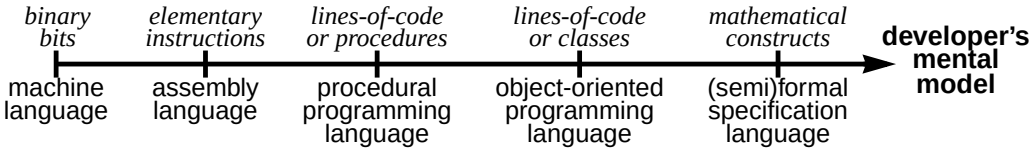
Review — Software Architecture Goals

- Control inherent software complexity
  - elevate abstraction levels
  - match developers' mental models



- Explicitly address a system's conceptual underpinnings
  - act on the blueprint instead of the system itself
  - address complexity
  - increase reuse and component marketplace potential
  - reduce development costs
  - shift development approach to a component-based philosophy

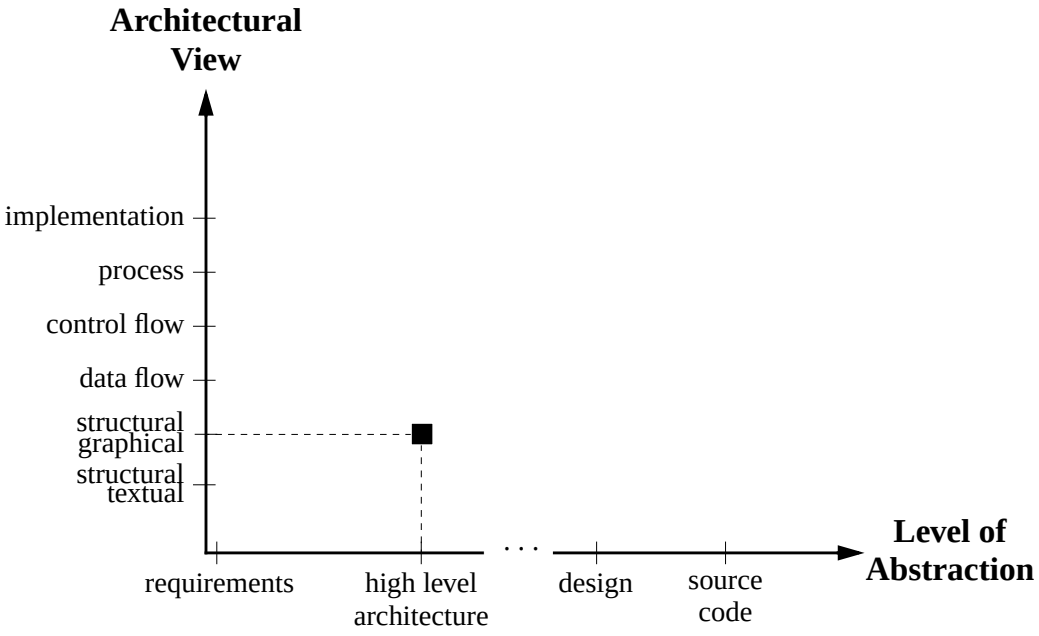
Review — Focus of Architectures

- System structure
- System-level properties
- Key role in the software lifecycle
  - a framework for satisfying requirements
  - technical basis for design
  - managerial basis for cost estimation & process management
  - effective basis for reuse
  - basis for consistency and dependency analysis
  - basis for implementation

