Protocols

What are Protocols?

- A protocol is a set of possibly interacting functions.

- Example:
  (defgeneric read-bytes (stream n))
  (defgeneric read-byte (stream))

Possible Interactions

- (defgeneric read-bytes (stream n))
  (defgeneric read-byte (stream))

- Option 1:
  (defmethod read-bytes (stream n)
  (let ((array (make-array n)))
  (loop for i below n do (setf (aref array i) (read-byte stream)))
  array))

Possible Interactions

- (defgeneric read-bytes (stream n))
  (defgeneric read-byte (stream))

- Option 2:
  (defmethod read-byte (stream)
  (let ((array (read-bytes 1)))
  (aref array 0)))

Possible Interactions

- (defgeneric read-bytes (stream n))
  (defgeneric read-byte (stream))

- Option 3: No interactions.
  (The two generic functions are independent.)

- Why does it matter?
  You have to know what you can override and what effects overriding has.

Protocols in CLOS

- CLOS defines some protocols for its operators.

make-instance

- (defmethod make-instance (class &rest initargs)

  ...

  (let ((instance (apply #’allocate-instance class initargs)))

    (apply #’initialize-instance instance initargs)

    instance))

initialize-instance

- (defmethod initialize-instance
    ((instance standard-object) &rest initargs)
    (apply #’shared-initialize instance t initargs))

Initialization Protocol

- make-instance calls allocate-instance and then initialize-instance.
- allocate-instance allocates storage.
- initialize-instance calls shared-initialize.
- shared-initialize assigns the slots.

change-class

- change-class modifies the object to be an instance of a different class and then calls update-instance-for-different-class.

- update-instance-for-different-class calls shared-initialize.

reinitialize-instance

- reinitialize-instance calls shared-initialize.

Class Redefinition

- When a class is redefined, make-instances-obsolete may be called.
- make-instances-obsolete ensures that update-instance-for-redefined-class is called for each instance.
- update-instance-for-redefined-class calls shared-initialize.
- It’s not specified when these steps happen, only that they happen!

Specialize Initialization

- You can specify methods on initialize-instance, reinitialize-instance, update-instance-for-different-class, update-instance-for-redefined-class and shared-initialize.

- Note: At least one required parameter should be specialized on one of your own classes!

Example

- (defclass circle ()
    ((radius :initarg :radius :accessor radius)
     (area :accessor area)))

- (defmethod initialize-instance :after
    ((object circle) &key radius)
  (setf (area object) (* pi radius radius)))

... or ...

- (defclass circle ()
    ((radius :initarg :radius :accessor radius)
     (area :accessor area)))

- (defmethod shared-initialize :after
    ((object circle) slot-names &key radius)
  (declare (ignore slot-names))
  (setf (area object) (* pi radius radius)))

