Polyglot
An Extensible Compiler Framework for Java

Nathaniel Nystrom
Michael R. Clarkson
Andrew C. Myers

Cornell University

Language extension

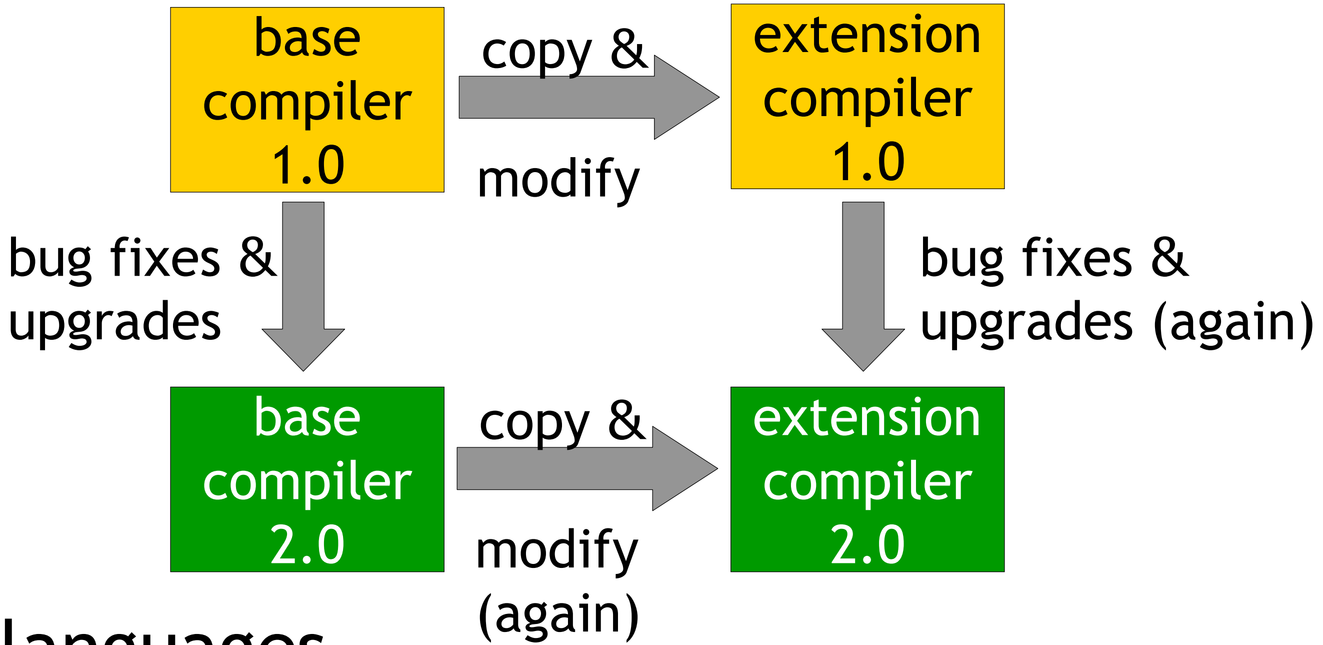
- Language designers often create extensions to existing languages
  - e.g., C++, PolyJ, GJ, Pizza, AspectJ, Jif, ArchJava, ESCJava, Polyphonic C#, ...
- Want to reuse existing compiler infrastructure as much as possible
- **Polyglot** is a framework for writing compiler extensions for Java

Requirements

- Language extension
  - Modify both syntax and semantics of the base language
  - Not necessarily backward compatible
- Goals:
  - Easy to build and maintain extensions
  - Extensibility should be **scalable**
    - No code duplication
  - Compilers for language extensions should be open to further extension

Rejected approaches

- In-place modification



- Macro languages
  - Limited to syntax extensions
  - Semantic checks *after* macro expansion

Polyglot

- Base compiler is a complete Java front end
- 25K lines of Java
  - Name resolution, inner class support, type checking, exception checking, uninitialized variable analysis, unreachable code analysis, …
- Can reuse and extend through inheritance

Scalable extensibility

Changes to the compiler should be proportional to changes in the language.

- Most compiler passes are **sparse**:

AST Nodes

Passes		+	**if**	x	*e*.f	=
	name resolution			🟦	🟦	
	type checking	🟦	🟦	🟦	🟦	🟦
	exception checking				🟦	
	constant folding	🟦				

Scalable extensibility

Changes to the compiler should be proportional to changes in the language.

- Most compiler passes are **sparse**:

AST Nodes

Passes

	+	**if**	x	*e*.f	=
name resolution	🟥		🟦	🟦	
type checking	🟥	🟦	🟦	🟦	🟦
exception checking	🟥			🟦	
constant folding	🟥				

Scalable extensibility

Changes to the compiler should be proportional to changes in the language.

- Most compiler passes are **sparse**:

AST Nodes

Passes

	+	**if**	x	*e*.f	=
name resolution			🟦	🟦	
type checking	🟨	🟦	🟦	🟦	🟦
exception checking				🟦	
constant folding	🟨				

Scalable extensibility

Changes to the compiler should be proportional to changes in the language.

