Как сделать встроенный в JVM профайлер, который не боится AOT компиляции?

Иван Углянский
Excelsior LLC

EXCELSIOR

Иван Углянский

- 6 лет в Excelsior
- JVM инженер в проекте Excelsior JET
  - разрабатываю рантайм

iugliansky@excelsior-usa.com

@dbg_nsk

EXCELSIOR

EXCELSIOR

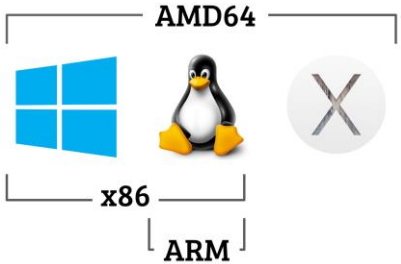
Java™ COMPATIBLE

Excelsior JET 14

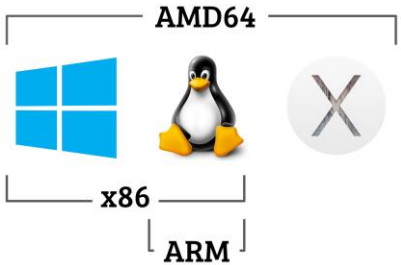
Enterprise Edition

Accelerate, protect, and deploy your Java™ applications.

Excelsior JET JVM



Excelsior JET JVM



Excelsior JET team blog: https://www.excelsiorjet.com/blog

JIT vs AOT

JIT vs AOT

JIT:

- ✓ получение профиля исполнения

EXCELSIOR

JIT vs AOT

JIT:

- ✓ получение профиля исполнения
- ✓ перекомпиляция горячих методов
- ✓ деоптимизации при необходимости

EXCELSIOR

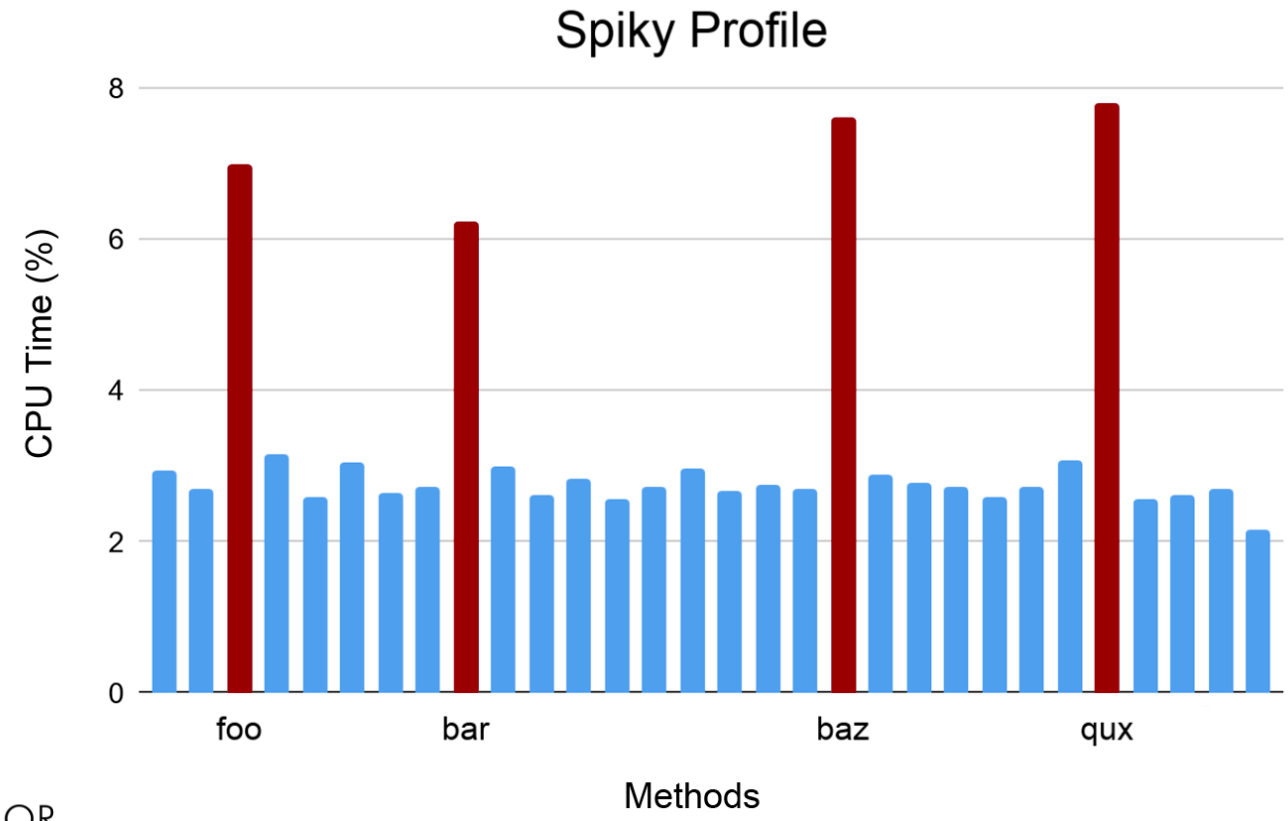
JIT vs AOT

JIT:

- ✓ получение профиля исполнения
- ✓ перекомпиляция горячих методов
- ✓ деоптимизации при необходимости
- ✓ эффективен на профиле с явными пиками

EXCELSIOR

JIT vs AOT



EXCELSIOR

JIT vs AOT

AOT:

EXCELSIOR

JIT vs AOT

AOT:

✓ only one shot

EXCELSIOR

JIT vs AOT

AOT:

- ✓ only one shot
- ✓ но ресурсов для оптимизаций больше

EXCELSIOR

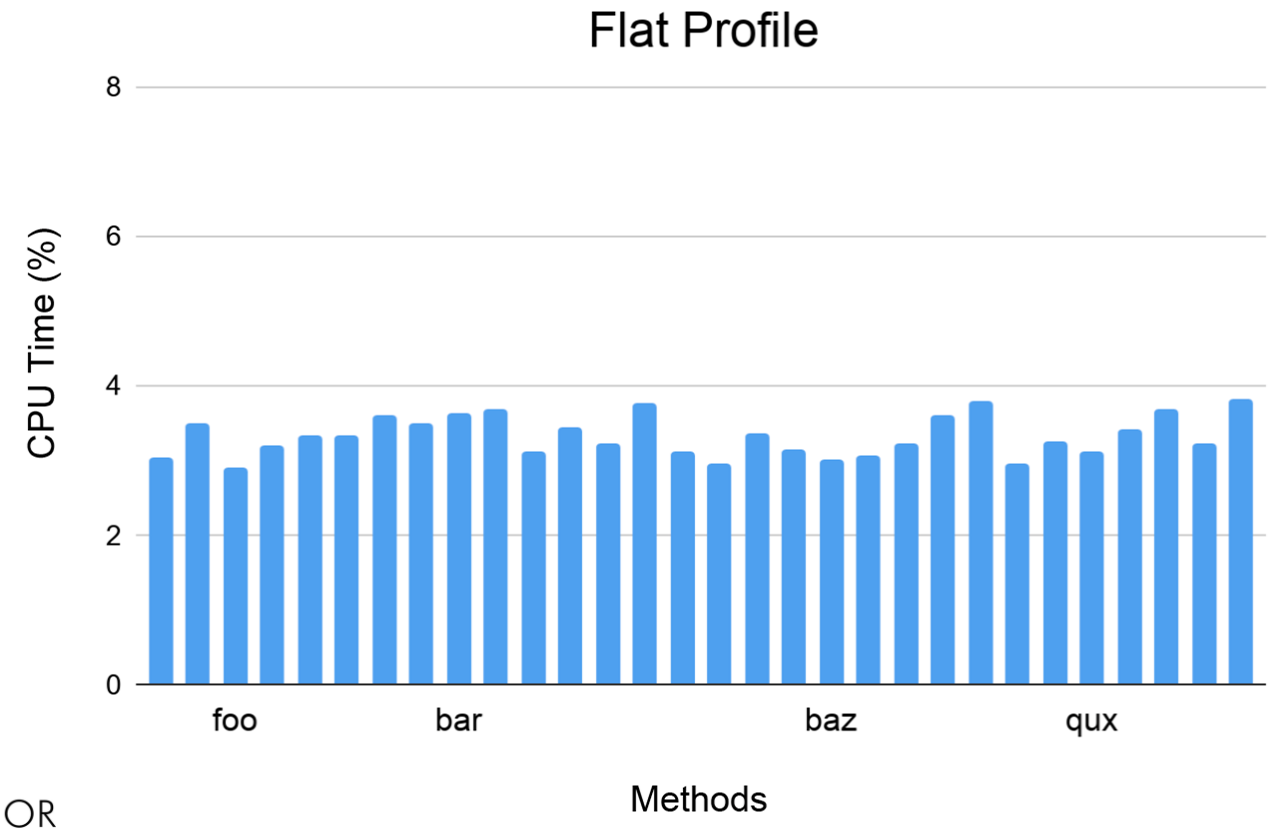
JIT vs AOT

AOT:

- ✓ only one shot
- ✓ но ресурсов для оптимизаций больше
- ✓ эффективен на плоском профиле

EXCELSIOR

JIT vs AOT



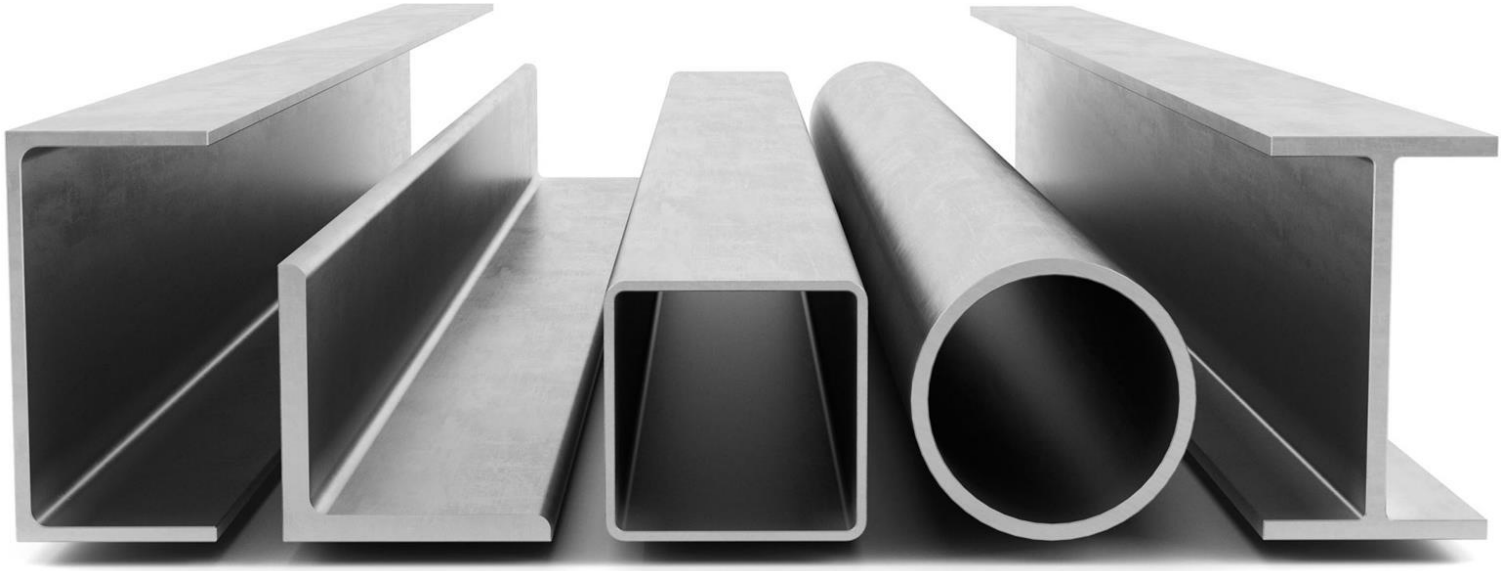
EXCELSIOR

JIT vs AOT

Можно ли взять лучшее из двух миров?

