A tour of CPython's runtime

Brandt Bucher (March 11th, 2024)

A tour of CPython's runtime

...and how you can speed up every Python process on the planet!

Brandt Bucher (March 11th, 2024)

Brandt Bucher

Brandt Bucher

- 2017: Started using Python.
- 2018: Contributed code to CPython.
- 2019: Joined Python's Triage Team.
- 2020: Joined Python's Core Development Team.
- 2021: Joined Microsoft's CPython Performance Engineering Team.
- 2022: Made CPython 3.11 25% faster!
- 2023: Implemented CPython's new JIT compiler.

Microsoft's CPython Performance Engineering Team

The "Faster CPython" Project

The "Faster CPython" Project

- Python's BDFL:
  - Guido van Rossum
- Four other Python core developers:
  - Mark Shannon
  - Irit Katriel
  - Eric Snow
  - Brandt Bucher
- One member of Python's triage team:
  - Michael Droettboom

The "Faster CPython" Project

- Python's BDFL:
  - Guido van Rossum
- Four other Python core developers:
  - Mark Shannon
  - Irit Katriel
  - Eric Snow
  - Brandt Bucher
- One member of Python's triage team:
  - Michael Droettboom

The "Faster CPython" Project

- Python's BDFL:
  - @gvanrossum
- Four other Python core developers:
  - @markshannon
  - @iritkatriel
  - @ericsnowcurrently
  - @brandtbucher
- One member of Python's triage team:
  - @mdboom

The "Faster CPython" Project

- faster-cpython
- faster-cpython/ideas
- faster-cpython/benchmarking-public

The "Faster CPython" Project

- California (Microsoft)
- Utah (Microsoft)
- Washington (Meta)
- South Dakota (Meta)
- Washington, D.C. (Microsoft)
- United Kingdom (Microsoft)
- Singapore (National University of Singapore)

The "Faster CPython" Project

- California (Microsoft)
- Utah (Microsoft)
- Washington (Meta)
- South Dakota (Meta)
- Washington, D.C. (Microsoft)
- United Kingdom (Microsoft)
- Singapore (National University of Singapore)

The "Faster CPython" Project

- California (Microsoft, UC Irvine?)
- Utah (Microsoft)
- Washington (Meta)
- South Dakota (Meta)
- Washington, D.C. (Microsoft)
- United Kingdom (Microsoft)
- Singapore (National University of Singapore)

Python

Python

- 33 years old!
- *Very* high-level
- *Very* widely used
- Dynamic
- Object-oriented
- Interpreted
- Automatic memory management
- Deep introspection and metaprogramming

Python

- 33 years old!
- *Very* high-level
- *Very* widely used
- Dynamic
- Object-oriented
- Interpreted
- Automatic memory management
- Deep introspection and metaprogramming

Python

- 33 years old!
- *Very* high-level
- *Very* widely used
- Dynamic
- Object-oriented
- Interpreted
- Automatic memory management
- Deep introspection and metaprogramming

Python

- Most objects have arbitrary mappings of attributes: `instance.__dict__`.
- Bytecode is a runtime object: `function.__code__`.
- *Frames* are runtime objects: `sys._getframe()`.
- Attribute/global name accesses and assignments can run arbitrary code.
- Even simple operators go through incredibly complex double-dispatching.

Python

- Most objects have arbitrary mappings of attributes: `instance.__dict__`.
- Bytecode is a runtime object: `function.__code__`.
- *Frames* are runtime objects: `sys._getframe()`.
- Attribute/global name accesses and assignments can run arbitrary code.
- Even simple operators go through incredibly complex double-dispatching.

Python

- Most objects have arbitrary mappings of attributes: `instance.__dict__`.
- Bytecode is a runtime object: `function.__code__`.
- *Frames* are runtime objects: `sys._getframe()`.
- Attribute/global name accesses and assignments can run arbitrary code.
- Even simple operators go through incredibly complex double-dispatching.

CPython

CPython

- Reference implementation of Python
- Used by ~100% of Python programmers
- Reference-counted (augmented with cyclic stop-the-world GC)
- Has an incredibly rich ecosystem of third-party C extensions
- Maintained by a few dozen active "core developers"
- Free and open-source
- python/cpython

Optimization

Optimization

- Build IR
- Check types
- Optimize
- Compile

Optimization

- Build IR
- Check types
- Optimize
- Compile

Optimization

- Build IR
- Profile
- Optimize
- Compile

Optimization

- Constant folding
- Dead code elimination
- Jump threading
- Liveness analysis
- Peephole optimizations
- Hot/cold splitting
- Common subexpression elimination
- Copy propagation
- Type propagation
- Constant propagation
- Constant promotion
- Guard elimination
- Loop peeling
- Loop-invariant code motion
- Inlining

Optimization

- Constant folding
- Dead code elimination
- Jump threading
- Liveness analysis
- Peephole optimizations
- Hot/cold splitting
- Common subexpression elimination
- Copy propagation
- Type propagation
- Constant propagation
- Constant promotion
- Guard elimination
- Loop peeling
- Loop-invariant code motion
- Inlining

Optimization

- Constant folding
- Dead code elimination
- Jump threading
- Liveness analysis
- Peephole optimizations
- Hot/cold splitting
- Common subexpression elimination
- Copy propagation
- Type propagation
- Constant propagation
- Constant promotion
- Guard elimination
- Loop peeling
- Loop-invariant code motion
- Inlining

Runtime Optimizations

Background

Background

- CPython 3.11:
  - Specializing adaptive interpreter profiles programs and optimizes them on-the-fly
- CPython 3.12:
  - Interpreter is generated from a DSL, allowing analysis and modification at build time
- CPython 3.13:
  - Internal pipeline for detecting, optimizing, and executing hot code paths

Background

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a
```

Background

```
for _ in range(n):
    a, b = b, a + b
```

Background

CPython 3.10: Bytecode

