Principles of Software Architecture

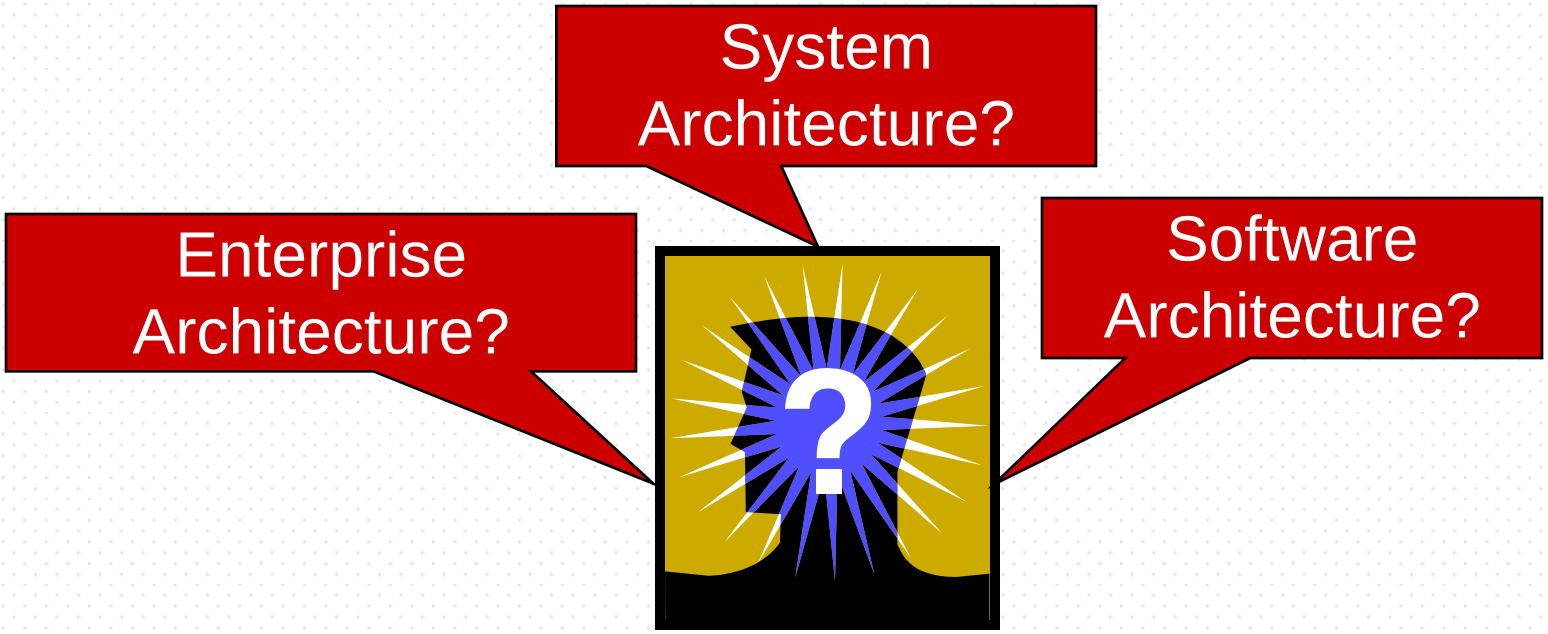
Part 1 continued:

Basic Software Architecture Concepts

Objectives of this Lecture

- Clarify common architecture terminology in use today.
  - enterprise, system, and software architecture
  - views, styles, patterns
- Discuss the role of a software architect.
- Define basic architectural concepts.

Related Terminology



Which do we have!? Which do we need!? What is architecture?

Enterprise Architectures

- *Enterprise architecture* is a means for describing business structures and processes that connect business structures.
  - Describes flows of information and activities between entities in an enterprise.
  - Enterprise architectures may or may not be supported by computer systems.
  - *Software design is not addressed explicitly.*

John A. Zachman, *"A Framework for Information Systems Architecture",* IBM Systems Journal, Vol. 26, No 3, 1987.

System Architecture

- *Systems Engineering* is a design and management discipline for designing and building large, complex, and inter-disciplinary systems*.
  - Describes the elements and interactions of a complete system, and their contribution toward the goal of the system.
  - Includes identifying and characterizing hardware AND software elements, but *not the substructure of the elements.*

*Rechtin, E. *Systems Architecting: Creating and Building Complex Systems*. Prentice-Hall, 1991.

Architecture versus Design

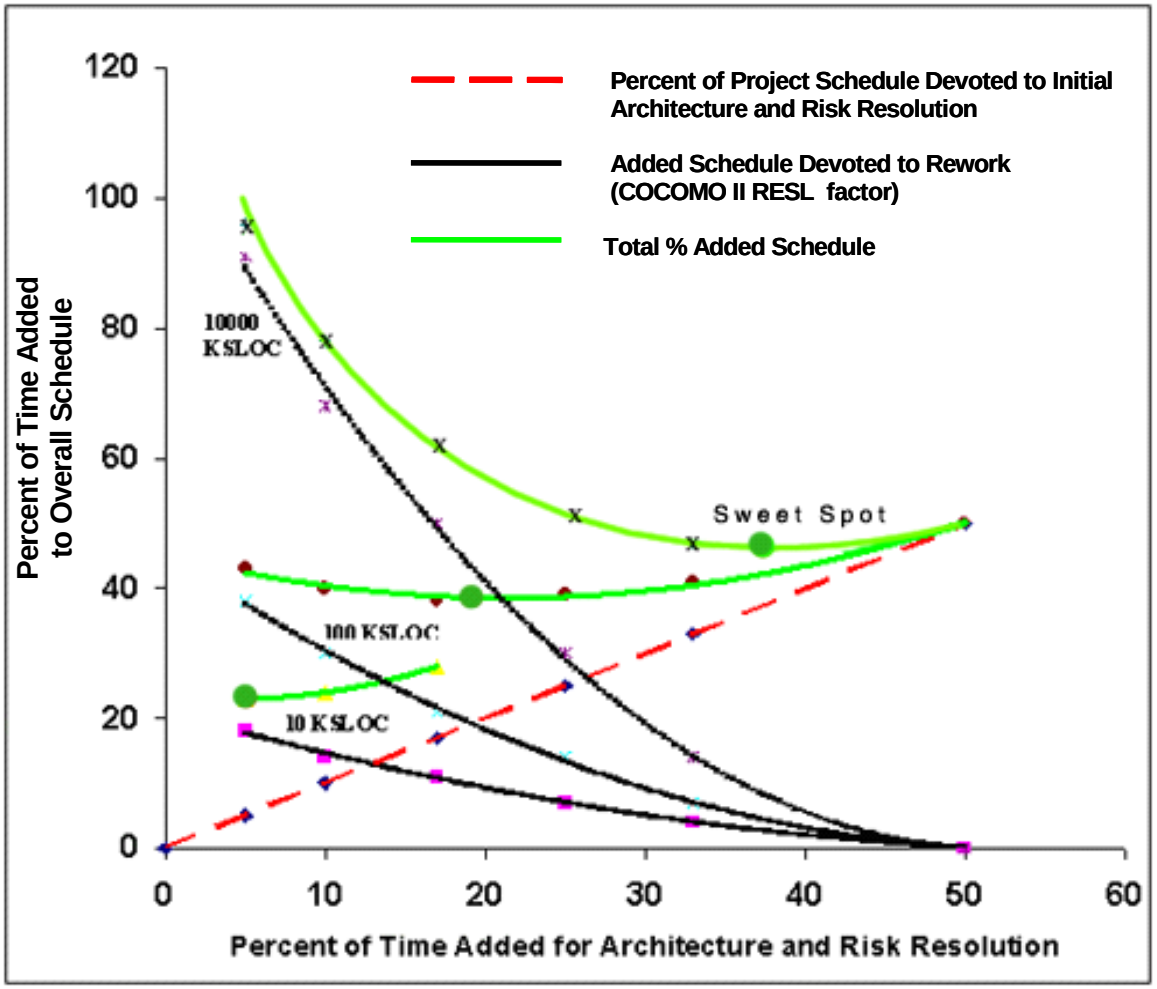
- Architecture is design, but not all design is architectural.
  - Architects intentionally limit their focus and avoid the details of how elements do what they do.
    - Detailed designs and implementation details are left to downstream engineers/experts.
  - Downstream engineers are expected to respect the architecture to ensure that properties promised by the architect are present in the product.

When Are We Done Architecting?

- The line between architecture and detailed element design is blurry.
  - The overall system, enterprise, software properties must be quantified and documented.
  - Downstream designers are able to understand and adhere to the constraints set forth by the architecture representation.
  - Rationale for architectural decisions are understood.

How Much Architecture is Enough?

Source: "Using Risk to Balance Agility and Discipline: A Quantitative Analysis," Barry Boehm.



