CS65K Robotics

Modelling, Planning and Control

Chapter 6: Control Architecture

Objectives

- What is the robot control architecture?

- Categories of Control Architecture

- Design Schematic

- Block Diagram

- General μP, ASIP, ASIC

- Hardware/Software Co-design

What is Robot Architecture?

What is a Robot Architecture?

- There are many different ways in which a robot control program can be put together. In order to program a robot in a structured and principled fashion, we use an appropriate robot control architecture.

- System developers have typically relied upon robotic architectures to guide the construction of robotic devices and for providing computational services (e.g., communications, processing, etc.) to **subsystems** and **components**.

Robot Architecture

ASIC for Robots

Computer Architecture and Operation System (ROS) for Robots
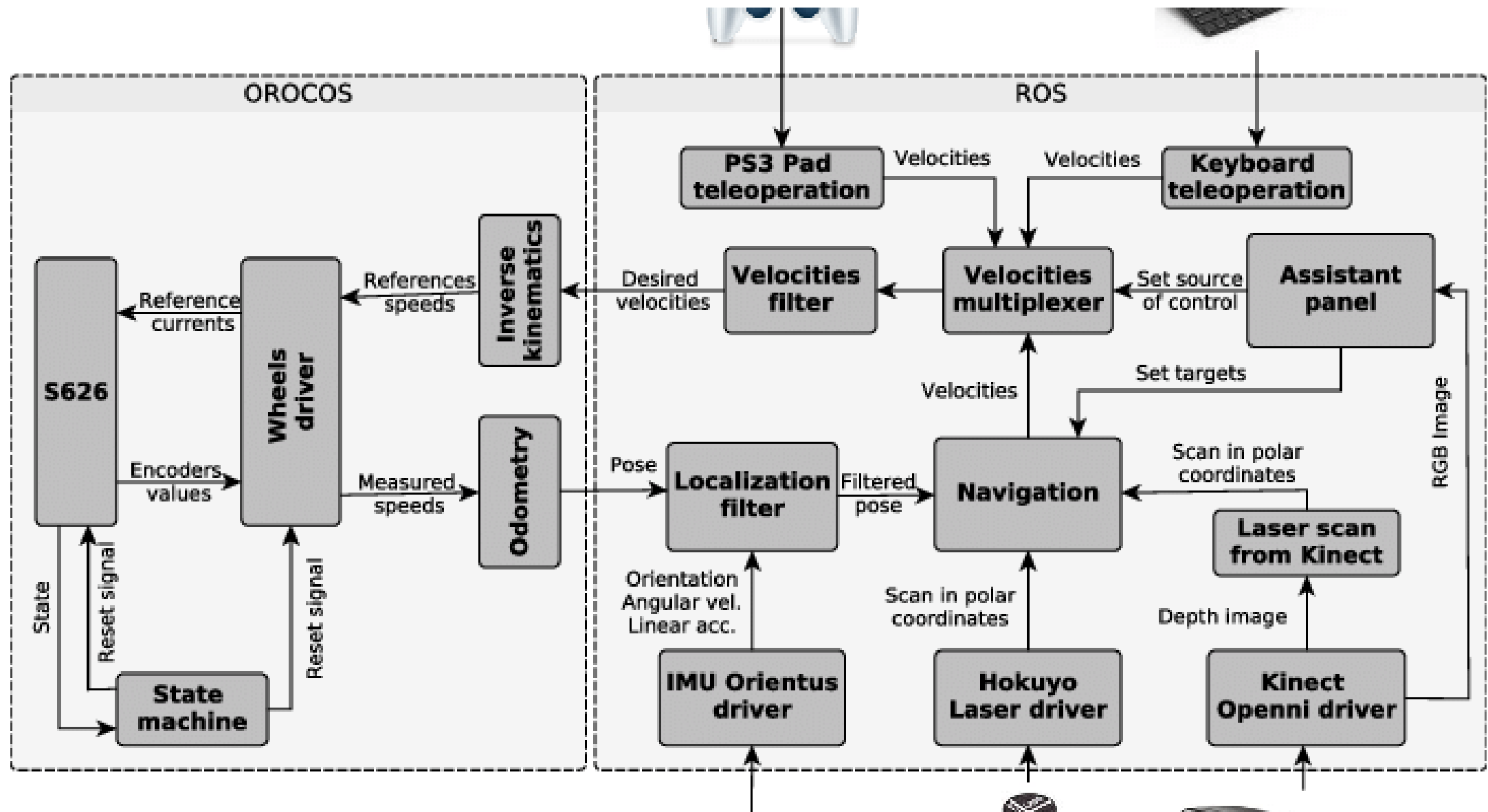
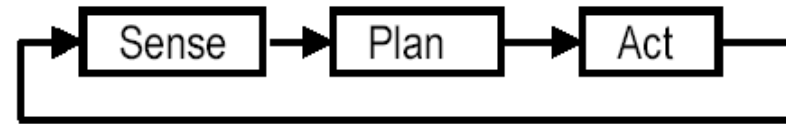Manual Controller

RGBD Camera

Display
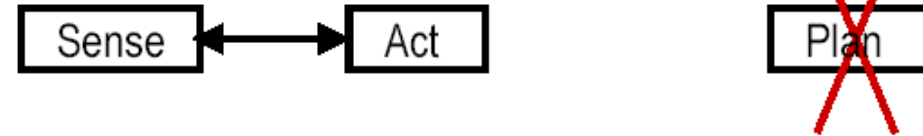
Laser Range Finder

Bumper

Categories of Robot Control

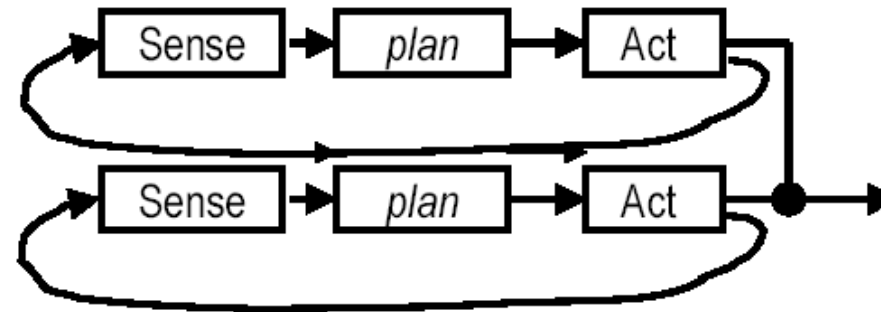1. Hierarchical:

Sense → Plan → Act
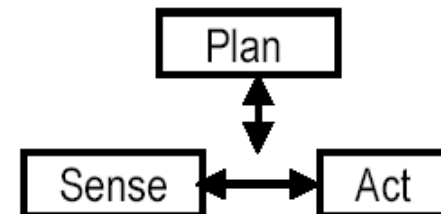
2. Reactive:

Sense ↔ Act   Plan

2+. Behavior-Based:

Sense → plan → Act

Sense → plan → Act

3. Hybrid deliberative/reactive:

Plan

Sense ↔ Act

Categories of Control

- **Deliberative Control** : Think hard, act later.
  - SPA, serial, complete each step first – then proceed
- **Reactive Control** : Don't think, (re)act.