EXCELSIOR

Profile-Guided Optimization



EXCELSIOR

Profile-Guided Optimization

1. компилируем приложение
2. запускаем, собираем профиль

EXCELSIOR

Profile-Guided Optimization

1. компилируем приложение
2. запускаем, собираем профиль
3. перекомпилируем "горячие" части

EXCELSIOR

Profile-Guided Optimization

1. компилируем приложение
2. запускаем, собираем профиль
3. перекомпилируем "горячие" части
4. ULTIMATE PERFORMANCE BOOST

EXCELSIOR

Profile-Guided Optimization

1. компилируем приложение
2. запускаем, собираем профиль
3. перекомпилируем "горячие" части
4. ULTIMATE PERFORMANCE BOOST

Visual Studio

EXCELSIOR

Profile-Guided Optimization

Но как получить профиль?

Profile-Guided Optimization

Требования:

1. Информативность ➢ Горячие методы + call chains

EXCELSIOR

Profile-Guided Optimization

Требования:

1. Информативность ➢ Горячие методы + call chains
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32

EXCELSIOR

Profile-Guided Optimization

Требования:

1. Информативность ➢ Горячие методы + call chains
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
3. Простота ➢ Включать профилировку VM опцией

EXCELSIOR

Profile-Guided Optimization

Требования:

1. Информативность ➢ Горячие методы + call chains
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
3. Простота ➢ Включать профилировку VM опцией
4. Эффективность ➢ Промышленное использование

EXCELSIOR

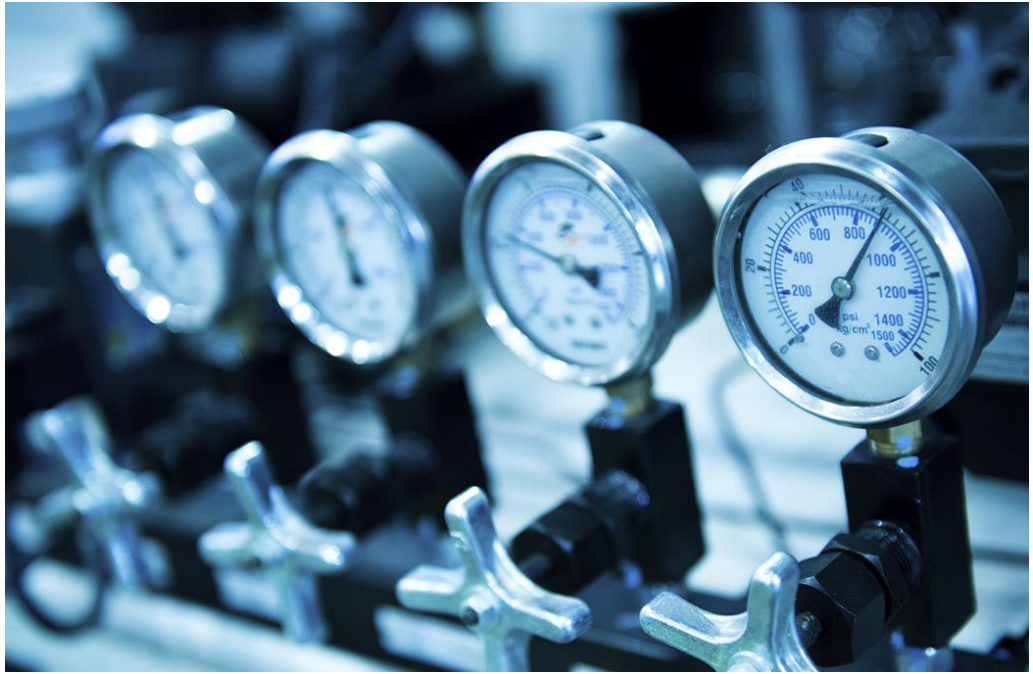
Profile-Guided Optimization

Требования:

1. Информативность ➢ Горячие методы + call chains
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
3. Простота ➢ Включать профилировку VM опцией
4. Эффективность ➢ Промышленное использование

EXCELSIOR

Instrumentation



EXCELSIOR

Instrumentation

В код добавляется учет статистики (циклы, методы, типы)

Точная* информация о профиле

EXCELSIOR

Instrumentation

В код добавляется учет статистики (циклы, методы, типы)

Точная* информация о профиле

EXCELSIOR

Instrumentation

В код добавляется учет статистики (циклы, методы, типы)

Точная* информация о профиле

Инструментировать можно:

- Java bytecode
- Native code

EXCELSIOR

Instrumentation

```
public class Test {
    ...
    public static int testMethod(int counts, int val) {
        int res = 0;
        for (int i = 0; i < counts; i++) {
            res += val;
        }
        return res;
    }
    ...
}
```

```
$ java -XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly Test
```

EXCELSIOR

Instrumentation

```
# {method} {0x0000000016930308} 'testMethod' '(II)I' in 'Test'
# parm0:    rdx       = int
# parm1:    r8        = int
#           [sp+0x40]  (sp of caller)
0x0000000002683b80: mov    DWORD PTR [rsp-0x6000],eax
0x0000000002683b87: push   rbp
0x0000000002683b88: sub    rsp,0x30
0x0000000002683b8c: movabs rax,0x16930560
;   {metadata(method data for {method} ... 'testMethod' ...}
0x0000000002683b96: mov    esi,DWORD PTR [rax+0xdc]
0x0000000002683b9c: add    esi,0x8
0x0000000002683b9f: mov    DWORD PTR [rax+0xdc],esi
0x0000000002683ba5: movabs rax,0x16930300
;   {metadata({method} ... 'testMethod' ... }
...
```

EXCELSIOR

Instrumentation

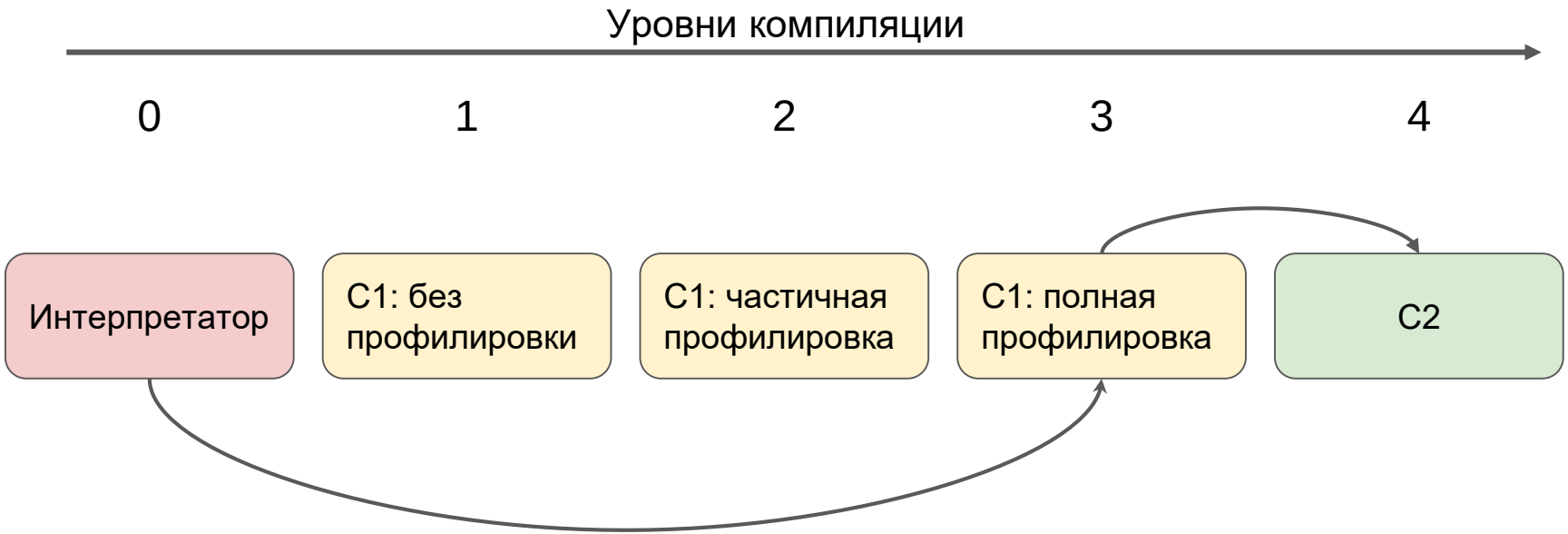
HotSpot C1 Compiler

```
# {method} {0x0000000016930308} 'testMethod' '(II)I' in 'Test'
# parm0:    rdx       = int
# parm1:    r8        = int
#           [sp+0x40]  (sp of caller)
0x0000000002683b80: mov    DWORD PTR [rsp-0x6000],eax
0x0000000002683b87: push   rbp
0x0000000002683b88: sub    rsp,0x30
0x0000000002683b8c: movabs rax,0x16930560
;   {metadata(method data for {method} ... 'testMethod' ...}
0x0000000002683b96: mov    esi,DWORD PTR [rax+0xdc]
0x0000000002683b9c: add    esi,0x8
0x0000000002683b9f: mov    DWORD PTR [rax+0xdc],esi
0x0000000002683ba5: movabs rax,0x16930300
;   {metadata({method} ... 'testMethod' ... }
...
```

инкремент счетчика

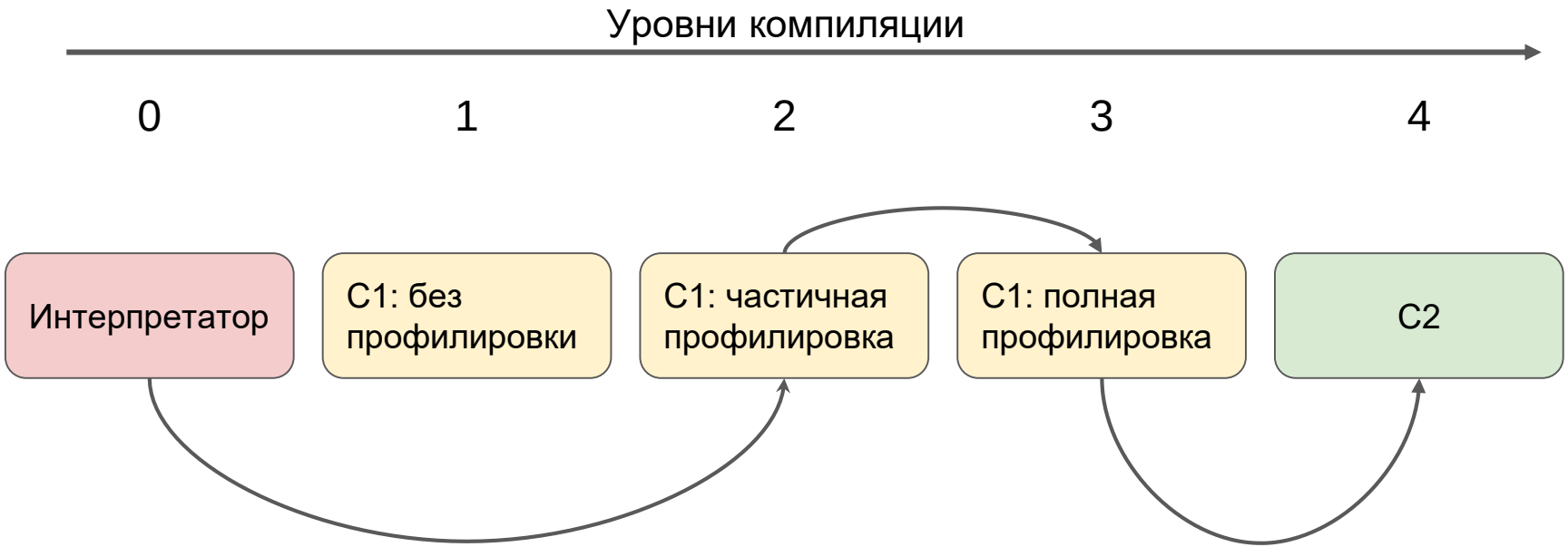
EXCELSIOR

Instrumentation



EXCELSIOR

Instrumentation



EXCELSIOR

Instrumentation

Почему нам в **AOT** не подходит инструментация?

