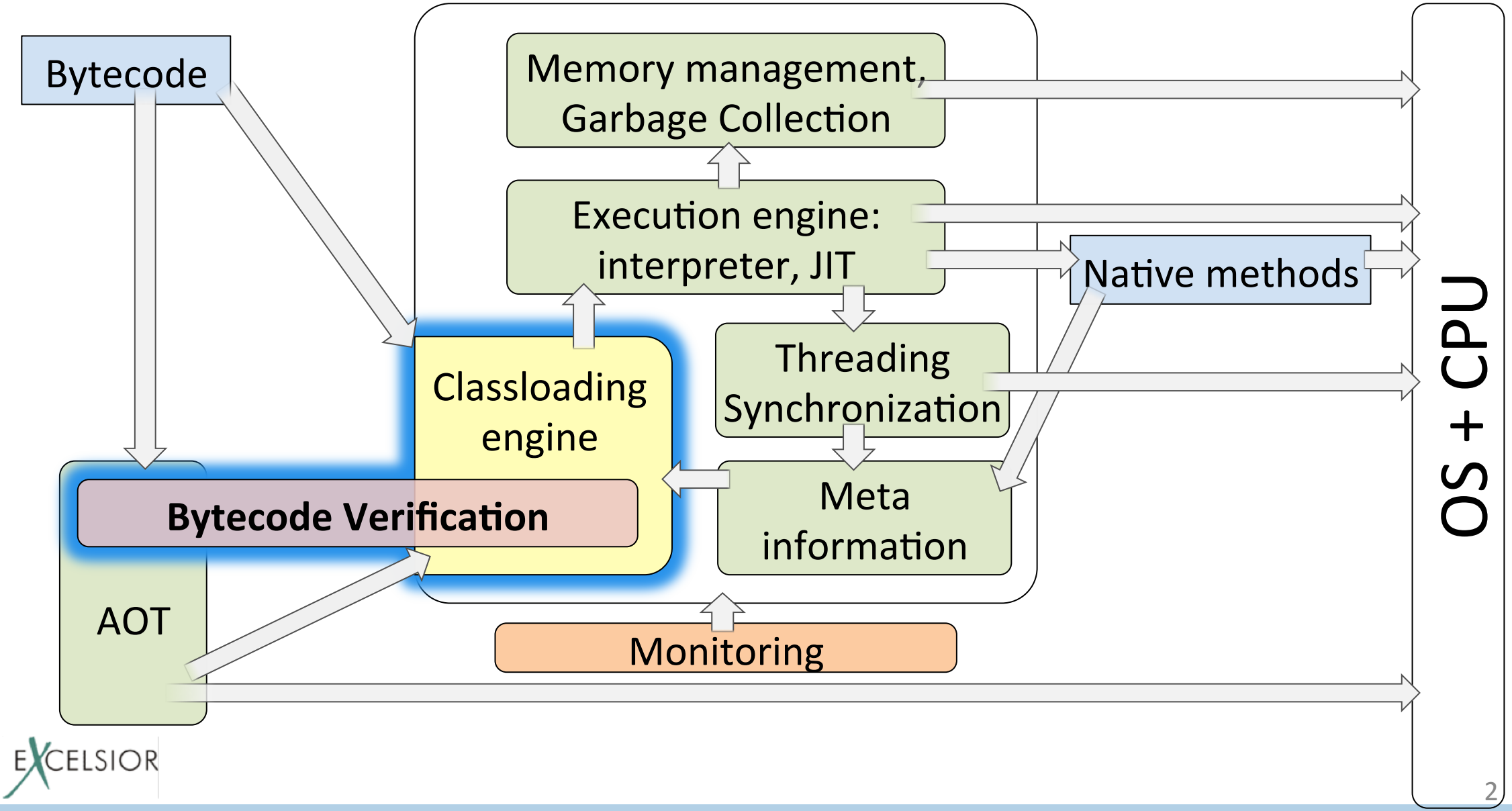
Верификация Java байткода: как, когда, а может отключить?

Никита Липский

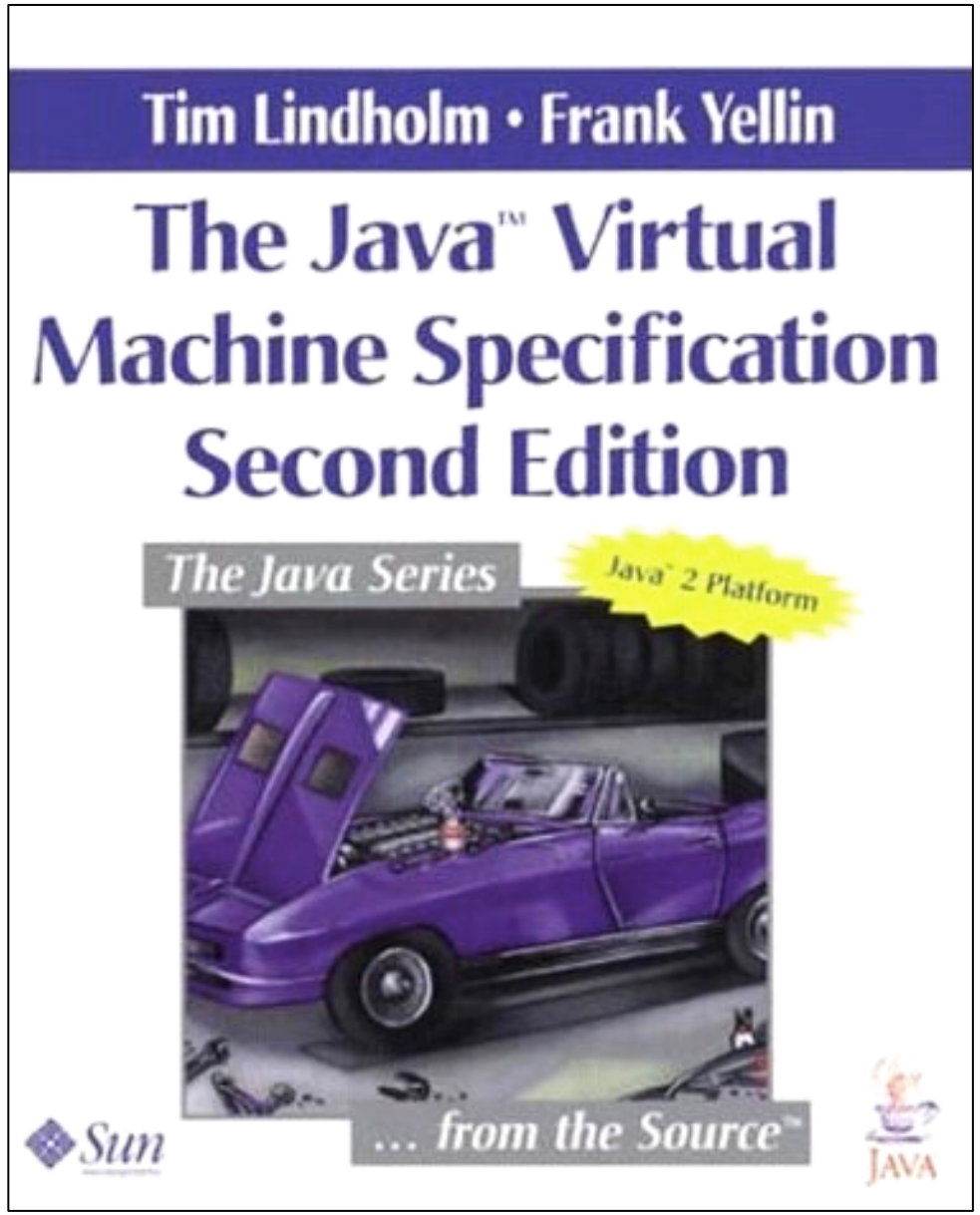
Excelsior LLC

EXCELSIOR

Анатомия JVM



EXCELSIOR


Tim Lindholm • Frank Yellin
The Java™ Virtual Machine Specification Second Edition
The Java Series
Java™ 2 Platform
... from the Source™
Sun
JAVA

про себя

- Инициатор проекта Excelsior JET
  - работал над проектом более 18 лет
  - как идейный вдохновитель
  - компиляторный инженер
  - руководитель
  - и много в каких еще ролях
- twitter: @pjBooms
- team blog: https://www.excelsiorjet.com/blog

EXCELSIOR

Из доклада вы узнаете

1. Как и когда работает верификатор
2. Может ли верификатор байткода замедлить исполнение программы и ее старт
3. Что бы было, если бы не было верификатора
4. Почему опасно его отключать
5. Как избежать VerifyError при порождении байткода

EXCELSIOR

Java class file & bytecode

EXCELSIOR

Java class file

Класс файл

Версия
Constant Pool
Имя класса, модификаторы
Суперкласс, суперинтрфейсы
Поля
Методы
Атрибуты

Пример: Constant Pool

CONSTANT_Integer: **100500**
CONSTANT_Float: **3.1415**
CONSTANT_String : **Hello World!**
CONSTANT_Class: **java.lang.String**
CONSTANT_FieldRef: **A.field@int**
CONSTANT_MethodRef: **B.method(IJ)F**
CONSTANT_InterfaceMethodRef: **I.foo()V**

EXCELSIOR

Java class file

- Поля, методы тоже имеют атрибуты (например, значения константных полей)

- Главный атрибут метода – это его код: Java байт-код

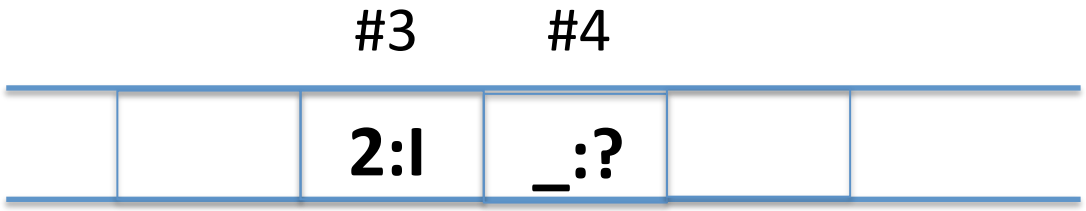
Java bytecode

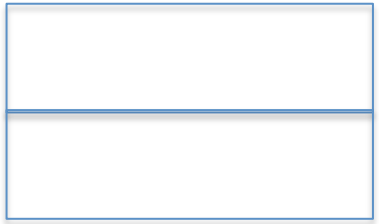
- Массив инструкций
- Стэк операндов инструкций метода
- Массив локальных переменных (аргументы метода, локальные переменные)

Java bytecode

Инструкция берет свои операнды со стэка и кладет результат на стэк. Пример:

```
0: iload 3
2: bipush 5
4: iadd
5: istore 4
7: …
```

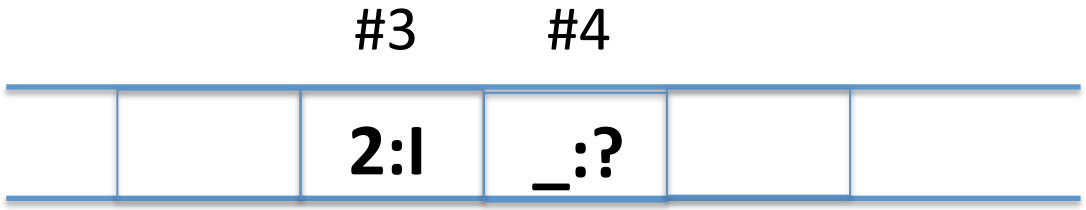


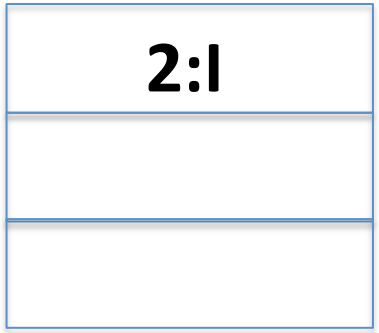


Java bytecode

Инструкция берет свои операнды со стэка и кладет результат на стэк. Пример:

```
0: iload 3
2: bipush 5
4: iadd
5: istore 4
7: …
```

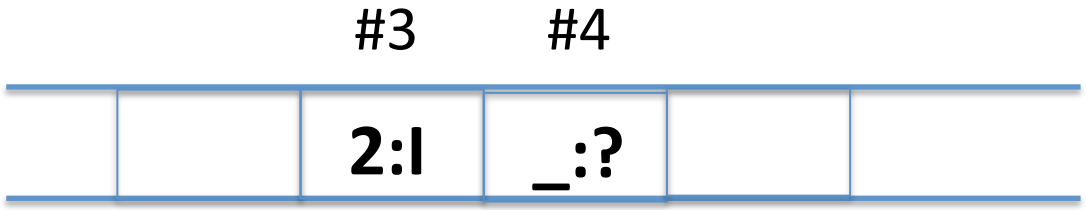


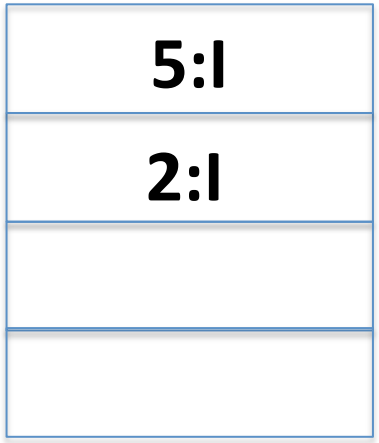


Java bytecode

Инструкция берет свои операнды со стэка и кладет результат на стэк. Пример:

```
0: iload 3
2: bipush 5
4: iadd
5: istore 4
7: …
```

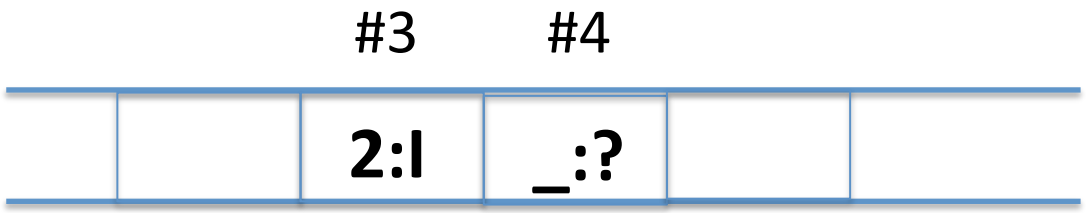


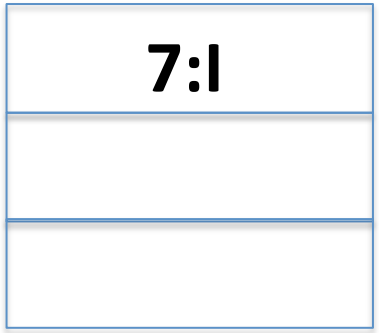


Java bytecode

Инструкция берет свои операнды со стэка и кладет результат на стэк. Пример:

```
0: iload 3
2: bipush 5
4: iadd
5: istore 4
7: …
```

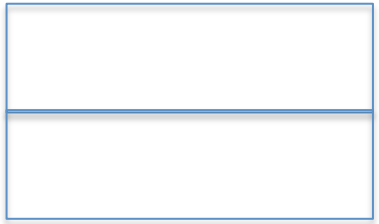




Java bytecode

Инструкция берет свои операнды со стэка и кладет результат на стэк. Пример:


```
0: iload 3
2: bipush 5
4: iadd
5: istore 4
7: …
```



```
0: bipush 2
2: bipush 2
4: iadd
5: goto 0
```

Что произойдет при исполнении этого байт-кода?

A: Программа зациклится

B: VerifyError

C: StackOverflowError

D: Не будет исполняться

Загрузка класс-файла.
Когда работает верификатор?

EXCELSIOR

Стадии загрузки класса в JVM

- Загрузка (loading)
- Линковка (linking)
- Инициализация (initializing)

Глава “5. Loading, Linking, and Initializing” спецификации JVM

EXCELSIOR

Процесс загрузки класса (создание класса)

- Читается class file
  - Проверяется корректность формата (может выбросить ClassFormatError)
- Создается ран-тайм представление класса в выделенной области памяти
  - runtime constant pool in Method Area aka **Meta Space** aka Permanent Generation
- Грузятся суперкласс, суперинтерфейсы

EXCELSIOR

Линковка

- Верификация байт-кода
- Подготовка
- Разрешение символьных ссылок

EXCELSIOR

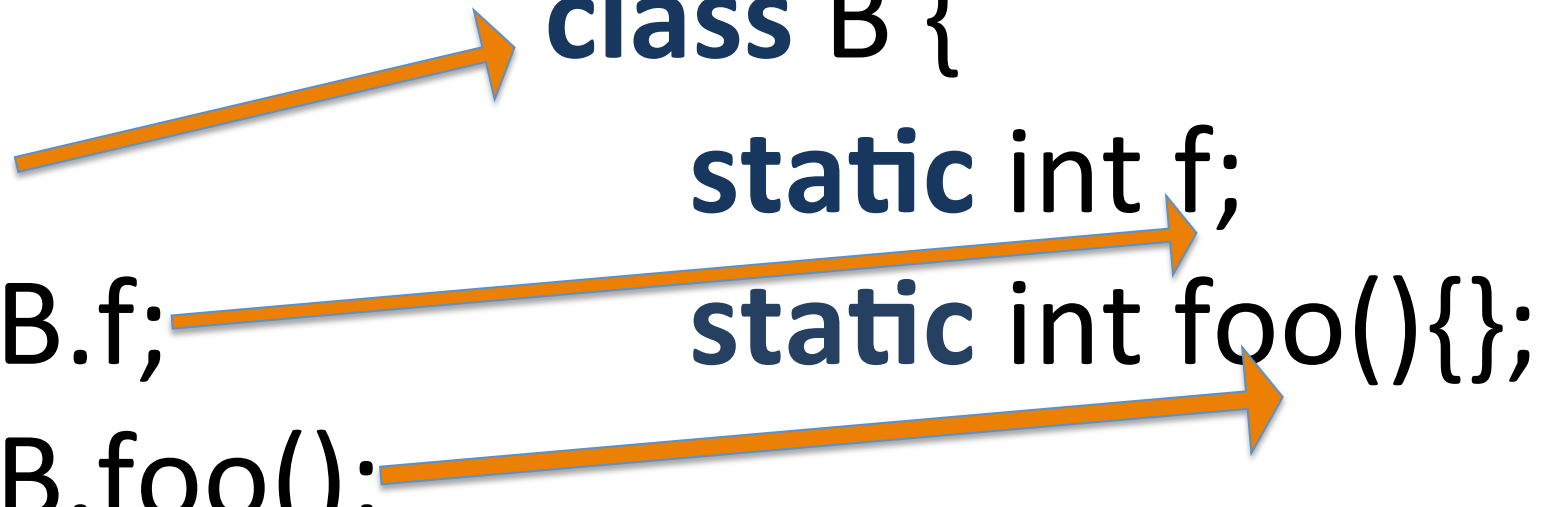
Разрешение символьных ссылок

A.java:

```
class A {
    B useB;
    int f1 = B.f;
    int f2 = B.foo();
}
```

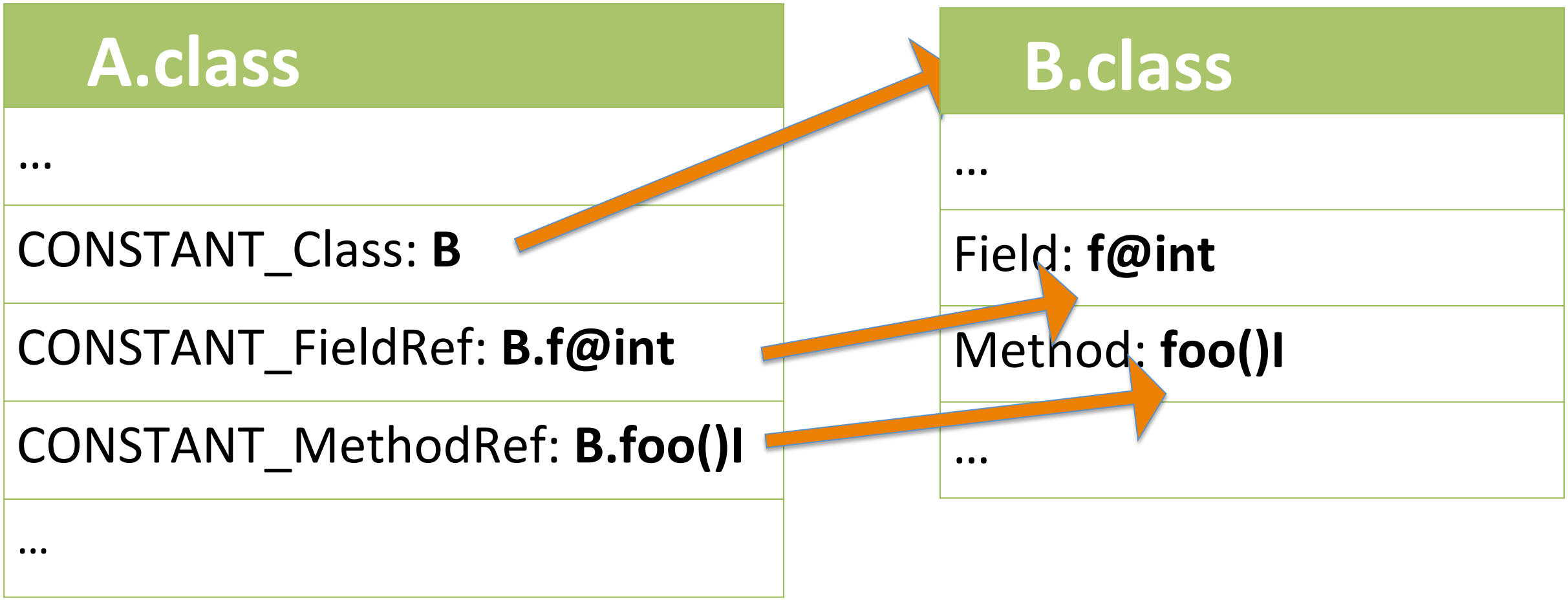
B.java:

```
class B {
    static int f;
    static int foo(){};
}
```



EXCELSIOR

Разрешение символьных ссылок



EXCELSIOR

Разрешение символьных ссылок

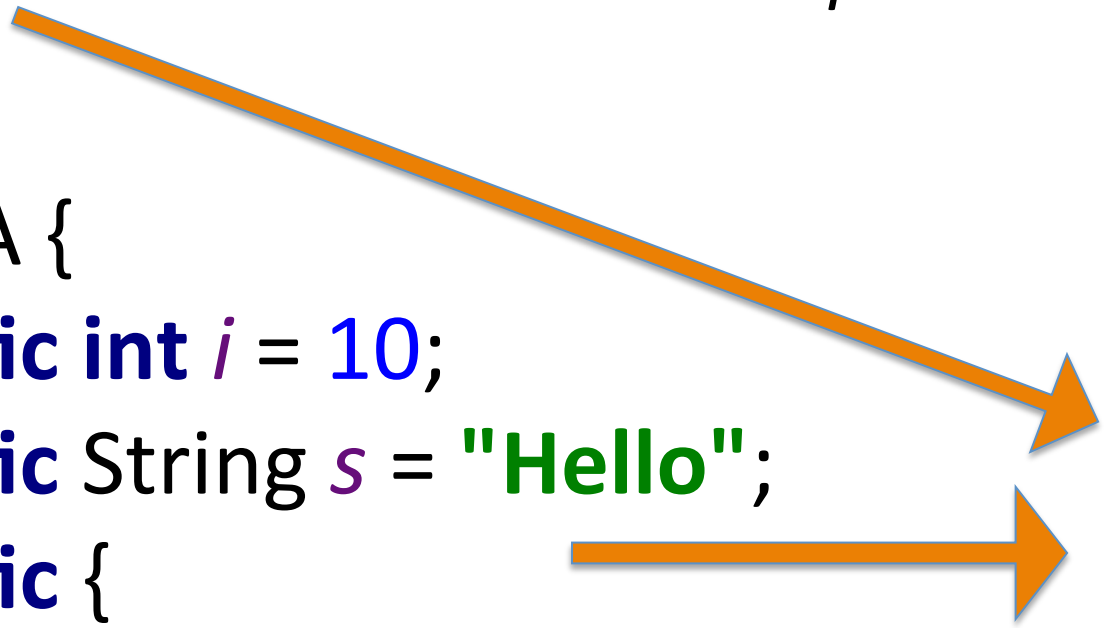
Класс может иметь ссылки на другие классы и поля, методы других классов

- Ленивое разрешение
  - ссылки разрешаются при первом доступе
- Энергичное разрешение
  - разрешаются все ссылки какие возможно

EXCELSIOR

Инициализация класса

Вызов статического *инициализатора* класса



```
class A {
    static int i = 10;
    static String s = "Hello";
    static {
        System.out.println(s);
    }
}
```

A.class
…
<clinit>: *i* = 10; *s* = "Hello"; System.***out***.println(*s*);
…

Порядок стадий загрузки класса*

- Класс должен быть полностью загружен прежде, чем слинкован

* Глава 5.4 Спецификации JVM

EXCELSIOR

Порядок стадий загрузки класса*

- Класс должен быть полностью загружен прежде, чем слинкован
- Класс должен быть полностью проверифицирован и подготовлен, прежде чем проинициализирован

