

# Module5-Case Study

Tying It All Together



# Architectural Comparisons

- Key Word In Context
- Oscilloscopes
- Mobile Robotics



# KWIC

- Key Word In Context
- Originally proposed in 1972 by Parnas
- Accepts an ordered set of lines
  - Each line is an ordered set of words
    - Each word is an ordered set of characters
- Circularly Shift by repeatedly removing first word and appending to end of line

# Example (Circular Shift)

- THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
- ... QUICK BROWN FOX JUMPS OVER THE LAZY DOG THE
- ... BROWN FOX JUMPS OVER THE LAZY DOG THE QUICK
- ... FOX JUMPS OVER THE LAZY DOG THE QUICK BROWN
- ... JUMPS OVER THE LAZY DOG THE QUICK BROWN FOX
- ... OVER THE LAZY DOG THE QUICK BROWN FOX JUMPS
- ... THE LAZY DOG THE QUICK BROWN FOX JUMPS OVER
- ... LAZY DOG THE QUICK BROWN FOX JUMPS OVER THE
- ... DOG THE QUICK BROWN FOX JUMPS OVER THE LAZY
- ... THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

# Example (Alphabetized Circular Shift)

- THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
- ... BROWN FOX JUMPS OVER THE LAZY DOG THE QUICK
- ... DOG THE QUICK BROWN FOX JUMPS OVER THE LAZY
- ... FOX JUMPS OVER THE LAZY DOG THE QUICK BROWN
- ... JUMPS OVER THE LAZY DOG THE QUICK BROWN FOX
- ... LAZY DOG THE QUICK BROWN FOX JUMPS OVER THE
- ... OVER THE LAZY DOG THE QUICK BROWN FOX JUMPS
- ... QUICK BROWN FOX JUMPS OVER THE LAZY DOG THE
- ... THE LAZY DOG THE QUICK BROWN FOX JUMPS OVER
- ... THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG



# Evaluation according to 5 Criteria

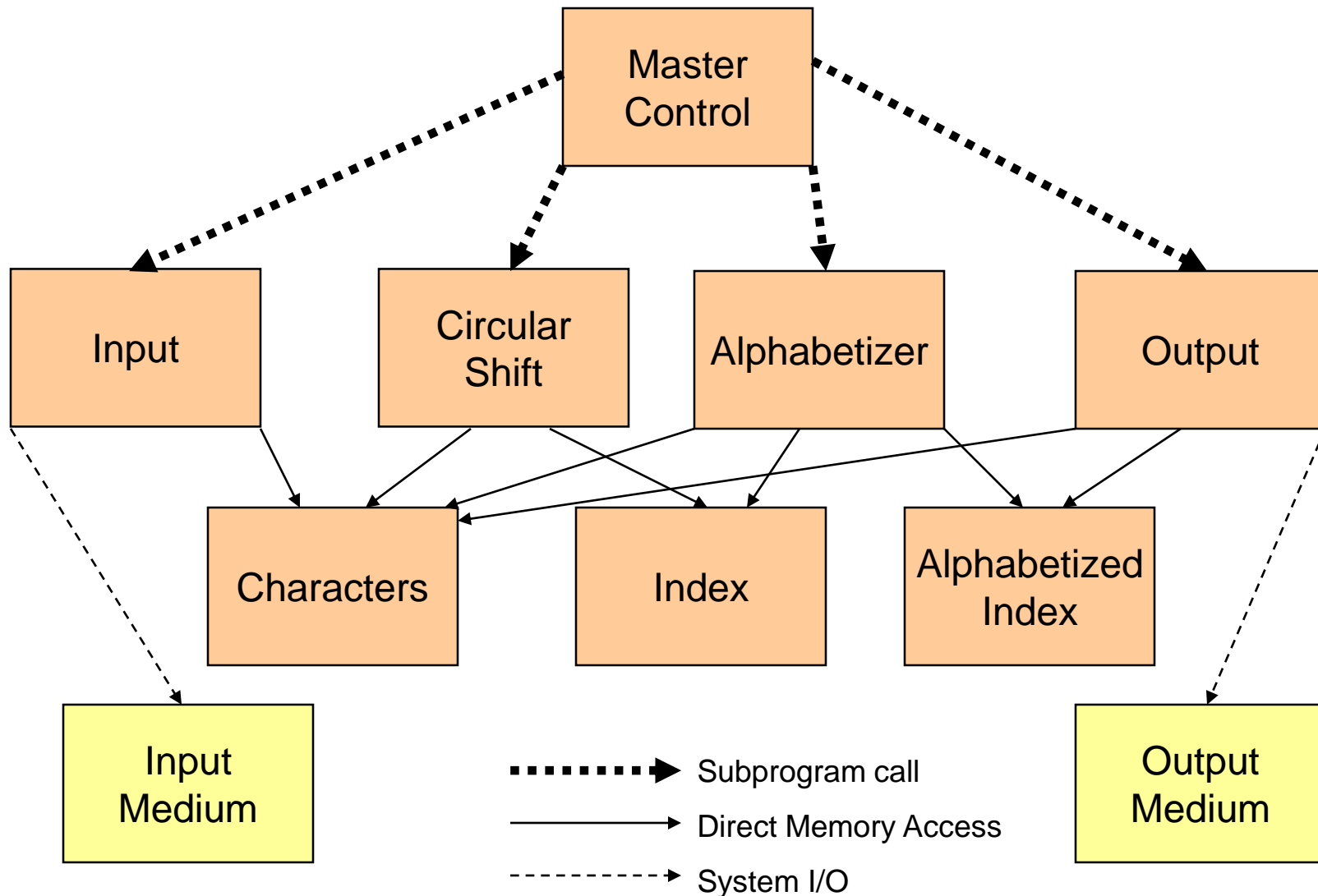
- Ease of changing processing algorithm
- Ease of changing data representation
- Ease of making enhancements to system function
- Performance
- Extent of reuse

# In Class Hands On Activity

- Teams of 4 to 5
- Utilizing the architecture styles presented last class, as a team create 1 possible architecture for the KWIC implementation
- Be prepared to share your drawing



