Unladen Swallow: Fewer coconuts, faster Python

Collin Winter
collinwinter@google.com
unladen-swallow@googlegroups.com

Why is Google doing this?

- Lots of Python at Google.
  - Python one of Google's three primary languages.
  - Python enables fast development, rapid prototyping.
  - Engineers should use the language they want...
  - ...and it should be fast.

- Biggest user: YouTube
  - YouTube is pure-Python.
  - #2 search site on the Internet, behind google.com.

Original Goals

- Make Python 5x faster.
- Source-compatible with existing Python code.
- Source-compatible with existing C extension modules.
- Focus on ease of migration.
- Open-source everything, merge back into CPython.

Why is Python slow?

```
def add(a, b):
    return a + b
```

- Everything is an object.
- Everything is a method call, eventually.
- Ducktyping means we can’t predict receiver types.

```
def add(a, b):
    return a + b
```

```
define %struct._object* @"PwPwIdd"(%struct._frame* %frame) {
entry:
  %tlsstackvar19 = alloca [2 x %struct._object*], align 4
  %tlsstackvar19.sub = getelementptr [2 x %struct._object*]* %tlsstackvar19, i32 0, i32 0
  %v0 = load %struct._ts** @_Py_ThreadsBase_Gaffered
  %stack_pointer = getelementptr %struct._frame* %frame, i32 0, i32 8
  %stack_bottom = load %struct._object*** %v_v_stackstack
  %f_stackmap = getelementptr %struct._frame* %frame, i32 0, i32 9
  %stack_top = load %struct._object*** %f_stackmap
  %name_tracing = getelementptr %struct._ts* %v0, i32 0, i32 5
  %name_tracing1 = load i32* %name_tracing
  %v1 = icmp eq i32 %name_tracing1, 0
  br i1 %v1, label %continue_entry, label %trace_enter_function

bail_to_interpreter:                                  ; preds = %call_trace, %trace_enter_function
  %f_one_ffvm = getelementptr %struct._frame* %frame, i32 0, i32 16
  %v2 = load i32* %f_one_ffvm
  store %struct._object*** %stack_bottom, %struct._object*** %f_stackmap
  %f_ffstack = getelementptr %struct._frame* %frame, i32 0, i32 19
  %v3 = load i8** %f_ffstack
  %f_localsplus8 = getelementptr %struct._frame* %frame, i32 0, i32 23
  %v4 = bitcast [1 x %struct._object*]* %f_localsplus to i8*
  %v5 = bitcast [2 x %struct._object*]* %tlsstackvar19 to i8*
  call void @llvm.memcpy.i64.i8* mcopy(i8* %v3, i8* %v4, i64 ptrtoint (%struct._object** getelementptr (%struct._object** null, i32 2) to i64), i32 1) nounwind
  %v6 = tail call %struct._object* @PyEval_EvalFrameEx(%struct._frame* %frame)
  ret %struct._object* %v6

trace_enter_function:                                 ; preds = %entry
  %f_bailed_from_ffvm = getelementptr %struct._frame* %frame, i32 0, i32 20
  store i8 1, i8* %f_bailed_from_ffvm
  br label %bail_to_interpreter

continue_entry:                                       ; preds = %entry
  %f_localsplus = getelementptr %struct._frame* %frame, i32 0, i32 23
  %v7 = bitcast [1 x %struct._object*]* %f_localsplus to i8*
  %v8 = bitcast [2 x %struct._object*]* %tlsstackvar19 to i8*
  call void @llvm.memcpy.i64.i8* mcopy(i8* %v7, i8* %v8, i64 ptrtoint (%struct._object** getelementptr (%struct._object** null, i32 2) to i64), i32 1) nounwind
  %f_lineno = getelementptr %struct._frame* %v5, i32 0, i32 17
  store i32 2, i32* %f_lineno
  %v9 = load i32* @_Py_TracingPossible
  %v10 = icmp eq i32 %v9, 0
  br i1 %v10, label %line_start, label %call_trace

call_trace:                                           ; preds = %PropagateExceptionOnNull_propagate
  call void @_PyEval_CallExcTrace(%struct._ts* %v0, %struct._frame* %frame)
  br label %unwind_loop_header.preheader

unwind_loop_header.preheader:                         ; preds = %PropagateExceptionOnNull_pass, %call_exc_trace, %PropagateExceptionOnNull_pass
  %unwind_reason.ph = phi i8 [ 8, %PropagateExceptionOnNull_pass ], [ 2, %call_exc_trace ], [ 2, %PropagateExceptionOnNull_propagate ]
