Yandex for developers *//>

YATALKS

Бэкенд

Сервис на userver: что снаружи и что под капотом

Антон Полухин,
эксперт-разработчик C++

Содержание

01 Насколько просто написать микросервис на C++ ?

02 Сделаем key-value сервис с использованием Postgres

03 А что там под капотом:

* Просто запрос к базе данных?..
* Динамические конфиги и RCU
* А что там ещё есть...

01

Просто ли написать микросервис на C++ ?

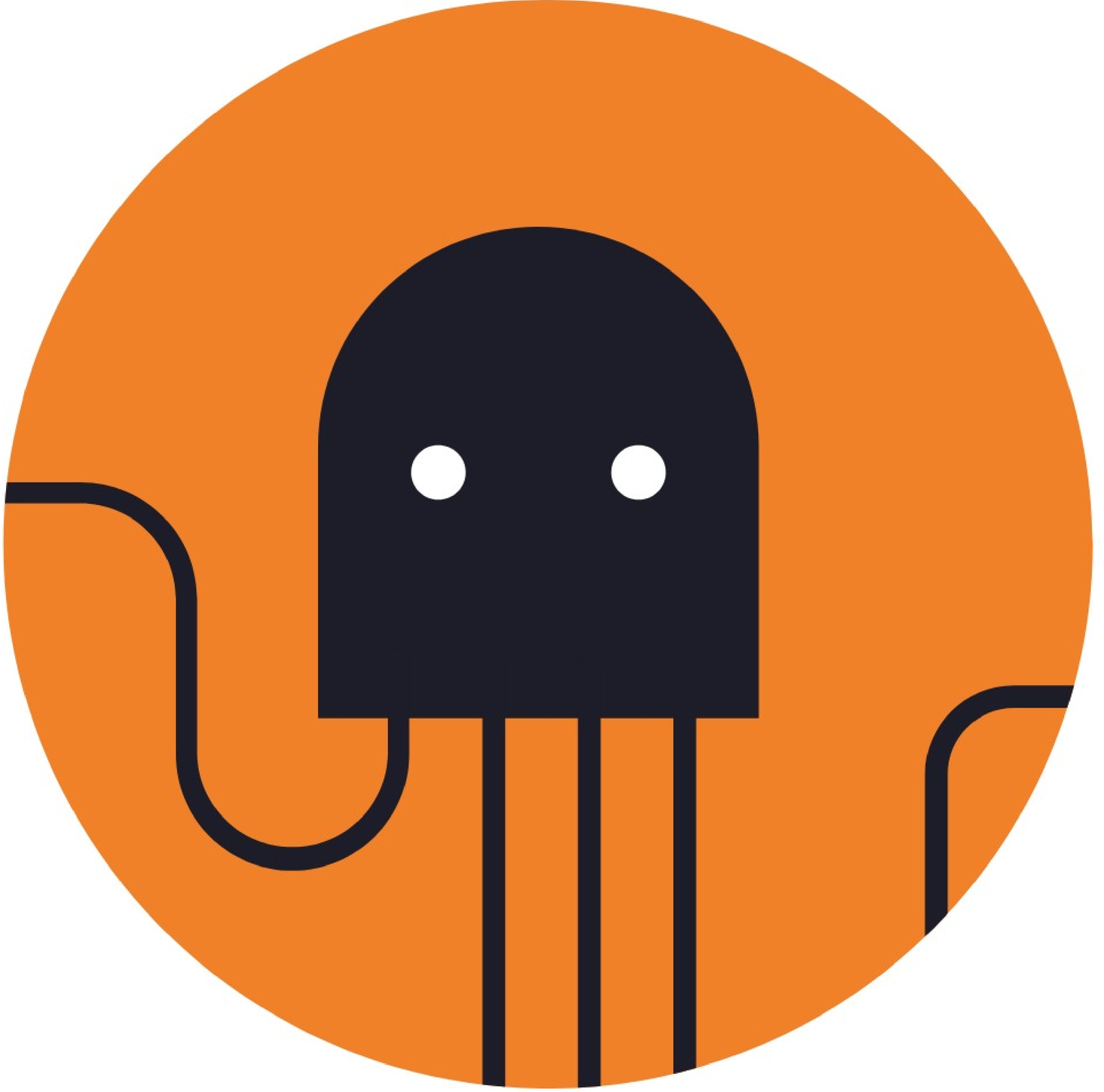
Hello world

```cpp
struct Hello final : public server::handlers::HttpHandlerBase {
  using server::handlers::HttpHandlerBase::HttpHandlerBase;

  static constexpr std::string_view kName = "handler-hello-sample";

  std::string HandleRequestThrow(
      const server::http::HttpRequest&,
      server::request::RequestContext&) const override {
    return "Hello world!\n";
  }
};

int main(int argc, char* argv[]) {
  const auto component_list = components::MinimalServerComponentList()  //
                                  .Append<Hello>();                      //
  utils::DaemonMain(argc, argv, component_list);
}
```



Hello world

```
struct Hello final : public server::handlers::HttpHandlerBase {
  using server::handlers::HttpHandlerBase::HttpHandlerBase;

  static constexpr std::string_view kName = "handler-hello-sample";

  std::string HandleRequestThrow(
      const server::http::HttpRequest&,
      server::request::RequestContext&) const override {
    return "Hello world!\n";
  }
};

int main(int argc, char* argv[]) {
  const auto component_list = components::MinimalServerComponentList()  //
                                  .Append<Hello>();                      //
  utils::DaemonMain(argc, argv, component_list);
}
```

Hello world

```
struct Hello final : public server::handlers::HttpHandlerBase {
  using server::handlers::HttpHandlerBase::HttpHandlerBase;

  static constexpr std::string_view kName = "handler-hello-sample";

  std::string HandleRequestThrow(
      const server::http::HttpRequest&,
      server::request::RequestContext&) const override {
    return "Hello world!\n";
  }
};

int main(int argc, char* argv[]) {
  const auto component_list = components::MinimalServerComponentList()  //
                                  .Append<Hello>();                      //
  utils::DaemonMain(argc, argv, component_list);
}
```

Hello world

```
struct Hello final : public server::handlers::HttpHandlerBase {
  using server::handlers::HttpHandlerBase::HttpHandlerBase;

  static constexpr std::string_view kName = "handler-hello-sample";

  std::string HandleRequestThrow(
      const server::http::HttpRequest&,
      server::request::RequestContext&) const override {
    return "Hello world!\n";
  }
};

int main(int argc, char* argv[]) {
  const auto component_list = components::MinimalServerComponentList()  //
                                  .Append<Hello>();                      //
  utils::DaemonMain(argc, argv, component_list);
}
```

Hello world

```
struct Hello final : public server::handlers::HttpHandlerBase {
  using server::handlers::HttpHandlerBase::HttpHandlerBase;

  static constexpr std::string_view kName = "handler-hello-sample";

  std::string HandleRequestThrow(
      const server::http::HttpRequest&,
      server::request::RequestContext&) const override {
    return "Hello world!\n";
  }
};

int main(int argc, char* argv[]) {
  const auto component_list = components::MinimalServerComponentList()  //
                                  .Append<Hello>();                      //
  utils::DaemonMain(argc, argv, component_list);
}
```

Hello world

```
struct Hello final : public server::handlers::HttpHandlerBase {
  using server::handlers::HttpHandlerBase::HttpHandlerBase;

  static constexpr std::string_view kName = "handler-hello-sample";

  std::string HandleRequestThrow(
      const server::http::HttpRequest&,
      server::request::RequestContext&) const override {
    return "Hello world!\n";
  }
};

int main(int argc, char* argv[]) {
  const auto component_list = components::MinimalServerComponentList()  //
                                  .Append<Hello>();                      //
  utils::DaemonMain(argc, argv, component_list);
}
```

Hello world

```
struct Hello final : public server::handlers::HttpHandlerBase {
  using server::handlers::HttpHandlerBase::HttpHandlerBase;

  static constexpr std::string_view kName = "handler-hello-sample";

  std::string HandleRequestThrow(
      const server::http::HttpRequest&,
      server::request::RequestContext&) const override {
    return "Hello world!\n";
  }
};

int main(int argc, char* argv[]) {
  const auto component_list = components::MinimalServerComponentList()  //
                                  .Append<Hello>();                      //
  utils::DaemonMain(argc, argv, component_list);
}
```

Hello world

```
struct Hello final : public server::handlers::HttpHandlerBase {
  using server::handlers::HttpHandlerBase::HttpHandlerBase;

  static constexpr std::string_view kName = "handler-hello-sample";

  std::string HandleRequestThrow(
      const server::http::HttpRequest&,
      server::request::RequestContext&) const override {
    return "Hello world!\n";
  }
};

int main(int argc, char* argv[]) {
  const auto component_list = components::MinimalServerComponentList()  //
                                  .Append<Hello>();                      //
  utils::DaemonMain(argc, argv, component_list);
}
```

02

Сделаем key-value сервис с использованием Postgres

Hello world #2

```
class KeyValue final : public server::handlers::HttpHandlerBase {
 public:
  static constexpr std::string_view kName = "handler-key-value";

  KeyValue(const components::ComponentConfig& config,
           const components::ComponentContext& context);

  std::string HandleRequestThrow(
      const server::http::HttpRequest& request,
      server::request::RequestContext&) const override;

 private:
  // ...
  storages::postgres::ClusterPtr pg_cluster_;
};
```

Hello world #2

```
class KeyValue final : public server::handlers::HttpHandlerBase {
 public:
  static constexpr std::string_view kName = "handler-key-value";

  KeyValue(const components::ComponentConfig& config,
           const components::ComponentContext& context);

  std::string HandleRequestThrow(
      const server::http::HttpRequest& request,
      server::request::RequestContext&) const override;

 private:
  // ...
  storages::postgres::ClusterPtr pg_cluster_;
};
```

Hello world #2

```
KeyValue::KeyValue(const components::ComponentConfig& config,
                   const components::ComponentContext& context)
    : HttpHandlerBase(config, context),
      pg_cluster_(
          context.FindComponent<components::Postgres>("key-value-database")
              .GetCluster()) {

  constexpr auto kCreateTable = R"~(
      CREATE TABLE IF NOT EXISTS key_value_table (
        key VARCHAR PRIMARY KEY,
        value VARCHAR
      )
    )~";

  using storages::postgres::ClusterHostType;
  pg_cluster_->Execute(ClusterHostType::kMaster, kCreateTable);
}
```

Hello world #2

```
KeyValue::KeyValue(const components::ComponentConfig& config,
                   const components::ComponentContext& context)
    : HttpHandlerBase(config, context),
      pg_cluster_(
          context.FindComponent<components::Postgres>("key-value-database")
              .GetCluster()) {

  constexpr auto kCreateTable = R"~(
      CREATE TABLE IF NOT EXISTS key_value_table (
        key VARCHAR PRIMARY KEY,
        value VARCHAR
      )
    )~";

  using storages::postgres::ClusterHostType;
  pg_cluster_->Execute(ClusterHostType::kMaster, kCreateTable);
}
```

Hello world #2

```
KeyValue::KeyValue(const components::ComponentConfig& config,
                   const components::ComponentContext& context)
    : HttpHandlerBase(config, context),
      pg_cluster_(
          context.FindComponent<components::Postgres>("key-value-database")
              .GetCluster()) {

  constexpr auto kCreateTable = R"~(
      CREATE TABLE IF NOT EXISTS key_value_table (
        key VARCHAR PRIMARY KEY,
        value VARCHAR
      )
    )~";

  using storages::postgres::ClusterHostType;
  pg_cluster_->Execute(ClusterHostType::kMaster, kCreateTable);
}
```

Hello world #2

```
KeyValue::KeyValue(const components::ComponentConfig& config,
                   const components::ComponentContext& context)
    : HttpHandlerBase(config, context),
      pg_cluster_(
          context.FindComponent<components::Postgres>("key-value-database")
              .GetCluster()) {

  constexpr auto kCreateTable = R"~(
      CREATE TABLE IF NOT EXISTS key_value_table (
        key VARCHAR PRIMARY KEY,
        value VARCHAR
      )
    )~";

  using storages::postgres::ClusterHostType;
  pg_cluster_->Execute(ClusterHostType::kMaster, kCreateTable);
}
```

Hello world #2

```
std::string KeyValue::HandleRequestThrow(
    const server::http::HttpRequest& request,
    server::request::RequestContext&) const {
  const auto& key = request.GetArg("key");

  switch (request.GetMethod()) {
    case server::http::HttpMethod::kGet:
      return GetValue(key, request);
    case server::http::HttpMethod::kPost:
      return PostValue(key, request);
    case server::http::HttpMethod::kDelete:
      return DeleteValue(key);
    default:
      throw server::handlers::ClientError(server::handlers::ExternalBody{
          fmt::format("Unsupported method {}", request.GetMethod())});
  }
}
```

Hello world #2

```
std::string KeyValue::HandleRequestThrow(
    const server::http::HttpRequest& request,
    server::request::RequestContext&) const {
  const auto& key = request.GetArg("key");

  switch (request.GetMethod()) {
    case server::http::HttpMethod::kGet:
      return GetValue(key, request);
    case server::http::HttpMethod::kPost:
      return PostValue(key, request);
    case server::http::HttpMethod::kDelete:
      return DeleteValue(key);
    default:
      throw server::handlers::ClientError(server::handlers::ExternalBody{
          fmt::format("Unsupported method {}", request.GetMethod())});
  }
}
```