- Most compiler passes are **sparse**:

AST Nodes

Passes

	+	**if**	x	*e*.f	=
name resolution			🟦	🟦	
type checking	🟦	🟦	🟦	🟦	🟦
exception checking				🟦	
constant folding	🟥	🟥	🟥	🟥	🟥

Scalable extensibility

Changes to the compiler should be proportional to changes in the language.

- Most compiler passes are **sparse**:

AST Nodes

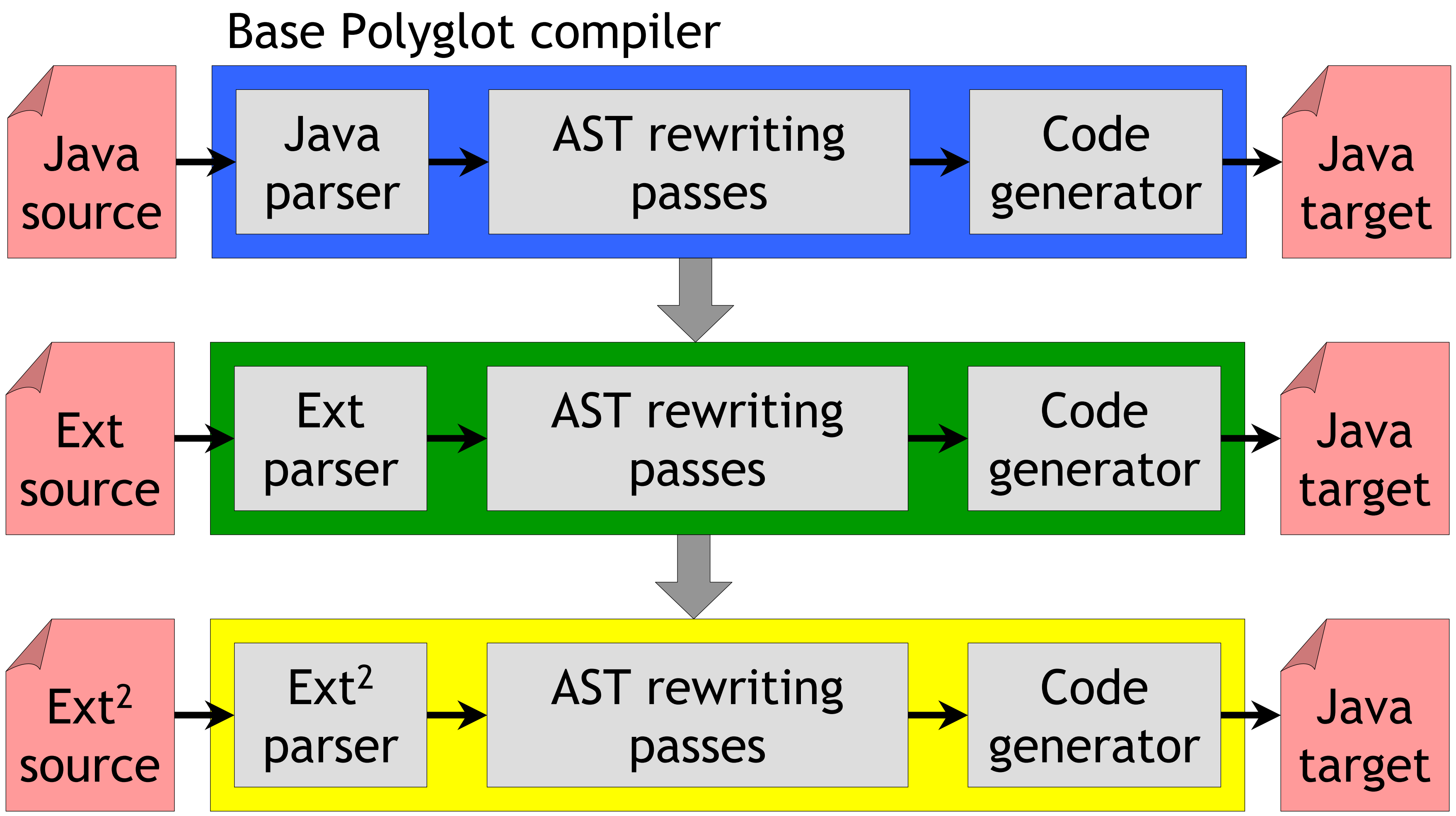
Passes

	+	**if**	x	*e*.f	=
name resolution			🟦	🟦	
type checking	🟦	🟦	🟦	🟦	🟦
exception checking				🟦	
constant folding	🟨				

Non-scalable approaches

		Easy to add or modify	
		Passes	**AST nodes**
Using	**Visitors**	✓	✗
	pass as AST node method ("naive OO")	✗	✓
	Polyglot	✓	✓

