Исходный код: скрытое знание и как его показать?

Евгений Зуев
Алексей Степанов
Университет Иннополис

The Problem

- Для больших и долгоживущих программных систем период их **эксплуатации** существенно превосходит время разработки. Эксплуатация программной системы подразумевает действия по ее изменению, улучшению, добавлению новых функций, исправлению ошибок. Все это требует активной работы с исходным текстом программы.

The Problem

- Для больших и долгоживущих программных систем период их **эксплуатации** существенно превосходит время разработки. Эксплуатация программной системы подразумевает действия по ее изменению, улучшению, добавлению новых функций, исправлению ошибок. Все это требует активной работы с исходным текстом программы.
- Во многих современных исследованиях утверждается, что на изучение и понимание программы уходит **около 70% рабочего времени** программиста, и только 5% времени занимает непосредственное написание и редактирование кода. Кроме того, отмечается, что ключевой и наиболее затратный по времени процесс разработчика в ходе создания программы — это выстраивание *мысленной модели кода*.

The Problem

- Для больших и долгоживущих программных систем период их **эксплуатации** существенно превосходит время разработки. Эксплуатация программной системы подразумевает действия по ее изменению, улучшению, добавлению новых функций, исправлению ошибок. Все это требует активной работы с исходным текстом программы.
- Во многих современных исследованиях утверждается, что на изучение и понимание программы уходит **около 70% рабочего времени** программиста, и только 5% времени занимает непосредственное написание и редактирование кода. Кроме того, отмечается, что ключевой и наиболее затратный по времени процесс разработчика в ходе создания программы — это выстраивание *мысленной модели кода*.
- Поэтому задача выявления семантики как компонентов программы, так и архитектуры системы в целом и представление (визуализация) их в форме, удобной для анализа, – является критически важной и актуальной, особенно учитывая тот факт, что в настоящее время для многих современных ЯП не существует адекватных инструментов для подобного анализа и визуализации.

Visualization: Requirements

- **Полнота**
  Должны быть представлены все аспекты программы, имеющие значение для ее понимания и анализа
- **Отчуждаемость**
  Визуальное представление программы не должно зависеть от среды разработки, в которой она создавалась, и не должно ею ограничиваться
- **Мобильность**
  Формат визуального представления не должен зависеть от конкретной операционной платформы
- **Интерактивность**
  Представление должно обеспечивать взаимодействие с читателем программы, варьируя степень детализации информации
- **Масштабируемость**
  Необходимо обеспечить отображение всех аспектов: как информации об отдельных сущностях, так и об архитектуре программы в целом.

Visualization: Two Visions

N.Wirth, J.Gutknecht

Programming-in-the small

Programming-in-the-large

Visualization: Two Visions

N.Wirth, J.Gutknecht
Programming-in-the small
Programming-in-the-large

Visualization-in-the small
Visualization-in-the-large

Visualization: Which Program Aspects to Represent 1

- **Общая структура текста программы**
  Форматирование, синтаксическая расцветка, схлопывание-развертывание структурных компонентов, навигация по тексту программы
- **Отношения сущностей «объявление-использование»**
  Где объявлена сущность и где и как она используется
- **Семантическая расцветка**
  Визуальное отображение различных категорий сущностей: переменные, функции, типы/классы
- **Информация об атрибутах сущности**
  Типы и спецификации переменных, структура составных типов, характеристики функций/методов
- **Скрытая семантика конструкций**
  Отображение смысла программной конструкции, не выраженное явно в ее текстовом представлении.