  - Direct connection between perception to action, no memory, no planning.
- **Hybrid Control** : Think and act independently, in parallel.

  - Deliberative and Reactive modules run independently at different time scales
- **Behavior-Based Control** : Think the way you act.
  - Distributed by behavioral task decomposition
  - Each behavior has its restricted planning and execution capabilities

Deliberative Paradigms

- Sense the world
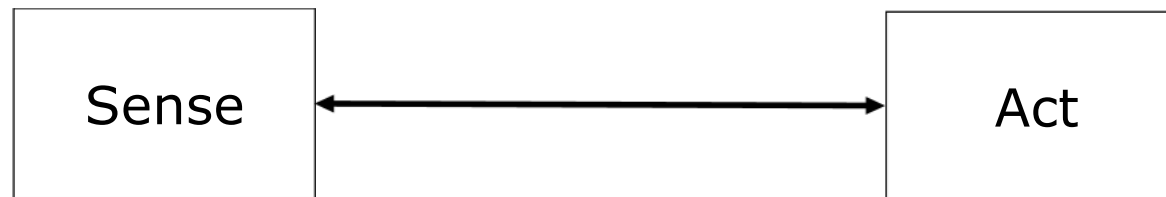- Update world mode and generate plan
- Knowledge representation + Automated reasoning
- Execute planned action
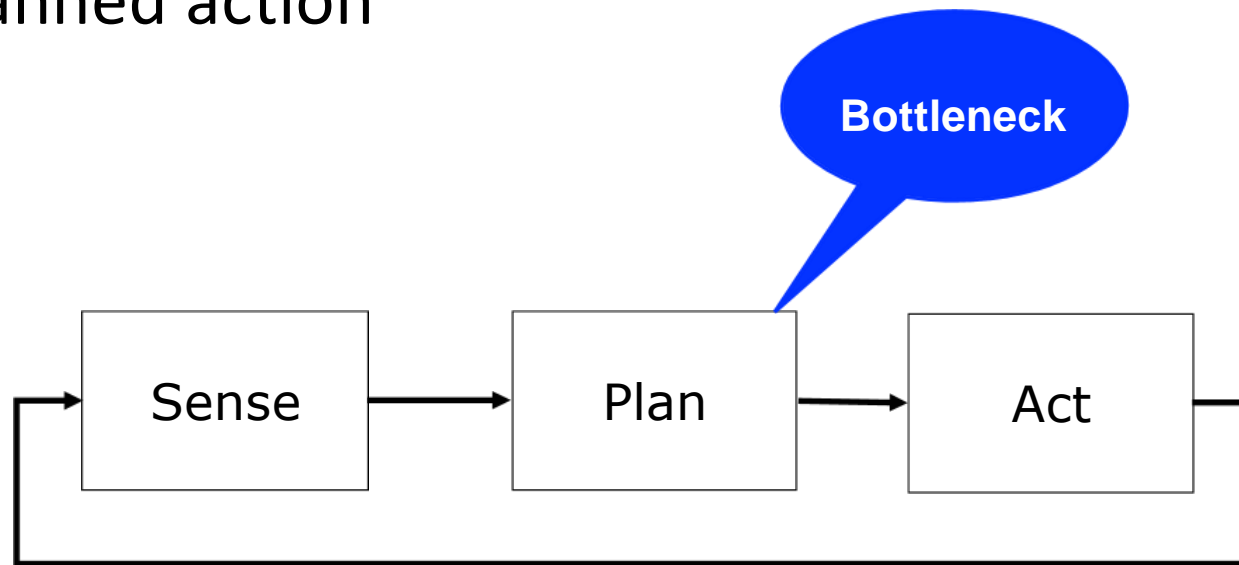
Reactive Paradigm

- No world model; no planning

- Direct mapping between sensor input and actuators output

- Very reactive to changes in sensor readings

- (Radical) answer to shortcomings of deliberative paradigm

| Sense | ←→ | Act |

Deliberative (symbolic)	Reactive (reflexive)
←——————→	—————————————————————→
Speed of response	
—————————————————————————→	
Predictive capabilities, completeness of world models	
←————————————————————	
Needs internal representation	No internal representation
Slow response	Real-time response
High-level intelligence (cognition)	Low-level intelligence
	Simple (analogue) computation

Comparison

Behavior-Based Architectures

- Overall controller composed of two parts
  - Task achieving controllers (e.g., simple state machines)
  - Arbitrating controller (also task specific)

- Controllers:
  - Are reactive (map perception to action)
  - Do not include world models, no planning
  - Are only capable of performing one task each
    - Should perform it very efficiently
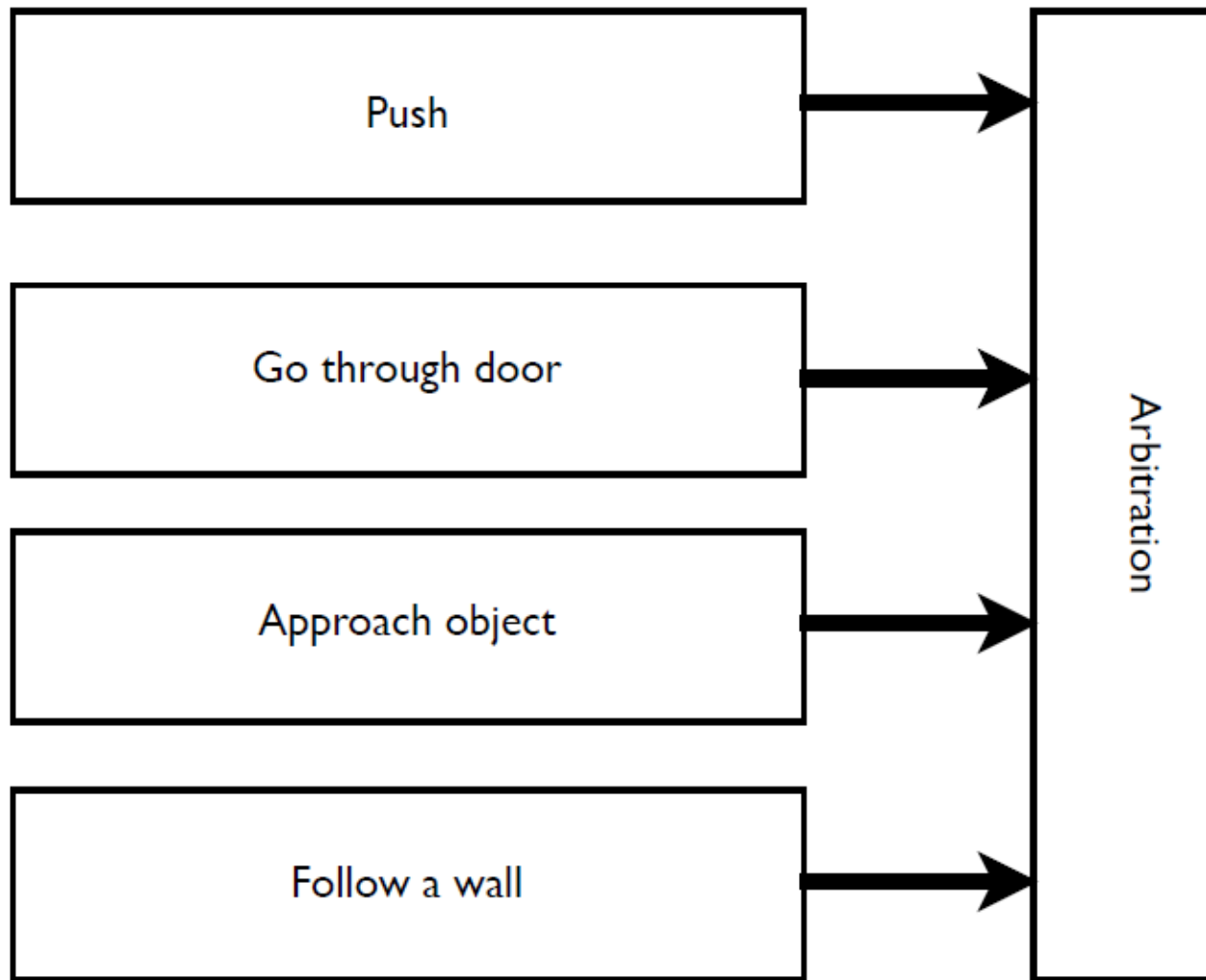
Behavior-Based Architectures

