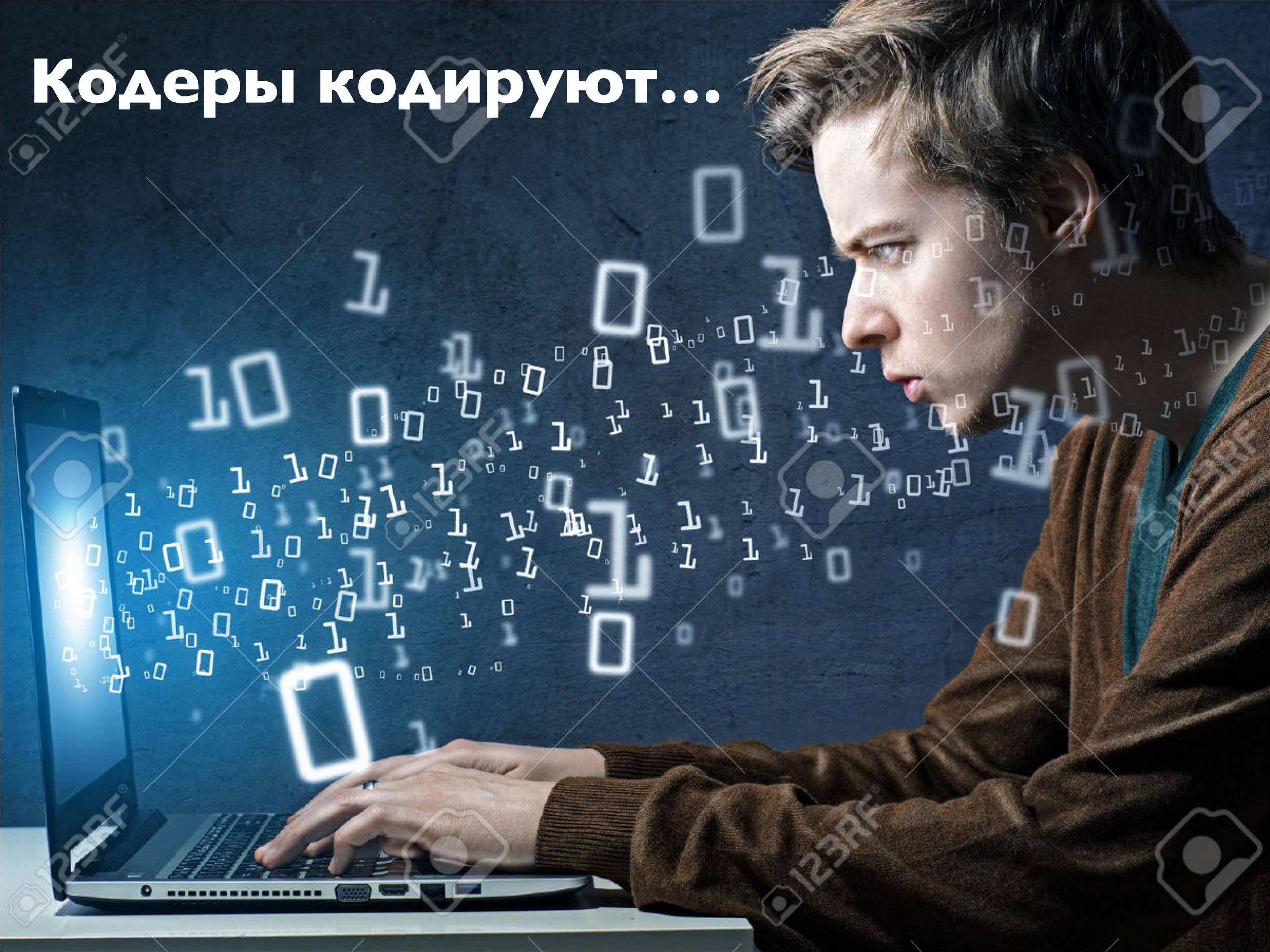
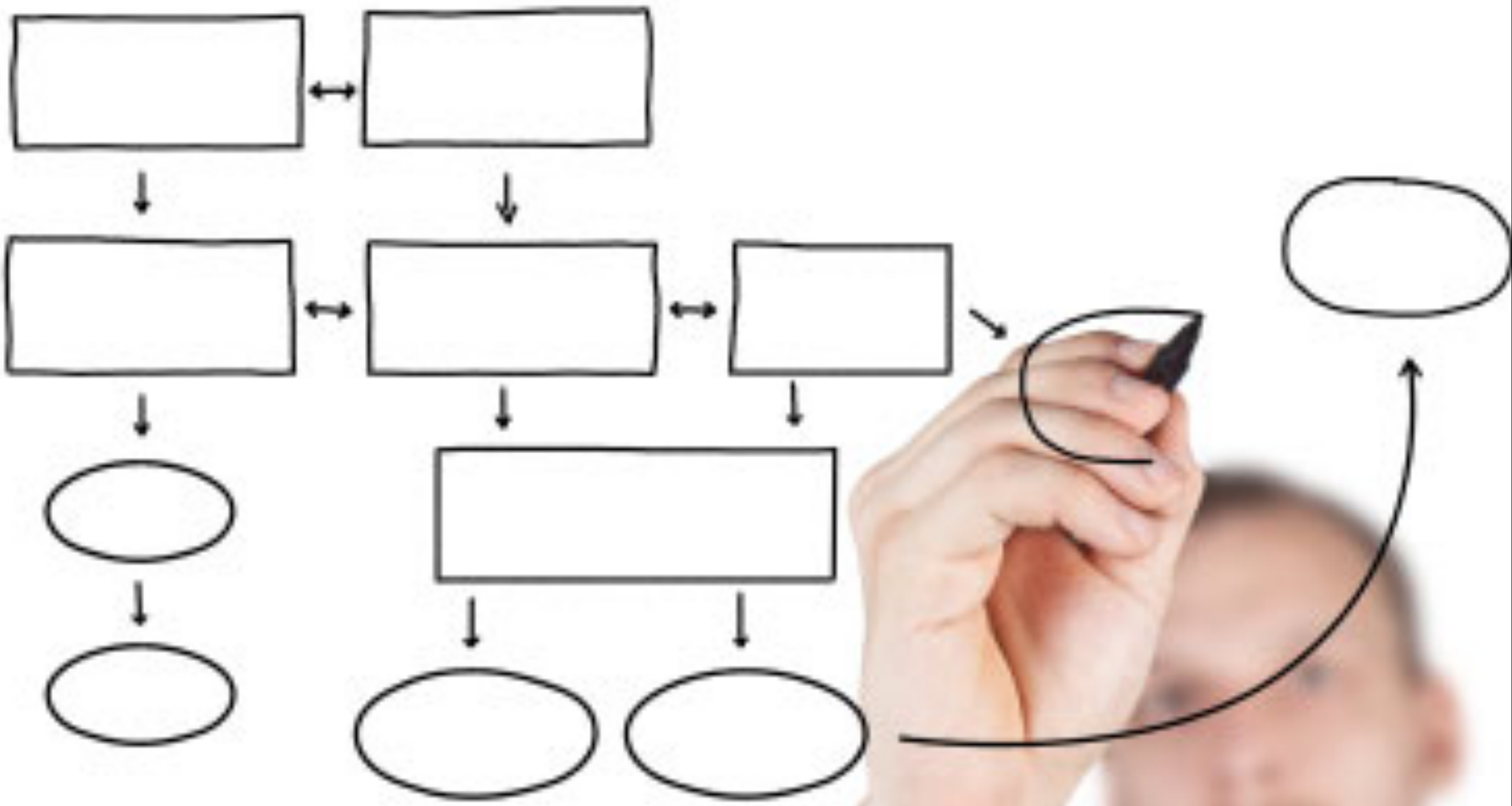
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лекция № | / **09**
08.04.2019 г.

