Architectural Design:

An Overview

Week 6: Architectural Design, Part 1

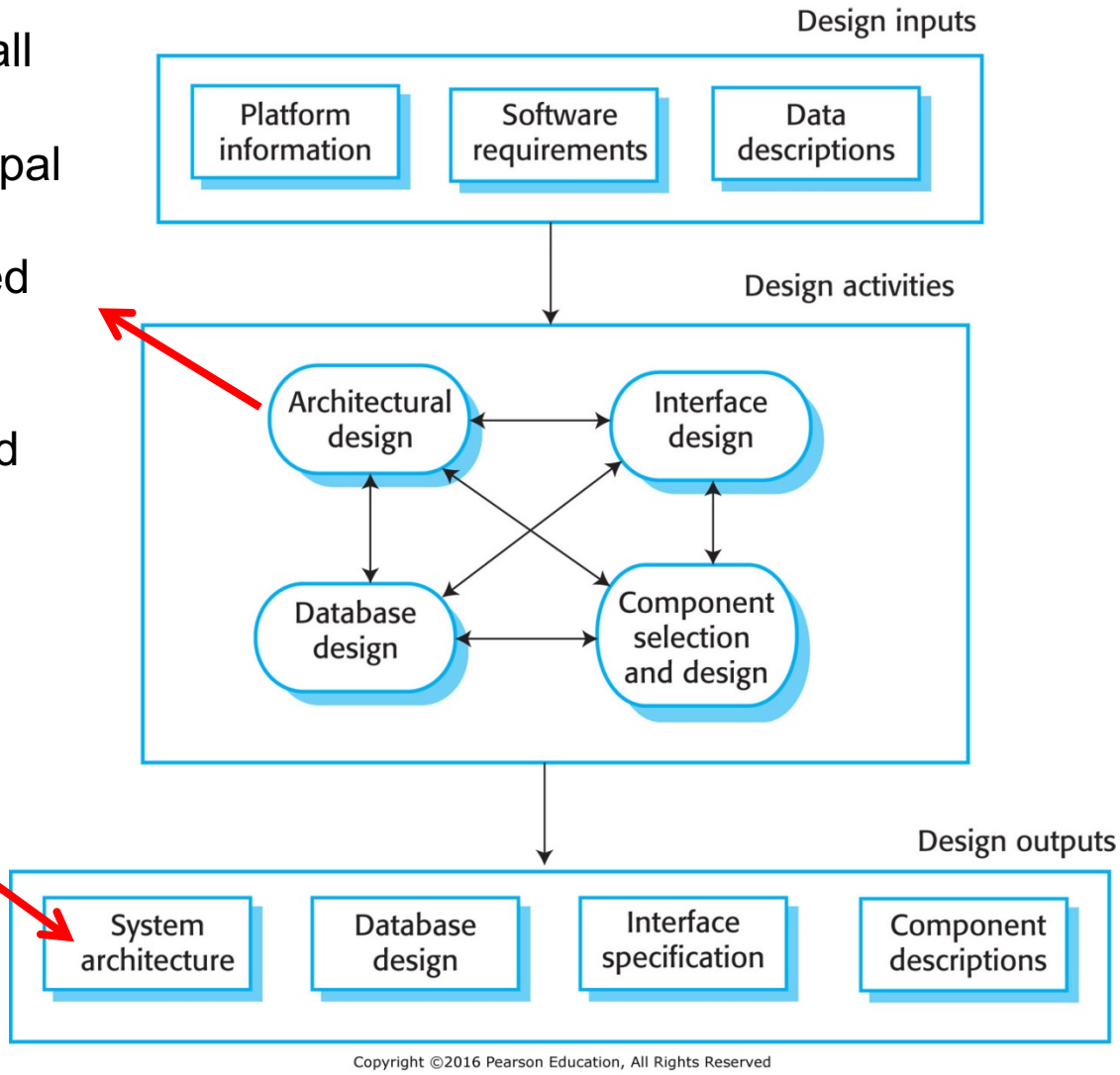
Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

A General Model of the Design Process

Identify the overall structure of the system, its principal components (sometimes called subsystems or modules), their relationships, and how they are distributed.

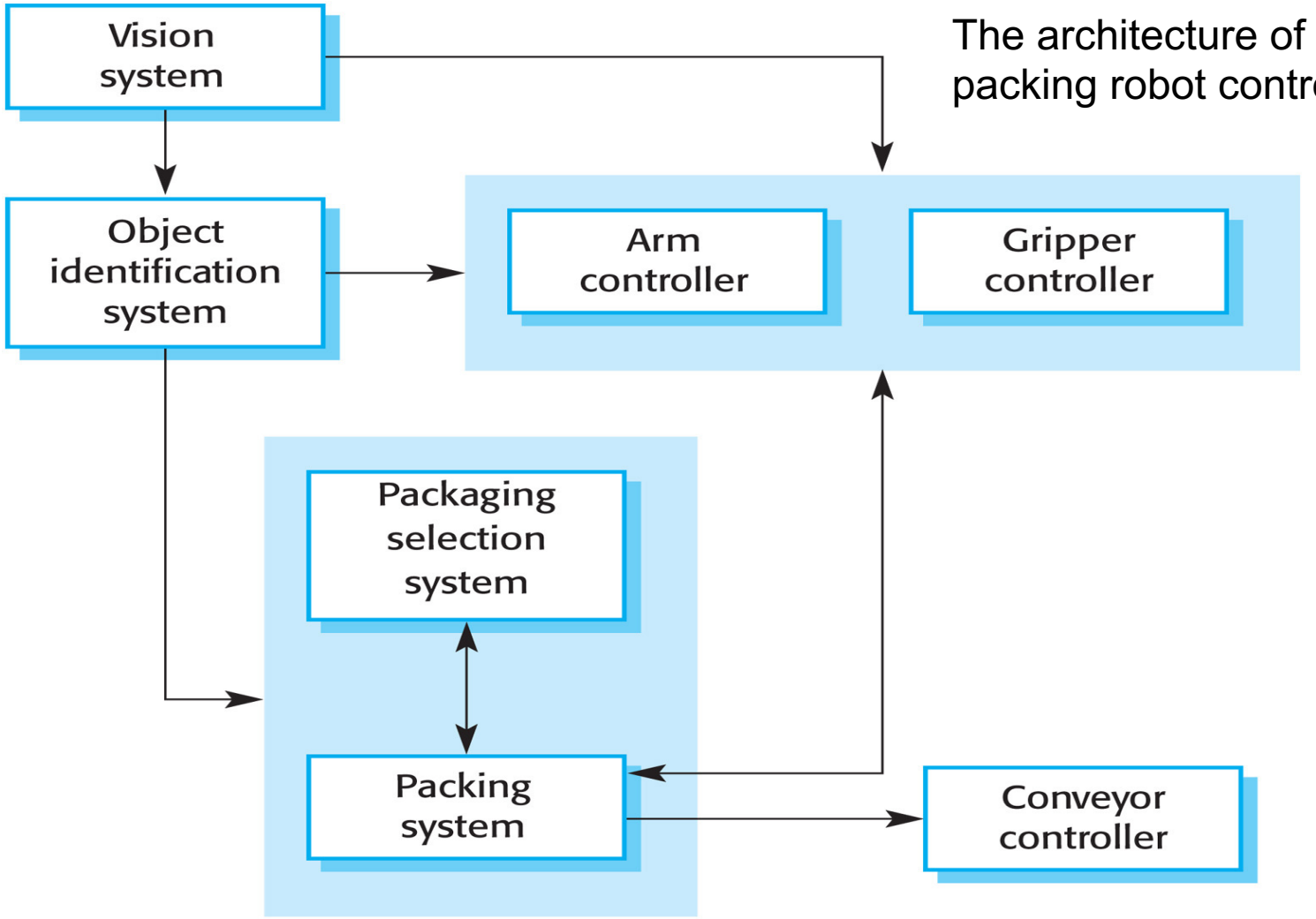
Its output



SRS Revisited

Section	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

Software Architecture: An Example



The architecture of a packing robot control system

Architectural Representations

- Simple, informal **block diagrams** showing entities and relationships are the most frequently used, even though they have been criticized for lacking semantics.
- They can be used:
  - As a way of facilitating discussion about the system design with system stakeholders and project planners, because they are not cluttered with detail
    - Stakeholders can relate to it and understand an abstract view of the system.
  - As a way of documenting an architecture that has been designed
    - The aim here is to produce a complete system model that shows the different components in a system, their interfaces, and their connections.

Architectural Abstraction

- You can also design software architectures at two levels of abstraction.
  - **Architecture in the small** is concerned with the architecture of individual programs.
    - At this level, we are concerned with the way that an individual program is decomposed into components.
  - **Architecture in the large** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components.
    - These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

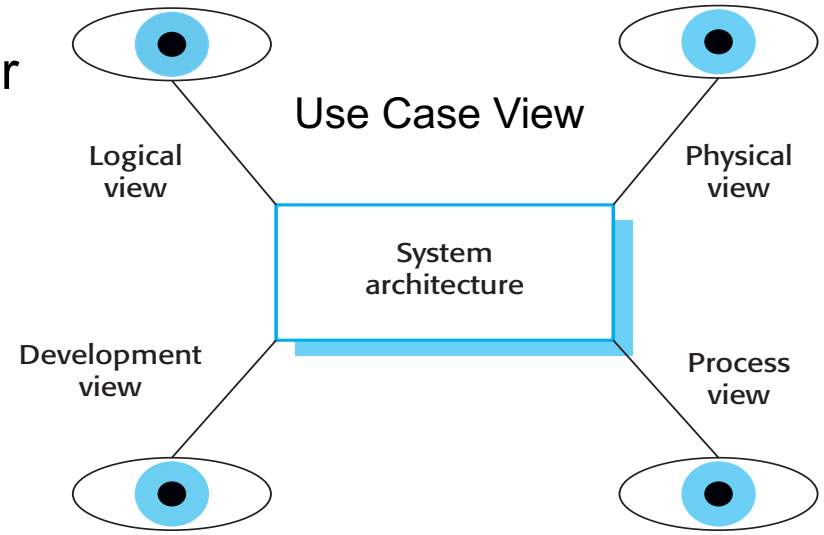
Four Perspectives of System Modeling

- An **external** perspective, where you model the context or environment of the system.
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural** perspective, where you model the organization of a system or the structure of the data that are processed by the system.
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events.

- These perspectives have much in common with Kruchten's 4 + 1 view of system architecture (Kruchten 1995).

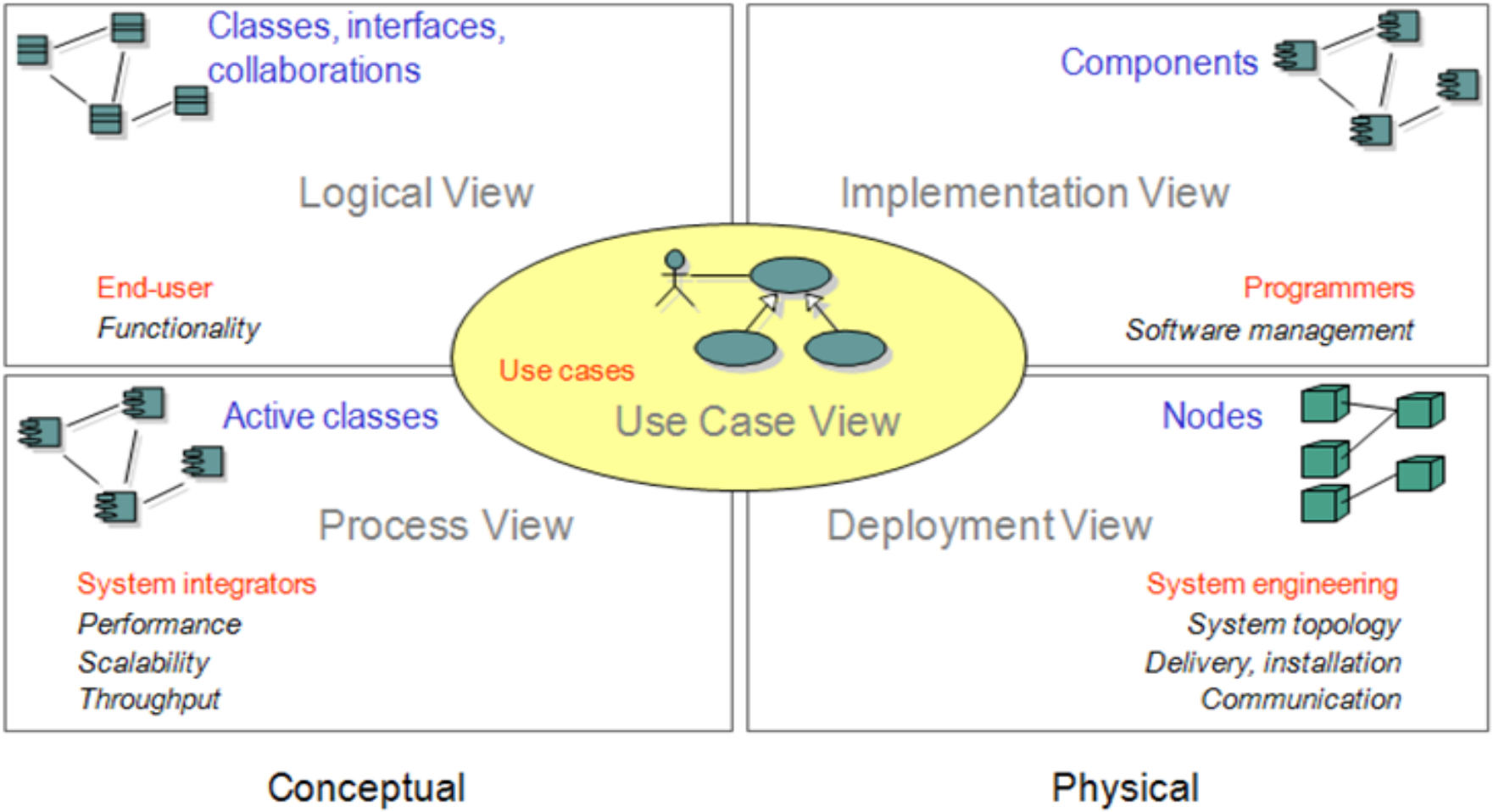
4 + 1 View Model of Software Architecture

- A **logical** view—shows the key abstractions in the system as objects or classes
- A **development** view— shows how the software is decomposed for development
- A **process** view— shows how, at run-time, the system is composed of interacting processes



- A **physical** view— shows the system hardware and how software components are distributed across the processors in the system
- Also includes **use cases** or scenarios (+1)

Five Architectural Views in UML



Importance of Software Architecture

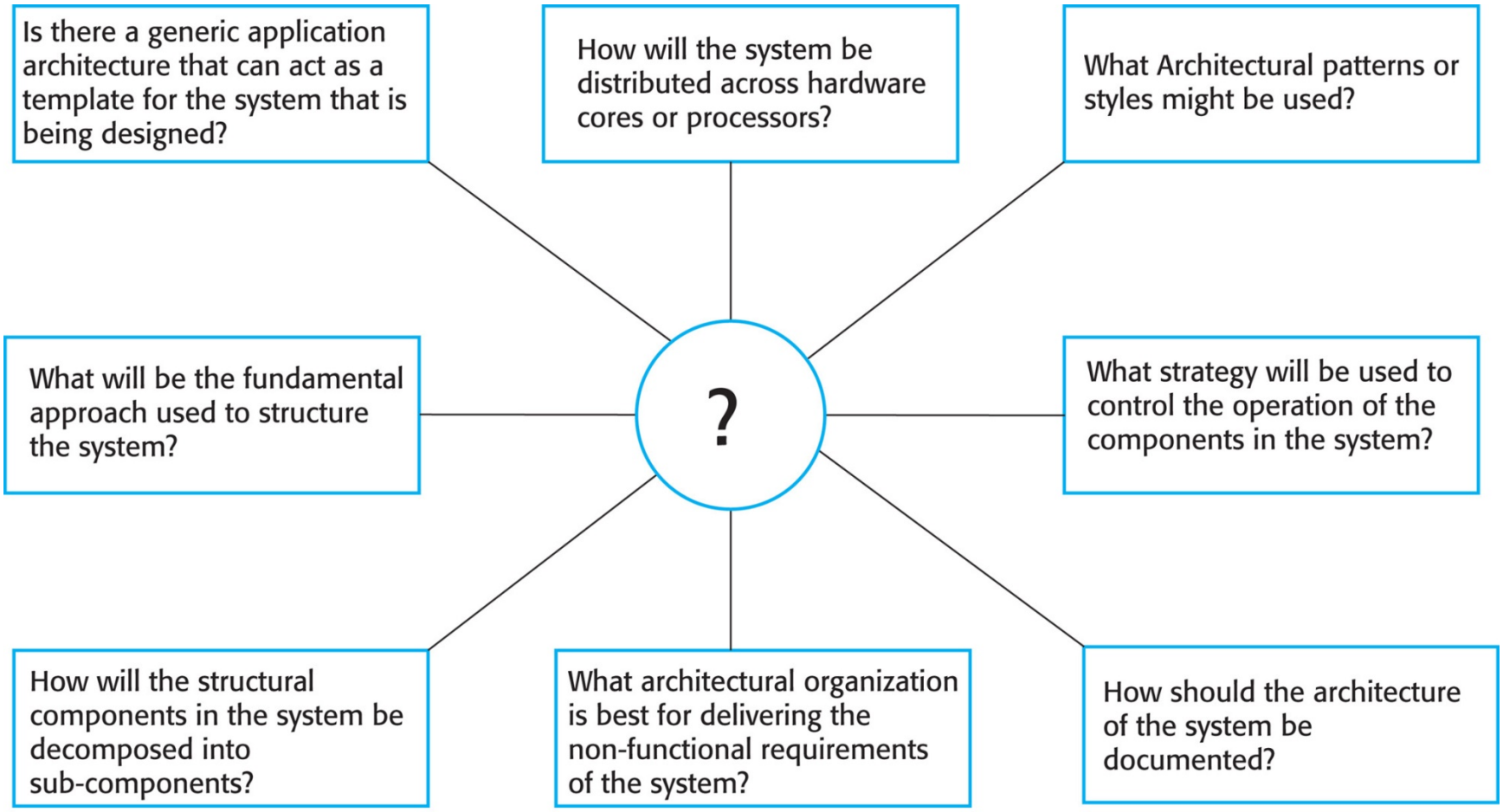
- Software architecture is important because it affects the performance, distributability, robustness, and maintainability of a system (Bosch 2000).
  - Individual components implement the functional system requirements.
  - The **nonfunctional requirements** depend on the system architecture, that is, the way in which these components are organized and communicate.
  - In many systems, nonfunctional requirements are also influenced by individual components, but there is no doubt that the architecture of the system is the dominant influence.

Bosch, J. 2000. *Design and Use of Software Architectures*. Addison-Wesley.