Behavior-Based Architectures

- Behaviors are the basic building block for robotics actions, with an overall emergent behavior obtained from their arbitration

- Behaviors support good software design principles due to modularity

- However, we have some assumptions:
  - We need to be able to decompose the task into appropriate "primitive" behaviors
  - Each "primitive" behavior requires only simple perceptual and physical talents

Behavior-based Architecture

Push	
Go through door	Arbitration
Approach object	
Follow a wall	

Different behaviour-based architectures have different approaches to provide arbitration
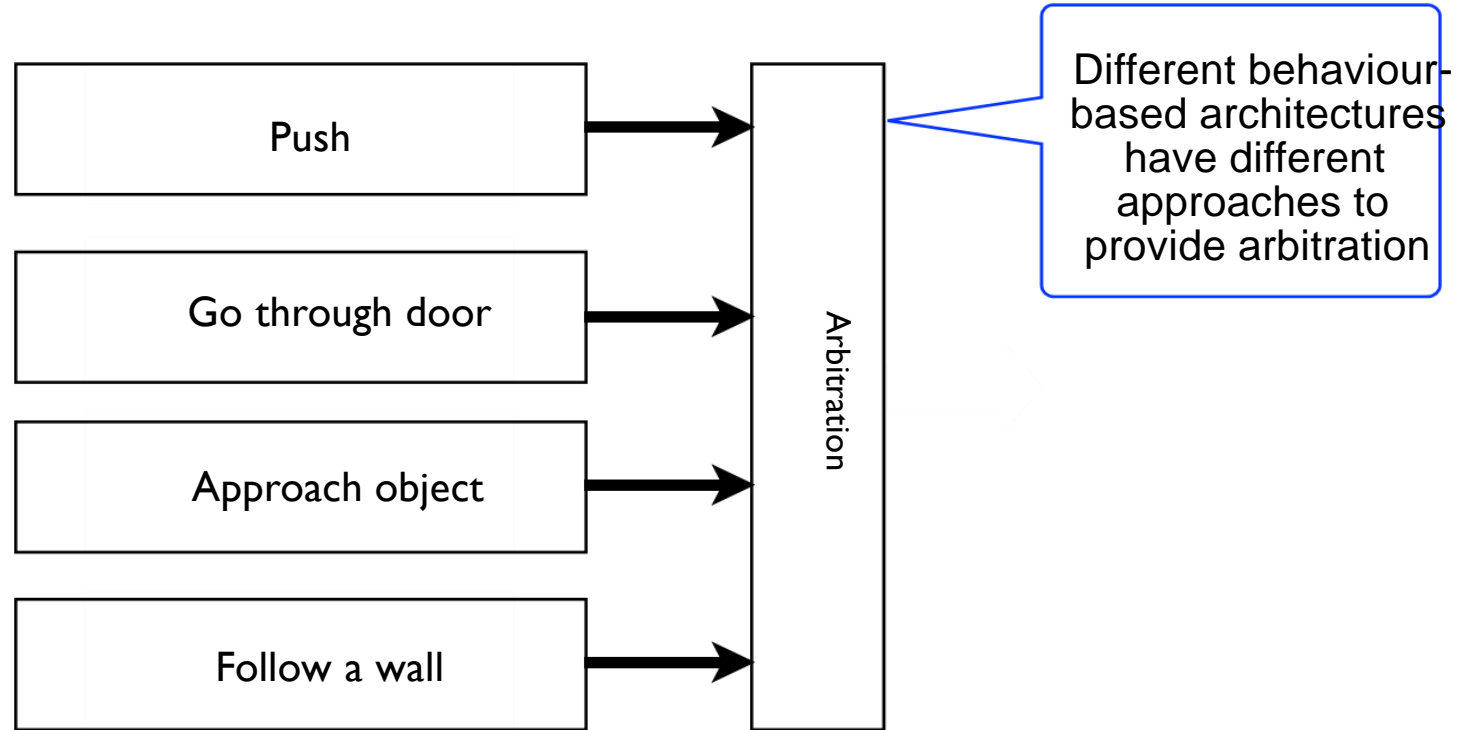
Hybrid Paradigm

- World model, used for planning

- Closed-loop, reactive control

- Many **architectures** of **today** follow this paradigm

```
          ┌─────────┐
          │  Plan   │
          └────┬────┘
               ↕
┌─────────┐         ┌─────────┐
│  Sense  │◄───────►│   Act   │
└─────────┘         └─────────┘
```
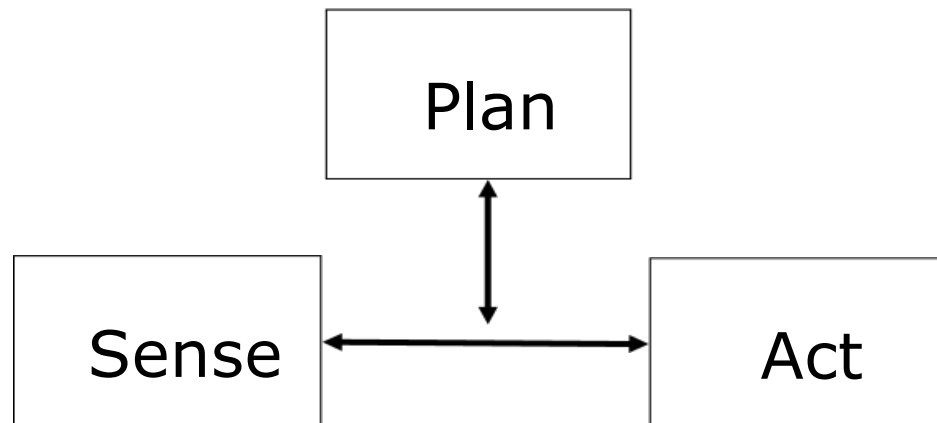
Choice of Robot Control Architecture

Choice of Robot Control Architecture

- In many cases, it is impossible to tell, just by observing a robot's behavior, what control architecture it is using. Only for simple robots, it is often the case.

- However, when it comes to more complex robots, i.e., robots that have to deal with complex environments and complex tasks, the control architecture becomes very important.