Polyglot architecture



Architecture details

- Parser written using **PPG**
  - Adds grammar inheritance to Java CUP
- AST nodes constructed using a **node factory**
  - Decouples node types from implementation
- AST rewriting passes:
  - Each pass lazily creates a new AST
  - From naive OO: traverse AST invoking a method at each node
  - From visitors: AST traversal factored out

Example: PAO

- Primitive types as subclasses of Object
- Changes type system, relaxes Java syntax
- Implementation: insert boxing and unboxing code where needed

```
HashMap m;
m.put("two", 2);
int v = (int) m.get("two");
```

$\Rightarrow$

```
HashMap m;
m.put("two", new Integer(2));
int v = ((Integer) m.get("two")).intValue();
```

PAO implementation

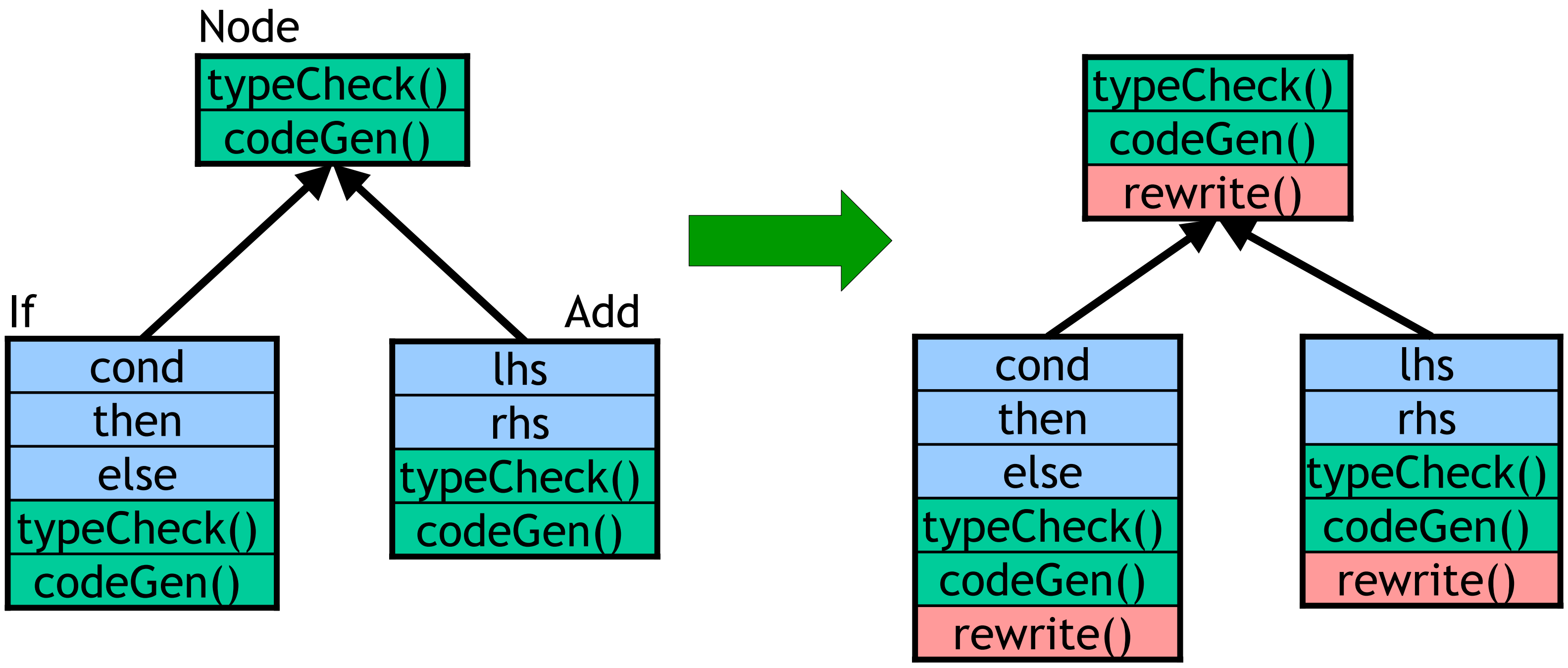
- Modify parser and type-checking pass to permit e instanceof int
    - Parser changes with PPG:

```
include "java.cup"
drop { rel_expr ::= rel_expr INSTANCEOF ref_type }
extend rel_expr ::= rel_expr:a INSTANCEOF type:b
  {: RESULT = node_factory.Instanceof(a, b); :}
```

