Приёмы ООП на примере реального проекта

Крюков Константин

Инженер по разработке ПО
YADRO

Введение: зачем и для кого?

Зачем доклад?

- Проект
- Архитектура
- ООП

Для кого доклад?

- Инженеры
- Архитекторы

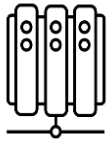
MeyerSAN: о дисках и СХД

Архитектура и паттерны

Ничто не бывает идеальным

Выводы

Кратко о предметной области: СХД и диски



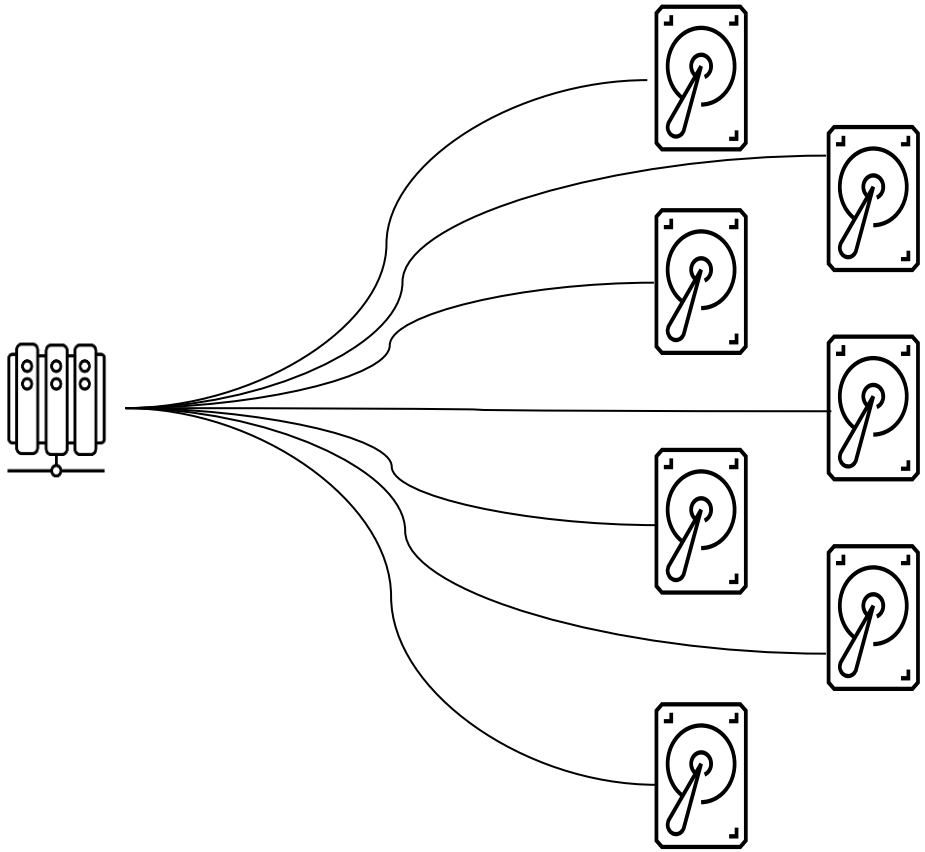
Системы хранения данных

Объединяют массивы дисков в единое хранилище, обеспечивающее надежность хранения и высокую скорость доступа.

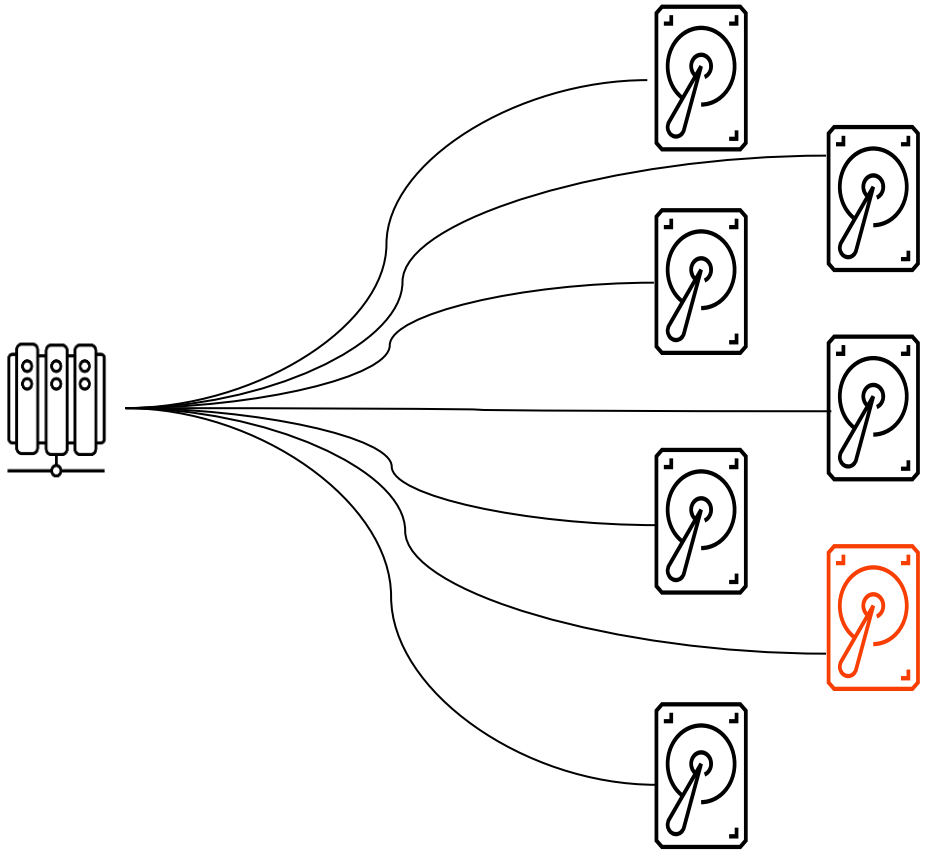
Диски - расходный материал

Средний срок службы одного жёсткого диска в составе СХД - 5 лет.
СХД детектируют проблемные диски, снижают на них нагрузку или исключают её совсем

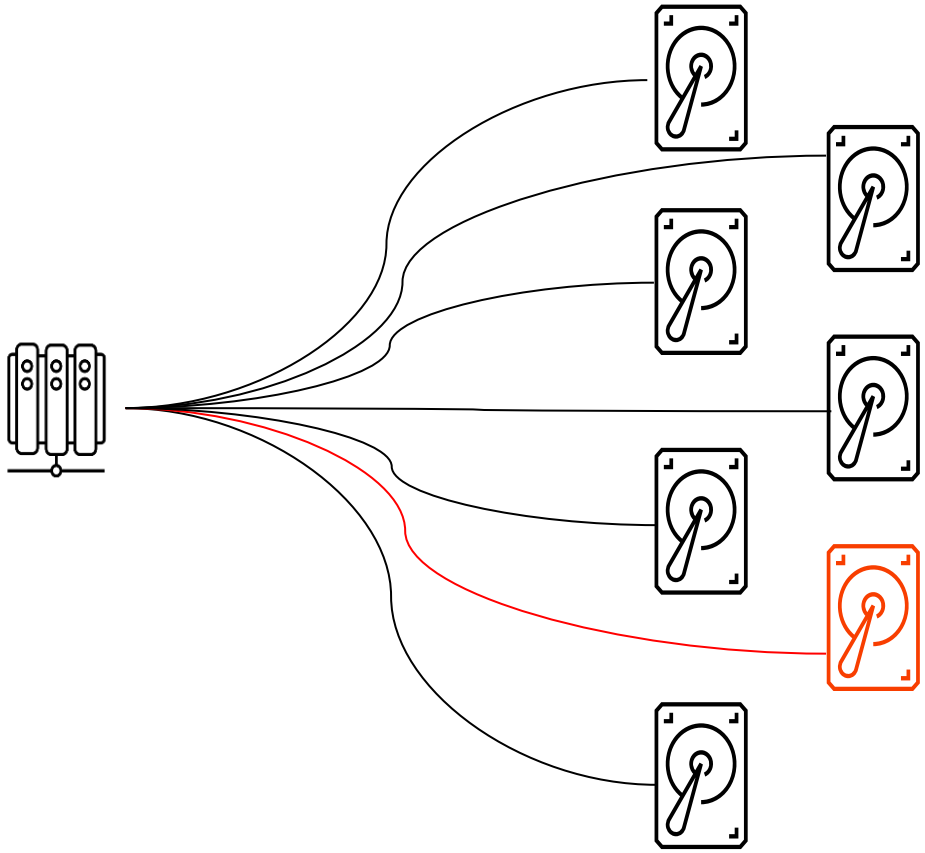
Кратко о предметной области: СХД и диски