Hello world #2

```
std::string KeyValue::HandleRequestThrow(
    const server::http::HttpRequest& request,
    server::request::RequestContext&) const {
  const auto& key = request.GetArg("key");

  switch (request.GetMethod()) {
    case server::http::HttpMethod::kGet:
      return GetValue(key, request);
    case server::http::HttpMethod::kPost:
      return PostValue(key, request);
    case server::http::HttpMethod::kDelete:
      return DeleteValue(key);
    default:
      throw server::handlers::ClientError(server::handlers::ExternalBody{
          fmt::format("Unsupported method {}", request.GetMethod())});
  }
}
```

Hello world #2

```
std::string KeyValue::HandleRequestThrow(
    const server::http::HttpRequest& request,
    server::request::RequestContext&) const {
  const auto& key = request.GetArg("key");

  switch (request.GetMethod()) {
    case server::http::HttpMethod::kGet:
      return GetValue(key, request);
    case server::http::HttpMethod::kPost:
      return PostValue(key, request);
    case server::http::HttpMethod::kDelete:
      return DeleteValue(key);
    default:
      throw server::handlers::ClientError(server::handlers::ExternalBody{
          fmt::format("Unsupported method {}", request.GetMethod())});
  }
}
```

Hello world #2

```
std::string KeyValue::HandleRequestThrow(
    const server::http::HttpRequest& request,
    server::request::RequestContext&) const {
  const auto& key = request.GetArg("key");

  switch (request.GetMethod()) {
    case server::http::HttpMethod::kGet:
      return GetValue(key, request);
    case server::http::HttpMethod::kPost:
      return PostValue(key, request);
    case server::http::HttpMethod::kDelete:
      return DeleteValue(key);
    default:
      throw server::handlers::ClientError(server::handlers::ExternalBody{
          fmt::format("Unsupported method {}", request.GetMethod())});
  }
}
```

Hello world #2

```
std::string KeyValue::DeleteValue(std::string_view key) const {
  auto res =
      pg_cluster_->Execute(storages::postgres::ClusterHostType::kMaster,
                           "DELETE FROM key_value_table WHERE key=$1", key);
  return std::to_string(res.RowsAffected());
}
```

Hello world #2

```
std::string KeyValue::DeleteValue(std::string_view key) const {
  auto res =
      pg_cluster_->Execute(storages::postgres::ClusterHostType::kMaster,
                           "DELETE FROM key_value_table WHERE key=$1", key);
  return std::to_string(res.RowsAffected());
}
```

Hello world #2

```
std::string KeyValue::DeleteValue(std::string_view key) const {
  auto res =
      pg_cluster_->Execute(storages::postgres::ClusterHostType::kMaster,
                           "DELETE FROM key_value_table WHERE key=$1", key);
  return std::to_string(res.RowsAffected());
}
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};

std::string KeyValue::GetValue(std::string_view key,
                               const server::http::HttpRequest& request) const {
  storages::postgres::ResultSet res = pg_cluster_->Execute(
      storages::postgres::ClusterHostType::kSlave, kSelectValue, key);

  if (res.IsEmpty()) {
    request.SetResponseStatus(server::http::HttpStatus::kNotFound);
    return {};
  }

  return res.AsSingleRow<std::string>();
}
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};

std::string KeyValue::GetValue(std::string_view key,
                               const server::http::HttpRequest& request) const {
  storages::postgres::ResultSet res = pg_cluster_->Execute(
      storages::postgres::ClusterHostType::kSlave, kSelectValue, key);

  if (res.IsEmpty()) {
    request.SetResponseStatus(server::http::HttpStatus::kNotFound);
    return {};
  }

  return res.AsSingleRow<std::string>();
}
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};

std::string KeyValue::GetValue(std::string_view key,
                               const server::http::HttpRequest& request) const {
  storages::postgres::ResultSet res = pg_cluster_->Execute(
      storages::postgres::ClusterHostType::kSlave, kSelectValue, key);

  if (res.IsEmpty()) {
    request.SetResponseStatus(server::http::HttpStatus::kNotFound);
    return {};
  }

  return res.AsSingleRow<std::string>();
}
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};

std::string KeyValue::GetValue(std::string_view key,
                               const server::http::HttpRequest& request) const {
  storages::postgres::ResultSet res = pg_cluster_->Execute(
      storages::postgres::ClusterHostType::kSlave, kSelectValue, key);

  if (res.IsEmpty()) {
    request.SetResponseStatus(server::http::HttpStatus::kNotFound);
    return {};
  }

  return res.AsSingleRow<std::string>();
}
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};

std::string KeyValue::GetValue(std::string_view key,
                               const server::http::HttpRequest& request) const {
  storages::postgres::ResultSet res = pg_cluster_->Execute(
      storages::postgres::ClusterHostType::kSlave, kSelectValue, key);

  if (res.IsEmpty()) {
    request.SetResponseStatus(server::http::HttpStatus::kNotFound);
    return {};
  }

  return res.AsSingleRow<std::string>();
}
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};

std::string KeyValue::GetValue(std::string_view key,
                               const server::http::HttpRequest& request) const {
  storages::postgres::ResultSet res = pg_cluster_->Execute(
      storages::postgres::ClusterHostType::kSlave, kSelectValue, key);

  if (res.IsEmpty()) {
    request.SetResponseStatus(server::http::HttpStatus::kNotFound);
    return {};
  }

  return res.AsSingleRow<std::string>();
}
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};

std::string KeyValue::GetValue(std::string_view key,
                               const server::http::HttpRequest& request) const {
  storages::postgres::ResultSet res = pg_cluster_->Execute(
      storages::postgres::ClusterHostType::kSlave, kSelectValue, key);

  if (res.IsEmpty()) {
    request.SetResponseStatus(server::http::HttpStatus::kNotFound);
    return {};
  }

  return res.AsSingleRow<std::string>();
}
```

Hello world #2

```
std::string KeyValue::PostValue(
    std::string_view key, const server::http::HttpRequest& request) const {
  const auto& value = request.GetArg("value");

  storages::postgres::Transaction transaction =
      pg_cluster_->Begin("sample_transaction_insert_key_value",
                         storages::postgres::ClusterHostType::kMaster, {});

  auto res = transaction.Execute(kInsertValue, key, value);
  if (res.RowsAffected()) {
    transaction.Commit();
    request.SetResponseStatus(server::http::HttpStatus::kCreated);
    return std::string{value};
  }

  res = transaction.Execute(kSelectValue, key);
  transaction.Rollback();
```

Hello world #2

```
std::string KeyValue::PostValue(
    std::string_view key, const server::http::HttpRequest& request) const {
  const auto& value = request.GetArg("value");

  storages::postgres::Transaction transaction =
      pg_cluster_->Begin("sample_transaction_insert_key_value",
                         storages::postgres::ClusterHostType::kMaster, {});

  auto res = transaction.Execute(kInsertValue, key, value);
  if (res.RowsAffected()) {
    transaction.Commit();
    request.SetResponseStatus(server::http::HttpStatus::kCreated);
    return std::string{value};
  }

  res = transaction.Execute(kSelectValue, key);
  transaction.Rollback();
```

Hello world #2

```
std::string KeyValue::PostValue(
    std::string_view key, const server::http::HttpRequest& request) const {
  const auto& value = request.GetArg("value");

  storages::postgres::Transaction transaction =
      pg_cluster_->Begin("sample_transaction_insert_key_value",
                         storages::postgres::ClusterHostType::kMaster, {});

  auto res = transaction.Execute(kInsertValue, key, value);
  if (res.RowsAffected()) {
    transaction.Commit();
    request.SetResponseStatus(server::http::HttpStatus::kCreated);
    return std::string{value};
  }

  res = transaction.Execute(kSelectValue, key);
  transaction.Rollback();
```

Hello world #2

```
std::string KeyValue::PostValue(
    std::string_view key, const server::http::HttpRequest& request) const {
  const auto& value = request.GetArg("value");

  storages::postgres::Transaction transaction =
      pg_cluster_->Begin("sample_transaction_insert_key_value",
                         storages::postgres::ClusterHostType::kMaster, {});

  auto res = transaction.Execute(kInsertValue, key, value);
  if (res.RowsAffected()) {
    transaction.Commit();
    request.SetResponseStatus(server::http::HttpStatus::kCreated);
    return std::string{value};
  }

  res = transaction.Execute(kSelectValue, key);
  transaction.Rollback();
```

Hello world #2

```
std::string KeyValue::PostValue(
    std::string_view key, const server::http::HttpRequest& request) const {
  const auto& value = request.GetArg("value");

  storages::postgres::Transaction transaction =
      pg_cluster_->Begin("sample_transaction_insert_key_value",
                         storages::postgres::ClusterHostType::kMaster, {});

  auto res = transaction.Execute(kInsertValue, key, value);
  if (res.RowsAffected()) {
    transaction.Commit();
    request.SetResponseStatus(server::http::HttpStatus::kCreated);
    return std::string{value};
  }

  res = transaction.Execute(kSelectValue, key);
  transaction.Rollback();
```

Hello world #2

```
std::string KeyValue::PostValue(
    std::string_view key, const server::http::HttpRequest& request) const {
  const auto& value = request.GetArg("value");

  storages::postgres::Transaction transaction =
      pg_cluster_->Begin("sample_transaction_insert_key_value",
                         storages::postgres::ClusterHostType::kMaster, {});

  auto res = transaction.Execute(kInsertValue, key, value);
  if (res.RowsAffected()) {
    transaction.Commit();
    request.SetResponseStatus(server::http::HttpStatus::kCreated);
    return std::string{value};
  }

  res = transaction.Execute(kSelectValue, key);
  transaction.Rollback();
```

Hello world #2

```
std::string KeyValue::PostValue(
    std::string_view key, const server::http::HttpRequest& request) const {
  const auto& value = request.GetArg("value");

  storages::postgres::Transaction transaction =
      pg_cluster_->Begin("sample_transaction_insert_key_value",
                         storages::postgres::ClusterHostType::kMaster, {});

  auto res = transaction.Execute(kInsertValue, key, value);
  if (res.RowsAffected()) {
    transaction.Commit();
    request.SetResponseStatus(server::http::HttpStatus::kCreated);
    return std::string{value};
  }

  res = transaction.Execute(kSelectValue, key);
  transaction.Rollback();
```

Hello world #2

```
std::string KeyValue::PostValue(
    std::string_view key, const server::http::HttpRequest& request) const {
  const auto& value = request.GetArg("value");

  storages::postgres::Transaction transaction =
      pg_cluster_->Begin("sample_transaction_insert_key_value",
                         storages::postgres::ClusterHostType::kMaster, {});

  auto res = transaction.Execute(kInsertValue, key, value);
  if (res.RowsAffected()) {
    transaction.Commit();
    request.SetResponseStatus(server::http::HttpStatus::kCreated);
    return std::string{value};
  }

  res = transaction.Execute(kSelectValue, key);
  transaction.Rollback();
```

03

Что там под капотом?

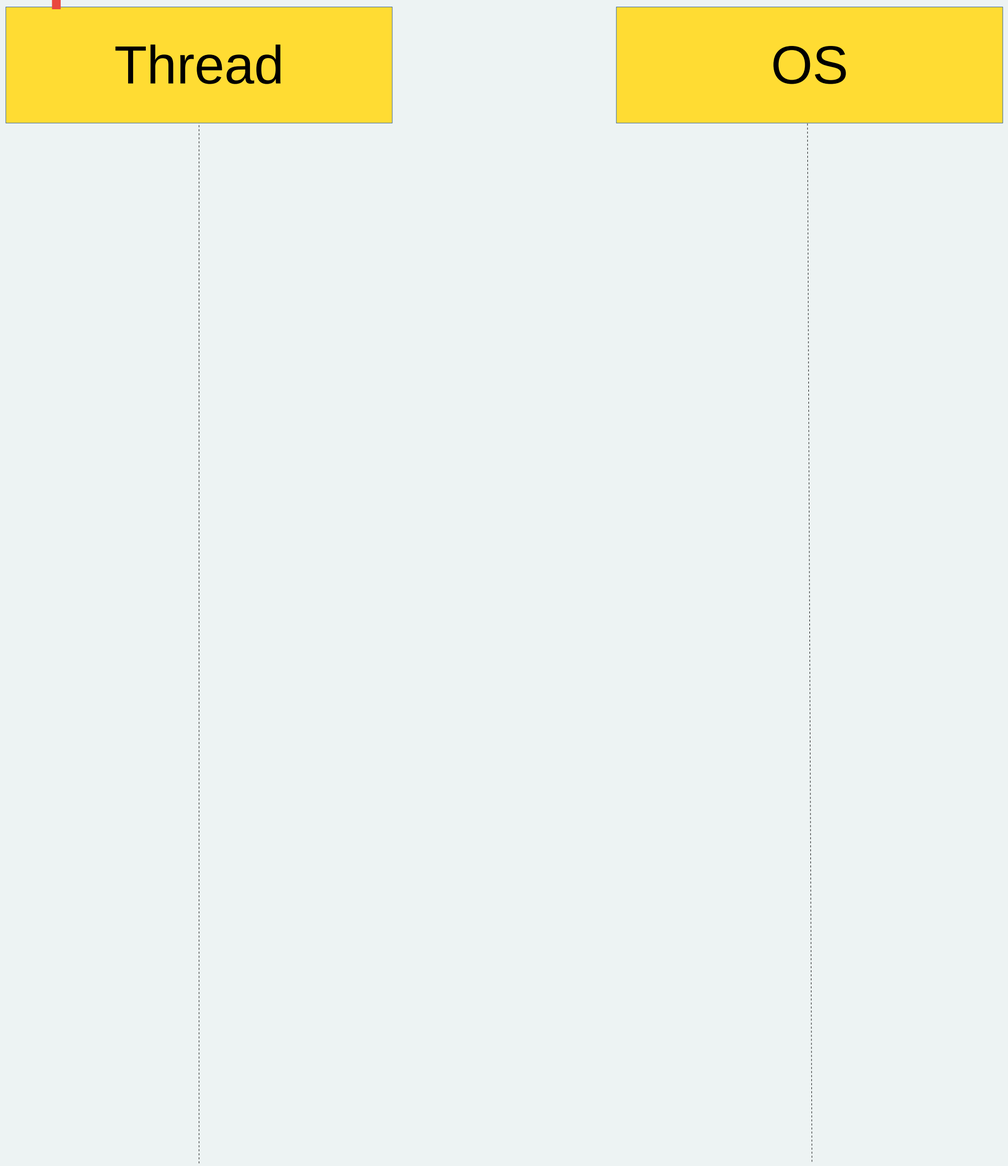
Hello world #2

```
pg_cluster_->Execute(storages::postgres::ClusterHostType::kMaster,
                     "DELETE FROM key_value_table WHERE key=$1", key);
```

