Human Computer Interaction (INSY4112)

CHAPTER 6

Design Rules and Implementation support

HCI IN THE SOFTWAREPROCESS

- **Design Rules**
  - The Principles to support usability,
  - Standards,
  - Guidelines,
  - Golden rules and heuristics,
  - HCI patterns
- **Implementation Support**
  - Elements of windowing systems,
  - Programming the application,
  - User interface management systems

Objectives

- To Produce a low-fidelity prototype for an interactive product based upon a simple list of interaction design principles and design rules.
- To identify Implementation supports in the development of user centered computer based information systems

Design Rules

- **Introduction to Design Rules and Implementation support**

- Designing for maximum usability is the goal of interactive systems design.
- Abstract principles offer a way of understanding usability in a more general sense, especially if we can express them within some coherent catalog.
- Design rules in the form of standards and guidelines provide direction for design, in both general and more concrete terms, in order to enhance the interactive properties of the system.

Design Rules

- **Introduction to Design Rules and Implementation support**
- The essential characteristics of good design are often summarized through 'golden rules' or heuristics.
- Design patterns provide a potentially generative approach to capturing and reusing design knowledge.

Design Rules

Design Rules

- Designing for maximum usability
    - The goal of interaction design
- Principles of usability
    - General understanding
- Standards and guidelines
    - Direction for design
- Design patterns
    - Capture and reuse design knowledge

Design Rules

➢ Types of Design Rules

I. Principles

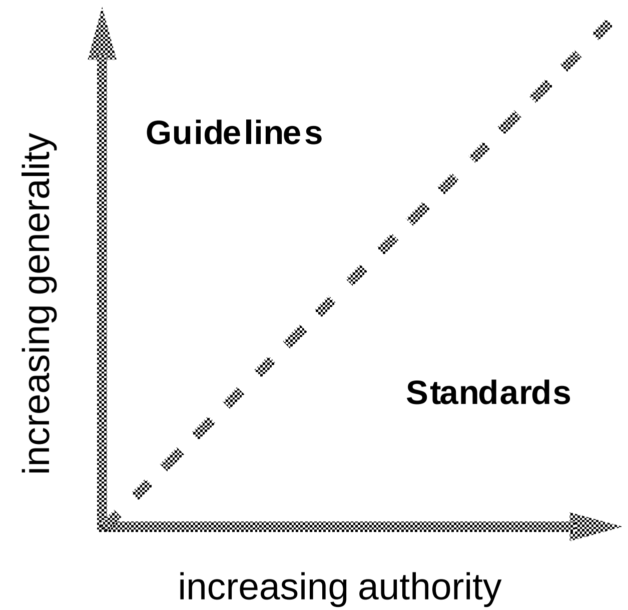
- abstract design rules
- low authority
- high generality

II. Standards

- specific design rules
- high authority
- limited application

III. Guidelines

- lower authority
- more general application



Design Rules

- Principles to support usability

1. Learnability
   - ✓the ease with which new users can begin effective interaction and achieve maximal performance
2. Flexibility
   - ✓the multiplicity of ways the user and system exchange information
3. Robustness
   - ✓the level of support provided the user in determining successful achievement and assessment of goal-directed behaviour

Design Rules

- Principles of learnability

1. Predictability
   - determining effect of future actions based on past interaction history
   - operation visibility
2. Synthesizability
   - assessing the effect of past actions
   - immediate vs. eventual honesty

Design Rules

➢Principles of learnability (ctd)

3. Familiarity
   – how prior knowledge applies to new system
   – guessability; affordance
4. Generalizability
   – extending specific interaction knowledge to new situations
5. Consistency
   – likeness in input/output behaviour arising from similar situations or task objectives

Design Rules

➢Principles of flexibility

1. Dialogue initiative
   - freedom from system imposed constraints on input dialogue
   - system vs. user pre-emptiveness
2. Multithreading
   - ability of system to support user interaction for more than one task at a time
   - concurrent vs. interleaving; multimodality
3. Task migratability
   - passing responsibility for task execution between user and system

- Principles of flexibility (ctd)

4. Substitutive
   - allowing equivalent values of input and output to be substituted for each other
   - representation multiplicity; equal opportunity
5. Customizability
   - modifiability of the user interface by user (adaptability) or system (adaptivity)

Design Rules

Principles of robustness

1. Absorbability
   - ability of user to evaluate the internal state of the system from its perceivable representation
   - browsability; defaults; reachability; persistence; operation visibility
2. Recoverability
   - ability of user to take corrective action once an error has been recognized
   - reachability; forward/backward recovery; commensurate effort

Design Rules

- Principles of robustness (ctd)

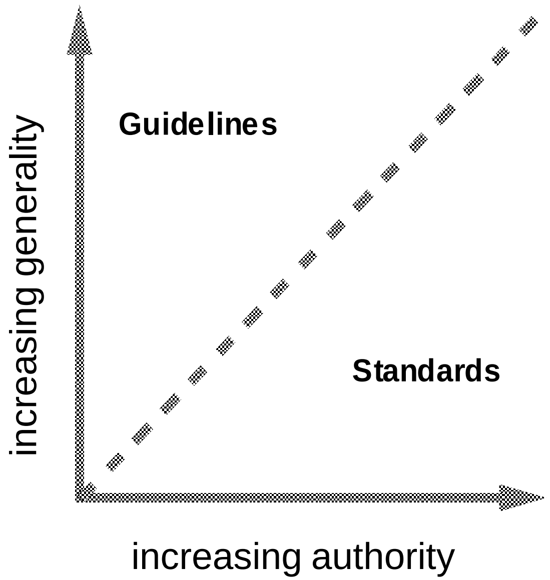
3. Responsiveness
   - how the user perceives the rate of communication with the system
   - Stability
4. Task conformance
   - degree to which system services support all of the user's tasks
   - task completeness; task adequacy

Design Rules

➢Using design rules

❖ **Design rules**

- Suggest how to increase usability
- Differ in generality and authority



Design Rules

II. Standards

- set by national or international bodies to ensure compliance by a large community of designers standards require sound underlying theory and slowly changing technology
- hardware standards more common than software high authority and low level of detail
- ISO 9241 defines usability as effectiveness, efficiency and satisfaction with which users accomplish tasks

Design Rules

III. Guidelines

- More suggestive and general
- Many textbooks and reports full of guidelines
- Abstract guidelines (principles) applicable during early life cycle activities
- Detailed guidelines (style guides) applicable during later life cycle activities
- Understanding justification for guidelines aids in resolving conflicts