EXCELSIOR

Instrumentation

Почему нам в **AOT** не подходит инструментация?

- Отдельный режим сборки для профилирования
- Инструментация **каждого** метода ⇒
- Плохая производительность

EXCELSIOR

Instrumentation

Требования:

1. Информативность ➢ Горячие методы + call chains
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
3. ~~Простота ➢ Включать профилировку VM опцией~~
4. ~~Эффективность ➢ Промышленное использование~~

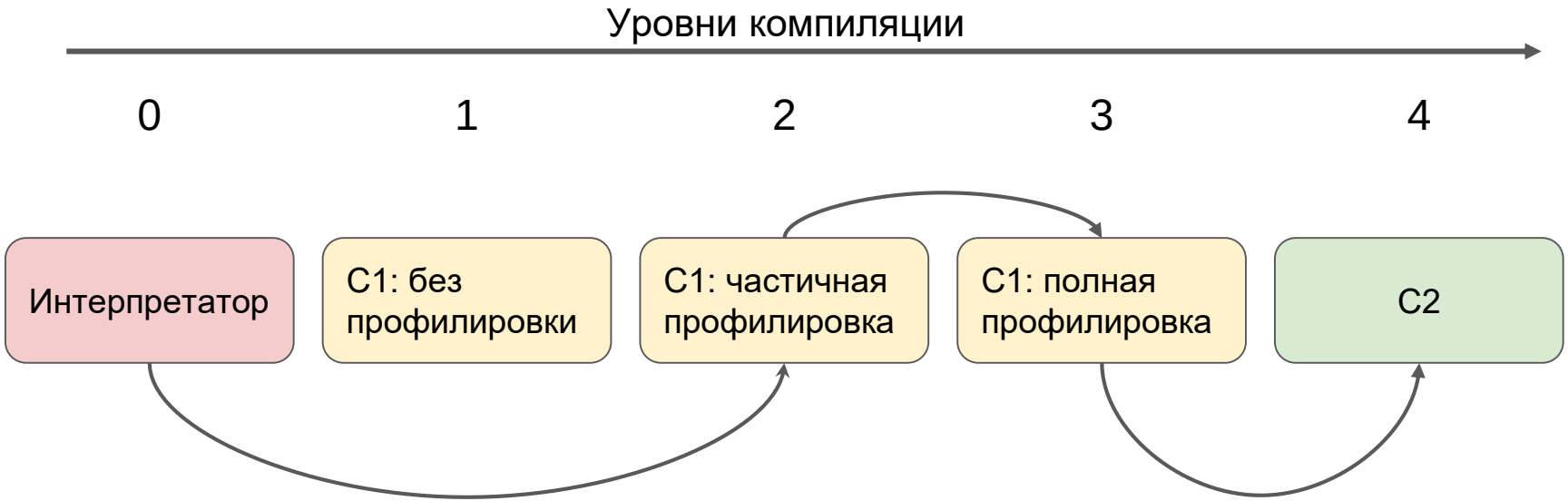
EXCELSIOR

Instrumentation

А как в AOT у HotSpot? (JEP-295)

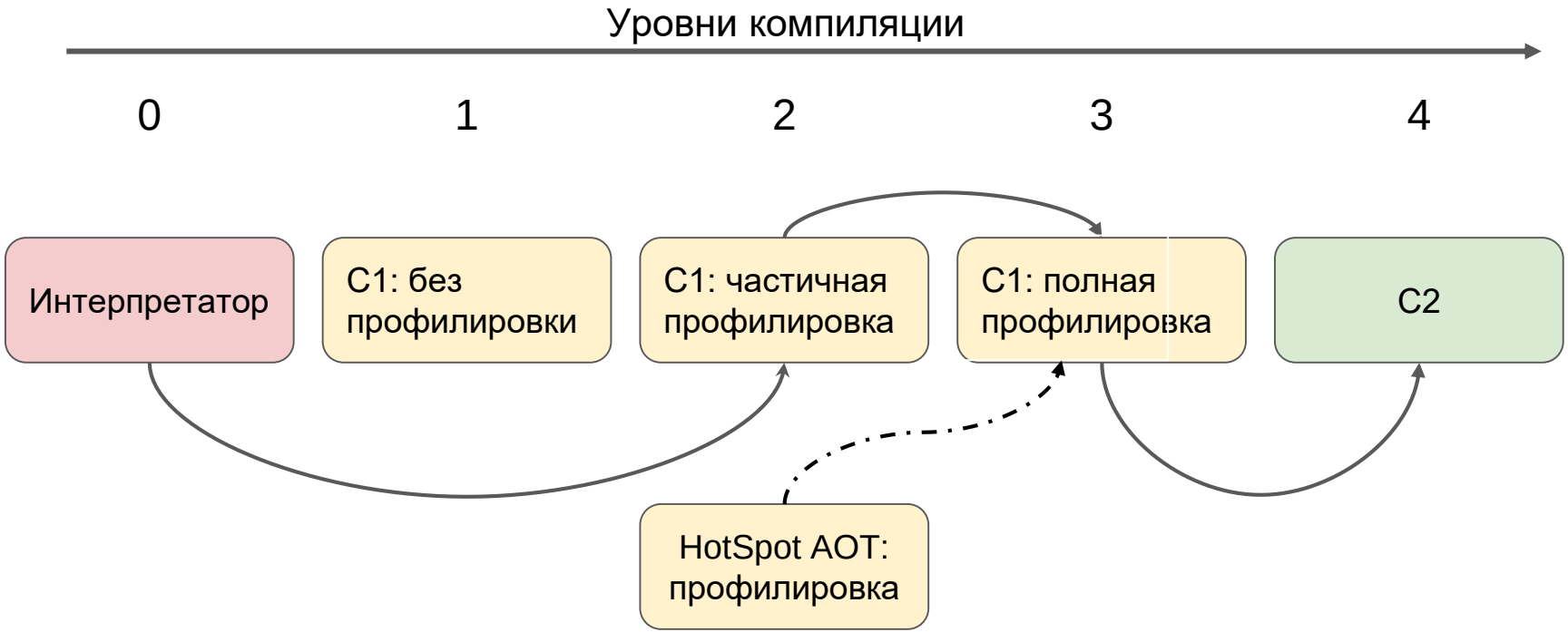
EXCELSIOR

Instrumentation



EXCELSIOR

Instrumentation



Instrumentation

Инструментация в HotSpot AOT:

- Обычная и *вероятностная* профилировка

EXCELSIOR

Instrumentation

Инструментация в HotSpot AOT:

- Обычная и *вероятностная* профилировка
- Вероятностная:
  - ➢ борьба с cache line ping-ponging

EXCELSIOR

Instrumentation

Инструментация в HotSpot AOT:

- Обычная и *вероятностная* профилировка
- Вероятностная:
  - ➢ борьба с cache line ping-ponging
  - ➢ счетчики срабатывают не каждый раз
  - ➢ if (random() & ((1 << prob_log) - 1) == 0) {...}

EXCELSIOR

Instrumentation

Инструментация в HotSpot AOT:

- Обычная и *вероятностная* профилировка
- Вероятностная:
  - ➢ борьба с cache line ping-ponging
  - ➢ счетчики срабатывают не каждый раз
  - ➢ if (random() & ((1 << prob_log) - 1) == 0) {...}

EXCELSIOR

Instrumentation

Требования:

1. Информативность ➢ Горячие методы + call chains
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
3. ~~Простота ➢ Включать профилировку VM опцией~~
4. ~~Эффективность ➢ Промышленное использование~~

EXCELSIOR

Sampling

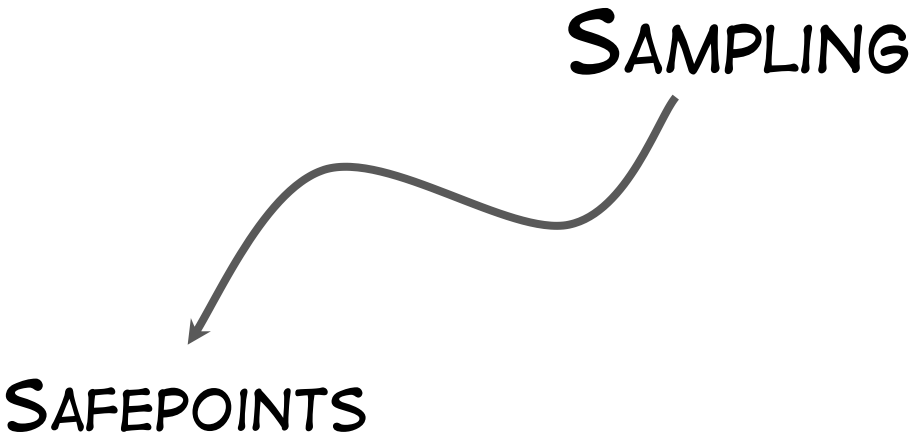
EXCELSIOR

Sampling

Раз в заданный промежуток времени проводится инспекция

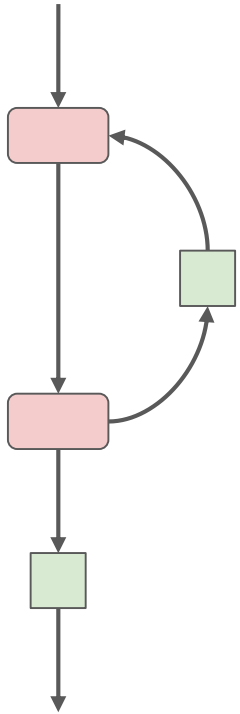
Качество профиля зависит от интервала семплирования, меньше — лучше

EXCELSIOR


SAMPLING
SAFEPOINTS

Safepoints

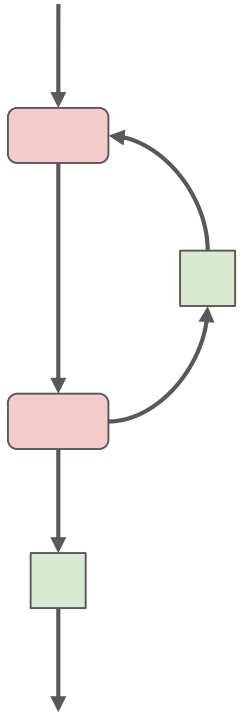
- Точки в коде, где доступен стек вызовов
- В них можно приостановить потоки



EXCELSIOR

Safepoints

- Точки в коде, где доступен стек вызовов
- В них можно приостановить потоки
- Достигаются (относительно) быстро
- Ухудшают производительность



