JITProf: Pinpointing JIT-Unfriendly JavaScript Code

Liang Gong[1], Michael Pradel[2], Koushik Sen[1]

[1] UC Berkeley, [2] TU Darmstadt

Motivation

JavaScript: One of the most popular languages

Performance: Crucial for responsiveness and energy-efficiency

JIT Compilers to the Rescue?

Just-in-time compilation:

Performance despite JavaScript's dynamism

JIT Compilers to the Rescue?

Just-in-time compilation:

Performance despite JavaScript's dynamism

But: Relies on regularity assumptions about code

Developers may write JIT-unfriendly code

Example

```
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
  node.left = this.root_;
  node.right = this.root_.right;
  ...
} else {
  node.right = this.root_;
  node.left = this.root_.left;
  ...
}
```

From Google's Octane-Splay benchmark

Example

```
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
  node.left = this.root_;
  node.right = this.root_.right;
  ...
} else {
  node.right = this.root_;
  node.left = this.root_.left;
  ...
}
```

Assumption broken: Properties initialized in inconsistent order

From Google's Octane-Splay benchmark

Example

```
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
  node.left = this.root_;
  node.right = this.root_.right;
  ...
} else {
  node.right = this.root_;
  node.left = this.root_.left;
  ...
}
```

Assumption broken: Properties initialized in inconsistent order

From Google's Octane-Splay benchmark

Example

```
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
  node.right = this.root_.right;
  node.left = this.root_;
  ...
} else {
  node.right = this.root_;
  node.left = this.root_.left;
  ...
}
```

Swapped lines:

15.1% faster

(in Chrome)

From Google's Octane-Splay benchmark

How To Find JIT-Unfriendly Code?

CPU profiling?

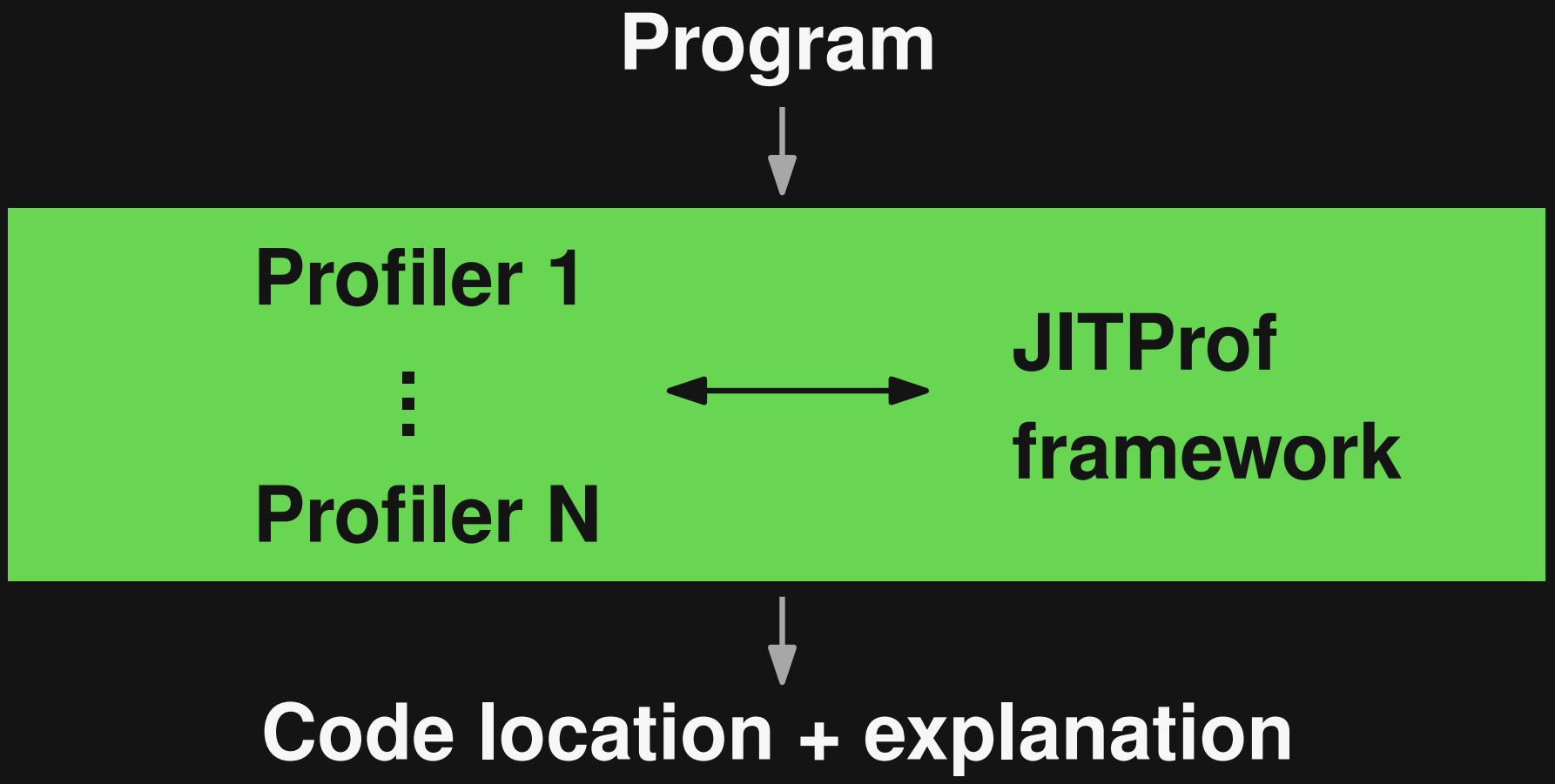
- Many hot functions
- Few optimization opportunities

Feedback from engine?

