Parrot VM

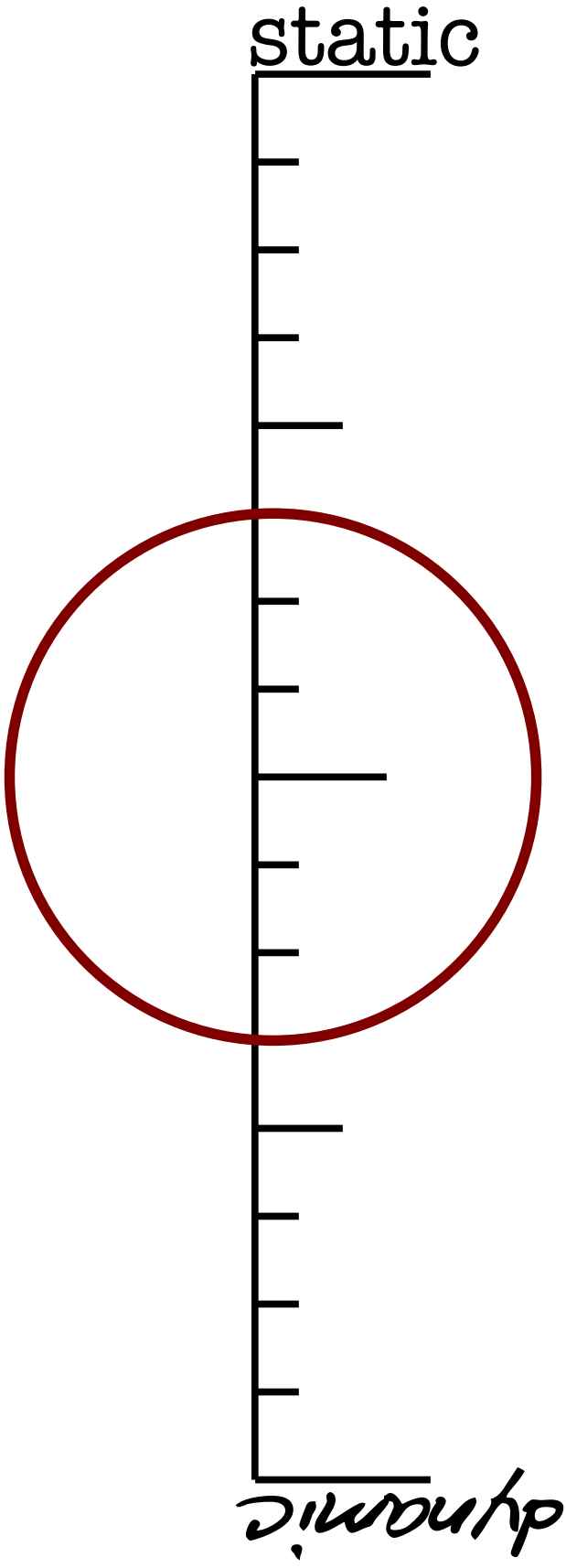
Allison Randal
Parrot Foundation &
O'Reilly Media, Inc.

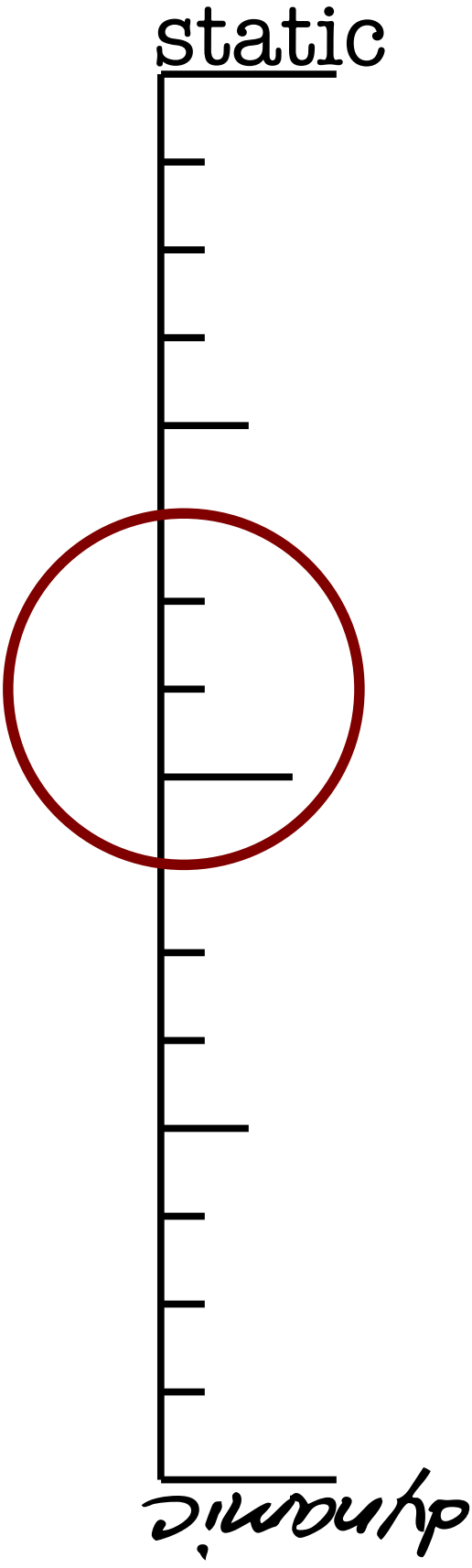
A different problem definition
inevitably leads to
a different solution space.