Design Rules

- Golden rules and heuristics

- “Broad brush” design rules
- Useful check list for good design
- Better design using these than using nothing!
- Different collections e.g.
  - Nielsen’s 10 Heuristics (see Chapter 9)
  - Shneiderman’s 8 Golden Rules
  - Norman’s 7 Principles

Design Rules

- Shneiderman's 8 Golden Rules

1. Strive for consistency
2. Enable frequent users to use shortcuts
3. Offer informative feedback
4. Design dialogs to yield closure
5. Offer error prevention and simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term memory load

Design Rules

➢Norman's 7 Principles

1. Use both knowledge in the world and knowledge in the head.
2. Simplify the structure of tasks.
3. Make things visible: bridge the gulfs of Execution and Evaluation.
4. Get the mappings right.
5. Exploit the power of constraints, both natural and artificial.
6. Design for error.
7. When all else fails, standardize.

Design Rules

HCI design patterns

- An approach to reusing knowledge about successful design solutions
- Originated in architecture
- A pattern is an invariant solution to a recurrent problem within a specific context.
- Examples
  - Light on Two Sides of Every Room (architecture)
  - Go back to a safe place (HCI)
- Patterns do not exist in isolation but are linked to other patterns in *languages* which enable complete designs to be generated

HCI design patterns (cont.)

- Characteristics of patterns
  - capture design practice not theory
  - capture the essential common properties of good examples of design
  - represent design knowledge at varying levels: social, organisational, conceptual, detailed
  - embody values and can express what is humane in interface design
  - are intuitive and readable and can therefore be used for communication between all stakeholders
  - a pattern language should be generative and assist in the development of complete designs.

Design Rules

Summary

- Principles for usability
  - Repeatable design for usability relies on maximizing benefit of one good design by abstracting out the general properties which can direct purposeful design
  - The success of designing for usability requires both creative insight (new paradigms) and purposeful principled practice
- Using design rules
  - Standards and guidelines to direct design activity

Implementation Support

Implementation Support

1. Programming tools
   - levels of services for programmers
2. Windowing systems
   - core support for separate and simultaneous user-system activity
3. Programming the application and control of dialogue
4. Interaction toolkits
   - bring programming closer to level of user perception
5. User interface management systems
   - controls relationship between presentation and functionality

Implementation Support

- Implementation Support

- **Programming tools** for interactive systems provide a means of effectively translating abstract designs and usability principles into an executable form. These tools provide different levels of services for the programmer.
- **Windowing systems** are a central environment for both the programmer and user of an interactive system, allowing a single workstation to support separate user–system threads of action simultaneously.

Implementation Support

Implementation Support

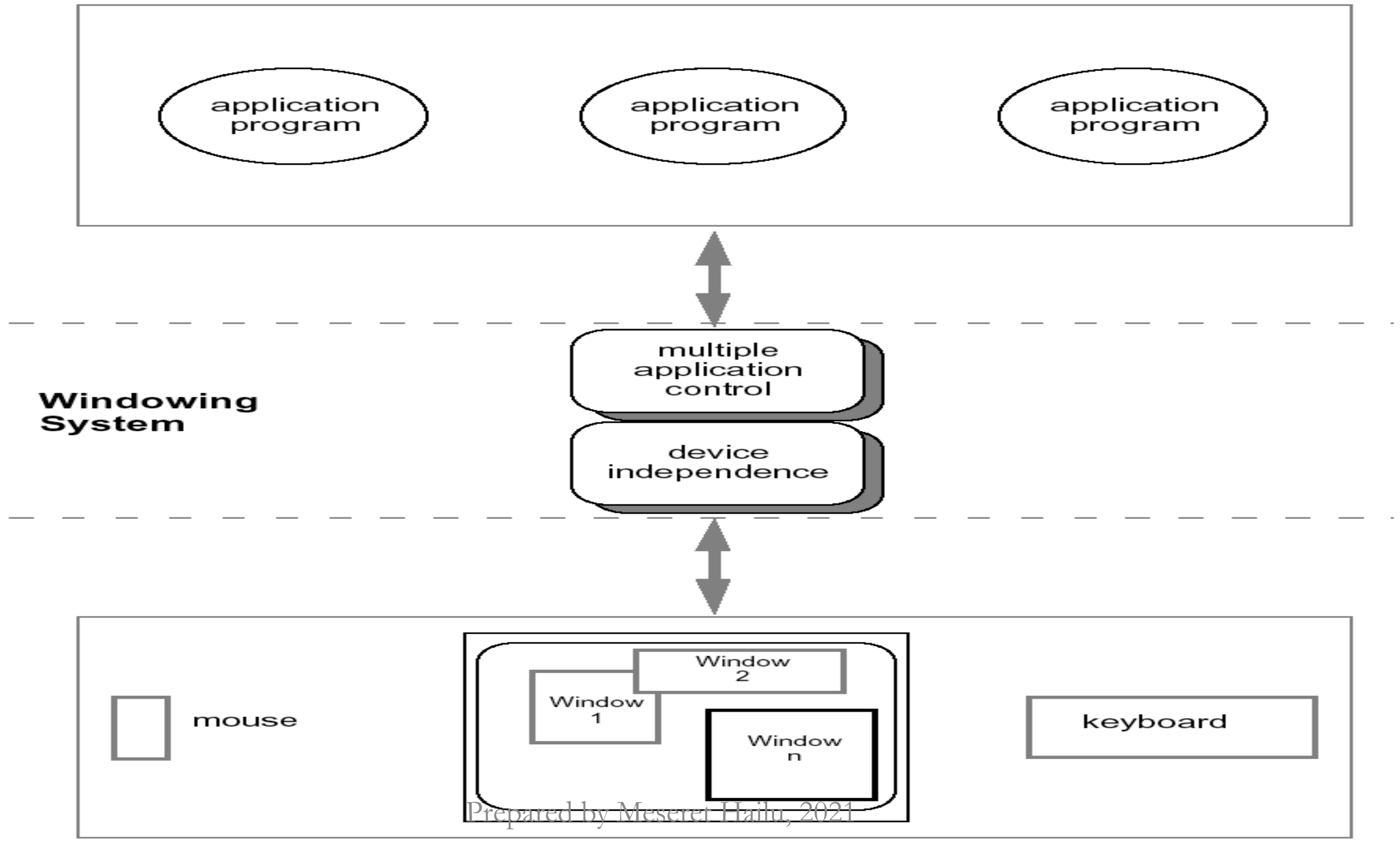
- **Interaction toolkits** abstract away from the physical separation of input and output devices, allowing the programmer to describe behaviors of objects at a level similar to how the user perceives them.
- **User interface management systems** are the final level of programming support tools, allowing the designer and programmer to control the relationship between the presentation objects of a toolkit with their functional semantics in the actual application.

