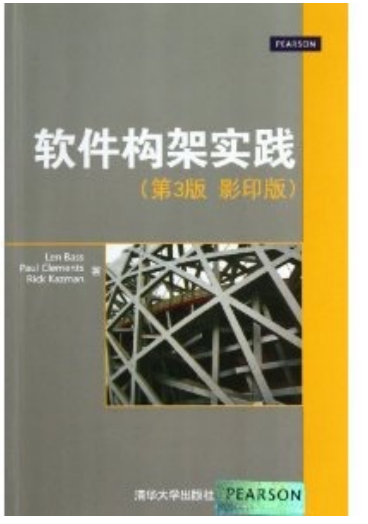
Lecture 1

Course Overview & Architectural Patterns

主讲教师： 王灿
Email: wcan@zju.edu.cn
TA: 李奇平 liqiping1991@gmail.com
Course FTP: ftp://sa:sa@10.214.51.13

Textbooks & References



- Textbook:
  - [BCK13] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 3 e, Pearson, 2013 清华大学出版社(2013)
- Reference:
  - [CBB+03] P. Clements, F. Bachmann, L. Bass, et al., Documenting Software Architectures: Views and Beyond, Addison-Wesley, 2003 清华大学出版社(2003)
  - [SG96] M. Shaw and D. Garlan, Software Architecture : Perspectives On an Emerging Discipline, Prentice Hall, 1996 清华大学出版社(1998), 科学出版社(2003)

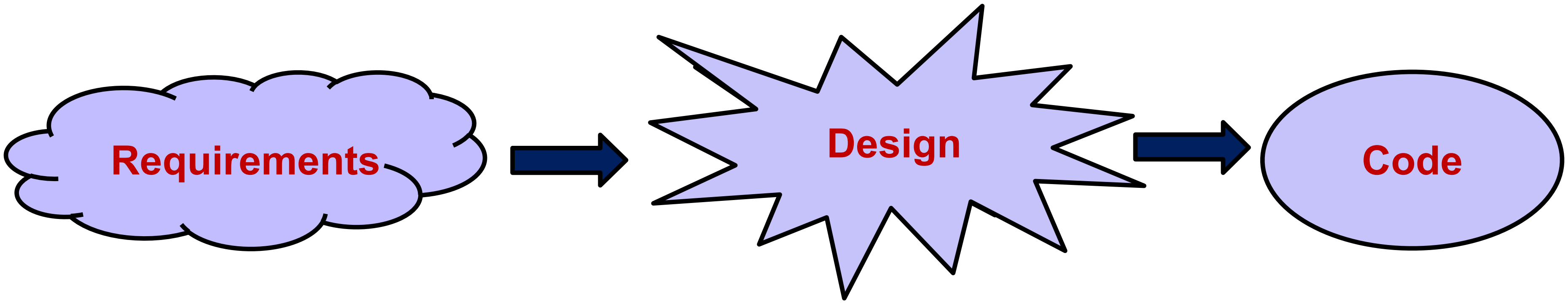
http://www.amazon.cn/%E5%9B%BE%E4%B9%A6/dp/B00BMK4FWG

