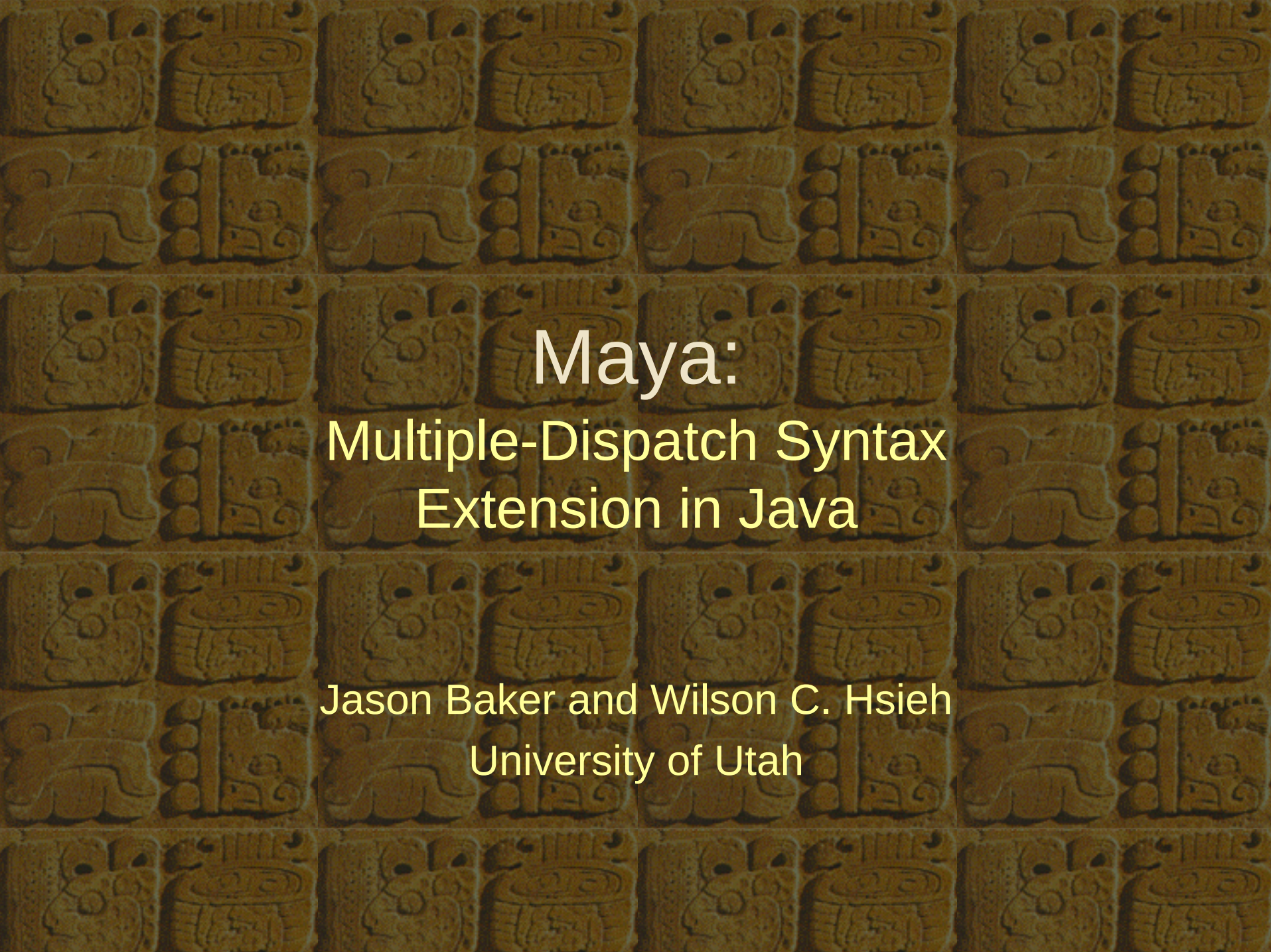


Maya:
Multiple-Dispatch Syntax Extension in Java

Jason Baker and Wilson C. Hsieh
University of Utah

Example

```
java.util.Vector v;
v.elements().foreach(String st) { /*…*/ }
```

Maya code

```
for (Enumeration enum$ = v.elements();
     enum$.hasMoreElements(); ) {
  String st;
  st = (String) enum$.nextElement();
  /* … */
}
```

Specialized Expansion

```
javamaya.util.Vector v;
v.elements().foreach(String st) { /*…*/ }
```

