chapter 8

implementation support

Implementation support

- programming tools
  - levels of services for programmers
- windowing systems
  - core support for separate and simultaneous user-system activity
- programming the application and control of dialogue
- interaction toolkits
  - bring programming closer to level of user perception
- user interface management systems
  - controls relationship between presentation and functionality

Introduction

How does HCI affect of the programmer?

Advances in coding have elevated programming

hardware specific

→ interaction-technique specific

Layers of development tools

- windowing systems
- interaction toolkits
- user interface management systems

Elements of windowing systems

Device independence

programming the abstract terminal device drivers

image models for output and (partially) input

- pixels
- PostScript (MacOS X, NextStep)
- Graphical Kernel System (GKS)
- Programmers' Hierarchical Interface to Graphics (PHIGS)

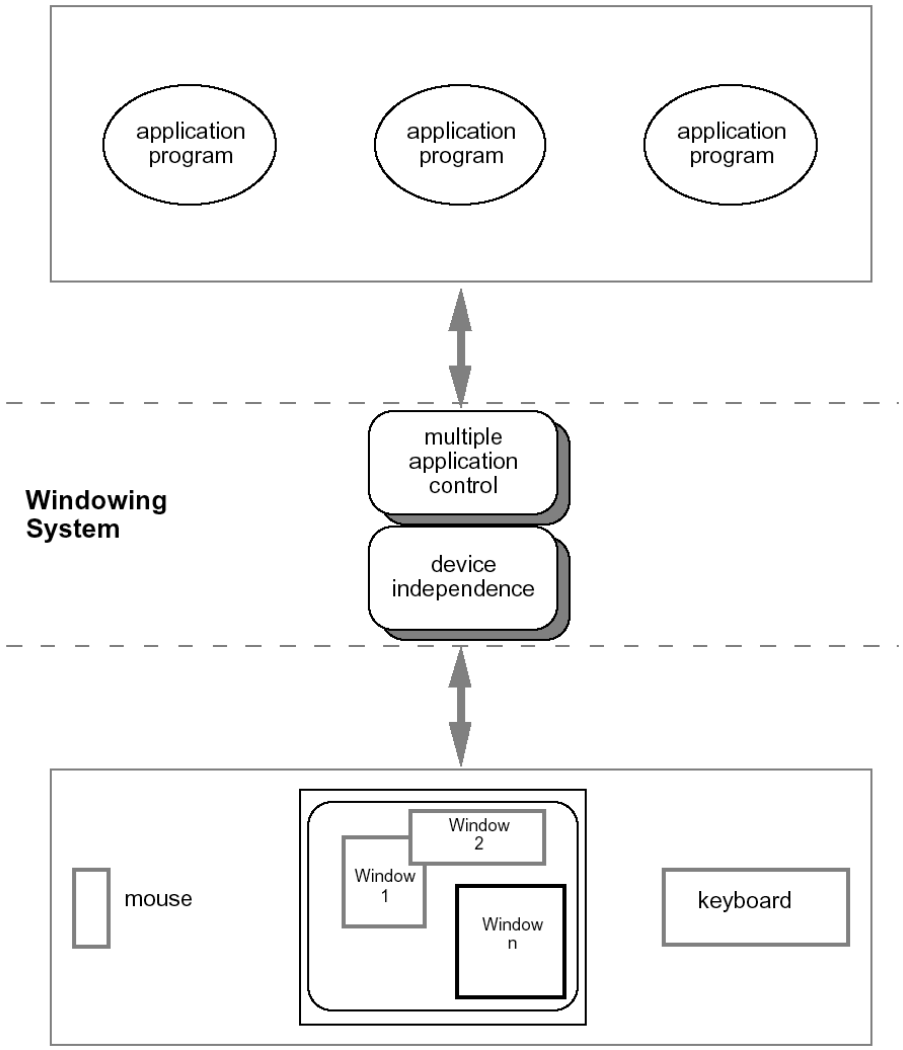
Resource sharing

achieving simultaneity of user tasks

window system supports independent processes

isolation of individual applications

roles of a windowing system



Architectures of windowing systems

three possible software architectures

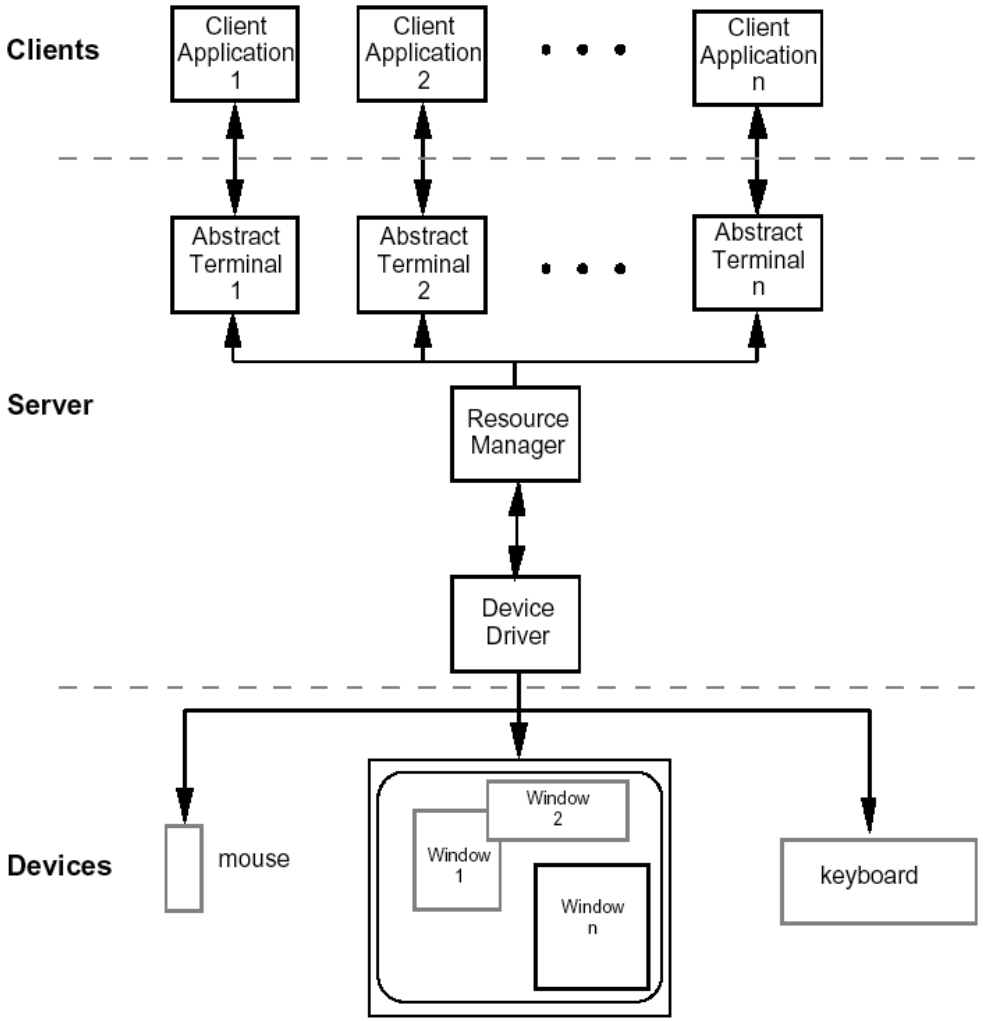
- all assume device driver is separate
- differ in how multiple application management is implemented

1. each application manages all processes
   - everyone worries about synchronization
   - reduces portability of applications

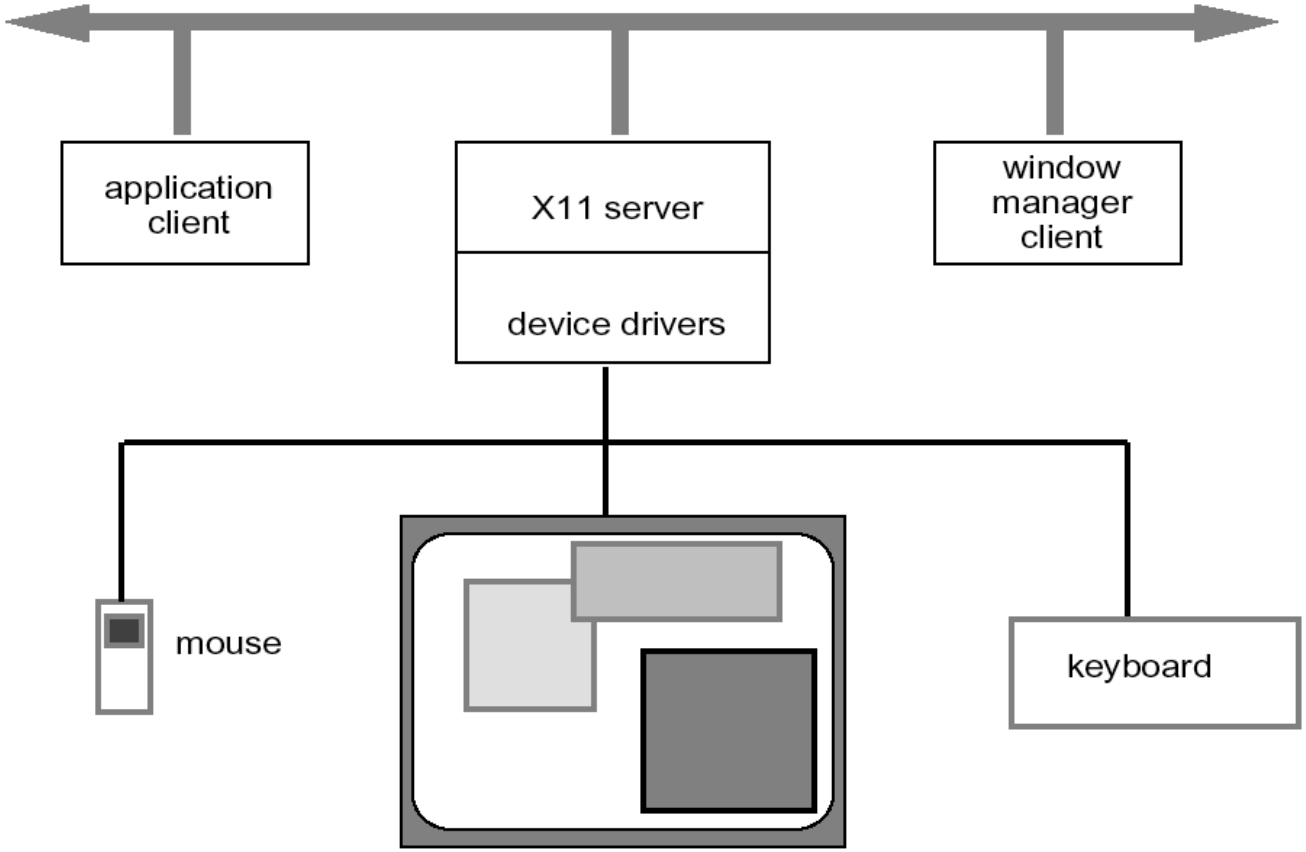
2. management role within kernel of operating system
   - applications tied to operating system

