Монолит vs Микросервисы

и как эффективно работать с последними в C++

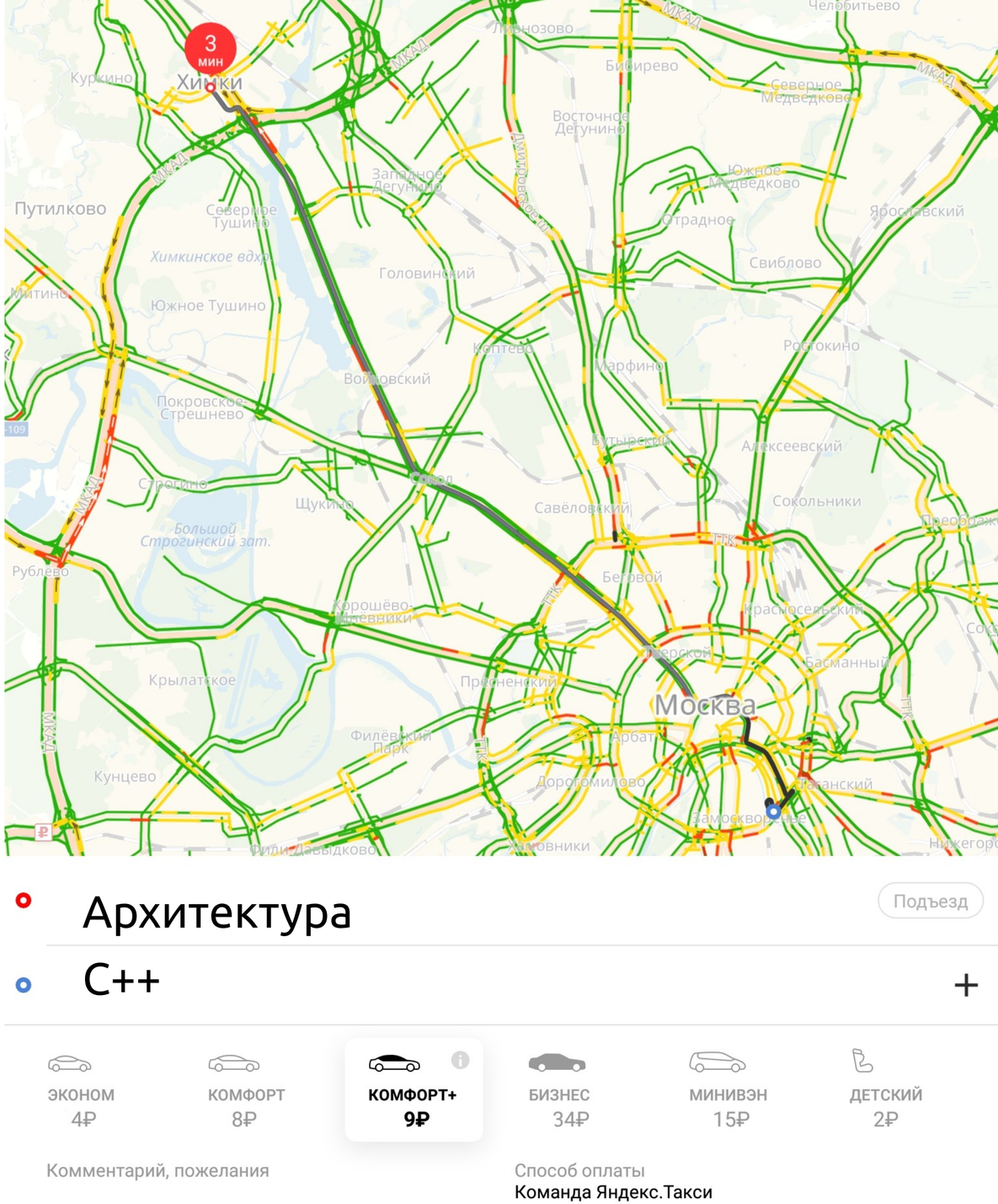
Полухин Антон

Antony Polukhin

Яндекс Go

Содержание

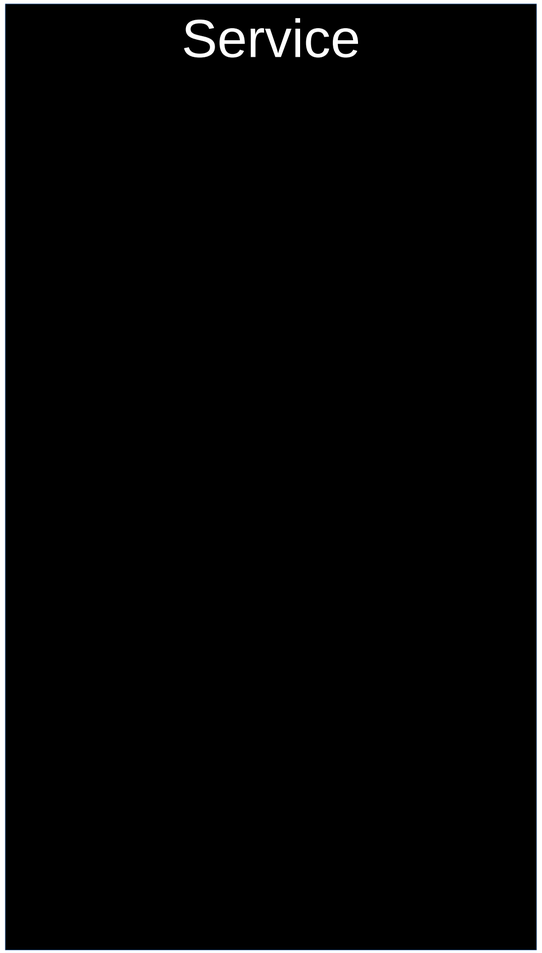
- Архитектуры
  - Монолит
  - Лучшая архитектура!
  - Микросервисы
- Выбор фреймворка
- Latency
- Динамические конфиги
- Фичи
- C++ хардкор с логами



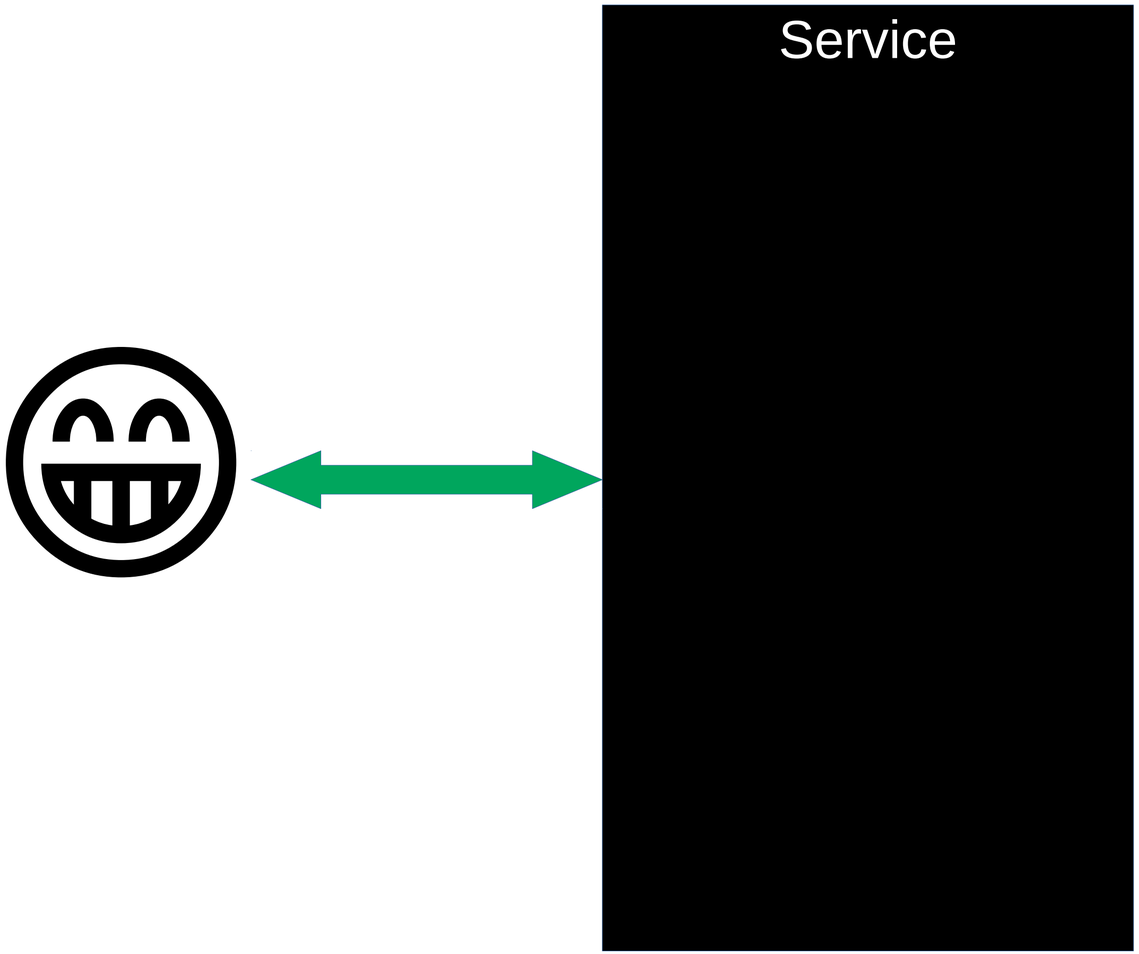
Монолит

Монолит

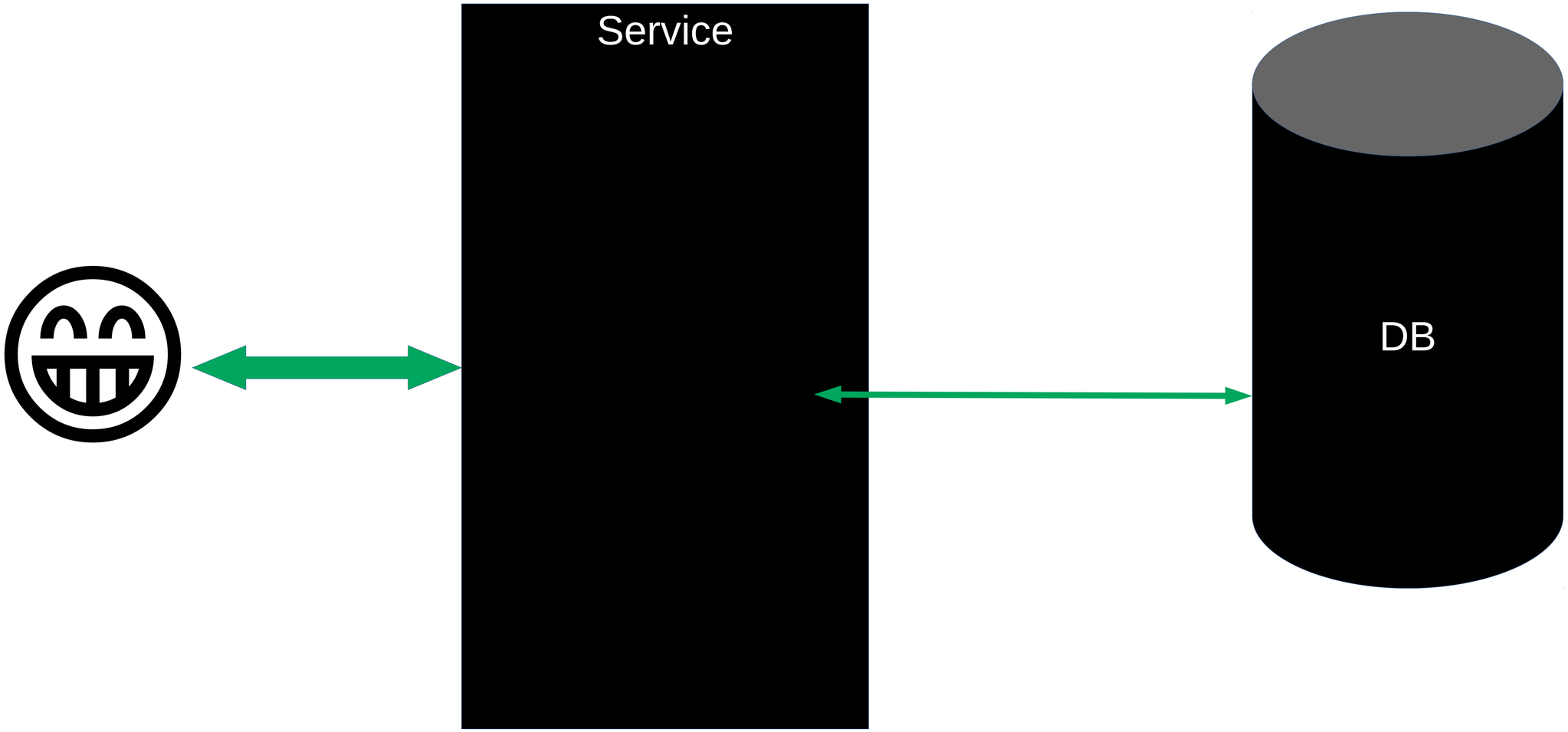
Монолит



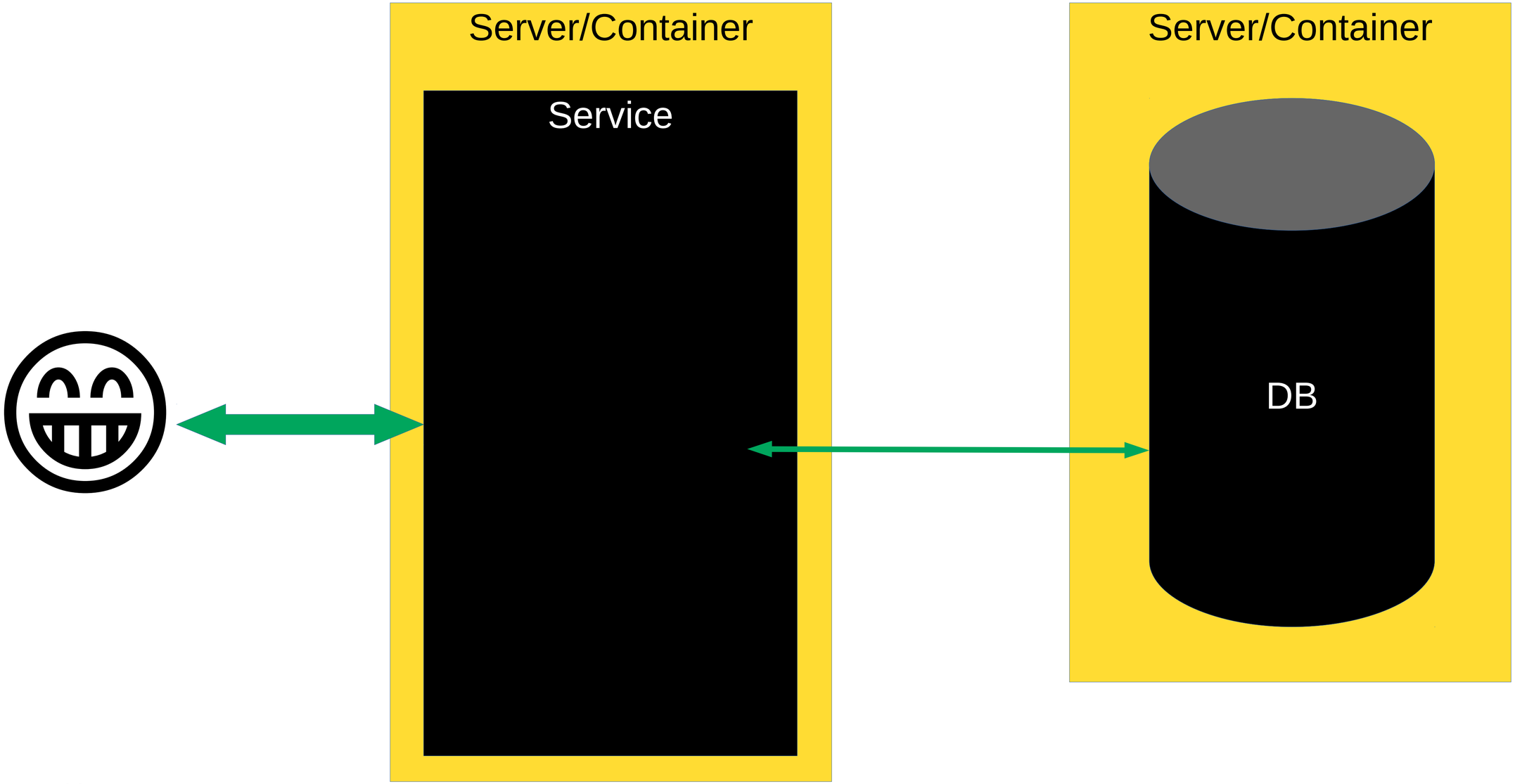
Монолит



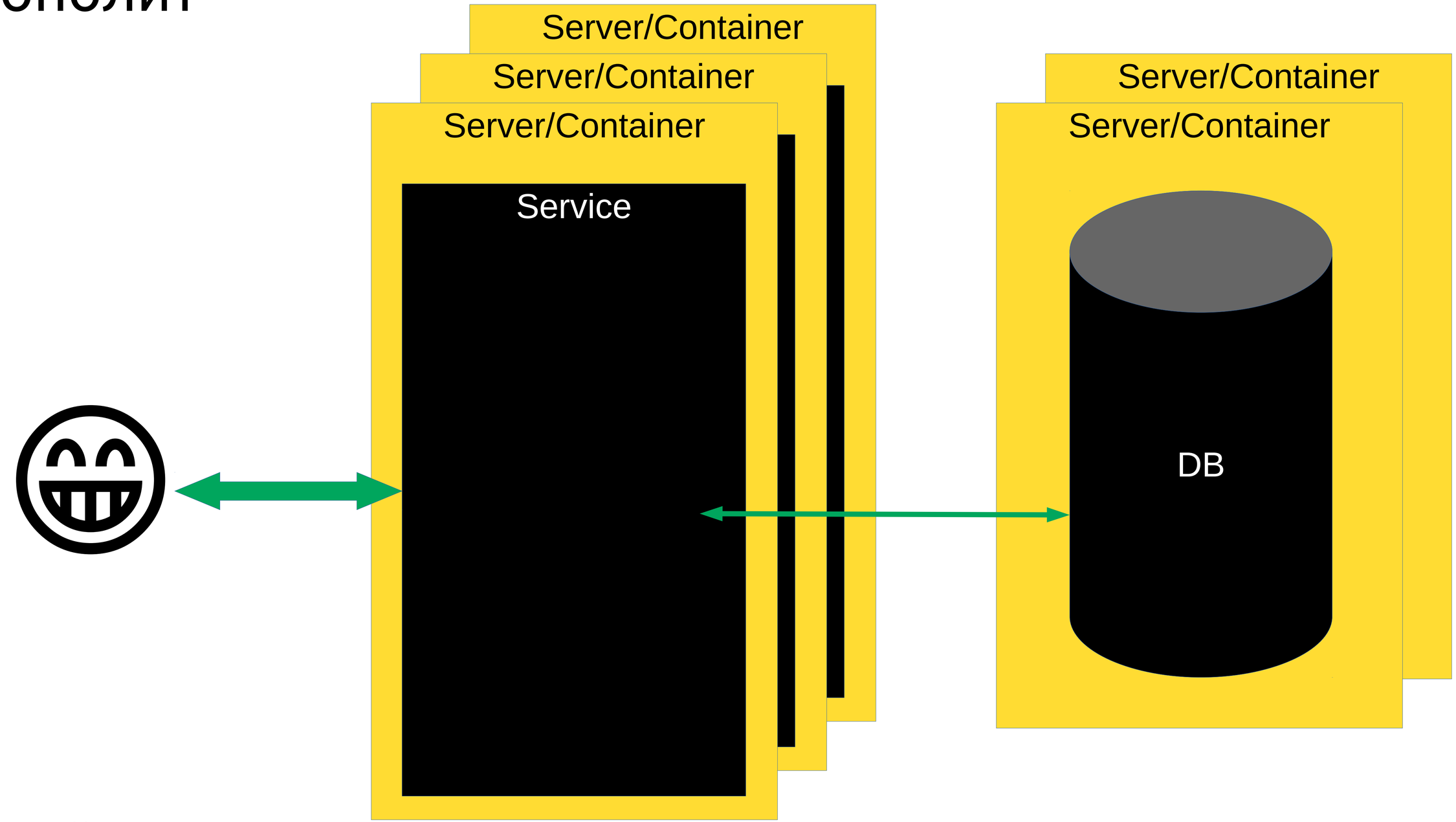
Монолит



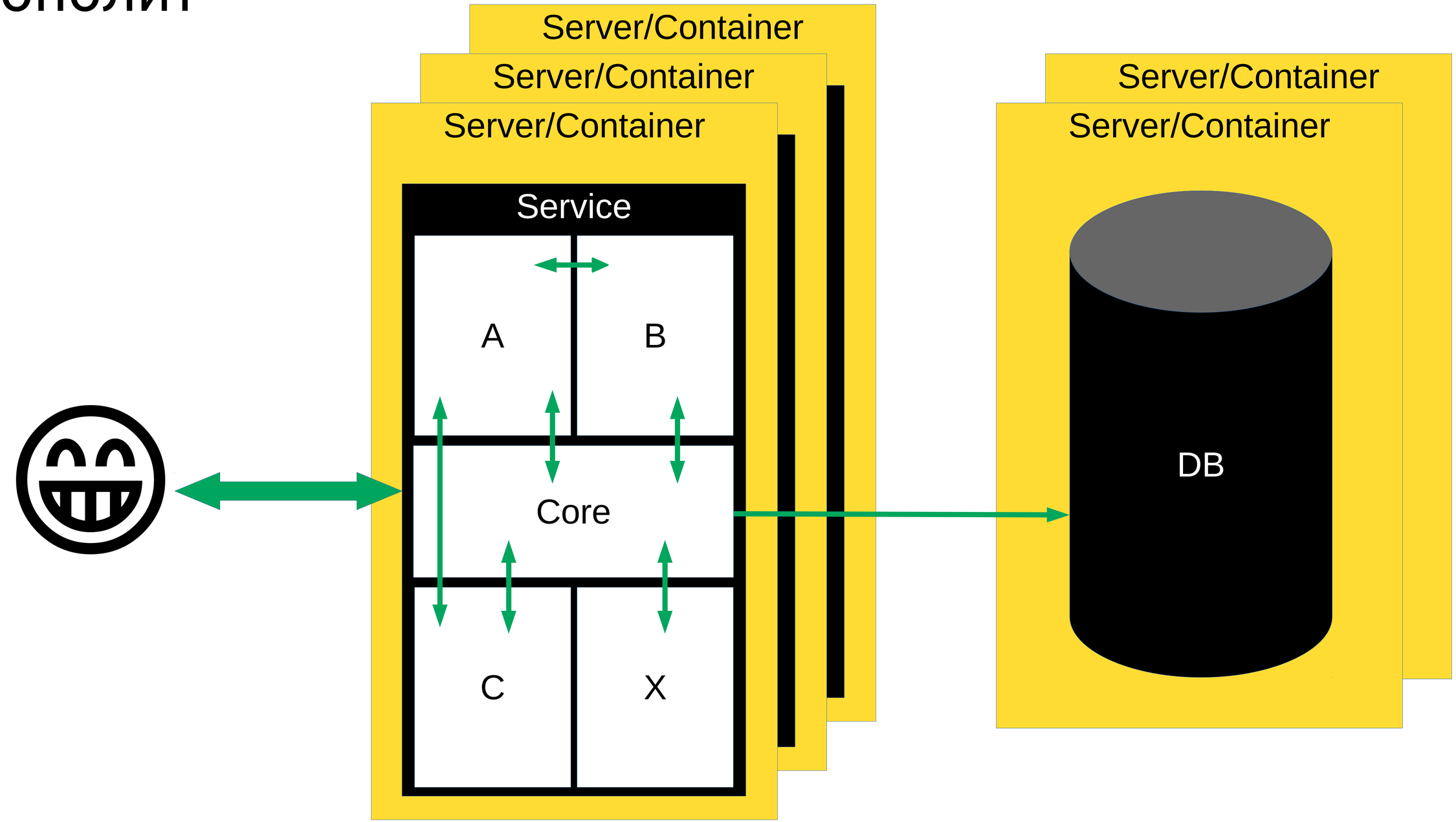
Монолит



Монолит

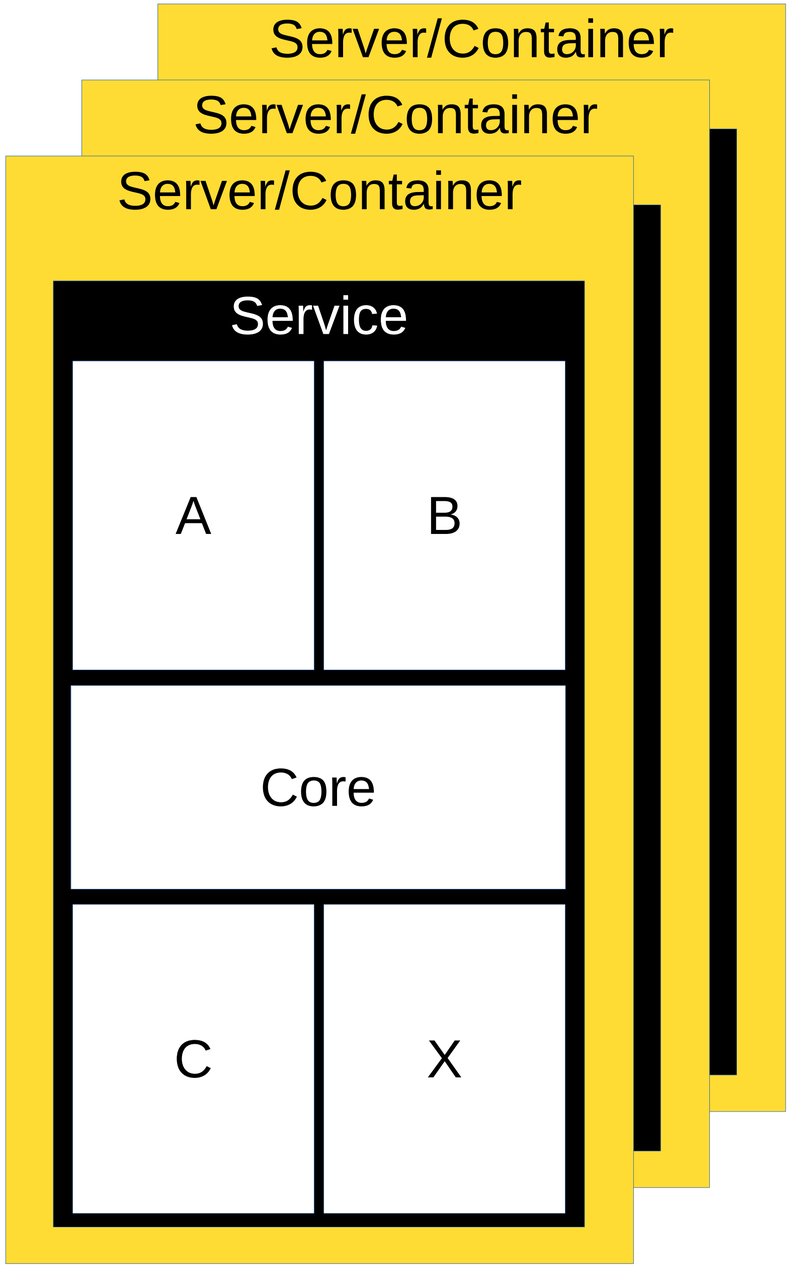

Server/Container
Server/Container
Server/Container
Service
Server/Container
Server/Container
DB

