Introduction: Hello, World Below

The course at a glance

Objectives:

- Understand how hardware and software systems are built, and how they work together
- Learn how to break complex problems into simpler ones
- Learn how large scale development projects are planned and executed
- Have fun.

Methodology:

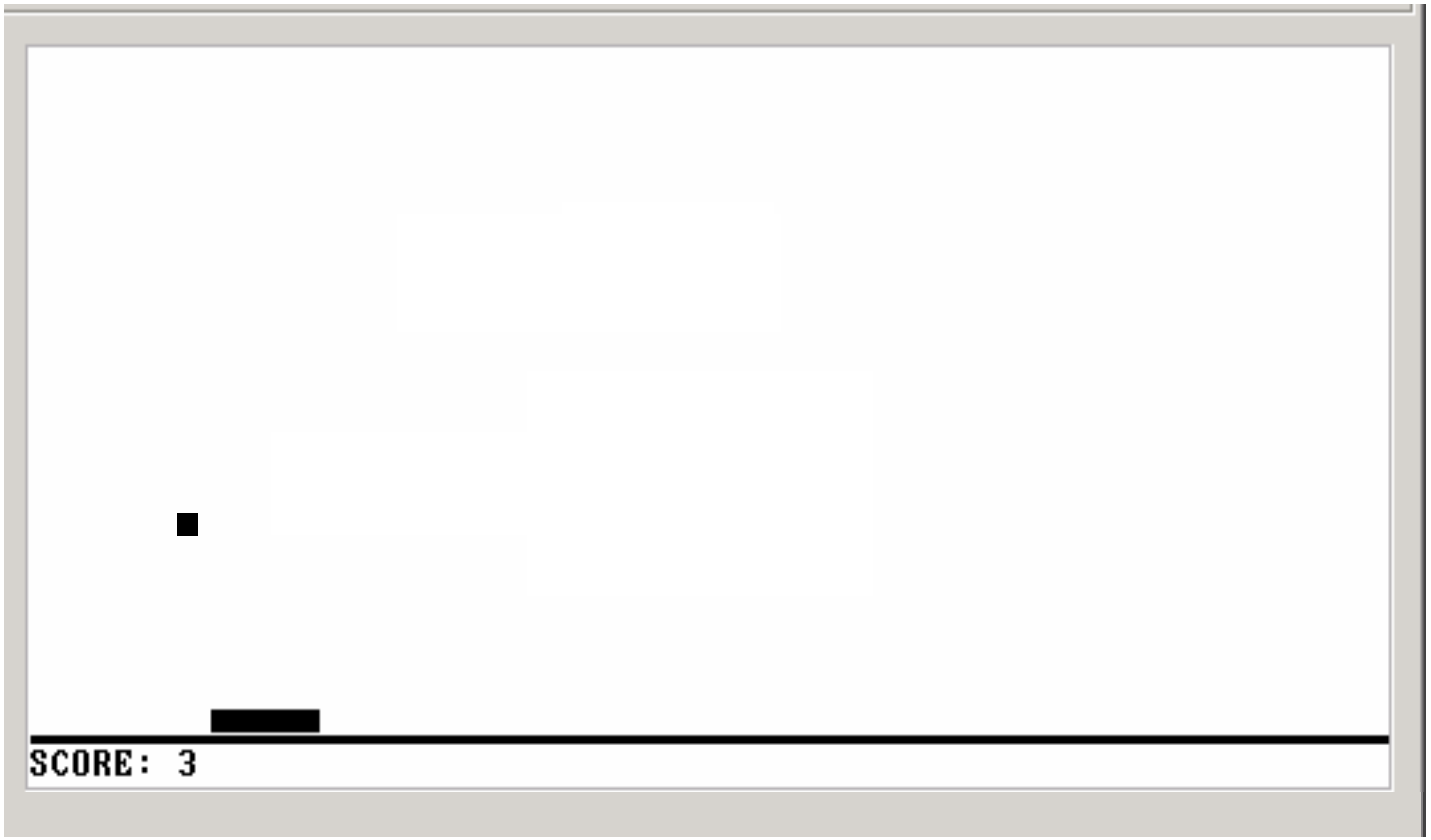
- Build a complete, general-purpose, and working computer system
- Play and experiment with this computer, at any level of interest.

Some course details

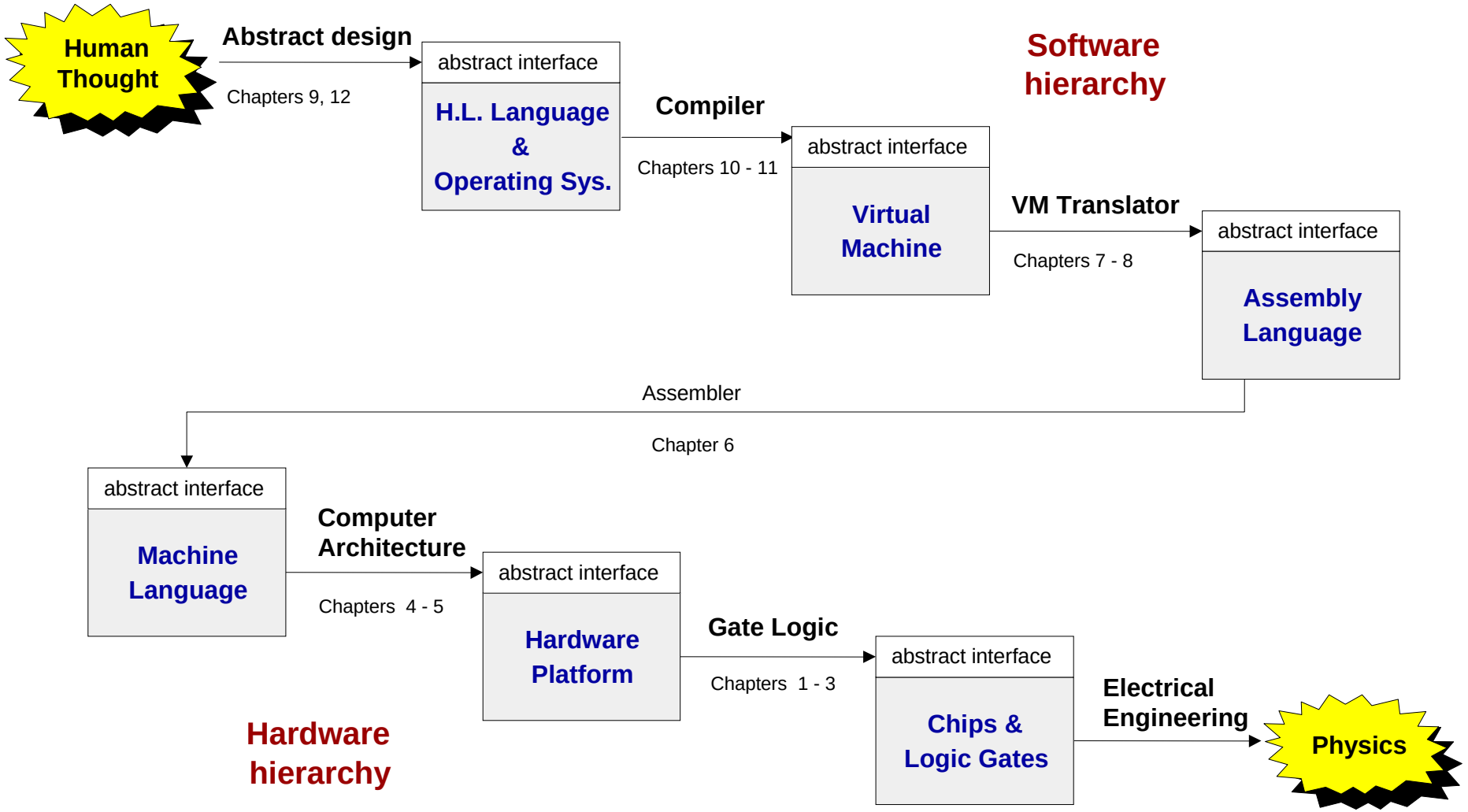
- 12 projects, can be done by pairs
- Hardware projects are done and simulated in HDL (Hardware Description Language)
- Software projects can be done in any programming language of your choice (we recommend Java)
- Projects philospohy: design (API) + all test materials are given, implementation done by students
- Tools (simulators, tutorials, test scripts)
- Book
- Q&A policy
- Exam.