- Engine-specific
- For experts only

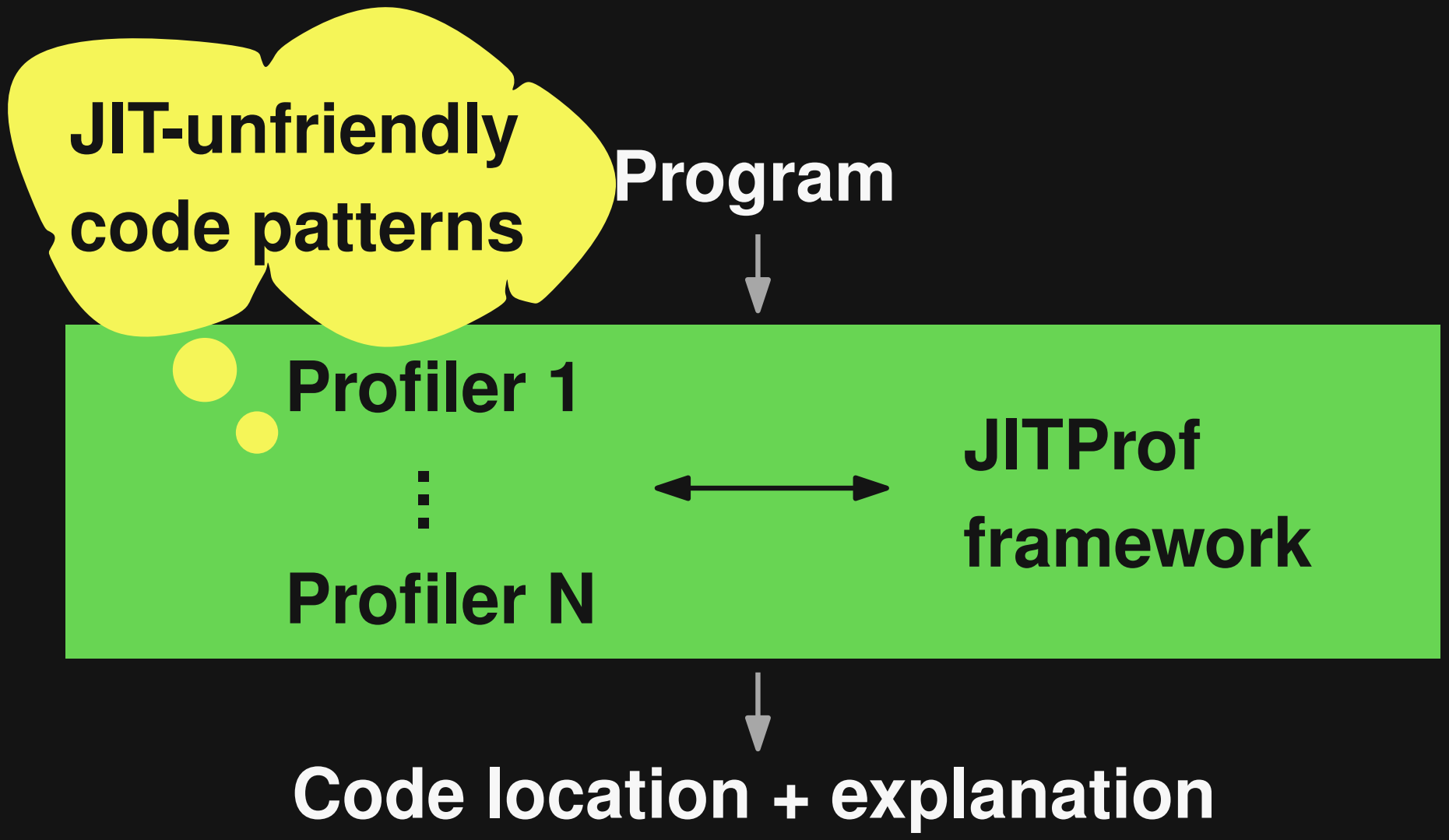
This Talk: JITProf

Profiler that pinpoints JIT-unfriendly code



This Talk: JITProf

Profiler that pinpoints JIT-unfriendly code



Pattern 1: Polymorphic Operations

```
function f(a, b) {
  return a + b;
}
f(2, 5);
f("ESEC", "FSE");
```

Pattern 1: Polymorphic Operations

```
function f(a, b) {
  return a + b;
}
f(2, 5);
f("ESEC", "FSE");
```

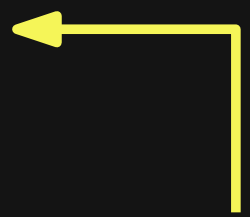

- **Location** that applies operation to **different types**
- JIT compiler cannot specialize code for plus operation

Pattern 2: Mostly Numeric Arrays

```
var array = [];
for (var i = 0; i < 1000; i++) {
  array[i] = i;
}
array[4] = "abc";
```

Pattern 2: Mostly Numeric Arrays

```
var array = [];
for (var i = 0; i < 1000; i++) {
  array[i] = i;
}
array[4] = "abc";
```



- Stores non-numeric value into a numeric array
- Forces change of array representation

Pattern 3: Inconsistent Obj. Layouts

```
var x = {};

x.a = "hello";
x.b = "world";




.. = x.a;
```

