Étoilé
Pragmatic Smalltalk

David Chisnall

August 25, 2011

Smalltalk is Awesome!

- Pure object-oriented system
- Clean, simple syntax
- Automatic persistence and many other great features

...but no one cares

- TIOBE ranks Smalltalk somewhere in the #51-100 range.
- In the same range as Io, Dylan, Eiffel, Inform...
- Behind Haskell (#34), Go (#24), Logo (#21)
- A long way behind Smalltalk-derivatives Ruby (#12), Objective-C (#6), and Java (#1)

Why?

- What do other languages have that Smalltalk doesn't?
- Large commercial backer, writing huge amounts of library code (e.g. Java)
- Easy interoperability with other languages

Language Report Cards

Smalltalk:

> *Very bright student. Excels in most subjects. Doesn't play well with others.*

Python:

> *Very friendly student. Always actively participates in group activities. May have undiagnosed learning difficulties.*

Heresy

- Smalltalk is not always the best tool for the job
- Try writing a video CODEC in Smalltalk
- It's possible, but you're fighting the language every step of the way

Smalltalk is a High-level Language

- High-level abstractions
- Good for most tasks
- Bad when you need close access to the hardware (e.g. SIMD)

Objective-C: C in Smalltalk Objects

- Created by Brad Cox and Tom Love in 1986 to package C libraries in Smalltalk-like classes
- Smalltalk object model
- C code in methods, message passing between objects
- Rich set of standard class libraries

