MYPY: A PYTHON VARIANT WITH SEAMLESS DYNAMIC AND STATIC TYPING

Jukka Lehtosalo
University of Cambridge Computer Laboratory

SPEAKER BIO

2000-2006 Software engineer (QPR Software and Kielikone)

2007-2009 Development manager (Kielikone)

2009→ PhD research (University of Cambridge)

Projects spanning web apps, Windows apps, mobile apps,distributed apps, document databases, information retrieval, pattern recognition, natural language processing, data mining, garbage collectors, compilers, localization tools, automated testing tools etc.

mypy is an experimental
Python variant
with seamless
dynamic and static typing

Mostly, but not 100%
compatible with
Python

mypy is an experimental
Python variant
with seamless
dynamic and static typing

Research project,
in development

mypy is an experimental
Python variant
with seamless
dynamic and static typing

mypy is an experimental
Python variant
with seamless
dynamic and static typing

Freely mix static and dynamic (duck) types
Combine the benefits of static and dynamic typing

LANGUAGE GAP

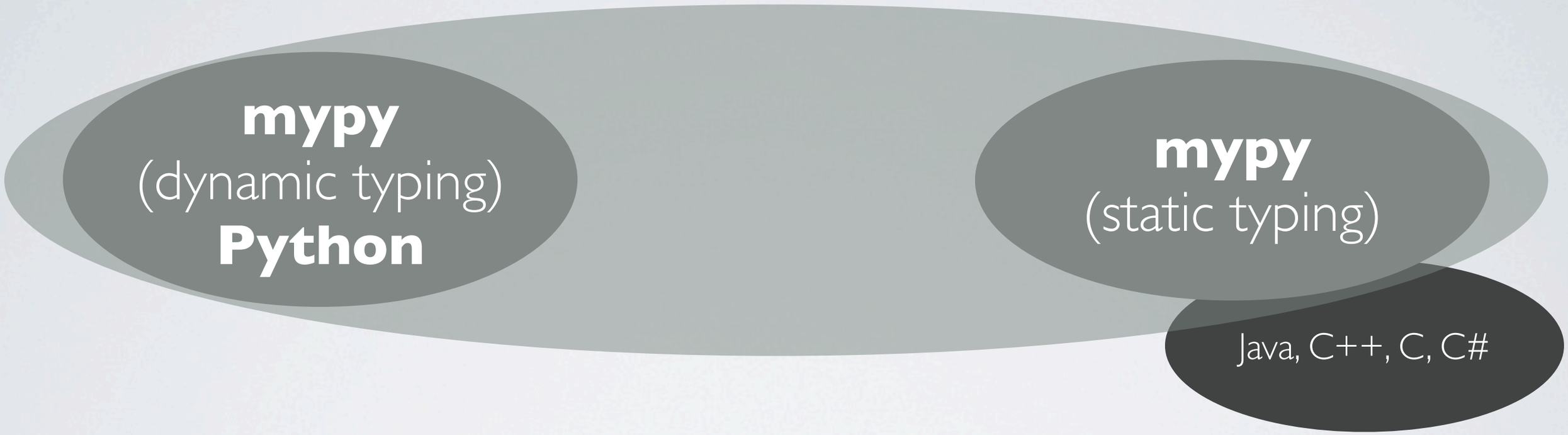
Python

Scripting, web development, UI, prototyping

Java, C++, C, C#

Heavy lifting

VISION



Scripting, web development, UI, prototyping

Heavy lifting

WHAT DOES IT LOOK LIKE?

mypy, dynamic typing

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b
```

mypy, static typing

```
void fib(int n):
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b
```

WHAT DOES IT LOOK LIKE?

mypy, dynamic typing

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b
```

Dynamically typed function

mypy, static typing

```
void fib(int n):
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b
```

WHAT DOES IT LOOK LIKE?

mypy, dynamic typing

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b
```

mypy, static typing

```
void fib(int n):
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b
```

Static type signature

WHAT DOES IT LOOK LIKE?

mypy, dynamic typing

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b
```

mypy, static typing

```
void fib(int n):
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b
```