Demo



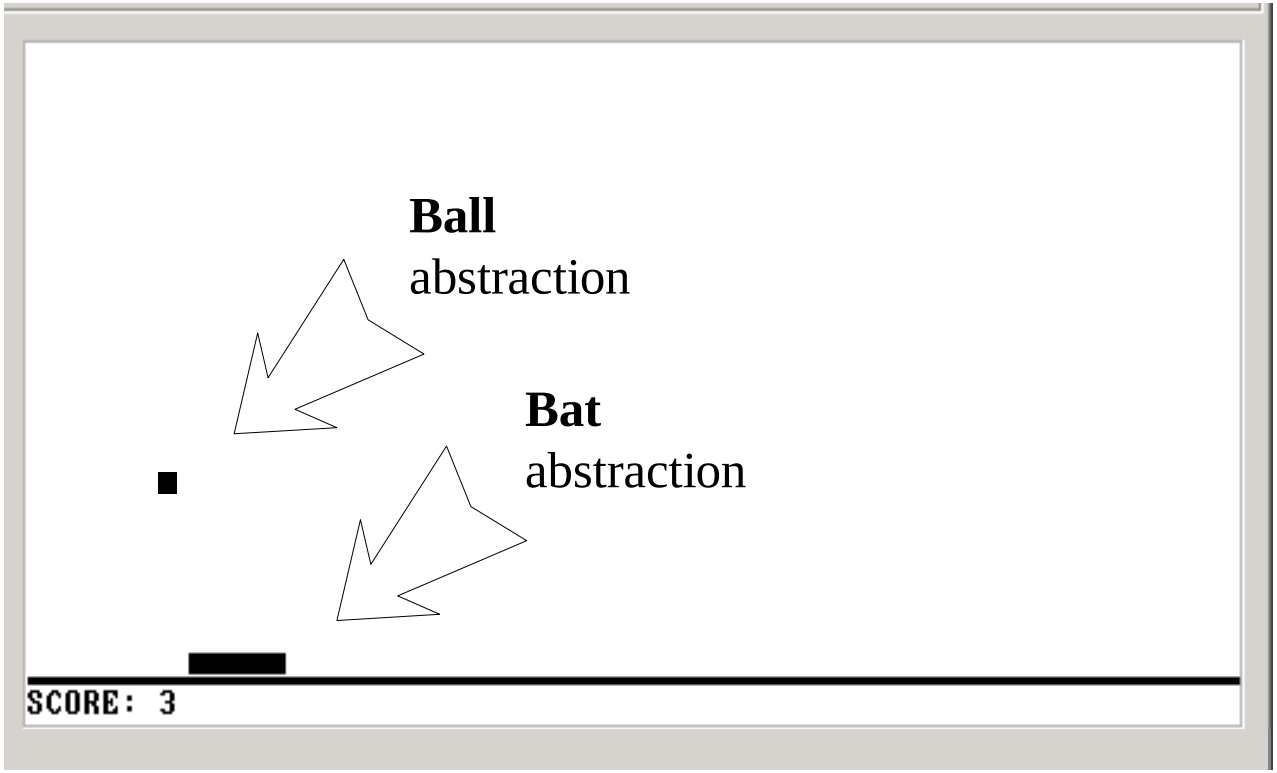


Course theme and structure

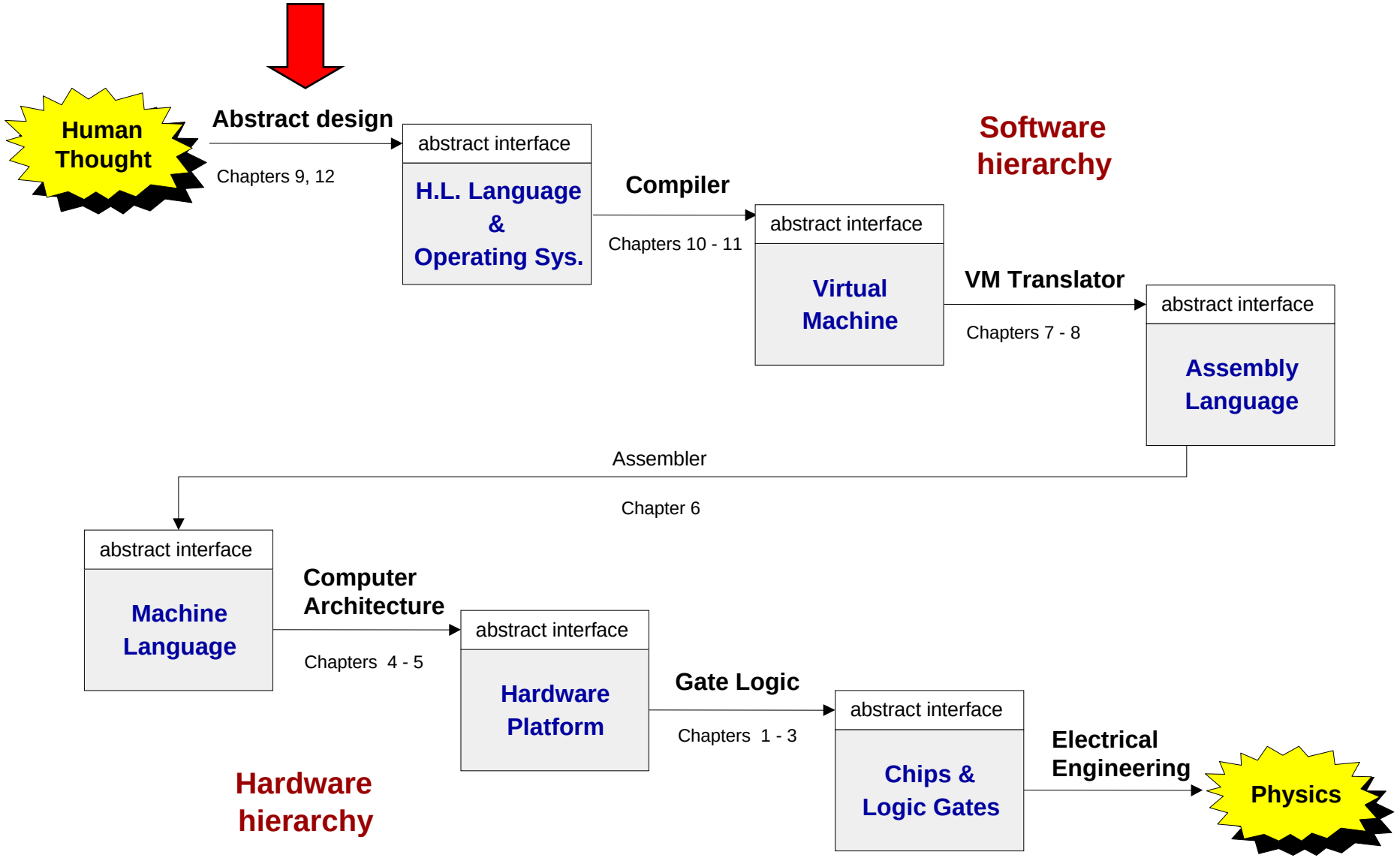


(Abstraction–implementation paradigm)

Application level: Pong

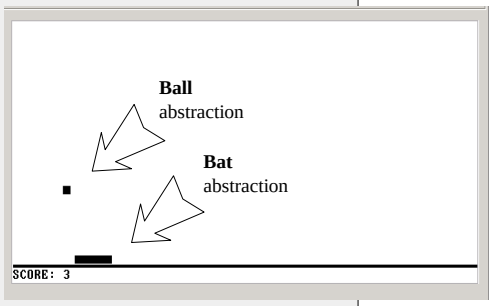


The big picture



High-level programming (Jack language)