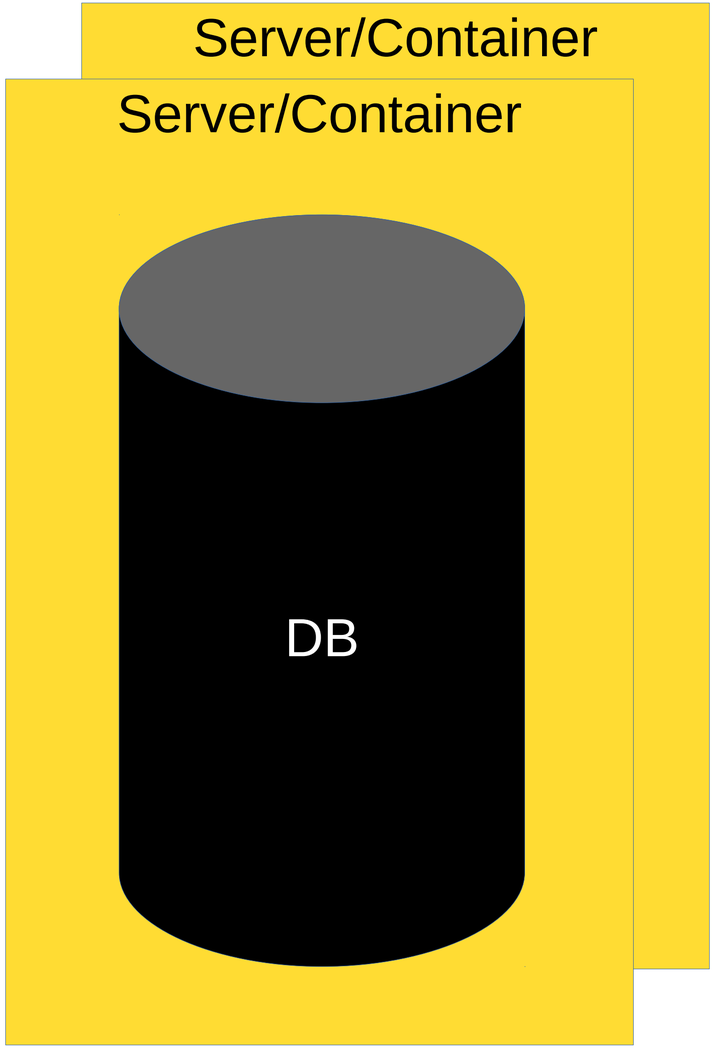
Монолит


Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

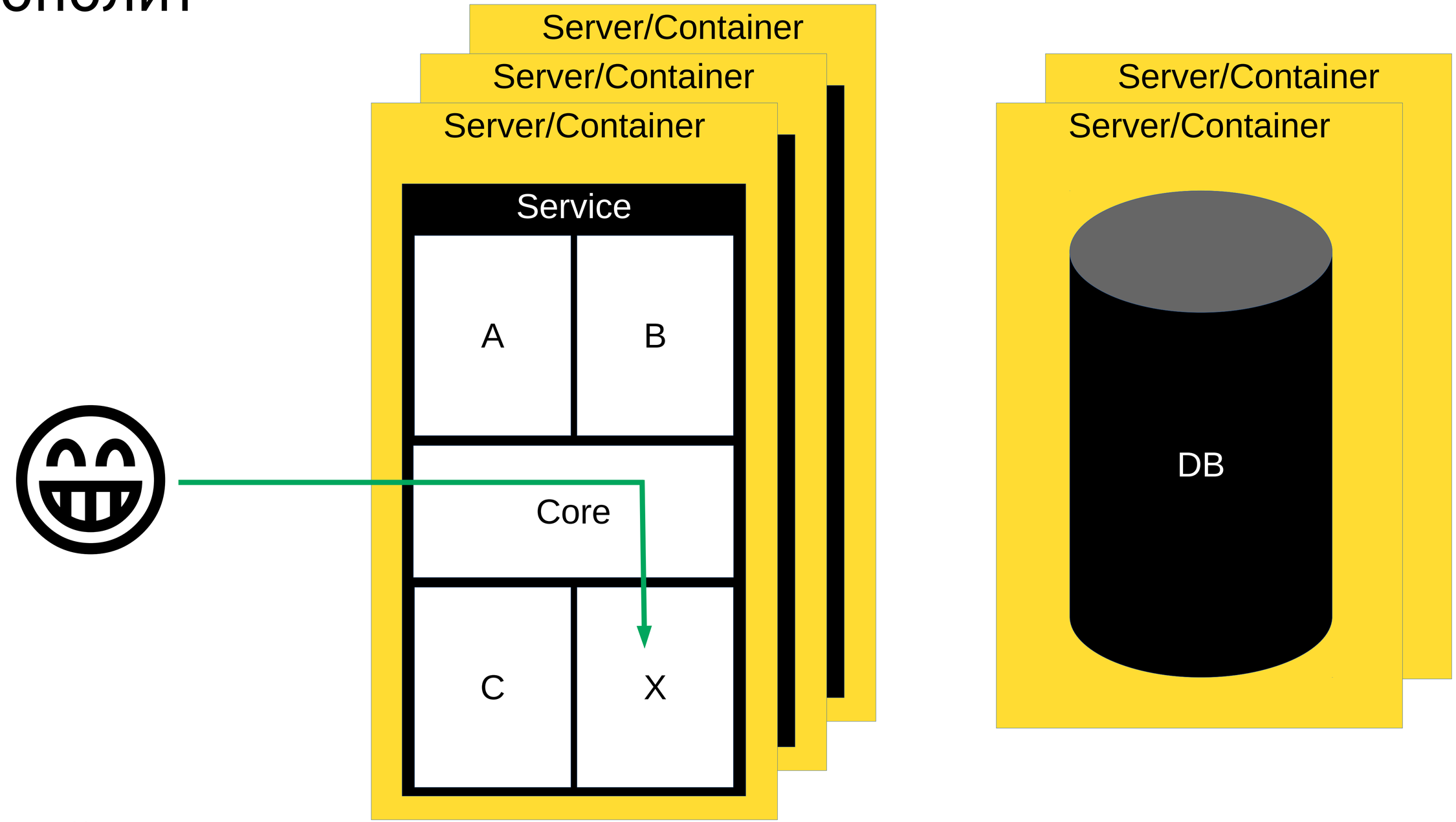
Минусы: доступность

Монолит

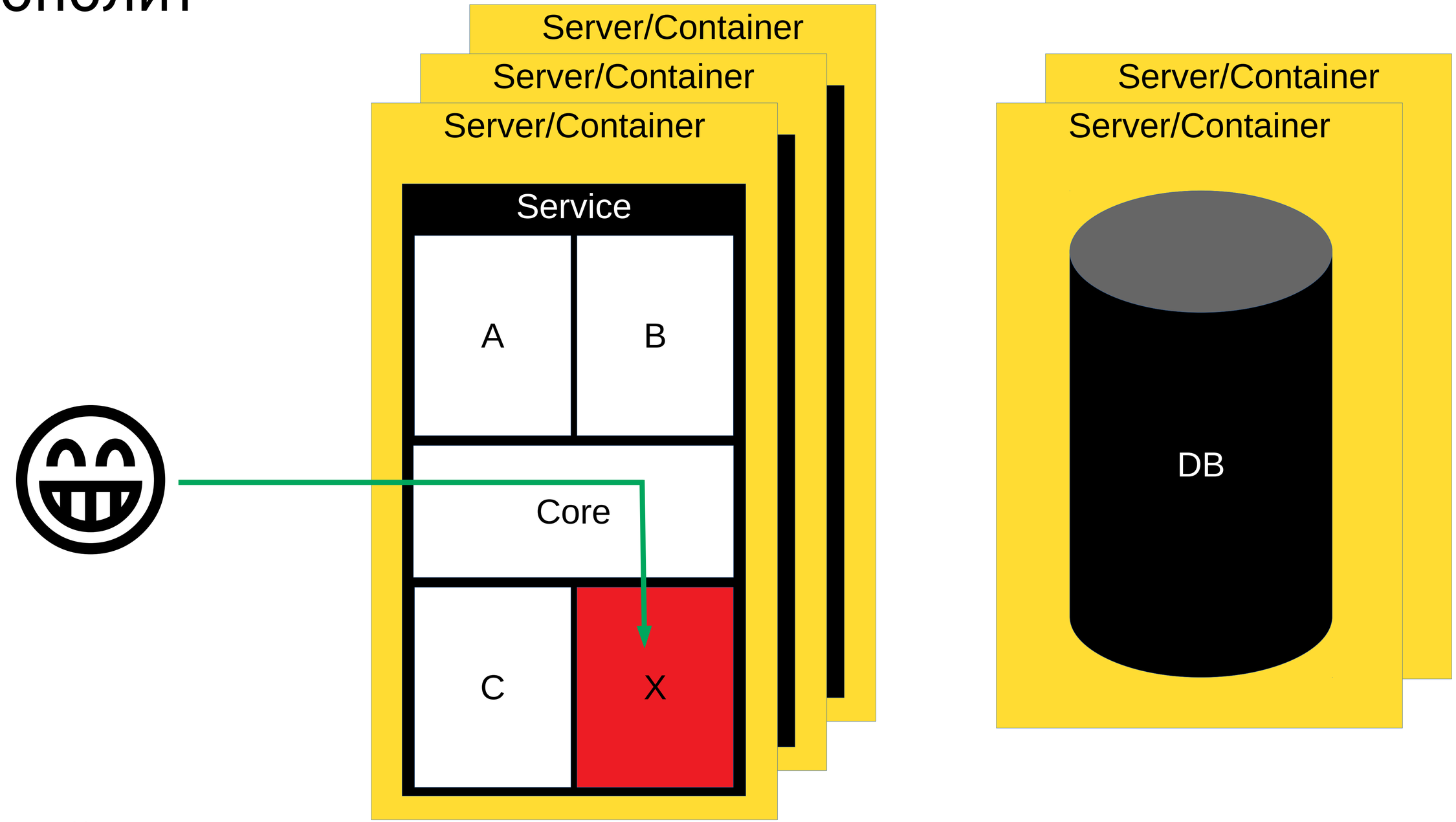

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X


Server/Container
Server/Container
DB

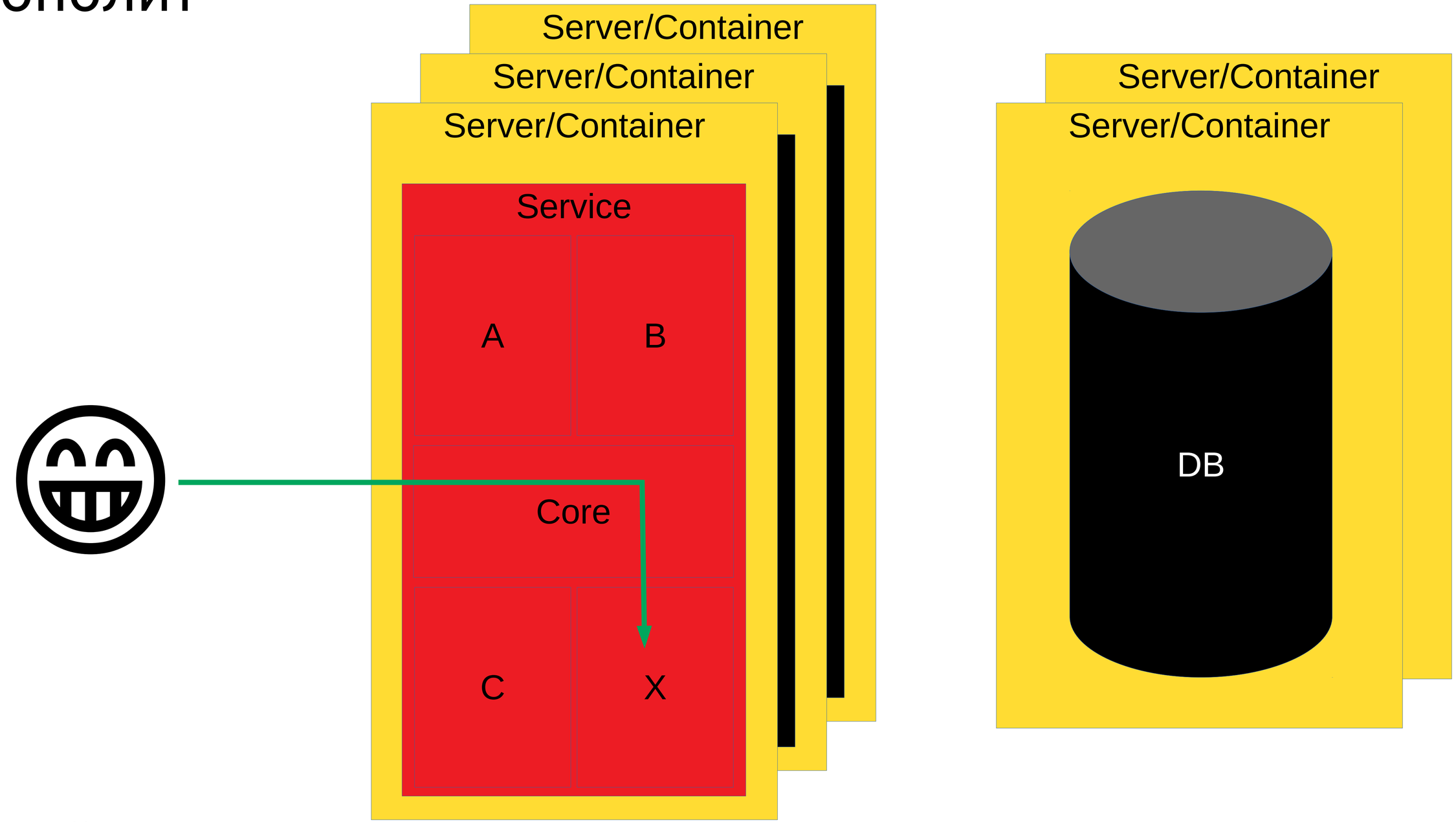
Монолит


Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

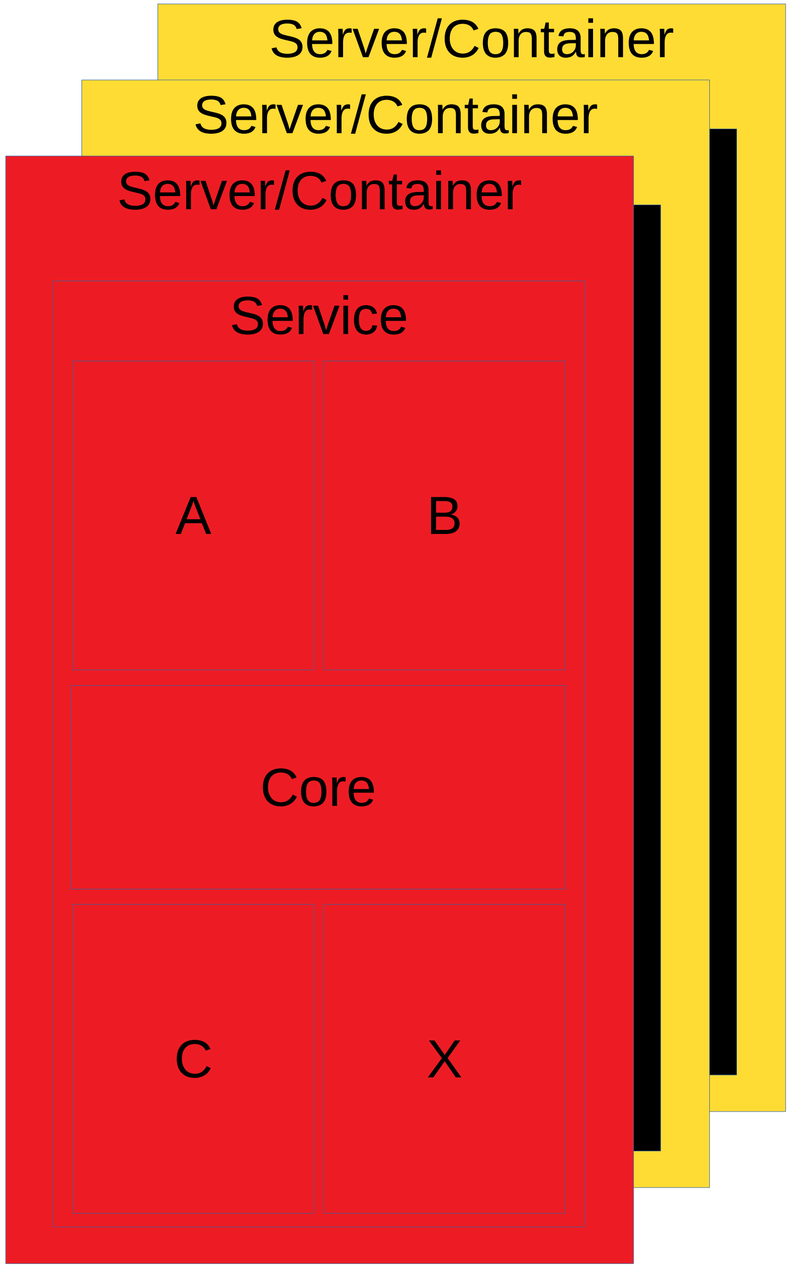
Монолит

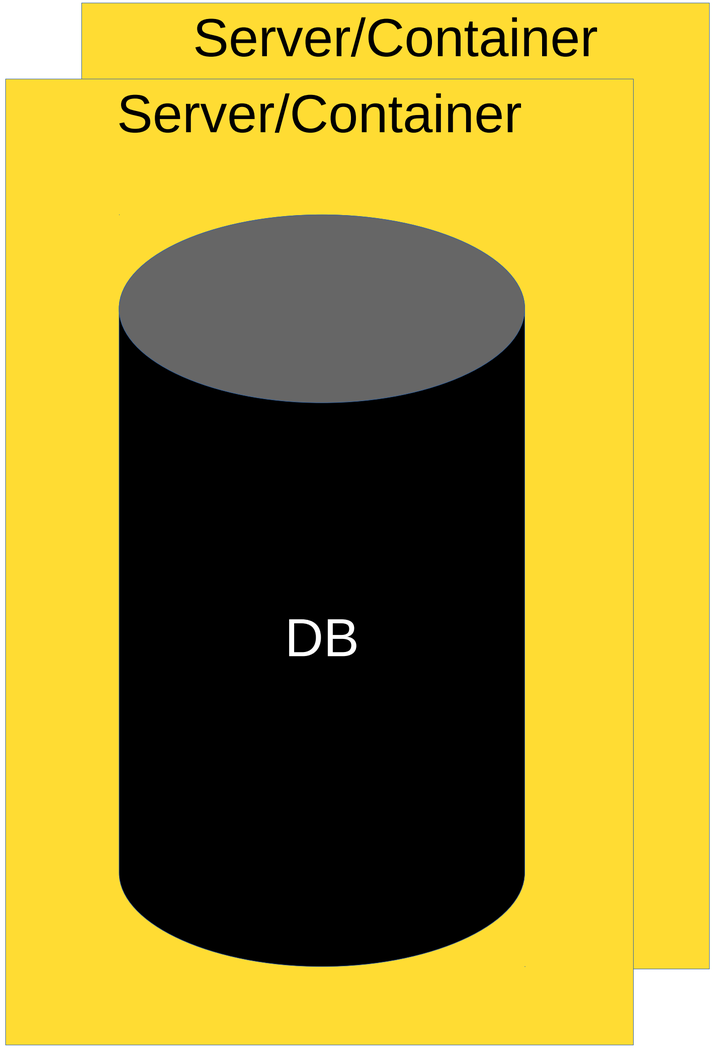

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

Монолит

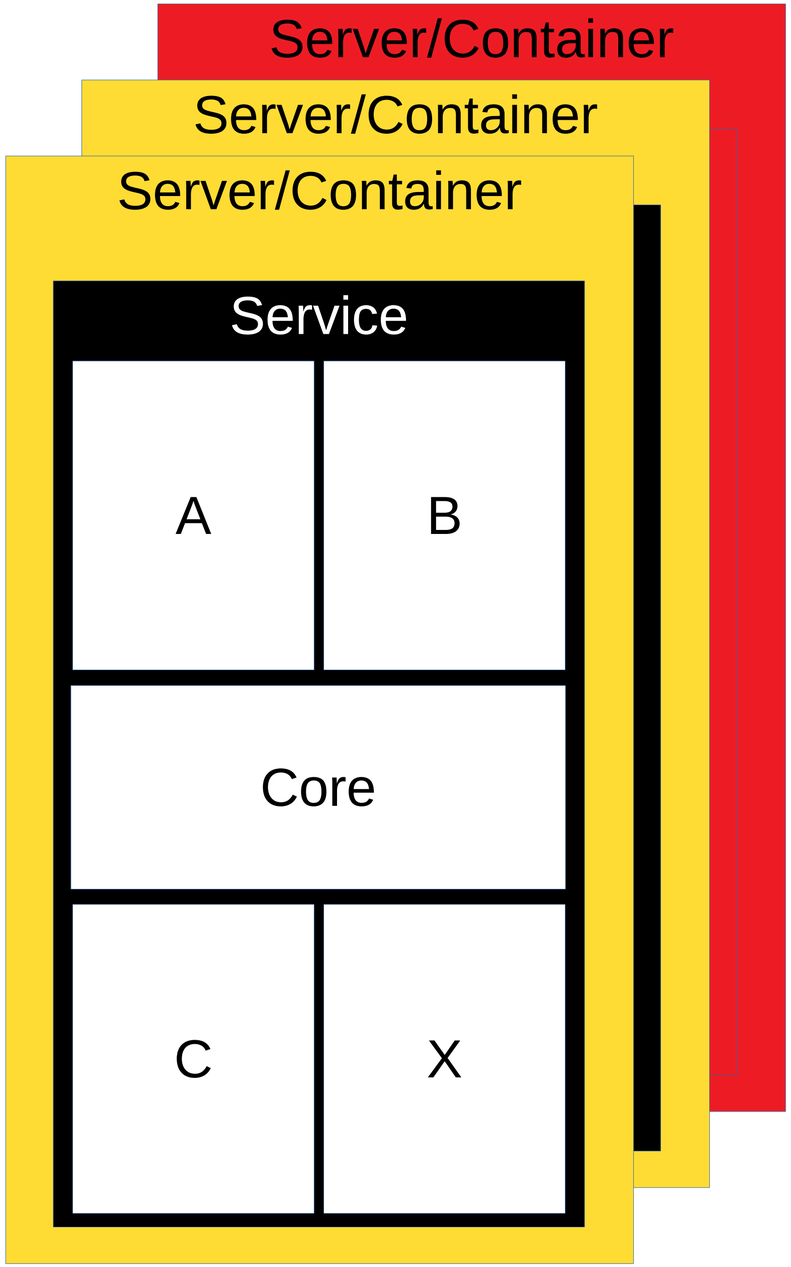

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

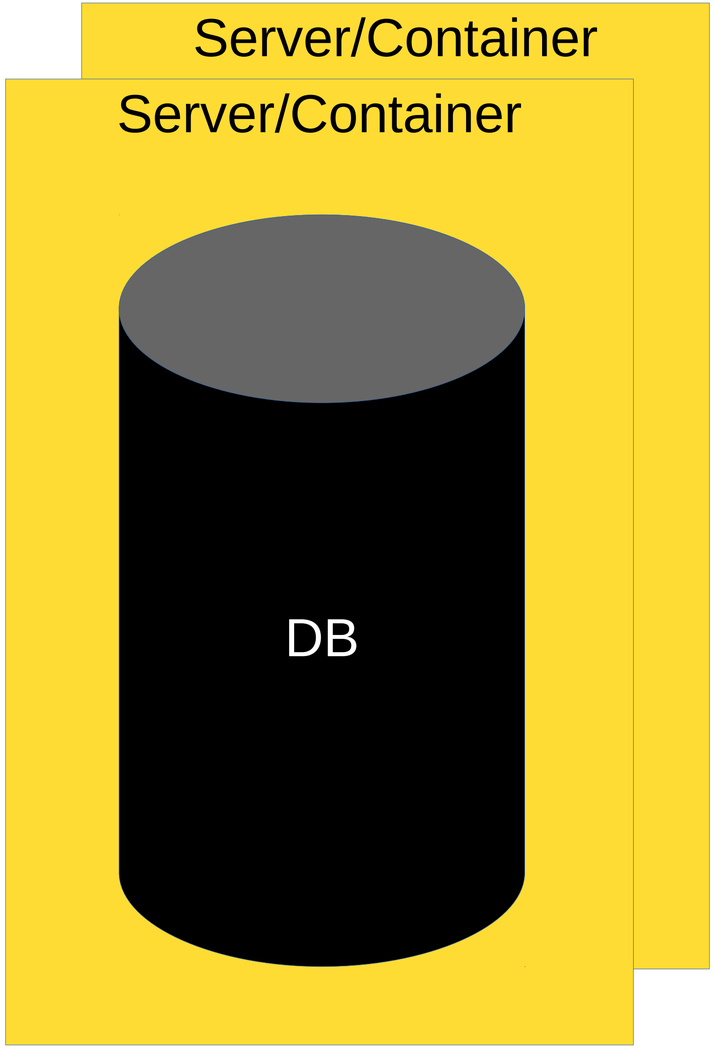
Монолит


Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X

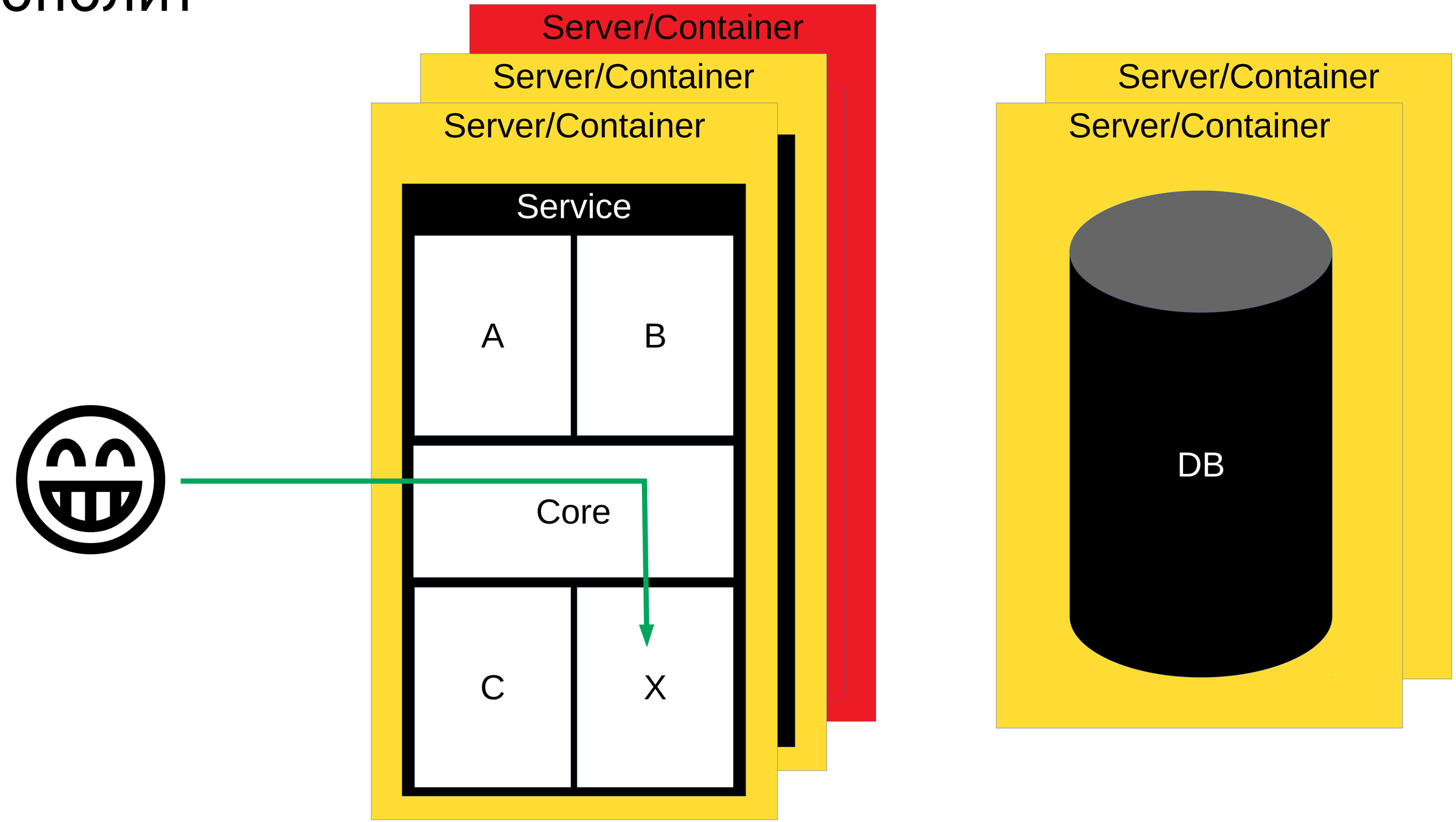

Server/Container
Server/Container
DB

Монолит

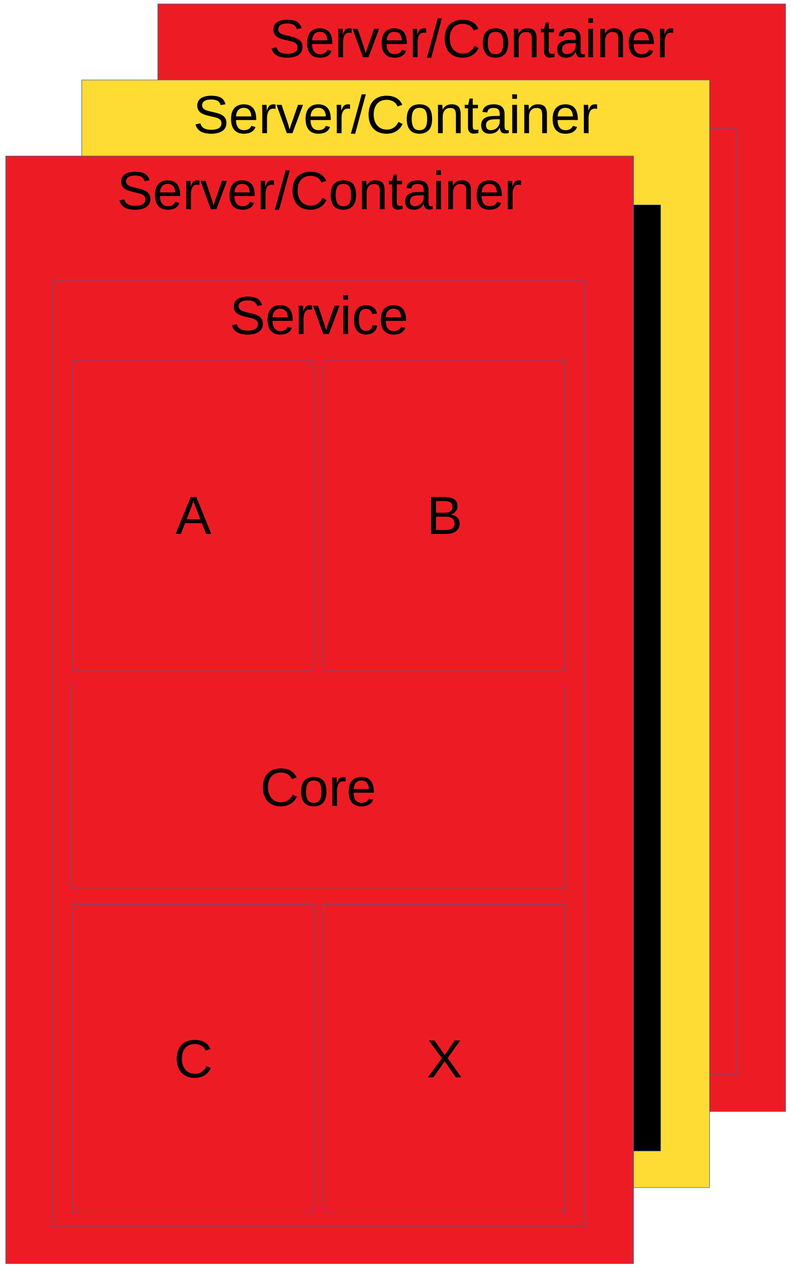

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X

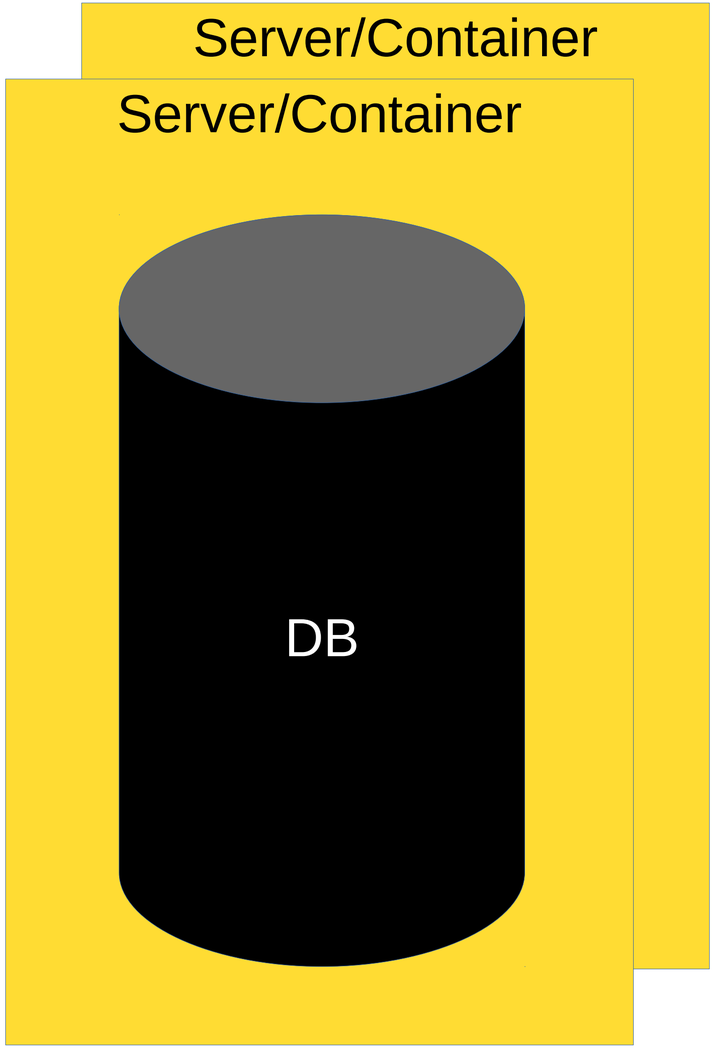

