The design and implementation of ZJIT & the next five years

Max Bernstein @ RubyKaigi 2026

Who I am

- **Cyclist**, climber, **baker**, reader, writer, teacher, **JIT compiler maker**
- Rubyist on and off since ~2010 (now committer!)
- ZJIT team cheerleader at Shopify

Bread

Joshua Tree National Park

Shimanami Kaido en route to RubyKaigi 2025!

Welcome to the first of several ZJIT talks

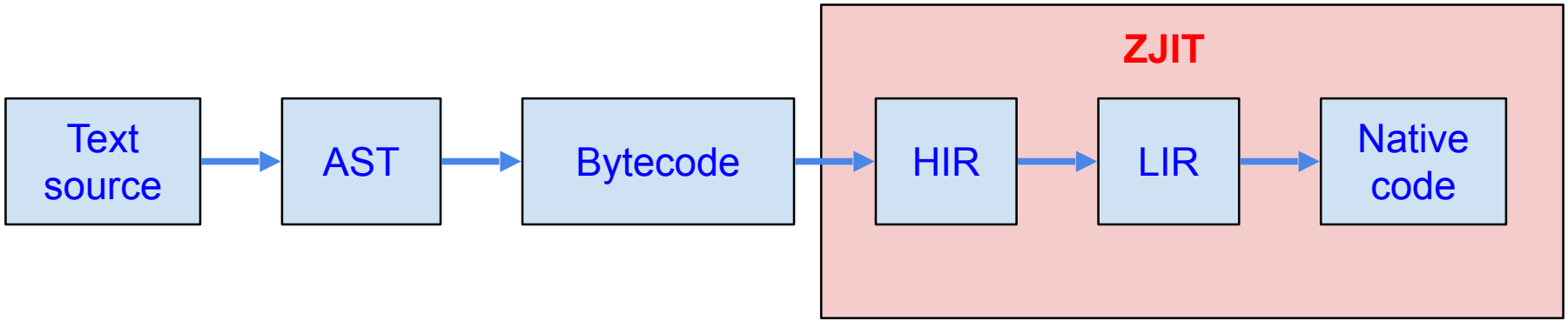
I’m going to talk about the big picture

- Brief overview of JITs
- The design and implementation of ZJIT
  - Brief differences from YJIT
  - What we changed since April 2025
  - Where these ideas come from
- & the next five years
  - What we are adding next
  - Where we want to be in five years

It’s going to be technical and hopefully understandable!

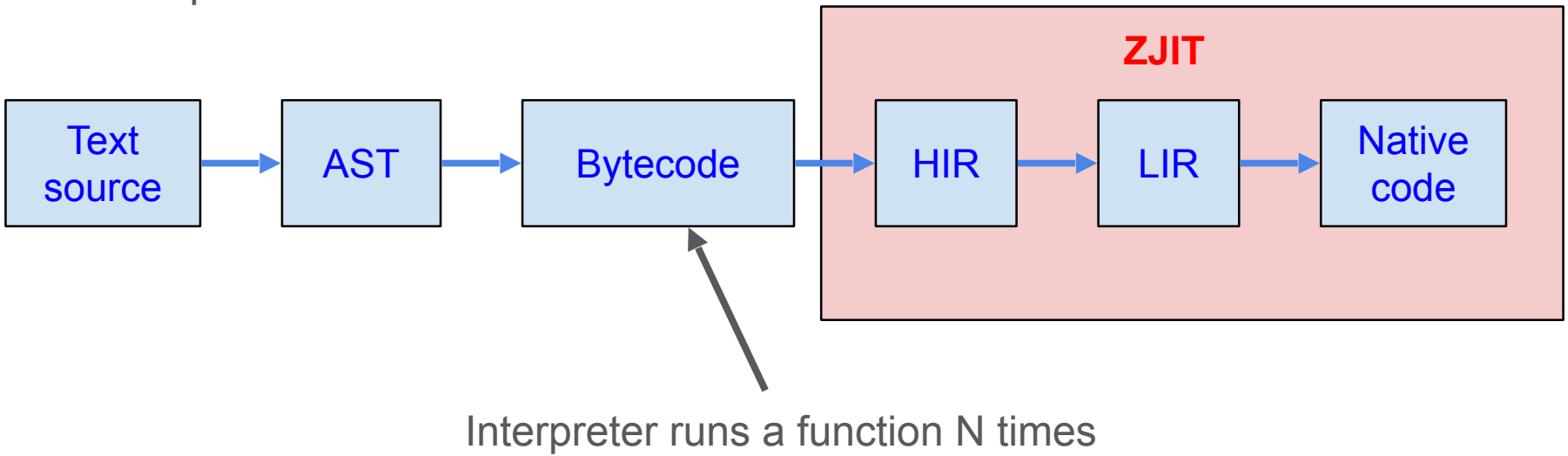
What is JIT?

- Compilation to native code happens while your code is running



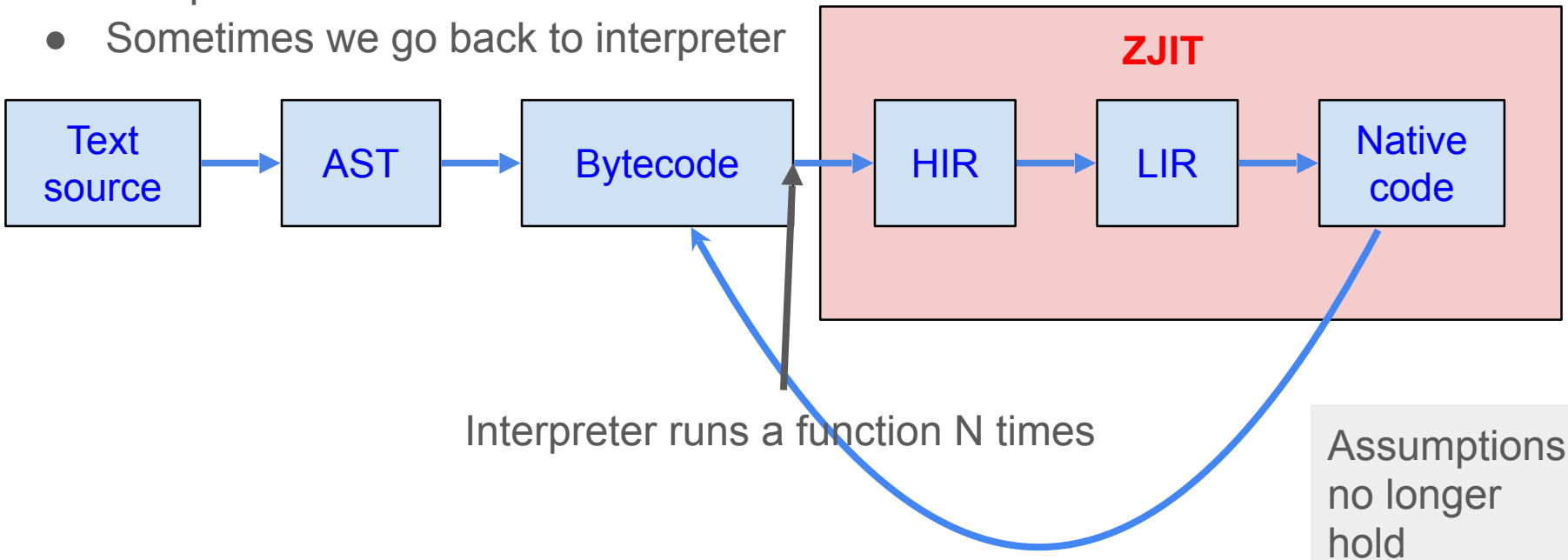
What is JIT?

- Compilation to native code happens while your code is running
- Interpreter runs first!



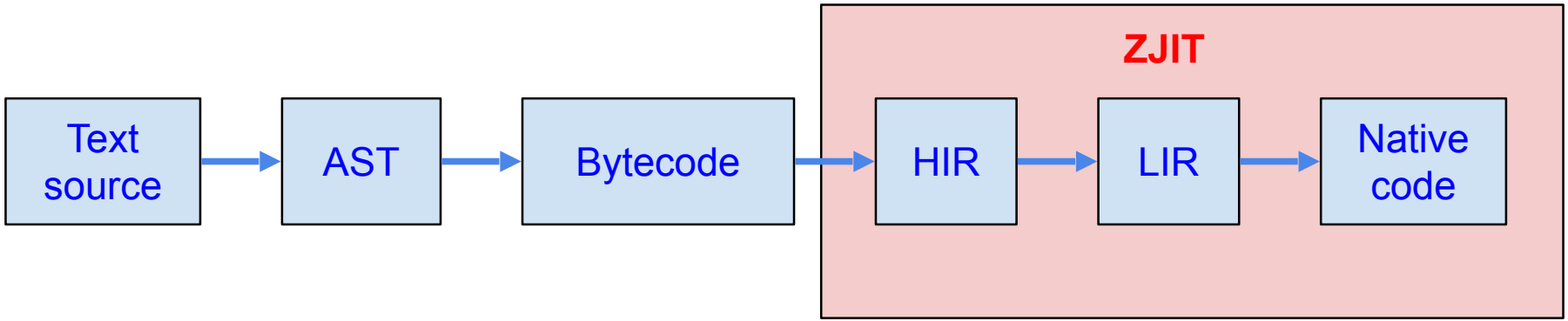
What is JIT?

- Compilation to native code happens while your code is running
- Interpreter runs first!
- Sometimes we go back to interpreter



What is JIT?

- Compilation to native code happens while your code is running



HIR: High level, conceptual, Ruby semantics
LIR: Low level, data locations, machine semantics

Partial comparison with YJIT

- HIR is new, high-level (“world of ideas”), static single assignment (SSA)
- Meant to be generated, optimized, cleaned up
- Supports new and exciting analysis, optimizations

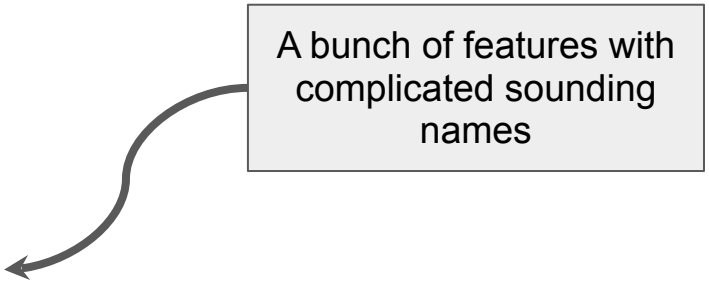
- Unlike YJIT’s (L)IR, which is low-level and mostly generated in one pass

We've been busy

- Exit from JIT code to interpreter
- More opcode support
- Lazy invalidation of code
- Re-compiling code with new information
- Much more strength reduction
- Specialize C functions into HIR
- Inline small Ruby functions
- Constant folding
- Monomorphic and polymorphic ivar caches
- Monomorphic and polymorphic method caches
- Effect analysis
- Redundant code elimination
- Lightweight frames
- Register allocation
- More Ruby in the stdlib
- Load-store forwarding
- Dead store elimination
- Partial static single information form
- Tracing and trace visualization

We've been busy

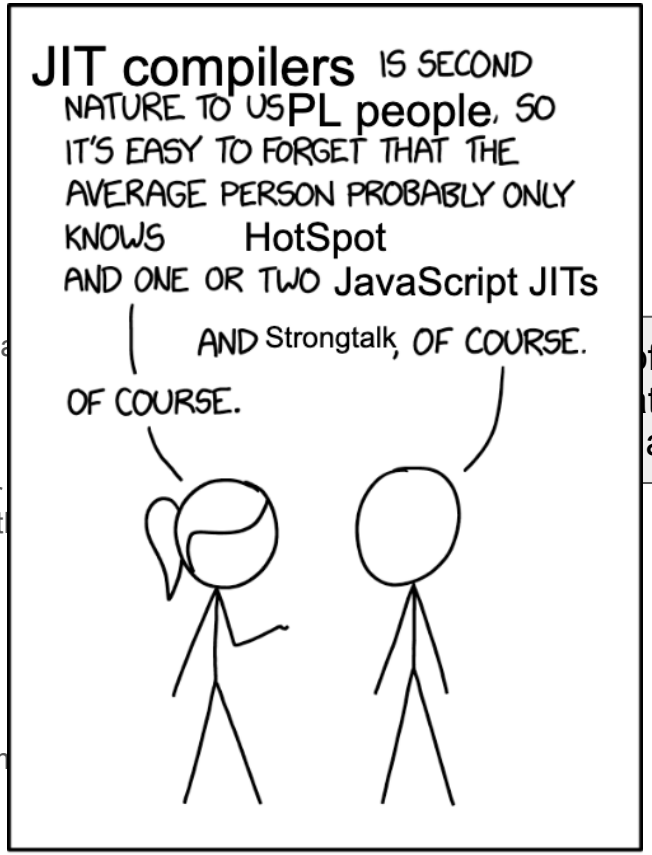
- Exit from JIT code to interpreter
- More opcode support
- Lazy invalidation of code
- Re-compiling code with new information
- Much more strength reduction
- Specialize C functions into HIR
- Inline small Ruby functions
- Constant folding
- Monomorphic and polymorphic ivar caches
- Monomorphic and polymorphic method caches
- Effect analysis
- Redundant code elimination
- Lightweight frames
- Register allocation
- More Ruby in the stdlib
- Load-store forwarding
- Dead store elimination
- Partial static single information form
- Tracing and trace visualization