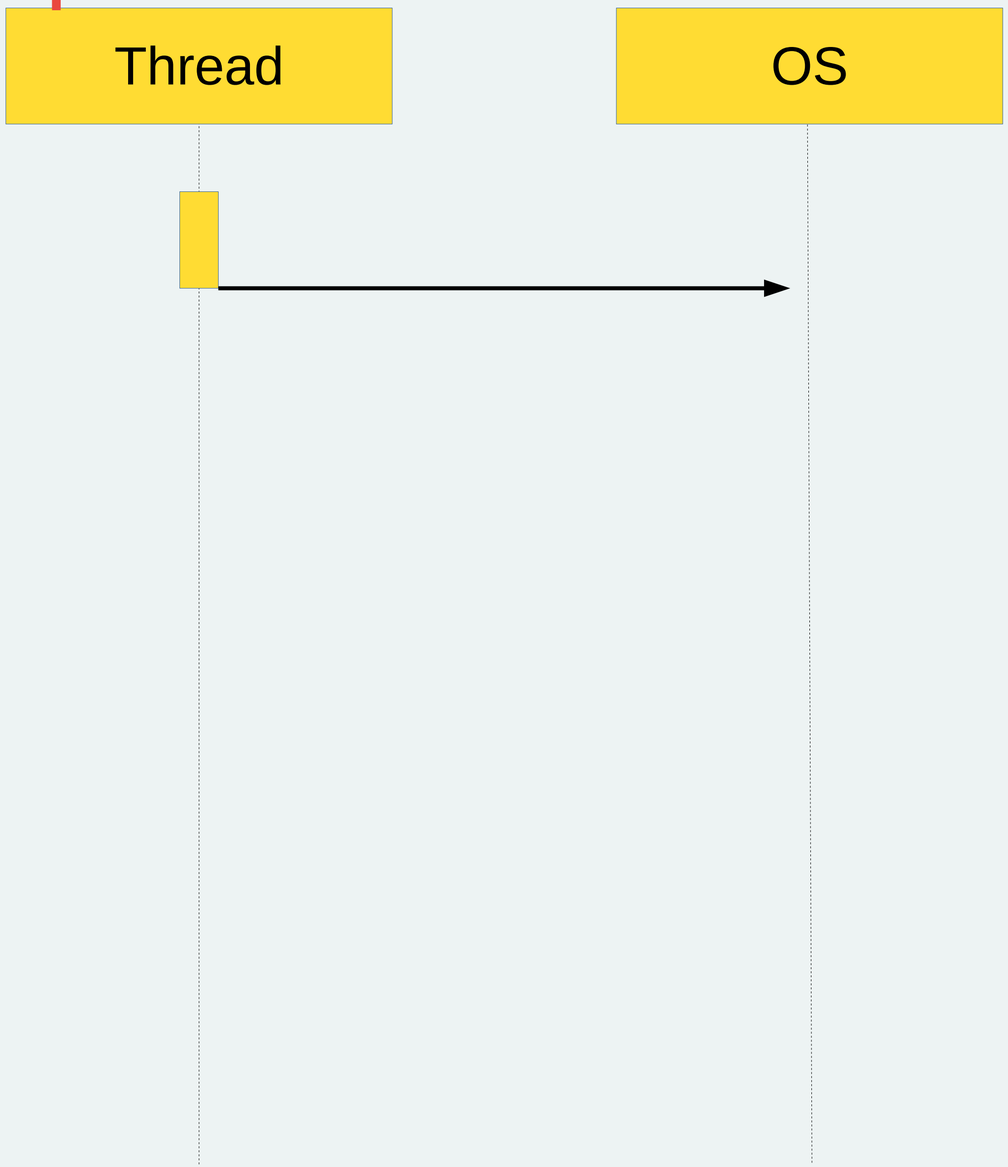
Hello world #2

```
// Создаёт prepared statement при первом использовании
pg_cluster_->Execute(storages::postgres::ClusterHostType::kMaster,
                     "DELETE FROM key_value_table WHERE key=$1", key);
```

