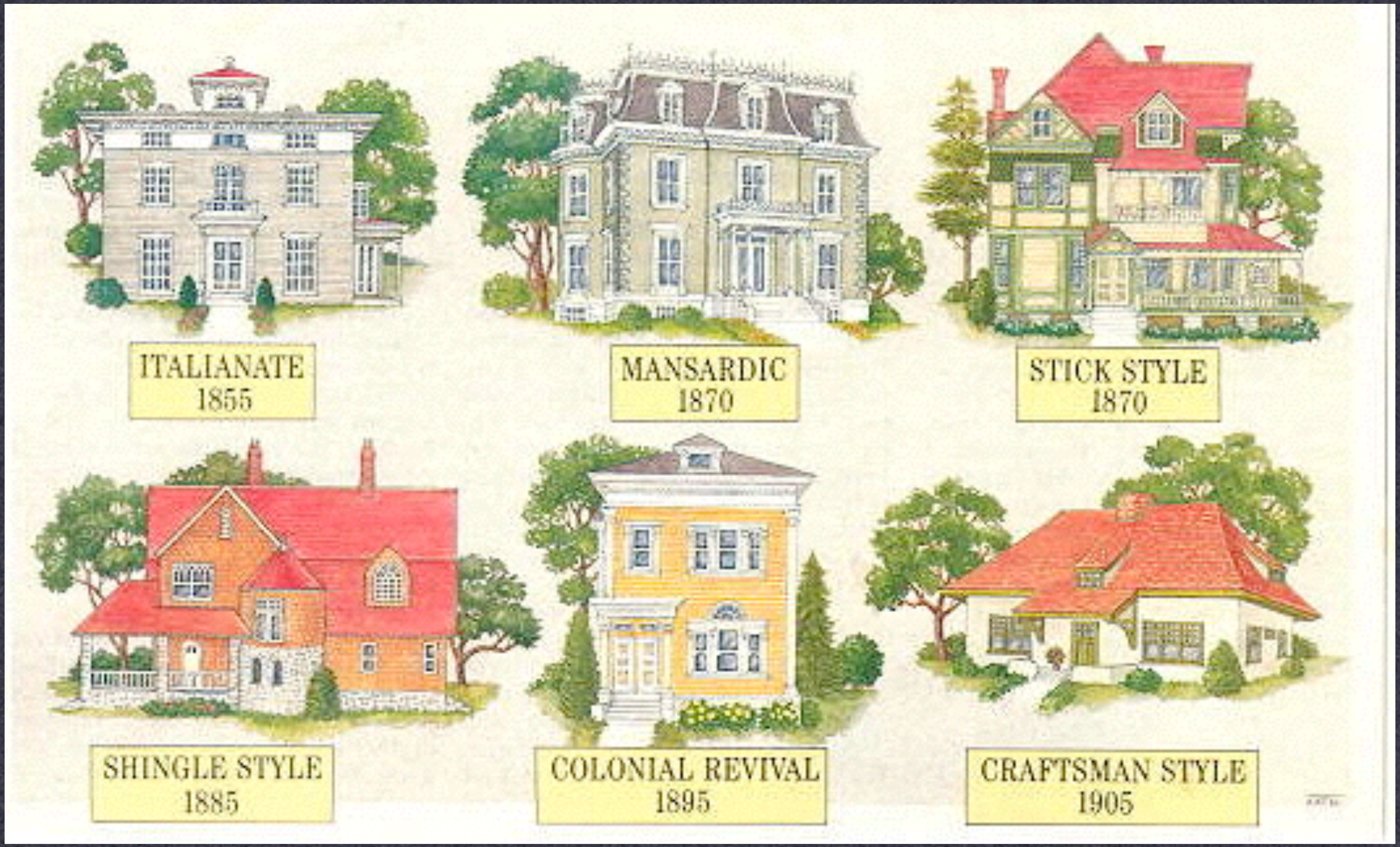
Material and some slide content from:
- Emerson Murphy-Hill
- Software Architecture: Foundations, Theory, and Practice
- Essential Software Architecture