Architecture and System Characteristics

- Performance
  - Localize critical operations and minimize communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localize safety-critical features in a small number of subsystems.
- Robustness
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

Architectural Design Decisions



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Data-Processing Systems and Transaction-Processing Systems

Week 6: Architectural Design, Part 1

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

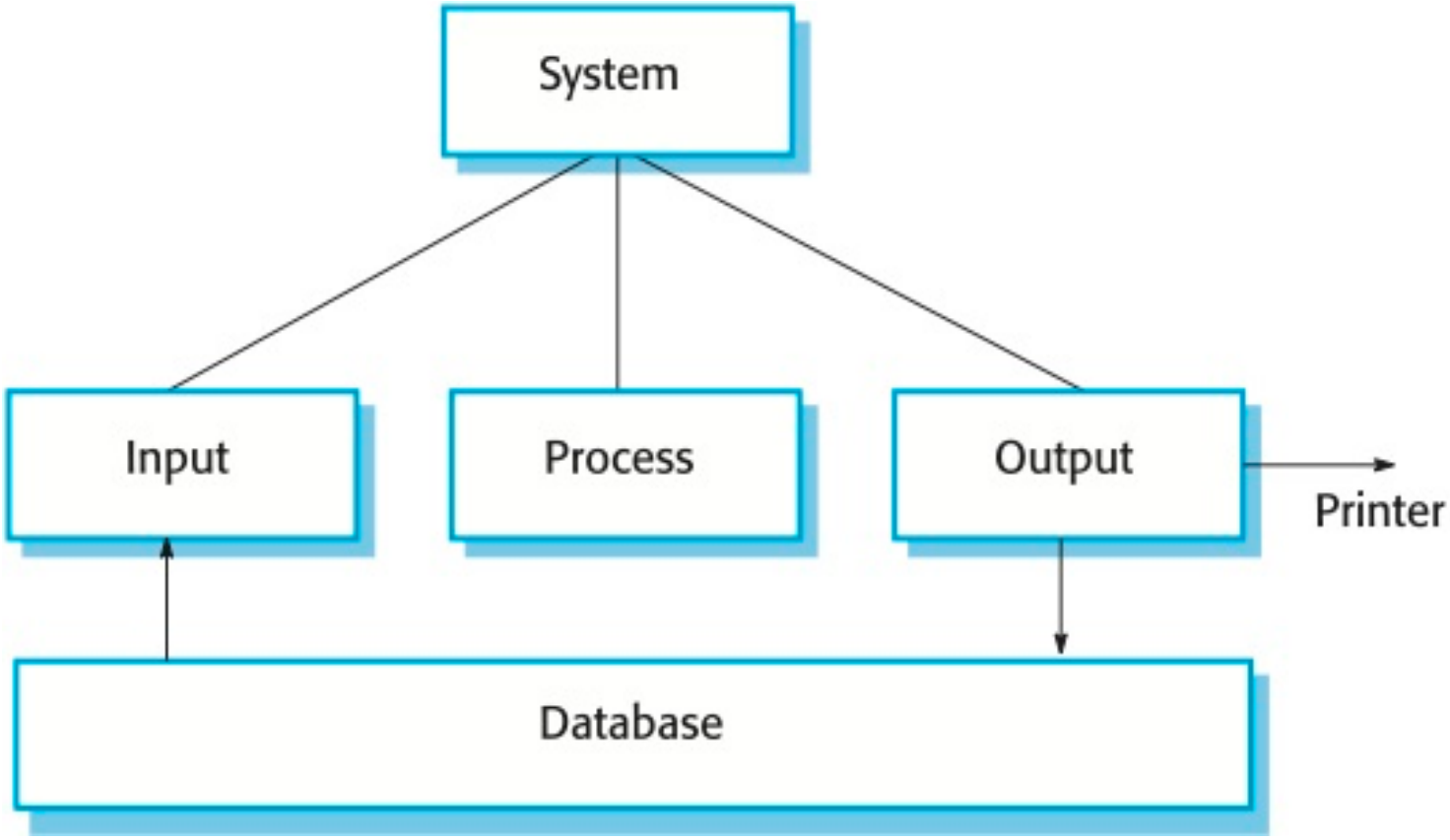
Four Common Types of Systems

- Architectural design is a creative process, and hence the process differs depending on the type of system being developed.
- Four common types of systems:
  - **Data-processing systems**
    - Data-driven applications that process data in **batches** without explicit user intervention during the processing
  - **Transaction-processing systems**
    - Data-centered applications that process user requests and update information in a system database
  - **Language-processing systems**
    - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system
  - **Event-processing systems**
    - Applications where system actions depend on interpreting events from the system's environment

Data-Processing Systems

- Data-processing systems are batch-processing systems where data are input and output in batches from a file or database rather than input from and output to a user terminal.
  - They select data from the input records
  - They take some actions, depending on the value of fields in the records.
  - They then write back the result of the computation to the database.
- Businesses rely on data-processing systems to support many aspects of their business such as paying salaries, calculating and printing invoices, maintaining accounts, issuing renewals for insurance policies, and so forth.

DPS: A Generic Architecture

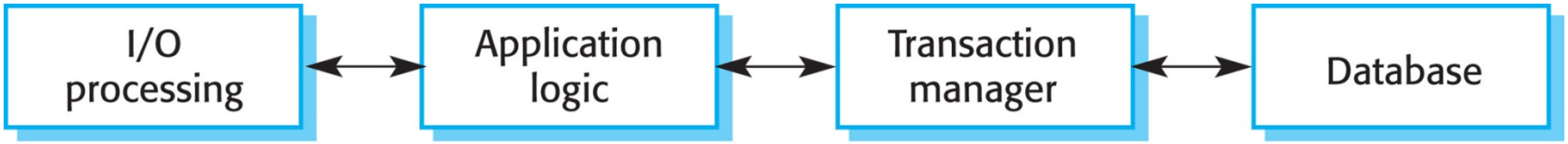


Transaction-Processing Systems

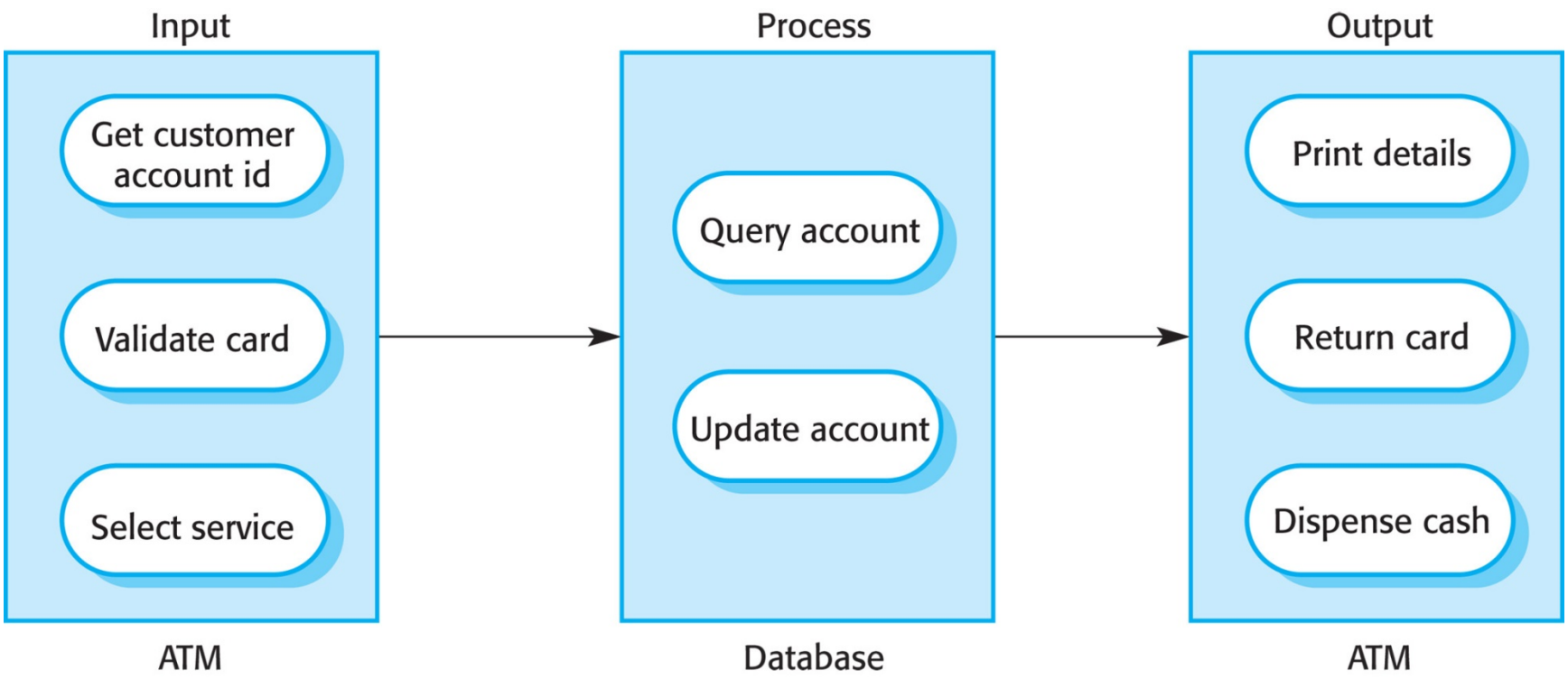
- Process user requests for information from a database or requests to update the database
  - From a user perspective a transaction is any coherent sequence of operations that satisfies a goal.
    - E.g., find the times of flights from New York to London
  - Users make asynchronous requests for service, which are then processed by a **transaction manager**.

TPS: A Generic Architecture

- Users make asynchronous requests for service, which are then processed by a **transaction manager**.

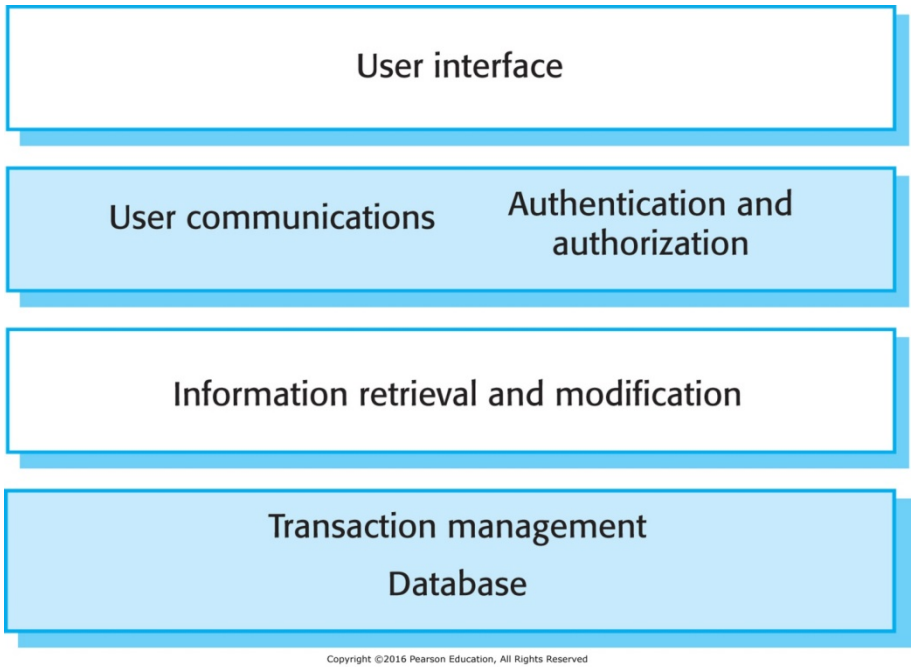


TPS: An ATM Example

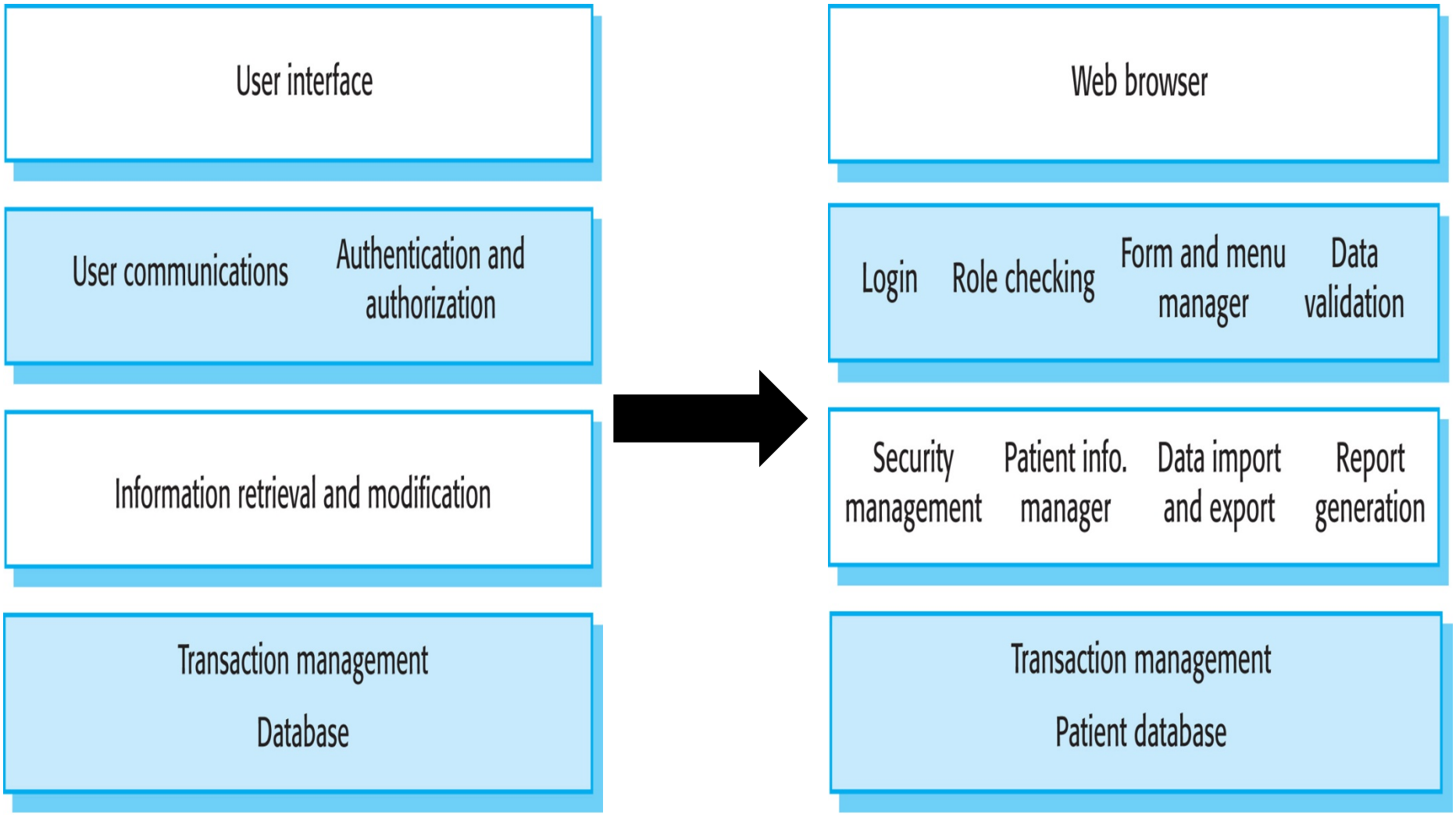


TPS: Information Systems

- Information systems have a generic architecture that can be organized as a layered (aka tiered) architecture.
  - They are considered as transaction-based systems, because interaction with these systems generally involves database transactions.
  - Layers usually include:
    - The user interface
    - User communications
    - Information retrieval
    - **System database**



The Architecture of the Mentcare System



Copyright ©2016 Pearson Education, All Rights Reserved

Web-Based Information Systems

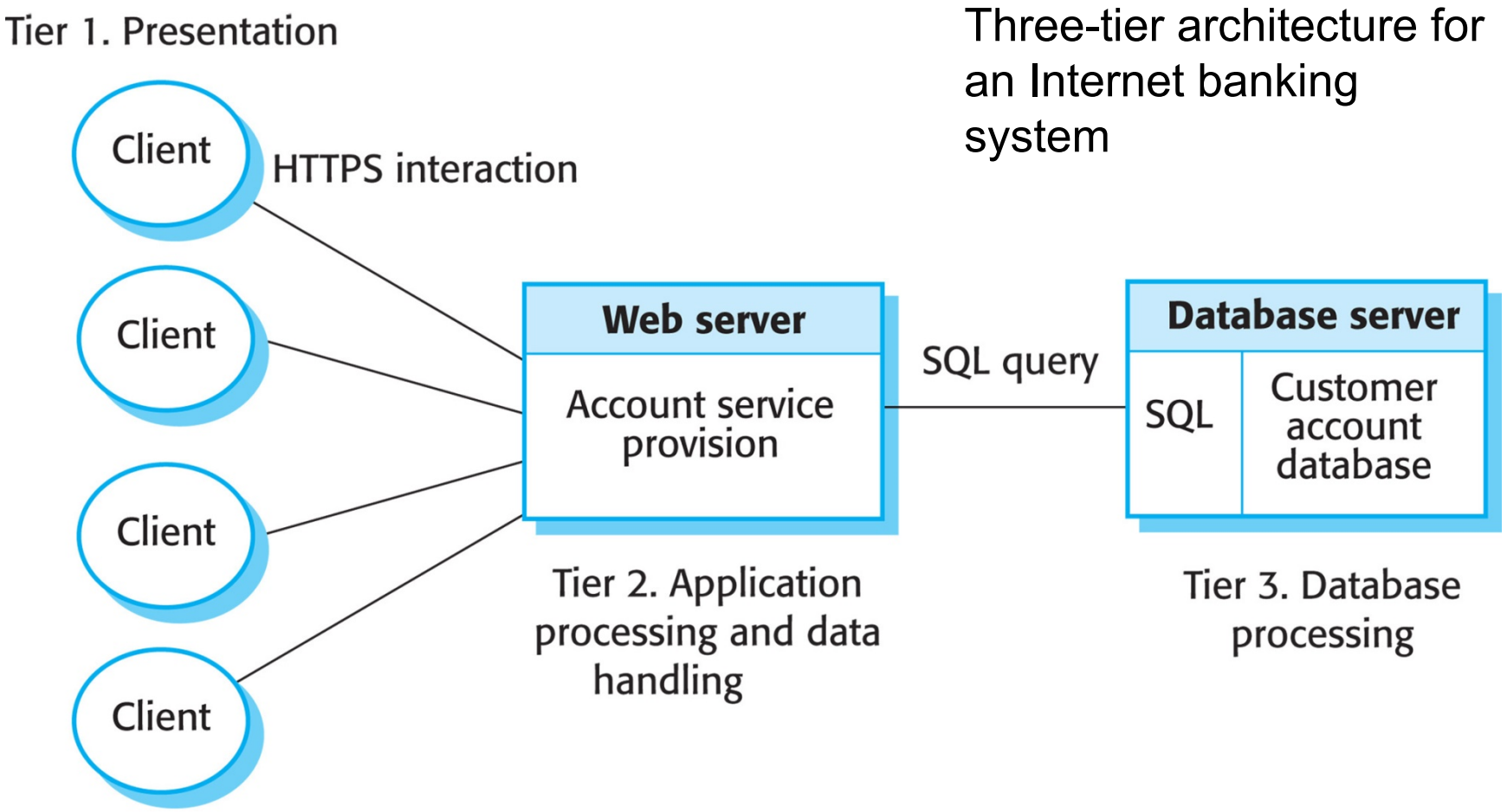
- Information systems are now usually **web-based systems** where the user interfaces are implemented using a web browser.
  - E.g., e-commerce systems are Internet-based information systems that accept electronic orders for goods or services and then arrange delivery of these goods or services to the customer.
  - In an e-commerce system, the application-specific layer includes additional functionality supporting a **shopping cart** in which users can place a number of items in separate transactions, then pay for them all together in a single transaction.