Slot Access Errors

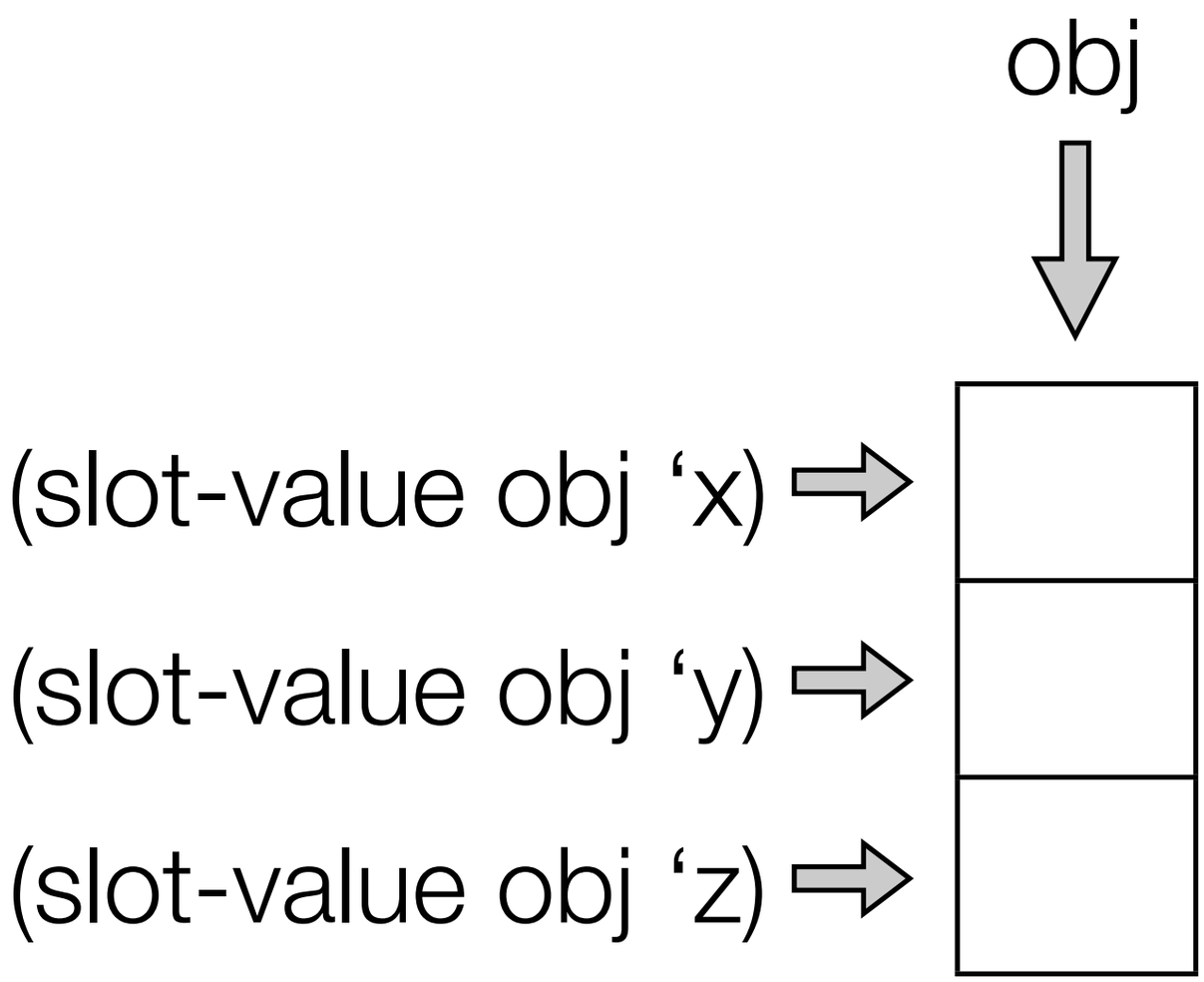
- When a slot to be accessed is unbound, slot-unbound is called.
- When a slot to be accessed is not defined, slot-missing is called.
- By default, these two functions signal errors.
- Example: simple-hash-class.lisp

The CLOS Metaobject Protocol

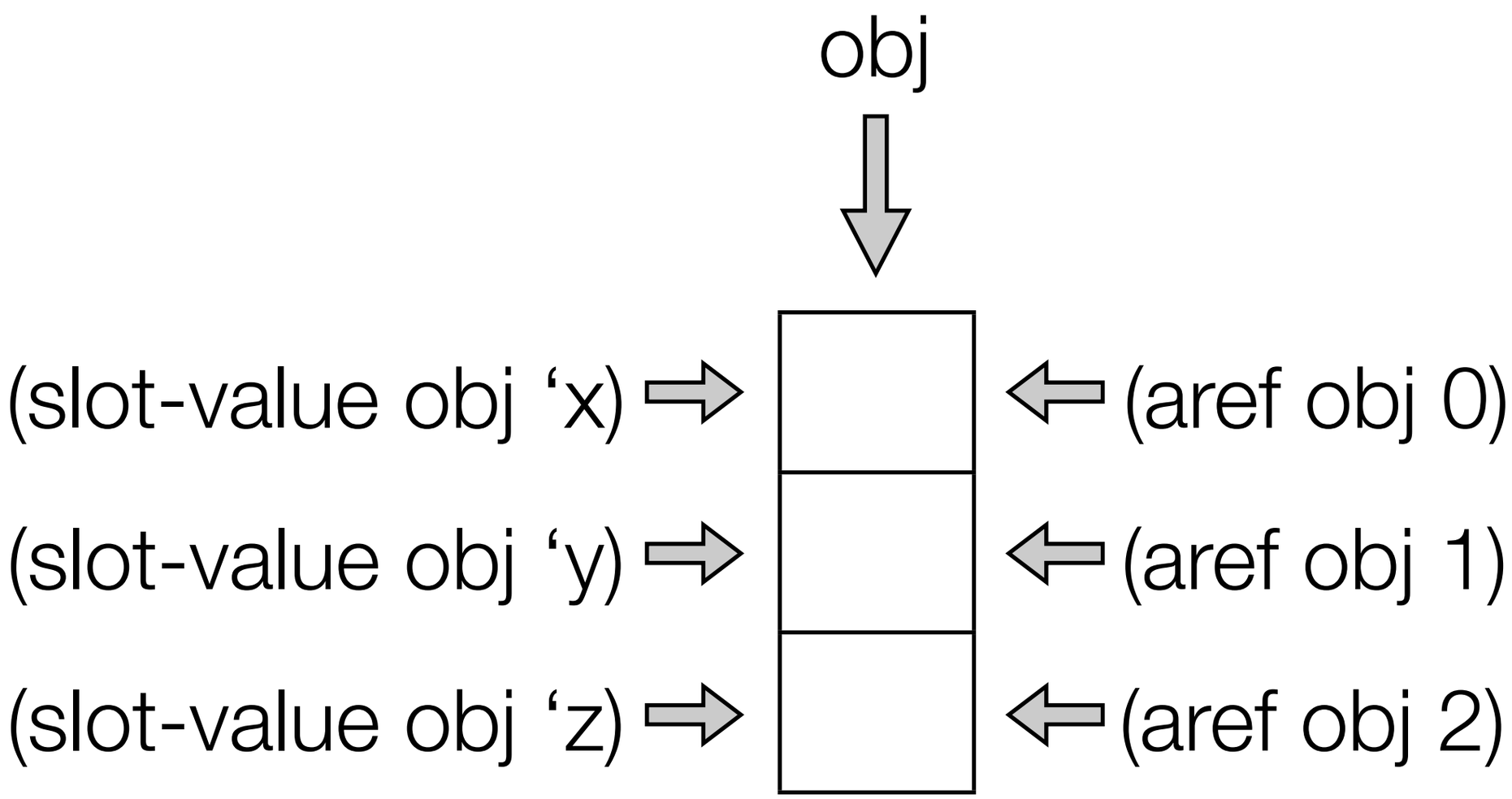
“An object has state, behavior, and identity.”