3. management role as separate application
   maximum portability

The client-server architecture



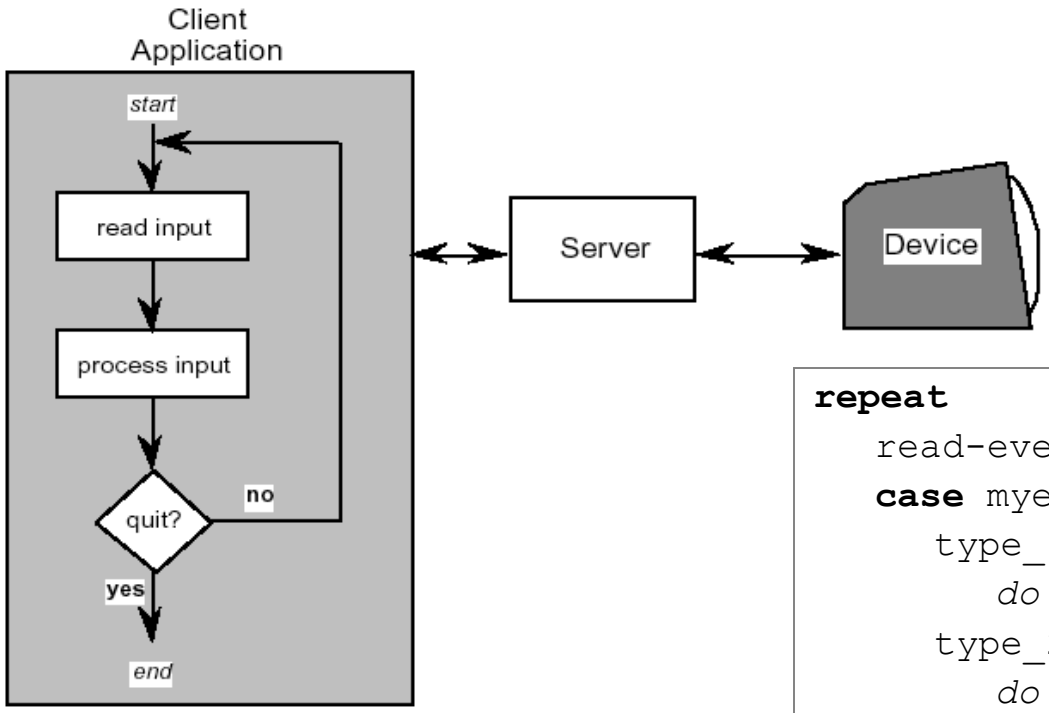
X Windows architecture



X Windows architecture (ctd)

- pixel imaging model with some pointing mechanism
- X protocol defines server-client communication
- separate window manager client enforces policies for input/output:
  - how to change input focus
  - tiled vs. overlapping windows
  - inter-client data transfer

Programming the application - 1
read-evaluation loop




```
repeat
   read-event(myevent)
   case myevent.type
      type_1:
         do type_1 processing
      type_2:
         do type_2 processing
      ...
      type_n:
         do type_n processing
   end case
end repeat
```

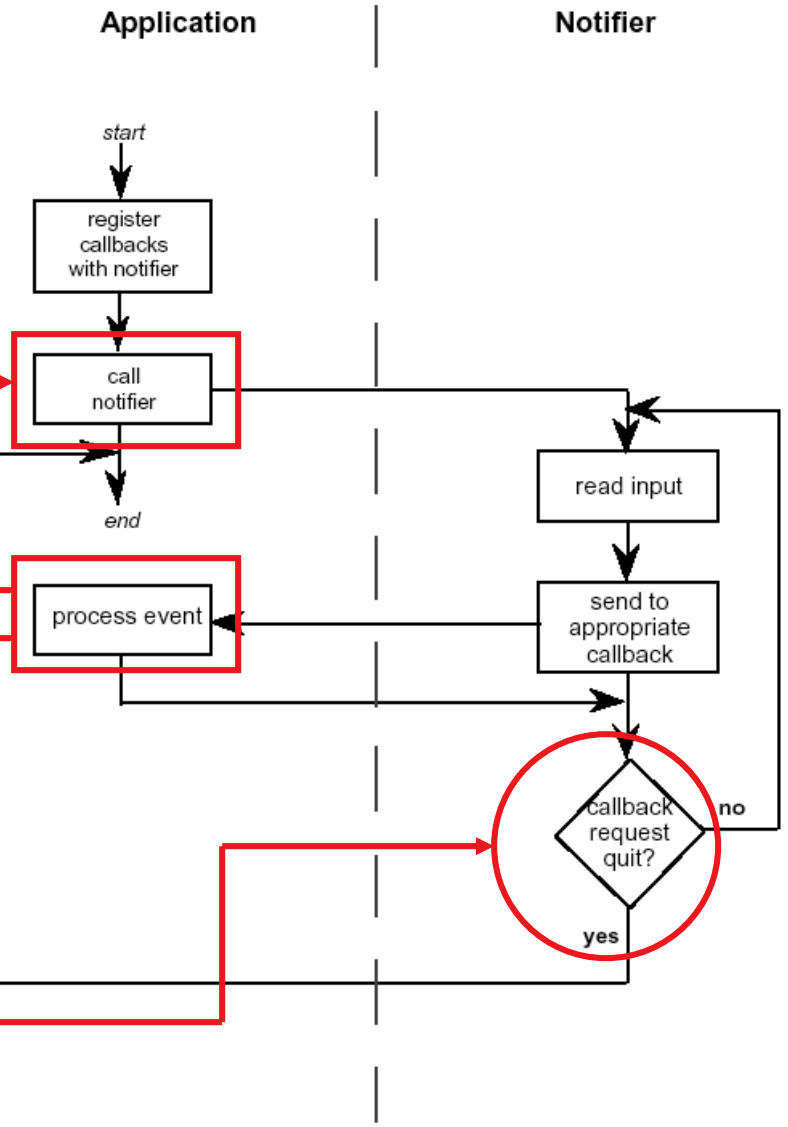
Programming the application - 1

notification-based

```
void main(String[] args) {
   Menu menu = new Menu();
   menu.setOption("Save");
   menu.setOption("Quit");
   menu.setAction("Save",mySave)
   menu.setAction("Quit",myQuit)
      ...
}

int mySave(Event e) {
   // save the current file
}

int myQuit(Event e) {
   // close down
}
```