Web-Based Information Systems

- These systems are often implemented as multitier **client/server architectures**.
  - The web server is responsible for all user communications, with the user interface implemented using a web browser.
  - The application server is responsible for implementing application-specific logic as well as information storage and retrieval requests.
  - The database server moves information to and from the database and handles transaction management.

- Nowadays, the application server is usually a part of the web server.

WIS: An Example



Three-tier architecture for an Internet banking system

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Language Processing Systems

Week 6: Architectural Design, Part 1

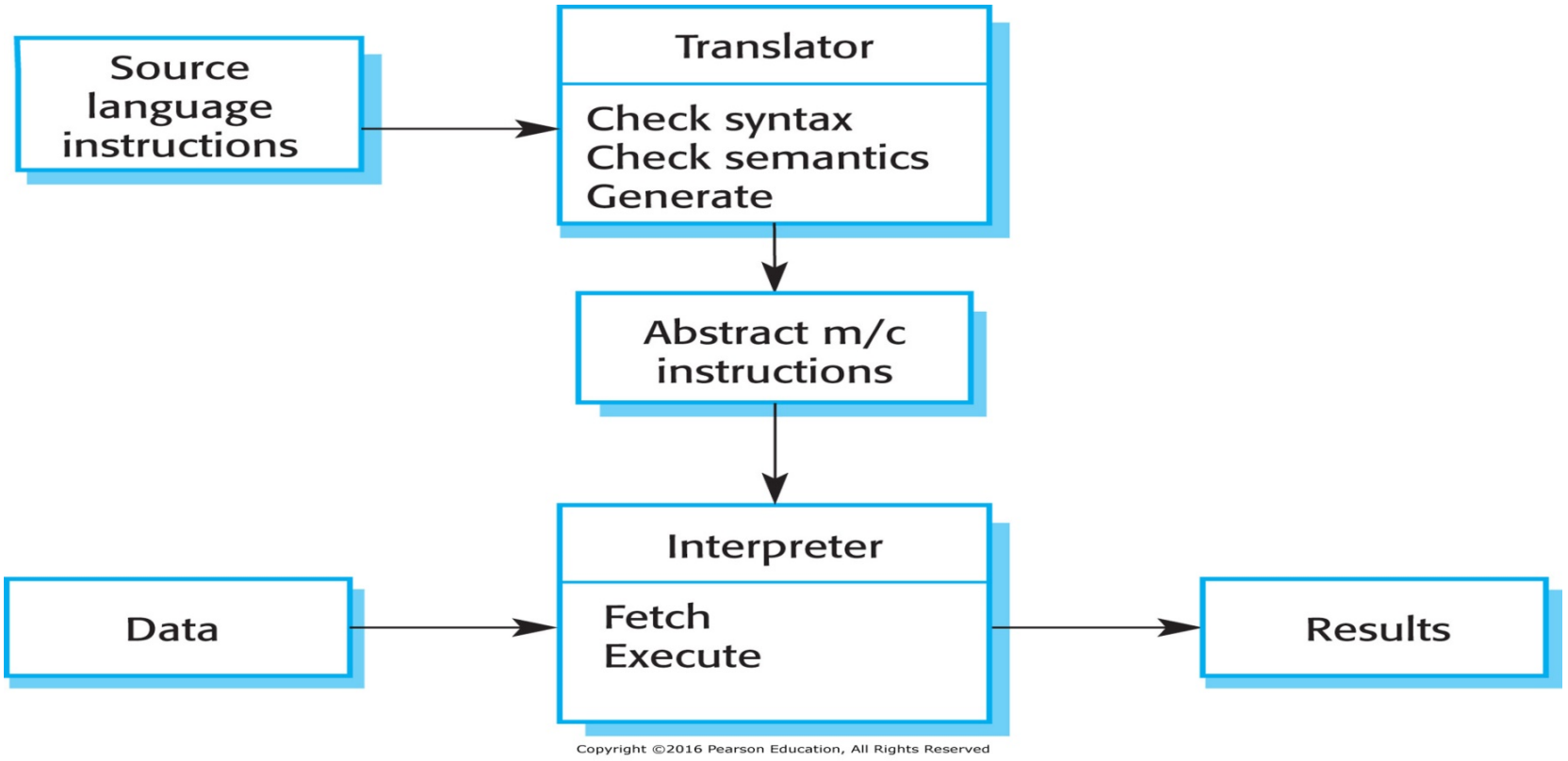
Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

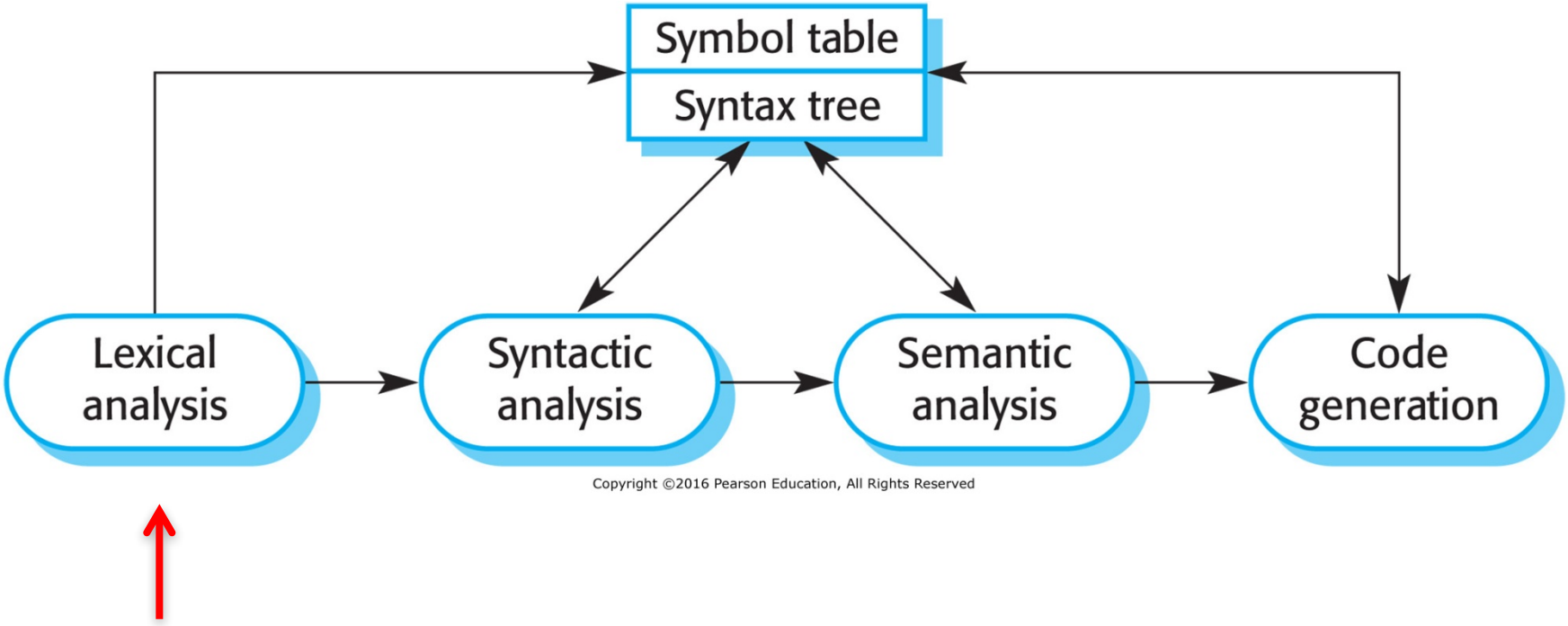
Language Processing Systems

- They are applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.
- They usually accept a natural or artificial language as input and generate some other representation of that language.
- May include an **interpreter** to act on the resulting representations (or instructions).

LPS: A Generic Architecture



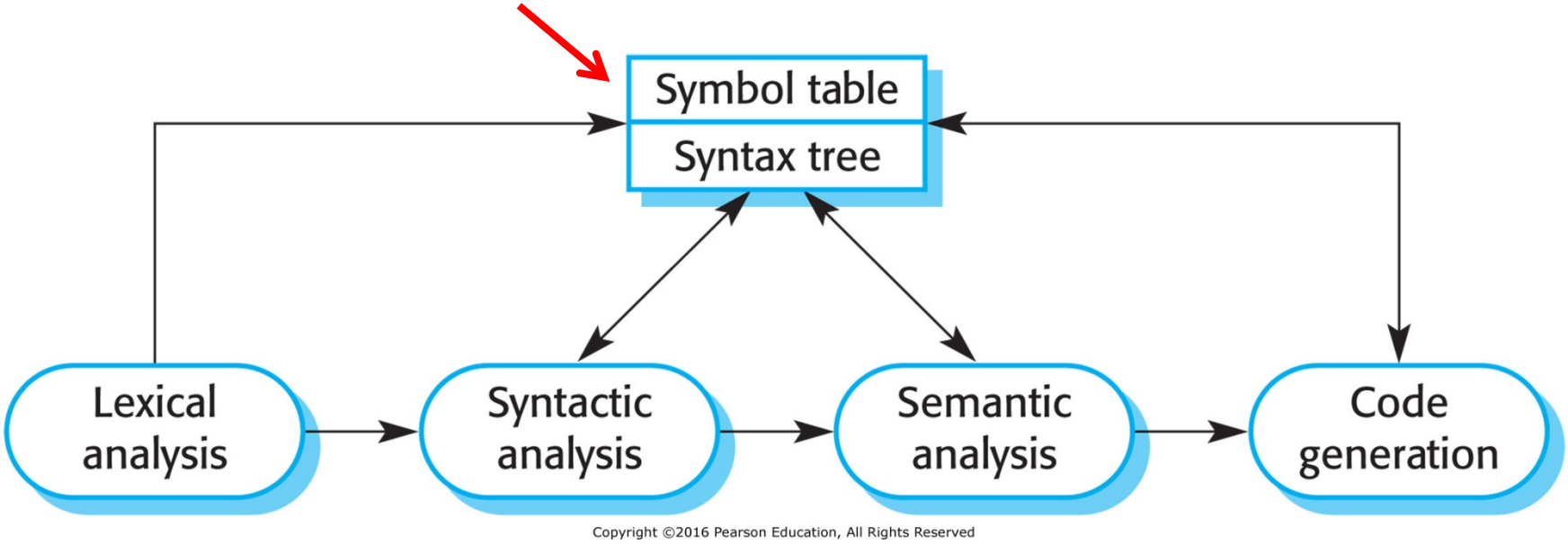
LPS: A Compiler Architecture



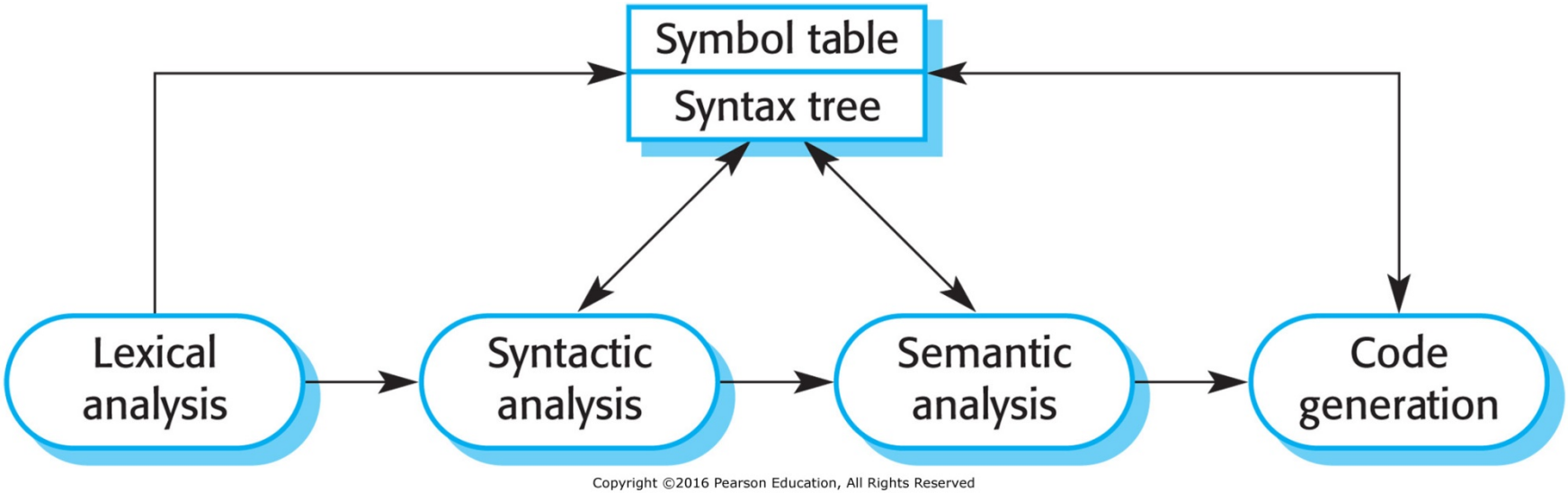
Takes the input program, tokenizes it into tokens, and converts them to an internal form

A Compiler Architecture

Holds information about the names of entities (variables, class names, object names, etc.) used in the program that is being compiled