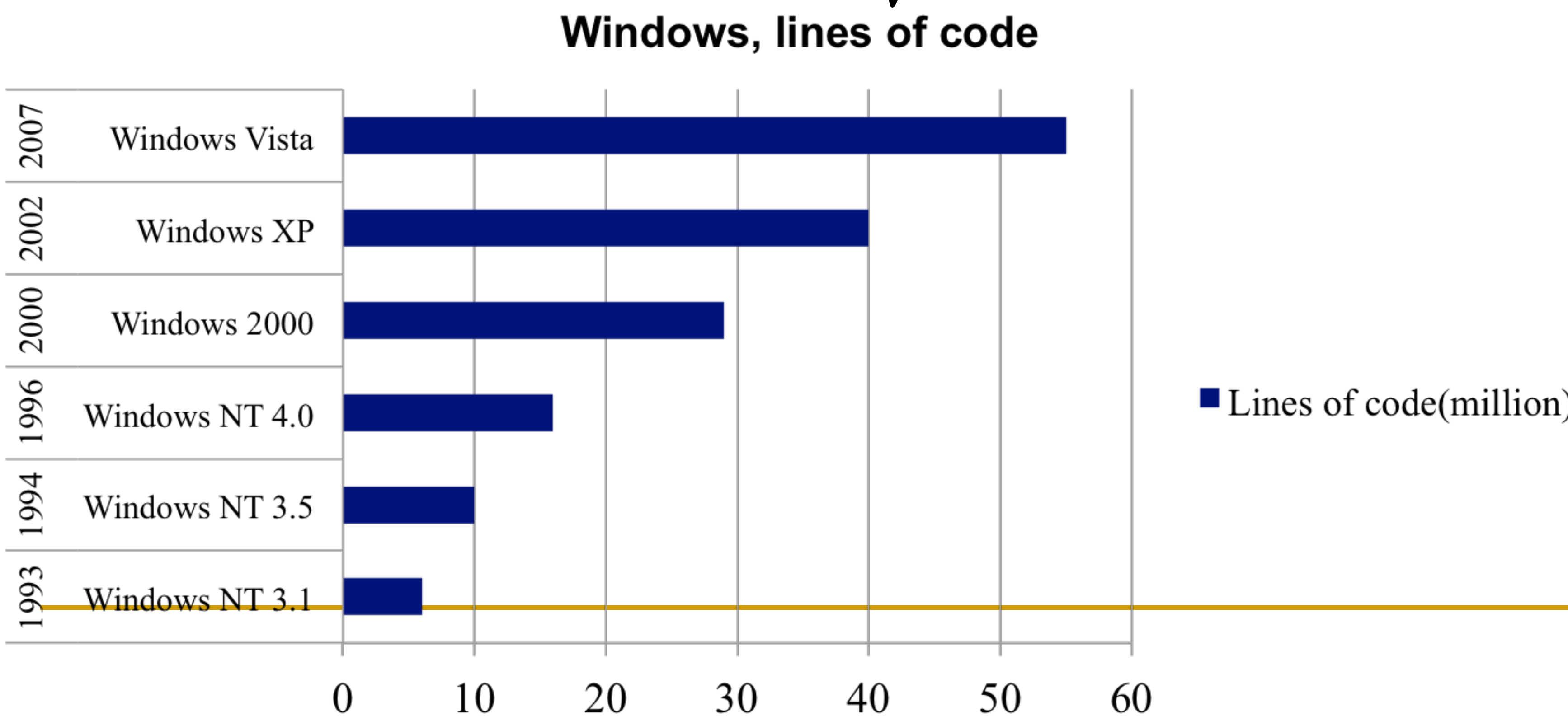
Grading Policy

- Assignments (30%)
  - 2~4 written assignments
- Final exam (70%)