```
for _ in range(n):    FOR_ITER
    a, b = b, a + b   STORE_FAST
                      LOAD_FAST_LOAD_FAST
                      LOAD_FAST
                      BINARY_OP
                      STORE_FAST_STORE_FAST
                      JUMP_BACKWARD
```

Background

CPython 3.10: Bytecode

```
for _ in range(n):    FOR_ITER
    a, b = b, a + b   STORE_FAST
                      LOAD_FAST_LOAD_FAST
                      LOAD_FAST
                      BINARY_OP
                      STORE_FAST_STORE_FAST
                      JUMP_BACKWARD                stack
```

Background

CPython 3.10: Bytecode

```
for _ in range(n):    FOR_ITER
    a, b = b, a + b   STORE_FAST
                      LOAD_FAST_LOAD_FAST
                      LOAD_FAST
                      BINARY_OP
                      STORE_FAST_STORE_FAST          stack
                      JUMP_BACKWARD                iterator
```

Background

CPython 3.10: Bytecode

```
for _ in range(n):    FOR_ITER
    a, b = b, a + b   STORE_FAST
                      LOAD_FAST_LOAD_FAST
                      LOAD_FAST
                      BINARY_OP
                      STORE_FAST_STORE_FAST
                      JUMP_BACKWARD
```

stack
next(iterator)
iterator

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST
                        JUMP_BACKWARD
```

stack

next(iterator)

iterator

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST
                        JUMP_BACKWARD
```

stack

next(iterator)

iterator

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP
                        STORE_FAST_STORE_FAST       stack
                        JUMP_BACKWARD             iterator
```

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP                     stack
                        STORE_FAST_STORE_FAST           b
                        JUMP_BACKWARD                iterator
```

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST                       stack
                        BINARY_OP                         a
                        STORE_FAST_STORE_FAST             b
                        JUMP_BACKWARD                 iterator
```

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):
    a, b = b, a + b
```

```
FOR_ITER_RANGE
STORE_FAST
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

```
_ = next(iterator)
```

stack
b
a
b
iterator

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST                    stack
                        BINARY_OP                    a + b
                        STORE_FAST_STORE_FAST          b
                        JUMP_BACKWARD              iterator
```

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST                     stack
                        BINARY_OP                     a + b
                        STORE_FAST_STORE_FAST           b
                        JUMP_BACKWARD                iterator
```

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST                     stack
                        BINARY_OP_ADD_INT             a + b
                        STORE_FAST_STORE_FAST           b
                        JUMP_BACKWARD               iterator
```

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE              _ = next(iterator)
    a, b = b, a + b     STORE_FAST                  b = a + b
                        LOAD_FAST_LOAD_FAST
                        LOAD_FAST
                        BINARY_OP_ADD_INT                 stack
                        STORE_FAST_STORE_FAST               b
                        JUMP_BACKWARD                   iterator
```

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):      FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b     STORE_FAST              b = a + b
                        LOAD_FAST_LOAD_FAST     a = b
                        LOAD_FAST
                        BINARY_OP_ADD_INT
                        STORE_FAST_STORE_FAST       stack
                        JUMP_BACKWARD             iterator
```

Background

CPython 3.11: Specialized Bytecode

```
for _ in range(n):    FOR_ITER_RANGE          _ = next(iterator)
    a, b = b, a + b   STORE_FAST              b = a + b
                      LOAD_FAST_LOAD_FAST     a = b
                      LOAD_FAST
                      BINARY_OP_ADD_INT
                      STORE_FAST_STORE_FAST         stack
                      JUMP_BACKWARD               iterator
```

Background

CPython 3.11: Specialized Bytecode

```
FOR_ITER_RANGE
STORE_FAST
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP_ADD_INT
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
FOR_ITER_RANGE
STORE_FAST
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP_ADD_INT
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
FOR_ITER_RANGE
STORE_FAST
```

```
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP_ADD_INT
```

```
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
FOR_ITER_RANGE
STORE_FAST
```

```
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP_ADD_INT
```

```
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE
STORE_FAST
```

```
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP_ADD_INT
```

```
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
LOAD_FAST_LOAD_FAST
LOAD_FAST
BINARY_OP_ADD_INT
```

```
STORE_FAST_STORE_FAST
JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

LOAD_FAST


BINARY_OP_ADD_INT
```

```
STORE_FAST_STORE_FAST


JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

BINARY_OP_ADD_INT
```

```
STORE_FAST_STORE_FAST


JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT
```

```
STORE_FAST_STORE_FAST


JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT
```

```
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_STORE_FAST

JUMP_BACKWARD
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT
```

```
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY
_SET_IP
_JUMP_TO_TOP
```

Background

CPython 3.13: Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT
```

```
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY
_SET_IP
_JUMP_TO_TOP
```

Background

CPython 3.13: Optimized Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT
```

```
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY
_SET_IP
_JUMP_TO_TOP
```

Background

CPython 3.13: Optimized Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_RANGE
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_GUARD_BOTH_INT
_BINARY_OP_ADD_INT
```

```
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY
_SET_IP
_JUMP_TO_TOP
```

Background

CPython 3.13: Optimized Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP

_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP

_BINARY_OP_ADD_INT
```

```
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY
_SET_IP
_JUMP_TO_TOP
```

Background

CPython 3.13: Optimized Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_SET_IP

_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE

_CHECK_VALIDITY
_SET_IP
_STORE_FAST
```

```
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP
_LOAD_FAST

_CHECK_VALIDITY
_SET_IP

_BINARY_OP_ADD_INT
```

```
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_STORE_FAST

_CHECK_VALIDITY
_SET_IP
_JUMP_TO_TOP
```

Background

CPython 3.13: Optimized Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY

                                _LOAD_FAST                  _STORE_FAST
_GUARD_NOT_EXHAUSTED_RANGE      _LOAD_FAST                  _STORE_FAST
_ITER_NEXT_RANGE