Review — Definitions

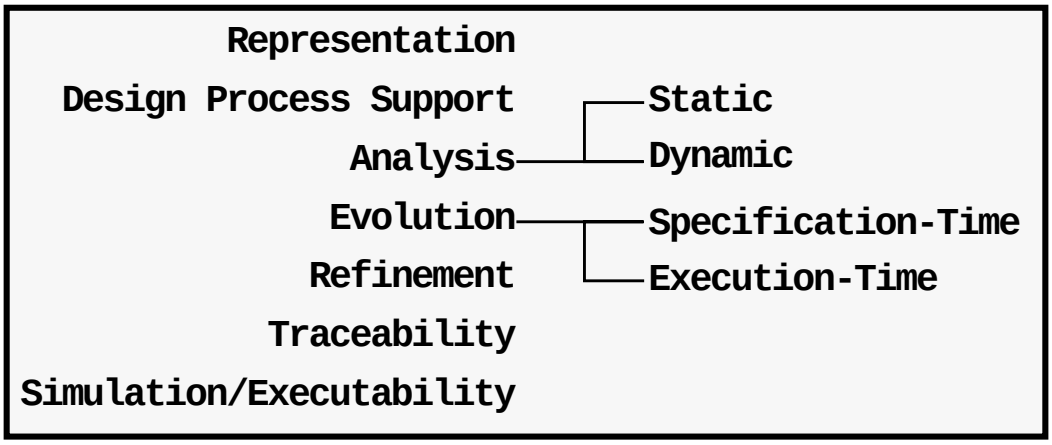
- Perry and Wolf
  - Software Architecture = { Elements, Form, Rationale }
- Shaw and Garlan
  - Software architecture [is a level of design that] involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns.
- Canonical building blocks
  - *component* — locus of computation and state
  - *connector* — element that models interactions among components and rules that govern those interactions
  - *configuration* — connected graph of components and connectors which describes architectural structure

Review — Architectural Perspectives



Review — Architectural Domains

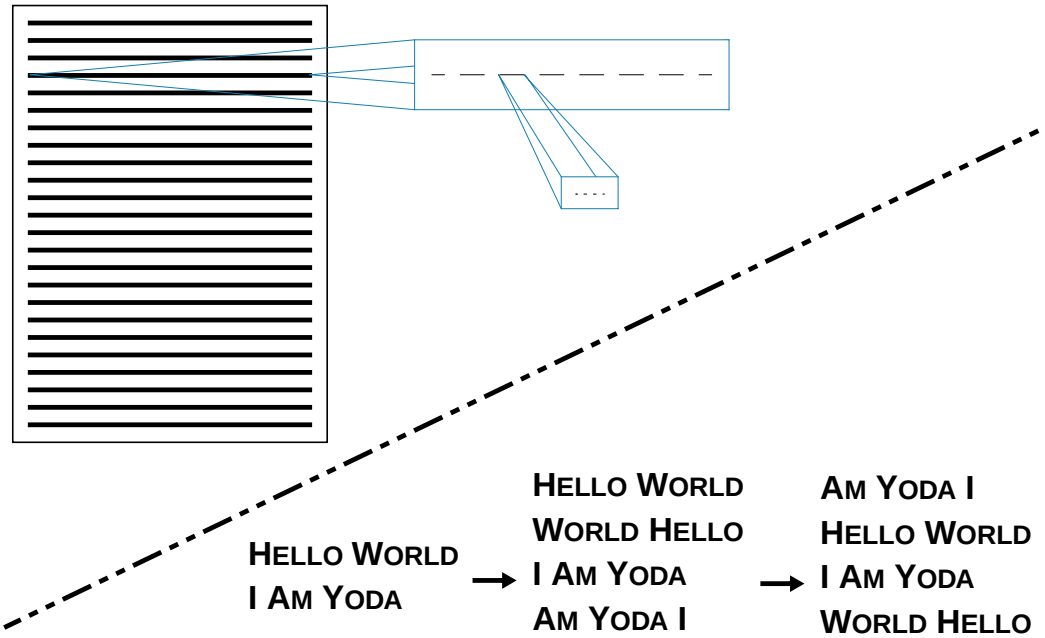
- Classes of problems or areas of concern in architecture



