From Metaprogramming to Aspect-Oriented Programmin

Éric Tanter
University of Chile

Outline

Metaprogramming and Reflection

Open Implementations

Aspect-Oriented Programming

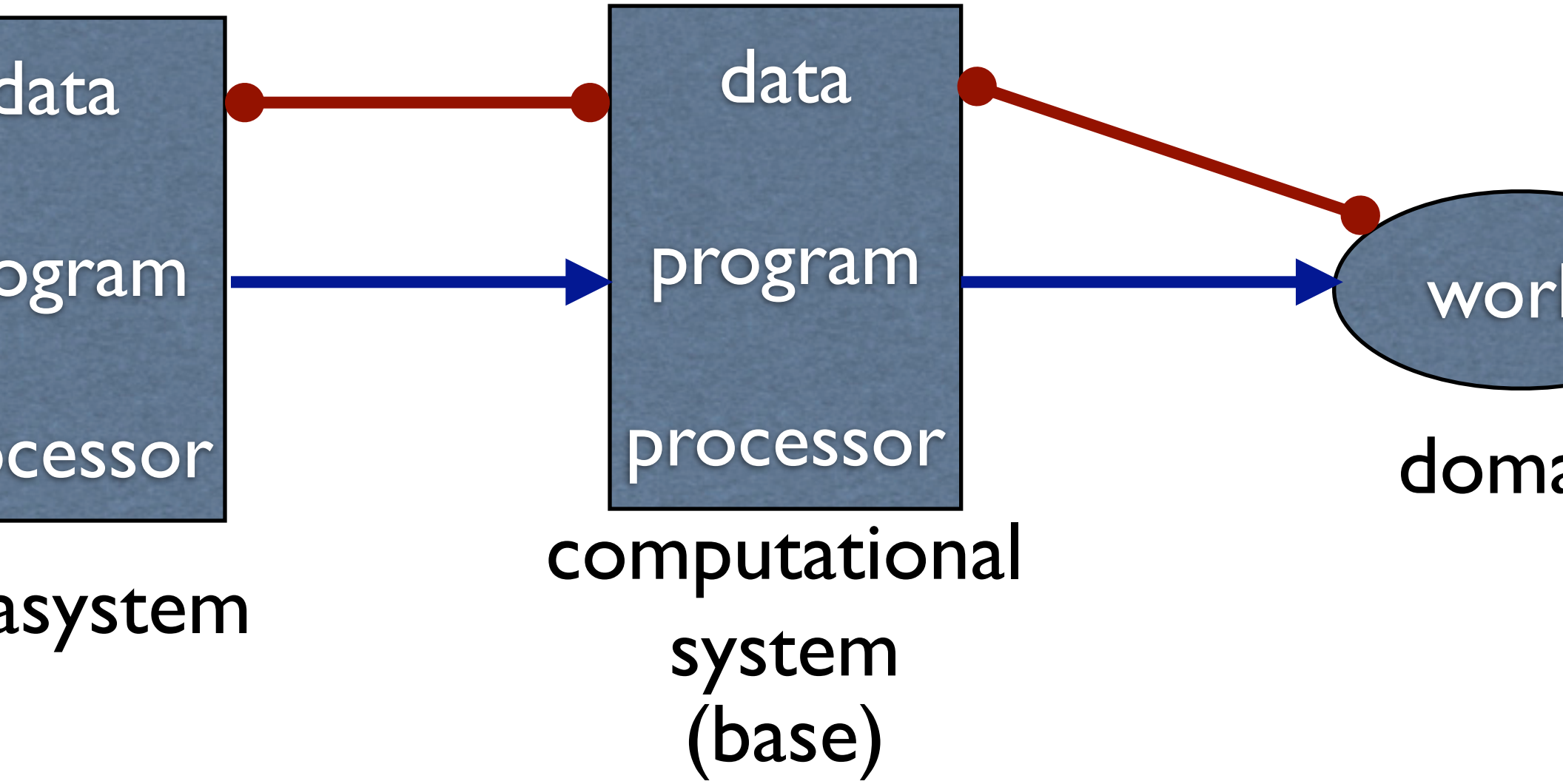
Metaprogramming & Reflection

Programs and Data

- Fundamental distinction appearing in 1830s
  - Charles Babbage's Difference Engine No.2
  - *store* with data, *mill* with programs
- von Neumann architecture (1958)
  - data and programs are stored in the *same* memory
  - a program could manipulate another program as d (and even itself!)
- Theoretical computer science
  - Turing machines: the universal TM (1936)

Concepts

[Mae



Reflection

Reflective system

- CS accessing its own metasystem

Seminal work of Brian C. Smith [Smith82]

- 3-LISP

process' integral ability to represent, operate on, a
herwise deal with itself in the the same way that it represen
erates on and deals with its primary subject matter."

