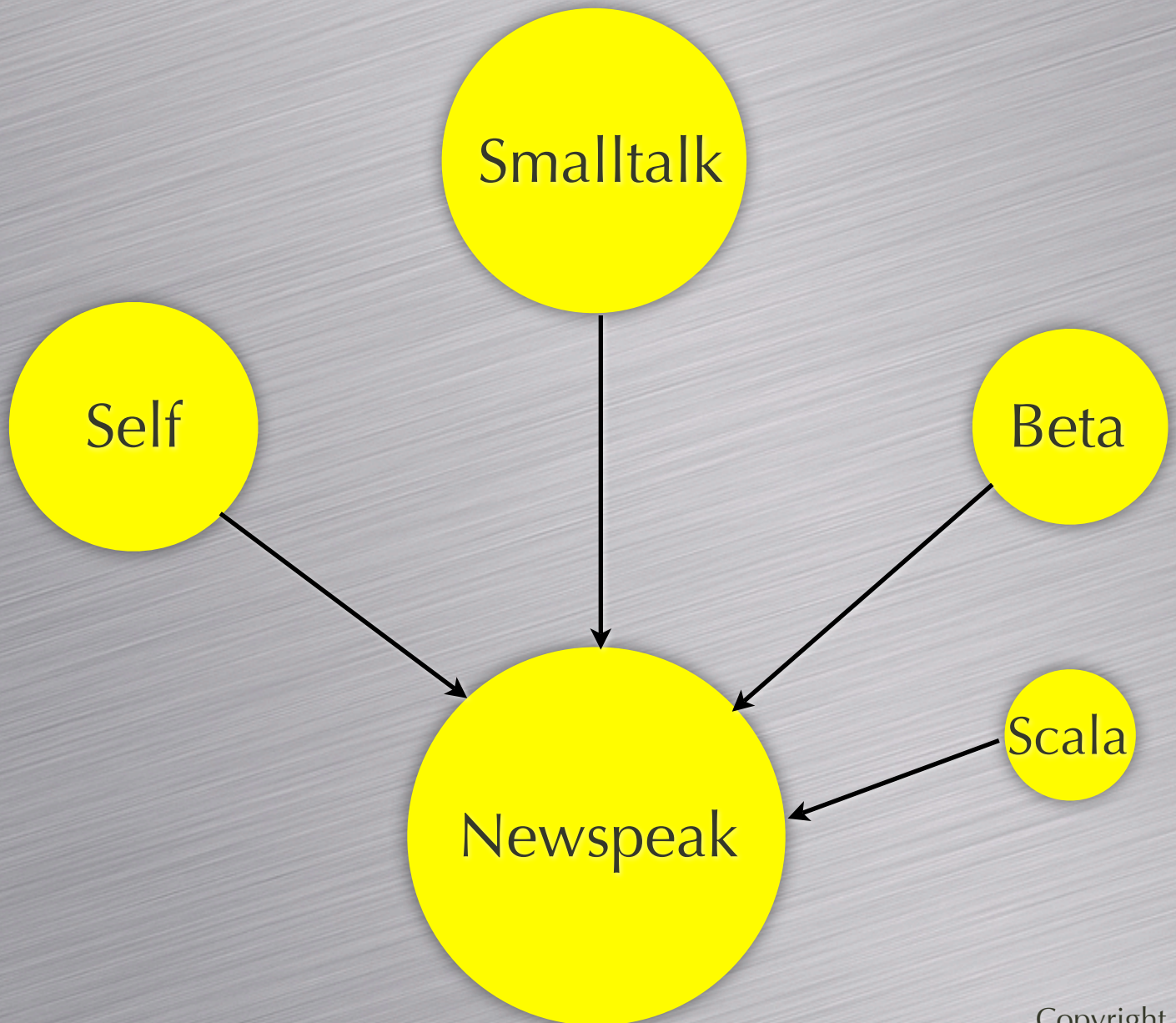
Newspeak

Gilad Bracha

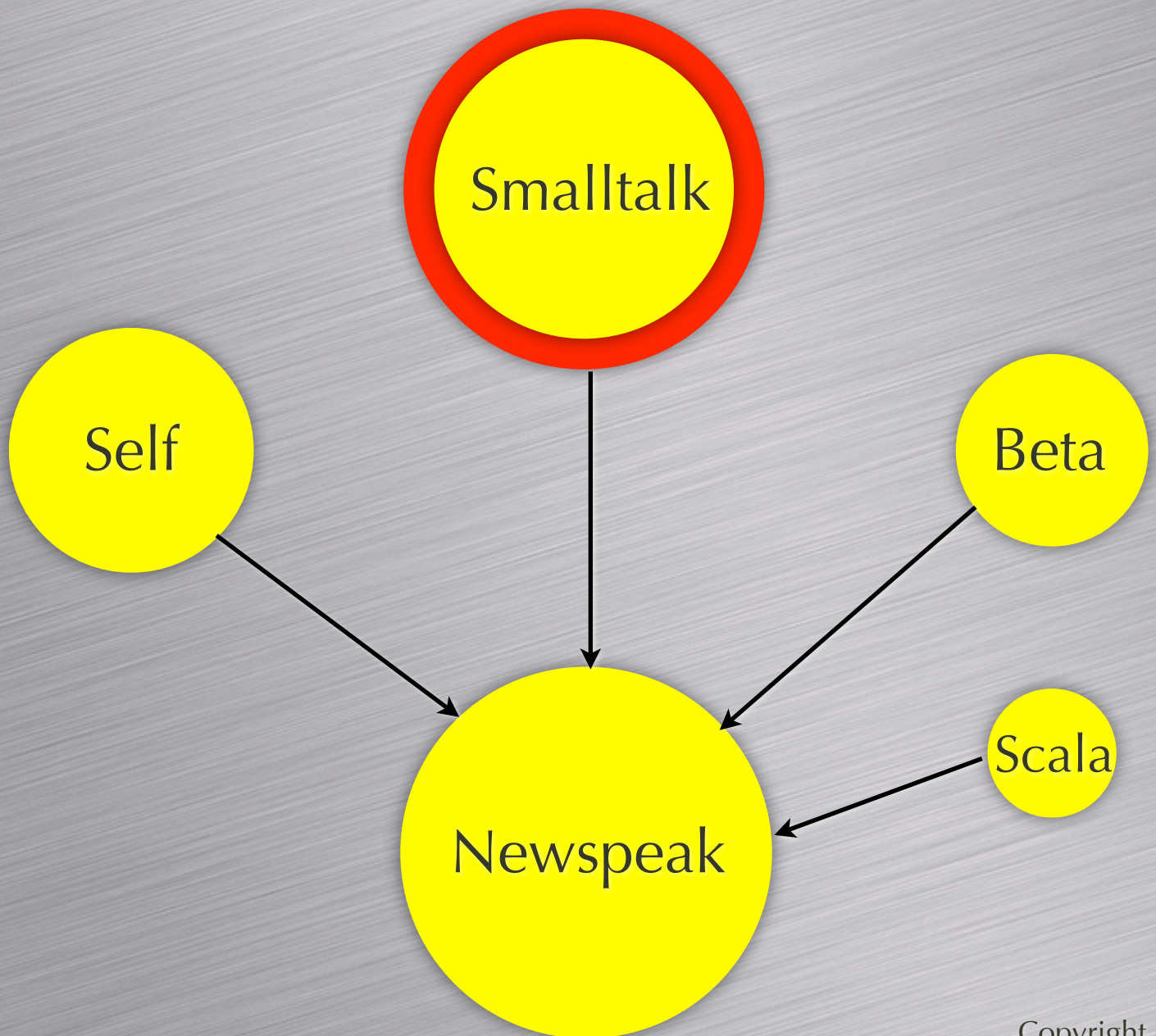
Distinguished Engineer

Cadence Design Systems

A Noble Inheritance



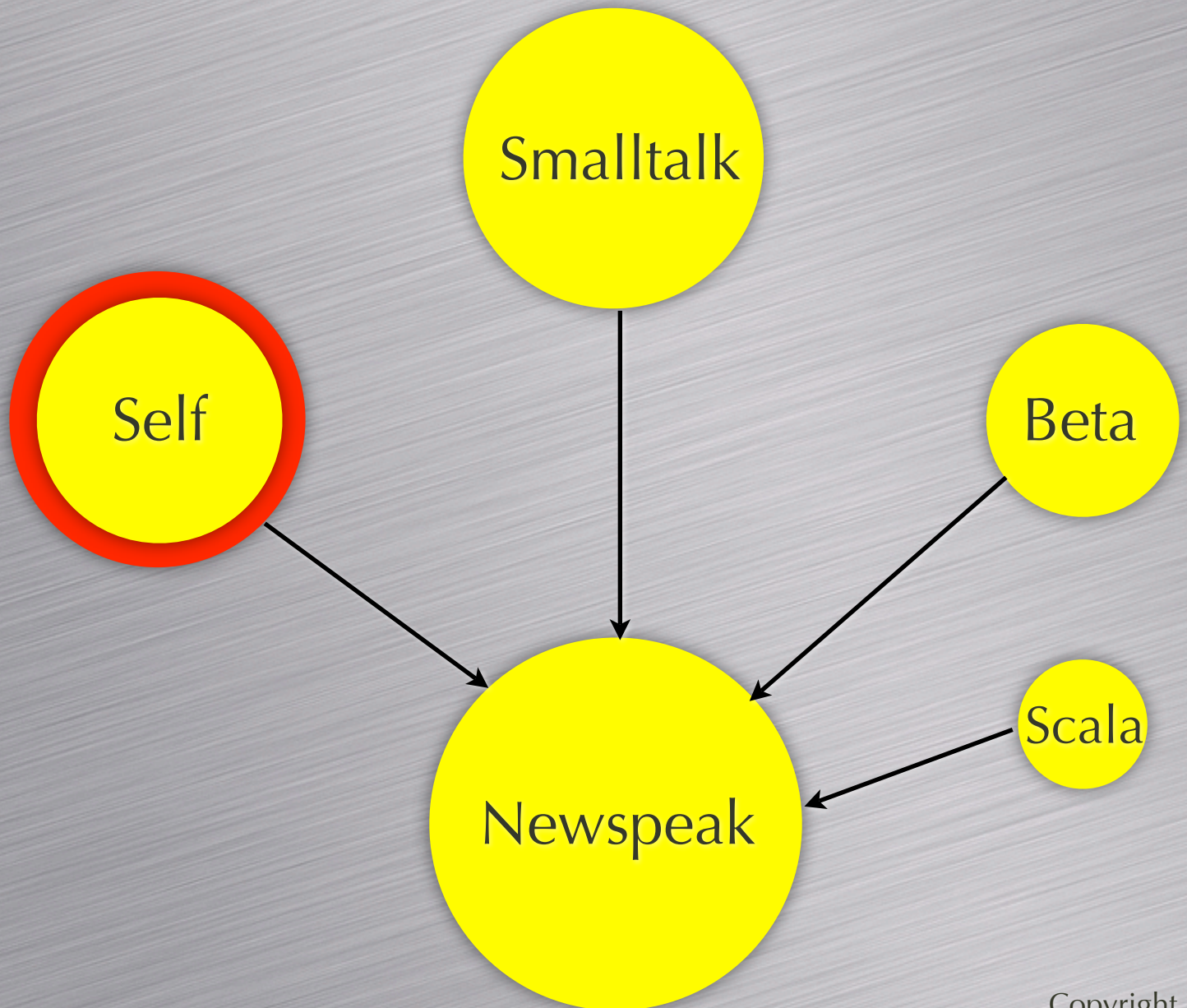
A Noble Inheritance



Influences of Smalltalk

- Class based
- Dynamically typed
- **Reflective**
  - Full hotswapping

A Noble Inheritance



Influences of Self

- **Message-based**
  - Loose coupling, not necessarily asynchrony

Influences of Self

- **Message-based**
  - Loose coupling, not necessarily asynchrony
  - No direct access to variables

Influences of Self

- **Message-based**
  - Loose coupling, not necessarily asynchrony
  - No direct access to variables

id = letter, (letter | digit) star.

Influences of Self

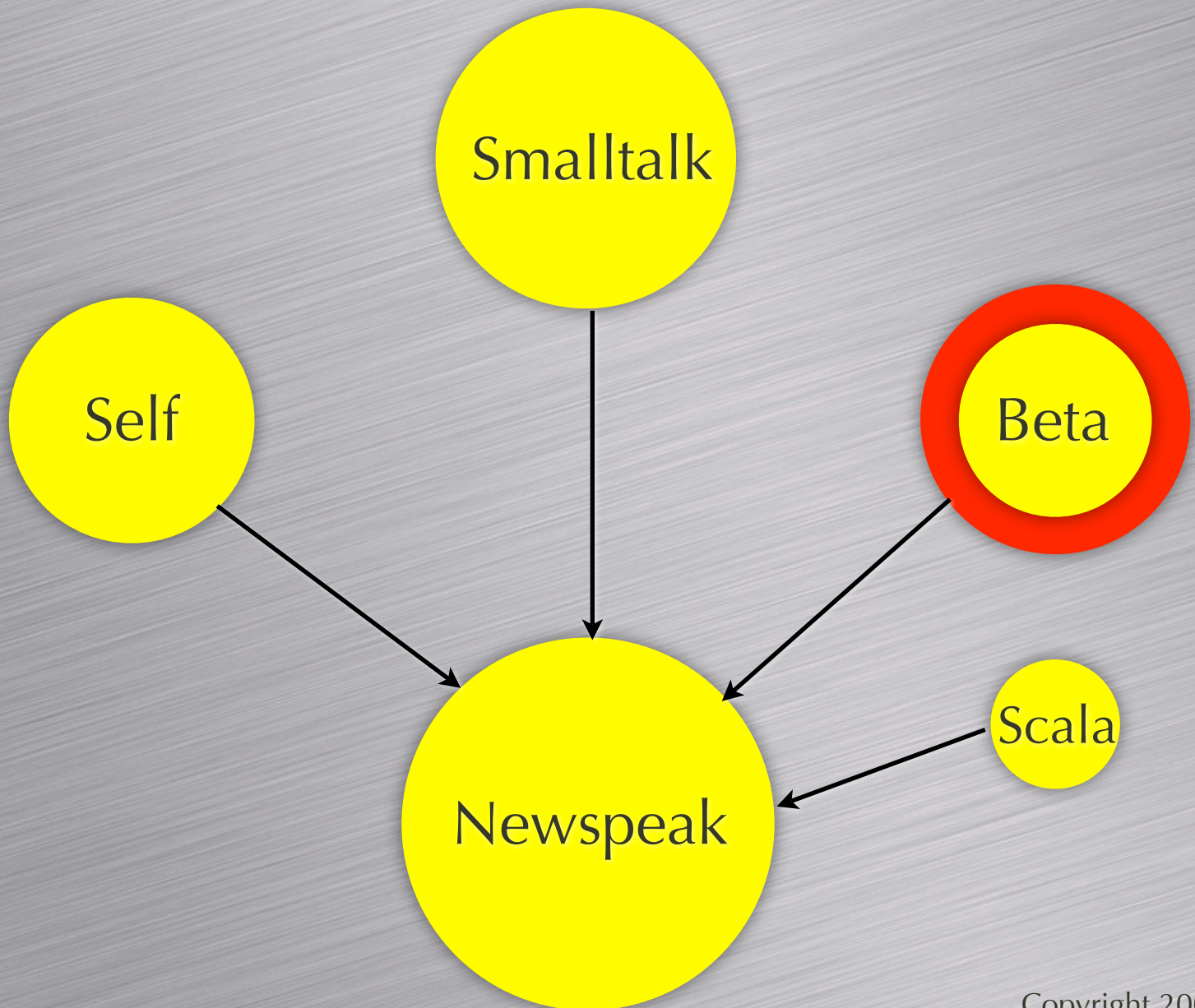
- **Message-based**
  - Loose coupling, not necessarily asynchrony
  - No direct access to variables

id ::= letter, (letter | digit) star.