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

Implementation Support

- roles of a windowing system



Implementation Support

Architectures of windowing systems

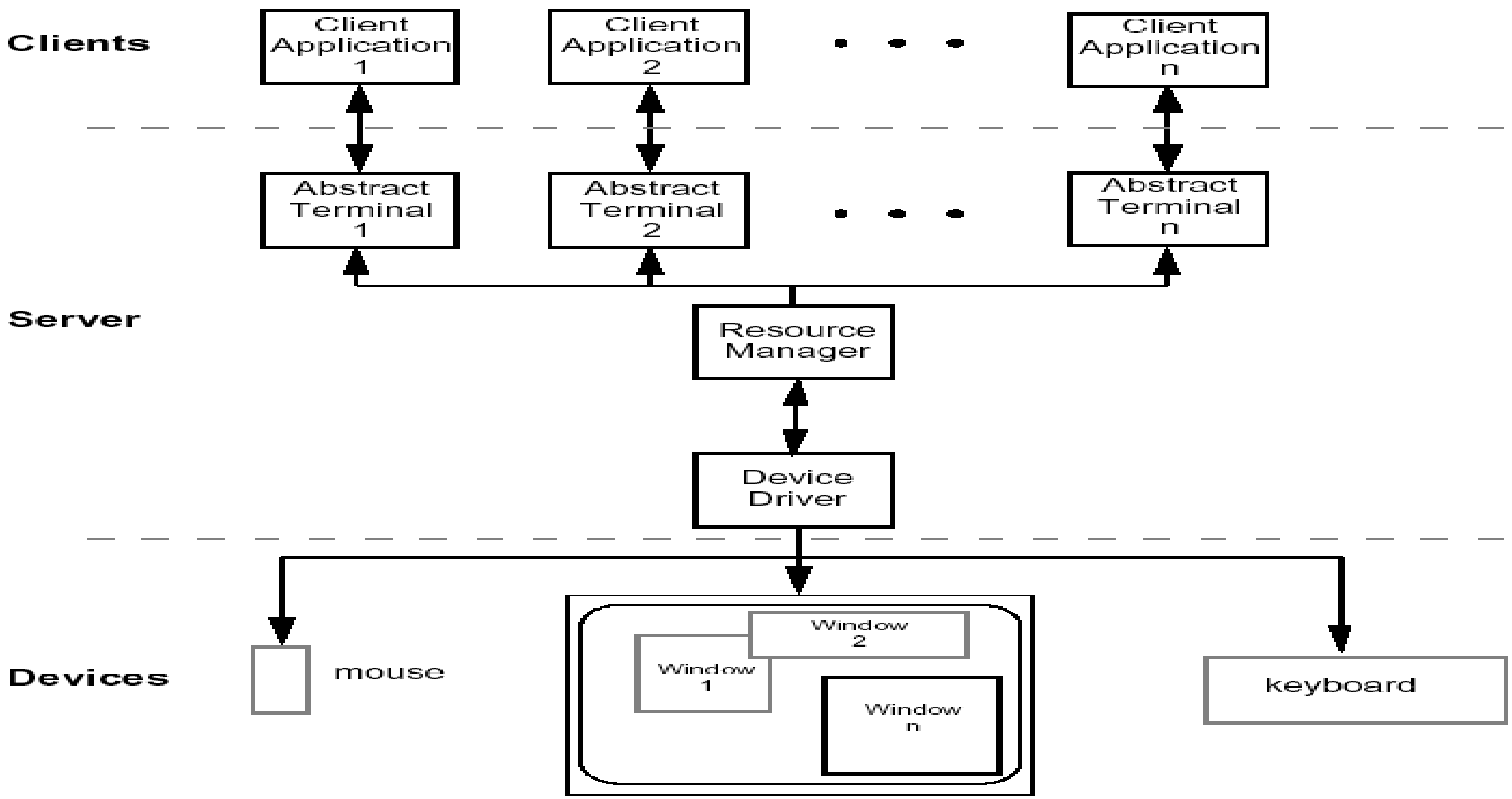
- Three possible software architectures
  - all assume device driver is separate
  - differ in how multiple application management is implemented