Server/Container
Server/Container
DB

Монолит

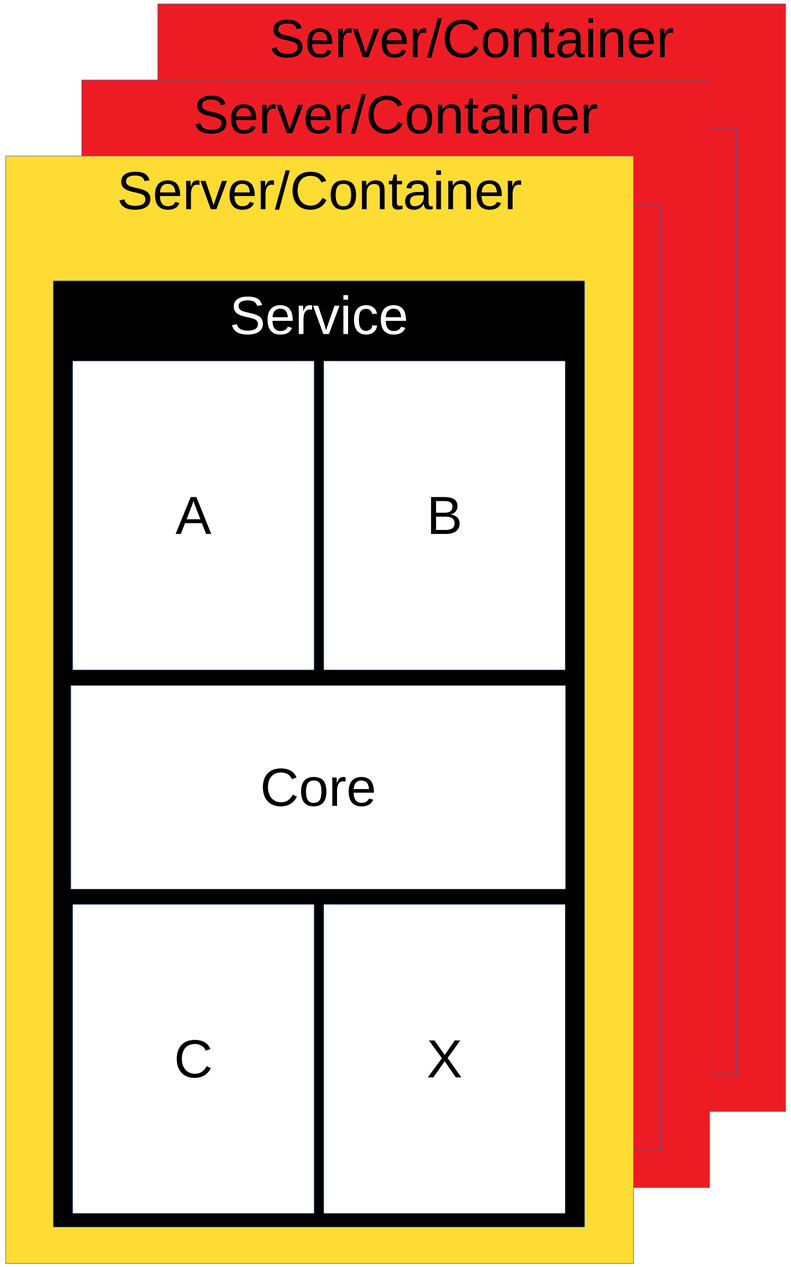

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

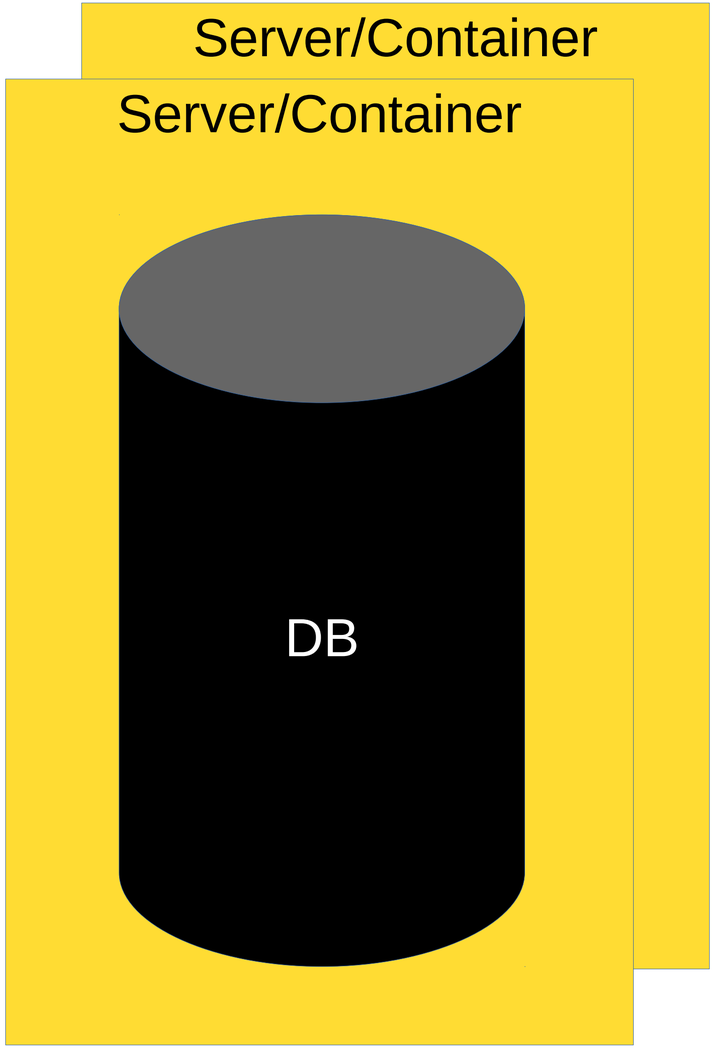
Монолит


Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X

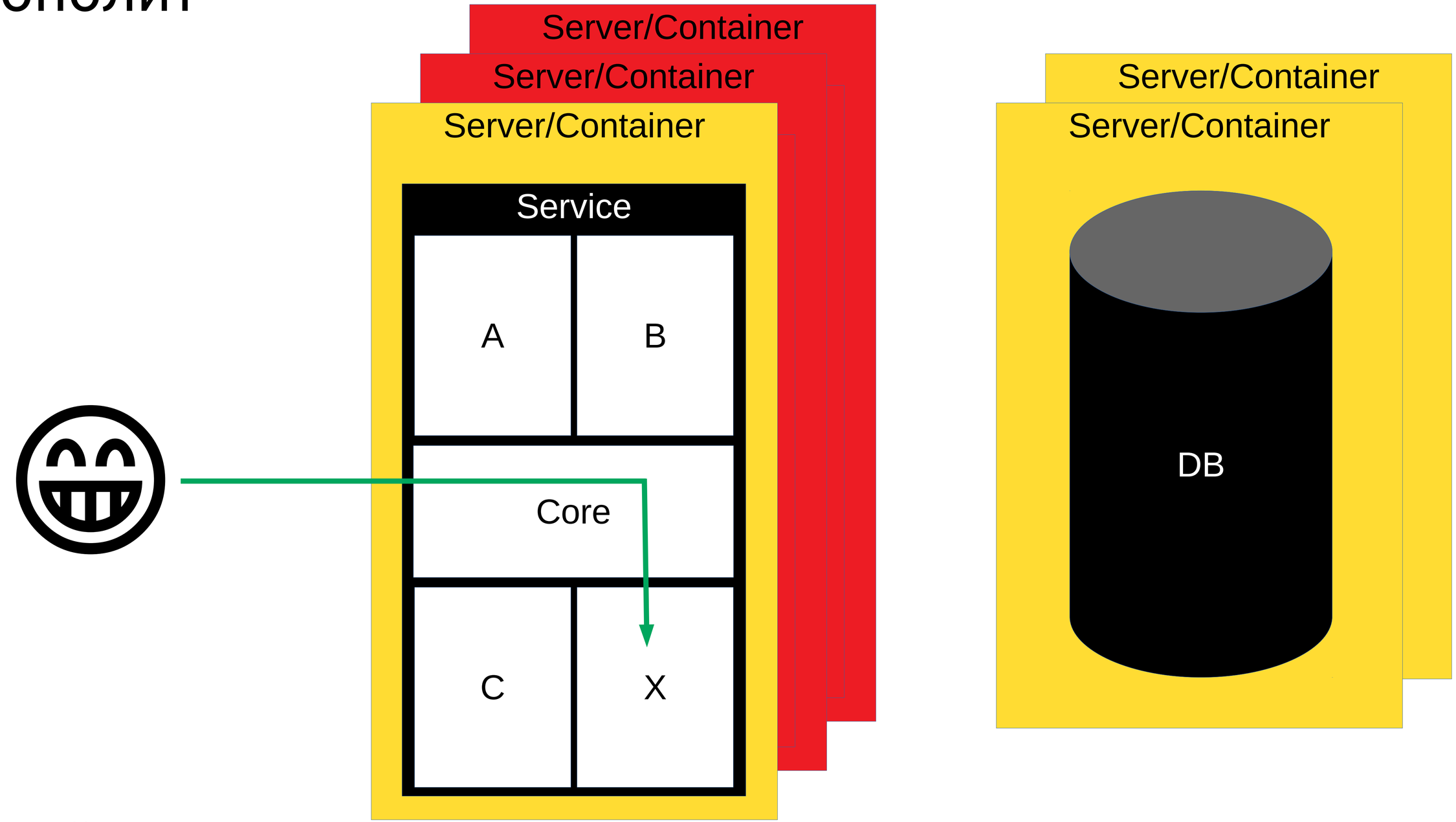

Server/Container
Server/Container
DB

Монолит

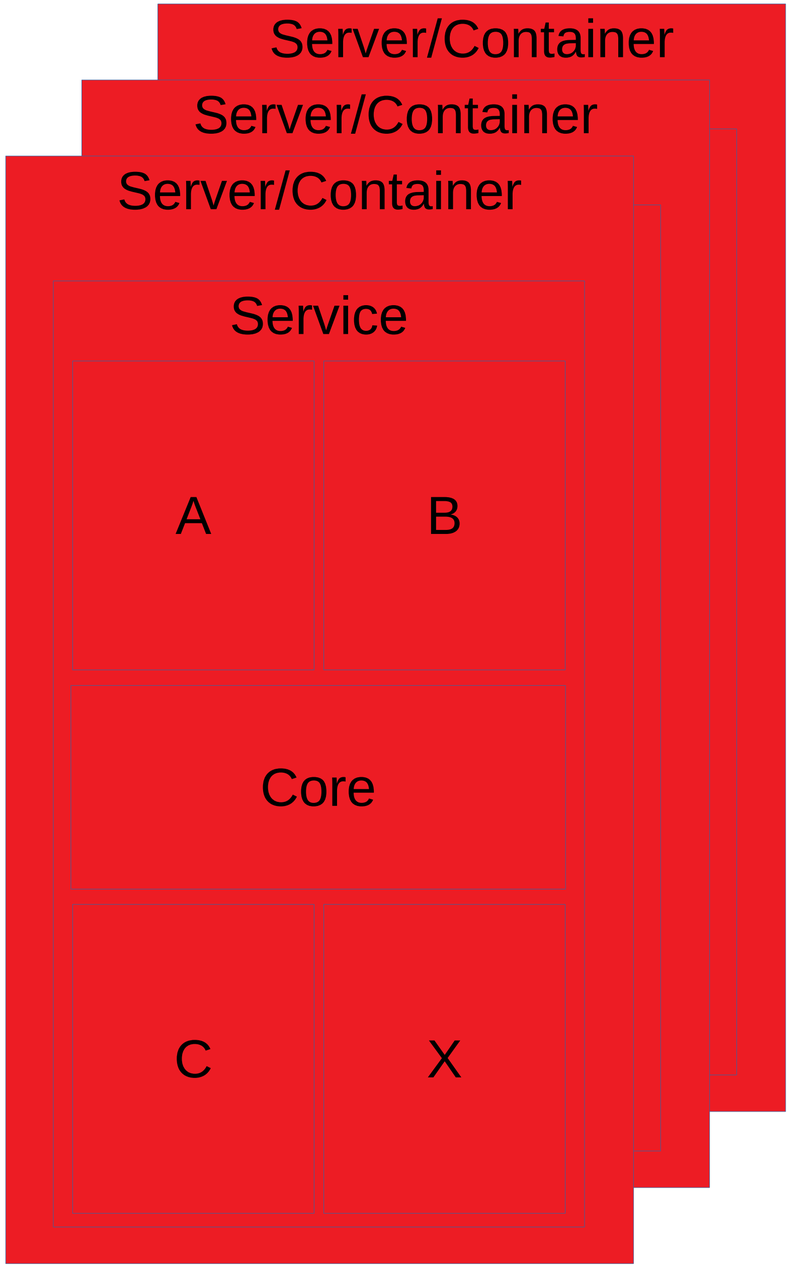

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X

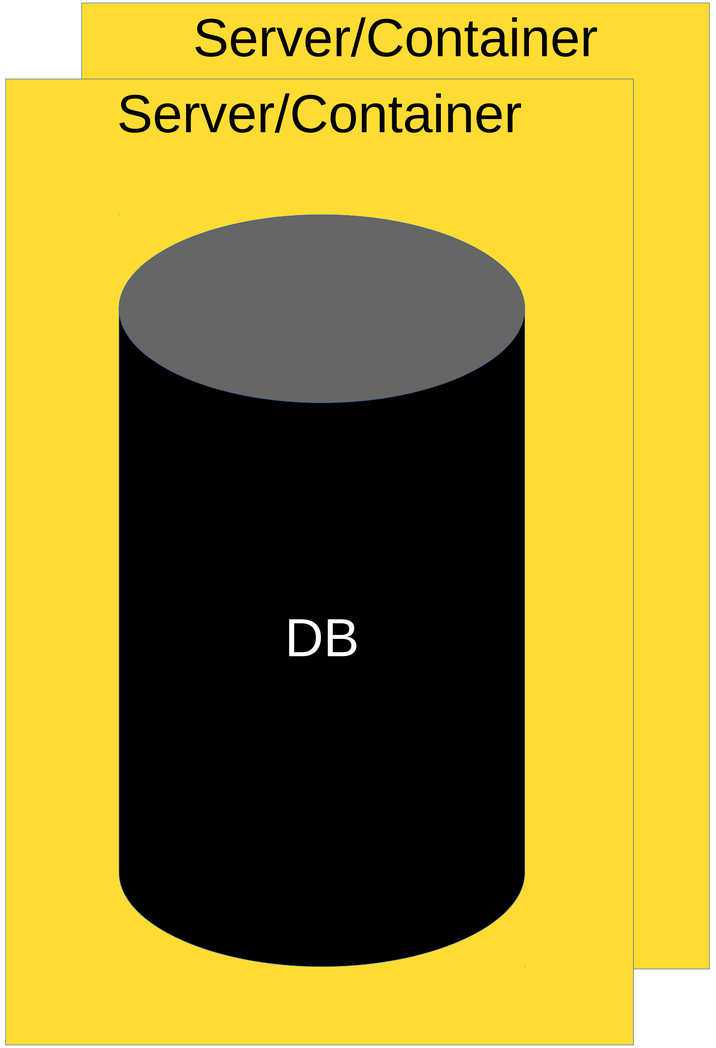

Server/Container
Server/Container
DB

Монолит

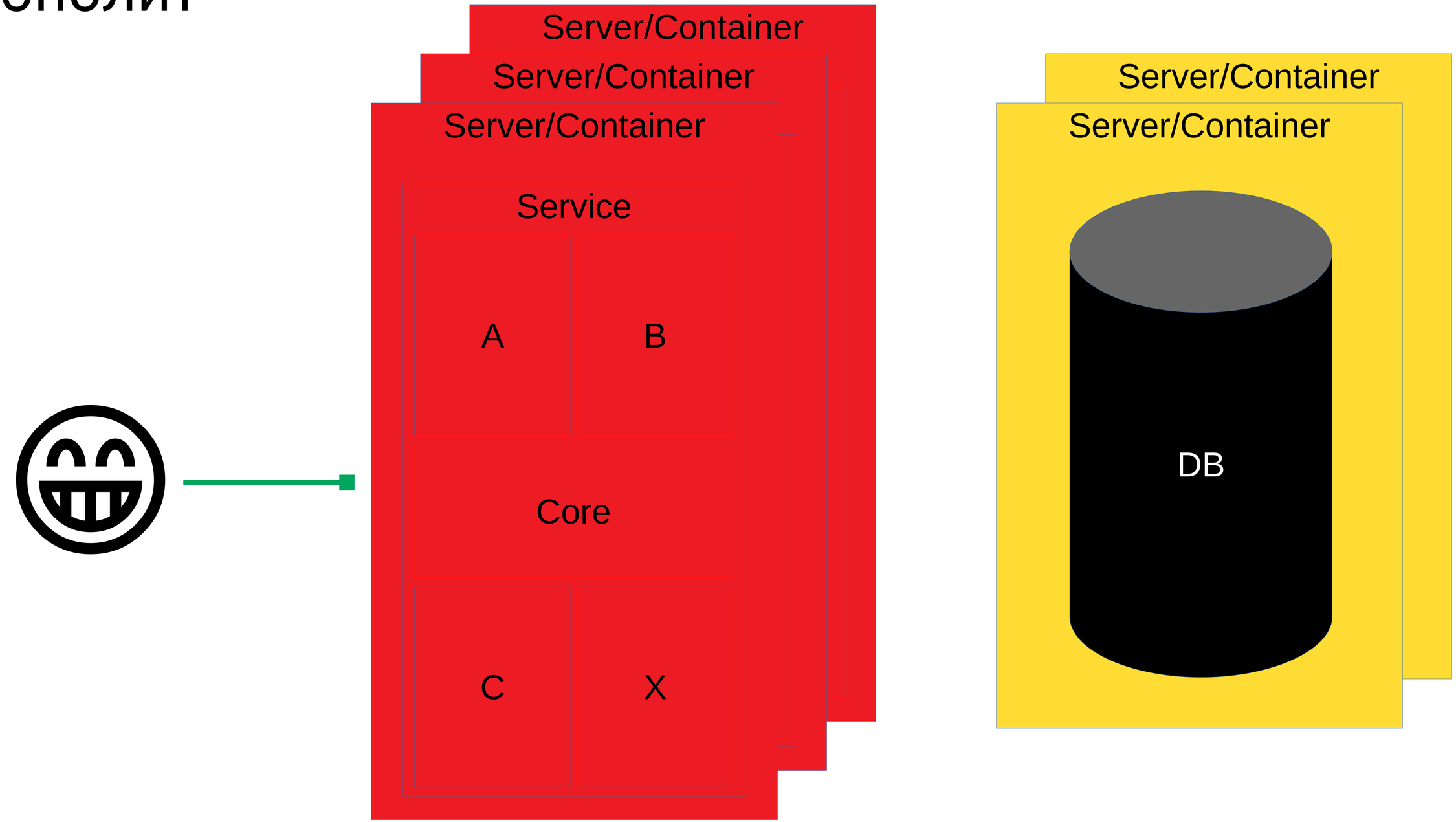

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

Монолит

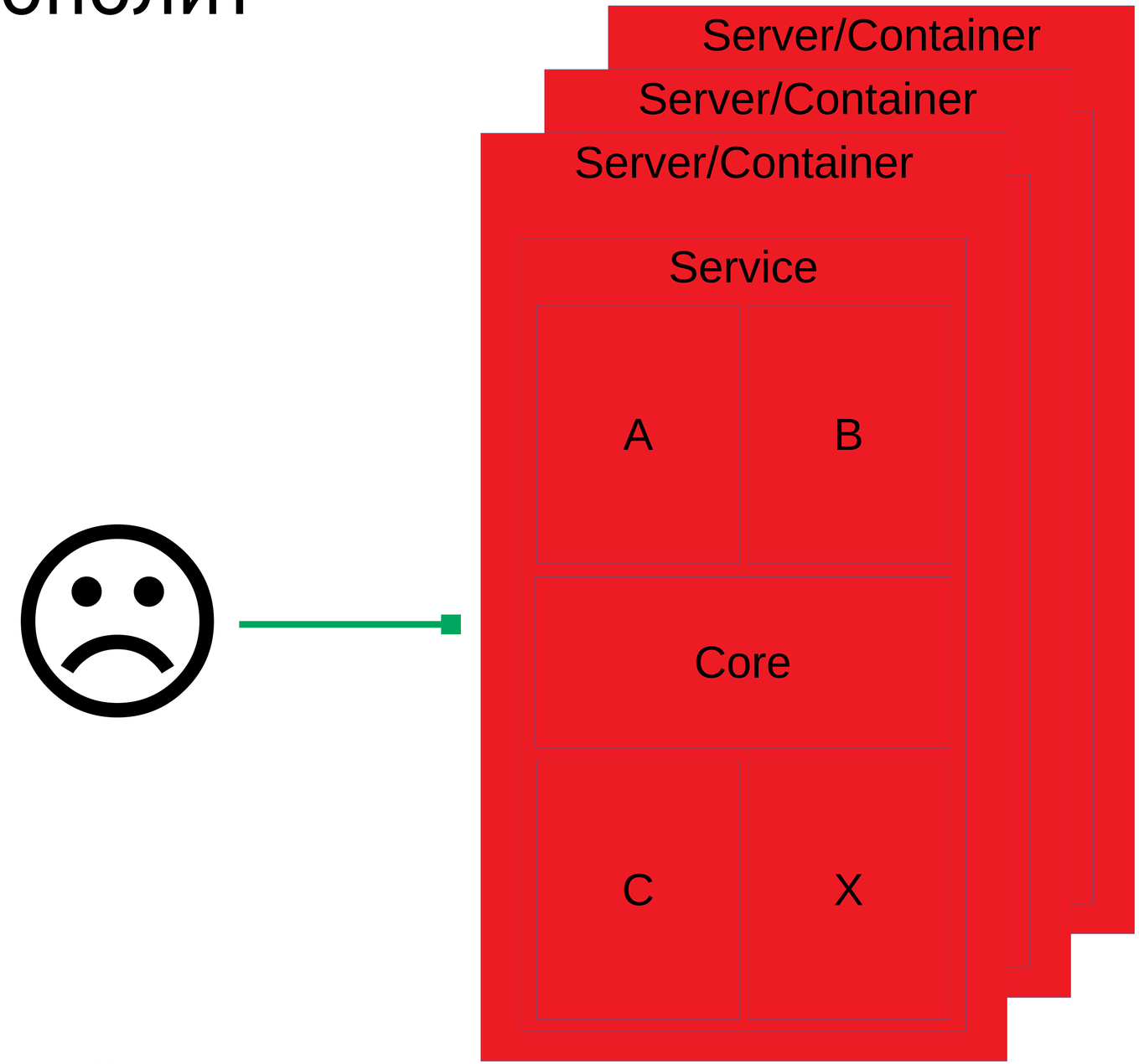

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X

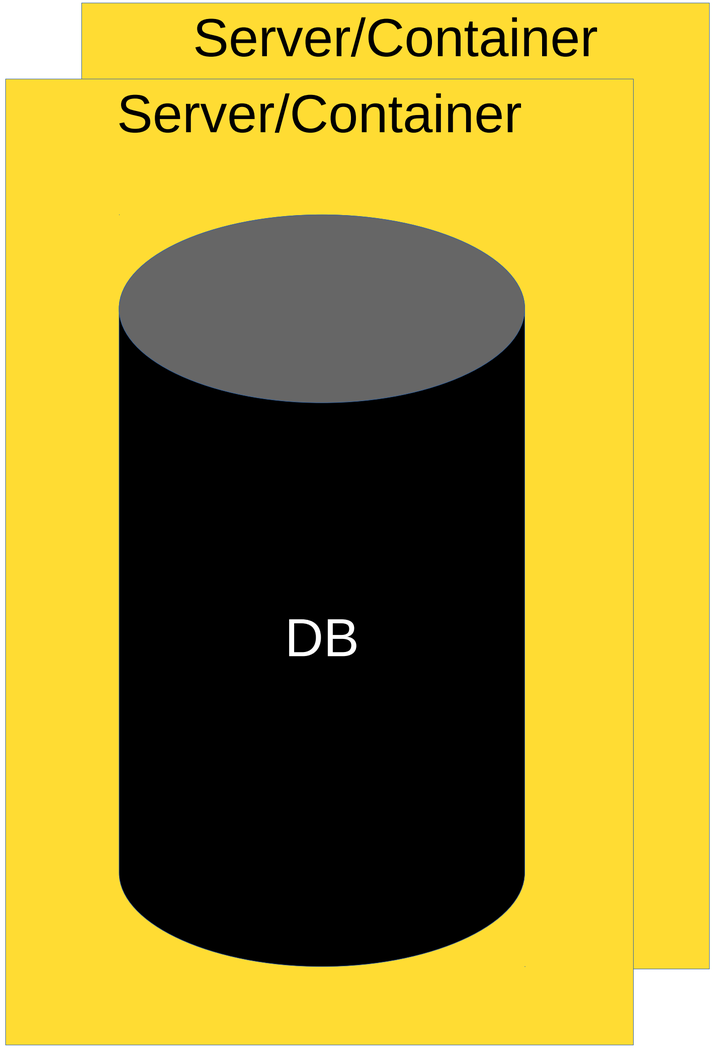

Server/Container
Server/Container
DB

Монолит

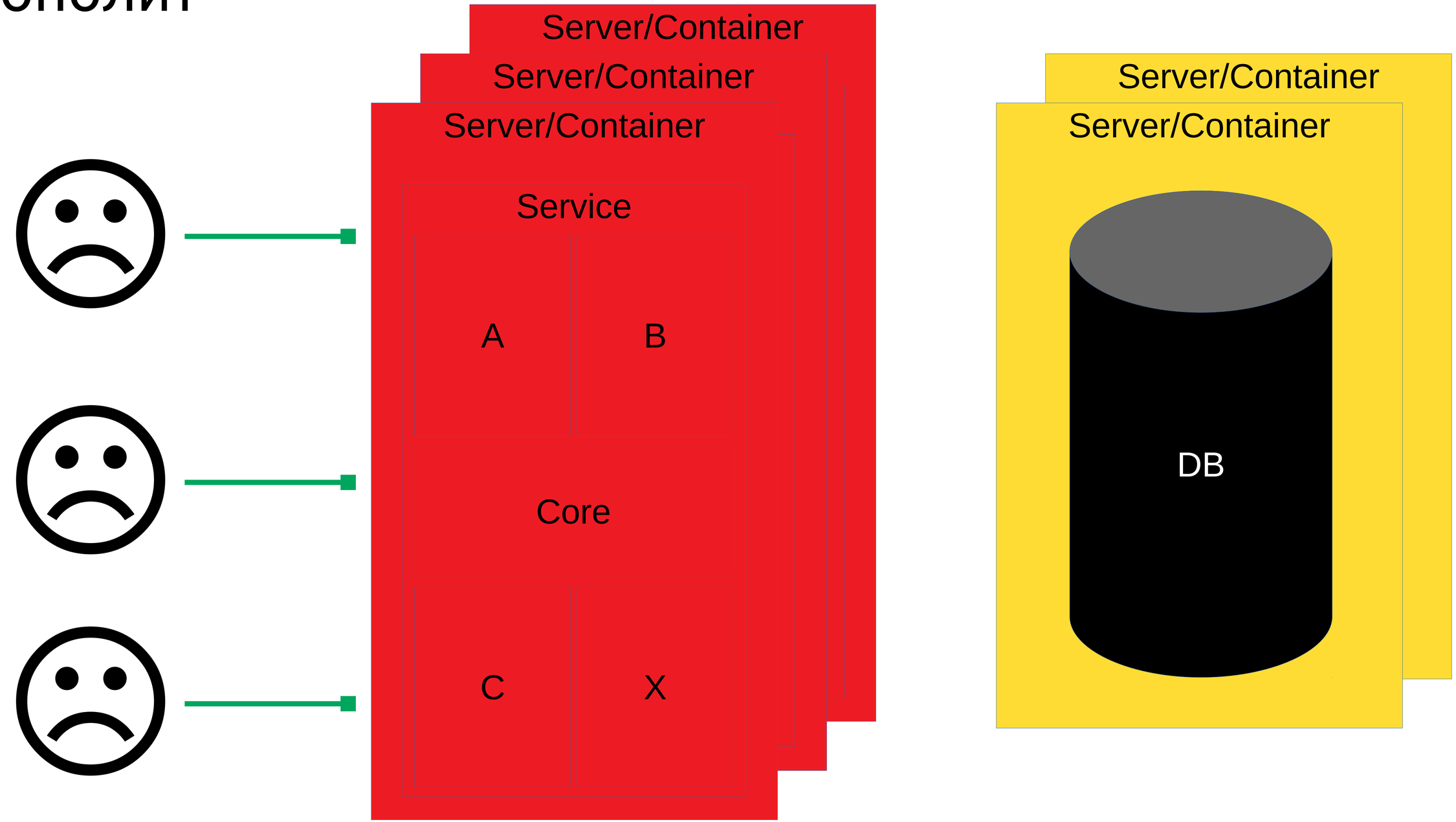

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

Монолит


Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X