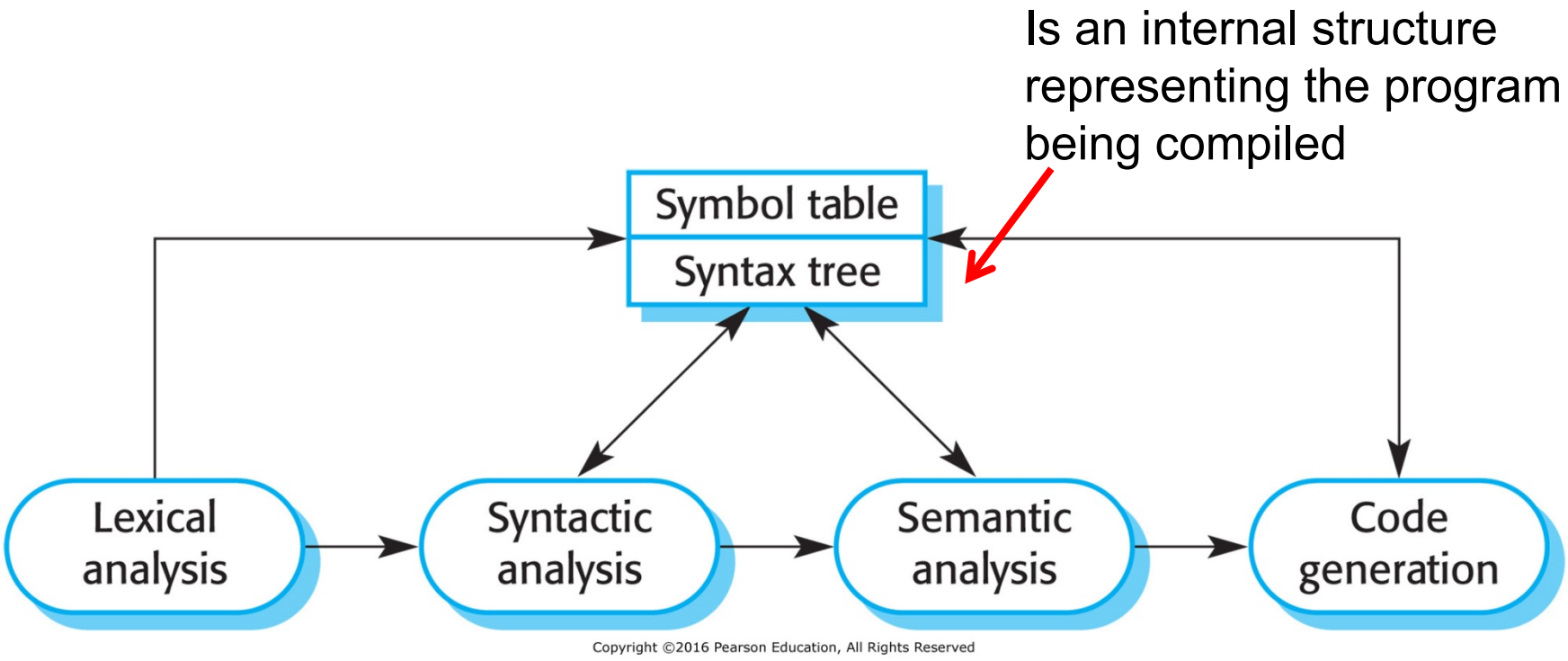


A Compiler Architecture

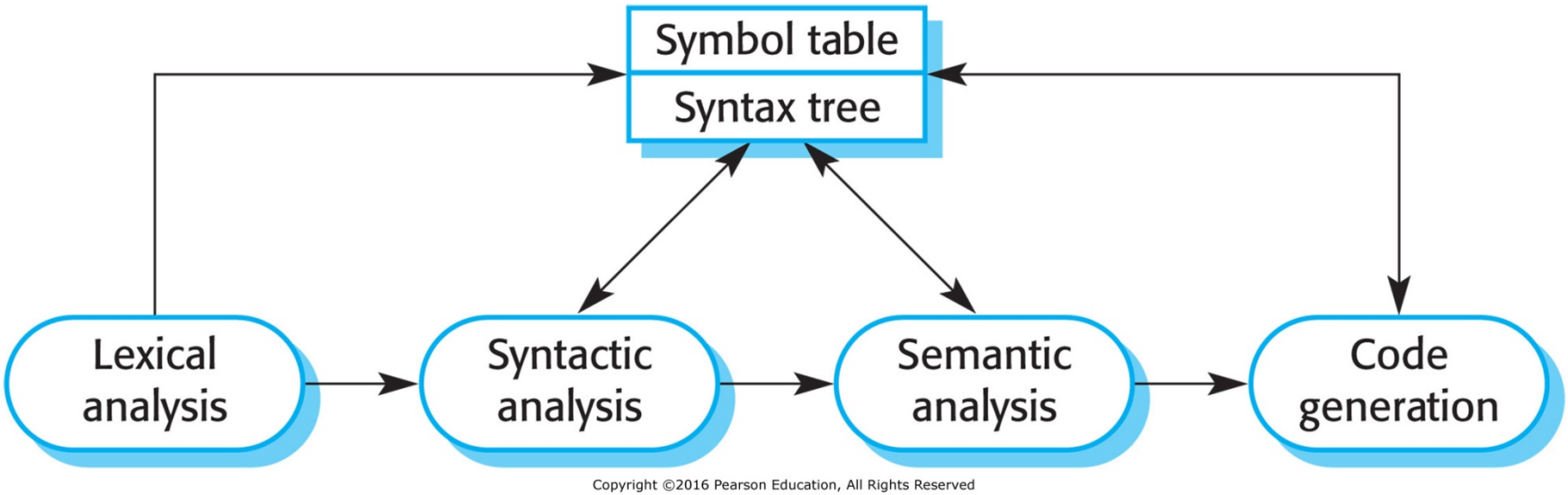


Checks the syntax of the language

A Compiler Architecture

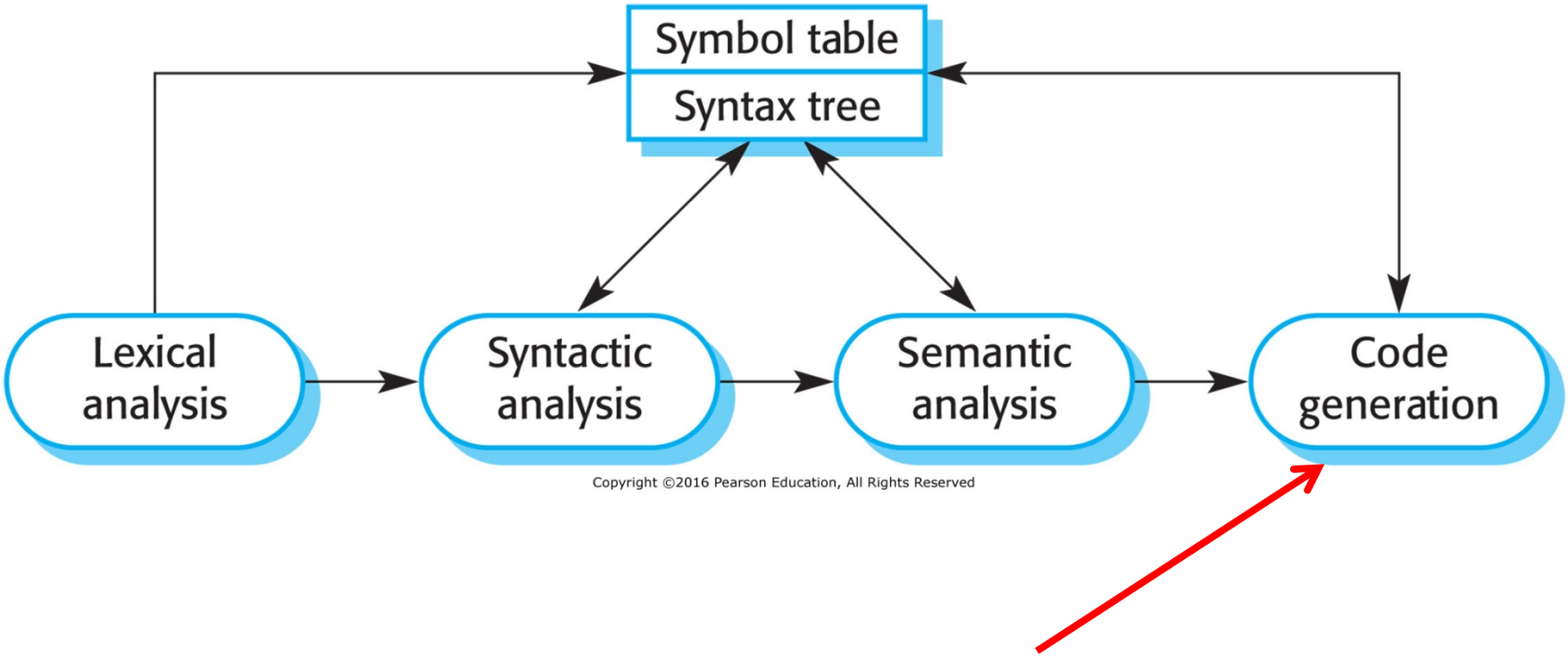


A Compiler Architecture



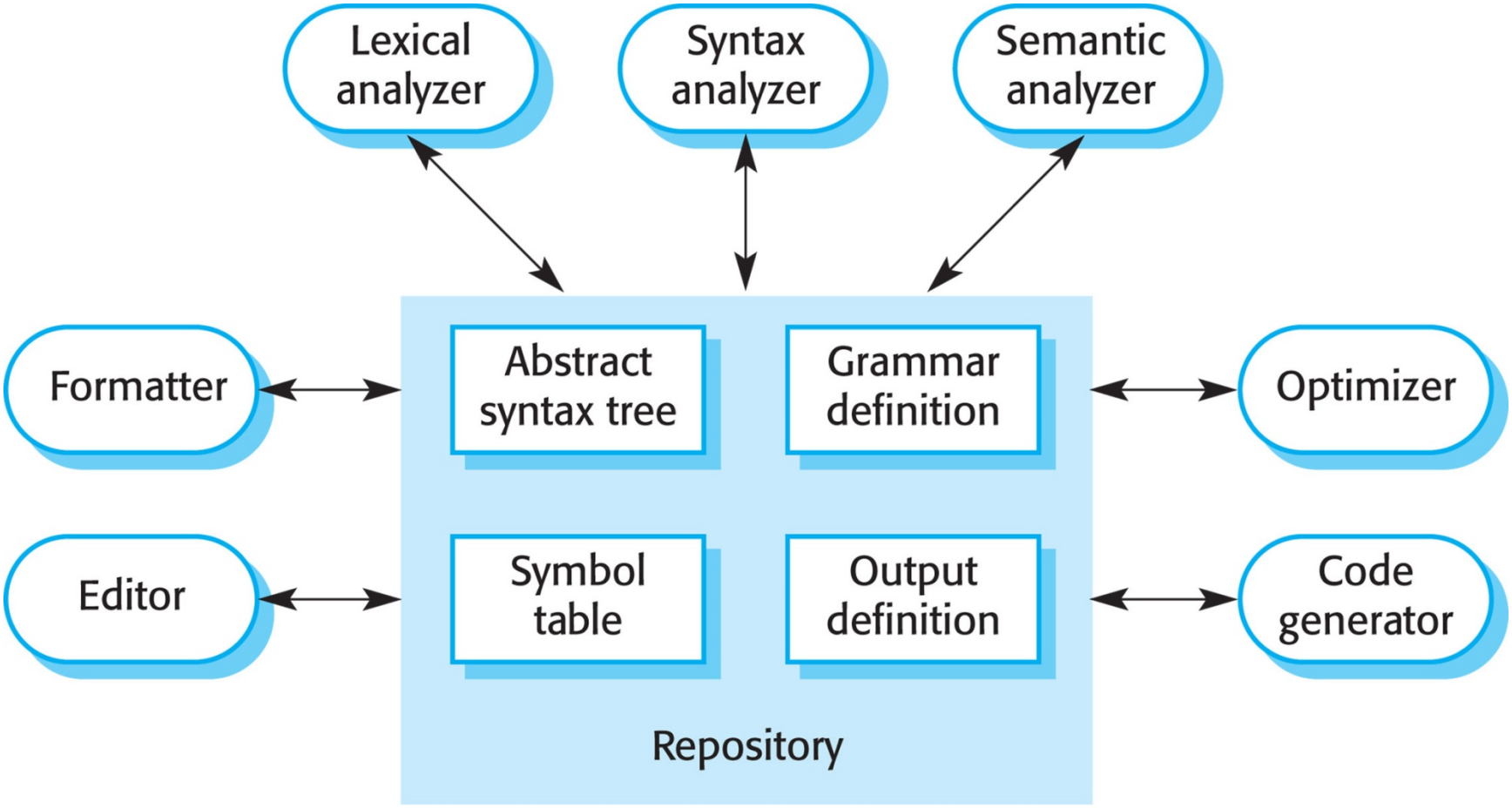
Uses information from the syntax tree and the symbol table to check the semantic correctness of the input program

A Compiler Architecture



"Walks" the syntax tree and generates machine code

A Repository Architecture for a Language-Processing System



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Event Processing Systems

Week 6: Architectural Design, Part 1

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Event-Processing Systems

- Event-processing systems respond to events in the system's **environment** or **user interface**.

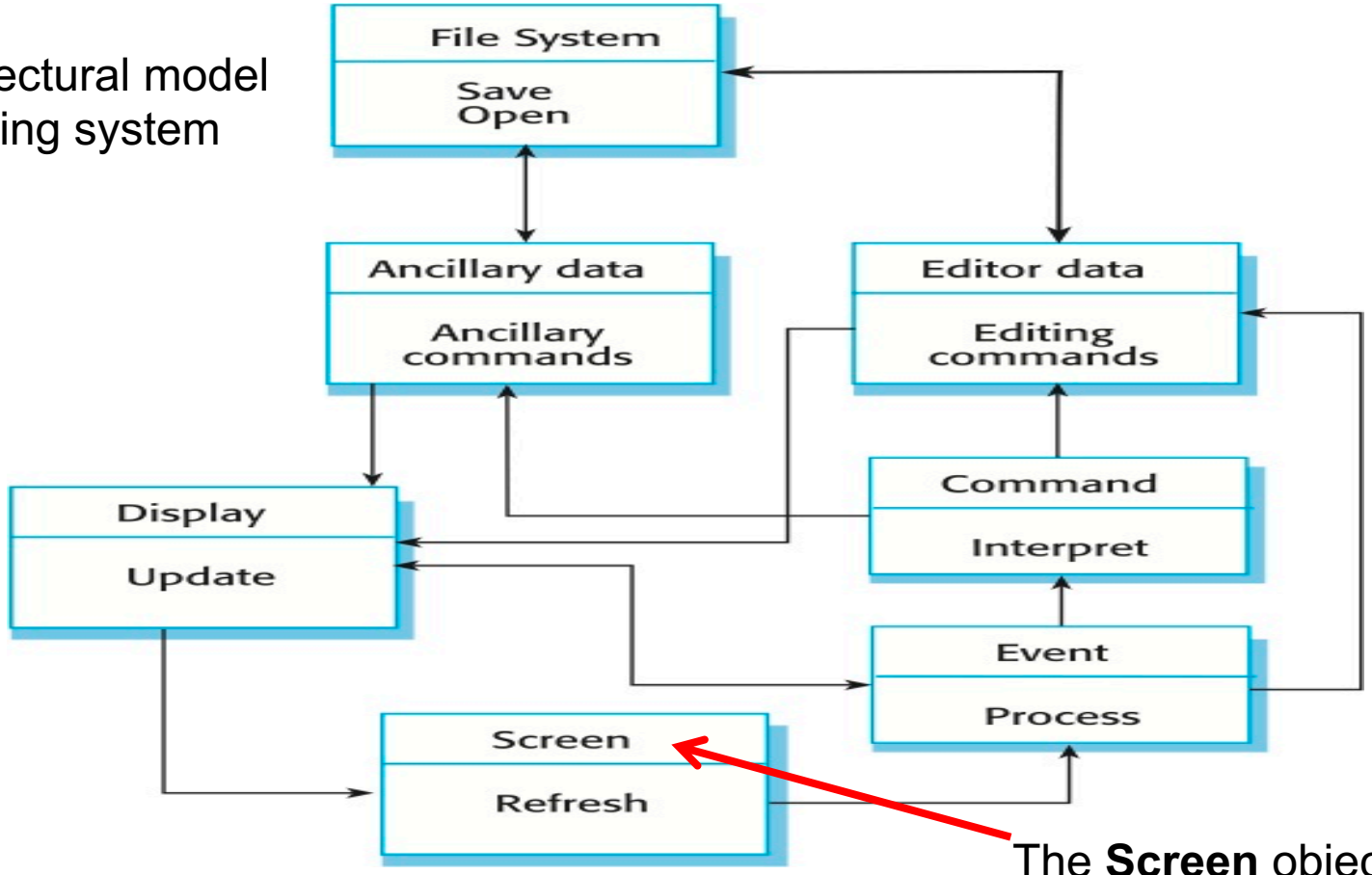
- The key characteristic of event-processing systems is that the timing of events is unpredictable and the system must be able to cope with these events when they occur.

Event-Processing Systems

- We all use such event-based systems on our own computers—word processors, games, and so on.
  - They are all driven by events from the user interface.
  - The system detects and interprets events.
  - User interface events represent implicit commands to the system, which takes some action to obey that command.
    - E.g. if you are using a word processor and you double-click on a word, the double-click event means “select that word.”

Event-Processing Systems

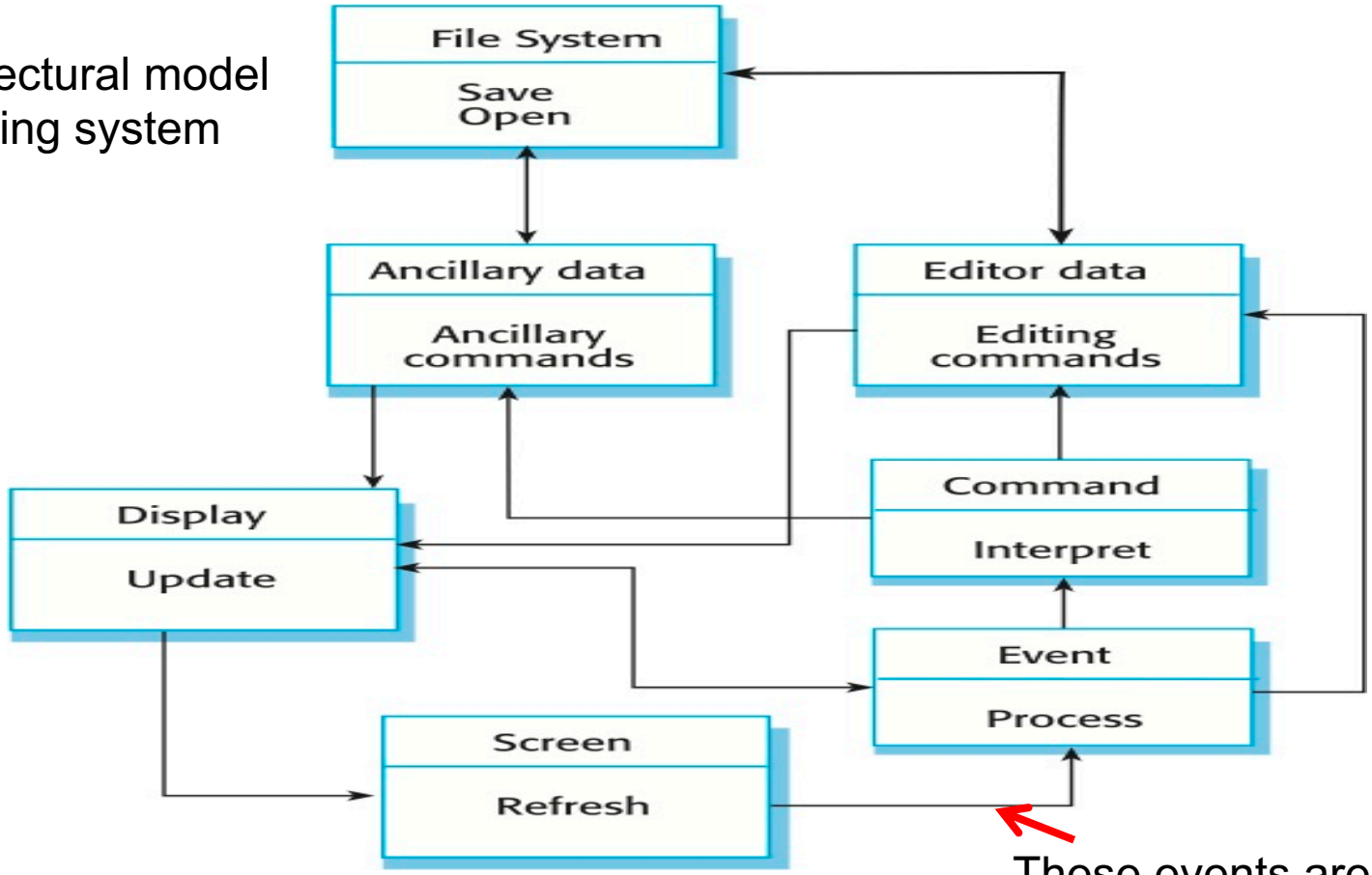
An architectural model of an editing system



The **Screen** object monitors the screen memory segment and detects events that occur.

Event-Processing Systems

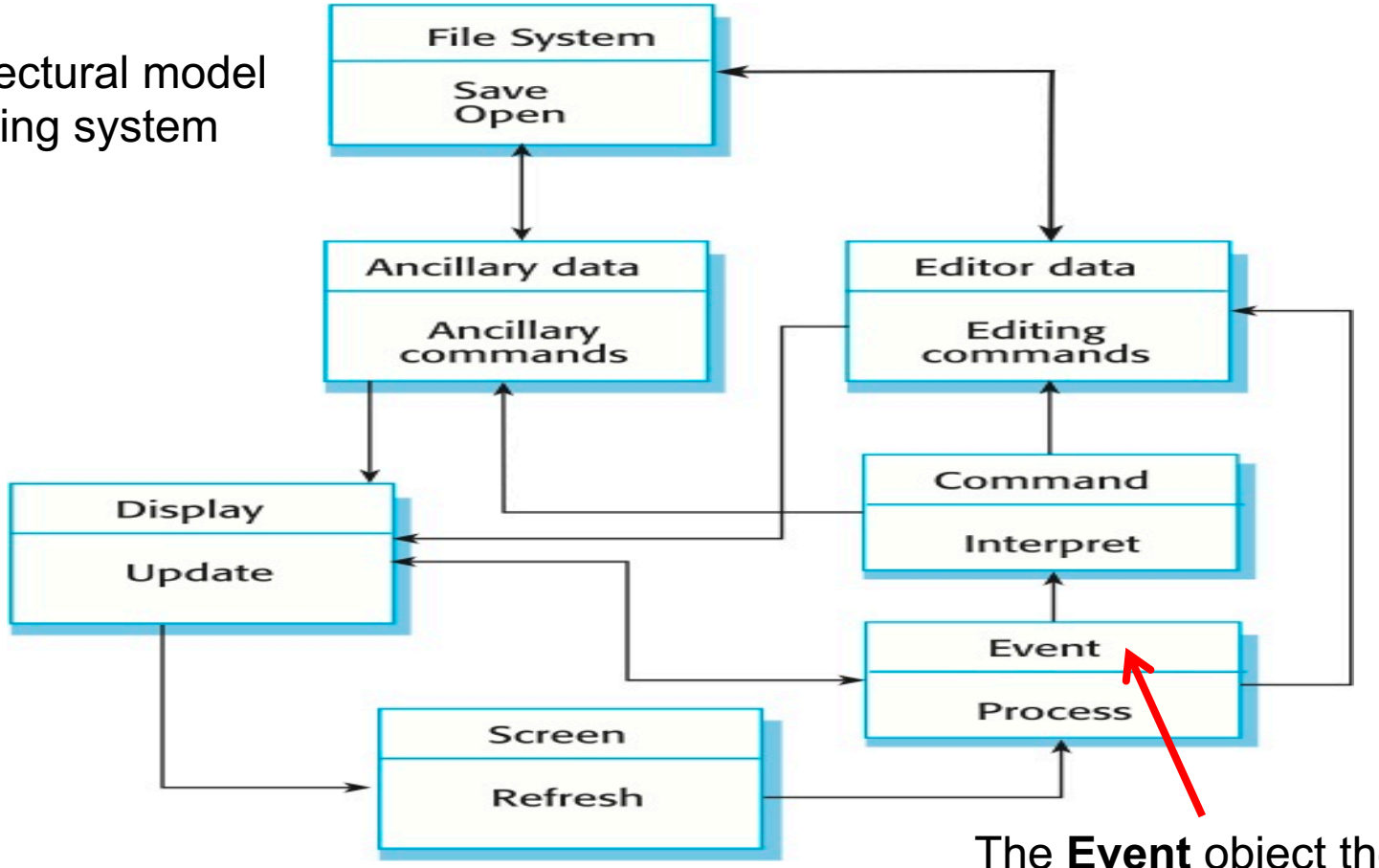
An architectural model of an editing system



These events are then passed to the event-processing object along with their screen coordinates.