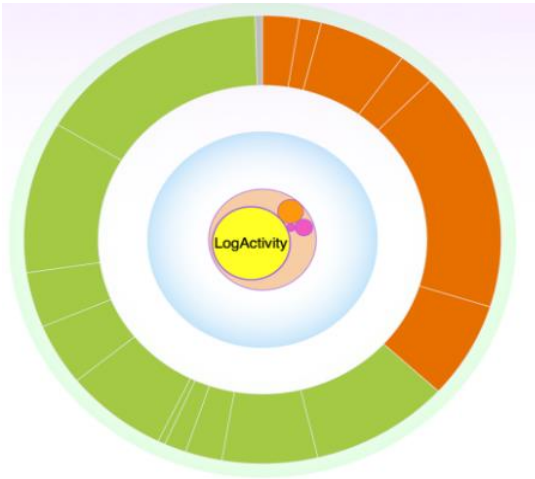
Visualization-in-the-small

Visualization: Which Program Aspects to Represent 2

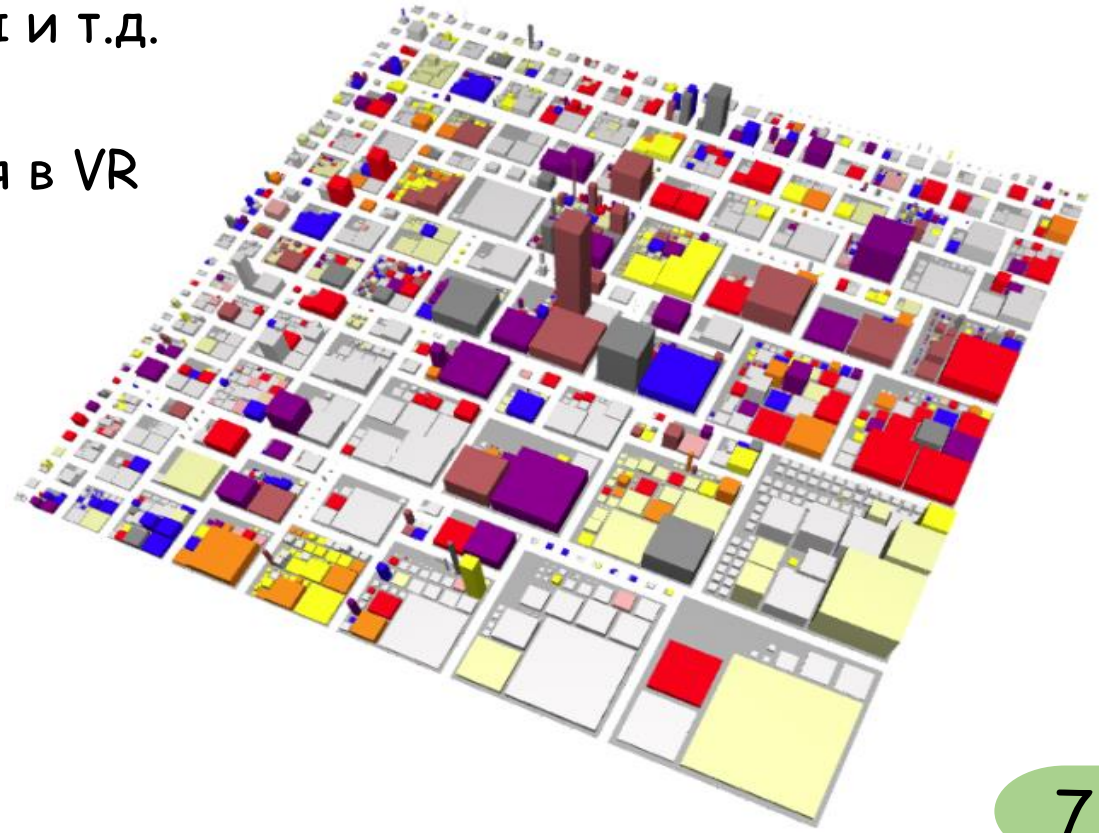
- **Информация о межмодульных и межкомпонентных связях**
  Характер зависимостей: динамическое/статическое импортирование, текстовое включение; отображение сопроцессов и concurrency
- **Информация об иерархии наследования (для ООП/классов)**
  Общая схема и характер наследования: единичное/множественное, обычное/виртуальное
- **Информация о природе программных компонентов**
  Класс, пакет, модуль, пространство имен, единица компиляции, …
- **Информация об общих характеристиках программных компонентов**
  Размер, структурные характеристики, метрики внутренней сложности, характер использования (интерфейс)

