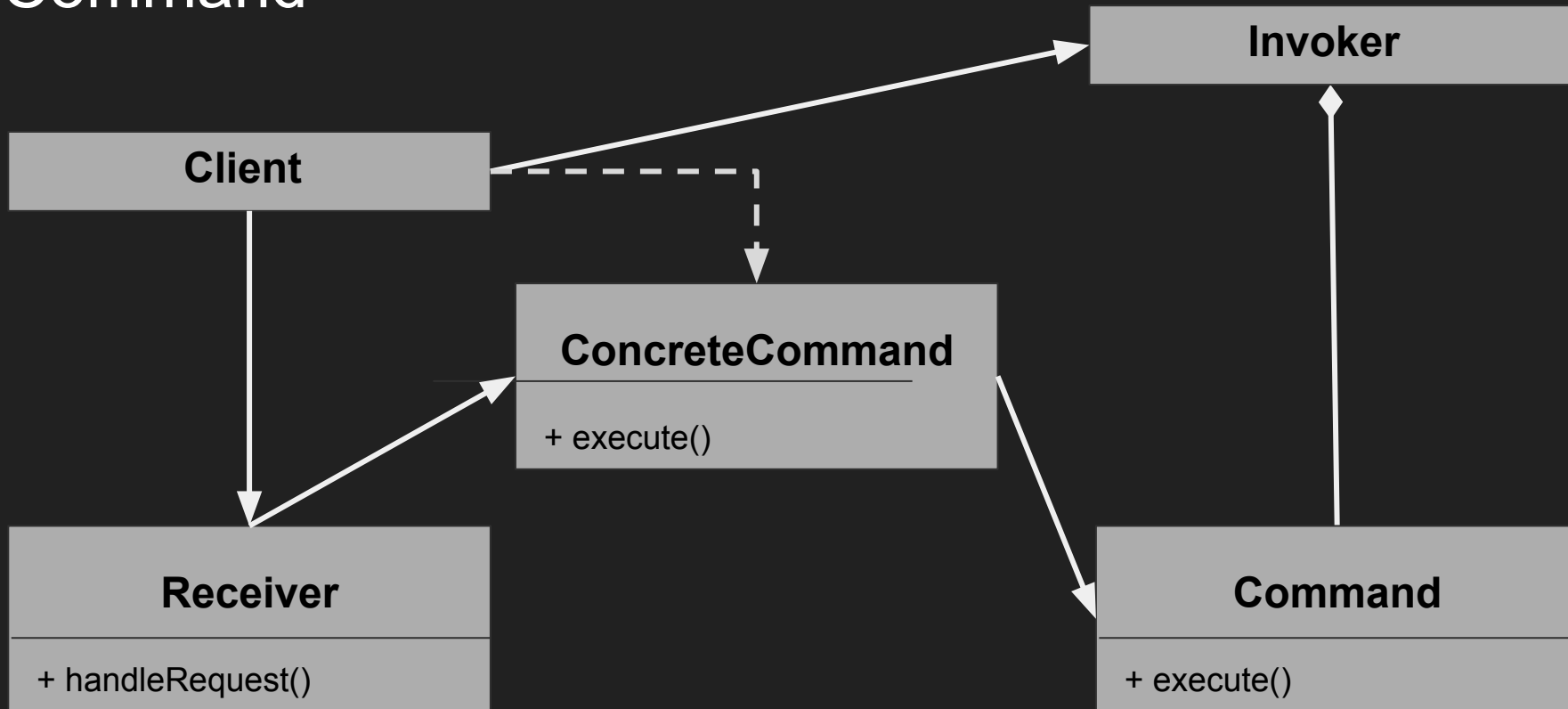


# Технологии программирования

Поведенческие паттерны. Command, Mediator,  
Interpreter, Iterator

# Command



# Command example

```
class CDocument {  
public:  
    CDocument() { ... }  
    void insert(size_t line, const std::string& str) { ... }  
    void remove(size_t line) { ... }
```

```
....  
};
```

```
class CCommand {  
protected:  
    Document * doc;  
  
public:  
    virtual ~Command() {}  
    virtual void do() = 0;  
    virtual void undo() = 0;  
    void set_document(CDocument* doc) { doc = _doc; }  
};
```