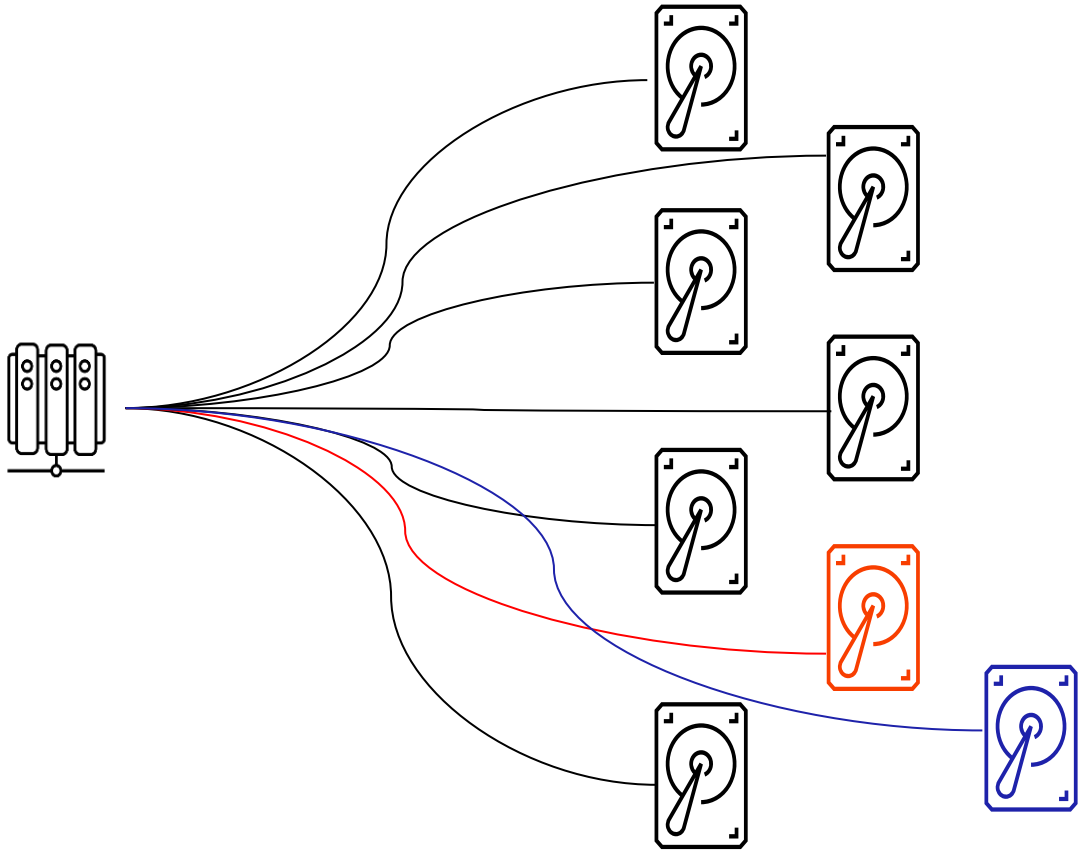
Кратко о предметной области: СХД и диски



Кратко о предметной области: СХД и диски

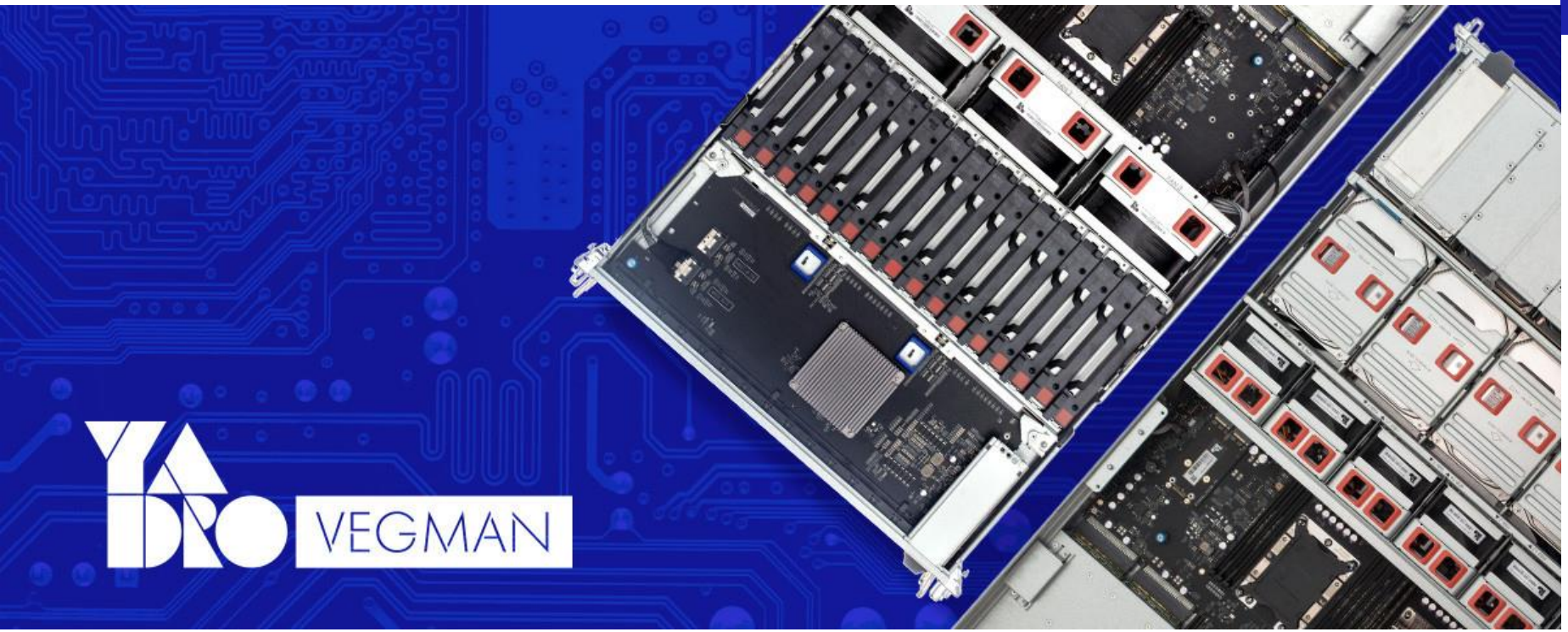


Кратко о предметной области: СХД и диски



Кратко о предметной области: MeyerSAN

Программно-аппаратный комплекс, основанный на сервере VEGMAN



Разработан для тестирования и валидации работы дисковых подсистем СХД

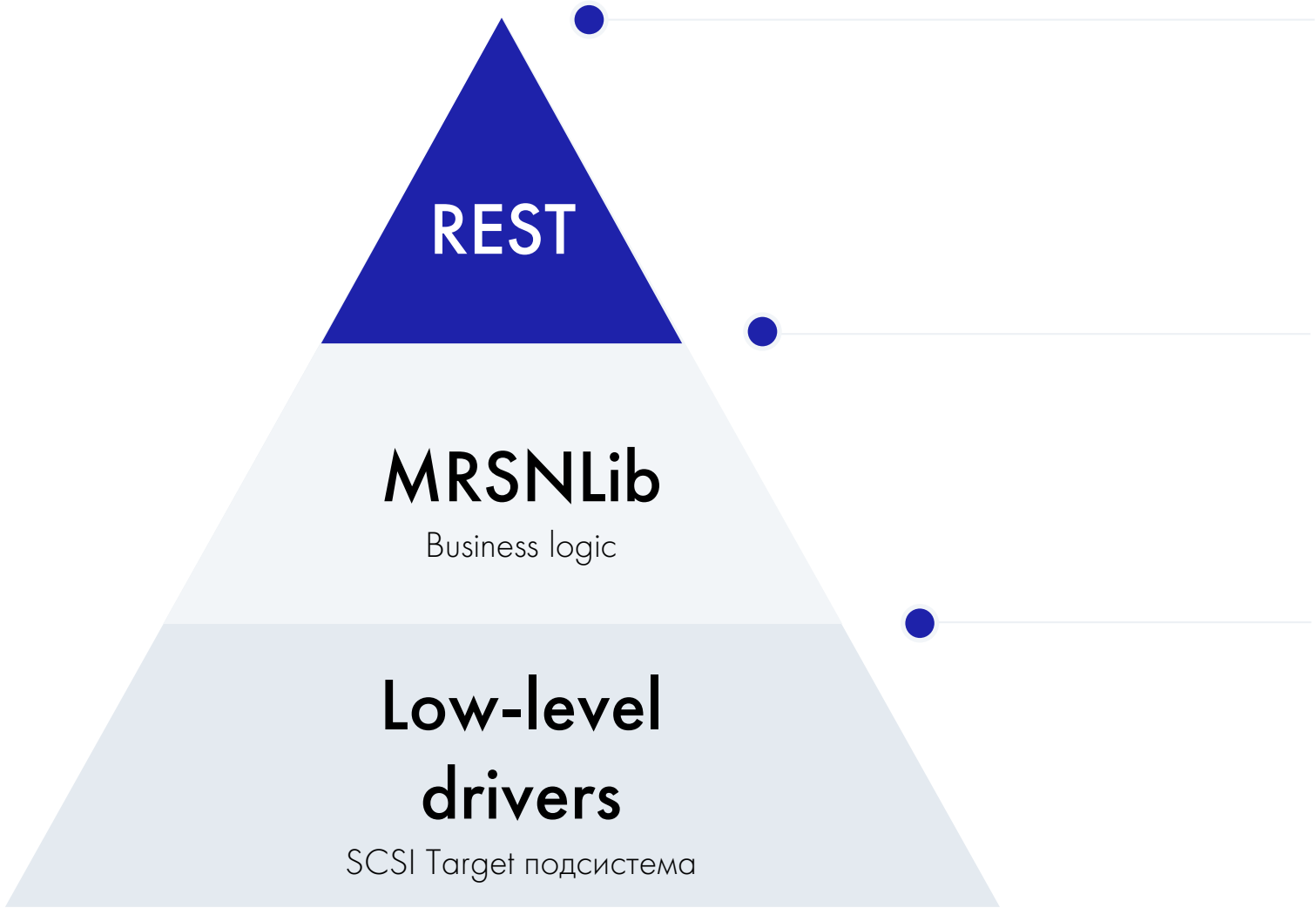
MeyerSAN: о дисках и СХД

Архитектура и паттерны

Ничто не бывает идеальным

Выводы

Программная архитектура MeyerSAN: общий обзор



MRSNMGMT (Python)