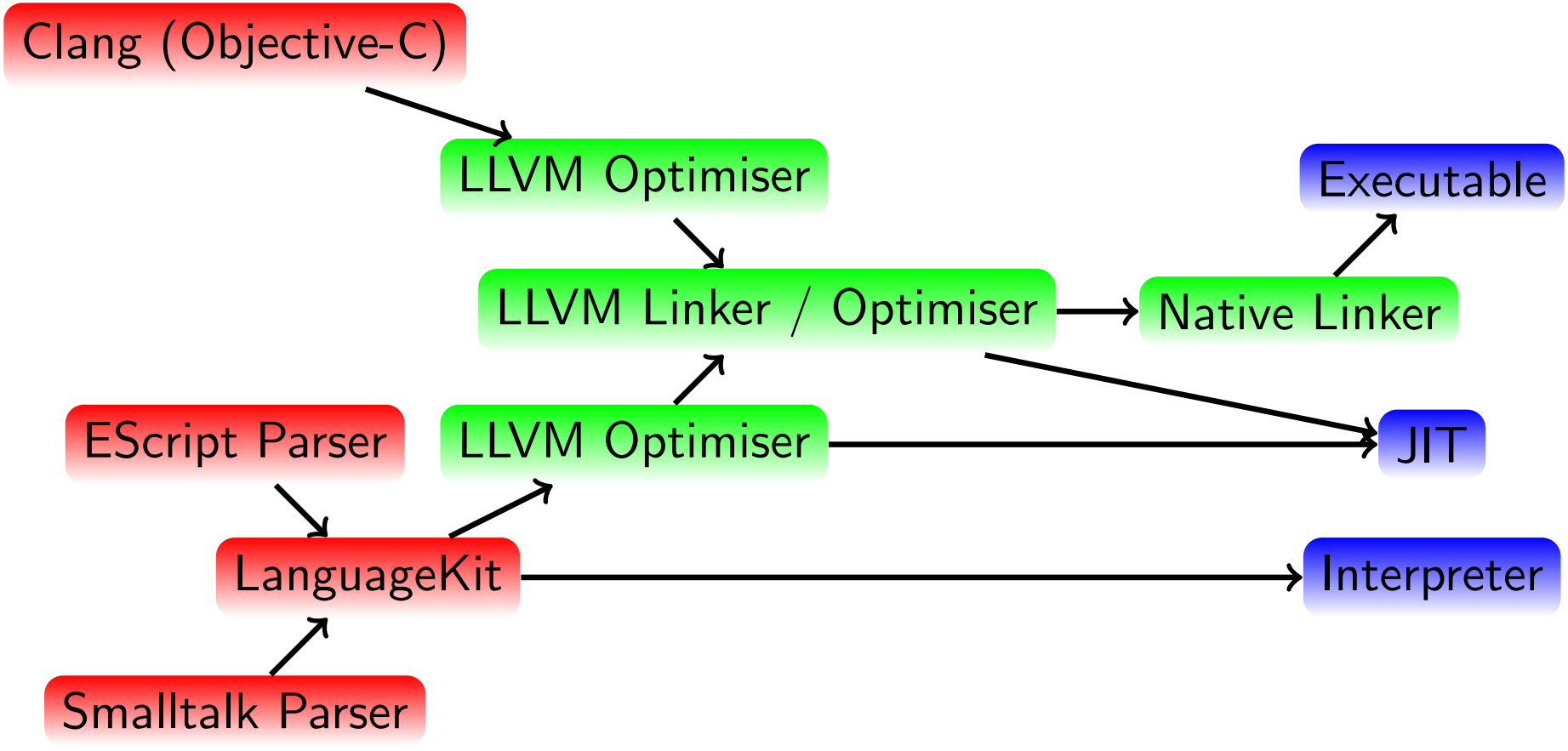
The Compiler and the Runtime

- The compiler generates calls to functions in the runtime library
- All Smalltalk-like features are implemented by runtime calls
- Calls to C libraries have the same cost as calling them from C
- Can incrementally deploy Objective-C code with C/C++ libraries

