Towards a New Model of Abstraction for the Engineering of Software

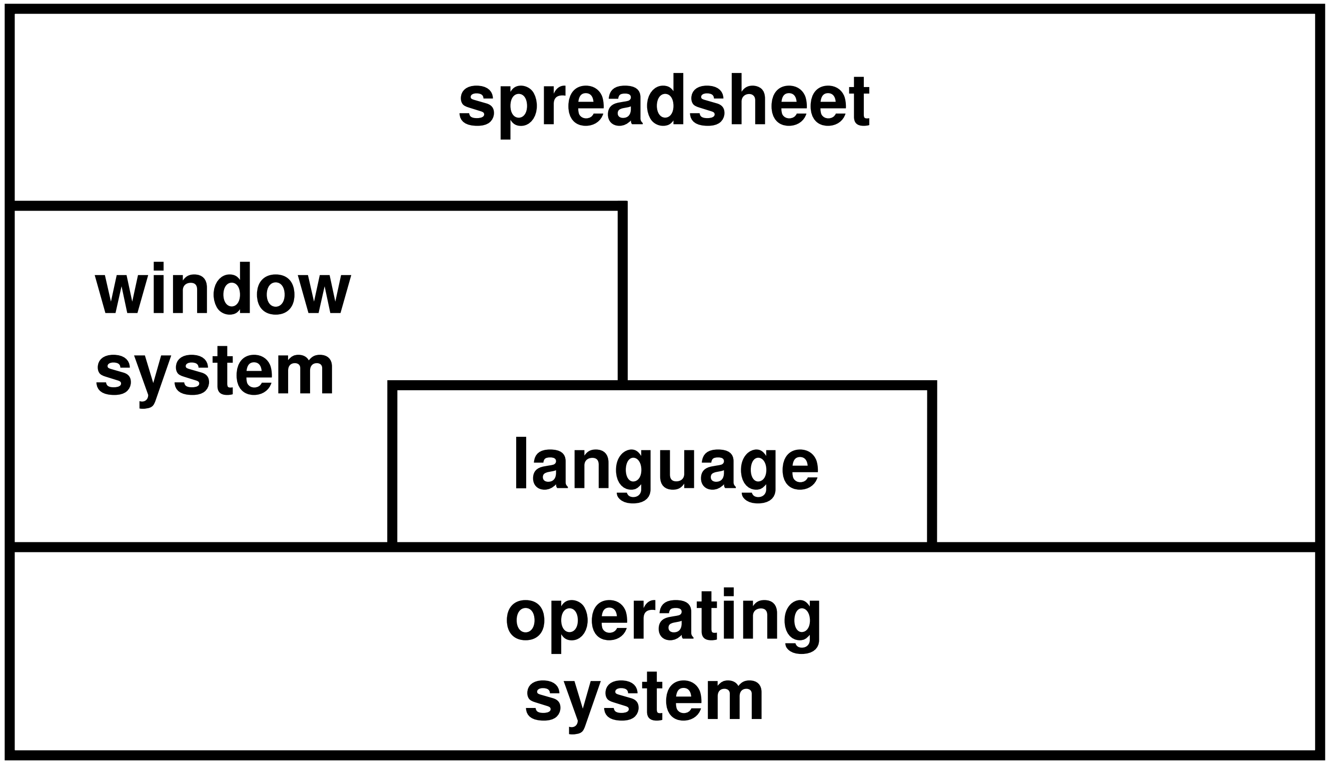
Gregor Kiczales
Embedded Computation Area
Xerox PARC
(gregor@parc.xerox.com)

A Crucial Idea

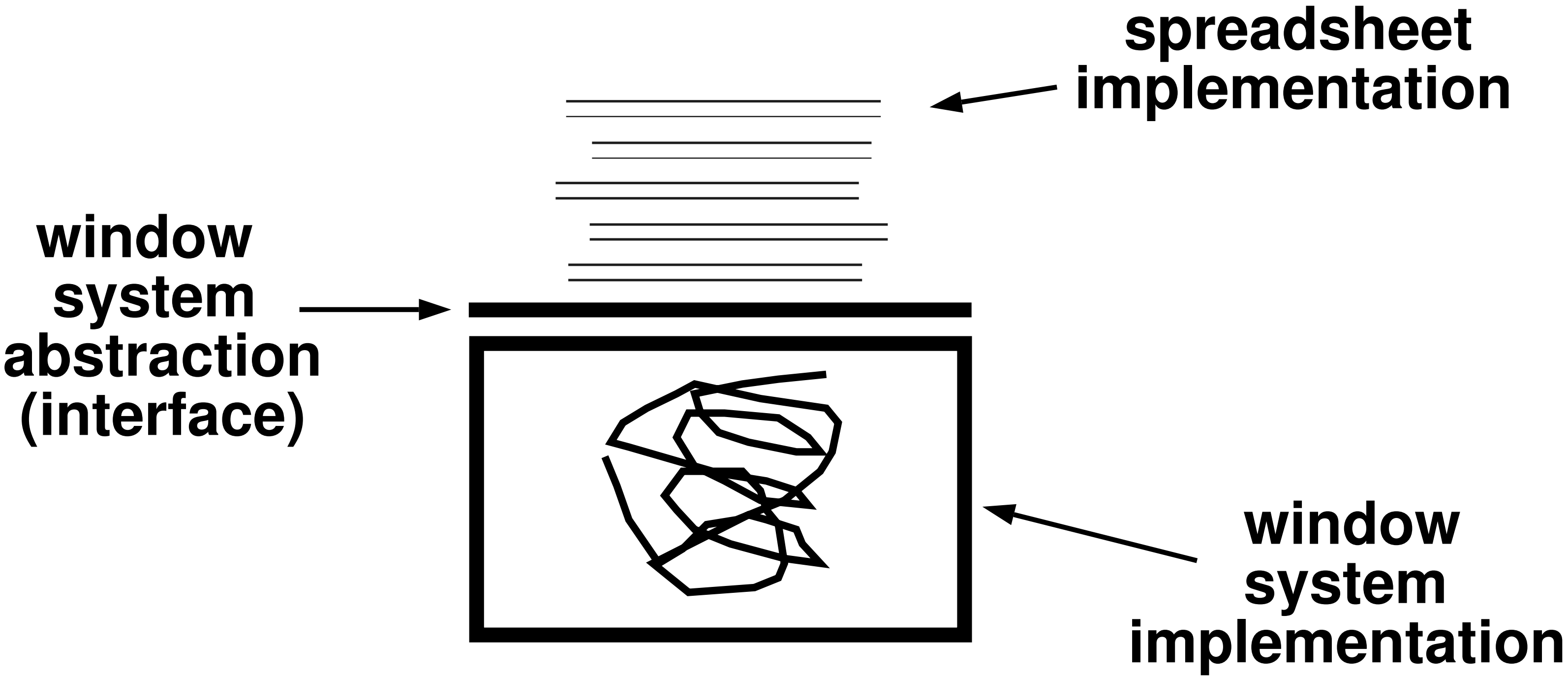
Engineers must master complexity.

Abstraction is a key tool for doing so.

Spreadsheet Display



Hiding Complexity



the abstraction provides useful functionality and "hides implementation details"

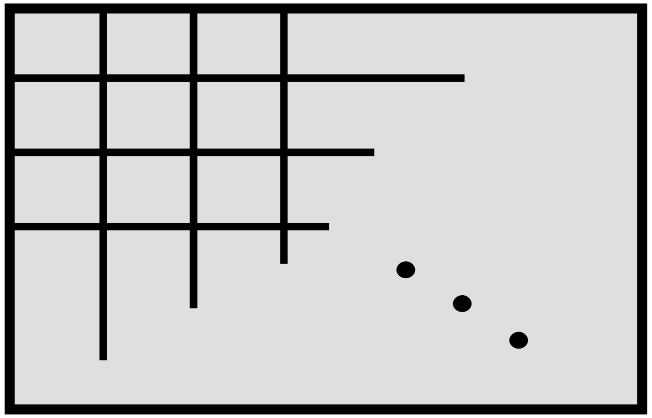
Communications of the ACM
December 1972, Vol. 15, No. 12

On The Criteria To Be Used in Decomposing Systems into Modules

D. L. Parnas
Carnegie-Mellon University

Every module in the second decomposition is characterized by its knowledge of a design decision which it hides from all others. Its interface or definition was chosen to reveal as little as possible about its inner workings.

It’s Simple

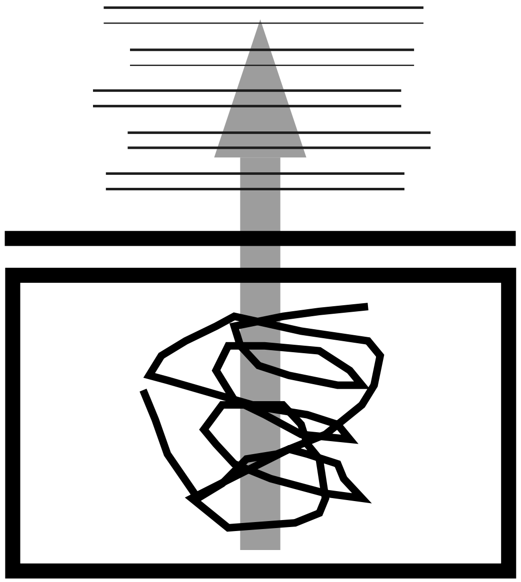


cells in which to click,
type, display etc.

```
i: 1→ 100
j: 1→ 100
make-window (~~~)
```

this looks fine...

But. . .
Performance?



the abstraction itself doesn't betray the implementation . . .

But, performance shows through

Mapping Dilemmas

strategy questions that affect performance differentially depending on client patterns of use

i.e. lightweight vs. heavyweight windows

abstraction barrier <u>locks</u> <u>in</u> (but doesn’t hide) mapping decisions

mismatch between client needs and mapping decision is a mapping conflict

Virtual Memory

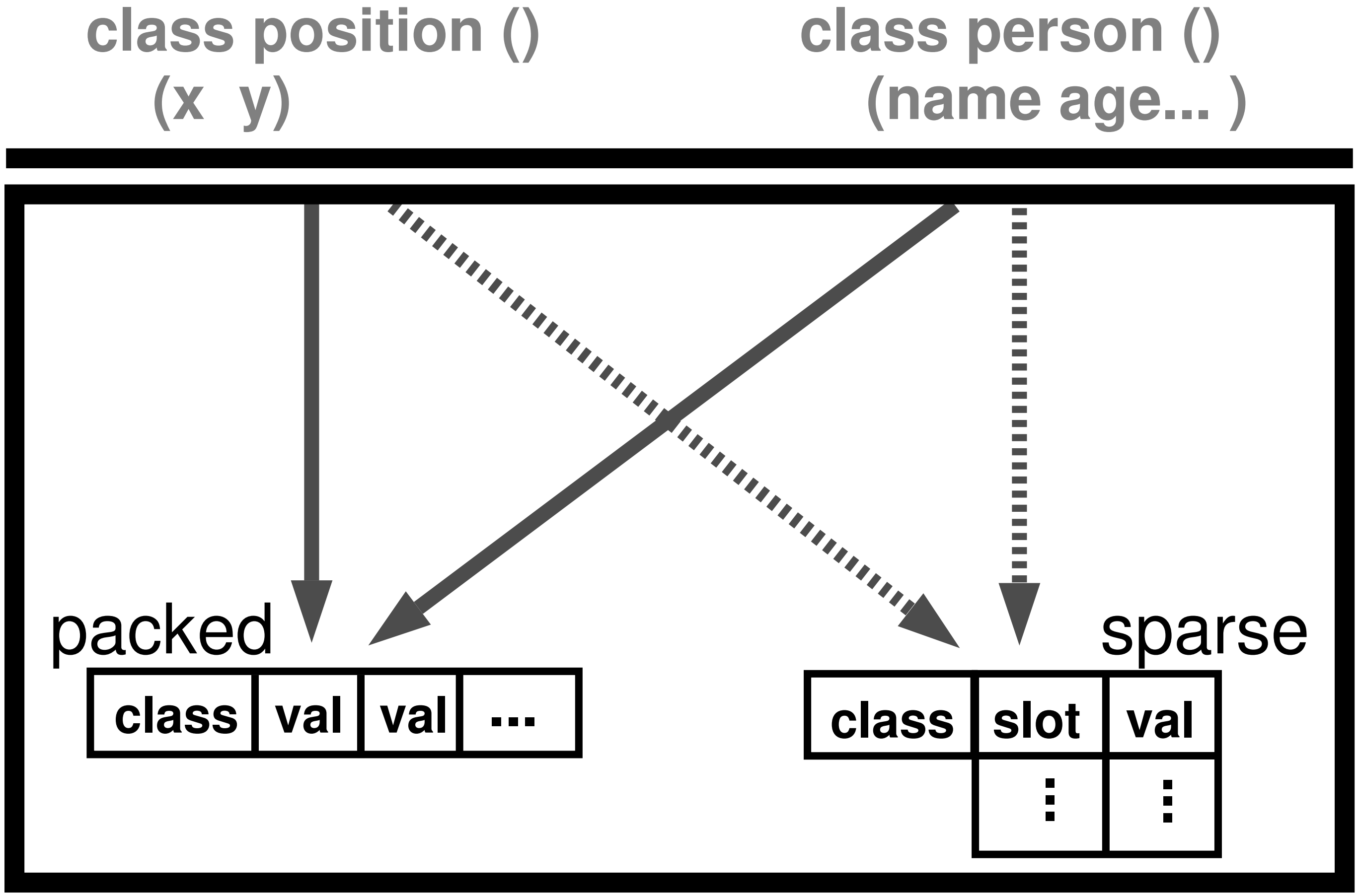
abstraction: lots of directly accessible memory (malloc, free, read, write)

dilemmas: page replacement policy, page replacement mechanism, grouping of addresses on pages

a program that happens to spread its 'working set' around is in trouble (databases, garbage collectors, graphics)

