



# Can Beautiful Languages Do Access Control?

Gilad Bracha & Ryan Macnak  
Ministry of Truth

# Newspeak Can!

- 🎧 I'll explain how, and why it is interesting

# Access Control is Messy

- Mostly dealt with statically
- Difficulties with binary compatibility, dynamic loading, reflection
- Runtime must be involved (e.g., Java)
- Dynamically typed languages tend to ignore the issue, or have very limited access control

# Object-based vs. Class-based Encapsulation

- In object-based encapsulation, privacy is per object
- In class-based encapsulation, privacy is per class

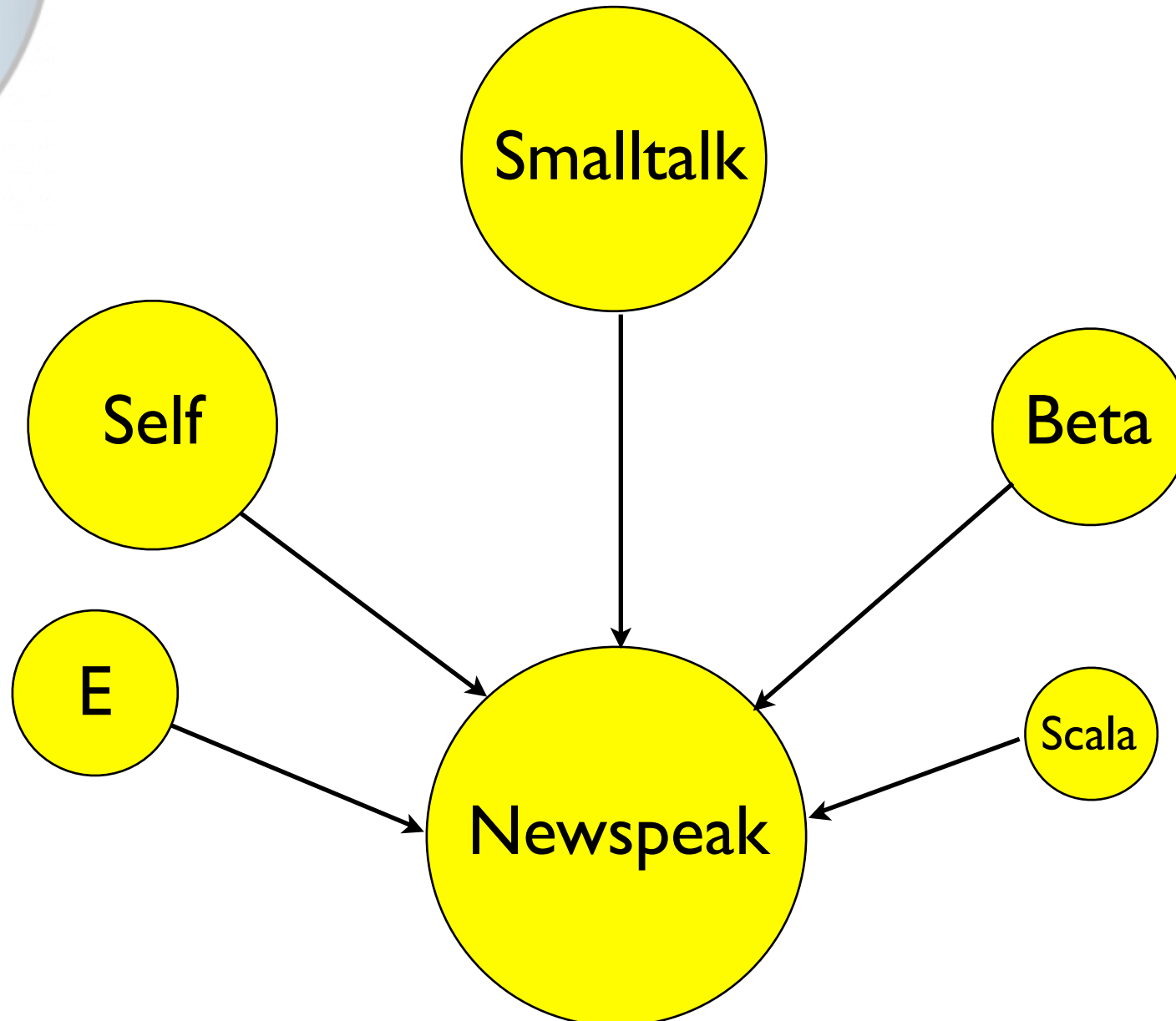
# Class-based Encapsulation

```
class C {  
    private int secret = 101;  
    int extractFrom(C c) {  
        return c.secret;  
    }  
}
```

# Object-based Encapsulation

```
class C {  
    private int secret = 101;  
    int extractFrom(C c) {  
        return c.secret; // error  
    }  
}
```

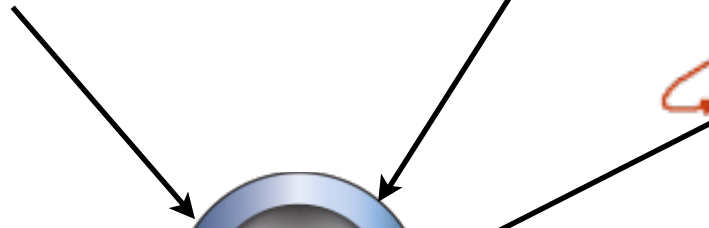
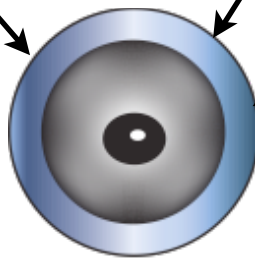
# Genealogy of Newspeak



# Newspeak 101



Self





# Goals

- ⦿ Modularity
- ⦿ Security
- ⦿ Reflectivity
- ⦿ Interoperability

# Newspeak

- Newspeak is a dynamic, class based language with two defining properties:
- All names are late bound
- No global namespace

# Newspeak

- ⦿ All names are late bound
- ⦿ No global namespace



# Message-based Programming

Every run time operation is a message send.



# Translation

Every run time operation is a virtual method call.



# No References to Fields



# Slots

Each slot declaration introduces getter method;  
If slot is mutable, also introduces setter, e.g.,

$t = 5.$

$v ::= \text{'abc'}$ .

# Representation Independence

- No code depends on our choice of storage representation:
  - Not clients
  - Not subclasses
  - Not even the class itself



# Uniform Reference

- ① All object properties are accessed the same way
- ① No distinction between methods and getters/setters
- ① No need to remember which is which
- ① No need to choose which to use

# No References to Classes

- Always use accessors
- Classes are first class objects
- Classes are always virtual
- Classes are always mixins
- Class hierarchy inheritance

# Newspeak

- ⦿ All names are late bound
- ⦿ No global namespace

# Nested Classes

- Newspeak modularity is based *exclusively* on classes
- No packages, modules, bundles, templates ...

# Goals

- **Modularity**
- Security
- Reflectivity
- Interoperability



# External Dependencies are Explicit

- ⦿ Module definition = Class not nested within another class
- ⦿ No access to surrounding namespace
- ⦿ All names locally declared or inherited from Object



# Modules are Sandboxes

- Factory method parameters are objects/capabilities that determine per-module sandbox



# Multiple Implementations

- Modules are objects, accessed via an interface
- Different implementations can co-exist



# Side by Side

- Module definitions are instantiated into stateful objects known as *modules*
- Easy to create multiple instances, with different parameters

# Side by Side Modules

*platform:: Platform new.*

*m1:: NewspeakParsing  
using: platform  
parseLib: (CombinatorialParsing  
usingLib: platform)*

*m2:: NewspeakParsing  
using: platform  
parseLib: (PackratParsing usingLib: platform)*



# Modules are Re-entrant

- Module definitions are deeply immutable
- Modules cannot step on each other's state

# Goals

- Modularity
- Security
- Reflectivity
- Interoperability

# Security

- Object-capability model (Miller06)
  - Object reachability defines authority

# Security

- Object-capability model (Miller06)
  - Object reachability defines authority
  - No static state
    - No Ambient Authority

# Security

- Object-capability model (Miller06)
  - Object reachability defines authority
  - No static state
    - No Ambient Authority
- Access control



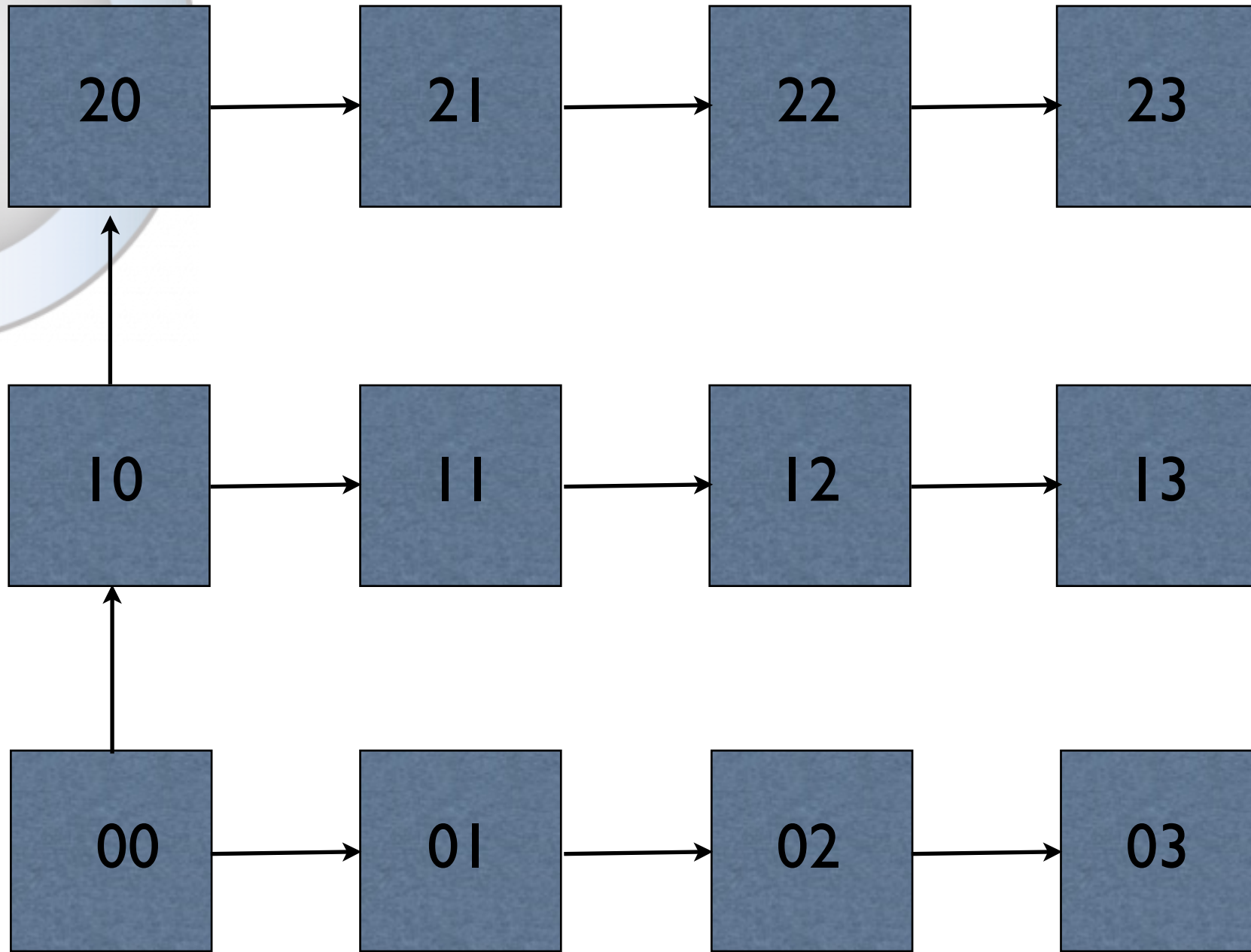
# Access Control in Newspeak

- Access control interacts with lexical structure (class nesting) and inheritance



# The Comb Rule

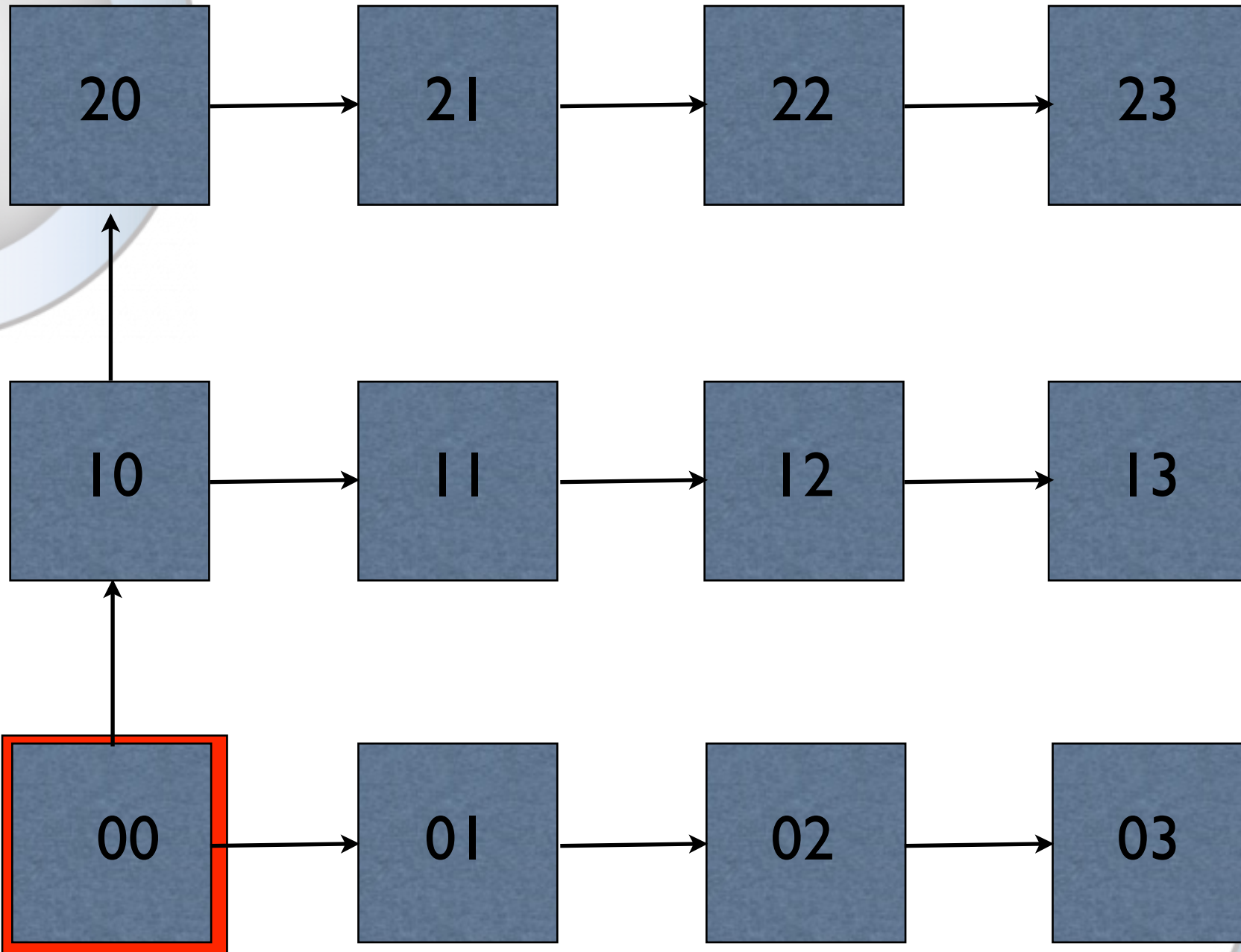
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