REST

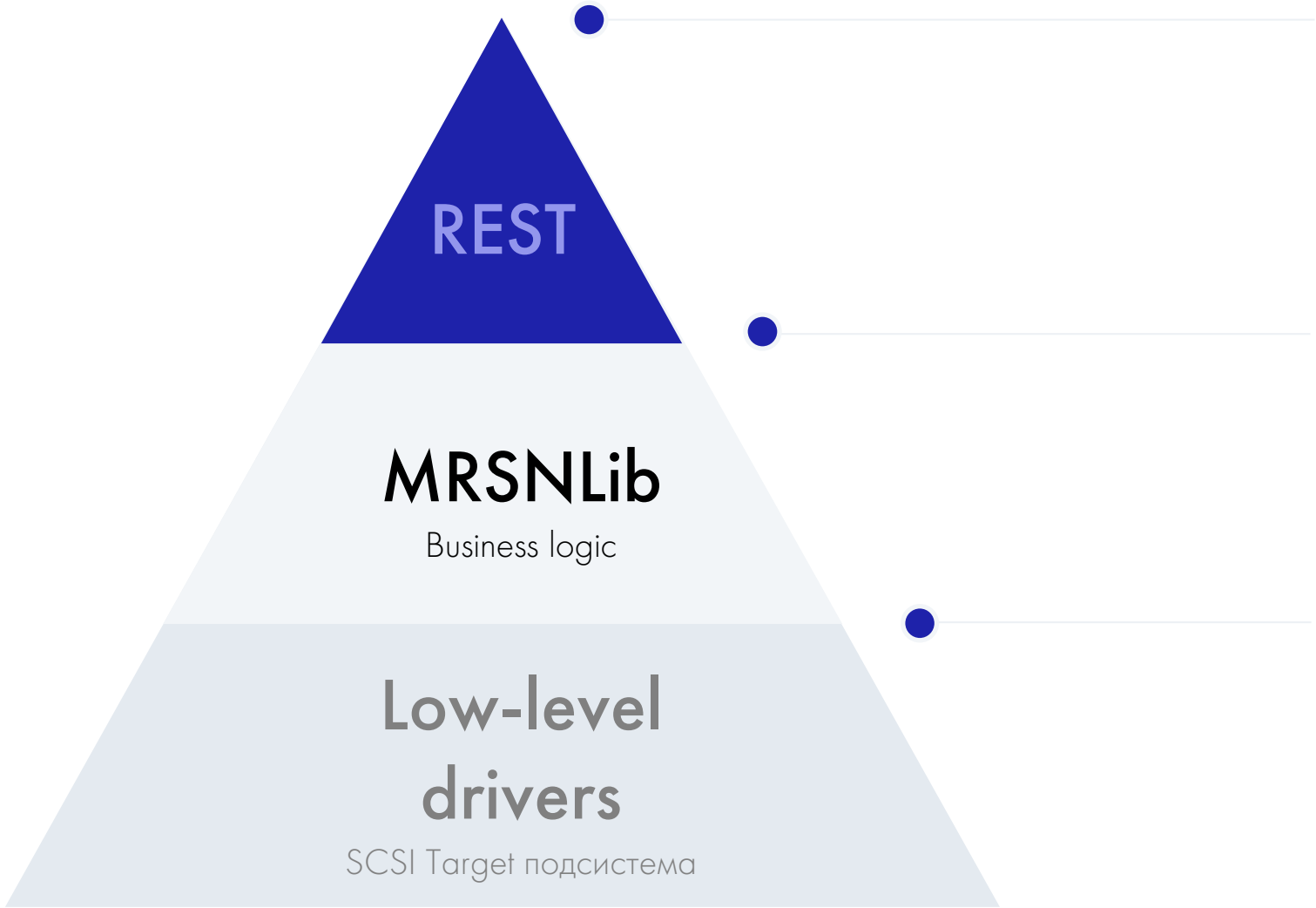
MRSNLib (C++23)

Реализует всю логику по обработке и модификации команд

STS (C)

Низкоуровневая реализация SCSI протоколов в виде target-подсистемы

Программная архитектура MeyerSAN: общий обзор



MRSNMGMT (Python)

REST

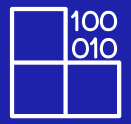
MRSNLib (C++23)

Реализует всю логику по обработке и модификации команд

STS (C)

Низкоуровневая реализация SCSI протоколов в виде target-подсистемы

Программная архитектура MeyerSAN: MRSNLib



Command

- Ключевая сущность в проекте
- Определяет команду к диску, которую необходимо обработать и на которую необходимо ответить

Modifier

- Сущность, ответственная за модификацию команды
- Позволяет внедрять ошибки в команды

Runtime

- Все остальные сущности
- Окружение, в котором происходит вся бизнес-логика
- Выполнение команд
- Применение модификаторов
- Подробно останавливаться не будем

Программная архитектура MeyerSAN: требования

01

Минимум времени для внедрения нового модификатора

Программная архитектура MeyerSAN: требования

01

Минимум времени для внедрения нового модификатора

02

Расширяемость в контексте возможных модификаторов

Программная архитектура MeyerSAN: требования

01

Минимум времени для внедрения нового модификатора

02

Расширяемость в контексте возможных модификаторов

03

Поддержка **базовых** и **протокол-специфичных** модификаторов

Программная архитектура MeyerSAN: требования

01

Минимум времени для внедрения нового модификатора

02

Расширяемость в контексте возможных модификаторов

03

Поддержка **базовых** и **протокол-специфичных** модификаторов

Для того, чтобы отвечать этим требованиям, мы воспользовались паттернами проектирования

Паттерны в MeyerSAN

01

Command

02

Decorator

03

Visitor

Паттерны в MeyerSAN: Command

Command.hpp:

```
class Command {
public:
    virtual OpCode opCode() const noexcept = 0;
    virtual void execute() = 0;
...
};
```

Паттерны в MeyerSAN: Command

Command.hpp:

```
class Command {
public:
    virtual OpCode opCode() const noexcept = 0;
    virtual void execute() = 0;
...
};
```

Паттерны в MeyerSAN: Command

ReadCommand.hpp:

```
class ReadCommand : public Command {
private:
    void execute() override {
        std::cout << "Do READ work!";
    }
};
```

WriteCommand.hpp:

```
class WriteCommand : public Command {
private:
    void execute() override {
        std::cout << "Do WRITE work!";
    }
};
```

Паттерны в MeyerSAN: Command

ReadCommand.hpp:

```
class ReadCommand : public Command {
private:
    void execute() override {
        std::cout << "Do READ work!";
    }
};
```

WriteCommand.hpp:

```
class WriteCommand : public Command {
private:
    void execute() override {
        std::cout << "Do WRITE work!";
    }
};
```

Паттерны в MeyerSAN: Command

main.cpp:

```
std::vector<sptr<Command>> getCommands();

int main() {
    for (auto cmd : getCommands())
        cmd->execute();
}
```

getCommands.cpp:

```
std::vector<sptr<Command>> getCommands() {
    auto cmds = getCommandsFromAPI();

    // filter known commands

    return std::vector<sptr<Command>>(cmds);
}
```

Паттерны в MeyerSAN: Command

main.cpp:

```
std::vector<sptr<Command>> getCommands();

int main() {
    for (auto cmd : getCommands())
        cmd->execute();
}
```

getCommands.cpp:

```
std::vector<sptr<Command>> getCommands() {
    auto cmds = getCommandsFromAPI();

    // filter known commands

    return std::vector<sptr<Command>>(cmds);
}
```

Паттерны в MeyerSAN: Command

01

Идеален для описания команды к диску

02

Независимость от рантайма

03

Открытость для расширения

Паттерны в MeyerSAN: Decorator

Паттерны в MeyerSAN: Наивный модификатор

Первичная реализация модификации

```
class Modifier {
public:
    virtual void modify(sptr<Command>) = 0;
    ...
};

int main() {
    std::vector<sptr<Command>> cmds{...};
    auto mod = make_sptr<DelayModifier>();

    for (auto cmd : cmds) {
        mod->modify(cmd);
        cmd->execute();
    }

    return 0;
}
```

Паттерны в MeyerSAN: Наивный модификатор

Первичная реализация модификации

```
class Modifier {
public:
    virtual void modify(sptr<Command>) = 0;
    ...
};

int main() {
    std::vector<sptr<Command>> cmds{...};
    auto mod = make_sptr<DelayModifier>();

    for (auto cmd : cmds) {
        mod->modify(cmd);
        cmd->execute();
    }

    return 0;
}
```

Паттерны в MeyerSAN: Наивный модификатор

Первичная реализация модификации

```
class Modifier {
public:
    virtual void modify(sptr<Command>) = 0;
    ...
};

int main() {
    std::vector<sptr<Command>> cmds{...};
    auto mod = make_sptr<DelayModifier>();

    for (auto cmd : cmds) {
        mod->modify(cmd);
        cmd->execute();
    }

    return 0;
}
```

Паттерны в MeyerSAN: Наивный модификатор

Первичная реализация модификации

```
class Modifier {
public:
    virtual void modify(sptr<Command>) = 0;
    ...
};

int main() {
    std::vector<sptr<Command>> cmds{...};
    auto mod = make_sptr<DelayModifier>();

    for (auto cmd : cmds) {
        mod->modify(cmd); // DelayModifier: modify before
        cmd->execute();
    }

    return 0;
}
```

Паттерны в MeyerSAN: Наивный модификатор

Первичная реализация модификации

```
class Modifier {
public:
    virtual void modify(sptr<Command>) = 0;
    ...
};

int main() {
    std::vector<sptr<Command>> cmds{...};
    auto mod = make_sptr<ReadCorruptModifier>();

    for (auto cmd : cmds) {
        cmd->execute();
        mod->modify(cmd); // ReadCorrupt: modify after
    }

    return 0;
}
```

Паттерны в MeyerSAN: Decorator

Modifier.hpp:

```
class Modifier {
public:
    virtual sptr<Command> modify(sptr<Command>) = 0;
    ...
};
```

Паттерны в MeyerSAN: Decorator

Modifier.hpp:

```
class Modifier {
public:
    virtual sptr<Command> modify(sptr<Command>) = 0;
    ...
};
```