# KWIC-Shared Data Solution



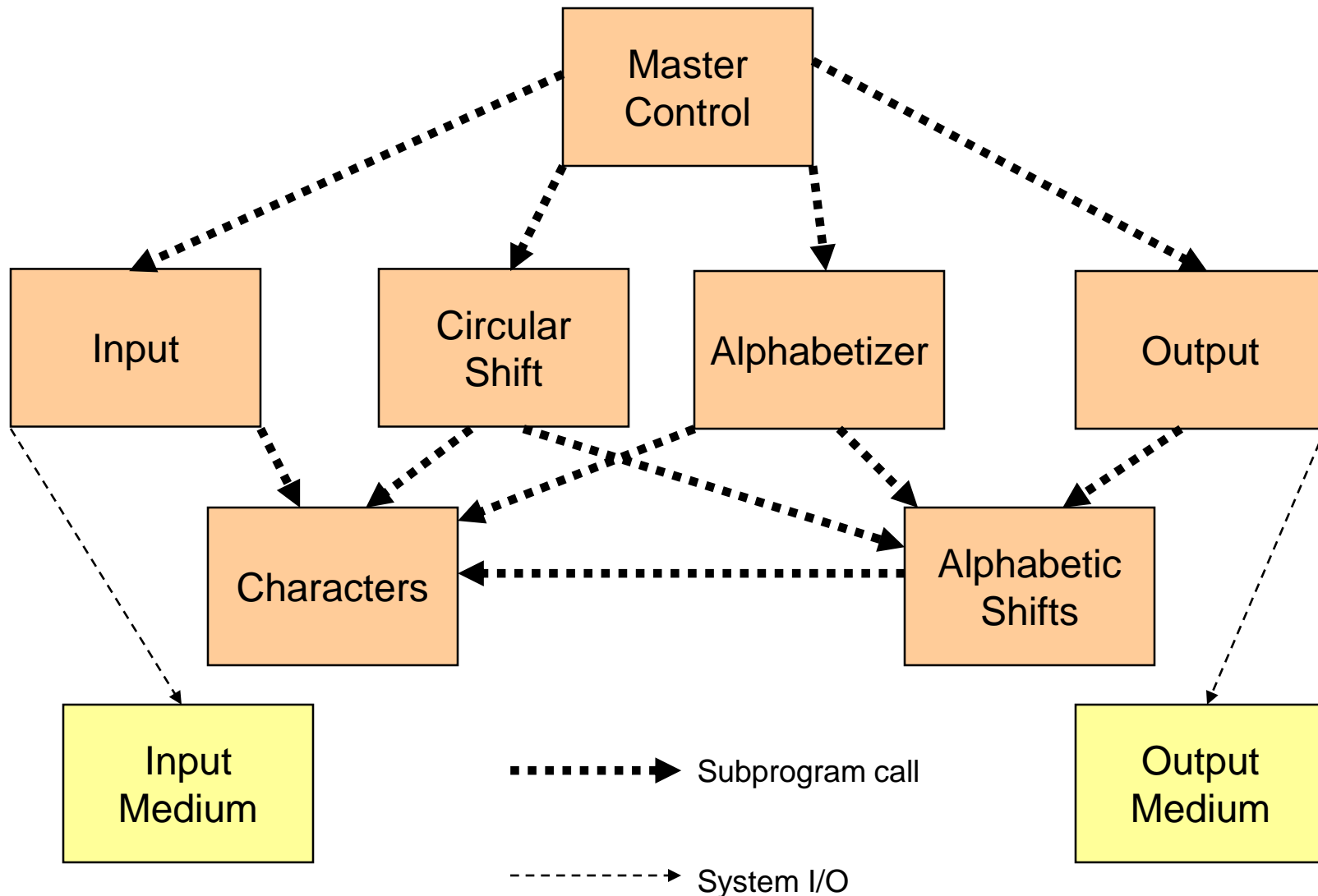




# KWIC-Shared Data Solution

- Master control ensures sequential access
- Efficient data represented
- Intuitive
- Change in data storage format affect almost all modules
- Change in processing algorithm not easily accomodated
- Not supportive of reuse

# KWIC-Abstract Data Solution

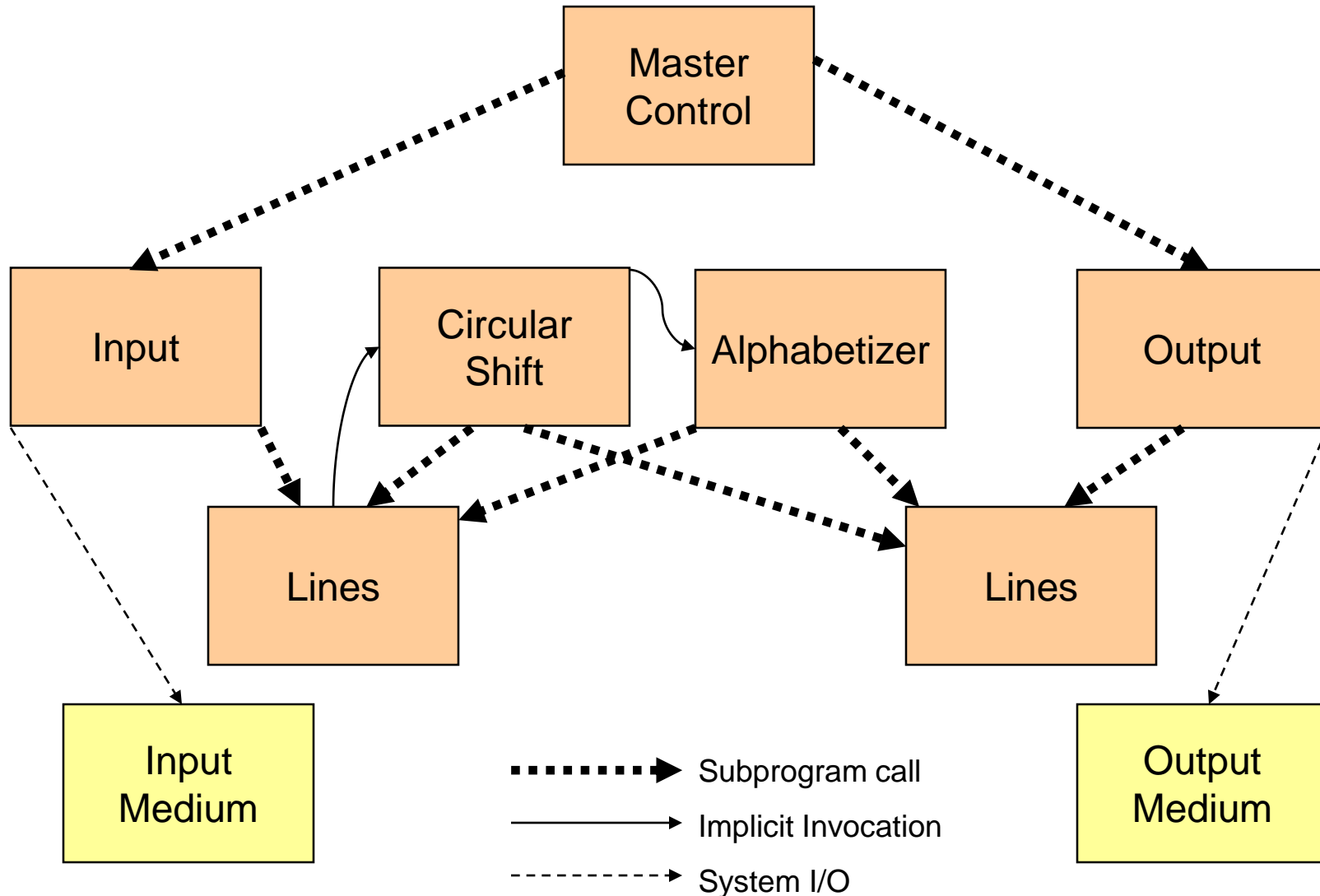




# KWIC-Abstract Data Solution

- Decompose like previous example
- Data no longer directly shared
- Each module provides an interface
- Good logical decomposition
- Algorithms and data representation can be changed in individual modules without affecting others
- Reuse is better supported because modules make fewer assumptions about each other
- Not suited to functional enhancements

