

Окопник Григорий

HardCode

- язык программирования общего назначения
- облачная платформа разработки и администрирования сервисов
- next gen IDE: система доставки IT контента

Фатальные ошибки времени исполнения

В современном мире любят 3 класса ошибок:

- Null pointer exception
- Index out of range
- Unhandled exception

Фатальные ошибки времени исполнения

Null pointer exception: пример на Go

```
type abc struct {
    field int
}

func getAbc() *abc {
    return nil
}

func main() {
    v := getAbc()           // Crash
    print(v.field)
    if v != nil {           // No crash, wow...
        print(v.field)
    } else {                // Crash
        print(v.field)
    }
    print(v.field)          // Crash
    if v != nil || 3*3 == 9 { // Extra condition
        print(v.field)      // Crash
    }
    if v == nil {            // Typo
        print(v.field)      // Crash
    }
}
```

Фатальные ошибки времени исполнения

Null pointer exception: почему?

Почему во всех языках присутствует эта “фича”??



Фатальные ошибки времени исполнения

Null pointer exception: пример как надо

```
struct abc {
    int field;
}

optional struct abc getAbc () { // optional abc == variant<struct abc|null>
    return null
}

void main () {
    optional struct abc v = getAbc ()
    print (v.field)          // Compile-time error
    let (struct abc& v_ref = v) {
        print (v_ref.field)   // No crash
    } else {
        print (v_ref.field)   // Compile-time error
    }
    print (v.field)          // Compile-time error
    if (let (abc& v_ref = v) || 3*3 == 9) { // Warning: "v_ref" can't be exposed
        print (v_ref.field)   // Compile-time error
    }
    if (let (abc& v_ref = v) && 3*3 == 9) {
        print (v_ref.field)   // No crash
    } else {
        print (v_ref.field)   // Compile-time error
    }
}
```

Фатальные ошибки времени исполнения

Index out of range: пример на Go

```
func getArray() []int {
    return []int{}
}

func main() {
    a := getArray()
    print(a[0])           // Crash
    if len(a) >= 1 {
        print(a[0])       // No crash, wow...
    }
    print(a[0])           // Crash
    if len(a) >= 1 || 3*3 == 9 { // Extra condition
        print(a[0])       // Crash
    }
    if len(a) >= 0 {      // Typo
        print(a[0])       // Crash
    }
}
```

Фатальные ошибки времени исполнения

Работа с индексами: пример как надо

```
int[] getArray () {
    return int[] ()
}

void main () {
    int[] a = getArray ()
    print (a[0])           // Compile-time error (Can only be accessed by index object!)
    let (index (a) i = 0) {
        print (a[i])       // No crash
    }
    print (a[0])           // Compile-time error
    if (let (index (a) i = 0) || 3*3 == 9) { // Warning: `i` is not exposed (should use `_`)
        print (a[i])       // Compile-time error
    }
    if (let (index (a) i = 0) && 3*3 == 9) {
        print (a[i])       // No crash
    } else {
        print (a[i])       // Compile-time error
    }
    mut int[] b = int[] (1, 2, 3) // Everything is const by default, we need keyword "mut" here
    let (index (b) i = 2) {
        b[i] = 500          // Can assign too
        print (b[i])
    }
}
```

Фатальные ошибки времени исполнения

Работа с индексами: проверки нужны редко

```
void sort2 (int& a, int& b)
{
    if (a > b)
        swap (a, b)
}

void sort (int[] elements) // No run-time index check
{
    for desc (word:ranged i: #elements - 1, 1) // If using "ranged" with no arguments
        for (word:ranged j: Ø, i - 1)           // range is statically deduced
            sort2 (elements[j], elements[j + 1])
}

void sort (int[] elements) // Same thing but verbose for clarity
{
    for desc (word:ranged (1, #elements - 1) i: #elements - 1, 1)
        for (word:ranged (Ø, #elements - 2) j: Ø, i - 1) {
            word:ranged (Ø, #elements - 2) index_a = j
            word:ranged (1, #elements - 1) index_b = j + 1
            sort2 (elements[index_a], elements[index_b])
        }
}
```

Фатальные ошибки времени исполнения

Exception vs ...

- Исключения явно отделяют обработку ошибочного сценария от обработки успешного
- Не нужно обрабатывать на каждом уровне
- Не получится использовать значение в случае ошибки
- Unhandled exception:
исключения легко не обработать на нужном уровне
- Неясны точки выхода из функции: кидает ли код исключение?

Фатальные ошибки времени исполнения

... vs Error object

- Нет Unhandled exception
- Явные точки выхода из функции: return нагляден, нет исключений
- Ветки обработки ошибки неотличимы от веток успешных сценариев
- Легко забыв сделать проверку значения Error object обработать некорректное значение
- Нужно явно обрабатывать на каждом слое

Фатальные ошибки времени исполнения

Что же выбрать: исключения или объекты ошибок?

Перефразируем мудрость древней касты бродячих торговцев.

Иногда все предложенные варианты плохи –
нужно придумать своё решение.



Фатальные ошибки времени исполнения

Гибридная обработка ошибок

Встраиваем паттерн “значение или ошибка”

```
// Some syntactic sugar

variant<Type|error@> val = get_value()
must (Type exact_val = val) {
    ...
}
    ::= let (Type exact_val = val) {
        ...
    } else return val.error

Type exact_val = must (val)
    ::= Type exact_val = val.value else return val.error

maybe TypeX    ::= variant<TypeX|error@>

optional TypeX ::= variant<TypeX|null>
```

Фатальные ошибки времени исполнения

Гибридная обработка ошибок: избыточность Go

```
func mix(a, b, alpha float64) (float64, error) {
    if alpha < 0.0 || alpha > 1.0 {
        return 0.0, errors.New("argument 'alpha' not in range[0.0..1.0]")
    }
    return (a*alpha + b*(1.0-alpha)), nil
}

type rgb struct {
    r, g, b float64
}

func mix_rgb(c1, c2 rgb, alpha float64) (rgb, error) {
    r, err := mix(c1.r, c2.r, alpha)
    if err != nil {
        return rgb{}, err
    }
    g, err := mix(c1.g, c2.g, alpha)
    if err != nil {
        return rgb{}, err
    }
    b, err := mix(c1.b, c2.b, alpha)
    if err != nil {
        return rgb{}, err
    }
    return rgb{
        r,
        g,
        b,
    }, nil
}

func main() {
    mixed, err := mix_rgb(rgb{0.1, 0.3, 0.25}, rgb{0.7, 0.2, 0.4}, 0.8)
    if err == nil {
        fmt.Printf("Color (%v, %v, %v)\n", mixed.r, mixed.g, mixed.b)
    } else {
        fmt.Printf("Failed to mix colors: %v\n", err.Error())
    }
}
```

Фатальные ошибки времени исполнения

Гибридная обработка ошибок: выразительность HardCode

```
maybe double mix (double& a, double& b, double& alpha) {
    if (alpha < 0.0 || alpha > 1.0) {
        return @@error ("argument 'alpha' not in range[0.0..1.0]")
    }
    return (a*alpha + b*(1.0 - alpha))
}

struct rgb {
    double r, g, b;
}

maybe struct rgb mix_rgb (struct rgb& c1, struct rgb& c2, double& alpha) {
    return (
        r = must mix (c1.r, c2.r, alpha), // Argument of "must" must be variant<typeof (lvalue)|error>
        g = must mix (c1.g, c2.g, alpha), // if argument of "must" contains error "return error" is called
        b = must mix (c1.b, c2.b, alpha), // else alternative value is returned
    )
}

void main () {
    match (mix_rgb ((r = 0.1, g = 0.3, b = 0.25), (r = 0.7, g = 0.2, b = 0.4), 0.8)) {
        case (struct rgb& mixed) {
            print ("Color (%v, %v, %v)\n", mixed.(r, g, b))
        }
        case (@error err) {
            print ("Failed to mix colors: %v\n", err.message)
        }
    } // No "missing default case" here
}
```

Фатальные ошибки времени исполнения

Самодокументация возвращаемых значений

Как у гугла

```
func some_function() (value *Type)
// Нужно ли проверять на nil?

func some_other_function() (value *Type, err error)
// Какие значения может вернуть функция: все 4 варианта?
// Нужно ли проверять value на nil если err == nil?
// Является ли value валидным (частичный результат) если err != nil или это ошибка реализации?
```

Фатальные ошибки времени исполнения

Самодокументация возвращаемых значений Как у людей

```
// Может быть null:  
optional Type* some_function()  
// Не может быть null:  
Type* some_function()  
  
// Ошибка или значение:  
maybe Type some_function()  
// Ошибка, значение или null:  
variant<Type|error@|null> Type some_function()  
// Возможно ошибка + всегда значение:  
optional error@ some_function(out Type value)  
// Возможно ошибка + возможно значение:  
optional error@ some_function(out optional Type value)
```

Фатальные ошибки времени исполнения

Гибридная обработка ошибок

Объект error

```
struct error {
    string domain; // не нужно заводить enum'ы
    string code; // не нужно заводить enum'ы
    ubyte[] message;
    polymorphic userdata;
    optional error@ child;
}
```

- обходится без enum'ов но хранит domain и code
- может хранить произвольные пользовательские данные

Тип `string`:

- задаётся строго литералами
- интернируется (как следствие, дедуплицируется)
- создаётся при линковке функции (не в момент выполнения)
- удаляется подсчётом ссылок
- операция сравнения тождественна сравнению указателей
- не является бутылочным горлышком в мультипоточном программировании

