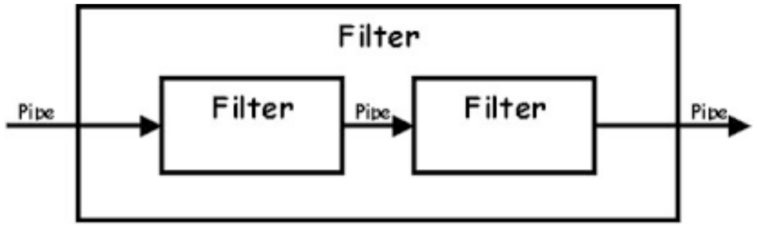
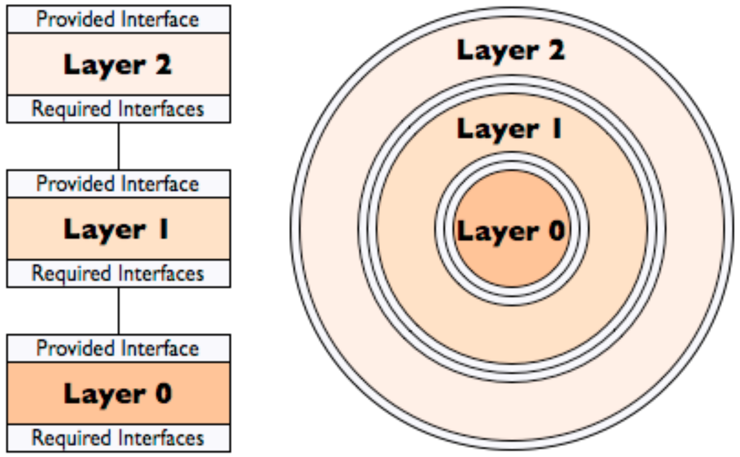




COMP 3004

Architectural Styles

Winter 2020

Instructor: Dr. Olga Baysal

Carleton
UNIVERSITY

Material and slides from:
Taylor et al.,
Reid Holmes

Topics

- **Software Architectural Styles**

Objectives

- What are the benefits / pitfalls of different architectural approaches?
- What are the phases of the design process?
- What are some alternative design strategies? When are they necessary?
- Define: abstraction, reification
- Identify key architectural style categories

Architectural Approaches

- **Creativity**
  - Engaging
  - Potentially unnecessary
  - Dangerous
- **Methodological**
  - Efficient when domain is familiar
  - Predictable outcome
  - Not always successful

Design Process

- **Feasibility stage**:
  - Identify a set of feasible concepts for the design as a whole
- **Preliminary design stage**:
  - Select and develop the best concept
- **Detailed design stage**:
  - Develop engineering descriptions of the concept
- **Planning stage**:
  - Evaluate and alter the concept to fit the requirements of production, distribution, consumption and product retirement

Potential Problems

- If the designer is unable to produce a set of feasible concepts, progress stops

- As problems and products increase in size and complexity, the probability that any one individual can successfully perform the first steps decreases

- As complexity increases or the experience of the designer is not sufficient, alternative approaches to the design process must be adopted

Alternative Design Strategies

- **Standard**
  - Linear model described earlier
- **Cyclic**
  - Process can revert to an earlier stage
- **Parallel**
  - Independent alternatives are explored in parallel
- **Adaptive** (“lay tracks as you go”)
  - The next design strategy of the design activity is decided at the end of a given stage
- **Incremental**
  - Each stage of development is treated as a task of incrementally improving the existing design

- The beast you fight: **Complexity**
  - A *complex* system can no longer be made by a single person
  - A very *complex* system can no longer be comprehended by a single person
- How to tackle complexity?

Identifying a Viable Strategy

- Use fundamental design tools: abstraction and modularity.
  - *But how?*
- Inspiration, where inspiration is needed. Predictable techniques elsewhere.
  - *But where is creativity required?*
- Applying own experience or experience of others.