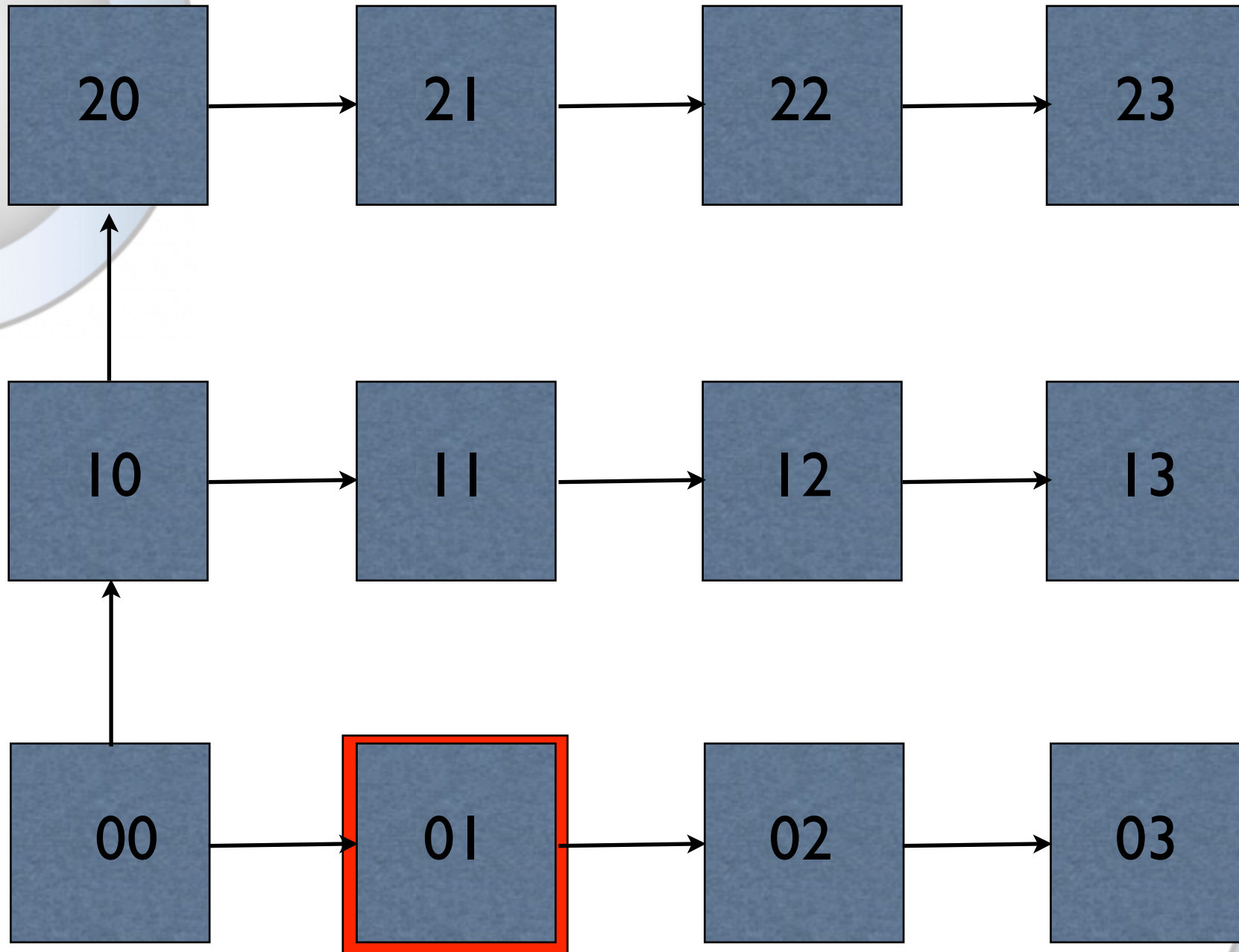
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

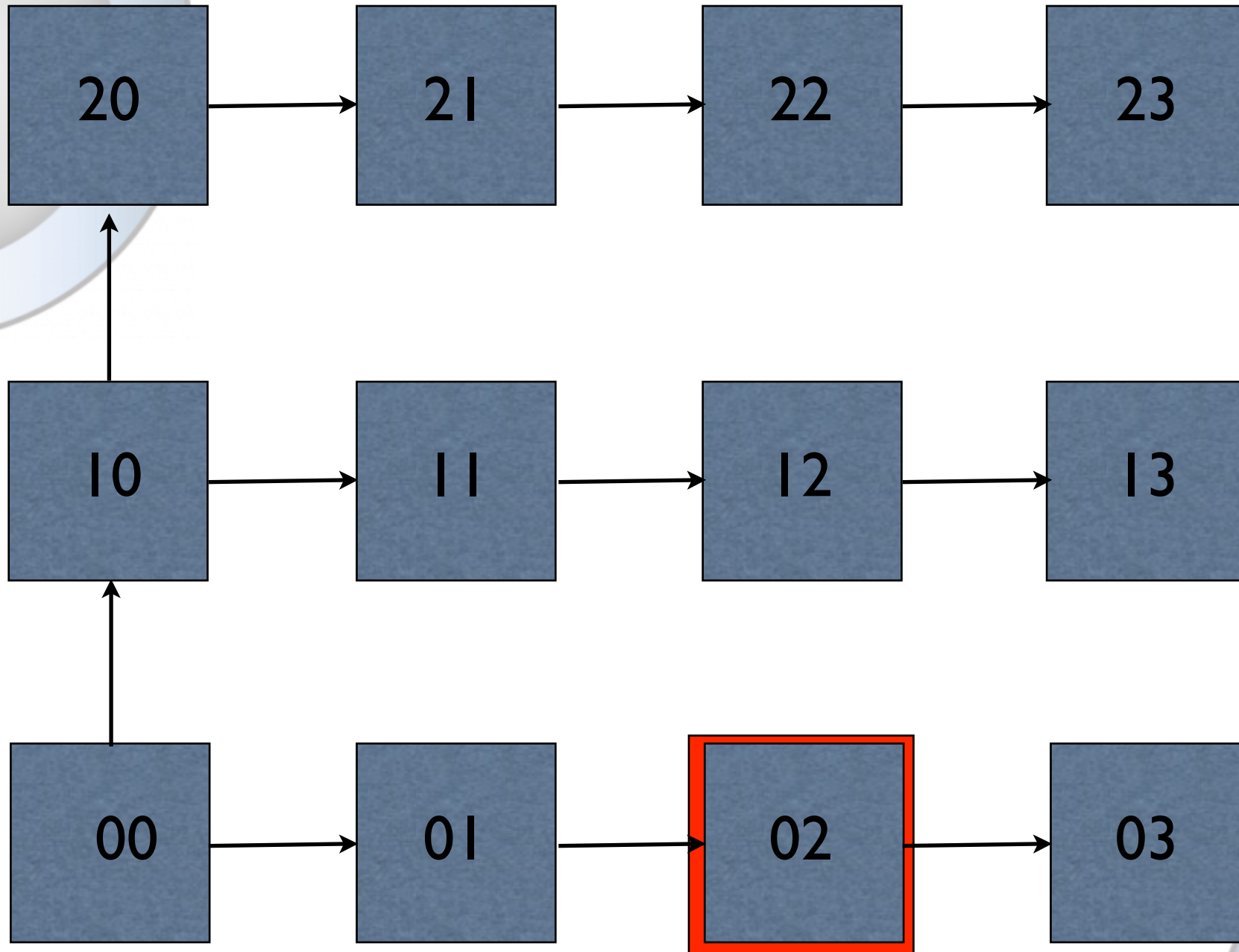
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

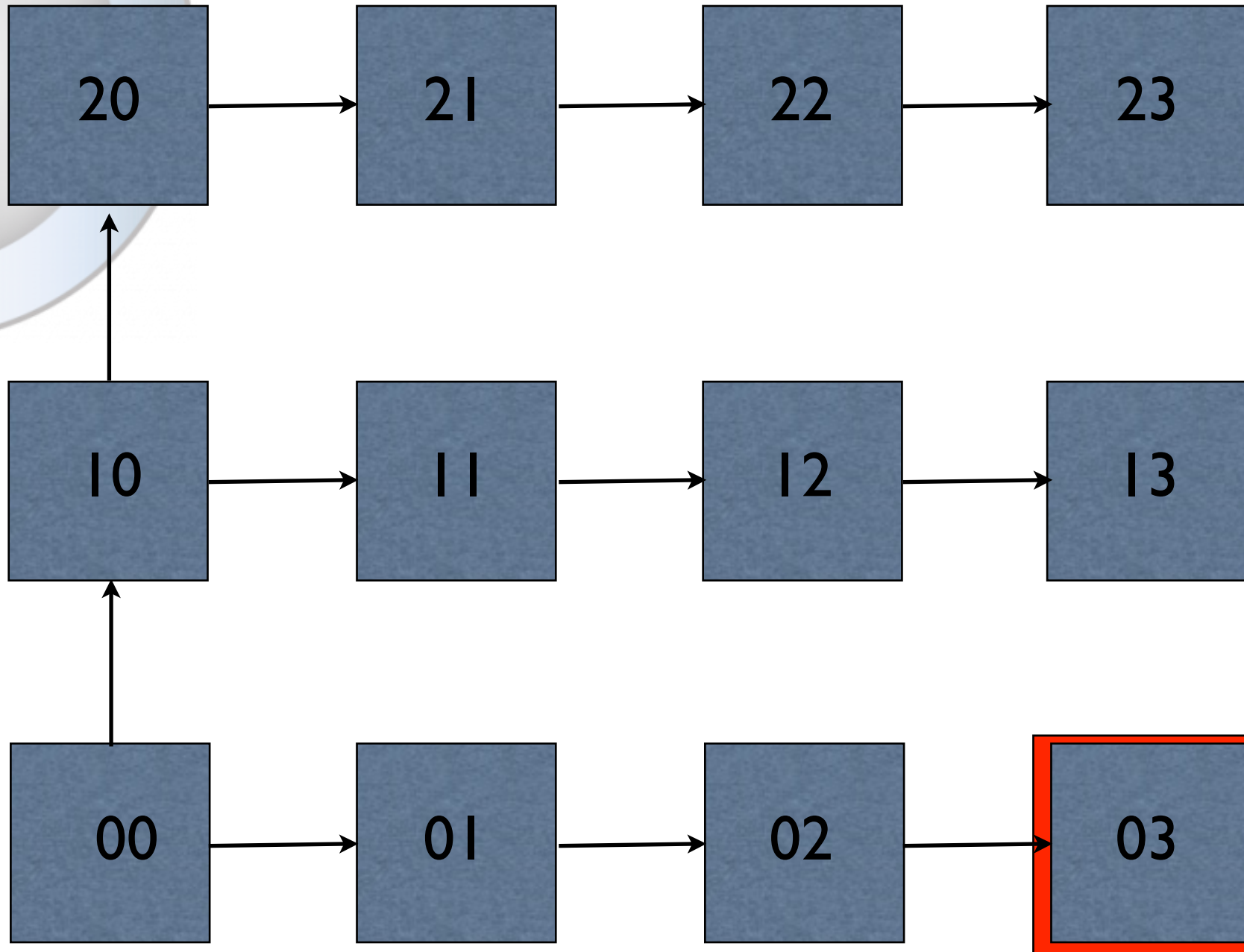
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

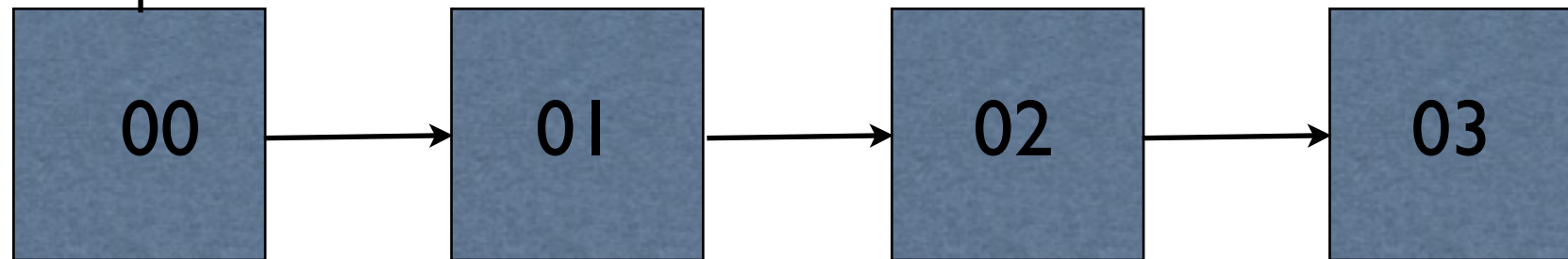
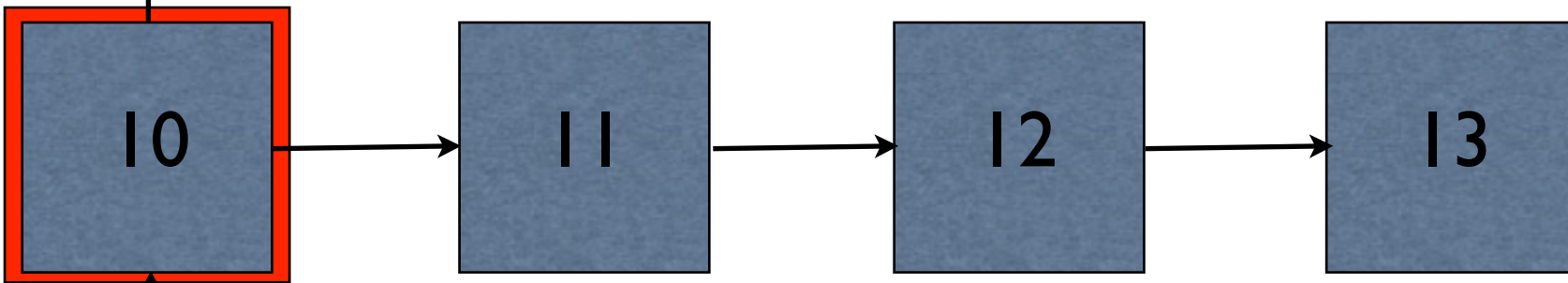
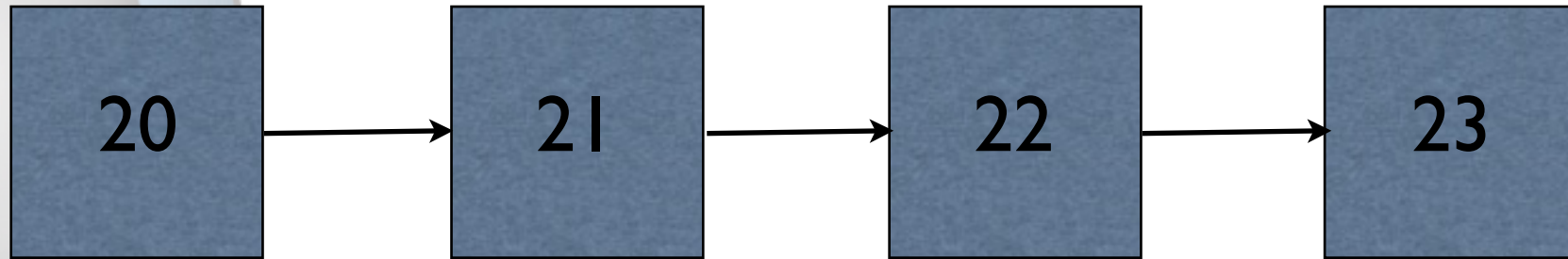
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