                                                            _SET_IP
                                _LOAD_FAST                  _JUMP_TO_TOP
_STORE_FAST

                                _SET_IP

                                _BINARY_OP_ADD_INT
```

Background

CPython 3.13: Optimized Micro-Op Traces (`-X uops`)

```
_CHECK_VALIDITY
_GUARD_NOT_EXHAUSTED_RANGE
_ITER_NEXT_RANGE
_STORE_FAST
_LOAD_FAST
_LOAD_FAST
_LOAD_FAST
_SET_IP
_BINARY_OP_ADD_INT
_STORE_FAST
_STORE_FAST
_SET_IP
_JUMP_TO_TOP
```

Just-In-Time Compilation

Just-In-Time Compilation

- Technical goals:
  - Remove interpretive overhead
  - Statically compile optimized traces
  - Reduce indirection:
    - "Burn in" constants, caches, and arguments
    - Move data off of frames and into registers
    - Bring hot code paths in-line

- Deployment goals:
  - Broad platform support
  - Few runtime dependencies
  - Low implementation complexity

Just-In-Time Compilation

- Technical goals:
  - Remove interpretive overhead
  - Statically compile optimized traces
  - Reduce indirection:
    - "Burn in" constants, caches, and arguments
    - Move data off of frames and into registers
    - Bring hot code paths in-line

- Deployment goals:
  - Broad platform support
  - Few runtime dependencies
  - Low implementation complexity

Just-In-Time Compilation

Copy-And-Patch Compilation

Copy-And-Patch Compilation

- Haoran Xu and Fredrik Kjolstad. 2021. Copy-and-Patch Compilation: A Fast Compilation Algorithm for High- Level Languages and Bytecode. Proc. ACM Program. Lang. 5, OOPSLA, Article 136 (October 2021), 30 pages. https://doi.org/10.1145/3485513

- Haoran Xu. 2023. Building a baseline JIT for Lua automatically. (12 March 2023). Retrieved from https://sillycross.github.io/2023/05/12/2023-05-12/.

- A way of automatically turning a C interpreter into a fast template JIT compiler

Copy-And-Patch Compilation

- Compared to WebAssembly baseline compiler (`Liftoff`):
  - 5x faster code generation
  - 50% faster code

- Compared to traditional JIT toolchain (`LLVM -O0`):
  - 100x faster code generation
  - 15% faster code

- Compared to an optimizing JIT with hand-written assembly (`LuaJIT`):
  - Faster on 13/44 benchmarks
  - Only 35% slower overall

Copy-And-Patch Compilation

- At runtime, walk over a sequence of bytecode instructions.
- For each:
  - Copy some static, pre-compiled machine code into executable memory
  - Patch up instructions that need to have runtime data encoded into them

Copy-And-Patch Compilation

- At runtime, walk over a sequence of bytecode instructions.
- For each:
  - Copy some static, pre-compiled machine code into executable memory
  - Patch up instructions that need to have runtime data encoded into them

Copy-And-Patch Compilation

- Copy some static, pre-compiled machine code into executable memory
- Patch up instructions that need to have runtime data encoded into them

Copy-And-Patch Compilation

- When linking or loading a relocatable object file (ELF, COFF, Mach-O, etc.):
  - Copy some static, pre-compiled machine code into executable memory
  - Patch up instructions that need to have runtime data encoded into them

Copy-And-Patch Compilation

```
case _LOAD_FAST:
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    break;
```

Copy-And-Patch Compilation

```
PyObject *value = frame->localsplus[oparg];
Py_INCREF(value);
*stack_pointer++ = value;
```

Copy-And-Patch Compilation

```
int MAGICALLY_INSERT_THE_OPARG;
int MAGICALLY_CONTINUE_EXECUTION(_PyInterpreterFrame *frame,
                                 PyObject **stack_pointer);
int
_load_fast(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    int oparg = MAGICALLY_INSERT_THE_OPARG;
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    return MAGICALLY_CONTINUE_EXECUTION(frame, stack_pointer);
}
```

Copy-And-Patch Compilation

```
int MAGICALLY_INSERT_THE_OPARG;
int MAGICALLY_CONTINUE_EXECUTION(_PyInterpreterFrame *frame,
                                 PyObject **stack_pointer);
int
_load_fast(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    int oparg = MAGICALLY_INSERT_THE_OPARG;
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    return MAGICALLY_CONTINUE_EXECUTION(frame, stack_pointer);
}
```

Copy-And-Patch Compilation

```
extern int MAGICALLY_INSERT_THE_OPARG;
extern int MAGICALLY_CONTINUE_EXECUTION(_PyInterpreterFrame *frame,
                                        PyObject **stack_pointer);
int
_load_fast(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    int oparg = &MAGICALLY_INSERT_THE_OPARG;
    PyObject *value = frame->localsplus[oparg];
    Py_INCREF(value);
    *stack_pointer++ = value;
    __attribute__((musttail))
    return MAGICALLY_CONTINUE_EXECUTION(frame, stack_pointer);
}
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  MAGICALLY_INSERT_THE_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  MAGICALLY_INSERT_THE_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  MAGICALLY_INSERT_THE_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  MAGICALLY_INSERT_THE_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
0f b7 05 00 00 00 00 movzwl (%rip), %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_GOTPCREL  MAGICALLY_INSERT_THE_OPARG - 0x4
1d: R_X86_64_GOTPCRELX MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
66 90 b8 00 00 00 00 nop; mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_32          MAGICALLY_INSERT_THE_OPARG
1d: R_X86_64_GOTPCRELX MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
66 90 b8 00 00 00 00 nop; mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_32          MAGICALLY_INSERT_THE_OPARG
1d: R_X86_64_GOTPCRELX MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
66 90 b8 00 00 00 00 nop; mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
ff 25 00 00 00 00    jmpq *(%rip)

03: R_X86_64_32          MAGICALLY_INSERT_THE_OPARG
1d: R_X86_64_GOTPCRELX MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
66 90 b8 00 00 00 00 nop; mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
e9 00 00 00 00 90    jmp 0x0; nop