Type inference

DYNAMIC VS STATIC TYPING

mypy, dynamic typing

```
def err():
    return 1 + 'x'   # no error (not called)
```

DYNAMIC VS STATIC TYPING

mypy, dynamic typing

```
def err():
    return 1 + 'x'   # runtime error (when called)
err()
```

DYNAMIC VS STATIC TYPING

mypy, dynamic typing

```
def err():
    return 1 + 'x'  # runtime error (when called)
err()
```

mypy, static typing

```
int err():
    return 1 + 'x'  # compile (static) error
```

TYPES

int, str, bool, MyClass

simple type

list<int>, dict<str, int>

generic type

tuple<int, str>

tuple type

any

dynamic (duck) type

func<str, int>

function type

EXAMPLE SCRIPT

```
import sys, re

if not sys.argv[1:]:
    raise RuntimeError('Usage: wordfreq FILE')

dict<str, int> d = {}

s = open(sys.argv[1]).read()
for word in re.sub('\W', ' ', s).split():
    d[word] = d.get(word, 0) + 1

# Use list comprehension
l = [(freq, word) for word, freq in d.items()]

for freq, word in sorted(l):
    print('%-6d %s' % (freq, word))
```

EXAMPLE SCRIPT

```
import sys, re

if not sys.argv[1:]:
    raise RuntimeError('Usage: wordfreq FILE')

dict<str, int> d = {}

s = open(sys.argv[1]).read()
for word in re.sub('\W', ' ', s).split():
    d[word] = d.get(word, 0) + 1

# Use list comprehension
l = [(freq, word) for word, freq in d.items()]

for freq, word in sorted(l):
    print('%-6d %s' % (freq, word))
```

EXAMPLE CLASS

dynamic typing

```
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def overdrawn(self):
        return self.balance < 0

my_account = BankAccount(15)
my_account.withdraw(5)
print(my_account.balance)
```

(source: Python Wiki)

EXAMPLE CLASS

static typing

```
class BankAccount:
    void __init__(self, int initial_balance=0):
        self.balance = initial_balance
    void deposit(self, int amount):
        self.balance += amount
    void withdraw(self, int amount):
        self.balance -= amount
    bool overdrawn(self):
        return self.balance < 0

my_account = BankAccount(15)
my_account.withdraw(5)
print(my_account.balance)
```

