Material and some slide content from:
- Emerson Murphy-Hill
- Software Architecture: Foundations, Theory, and Practice
- Essential Software Architecture

Architectural Styles

Reid Holmes

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, and SoC
- Identify key architectural style categories

Architectural approaches

- Creative
  - Engaging
  - Potentially unnecessary
  - Dangerous
- Methodical
  - Efficient when domain is familiar
  - Predictable outcome
  - Not always successful

Design process

1. Feasibility stage:
   - Identify set of feasible concepts
2. Preliminary design stage:
   - Select and develop best concept
3. Detailed design stage:
   - Develop engineering descriptions of concept
4. Planning stage:
   - Evaluate / alter concept to fit requirements, also team allocation / budgeting

Abstraction

Definition:

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

Top down

> Specify ‘down’ to details from concepts

Bottom up

> Generalize ‘up’ to concepts from details

Reification:

> “The conversion of a concept into a thing”

Level of discourse

- Consider application as a whole
  - e.g., stepwise refinement
- Start with sub-problems
  - Combine solutions as they are ready
- Start with level above desired application
  - e.g., consider simple input as general parsing

Separation of Concerns

- Decomposition of problem into independent parts
- In arch, separating components and connectors
- Complicated by:
  - Scattering:
    - Concern spread across many parts
      - e.g., logging
  - Tangling:
    - Concern interacts with many parts
      - e.g., performance

Architectural styles

- Some design choices are better than others
  - Experience can guide us towards beneficial sets of choices (patterns) that have positive properties
    - Such as?
- An architectural style is a named collection of architectural design decisions that:
  - Are applicable to a given context
  - Constrain design decisions
  - Elicit beneficial qualities in resulting systems

Architectural styles

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.

e.g., Three-tier architectural pattern:

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

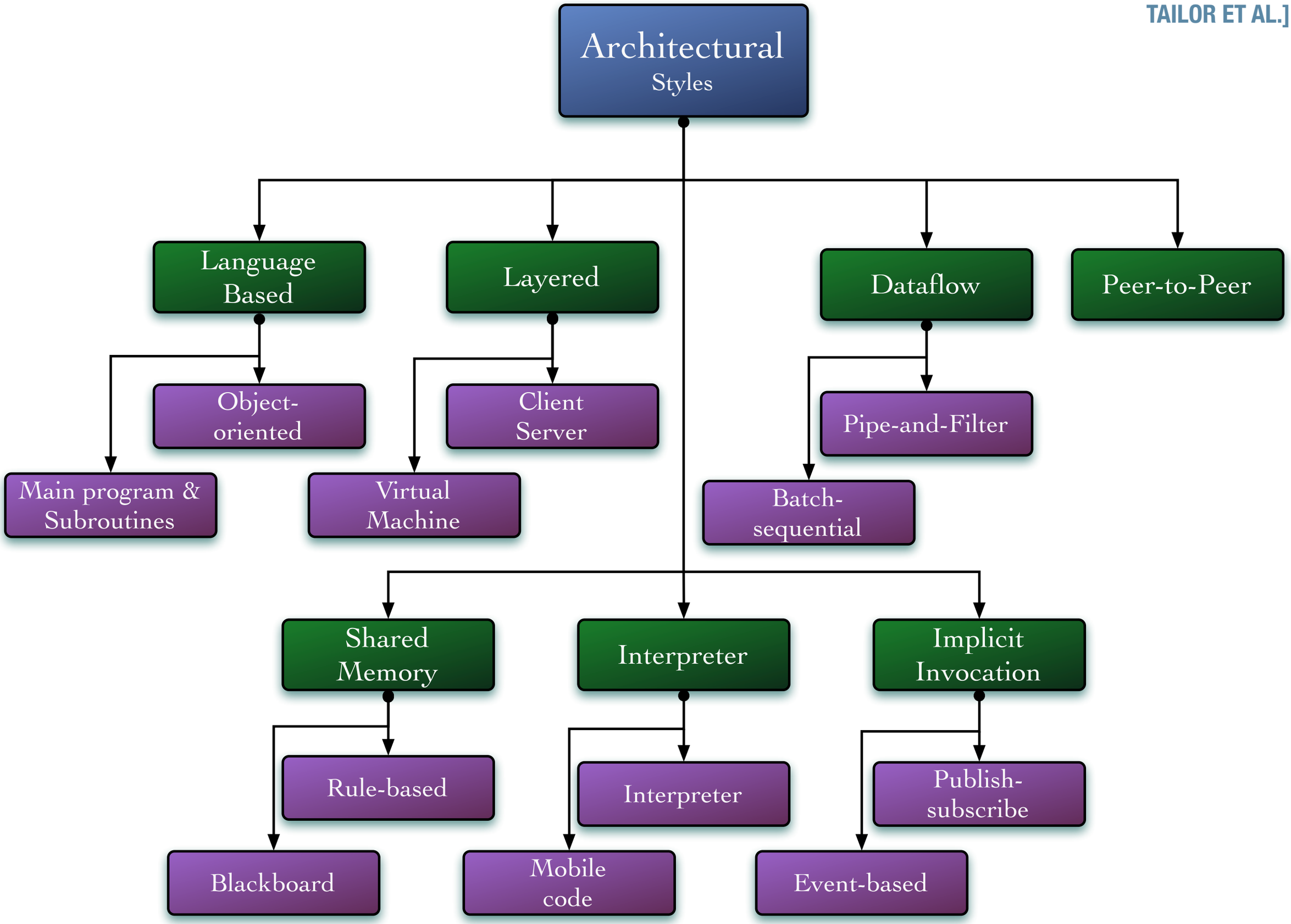
“Pure” architectural styles

- 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.