* Глава 5.4 Спецификации JVM

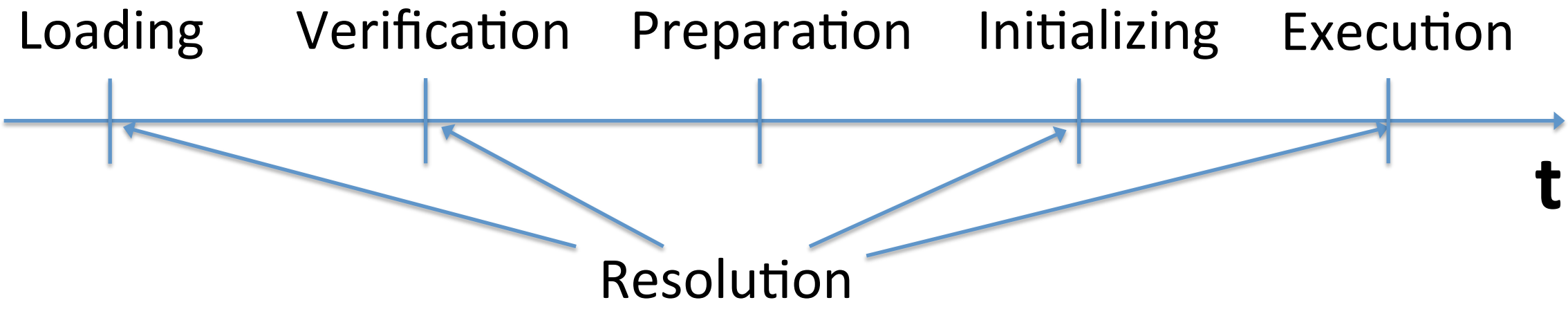
EXCELSIOR

Порядок стадий загрузки класса*

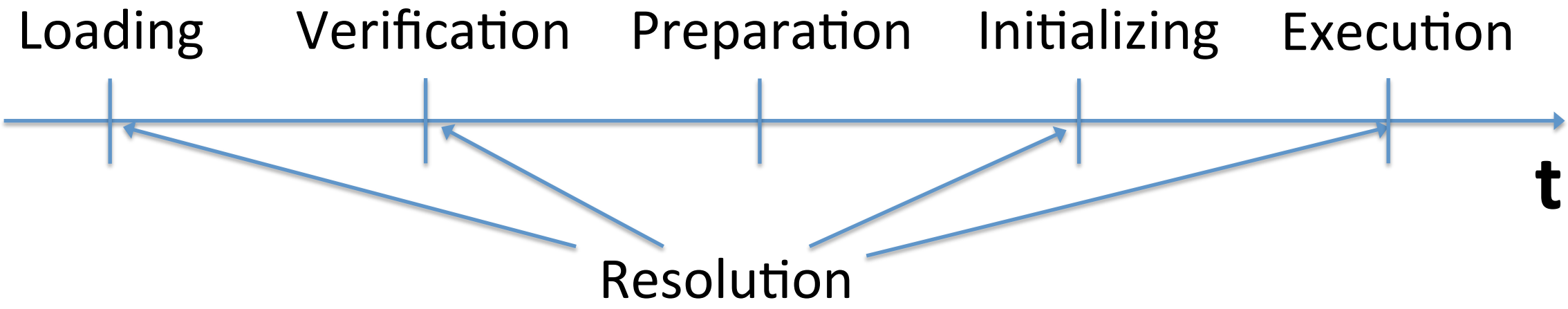
- Класс должен быть полностью загружен прежде, чем слинкован
- Класс должен быть полностью проверифицирован и подготовлен, прежде чем проинициализирован
- Ошибки разрешения ссылок, должны происходить, когда ссылка реально требуется

* Глава 5.4 Спецификации JVM

Порядок стадий загрузки класса



Порядок стадий загрузки класса



Вывод: *верификация класса происходит до того как какой-либо байткод какого-либо метода этого класса исполнится*

```
0: bipush 2
2: bipush 2
4: iadd
5: goto 0
```

Что произойдет при исполнении этого байт-кода?

A: Программа зациклится

B: VerifyError

C: StackOverflowError

D: Не будет исполняться

Верификация Java байткода

EXCELSIOR

Верификация Java байткода

Верификация Java байткода:

- проверка на статические ограничения (static constraints)
- проверка на структурные ограничения (structural constraints)

Статические ограничения

- байткод не пуст и короче 65536 байт
- все инструкции идут одна за другой (нет “дыр”)
- все инструкции имеют известную JVM мнемонику (opcode)
- переходы внутри байткода не выходят за его пределы и указывают на начало инструкции (не в середину)
- локальные переменные загружаются из корректных слотов регистра локальных переменных (не выходят за его пределы)
- И т.д.

EXCELSIOR

Статические ограничения

Проверяются за два прохода:

- Строится массив инструкций, каждая инструкция имеет адрес (смещение внутри байткода)
- Проверяется, что все переходы корректны

На выходе получается управляющий граф (CFG) байткода метода

EXCELSIOR

Структурные ограничения

- Глубина стека в каждой инструкции должна быть определенной величины для любого пути исполнения

EXCELSIOR

Структурные ограничения

- Глубина стека в каждой инструкции должна быть определенной величины для любого пути исполнения
- При исполнении любой инструкции не должна нарушится типовая совместимость по присваиванию (assign compatibility).

Структурные ограничения

- Глубина стека в каждой инструкции должна быть определенной величины для любого пути исполнения
- При исполнении любой инструкции не должна нарушится типовая совместимость по присваиванию (assign compatibility).
  - Пример: для инструкции putstatic C.f на вершине стэка должно лежать значение, тип которого совместим по присваиванию с типом поля C.f (для любого пути исполнения данной инструкции)

EXCELSIOR

Структурные ограничения

Структурные ограничения оперируют термином “*для любого пути исполнения*”. Однако верификация байткода должна происходить **до исполнения** байткода.

Вопрос: Как это возможно?

Структурные ограничения

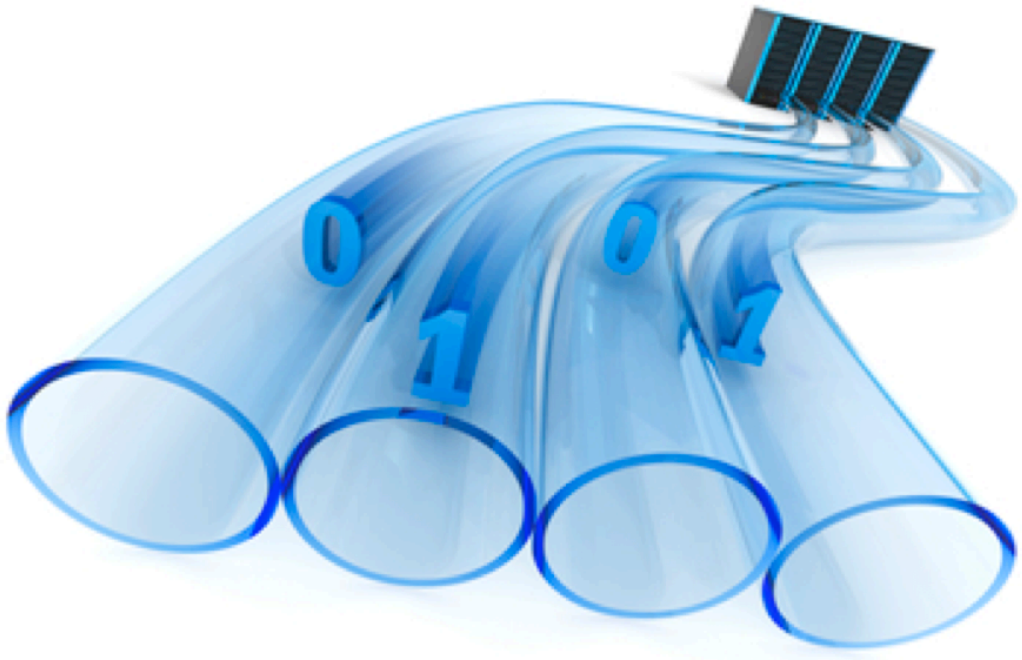
Структурные ограничения оперируют термином “*для любого пути исполнения*”. Однако верификация байткода должна происходить **до исполнения** байткода.

Вопрос: Как это возможно?