# KWIC-Implicit Invocation Solution

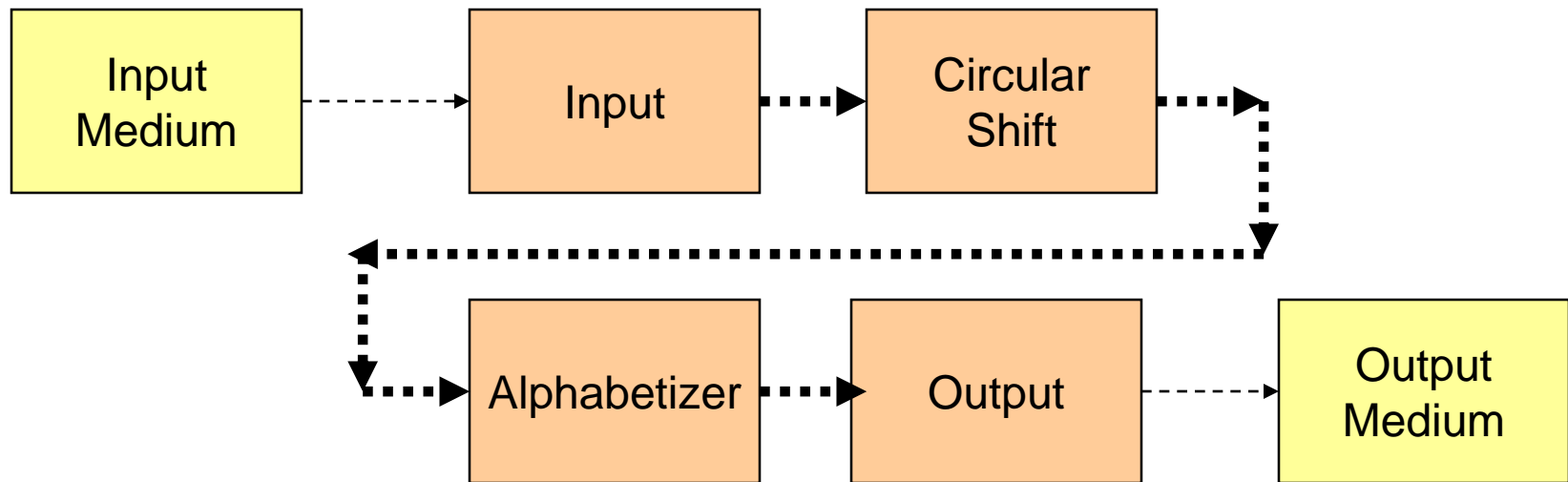




# KWIC-Implicit Invocation Solution

- Shared data similar to first solution but access data abstractly
  - Data accessed as a set or list
- Computations are invoked implicitly as data is modified
- Interaction based on an “active model”
- Supports functional enhancements
  - Register new modules to be invoked on data-changing events
- Insulates computations from changes in data representation
- Supports reuse – modules rely on existence of externally triggered events
- Difficult to control processing order
- Tend to use more space

# KWIC-Pipe-and-Filter Solution



.....➡ Pipe

-----➡ System I/O



# KWIC-Pipe-and-Filter Solution

- Four filters
  - Input
  - Shift
  - Alphabetize
  - Output
- Control distributed
- Intuitive flow
- New functions easily added as new filter
- Hard to modify design to support interactive
  - Delete?

# KWIC Summary Comparison

	Shared Data	Abstract Data Type	Implicit Invocation	Pipe and Filter
Change in Algorithm	-	-	+	+
Change in data representation	-	+	-	-
Change in function	+	-	+	+
Performance	+	+	-	-
Reuse	-	+	-	+

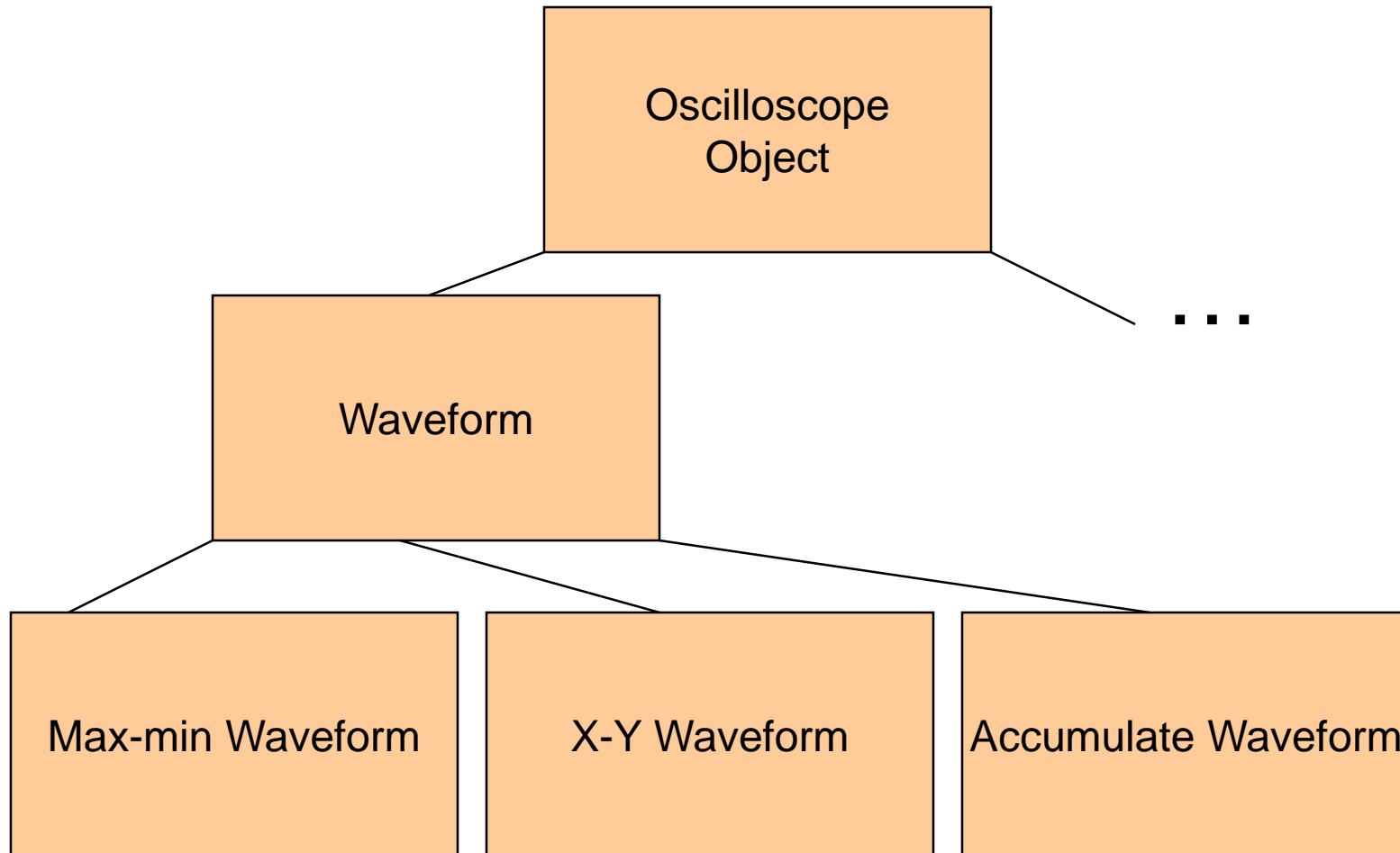


# Oscilloscope Example

- Tektronix
- Collaborative between Tektronix and Computer Research Lab
- Purpose: develop a reusable system architecture for oscilloscopes
  - Sample electrical signals
  - Display pictures (called traces)
- Modern oscilloscopes rely on complex software
- Goals
  - Reuse
  - Performance – rapid configuration for user mode