Фатальные ошибки времени исполнения

Работа со стэком ошибок

```
maybe struct rgb mix_rgb (struct rgb& c1, struct rgb& c2, double& alpha) {
    double r = @must ("Failed to interpolate red component") mix (c1.r, c2.r, alpha)
    double g = @must ("mix_rgb", "interpolation error", "Failed to interpolate green component") mix (c1.g, c2.g, alpha)
    match (mix (c1.b, c2.b, alpha)) {
        case (double& b) {
            return (r = r, g = g, b = b)
        }
        case (error@ err) {
            return @@error ("mix_rgb", "interpolation error", "Failed to interpolate blue component", err)
        }
    }
}
```

Фатальные ошибки времени исполнения

Гибридная обработка ошибок

Итог

- Синтаксис явно отделяют обработку ошибочного сценария от обработки успешного
- Нужно лаконично проксировать на каждом уровне:
позволяет отслеживать пути распространения ошибок
- Явные точки выхода из функции: `return` и `must` наглядны, нет исключений
- Не получится использовать (невалидное) значение в случае ошибки
- Нет `Unhandled exception`
- Явно определяются варианты возвращаемых значений
- Стэк ошибок и домены позволяют лучше и проще детализировать ошибку

Средства синхронизации

Бессменные проблемы синхронизации

- Race condition:
непонятно, что когда произойдёт
- Deadlock:
намертво зависшая программа
- Livelock:
что-то происходит, но программа висит
- Сложно отловить ошибки тестами

Средства синхронизации

Deadlock в C++

```
int main() {
    mutex m1, m2;
    thread t1([&m1, &m2] {
        cout << "Thread 1: 1. Acquiring m1." << endl;
        m1.lock();
        this_thread::sleep_for(chrono::milliseconds(10));
        cout << "Thread 1: 2. Acquiring m2." << endl;
        m2.lock();
        cout << "Thread 1: 3. Not reached" << endl;
    });
    thread t2([&m1, &m2] {
        cout << "Thread 2: 1. Acquiring m2." << endl;
        m2.lock();
        this_thread::sleep_for(chrono::milliseconds(10));
        cout << "Thread 2: 2. Acquiring m1." << endl;
        m1.lock();
        cout << "Thread 2: 3. Not reached" << endl;
    });

    t1.join();
    t2.join();

    return 0;
}
```

Средства синхронизации

Зачем нужны проблемы синхронизации?



Средства синхронизации

Как избавиться от проблем синхронизации?

Следуем следующим правилам:

- Не даём разработчику mutex'ы
- Вместо прямой работы с mutex'ами помечаем объекты требующие общего доступа ключевым словом `shared`
- Shared объекты не вкладываются
- Один поток может запросить доступ не больше чем к одному shared объекту одновременно

Средства синхронизации

Пример использования shared объектов

```
optional<error> func (context, // Control context
                        shared mut massive_container& big_object1,
                        shared mut massive_container& big_object2)
{
    big_object1.a.do_something () // Compilation error: shared object not locked
    must lock (big_object1, context) {
        big_object1.a.do_something ()
        big_object1.b.do_something2 ()
        lock (big_object2, context) { // Compilation error: nested lock
            big_object2.a.do_something ()
        }
    } /* else return error ("Cancelled by context") */
    let lock (big_object2, context) {
        big_object2.a.do_something ()
    } else {
        return error@ ("Object access timed out")
    }
    big_object1.a.do_something () // Compilation error: shared object not locked
    return null
}
```

Средства синхронизации

Что на горизонте

- Компилятор гарантирует единоличное использование `shared` объекта потоком
- Ошибки синхронизации не допускаются компилятором
- Встроенный анализ позволяет гарантировать завершение функции:
`Lifelock` не возможен
- А значит и `Deadlock` невозможен
- Поддерживается `RW lock` версия `shared` объекта
- Логическое зависание возможно из-за сетевого взаимодействия.
Это тоже можно анализировать.
Для этого нужен сетевой протокол со строгой событийной моделью.
- Конкурентные структуры и безопасные вложенные замки не входят в MVP

Модель памяти

Старый тусклый мир

- Как правило, RAII и GC не комбинируются
- В случае когда RAII и GC комбинируются, RAII нужно явно форсировать
- GC периодически зависит
- Нет гарантий иммутабельности объектов
- Случайные копирования объектов
- Zombie-объекты
- Ненужная инициализация и неинициализированные объекты

Модель памяти

Храбрый новый мир

- RAII даже для объектов попадающих в GC
- Нет необходимости в zombie-объектах
- GC без зависимостей: уже есть справочная реализация неблокирующего GC.
Самая длинная пауза: 1 atomic на батч событий в GC.
- Строгие гарантии иммутабельности объектов (фундамент sandboxing'a)
- Инициализация и изменение – отдельные операции.
- Строгая проверка инициализации объекта перед использованием.
Инициализация в любом месте перед использованием строго 1 раз.
Включая ветвления и инициализацию через передачу ссылок.
- Максимальная дружелюбность к автоматическим SIMD оптимизациям.
- Вероятность случайного копирования объектов крайне мала: синтаксис.
- Copy-конструкторы не нужны.
- Все функции stateless: изоляция модулей, динамическая компиляция, тестируемость.
- Анализируемая глубина стэка.

Модель памяти

Garbage Collector курильщика

- Каждое элементарное действие является точкой синхронизации потоков
- Существуют $O(N)$ паузы
(несмотря на эвристические оптимизации)

Модель памяти

Garbage Collector здорового человека

- Точкой синхронизации является отправка батча событий в обработчик графа:
 - » после удаления связи
 - » перед отправкой объекта (обрабатываемого GC) в другой поток
- Паузой является одна атомарная операция
- Обработка графа осуществляется за счёт сбора событий в очередь вместо моментного применения

Модель памяти



Модель памяти

Freezing GC, no RAll, Zombie-objects, ...

```
function undefined_file_close_time()
    local fh = assert(io.open("data1.txt", "w"))
    fh:write("hello\n")
    -- Lua GC: 'fh' is closed in unpredictable time
end

function zombie_object()
    local fh, err = io.open("data2.txt", "w")
    if fh then
        fh:write("hello\n")
        fh:close()
        -- 'fh' is zombie-object here
    else
        print(err)
    end
end
```

Модель памяти

Хватит это терпеть!

```
void basic_raii ()  
{  
    variant<mut io::file@|error@> maybe_fh = file_system.open ("data1.txt", io.READ_ONLY)  
    match (maybe_fh) {  
        case (mut io::file& fh) {  
            let (ubyte[]& data = fh.read_all ()) {  
                stdout.write (data)  
                stdout.write ("\n")  
            }  
        }  
        case (error@ err) {  
            stderr.println (err.description ())  
        }  
    }  
    // file is closed as in basic RAII  
}
```

Модель памяти

Хватит это терпеть!

```
class gl_utils.mesh {
    ubyte[] url;
    ubyte[] local_path;
    .geometry geometry; /* .geometry <=> gl_utils.mesh.geometry */
    mut variant<mut io.file@|error@|null>* fh; // global reference: handled by GC
}

optional error@ load_mesh (mut struct mesh& mesh)
{
    variant<ubyte[]|error@> data // Not initialized here
    match (mesh.fh) { // Initializing 'data' in each branch
        case (mut io.file& fh) {
            data = fh.read_all ()
            // `mesh.fh := null` would be compile-time error
        }
        case (error@ err) {
            data = err
        }
        case (null) {
            data = error@ ("Resource already loaded")
        }
    }
    mesh.fh := null
    // No zombie object here
    return mesh.parse_mesh (must data)
}
```

Модель памяти

Параметры функции

- 3 типа параметров:
 - иммутабельные
 - неинициализированные
 - мутабельные
- Все аргументы передаются только по ссылкам
- Локальные ссылки не покидают скоуп
- Функции имеют доступ только к аргументам и объекту (если функция – метод)
- Вложенная инициализация оптимальна
- Доступ к возвращаемому значению всегда оптимален (нет неоднозначности RVO)

Модель памяти

Параметры функции

```
void check_eq (int& value, out bool& equal)
{
    equal = value == max
}

bool acc_max (int& next_value, out bool& equal, mut int& max)
{
    if (next_value > max) {
        max := next_value
        equal = false
        return true // Sugar for { retval = true; return }
    } else {
        check_eq (next_value, equal)
        return false // Sugar for { retval = false; return }
    }
}
```

Модель памяти

Параметры функции

```
variant<double|double[2]|null|enum .> solve_square_equation (double& a, double& b, double& c)
{
    if (math.abs (a) < math.epsilon ()) {
        if (math.abs (b) < math.epsilon ()) {
            if (math.abs (b) < math.epsilon ()) {
                retval = enum .any
            } else {
                retval = null
            }
        } else {
            retval = c/b
        }
        return
    }

    double d = b*b - 4.0*a*c
    if (d >= 0.0) {
        double sqrt_d = math.sqrt (d)
        return double[2] ((b - sqrt_d)/(-2.0*a), (b + sqrt_d)/(-2.0*a)) // Sugar for { retval = VALUE; return }
    }
    retval = enum .complex
}
```

Шаблоны

Шаблоны в C++, Java, C#



Шаблоны

Шаблоны в HardCode

- Шаблонные аргументы строго типизированы
- Статический полиморфизм – динамически полиморфный объект с подстановкой
- Или простыми словами:
 - » Полиморфный объект определяется единожды
 - » И может использоваться как динамически, так и статически

ООП в мире

"I made up the term object-oriented, and I can tell you I did not have C++ in mind."