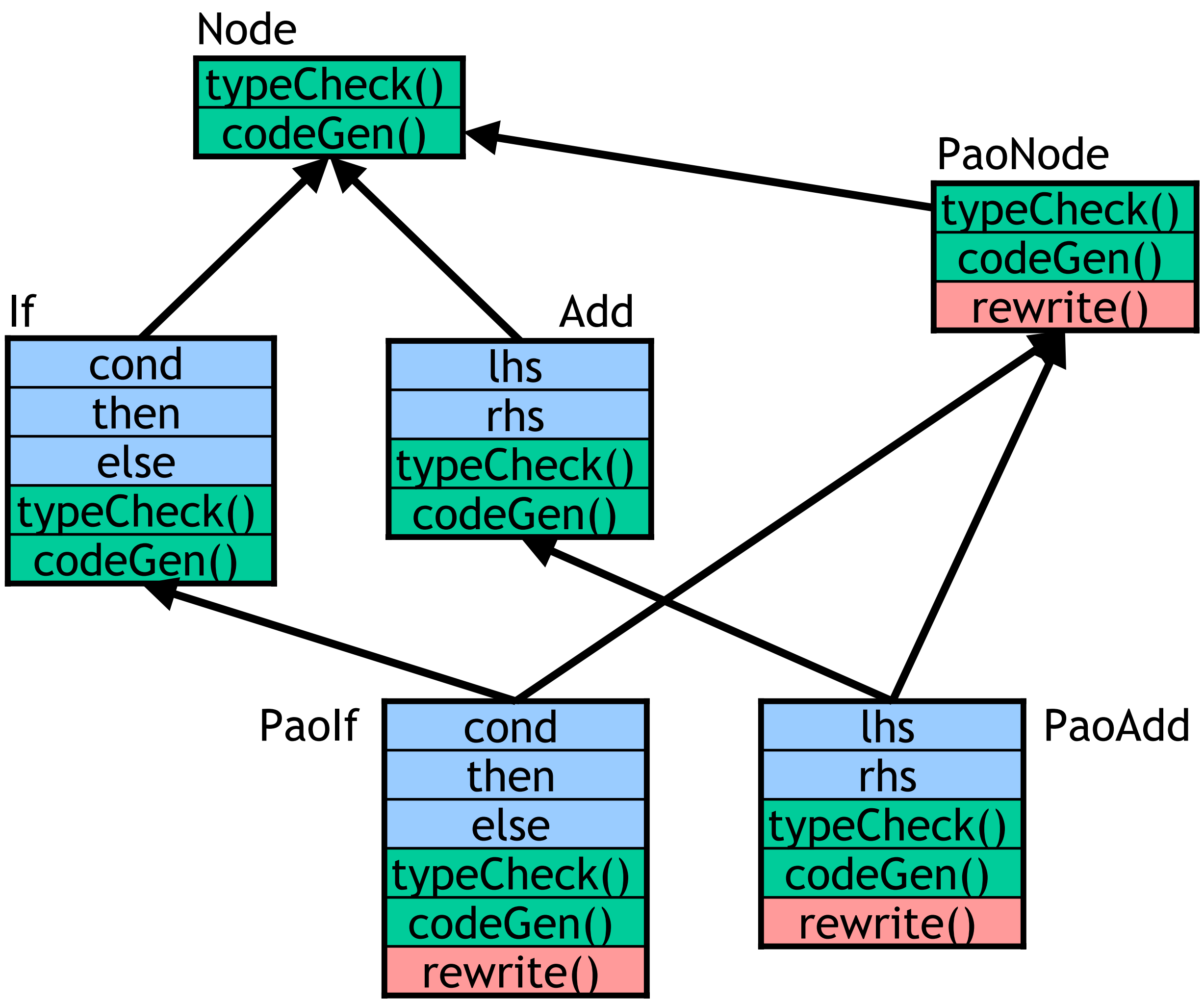
- Add one new pass to insert boxing and unboxing code