Influences of Self

- Mirror based reflection

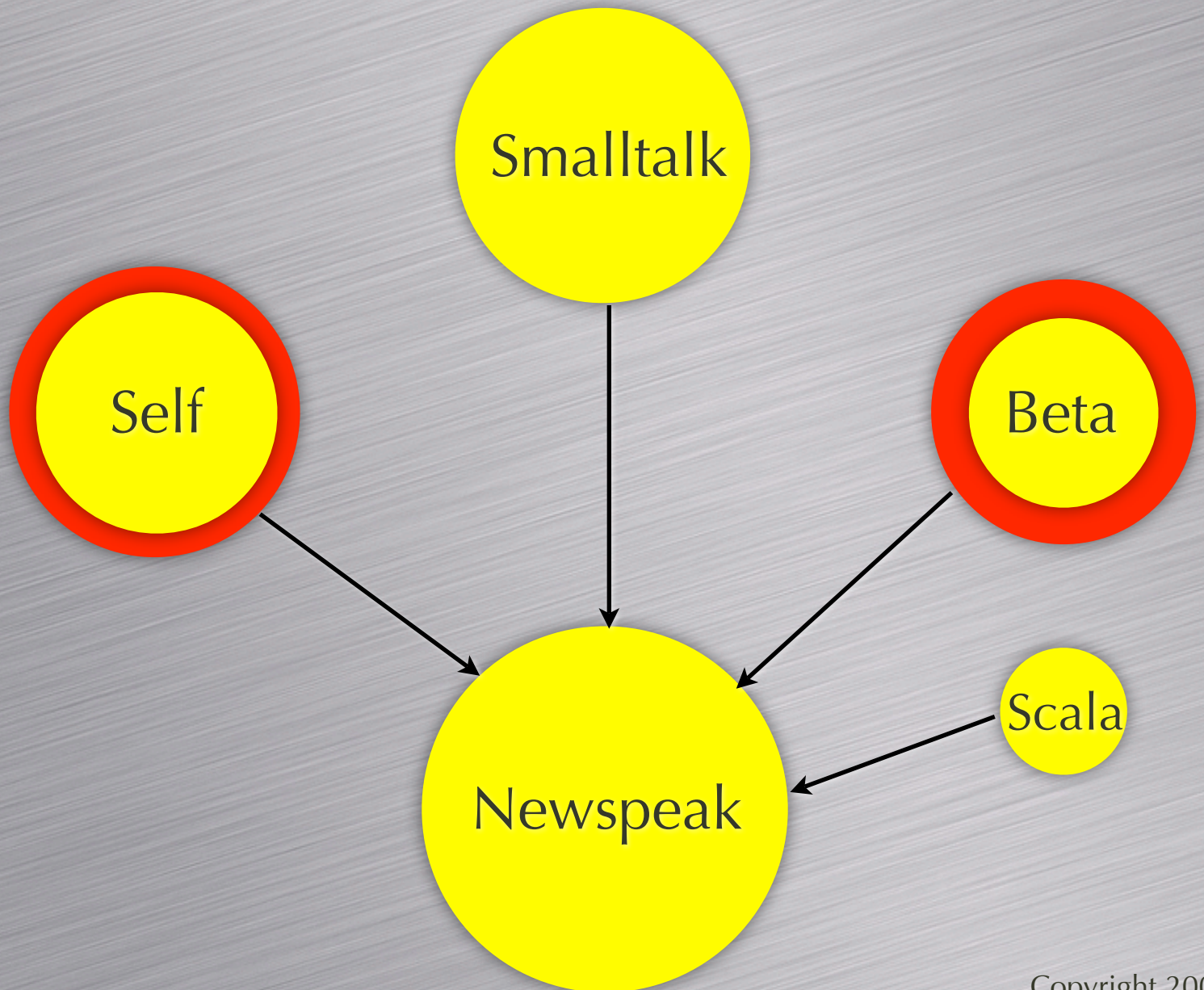
A Noble Inheritance



Influences of Beta

- Nested Classes

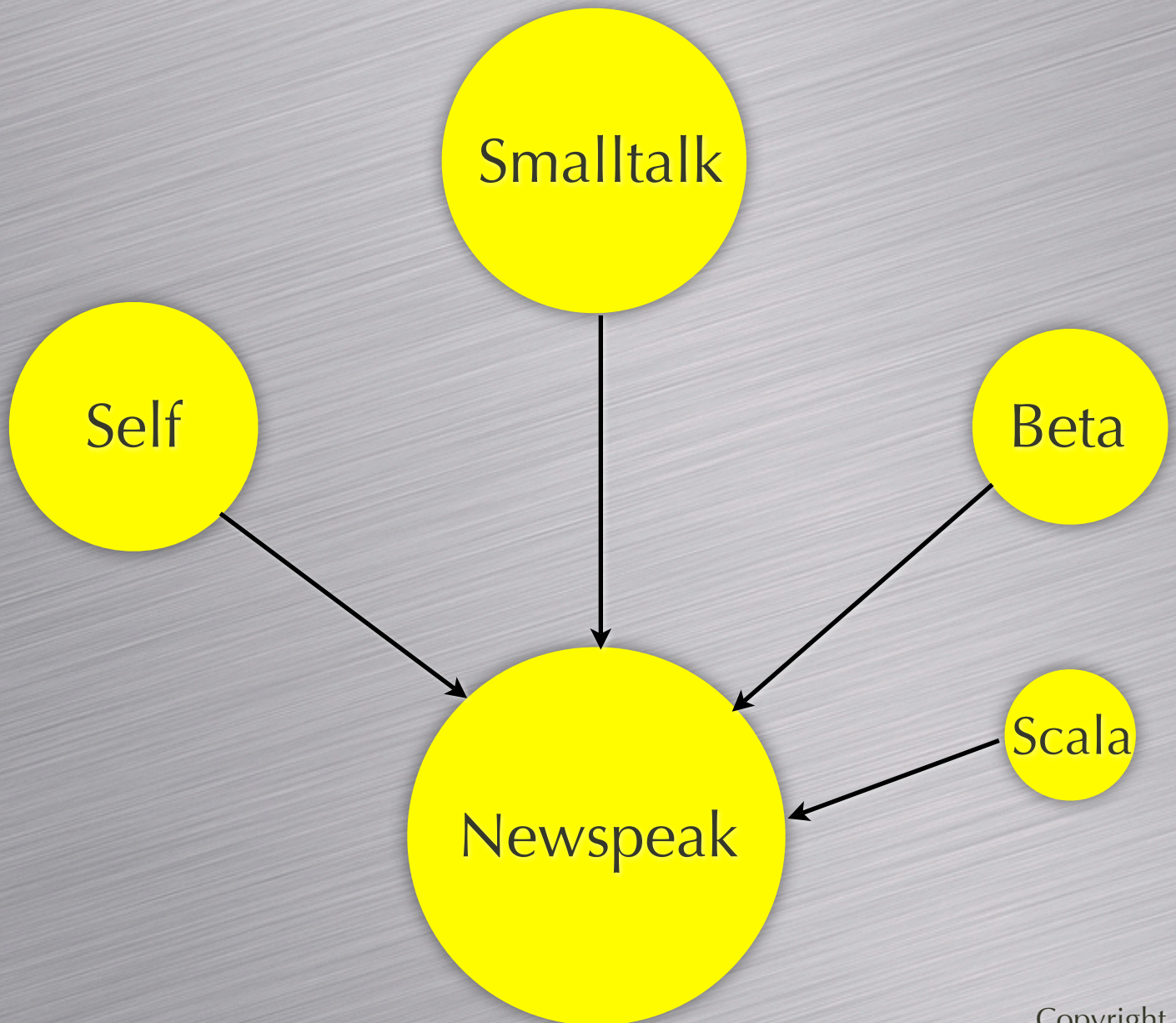
A Noble Inheritance



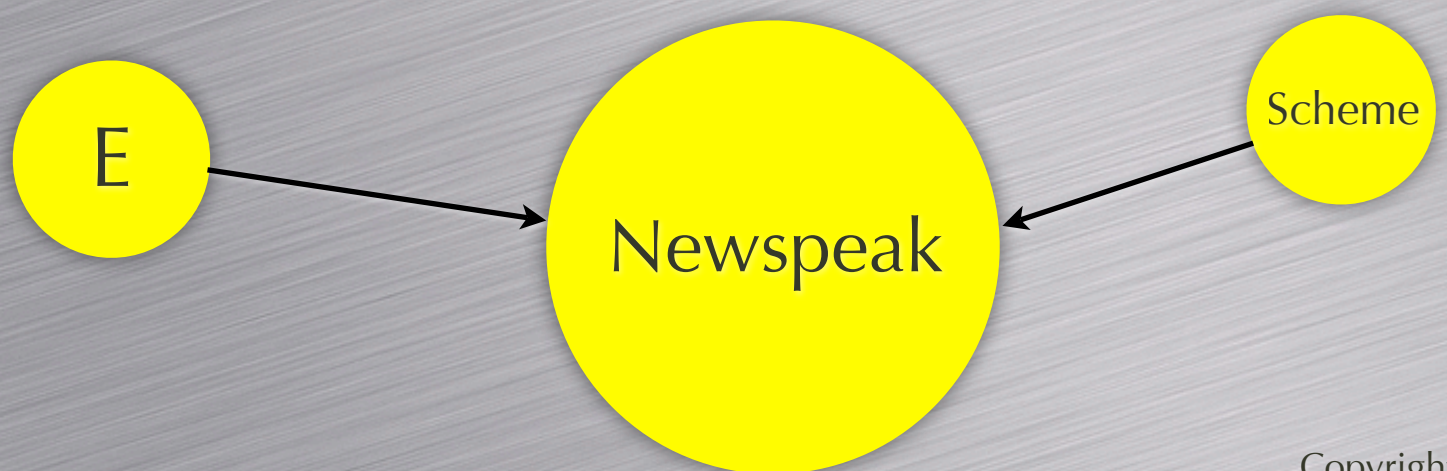
Class Nesting + Message based Programming

- All nested classes are virtual classes
- All classes are mixins
- Class Hierarchy inheritance

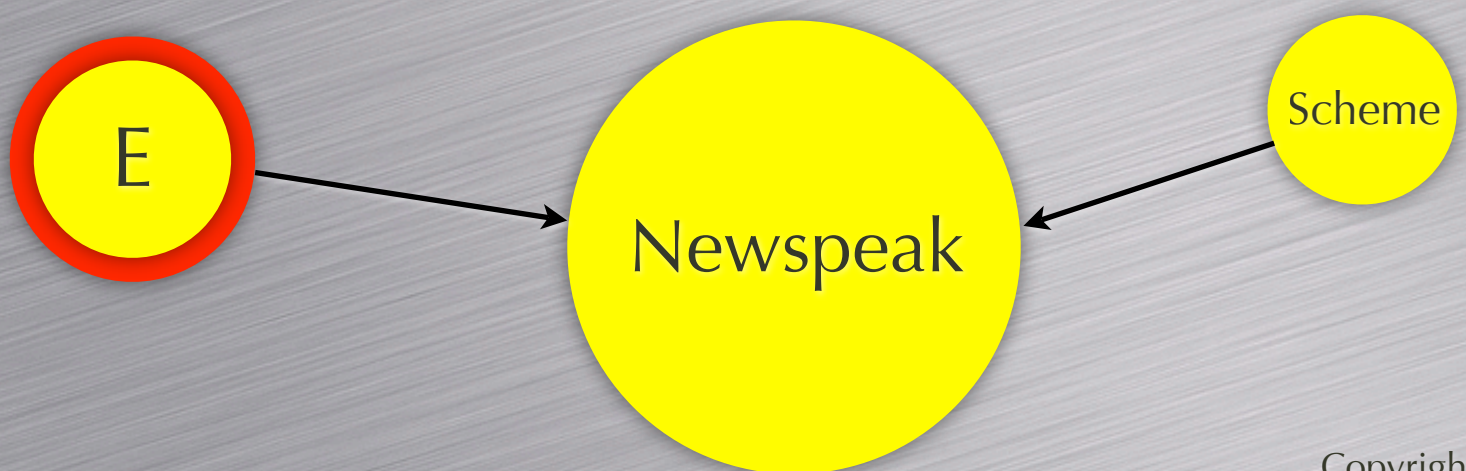
A Noble Inheritance



Commonalities



Commonalities



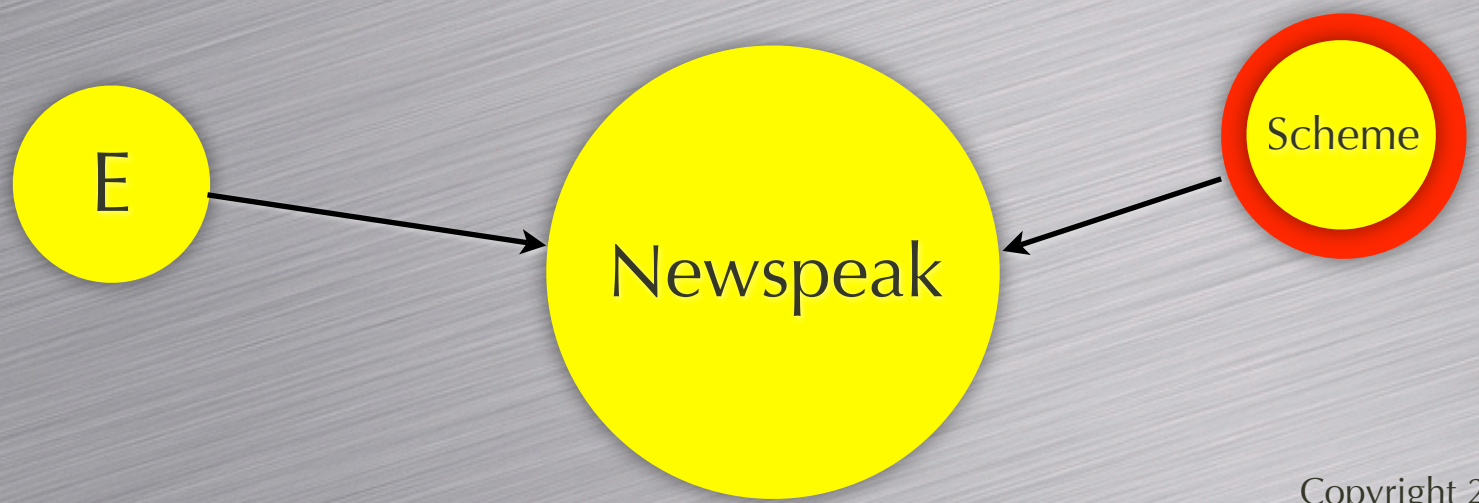
Commonalities with E

- **Security**
  - Object-capability model
    - public, protected and private messages
  - No static state

Uncommonalities with E

- Unlike E, Newspeak is reflective
  - Mirrors act as capabilities for reflection