(Grady Booch, 1991)

OOP: State



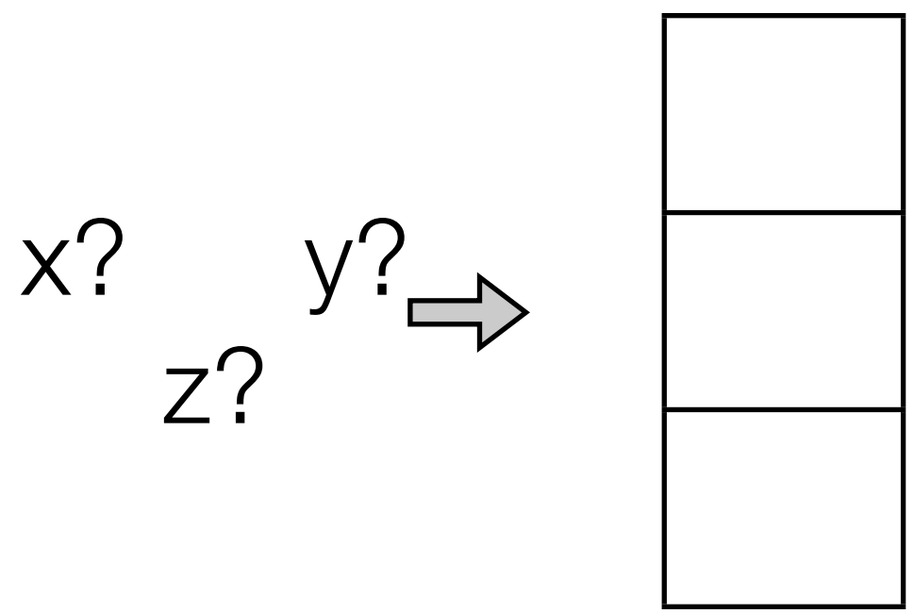
OOP: State


obj
(slot-value obj ‘x)
(slot-value obj ‘y)
(slot-value obj ‘z)
(aref obj 0)
(aref obj 1)
(aref obj 2)

How to map slots?

```
(defclass point ()
   (x y))

(defclass point-3d (point)
   (z))
```



How to map slots?

1. compute class precedence list
2. compute slots
3. determine slot locations

How to map slots?

1. (compute-class-precedence-list ...)
2. (compute-slots ...)
3. (slot-definition-location ...)