Кодеры кодируют...





Проектировщики проектируют...

Разработчики разрабатывают...

- *Задача проектирования:* сформулировать, что именно кодировать и в рамках какой структуры.

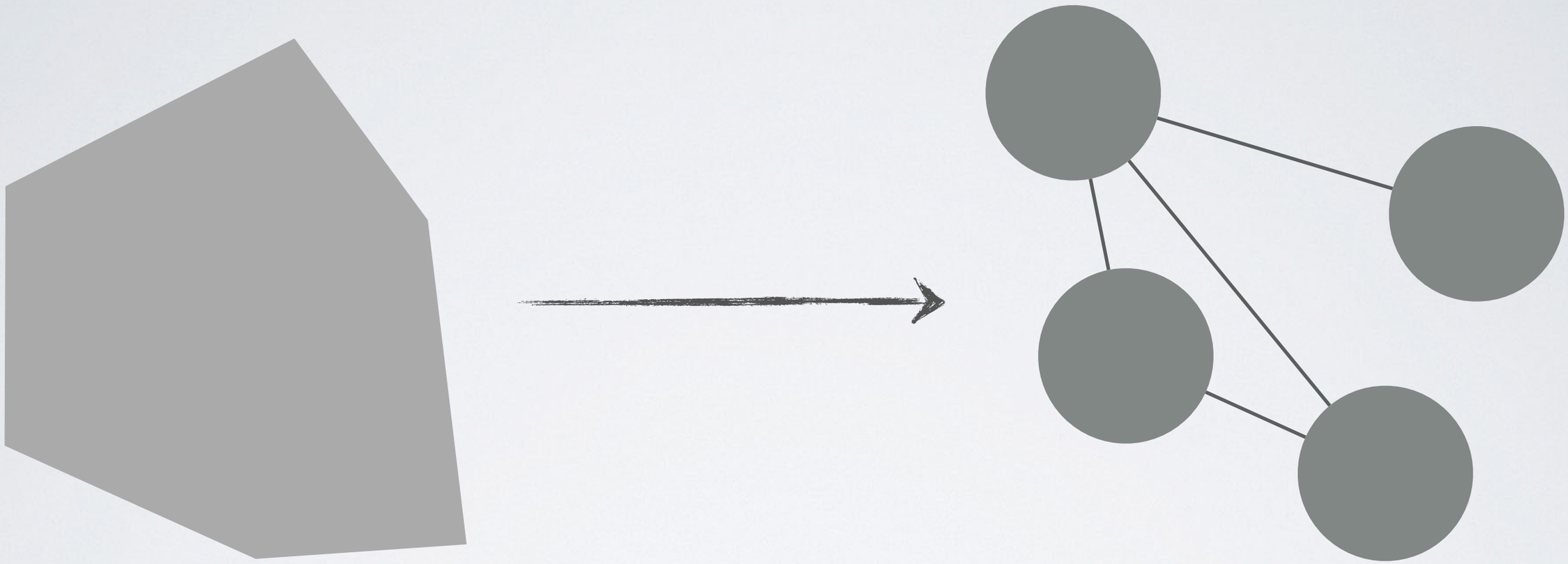
- Предполагается, что мы (или кто-то другой) дружим с языком, стандартной библиотекой и т.д. и сможем закодировать.