Architectural Style Intro

Reid Holmes

BOLD == 2% PROJECT BONUS

- NomNom!
- TripJournal
- Let’s Do This
- Recipal
- Yippee!
- DSD
- **Locl**
- HeHeDelivery
- Quick Byte
- iLoc
- Tether
- **PEPPA**
- Food Oracle
- Spoon Feed Me
- Wutudu
- Kontribute
- Swap ‘Em
- DigiLib
- Hoot
- Alarm Plus
- Tuneline
- Where’d I Park
- LyfeCraft
- EventHub
- **Run To Click**
- Reflect
- Picture This!
- Grouplus
- Tracktr
- Trading Fever

Activity Review

- Reflect on the activities:
  - What was the point of each one?
  - What should I have learned?
  - What kinds of questions should I be able to answer based on what we did in class?

1/4

Architectural Analogy

- Kitchen design activity.
- What are the architectural components?
  - How are they related to each other?
  - What connectors exist?
  - Why did you choose they components / connectors / topology you did?
  - How do the connectors bind the components?
  - Why is software arch. like traditional arch.?
  - Why is software arch. not like traditional arch.?

2/4

Architectural Decomposition

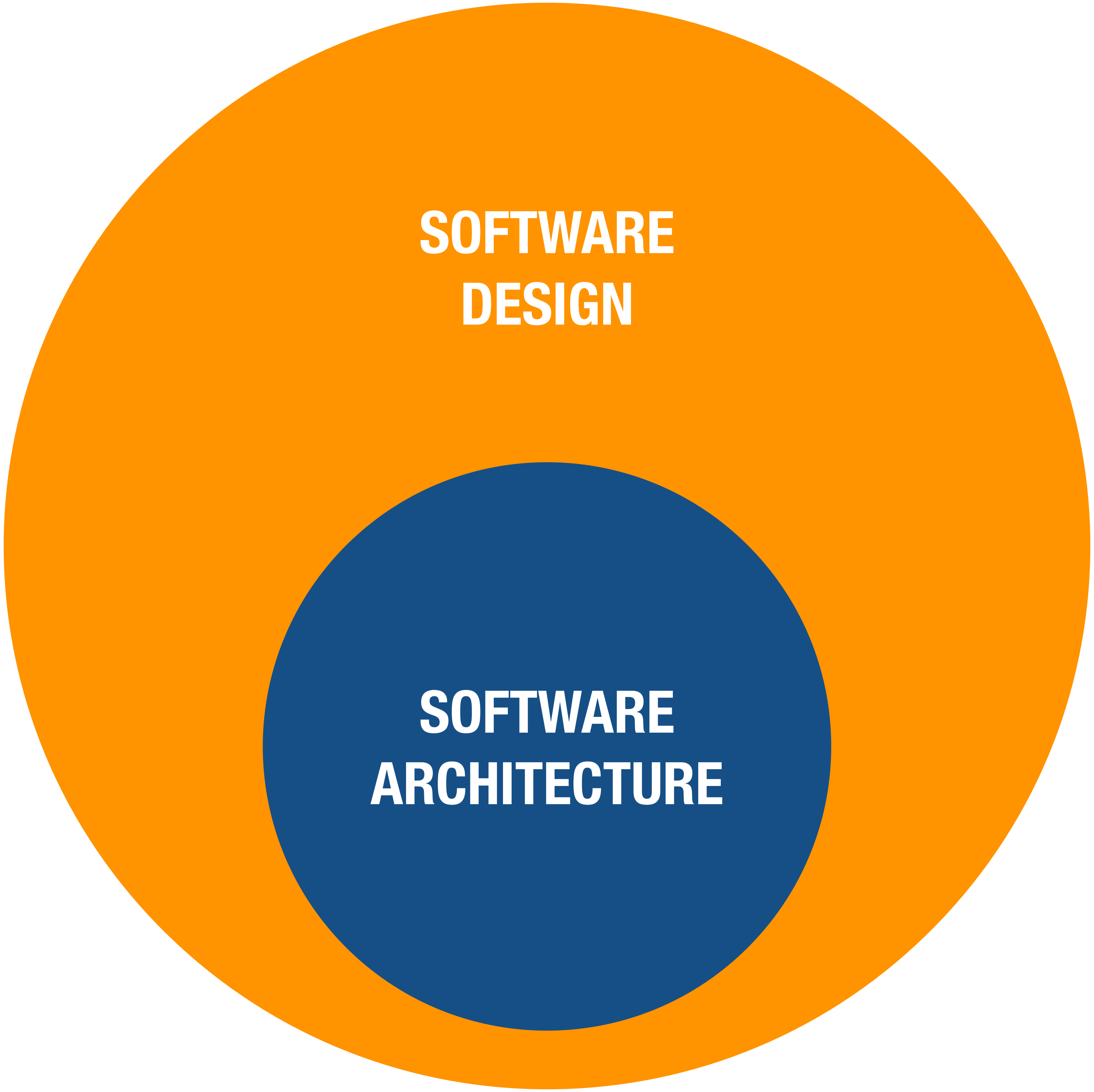
- Generate an architecture for an *automated shopping cart*.
  - Identify the key components and connectors.
  - Derive a system topology.
  - Justify your decomposition.
    - Why these components?
    - Does the architecture adequately capture the broad system goals?
    - What are the strengths and weaknesses of the proposed architecture?