  %retval_addr.0.ph = phi %struct._object* [ %tmrcp_result, %PropagateExceptionOnNull_pass ], [ null, %call_exc_trace ], [ null, %PropagateExceptionOnNull_propagate ]
  %unwind_reason = phi i8 [ %unwind_reason.ph, %unwind_loop_header.preheader ], [ 2, %call_exc_trace ], [ 2, %PropagateExceptionOnNull_propagate ]
  %unwind_retval_addr_exception = icmp eq i8 %unwind_reason, 2
  br label %pop_loop5

pop_loop5:                                            ; preds = %13, %18, %pop_stack6, %unwind_loop_header.preheader
  %stack_pointer_addr.2 = phi %struct._object*** [ %stack_bottom, %unwind_loop_header.preheader ], [ %10, %pop_stack6 ], [ %10, %18 ], [ %10, %13 ]
  %9 = icmp sgt %struct._object*** %stack_pointer_addr.2, %stack_bottom
  br i1 %9, label %pop_stack6, label %pop_done7

pop_stack6:                                           ; preds = %pop_loop5
  %10 = getelementptr %struct._object*** %stack_pointer_addr.2, i32 -1
  %11 = load %struct._object** %10
  %12 = icmp eq %struct._object* %11, null
  br i1 %12, label %pop_loop5, label %13

; <label>:13                                          ; preds = %pop_stack6
  %14 = getelementptr %struct._object* %11, i32 0, i32 0
  %15 = load i32* %14
  %16 = add i32 %15, -1
  store i32 %16, i32* %14
  %17 = icmp eq i32 %16, 0
  br i1 %17, label %18, label %pop_loop5

; <label>:18                                          ; preds = %13
  %19 = getelementptr %struct._object* %11, i32 0, i32 1
  %20 = load %struct._typeobject** %19
  %21 = getelementptr %struct._typeobject* %20, i32 0, i32 6
  %22 = load void (%struct._object*)** %21
  tail call void %22(%struct._object* %11) nounwind
  br label %pop_loop5

pop_done7:                                            ; preds = %pop_loop5
  %23 = icmp eq i8 %unwind_reason.ph, 8
  %retval_addr.2 = select i1 %23, %struct._object* %retval_addr.0.ph, %struct._object* null
  %24 = icmp eq i8 %unwind_reason, 2
  %25 = icmp eq i32 %24, 0
  br i1 %25, label %check_frame_exception, label %trace_leave_function

check_frame_exception:                                ; preds = %32, %trace_raised, %trace_leave_function, %pop_done7, %27
  %retval_addr.2 = phi %struct._object* [ null, %27 ], [ %retval_addr.1, %pop_done7 ], [ %retval_addr.1, %trace_leave_function ], [ null, %trace_raised ], [ null, %32 ]
  %tstate0 = getelementptr %struct._ts* %v0, i32 0, i32 2
  %7tstate = load %struct._frame** %tstate0
  %26 = getelementptr %struct._frame* %7tstate, %struct._frame* %frame, i32 0, i32 11
  %7tstate1 = load %struct._exc_type* = load %struct._object** %f_exc_type
  %28 = icmp eq %struct._object* %7tstate1, %struct._exc_type*, null
  br i1 %28, label %check_return, label %leave_frame_exception

trace_leave_function:                                 ; preds = %pop_done7
  %in_return = icmp eq i8 %unwind_reason, addr.0.ph, 8
  %in_exception = icmp eq i8 %unwind_reason, addr.0.ph, 2
  %27 = zext i1 %in_return to i8
  %29 = icmp eq %struct._object* %retval_addr.0, null
  %30 = call i32 @_PyEval_TraceLeaveFunction(%struct._ts* %0, %struct._frame* %frame, %struct._object* %retval_addr.1, i8 %27, i8 %28)
  %30 = icmp eq i32 %30, 0
  br i1 %30, label %check_frame_exception, label %tracer_raised

tracer_raised:                                        ; preds = %trace_leave_function
  %31 = icmp eq %struct._object* %retval_addr.1, null
  br i1 %31, label %check_frame_exception, label %32

; <label>:32                                          ; preds = %tracer_raised
  %33 = getelementptr %struct._object* %retval_addr.1, i32 0, i32 0
  %34 = load i32* %33
  %35 = add i32 %34, -1
  store i32 %35, i32* %33
  %36 = icmp eq i32 %35, 0
  br i1 %36, label %37, label %check_frame_exception

; <label>:37                                          ; preds = %32
  %38 = getelementptr %struct._object* %retval_addr.1, i32 0, i32 1
  %39 = load %struct._typeobject** %38