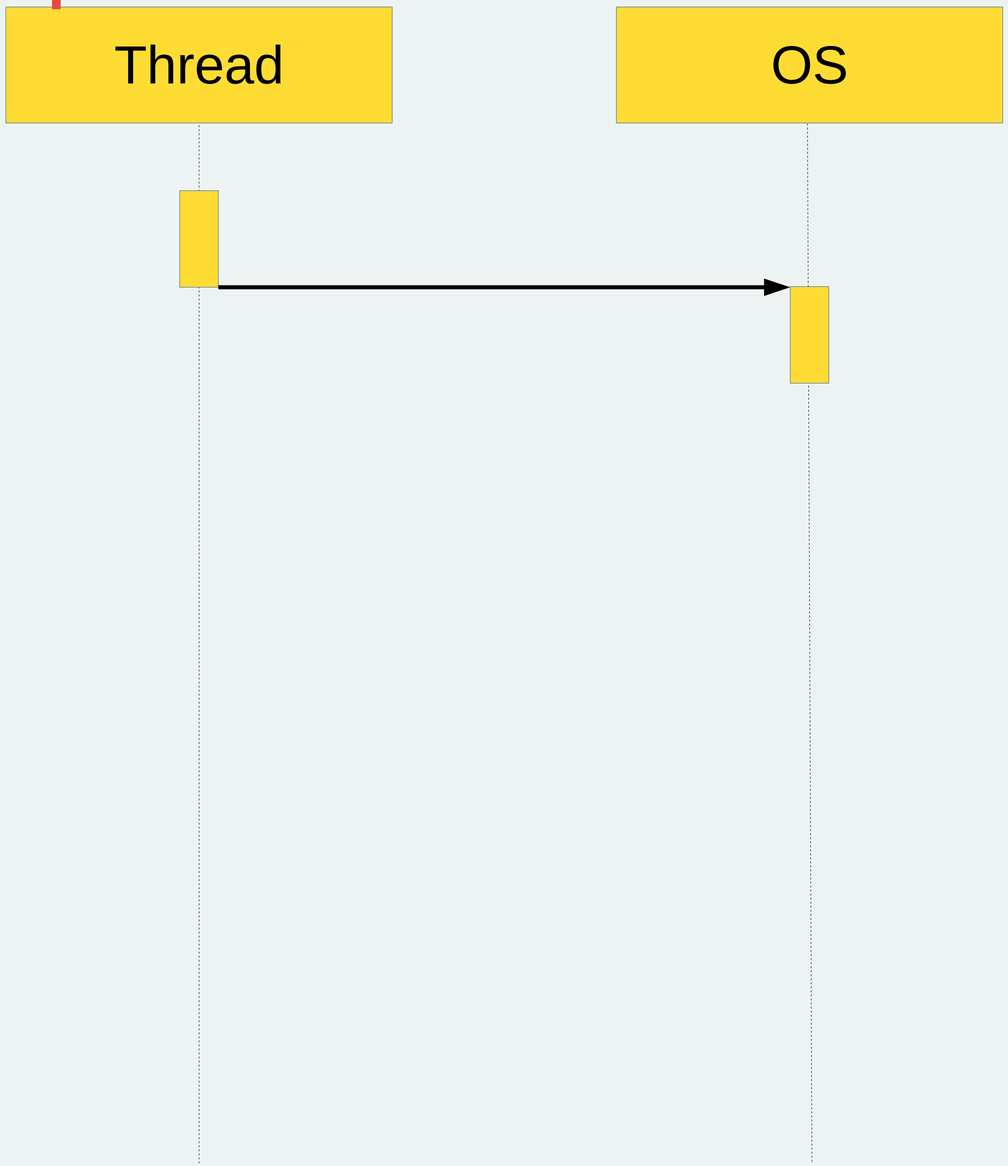
Обычный Request



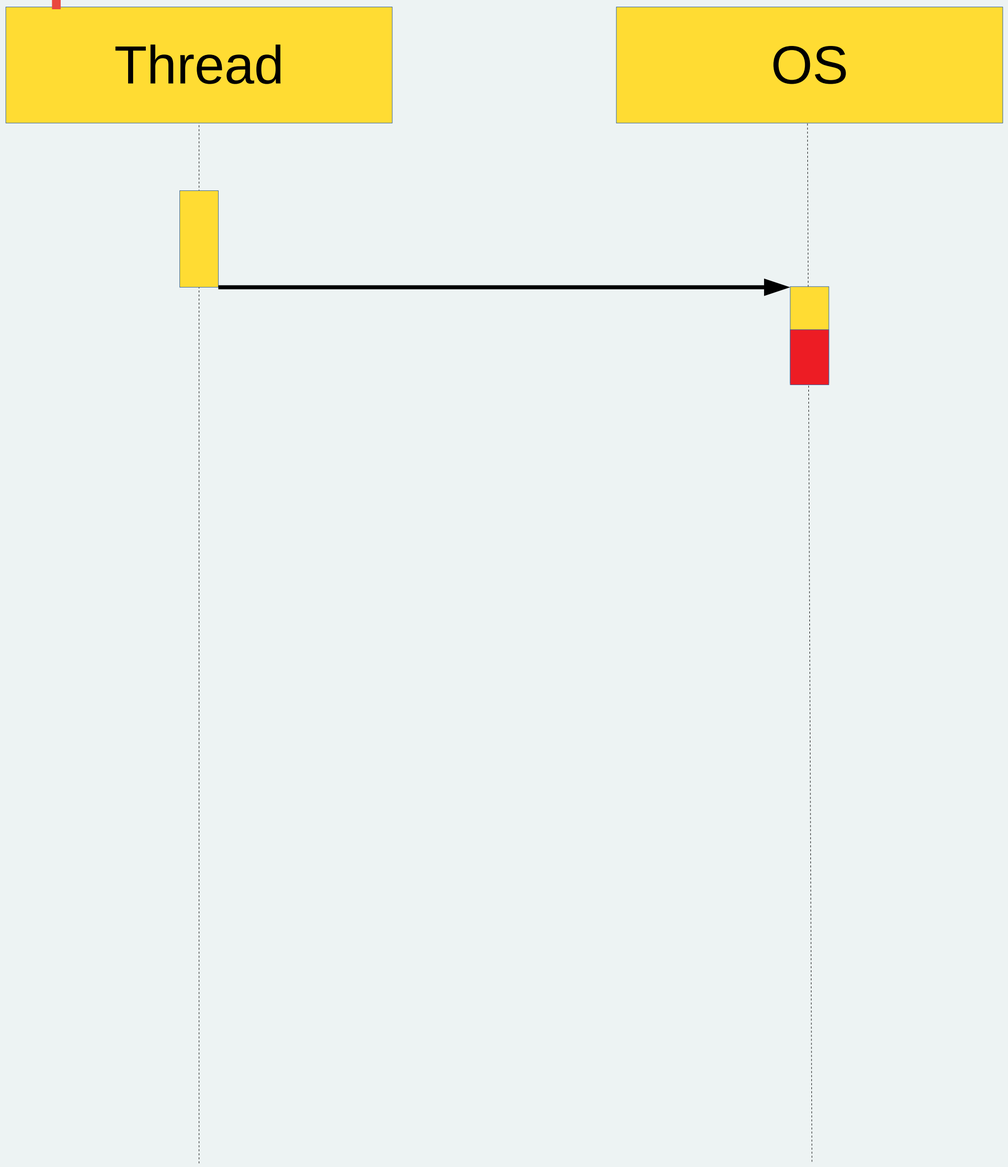
Обычный Request



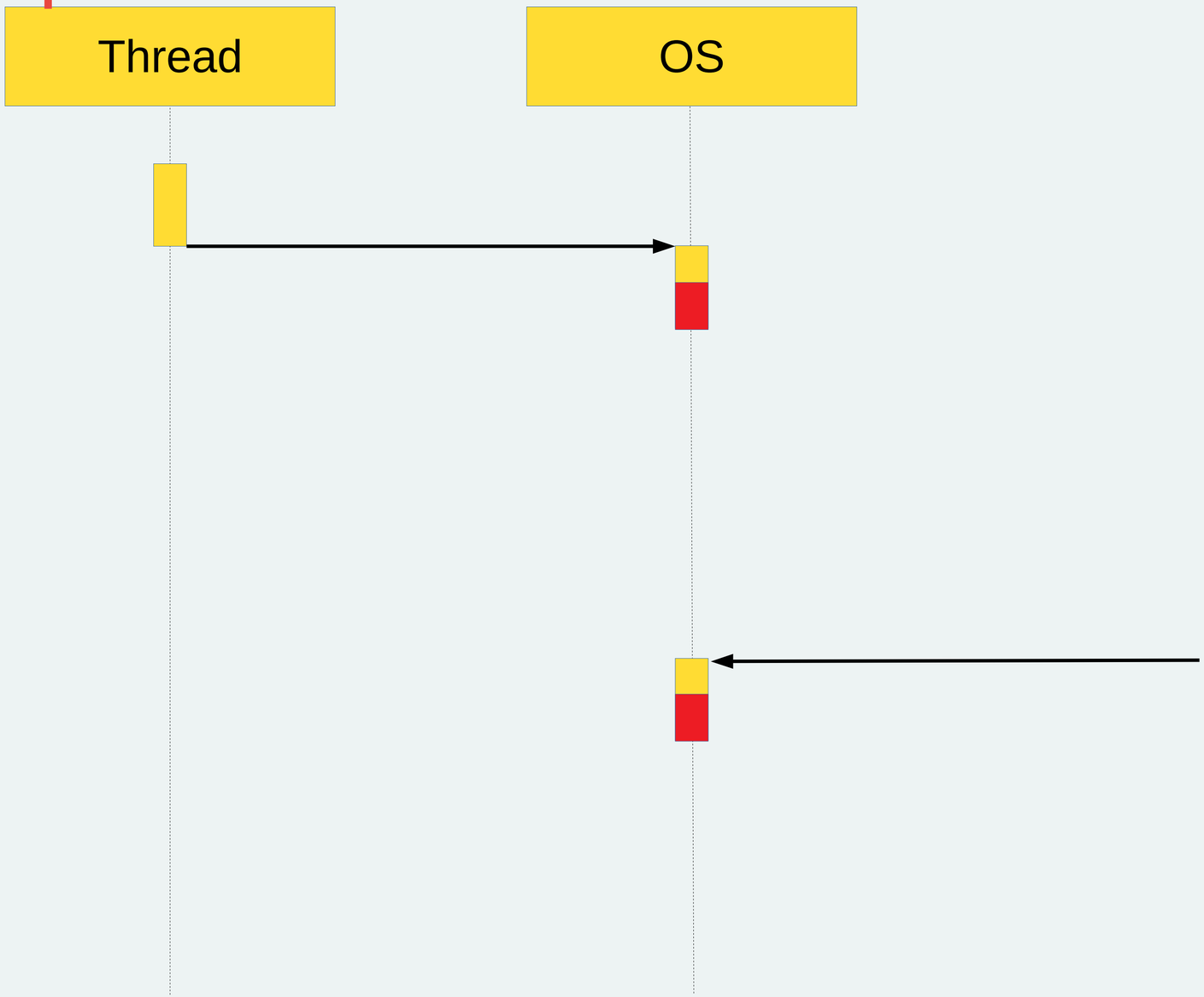
Обычный Request



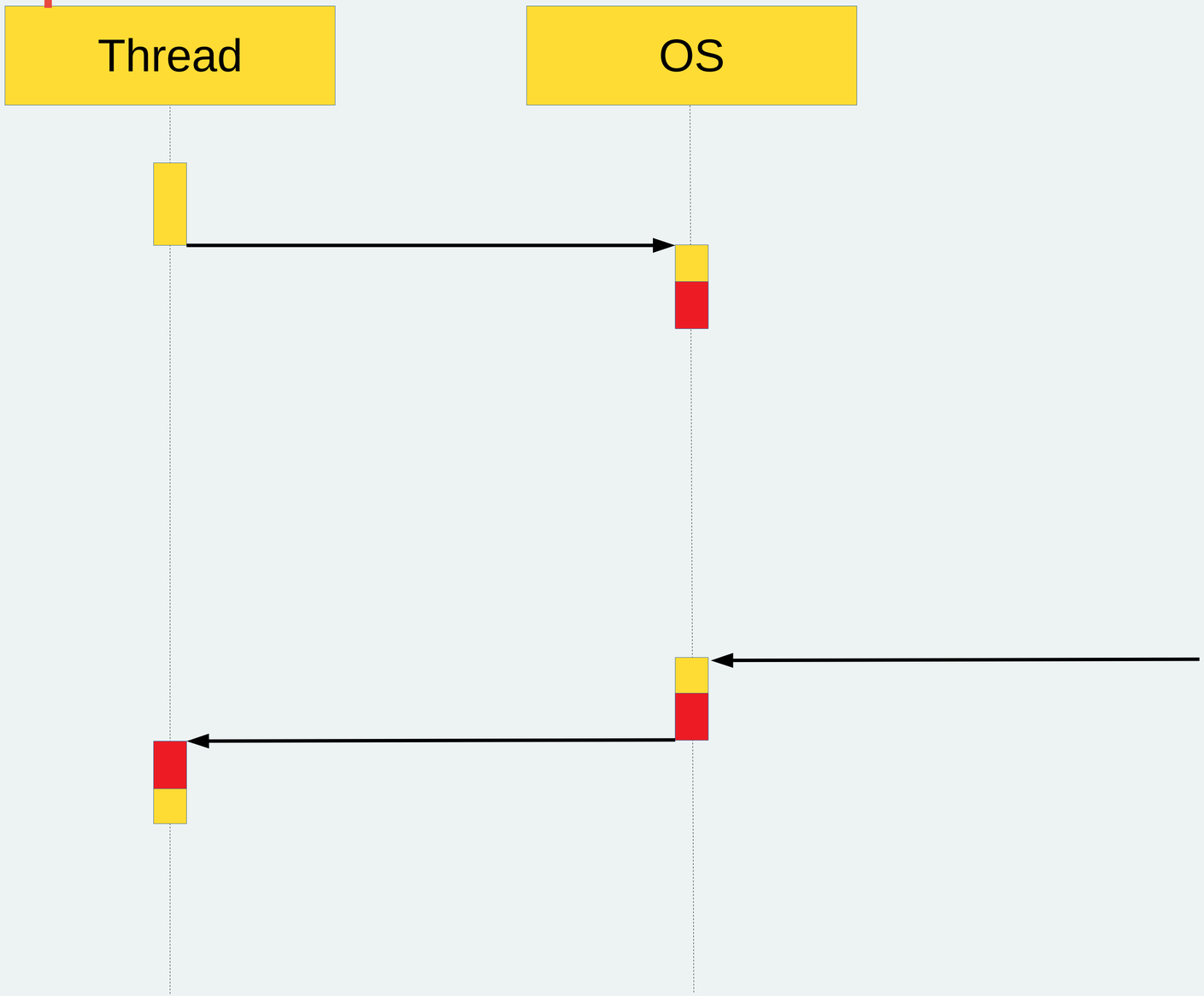
Обычный Request



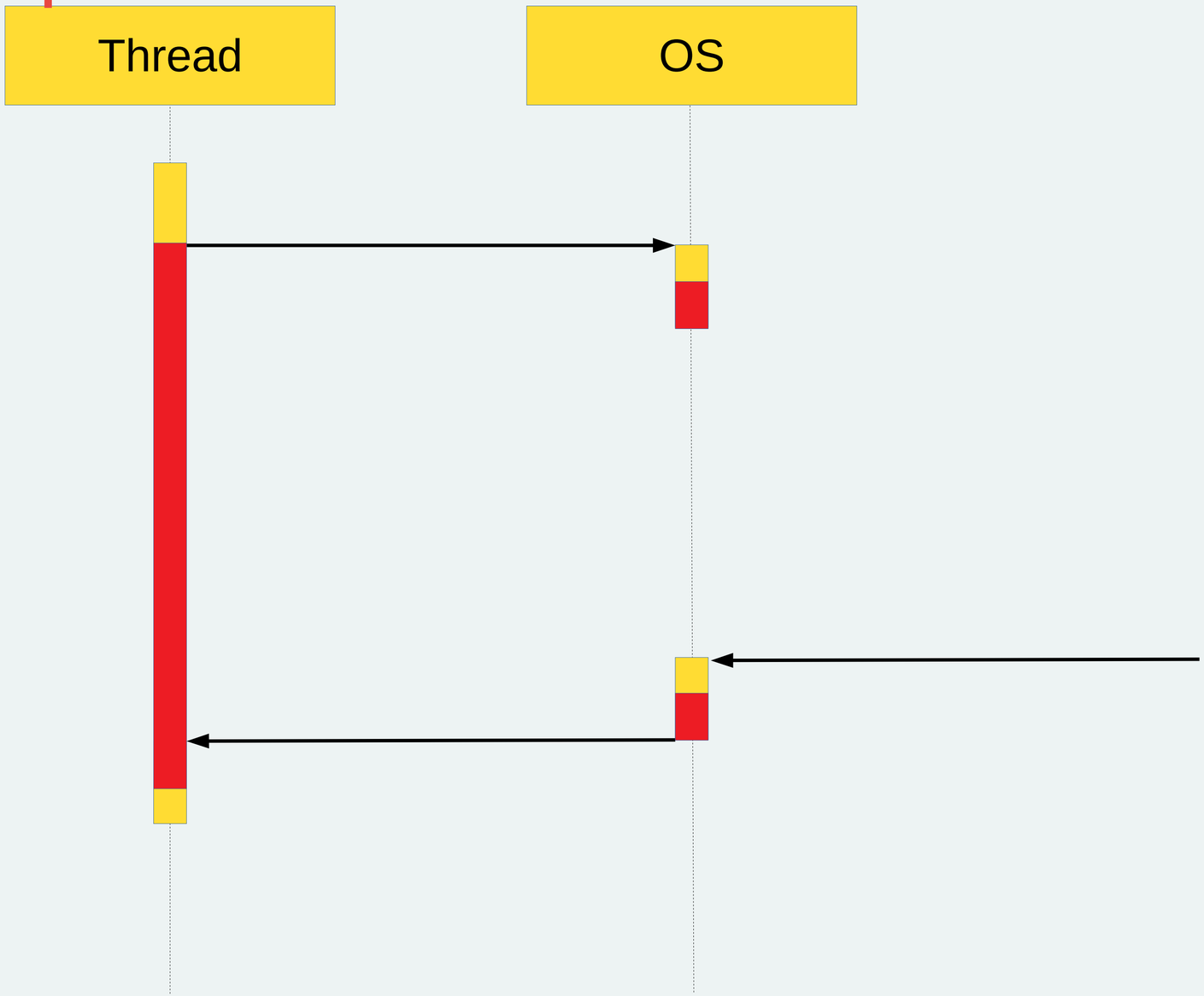
Обычный Request

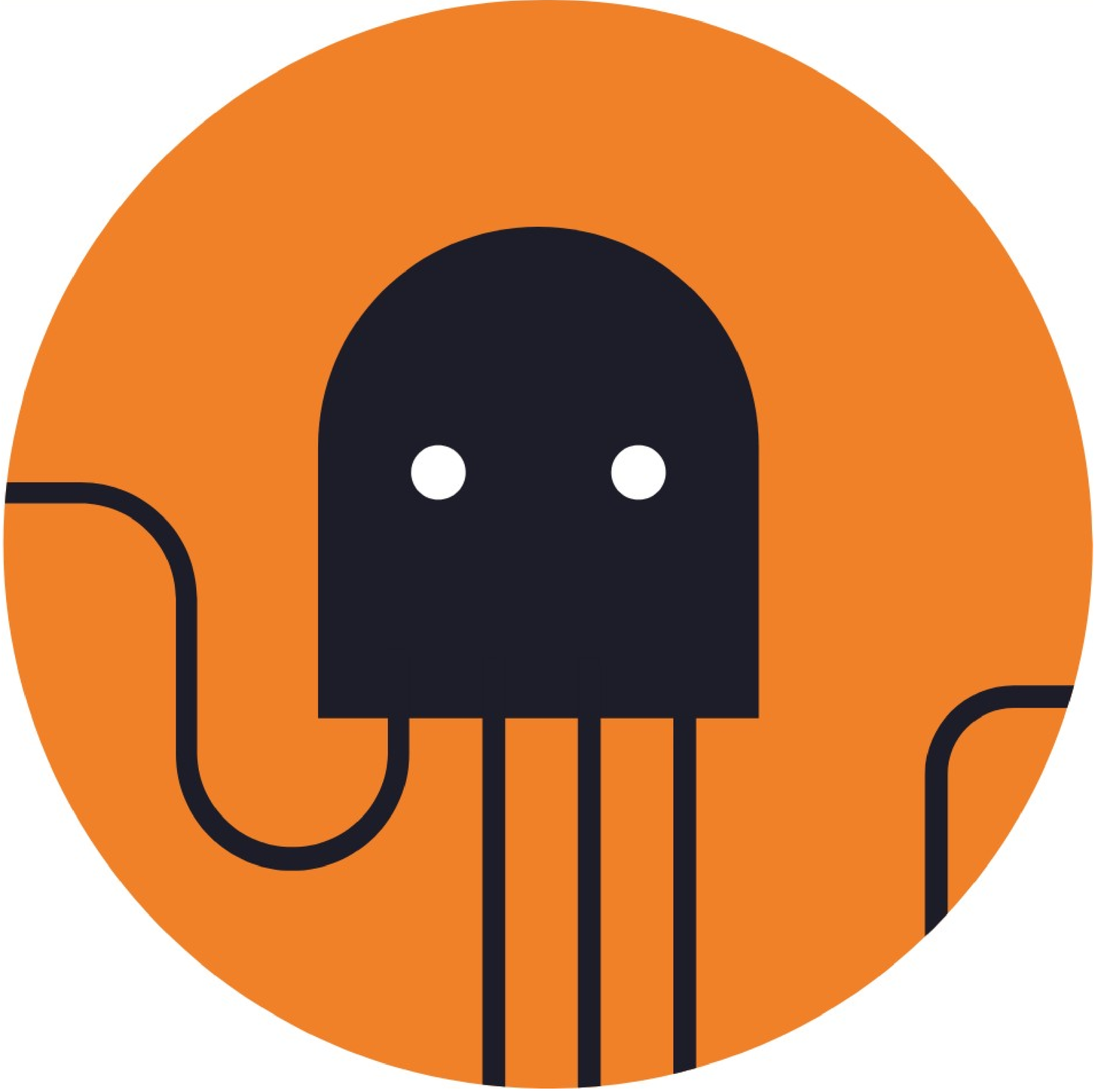