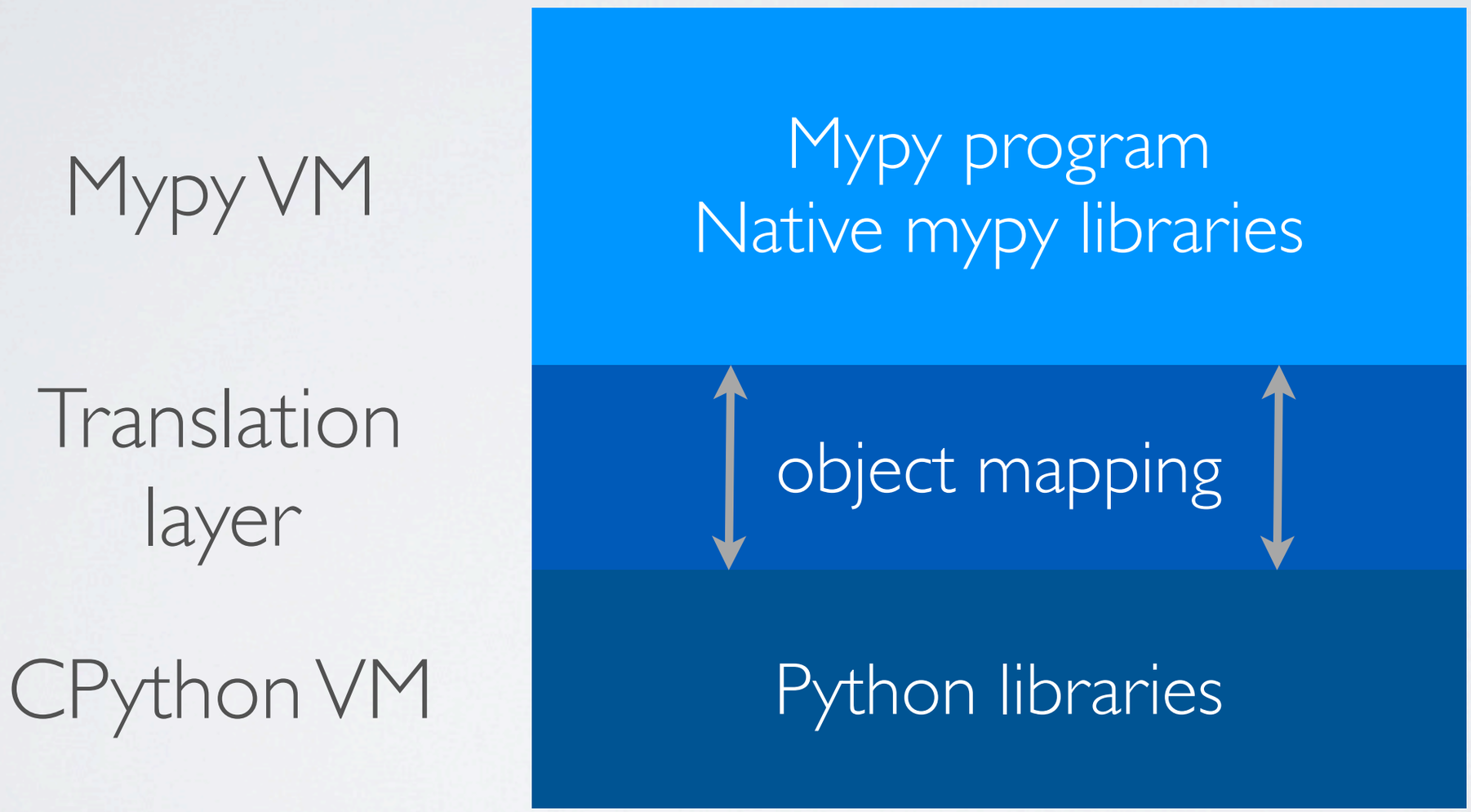
EXAMPLE CLASS

static typing

```
class BankAccount:
    void __init__(self, int initial_balance=0):
        self.balance = initial_balance
    void deposit(self, int amount):
        self.balance += amount
    void withdraw(self, int amount):
        self.balance -= amount
    bool overdrawn(self):
        return self.balance < 0

my_account = BankAccount(15)
my_account.withdraw(5)
print(my_account.balance)
```