  %40 = getelementptr %struct._typeobject* %39, i32 0, i32 6
  %41 = load void (%struct._object*)** %40
  tail call void %41(%struct._object* %retval_addr.1) nounwind
  br label %check_frame_exception

have_frame_exception:                                 ; preds = %check_frame_exception
  call void @_PyEval_RaiseIfExcInfo(%struct._ts* %0)
  ret %struct._object* %retval_addr.2

finish_return:                                        ; preds = %check_frame_exception
  ret %struct._object* %retval_addr.2

line_start:                                           ; preds = %continue_entry
  %tlsAST_loaded = load %struct._object** %tlsstackvar19.sub
  %42 = getelementptr %struct._object** %AST_loaded, i32 0, i32 0
  %43 = load i32* %42
  %44 = add i32 %43, 1
  store i32 %44, i32* %42
  store %struct._object** %tlsAST_loaded, %struct._object** %stack_bottom
  %45 = getelementptr [2 x %struct._object*]* %tlsstackvar19, i32 0, i32 1
  %46 = getelementptr [2 x %struct._object*]* %tlsstackvar19, i32 0, i32 1
  %tlsAST_loaded12 = load %struct._object** %46
  %47 = getelementptr %struct._object** %AST_loaded12, i32 0, i32 0
  %48 = load i32* %47
  %49 = add i32 %48, 1
  store i32 %49, i32* %47
  store %struct._object** %AST_loaded12, %struct._object** %45
  %50 = load %struct._object** %stack_bottom
  %51tmp_result = call %struct._object* @PyNumber_Add(%struct._object* %50, %struct._object* %tlsAST_loaded12)
  %52 = getelementptr %struct._object* %50, i32 0, i32 0
  %53 = load i32* %52
  %54 = add i32 %53, -1
  store i32 %54, i32* %52
  %55 = icmp eq i32 %53, 1
  br i1 %55, label %56, label %_PyLlvm_WrapDecref.exit17

; <label>:55                                          ; preds = %line_start
  %56 = getelementptr %struct._object* %50, i32 0, i32 1
  %57 = load %struct._typeobject** %56
  %58 = getelementptr %struct._typeobject* %57, i32 0, i32 6
  %59 = load void (%struct._object*)** %58
  call void %59(%struct._object* %50) nounwind
  br label %_PyLlvm_WrapDecref.exit17

_PyLlvm_WrapDecref.exit17:                            ; preds = %line_start, %55
  %60 = load i32* %47
  %61 = add i32 %60, 1
  store i32 %61, i32* %47
  %62 = icmp eq i32 %61, 0
  br i1 %62, label %63, label %_PyLlvm_WrapDecref.exit18

; <label>:63                                          ; preds = %_PyLlvm_WrapDecref.exit17
  %64 = getelementptr %struct._object** %AST_loaded12, i32 0, i32 1
  %65 = load %struct._typeobject** %64
  %66 = getelementptr %struct._typeobject* %65, i32 0, i32 6
  %67 = load void (%struct._object*)** %66
  call void %67(%struct._object* %AST_loaded12) nounwind
  br label %_PyLlvm_WrapDecref.exit18

_PyLlvm_WrapDecref.exit18:                            ; preds = %_PyLlvm_WrapDecref.exit17, %63
  %68 = icmp eq %struct._object* %tmrcp_result, null
  br i1 %68, label %PropagateExceptionOnNull_propagate, label %PropagateExceptionOnNull_pass

call_trace:                                           ; preds = %continue_entry
  %f_last1 = getelementptr %struct._frame* %frame, i32 0, i32 15
  store i32, i32* %f_last1
  %f_bailed_from_ffvm11 = getelementptr %struct._frame* %frame, i32 0, i32 20
  store i8 0, i8* %f_bailed_from_ffvm11
  br label %bail_to_interpreter

PropagateExceptionOnNull_propagate:                   ; preds = %_PyLlvm_WrapDecref.exit18
  %69 = call i32 @PyTraceBack_Here(%struct._frame* %frame)
  %v_traceline = getelementptr %struct._ts* %0, i32 0, i32 3
  %70 = call i32* @_PyTrace_Line(%struct._ts* %0, %struct._frame* %frame, %struct._object*** %v_traceline)
  %71 = icmp eq i32 (%struct._object*, %struct._frame*, i32, %struct._object*)** %70, null
  br i1 %71, label %unwind_loop_header.preheader, label %call_exc_trace
PropagateExceptionOnNull_pass:                        ; preds = %_PyLlvm_WrapDecref.exit18
  store %struct._object** %tmrcp_result, %struct._object*** %stack_bottom
  br label %unwind_loop_header.preheader
}
```