# Oscilloscope Attempt #1

## Object Oriented Model



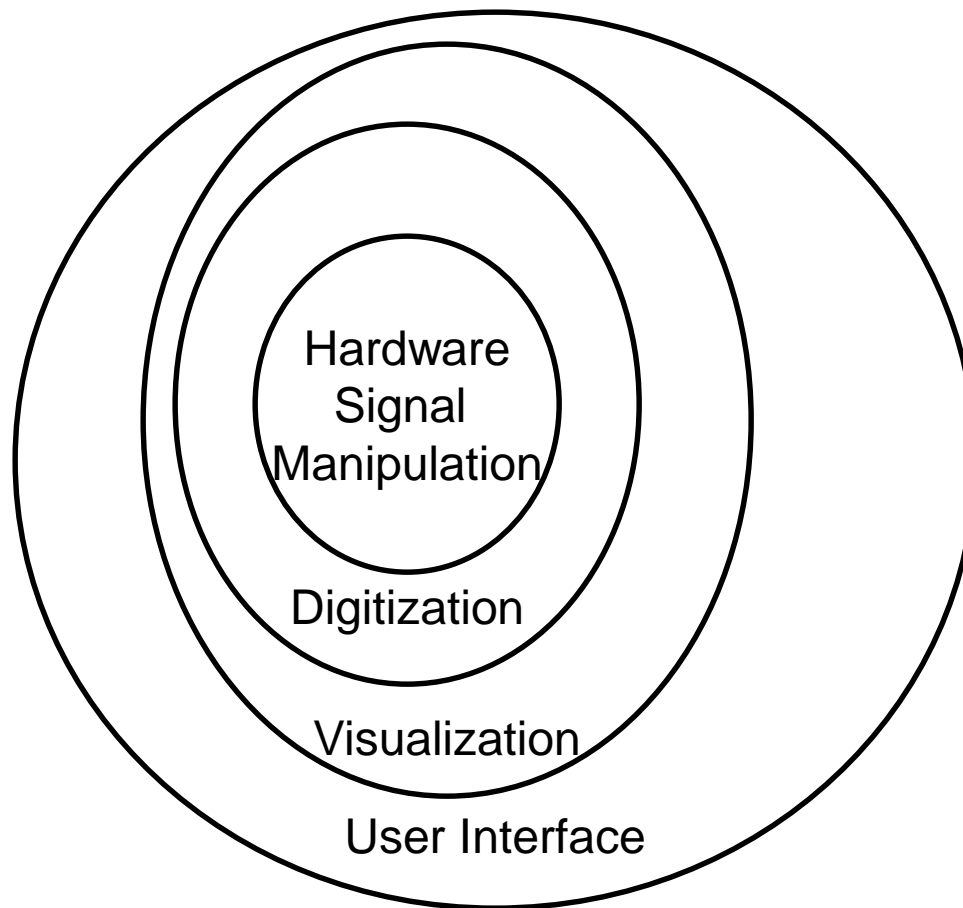
# Oscilloscope Attempt #1

## Object Oriented Model

- Many types of data identified
  - Waveforms
  - Signals
  - Measurements
  - Trigger modes
  - ...
- No overall model explained how the types fit together
- No obvious solution
- Useful exercise but not a useful model

# Oscilloscope Attempt #2

## Layered Model





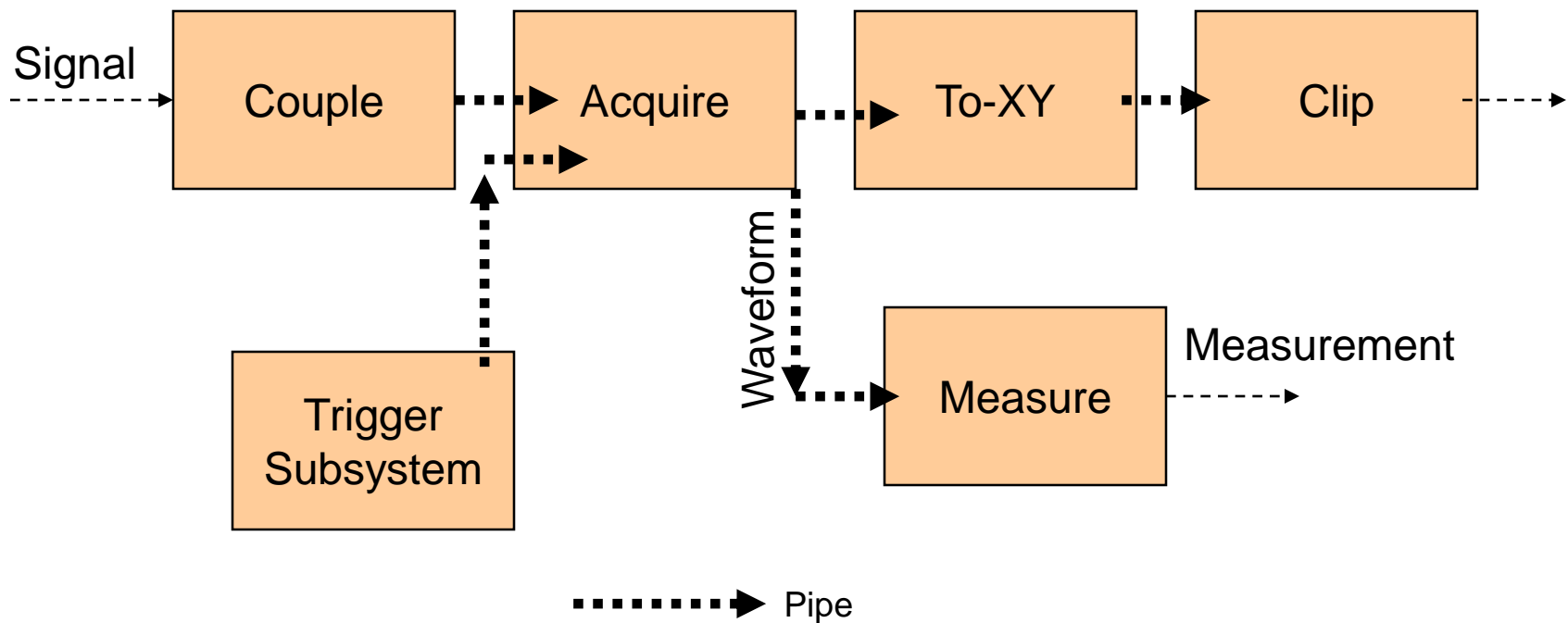
# Oscilloscope Attempt #2

## Layered Model

- Intuitive
- Partitioned functions
- Boundaries of abstraction enforced by layers conflicted with needs for interaction among various functions
  - e.g. Are all user interactions in form of visual?

