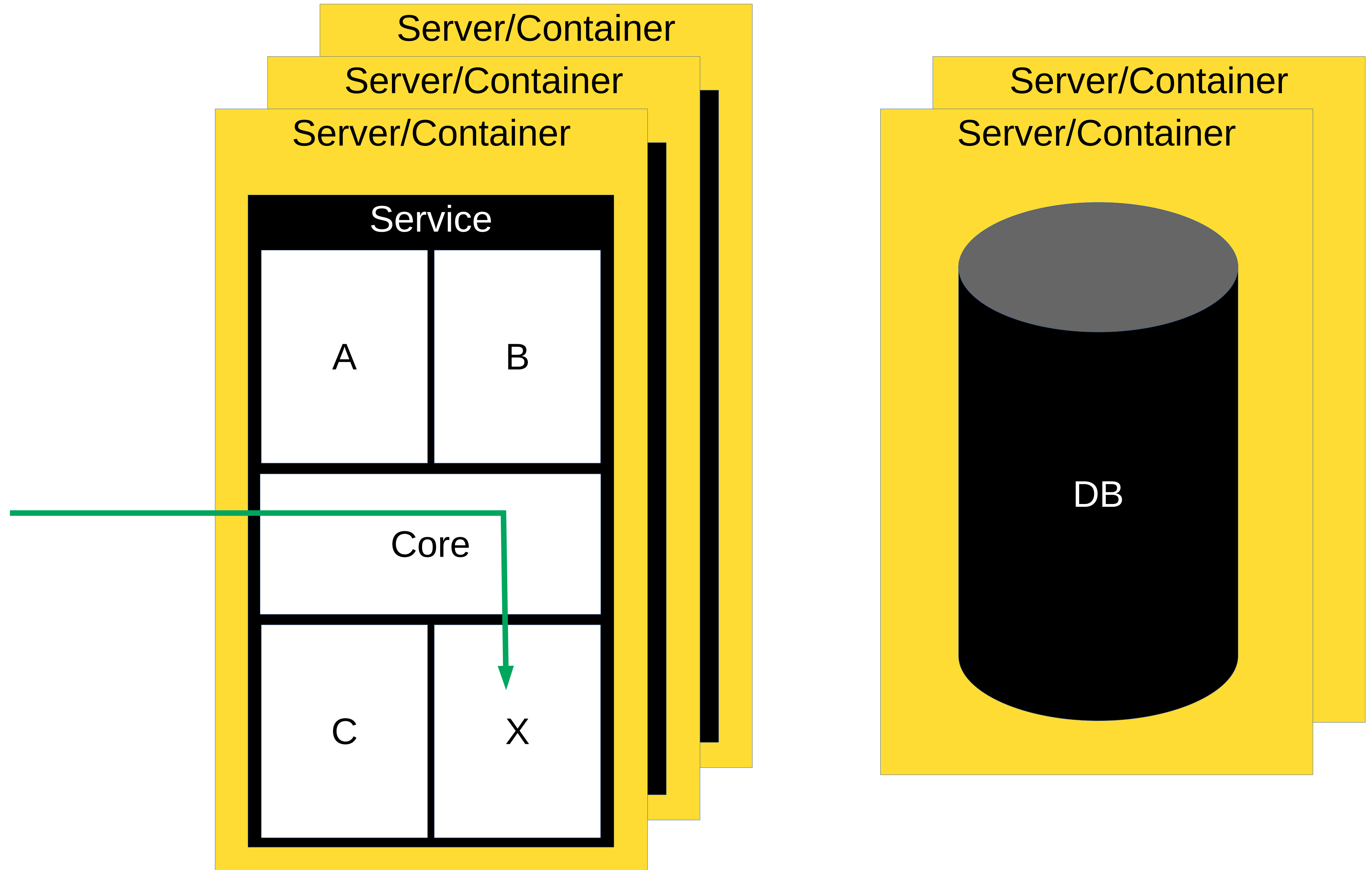
- Many small teams
- Huge C++ codebase
- Monolith architecture

# Safety

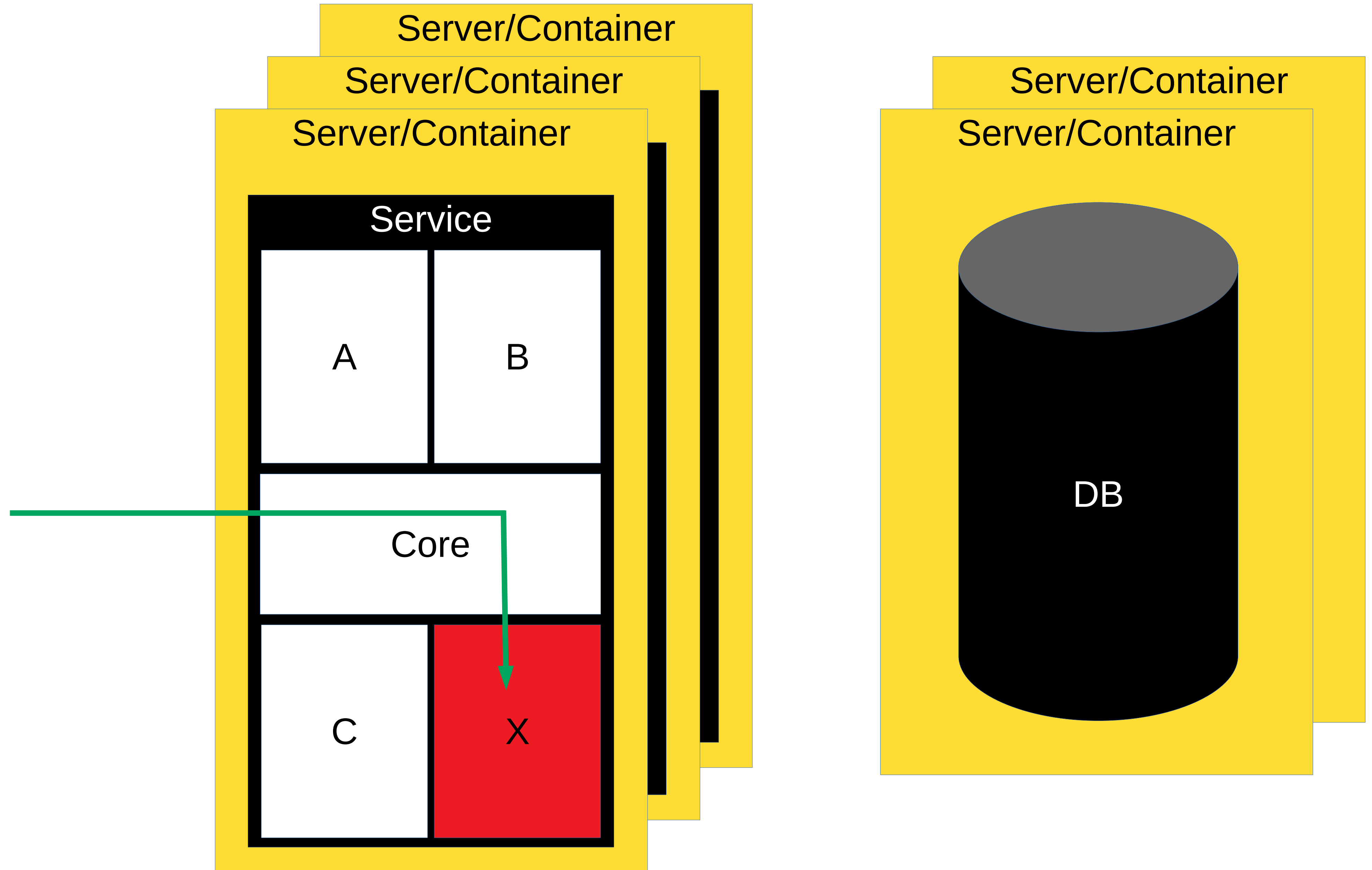




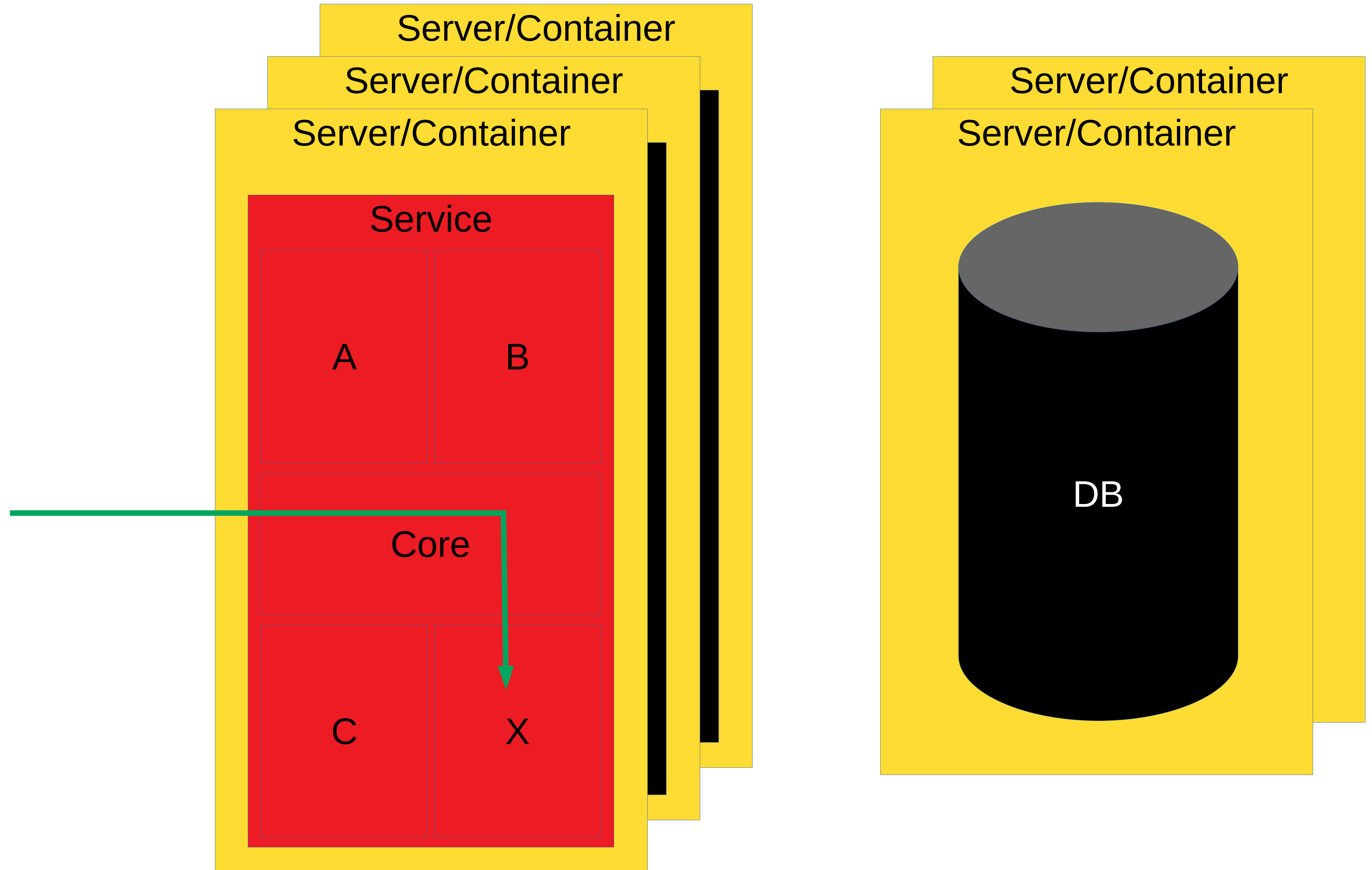
# Safety



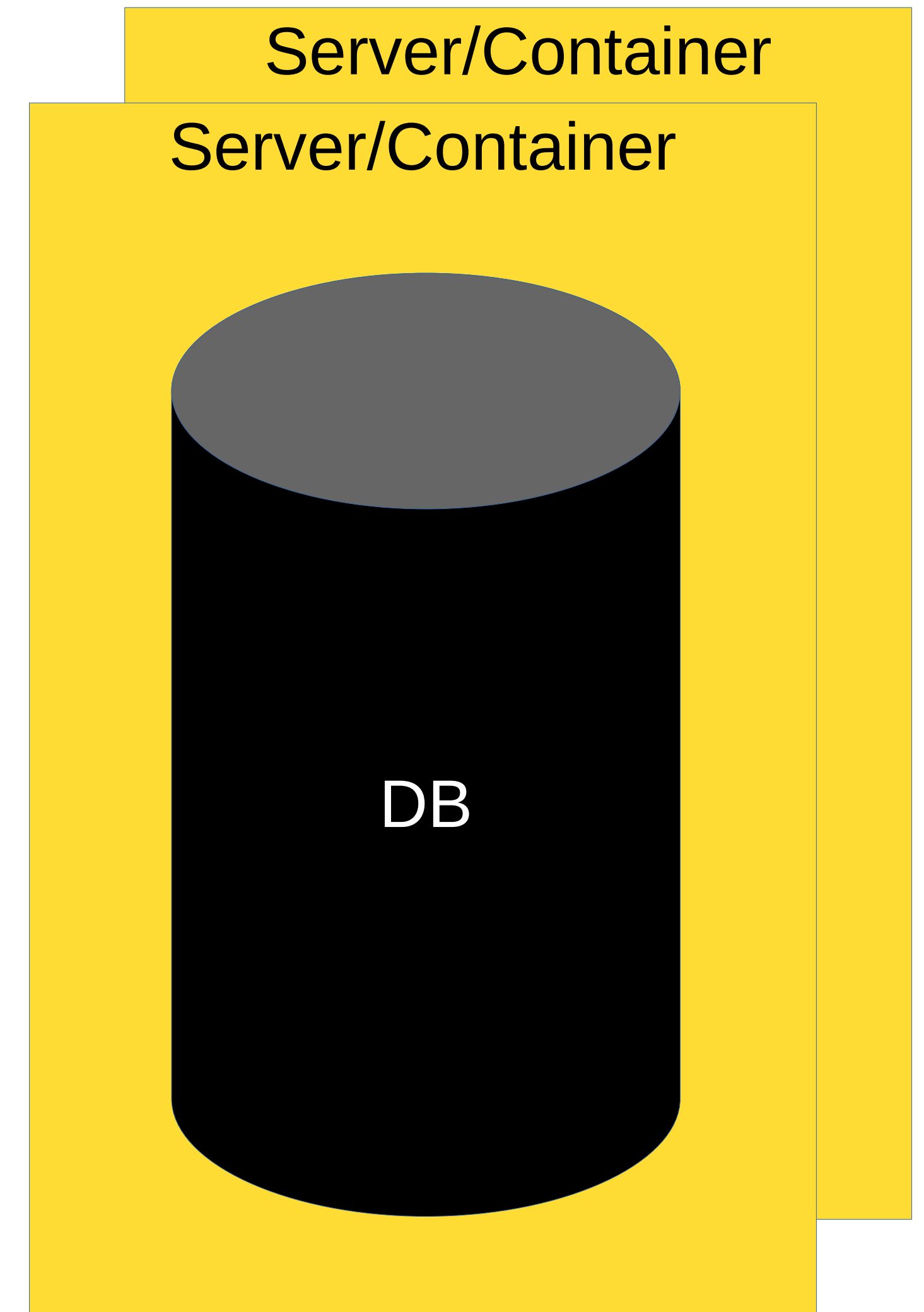
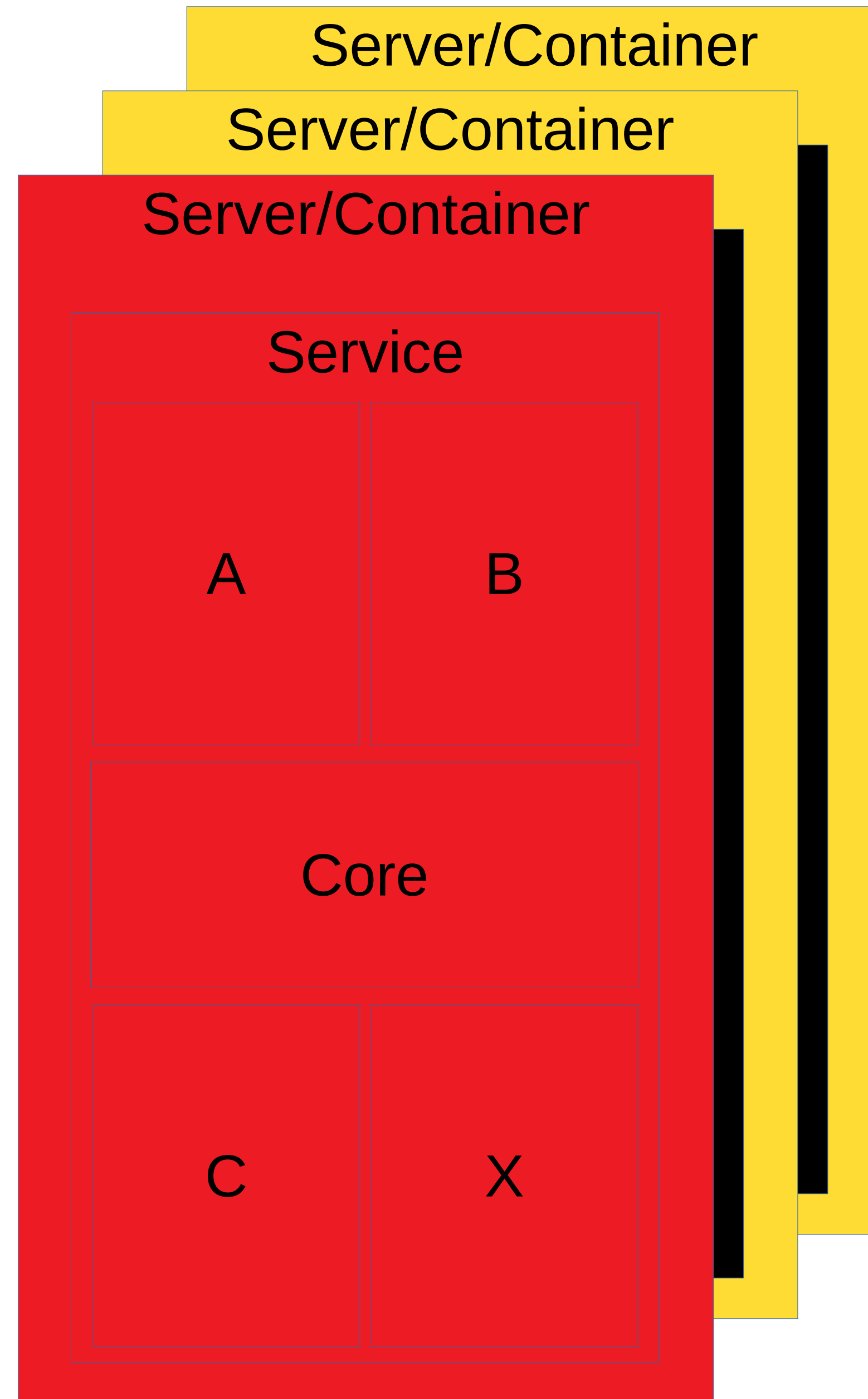
# Safety