Event-Processing Systems

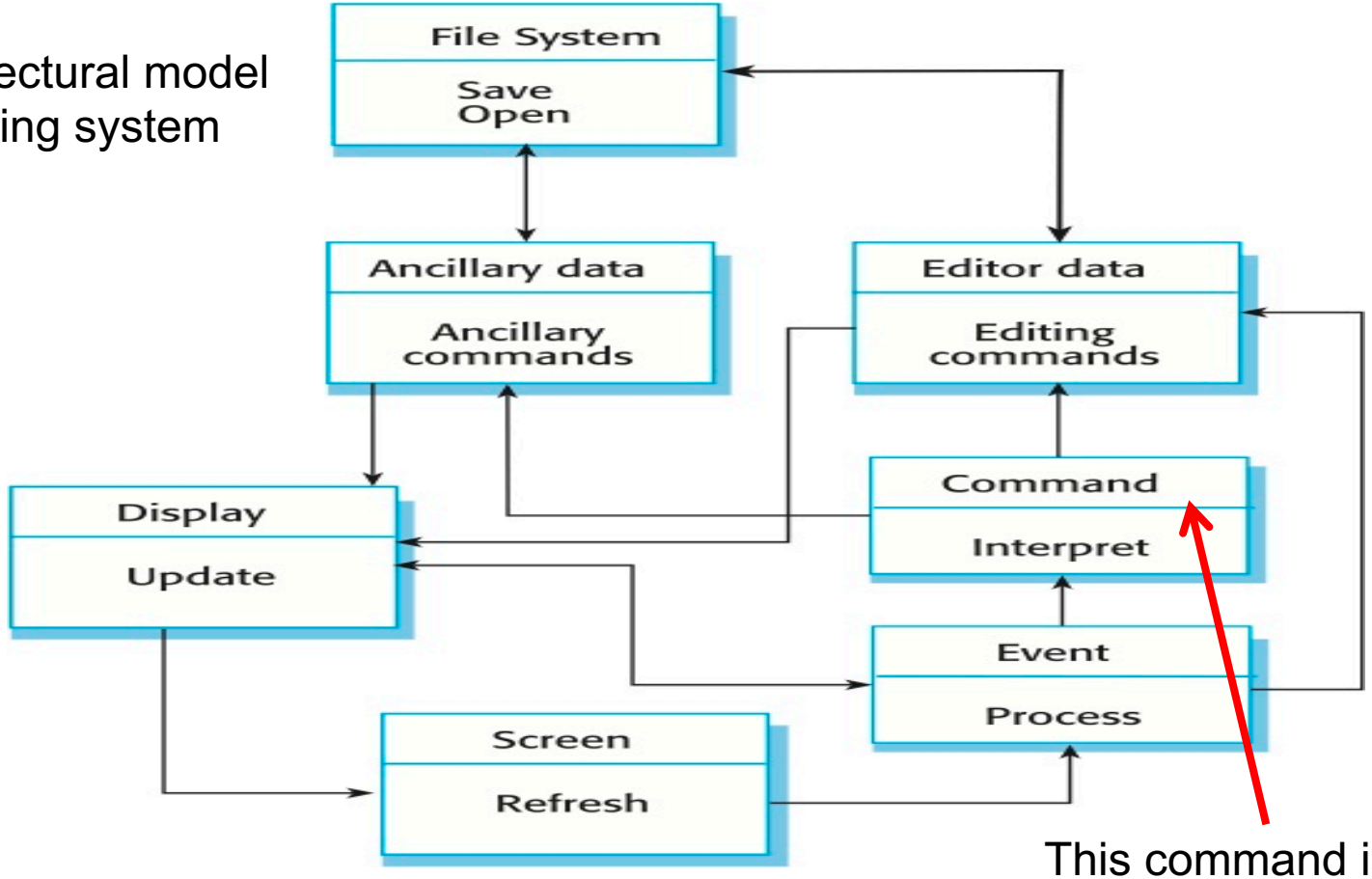
An architectural model of an editing system



The **Event** object then uses knowledge of what is displayed to translate this into the appropriate editing command.

Event-Processing Systems

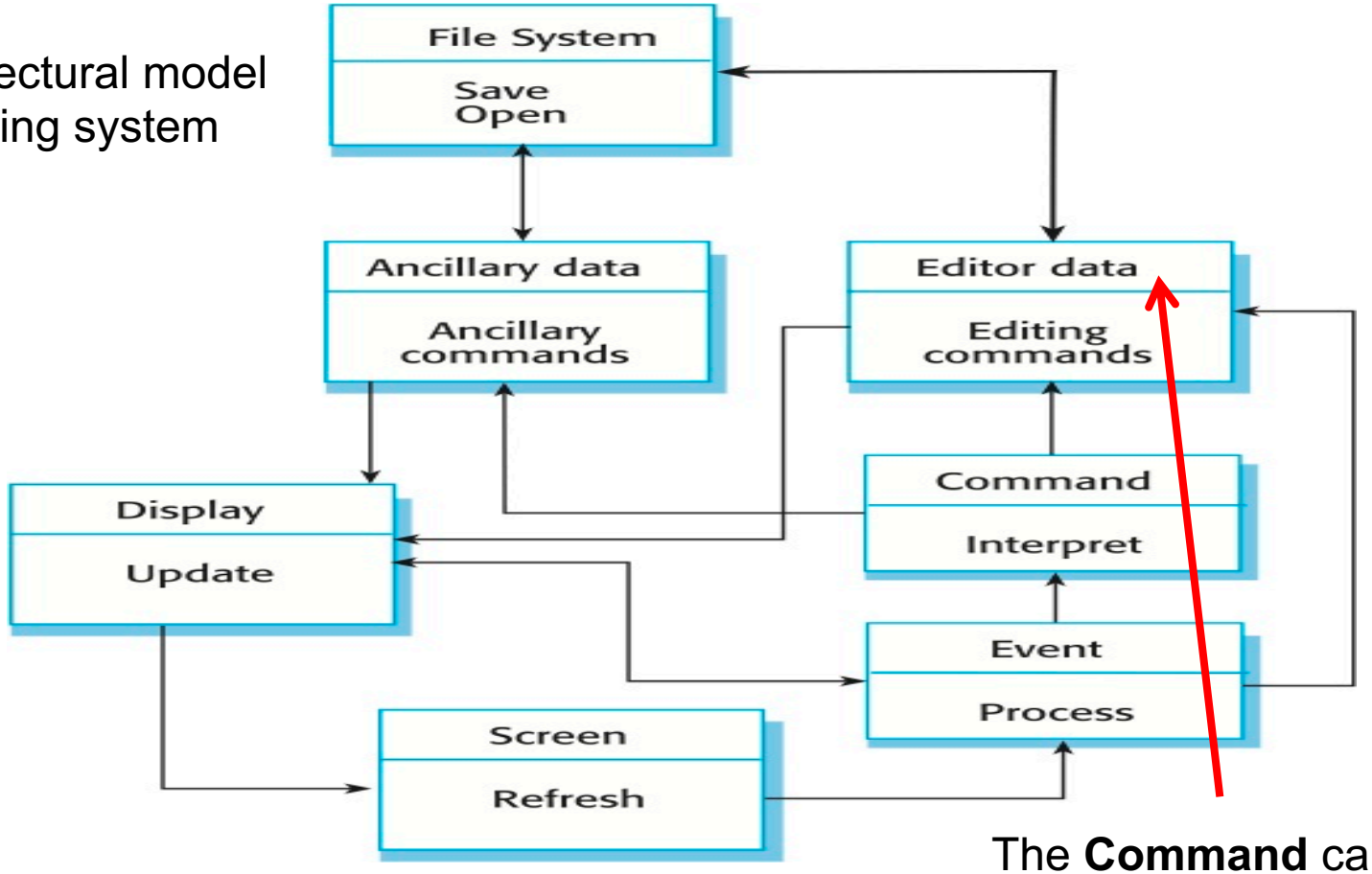
An architectural model of an editing system



This command is then passed to the **Command** object responsible for command interpretation.

Event-Processing Systems

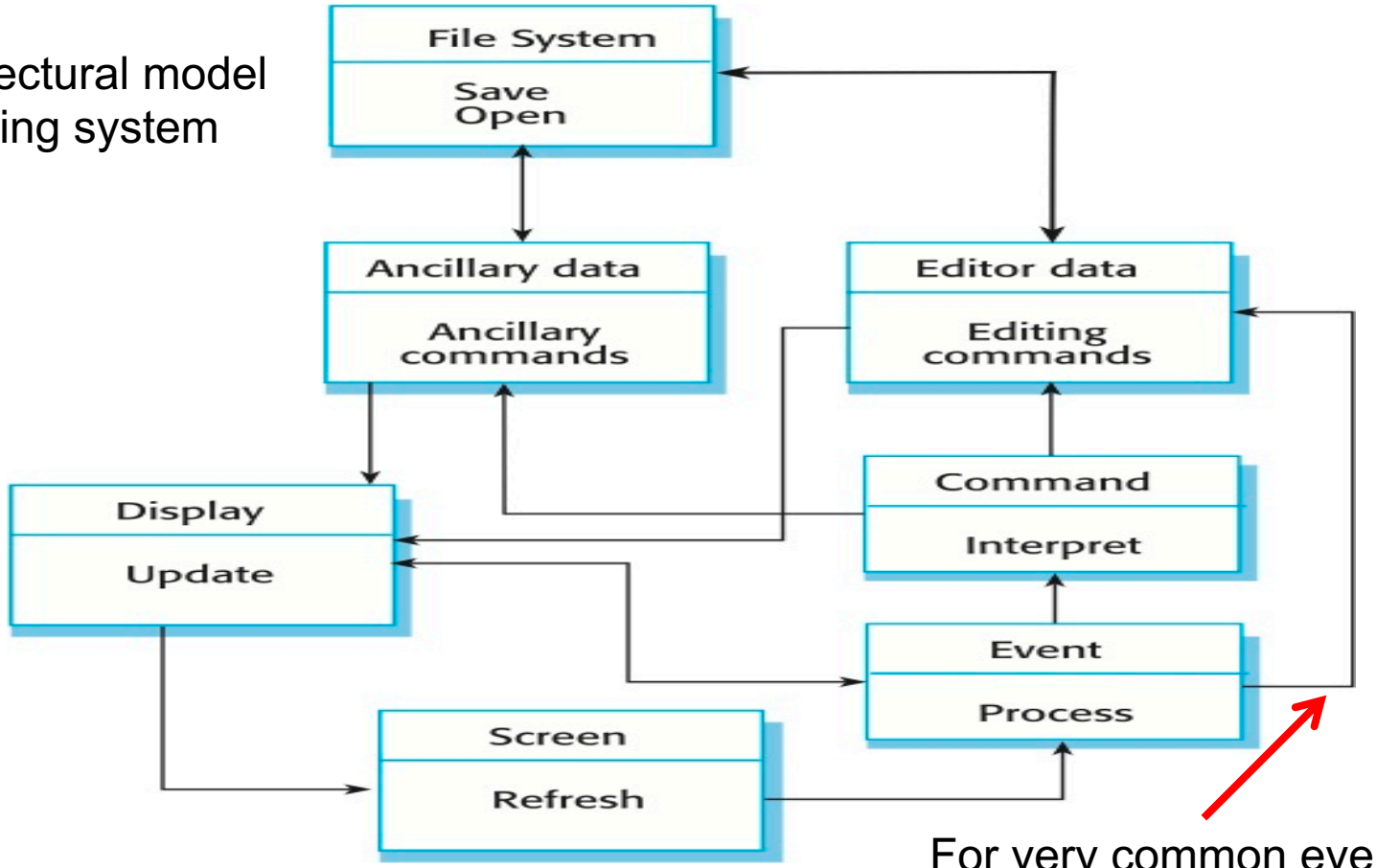
An architectural model of an editing system



The **Command** calls the appropriate method in the **Editor data** object to execute the command.

Event-Processing Systems

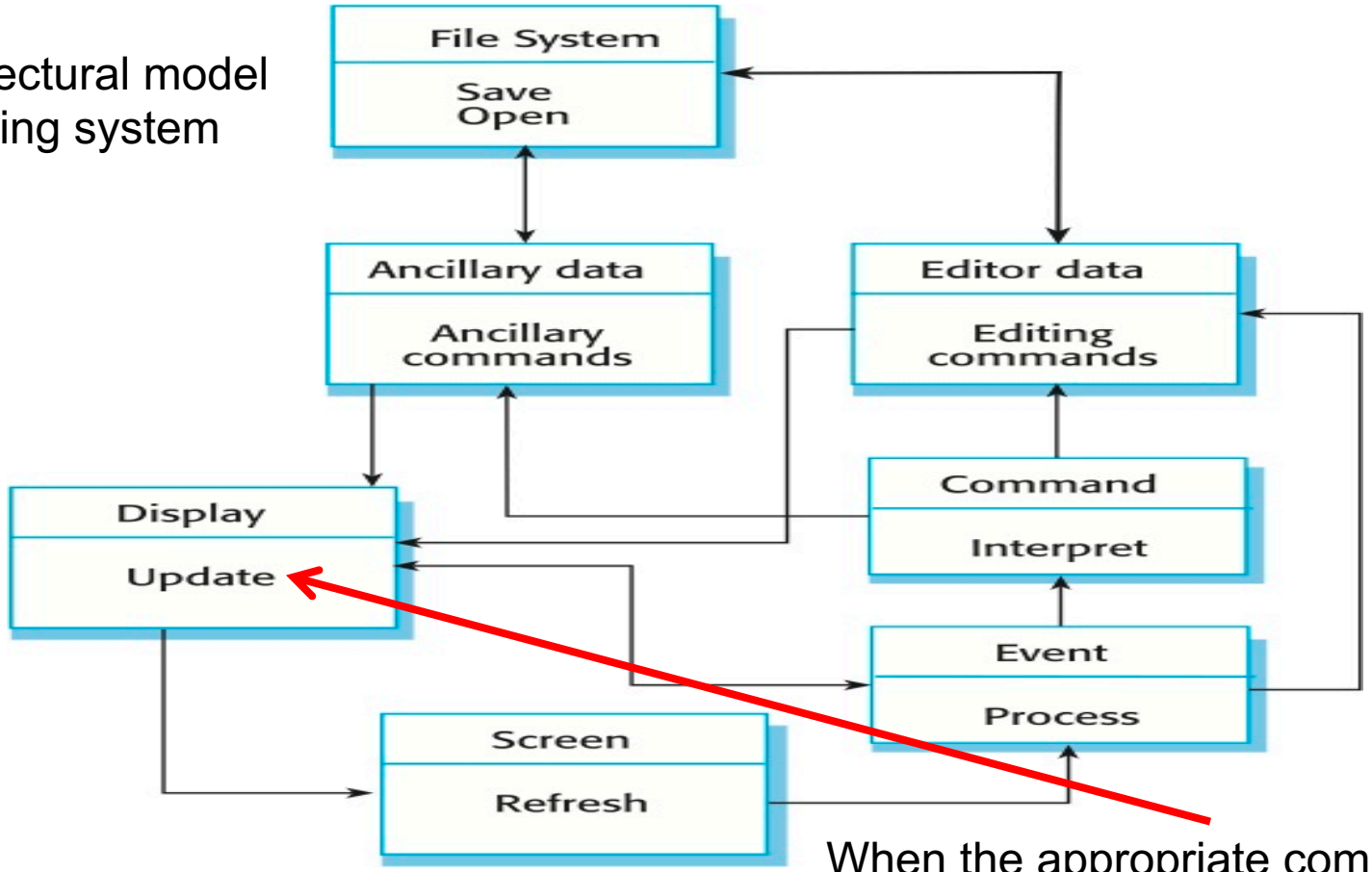
An architectural model of an editing system



For very common events, such as **mouse clicks** or **key presses**, the event object can communicate directly with the data structure.

