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

Architectural Styles

Reid Holmes

AV contact

- Dwight Schmidt <de2schmi@uwaterloo.ca> will be managing AV for the demos
- It would be nice if each group emailed him to let him know what kind of devices you'll be using for both the prototypes and the final demos
- Let him know:
  - That you're from CS 446
  - What make/model devices you will use
  - If your device has any video-out capabilities

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

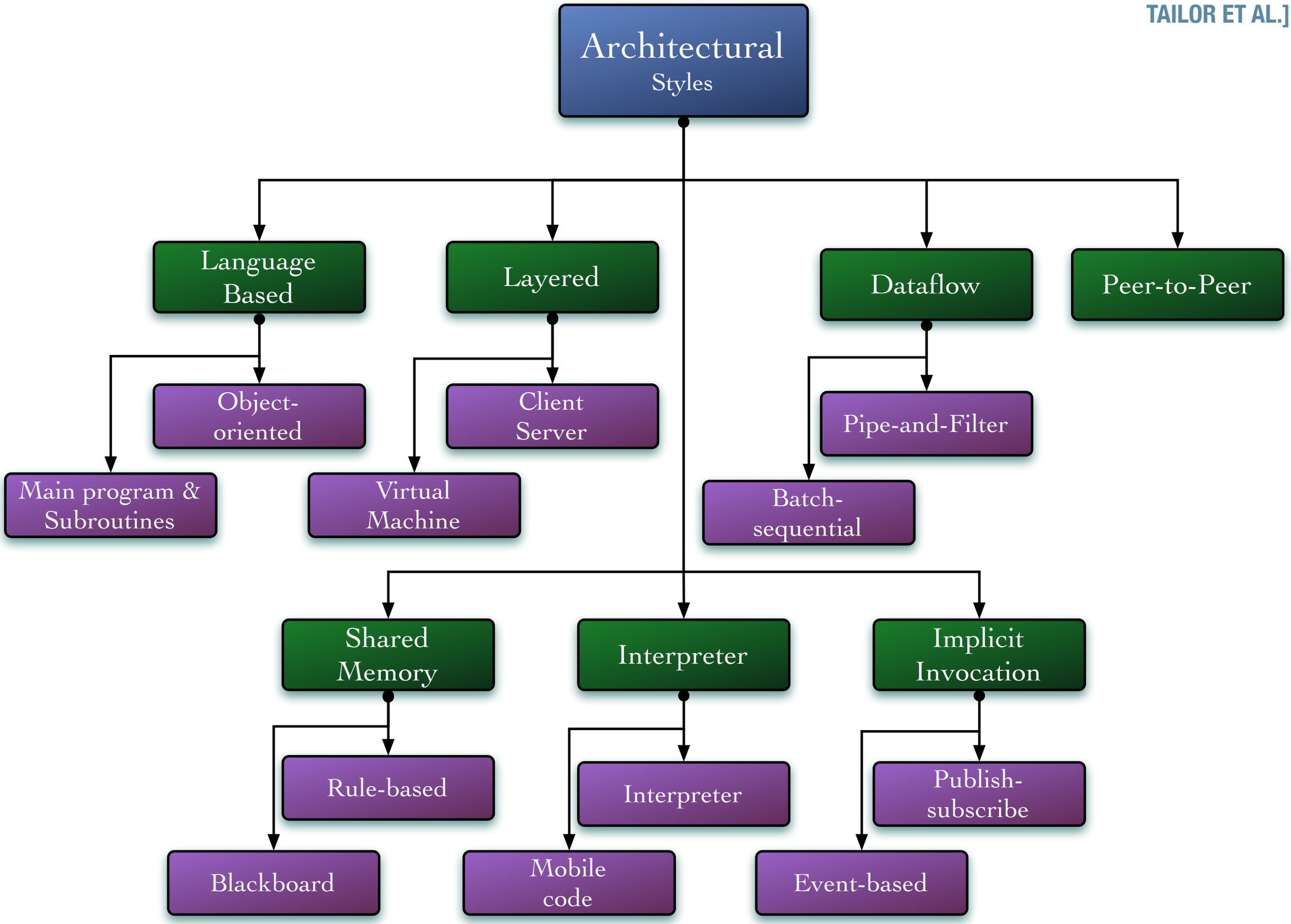
“Pure” architectural styles

- 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.]