MYPY LIBRARY SUPPORT



NO GIL

CPython uses Global Interpreter Lock
⇒ only one thread can usually run at a time

NO GIL

CPython uses Global Interpreter Lock
⇒ only one thread can usually run at a time

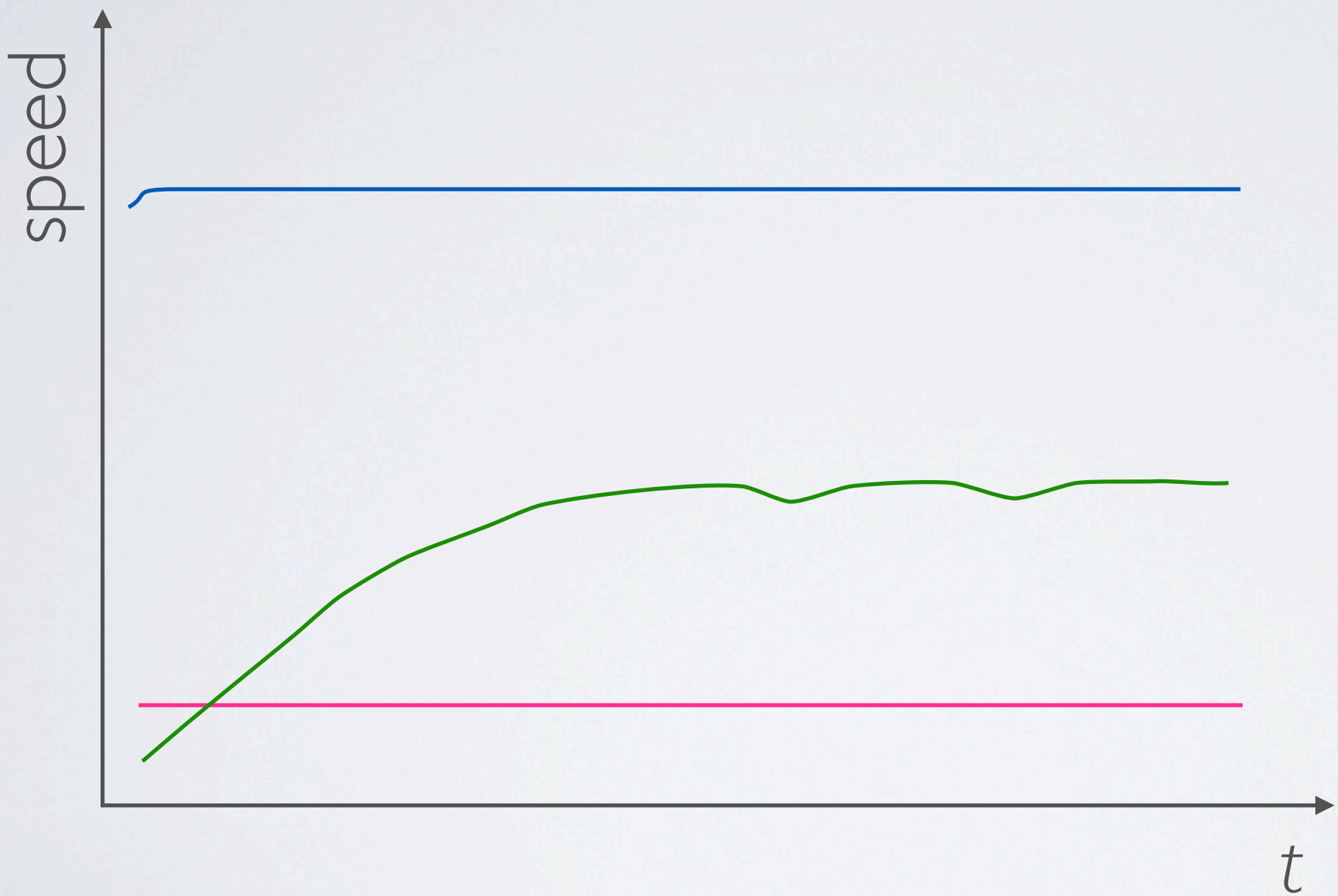
Not ok in the multicore era!

Mypy will have a new†, custom VM without GIL.

† Based on the Alore VM (http://www.alorelang.org/)

WHY STATIC TYPING?

PERFORMANCE



mypy
static typing, ahead-of-time compiler

PyPy
JIT compiler

CPython
interpreter

(conceptual, not in scale)

TYPE CHECKING

Static type checker pinpoints errors in your program, before running it

```
some_function(foo[n], self.bar(x))
                                 ^
StaticError: Argument has invalid type list<str>,
  list<int> expected
```

- Less debugging
- Easier maintenance and refactoring

TOOL SUPPORT

Static type information makes many tools more powerful:

- IDEs with **reliable** and **precise** code completion
- Refactoring tools
- Static analysis and linting tools