03: R_X86_64_32        MAGICALLY_INSERT_THE_OPARG
1c: R_X86_64_PC32      MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
66 90 b8 00 00 00 00 nop; mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
e9 00 00 00 00 90    jmp 0x0; nop

03: R_X86_64_32        MAGICALLY_INSERT_THE_OPARG
1c: R_X86_64_PC32      MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
66 90 b8 00 00 00 00 nop; mov $0x0, %eax
48 8b 44 c7 48       movq 0x48(%rdi,%rax,8), %rax
8b 08                movl (%rax), %ecx
ff c1                incl %ecx
74 02                je 0x14
89 08                movl %ecx, (%rax)
48 89 06             movq %rax, (%rsi)
48 83 c6 08          addq $0x8, %rsi
e9 00 00 00 00 90    jmp 0x0; nop

03: R_X86_64_32        MAGICALLY_INSERT_THE_OPARG
1c: R_X86_64_PC32      MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
b8 00 00 00 00          mov $0x0, %eax
48 8b 44 c7 48          movq 0x48(%rdi,%rax,8), %rax
8b 08                   movl (%rax), %ecx
ff c1                   incl %ecx
74 02                   je 0x12
89 08                   movl %ecx, (%rax)
48 89 06                movq %rax, (%rsi)
48 83 c6 08             addq $0x8, %rsi
e9 00 00 00 00          jmp 0x0

01: R_X86_64_32           MAGICALLY_INSERT_THE_OPARG
1a: R_X86_64_PC32         MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
b8 00 00 00 00          mov $0x0, %eax
48 8b 44 c7 48          movq 0x48(%rdi,%rax,8), %rax
8b 08                   movl (%rax), %ecx
ff c1                   incl %ecx
74 02                   je 0x12
89 08                   movl %ecx, (%rax)
48 89 06                movq %rax, (%rsi)
48 83 c6 08             addq $0x8, %rsi
e9 00 00 00 00          jmp 0x0

01: R_X86_64_32           MAGICALLY_INSERT_THE_OPARG
1a: R_X86_64_PC32         MAGICALLY_CONTINUE_EXECUTION - 0x4
```

Copy-And-Patch Compilation

```
b8 00 00 00 00          mov $0x0, %eax
48 8b 44 c7 48          movq 0x48(%rdi,%rax,8), %rax
8b 08                   movl (%rax), %ecx
ff c1                   incl %ecx
74 02                   je 0x12
89 08                   movl %ecx, (%rax)
48 89 06                movq %rax, (%rsi)
48 83 c6 08             addq $0x8, %rsi


01: R_X86_64_32           MAGICALLY_INSERT_THE_OPARG
```

Copy-And-Patch Compilation

```
static const unsigned char _LOAD_FAST_code_body[25] = {
      b8,   00,   00,   00,   00,   48,   8b,   44,
      c7,   48,   8b,   08,   ff,   c1,   74,   02,
      89,   08,   48,   89,   06,   48,   83,   c6,
      08,
};

static const Hole _LOAD_FAST_code_holes[1] = {
    {  01, R_X86_64_32, MAGICALLY_INSERT_THE_OPARG, 0x0},
};
```

Copy-And-Patch Compilation

```
static const unsigned char _LOAD_FAST_code_body[25] = {
    0xb8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x44,
    0xc7, 0x48, 0x8b, 0x08, 0xff, 0xc1, 0x74, 0x02,
    0x89, 0x08, 0x48, 0x89, 0x06, 0x48, 0x83, 0xc6,
    0x08,
};

static const Hole _LOAD_FAST_code_holes[1] = {
    {0x01, R_X86_64_32, MAGICALLY_INSERT_THE_OPARG, 0x0},
};
```

Copy-And-Patch Compilation

- Build time:
  - ~900 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple C (hand-written)

Copy-And-Patch Compilation

- Build time:
  - ~900 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple-ish C (hand-written)

Copy-And-Patch Compilation

- Build time:
  - ~900 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple-ish C (hand-written)
  - ~4000 lines of simple C (generated)

Copy-And-Patch Compilation

- Build time:
  - ~900 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple-ish C (hand-written)
  - ~4000 lines of simple C (generated)
  - No dependencies

Copy-And-Patch Compilation

- Build time:
  - ~900 lines of complex Python
  - ~100 lines of complex C
  - LLVM dependency

- Run time:
  - ~400 lines of simple-ish C (hand-written)
  - ~4000 lines of simple C (generated)
  - No dependencies

Platform Support

Platform Support

x86-64

- `x86_64-apple-darwin/clang`
- `x86_64-pc-windows-msvc/msvc`
- `x86_64-unknown-linux-gnu/clang`
- `x86_64-unknown-linux-gnu/gcc`

Platform Support

x86 and x86-64

- i686-pc-windows-msvc/msvc
- x86_64-apple-darwin/clang
- x86_64-pc-windows-msvc/msvc
- x86_64-unknown-linux-gnu/clang
- x86_64-unknown-linux-gnu/gcc

Platform Support

AArch64, x86, and x86-64

- aarch64-apple-darwin/clang
- aarch64-pc-windows-msvc/msvc
- aarch64-unknown-linux-gnu/clang
- aarch64-unknown-linux-gnu/gcc
- i686-pc-windows-msvc/msvc
- x86_64-apple-darwin/clang
- x86_64-pc-windows-msvc/msvc
- x86_64-unknown-linux-gnu/clang
- x86_64-unknown-linux-gnu/gcc

Future Work

Future Work

Instruction Variants

Future Work

Instruction Variants