Architectural Tradeoffs

- Generate an architecture for a *context-aware notification system.*
  - Identify NFPs for a given stakeholder.
    - Justify why those NFPs matter.
  - Determine how those NFPs influence the architecture of the system.
  - Compare the architectures derived when different stakeholders care about divergent NFPs.
  - Understand how NFPs can be in tension with each other.

Completeness & Consistency

- The Spec is Right.
  - For a given system description, can we identify:
    - Aspects that are inconsistent
    - Aspects that are incomplete
  - How can we build a description that all stakeholders can understand and reason about?
  - What is the right level of abstraction for an architectural document?
  - What tools and techniques can help us generate complete and consistent system descriptions?


SOFTWARE
DESIGN
SOFTWARE
ARCHITECTURE

[TAILOR ET AL.]

Architectural styles

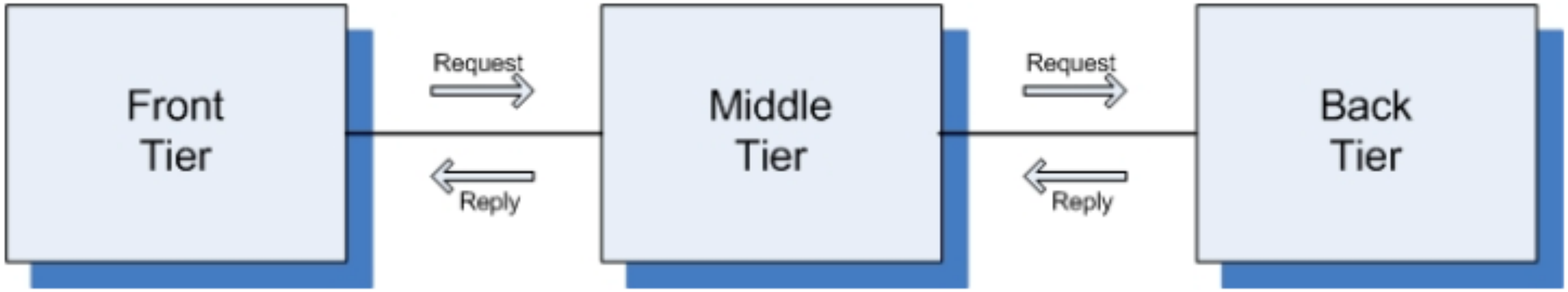
- Some design choices are better than others
  - Experience can guide us towards beneficial sets of choices (patterns) that have positive properties
- An architectural style is a named collection of architectural design decisions that:
  - Are applicable to a given context
  - Constrain design decisions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

[TAILOR ET AL.]