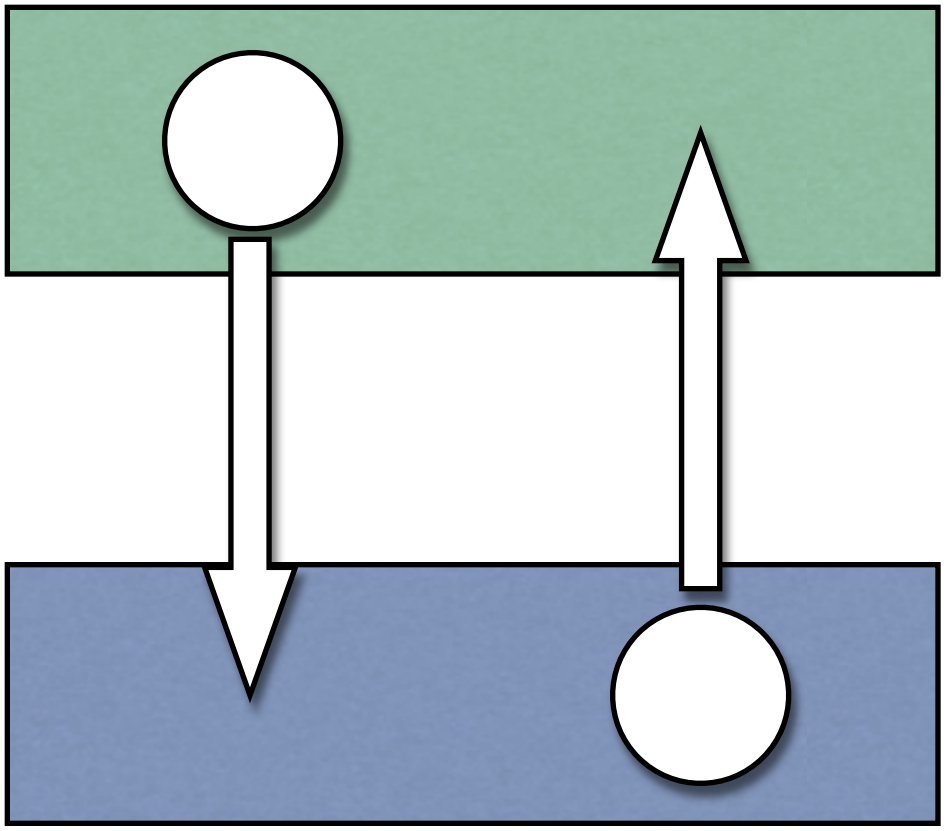
Pattie Maes [Maes87]

- study of OOP in reflective architectures

Reflection and Adaptation

Reflection operators [FriedmanWand84]

etalevel
valuator)



reificatio
absorptio

se level
rogram)