Ответ: С помощью статического потокового анализа

EXCELSIOR

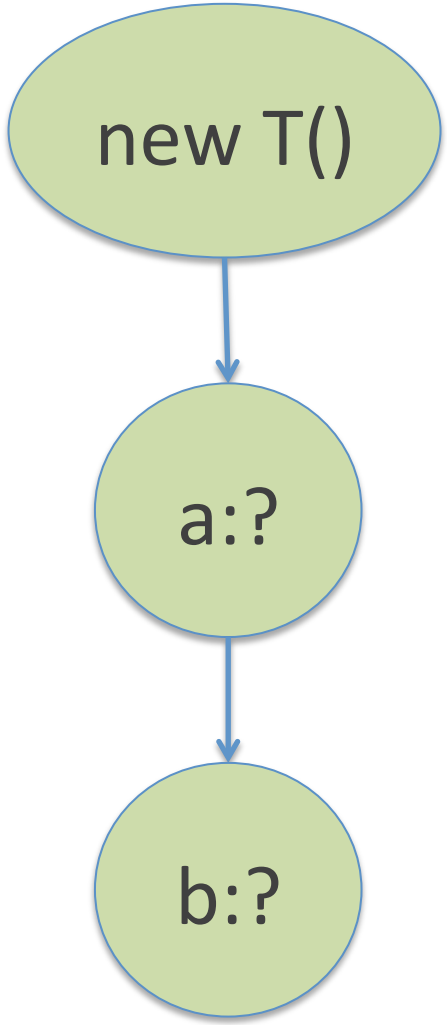
Статический потоковый анализ



EXCELSIOR

Пример: вывод типов

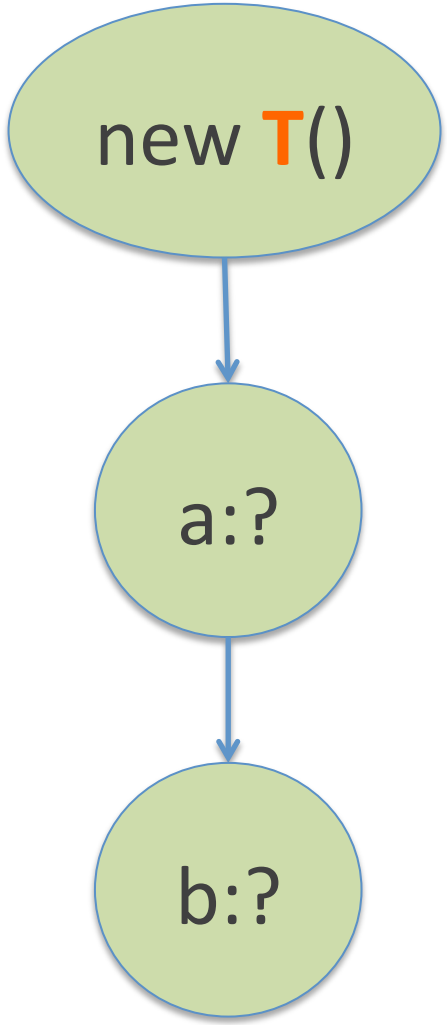
```
var a = new T()
var b = a
```



EXCELSIOR

Пример: вывод типов

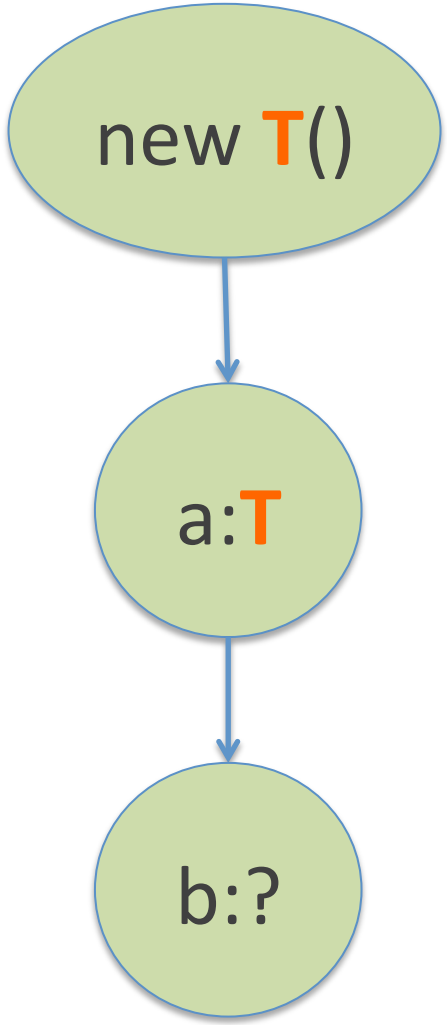
```
var a = new T()
var b = a
```



EXCELSIOR

Пример: вывод типов

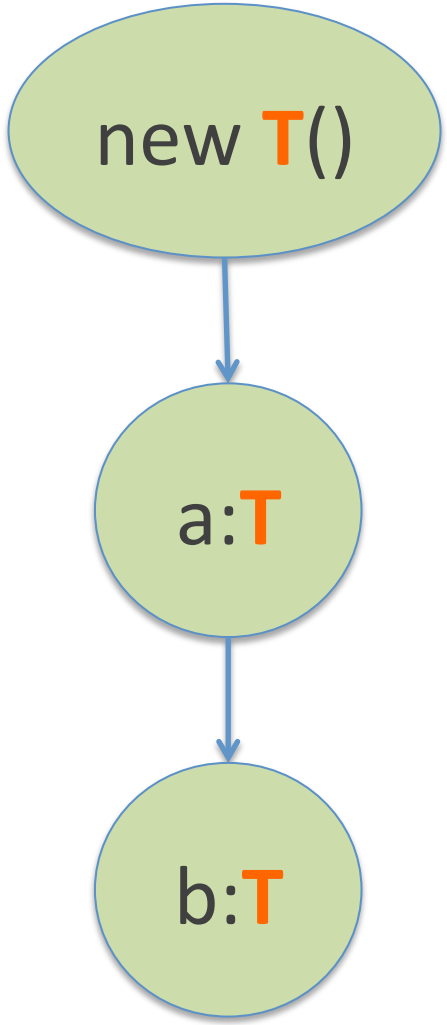
```
var a = new T()
var b = a
```



EXCELSIOR

Пример: вывод типов

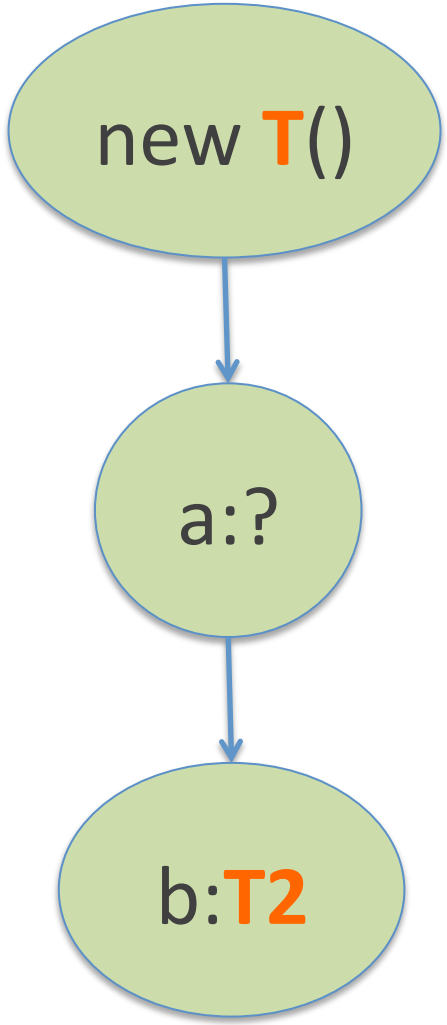
```
var a = new T()
var b = a
```



EXCELSIOR

Пример: проверка типов

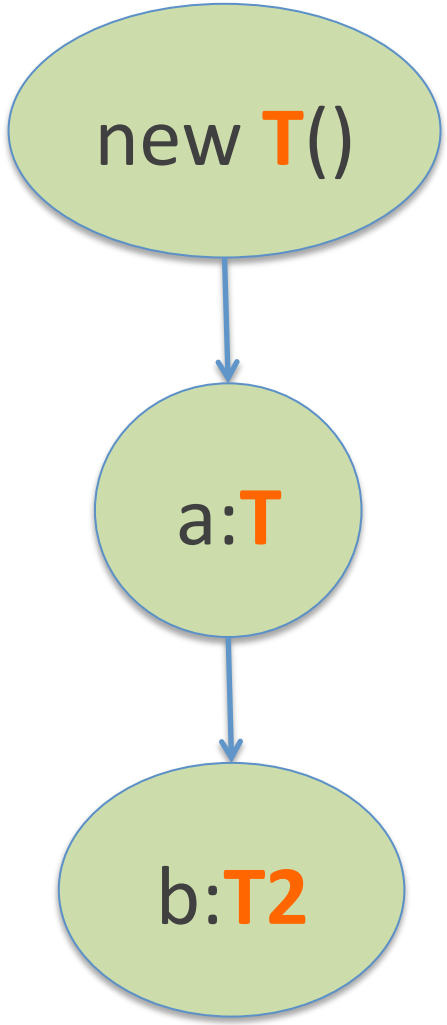
```
var a = new T()
T2 b = a
```



EXCELSIOR

Пример: проверка типов

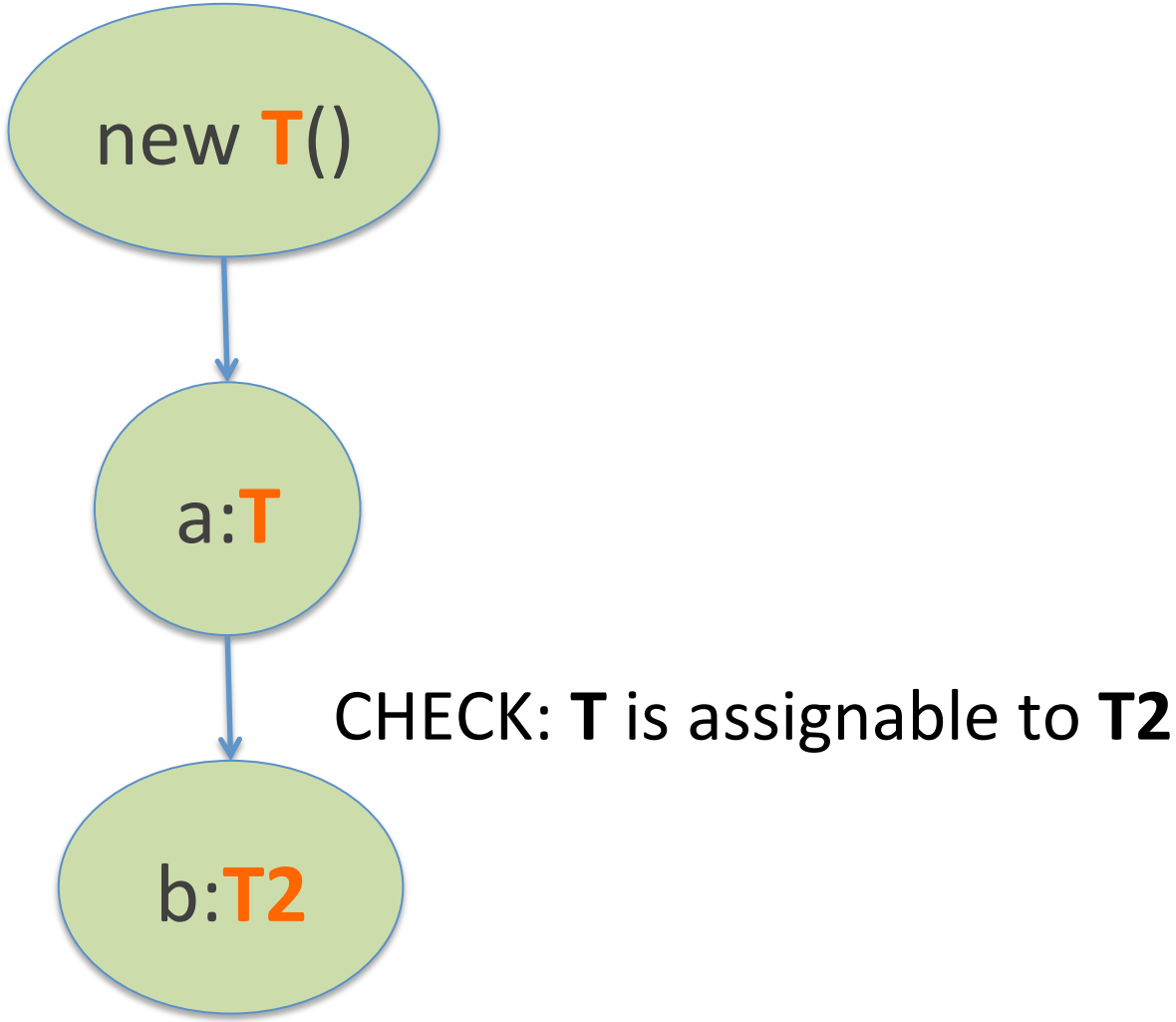
```
var a = new T()
T2 b = a
```



EXCELSIOR

Пример: проверка типов

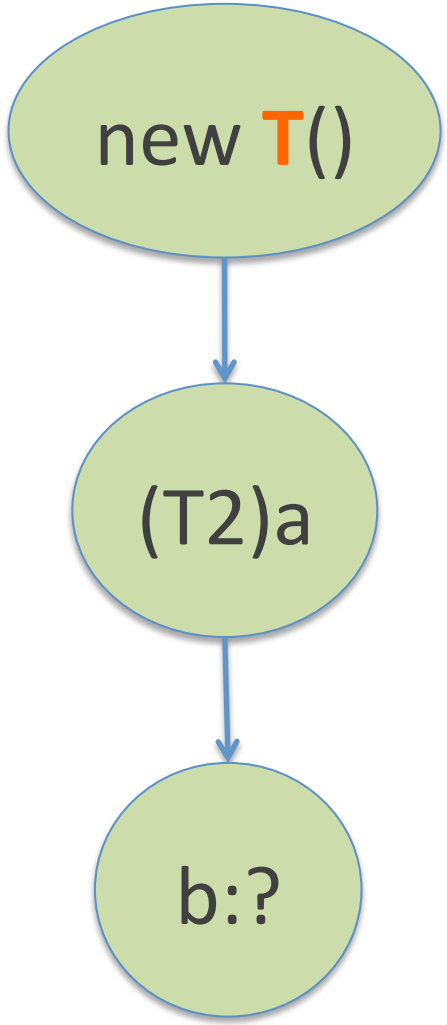
```
var a = new T()
T2 b = a
```



EXCELSIOR

Пример: преобразование типов

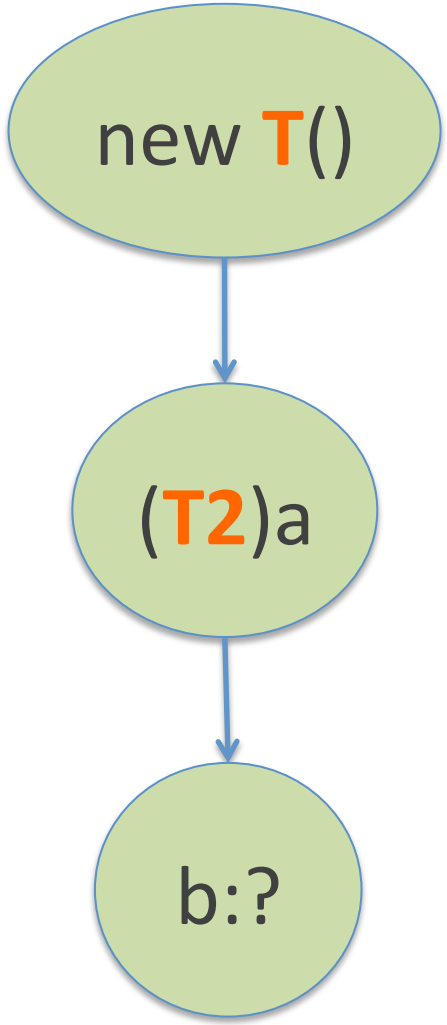
```
var a = new T()
var b = (T2)a
```



EXCELSIOR

Пример: преобразование типов

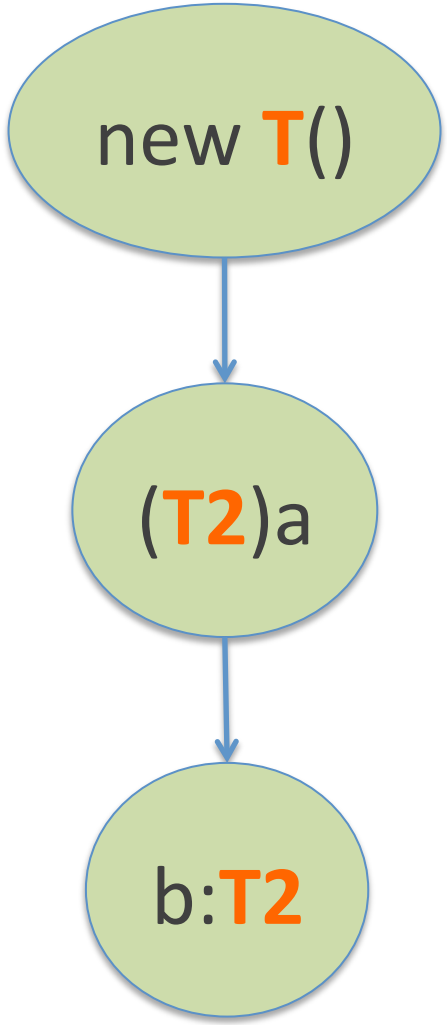
```
var a = new T()
var b = (T2)a
```



EXCELSIOR

Пример: преобразование типов

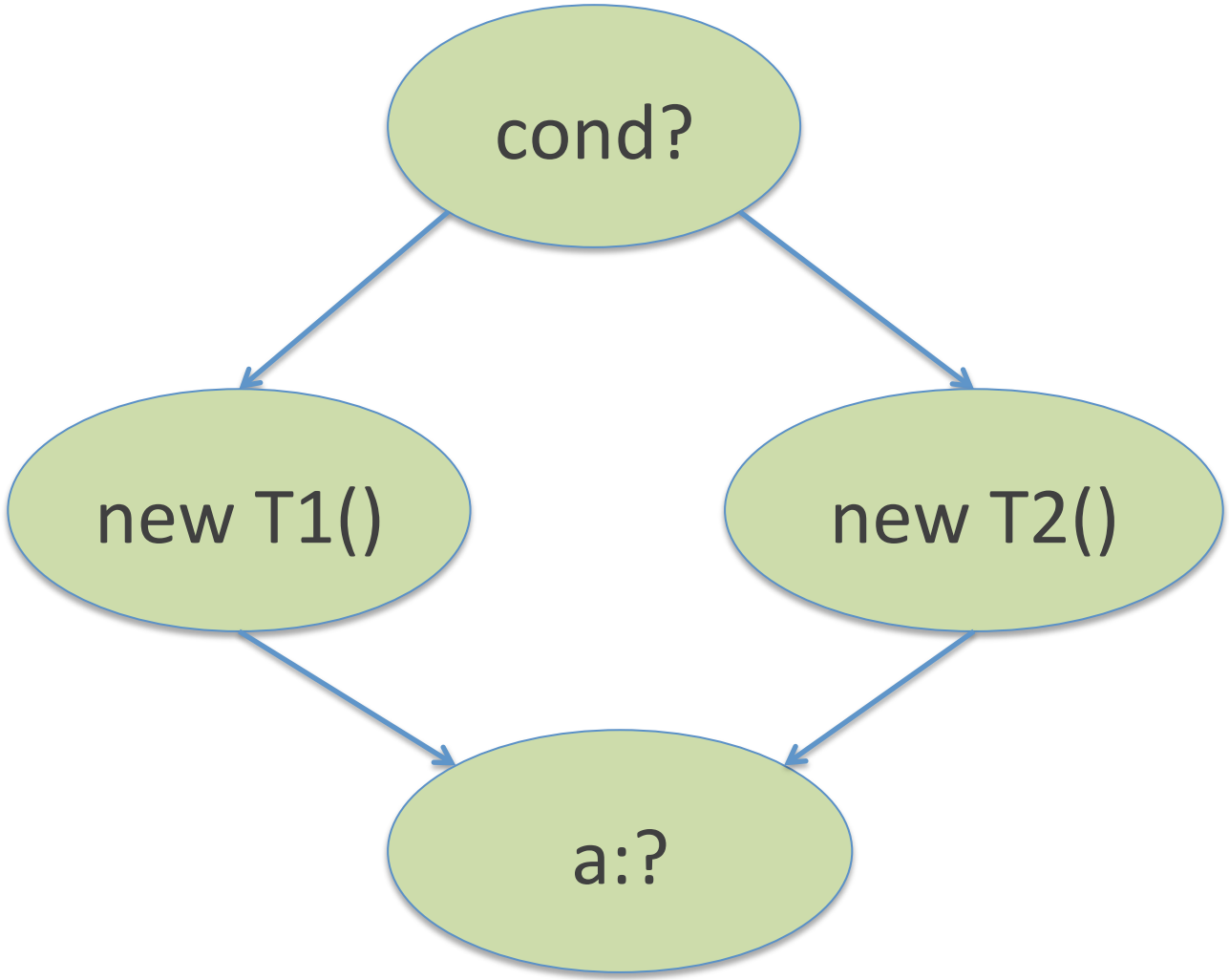
```
var a = new T()
var b = (T2)a
```



EXCELSIOR

Пример: слияние типов

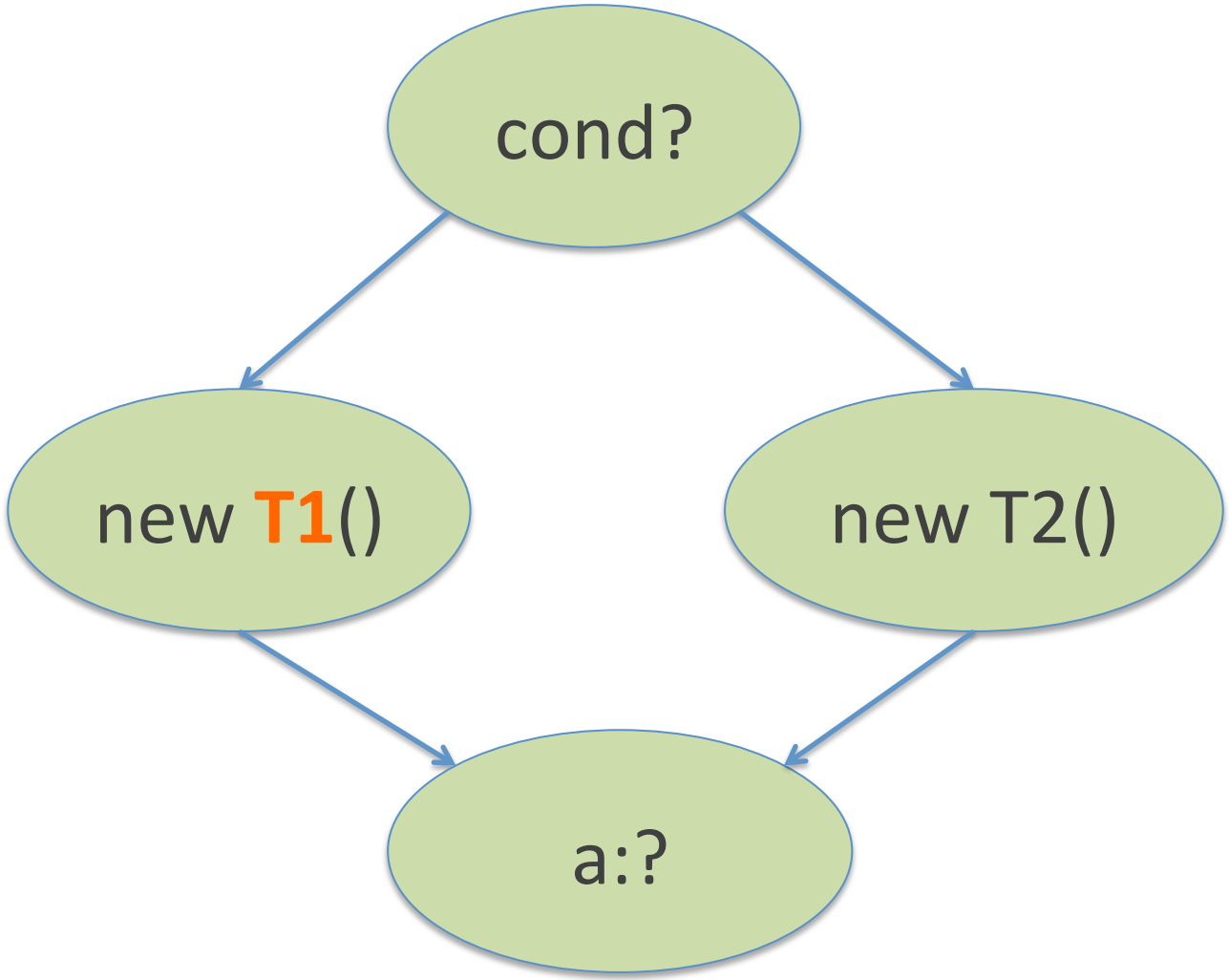
```
var a = cond?
        new T1() :
        new T2()
```



EXCELSIOR

Пример: слияние типов

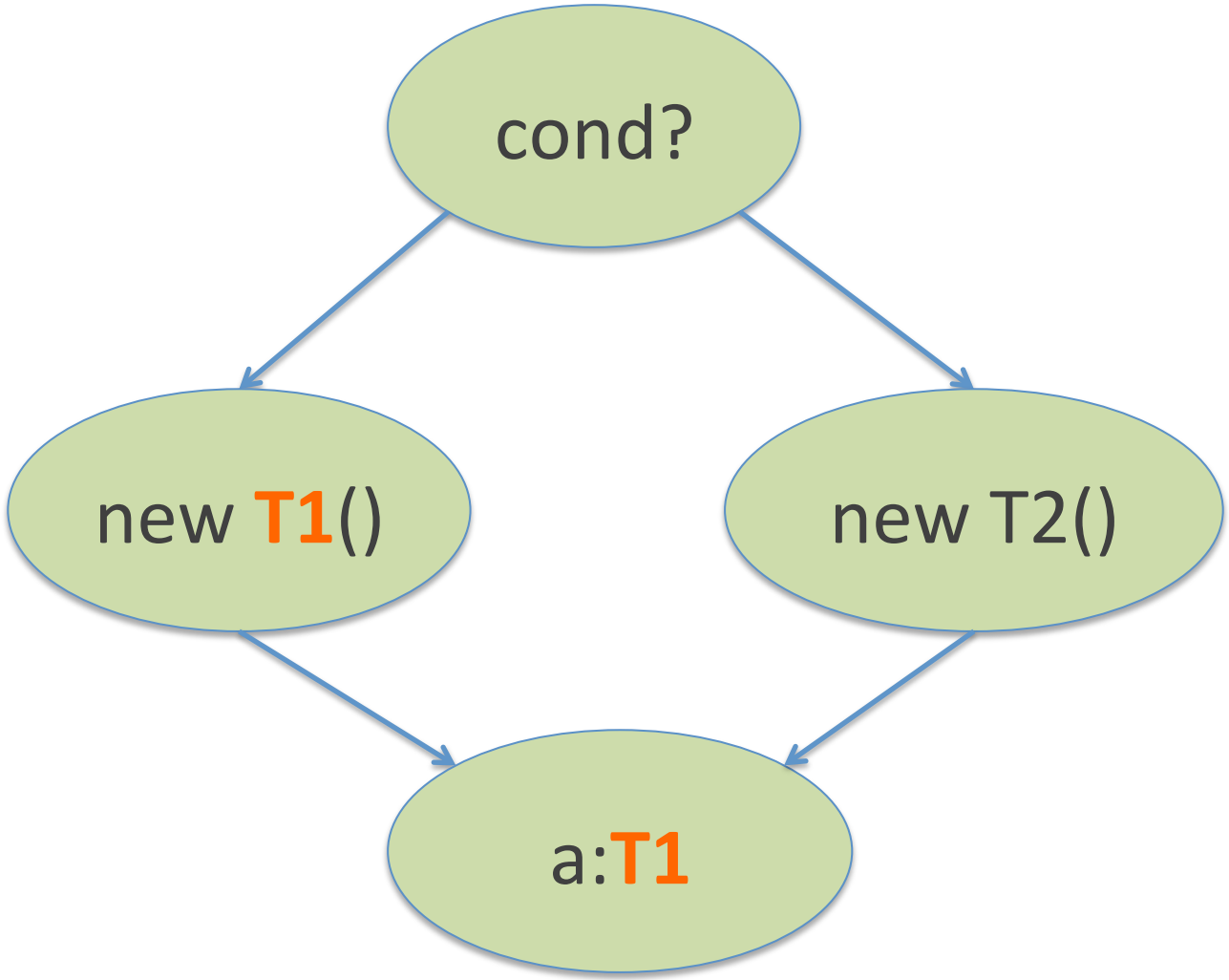
```
var a = cond?
        new T1() :
        new T2()
```



EXCELSIOR

Пример: слияние типов

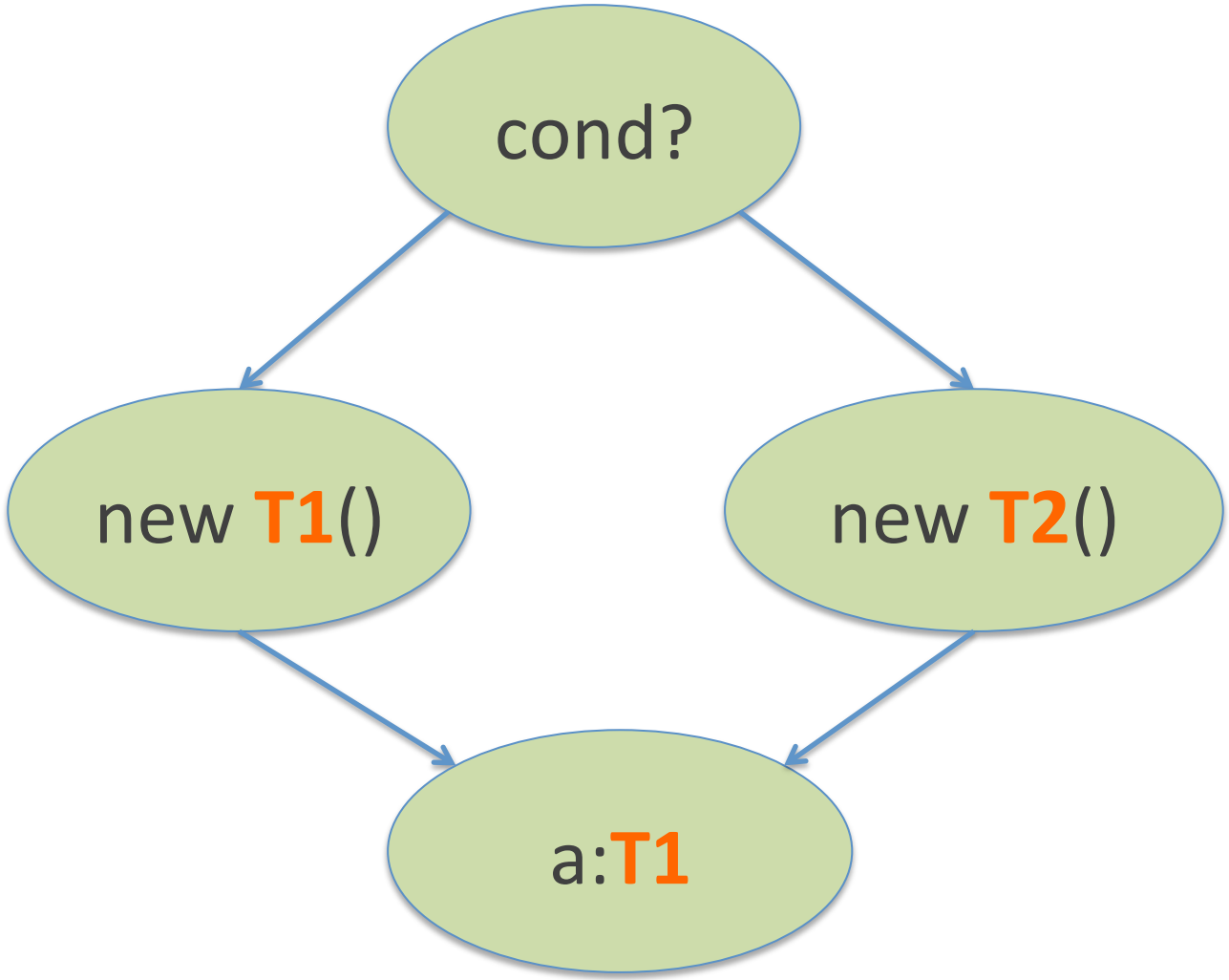
```
var a = cond?
        new T1() :
        new T2()
```



EXCELSIOR

Пример: слияние типов

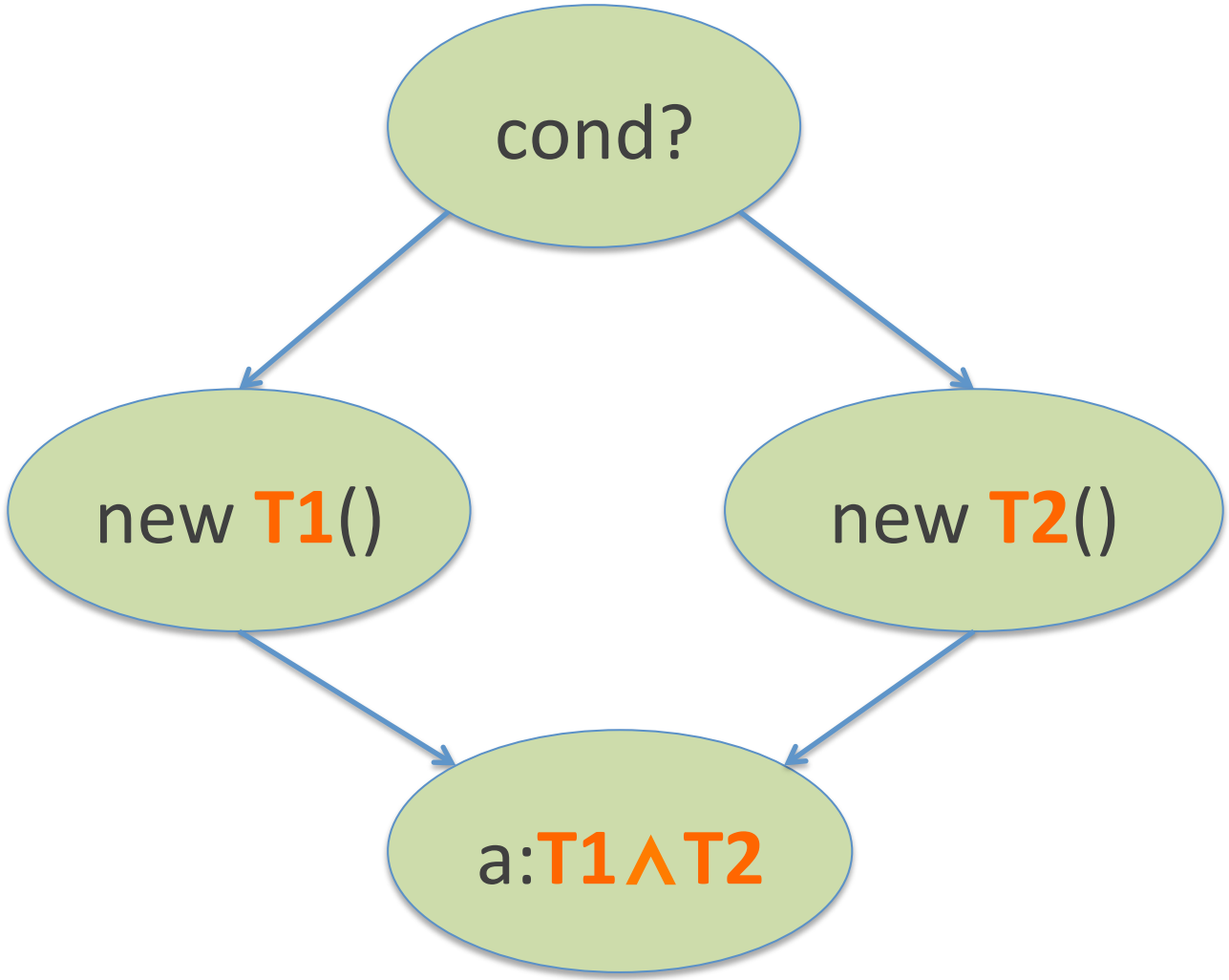
```
var a = cond?
        new T1() :
        new T2()
```



EXCELSIOR

Пример: слияние типов

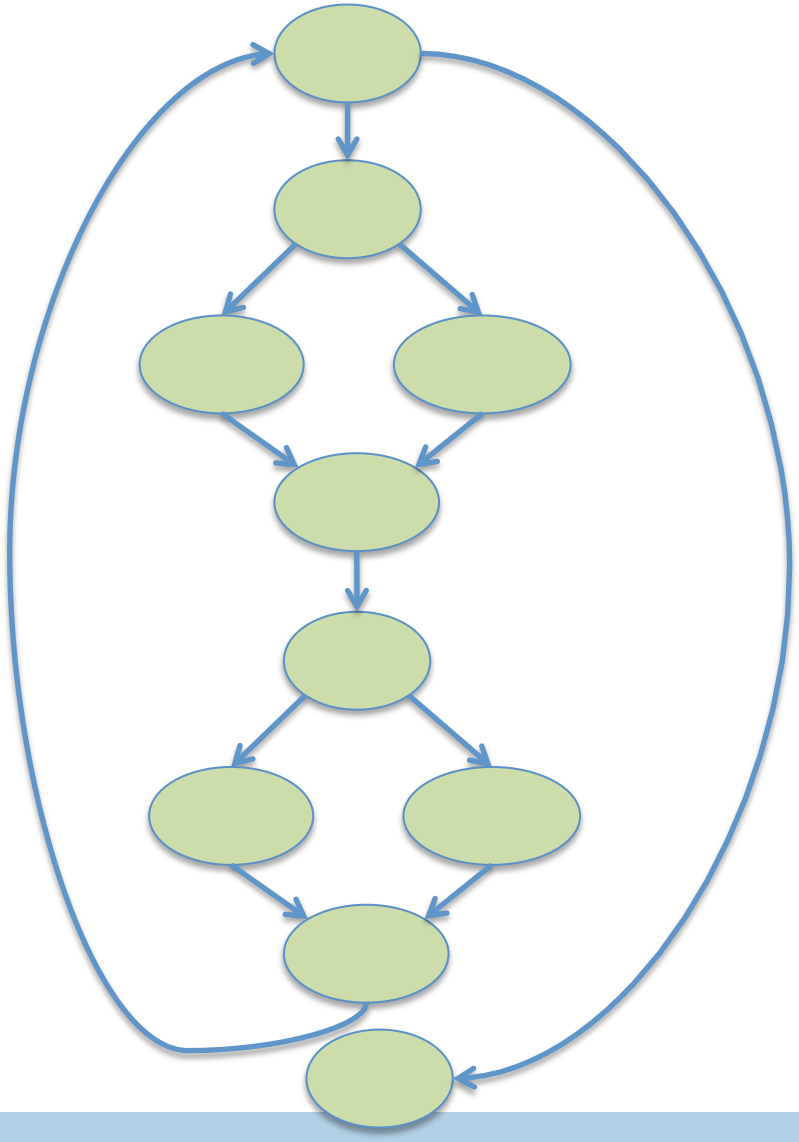
```
var a = cond?
        new T1() :
        new T2()
```



EXCELSIOR

Пример: условия, циклы

```
var a = null
var b = null
while (cond1) {
    a = (cond2) ?
            b :
            new T1();
    b = (cond3) ?
            a :
            new T2();
}
```



EXCELSIOR

Статический потоковый анализ

Статический потоковый анализ имеет дело с:

Статический потоковый анализ

Статический потоковый анализ имеет дело с:

- Потоковым графом (управляющий граф программы)

EXCELSIOR

Статический потоковый анализ

Статический потоковый анализ имеет дело с:

- Потоковым графом (управляющий граф программы)
- Пометками на вершинах (свойства, вычисляемые факты)
  - Начальная разметка – известные факты в начале

EXCELSIOR

Статический потоковый анализ

Статический потоковый анализ имеет дело с:

- Потоковым графом (управляющий граф программы)
- Пометками на вершинах (свойства, вычисляемые факты)
  - Начальная разметка – известные факты в начале
- Свойства *текут* от вершины к вершине

