What to expect from this talk
What to expect from this talk

Python ∪ Prolog
What to expect from this talk

Python ∪ PHP
Our problem

We want better programming languages. But better always seems to end up bigger.
We want **better** programming languages
We want better programming languages

But better always seems to end up bigger
Underlying language composition challenges
Underlying language composition challenges

Python

Bridge

PHP
Underlying language composition challenges

Python
runtime
syntax

Bridge
runtime
syntax

PHP
runtime
syntax
Underlying language composition challenges

Language boxes

Python

syntax

runtime

PHP

syntax

runtime

Bridge

syntax

runtime
Underlying language composition challenges

- **Syntax**
  - Python
  - PHP

- **Runtime**
  - Python
  - PHP

- **Language boxes**
  - Bridge
    - Syntax
    - Runtime

- **Composed meta-tracing VMs**
Syntax composition

PL X

<grammar>

expr::= ...
term::= ...
    | ...
    | ...
func ::= ...

PL Y

<program>

for (j : js) {
    doStuff();
}
.
.
.
Syntax composition

PL X
<grammar>
expr ::= ...
term ::= ...
| ...
| ...
func ::= ...

PL Y
<program>
for (j : js) {
    doStuff();
}
.
.
.

Parser
Syntax composition

Parser

Parse Tree

PL X
<grammar>
expr::= ...
term::= ...
    | ...
    | ...
func ::= ...

PL Y
<program>
for (j : js) {
doStuff();
}
.
.
.

Parser

Parse Tree
Syntax composition

PL X
<grammar>

expr ::= ...
term ::= ...
    | ...
    | ...
func ::= ...

PL Y
<program>

for (j : js) {
    doStuff();
}
.
.
.

LR
Parse Tree
Syntax composition

PL X
<grammar>
expr ::= ...
term ::= ...
    | ...
    | ...
func ::= ...

PL Y
<program>
for (j : js) {
    doStuff();
}
.
.
.

Parse Tree
Undefined

LR

Software Development Team

5 / 22 http://soft-dev.org/
Syntax composition

PL X
<grammar>
expr ::= ...
term ::= ...
    | ...
    | ...
func ::= ...

PL Y
<program>
for (j : js) {
    doStuff();
}
.
.
.

Generalised
Parse Tree
PL X
<grammar>
expr::= ...
term::= ...
    | ...
    | ...
func ::= ...

PL Y
<program>
for (j : js) {
    doStuff();
}
.
.
.

Generalised Parse Tree
Ambiguous
 Syntax composition

PL X
<grammar>
expr ::= ...
term ::= ...
    | ...
    | ...
func ::= ...

PL Y
<program>
for (j : js) {
    doStuff();
}
.
.
.

PEG
Parse Tree

http://soft-dev.org/
Syntax composition

PL X
<grammar>
expr ::= ...
term ::= ...
    | ...
    | ...
func ::= ...

PL Y
<program>
for (j : js) {
doStuff();
}
.
.
.

PEG
Parse Tree
Shadows

Software Development Team
The only choice?
The only choice?

SDE
Challenge: SDE’s power + a text editor feel?
Eco demo
Runtime composition

PL X
Interpreter

C/C++

PL Y
Interpreter

http://soft-dev.org/
Runtime composition

Too slow

C/C++
Runtime composition

PL X
  Interpreter

PL Y
  Interpreter

C/C++
Runtime composition

Too much engineering
Runtime composition

PL X

Interpreter

PL Y

Interpreter

JVM/CLR

JIT Compiler
Runtime composition

PL X
Interpreter
JIT Compiler
PL Y
Interpreter
JVM/CLR

Semantic mismatch
Runtime composition

PL X

PL Y

Interpreters

Glue

Meta-tracing

PL Z

Interpreter

Tracing JIT

9 / 22 http://soft-dev.org/
Warning: draft numbers ahead
## Absolute timing comparison