Pattern 3: Inconsistent Obj. Layouts

```
var x = {};

x.a = "hello";
x.b = "world";
```

```
.. = x.a;
```

- Engine keeps track of ”hidden class”:

  Offset of `a`: 0

- Inline caching to optimize property access:

  Replace `x.a` with `x[0]`

Pattern 3: Inconsistent Obj. Layouts

```
var x = {};
if (..) {
  x.a = "hello";
  x.b = "world";
} else {
  x.b = "world";
  x.a = "hello";
}
.. = x.a;
```

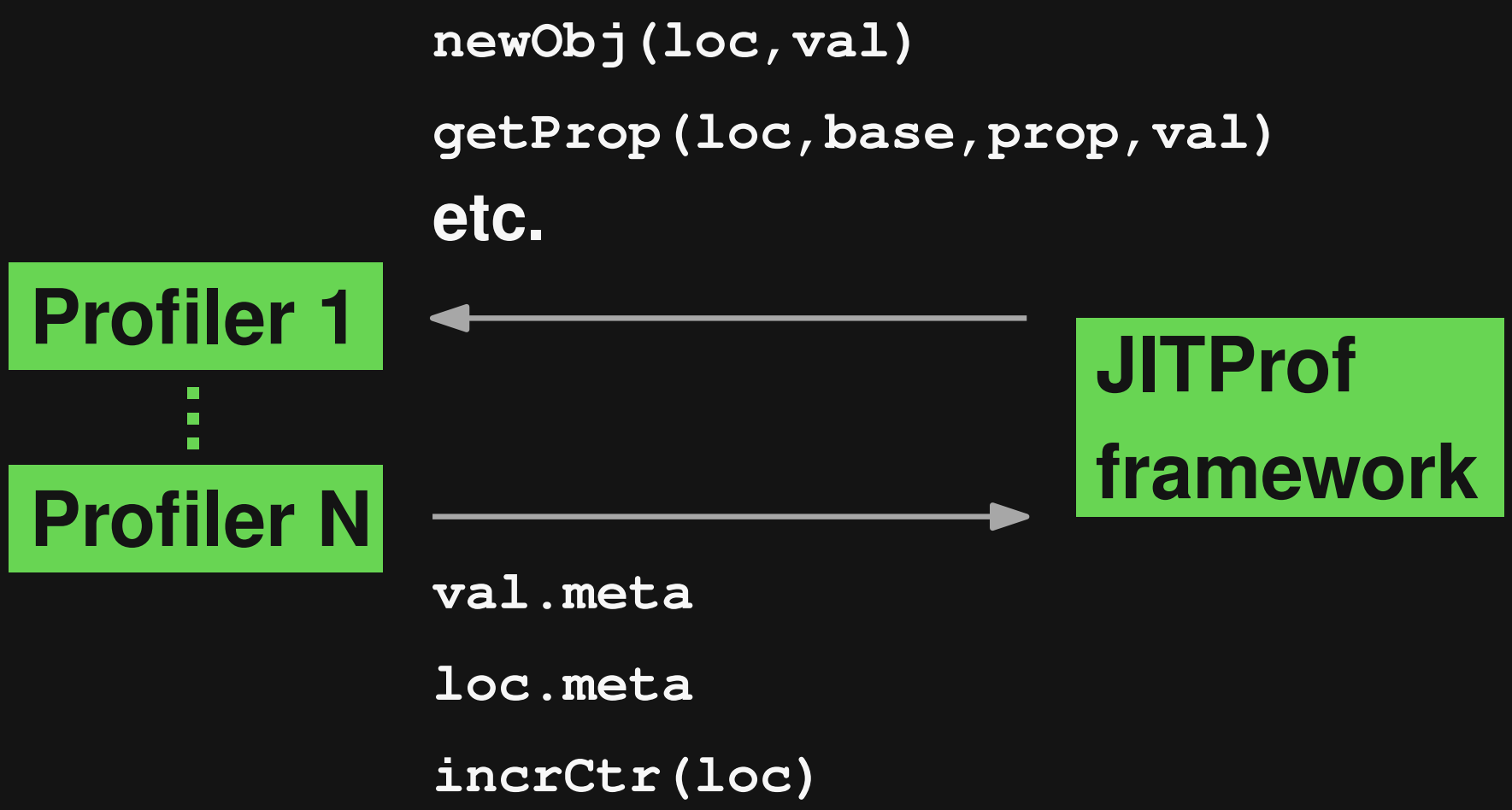
- Engine keeps track of ”hidden class”:

  Offset of `a`: 0 or 1

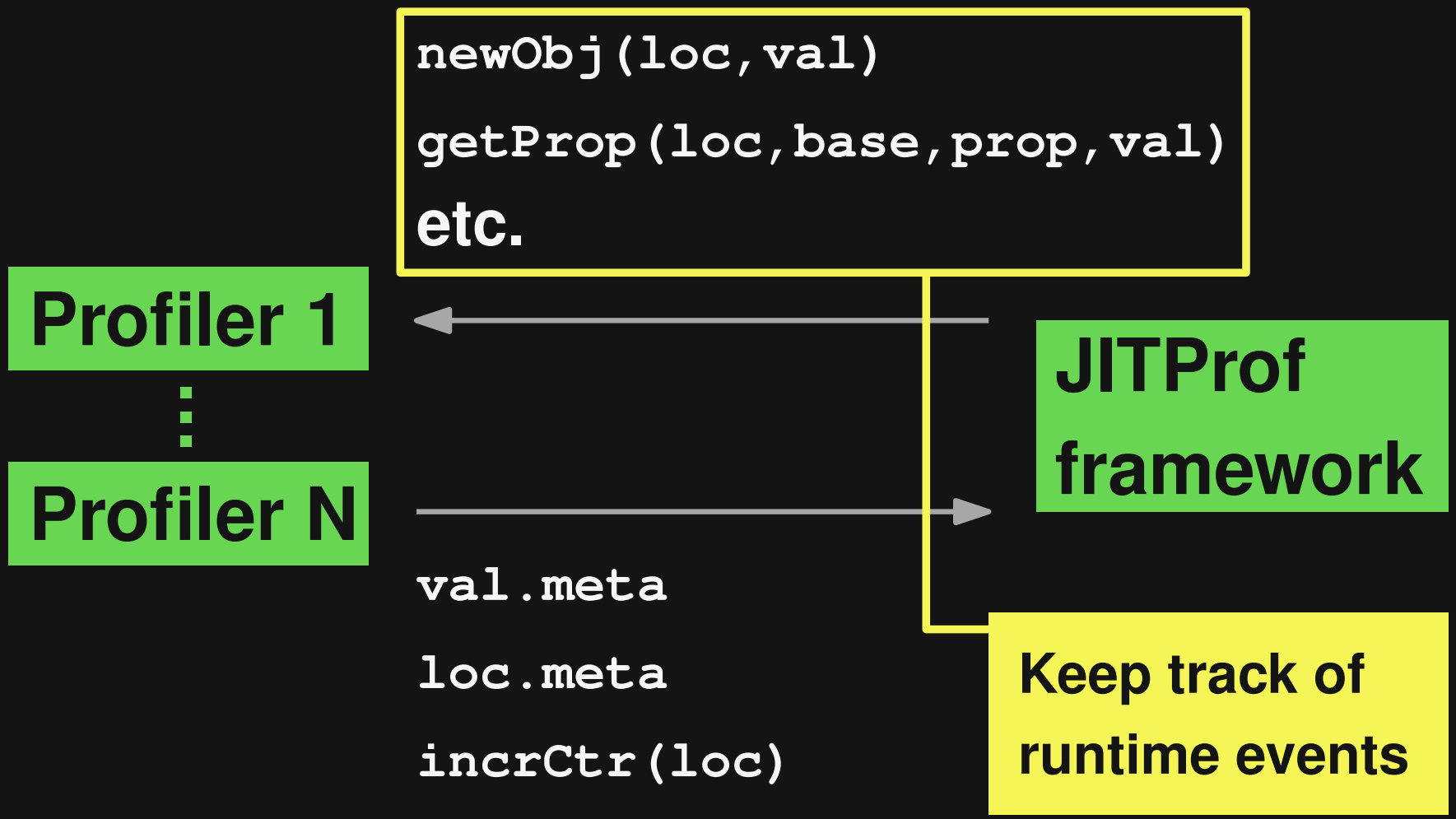
- **Inline caching fails**:

  Cannot replace `x.a` with fixed offset

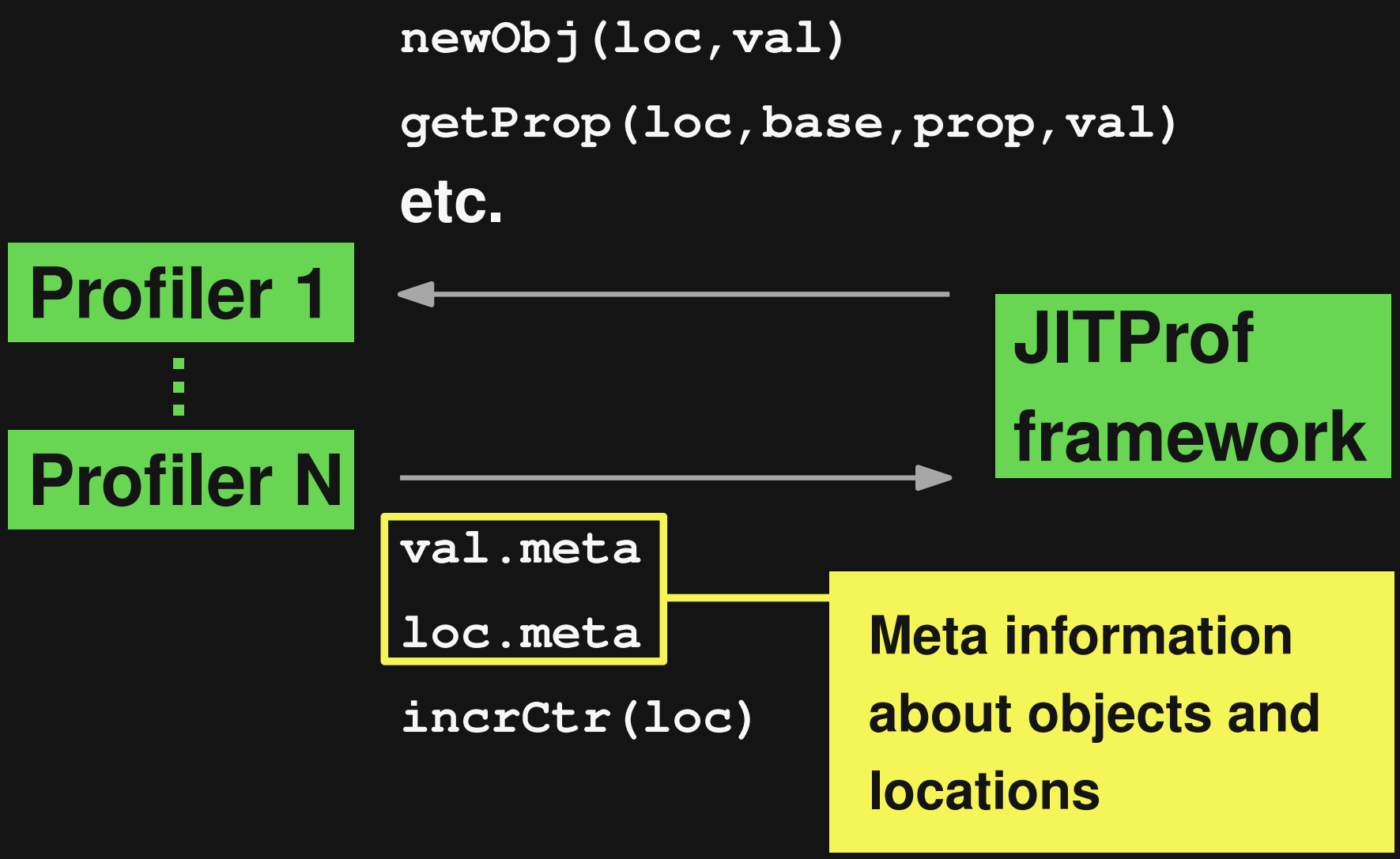
Framework and API



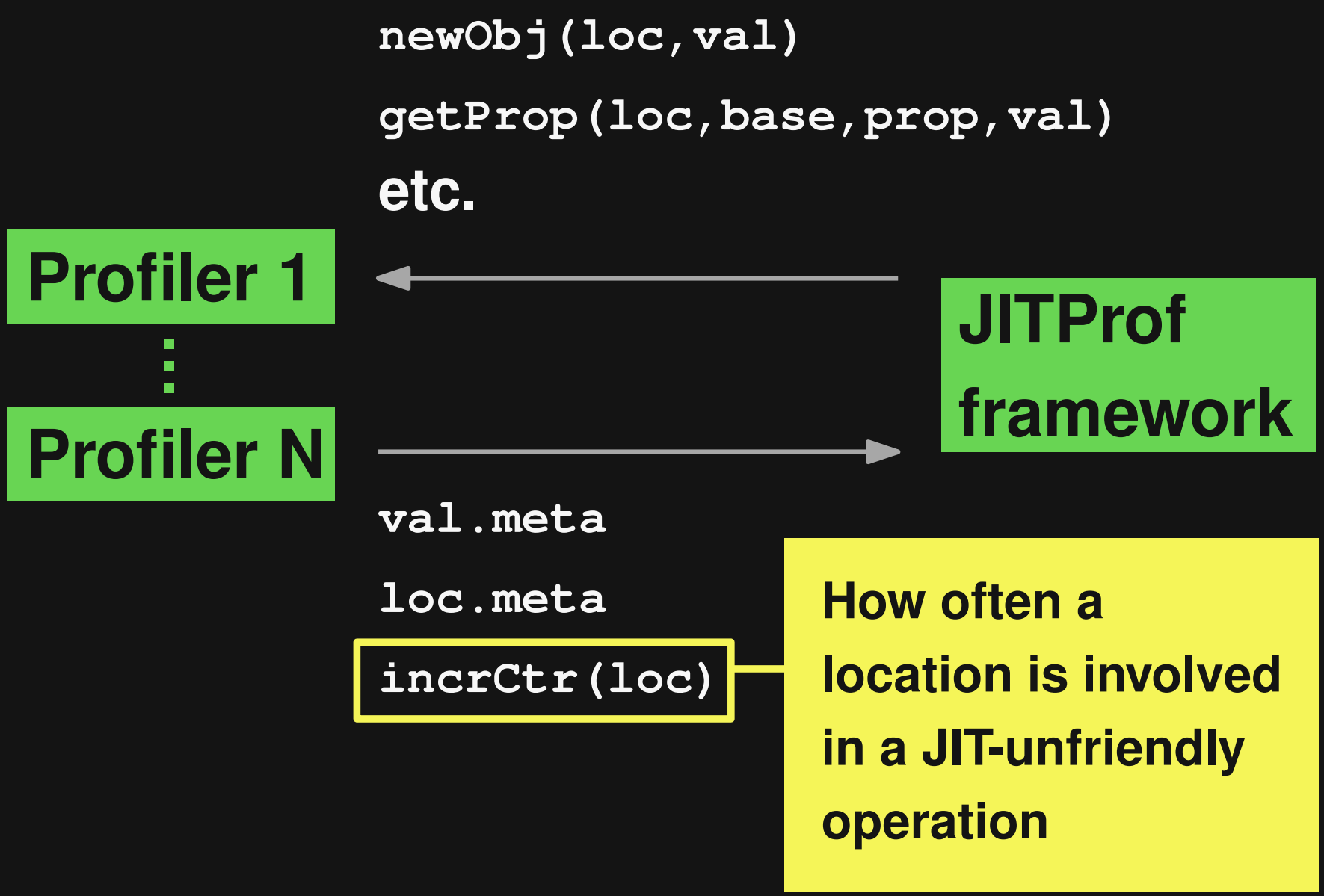
Framework and API



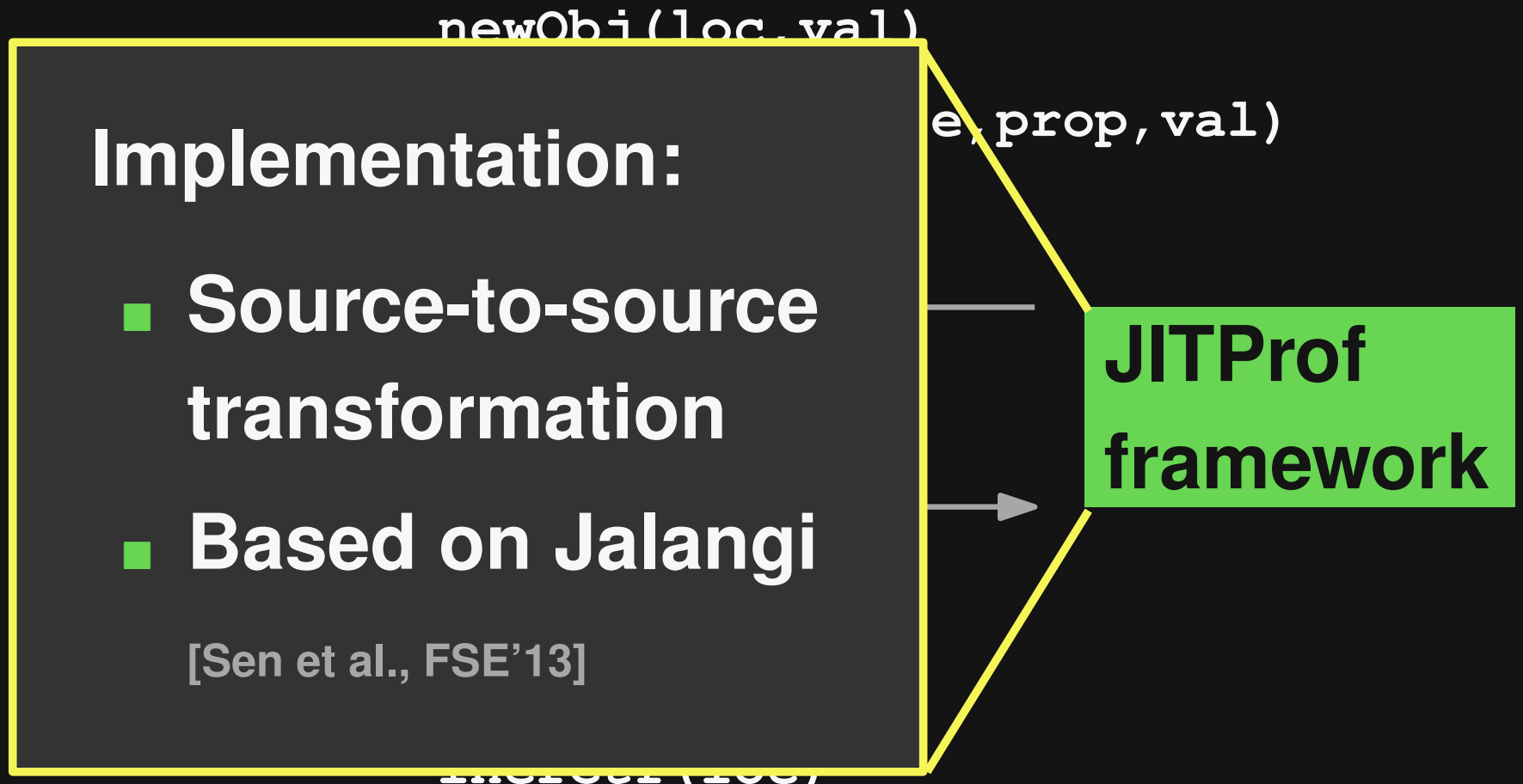
Framework and API



Framework and API



Framework and API



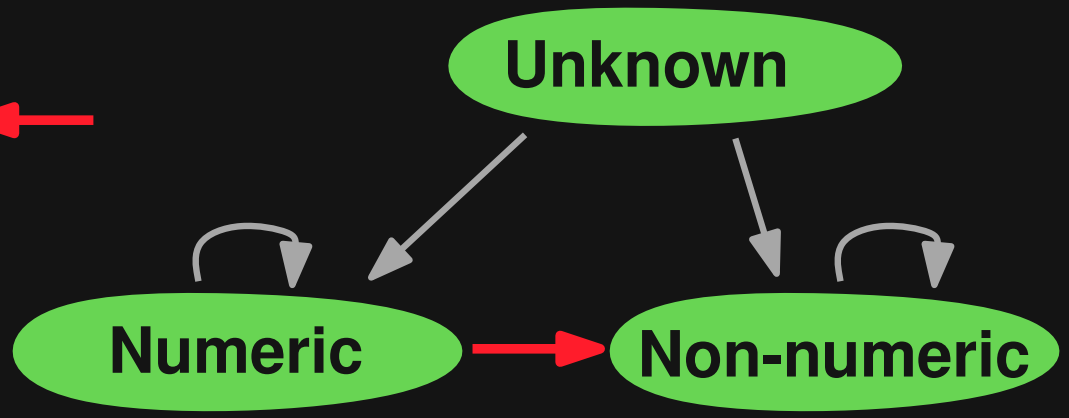
Profiler 1: Polymorphic Operations

```
function f(a, b) {
  return a + b;
}
f(2, 5);
f("ESEC", "FSE");
```

- Track types of operands of unary and binary operations
- Different than last time? Increment unfriendliness counter

Profiler 2: Mostly Numeric Arrays

```
var array = [];
for (var i = 0; i < 1000; i++) {
  array[i] = i;
}
array[4] = "abc";
```