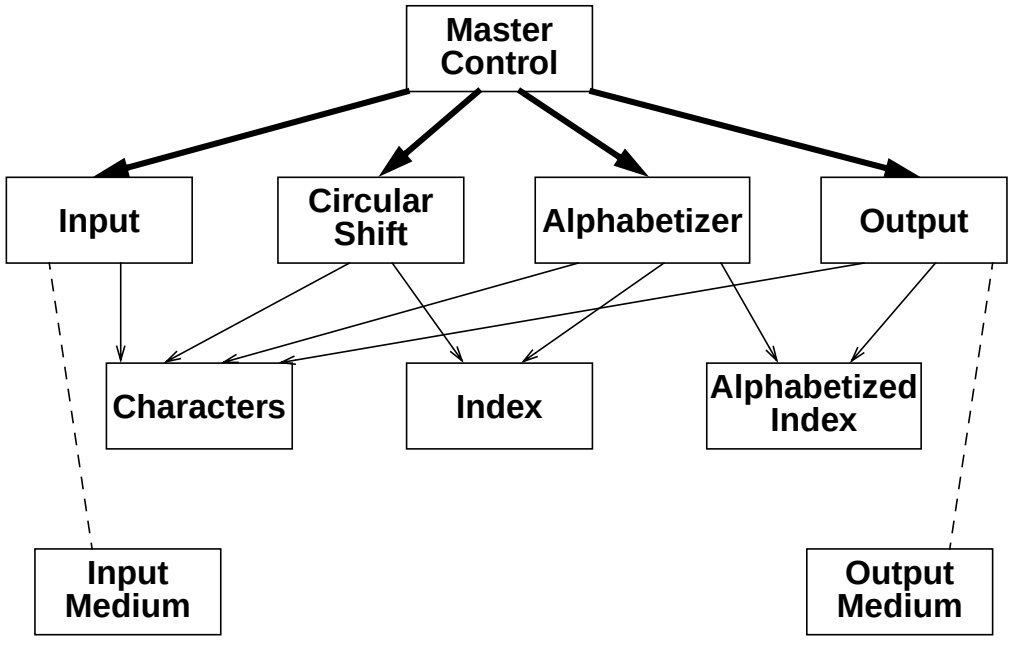
Scope of Software Architectures

- Every system has an architecture
- Details of the architecture are a reflection of system requirements and trade-offs that made to satisfy them
- Possible decision factors
  - performance
  - compatibility with legacy software
  - planning for reuse
  - distribution profile
    - current and future
  - safety, security, fault tolerance
  - evolvability
    - changes to processing algorithms
    - changes to data representation
    - modifications to the structure/functionality

Case Study: Key Word In Context



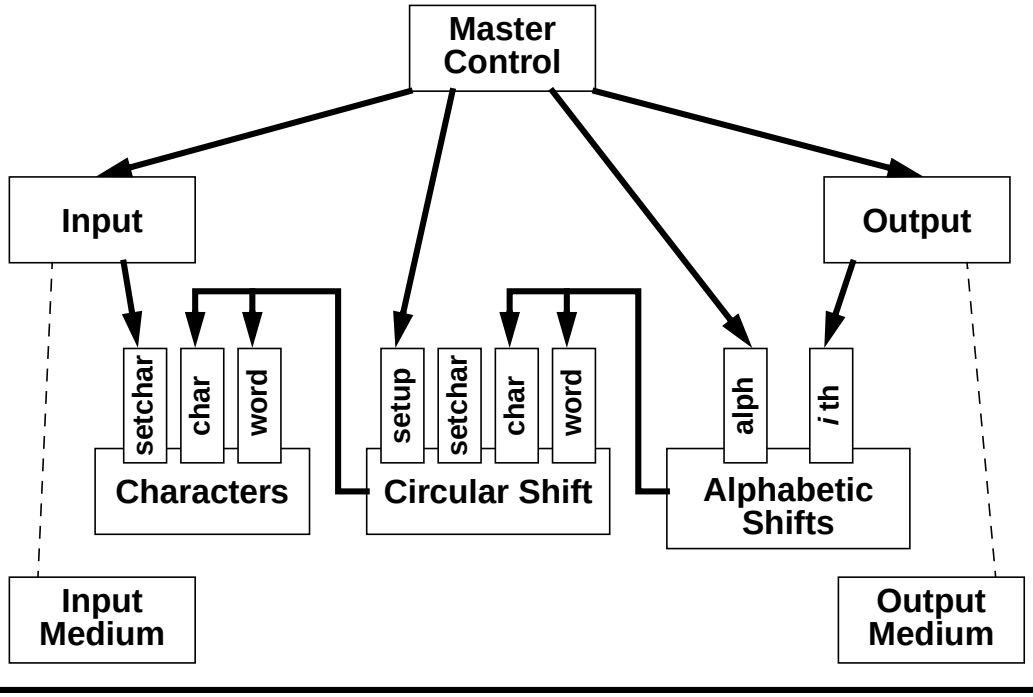
KWIC Architecture: Shared Data



Shared Data Pros and Cons

+ Efficiency
  - → shared data
  - → efficient data representation
  - → sequential data access
+ Intuitive structure
- Changeability
  - → data format not abstracted away
  - → functional elements dependent on data representation
- Support for reuse

