The Nature of Computing

INEL 4206 – Microprocessors

Lecture 2

Bienvenido Vélez Ph. D.

School of Engineering

University of Puerto Rico - Mayagüez

Some Inaccurate (Although Popular) Perceptions of Computing

- Computing = (Electronic) Computers
- Computing = Programming
- Computing = Software

Computing = Computers

Computing is about solving problems using computers

A.K.A. The Computing Device View of Computing

Computing = Programming

Computing is about writing
programs for computers

A.K.A. The Programming Language view of Computing

Computing = Software

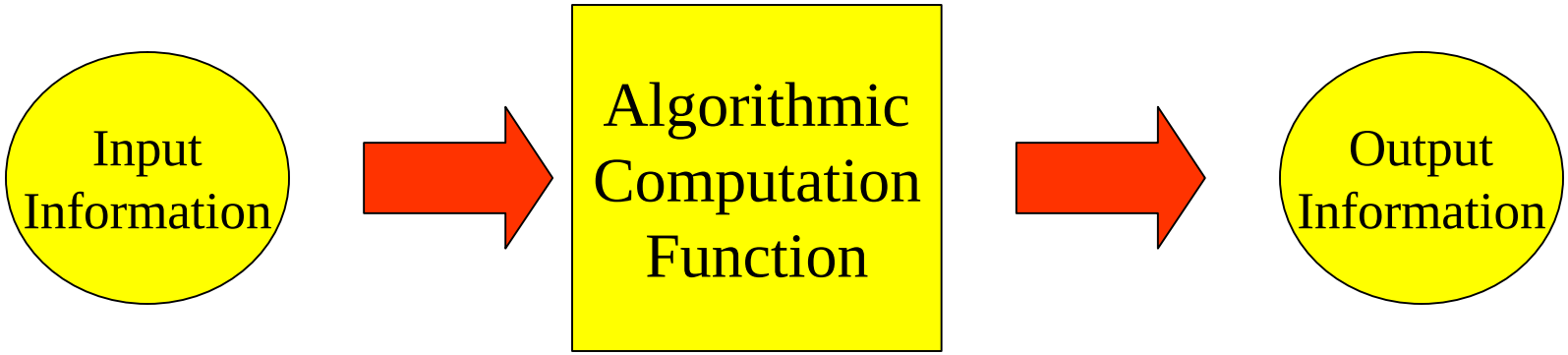
Computing is not concerned with hardware design

A.K.A. The “Floppy Disk” view of Computing

Outline

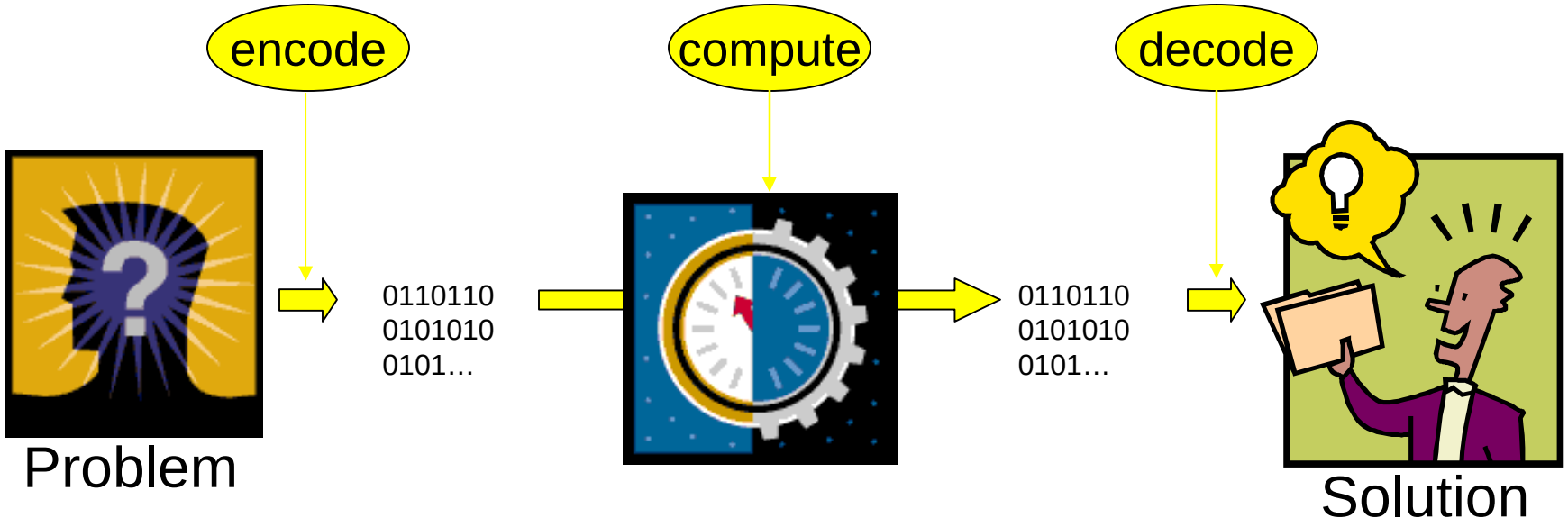
- What is Computing?
- Computing Models and Computability
- Interpretation and Universal Computers
- Abstraction and Building Blocks

What is computing then?