- Track state of each array
- Numeric becomes non-numeric?
  Increment unfriendliness counter

Profiler 3: Inconsistent Obj. Layouts

```
x = {};
if (..) {
  x.a = "hi";
  x.b = "world";
} else {
  x.b = "world";
  x.a = "hi";
}
.. = x.a;
```

- Track hidden classes of objects and simulate inline caching
- Cache miss? Increment unfriendliness counter

Other Patterns and Profilers

- Binary operations on `undefined`
- Non-contiguous arrays
- Undefined array elements
- Unnecessary use of generic arrays

Other Patterns and Profilers

- Binary operations on `undefined`
- Non-contiguous arrays
- Undefined array elements
- Unnecessary use of generic arrays

53 – 278 lines of code per profiler

Easy to extend with new profilers

Sampling

Control overhead at two levels

- **Function level sampling**:
  Selectively run original or instrumented code
- **Instruction level sampling**:
  Selectively report events to profilers

Decaying sampling strategy

Evaluation

Setup

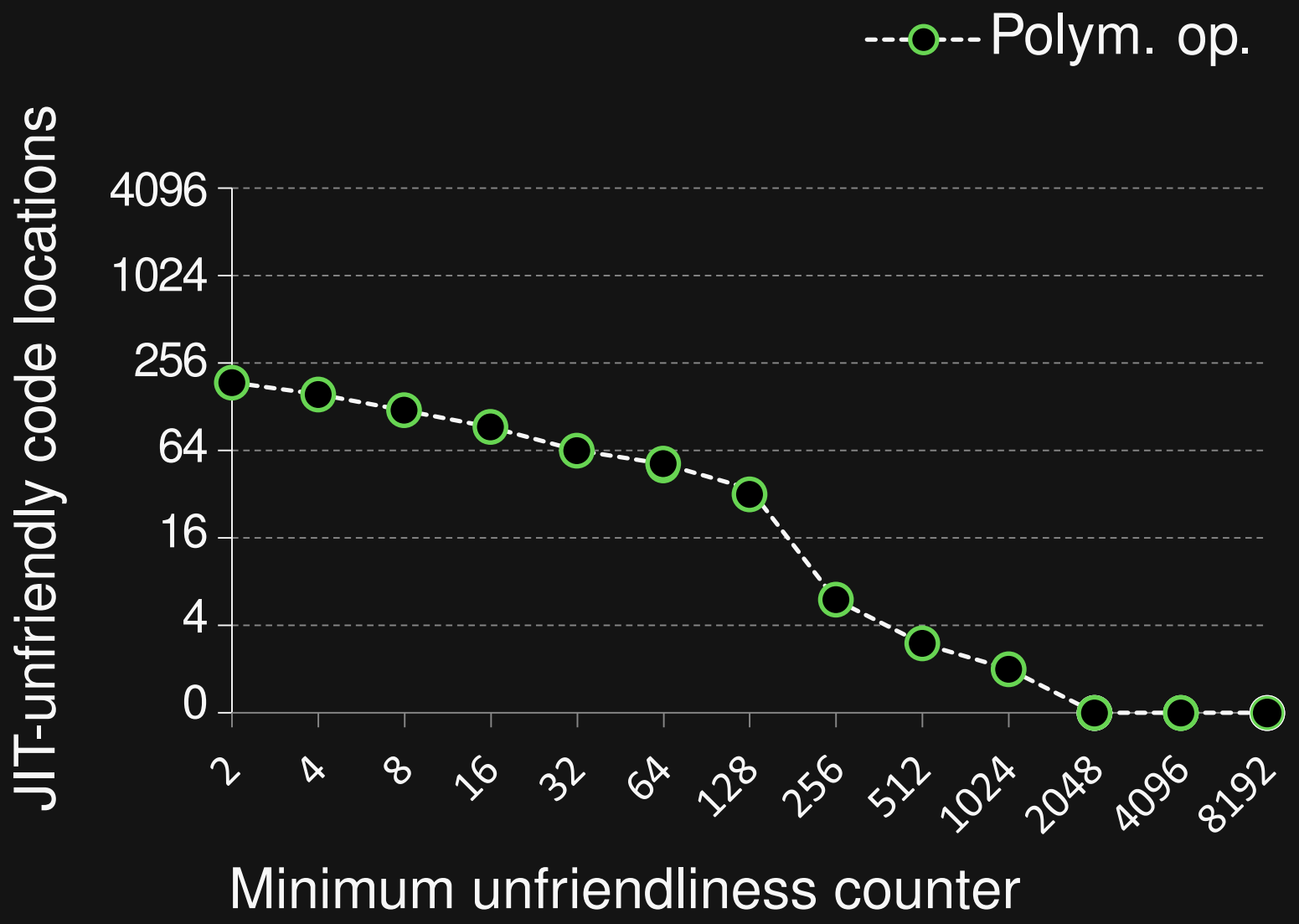
- Top 50 web sites
- Octane & SunSpider benchmarks
- Firefox + Chrome

Questions

- Prevalence of JIT-unfriendly code
- Effectiveness
- Compare: CPU profiling
- Overhead