Implementing a new pass

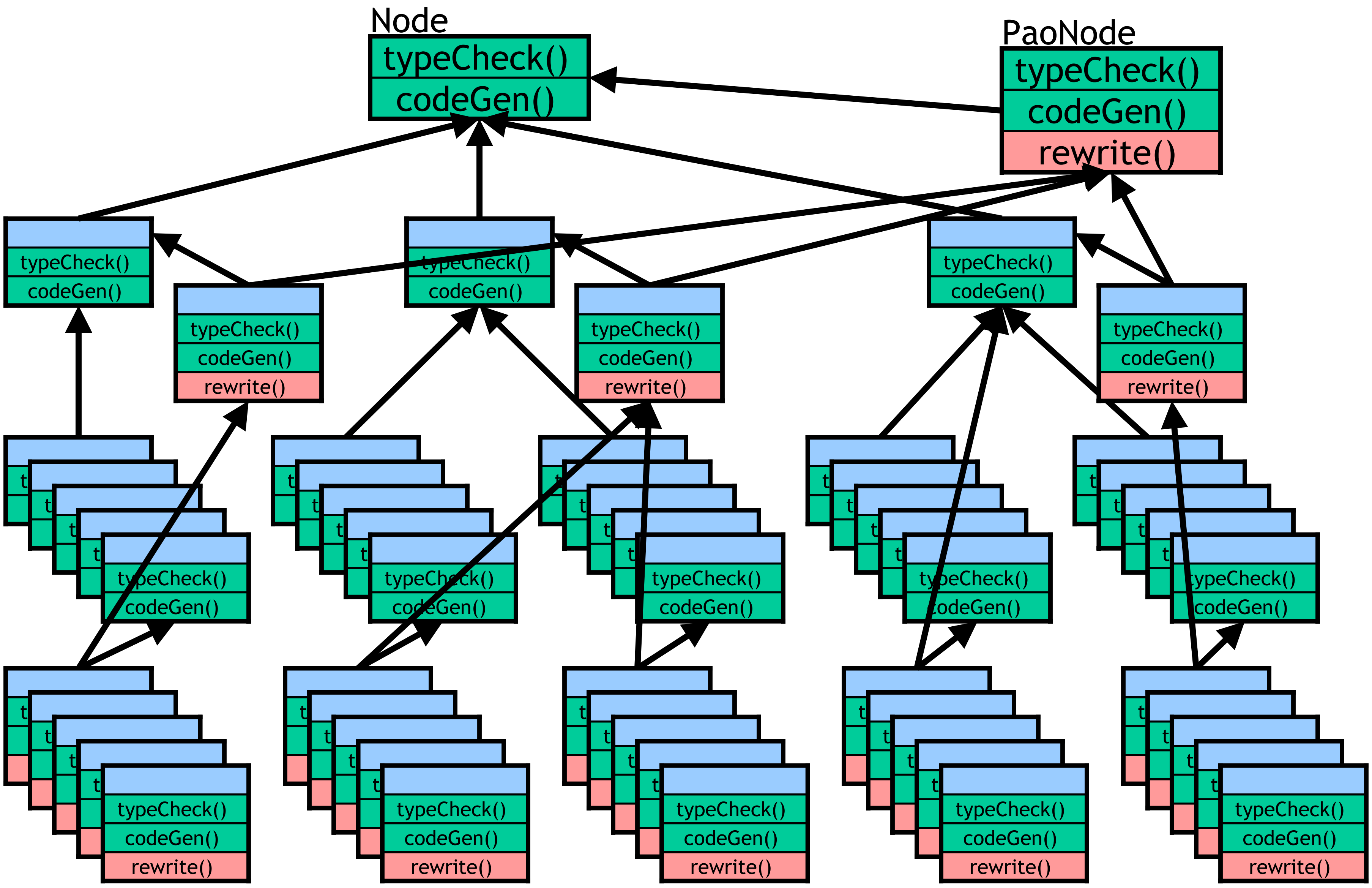
- Want to extend Node interface with rewrite() method
  - Default implementation: identity translation
  - Specialized implementations: boxing and unboxing
- **Mixin extensibility**: extensions to a base class should be inherited by subclasses