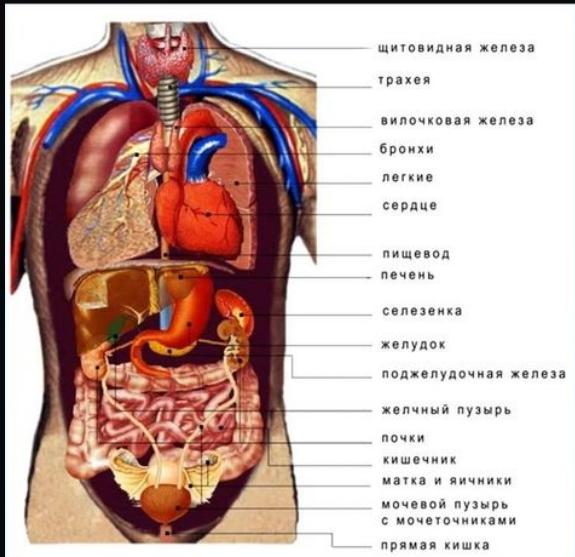
-- Alan Kay, OOPSLA '97

ООП в HardCode

- Ориентированность на динамический полиморфизм в GameDev: Entity модель
- Composition over Inheritance
- Класс всегда реализует заданный интерфейс, причём, строго один
- Классы не наследуются

Composition over Inheritance

Композиция



лучше

Наследования

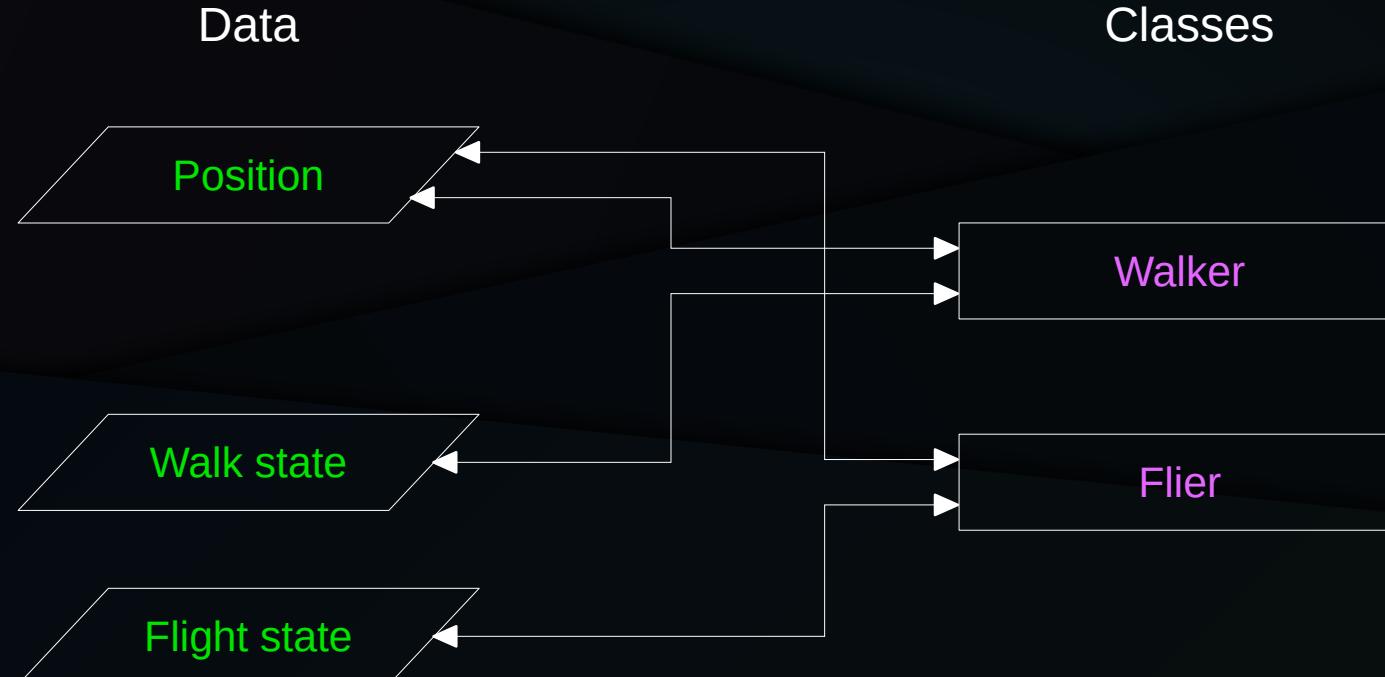


Пространство

Время

Динамический полиморфизм

Данные и функции: многие ко многим



Динамический полиморфизм

Данные и функции: многие ко многим

```
// Basic data sections
class position {
    vec3 translation
    quat orientation
}
class walk_state {
    bool forward, back, left, right
    double phase
}
class flight_state {
    bool forward, back
    double flap_period
    double phase
}

// Proxy data sections
class walker: implements walker {
    mut class position* position
    mut class walk_state* walk_state
}
class flyer: implements flyer {
    mut class position* position
    mut class flight_state* flight_state
}

// Interfaces
interface walker {
    void start_move (enum .direction& direction)
    void stop_move (enum .direction& direction)
    void update (double& delta_time)
}
interface flyer {
    void start_move (enum .direction& direction)
    void stop_move (enum .direction& direction)
    void set_flap_period (double& flap_period)
    void update (double& delta_time)
}

{
    entity cute_thing = world.new_entity ()
    cute_thing.extend (walker ()) // +position, +walk_state
    cute_thing.extend (flyer ()) // +flight_state
}
```

Хранение кода

Объектная модель в C++

Директория (вкладываются)



Файл (не вкладываются)



Namespace (вкладываются и пересекаются)



Класс (вкладываются)



Метод/Функция (не вкладываются)

Хранение кода

Demo

В интерактивной презентации здесь была бы демонстрация предыдущей реализации концепции хранения кода, описанной ниже

Хранение кода

Специфика HardCode

- Код хранится в специальной БД версионно
- Все сущности (интерфейсы, классы, энумерации, функции, ...) хранятся в единообразном дереве
- Линковочная информация генерируется автоматически и хранится явно
- Принимается только валидный код
- Доступ осуществляется через IDE
- Линковка реализуется динамически
- Зависимости модулей энкапсулируются

Хранение кода

Преимущества решения в HardCode

- Разработка “на лету”
- Единообразная структура сущностей с гибкой факторизацией
- Строгая testableность кода, не касающегося IO
- Mock-driven design “из коробки” для тестирования IO
- Embedding (как в Lua, Python, JS и т. п., но лучше)
- Sandboxing (как в Lua но лучше)
- Консистентная (кодо)генерация с серверной стороны (например, Protobuf)
- Управление процессами сопровождающими разработку:
 - Запуск автотестов
 - Pull request, rebase, virtual rebase, review
 - Валидация (на сервере) содержимого продовых веток

Хранение кода

Capabilities (фичи задаваемые в ветках)

- Позволять использовать GC
- Позволять использовать DAG кучу
- Разрешать функции без гарантий завершения
- Разрешать грязный код (с WARNING'ами):
полезный для процесса разработки, недопустимый для прода
- Разрешать управление потоками
- Позволять только чистую историю
(аналог `merge --no-ff` только когда возможен `merge --ff-only`)

Хранение кода

Пример рефакторинга: исходная версия

```
template <interface real $real>
math.mat4 math.mat4.inverse_affine () {
    mut $real v0 = m[2][0]*m[3][1] - m[2][1]*m[3][0]
    mut $real v1 = m[2][0]*m[3][2] - m[2][2]*m[3][0]
    mut $real v2 = m[2][0]*m[3][3] - m[2][3]*m[3][0]
    mut $real v3 = m[2][1]*m[3][2] - m[2][2]*m[3][1]
    mut $real v4 = m[2][1]*m[3][3] - m[2][3]*m[3][1]
    mut $real v5 = m[2][2]*m[3][3] - m[2][3]*m[3][2]

    $real t00 = + (v5*m[1][1] - v4*m[1][2] + v3*m[1][3])
    $real t10 = - (v5*m[1][0] - v2*m[1][2] + v1*m[1][3])
    $real t20 = + (v4*m[1][0] - v2*m[1][1] + v0*m[1][3])
    $real t30 = - (v3*m[1][0] - v1*m[1][1] + v0*m[1][2])

    $real inv_det = 1.0/(t00*m[0][0] + t10*m[0][1] + t20*m[0][2] + t30*m[0][3])

    retval[0][0] = t00*inv_det
    retval[1][0] = t10*inv_det
    retval[2][0] = t20*inv_det
    retval[3][0] = t30*inv_det

    retval[0][1] = - (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][1] = + (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][1] = - (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][1] = + (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det

    v0 := m[1][0]*m[3][1] - m[1][1]*m[3][0]
    v1 := m[1][0]*m[3][2] - m[1][2]*m[3][0]
    v2 := m[1][0]*m[3][3] - m[1][3]*m[3][0]
    v3 := m[1][1]*m[3][2] - m[1][2]*m[3][1]
    v4 := m[1][1]*m[3][3] - m[1][3]*m[3][1]
    v5 := m[1][2]*m[3][3] - m[1][3]*m[3][2]

    retval[0][2] = + (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][2] = - (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][2] = + (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][2] = - (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det

    v0 := m[2][1]*m[1][0] - m[2][0]*m[1][1]
    v1 := m[2][2]*m[1][0] - m[2][0]*m[1][2]
    v2 := m[2][3]*m[1][0] - m[2][0]*m[1][3]
    v3 := m[2][2]*m[1][1] - m[2][1]*m[1][2]
    v4 := m[2][3]*m[1][1] - m[2][1]*m[1][3]
    v5 := m[2][3]*m[1][2] - m[2][2]*m[1][3]

    retval[0][3] = - (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][3] = + (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][3] = - (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][3] = + (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det
}
```