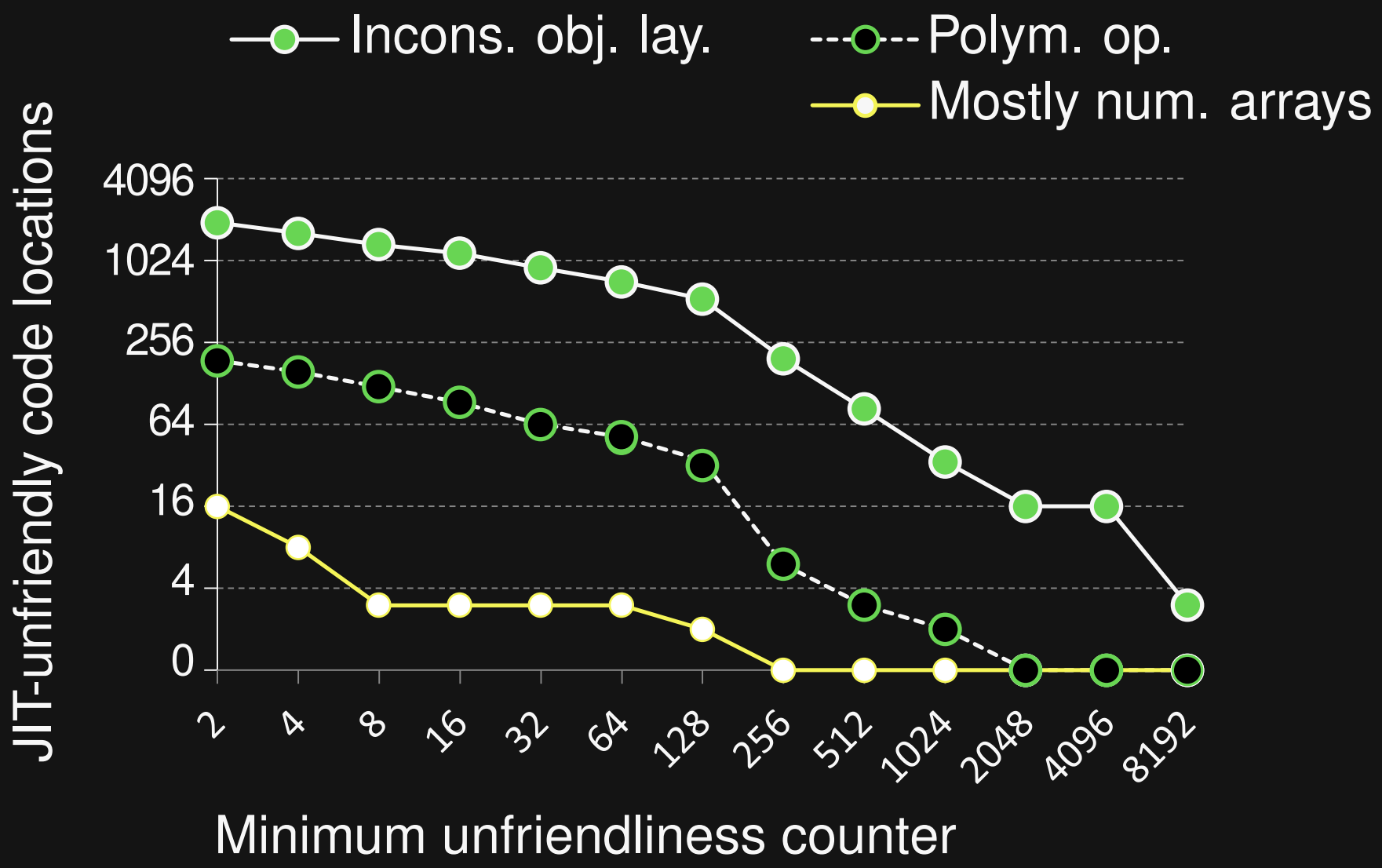
Prevalence of JIT-Unfriendly Code

Total of top 50 web sites



Prevalence of JIT-Unfriendly Code

Total of top 50 web sites



Optimization Opportunities

Found 15 optimization opportunities in benchmarks

Avoiding JIT-unfriendliness improves performance

- 1.1%–26.3% improvement (med.: 6.5%)
- 1–18 changed lines of code (med.: 2)

Examples (1)

```
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
  node.left = this.root_;
  node.right = this.root_.right;
  ...
} else {
  node.right = this.root_;
  node.left = this.root_.left;
  ...
}
```

Improvement:

- **3.5% in Firefox**
- **15.1% in Chrome**

Examples (2)

```
String.leftPad = function (val, size, ch) {
  var result = new String(val);

  if (ch == null) { ch = " "; }
  while (result.length < size) {
    result = ch + result;
  }
  return result;
}
```

Examples (2)

```
String.leftPad = function (val, size, ch) {
  var result = val + "";
  var tmp = new String(val);
  if (ch == null) { ch = " "; }
  while (result.length < size) {
    result = ch + result;
  }
  return result;
}
```

Improvement:

- 19.7% in Firefox
- 22.4% in Chrome

Comparison to CPU Profiling

	Rank of optimization opportunities	Precision
JITProf	1 or 2	Statement
CPU profilers	Higher for 76% of found opportunities	Function

Effect of Sampling

Overhead imposed by profiler

- Without sampling: 627x (median)
- **With sampling: 18x** (median)
  (no loss of optimization opportunities)

Acceptable for in-house performance analysis

Conclusion

JITProf:

Profiling to find JIT-unfriendly code

- Framework + 7 profilers
- Easily extensible
- Engine-independent

https://github.com/Berkeley-Correctness-Group/JITProf

Conclusion

JITProf:

Profiling to find JIT-unfriendly code

- Framework + 7 profilers
- Easily extensible
- Engine-independent

Thanks!

https://github.com/Berkeley-Correctness-Group/JITProf