Introduction to Computers and Programming

Prof. I. K. Lundqvist
kristina@mit.edu , 33-318

(Updated by Heidi Perry, hperry@draper.com & Emily Braunstein, ebraunstein@draper.com)

Readings: Ada/Mindstorms 2.0 User's Guide and Reference Manual
http://www.usafa.af.mil/dfcs/Ada_Mindstorms_manual.htm

Lecture 4
Oct 12 2005

Today

- Lego Mindstorms Logistics
- RCX 2.0 Hardware Architecture
- RCX 2.0 Software Architecture & Basics of Programming with Legos
- Programming Example

Lego Mindstorms Logistics

- Teams of 3 (or 4) per 1 Lego kit
  - Collaborate on the pset questions that involve Lego Mindstorms
- Instructions on taking ownership of the kits will be distributed via email
- Handle the kits and parts responsibly
  - Kits will be used in upcoming courses (16.00, 16.35 and 16.unified)

The Lego Mindstorms Robotics Invention System 2.0

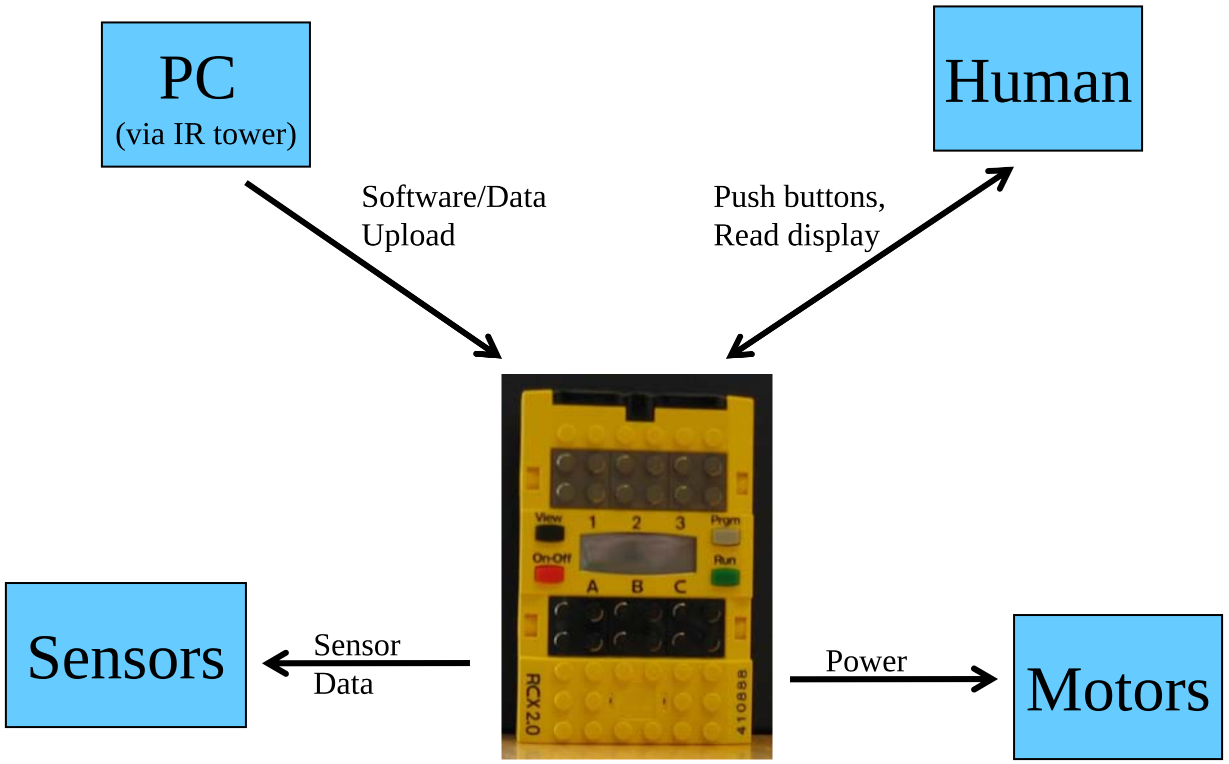
- RCX™ Microcomputer
- CD-ROM Software
- USB Infrared Transmitter
- 718 pieces, including:
  - 2 Motors
  - 2 Touch Sensors
  - 1 Light Sensor
  - (+1 Rotation Sensor)

http://www.latoys.com/lego/images/lego_3804.jpg http://www.gibsonteched.com/media/legomindstormparts2.jpg

Today

- Lego Mindstorms Logistics
- RCX 2.0 Hardware Architecture
- RCX 2.0 Software Architecture & Basics of Programming with Legos
- Programming Example

RCX Context Overview



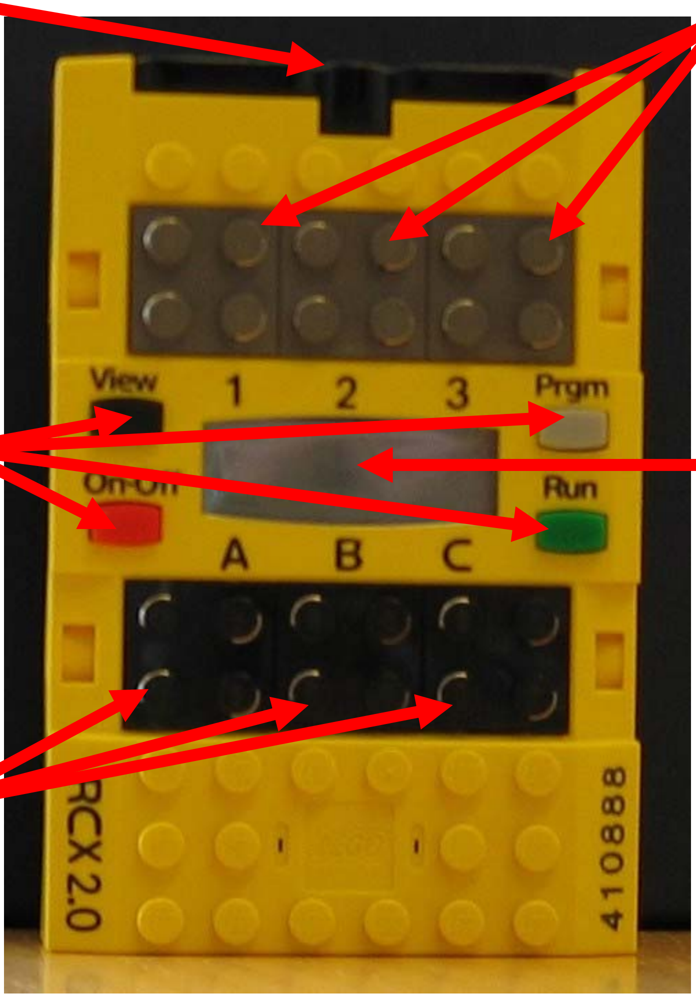
RCX 2.0 Inputs and Outputs

IR Port

used to communicate with IR tower (this provides a connection to a PC for things like software uploads)