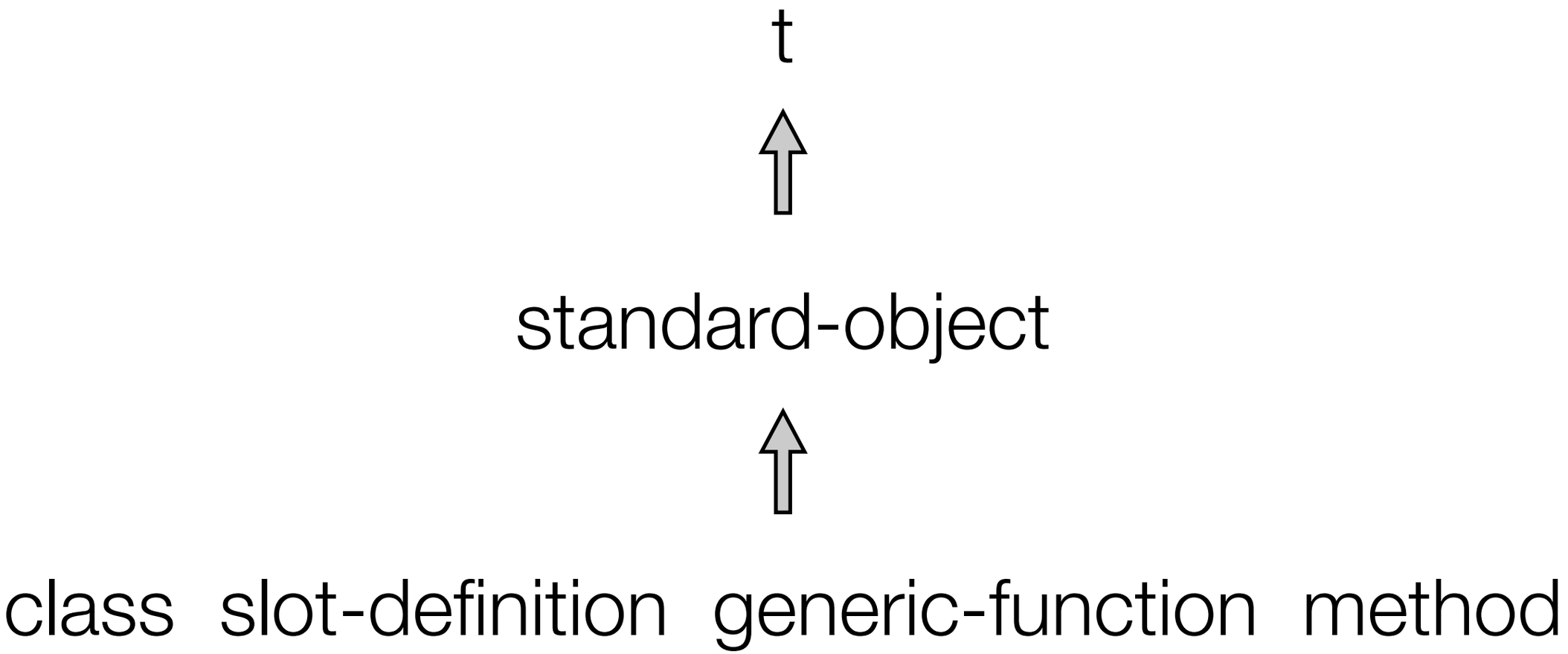
The Idea!

- Turn compute-class-precedence-list, compute-slots, and so on, into generic functions!