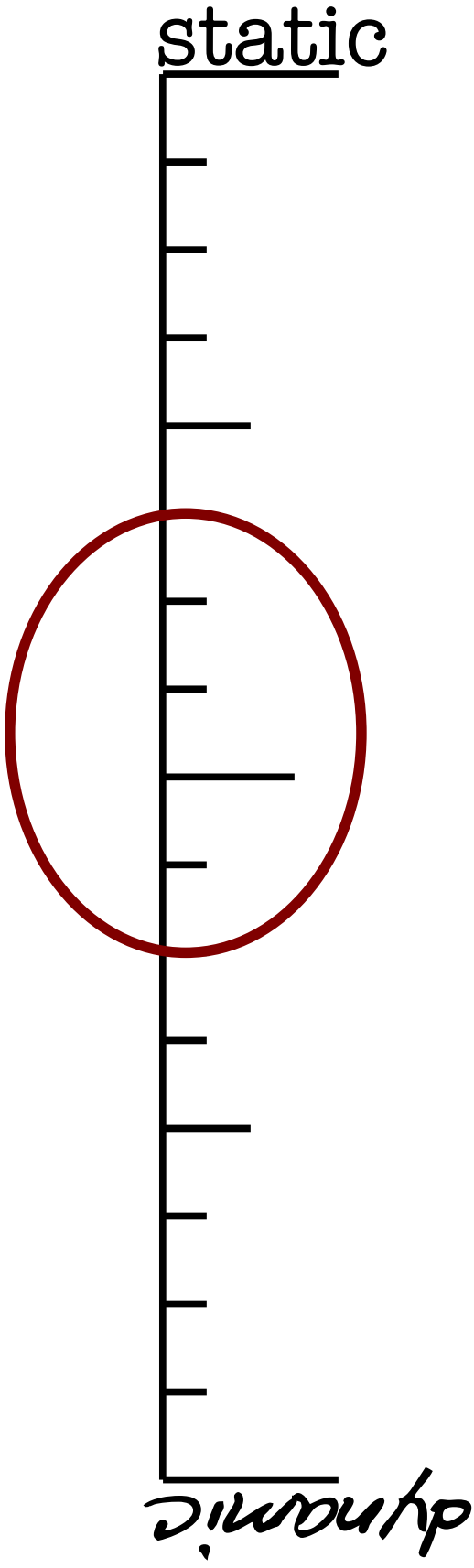
static

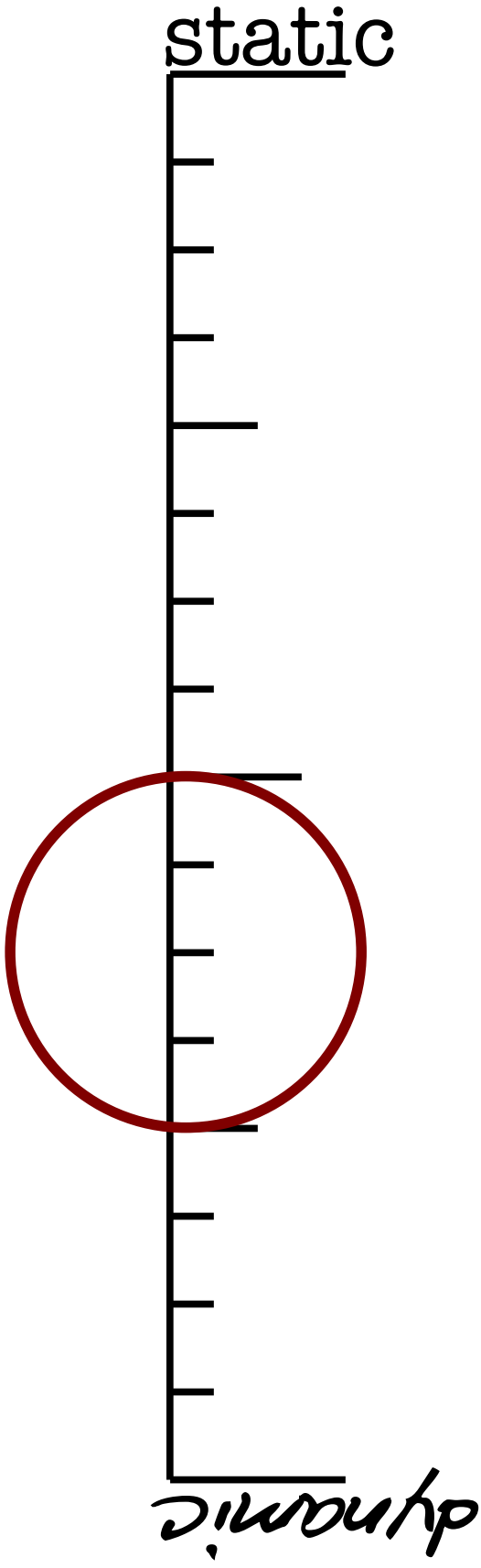
dynamic

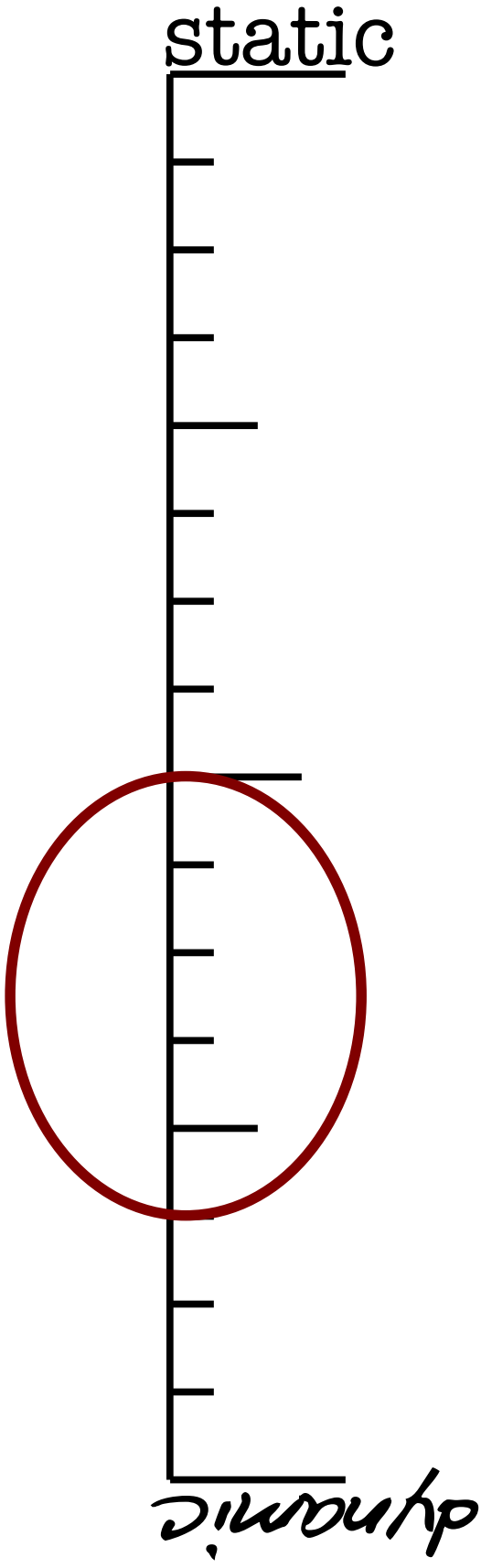
static
dynamic


static
dynamic


static
dynamic


static
dynamic


static
dynamic


static
dynamic

Dynamic Features

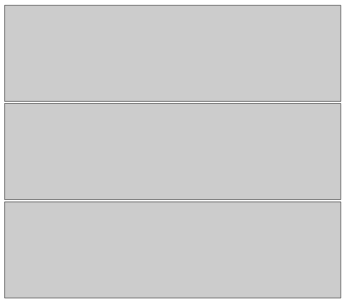
- Runtime vs. compile-time
- Extend code (eval, load)
- Define/modify classes
- Alter type system
- Higher-order functions
- Closures, Continuations, Coroutines
- Anything can change

Goals

- Revolution...
- ...becoming Establishment
- Powerful tools
- Portability
- Interoperability
- Language evolution