Visualization-in-the-large

Visualization: Existing Approaches

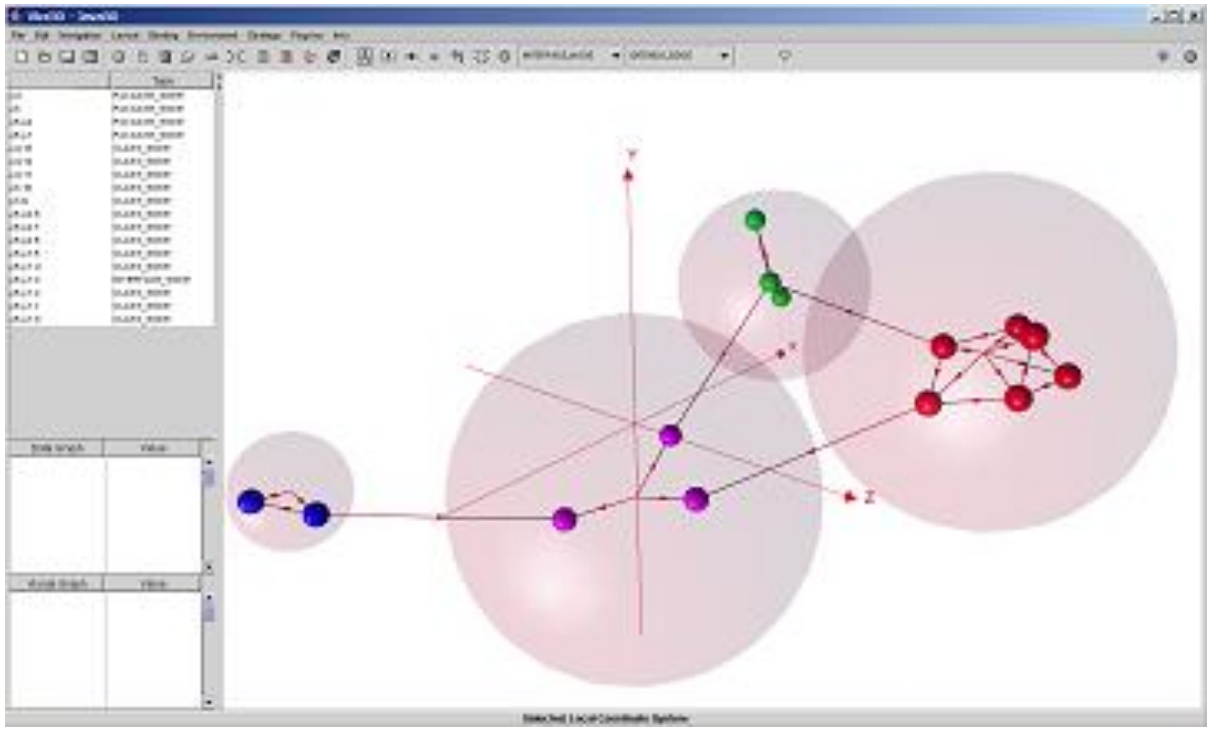


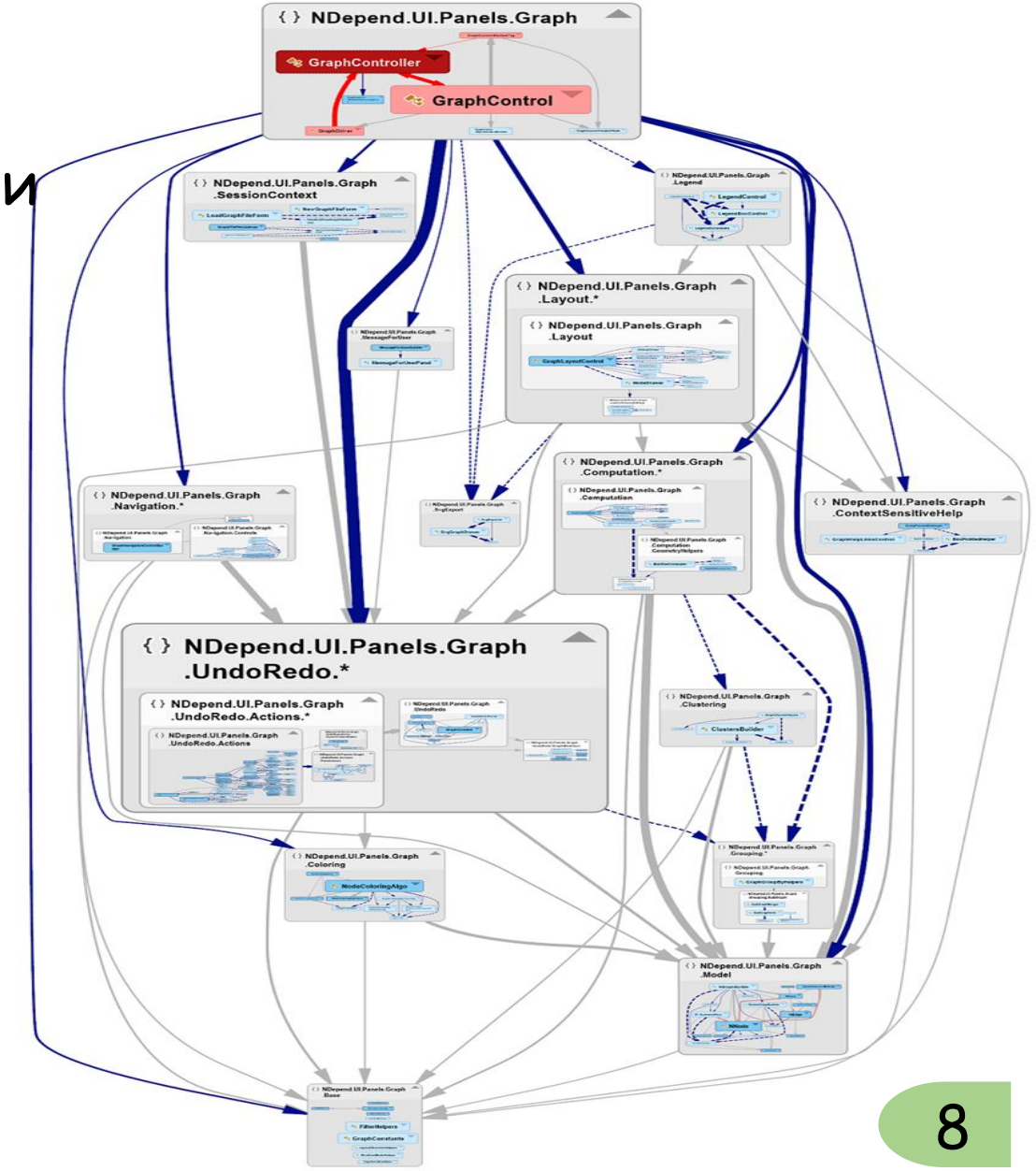
- **Samoa**
  Графическое представление мобильных приложений.
  Относительные показатели размера различных модулей
  и внешних вызовов.
- **Code City**
  Использует "City Metaphor" для представления структуры программы: классы - здания, модули - районы и т.д.
- **EvoStreets**
  Расширение идеи Code City для использования в VR



Visualization: Existing Approaches

- **NDepend**
  Программная система для визуализации кода C# с множеством представлений.
- **Vizz3D**
  Представление структуры в виде трехмерного графа





Visualization: How to Implement

Откуда брать информацию о программе?

- Документация
- Объектный код
- Байт-код/метаданные
- **Исходный код**

==> Единственный источник информации – **исходный код программы**.

Во всех остальных источниках ключевая информация потеряна

Visualization: How to Implement

Откуда брать информацию о программе?

- Документация
- Объектный код
- Байт-код/метаданные
- **Исходный код**

==> Единственный источник информации – **исходный код программы**.

Во всех остальных источниках ключевая информация потеряна

Как получить информацию?

- Анализировать непосредственно текст программы
- Воспользоваться **результатом анализа, производимого компиляторами**