Object Systems

abstraction: objects with slots
dilemma: layout strategy



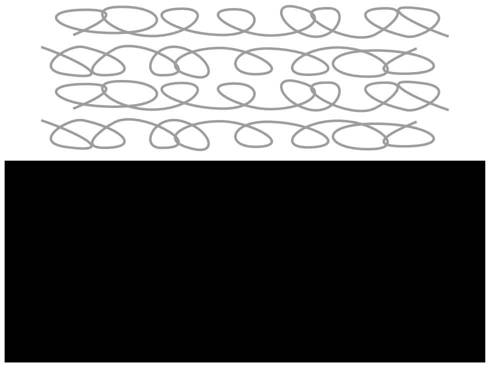
Resultant Complexity

hematomas of duplication

- spreadsheet
- person class
- database

coding between the lines

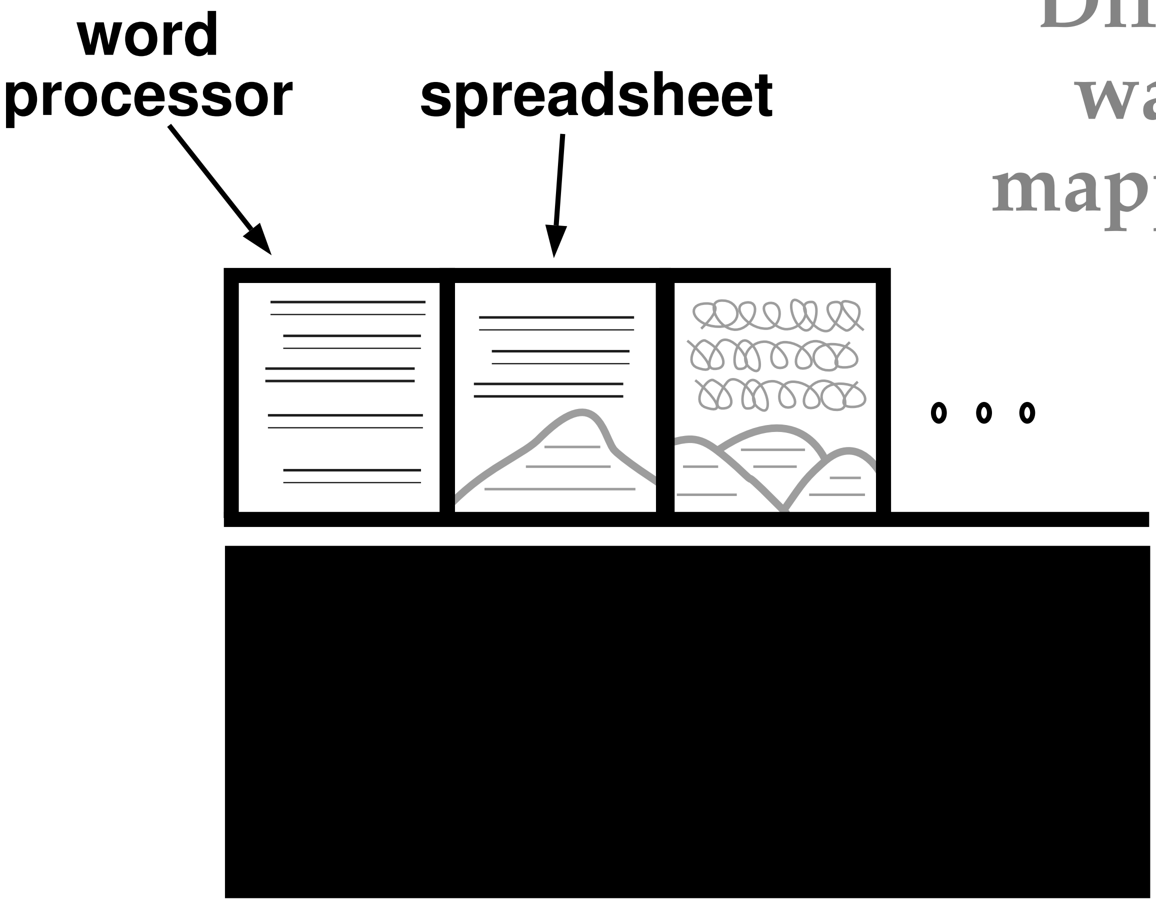
- garbage collector
- compiler
- graphics



Stressed by Multiple Clients

aka 'Reuse'

Different clients want different mapping decisions.



SigPLAN Notices 15, 3 (March 1980)

Toward Relaxing Assumptions in Languages and Their Implementations

Mary Shaw and Wm. A. Wulf
Carnegie-Mellon University

Traditionally, the designers and implementators of programming languages have made a number of decisions about the nature and representation of various language features that the authors feel are unnecessarily pre-emptive. For example, such decisions are commonly made about arrays: most languages support only rectangular arrays,

---

Pre-emptive decisions need not be limited to data and data-related aspects of a language. Subprograms are a good example where the language designer selects a particular mix of facilities and the implementor selects a single strategy for implementing that mix. Both the designer and implementor have only notions of typical use available; they are making the decisions too soon.

- *Discouraging the use of high-level languages:* Although most people now agree that the use of high-level languages is desirable, the fact remains that many major systems are still written in assembly language.

What to do?

avoid mapping dilemmas?

pretend it isn’t a problem?

document mapping decisions?

"here’s the sources, have fun"?

give up?

Information Processing 74 - North-Holland Publishing Company (1974)

On the Design of Programming Languages

Niklaus Wirth
Institut fur Informatik, Eidg. Technische Hochschule

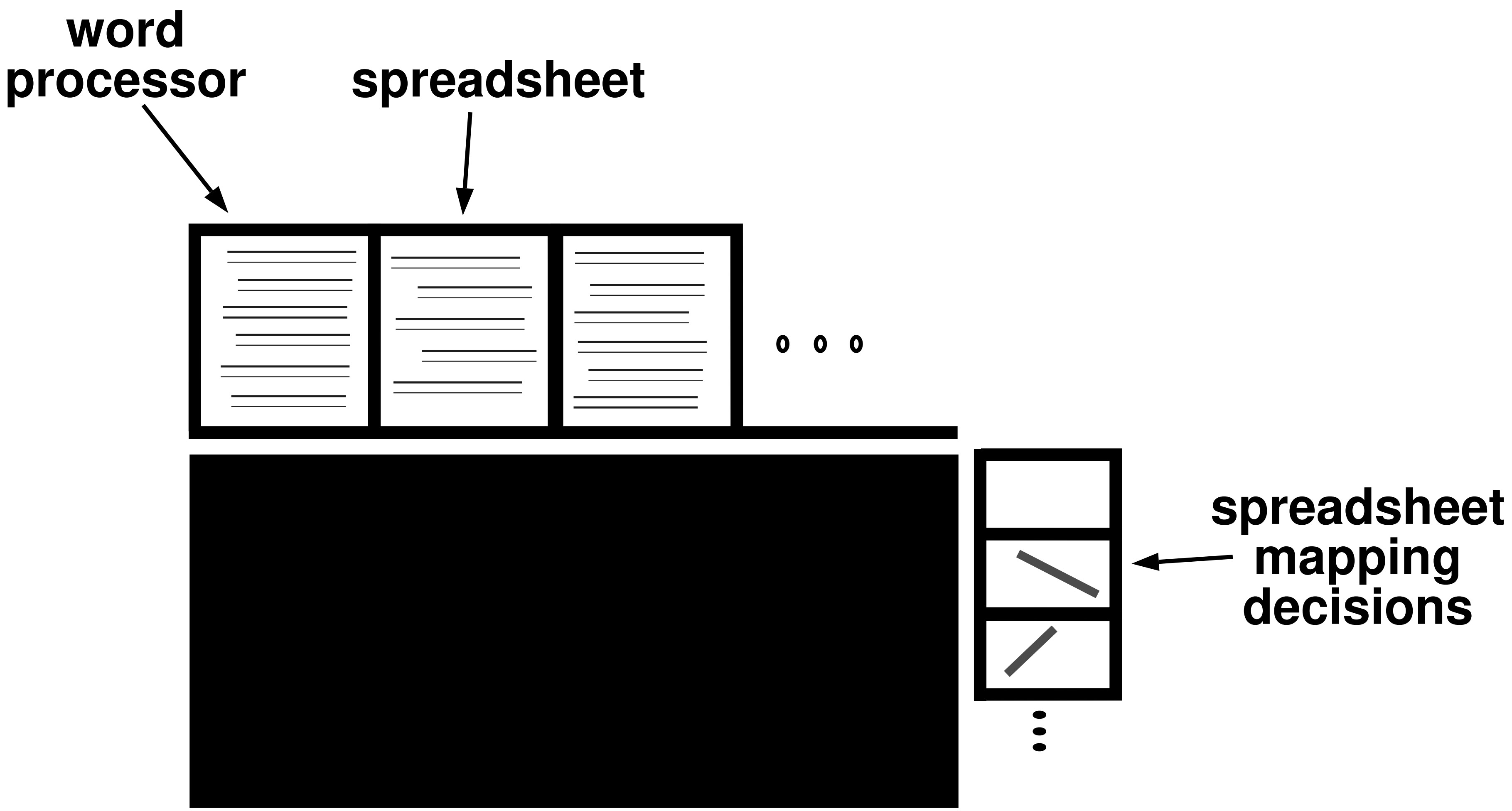
In fact, I found that a large number of programs perform poorly because of the language's tendency to hide "what is going on" with th misguided intention of "not bothering the programmer with details".