Why is Python slow?

```
>>> x = dict()
>>> len(x)
```

Why is Python slow?

```
>>> x = dict()
>>> len(x)
```

...len() calls PyObject_Size()

Why is Python slow?

```
>>> x = dict()
>>> len(x)
```

...len() calls PyObject_Size()
...which looks up x->ob_type->tp_as_sequence->sq_length

Why is Python slow?

```
>>> x = dict()
>>> len(x)
```

...len() calls PyObject_Size()
...which looks up x->ob_type->tp_as_sequence->sq_length
...which is NULL, so call PyMapping_Size()

Why is Python slow?

```
>>> x = dict()
>>> len(x)
```

...len() calls PyObject_Size()
...which looks up x->ob_type->tp_as_sequence->sq_length
...which is NULL, so call PyMapping_Size()
...which looks up x->ob_type->tp_as_mapping->mp_length

Why is Python slow?

```
>>> x = dict()
>>> len(x)
```

...len() calls PyObject_Size()
...which looks up x->ob_type->tp_as_sequence->sq_length
...which is NULL, so call PyMapping_Size()
...which looks up x->ob_type->tp_as_mapping->mp_length
...which is a function pointer to dict_length()

Why is Python slow?

```
>>> x = dict()
>>> len(x)
```

...len() calls PyObject_Size()
...which looks up x->ob_type->tp_as_sequence->sq_length
...which is NULL, so call PyMapping_Size()
...which looks up x->ob_type->tp_as_mapping->mp_length
...which is a function pointer to dict_length()
...which returns x->ma_used

Why is Python slow?

Very late-binding
Can’t even make assumptions about builtin functions, types
Very hard to predict, have to measure

Why is Python slow?

```
def foo(x):
    yield len(x)
    yield len(x)

>>> g = foo(range(5))
>>> g.next()
5
>>> len = lambda y: 8
>>> g.next()
```


Very late-binding
Can’t even make assumptions about builtin functions, types
Very hard to predict, have to measure

Why is Python slow?

```
def foo(x):
    yield len(x)
    yield len(x)

>>> g = foo(range(5))
>>> g.next()
5
>>> len = lambda y: 8
>>> g.next()
8
```


Very late-binding
Can’t even make assumptions about builtin functions, types
Very hard to predict, have to measure

Unladen Swallow

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.
- Assume the program is less dynamic than the language.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.
- Assume the program is less dynamic than the language.
    - Globals/builtins aren't overridden once set.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.
- Assume the program is less dynamic than the language.
    - Globals/builtins aren't overridden once set.
    - Classes never change.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.

- Assume the program is less dynamic than the language.
    - Globals/builtins aren't overridden once set.
    - Classes never change.
    - Objects never change type.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.

- Assume the program is less dynamic than the language.
    - Globals/builtins aren't overridden once set.
    - Classes never change.
    - Objects never change type.
    - You never debug/profile your code.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.

- Assume the program is less dynamic than the language.
    - Globals/builtins aren't overridden once set.
    - Classes never change.
    - Objects never change type.
    - You never debug/profile your code.
    - Locals aren't deleted.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.