# Safety

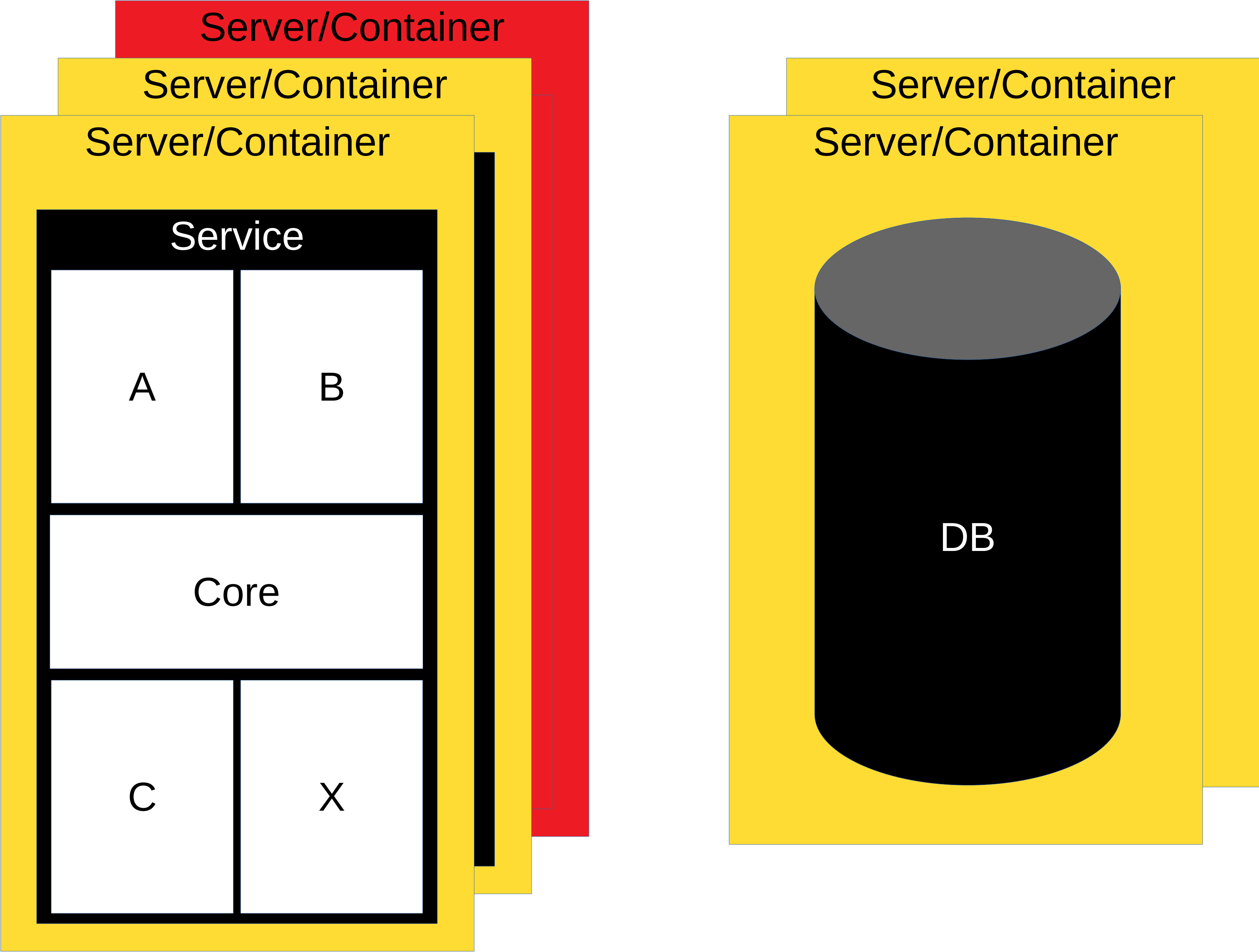


# Safety

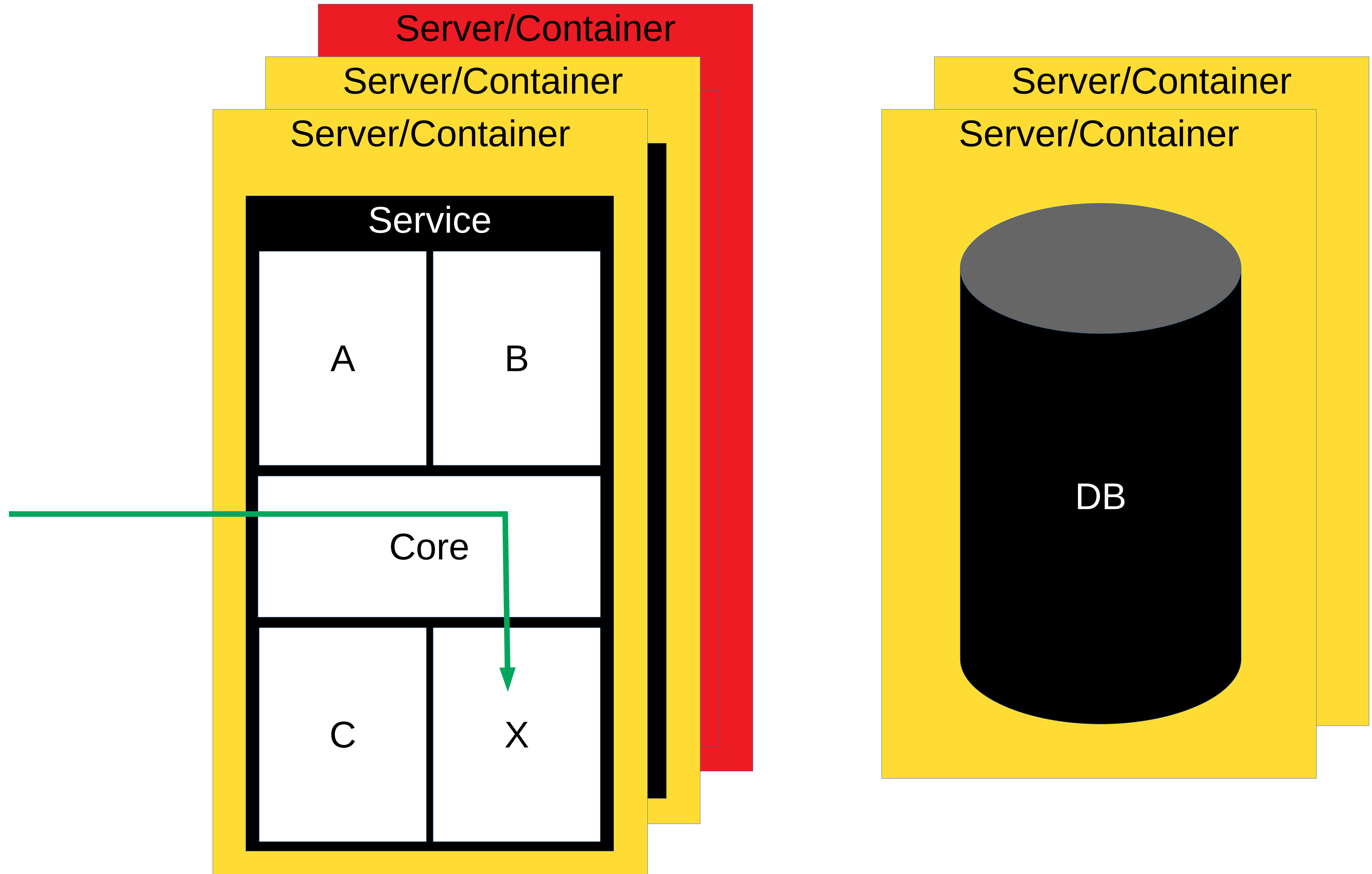




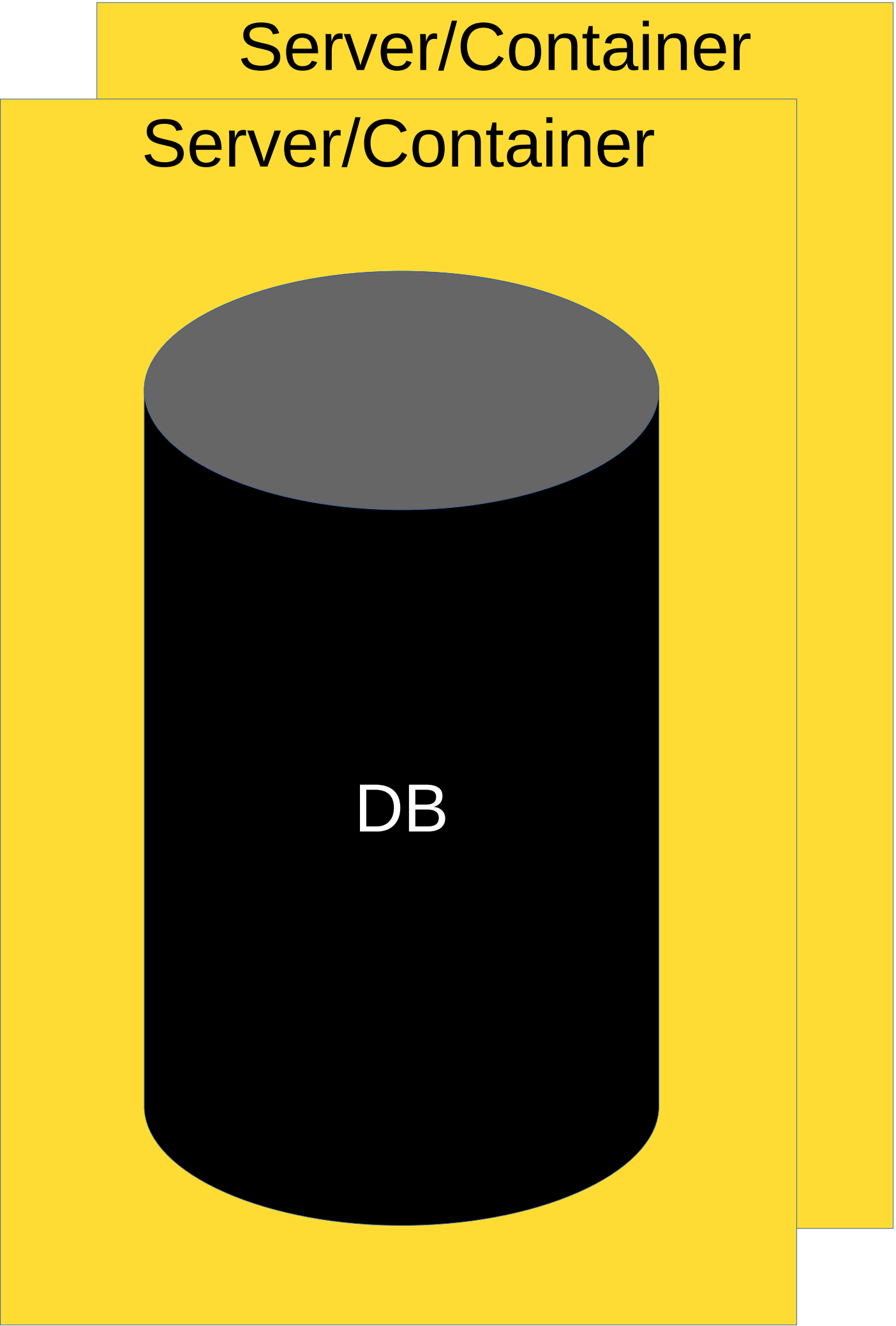
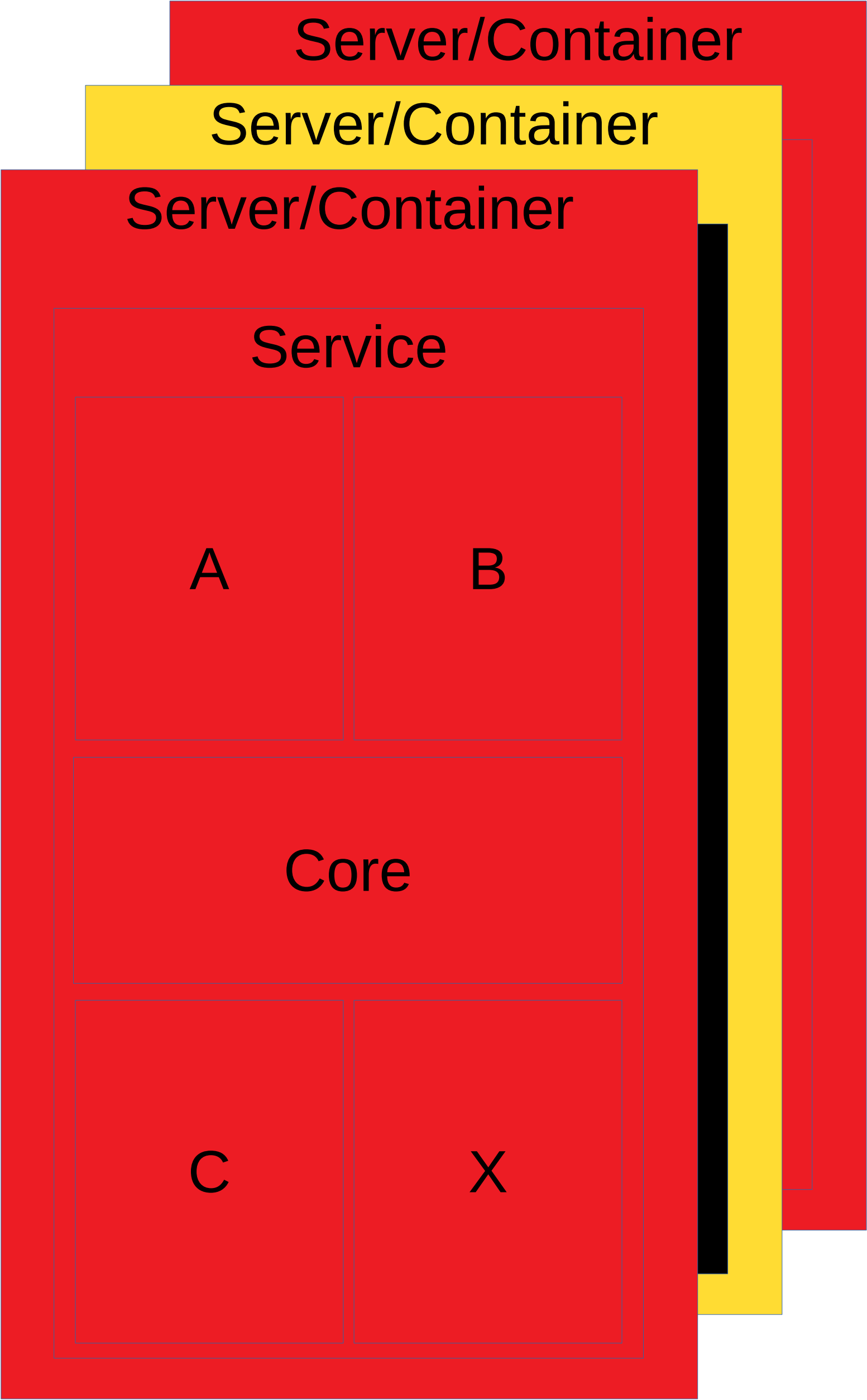
# Safety



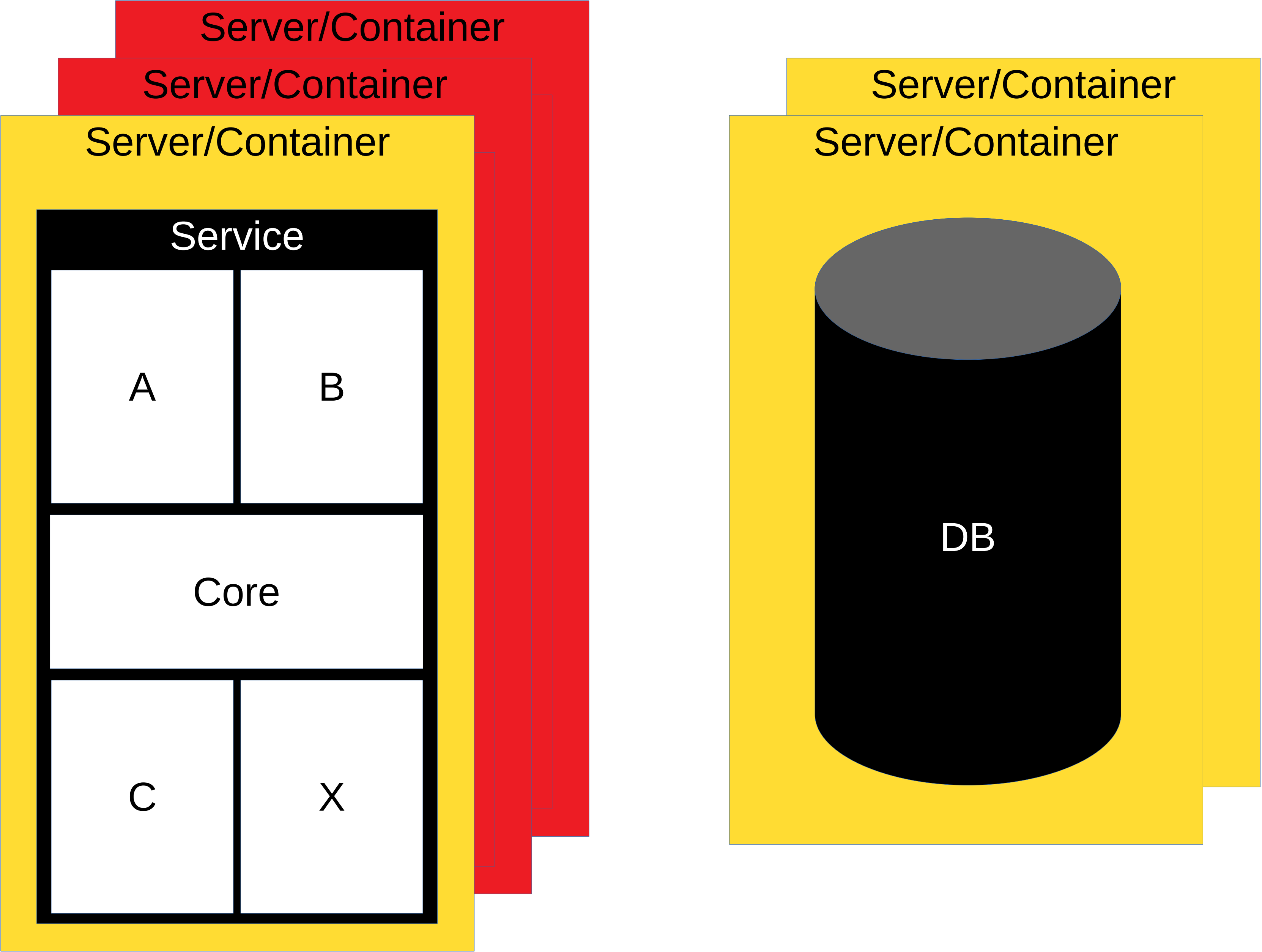
# Safety



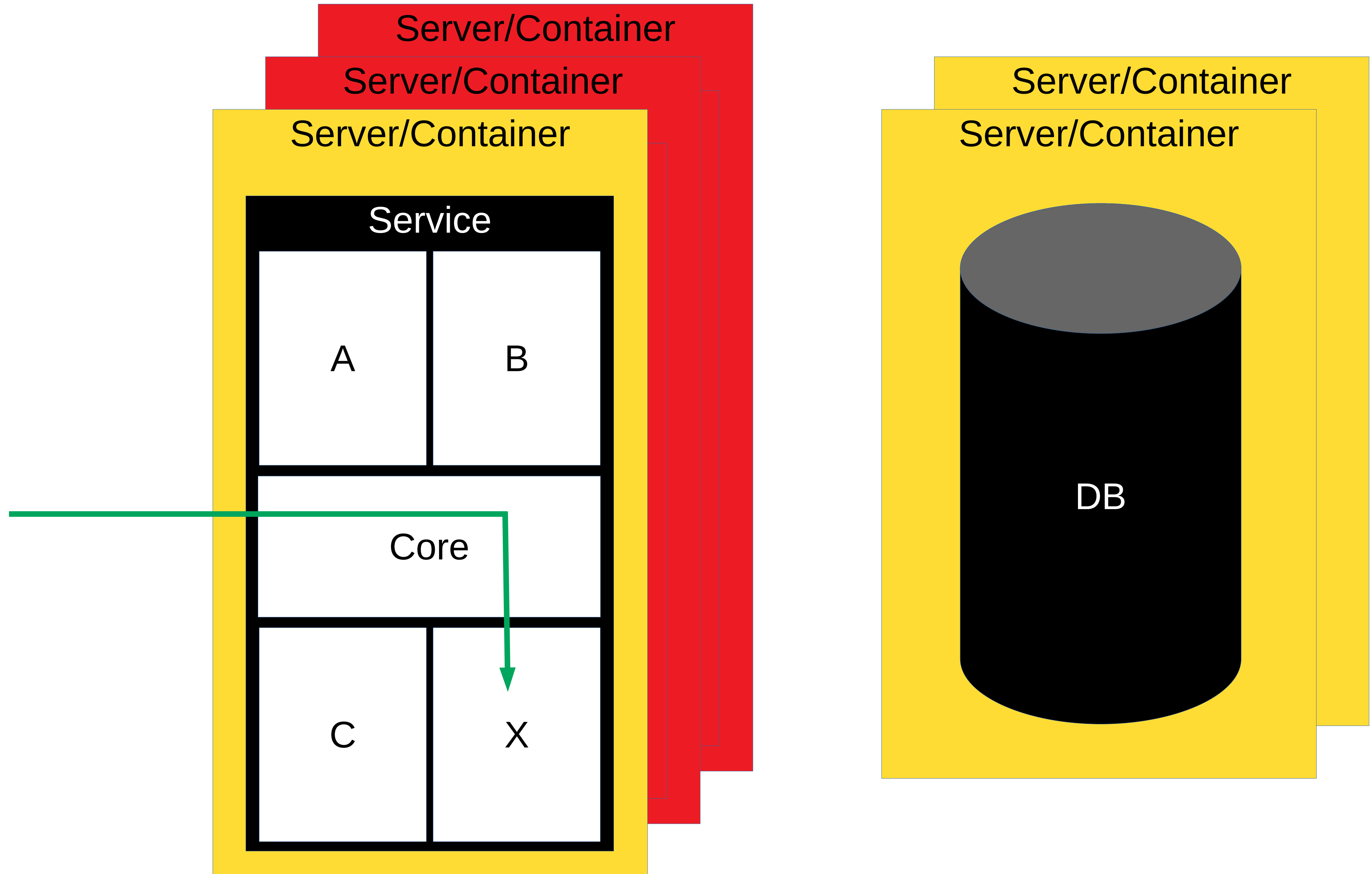
# Safety



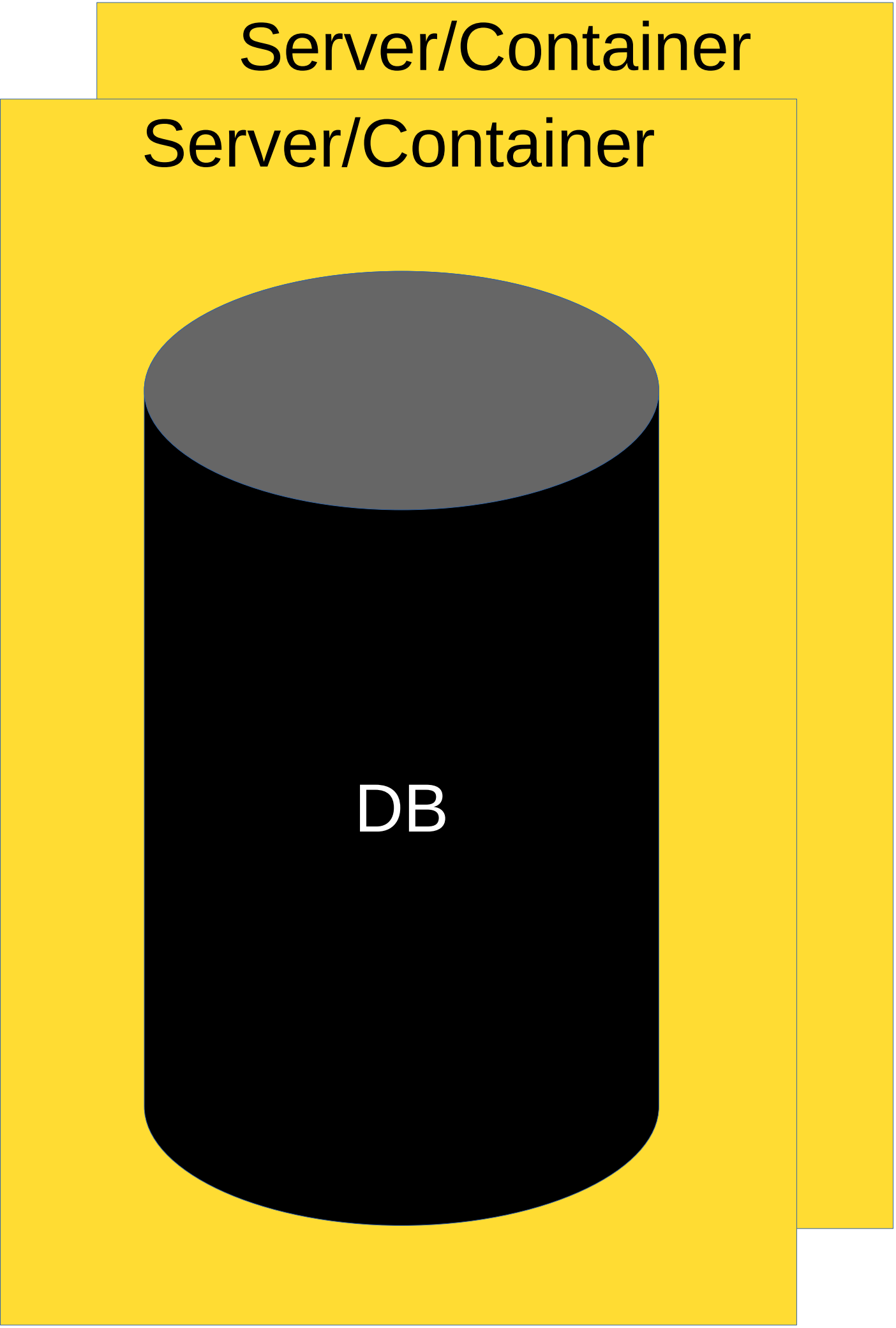
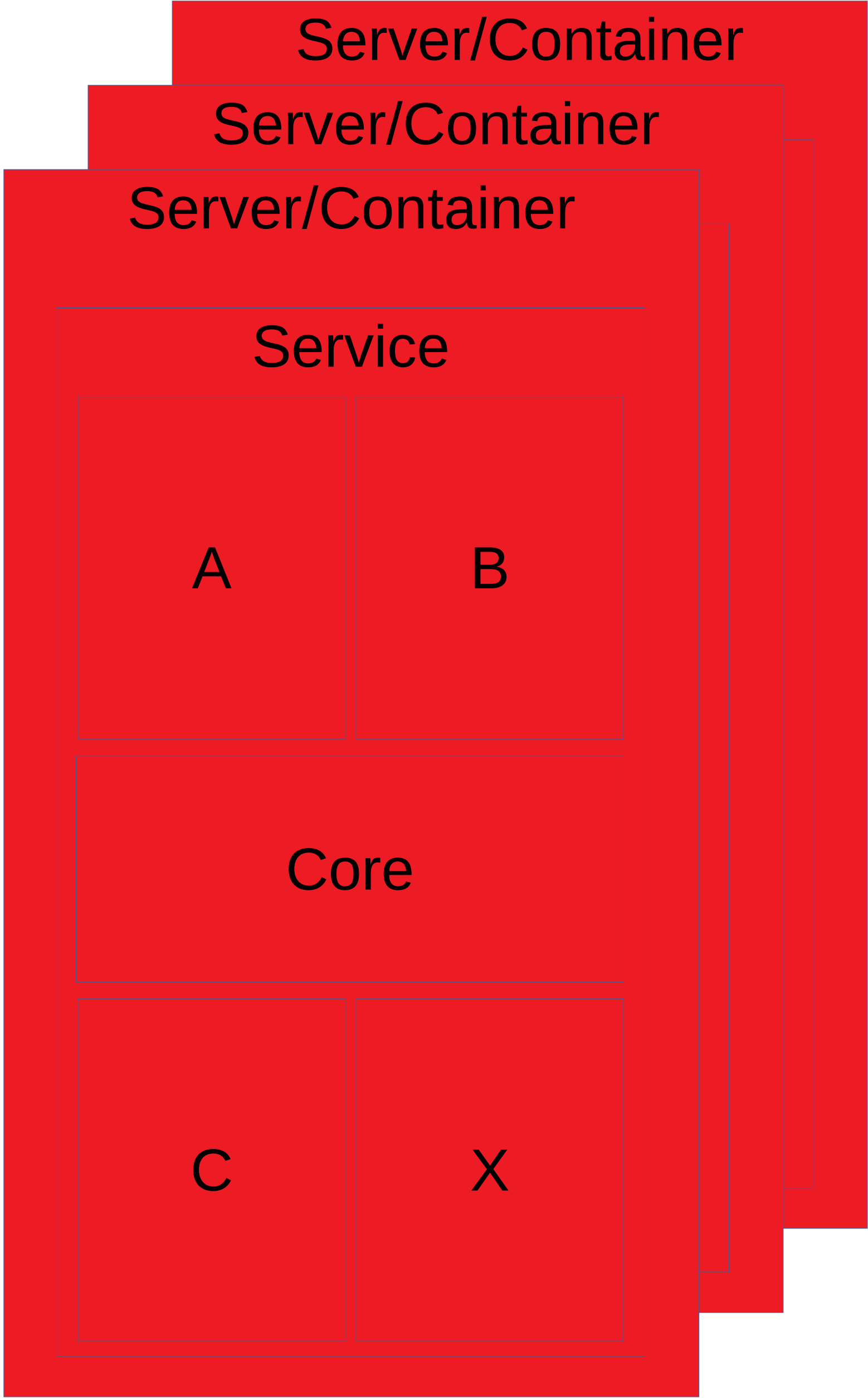
# Safety