Commonalities



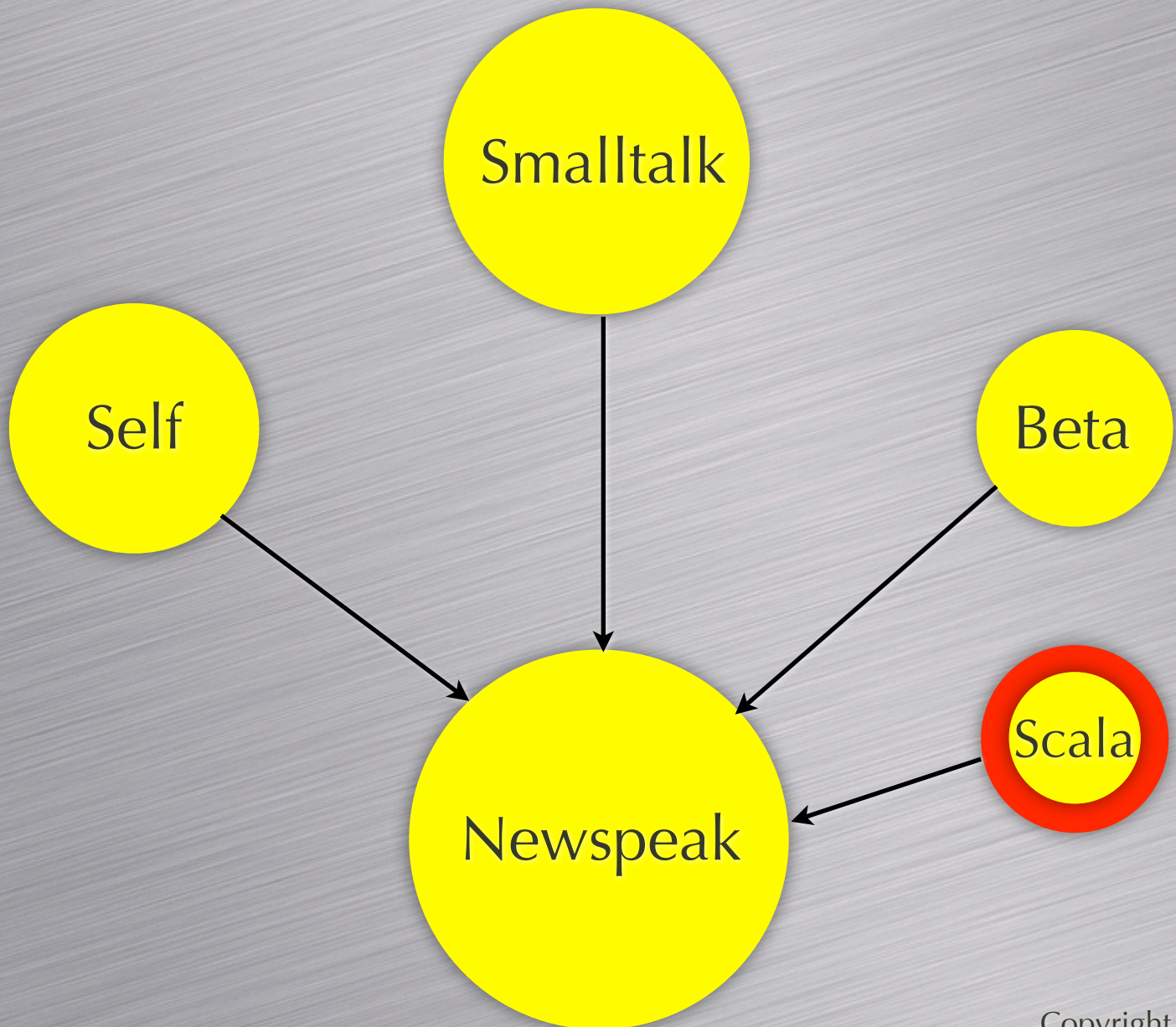
Commonalities with PLT Scheme

- **Modularity**
  - Component style modules

Module Definitions

- Classes not nested within another class
- Deeply Immutable
- Instantiated into stateful objects known as ***modules***
- No access to surrounding namespace
- Constructor parameters are objects/ capabilities that determine per-module sandbox

A Noble Inheritance



Influences of Scala

- Some details of object construction
- Pattern Matching via Extractors?

Parser Combinators

BNF

*id = letter (letter | digit) **

Parser Combinators

BNF

*id = letter (letter | digit) **

Newspeak

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

How it Works

id = letter, (letter | digit) star.

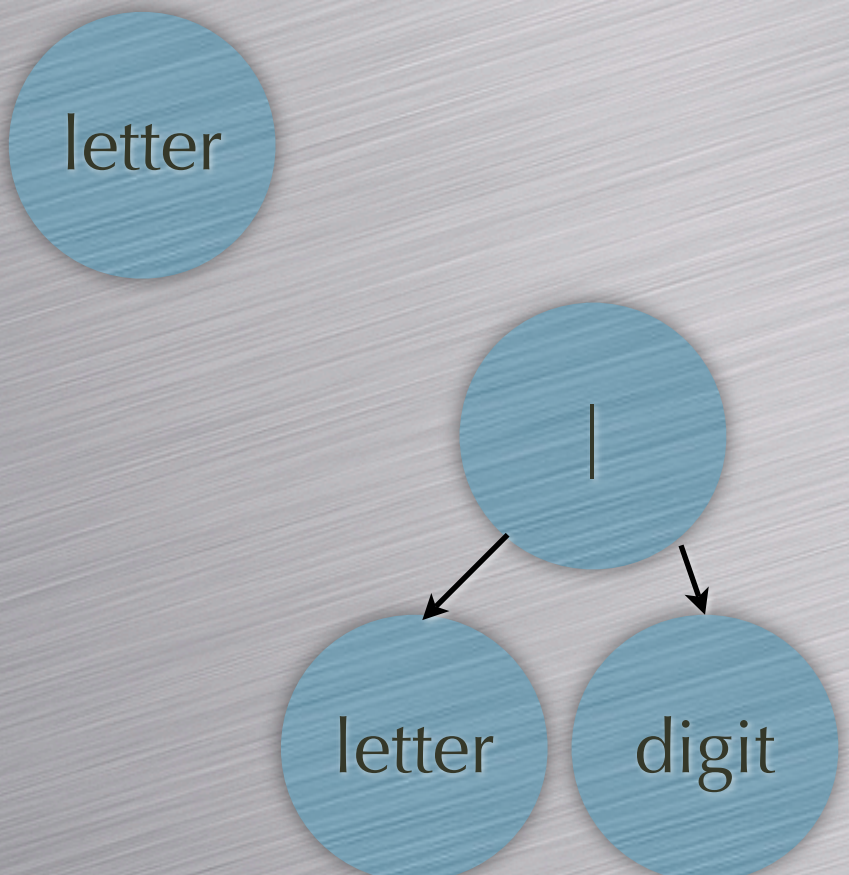
How it Works

id = letter, (letter | digit) star.

How it Works

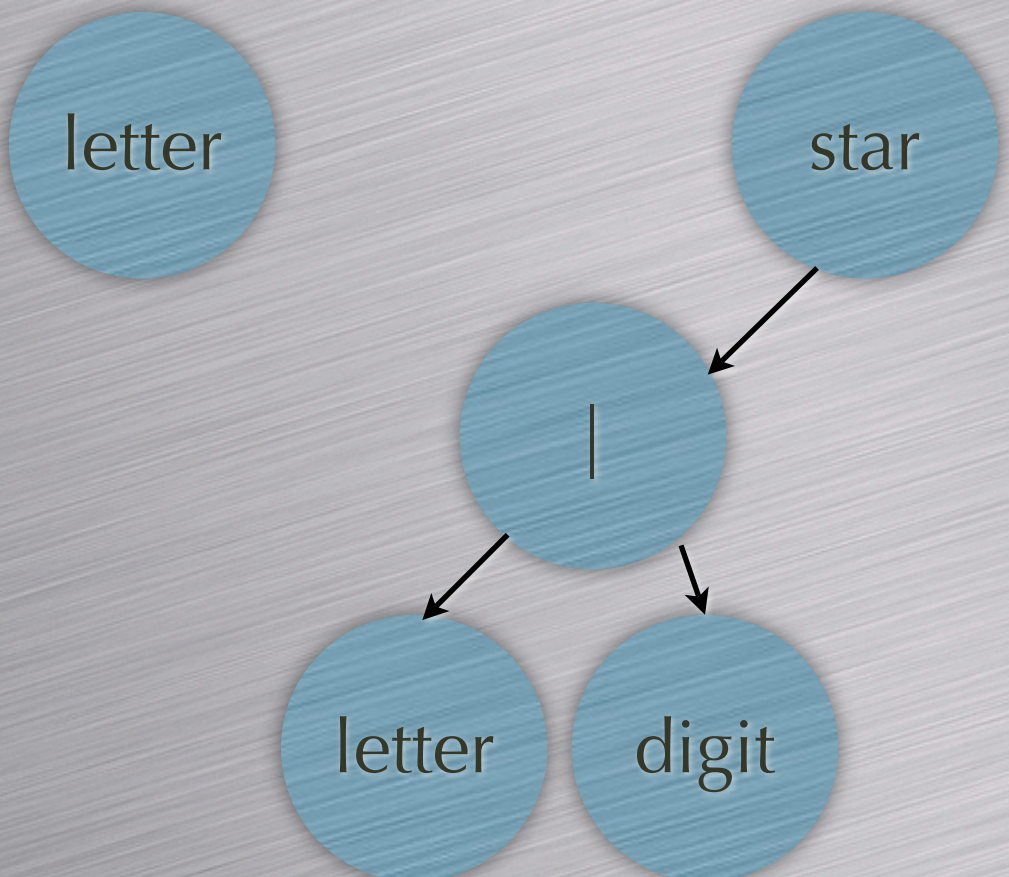
id = letter, (letter | digit) star.

How it Works



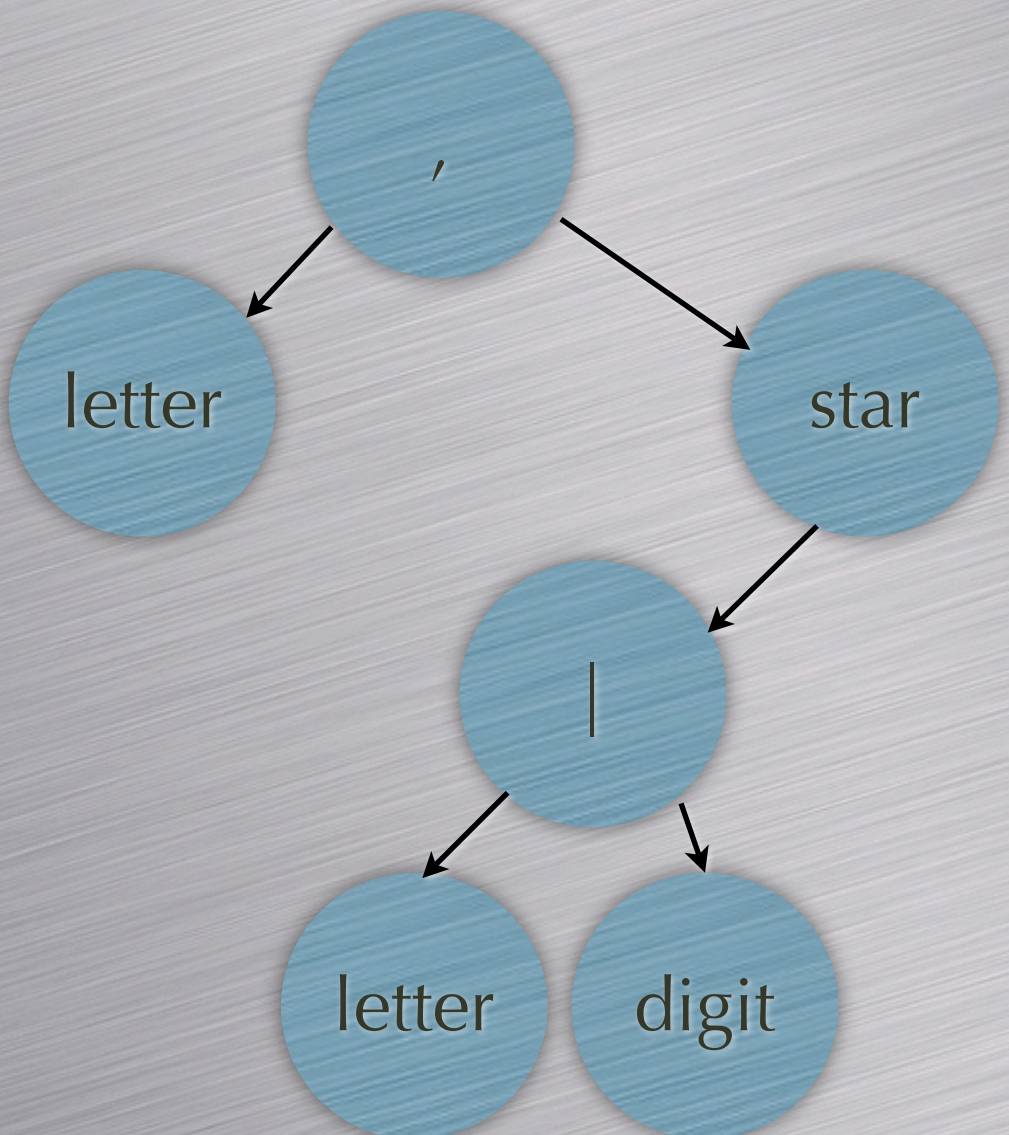
id = letter, (letter | digit) star.

How it Works



id = letter, (letter | digit) star.

How it Works



id = letter, (letter | digit) star.

Why is this Pretty?

id = letter, (letter | digit) star.

Why is this Ugly?

```
id = letter().seq(letter().or(digit()).star());
```

Why is this Ugly?

id = letter().seq(letter().or(digit()).star());

vs.

*id = letter (letter | digit) **

vs.

id = letter, (letter | digit) star.

Why is this Ugly?

```
id = letter().seq(letter().or(digit()).star());
```

vs.

```
id = letter (letter | digit) *
```

vs.

```
id = letter, (letter | digit) star.
```

Why is it Ugly?

A programming language is low level when its programs require attention to the irrelevant

- Alan Perlis

A Complete Grammar

```
class ExampleGrammar1 = ExecutableGrammar (
|
digit = charBetween: $0 and: $9.
letter = (charBetween: $a and:$z) |
         (charBetween: $A and:$Z).
id = letter, (letter | digit) star.
identifier = tokenFor: id.
hat = tokenFromChar: $^.
expression = identifier.
returnStatement = hat, expression.
|
)()
```

Building an AST

```
returnStatement = hat, expression
                wrapper:[:r : e |
                        ReturnStatAST on: e
                ]
```

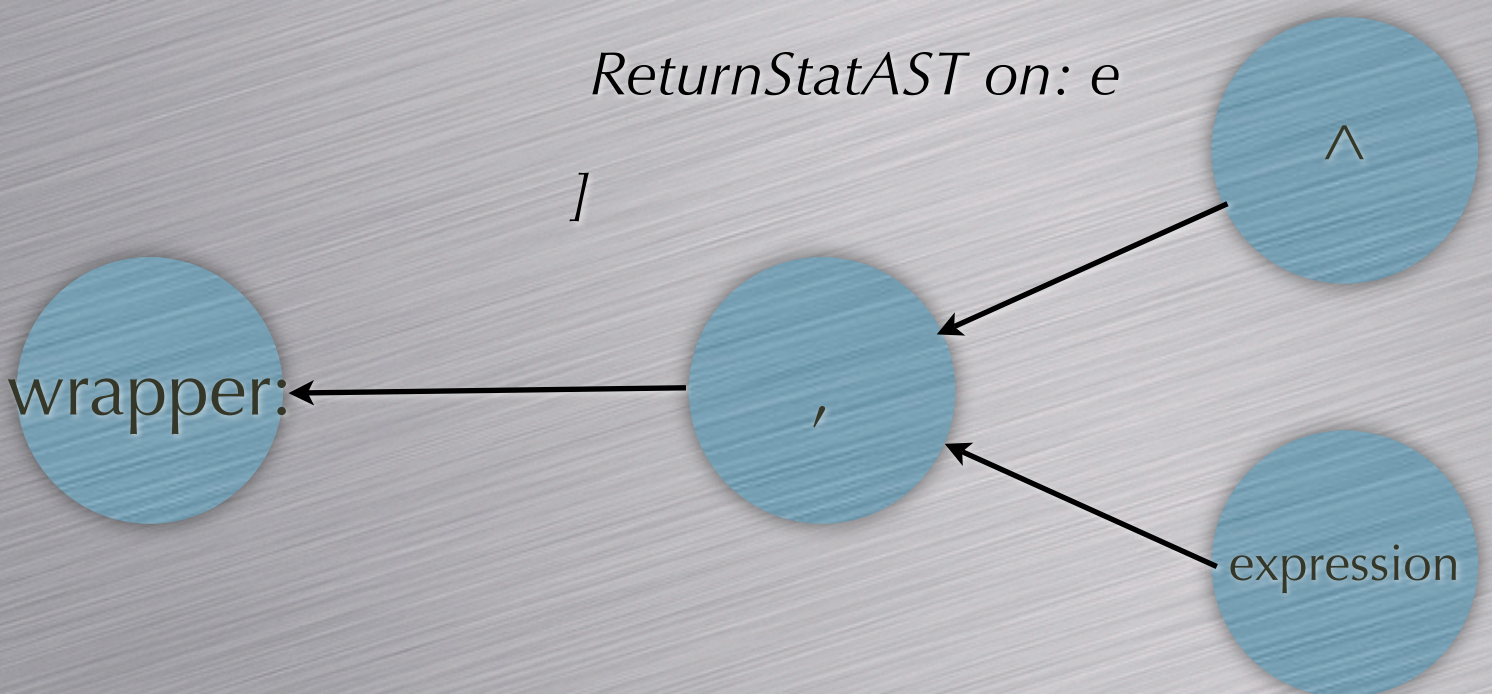
Building an AST

```
returnStatement = hat, expression

    wrapper:[:r : e |

        ReturnStatAST on: e

    ]
```