Обычный Request



Обычный Request

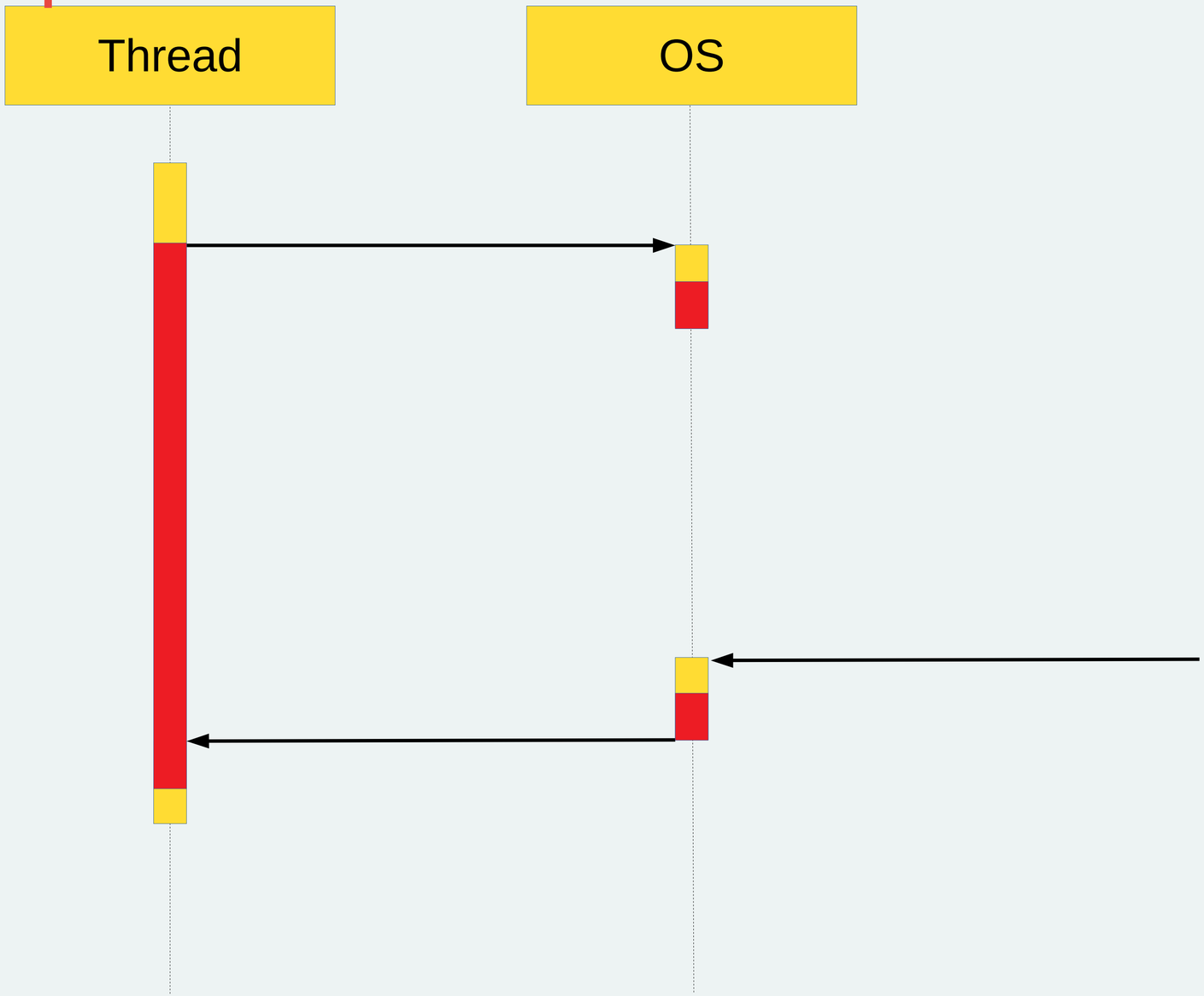




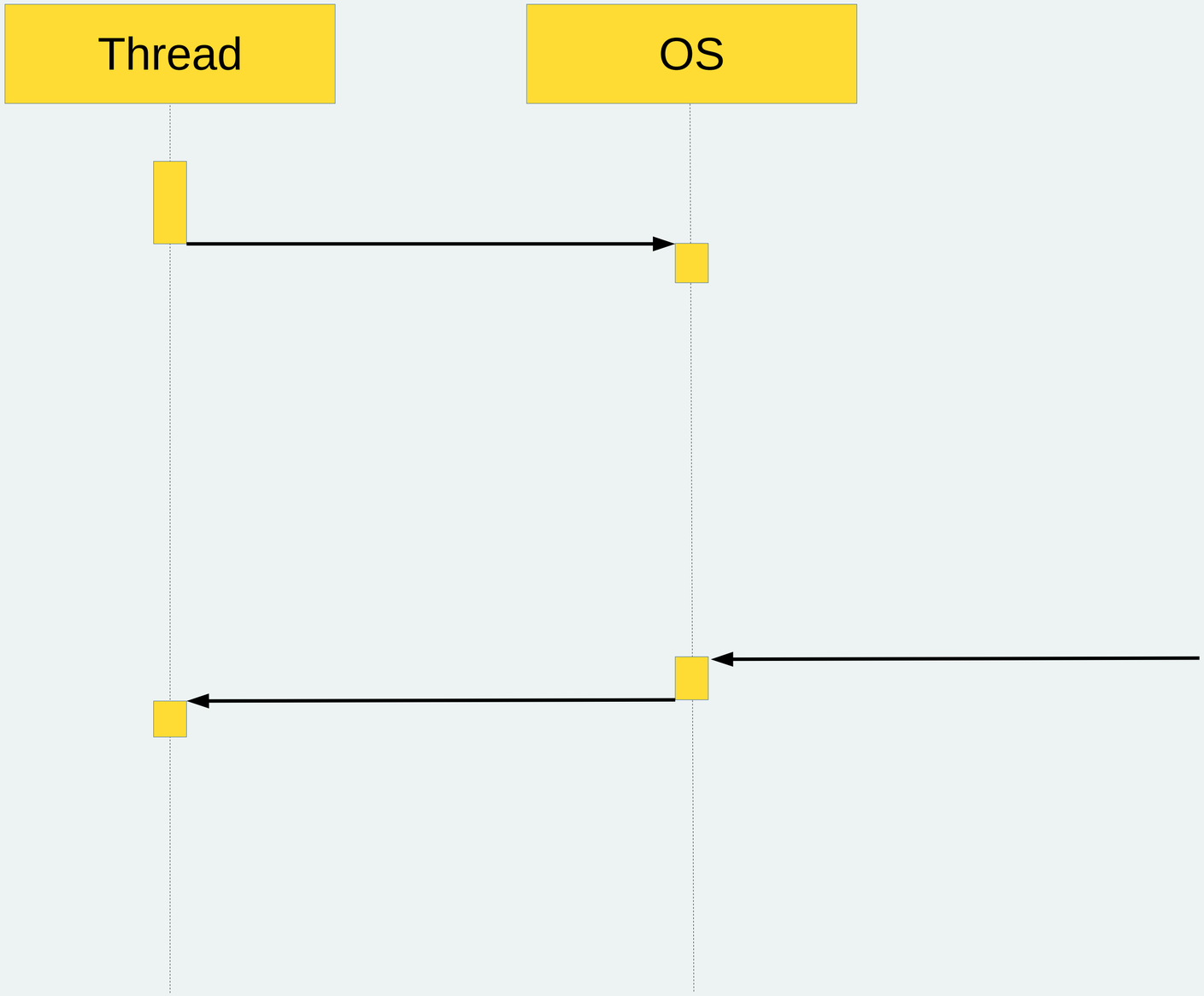
Hello world #2

```
// Асинхронное выполнение, std::thread переиспользуется
pg_cluster_->Execute(storages::postgres::ClusterHostType::kMaster,
                     "DELETE FROM key_value_table WHERE key=$1", key);
```