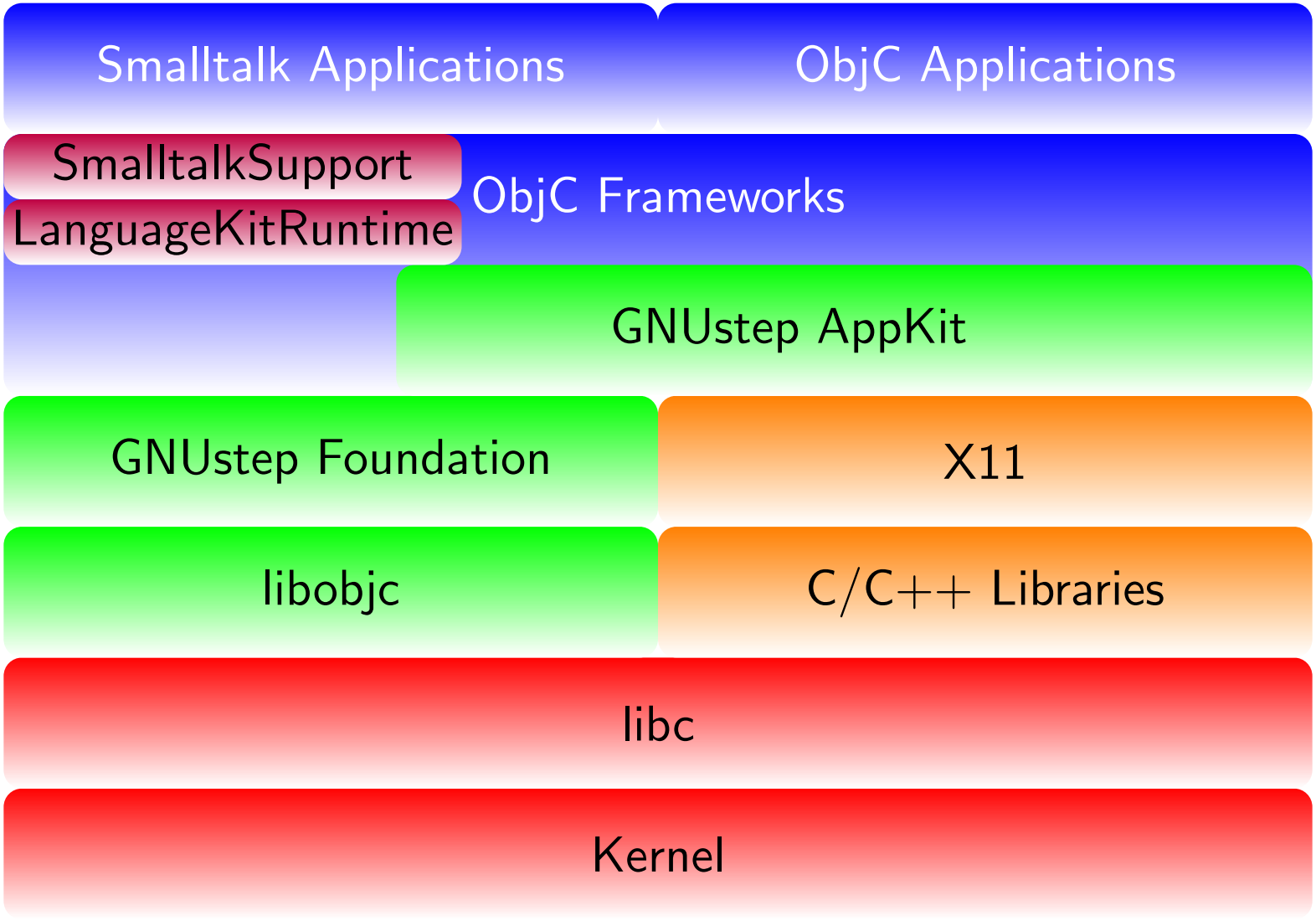
Pragmatic Smalltalk

- Dialect of Smalltalk used by Étoilé
- Implements Smalltalk-80 language, but not the standard libraries
- Compiles to native code (JIT or static compiler)
- Emits code ABI-compatible with Objective-C

Compiler Architecture



Execution Architecture



LanguageKit

- AST for representing Smalltalk-like languages
- Interpreter for directly executing the AST
- LLVM-based code generation back end for compiling
- Written in Objective-C

Compiling Smalltalk: The Hard Bits

- Small integers
- Blocks
- Non-local returns
- Memory management

Small Objects

- Objects hidden in pointers (e.g. small integers)
- Very common operations implemented as (hand optimised) C functions
- Inlined during compilation
- Objective-C runtime modified to allow sending messages to small objects
- Totally interoperable with Objective-C: small integers are just NSSmallInt instances, can be used anywhere NSNumber is expected
- Very fast: almost the same speed as C integer arithmetic - Fibonacci benchmark ran the same speed as GCC 4.2.1

Blocks

- Objective-C now supports blocks
- LanguageKit uses the same ABI
- Smalltalk and Objective-C blocks can be used interchangeably.

Non-Local Returns

- Returns from blocks
- Ugly feature, should never have been allowed in the language
- Implemented using same mechanism as exceptions (DWARF stack unwinding)

Memory Management

- Objective-C can use tracing GC or reference counting
- LanguageKit supports GC or automatic reference counting (ARC)
- Optimisation passes remove redundant retain / release operations in ARC mode.

Objective-C[++] is our Foreign Function Interface

- Objective-C and Smalltalk classes are the same
- Categories can be written in either language
- Methods written in Smalltalk and Objective-C are indistinguishable
- Calling C++ from Objective-C++ is trivial

Sending Messages to C

Writing a method just to call a C function is cumbersome (and slow!)

```
"Smalltalk code:"
C sqrt: 42.
C fdim: {60. 12}.
C NSLocation: l InRange: r.
```

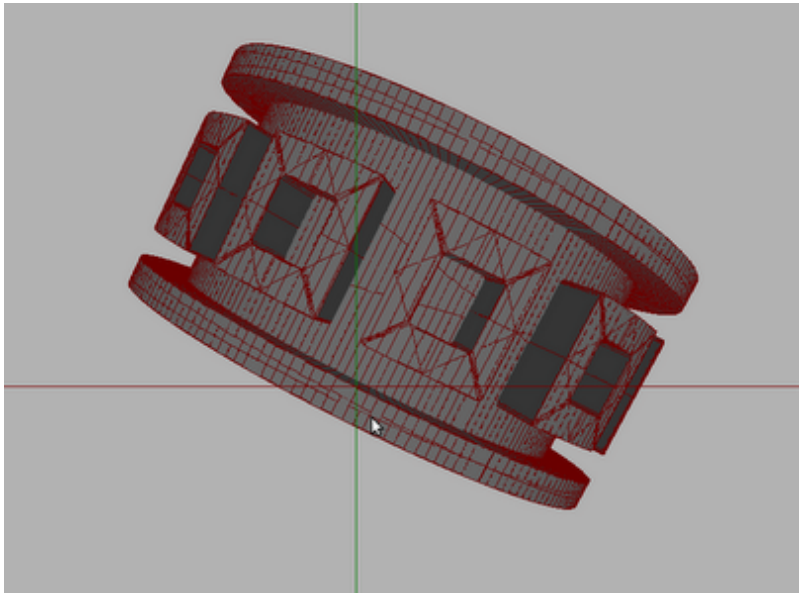
Generates exactly the same code as:

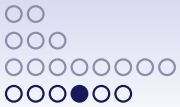
```
// C code
sqrt(42);
fdim(60, 12);
NSLocationInRange(l, r);
```

No bridging code, no custom wrappers, just native function calls in the compiled code.