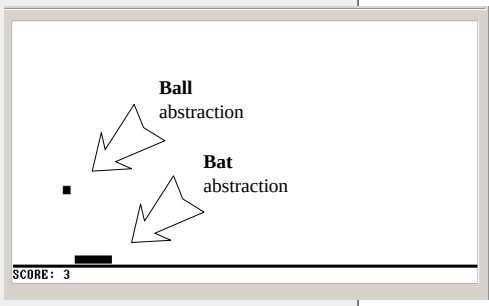
```
/** A Graphic Bat for a Pong Game */
class Bat {
    field int x, y;           // screen location of the bat's top-left corner
    field int width, height;  // bat's width & height

    // The class constructor and most of the class methods are omitted

    /** Draws (color=true) or erases (color=false) the bat */
    method void draw(boolean color) {
       do Screen.setColor(color);
       do Screen.drawRectangle(x,y,x+width,y+height);
       return;
    }

    /** Moves the bat one step (4 pixels) to the right. */
    method void moveR() {
       do draw(false);  // erase the bat at the current location
       let x = x + 4;   // change the bat's X-location
       // but don't go beyond the screen's right border
        if ((x + width) > 511) {
           let x = 511 - width;
       }
       do draw(true);  // re-draw the bat in the new location
       return;
    }
}
```



High-level programming (Jack language)

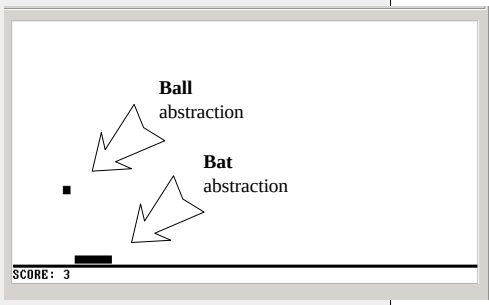
```
/** A Graphic Bat for a Pong Game */
class Bat {
    field int x, y;           // screen location of the bat's top-left corner
    field int width, height;  // bat's width & height

    // The class constructor and most of the class methods are omitted

    /** Draws (color=true) or erases (color=false) the bat */
    method void draw(boolean color) {
       do Screen.setColor(color);
       do Screen.drawRectangle(x,y,x+width,y+height);
       return;
    }

    /** Moves the bat one step (4 pixels) to the right. */
    method void moveR() {
       do draw(false);  // erase the bat at the current location
       let x = x + 4;   // change the bat's X-location
       // but don't go beyond the screen's right border
        if ((x + width) > 511) {
           let x = 511 - width;
       }
       do draw(true);  // re-draw the bat in the new location
       return;
    }
}
```

A typical call to an operating system method

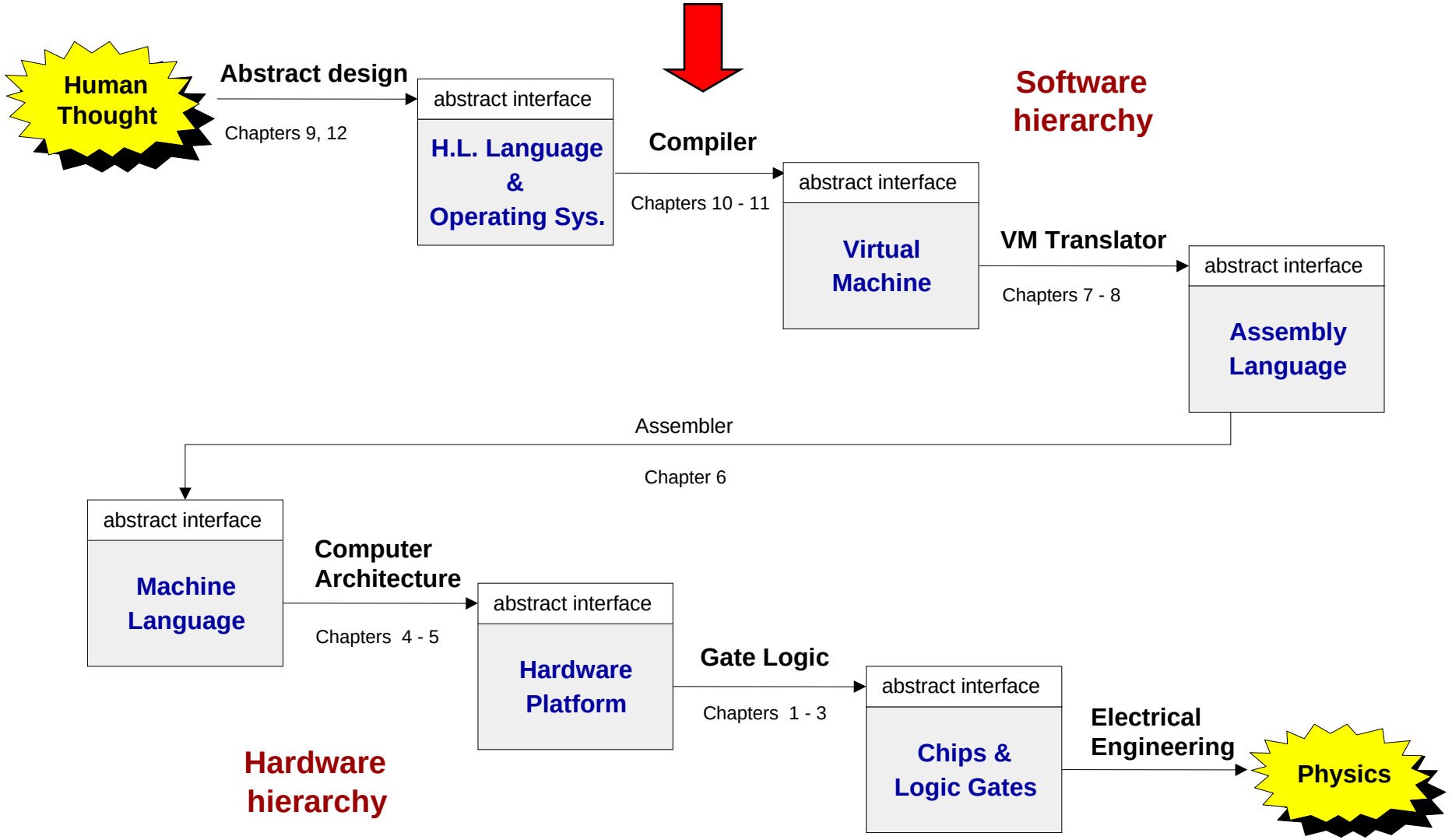


Operating system level (Jack OS)

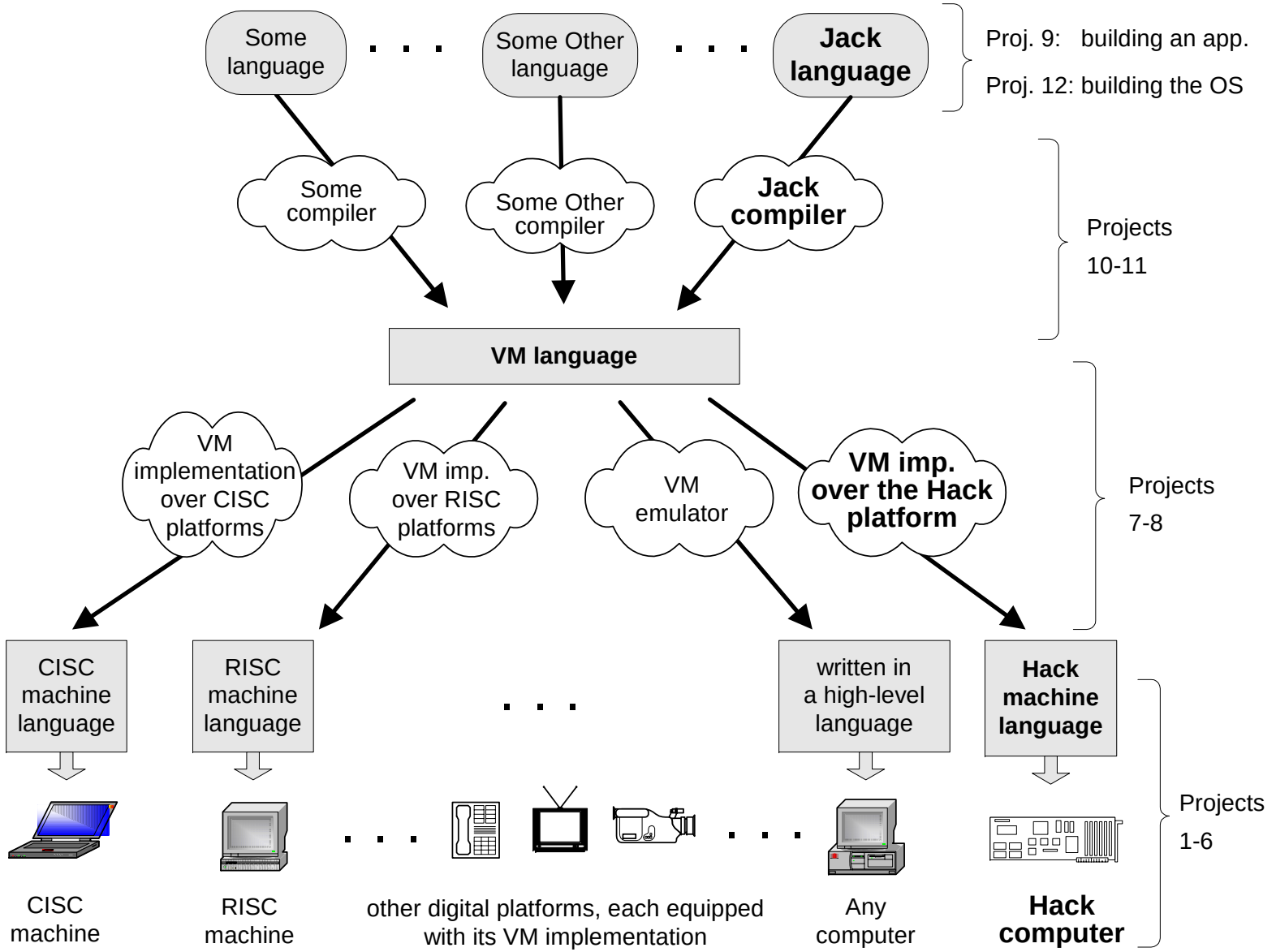
```
/** An OS-level screen driver that abstracts the computer's physical screen */
class Screen {
     static boolean currentColor;  // the current color

     // The Screen class is a collection of methods, each implementing one
     // abstract screen-oriented operation.  Most of this code is omitted.

     /** Draws a rectangle in the current color. */
     // the rectangle's top left corner is anchored at screen location (x0,y0)
     // and its width and length are x1 and y1, respectively.
     function void drawRectangle(int x0, int y0, int x1, int y1) {
         var int x, y;
         let x = x0;
         while (x < x1) {
             let y = y0;
             while(y < y1) {
                do Screen.drawPixel(x,y);
                let y = y+1;
             }
             let x = x+1;
         }
     }
}
```