trospection: program *observes* its evaluator st

Structure and Behavior

tructural reification

implicit structures accessible *as first-class entities*

eg. classes, methods, fields in Java Reflection API

ehavioral reification

implicit events of execution accessible *as first-class e*

eg. MethodCall object with receiver, method, args

eg. creation, field access, finalization, etc., as objects

Reflection and OOP

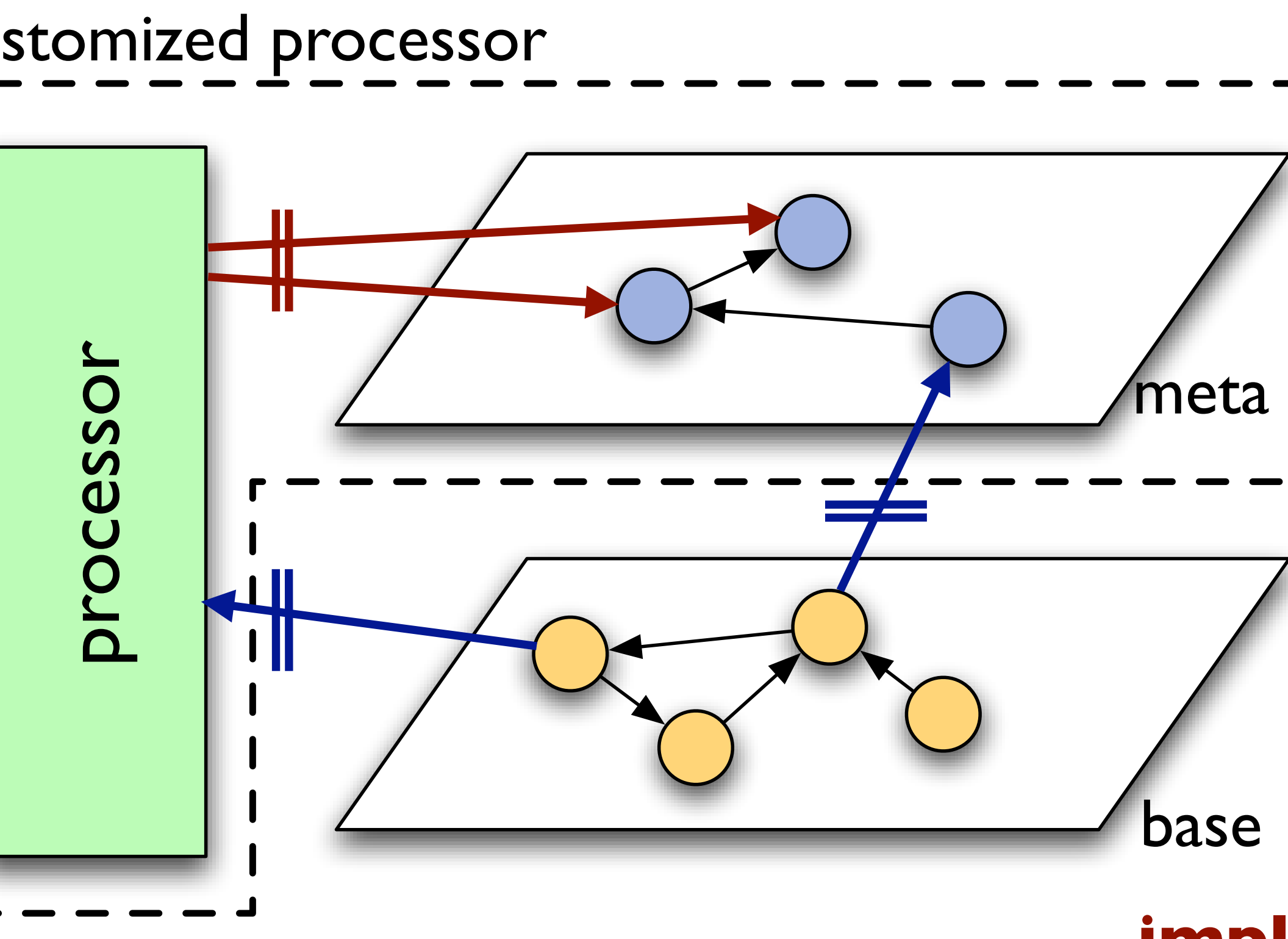
A Good Match

reflection provides *power* for adaptation

OOP provides *structure* and *locality*

- encapsulation
- message passing
- object-oriented interfaces
- proper decomposition
- incremental specialization of default implementatio

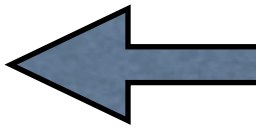
MetaObject Protocols



Example

Simple MOP for controlling method calls

```
s TraceMetaobject extends Metaobject {
blic Object handleCall(MethodCall call){
rint("before calling "+ call.getMethodName());
Object result = call.perform();
rint("returning with result "+ result.toString());
```



```
tor v = MOP.create(Vector.class, new TraceMetaobject(
```

implicit protocol

explicit protocol

Example: Concurrency

```
ffer {
ct[] elements = new Object[MAX];
p = 0;
 Object get(){ return elements[top--]; }
 void put(Object o){ elements[top++] = o; }
```