Choice of Robot Control Architecture

- The different properties of an environment that will impact the robot's controller (and therefore the choice of control architecture):
  - noisy,
  - speed/response time of sensors and effectors
  - total/partial hidden state/ observable
  - discrete v. continuous state ; static v. dynamic …

- Similarly, the properties of the robot's task impact the choice of the control architecture. The task requirements can constrain the architecture choice.
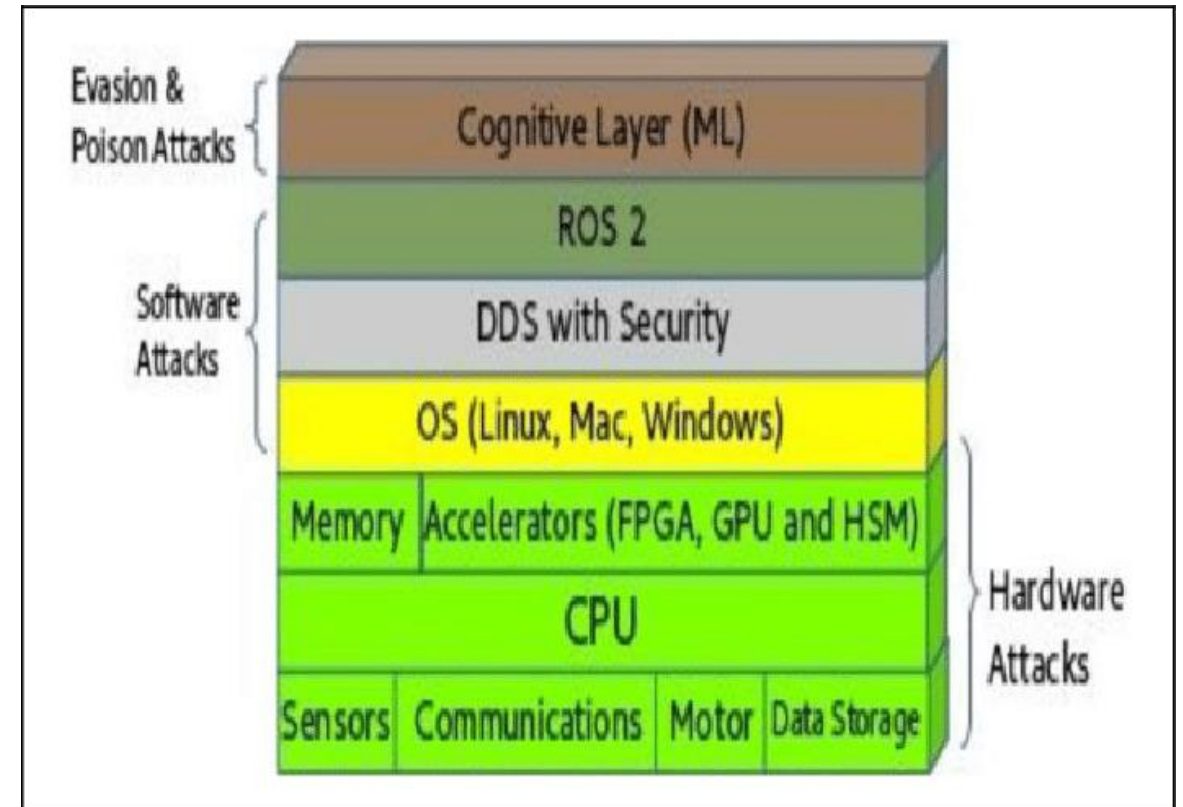
Hardware Types

- Reactive Filters
- ASIC-based (FSM)
- DSP/Micro-processor-based
- Deep-learning Neural network

System Types

- Parallel Paradigm vs. Serial Paradigm
- Asynchronous (Event-driven) vs. Synchronous

Operation System

- ROS (Robot Operation System)
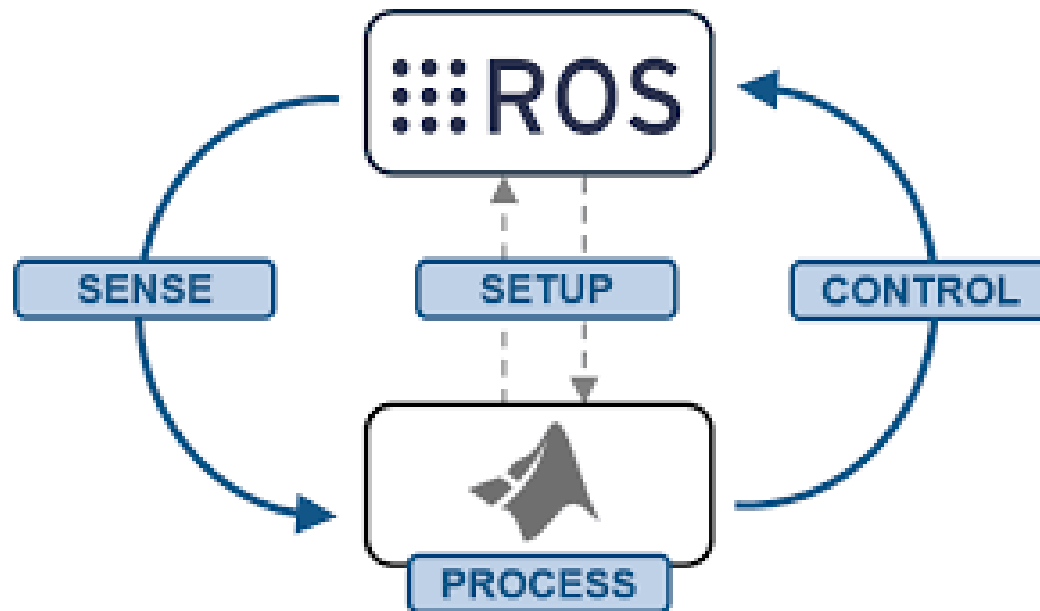
Some Criteria for Selecting a Control Architecture

- **support for parallelism**: the ability of the architecture to execute parallel processes/behaviors at the same time.
- **hardware targetability**:
  - how well the architecture can be mapped onto real-robot sensors and effectors.
  - how well the computation can be mapped onto real processing elements (microprocessors).
- **run-time flexibility:** does the architecture allow run-time adjustment and reconfiguration? It is important for adaptation/learning.
- **modularity:** how does the architecture address encapsulation of control, how does it treat abstraction?
  - Does it allow many levels, going from feedback loops to primitives to agents?
  - Does it allow re-use of software?

Some Criteria for Selecting a Control Architecture

- **niche targetability:** how well the architecture allows the robot to deal with its environment

- **robustness:** how well does the architecture perform if individual components fail? How well does it enable and facilitate writing controllers capable of *fault tolerance*?

- **ease of use:** how easy to use and accessible is the architecture? Are there programming tools and expertise?

- **performance:** how well does the robot perform using the architecture? Does it act in real-time? Does it get the job done? Is it failure-prone?

Criteria

- Response Time

- Environmental Model

- Internal States

Response Time Scale

Response Time-scale is an important way of distinguishing control architectures.

- **Reactive systems** respond to the real-time requirements of the environment,
- while **deliberative system** look ahead (plan) and thus work on a longer response time-scale.
- **Hybrid systems** must combine the two time-scales in an effective way, usually requiring a middle layer; consequently they are often called *three-layer architectures*.
- Finally, **behavior-based systems** attempt to bring the different response time-scales closer together by distributing slower computation over concurrent behavior modules.