Computing is the study of Computation:
the process of transforming information

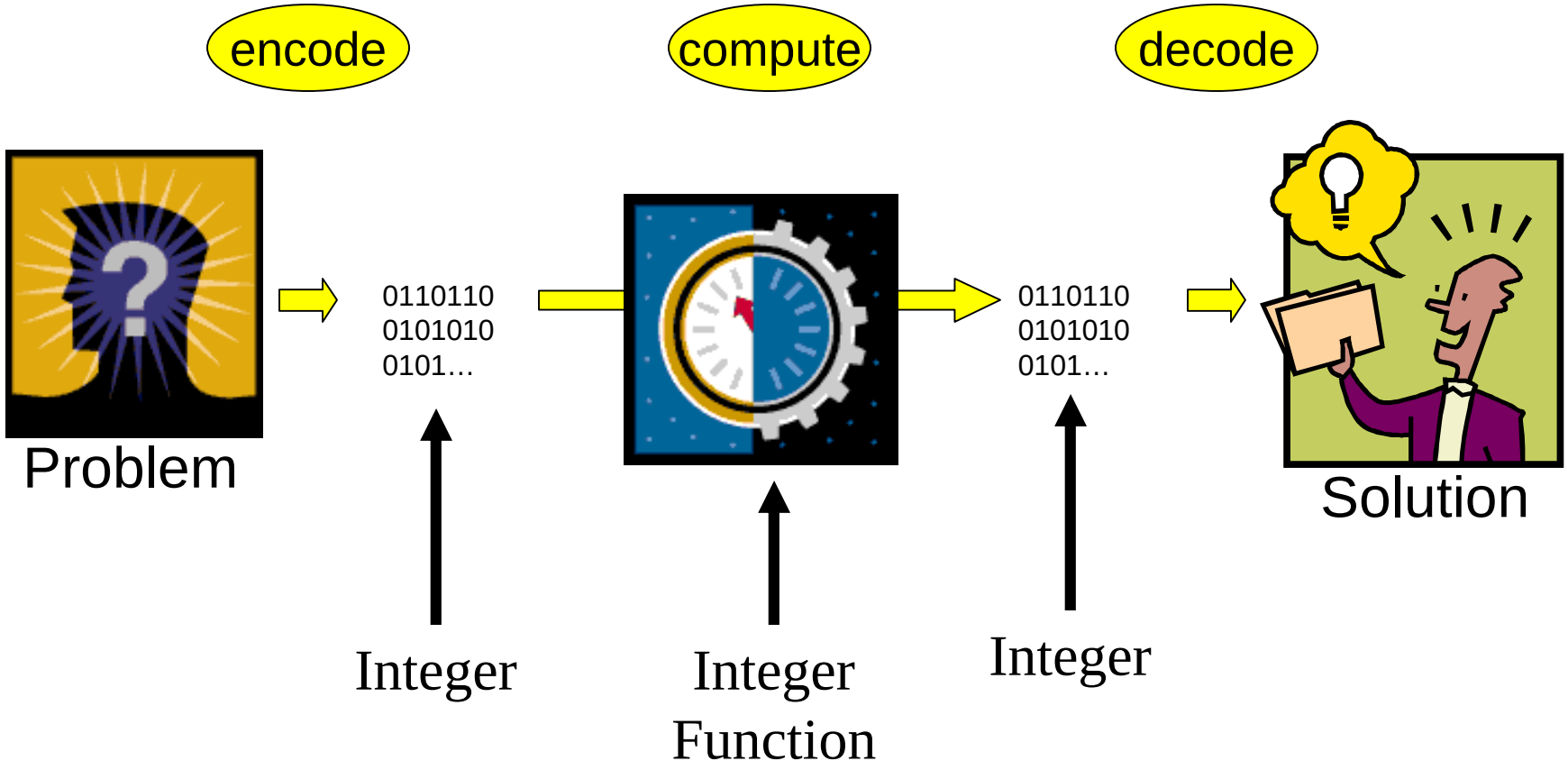
The Computation Process



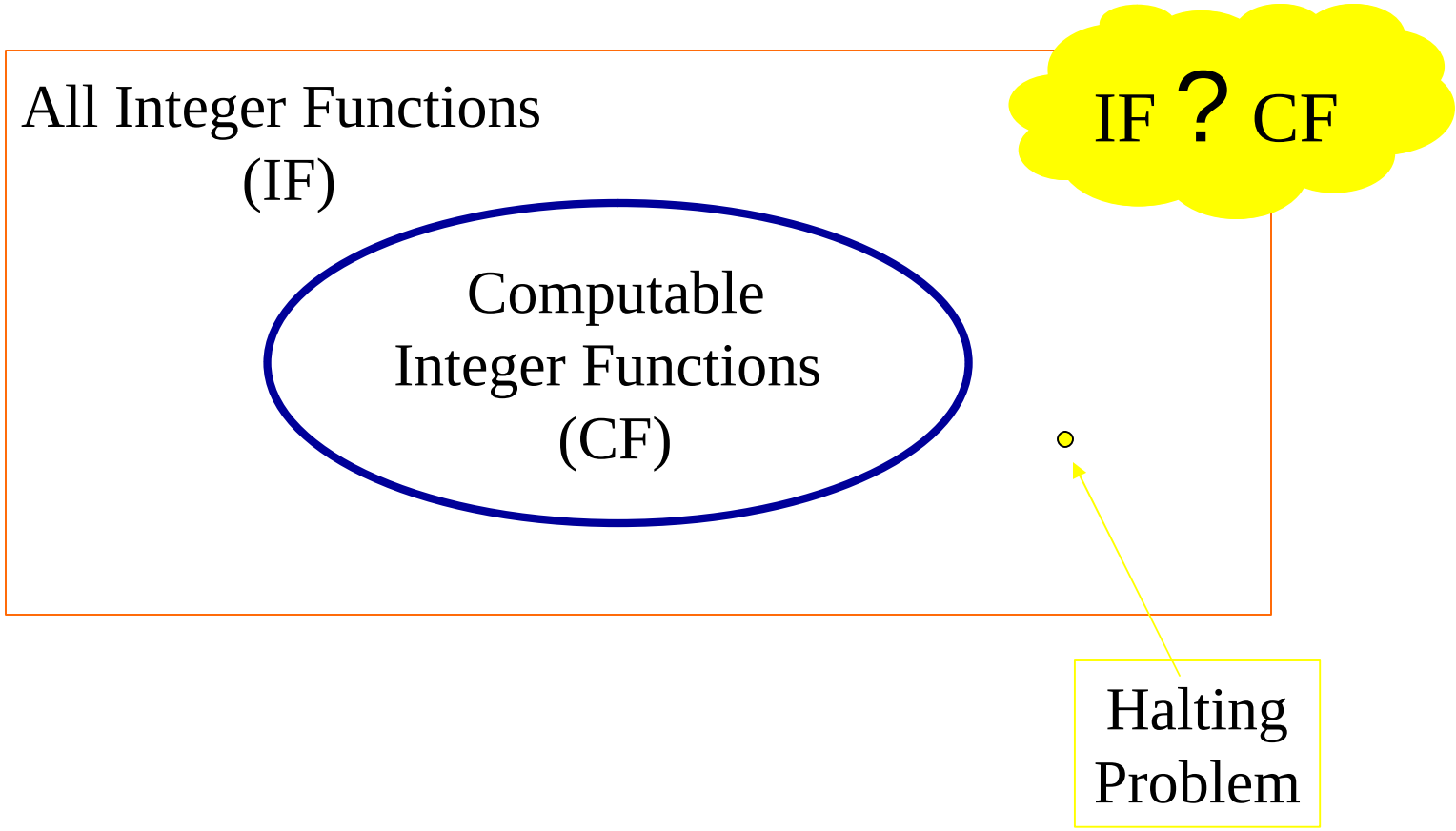
Fundamental Questions Addressed by the Discipline of Computing