The big picture



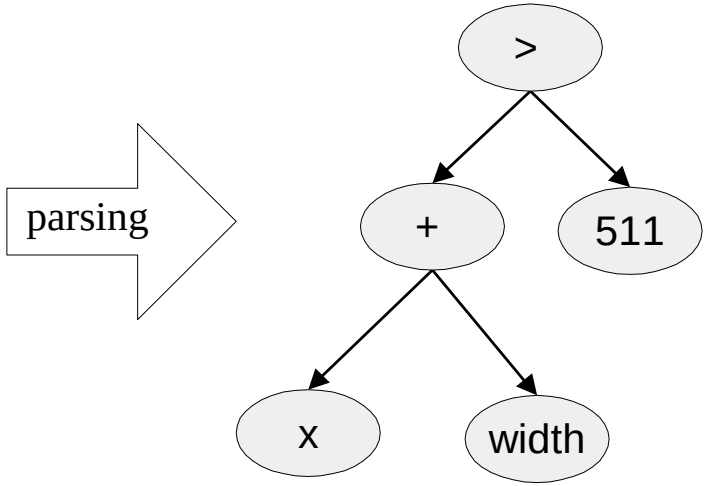
A modern compilation model



Compilation 101: syntax analysis

Source code

(x+width)>511

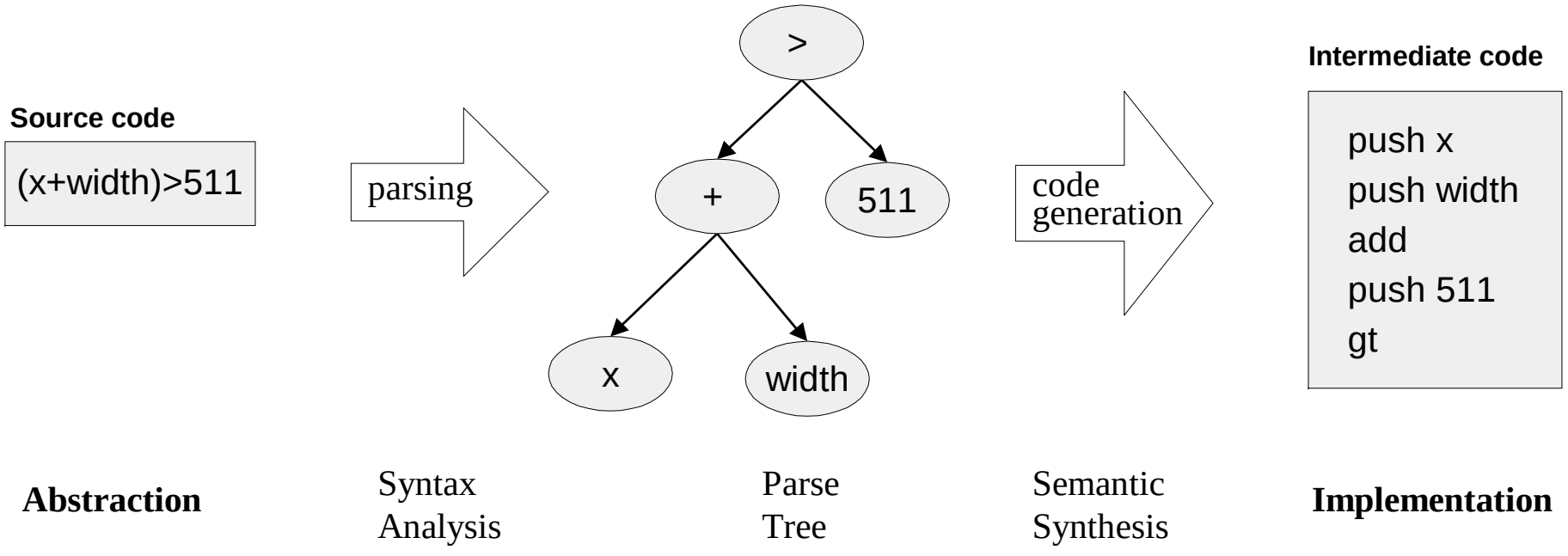


Abstraction

Syntax Analysis

Parse Tree

Compilation 101: code generation



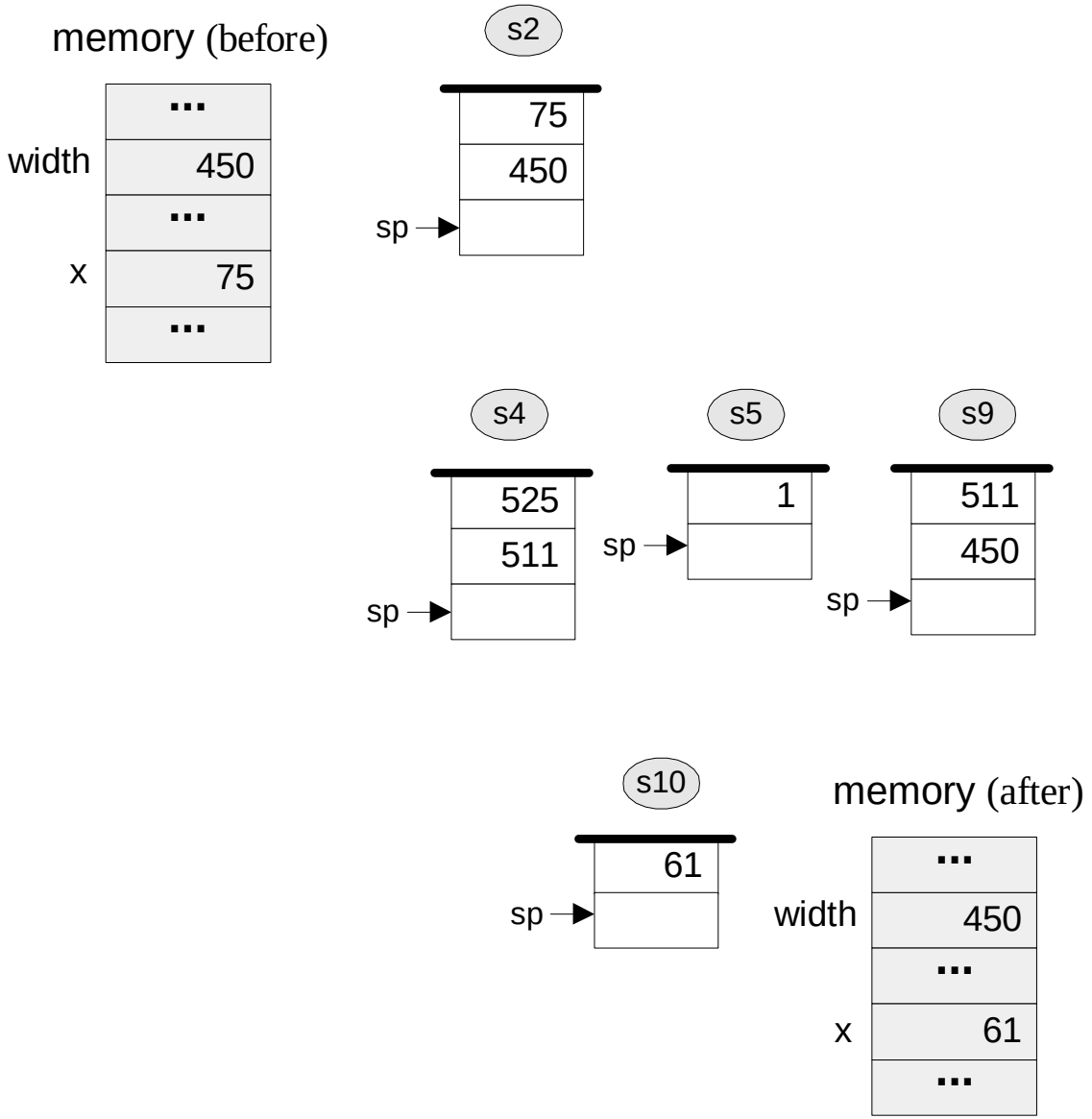
Observations:

- Modularity
- Abstraction / implementation interplay
- The implementation uses abstract services from the level below.

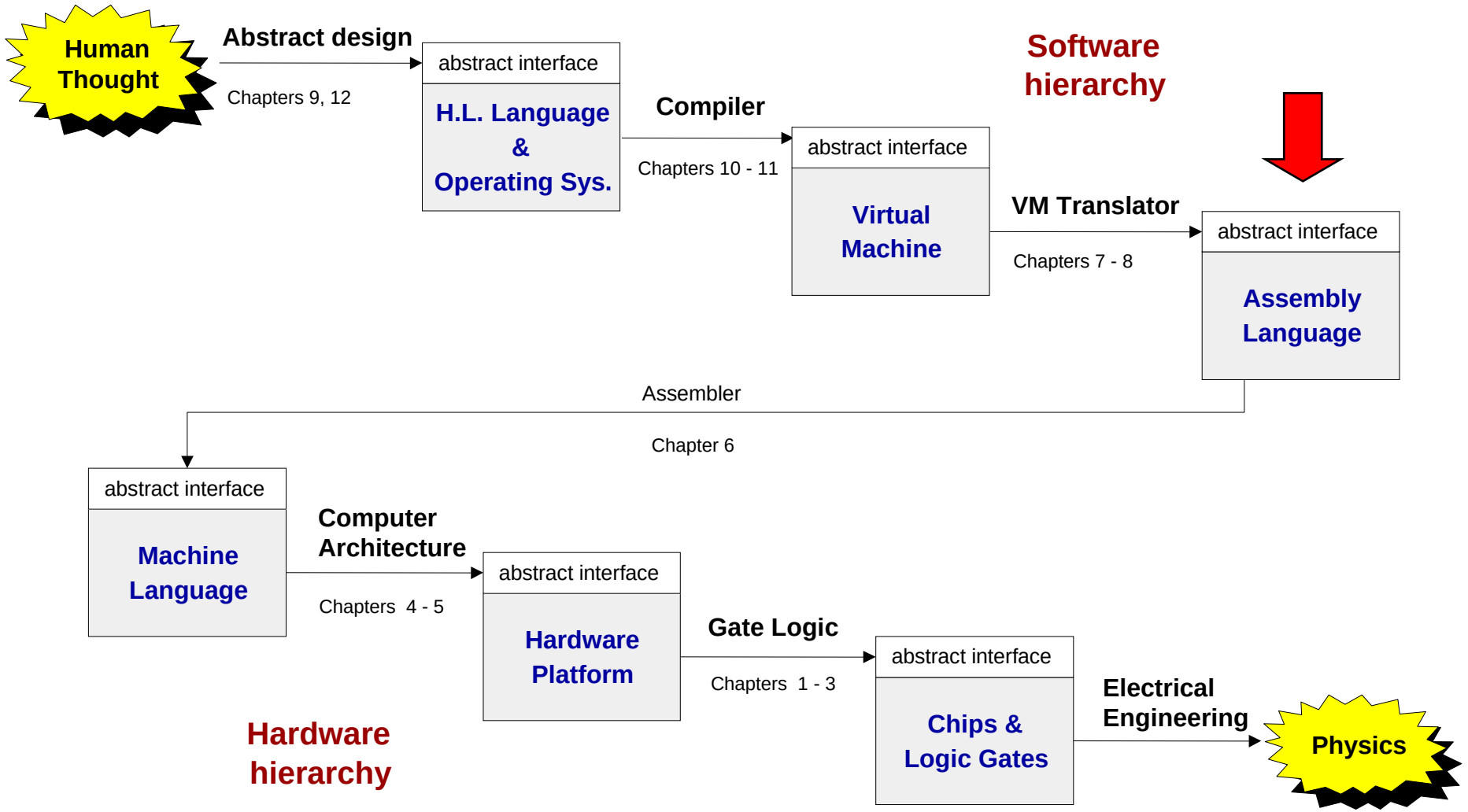
The Virtual Machine (our stack-based VM, which is quite similar to the JVM)

```
if ((x+width)>511) {
   let x=511-width;
}
```

```
// VM implementation
   push x        // s1
   push width    // s2
   add           // s3
   push 511      // s4
   gt            // s5
   if-goto L1    // s6
   goto L2       // s7
L1:
   push 511      // s8
   push width    // s9
   sub           // s10
   pop x         // s11
L2:
...
```



The big picture



Low-level programming (on the Hack computer)

For now, ignore all details!

Virtual machine program

```
...
   push x
   push width
   add
   push 511
   gt
   if-goto L1
   goto L2
L1:
   push 511
   push width
   sub
   pop x
L2:
...
```

Low-level programming (on the Hack computer)

For now, ignore all details!

Virtual machine program

```
...
   push x
   push width
   add
   push 511
   gt
   if-goto L1
   goto L2
L1:
   push 511
   push width
   sub
   pop x
L2:
...
```