EXCELSIOR

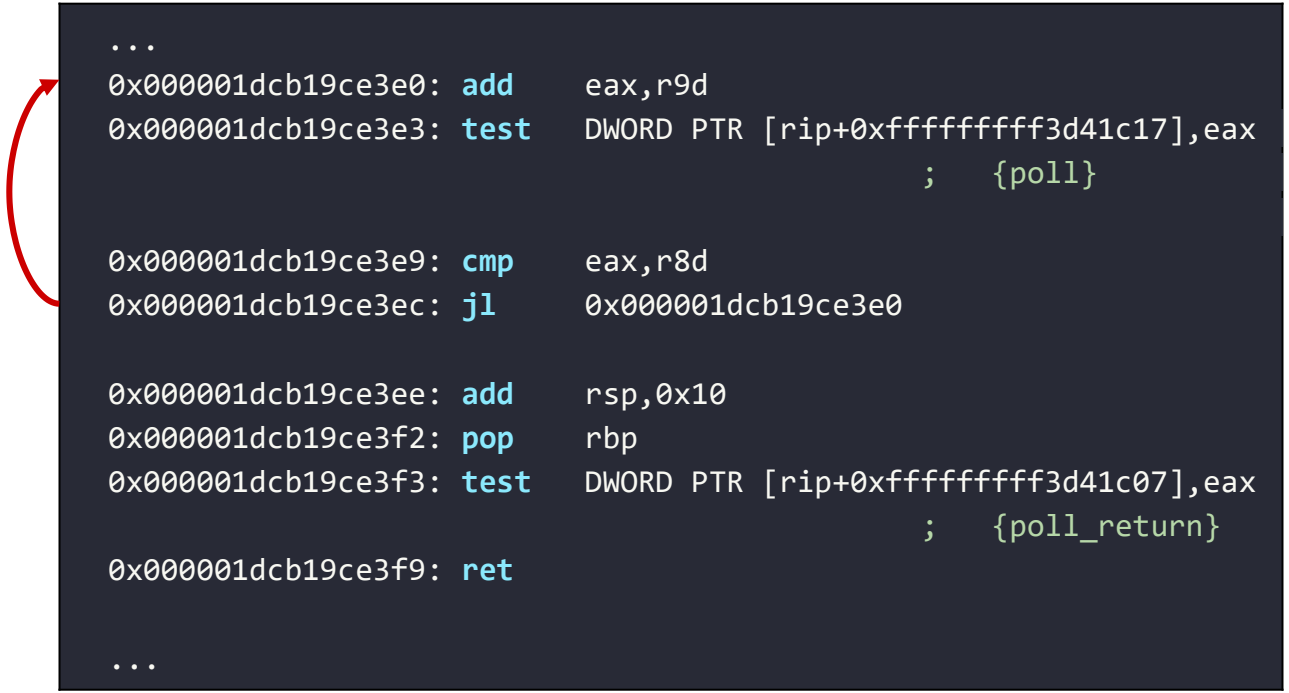
Safepoints

```
public class Test {
    . . .
    public static int test(int limit, int step) {
        int res = 0;
        while (res < limit) {
            res += step;
        }
        return res;
    }
    . . .
}
```

```
$ java -XX:-TieredCompilation -XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly Test
```

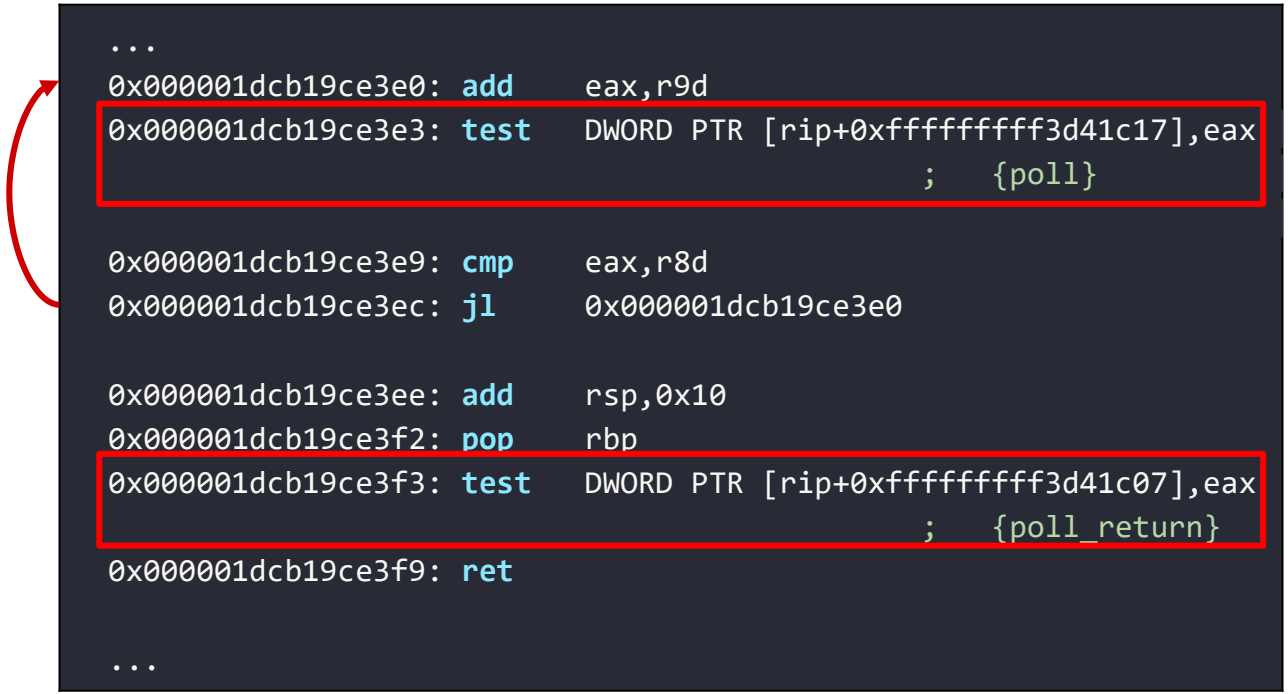
EXCELSIOR

Safepoints



EXCELSIOR

Safepoints

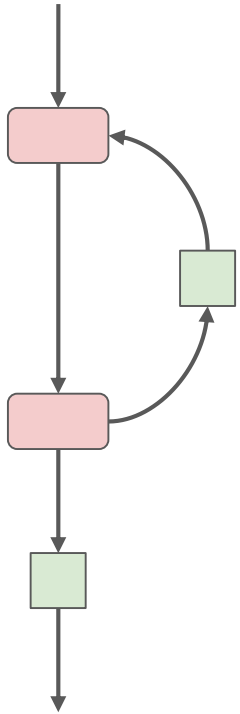


EXCELSIOR

Safepoints

Safepoints based profilers:

- «Включают» остановку на safepoints
- Ждут, пока потоки достигнут sp
- Собирают стеки вызовов
- «Выключают» остановку на safepoints



EXCELSIOR

Safepoints

Почему нам не подходит семплирование через safe points?

Safepoints

Почему нам не подходит семплирование через safe points?

1. Слепые зоны

EXCELSIOR

Safepoints

```
public class Test2 {
    ...
    public int testMethod(int value) {
        int res = 0;
        for (int i = 0; i < 100000; i++) {
            res += value;
        }
        witness();
        return res;
    }
    ...
}
```

```
$ java -XX:-TieredCompilation -XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly Test2
```

EXCELSIOR

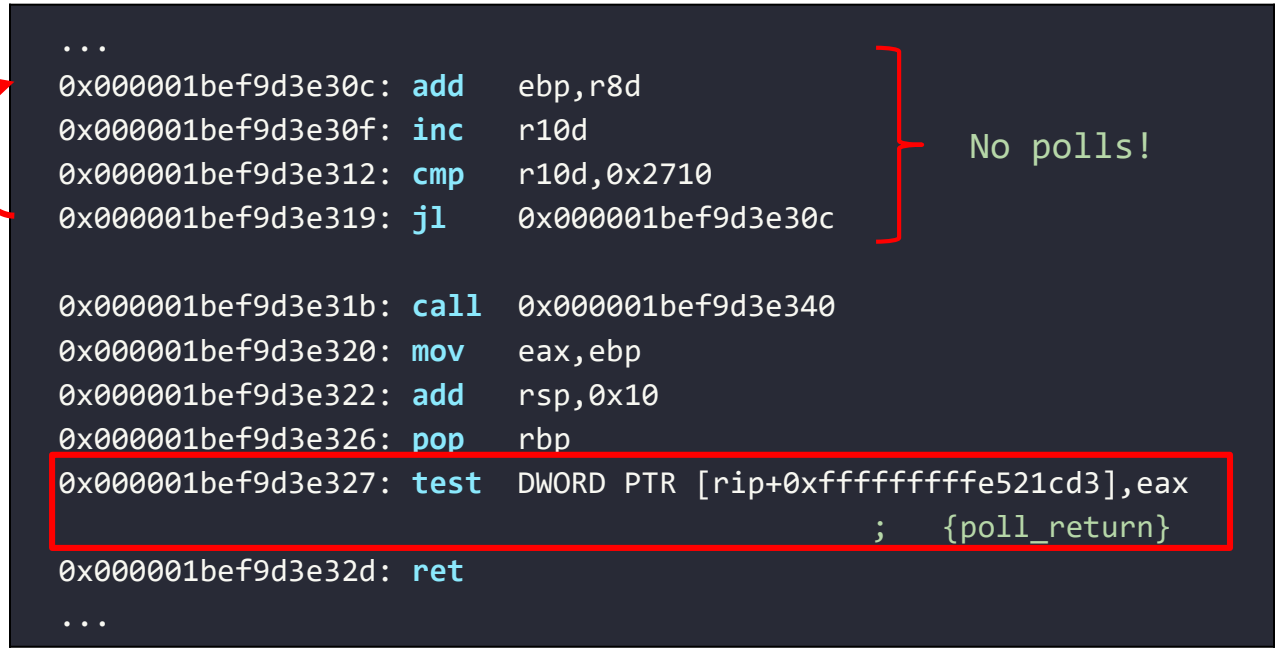
Safepoints

```
...
0x000001bef9d3e30c: add   ebp,r8d
0x000001bef9d3e30f: inc   r10d
0x000001bef9d3e312: cmp   r10d,0x2710
0x000001bef9d3e319: jl    0x000001bef9d3e30c

0x000001bef9d3e31b: call  0x000001bef9d3e340
0x000001bef9d3e320: mov   eax,ebp
0x000001bef9d3e322: add   rsp,0x10
0x000001bef9d3e326: pop   rbp
0x000001bef9d3e327: test  DWORD PTR [rip+0xfffffffffe521cd3],eax
                                              ;   {poll_return}
0x000001bef9d3e32d: ret
...
```

EXCELSIOR

Safepoints



EXCELSIOR

Safepoints

Попадет в профиль

(вместо testMethod)

```
public class Test2 {
    ...
    public int testMethod(int value) {
        int res = 0;
        for (int i = 0; i < 100000; i++) {
            res += value;
        }
        witness();
        return res;
    }
    ...
}
```

Слепая зона

EXCELSIOR

Safepoints

Почему нам не подходит семплирование через safe points?

1. Слепые зоны
2. Непредсказуемое время инспекций

EXCELSIOR

Safepoints

Почему нам не подходит семплирование через safe points?