# Oscilloscope Attempt #3

## Pipe-and-Filter



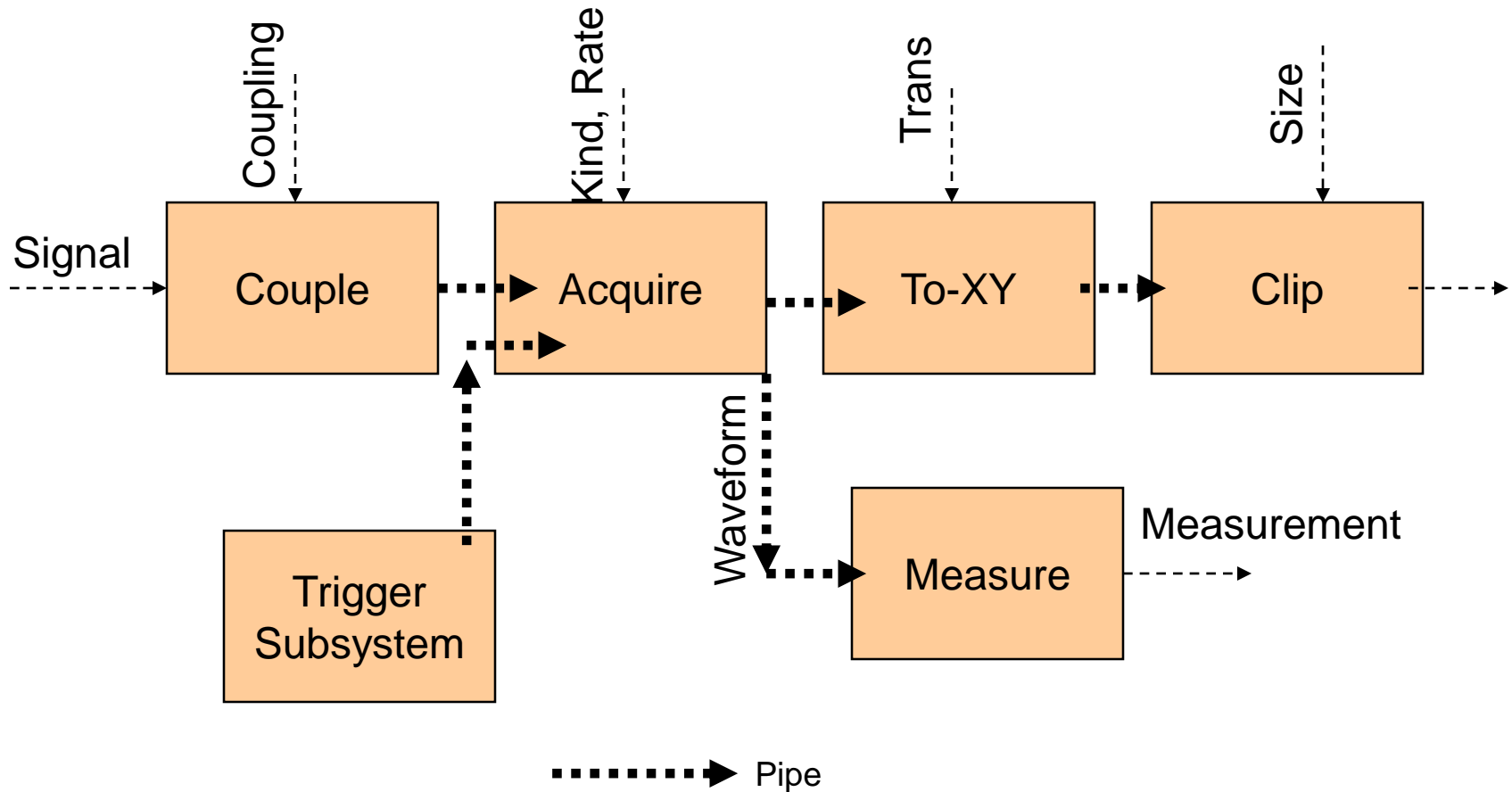
# Oscilloscope Attempt #3

## Pipe-and-Filter

- Incremental Transformers of data
- Signal transformers condition external signals
- Acquisition transformers derive digitized waveforms
- Display transformers convert waveforms into visual data
- No isolation of functions into separate partitions
  - e.g. signal data can feed directly into display filters
- Matched engineer's view of signal processing data flow
- Not clear how the user should interact

# Oscilloscope Attempt #4

## Modified Pipe-and-Filter







# Oscilloscope Attempt #4

## Modified Pipe-and-Filter

- Adds control interface for user interface
- Allows external entity to set parameters of operations for the filter
- Inputs allow for configuration
- Decouples signal-processing functions from the actual user interface
- No assumptions about how communication with user works
- Isolates changes
- Poor performance so improvements needed



# What did these two examples show?

- Different architectural styles have different effects on the solution to a set of problems
- Architectural design adapted from pure forms to specialized styles to meet the needs of the specific domain



# Mobile Robotics

- Controls a manned or partially manned vehicle such as car, submarine, space vehicle
- Software for control is challenging problem
- External sensors and actuators
- Real time response
- Response rate must match environment
- Functions include
  - Acquire input
  - Control motion
  - Plan path
- Complications include obstacles, power, mechanical limitations