going with the grain

- system style affects the interfaces
  - modal dialogue box
    - easy with event-loop (just have extra read-event loop)
    - hard with notification (need lots of mode flags)
  - non-modal dialogue box
    - hard with event-loop (very complicated main loop)
    - easy with notification (just add extra handler)

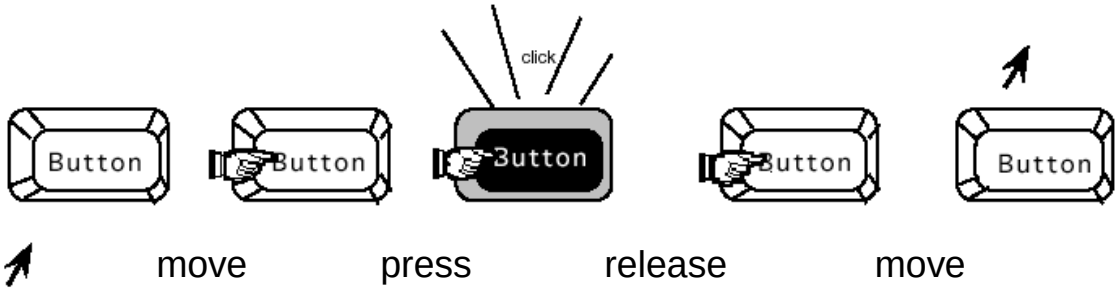
beware!

if you don't explicitly design it will just happen
implementation should not drive design

Using toolkits

Interaction objects

- input and output intrinsically linked



Toolkits provide this level of abstraction

- programming with interaction objects (or
- techniques, widgets, gadgets)
- promote consistency and generalizability
- through similar look and feel
- amenable to object-oriented programming

interfaces in Java

- Java toolkit – AWT (abstract windowing toolkit)
- Java classes for buttons, menus, etc.
- Notification based;
  - AWT 1.0 – need to subclass basic widgets
  - AWT 1.1 and beyond -– callback objects
- Swing toolkit
  - built on top of AWT – higher level features
  - uses MVC architecture (see later)

User Interface Management Systems (UIMS)

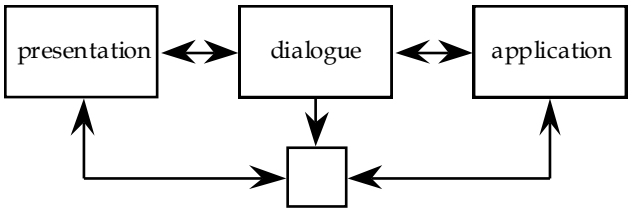
- UIMS add another level above toolkits
  - toolkits too difficult for non-programmers
- concerns of UIMS
  - conceptual architecture
  - implementation techniques
  - support infrastructure
- non-UIMS terms:
  - UI development system (UIDS)
  - UI development environment (UIDE)
    - e.g. Visual Basic

UIMS as conceptual architecture

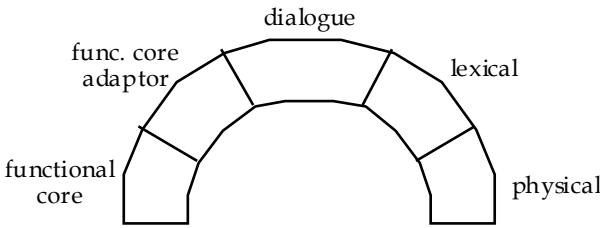
- *separation* between application semantics and presentation
- improves:
  - portability – runs on different systems
  - reusability – components reused cutting costs
  - multiple interfaces – accessing same functionality
  - customizability – by designer and user

UIMS tradition – interface layers / logical components

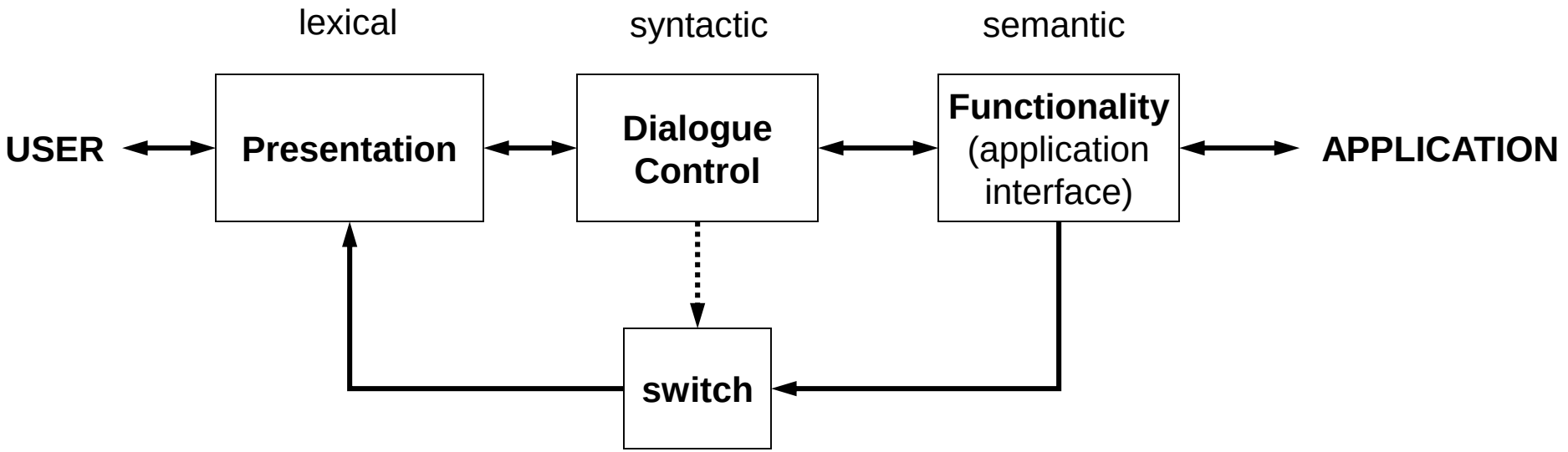
- linguistic: lexical/syntactic/semantic
- Seeheim:



- Arch/Slinky



Seeheim model



conceptual vs. implementation

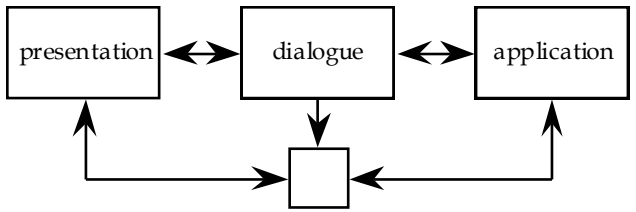
Seeheim

- arose out of implementation experience
- but principal contribution is conceptual
- concepts part of 'normal' UI language

… because of Seeheim …
… we think differently!

e.g. the lower box, the switch

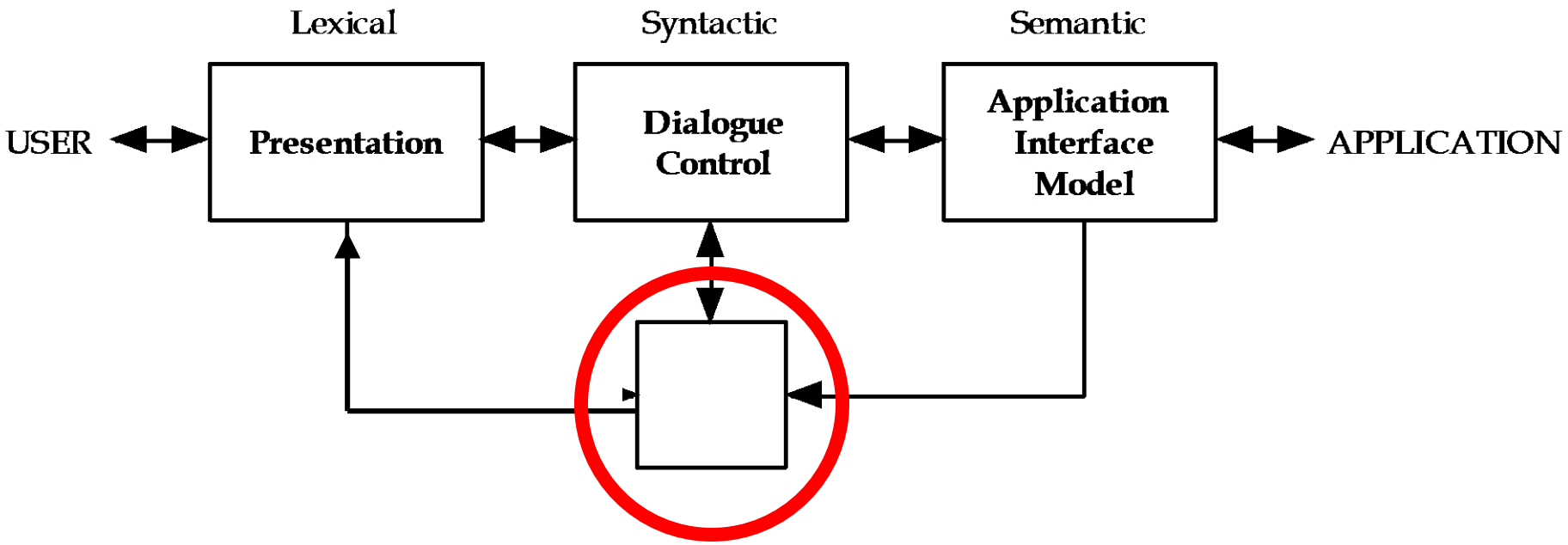
- needed for implementation
- but not conceptual