- Assume the program is less dynamic than the language.
    - Globals/builtins aren't overridden once set.
    - Classes never change.
    - Objects never change type.
    - You never debug/profile your code.
    - Locals aren't deleted.
    - sys._getframe() doesn’t matter.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow

- Compile to machine code.

- Assume the program is less dynamic than the language.
  - Globals/builtins aren't overridden once set.
  - Classes never change.
  - Objects never change type.
  - You never debug/profile your code.
  - Locals aren't deleted.
  - sys._getframe() doesn’t matter.
  - exec is never used.

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

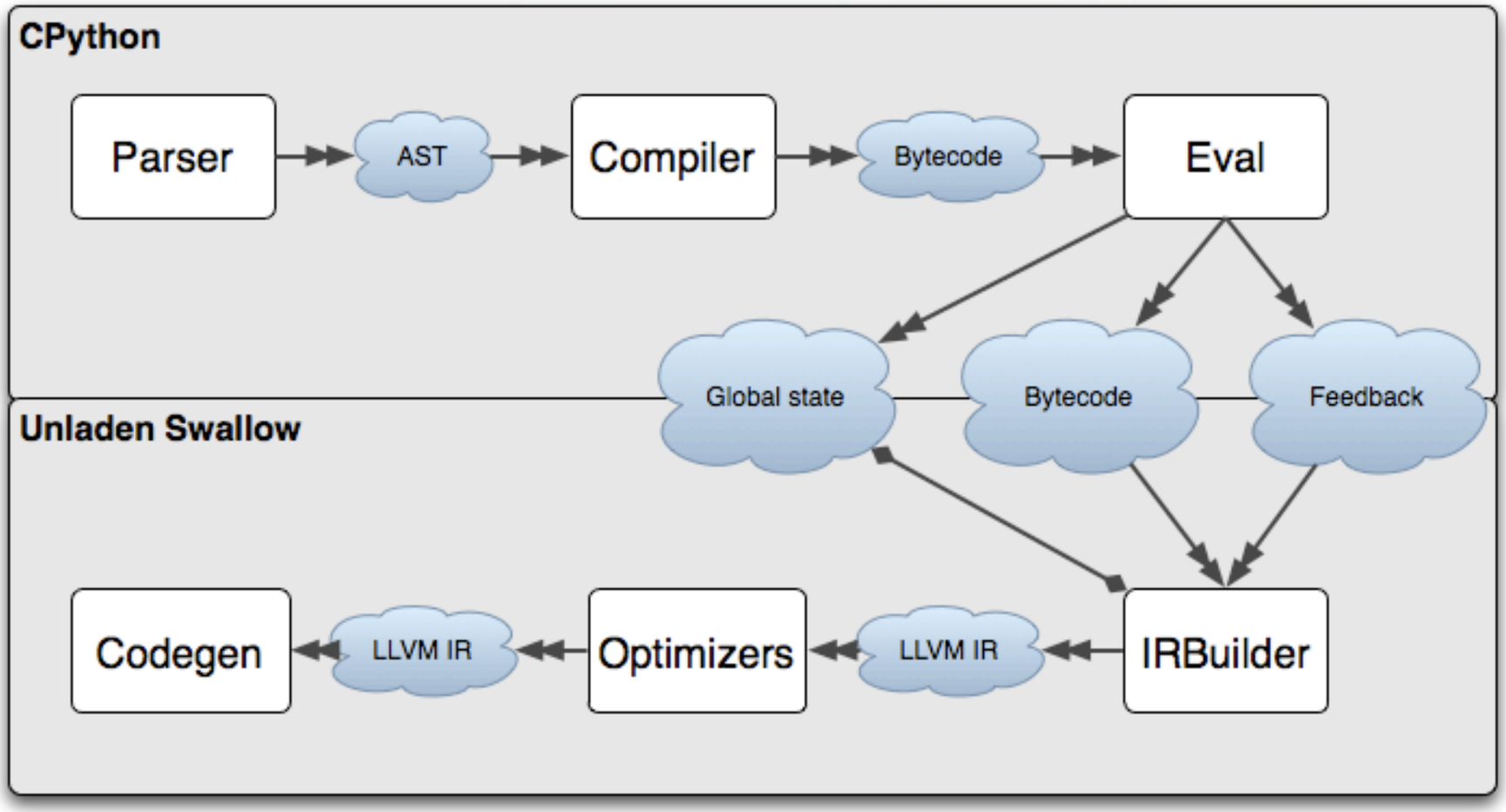
Unladen Swallow

- Compile to machine code.