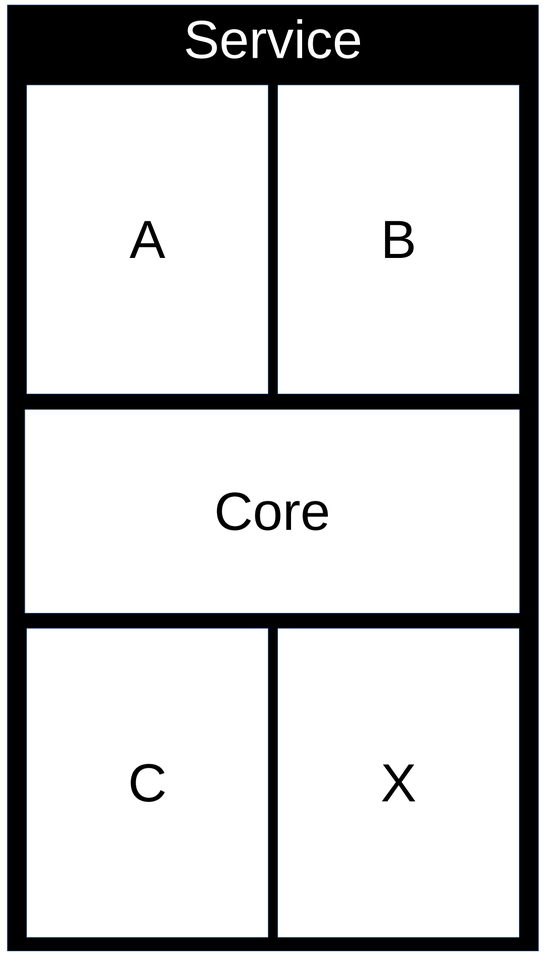

Server/Container
Server/Container
DB

Монолит

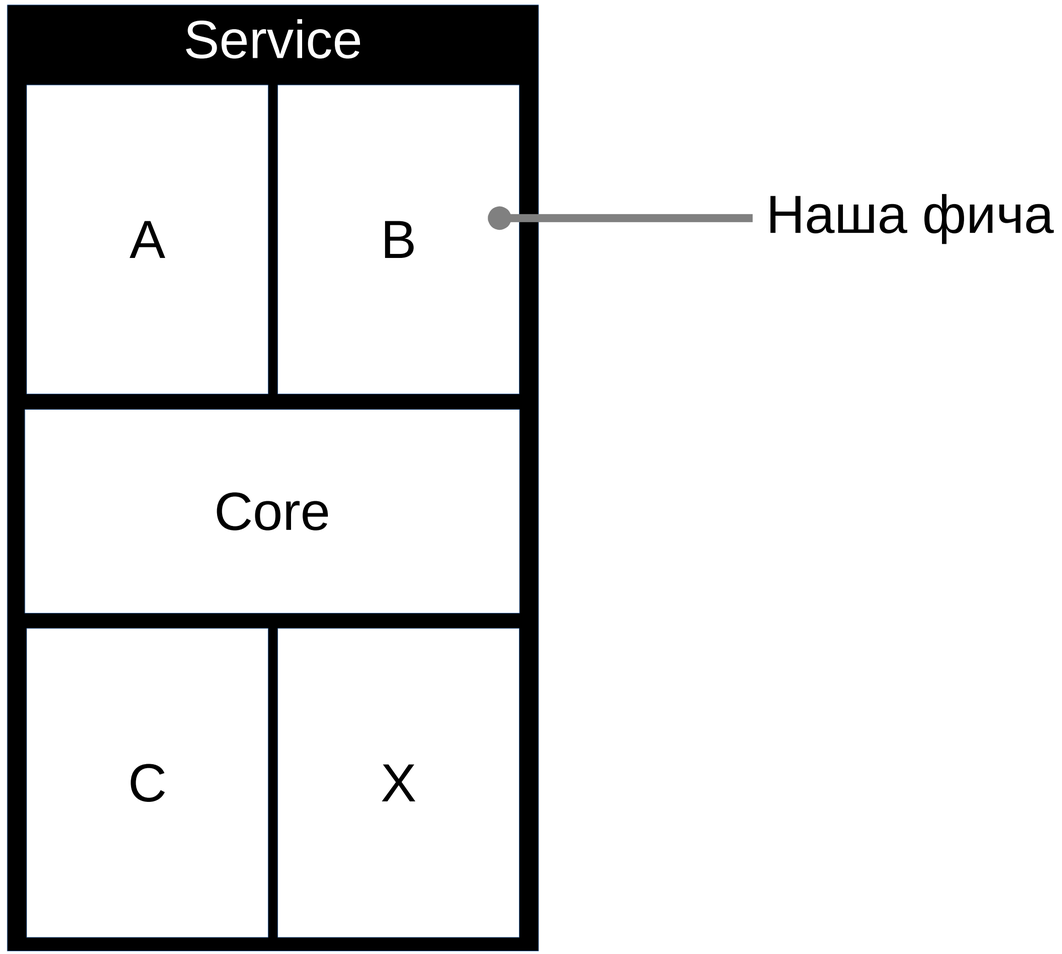

Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

Минусы: время деплоя

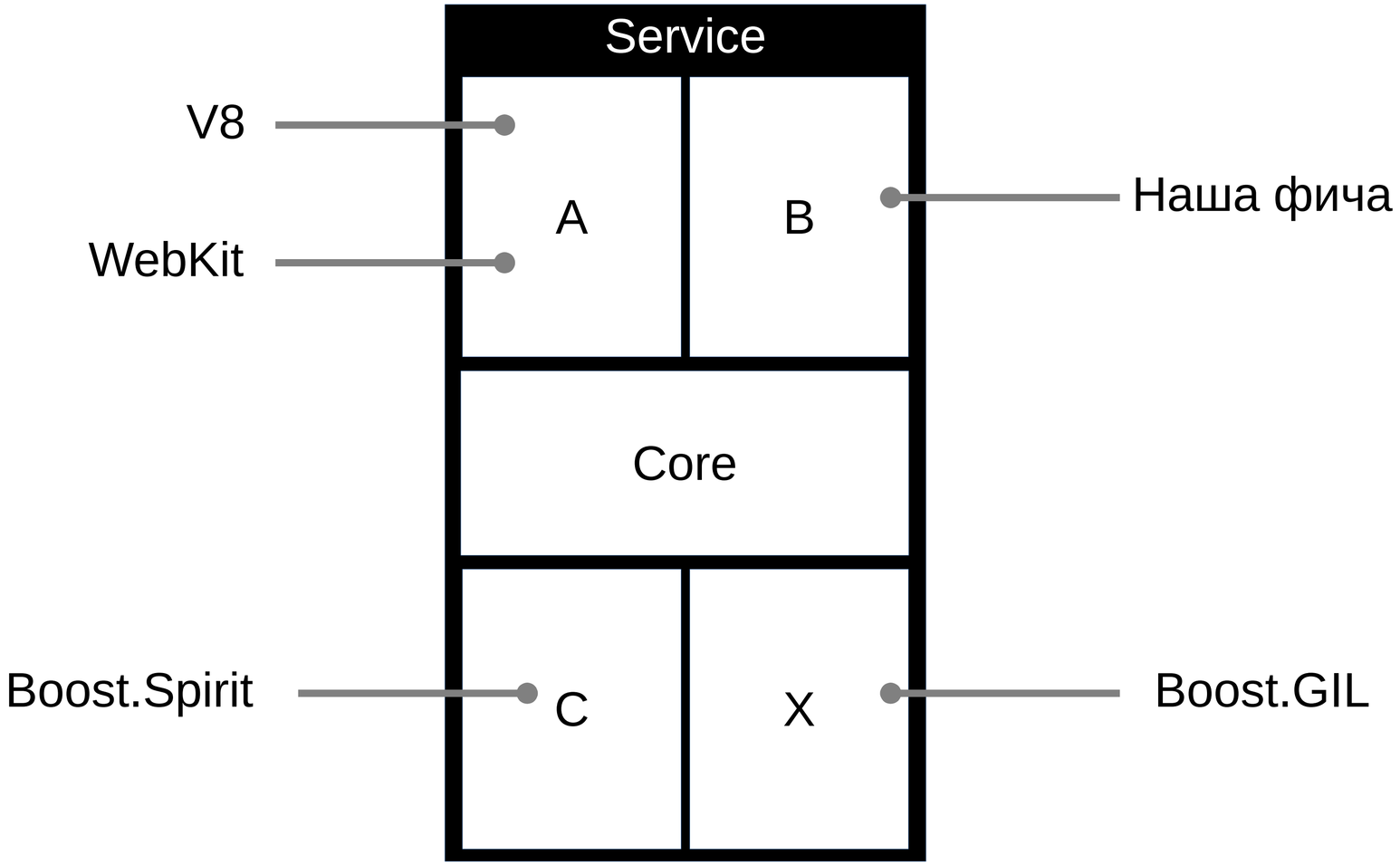
Монолит



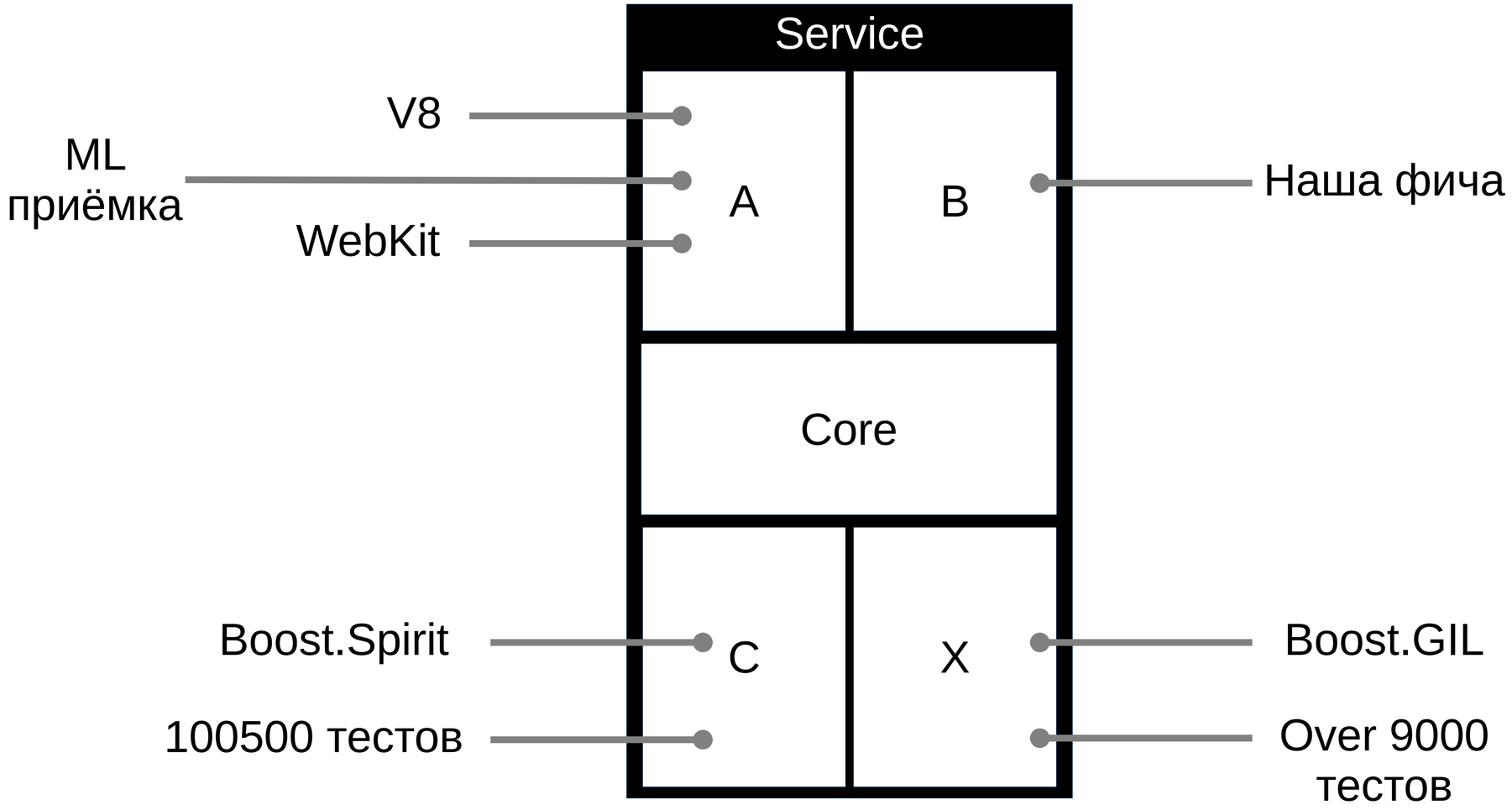
Монолит



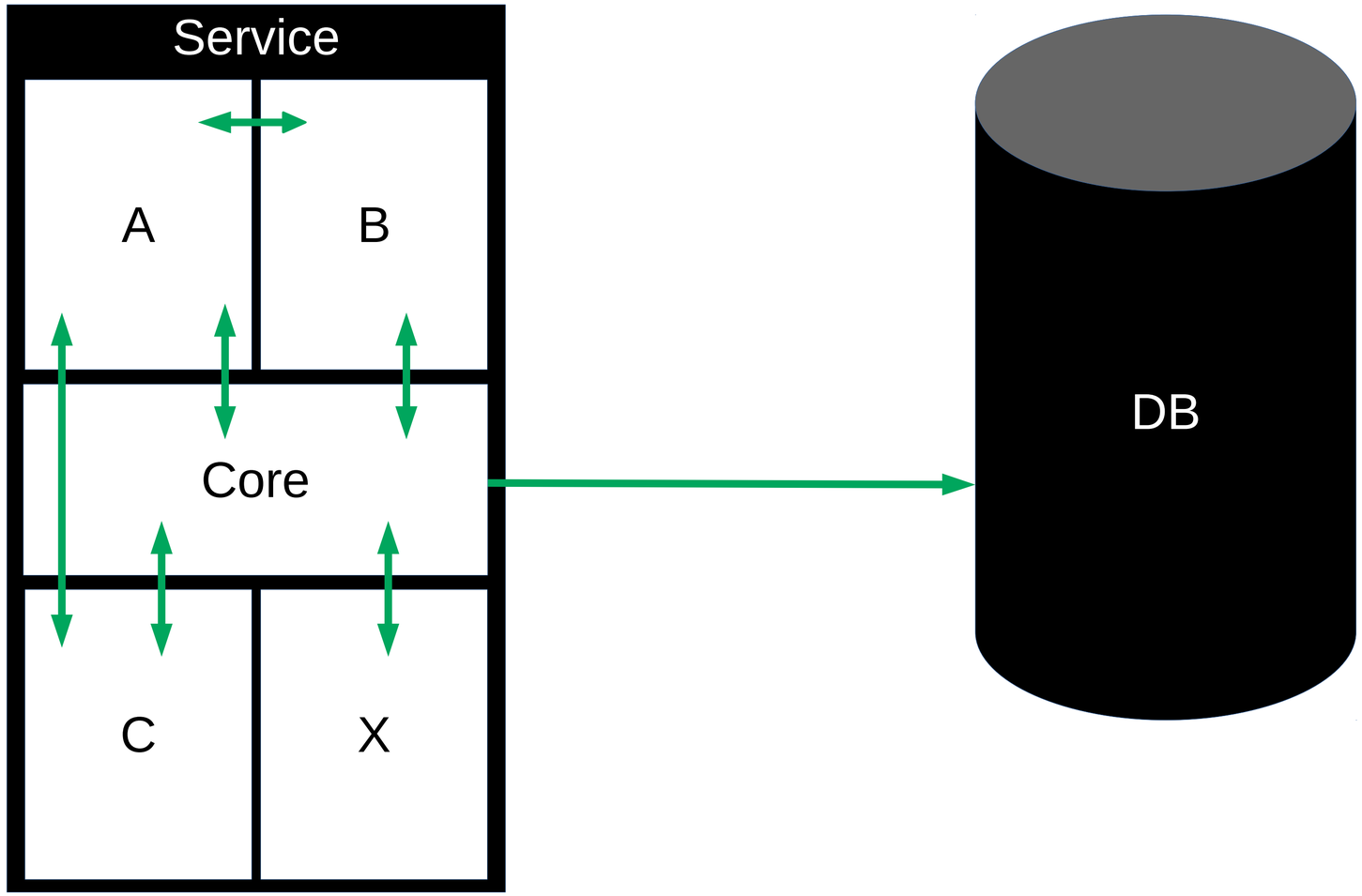
Монолит


Service
V8
WebKit
A
B
Наша фича
Core
Boost.Spirit
C
X
Boost.GIL

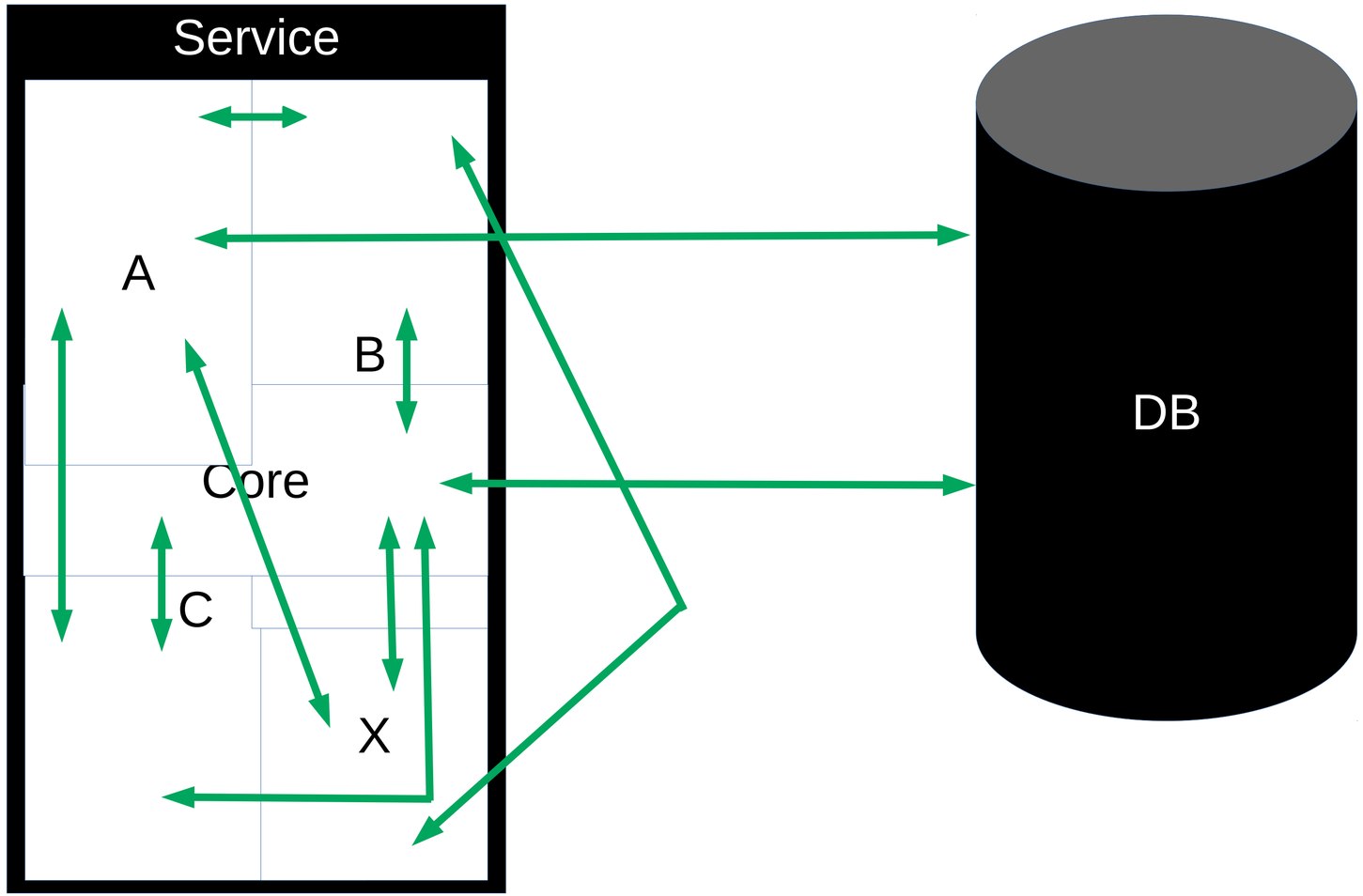
Монолит


Service
V8
ML
приёмка
WebKit
A
B
Наша фича
Core
Boost.Spirit
100500 тестов
C
X
Boost.GIL
Over 9000
тестов

Минусы: тесное взаимодействие


Service
A
B
Core
C
X
DB

Монолит



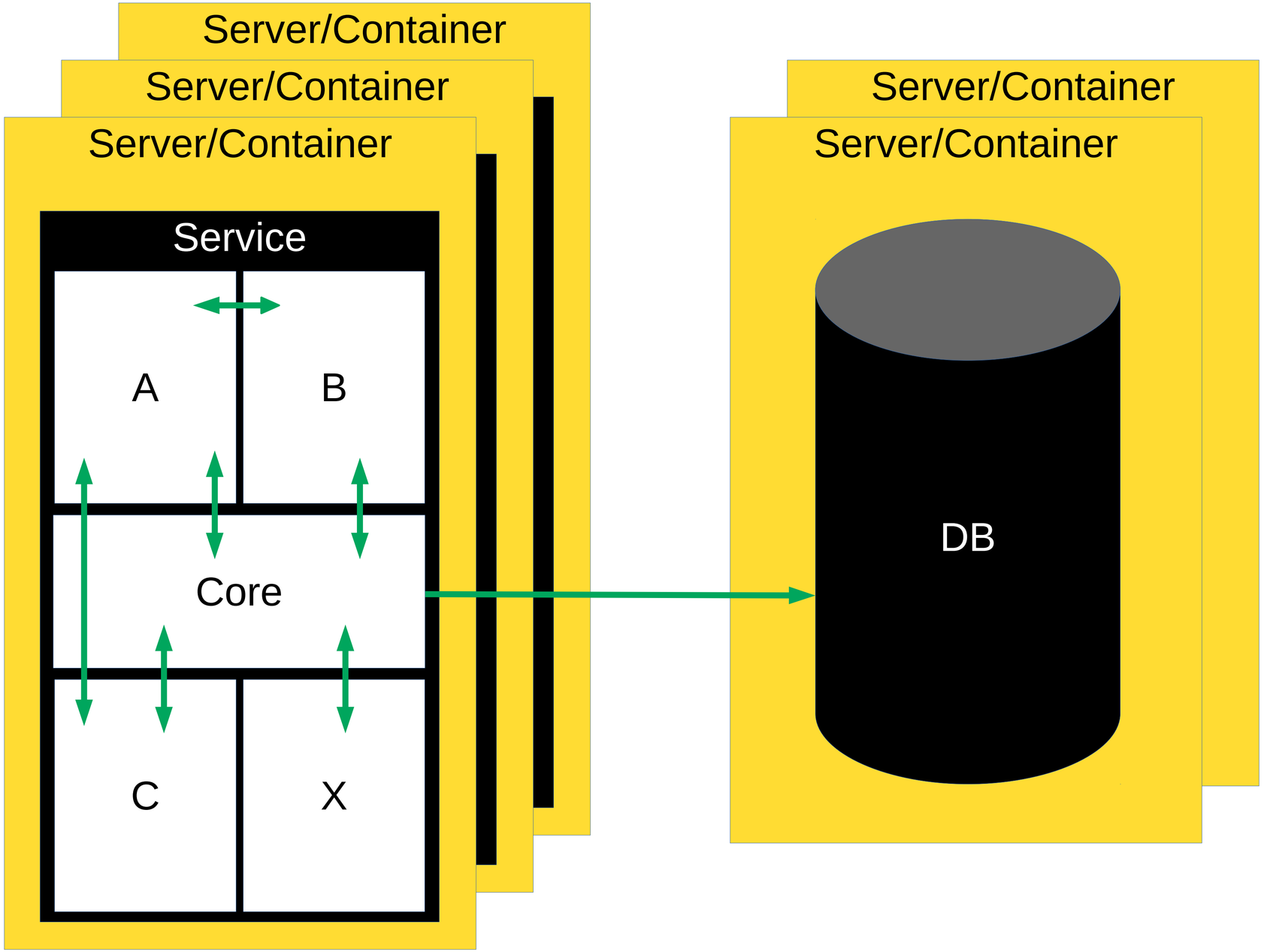
Нужна другая архитектура!

Самая лучшая архитектура это…

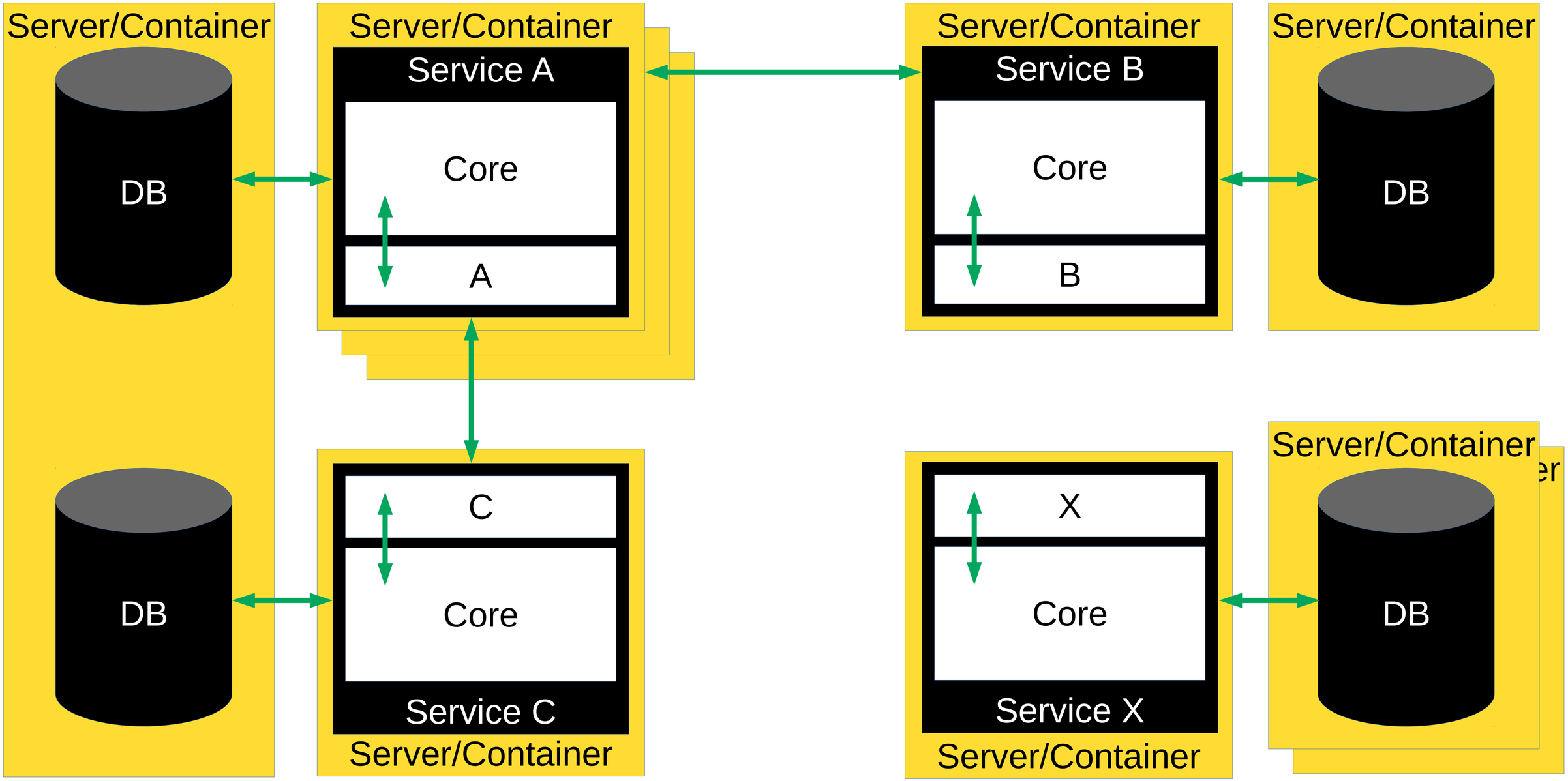
...та, которая ВАМ удобна!

Мы выбрали микросервисы

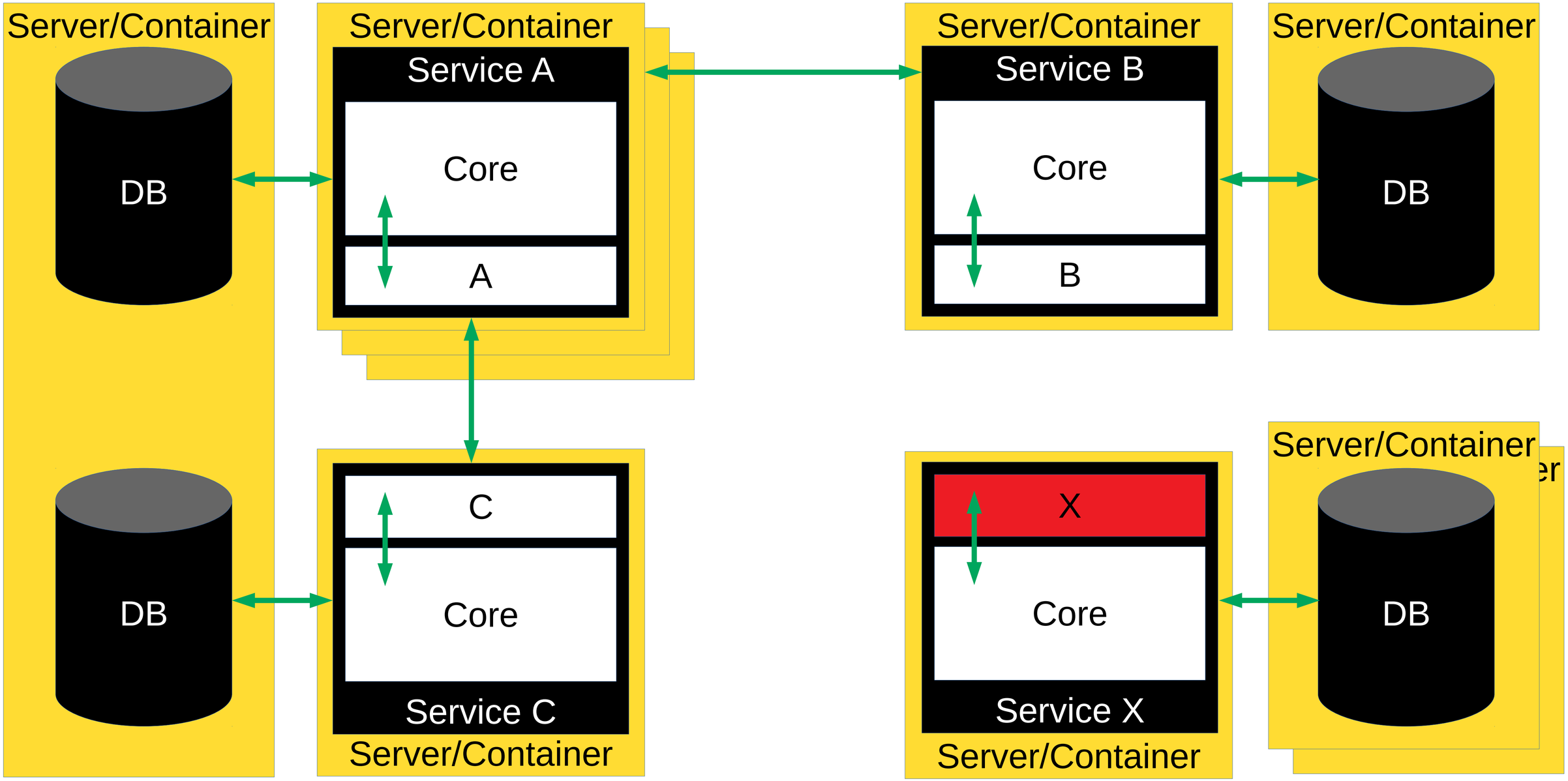
Монолит


Server/Container
Server/Container
Server/Container
Service
A
B
Core
C
X
Server/Container
Server/Container
DB