- What is the nature of computation?
- What can be computed?
- What can be computed efficiently?
- How to build practical computing devices?

Computers as Integer Functions

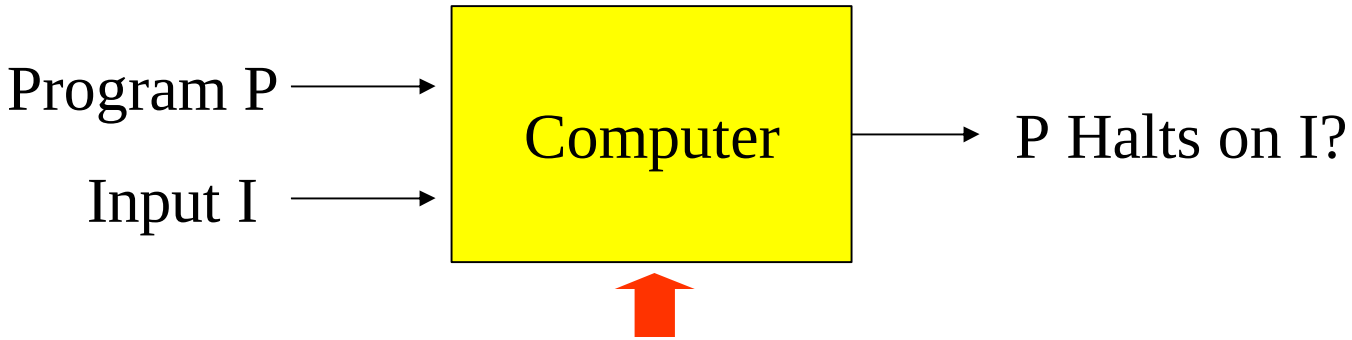


Computability



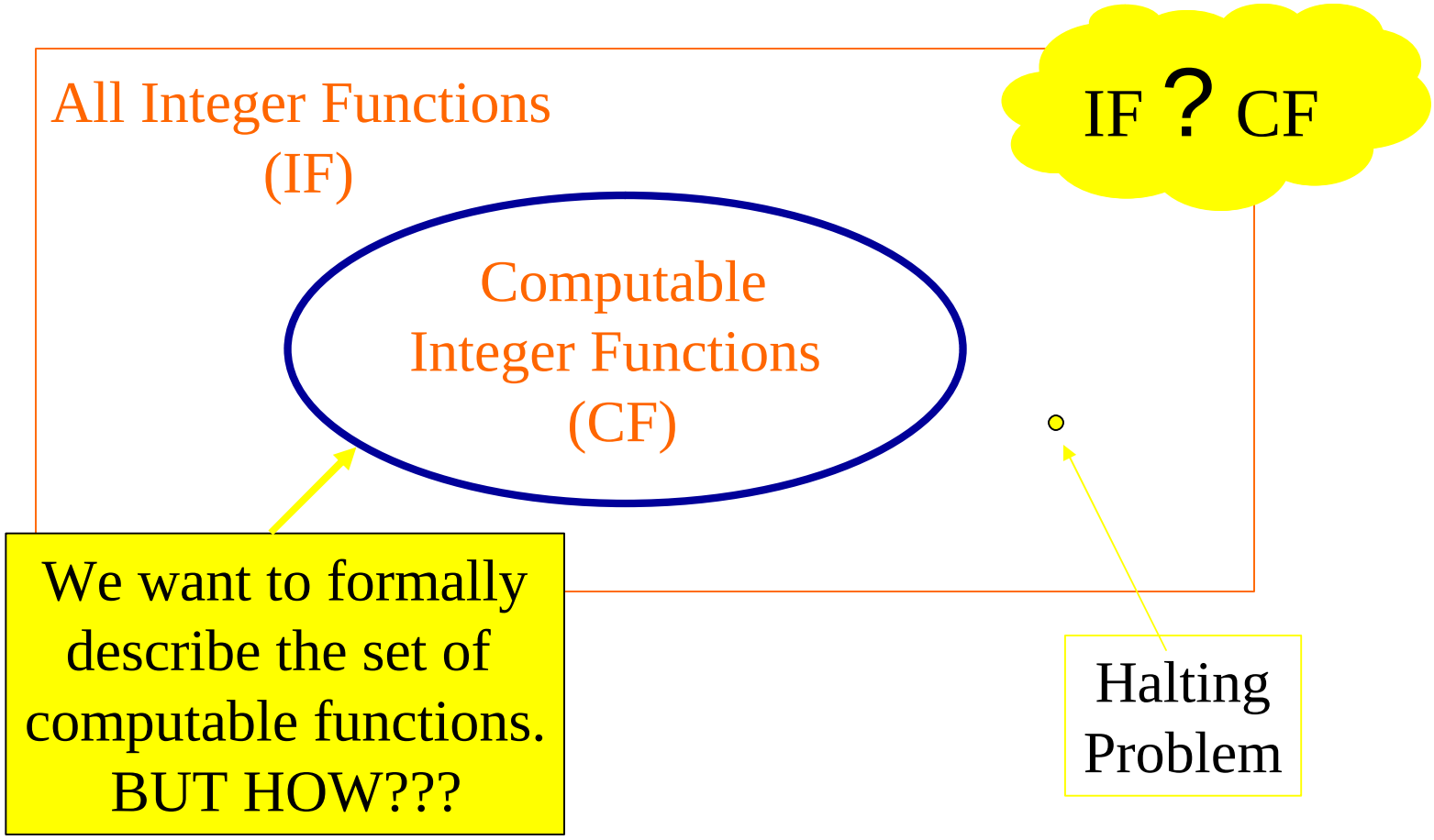
The Halting Problem
(Alan Turing 1936)

Given a program P and an input I to P
determine if P stops on I



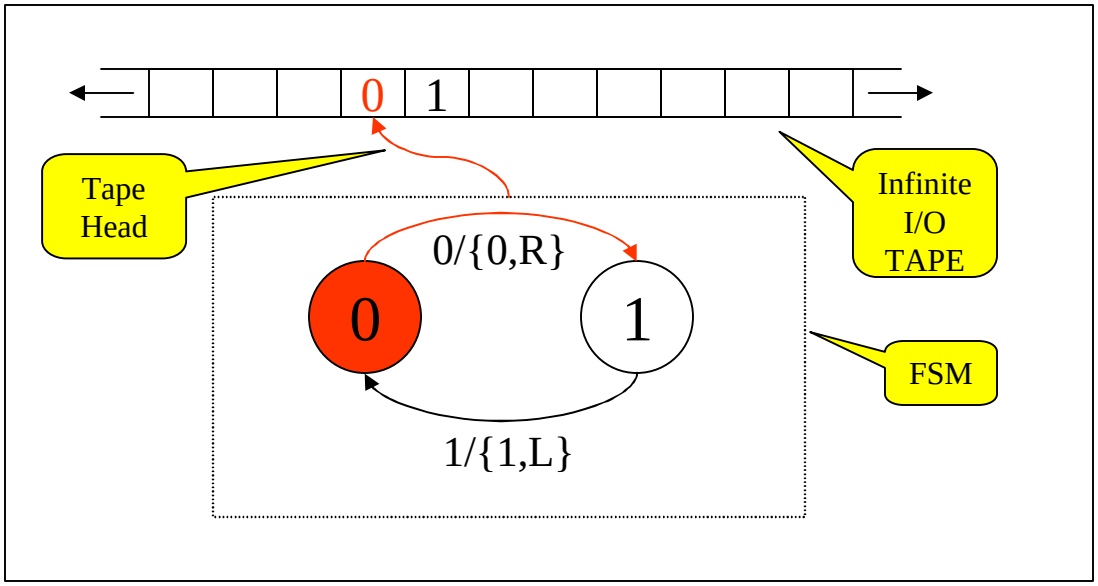
We cannot build this machine … period

Computability



Mathematical Computers: The Turing Machine (1936)