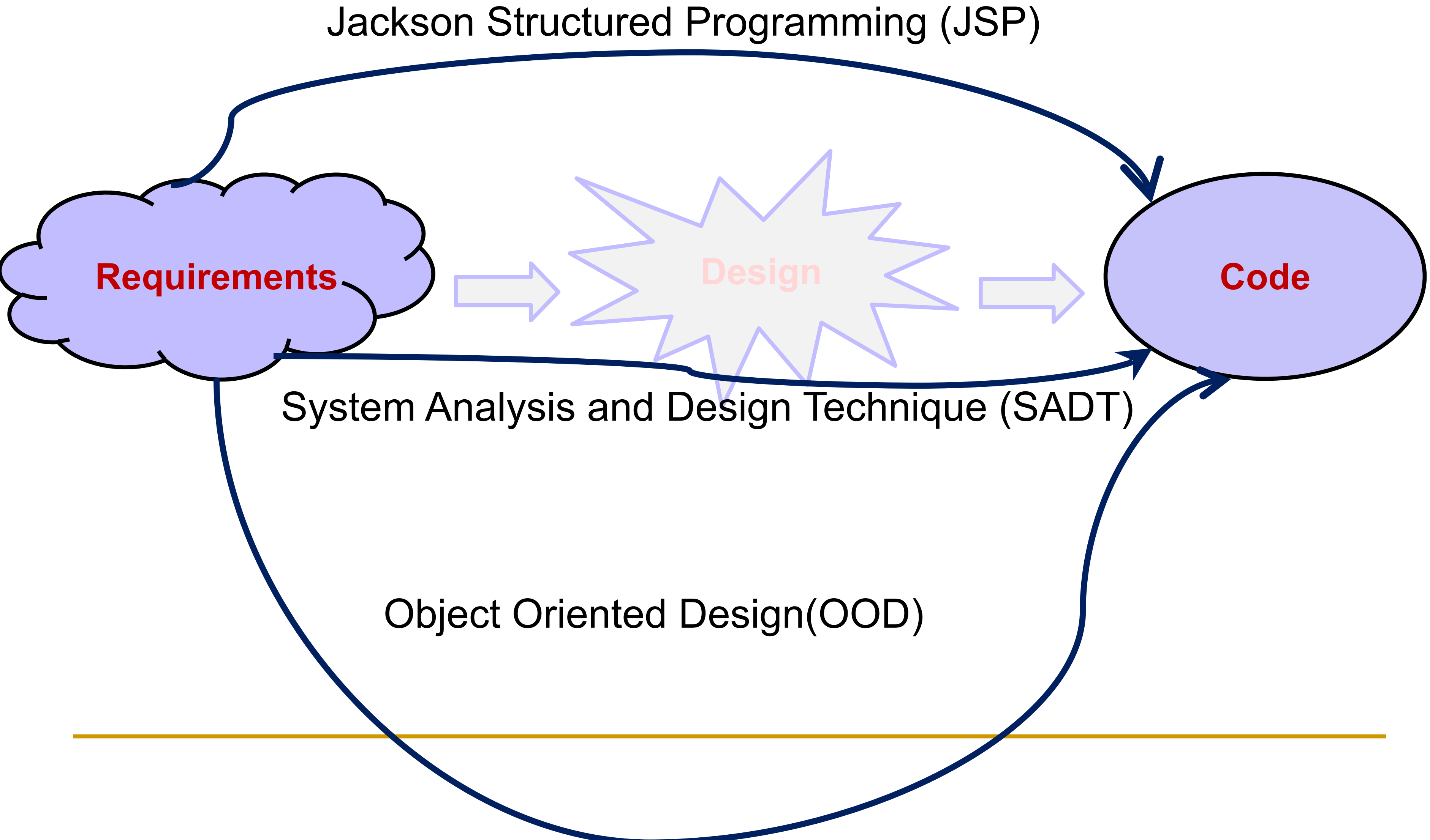
Challenge

Ever increasing size and complexity of software systems





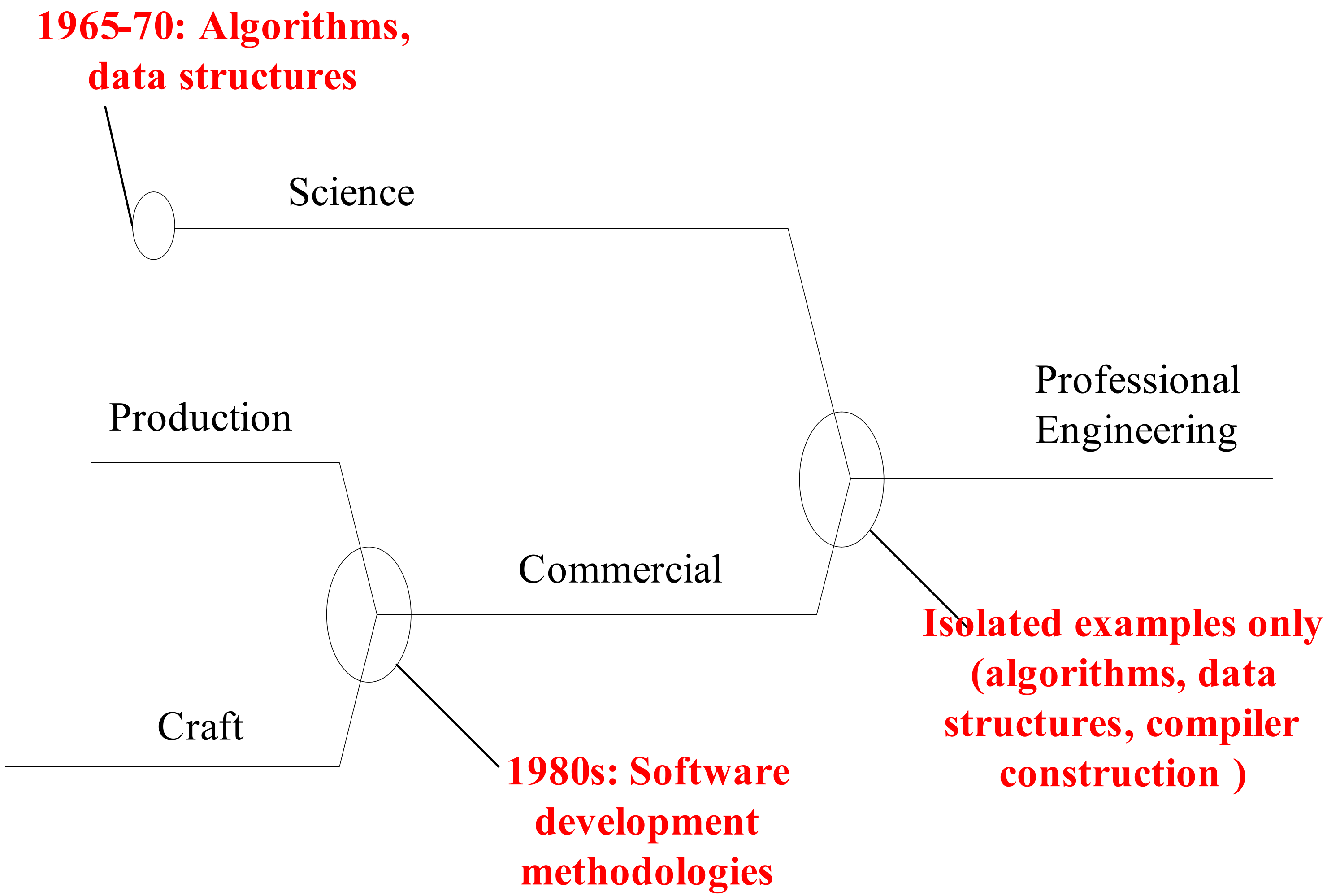
Software Development Methods



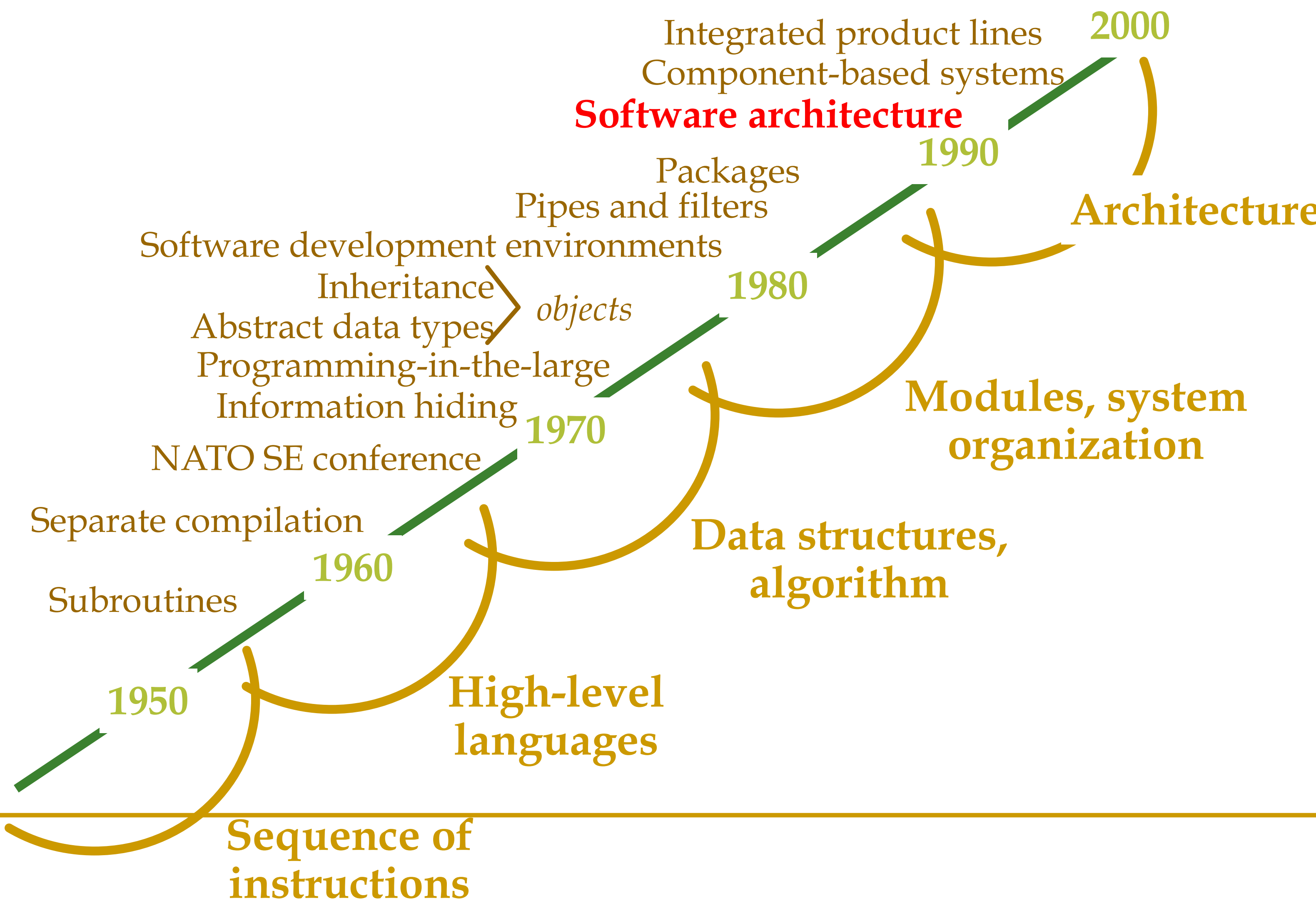
Design Levels

- Computer hardware design levels
  - Circuit
  - Logic design level
  - Programming level
  - PMS level
- Software design levels
  - Executable