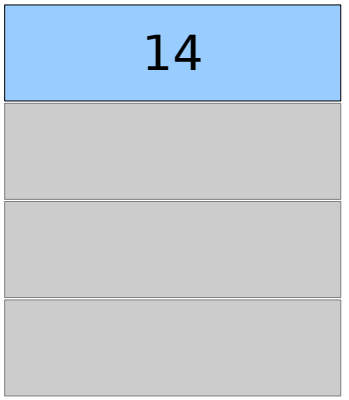
Register-based

- Stack operations



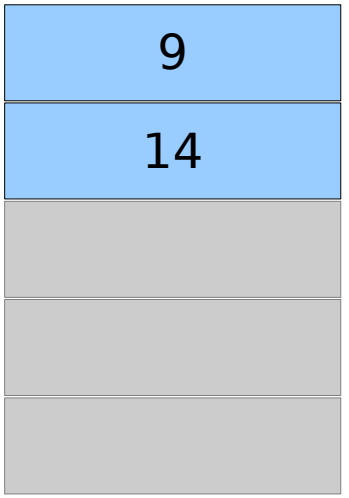
Register-based

- Stack operations



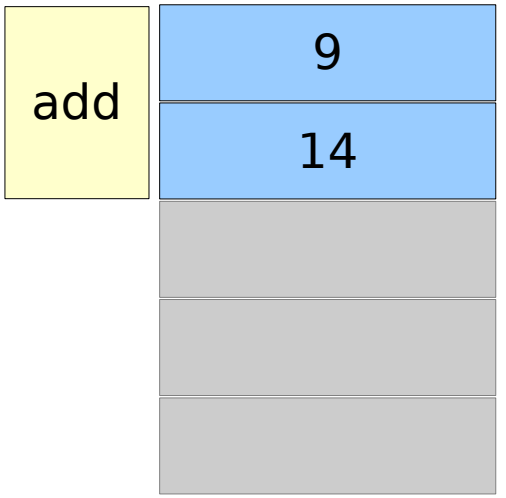
Register-based

- Stack operations



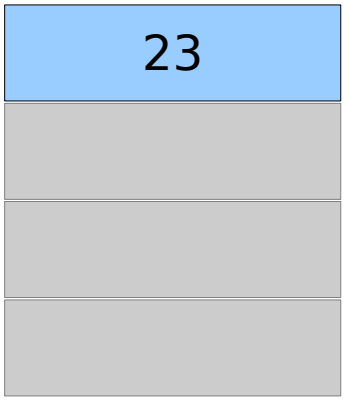
Register-based

- Stack operations



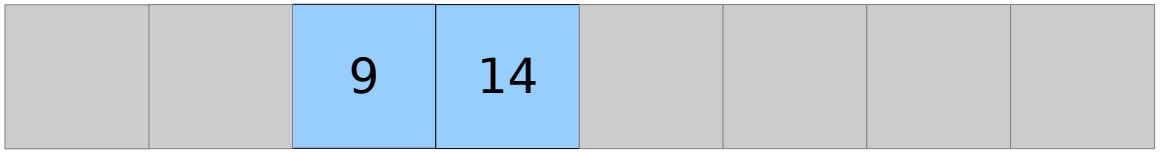
Register-based

- Stack operations



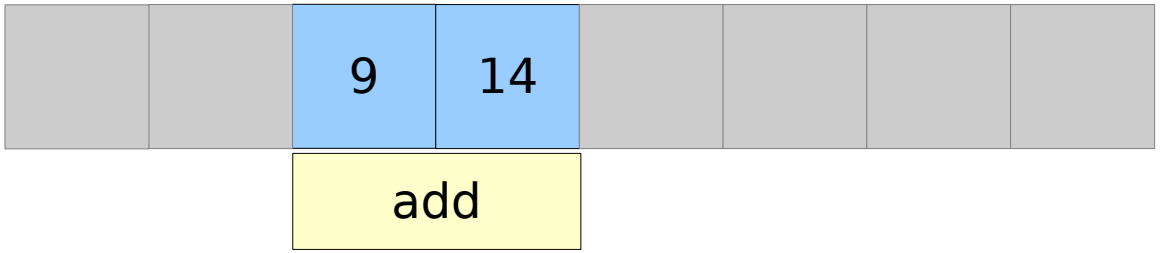
Register-based

- Stack operations
- Register operations



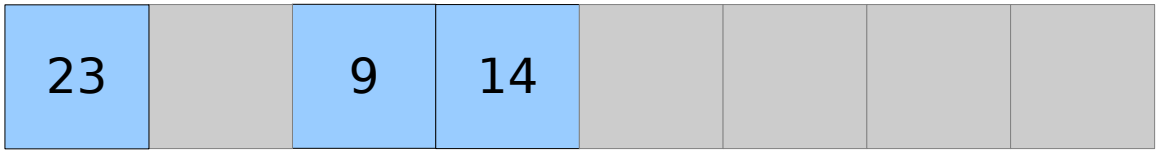
Register-based

- Stack operations
- Register operations



Register-based

- Stack operations
- Register operations



Register-based

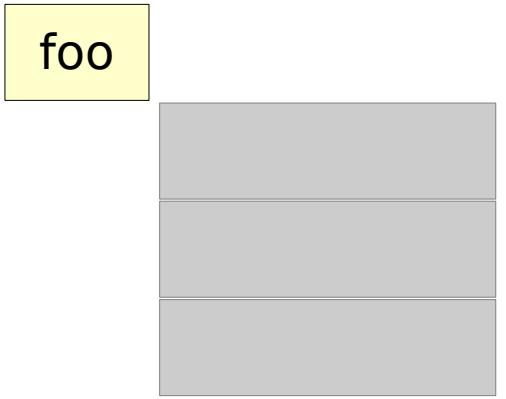
- Stack operations