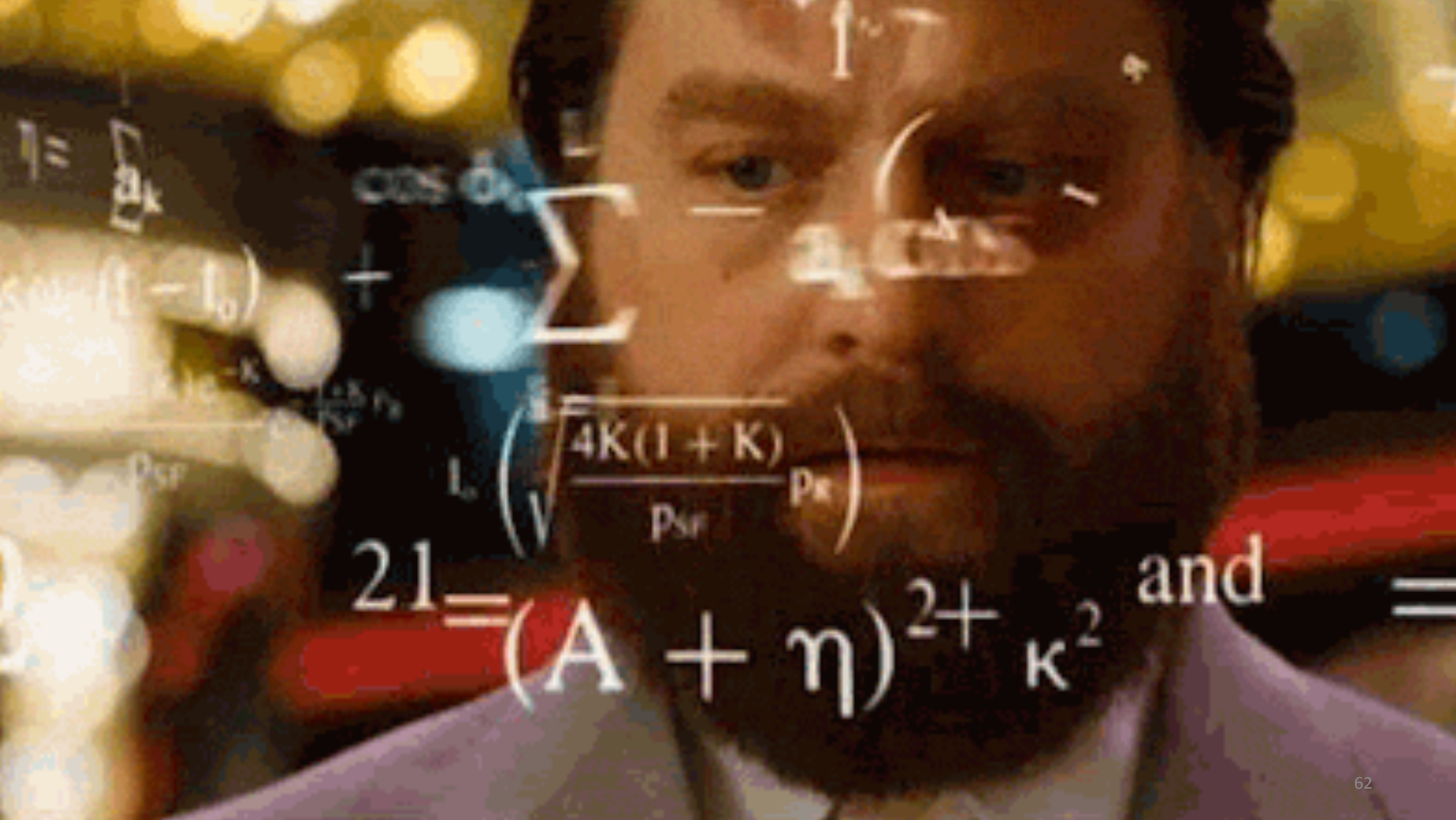
EXCELSIOR

Статический потоковый анализ

Статический потоковый анализ имеет дело с:

- Потоковым графом (управляющий граф программы)
- Пометками на вершинах (свойства, вычисляемые факты)
  - Начальная разметка – известные факты в начале
- Свойства *текут* от вершины к вершине
- Свойства *преобразуются* на входе в вершину и *сливаются* с уже имеющимися у вершины

EXCELSIOR


$\sum$
$\sqrt{\frac{4K(1+K)}{P_{sr}}}$
$21 = (A+\eta)^2 + \kappa^2$ and

Полурешетка свойств

Полурешетка – $\langle L, \wedge \rangle$:

$\forall x, y, z \in L$

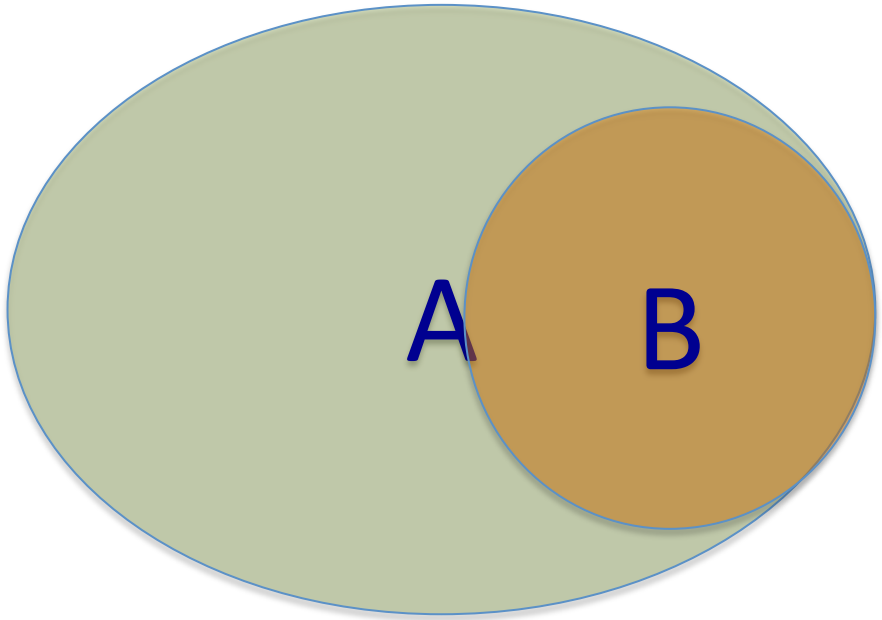
1. $x \wedge x = x$ (самоприменимость)
2. $x \wedge y = y \wedge x$ (коммутативность)
3. $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ (ассоциативность)

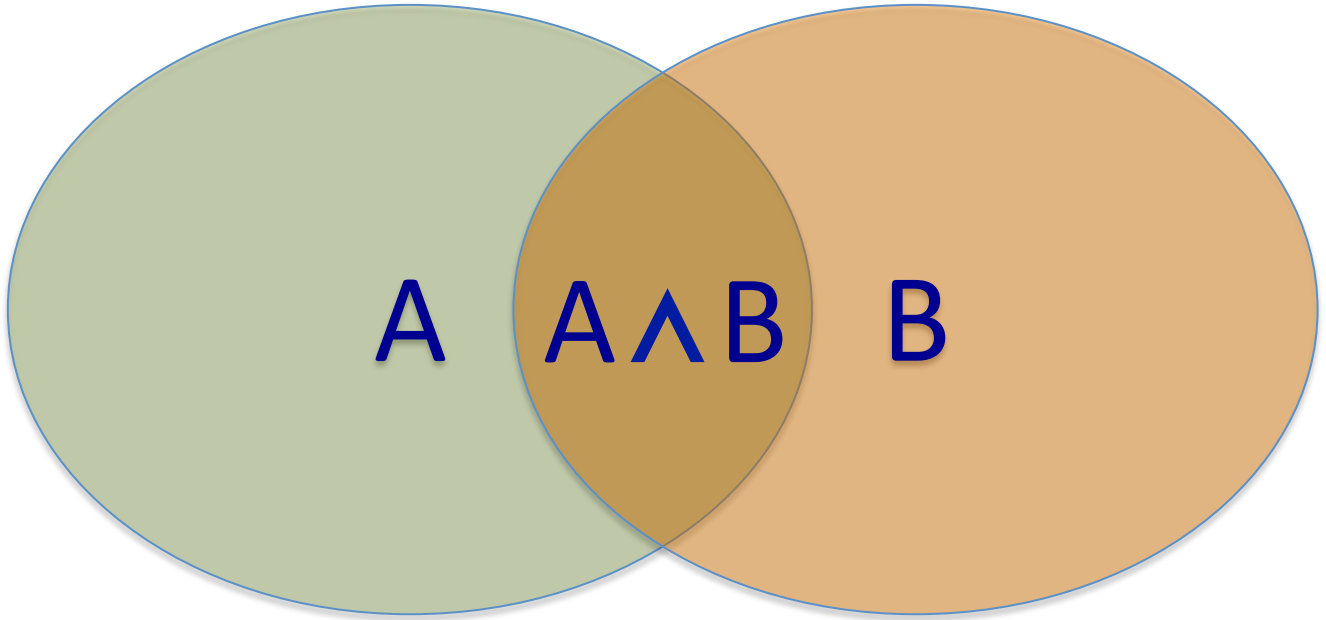
Частичный порядок – $\leq$:

$x \leq y \Leftrightarrow_{def} x \wedge y = x$

$x < y \Leftrightarrow_{def} x \wedge y = x \quad \& \quad x \neq y$

Пример: множества





B < A

C:=A∧B, C<A, C<B

EXCELSIOR

Полурешетка свойств

Полурешетка *свойств* – это ограниченная решетка с обрывающимися строго убывающими цепями:

1. $\forall\, x_1 > x_2 > \ldots\,, \exists k\colon \nexists y \in L,\ x_k > y$

2. $\forall x, \exists b_x : \forall\ \text{chain} \in \{x > x_1 > \ldots > x_k\},\ |\text{chain}| < b_x$ (ограниченность),
   $B_L := \max\{b_x\}$ – высота полурешетки

3. $\exists \bot \in L\colon \bot \wedge x = \bot,\ \forall x \in L$ (нуль, “дно”)

4. $\exists \top \in L\colon \top \wedge x = x,\ \forall x \in L$ (единица, "штопор")

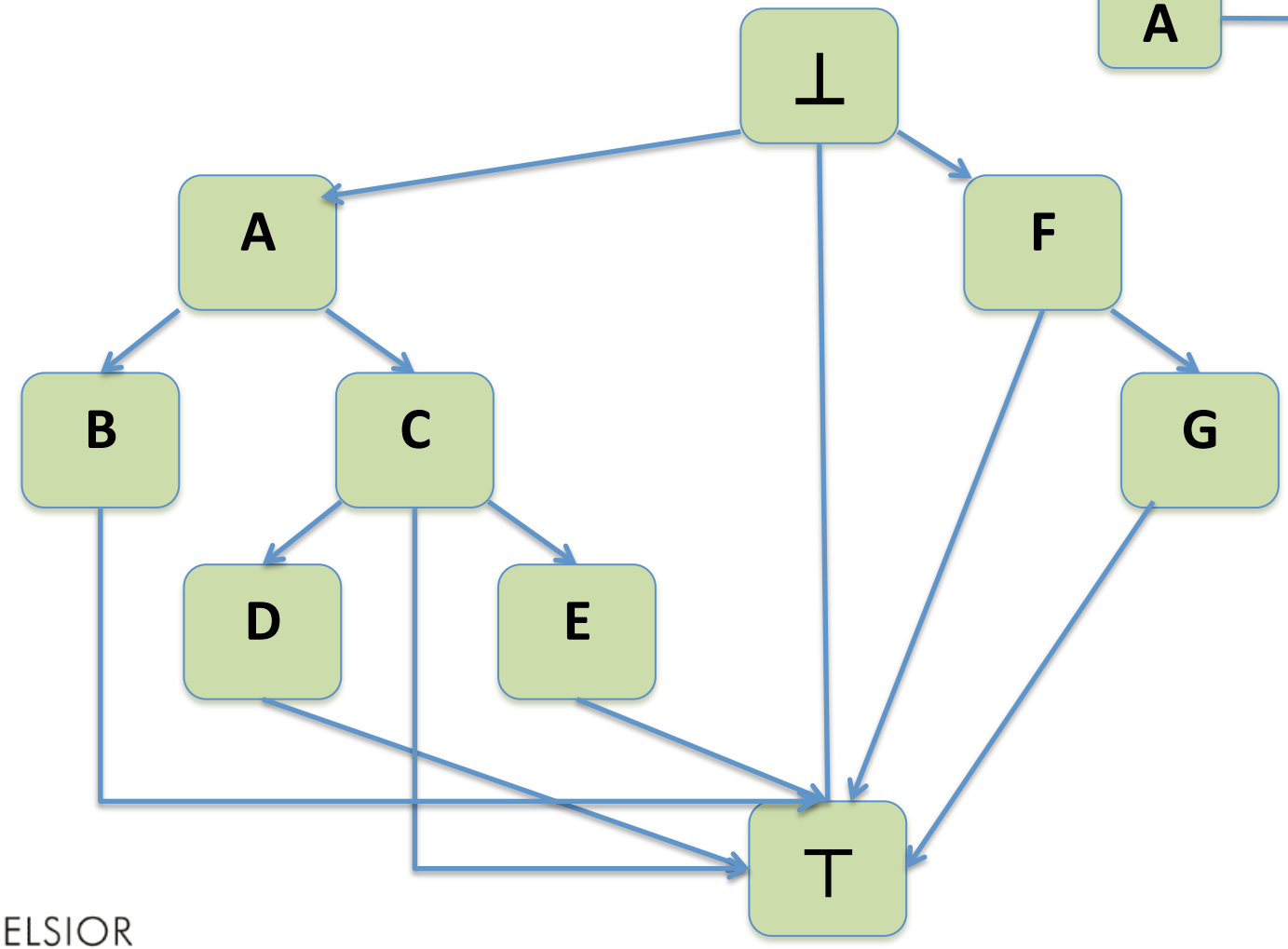
EXCELSIOR

Примеры полурешеток

- Множества

- Полурешетка (!) иерархии классов
  $\wedge$:= LCA (lowest common ancestor)