- Разработка = проектирование + кодирование.

ОБЩИЕ ТРЕБОВАНИЯ К КЛАССАМ

Как определить, что мы всё делаем правильно?

I. ДЕКОМПОЗИЦИЯ

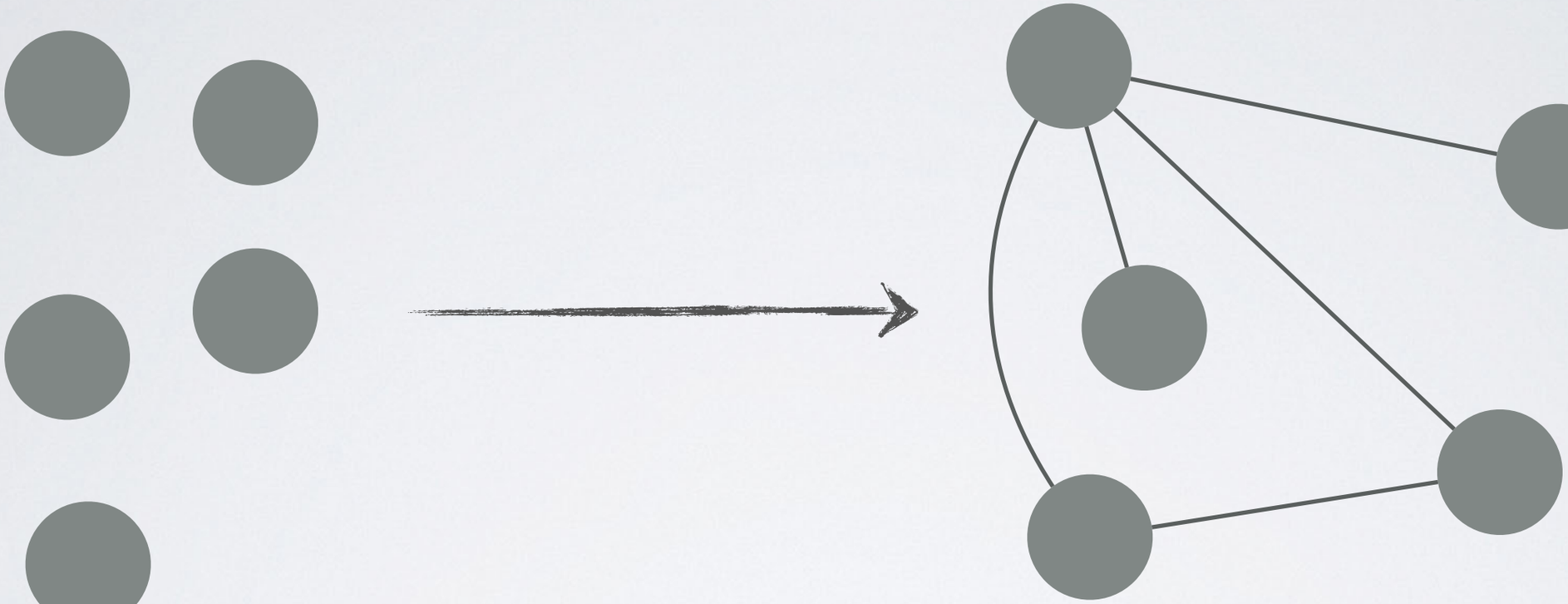


Задача

Подзадачи

Подзадачи являются самостоятельными!