Smalltalk in the Terminal

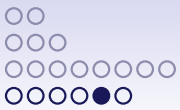
```
$./br "viewSpace: (Shapeways ringBlopperPath:
            (env objectForKey:'BLOPPER')
                                    order: 1
                                ringSize: 10
                               thickness: 3.0
                                  height: 15.0)"
```





In Shell Scripts

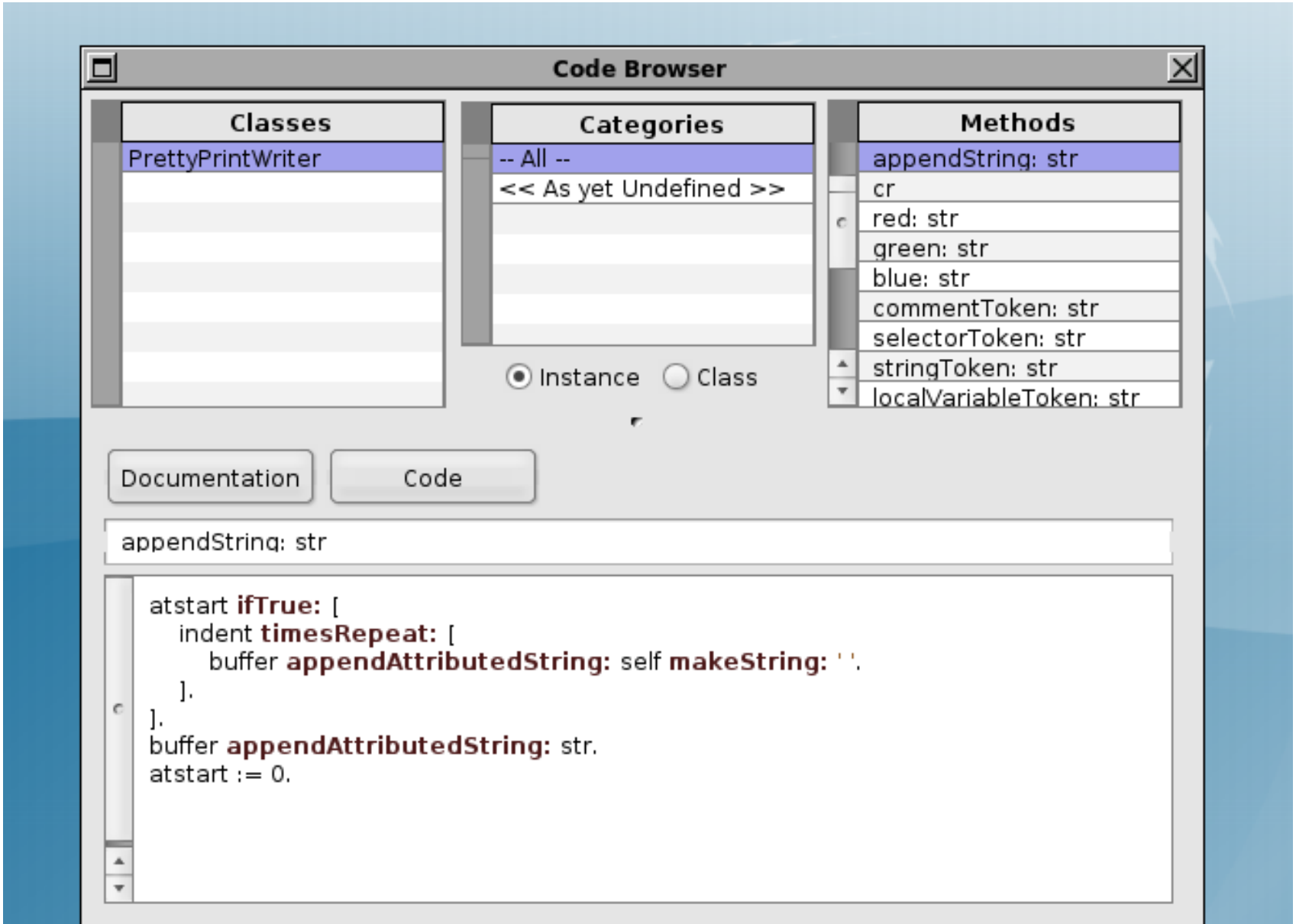
```
$ cat foo.st
#!/Local/Tools/edlc -f
NSObject subclass: SmalltalkTool [
  run [ | info |
    info := NSProcessInfo processInfo.
    ETTranscript show: 'Hello, world?' ; cr;
      show: 'sqrt(2): '; show: (C sqrt: 2) ; cr;
      show: 'Arguments: ' ; show: (info arguments); cr.
  ]
]
$ ./foo.st This is a shell script
Hello, world?
sqrt(2): 1.414214
Arguments: ("/Local/Tools/edlc", "-f", "./foo.st", This,
    is, a, shell, script)
```