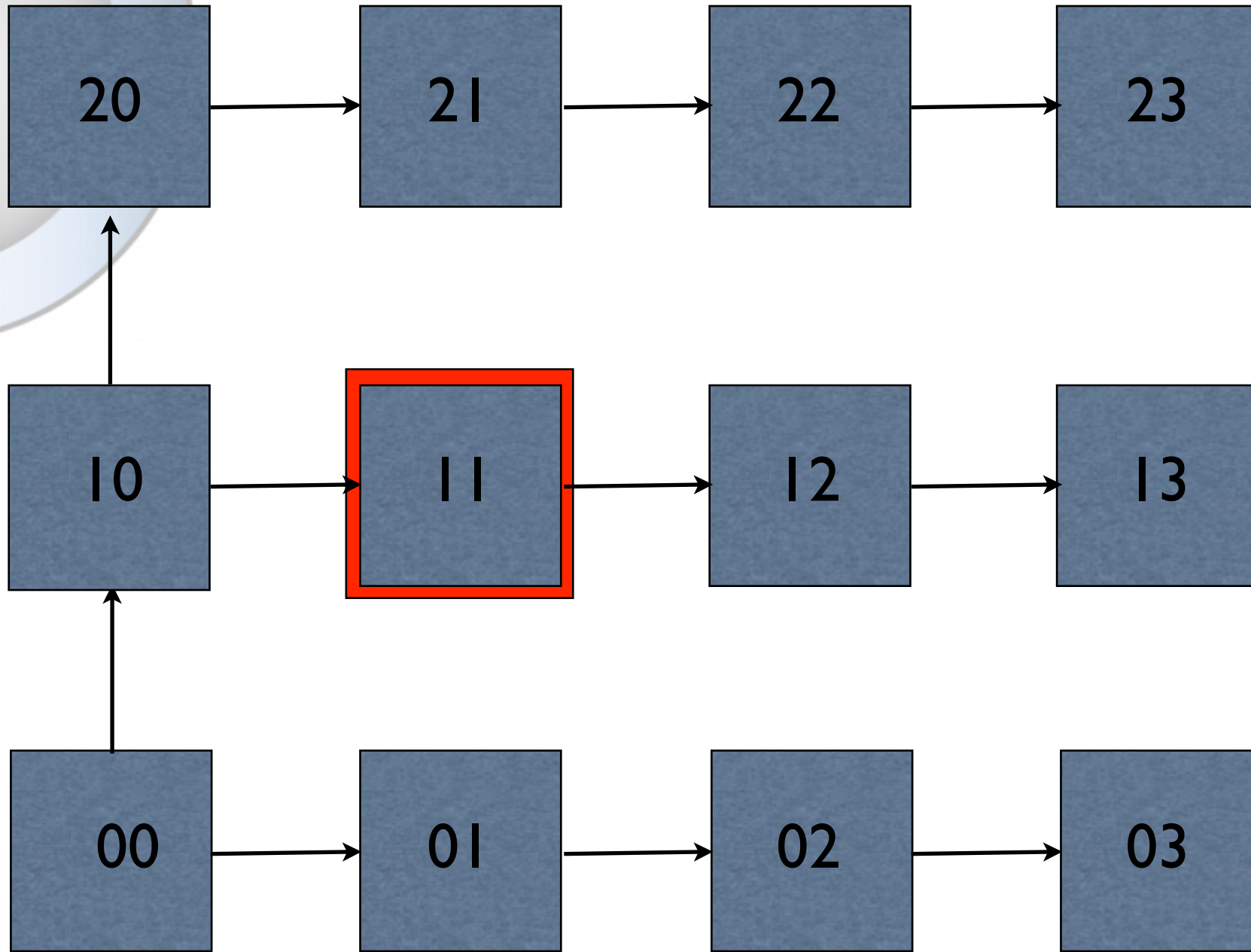
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

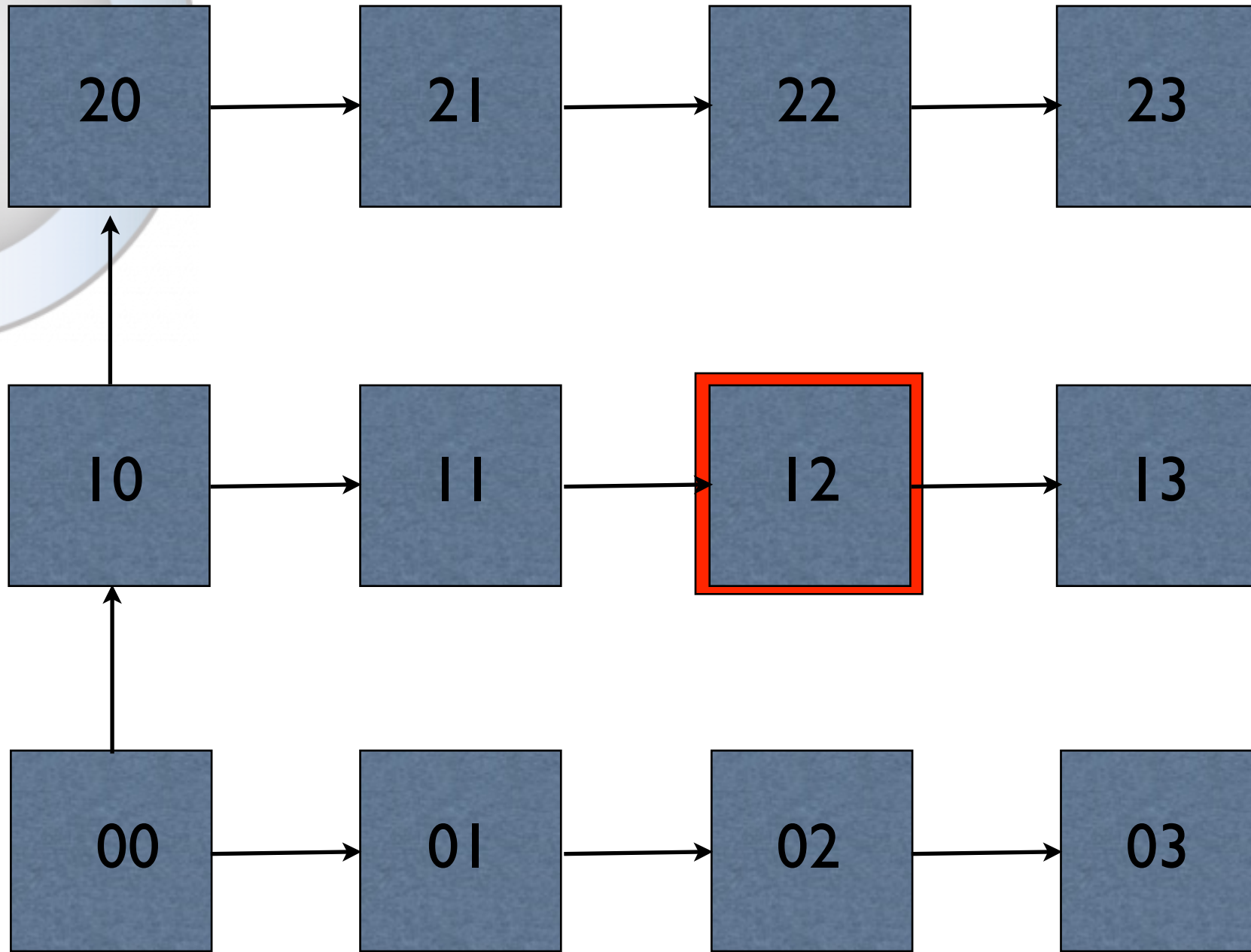
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i

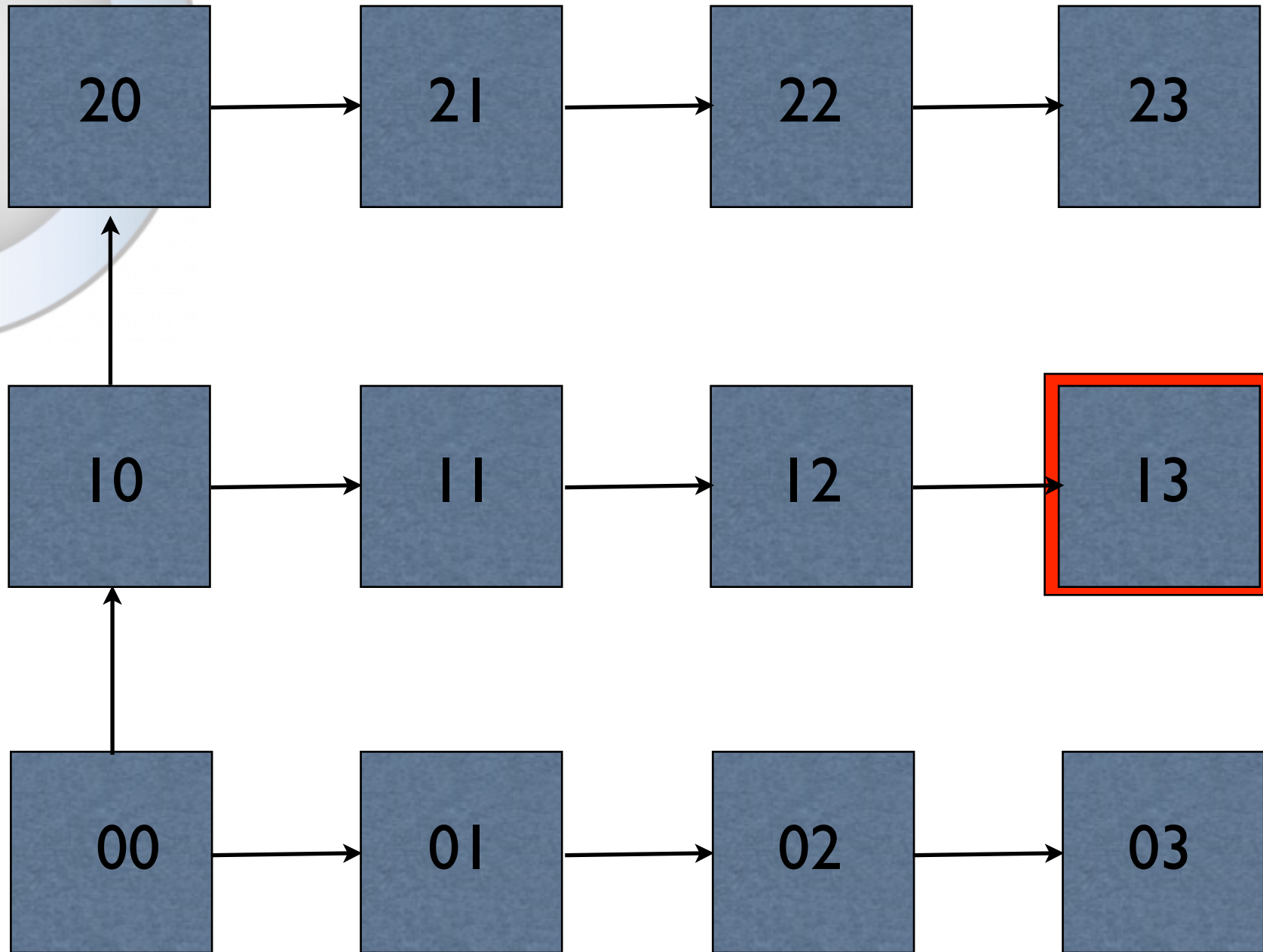


inheritance chain ----->



# The Comb Rule

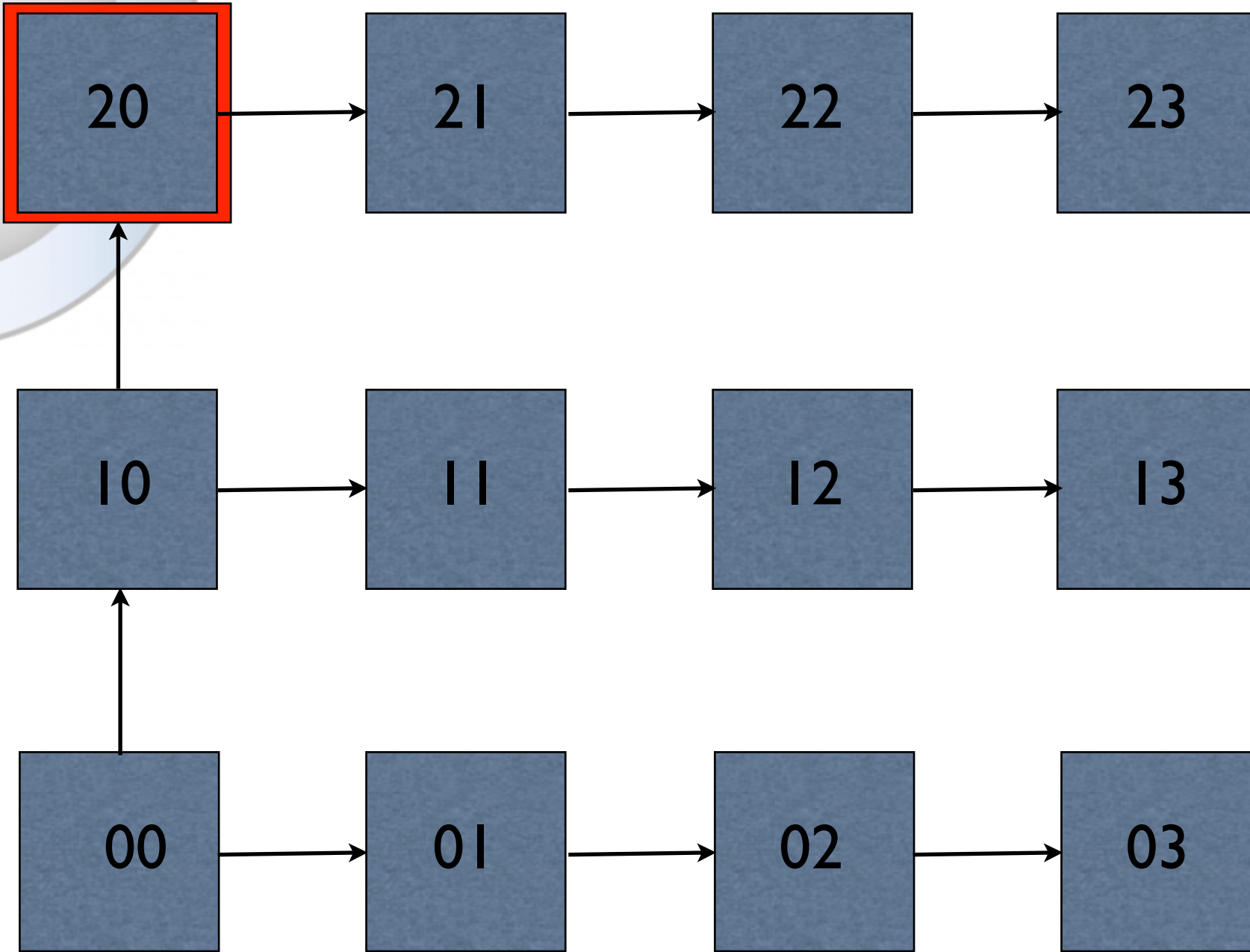
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

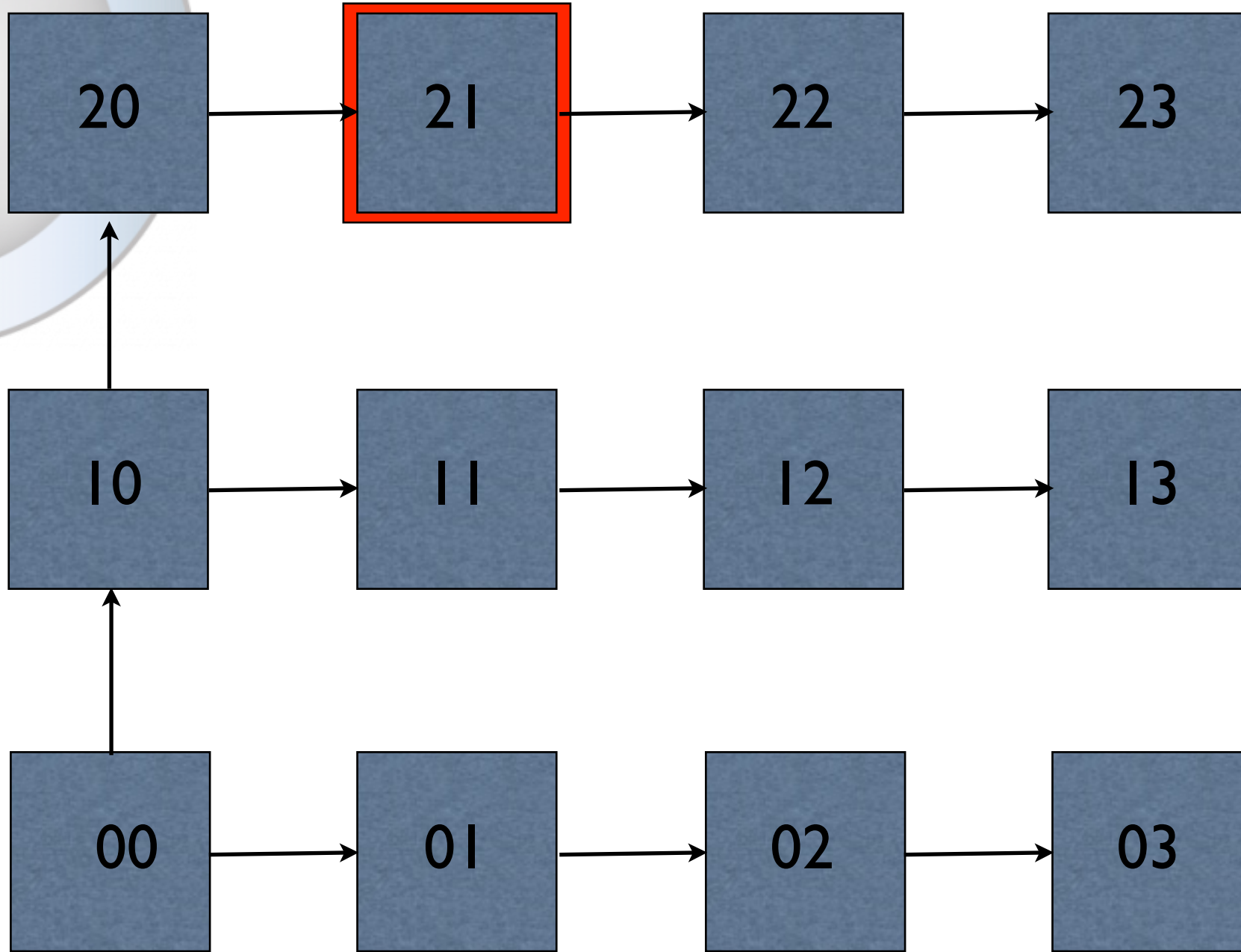
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