Open Implementation

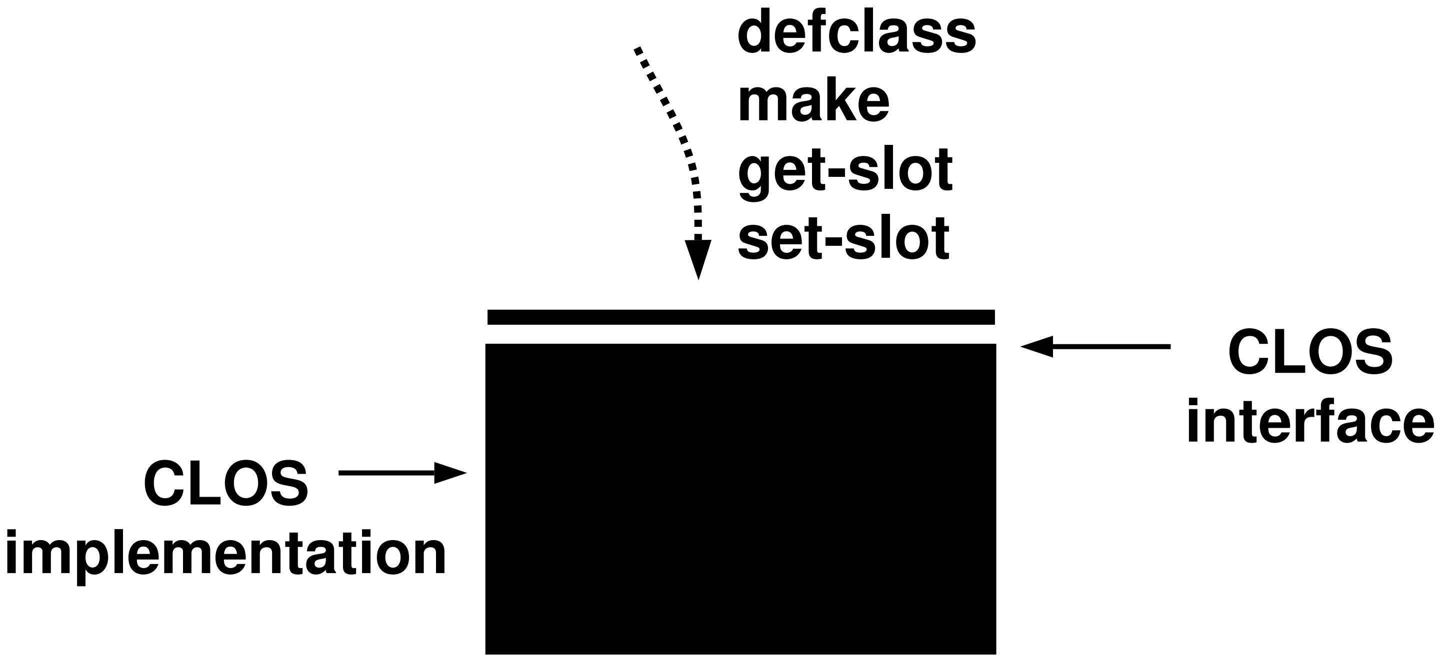
a quality default implementation & clients can adjust mapping decisions

- mapping decisions, not details
- scope control
- incrementality
- separation of concerns
  "if you can’t conquer, at least divide"
- retains benefits of black-box approach

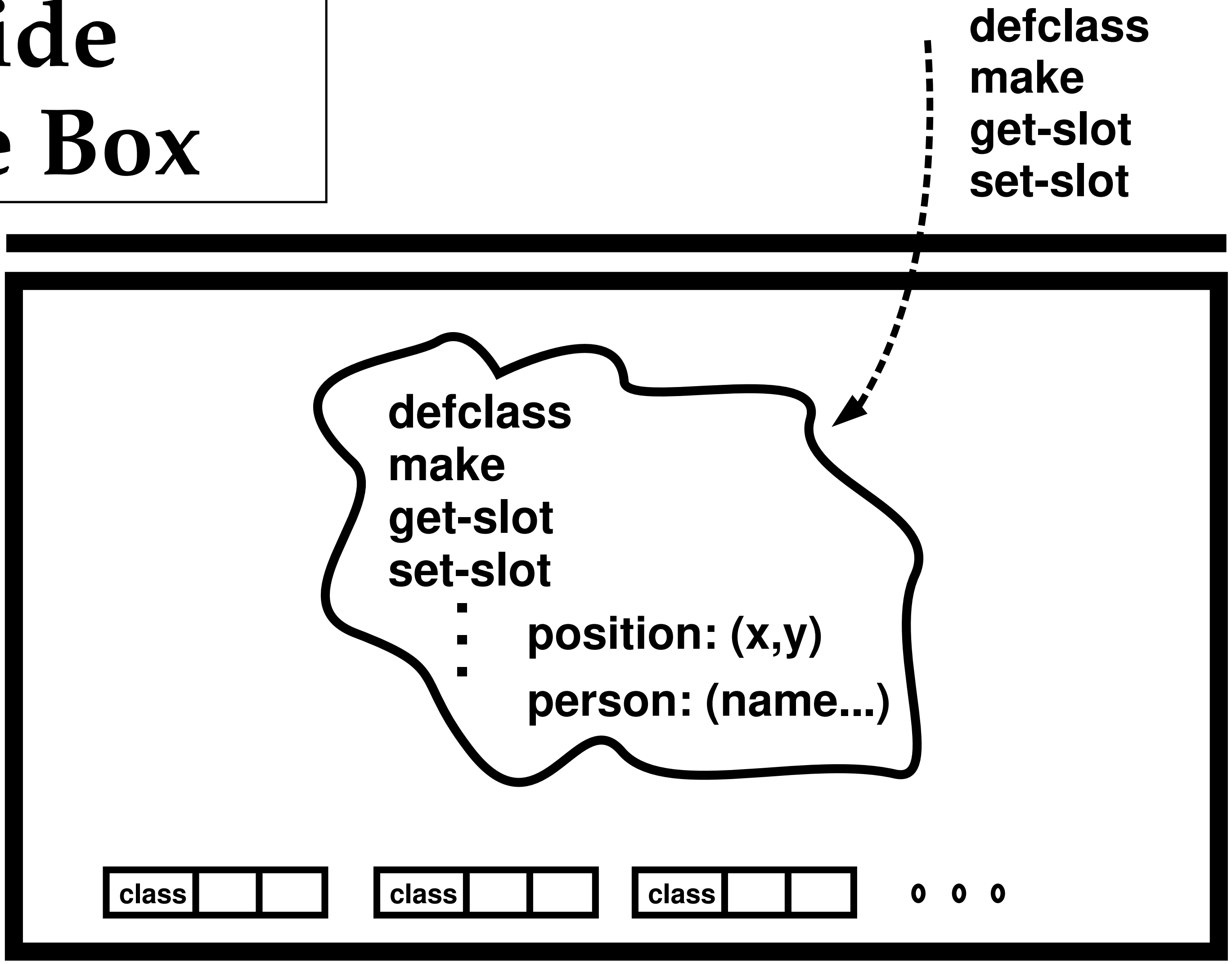
Client-Specific Mapping Decisions



Traditional Black Box

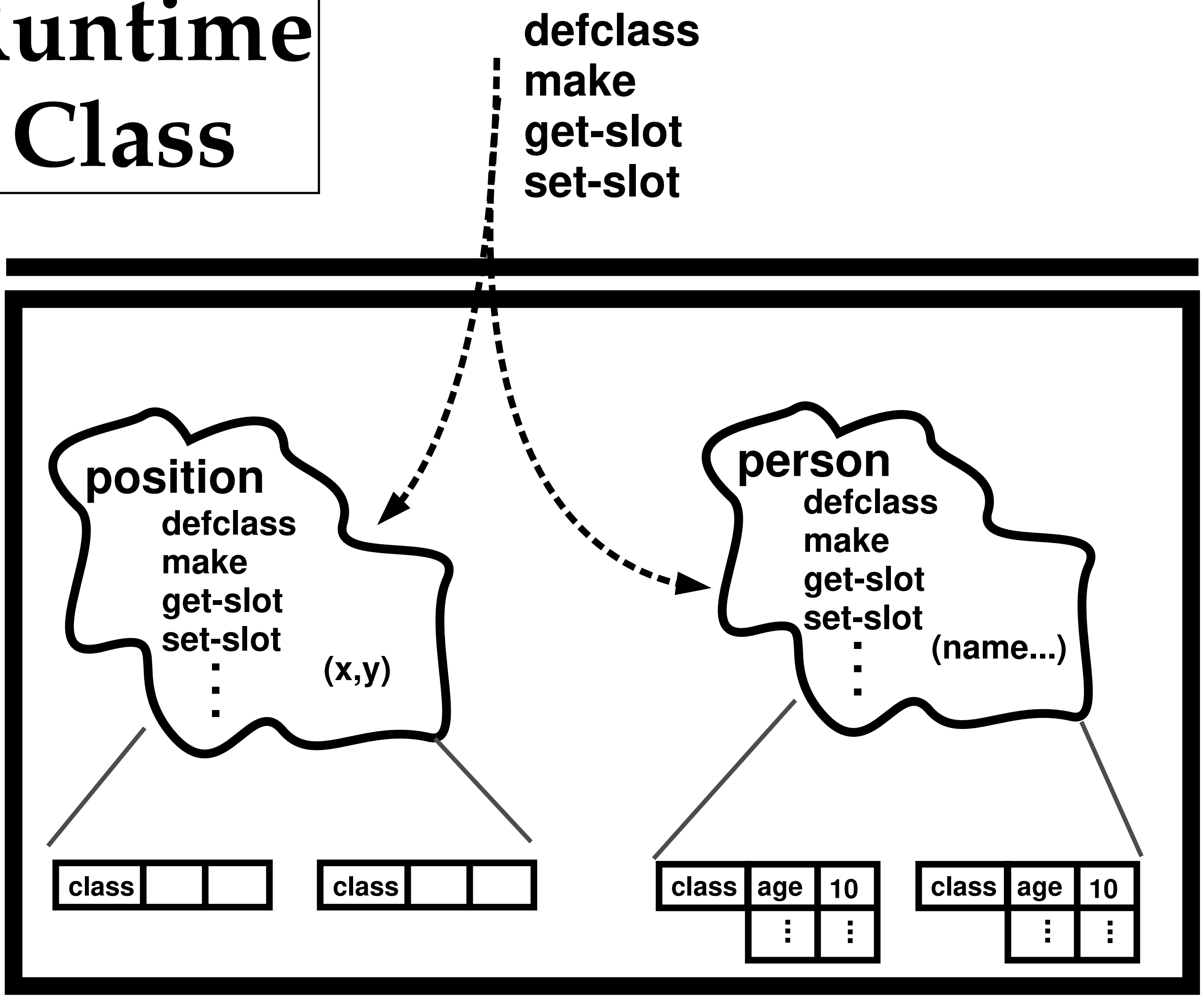


Inside The Box

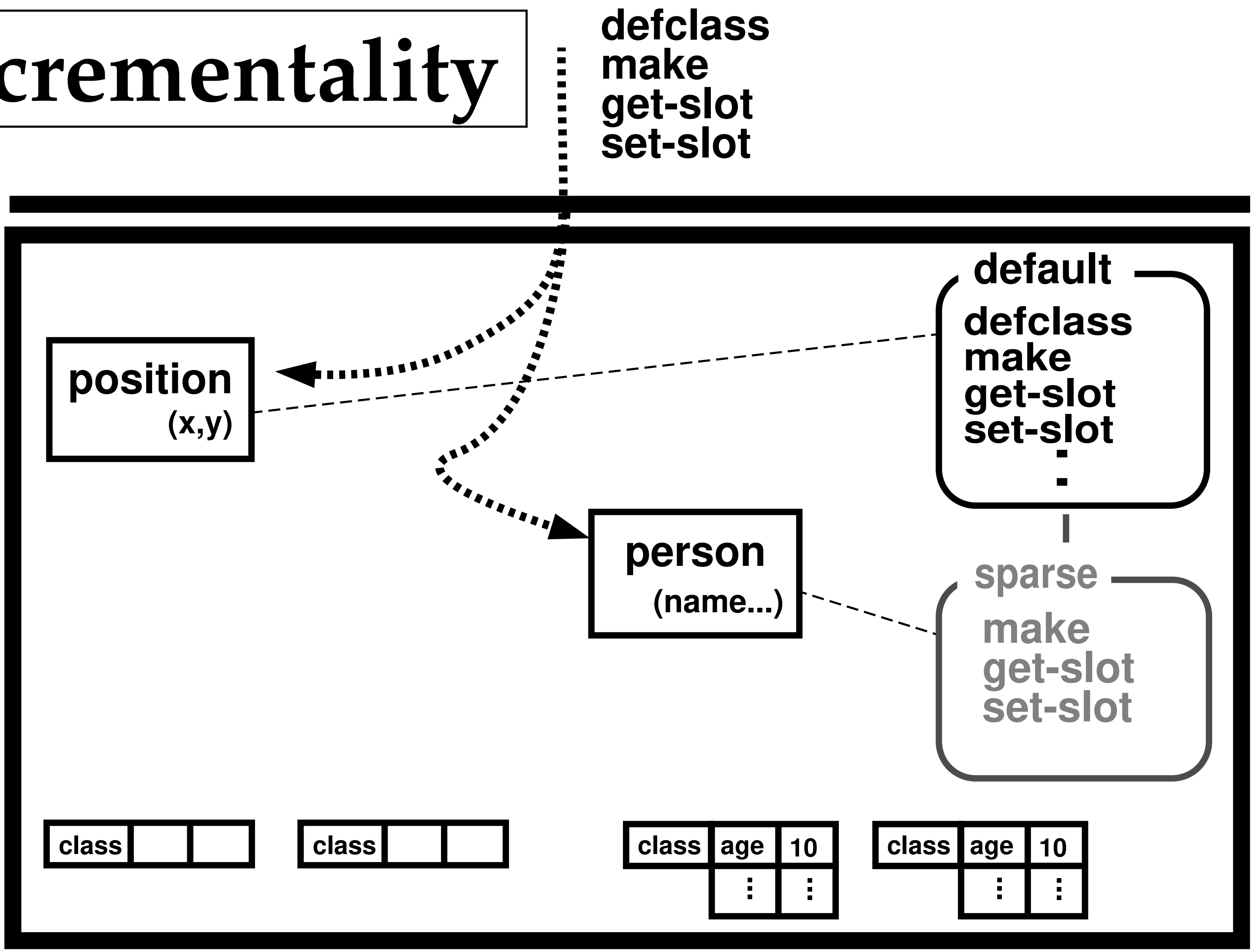


the implementation of object system primitives maps objects down onto more primitive structures

A Runtime Per Class



Incrementality



OOP provides inheritance and overriding

The Meta Code

```
(defclass sparse (default) ())

(defmethod make ((r sparse))
  (let ((new (allocate-instance r 1)))
    (%set new 0 '())
    new))

(defmethod get-slot ((r sparse object slot)
  (assoc (%ref object 0) slot))

(defmethod set-slot . . .)
```

Recovering Overhead

meta-level protocol adds too much overhead

two approaches:

- automatic techniques
  constant folding, partial-evaluation...