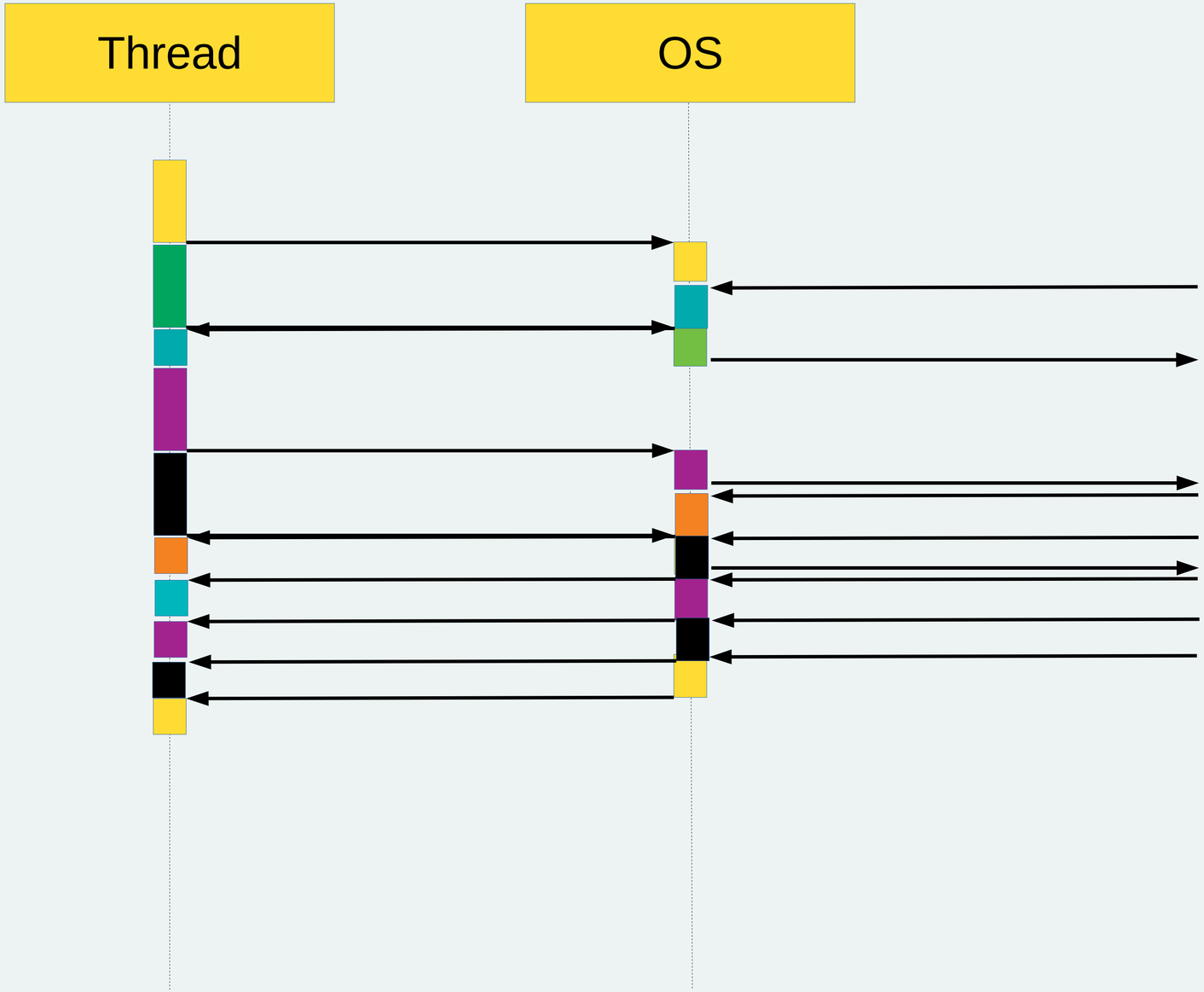
Обычный Request



Userver Request



Userver Request



Userver Request

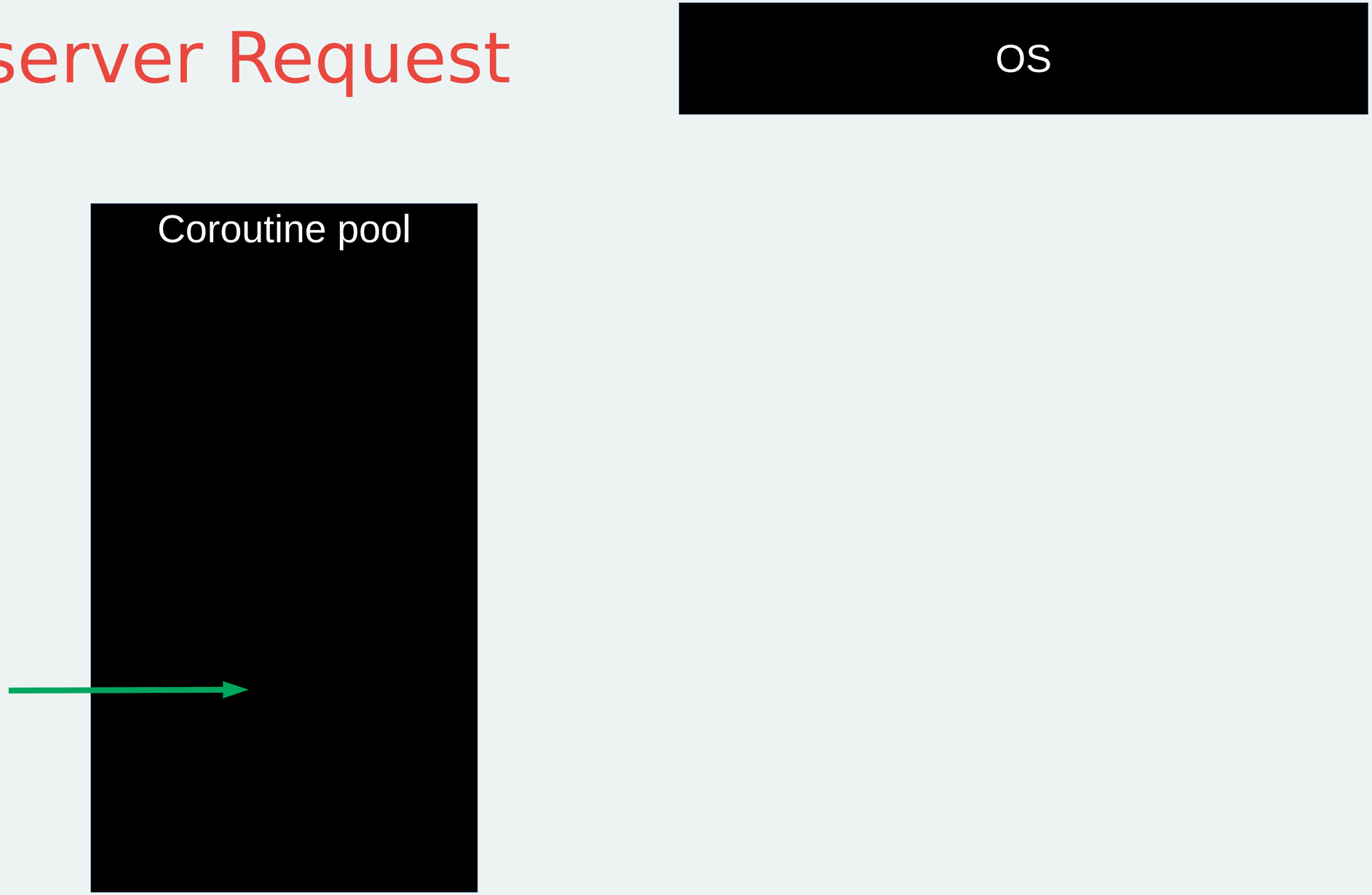
OS

Request

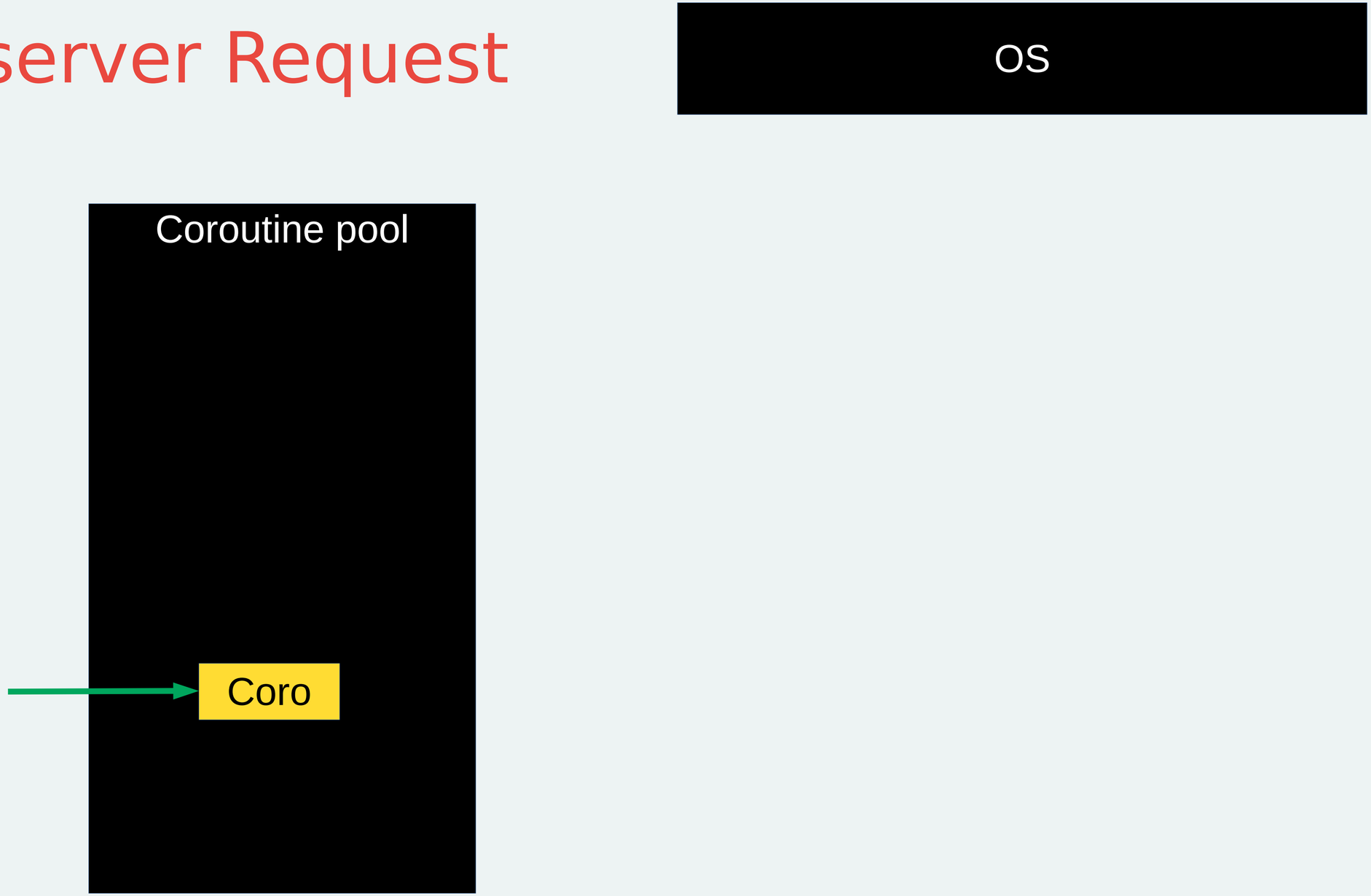
Userver Request

OS

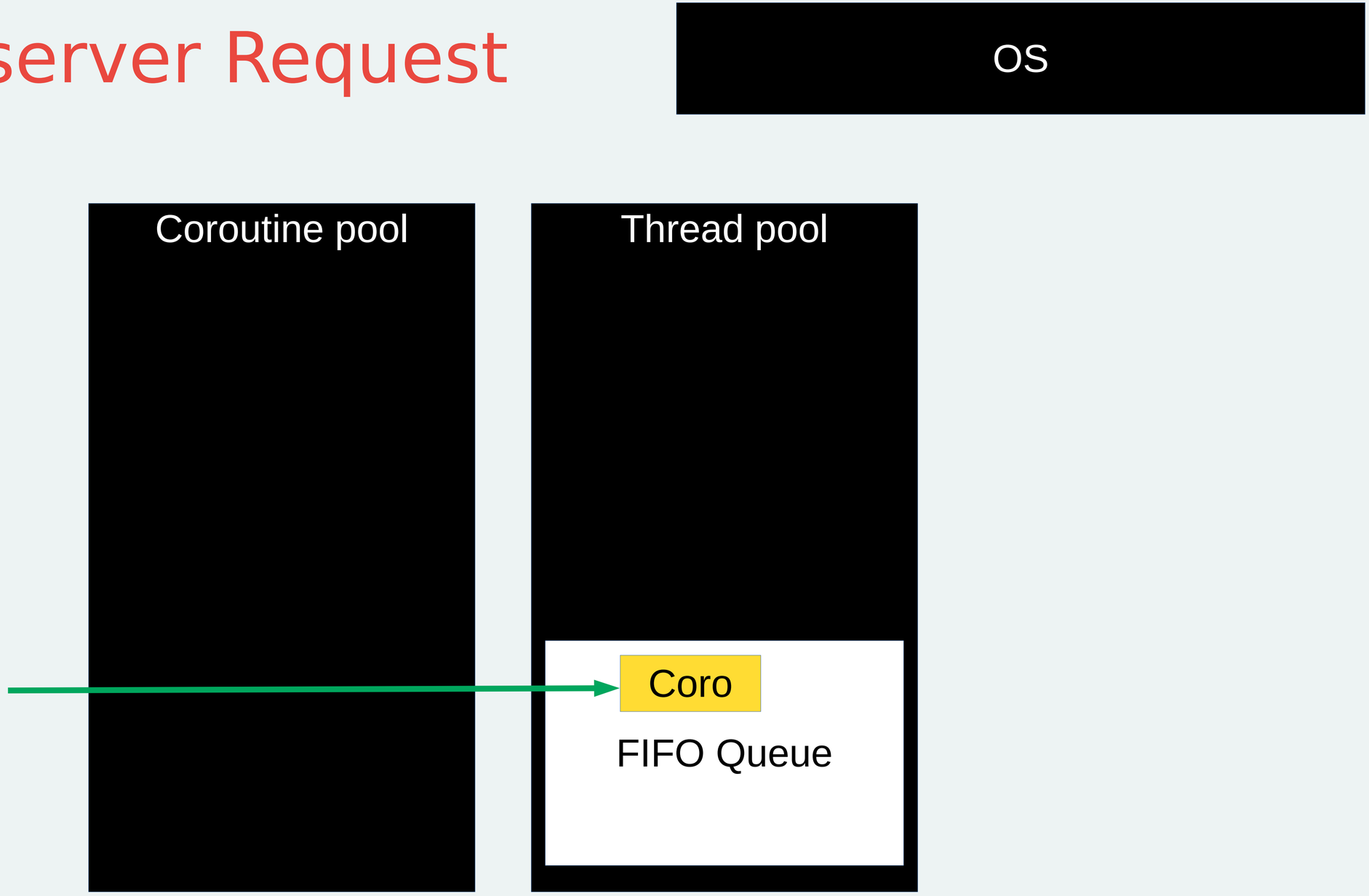
Userver Request



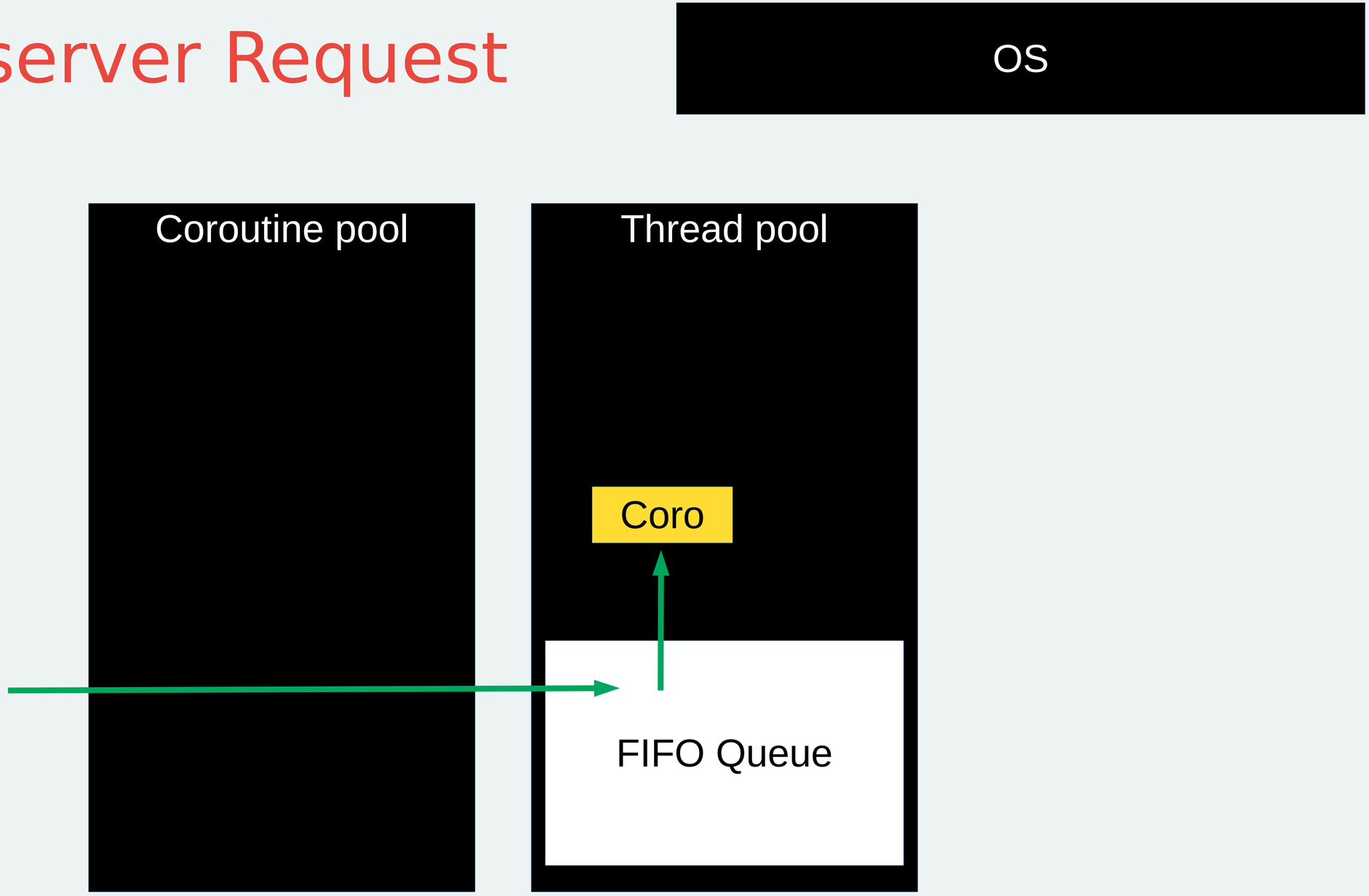
Userver Request



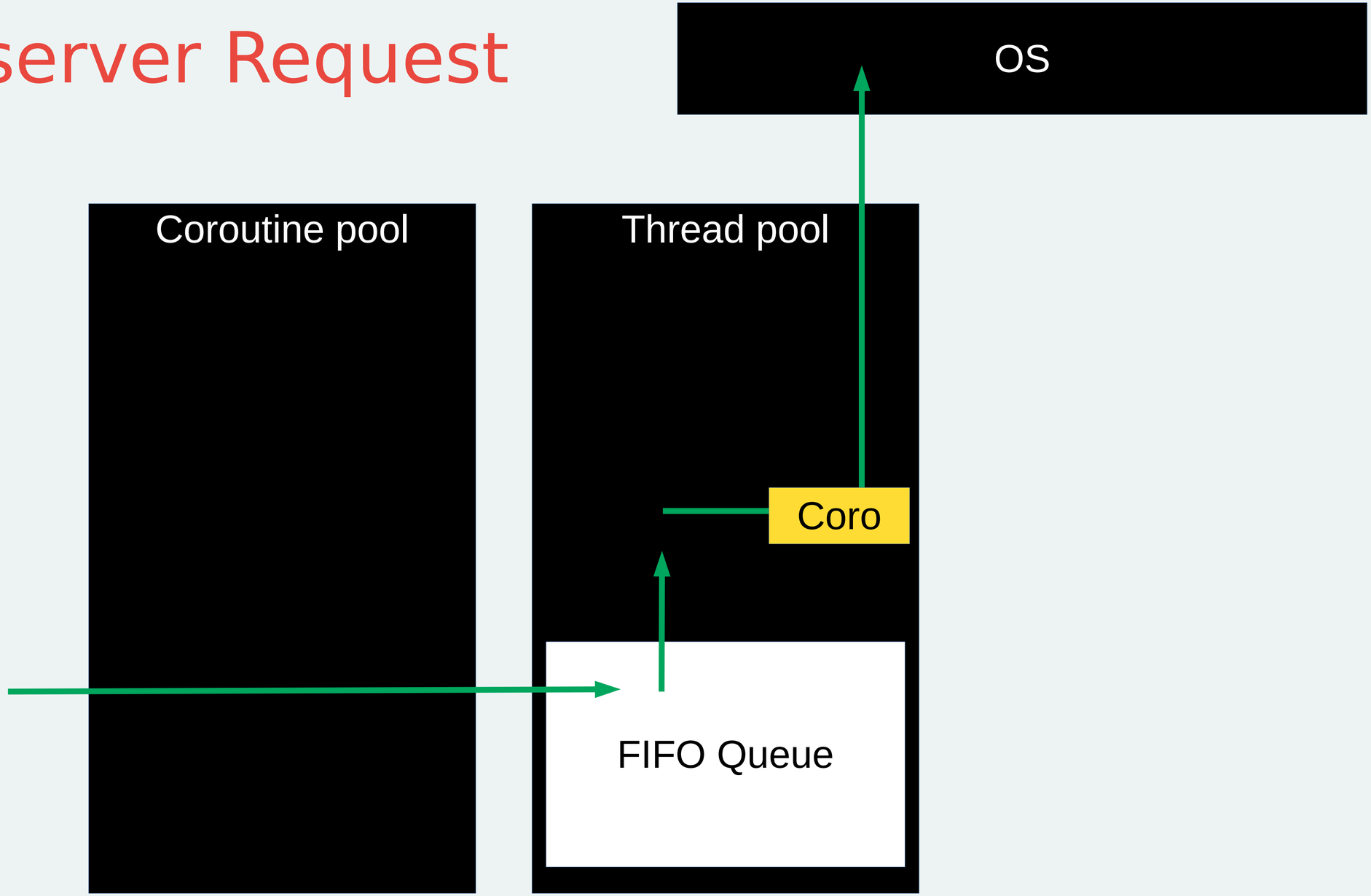
Userver Request



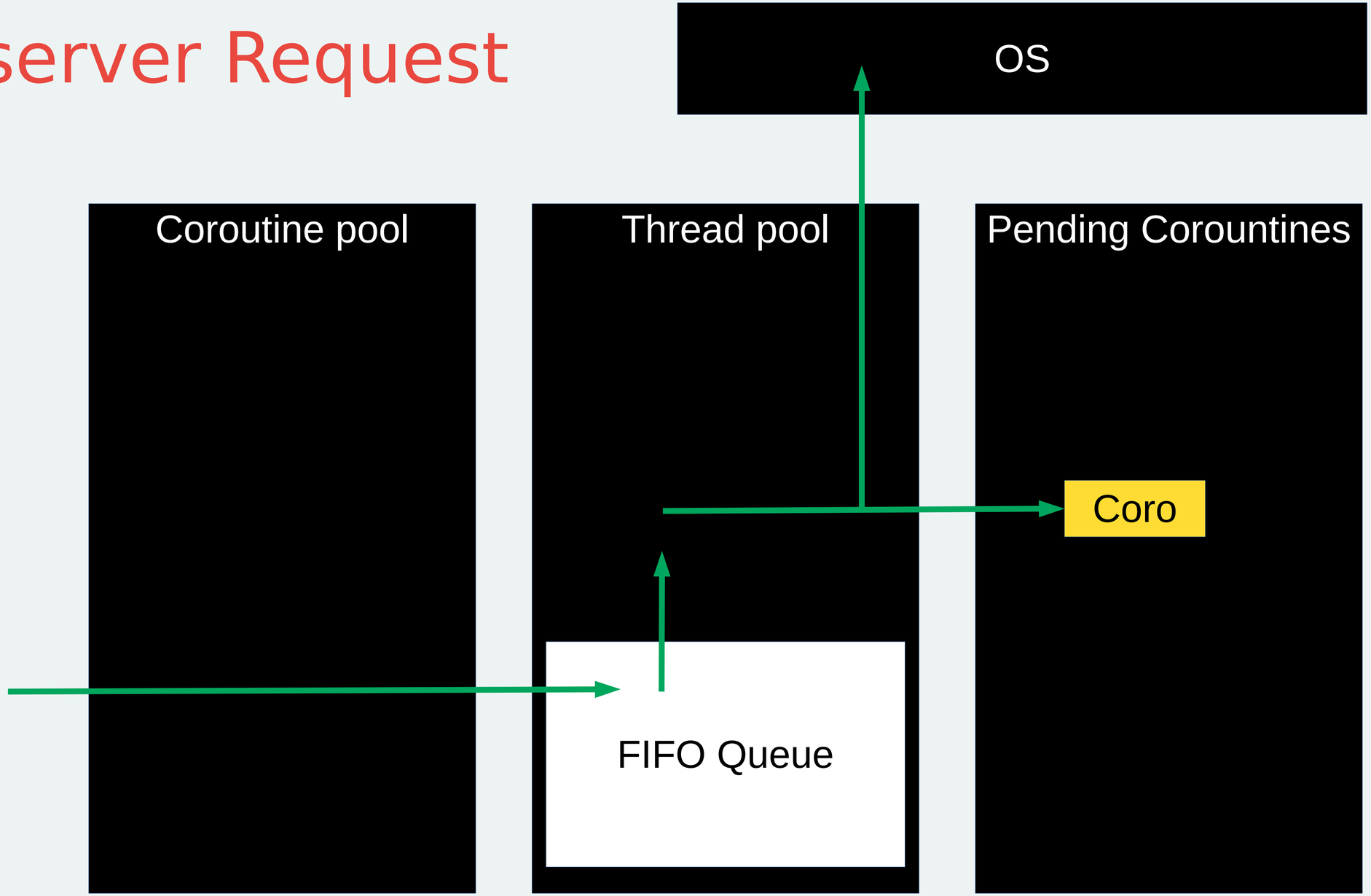
Userver Request