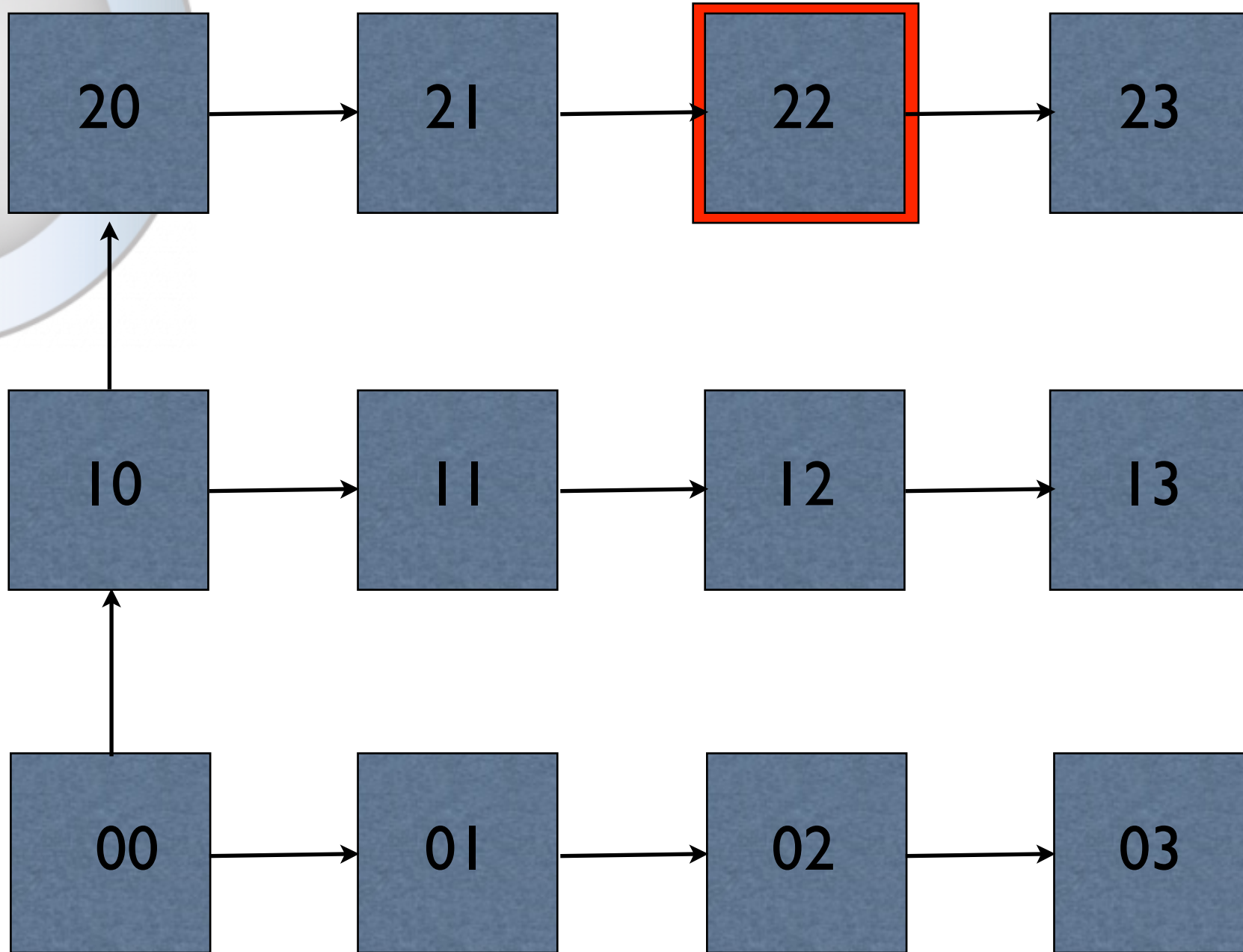
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

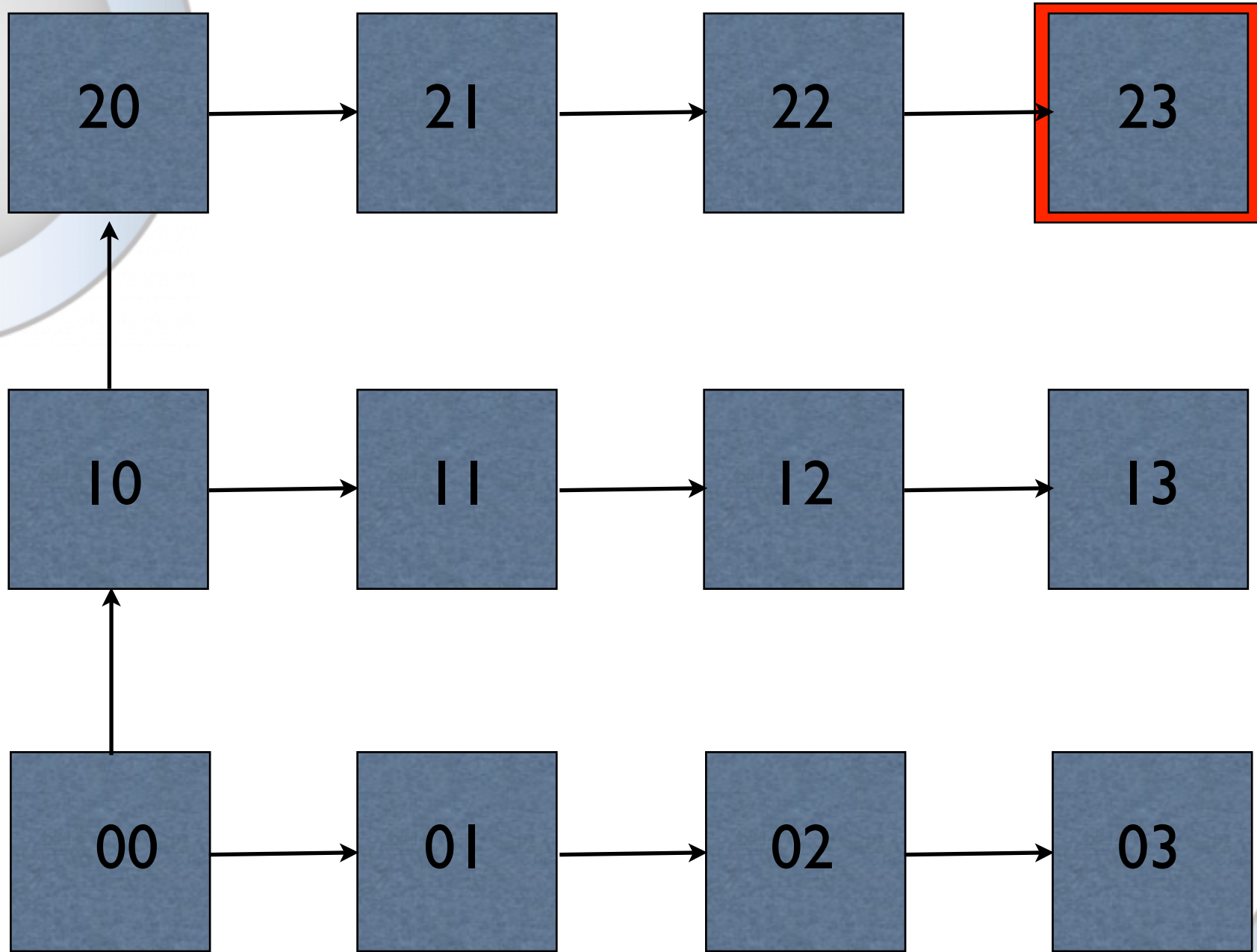
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Comb Rule

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# A Problem

```
class Sup ()()  
class Outer = (  
    m = (^91)  
    public class Inner = Sup () (  
        public foo = (^m)  
        “case 1:Outer new Inner new foo = 91”  
    )  
)
```

# When Code Evolves

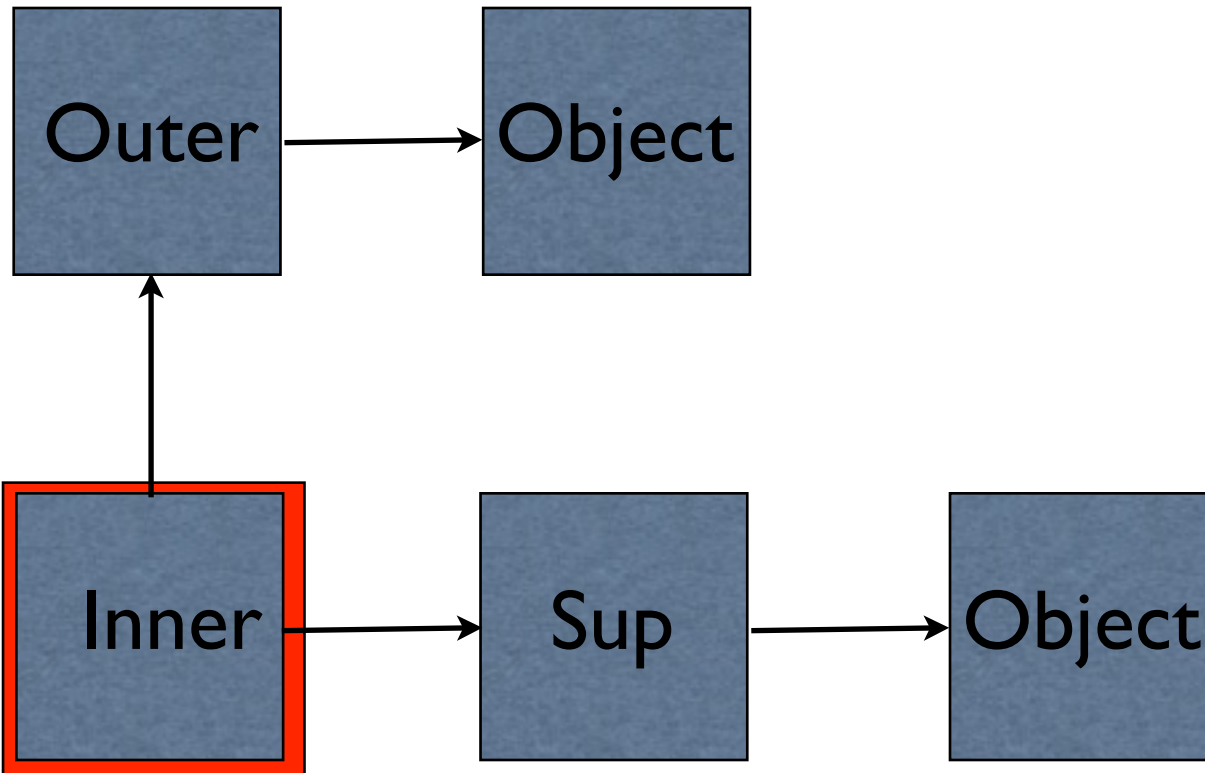
```
class Sup { int m { return 42;}}  
class Outer {  
    int m() { return 91;}  
    class Inner extends Sup {  
        int foo() {return m();}  
        // case 2: new Outer.Inner().foo() = 42  
    }  
}
```