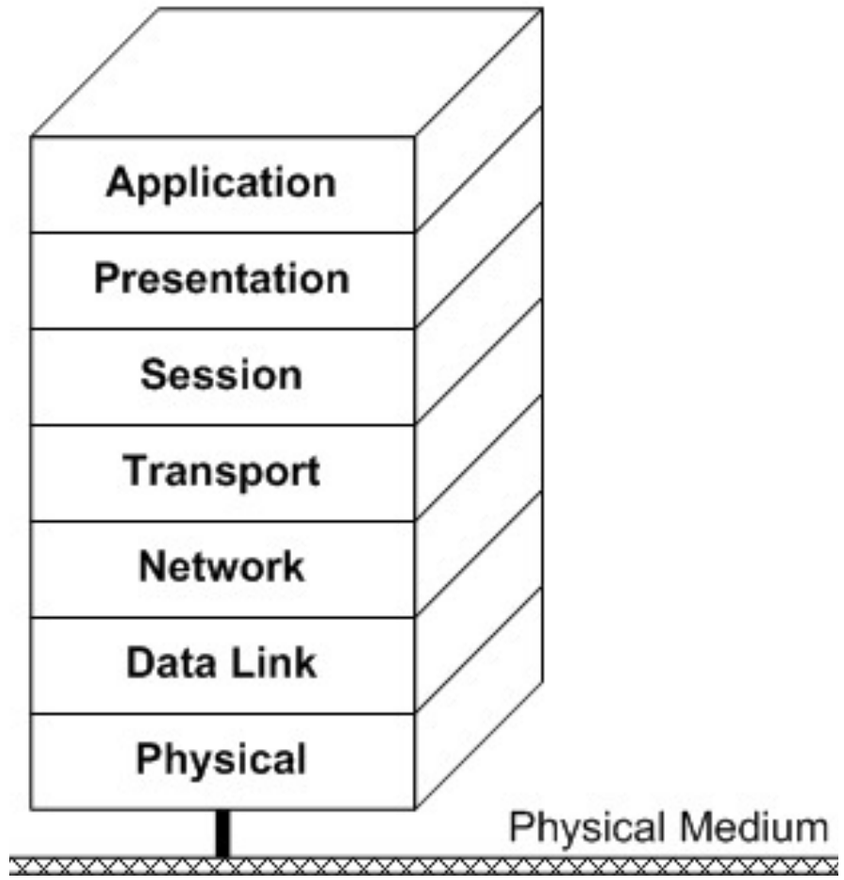
Language-based

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

Examples:

Layered

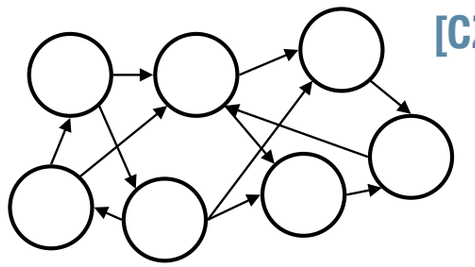
- Layered systems are hierarchically organized providing services to upper layers and acting as clients for lower layers
- Lower levels provide more general functionality to more specific upper layers
- In strict layered systems, layers can only communicate with adjacent layers

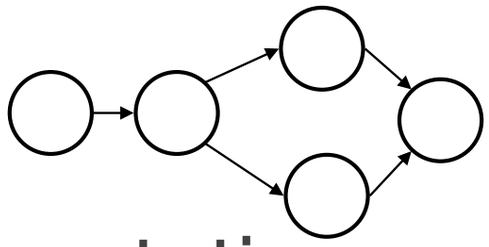


Examples:

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:

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:

Interpreter

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

Examples:

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:

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

Style: Client-server

Style: Client-server

- Clients communicate with server which performs actions and returns data. Client initiates communication.
- Components:
  - Clients and server.
- Connections:
  - Protocols, RPC.
- Data elements:
  - Parameters and return values sent / received by connectors.
- Topology:
  - Two level. Typically many clients.

Style: Client-server

- Additional constraints:
  - Clients cannot communicate with each other.
- Qualities:
  - Centralization of computation. Server can handle many clients.
- Typical uses:
  - Applications where: client is simple; data integrity important; computation expensive.
- Cautions:
  - Bandwidth and lag concerns.

Style: Blackboard

Style: Blackboard

- Independent programs communicate exclusively through shared global data repository.
- Components:
  - Independent programs (knowledge sources), blackboard.
- Connections:
  - Varies: memory reference, procedure call, DB query.
- Data elements:
  - Data stored on blackboard.
- Topology:
  - Star; knowledge sources surround blackboard.

Style: Blackboard

- Variants:
  - Pull: clients check for blackboard updates.
  - Push: blackboard notifies clients of updates.
- Qualities:
  - Efficient sharing of large amounts of data. Strategies to complex problems do not need to be pre-planned.
- Typical uses:
  - Heuristic problem solving.
- Cautions:
  - Not optimal if regulation of data is needed or the data frequently changes and must be updated on all clients.

Style: Publish-subscribe

Style: Publish-subscribe

- Subscribers register for specific messages or content. Publishers maintain registrations and broadcast messages to subscribers as required.
- Components:
  - Publishers, subscribers, proxies.
- Connections:
  - Typically network protocols.
- Data elements:
  - Subscriptions, notifications, content.
- Topology:
  - Subscribers connect to publishers either directly or through intermediaries.

Style: Publish-subscribe

- Variants:
  - Complex matching of subscribers and publishers can be supported via intermediaries.
- Qualities:
  - Highly-efficient one-way notification with low coupling.
- Typical uses:
  - News, GUI programming, network games.
- Cautions:
  - Scalability to large numbers of subscriber may require specialized protocols.

Style: Event-based

Style: Event-based

- Independent components asynchronously emit and receive events.
- Components:
  - Event generators / consumers.
- Connections:
  - Event bus.
- Data elements:
  - Events.
- Topology:
  - Components communicate via bus, not directly.