Grammar

```
returnStatement = hat, expression

        wrapper:[:r : e |

                ReturnStatAST on: e

        ]
```

Semantic action

```
returnStatement = hat, expression
                    wrapper:[:r : e |
                                ReturnStatAST on: e
                    ]
```

Factor out Grammar

Superclass (pure grammar specification):

returnStatement = hat, expression

Subclass (AST builder):

```
returnStatement = (

        super returnStatement

            wrapper:[:r : e |

                ReturnStatAST on: e

            ] )
```

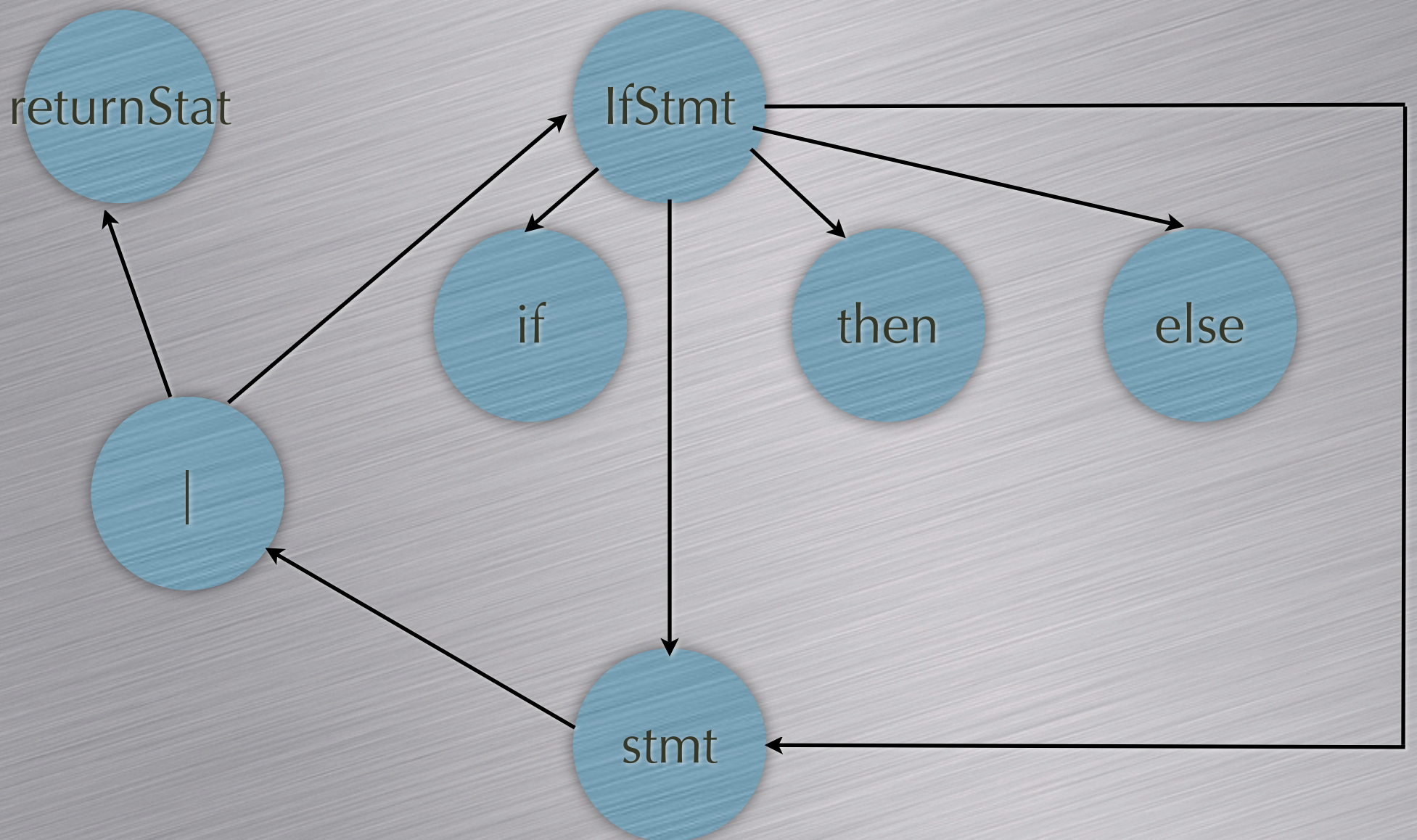
Modular Parser

```
class ExampleParser1= ExampleGrammar1 () (
  id = (
    ^super id
      wrapper:[:fst :snd | fst asString, (String withAll: snd)]
  )
  identifier = (
    ^super identifier
      wrapper:[:v |VariableAST new name: v token;
                                  start: v start; end: v end].
  )
  returnStatement = (
    ^super returnStatement
      wrapper:[:r :e | ReturnStatAST new expr:e;
                                  start: r start; end: e end].
)
```

Extending a Grammar

```
class ExampleGrammar2 = ExampleGrammar1 (
|
if = tokenFromSymbol:#if.
then = tokenFromSymbol:#then.
else = tokenFromSymbol:#else.
ifStatement = if, expression, then, statement, else, statement.
statement = ifStatement | returnStatement.
|
)()
```