  - Code
  - *Architecture*

Evolution of Software Engineering



Why are We Now in Software Architecture



What Is Software Architecture?

- [BCK13] The software architecture of a system is the ***set of structures*** needed to reason about the system, which comprise software elements, relations among them, and properties of both.

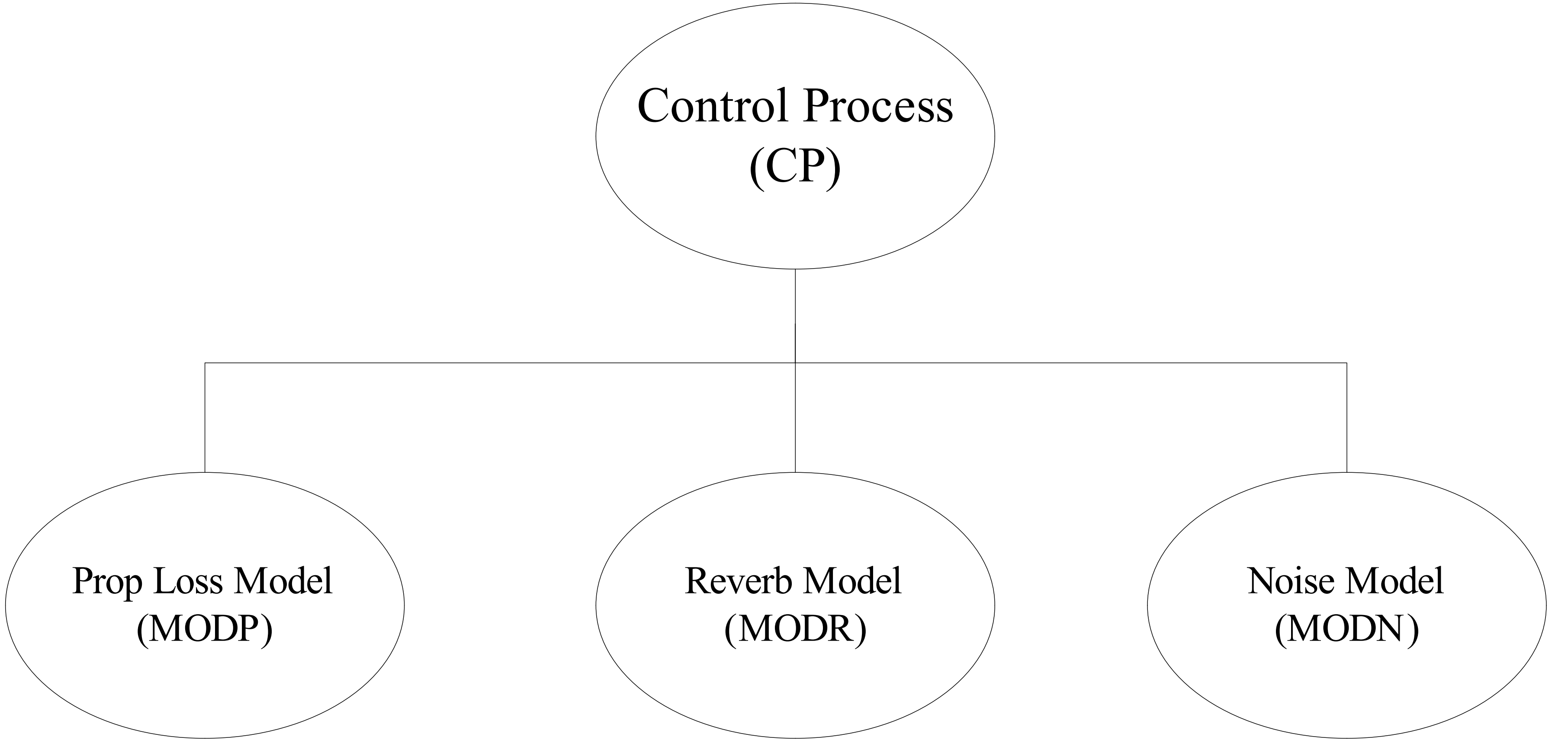
Implications (1)