- Assume the program is less dynamic than the language.
  - Globals/builtins aren't overridden once set.
  - Classes never change.
  - Objects never change type.
  - You never debug/profile your code.
  - Locals aren't deleted.
  - sys._getframe() doesn’t matter.
  - exec is never used.
  - ...

Prior work:
- Rubinius (Ruby)
- V8 (Chrome), TraceMonkey (Firefox), SquirrelFish Extreme (Safari)
- Pysco, PyPy
- HotSpot JVM, StrongTalk
- Self 93
- SmallTalk

Unladen Swallow



- Bytecode compiler based on IRBuilder
- Clang used to compile parts of the runtime to IR for inlining

Make it faster: specialize dynamically

```
def escape(value):
  from django.utils.safestring import mark_for_escaping
  return mark_for_escaping(value)
```

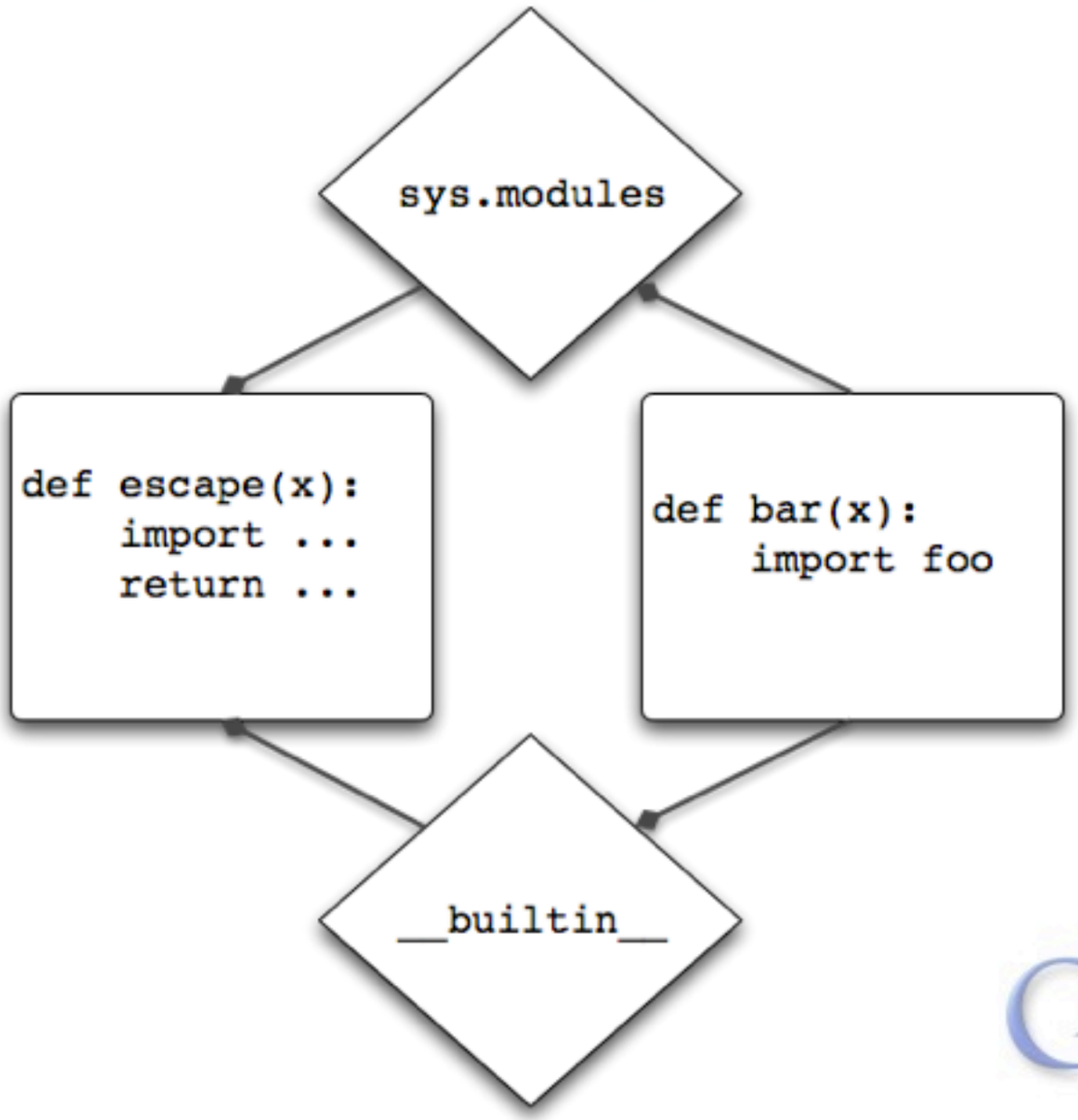
```
  0 LOAD_CONST               1 (-1)
  3 LOAD_CONST               2 (('mark_for_escaping',))
  6 IMPORT_NAME              0 (django.utils.safestring)
  9 IMPORT_FROM              1 (mark_for_escaping)
 12 STORE_FAST               1 (mark_for_escaping)
 15 POP_TOP
 16 LOAD_FAST                1 (mark_for_escaping)
 19 LOAD_FAST                0 (value)
 22 CALL_FUNCTION            1
 25 RETURN_VALUE
```