Паттерны в MeyerSAN: Decorator

InquiryCorruptModifier.hpp:

```
class InquiryCorruptModifier : public Modifier {
private:
    class CorruptedInquiryCommand : public Command {
        sptr<Command> original_;
        ...
        void execute() override {
            // modify before (delay)
            original_->execute(); // optional call
            // modify after (corrupt)
        }
    };
public:
    sptr<Command> modify(sptr<Command> cmd) override {
        if (cmd->opCode() == OpCode::INQUIRY)
            return make_sptr<CorruptedInquiryCommand>(std::move(cmd));
        return cmd;
    }
};
```

Паттерны в MeyerSAN: Decorator

InquiryCorruptModifier.hpp:

```
class InquiryCorruptModifier : public Modifier {
private:
    class CorruptedInquiryCommand : public Command {
        sptr<Command> original_;
        ...
        void execute() override {
            // modify before (delay)
            original_->execute(); // optional call
            // modify after (corrupt)
        }
    };
public:
    sptr<Command> modify(sptr<Command> cmd) override {
        if (cmd->opCode() == OpCode::INQUIRY)
            return make_sptr<CorruptedInquiryCommand>(std::move(cmd));
        return cmd;
    }
};
```

Паттерны в MeyerSAN: Decorator

InquiryCorruptModifier.hpp:

```
class InquiryCorruptModifier : public Modifier {
private:
    class CorruptedInquiryCommand : public Command {
        sptr<Command> original_;
        ...
        void execute() override {
            // modify before (delay)
            original_->execute(); // optional call
            // modify after (corrupt)
        }
    };
public:
    sptr<Command> modify(sptr<Command> cmd) override {
        if (cmd->opCode() == OpCode::INQUIRY)
            return make_sptr<CorruptedInquiryCommand>(std::move(cmd));
        return cmd;
    }
};
```

Паттерны в MeyerSAN: Decorator

InquiryCorruptModifier.hpp:

```
class InquiryCorruptModifier : public Modifier {
private:
    class CorruptedInquiryCommand : public Command {
        sptr<Command> original_;
        ...
        void execute() override {
            // modify before (delay)
            original_->execute(); // optional call
            // modify after (corrupt)
        }
    };
public:
    sptr<Command> modify(sptr<Command> cmd) override {
        if (cmd->opCode() == OpCode::INQUIRY)
            return make_sptr<CorruptedInquiryCommand>(std::move(cmd));
        return cmd;
    }
};
```

Паттерны в MeyerSAN: Decorator

InquiryCorruptModifier.hpp:

```
class InquiryCorruptModifier : public Modifier {
private:
    class CorruptedInquiryCommand : public Command {
        sptr<Command> original_;
        ...
        void execute() override {
            // modify before (delay)
            original_->execute(); // optional call
            // modify after (corrupt)
        }
    };
public:
    sptr<Command> modify(sptr<Command> cmd) override {
        if (cmd->opCode() == OpCode::INQUIRY)
            return make_sptr<CorruptedInquiryCommand>(std::move(cmd));
        return cmd;
    }
};
```

Паттерны в MeyerSAN: Decorator

InquiryCorruptModifier.hpp:

```
class InquiryCorruptModifier : public Modifier {
private:
    class CorruptedInquiryCommand : public Command {
        sptr<Command> original_;
        ...
        void execute() override {
            // modify before (delay)
            original_->execute(); // optional call
            // modify after (corrupt)
        }
    };
public:
    sptr<Command> modify(sptr<Command> cmd) override {
        if (cmd->opCode() == OpCode::INQUIRY)
            return make_sptr<CorruptedInquiryCommand>(std::move(cmd));
        return cmd;
    }
};
```

Паттерны в MeyerSAN: Decorator

InquiryCorruptModifier.hpp:

```
class InquiryCorruptModifier : public Modifier {
private:
    class CorruptedInquiryCommand : public Command {
        sptr<Command> original_;
        ...
        void execute() override {
            // modify before (delay)
            original_->execute(); // optional call
            // modify after (corrupt)
        }
    };
public:
    sptr<Command> modify(sptr<Command> cmd) override {
        if (cmd->opCode() == OpCode::INQUIRY)
            return make_sptr<CorruptedInquiryCommand>(std::move(cmd));
        return cmd;
    }
};
```

Паттерны в MeyerSAN: Decorator

01

Модификация команд через обёртку

02

Поддержка множественной модификации

03

Гибкость модификации

Паттерны в MeyerSAN: Visitor

Command.hpp:

Паттерны в MeyerSAN: Visitor

Command.hpp:

```
class Command {
public:
    virtual OpCode opCode() const noexcept = 0;
    virtual void execute() = 0;
...
};
```

Паттерны в MeyerSAN: Visitor

Command.hpp:

```
class Command {
public:
    virtual OpCode opCode() const noexcept = 0;
    virtual void execute() = 0;
...
};
```

Паттерны в MeyerSAN: Visitor

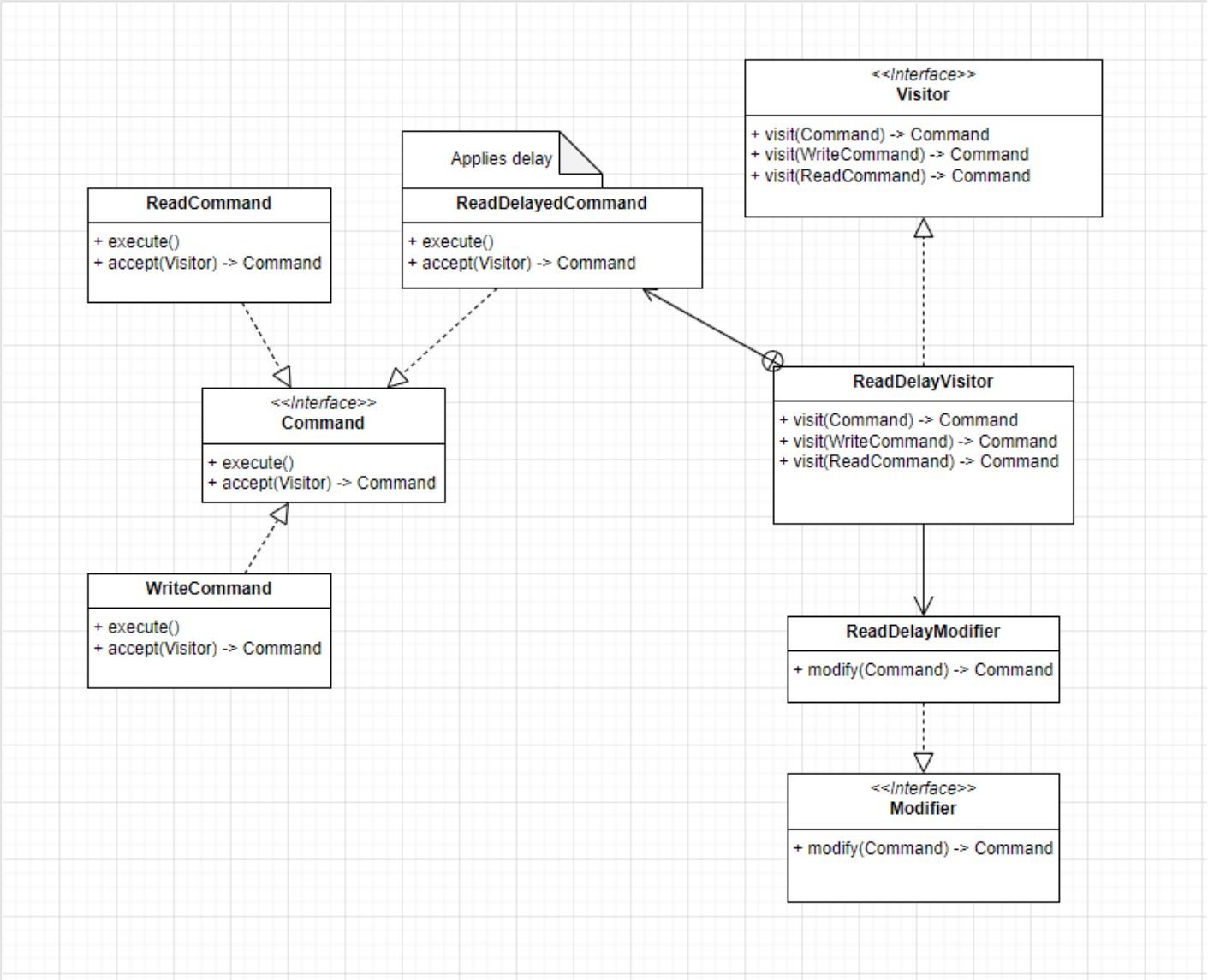
Command.hpp:

```
class Command {
public:
    virtual OpCode opCode() const noexcept = 0;
    virtual void execute() = 0;
...
};
```