Environmental Models

- Another key distinguishing feature between architectures is **representation** of the world/environment, also called **environmental modeling.**

- Some tasks and architectures involve storing information about the environment **internally**, in the form of an **internal representation** of the environment.

Modeling

- For example, while exploring a maze, a robot may want to remember a sequence of moves it has made (e.g., "left, left, right, straight, right, left"), so it can back-track and find its way.
  - Thus, the robot is constructing a representation of its path through the maze.
  - The robot can also build a *map* of the maze, by drawing it using exact lengths of corridors and distances between walls, etc. .
  - This is also a representation of its environment, a model of the world.

- If two robots are working together, and one is much slower than the other, if the fast robot remembers/learns that the other is always slower, that is also a type of a model of the world, in this case, a model of the other robot.
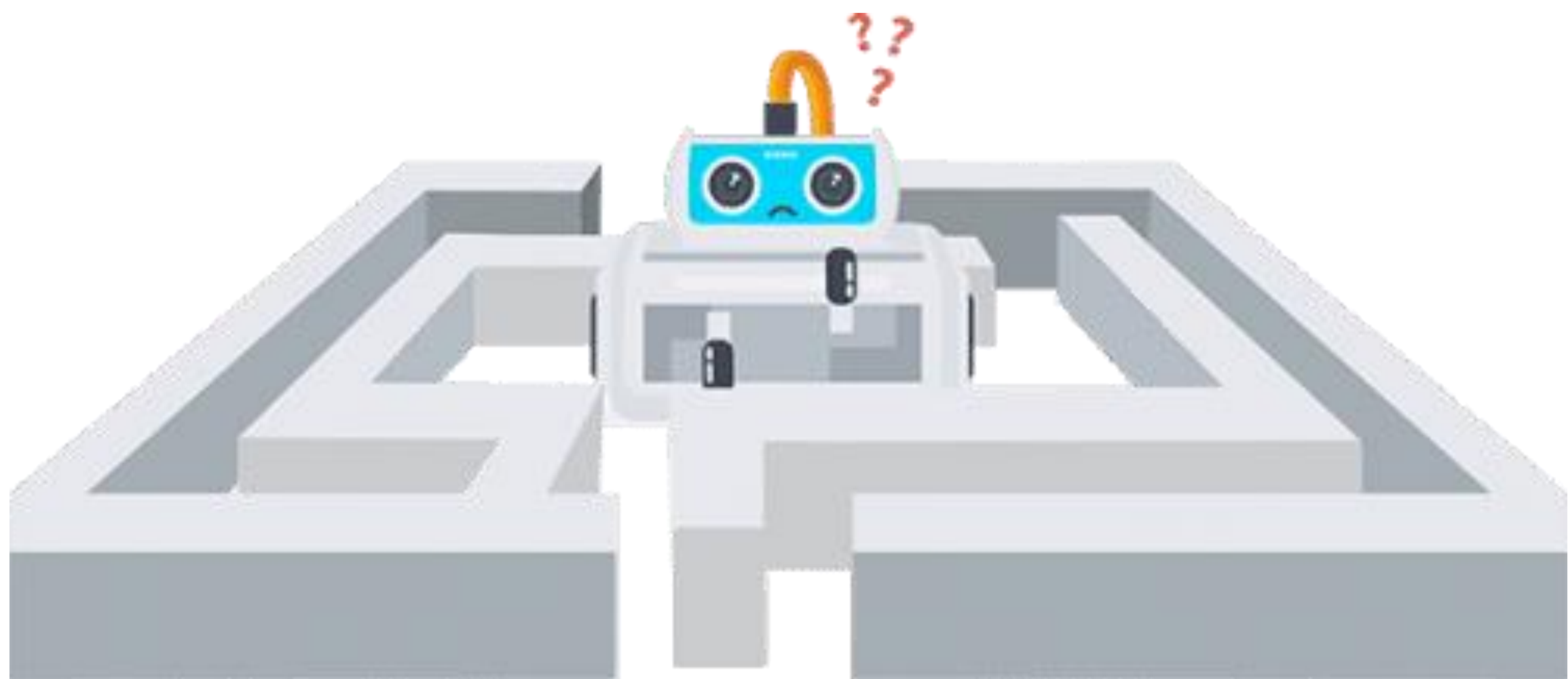
Different Environment Models.

There are numerous aspects of the world that a robot can represent/model, and numerous ways in which it can do it, including:

- **Spatial**   metric or topological: maps, navigable spaces, structures
- **Objects**   instances of detectable things in the world
- **Actions**   outcomes of specific actions on the self and environment
- **self/ego**   stored proprioception: sensing internal state,

                              self- limitations, etc.

- **Intentional**   goals, intended actions, plans
- **Symbolic**   abstract encoding of state/information

Smart Robotic Control

Skill Acquisition & Replay

Deep Neural Network

Reinforcement Learning

Sense | Act

Controller

Robot
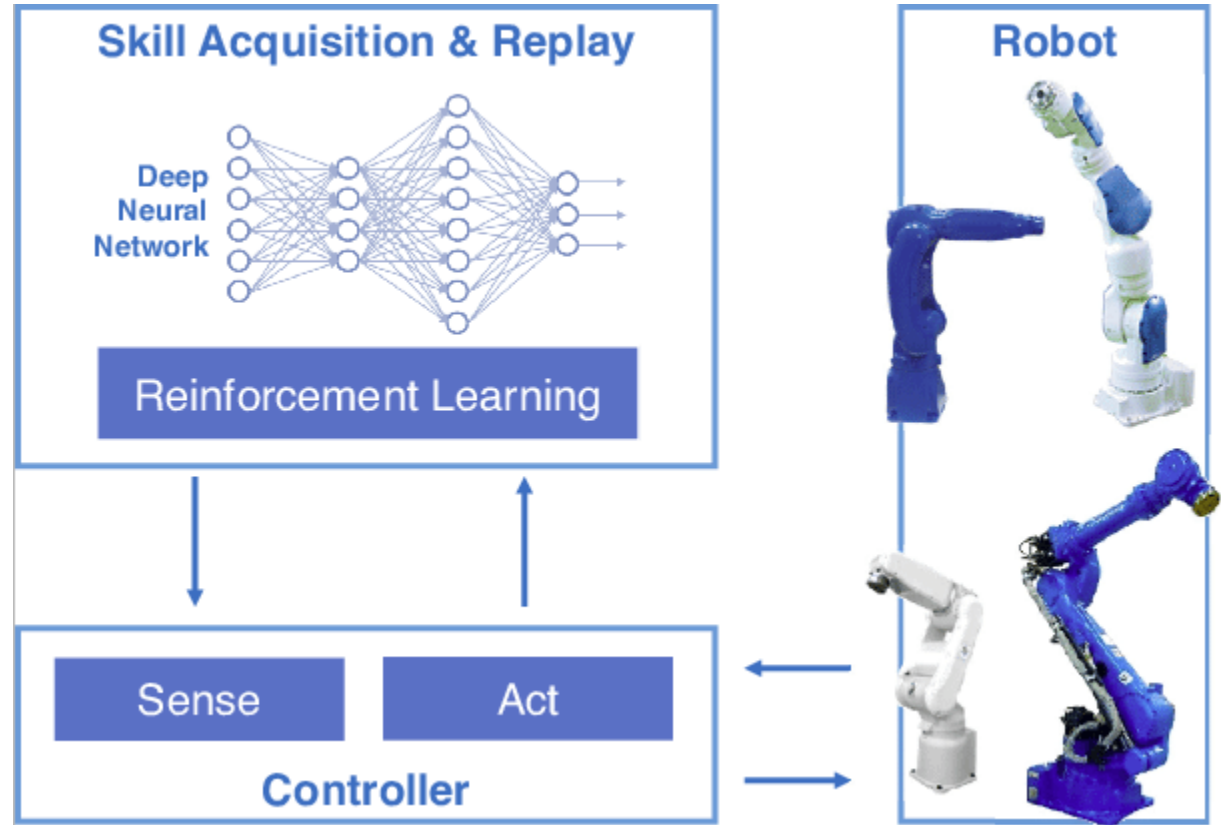
Case Study 1: Building a Temperature Control