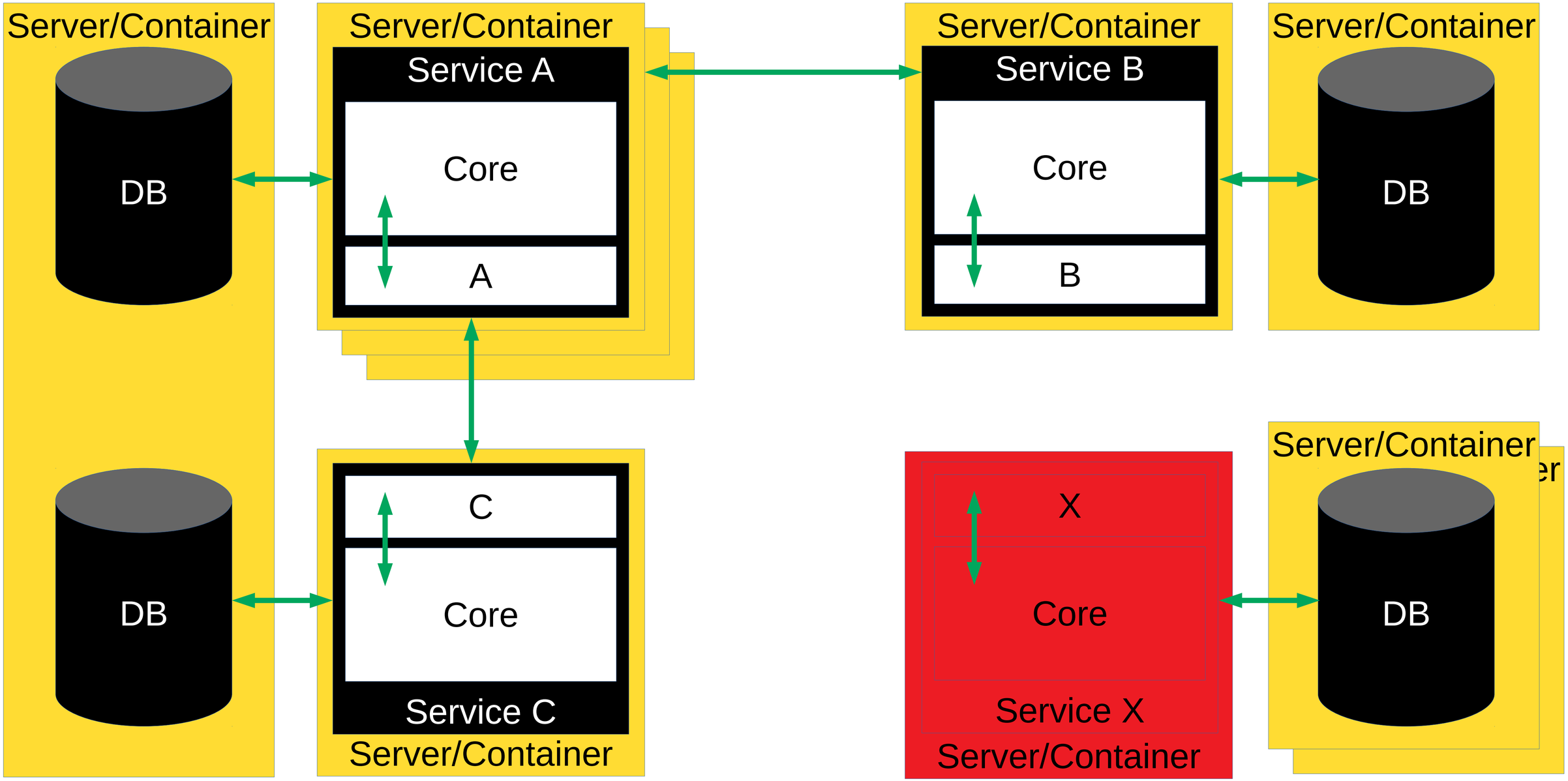
Микросервисы



Микросервисы



Микросервисы



Выбор фреймворка

Требования к фреймворку

Требования к фреймворку

- Возможность переиспользовать имеющийся C++ код

Требования к фреймворку

- Возможность переиспользовать имеющийся C++ код
  - и C++ разработчиков ☺

Требования к фреймворку

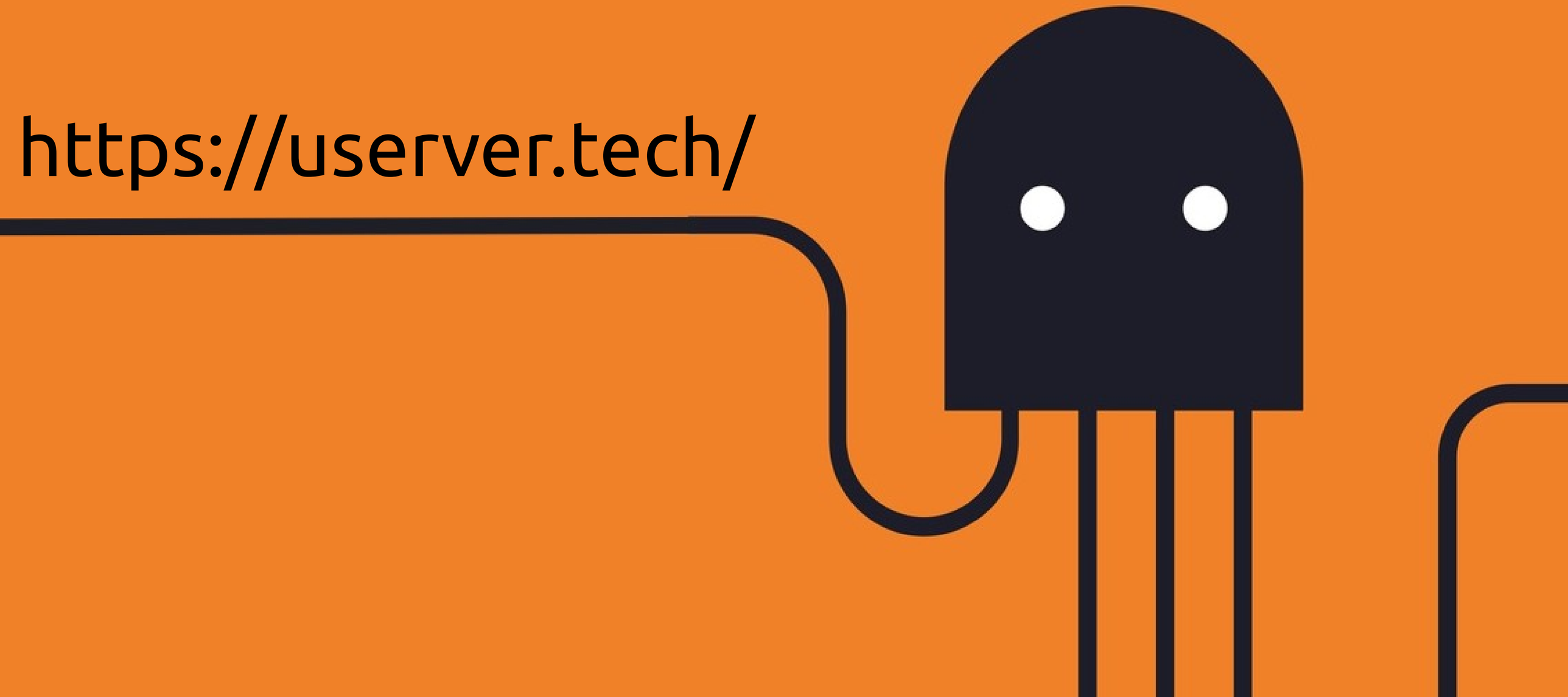
- Возможность переиспользовать имеющийся C++ код
  - и C++ разработчиков 😊
- Ориентация на IO-bound приложения

Требования к фреймворку

- Возможность переиспользовать имеющийся C++ код
  - и C++ разработчиков 😊
- Ориентация на IO-bound приложения
- Простота использования

Так начался userver

https://userver.tech/



IO-bound приложения и простота использования

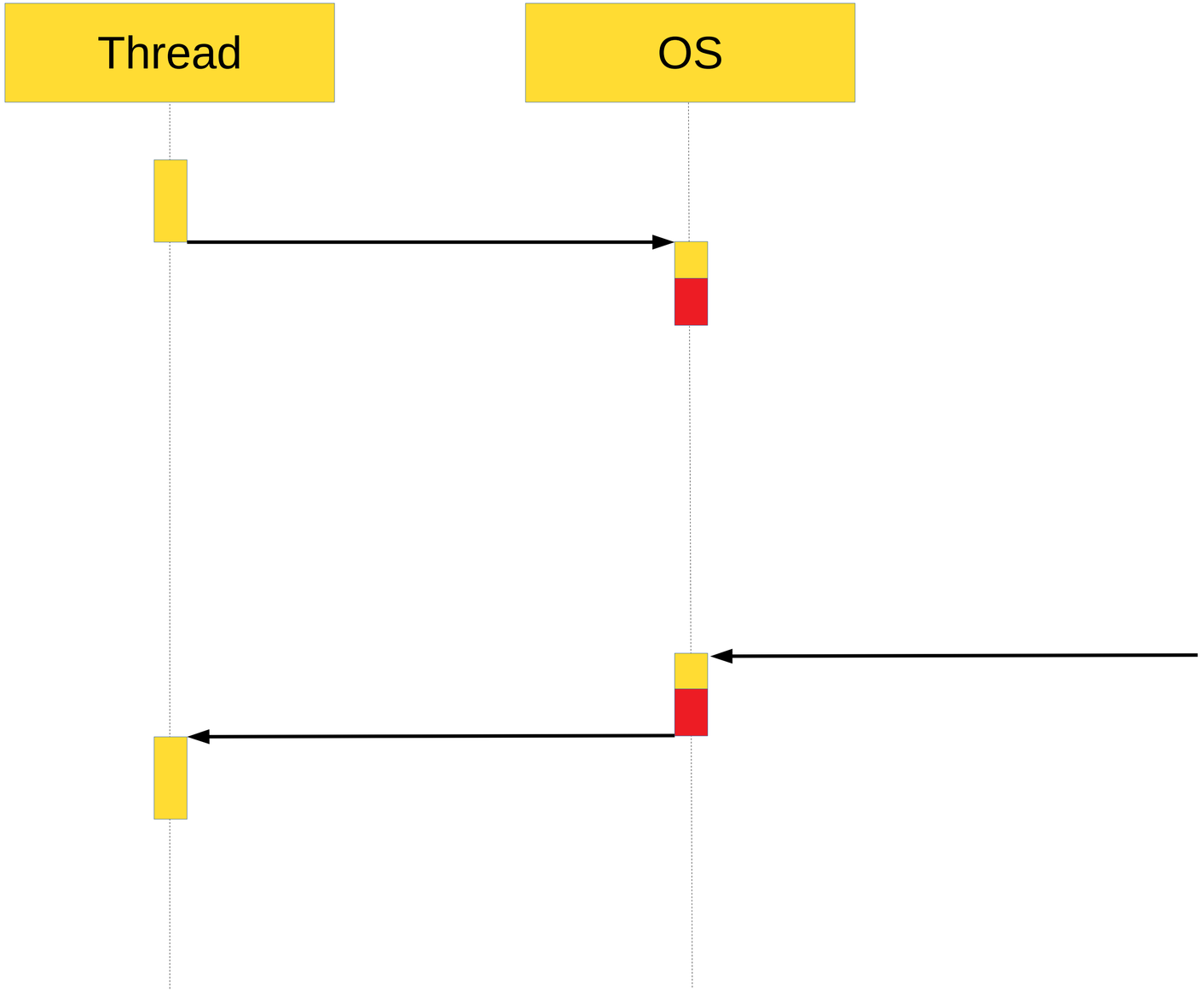
userver

```
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>()};
}
```

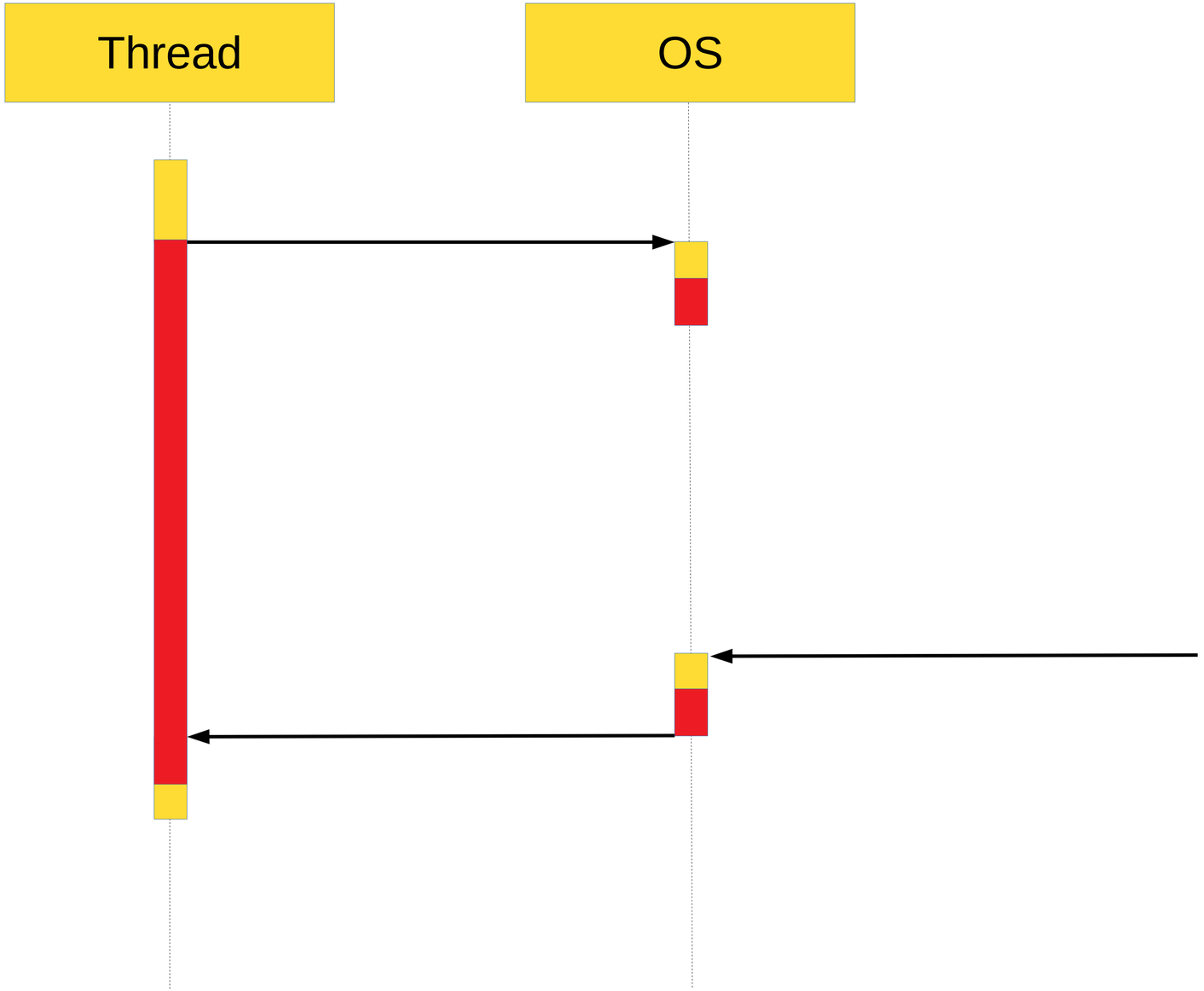
userver

```
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>()};
}
```

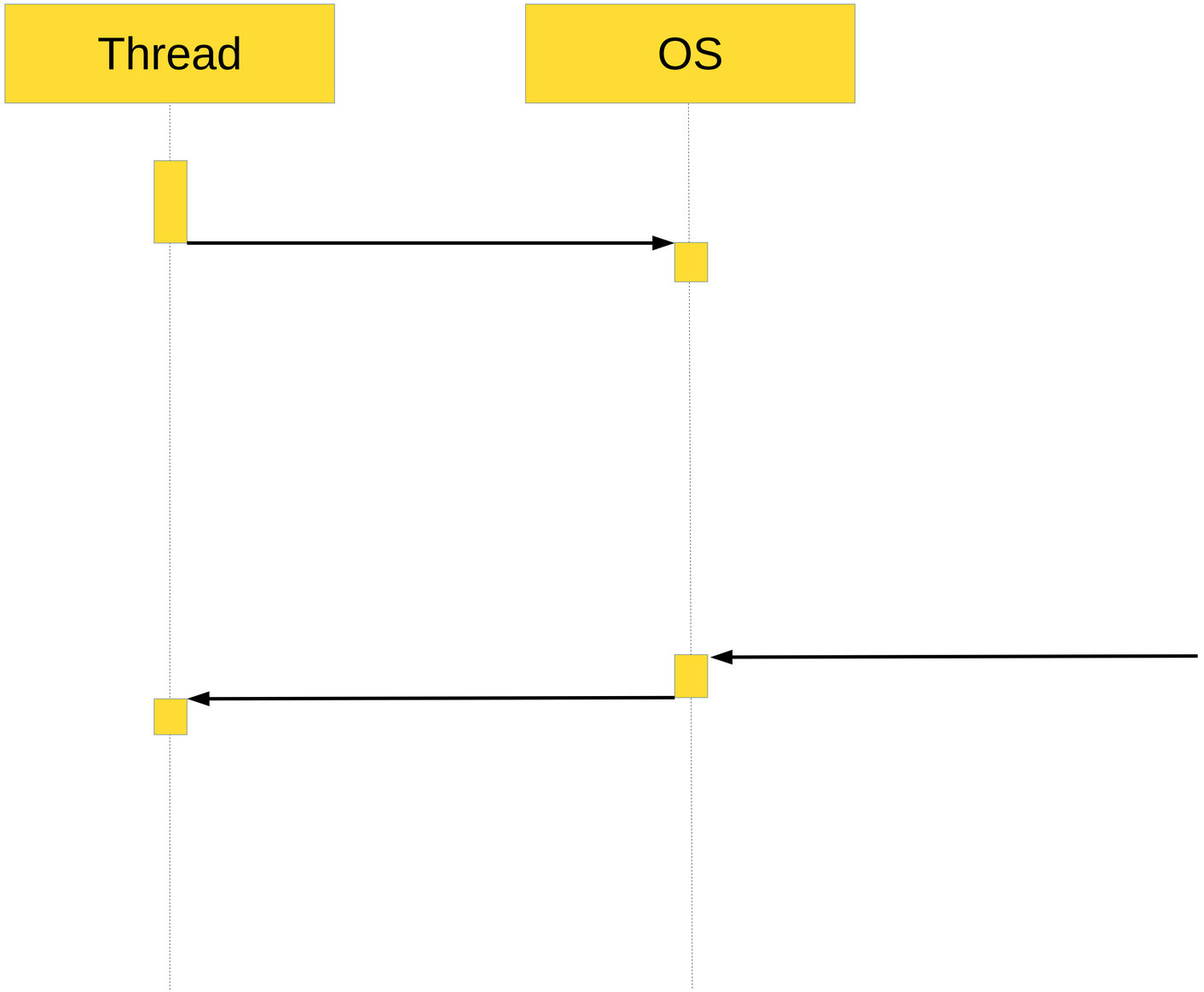
Синхронный подход (не userver!)



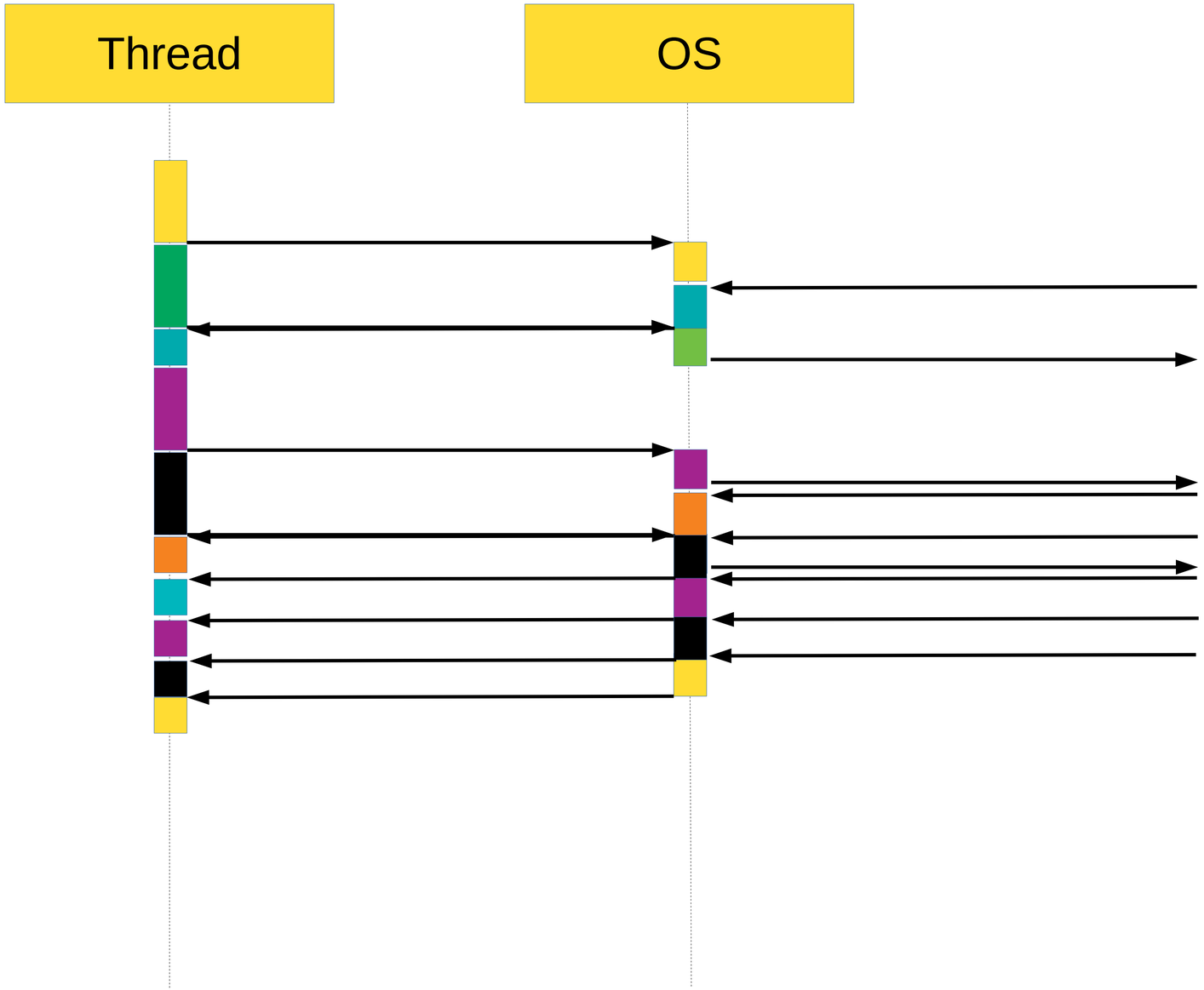
Синхронный подход (не userver!)



userver



userver



userver

```
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>()};
}
```

HE userver

```
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;
              });
```

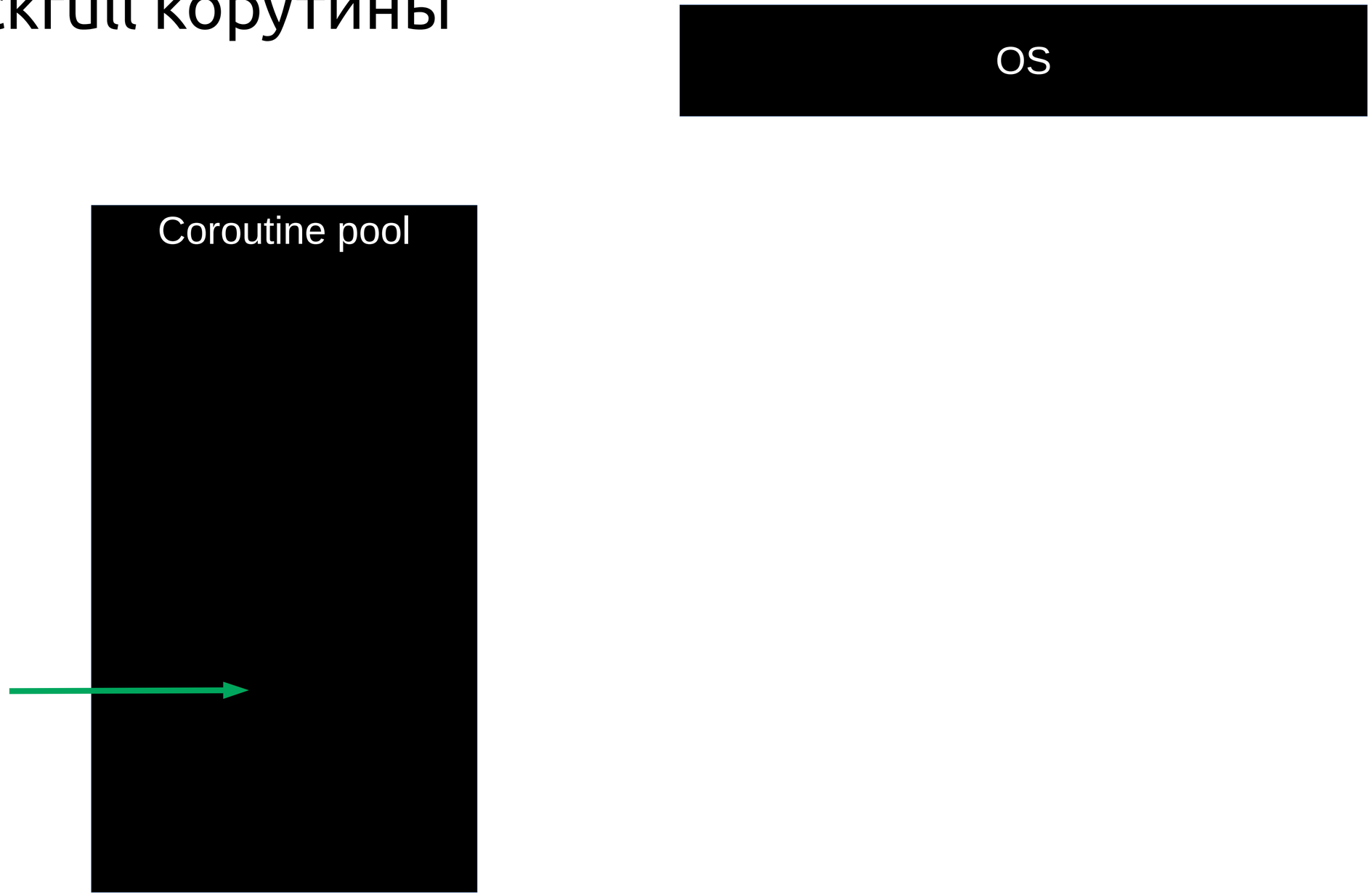
HE userver

```
            }
            *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>()};
            });
          });
        });
      });
    });
  }
```

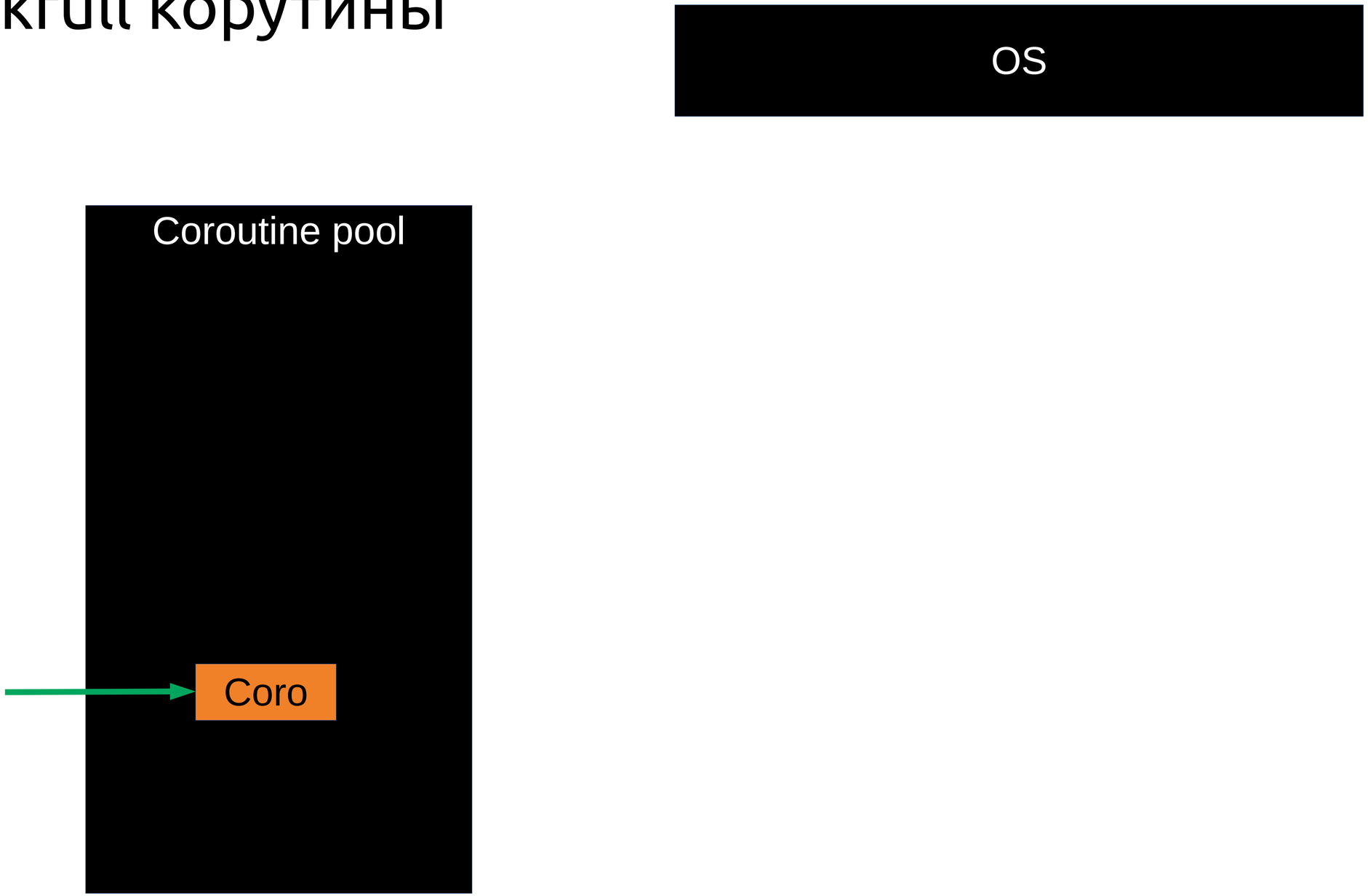
Как это сделать?