Хранение кода

Пример рефакторинга: шаг 1

```
math.mat4 math.mat4.inverse_affine()
{
    $real inv_det
    {
        $real v0 = m[2][0]*m[3][1] - m[2][1]*m[3][0]
        $real v1 = m[2][0]*m[3][2] - m[2][2]*m[3][0]
        $real v2 = m[2][0]*m[3][3] - m[2][3]*m[3][0]
        $real v3 = m[2][1]*m[3][2] - m[2][2]*m[3][1]
        $real v4 = m[2][1]*m[3][3] - m[2][3]*m[3][1]
        $real v5 = m[2][2]*m[3][3] - m[2][3]*m[3][2]

        $real t00 = + (v5*m[1][1] - v4*m[1][2] + v3*m[1][3])
        $real t10 = - (v5*m[1][0] - v2*m[1][2] + v1*m[1][3])
        $real t20 = + (v4*m[1][0] - v2*m[1][1] + v0*m[1][3])
        $real t30 = - (v3*m[1][0] - v1*m[1][1] + v0*m[1][2])

        inv_det = 1.0/(t00*m[0][0] + t10*m[0][1] + t20*m[0][2] + t30*m[0][3])

        retval[0][0] = t00*inv_det
        retval[1][0] = t10*inv_det
        retval[2][0] = t20*inv_det
        retval[3][0] = t30*inv_det

        retval[0][1] = - (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
        retval[1][1] = + (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
        retval[2][1] = - (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
        retval[3][1] = + (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det
    }

    $real v0 = m[1][0]*m[3][1] - m[1][1]*m[3][0]
    $real v1 = m[1][0]*m[3][2] - m[1][2]*m[3][0]
    $real v2 = m[1][0]*m[3][3] - m[1][3]*m[3][0]
    $real v3 = m[1][1]*m[3][2] - m[1][2]*m[3][1]
    $real v4 = m[1][1]*m[3][3] - m[1][3]*m[3][1]
    $real v5 = m[1][2]*m[3][3] - m[1][3]*m[3][2]

    retval[0][2] = + (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][2] = - (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][2] = + (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][2] = - (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det
}

{
    $real v0 = m[2][1]*m[1][0] - m[2][0]*m[1][1]
    $real v1 = m[2][2]*m[1][0] - m[2][0]*m[1][2]
    $real v2 = m[2][3]*m[1][0] - m[2][0]*m[1][3]
    $real v3 = m[2][2]*m[1][1] - m[2][1]*m[1][2]
    $real v4 = m[2][3]*m[1][1] - m[2][1]*m[1][3]
    $real v5 = m[2][3]*m[1][2] - m[2][2]*m[1][3]

    retval[0][3] = - (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][3] = + (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][3] = - (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][3] = + (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det
}
```

Хранение кода

Пример рефакторинга: шаг 2

```
void math.mat4.inverse_affine.`.fill_col01` (out math.mat4<$real> ret, out $real inv_det)
{
    $real v0 = m[2][0]*m[3][1] - m[2][1]*m[3][0]
    $real v1 = m[2][0]*m[3][2] - m[2][2]*m[3][0]
    $real v2 = m[2][0]*m[3][3] - m[2][3]*m[3][0]
    $real v3 = m[2][1]*m[3][2] - m[2][2]*m[3][1]
    $real v4 = m[2][1]*m[3][3] - m[2][3]*m[3][1]
    $real v5 = m[2][2]*m[3][3] - m[2][3]*m[3][2]

    $real t00 = + (v5*m[1][1] - v4*m[1][2] + v3*m[1][3])
    $real t10 = - (v5*m[1][0] - v2*m[1][2] + v1*m[1][3])
    $real t20 = + (v4*m[1][0] - v2*m[1][1] + v0*m[1][3])
    $real t30 = - (v3*m[1][0] - v1*m[1][1] + v0*m[1][2])

    inv_det = 1.0/(t00*m[0][0] + t10*m[0][1] + t20*m[0][2] + t30*m[0][3])

    retval[0][0] = t00*inv_det
    retval[1][0] = t10*inv_det
    retval[2][0] = t20*inv_det
    retval[3][0] = t30*inv_det

    retval[0][1] = - (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][1] = + (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][1] = - (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][1] = + (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det
}
```

Хранение кода

Пример рефакторинга: шаг 2

```
void math.mat4.inverse_affine.`.fill_col2` (out math.mat4<$real> ret, $real& inv_det)
{
    $real v0 = m[1][0]*m[3][1] - m[1][1]*m[3][0]
    $real v1 = m[1][0]*m[3][2] - m[1][2]*m[3][0]
    $real v2 = m[1][0]*m[3][3] - m[1][3]*m[3][0]
    $real v3 = m[1][1]*m[3][2] - m[1][2]*m[3][1]
    $real v4 = m[1][1]*m[3][3] - m[1][3]*m[3][1]
    $real v5 = m[1][2]*m[3][3] - m[1][3]*m[3][2]

    retval[0][2] = + (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][2] = - (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][2] = + (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][2] = - (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det
}
```

Хранение кода

Пример рефакторинга: шаг 2

```
void math.mat4.inverse_affine.`fill_col3` (
    out math.mat4<$real> ret, $real& inv_det)
{
    $real v0 = m[2][1]*m[1][0] - m[2][0]*m[1][1]
    $real v1 = m[2][2]*m[1][0] - m[2][0]*m[1][2]
    $real v2 = m[2][3]*m[1][0] - m[2][0]*m[1][3]
    $real v3 = m[2][2]*m[1][1] - m[2][1]*m[1][2]
    $real v4 = m[2][3]*m[1][1] - m[2][1]*m[1][3]
    $real v5 = m[2][3]*m[1][2] - m[2][2]*m[1][3]

    retval[0][3] = - (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][3] = + (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][3] = - (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][3] = + (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det
}
```

Хранение кода

Пример рефакторинга: шаг 2

```
- math
- mat4
- inverse_affine
  - .fill_col01
  - .fill_col2
  - .fill_col3

math.mat4 math.mat4.inverse_affine ()
{
    $real inv_det
    .fill_col01 (retval=, inv_det=)
    .fill_col2 (retval=, inv_det)
    .fill_col3 (retval=, inv_det)
}
```

Хранение кода

Пример рефакторинга: шаг 3

```
void math.mat4.inverse_affine.`.fill_col01`.`calc_det` ($real& t00, $real& t10, $real& t20, $real& t30)
{
    return 1.0/(t00*m[0][0] + t10*m[0][1] + t20*m[0][2] + t30*m[0][3])
}
```

Хранение кода

Пример рефакторинга: шаг 3

```
void math.mat4.inverse_affine.`.fill_col01` (out math.mat4<$real> ret, out $real inv_det)
{
    $real v0 = m[2][0]*m[3][1] - m[2][1]*m[3][0]
    $real v1 = m[2][0]*m[3][2] - m[2][2]*m[3][0]
    $real v2 = m[2][0]*m[3][3] - m[2][3]*m[3][0]
    $real v3 = m[2][1]*m[3][2] - m[2][2]*m[3][1]
    $real v4 = m[2][1]*m[3][3] - m[2][3]*m[3][1]
    $real v5 = m[2][2]*m[3][3] - m[2][3]*m[3][2]

    $real t00 = + (v5*m[1][1] - v4*m[1][2] + v3*m[1][3])
    $real t10 = - (v5*m[1][0] - v2*m[1][2] + v1*m[1][3])
    $real t20 = + (v4*m[1][0] - v2*m[1][1] + v0*m[1][3])
    $real t30 = - (v3*m[1][0] - v1*m[1][1] + v0*m[1][2])

    inv_det = .calc_det (t00, t10, t20, t30)

    retval[0][0] = t00*inv_det
    retval[1][0] = t10*inv_det
    retval[2][0] = t20*inv_det
    retval[3][0] = t30*inv_det

    retval[0][1] = - (v5*m[0][1] - v4*m[0][2] + v3*m[0][3])*inv_det
    retval[1][1] = + (v5*m[0][0] - v2*m[0][2] + v1*m[0][3])*inv_det
    retval[2][1] = - (v4*m[0][0] - v2*m[0][1] + v0*m[0][3])*inv_det
    retval[3][1] = + (v3*m[0][0] - v1*m[0][1] + v0*m[0][2])*inv_det
}
```

Хранение кода

Пример рефакторинга: шаг 3

```
- math
- mat4
  - inverse_affine
    - .fill_col01
      - .calc_det
    - .fill_col2
    - .fill_col3

math.mat4 math.mat4.inverse_affine ()
{
    $real inv_det
    .fill_col01 (retval=, inv_det=)
    .fill_col2 (retval=, inv_det)
    .fill_col3 (retval=, inv_det)
}
```

Динамическая линковка

Зачем нужна динамическая типизация?

Только для упрощения динамической линковки (пример на JS):

```
function (a, b) {  
    console.log(a, b)  
}  
  
function (...args) {  
    var a = args[0]  
    var b = args[1]  
}
```

- все функции идентичны для линкера

Динамическая линковка

Динамическая линковка, статическая типизация

- Удобно благодаря генерации и хранению линковочной информации в VCS и runtime
- Производительнее динамической типизации
- Исключает (не само по себе) фатальные ошибки времени исполнения
- Позволяет автоматически перекомпилировать зависимые функции
- Embedding
- Sandboxing
- Настоящий Embedding и Sandboxing в HardCode

Анализ сложности

Заблуждение из школы

Тезис:

«Нельзя доказать завершимость
произвольной Тюринг-полной программы»



Заблуждение:

«Нас интересуют Тюринг-полные программы»

Анализ сложности

Ещё одно заблуждение из школы

«Рекурсии нужны»

Анализ сложности

Альтернативное мнение

- Рекурсии очень нужны!
Чтобы писать неанализируемые программы.
- Зависающие программы очень нужны!
Чтобы разработчик оставался на поддержке пожизненно.
Job security как она есть.