➠ Better productivity and quality

SEAMLESS DYNAMIC AND STATIC TYPING

Use the best tool for the job

Use dynamic typing when it makes sense
Use static typing when it makes sense

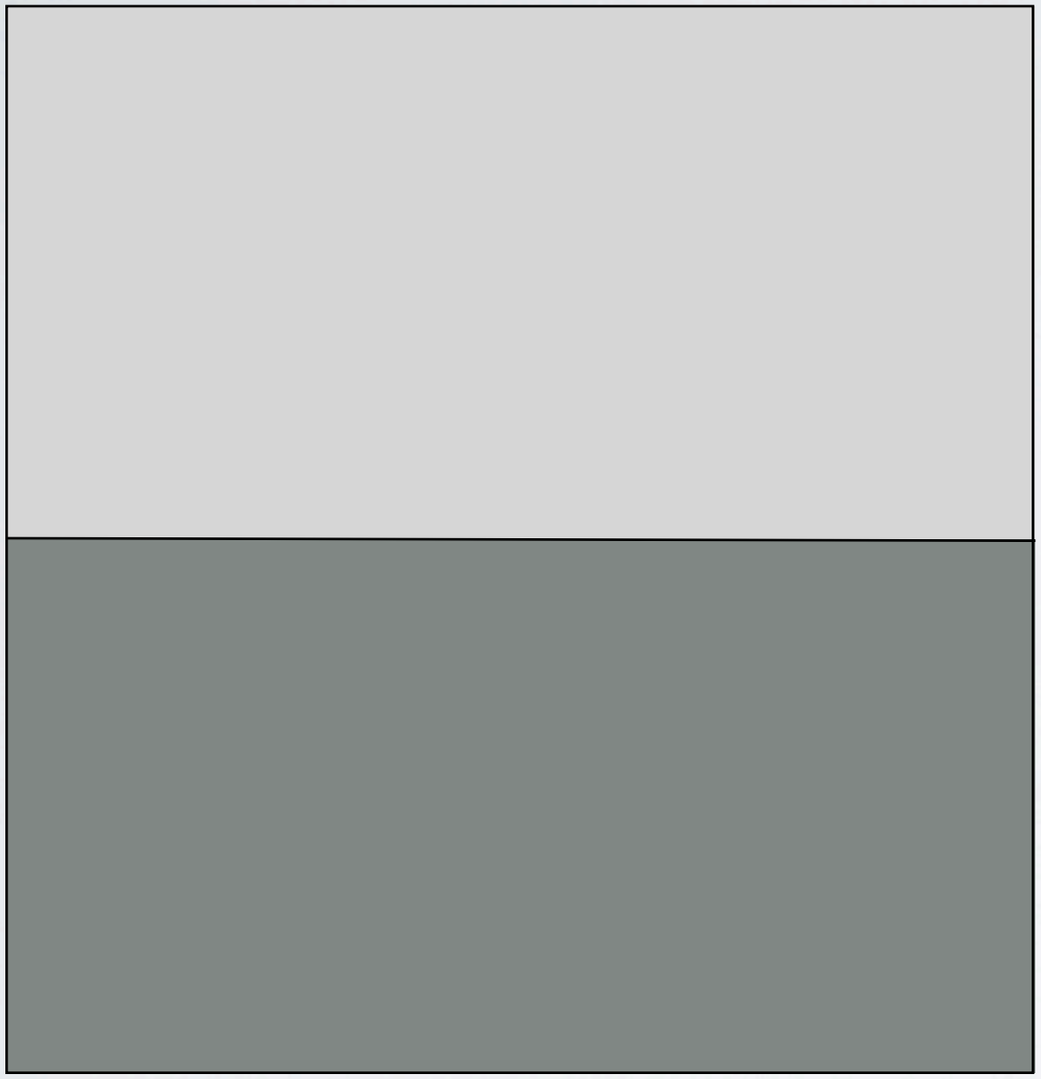
As real programs are not uniform: *use both*

Static typing may be a win

- if efficiency is important
- if your project is large or complex
- if you need to maintain code for several years