Event-Processing Systems

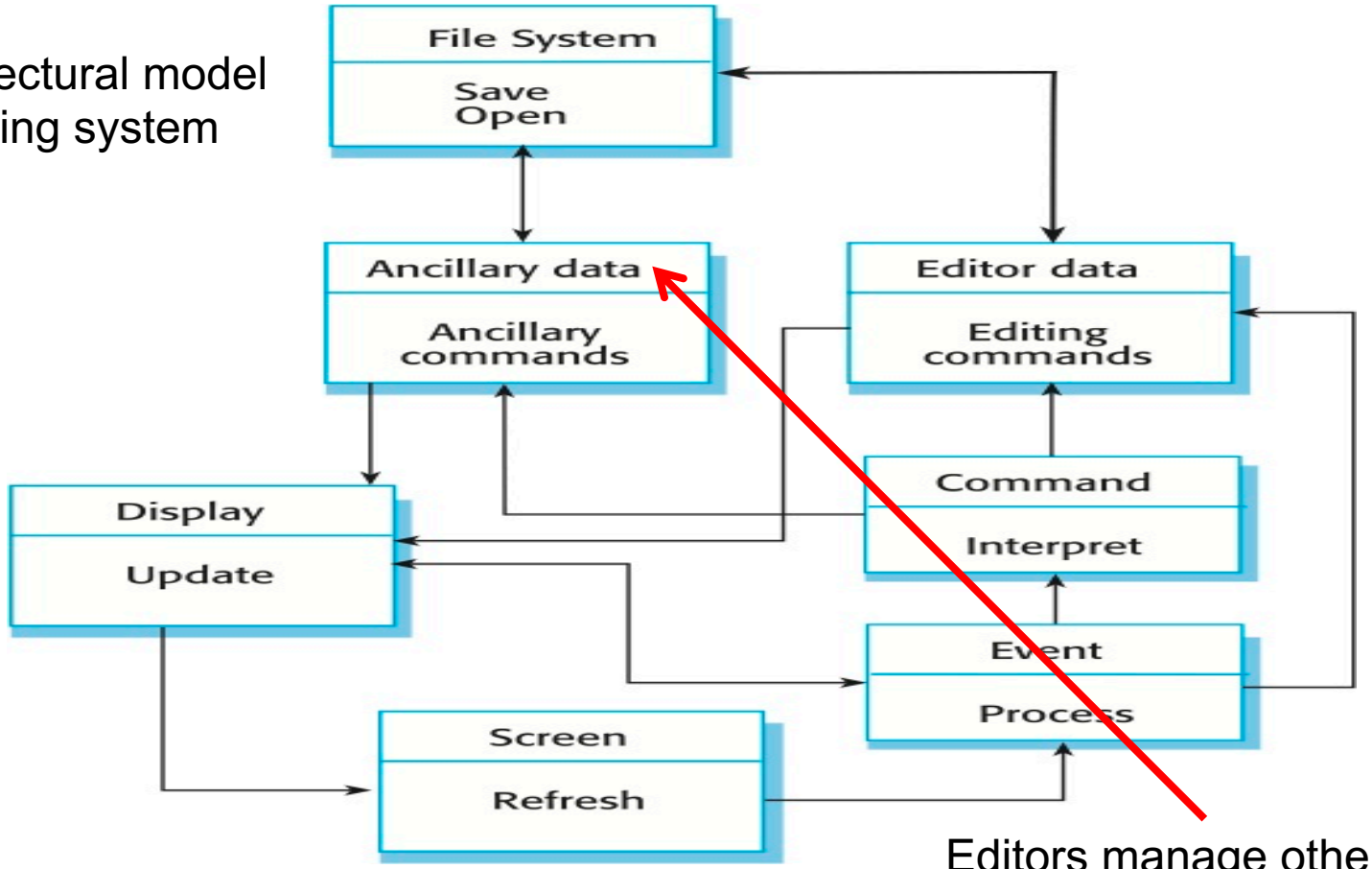
An architectural model of an editing system



When the appropriate command method is called, it updates the data structure and calls the **Update** method in **Display** to display the modified data.

Event-Processing Systems

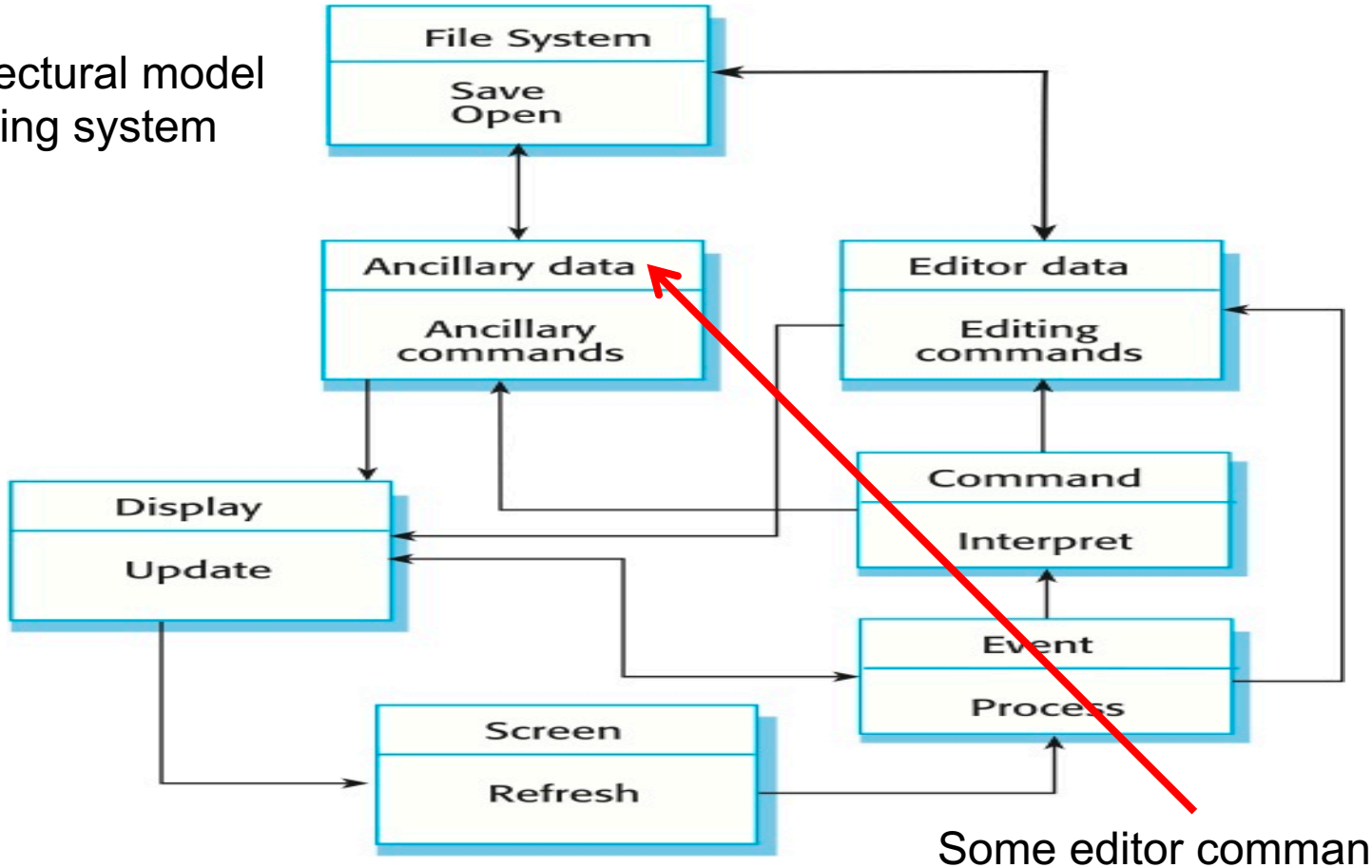
An architectural model of an editing system



Editors manage other data such as **styles** and **preferences**. They have been bundled together under **Ancillary data**.

Event-Processing Systems

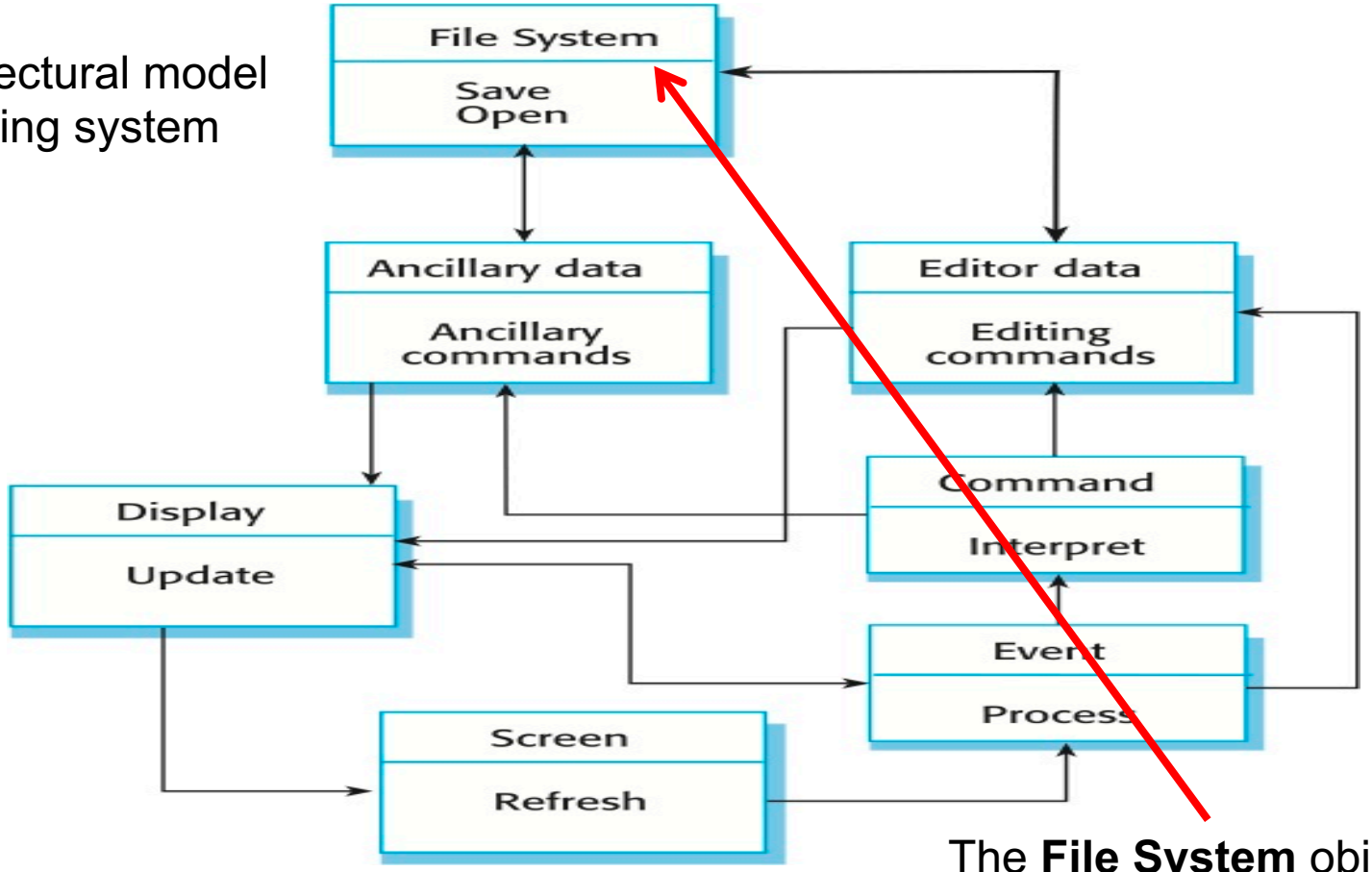
An architectural model of an editing system



Some editor commands, such as a command to initiate a **spelling check**, are implemented by a method in this object.

Event-Processing Systems

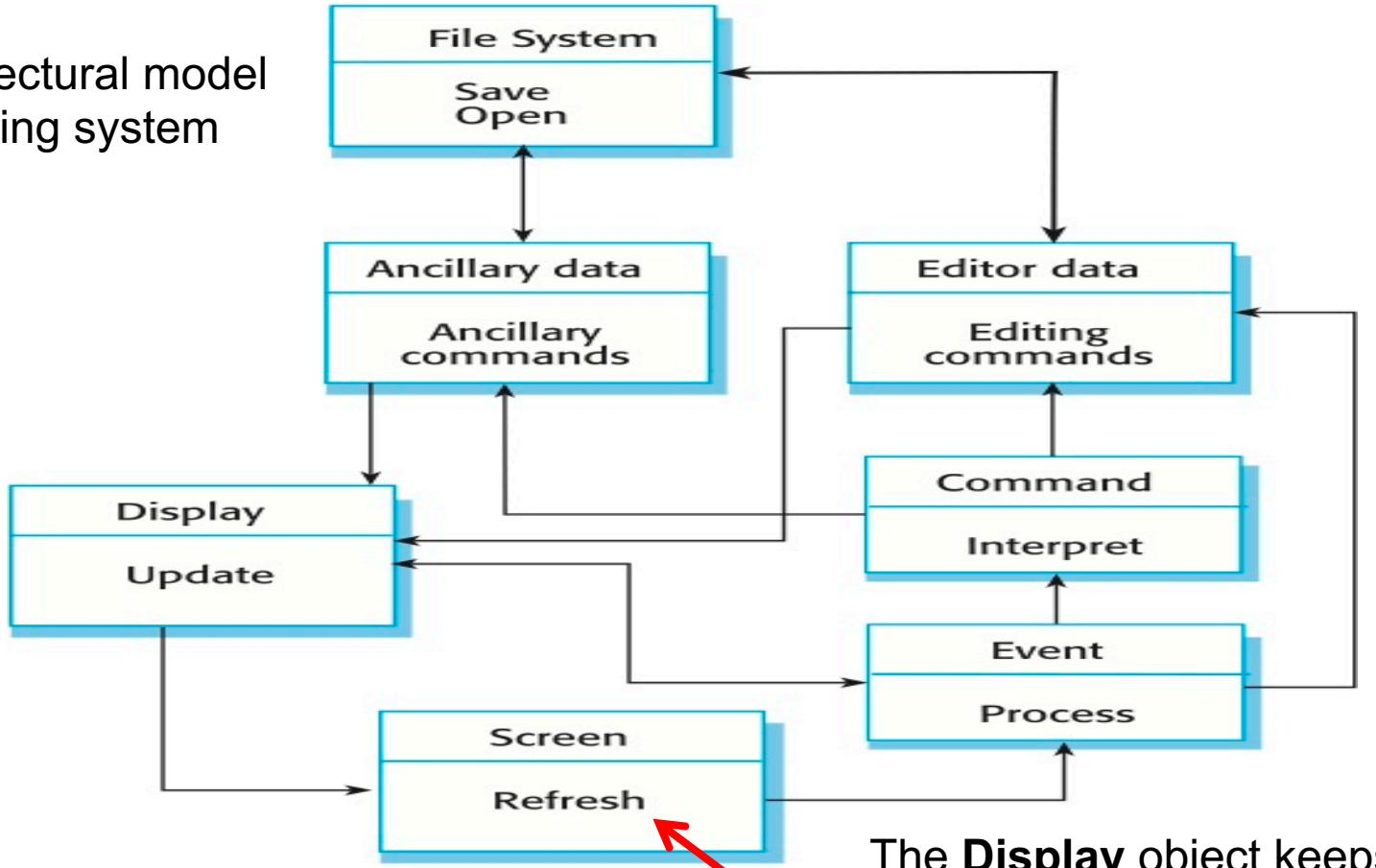
An architectural model of an editing system



The **File System** object handles all opening and saving of files. These can be either editor data or ancillary data files.

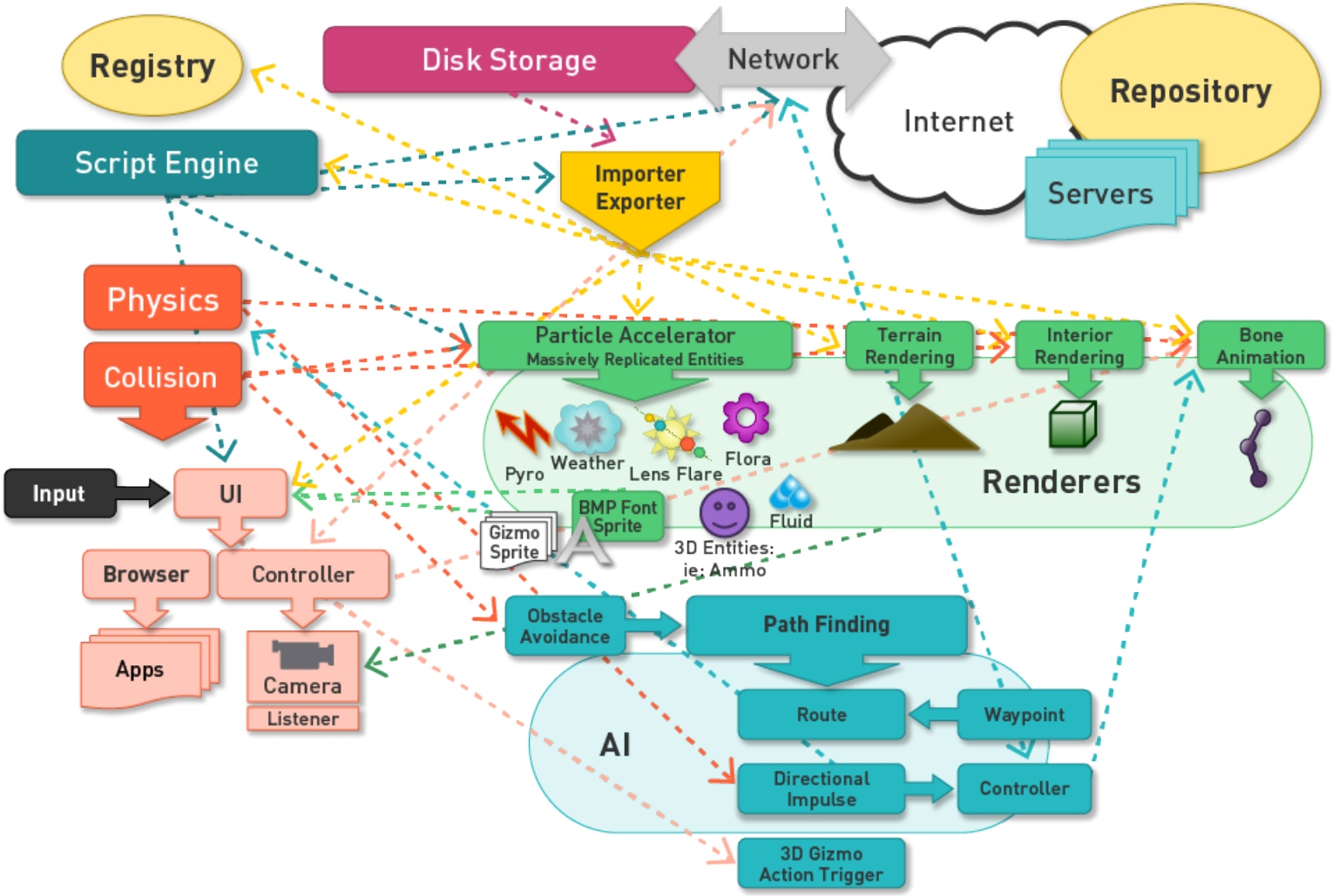
Event-Processing Systems

An architectural model of an editing system



The **Display** object keeps track of the organization of the screen display. It calls the Refresh method in Screen when the display has been changed.