- curry the protocol
  - meta-level messages are sent only once, they return a memoizable-code fragment
  - inline resulting fragments

Curry
The Protocol

make → how-to-make
get-slot → how-to-get-slot
set-slot → how-to-set-slot

return “residual”

```
(defmethod how-to-make ((r sparse))
  #'(lambda ()
      (let ((new (allocate-instance r 1)))
        (%set new 0 '())
        new)))

(defmethod how-to-get-slot ((r sparse) slot-name)
  #'(lambda (object)
      (assoc (%ref object 0)
             slot-name)))

(defmethod how-to-set-slot . . .)
```

More Details

‘Art of the Metaobject Protocol’

‘Open Implementations and Metaobject Protocols’

EuLisp (Telos)

IMSA’92 Workshop

Also papers in:
ftp://ftp.parc.xerox.com/pub/openimpls

Other PARC MOPs

Silica [Rao]

window system

"weight"
(cache or recompute all the time)

Anibus [Rodriguez]

statically parallelizing Scheme Compiler

distribution of work
distribution of data
synchronization

Intrigue [Lamping..] [Kiczales..]

Scheme Compiler

closure
pair
environment

layout of runtime structures

Joshua*: Uniform Access to Heterogeneous Knowledge Structures
or
Why Joshing is Better than Conniving or Planning

Steve Rowley, Howard Shrobe, Robert Cassels
Symbolics Cambridge Research Center
Walter Hamscher
MIT Artificial Intelligence Laboratory

Its power comes from the observation that there is a Protocol of Inference consisting of a small set of abstract actions, each of which can be implemented in many ways.

High Performance Fortran: Status Report

Guy L. Steele Jr.
Thinking Machines Corporation

The underlying intuitions are that (a) an operation on two or more data objects is likely to be carried out much faster if they all reside in the same processor, and (b) it may be possible to carry out many such operations concurrently if they can be performed on different processors.

The HPF directives address point (a) by providing a way to direct the compiler to ensure that certain data objects will reside in the same processor:

```
       REAL A(1000,1000),B(998,998)
CHPF$  ALIGN B(I,J) WITH A(I+1,J+1)
CHPF$  DISTRIBUTE A(*,BLOCK)
```

The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System

Michael Young, Avadis Tevanian, Richard Rashid, David Golub,
Jeffrey Eppinger, Jonathan Chew, William Bolosky,
David Black and Robert Baron
Carnegie-Mellon University

An important component of the Mach design is the use of memory objects which can be managed either by the kernel or by user programs through a message interface. This feature allows applications such as transaction management systems to participate in decisions regarding secondary storage management and page replacement.

Technical Report 90-09-05

Extending the Mach External Pager Interface to Allow User-Level Page Replacement Policies

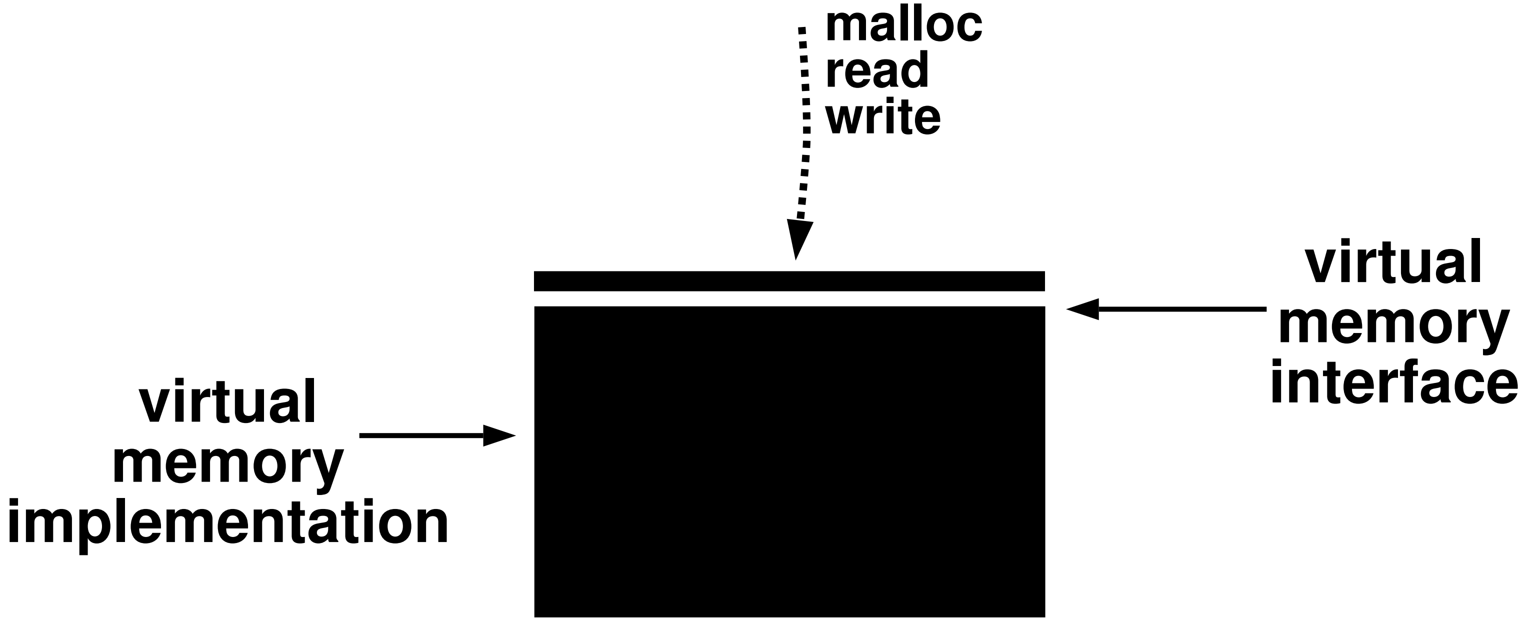
Dylan McNamee & Katherine Armstrong
Department of Computer Science & Engineering
University of Washington

OOPSLA'93

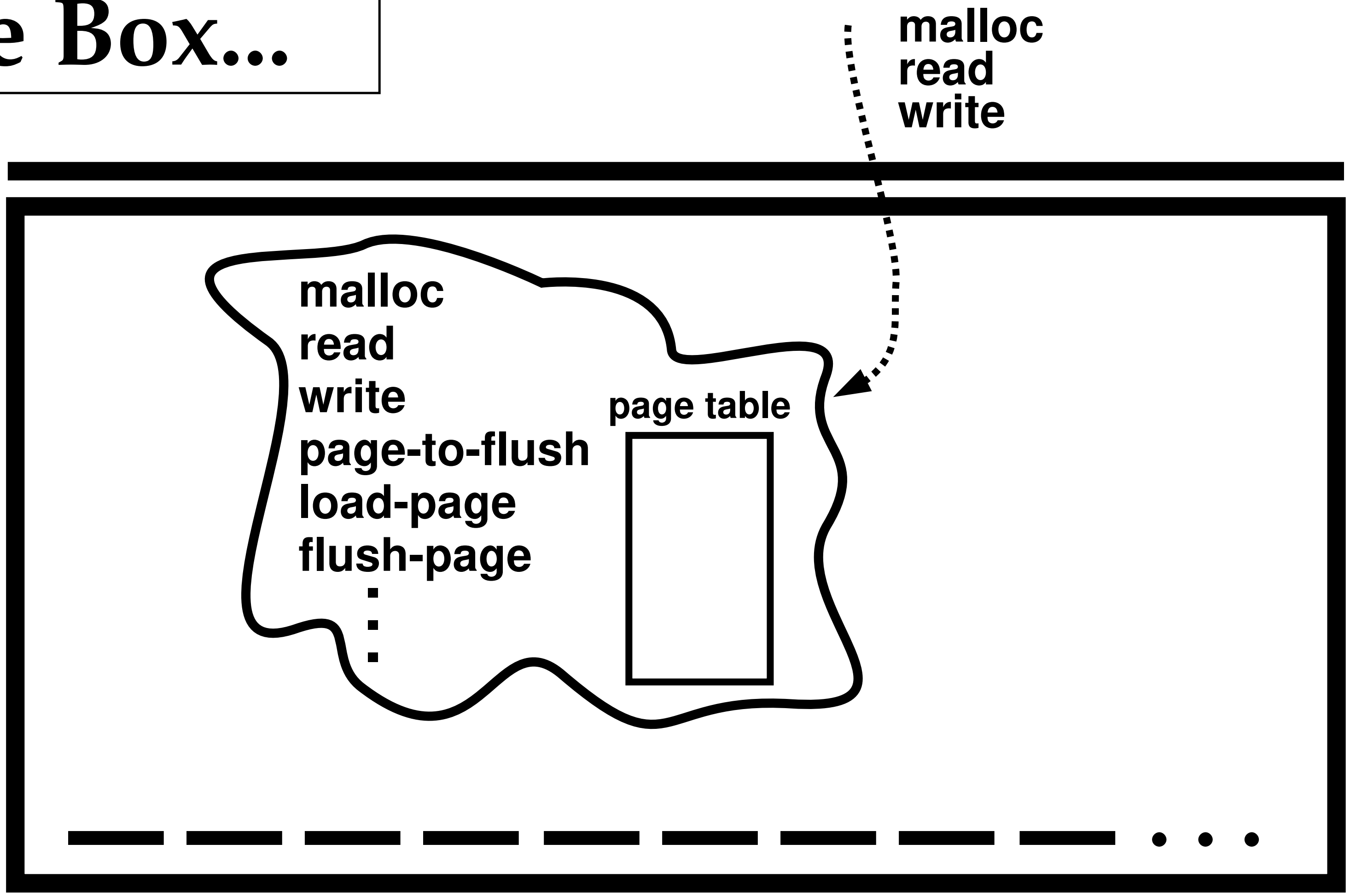
Tools for the Development of Application-Specific Virtual Memory Management

Keith Drueger, David Loftesness, Amin Vahdat, and Thomas Anderson
Computer Science Division
University of California