# Safety

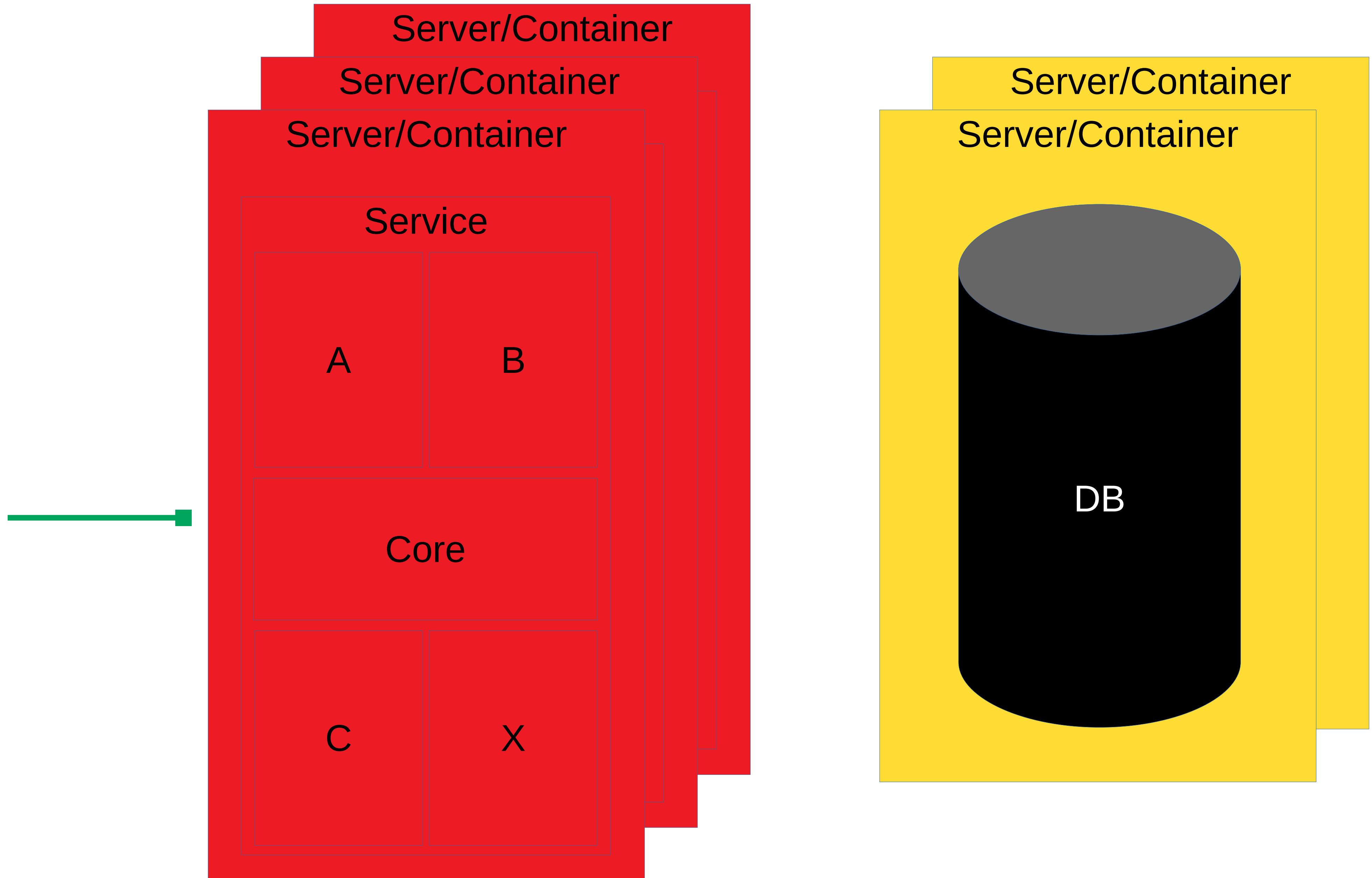


# Safety

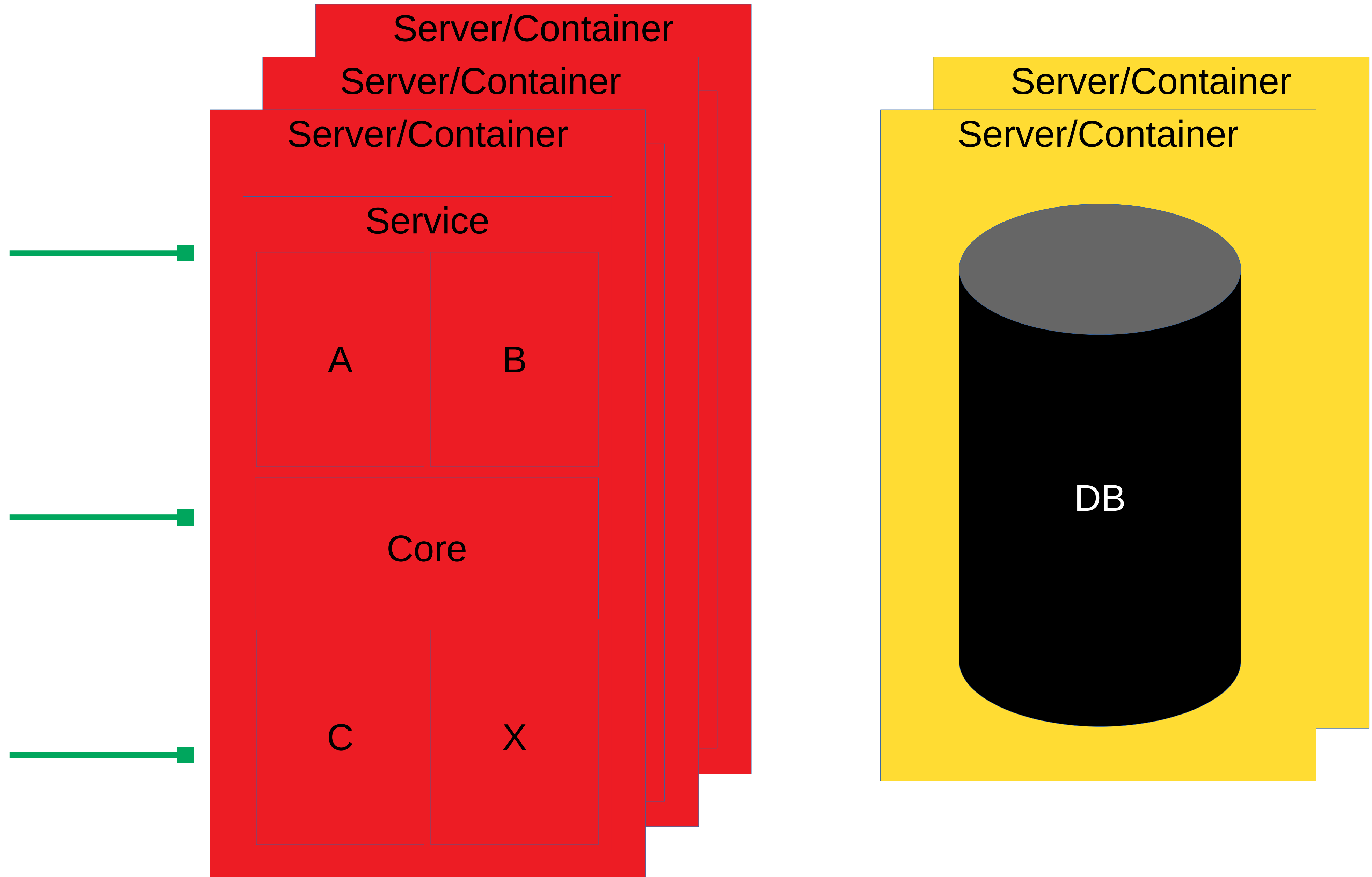




# Safety



# Safety



# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- **Safety**
- **Scalability**

We already have:

- **Many small teams**
- Huge C++ codebase
- **Monolith architecture**

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Monolith architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- ~~Monolith~~ Microservice architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- **Microservice architecture**

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Microservice architecture



# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Microservice architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Microservice architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Microservice architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Microservice architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Microservice architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Microservice architecture

# The Task

We need:

- Efficiency
- Simplicity of development
- High development speed
- Safety
- Scalability

We already have:

- Many small teams
- Huge C++ codebase
- Microservice architecture



# The Pains

# The Pains

# The Pains



Existing C++ frameworks:

# The Pains



Existing C++ frameworks:

- Provoke callback hell

# The Pains



Existing C++ frameworks:

- Provoke callback hell that slows down development and provoke errors

# The Pains



## Existing C++ frameworks:

- Provoke callback hell that slows down development and provoke errors
- No nice way to do experiments, downtimes are unavoidable

# The Pains



## Existing C++ frameworks:

- Provoke callback hell that slows down development and provoke errors
- No nice way to do experiments, downtimes are unavoidable

## Microservices:

# The Pains

## Existing C++ frameworks:

- Provoke callback hell that slows down development and provoke errors
- No nice way to do experiments, downtimes are unavoidable

## Microservices:

- Increase latencies



# The Pains

## Existing C++ frameworks:

- Provoke callback hell that slows down development and provoke errors
- No nice way to do experiments, downtimes are unavoidable

## Microservices:

- Increase latencies

C++

# The Pains

## Existing C++ frameworks:

- Provoke callback hell that slows down development and provoke errors
- No nice way to do experiments, downtimes are unavoidable

## Microservices:

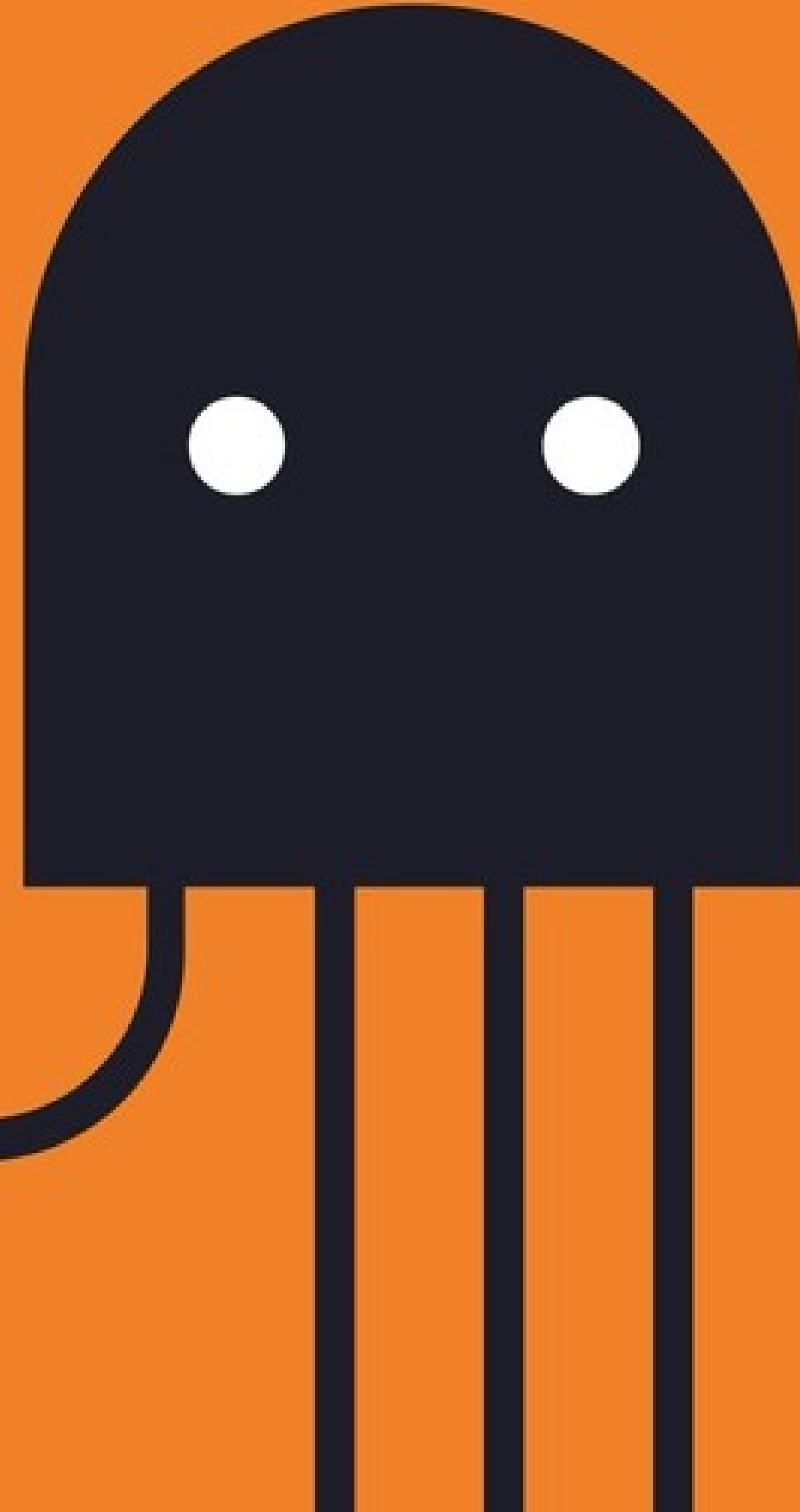
- Increase latencies

## C++

- Frightens fragile minds Ж:-)

# So we made our own framework

<https://userver.tech/>



# Callback Hell

# Callback Hell



```
void View::Handle(Request&& request, const Dependencies& dependencies, Response
response) {
    dependencies.pg->GetCluster(
        [request = std::move(request), response](auto cluster)
        {
            cluster->Begin(storages::postgres::ClusterHostType::kMaster,
                [request = std::move(request), response](auto& trx)
                {
                    const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
                    psql::Execute(trx, statement, request.id,
                        [request = std::move(request), response, trx = std::move(trx)](auto& res)
                        {
                            auto row = res[0];
                            if (!row["ok"].As<bool>()) {
                                if (LogDebug()) {
                                    GetSomeInfoFromDb([id = request.id](auto info) {
                                        LOG_DEBUG() << id << " is not OK of " << info;
                                    });
                                }
                            }
                        }
                    );
                }
            );
        }
    );
}
```

# Callback Hell

```
    }
    *response = Response400{};
}
psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar,
    [row = std::move(row), trx = std::move(trx), response]()
{
    trx.Commit([row = std::move(row), response]() {
        *response = Response200{row["baz"].As<std::string>()};
    });
});
});
});
});
}
```

# Callback Hell



# ~~Callback Hell~~ Coroutines

# Coroutines



```
Response View::Handle(Request&& request, const Dependencies& dependencies) {  
    auto cluster = co_await dependencies.pg->GetCluster();  
  
    auto trx = co_await cluster->Begin(postgres::ClusterHostType::kMaster);  
  
    const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";  
    auto row = co_await psql::Execute(trx, statement, request.id)[0];  
    if (!row["ok"].As<bool>()) {  
        LOG_DEBUG() << request.id << " is not OK of "  
            << co_await GetSomeInfoFromDb();  
        return Response400();  
    }  
  
    co_await psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);  
    co_await trx.Commit();  
  
    return Response200{row["baz"].As<std::string>()};  
}
```

# Coroutines



```
Response View::Handle(Request&& request, const Dependencies& dependencies) {  
    auto cluster = co_await dependencies.pg->GetCluster();  
  
    auto trx = co_await cluster->Begin(postgres::ClusterHostType::kMaster);  
  
    const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";  
    auto row = co_await psql::Execute(trx, statement, request.id)[0];  
    if (!row["ok"].As<bool>()) {  
        LOG_DEBUG() << request.id << " is not OK of "  
            << co_await GetSomeInfoFromDb();  
        return Response400();  
    }  
  
    co_await psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);  
    co_await trx.Commit();  
  
    return Response200{row["baz"].As<std::string>()};  
}
```

# Coroutines



```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
    auto cluster = co_await dependencies.pg->GetCluster();

    auto trx = co_await cluster->Begin(postgres::ClusterHostType::kMaster);

    const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";
    auto row = co_await psql::Execute(trx, statement, request.id)[0];
    if (!row["ok"].As<bool>()) {
        LOG_DEBUG() << request.id << " is not OK of "
            << co_await GetSomeInfoFromDb();
        return Response400();
    }

    co_await psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);
    co_await trx.Commit();

    return Response200{row["baz"].As<std::string>()};
}
```

# It's 2017

It's 2017 – C++ does not have  
coroutines

# Stackless Coroutines



```
Response View::Handle(Request&& request, const Dependencies& dependencies) {  
    auto cluster = co_await dependencies.pg->GetCluster();  
    auto trx = co_await cluster->Begin(postgres::ClusterHostType::kMaster);  
  
    const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";  
    auto row = co_await psql::Execute(trx, statement, request.id)[0];  
    if (!row["ok"].As<bool>()) {  
        LOG_DEBUG() << request.id << " is not OK of "  
            << co_await GetSomeInfoFromDb();  
        return Response400();  
    }  
  
    co_await psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);  
    co_await trx.Commit();  
  
    return Response200{row["baz"].As<std::string>()};  
}
```

# Stackfull Coroutines



```
Response View::Handle(Request&& request, const Dependencies& dependencies) {  
    auto cluster = dependencies.pg->GetCluster();  
    auto trx = cluster->Begin(storages::postgres::ClusterHostType::kMaster);  
  
    const char* statement = "SELECT ok, baz FROM some WHERE id = $1 LIMIT 1";  
    auto row = psql::Execute(trx, statement, request.id)[0];  
    if (!row["ok"].As<bool>()) {  
        LOG_DEBUG() << request.id << " is not OK of "  
            << GetSomeInfoFromDb();  
        return Response400();  
    }  
  
    psql::Execute(trx, queries::kUpdateRules, request.foo, request.bar);  
    trx.Commit();  
  
    return Response200{row["baz"].As<std::string>()};  
}
```



# Experiments

# Experiments

# New functionality

For example, we plan to introduce the new functionality «payed roads»

# New functionality

For example, we plan to introduce the new functionality «payed roads»

- Code is written

# New functionality

For example, we plan to introduce the new functionality «payed roads»

- Code is written
- Code is tested

# New functionality

For example, we plan to introduce the new functionality «payed roads»

- Code is written
- Code is tested
- It breaks in production

# What to do?

# Wrong solutions



# Wrong solutions

- Fix the code and redeploy

# Wrong solutions

- Fix the code and redeploy
- Or change the configuration files and redeploy

# The Right Solution

# The Right Solution – Dynamic Configs

# The Right Solution

# The Right Solution

Service that distributes configs

# The Right Solution

Service that distributes configs:

- Change the config value from the browser

# The Right Solution

Service that distributes configs:

- Change the config value from the browser
- The change is applied automatically



# The Right Solution

Service that distributes configs:

- Change the config value from the browser
- The change is applied automatically

Features:

# The Right Solution

Service that distributes configs:

- Change the config value from the browser
- The change is applied automatically

Features:

- Safe deployment of a new functionality

# The Right Solution

Service that distributes configs:

- Change the config value from the browser
- The change is applied automatically

Features:

- Safe deployment of a new functionality
- Experiments

# The Right Solution

Service that distributes configs:

- Change the config value from the browser
- The change is applied automatically

Features:

- Safe deployment of a new functionality
- Experiments
- Limits/timeouts/log levels/degradation...