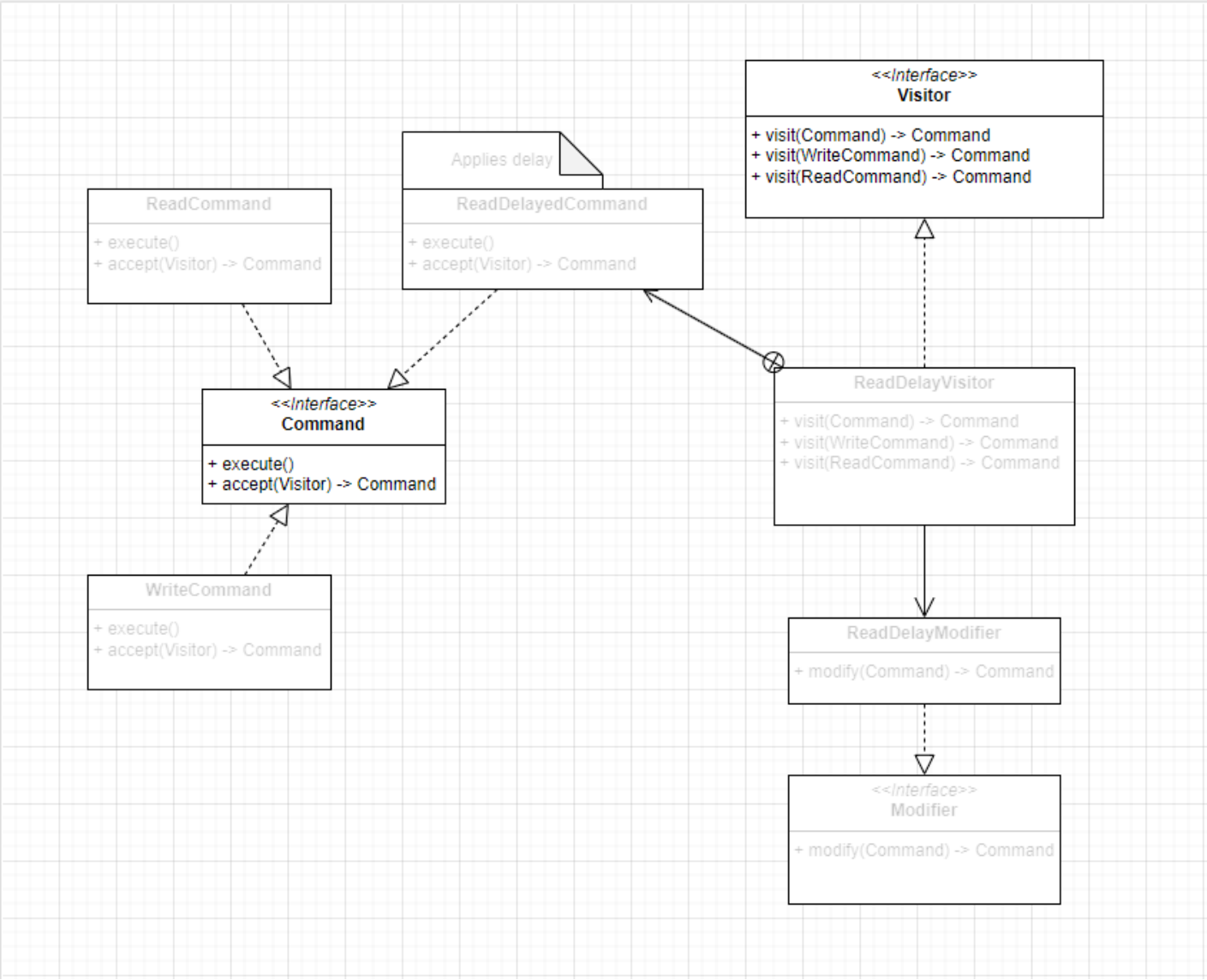
ReadCorruptModifier.hpp:

```
class ReadCorruptModifier : public Modifier {
...
public:
    sptr<Command> modify(sptr<Command> cmd) override {
        if (cmd->opCode() == OpCode::READ10) // ???
            return make_sptr<CorruptedReadCommand>(std::move(cmd));
        return cmd;
    }
};
```

Паттерны в MeyerSAN: Visitor в картинках



Паттерны в MeyerSAN: Visitor в картинках

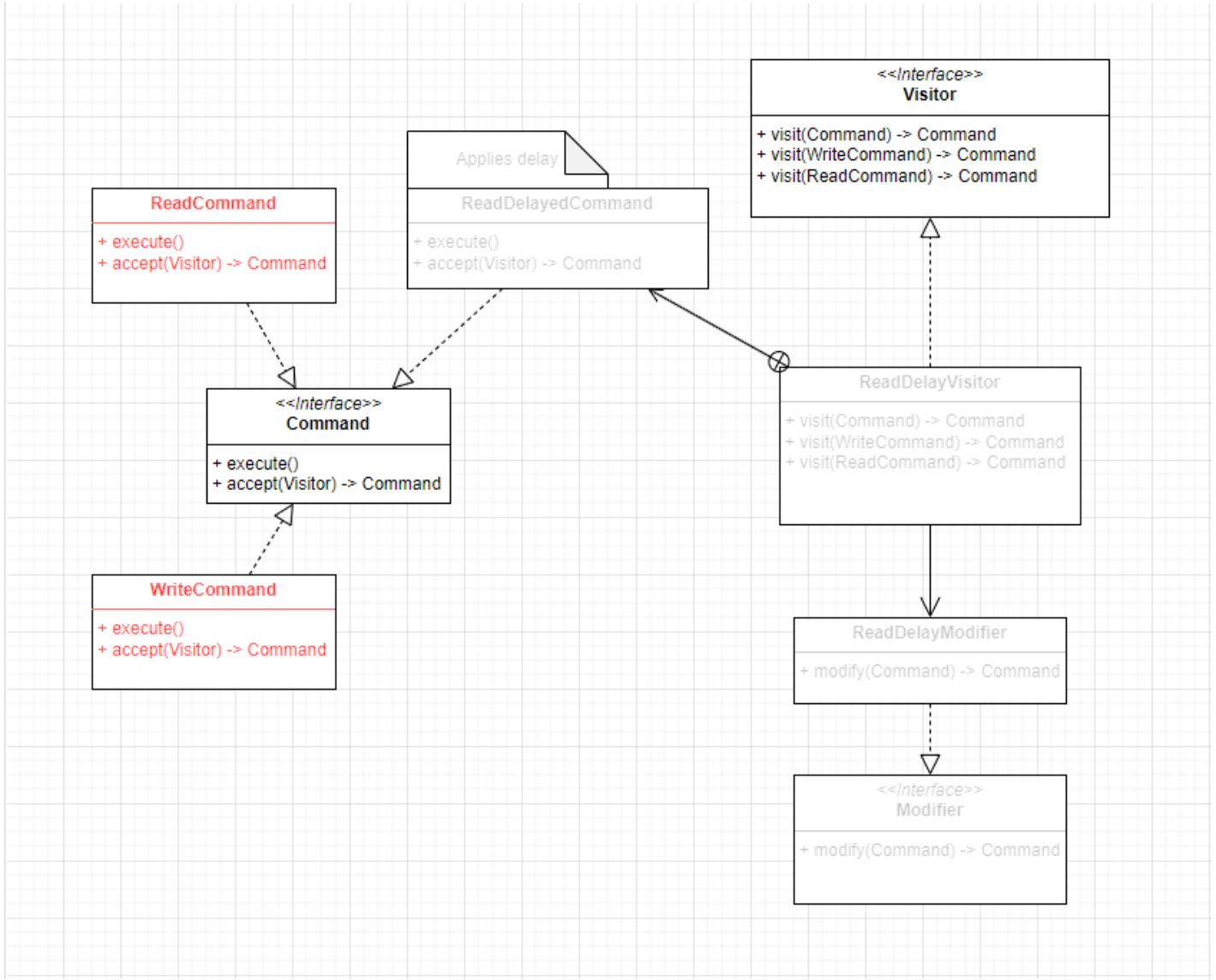


Паттерны в MeyerSAN: Visitor

Command.hpp:

```
class Command : public std::enable_shared_from_this<Command> {
public:
    virtual void execute() = 0;
    virtual sptr<Command> accept(Visitor&) = 0;
    ...
};

class ReadCommand;
class WriteCommand;
...

class Visitor {
public:
    virtual sptr<Command> visit(sptr<Command> cmd) = 0;
    virtual sptr<Command> visit(sptr<ReadCommand> cmd) = 0;
    virtual sptr<Command> visit(sptr<WriteCommand> cmd) = 0;
};
```