# Command example

```
class CCmdInsert : public CCommand {
public:
    InsertCommand(size_t line, const std::string& str): m_line(line), m_str(str) {}
    void do() { doc->insert(m_line, m_str); }
    void undo() { doc->remove(m_line); }

private:
    size_t m_line;
    std::string m_str;
};
```

# Command example

// receiver/client code

```
command = new CCmdInsert(line, "hello world");  
command->set_document(&document);  
command->do();  
done.push_back(command);
```

# Relations

Command & Prototype

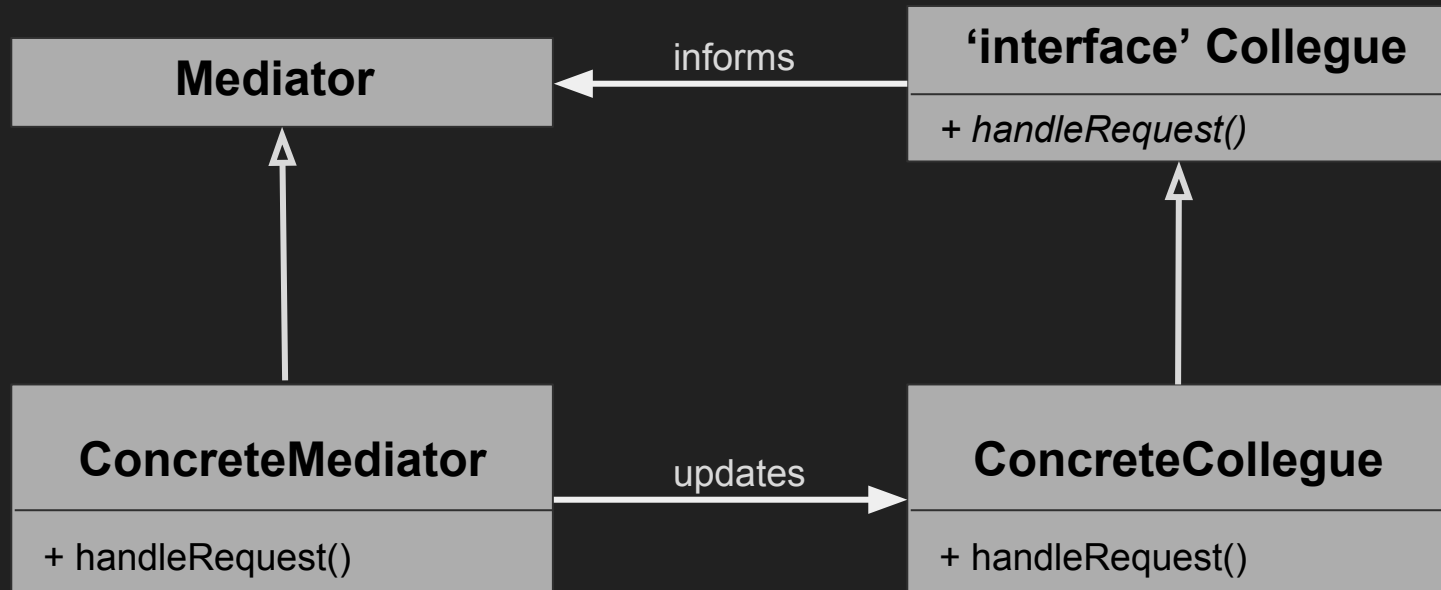
Command vs Strategy

# Когда использовать

необходимо  
параметризовать объекты  
выполняемым действием

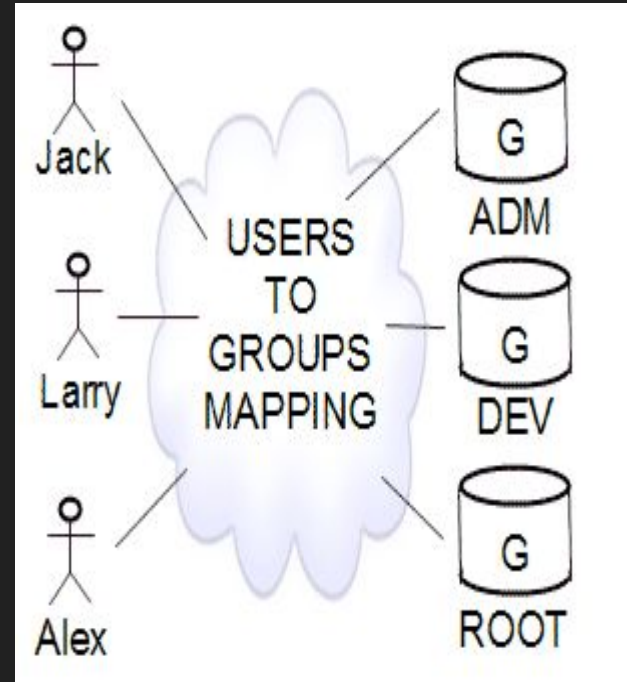
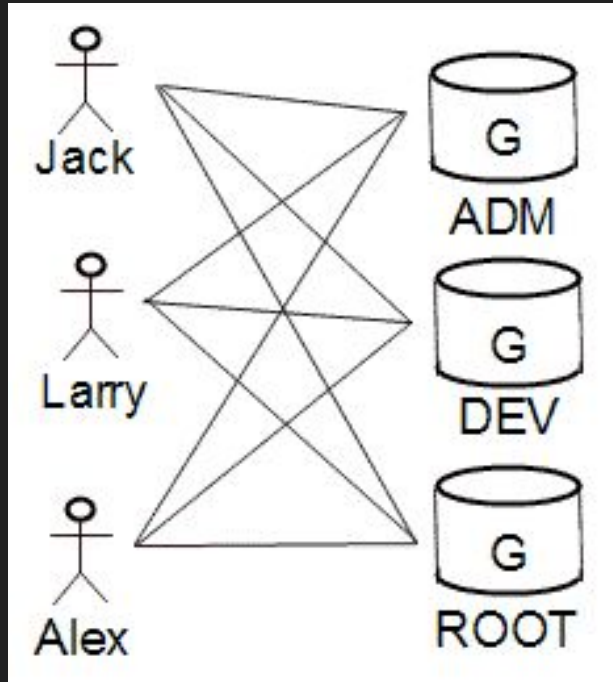
необходимо ставить  
операции в  
очередь/передавать их по  
сети/выполнять по  
расписанию