- Register operations
- Fewer instructions
- Hardware registers
- Register spilling
- Flexible register sets

Continuation Passing Style

- Stack-based control flow

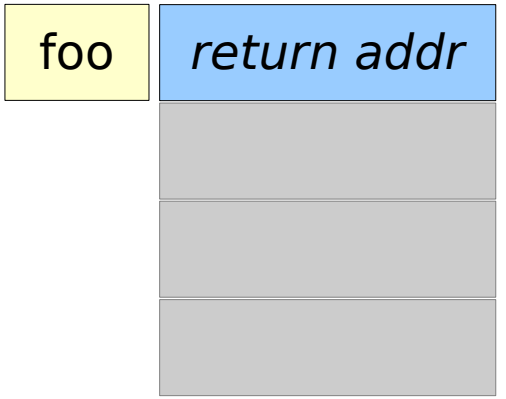
Continuation Passing Style

- Stack-based control flow



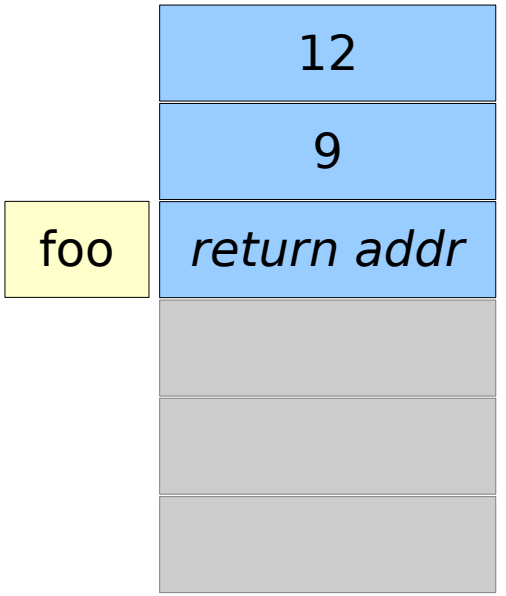
Continuation Passing Style

- Stack-based control flow



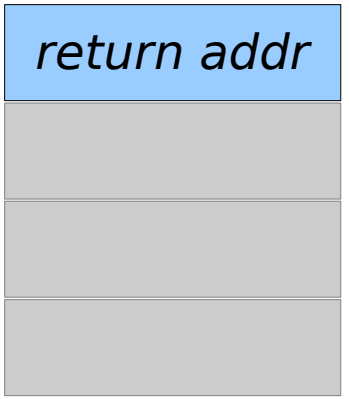
Continuation Passing Style

- Stack-based control flow



Continuation Passing Style

- Stack-based control flow

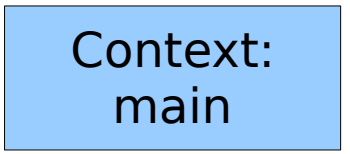


Continuation Passing Style

- Stack-based control flow

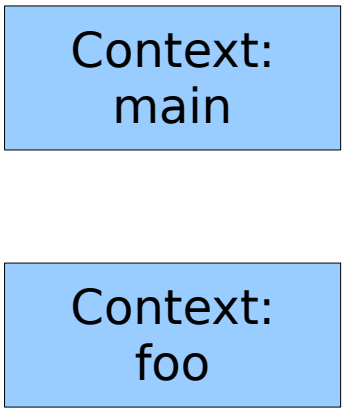
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