Input Ports

used to connect sensors



Push Buttons

used for direct human interaction

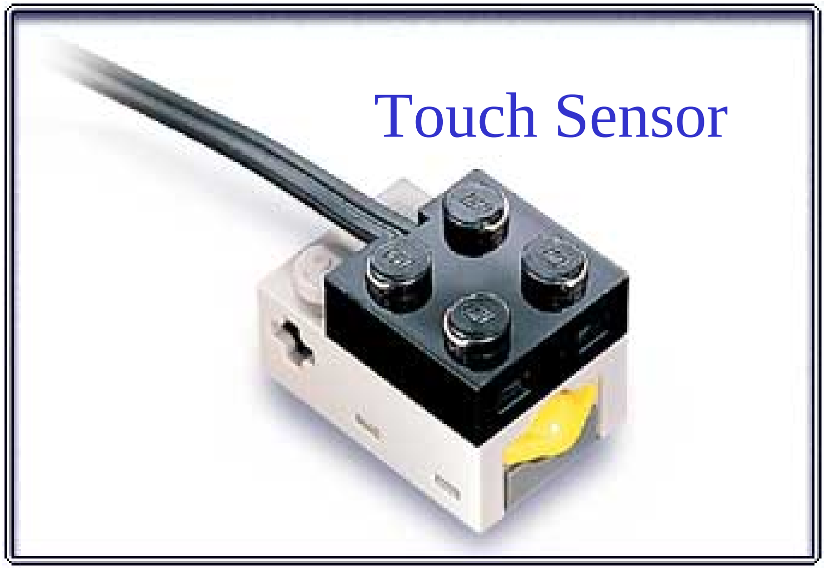
Display

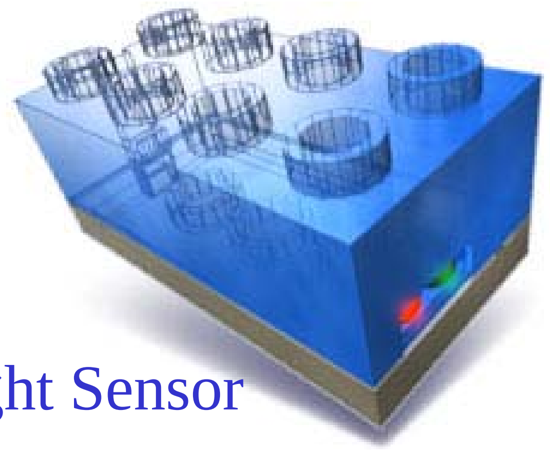
used to display small amounts of data for human interaction