Abstraction

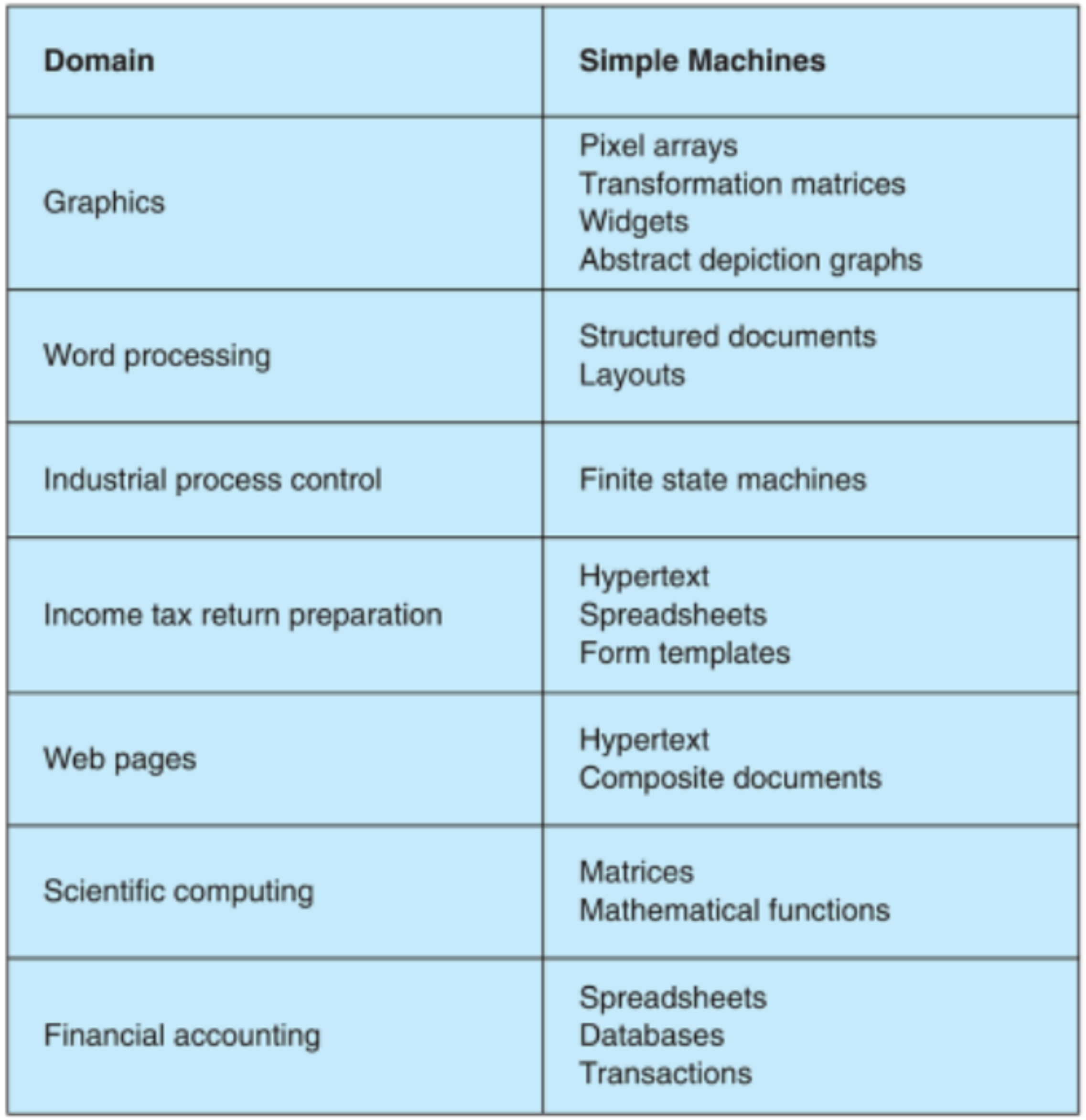
- **Definition**

  “A concept or idea not associated with a specific instance”