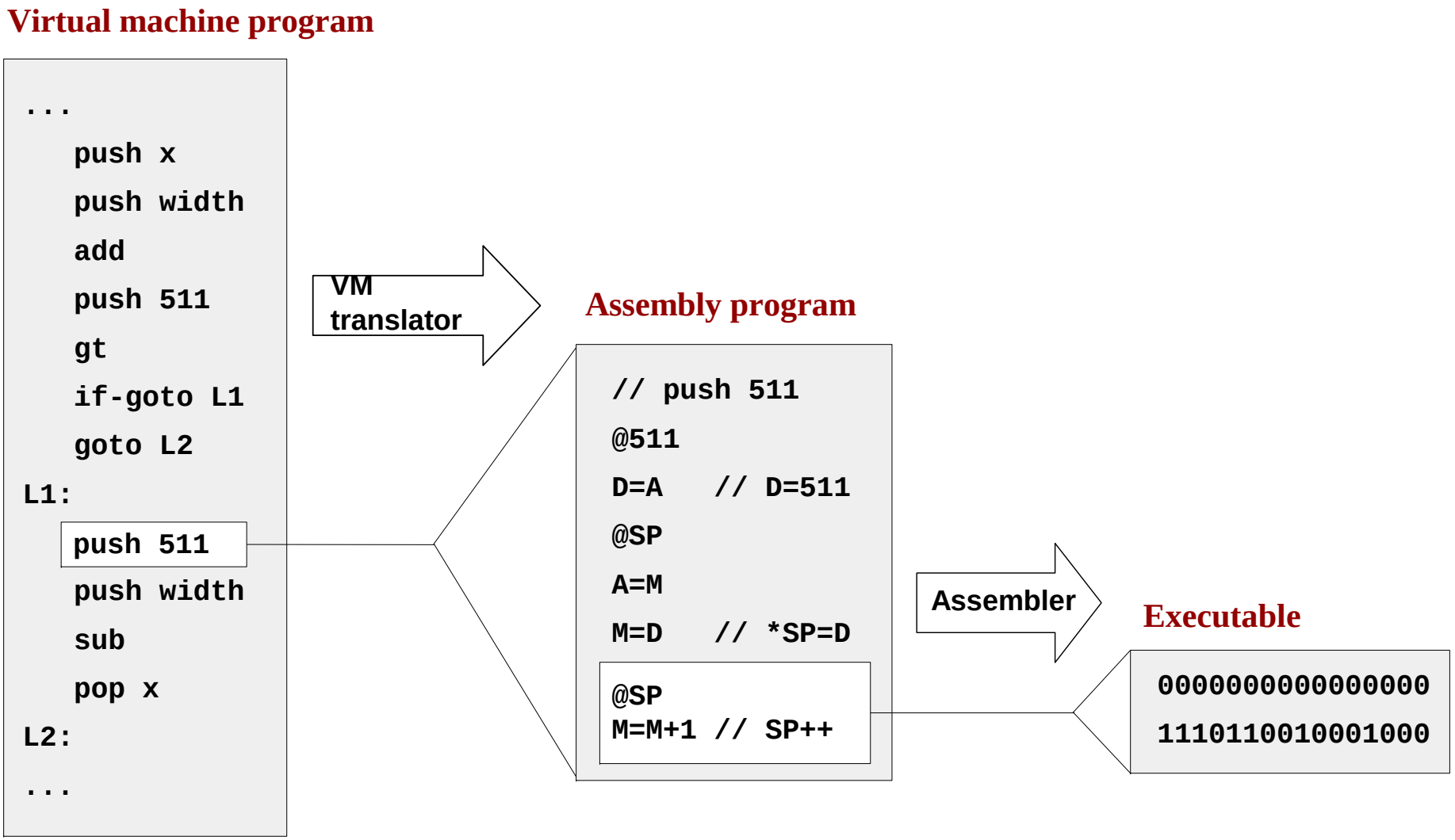
VM translator

Assembly program

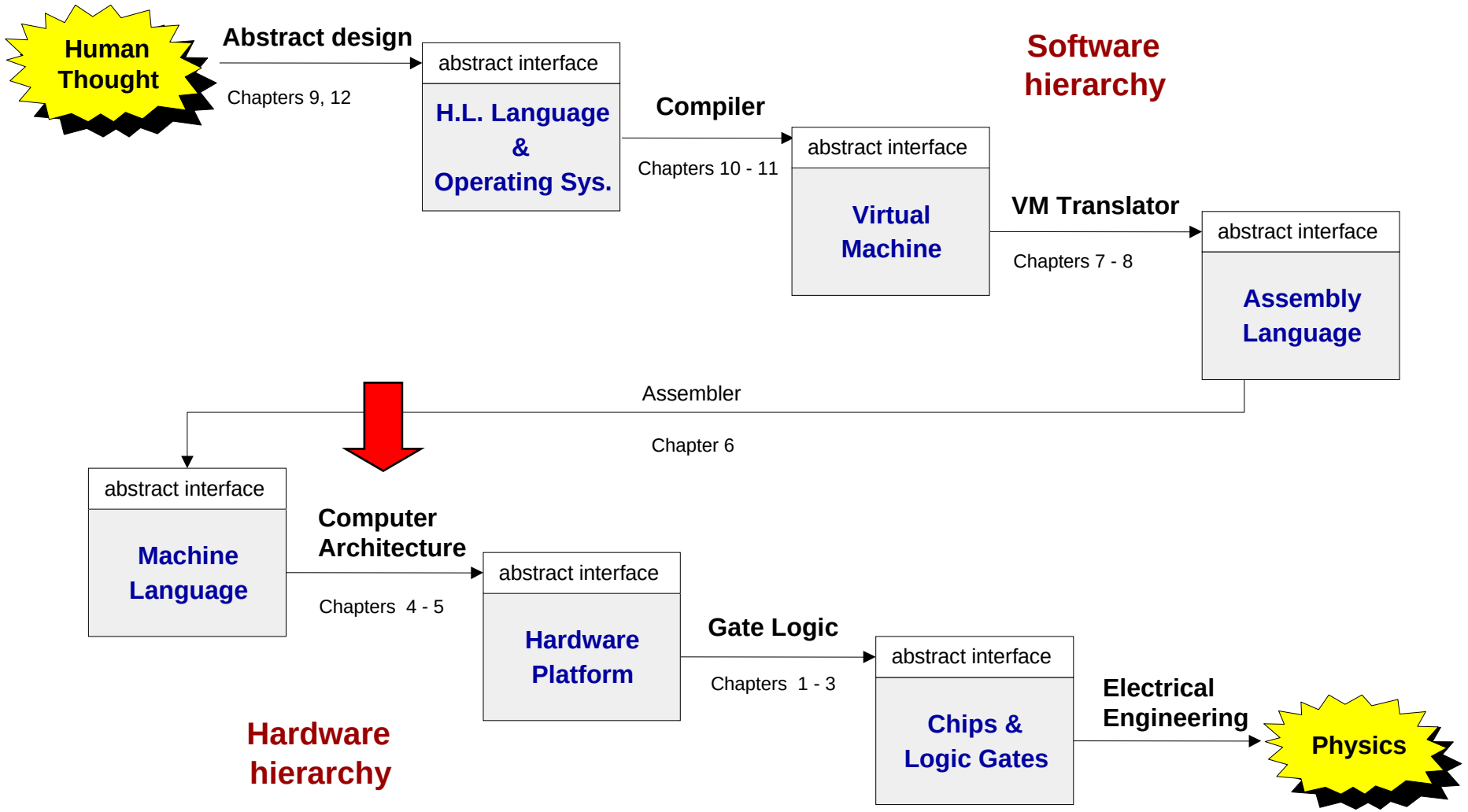
```
// push 511
@511
D=A   // D=511
@SP
A=M
M=D   // *SP=D
@SP
M=M+1 // SP++
```

Low-level programming (on the Hack computer)

For now, ignore all details!



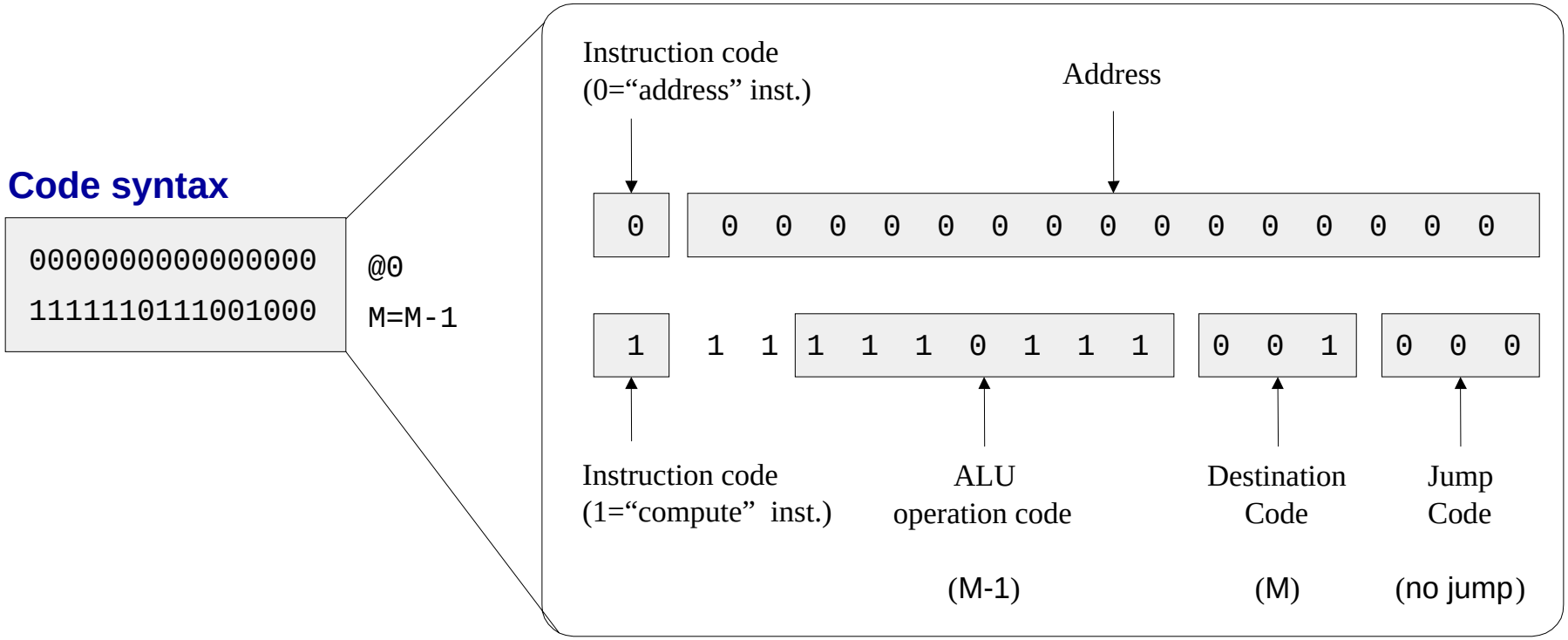
The big picture



Machine language semantics (Hack platform)