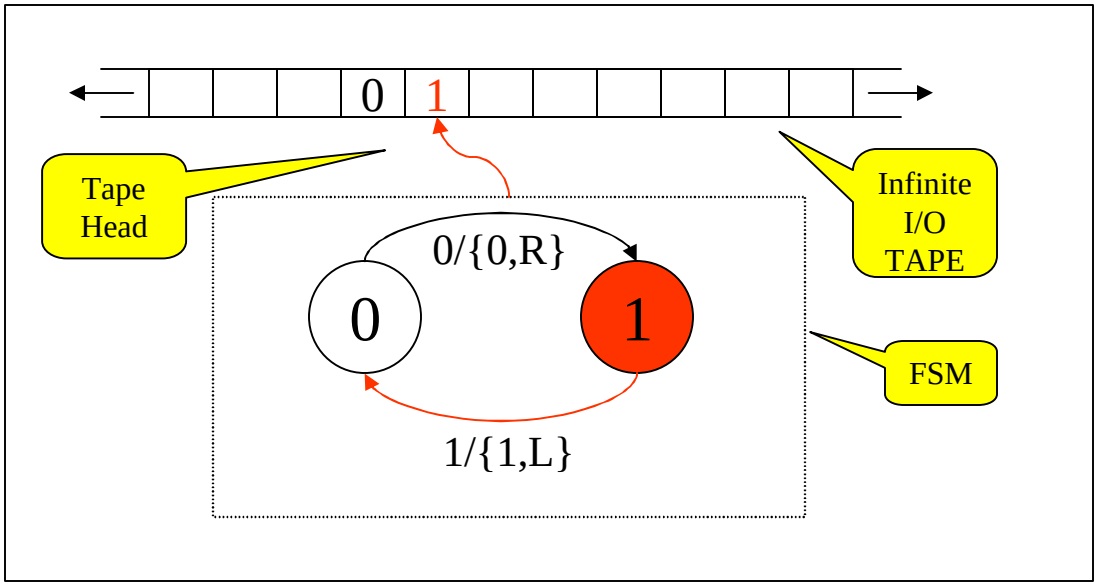
Alan Turing



Turing demonstrated how to solve several problems using his LCM computing model

Mathematical Computers: The Turing Machine (1936)

Alan Turing



Turing demonstrated how to solve several problems using his LCM computing model