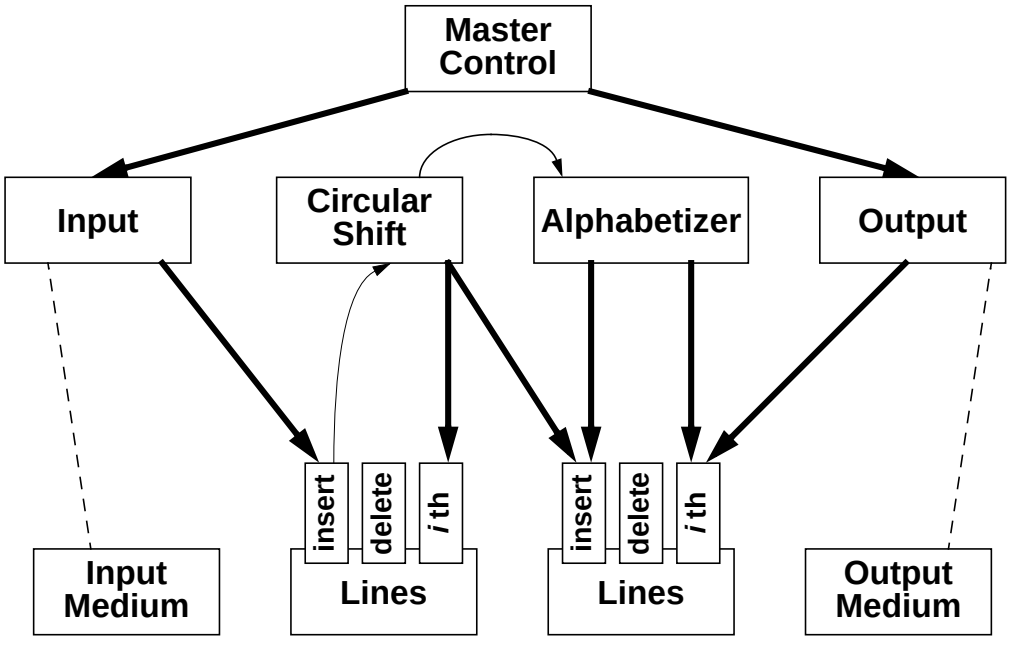
KWIC Architecture: ADT



ADT Pros and Cons

+ Intuitive structure
+ Changeability
  - → data format abstracted away inside ADTs
  - → modification of the processing algorithm isolated to individual modules
+ Support for reuse
  - → fewer assumptions about the rest of the system
- Expansion of functionality
  - → sacrifice either conceptual simplicity or performance

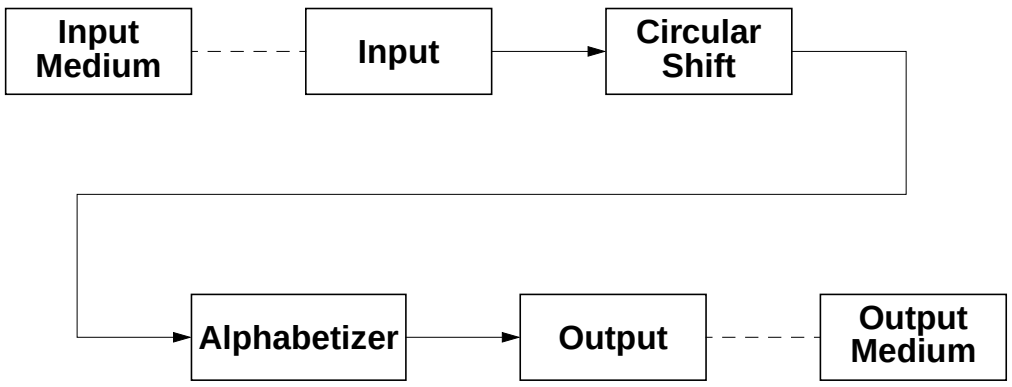
KWIC Architecture: Implicit Invocation



Implicit Invocation Pros and Cons

+ Intuitive structure
+ Data format abstracted away and “activated”
+ Changeability
  - → functional enhancements easy
  - → computation separate from data representation
+ Support for reuse
  - → modules rely on events, not other modules
- Processing order
- Efficiency
  - → data-driven solution leads to a bigger footprint