1. Слепые зоны
2. Непредсказуемое время инспекций

Why (Most) Sampling Java Profilers Are F*cking Terrible

by Nitsan Wakart

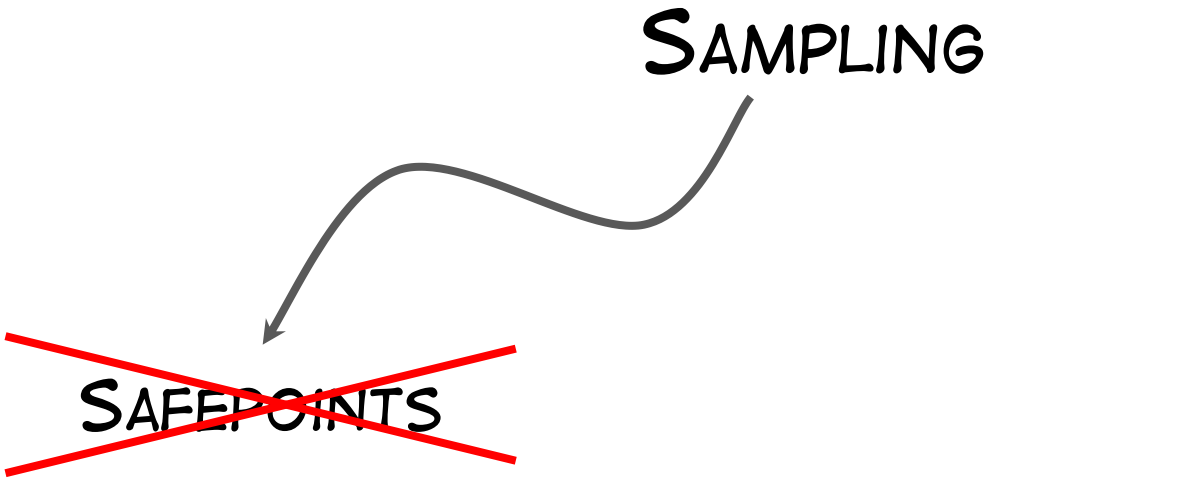
EXCELSIOR

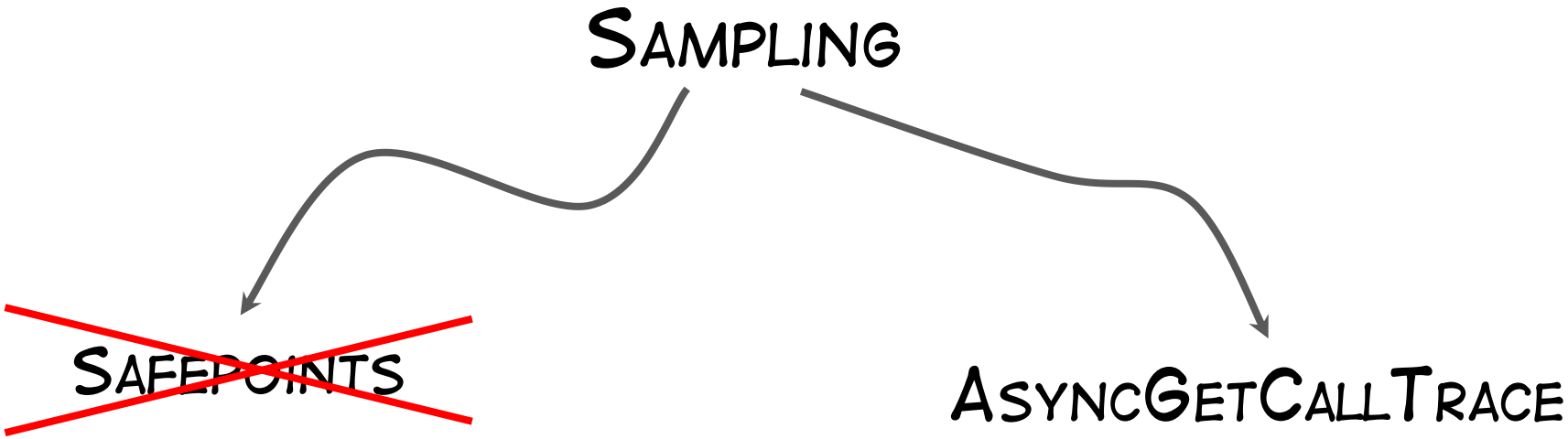
Safepoints

Требования:

1. ~~Информативность~~ ➢ ~~Горячие методы + call chains~~
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
3. Простота ➢ Включать профилировку VM опцией
4. Эффективность ➢ Промышленное использование

EXCELSIOR


SAMPLING
SAFEPOINTS



EXCELSIOR

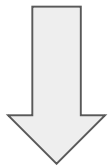
AsyncGetCallTrace

- OpenJDK internal API
- Posix Signals, SIGPROF
- В обработчике вызов AsyncGetCallTrace

EXCELSIOR

AsyncGetCallTrace

- OpenJDK internal API
- Posix Signals, SIGPROF
- В обработчике вызов AsyncGetCallTrace

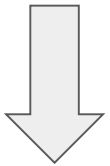


- Стек вызовов ВНЕ safepoint

EXCELSIOR

AsyncGetCallTrace

- OpenJDK internal API
- Posix Signals, SIGPROF
- В обработчике вызов AsyncGetCallTrace



- Стек вызовов ВНЕ safepoint
- Честность
- Не зависит от количества потоков

EXCELSIOR

AsyncGetCallTrace

Ложка дегтя

```
enum {
    ticks_no_Java_frame         =  0, // new thread
    ticks_no_class_load         = -1, // jmethodIds are not available
    ticks_GC_active             = -2, // GC action
    ticks_unknown_not_Java      = -3, // ¯\_(ツ)_/¯
    ticks_not_walkable_not_Java = -4, // ¯\_(ツ)_/¯
    ticks_unknown_Java          = -5, // ¯\_(ツ)_/¯
    ticks_not_walkable_Java     = -6, // ¯\_(ツ)_/¯
    ticks_unknown_state         = -7, // ¯\_(ツ)_/¯
    ticks_thread_exit           = -8, // dying thread
    ticks_deopt                 = -9, // mid-deopting code
    ticks_safepoint             = -10 // ¯\_(ツ)_/¯
};
```

EXCELSIOR

AsyncGetCallTrace

Ложка дегтя №2

- OpenJDK internal API
- Posix Signals, SIGPROF

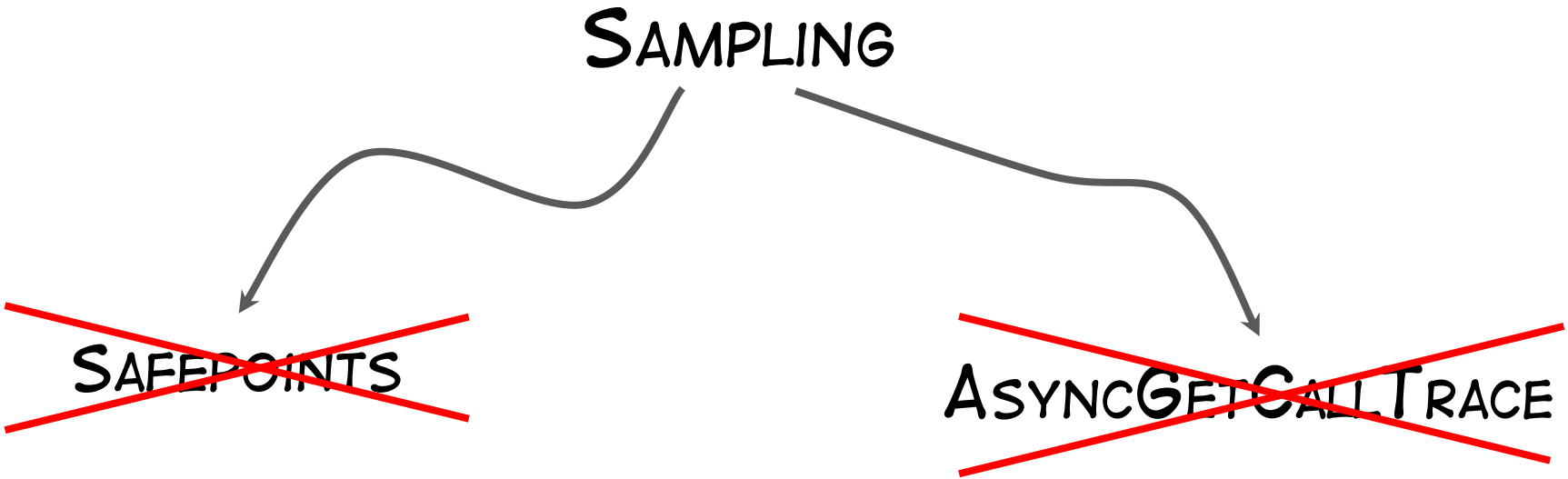
EXCELSIOR

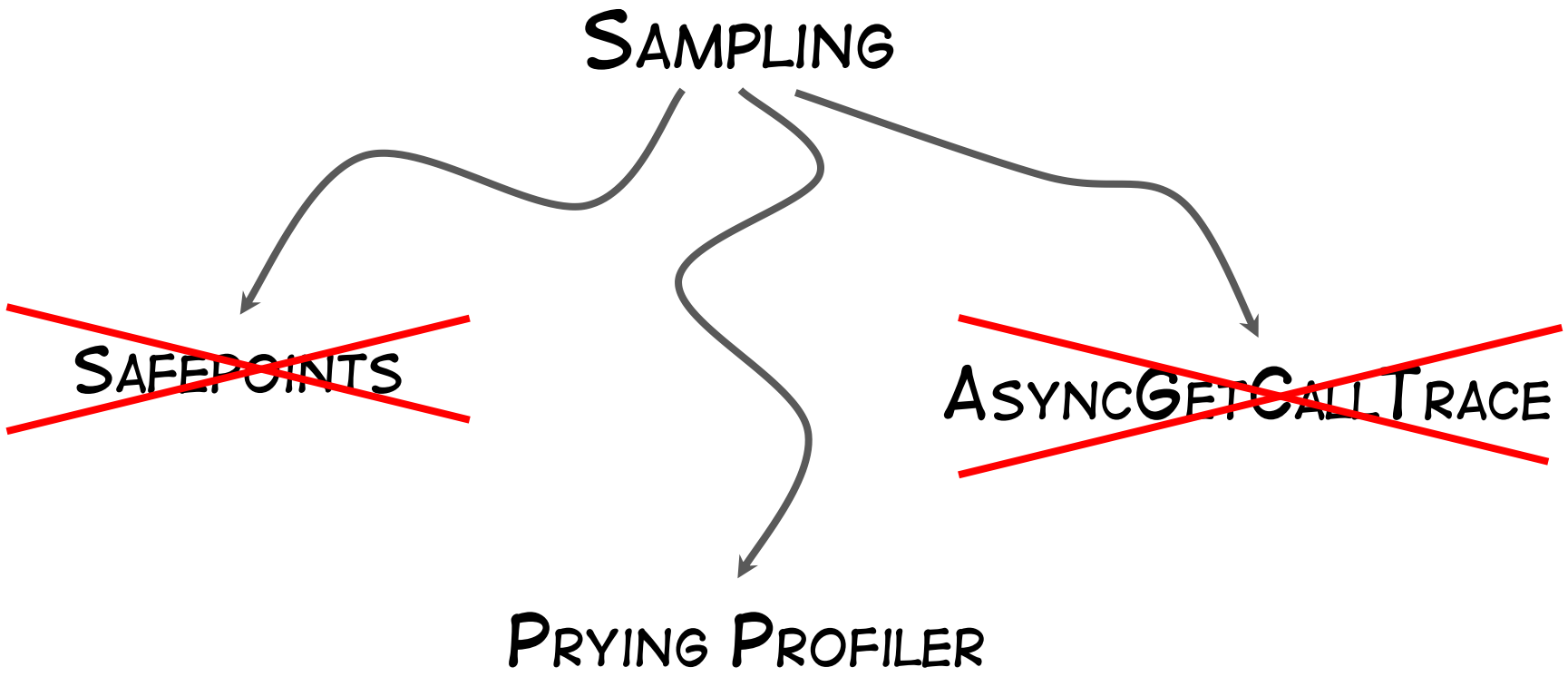
AsyncGetCallTrace

Требования:

1. Информативность ➢ Горячие методы + call chains
2. ~~Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32~~
3. Простота ➢ Включать профилировку VM опцией
4. Эффективность ➢ Промышленное использование

EXCELSIOR


SAMPLING
SAFEPOINTS
ASYNCGETCALLTRACE


Sampling
Safepoints
AsyncGetCallTrace
Prying Profiler

Prying Profiler

Профилировщик в Excelsior JET

Posix Signals

Остановка потока
в произвольной точке

EXCELSIOR

Prying Profiler

Профилировщик в Excelsior JET

Posix Signals

Остановка потока
в произвольной точке

- `WinBase.SuspendThread(HANDLE)`
- `pthread_kill(pthread_t, int)`

EXCELSIOR

Prying Profiler

Профилировщик в Excelsior JET

Честность

Переносимость

EXCELSIOR

Prying Profiler

Профилировщик в Excelsior JET

↓ Честность

↑ Переносимость

EXCELSIOR

Prying Profiler

Схема работы:

1. Отдельный поток для профилировщика
2. Инспекции раз в заданный интервал

EXCELSIOR

Prying Profiler

Схема работы:

1. Отдельный поток для профилировщика
2. Инспекции раз в заданный интервал
    a. Останавливаем очередной поток системными средствами
    b. Восстанавливаем стек вызовов для потока
    (не только в safepoint)
    c. Возобновляем работу потока

EXCELSIOR

Prying Profiler

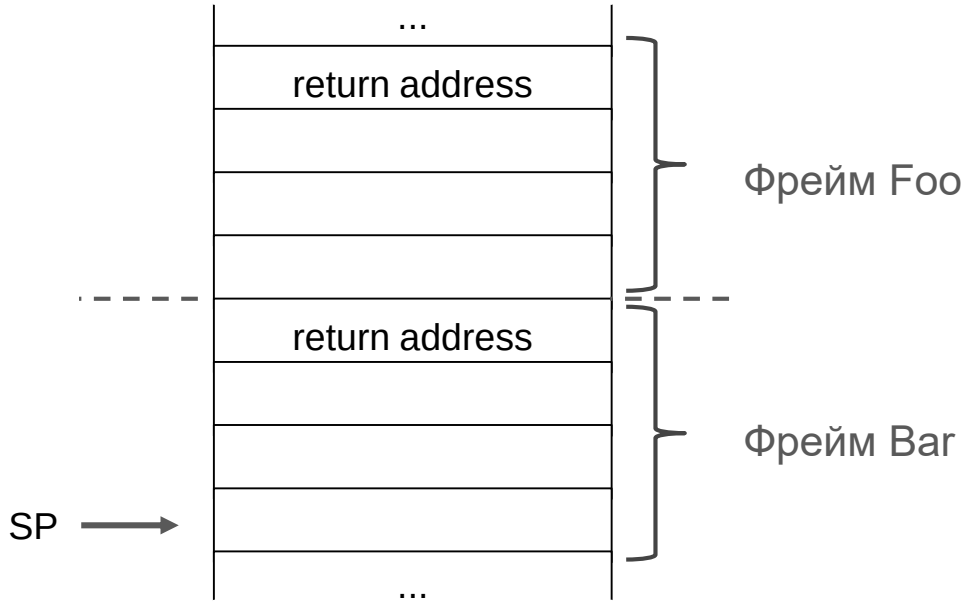
Как получить стек вызовов в произвольной точке?