Visualization: How to Implement

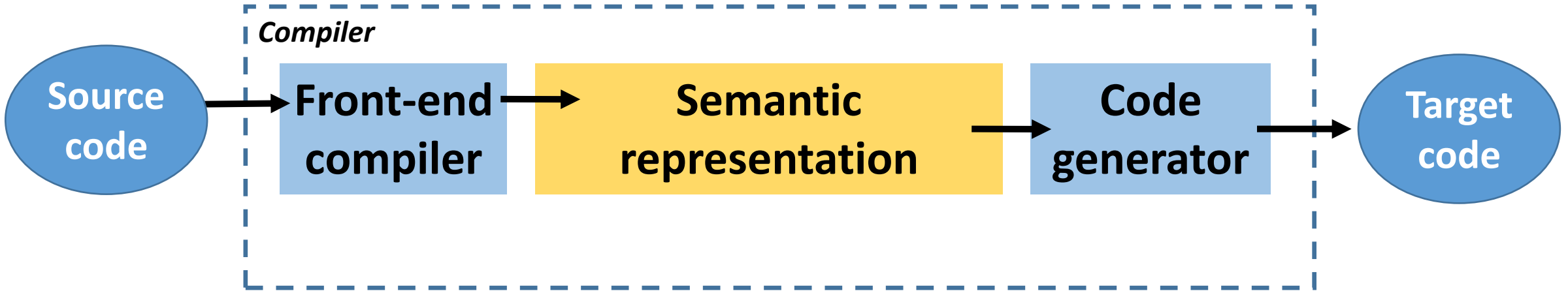
В каком виде отображать информацию?

- Общий ответ:
  В любом формате, который допускает реализацию требований полноты, отчуждаемости, мобильности, интерактивности, масштабируемости, см. слайд 3.

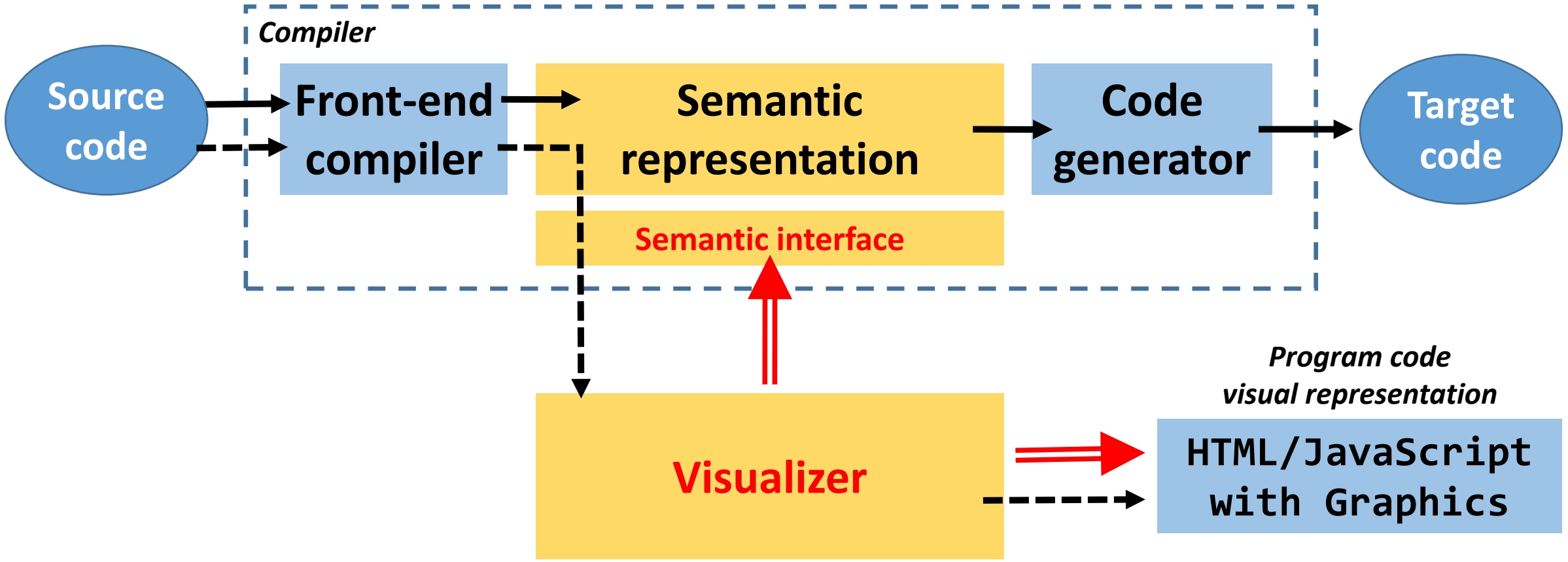
- Потенциальные кандидаты:
  HTML/CSS/JavaScript
  PDF
  RTF
  LaTex
  …

HTML
- Доступен на любой платформе
- Стандартный формат
- Поддержка навигации и других необходимых функций
- Поддержка графики
- Интерактивность

Visualization: The Implementation Scheme

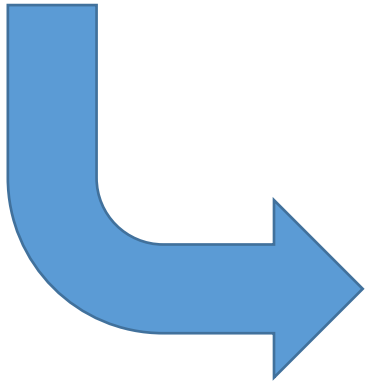


Visualization: The Implementation Scheme



Hidden Semantics: C++ Example 1

```
class C {
  public:
    int a, b;
};
```



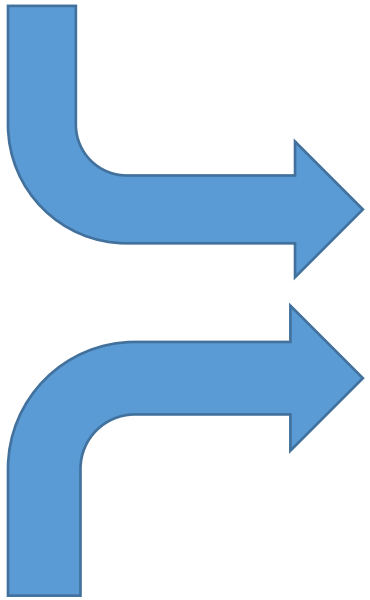
```
class C {
  public:
    C() { a = 0; b = 0; }
    C(const C& c) { a = c.a; b = c.b; }
     ...
    int a, b;
};
```