Event-Processing Systems: Games



https://forum.thegamecreators.com/thread/165850

EPS: Real-Time Systems

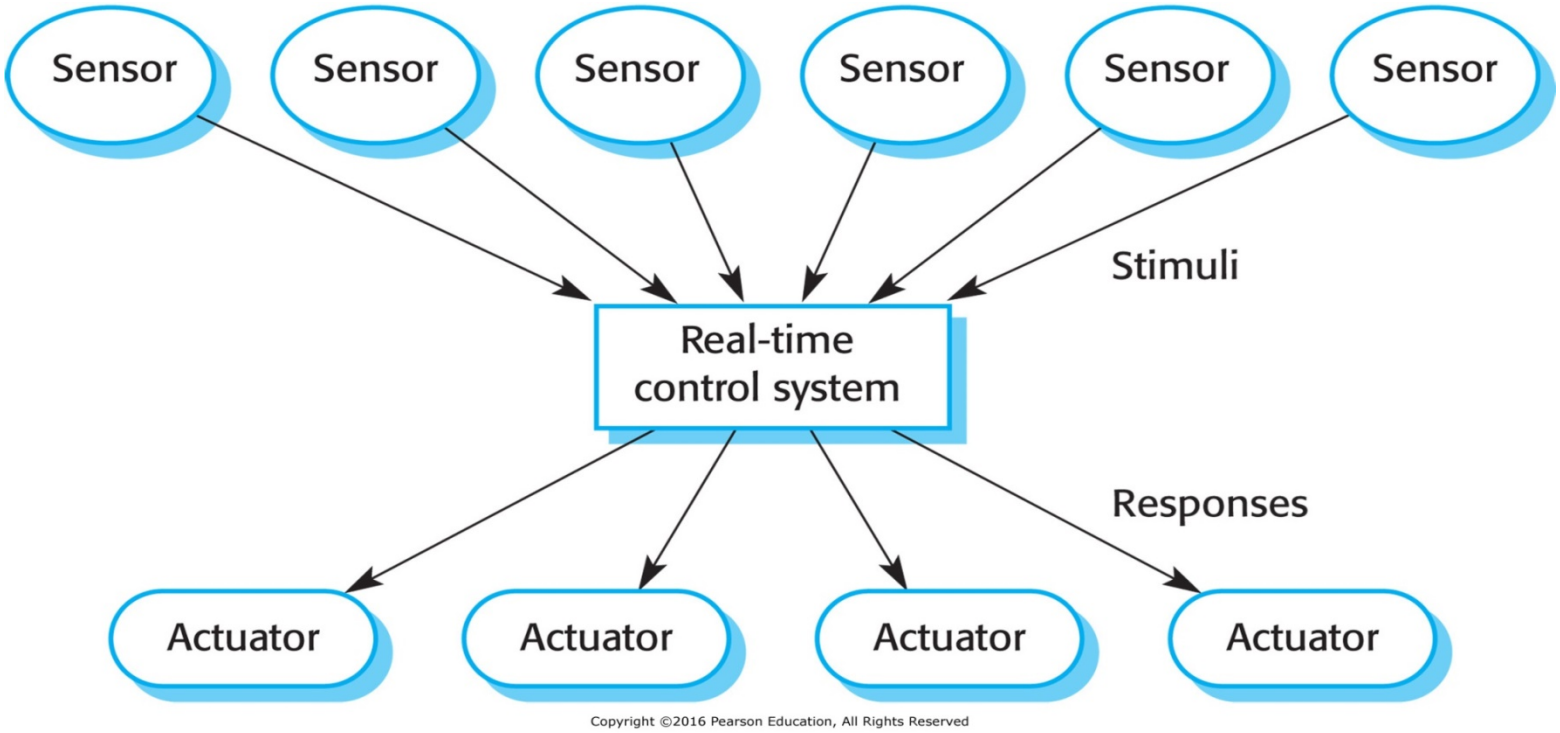
- **Real-time systems**, which take action in real time in response to some external stimulus, are also event-processing systems.
  - For real-time systems, events are not usually user interface events but events associated with the system's (physical) environment.

RTS: A Definitions

- A **real-time system** is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced.
  - A **soft** real-time system is a system whose operation is degraded if results are not produced according to the specified timing requirements.
  - A **hard** real-time system is a system whose operation is incorrect if results are not produced according to the timing specification.

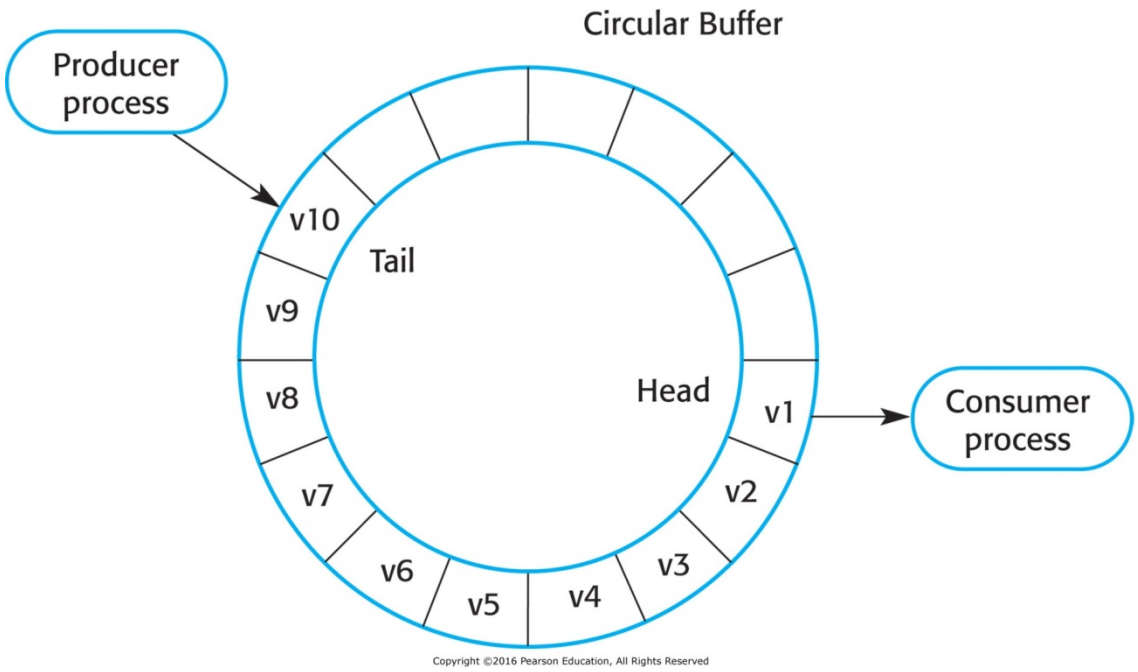
EPS: Real-Time Systems

- Because of the need for real-time response to unpredictable events, these real-time systems are normally organized as a set of **cooperating processes**.

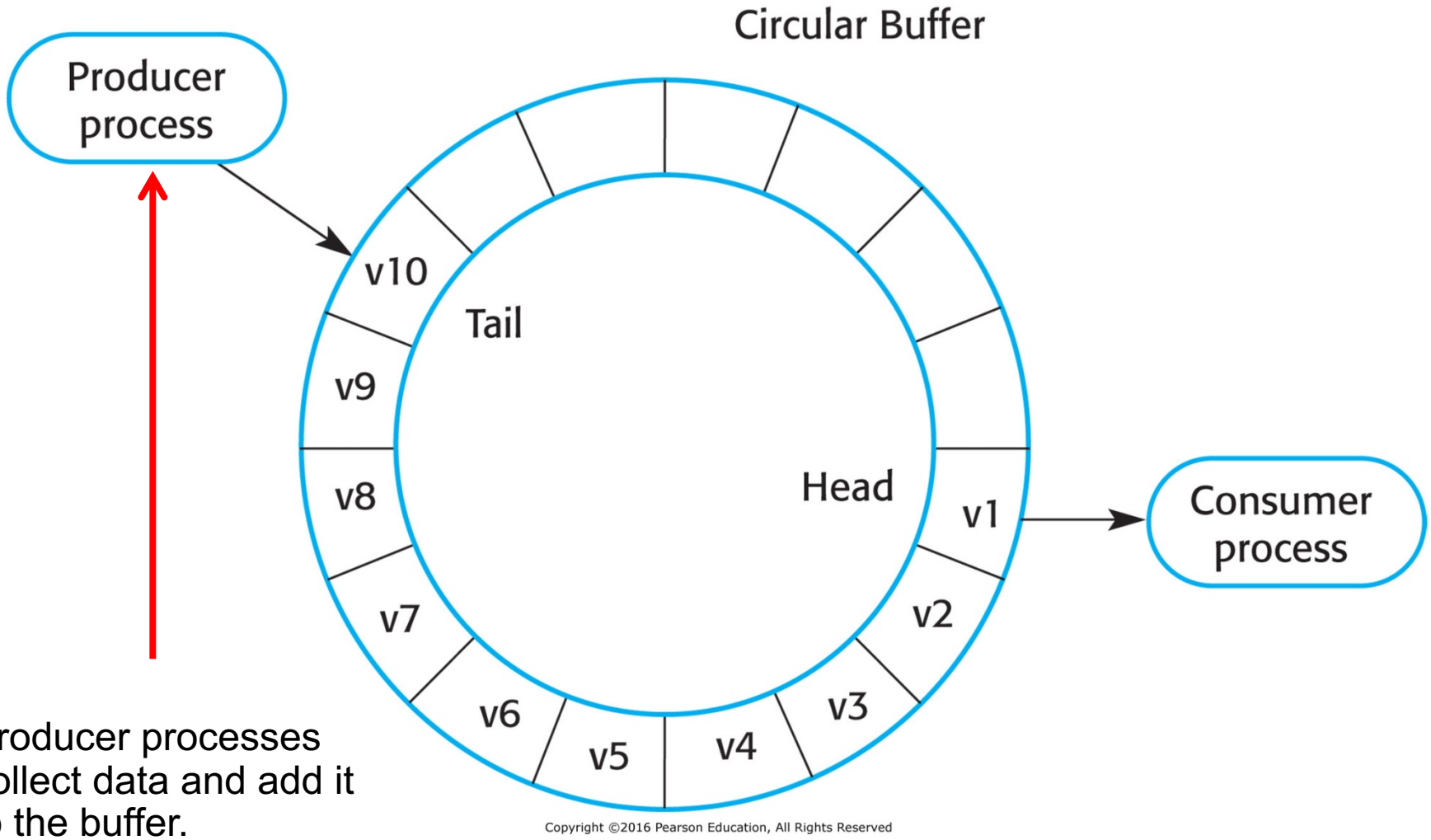


EPS: Real-Time Systems

- Processes in a real-time system have to be coordinated and share information.
  - Process coordination mechanisms ensure **mutual exclusion** to shared resources.

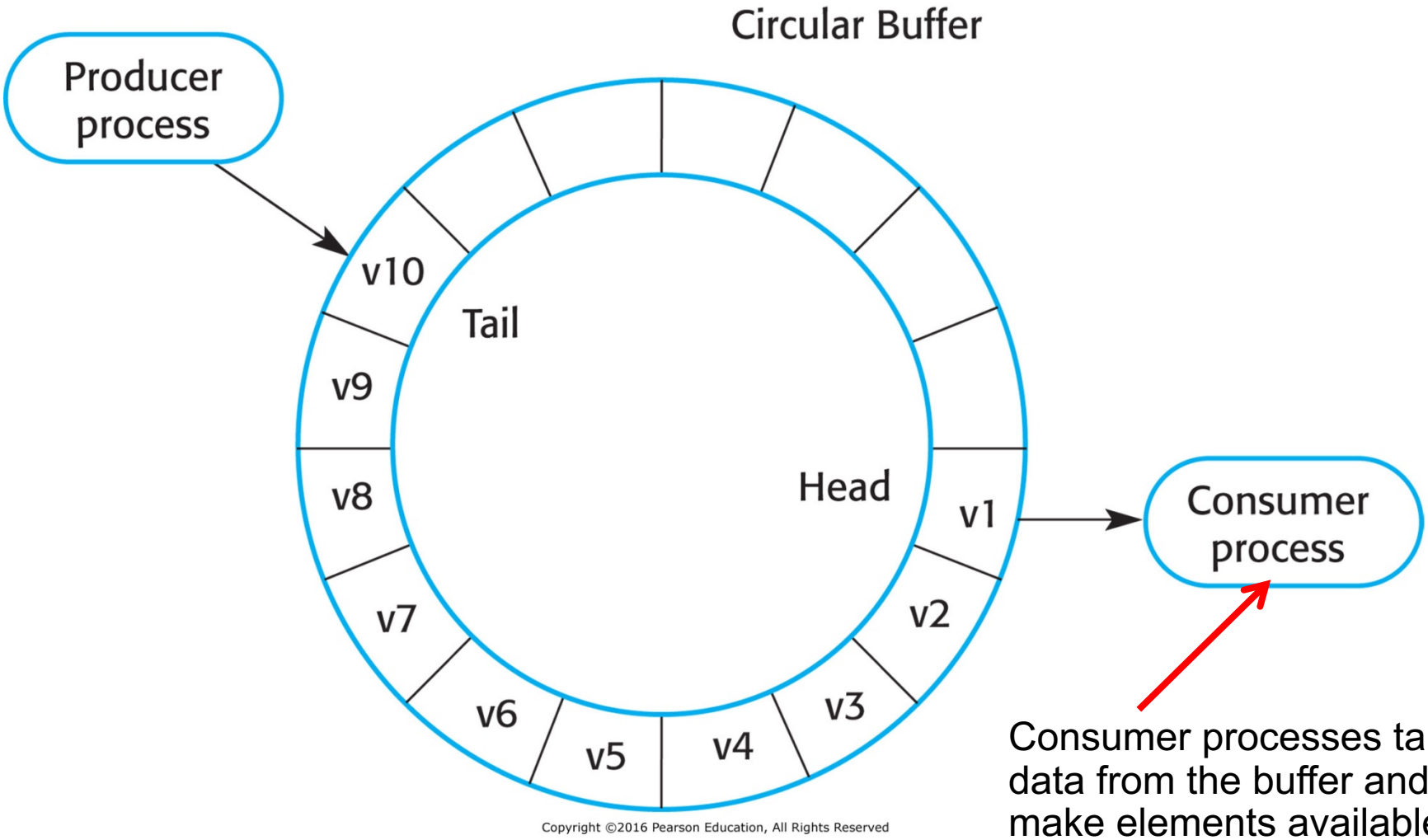


Producer/Consumer Processes



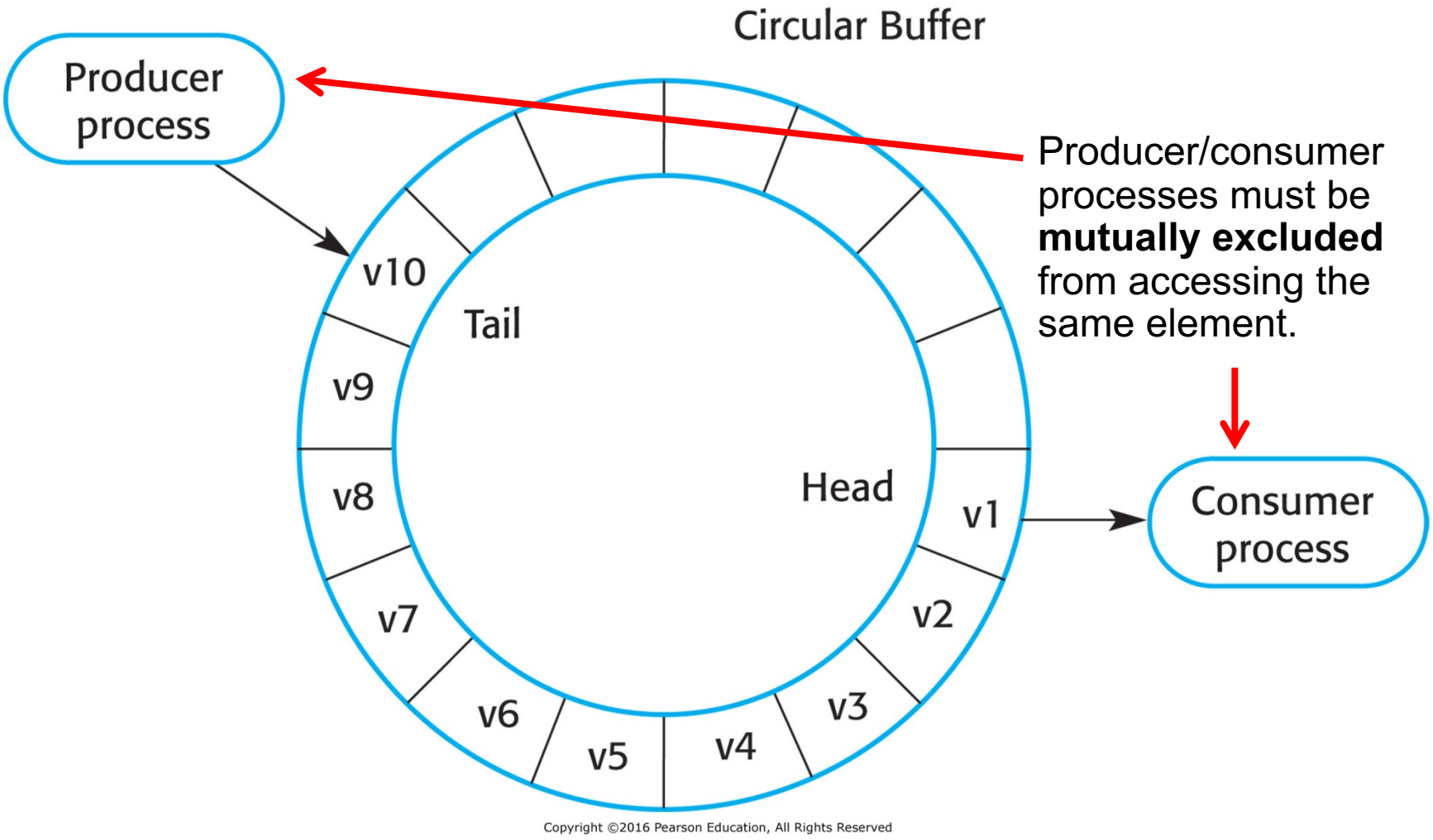
Producer processes collect data and add it to the buffer.