- Bottom up
  - Generalize “up” to concepts from details
- Top down
  - Specify “down” to details from concepts
- **Reification:**
  - “The conversion of a concept into a thing”

Abstraction and the Simple Machines

- Search for a simple machine that serves as an abstraction of a potential system that will perform the required task
- Every application domain has its common simple machines



Domain	Simple Machines
Graphics	Pixel arrays Transformation matrices Widgets Abstract depiction graphs
Word processing	Structured documents Layouts
Industrial process control	Finite state machines
Income tax return preparation	Hypertext Spreadsheets Form templates
Web pages	Hypertext Composite documents
Scientific computing	Matrices Mathematical functions
Financial accounting	Spreadsheets Databases Transactions

Taylor et al.

Level of Discourse

- Any attempt to use abstraction as a tool must choose a level of discourse, and once that is chosen, must choose the terms of discourse
- *Alternative 1:* consider application as a whole (e.g., step-wise refinement)
- *Alternative 2:* start with sub-problems
  - Combine solutions as they are ready
- *Alternative 3:* start with level above desired application
  - E.g., consider simple input as general parsing

Separation of Concerns

- Separation of concerns is the decomposition of a problem into independent parts

- In architecture, separating components and connectors

- The difficulties arise when the issues are either actually or apparently intertwined

- Separations of concerns frequently involves many tradeoffs

- Total independence of concepts may not be possible
  - *Scattering*: concern spread across many parts (e.g., logging)
  - *Tangling*: concern interacts with many parts (e.g., performance)

Architectural Style

- Recognize common patterns
  - build new systems as **variation** of old systems
- Selecting the right architecture
  - critical to success

Architectural Style

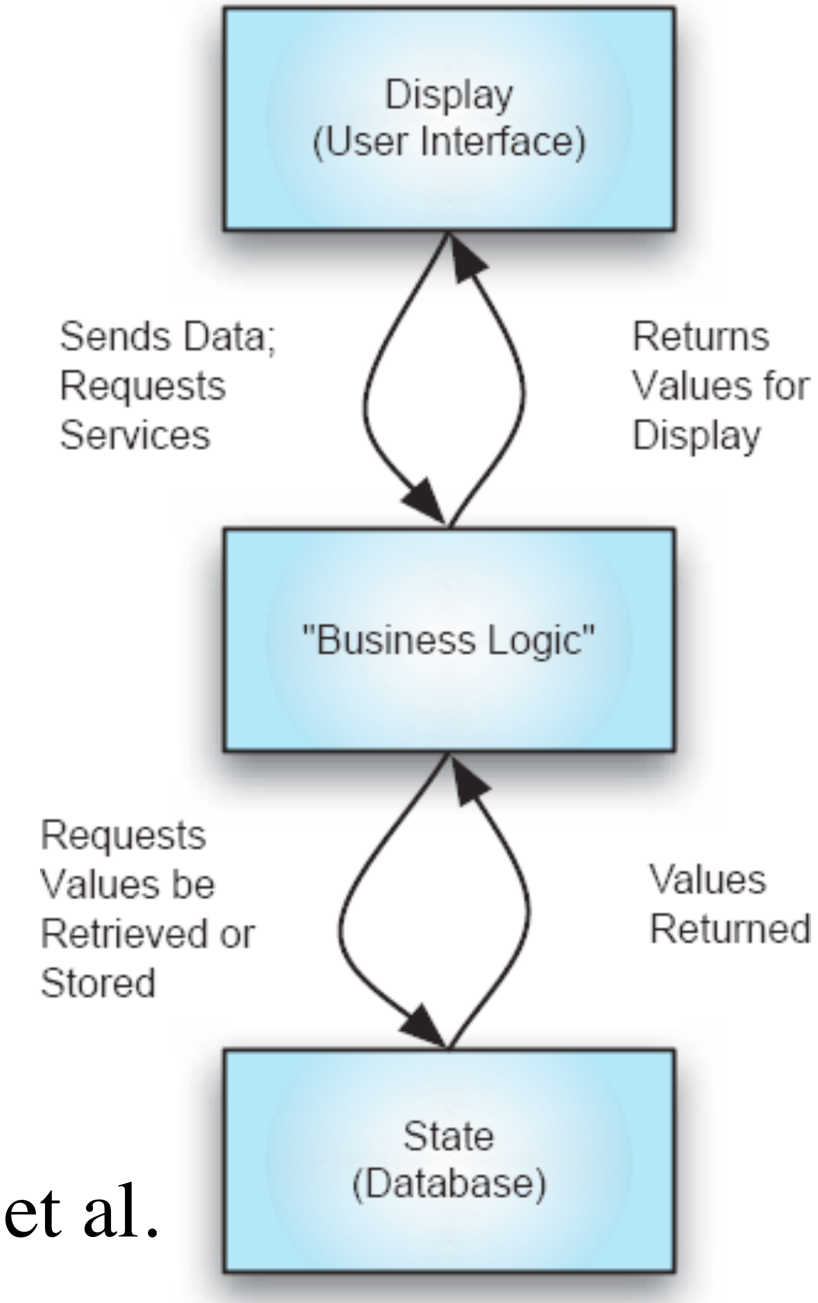
- **An architectural style** is a named collection of architectural design decisions that:
  - are applicable in a given development context
  - constrain design decisions
  - elicit beneficial qualities in each resulting system.