Inheritance is inadequate

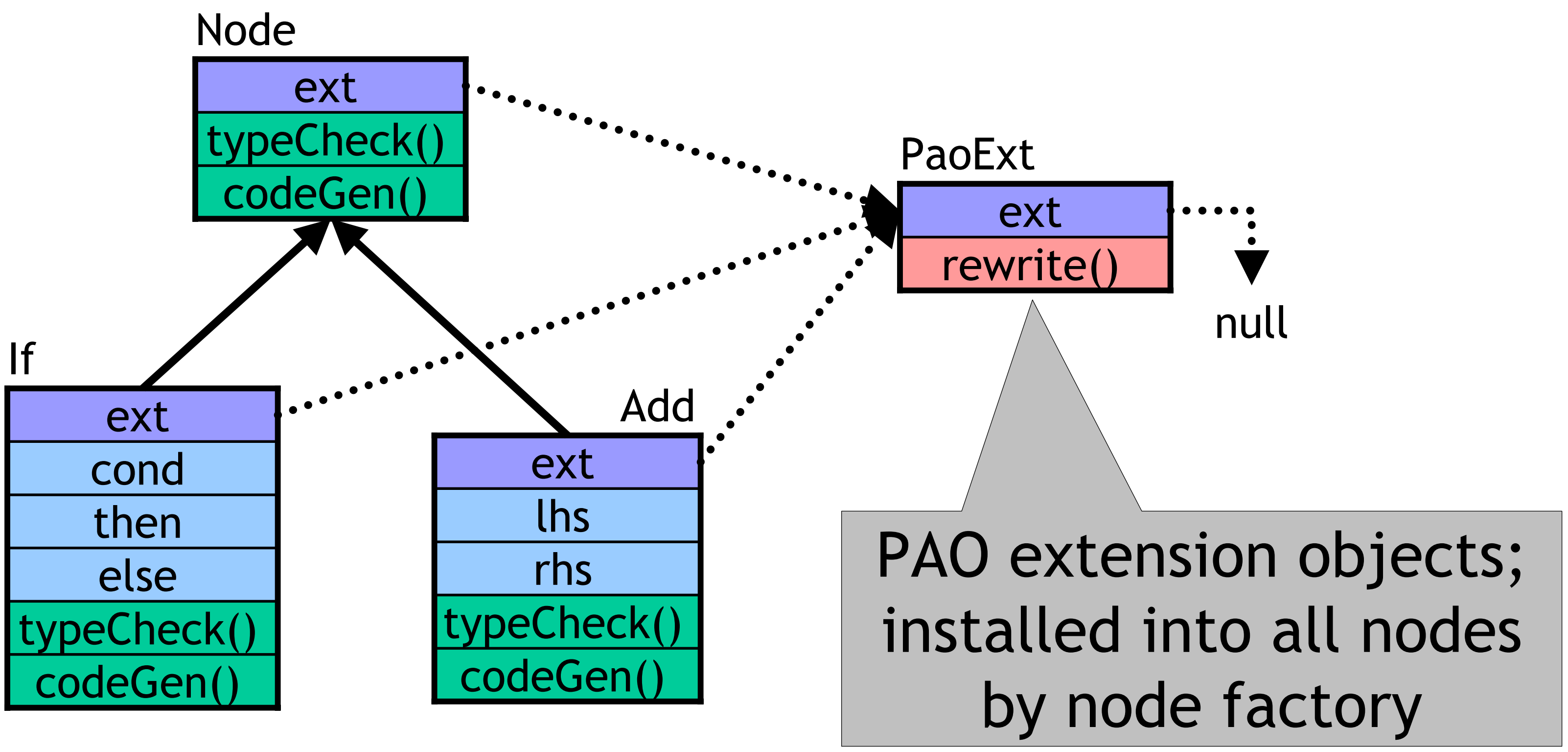


Inheritance is inadequate



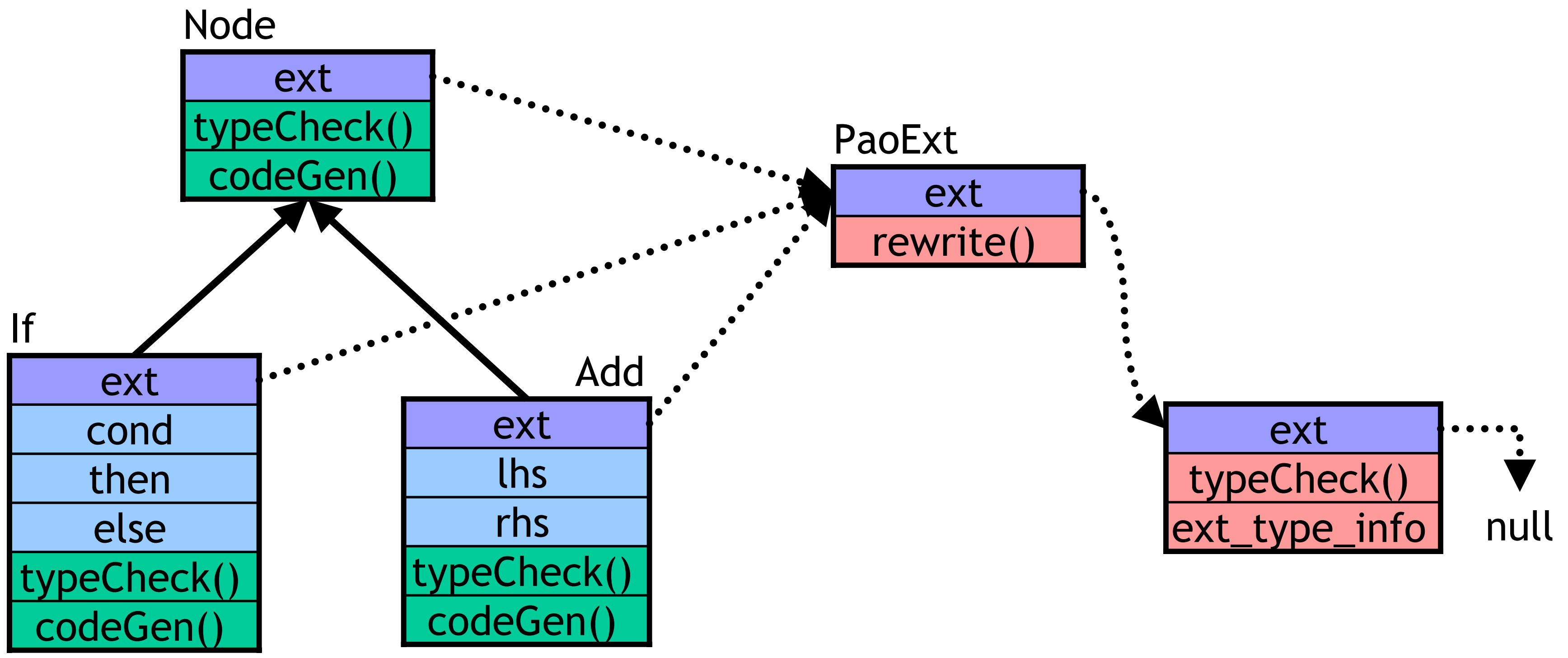
Extension objects

Use composition to mixin methods and fields into AST node classes



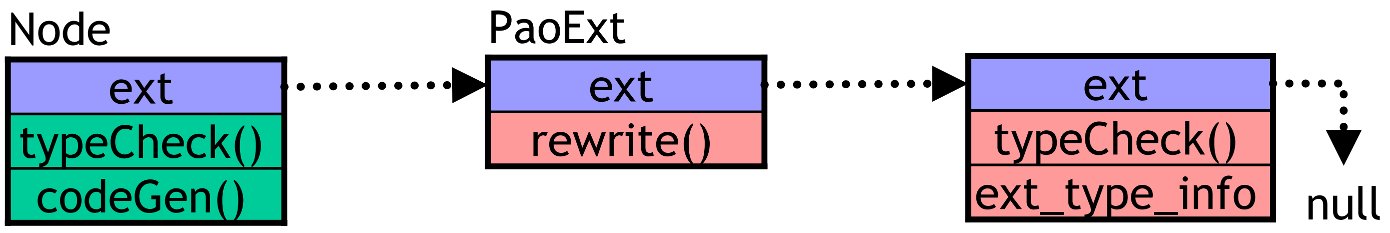
Extension objects