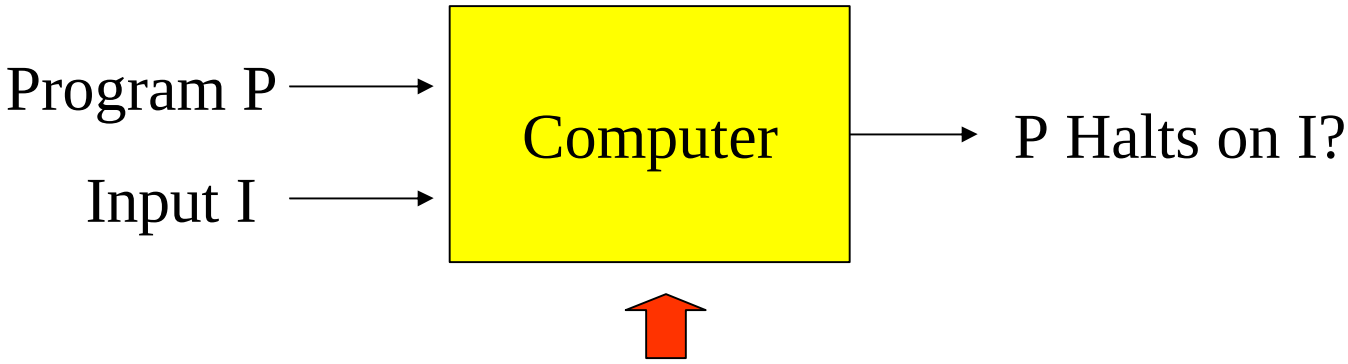
A TM Machine Recognizing $a^n b^n$

Rutgers Turing Machine Example

http://www.rci.rutgers.edu/~cfs/472_html/TM/anbnTM.html

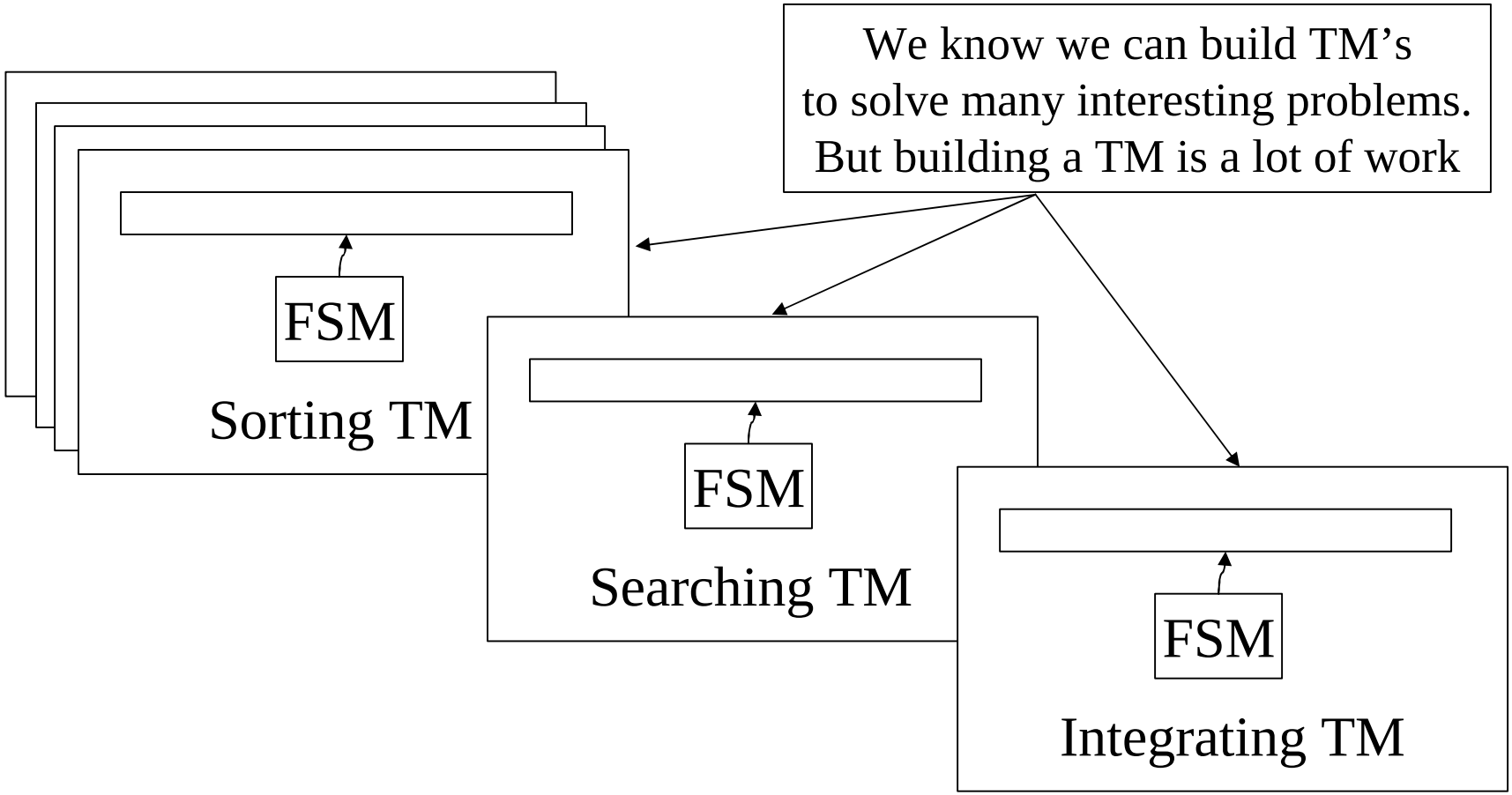
The Halting Problem
Proof Sketch (Blackboard)