Example 1: Building Temperature Control

- One well-known example of a controller is the **temperature control** of a building in winter includes a heater (actuator), sensor (temperature thermistor), and controller (thermostat). The controller switches the heater on when the temperature drops below a certain level and switches the heater off when the temperature rises above a certain level. A disturbance variable may be changes in the outside air temperature or a change in indoor temperature due to a door opening.

Example 1: Building Temperature Control

Element	Temperature Control
Actuator	Valve or switch in heater, Fuel to the furnace
Controller Set Point	Desired temperature indoors
Sensor	Temperature sensor such as a thermocouple or thermistor
Disturbance	Doors opening, wind, temperature outside
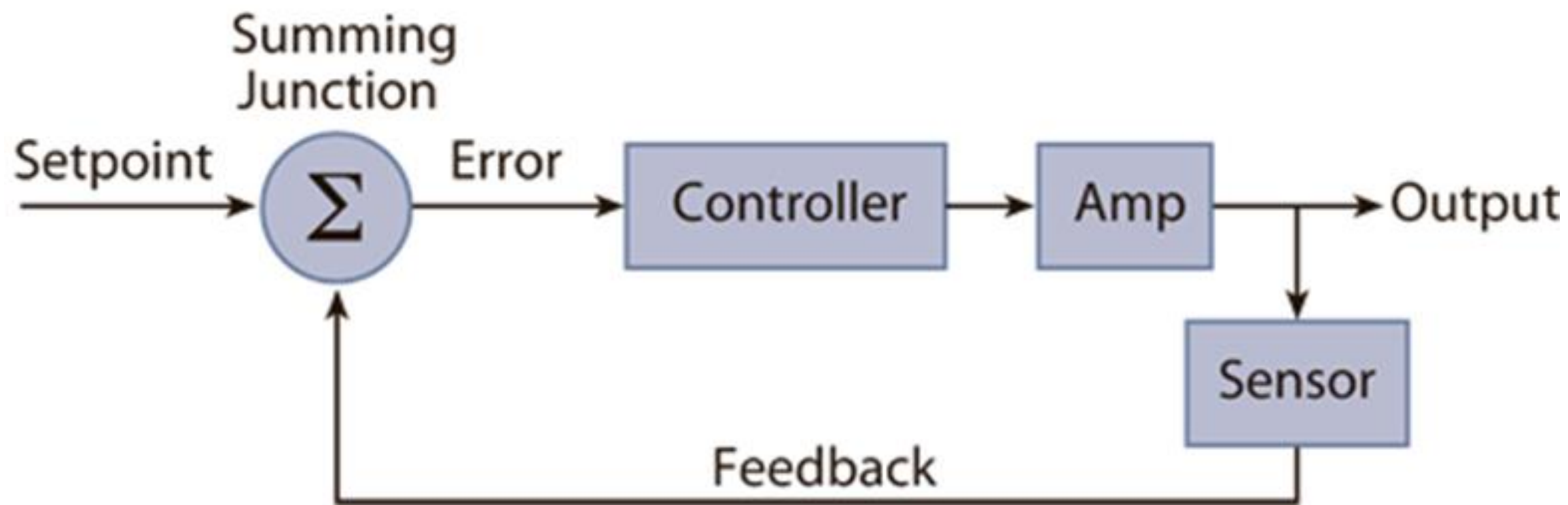
Case Study 2: Automobile Speed Control

Example 2: Automobile Speed Control

- Consider an automobile with an automatic cruise control. The driver may adjust the velocity set point for the controller. The controller adjusts the gas pedal position (actuator) in response to changes in the speedometer reading (sensor).

- A disturbance variable may be an approaching hill or wind that would cause a deviation of the speed from the desired set point.

Example 2: Automobile Speed Control

Element	Cruise Control
Actuator	Gas Pedal Position, Fuel to the Engine
Controller Set Point	Desired Speed (mph or km/sec)
Sensor	Speedometer, measured velocity
Disturbance	Hills, wind, other cars

Process Schematic

SECTION 1
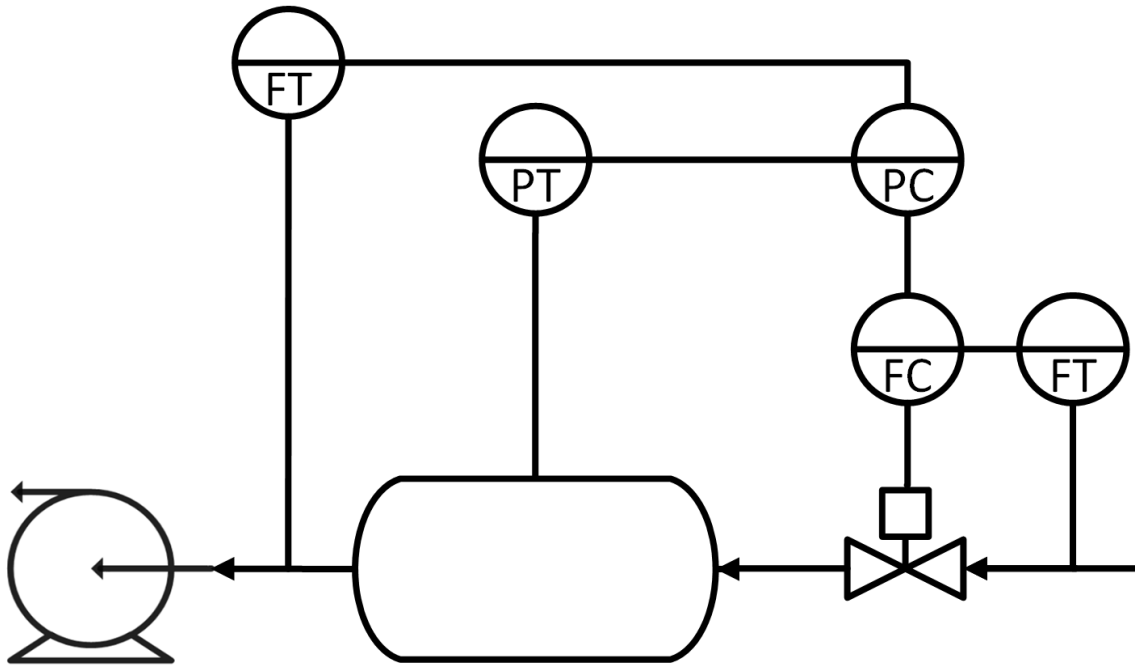
Process Schematic

- A process schematic can be augmented with circles that reveal the type of transmitter or controller.

