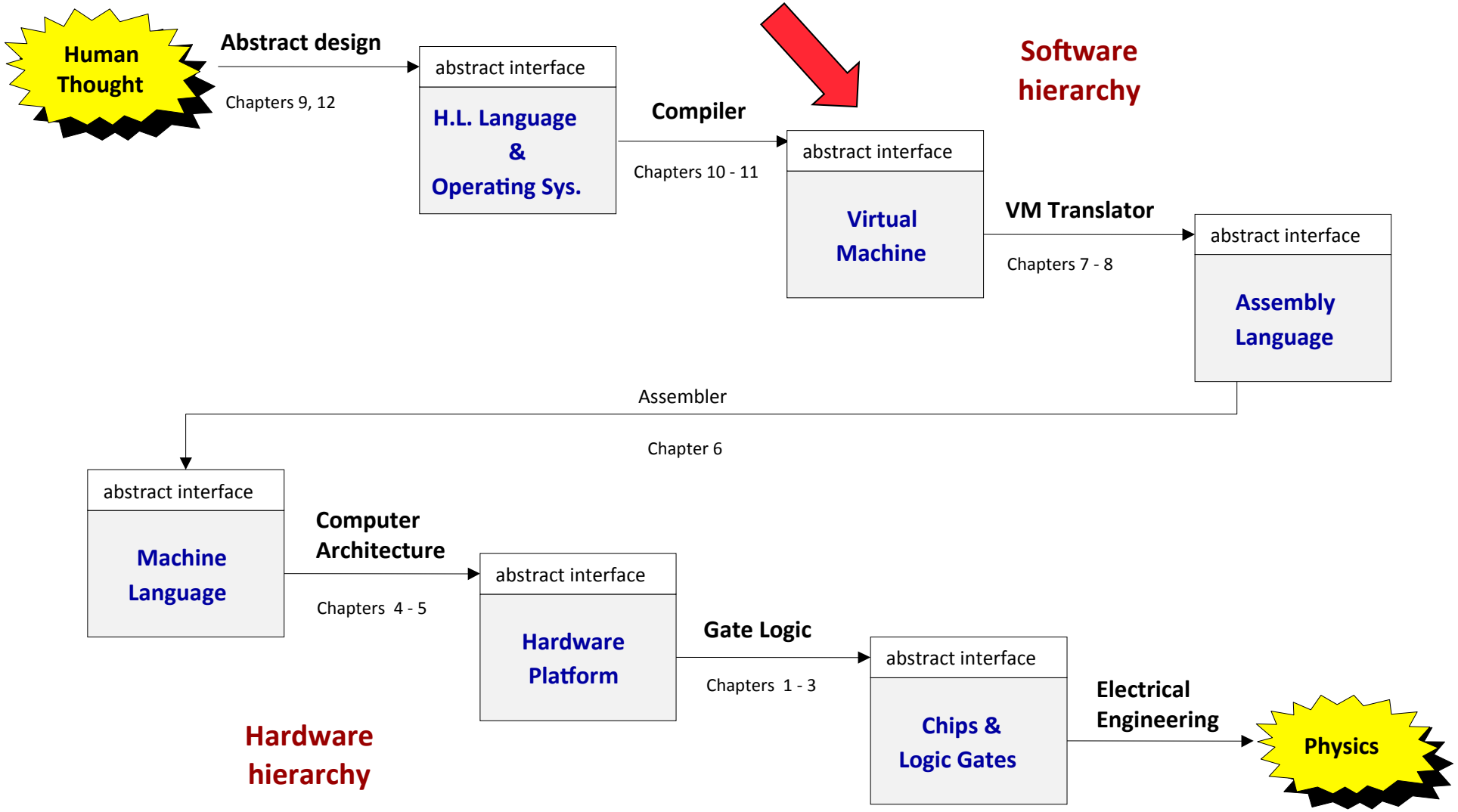
School of Computer Science

COMP SCI 2000 Computer Systems
Lecture 12

adelaide.edu.au

seekLIGHT

Review - where we are:



Motivation

Jack code (example)

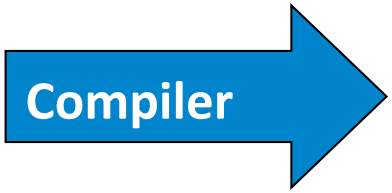
```
class Main
{
  static int x;

  function void main()
  {
    // Inputs and multiplies two numbers
    var int a, b, x;
    let a = Keyboard.readInt("Enter a number");
    let b = Keyboard.readInt("Enter a number");
    let x = mult(a,b);
    return;
  }
}

  // Multiplies two numbers.
  function int mult(int x, int y)
  {
    var int result, j;
    let result = 0; let j = y;
    while ~(j = 0)
    {
      let result = result + x;
      let j = j - 1;
    }
    return result;
  }
}
```

Our ultimate goal:

Translate high-level programs into executable code.