stackfull корутины

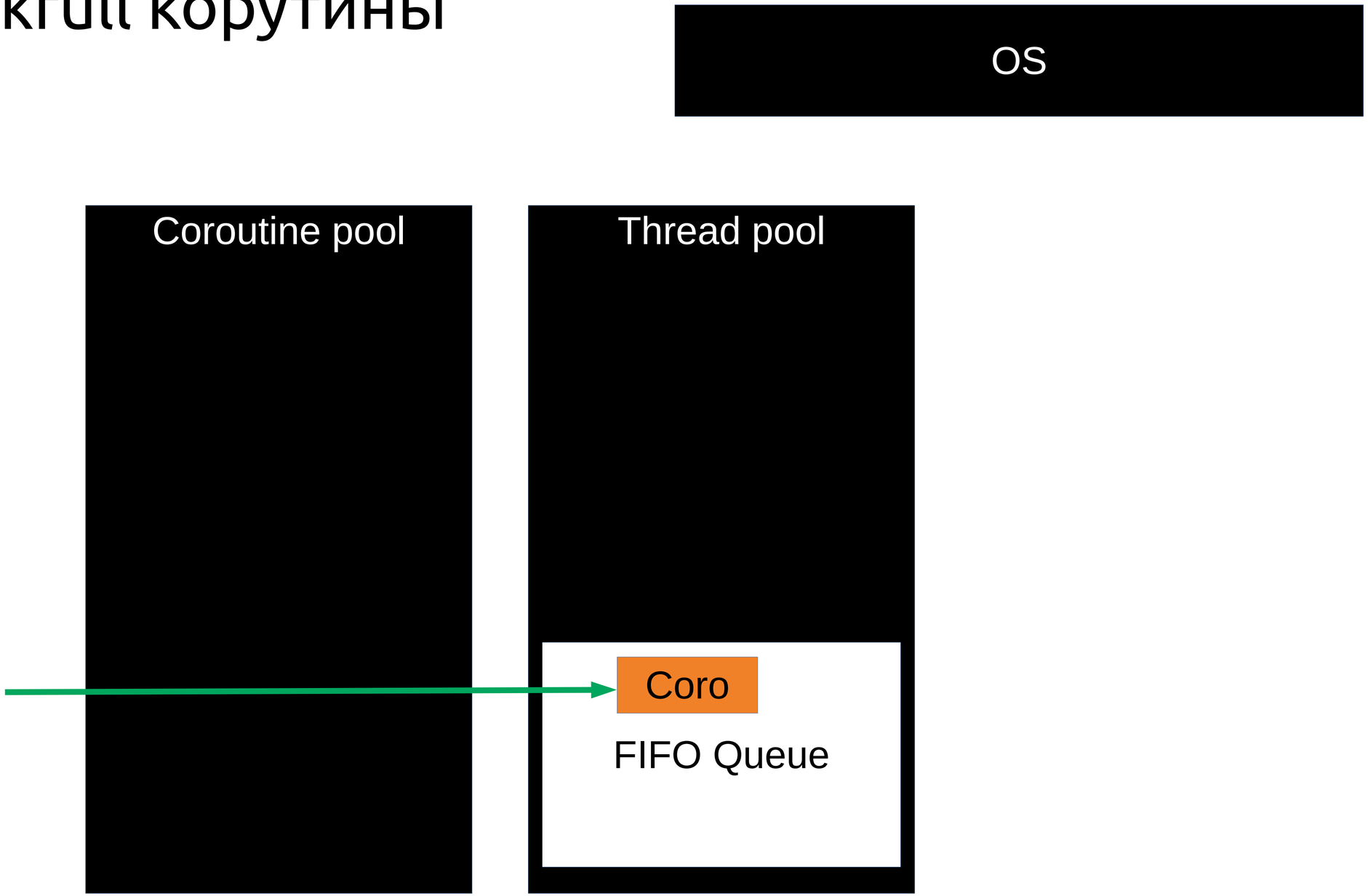
stackfull корутины



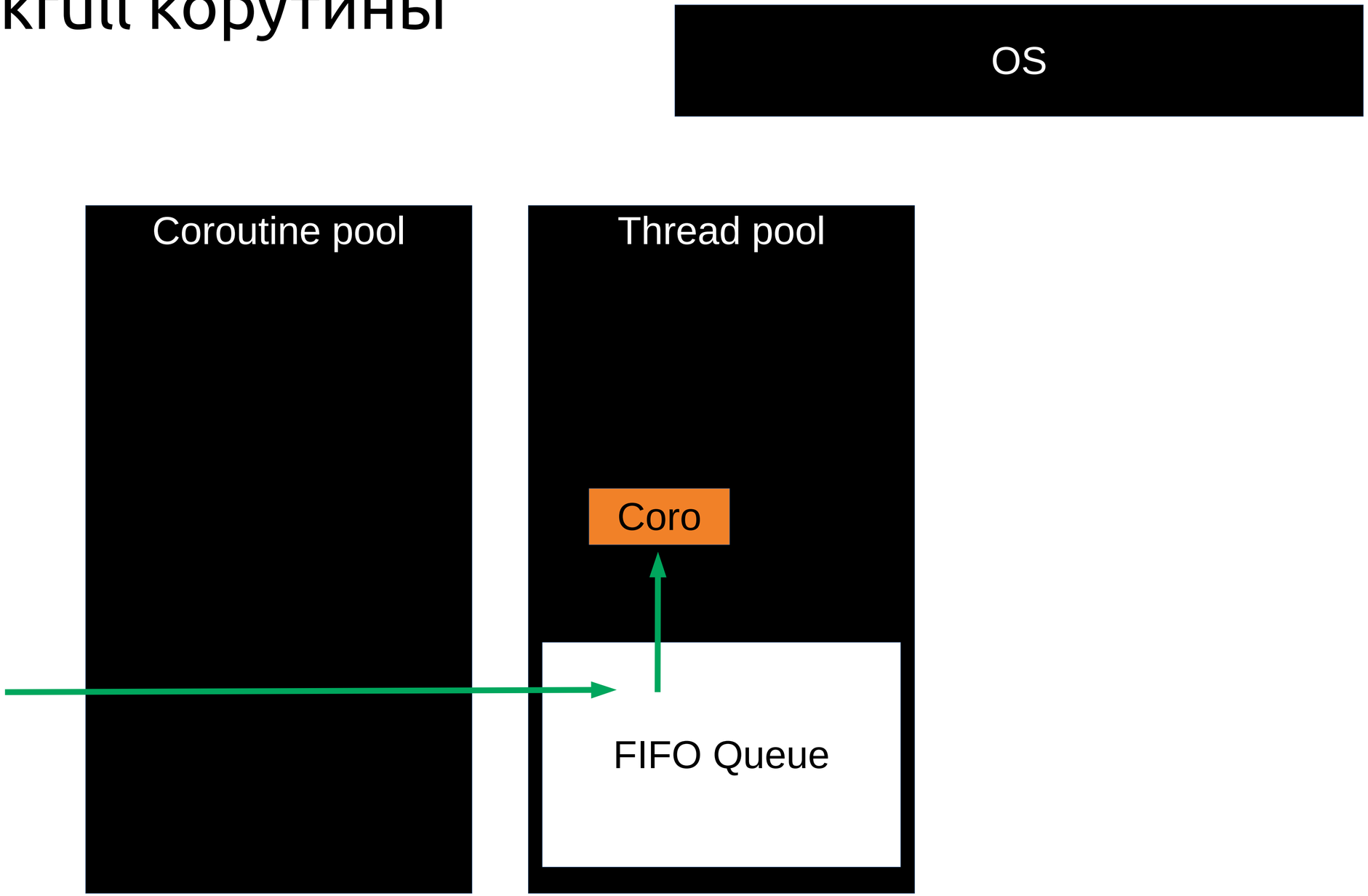
stackfull корутины



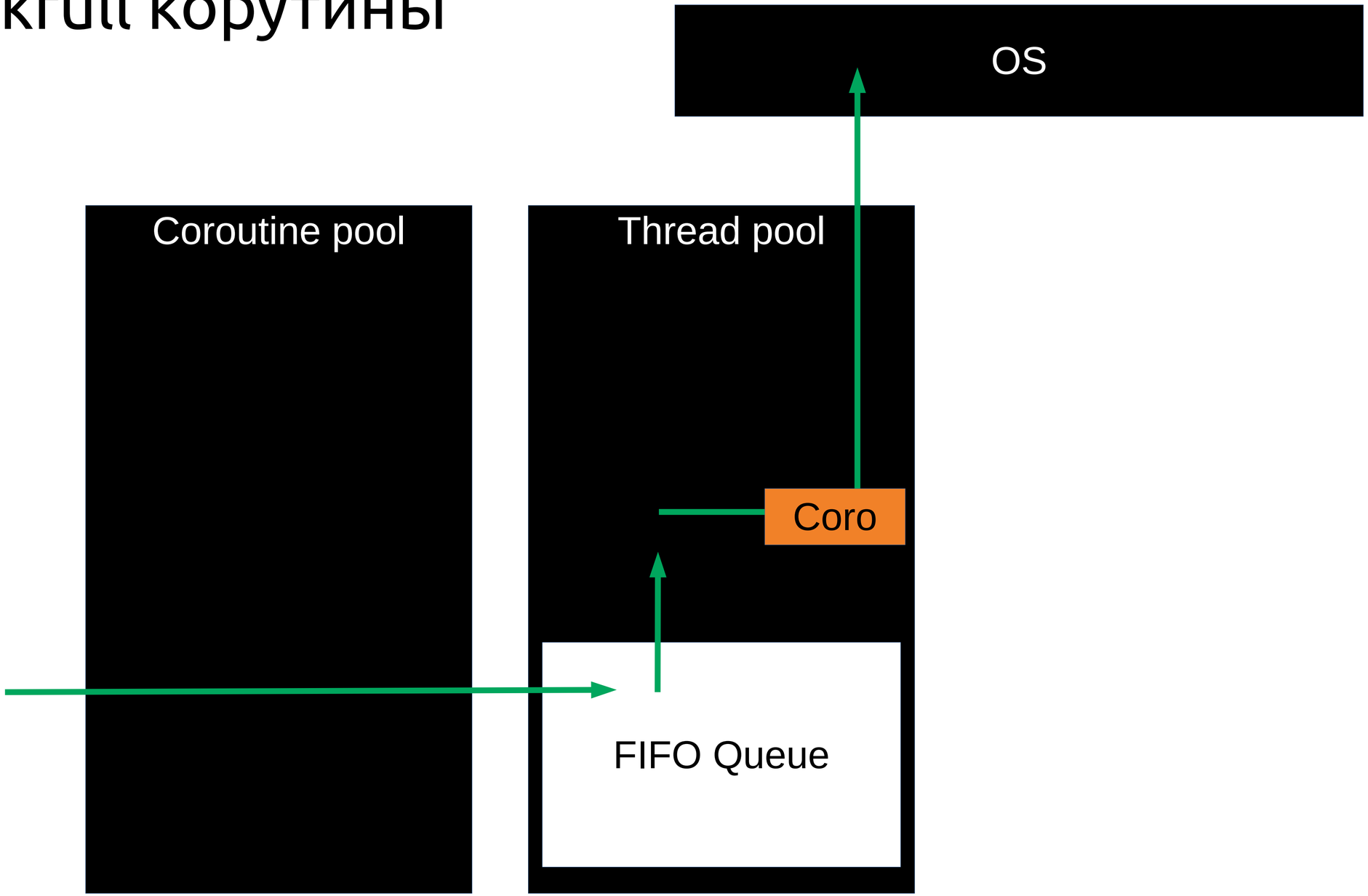
stackfull корутины



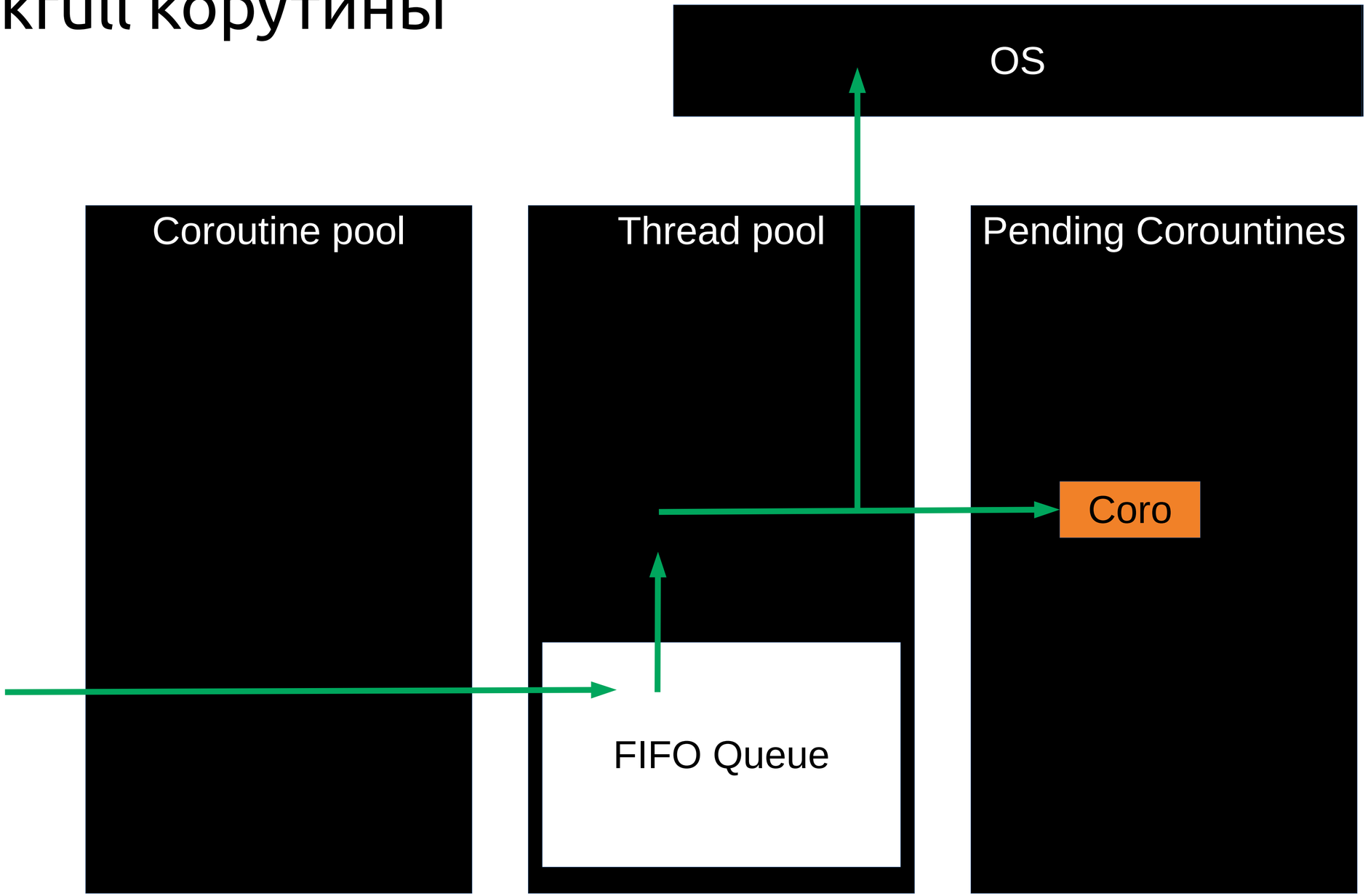
stackfull корутины



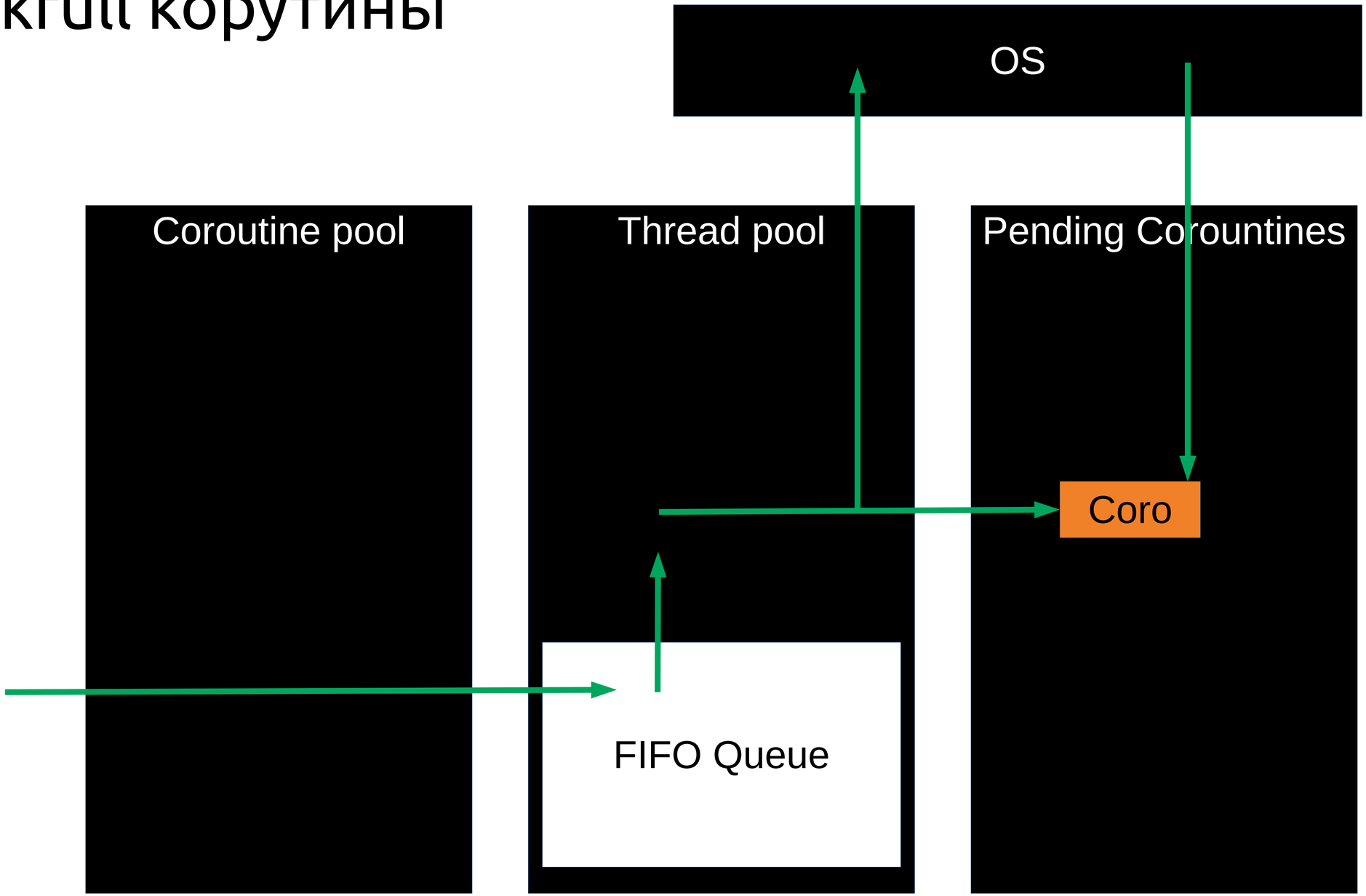
stackfull корутины



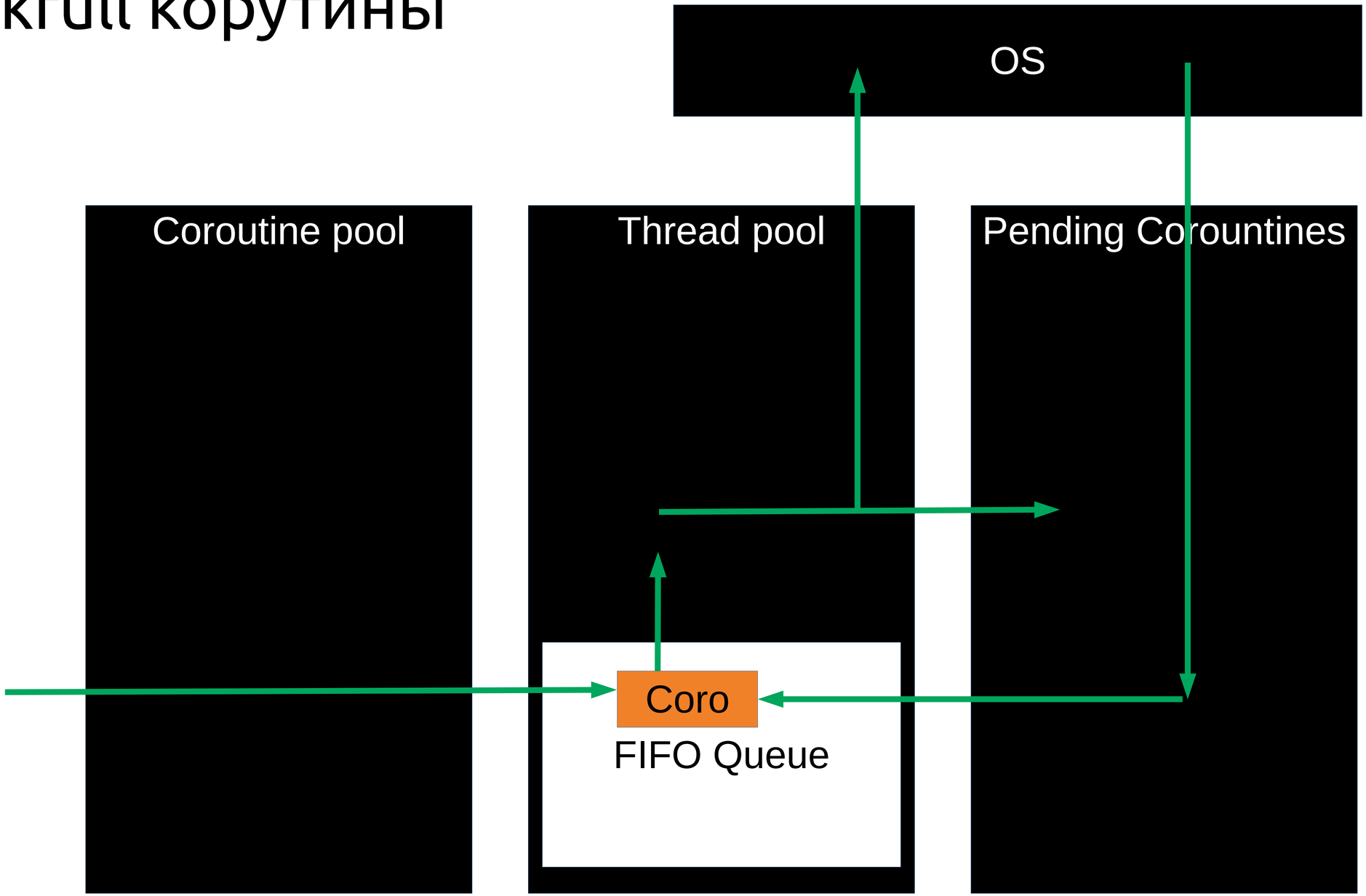
stackfull корутины



stackfull корутины



stackfull корутины



userver

```
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>()};
}
```

userver

```
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>()};
}
```

userver

```
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>()};
}
```

userver

```
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>()};
}
```

userver

```
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>()};
}
```

userver

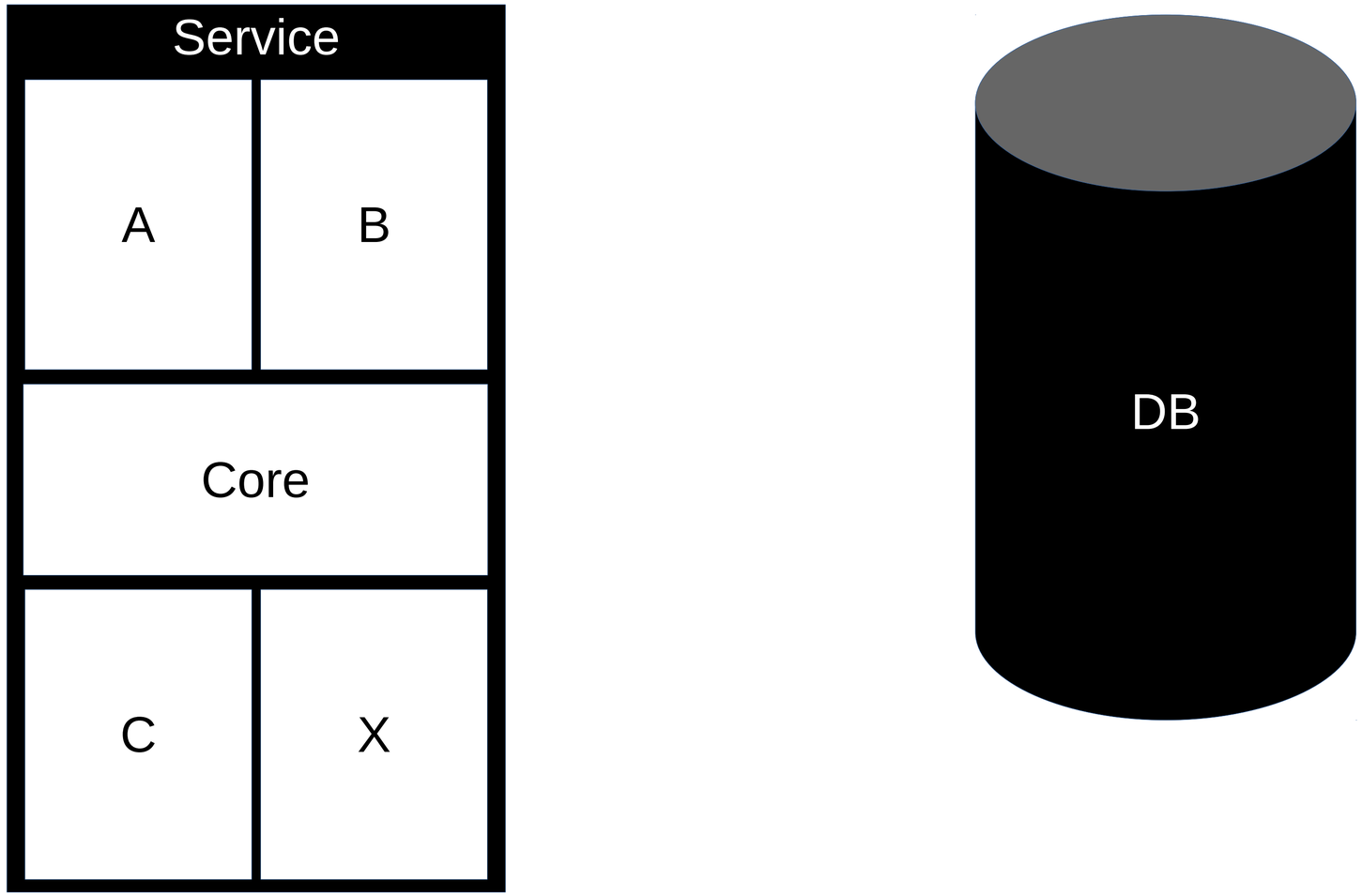
```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
  auto cluster = dependencies.pg->GetCluster();                          // ⚡
  auto trx = cluster->Begin(storages::postgres::ClusterHostType::kMaster);  // ⚡

  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>()};
}
```

userver

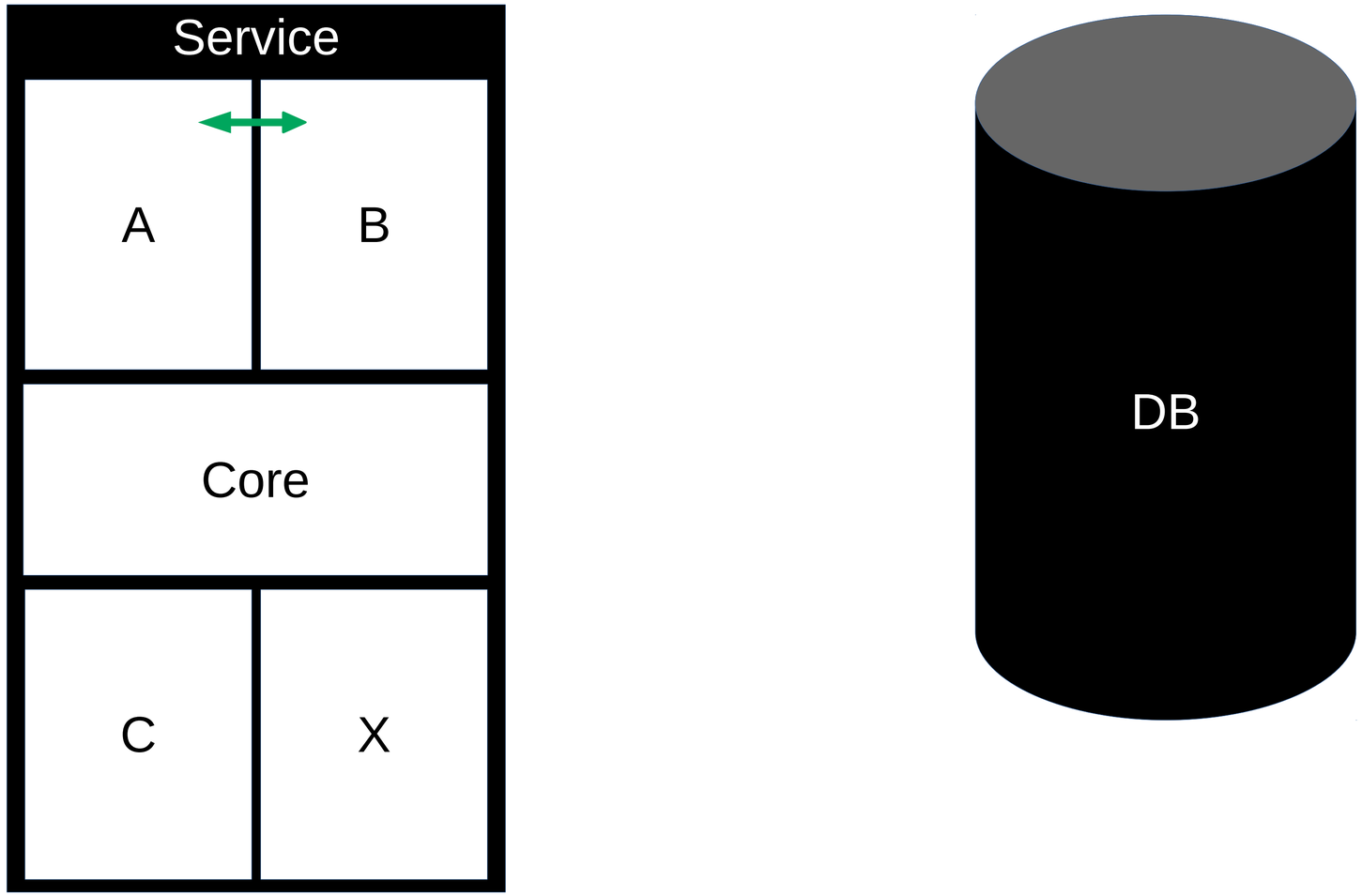
```
Response View::Handle(Request&& request, const Dependencies& dependencies) {
  auto cluster = dependencies.pg->GetCluster();
  auto trx = cluster->Begin(storages::postgres::ClusterHostType::kMaster);

  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>()};
}
```