EXCELSIOR

Prying Profiler

Как получить стек вызовов в произвольной точке?

- *Foo* вызывает *Bar*
- Инспектируем поток



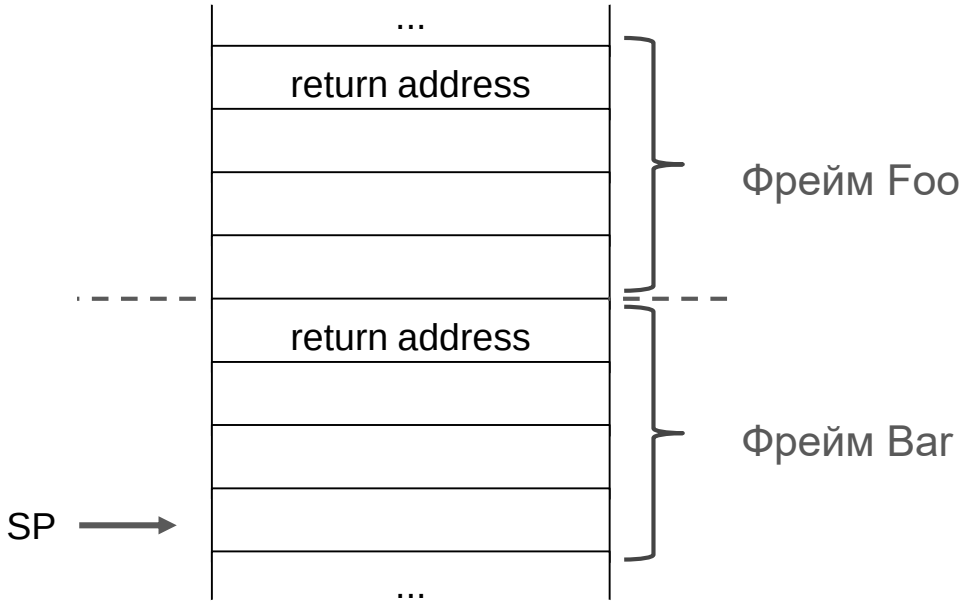
EXCELSIOR

Prying Profiler

Как получить стек вызовов в произвольной точке?

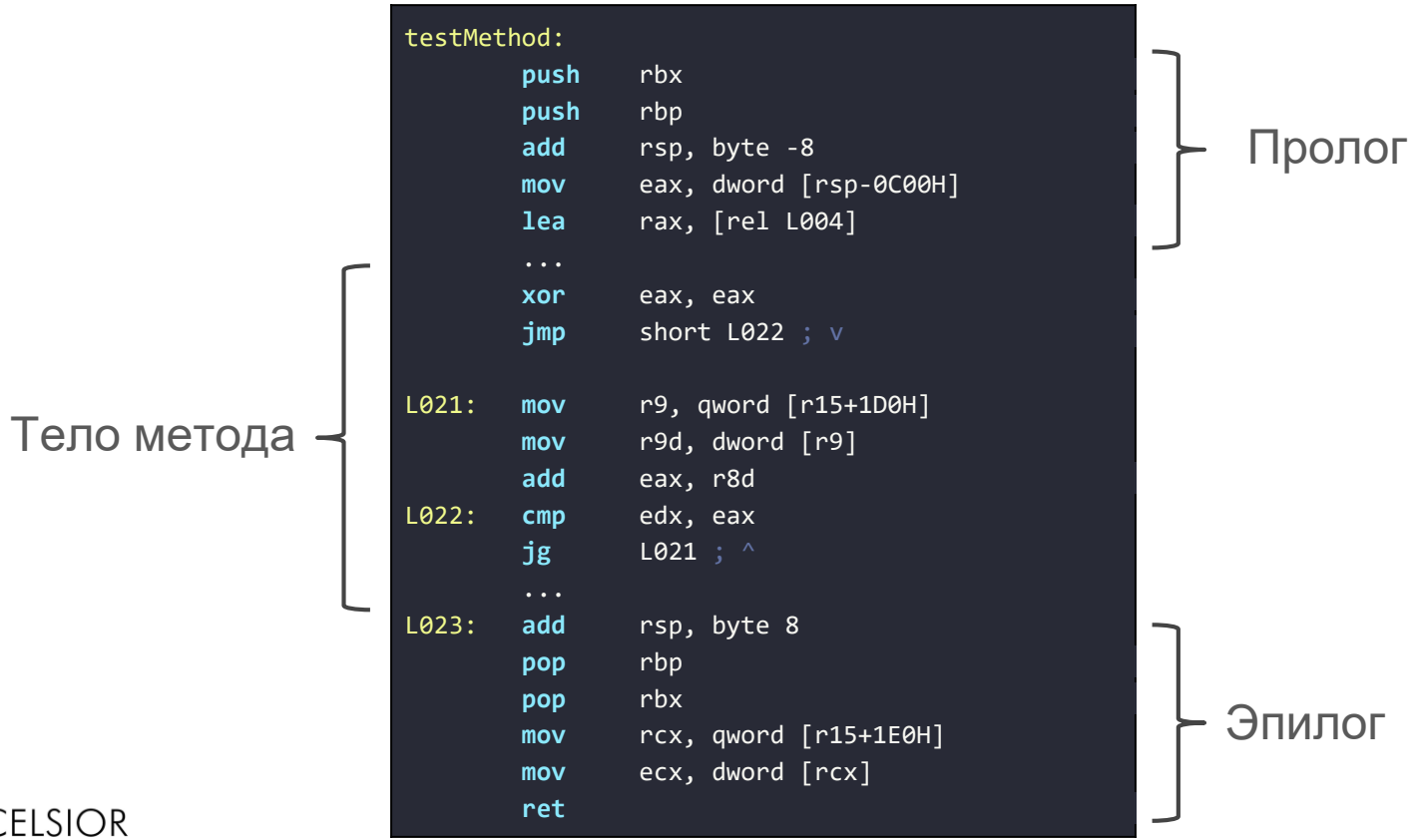
- *Foo* вызывает *Bar*
- Инспектируем поток

1. По IP находим *Bar*
2. Осталось найти фрейм Foo



EXCELSIOR

Prying Profiler



EXCELSIOR

Prying Profiler

Исполняли тело метода ⇒ фрейм сформирован (почти safepoint)

```
testMethod:
        push    rbx
        push    rbp
        add     rsp, byte -8
        mov     eax, dword [rsp-0C00H]
        lea     rax, [rel L004]
        ...

        xor     eax, eax
        jmp     short L022 ; v

L021:   mov     r9, qword [r15+1D0H]
        mov     r9d, dword [r9]
        add     eax, r8d
L022:   cmp     edx, eax
        jg      L021 ; ^
        ...
L023:   add     rsp, byte 8
        pop     rbp
        pop     rbx
        mov     rcx, qword [r15+1E0H]
        mov     ecx, dword [rcx]
        ret
```

EXCELSIOR

Prying Profiler

Исполняли пролог или эпилог ⇒

фрейма еще (уже) нет

Знаем, как искать caller фрейм

```
testMethod:
        push    rbx
        push    rbp
        add     rsp, byte -8
        mov     eax, dword [rsp-0C00H]
        lea     rax, [rel L004]
        ...

        xor     eax, eax
        jmp     short L022 ; v

L021:   mov     r9, qword [r15+1D0H]
        mov     r9d, dword [r9]
        add     eax, r8d
L022:   cmp     edx, eax
        jg      L021 ; ^
        ...
L023:   add     rsp, byte 8
        pop     rbp
        pop     rbx
        mov     rcx, qword [r15+1E0H]
        mov     ecx, dword [rcx]
        ret
```

EXCELSIOR

Prying Profiler

```
public int lightweightMethod(int p) {
    int res = 1;
    for (int i = 0; i < 100; i++) {
        res *= p;
    }
    return res;
}
```

Особый метод:

- Исключений не бросает
- Safepoints в теле нет
- Call stack не сканируется

EXCELSIOR

Prying Profiler

```
lightweightMethod:
        mov     eax, 1
        xor     ecx, ecx
        jmp     short L018 ; v

L017:   imul    eax, edx
        add     ecx, byte 1
L018:   cmp     ecx, byte 100
        jl      L017 ; ^
        mov     rcx, qword [r15+1E0H]
        mov     ecx, dword [rcx]
        ret
```

Оптимизация: фрейм вообще не строится

(никто ведь не будет сканировать call stack)

EXCELSIOR

Prying Profiler

```
lightweightMethod:
        mov     eax, 1
        xor     ecx, ecx
        jmp     short L018 ; v

L017:   imul    eax, edx
        add     ecx, byte 1
L018:   cmp     ecx, byte 100
        jl      L017 ; ^
        mov     rcx, qword [r15+1E0H]
        mov     ecx, dword [rcx]
        ret
```

Худший случай:

- Достоверной информации о caller фрейме нет
- Используются эвристики (ищем ближайший хороший фрейм)
- Редкий случай, умеем отличать в профиле

EXCELSIOR

Prying Profiler

Что попадает в профиль?