# Searching for $m$

^  
|  
L  
e  
x  
i

c  
a  
l

c  
h  
a  
i



inheritance chain ----->

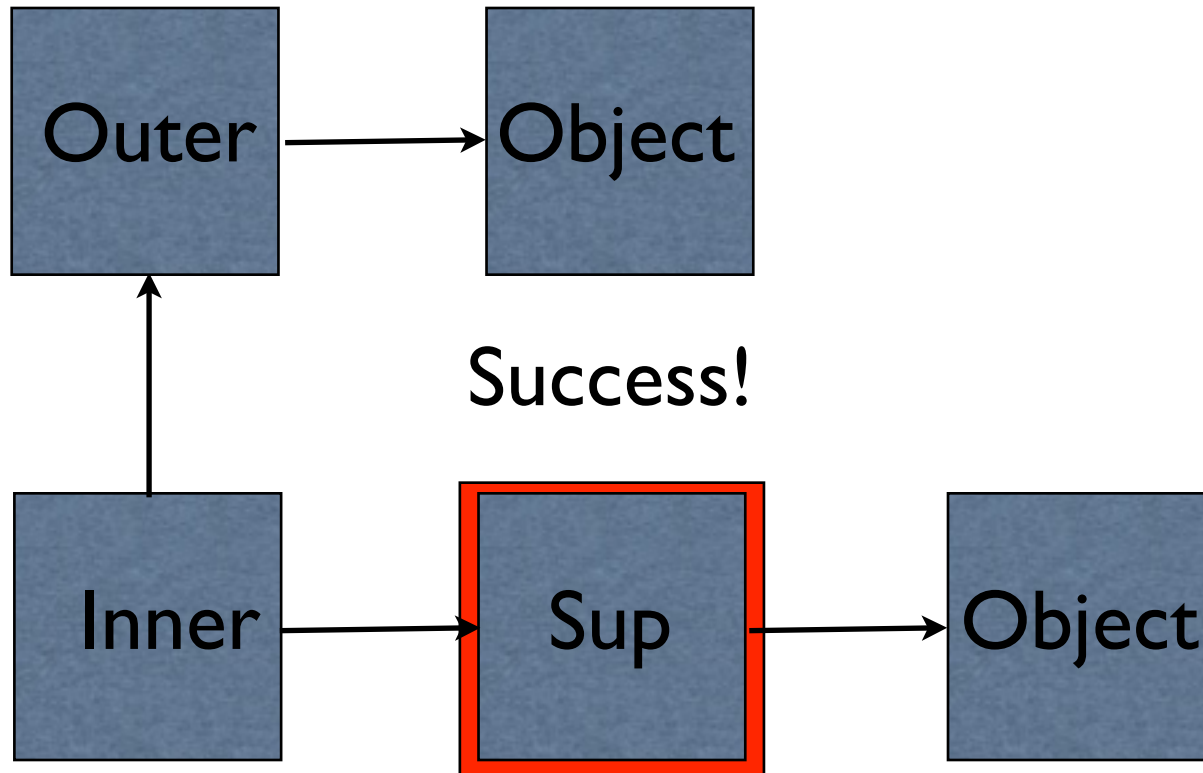


# Searching for $m$

^  
|  
L  
e  
x  
i

c  
a  
l

c  
h  
a  
i



inheritance chain ----->

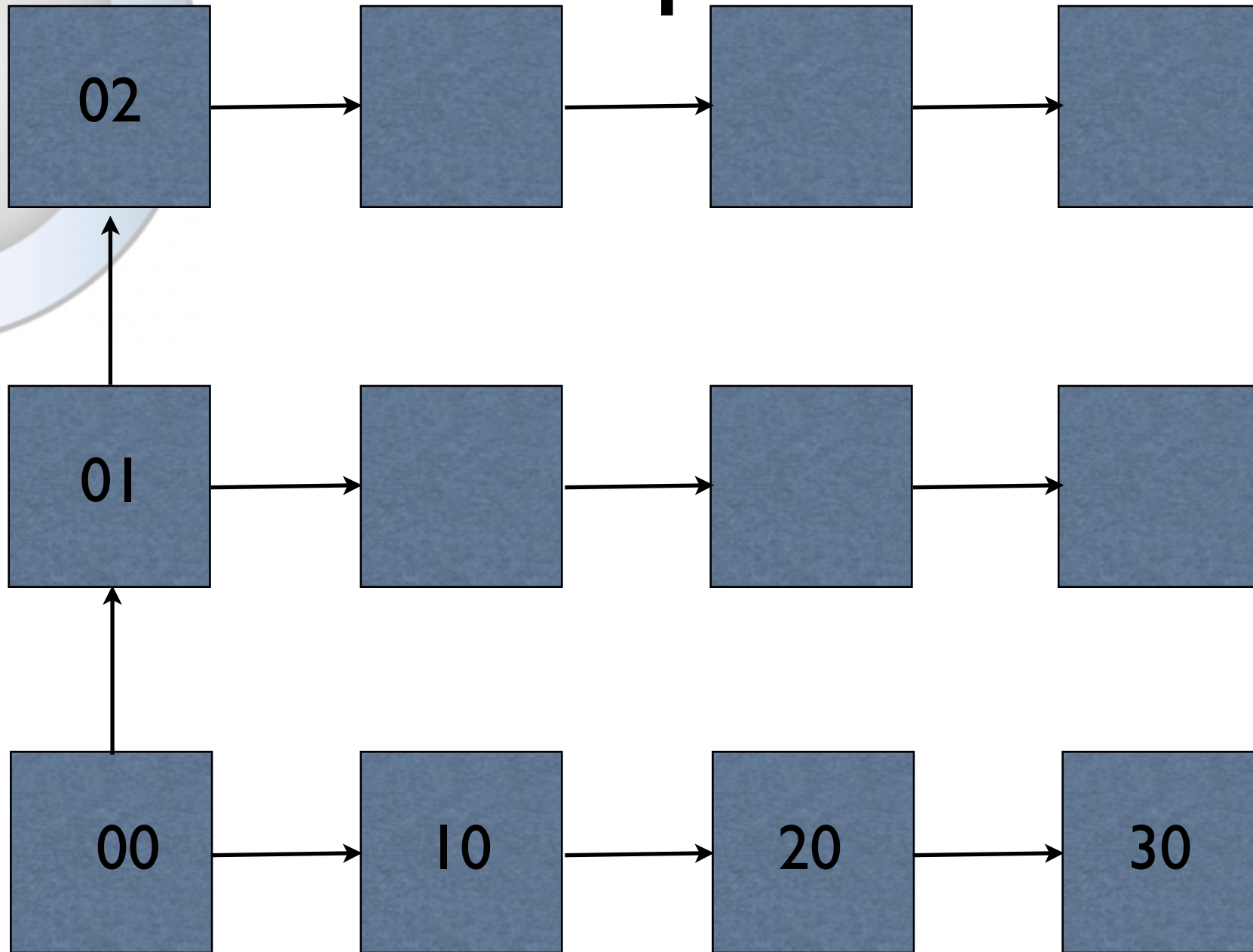


# Priority to Lexical Scope

- Newspeak gives priority to lexically visible declarations

# The Newspeak Rule

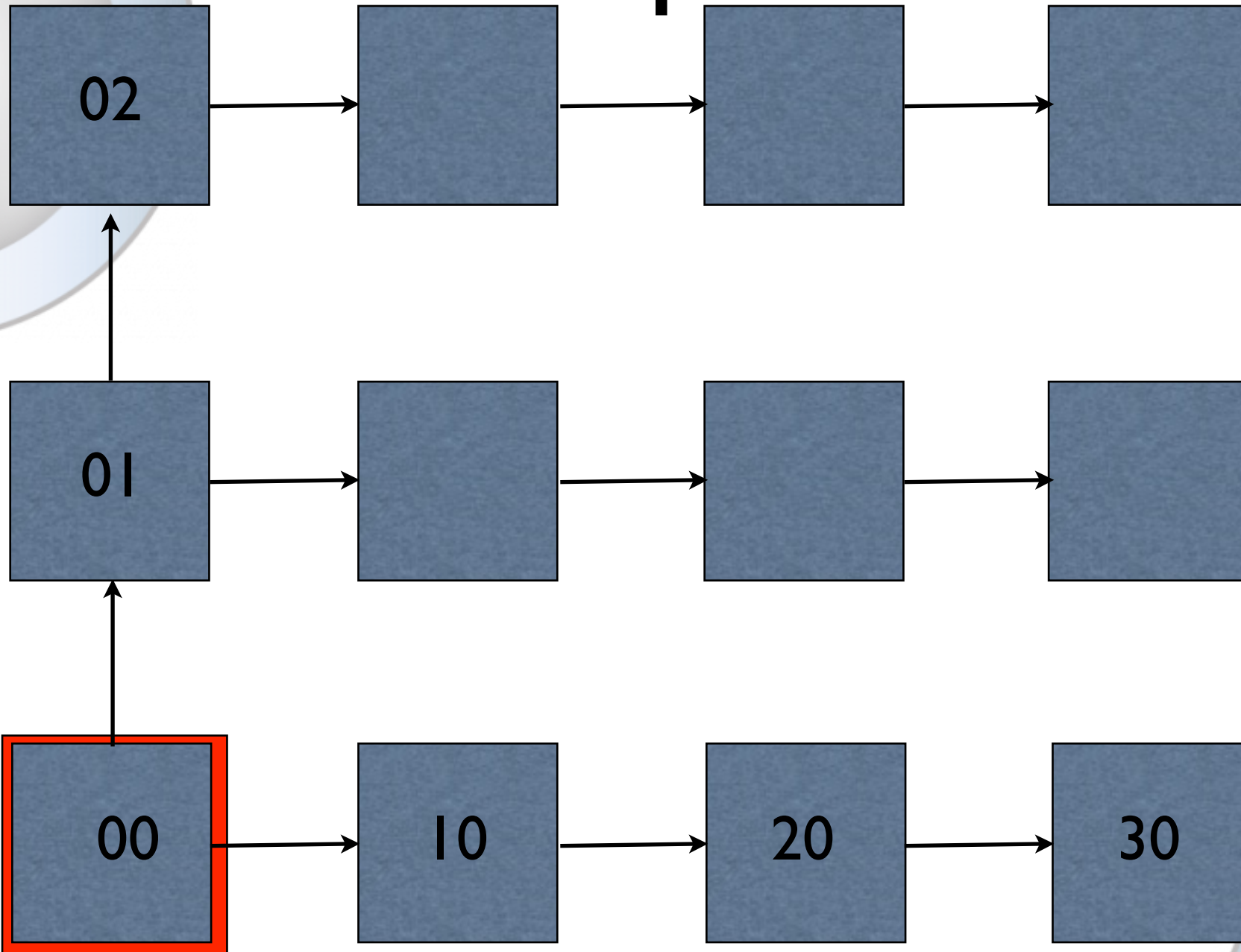
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

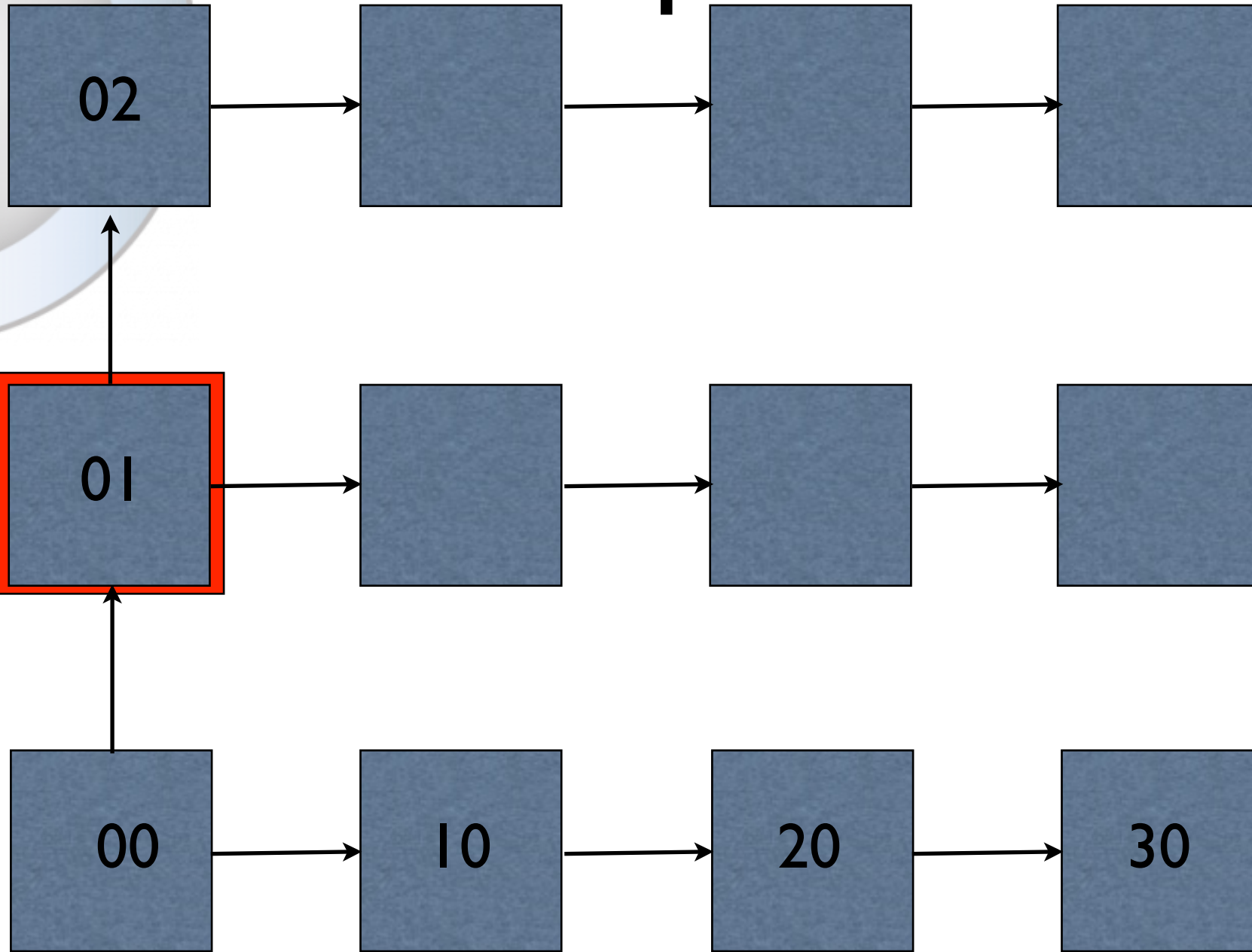
# The Newspeak Rule

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



# The Newspeak Rule

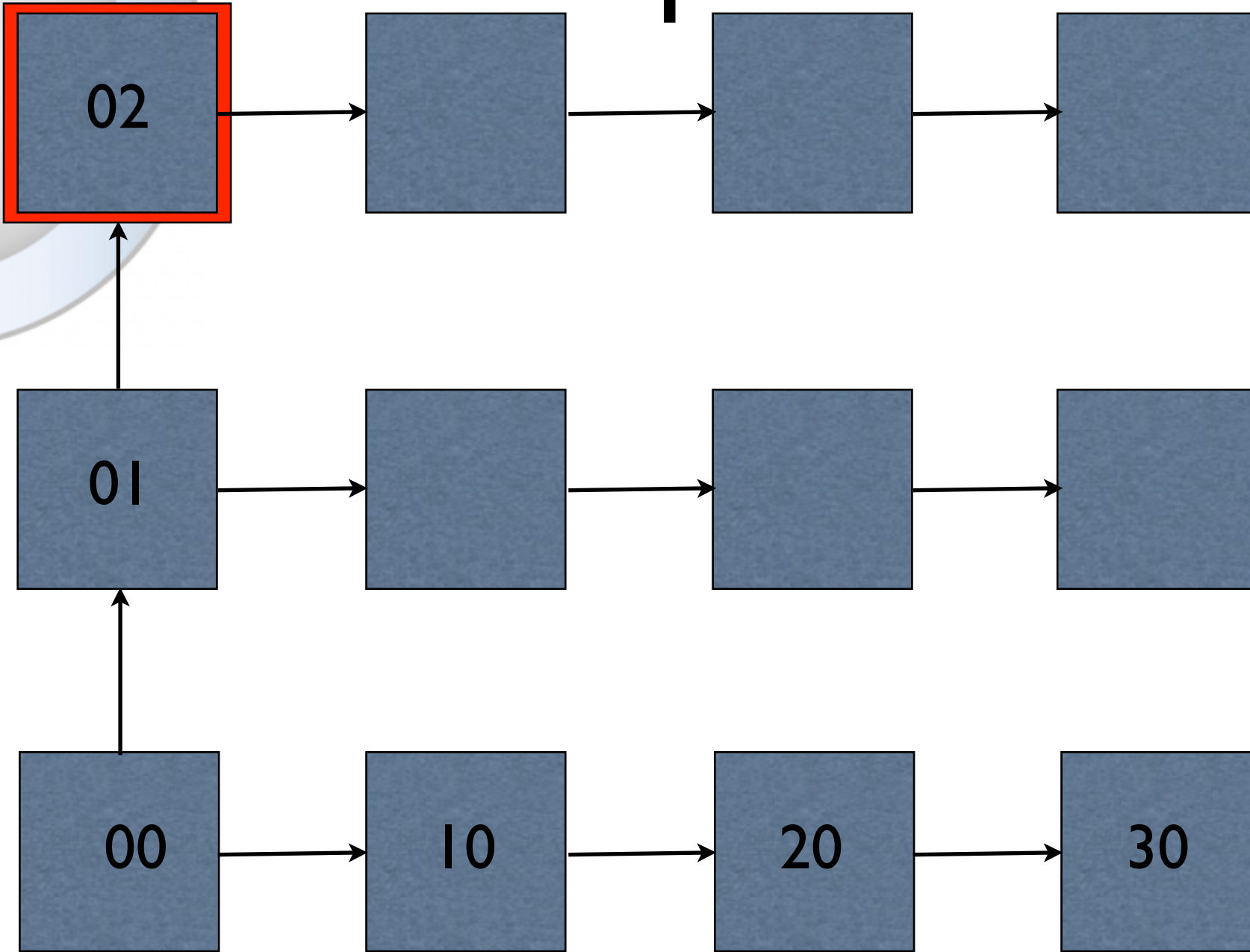
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Newspeak Rule

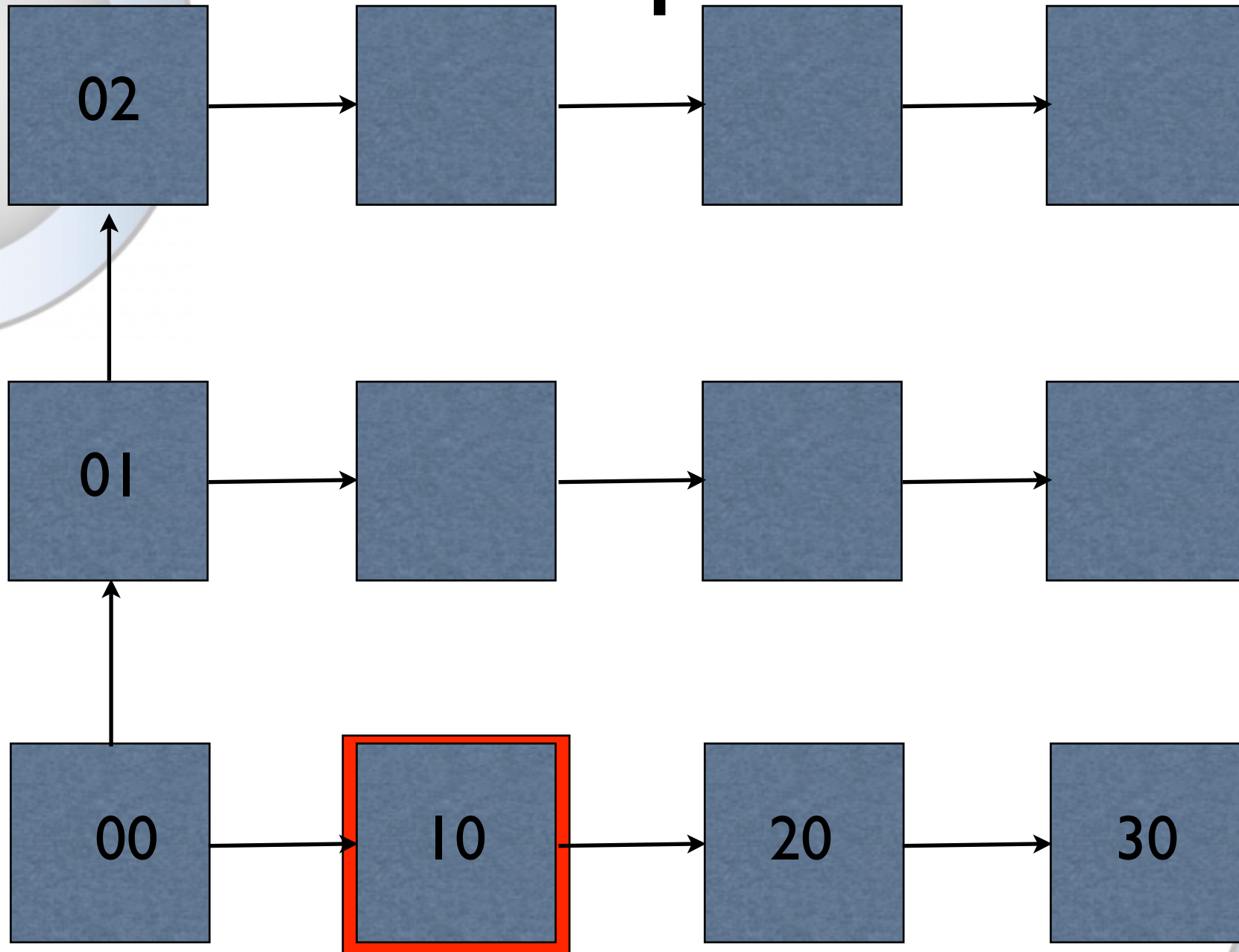
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Newspeak Rule