Key Points

- Projects have a "sweet spot" of investment into "front-end" activities, including architectural design
  - Too little => more rework later
  - Too much => wasted effort
- Amount of depends on size of system
  - Small (10 KLoC) => 5-10%
  - Medium (100 KLoC) => 20-25%
  - Large (10 MLoC) => 35-40%

Role of a Software Architect - 1

- Architects are generally concerned with
  - refining and clarifying expected product properties
  - ensuring that necessary properties are present in the finished product
  - partitioning and structure
  - constraining downstream designers and implementers

Role of a Software Architect - 2

- System, software, and/or enterprise design is typically done by teams of architects. The software architect
  - coordinates and/or leads the design, documentation, and evaluation of the software architecture
  - participates in larger system/enterprise design

Role of a Software Architect - 3

- Provide technical leadership by
  - guiding the exploration and definition of stakeholder needs
  - guiding prototyping efforts to understand technology or stakeholder requirements
  - overseeing detailed element design and construction
  - guiding element and/or product test
  - keeping abreast of new/emerging technology
  - guiding lifecycle maintenance efforts

Concepts of Software Architecture

- Views
  - Why they are necessary
  - What distinguishes different kinds of views
- Styles & Patterns
  - Why they are necessary
  - What distinguishes them
- Module Views
  - Elements
  - What it's good for
  - Layered styles
- Component and Connector Views
  - Elements
  - What it's good for
  - Styles overview

Recall our definition

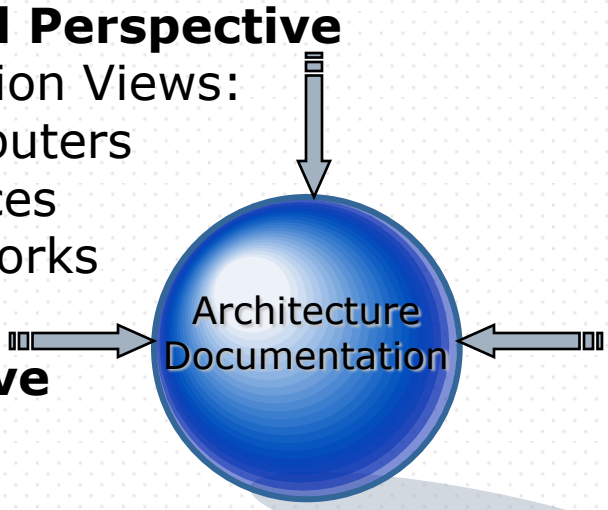
The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them and properties of both.

What is a Structure? – 1

- Software architecture is an abstraction of the structures that comprise the software that is part of a software-intensive system.
- Structures are implementation-oriented.
- Software has many structures, such as
  - code
  - processes/threads
  - files

What Is A Structure? – 2

- A representation of a structure is usually called *a view* of the system.
- Software architecture documentation is comprised of a collection of views.

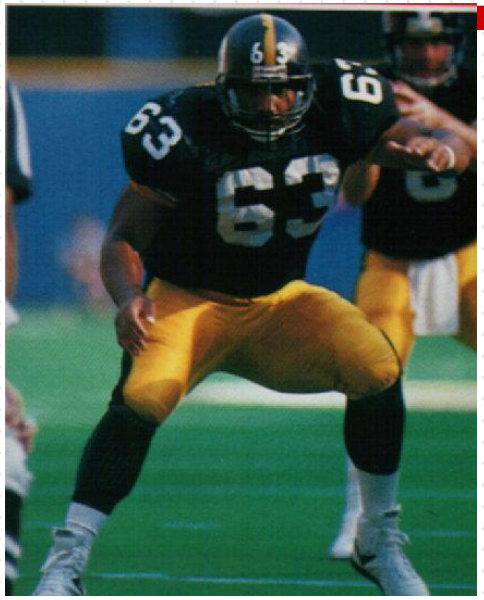


Physical Perspective
- Allocation Views:
  - computers
  - devices
  - networks

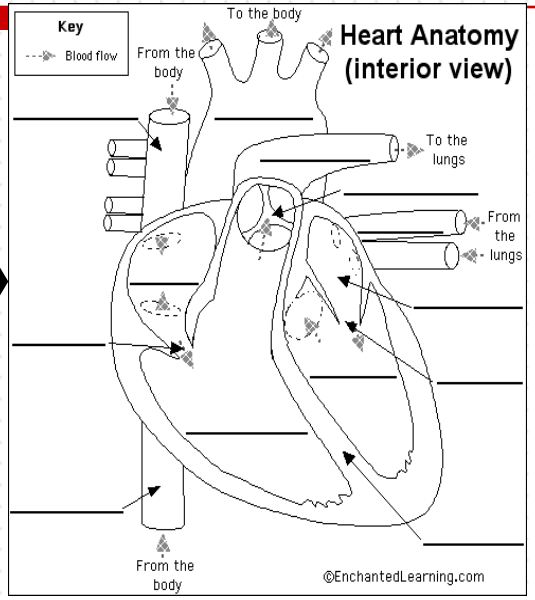
Static Perspective
- Module Views:
  - classes
  - functions
  - interfaces

Runtime Perspective
- Component-and-connector Views:
  - processes
  - sequence diagrams
  - dataflow

Structures and Views – 1



A human body is comprised of multiple *structures.*

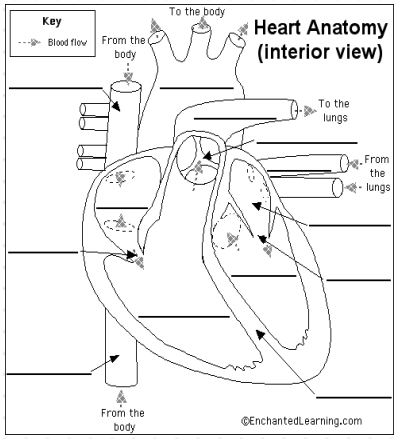


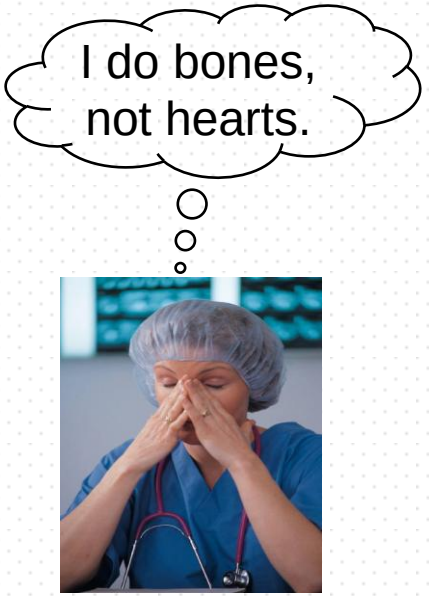
a *static view* of one human *structure*

a *dynamic view* of that *structure*

One body has many structures, and those structures have many views. So it is with software…

Structures And Views – 2





These *views* are needed by the cardiologist…

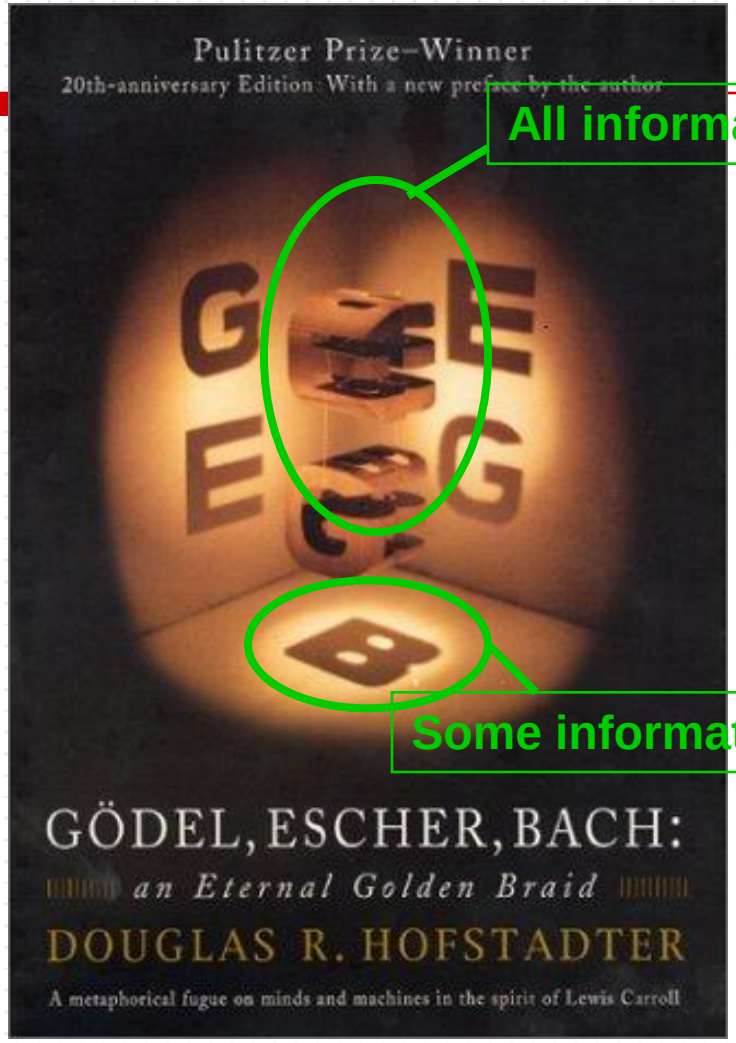
…but will these *views* work for the orthopedist?

Different stakeholders are interested in different structures.

Views must represent the structures that the stakeholders are interested in. *So it is with software…*

Views

- A view is a representation of a set of system elements and the relations associated with them.
- Not *all* system elements
  -- *some* of them.
- A view selects *element types* and *relation types* of interest, and shows those.



What Views Are Available? -1

- Plenty! Too many!
- An architect needs a way to choose the useful ones
- One thing that would help is to organize the views into *broad categories*

What Views Are Available? -2

- An architect must consider the system in three ways:
  1. How is it structured as a set of code units?
  2. How is it structured as a set of elements that have run-time behavior and interactions?
  3. How does it relate to non-software structures in its environment?

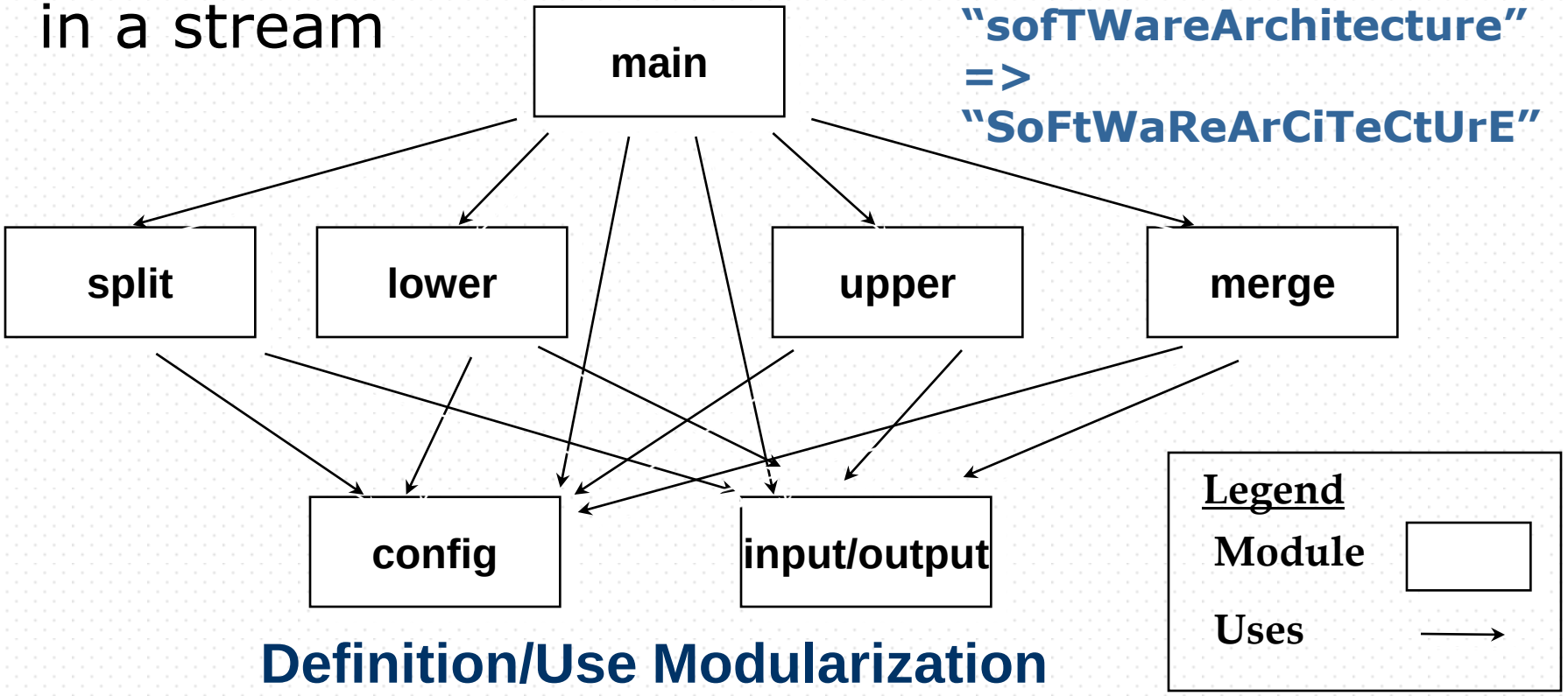
What Views Are Available? -3

This suggests looking for 3 kinds of views:

1. How it is structured as a set of code units
   module views (*Module Views*)
2. How it is structured as a set of elements that have run-time behavior and interactions
   component-and-connector views (*C&C Views*)
3. How it relate to non-software structures in its environment?
   allocation views (*Allocation Views*)

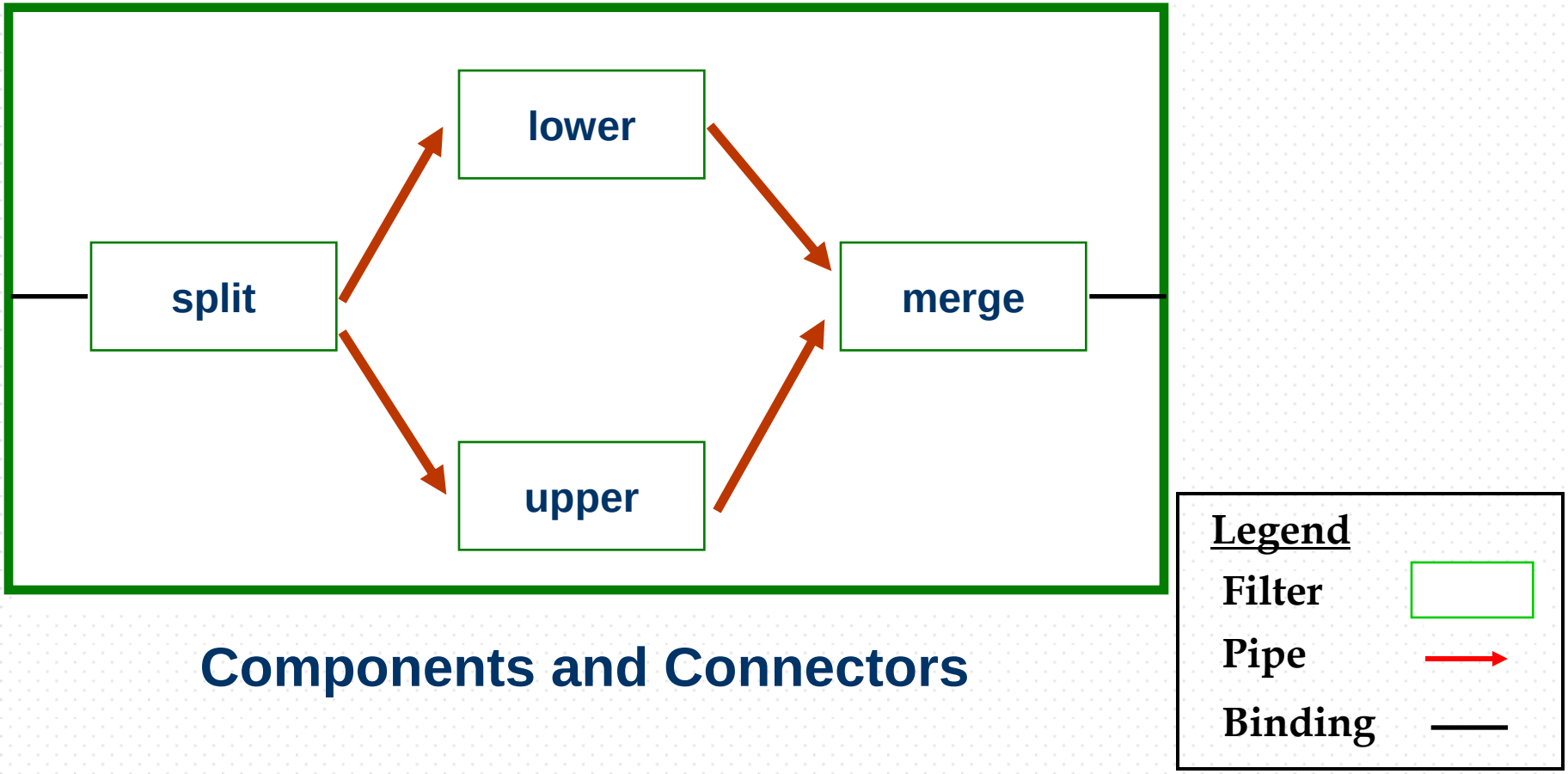
Example: Alternating Characters Code (Module) View

Produce alternating case of characters in a stream



Definition/Use Modularization

Example continued: Run-time (C&C) View



Architectural Styles -1

- This still leaves us with enormous latitude.
- Within each kind of view, recurring forms have been widely observed in different systems.
- These forms, or patterns, are worth capturing because they have known properties and can be re-used.
- We call these styles:

  An architectural style is a set of element and relation types, together with a set of constraints on how they can be used.

Architectural Styles -2

- When experts work on a problem it is unusual for them to invent new solutions.
  - They recall a similar problem and reuse the essence of the solution.
- Abstracting and codifying specific problem-solution pairs and distilling the common features leads to *styles* or *patterns*.
- Styles can help designers and builders leverage the collective experience of a community of designers and builders.

Architectural Styles -3

- Styles are widely known in the literature. They represent "tools" in the architect's "bag of tricks."
  - Note that styles exist independent of any system.
  - Two different systems can use the same style.
- A system is likely to be composed of more than one style
  - Several styles might be combined to form a hybrid.
  - Different parts of a system in different styles.
  - Different styles at different levels of abstraction.

Preliminary Summary: Perspectives, Views, and Styles

- *Perspectives* reflect the three broad ways an architect must consider a system:
  - Static: units of implementation (module views)
  - Dynamic: run-time units (C&C views)
  - Physical: relation to non-software structures (allocation views)
- Even within a perspective, many choices remain: how elements are restricted, how they are related to each other, how they are used or configured. These choices, when made, yield *styles*.

Module Views

- Elements: Modules. A module is a code unit that implements a set of responsibilities.
- Relations: Relations among modules include
  - A is part of B. This defines a part-whole relation among modules
  - A depends on B. This defines a dependency relation among modules.
  - A is a B. This defines specialization and generalization relations among modules.

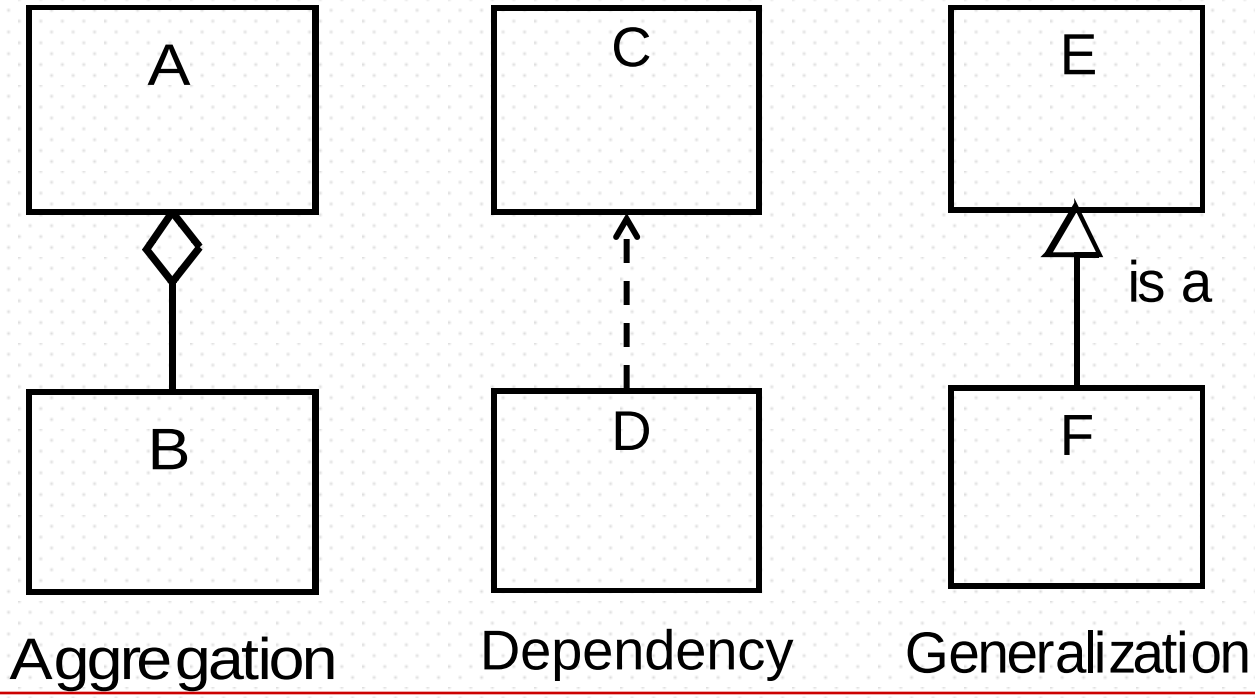
What are Module Views Used For?

- Construction: These are the blueprints for the code. Modules are assigned to teams for implementation. Modules are often the unit for subsequent design (e.g., of interfaces).

- Analysis: Traceability and impact analysis rely on implementation units. Project management, budgeting, planning, and tracking often use modules.

Notations for Module Views

- Informal: box-and-line, with nesting
- UML: Class diagrams



Module Styles

- Decomposition Style
  - Hierarchical decomposition of modules
  - Supports concurrent development
- Generalization Style
  - Specialization hierarchy
  - Supports reuse; managing large numbers of definitions
- Layered Style
  - Virtual machines
  - Supports portability, reuse

Decomposition Style

- Elements: Modules
- Relations:
  - Is part of. Criteria for decomposition vary:
    - achievement of modifiability
    - build vs. buy
    - product lines
- What it's for:
  - A starting point. Many architects assign functions to modules as a prelude to detailed design.
  - Change/impact analysis
  - Basis for work assignments

Decomposition Style: Example

- Behavior-Hiding Module
  - Function Driver Module
    - Air Data Computer Module
    - Audible Signal Module
    - Computer Fail Signal Module
    - Doppler Radar Module
    - Flight Information Display Module
    - Forward Looking Radar Module
    - Head-Up Display Module
    - Inertial Measurement Set Module
    - Panel Module
    - Projected Map Display Set Module
    - Shipboard Inertial Nav. Sys. Mod.
    - Visual Indicator Module
    - Weapon Release Module
    - Ground Test Module
  - Shared Services Module
    - Mode Determination Module
    - Panel I/O Support Module
    - Shared Subroutine Module
    - Stage Director Module
    - System Value Module

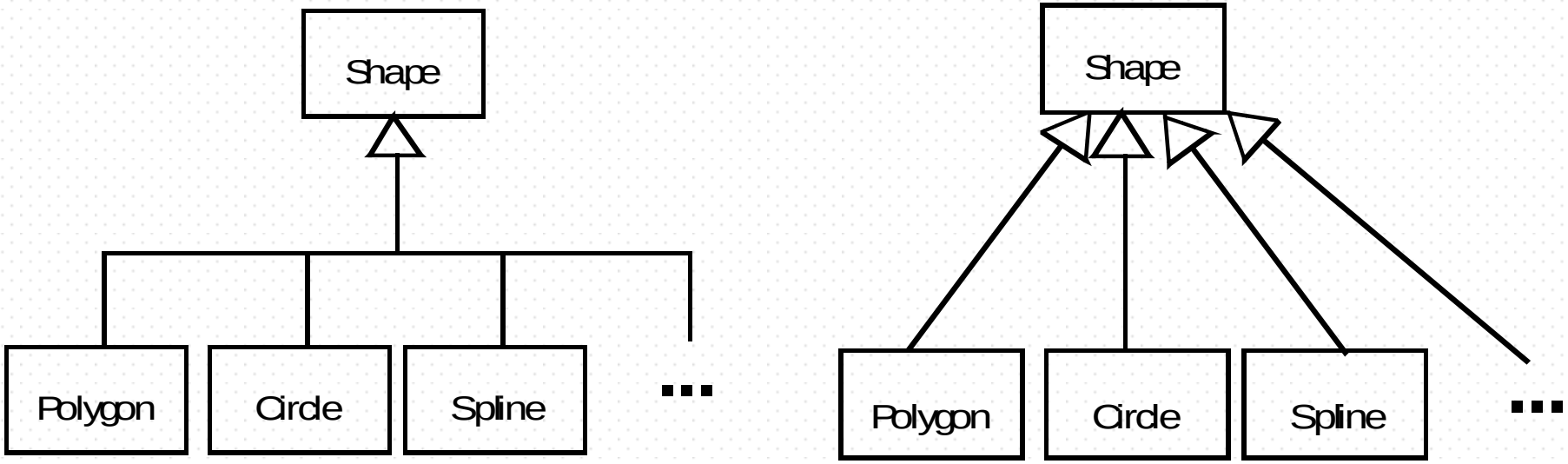
- Software Decision Module
  - Application Data Type Module
    - Numeric Data Type Module
    - State Transition Event Mod.
  - Data Banker Module
    - Singular Values Module
    - Complex Event Module
  - Filter Behavior Module
  - Physical Models Module
    - Aircraft Motion Module
    - Earth Characteristics Module
    - Human Factors Module
    - Target Behavior Module
    - Weapon Behavior Module
  - Software Utility Module
    - Power-Up Initialization Module
    - Numerical Algorithms Module
  - System Generation Module
    - System Generation Parameter Mod.
    - Support Software Module

Generalization Style

- Elements: Modules
- Relations: Generalizes, an "is a" relation
- Properties: Interface or implementation inheritance?
- What it's for
  - Basis for object-oriented designs
  - Providing for evolution and extension
  - Reuse

Generalization Style: Notations

- Formal:
  - Programming languages
  - UML



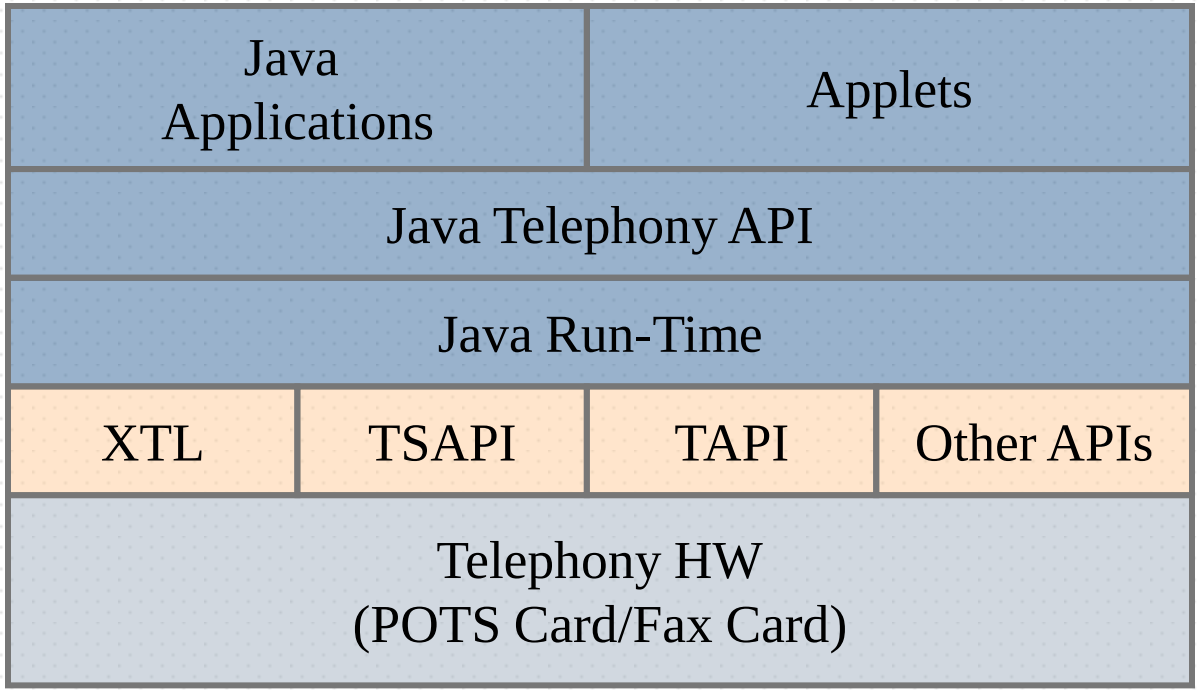
Layered Style

- Elements: Layers, a virtual machine
- Relation:
  - *Allowed-to-use*, a specialization of *depends-on*
- Rules:
  - Every piece of software is assigned to exactly one layer.
  - Software in a layer is allowed to use software in {any lower layer, next lower layer}
  - Software in a layer {is, is not} allowed to use other software in same layer.

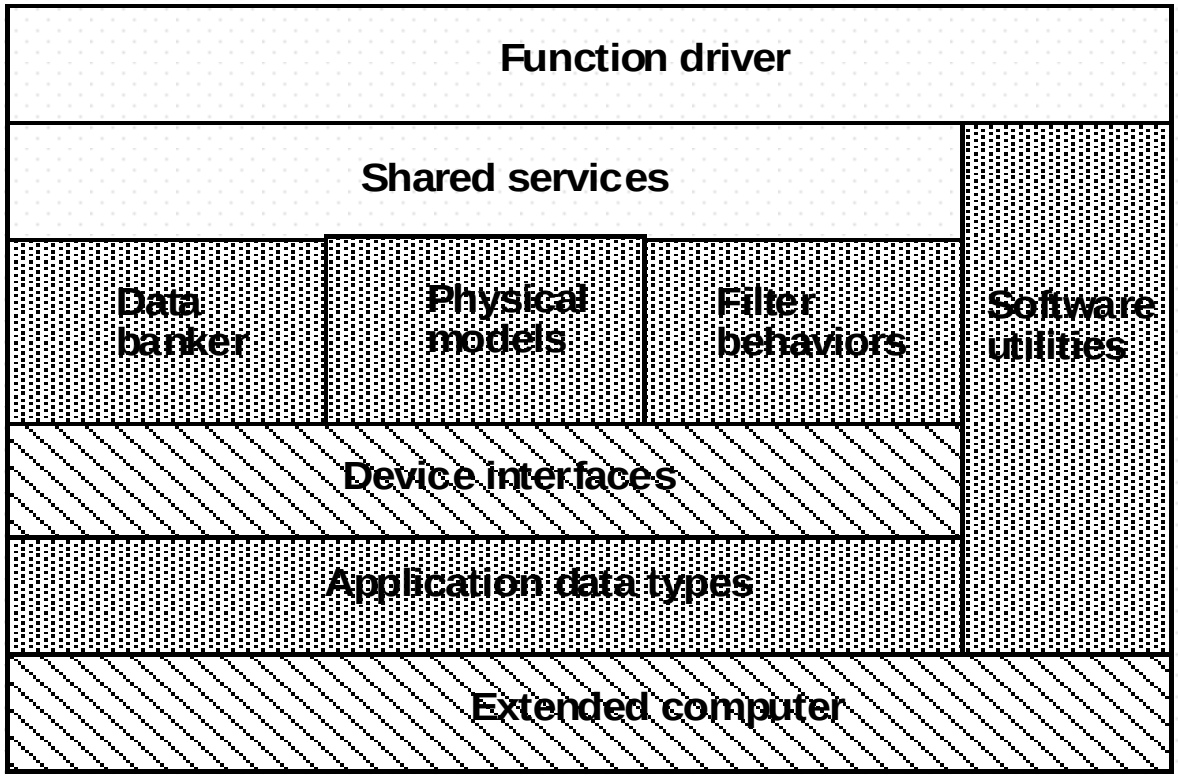
Layered Style (cont'd.)

- What it's for
  - Portability
  - Fielding subsets, incremental development
  - Separation of concerns
- Variations (many)
  - Segmented layers: Dividing a layer into segments (or sub-modules), with *allowed-to-use* relations between the segments within a layer and segments between layers.

Example 1: JavaPhone



Example 2

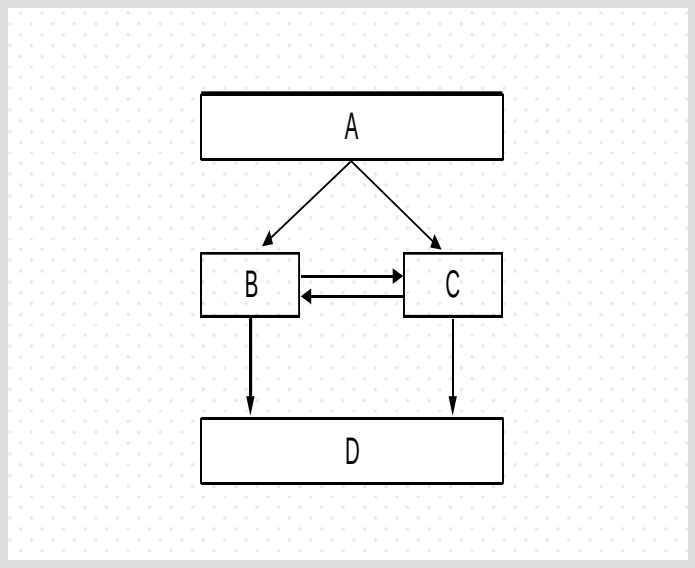


Key:

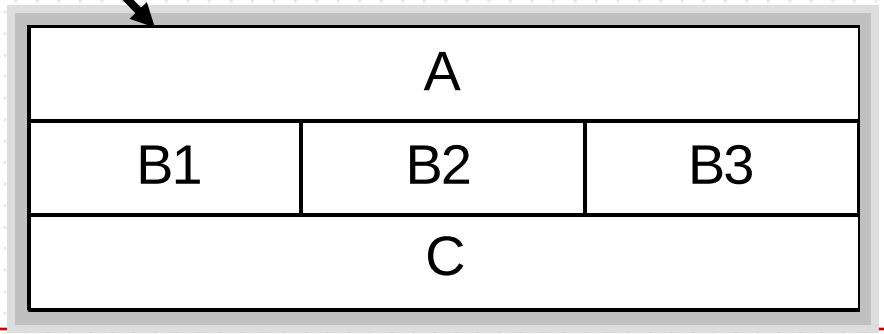
- Behavior-hiding module
- Software decision hiding module
- Hardware hiding module

Layered Style: Informal notations

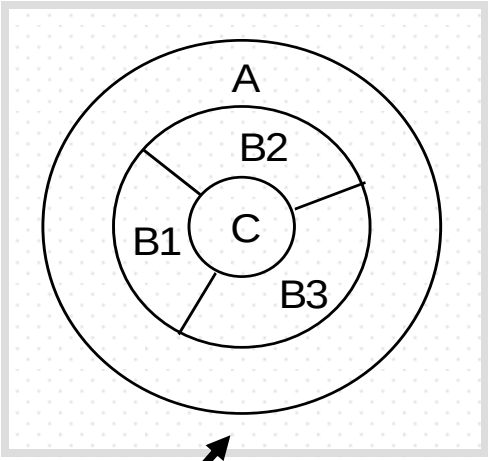
- Boxes and arrows



- Stacks of boxes



What does this mean?



- Concentric Rings

Component-and-connector (C&C) Views

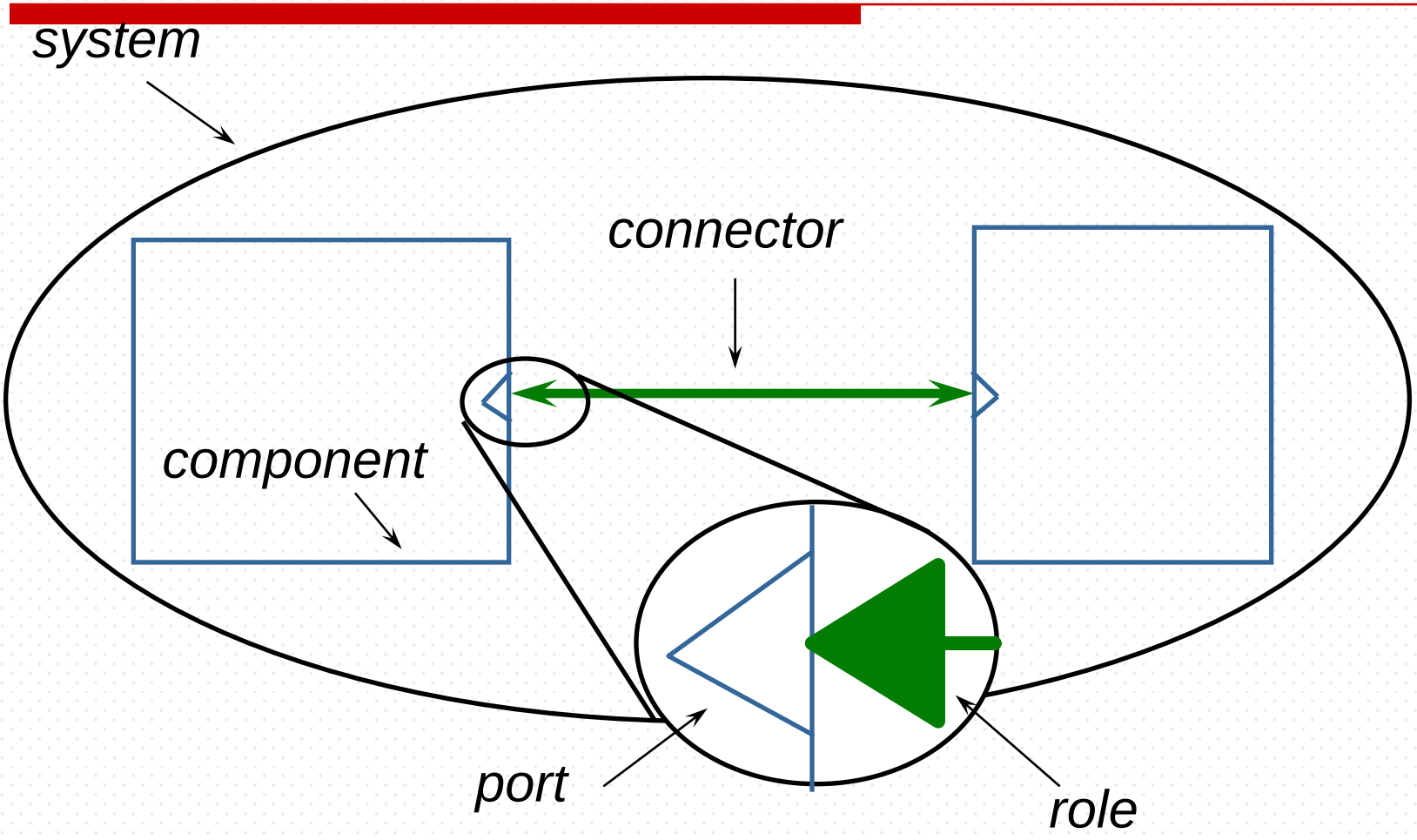
- Elements:
  - Components: principal units of run-time computation and data stores
  - Connectors: interaction mechanisms
- Relations: Attachment of components' *ports* to connectors' *roles* (interfaces with protocols)
- Properties: specify information for construction & analysis
  - Quality attributes (to help analysis)
  - Others, depending on style

What Are C&C Views Used For?

- Construction: Define how the system will appear at run time, and therefore determines what kind of behavior must be built in. Also, pathways of interaction, and communication mechanisms.

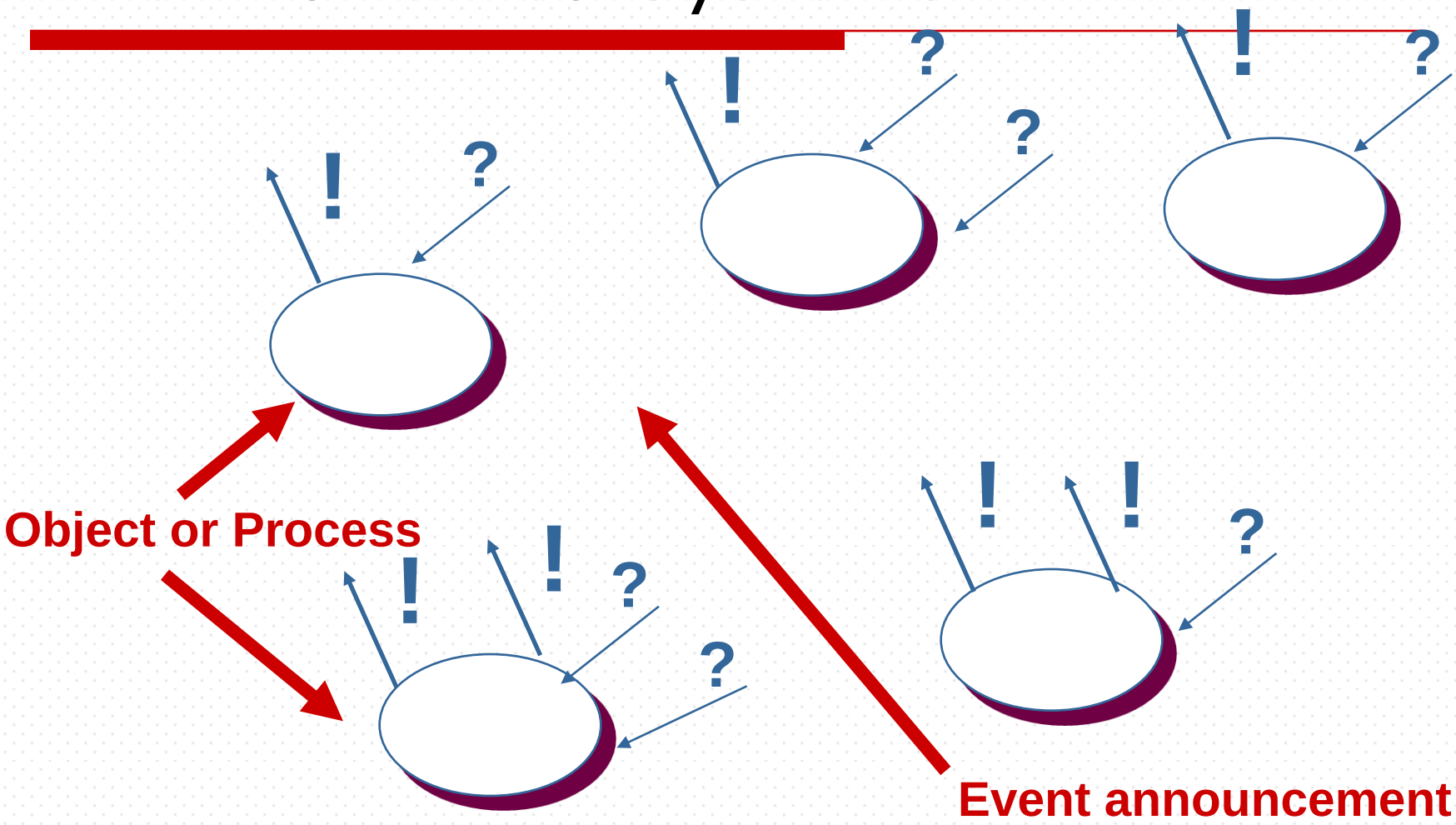
- Analysis: Runtime properties of a system, such as availability, performance, aspects of security, reliability, …

C&C Structural Concepts



Example C&C Style: Pipes and Filters

Example C&C Style: Event-oriented Systems



A Taxonomy of C&C Styles

- Data flow
  - batch sequential
  - pipes and filters
  - process control
- Call-return
  - main program-subroutine
  - object-oriented
  - component-based
  - peer-to-peer
  - service-oriented
  - N-tiered
- Event-based
  - asynchronous messaging
  - publish-subscribe
  - implicit invocation
  - data-triggered
- Data-centered
  - repository
  - blackboard
  - shared variables

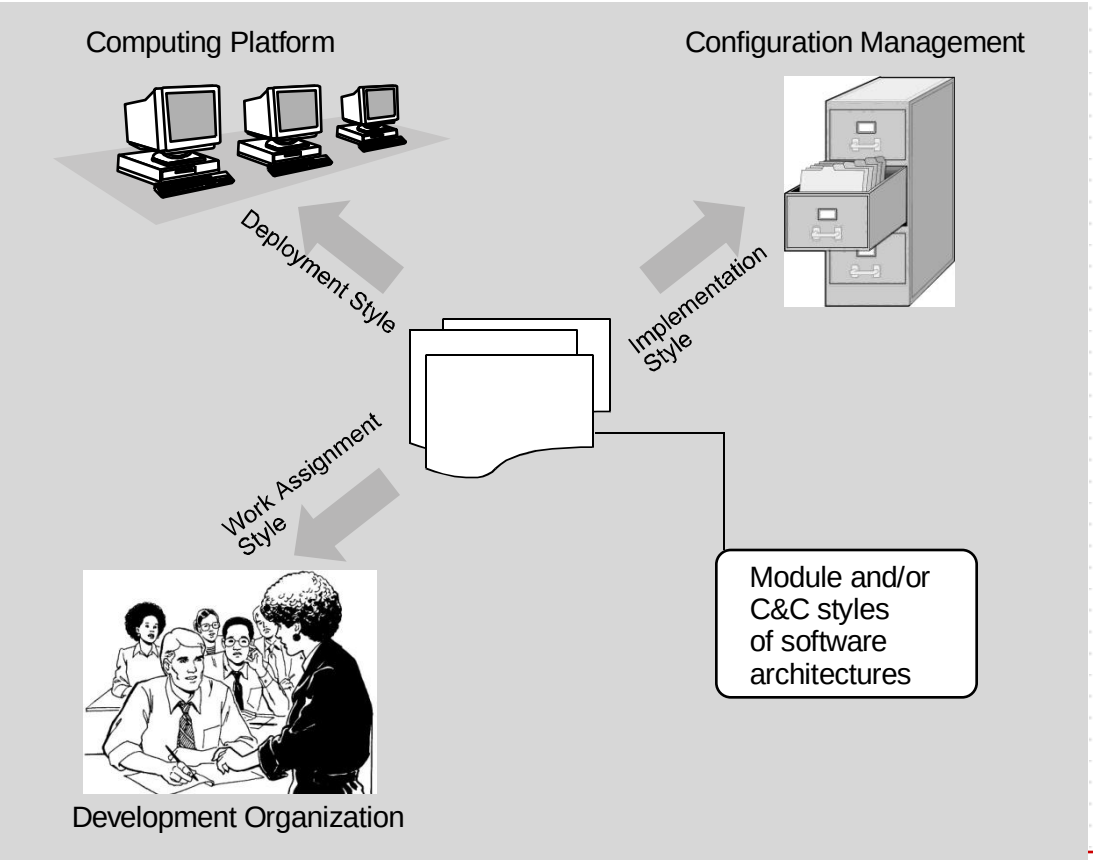
Allocation Views

- Elements:
  - Software elements (as defined in module or C&C styles)
  - Environment elements
- Relations: *allocated-to*

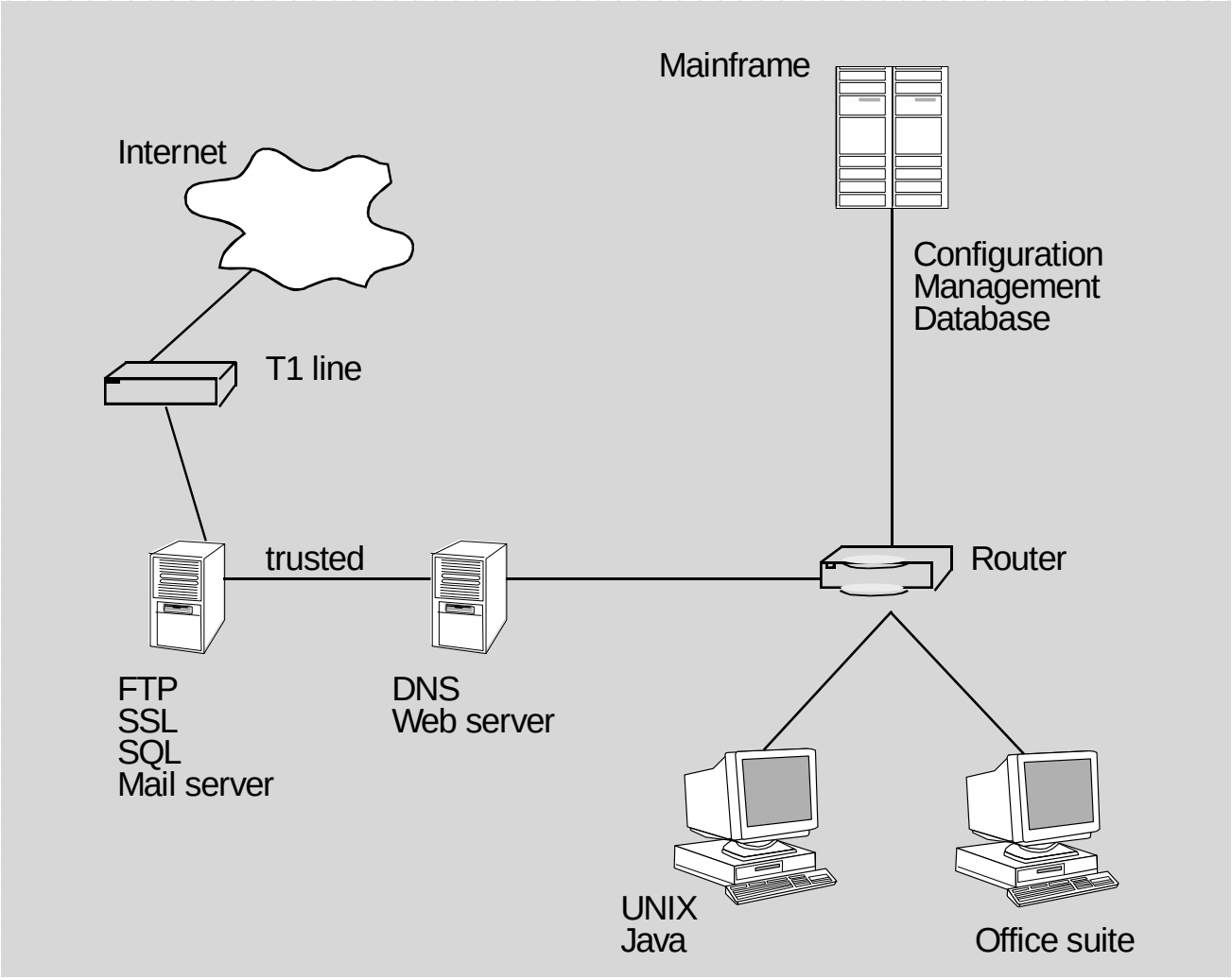
Styles of allocation views

- Deployment style
  - Allocates software elements to processing and communication nodes
  - Properties include those necessary to calculate (and achieve) performance, availability
- Implementation style
  - Allocates software elements to structures in the development environment's file systems
  - Properties include files and capacities
- Work assignment style
  - Allocates software elements to organizational work units
  - Properties include skill sets

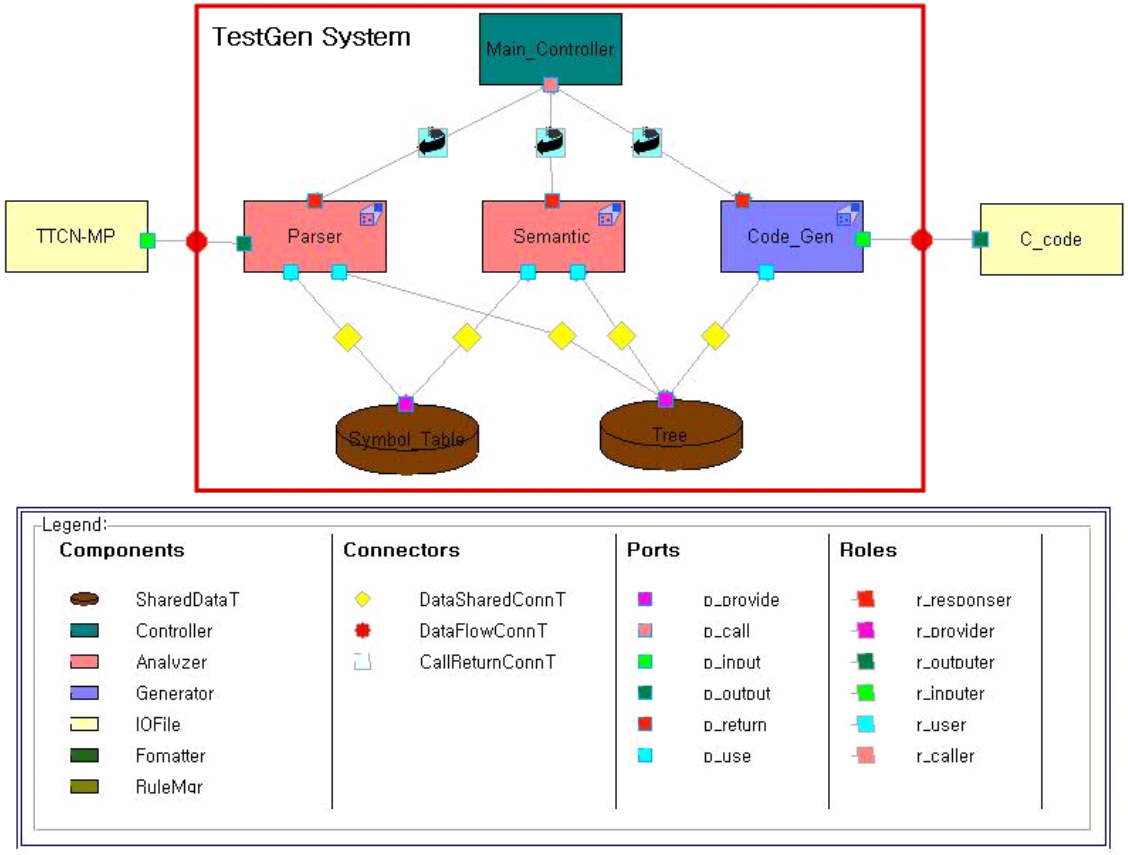
Allocation viewtype: Software elements and environment elements



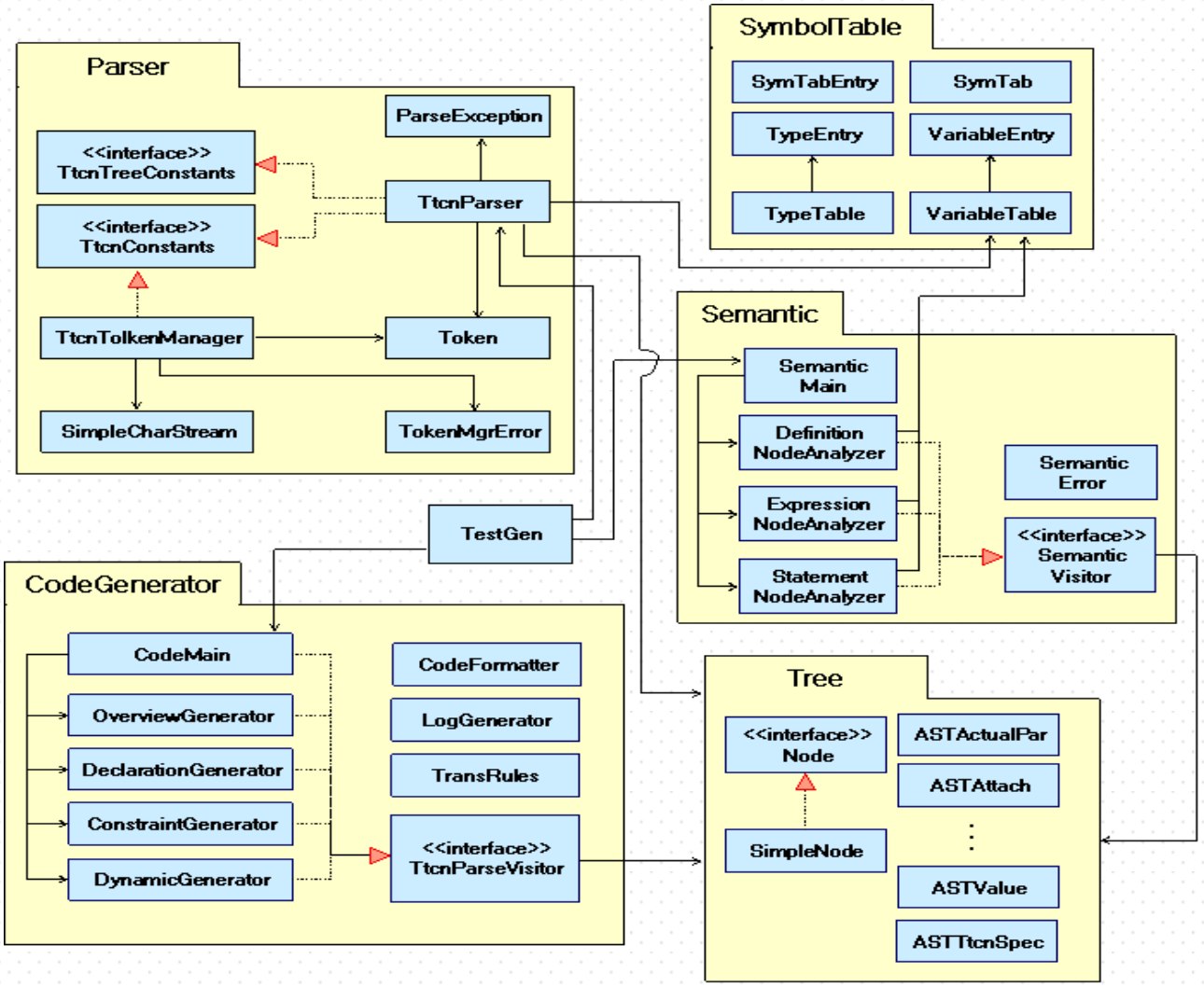
Deployment Style: Example

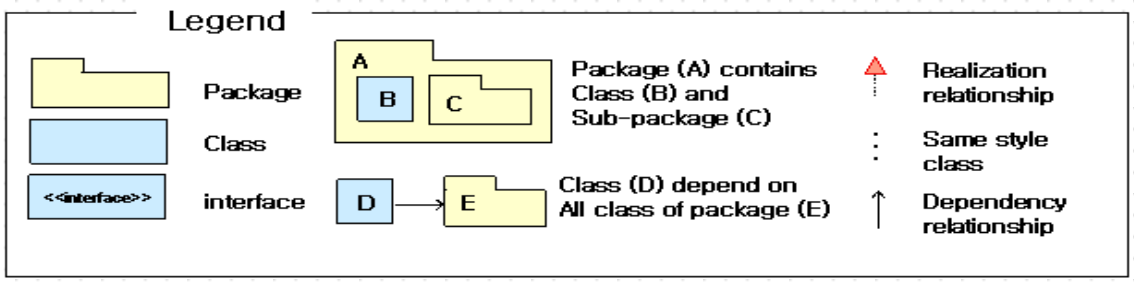


C&C View of Test Parsing System

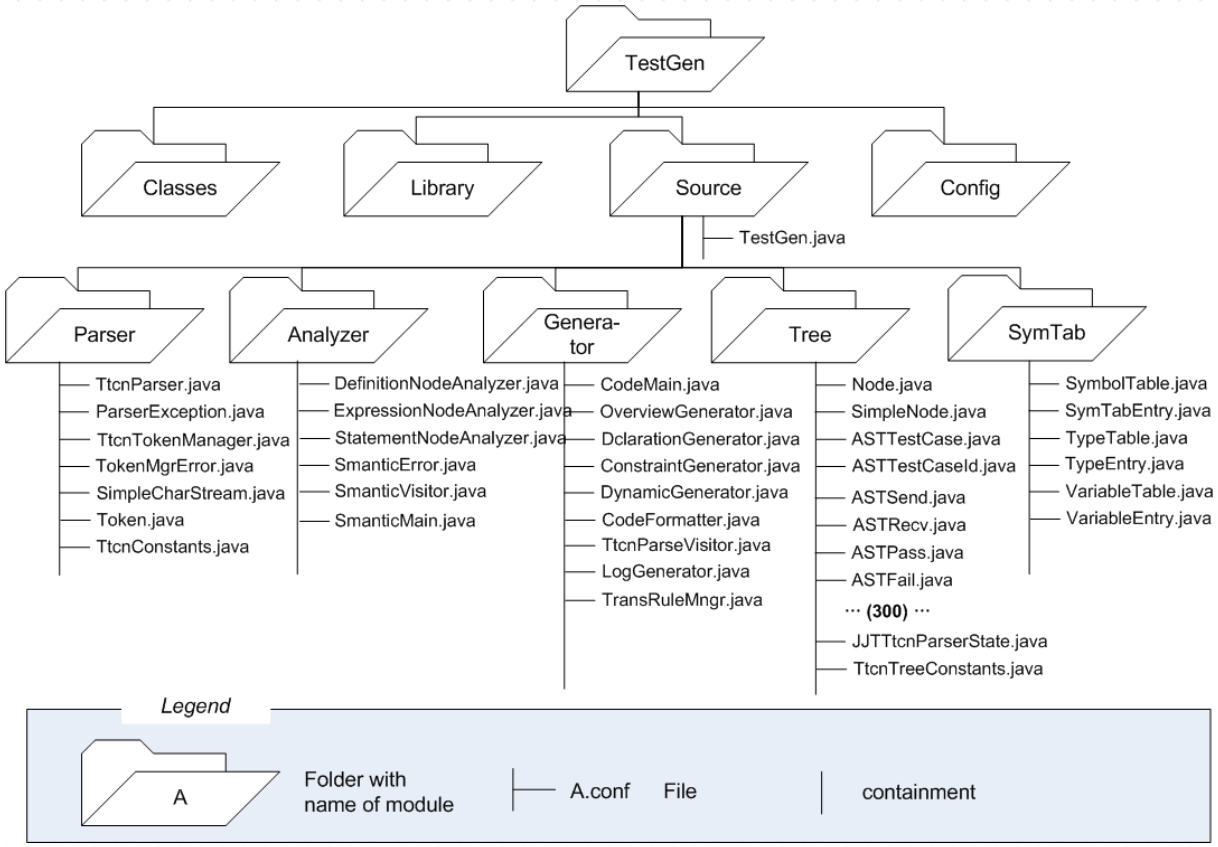


Module Decomposition Style





Implementation Style for the same system



Summary

- We discussed the differences between system, enterprise, and software architecture.
- We introduced:
  - elements, relations, structures, views, and styles

Session Summary (continued)

- Views allow us to manage what we say about an architecture
- Perspectives determine the category of view
- Three primary categories of view
  - Module, C&C, Allocation
- Each has many styles
  - Module: Decomposition, Layered, …
  - C&C: Pipe & Filter, Client-server, …
  - Allocation: Deployment, …

Book References

- *Software Architecture in Practice, 2nd Edition* Bass, Clemens, Kazman, Addison Wesley, 2003.
- *Software Architecture: Perspectives on an Emerging Discipline,* Shaw, Garlan Prentice-Hall, 1996.
- *Documenting Software Architecture: Views and Beyond,* Clements et al., Addison Wesley, 2003.