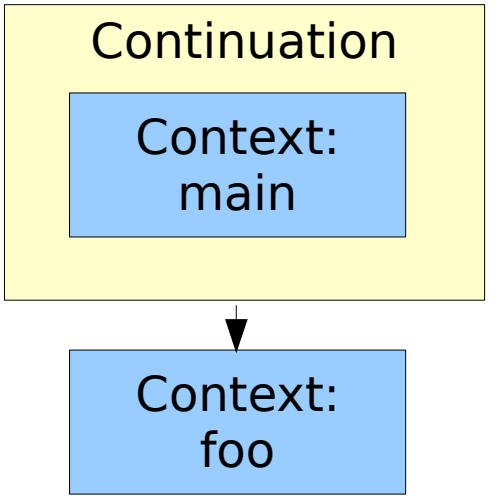
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



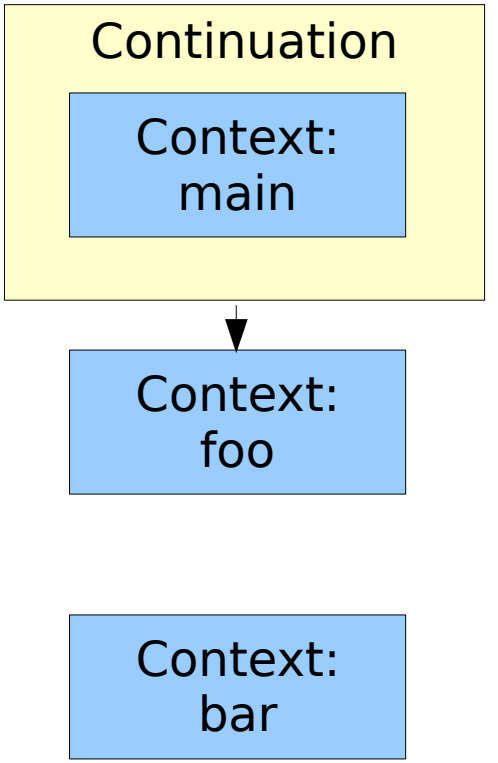
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



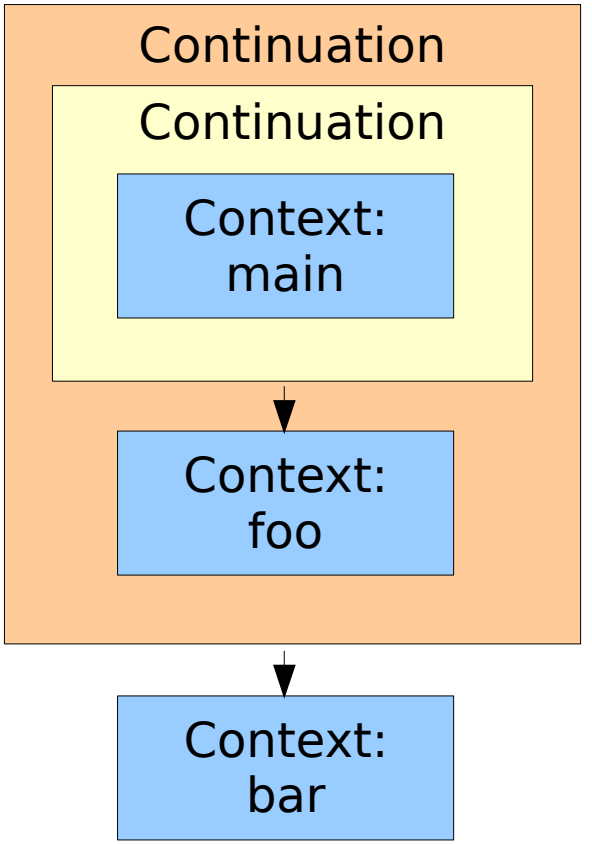
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



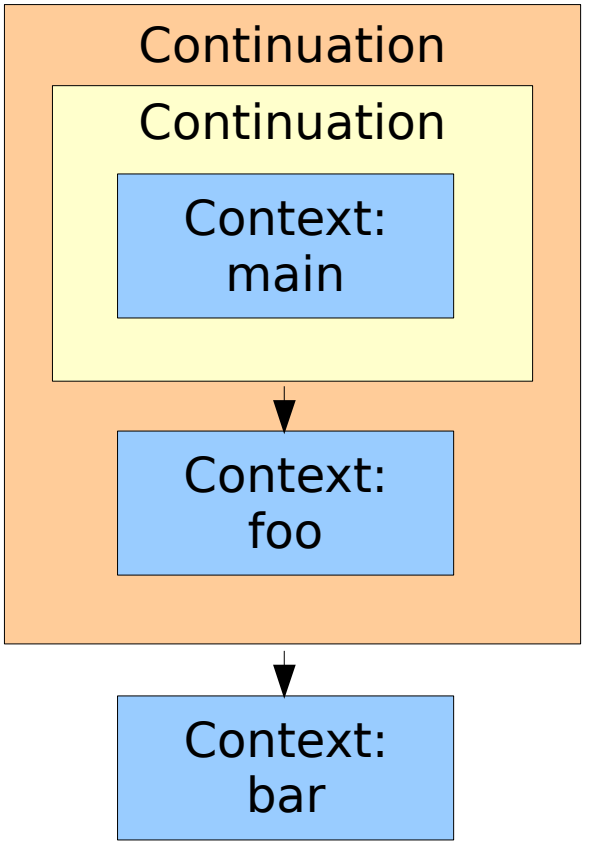
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow
- Linked list
- Cheap continuations
- Deeply nested contexts
- Tail recursion



Dynamic Dispatch

- Runtime variation
- Optional parameters
- Named parameters
- Aggregating parameters
- Multiple dispatch
- Dispatch by value