We've been busy

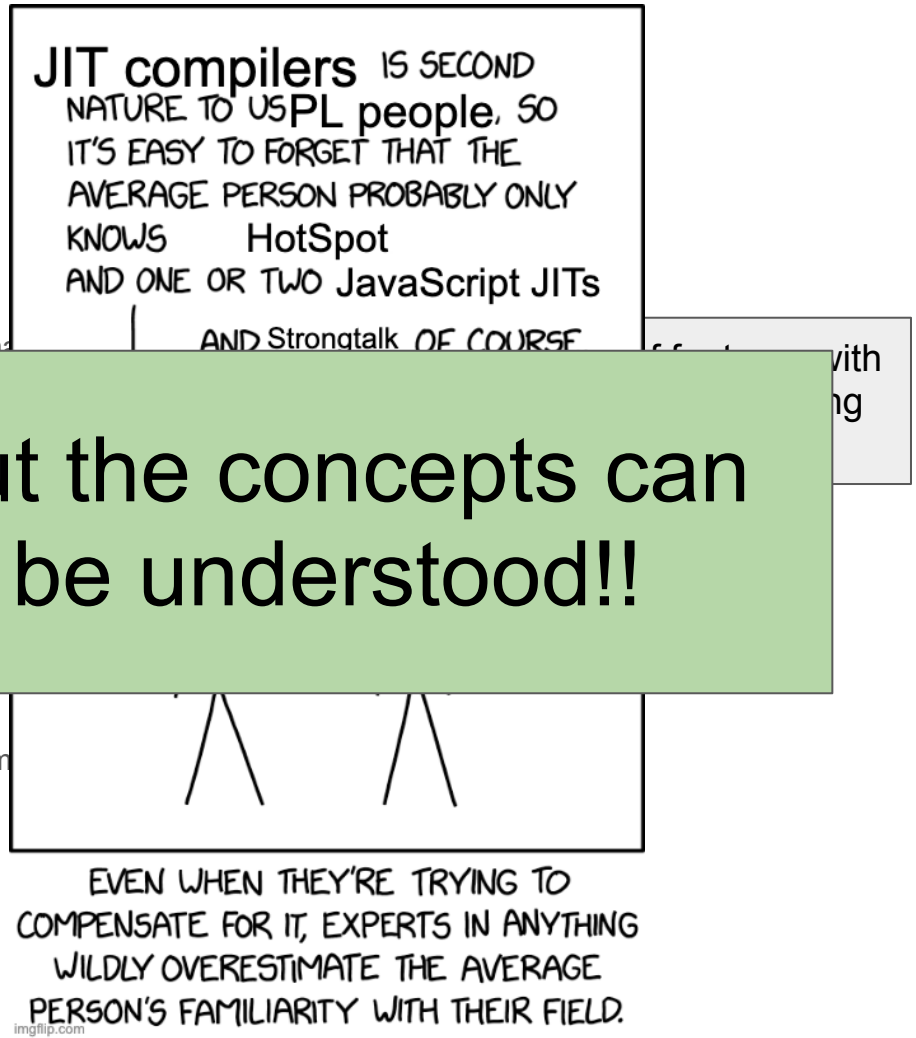
- Exit from JIT code to interpreter
- More opcode support
- Lazy invalidation of code
- Re-compiling code with new informa
- Much more strength reduction
- Specialize C functions into HIR
- Inline small Ruby functions
- Constant folding
- Monomorphic and polymorphic ivar
- Monomorphic and polymorphic met
- Effect analysis
- Redundant code elimination
- Lightweight frames
- Register allocation
- More Ruby in the stdlib
- Load-store forwarding
- Dead store elimination
- Partial static single information form
- Tracing and trace visualization

f features with
ted sounding
ames



We've been busy

- Exit from JIT code to interpreter
- More opcode support
- Lazy invalidation of code
- Re-compiling code with new informa
- Much more strength r
- Specialize C function
- Inline small Ruby fun
- Constant folding
- Monomorphic and po
- Monomorphic and po
- Effect analysis
- Redundant code elim
- Lightweight frames
- Register allocation
- More Ruby in the std
- Load-store forwarding
- Dead store elimination
- Partial static single information form
- Tracing and trace visualization



But the concepts can be understood!!

The design and implementation of ZJIT

Small motivation

- Instance variables are straightforward, right?
- Just a key-value store on each object
- Well,
  - Frozen-ness
  - Have to make storage fast
  - Interactions with Ractors
  - Interactions with Boxes

- Let’s look at some HIR to understand the mechanics…

```
1 class C
2   def initialize
3     @a = 1
4     @b = 2
5     @c = 3
6   end
7 end
```

Strategy

- Expand interpreter semantics into IR
- Turn very dynamic operations into static low-level operations
- Assumptions (Guard, PatchPoint) make sure expected patterns hold

```
1 class C
2   def initialize
3     @a = 1
4     @b = 2
5     @c = 3
6   end
7 end
```

HIR

@a @b @c

```
Block 3
  6 Param                                   BasicObject
  9 PatchPoint NoTracePoint
 10 Const Value(1)                          Fixnum[1]
 12 PatchPoint SingleRactorMode
 35 GuardType v6, HeapBasicObject           HeapBasicObject
 36 LoadField v35, :_shape_id@0x4           CShape
 37 GuardBitEquals v36, CShape(0x80000)     CShape[0x80000]
 38 StoreField v35, :@a@0x10, v10
 39 WriteBarrier v35, v10
 40 Const CShape(0x80095)                   CShape[0x80095]
 41 StoreField v35, :_shape_id@0x4, v40
 14 RefineType v6, HeapBasicObject          HeapBasicObject
 16 PatchPoint NoTracePoint
 17 Const Value(2)                          Fixnum[2]
 19 PatchPoint SingleRactorMode
 45 StoreField v14, :@b@0x18, v17
 46 WriteBarrier v14, v17
 47 Const CShape(0x80096)                   CShape[0x80096]
 48 StoreField v14, :_shape_id@0x4, v47
 21 RefineType v14, HeapBasicObject         HeapBasicObject
 23 PatchPoint NoTracePoint
 24 Const Value(3)                          Fixnum[3]
 27 PatchPoint SingleRactorMode
 52 StoreField v21, :@c@0x20, v24
 53 WriteBarrier v21, v24
 54 Const CShape(0x80097)                   CShape[0x80097]
 55 StoreField v21, :_shape_id@0x4, v54
 31 PatchPoint NoTracePoint
 32 CheckInterrupts
 33 Return v24
```

Strategy

- Expand interpreter semantics into IR
- Turn very dynamic operations into static low-level operations
- Assumptions (Guard, PatchPoint) make sure expected patterns hold

```
1 class C
2   def initialize
3     @a = 1
4     @b = 2
5     @c = 3
6   end
7 end
```

HIR

@a @b @c

Block 3

6	Param	BasicObject
9	PatchPoint NoTracePoint	
10	Const Value(1)	
12	PatchPoint SingleRactorMode	
35	GuardType v6, HeapBasicObject	
36	LoadField v35, :_shape_id@0x	
37	GuardBitEquals v36, CShape(0x80000)	CShape[0x80000]
38	StoreField v35, :@a@0x10, v10	
39	WriteBarrier v35, v10	
40	Const CShape(0x80095)	CShape[0x80095]
41	StoreField v35, :_shape_id@0x4, v40	
14	RefineType v6, HeapBasicObject	HeapBasicObject
16	PatchPoint NoTracePoint	
17	Const Value(2)	Fixnum[2]
19	PatchPoint SingleRactorMode	
45	StoreField v14, :@b@0x18, v17	
46	WriteBarrier v14, v17	
47	Const CShape(0x80096)	CShape[0x80096]
48	StoreField v14, :_shape_id@0x4, v47	
21	RefineType v14, HeapBasicObject	HeapBasicObject
23	PatchPoint NoTracePoint	
24	Const Value(3)	Fixnum[3]
27	PatchPoint SingleRactorMode	
52	StoreField v21, :@c@0x20, v24	
53	WriteBarrier v21, v24	
54	Const CShape(0x80097)	CShape[0x80097]
55	StoreField v21, :_shape_id@0x4, v54	
31	PatchPoint NoTracePoint	
32	CheckInterrupts	
33	Return v24	

Shape observed in interpreter

Strategy

- Expand interpreter semantics into IR
- Turn very dynamic operations into s
- A s

HIR

@a

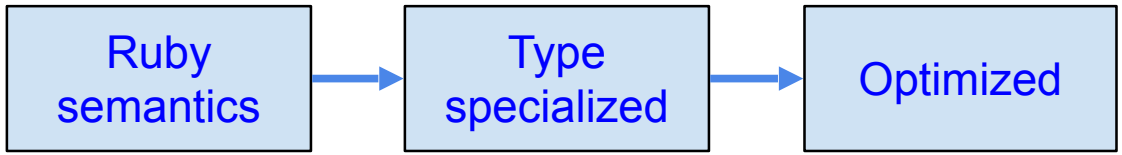
Block 3

6	Param	BasicObject
9	PatchPoint NoTracePoint	
10	Const Value(1)	
12	PatchPoint SingleRactorMode	
35	GuardType v6, HeapBasicObject	
36	LoadField v35, :_shape_id@0x	
37	GuardBitEquals v36, CShape(0x80000)	CShape[0x80000]
38	StoreField v35, :@a@0x10, v10	
39	WriteBarrier v35, v10	
40	Const CShape(0x80095)	CShape[0x80095]
41	StoreField v35, :_shape_id@0x4, v40	
14	RefineType v6, HeapBasicObject	HeapBasicObject
16	PatchPoint NoTracePoint	
17	Const Value(2)	Fixnum[2]
19	PatchPoint SingleRactorMode	
45	StoreField v14, :@b@0x18, v17	
46	WriteBarrier v14, v17	
47	Const CShape(0x80096)	CShape[0x80096]
48	StoreField v14, :_shape_id@0x4, v47	
21	RefineType v14, HeapBasicObject	HeapBasicObject
23	PatchPoint NoTracePoint	
24	Const Value(3)	Fixnum[3]
27	PatchPoint SingleRactorMode	
52	StoreField v21, :@c@0x20, v24	
53	WriteBarrier v21, v24	
54	Const CShape(0x80097)	CShape[0x80097]
55	StoreField v21, :_shape_id@0x4, v54	
31	PatchPoint NoTracePoint	
32	CheckInterrupts	
33	Return v24	

Shape observed in interpreter

Strategy

- Re-use dynamic language specialization from the OGs (Smalltalk-80, Self, Strongtalk, V8, JavaScriptCore, PyPy)
  - Object shapes
  - Speculation (monomorphic and polymorphic)
  - Storage strategies
  - Inlining
  - Fast calls
- Then re-use much of design from Java JITs (Jikes, C1, C2, Graal)
  - HIR/LIR design
  - Value numbering
  - Constant/branch folding
  - Register allocator
  - Compile policy



We'll come back to this with specific "canary" examples…

What we’re working on now

Now

- Currently working on method inlining prototype
  - Maybe Kevin will give a talk next year
- Currently working on making “lightweight frames”
  - See Kokubun’s talk
- Currently working on more memory optimization
  - See Jacob’s talk
- Speeding up allocation
  - Maybe John will give a talk next year
- FFI
  - See Aaron’s talk
- More tracing & profiling

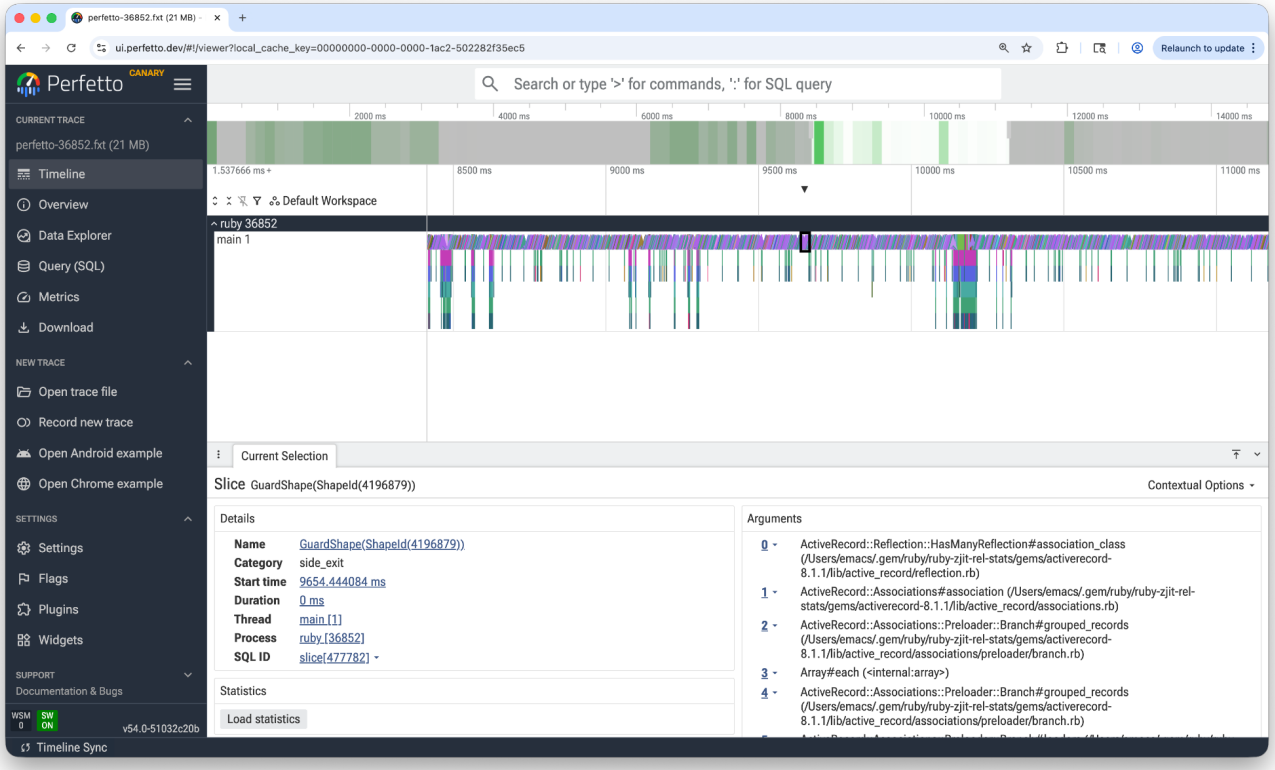
Tracing: where do we focus our attention?