# Dynamic Configs

```
int Component::DoSomething() const {  
    const auto runtime_config = config_.GetSnapshot();  
    return runtime_config[kMyConfig];  
}
```

# Dynamic Configs

```
int Component::DoSomething() const {  
    const auto runtime_config = config_.GetSnapshot();  
    return runtime_config[kMyConfig];  
}
```

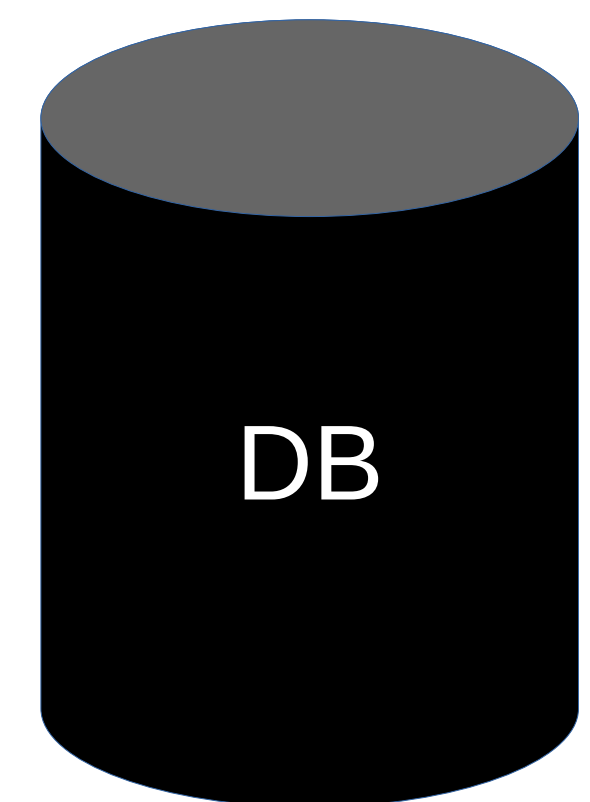
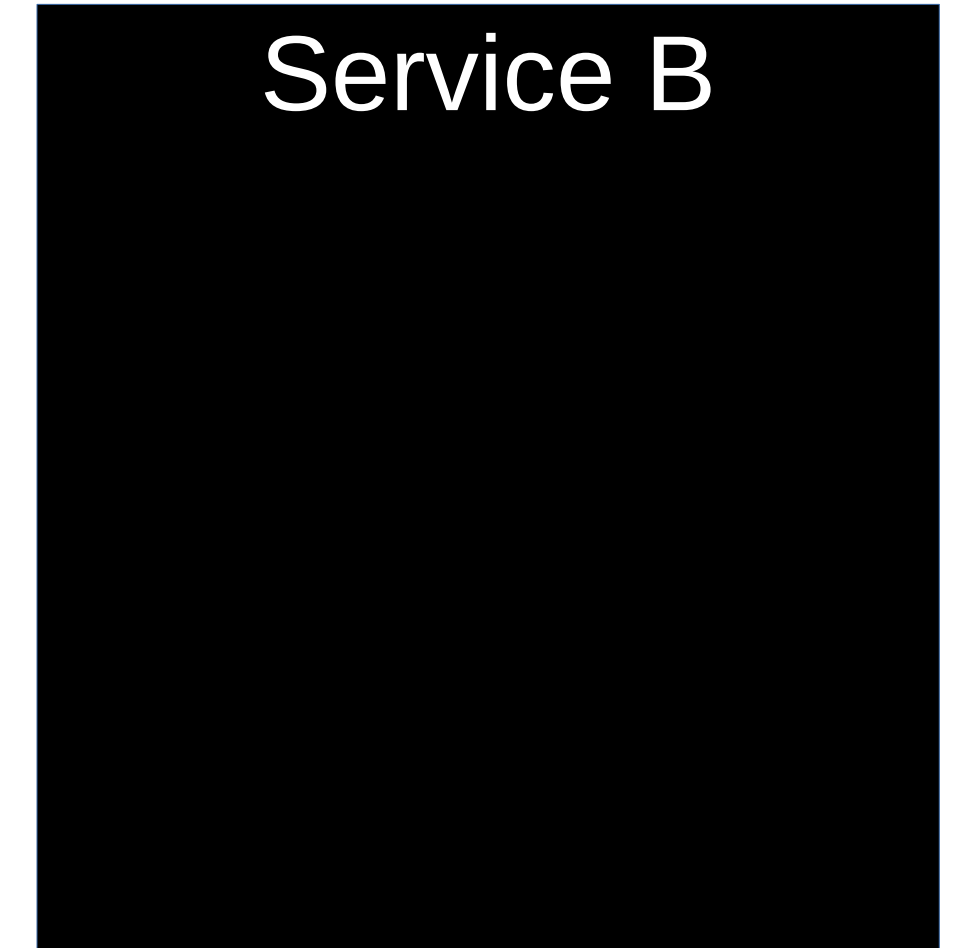
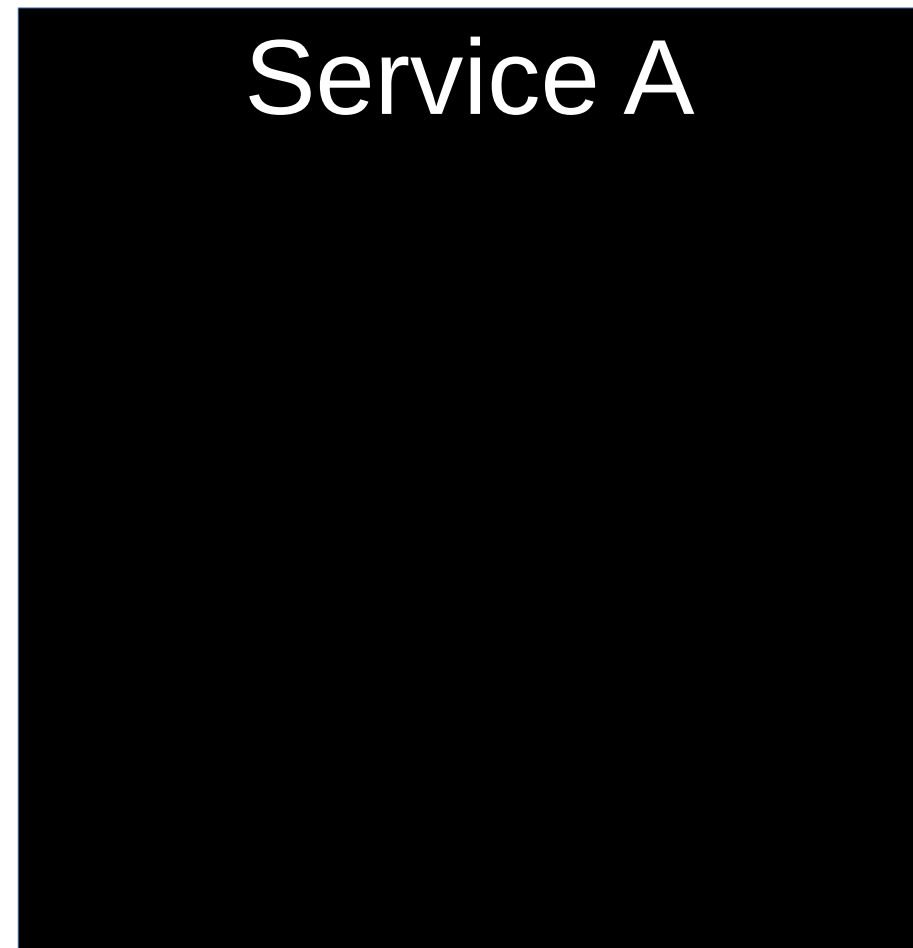
# Dynamic Configs

```
int Component::DoSomething() const {  
    const auto runtime_config = config_.GetSnapshot();  
    return runtime_config[kMyConfig];  
}
```

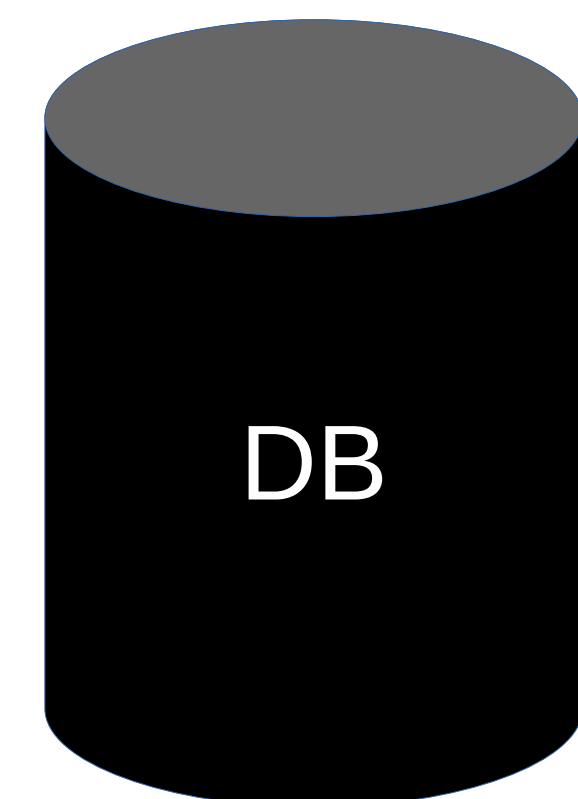
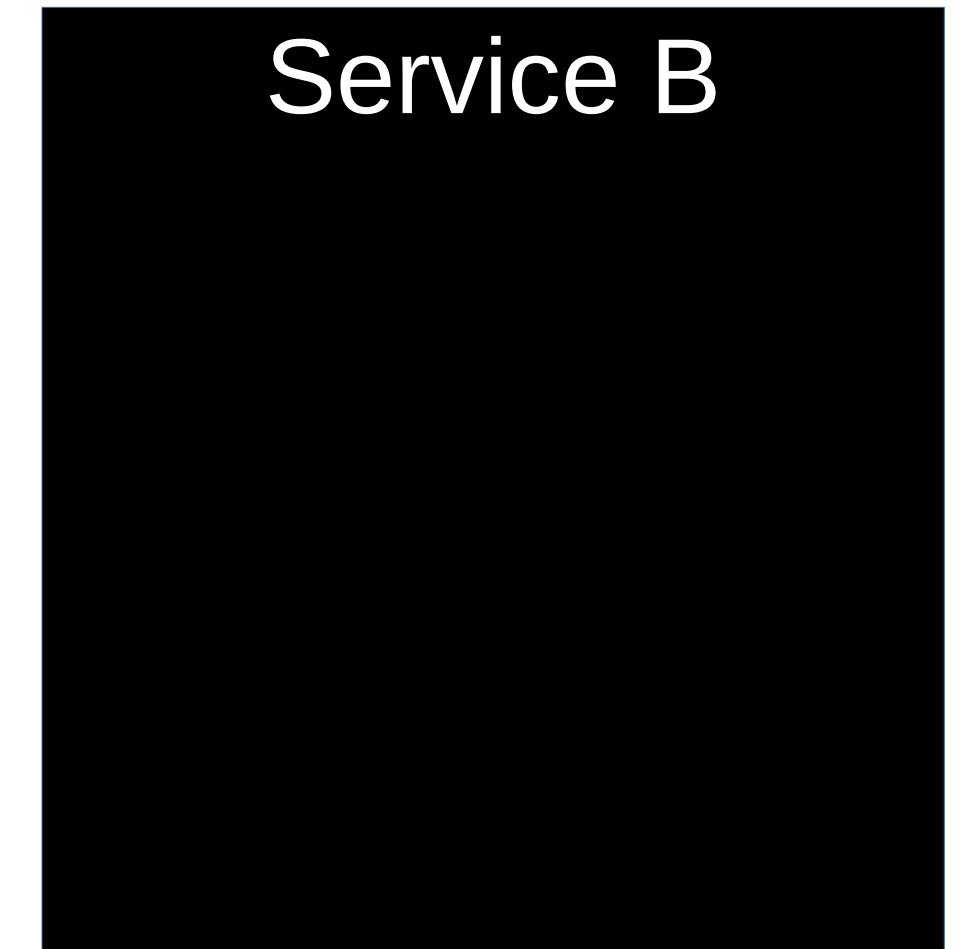
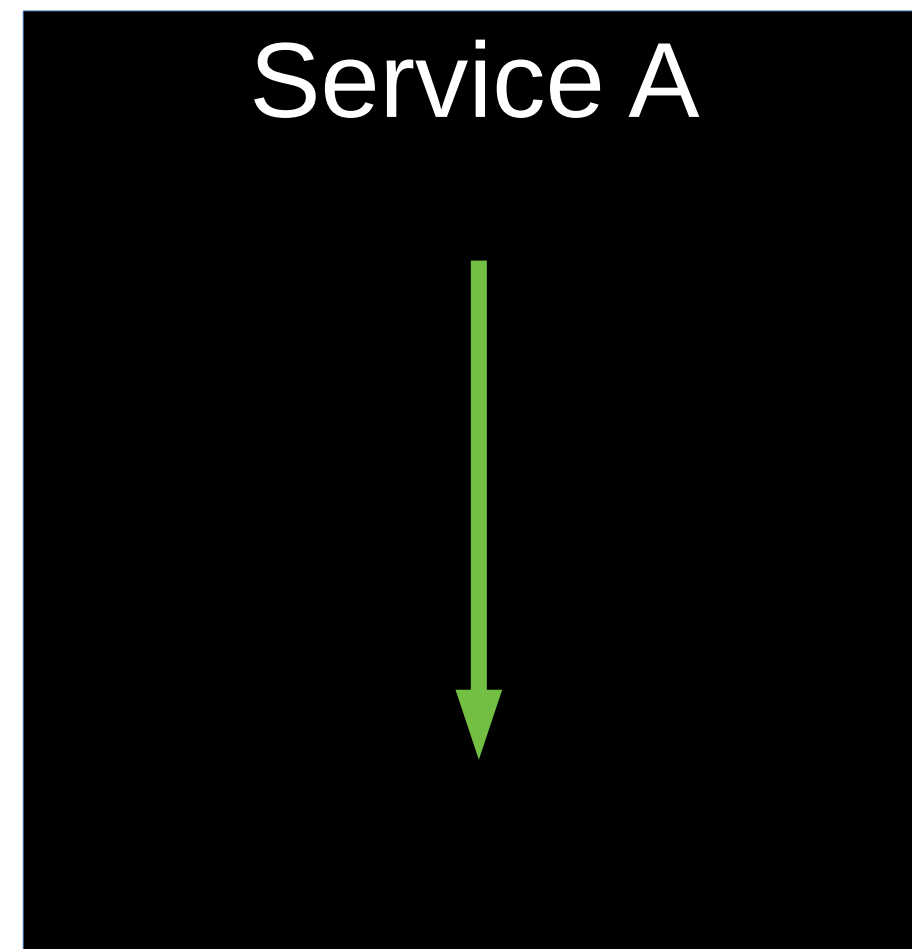
# Latency



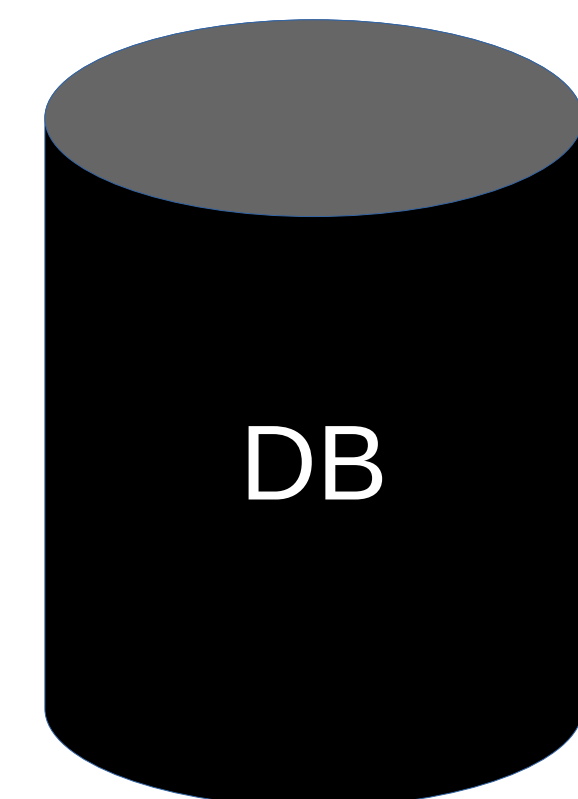
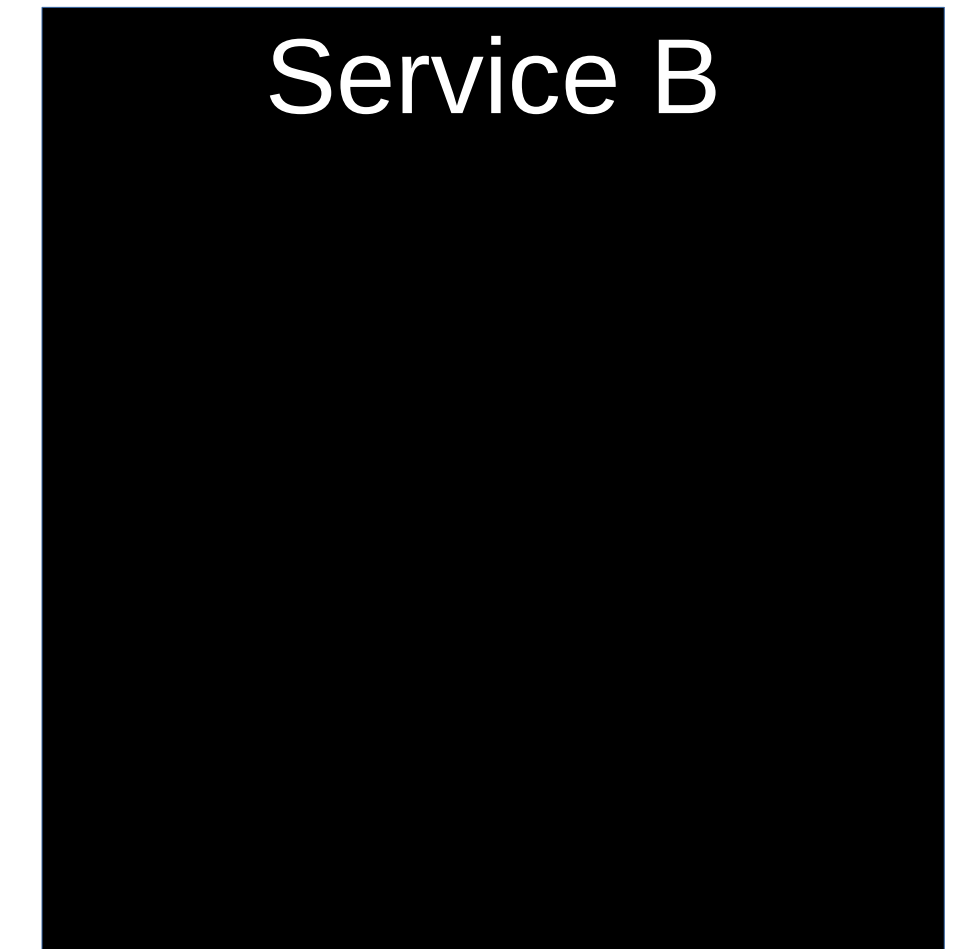
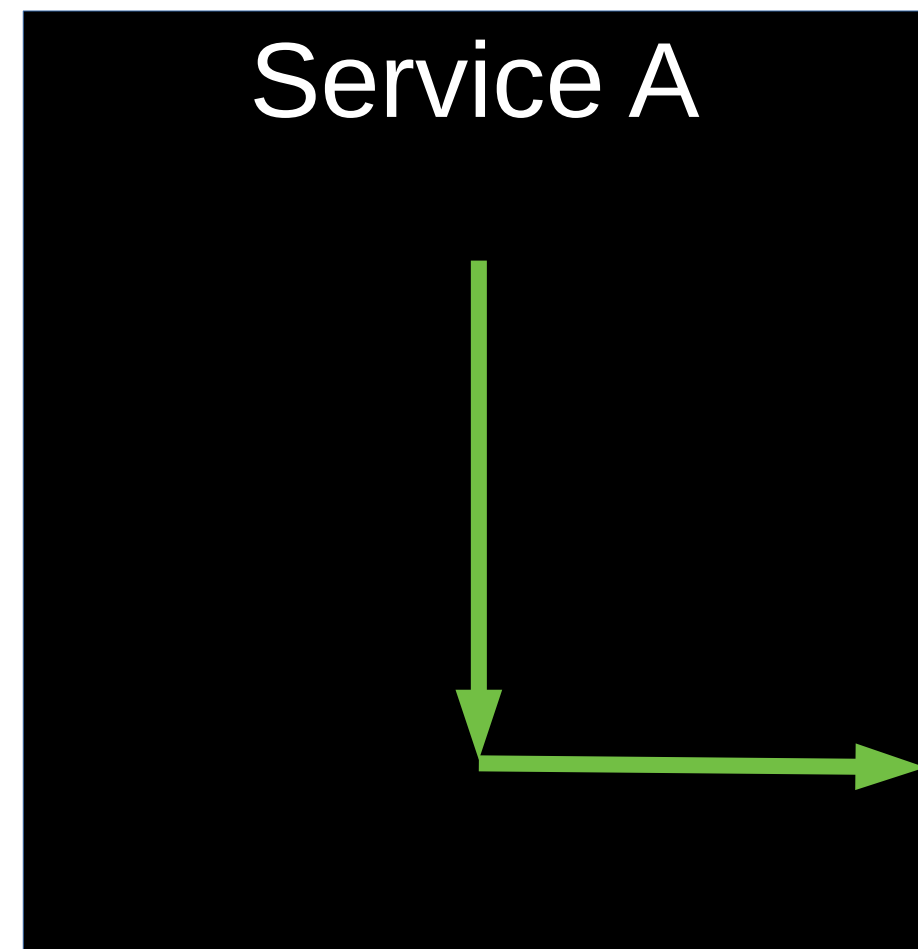
# A Request