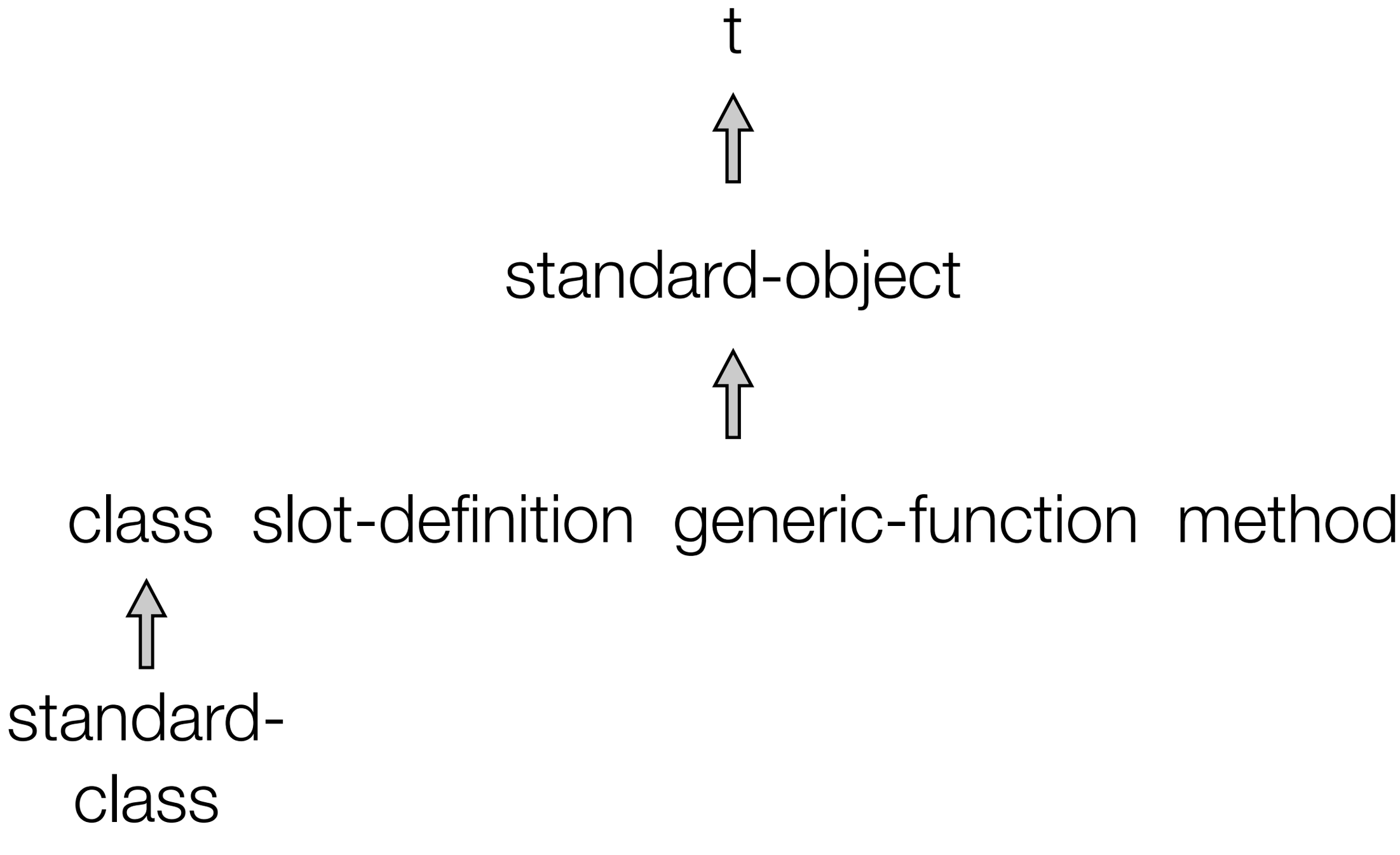
- Allow changes to the CLOS object model!

- Question: How to distinguish between standard and non-standard behavior?