For now, ignore all details!

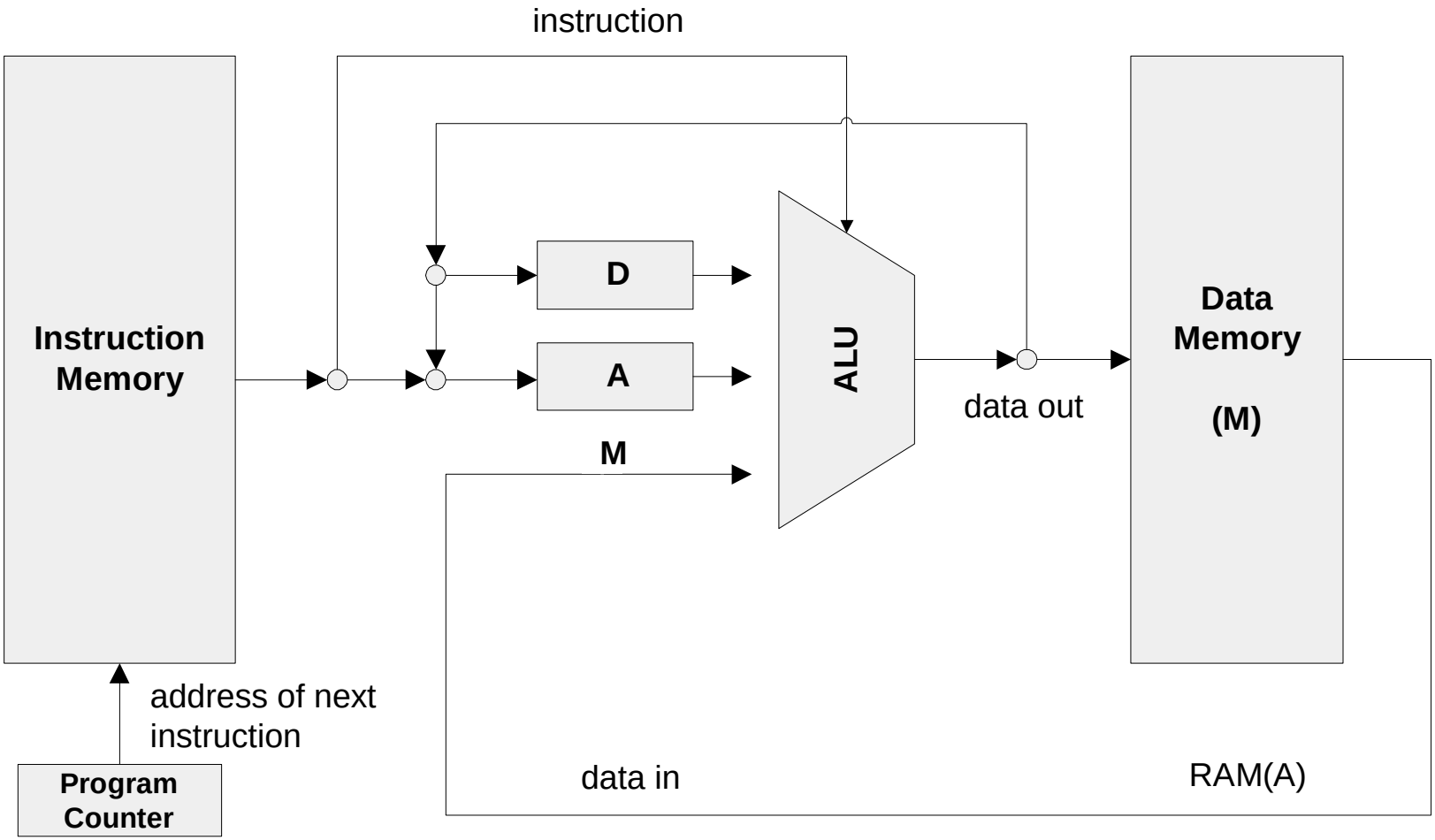
Code semantics, as interpreted by the Hack hardware platform



- We need a hardware architecture that will realize this semantics
- The hardware platform should be designed to:
  - Parse instructions, and
  - Execute them.

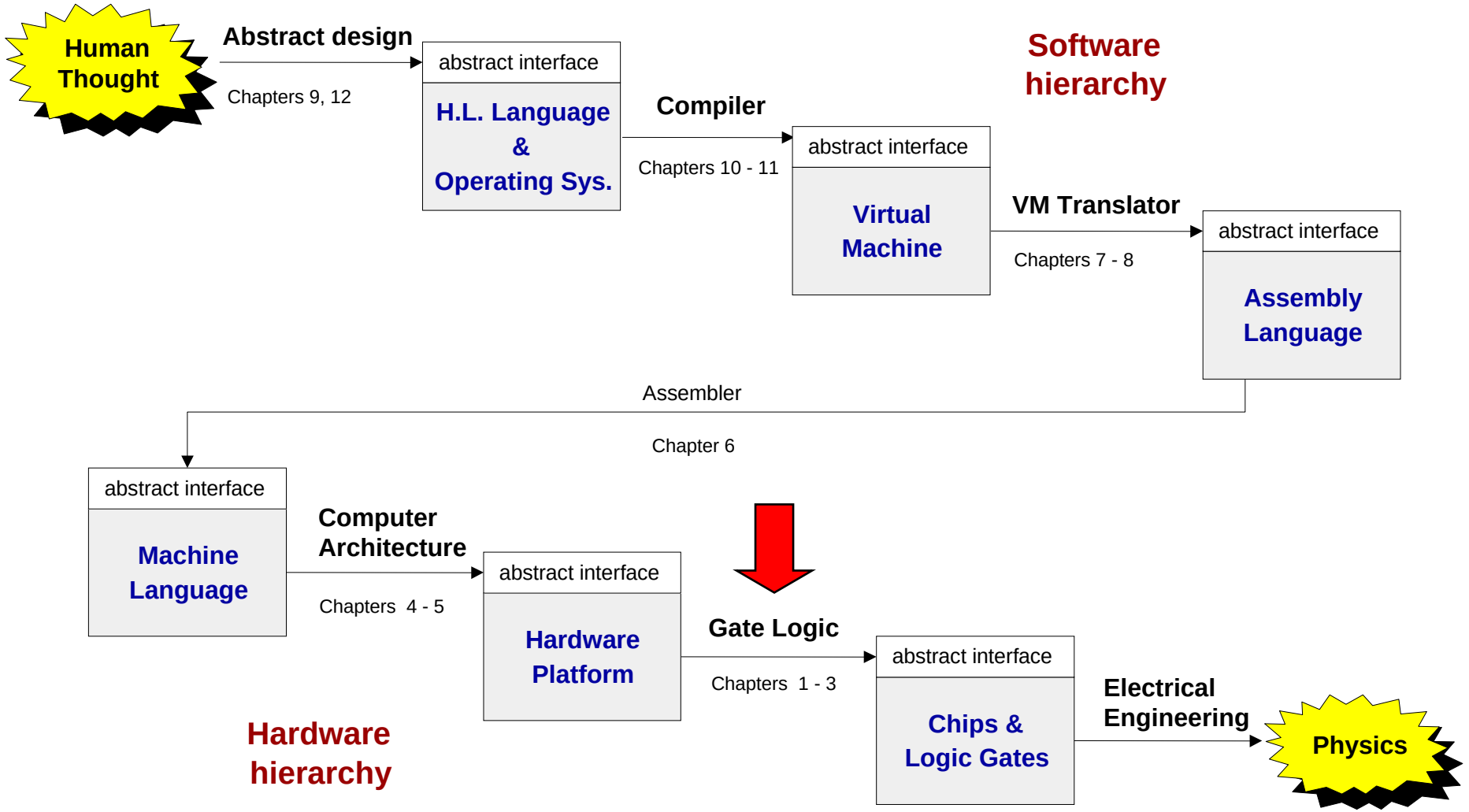
Computer architecture (Hack platform, approx.)