Анализ сложности

Ну а серьёзно

- Вспомогательные вещи нужные при разработке:
 - рекурсии
 - непрерываемые бесконечные циклы
 - неиспользованные переменные и прочие недоделки, вызывающие сообщения уровня WARNING компилятора

Рекурсии и бесконечные циклы удобны в процессе реализации сложных рекурсивных алгоритмов. Работа с “грязным” кодом удобна для рефакторинга и отладки (попробуйте порефакторить в Golang – посетят суицидальные мысли). Но всё это неприемлимо в production ветке.
- Обычно рекурсии не сильно дружат с анализом.
А вот рекурсивные паттерны можно попробовать подружить.

Анализ сложности

Программы здорового человека

- Программа должна гарантировать прерываемость вне зависимости от входных данных.
Это гарантируется **языком программирования**.
- В языке программирования исключается неанализируемое поведение:
 - незащищённые по памяти операции
 - оператор безусловного перехода `goto`
 - циклы без строгих условий выхода (можно в ветке разработки)
 - неопределённое поведение из-за ошибок синхронизации
 - бутылочные горлышки управления памятью
 - рекурсии (можно в ветке разработки)

Анализ сложности

Пример анализа: Heapsort

```
void heapify (mut int[]&:check (#this > 0) elements)
void extract_top (mut int[]&:check (#this > 0) elements)
void drown_top (mut int[]&:check (#this > 0) elements)

void sort (mut int[]& elements)
{
    mut int[]&:check (#this > 0) elements_ne = must elements
    heapify (elements_ne)
    index (elements_ne) head = 0
    for desc (index (elements_ne) c: #elements_ne - 1, 0) {
        extract_top (elements_ne[head:c + 1])
        drown_top (elements_ne[head:c])
    }
}
```

Анализ сложности

Реализация Heapsort

```
void heapify (mut int[]&:check (#this > 0) elements)
{
    word:check (#this > 0) element_count = #elements
    word height = log2floor (element_count)
    for (word h: 0, height) {
        word b = (1 << height) - 2
        word e = (1 << h) >> 1
        for desc (word i: b, e - 1) {
            index (elements) idx1 = must i
            index (elements) idx2 = must (i*2 + 1)
            if (elements[idx1] < elements[idx2])
                swap (elements[idx1], elements[idx2])
            index (elements) idx3 = must (idx2 + 1)
            if (elements[idx1] < elements[idx3])
                swap (elements[idx1], elements[idx3])
        }
    }
}
```

```
void heapify (int* elements, ssize_t count)
{
    ssize_t height = log2floor (count); // Undefined behavior for count <= 0
    for (ssize_t h = 0; h < height; ++h) {
        ssize_t b = (1 << height) - 2;
        ssize_t e = (1 << h) >> 1;
        for (ssize_t i = b; i >= e; --i) {
            ssize_t idx1 = i;
            if (idx1 >= count)
                return;
            ssize_t idx2 = i*2 + 1;
            if (idx2 >= count)
                return;
            if (elements[idx1] < elements[idx2])
                swap (&elements[idx1], &elements[idx2]);
            ssize_t idx3 = idx2 + 1;
            if (idx3 >= count)
                return;
            if (elements[idx1] < elements[idx3])
                swap (&elements[idx1], &elements[idx3]);
        }
    }
}
```

Анализ сложности

Реализация Heapsort

```
void extract_top (mut int[]&:check (#this > Ø) elements)
{
    index (elements) head = Ø
    index (elements) tail = #elements - 1
    swap (elements[head], elements[tail])
}
```

```
void extract_top (int* elements, ssize_t count)
{
    swap (&elements[Ø], &elements[count - 1]);
}
```

Анализ сложности

Реализация Heapsort

```
void drown_top (mut int[]&:check (#this > 0) elements)
{
    word:check (this > 0) element_count = #elements
    word height = log2floor (element_count)
    index (elements) idx = 0
    for (word _ : 0, height) {
        index (elements) idx1 = idx
        index (elements) idx2 = must (idx*2 + 1)
        let (index (elements) idx3 = idx2 + 1) {
            if (elements[idx1] < elements[idx2] &&
                elements[idx2] > elements[idx3]) {
                swap (elements[idx1], elements[idx2])
                idx = idx2
            } else if (elements[idx1] < elements[idx3]) {
                swap (elements[idx1], elements[idx3])
                idx = idx3
            } else {
                return
            }
        } else {
            if (elements[idx1] < elements[idx2])
                swap (elements[idx1], elements[idx2])
            return
        }
    }
}
```

```
void drown_top (int* elements, ssize_t count)
{
    ssize_t idx = 0;
    ssize_t height = log2floor (count); // Undefined behavior for count <= 0
    for (ssize_t i = 0; i < height; ++i) {
        ssize_t idx1 = idx;
        ssize_t idx2 = idx*2 + 1;
        ssize_t idx3 = idx2 + 1;
        if (idx2 >= count)
            return;
        if (idx3 >= count) {
            if (elements[idx1] < elements[idx2])
                swap (&elements[idx1], &elements[idx2]);
            return;
        }
        if (elements[idx1] < elements[idx2] &&
            elements[idx2] > elements[idx3]) {
            swap (&elements[idx1], &elements[idx2]);
            idx = idx2;
        } else if (elements[idx1] < elements[idx3]) {
            swap (&elements[idx1], &elements[idx3]);
            idx = idx3;
        } else {
            return;
        }
    }
}
```

Анализ сложности

Реализация Heapsort

```
void sort (mut int[]& elements)
{
    mut int[]:<check (#this > Ø) elements_ne = must elements
    heapify (elements_ne)
    index (elements_ne) head = Ø
    for desc (index (elements_ne) c: #elements_ne - 1, Ø) {
        extract_top (elements_ne[head:c + 1])
        drown_top (elements_ne[head:c])
    }
}
```

```
void sort (int* elements, ssize_t count)
{
    if (count <= Ø)
        return;
    heapify (elements, count);
    for (ssize_t c = count; c > 1; --c) {
        extract_top (elements, c);
        drown_top (elements, c - 1);
    }
}
```

Анализ сложности

Heapsort: классификация

```
void heapify (mut int[]&:check (#this > 0) elements)
{
    word:check (<this > 0) element_count = #elements // load (ram, 64)
    word height = log2floor (element_count) // bitwise (64)
    for (word h: 0, height) {
        // log2floor (#elements) x iarith (64)
        // log2floor (#elements) x icmp (64)
        // log2floor (#elements) x {
        word b = (1 << height) - 2 // bitwise (64)
        word e = (1 << h) >> 1 // iarith (64)
        for desc (word i: b, e - 1) {
            // bitwise (64)
            // copy (reg, 64)
            // copy (reg, 64)
            // copy (reg, 64)
            // copy (reg, 64)
            // iarith (64)
            // (((1 << h) >> 1) - (1 << log2floor (#elements)) + 2) x iarith (64)
            // (((1 << h) >> 1) - (1 << log2floor (#elements)) + 2) x icmp (64)
            // (((1 << h) >> 1) - (1 << log2floor (#elements)) + 2) x {
            index (elements) idx1 = must i // copy (reg, 64)
            index (elements) idx2 = must (i*2 + 1) // icmp (reg, 64)
            if (elements[idx1] < elements[idx2]) // iarith (64)
                // iarith (64)
                // icmp (reg, 64)
                // load (ram, 32)
                // load (ram, 32)
                // icmp (32)
                swap (elements[idx1], elements[idx2]) // load (ram, 32)
                // load (ram, 32)
                // copy (reg, 32)
                // store (ram, 32)
                // store (ram, 32)
                // iarith (64)
                index (elements) idx3 = must (idx2 + 1) // icmp (reg, 64)
                if (elements[idx1] < elements[idx3]) // load (ram, 32)
                    // load (ram, 32)
                    // icmp (32)
                    swap (elements[idx1], elements[idx3]) // load (ram, 32)
                    // load (ram, 32)
                    // copy (reg, 32)
                    // store (ram, 32)
                    // store (ram, 32)
                    // } // (b - e)
            } // log2floor(#elements)
        }
    }
}
```

Анализ сложности

Heapsort: классификация

```
void extract_top (mut int[]&:check (#this > Ø) elements)
{
    index (elements) head = Ø                                // copy (reg, 64)
    index (elements) tail = #elements - 1                   // iarith (64)
    swap (elements[head], elements[tail])                  // load (ram, 32)
                                                               // load (ram, 32)
                                                               // copy (reg, 32)
                                                               // store (ram, 32)
                                                               // store (ram, 32)
}
}
```

Анализ сложности

Heapsort: классификация

```
void drown_top (mut int[]&:check (#this > 0) elements)
{
    word:check (this > 0) element_count = #elements
    word height = log2floor (element_count)
    index (elements) idx = 0
    for (word _ : 0, height) {

        index (elements) idx1 = idx
        index (elements) idx2 = must (idx*2 + 1)

        let (index (elements) idx3 = idx2 + 1) {
            if (elements[idx1] < elements[idx2] &&
                elements[idx2] > elements[idx3]) {
                swap (elements[idx1], elements[idx2])
                idx = idx2
            } else if (elements[idx1] < elements[idx3]) {
                swap (elements[idx1], elements[idx3])
                idx = idx3
            } else {
                return
            }
        } else {
            if (elements[idx1] < elements[idx2])
                swap (elements[idx1], elements[idx2])
        }
    }
    return
}
```

Анализ сложности

Heapsort: классификация

```
void sort (mut int[]& elements)
{
    mut int[]:<check (#this > 0) elements_ne = must elements // icmp (64)
    heapify (elements_ne) // call "heapify (elements_ne)"
    index (elements_ne) head = 0 // set (reg, 64)
    for desc (index (elements_ne) c: #elements_ne - 1, 0) {
        // iarith (64)
        // (#elements - 1) x iarith (64)
        // (#elements - 1) x icmp (64)
        // (#elements - 1) x {
        extract_top (elements_ne[head:c + 1]) // iarith (64)
        // slice
        // call "extract_top (elements_ne[head:c + 1])"
        // copy (64)
        // slice
        // call "drown_top (elements_ne[head:c])"
        // }
    }
}
```