Style: Event-based

- Variants:
  - May be push or pull based (with event bus).
- Qualities:
  - Highly scalable. Easy to evolve. Effective for heterogenous applications.
- Typical uses:
  - User interfaces. Widely distributed applications (e.g., financial markets, sensor networks).
- Cautions:
  - No guarantee event will be processed. Events can overwhelm clients.

Style: Mobile code

Style: Mobile code

- Code and state move to different hosts to be interpreted.
- Components:
  - Execution dock, compilers / interpreter.
- Connections:
  - Network protocols.
- Data elements:
  - Representations of code, program state, data.
- Topology:
  - Network.

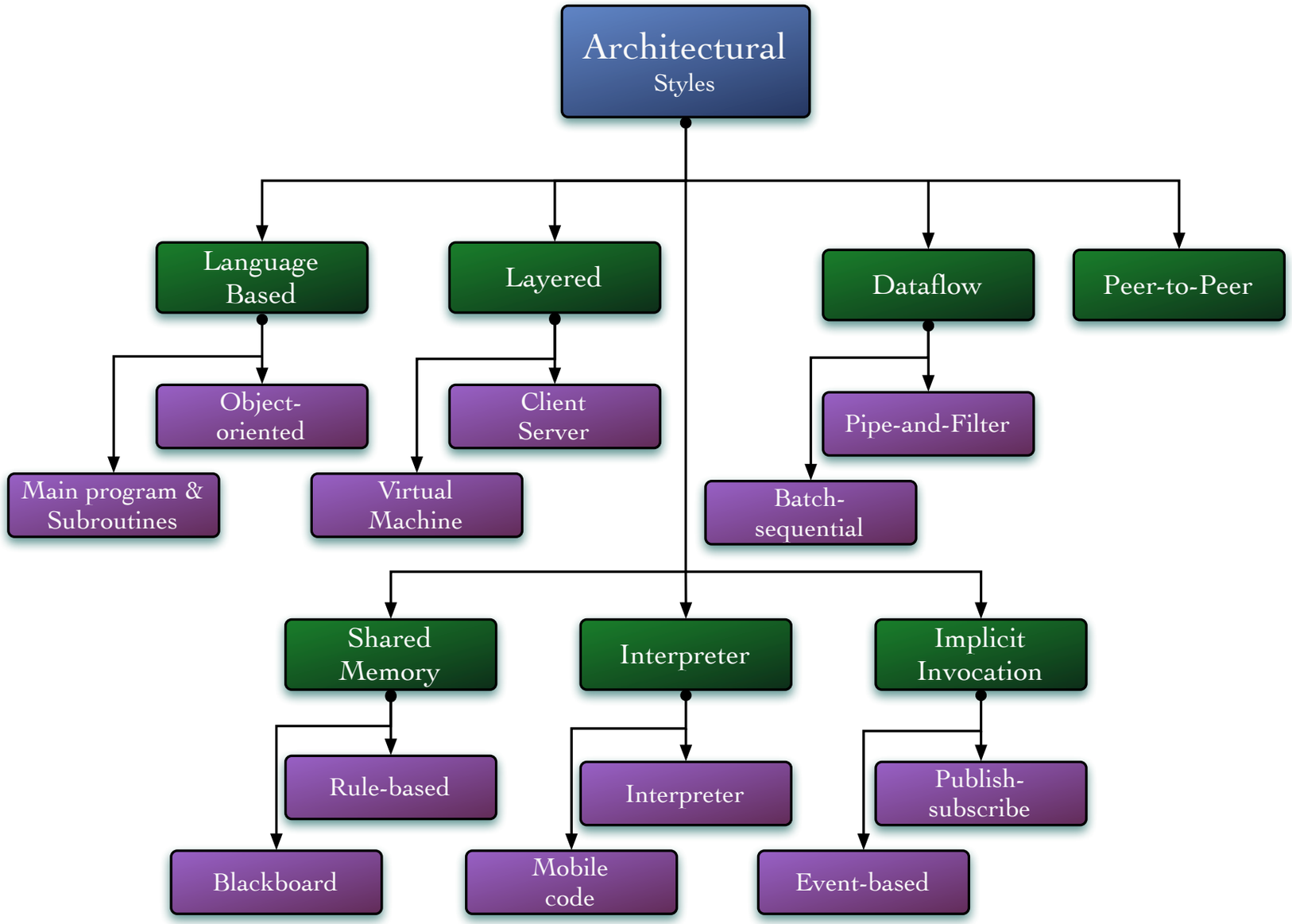
Style: Mobile code

- Variants:
  - Code-on-demand, remote evaluation, and mobile agent.
- Qualities:
  - Dynamic adaptability.
- Typical uses:
  - For moving code to computing locations that are closer to the large data sets being operated on.
- Cautions:
  - Security. Transmission costs. Network reliability.

[TOPOLOGY FROM TAILOR ET AL.]

Activity

- Design using an assigned pattern.
  - What are the components, connectors, and topology?



Activity followup

- Discussion revealed that designing FB using:
  - Event-based
  - Blackboard
  - Pipe-and-filter
  - Main and subroutine