Полурешетка иерархии классов



A → B := B extends A

⊥ := j.l.Object

⊤ := null

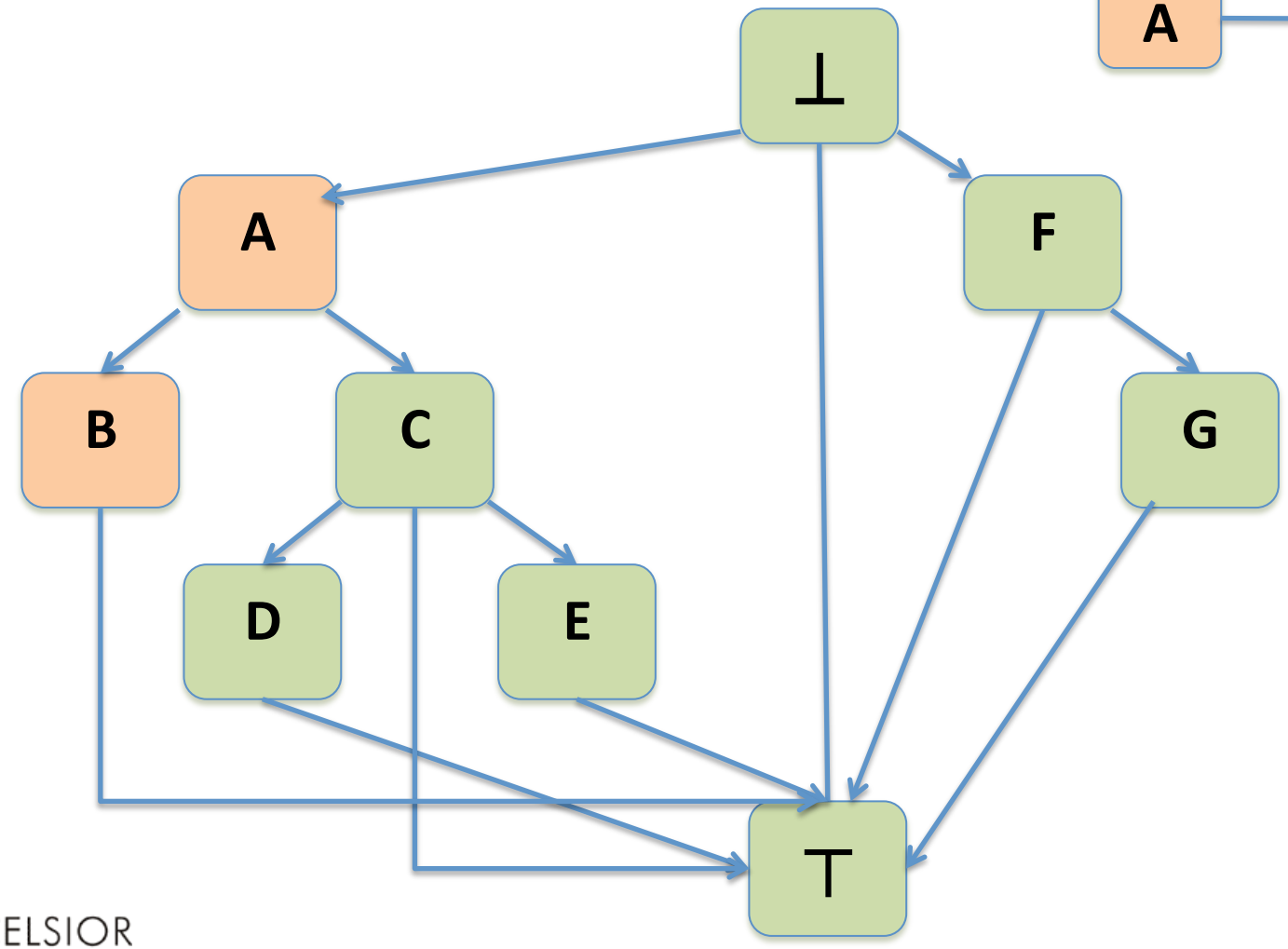
B ∧ C = A

D ∧ E = C

B ∧ E = A

EXCELSIOR

Полурешетка иерархии классов



A → B := B extends A

⊥ := j.l.Object

⊤ := null

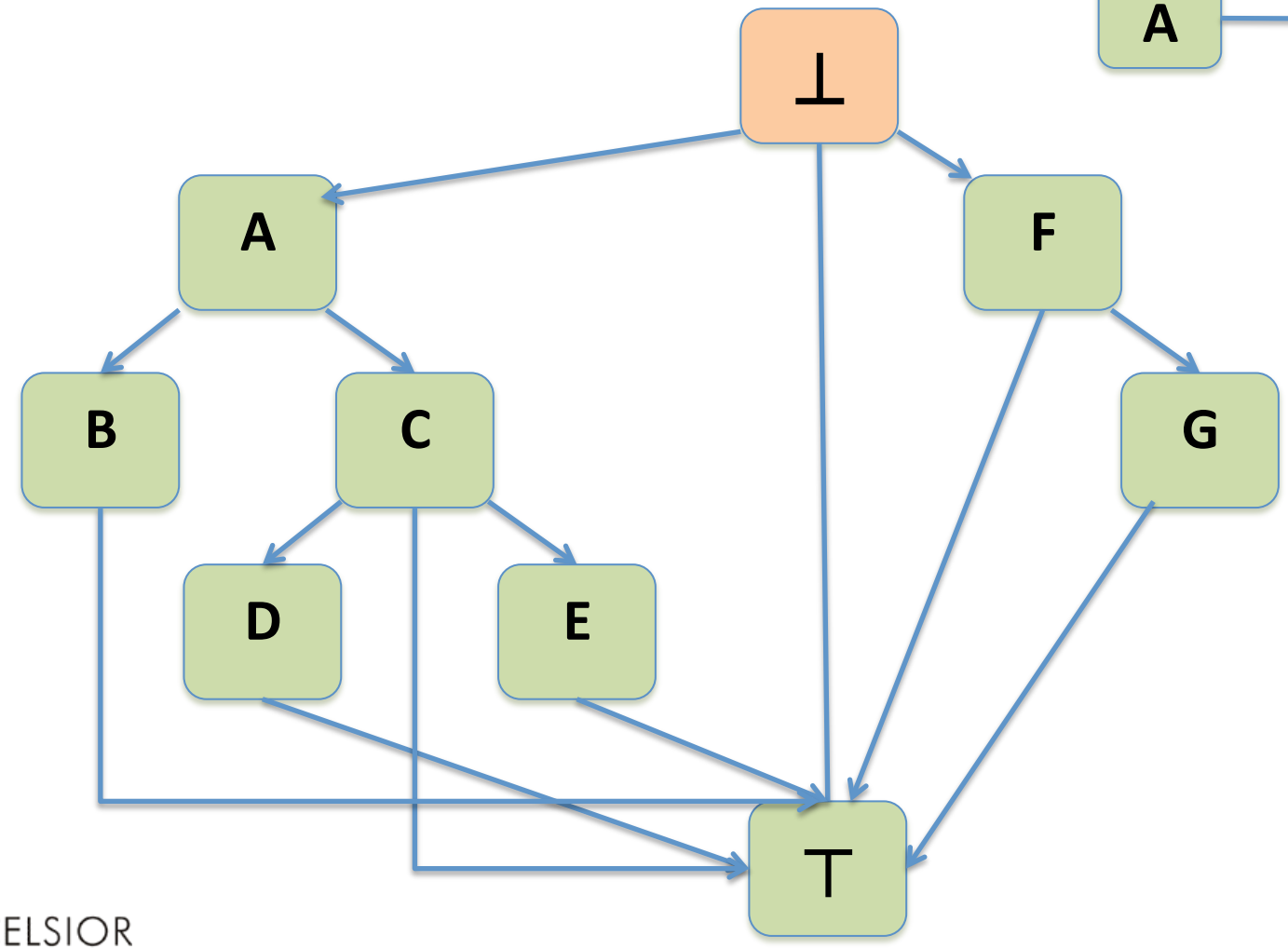
B ∧ C = A

D ∧ E = C

B ∧ E = A

EXCELSIOR

Полурешетка иерархии классов



A → B := B extends A

⊥ := j.l.Object

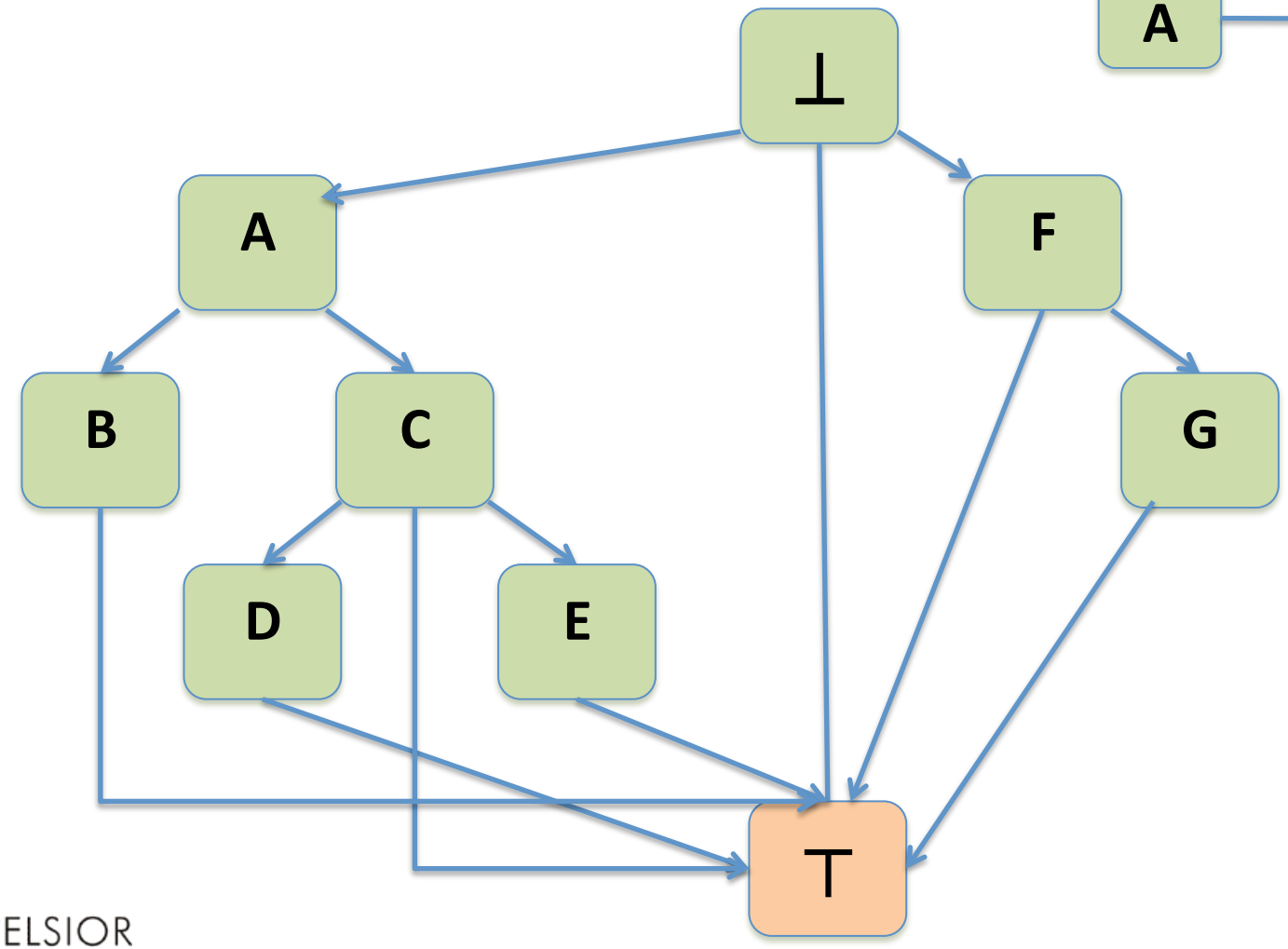
⊤ := null

B ∧ C = A

D ∧ E = C

B ∧ E = A

Полурешетка иерархии классов



A → B := B extends A

⊥ := j.l.Object

⊤ := null

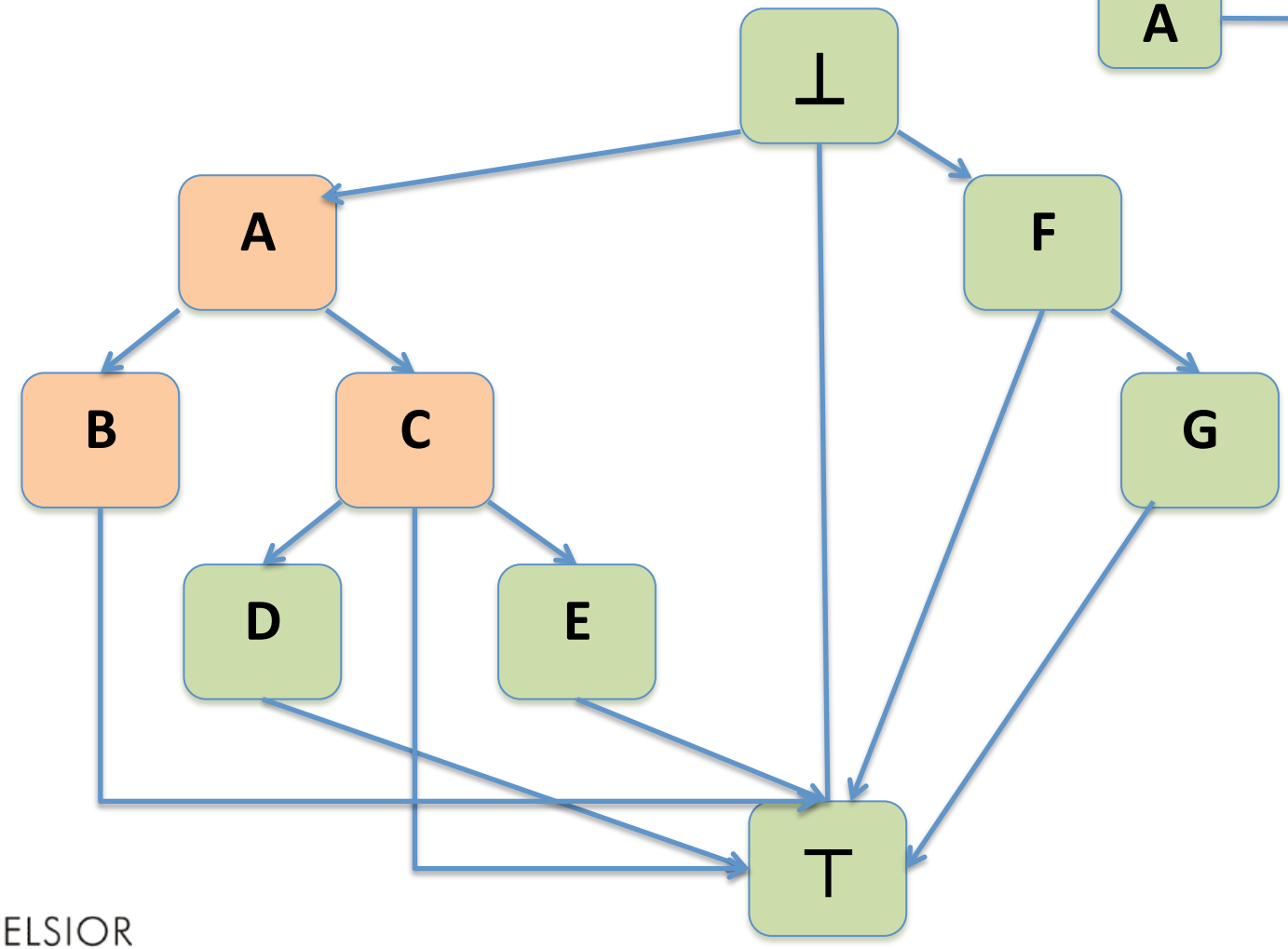
B ∧ C = A

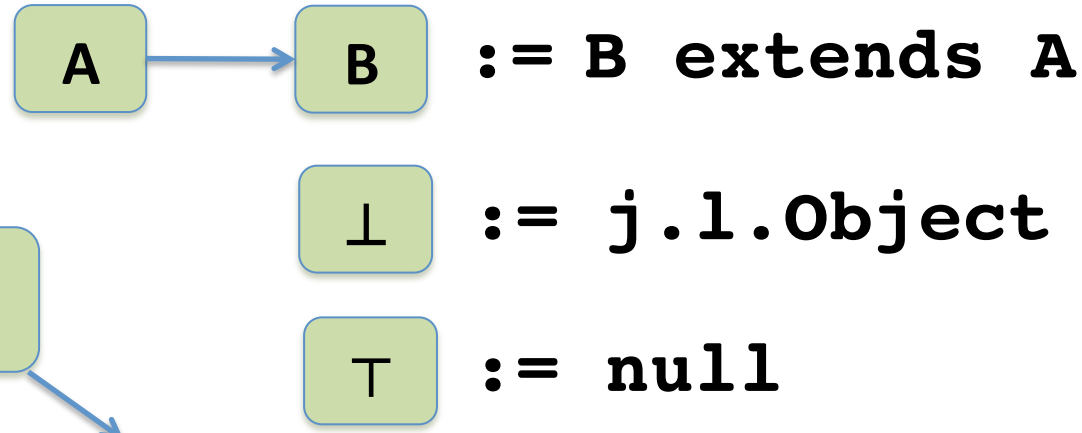
D ∧ E = C

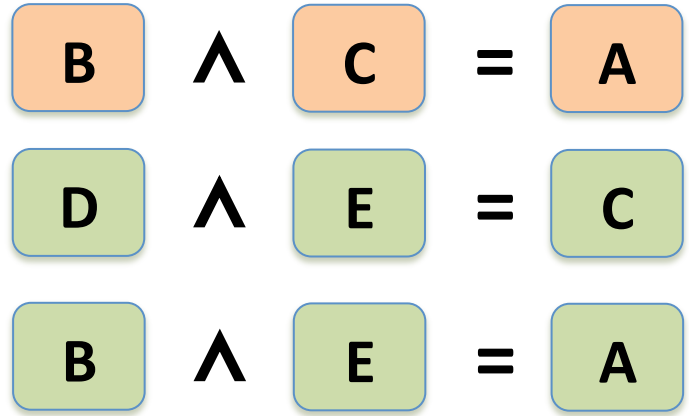
B ∧ E = A

EXCELSIOR

Полурешетка иерархии классов

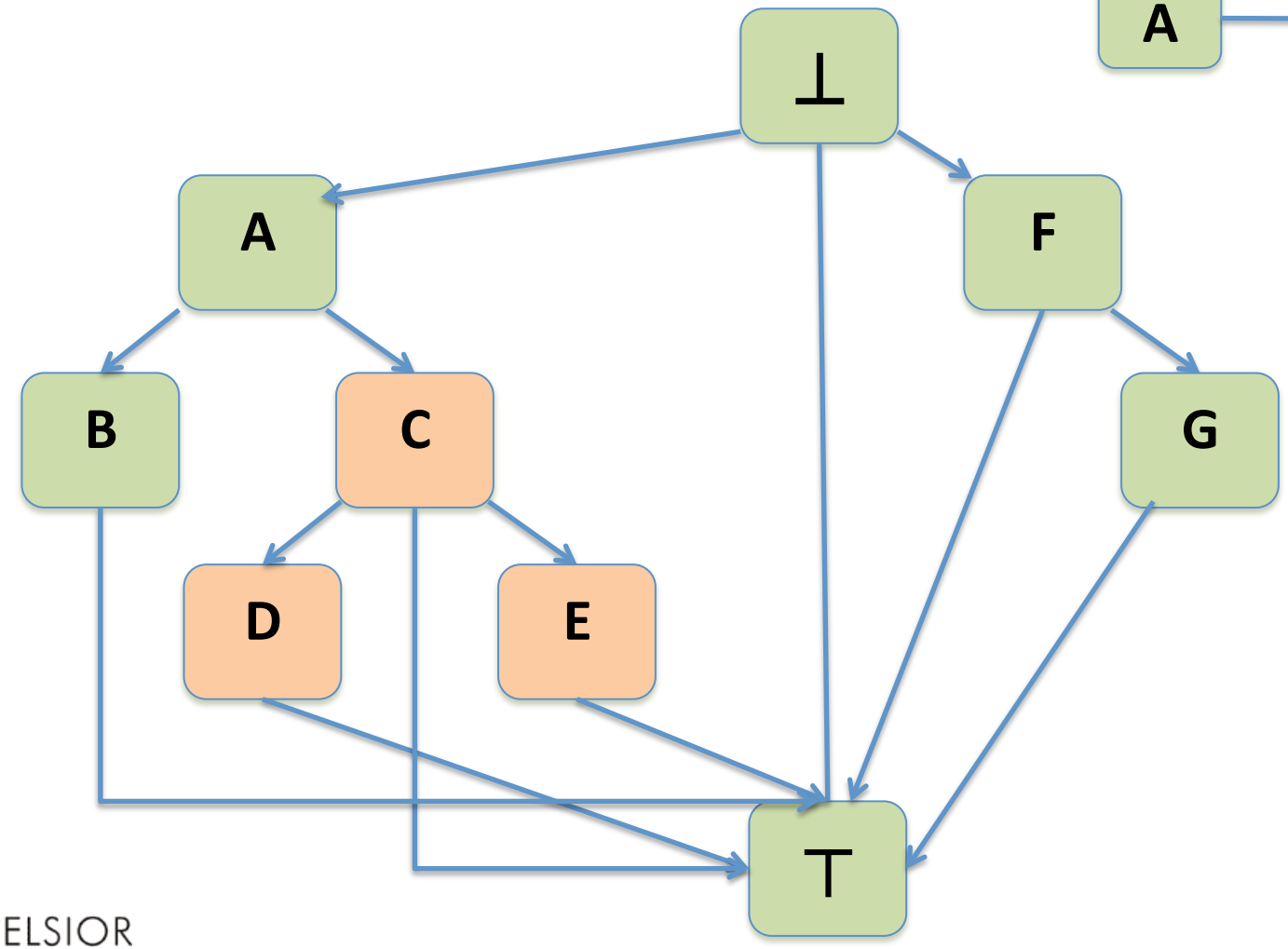


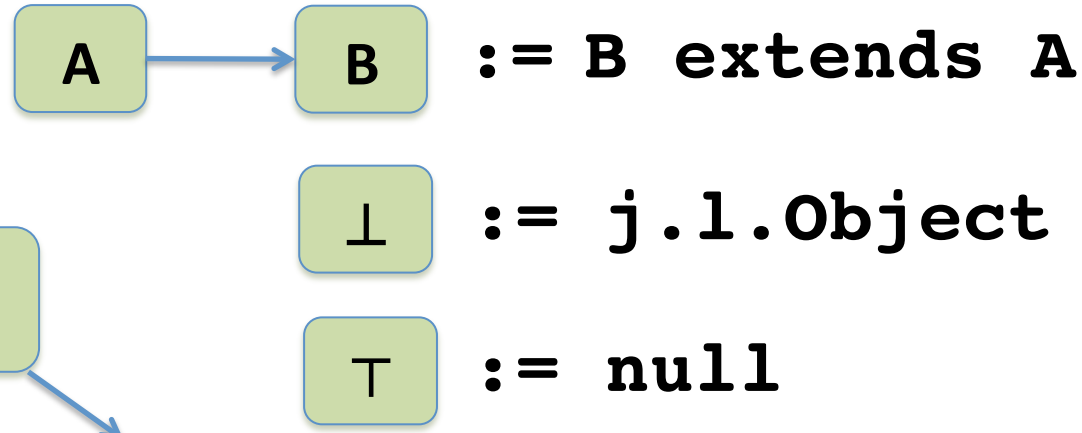


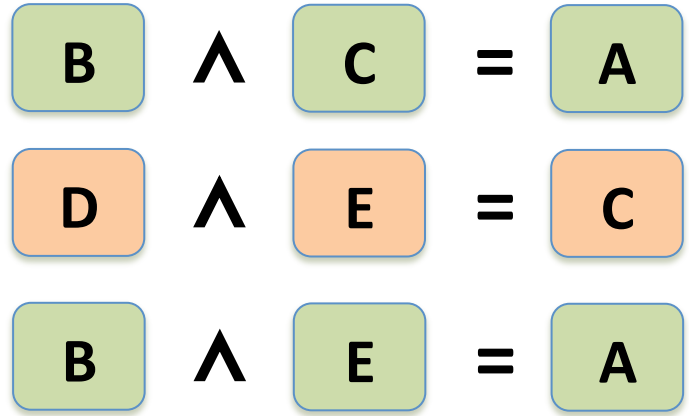


EXCELSIOR

Полурешетка иерархии классов

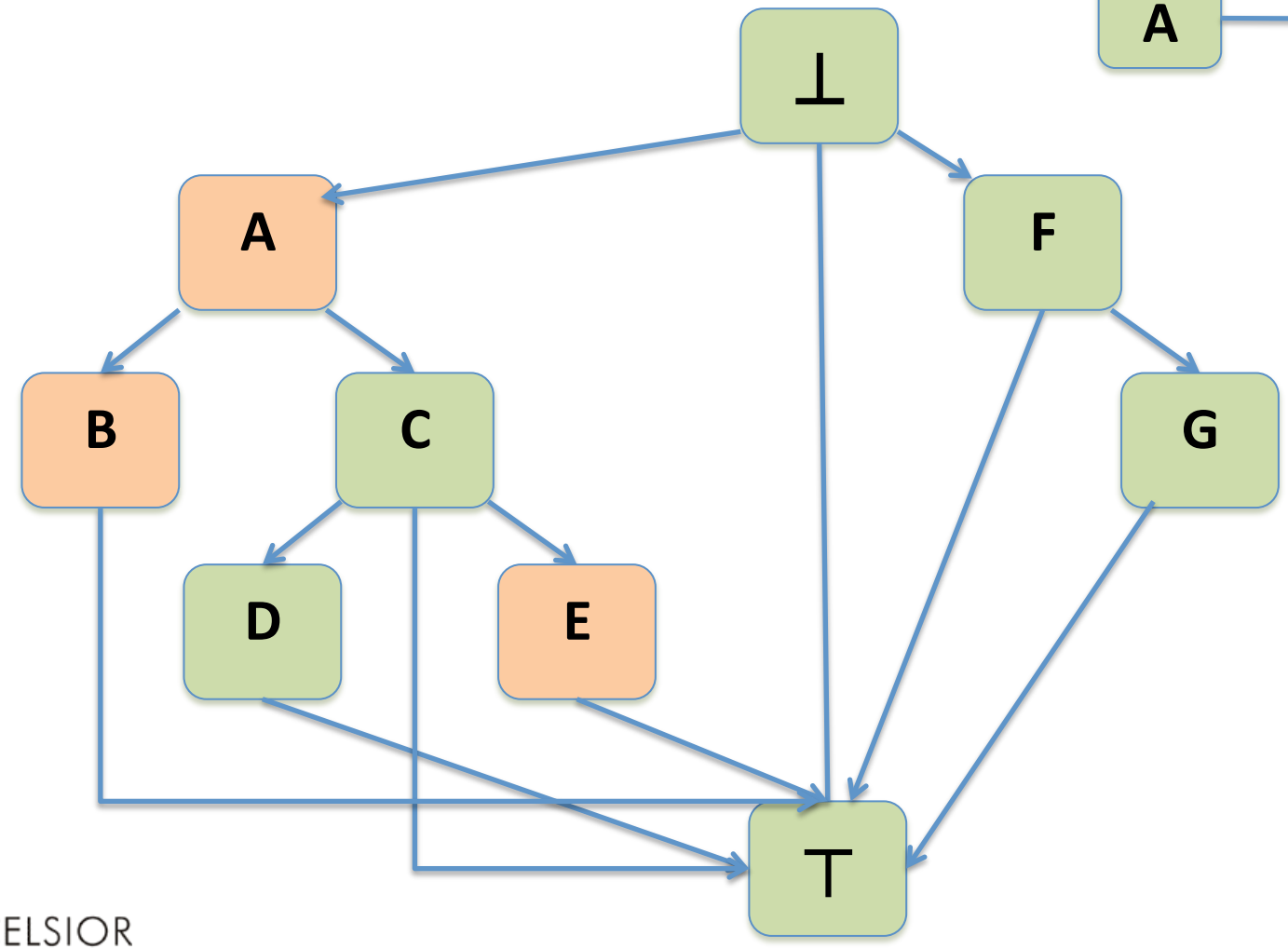






EXCELSIOR

Полурешетка иерархии классов



A → B := B extends A

⊥ := j.l.Object

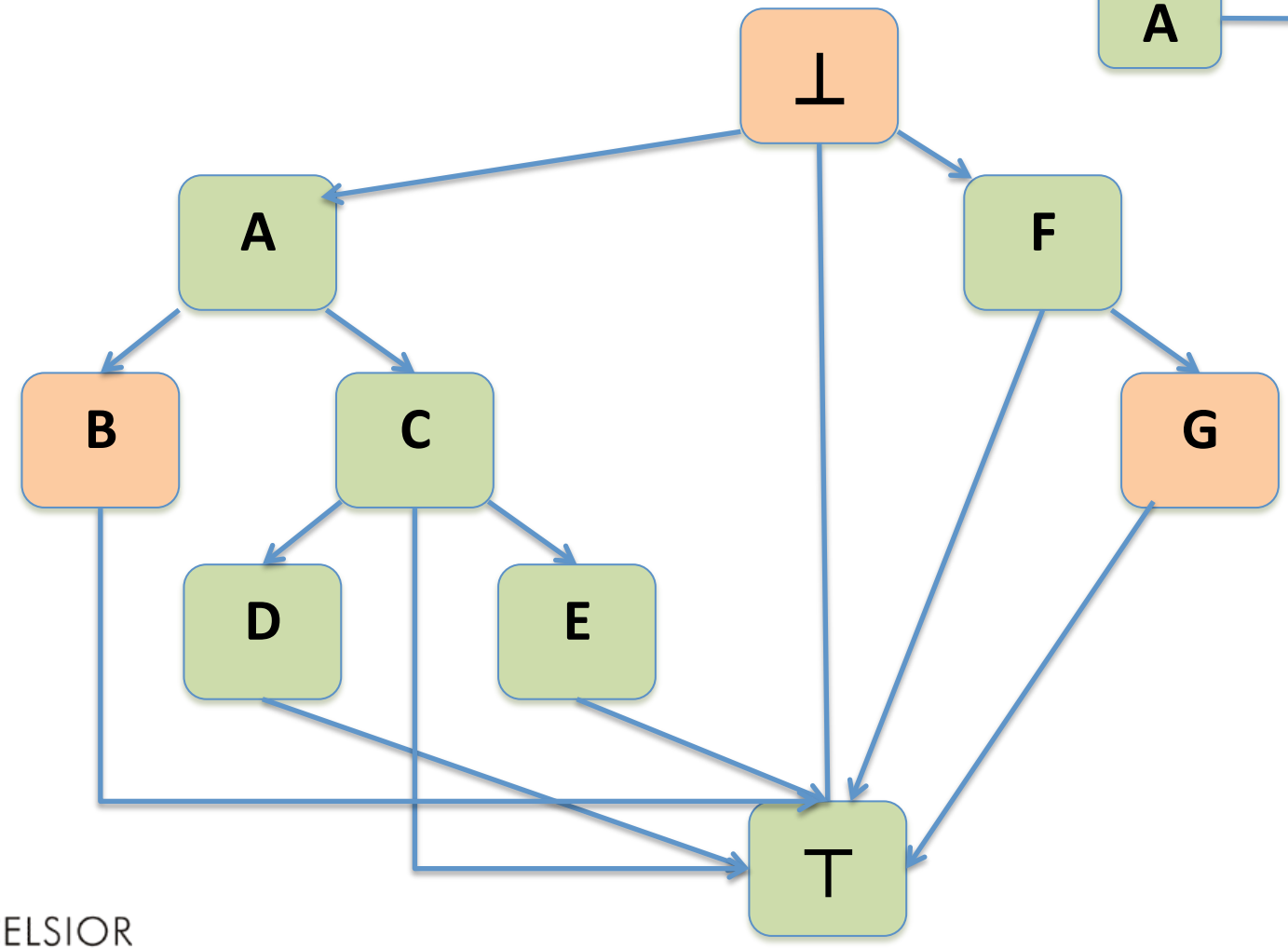
⊤ := null

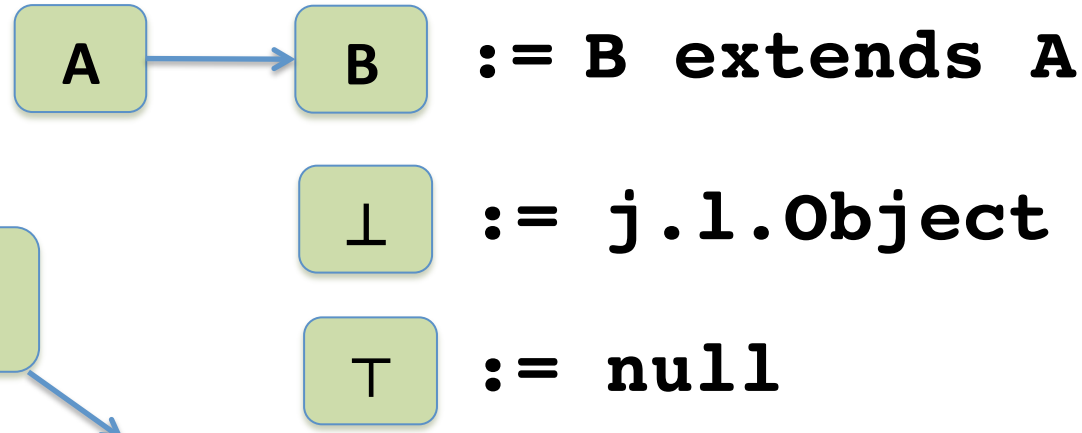
B ∧ C = A

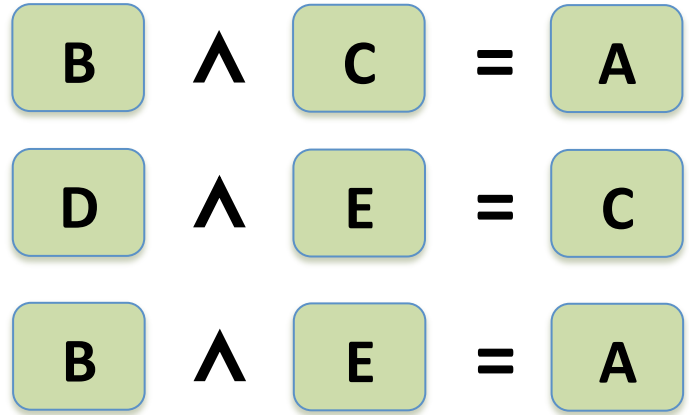
D ∧ E = C

B ∧ E = A

Полурешетка иерархии классов







EXCELSIOR

Потоковые функции

Пусть $<L, \wedge>$ – полурешетка свойств

f: $L \to L$ – монотонная функция:

$$\forall x, y \in L,\ x \leq y \Rightarrow f(x) \leq f(y)$$

Свойство GKUW (Graham, Kam, Ullman, Wegman):

f – монотонная функция $\Leftrightarrow$

$$\forall x, y \in L: f(x \wedge y) \leq f(x) \wedge f(y)$$

Потоковые функции

Пусть $<L, \wedge>$ – полурешетка свойств

f: $L \to L$ – монотонная функция:

$$\forall x, y \in L,\ x \le y \Rightarrow f(x) \le f(y)$$

Свойство GKUW (Graham, Kam, Ullman, Wegman):

f – монотонная функция $\Leftrightarrow$

$$\forall x, y \in L: f(x \wedge y) \le f(x) \wedge f(y)$$

f – *дистрибутивная функция* $\Leftrightarrow_{def}$

$$\forall x, y \in L: f(x \wedge y) = f(x) \wedge f(y)$$

Пример дистрибутивной функции

Операция преобразования типа (cast):