Анализ сложности

Правила назначения стоимости

```
load (ram) = 1 + 1 mt (ram)
store (ram) = 1 + 1 mt (ram)
copy (reg) = 1
set (reg) = 1
bitwise = 2
bool = 2
iarith = 5
icmp = 2
slice = 2 + 2 mt (ram)
```

Анализ сложности

Применение стоимости

```
void heapify (mut int[]&:check (#this > 0) elements)
{
    word:check (#this > 0) element_count = #elements
    word height = log2floor (element_count)
    for (word h: 0, height) {
        // 1 + 1 mt (ram)
        // 2
        // 1
        // log2floor (#elements) x 5
        // log2floor (#elements) x 2
        // log2floor (#elements) x {
        //   2
        //   5
        //   2
        //   2
        //   1
        //   5
        //   (((1 << h) >> 1) - (1 << log2floor (#elements)) + 2) x 5
        //   (((1 << h) >> 1) - (1 << log2floor (#elements)) + 2) x 2
        //   (((1 << h) >> 1) - (1 << log2floor (#elements)) + 2) x {
            index (elements) idx1 = must i
            // 1
            // 2
            // 5
            // 5
            // 2
            if (elements[idx1] < elements[idx2])
                // 1 + 1 mt (ram)
                // 1 + 1 mt (ram)
                swap (elements[idx1], elements[idx2])
                // 1 + 1 mt (ram)
                // 1 + 1 mt (ram)
                // 1
                // 1 + 1 mt (ram)
                // 1 + 1 mt (ram)
            index (elements) idx3 = must (idx2 + 1)
            // 5
            // 2
            if (elements[idx1] < elements[idx3])
                // 1 + 1 mt (ram)
                // 1 + 1 mt (ram)
                swap (elements[idx1], elements[idx3])
                // 1 + 1 mt (ram)
                // 1 + 1 mt (ram)
                // 1
                // 1 + 1 mt (ram)
                // 1 + 1 mt (ram)
                // } // (b - e)
        } // log2floor(#elements)
    }
}
```

Анализ сложности

Применение стоимости

```
void heapify (mut int[]&:check (#this > 0) elements)
{
    // 4 + 1 mt (ram)
    // log2floor (#elements) x 24
    // log2floor (#elements) x { | h: [0, log2floor (#elements)); asc
    //     (((1 << h) >> 1) - (1 << log2floor (#elements)) + 2) x 47 + 12 mt (ram)
    // }
}
void extract_top (mut int[]&:check (#this > 0) elements)
{
    // 13 + 4 mt (ram)
}
void drown_top (mut int[]&:check (#this > 0) elements)
{
    // 5 + 1 mt (ram)
    // log2floor (#elements) x 45 + 10 mt (ram)
}
void sort (mut int[]& elements)
{
    // 8
    // call "heapify (elements_ne)"
    // (#elements - 1) x 17 + 4 mt (ram)
    // (#elements - 1) x { | c: [#elements_ne - 1, 0); desc
    //     17 + 4 mt (ram)
    //     call "extract_top (elements_ne[0:c + 1])"
    //     call "drown_top (elements_ne[0:c])"
    // }
}
```

Анализ сложности

Применение стоимости

```
void sort (mut int[]& elements)
{
    // 12 + 1 mt (ram)
    // log2floor (#elements) x 24
    // log2floor (#elements) x { | h: [0, log2floor (#elements)); asc
    //     max (((1 << h) >> 1) - (1 << log2floor (#elements)) + 2, 0) x 47 + 12 mt (ram)
    // }
    // max (0, #elements - 1) x 53 + 13 mt (ram)
    // max (0, #elements - 1) x { | c: [#elements - 1, 0); desc
    //     log2floor (c + 1) x 45 + 10 mt (ram)
    // }
}
```

Анализ сложности

Применение стоимости (Пузырьковая сортировка)

```
void bubble_sort (mut int[]& elements)
{
    // 1
    // max (0, #elements - 1) x 13
    // max (0, #elements - 1) x {
    //     j x { | j: [#elements - 2, 0); desc
    //           16 + 6 mt (ram)
    //     }
    // }
}
```

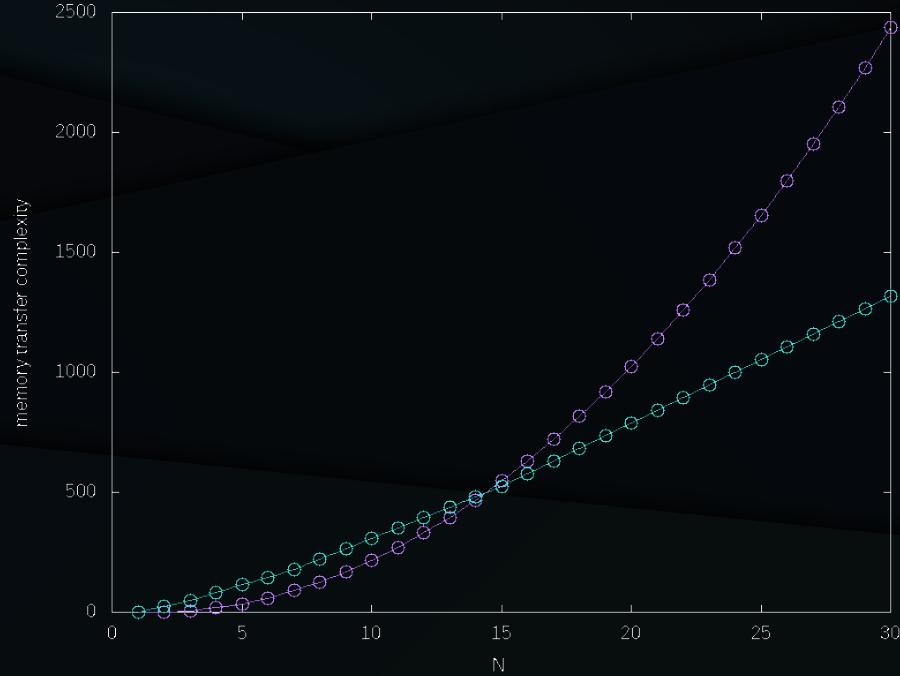
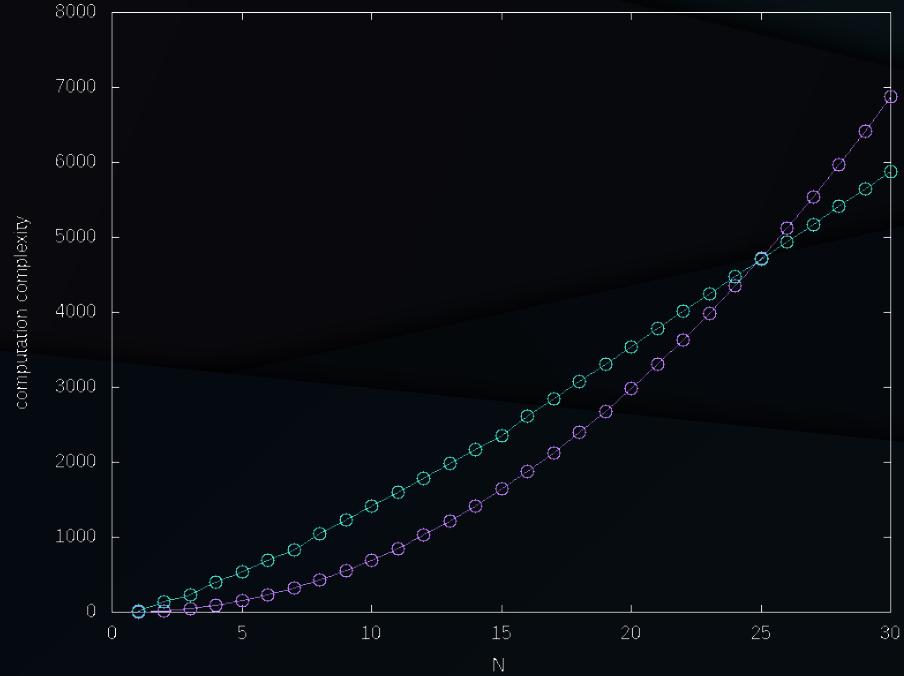
Анализ сложности

Подстановка размерностей

```
void test_heap_sort ()
{
{
    mut int[]& elements = make_random(8)
    heap_sort(elements) // 1040 + 222 mt
}
{
    mut int[]& elements = make_random(27)
    heap_sort(elements) // 5176 + 1159 mt
}
}
void test_bubble_sort ()
{
{
    mut int[]& elements = make_random(8)
    bubble_sort(elements) // 428 + 126 mt
}
{
    mut int[]& elements = make_random(27)
    bubble_sort(elements) // 5539 + 1950 mt
}
}
```

Анализ сложности

Объединение алгоритмов



Анализ сложности

Анализ конкретного случая

- Учитываем условные переходы `if`, `let`, `must`, `match`
- Учитываем `return` и `break`
- Получаем стоимость реально (а не возможно) выполненных инструкций

Анализ сложности

Анализ распределения

- Генерируем случайные данные
- Или берём данные с прода
- А иногда даже автоматически генерим полное покрытие
- Получаем стоимость реально (а не возможно) выполненных инструкций

