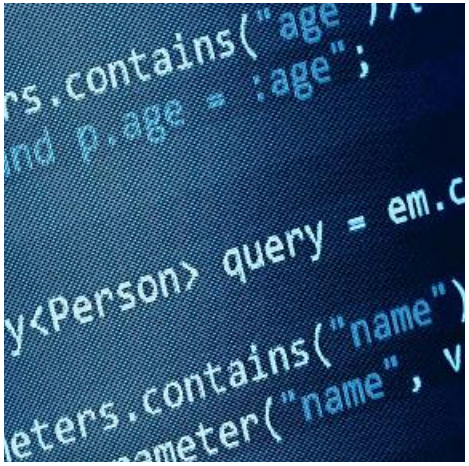
CS3500 Software Engineering

2017/2018



Dept. Computer Science
Dr. Klaas-Jan Stol

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Welcome to CS3500

Software Architecture

Part II

After studying this material and associated papers, you should be able to:

- Define software architecture, component, module
- Explain why different views are necessary to document software architectures.
- Be able to recognize the 3 different view types: module views, component & connector views, and allocation views.

Why learn this theory stuff?

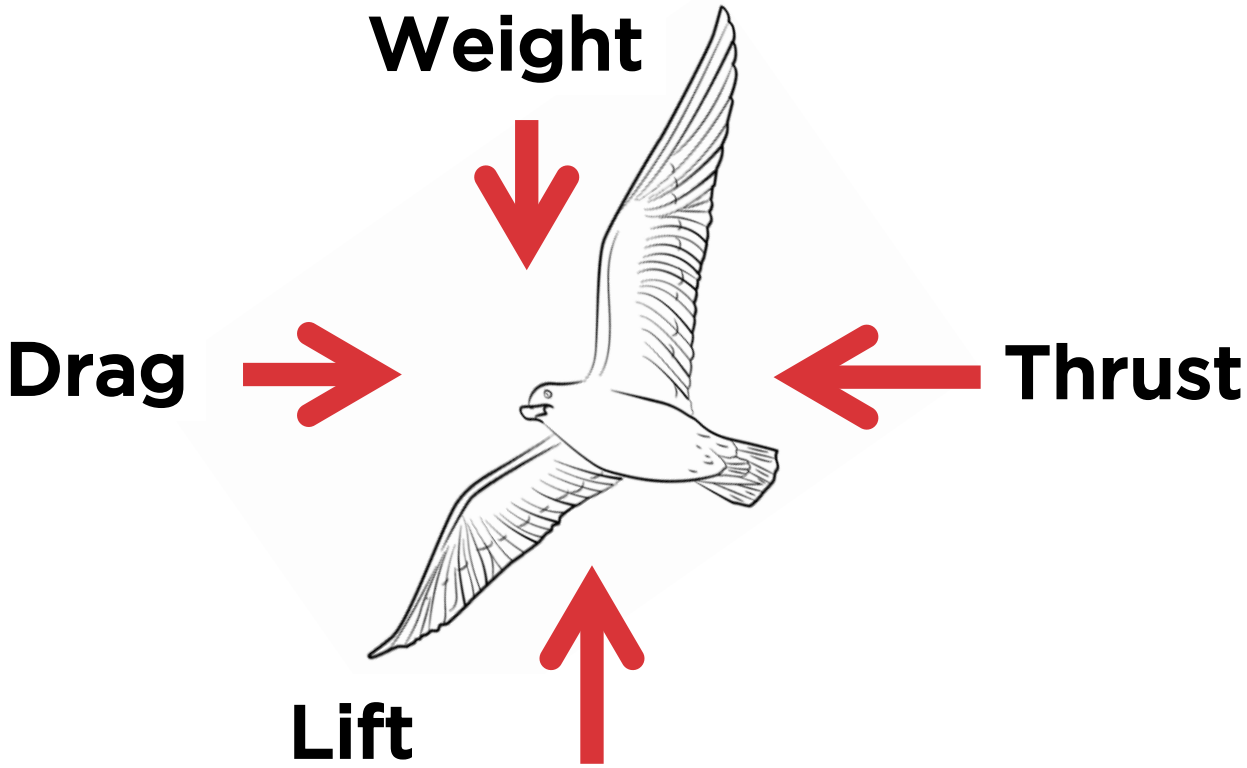
> There is nothing as practical as a good theory.
>
> —Kurt Lewin (1890-1947)

INTERMEZZO

Theory of Flight (and aerodynamics) helps to design airplanes

Key terms:

- Lift
- Weight
- Thrust
- Drag



Airplanes fly if:
Lift > Weight &&
Thrust > Drag

Theory informs practice

Theory

- **Definition of terms:**
  expand your vocabulary.

- **Abstraction:**
  Enables us to reason about systems without building & failing.

Contents

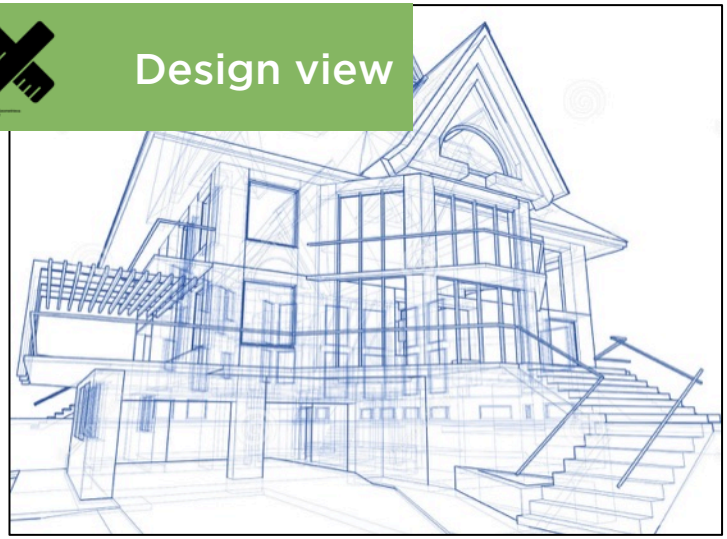
Different stakeholders, different views

- In building construction, different stakeholders have different concerns

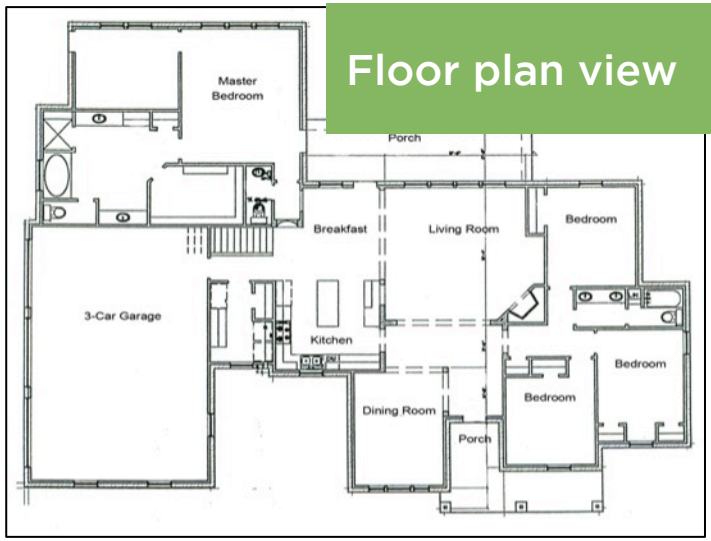
- Each stakeholder has different views on the project:
  - An architect focuses on design and integrity
  - An electrician focuses on wiring and sockets
  - A plumber focuses pipes
  - Builders construct walls and roofs
  - *... etc.*

Civil engineering as a metaphor for SA

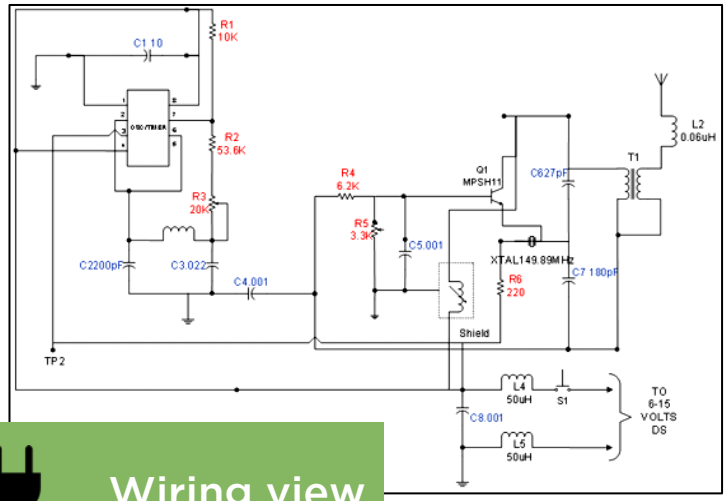
Design view



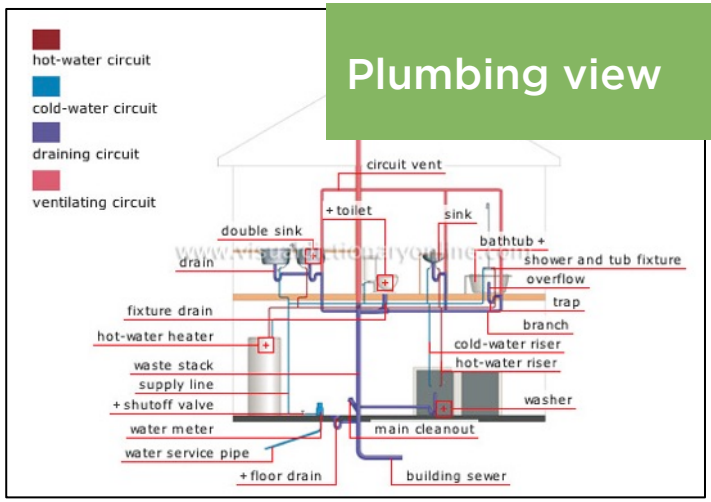
Floor plan view



Wiring view



Plumbing view



Architectural views

- Different models to capture architectural views:
  - Kruchten's 4+1 views model
  - Siemens Four View model (S4V)
  - Rozanski & Woods' model
  - Philips' CAFCR model
  - IEEE 1471-2000 standard
  - Clements et al.'s Views and Beyond approach

- Sets of views vary across these approaches, but are fundamentally similar.

Which is best?

Which “views” model you pick is a matter of preference.

Same ~~pizza~~ system, different slicing.

Need for multiple views: why & how

Why? Cannot capture everything in 1 diagram.

How? 3 Questions to ask of the architecture:

1. How is the system structured in terms of implementation units?
2. How is the system structured in terms of runtime behavior and interactions?
3. How does the software relate to non-software structures in its environment?

On notations

- UML is a standard for drawing architecture diagrams.
- Many architecture documents do not use UML,
  - Use informal notations instead
- We will be using UML.
  - But used examples and papers might not

The Architecture of Open Source Applications

EXAMPLE

Architecture of Audacity

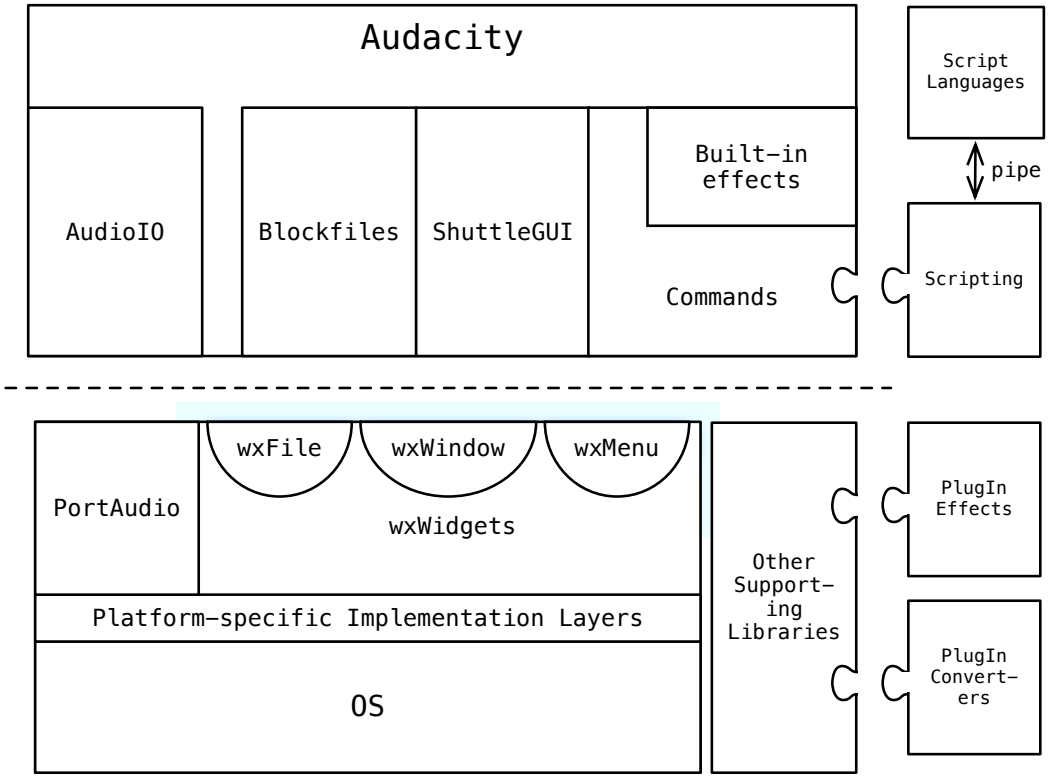


Figure 2.1: Layers in Audacity

Which question does this diagram answer?

1. Organization of implementation units (modules)?
2. Organization in terms of runtime behavior and interactions?
3. How do software elements map to environment?

Or a bit of all three?

EXAMPLE

Architecture of BASH shell

Same for this diagram?

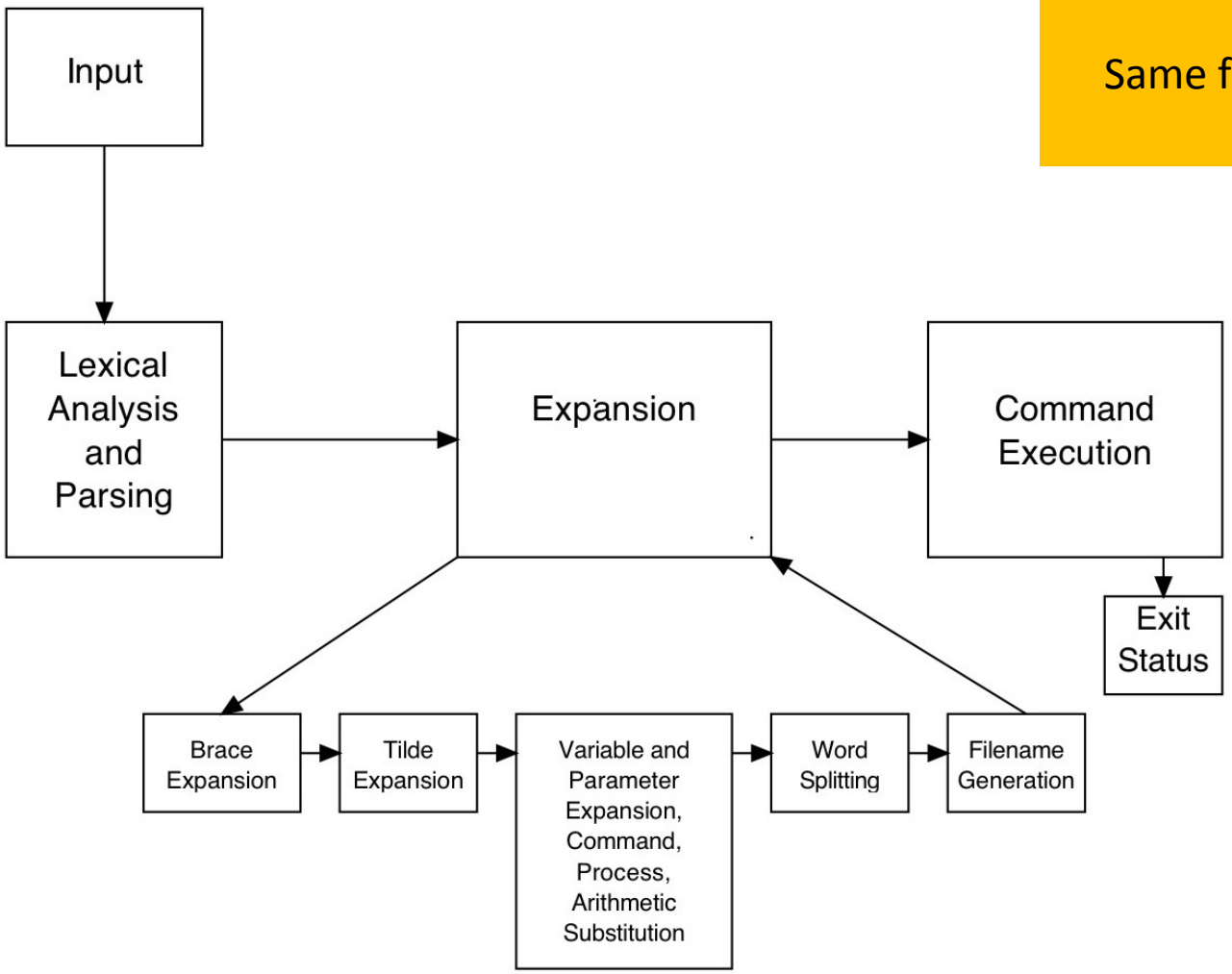
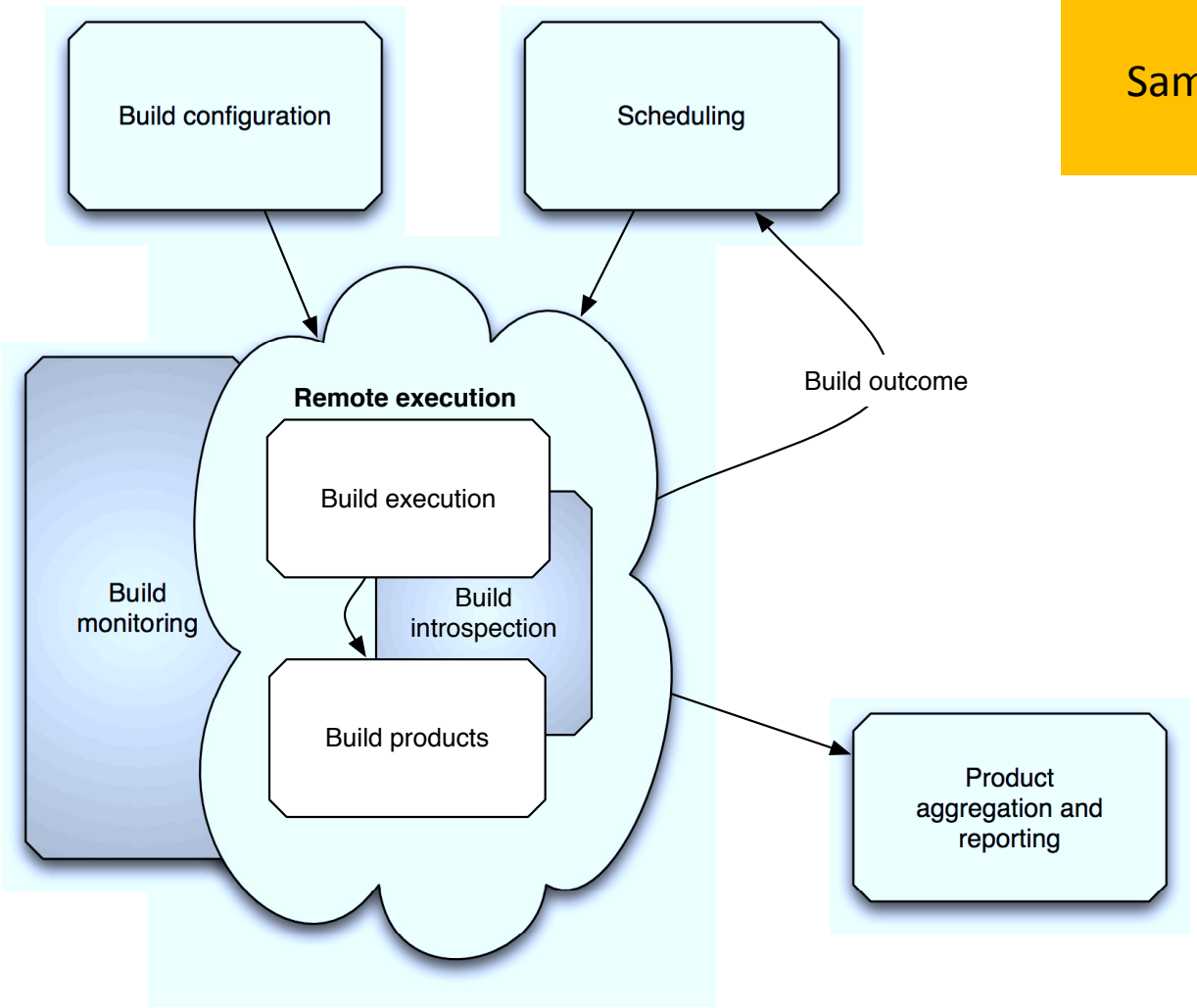


Figure 3.1: Bash Component Architecture

EXAMPLE

Architecture of a CI system



Same for this diagram?

Figure 6.1: Internals of a Continuous Integration System

EXAMPLE

PyPI: Python Package Index

Architecture of PyPI

Same for this diagram?

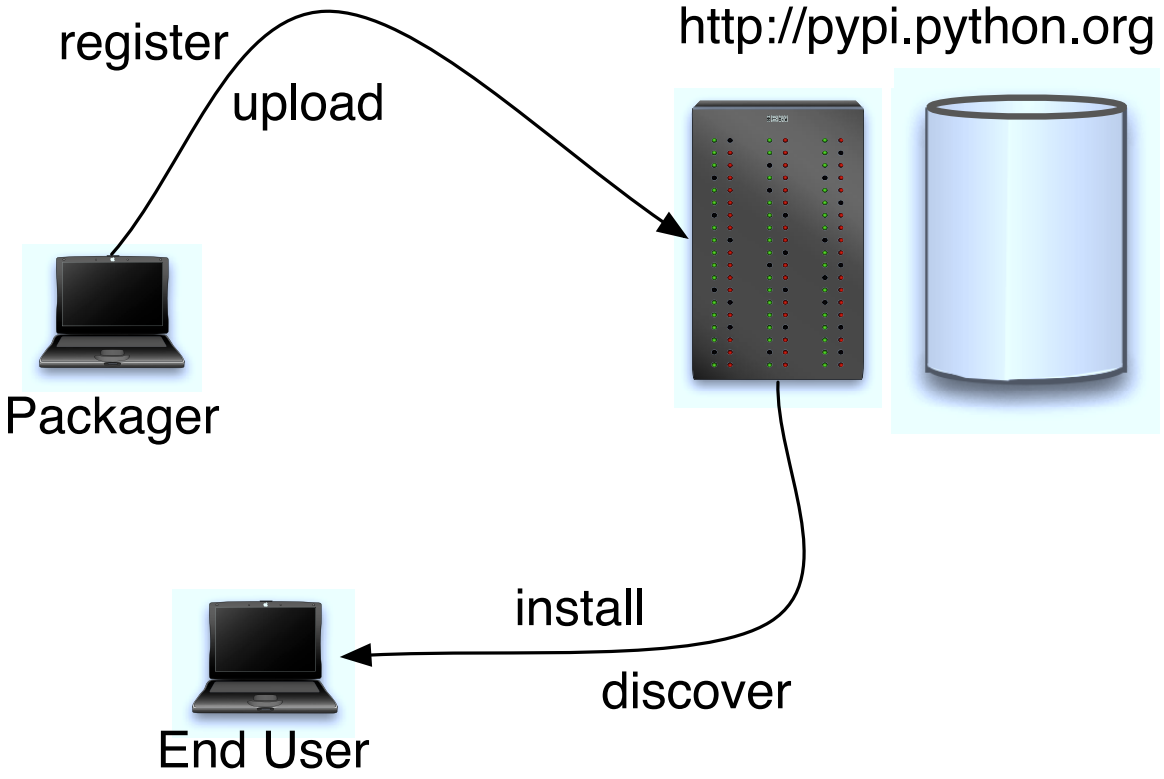


Figure 14.3: PyPI Workflow

EXAMPLE

PyPI: Python Package Index

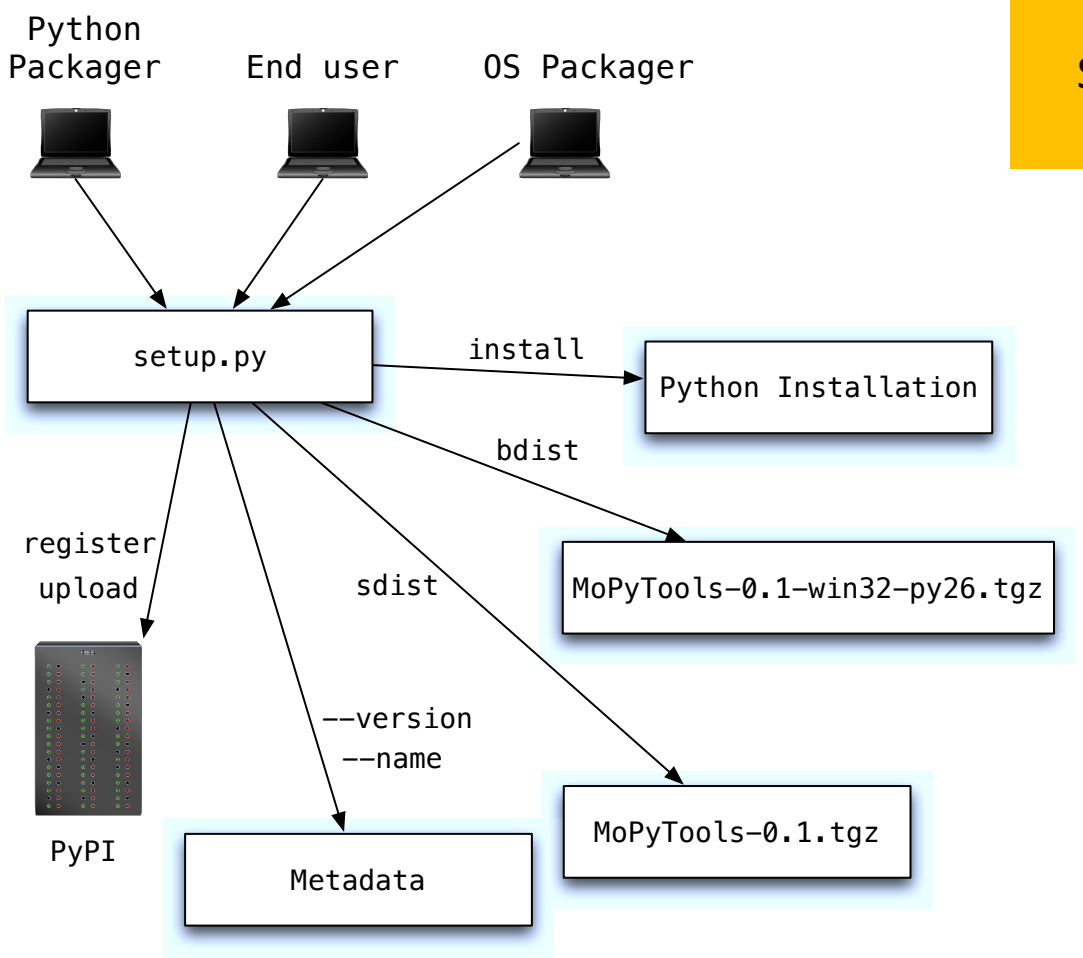


Figure 14.1: Setup

Same for this diagram?

EXAMPLE

Space Shuttle Primary Avionics System

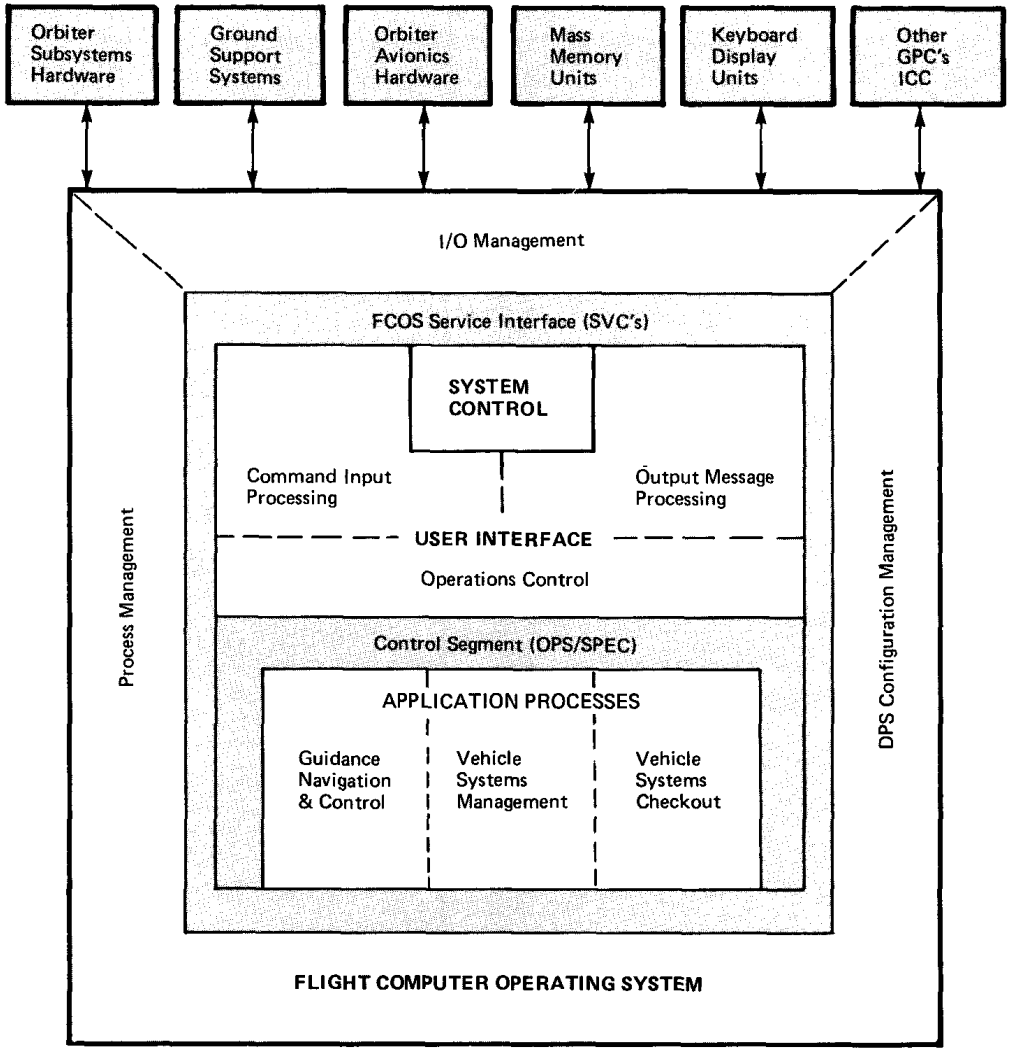


FIGURE 6. Software Architecture (Isolation of the applications software from the external environment is accomplished by the PASS architecture.)

And, same again for this diagram?

Why “reason” about it? Need answers!

- What do these diagrams mean?
- What are the strengths and weaknesses of these architectures?
- Are these the “right” architectures?
- To answer these questions, we first need to document architectures.

SECTION IV

Module Views

Clements et al.'s "Views and Beyond" method

1.	2.	3.
What are they	Why use 'm?	Examples

Module views

Modules are units of implementation.

A module view documents module structures of a system's architecture.

What modules may look like:

- Class, package, library
- Configuration file (e.g. XML)
- Specification file (e.g. BNF for parsers)

Relations among them:

1. Is-part-of
2. Depends-on
3. Is-a

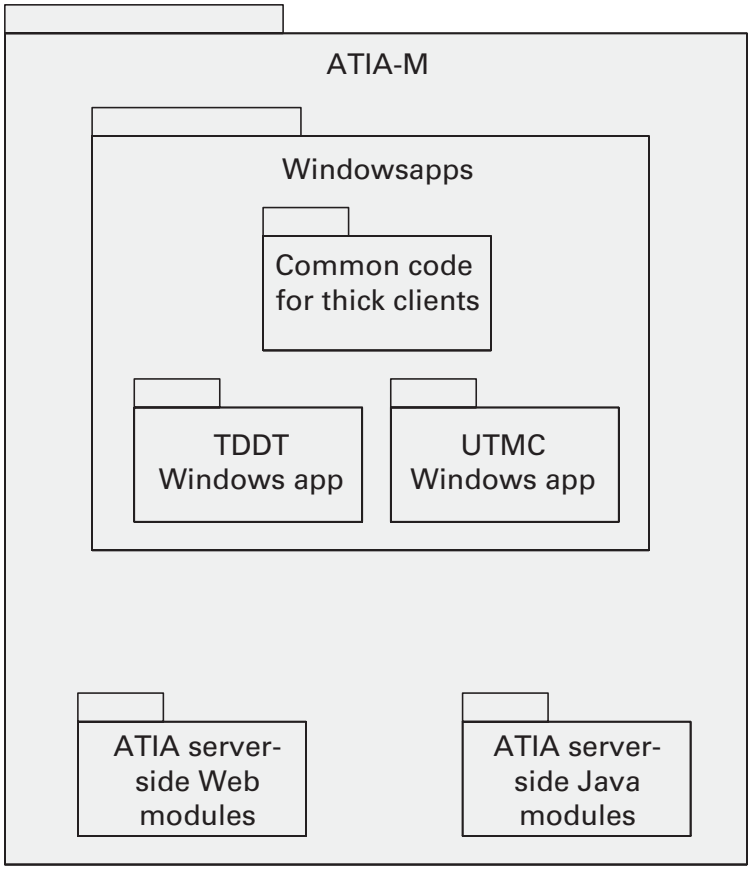
Why use module views?

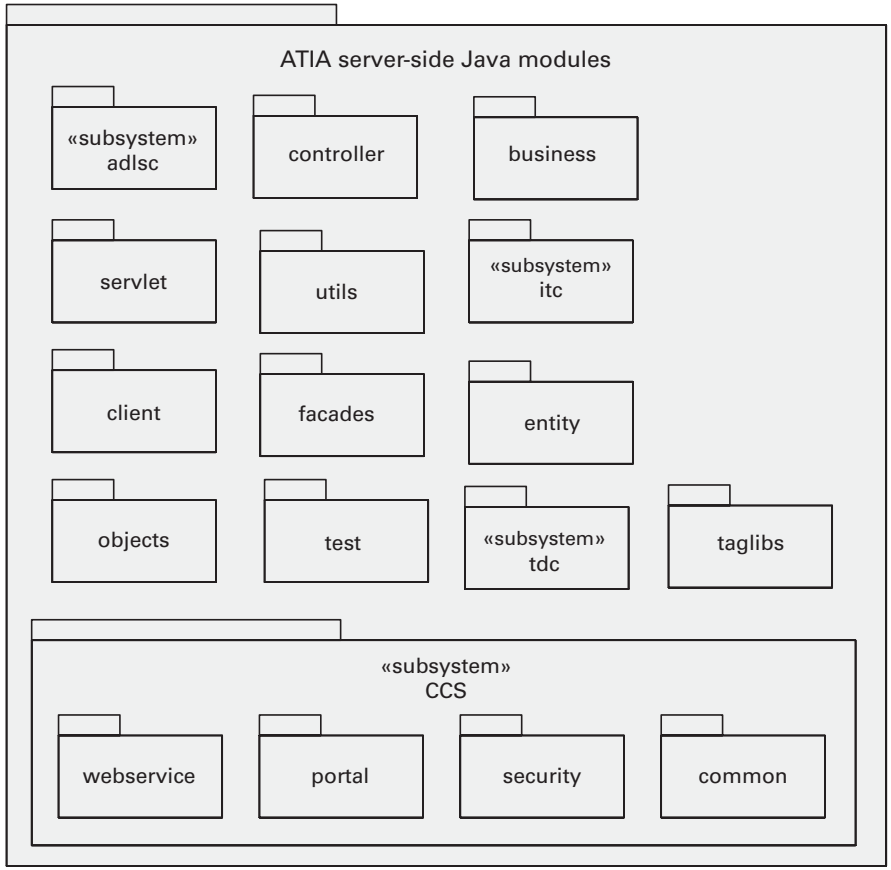
Module views are used for:

1. Construction
   provide blueprint for source code, data store
2. Analysis
   e.g. requirements traceability, impact analysis
3. Communication
   explain system functionality and structure

EXAMPLE

Decomposition (is-part-of) of ATIA-M



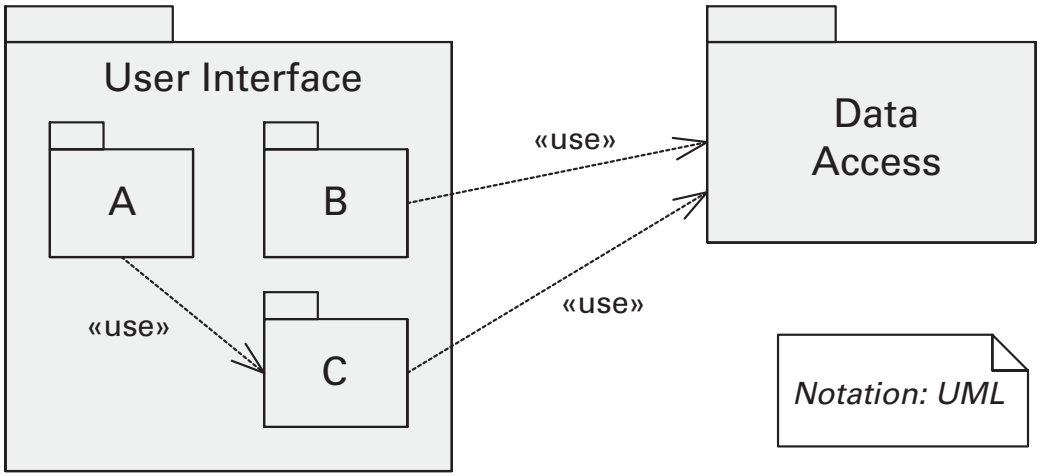


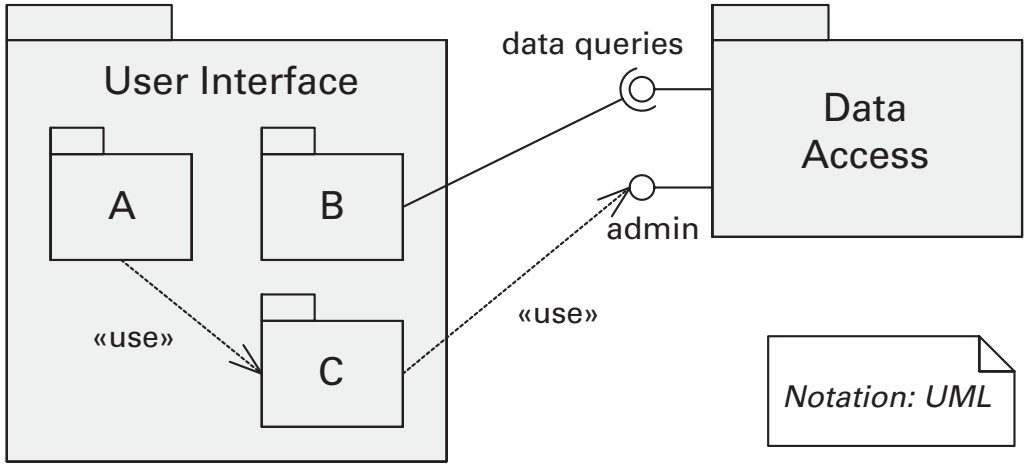
Notation: UML

Notation: UML

EXAMPLE

Two types of <<use>> (depends-on) notation



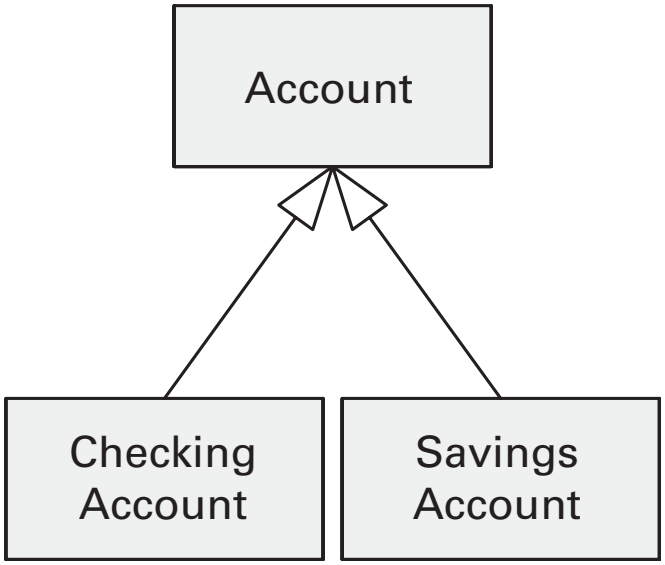


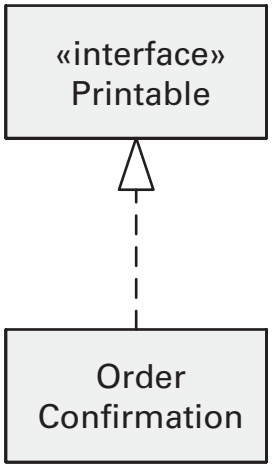
Either use:

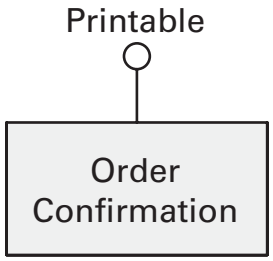
- **<<use>> dependency**
- **socket-lollipop.**

EXAMPLE

Is-a generalization relationship







Notation: UML

SECTION IV

Component & Connector Views

Clements et al.'s "Views and Beyond" method

1.	2.	3.
What are C&C views?	Why use 'm	Examples

Component & Connector views

A component is a runtime entity that can be deployed independently.

A C&C view documents elements that have a runtime presence.

How are components implemented?

- Services
- Threads, processes
- Clients, servers

How are connectors implemented?

- Pipes or queues
- Request/reply protocols
- Direct/indirect/event-driven invocation

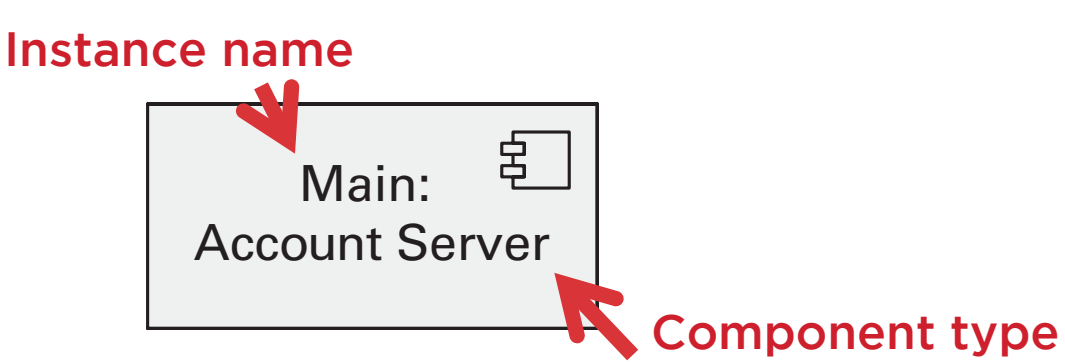
C&C ... & Ports & Roles

Components: runtime entity that does processing.

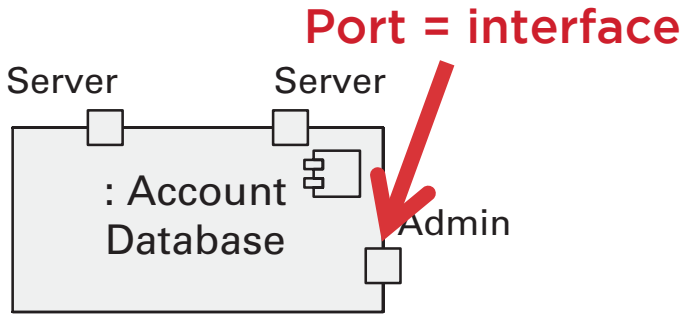
Connector: entity that links components.

Ports: interface of a component.

Role: interface of connector



A component instance ("Main") of type "Account Server", without any ports defined.



An anonymous component instance of type "Account Database", with 3 ports defined.

Why use C&C views?

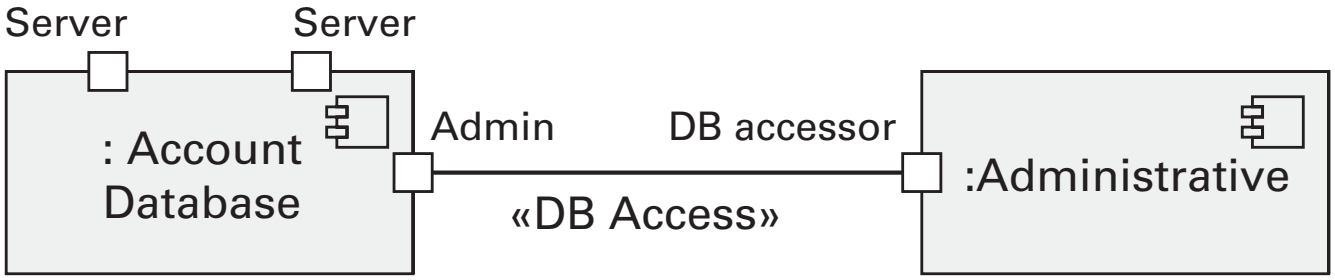
Relations among them:

1. Attachment: component ports link to connector roles
2. Interface delegation: internal ports link to external component ports (for subsystems)

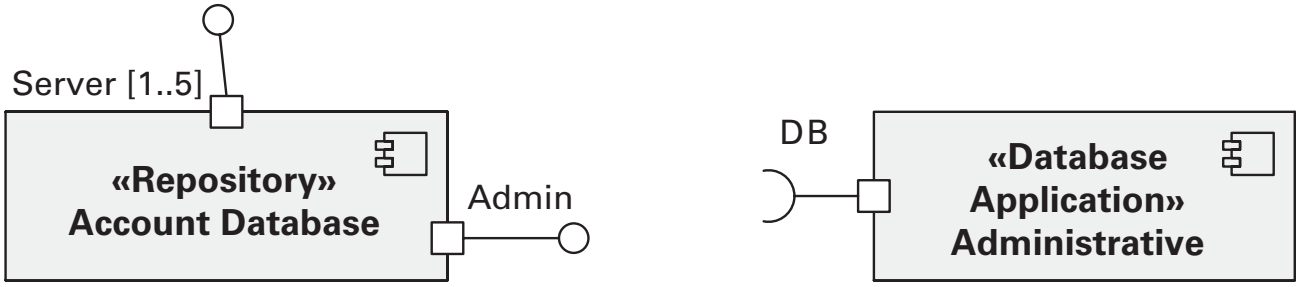
C&C views are used for:

1. Show how it works
   how are components linked?
2. Guide development
   e.g. specify structure and behavior of runtime elements
3. Analysis
   helps to reason about runtime system quality attributes, e.g. performance, reliability, availability

C&C in UML



Account Database and Administrative component instances linked with a <<DB Access>> connector

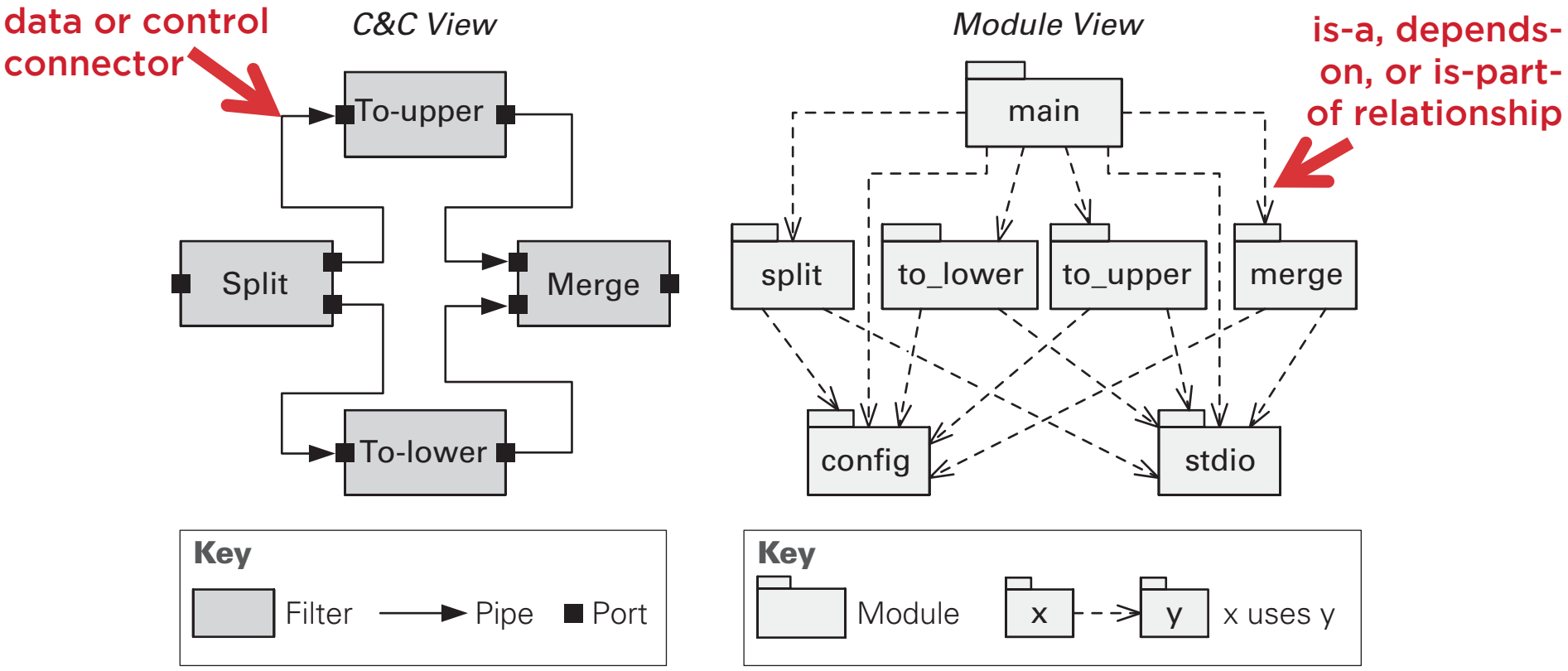


Account Database and Administrative component types, with “lollipop” provides and requires interfaces

Module view and C&C view

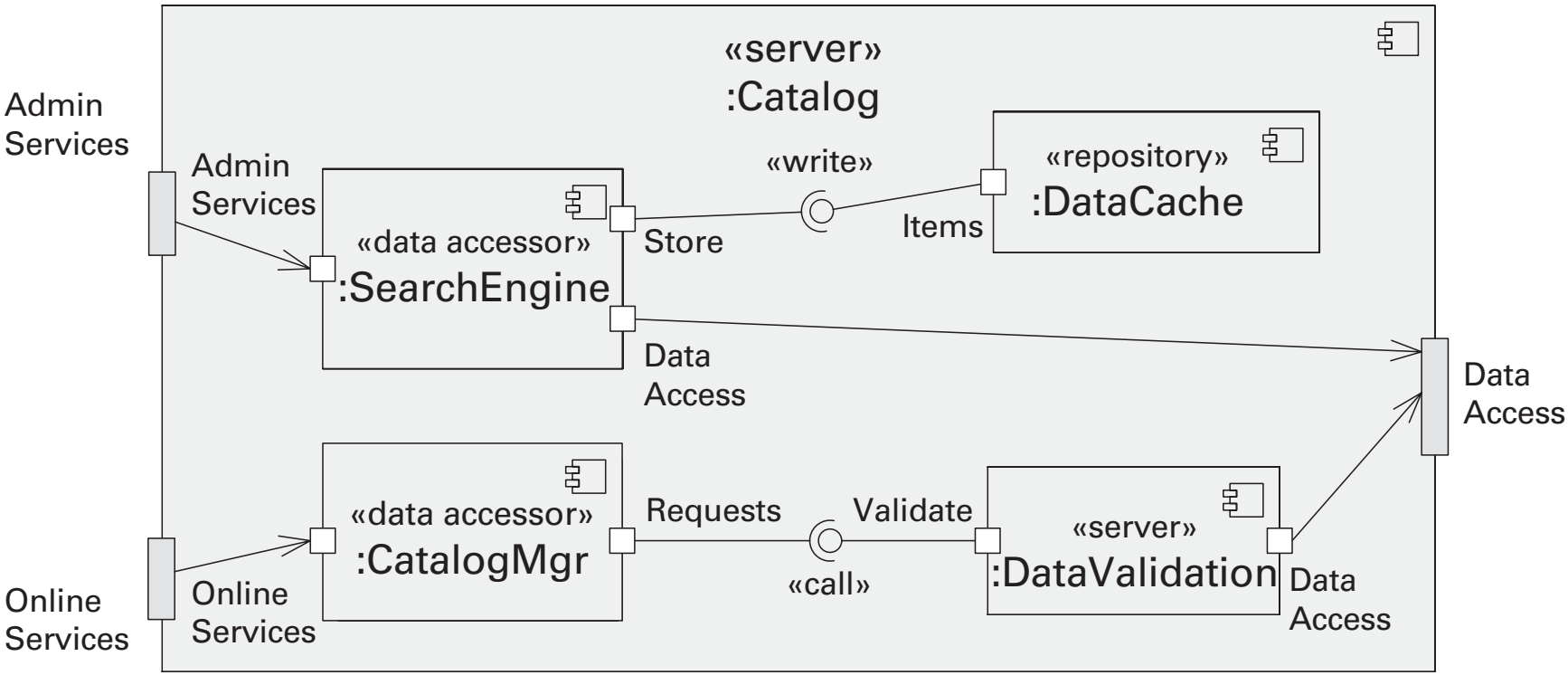
What's the difference?

While terminology mirrors **component vs. module**, the difference discussed here refers to whether it's a module or C&C **VIEW**.



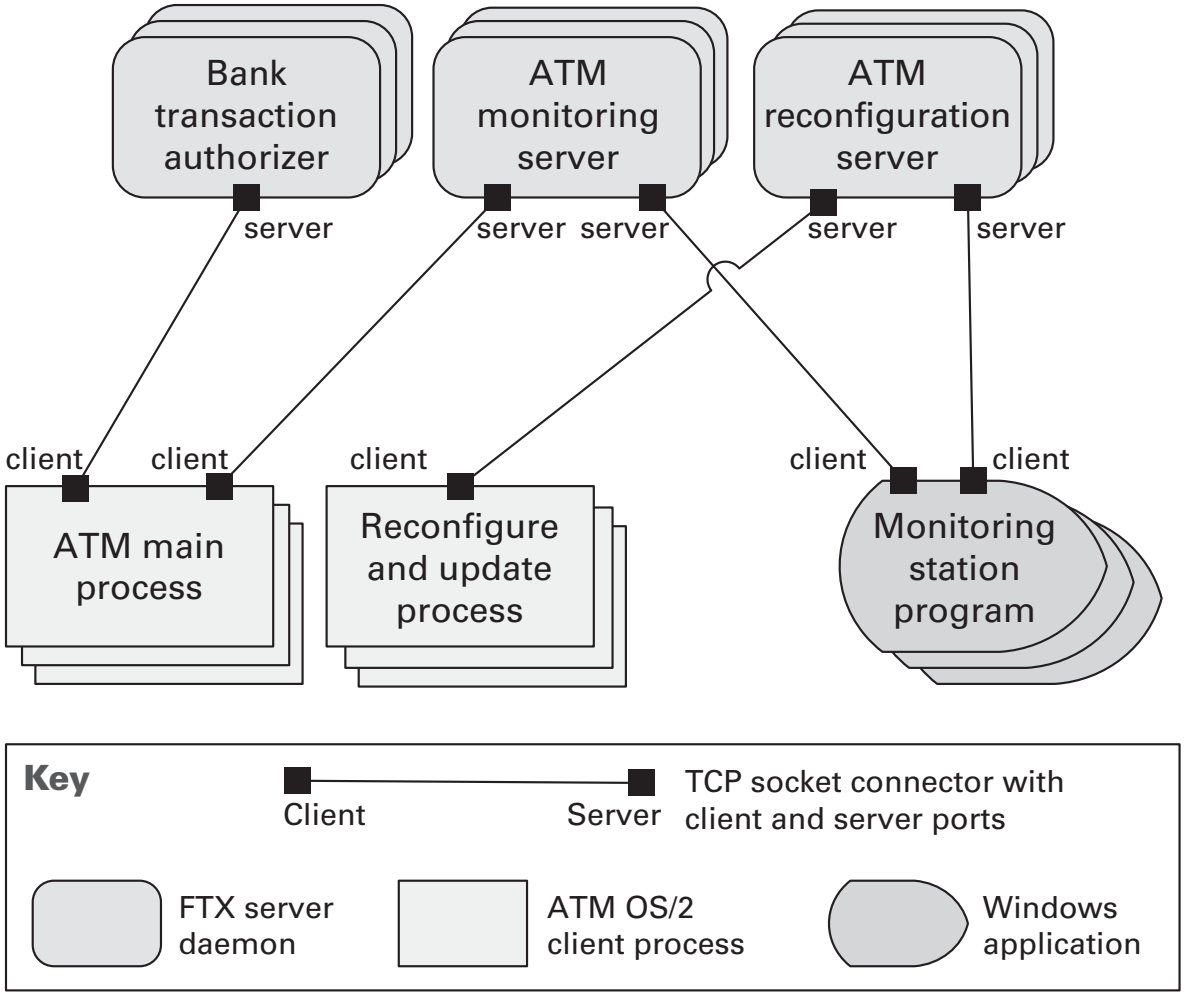
EXAMPLE

A “Catalog” server subsystem (nested components)



EXAMPLE

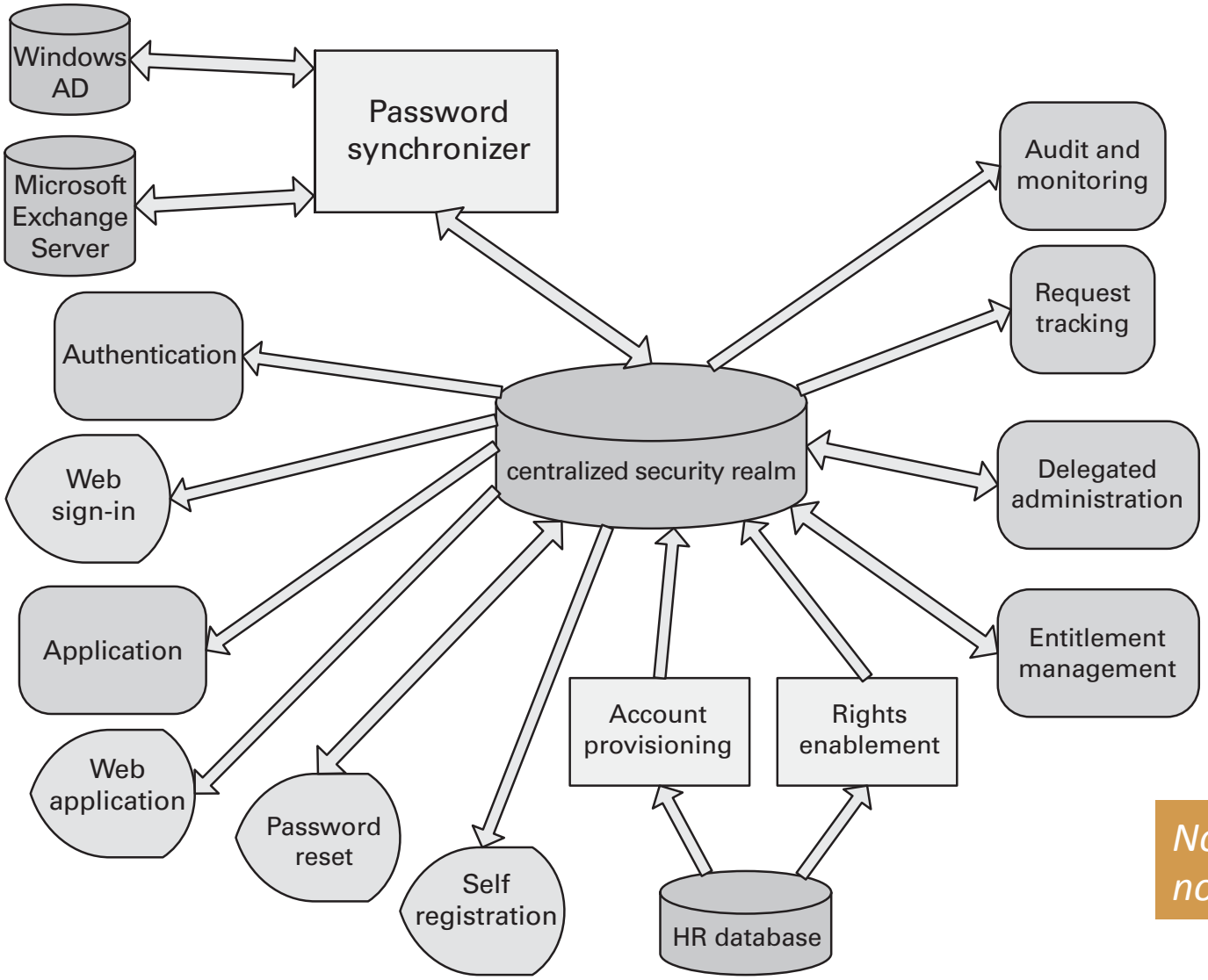
Client-server architecture of an ATM



Note: this is not UML!

EXAMPLE

Enterprise access management system



Note: this is not UML!

SECTION IV

Allocation Views

Clements et al.'s "Views and Beyond" method

1.	2.	3.
What are they	3 types of allocation views	Examples

Allocation views

Allocation views map software elements (module or C&C) to non-software elements.

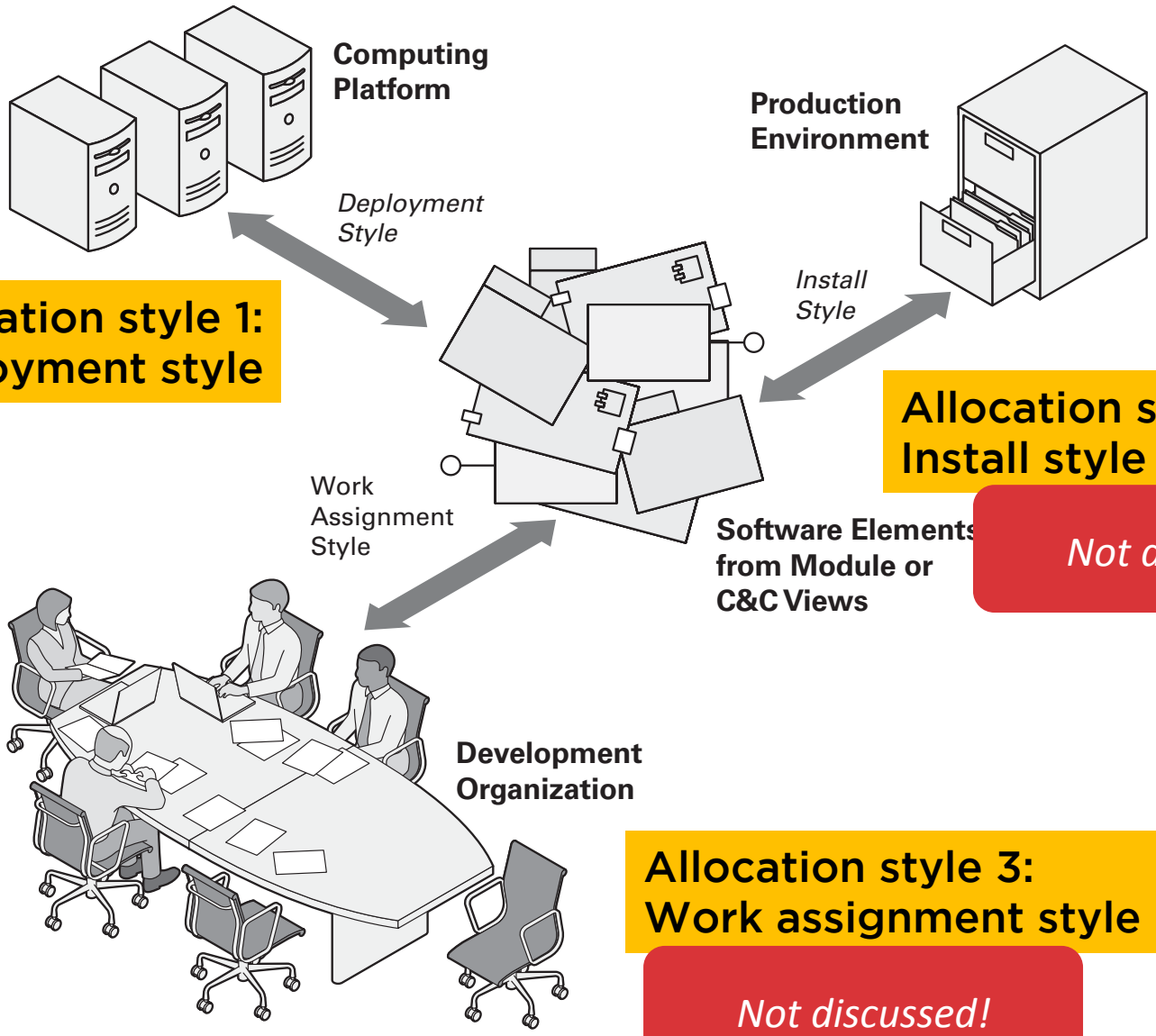
Non-software elements = environment, e.g.:

- Computing & communication hardware (CPU, memory, network, etc.)
- File management systems

Software elements have properties that are required from the environment.

The environmental element has properties that are provided to the software.

3 types of allocation styles



Allocation style 1: Deployment style

Allocation style 2: Install style

Not discussed!

Allocation style 3: Work assignment style

Not discussed!

Deployment style

Software elements:

- C&C view elements

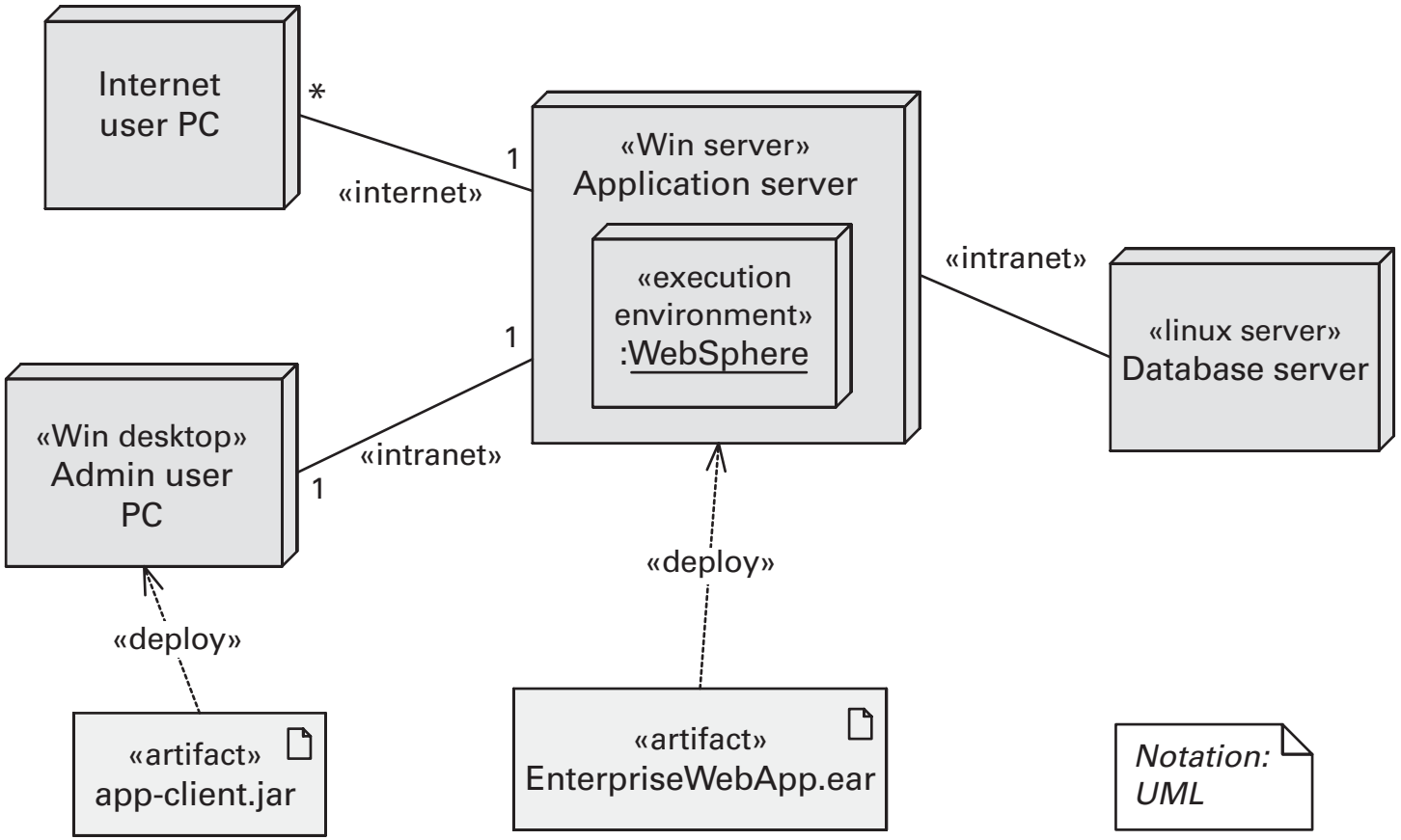
Useful to document features required from hardware: processing power, memory, capacity, etc.

Environmental elements:

- Hardware, e.g.:
  - Processor
  - Memory
  - Disk
  - Network

EXAMPLE

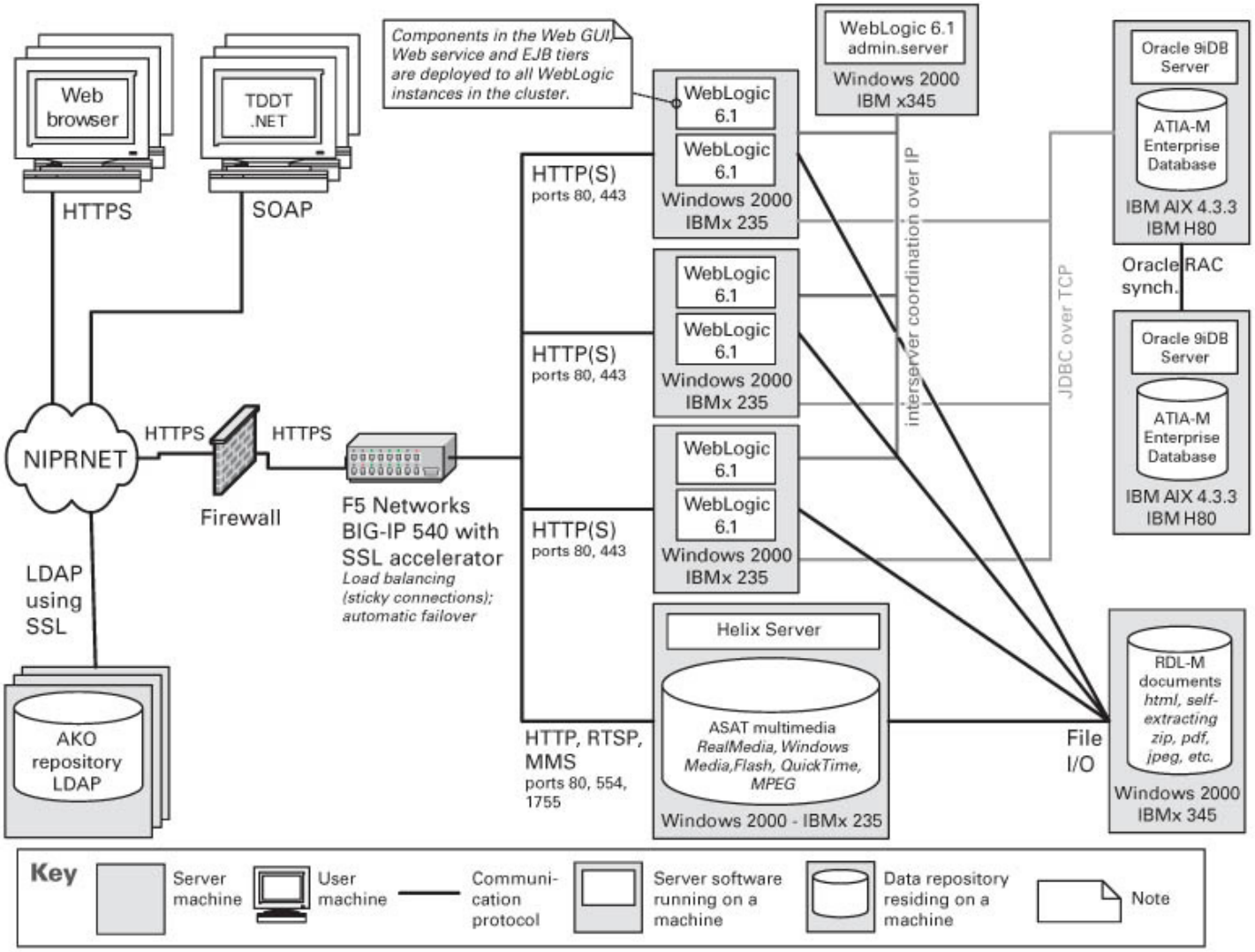
US Army Training Information Architecture-Migrated System



EXAMPLE

US Army Training Information Architecture-Migrated System

Note: this is not UML!



Why use the deployment style allocation view?

Analyzing runtime quality attributes:

- performance
- availability
- reliability
- Security

Estimating cost

- e.g. hardware needed

Understanding and explaining where the software runs.

EXAMPLE

Revisiting: Architecture of Audacity

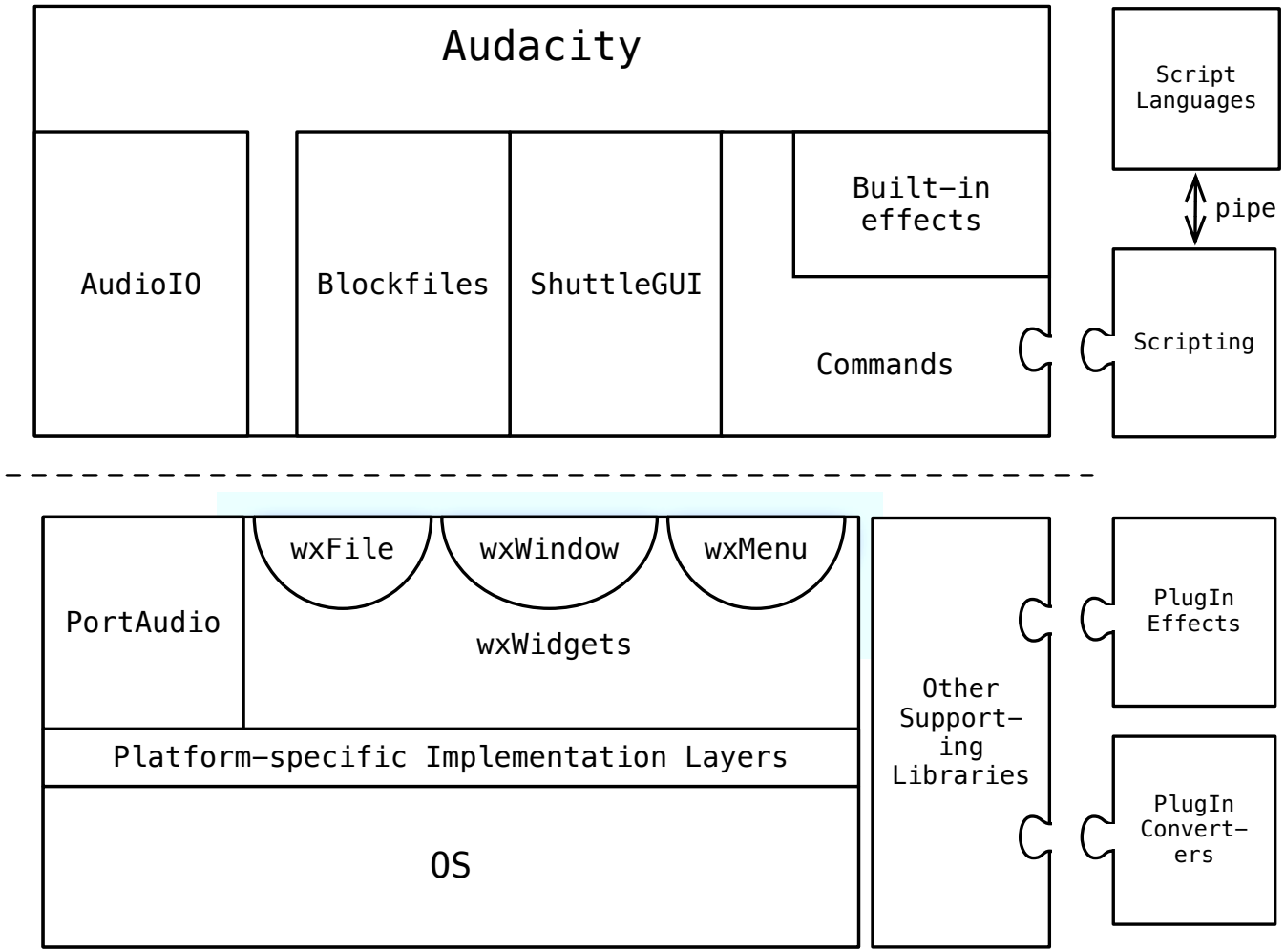
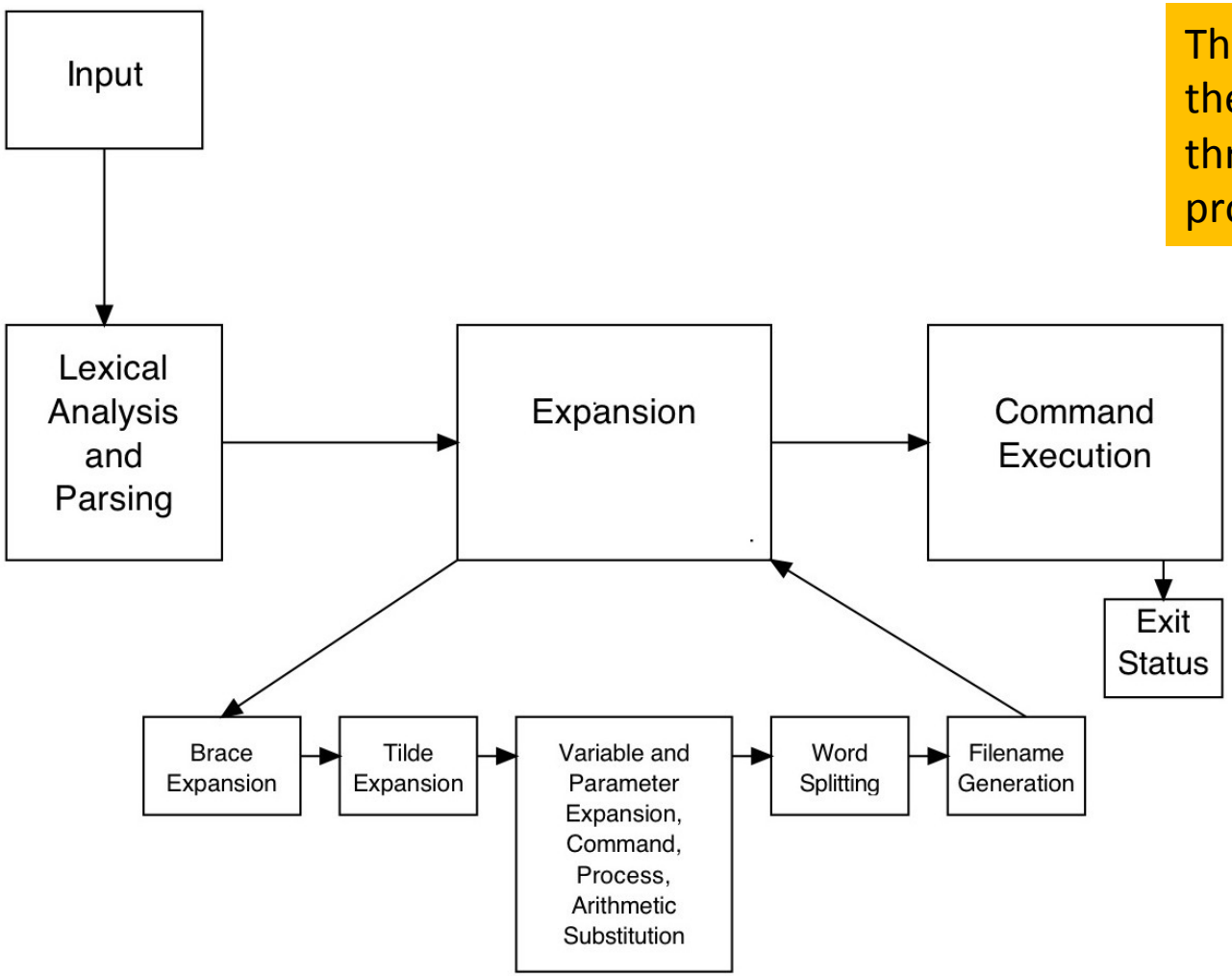


Figure 2.1: Layers in Audacity

This diagram shows the modular structure of Audacity.

EXAMPLE

Revisiting: Architecture of BASH shell



This diagram shows the "flow" of input through the BASH program.

Figure 3.1: Bash Component Architecture

EXAMPLE

Revisiting PyPI: Python Package Index

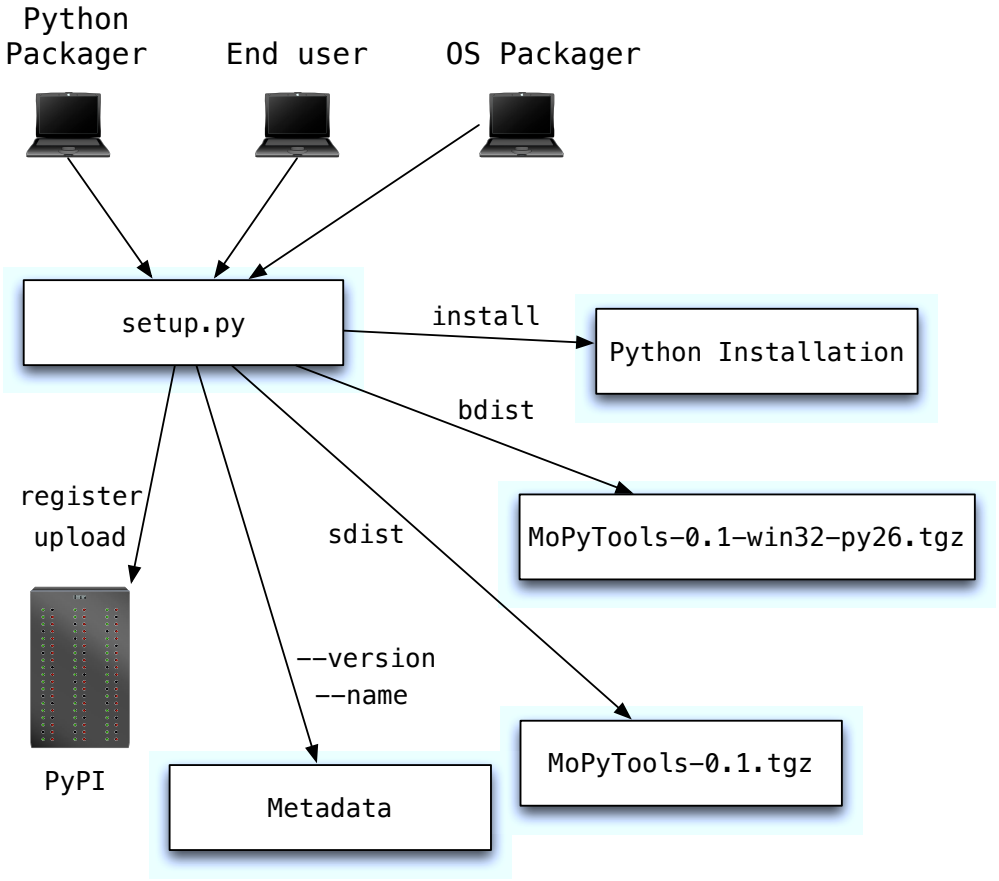


Figure 14.1: Setup

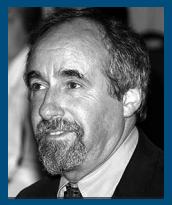
This diagram illustrates what happens when running PyPI:

- Option “install” runs installation
- Option “bdist” builds a distribution
- Option “sdist” builds a source distribution
- Option “--version” and “--name” provides metadata
- Option “register” and “upload” registers/ uploads packages to PyPI.

Reading Assignment

“Architectural Mismatch: Why it’s hard to build systems out of existing parts”

By: David Garlan, Robert Allen, and John Ockerbloom




Originally published in:
International Conference on Software Engineering (ICSE) 1995, pages 179-185.

Estimated reading time:
5½ pages, 50 min.

Summary

- Software Architecture important to reason about systems and understand strengths and weaknesses.
- Documenting software architectures: UML not widely used, but it is a good idea for consistency.
- 3 different types of views:
  - Module views
  - Component & Connector views
  - Allocation views

Thank you for your attention

Questions & suggestions can be sent to:
k.stol@ucc.ie