Virtual Memory

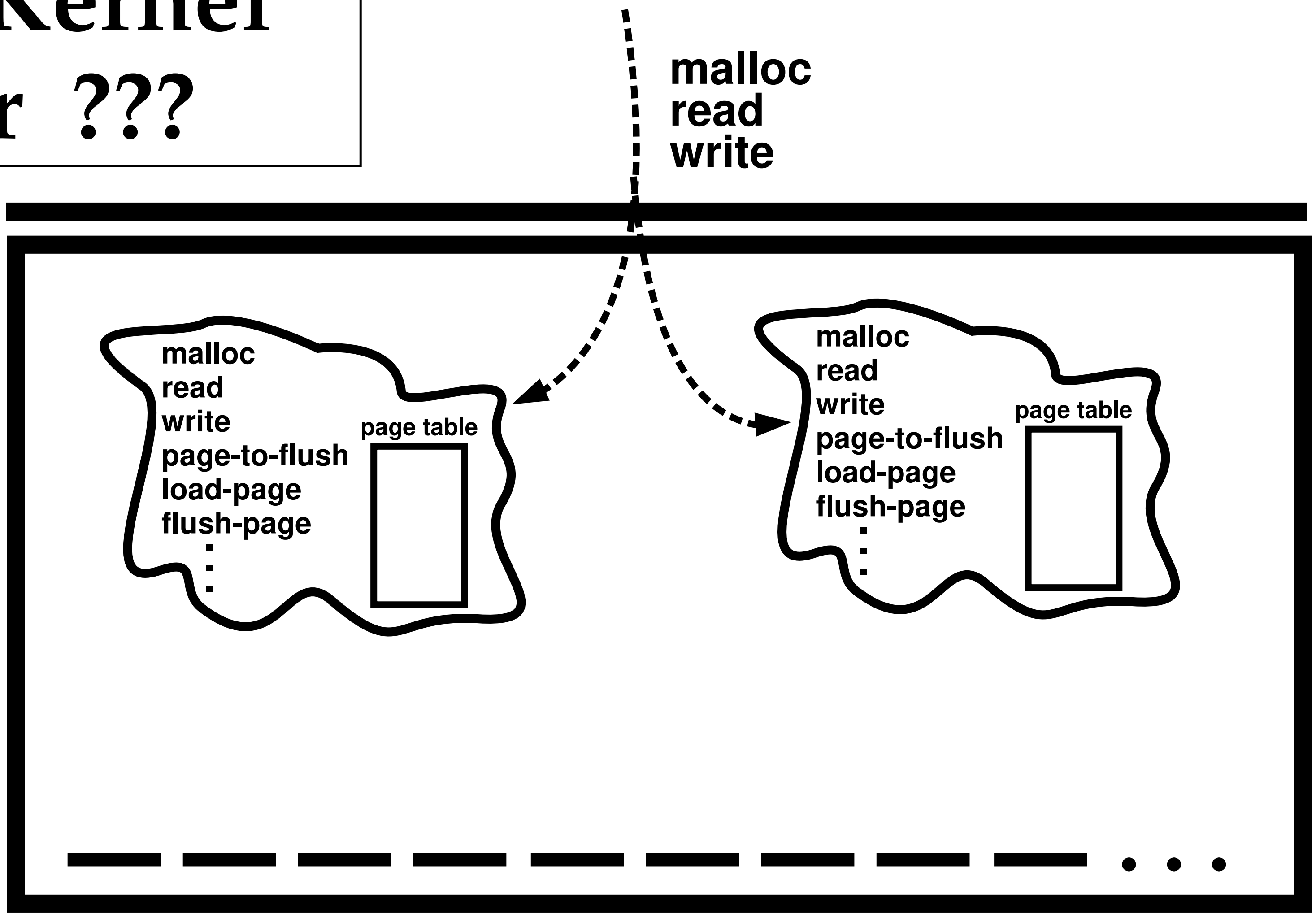


Inside the Box...



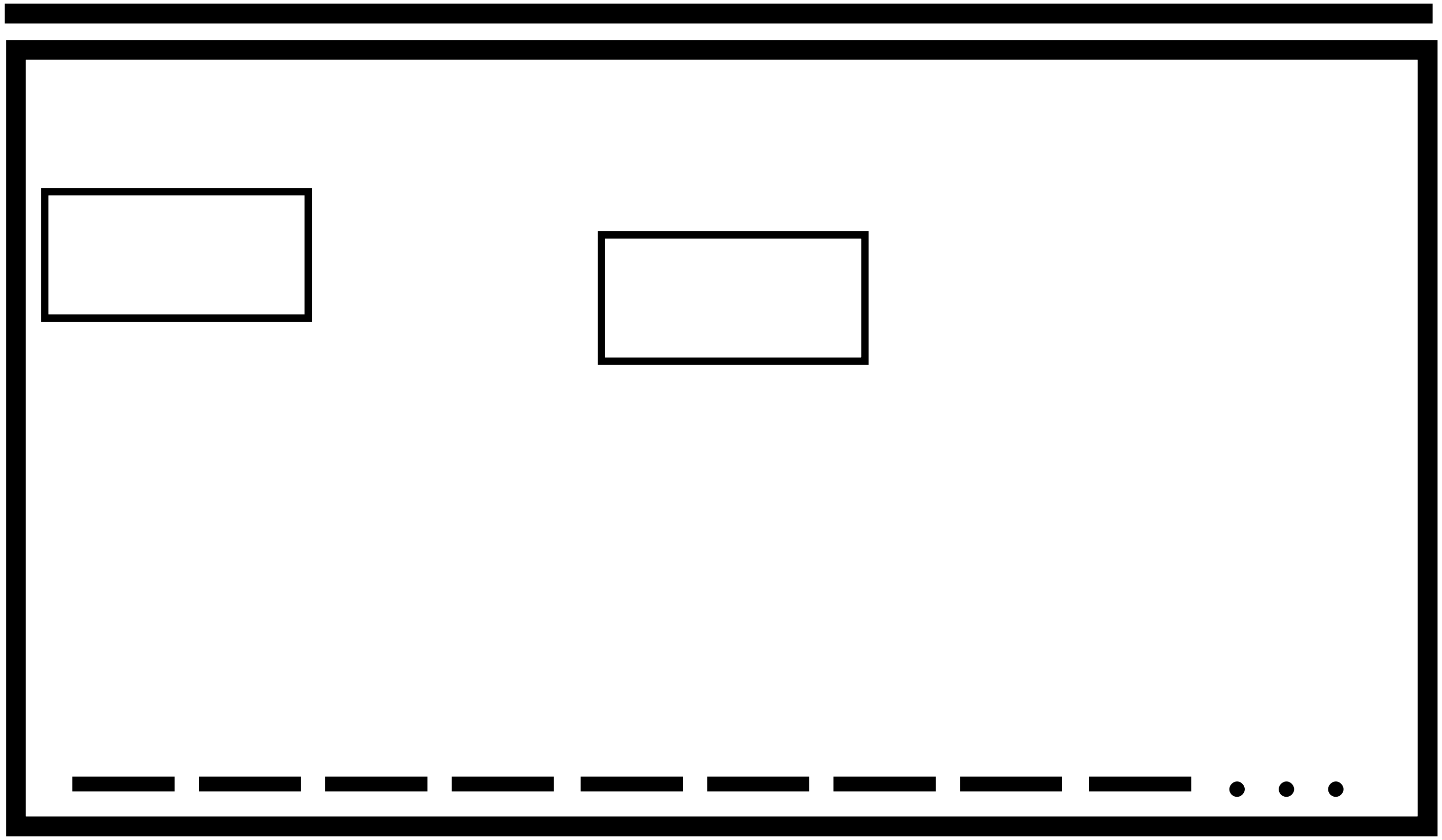
implementation maps virtual addresses to physical pages

A Kernel Per ???

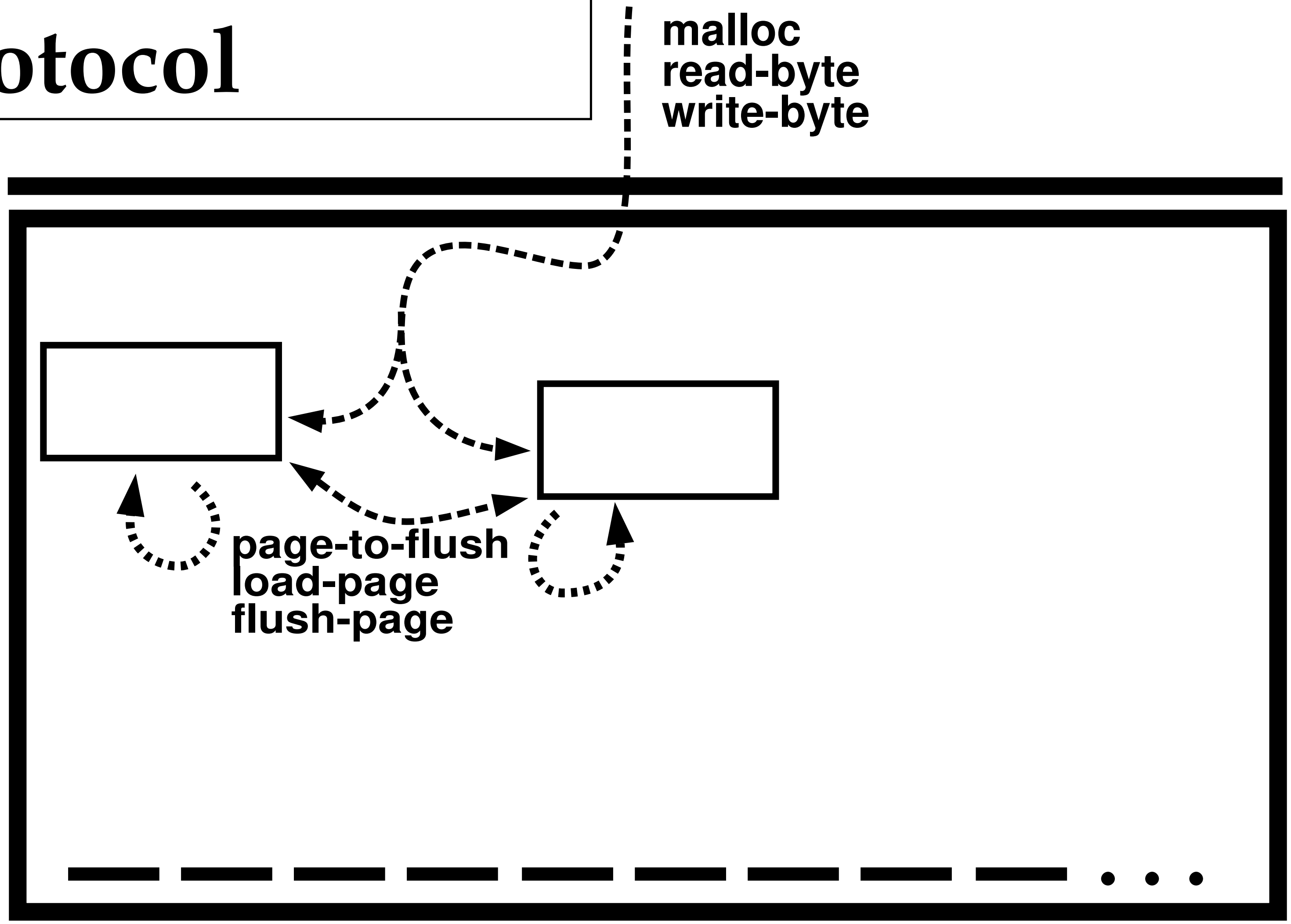


we'd like different regions of memory to have different paging policies and/or mechanisms