Анализ сложности

Стохастический анализ

- Хэш массивы имеют худший случай $O(N)$ для доступа к элементу
- В таком случае анализ худшего случая бесполезен
- Придётся ввести дополнительный параметр к анализу:
послабление к функциям сильно зависящим от распределения данных
- Объекты, отмеченные как стохастические должны быть
подконтрольны компилятору (подобно IO объектам)
- Таким анализ функций использующих стохастические объекты
будет производиться с учётом усреднённого распределения данных

Анализ сложности

Анализ сложности до точки прерывания

- Функции, работающие с системными ресурсами, например сетью, основное время работы находятся в прерываемом ожидании
- В таком случае нас также интересует, какова сложность самого длинного непрерываемого участка
- Прерывание вызывается изменением состояния системного ресурса, в том числе, таймаутом
- Само прерывание гарантируется компилятором

Анализ сложности

Анализ сложности до точки прерывания

```
variant<message|error|null> read_and_parse (context, mut io.socket& socket, word& size_limit)
{
    mut request_parser parser = {}
    loop (!parser.finished ()) {
        select (context, socket) {           // Interruption point
            case (error@ err) {
                return err
            }
            case (enum io.EOF) {
                break
            }
            case (enum context.interrupt) {
                break
            }
        }
        loop {
            ubyte[] buffer = socket.read (4096) // Non-blocking read
            if (#buffer == 0)
                break
            parser.consume (buffer)
            select (context, socket) {           // Interruption point
                case (enum context.interrupt) {
                    break
                }
            }
            if (parser.finished ())
                break
        }
    }
    return parser.message ()
}
```

Анализ сложности

Корутины в HardCode

- Корутины реализованы как генераторы – специальный “`reentrant`” функции
- Позволяют использовать произвольные планировщики:
 - » с приоритетом на скорость завершения
 - » с приоритетом на параллелизм
 - » самописные – на что хватит фантазии разработчика
- Анализ сложности до точки прерывания позволяет находить нужный баланс между отзывчивостью и производительностью
- “`yield`” возможен только непосредственно в теле “`reentrant`” функций

Анализ сложности

Анализ инструкций CPU

- Позволяет точнее оценивать сложность на конкретном CPU
- Не требует доступа к конкретной модели CPU
- Не входит в MVP

Анализ сложности

Анализ с учётом кэширования CPU

- Овеян мистическим полумраком
- Моделирование кэширования теоретически возможно
- Надёжность такого моделирования требует исследования
- В MVP не входит

Анализ сложности

Применение при разработке сервисов

- Определяем сложность худшего случая полного вычисления
- Определяем сложность худшего случая вычисления между точками прерывания (если есть)
- Если нас устраивает сложность полного вычисления, считаем разработку компоненты завершённой
- Иначе, если нас устраивает сложность вычисления между точками прерывания, считаем разработку компоненты завершённой
- Иначе дорабатываем программу и повторяем анализ

Анализ сложности

Общее применение в IDE

- Сравнение производительности алгоритмов
- Поиск ближайшей компоненты для оптимизации (бутылочного горлышка)
- Регрессионный анализ
- Анализ на боевых данных

Sandboxing

Что такое Sandboxing

```
#!/usr/bin/lua

require "canvas"

local user_func_source = get_untrusted_function()      -- Получаем функцию из недоверенного источника
--[[[
draw_pixel (10, 20, 0xff0000)
draw_pixel (30, 40, 0x00ff00)
draw_pixel (200, 500, 0x0000ff)
]]]

local env_proxy = {
    draw_pixel = canvas.draw_pixel,                      -- Пользовательской среде доступна только эта функция
}

local env = setmetatable ({}, {__index = env_proxy}) -- В таком режиме пользовательскую среду нельзя изменить изнутри

local user_func = assert(load (user_func_source, "user func", "t", env))

user_func ()                                         -- Безопасно запускаем недоверенную функцию
```

Sandboxing

Sandboxing в наши дни

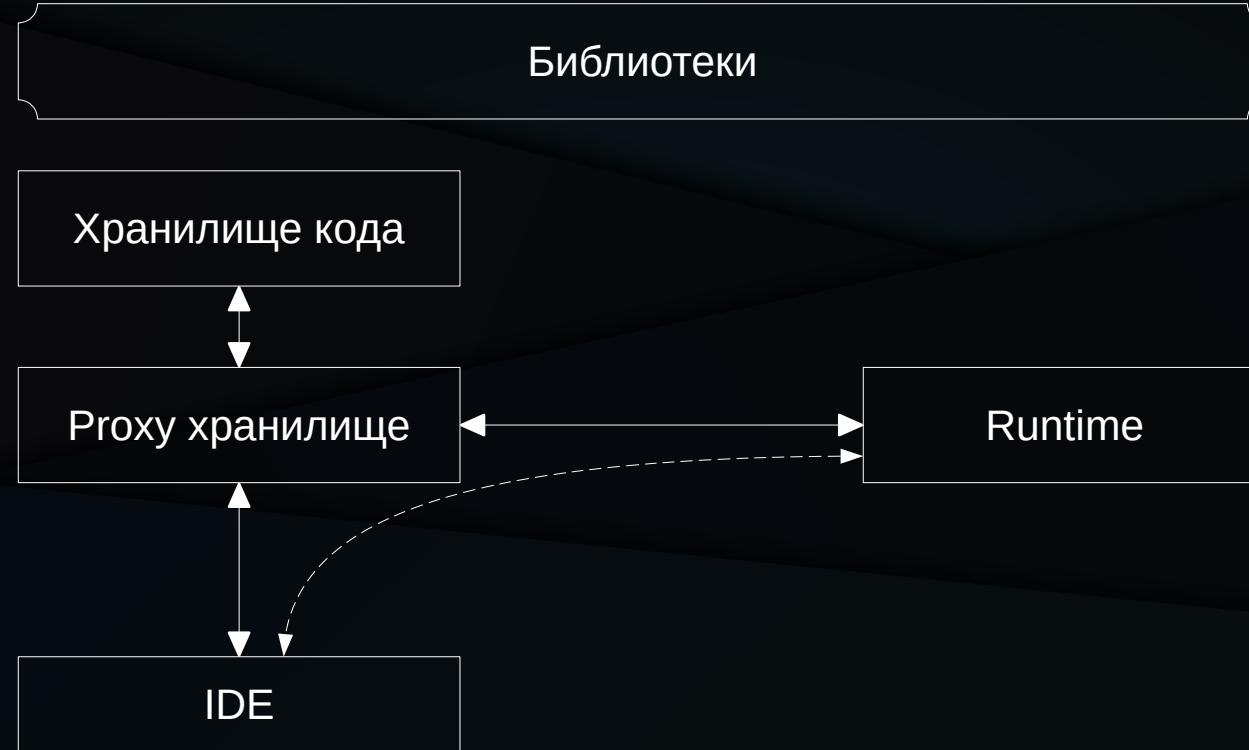
```
function small_little_fast_user_callback()
    while true do
        -- Функцию не прервать не убивая приложения!!
    end
end
```

Sandboxing

Sandboxing в HardCode

- Позволяет запрещать функции:
 - » без гарантий завершаемости
 - » превышающие квоту сложности выполнения
 - » превышающие квоту сложности пути до точки прерывания
- Гарантирует заданную отзывчивость
- Имеет производительность основной среды со статической типизацией
- Всё вышеперечисленное распространяется на Embedding

Архитектура



Архитектура

Библиотеки

- Синтаксический анализатор
- Семантический анализатор
- Линкер
- Бэкэнды:
 - » инструкции в памяти
 - » код на C
 - » инструкции VM (для отладочного режима)
- Анализатор операций
- Обработчик runtime
 - » Ядро
 - » Garbage Collector
 - » Загрузчик модулей для байндингов
- Сериализатор/десериализатор (для хранилища кода)

Сравнение с другими языками

Ниже всех, Выше всех

- Высокоуровневая семантика
 - » Отсутствие необрабатываемых ошибок времени выполнения
 - » Отсутствие ошибок синхронизации
 - » Удобная модель памяти: RAII и GC одновременно
 - » Динамический полиморфизм из коробки
 - » Динамическая линковка, статическая типизация
 - » Настоящий Embedding, настоящий Sandboxing
 - » Строгая анализируемость
 - » Консистентное версионное хранение кода
- Низкоуровневая оптимизация
 - » Оптимизированная модель памяти
 - » Возможность произвольно инлайнить функции
 - » Предсказуемое использование стэка
 - » Адаптивность соглашения вызова функции
 - » Строгие семантические гарантии для оптимизации семантического дерева
 - » Строгие оптимизации на основе правил strict aliasing

Применение

- Замена всех популярных языков общего назначения
- Конструирование DSL: Domain Specific Language
- Браузерная платформа общего назначения
- Service as a Service платформа
- Embedding в GameDev

Платформа Service as a Service

Разработка сервисов

- На сегодняшний день практики разработки сервисов сводятся к docker-compose: настройка всей сборки может занять пару недель
- Платформа Service as a Service позволяет поднять всю связку сервисов в 2 клика
- Для разработчика нет разницы между монолитом и микросервисами
- Все данные находятся на сервере, разработка производится через 1 клиентскую программу



- Эффективная разработка, гибкая оптимизация
- Быстрая включаемость и высокая мобильность разработчика

Платформа Service as a Service

Что это?