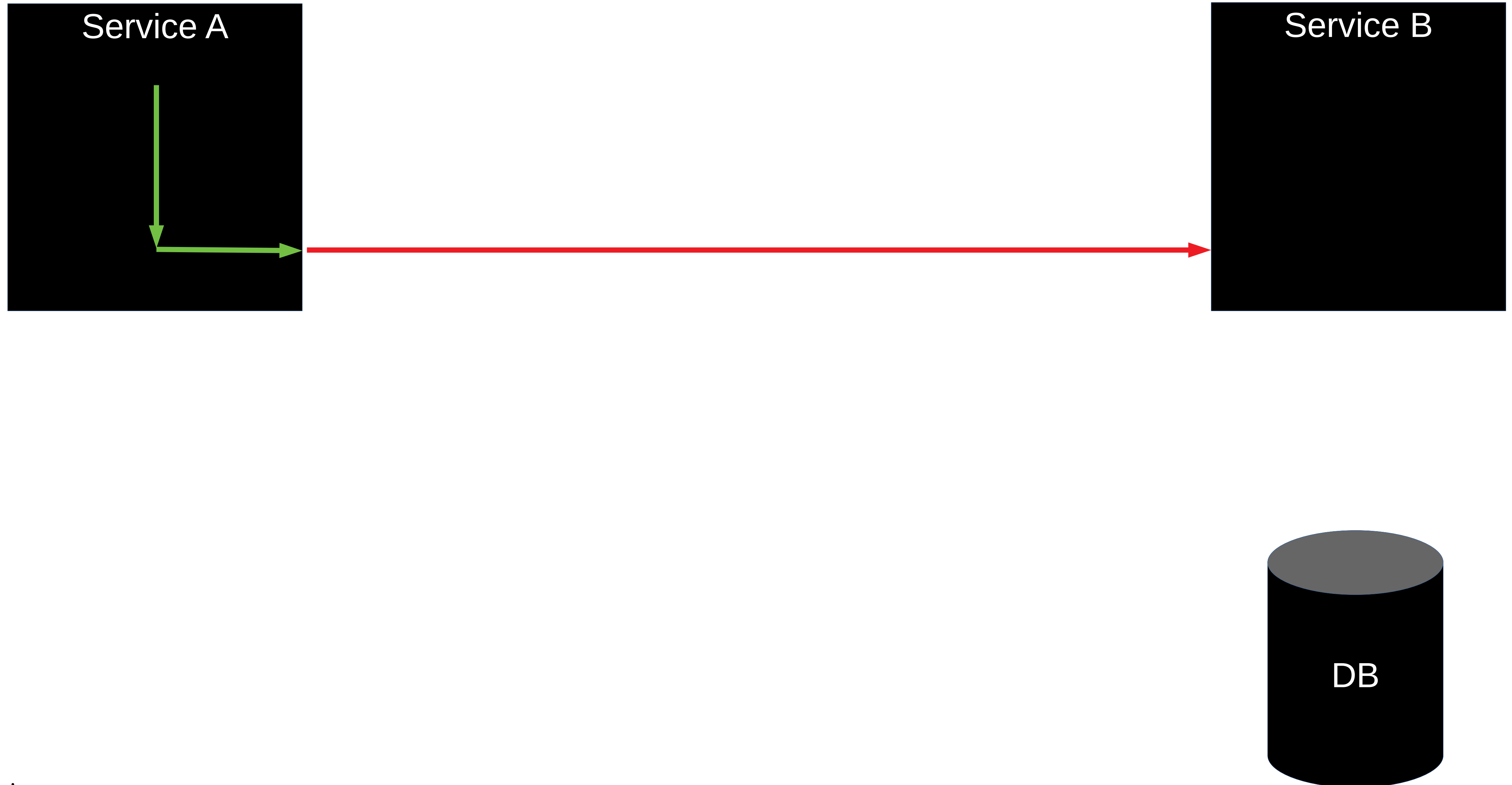
# A Request



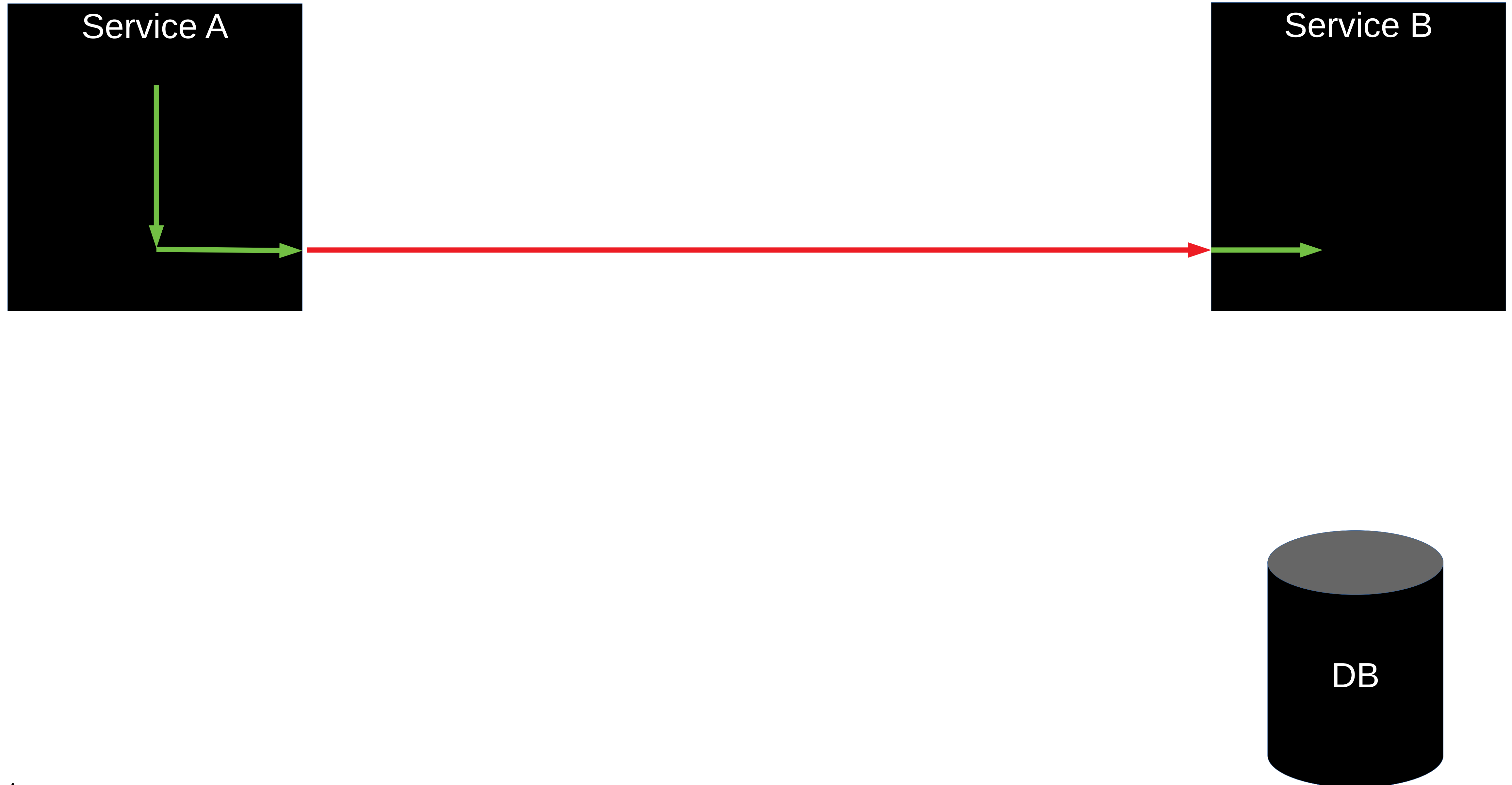
# A Request



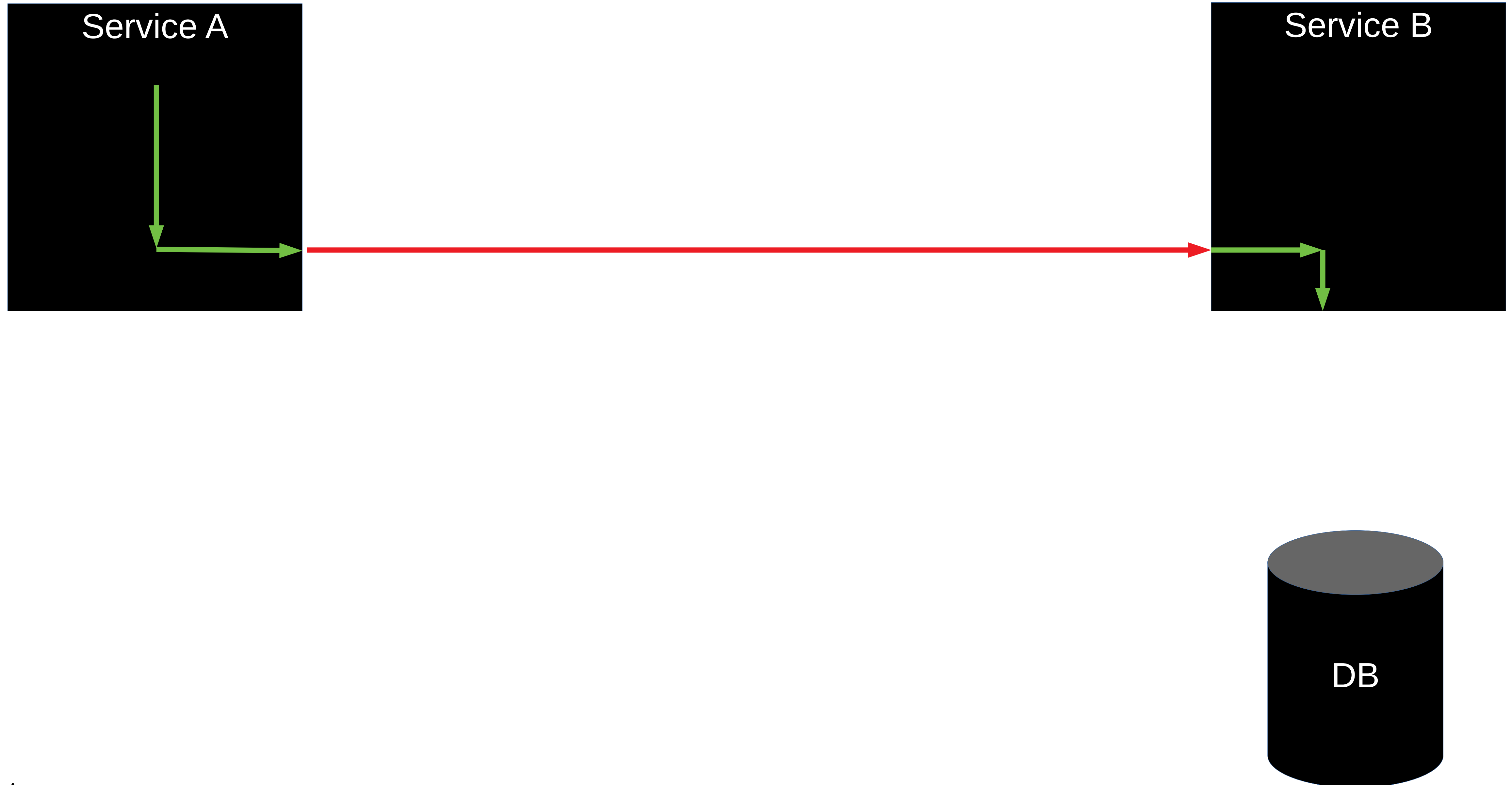
# A Request



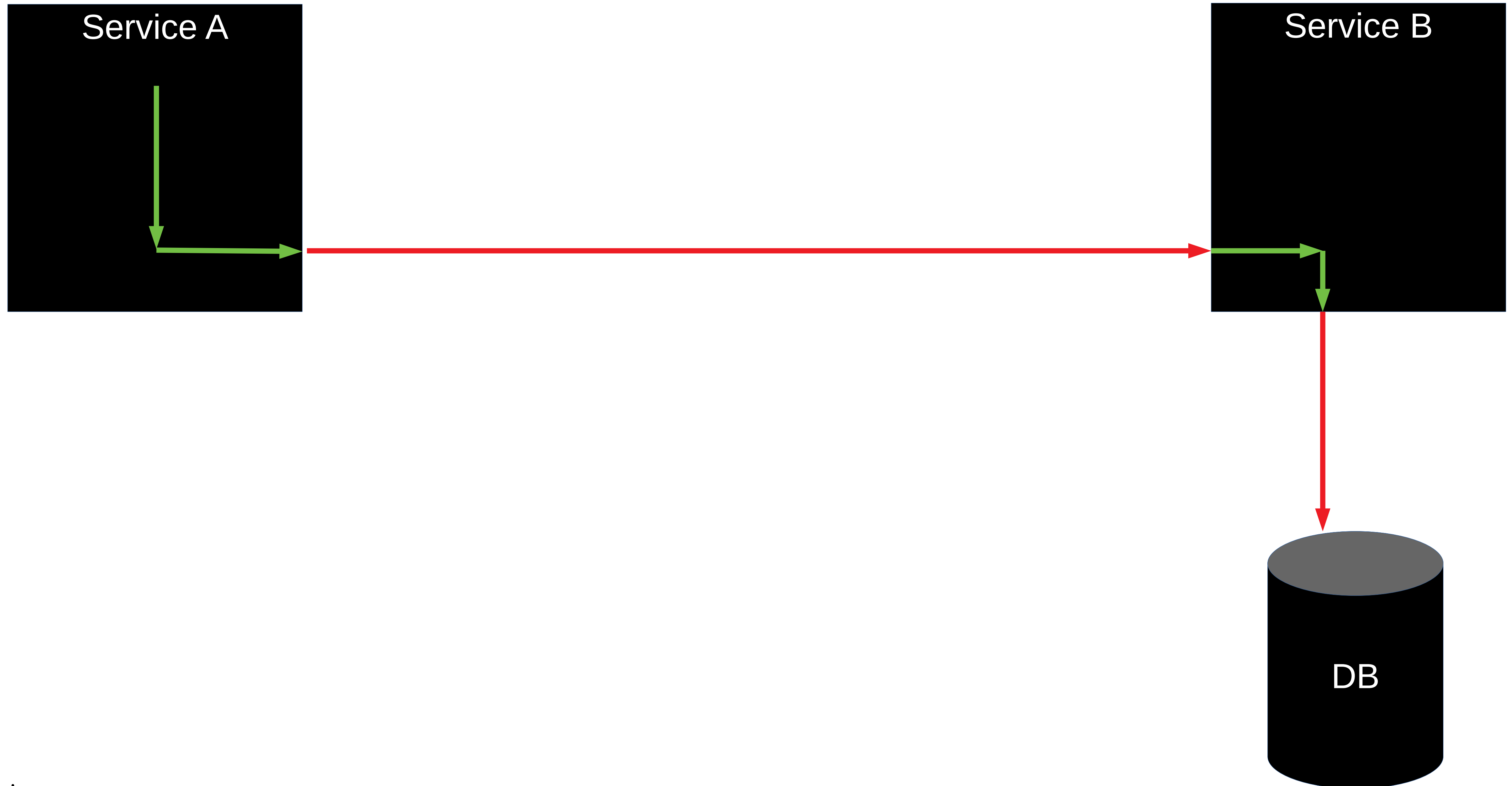
# A Request



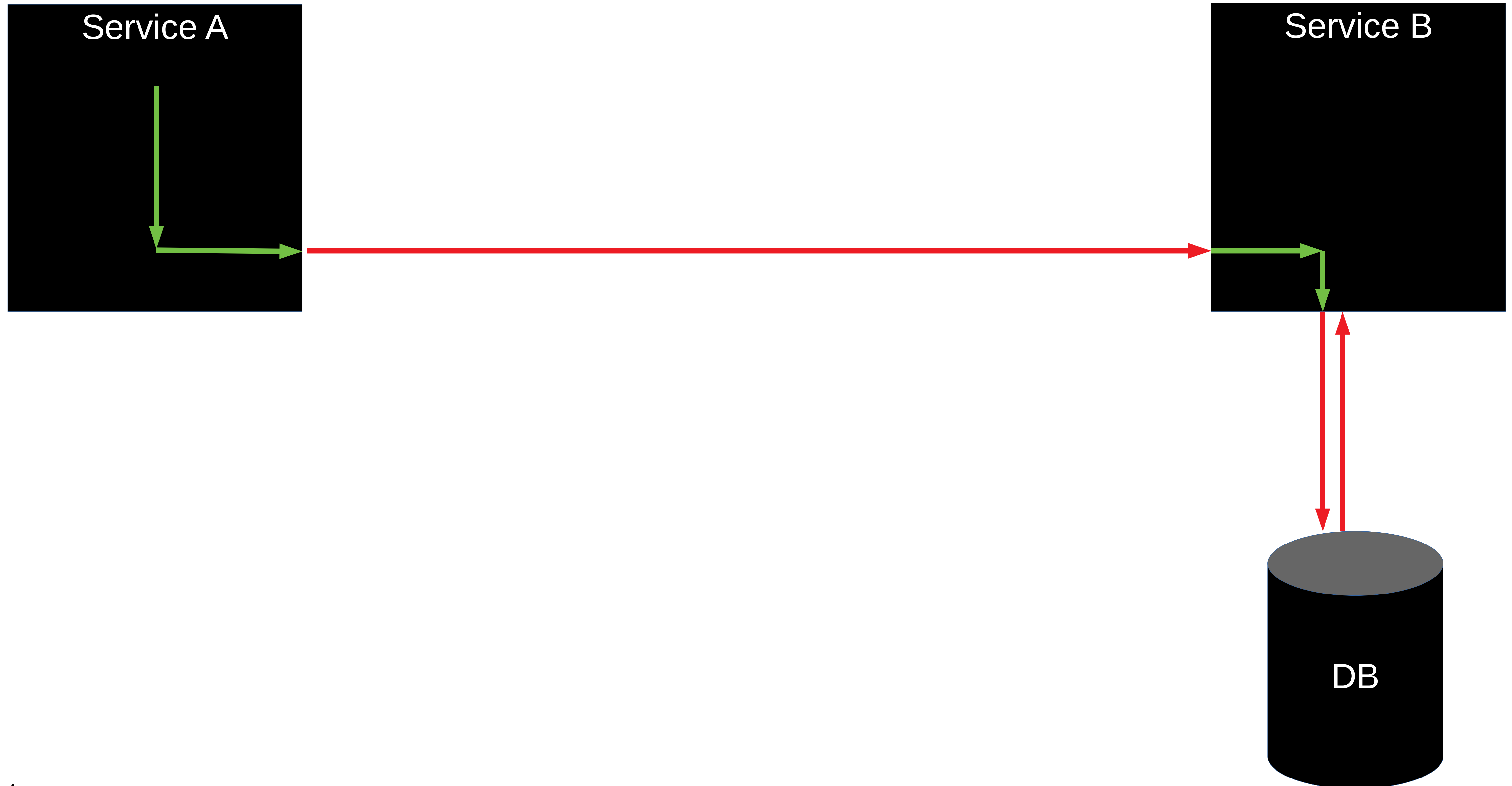
# A Request



# A Request

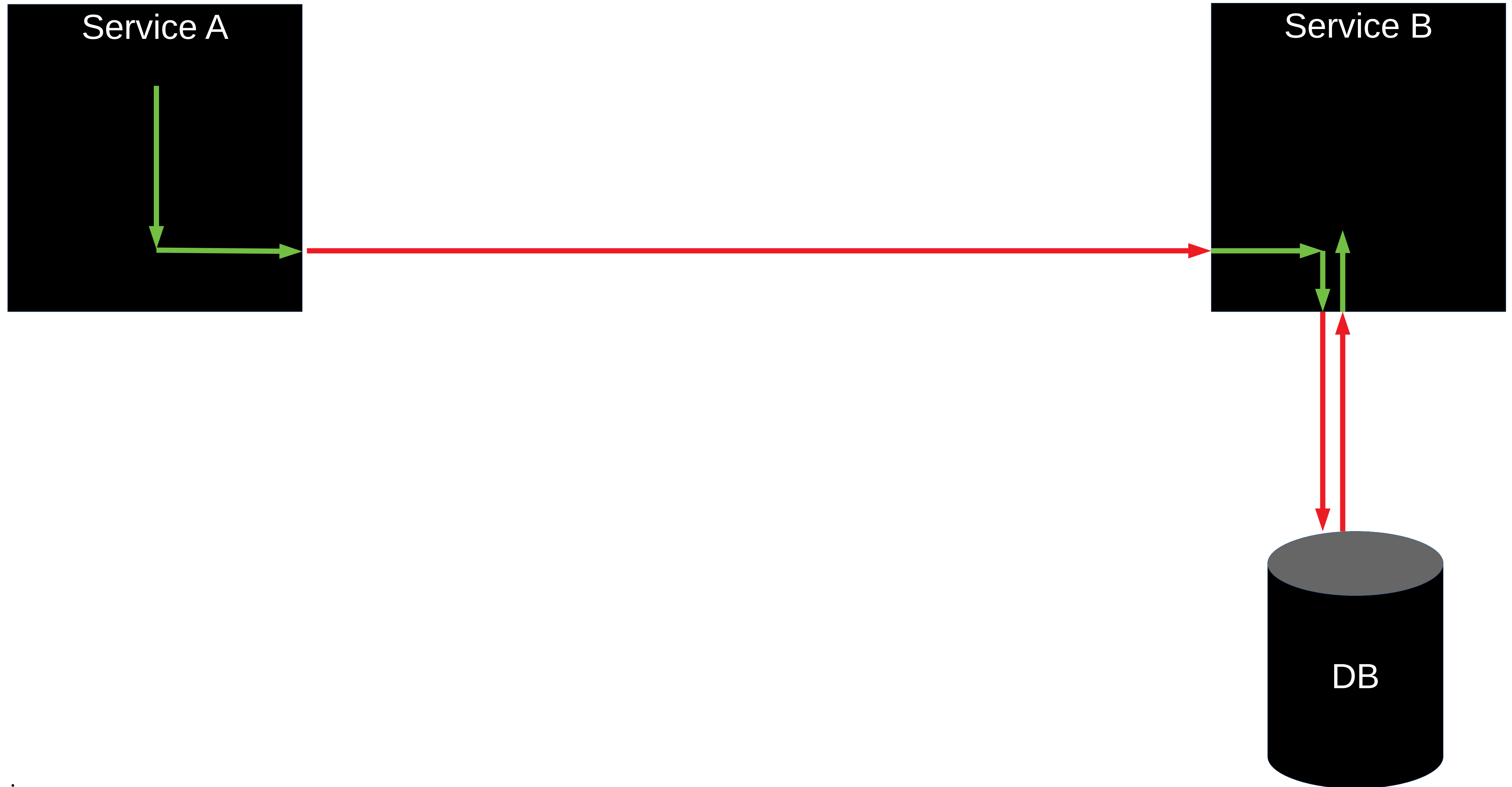


# A Request

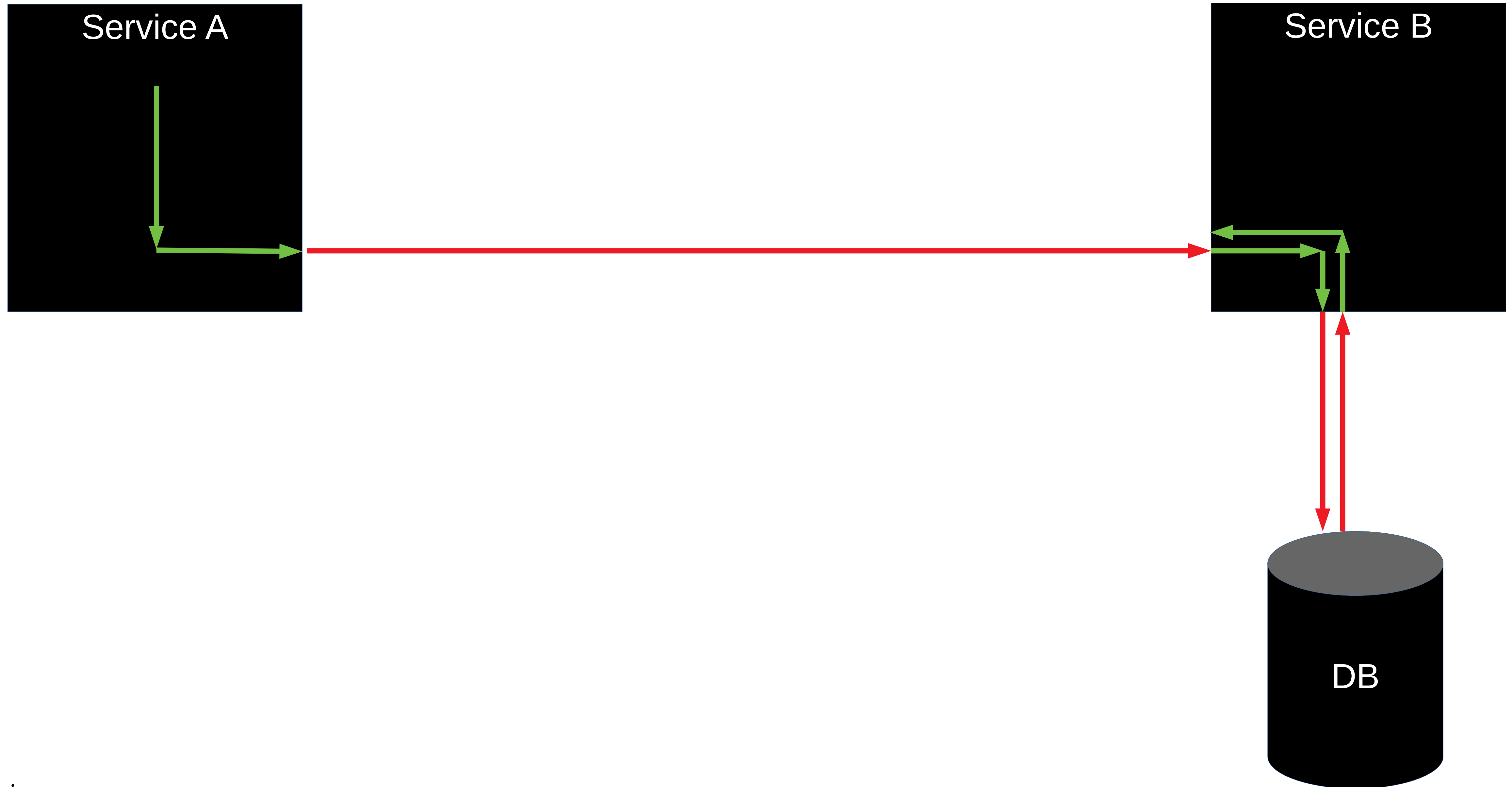