Паттерны в MeyerSAN: Visitor

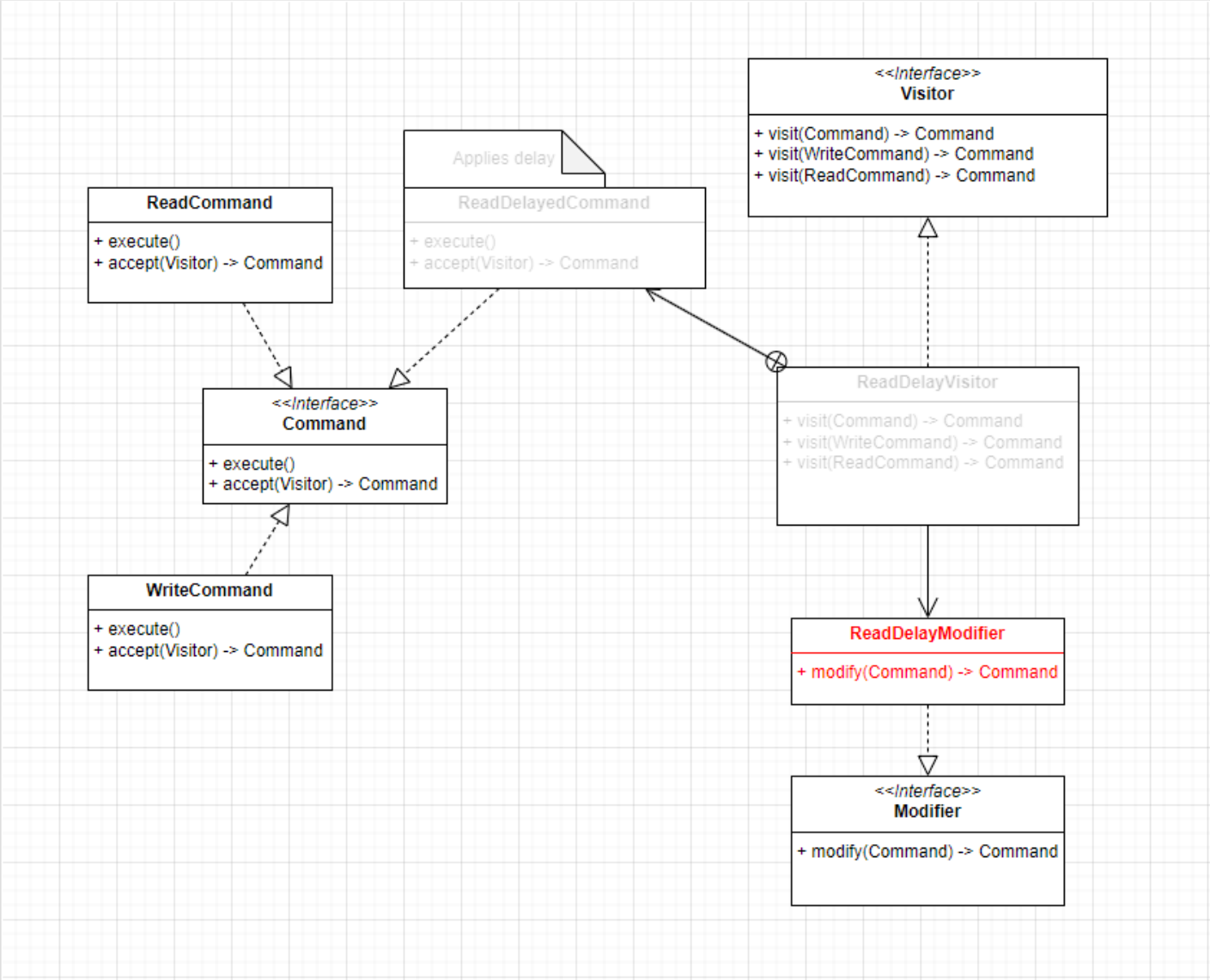


Паттерны в MeyerSAN: Visitor

IOCommand.hpp

```
class ReadCommand : public Command {
public:
    void execute() override { std::cout << “ReadCommand” << std::endl; }
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(
            std::static_pointer_cast<ReadCommand>(shared_from_this())
        );
    }
};

class WriteCommand : public Command {
public:
    void execute() override { std::cout << “WriteCommand << std::endl; }
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(
            std::static_pointer_cast<WriteCommand>(shared_from_this())
        );
    }
};
```

Паттерны в MeyerSAN: Visitor в картинках

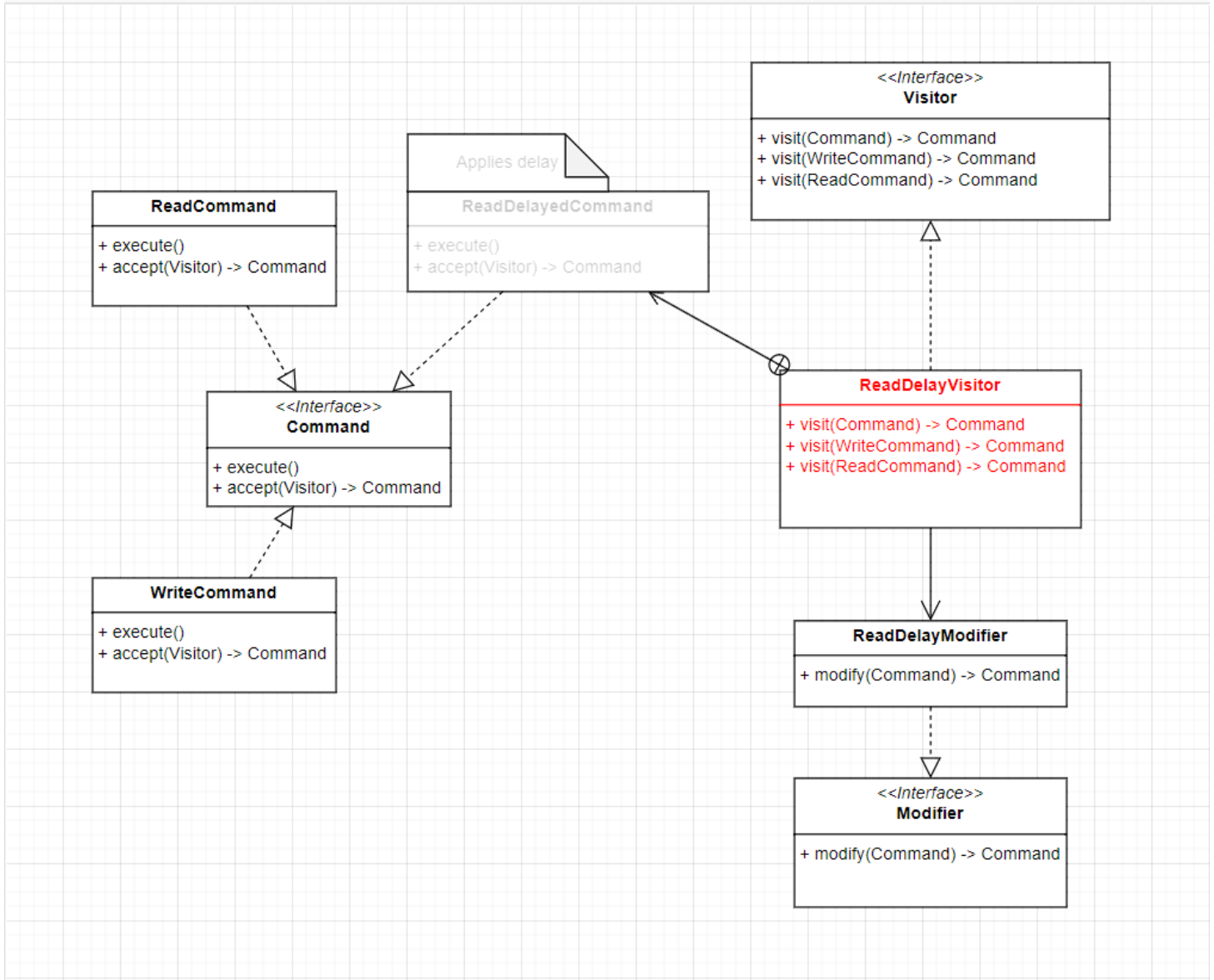


Паттерны в MeyerSAN: Visitor

ReadDelayModifier.hpp

```
class ReadDelayModifier : public Modifier {
    sptr<Command> modify(sptr<Command> cmd) override {
        ReadDelayVisitor visitor; // stateless

        return cmd->accept(visitor);
    }
};
```

Паттерны в MeyerSAN: Visitor в картинках



Паттерны в MeyerSAN: Visitor

ReadDelayVisitor.hpp

```
class ReadDelayVisitor : public Visitor {
    class DelayedReadCommand;

public:
    sptr<Command> visit(sptr<Command> cmd);
    sptr<Command> visit(sptr<ReadCommand> cmd);
    sptr<Command> visit(sptr<WriteCommand> cmd);
};
```

Паттерны в MeyerSAN: Visitor

ReadDelayVisitor.hpp

```
class ReadDelayVisitor : public Visitor {
    class DelayedReadCommand;

public:
    sptr<Command> visit(sptr<Command> cmd);
    sptr<Command> visit(sptr<ReadCommand> cmd);
    sptr<Command> visit(sptr<WriteCommand> cmd);
};
```

Паттерны в MeyerSAN: Visitor

ReadDelayVisitor.cpp

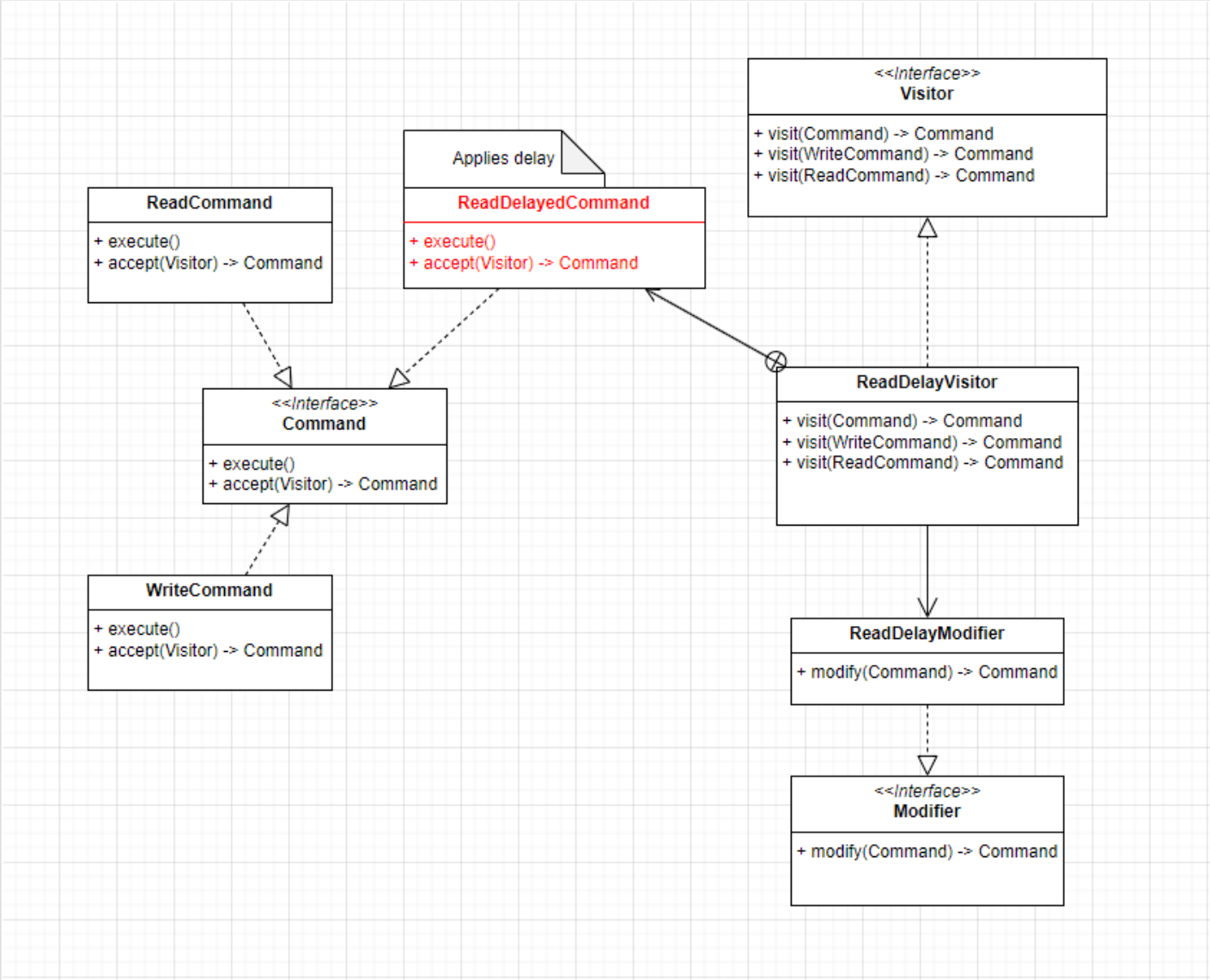
```
sptr<Command> ReadDelayVisitor::visit(sptr<Command> cmd) {
    return cmd;
}

sptr<Command> ReadDelayVisitor::visit(sptr<ReadCommand> cmd) {
    return make_sptr<DelayedReadCommand>(std::move(cmd));
}

sptr<Command> ReadDelayVisitor::visit(sptr<WriteCommand> cmd) {
    return cmd;
}
```