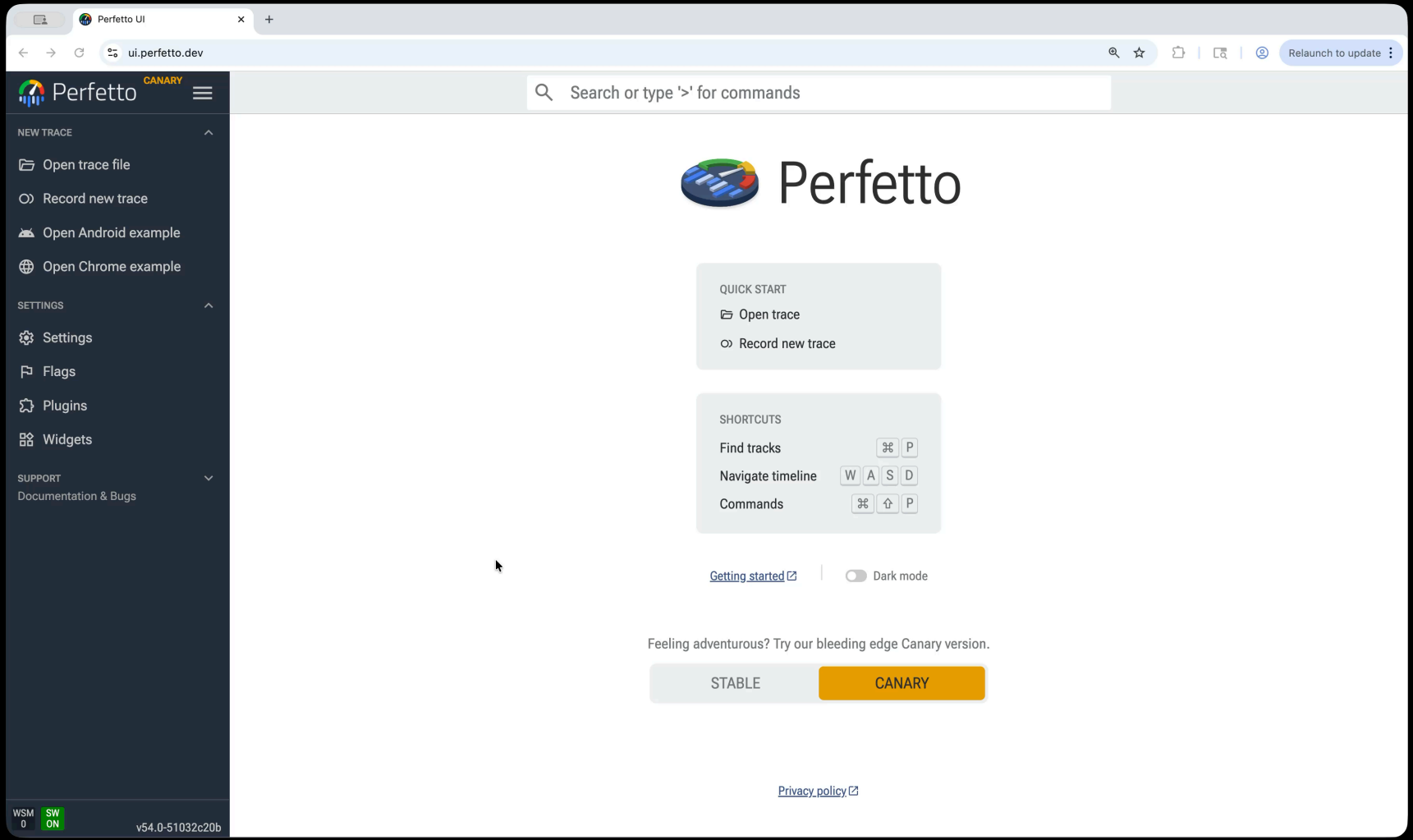
Many

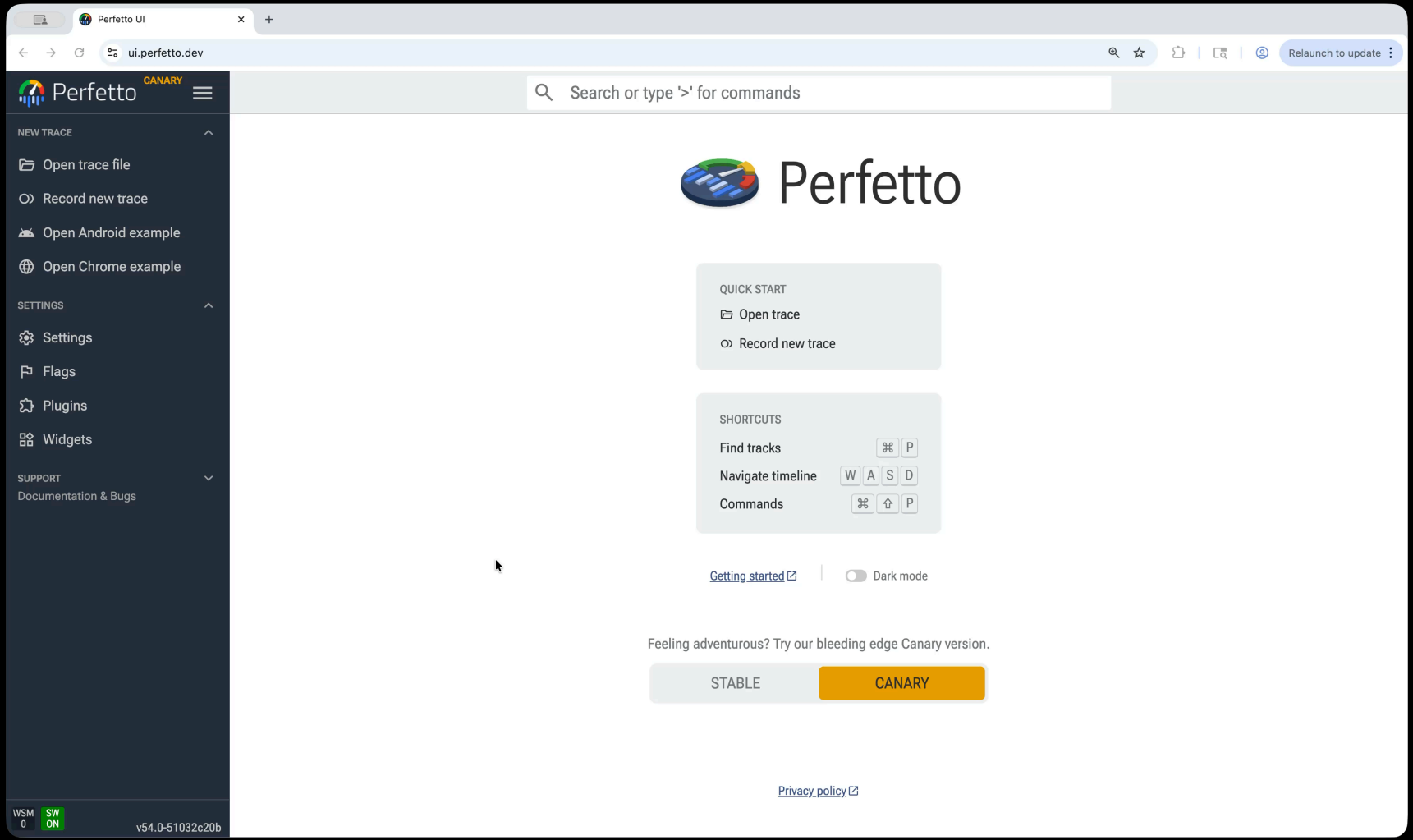
Slow

Events

:(




Perfetto UI
ui.perfetto.dev
Relaunch to update
Perfetto
CANARY
Search or type '>' for commands
NEW TRACE
Open trace file
Record new trace
Open Android example
Open Chrome example
SETTINGS
Settings
Flags
Plugins
Widgets
SUPPORT
Documentation & Bugs
Perfetto
QUICK START
Open trace
Record new trace
SHORTCUTS
Find tracks
Navigate timeline
Commands
Getting started
Dark mode
Feeling adventurous? Try our bleeding edge Canary version.
STABLE
CANARY
Privacy policy
WSM 0
SW ON
v54.0-51032c20b


Perfetto UI
ui.perfetto.dev
Relaunch to update
Perfetto
CANARY
Search or type '>' for commands
NEW TRACE
Open trace file
Record new trace
Open Android example
Open Chrome example
SETTINGS
Settings
Flags
Plugins
Widgets
SUPPORT
Documentation & Bugs
Perfetto
QUICK START
Open trace
Record new trace
SHORTCUTS
Find tracks
Navigate timeline
Commands
Getting started
Dark mode
Feeling adventurous? Try our bleeding edge Canary version.
STABLE
CANARY
Privacy policy
WSM 0
SW ON
v54.0-51032c20b

Goal: at least 5x speedup

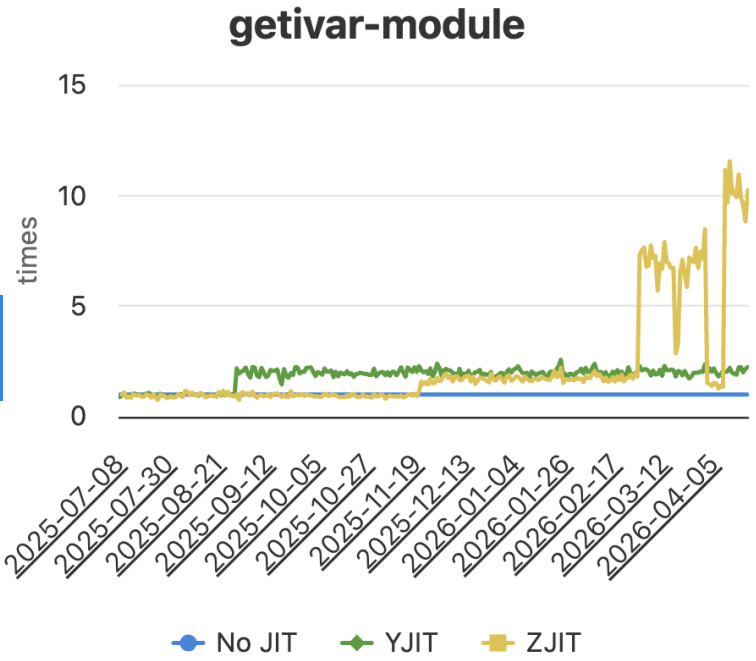

12 MONTHS. 6 ENGINEERS.
A DYNAMIC LANGUAGE COMPILER.
WHAT COULD POSSIBLY
GO WRONG?
ICE CUBE
ARE WE
THERE
YET?
REVOLUTION STUDIOS PRESENTS A CUBE VISION PRODUCTION A FILM BY BRIAN LEVANT STARRING ICE CUBE "ARE WE THERE YET?" NIA LONG

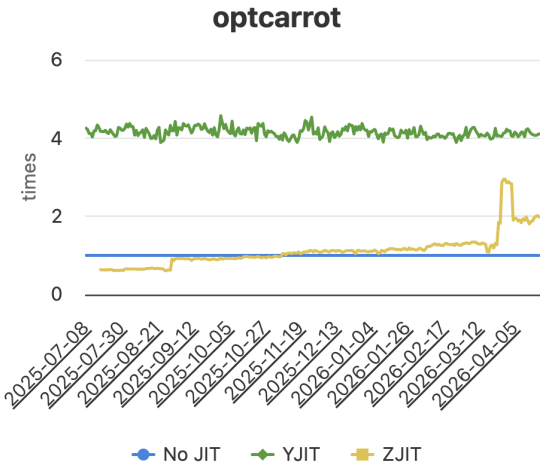
Some benchmarks (higher is better)

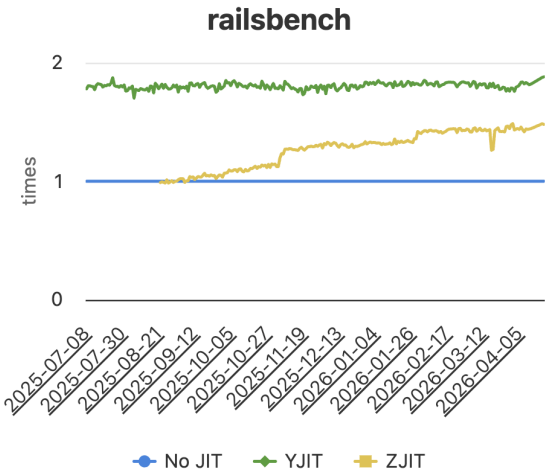
ZJIT

YJIT

Interpreter







Graphs tell part of the story, so…

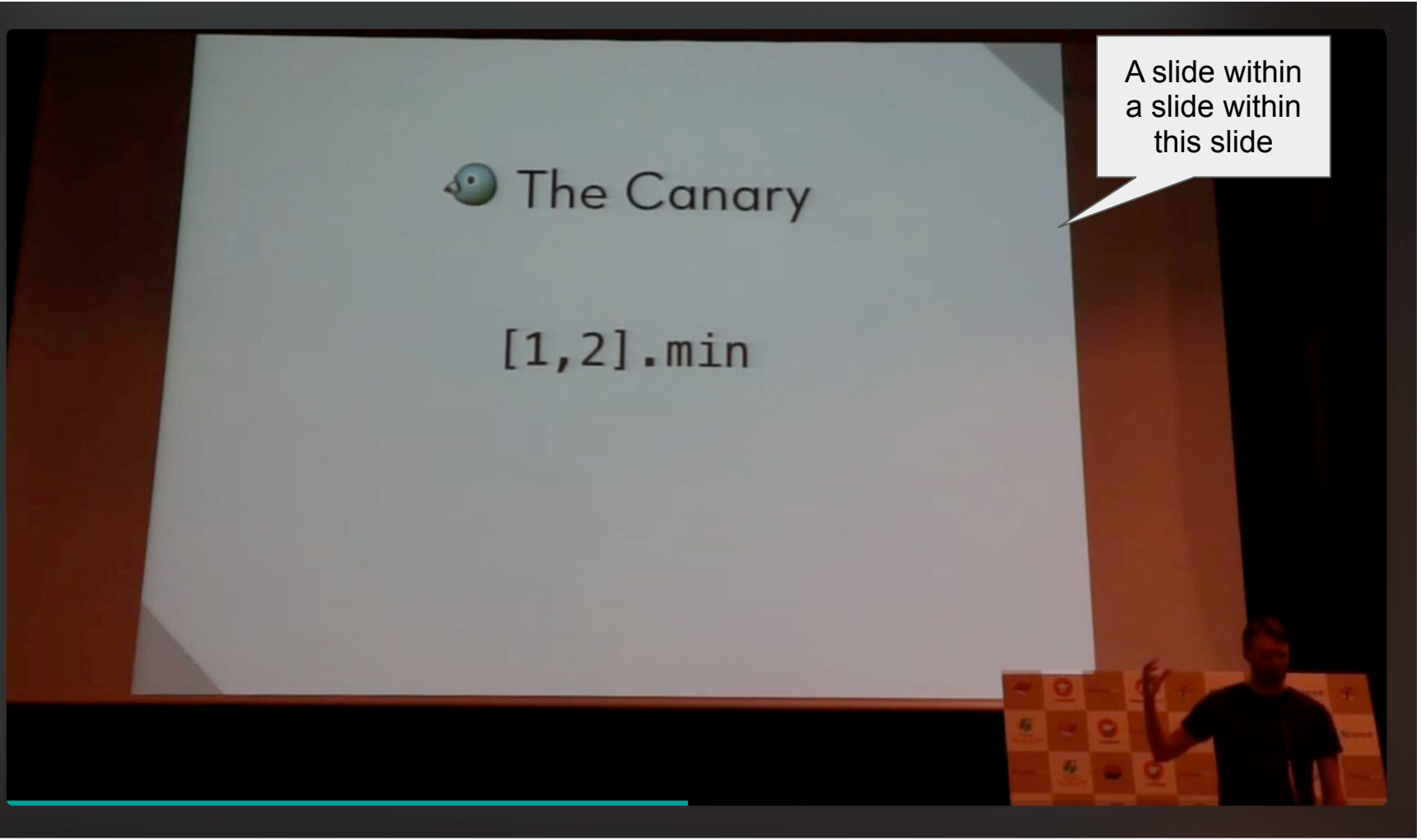
Let’s go back in time to 2022.