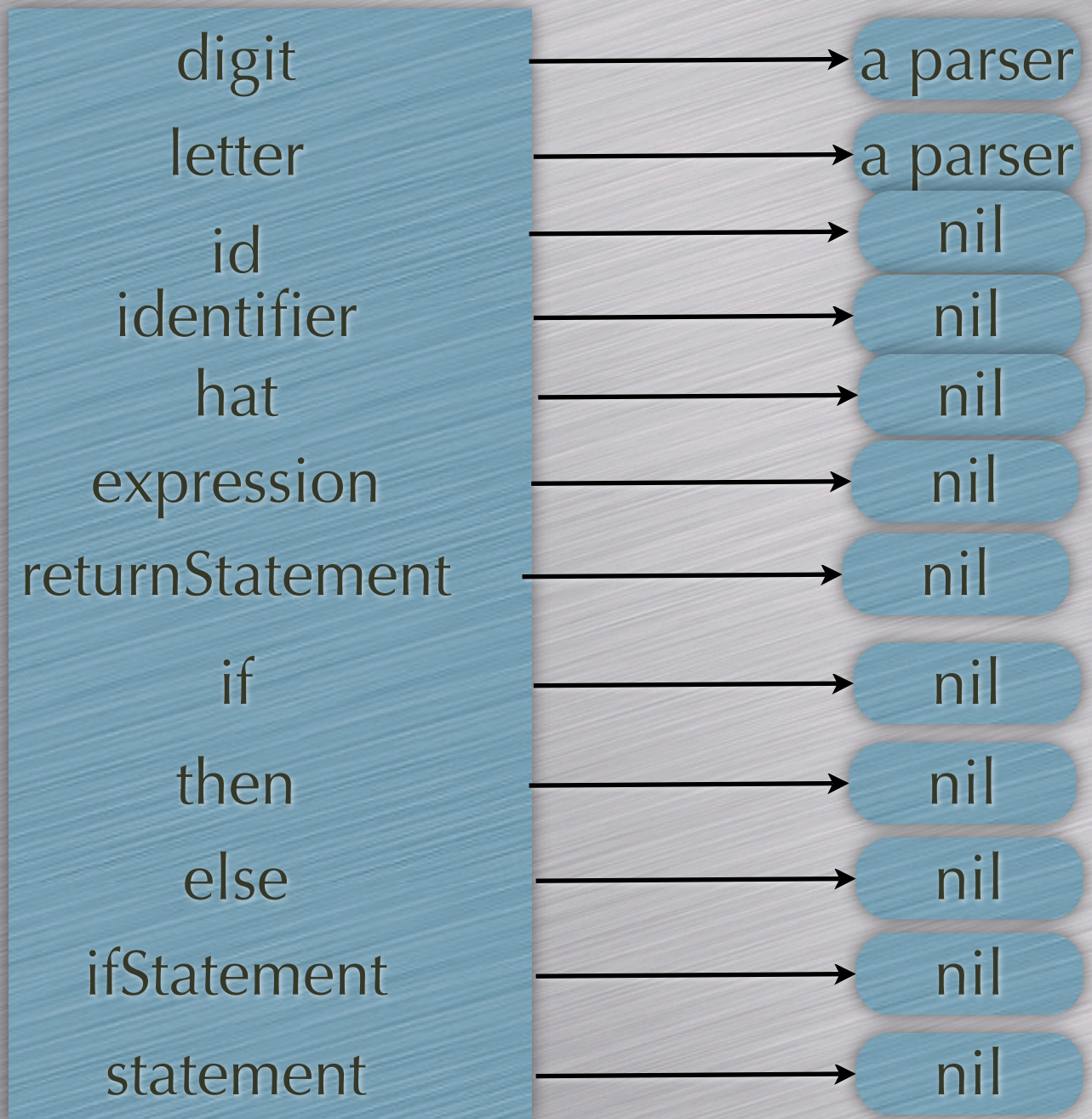
Mutual Recursion



Mutual Recursion

Mutual Recursion

Mutual Recursion



Mutual Recursion

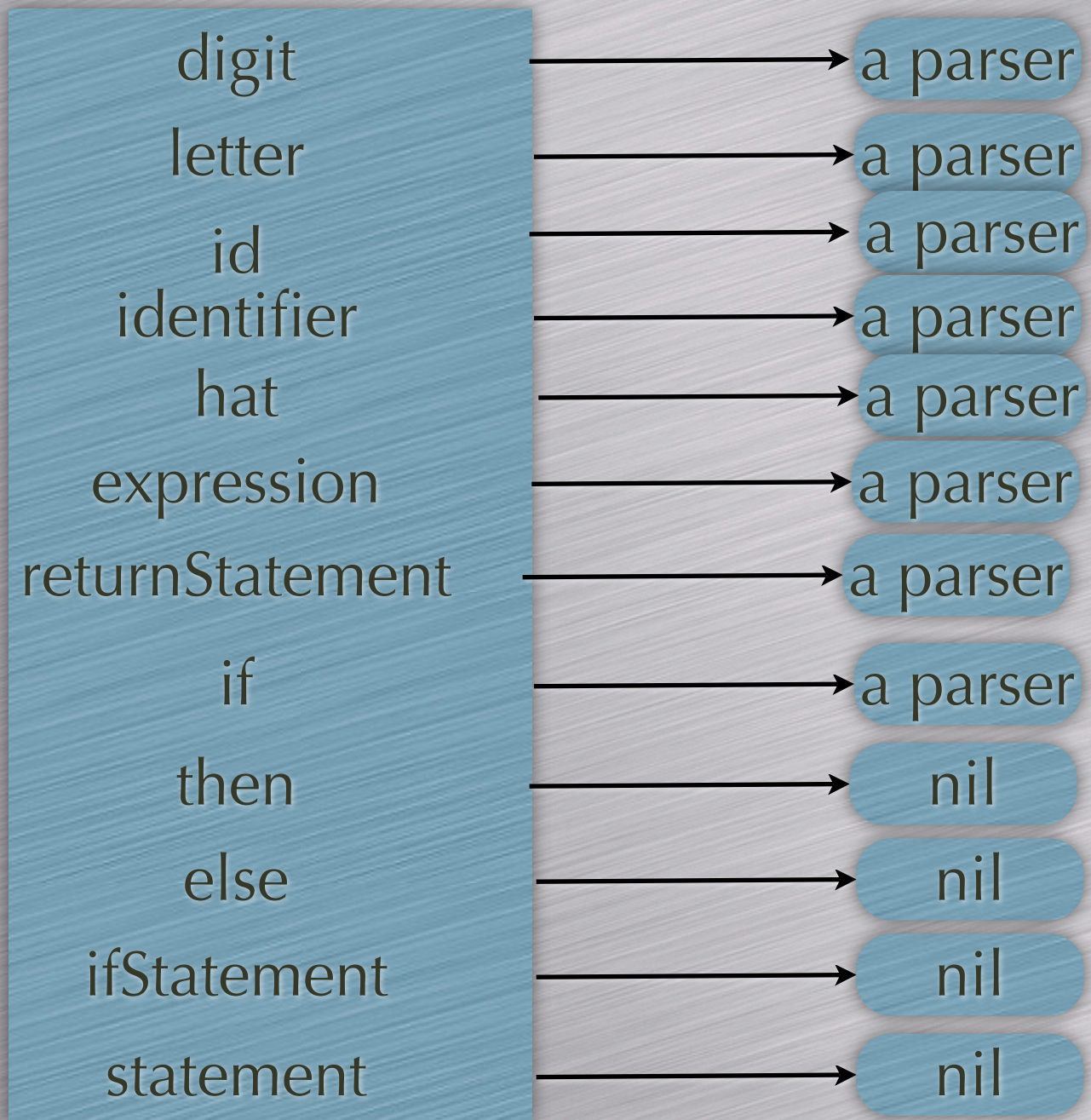
Mutual Recursion

Mutual Recursion

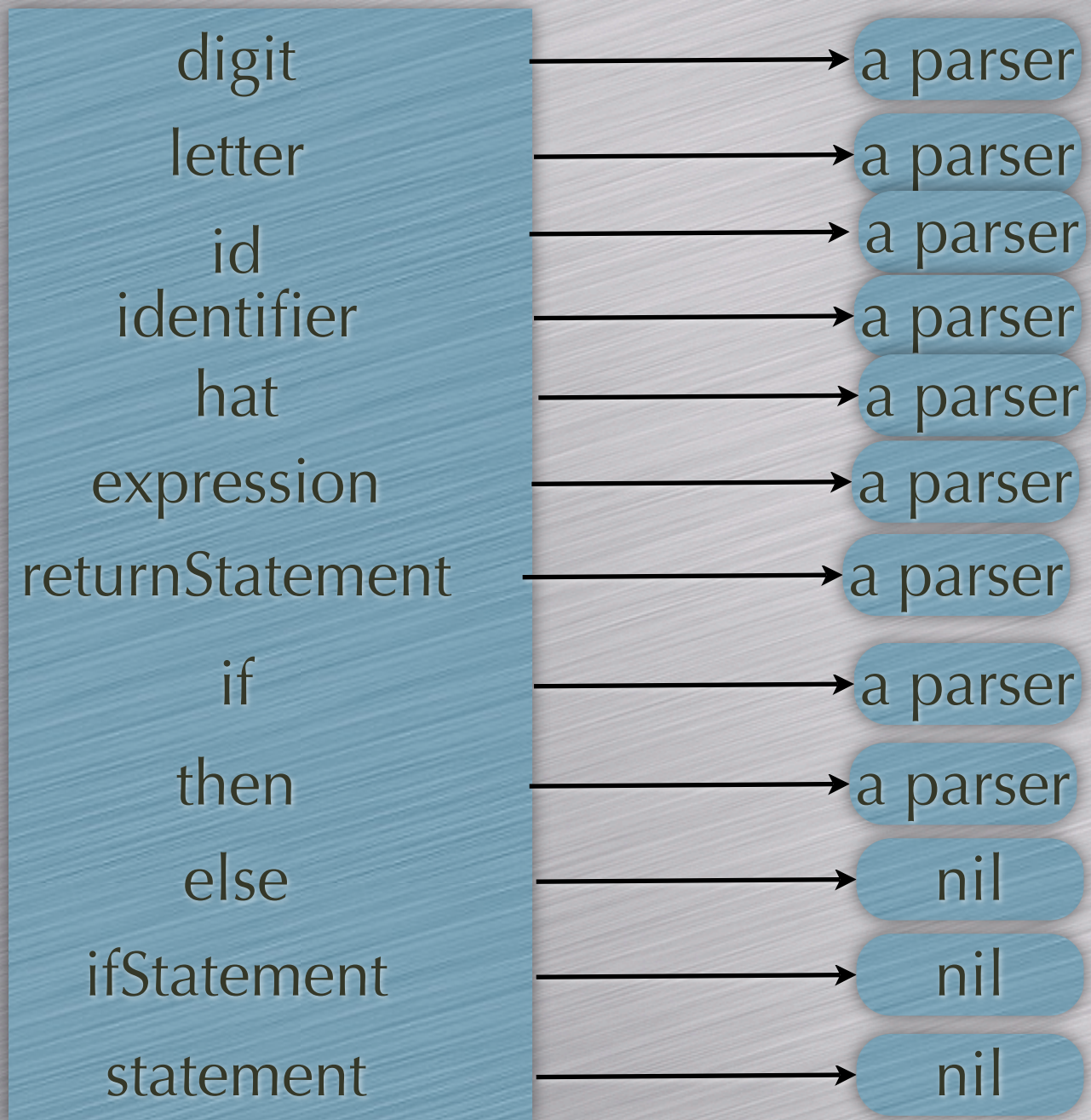
Mutual Recursion

Mutual Recursion

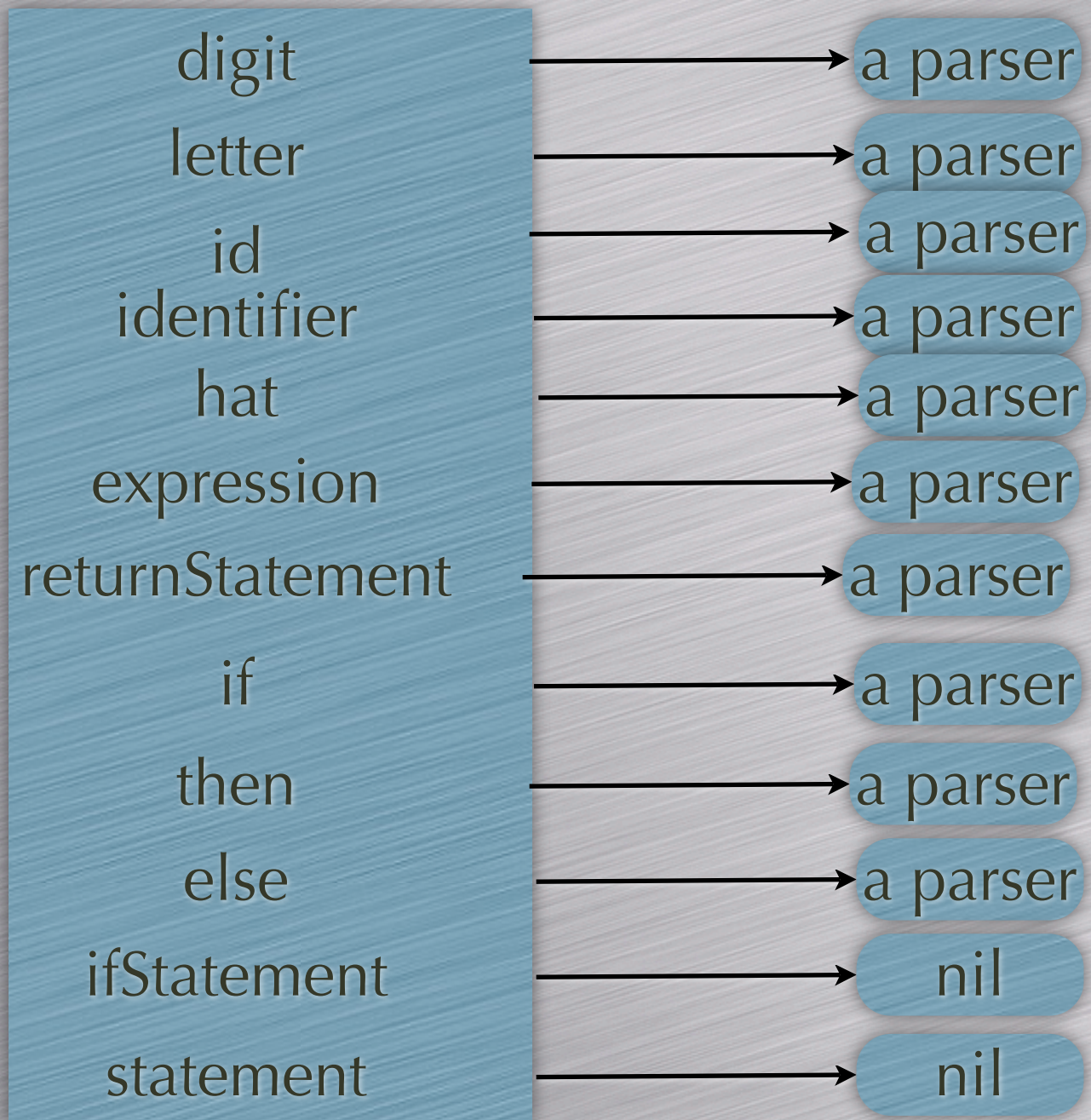
Mutual Recursion



Mutual Recursion



Mutual Recursion



Mutual Recursion

Mutual Recursion

- Addressed with laziness in Haskell
- Can be addressed with closures

ifStatement = if, [expression], [then], [statement],

[else], [statement].

statement = ifStatement | [returnStatement].

Mutual Recursion

- Addressed with laziness in Haskell
- Can be addressed with closures

ifStatement = if, [expression], [then], [statement],

[else], [statement].

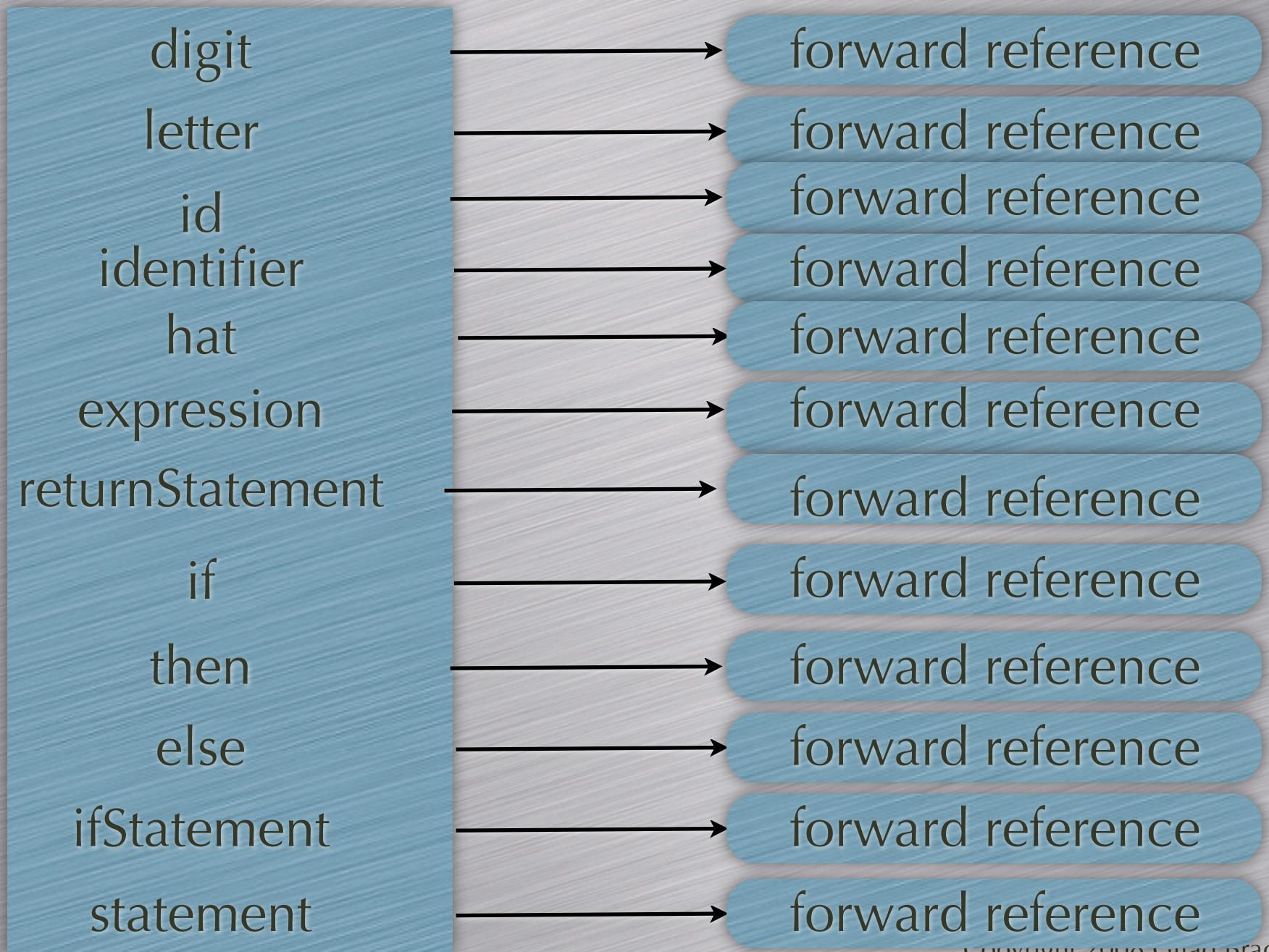
statement = ifStatement | [returnStatement].

Being Reflective rather than Lazy

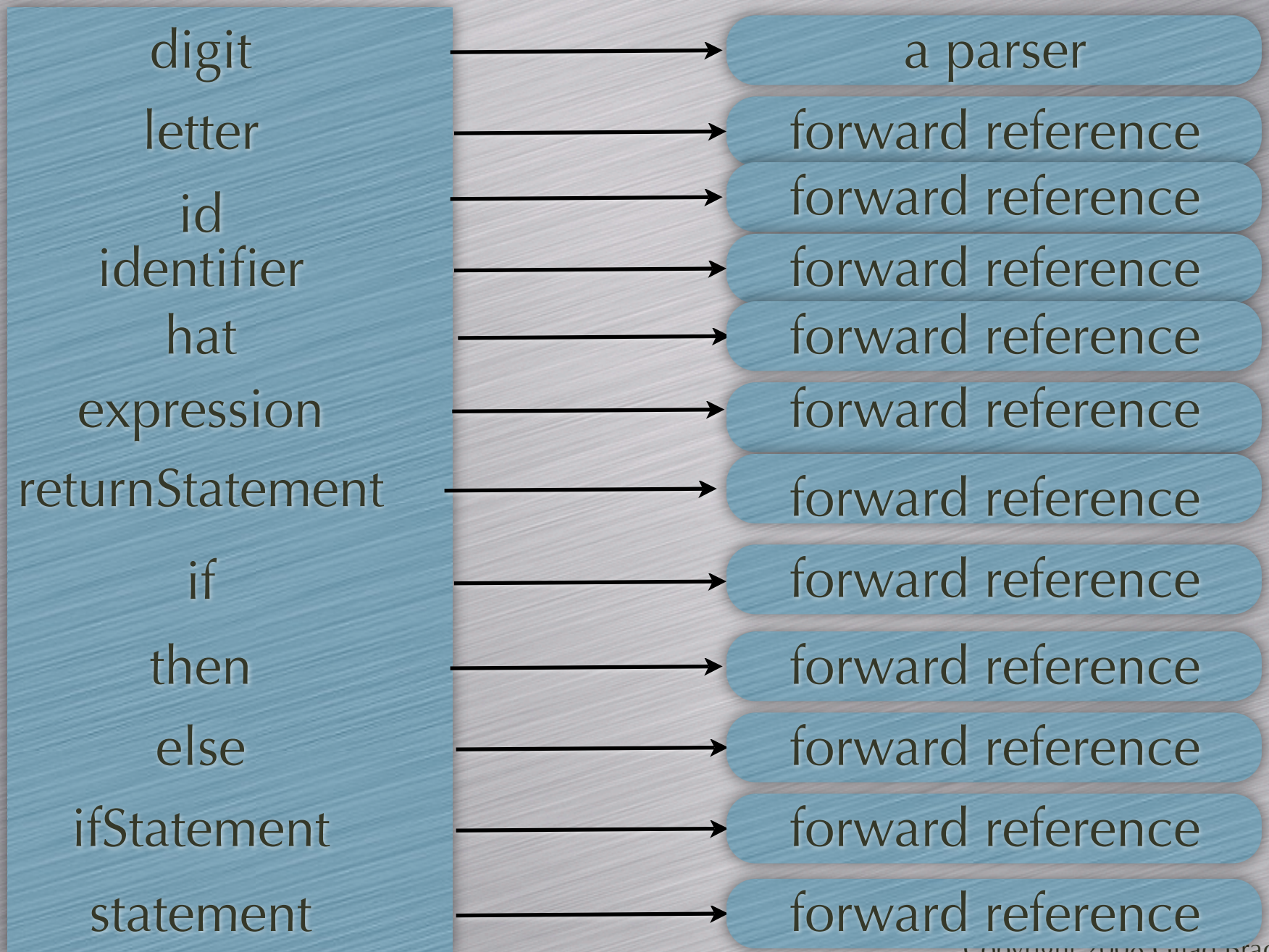
Use Reflection instead of Laziness.

Framework initializes all slots to “stand-in” parsers which act as forward references