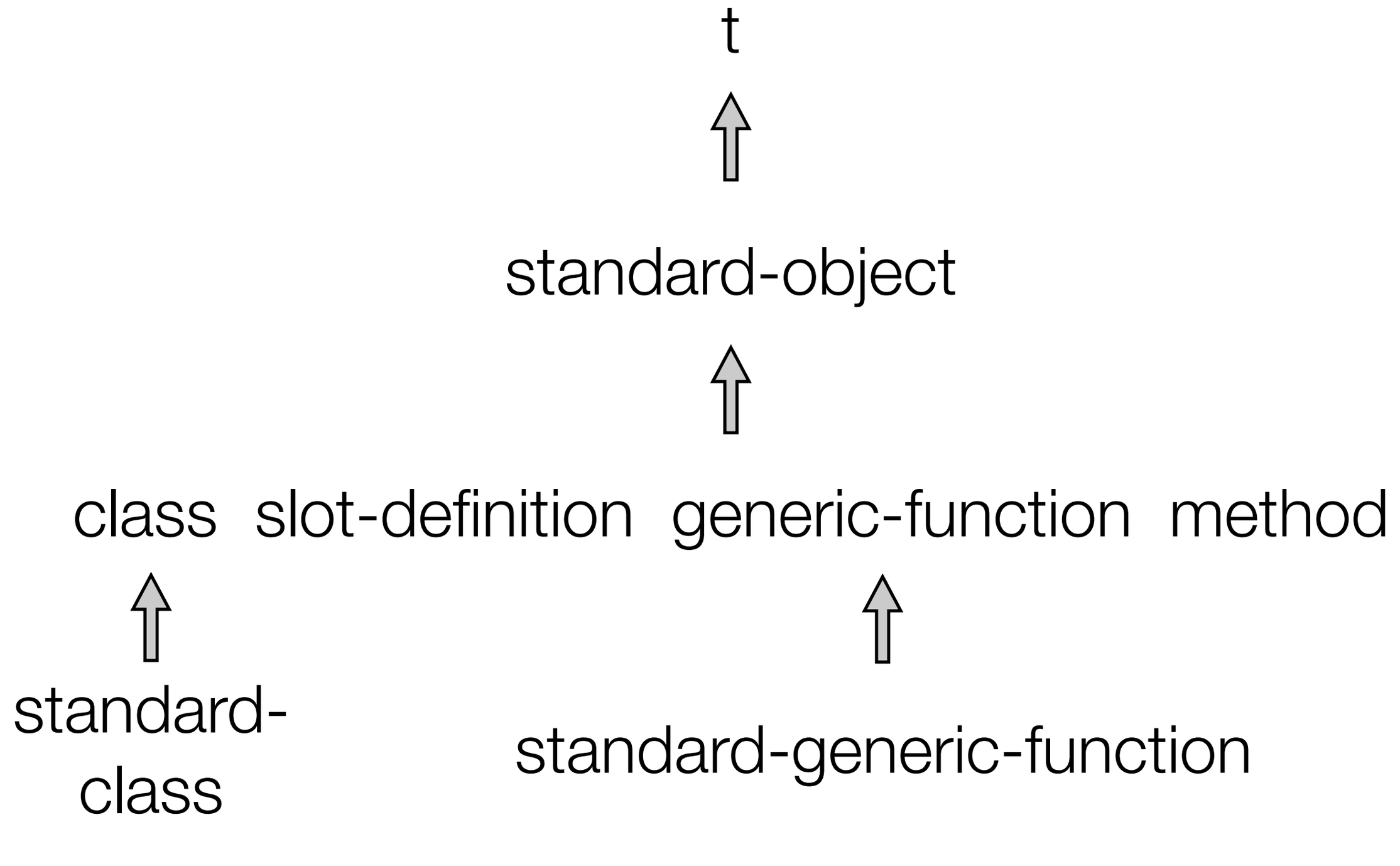
Metaobject classes



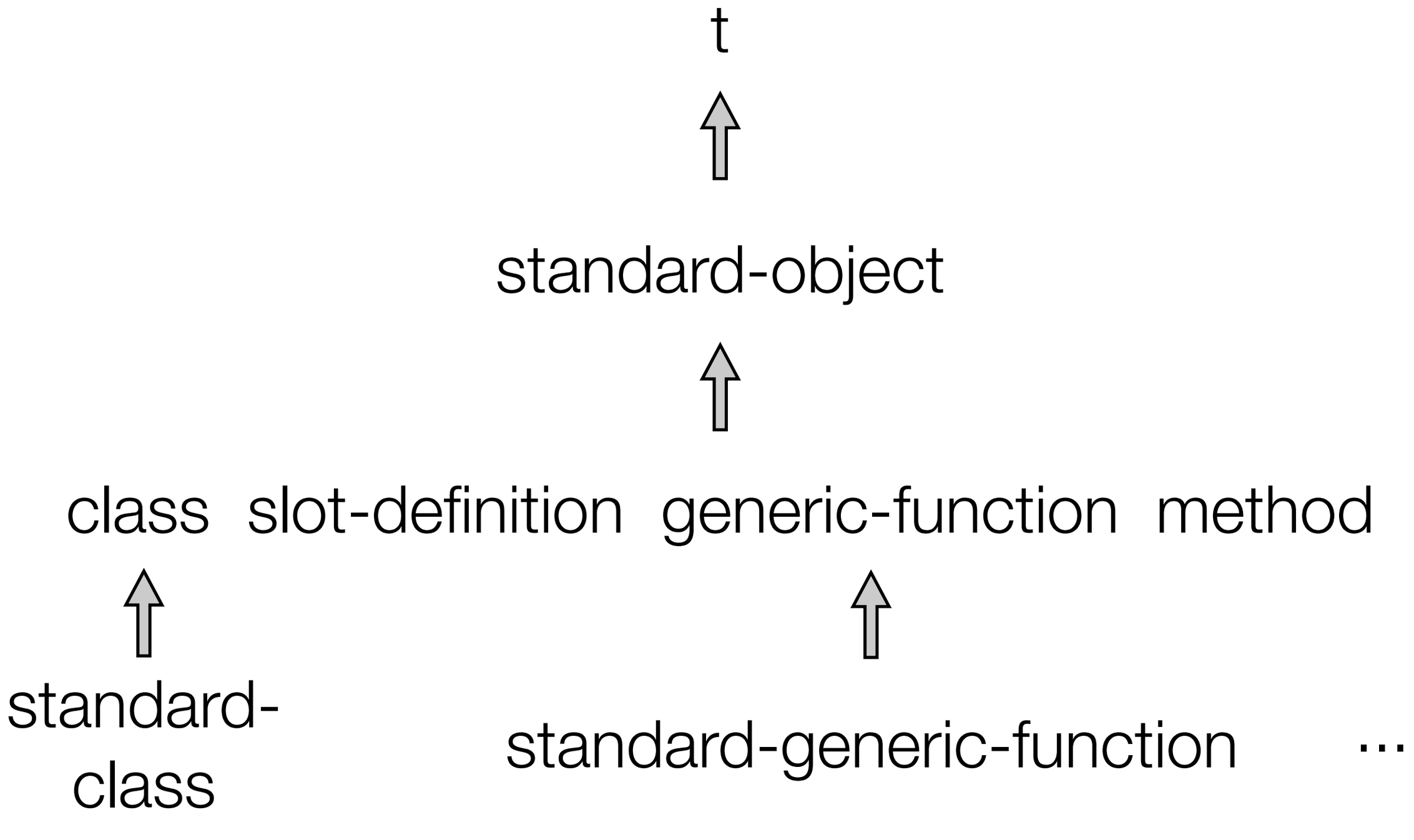
Metaobject classes



Metaobject classes



Metaobject classes



Class metaobjects

- (defclass persistent-class (standard-class)
  ((database-connection ...)))

- (defclass person ()
  ((name ...)
   (address ...))
  (:metaclass persistent-class))

Instancte Structure Protocol

- (defmethod person-name ((object person))
  (slot-value object ‘name))

- (defun slot-value (object slot)
  (slot-value-using-class
  (class-of object) object slot))

- (defmethod slot-value-using-class
  ((class standard-class) object slot)
  (aref ...))