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

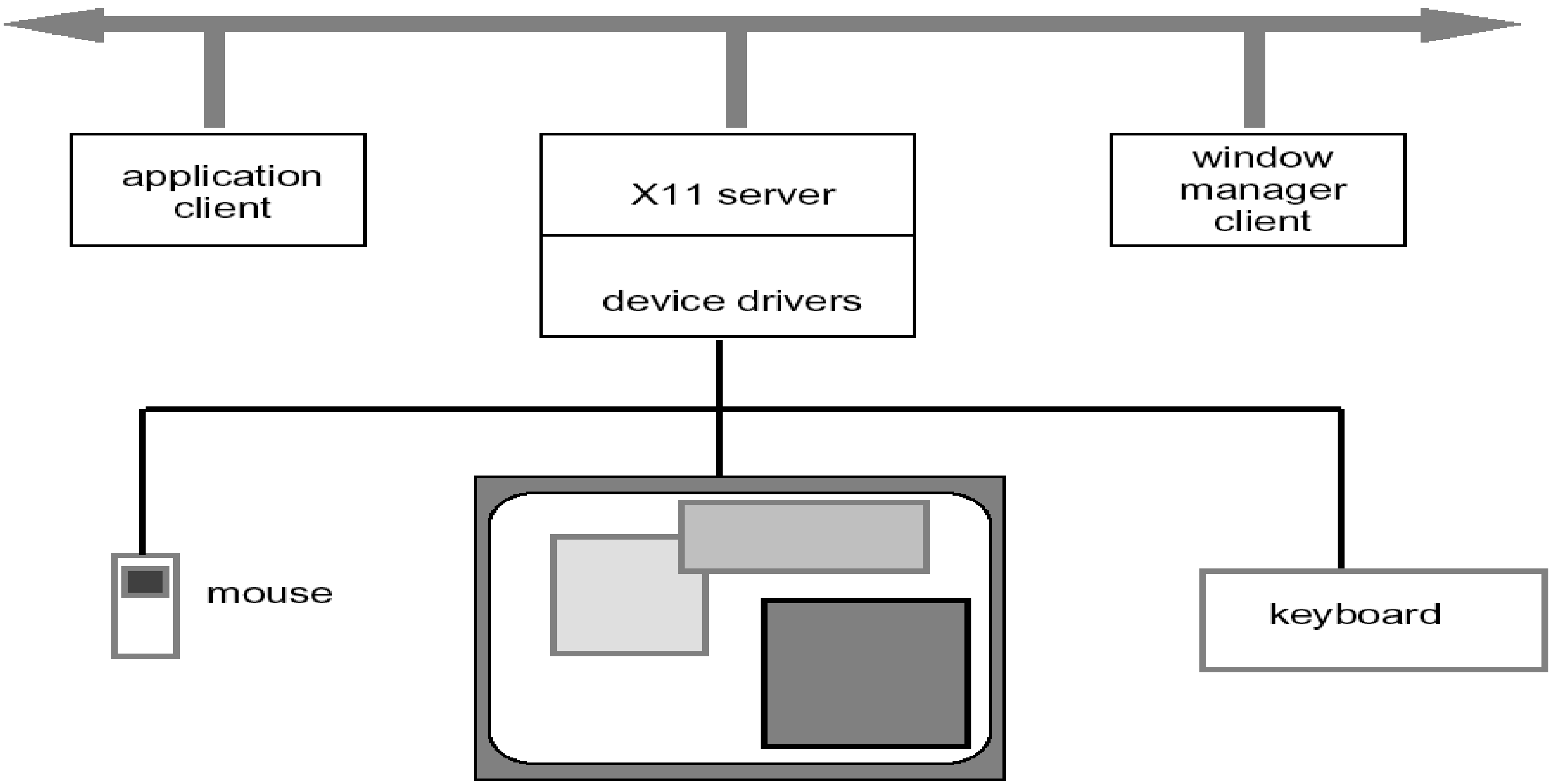
2. Management role within kernel of operating system
   - applications tied to operating system

3. Management role as separate application
   - maximum portability

- The client-server architecture



- X Windows architecture

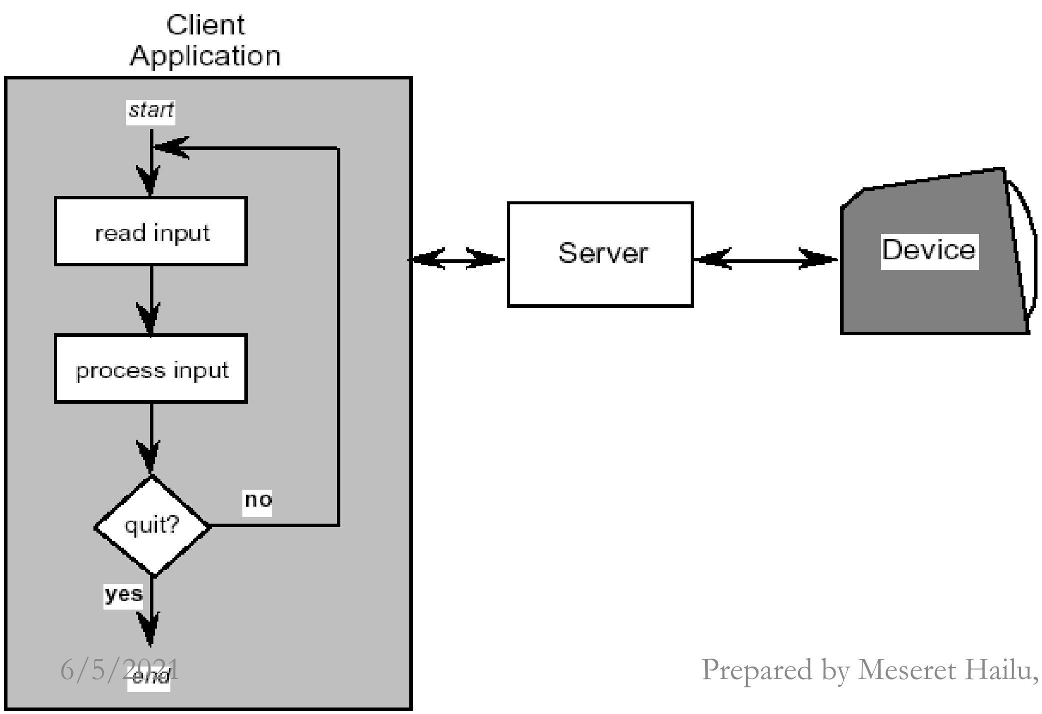


X Windows architecture (ctd)

- 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



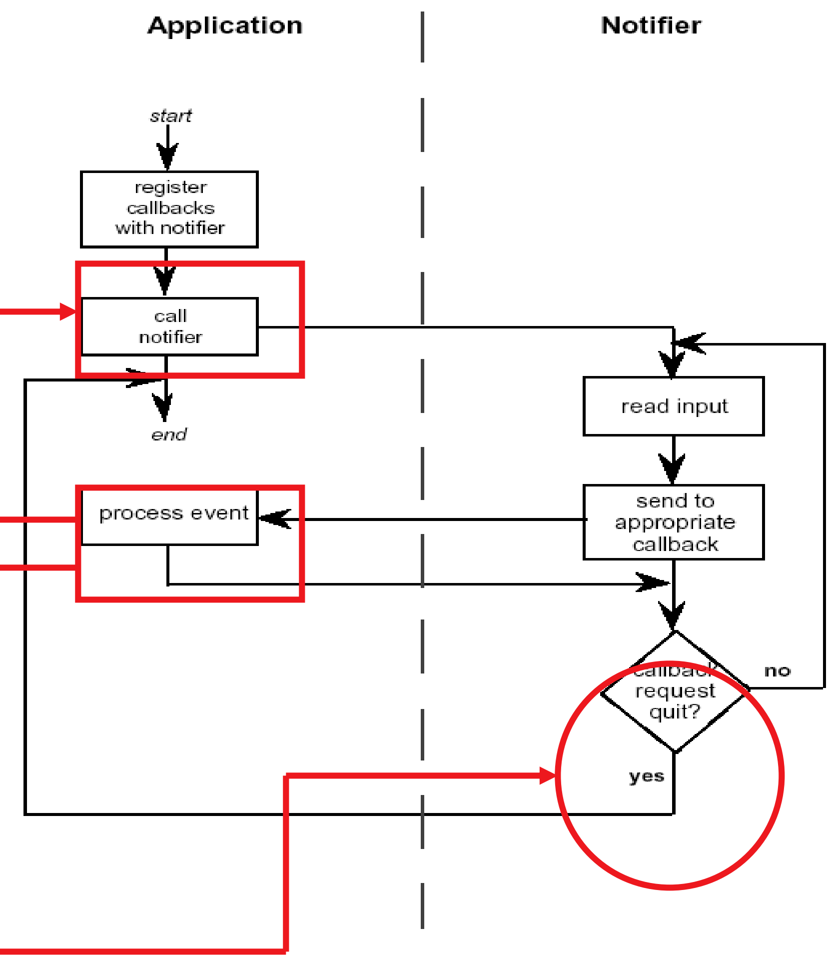

```
repeat
   read-event(myevent)
   case myevent.type
      type_1:
         do type_1 processing
      type_2:
         do type_2 processing
      ...
      type_n:
         do type_n processing
   end case
end repeat
```