Паттерны в MeyerSAN: Visitor

ReadDelayVisitor.cpp

```
sptr<Command> ReadDelayVisitor::visit(sptr<Command> cmd) {
    return cmd;
}

sptr<Command> ReadDelayVisitor::visit(sptr<ReadCommand> cmd) {
    return make_sptr<DelayedReadCommand>(std::move(cmd));
}

sptr<Command> ReadDelayVisitor::visit(sptr<WriteCommand> cmd) {
    return cmd;
}
```

Паттерны в MeyerSAN: Visitor в картинках



Паттерны в MeyerSAN: Visitor

ReadDelayVisitor.hpp

```
class ReadDelayVisitor : public Visitor {
    class DelayedReadCommand : public Command {
        void execute() override {
            std::cout << “ReadDelay” << std::endl;
            original_->execute();
        }
    };

public:
    sptr<Command> visit(sptr<Command> cmd) { return cmd; }
    sptr<Command> visit(sptr<ReadCommand> cmd) {
        return make_sptr<DelayedReadCommand>(std::move(cmd));
    }
    sptr<Command> visit(sptr<WriteCommand> cmd) { return cmd; }
};
```

Паттерны в MeyerSAN: Visitor

ReadDelayVisitor.hpp

```
class ReadDelayVisitor : public Visitor {
    class DelayedReadCommand : public Command {
        void execute() override {
            std::cout << “ReadDelay” << std::endl;
            original_->execute();
        }
    };

public:
    sptr<Command> visit(sptr<Command> cmd) { return cmd; }
    sptr<Command> visit(sptr<ReadCommand> cmd) {
        return make_sptr<DelayedReadCommand>(std::move(cmd));
    }
    sptr<Command> visit(sptr<WriteCommand> cmd) { return cmd; }
};
```

Паттерны в MeyerSAN: Visitor

main.cpp

```
int main() {
    sptr<Modifier> mod = make_sptr<ReadDelayModifier>();

    auto cmds = std::vector<sptr<Command>>{
        make_sptr<ReadCommand>(),
        make_sptr<WriteCommand>()
    };

    for (auto cmd : cmds)
        mod->modify(cmd)->execute();

    return 0;
}

// Output: ReadDelay ReadCommand WriteCommand
```

Паттерны в MeyerSAN: Visitor и его проблемы

IOCommand.hpp

```
class ReadCommand : public Command {
public:
    void execute() override { std::cout << “ReadCommand” << std::endl; }
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(
                std::static_pointer_cast<ReadCommand>(shared_from_this())
        );
    }
};

class WriteCommand : public Command {
public:
    void execute() override { std::cout << “WriteCommand << std::endl; }
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(
                std::static_pointer_cast<WriteCommand>(shared_from_this())
        );
    }
};
```

Паттерны в MeyerSAN: Visitor и его проблемы

IOCommand.hpp but a little bit more real

```
class ReadCommand : public Command {
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(std::static_pointer_cast<ReadCommand>(shared_from_this()));
    }
};

class WriteCommand : public Command {
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(std::static_pointer_cast<WriteCommand>(shared_from_this()));
    }
};

class MgmtCommand : public Command {
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(std::static_pointer_cast<MgmtCommand>(shared_from_this()));
    }
};

...
```

Паттерны в MeyerSAN: Visitor и его проблемы

IOCommand.hpp but a little bit more real

```
class ReadCommand : public Command {
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(std::static_pointer_cast<ReadCommand>(shared_from_this()));
    }
};

class WriteCommand : public Command {
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(std::static_pointer_cast<WriteCommand>(shared_from_this()));
    }
};

class MgmtCommand : public Command {
    sptr<Command> accept(Visitor& visitor) override {
        return visitor.visit(std::static_pointer_cast<MgmtCommand>(shared_from_this()));
    }
};

...
```

Паттерны в MeyerSAN: Visitor

VisitorsMixin.hpp

```
template<typename CommandType, typename VisitorType = Visitor>
class VisitorMixin : public Command {
    sptr<Command> accept(VisitorType& visitor) override {
        return visitor.visit(std::static_pointer_cast<CommandType>(shared_from_this()));
    }
}; // Mixin!

class Command : public VisitorMixin<Command> {
       ...
};

class ReadCommand : public VisitorMixin<ReadCommand> {
       ...
};

class WriteCommand : public VisitorMixin<WriteCommand> {
       ...
};
```

Паттерны в MeyerSAN: Visitor

VisitorsMixin.hpp

```
template<typename CommandType, typename VisitorType = Visitor>
class VisitorMixin : public Command {
    sptr<Command> accept(VisitorType& visitor) override {
        return visitor.visit(std::static_pointer_cast<CommandType>(shared_from_this()));
    }
}; // Mixin!

class Command : public VisitorMixin<Command> {
       ...
};

class ReadCommand : public VisitorMixin<ReadCommand> {
       ...
};

class WriteCommand : public VisitorMixin<WriteCommand> {
       ...
};
```

Паттерны в MeyerSAN: Visitor

Или...

Паттерны в MeyerSAN: Visitor

C++23!

```
class proxy_shared_from_this : public std::enable_shared_from_this<proxy_shared_from_this>
{
public:
    template <typename Self>
    auto get_shared_from_this(this Self& self) {
        return std::static_pointer_cast<Self>(self.shared_from_this());
    }
};

class Command : public proxy_shared_from_this {
        ...
};
```

Паттерны в MeyerSAN: Visitor

01

Независимость логики команды и логики её модификации

02

Отсутствуют методы, “раздувающие” интерфейс

MeyerSAN: о дисках и СХД

Архитектура и паттерны

Ничто не бывает идеальным

Выводы

Ничто не бывает идеально

main.cpp

Ничто не бывает идеально

main.cpp

```
int main() {
    sptr<Modifier> mod = make_sptr<ReadDelayModifier>();

    auto cmds = std::vector<sptr<Command>>{
        make_sptr<ReadCommand>(),
        make_sptr<WriteCommand>()
    };

    for (auto cmd : cmds)
        mod->modify(cmd)->execute();

    return 0;
}
```

Ничто не бывает идеально

main.cpp

```
int main() {
    sptr<Modifier> mod = make_sptr<ReadDelayModifier>();

    auto cmds = std::vector<sptr<Command>>{
        make_sptr<ReadCommand>(),
        make_sptr<WriteCommand>()
    };

    for (auto cmd : cmds)
        mod->modify(cmd)->execute();

    return 0;
}
```

Ничто не бывает идеально

main.cpp

```
int main() {
    std::vector<sptr<Modifier>> mods = {
        make_sptr<ReadDelayModifier>(),
        make_sptr<ErrorInjectingModifier>(), ...
    };

    auto cmds = std::vector<sptr<Command>>{
        make_sptr<ReadCommand>(), ...
    };

    for (auto& cmd : cmds) {
        for (auto& mod : mods) { cmd = mod->modify(cmd); }
    }
}
```