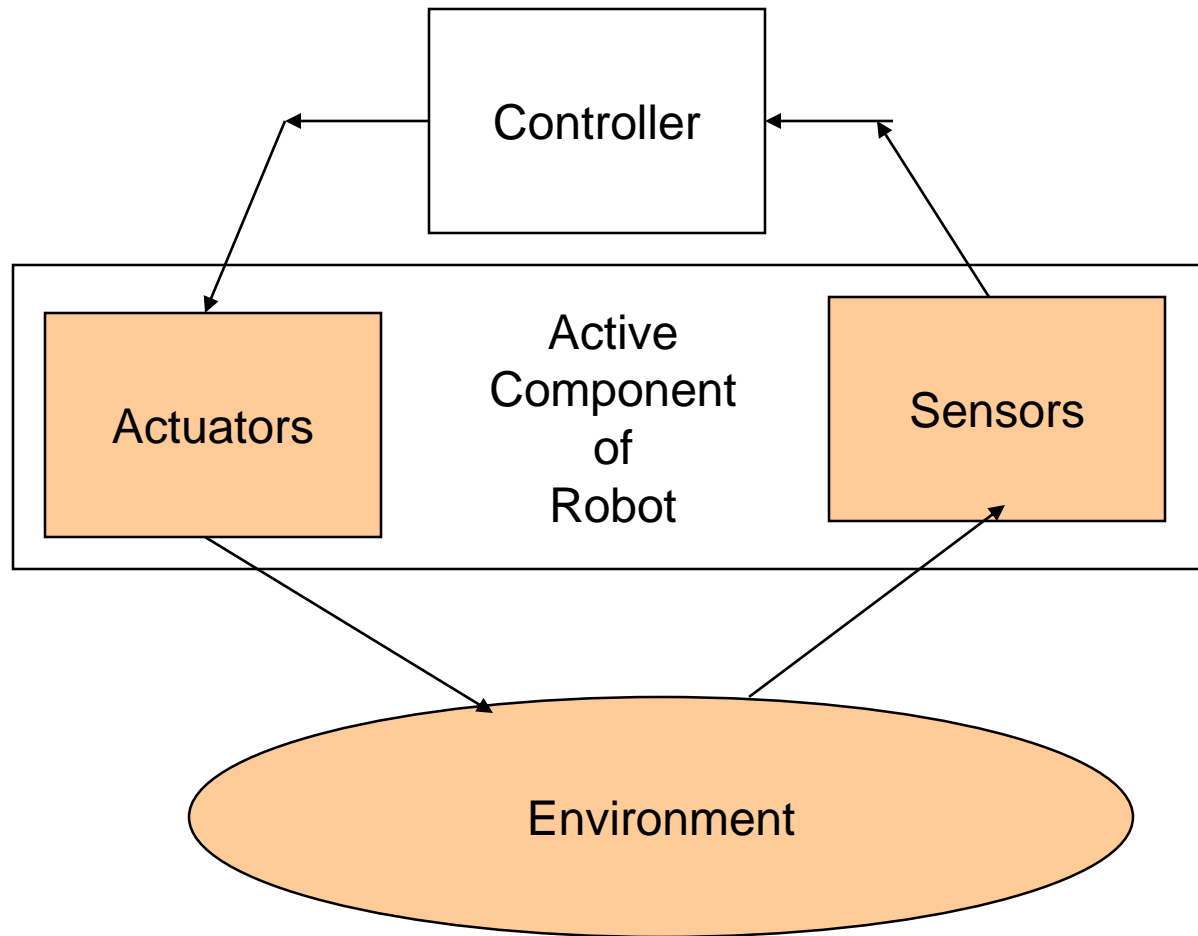
# Mobile Robotics

## Design Considerations

- Req1 – Architecture must accommodate *deliberative and reactive behavior*. Robot must coordinate the actions it undertakes to achieve its designated objective with the reactions forced on it by the environment
- Req2 -- The architecture must allow for *uncertainty*. The circumstances of a robot's operation are never fully predictable. The architecture must provide a framework in which the robot can act even when faced with incomplete or unreliable information
- Req3 – The architecture must *account for dangers* inherent in the robot's operation and its environment. By considering fault tolerance, safety, and performance the architecture must help maintain the integrity of the robot, its operators, and its environment. Problems like reduced power supply, dangerous vapors, or unexpectedly opening doors should not lead to disaster
- Req4 – The architecture must give the designer *flexibility*. Application development for mobile robots frequently requires experimentation and reconfiguration. More over, changes in tasks may require regular modification.

# Mobile Robotics Solution #1

## Control Loop



# How The Requirements are Handled by Solution #1

- Req1 – Closed loop simplicity captures basic interaction between robot and outside. Drawback is unpredictability. No way to leverage ability to decompose into cooperating components.
- Req2 – Biased towards reducing the unknowns through iteration (trial and error with action and reaction)
- Req3 – Simplicity enables easy duplication and reduces error brought forth by complexity
- Req4 – Major components are separated and can be replaced independently. Refinement handled by designers and not architects.

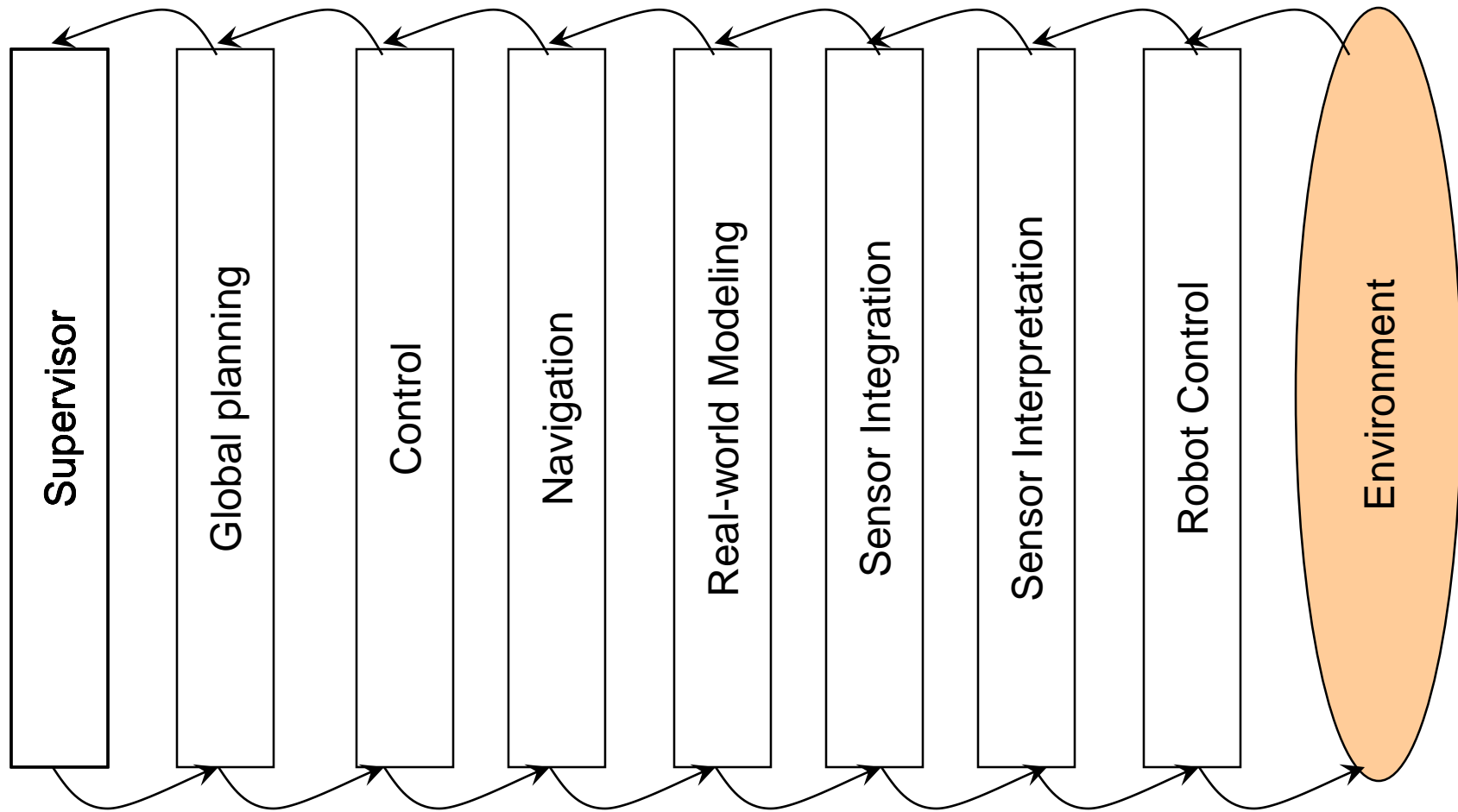


# Solution #1

- Closed-loop
- Appropriate for simple robotic systems that
  - handle only a small number of external events
  - tasks do not require complex decomposition

# Mobile Robotics Solution #2

## Layered






# Layer Services

- Level 1 – Robot Control Routines (gears, motors, joints)
- Level 2/3 – Input from real world (sensor interpretation and sensor integration)
- Level 4 – Maintain robot's model of the world
- Level 5 – Manages navigation of robot
- Level 6/7 – Schedule and plan robot's actions (dealing with problems and replanning)
- Level 8 – User Interface & Supervisory Functions