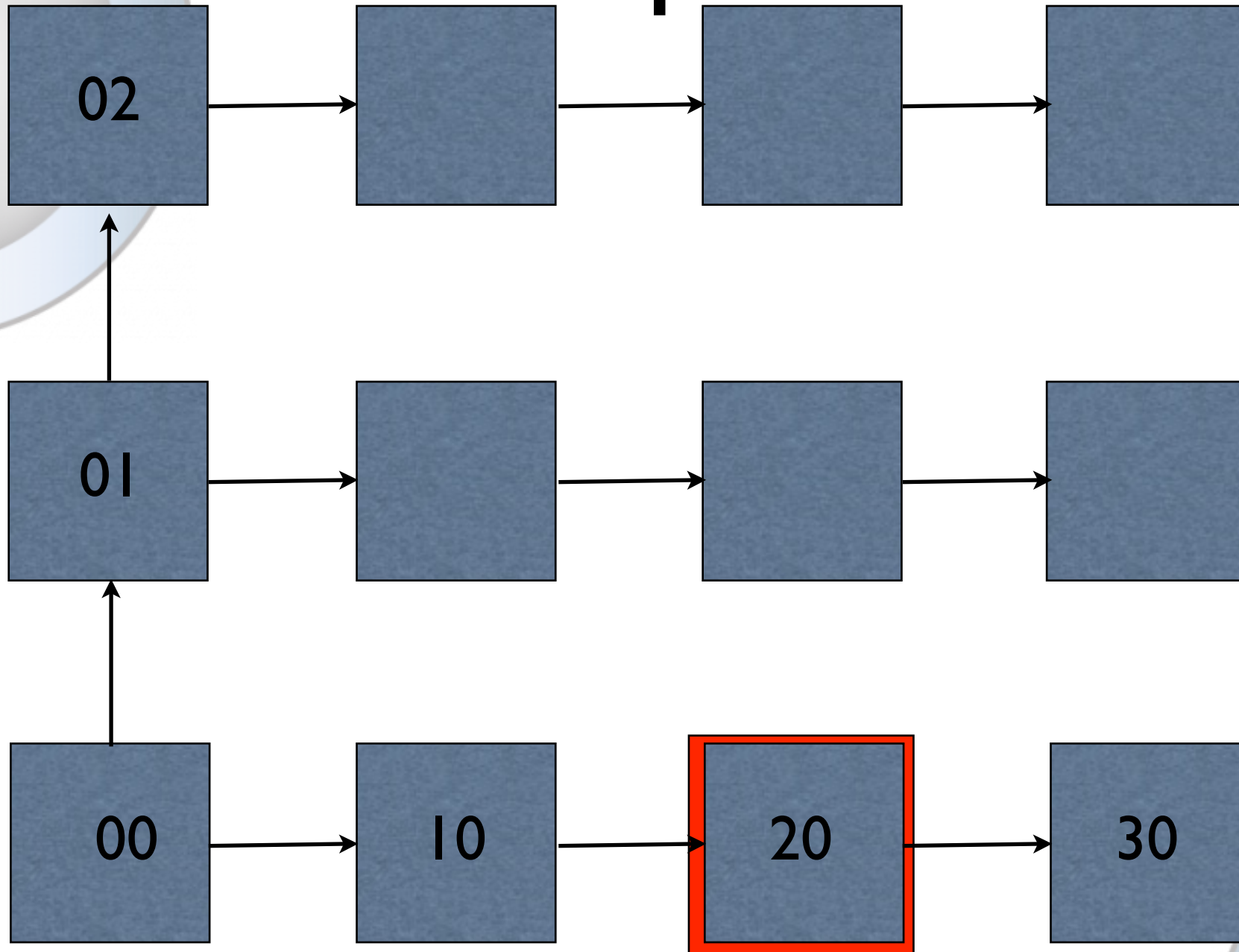
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Newspeak Rule

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i

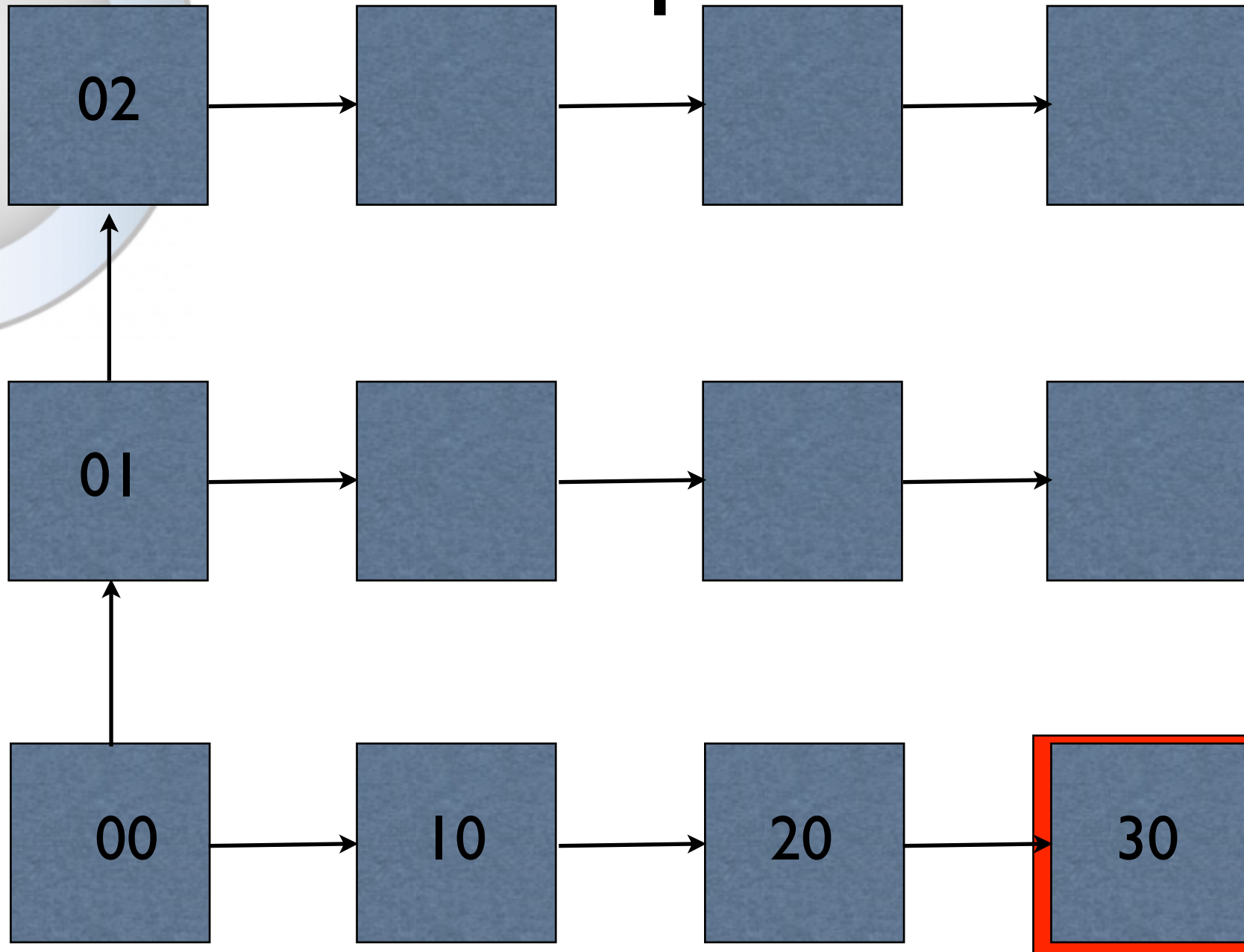


inheritance chain ----->



# The Newspeak Rule

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



# Priority to Lexical Scope

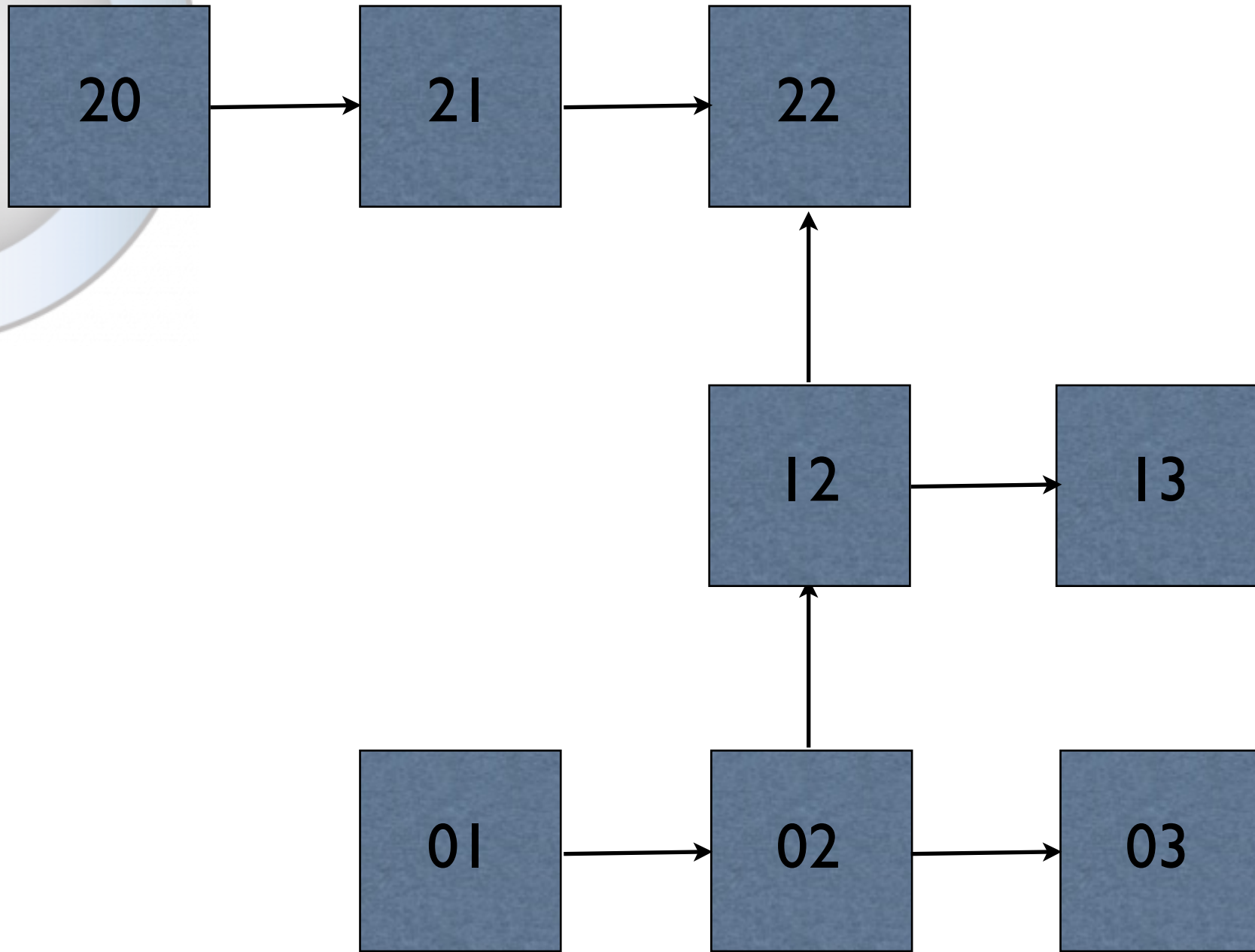
- Newspeak gives priority to lexically visible declarations
- But once a lexical level has been selected, inheritance can have an effect; lexical declaration can be overridden

# Real Hierarchies

- Can be more complex
- Lexical hierarchy may be subclassed differently at different levels
- Superclasses may in the same or different lexical hierarchies

# Real Hierarchies

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# Access Control in Newspeak

- Newspeak provides three levels of accessibility:
  - Public
  - Protected
  - Private

# Access Control in Newspeak

- Private and protected members can be seen by nested classes
- Enclosing classes cannot see private or protected members of nested classes
- Subclasses are never aware of private members of superclasses and vice versa

# Access Control in Newspeak

- An object may access a protected member only if
  - it is a member of the object or
  - A lexically visible member of an enclosing object

# Ordinary Sends

*e msg (\* aka e.msg \*)*

- Lookup *msg* in class of receiver; if **public**, execute. If **protected** fail (DNU). Ignore **private** versions
- If not found, recurse upwards (until **Object**).



# Self Sends

***self msg (\* aka this.msg \*)***

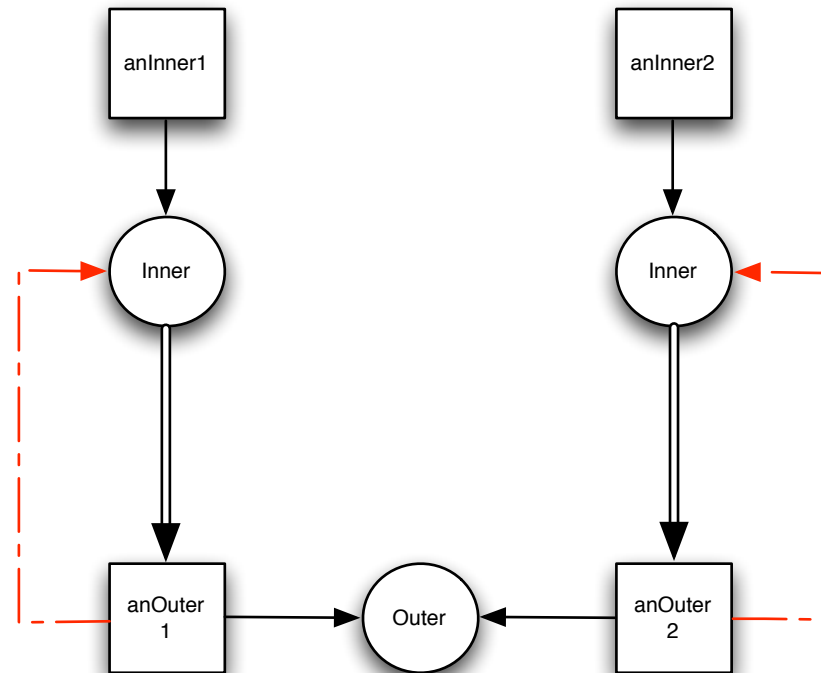
- If the immediately enclosing class declares a **private msg**, execute it
- Otherwise, lookup **public** or **protected msg** in class of the receiver; ignore **private** versions
- If not found, recurse upwards (until **Object**).

# Super Sends

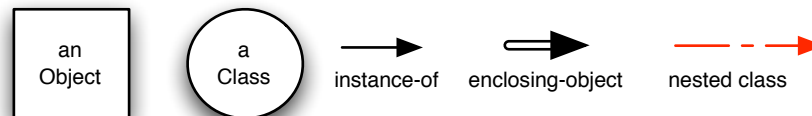
***super msg (\* aka super.msg \*)***

- Lookup **public** or **protected msg** in superclass of receiver; ignore **private** versions.
- If not found, recurse upwards (until **Object**).