# Mediator

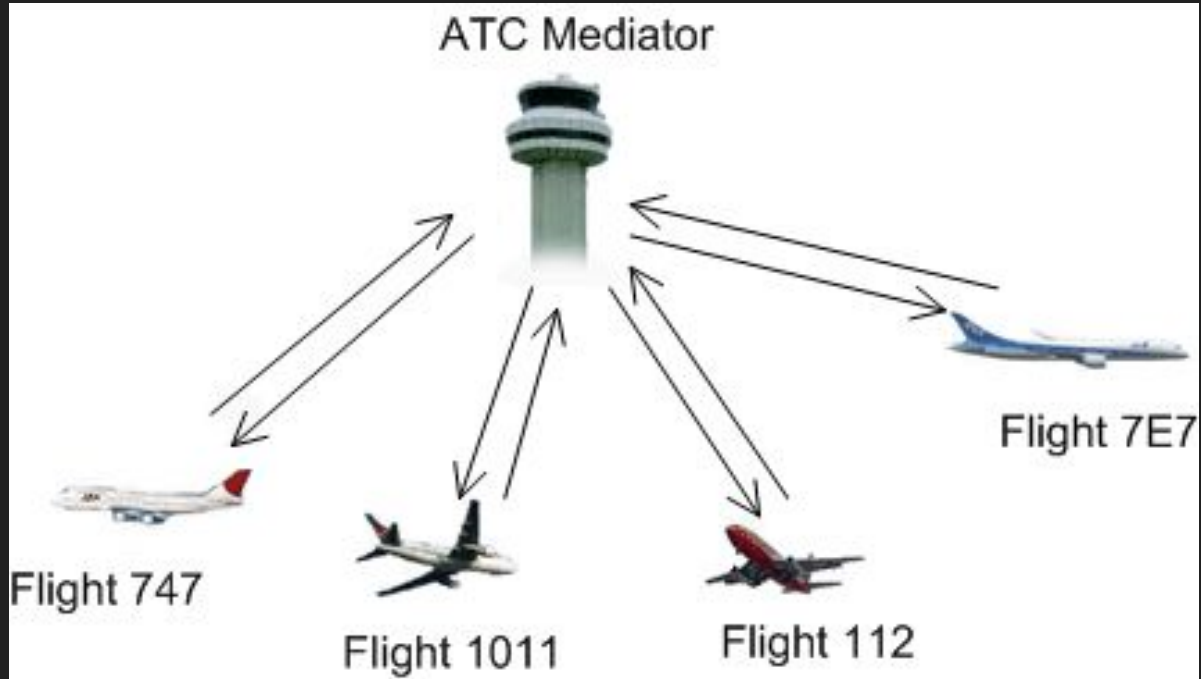




# Mediator example



# Mediator example



# Mediator example

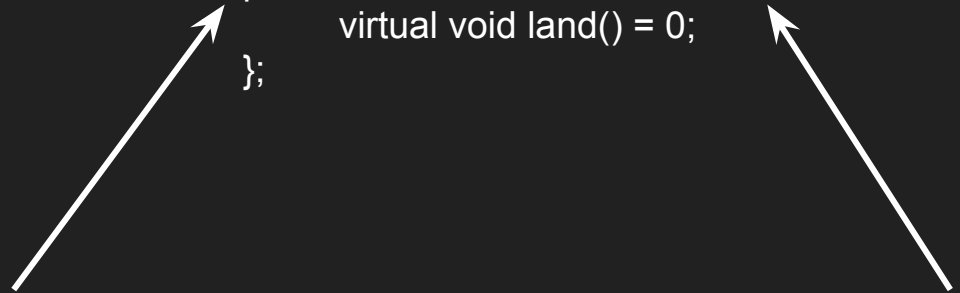
```
class IATC {    // mediator
public:
    virtual bool accept_landing() = 0;
};
```



```
class CATC: public IATC {    // concrete mediator
public:
    bool accept_landing() {
        // check ...
        return true;
    }
    ....
};
```

# Mediator example

```
class IAirplain { // colleague
public:
    virtual void land() = 0;
};
```



The diagram shows two white arrows pointing upwards from the concrete classes to the interface. One arrow originates from the CFlight747 class and points to the IAirplain class. The other arrow originates from the CFlight7E7 class and points to the IAirplain class.

```
class CFlight747 : public IAirplain { // concrete colleague
public:
    CFlight747(IATC* atc) : m_atc(atc) {}
    bool land() {
        if (!atc->accept_landing()) return false;
        // land
        return true;
    }
private:
    IATC* m_atc;
};
```

```
class CFlight7E7 : public IAirplain { // concrete colleague
public:
    CFlight7E7(IATC* atc) : m_atc(atc) {}
    bool land() {
        if (!atc->accept_landing()) return false;
        // land
        return true;
    }
private:
    IATC* m_atc;
};
```

# Mediator example

// client code

```
CFlight747 airplain(atc);  
while(true) {  
    airplain.land();  
    wait(timeout);  
}
```