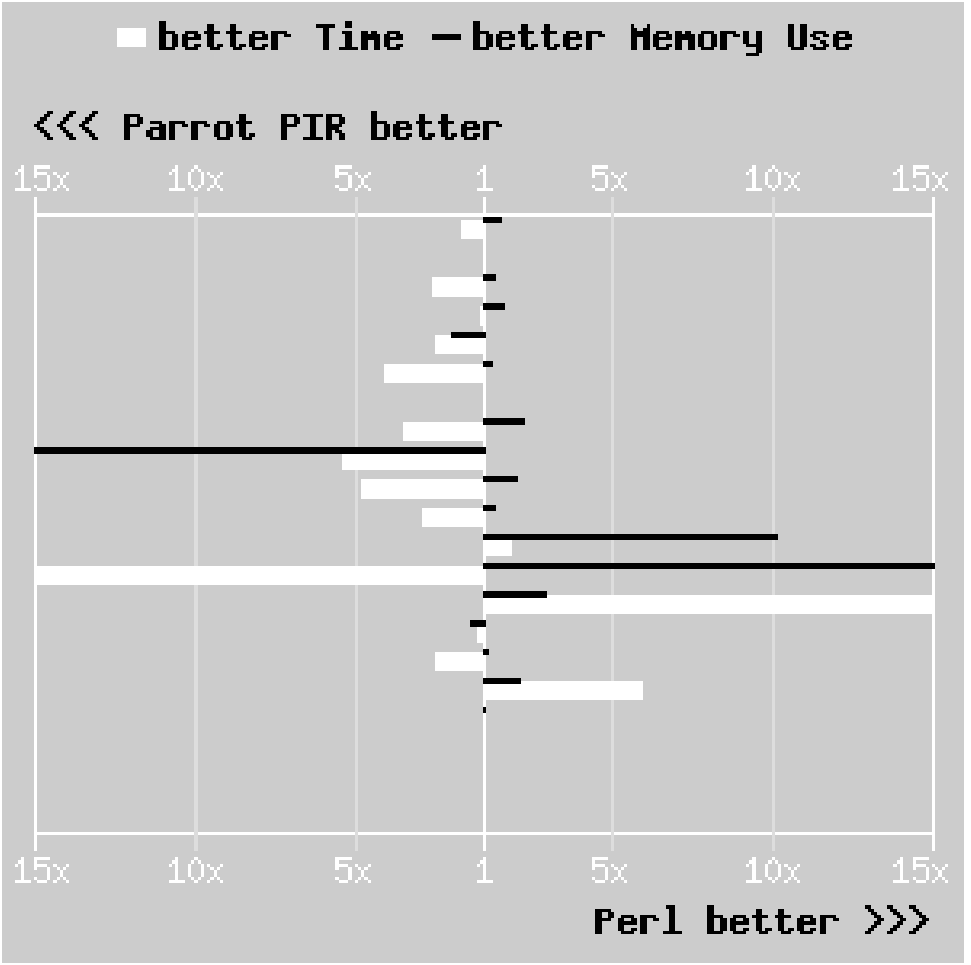
Dynamic Dispatch

- Difficult?
- Static call munging
- Function as object
- Integrate CPS
- Encapsulated dispatch
- Concurrency

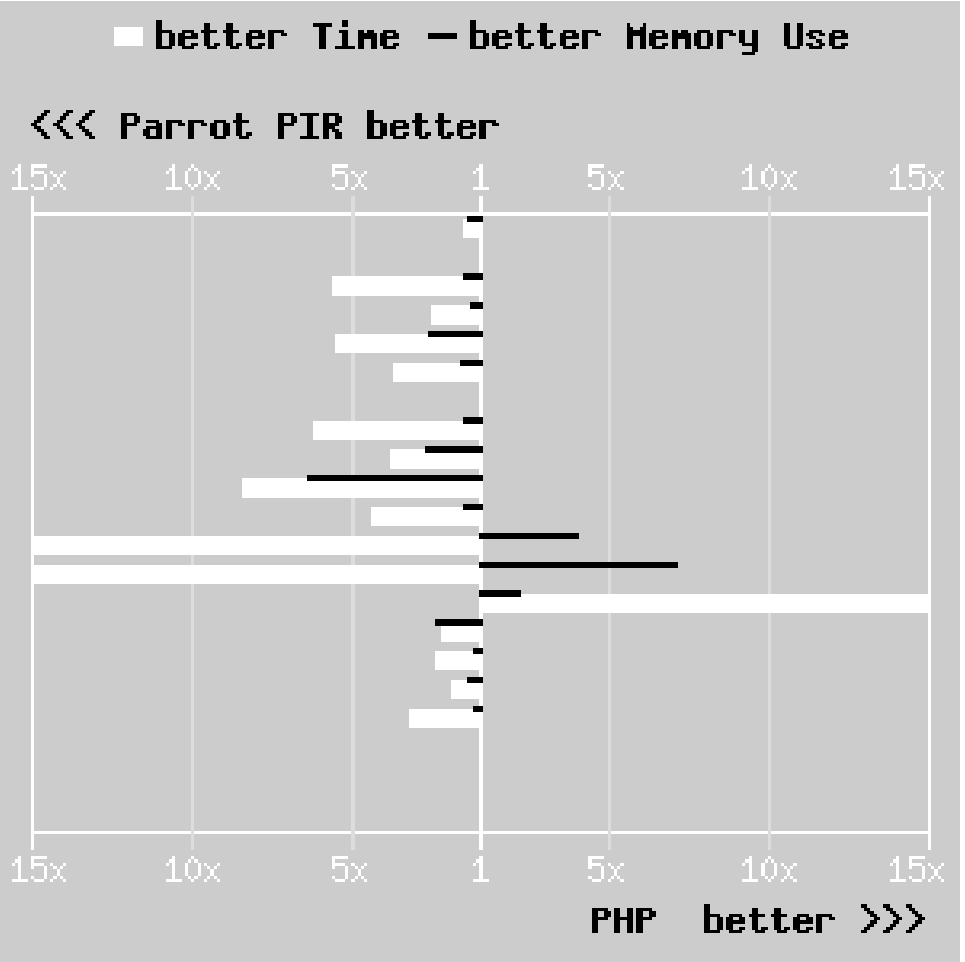
Optimization

- Procrastination principle
- Late binding
- Lazy evaluation
- Throw-away code
- Scripting
- Startup time