(T)v

Дистрибутивная функция

EXCELSIOR

Пример дистрибутивной функции

Операция преобразования типа (cast):

$$(T)v$$

Дистрибутивная функция:

$$Cast(T, Tv) := T < Tv \ ? \ Tv : T, \text{ где } Tv := Type(v)$$

EXCELSIOR

Пример дистрибутивной функции

Операция преобразования типа (cast):

$$(T)v$$

Дистрибутивная функция:

$$Cast(T, Tv) := T < Tv \ ? \ Tv : T, \text{ где } Tv := Type(v)$$

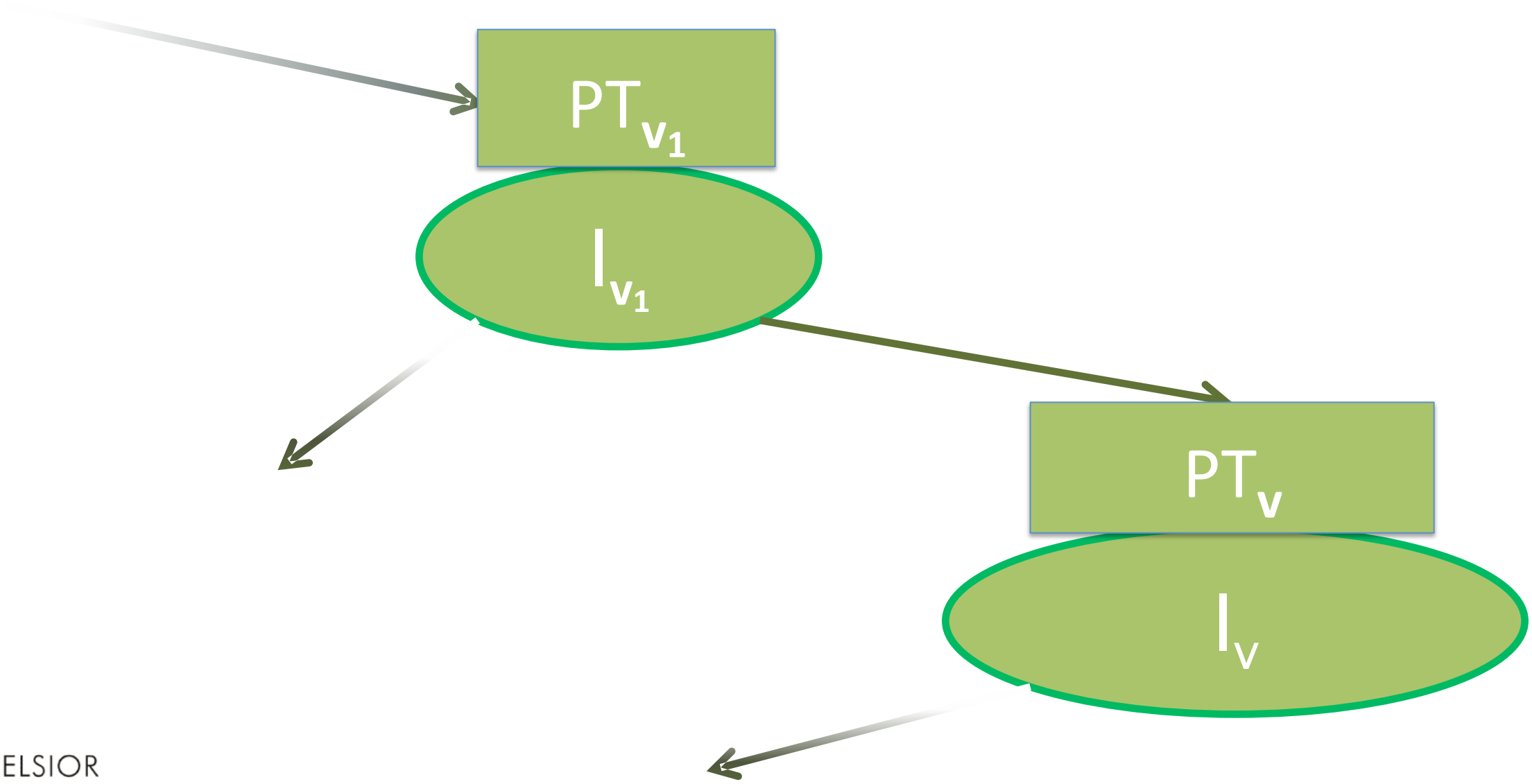
$$Cast(T, Tv1 \wedge Tv2) == Cast(T, Tv1) \wedge Cast(T, Tv2)$$

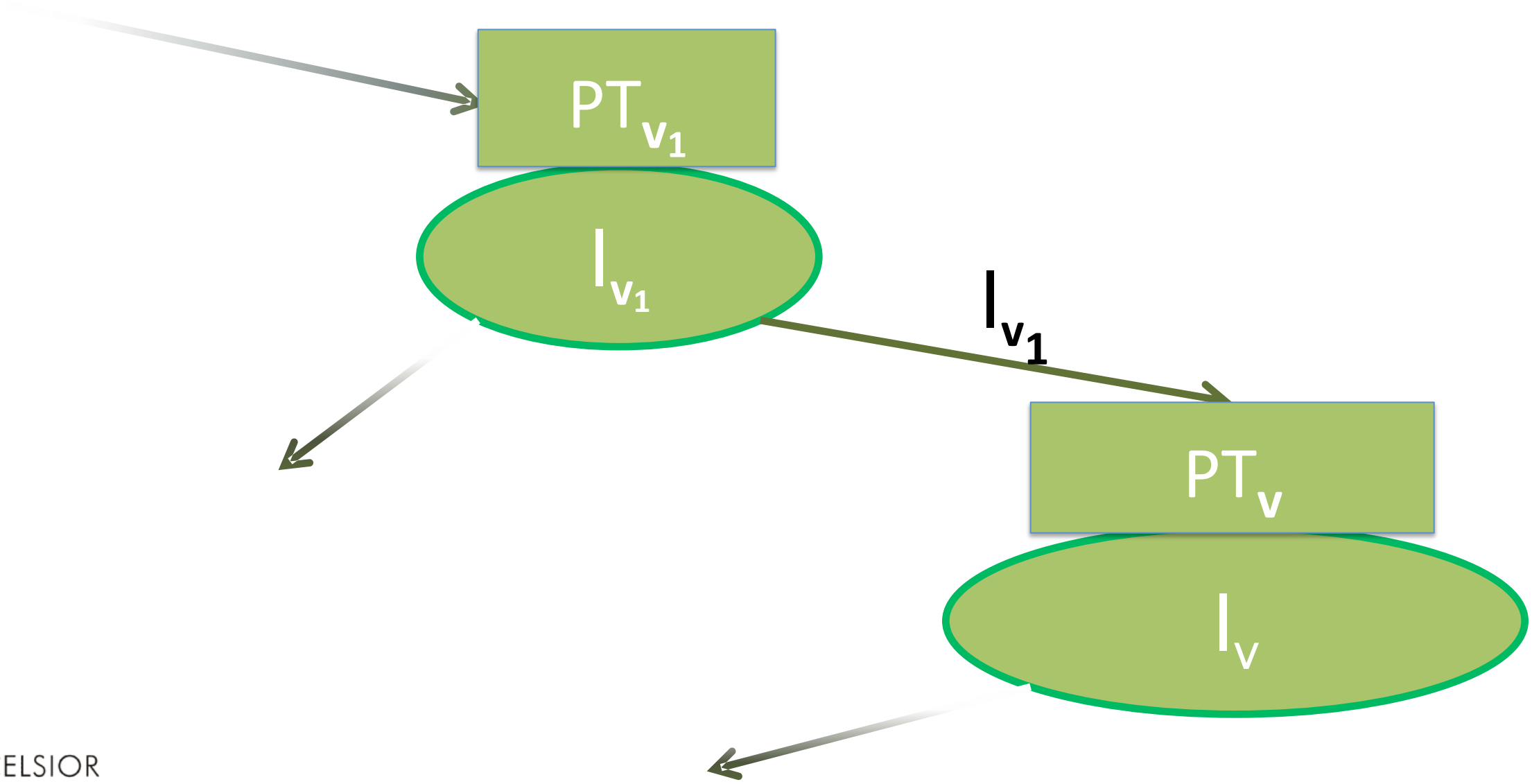
EXCELSIOR

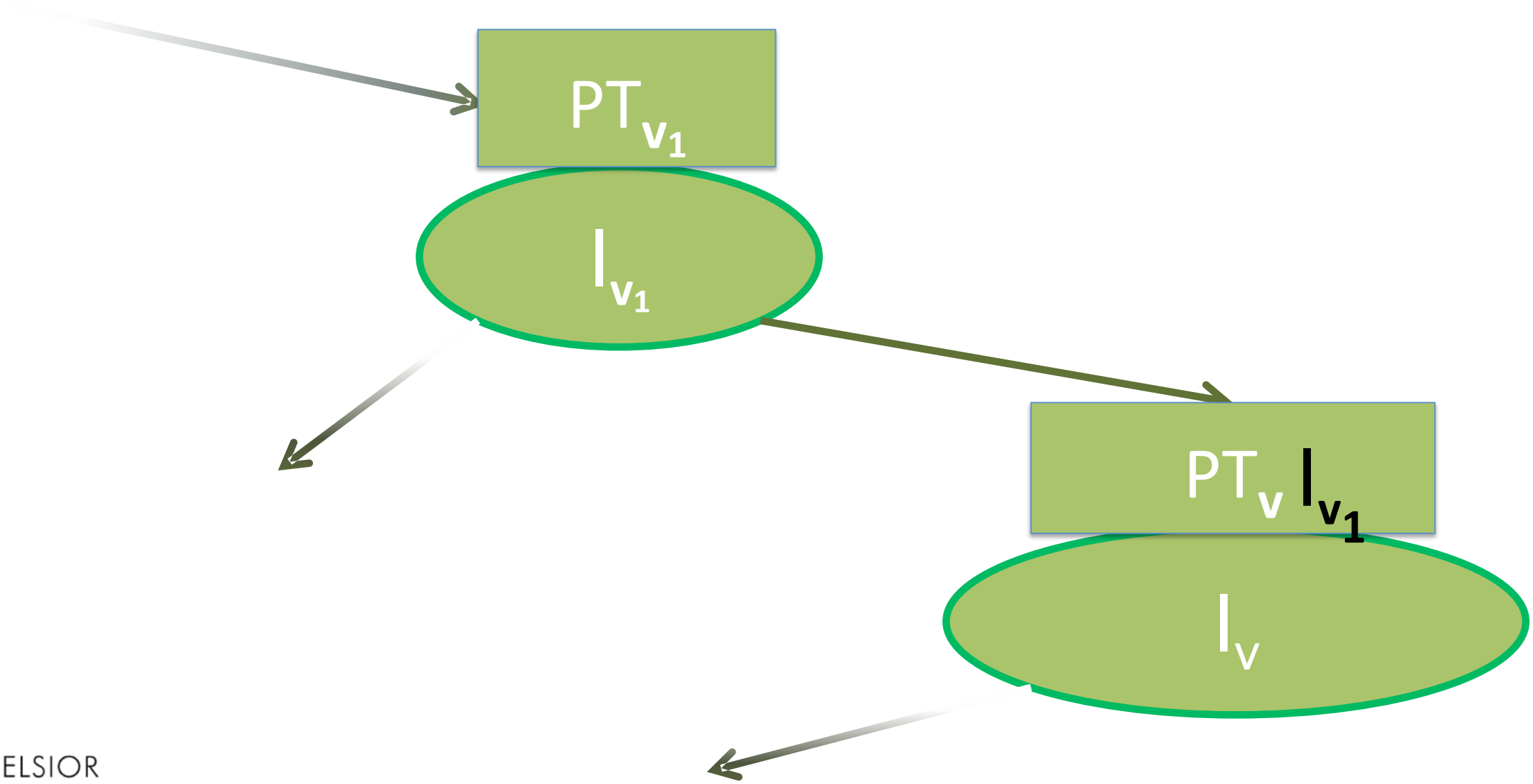
Постановка задачи потокового анализа

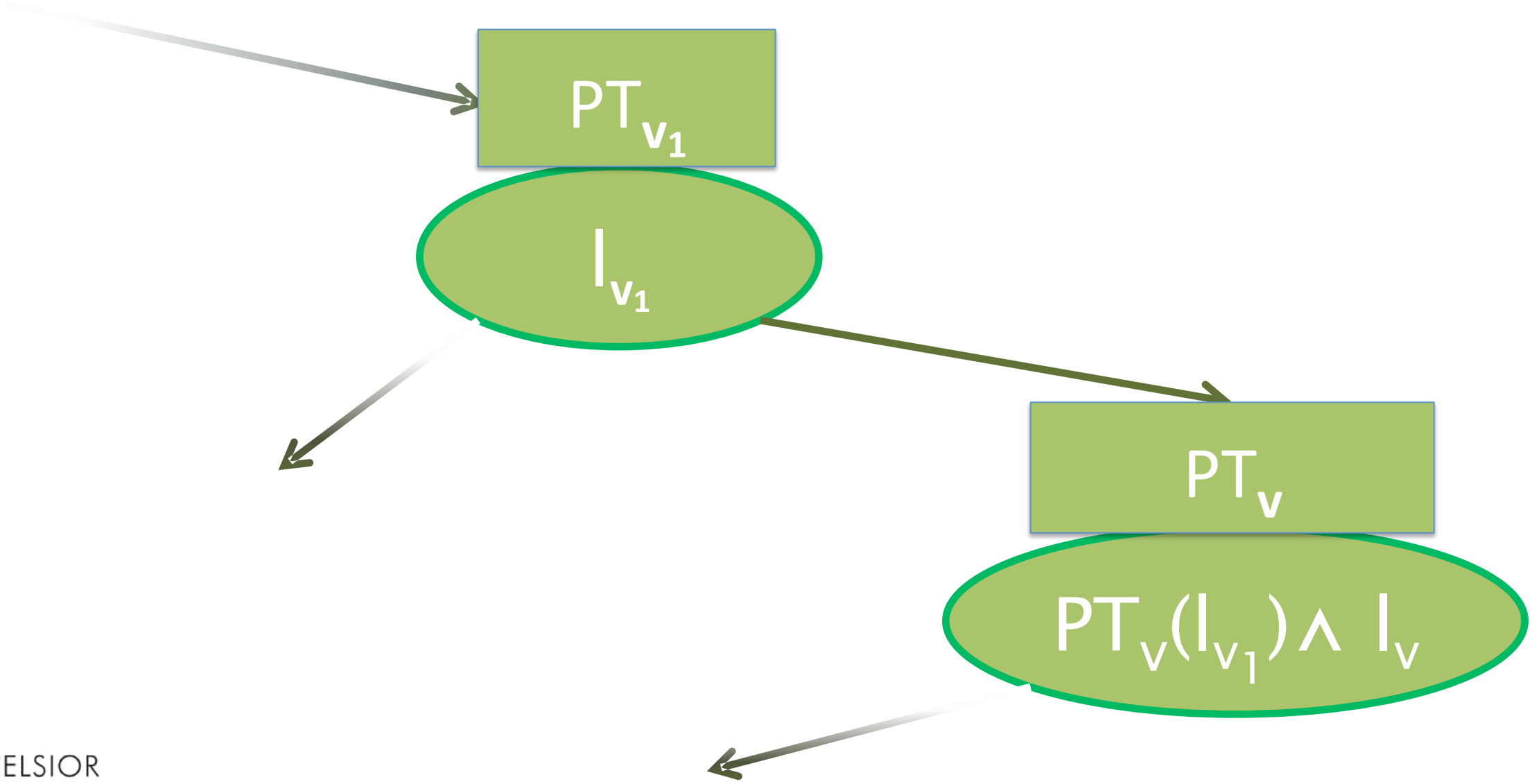
Meet-over-all-paths (пересечение по всем путям) – (желаемое) решение задачи потокового анализа

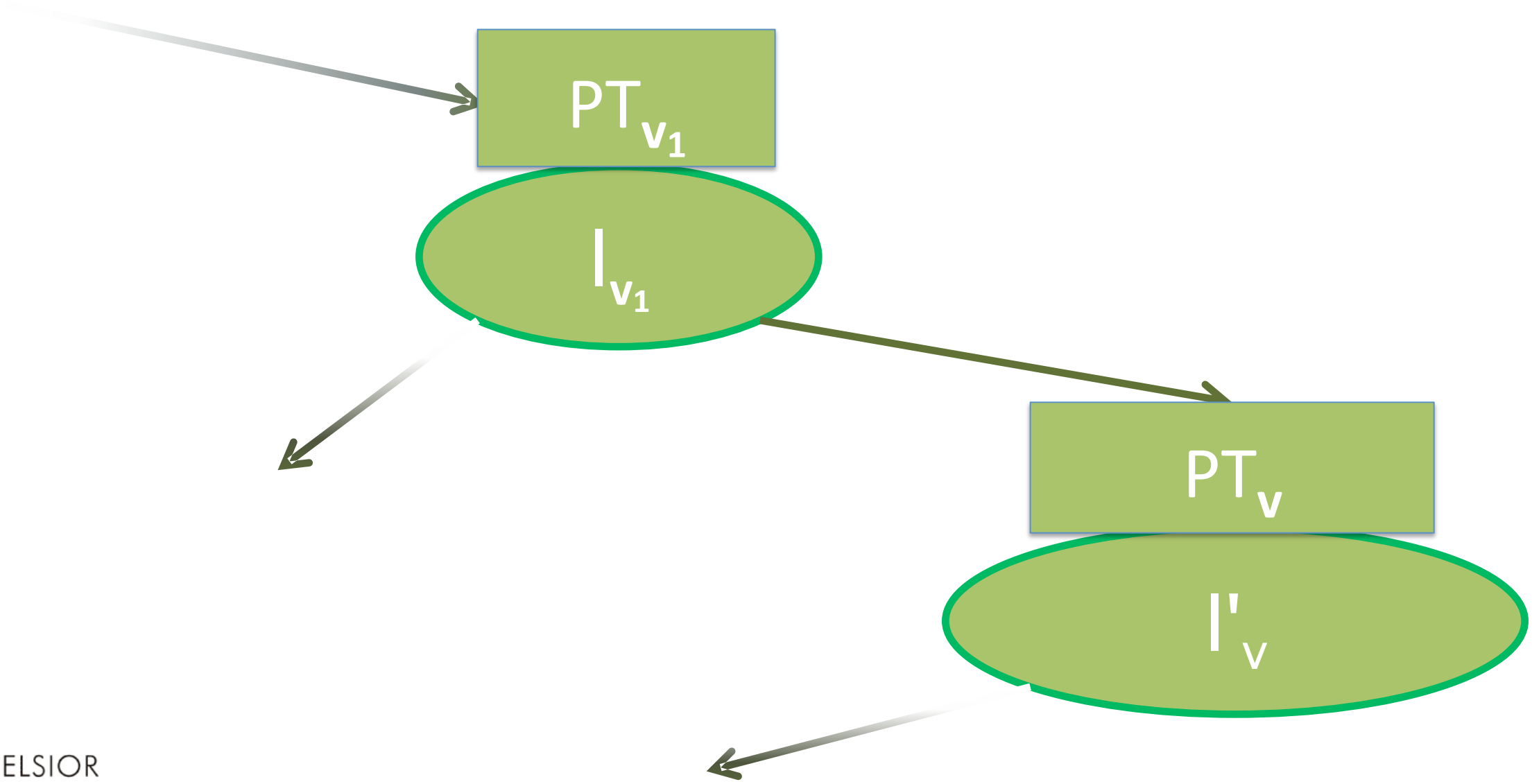
$$MOP(v) = \wedge_{p \in Path(v)} f_p(Init),$$


PT_{v_1}
I_{v_1}
PT_v
I_v


PT_{v_1}
I_{v_1}
I_{v_1}
PT_v
I_v


PT_{v_1}
I_{v_1}
PT_v I_{v_1}
I_v


PT_{v_1}
I_{v_1}
PT_v
$PT_v(I_{v_1}) \wedge I_v$



EXCELSIOR

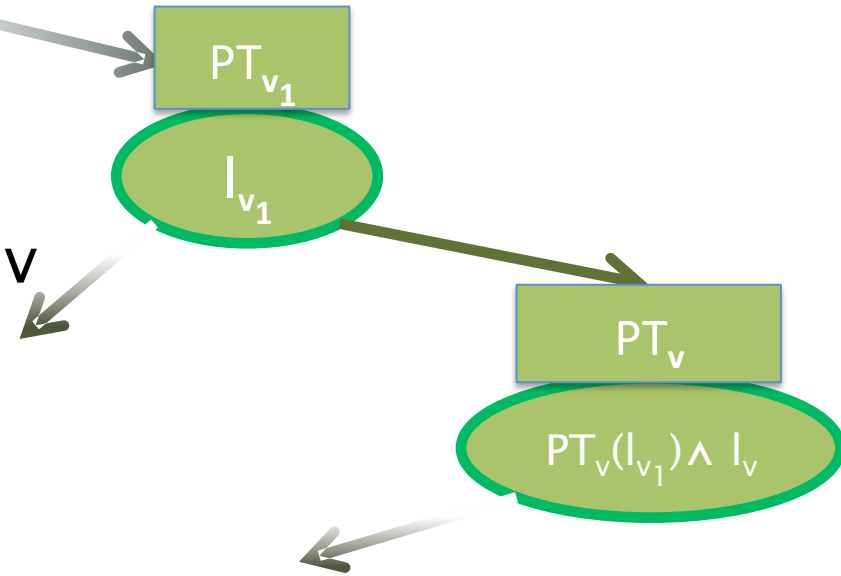
Приближенное решение (стационарная разметка – MFP)

Правило разметки (ПР):

Пусть $l_v \in L$ – текущая пометка вершины v,

$(v_1, v) \in E$

$l'_v =_{def} PT_v(l_{v_1}) \wedge l_v$ – следующая пометка вершины v



Достижимая разметка :

M: V → L: получается из Init применением *(ПР)

Стационарная разметка (MFP):

Достижимая разметка не меняющаяся после любого применения ПР

Свойства MFP

1. Алгоритм MFP сходится
2. Всегда дает один и тот же результат не зависимо от порядка обхода графа
3. MFP(v) $\leq$ MOP(v), $\forall v \in V$ (безопасность/ корректность)

Свойства MFP
Теорема Килдалла

Let DF $=<G, Env = <L,\wedge,F>, Init, PT>$:

$\forall f \in F \Rightarrow$ f – дистрибутивная(!), тогда:

$$MFP(v) = MOP(v), \forall v \in V$$

(оптимальность)

Верификация байткода методом вывода типов (type inference verifier)

EXCELSIOR

Верификация байткода методом вывода типов

Type inference verifier – потоковый анализатор:

- Потоковый граф – управляющий граф (CFG)
- Решетка свойств – множество StackMapFrame
- Преобразователи свойств определены для каждой инструкции байткода, согласно их семантики

Stack Map Frame

Stack Map Frame – свойство, вычисляемое для каждой инструкции данного байткода метода:

- Stack: глубина и типы значений на стеке

Stack Map Frame

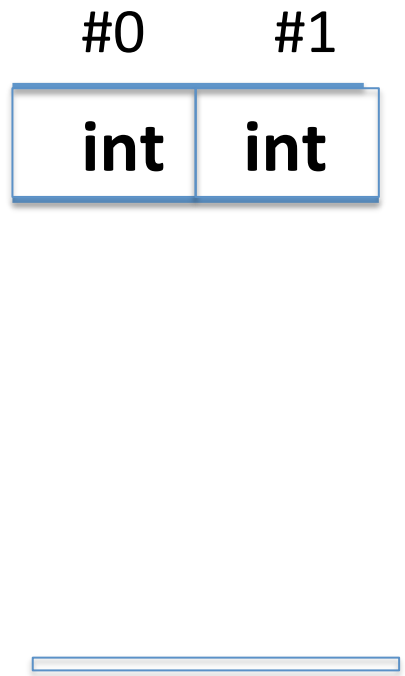
Stack Map Frame – свойство, вычисляемое для каждой инструкции данного байткода метода:

- Stack: глубина и типы значений на стеке
- Locals: типы локальных переменных в регистре локальных переменных

Пример

```
public static int add(int a, int b) { return a + b; }
```

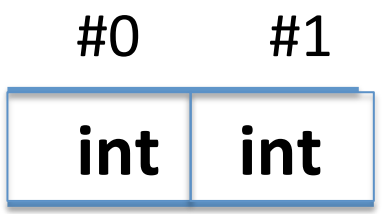
```
0: iload 0
2: iload 1
4: iadd
5: ireturn
```



Пример

```
public static int add(int a, int b) { return a + b; }
```

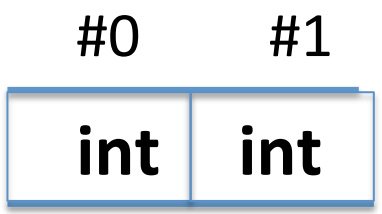
```
0: iload 0
2: iload 1
4: iadd
5: ireturn
```



Пример

```
public static int add(int a, int b) { return a + b; }
```

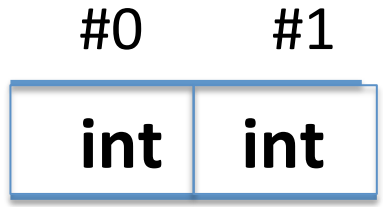
```
0: iload 0
2: iload 1
4: iadd
5: ireturn
```



Пример

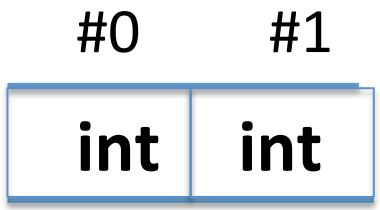
```
public static int add(int a, int b) { return a + b; }
```

```
0: iload 0
2: iload 1
4: iadd
5: ireturn
```



Пример

```
public static int add(int a, int b) { return a + b; }
```

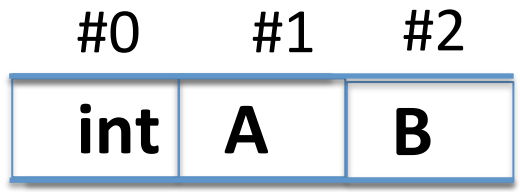


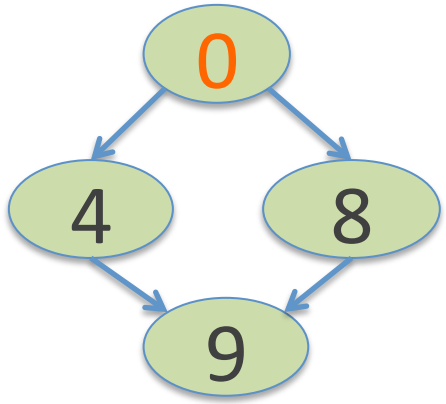

```
0: iload 0
2: iload 1
4: iadd
5: ireturn
```