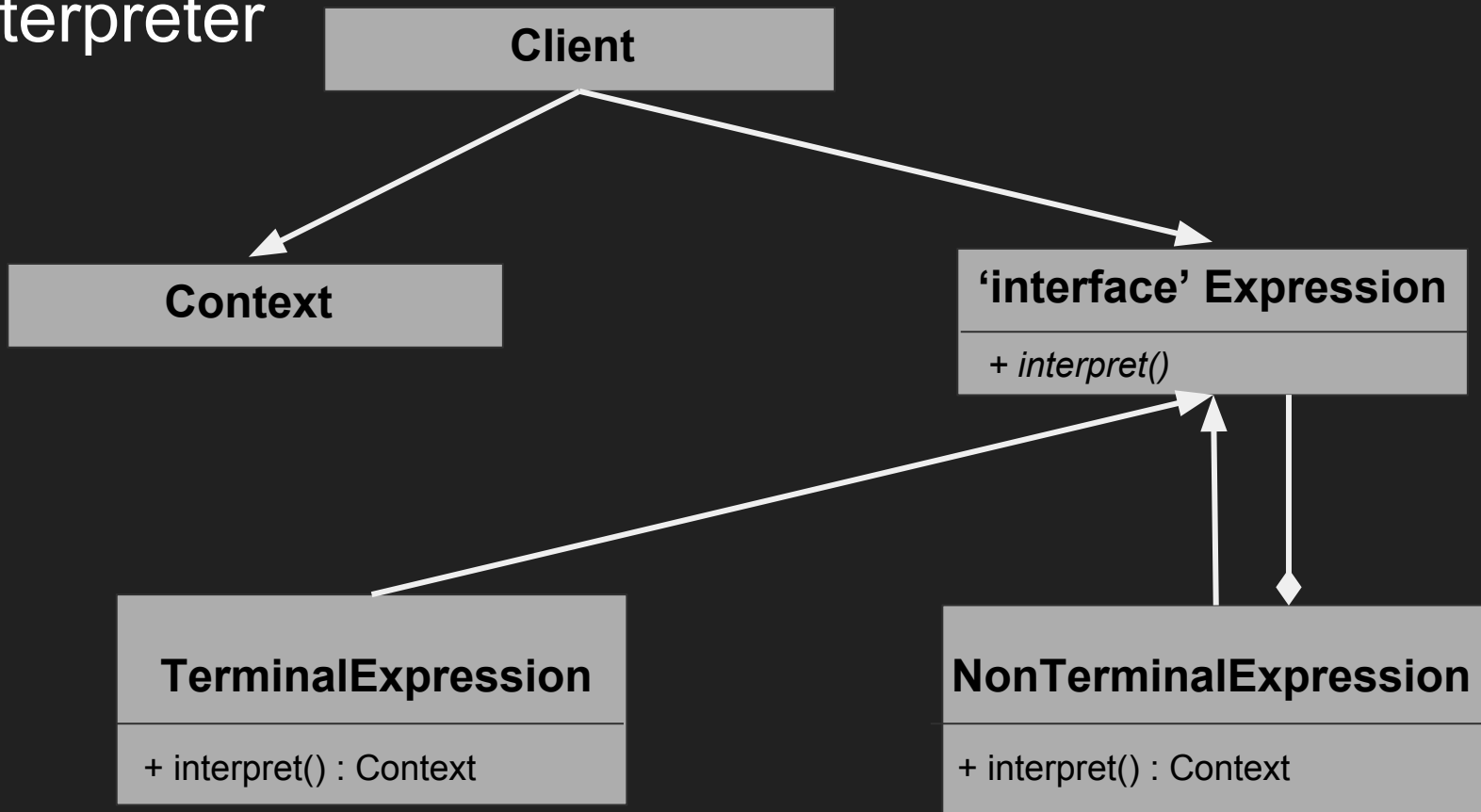
# Relations

Mediator vs Facade

# Когда использовать

сложно менять классы из-за  
множества связей

# Interpreter





# Interpreter use cases

валидация правил  
(regexp)