- Java методы попадают в профиль
  - точные или эвристические хиты
- Native методы распознаются, но игнорируются
- Часть потоков отфильтровываются
  - служебные потоки JVM
  - потоки исполняющие OS

EXCELSIOR

Prying Profiler

интервал между инспекциями — 1ms

издержки производительности — ~10%

достаточная информативность и точность

EXCELSIOR

Prying Profiler

Intel® VTune™ Amplifier	
Method name	CPU Time
Collections.longStaticBTree.getObject	12.5662%
Util.DisplayScreen.putText	4.3703%
Collections.StringBTreeNode.SearchNode	2.5284%
Util.DisplayScreen.putDollars	2.5115%
StockLevelTransaction.process	2.4576%

EXCELSIOR

Prying Profiler

Intel® VTune™ Amplifier	
Method name	CPU Time
Collections.longStaticBTree.getObject	12.5662%
Util.DisplayScreen.putText	4.3703%
Collections.StringBTreeNode.SearchNode	2.5284%
Util.DisplayScreen.putDollars	2.5115%
StockLevelTransaction.process	2.4576%

Excelsior JET Prying Profiler	
Method name	Hits
Collections.longStaticBTree.getObject	14.659%
Util.DisplayScreen.putText	4.690%
Util.DisplayScreen.putDollars	3.029%
Collections.StringBTreeNode.SearchNode	3.013%
StockLevelTransaction.process	2.713%

EXCELSIOR

Prying Profiler

интервал между инспекциями — 1ms

издержки производительности — ~10%

достаточная информативность и точность

EXCELSIOR

Prying Profiler

интервал между инспекциями — 1ms
(варьируется)

издержки производительности — ~10%
(есть контрпримеры)

достаточная информативность и точность

EXCELSIOR

Prying Profiler

Требования:

1. Информативность ➢ Горячие методы + call chains
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
3. Простота ➢ Включать профилировку VM опцией
4. Эффективность ➢ Промышленное использование

EXCELSIOR

Prying Profiler

Требования:

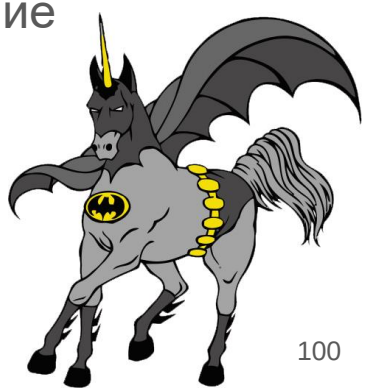
✓	1. Информативность	➢ Горячие методы + call chains
✓	2. Универсальность	➢ Windows/macOS/Linux + x86/AMD64/ARM32
✓	3. Простота	➢ Включать профилировку VM опцией
	4. Эффективность	➢ Промышленное использование

EXCELSIOR

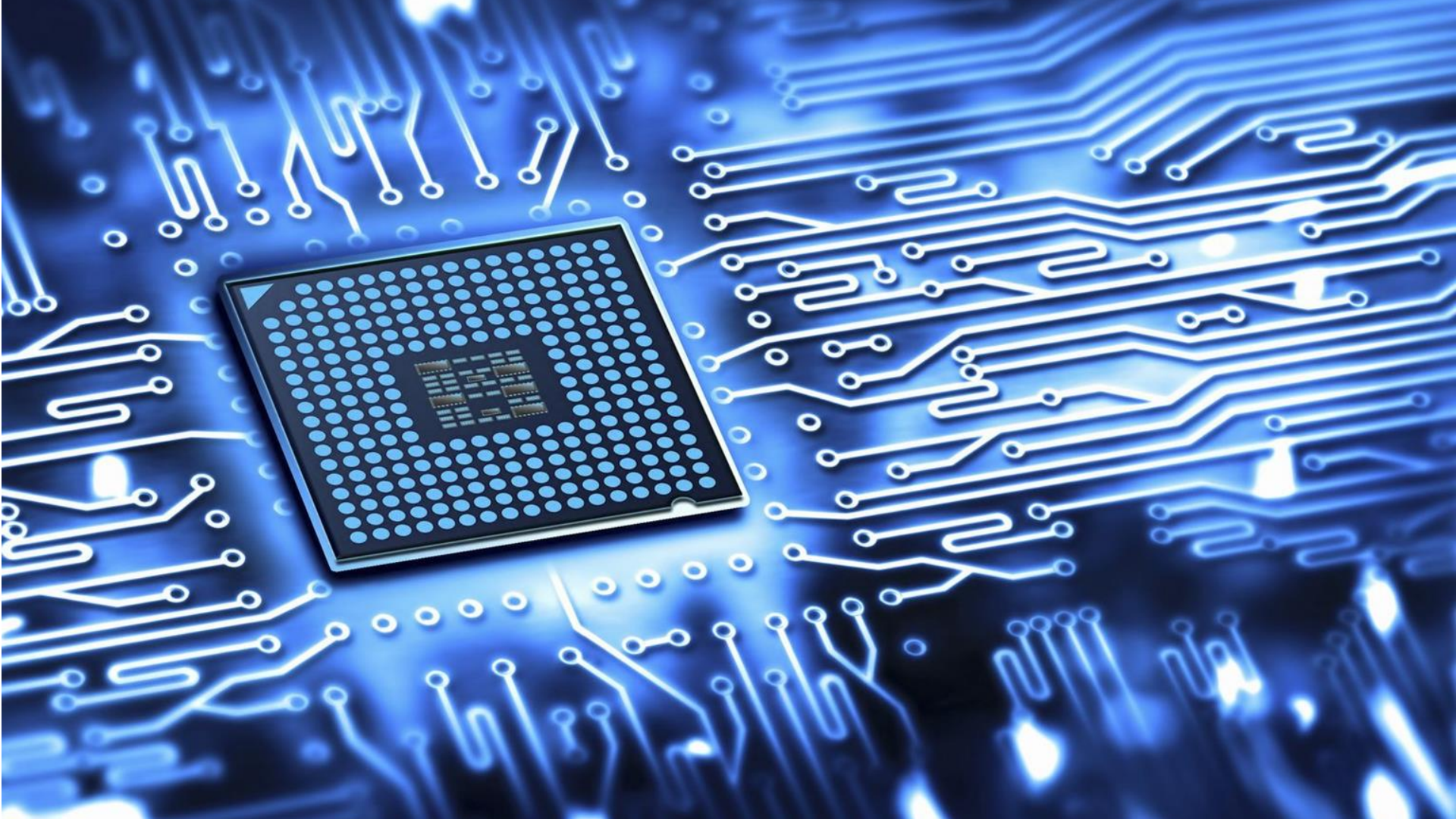
Prying Profiler

Требования:

- ✓ 1. Информативность ➢ Горячие методы + call chains
- ✓ 2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
- ✓ 3. Простота ➢ Включать профилировку VM опцией
- ✓ 4. Эффективность ➢ Промышленное использование

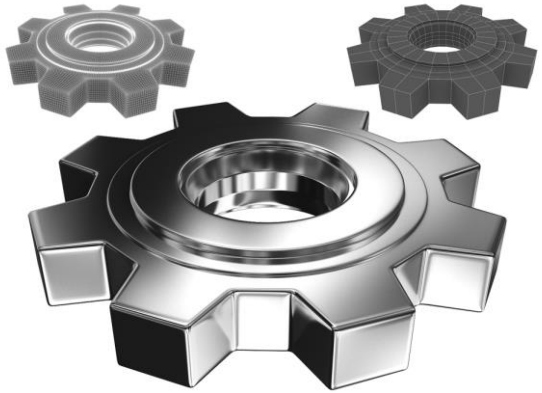


EXCELSIOR



Hardware Profiling

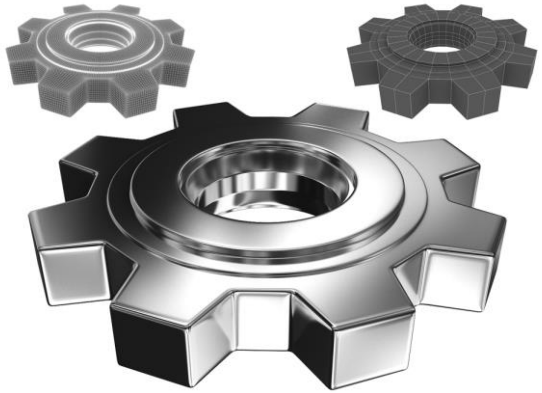
- Hardware Counters
- Семплирование при переполнении



Hardware Profiling

- Hardware Counters
- Семплирование при переполнении

+ Минимальные издержки
+ Высокая частота семплирования



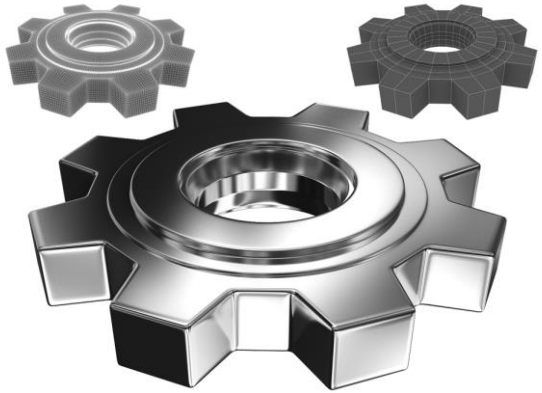
EXCELSIOR

Hardware Profiling

- ➢ Hardware Counters
- ➢ Семплирование при переполнении

- \+ Минимальные издержки
- \+ Высокая частота семплирования

- − CPU- и OS-специфично
- − Privileged instructions



EXCELSIOR

Hardware Profiling

- Linux Perf Events
  - kernel 2.6.31 and above

Hardware Profiling

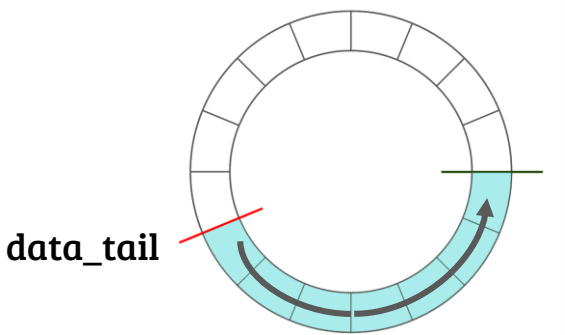
- Clairvoyance Profiler
- Linux Perf Events
  - kernel 2.6.31 and above

Hardware Profiling

- Clairvoyance Profiler
  - **perf_event_open**
- Linux Perf Events
  - kernel 2.6.31 and above

Hardware Profiling

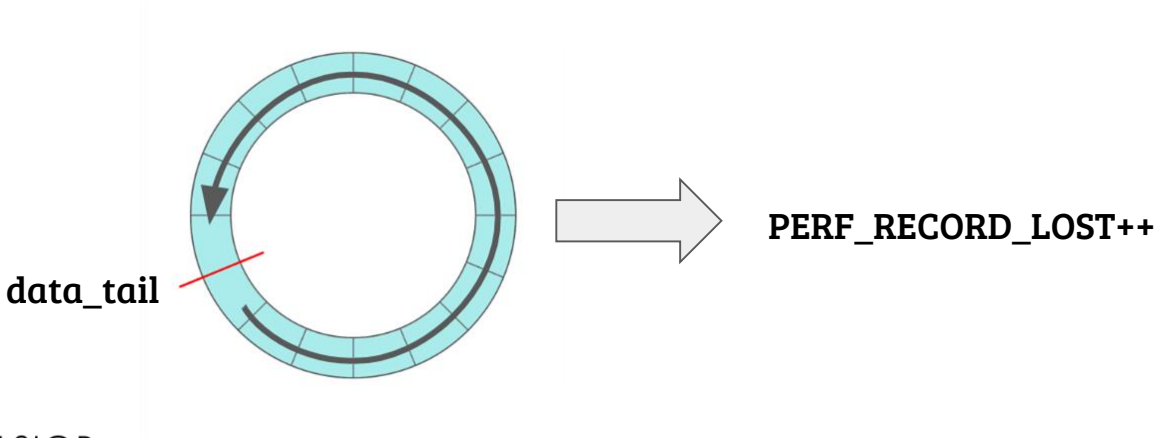
- Clairvoyance Profiler
  - **perf_event_open**
  - CPU записывает семлы в ring buffer
- Linux Perf Events
  - kernel 2.6.31 and above