```
static const unsigned char _LOAD_FAST_code_body[25] = {
    0xb8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x44,
    0xc7, 0x48, 0x8b, 0x08, 0xff, 0xc1, 0x74, 0x02,
    0x89, 0x08, 0x48, 0x89, 0x06, 0x48, 0x83, 0xc6,
    0x08,
};

static const Hole _LOAD_FAST_code_holes[1] = {
    {0x01, R_X86_64_32, MAGICALLY_INSERT_THE_OPARG, 0x0},
};
```

Future Work

Instruction Variants

```
static const unsigned char _LOAD_FAST_code_body[25] = {
    0xb8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8b, 0x44,
    0xc7, 0x48, 0x8b, 0x08, 0xff, 0xc1, 0x74, 0x02,
    0x89, 0x08, 0x48, 0x89, 0x06, 0x48, 0x83, 0xc6,
    0x08,
};

static const Hole _LOAD_FAST_code_holes[1] = {
    {0x01, R_X86_64_32, MAGICALLY_INSERT_THE_OPARG, 0x0},
};
```

Future Work

Instruction Variants

```
static const unsigned char _LOAD_FAST_0_code_body[19] = {
    0x48, 0x8b, 0x47, 0x48, 0x8b, 0x08, 0xff, 0xc1,
    0x74, 0x02, 0x89, 0x08, 0x48, 0x89, 0x06, 0x48,
    0x83, 0xc6, 0x08,

};

static const Hole _LOAD_FAST_code_holes[1] = {

};
```

Future Work

Superinstructions

Future Work

Superinstructions

```
static const unsigned char _TO_BOOL_INT_code_body[190] = {
    0x41, 0x57, 0x41, 0x56, 0x41, 0x54, 0x53, 0x50, 0x48, 0x89, 0xd3, 0x49, 0x89, 0xf6, 0x49, 0x89,
    0xff, 0x48, 0x8b, 0x7e, 0xf8, 0x48, 0x8b, 0x47, 0x08, 0x48, 0x3b, 0x05, 0x00, 0x00, 0x00, 0x00,
    0x74, 0x54, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x05, 0x00,
    0x00, 0x00, 0x00, 0x48, 0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x8b, 0x11, 0xff, 0xc2, 0x74,
    0x14, 0x89, 0x11, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x8c,
    0x08, 0x80, 0x00, 0x00, 0x00, 0x48, 0x89, 0x83, 0x28, 0x01, 0x00, 0x00, 0x48, 0x8b, 0x41, 0x70,
    0x4c, 0x89, 0xff, 0x4c, 0x89, 0xf6, 0x48, 0x89, 0xda, 0x48, 0x83, 0xc4, 0x08, 0x5b, 0x41, 0x5c,
    0x41, 0x5e, 0x41, 0x5f, 0xff, 0xe0, 0x8b, 0x47, 0x10, 0x83, 0xe0, 0x03, 0x83, 0xf8, 0x01, 0x75,
    0x09, 0x4c, 0x8b, 0x25, 0x00, 0x00, 0x00, 0x00, 0xeb, 0x1c, 0x48, 0x8b, 0x07, 0x4c, 0x8b, 0x25,
    0x00, 0x00, 0x00, 0x00, 0x85, 0xc0, 0x78, 0x0e, 0x48, 0xff, 0xc8, 0x48, 0x89, 0x07, 0x75, 0x06,
    0xff, 0x15, 0x00, 0x00, 0x00, 0x00, 0x4d, 0x89, 0x66, 0xf8, 0x4c, 0x89, 0xff, 0x4c, 0x89, 0xf6,
    0x48, 0x89, 0xda, 0x48, 0x83, 0xc4, 0x08, 0x5b, 0x41, 0x5c, 0x41, 0x5e, 0x41, 0x5f,
};
```

Future Work

Superinstructions

```
static const unsigned char _TO_BOOL_INT_code_body[190] = {
    0x41, 0x57, 0x41, 0x56, 0x41, 0x54, 0x53, 0x50, 0x48, 0x89, 0xd3, 0x49, 0x89, 0xf6, 0x49, 0x89,
    0xff, 0x48, 0x8b, 0x7e, 0xf8, 0x48, 0x8b, 0x47, 0x08, 0x48, 0x3b, 0x05, 0x00, 0x00, 0x00, 0x00,
    0x74, 0x54, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x05, 0x00,
    0x00, 0x00, 0x00, 0x48, 0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x8b, 0x11, 0xff, 0xc2, 0x74,
    0x14, 0x89, 0x11, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x8c,
    0x08, 0x80, 0x00, 0x00, 0x00, 0x48, 0x89, 0x83, 0x28, 0x01, 0x00, 0x00, 0x48, 0x8b, 0x41, 0x70,
    0x4c, 0x89, 0xff, 0x4c, 0x89, 0xf6, 0x48, 0x89, 0xda, 0x48, 0x83, 0xc4, 0x08, 0x5b, 0x41, 0x5c,
    0x41, 0x5e, 0x41, 0x5f, 0xff, 0xe0, 0x8b, 0x47, 0x10, 0x83, 0xe0, 0x03, 0x83, 0xf8, 0x01, 0x75,
    0x09, 0x4c, 0x8b, 0x25, 0x00, 0x00, 0x00, 0x00, 0xeb, 0x1c, 0x48, 0x8b, 0x07, 0x4c, 0x8b, 0x25,
    0x00, 0x00, 0x00, 0x00, 0x85, 0xc0, 0x78, 0x0e, 0x48, 0xff, 0xc8, 0x48, 0x89, 0x07, 0x75, 0x06,
    0xff, 0x15, 0x00, 0x00, 0x00, 0x00, 0x4d, 0x89, 0x66, 0xf8, 0x4c, 0x89, 0xff, 0x4c, 0x89, 0xf6,
    0x48, 0x89, 0xda, 0x48, 0x83, 0xc4, 0x08, 0x5b, 0x41, 0x5c, 0x41, 0x5e, 0x41, 0x5f,
};
static const unsigned char _GUARD_IS_TRUE_POP_code_body[84] = {
    0x48, 0x8b, 0x46, 0xf8, 0x48, 0x83, 0xc6, 0xf8, 0x48, 0x3b, 0x05, 0x00, 0x00, 0x00, 0x00, 0x74,
    0x43, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x05, 0x00, 0x00,
    0x00, 0x00, 0x48, 0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x44, 0x8b, 0x01, 0x41, 0xff, 0xc0,
    0x74, 0x15, 0x44, 0x89, 0x01, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48,
    0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x48, 0x89, 0x82, 0x28, 0x01, 0x00, 0x00, 0x48, 0x8b,
    0x41, 0x70, 0xff, 0xe0,
};
```

Future Work

Superinstructions