II. МОДУЛЬНАЯ КОМПОЗИЦИЯ



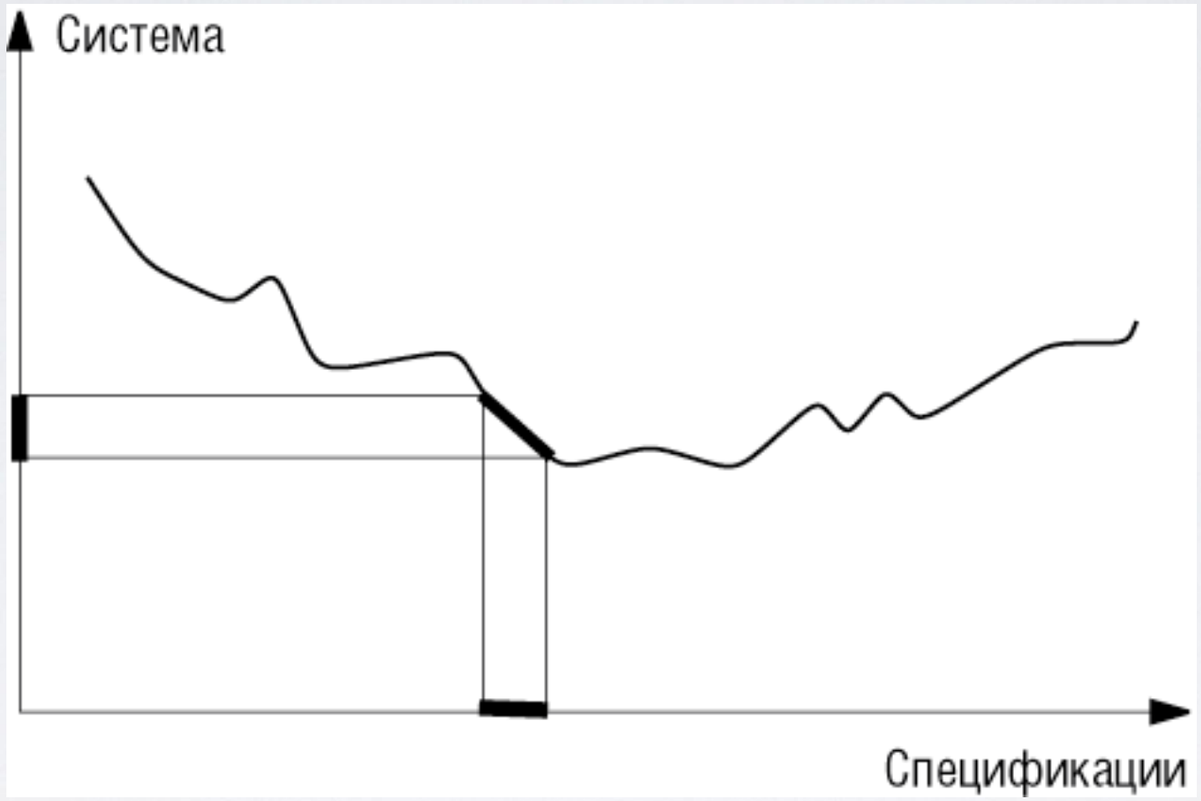
Модули свободно объединяются для создания новых систем

III. МОДУЛЬНАЯ ПОНЯТНОСТЬ

```
class Class1 {

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args) {

        // Logon
        BTULicenseManager licenseManager = new BTULicenseManager();

        // put your valid license here
        string  networkLicense = "00000000-0000-0000-0000-000000000000-00000000";
        string  password;

        password = "";
        licenseManager.Logon( networkLicense, password );

        Console.WriteLine( "Logged on." );

        string  fullName = @"C:\Documents and Settings\rkuo.SNAPSTREAM\My Documents\My Videos\South Park-(Freak Strike)-2004-08-17-0.mpg";

        BTULibrary library = new BTULibrary();

        // Get properties
        PVSPropertyBag bag = library.GetMediaByFullName( fullName );

        // Print properties to the console
        Console.WriteLine( "Properties of {0}", fullName );
        foreach( PVSProperty prop in bag.Properties ) {
            Console.WriteLine( "Property: {0}, {1}", prop.Name, prop.Value );
        }

        // Put the PVSPropertyBag into a more friendly collection class.
        // It's a good idea for you to write a friendlier wrapper class that
        // would allow you to add and remove properties and cast back to
        // the PVSPropertyBag type on the fly.
        ArrayList aProperties = new ArrayList( bag.Properties );

        // Change the "EpisodeDescription" property
        foreach( PVSProperty prop in aProperties ) {
            if( prop.Name == "EpisodeDescription" ) {
                prop.Value = "The boys compete to appear on a talk show. (Edited by Beyond TV Framework)";
            }
        }

        // Create a new PVSPropertyBag with the edited property
        PVSPropertyBag newBag = new PVSPropertyBag();
        newBag.Properties = (PVSProperty[])aProperties.ToArray( typeof(PVSProperty) );

        // This method will edit the recording
        library.EditMedia( fullName, newBag );

        // Print properties to the console and verify the change
        Console.WriteLine( "Edited properties of {0}", fullName );
        foreach( PVSProperty prop in bag.Properties ) {
            Console.WriteLine( "Property: {0}, {1}", prop.Name, prop.Value );
        }

        // Pause so you can see the output, hit enter to continue
        Console.WriteLine( "Press any key to exit..." );
        Console.ReadLine();
        return;
    }
}
```