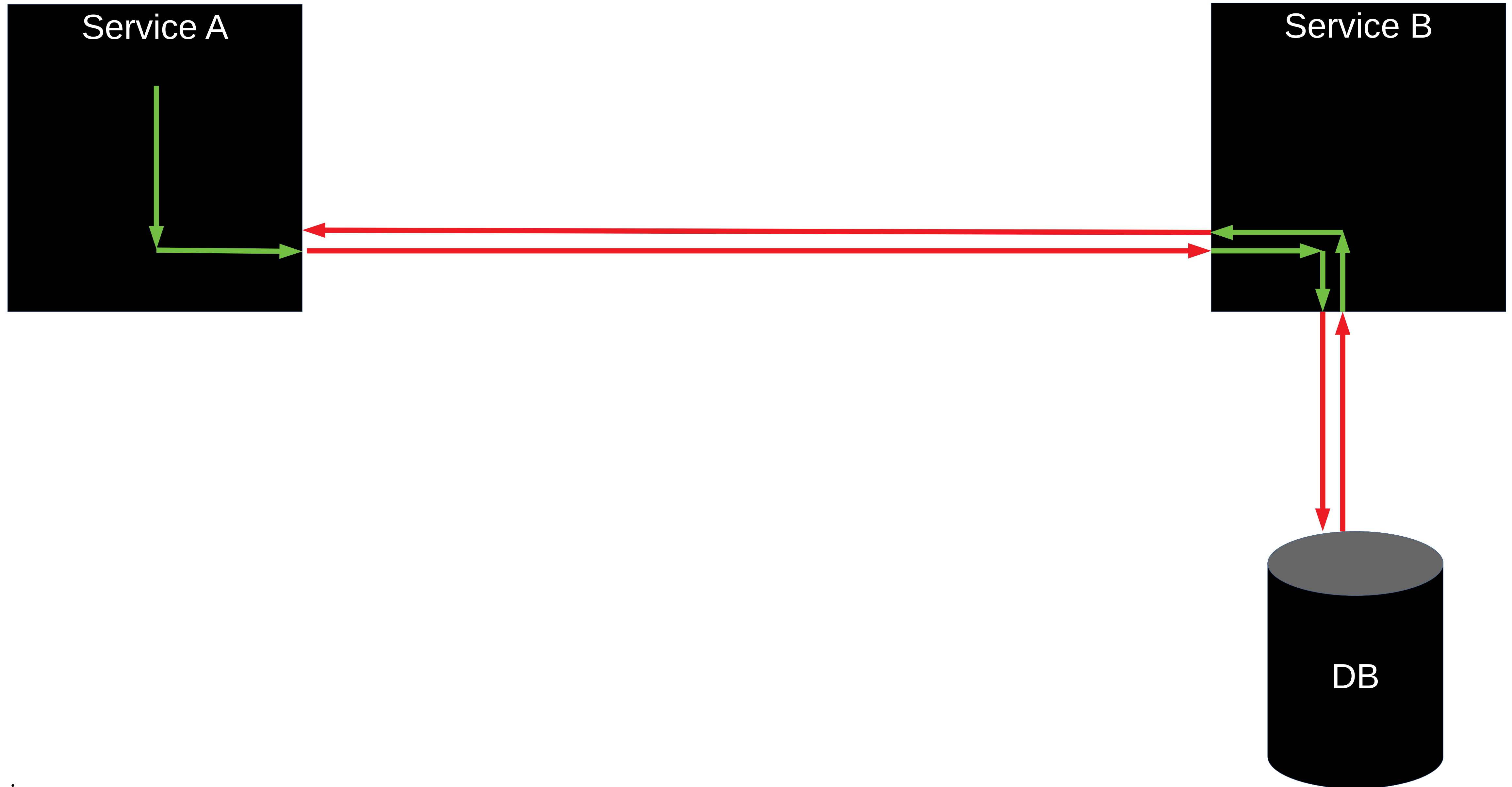
# A Request



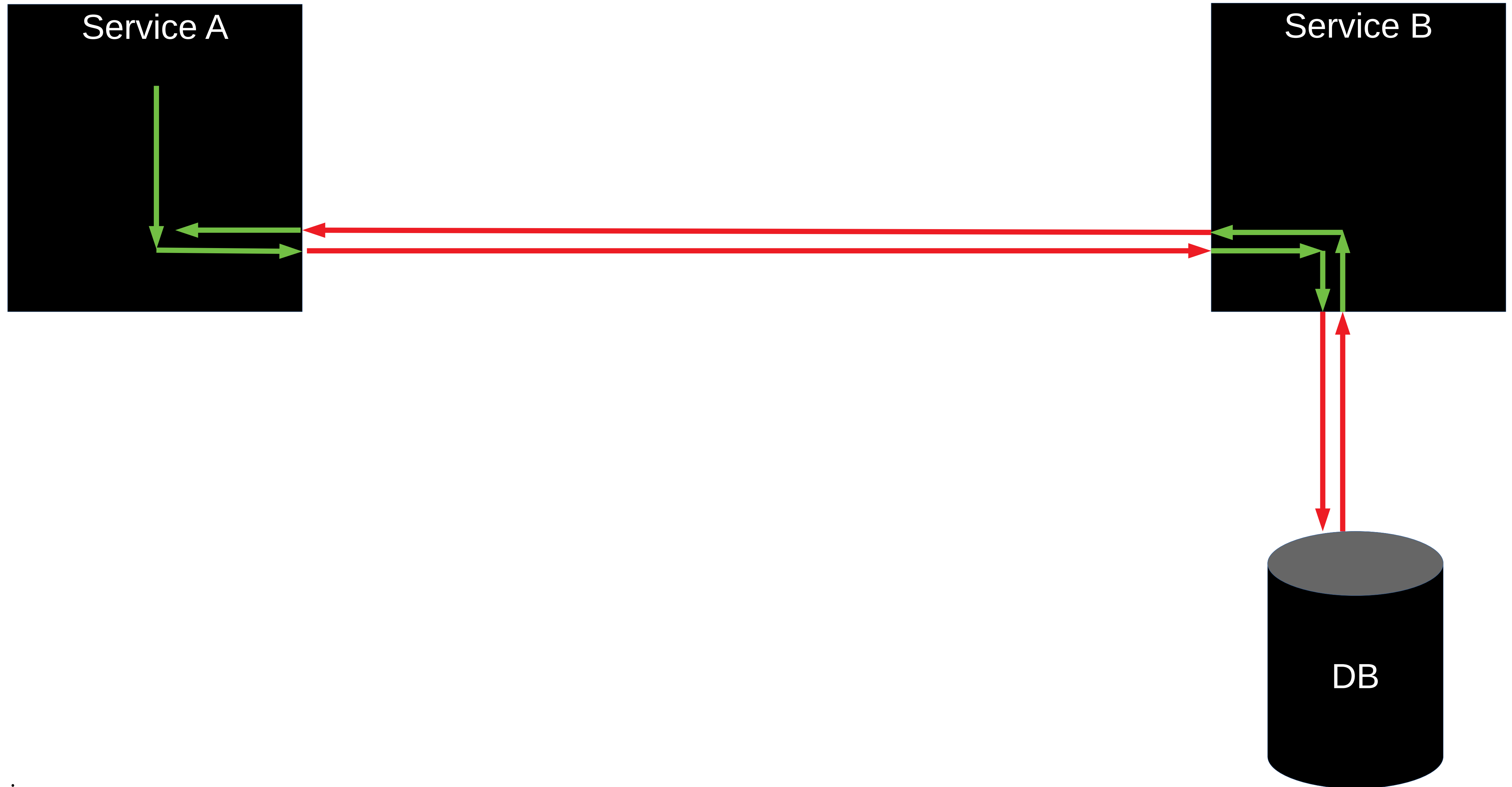
# A Request



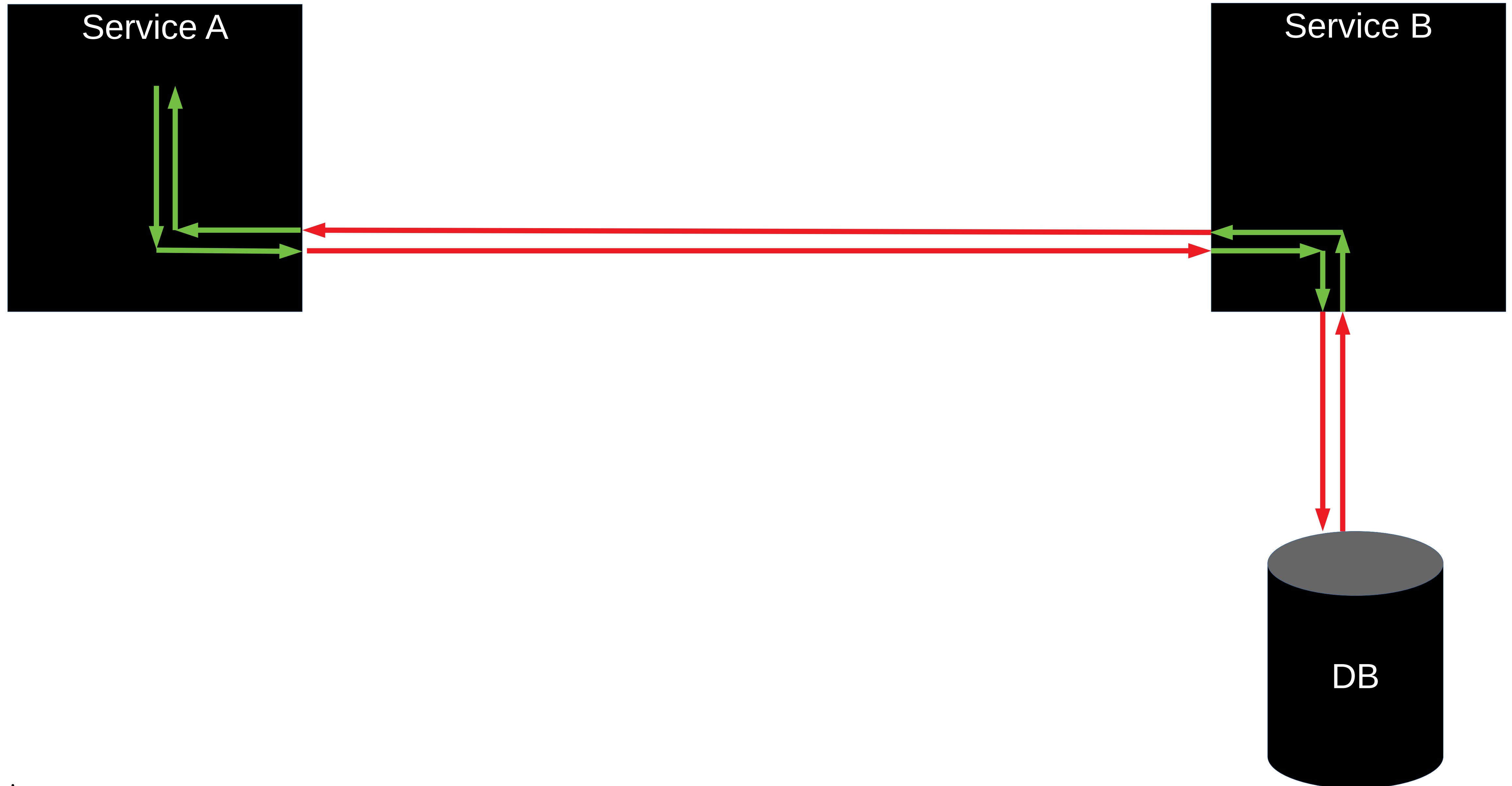
# A Request



# A Request

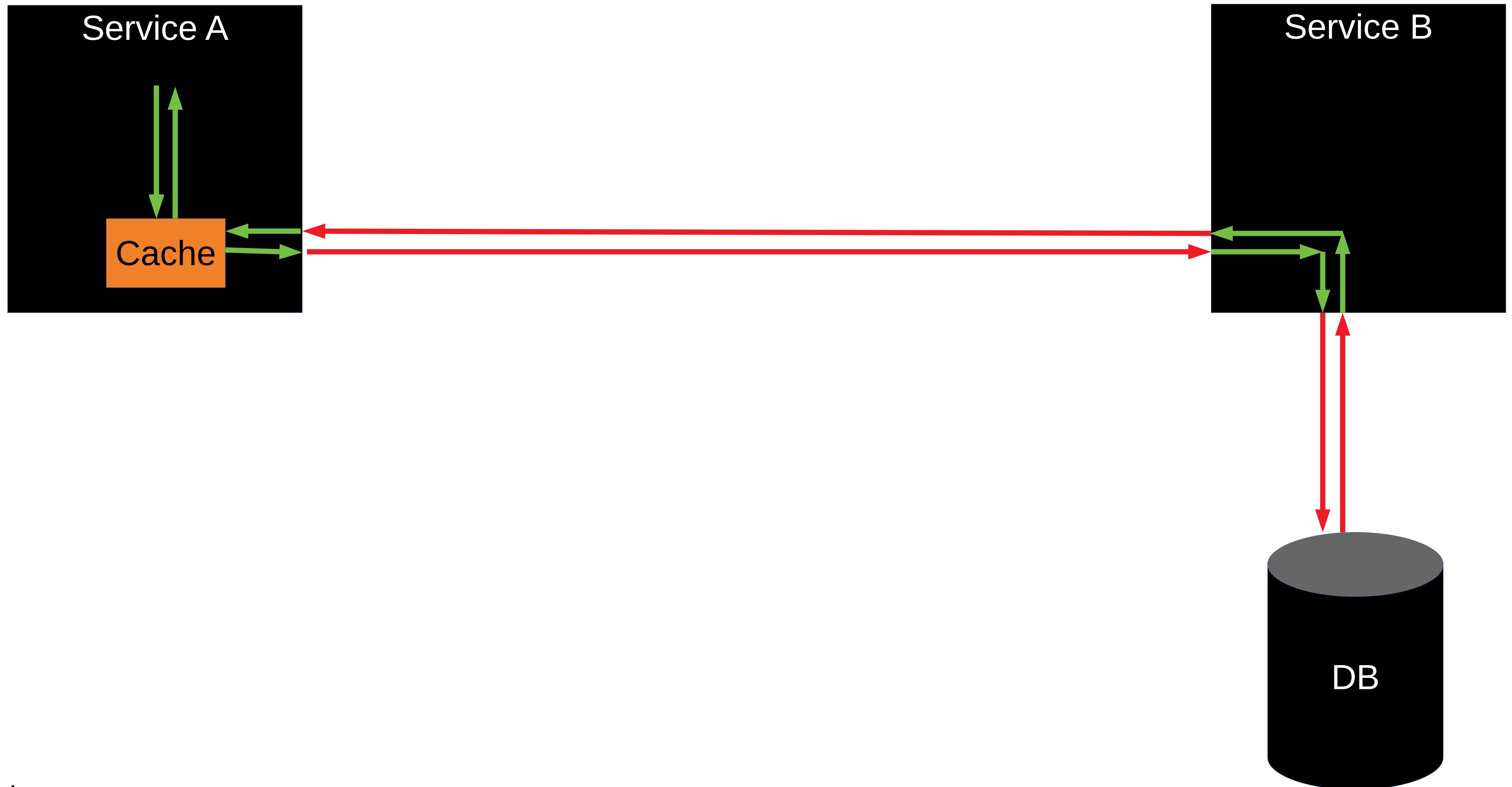


# A Request

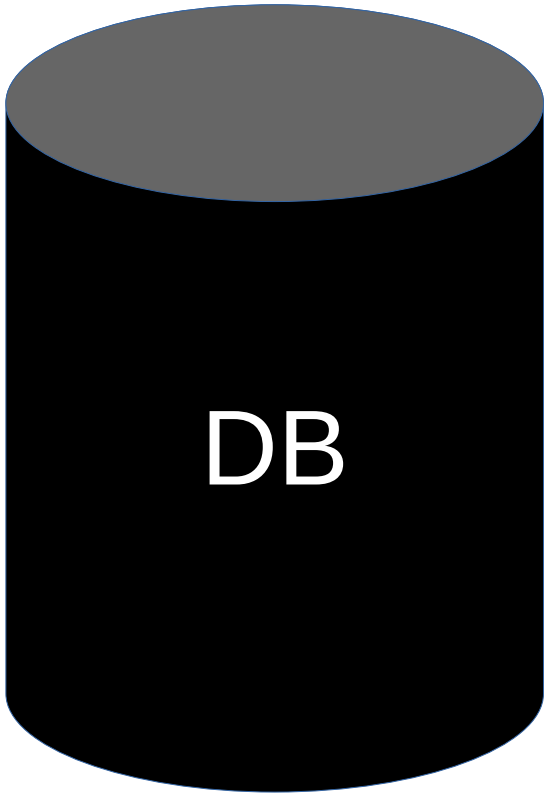
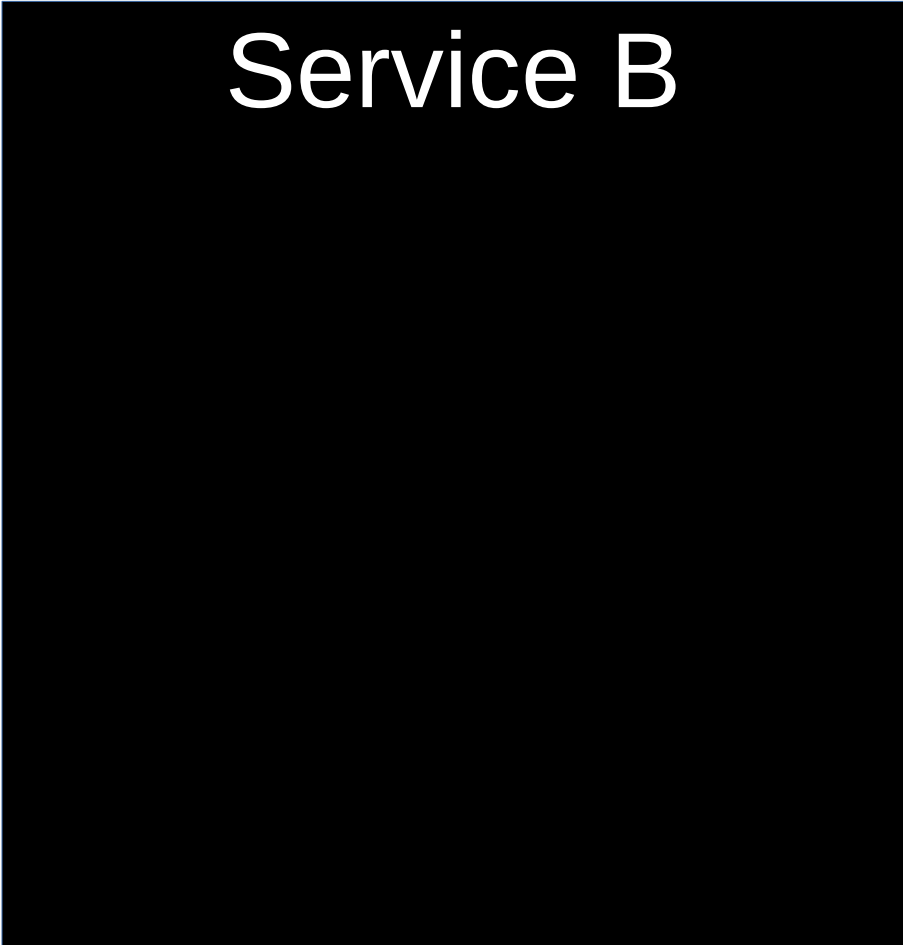
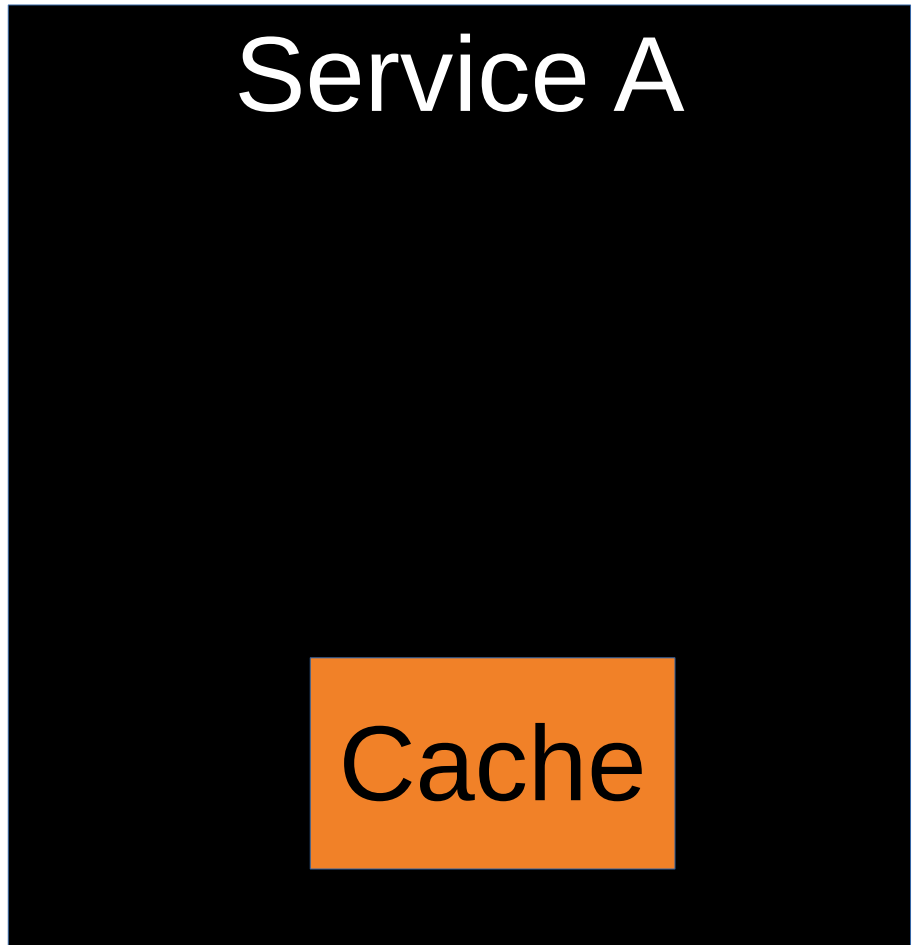