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:



[CZARNECKI]

Architectural styles

- Defines a family of architectures that are constrained by:
  - Component/connector vocabulary
  - Topology
  - Semantic constraints
- When describing styles diagrammatically:
  - Nodes == components (e.g., procedures, modules, processes, databases, …)
  - Edges == connectors (e.g., procedure calls, events, db queries, pipes, …)

[CZARNECKI]

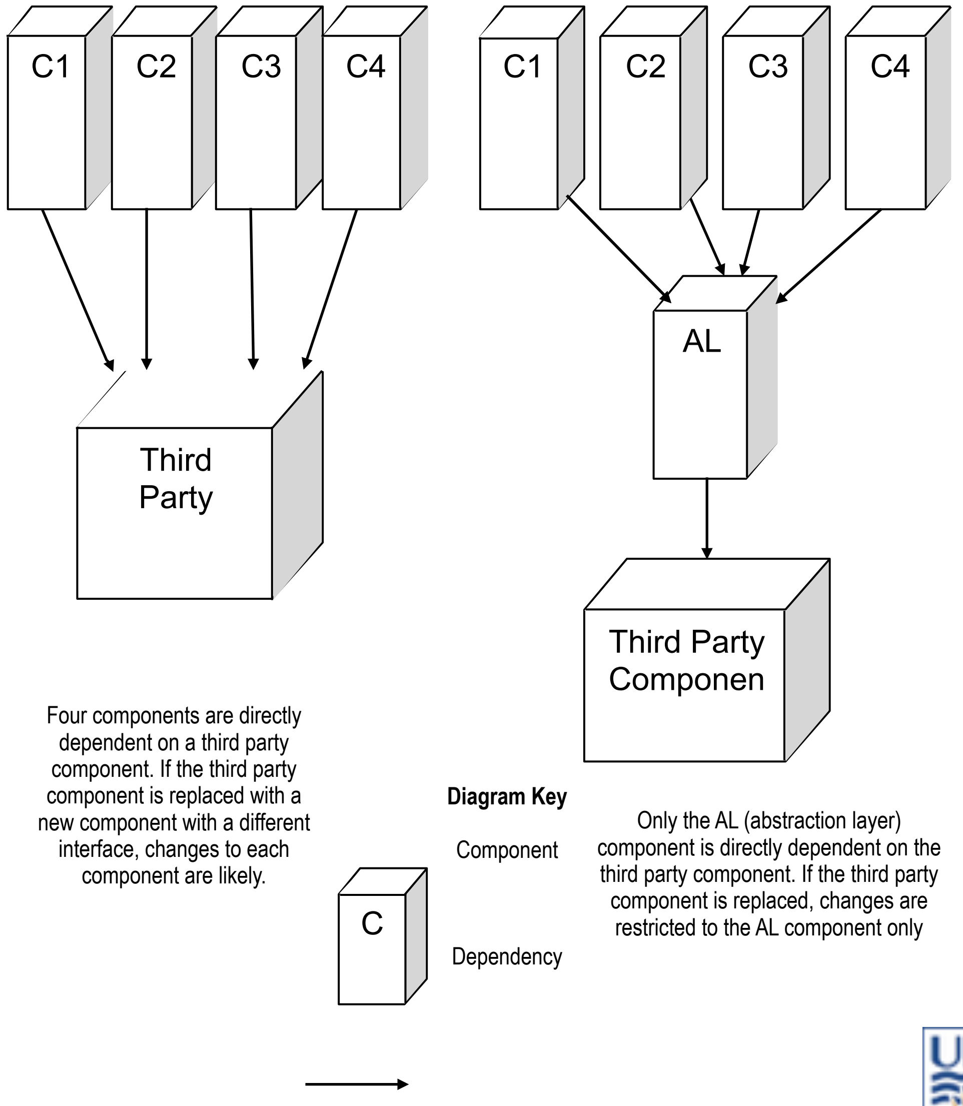
Understanding a style

- What is the structural pattern?
- What is the underlying computational model?
- What are the essential invariants of the style?
- What are some common usage examples?
- What are the style's advantages and disadvantages?
- What are some common specializations?

[GORTON]

Structure and Dependencies

- All styles minimize coupling in a specific way
- Excessive dependencies are not a good idea.
- Key issue:
  - Identifying likely change points.
  - Reduce direct dependencies on these points.