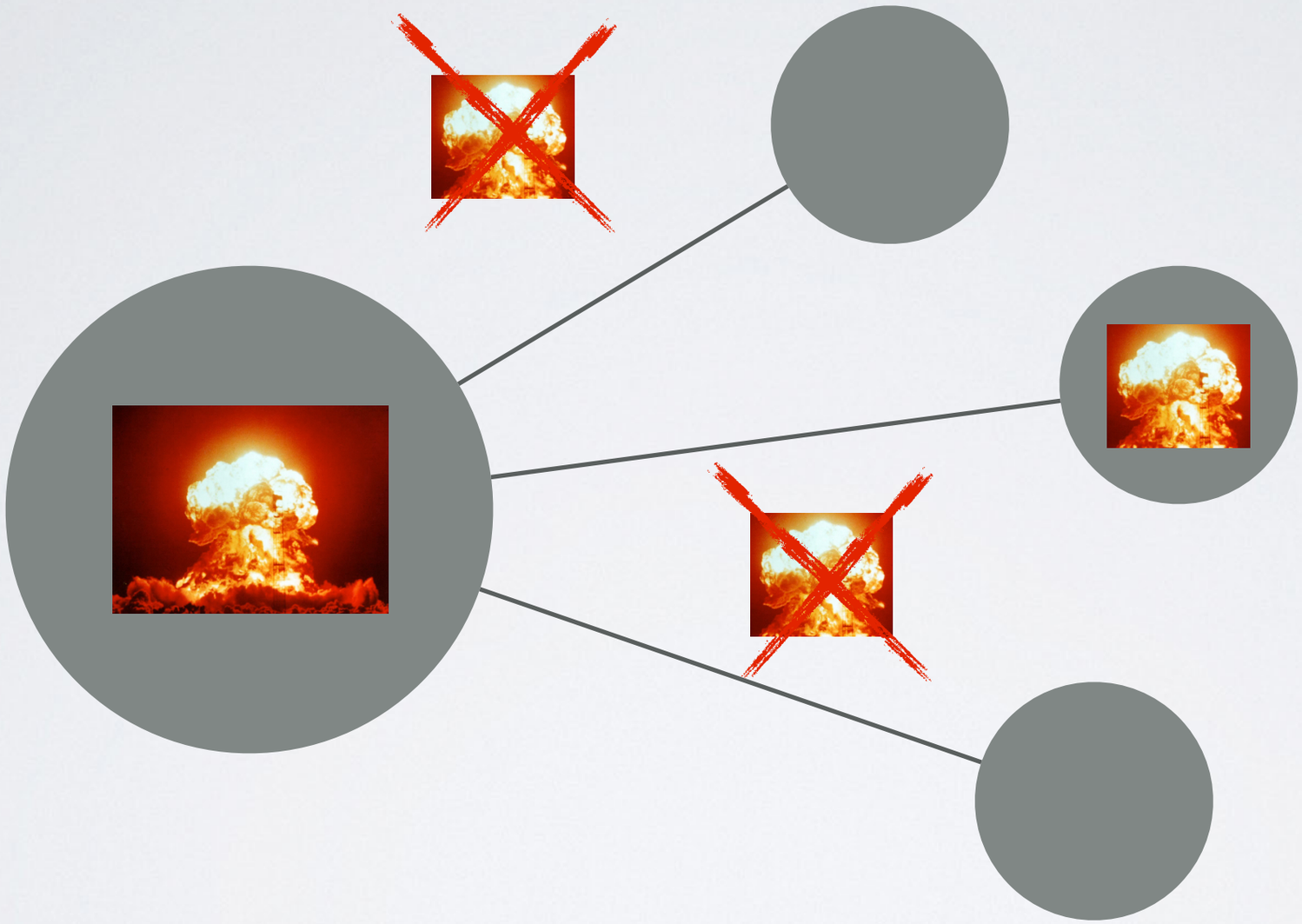
Можно понять содержание каждого модуля, не зная текста остальных

IV. МОДУЛЬНАЯ НЕПРЕРЫВНОСТЬ



Незначительное изменение спецификаций приведет к изменению небольшого числа модулей

V. МОДУЛЬНАЯ ЗАЩИЩЕННОСТЬ



Аварийная ситуация в одном из модулей ограничится только этим модулем или максимум несколькими соседними

ДЕМОНСТРАЦИОННАЯ ПРОГРАММА

Файл:

```
1 2 3 4 5
7 8 9 10 11
12 13 14 15 16
```

Вывод:

```
Median of medians = 9
```

ПОПЫТКА I

Берём и колбасим, не задумываясь

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int getMedian(vector<int> &numbers) {
    sort(numbers.begin(), numbers.end());

    return numbers[numbers.size() / 2];
}

// продолжение на следующем слайде…
```

```cpp
int main(int argc, char **argv) {
    if (argc != 2)
        return 1;

    ifstream ifs(argv[1]);
    if (!ifs)
        return 2;

    vector<int> medians;
    string line;

    while (getline(ifs, line)) {
        istringstream iss(line);
        vector<int> numbers;
        int n = 0;

        while (iss >> n)
            numbers.push_back(n);

        if (numbers.size() > 0)
            medians.push_back(getMedian(numbers));
    }

    if (medians.size() > 2)
        cout << "Median of medians = " << getMedian(medians) << endl;
    else
        cout << "Not enough lines" << endl;