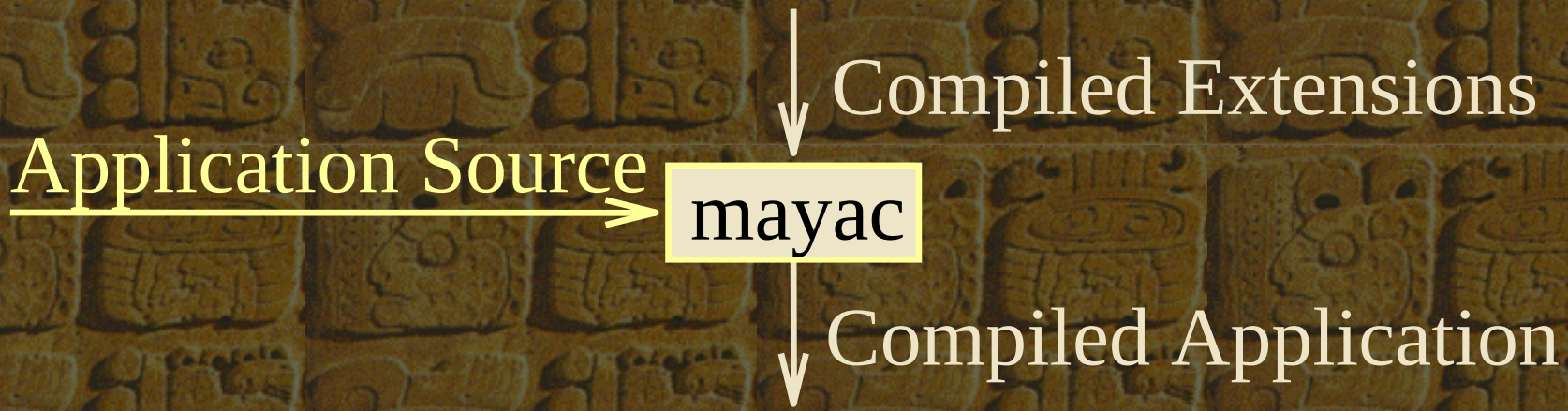
other Maya code

```
{ Vector v$ = v;
  int len$ = v$.size();
  Object[] arr$ = v$.getElementData();
  for (int i$ = 0; i$ < len$; i$++) {
    String st = (String) arr$[i$];
    /* … */ }}
```