- Some design choices are better than others
  - Experience can guide us towards beneficial sets of choices (patterns) that have positive properties

Architectural Patterns

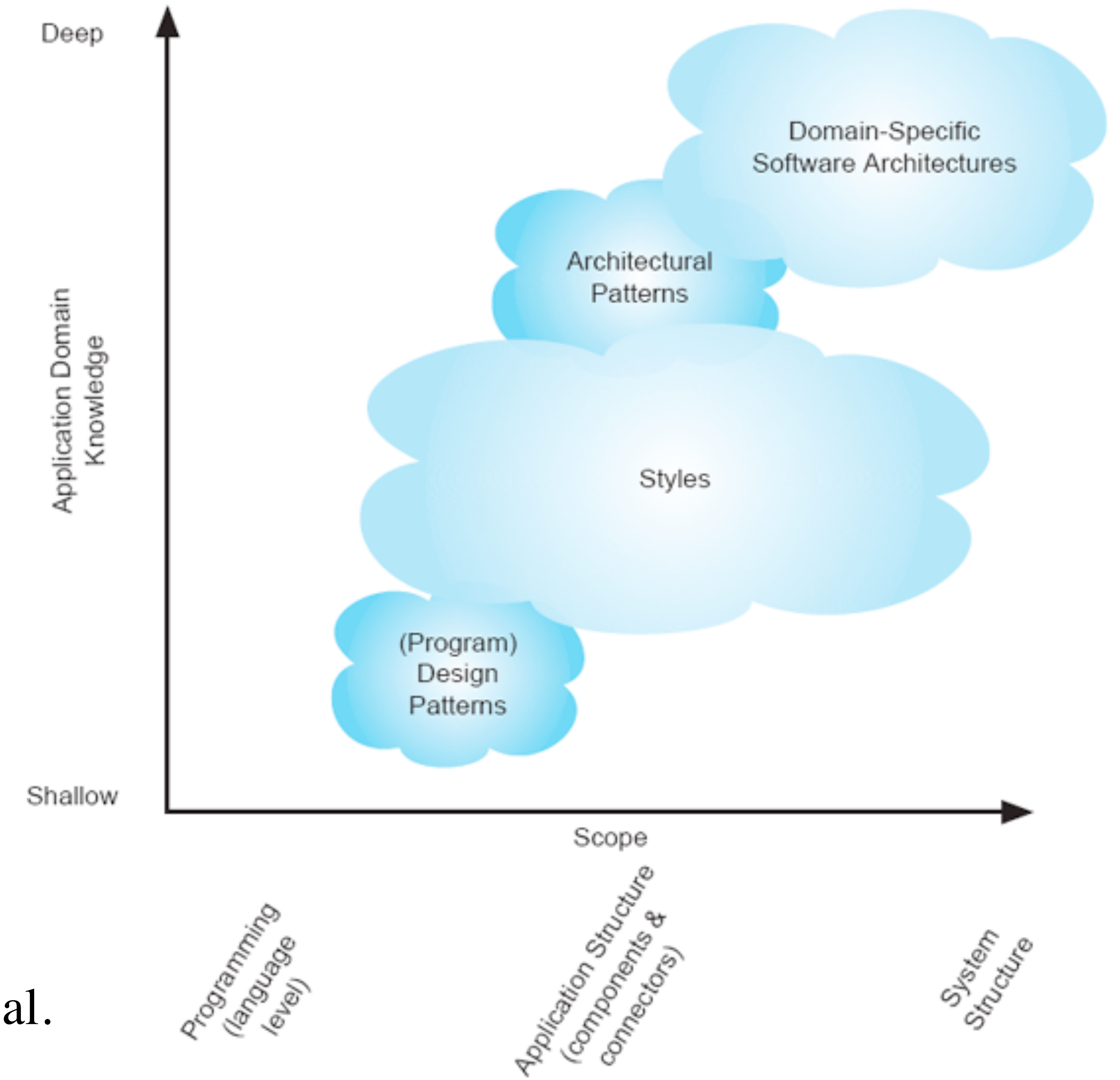
- A set of architectural design decisions that are applicable to a recurring design problem, and parameterized to account for different software development contexts in which that problem appears