Given a program P and an input I to P
determine if P stops on I



Cannot build this … period

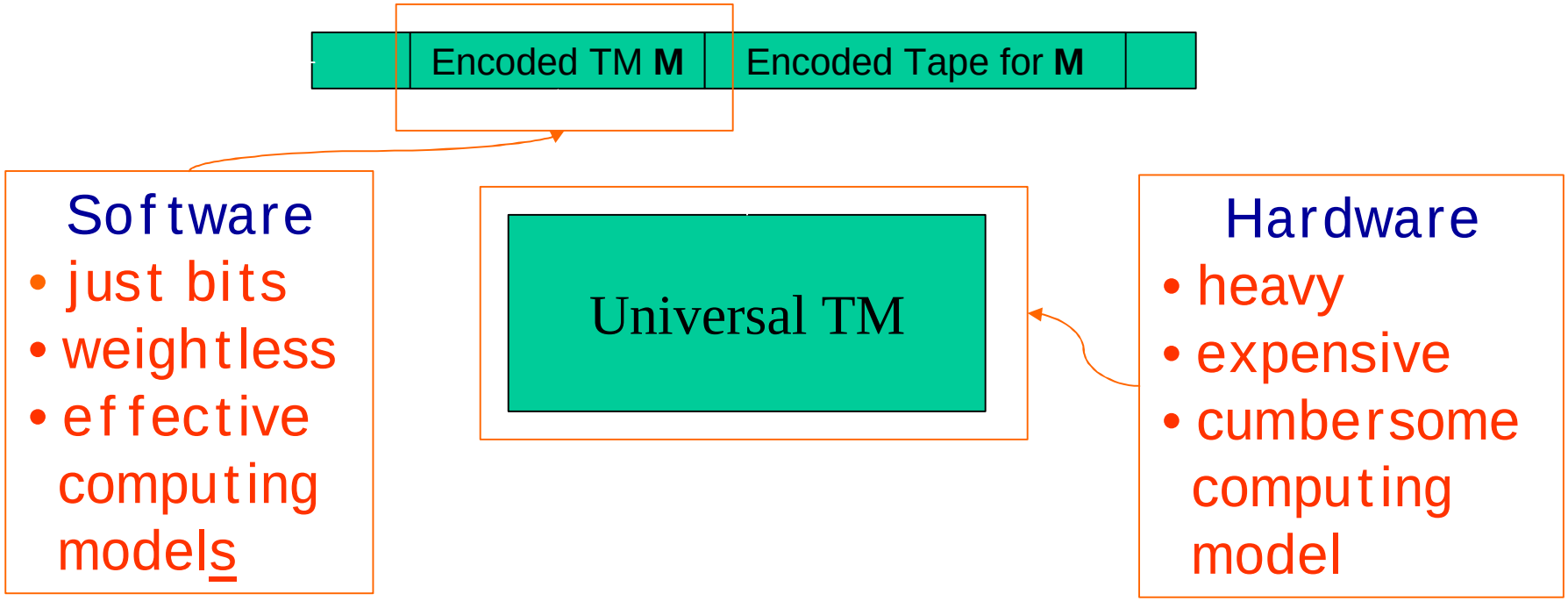
Ad-hoc Turing Machines



Can we build ONE general purpose TM?

The Universal Turing Machine (UTM)

The Paradigm for Modern General Purpose Computers



- Capable of Emulating Every other TM
- Shown computable by Alan Turing (1936)

BIG IDEAS: INTERPRETATION & PROGRAMMABILITY!!!

Other Familiar Models of Computation

- Combinational Circuits
- Sequential Circuits (FSM's)
- Pentium Instruction Set Architecture
- Lambda Calculus
- Recursive Functions
- C, C++, Java, C#, etc…

Can you tell which ones are Turing Universal,
or which ones can emulate any other Turing Machine?