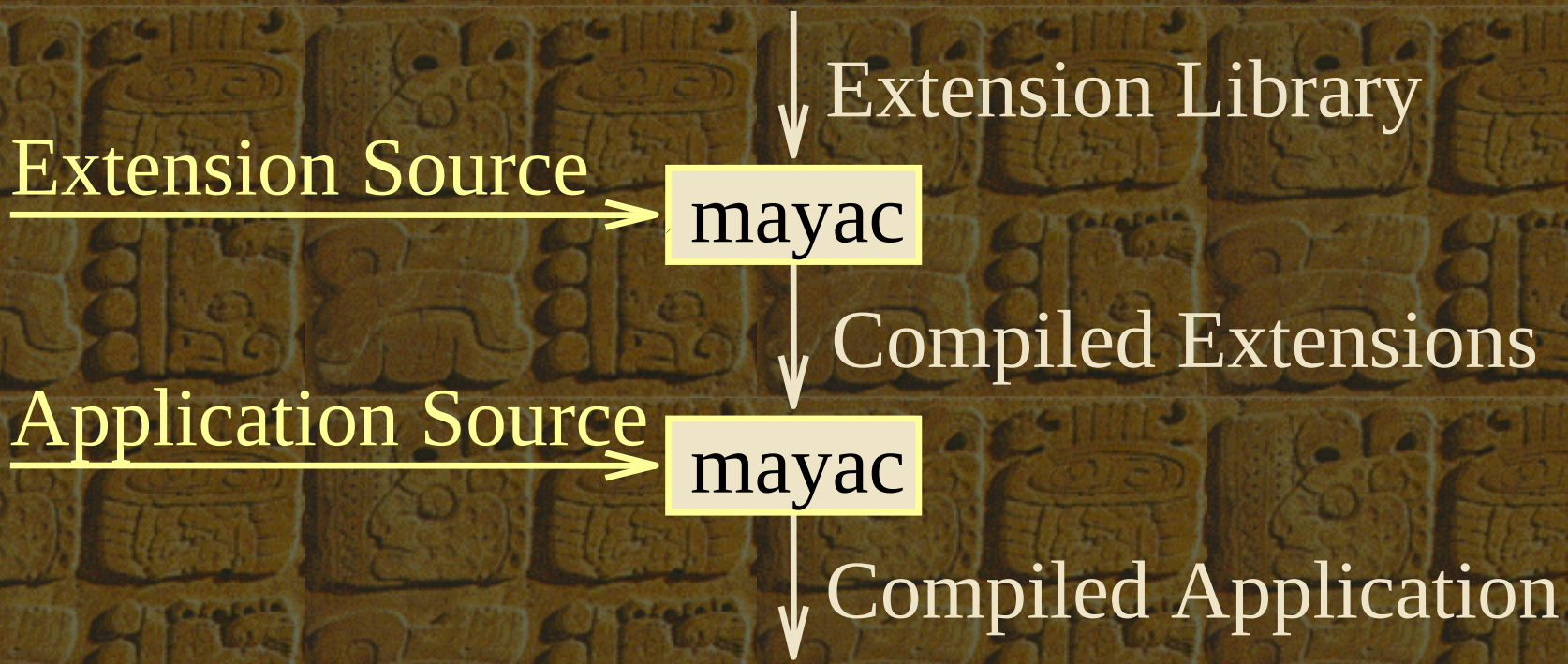
Maya: Java With Macros

- Programs that transform syntax
- Best chosen for each occurrence
  - based on syntactic structure,
  - and static type information

Maya's Extensible Compiler



Maya's Extensible Compiler



Extensible Parser

```
S → S + T    { new S($0, $2) }

T → Lit      { $0 }
```

- Provide a fixed hierarchy of AST node types
- And a grammar that uses these types as symbols

Adding Productions

```
S → S + T    { new S($0, $2) }

T → Lit      { $0 }
S → inc(S)   { new S{ 1 + $2 } }
```

- Allow grammar to be extended
- New actions defined in terms of existing node types

Overriding Actions

```
S → S + T    { new S($0, $2) }
             { new S{ $0.concat($2) } }
T → Lit      { $0 }
S → inc(S)   { new S{ 1 + $2 } }
```

- Several Mayans defined on a single production
- Dispatch to best match for nodes on the parse stack

Key Features

- Hygienically operate on ASTs
- Generate ASTs through robust and flexible template mechanism
- Dispatched on rich set of specializers
  - In particular, static types of expressions

Structuring Extensions

- Compiler loads Mayans by calling an interface method
- Mayans, like class declarations, may be local
- Mayans may be loaded locally to a lexical scope

Parse Order

```
maya.util.Vector v;
v.elements().foreach(String st) { /*…*/ }
```

- Mayans execute during parsing
- Mayans are dispatched based on static types of arguments
- Some arguments can only be checked after expansion
- Lazy parsing made explicit in grammar

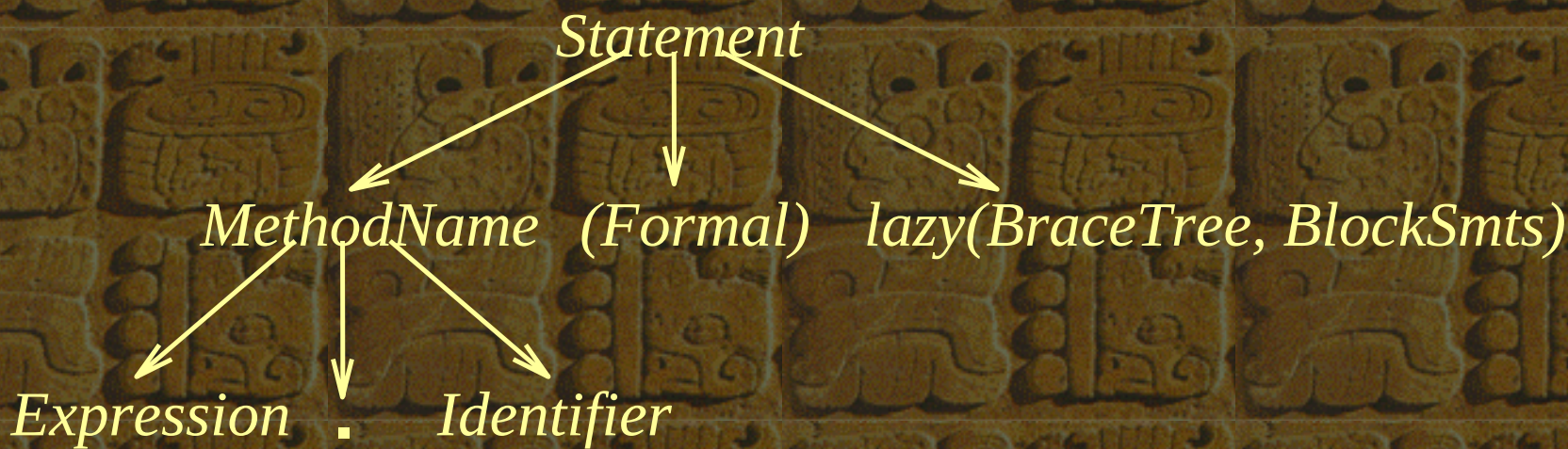
Defining a Production

- syntax for
  `v.elements().foreach(String st){…}`
- *Statement* → *MethodName* **(** *Formal* **)** *LazyBlock*
- In Maya syntax:

```
abstract Statement
syntax(MethodName(Formal)
       lazy(BraceTree, BlockStmts));
```