KWIC Architecture: Pipe and Filter



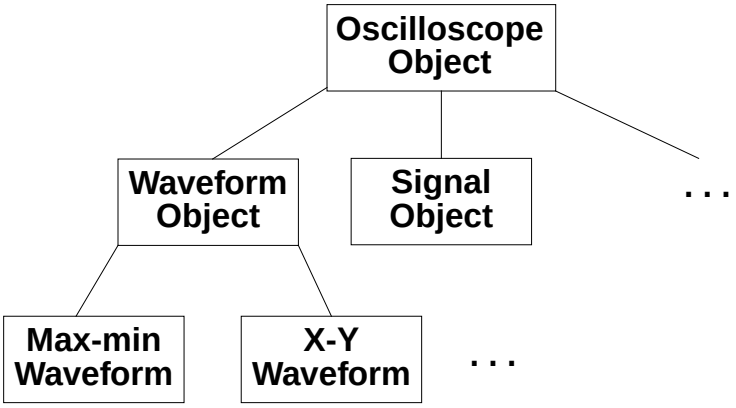
Pipe&Filter Pros and Cons

+ Intuitive structure and processing flow
+ Support for reuse
  - → filters operate in isolation
  - → expect only data of particular format
+ Changeability
  - → easy addition of new filters
- Impossible to evolve into an interactive system
- Efficiency
  - → each filter copies all data to its output ports

Instrumentation Software

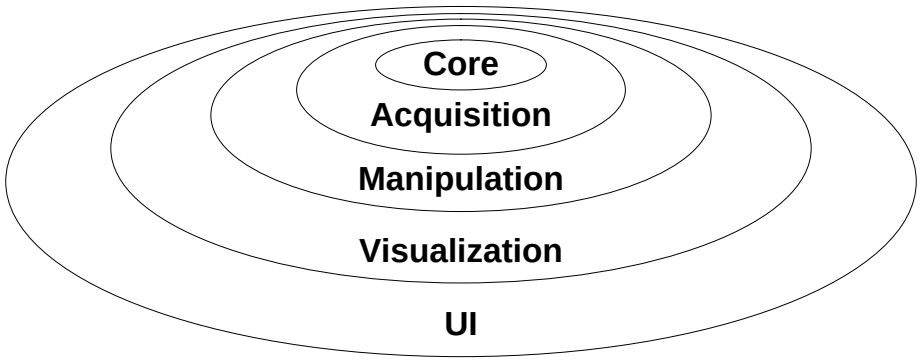
- Oscilloscopes are instrumentation devices that
  - → transform electrical signals into visual images
  - → perform measurements on signals
  - → support multiple user displays
- Oscilloscopes are complex
  - perform many measurements
  - require a lot of storage
  - interface with other instruments and computer networks
  - provide a sophisticated UI
- Goal: develop a reusable system architecture for oscilloscopes
  - develop a common “core” architecture
  - grow a product line around that core
  - allow expansion into other domains

Oscilloscope Architecture — OO



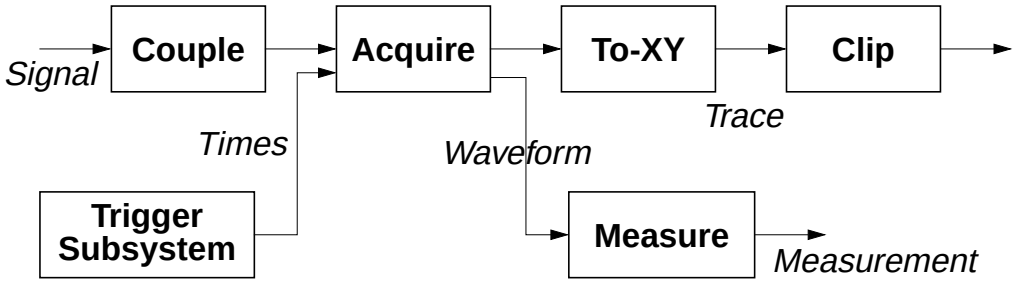
- No overall model of how the data types fit together
- Problem of partitioning the functionality

Oscilloscope Architecture — Layered



+ Intuitively appealing
- Wrong for the application domain
  - actual oscilloscope functions cross layers