- Service as a Service – эволюция Function as a Service
- Function as a Service – облачный сервис выполнения загруженных функций
- Service as a Service – облачный сервис конструирования сервисов

Платформа Service as a Service

Function as a Service в наши дни

- Можем инициировать вызов функции по сети
- Для каждого вызова функции нужен отдельный процесс
- Завершение функции гарантируется таймаутом
(по истечении процесс обрабатывает запрос убивается)
- Существует реализация stateful functions
- Можно строить дерево вызовов
- Парадигма напоминает комбинаторный генератор оверхедов

Платформа Service as a Service

Service as a Service в HardCode

- Используем полноценный API не ограниченный вызовом функций
- API: эволюция gRPC + SCTP поверх UDP
- API сервиса хранится в версионной БД, гарантирующей консистентность расширения API
- Формат сериализации данных: эволюция Protocol buffers + Flat buffers
- Развитый сетевой протокол API
- Кластерный дизайн из коробки
- Выбор, выносить ли функциональность в отдельный микросервис происходит на этапе конфигурации, а не разработки
- Событийная модель описана в протоколе: завершение выполнение запроса анализируемо
- Завершение клиентской программы, использующей сервис, анализируемо

Платформа Service as a Service

Я ненавижу TCP!!

- Прогрев пула TCP соединений
- Лишние round trip'ы установления соединений
- Нет управления дублированием пакетов
- Повторная отправка пакетов настраивается на уровне ядра
- Таймауты TCP соединения настраиваются на уровне ядра
- Таймауты TCP соединения не сообщаются удалённой стороне на уровне ядра
- TCP соединения не переживают перезапуск прокси серверов
- TCP соединения не переносятся между сетевыми маршрутами
- Поддерживается порядок сообщений там, где он не нужен
- Реализации на базе TCP привязывают сессии к TCP соединениям (gRPC)
- Логику индемпотентности нужно реализовывать самостоятельно



Платформа Service as a Service

О неуместном дизайне: TCP, SCTP и IPsec

- TCP подходит для Hello World, а дальше только мешает
- TCP, SCTP и IPsec реализованы на том уровне, где невозможно регулярное обновление
- Опыт IPv6 и SCTP показал, насколько медленно обновляется прошивка как домашнего, так и магистрального сетевого оборудования
- Логика управления сообщениями и потоками требует тесной интеграции с шифрованием.
Например, в QUIC используется стэпллинг, которого нет в dTLS
- Логика управления сообщениями и потоками и реализация шифрования требуют частого обновления
- Отсюда очевидно: управление сообщениями и потоками с шифрованием должны быть реализованы поверх UDP

Адекватно спроектированный сетевой протокол

- Сетевой протокол целиком построен поверх UDP
- Всё взаимодействие осуществляется через один порт, протокол имеет встроенную реализацию keepalive (опыт на базе проблем с RTP)
- Управление сессиями – часть протокола:
 - сессия переживает как любые сетевые ошибки, так и перезагрузку сервера (если нужно)
 - входящие балансировщики облака не маршрутизируют, а проксируют
 - схема API сервиса (как в gRPC) позволяет осуществлять запросы как привязанные, так и не привязанные к сессии
- Схема API сервиса позволяет описать как поточный обмен, так и одиночные дейтаграммы внутри сессии, а также поточный обмен вне сессии
- Протокол реализует отправку сообщений на базе пула зашифрованных потоков хранящих состояние (используемых через циклический буфер) совмещая таким образом отзывчивость UDP с безопасностью TLS (в отличие от dTLS)
- Управление дублированием и повторной отправкой сообщений так же является частью протокола

Service as a Service platform

Возможности платформы

- Сервис сводится к проектированию API сервисов и реализации обработчиков запросов (как FaaS + Swagger, но не из желудей и спичек)
- Большая часть обёрточной логики уже реализована, конфигурируема и интегрируема платформой
К примеру метрики и логи собираются без потерь “из коробки”
- История развёртывания и клиентских запросов хранится и управляется платформой
- Управление ресурсами осуществляется на базе анализа и умного профилирования
- Данные анализа и профилирования собираются и хранятся платформой
- Детальная изоляция ресурсов осуществляется на уровне языка программирования:
 - Не требуется контейнеров (как в Docker’е)
 - Не требуется даже отдельных процессов для каждого сервиса
- Тем не менее, при необходимости доступна изоляция на уровне системных ресурсов в продакшене, а также возможно создание изолированных сред для разработки в случае небезопасных этапов разработки, таких как разработка байндингов и адаптация рекурсивных алгоритмов

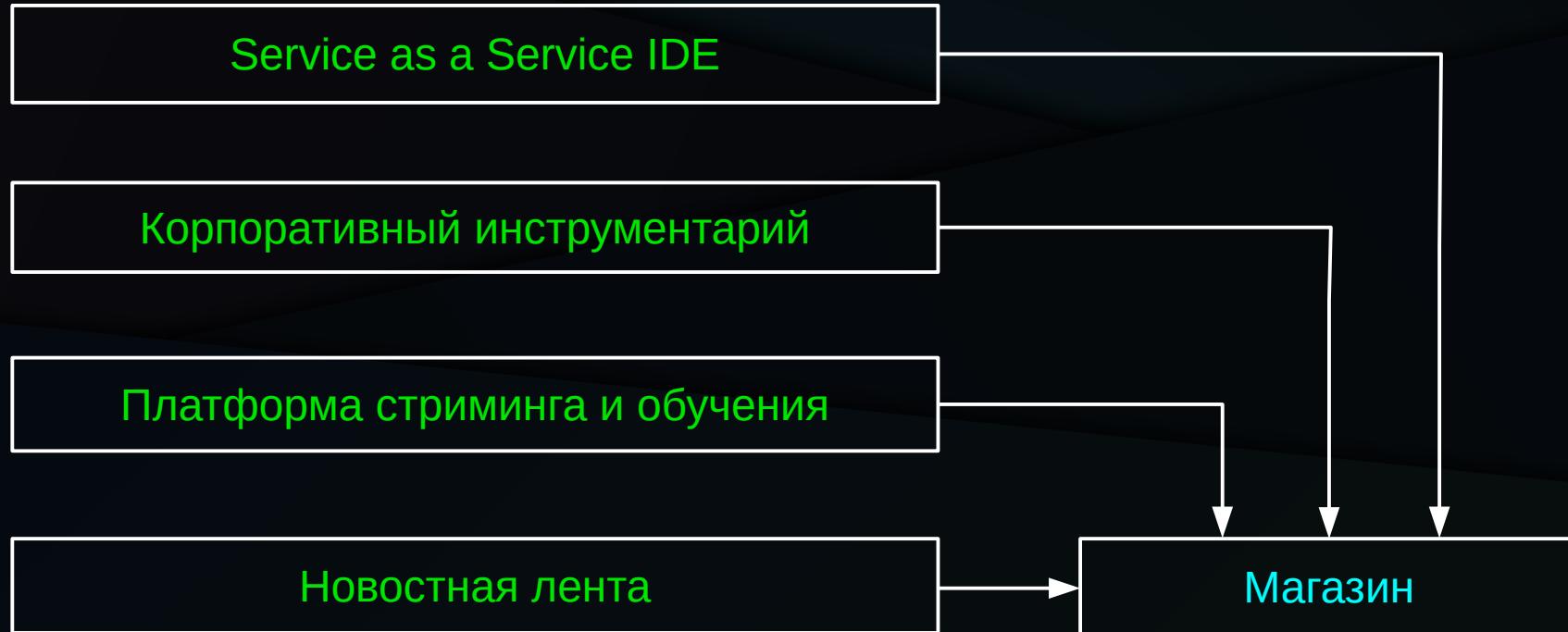
Service as a Service platform

Больше возможностей платформы

- Легковесное развертывание:
 - » деплоим только изменения
 - » не дублируем данные и функции
- Сложные схемы развертывания доступны “из коробки” для любого сервиса
- Централизованная конфигурация сервисов
- Первичное развертывание в облаке не требует конфигурации,*те же 2 клика, что и форк на GitHub’е*
- Дополнительная конфигурация минималистична
в силу отсутствия огромного количества ненужных сущностей и концепций
- Не доверенный код правильно запускается “из коробки”

NextGen IDE

Платформа распространения контента IT инфраструктуры



Service as a Service IDE

- IDE для разработки сервисов
 - Интеграция с системой контроля версий
 - Строгий семантический анализ, тестируемость, рефакторинг и отладка
 - Управление развёртыванием и настройка CI/CD
 - IDE является основным и единственным инструментом разработчика
- ↓
- Высокая надёжность и эффективность разработки, отладки, развёртывания и решения инцидентов

Корпоративный инструментарий

- Глубоко интегрированный трэкер задач (аналог: **Jira**)
- HR портал (корпоративные решения)/социальная сеть (аналог: **Linkedin**)
- Мессенджер (аналог: **Slack**)
- Система уведомлений (аналог: **почта**)



- Готовые средства для корпоративных задач
- Настройка рабочих процессов перекладывается с менеджеров на ПО
- Унификация управления временем
- Эффективное управление кадрами/карьерой

Платформа стриминга и обучения

- Стreamинг видео и работы с кодом
- Коллективное редактирование кода
- Встраиваемая система обучения
- Геймификация обучения
- Встраиваемый вопросник
с гарантированно рабочими сниппетами (аналог: *Stack Overflow*)



- Эффективное обучение кадров и обмен опытом
- Легкодоступный и управляемый найм кадров

Новостная лента

- Рекомендации проектов
- Рекомендации расширений IDE
- Рекомендации обучающих курсов
- Рекомендации вакансий
- Статьи о HardCode и инфраструктуре
- Новости IT-индустрии



- Актуальная релевантная информация для разработчика
- Широкие рекламные возможности

Магазин: собственные и партнёрские решения

- Ресурсы облачной платформы *Service as a Service*
- Расширения IDE: отладчики, анализаторы, редакторы, линтеры и т. д.
- Корпоративный инструментарий: обширное множество компонент
- Обучающие курсы, обзоры, хакатоны, подписки на стримы
(близкие аналоги: YouTube, Twitch)
- Библиотеки и сервисы (аналог: GitHub)
- Клиентские приложения (аналог: Steam)
- Игровые движки (аналог: Unreal)
- Игры
- Браузерная платформа