Four components are directly dependent on a third party component. If the third party component is replaced with a new component with a different interface, changes to each component are likely.

Only the AL (abstraction layer) component is directly dependent on the third party component. If the third party component is replaced, changes are restricted to the AL component only

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

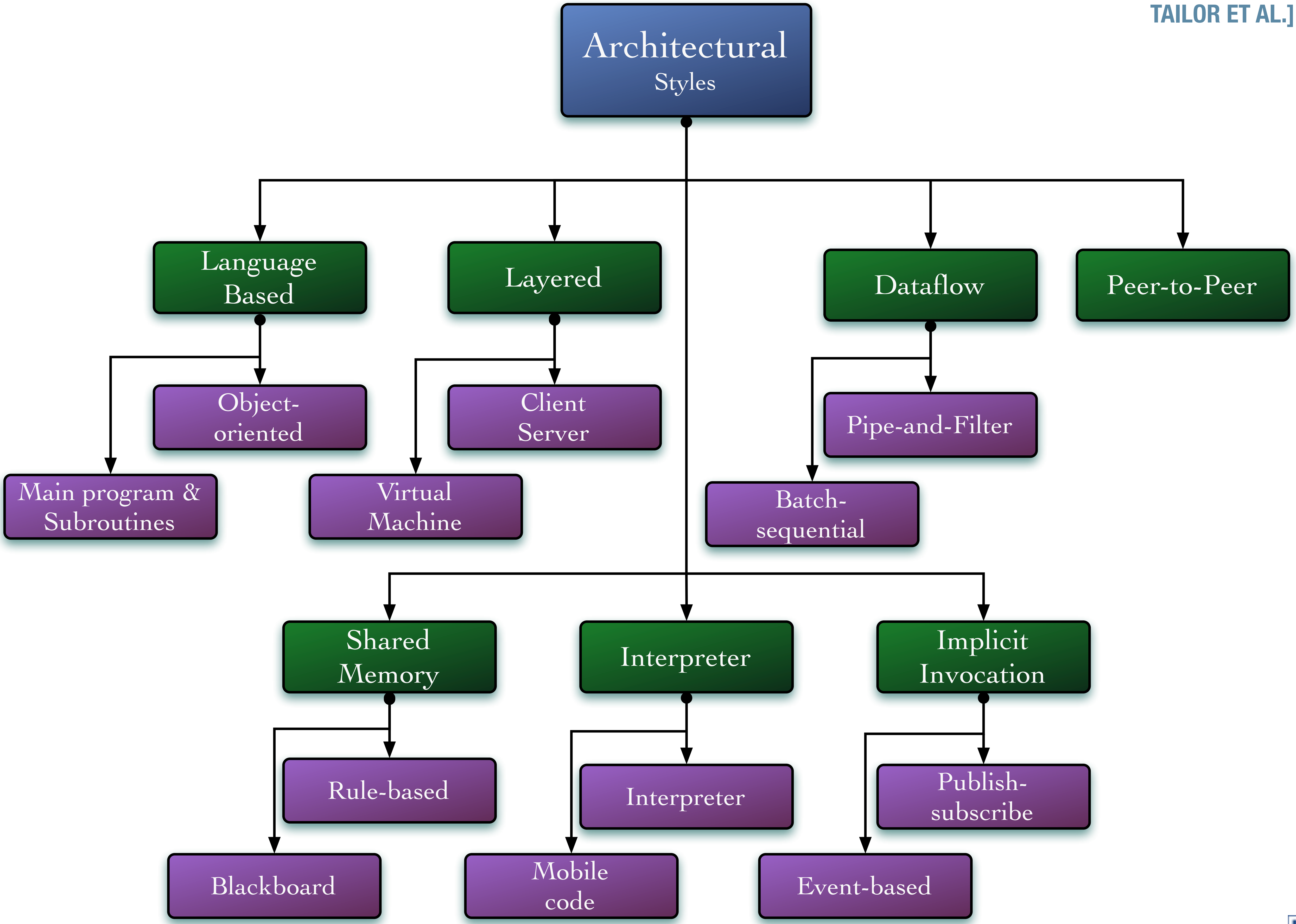
[CZARNECKI]