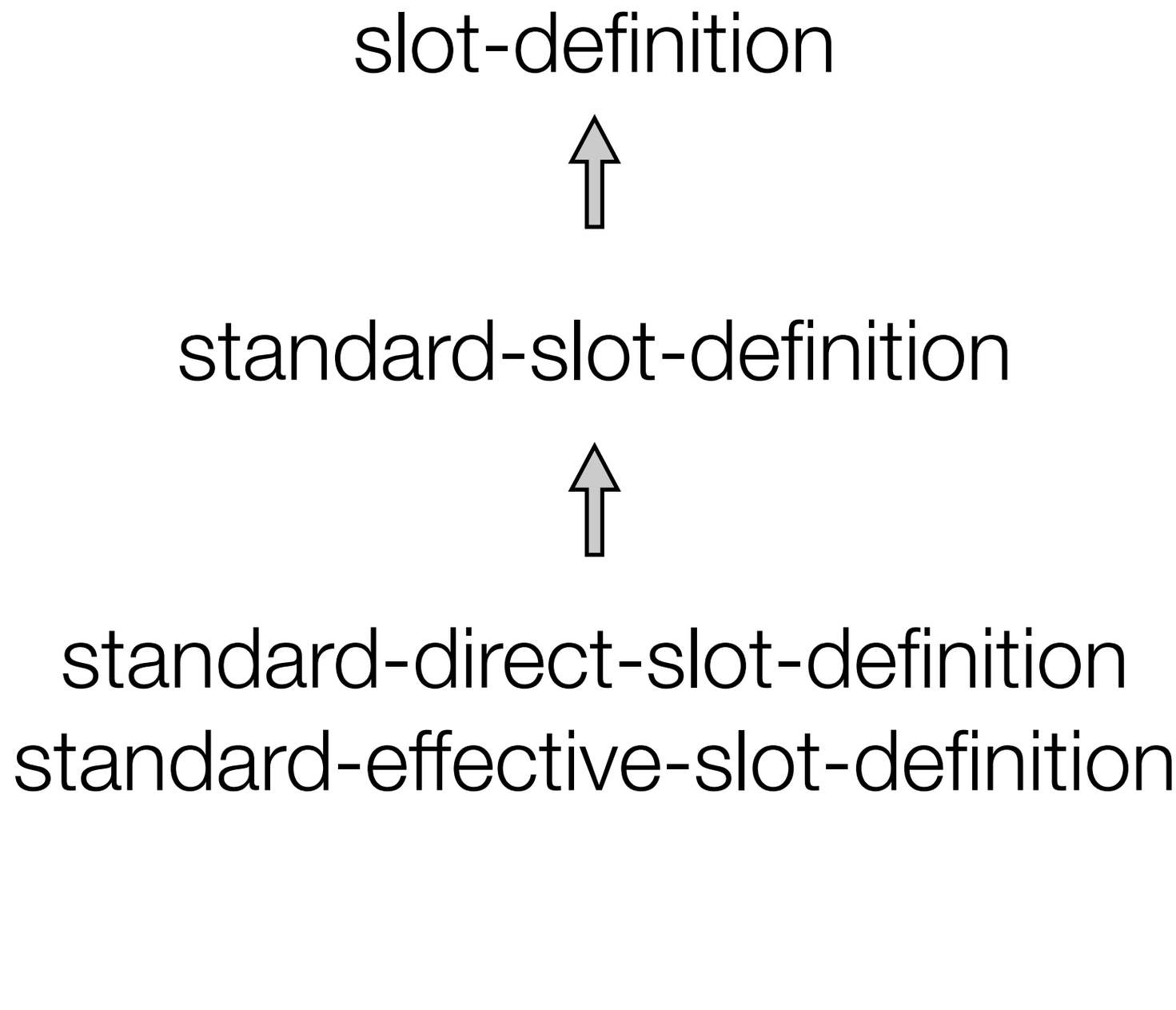
Instance Structure Protocol

- (defmethod slot-value-using-class
  ((class persistent-class) object slot)
  (fetch-slot-from-database ...))

Other Protocols

- Initialization protocols
- Class finalization protocol
- Instance structure protocol
- Funcallable instances
- Generic function invocation protocol
- Dependent maintenance protocol

Slot metaobjects



“Python” objects

1. Define a mix-in for hashtable-based slots.
2. Ensure that this mix-in is used.
3. Modify the slot access protocol.

Metaobject Initialization

- Metaobjects are initialized, like base objects, via initialize-instance.
- Class metaobjects and generic function metaobjects support reinitialize-instance.
- Metaobjects do not support change-class.

Class Initialization

- Class re/initialization calls...
    - direct-slot-definition-class
    - reader-method-class / writer-method-classs
    - validate-superclass
    - add-direct-subclass / remove-direct-subclass

Metaobject Initialization

- Only define :around methods on initialize-instance and/or reinitialize-instance.
- Don’t define methods on shared-initialize and/or update-instance-for-...-class.

Class Finalization

- finalize-inheritance is called for a class metaobject before the first instance is allocated with allocate-instance.

- finalize-inheritance calls...
  - compute-class-precedence-list
  - compute-slots
  - compute-default-initargs

compute-slots

- compute-slots calls...
  - effective-slot-definition-class
  - compute-effective-slot-definition

Instance Structure Protocol

- slot-value => slot-value-using-class
- (setf slot-value) => (setf slot-value-using-class)
- slot-boundp => slot-boundp-using-class
- slot-makunbound => slot-makunbound-using-class

Instance Structure Protocol

- Low-level access via
  standard-instance-access / funcallable-standard-instance-access
  (taking object + index as arguments)

Funcallable Instances

- funcallable-standard-class is like standard-class, except one can use set-funcallable-instance-function to assign a function definition with instances of such classes.

Generic Function Invocation Protocol

- Generic functions are funcallable instances.
- The “funcallable” function is determined by compute-discriminating-function.
- The discriminating is (re)computed by initialize-instance, reinitialize-instance, add-method and remove-method.

Generic Function Invocation Protocol

- The default discriminating function calls...
  - compute-applicable-methods-using-classes
  - maybe compute-applicable-methods
  - compute-effective-method

Dependents

- Whenever something is changed (initialized, reinitialized, methods added or removed) one can be notified about such changes.

User Macros

- defclass expands into a call to ensure-class which in turn calls ensure-class-using-class.

- defgeneric expands into a call to ensure-generic-function which in turn calls ensure-generic-function-using-class.

User Macros

- defmethod expands into an instantiation of a method metaobject and a call to add-method.

- The method body is processed by make-method-lambda.

- (make-method-lambda inserts definitions for call-next-method and next-method-p, among other things.)

Warning

- The CLOS Metaobject Protocol is not an official (ANSI) standard.
- Except for SBCL, no CL implementation supports the CLOS MOP completely!
- Some do not support it at all.

Literature & Software

- Andreas Paepcke, “User-level Language Crafting”
- G. Kiczales, J. des Rivieres, D. Bobrow, “The Art of the Metaobject Protocol”
- http://www.lisp.org/mop/
- http://common-lisp.net/project/closer/

MOPs for other languages

- Scheme: http://community.schemewiki.org/?object-systems
- Smalltalk: http://www.laputan.org/#Reflection
- C++: Ira Forman, Scott Danforth, “Putting Metaclasses to Work”
- Java: Ira Forman, Nate Forman, “Java Reflection in Action”