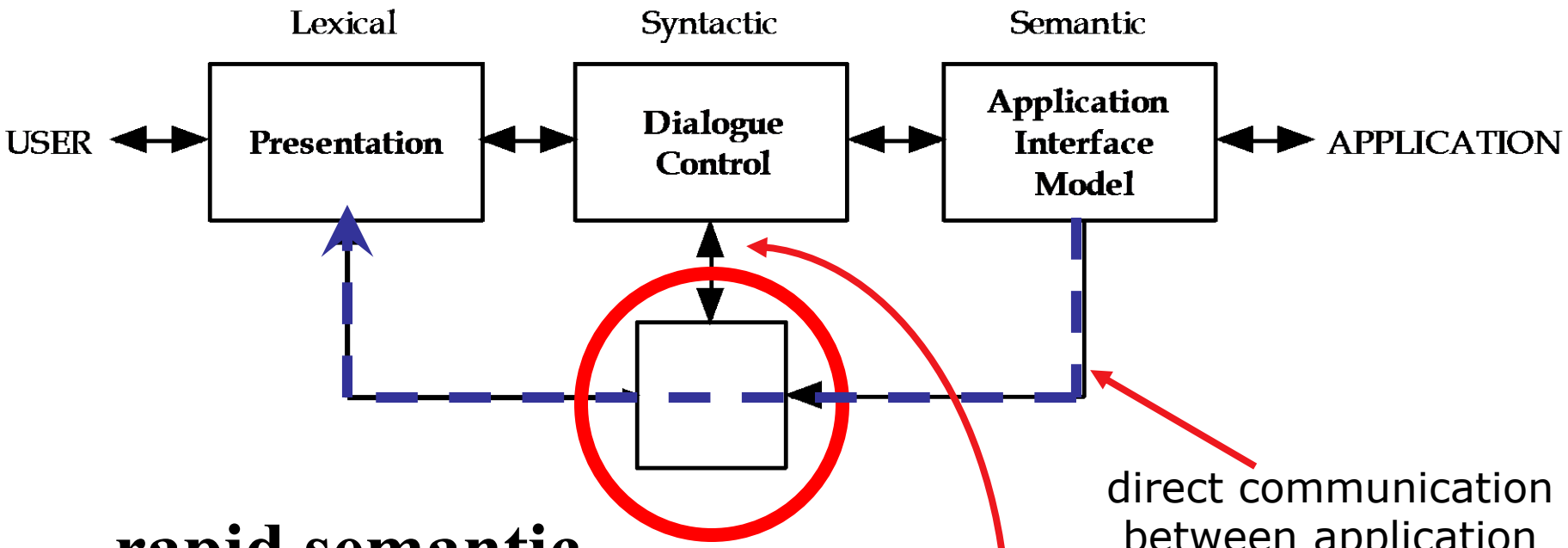
semantic feedback

- different kinds of feedback:
  - lexical – movement of mouse
  - syntactic – menu highlights
  - semantic – sum of numbers changes
- semantic feedback often slower
  - use rapid lexical/syntactic feedback
- but may need rapid semantic feedback
  - freehand drawing
  - highlight trash can or folder when file dragged

what's this?


Lexical
Syntactic
Semantic
USER
Presentation
Dialogue Control
Application Interface Model
APPLICATION