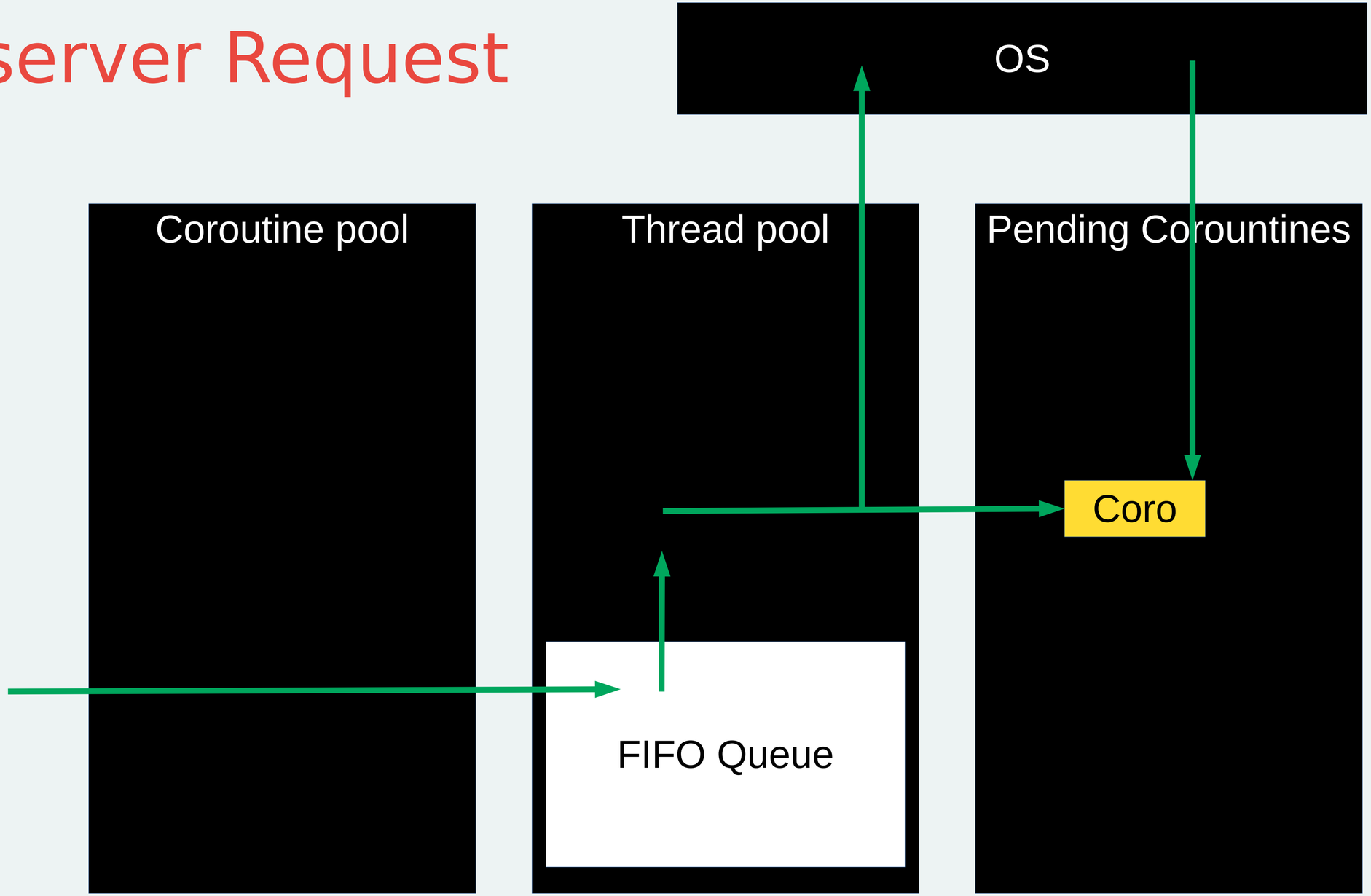
Userver Request



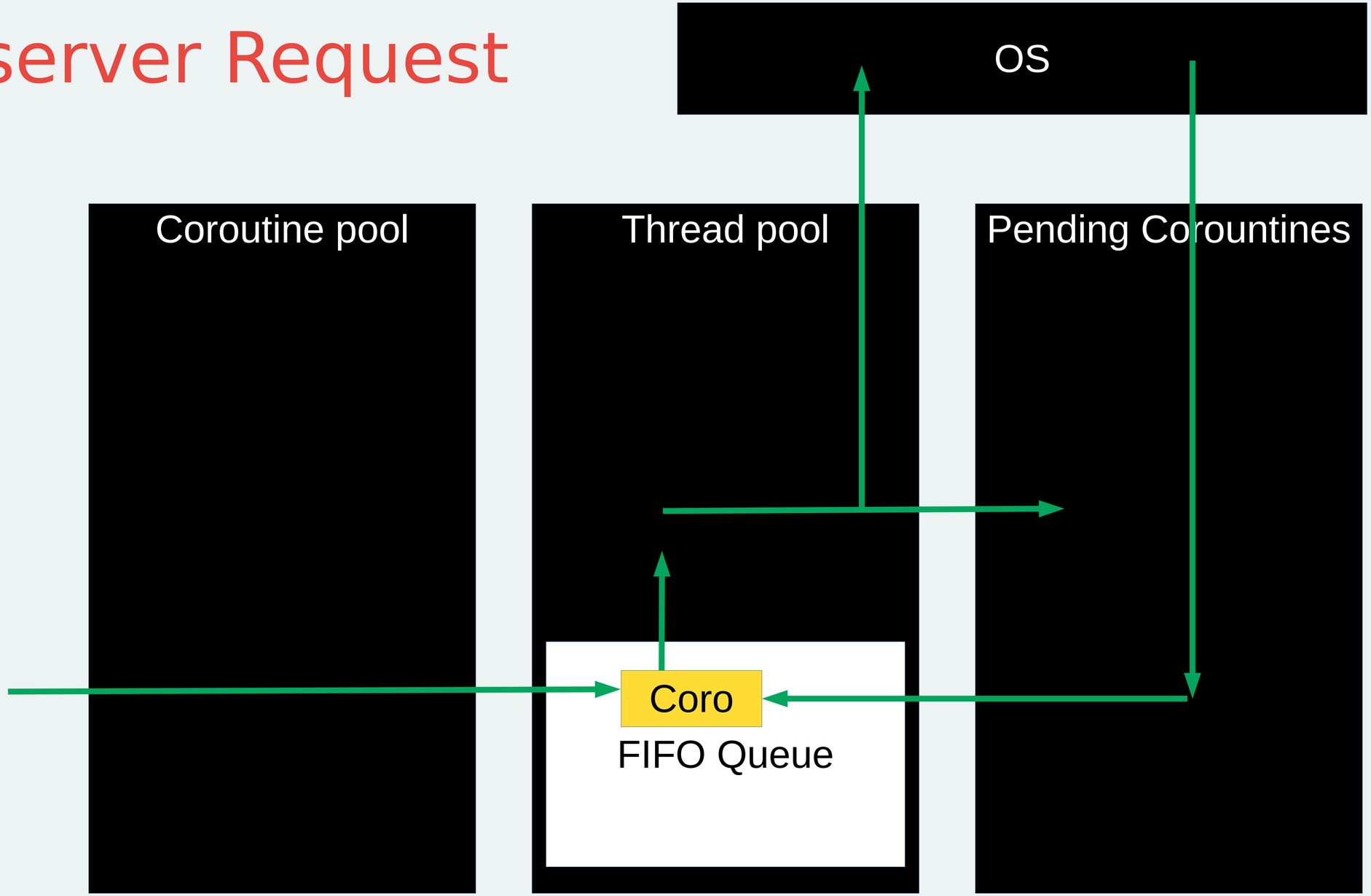
Userver Request



Userver Request



Userver Request



03

Динамические конфиги и RCU

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};


auto transaction = pg_cluster_->Begin("sample_transaction_insert_key_value", {});
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};


auto transaction = pg_cluster_->Begin("sample_transaction_insert_key_value", {});
```

Hello world #2