# ~~Latency~~ Caches

# Cache

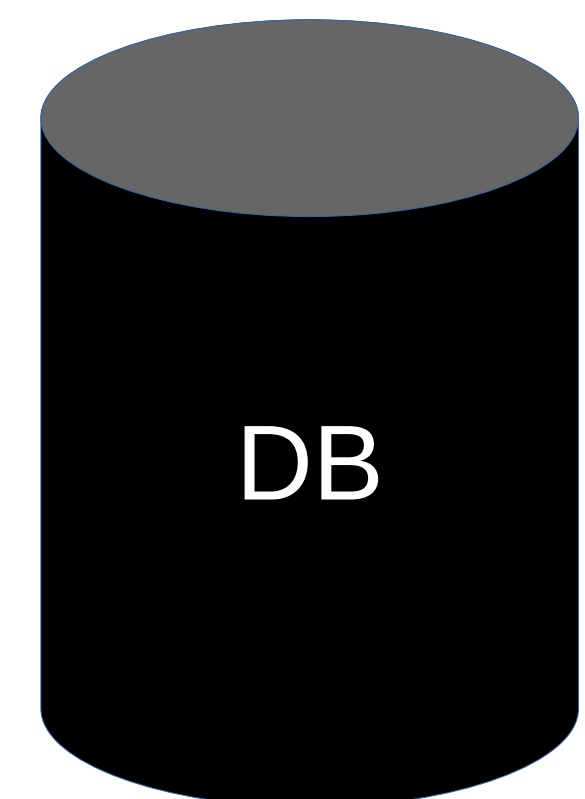
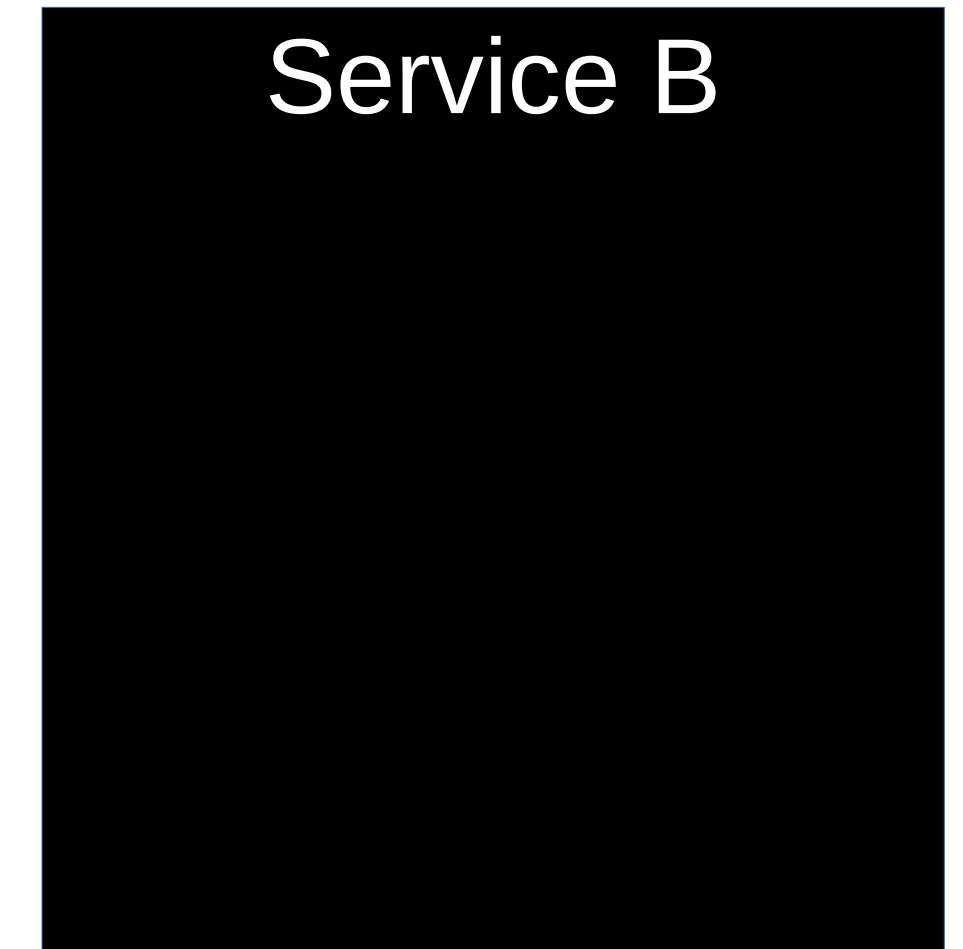
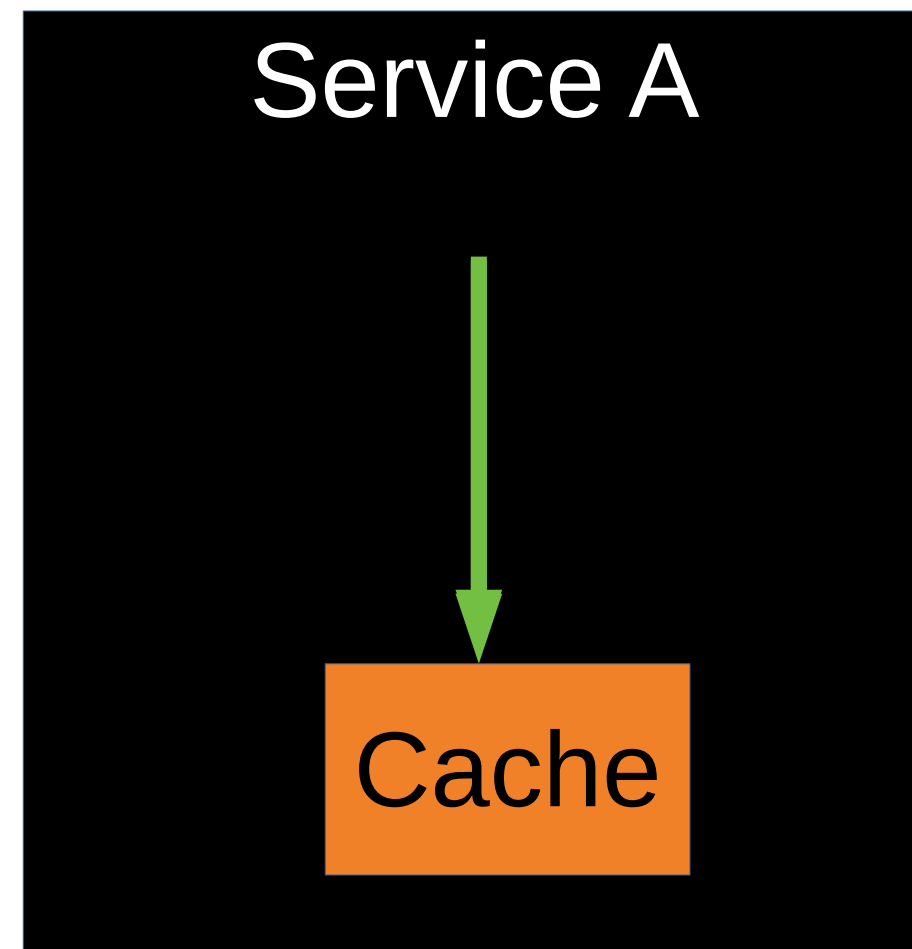


# Cache

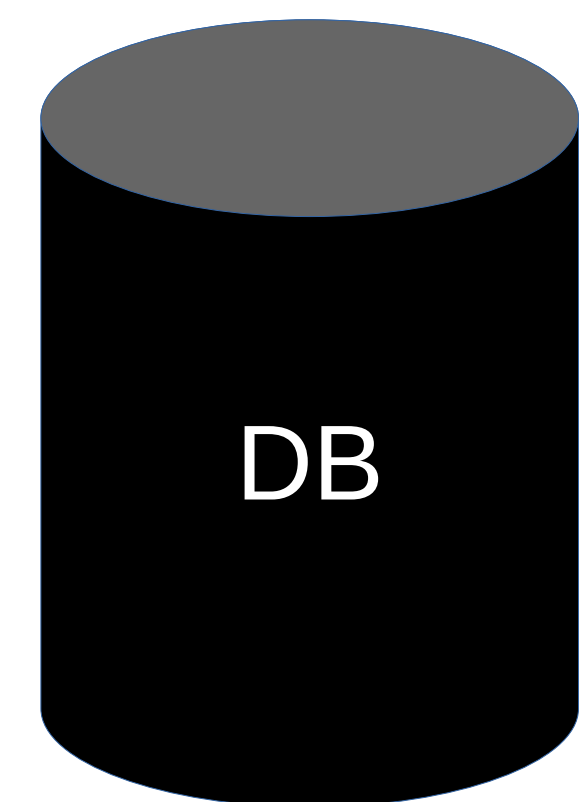
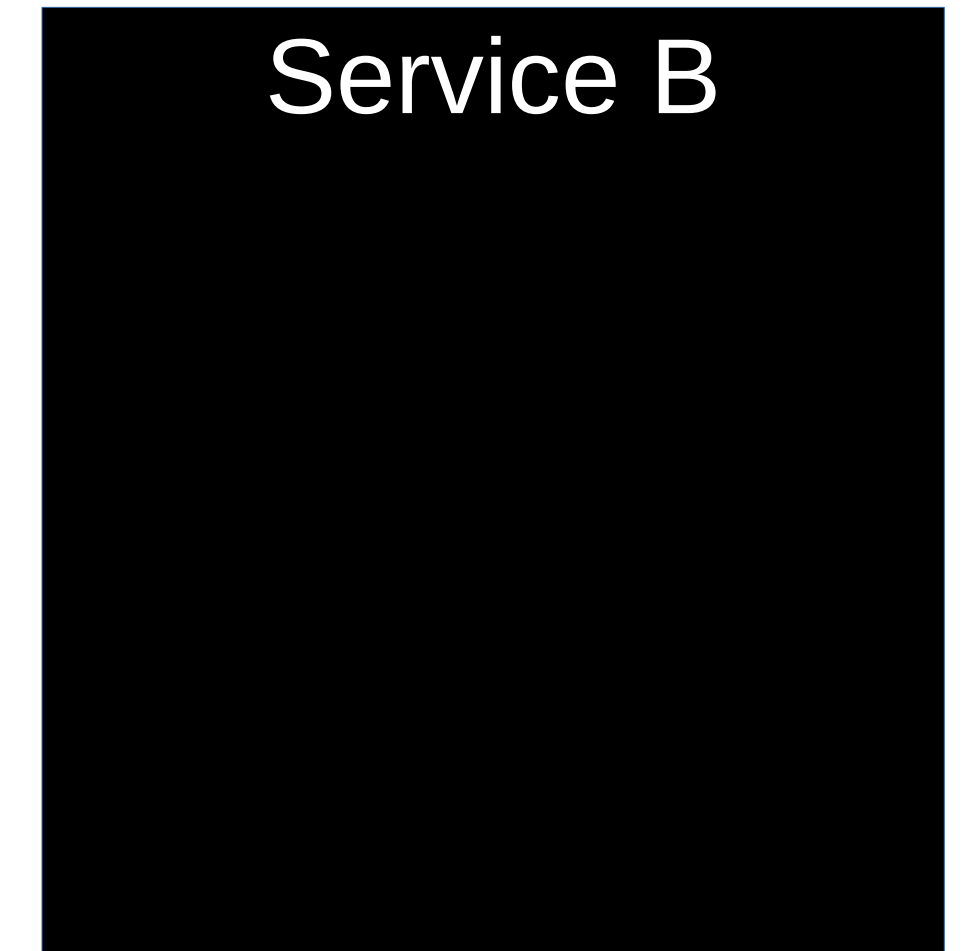
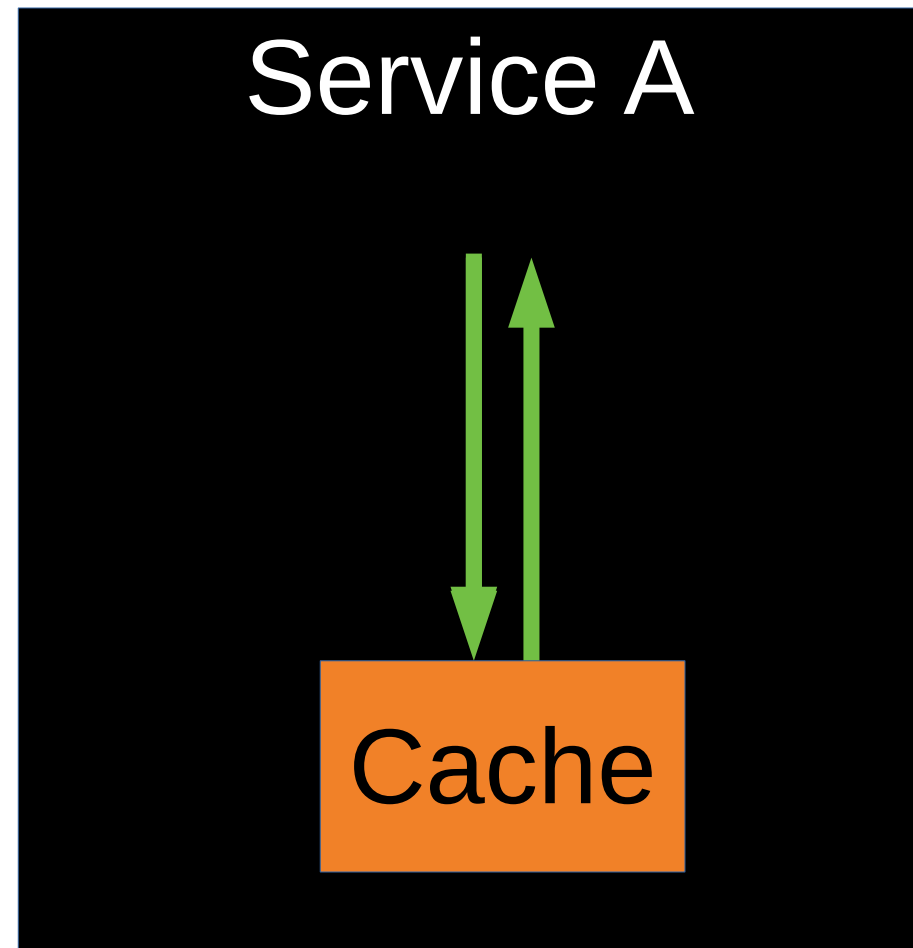




# Cache



# Cache



# All Sorts of Caches

# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`

# Postgre Cache Policy

```
struct AssortmentTraitCachePolicy {  
    static constexpr std::string_view kName = "assortment-trait-cache";  
  
    using ValueType = Assortment;  
    static constexpr auto kKeyMember = &Assortment::item_id;  
    static const storages::postgres::Query kQuery = "SELECT a, b, c FROM table";  
    static constexpr auto kUpdatedField = "update_time";  
    using UpdatedFieldType = storages::postgres::TimePointTz;  
};  
  
using AssortmentCache = components::PostgreCache<AssortmentTraitCachePolicy>;
```

# Postgre Cache Policy

```
struct AssortmentTraitCachePolicy {  
    static constexpr std::string_view kName = "assortment-trait-cache";  
  
    using ValueType = Assortment;  
    static constexpr auto kKeyMember = &Assortment::item_id;  
    static const storages::postgres::Query kQuery = "SELECT a, b, c FROM table";  
    static constexpr auto kUpdatedField = "update_time";  
    using UpdatedFieldType = storages::postgres::TimePointTz;  
};  
  
using AssortmentCache = components::PostgreCache<AssortmentTraitCachePolicy>;
```

# Postgre Cache Policy

```
struct AssortmentTraitCachePolicy {  
    static constexpr std::string_view kName = "assortment-trait-cache";  
  
    using ValueType = Assortment;  
    static constexpr auto kKeyMember = &Assortment::item_id;  
    static const storages::postgres::Query kQuery = "SELECT a, b, c FROM table";  
    static constexpr auto kUpdatedField = "update_time";  
    using UpdatedFieldType = storages::postgres::TimePointTz;  
};  
  
using AssortmentCache = components::PostgreCache<AssortmentTraitCachePolicy>;
```

# Postgre Cache Policy

```
struct AssortmentTraitCachePolicy {  
    static constexpr std::string_view kName = "assortment-trait-cache";  
  
    using ValueType = Assortment;  
    static constexpr auto kKeyMember = &Assortment::item_id;  
    static const storages::postgres::Query kQuery = "SELECT a, b, c FROM table";  
    static constexpr auto kUpdatedField = "update_time";  
    using UpdatedFieldType = storages::postgres::TimePointTz;  
};  
  
using AssortmentCache = components::PostgreCache<AssortmentTraitCachePolicy>;
```



# Postgre Cache Policy

```
struct AssortmentTraitCachePolicy {  
    static constexpr std::string_view kName = "assortment-trait-cache";  
  
    using ValueType = Assortment;  
    static constexpr auto kKeyMember = &Assortment::item_id;  
    static const storages::postgres::Query kQuery = "SELECT a, b, c FROM table";  
    static constexpr auto kUpdatedField = "update_time";  
    using UpdatedFieldType = storages::postgres::TimePointTz;  
};  
  
using AssortmentCache = components::PostgreCache<AssortmentTraitCachePolicy>;
```

# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`

# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`
- `components::MongoCache< MongoCacheTraits >`

# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`
- `components::MongoCache< MongoCacheTraits >`
- `components::CachingComponentBase`

# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`
- `components::MongoCache< MongoCacheTraits >`
- `components::CachingComponentBase`
  
- LRU

# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`
- `components::MongoCache< MongoCacheTraits >`
- `components::CachingComponentBase`
- LRU:
  - `cache::LruCacheComponent< Key, Value, Hash, Equal >`

# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`
- `components::MongoCache< MongoCacheTraits >`
- `components::CachingComponentBase`
- LRU:
  - `cache::LruCacheComponent< Key, Value, Hash, Equal >`
  - `cache::ExpirableLruCache< Key, Value, Hash, Equal >`

# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`
- `components::MongoCache< MongoCacheTraits >`
- `components::CachingComponentBase`
- LRU:
  - `cache::LruCacheComponent< Key, Value, Hash, Equal >`
  - `cache::ExpirableLruCache< Key, Value, Hash, Equal >`
- Containers



# All Sorts of Caches

- `components::PostgreCache< PostgreCachePolicy >`
- `components::MongoCache< MongoCacheTraits >`
- `components::CachingComponentBase`
- LRU:
  - `cache::LruCacheComponent< Key, Value, Hash, Equal >`
  - `cache::ExpirableLruCache< Key, Value, Hash, Equal >`
- Containers:
  - `cache::NWayLRU< T, U, Hash, Equal >`
  - `cache::LruMap< T, U, Hash, Equal >`
  - `cache::LruSet< T, Hash, Equal >`

# C++ scares

C++ ~~scars~~ aids

# Feedback

# Feedback

«I'm a C# developer. I had no idea that coding in C++ is easy»

# Feedback

«I'm a C# developer. I had no idea that coding in C++ is easy»

«No matter how unbelievable it sounds...

# Feedback

«I'm a C# developer. I had no idea that coding in C++ is easy»

«No matter how unbelievable it sounds...  
...it really doesn't take much longer to write services on **uServer** than on Python»

# RAII



# C++ Hello World

```
char* get1();  
char* get2();  
void do_something(const char* s);
```



# C++ Hello World

```
char* get1();  
char* get2();  
void do_something(const char* s);  
  
char* str_plus(const char* s1, const char* s2) {  
    unsigned len = strlen(s1) + strlen(s2) + 1;  
  
}
```

# C++ Hello World

```
char* get1();  
char* get2();  
void do_something(const char* s);  
  
char* str_plus(const char* s1, const char* s2) {  
    unsigned len = strlen(s1) + strlen(s2) + 1;  
  
}
```

# C++ Hello World

```
char* get1();  
char* get2();  
void do_something(const char* s);  
  
char* str_plus(const char* s1, const char* s2) {  
    unsigned len = strlen(s1) + strlen(s2) + 1;  
  
}
```

# C++ Hello World

```
char* get1();  
char* get2();  
void do_something(const char* s);  
  
char* str_plus(const char* s1, const char* s2) {  
    unsigned len = strlen(s1) + strlen(s2) + 1;  
  
}
```

# C++ Hello World

```
char* get1();  
char* get2();  
void do_something(const char* s);  
  
char* str_plus(const char* s1, const char* s2) {  
    unsigned len = strlen(s1) + strlen(s2) + 1;  
    char* result = (char*)malloc(len);  
  
}
```

# C++ Hello World

```
char* get1();  
char* get2();  
void do_something(const char* s);  
  
char* str_plus(const char* s1, const char* s2) {  
    unsigned len = strlen(s1) + strlen(s2) + 1;  
    char* result = (char*)malloc(len);  
    strcat(result, s1);  
    strcat(result, s2);  
  
}
```



# C++ Hello World

```
char* get1();  
char* get2();  
void do_something(const char* s);  
  
char* str_plus(const char* s1, const char* s2) {  
    unsigned len = strlen(s1) + strlen(s2) + 1;  
    char* result = (char*)malloc(len);  
    strcat(result, s1);  
    strcat(result, s2);  
    return result;  
}
```

# C++ Hello World

```
char* get1();
char* get2();
void do_something(const char* s);

char* str_plus(const char* s1, const char* s2) {
    unsigned len = strlen(s1) + strlen(s2) + 1;
    char* result = (char*)malloc(len);
    strcat(result, s1);
    strcat(result, s2);
    return result;
}

void example1() {

}
```

# C++ Hello World

```
char* get1();
char* get2();
void do_something(const char* s);

char* str_plus(const char* s1, const char* s2) {
    unsigned len = strlen(s1) + strlen(s2) + 1;
    char* result = (char*)malloc(len);
    strcat(result, s1);
    strcat(result, s2);
    return result;
}

void example1() {
    char *s1 = get1(), *s2 = get2();

}
```

# C++ Hello World

```
char* get1();
char* get2();
void do_something(const char* s);

char* str_plus(const char* s1, const char* s2) {
    unsigned len = strlen(s1) + strlen(s2) + 1;
    char* result = (char*)malloc(len);
    strcat(result, s1);
    strcat(result, s2);
    return result;
}

void example1() {
    char *s1 = get1(), *s2 = get2();
    char* result = str_plus(s1, s2);

}
```

# C++ Hello World

```
char* get1();
char* get2();
void do_something(const char* s);

char* str_plus(const char* s1, const char* s2) {
    unsigned len = strlen(s1) + strlen(s2) + 1;
    char* result = (char*)malloc(len);
    strcat(result, s1);
    strcat(result, s2);
    return result;
}

void example1() {
    char *s1 = get1(), *s2 = get2();
    char* result = str_plus(s1, s2);
    do_something(result);
}
```

# C++ Hello World

```
char* get1();
char* get2();
void do_something(const char* s);

char* str_plus(const char* s1, const char* s2) {
    unsigned len = strlen(s1) + strlen(s2) + 1;
    char* result = (char*)malloc(len);
    strcat(result, s1);
    strcat(result, s2);
    return result;
}

void example1() {
    char *s1 = get1(), *s2 = get2();
    char* result = str_plus(s1, s2);
    do_something(result);
    free(result);
}
```

# C++ Hello World

```
char* get1();
char* get2();
void do_something(const char* s);

char* str_plus(const char* s1, const char* s2) {
    unsigned len = strlen(s1) + strlen(s2) + 1;
    char* result = (char*)malloc(len);
    strcat(result, s1);
    strcat(result, s2);
    return result;
}

void example1() {
    char *s1 = get1(), *s2 = get2();
    char* result = str_plus(s1, s2);
    do_something(result);
    free(result);
    // free(s1); ???
    // free(s2); ???
}
```

That was not a C++ code!!!