Extension objects have their own ext field to leave extension open



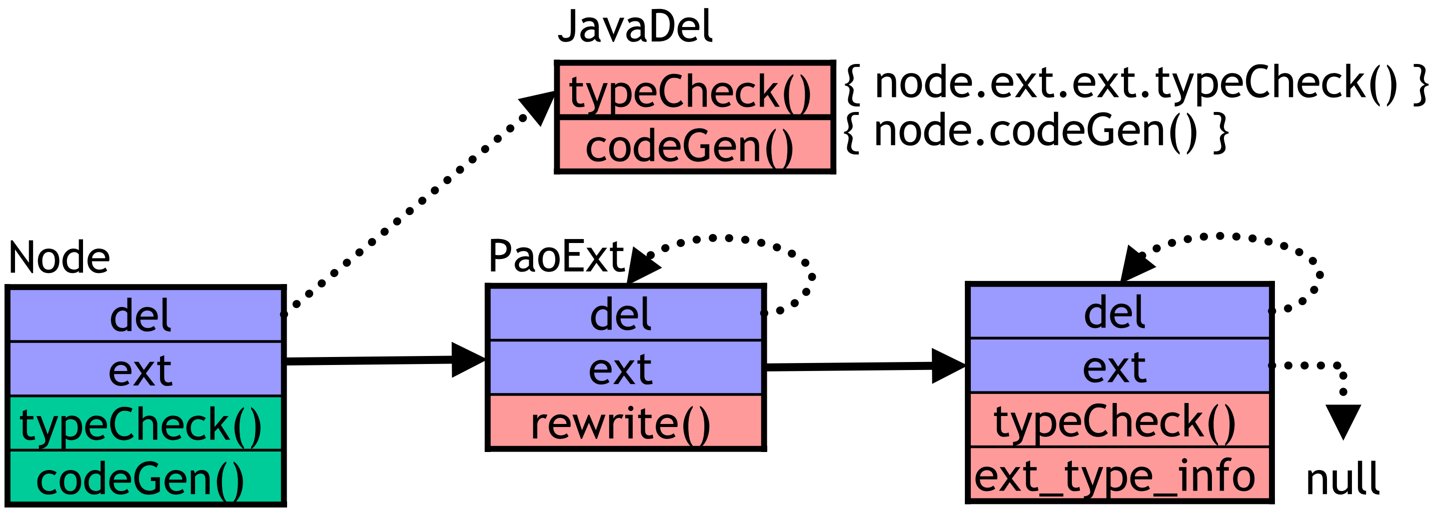
Method invocation

- A method may be implemented in the node or in any one of several extension objects.
- Extension should call `node.ext.ext.typeCheck()`
- Base compiler should call: `node.typeCheck()`
- Cannot hardcode the calls