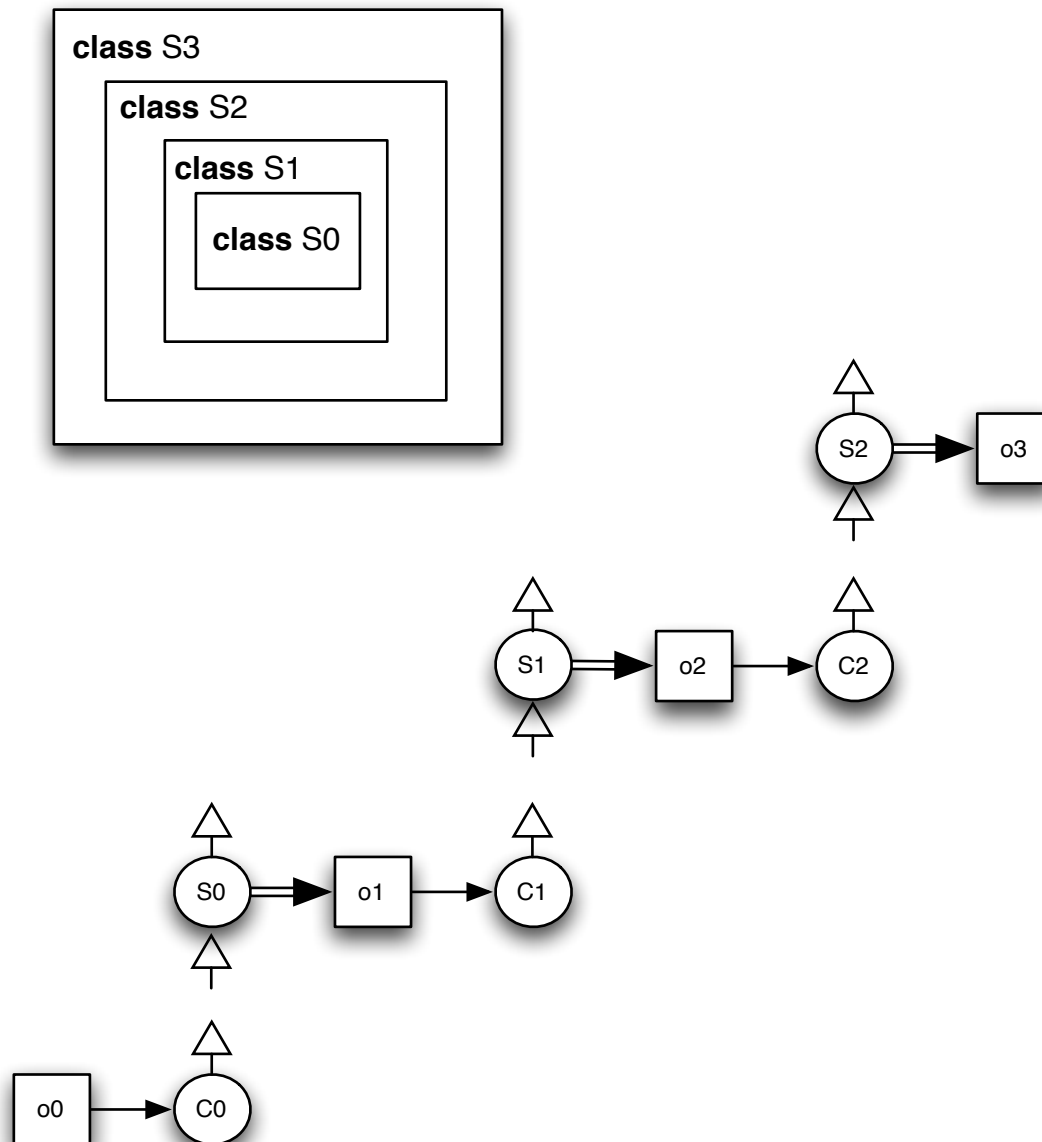
# Enclosing Objects



## Legend



# Enclosing Objects



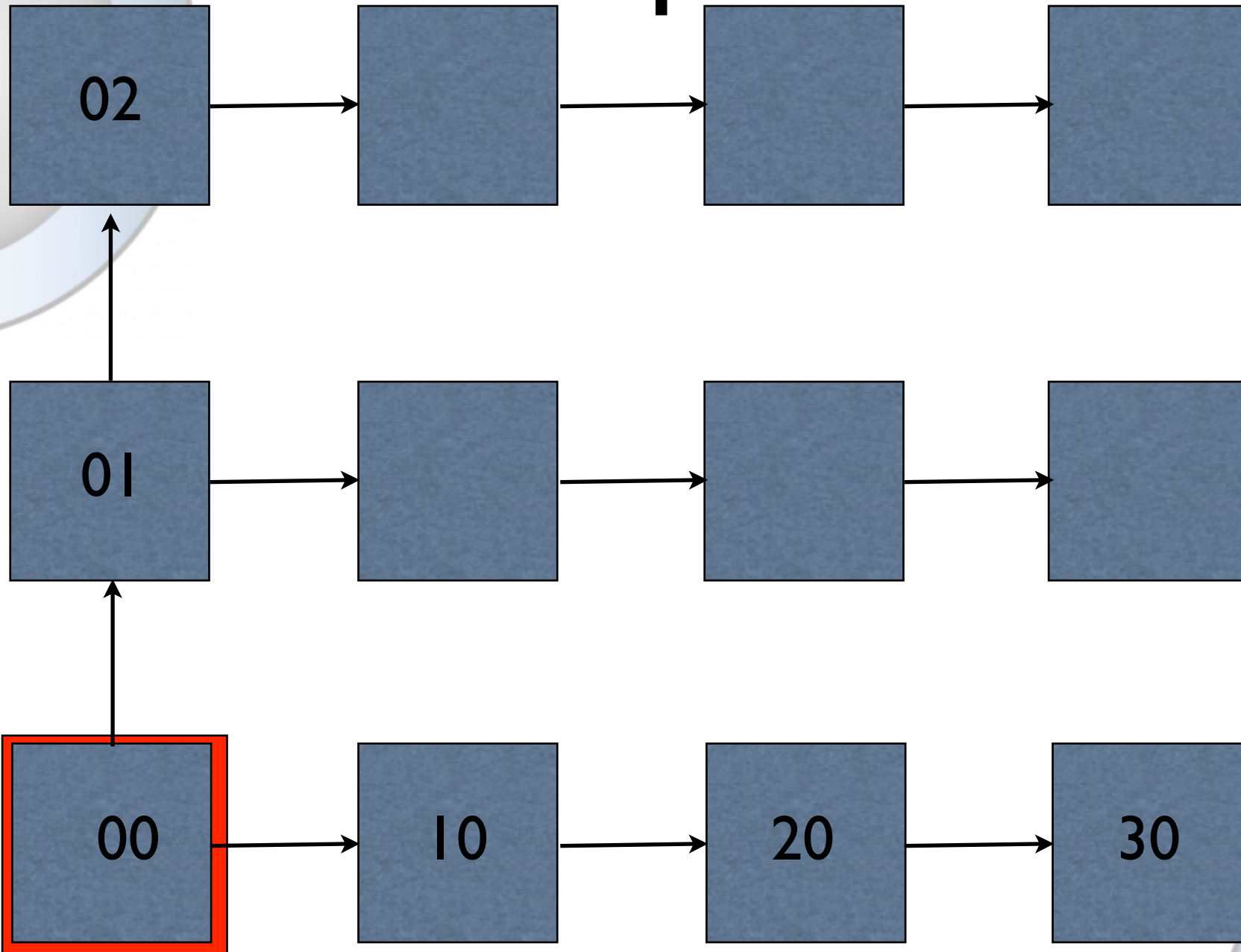
# Implicit Receiver Sends

***msg*** (\* aka *msg* \*)

- Lookup declaration of ***msg*** in immediately surrounding class.
- If not found, recurse up lexical scope until top.

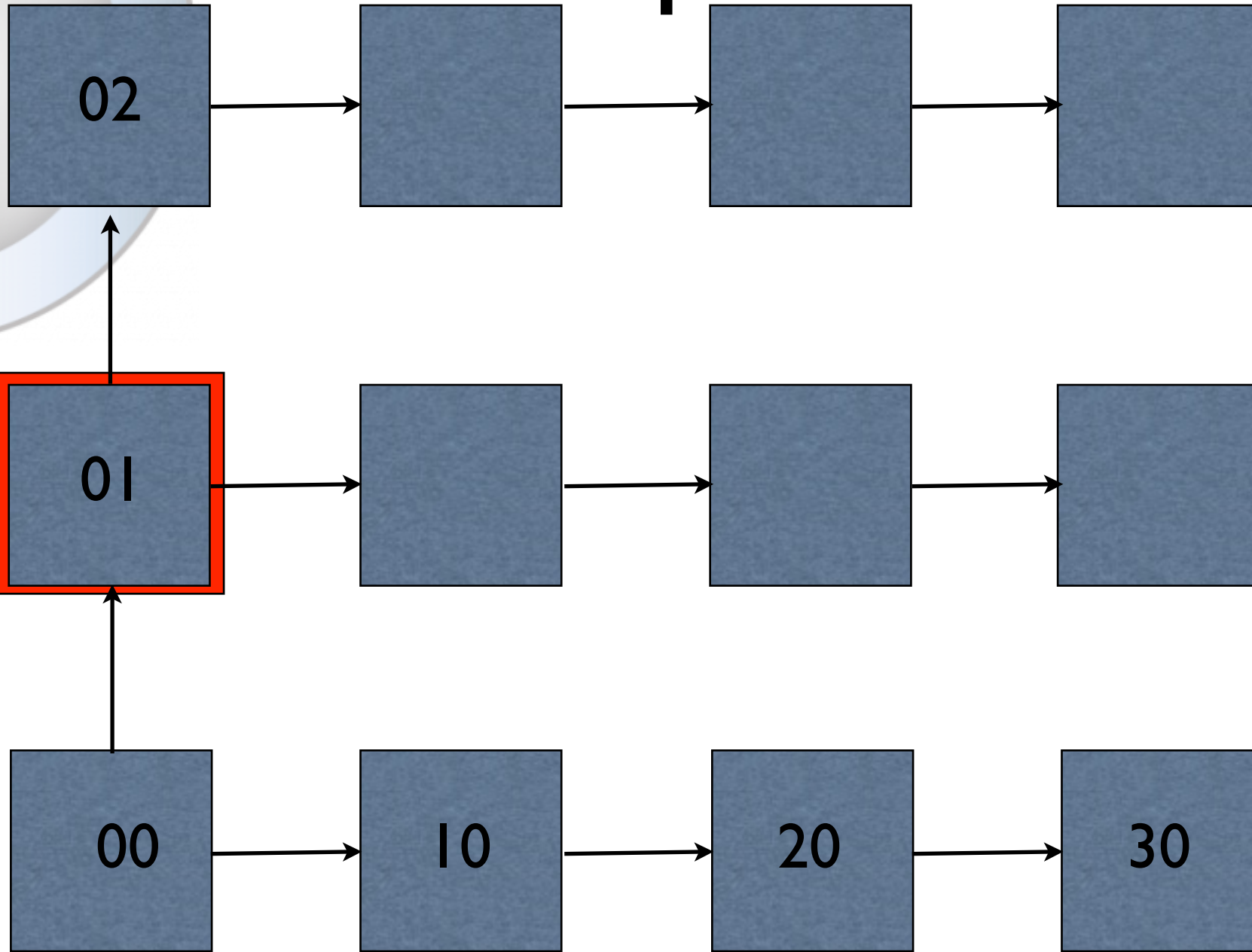
# The Newspeak Rule

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



# The Newspeak Rule

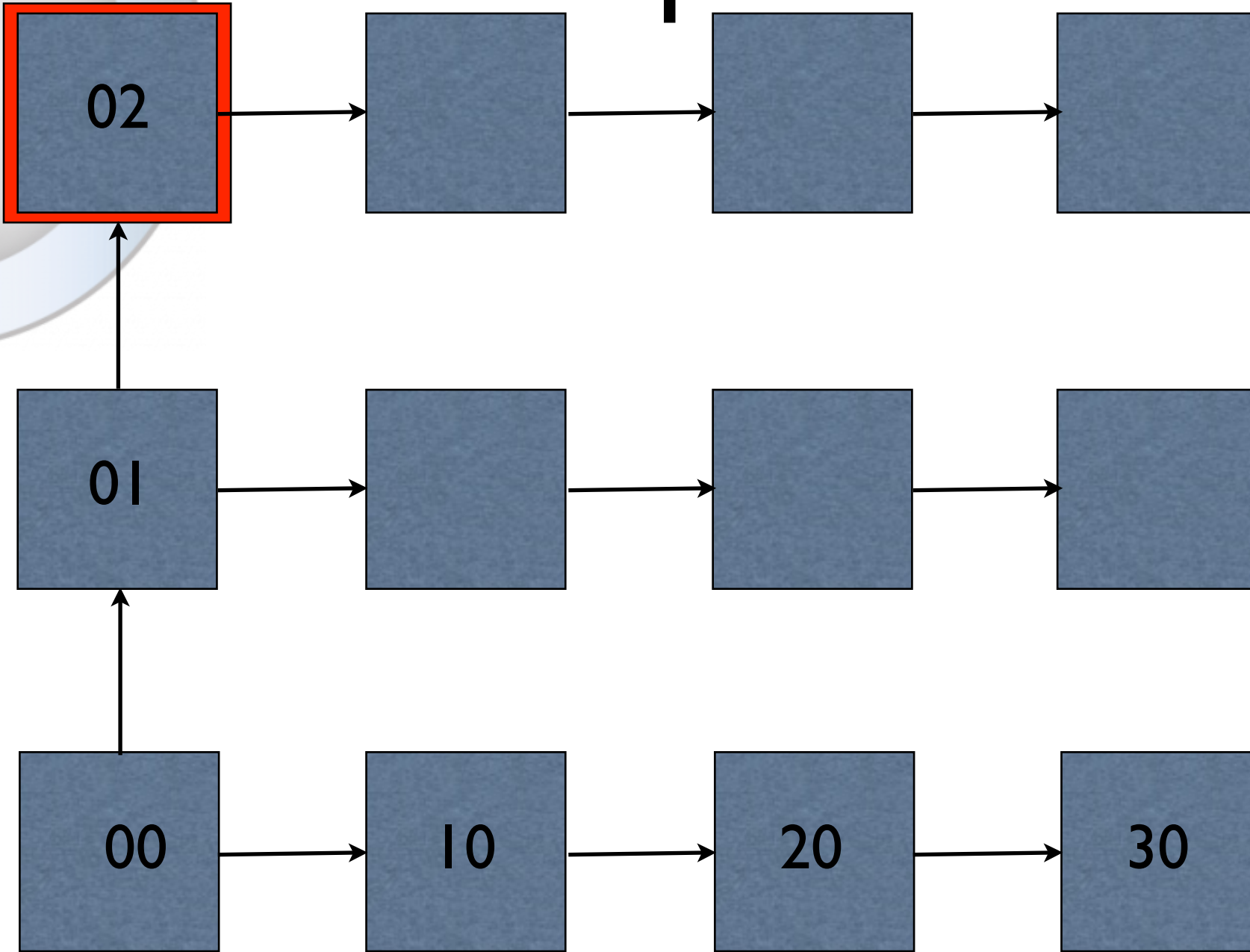
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Newspeak Rule

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->



# Implicit Receiver Sends

*msg* (\* aka *msg* \*)

- If a lexically visible declaration has been found, then if *msg* is **private**, execute.
- If *msg* is not **private**, let *r* be corresponding enclosing object. lookup **public** or **protected** *msg* in class of *r*
- if not found recurse upwards (until **Object**).