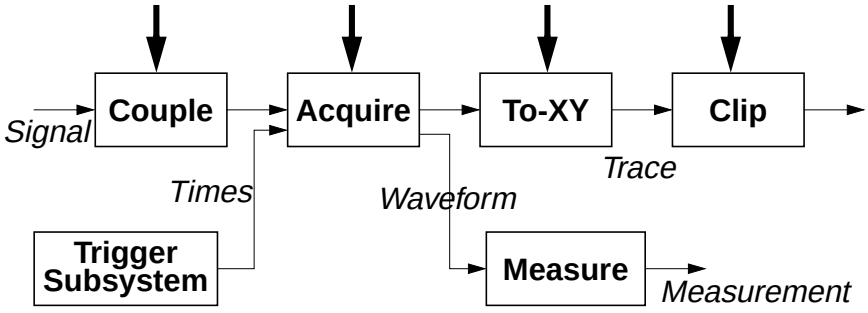
Oscilloscope Architecture — Pipe&Filter



+ Functions not isolated into separate partitions
+ Data flow nature of signal processing is reflected
+ Allows combination and substitution of software and hardware components
- Does not enable the user to interact with the system

Oscilloscope Architecture — Modified Pipe&Filter

- Solution: add control interfaces to filters



+ Explicates modifiable parts of a filter
+ Decouples signal processing functions from the UI
- Poor performance
  - each filter copies data
  - slow filters present bottlenecks
  - → alleviated by flexible pipes (connectors)

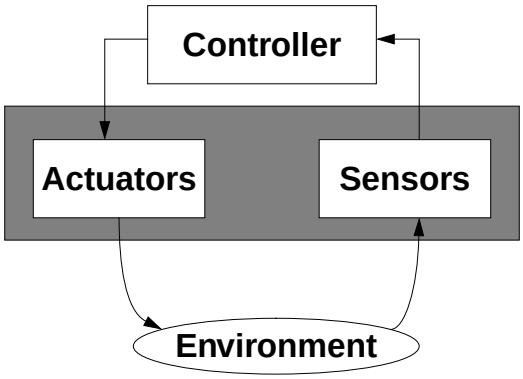
Mobile Robotics

- Manned or partially manned vehicles
- Uses
  - space exploration
  - hazardous waste disposal
  - underwater exploration
- Issues
  - → interface with external sensors and actuators
  - → real-time response to stimuli
  - → response to obstacles
  - → sensor input fidelity
  - → power failures
  - → mechanical limitations
  - → unpredictable events

Basic Mobile Robot Architectural Requirements

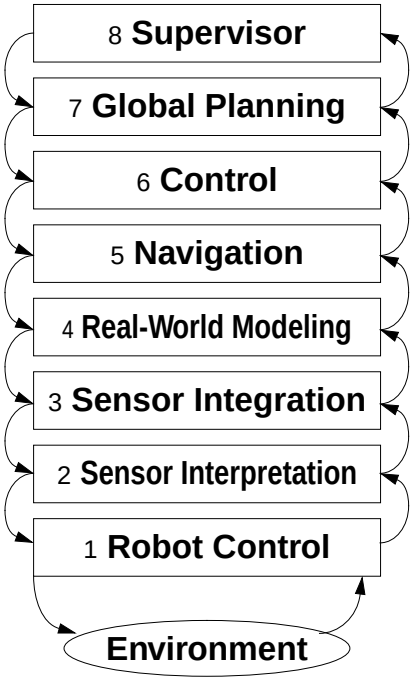
- Accommodate goal accomplishment in the face of **obstacles**
- Allow for **uncertainty** resulting from incomplete or unreliable information
- Handle **dangers** introduced by the environment
  - fault tolerance
  - safety
  - performance
- Exhibit **flexibility**
  - experimentation
  - reconfiguration
  - regular modification

Mobile Robot Architecture — Control Loop



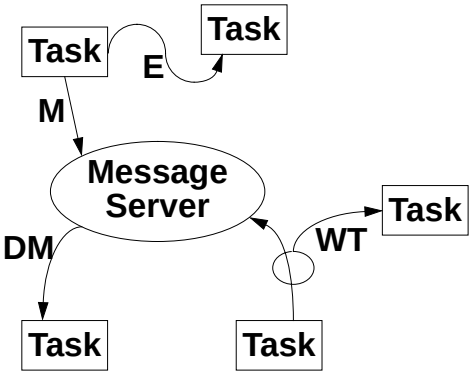
- \- Obstacles
- \- Uncertainty
- \+ Dangers
- \+ Flexibility