Минусы микросервисов: latency

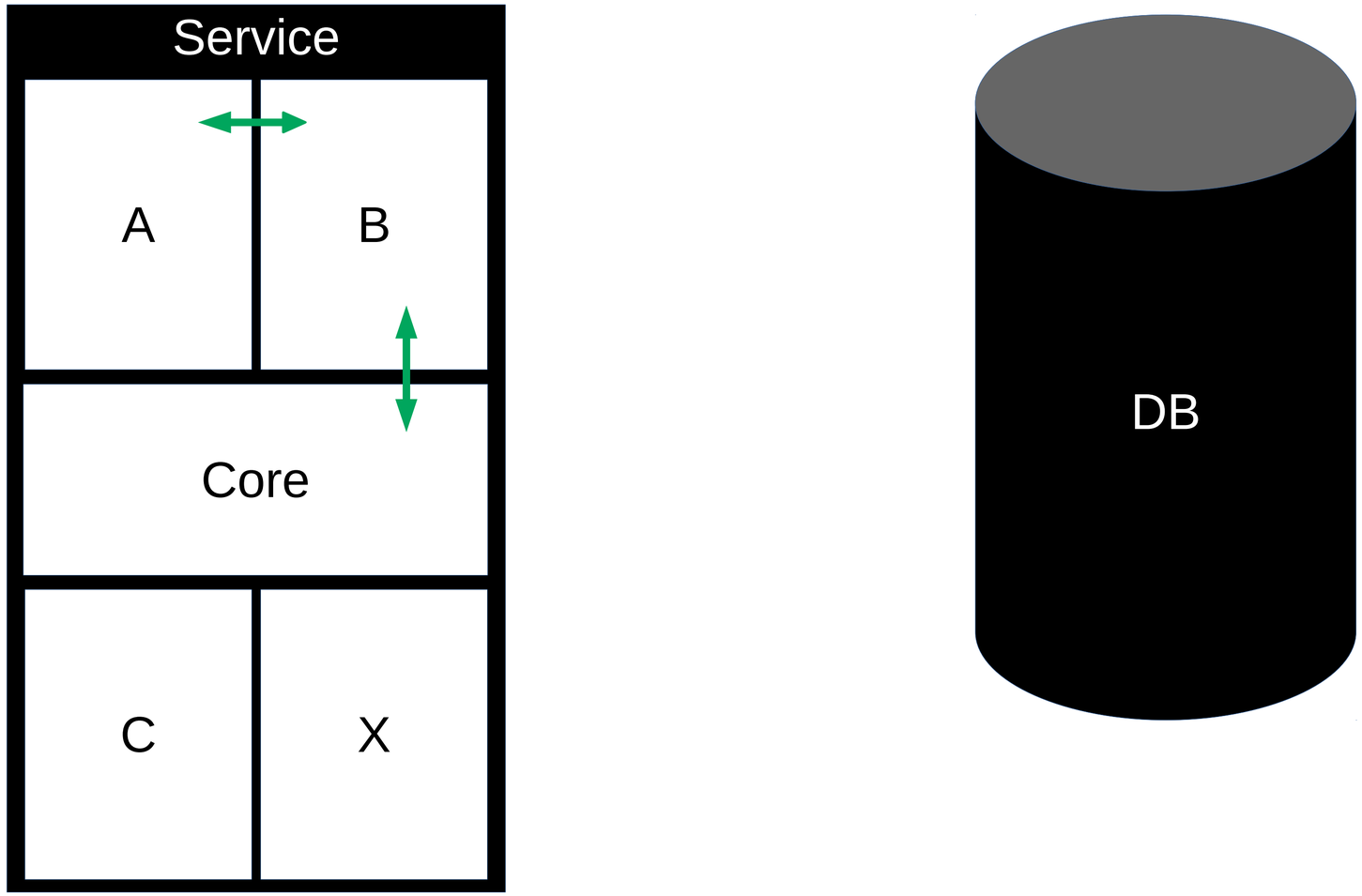
Взаимодействия в монолите



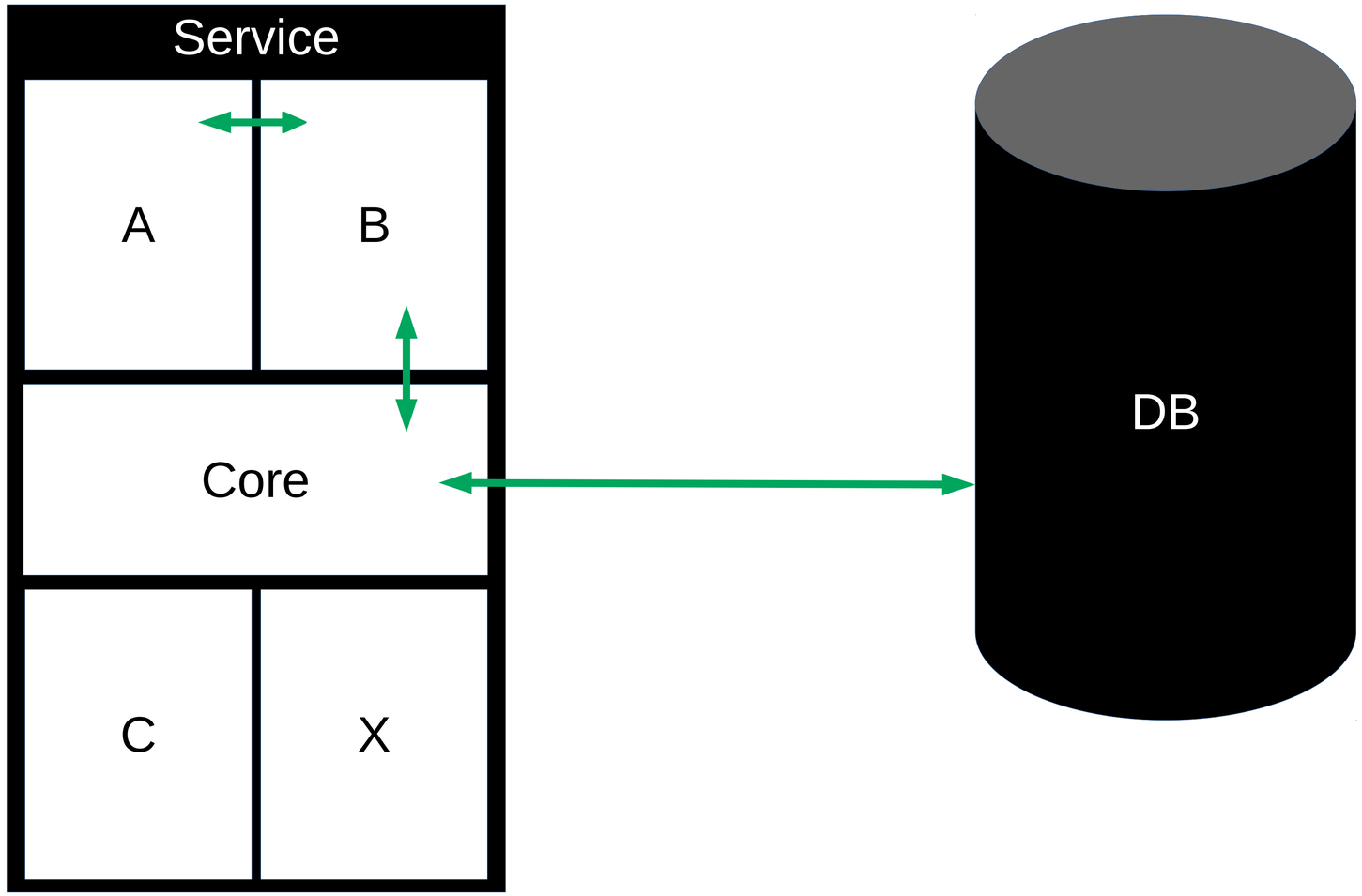
Взаимодействия в монолите



Взаимодействия в монолите



Взаимодействия в монолите

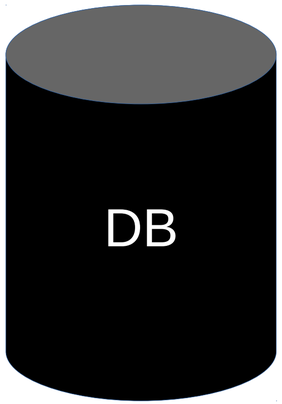


Запрос

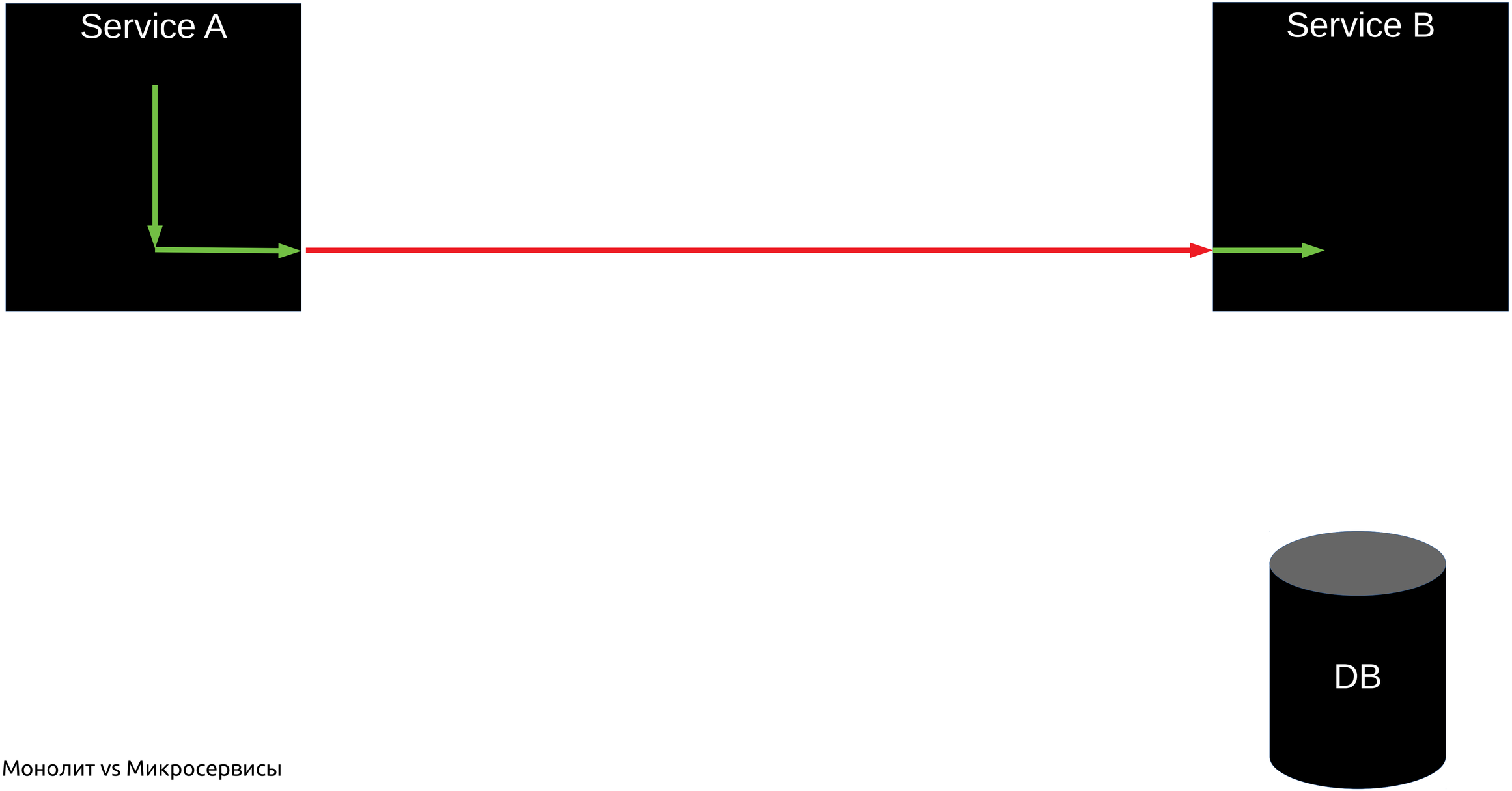
Запрос

Запрос

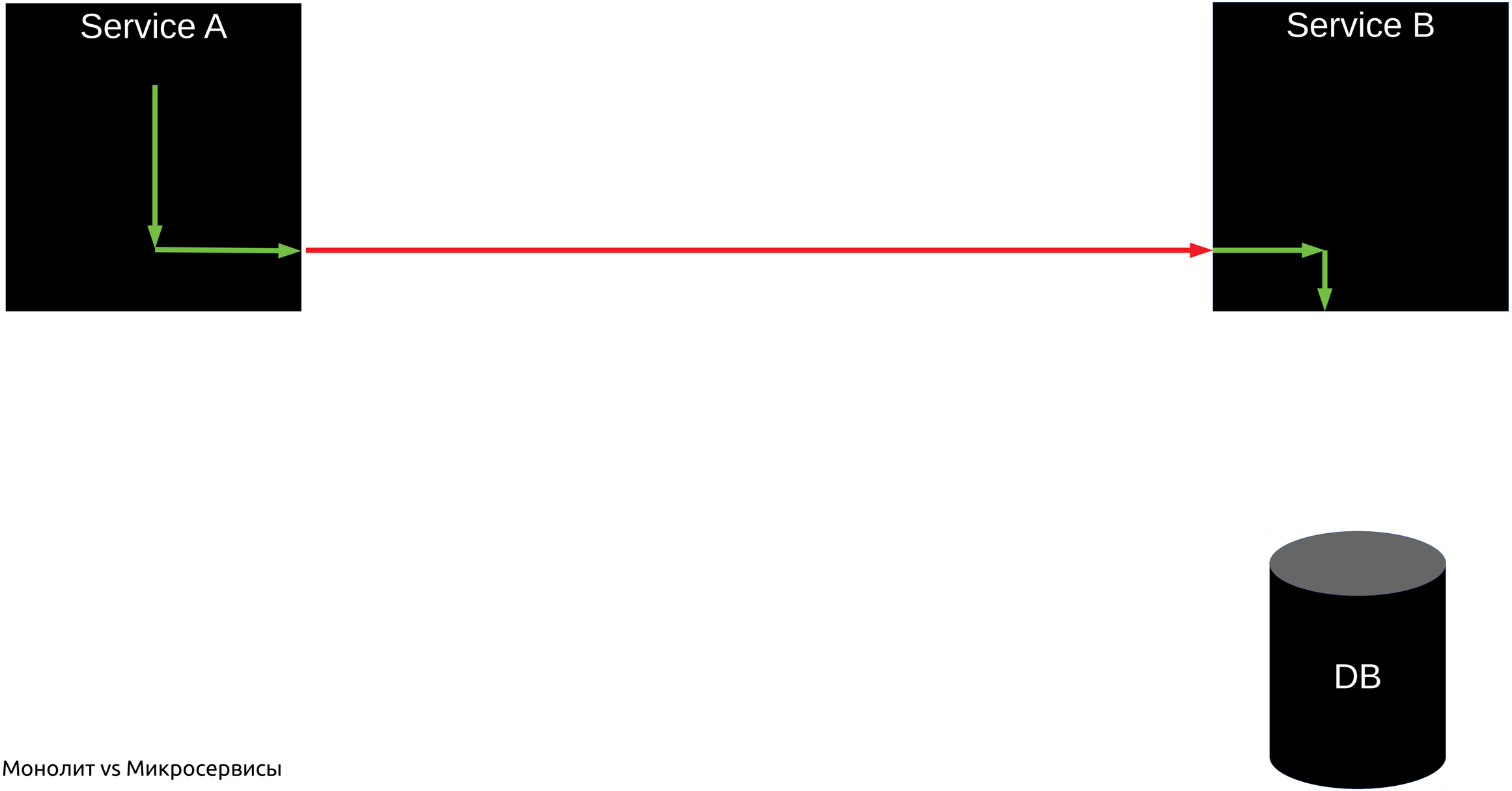
Запрос



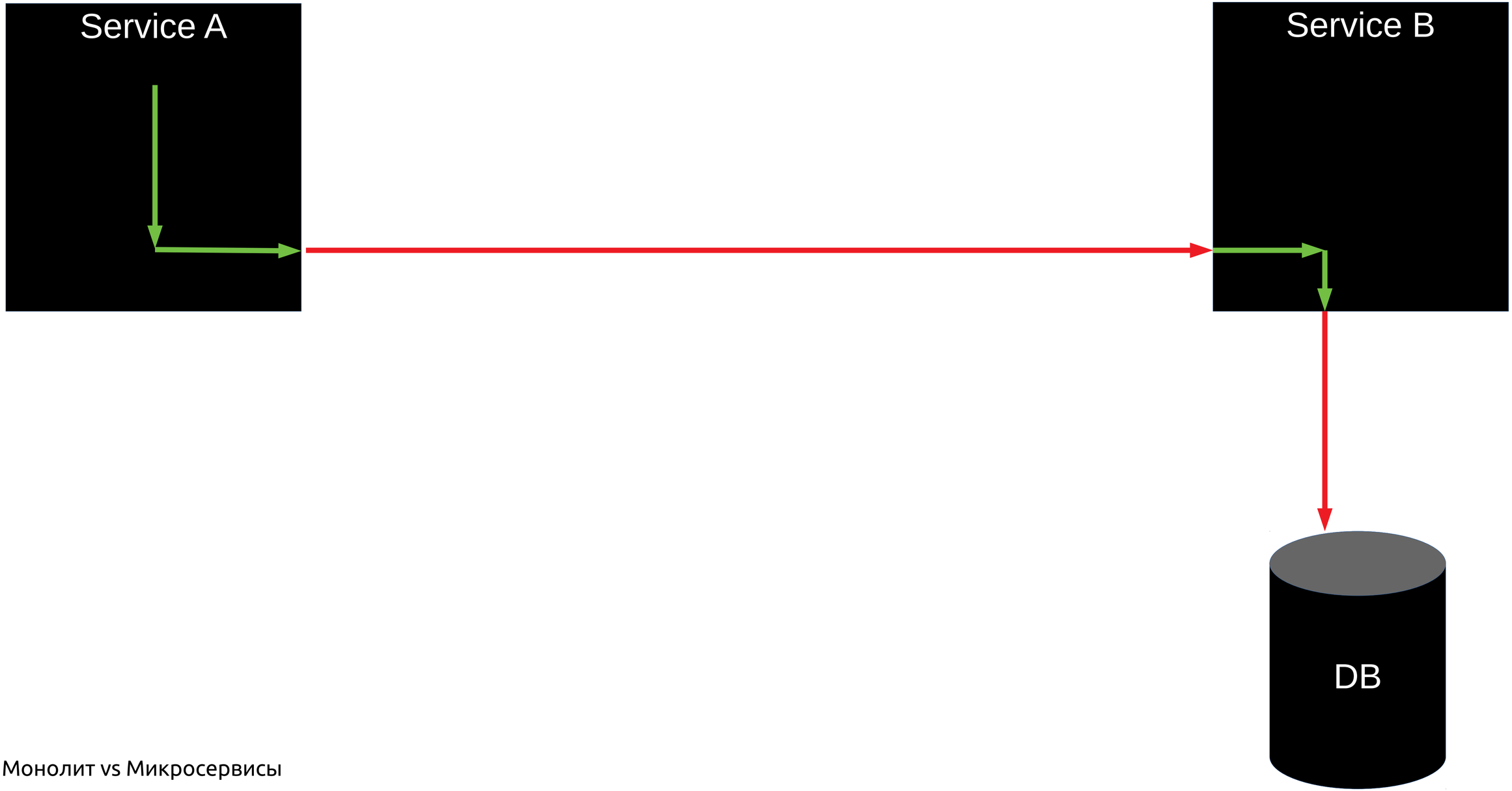
Запрос



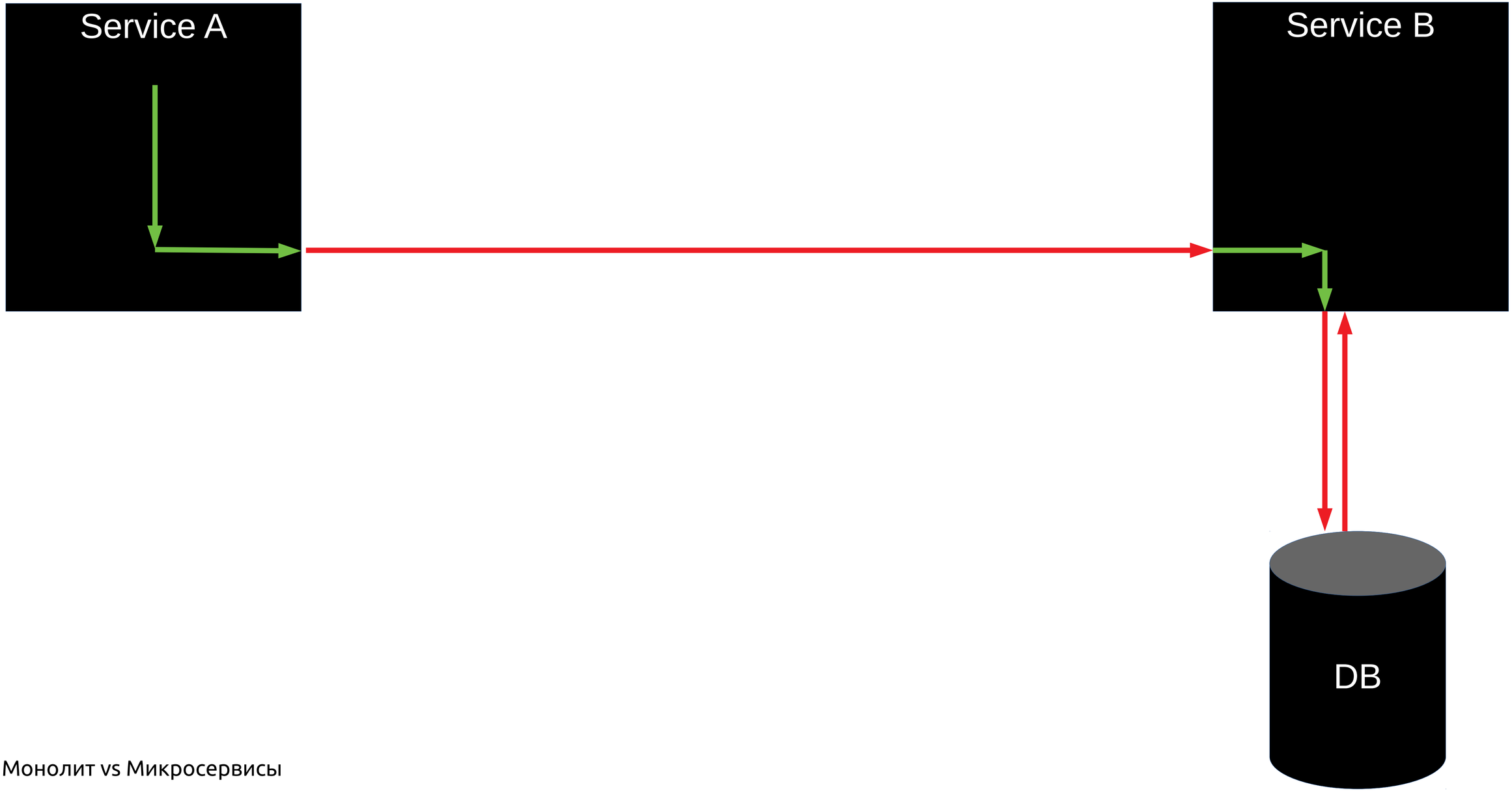
Запрос



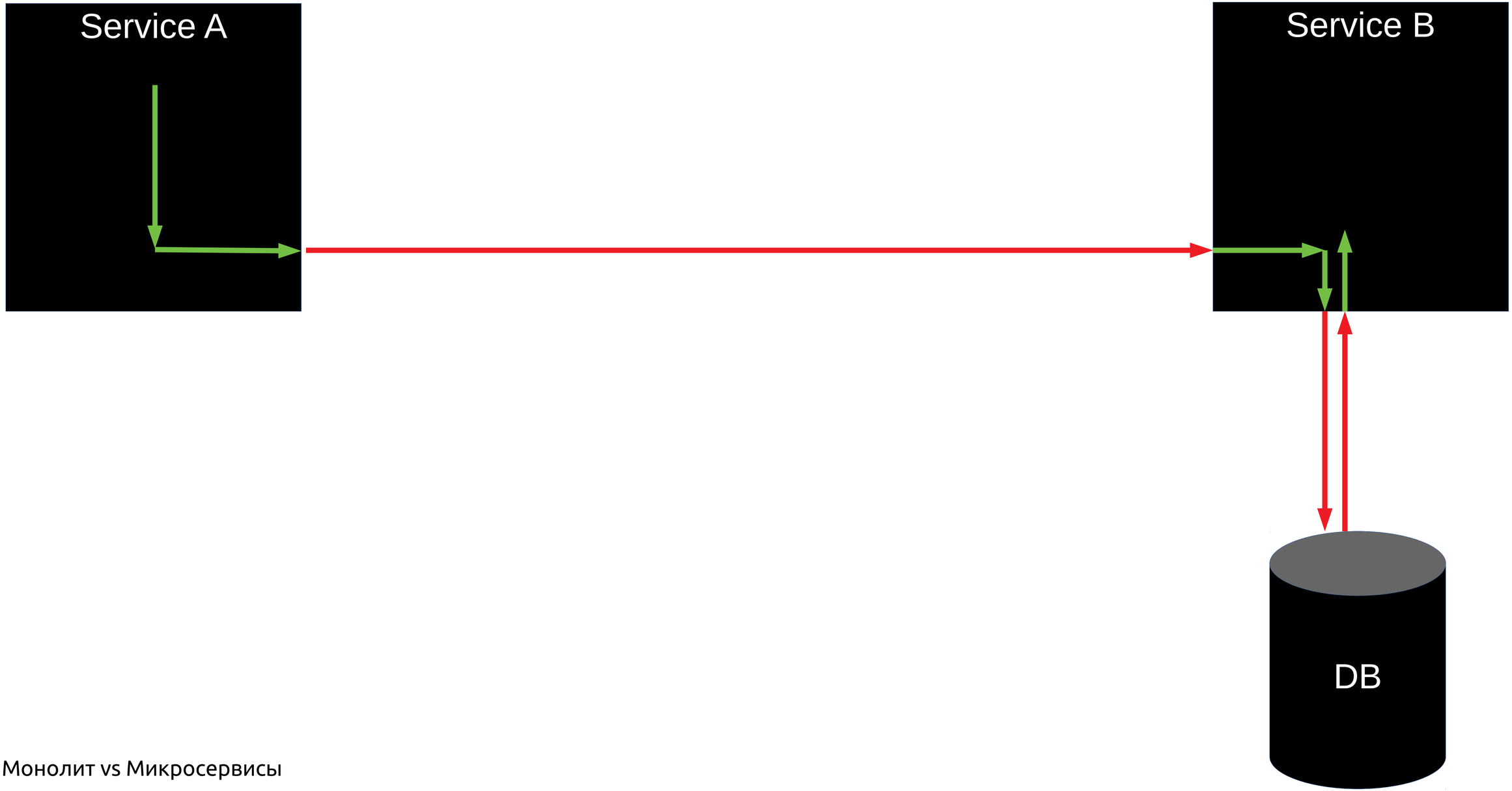
Запрос



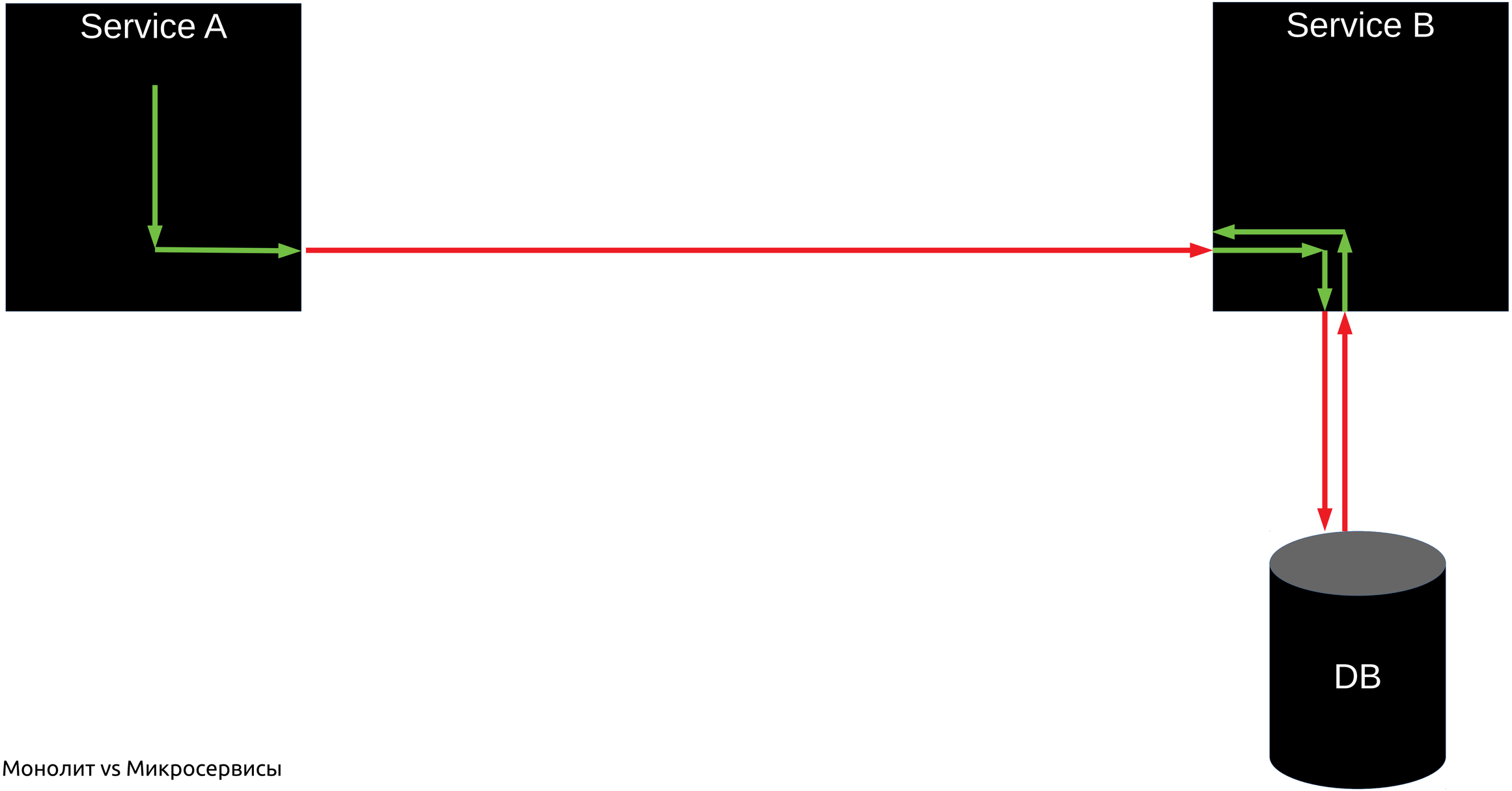
Запрос



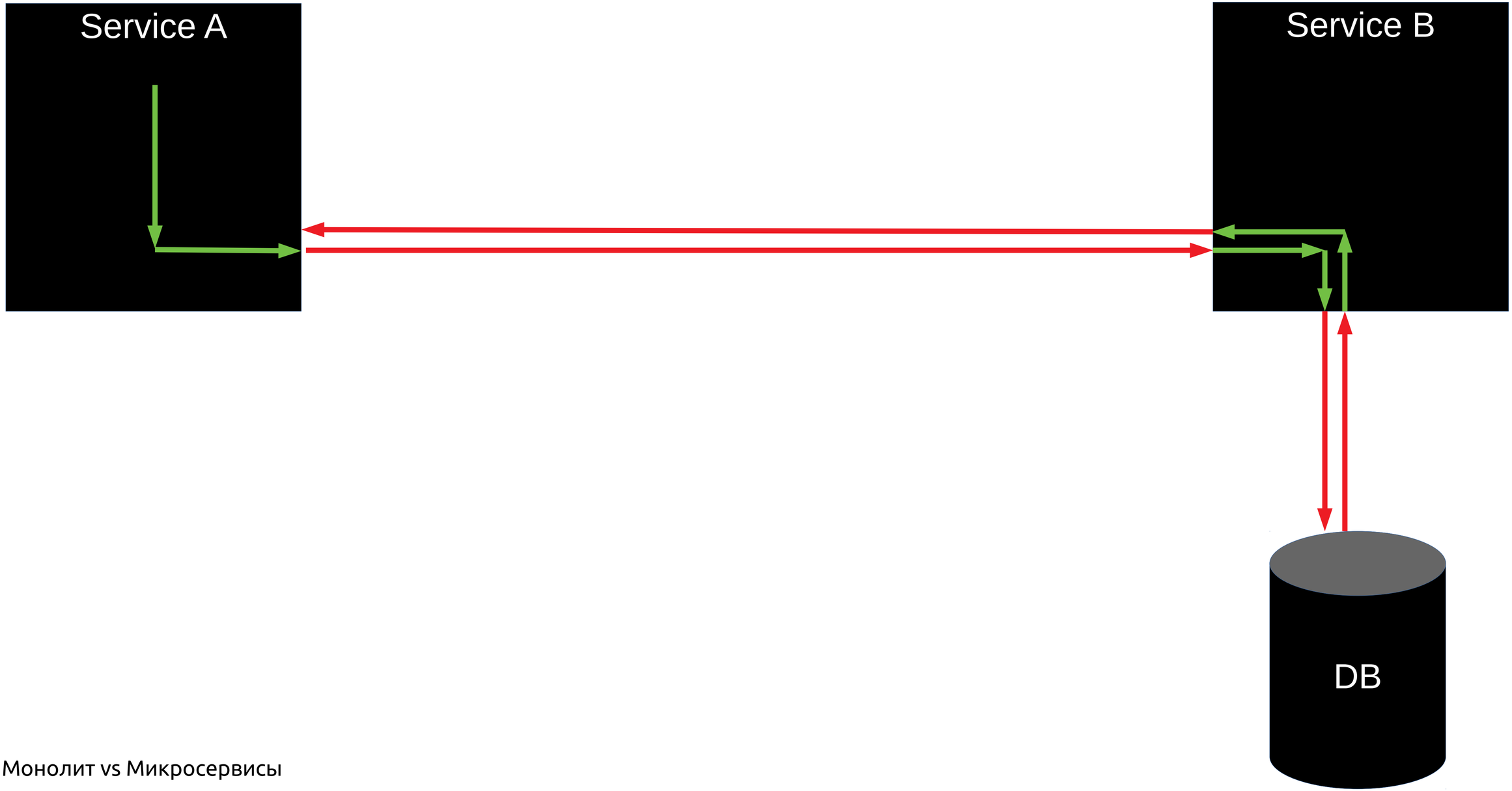
Запрос



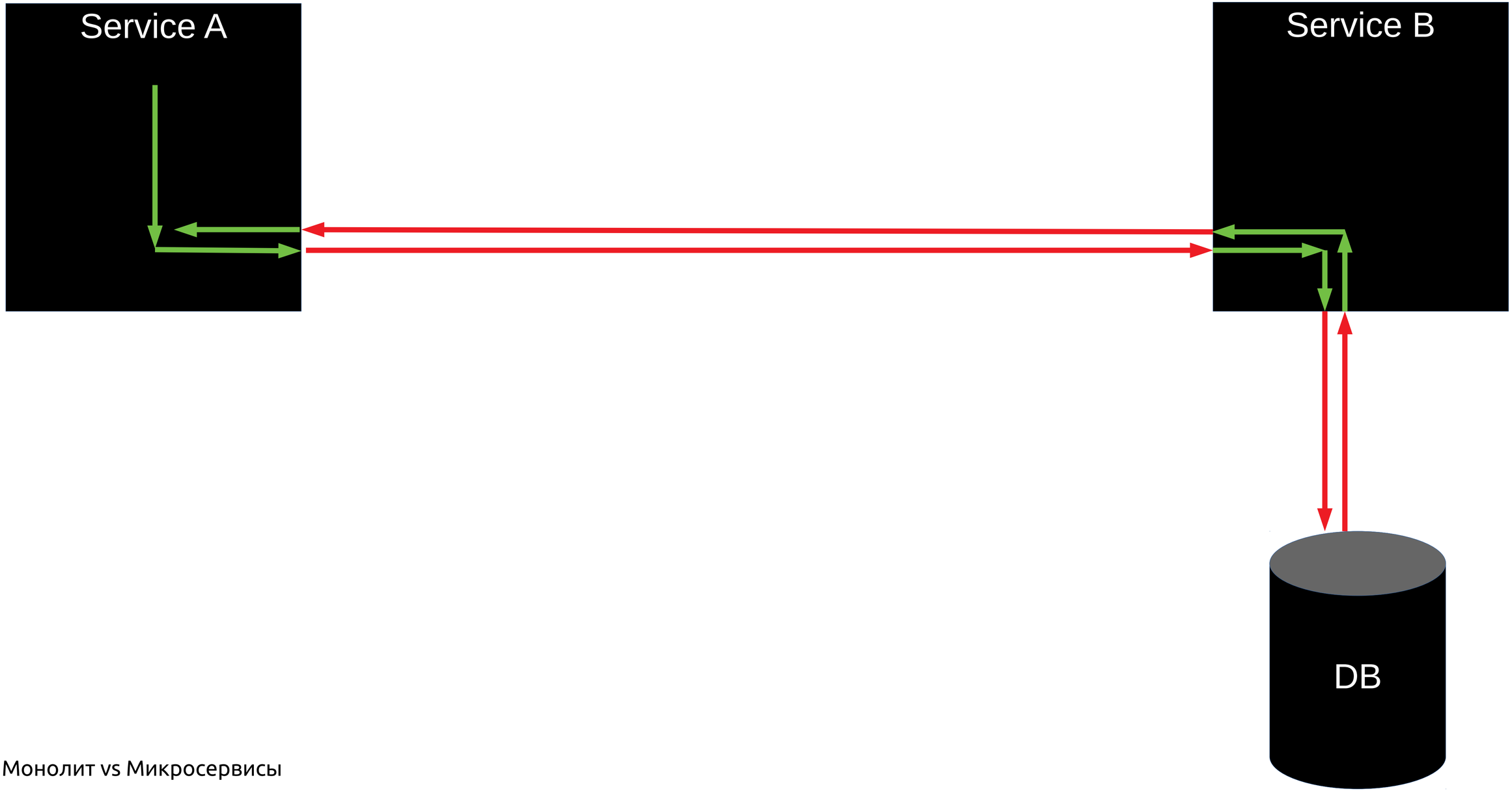
Запрос



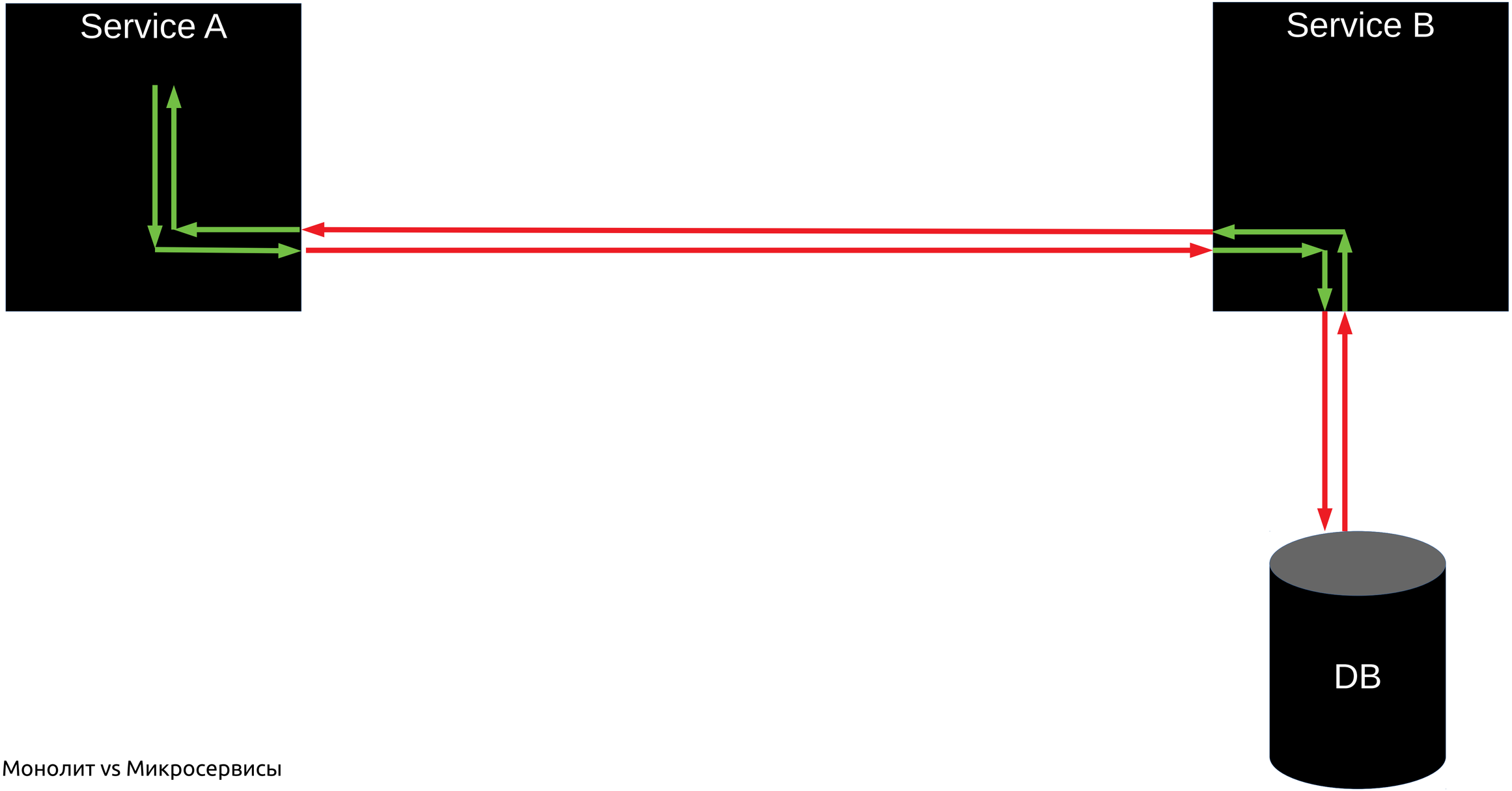
Запрос



Запрос

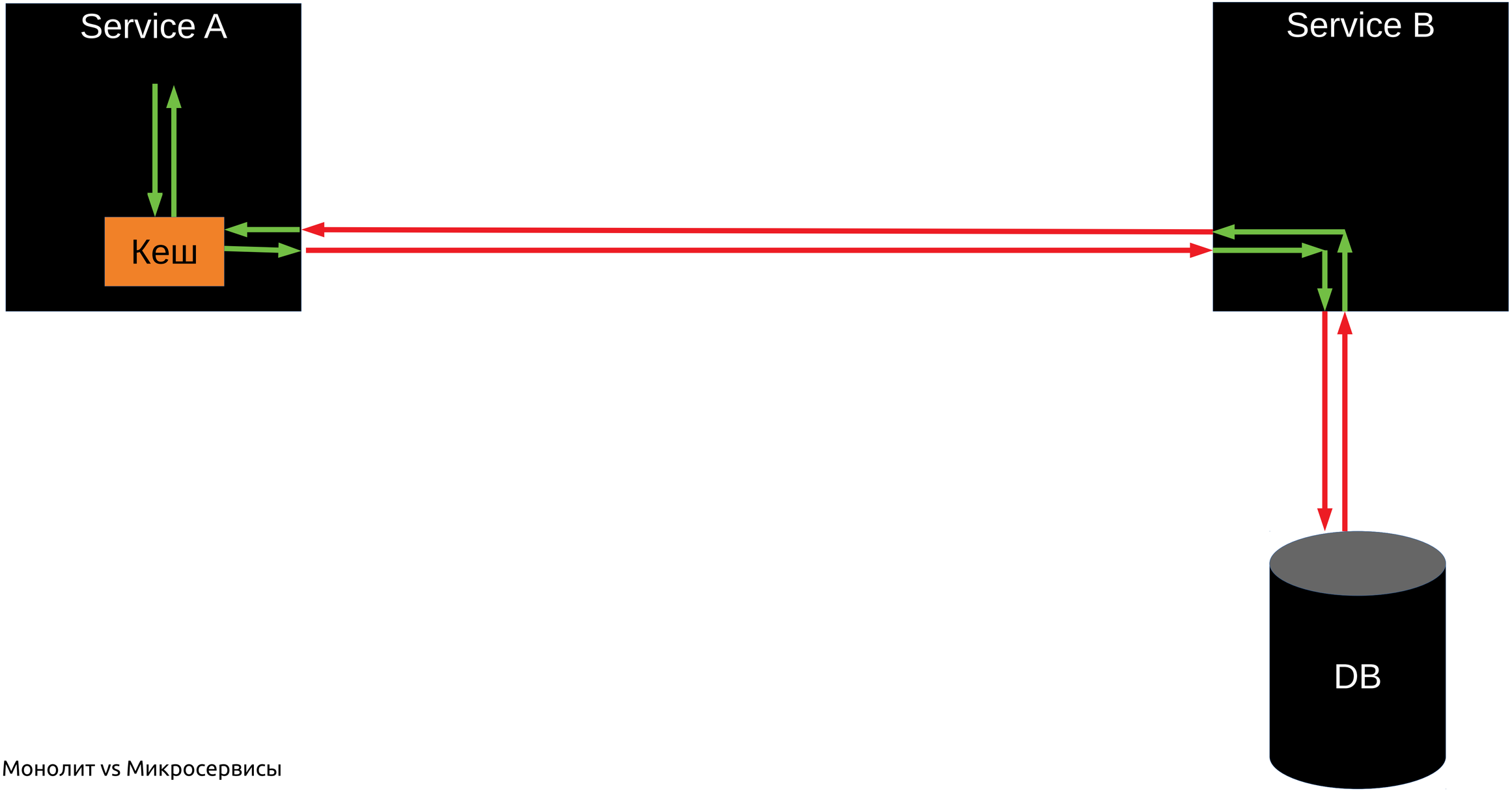


Запрос



userver: кеши

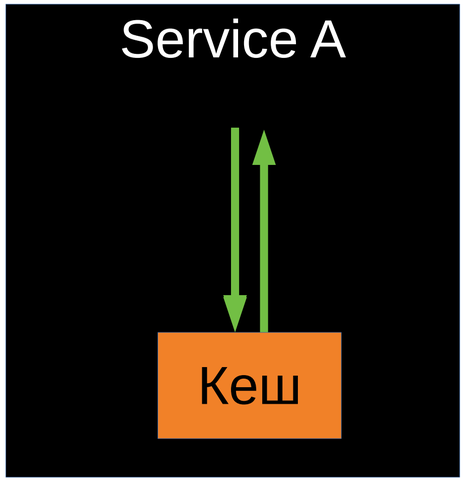
Кеши

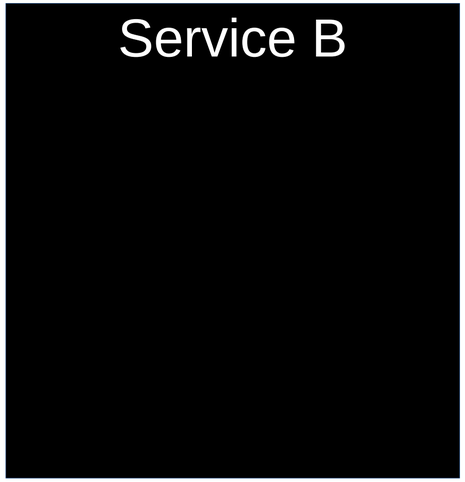


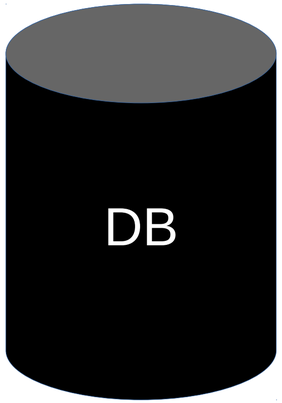
Кеши

Кеши

Кеши







Разновидности кешей

Разновидности кешей

- 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>;
```

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 файл конфига

```
assortment-trait-cache:
    pgcomponent: psql-nomenclature
    full-update-interval: 1h
    update-interval: 5m
```

Разновидности кешей

- components::PostgreCache< PostgreCachePolicy >

Разновидности кешей

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

Разновидности кешей

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

Разновидности кешей

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

- LRU

Разновидности кешей

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

- LRU
  - cache::LruCacheComponent< Key, Value, Hash, Equal >

Разновидности кешей

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

- LRU
  - cache::LruCacheComponent< Key, Value, Hash, Equal >
  - cache::ExpirableLruCache< Key, Value, Hash, Equal >

Разновидности кешей

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

- LRU
  - cache::LruCacheComponent< Key, Value, Hash, Equal >
  - cache::ExpirableLruCache< Key, Value, Hash, Equal >

- Контейнеры

Разновидности кешей

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

- LRU
  - cache::LruCacheComponent< Key, Value, Hash, Equal >
  - cache::ExpirableLruCache< Key, Value, Hash, Equal >

- Контейнеры:
  - cache::NWayLRU< T, U, Hash, Equal >
  - cache::LruMap< T, U, Hash, Equal >
  - cache::LruSet< T, Hash, Equal >

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->** () &&

Без SharedReadablePtr

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

Без SharedReadablePtr

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

Без SharedReadablePtr

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

SharedReadablePtr

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

SharedReadablePtr

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

SharedReadablePtr

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

userver: динамические конфиги

userver

Фичи

- PostgreSQL

Фичи

- PostgreSQL
- Mongo

Фичи

- PostgreSQL
- Mongo
- Clickhouse

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)
- Formats (JSON, YAML, BSON, …)

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)
- Formats (JSON, YAML, BSON, …)
- Tracing, Metrics

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)
- Formats (JSON, YAML, BSON, …)
- Tracing, Metrics
- Subprocesses

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)
- Formats (JSON, YAML, BSON, …)
- Tracing, Metrics
- Subprocesses
- Rcu, Queue, Subscriptions, MutexSet

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)
- Formats (JSON, YAML, BSON, …)
- Tracing, Metrics
- Subprocesses
- Rcu, Queue, Subscriptions, MutexSet
- Deadlines / Timeouts

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)
- Formats (JSON, YAML, BSON, …)
- Tracing, Metrics
- Subprocesses
- Rcu, Queue, Subscriptions, MutexSet
- Deadlines / Timeouts
- Caches and cache dumps

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)
- Formats (JSON, YAML, BSON, …)
- Tracing, Metrics
- Subprocesses
- Rcu, Queue, Subscriptions, MutexSet
- Deadlines / Timeouts
- Caches and cache dumps
- Decimal, FastPimpl, Containers

Фичи

- PostgreSQL
- Mongo
- Clickhouse
- Redis
- Socket, TLS
- Mutex/ConditionVariable/Semaphore…
- Logs
- Dynamic Configs
- HTTP server/client
- gRPC
- Periodic Tasks
- DistLocks
- Unit testing
- Functional Testing (Testuite)
- Formats (JSON, YAML, BSON, …)
- Tracing, Metrics
- Subprocesses
- Rcu, Queue, Subscriptions, MutexSet
- Deadlines / Timeouts
- Caches and cache dumps
- Decimal, FastPimpl, Containers
- Stacktraces, PFR

userver: C++ хардкор в логах

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {  \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

```
LOG(kInfo) << DebugOutput();
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))

template <class NameHolder, int Line>
struct EntryStorage final {
  static inline StaticLogEntry entry{NameHolder::Get(), Line};
};
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))

template <class NameHolder, int Line>
struct EntryStorage final {
  static inline StaticLogEntry entry{NameHolder::Get(), Line};
};
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))

template <class NameHolder, int Line>
struct EntryStorage final {
  static inline StaticLogEntry entry{NameHolder::Get(), Line};
};
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))

template <class NameHolder, int Line>
struct EntryStorage final {
  static inline StaticLogEntry entry{NameHolder::Get(), Line};
};
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))

template <class NameHolder, int Line>
struct EntryStorage final {
  static inline StaticLogEntry entry{NameHolder::Get(), Line};
};
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {  \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))

template <class NameHolder, int Line>
struct EntryStorage final {
  static inline StaticLogEntry entry{NameHolder::Get(), Line};
};
```

userver

```
#define LOG(lvl)                                                    \
      !USERVER_NAMESPACE::logging::ShouldLog(lvl) && []() -> bool {   \
        struct NameHolder {                                         \
          static constexpr const char* Get() noexcept {             \