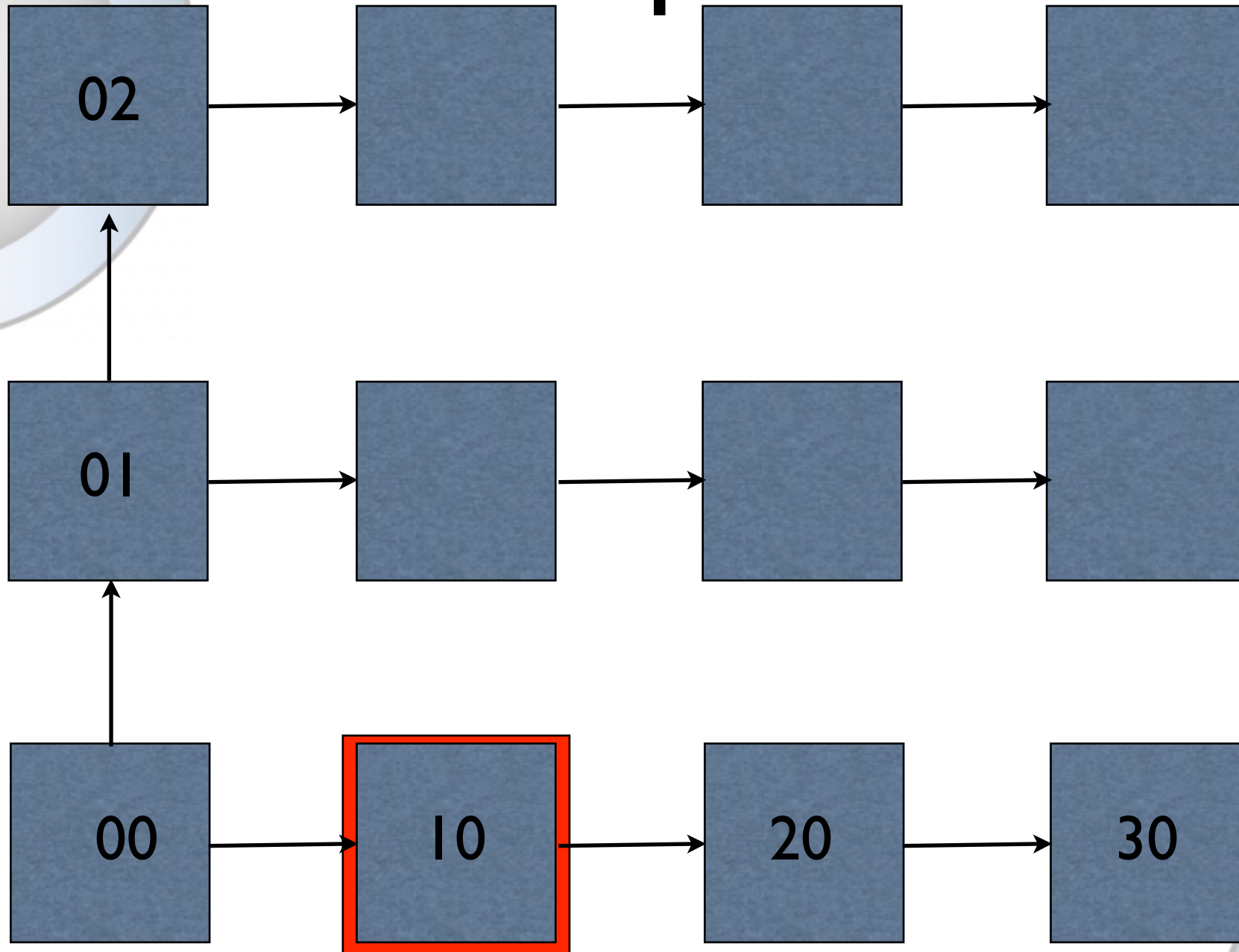
# Implicit Receiver Sends

*msg (\* aka msg \*)*

- If no lexically visible declaration was found, lookup **public** or **protected** *msg* in superclass of receiver; ignore **private** versions.
- If not found, recurse upwards (until ***Object***).

# The Newspeak Rule

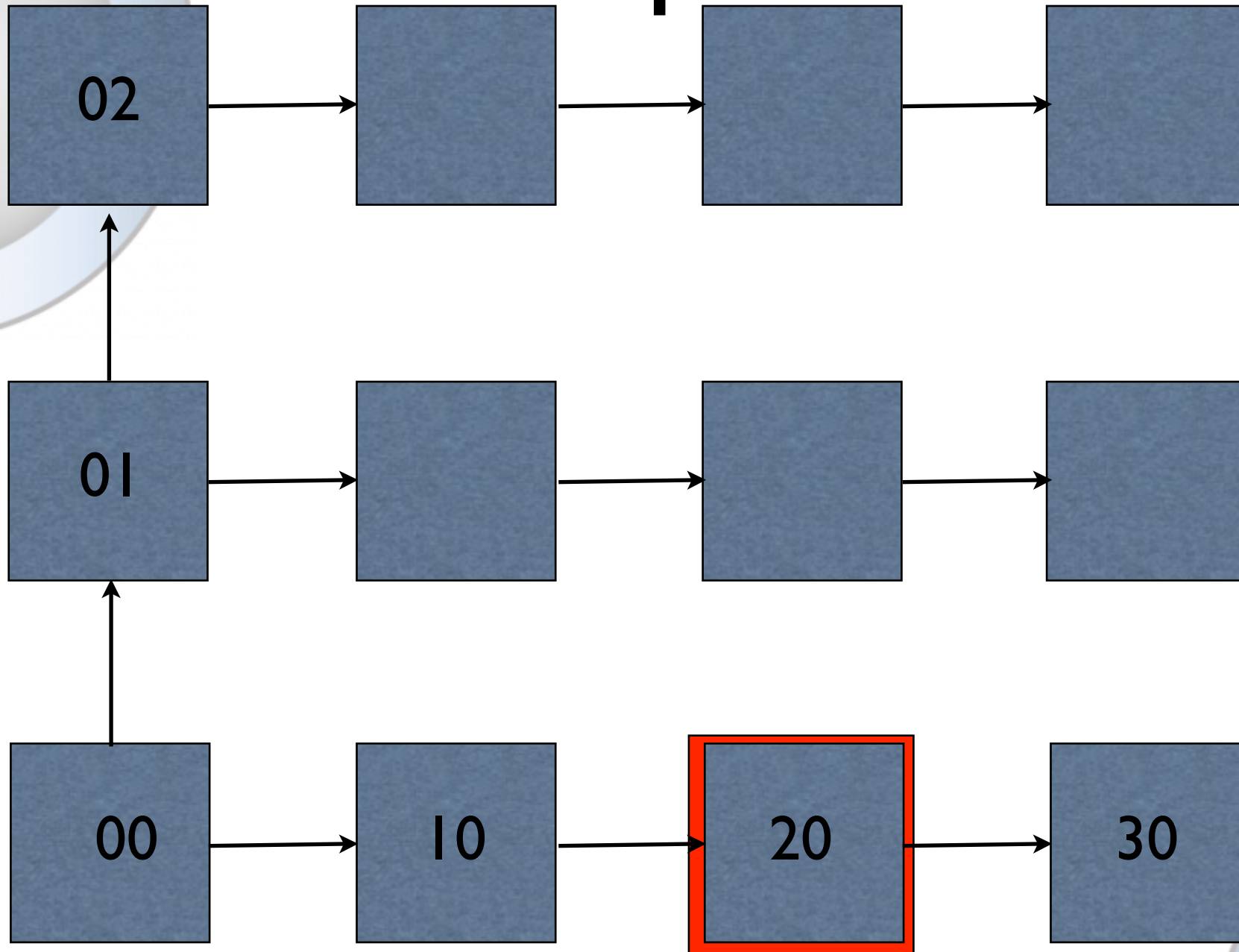
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Newspeak Rule

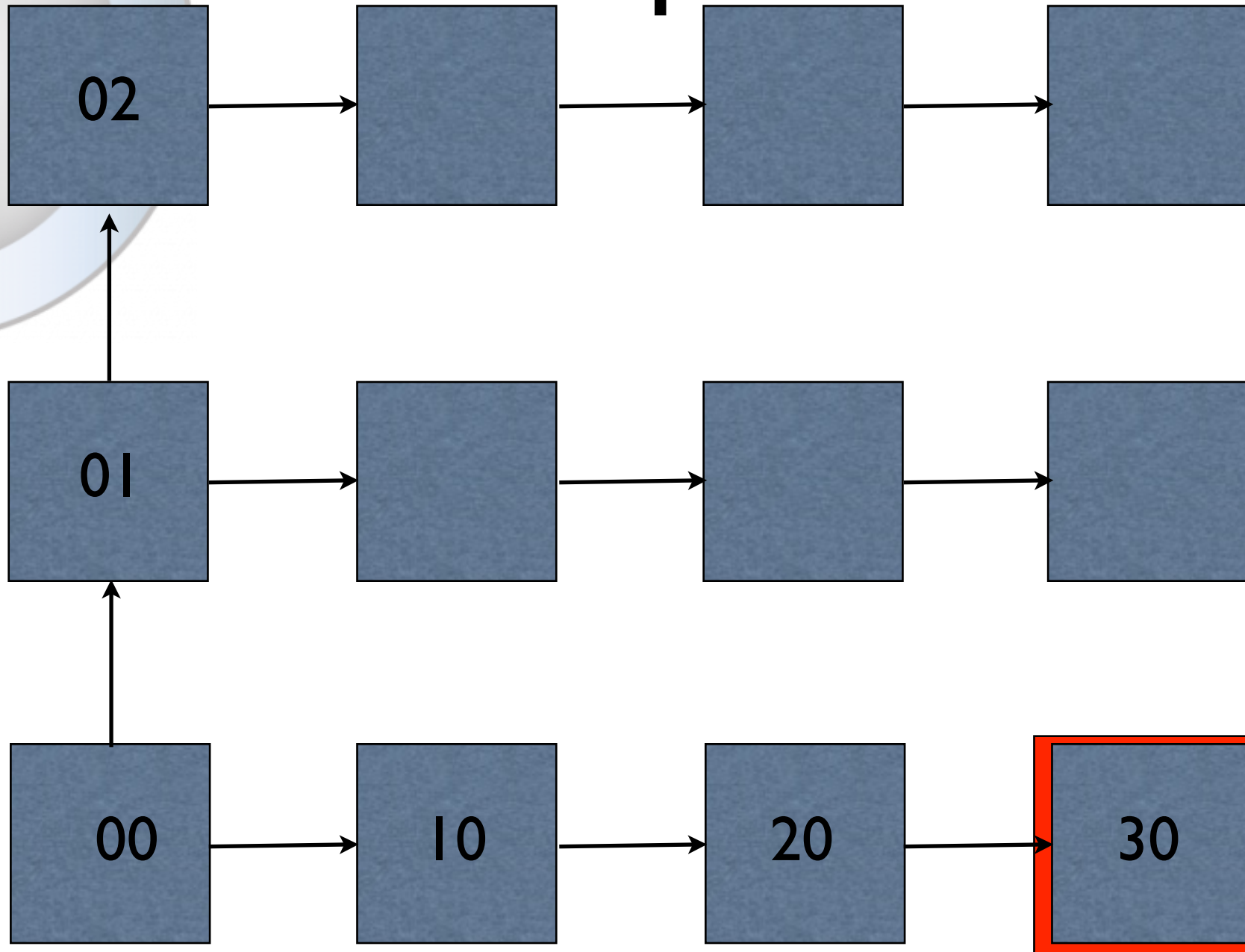
^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# The Newspeak Rule

^  
|  
L  
e  
x  
i  
c  
a  
l  
  
c  
h  
a  
i



inheritance chain ----->

# Status & Implementation

- Access control implemented in both Newspeak-to-Javascript compiler and in Newspeak VM interpreter (but not yet in JIT).
- Each send has corresponding byte code in VM version
- Platform mostly converted

# Experience

- The Newspeak platform includes GUI, IDE, Core libraries.
- Code base developed without access control
- Conversion effort: a few person weeks
- In some cases we have over publicized

# Further Reading

- <http://newspeaklanguage.org>
- ECOOP 2010 paper is under documents



# Related Work

- ⦿ Self
- ⦿ Smalltalk
- ⦿ Beta, gBeta, Virtual Classes
- ⦿ PLT Scheme/Units
- ⦿ Scala
- ⦿ ML
- ⦿ CLU, Modula, Ada, Oberon ...
- ⦿ Much more

# Credits

- Peter Ahe
- Vassili Bykov
- Felix Geller
- Yaron Kashai
- Matthias Kleine
- Ryan Macnak
- Bill Maddox
- Eliot Miranda
- Philipp Tessenow
- Bob Westergaard

# Volunteers

● Joshua  
Benuck

● Nikolay  
Botev

● Luis Diego  
Fallas

● John  
Hedditch

● Raffaello  
Giulietti

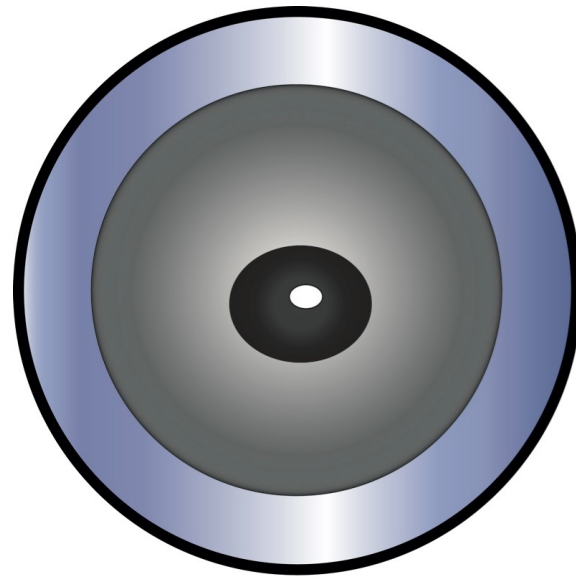
● Yardena  
Meymann

● Stephen Pair

● David Pennell

● Steve Rees


● Vadim Tsushko



# Newspeak

It's double plus good

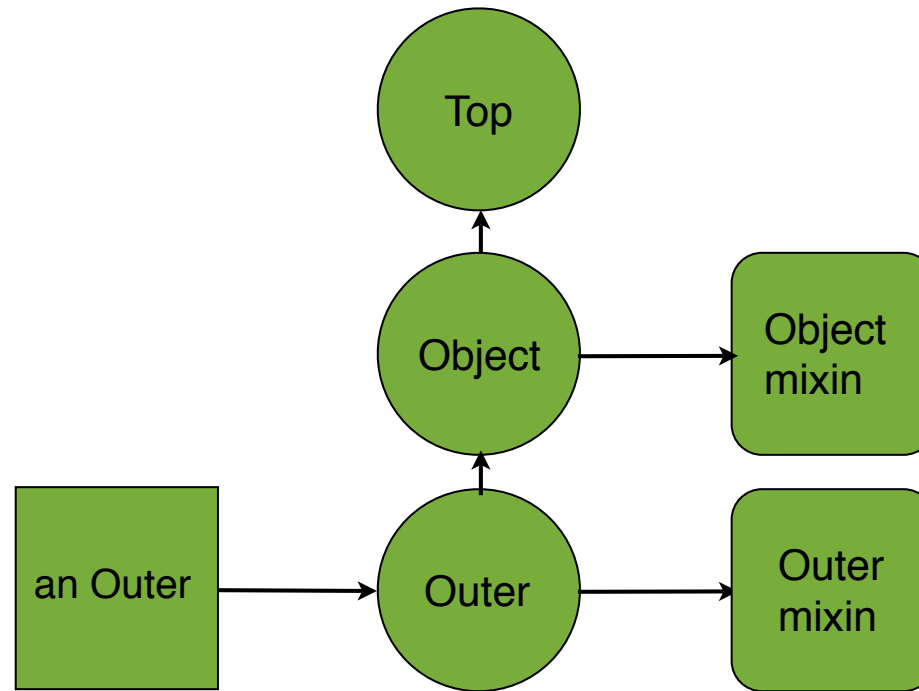
*This file is licensed under the [Creative Commons Attribution ShareAlike 3.0 License](#). In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license identical to this one. [Official license](#).*

*The Newspeak eye  used in the bullets, slide background etc. was designed by Victoria Bracha and is used by permission. Newspeak doubleplusgood logo designed by Hila Rachmian and used by permission.*



# Backup

# Classes and Mixins



# Outer Sends

*outer N msg (\* no obvious analog \*)*

- ① Find innermost enclosing class named **C**. If a **private msg** is defined in **C**, execute.
- ② If not found, let *r* be corresponding enclosing object. Lookup **public** or **protected msg** in class of *r*; ignore **private** versions.
- ③ If not found, recurse upwards (until **Object**).



# Observations

***self msg ~ outer C msg***

*where C is immediately enclosing class*

***msg ~ outer N msg***

*where N is innermost lexically enclosing class that declares msg*

# Common Operations

- Protected lookup
  - Lookup **public** or **protected *msg*** in a class; ignore **private** versions.
  - If not found, recurse up the class hierarchy (until ***Object***).
- Used in super, outer (and self/implicit receiver) sends

# Common Operations

- Find enclosing object
  - Lookup *msg* in class of receiver; if **public**, execute. If **protected** fail (DNU). Ignore **private** versions.
- Used in ordinary sends