thread sa

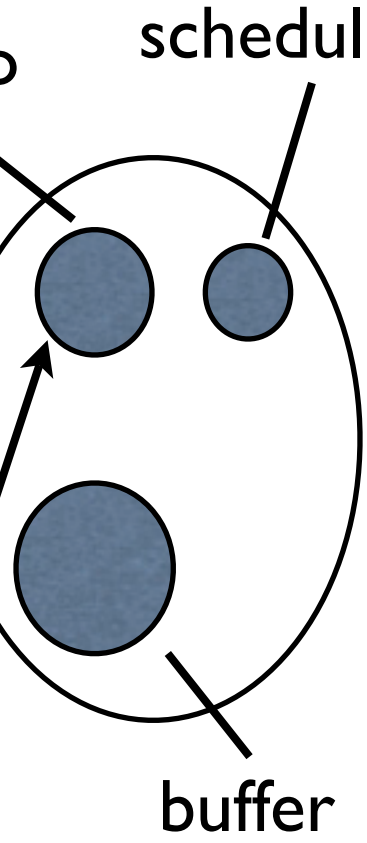
```
ynchronized Object get()
rows InterruptedException {
op == 0)
);
 o = elements[top--];
All();
 o;
```

```
public synchronized void put(O
            throws InterruptedExc
  while(top == MAX)
    wait();
  elements[top++] = o;
  notifyAll();
}
```

[C

Sequential Object Monitors

- based on a MOP for controlling method calls




```
schedule: Buffer with: BufferScheduler
```

```
class BufferScheduler extends Schedule
   Buffer buf = ... ;
   void schedule(){
      if(buf.isEmpty()) scheduleOldest("pu
      else if(buf.isFull()) scheduleOldest("
      else scheduleOldest();
}}
```

explicit protocol implicit prot

Open Implementations

Open Implementations

Reflection in the context of programming langua

- reify structure or execution semantics of programs

Notion can be generalized

- "implementational reflection"
- led to "open implementations"

plementational Reflect

[Rac

Reflection can be used to build malleable syste
of all kind

- not only interpreters and compilers!
- systems also depend on other systems they *use*

computational reflection	reflective architecture
implementational reflection	**open implementation**

Definitions

[Rac

ementational reflection

Reflection that involves inspecting and/or manipulating the implementation structures of *other systems* used by a program

n implementation

A system with an *open implementation* provides (at least) two linked interfaces to its clients:

1. a *base-level* interface to the functionality

2. a *metalevel* interface that reveals *some aspects*

Duality...