- Architecture is a set of software structures
  - No single structure holds claim to being the architecture
- Three important categories of architecture:
  - Module structures
  - Component-and-connector structures
  - Allocation structures

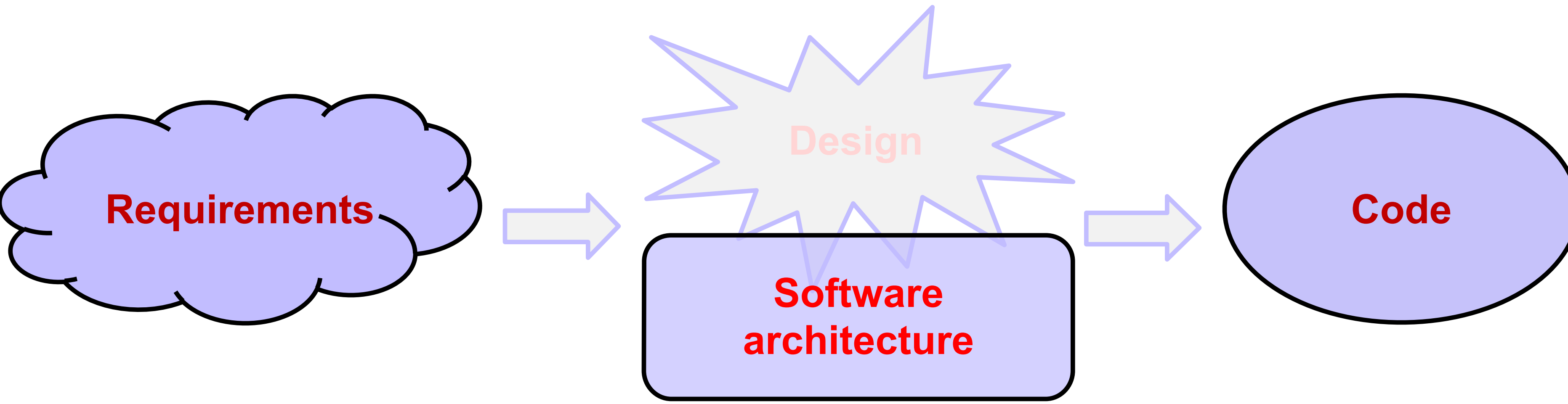
Implications (2)

- Architecture is an abstraction
  - It suppresses the internal information of the elements
- Every Software System Has a Software Architecture
  - Software architecture VS. the *representation*
- Architecture includes behavior
- Not all architectures are good architectures

Is This a Software Architecture?



The Role of Software Architecture



- Composition of large-scale components
- System-level abstractions
- Reuse of system-level design idioms
- At this moment, software architecture can be simply defined as the computational components in a system and interactions among those components.

Structural Issues

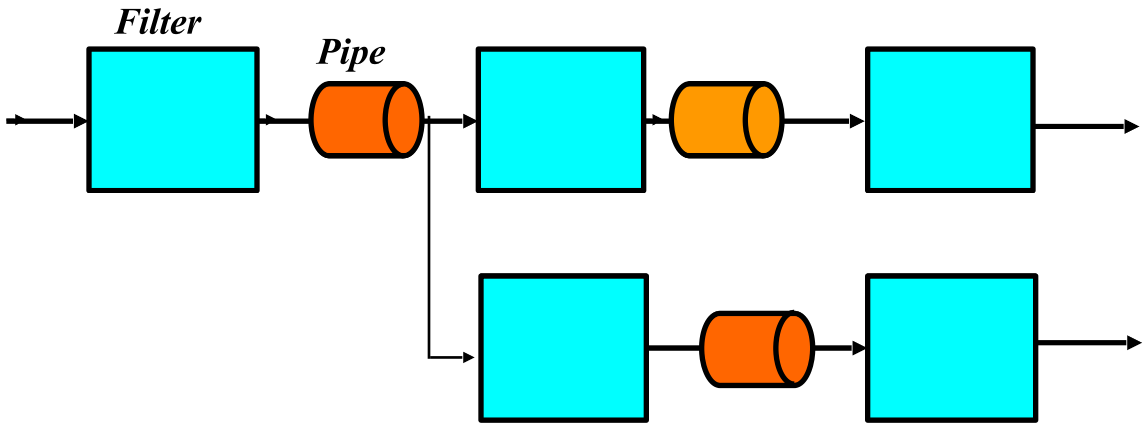
- Structural issues are the major concerns of software architecture
  - Organization of the systems as in the composition of components
  - Assignment of functionality to design elements
  - Composition of design elements