➢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!

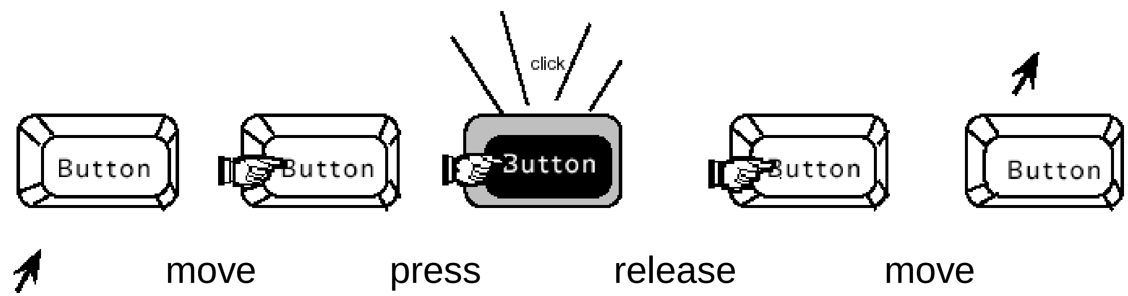
if you don't explicitly design it will just happen
implementation should not drive design

Implementation Support

➢ 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

Implementation Support

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 -– call-back 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

Implementation Support

- UIMS as conceptual architecture

- *Separation* between application semantics and presentation

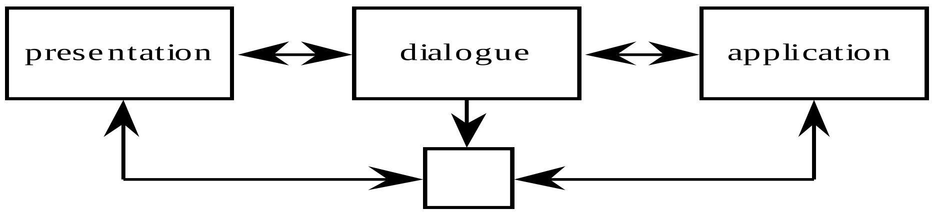
- Improves:
  - **portability –** runs on different systems
  - **reusability** – components reused cutting costs
  - **multiple interfaces** – accessing same functionality
  - **customizability** – by designer and user

Implementation Support

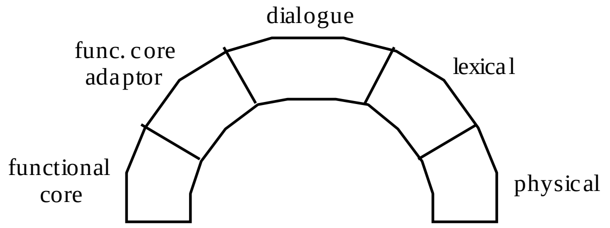
- UIMS tradition – interface layers / logical components

- linguistic: lexical/syntactic/semantic

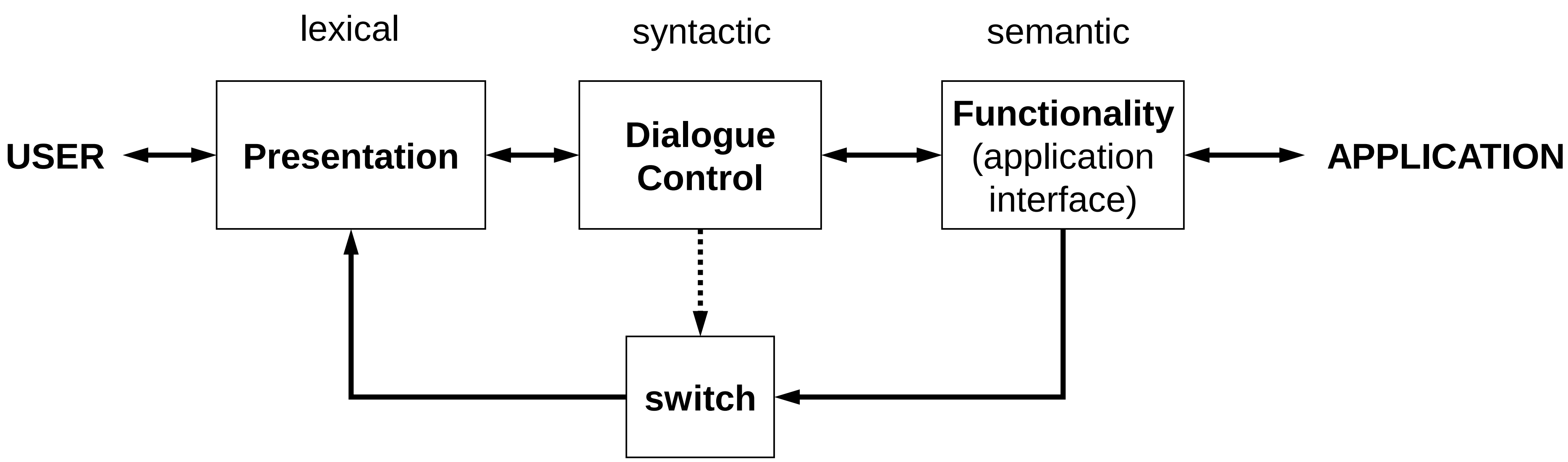
- Seeheim:



- Arch/Slinky



- Seeheim model

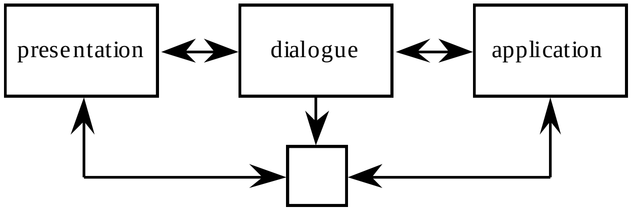


Implementation Support

- 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

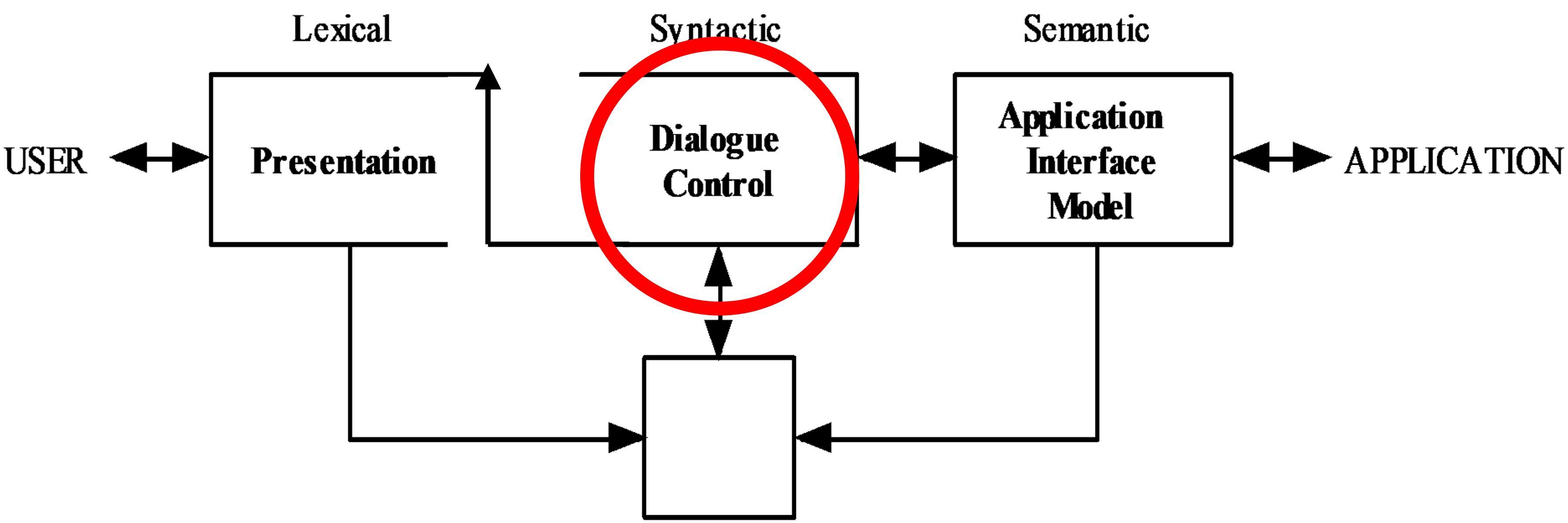


semantic feedback

- 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

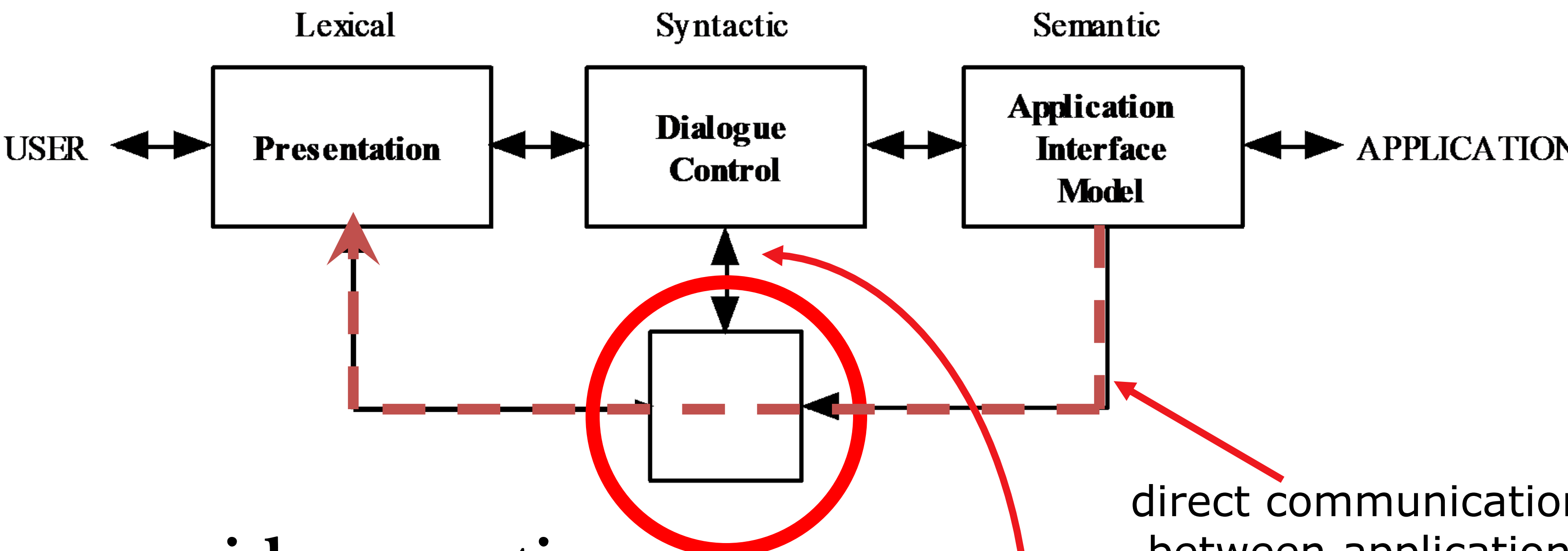
Implementation Support

- what's this?