    return 0;
}
```

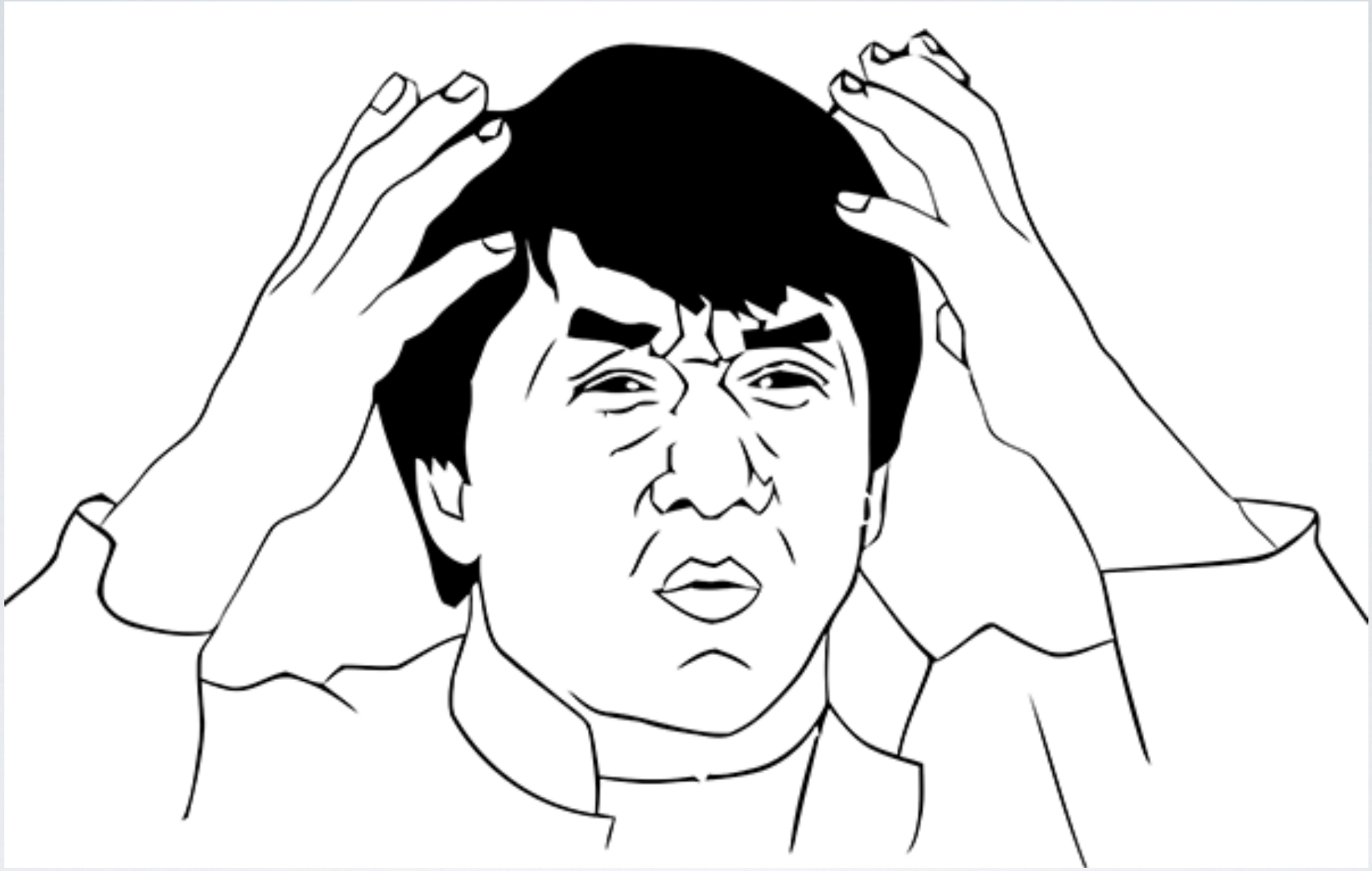
ПОПЫТКА 2

Чтобы программа стала объектно-ориентированной, наверное, надо добавить хотя бы один класс?

```
// те же #include
#include <exception>

using namespace std;

class Medianer {
public:
    Medianer(const char *filename)
            : ifs(filename) {
        if (!ifs)
            throw FileErrorException();
    }

    int getMedian();

    class Exception : public std::exception {};
    class FileErrorException : public Exception {};
    class NotEnoughException : public Exception {};
private:
    ifstream ifs;

    int calcMedian(vector<int> &numbers);
};

// продолжение на следующем слайде…
```

```
int Medianer::calcMedian(vector<int> &numbers) {
    sort(numbers.begin(), numbers.end());

    return numbers[numbers.size() / 2];
}

int Medianer::getMedian() {
    vector<int> medians;
    string line;

    while (getline(ifs, line)) {
        istringstream iss(line);
        vector<int> numbers;
        int n = 0;

        while (iss >> n)
            numbers.push_back(n);

        if (numbers.size() > 0)
            medians.push_back(calcMedian(numbers));
    }

    if (medians.size() > 2)
        return calcMedian(medians);
    else
        throw NotEnoughException();
}