Пример 2

```
class A extends C{}; class B extends C{};
static C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```

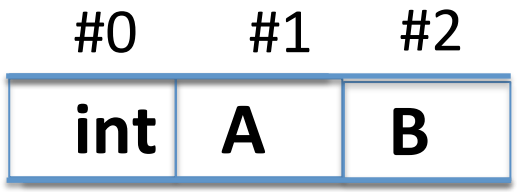


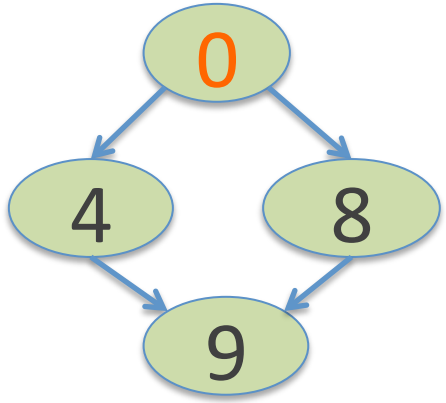


Пример 2

```
class A extends C{}; class B extends C{};
static C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```



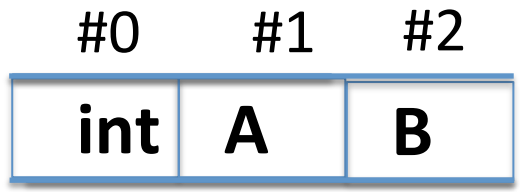


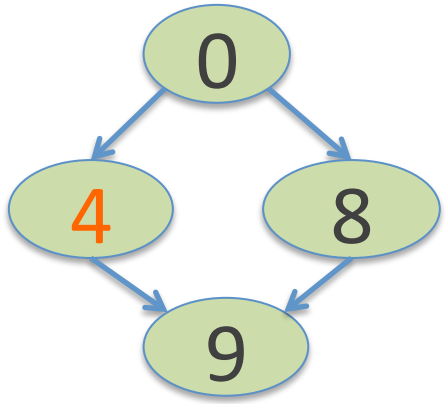
EXCELSIOR

Пример 2

```
class A extends C{}; class B extends C{};
static C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```

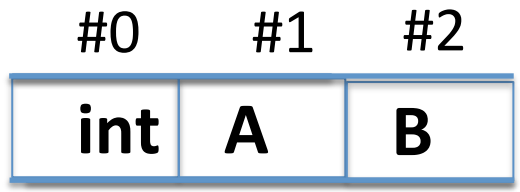


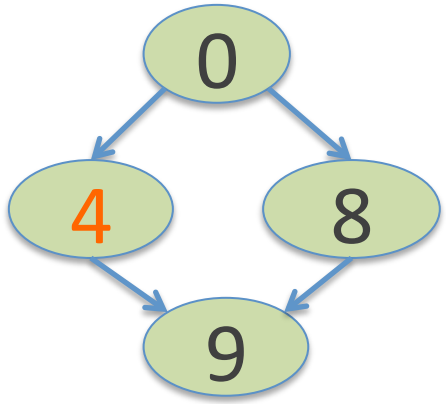


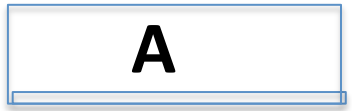
Пример 2

```
class A extends C{}; class B extends C{};
static C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```



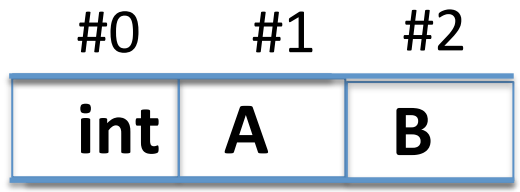


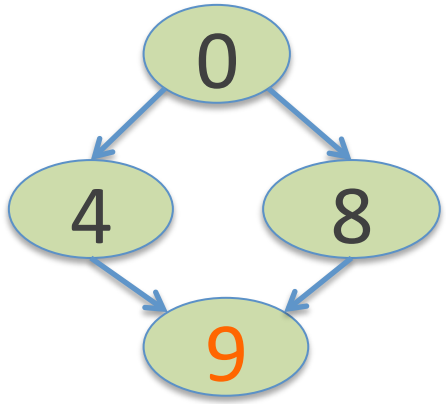


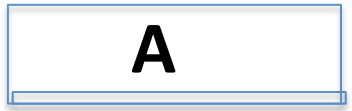
Пример 2

```
class A extends C{}; class B extends C{};
static C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```



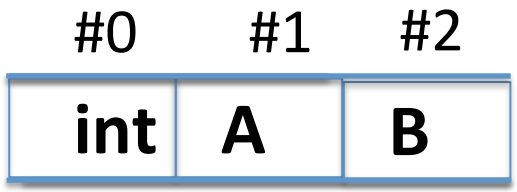


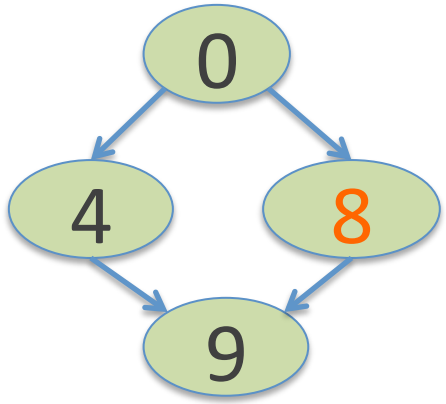


Пример 2

```
class A extends C{}; class B extends C{};
static C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```

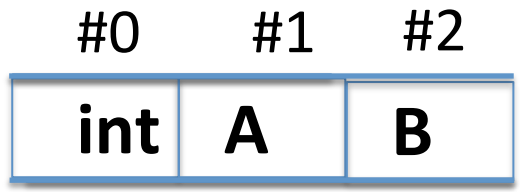


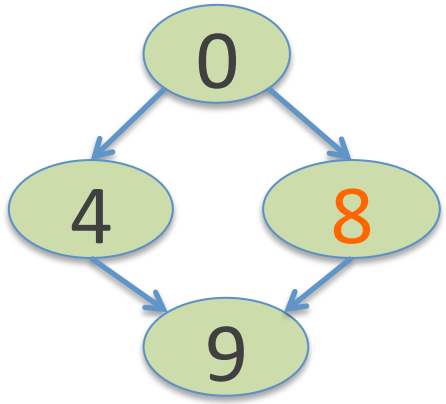


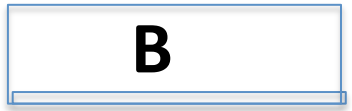
Пример 2

```
class A extends C{}; class B extends C{};
public C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```



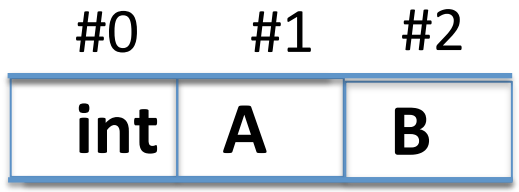


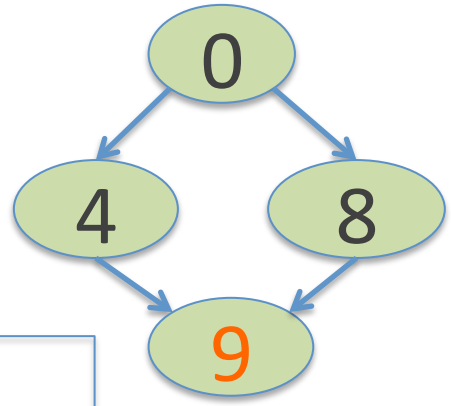


Пример 2

```
class A extends C{}; class B extends C{};
static C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```

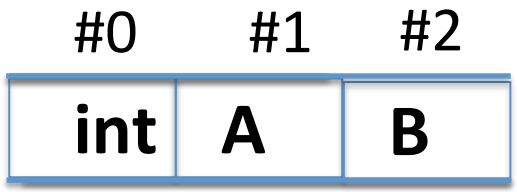


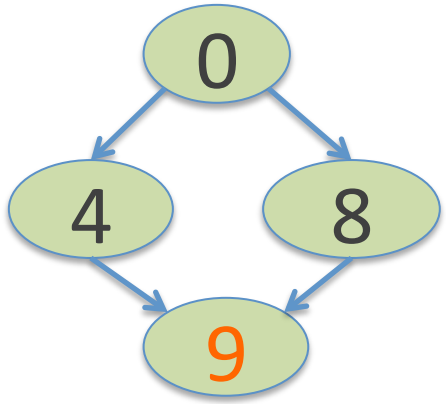


Пример 2

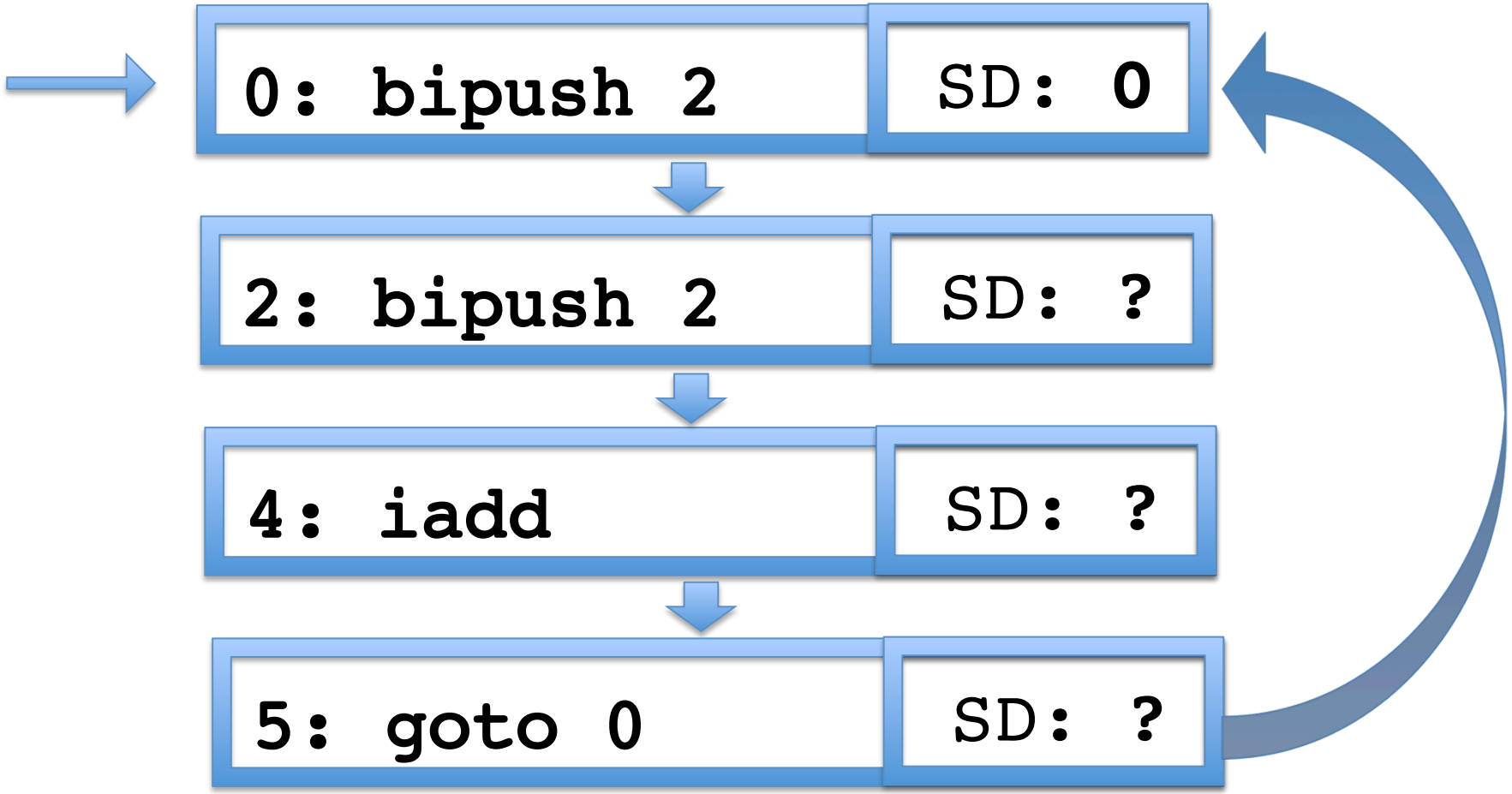
```
class A extends C{}; class B extends C{};
static C ifcond(boolean cond, A a, B b) {return cond?a:b;}
```

```
 0: iload_0
 1: ifeq 8
 4: aload_1
 5: goto 9
 8: aload_2
 9: areturn
```





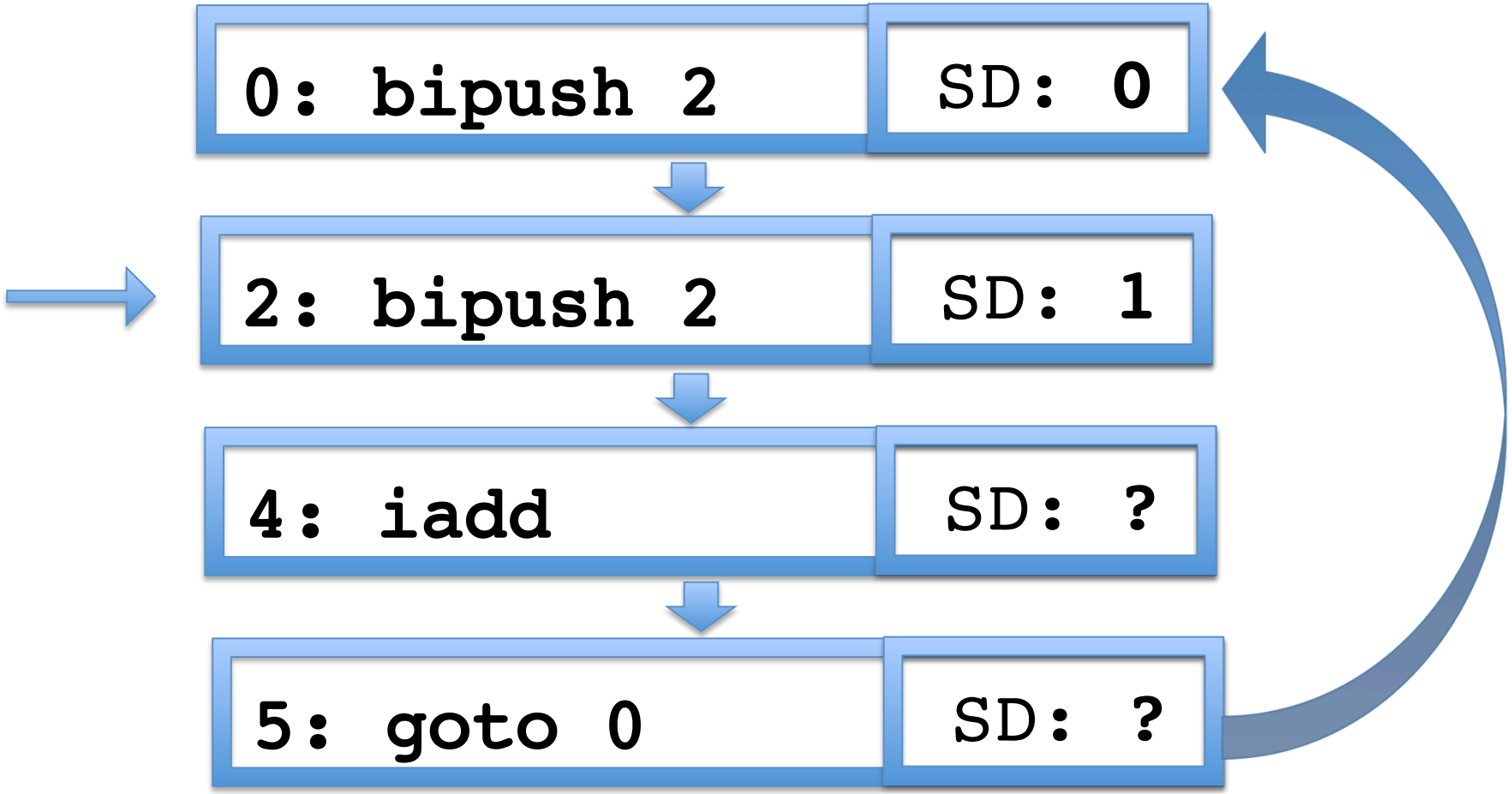
Пример 3



SD: Stack Depth

EXCELSIOR

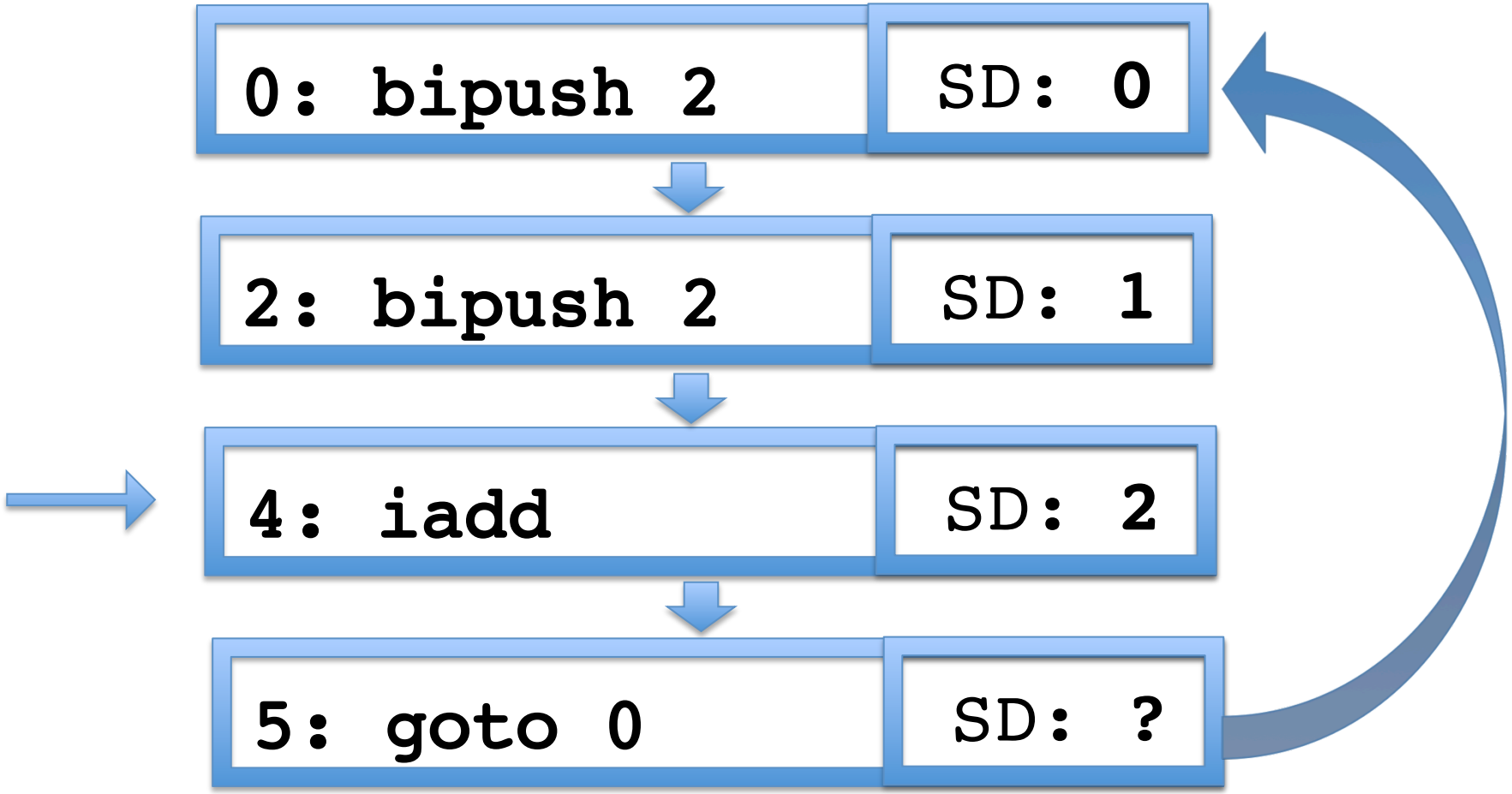
Пример 3



SD: Stack Depth

EXCELSIOR

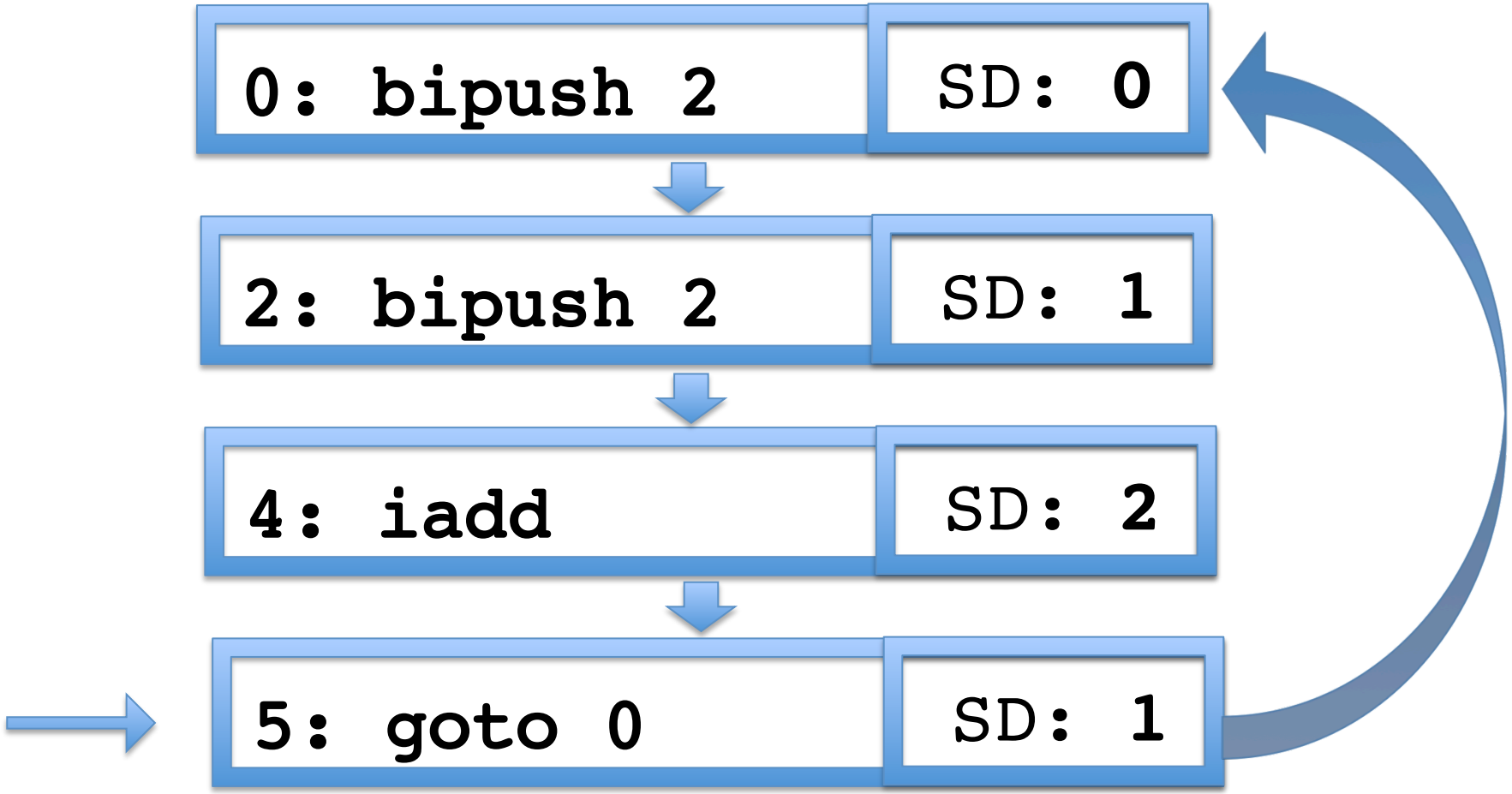
Пример 3



SD: Stack Depth

EXCELSIOR

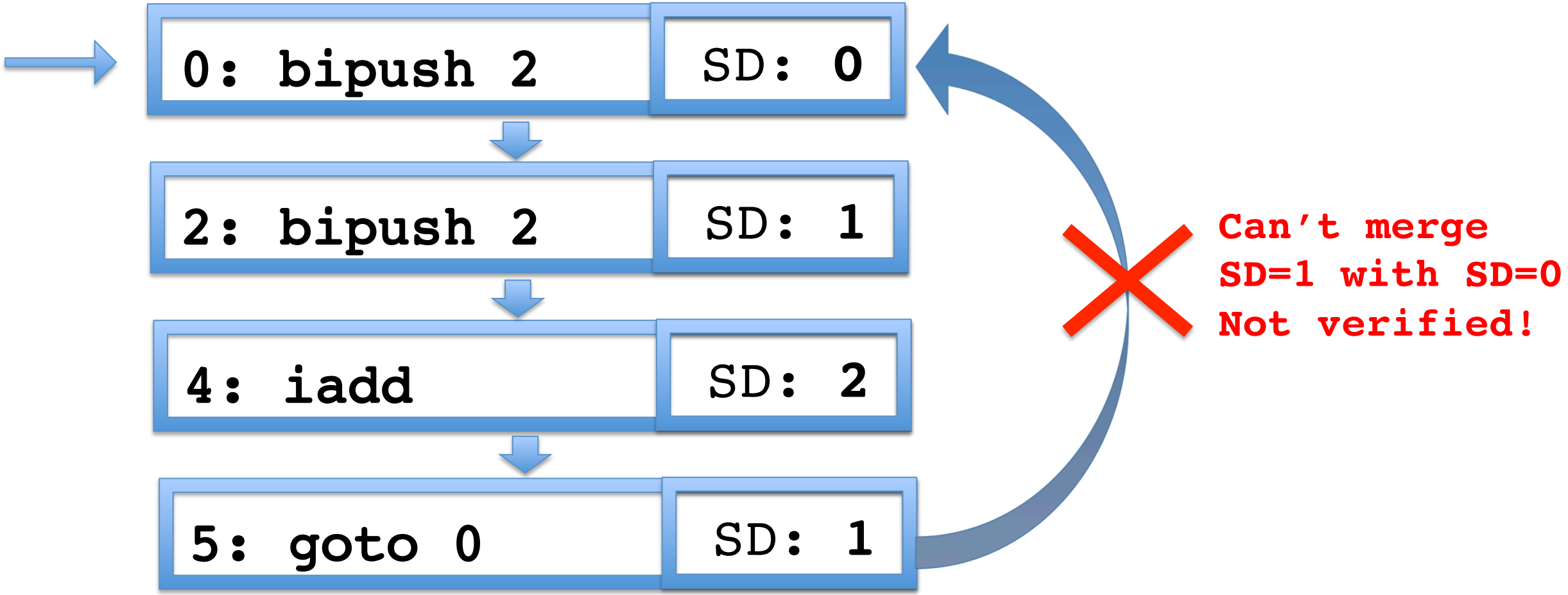
Пример 3



SD: Stack Depth

EXCELSIOR

Пример 3



EXCELSIOR

Верификация байткода методом вывода типов

Каждая инструкция байткода определяет преобразователь свойств:

F(opcode): StackMapFrame -> StackMapFrame

EXCELSIOR

Верификация байткода методом вывода типов

Эти преобразователи дистрибутивны!