парсинг конфигов

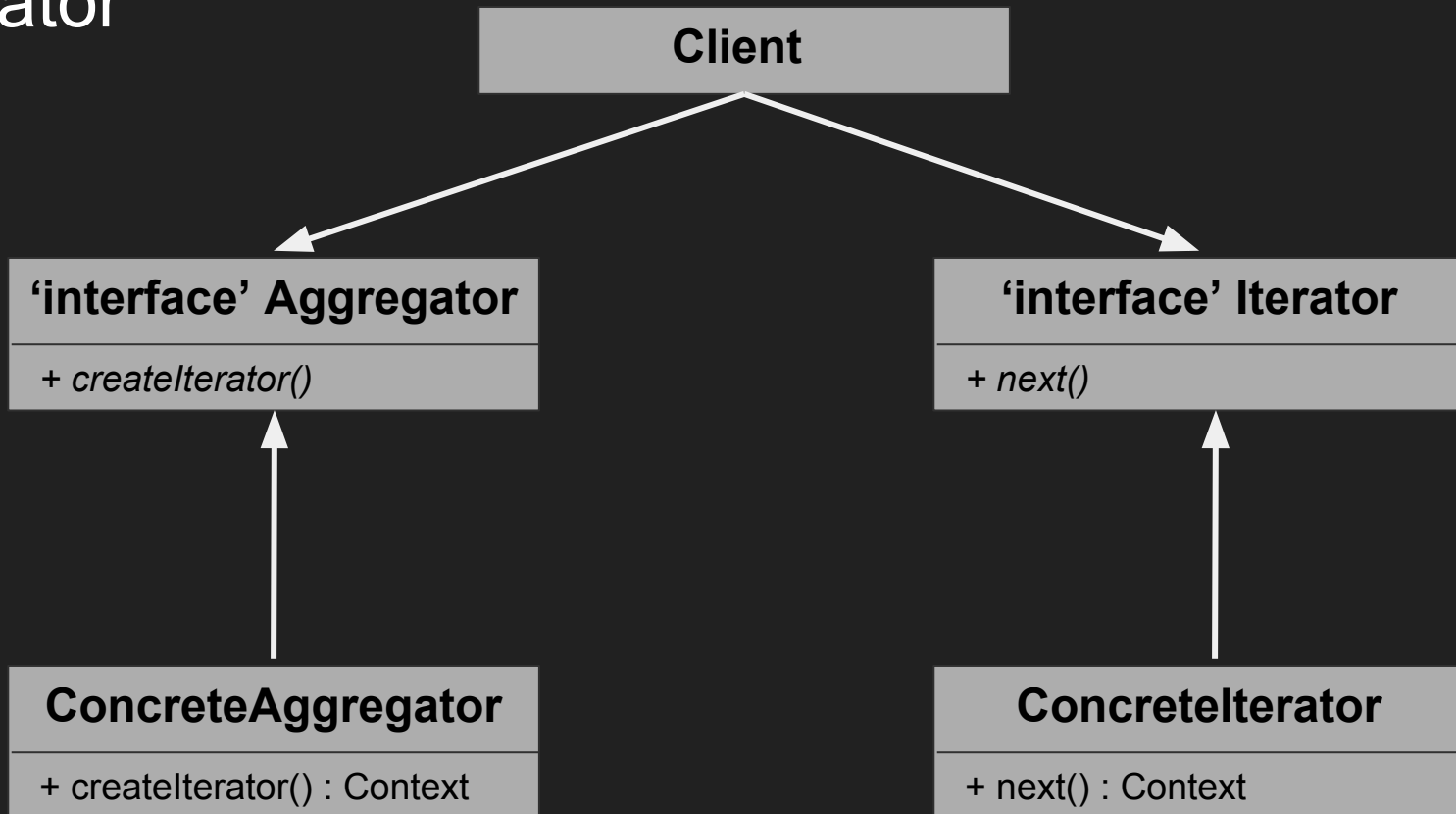
DSL

# Relations

Interpreter & Composite

Interpreter & Iterator

# Iterator



# Relations

Iterator & Composite

Iterator & Factory method

# Когда использовать

хочется скрыть от клиента  
сложность внутреннего  
устройства структуры  
данных

нужно иметь несколько  
вариантов обхода одной и  
той же структуры данных

хочется иметь единый  
интерфейс обхода  
различных структур данных