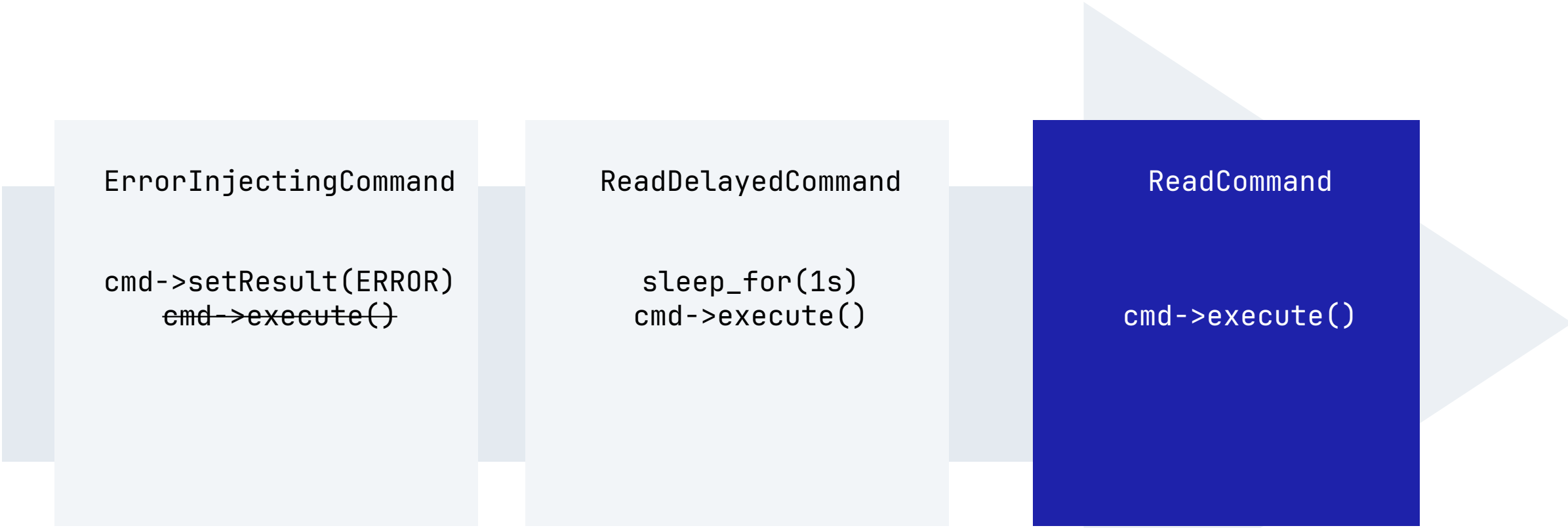
Ничто не бывает идеально

main.cpp

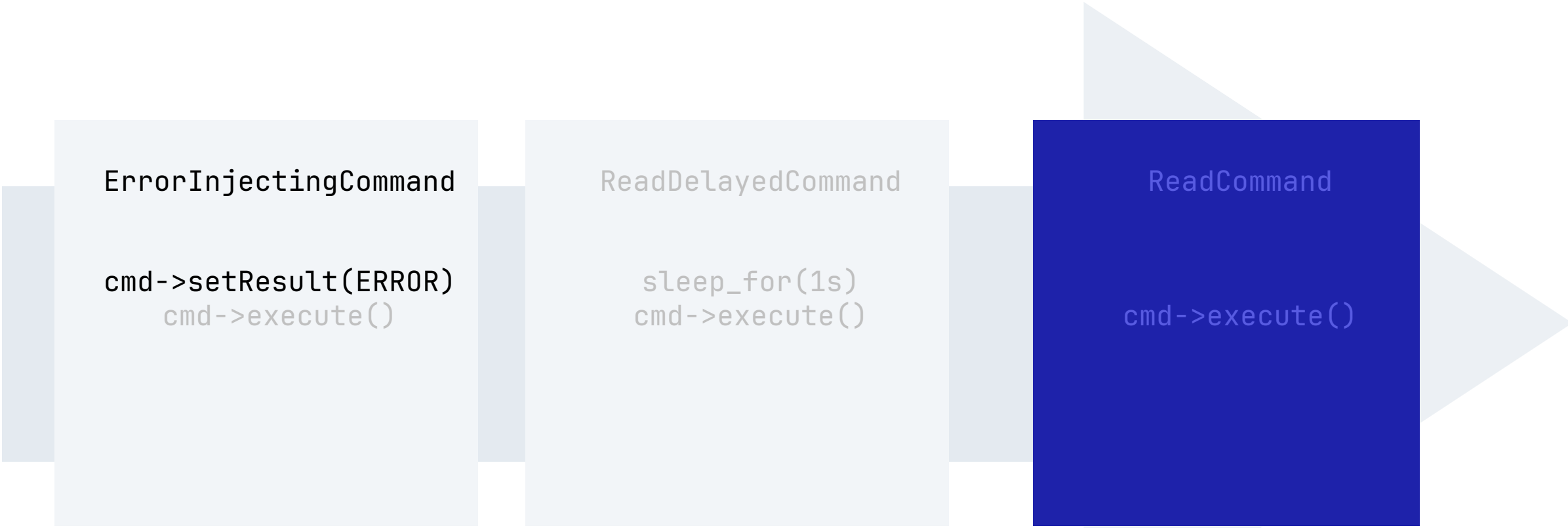
```
int main() {
    std::vector<sptr<Modifier>> mods = {
        make_sptr<ReadDelayModifier>(),
        make_sptr<ErrorInjectingModifier>(), ...
    };

    auto cmds = std::vector<sptr<Command>>{
        make_sptr<ReadCommand>(), ...
    };

    for (auto& cmd : cmds) {
        for (auto& mod : mods) { cmd = mod->modify(cmd); }
    }
}
```

Ничто не бывает идеально

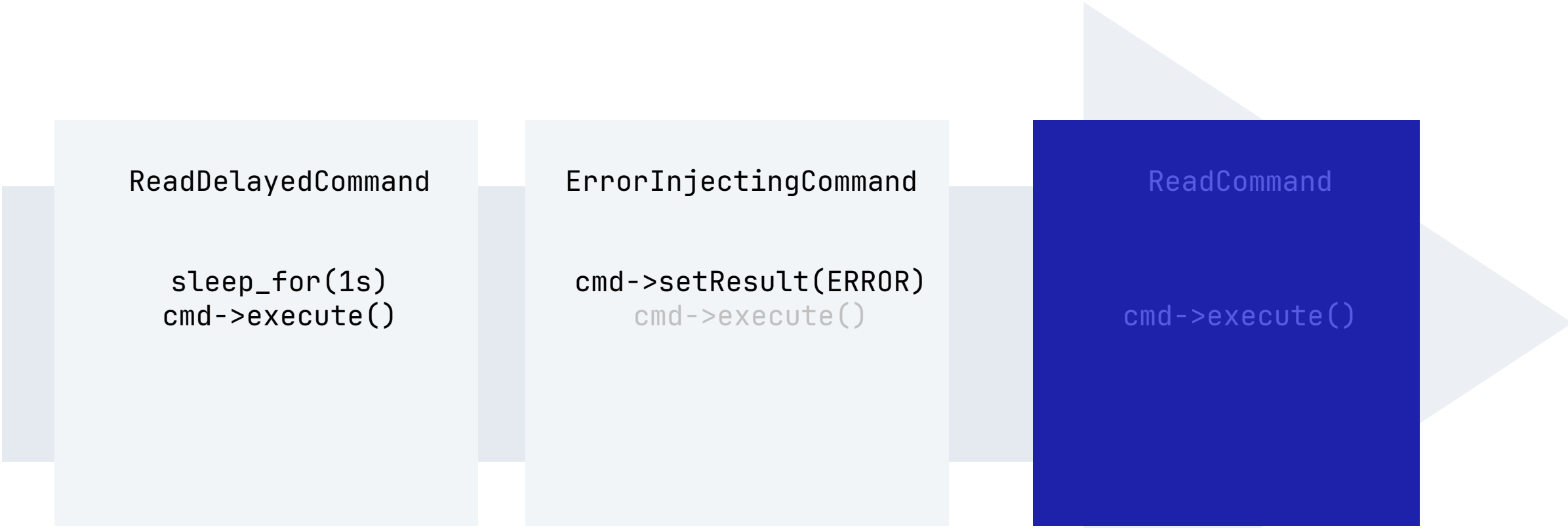
Ничто не бывает идеально



Ничто не бывает идеально



Ничто не бывает идеально: решение



Ничто не бывает идеально

main.cpp

```
int main() {
    std::vector<sptr<Modifier>> mods = {...};

    auto cmds = std::vector<sptr<Command>>{
        make_sptr<ReadCommand>(), ...
    };

    for (auto& cmd : cmds) {
        for (auto& mod) { cmd = mod->modify(cmd); }
    }

    for (auto cmd : cmds)
        cmd->execute();
}
```

Ничто не бывает идеально

main.cpp

```
int main() {
    // must be ordered with a certain criteria
    std::vector<sptr<Modifier>> mods = {...};

    auto cmds = std::vector<sptr<Command>>{
        make_sptr<ReadCommand>(), ...
    };

    for (auto& cmd : cmds) {
        for (auto& mod) { cmd = mod->modify(cmd); }
    }

    for (auto cmd : cmds)
        cmd->execute();
}
```

Ничто не бывает идеально

Ничто не бывает идеально

- Снижение производительности из-за обёрток и абстракций

Ничто не бывает идеально

- ~~Снижение производительности из-за обёрток и абстракций~~
- Трата времени на архитектурные сессии

Ничто не бывает идеально

- ~~Снижение производительности из-за обёрток и абстракций~~
- ~~Трата времени на архитектурные сессии~~
- Неожиданные юз-кейсы приводят к трудоёмким архитектурным патчам

Ничто не бывает идеально

- ~~Снижение производительности из-за обёрток и абстракций~~
- ~~Трата времени на архитектурные сессии~~
- ~~Неожиданные юз-кейсы приводят к трудоёмким архитектурным патчам~~

MeyerSAN: о дисках и СХД

Архитектура и паттерны

Ничто не бывает идеальным

Выводы

Выводы

Выводы

- ООП - это хорошо

Выводы

- ООП - это хорошо?

Выводы

- ООП - это хорошо, но для своих юз-кейсов

Выводы

- ООП - это хорошо, но для своих юз-кейсов
- Правильное использование паттернов повышает гибкость системы

Выводы

- ООП - это хорошо, но для своих юз-кейсов
- Правильное использование паттернов повышает гибкость системы
- При использовании абстракций уделите внимание вопросам производительности

Выводы

- ООП - это хорошо, но для своих юз-кейсов
- Правильное использование паттернов повышает гибкость системы
- При использовании абстракций уделите внимание вопросам производительности
- Закладывайте фундамент в точках потенциального расширения логики

Выводы

- ООП - это хорошо, но для своих юз-кейсов
- Правильное использование паттернов повышает гибкость системы
- При использовании абстракций уделите внимание вопросам производительности
- Закладывайте фундамент в точках потенциального расширения логики
- Не жалейте время на архитектурные сессии

Выводы

- ООП - это хорошо, но для своих юз-кейсов
- Правильное использование паттернов повышает гибкость системы
- При использовании абстракций уделите внимание вопросам производительности
- Закладывайте фундамент в точках потенциального расширения логики
- Не жалейте время на архитектурные сессии
- Нет универсального подхода или инструмента: выбирайте с умом!

БУДУЩЕЕ
В НАШИХ
РУКАХ