- Two letters in the circle name indicate the quantity being measured and whether it is a transmitter or a controller.

Measurement (Letter 1)	Description
A	Analyzer (mole or mass fraction)
C	Concentration
F	Flow
I	Current
L	Level
P	Pressure
R	Resistance
S	Speed
T	Temperature
V	Vibration
X	Miscellaneous

Device (Letter 2)	Description
C	Controller
E	Element
I	Indicator
M	Motor
S	Switch
T	Transmitter/Transducer
V	Valve
X	Miscellaneous
Z	Safety Device

Process Schematic

The diagram above has a *FT=Flow Transmitter*, *PT=Pressure Transmitter*, *PC=Pressure Controller*, *FC=Flow Controller*, and *FT=Flow Transmitter*. The *FT/FC* and *PT/PC* both form feedback loops because they measure and control the same quantity. The *FC* receives a set point from the *PC* above to create a **cascade controller** of two feedback loops. The upper *FT* measures a pressure disturbance and provides a **feedforward element** to the pressure control. Feedforward and cascade controllers are added to reject additional disturbances and are more advanced than common feedback control.

Block Diagram

SECTION 1

Block Diagrams

- Block diagrams show the blocks of a control system with the interconnections that determine the direction and connections of inputs and outputs. A feedback control system consists of a sensor, actuator and controller that are connected with information flowing in a loop.

- The loop is created with the sensor providing information to the controller. The controller changes the controller output that then changes the process. The process is measured again and the cycle repeats.

Block Diagram
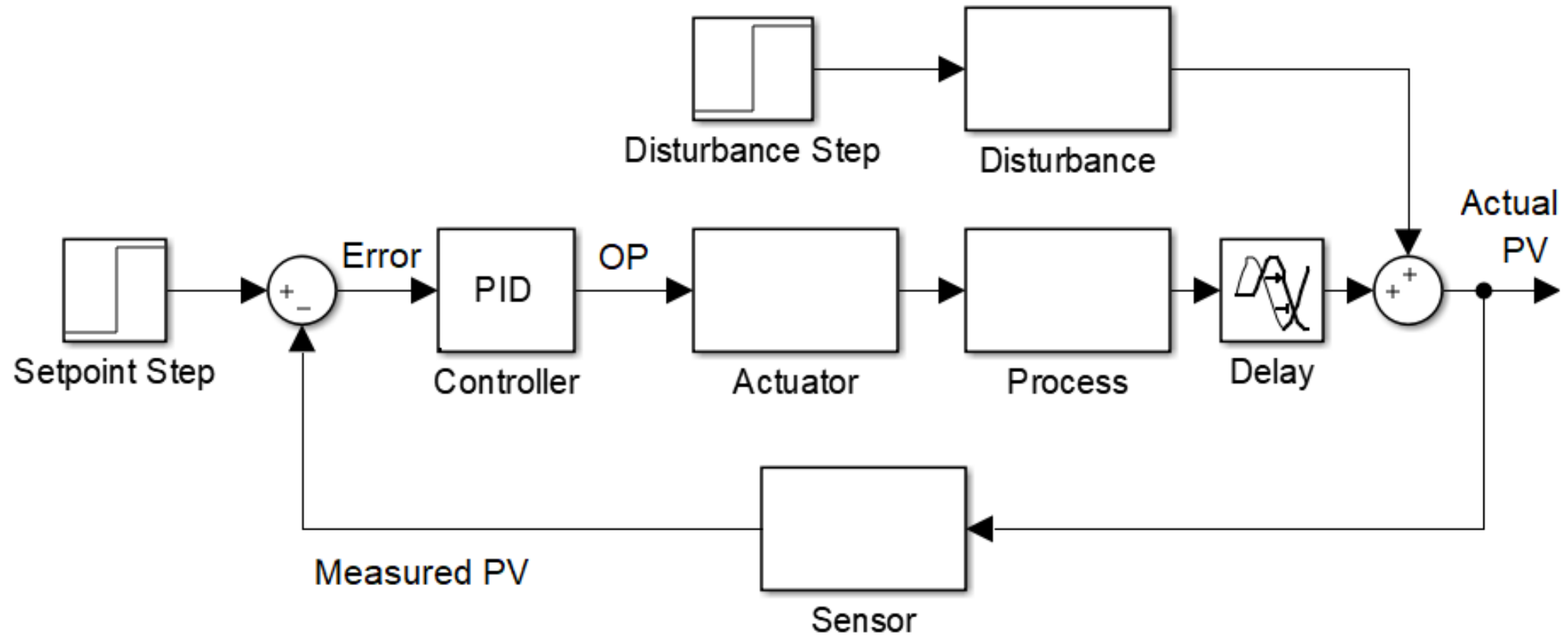
Block Diagram

- Block diagrams are different than a process diagram in that it is a diagram of the flow of information, not necessarily how the pieces of equipment are physically placed.

Control Terminology

SECTION 1

Control Terminology

- There is different terminology when talking about common controllers such as Proportional Integral Derivative (PID) or advanced controllers such as Model Predictive Control (MPC).

- Below is a table of some of the terminology and associated abbreviations.

Control Terminology

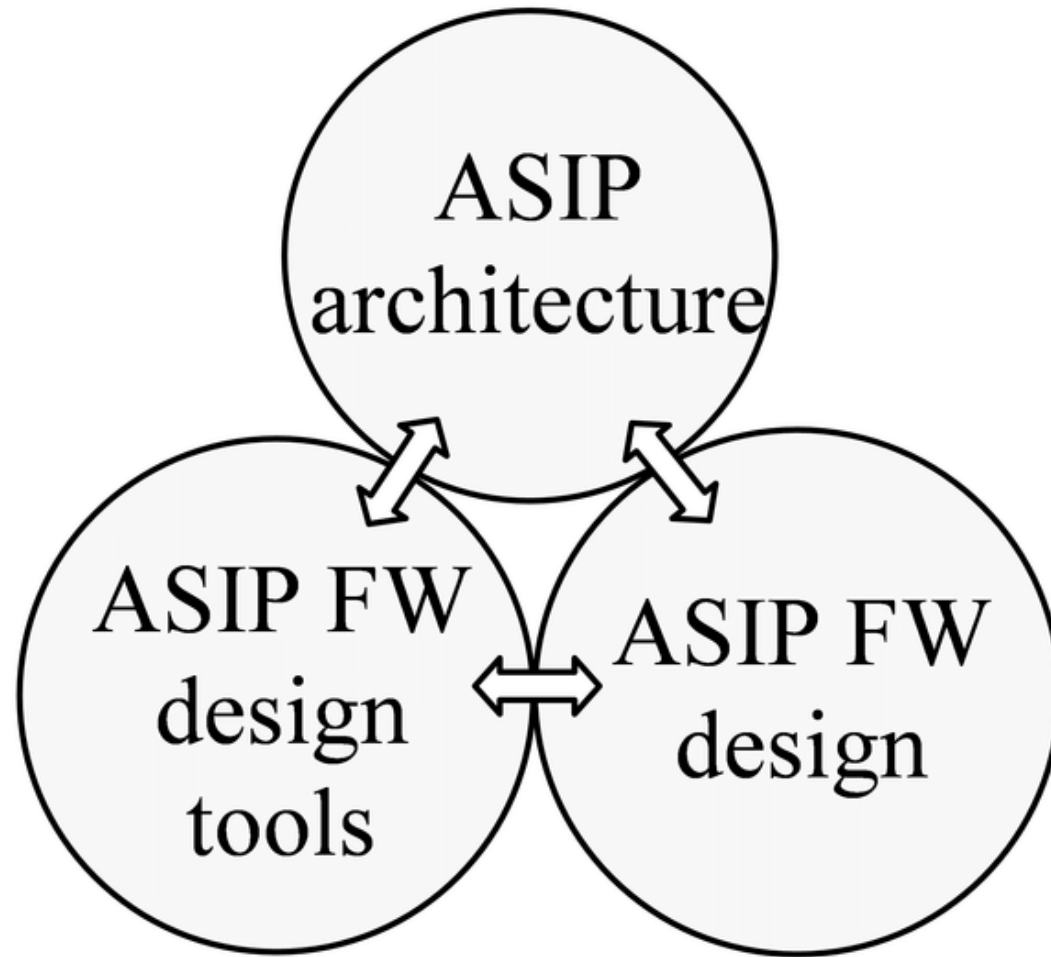| Element	Common Control (PID)	Advanced Control (MPC)
Actuator	Controller Output (CO) or Output (OP)	Manipulated Variable (MV)
Controller	Set Point (SP)	Set Point (SP) or Range (SPHI/SPLO)
Sensor	Process Variable (PV)	Controlled Variable (CV)
Disturbance	Disturbance Variable (DV)	Disturbance Variable (DV)