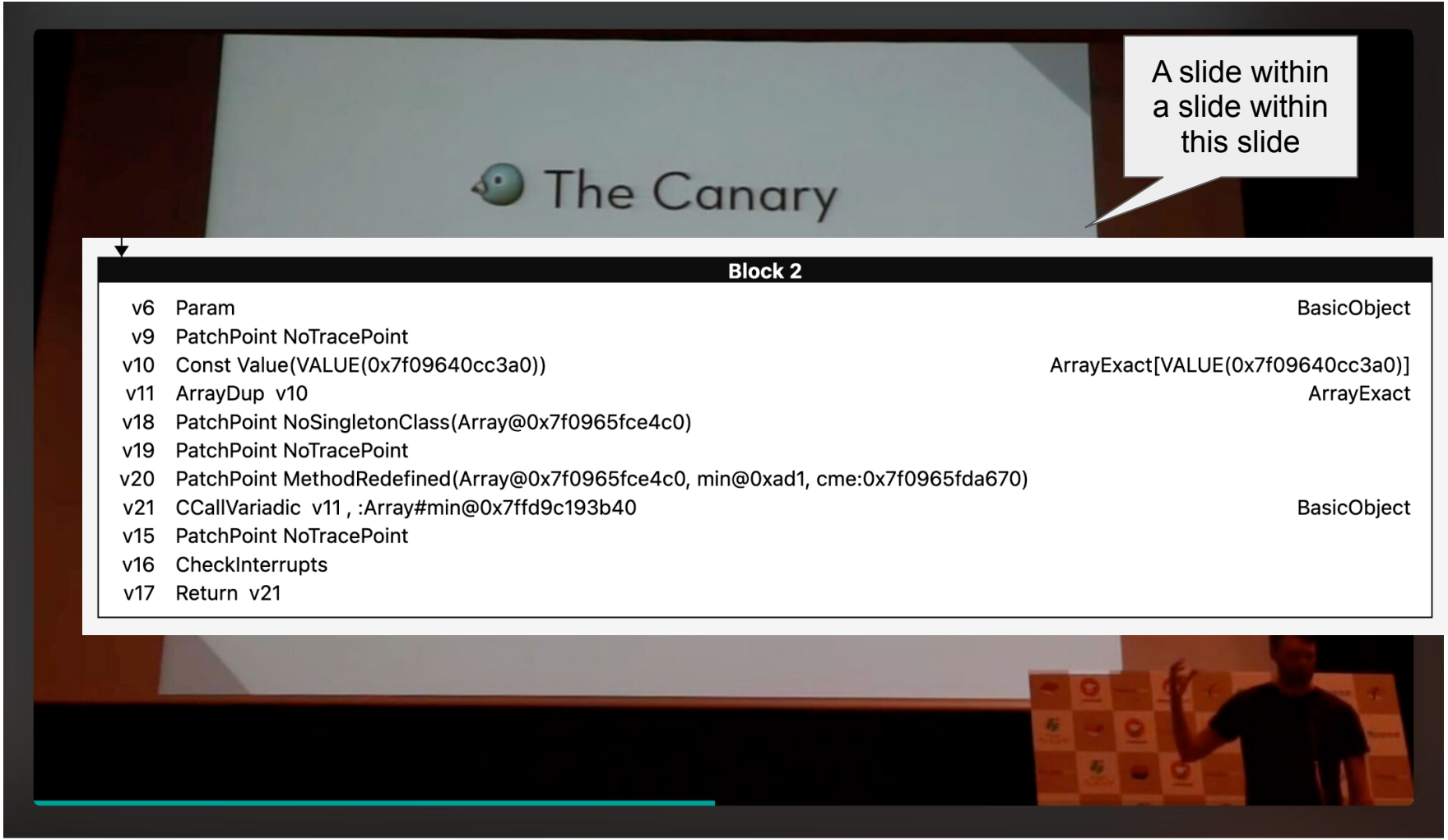
My wish on Ruby 4 JIT

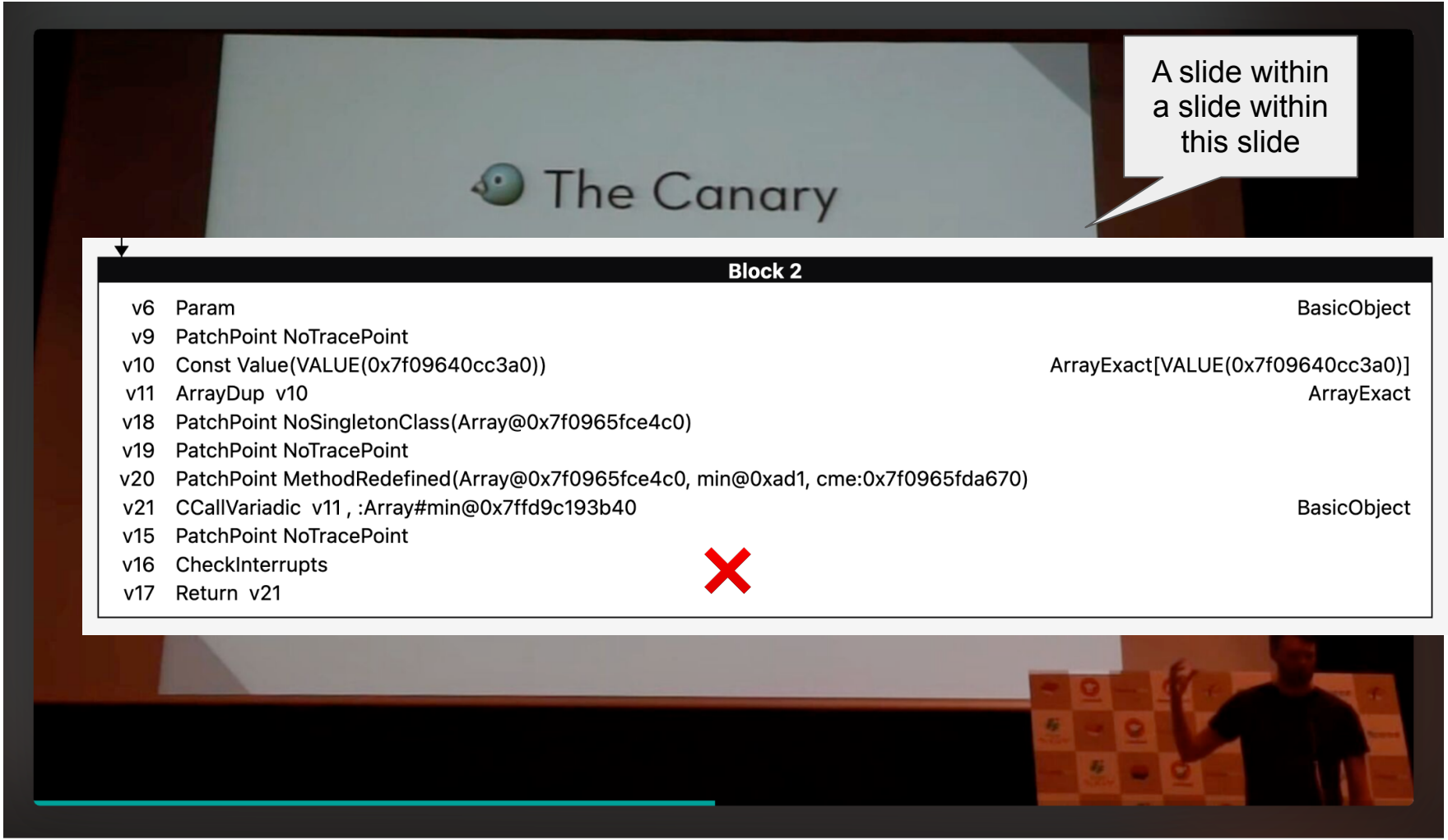
- I want Ruby 4 to be as fast as **Java** or **JavaScript**
- Ruby 4's performance should be a reason to leave **Python**

From Kokubun

Now we’re going to look at a lot of HIR


A slide within a slide within this slide
The Canary
[1,2].min


The Canary
A slide within a slide within this slide
Block 2
v6 Param BasicObject
v9 PatchPoint NoTracePoint
v10 Const Value(VALUE(0x7f09640cc3a0)) ArrayExact[VALUE(0x7f09640cc3a0)]
v11 ArrayDup v10 ArrayExact
v18 PatchPoint NoSingletonClass(Array@0x7f0965fce4c0)
v19 PatchPoint NoTracePoint
v20 PatchPoint MethodRedefined(Array@0x7f0965fce4c0, min@0xad1, cme:0x7f0965fda670)
v21 CCallVariadic v11 , :Array#min@0x7ffd9c193b40 BasicObject
v15 PatchPoint NoTracePoint
v16 CheckInterrupts
v17 Return v21


The Canary
A slide within a slide within this slide
Block 2
v6 Param BasicObject
v9 PatchPoint NoTracePoint
v10 Const Value(VALUE(0x7f09640cc3a0)) ArrayExact[VALUE(0x7f09640cc3a0)]
v11 ArrayDup v10 ArrayExact
v18 PatchPoint NoSingletonClass(Array@0x7f0965fce4c0)
v19 PatchPoint NoTracePoint
v20 PatchPoint MethodRedefined(Array@0x7f0965fce4c0, min@0xad1, cme:0x7f0965fda670)
v21 CCallVariadic v11 , :Array#min@0x7ffd9c193b40 BasicObject
v15 PatchPoint NoTracePoint
v16 CheckInterrupts
v17 Return v21

Ruby 4 Canary

```ruby
ONE = 1
def two = 2

def canary
  [ONE, two].include?(1)
end
```

- true is mov-ed (immediate)
- No opt_* VM instruction
- Constant folding
- Ruby / C method inlining

From Kokubun

Ruby 4 Canary

Block 2

v6	Param	BasicObject
v9	PatchPoint NoTracePoint	
v24	PatchPoint SingleRactorMode	
v25	PatchPoint StableConstantNames(0x5573b44e5bb0, ONE)	
v26	Const Value(1)	Fixnum[1]
v27	PatchPoint NoSingletonClass(Object@0x7fc3b9e3ed10)	
v28	PatchPoint MethodRedefined(Object@0x7fc3b9e3ed10, two@0xefc1, cme:0x7fc39eb0c160)	
v29	GuardType v6 , HeapObject[class_exact*:Object@VALUE(0x7fc3b9e3ed10)]	HeapObject[class_exact*:Object@VALUE(0x7fc3b9e3ed10)]
v31	IncrCounter inline_iseq_optimized_send_count	
v32	Const Value(2)	Fixnum[2]
v16	Const Value(1)	Fixnum[1]
v18	PatchPoint BOPRedefined(ARRAY_REDEFINED_OP_FLAG, BOP_INCLUDE_P)	
v19	ArrayInclude v26 , v32 | v16	BoolExact
v21	PatchPoint NoTracePoint	
v22	CheckInterrupts	
v23	Return v19	

From Kokubun

Ruby 4 Canary

Block 2

v6	Param	BasicObject
v9	PatchPoint NoTracePoint	
v24	PatchPoint SingleRactorMode	
v25	PatchPoint StableConstantNames(0x5573b44e5bb0, ONE)	
v26	Const Value(1)	Fixnum[1]
v27	PatchPoint NoSingletonClass(Object@0x7fc3b9e3ed10)	
v28	PatchPoint MethodRedefined(Object@0x7fc3b9e3ed10, two@0xefc1, cme:0x7fc39eb0c160)	
v29	GuardType v6 , HeapObject[class_exact*:Object@VALUE(0x7fc3b9e3ed10)]	HeapObject[class_exact*:Object@VALUE(0x7fc3b9e3ed10)]
v31	IncrCounter inline_iseq_optimized_send_count	
v32	Const Value(2)	Fixnum[2]
v16	Const Value(1)	Fixnum[1]
v18	PatchPoint BOPRedefined(ARRAY_REDEFINED_OP_FLAG, BOP_INCLUDE_P)	
v19	ArrayInclude v26 , v32 | v16	BoolExact
v21	PatchPoint NoTracePoint	
v22	CheckInterrupts	
v23	Return v19	

✗ (kinda)

From Kokubun

Ruby 4 Canary'

```ruby
@one = 1

def canary(two = 2)
  [@one, two].include?(1)
end
```

- Single branch instruction to access **@one**
- Single register to access **two**
- No heap allocation
- No stack frame

From Kokubun

Ruby 4 Canary'