Implementation Support

- the bypass/switch



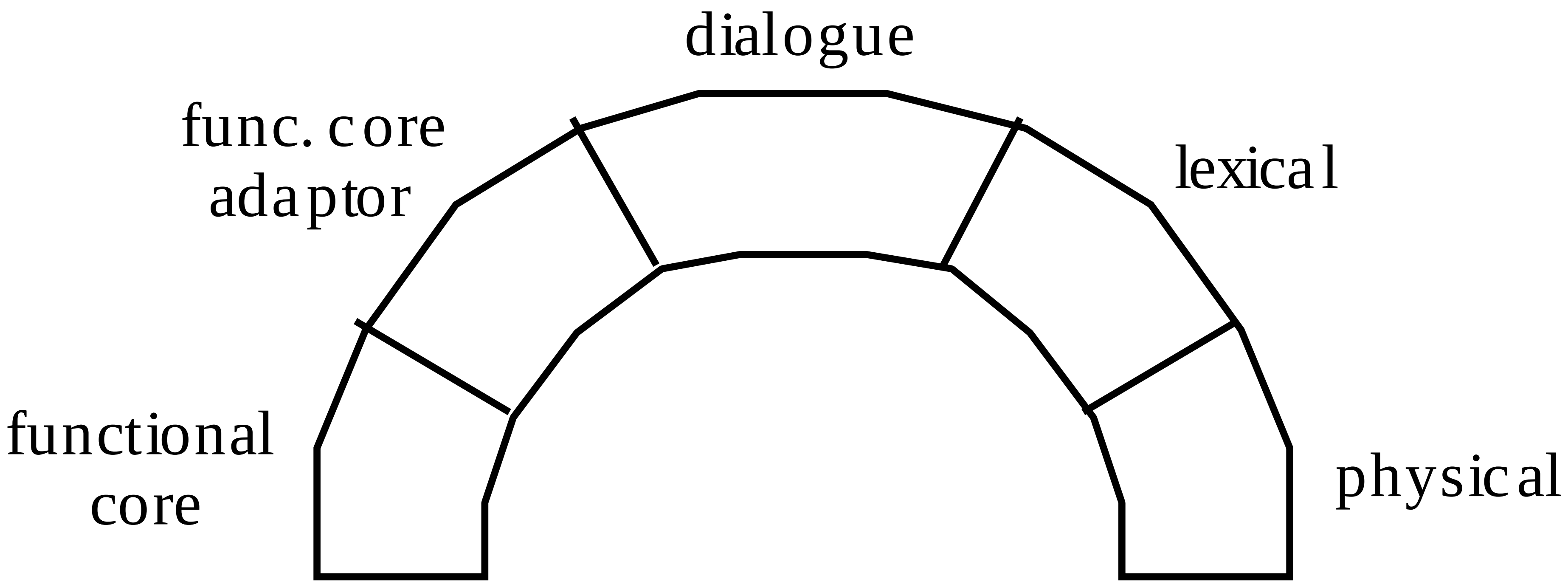
rapid semantic feedback

direct communication between application and presentation

but regulated by dialogue control

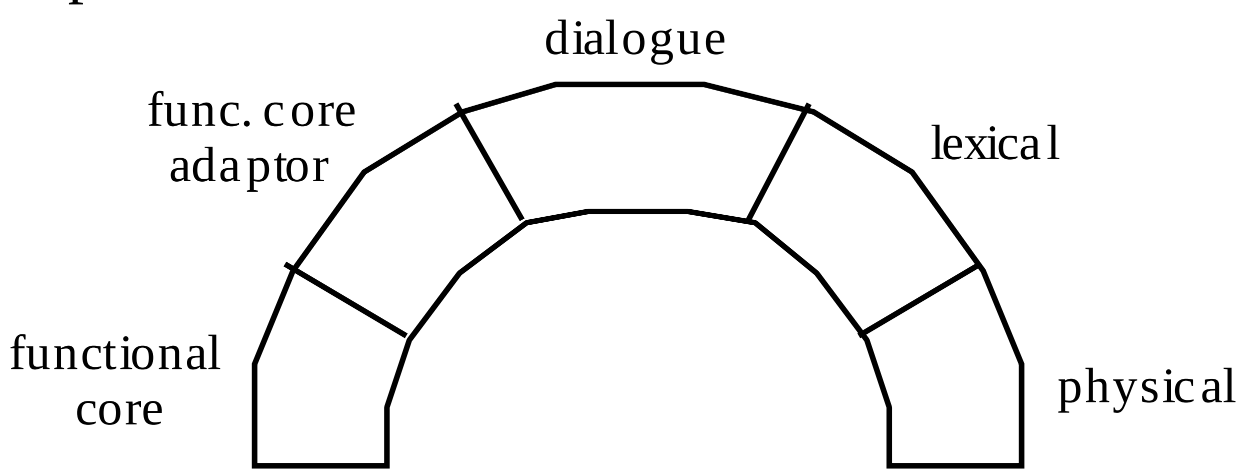
Implementation Support

- more layers!