Design

General Purpose Processors

❖ Such as x86, ARM

❖ High-end processors consume thousands of designer-years.

❖ Aim to MAX flexibility for all applications

❖ Compiler and OS must be designed for all applications, entry level is too high

❖ x86 price is high

ASIP

❖ Is designed for a domain of applications

❖ Its assembly instruction set is designed to accelerate most appearing function and critical functions.

❖ The hardware cost and power consumption are relatively much lower. The price can be very low under volume sales.

❖ It is usually for predictable computing

ASIC

❖ Non-programmable, usually can reach the lowest power and silicon cost for only one application.

❖ ASIC was a dominant solution when the level of integration was limited.

❖ Because of the high NRE cost, it will be gradually less popular
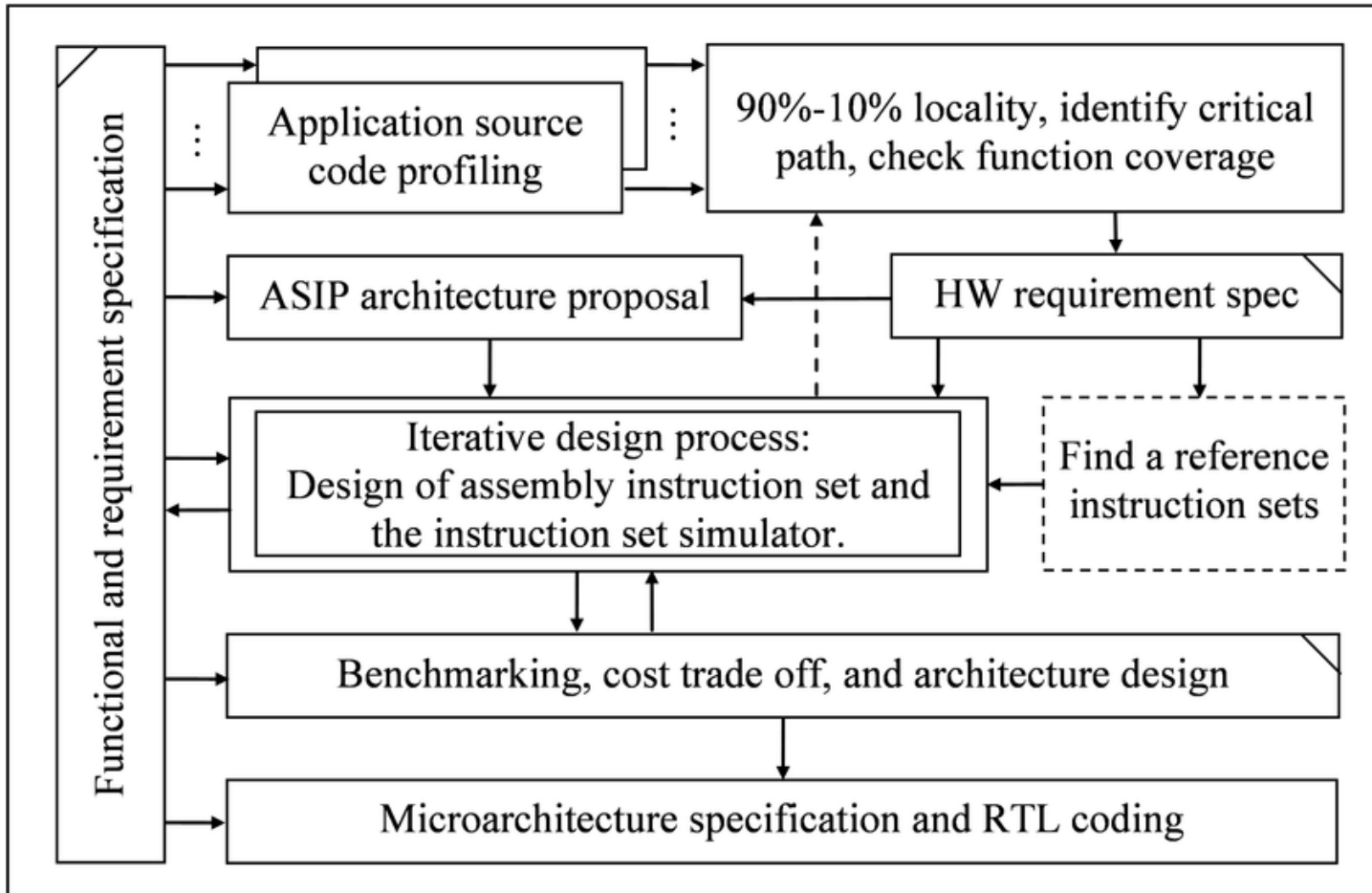
DSP ASIP
Hardware Design Flow

(a) Behavior model

(b) HW implementation

The highest functional coverage

Low data access latency

Handling control complexity

Low memory cost

The highest performance

Low silicon cost

ASIP instruction set

Low design cost

Low power consumption

High I/O bandwidth

Compiler friendly instruction set

Behavior code adapt to HW (1)	(1)	Build assembly library
Compile C code to assembly (2)		Assembly coding (3)
(4) Link objective codes and allocate codes to program memory		
Generate finally binary code and run it on simulator (5)(6)		
Verify codes, benchmarking, and release the design (5)(6)		

(a) The method translating source code to qualified assembly code

(1)	C-HW-adapter
(2)	C-compiler
(3)	Assembler
(4)	Objective linker
(5)	ASM simulator
(6)	Debugger

(b) Toolchain