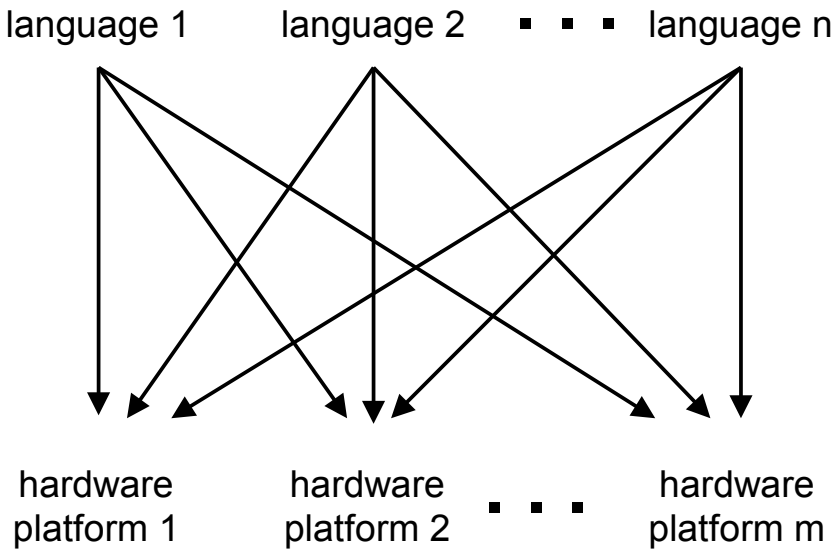


Hack code

```
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000000000000
1111010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000000010001
0000000000010000
1110111111001000
0000000000010001
1110101010001000
0000000000010000
1111110000010000
0000000000000000
1111010011010000
0000000000010010
1110001100000001
0000000000010000
1111110000010000
0000000000010001
  ...
```

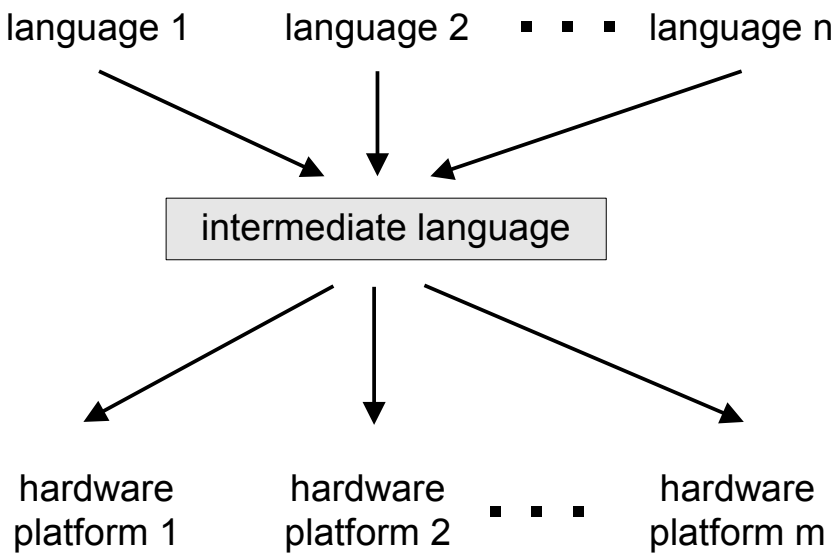
Compilation models

direct compilation:



requires $n \bullet m$ translators

2-tier compilation:

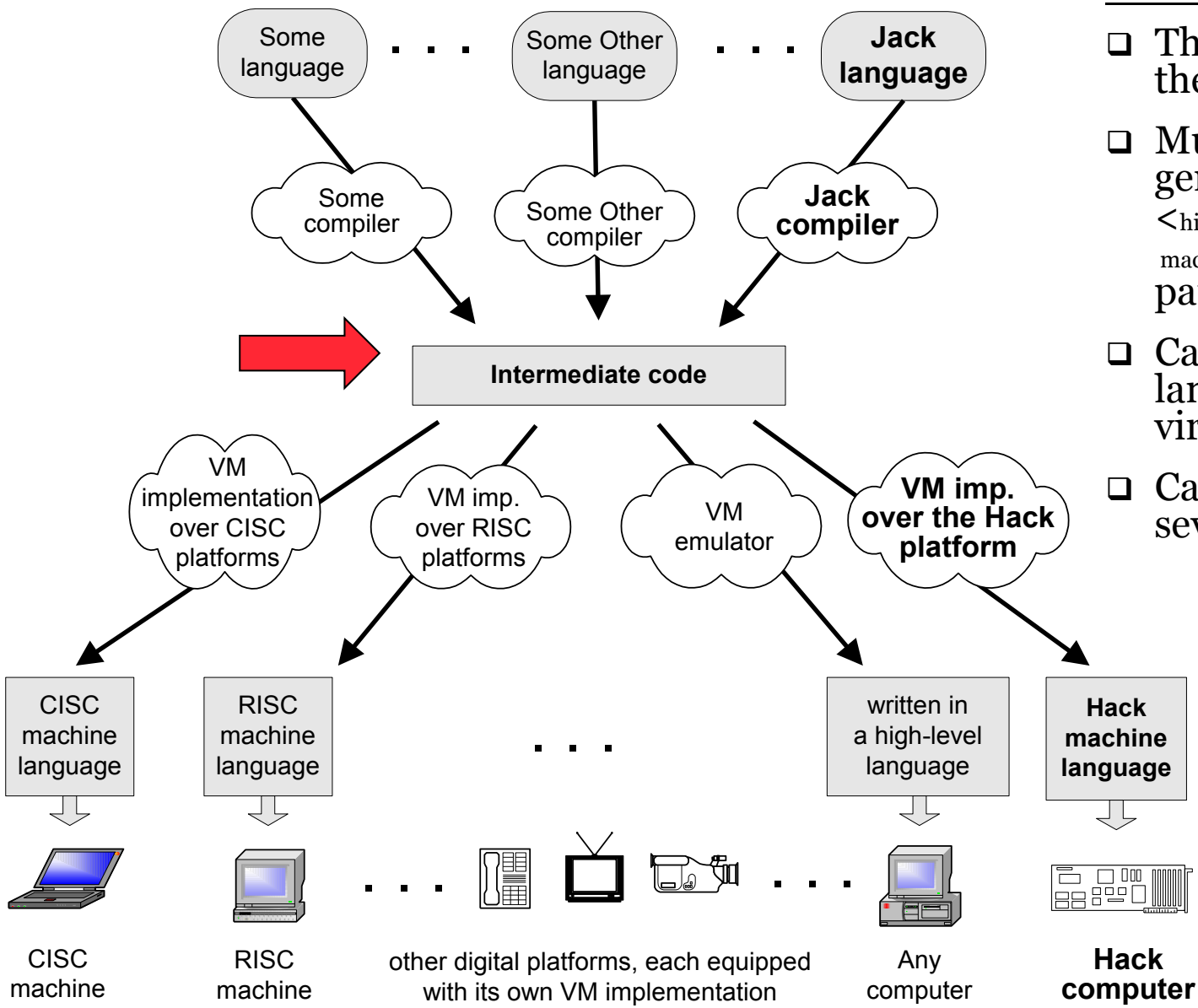


requires $n + m$ translators

Two-tier compilation:

- First compilation stage: depends only on the details of the source language
- Second compilation stage: depends only on the details of the target language.