Regions as Explicit Objects

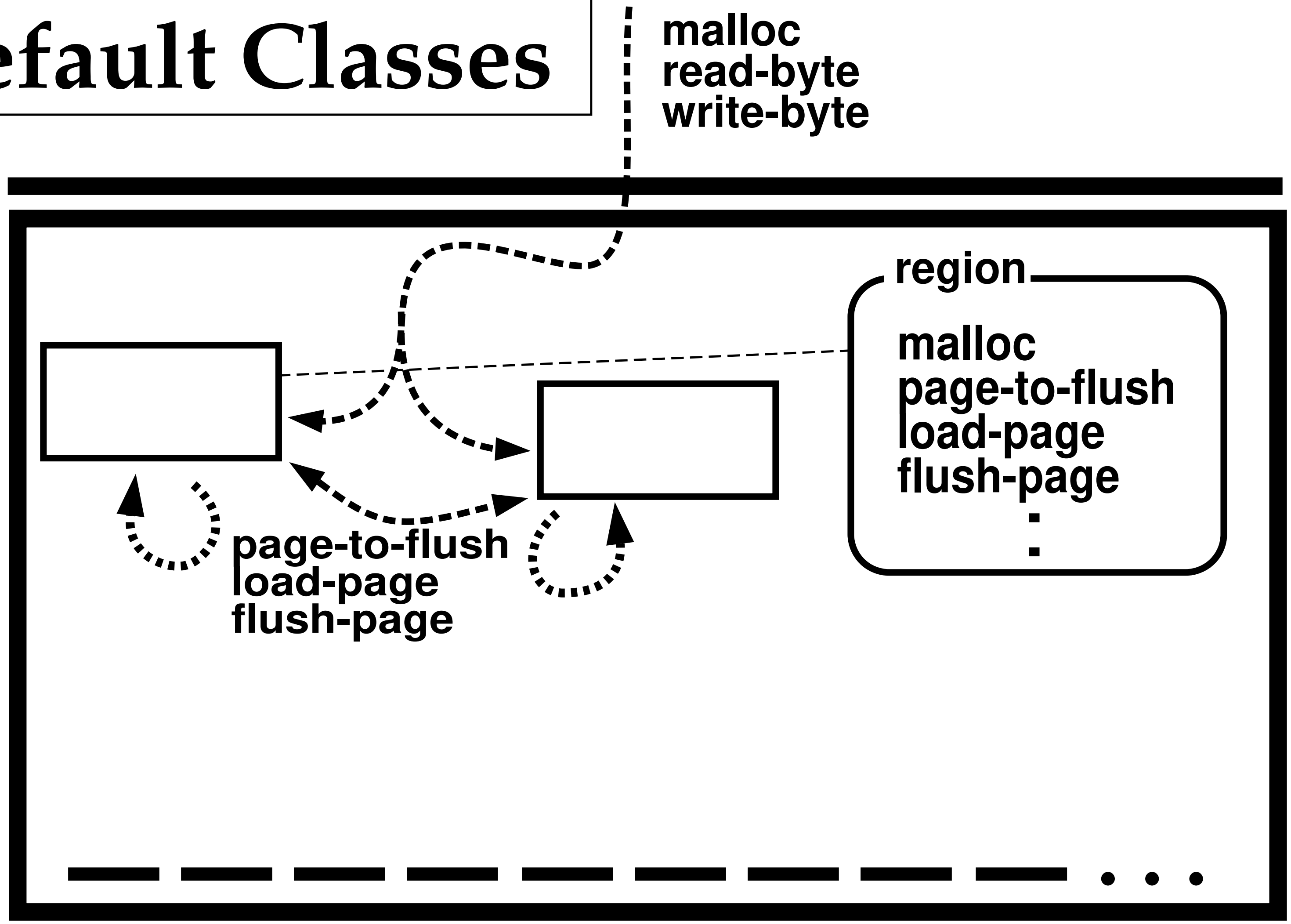


A Well-Defined Protocol

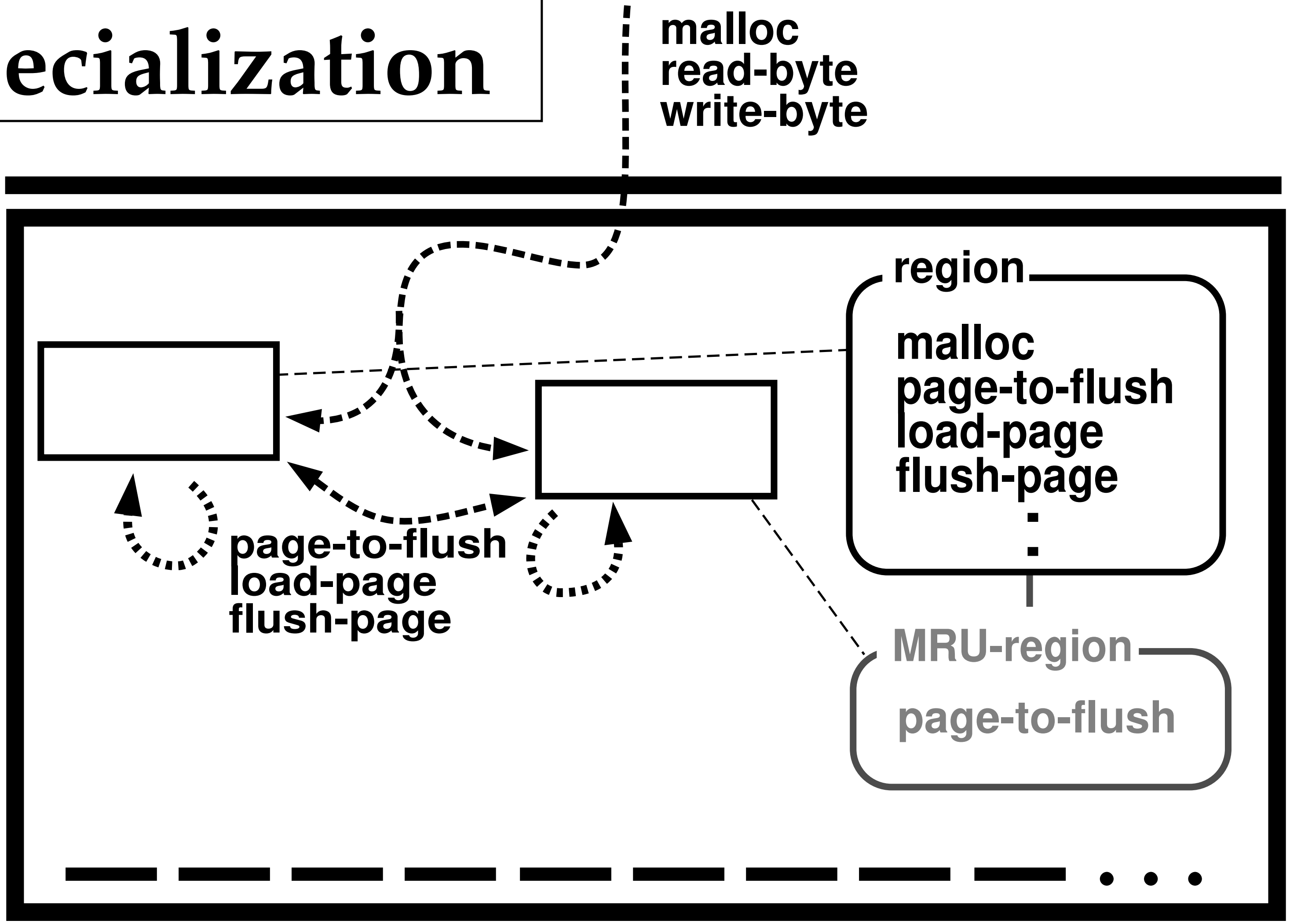


OOP provides polymorphism

With Default Classes



That Allow Specialization



OOP provides inheritance and overriding

Metaobject Protocols

reified units of interface divide program (class, region)

internal protocol among metaobjects partitions implementation and exposes mapping dilemmas (get-slot, page-to-flush)

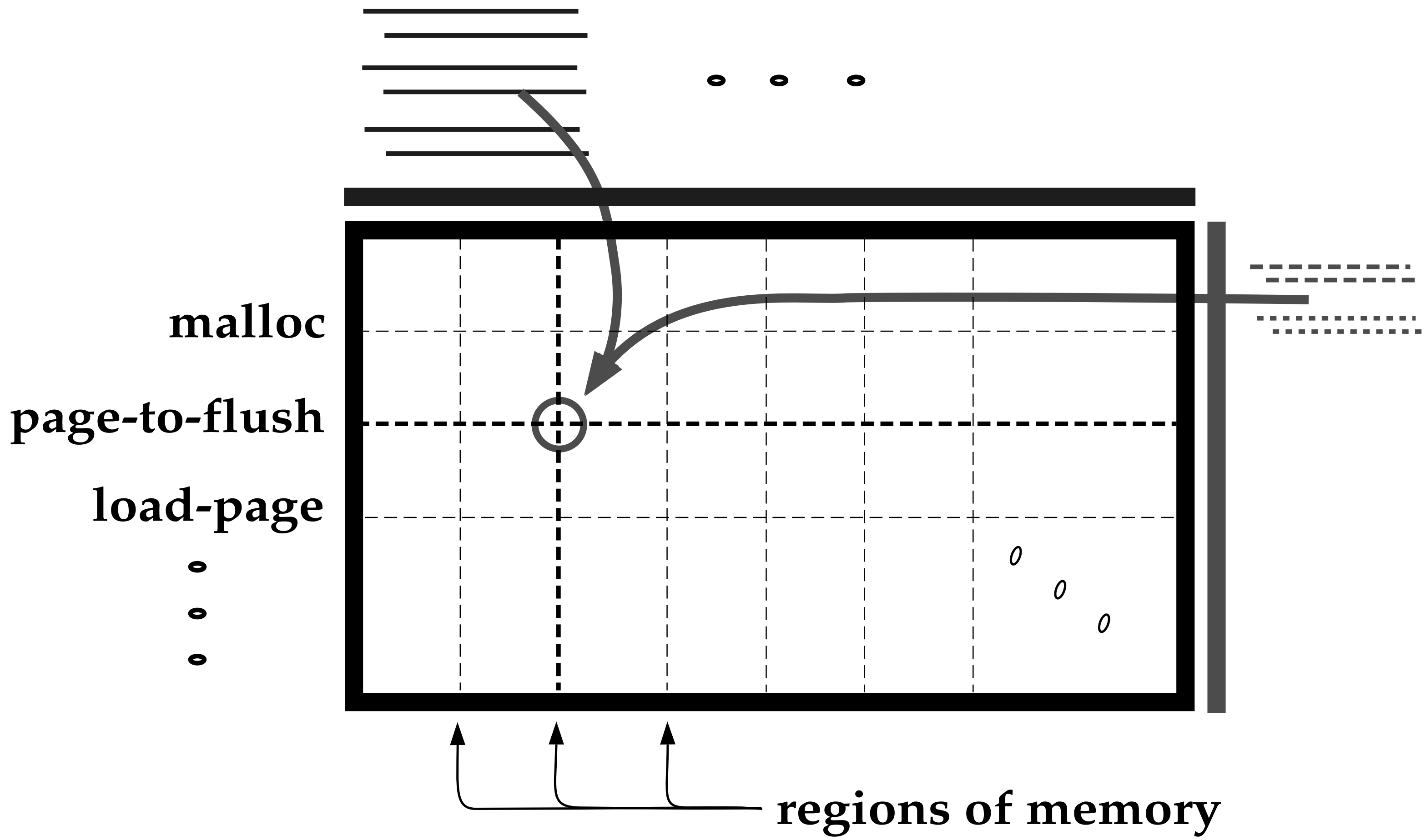
default implementation in default classes

cross product allows replacement of part of the implementation underlying part of the client program

$$\textbf{class} \times \textbf{get-slot}$$
$$\textbf{region} \times \textbf{page-to-flush}$$

Coupled Programs

normal programs build on top of the normal interface, while associated adjustment programs locally adjust the normal interface's implementation



An Open Scheme Compiler

Mapping dilemmas:

- closure layout
- pair layout
- environment layout
- ⋮
- calling convention
- memory management

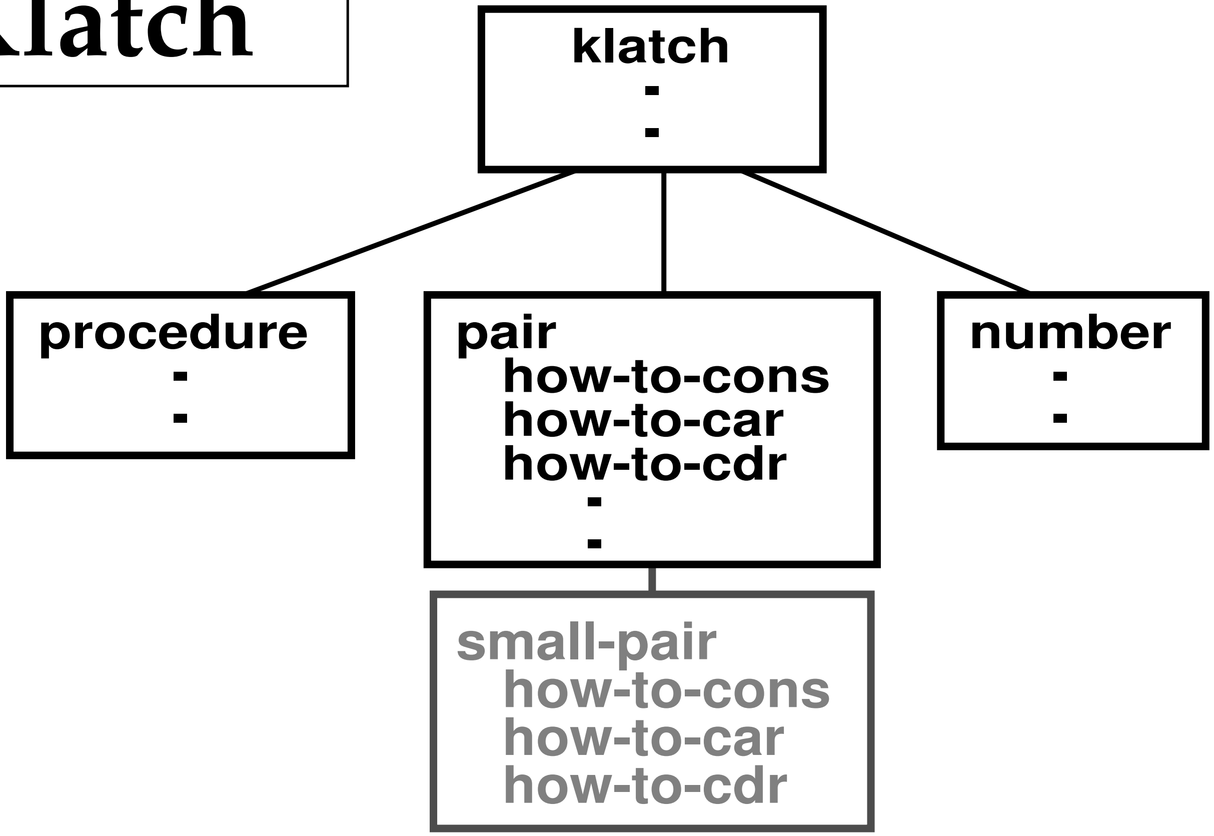
simple, elegant programs that are as fast as hand-coded C ?