```
static const unsigned char _TO_BOOL_INT_code_body[190] = {
    0x41, 0x57, 0x41, 0x56, 0x41, 0x54, 0x53, 0x50, 0x48, 0x89, 0xd3, 0x49, 0x89, 0xf6, 0x49, 0x89,
    0xff, 0x48, 0x8b, 0x7e, 0xf8, 0x48, 0x8b, 0x47, 0x08, 0x48, 0x3b, 0x05, 0x00, 0x00, 0x00, 0x00,
    0x74, 0x54, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x05, 0x00,
    0x00, 0x00, 0x00, 0x48, 0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x8b, 0x11, 0xff, 0xc2, 0x74,
    0x14, 0x89, 0x11, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x8c,
    0x08, 0x80, 0x00, 0x00, 0x00, 0x48, 0x89, 0x83, 0x28, 0x01, 0x00, 0x00, 0x48, 0x8b, 0x41, 0x70,
    0x4c, 0x89, 0xff, 0x4c, 0x89, 0xf6, 0x48, 0x89, 0xda, 0x48, 0x83, 0xc4, 0x08, 0x5b, 0x41, 0x5c,
    0x41, 0x5e, 0x41, 0x5f, 0xff, 0xe0, 0x8b, 0x47, 0x10, 0x83, 0xe0, 0x03, 0x83, 0xf8, 0x01, 0x75,
    0x09, 0x4c, 0x8b, 0x25, 0x00, 0x00, 0x00, 0x00, 0xeb, 0x1c, 0x48, 0x8b, 0x07, 0x4c, 0x8b, 0x25,
    0x00, 0x00, 0x00, 0x00, 0x85, 0xc0, 0x78, 0x0e, 0x48, 0xff, 0xc8, 0x48, 0x89, 0x07, 0x75, 0x06,
    0xff, 0x15, 0x00, 0x00, 0x00, 0x00, 0x4d, 0x89, 0x66, 0xf8, 0x4c, 0x89, 0xff, 0x4c, 0x89, 0xf6,
    0x48, 0x89, 0xda, 0x48, 0x83, 0xc4, 0x08, 0x5b, 0x41, 0x5c, 0x41, 0x5e, 0x41, 0x5f,
};
static const unsigned char _GUARD_IS_TRUE_POP_code_body[84] = {
    0x48, 0x8b, 0x46, 0xf8, 0x48, 0x83, 0xc6, 0xf8, 0x48, 0x3b, 0x05, 0x00, 0x00, 0x00, 0x00, 0x74,
    0x43, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x05, 0x00, 0x00,
    0x00, 0x00, 0x48, 0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x44, 0x8b, 0x01, 0x41, 0xff, 0xc0,
    0x74, 0x15, 0x44, 0x89, 0x01, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48,
    0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x48, 0x89, 0x82, 0x28, 0x01, 0x00, 0x00, 0x48, 0x8b,
    0x41, 0x70, 0xff, 0xe0,
};
```

Future Work

Superinstructions

```
static const unsigned char _TO_BOOL_INT__GUARD_IS_TRUE_POP_code_body[274] = {
    0x41, 0x57, 0x41, 0x56, 0x41, 0x54, 0x53, 0x50, 0x48, 0x89, 0xd3, 0x49, 0x89, 0xf6, 0x49, 0x89,
    0xff, 0x48, 0x8b, 0x7e, 0xf8, 0x48, 0x8b, 0x47, 0x08, 0x48, 0x3b, 0x05, 0x00, 0x00, 0x00, 0x00,
    0x74, 0x54, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x05, 0x00,
    0x00, 0x00, 0x00, 0x48, 0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x8b, 0x11, 0xff, 0xc2, 0x74,
    0x14, 0x89, 0x11, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x8c,
    0x08, 0x80, 0x00, 0x00, 0x00, 0x48, 0x89, 0x83, 0x28, 0x01, 0x00, 0x00, 0x48, 0x8b, 0x41, 0x70,
    0x4c, 0x89, 0xff, 0x4c, 0x89, 0xf6, 0x48, 0x89, 0xda, 0x48, 0x83, 0xc4, 0x08, 0x5b, 0x41, 0x5c,
    0x41, 0x5e, 0x41, 0x5f, 0xff, 0xe0, 0x8b, 0x47, 0x10, 0x83, 0xe0, 0x03, 0x83, 0xf8, 0x01, 0x75,
    0x09, 0x4c, 0x8b, 0x25, 0x00, 0x00, 0x00, 0x00, 0xeb, 0x1c, 0x48, 0x8b, 0x07, 0x4c, 0x8b, 0x25,
    0x00, 0x00, 0x00, 0x00, 0x85, 0xc0, 0x78, 0x0e, 0x48, 0xff, 0xc8, 0x48, 0x89, 0x07, 0x75, 0x06,
    0xff, 0x15, 0x00, 0x00, 0x00, 0x00, 0x4d, 0x89, 0x66, 0xf8, 0x4c, 0x89, 0xff, 0x4c, 0x89, 0xf6,
    0x48, 0x89, 0xda, 0x48, 0x83, 0xc4, 0x08, 0x5b, 0x41, 0x5c, 0x41, 0x5e, 0x41, 0x5f, 0x48, 0x8b,
    0x46, 0xf8, 0x48, 0x83, 0xc6, 0xf8, 0x48, 0x3b, 0x05, 0x00, 0x00, 0x00, 0x00, 0x74, 0x43, 0x8b,
    0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x05, 0x00, 0x00, 0x00, 0x00,
    0x48, 0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x44, 0x8b, 0x01, 0x41, 0xff, 0xc0, 0x74, 0x15,
    0x44, 0x89, 0x01, 0x8b, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x8c,
    0x08, 0x80, 0x00, 0x00, 0x00, 0x48, 0x89, 0x82, 0x28, 0x01, 0x00, 0x00, 0x48, 0x8b, 0x41, 0x70,
    0xff, 0xe0,
};
```

Future Work

Superinstructions