Role of context

- Neitzsche believed that all judgements were heavily dependent on individual perspective and that truth was the subject to interpretation
- The role of context is fundamental to the decisions surrounding your architecture
- Two very similar applications may require fundamentally different architectures for seemingly trivial reasons

[TOPOLOGY FROM TAILOR ET AL.]



Language-based

- Influenced by the languages that implement them
- Lower-level, very flexible
- Often combined with other styles for scalability

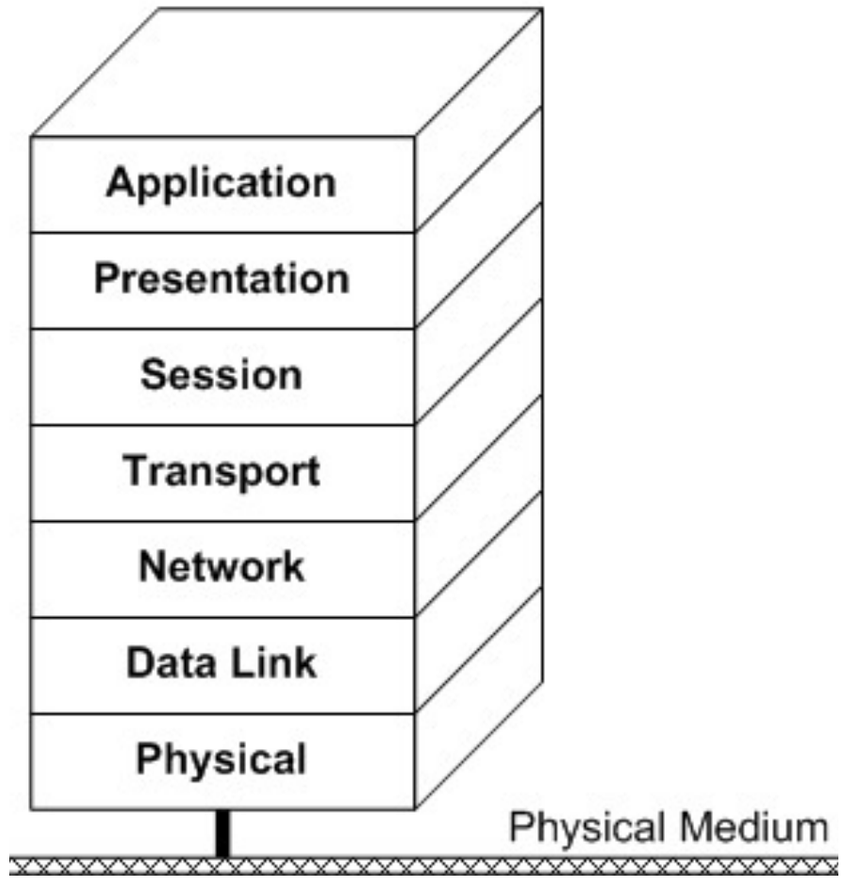
Examples:

Main & subroutine

Object-oriented

Layered

- Layered systems are hierarchically organized providing services to upper layers and acting as clients for lower layers
- Lower levels provide more general functionality to more specific upper layers
- In strict layered systems, layers can only communicate with adjacent layers



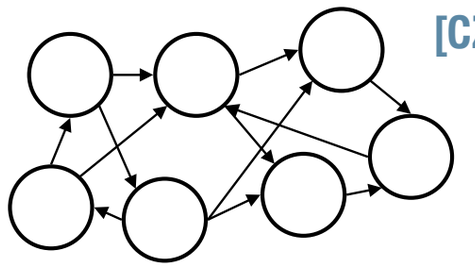
Examples:

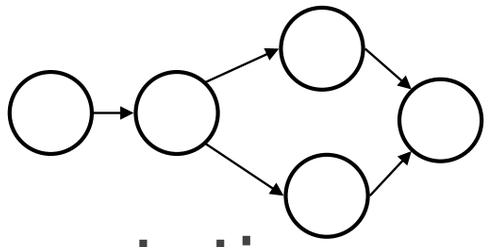
Virtual machine

Client-server

Dataflow

[CZARNECKI]





- A data flow system is one in which:
  - The availability of data controls computation
  - The structure of the design is determined by the orderly motion of data between components
  - The pattern of data flow is explicit
  - Variations:
    - Push vs. pull
    - Degree of concurrency
    - Topology

Examples:

Batch-sequential

Pipe-and-filter

Shared state

- Characterized by:
  - Central store that represents system state
  - Components that communicate through shared data store
- Central store is explicitly designed and structured

Examples:

Blackboard

Rule-based

Interpreter

- Commands interpreted dynamically
- Programs parse commands and act accordingly, often on some central data store

Examples:

Interpreter

Mobile code

Implicit invocation

- In contrast to other patterns, the flow of control is “reversed”
- Commonly integrate tools in shared environments
- Components tend to be loosely coupled
- Often used in:
  - UI applications (e.g., MVC)
  - Enterprise systems
    - (e.g., WebSphere)

Examples:

Publish-subscribe

Event-based

Peer to Peer

- Network of loosely-coupled peers
- Peers act as clients and servers
- State and logic are decentralized amongst peers
- Resource discovery a fundamental problem