Block 4

v23	Param	BasicObject
v24	Param	BasicObject
v27	PatchPoint NoTracePoint	
v28	PatchPoint SingleRactorMode	
v42	GuardType v23 , HeapBasicObject	HeapBasicObject
v43	LoadField v42 , :_shape_id@0x4	CShape
v44	GuardBitEquals v43 , CShape(0x400090)	CShape[0x400090]
v45	LoadField v42 , :@one@0x10	BasicObject
v32	PatchPoint NoEPEscape(canary)	
v34	Const Value(1)	Fixnum[1]
v36	PatchPoint BOPRedefined(ARRAY_REDEFINED_OP_FLAG, BOP_INCLUDE_P)	
v37	ArrayInclude v45 , v24 | v34	BoolExact
v39	PatchPoint NoTracePoint	
v40	CheckInterrupts	
v41	Return v37	

From Kokubun

Ruby 4 Canary'

Block 4

v23	Param	BasicObject
v24	Param	BasicObject
v27	PatchPoint NoTracePoint	
v28	PatchPoint SingleRactorMode	
v42	GuardType v23 , HeapBasicObject	HeapBasicObject
v43	LoadField v42 , :_shape_id@0x4	CShape
v44	GuardBitEquals v43 , CShape(0x400090)	CShape[0x400090]
v45	LoadField v42 , :@one@0x10	BasicObject
v32	PatchPoint NoEPEscape(canary)	
v34	Const Value(1)	Fixnum[1]
v36	PatchPoint BOPRedefined(ARRAY_REDEFINED_OP_FLAG, BOP_INCLUDE_P)	
v37	ArrayInclude v45 , v24 | v34	BoolExact
v39	PatchPoint NoTracePoint	
v40	CheckInterrupts	
v41	Return v37	

⚠️🤷😅 (kinda)

From Kokubun

Example

- Ruby code:

```
def loop
  i = 0; while i < 100_000; i += 1; end
  i
end
```

- GCC optimized x86-64 code:

```
...
movl   $200001, %eax
...
ret
```

- There is no loop
- JVM can not do this

From Kokubun

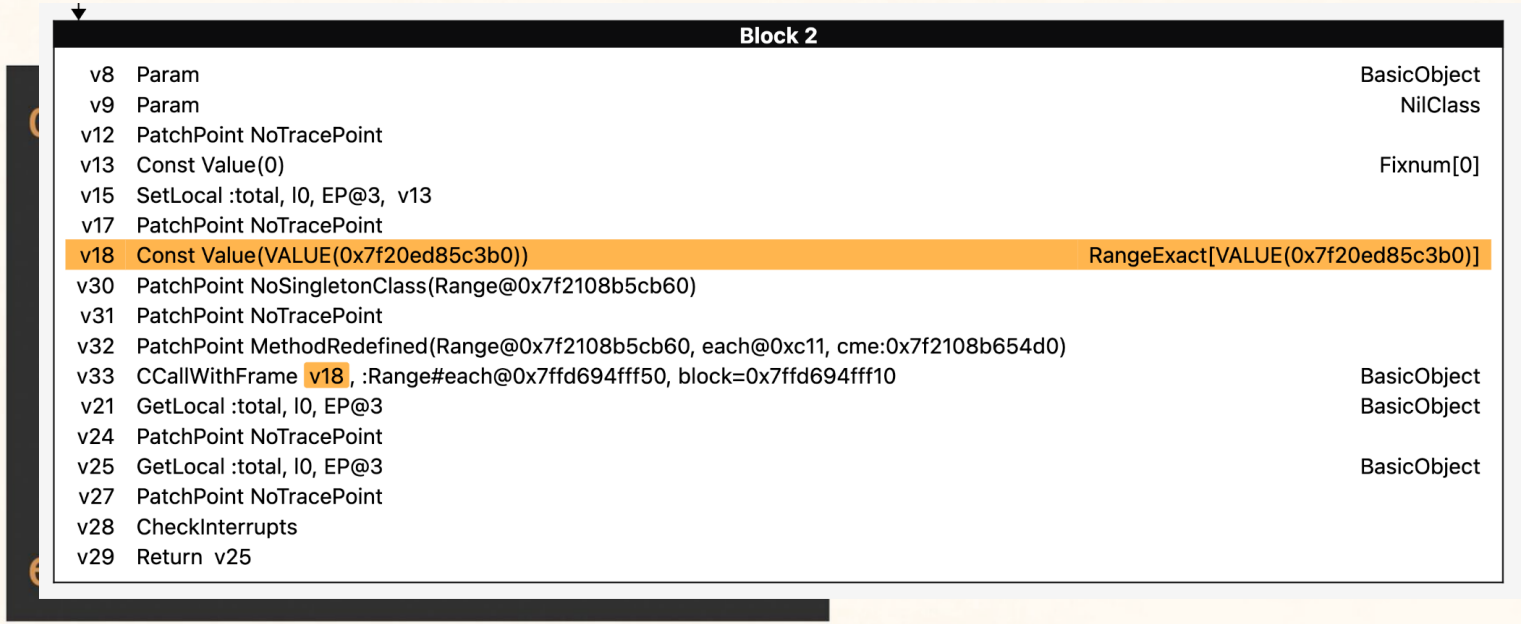
Ruby 4 Canary 2

```ruby
def canary2
  total = 0
  (1..100_000).each do |i|
    total += i
  end
  total
end
```