```
static const unsigned char _TO_BOOL_INT__GUARD_IS_TRUE_POP_code_body[233] = {
    0x41, 0x57, 0x41, 0x56, 0x53, 0x49, 0x89, 0xd7, 0x48, 0x89, 0xf3, 0x49, 0x89, 0xfe, 0x48, 0x8b,
    0x7e, 0xf8, 0x48, 0x8b, 0x47, 0x08, 0x48, 0x3b, 0x05, 0x00, 0x00, 0x00, 0x00, 0x74, 0x36, 0x48,
    0x8b, 0x05, 0x00, 0x00, 0x00, 0x00, 0x49, 0x89, 0x87, 0x28, 0x01, 0x00, 0x00, 0x4c, 0x29, 0xf3,
    0x48, 0x83, 0xc3, 0xb8, 0x48, 0xc1, 0xeb, 0x03, 0x41, 0x89, 0x5e, 0x40, 0x49, 0x8b, 0x06, 0x8b,
    0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x04, 0x48, 0x48, 0x05, 0xc8, 0x00, 0x00, 0x00, 0x5b,
    0x41, 0x5e, 0x41, 0x5f, 0xc3, 0x48, 0x83, 0xc3, 0xf8, 0x8b, 0x47, 0x10, 0x83, 0xe0, 0x03, 0x83,
    0xf8, 0x01, 0x75, 0x58, 0x48, 0x8b, 0x05, 0x00, 0x00, 0x00, 0x00, 0x48, 0x89, 0x03, 0x8b, 0x0d,
    0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x05, 0x00, 0x00, 0x00, 0x00, 0x48,
    0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00, 0x00, 0x8b, 0x11, 0xff, 0xc2, 0x74, 0x14, 0x89, 0x11, 0x8b,
    0x0d, 0x00, 0x00, 0x00, 0x00, 0x48, 0xc1, 0xe1, 0x04, 0x48, 0x8b, 0x8c, 0x08, 0x80, 0x00, 0x00,
    0x00, 0x49, 0x89, 0x87, 0x28, 0x01, 0x00, 0x00, 0x48, 0x8b, 0x41, 0x70, 0x4c, 0x89, 0xf7, 0x48,
    0x89, 0xde, 0x4c, 0x89, 0xfa, 0x5b, 0x41, 0x5e, 0x41, 0x5f, 0xff, 0xe0, 0x48, 0x8b, 0x07, 0x85,
    0xc0, 0x78, 0x0e, 0x48, 0xff, 0xc8, 0x48, 0x89, 0x07, 0x75, 0x06, 0xff, 0x15, 0x00, 0x00, 0x00,
    0x00, 0x48, 0x8b, 0x05, 0x00, 0x00, 0x00, 0x00, 0x48, 0x89, 0x03, 0x4c, 0x89, 0xf7, 0x48, 0x89,
    0xde, 0x4c, 0x89, 0xfa, 0x5b, 0x41, 0x5e, 0x41, 0x5f,


};
```

Future Work

Stack Caching

- M. Anton Ertl. 1995. Stack caching for interpreters. In Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation (PLDI '95). Association for Computing Machinery, New York, NY, USA, 315–327. https://doi.org/10.1145/207110.207165

Future Work

Stack Caching

```
int
_binary_op_add_int(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *lhs = stack_pointer[-2];
    PyObject *rhs = stack_pointer[-1];
    PyObject *res = _PyLong_Add(lhs, rhs);
    if (res == NULL) return -1;
    Py_DECREF(lhs);
    Py_DECREF(rhs);
    stack_pointer[-2] = res;
    stack_pointer -= 1;
    __attribute__((musttail))
    return MAGIC_CONTINUATION(frame, stack_pointer);
}
```

Future Work

Stack Caching

```
int
_binary_op_add_int(_PyInterpreterFrame *frame, PyObject **stack_pointer)
{
    PyObject *lhs = stack_pointer[-2];
    PyObject *rhs = stack_pointer[-1];
    PyObject *res = _PyLong_Add(lhs, rhs);
    if (res == NULL) return -1;
    Py_DECREF(lhs);
    Py_DECREF(rhs);
    stack_pointer[-2] = res;
    stack_pointer -= 1;
    __attribute__((musttail))
    return MAGIC_CONTINUATION(frame, stack_pointer);
}
```

Future Work

Stack Caching

```
int
_binary_op_add_int(_PyInterpreterFrame *frame, PyObject **stack_pointer,
                   PyObject *tos)
{
    PyObject *lhs = stack_pointer[-2];
    PyObject *rhs = stack_pointer[-1];
    PyObject *res = _PyLong_Add(lhs, rhs);
    if (res == NULL) return -1;
    Py_DECREF(lhs);
    Py_DECREF(rhs);
    stack_pointer[-2] = res;
    stack_pointer -= 1;
    __attribute__((musttail))
    return MAGIC_CONTINUATION(frame, stack_pointer, tos);
}
```

Future Work

Stack Caching

```
int
_binary_op_add_int(_PyInterpreterFrame *frame, PyObject **stack_pointer,
                   PyObject *empty)
{
    PyObject *lhs = stack_pointer[-2];
    PyObject *rhs = stack_pointer[-1];
    PyObject *res = _PyLong_Add(lhs, rhs);
    if (res == NULL) return -1;
    Py_DECREF(lhs);
    Py_DECREF(rhs);
    stack_pointer -= 2;
    __attribute__((musttail))
    return MAGIC_CONTINUATION(frame, stack_pointer, res);
}
```

Future Work

Stack Caching