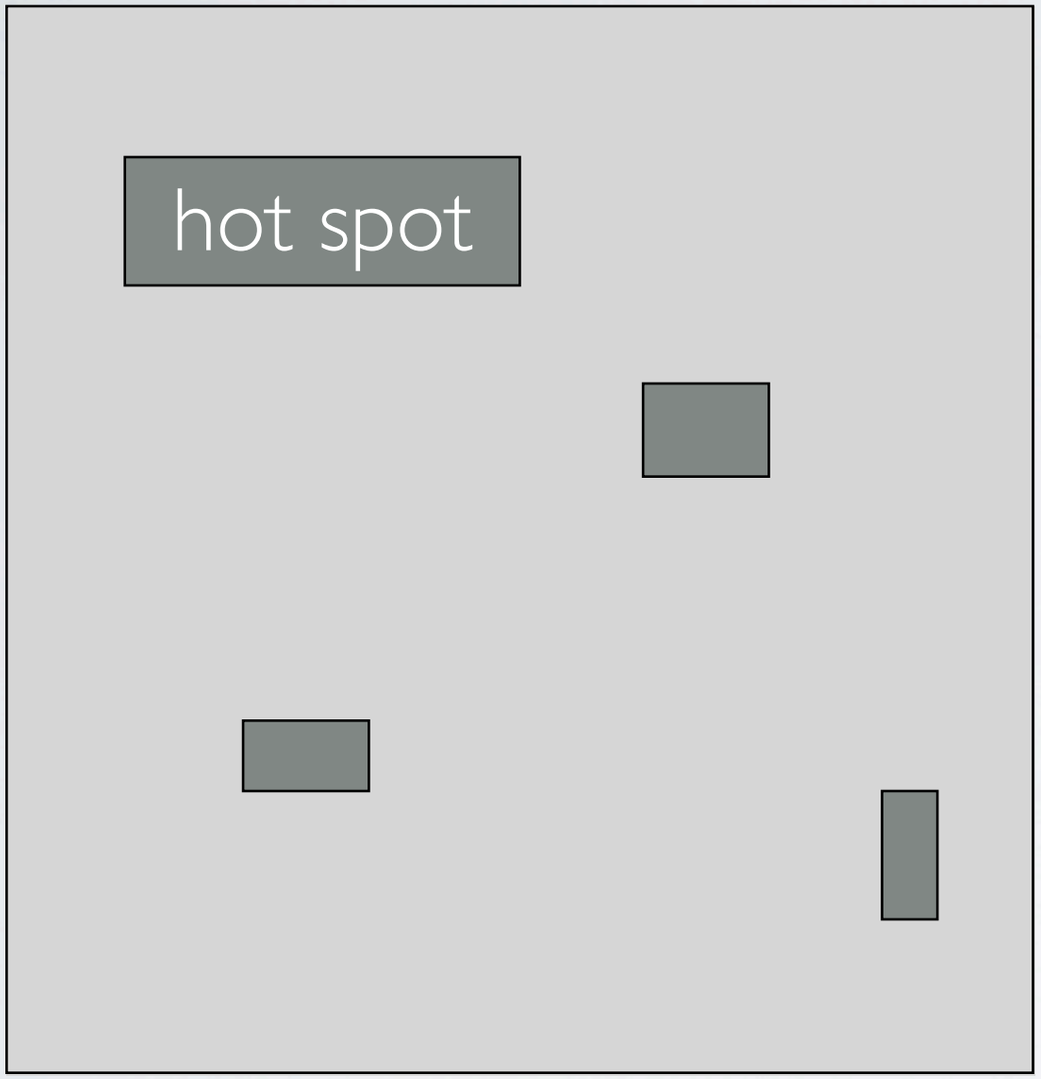
Even when the above are true, dynamic typing is useful

- for tests
- for the user interface
- for glue code
- for extensions, scripts and plugins