Here's a C++ code:

# C++ Hello World

```
std::string get_str1();  
std::string get_str2();  
void do_something(const char* s);  
  
void example2() {  
    auto result = get_str1() + get_str2();  
    do_something(result.c_str());  
}
```

# C++ Hello World

```
std::string get_str1();  
std::string get_str2();  
void do_something(const char* s);  
  
void example2() {  
    auto result = get_str1() + get_str2();  
    do_something(result.c_str());  
}
```

# C++ Hello World

```
std::string get_str1();  
std::string get_str2();  
void do_something(const char* s);  
  
void example2() {  
    auto result = get_str1() + get_str2();  
    do_something(result.c_str());  
}
```

# RAII

# RAII, -Wall

# RAII, -Wall, sanitizers

RAII, -Wall, sanitizers, clang-tidy



RAII, -Wall, sanitizers, clang-tidy,  
asserts

# Compile Time is your friend!

# Catching Bugs at Compile Time

# Catching Bugs at Compile Time

```
const auto& name = cache.Get()->name;
```

# Catching Bugs at Compile Time

```
const auto& name = cache.Get()->name;  
DoSomething(name);
```

# Catching Bugs at Compile Time

```
const auto& name = cache.Get()->name;  
DoSomething(name);
```

```
// const auto& name = cache.Get()->name;  
//           ~~~~~^  
// error: keep the pointer before using, please
```

# SharedReadablePtr

```
utils::SharedReadablePtr< T > Get () const
```

```
utils::SharedReadablePtr< T > GetUnsafe () const
```

# SharedReadablePtr

```
const T & operator* () const &noexcept
```

```
const T & operator* () &&
```

```
const T * operator-> () const &noexcept
```

```
const T * operator-> () &&
```



# Catching Bugs at Compile Time

```
const auto& name = cache.Get()->name;  
DoSomething(name);
```

```
// const auto& name = cache.Get()->name;  
//               ~~~~~^  
// error: keep the pointer before using, please
```

# Catching Bugs at Compile Time

```
const auto& name = cache.Get()->name;  
DoSomething(name);
```

```
// const auto& name = cache.Get()->name;  
//                ~~~~~^  
// error: keep the pointer before using, please
```

# static\_assert

# static\_assert everything

```
template <typename T>
T Value::As() const {

    static_assert(formats::common::kHasParseTo<Value, T>,
        "There is no `Parse(const Value&, formats::parse::To<T>)` "
        "in namespace of `T` or `formats::parse`. "
        "Probably you forgot to include the "
        "<formats/parse/common_containers.hpp> or you "
        "have not provided a `Parse` function overload.");

    return Parse(*this, formats::parse::To<T>{});
}
```

# static\_assert everything

```
template <typename T>
T Value::As() const {

    static_assert(formats::common::kHasParseTo<Value, T>,
        "There is no `Parse(const Value&, formats::parse::To<T>)` "
        "in namespace of `T` or `formats::parse`. "
        "Probably you forgot to include the "
        "<formats/parse/common_containers.hpp> or you "
        "have not provided a `Parse` function overload.");

    return Parse(*this, formats::parse::To<T>{});
}
```

# Modern C++

# Modern C++17

# C++17



# C++17

- `std::optional`

# C++17

- `std::optional`
- `std::variant`

# C++17

- `std::optional`
- `std::variant`
- `[[nodiscard]]`

# C++17

- `std::optional`
- `std::variant`
- `[[nodiscard]]`
- `std::string_view`

# C++17

- `std::optional`
- `std::variant`
- `[[nodiscard]]`
- `std::string_view`
- guaranteed copy elision

# C++17

- `std::optional`
- `std::variant`
- `[[nodiscard]]`
- `std::string_view`
- guaranteed copy elision
- `if constexpr`

# Testsuite +

# Tests

```
async def test_ping(service_client):  
    response = await service_client.get('/hello')  
    assert response.status == 200  
    assert response.content == b'Hello world!\n'
```



# Tests

```
async def test_ping(service_client):  
    response = await service_client.get('/hello')  
    assert response.status == 200  
    assert response.content == b'Hello world!\n'
```

# Tests

```
async def test_ping(service_client):  
    response = await service_client.get('/hello')  
    assert response.status == 200  
    assert response.content == b'Hello world!\n'
```

# Tests

```
async def test_ping(service_client):  
    response = await service_client.get('/hello')  
    assert response.status == 200  
    assert response.content == b'Hello world!\n'
```

# Tests

```
async def test_ping(service_client):  
    response = await service_client.get('/hello')  
    assert response.status == 200  
    assert response.content == b'Hello world!\n'
```

# The Result

# The Result



# The Result



- Efficiency

# The Result



- Efficiency → C++ & async IO



# The Result



- Efficiency → C++ & async IO
- Simplicity of development

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices



# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions

# The Result

- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies

# The Result

- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches

# The Result

- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches
- C++ scares

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches
- C++ scares → well designed solutions

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches
- C++ scares → well designed solutions
- Databases

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches
- C++ scares → well designed solutions
- Databases, Dynamic configs

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches
- C++ scares → well designed solutions
- Databases, Dynamic configs, Tracing



# The Result

- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches
- C++ scares → well designed solutions
- Databases, Dynamic configs, Tracing, Metrics

# The Result

- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches
- C++ scares → well designed solutions
- Databases, Dynamic configs, Tracing, Metrics, Deadlines

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

## Solutions:

- Latencies → caches
- C++ scares → well designed solutions
- Databases, Dynamic configs, Tracing, Metrics, Deadlines, Distlocks

# The Result



- Efficiency → C++ & async IO
- Simplicity of development → microservices & stackfull coroutines
- High development speed → testsuite & compile time & ready solutions
- Safety → compile time & Yandex scale tested tools
- Scalability → microservices

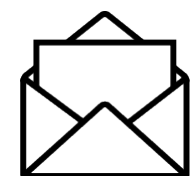
## Solutions:

- Latencies → caches
- C++ scares → well designed solutions
- Databases, Dynamic configs, Tracing, Metrics, Deadlines, Distlocks
- ...

Thanks for watching!

# Antony Polukhin

C++ Expert developer, Team Lead



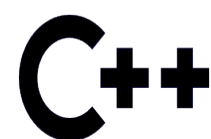
[antoshkka@gmail.com](mailto:antoshkka@gmail.com)



[antoshkka@yandex-team.ru](mailto:antoshkka@yandex-team.ru)

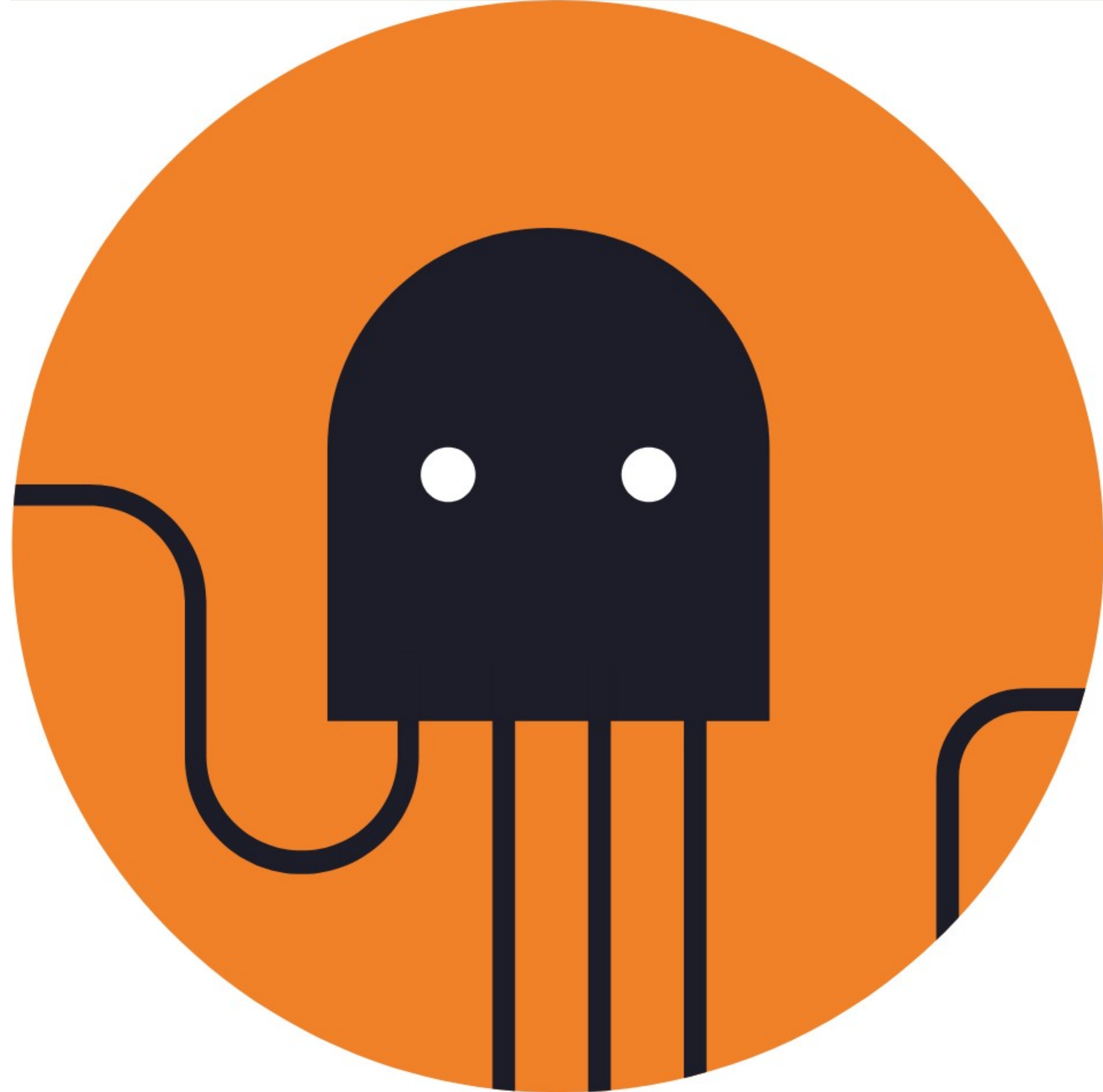


<https://github.com/apolukhin>



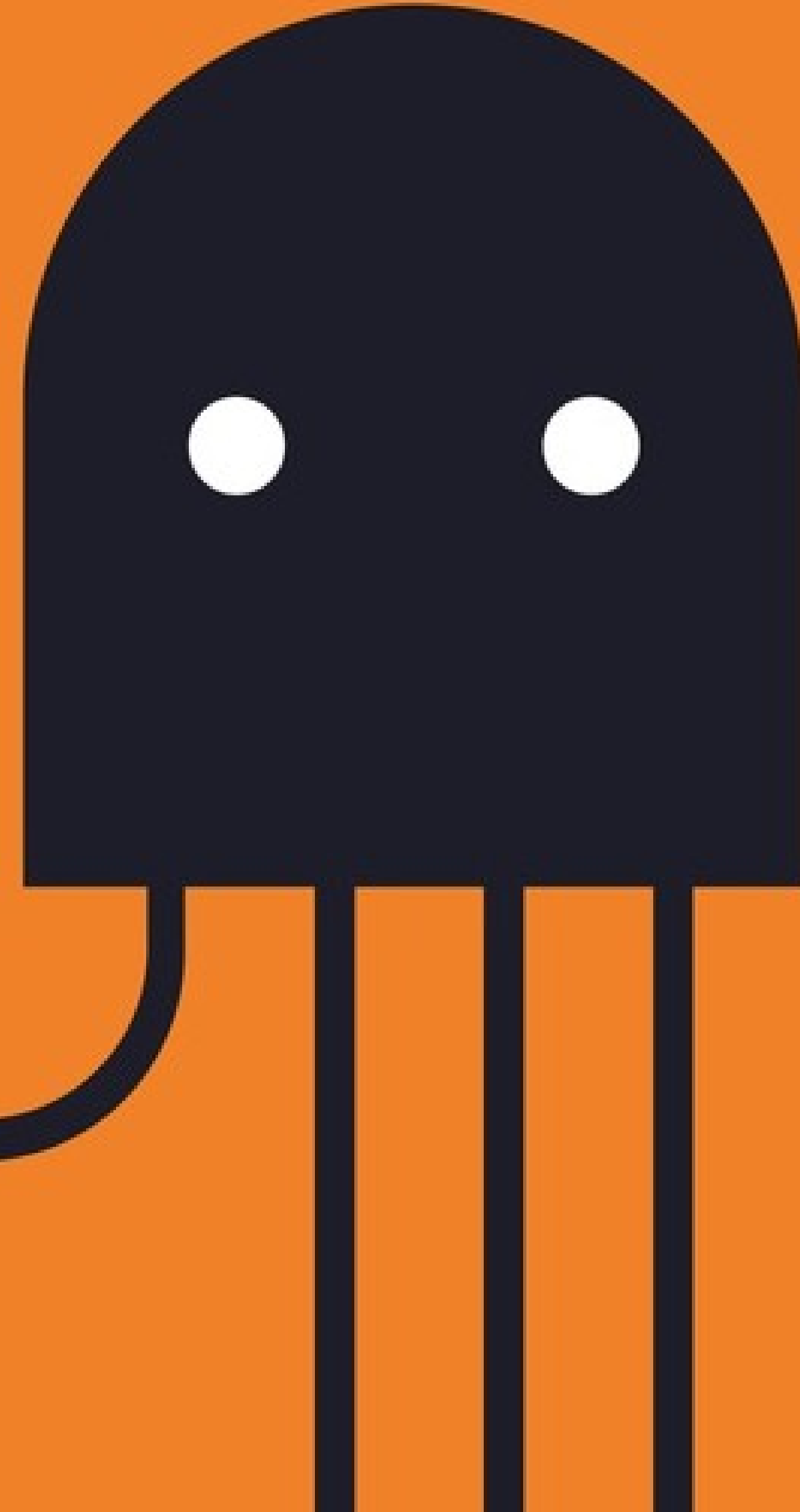
РГ21 C++ РОССИЯ

<https://stdcpp.ru/>



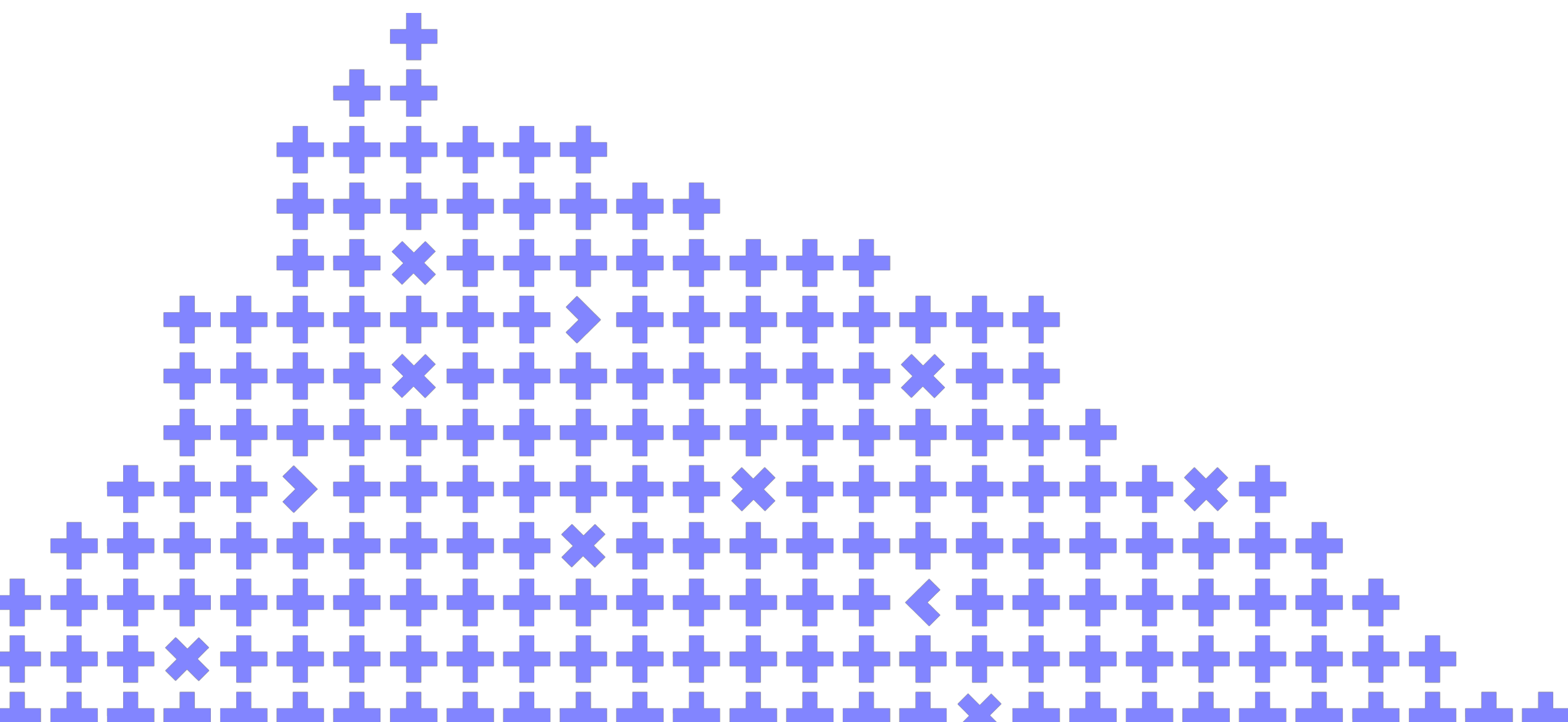
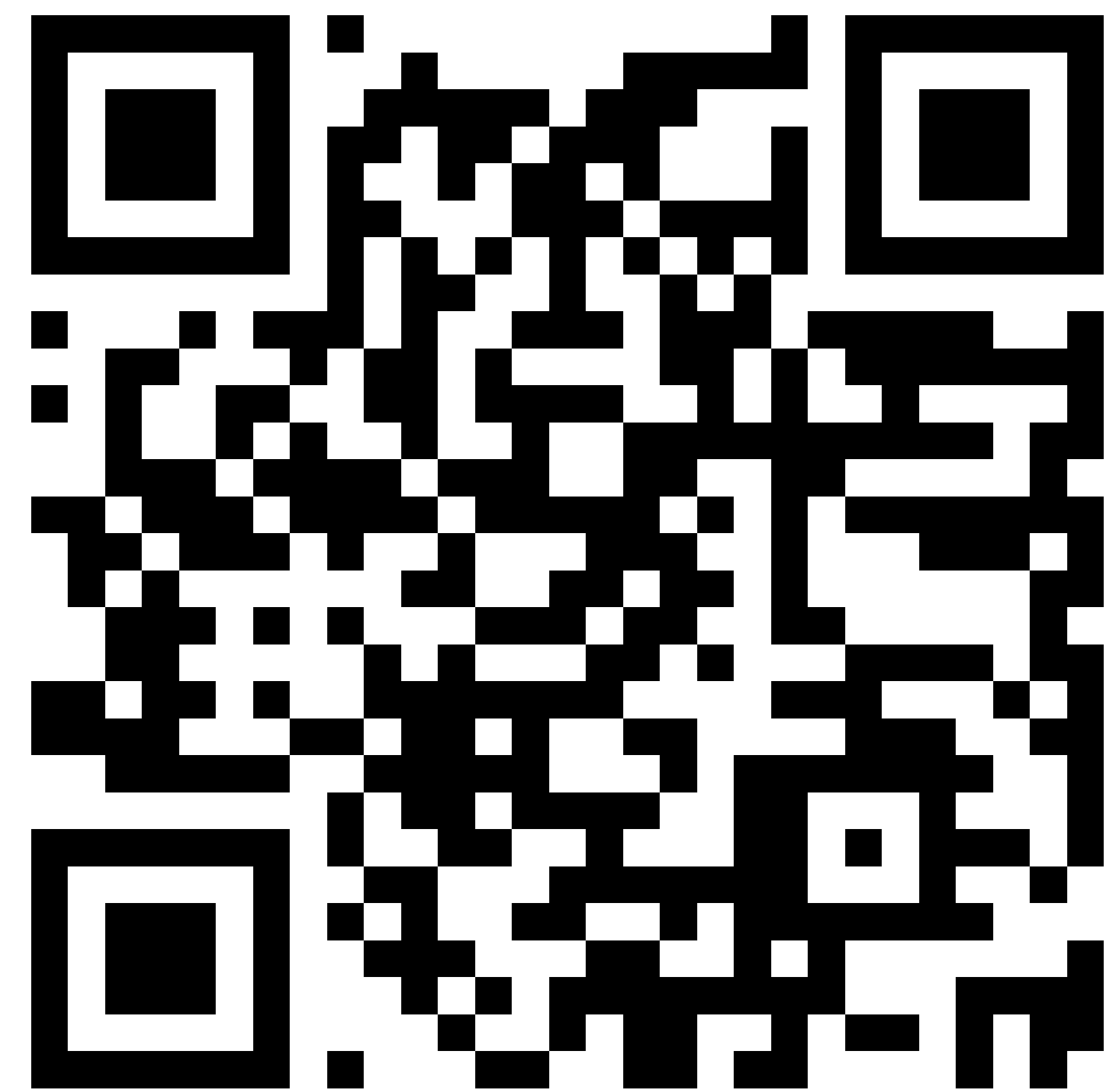
<https://github.com/userver-framework>

<https://userver.tech/>



**Leave your feedback!**

**You can rate the talk and  
give a feedback on what  
you've liked or what  
could be improved**



Co-organizer