the bypass/switch

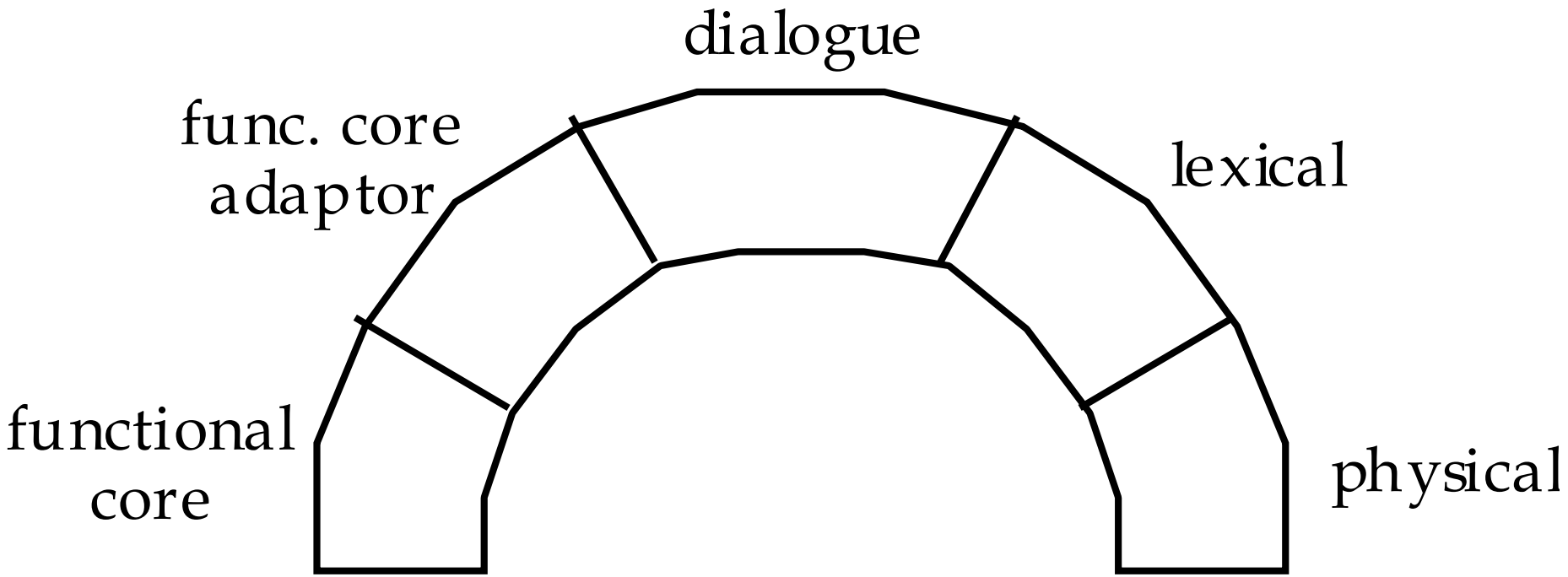


rapid semantic feedback

direct communication between application and presentation

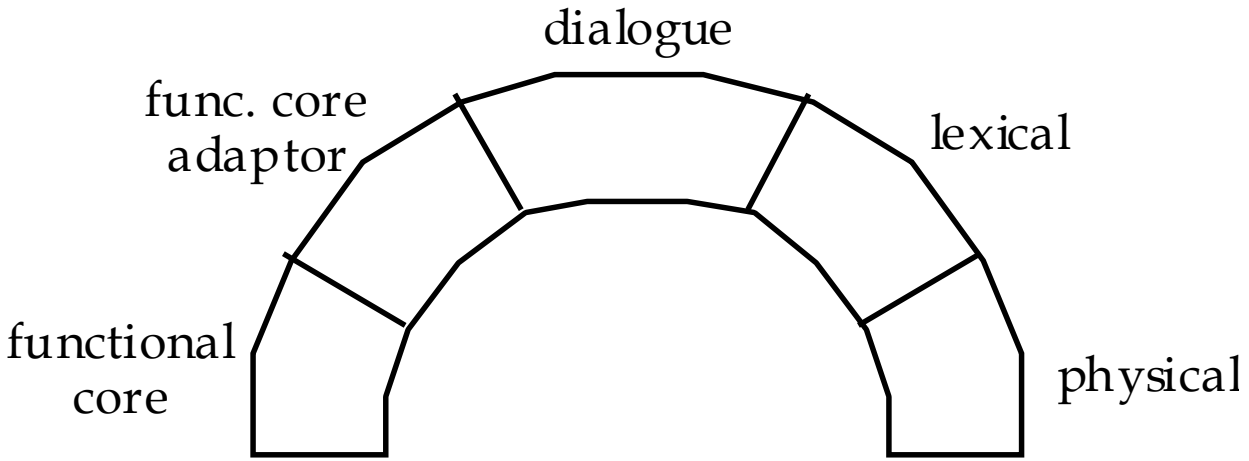
but regulated by dialogue control

more layers!


dialogue
func. core adaptor
lexical
functional core
physical

Arch/Slinky

- more layers! – distinguishes lexical/physical
- like a 'slinky' spring different layers may be thicker (more important) in different systems
- or in different components

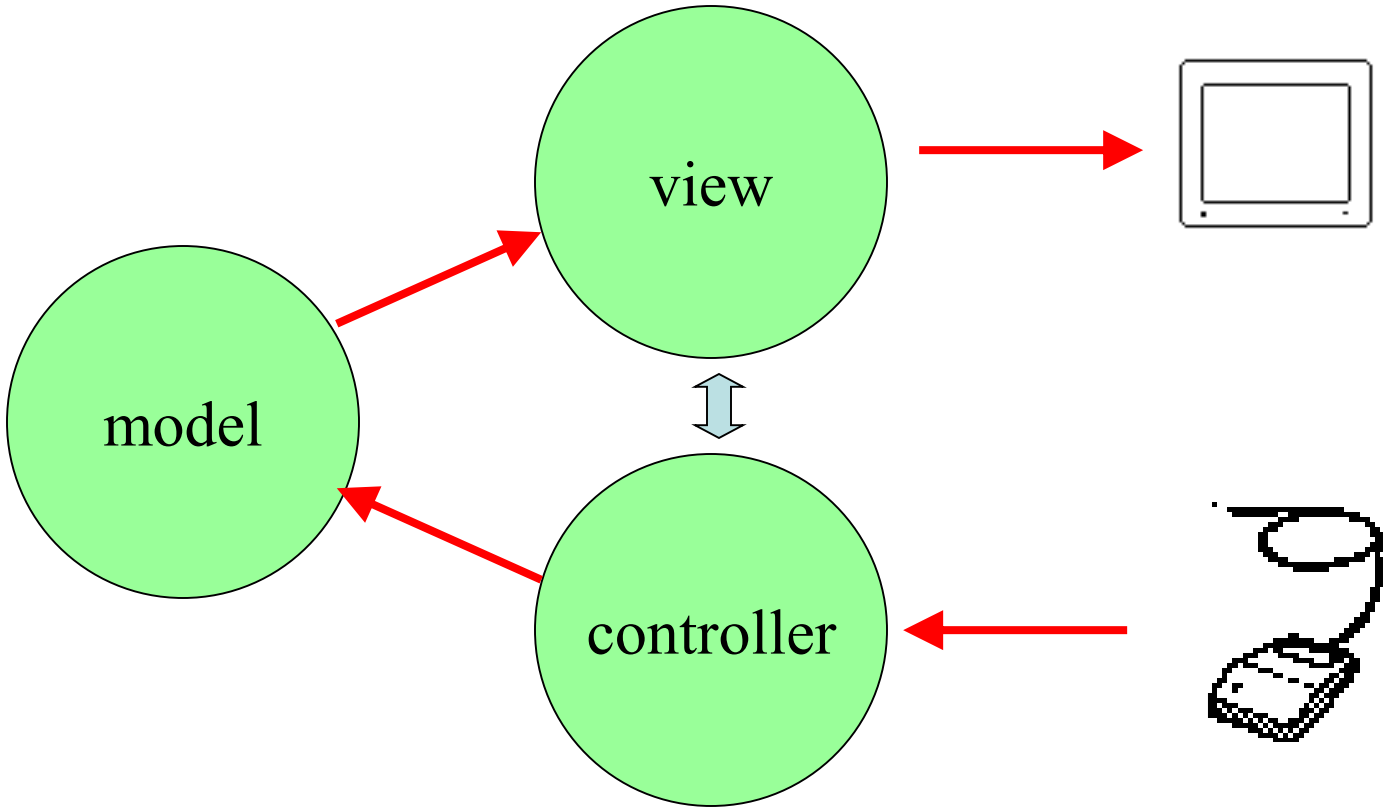


monolithic vs. components

- Seeheim has big components
- often easier to use smaller ones
  - esp. if using object-oriented toolkits
- Smalltalk used MVC – model–view–controller
  - model – internal logical state of component
  - view – how it is rendered on screen
  - controller – processes user input