Church's Thesis

Alonso Church

"Any realizable computing device can be simulated by a Turing machine"

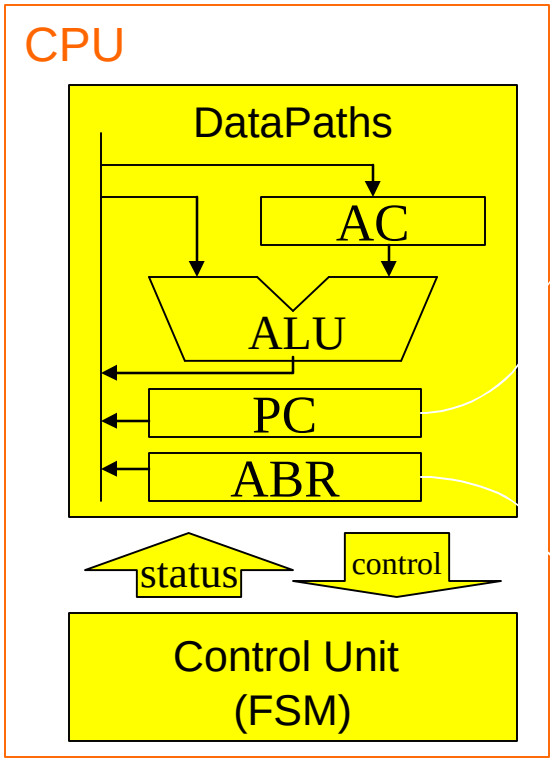
"All the models of computation yet developed, and all those that may be developed in the future, are equivalent in power."

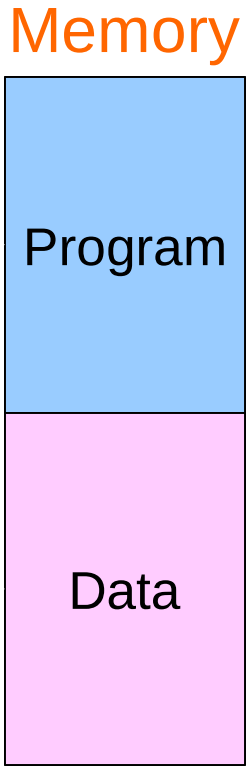
Issues not considered: Size, Programmability, Performance
But they must be considered if one is to build …

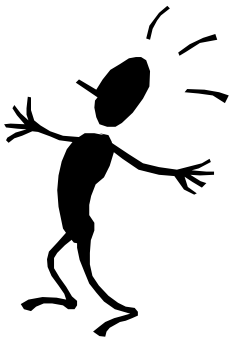
Practical Universal Computers

(John) Von Neumann Architecture (1945)


John von Neumann, 1950's



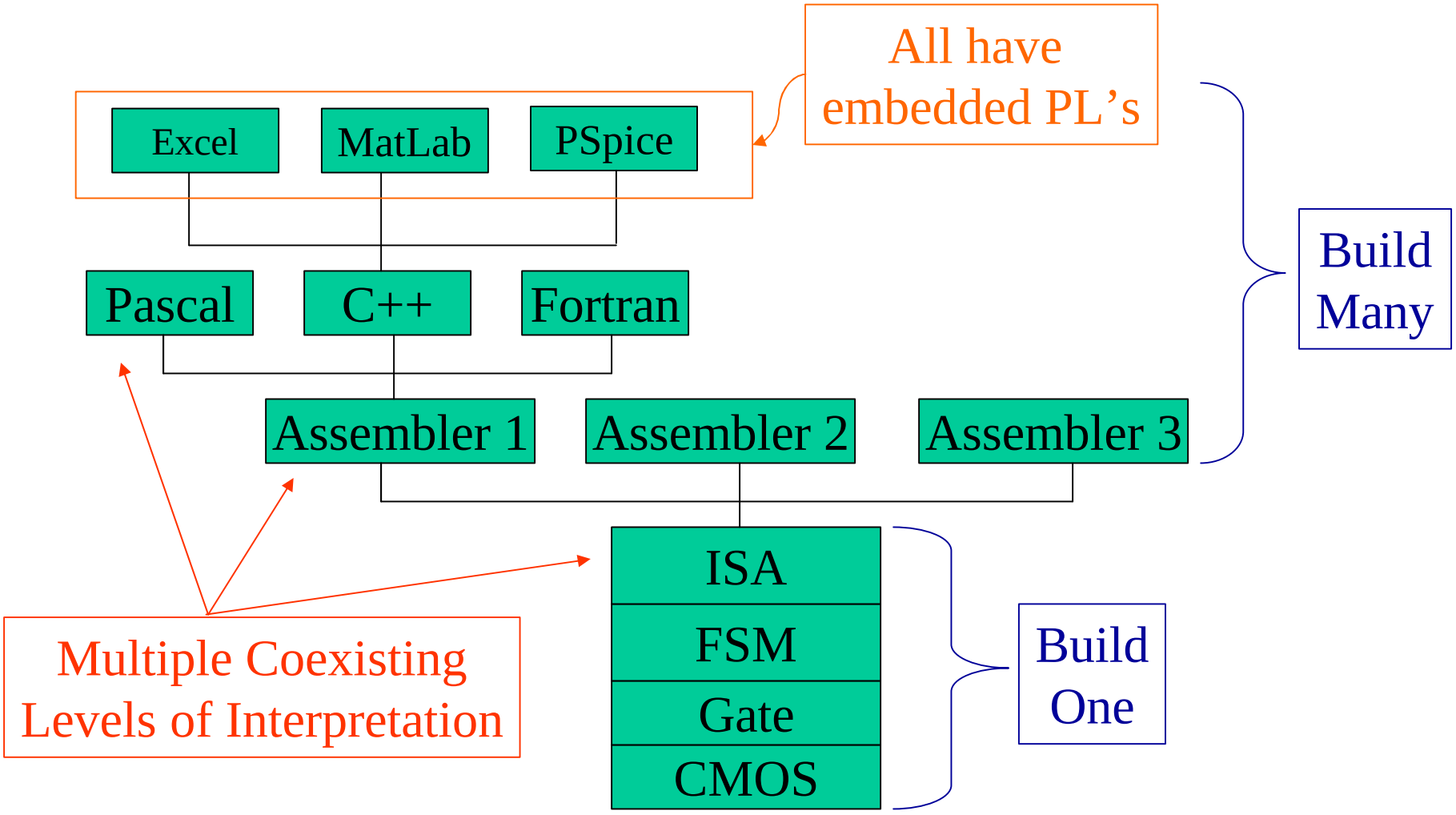




CPU is a universal TM

An interpreter of some programming language (PL)