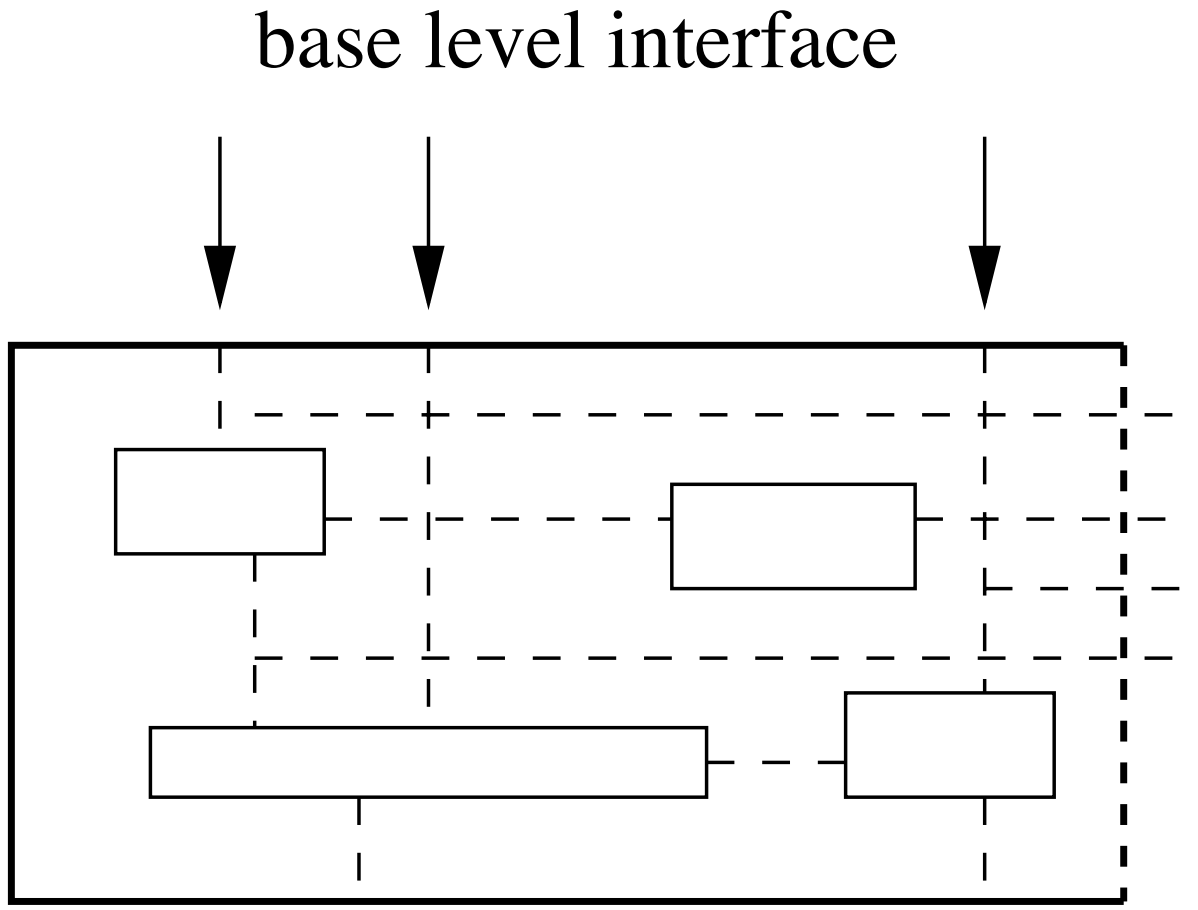
Computational <> Implementational

a language interpreter is the implementation of language

the interface of any system can be seen as an interpreter for that language

different characterizations of the same essential ability

Structure of an OI



talevel interface

oints at which base-level behavior can be customized
fferent semantics and/or performance

ausal connection is trivial

lOPs = OI of interpreters [Kiczales+91]

...ing Up Implementati

...ontrary to the black-box abstraction principle!!

...ny realistic system implies a number of **tradeo**

- the higher the level is, the more tradeoffs [K
- no single fixed implementation will satisfy all users

...nd a large number of programs perform poorly because
...uage's tendency to hide "what is going on" with the misgu
...intention of "not bothering the programmer with details"

[W

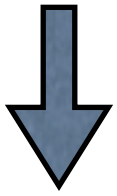
A Classical Example

A performance issue in class-based languages

- how are instance variables (slots) implemented?

```
(defclass position ()
  (x y))
```

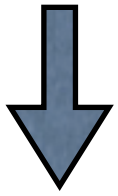
many instances,
with slots always used



array like representation

```
(defclass person ()
  (name age address email))
```

many instances,
only few slots used in one
given instance



hashtable like representation

The CLOS MOP

Open implementation of CLOS

[Kiczale

```
(allocate-instance class)
(get-value class instance slot-name)
(set-value class instance slot-name new-value)
```

```
efclass hashtable-class (std-class) ())
efmethod allocate-instance ((c hashtable-class))
..allocate a hashtable to store the slots...
efmethod get-value...) (defmethod set-value...)
```

```
(defclass person()
  (name age address email...)
```

Philosophy

Black-box abstraction: tricks/hacks

White-box abstraction (eg. open source)

- no guarantee that code is well-enough structured

Open implementations:

- reify *some* aspects of the implementation

"open up the implementation, but in a principled way"

[Kic

blicitly focusing on the metalevel as a separate and first-
rface [...] forces a greater attention to exposing importa