MVC

model - view - controller



MVC issues

- MVC is largely pipeline model:

  input → control → model → view → output

- but in graphical interface
  - input only has meaning in relation to output

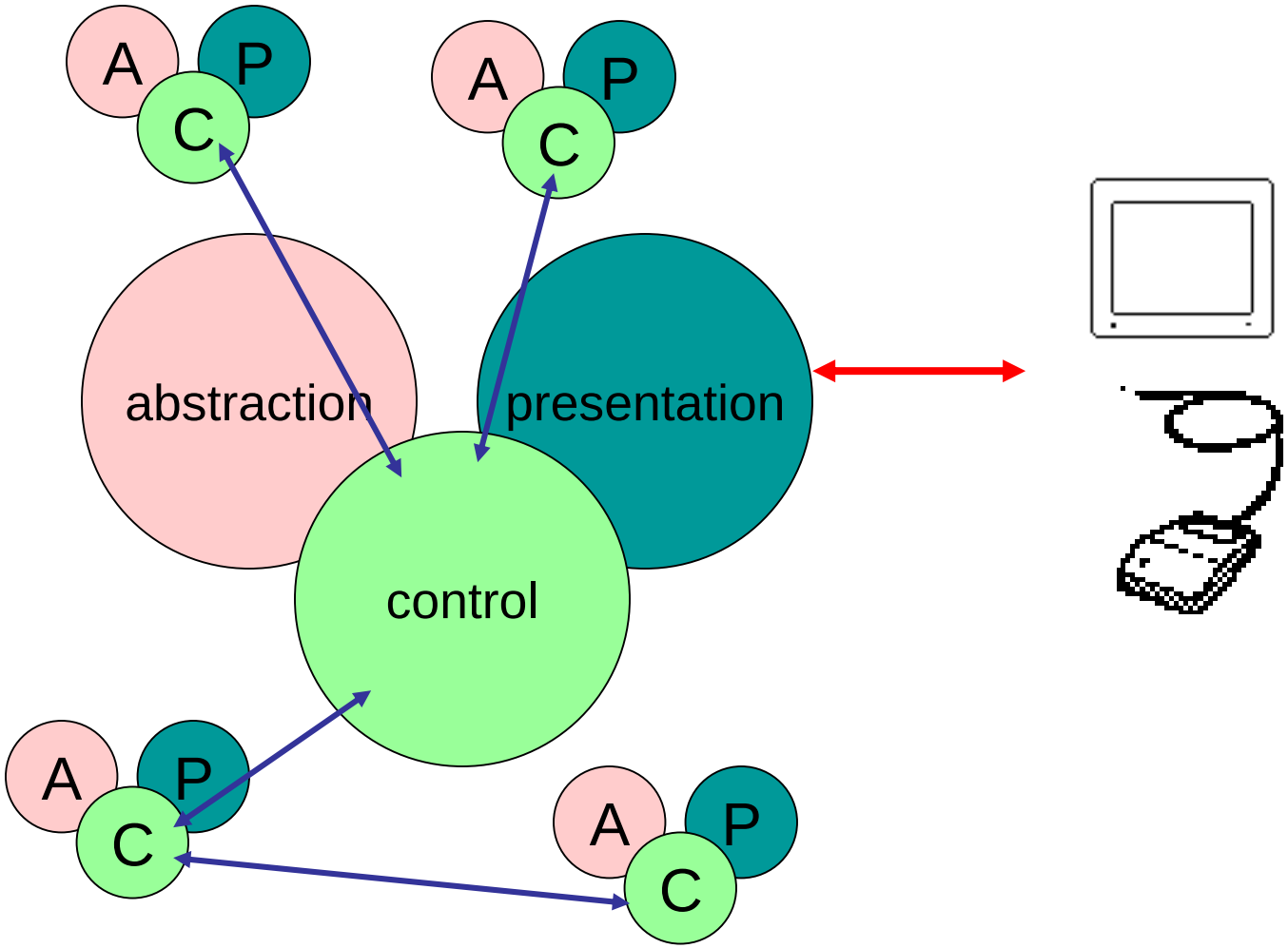
  e.g. mouse click
  - need to know *what* was clicked
  - controller has to decide what to do with click
  - but view knows what is shown where!
- in practice controller 'talks' to view
  - separation not complete

PAC model

- PAC model closer to Seeheim
  - abstraction – logical state of component
  - presentation – manages input and output
  - control – mediates between them
- manages hierarchy and multiple views
  - control part of PAC objects communicate
- PAC cleaner in many ways …
  but MVC used more in practice
  (e.g. Java Swing)

PAC

presentation - abstraction - control



Implementation of UIMS

- Techniques for dialogue controller
  - menu networks
  - grammar notations
  - declarative languages
  - graphical specification
  - state transition diagrams
  - event languages
  - constraints

  – for most of these see chapter 16

- N.B. constraints
  – instead of what *happens* say what should be *true*
  – used in groupware as well as single user interfaces
    (ALV - abstraction–link–view)

see chapter 16 for more details on several of these

graphical specification

- what it is
  - draw components on screen
  - set actions with script or links to program
- in use
  - with raw programming most popular technique
  - e.g. Visual Basic, Dreamweaver, Flash
- local vs. global
  - hard to 'see' the paths through system
  - focus on what can be seen on one screen

The drift of dialogue control

- internal control
  (e.g., read-evaluation loop)
- external control
  (independent of application semantics or presentation)
- presentation control
  (e.g., graphical specification)

Summary

Levels of programming support tools

- Windowing systems
  - device independence
  - multiple tasks
- Paradigms for programming the application
  - read-evaluation loop
  - notification-based
- Toolkits
  - programming interaction objects
- UIMS
  - conceptual architectures for separation
  - techniques for expressing dialogue