Mobile Robot Architecture — Layered



- \- Obstacles
- \+ Uncertainty
- \+ Dangers
- \- Flexibility

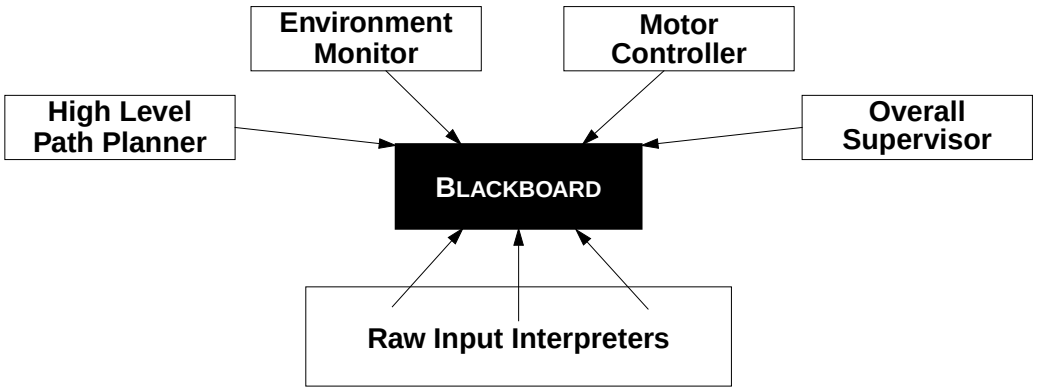
Mobile Robot Architecture — Implicit Invocation



+ Obstacles
- Uncertainty
+ Dangers
+ Flexibility

- Task trees — hierarchies of tasks
  - tasks temporally interdependent
  - → allows specification of selective concurrency
- Tasks communicate by multicasting messages
  - a server directs messages to registered tasks

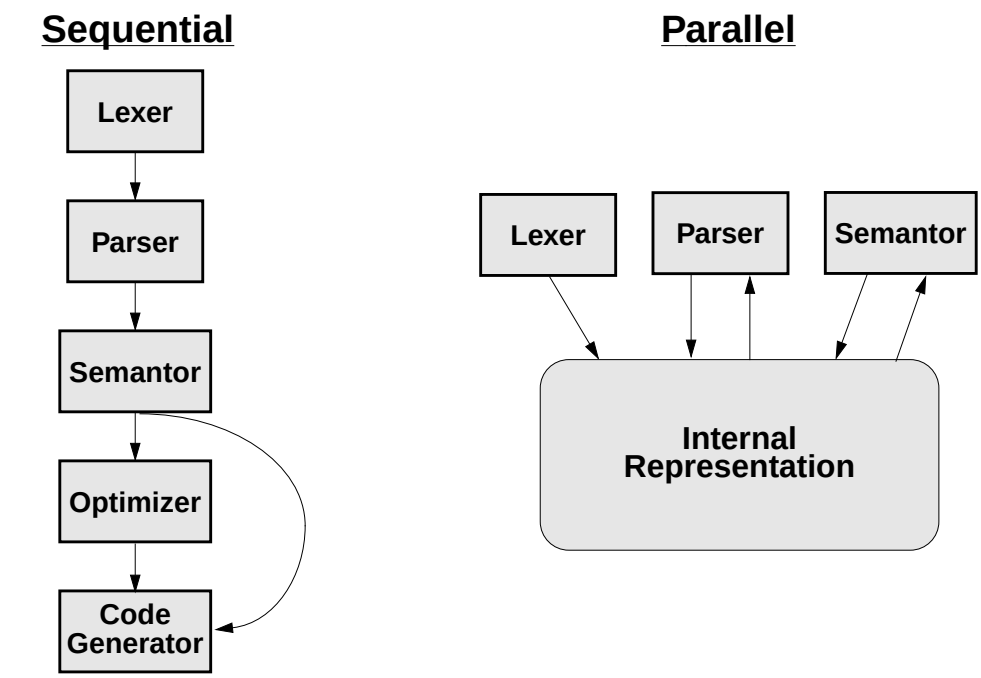
Mobile Robot Architecture — Blackboard



- Obstacles
- Uncertainty
- Dangers

+ Flexibility

Compiler Architecture Revisited



Compiler Architecture Pros and Cons

Sequential

+ Conceptual simplicity
+ Architecture reflects control flow
- Performance

Parallel

+ Performance
+ Adaptability
- Synchronization
- Coordination
  → analysis and testing