<table>
<thead>
<tr>
<th>VM</th>
<th>Benchmark</th>
<th>Python</th>
<th>Prolog</th>
<th>Python → Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPython-SWI</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmallFunc</td>
<td>0.125s ± 0.006</td>
<td>0.257s ± 0.001</td>
<td>28.893s ± 0.175</td>
<td></td>
</tr>
<tr>
<td>Loop1Arg0Result</td>
<td>2.924s ± 0.215</td>
<td>7.352s ± 0.037</td>
<td>9.310s ± 0.065</td>
<td></td>
</tr>
<tr>
<td>Loop1Arg1Result</td>
<td>4.184s ± 0.028</td>
<td>18.890s ± 0.082</td>
<td>20.865s ± 0.050</td>
<td></td>
</tr>
<tr>
<td>NondetLoop1Arg1Result</td>
<td>7.531s ± 0.065</td>
<td>18.643s ± 0.159</td>
<td>667.682s ± 5.594</td>
<td></td>
</tr>
<tr>
<td>TermConstruction</td>
<td>264.415s ± 1.815</td>
<td>48.819s ± 0.208</td>
<td>2185.150s ± 14.251</td>
<td></td>
</tr>
<tr>
<td>Lists</td>
<td>9.374s ± 0.046</td>
<td>25.148s ± 0.182</td>
<td>2207.304s ± 12.344</td>
<td></td>
</tr>
<tr>
<td><strong>Unipyication</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmallFunc</td>
<td>0.001s ± 0.000</td>
<td>0.006s ± 0.001</td>
<td>0.001s ± 0.000</td>
<td></td>
</tr>
<tr>
<td>Loop1Arg0Result</td>
<td>0.085s ± 0.000</td>
<td>0.086s ± 0.000</td>
<td>0.087s ± 0.000</td>
<td></td>
</tr>
<tr>
<td>Loop1Arg1Result</td>
<td>0.112s ± 0.000</td>
<td>0.114s ± 0.000</td>
<td>0.115s ± 0.000</td>
<td></td>
</tr>
<tr>
<td>NondetLoop1Arg1Result</td>
<td>0.500s ± 0.002</td>
<td>0.548s ± 0.064</td>
<td>2.674s ± 0.010</td>
<td></td>
</tr>
<tr>
<td>TermConstruction</td>
<td>6.053s ± 0.218</td>
<td>2.444s ± 0.002</td>
<td>36.069s ± 0.171</td>
<td></td>
</tr>
<tr>
<td>Lists</td>
<td>0.845s ± 0.002</td>
<td>1.416s ± 0.003</td>
<td>5.056s ± 0.026</td>
<td></td>
</tr>
<tr>
<td><strong>Jython-tuProlog</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmallFunc</td>
<td>0.088s ± 0.002</td>
<td>3.050s ± 0.036</td>
<td>52.294s ± 0.371</td>
<td></td>
</tr>
<tr>
<td>Loop1Arg0Result</td>
<td>1.078s ± 0.007</td>
<td>206.590s ± 2.884</td>
<td>199.963s ± 1.784</td>
<td></td>
</tr>
<tr>
<td>Loop1Arg1Result</td>
<td>2.145s ± 0.175</td>
<td>293.311s ± 4.270</td>
<td>294.781s ± 4.746</td>
<td></td>
</tr>
<tr>
<td>NondetLoop1Arg1Result</td>
<td>7.939s ± 0.341</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>TermConstruction</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>Lists</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>VM</td>
<td>Benchmark</td>
<td>Python→Prolog</td>
<td>Python→Prolog</td>
<td>Python→Prolog</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python</td>
<td>Prolog</td>
<td>Unipycation</td>
</tr>
<tr>
<td>CPython-SWI</td>
<td>SmallFunc</td>
<td>231.770× ±10.154</td>
<td>112.567× ±0.934</td>
<td>27821.079× ±1896.725</td>
</tr>
<tr>
<td></td>
<td>Loop1Arg0Result</td>
<td>3.184× ±0.232</td>
<td>1.266× ±0.011</td>
<td>107.591× ±0.779</td>
</tr>
<tr>
<td></td>
<td>Loop1Arg1Result</td>
<td>4.987× ±0.039</td>
<td>1.105× ±0.006</td>
<td>181.899× ±0.444</td>
</tr>
<tr>
<td></td>
<td>NondetLoop1Arg1Result</td>
<td>88.654× ±1.026</td>
<td>35.814× ±0.389</td>
<td>249.737× ±2.244</td>
</tr>
<tr>
<td></td>
<td>TermConstruction</td>
<td>8.264× ±0.081</td>
<td>44.760× ±0.348</td>
<td>60.583× ±0.487</td>
</tr>
<tr>
<td></td>
<td>Lists</td>
<td>235.459× ±1.742</td>
<td>87.772× ±0.789</td>
<td>436.609× ±3.494</td>
</tr>
<tr>
<td>Unipycaction</td>
<td>SmallFunc</td>
<td>1.295× ±0.086</td>
<td>0.182× ±0.036</td>
<td>1.000×</td>
</tr>
<tr>
<td></td>
<td>Loop1Arg0Result</td>
<td>1.020× ±0.001</td>
<td>1.012× ±0.002</td>
<td>1.000×</td>
</tr>
<tr>
<td></td>
<td>Loop1Arg1Result</td>
<td>1.025× ±0.002</td>
<td>1.002× ±0.002</td>
<td>1.000×</td>
</tr>
<tr>
<td></td>
<td>NondetLoop1Arg1Result</td>
<td>5.349× ±0.035</td>
<td>4.879× ±0.631</td>
<td>1.000×</td>
</tr>
<tr>
<td></td>
<td>TermConstruction</td>
<td>5.959× ±0.224</td>
<td>14.756× ±0.069</td>
<td>1.000×</td>
</tr>
<tr>
<td></td>
<td>Lists</td>
<td>5.982× ±0.034</td>
<td>3.569× ±0.019</td>
<td>1.000×</td>
</tr>
<tr>
<td>Jython-tuProlog</td>
<td>SmallFunc</td>
<td>592.904× ±14.602</td>
<td>17.143× ±0.259</td>
<td>50354.204× ±3330.993</td>
</tr>
<tr>
<td></td>
<td>Loop1Arg0Result</td>
<td>185.460× ±2.182</td>
<td>0.968× ±0.017</td>
<td>2310.844× ±21.996</td>
</tr>
<tr>
<td></td>
<td>Loop1Arg1Result</td>
<td>137.427× ±11.805</td>
<td>1.005× ±0.022</td>
<td>2569.873× ±41.331</td>
</tr>
<tr>
<td></td>
<td>NondetLoop1Arg1Result</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td></td>
<td>TermConstruction</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td></td>
<td>Lists</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
</tbody>
</table>
Warning: even draftier numbers ahead!
## Composed Richards vs. other VMs