Automatically generated:

- Default constructor
- Copy constructor
- Move constructor
- Copy assignment operator
- Destructor
- …

Hidden Semantics: C++ Example 2

Constructor call



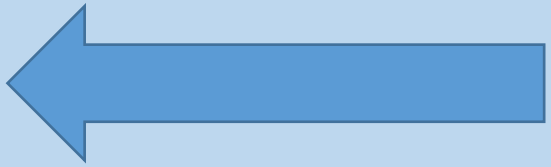
```
class C {
    ...
};
void f(int s)
{
    C c(s);  // Local object
    ...;
}
```

```
c.C::~C();
```

Destructor call

Hidden Semantics: C++ Example 3

```
class C {
    int m;
public:
    operator bool() { return m>0; }
    ...
};
...
void f(C& c)
{
    if ( c ) {
        ...
    }
}
```



```
if ( c.operator bool() )
```

Hidden Semantics: C++ Example 4

```
class C { ... };

void f()
{
    C c1;               // Default ctor
    C c2 = C(1);        // Two constructors
    C c3 = c1 + c2;     // Operator func + ctor

    int x = ...;
    double y = ...;
    int z = x + y;      // Two std conversions
}
```

The First Approach: The Go Language

snake_p2p

- cmd
- core
  - errors.go
  - game_events.go
- engine
  - console
    - game.go
    - gui.go
- protocol
- README.md
- discovery.go
- node.go

engine\console\game.go

```
17          "github.com/rs/zerolog/log"
18          //"github.com/sanit
19  )
20
21  type Snake struct {
22          Alive bool
23          Body  []core.Coord
24          Head  core.Coord
25          Style tcell.Style
26  }
27
28  type GameUI struct {
29          gi          *game.GameInstance
30          Snakes      map[peer.ID]*Snake
31          Food        map[int]core.Coord
32          bound       Boundary
33          moveNum     int
34          AliveSnakes int
35          foodLastID  int
36          Over        bool
37          Successful  bool
38          WinnerID    peer.ID
39          r           *rand.Rand
40  }
41
42  // add food every N moves
43  const N = 5
44
45  func NewGame(gi *game.GameInstance) *GameUI {
46          return &GameUI{
47                  gi:         gi,
48                  Food:       make(map[int]core.Coord),
49                  Snakes:     make(map[peer.ID]*Snake),
```

```
struct {
        done chan struct{}
        finishedCh c
        streams map[
        selfID ID
        Seed int64
        recv chan in
        moves chan playerMove
        mu Mutex
}
```

```
struct {
        ID ID
        Dir Direction
}
```

Visualization-in-the-small

The First Approach: The Go Language

snake_p2p

- cmd
- core
  - errors.go
  - game_events.go
- engine
  - console
    - game.go
    - gui.go
- protocol
- README.md
- discovery.go
- node.go

engine\console\gui.go

```
11          "os"
12          "strconv"
13          "time"
14  )
15
16  type GatherUI struct {
17          h *snake.Node
18          app *tview.Application
19          flex *tview.Flex
20          myGatherPoint *tview.TextView
21          gameList *tview.Table
22          createBtn *tview.Button
23          newGame *tview.InputField
24          maxPlayers int
25          gatherPoints map[string]*gather.Gather
26  }
27
28  func addRow(table *tview.Table, msg *gather.GatherPointMessage, row int, color tcell.Color) {
29          ID := msg.ConnectTo.ID.Pretty()
30          tableCell := tview.NewTableCell(ID).
31                  SetTextColor(color).
32                  SetAlign(tview.AlignCenter).
33                  SetExpansion(1)
34          table.SetCell(row, 0, tableCell)
```

```
struct {
        ConnectTo AddrInfo
        TTL Duration
        DesiredPlayerCount uint
        CurrentPlayerCount uint
}
```

```
struct {
        ID ID
        Addrs []Multiaddr
}
```

Visualization-in-the-small

Declaration of GatherPointMessage

protocol\gather\messages.go

```
18  type GatherPointMessage struct {
```

Usages of GatherPointMessage

engine\console\gui.go

```
25  gatherPoints map[string]*gather.GatherPointMessage
28  func addRow(table *tview.Table, msg *gather.GatherPointMessage, row int, color tcell.Color) {
52  g.gatherPoints = make(map[string]*gather.GatherPointMessage)
```

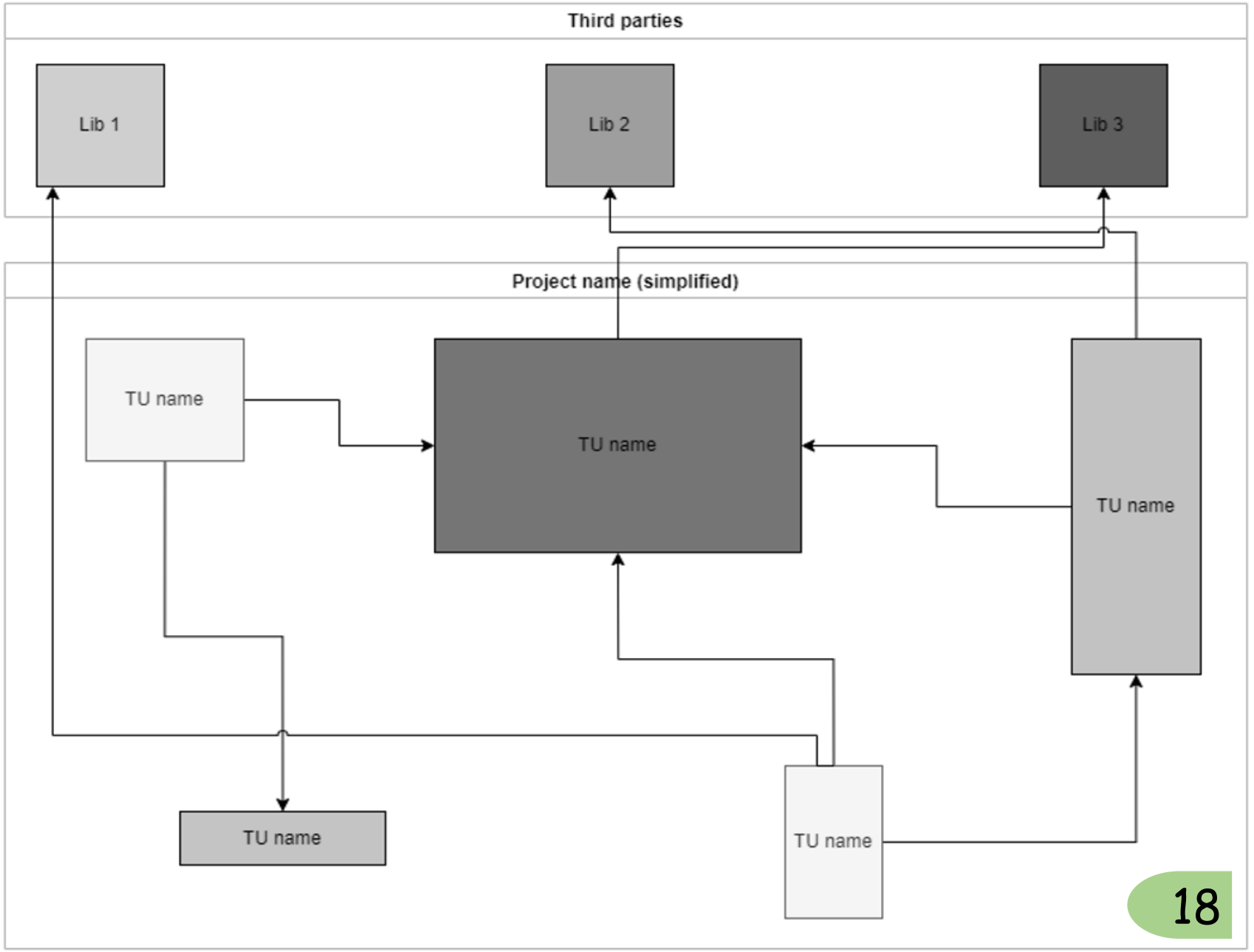
cmd\gather_test\main.go

The Second Approach: The C++ Language

Представление системы

Внешние зависимости проекта: части кода, не относящиеся к кодовой базе проекта. **Цвет** зависит от частоты их использования в проекте.

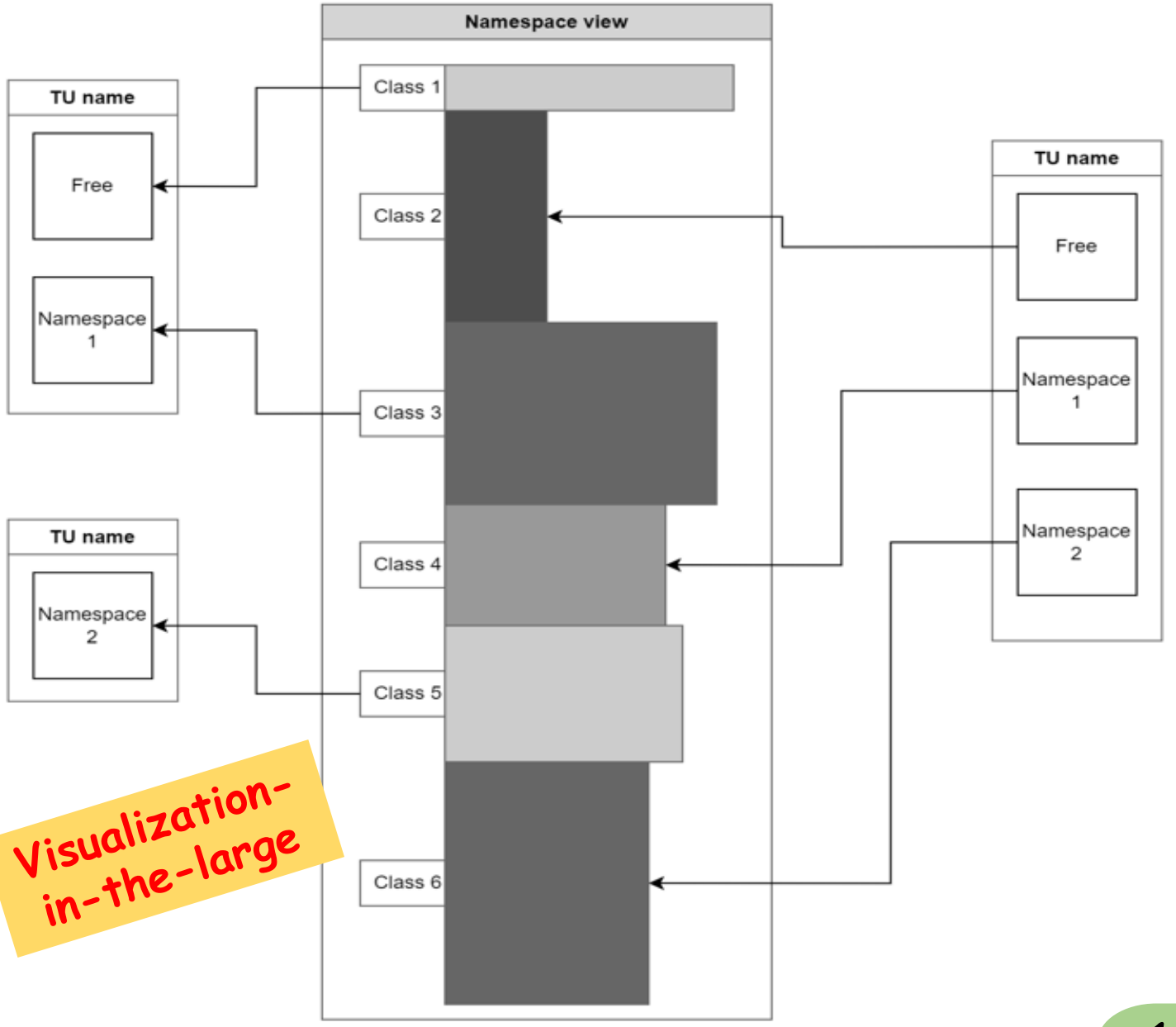
Высота: общее число сущностей в единице трансляции
Ширина: количество строк кода
Цвет: использование сущностей из этой единицы трансляции в других.
Стрелки: отношение использования. ET, из которой выходит стрелка, использует внутри себя сущности ET, в которую стрелка входит.



The Second Approach: The C++ Language

Представление **пространства имен** вместе с зависимостями:

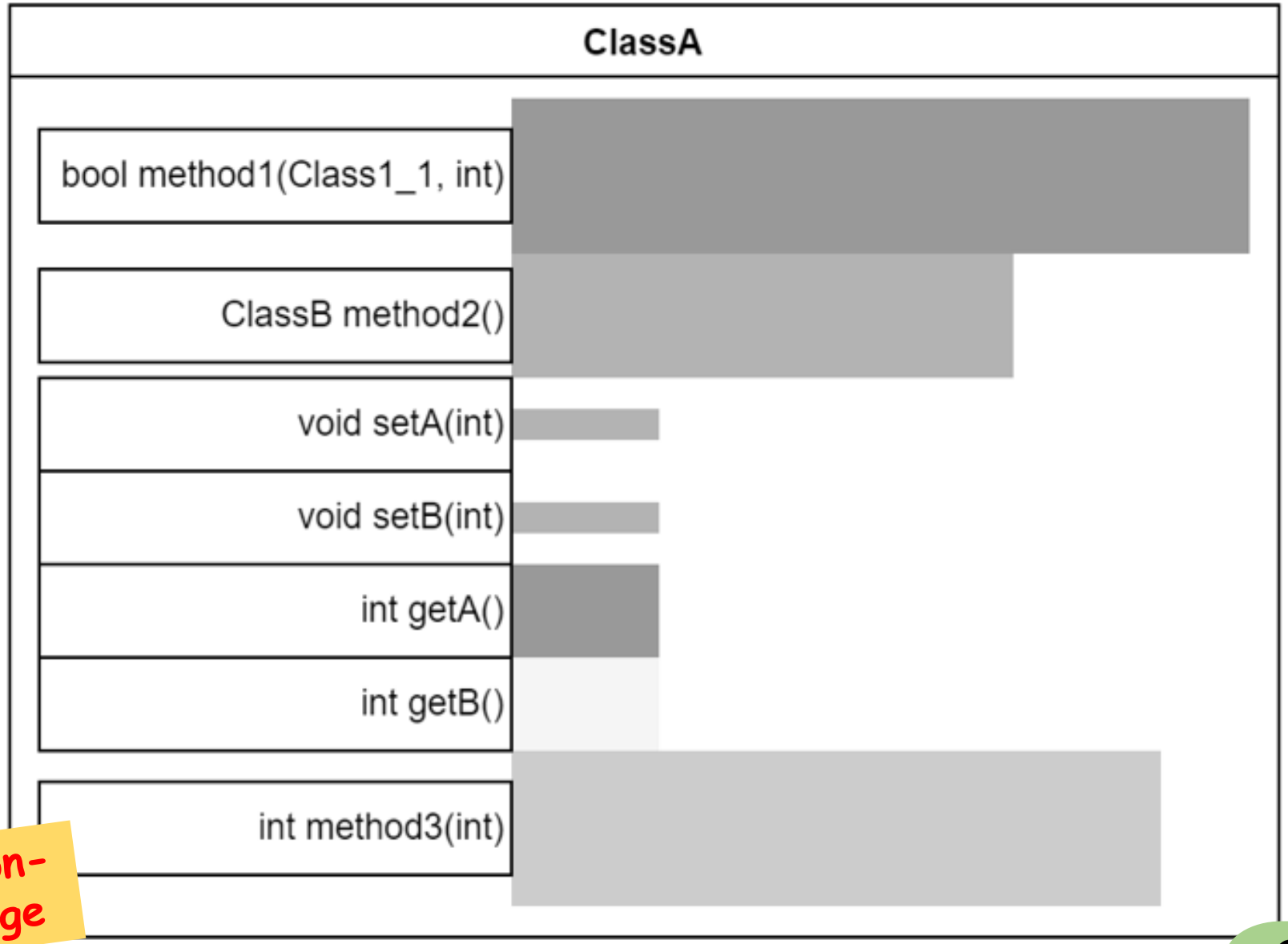
- **Ширина** - Количество строк кода
- **Высота** - Количество методов и переменных внутри класса
- **Цвет** - Частота использования вне пространства имен