  - Global control structures
  - Protocols for communication, synchronization and access
  - Physical distribution
  - Scaling and performance
  - Dimensions of evolutions

Some Common Architectural Patterns

- An architectural pattern is a description of elements and relation types together with a set of constraints on how they may be used.
- We approach specific architectural styles by following features:
  - The types of elements
  - The underlying computational model
  - Advantages and disadvantages
  - Some common examples of its use

Pipes and Filters: Model (1)

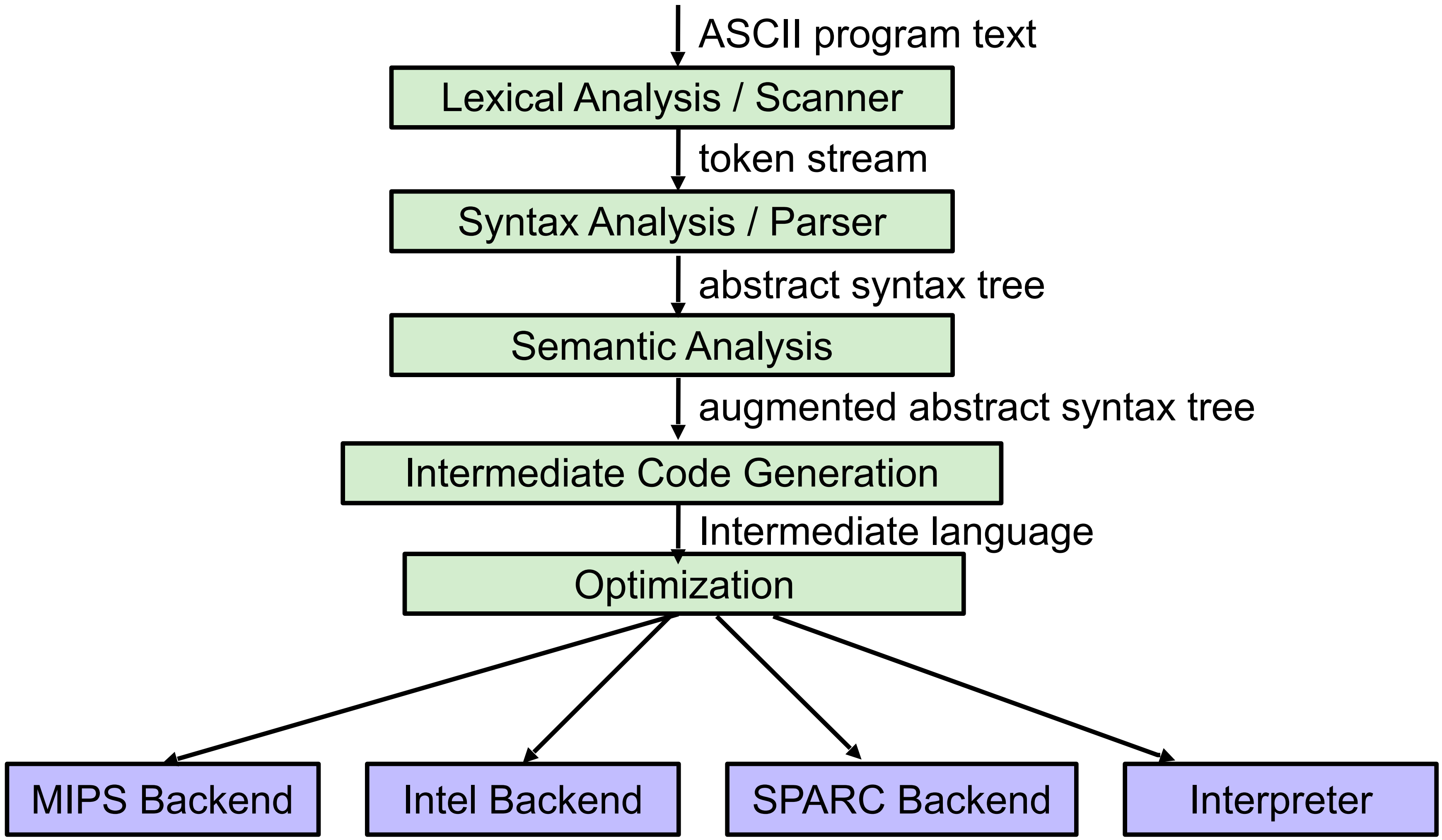


- A structure for systems that process a stream of data
  - Each processing step is encapsulated in a filter component
  - Data is passed through pipes between adjacent filters (single direction)

Pipes and Filters: Model (2)

- Elements: filters and pipers
  - Filter components are the processing units of the pipeline
    - Filters must be independent entities
  - Pipes denote the connections between filters
- Examples
  - Compilers
  - Unix shell programs (pipe)
  - A degenerate case: when each filter processes all of its input data as a single entity (a batch sequential system)