Make it faster: specialize dynamically

```
6 IMPORT_NAME                0 (django.utils.safestring)
```

```
if (world_has_not_changed) {
    x = (PyObject *)11712432
}
else {
    goto bail_to_interpreter
}
```

Make it faster: `if (world_has_not_changed) {`



Make it faster: specialize dynamically

```
def template(data, rows):
    for row in rows:
        data.append("<tr>")
        for col in row:
            data.append("<td>%s</td>" % col)
        data.append("</tr>")
```

```
>>> our_data = []
>>> template(our_data, our_rows)
>>> print "".join(our_data)
```


Highlights:
- data.append() lookups can be optimized to direct calls
- guard on type version
- "%s" % operator can be specialized

Zoom keys:
Apple option plus

Make it faster: call sites

```
data.append("<td>%s</td>" % col)
```

```
LOAD_FAST          0 (data)
LOAD_ATTR          0 (append)
LOAD_CONST         1 ('<td>%s</td>')
LOAD_FAST          1 (col)
BINARY_MODULO
CALL_FUNCTION      1
```


Highlights:
- data.append() lookups can be optimized to direct calls
- guard on type version
- "%s" % operator can be specialized

Zoom keys:
Apple option plus

Make it faster: call sites

```
LOAD_FAST          0 (data)
LOAD_ATTR          0 (append)
...
CALL_FUNCTION      1
```

```
data = locals[0];
if (Py_TYPE_VERSION(data) != EXPECTED_TYPE)
    goto bail_to_interpreter;
// LOAD_CONST
// LOAD_FAST
// BINARY_MODULO
retval = list_append(data, modulo_result);
```

Highlights:
- data.append() lookups can be optimized to direct calls
- guard on type version
- “%s” % operator can be specialized

Zoom keys:
Apple option plus

Top-down Inlining Opportunities

```
>>> x = dict()
>>> len(x)
```

...len() calls PyObject_Size()
...which looks up x->ob_type->tp_as_sequence->sq_length
...which is NULL, so call PyMapping_Size()
...which looks up x->ob_type->tp_as_mapping->mp_length
...which is a function pointer to dict_length()
...which returns x->ma_used

Python Objects: constant(ish)

```
struct PyDictObject {
    Py_ssize_t ob_refcnt;
    PyTypeObject *ob_type;
    Py_ssize_t ma_fill;
    Py_ssize_t ma_used;
    ...
};

PyTypeObject PyDict_Type = {
    ...
    &dict_as_sequence,
    &dict_as_mapping,
    ...
};
```


- Mirror locals[2] to LLVM IR by walking a pointer tree
- Use a custom AA pass to mark most of the object as constant.
- Constant propagation will remove the second test.