The Second Approach: The C++ Language

Представление класса

- **Ширина** - количество строк кода
- **Высота** - количество используемых внутри функции обращений (вызовы функции и доступ к переменным)
- **Цвет** - частота использования

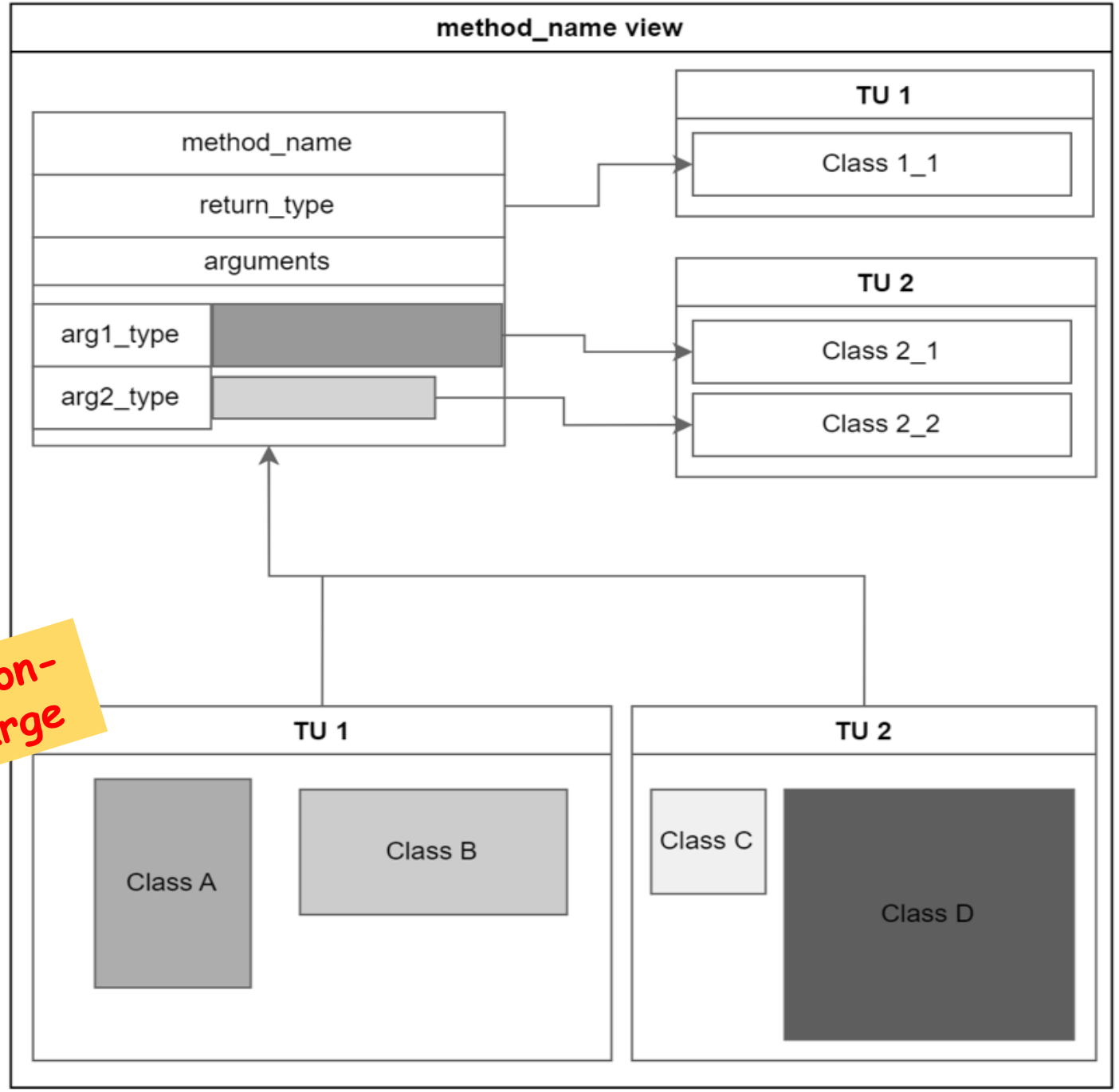


Visualization-in-the-large

The Second Approach: The C++ Language

Представление метода

- Стрелки к правой части указывают на места определений соответствующих типов.
- В нижней части находятся сущности, которые используют данный метод



Visualization-in-the-large

Current & Future Research: Four Dimensions

- (Вширь)

  Как наиболее адекватно, наглядно и единообразно представить типичные свойства и конструкции ЯП.

  Разрабатываем прототипные реализации visualization-in-the-small для различных ЯП.

- (Вглубь)

  Как наглядно представить архитектуру программы

  Для некоторых языков (C++) проводится более глубокий анализ форм и способов визуального представления in-the-large.

- (Вбок)

  Как унифицировать принципы генерации визуального представления, сделав единый интерфейс генерации

- (Вверх)

  Интеграция двух форм представления – «small» и «large»

The Whole Project Configuration

Language	Visualization		Status	Contributors
	In-the-small	In-the-large		
Go	+		Completed	Артем Баханов Михаил Кусков Альфия Муссабекова
Go		Co-programming	In progress	Владимир Гордеев
Swift		Program structure	In progress	Игорь Белов Роман Набиуллин
Dart	+		In progress	Тимур Нугаев
Rust	+		In progress	Лев Лимаренко
C	+		In progress	Антон Доспехов
C++	+	Program architecture	In progress	Алексей Степанов
C#	+		In progress	Владимир Калабухов

Thank you!