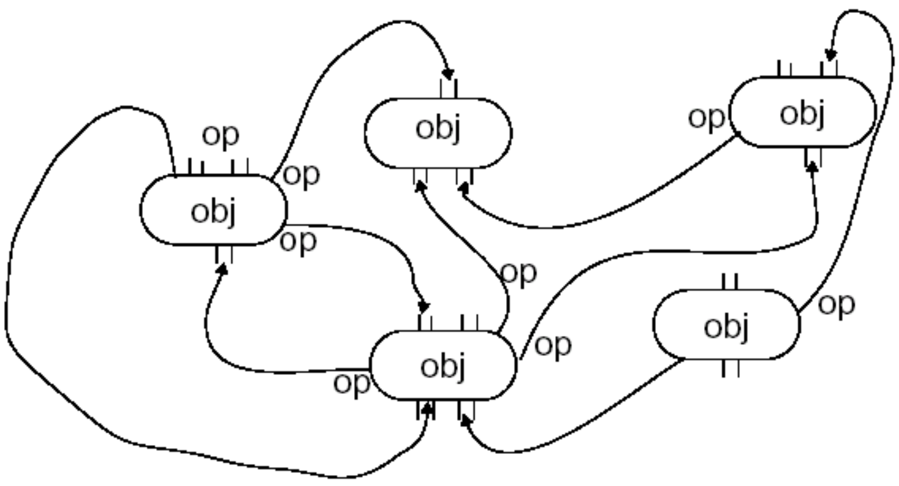
Pipes and Filters: An Example



Pipes and Filters: Pros and Cons

- Pros:
  - **Clean design**
  - Flexibility by filter exchange
  - Flexibility by recombination
  - No intermediate files necessary
  - Reuse of Filter components
  - Rapid prototyping of pipelines
  - Efficiency by parallel processing
- Cons:
  - **Sharing state information is expensive or inflexible**
  - Error handling
  - Data transformation overhead
  - Efficiency gain by parallel processing is often an illusion

Data Abstraction/Object Oriented



- Object Oriented: Model
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Reuse/Maintenance: encapsulation/abstraction promotes separation of concerns

Object Oriented: Pros

- Problem decomposition
  - Natural correspondence with real-world entities
  - Inheritance allows shared definitions
- Maintenance and reuse
  - Decreased coupling (change propagation)
  - Increased reusability (especially frameworks)
- Protection of internal representations
  - Encapsulation allows data/state integrity to be preserved

Object Oriented: Cons

- Design is harder: forces more up-front brain-work
- Inheritance: often non-intuitive
- Maintenance: need additional structure—one level of objects is too flat

- Side effects: many objects can access a single resource
- ***Identity: need to know (import) an object/ method's name (explicit invocation)***

Event Based Systems: Model

- Components: objects or processes
  - Interface defines allowable incoming events
  - Interface defines allowable outgoing events
- Connections: event-procedure bindings
  - Procedures are registered with events
  - Components interact by “announcing” events
  - Upon receiving an event, its associated procedures are implicitly invoked
  - ***Order of invocation is non-deterministic***

Event Based Systems: Pros

- Problem decomposition
  - Computation and coordination are separate: *objects are more independent*
- System maintenance and reuse
  - No hard-wired (static) name dependencies
  - Eases system evolution: use new objects simply by registering them
  - eases integration
- Performance
  - Invocations can be parallelized

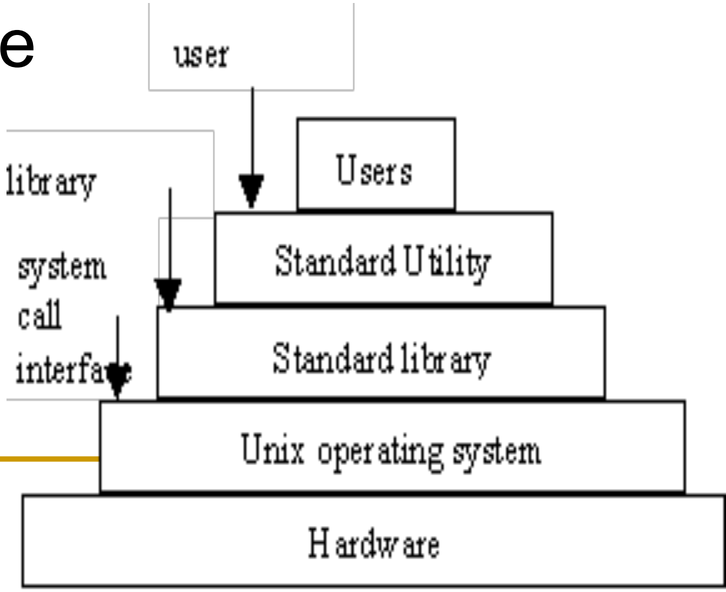
Event Based Systems: Cons

- Problem decomposition
  - ***No control over order of invocation***
  - Correctness difficult to ensure
  - Exchange of data
- System Maintenance and Reuse
  - Requires a centralized “yellow pages” of who knows what: events, registrations, dispatch policies
- Performance
  - Indirection/communication imply some performance penalty

Layered Systems: Model

- A hierarchical organization, with each layer
  - Providing service to the layer above it
  - Serving as a client to the layer below
- A "virtual machine"
  - Encapsulation of layer implementations
- The connectors are defined by the protocols that determine how the layer will interact
  - Topological constraints: limiting interactions to adjacent layers

Unix:



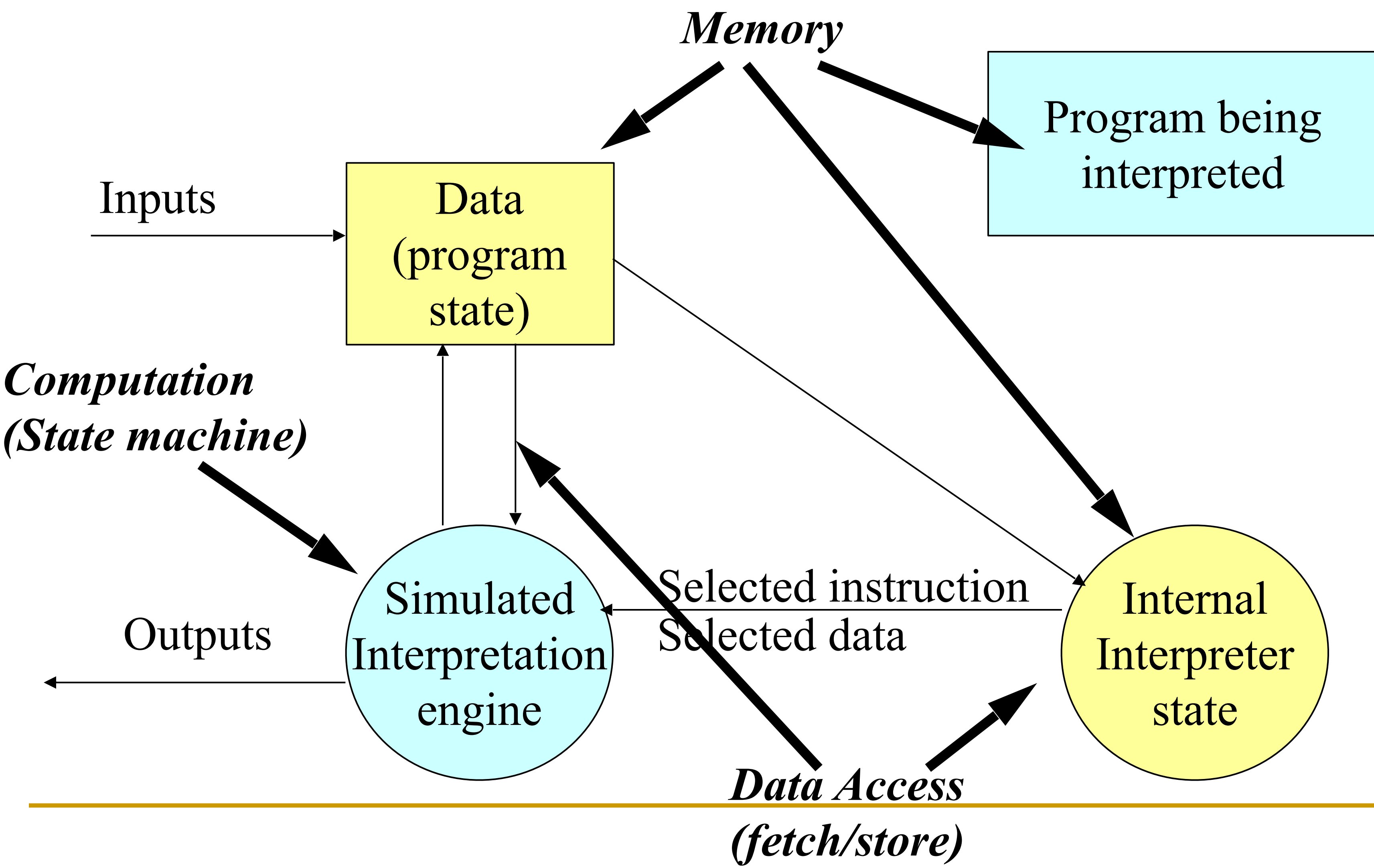
Layered Systems: Pros

- Support designs based upon increasing levels of abstraction
- Aids in portability
  - each layer is an abstract “virtual machine”
- Aids in modifiability
  - each layer interacts with at most 2 others
- Support reuse
  - Maintaining the same interface to the adjacent layers

Layered Systems: Cons

- Performance penalty
- May be the wrong model
  - e.g. user may need to control low-level functionalities
- Finding the right levels of abstractions is difficult
  - Esp. in a standardized layered model. e.g. ISO OSI reference model VS. TCP/IP
  - If the abstractions are wrong, layers need to be bridged
    - “Layer bridging” often ruins the model

Interpreters: Model



Interpreters: Model (2)

- Execution engine in software
  - A virtual machine to close the gap between the computing engine expected by the semantics of the program and the computing engine available in hardware.
- Data:
  - Program being interpreted
  - Program state data
  - Interpreter state data
- Control:
  - Interpretation engine state
  - Simulated system state

Interpreters: Pros and Cons

- Advantages
  - Functionality:
    - can simulate non-native functionality
  - Testing:
    - can simulate “disaster” modes (e.g. for safety-critical applications)
  - Flexibility:
    - very general-purpose tool
- Disadvantages
  - Efficiency:
    - much, much slower than hardware
    - much slower than compiled system
  - Testing:
    - additional layer of software to be verified

Assignment

- Read Chapter 1 & 3 of the textbook.
- Read the story of the *Vasa* (Wikipedia entry: http://en.wikipedia.org/wiki/Vasa_(ship))