EXCELSIOR

Верификация байткода методом вывода типов

Эти преобразователи дистрибутивны!

Значит верификатор строит точное решение задачи верификации за конечное время без исполнения байткода!

EXCELSIOR

Верификация байткода методом вывода типов

Эти преобразователи дистрибутивны!

Значит верификатор строит точное решение задачи верификации за конечное время без исполнения байткода!

Но за какое время?

EXCELSIOR

Оценка сложности алгоритма MFP

Утверждение:

Если обходить потоковый граф в глубину, то временная сложность алгоритма MFP (A_{MFP}):

$$T(A_{MFP}) = O(B_L * A * |V|),$$

Где A – количество обратных дуг в потоковом графе

$A = O(|V|)$, на практике $A \leq 3$

Верификация байткода методом вывода типов

Время работы верификатора зависит от:

- Количества инструкций
- Количества циклов
- Количества локальных переменных
- Максимальной глубины стека операндов
- Высоты иерархии классов

EXCELSIOR

Верификация байткода методом вывода типов

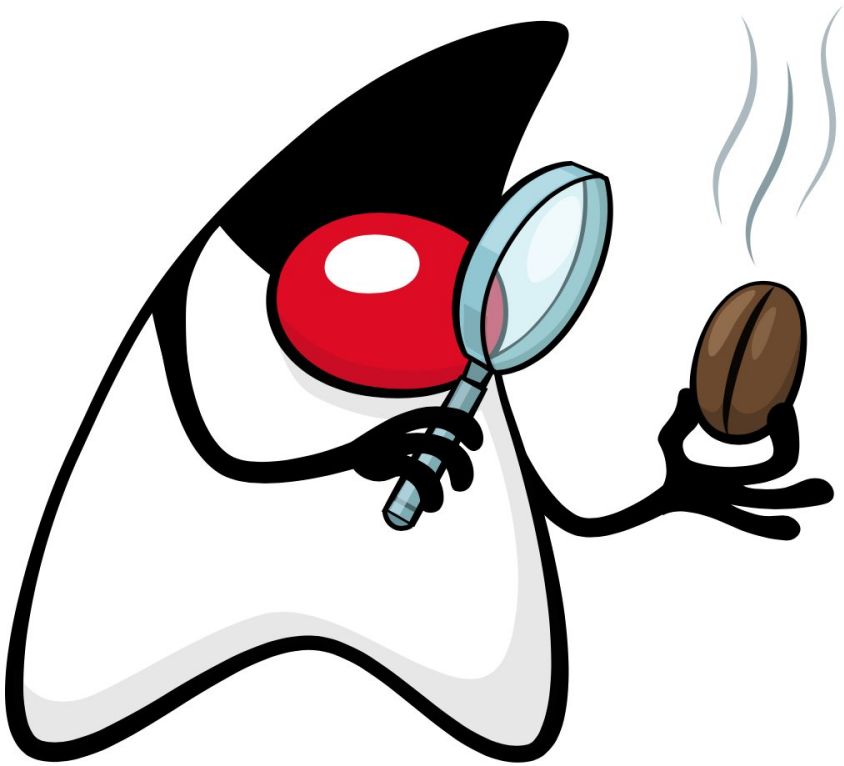
Время работы верификатора зависит от:

- Количества инструкций
- Количества циклов
- Количества локальных переменных
- Максимальной глубины стека операндов
- Высоты иерархии классов

Дорого?

EXCELSIOR

Верификация байткода методом проверки типов (type checking verifier)



EXCELSIOR

Верификация байткода методом проверки типов

Вопрос: Как ускорить верификатор?

EXCELSIOR

Верификация байткода методом проверки типов

Вопрос: Как ускорить верификатор?

Идея: А что если мы знаем MFP решение верификатора методом вывода типов?

EXCELSIOR

Верификация байткода методом проверки типов

Вопрос: Как ускорить верификатор?

Идея: А что если мы знаем MFP решение верификатора методом вывода типов?

Bingo! Тогда нам достаточно за один проход по байткоду проверить, что неподвижная точка (MFP) действительно неподвижная!

(не меняется при применении преобразователей свойств инструкций)

EXCELSIOR

Верификация байткода методом проверки типов

Split Verifier aka Type checking verifier:

- У байткода версии Java 6 может быть атрибут StackMapTable (с Java 7 обязан быть)

Верификация байткода методом проверки типов

Split Verifier aka Type checking verifier:

- У байткода версии Java 6 может быть атрибут StackMapTable (с Java 7 обязан быть)
- StackMapTable – это список StackMapFrame

Верификация байткода методом проверки типов

Split Verifier aka Type checking verifier:

- У байткода версии Java 6 может быть атрибут StackMapTable (с Java 7 обязан быть)
- StackMapTable – это список StackMapFrame
- StackMapFrame есть для каждой начальной инструкции линейных участков байткода, кроме первого

Верификация байткода методом проверки типов

Split aka Type checking verifier за один проход по байткоду:

- вычисляет недостающие StackMapFrame: для каждой инструкции *проверяет входящие типы*, вычисляет исходящие (переносит ниже)

Верификация байткода методом проверки типов

Split aka Type checking verifier за один проход по байткоду:

- вычисляет недостающие StackMapFrame: для каждой инструкции *проверяет входящие типы*, вычисляет исходящие (переносит ниже)

- проверяет что StackMapFrame из StackMapTable атрибута не меняются при переносе StackMapFrame из инструкций выше (*проверка неподвижности точки*)

EXCELSIOR

Роль верификатора байткода в JVM

EXCELSIOR

Роль верификатора байткода в JVM

Вопросы:

1. Может ли верификатор байткода замедлить исполнение ?
2. Может ли верификатор байткода замедлить старт?
3. Что было бы, если бы не было верификатора байткода?
4. А что будет, если его отключить?

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение?

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение?

А исполнение чего?

EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение?

А исполнение чего?

- Исполнение байткода метода
- Исполнение горячего кода программы
- Исполнение программы

EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **байткода метода**?

EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **байткода метода**?

Вспомним: Верификатор работает до исполнения байткода!

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **байткода метода**?

Вспомним: Верификатор работает до исполнения байткода!

Ответ: Нет!

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **программы**?

EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **программы**?

Верификатор работает во время загрузки класса, загрузка классов работает во время исполнения программы

EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **программы**?

Верификатор работает во время загрузки класса, загрузка классов работает во время исполнения программы

Ответ: вообще говоря, да

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **горячего кода**?

EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **горячего кода**?

Во время исполнения горячего кода, все классы, с которым он работает, были загружены пока код был холодным.

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить исполнение **горячего кода**?

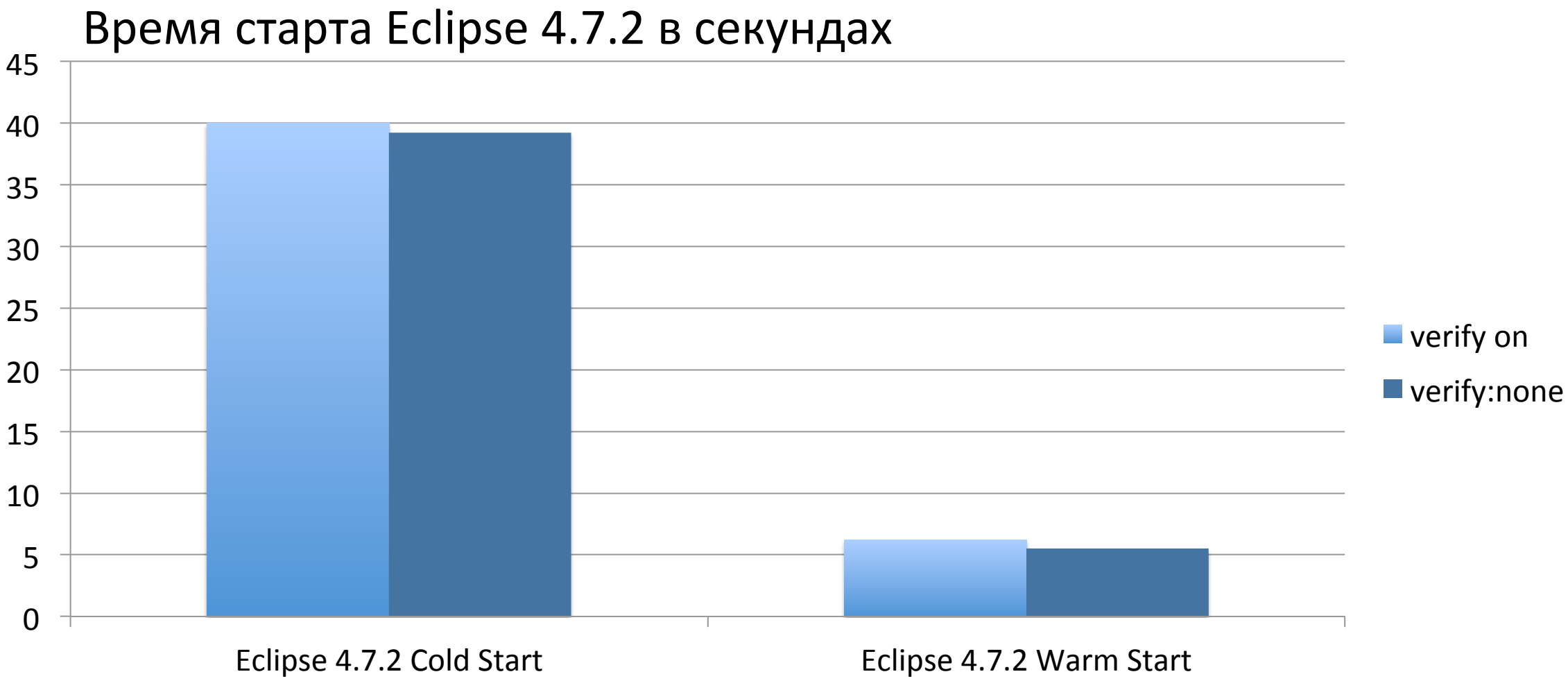
Во время исполнения горячего кода, все классы, с которым он работает, были загружены пока код был холодным.

Ответ: нет

Роль верификатора байткода в JVM

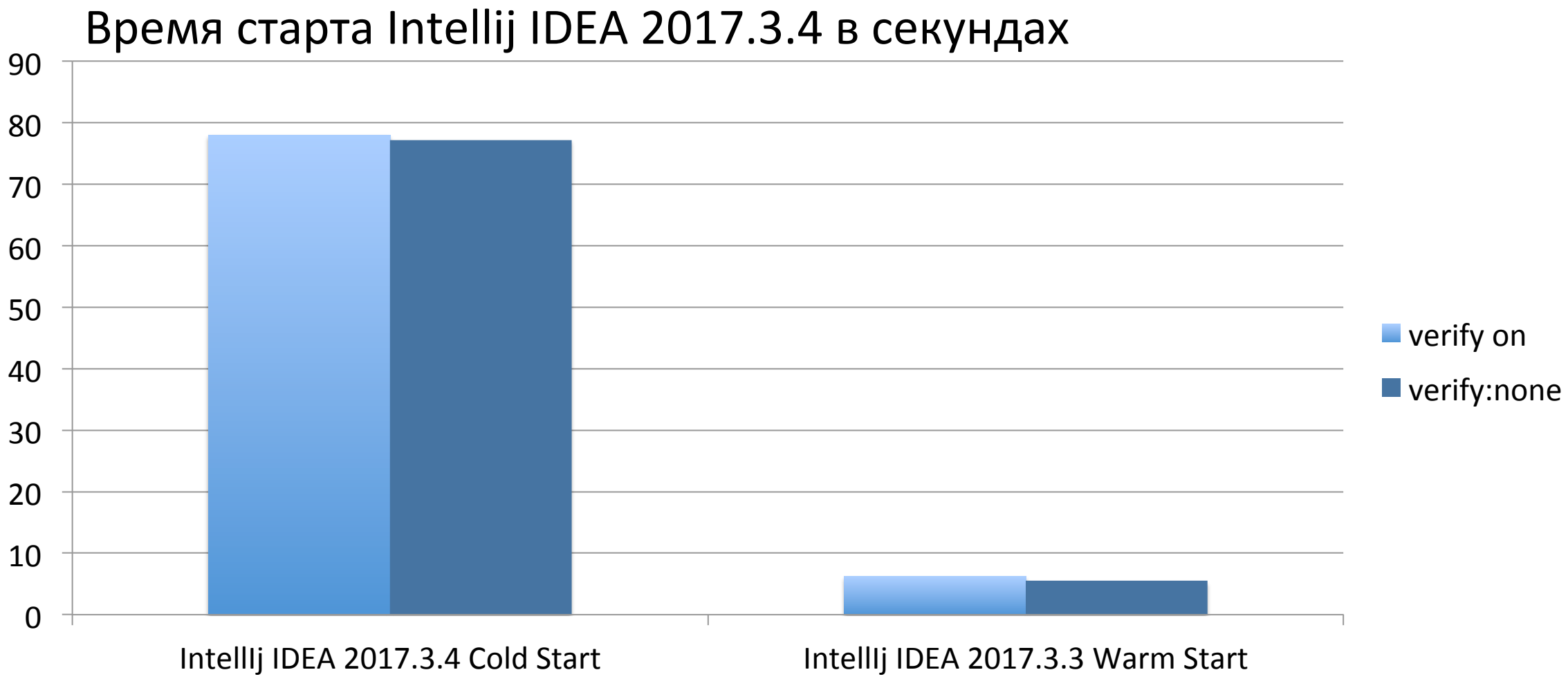
Вопрос: Может ли верификатор байткода замедлить старт программы?

Верификатор и время старта



EXCELSIOR

Верификатор и время старта



EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить старт программы?

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить старт программы?

Ответ: незначительно

Роль верификатора байткода в JVM

Вопрос: Может ли верификатор байткода замедлить старт программы?

Ответ: незначительно

У JVM с AOT компилятором верификация на старте может не происходить

Роль верификатора байткода в JVM

Вопрос: Что было бы, если бы не было верификатора?

EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: Что было бы, если бы не было верификатора?

Ответ: пришлось бы вставлять проверки во время исполнения:

Роль верификатора байткода в JVM

Вопрос: Что было бы, если бы не было верификатора?

Ответ: пришлось бы вставлять проверки во время исполнения:

- для инструкций работающих со стеком, проверять что стек не пуст и не переполнится

Роль верификатора байткода в JVM

Вопрос: Что было бы, если бы не было верификатора?

Ответ: пришлось бы вставлять проверки во время исполнения:

- для инструкций работающих со стеком, проверять что стек не пуст и не переполнится
- для того, чтобы сложить два числа проверять, что на стеке лежат два числа

Роль верификатора байткода в JVM

Вопрос: Что было бы, если бы не было верификатора?

Ответ: пришлось бы вставлять проверки во время исполнения:

- для инструкций работающих со стеком, проверять что стек не пуст и не переполнится
- для того, чтобы сложить два числа проверять, что на стеке лежат два числа

Как бы быстро тогда работали Java программы?

EXCELSIOR

Роль верификатора байткода в JVM

Факт: Интерпретатор не проверяет выходы за пределы стека операндов и типовую совместимость: если надо сложить два числа, он просто берет их со стека и складывает.

Роль верификатора байткода в JVM

Факт: Интерпретатор не проверяет выходы за пределы стека операндов и типовую совместимость: если надо сложить два числа, он просто берет их со стека и складывает.

Факт: JIT компилятор при генерации машинного кода тоже не вставляет такие проверки в машинный код.

Роль верификатора байткода в JVM

Факт: Интерпретатор не проверяет выходы за пределы стека операндов и типовую совместимость: если надо сложить два числа, он просто берет их со стека и складывает.

Факт: JIT компилятор при генерации машинного кода тоже не вставляет такие проверки в машинный код.

Вывод: Наличие верификатора – первопричина почему JVM исполняет Java программы **быстро**!

Роль верификатора байткода в JVM

Вопрос: А что будет, если верификатор отключить?

Роль верификатора байткода в JVM

Вопрос: А что будет, если верификатор отключить?

```
# A fatal error has been detected by the Java Runtime Environment:
Exception in thread "main" java.lang.StackOverflowError
        at test.foo(test.j)
        at test.main(test.j:3)
#
#  Internal Error (javaCalls.cpp:53), pid=8012, tid=16396
#  guarantee(!thread->is_Compiler_thread()) failed: cannot make java calls
from the compiler
#
# If you would like to submit a bug report, please visit:
#   http://bugreport.java.com/bugreport/crash.jsp
```

EXCELSIOR

Роль верификатора байткода в JVM

Вопрос: А что будет, если верификатор отключить?

Ответ: Отключение верификатора может приводить к развалам JVM, что в свою очередь открывает потенциальные дыры в безопасности.

Роль верификатора байткода в JVM

Вопрос: А что будет, если верификатор отключить?

Ответ: Отключение верификатора может приводить к развалам JVM, что в свою очередь открывает потенциальные дыры в безопасности.

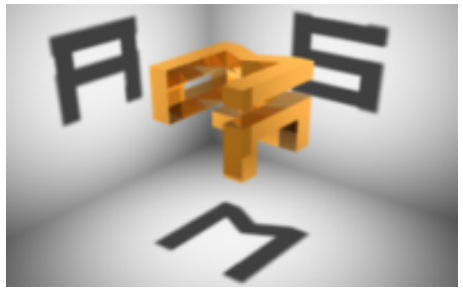
Вывод: Наличие верификатора делает исполнение байткода не только эффективным, но и **безопасным**!

Верификатор и библиотеки генерации байткода

EXCELSIOR

Библиотеки генерации байткода

- Низкоуровневые
  - ASM
  - BCEL



- Высокоуровневые
  - Javassist
  - cglib
  - ByteBuddy

ASM библиотека

ASM библиотека

- Позволяет полностью контролировать содержимое выходных классов, байткода

EXCELSIOR

ASM библиотека

ASM библиотека

- Позволяет полностью контролировать содержимое выходных классов, байткода
- Как следствие низкоуровневая: вы работаете непосредственно с инструкциями байткода

ASM библиотека

ASM: пример кода

```
MethodVisitor methodVisitor = … ;
methodVisitor.visitVarInsn(Opcodes.ILOAD, 3);
methodVisitor.visitIntInsn (Opcodes.BIPUSH, 5);
methodVisitor.visitInsn    (Opcodes.IADD);
methodVisitor.visitVarInsn(Opcodes.ISTORE, 4);
```

ASM библиотека

- Нужно знать семантику байткодных инструкций

ASM библиотека

- Нужно знать семантику байткодных инструкций
- Легко породить неверифицируемый байткод

EXCELSIOR

ASM библиотека

- Нужно знать семантику байткодных инструкций
- Легко породить неверифицируемый байткод
- ASM умеет вычислять StackMapTable атрибут
  - с помощью того же потокового анализа

ASM библиотека

- Нужно знать семантику байткодных инструкций
- Легко породить неверифицируемый байткод
- ASM умеет вычислять StackMapTable атрибут
  - с помощью того же потокового анализа
- Можно задавать StackMapTable атрибут руками
  - для скорости; автоматическое вычисление замедляет ASM в 2 раза

Библиотеки генерации байткода

Проблема: некоторые (старые) библиотеки, манипулирующие байткодом, не умеют вычислять StackMapFrame

Библиотеки генерации байткода

Проблема: некоторые (старые) библиотеки, манипулирующие байткодом, не умеют вычислять StackMapFrame

В результате вы можете получить ошибку:

java.lang.VerifyError: Expecting a *stackmap frame* at branch

EXCELSIOR

Библиотеки генерации байткода

Проблема: некоторые (старые) библиотеки, манипулирующие байткодом, не умеют вычислять StackMapFrame

Можно отключить Type Checking Verifier: **-XX:-UseSplitVerifier**

Библиотеки генерации байткода

Проблема: некоторые (старые) библиотеки, манипулирующие байткодом, не умеют вычислять StackMapFrame

Можно отключить Type Checking Verifier: **-XX:-UseSplitVerifier**

Тогда будет работать Type Inference Verifier

EXCELSIOR

Библиотеки генерации байткода

Проблема: некоторые (старые) библиотеки, манипулирующие байткодом, не умеют вычислять StackMapFrame

Можно отключить Type Checking Verifier: **-XX:-UseSplitVerifier**

Тогда будет работать Type Inference Verifier

- Развал JVM не случится

EXCELSIOR

Библиотеки генерации байткода

Проблема: некоторые (старые) библиотеки, манипулирующие байткодом, не умеет вычислять StackMapFrame

Можно отключить Type Checking Verifier: **-XX:-UseSplitVerifier**

Тогда будет работать Type Inference Verifier

- Но старый верификатор не знает про invokedynamic
- Случится VerifyError при использовании лямбд

EXCELSIOR

Библиотеки генерации байткода

Проблема: некоторые (старые) библиотеки, манипулирующие байткодом, не умеет вычислять StackMapFrame

Решение: не использовать старые библиотеки.

Используя ASM, вычислять StackMapFrame самим или с помощью ASM.

EXCELSIOR

Пример

https://github.com/pjBooms/sample-agent-java-bytecode-verifier

EXCELSIOR

Заключение

Верификатор Java байткода:

Заключение

Верификатор Java байткода:

- Незаслуженно обделенная вниманием часть JVM

EXCELSIOR

Заключение

Верификатор Java байткода:

- Незаслуженно обделенная вниманием часть JVM
- Первопричина почему JVM исполняет программы быстро
  (практически на уровне низкоуровневых языков)

Заключение

Верификатор Java байткода:

- Незаслуженно обделенная вниманием часть JVM
- Первопричина почему JVM исполняет программы быстро
  (практически на уровне низкоуровневых языков)
- Гарантирует безопасность исполнения

Заключение

Верификатор Java байткода:

- Незаслуженно обделенная вниманием часть JVM
- Первопричина почему JVM исполняет программы быстро
  (практически на уровне низкоуровневых языков)
- Гарантирует безопасность исполнения
- Отключать верификатор не имеет никакого практического смысла

Заключение

Верификатор Java байткода:

- Незаслуженно обделенная вниманием часть JVM
- Первопричина почему JVM исполняет программы быстро (практически на уровне низкоуровневых языков)
- Гарантирует безопасность исполнения
- Отключать верификатор не имеет никакого практического смысла
- Зная как работает верификатор, порождать байткод легко

EXCELSIOR

Вопросы и ответы

Никита Липский,
Excelsior

nlipsky@excelsior-usa.com
twitter: @pjBooms

EXCELSIOR