// продолжение на следующем слайде…
```

```
int main(int argc, char **argv) {
    if (argc != 2)
        return 1;

    try {
        Medianer m(argv[1]);
        int med = m.getMedian();

        cout << "Median of medians = " << med << endl;
    }
    catch (Medianer::FileErrorException &) {
        return 2;
    }
    catch (Medianer::NotEnoughException &) {
        cout << "Not enough lines" << endl;
    }

    return 0;
}
```



ХОРОШАЯ ПРОГРАММА?

ПРИНЦИП ОДНОЙ ЗОНЫ ОТВЕТСТВЕННОСТИ

- Single responsibility principle (SRP).

- **Должна существовать только одна причина, которая может привести к изменению класса.**

Класс
Rectangle

Зона ответственности 1.
Вычисление площади

```
class Rectangle {
public:
    double area() const;    // вычислить площадь
    void draw();            // нарисовать прямоугольник

private:
    // ...
};
```

Зона ответственности 2.
Отрисовка прямоугольника

Класс Rectangle

Класс GuiRectangle

```
class Rectangle {
public:
    // вычислить площадь
    double area() const;

private:
    // ...
};
```

```
class GuiRectangle {
public:
    // нарисовать прямоугольник
    void draw();

private:
    Rectangle rect;
};
```

Разделяем на два класса и
используем *композицию*

ПОПЫТКА 3

Разнесём зоны ответственности…

```cpp
// те же #include
vector<vector<int>> readNumbers(istream &is) {
    string line;
    vector<vector<int>> result;

    while (getline(is, line)) {
        istringstream iss(line);
        vector<int> line_numbers;
        int n = 0;

        while (iss >> n)
            line_numbers.push_back(n);

        if (line_numbers.size() > 0)
            result.push_back(line_numbers);
    }

    return result;
}

int getMedian(vector<int> &numbers) {
    sort(numbers.begin(), numbers.end());

    return numbers[numbers.size() / 2];
}

// продолжение на следующем слайде…
```

```
int medianOfMedians(vector<vector<int>> &vec) {
    vector<int> medians;

    transform(vec.begin(), vec.end(), back_inserter(medians), getMedian);

    return getMedian(medians);
}

int main(int argc, char **argv) {
    if (argc != 2)
        return 1;

    ifstream ifs(argv[1]);
    if (!ifs)
        return 2;

    auto numbers = readNumbers(ifs);

    if (numbers.size() > 2)
        cout << "Median of medians = " << medianOfMedians(numbers) << endl;
    else
        cout << "Not enough lines" << endl;

    return 0;
}
```

ФУНКЦИОНАЛЬНЫЙ СТИЛЬ

- Зоны ответственности полностью разнесены.
- Состояние полностью инкапсулировано внутри функции.
- Объекты не пригодились. В более сложных случаях функцию можно заменить объектом-функтором, инкапсулирующим состояние в процессе вычислений.
- **Проблема:** разделение через структуру данных не всегда эффективно.

ПОПЫТКА 4

Таки разделим на классы…

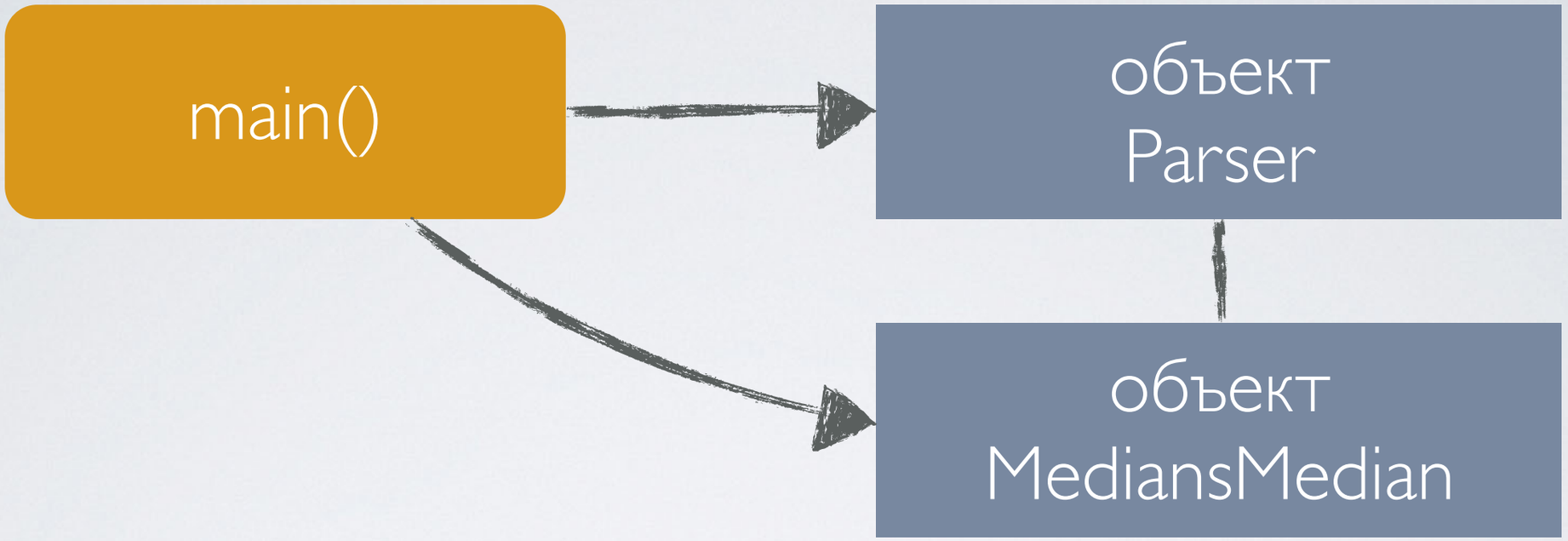
```
// те же #include
class Parser {
public:
    Parser(istream &_is) : is(_is) {}

    vector<int> getNext() {
        string line;

        while (getline(is, line)) {
            istringstream iss(line);
            vector<int> line_numbers;
            int n = 0;

            while (iss >> n)
                line_numbers.push_back(n);

            if (line_numbers.size() > 0)
                return line_numbers;
        }

        return vector<int>();
    }
private:
    istream &is;
};