```
int
_binary_op_add_int(_PyInterpreterFrame *frame, PyObject **stack_pointer,
                   PyObject *rhs)
{
    PyObject *lhs = stack_pointer[-1];
    PyObject *res = _PyLong_Add(lhs, rhs);
    if (res == NULL) return -1;
    Py_DECREF(lhs);
    Py_DECREF(rhs);
    stack_pointer -= 1;
    __attribute__((musttail))
    return MAGIC_CONTINUATION(frame, stack_pointer, res);
}
```

Future Work

Stack Caching

```
int
_binary_op_add_int(_PyInterpreterFrame *frame, PyObject **stack_pointer,
                   PyObject *rhs, PyObject *lhs)
{
    PyObject *res = _PyLong_Add(lhs, rhs);
    if (res == NULL) return -1;
    Py_DECREF(lhs);
    Py_DECREF(rhs);
    __attribute__((musttail))
    return MAGIC_CONTINUATION(frame, stack_pointer, NULL, res);
}
```

Future Work

Stack Caching

```
__attribute__((preserve_none)) int
_binary_op_add_int(_PyInterpreterFrame *frame, PyObject **stack_pointer,
                   PyObject *tos1, PyObject *tos2, PyObject *tos3,
                   PyObject *tos4, PyObject *tos5, PyObject *tos6,
                   PyObject *tos7, PyObject *tos8, PyObject *tos9)
{
    PyObject *res = _PyLong_Add(tos2, tos1);
    if (res == NULL) return -1;
    Py_DECREF(tos2);
    Py_DECREF(tos1);
    __attribute__((musttail))
    return MAGIC_CONTINUATION(frame, stack_pointer, NULL, res,
                              tos3, tos4, tos5, tos6, tos7, tos8, tos9);
}
```

Future Work

Trace Stitching

- Andreas Gal, Brendan Eich, Mike Shaver, David Anderson, David Mandelin, Mohammad R. Haghighat, Blake Kaplan, Graydon Hoare, Boris Zbarsky, Jason Orendorff, Jesse Ruderman, Edwin W. Smith, Rick Reitmaier, Michael Bebenita, Mason Chang, and Michael Franz. 2009. Trace-based just-in-time type specialization for dynamic languages. In Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '09). Association for Computing Machinery, New York, NY, USA, 465–478. https://doi.org/10.1145/1542476.1542528

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    s = ""
    if not i % 3: bail()
    if not i % 5: bail()
    if s: bail()
    print(i)
    goto T0
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    s = ""
    if not i % 3: bail()
    if not i % 5: bail()
    print(i)
    goto T0
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    s = ""
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: bail()
T2: bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: s = ""
    bail()
T2: s = ""
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: s = ""
    bail()
T2: s = ""
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: s = ""
    s += "fizz"
    if not i % 5: bail()
    if s: print(s)
    else: bail()
    goto T0
T2: s = ""
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: s = "fizz"
    if not i % 5: bail()
    if s: print(s)
    else: bail()
    goto T0
T2: s = ""
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: s = "fizz"
    if not i % 5: bail()
    print("fizz")
    goto T0
T2: s = ""
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: s = "fizz"
    if not i % 5: goto T3
    print("fizz")
    goto T0
T2: s = ""
    bail()
T3: bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: s = ""
    bail()
T3: s = "fizz"
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: s = ""
    bail()
T3: s = "fizz"
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: s = ""
    s += "buzz"
    if s: print(s)
    else: print(i)
    goto T0
T3: s = "fizz"
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: s = "buzz"
    if s: print(s)
    else: print(i)
    goto T0
T3: s = "fizz"
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: s = "buzz"
    print("buzz")
    goto T0
T3: s = "fizz"
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: s = "fizz"
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: s = "fizz"
    bail()
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: s = "fizz"
    s += "buzz"
    if s: print(s)
    else: print(i)
    goto T0
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: s = "fizzbuzz"
    if s: print(s)
    else: print(i)
    goto T0
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: s = "fizzbuzz"
    print("fizzbuzz")
    goto T0
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: print("fizzbuzz")
    goto T0
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: print("fizzbuzz")
    goto T0
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
...
```

```
T0: if exhausted(iterator): bail()
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: print("fizzbuzz")
    goto T0
```

Future Work

Trace Stitching

```
for i in range(n):
    s = ""
    if not i % 3: s += "fizz"
    if not i % 5: s += "buzz"
    if s: print(s)
    else: print(i)
...
```

```
T0: if exhausted(iterator): goto T4
    i = next(iterator)
    if not i % 3: goto T1
    if not i % 5: goto T2
    print(i)
    goto T0
T1: if not i % 5: goto T3
    print("fizz")
    goto T0
T2: print("buzz")
    goto T0
T3: print("fizzbuzz")
    goto T0
T4: ...
```

Future Work

Trace Stitching

- Works for:
  - ...explicit control flow.
  - ...polymorphic code.
  - ...pretty much any reason we might branch in JIT code.

Other Projects

Other Projects

- Better benchmarks, with more emphasis on modern idioms.
- True function inlining.
- Incremental GC.
- Improving the object model.
- Free-threading.
- Subinterpreters.
- ...?

Python Developer’s Guide

Python Developer's Guide

- devguide.python.org
- devguide.python.org/internals
- devguide.python.org/internals/parser
- devguide.python.org/internals/compiler
- devguide.python.org/internals/interpreter
- devguide.python.org/internals/garbage-collector

faster-cpython/ideas

Thank you!

@brandtbucher

Thank you!

@brandtbucher | brandt@python.org

Thank you!

@brandtbucher | brandt@python.org | https://xkcd.com/451