Implementation Support

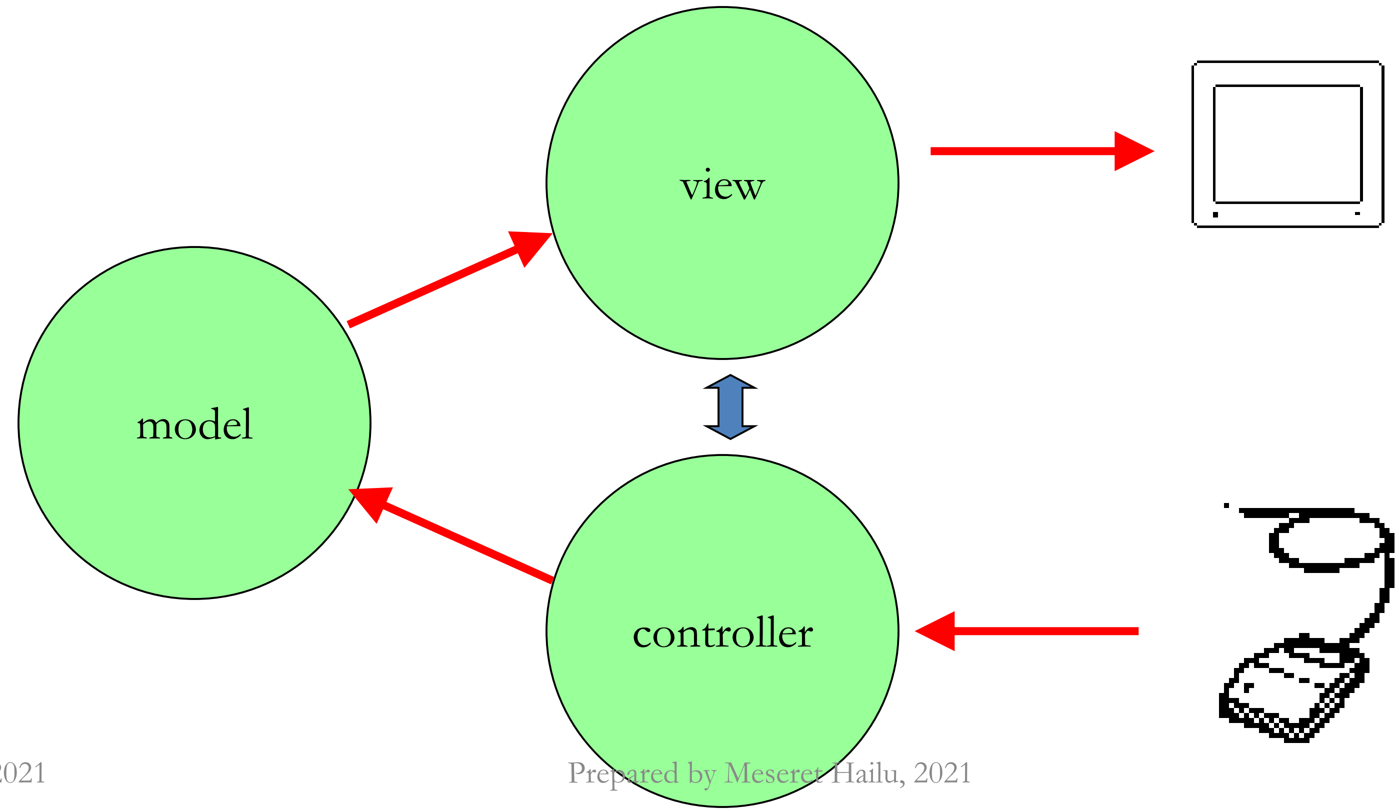
- Arch/Slinky
  - more layers! – distinguishes lexical/physical
  - like a 'slinky' spring different layers may be thicker (more important) in different systems
  - or in different components



- 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

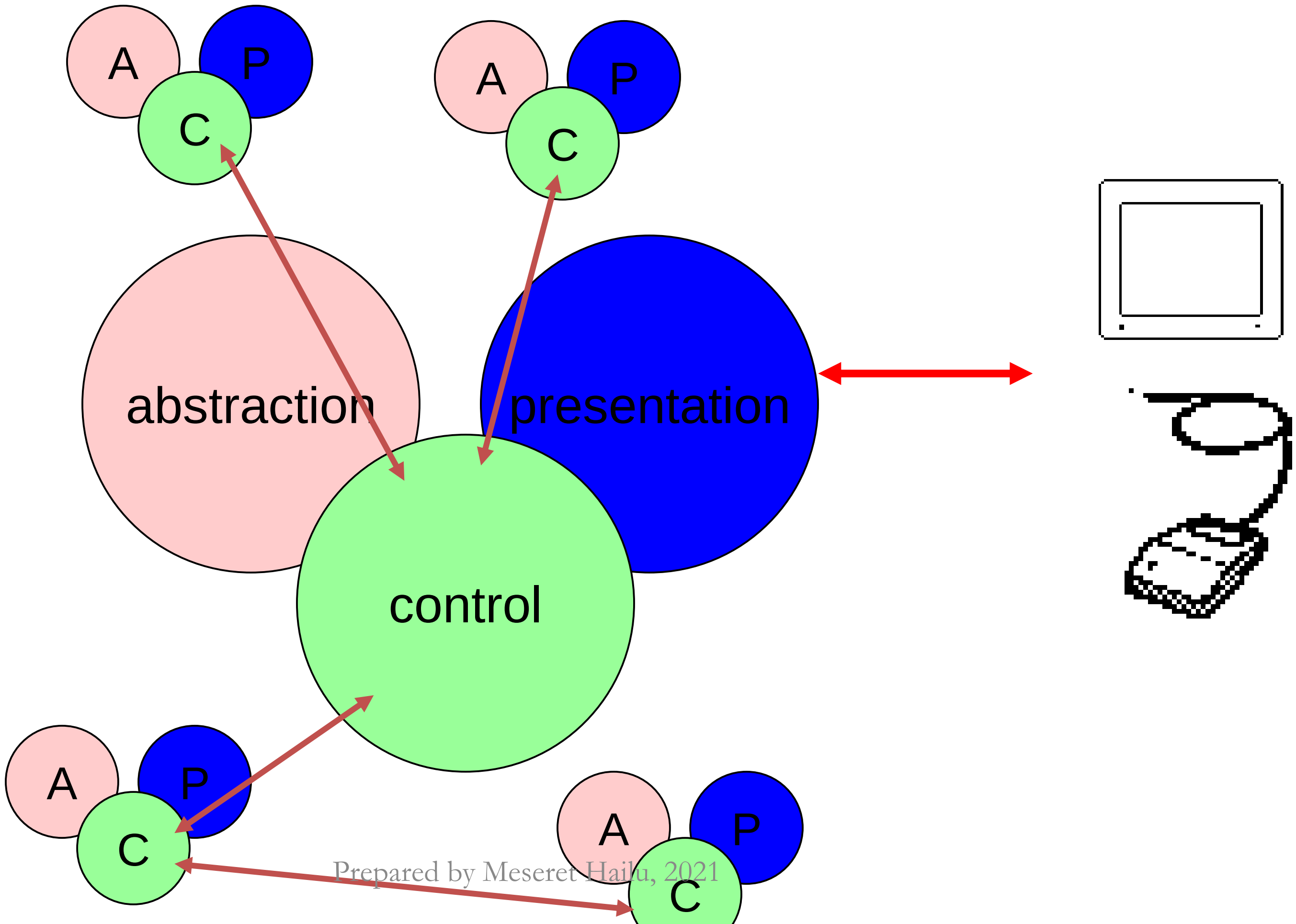
- PAC model closer to Seeheim
  - presentation – manages input and output
  - abstraction – logical state of component
  - control – mediates between them
- manages hierarchy and multiple views
  - control part of PAC objects communicate
- PAC cleaner in many ways …

  but MVC used more in practice

  (e.g. Java Swing)

Implementation Support

- PAC (Presentation - Abstraction - Control)



Implementation Support

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)

Implementation Support

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

Implementation Support

- 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

End of unit Six

- Seeheim model