\* To be usable, strict layer interaction not enforced



# How The Requirements are Handled by Solution #2

- Req1 – More components available for delegation and abstraction to guide design
- Req2 – Existence of abstraction layers addresses the need for managing uncertainty (more knowledge as you move up layers)
- Req3 – Abstraction enables but communication will need to be short circuited to be usable
- Req4 – Complex relationships difficult to decipher

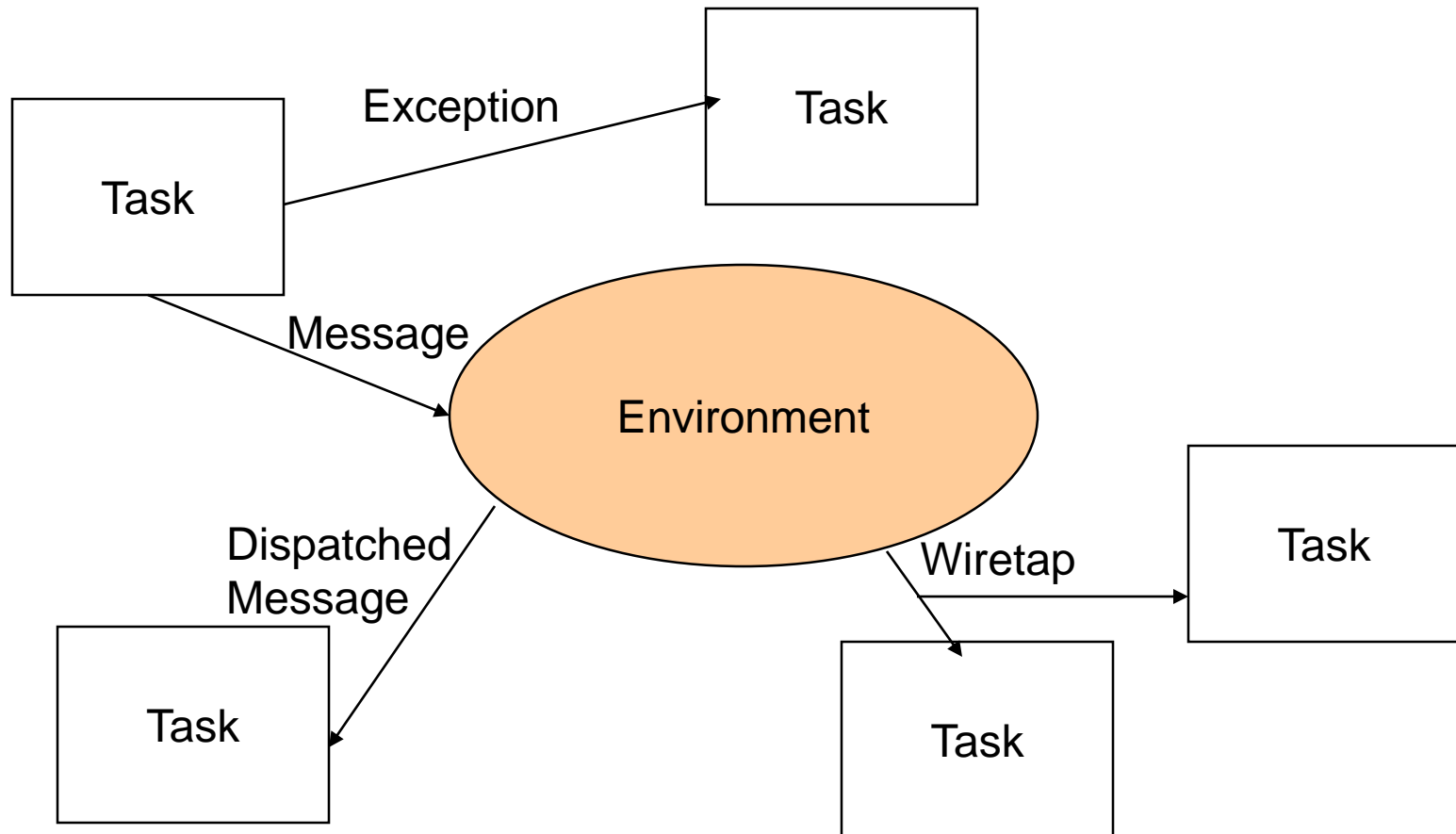


# Solution #2

- Layered
- Abstraction levels provide a framework for organizing components
- Precise roles for each layer
- Major drawback: actual implementation has greater level of detail required and communication patterns in real time do not follow orderly scheme implied