“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

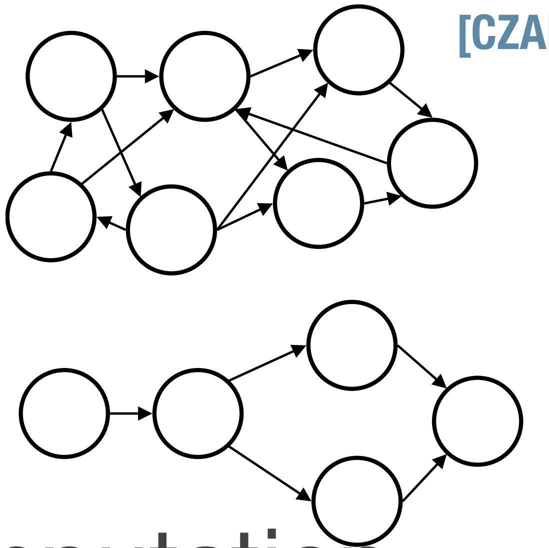
Examples:

Main & subroutine

Object-oriented

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:

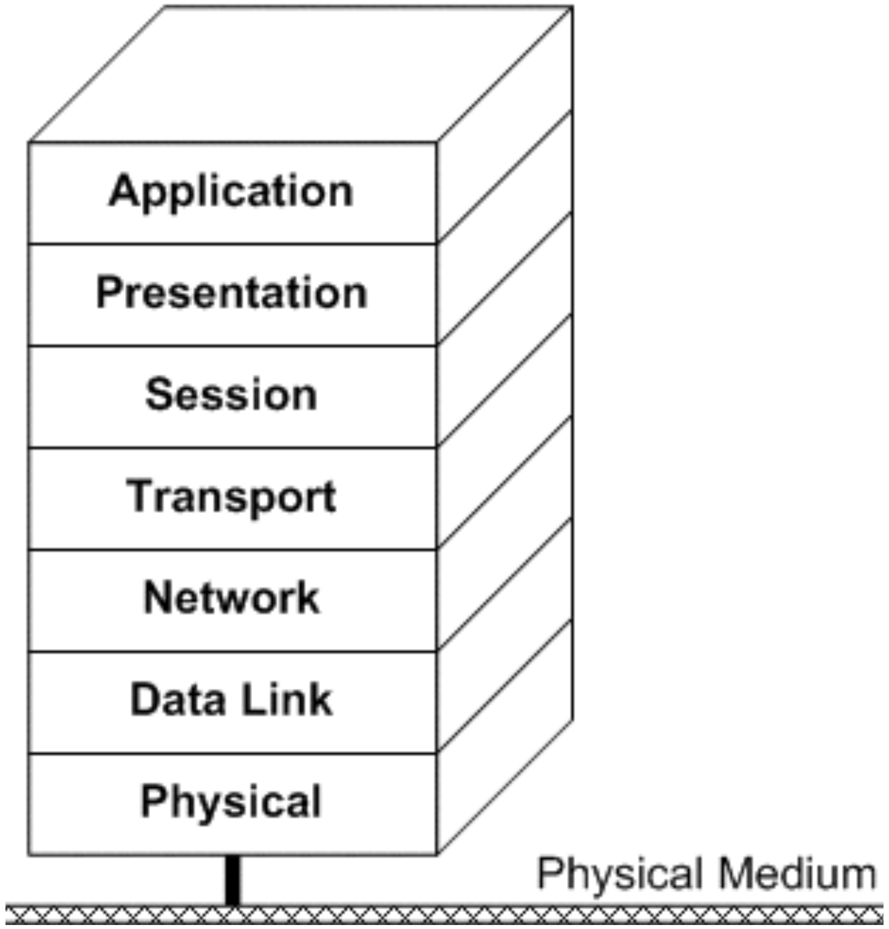
Blackboard

Rule-based

[CZARNECKI]

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

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