- 5000050000 is mov-ed (immediate)
- Ruby -> C -> Ruby inlining

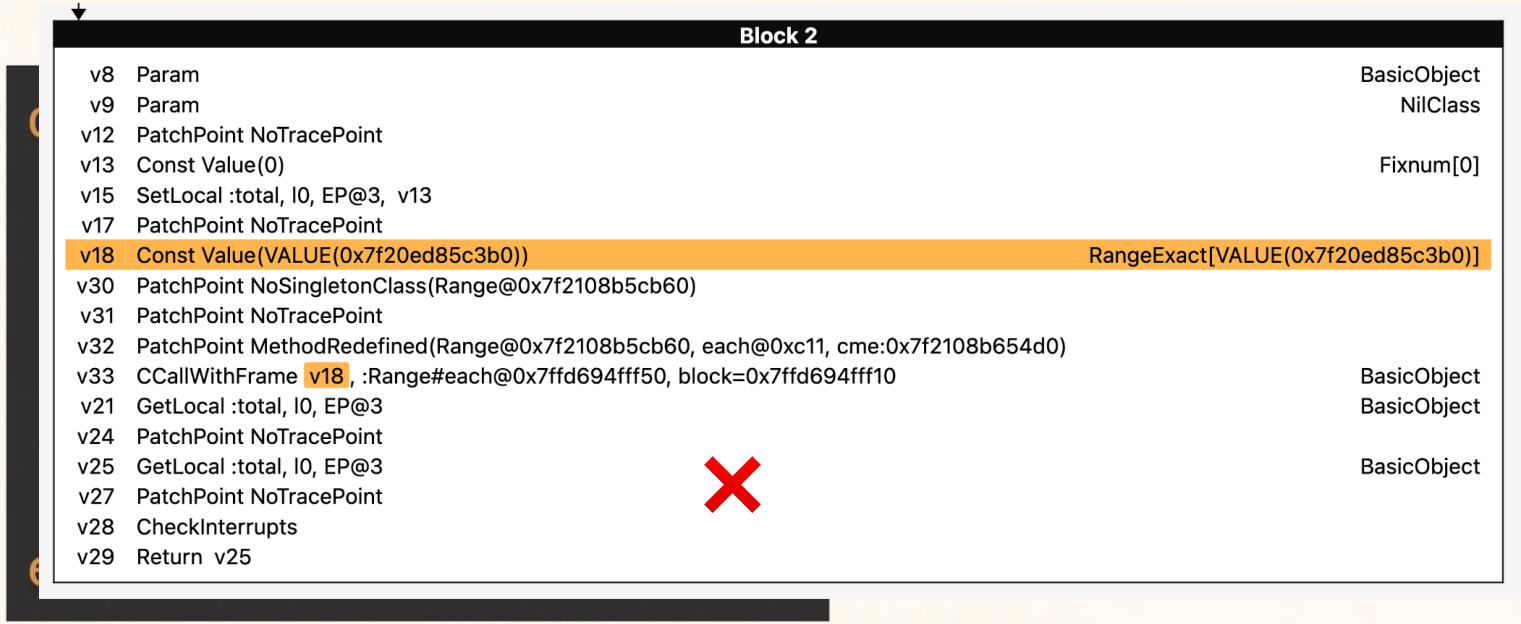
From Kokubun

Ruby 4 Canary 2



From Kokubun

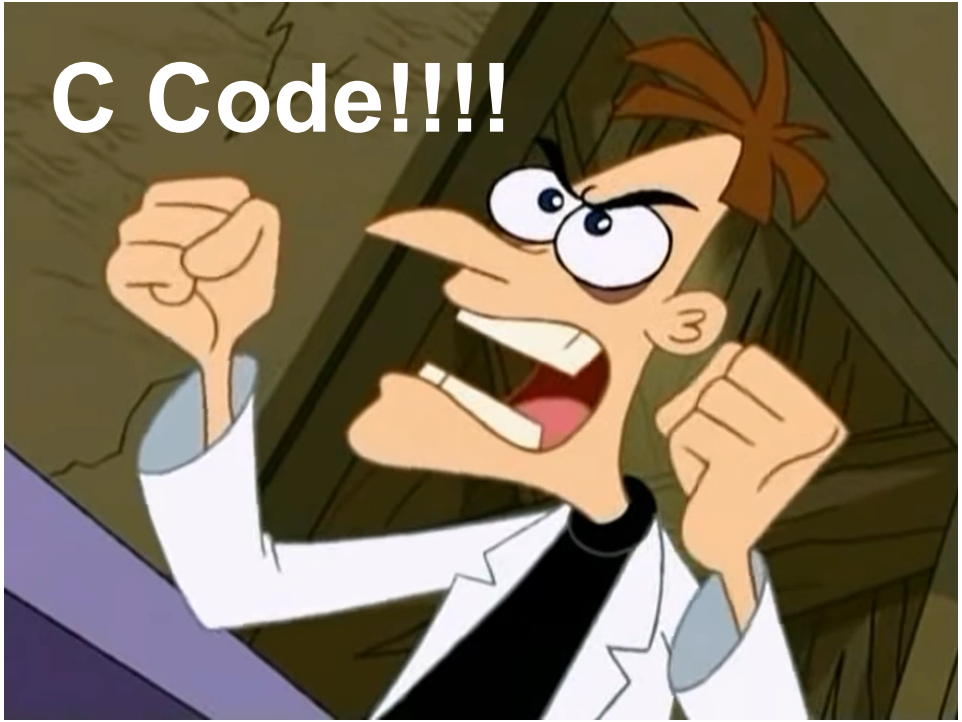
Ruby 4 Canary 2



From Kokubun

```
static VALUE
range_each(VALUE range)
{
    VALUE beg, end;
    long i;

    RETURN_SIZED_ENUMERATOR(range, 0, 0, range_enum_size);

    beg = RANGE_BEG(range);
    end = RANGE_END(range);

    if (FIXNUM_P(beg) && NIL_P(end)) {
        range_each_fixnum_endless(beg);
    }
    else if (FIXNUM_P(beg) && FIXNUM_P(end)) { /* fixnums are special */
        return range_each_fixnum_loop(beg, end, range);
    }
    else if (RB_INTEGER_TYPE_P(beg) && (NIL_P(end) || RB_INTEGER_TYPE_P(end))) {
        if (SPECIAL_CONST_P(end) || RBIGNUM_POSITIVE_P(end)) { /* end >= FIXNUM_MIN */
            if (!FIXNUM_P(beg)) {
                if (RBIGNUM_NEGATIVE_P(beg)) {
                    do {
                        rb_yield(beg);
                    } while (!FIXNUM_P(beg = rb_big_plus(beg, INT2FIX(1))));
                    if (NIL_P(end)) range_each_fixnum_endless(beg);
                    if (FIXNUM_P(end)) return range_each_fixnum_loop(beg, end, range);
                }
                else {
                    if (NIL_P(end)) range_each_bignum_endless(beg);
                    if (FIXNUM_P(end)) return range;
                }
            }
            if (FIXNUM_P(beg)) {
                i = FIX2LONG(beg);
                do {
                    rb_yield(LONG2FIX(i));
                } while (POSFIXABLE(++i));
                beg = LONG2NUM(i);
            }
            ASSUME(!FIXNUM_P(beg));
            ASSUME(!SPECIAL_CONST_P(end));
        }
        if (!FIXNUM_P(beg) && RBIGNUM_SIGN(beg) == RBIGNUM_SIGN(end)) {
            if (EXCL(range)) {
                while (rb_big_cmp(beg, end) == INT2FIX(-1)) {
                    rb_yield(beg);
                    beg = rb_big_plus(beg, INT2FIX(1));
                }
            }
            else {
                VALUE c;
                while ((c = rb_big_cmp(beg, end)) != INT2FIX(1)) {
                    rb_yield(beg);
                    if (c == INT2FIX(0)) break;
                    beg = rb_big_plus(beg, INT2FIX(1));
                }
            }
        }
    }
    else if (SYMBOL_P(beg) && (NIL_P(end) || SYMBOL_P(end))) { /* symbols are special */
        beg = rb_sym2str(beg);
        if (NIL_P(end)) {
            rb_str_upto_endless_each(beg, sym_each_i, 0);
        }
        else {
            rb_str_upto_each(beg, rb_sym2str(end), EXCL(range), sym_each_i, 0);
        }
    }
    else {
        VALUE tmp = rb_check_string_type(beg);

        if (!NIL_P(tmp)) {
            if (!NIL_P(end)) {
                rb_str_upto_each(tmp, end, EXCL(range), each_i, 0);
            }
            else {
                rb_str_upto_endless_each(tmp, each_i, 0);
            }
        }
        else {
            if (!discrete_object_p(beg)) {
                rb_raise(rb_eTypeError, "can't iterate from %s",
                         rb_obj_classname(beg));
            }
            if (!NIL_P(end))
                range_each_func(range, each_i, 0);
            else {
                for (;; beg = rb_funcallv(beg, id_succ, 0, 0))
                    rb_yield(beg);
            }
        }
    }
    return range;
}
```

```c
static VALUE
range_each(VALUE range)
{
    VALUE beg, end;
    long i;

    RETURN_SIZED_ENUMERATOR(range, 0, 0, range_enum_size);

    beg = RANGE_BEG(range);
    end = RANGE_END(range);

    if (FIXNUM_P(beg) && NIL_P(end)) {
        range_each_fixnum_endless(beg);
    }
    else if (FIXNUM_P(beg) && FIXNUM_P(end)) { /* fixnums are special */
        return range_each_fixnum_loop(beg, end, range);
    }
    else if (RB_INTEGER_TYPE_P(beg) && (NIL_P(end) || RB_INTEGER_TYPE_P(end))) {
        if (SPECIAL_CONST_P(end) || RBIGNUM_POSITIVE_P(end)) { /* end >= FIXNUM_MIN */
            if (!FIXNUM_P(beg)) {
                if (RBIGNUM_NEGATIVE_P(beg)) {
                    do {
                        rb_yield(beg);
                    } while (!FIXNUM_P(beg = rb_big_plus(beg, INT2FIX(1))));
                    if (NIL_P(end)) range_each_fixnum_endless(beg);
                    if (FIXNUM_P(end)) return range_each_fixnum_loop(beg, end, range);
                }
                else {
                    if (NIL_P(end)) range_each_bignum_endless(beg);
                    if (FIXNUM_P(end)) return range;
                }
            }
            if (FIXNUM_P(beg)) {
                i = FIX2LONG(beg);
                do {
                    rb_yield(LONG2FIX(i));
                } while (POSFIXABLE(++i));
                beg = LONG2NUM(i);
            }
            ASSUME(!FIXNUM_P(beg));
            ASSUME(!SPECIAL_CONST_P(end));
        }
        if (!FIXNUM_P(beg) && RBIGNUM_SIGN(beg) == RBIGNUM_SIGN(end)) {
            if (EXCL(range)) {
                while (rb_big_cmp(beg, end) == INT2FIX(-1)) {
                    rb_yield(beg);
                    beg = rb_big_plus(beg, INT2FIX(1));
                }
            }
            else {
                VALUE c;
                while ((c = rb_big_cmp(beg, end)) != INT2FIX(1)) {
                    rb_yield(beg);
                    if (c == INT2FIX(0)) break;
                    beg = rb_big_plus(beg, INT2FIX(1));
                }
            }
        }
    }
    else if (SYMBOL_P(beg) && (NIL_P(end) || SYMBOL_P(end))) { /* symbols are special */
        beg = rb_sym2str(beg);
        if (NIL_P(end)) {
            rb_str_upto_endless_each(beg, sym_each_i, 0);
        }
        else {
            rb_str_upto_each(beg, rb_sym2str(end), EXCL(range), sym_each_i, 0);
        }
    }
    else {
        VALUE tmp = rb_check_string_type(beg);

        if (!NIL_P(tmp)) {
            if (!NIL_P(end)) {
                rb_str_upto_each(tmp, end, EXCL(range), each_i, 0);
            }
            else {
                rb_str_upto_endless_each(tmp, each_i, 0);
            }
        }
        else {
            if (!discrete_object_p(beg)) {
                rb_raise(rb_eTypeError, "can't iterate from %s",
                         rb_obj_classname(beg));
            }
            if (!NIL_P(end))
                range_each_func(range, each_i, 0);
            else
                for (;; beg = rb_funcallv(beg, id_succ, 0, 0))
                    rb_yield(beg);
        }
    }
    return range;
}
```



1. Constants

Ruby source

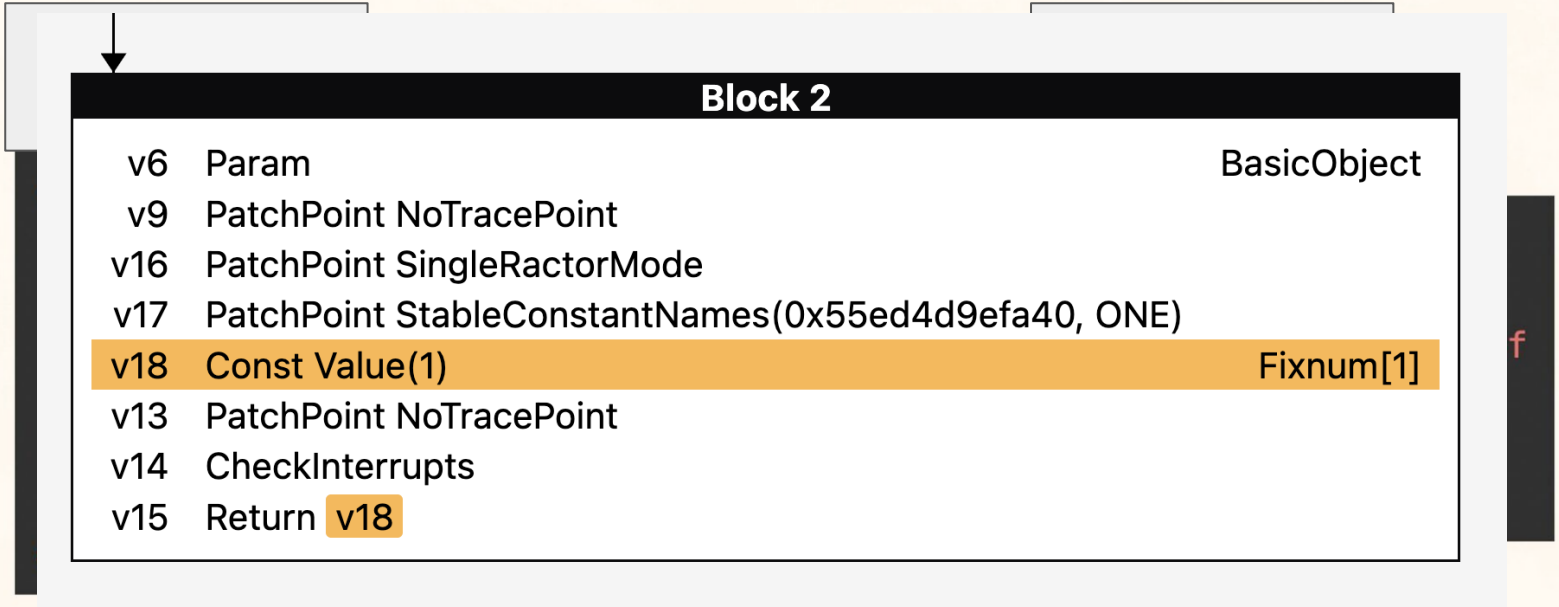
```ruby
class Namespace
  ONE = 1

  def one
    ONE
  end
end
```

"JIT code" expressed with Ruby syntax

```ruby
class Namespace
  def one_jit
    raise NotOptimized if self.cref != :cref
    1
  end
end
```

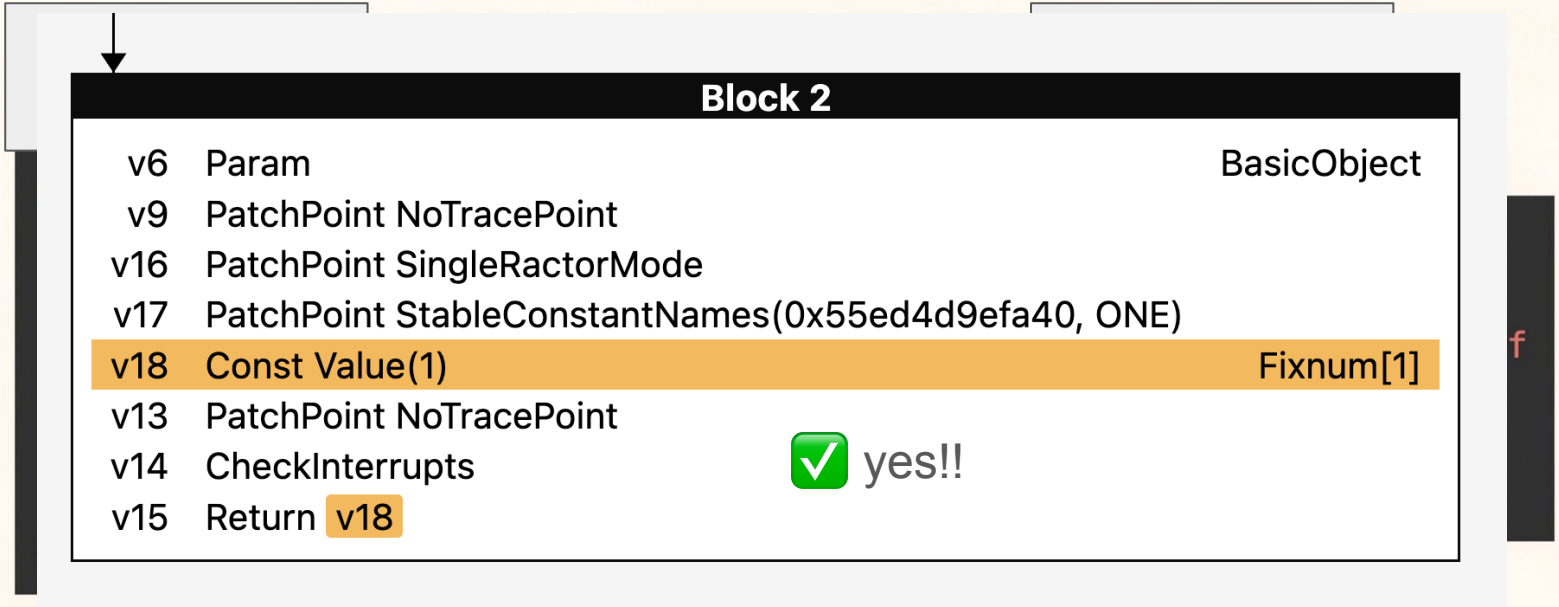
From Kokubun

1. Constants



From Kokubun

1. Constants



From Kokubun

1. Constants

```ruby
class Namespace
  ONE = 1
  TWO = 2

  def three
    ONE + TWO
  end
end
```

```ruby
class Namespace
  ONE = 1
  TWO = 2

  def three_jit
    raise NotOptimized if self.cref != :cref
    1 + 2
  end
end
```

From Kokubun

1. Constants

```
class
  ONE
  TWO

  def
    ON
  end
end
```

Block 2

v6	Param	BasicObject
v9	PatchPoint NoTracePoint	
v22	PatchPoint SingleRactorMode	
v23	PatchPoint StableConstantNames(0x559ecc7c7ab0, ONE)	
v24	Const Value(1)	Fixnum[1]
v25	PatchPoint SingleRactorMode	
v26	PatchPoint StableConstantNames(0x559ecc7e5140, TWO)	
v27	Const Value(2)	Fixnum[2]
v16	PatchPoint NoTracePoint	
v28	PatchPoint NoTracePoint	
v29	PatchPoint MethodRedefined(Integer@0x7f7c372aae00, +@0x2b, cme:0x7f7c372b1670)	
v32	Const Value(3)	Fixnum[3]
v31	IncrCounter inline_cfunc_optimized_send_count	
v19	PatchPoint NoTracePoint	
v20	CheckInterrupts	
v21	Return v32	

```
ref
```

From Kokubun

1. Constants

```
class
  ONE
  TWO

  def
    ON
  end
end
```

Block 2

v6	Param	BasicObject
v9	PatchPoint NoTracePoint	
v22	PatchPoint SingleRactorMode	
v23	PatchPoint StableConstantNames(0x559ecc7c7ab0, ONE)	
v24	Const Value(1)	Fixnum[1]
v25	PatchPoint SingleRactorMode	
v26	PatchPoint StableConstantNames(0x559ecc7e5140, TWO)	
v27	Const Value(2)	Fixnum[2]
v16	PatchPoint NoTracePoint	
v28	PatchPoint NoTracePoint	
v29	PatchPoint MethodRedefined(Integer@0x7f7c372aae00, +@0x2b, cme:0x7f7c372b1670)	
v32	Const Value(3)	Fixnum[3]
v31	IncrCounter inline_cfunc_optimized_send_count	
v19	PatchPoint NoTracePoint	
v20	CheckInterrupts	
v21	Return v32	

✅ yes!!

```
ref
```

From Kokubun

2. Variables

```ruby
def instance_var
  @ivar = 1
  @ivar
end
```



```ruby
@shape = { frozen: false, ivar: 0 }

def instance_var_jit
  if @shape == { frozen: false, ivar: 0 }
    raise NotOptimized
  end
  @heap[0] = 1
  @heap[0]
end
```

EN Keynote

The Future Shape of Ruby Objects

Chris Seaton @chrisseaton

2021

Implementing Object Shapes in CRuby

Jemma Issroff
@jemmaissroff

EN

2022 (tomorrow)

From Kokubun

```
def in
  @iva
  @iva
end
```

Block 3

v6	Param	BasicObject
v9	PatchPoint NoTracePoint	
v10	Const Value(1)	Fixnum[1]
v12	PatchPoint SingleRactorMode	
v24	GuardType v6 , HeapBasicObject	HeapBasicObject
v25	LoadField v24 , :_shape_id@0x4	CShape
v26	GuardBitEquals v25 , CShape(0x80095)	CShape[0x80095]
v27	StoreField v24 , :@ivar@0x10, v10	
v28	WriteBarrier v24 , v10	
v16	PatchPoint NoTracePoint	
v17	PatchPoint SingleRactorMode	
v31	GuardBitEquals v25 , CShape(0x80095)	CShape[0x80095]
v21	CheckInterrupts	
v22	Return v10	

From Kokubun

2021 2022 (tomorrow)

```
def in
  @iva
  @iva
end
```

Block 3

v6	Param	BasicObject
v9	PatchPoint NoTracePoint	
v10	Const Value(1)	Fixnum[1]
v12	PatchPoint SingleRactorMode	
v24	GuardType v6 , HeapBasicObject	HeapBasicObject
v25	LoadField v24 , :_shape_id@0x4	CShape
v26	GuardBitEquals v25 , CShape(0x80095)	CShape[0x80095]
v27	StoreField v24 , :@ivar@0x10, v10	
v28	WriteBarrier v24 , v10	
v16	PatchPoint NoTracePoint	
v17	PatchPoint SingleRactorMode	
v31	GuardBitEquals v25 , CShape(0x80095)	CShape[0x80095]
v21	CheckInterrupts	
v22	Return v10	

✅ yes!!