EXCELSIOR

Hardware Profiling

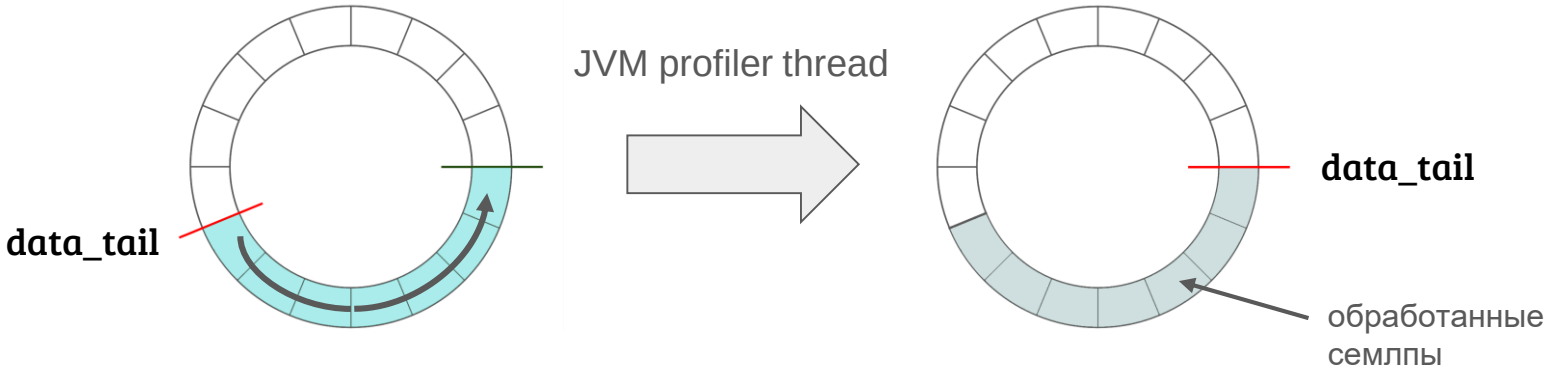
- Clairvoyance Profiler
  - **perf_event_open**
  - CPU записывает семлы в ring buffer
- Linux Perf Events
  - kernel 2.6.31 and above



EXCELSIOR

Hardware Profiling

- Clairvoyance Profiler
  - **perf_event_open**
  - CPU записывает семлы в ring buffer
  - JVM обрабатывает буфер, обновляет **data_tail**
- Linux Perf Events
  - kernel 2.6.31 and above



EXCELSIOR

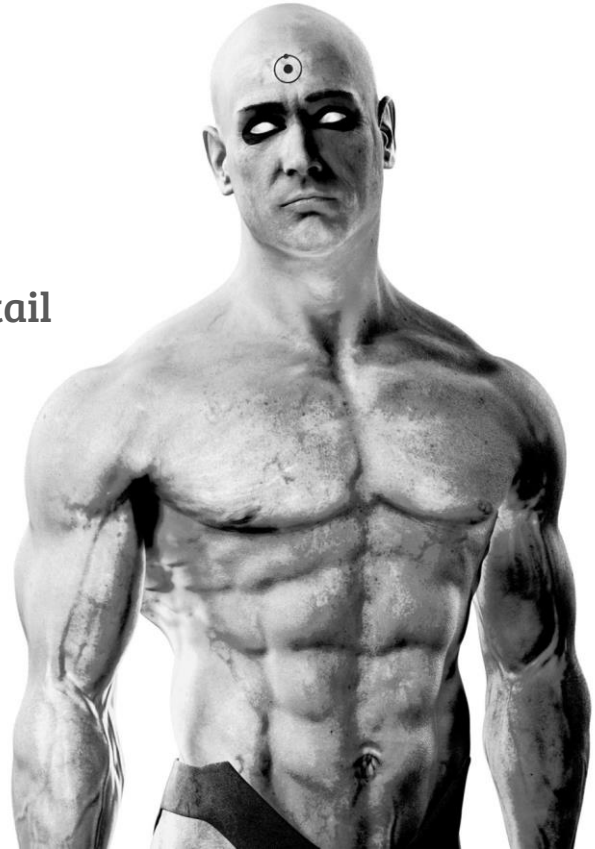
Hardware Profiling

- Clairvoyance Profiler
  - **perf_event_open**
  - CPU записывает семлы в ring buffer
  - JVM обрабатывает буфер, обновляет **data_tail**
  - издержки ~2%
- Linux Perf Events
  - kernel 2.6.31 and above

EXCELSIOR

Hardware Profiling

- Clairvoyance Profiler
  - **perf_event_open**
  - CPU записывает семлы в ring buffer
  - JVM обрабатывает буфер, обновляет **data_tail**
  - издержки ~2%



EXCELSIOR

Hardware Profiling

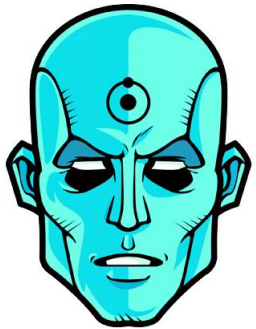
Требования:

✓ 1. Информативность ➢ Горячие методы + call chains

~~2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32~~

✓ 3. Простота ➢ Включать профилировку VM опцией

✓✓ 4. Эффективность ➢ Промышленное использование



EXCELSIOR

Hardware Profiling

Требования:

✓ 1. Информативность ➢ Горячие методы + call chains

~~2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32~~

✓ 3. Простота ➢ Включать профилировку VM опцией

✓✓ 4. Эффективность ➢ Промышленное использование

Но есть backup path — Prying Profiler!

EXCELSIOR

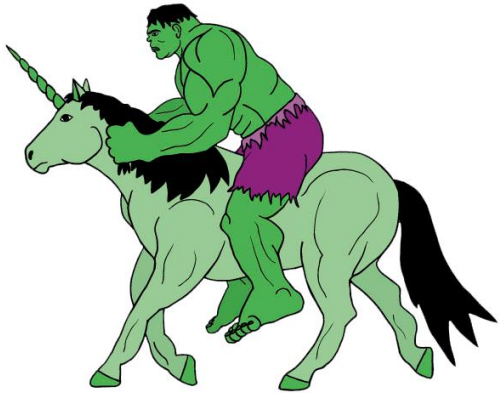
Заключение

Сейчас:

- Prying Profiler включен в Excelsior JET
- Profile-Guided Optimization
- Отладка

Потом:

- Clairvoyance Profiler
- Общедоступность профилировщика

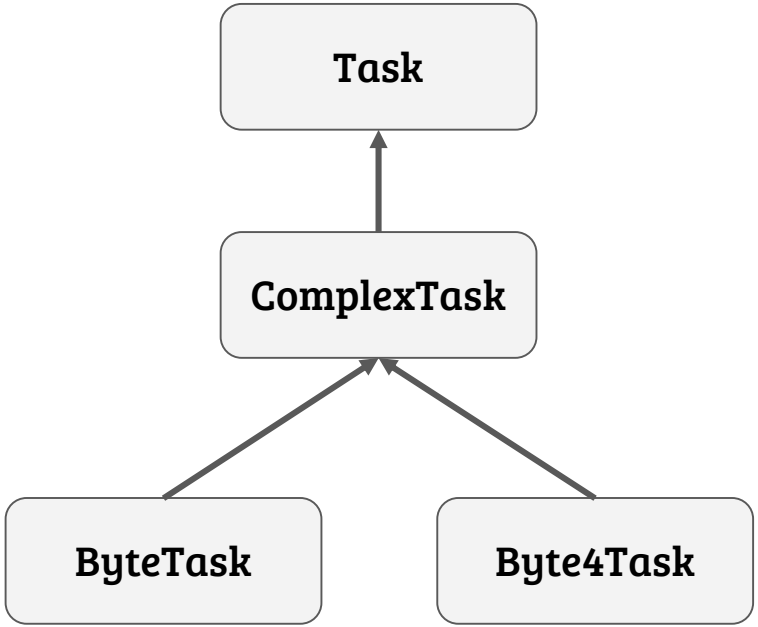


EXCELSIOR

Demo

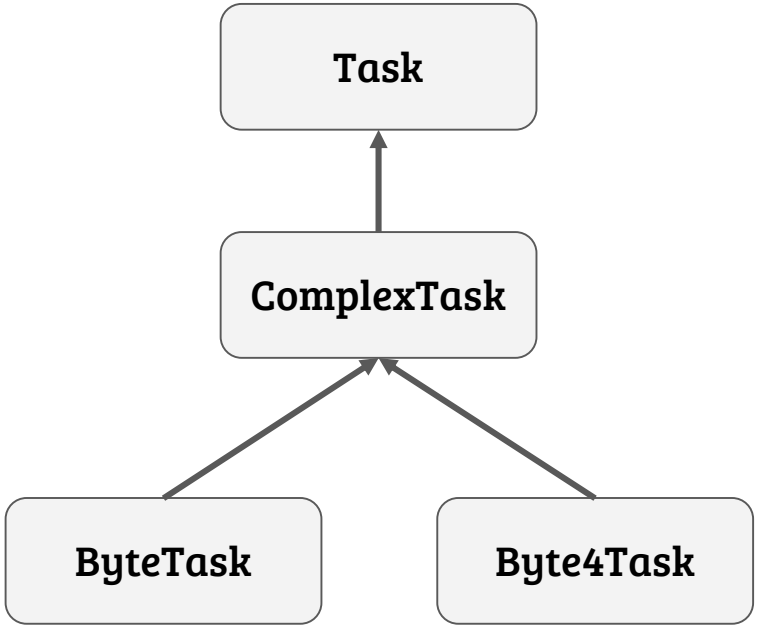



Demo



EXCELSIOR

Demo



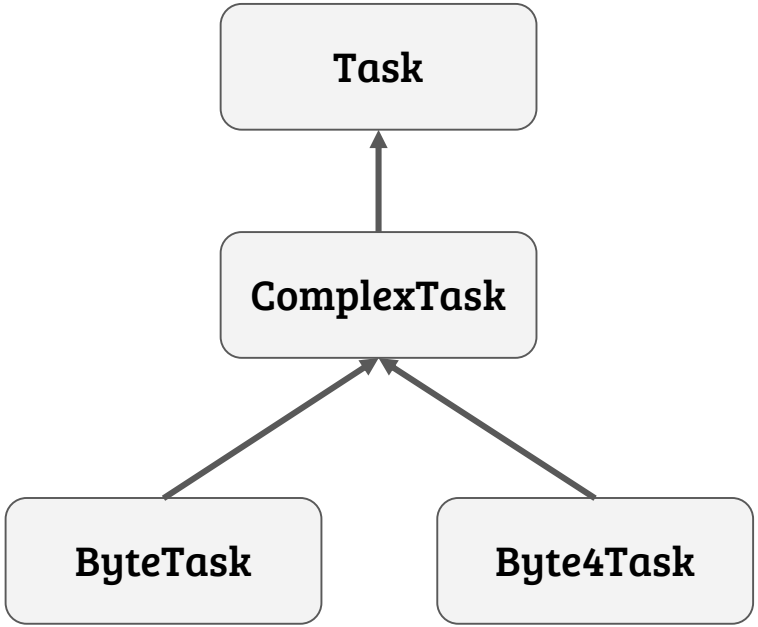

```
static void initData() {
    dataArr = new Task[DATA_SIZE];

    Task t1 = new Byte4Task();
    Task t2 = new ByteTask ();

    for (int i = 0; i < dataArr.length; i++) {
        dataArr[i] = (i % 64 == 0) ? t1 : t2;
    }
}
```

EXCELSIOR

Demo




```
static void initData() {
    dataArr = new Task[DATA_SIZE];

    Task t1 = new Byte4Task();
    Task t2 = new ByteTask ();

    for (int i = 0; i < dataArr.length; i++) {
        dataArr[i] = (i % 64 == 0) ? t1 : t2;
    }
}
...
    for (Task t : dataArr) {
        res += t.compute();
    }
...
```

EXCELSIOR

Demo

Результаты:

OpenJDK 1.8.0_151-1b, Windows, x86_amd64	803 ms
JET 14.0 Windows-amd64	4 388 ms
JET 14.0 Windows-amd64 + Profiler	4 811 ms
JET 14.0 Windows-amd64 + PGO	525 ms

Пример на github:

https://github.com/excelsior-oss/excelsior-jet-samples/tree/master/pgo-bench

EXCELSIOR

Q & A

iugliansky@excelsior-usa.com

@dbg_nsk

https://www.excelsiorjet.com/blog

EXCELSIOR

Secret slide

А что по поводу JMC/JFR?

- Видит между safepoints только с `-XX:DebugNonSafepoints`
- Минимальный интервал семплирования 10ms
- Универсальность (работает и на Windows)

EXCELSIOR

Secret slide

А что по поводу JMC/JFR?

1. Информативность ➢ Горячие методы + call chains
2. Универсальность ➢ Windows/macOS/Linux + x86/AMD64/ARM32
3. ~~Простота ➢ Включать профилировку VM опцией~~
4. Эффективность ➢ Промышленное использование

EXCELSIOR