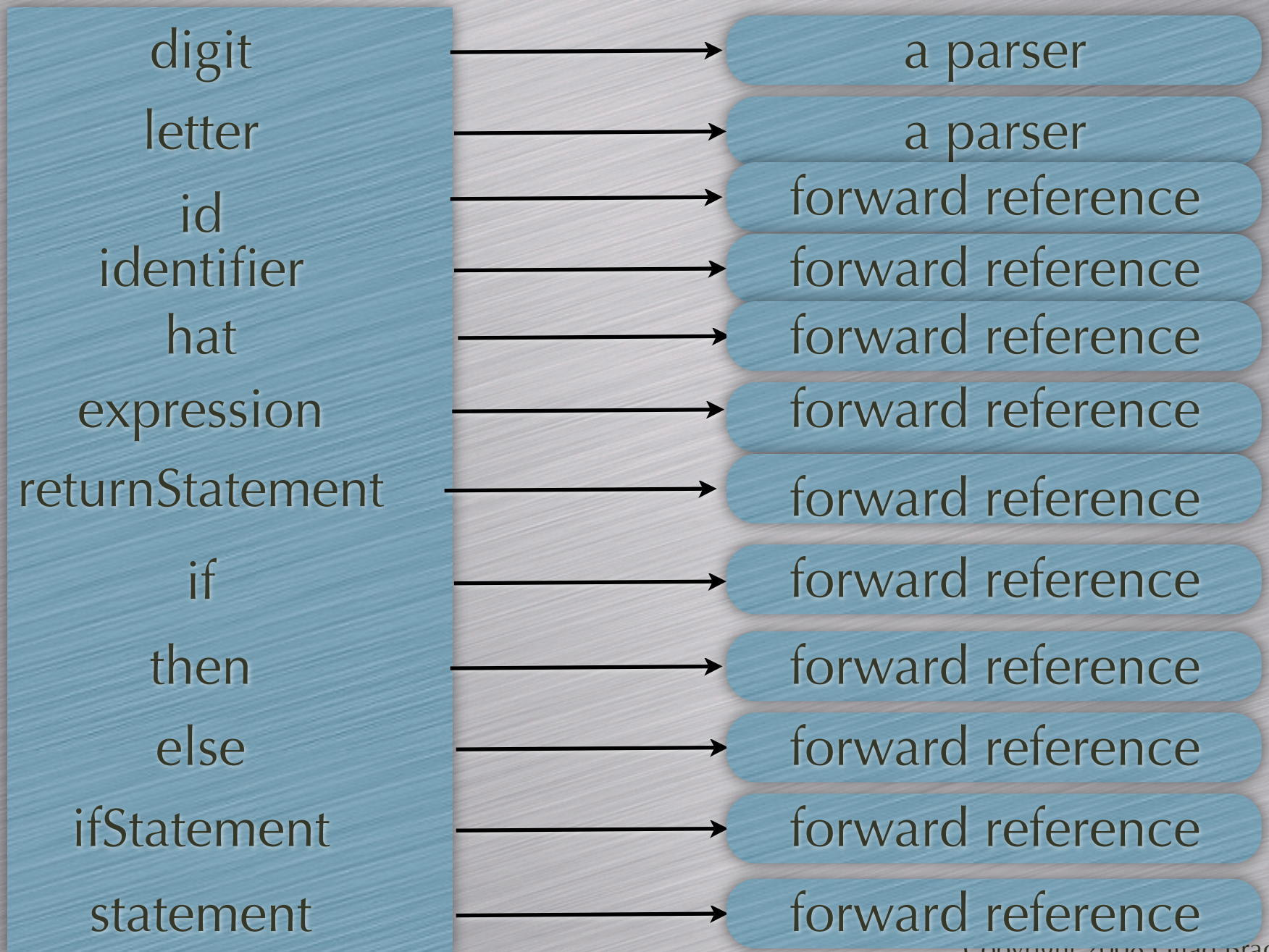
Mutual Recursion



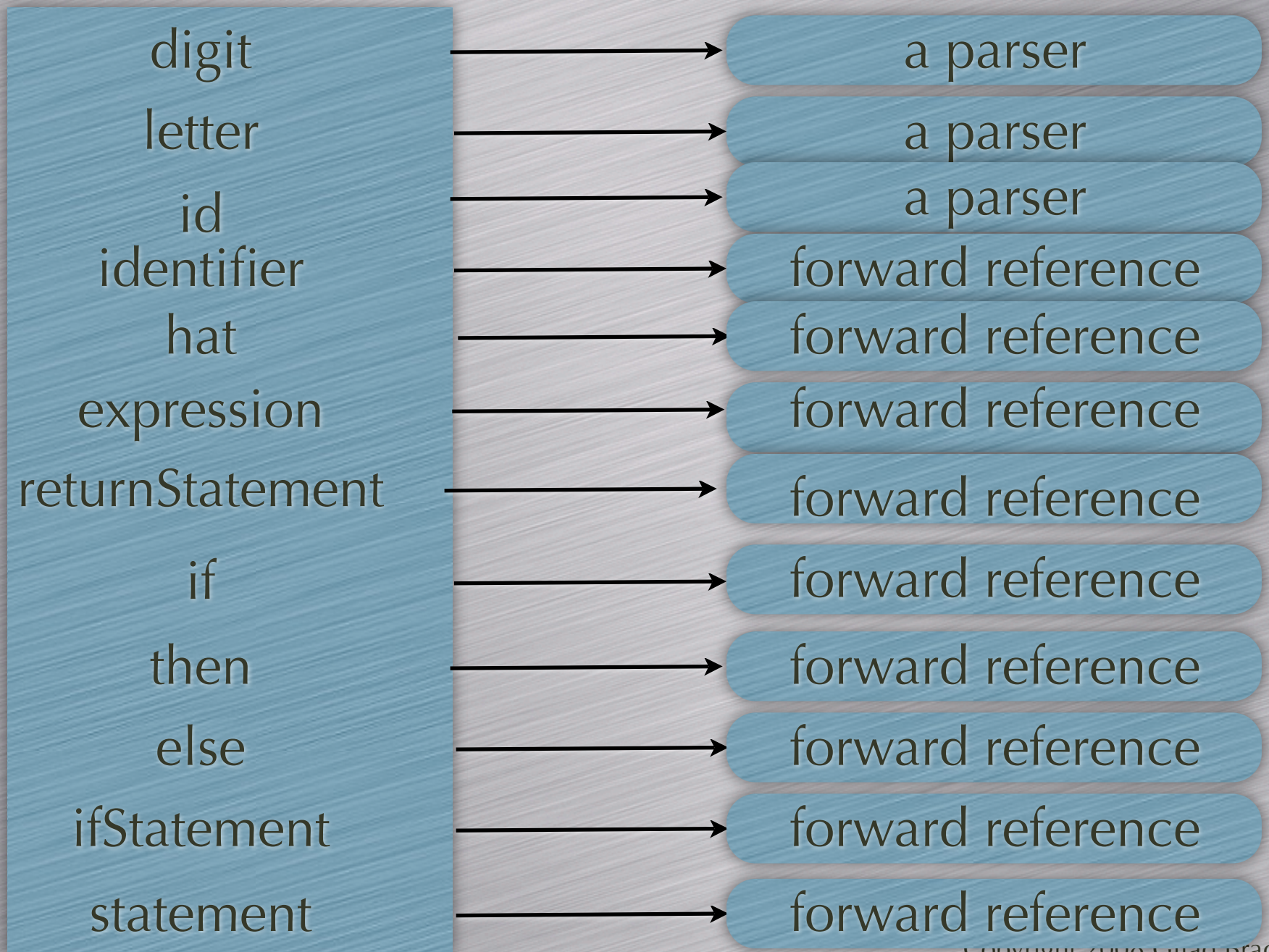
Mutual Recursion



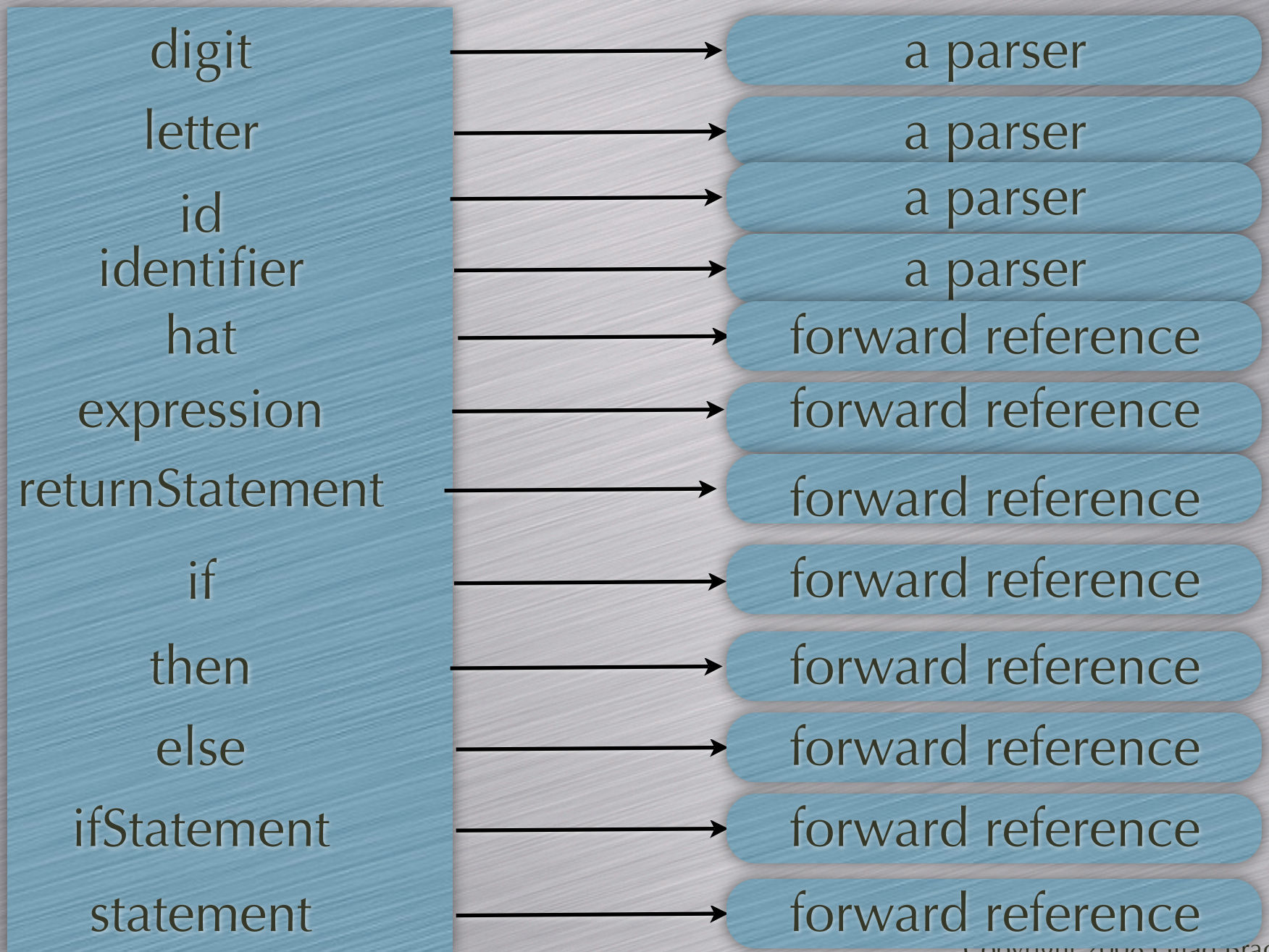
Mutual Recursion



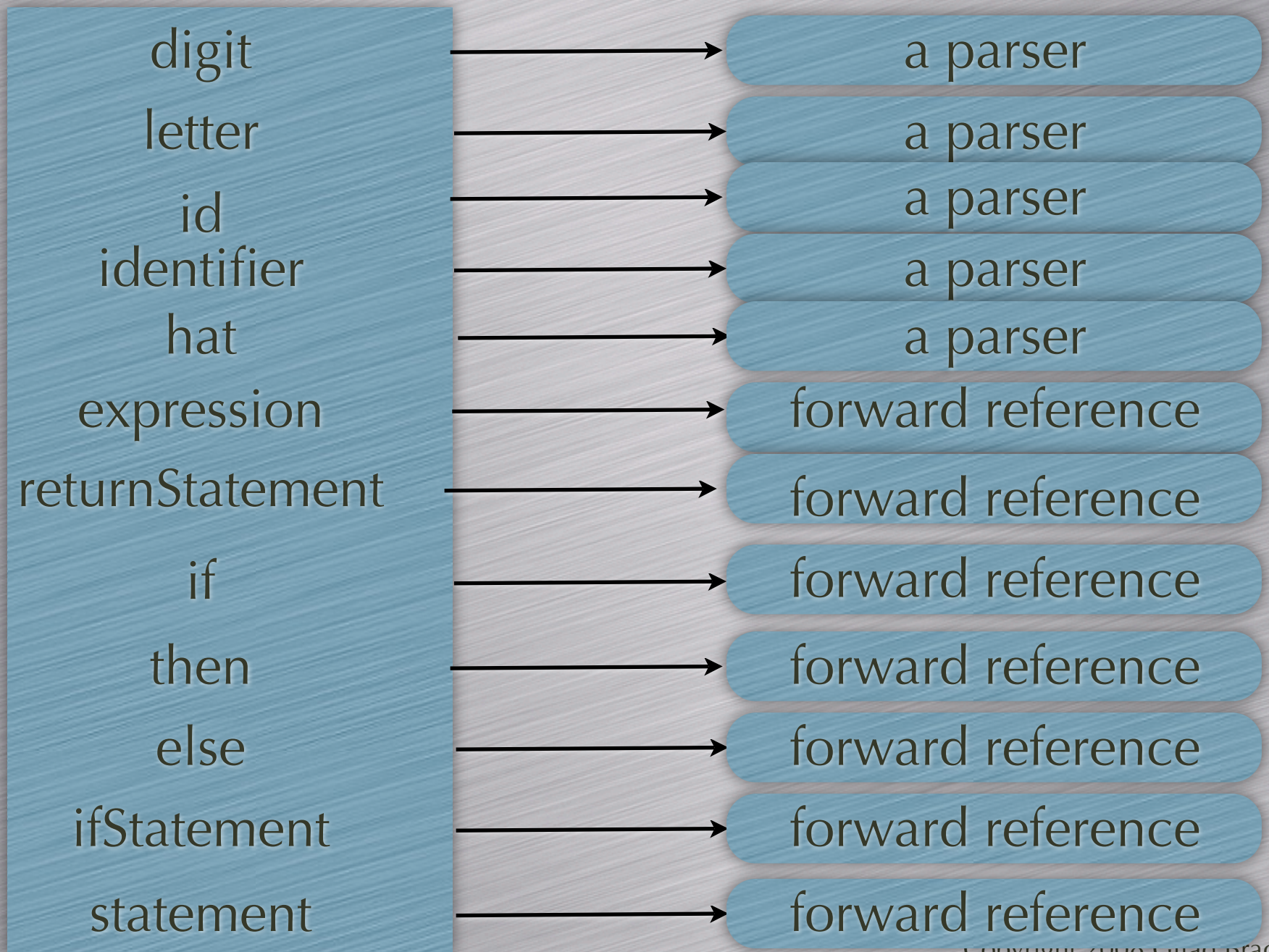
Mutual Recursion



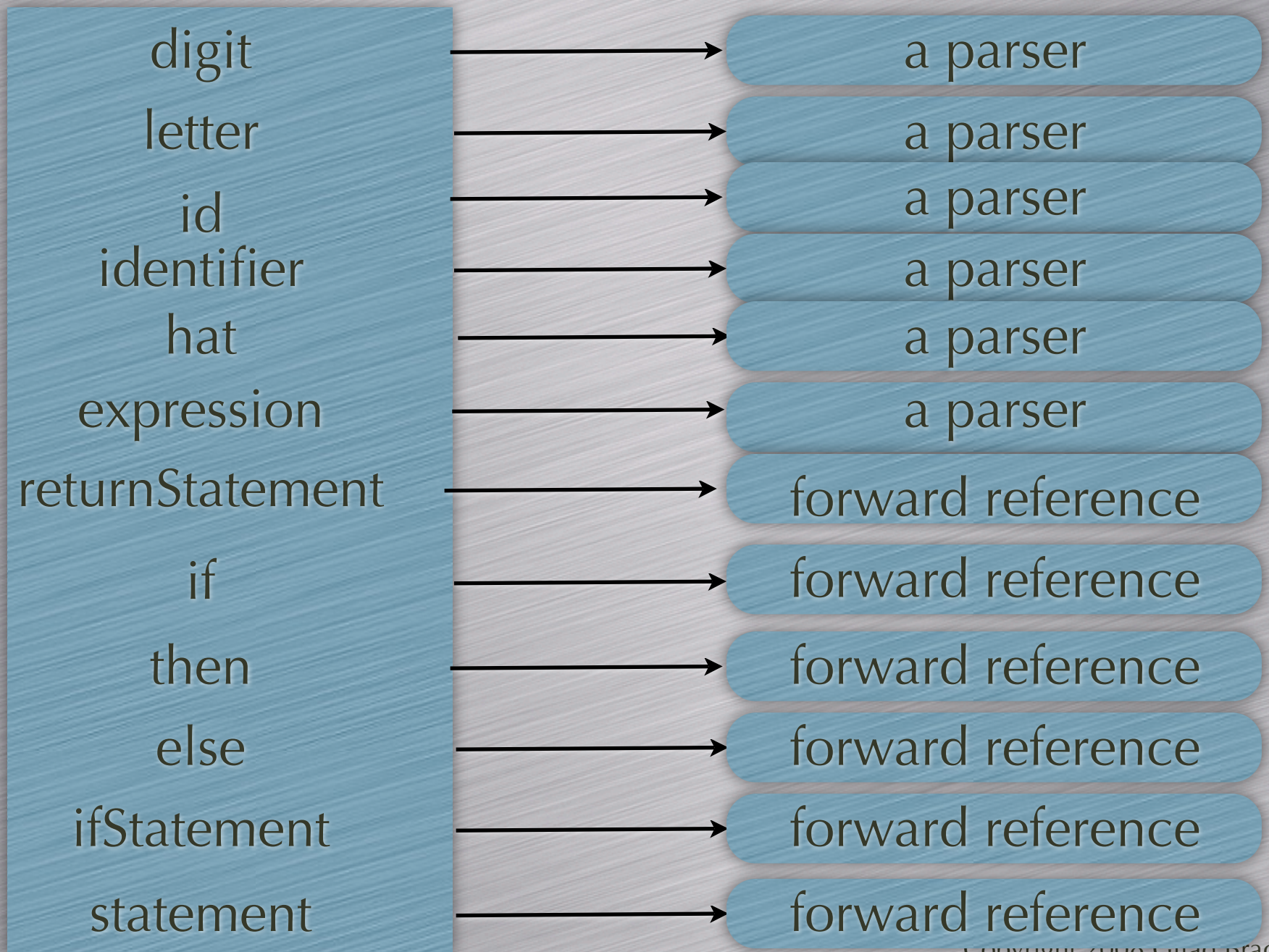
Mutual Recursion



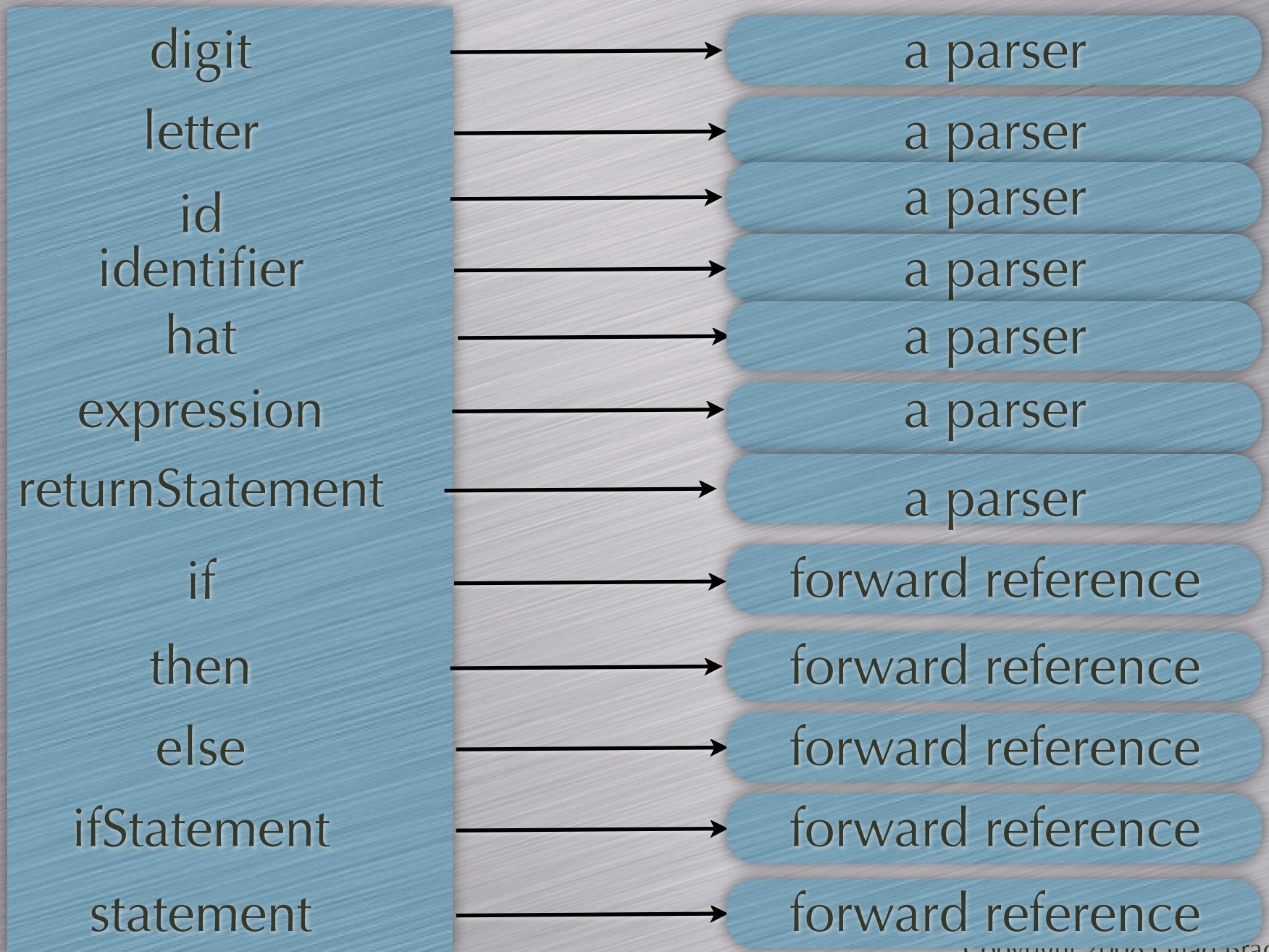
Mutual Recursion



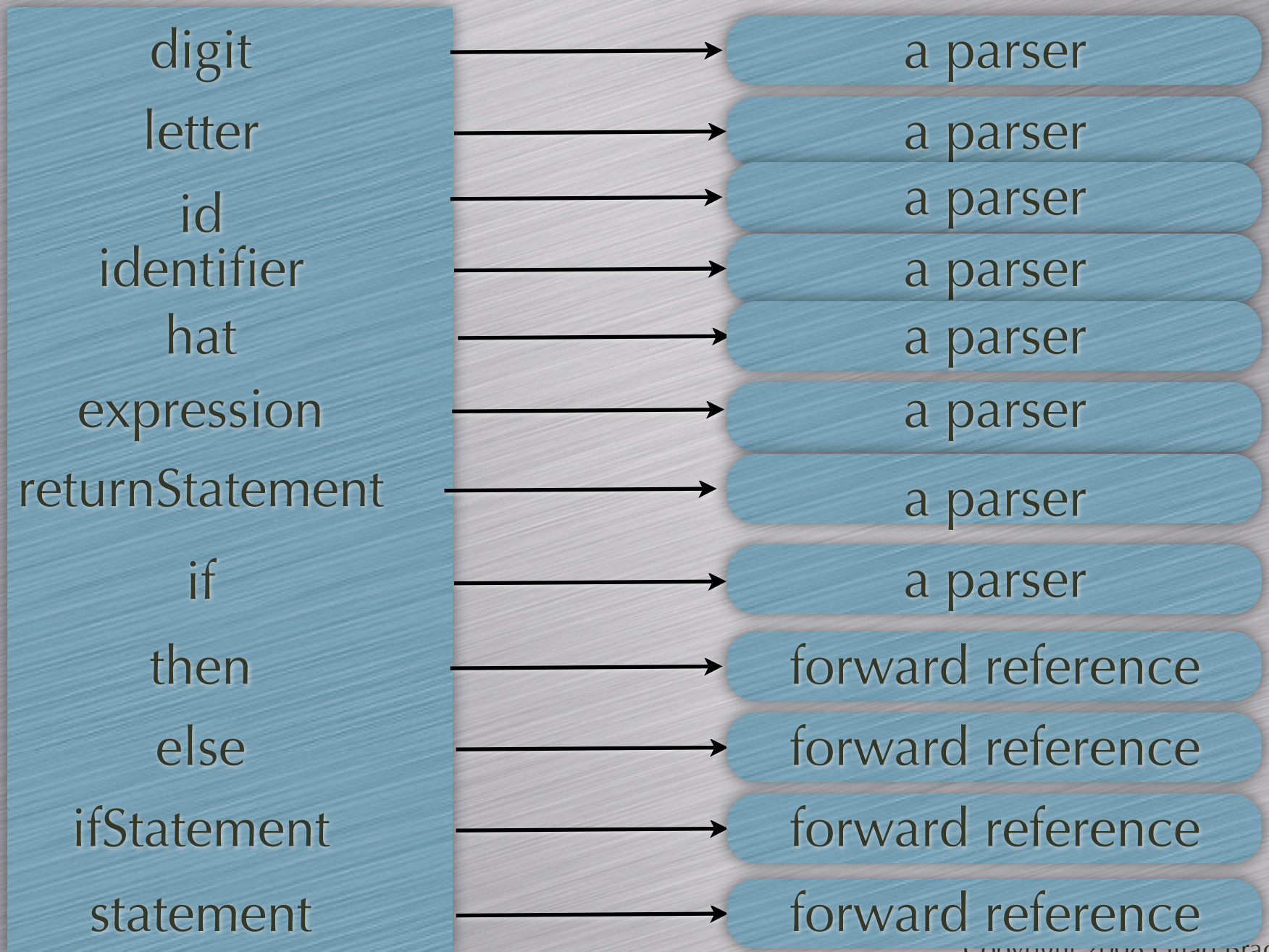
Mutual Recursion



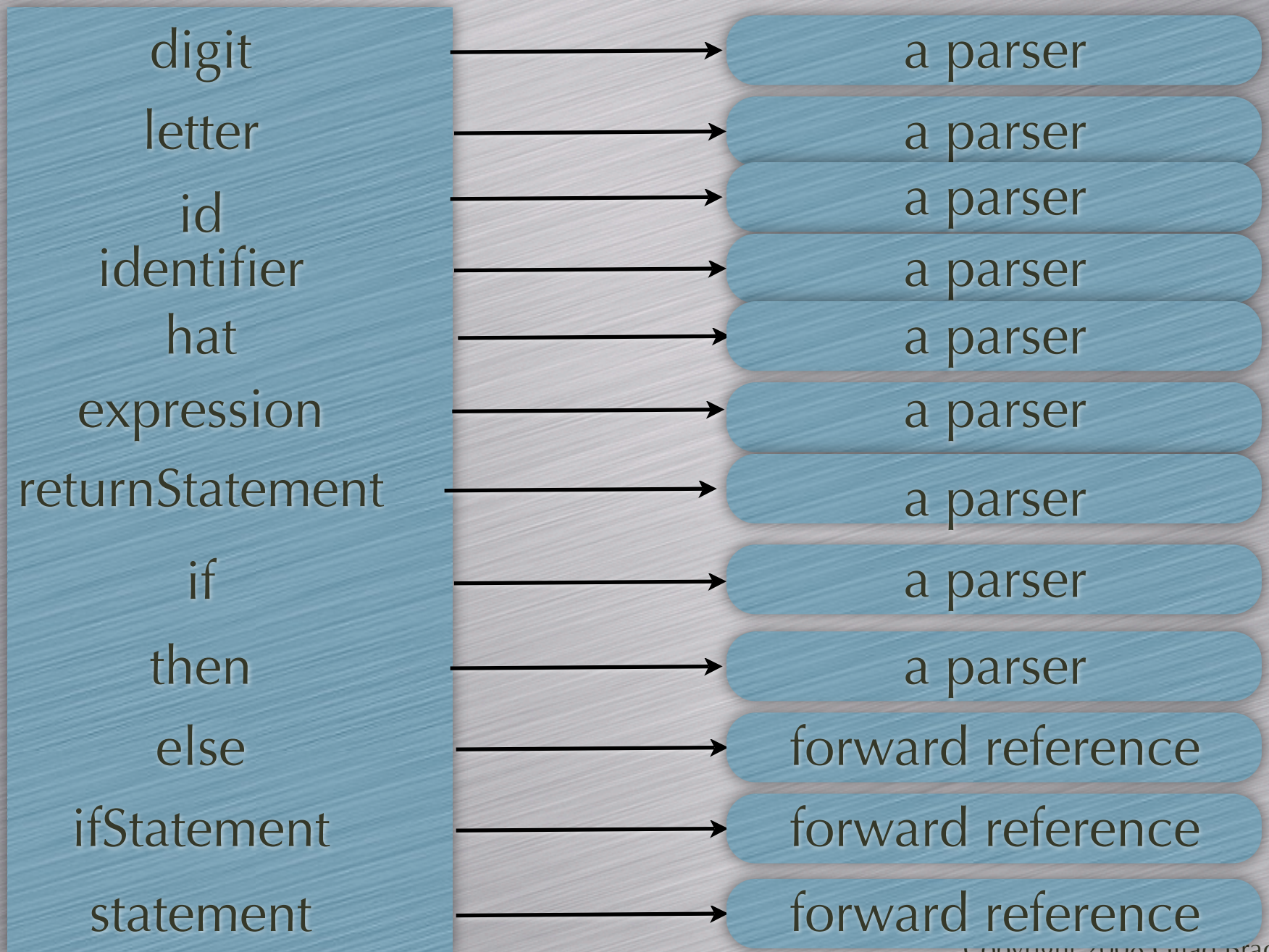
Mutual Recursion



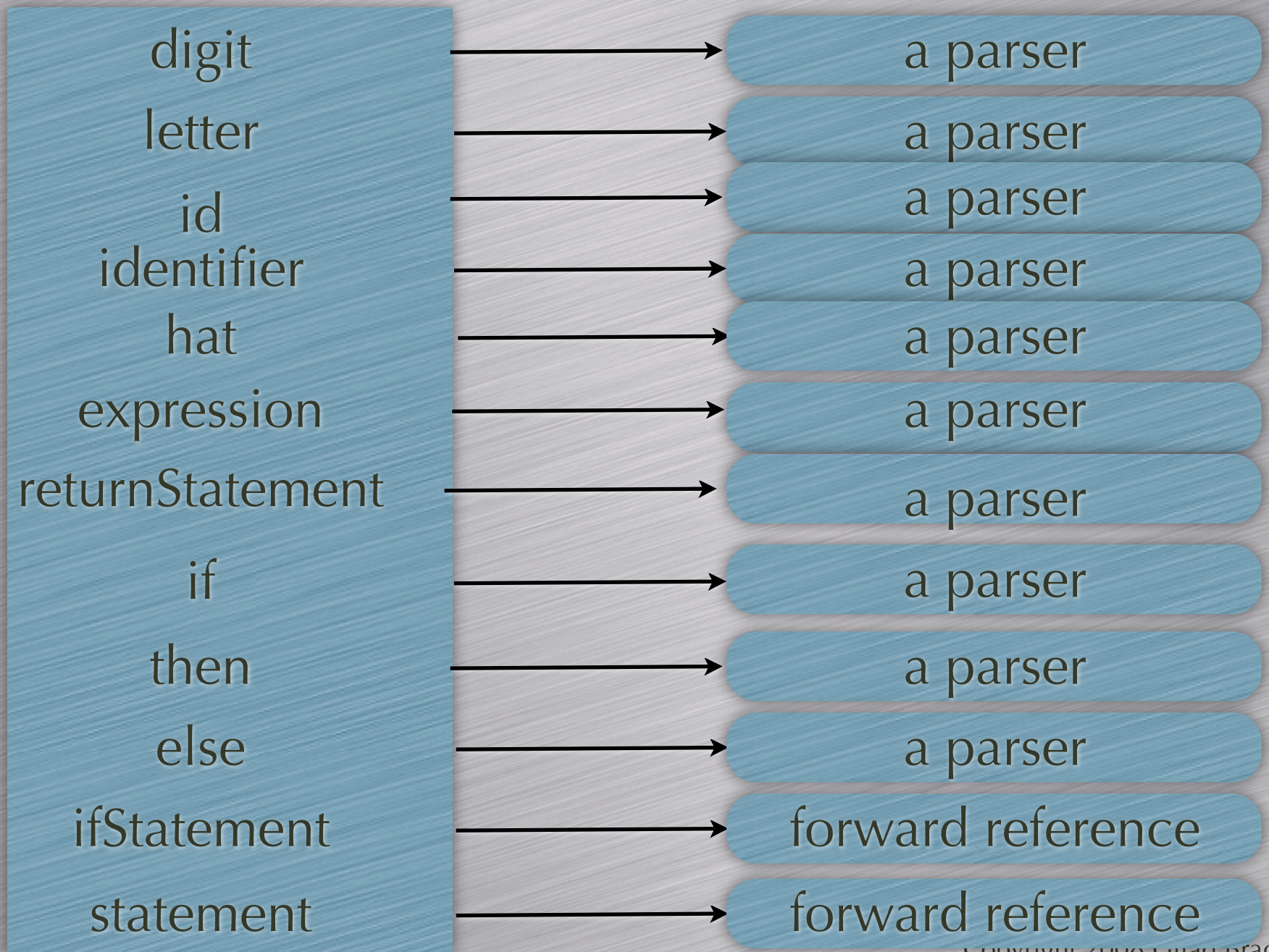
Mutual Recursion



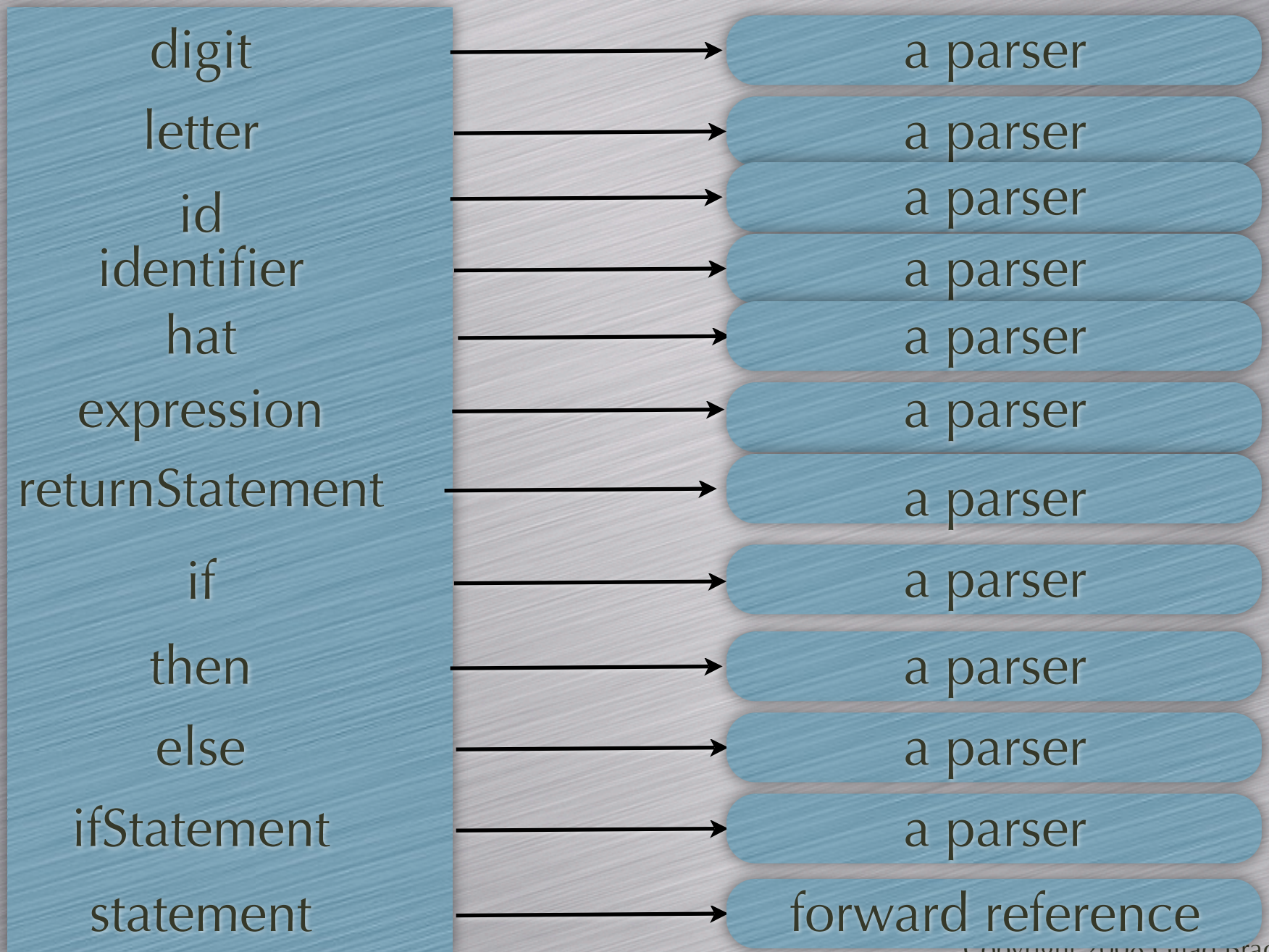
Mutual Recursion



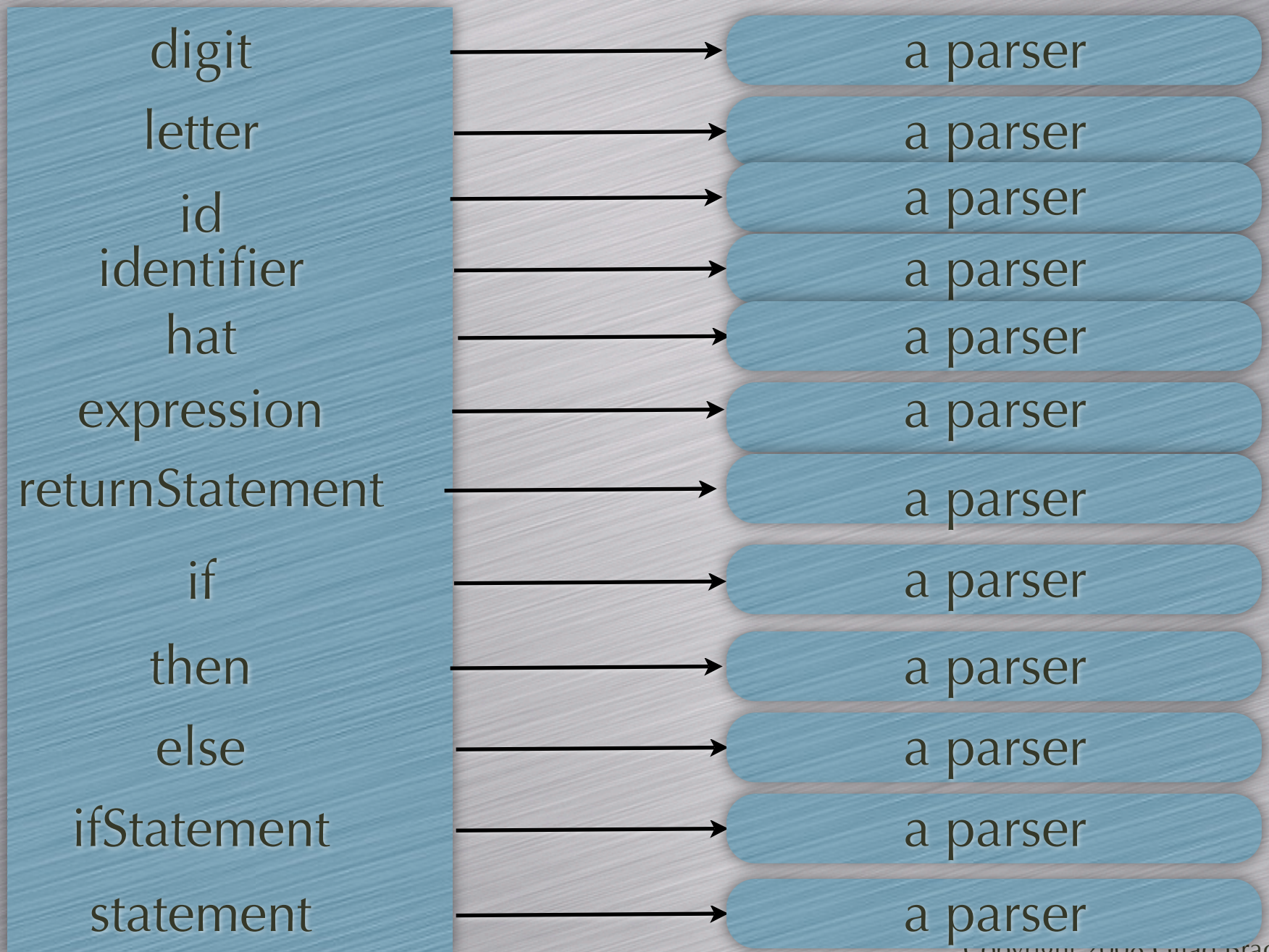
Mutual Recursion



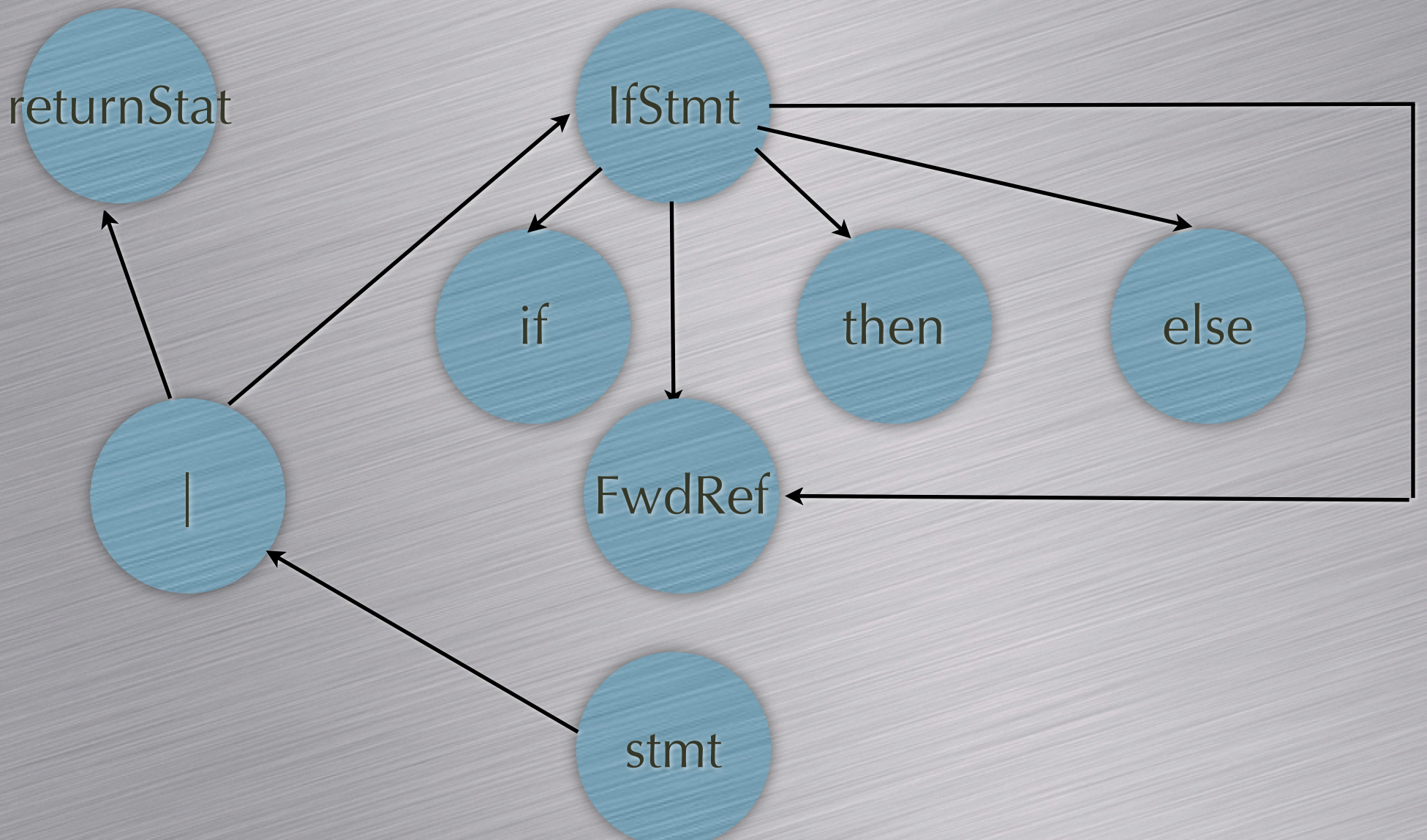
Mutual Recursion



Mutual Recursion



Mutual Recursion



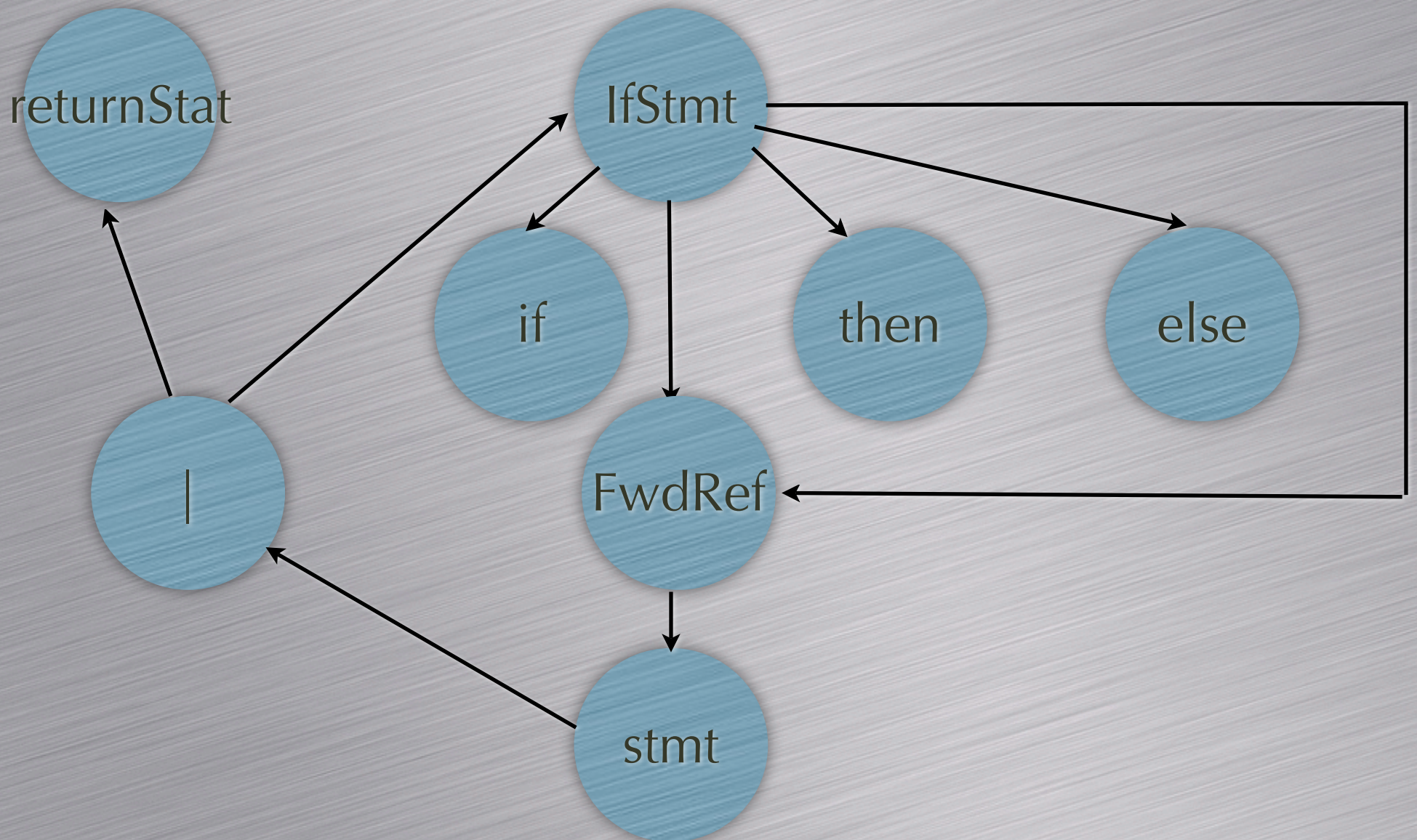
Being Reflective rather than Lazy

Use Reflection instead of Laziness.