Compiled Shell Scripts

```
$ edlc -c -f foo.st
$ llc foo.bc
$ clang -fobjc-nonfragile-abi run.m foo.s
 -lEtoileFoundation -lLanguageKitRuntime
$ ./a.out this is the same shell script
Hello, world?
sqrt(2): 1.414214
Arguments: ("./a.out", this, is, the, same, shell,
  script)
$ edlc -i -f foo.st In the interpreter
Hello, world?
sqrt(2): 1.414214
Arguments: (edlc, "-i", "-f", "foo.st", In, the,
  interpreter)
```

Or in the GUI...


Code Browser
Classes
PrettyPrintWriter
Categories
-- All --
<< As yet Undefined >>
Methods
appendString: str
cr
red: str
green: str
blue: str
commentToken: str
selectorToken: str
stringToken: str
localVariableToken: str
Instance
Class
Documentation
Code
appendString: str
atstart ifTrue: [
indent timesRepeat: [
buffer appendAttributedString: self makeString: ' '.
].
].
buffer appendAttributedString: str.
atstart := 0.

What Makes Things Slow?

- Small integer arithmetic
- Boxing
- Dynamic message lookup
- Memory management operations

Small Objects Support in Libobjc

- Allows Smalltalk SmallInts to be returned to ObjC code (no boxing required)
- Removes the need for LanguageKit to add conditionals around every message send
- Approximately 40% reduction in code size
- Smaller code means better instruction cache usage

Lookup Caching

- New lookup function returns a pointer to a structure
- Structure contains a version
- Version incremented if the method is replaced
- Safe automatic caching now possible
- Optimisation pass caches all lookups in loops and to classes
- Optimisations shared between Objective-C and Smalltalk

Speculative Inlining

- C can insert copies of functions where they are used
- Objective-C message sends may map to different methods
- But we can guess one...
- ...inline it...
- ...and wrap it in a test to see if we guessed right

A Microbenchmark

- Simple message sends in a loop.
- Target: using C function calls takes 3 seconds
- Infinitely fast lookup function would therefore take 6 seconds

Optimisation	Time (s)
None	10
Standard LLVM Opts	8
Auto-Caching	4.6
Auto-Caching + LLVM Opts	3.5
Auto-Caching + Speculative Inlining	2

Are we 'fast enough' yet?

Don't Trust Microbenchmarks

(Including the last slide!)

- Algorithmic improvements make much a more noticeable difference to performance
- Loose coupling makes specialisation easier.
- Case study: Integrating with libicu
- The problem: libicu uses its own (very efficient) unicode string representation

ICU Strings in C++

- `std::string` is the standard string class
- Non-virtual methods for speed
- Not flexible enough: everyone implements their own (e.g. `llvm::StringRef`, `qt::QString`, etc.)
- Using libicu strings involves $O(n)$ operations, copying characters between representations
- This problem happens at every library boundary

ICU Strings in Objective-C

- `NSString` is the standard string class
- Abstract superclass, (hidden) concrete subclasses
- Very flexible, used everywhere.
- Using libicu strings involves an $O(1)$ operation, wrapping the libicu string type in an `NSString` subclass.
- Objective-C wrapper can be used in Pragmatic Smalltalk directly

Result: The 'slower' language encourages $O(1)$ algorithms where the 'faster' language encourages $O(n)$ algorithms.

Object Planes

- GNUstep runtime takes the sender as argument to lookup function
- Object Planes are now possible with the Objective-C runtime
- Messages intercepted when travelling between groups of objects
- Implicit concurrency, access control, automatic serialisation...

Building on Pragmatic Smalltalk

- Etoile aims to build a modern desktop environment
- Lots of frameworks
- User-visible code coming Real Soon Now

EtoileFoundation

- Higher order messaging
- Traits
- Futures
- Prototypes

EtoileUI

- High-level UI abstraction
- Introspective UI

CoreObject

- Automatic persistence and versioning
- Stores objects, not files
- Handles diff and merge on object graphs

EtoileBehavior

- Bundle, automatically loaded by all GNUstep applications
- Loads LanguageKit bundles - Smalltalk code injected into all running apps

Questions?

Gratuitous book plug!