            return USERVER_FILEPATH;                                \
          }                                                         \
        };                                                          \
        return !USERVER_NAMESPACE::logging::impl::EntryStorage<     \
                    NameHolder, __LINE__>::entry.ShouldLog();       \
      }(),                                                          \
      ? USERVER_NAMESPACE::logging::impl::Noop{}                    \
      : DO_LOG_TO(USERVER_NAMESPACE::logging::DefaultLoggerOptional(), (lvl))

template <class NameHolder, int Line>
struct EntryStorage final {
  static inline StaticLogEntry entry{NameHolder::Get(), Line};
};
```

userver

```
class StaticLogEntry final {
 public:
  StaticLogEntry(const char* path, int line) noexcept;

  StaticLogEntry(StaticLogEntry&&) = delete;
  StaticLogEntry& operator=(StaticLogEntry&&) = delete;

  bool ShouldLog() const noexcept;

 private:
  static constexpr std::size_t kContentSize =
      compiler::SelectSize().For64Bit(40).For32Bit(24);
  alignas(void*) std::byte content[kContentSize];
};
```

userver

```
class StaticLogEntry final {
 public:
  StaticLogEntry(const char* path, int line) noexcept;

  StaticLogEntry(StaticLogEntry&&) = delete;
  StaticLogEntry& operator=(StaticLogEntry&&) = delete;

  bool ShouldLog() const noexcept;

 private:
  static constexpr std::size_t kContentSize =
      compiler::SelectSize().For64Bit(40).For32Bit(24);
  alignas(void*) std::byte content[kContentSize];
};
```

userver

```
class StaticLogEntry final {
 public:
  StaticLogEntry(const char* path, int line) noexcept;

  StaticLogEntry(StaticLogEntry&&) = delete;
  StaticLogEntry& operator=(StaticLogEntry&&) = delete;

  bool ShouldLog() const noexcept;

 private:
  static constexpr std::size_t kContentSize =
      compiler::SelectSize().For64Bit(40).For32Bit(24);
  alignas(void*) std::byte content[kContentSize];
};
```

userver

```
class StaticLogEntry final {
 public:
  StaticLogEntry(const char* path, int line) noexcept;

  StaticLogEntry(StaticLogEntry&&) = delete;
  StaticLogEntry& operator=(StaticLogEntry&&) = delete;

  bool ShouldLog() const noexcept;

 private:
  static constexpr std::size_t kContentSize =
      compiler::SelectSize().For64Bit(40).For32Bit(24);
  alignas(void*) std::byte content[kContentSize];
};
```

userver

```
class StaticLogEntry final {
 public:
  StaticLogEntry(const char* path, int line) noexcept;

  StaticLogEntry(StaticLogEntry&&) = delete;
  StaticLogEntry& operator=(StaticLogEntry&&) = delete;

  bool ShouldLog() const noexcept;

 private:
  static constexpr std::size_t kContentSize =
      compiler::SelectSize().For64Bit(40).For32Bit(24);
  alignas(void*) std::byte content[kContentSize];
};
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}

namespace bi = boost::intrusive;
struct LogEntryContent {
  LogEntryContent(const char* path, int line) : line(line), path(path) {}

  std::atomic<bool> should_log{false};
  const int line;
  const char* const path;
  bi::set_base_hook<bi::optimize_size<true>, bi::link_mode<bi::normal_link>> hook;
};
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}

namespace bi = boost::intrusive;
struct LogEntryContent {
  LogEntryContent(const char* path, int line) : line(line), path(path) {}

  std::atomic<bool> should_log{false};
  const int line;
  const char* const path;
  bi::set_base_hook<bi::optimize_size<true>, bi::link_mode<bi::normal_link>> hook;
};
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}

namespace bi = boost::intrusive;
struct LogEntryContent {
  LogEntryContent(const char* path, int line) : line(line), path(path) {}

  std::atomic<bool> should_log{false};
  const int line;
  const char* const path;
  bi::set_base_hook<bi::optimize_size<true>, bi::link_mode<bi::normal_link>> hook;
};
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}

namespace bi = boost::intrusive;
struct LogEntryContent {
  LogEntryContent(const char* path, int line) : line(line), path(path) {}

  std::atomic<bool> should_log{false};
  const int line;
  const char* const path;
  bi::set_base_hook<bi::optimize_size<true>, bi::link_mode<bi::normal_link>> hook;
};
```

userver

```
StaticLogEntry::StaticLogEntry(const char* path, int line) noexcept {
  utils::impl::AssertStaticRegistrationAllowed();
  UASSERT(path);
  UASSERT(line);

  auto* item = new (&content) LogEntryContent(path, line);
  GetAllLocations().insert(*item);
}

namespace bi = boost::intrusive;
struct LogEntryContent {
  LogEntryContent(const char* path, int line) : line(line), path(path) {}

  std::atomic<bool> should_log{false};
  const int line;
  const char* const path;
  bi::set_base_hook<bi::optimize_size<true>, bi::link_mode<bi::normal_link>> hook;
};
```

Что в итоге получилось?

userver

```
void AddDynamicDebugLog(const std::string& location_relative, int line);

void RemoveDynamicDebugLog(const std::string& location_relative, int line);

const LogEntryContentSet& GetDynamicDebugLocations();
```

Итого

Итоги

Итоги

Благодаря микросервисам у нас:

Итоги

Благодаря микросервисам у нас:

- Быстрые выкатки

Итоги

Благодаря микросервисам у нас:

- Быстрые выкатки
- Подросла надёжность

Итоги

Благодаря микросервисам у нас:

- Быстрые выкатки
- Подросла надёжность
- Прогон тестов занимает минуты

Итоги

Благодаря микросервисам у нас:

- Быстрые выкатки
- Подросла надёжность
- Прогон тестов занимает минуты

userver

Итоги

Благодаря микросервисам у нас:

- Быстрые выкатки
- Подросла надёжность
- Прогон тестов занимает минуты

userver

- Позволил переиспользовать код

Итоги

Благодаря микросервисам у нас:

- Быстрые выкатки
- Подросла надёжность
- Прогон тестов занимает минуты

userver

- Позволил переиспользовать код
- Упрощает и ускоряет разработку

Итоги

Благодаря микросервисам у нас:

- Быстрые выкатки
- Подросла надёжность
- Прогон тестов занимает минуты

userver

- Позволил переиспользовать код
- Упрощает и ускоряет разработку

- Содержит под капотом много весёлого и интересноего 😊

Спасибо

Полухин Антон

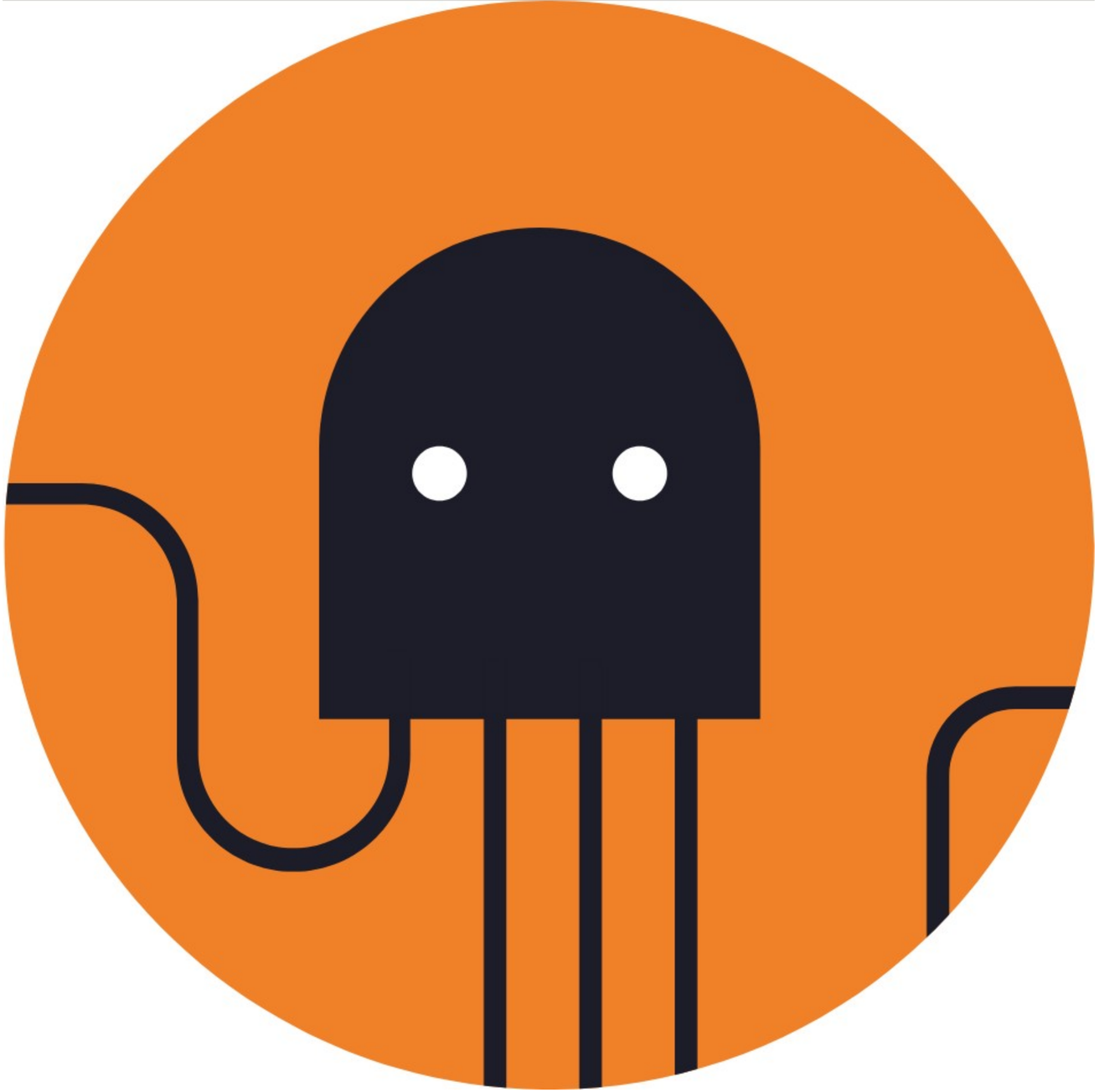
Эксперт-разработчик C++

antoshkka@gmail.com

antoshkka@yandex-team.ru

https://github.com/apolukhin

https://stdcpp.ru/



https://github.com/userver-framework