OO Advantages

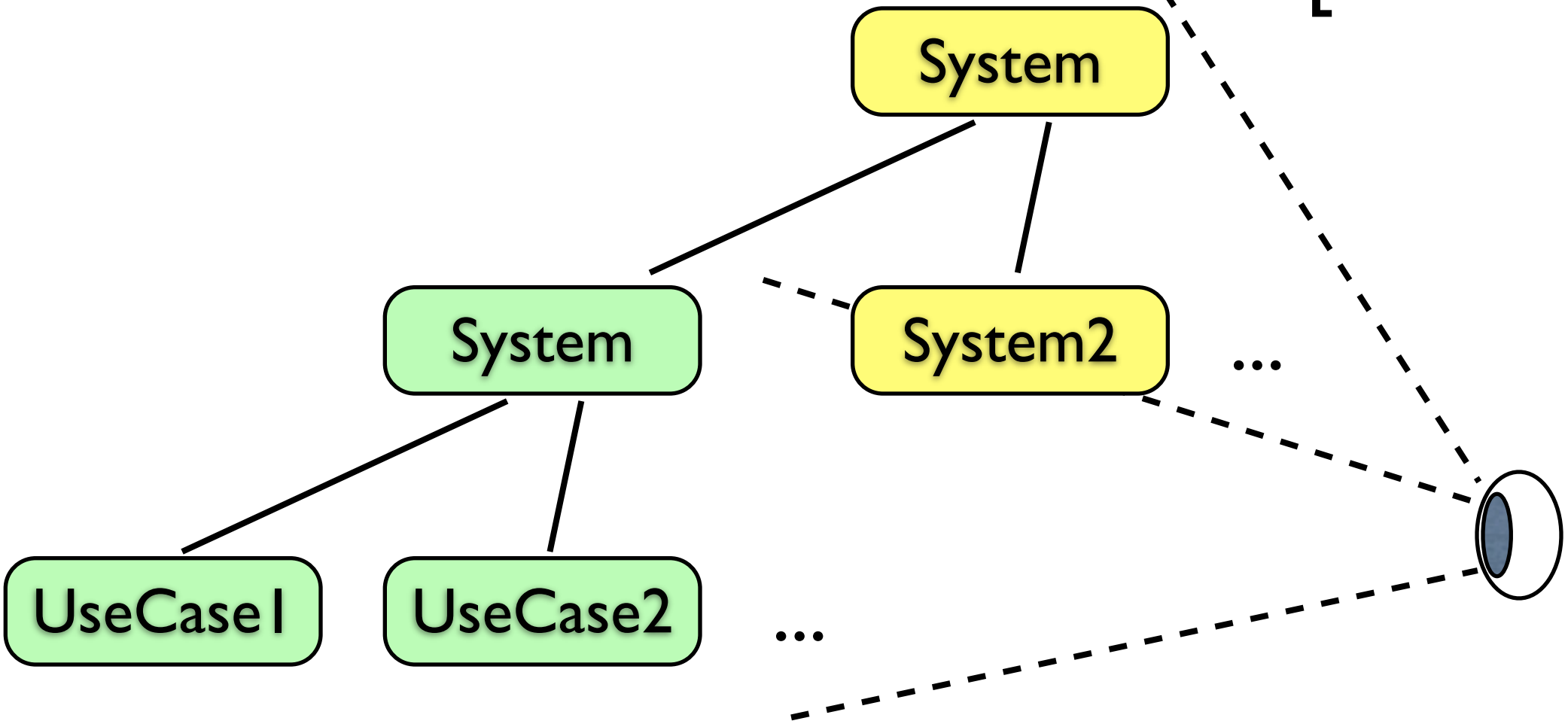
- pport *variability* in a system's implementation
  - performance tuning
  - semantic customization
- o need for direct support for rarely-needed fea
  - eg. CLOS standard: backward compatibility vs. new c
  - support a "CLOS region" rather than a "CLOS poin

Open implementations

- languages: CLOS [Kiczales+91]
- compilers [Lamping+92, Chiba95]
- operating systems: ApertOS [Yokote92]
- window systems: Silica [Rao91]
- reflective systems: Reflex [Tanter+01]

Providing OI

[Kiczales



OI design: think of a *range* of systems

- which range?
- iterative process, feedback, refinement

Interface styles

Declarative style

[Kiczales

- *declare* expected usage of the module

keSet("n=5, insert=high")

+ user friendly, abstract
- automatic choice can

Strategy style

- choose strategy in fixed list

keSet("LinkedList")

+ precise selection
- might choose badly
- limited set of strategie

Layered style

- strategy + possible to *provide* a new strategy

keSet("mySet")

[strategy]

Reflection and OIs

[Ra

OI Interface styles as *design patterns*

- eg. Strategy for strategy and layered OI styles
- but *explicit* representation of *any* aspect of a syste
  can compromise efficiency badly
- *lazy reification* for making implementation state ex

Reflection/MOPs

- implicit and selective reification of some aspects
- possibly adaptable at runtime

OIs come from generalization of reflection,

Mastering Locality

Fundamental though hard to grasp

many MOP architectures were locality experim

5 coarse notions discussed in [Kiczales93+]

feature	access individual features
textual	indicate what behavior to change
object	possibly per-object basis
strategy	affect individual strategy
implementation	simple change simple, incrementality

Towards AOP

Kiczales effort to understand locality led to AO

very often, the concepts that are most natural to use at
neta-level ***cross-cut*** *those provided at the base level."*

We are, in essence, trying to find a way to provide two eff
ws of a system through ***cross-cutting localities****."*

he structure of complex systems is such that it is natural
ople to make this jump from one locality to another, and
have to find a way to support that*."*

[Kiczales