Declaring a Mayan

```
Statement syntax
EForEach(Expression:Enumeration enumExp
         . foreach(Formal var)
         lazy(BraceTree, BlockStmts) body))
{ /* Mayan body */ }
```



Parameter Specializers

- Runtime types accepted by a Mayan
- Determine dispatch order
- Maya support other specializers
  - substructure
  - static type of expression
  - token value
  - type denoted by name

Parameter Specializers

```
Statement syntax
EForEach(Expression:Enumeration enumExp
         . foreach(Formal var)
         lazy(BraceTree, BlockStmts) body))
{ /* Mayan body */ }
```

- substructure
- static type of expression
- token value
- type denoted by name

Parameter Specializers

```
Statement syntax
EForEach(Expression:Enumeration enumExp
         . foreach(Formal var)
         lazy(BraceTree, BlockStmts) body))
{ /* Mayan body */ }
```

- substructure
- static type of expression
- token value
- type denoted by name

Parameter Specializers

```
Statement syntax
EForEach(Expression:Enumeration enumExp
         . foreach(Formal var)
         lazy(BraceTree, BlockStmts) body))
{ /* Mayan body */ }
```

- substructure
- static type of expression
- token value
- type denoted by name

Parameter Specializers

```
Statement syntax
EForEach(Expression:Enumeration enumExp
         . foreach(Formal var)
         lazy(BraceTree, BlockStmts) body))
{ /* Mayan body */ }
```

- substructure
- static type of expression
- token value
- type denoted by name

Templates and Hygiene

- Easy way to construct syntax trees:

```
new AddExpr(
  new IntegerLiteral(1),
  new MulExpr(/* ... */))
```

vs.

```
new Expression {
  1 + 2 * 3
}
```

- Statically ensure
  - syntactic correctness
  - hygiene and referential transparency
  - references to bound variables

Template Example

```
Expression enumExp; Formal var; DelayedStmt body;


return new Statement {
  for (Enumeration enum = $_______;
       enum.hasMoreElements(); ) {
    $_____________________
    $_____________________________
      = ($________) enum.nextElement();
    $____
  }
};
```

Unquoting Parameters

```
Expression enumExp; Formal var; DelayedStmt body;


return new Statement {
  for (Enumeration enum = $enumExp;
       enum.hasMoreElements(); ) {
    $_____________________
    $_____________________________
      = ($________) enum.nextElement();
    $body
  }
};
```

Other Expressions

```
Expression enumExp; Formal var; DelayedStmt body;
final StrictTypeName castType
  = StrictTypeName.make(var.getType());
return new Statement {
  for (Enumeration enum = $enumExp;
       enum.hasMoreElements(); ) {
    $(DeclStmt.make(var))
    $(Reference.makeExpr(var))
      = ($castType) enum.nextElement();
    $body
  }
};
```

Hygiene

```
Expression enumExp; Formal var; DelayedStmt body;
final StrictTypeName castType
  = StrictTypeName.make(var.getType());
return new Statement {
  for (Enumeration enum = $enumExp;
       enum.hasMoreElements(); ) {
    $(DeclStmt.make(var))
    $(Reference.makeExpr(var))
      = ($castType) enum.nextElement();
    $body
  }
};
```

Pattern Parsing

- Used when compiling Mayan argument trees and template definitions
- Patterns contain both terminal and non-terminal symbols
- Pattern parser generates parse tree with non-terminal leaves
- Used to statically implement hygiene

MultiJava Features

- Clifton et al. [OOPSLA 2000]
- Multiple dispatch
  - static checks for ambiguous and incomplete generic functions
- Open classes
  - external methods can be added to classes
- Implemented in KJC (20k lines)

MultiJava in Maya

- 2500 lines of code
- Maya provides necessary features
  - static type information
  - syntax extension
  - overriding behavior of syntax
- Maya supports large syntax extensions
  - local Mayans
  - lexically scoped Mayan imports

Related Work

- JSE [Bachrach and Playford 2001]
- JTS [Batory et al. 1998]
  - not appropriate for simple macros
  - doesn't do type checking
  - lacks pattern parser
- OpenJava [Tatsubori et al. 2000]
  - limited ability to define syntactic forms
  - single dispatch

Conclusions

- Maya's features provide flexibility
  - static type information
  - flexible templates and pattern matching
  - detection of errors in templates
- Through novel techniques
  - lazy parsing
  - pattern parsing

http://www.cs.utah.edu/~jbaker/maya