Top-down Inlining: len()

```
static PyObject *
builtin_len(PyObject *self, PyObject *o)
{
    Py_ssize_t res;
    {
        PySequenceMethods *m;
        if (o == NULL) {
            // Error
        }
        m = o->ob_type->tp_as_sequence;
        if (m && m->sq_length)
            res = m->sq_length(o);
        else {
            PyMappingMethods *m;
            if (o == NULL) {
                // Error
            } else {
                m = o->ob_type->tp_as_mapping;
                if (m && m->mp_length)
                    res = m->mp_length(o);
                else
                    // Error
            }
        }
    }
    if (res < 0 && PyErr_Occurred())
        return NULL;
    return PyInt_FromSsize_t(res);
}
```


- Mirror locals[2] to LLVM IR by walking a pointer tree
- Use a custom AA pass to mark most of the object as constant.
- Constant propagation will remove the second test.

Top-down Inlining: result

```
static PyObject *
builtin_len(PyObject *self, PyObject *v)
{
  return PyInt_FromSsize_t(((PyDictObject *)v)->ma_used);
}
```


- Mirror locals[2] to LLVM IR by walking a pointer tree
- Use a custom AA pass to mark most of the object as constant.
- Constant propagation will remove the second test.

Measurement & Testing

- Benchmarks:
  - 32 benchmarks (and counting)!
  - YouTube hotspots: Spitfire templates, pickling.
  - Libraries: pickling, regexes, html5lib.
  - Apps: 2to3, Django, Rietveld, SpamBayes, etc.
  - Microbenchmarks: GC, IO, string operations.
  - Now in use by PyPy, soon by Jython.

- Correctness:
  - SWIGed code, extensions used by YouTube.
  - Google's large Python codebase.
  - Large Python projects: Django, NumPy, Twisted, etc.
  - Randomized testing.

Performance vs CPython 2.6.4

Benchmark	CPython	Unladen	Change
2to3	25.13 s	24.87 s	1.01x faster
django	1.08 s	0.68 s	1.59x faster
html5lib	14.29 s	13.20 s	1.08x faster
nbody	0.51 s	0.28 s	1.84x faster
rietveld	0.75 s	0.55 s	1.37x faster
slowpickle	0.75 s	0.55 s	1.37x faster
slowspitfire	0.83 s	0.61 s	1.36x faster
slowunpickle	0.33 s	0.26 s	1.26x faster
spambayes	0.31 s	0.34 s	1.10x slower

Performance vs Jython 2.5.1

Benchmark	Jython	Unladen	Change
ai	1.58	0.52	3.07x faster
django	2.0	0.8	2.48x faster
float	0.33	0.07	3.49x faster
rietveld	2.45	0.59	4.17x faster
slowpickle	0.86	0.56	1.54x faster
slowspitfire	2.76	0.63	4.39x faster
slowunpickle	0.38	0.27	1.37x faster

Performance vs PyPy

Benchmark	PyPy	Unladen	Change
ai	0.61	0.51	1.1921x faster
django	0.67	0.68	1.0136x slower
float	0.03	0.07	2.7108x slower
html5lib	20.04	16.42	1.2201x faster
pickle	17.70	1.09	16.2465x faster
rietveld	1.09	0.59	1.8597x faster
slowpickle	0.43	0.56	1.2956x slower
slowspitfire	2.50	0.63	3.9853x faster
slowunpickle	0.26	0.27	1.0585x slower
unpickle	28.45	0.78	36.6427x faster

Memory Usage vs CPython 2.6.4

Benchmark	CPython	Unladen	Change
2to3	26396 kb	46896 kb	1.77x
django	10028 kb	27740 kb	2.76x
html5lib	150028 kb	173924 kb	1.15x
nbody	3020 kb	16036 kb	5.31x
rietveld	15008 kb	46400 kb	3.09x
slowpickle	4608 kb	16656 kb	3.61x
slowspitfire	85776 kb	97620 kb	1.13x
slowunpickle	3448 kb	13744 kb	3.98x
spambayes	7352 kb	46480 kb	6.32x

Looking Back

Google™

Sunday, February 21, 2010

- Q1 release
- Tool detours: gdb, oProfile
- LLVM problems

- What we did right: code reuse,

GIL?

Google™

Merger

Google™

Sunday, February 21, 2010

- PEP is out for review
- Review going well
- Good feedback from community:
    - Prioritize -j never performance
    - Shared linking for LLVM
    - No C++ at all under --without-llvm build

Fin

Questions?

http://code.google.com/p/unladen-swallow/

collinwinter@google.com
unladen-swallow@googlegroups.com
#unladenswallow on OFTC