# Mobile Robotics Solution #3

## Implicit Invocation with Task Control Architecture

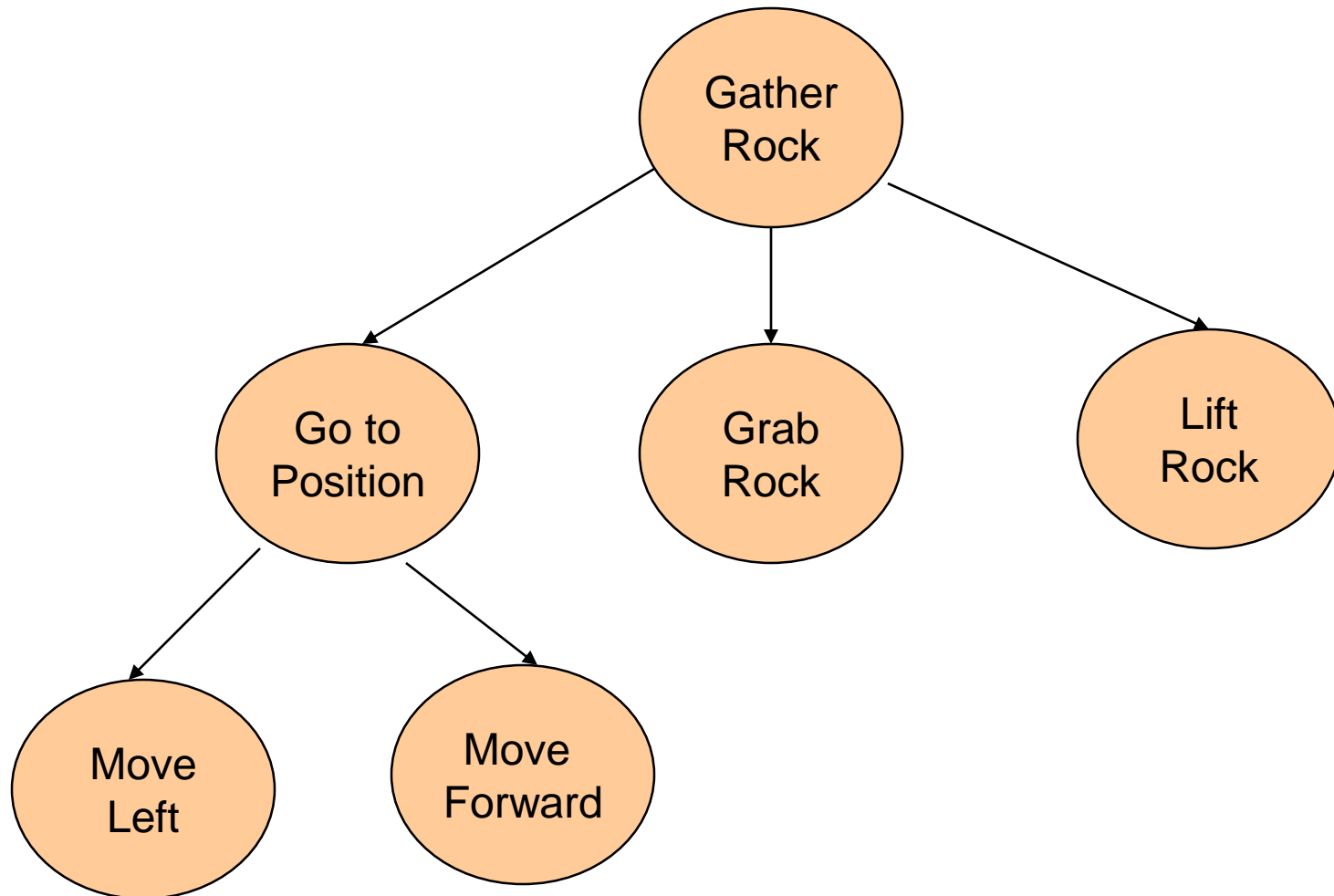





# Task-Control Architecture

- Hierarchies of tasks or task trees
- Software designer can define temporal dependencies between pairs of tasks (e.g. A must complete before B starts) allows selective concurrency
- Exceptions – certain conditions cause the execution of an associated exception handler
- Wiretapping – Messages can be intercepted by task superimposed on an existing task tree
- Monitors – Read information and execute some action if data fulfills certain criteria

# Sample Task Tree





# How The Requirements are Handled by Solution #3

- Req1 – Clear cut separation of action (behavior embodied in task tree) and reaction (behavior dictated by extraneous events and circumstances). Explicitly incorporates concurrent agents
- Req2 – Imponderables would require tentative task tree to be built and modified by exception handlers
- Req3 – Exception, wiretapping, and monitoring features take into account the needs for performance, safety, and fault tolerance
- Req4 – Use of implicit invocation makes incremental development and replacement of components straightforward



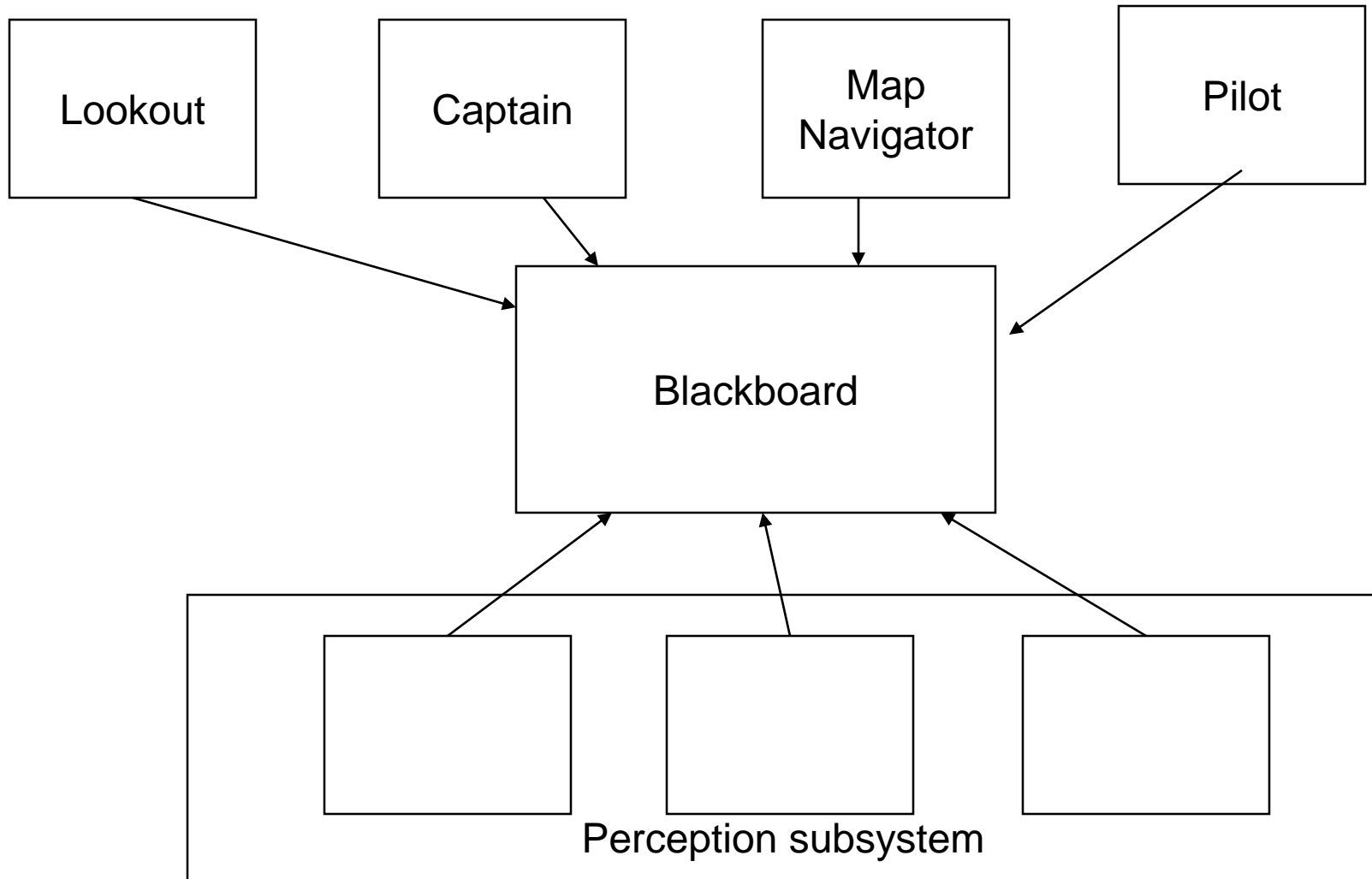
# Solution #3

- Implicit Invocation Task Control
- Comprehensive set of features for coordinating task
- Respect requirements for quality and ease of development
- Richness of scheme makes it appropriate for complex robot projects



# Mobile Robotics Solution #4


## Blackboard





# Blackboard

- Captain – overall supervisor
- Map Navigator – high level path planner
- Lookout – monitors the environment
- Pilot – low level path planner and motor controller
- Perception subsystem: modules that accept raw input from multiple sensors and integrate into a coherent interpretation



# How The Requirements are Handled by Solution #4

- Req1 – Components communication via shared repository blackboard but control flow has to be coerced as components cannot communicate directly
- Req2 – Blackboard resolves conflicts or uncertainty
- Req3 – Can define module to monitor the blackboard for signs of unexpected occurrences (like wiretap)
- Req4 – Supports concurrency and decouples senders from receivers, thus facilitating maintenance



# Solution #4

- Blackboard
- Capable of modeling cooperation of tasks for coordination and resolving uncertainty
- implicit invocation with shared database

# Mobile Robot Summary Comparison

	Control Loop	Layers	Implicit Invocation	Blackboard
Task Coordination	+-	-	++	+
Dealing with Uncertainty	-	+-	+-	+
Fault Tolerance	+-	+-	++	+
Safety	+-	+-	++	+
Performance	+-	+-	++	+
Flexibility	+-	-	+	+