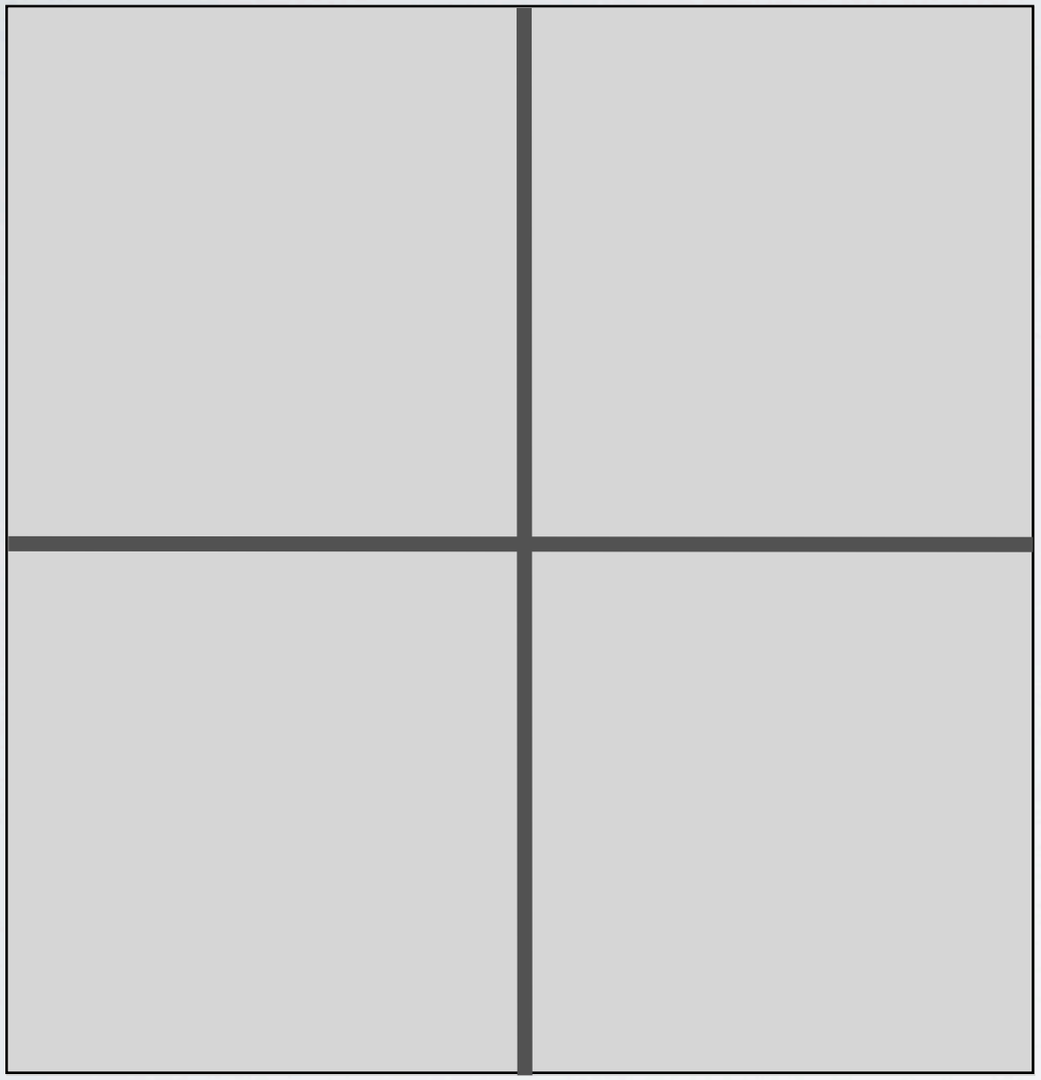


Dynamically typed program

Statically typed libraries



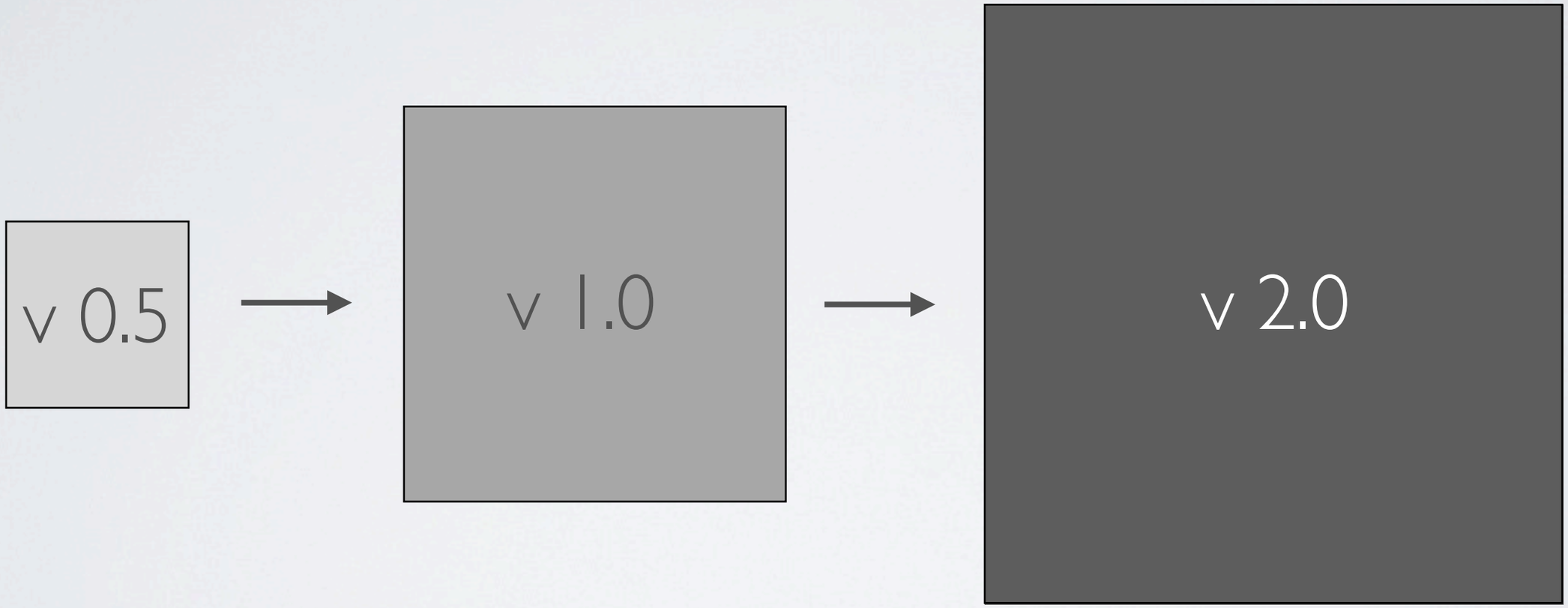
Sprinkle static types
for better performance



Statically typed interfaces

Dynamically typed implementations

Evolve programs from dynamic to static typing



IMPLEMENTATION PLAN

All dates tentative!

Phase 1 (2012)
Mypy type checker + mypy-to-Python translator
Phase 2 (2013)
Compiler to native code (C/LLVM back end), new VM
Phase N (?)
More Python features
IDE
JVM back end?
Android port? (etc.)

MYPY VS OTHER PROJECTS

CYTHON

- Cython is a Python variant that compiles to Python C modules
- Cython can speed up tight loops by a lot (100 x or more) by using type declarations
- Cython programs can also call C functions directly
- Similar to Pyrex

```
cpdef int sum3d(int[:, :, :] a):
    cdef int total = 0
    I = a.shape[0]
    J = a.shape[1]
    K = a.shape[2]
    for i in range(I):
        for j in range(J):
            for k in range(K):
                total += a[i, j, k]
    return total
```

(adapted from Cython Users Guide)

MYPY VS CYTHON

Mypy (planned)	Cython
Powerful type system	Simple type system
General code speedup	Speedups mainly to low-level code
New, efficient VM	CPython VM
No GIL	GIL still a problem

SHED SKIN

- Compiler for a statically typed subset of Python
- Large speedups to some programs (100x or more)
- Code looks like Python (explicit type declarations not needed)
  - Whole-program type inference

MYPY VS SHED SKIN

Mypy (planned)	Shed Skin
Large set of Python features	Restricted Python feature set
Dynamic and static typing	Static typing only
Modular, (relatively) fast compiles	Very slow compilation for large programs
Python lib access layer	Not many Python libs

MYPY VS RPYTHON

- RPython is a Python subset used by PyPy to implement the JIT compiler
- RPython uses whole-program type inference (like Shed Skin)
- Differences vs mypy: similar to mypy vs Shed Skin

CONCLUSION

Tell me what you think!

All help and feedback is very welcome!

Web site:

http://www.mypy-lang.org

Thank you!

ANDROID: OVERVIEW

- Over 500 M Android devices activated
- Android native APIs mostly based on Java (Dalvik)
- Performance important for apps
  - Cheap devices with low-end CPUs
  - UI responsiveness key to good user experience
  - Battery conservation

MYPY ON ANDROID?

Android Wishlist

1. Python syntax + libs
2. Good performance
3. Easy native API access

Can't have all of these right now

- Mypy is a potential solution (by compiling to Dalvik)
- But there's a lot of work to do...