Units of Interface?

"klatch" is the transitive closure of
procedures: lambdas and calls
pairs: cons and car/cdr
.
.
.

```
(make-position .. ..)

                  (make-position .. ..)

(define make-position (lambda (x y) (cons x y)))

(define pos-x (lambda (pos) (car pos)))
(define pos-y (lambda (pos) (cdr pos)))

        (pos-x ..)          (pos-x ..)
          (pos-y ..)            (pos-y ..)
```

Classes of Klatch



it's a bit more complicated of course!

Other Examples

lots of similar work and ideas, at different levels of generality

mapping dilemmas

high-level functionality that needs to be efficient

<u>Languages</u>
- pragmas
- Joshua
- Hermes
- HPF
- parallelization
- .
- .
- .

<u>OS</u>
- VM/370
- Mach
- [McNamee. . .]
- Scheduler Activations
- Apertos
- FLEX
- Taligent
- Choices
- Spring
- .
- .

<u>Distributed Sys</u>
- ABCL/R
- Open C++
- AL-1/D
- OLE
- .
- .

<u>Databases</u>
- .
- .

ACM TOCS Feb. 1992

Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism

Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, and Henry M. Levy
Department of Computer Science and Engineering
University of Washington

This paper addresses this dilemma. First, we argue that the performance of kernel threads is *inherently* worse than that of user-level threads, rather than this being an artifact of existing implementations; we thus argue that managing parallelism at the user level is essential to high-performance parallel computing.

The difficulty in achieving these goals in a multiprogrammed multiprocessor is that the necessary control and scheduling information is distributed between the kernel and each application's address space. The kernel, in order to effectively allocate processors to applications, needs access to user-level scheduling information (e.g., how much parallelism there is in each address space). The user-level support software, in order to effectively manage parallelism, needs to be aware of certain state transitions that are hidden in the kernel (e.g., processor re-allocations and I/O events).

Appears in ACM Operating Systems Review,
V 27(2), April 1993, pp. 21-34

A Status Report on Research in Transparent Informed Prefetching

R. Hugo Patterson, Garth A. Gibson, M. Satyanarayanan
Carnegie Mellon University

Abstract

This paper focuses on extending the power of caching and prefetching to reduce file read latencies by exploiting application level hints about future I/O accesses. We argue that systems that disclose high-level knowledge can transfer optimization information across module boundaries in a manner consistent with sound software engineering principles. Such Transparent Informed Prefetching (TIP) systems provide a technique for converting the high throughput of new technologies such as disk arrays and log-structured file systems into low latency for . . .

The Case for Run-Time Replaceable Kernel Modules

Richard Draves
Microsoft Research

1. Introduction

Kernel modules encapsulate too many policy and implementation trade-offs. No matter how you design your VM, IPC, scheduling, or IO sub-systems, you will make some customers unhappy. I argue that an operating system kernel that allows the run-time *replacemen*t of modules is an appropriate solution, especially for consumer-oriented environments.

The replacement of core kernel modules allows applications to solve feature-deficiency, performance, and version-skew problems. Seamless replacement at run time allows other active applications to continue undisturbed. For example, a primitive virtual memory system might be replaced with one that supports mapped files and copy-on-write. A time-sharing scheduler might be upgraded with one that supports dead-line scheduling. An interprocess communication facility might be replaced to make a particular combination of communication options more efficient. An application known to tickle a bug in an old version of the MMU-handling code might load a more recent version of this machine-dependent code.

submitted to IWOOOS '93

Efficient Support for Incremental Customization of OS Services

Peter Druschel
Department of Computer Science
The University of Arizona

Abstract

This paper argues that efficient, incremental customization of OS services can be achieved using a two-fold strategy: an object-oriented architecture that relies on composition to facilitate code reuse and customization; and, an OS structure that places a minimal set of trusted functions into the kernel, with all remaining services collocated with application code in user-level protection domains

A Customisable Memory Management Framework

Giuseppe Attardi
Tito Flagella

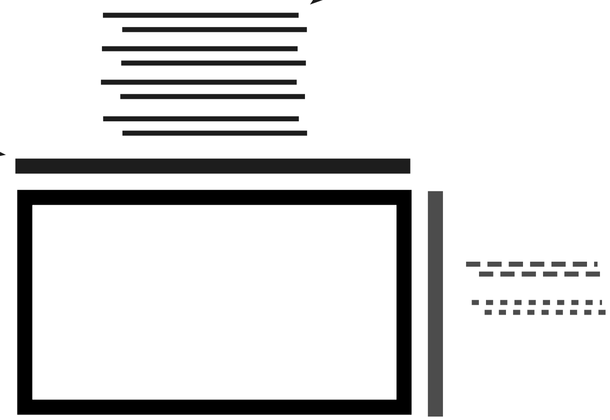
Abstract

Memory management is a critical issue for many large object-oriented applications, but in C++ only explicit memory reclamation through the **delete** operator is generally available. We analyse different possibilities for memory management in C++ and present a dynamic memory management framework which can be customised to the need of specific applications.

Two Interfaces

what to build on top of Object System, Virtual Memory, Window System...

Base interface provides functionality



adjustments to Object System, Virtual Memory, Window System...

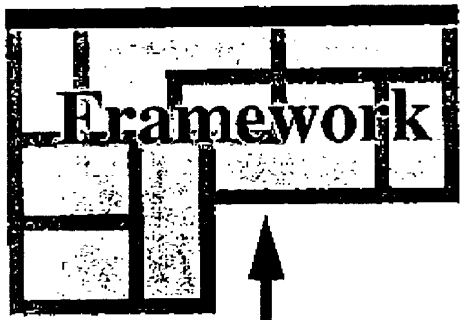
Meta interface provides control over mapping decisions

″separating control over mapping dilemmas“ replaces ″hiding implementation“

Use of Frameworks

Client API

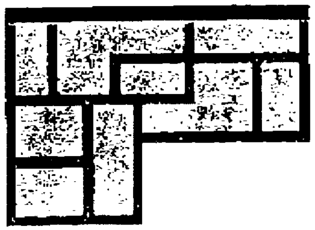
Provides services
Looks like a class library
Instantiate and use
Your code calls it



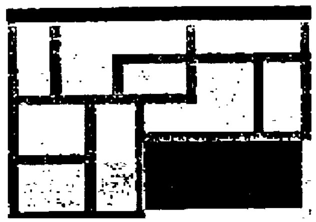
= Implementation + Integration

Framework API

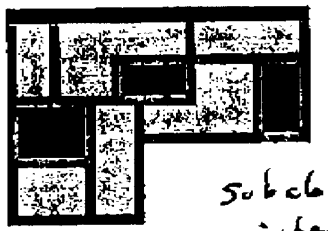
Provides customizability
Looks like an OO design
Subclass and override
Calls your code



Use "as is"



Complete



Customize

subclass & override internal objects

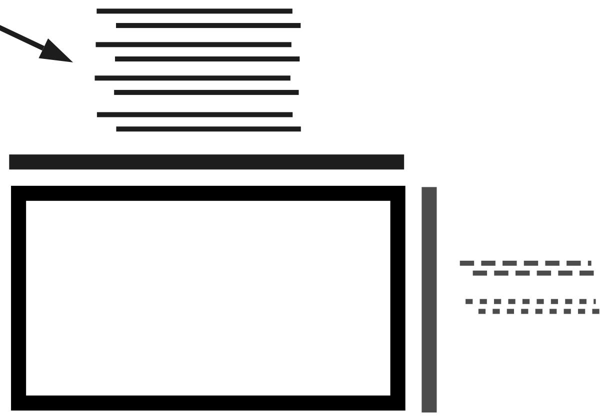
Taligent.

Debbie Coutant 4/11/94

Basic Design Goal

allow client programmer to alternately focus their attention mostly on one program or the other

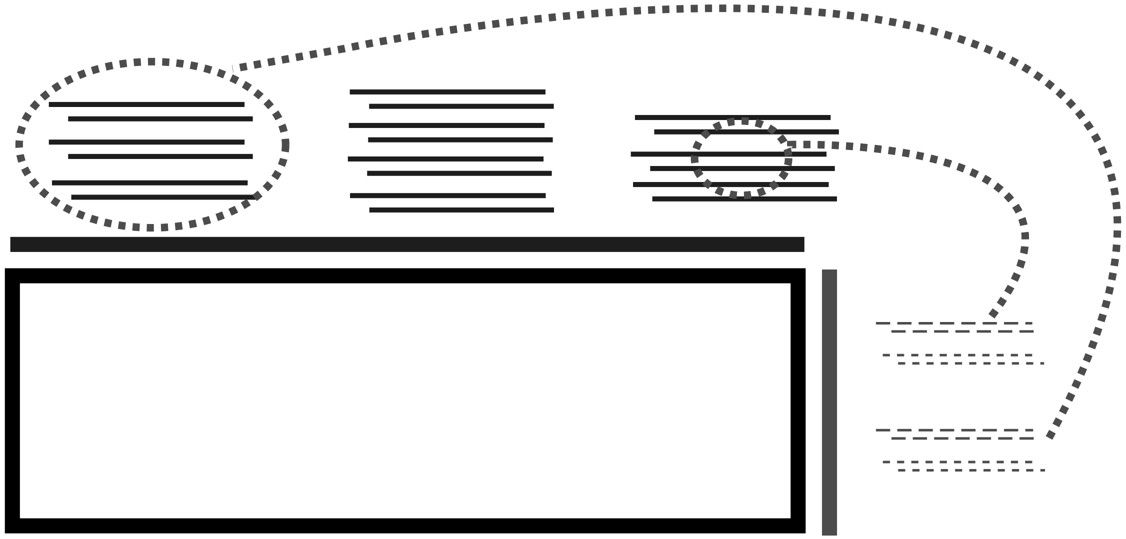
program optimistically



adjust to meet optimism

”if you can’t conquer, at least divide“

Secondary Principles of Modularity



incrementality
deltas from good default

scope control
what is affected

interoperability

Success Metrics

- reduced complexity of <u>client</u> <u>code</u>
- reduced size of <u>client</u> <u>code</u>
- increased reuse of client and substrate code
- interoperability benefit

But, these systems are definitely harder to design and build.

Main Technical Problems

Localizing the change
- traditional OO -> "cartesian coordinates"
- Traces
- Spindles

Recovering the dispatch overhead
- partial evaluation, currying . . .

Safety
- separate address spaces
- safe meta-language

It's All About Binding Time

Substrate Design | Application Design | Compile | Load | Run

← easier to recover decision making overhead

more is known, potentially better decisions →

$$\frac{T_{\text{worst decision}}}{T_{\text{best decision}}}$$

Desire

$$\frac{T_{\text{overhead}}}{T_{\text{best}}}$$

Cost

FLEX: A Tool for Building Efficient and Flexible Systems

John B. Carter, Bryan Ford, Mike Hibler, Ravindra Kuramkote, Jeffrey Law, Jay Lepreau, Douglas B. Orr, Leigh Stroller, and Mark Swanson
University of Utah, Department of Computer Science

Abstract

Modern operating systems must support a wide variety of services for a diverse set of users. Designers of these systems face a tradeoff between functionality and performance. Systems like Mach provide a set of general abstractions and attempt to handle every situation, which can lead to poor performance for common cases. Other systems, such as Unix, provide a small set of abstractions that can be made very efficient, at the expense of functionality. We are implementing a flexible systems building tool, FLEX, that allows us to support a powerful operating systems interface efficiently by constructing specialized module implementations at runtime. FLEX improves the performance of *existing* systems by optimizing interprocess communications paths and relocating servers and clients to reduce communications overhead. These facilities improve the performance of Unix system calls on Mach from 20-400%. Furthermore, FLEX can dynamically extend the kernel in a controlled fashion, which gives user programs access to privileged data and devices not envisioned by the original operating system implementor.