From Kokubun

2021 2022 (tomorrow)

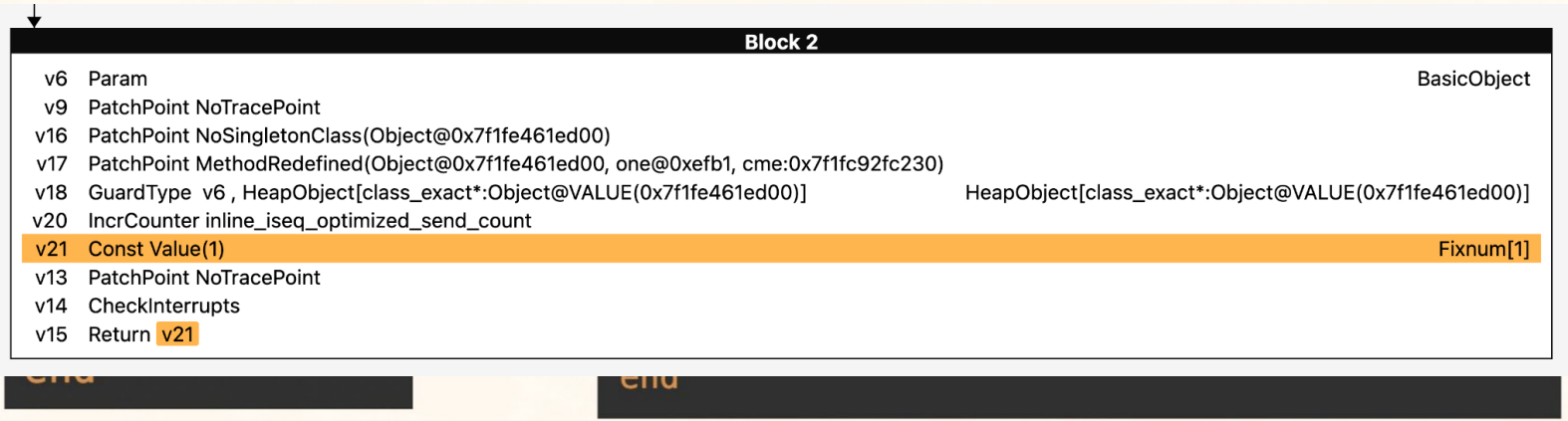
3. Method calls

```ruby
def one = 1

def method_call
  one
end
```

```ruby
CACHE = [method(:one), Namespace]

def method_call_jit
  raise NotOptimized if CACHE[1] != self.class
  1
end
```

From Kokubun

3. Method calls



From Kokubun

3. Method calls

```
Block 2
  v6  Param                                                                                  BasicObject
  v9  PatchPoint NoTracePoint
 v16  PatchPoint NoSingletonClass(Object@0x7f1fe461ed00)
 v17  PatchPoint MethodRedefined(Object@0x7f1fe461ed00, one@0xefb1, cme:0x7f1fc92fc230)
 v18  GuardType  v6 , HeapObject[class_exact*:Object@VALUE(0x7f1fe461ed00)]                 HeapObject[class_exact*:Object@VALUE(0x7f1fe461ed00)]
 v20  IncrCounter inline_iseq_optimized_send_count
 v21  Const Value(1)                                                                        Fixnum[1]
 v13  PatchPoint NoTracePoint
 v14  CheckInterrupts
 v15  Return v21
```

✅ yes!!

From Kokubun

3. Method calls

- Code locality
- Method inlining: C ⇔ Ruby
- Pass arguments with native ABI
- Deoptimization on redefinition or interruption (or TracePoint)

From Kokubun

3. Method calls

- Code locality ?
- Method inlining: C ⇔ Ruby
- Pass arguments with native ABI
- Deoptimization on redefinition or interruption (or TracePoint)

From Kokubun

3. Method calls

- Code locality ❓
- Method inlining: C ⇔ Ruby ✅ (ish)
- Pass arguments with native ABI
- Deoptimization on redefinition or interruption (or TracePoint)

From Kokubun

3. Method calls

- Code locality ❓
- Method inlining: C ⇔ Ruby ✅
  (ish)
- Pass arguments with native ABI ✅
- Deoptimization on redefinition or interruption (or TracePoint)

From Kokubun

3. Method calls

- Code locality ❓
- Method inlining: C ⇔ Ruby ✅ (ish)
- Pass arguments with native ABI ✅
- Deoptimization on redefinition or interruption (or TracePoint) ✅

From Kokubun

4. Garbage collection

```ruby
Point = Struct.new(:x, :y)

def object_alloc
  point = Point.new(1, 2)
  point.x + point.y
end
```

```ruby
Point = Struct.new(:x, :y)

def object_alloc_vm
  GC.start
  point = malloc(Point)
  point.x = 1
  point.y = 2
  point.x + point.y
end
```

From Kokubun

4. Garbage collection

```ruby
Point = Struct.new(:x, :y)

def object_alloc
  point = Point.new(1, 2)
  point.x + point.y
end
```

```ruby
Point = Struct.new(:x, :y)

def object_alloc_jit
  point = STACK.address
  STACK.address += 64
  point.x = 1
  point.y = 2
  point.x + point.y
  STACK.address -= 64
end
```

From Kokubun

4. Garbage collection

```ruby
Point = Struct.new(:x, :y)

def object_alloc
  point = Point.new(1, 2)
  point.x + point.y
end
```

```ruby
def object_alloc_jit
  point_x = 1
  point_y = 2
  point_x + point_y
end
```

From Kokubun

4.

```
Point = Stru

def object_a
  point = Po
  point.x +
end
```

Block 2

v8	Param	BasicObject
v9	Param	NilClass
v12	PatchPoint NoTracePoint	
v67	PatchPoint SingleRactorMode	
v68	PatchPoint StableConstantNames(0x55675677be40, Point)	
v69	Const Value(VALUE(0x7f57150367a0))	Class[Point@0x7f57150367a0]
v16	Const Value(nil)	NilClass
v19	Const Value(1)	Fixnum[1]
v21	Const Value(2)	Fixnum[2]
v70	PatchPoint MethodRedefined(Point@0x7f57150367a0, new@0x1221, cme:0x7f571501c160)	
v25	ObjectAlloc v69	HeapBasicObject
v80	PatchPoint NoSingletonClass(Point@0x7f57150367a0)	
v81	PatchPoint NoTracePoint	
v82	PatchPoint MethodRedefined(Point@0x7f57150367a0, initialize@0xc71, cme:0x7f5730348170)	
v83	GuardType v25 , HeapObject[class_exact:Point]	HeapObject[class_exact:Point]
v84	CCallVariadic v83 , :Struct#initialize@0x7ffce3adc850, v19 , v21	BasicObject
v29	CheckInterrupts	
v50	PatchPoint NoEPEscape(object_alloc)	
v52	PatchPoint NoTracePoint	
v72	PatchPoint NoSingletonClass(Point@0x7f57150367a0)	
v73	PatchPoint MethodRedefined(Point@0x7f57150367a0, x@0xb5b1, cme:0x7f571501c040)	
v74	GuardType v25 , HeapObject[class_exact:Point]	HeapObject[class_exact:Point]
v75	LoadField v74 , :x@0x10	BasicObject
v57	PatchPoint NoEPEscape(object_alloc)	
v76	PatchPoint NoSingletonClass(Point@0x7f57150367a0)	
v77	PatchPoint MethodRedefined(Point@0x7f57150367a0, y@0xefd1, cme:0x7f571501bf50)	
v78	GuardType v25 , HeapObject[class_exact:Point]	HeapObject[class_exact:Point]
v79	LoadField v78 , :y@0x18	BasicObject
v61	PatchPoint NoTracePoint	
v85	PatchPoint NoTracePoint	
v86	PatchPoint MethodRedefined(Integer@0x7f573035ae00, +@0x2b, cme:0x7f5730361670)	
v87	GuardType v75 , Fixnum	Fixnum
v88	GuardType v79 , Fixnum	Fixnum
v89	FixnumAdd v87 , v88	Fixnum
v90	IncrCounter inline_cfunc_optimized_send_count	
v64	PatchPoint NoTracePoint	
v65	CheckInterrupts	
v66	Return v89	

4.

```
Point = Stru

def object_a
  point = Po
  point.x +
end
```

Block 2

v8	Param	BasicObject
v9	Param	NilClass
v12	PatchPoint NoTracePoint	
v67	PatchPoint SingleRactorMode	
v68	PatchPoint StableConstantNames(0x55675677be40, Point)	
v69	Const Value(VALUE(0x7f57150367a0))	Class[Point@0x7f57150367a0]
v16	Const Value(nil)	NilClass
v19	Const Value(1)	Fixnum[1]
v21	Const Value(2)	Fixnum[2]
v70	PatchPoint MethodRedefined(Point@0x7f57150367a0, new@0x1221, cme:0x7f571501c160)	
v25	ObjectAlloc v69	HeapBasicObject
v80	PatchPoint NoSingletonClass(Point@0x7f57150367a0)	
v81	PatchPoint NoTracePoint	
v82	PatchPoint MethodRedefined(Point@0x7f57150367a0, initialize@0xc71, cme:0x7f5730348170)	
v83	GuardType v25 , HeapObject[class_exact:Point]	HeapObject[class_exact:Point]
v84	CCallVariadic v83 , :Struct#initialize@0x7ffce3adc850, v19 , v21	BasicObject
v29	CheckInterrupts	
v50	PatchPoint NoEPEscape(object_alloc)	
v52	PatchPoint NoTracePoint	
v72	PatchPoint NoSingletonClass(Point@0x7f57150367a0)	
v73	PatchPoint MethodRedefined(Point@0x7f57150367a0, x@0xb5b1, cme:0x7f571501c040)	
v74	GuardType v25 , HeapObject[class_exact:Point]	HeapObject[class_exact:Point]
v75	LoadField v74 , :x@0x10	BasicObject
v57	PatchPoint NoEPEscape(object_alloc)	
v76	PatchPoint NoSingletonClass(Point@0x7f57150367a0)	
v77	PatchPoint MethodRedefined(Point@0x7f57150367a0, y@0xefd1, cme:0x7f571501bf50)	
v78	GuardType v25 , HeapObject[class_exact:Point]	HeapObject[class_exact:Point]
v79	LoadField v78 , :y@0x18	BasicObject
v61	PatchPoint NoTracePoint	
v85	PatchPoint NoTracePoint	
v86	PatchPoint MethodRedefined(Integer@0x7f573035ae00, +@0x2b, cme:0x7f5730361670)	
v87	GuardType v75 , Fixnum	Fixnum
v88	GuardType v79 , Fixnum	Fixnum
v89	FixnumAdd v87 , v88	Fixnum
v90	IncrCounter inline_cfunc_optimized_send_count	
v64	PatchPoint NoTracePoint	
v65	CheckInterrupts	
v66	Return v89	

✗ not yet

So: mixed bag…

…how do we succeed?

We need inlining, effects, escape analysis, loop unrolling

Inlining motivation: sample Ruby code

```
def test(items)
  sum = 0
  items.each do |x|
    sum += x
  end
  sum
end

test([1,2,3])
```

Method call for `each`

Method call per element

The promise of inlining

```
def test(items)
  sum = 0
  items.each do |x|
    sum += x
  end
  sum
end

class Array
  def each
    _i = 0
    while _i < items.size
      yield self[_i]
      _i += 1
    end
    self
  end
end

30.times do
  test [1,2,3]
end
```

```
def test(items)
  sum = 0
  _i = 0
  while _i < items.size
    yield items[_i] to (|x|
      sum += x
    end)
    _i += 1
  end
  sum
end

30.times do
  test [1,2,3]
end
```

```
def test(items)
  sum = 0
  _i = 0
  while _i < items.size
    sum += items[_i]
    _i += 1
  end
  sum
end

30.times do
  test [1,2,3]
end
```

```
bb2(v16:BasicObject, v17:BasicObject, v18:NilClass, v19:NilClass,
v20:NilClass, v21:NilClass):
  v25:Fixnum[0] = Const Value(0)
  v29:Fixnum[0] = Const Value(0)
  v33:NilClass = Const Value(nil)
  v108:ArrayExact = GuardType v17, ArrayExact
  v110:Fixnum = ArrayLength v108
  CheckInterrupts
  Jump bb4(v16, v17, v25, v29, v33, v110)
bb4(v45:BasicObject, v46:BasicObject, v47:Fixnum, v48:Fixnum,
v49:NilClass, v50:Fixnum):
  v115:BoolExact = FixnumLt v48, v50
  CheckInterrupts
  v61:CBool = Test v115
  IfTrue v61, bb3(v45, v46, v47, v48, v49, v50)
  CheckInterrupts
  Return v47
bb3(v74:BasicObject, v75:BasicObject, v76:Fixnum, v77:Fixnum,
v78:NilClass, v79:Fixnum):
  v120:ArrayExact = GuardType v75, ArrayExact
  v122:BasicObject = ArrayArefFixnum v120, v77
  v127:Fixnum = GuardType v122, Fixnum
  v128:Fixnum = FixnumAdd v76, v127
  v99:Fixnum[1] = Const Value(1)
  v133:Fixnum = FixnumAdd v77, v99
  Jump bb4(v74, v75, v128, v133, v78, v79)
```

Compiler optimizations have a field day with this version

Inline Array#each

Inline the block

Ruby peeled away!

Inlining enables monomorphization

```
class HashWithIndifferentAccess
  def [](key)
    @hash[key.to_s]
  end
end
```

key is either String or Symbol
:(

```
@params = HWIA.new

def function_a
  @params[:my_param]
end

def function_b
  @params['other_param']
end
```

Inlining enables monomorphization

```
class HashWithIndifferentAccess
  def [](key)
    @hash[key.to_s]
  end
end
```

key is either String or Symbol
:(

```
@params = HWIA.new

def function_a
  @params[:my_param]
end

def function_b
  @params['other_param']
end
```

Inline!