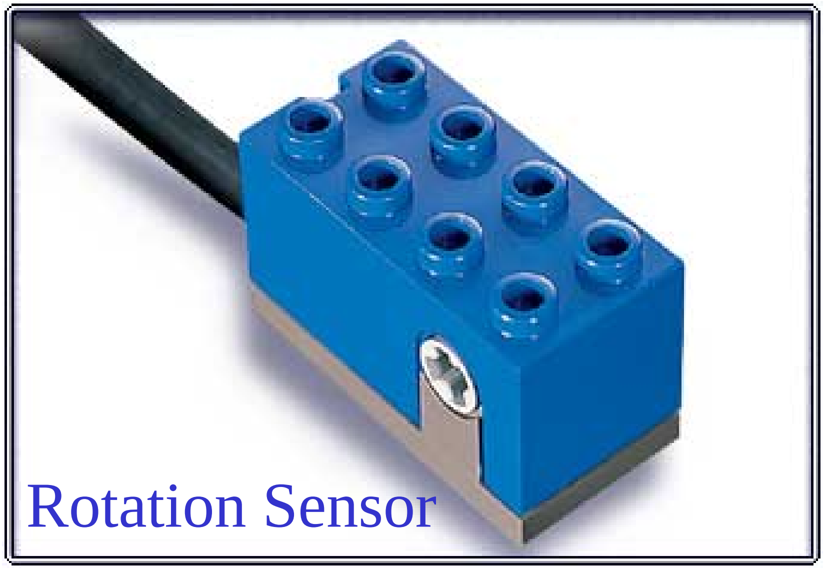
Output Ports

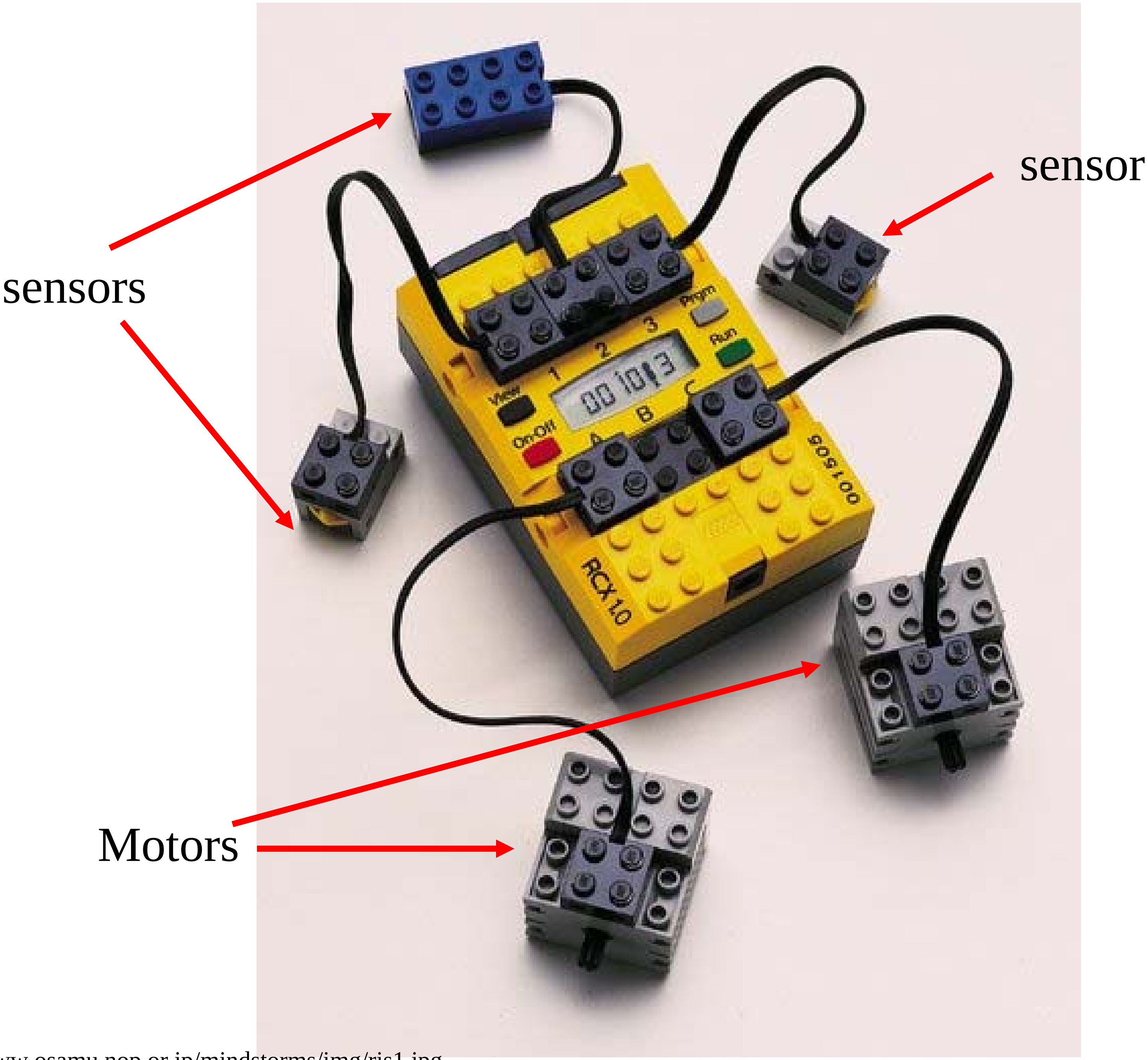
used to connect motors

RCX Sensors


Touch Sensor


Light Sensor


Rotation Sensor



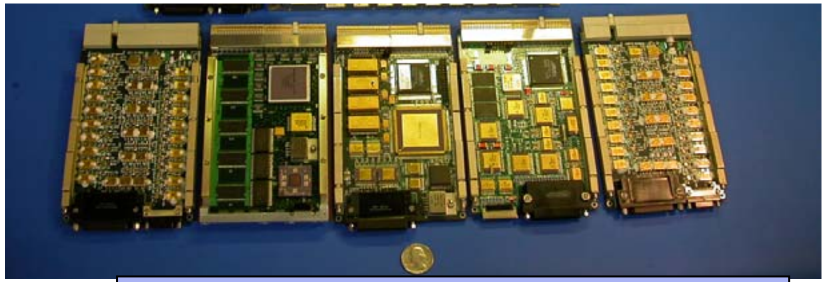
http://www.osamu.nop.or.jp/mindstorms/img/ris1.jpg

Today

- Lego Mindstorms Logistics
- RCX 2.0 Hardware Architecture
- RCX 2.0 Software Architecture & Basics of Programming with Legos
- Programming Example