- **Three-tiered** architectural pattern:
  - State-Logic-Display
  - Application examples:
    - Business applications
    - Multi-player games
    - Web-based applications


Display
(User Interface)
Sends Data; Requests Services
Returns Values for Display
"Business Logic"
Requests Values be Retrieved or Stored
Values Returned
State
(Database)

Taylor et al.

Styles vs. Patterns


Deep
Application Domain Knowledge
Shallow
Domain-Specific Software Architectures
Architectural Patterns
Styles
(Program) Design Patterns
Scope
Programming (language level)
Application Structure (components & connectors)
System Structure

Taylor et al.

Good Properties of an Architecture

- Result in a consistent set of principled techniques
- Resilient in the face of (inevitable) changes
- Source of guidance through product lifetime
- Reuse of established engineering knowledge

Basic Properties of Styles

- A **vocabulary** of design elements
  - Component and connector types; data elements
  - e.g., pipes, filters, objects, servers
- A set of **configuration rules**
  - Topological constraints that determine allowed compositions of elements
  - e.g., a component may be connected to at most two other components
- A **semantic interpretation**
  - Compositions of design elements have well-defined meanings
- Possible analyses of systems built in a style

Benefits of Using Styles

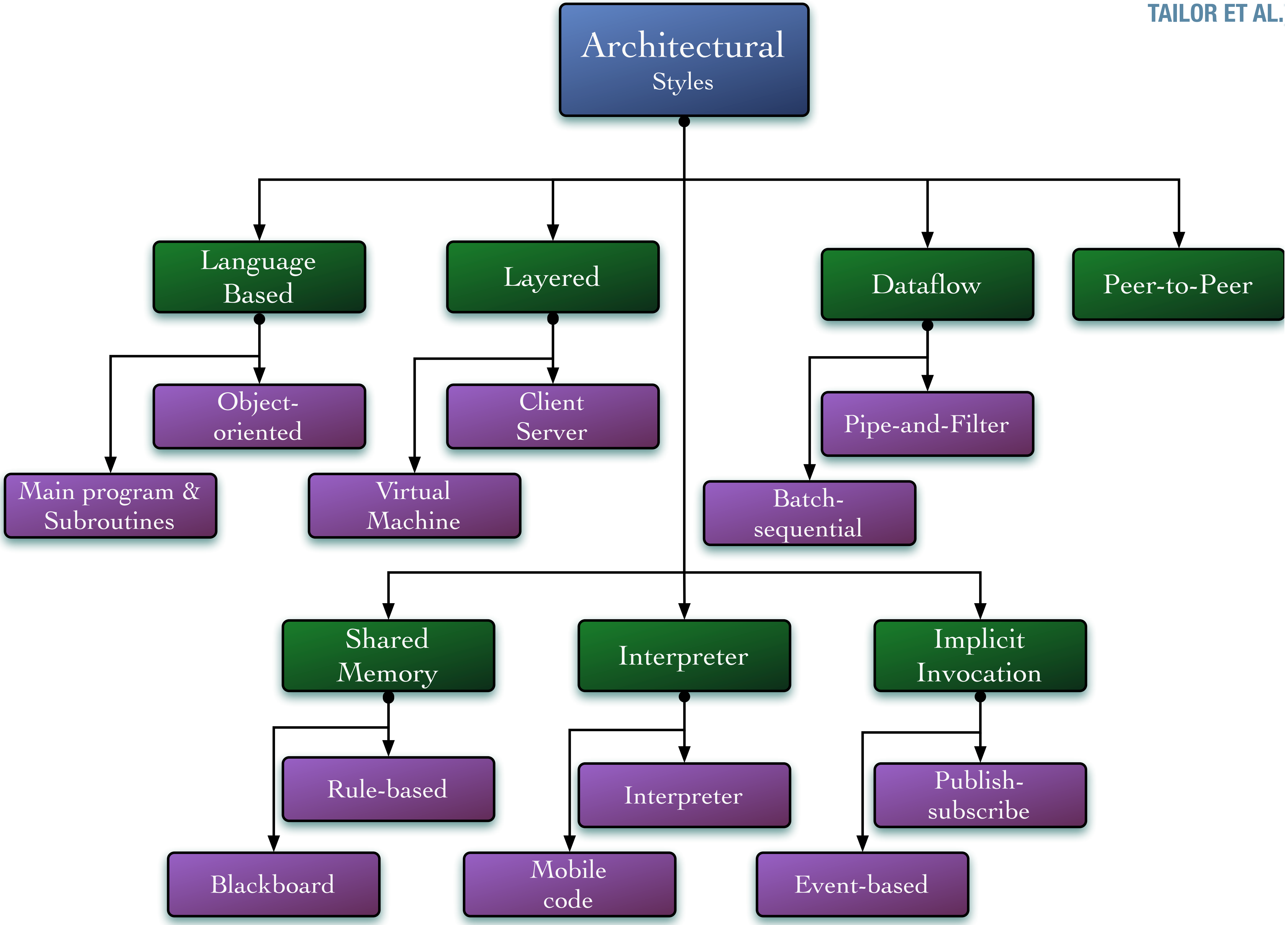
- **Design reuse**
  - Well-understood solutions applied to new problems
- **Code reuse**
  - Shared implementations of invariant aspects of a style
- **Understandability of system organization**
  - A phrase such as "client-server" conveys a lot of information
- **Reducing architectural drift and erosion**
- **Style-specific analyses**
  - Enabled by the constrained design space
- **Visualizations**
  - Style-specific depictions matching engineers' mental models

“Pure” Architectural Styles

- “Pure” architectural styles are rarely used in practice
- Systems in practice:
  - Regularly deviate from pure styles
  - Typically feature many architectural styles
- Architects must understand the “pure” styles to understand the strength and weaknesses of the style, as well as the consequences of deviating from the style

Architectural Styles

- TAILOR ET AL.



Next Class

- Architectural Styles [to be continued]