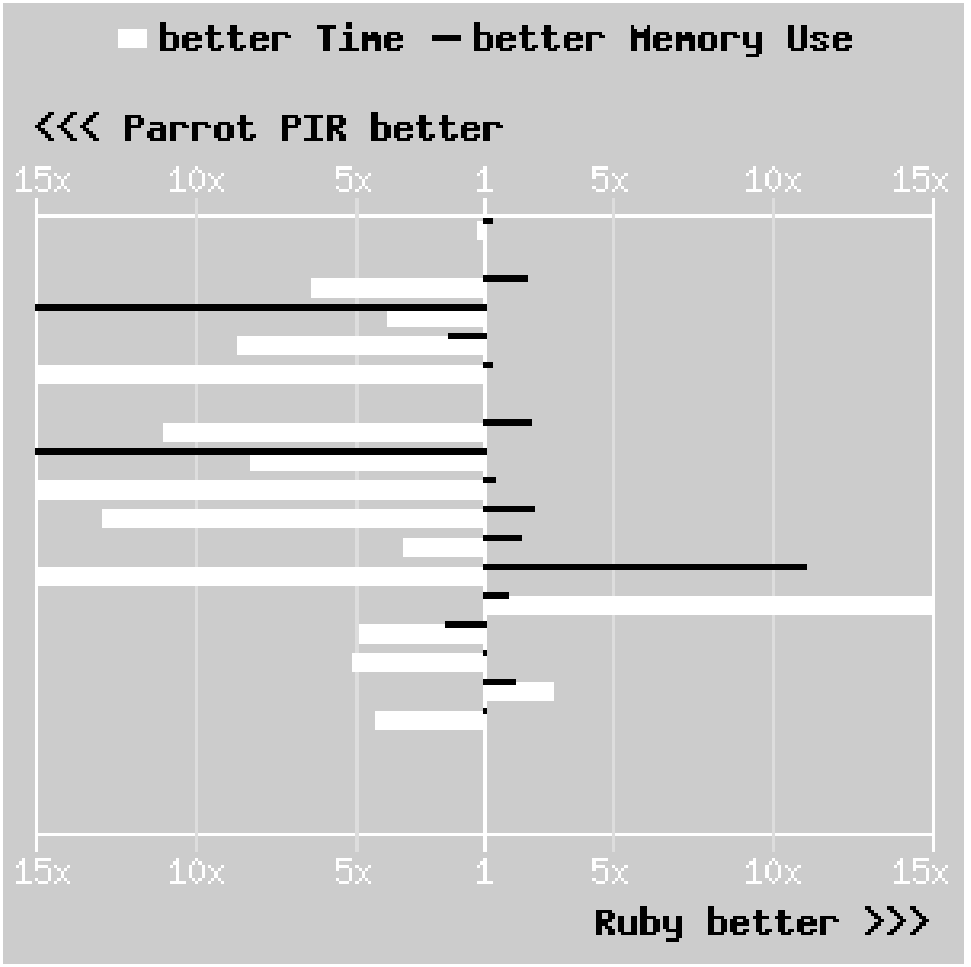
Parrot to Perl



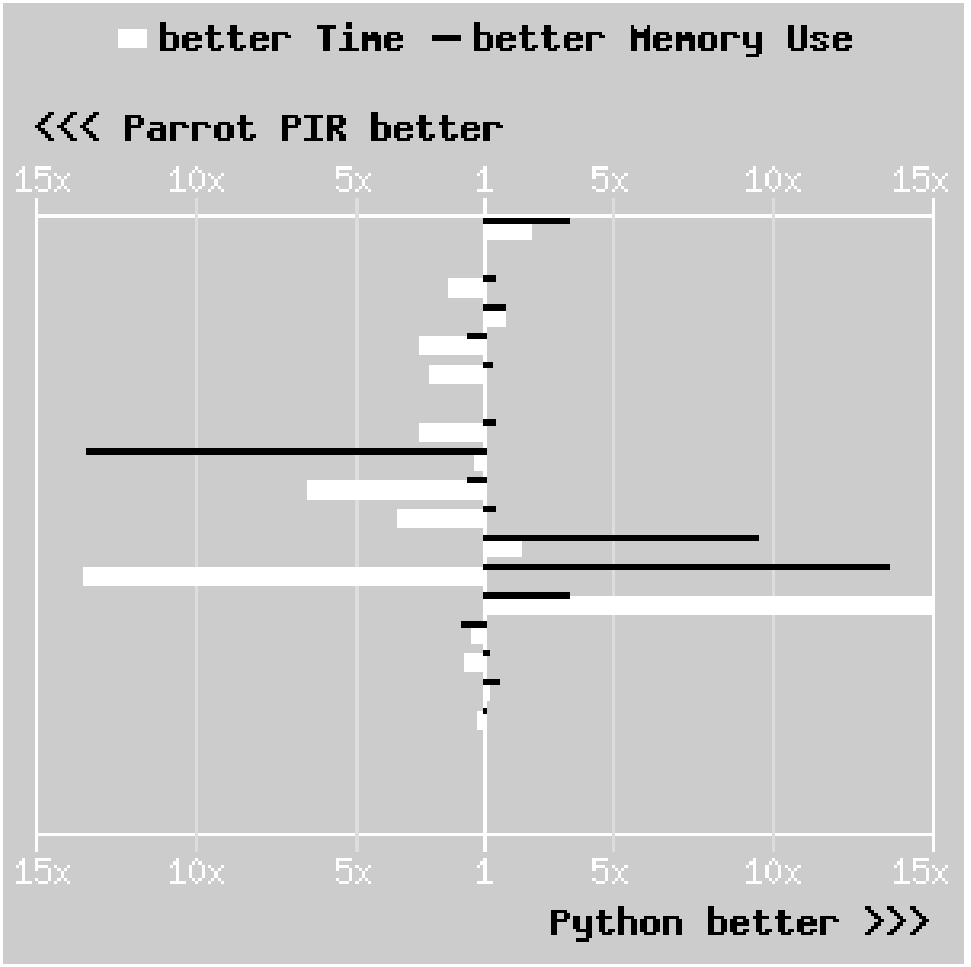
Parrot to PHP



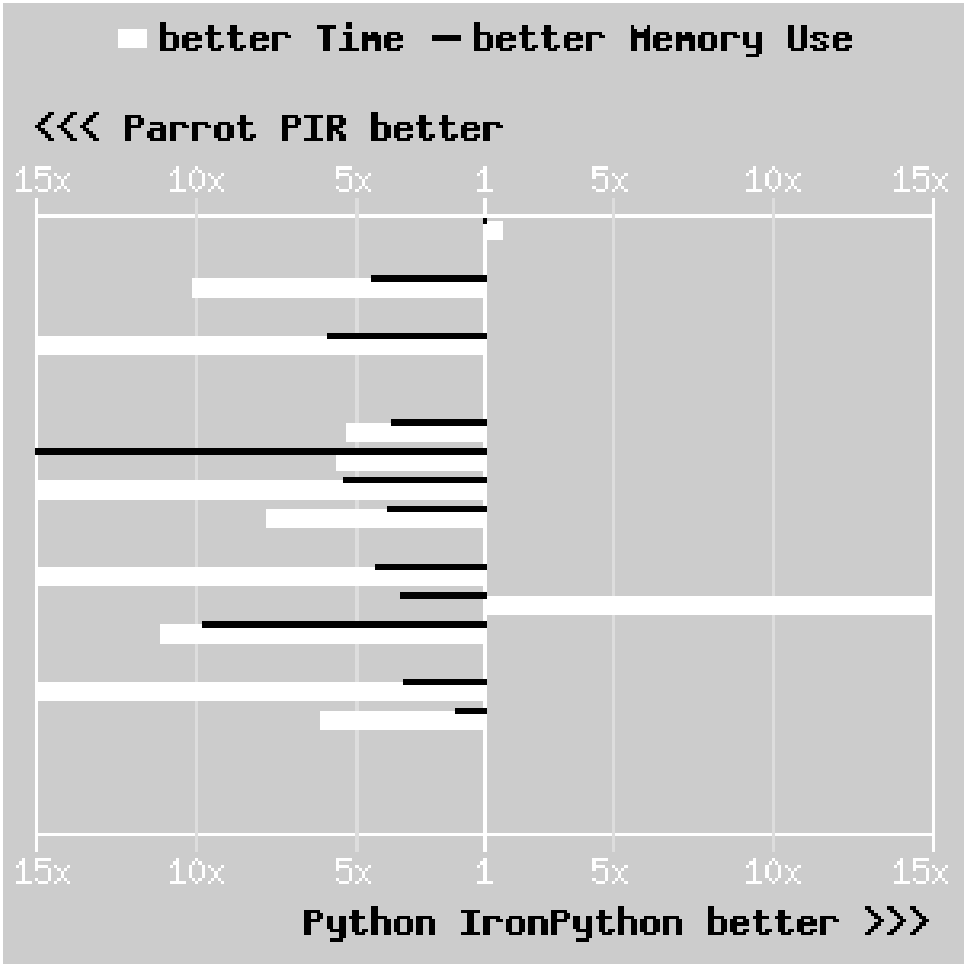
Parrot to Ruby



Parrot to Python



Parrot to IronPython



Dynamic Parser

- Regular expressions
- Recursive descent
- Operator precedence parser
- Cheap backtracking
- Tree transformation

Questions?

- Further Reading
  - “Continuations and advanced flow control” by Jonathan Bartlett *<http://www.ibm.com/developerworks/linux/library/l-advflow.html>*
  - “The case for virtual register machines” by Brian Davis, et al. *<http://portal.acm.org/citation.cfm?id=858575>*

Pynie

- Download

```
http://www.parrotcode.org
```

- Build

```
$ perl Configure.PL
$ make test
```

- Language

```
$ cd languages/pynie
$ make test
```

Pynie

- hello.py

```
print "Hello, World!"
```

- Run

```
$ parrot pynie.pir hello.py
```

pynie.pir

- 67 lines
- Half documentation

```
c = hllcompiler.new()
c.language('Pynie')
c.parsegrammar('Pynie::Grammar')
c.parseactions('Pynie::Grammar::Actions')
```

Grammar.pg

- Parser

```
token stmt_list {
  <simple_stmt> [ ';' <simple_stmt> ]*
';'?
  {*}
}
```

- Familiar?

```
stmt_list ::=
  simple_stmt (";" simple_stmt)* [";"]
```

Actions.pct

- Transform to AST

```
method identifier($/) {
    make PAST::Var.new( :name( ~$/ ),
                        :scope('package'),
                        :node($/)
                      );
}
```