```
const storages::postgres::Query kSelectValue{
    "SELECT value FROM key_value_table WHERE key=$1",
    storages::postgres::Query::Name{"sample_select_value"},
};


auto transaction = pg_cluster_->Begin("sample_transaction_insert_key_value", {});
```

Dynamic config

```
"POSTGRES_QUERIES_COMMAND_CONTROL": {
  "sample_select_value": {
    "network_timeout_ms": 70,
    "statement_timeout_ms": 40
  },
  "sample_transaction_insert_key_value": {
    "network_timeout_ms": 200,
    "statement_timeout_ms": 150
  }
}
```

Dynamic Config

```
"POSTGRES_CONNECTION_POOL_SETTINGS": {
  "key-value-database": {
    "min_pool_size": 8,
    "max_pool_size": 15,
    "max_queue_size": 200
  }
},
"POSTGRES_STATEMENT_METRICS_SETTINGS": {
  "key-value-database": {
    "max_statement_metrics": 5
  }
},
```

Configs

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

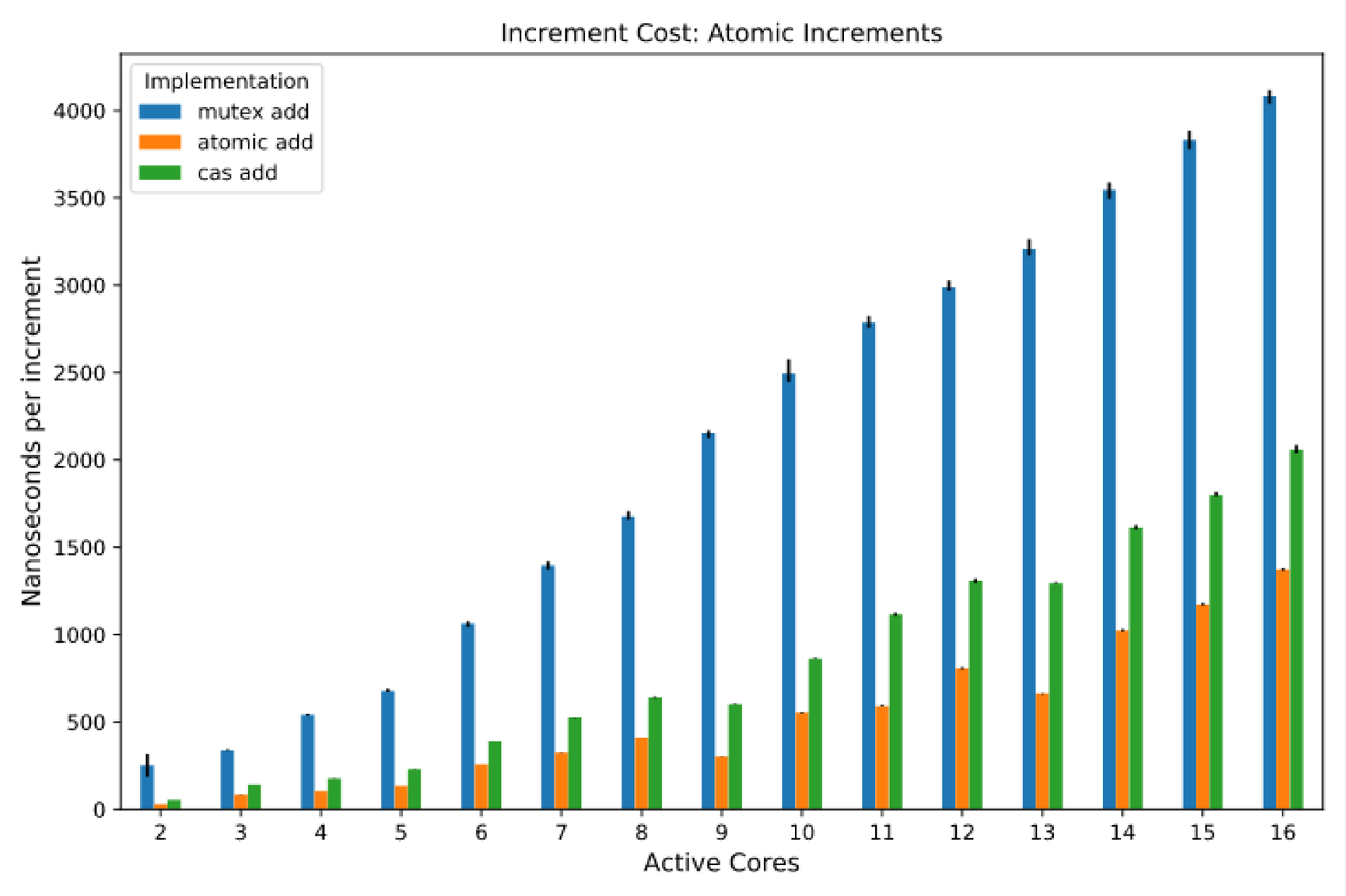
Configs

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

Configs

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

Hazard Pointer



Hazard Pointer

10k RPS

Hazard Pointer

10k RPS

- mutex: 4us * 10 000 * 2 == 80ms

Hazard Pointer

10k RPS

- mutex: 4us * 10 000 * 2 == 80ms
- atomic<shared_ptr>: 1us * 10 000 * 2 == 20ms

Hazard Pointer

10k RPS

- mutex: 4us * 10 000 * 2 == 80ms
- atomic<shared_ptr>: 1us * 10 000 * 2 == 20ms
- hazard pointer: 20ns * 10 000 * 2 == <1ms

Hazard Pointer

10k RPS

- mutex: 4us * 10 000 * 2 == 80ms
- atomic<shared_ptr>: 1us * 10 000 * 2 == 20ms
- hazard pointer: 20ns * 10 000 * 2 == <1ms (нет rwm операции на горячем пути!)

Configs

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

Configs

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

Configs

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

```
void LoggingConfigurator::OnConfigUpdate(const Snapshot& config) {
  tracing::Tracer::SetNoLogSpans(tracing::NoLogSpans{config[kNoLogSpans]});
}
```

03

Что ещё есть?

В userver есть всё для разработки

В userver есть всё для разработки

Асинхронные версии:

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB
- Redis

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB
- Redis
- HTTP

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач

В userver есть всё для разработки

Асинхронные версии:

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...
- Отмены, deadline propagation

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...
- Отмены, deadline propagation
- JSON/YAML/BSON/...

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...
- Отмены, deadline propagation
- JSON/YAML/BSON/...
- Кеши

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...
- Отмены, deadline propagation
- JSON/YAML/BSON/...
- Кеши
- Контейнеры

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...
- Отмены, deadline propagation
- JSON/YAML/BSON/...
- Кеши
- Контейнеры
- StrongTypedef, FastPimpl...

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...
- Отмены, deadline propagation
- JSON/YAML/BSON/...
- Кеши
- Контейнеры
- StrongTypedef, FastPimpl...
- Стректрейсы, календари...

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...
- Отмены, deadline propagation
- JSON/YAML/BSON/...
- Кеши
- Контейнеры
- StrongTypedef, FastPimpl...
- Стректрейсы, календари...
- Utest, death tests

В userver есть всё для разработки

Асинхронные версии

- PostgreSQL
- MongoDB
- Redis
- HTTP
- GRPC
- Sockets
- Resolver
- Примитивов синхронизации
- Запуск задач
- Таймеры

А ещё:

- Logging, tracing
- Статистика, перцентили...
- Отмены, deadline propagation
- JSON/YAML/BSON/...
- Кеши
- Контейнеры
- StrongTypedef, FastPimpl...
- Стректрейсы, календари...
- Utest, death tests
- Динамические конфиги

Итого

Итого

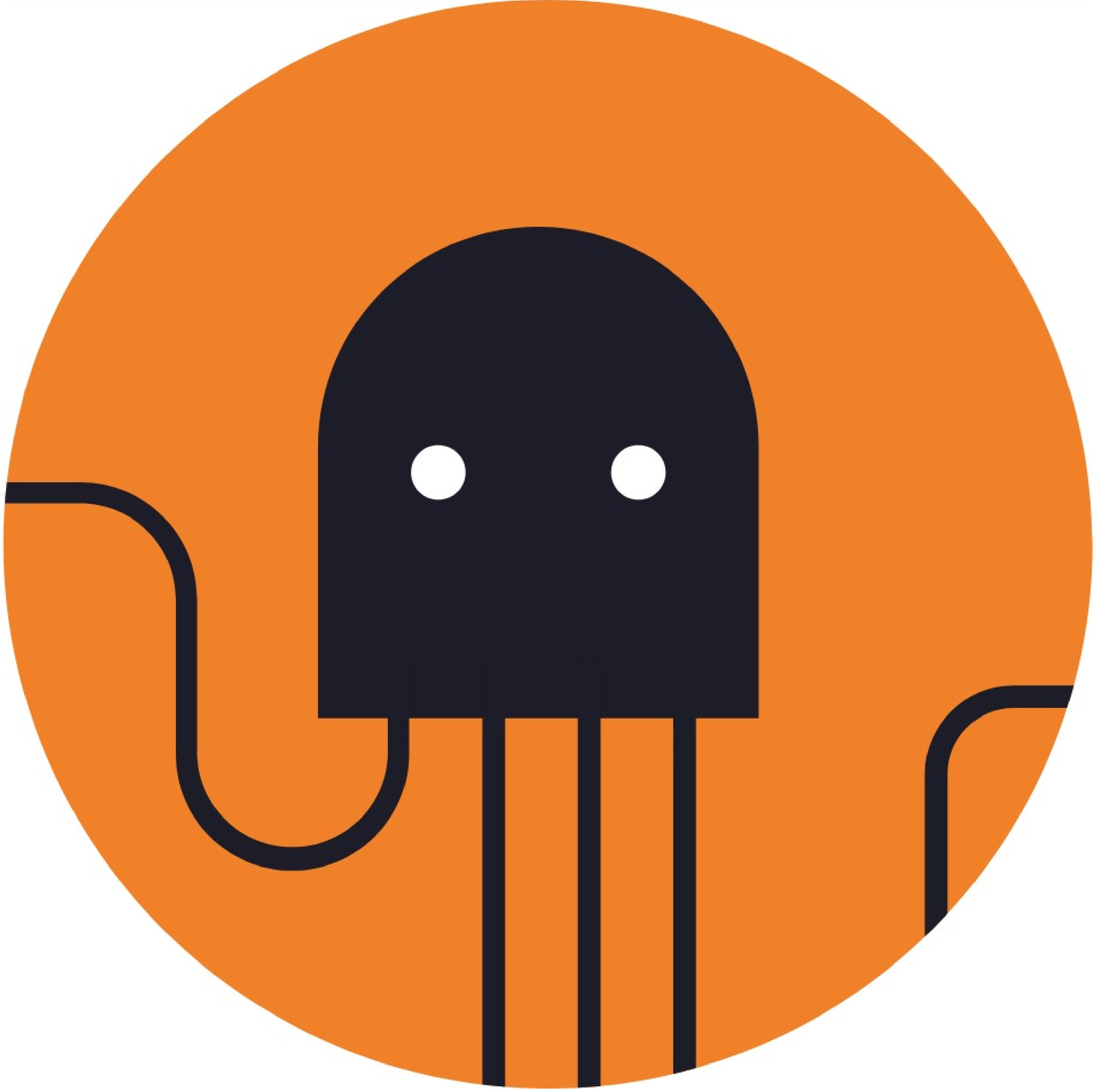
- Быстрая разработка новых сервисов

Итого

- Быстрая разработка новых сервисов
- Простой линейный код

Итого

- Быстрая разработка новых сервисов
- Простой линейный код
- Эффективность с точки зрения ресурсов



Yandex for developers *//>

YATALKS

Бэкенд

Спасибо

Антон Полухин

Эксперт-разработчик C++

antoshkka@gmail.com
@antoshkka