Framework initializes all slots to “stand-in” parsers which act as forward references

When grammar is completely initialized, stand-ins are connected to originals and forward calls to them

Mutual Recursion



Composing Parsers

```
class ExampleParser2 = ExampleParser1 mixin |>
                                        ExampleGrammar2 ()
(
  ifStatement = (
    ^super ifStatement
      wrapper:[:ifKw :e :thenKw :s1 :elseKw :s2 |
                  IfStatAST new cond: e;
                                  trueBranch: s1;
                                  falseBranch: s2;
                                  start: ifKw start;
                                  end: s2 end
                ]
  )
)
```

Grammar is shared executable specification

- Shareable among several IDEs/ compilers that require distinct ASTs
- Shareable among other language tools: syntax colorizers, postprocessing tools

Connections

- Mirrors
  - Self
  - Strongtalk, JDI, APT ... See OOPSLA 04
- No static
  - Scala
  - E

Connections

- Security
  - E (Miller 06)
  - Java
- Modules
  - Jigsaw, 1991
  - Units
  - ML
  - Fortress

Connections

- Message-based programming
  - Emerald, Trellis/Owl
  - Smalltalk
  - Self
- Virtual Types
  - Beta
  - gBeta
  - Scala, ...
- Hierarchy inheritance: Ossher & Harrisson 92, Cook 89

Connections

- Parser Combinators.
  - Too much to survey; some highlights:
    - Parsec (Haskell parser combinator library)
    - Swierstra et al.
    - Sparsec (Scala parser combinator library)
    - PEGs (Ford, POPL04, ICFP)
    - Tedir (Plesner-Hansen's masters thesis)
    - Ometa (OOPLA 07 DLS)

Conclusions

- Natural and powerful synergy between:
  - Message-based programming
  - Component style modularity
  - Virtual classes, mixins, class hierarchy inheritance
  - Object capability model and security
  - Mirror based reflection
  - Actor style concurrency
  - Pluggable types