Computing in Perspective



Programming is the art/science of encoding algorithms

Computing in Perspective

people

Application Programs

High Level Language

INEL 4206

Low Level Language (ISA)

computers

Computer Human Interaction, User Interfaces

CS1/CS2, Programming, Data Structures

Programming Languages, Compilers

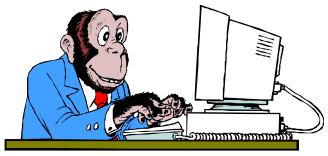
Computer Architecture

Each abstraction layer implements an INTERPRETER of some computing/programming model

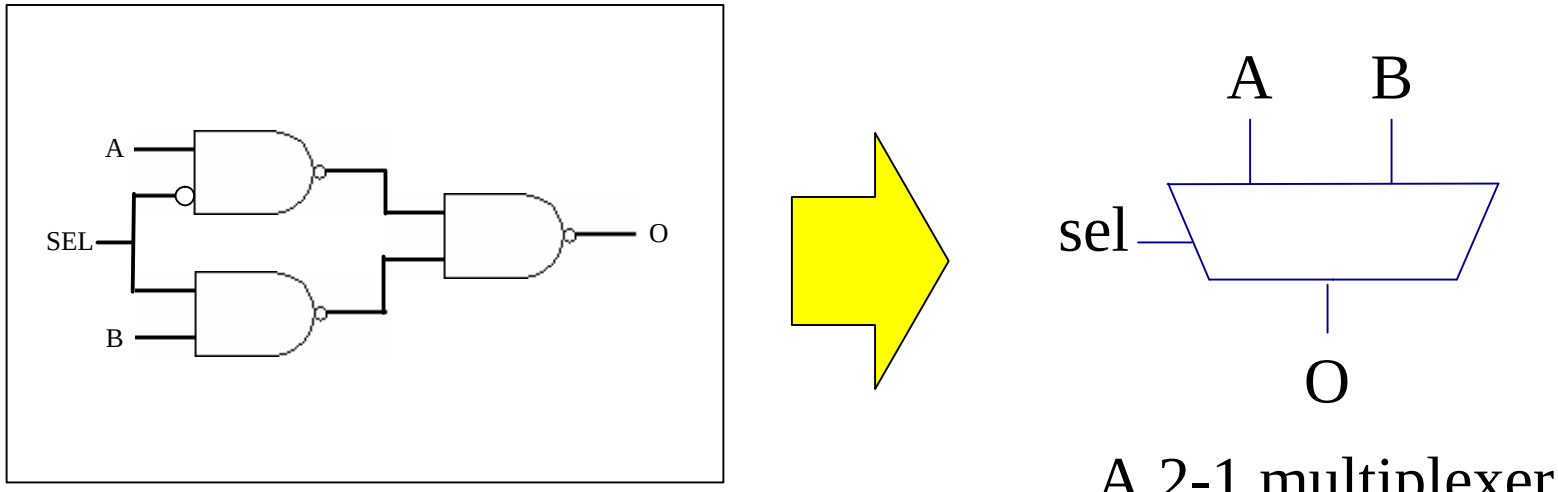
Why Abstraction Layers?

- Resilience to change:
  - Each layer provides a level of indirection
- Divide and Conquer Approach:
  - Can work on one small semantic gap at a time
- Building Block Approach:
  - Can build many higher layer on same l

Because we know of no other way of doing anything



Hardware Building Blocks



A 2-1 multiplexer

Gate-Level Logic Provides a Computing Model

Software Building Blocks

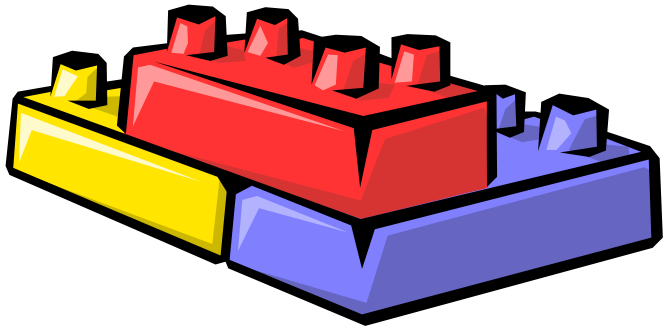
The function is one of the most ubiquitous abstraction tools

```
// MUX – Implements a 2-1 binary multiplexer
bool MUX(bool a, bool b, bool sel) {
  switch(sel) {
    case 0: return a; break;
    case 1: return b; break;
  }
}
```

Other abstraction tools include:
structures, classes, modules

What Makes a Good Building Block

- Provides a clear and simple contract
- The contract hides irrelevant detail
- The contract is general and orthogonal
- The contract is easy to remember



Some Properties of a Good Programming Language

- Does not hide expressive power of lower layers
- Can be efficiently interpreted
- Provides adequate higher level abstractions
- Provides a variety of constructs for creating new abstractions, layers and modules
- Achieves all of the above with minimal complexity

Summary

- Computing = Information Transformation
- Information Transformation = Integer Functions
- Some integer functions are not computable
- Turing Machine computations = All computations
- Universal Computer = Universal TM
- Interpretation => Programmability => Flexibility
- Building blocks are abstract contracts

Summary

"Computer Science is no more about computers than Astronomy is about telescopes"

E. W. Dijkstra
1930-2002
1972 Turing Award