For now, ignore all details!



- A typical Von Neumann machine

The big picture



Logic design

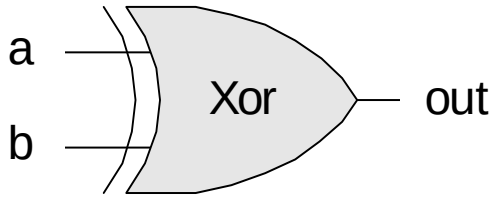
- Combinational logic (leading to an **ALU**)
- Sequential logic (leading to a **RAM**)
- Putting the whole thing together (leading to a **Computer**)

Using … *gate logic.*

Gate logic

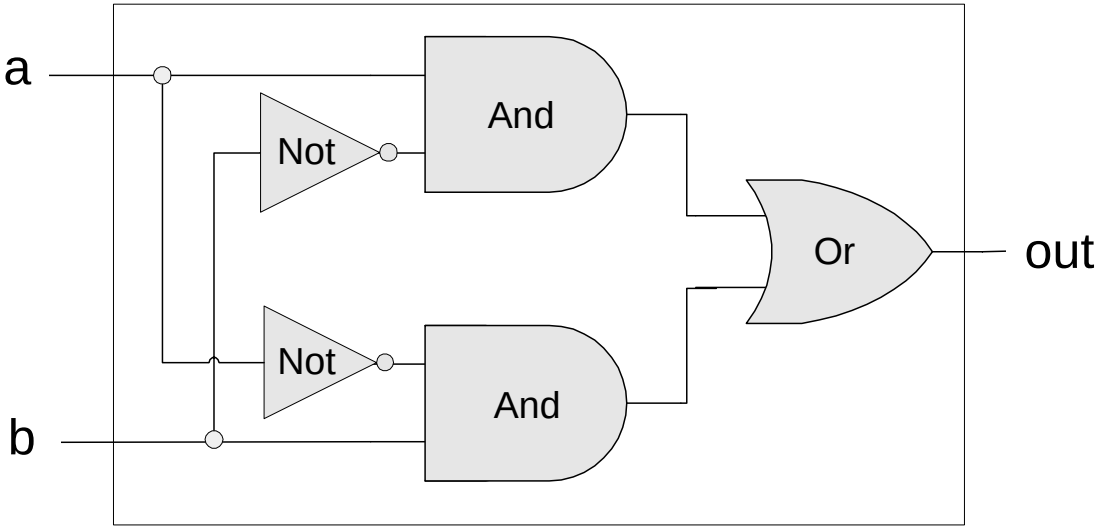
- Hardware platform = inter-connected set of chips
- Chips are made of simpler chips, all the way down to logic gates
- Logic gate = HW element that implements a certain Boolean function
- Every chip and gate has an *interface*, specifying WHAT it is doing, and an *implementation*, specifying HOW it is doing it.

Interface

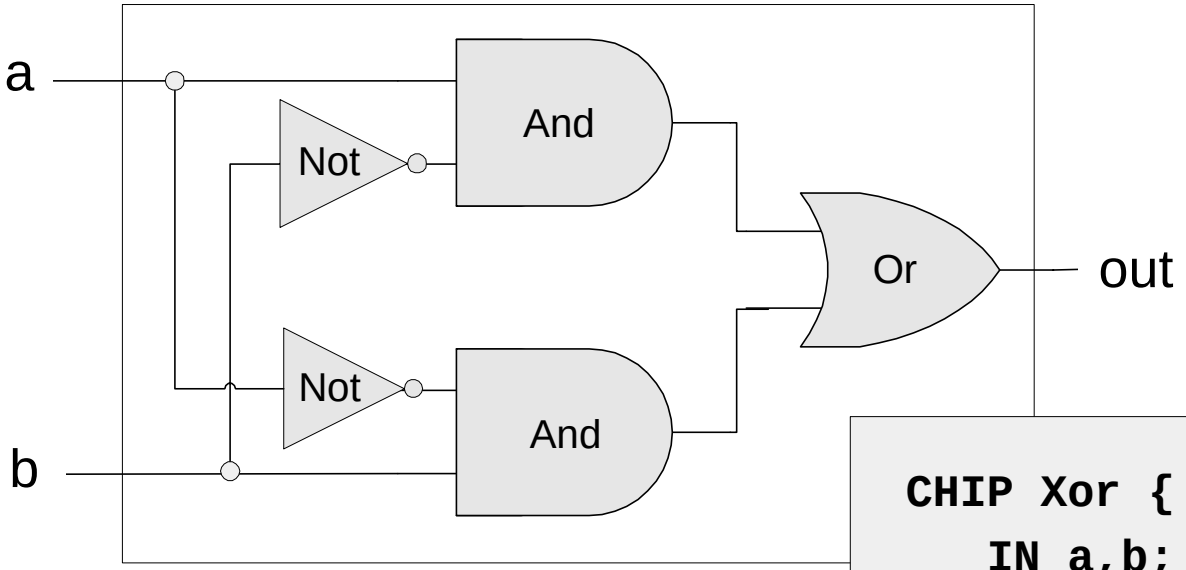


a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Implementation



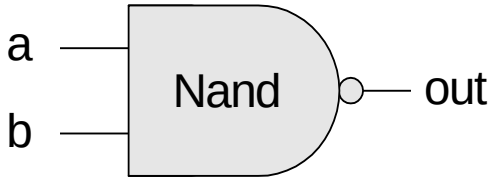
Hardware Description Language (HDL)




```
CHIP Xor {
    IN a,b;
    OUT out;
    PARTS:
    Not(in=a,out=Nota);
    Not(in=b,out=Notb);
    And(a=a,b=Notb,out=w1);
    And(a=Nota,b=b,out=w2);
    Or(a=w1,b=w2,out=out);
}
```

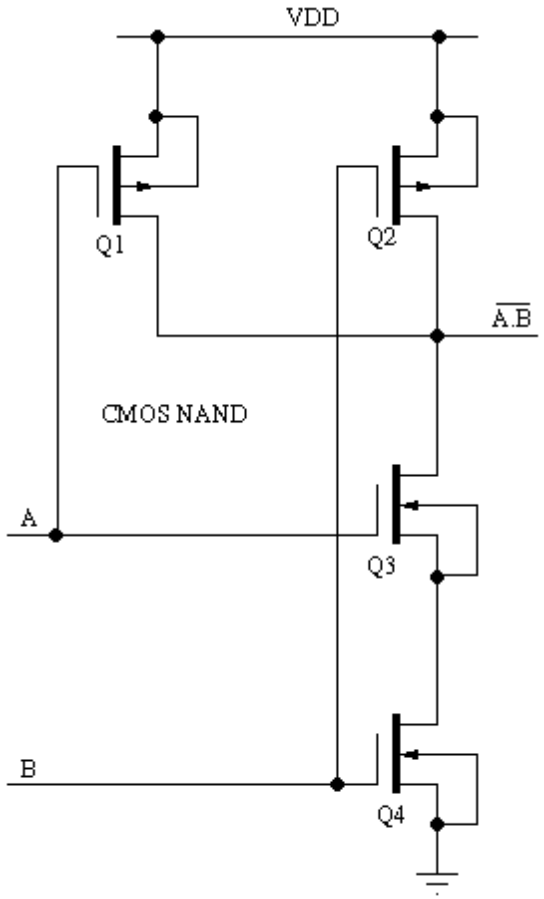
The tour ends:

Interface



a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

One implementation option (CMOS)



The tour map, revisted