Producer/Consumer Processes

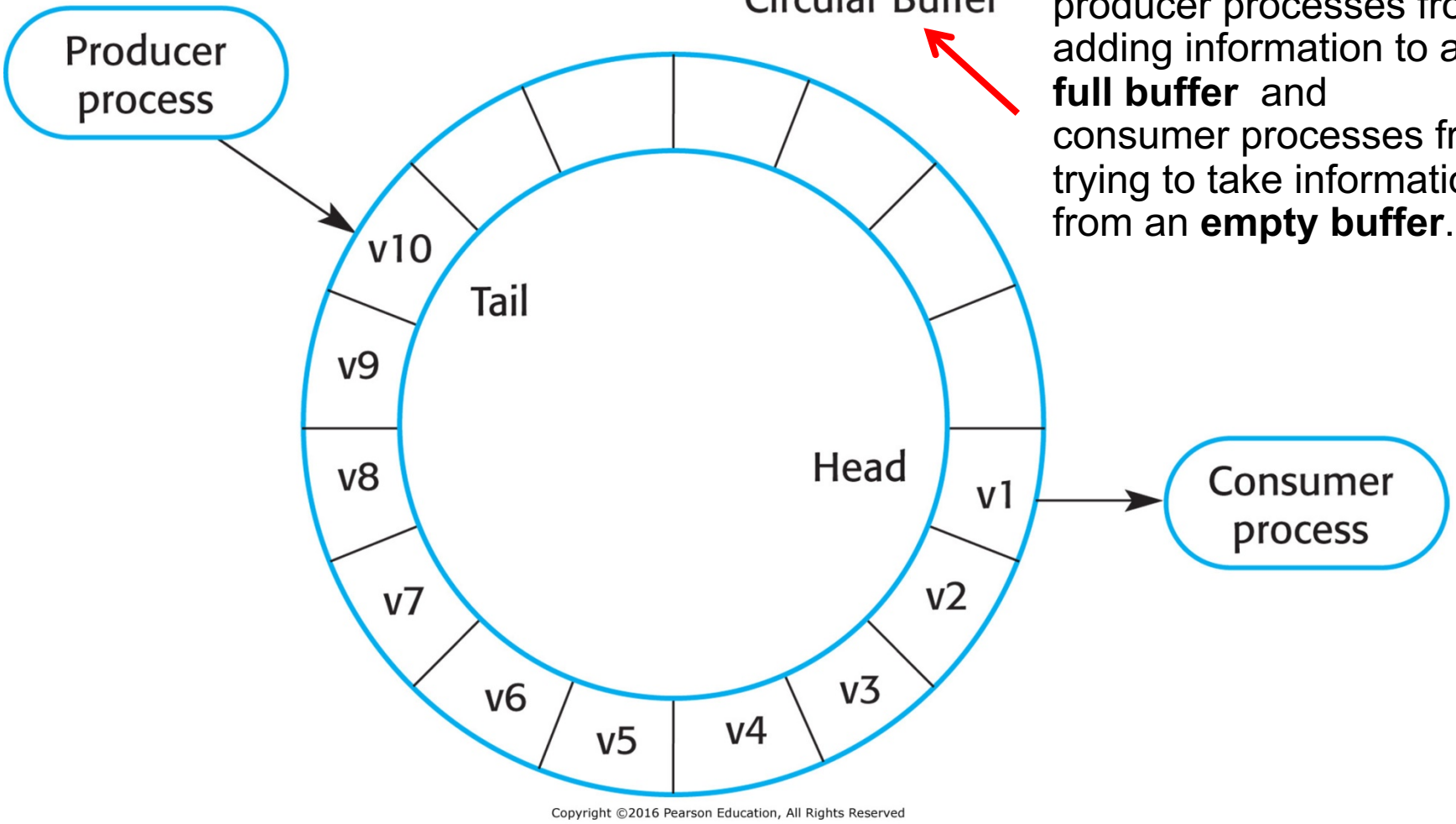


Consumer processes take data from the buffer and make elements available.

Producer/Consumer Processes



Producer/Consumer Processes



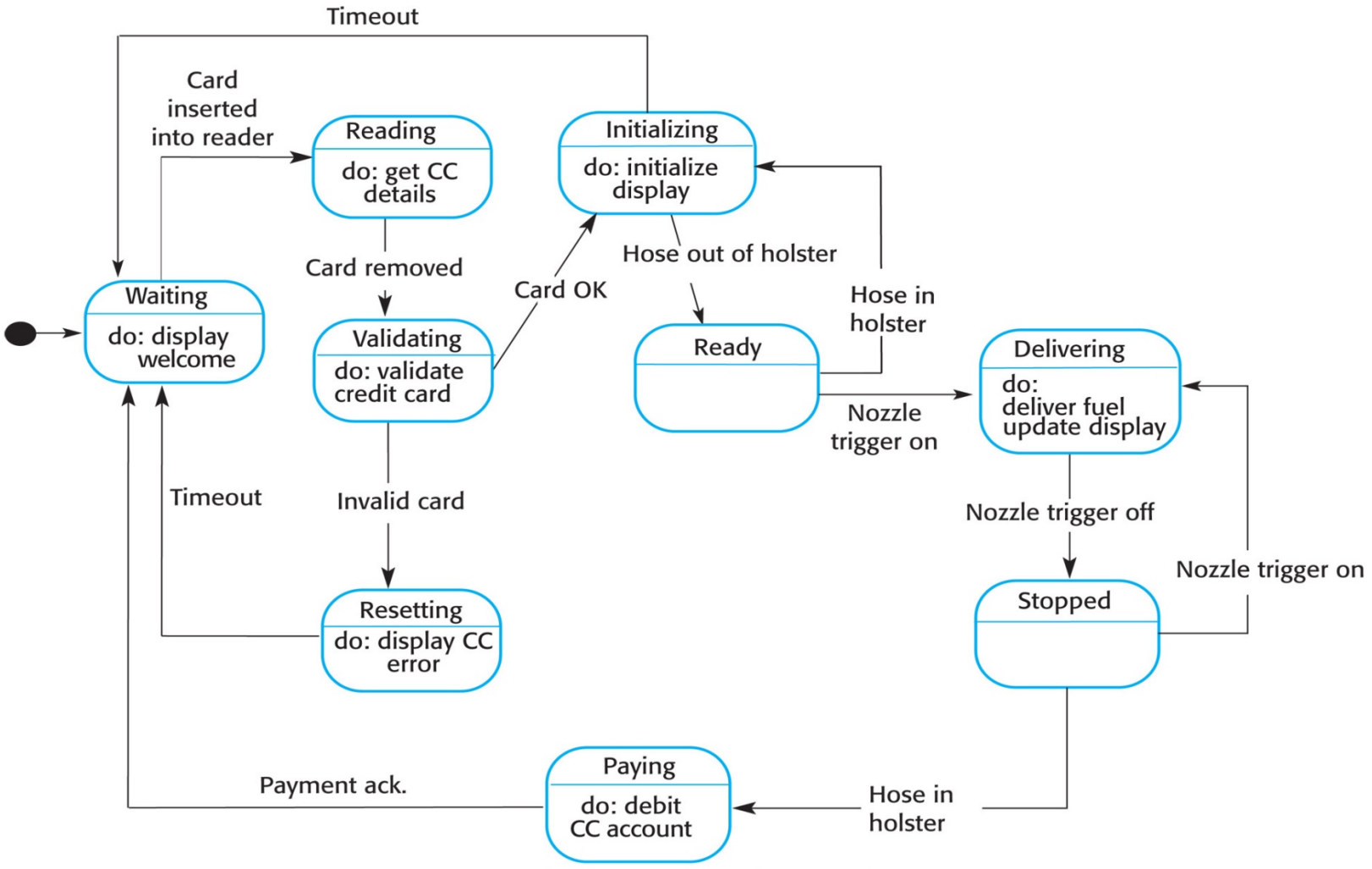
The buffer must stop producer processes from adding information to a **full buffer** and consumer processes from trying to take information from an **empty buffer**.

Figure 21.4 Producer/consumer processes sharing a circular buffer

Real-Time System Modeling

- The effect of a stimulus in a real-time system may trigger a transition from one state to another.
  - And hence state models are therefore often used to describe real-time systems.
  - **UML state diagrams** may be used to show the states and state transitions in a real-time system.

State Diagram/Model of a Gas Pump



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Distributed Systems:

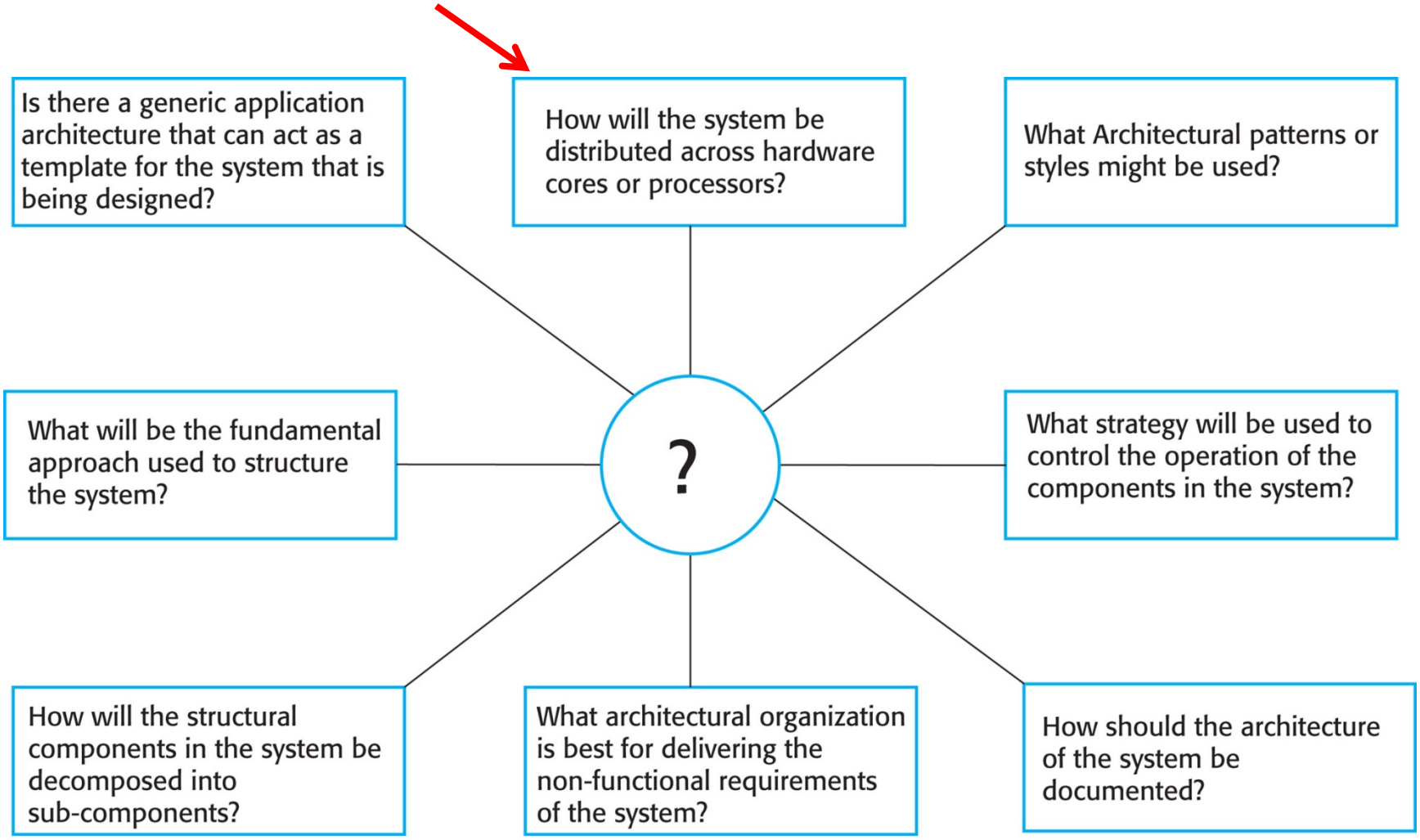
An Overview

Week 6: Architectural Design, Part 1

Edmund Yu, PhdD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Architectural Design Decisions



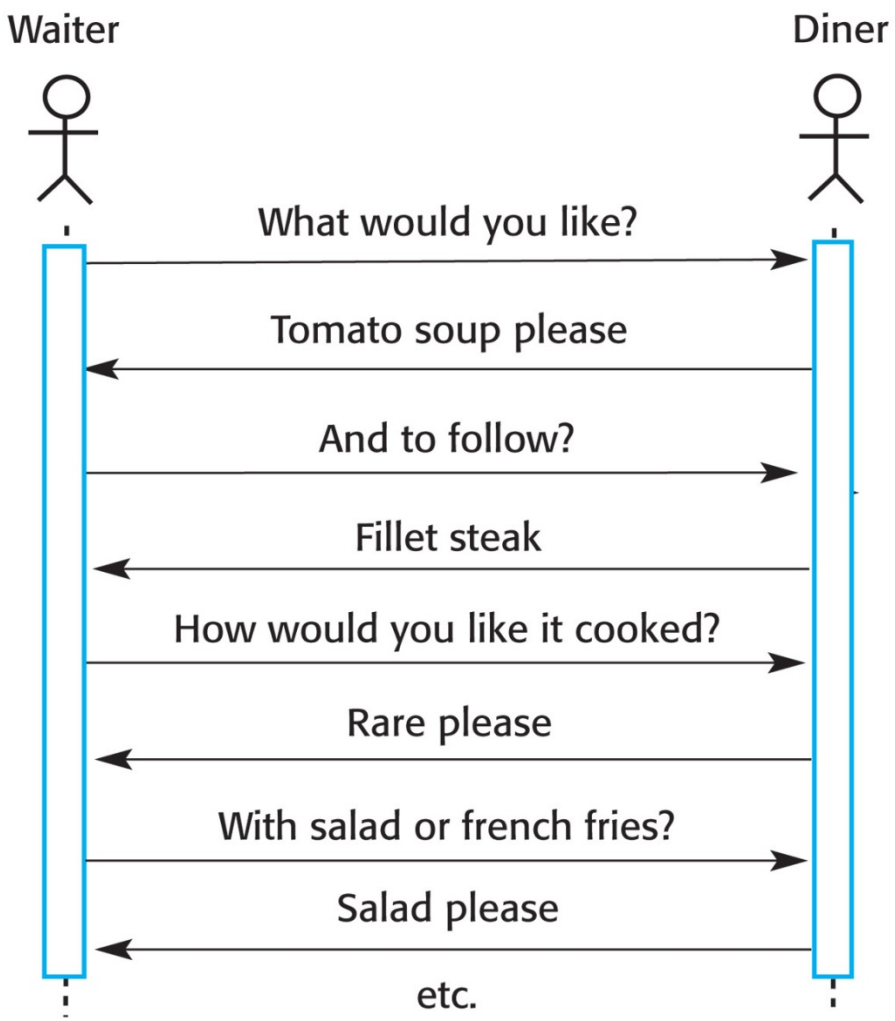
Distributed Systems

- Virtually all large computer-based systems are now distributed systems.
  - "a collection of independent computers that appears to the user as a single coherent system."
  - Information processing is distributed over a collection of independent computers rather than confined to a single machine.
- Distributed systems are also more complex than single-processor systems.
  - Different parts of the distributed system are independently managed.
  - No single authority in charge of the system, so top-down control is impossible.

Models of Interaction

- Two types of interaction between computers in a distributed system
  - **Procedural interaction**: One computer calls on a known service offered by another computer and waits for a response.
  - **Message-based interaction**: The sending computer sends information about what is required to another computer. There is no necessity to wait for a response.

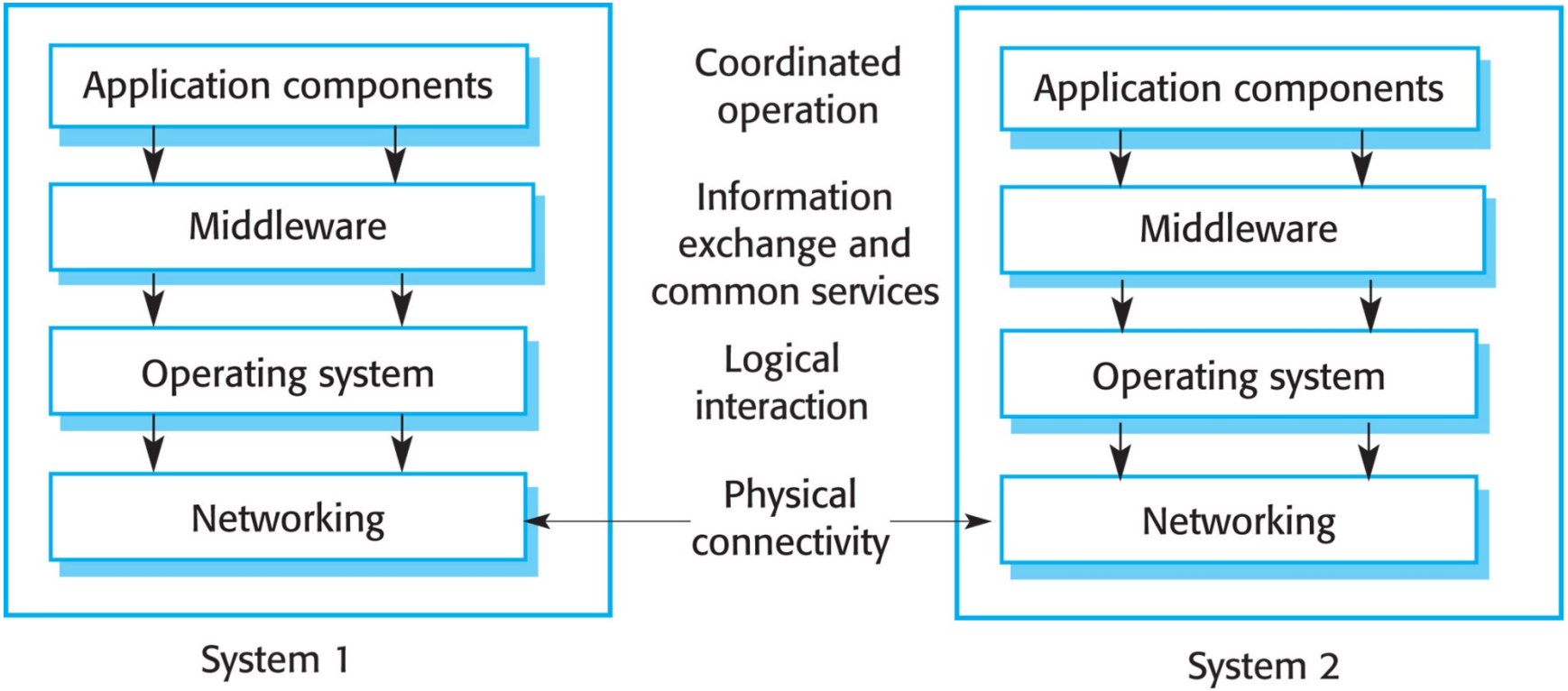
Procedural Interaction



Procedural Interaction: RPCs

- Procedural interaction in a distributed system is usually implemented using **remote procedure calls** (RPC).
- In a remote procedure call, one component calls another component as if it were a local procedure or method.
  - The **middleware** in the system intercepts this call and passes it to a remote component.
  - This carries out the required computation and, via the middleware, returns the result to the calling component.
- A problem with RPCs is:
  - The caller and the callee need to be available at the time of the communication.
  - They must know how to refer to each other.

Middleware in a Distributed System



Message-Based Interactions

- Message-based interaction normally involves one component creating a message that details the services required from another component.
  - Through the system middleware, this is sent to the receiving component.
  - The receiver parses the message, carries out the computations, and creates a message for the sending component with the required results.
- In a message-based approach, it is not necessary for the sender and receiver of the message to be aware of each other.
  - They can simply communicate with the **middleware**.

Message-Based Interaction

Message-based interaction between a waiter and the kitchen staff

```
<starter>
   <dish name = “soup” type = “tomato” />
   <dish name = “soup” type = “fish” />
   <dish name = “pigeon salad” />
</starter>
<main course>
   <dish name = “steak” type = “sirloin” cooking = “medium” />
   <dish name = “steak” type = “fillet” cooking = “rare” />
   <dish name = “sea bass”>
</main>
<accompaniment>
   <dish name = “french fries” portions = “2” />
   <dish name = “salad” portions = “1” />
</accompaniment>
```

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE