Robotic Control Architecture

Lecture Outline

- Reactive control
  - Perceptual states, command arbitration
- Subsumption architecture
  - Control system organization / decomposition
  - Designing in Subsumption
    - Layering AFSMs
  - Using the world as a model
  - Example:
    - collecting soda cans

Lecture Outline

- Control system organization / decomposition
- Designing in Subsumption
  - Layering AFSMs
- Using the world as a model
- Example:
  - collecting soda cans

Reactive Control

- Reactive control is based on tight (feedback) loops connecting a robot's sensors with its effectors

- Purely reactive systems do not use any internal representations of the environment, and do not look ahead: they react to the current sensory information

Collections of Reactive Rules

- Reactive systems consist of *collections of reactive rules* that map specific situations to specific actions
- Situations are extracted directly from sensors and actions are made directly with effectors
- They use minimal, if any, internal state
- What kinds of sensor-effector mappings can we have?

Sensor->Effector Mapping

- A reactive system can divide its perceptual world into mutually exclusive (unique) situations
- Then, only one situation can be triggered at a time (by any possible sensory input), and only one action will be activated as a result
- *This is the simplest form of a reactive control system*

Unique Perceptual States

- *It is often too difficult to split up all possible situations this way, or it may require unnecessary encoding*
- To have mutually-exclusive sensory inputs, the controller must encode rules for all possible sensory combinations (inputs from all sensors)
- There is an exponential number of such sensory states

Complete Perceptual Space

- If we enumerate all possible sensory states (also called *input states*) of the system, we get the perceptual space
- A complete mapping is necessary in order to be sure the system can respond to absolutely all possibilities
- This mapping is done while the system is being designed, and it is tedious and and it results in a large look up table

Lookup Speed

- The lookup table takes space to store in a robot.
- It also takes time to search…
- ...unless some parallel look up technique is used
- => *Reactive systems use parallel, concurrently running reactive rules*
- => *They require a parallel control architecture*

Multi-Tasking

- => // *The underlying programming language must have the ability to multi-task, i.e., execute several processes/pieces of code in parallel*
- If a system cannot monitor its sensors in parallel, it has to check them in a sequence/series
- Thus it may miss the onset of some event, or the entire event, and fail to react in time

Incomplete Mappings

- Because of their large size, complete mappings are not used in hand-designed reactive systems
- Instead, key situations trigger appropriate actions, and *default actions* are used to cover all other cases
- Human designers can reduce the input space to only the situations that matter, and so greatly simplify the control system

Arbitration

- If the rules are not triggered by unique mutually-exclusive conditions, more than one rule can be triggered at the same time
- This results in two or more different actions being output by the system
- *Deciding among multiple actions or behaviors is called arbitration*
- Arbitration is in general a difficult problem

Encoding of Arbitration

- Arbitration can be done based on:
  - a fixed priority hierarchy
    - rules have pre-assigned priorities
  - a dynamic hierarchy
    - rules priorities change at run-time
  - fusion/voting
    - rules are averaged
  - learning
    - rule priorities may be initialized and are learned at run-time, once or continuously

Subsumption Architecture:
System Decomposition

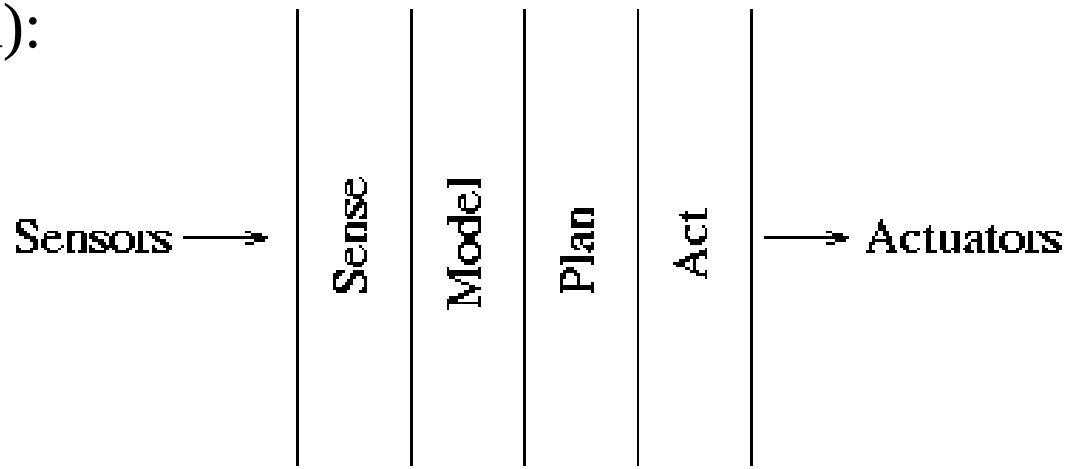
- The traditional approach to control was:
  - the Sense-Plan-Act (SPA) approach
  - inherently sequential (horizontal)
- This produces deliberative architectures
- Subsumption Architecture was introduced as an alternative to SPA
- Subsumption produces inherently parallel (vertical) systems

Vertical v. Horizontal Systems

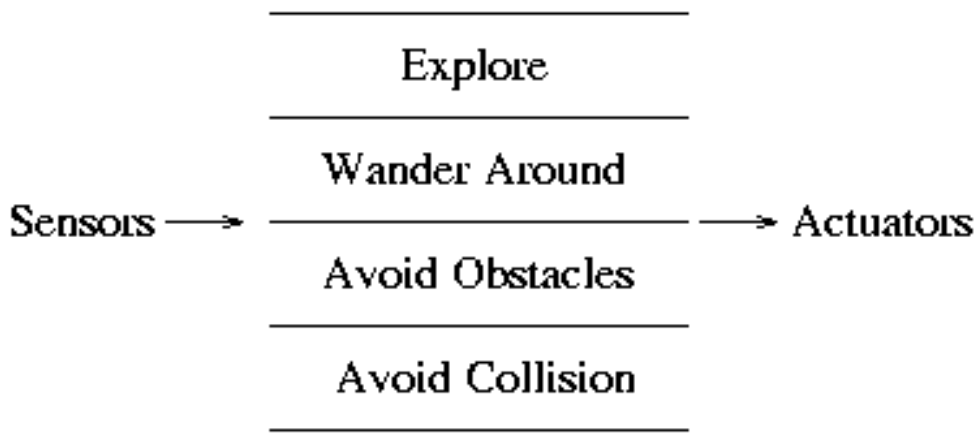
- Vertical
  - All processing modules have access to sensor data
  - Processing modules have access to actuators
- Horizontal
  - Only the first layer of processing modules have access to sensor data
  - Only the last layer of processing modules have access to actuators

Vertical v. Horizontal Systems

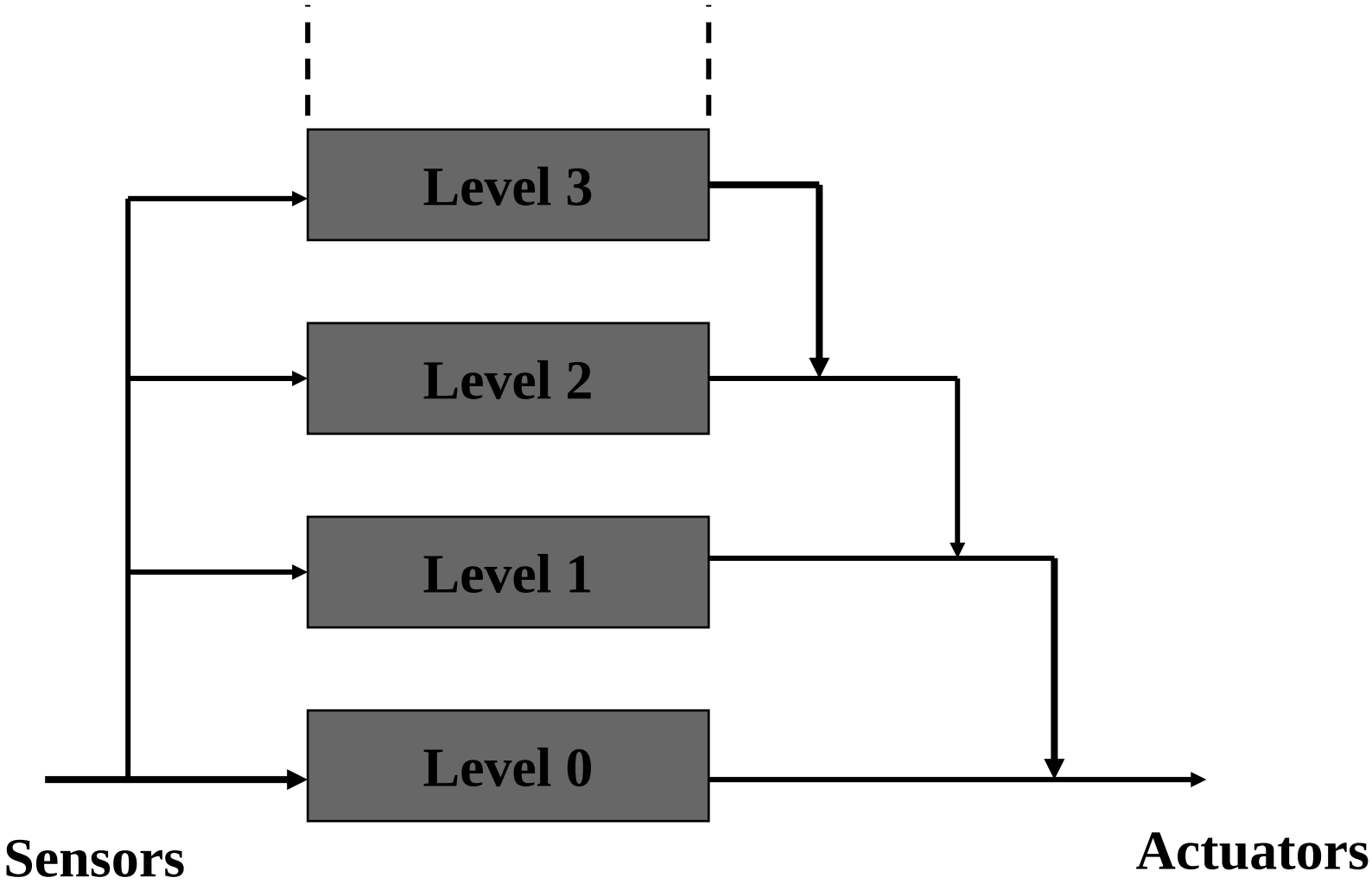
Traditional (SPA):



Subsumption:



Subsumption Architecture



Subsumption Architecture

- Guiding principles of the architecture:
  - systems are built from the bottom up
  - components are task-achieving actions/behaviors (not functional modules)
  - all rules can be executed in parallel
  - components are organized in layers, from the bottom up
  - lowest layers handle most basic tasks
  - newly added components and layers exploit the existing ones...

Guiding Principles

- More guiding principles:
  - each component provides and does not disrupt a tight coupling between sensing and action
  - there is no need for internal models: "the world is its own best model”
- The Subsumption Architecture is constrained to the above principles and attempts to enforce them in robot controllers

Subsumption Illustration

- Layer 0 may be influenced by layer 1.
- *If layer 1 fails, layer 0 is unaffected.*
- Layer 1 can:
  - inhibit the outputs of layer 0 or
  - suppress the inputs of layer 0
- Those are the only types of interactions between layers/modules
- This process is continued all the way up

Growing Systems

- Subsumption systems grow from the bottom up, and layers can keep being added, depending on the tasks of the robot.
- *How exactly layers are split up depends on the specifics of the robot, the environment, and the task.*
- There is no strict recipe, but some solutions are better than others, and most are derived empirically

Pros and Cons

- Some critics consider the lack of detail about designing layers to be a weakness of the approach.
- Others feel it is a strength, allowing for innovation and creativity.
- Subsumption has been used on a vast variety of effective implemented robotic systems.
- *It was the first architecture to demonstrate many working robots.*

Biological Inspiration

- The inspiration behind the Subsumption Architecture is the evolutionary process, which introduces new competencies based on the existing ones.
- Complete creatures are not thrown out and new ones created from scratch; instead, solid, useful substrates are used to build up to more complex capabilities.

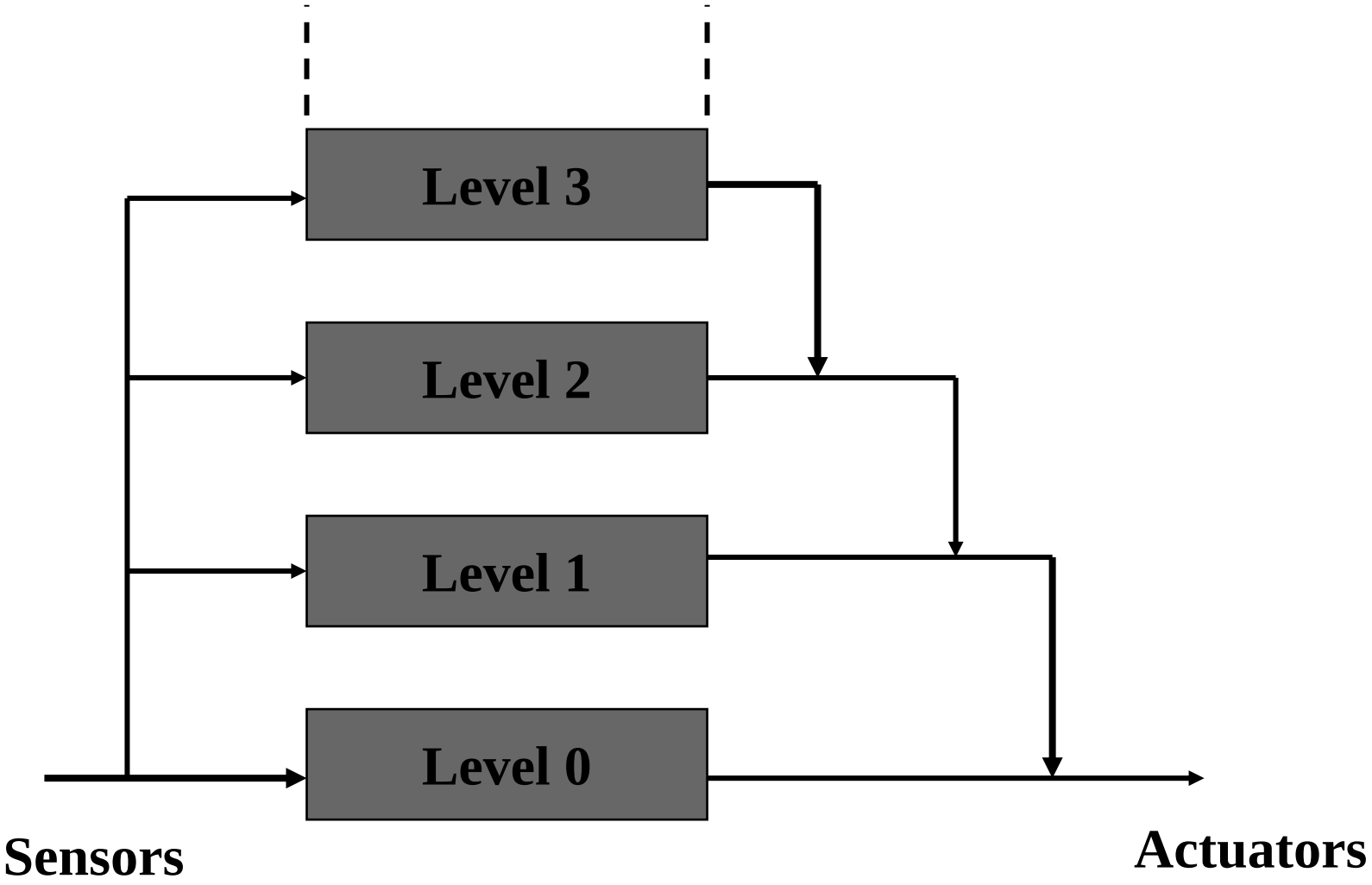
Practical Implications

- The approach results in modular, incremental development of the system, and thus a more *robust design*
- The approach also produces *reusable modules and code*; rules and layers can be reused on different robots and for different tasks

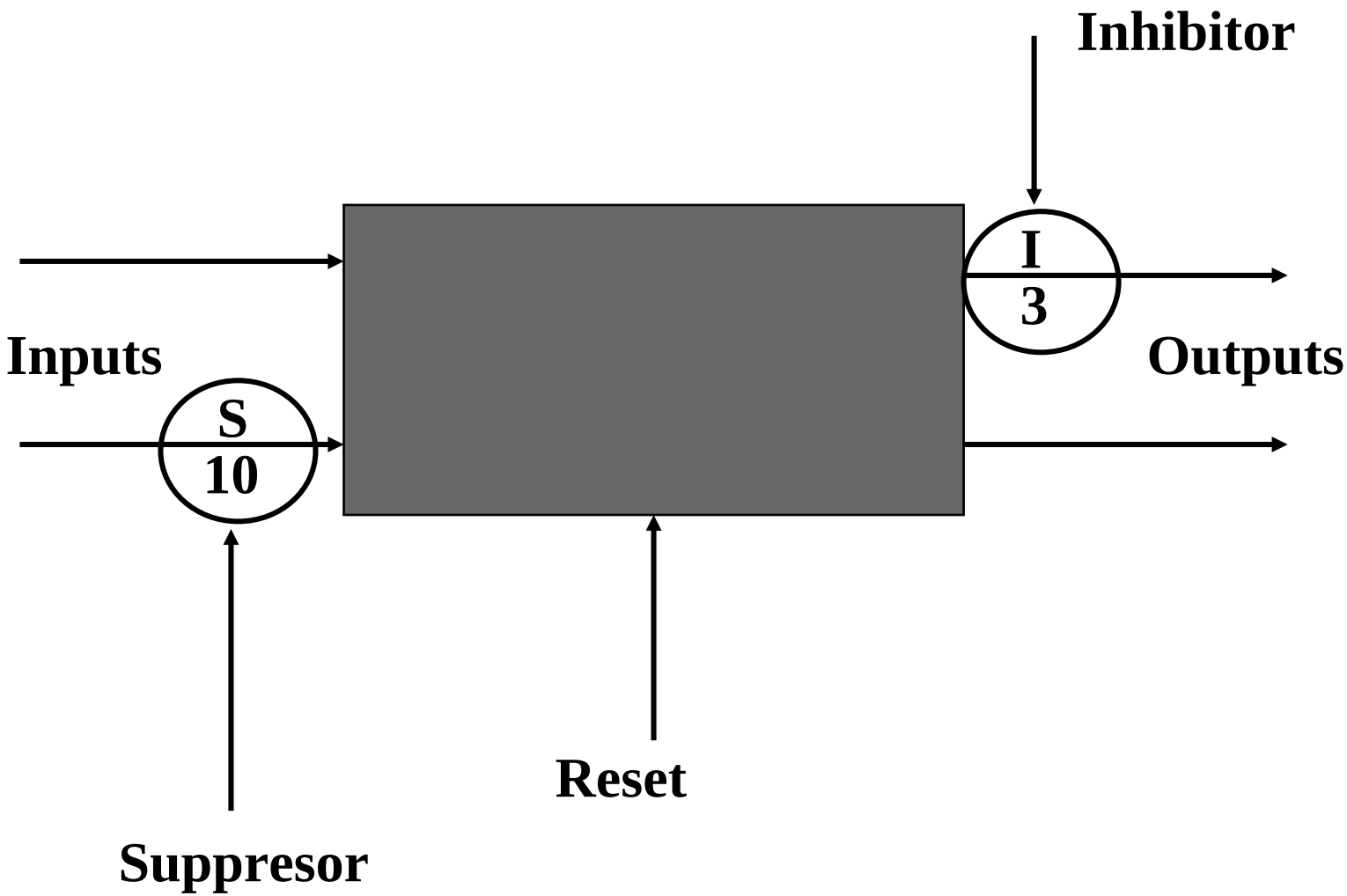
Subsumption Language

- The original Subsumption Architecture was implemented using a language called the Subsumption Language
- It was based on finite state machines (FSMs) augmented with a very small amount of state (AFSMs)
- AFSMs were implemented in Lisp

Subsumption Control



Structure of Modules (AFSMs)



Augmented Finite State Machines

- The structure of FSMs is convenient for incremental system design.
- An AFSM can be in one state at a time, can receive one or more inputs, and send one or more outputs.
- AFSMs are connected by communication wires, which pass input and output messages between them.

Networks of AFSMs

- => *A Subsumption Architecture controller, using the AFSM-based programming language, is a network of AFSMs.*
- The network is divided into layers
- Once a basic competence is achieved (e.g., moving around safely), it is labeled layer 0.
- The next layer (1) is added, and communication is done through wires

Layering in AFSM Networks

- Layer 1 (e.g., one that looks for objects and collects them), can then be added on top
- The use of layers is meant to *modularize the reactive system*, so it is bad design to put a lot of behaviors within a single layer.
- Also, it is bad design to put a large number of connections between the layers, so that they are *strongly coupled.*

Module Independence

- *Strong coupling implies dependence between modules, which violates the modularity of the system.*
- If modules are interdependent, they are not as robust to failure.
- In Subsumption, if higher layers fail, the lower ones remain unaffected.
- Thus Subsumption has one-way independence between layers.
- Two-way independence is not practical, why?

Layering in AFSM Networks

- With upward independence, a higher layer can always use a lower one
- => layer 1 certainly can and should be coupled with layer 0. How?
- …by using suppression and inhibition

- *Do we always have to use these wires to communicate between parts of the system?*

Using the World

- Coupling between layers, and even between AFSMs, need not be through the system itself (i.e., not through explicit communication wires)
- ***It could be through the world.***
- How?
- E.g.: one Subsumption robot (Herbert) collected soda cans and took them home this way.

World Is Its Own Best Model

- This is a key principle of reactive systems & Subsumption Architecture:
- **Use the world as its own best model**
- If the world can provide the information directly (through sensing), it is best to get it that way, than to store it internally in a representation (which may be large, slow, expensive, and outdated).
- Can we always do this?

Arbitration in Subsumption

- Arbitration: deciding who has control
- Inhibition: prevents output signals from reaching effectors
- Suppression: replaces input signal with the suppressing message
- The above two are the only mechanisms for coordination
- => Results in priority-based arbitration
  - the rule or layer with higher priority takes over, i.e., has control of the AFSM

Designing in Subsumption

- Qualitatively specify the overall behavior needed for the task
- Decompose that into specific and independent behaviors (layers)
- The layers should be bottom-up and consisting of disjoint actions
- Ground low-level behaviors in the robot's sensors and effectors
- Incrementally build, test, and add

Herbert

- Searched for and grabbed empty soda cans returning them to the trash
- MIT AI lab: late 1980s
- Named after Herb Simon, an AI Pioneer

Herbert

- First it searched for soda cans;
- When it found one it grabbed it, picked it up, weighed it, ...
- if it was heavy (full), put it down, if it was empty, picked it up,
- …tucked its arm in, and headed home.
- It did not keep internal state about what it had done and what it should do next.
  - How?

More on Herbert

- It just sensed!
- When it sensed the can, it reached out. When it had an empty can, it tucked the arm in. When the arm was tucked in, it went home...
- There is no internal wire between the AFSMs that achieve can finding, grabbing, arm tucking, and going home.
- Still, the events are all executed in proper sequence. Why?

Even More on Herbert

- *Because the relevant parts of the control system interact and activate each other through sensing the world.*
- Herbert used vision and a laser striper to find soda cans, and sonars and IRs to wander around safely.
- Herbert was implemented by Jon Connell, using individual processors for each AFSM and real wires for each connection in the architecture

Some Subsumption Robots

- Genghis, Tito, Allen, Herbert, Seymore, Toto, Tom & Jerry:

More Subsumption Robots

- Practically demonstrated on navigation, 6-legged walking, chasing, soda-can collection, etc.

Subsumption Evaluation

- **Strengths:**
- Reactivity (speed, real-time nature)
- Parallelism
- Incremental design => robustness
- Generality
- **Weaknesses:**
- Inflexibility at run-time
- Expertise needed in design