Active Abstractions

Waldemar Horwat
Massachusetts Institute of Technology

The idea of *abstraction* -- hiding irrelevant details of a module while defining an interface to specify its useage -- is central to computer science. In most of today's programming languages, abstractions are *passive* -- unaware of and unable to take advantage of structure in how the program uses them, except possibly by extrapolating from past events. Authors of passive abstractions cannot, in general, express to the compiler how such abstractions are to be optimized; optimization is limited to what the compiler can infer from primitive type properties. In contrast, an *active* abstraction is able to affect how it is represented using knowledge derived from the program that uses it.

Efficient Software-Based Fault Isolation

Robert Wahbe, Steven Lucco, Thomas E. Anderson, Susan L. Graham
Computer Science Division
University of California, Berkeley

In this paper, we present a software approach to implementing fault isolation within a single address space. Our approach has two parts. First, we load the code and data for a distrusted module into its own *fault domain*, a logically separate portion of the application's address space. Second, we modify the object code of a distrusted module to prevent it from writing or jumping to an address outside its fault domain.

Service without Servers

Chris Maeda and Brian N. Bershad
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh PA 15213
cmaeda@cs.cmu.edu, bershad@cs.cmu.edu

August 1, 1993

Threads and Input/Output in the Synthesis Kernel

Henry Massalin and Calton Pu
Department of Computer Science
Columbia University

We have introduced the principles of code synthesis[5], which makes the Synthesis kernel fast for several reasons. First, frequently executed Synthesis kernel calls are "compiled" and optimized at run-time using ideas similar to currying and constant folding. For example, when we open a file for input, a custom-made (thus short and fast) read routine is returned for later read calls.

Other Problems

How to identify dilemmas

Methodology of use

Specification

Verification

Going through multiple layers

Functionality Dilemmas

How To Identify Dilemmas?

iteration
not first-time out
participatory design

dialogue

implementor ⟵⟶ **users**

bug reports
code
⋮

Methodology of Use

end-users

local support

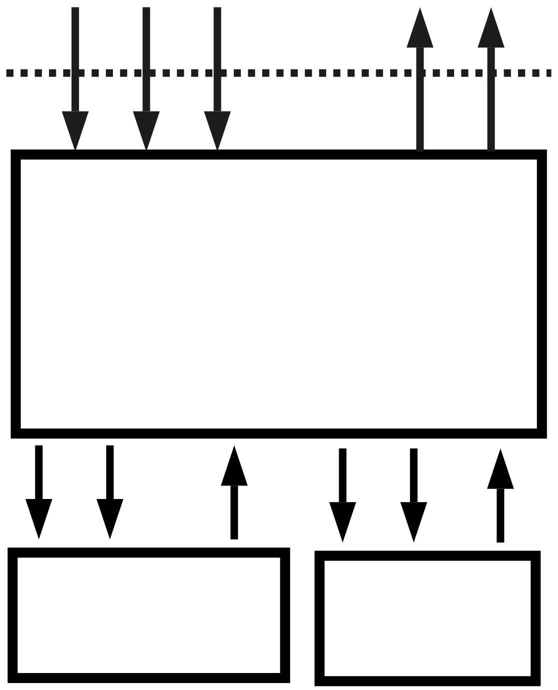
customer support

staff programmer

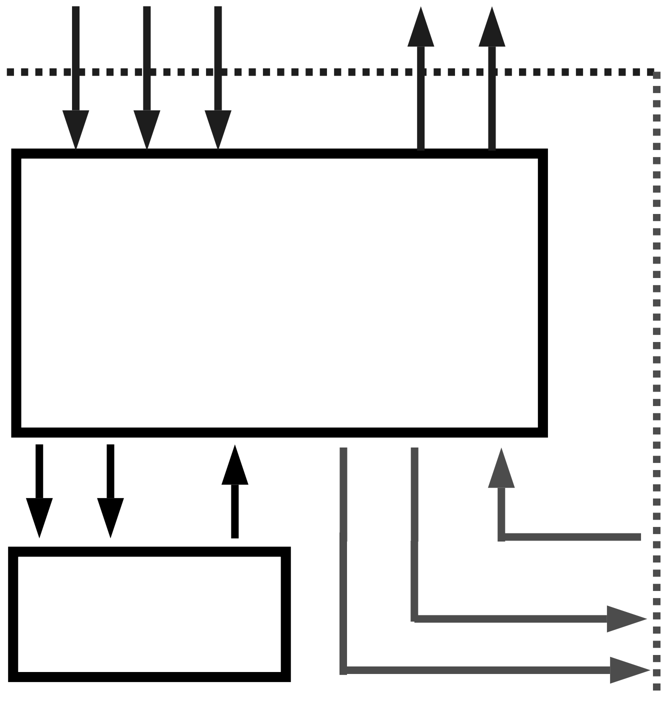
wizard

more declarative
more localized
more robust

Verification



traditional verification techniques can depend on verification of sub-components



in open implementations, client-supplied sub-components can be verified in the same way

Functionality Dilemmas

CLOS

single vs. multiple inheritance
multiple inheritance behavior
persistence

Window System

de-exposed typeout
overlapping vs. tiling

Virtual Memory

which backing store?

Policy/Mechanism Separation in Hydra

R. Levin, E. Cohen, W. Corwin, F. Pollack, W.Wulf
Carnegie-Mellon University

Abstract

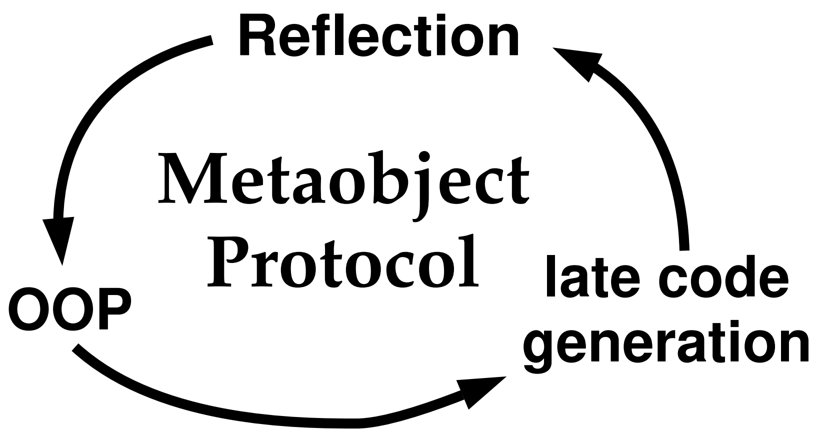
The extent to which resource allocation policies are entrusted to user-level software determines in large part the degree of flexibility present in an operating system. In Hydra the determination to separate mechanism and policy is established as a basic design principle and is implemented by the construction of a kernel composed (almost) entirely of mechanisms. This paper presents three such mechanisms (scheduling, paging, protection) and examines how external policies which manipulate them may be constructed. It is shown that the policy decisions which remain embedded in the kernel exist for the sole purpose of arbitrating conflicting requests for physical resources, and then only to the extent of guaranteeing fairness.

3 Points

Need: Open Implementations
(client control of mapping decisions)

Strategy: Separate access to functionality from control over mapping decisions. (Separation of concerns)

Technology:



To Do

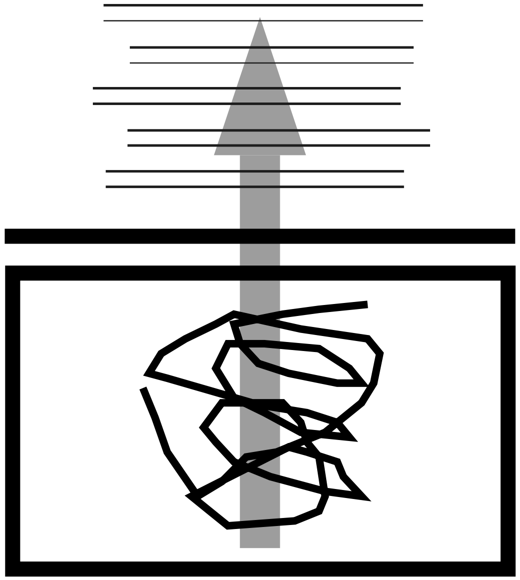
Improve Understanding of OI
- **Principles, Techniques, Methodologies...**

Work Out Implications
- **High-Level Computing Can Work?**

Improve Ad-Hoc OI efforts
- **Operating Systems**
- **Distributed Computing**
- ⋮

Identify new systems that should be OI
- **Emerging standard APIs**
- ⋮

What's Really Going On?

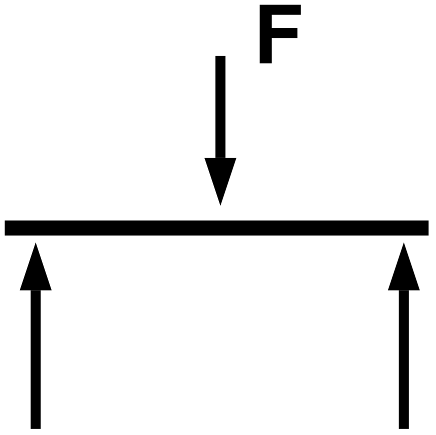


the abstraction is only a <u>partial</u> description of what goes on

the actual process consumes power, consumes memory, moves disk, interacts with network. . .

The Need for Abstraction

any real system is too complex to think about all at once. . .

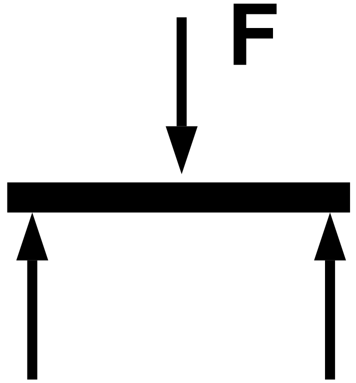


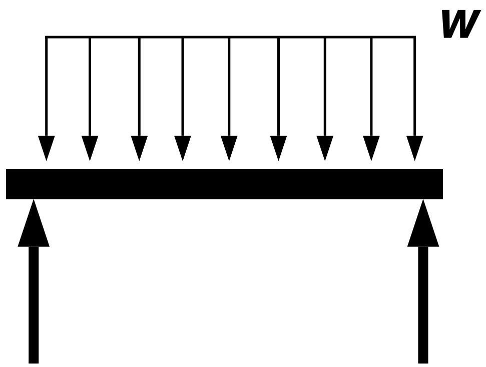
abstract descriptions capture some aspects of the system while eliding others

shear forces, creep, quantum mechanics, stability, corrosion resistance, dynamics, esthetic integrity. . .

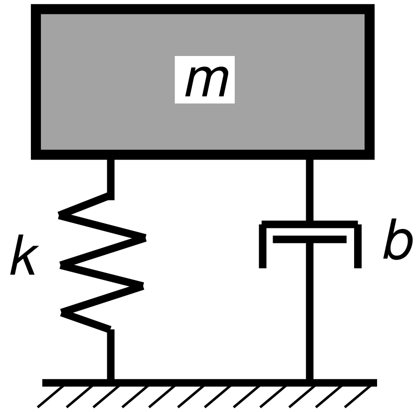
Multiple Descriptions

some capture more or less detail





others capture different issues entirely



other engineering disciplines have conventions for forming and coordinating multiple descriptions that capture <u>different</u> <u>aspects</u> of a system

Programs are Special

```
i: 1 → 100
j: 1 → 100
make-window (..)
```

abstract descriptions -- many things aren't said

can be "run" to automatically produce complete behavior

the substrate implementation fills in the missing pieces

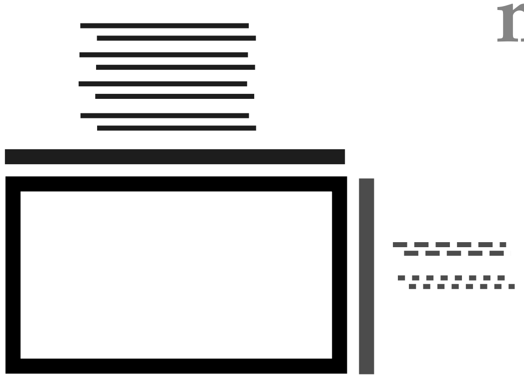
Long-Term Goal

the best
of
both worlds

abstract descriptions...
to manage complexity

multiple descriptions...
to cover all of what matters

automatic combination and execution...
because that’s what special about software



multiple, automatically combined and executed descriptions

a step in this direction...