Recall from Lecture 1…

Sample Aerospace Computing System Components



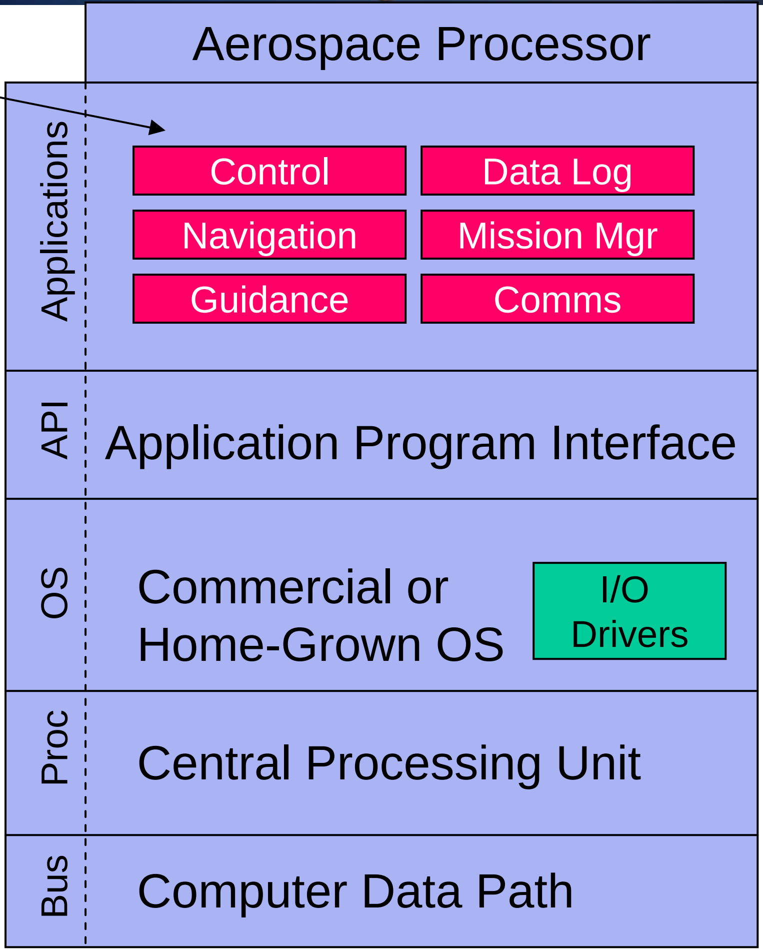
Programming used to create applications

Applications

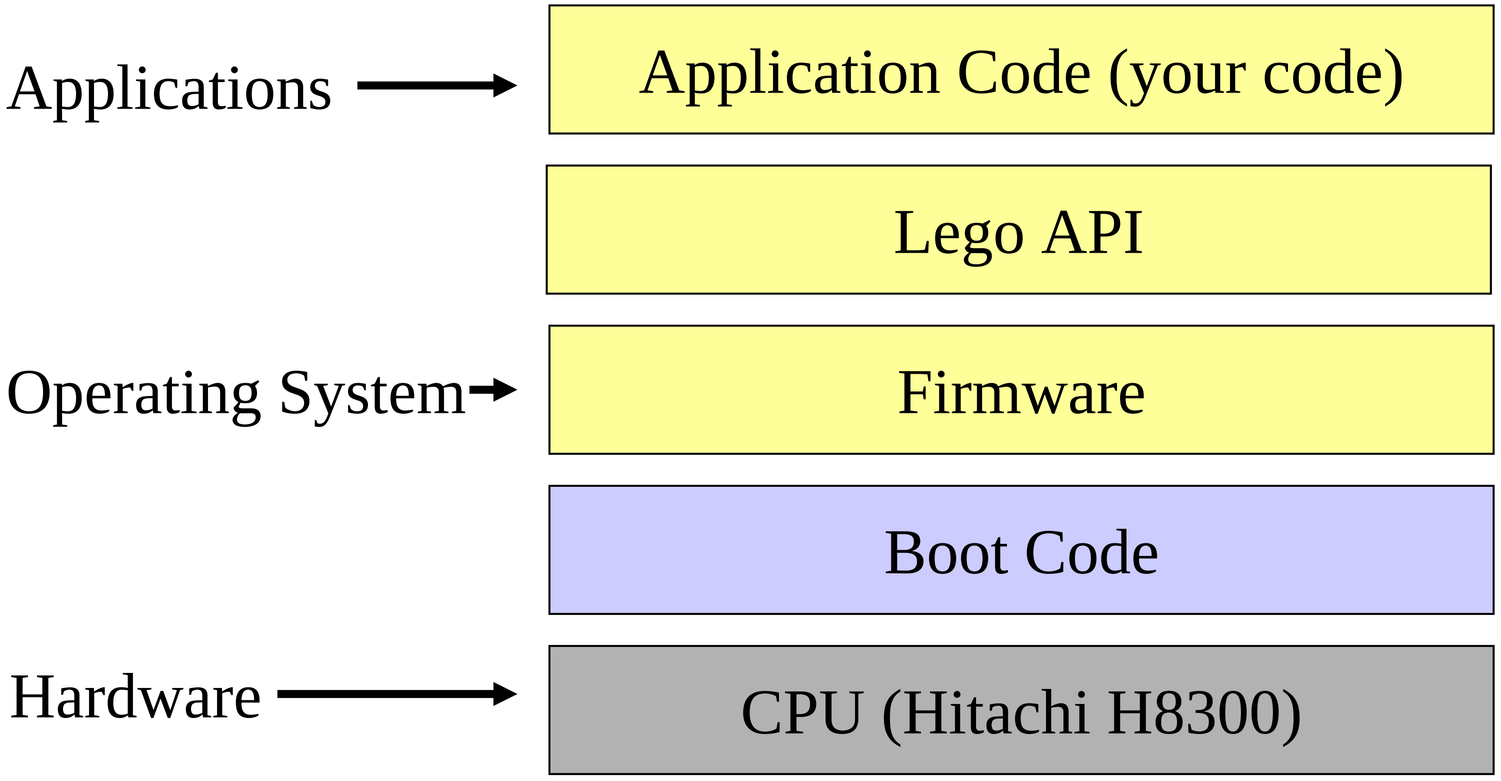
Operating System

Hardware

Information Flow (Data)



What does this mean in the lego world?



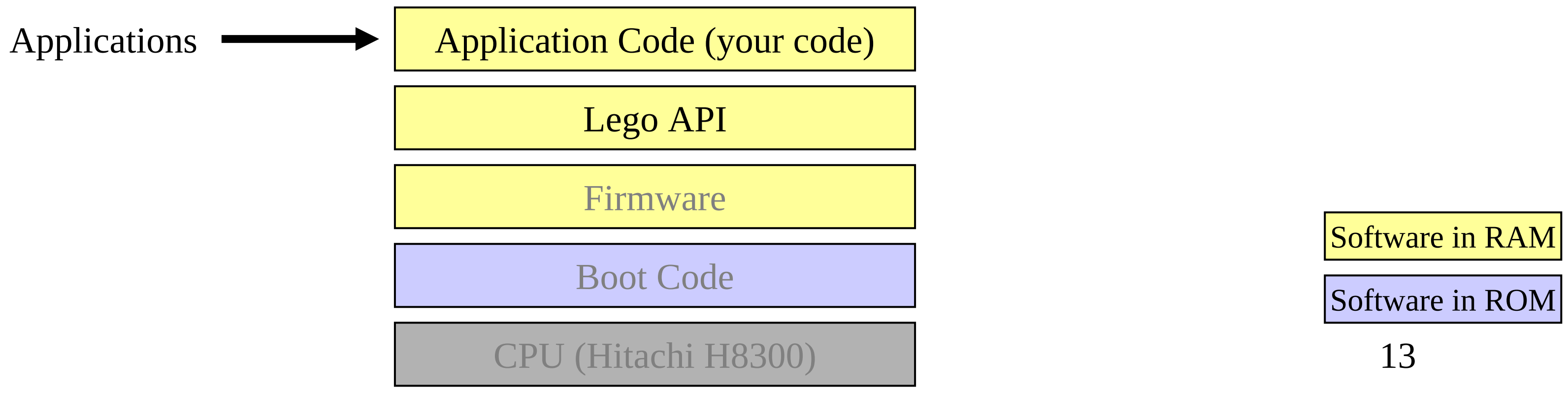
Read Only Memory (ROM), contains software and data that is permanently written to the ROM memory chip.

Random Access Memory (RAM), contains software and data that is temporarily written to the RAM memory chip. RAM memory chips requires a continuous power supply to data.

🟨	Software in RAM
🟪	Software in ROM

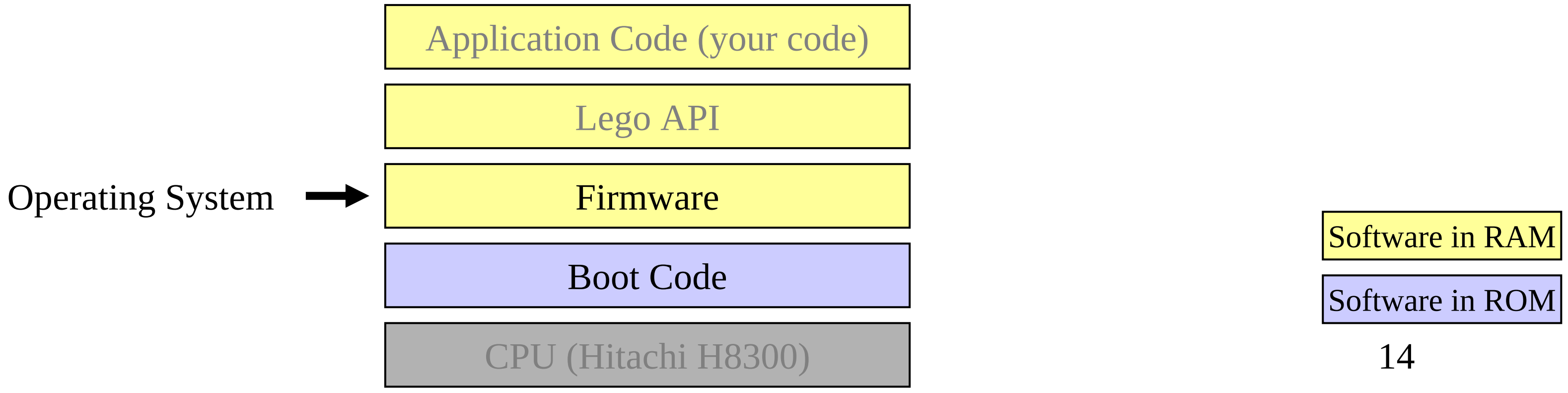
What is an Application? What is an API?

- Application Code is the code that draws on the levels below it to solves the specific problem that the system is addressing.
  - This is the code that changes the most from system to system.
- An API, or Application Program Interface, provides a generic set of capabilities that help the application to interface with the hardware. An Interface is a layer between two systems that defines the interaction between those systems. It is useful because it allows one system to use the other without a complete understanding of how the second system works.



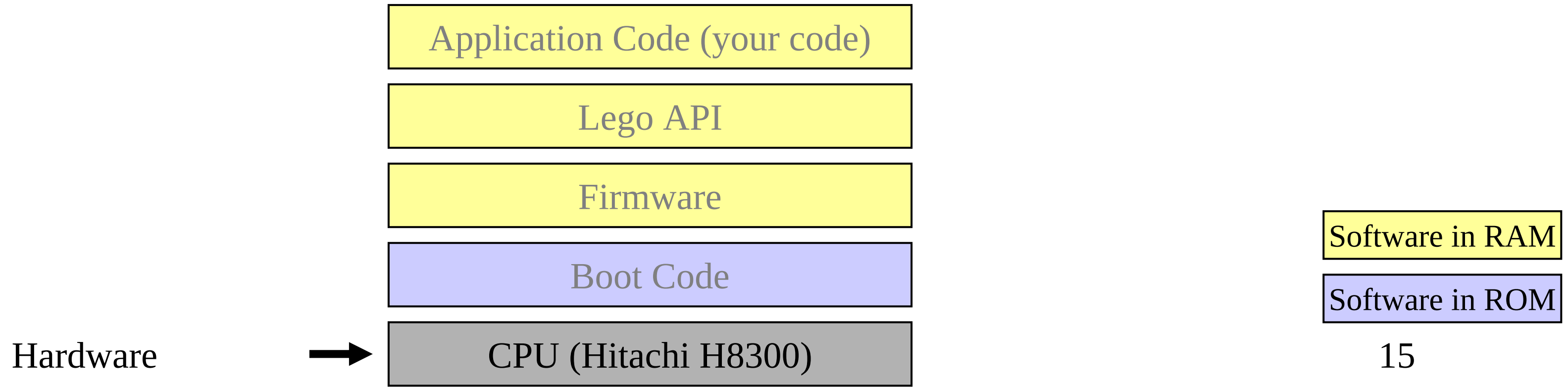
What is Firmware? What is Boot Code?

- Much like an API, an operating system provides basic system functions that allows the application code to interface with the underlying hardware. For the RCX system these functions are provided by the Firmware and Boot Code.
- For the RCX system, the operating system functions are provided by the Firmware and the Boot Code.
- The Boot Code:
  - Provides the code that is first executed when the RCX is powered on (this allows software to be loaded onto the RCX)
  - Provides Input, output, and display signal conversion
  - Provides IR port communication
- The Firmware:
  - Assists in executing the application
  - Is the Lego API's interface to the boot code i/o functions
  - Is called "firmware" because it is software that is normally not altered



What is a CPU?

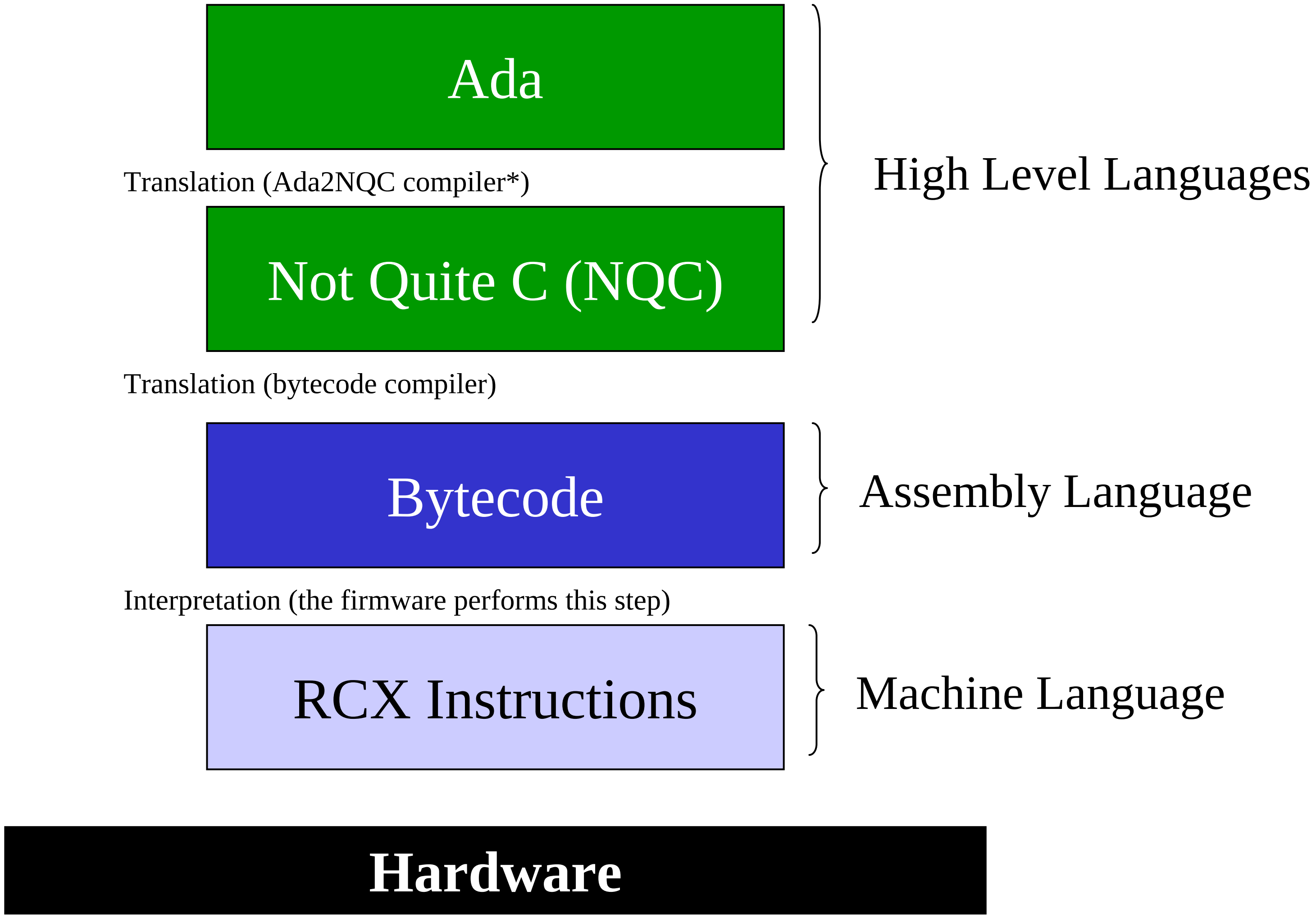
- A Central Processing Unit (CPU) is a hardware devices that can execute a simple mathematical instruction set. It is the brains of a computer.
- Ultimately all code that is written has to be converted into mathematical instructions that the processor can execute.



How does the conversion to machine instructions happen? Why bother?

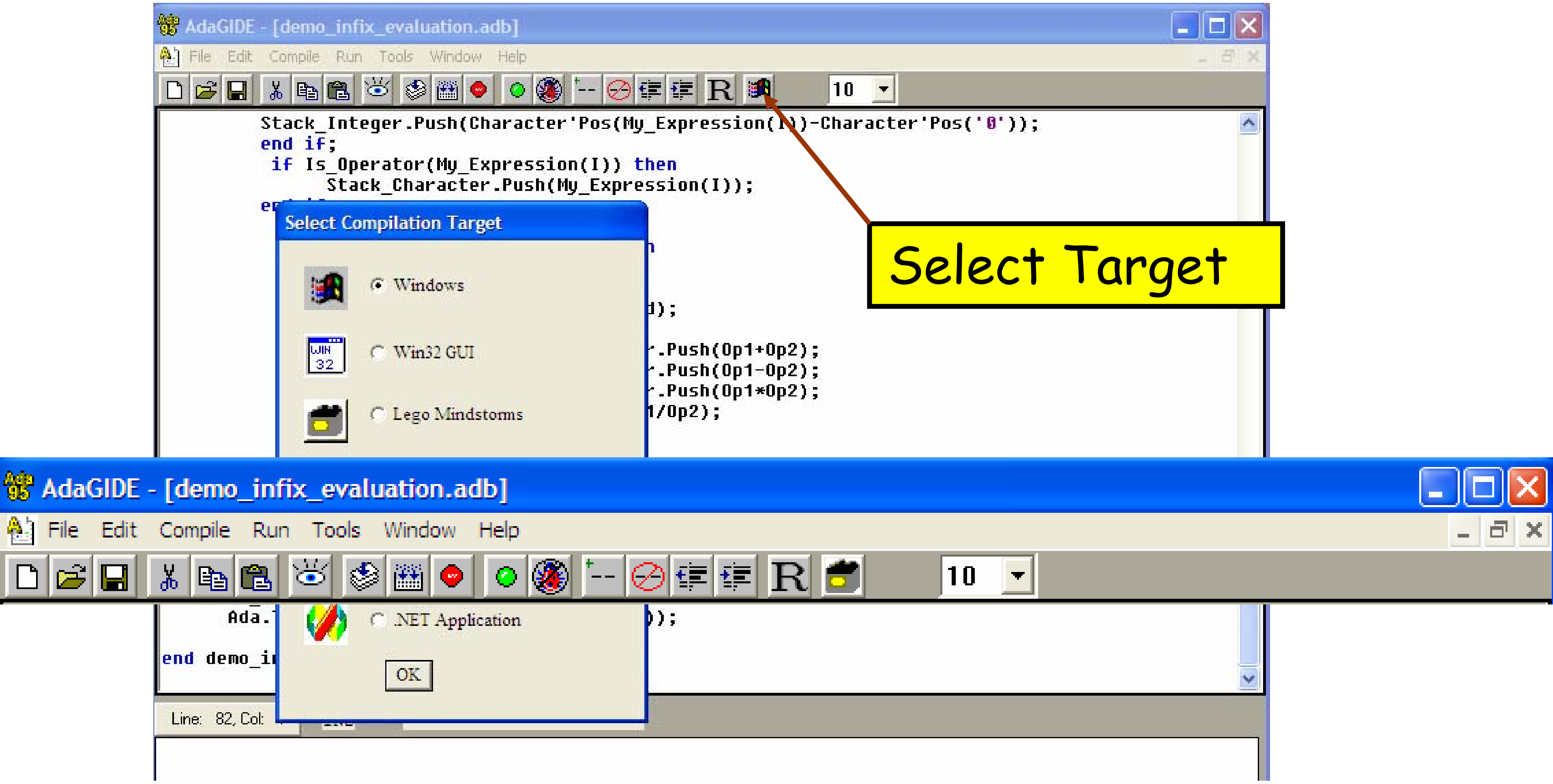
- Machine instructions are very difficult to write code in because the set of instructions is very limited -- it takes a lot of code to do very basic things.
- Code can be *translated* from one language to another. A computer language is translated into another language using a compiler. The compiler reads in a file containing a program in one language and produces a file containing a program in another language.
- Code can also be *interpreted* from one language to another. A computer language is interpreted into another language using an interpreter. The interpreter reads in a file containing a program in one language and figures out what machine instructions are needed to execute each statement, one at a time, and asks the CPU to execute those statements.

Ada to RCX Machine Language


Ada
Translation (Ada2NQC compiler*)
Not Quite C (NQC)
High Level Languages
Translation (bytecode compiler)
Bytecode
Assembly Language
Interpretation (the firmware performs this step)
RCX Instructions
Machine Language
Hardware

*The Ada2NQC compiler supports only a subset of the Ada language. Details are in the Ada Mindstorms manual

Using AdaGIDE to Build Lego Programs


AdaGIDE - [demo_infix_evaluation.adb]
File Edit Compile Run Tools Window Help
10
Select Compilation Target
Windows
Win32 GUI
Lego Mindstorms
.NET Application
OK
Line: 82, Col:
Select Target

```
        Stack_Integer.Push(Character'Pos(My_Expression(I))-Character'Pos('0'));
        end if;
         if Is_Operator(My_Expression(I)) then
              Stack_Character.Push(My_Expression(I));
```

Select Compilation Target: [✓ Windows] Win32 GUI Lego Mindstorms .NET Application

http://www.usafa.af.mil/dfcs/Ada_Mindstorms_manual.htm

Question

Have you used Lego Mindstorms before?

1. *No* previous experience
2. I have *some* experience using Lego Mindstorms
3. I feel *confident* in building Lego Mindstorms robots and/or programming them

Details of the Lego API

- Much like the Text_IO APIs, the Lego API defines functions to help us interface with inputs and outputs.
  - RCX Input Ports (sensors)
  - RCX Output Ports (motors)
  - Other RCX capabilities
- Lego API defines its own special types as part of this API.
- To use the Lego API use the Ada keywords **with** and **use**.

Using the Sensors

- In order to use a sensor port, you need to first configure it.
- Configuring the port tells the software that the sensor is there and how it should be used.
- Configuration includes:
  - The port the sensor is connected to
  - The type of sensor it is (light, touch, etc)
  - How the output data from the sensor should be formatted

Data Types Used for Sensor Ports

```
type Sensor_Port is (Sensor_1, Sensor_2, Sensor_3);  -- the 3 input connections

type Sensor_Type is
   (Type_Touch,      -- a touch sensor
    Type_Temperature,-- a temperature sensor
    Type_Light,       -- a light sensor
    Type_Rotation);   -- a rotation sensor

type Sensor_Mode is (
      Mode_Raw,   -- number from 0 to 1023
      Mode_Bool,  -- 0 or 1
      Mode_Edge,  -- counts number of Boolean transitions
      Mode_Pulse,-- counts number of Boolean periods
      Mode_Percent,       -- number from 1 to 100
      Mode_Celsius,       -- degrees C
      Mode_Fahrenheit,    -- degrees F
      Mode_Rotation);     -- counts rotations
```

Data Types Used for Sensor Ports

```
type Configuration is (
   Config_Touch,              -- type = type_touch, mode = mode_bool
   Config_Pulse,              -- type = type_touch, mode = mode_pulse
   Config_Edge,               -- type = type_touch, mode = mode_edge
   Config_Light,              -- type = type_light, mode = mode_percent
   Config_Rotation,           -- type = type_rotation, mode = mode_rotation
   Config_Celsius,            -- type = type_temperature, mode = mode_celsius
   Config_Fahrenheit);        -- type = type_temperature, mode = mode_fahrenheit
```

- The Configuration type provides common configurations for sensors.
- Use the following API call to quickly configuration a sensor (other API calls can be used if a different sensor configuration is desired):

```
procedure Config_Sensor
   (Sensor : in Sensor_Port; --const
    Config : in Configuration  );
```

- Example sensor configuration call:

```
Config_Sensor(Sensor => Sensor_1,
              Configuration => Config_Touch);
```

Data Types Used for Output Ports

```
type Output_Port is (Output_A, Output_B, Output_C); -- the 3 output connections

type Output_Mode_Type is (Output_Mode_On, Output_Mode_Off);

type Output_Direction is (Output_Direction_Forward,
                          Output_Direction_Reverse,
                          Output_Direction_Toggle); -- changes port direction

type Power_Type is range 0..7;
   Power_Low   :  constant Power_Type := 1;
   Power_Half  :  constant Power_Type := 4;
   Power_High  :  constant Power_Type := 7;
```

- Procedure calls provide shortcuts for many of the output port interactions. For example, to turn on output A use:

```
      Output_On(Output => Output_A);
```

Today

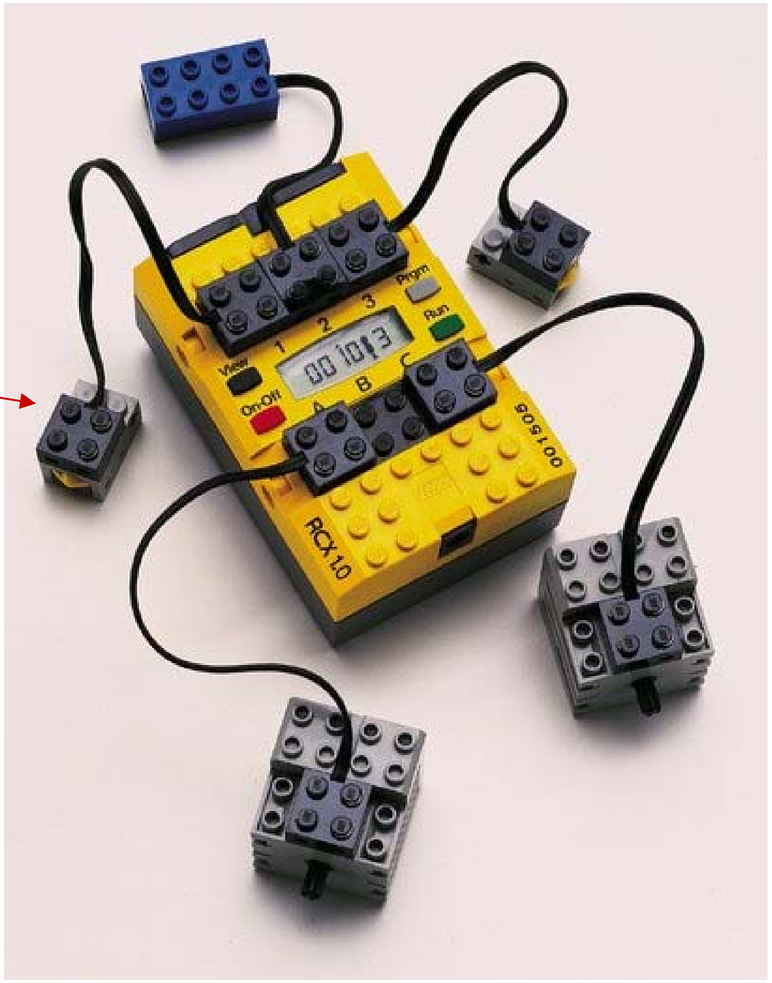
- Lego Mindstorms Logistics
- RCX 2.0 Hardware Architecture
- RCX 2.0 Software Architecture & Basics of Programming with Legos
- Programming Example

Rover Problem Specification

- The Angry “Parked-In” Rover
  - The rover drives forward until it hits something
  - Then, drives backward until it hits something
  - Repeat this forever even though playing “bumper cars” will never get it out of the parking space

Parked-in Rover

- Data Requirements & Formulas
  - Problem Inputs:
    - Touch sensor values
  - Problem Outputs:
    - Power to drive motors forward
    - Power to drive motors in reverse



Parked-in Rover

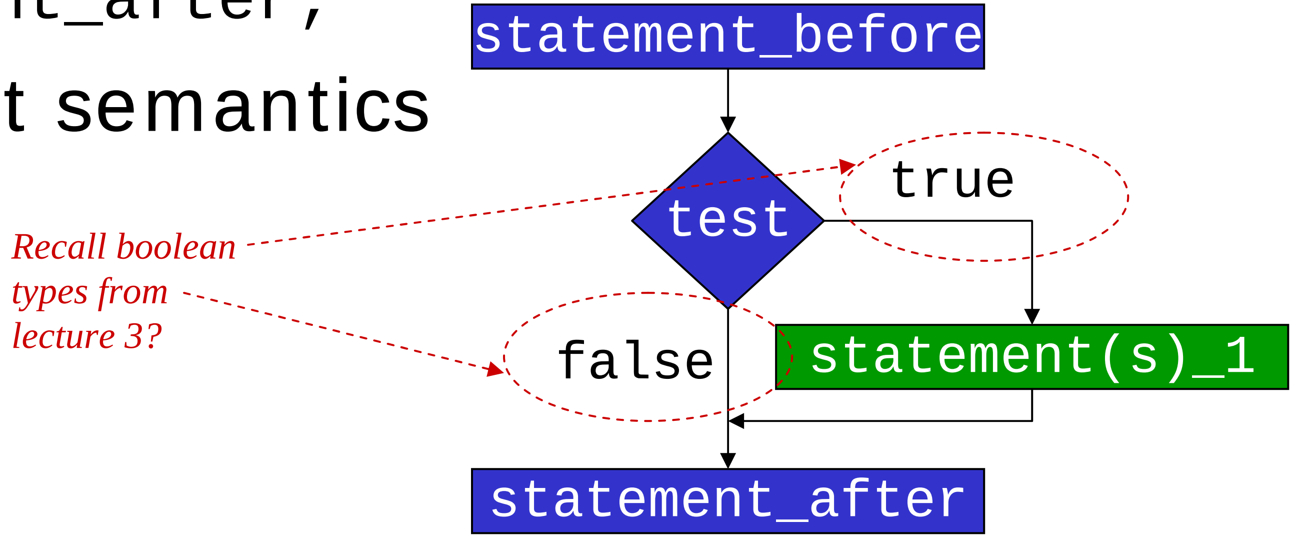
- Data Requirements & Formulas
  - Relevant Formula / Preconditions
    - Front touch sensor is on sensor_1 input
    - Rear touch sensor is on sensor_3 input
    - Drive motor receives high_power
    - Drive motor is on Output_A
  - Postconditions
    - Rover moves forward and backward forever
      (or until the batteries wear down!!)

if-then Statements

- Statement form
  - statement_before;

```
if test then
  statement(s)_1;
end if;
statement_after;
```

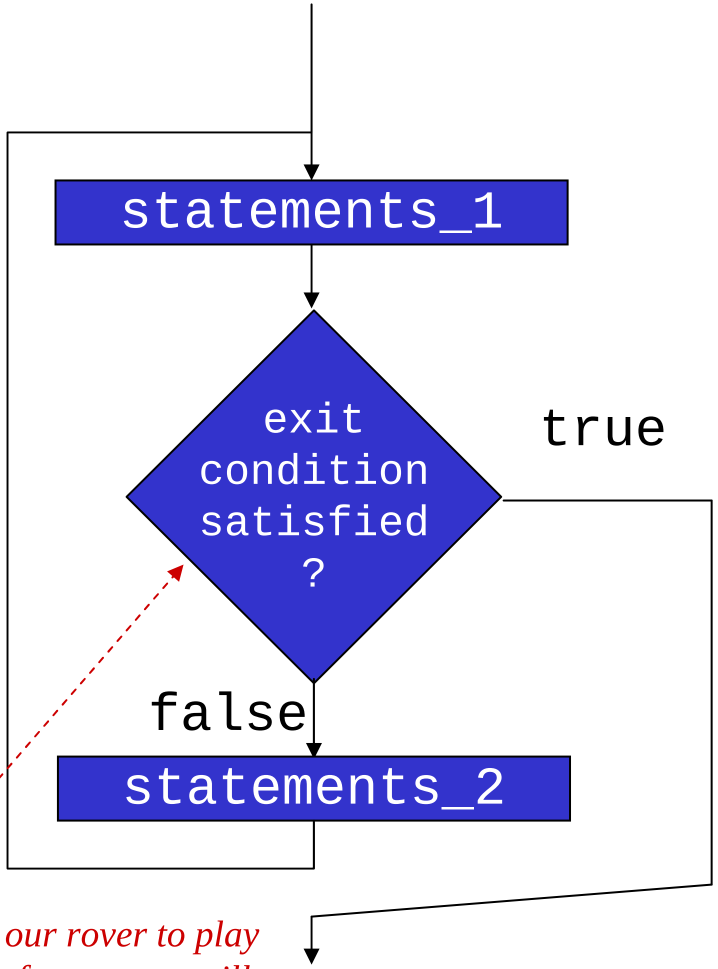
- Statement semantics



MORE ON THIS IN LECTURE 5

General Loop Statement

```
loop
   statements_1;
   exit when test;
   statements_2;
end loop;
```



We need this Ada construct to drive forward and backward forever

In order for our rover to play bumper cars forever, we will not put an exit condition in our rover code loop

MORE ON THIS IN LECTURE 5

Lego Mindstorms Parked-In Rover

```
-----------------------------------------------------------------------------
-- Lego Mindstorms Demo 1 -- Touch Sensors
-- a.k.a. "Parked-In"
-- David Johnson, C. S. Draper Laboratory
-- For MIT Unified
--
-- October 6, 2005
--
-- Goal:  The car drives forward until it hits something, and then goes in
--        reverse until it hits... Just like what happens when you get parked-
--        in on some side-street in downtown Boston.
-----------------------------------------------------------------------------

with Lego;
use Lego;

procedure Demo1_Ada_Touchsensors is

   -- Define how we will address each motor and sensor
   Drive_Motor : constant Output_Port := Output_A;
   Front_Touch : constant Sensor_Port := Sensor_1;
   Rear_Touch  : constant Sensor_Port := Sensor_3;

begin
   -- Initialize the Wheels
   Output_Power(
      Output => Drive_Motor,
      Power  => Power_High);

   -- Initialize the Touch Sensors
   Config_Sensor(
      Sensor => Front_Touch,
      Config => Config_Touch);

   Config_Sensor(
      Sensor => Rear_Touch,
      Config => Config_Touch);
```

Lego Mindstorms Parked-In Rover (page 2)

```
   -- Start by Driving Forward
   Output_On_Forward(Output => Drive_Motor);

   -- It never ends ...
   loop

      -- Start backing up if the car hits something
      if Get_Sensor_Value(Sensor => Front_Touch) = 1 then
         Output_Reverse(Output => Drive_Motor);
      end if;

      -- Make the escape if we hit something behind us
      if Get_Sensor_Value(Sensor => Rear_Touch) = 1 then
         Output_Forward(Output => Drive_Motor);
      end if;

   end loop;

end Demo1_Ada_Touchsensors;
```

Lego Project Work

- Teams of Unified Students have been identified (for p-set and system problem work with legos)
- Pick up your lego kit in Unified office (available in office hours this afternoon)
  - Lego kits are numbered by team
- You can bring a laptop computer to recitation on Thursday if you'd like help in setting up the development environment and/or practice with the robots