// продолжение на следующем слайде…
```

```
int getMedian(vector<int> &numbers) {
    sort(numbers.begin(), numbers.end());
    return numbers[numbers.size() / 2];
}

class MediansMedian {
public:
    MediansMedian(Parser &p) : parser(p) {}

    int result() {
        vector<int> medians, line_numbers;

        while (!(line_numbers = parser.getNext()).empty())
            medians.push_back(getMedian(line_numbers));

        if (medians.size() > 2)
            return getMedian(medians);
        else
            throw NotEnoughException();
    }

    class NotEnoughException : public std::exception {};

private:
    Parser &parser;
};
// продолжение на следующем слайде…
```

```
int main(int argc, char **argv) {
    if (argc != 2)
        return 1;

    ifstream ifs(argv[1]);
    if (!ifs)
        return 2;

    try {
        Parser p(ifs);
        int result = MediansMedian(p).result();

        cout << "Median of medians = " << result << endl;
    }
    catch (MediansMedian::NotEnoughException &) {
        cout << "Not enough lines" << endl;
    }

    return 0;
}
```

```
// Альтернативный интерфейс для Parser
class Parser {
public:
    Parser(istream &_is) : is(_is) {}

    // было: vector<int> getNext();
    bool getNext(vector<int> &result) {
        string line;
        while (getline(is, line)) {
            result.clear();

            // ....
        }

        return false;
    }

private:
    istream &is;
};
```

ОО СТИЛЬ (НАКОНЕЦ-ТО!)



- Разбор файла и вычисления разнесены по разным классам.
- Класс **MediansMedian** работает с классом **Parser**, предоставляющим интерфейс в виде метода **getNext()**.
- При необходимости мы можем выделить интерфейс в базовый класс и избавиться от зависимости от конкретного класса **Parser**.

```
class BaseParser {
public:
    virtual vector<int> getNext() = 0;
};

class Parser : public BaseParser {
public:
    // ...
};

class MediansMedian {
public:
    MediansMedian(BaseParser &p) : parser(p) {}
    // ...
};
```

Базовый класс парсера

ПОПЫТКА 5

Кто сказал, что разделение на классы единственное?!

```
// те же #include
class ParserClient {
public:
    virtual void start() {}
    virtual void processRow(vector<int> &row) {}
    virtual void finish() {}
};

class Parser {
    istream &is;
public:
    Parser(istream &_is) : is(_is) {}

    void parse(ParserClient &client) {
        client.start();
        string line;

        while (getline(is, line)) {
            istringstream iss(line);
            vector<int> line_numbers;
            int n = 0;

            while (iss >> n)
                line_numbers.push_back(n);

            if (line_numbers.size() > 0)
                client.processRow(line_numbers);
        }
        client.finish();
    }
};
// продолжение на следующем слайде…
```

```
int getMedian(vector<int> &numbers) {
    sort(numbers.begin(), numbers.end());
    return numbers[numbers.size() / 2];
}

////////////////////////////////////////////////////////////

class MediansMedian : public ParserClient {
public:
    virtual void processRow(vector<int> &row) {
        medians.push_back(getMedian(row));
    }

    bool isEnough() const { return medians.size() > 2; }

    int result() { return getMedian(medians); }

private:
    vector<int> medians;
};

// продолжение на следующем слайде…
```

```
int main(int argc, char **argv) {
    if (argc != 2)
        return 1;

    ifstream ifs(argv[1]);
    if (!ifs)
        return 2;

    MediansMedian medmed;
    Parser p(ifs);
    p.parse(medmed);

    if (medmed.isEnough())
        cout << "Median of medians = " << medmed.result() << endl;
    else
        cout << "Not enough lines" << endl;

    return 0;
}
```

ВЫВОДЫ

- Банальная реализация задачи тесно связывает ввод данных и вычисления. Нужно разделять.
- Один подход: сначала всё ввести, сохранить, потом вычислять.
- Другой подход: разнести зоны ответственности по различным классам (**Parser**, **MediansMedian**). В такой паре один класс будет «активным», другой — «пассивным».
- Для «пассивного» класса полезно выделить базовый класс с интерфейсом и зависеть от него, это позволит использовать разные классы.
- Используя шаблоны классов, можно выделить «алгоритм» из **MediansMedian** и получить возможность легко создать **AveragesAverage** и даже **AveragesMedian!**

КОНЕЦ ДЕВЯТОЙ ЛЕКЦИИ

```
Lecture().end();
```