<table>
<thead>
<tr>
<th>Type</th>
<th>VM</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyPy</td>
<td>2.4.0</td>
<td>0.370 ± 0.000</td>
</tr>
<tr>
<td>Hippy</td>
<td></td>
<td>0.553 ± 0.008</td>
</tr>
<tr>
<td>Mono</td>
<td>Bridge</td>
<td>0.556 ± 0.006</td>
</tr>
<tr>
<td></td>
<td>HHVM 3.2.0</td>
<td>5.353 ± 0.262</td>
</tr>
<tr>
<td></td>
<td>ZEND 5.4.4</td>
<td>10.406 ± 0.106</td>
</tr>
</tbody>
</table>
## Composed Richards vs. other VMs

<table>
<thead>
<tr>
<th>Type</th>
<th>VM</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PyPy 2.4.0</td>
<td>0.370 ± 0.000</td>
</tr>
<tr>
<td></td>
<td>Hippy</td>
<td>0.553 ± 0.008</td>
</tr>
<tr>
<td>Mono</td>
<td>Bridge</td>
<td>0.556 ± 0.006</td>
</tr>
<tr>
<td></td>
<td>HHVM 3.2.0</td>
<td>5.353 ± 0.262</td>
</tr>
<tr>
<td></td>
<td>ZEND 5.4.4</td>
<td>10.406 ± 0.105</td>
</tr>
<tr>
<td>Composed</td>
<td>Bridge</td>
<td>0.936 ± 0.038</td>
</tr>
</tbody>
</table>
Datatype conversion

PHPObject

PHPRoot

PHPInt

PHPFunc
Datatype conversion

```
PHPRoot
PHPObject  PHPInt  PHPFunc
PyRoot
PyObject  PyInt  PyFunc
```

16 / 22 http://soft-dev.org/
Datatype conversion: primitive types

PHP

2 : PHPInt

Python
Datatype conversion: user types

PHP

Python
Datatype conversion: user types

PHP

\[ o : \text{PHPObject} \]

Python
Datatype conversion: user types

PyRoot

PyObject  PyInt  PyFunc
Datatype conversion: user types

```
 PyRoot
  ▲
 PyObject  PyInt  PyFunc
               ▼
 ProxiedPHPObject
```

http://soft-dev.org/
Datatype conversion: user types

PyRoot

PyObject

PyInt

PyFunc

ProxiedPHPObject

php_obj : PHPObject
Datatype conversion: user types

PHP

Python

o : PHPObjec
Datatype conversion: user types

PHP

o : PHPObject

Python

:ProxiedPHPObject
Datatype conversion: user types

PHP

: ProxiedPHPObject

Python

php_obj
Datatype conversion: user types

PHP

\[ o : \text{PHPObject} \]

Python

\[ :\text{ProxiedPHPObject} \]

\[ \text{Immutable field} \]

\[ \text{php_obj} \]
Some thoughts

- Critical: single meta-language (e.g. RPython).
Some thoughts

- Critical: single meta-language (e.g. RPython).
- Simplicity: good performance, yet understandable.
Some thoughts

• Critical: single meta-language (e.g. RPython).
• Simplicity: good performance, yet understandable.
• Immutable wrappers give near-native performance.
Some thoughts

- Critical: single meta-language (e.g. RPython).
- Simplicity: good performance, yet understandable.
- Immutable wrappers give near-native performance.
- Whole new world of challenges for language designers & formalisers.
What can we use this for?
What can we use this for?

COBOL  →  Java
What can we use this for?

"Big Bang" translation

COBOL  ➔  Java
What can we use this for?

COBOL  COOava  Java
What can we use this for?

Gradual migration

COBOL  COava  Java
Summary

Language boxes

Syntax

Runtime

Python

Syntax

Runtime

PHP

Syntax

Runtime

Bridge

Syntax

Runtime

Composed meta-tracing VMs

21 / 22 http://soft-dev.org/
http://soft-dev.org/