More context: always Symbol

```
def function_a
  @params.hash[:my_param.to_s]
end
```

```
def function_b
  @params.hash['other_param']
end
```

More context: always String

Optimizations work best *together*

So we’re working on that. Also,

ZJIT later this year

- Lightweight frames
  - Only fill in VM frames when the VM needs to read them, like when raising exceptions
  - Also helps with inlining
- Smarter policy (a configurable state machine!)
  - Profile
  - Compile and re-compile
  - Inline
- Better code layout
- Cleaned-up code (value numbering, …)
- Store and load forwarding

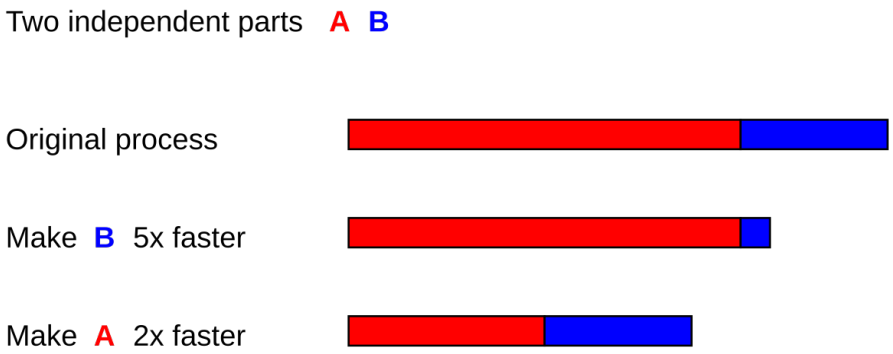
…but does that get us to 5x? 10x?

Not if we’re calling C code

Amdahl's law

- What if Ruby is only 20% of CPU time?

- …and the rest is spent in C builtins or extensions?



From Wikipedia

We need cooperation from the runtime

…and the next five years

- More builtin C→Ruby
  - Loop unrolling
    - e.g. [a, b, c].include? d
    - e.g. [a, b, c].hash
- Better whole-runtime tracing and profiling
- Fast inline thread-local bump pointer allocation
- Background thread compilation
- Threading-aware storage strategies (cc Benoit)

- More interesting type lattice (integer ranges, known bits, etc)
- Escape analysis, scalar replacement, and allocation removal
- Tuning the inliner (goes on forever)
- Verification
  - In optimizations
  - In register allocation
- Big analysis of continuous production profiles
- Compilation tiers?
- SBBV?

Now for some wilder stuff

Rewrite in Ruby: Maybe a dialect?

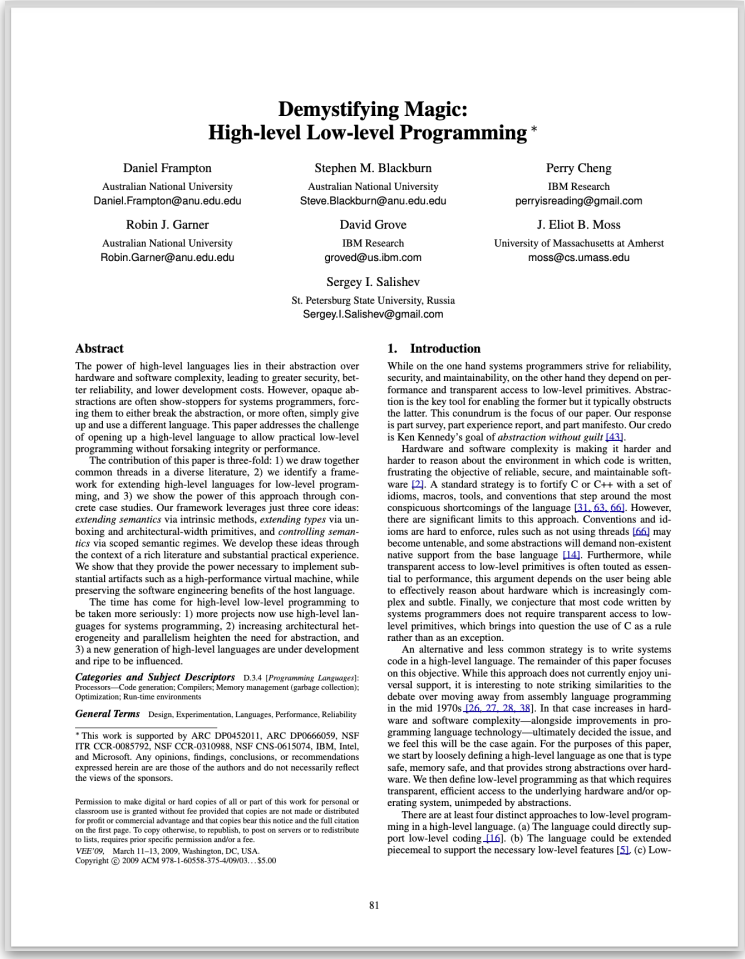
Need a single source of truth!

“PreRuby” compiles to C and Ruby???

Native, JITable

Like PreScheme/RPython/Slang/MLton

I’m spitballing here, idk



Demystifying Magic:
High-level Low-level Programming *

Daniel Frampton
Australian National University
Daniel.Frampton@anu.edu.edu

Stephen M. Blackburn
Australian National University
Steve.Blackburn@anu.edu.edu

Perry Cheng
IBM Research
perryisreading@gmail.com

Robin J. Garner
Australian National University
Robin.Garner@anu.edu.edu

David Grove
IBM Research
groved@us.ibm.com

J. Eliot B. Moss
University of Massachusetts at Amherst
moss@cs.umass.edu

Sergey I. Salishev
St. Petersburg State University, Russia
Sergey.I.Salishev@gmail.com

Abstract

The power of high-level languages lies in their abstraction over hardware and software complexity, leading to greater security, better reliability, and lower development costs. However, opaque abstractions are often show-stoppers for systems programmers, forcing them to either break the abstraction, or more often, simply give up and use a different language. This paper addresses the challenge of opening up a high-level language to allow practical low-level programming without forsaking integrity or performance.

The contribution of this paper is three-fold: 1) we draw together common threads in a diverse literature, 2) we identify a framework for extending high-level languages for low-level programming, and 3) we show the power of this approach through concrete case studies. Our framework leverages just three core ideas: *extending semantics* via intrinsic methods, *extending types* via unboxing and architectural-width primitives, and *controlling semantics* via scoped semantic regimes. We develop these ideas through the context of a rich literature and substantial practical experience. We show that they provide the power necessary to implement substantial artifacts such as a high-performance virtual machine, while preserving the software engineering benefits of the host language.

The time has come for high-level low-level programming to be taken more seriously: 1) more projects now use high-level languages for systems programming, 2) increasing architectural heterogeneity and parallelism heighten the need for abstraction, and 3) a new generation of high-level languages are under development and ripe to be influenced.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—Code generation; Compilers; Memory management (garbage collection); Optimization; Run-time environments

General Terms Design, Experimentation, Languages, Performance, Reliability

* This work is supported by ARC DP0452011, ARC DP0666059, NSF ITR CCR-0085792, NSF CCR-0310988, NSF CNS-0615074, IBM, Intel, and Microsoft. Any opinions, findings, conclusions, or recommendations expressed herein are are those of the authors and do not necessarily reflect the views of the sponsors.

VEE'09, March 11–13, 2009, Washington, DC, USA.

1. Introduction

While on the one hand systems programmers strive for reliability, security, and maintainability, on the other hand they depend on performance and transparent access to low-level primitives. Abstraction is the key tool for enabling the former but it typically obstructs the latter. This conundrum is the focus of our paper. Our response is part survey, part experience report, and part manifesto. Our credo is Ken Kennedy’s goal of *abstraction without guilt* [43].

Hardware and software complexity is making it harder and harder to reason about the environment in which code is written, frustrating the objective of reliable, secure, and maintainable software [2]. A standard strategy is to fortify C or C++ with a set of idioms, macros, tools, and conventions that step around the most conspicuous shortcomings of the language [31, 63, 66]. However, there are significant limits to this approach. Conventions and idioms are hard to enforce, rules such as not using threads [66] may become untenable, and some abstractions will demand non-existent native support from the base language [14]. Furthermore, while transparent access to low-level primitives is often touted as essential to performance, this argument depends on the user being able to effectively reason about hardware which is increasingly complex and subtle. Finally, we conjecture that most code written by systems programmers does not require transparent access to low-level primitives, which brings into question the use of C as a rule rather than as an exception.

An alternative and less common strategy is to write systems code in a high-level language. The remainder of this paper focuses on this objective. While this approach does not currently enjoy universal support, it is interesting to note striking similarities to the debate over moving away from assembly language programming in the mid 1970s [26, 27, 28, 38]. In that case increases in hardware and software complexity—alongside improvements in programming language technology—ultimately decided the issue, and we feel this will be the case again. For the purposes of this paper, we start by loosely defining a high-level language as one that is type safe, memory safe, and that provides strong abstractions over hardware. We then define low-level programming as that which requires transparent, efficient access to the underlying hardware and/or operating system, unimpeded by abstractions.

There are at least four distinct approaches to low-level programming in a high-level language. (a) The language could directly support low-level coding [16]. (b) The language could be extended piecemeal to support the necessary low-level features [5], (c) Low-

Ahead-of-time???

- A very difficult problem: how do we quiesce and rehydrate runtime state?
- Another difficult problem: how do we optimize a dynamic language without running it?
- Either massive slow analysis & restricting dynamic behavior after compile
  OR
  Skip a bunch of optimizations
  OR
  Try to record and link “profiles”, like PGO
- Maybe skip this entirely, do something at application layer like CRIU

That’s all, folks

Thank you for committing

Thank you, developers

Thank you, developers

developers

Thank you, developers

developers developers

Thank you, developers

developers developers developers

Thank you, developers

developers developers developers


Thank you, developers

developers developers developers



Thank you, developers

developers developers developers



Thank you, developers

developers developers developers



Thank you, developers

developers developers developers




Thank you, developers

developers developers developers



The 2025 team…

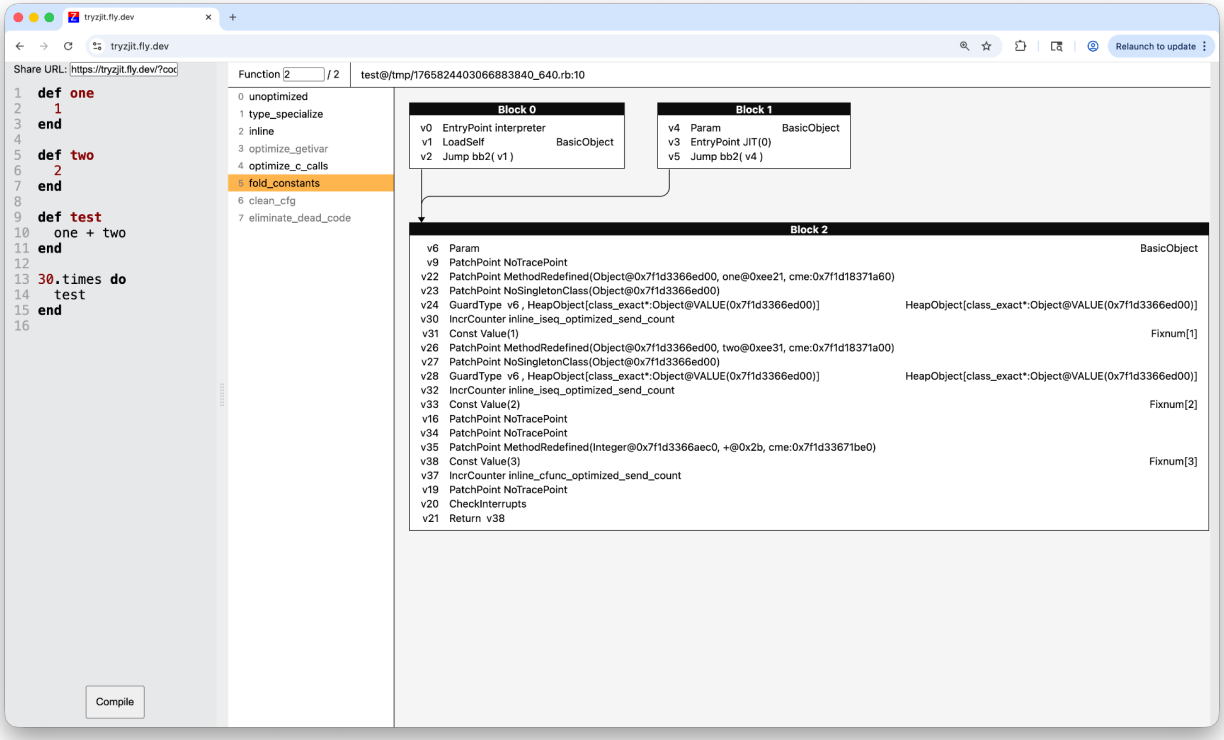
Aaron Patterson
Aiden Fox Ivey
Alan Wu
Jacob Denbeaux
Kevin Menard
Max Bernstein
Maxime Chevalier-Boisvert
Randy Stauner
Stan Lo
Takashi Kokubun

…and Rubyists like you

Abrar Habib
Alex Rocha
André Luiz Tiago Soares
Benoit Daloze
cui
Daniel Colson
Donghee Na
Eileen Alayce
Étienne Barrié
Godfrey Chan
Goshanraj Govindaraj
Hiroshi SHIBATA
Hoa Nguyen
Jean Boussier
Jeff Zhang
Jeremy Evans
John Hawthorn
Ken Jin
Luke Gruber
Matt Valentine-House
Max Leopold
Nery Campusano
Nobuyoshi Nakada
Nozomi Hijikata
Peter Zhu
ReLU
Satoshi Tagomori
Shannon Skipper
Steven Webb
Tavian Barnes
Tobias Lütke
Tomás Coêlho
Charlotte Wen

What you should do now

Mess around with ZJIT



tryzjit.fly.dev

Maybe ruby --zjit on a small (non-production) program…

Join the ZJIT Zulip

- We’re very friendly
- And like to talk about compilers

zjit.zulipchat.com

Come talk to me in person!

Thank you.　ご清聴ありがとうございました。

zjit.dev