Delegate objects

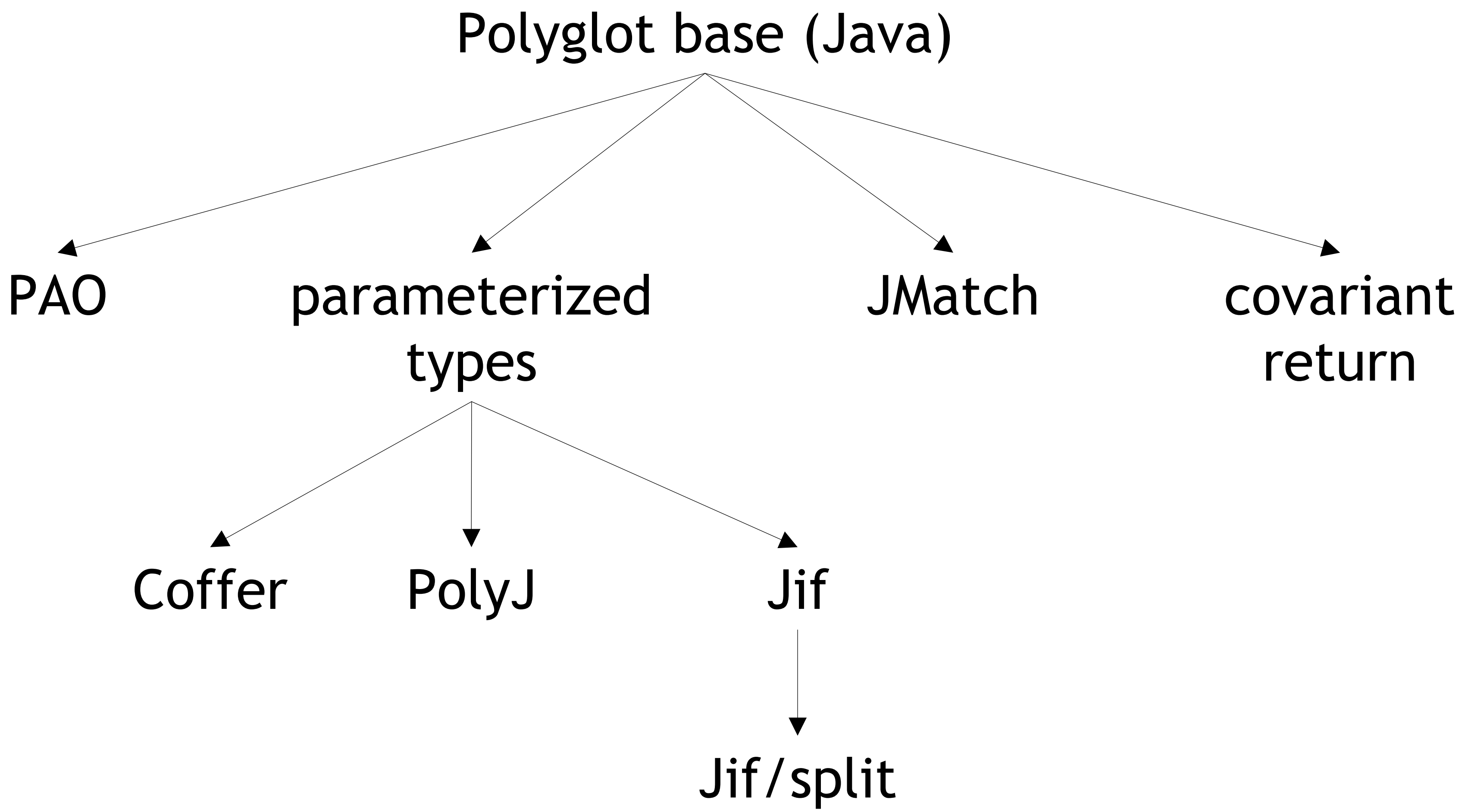
- Each node & extension object has a del field
- **Delegate object** implements same interface as node or ext
- Directs call to appropriate method implementation
  - Ex: node.del.typeCheck()
  - Ex: node.ext.del.rewrite()
- Run-time overhead < 2%



Scalable extensibility

- To add a new pass:
  - Use an extension object to mixin default implementation of the pass for the `Node` base class
  - Use extension objects to mixin specialized implementations as needed
- To change the implementation of an existing pass
  - Use delegate object to redirect to method providing new implementation
- To create an AST node type:
  - Create a new subclass of `Node`
  - Or, mixin new fields to existing node using an extension object

Polyglot family tree



Results

- Can build small extensions in hours or days
- 10% of base code is interfaces and factories

Extension	# Tokens	% of Base
Polyglot base (Java)	166K	100
Jif	129K	78
JMatch	108K	65
Jif/split	99K	60
PolyJ	79K	48
Coffer	24K	14
PAO	6.1K	3.6
parameterized types	3.2K	2
covariant return	1.6K	1
javac 1.1	132K	80

Related work

- Other extensible compilers
  - e.g., CoSy, SUIF
  - e.g., JastAdd, JaCo
- Macros
  - e.g., EPP, Java Syntax Extender, Jakarta
  - e.g., Maya
- Visitors
  - e.g., staggered visitors, extensible visitors

Conclusions

- Several Java extensions have been implemented with Polyglot
- Programmer effort scales well with size of difference with Java
- Extension objects and delegate objects provide **scalable extensibility**
- Download from:

  http://www.cs.cornell.edu/projects/polyglot

Acknowledgments

Brandon Bray	JMatch
Michael Brukman	PPG
Steve Chong	Jif, Jif/split, covariant return
Matt Harren	JMatch
Aleksey Kliger	JLtools, PolyJ
Jed Liu	JMatch
Naveen Sastry	JLtools
Dan Spoonhower	JLtools
Steve Zdancewic	Jif, Jif/split
Lantian Zheng	Jif, Jif/split

http://www.cs.cornell.edu/projects/polyglot

Questions?