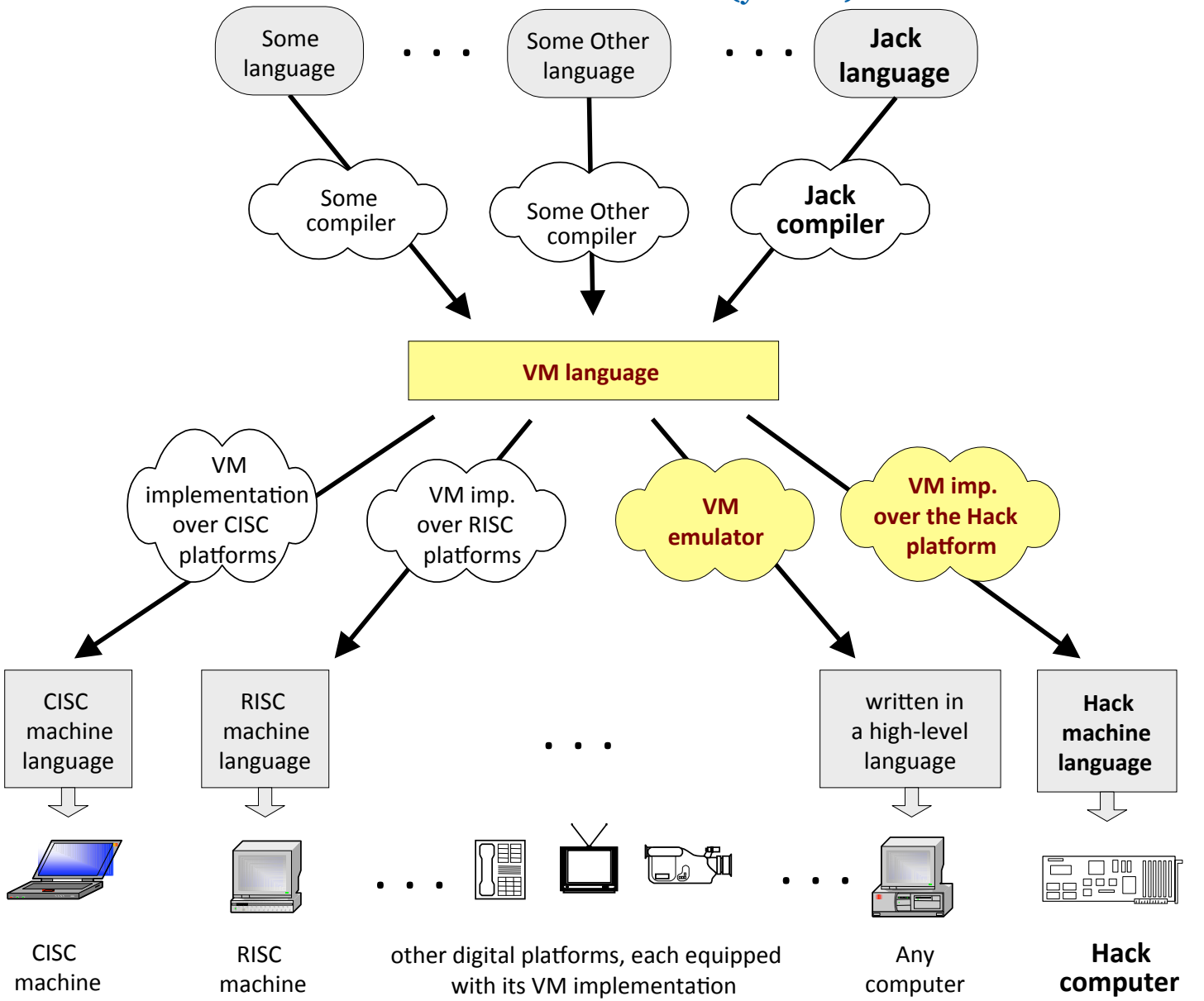
The big picture



The intermediate code:

- The interface between the 2 compilation stages
- Must be sufficiently general to support many <high-level language, machine-language> pairs
- Can be modeled as the language of an abstract virtual machine (VM)
- Can be implemented in several different ways.

Focus of this lecture (yellow):



Book chapters and Course projects:

9, 10, 11, 12

7, 8 (this and the next lecture)

1, 2, 3, 4, 5, 6

Other Examples of Virtual Machines

- Worksheet Lecture 12, Question 1
  - Examples of virtual machines and their uses

The VM model and language

Perspective:

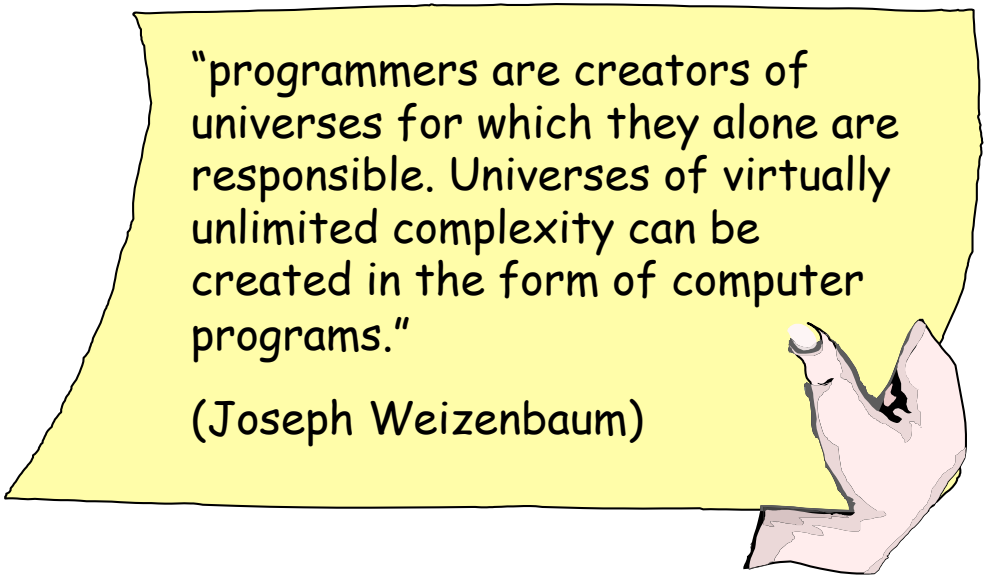
From here till the end of the next lecture we describe the VM model used in the Hack-Jack platform

Other VM models (like Java's JVM/JRE and .NET's IL/CLR) are similar in spirit but differ in scope and details.

Several different ways to think about the notion of a virtual machine:

- **Abstract software engineering view:**
  the VM is an interesting abstraction that makes sense in its own right
- **Practical software engineering view:**
  the VM code layer enables "managed code" (e.g. enhanced security)
- **Pragmatic compiler writing view:**
  a VM architecture makes writing a compiler much easier
  (as we'll see later in the course)
- **Opportunistic empire builder view:**
  a VM architecture allows writing high-level code once and have it run on many target platforms with little or no modification.

Yet another view (poetic)



Our VM model + language are an example of one such universe.

Lecture plan

Goal: Specify and implement a VM model and language:

Arithmetic / Boolean commands

```
add
sub
neg
eq
gt
lt
and
or
not
```

This lecture

Memory access commands

```
pop x   (pop into x, which is a variable)
push y  (y being a variable or a constant)
```

Program flow commands

```
label     (declaration)
goto      (label)
if-goto   (label)
```

Next lecture

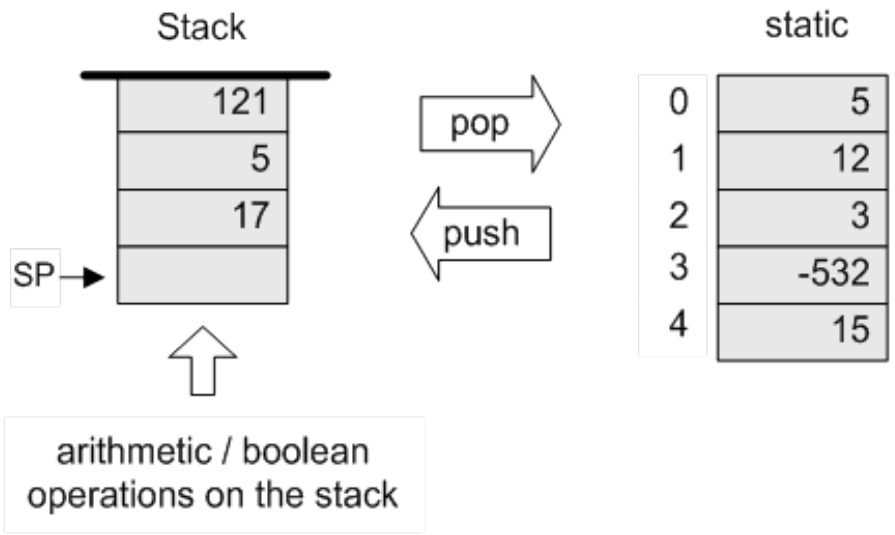
Function calling commands

```
function  (declaration)
call      (a function)
return    (from a function)
```

Our game plan: (a) describe the VM abstraction (above)
(b) propose how to implement it over the Hack platform.

Our VM model is *stack-oriented*

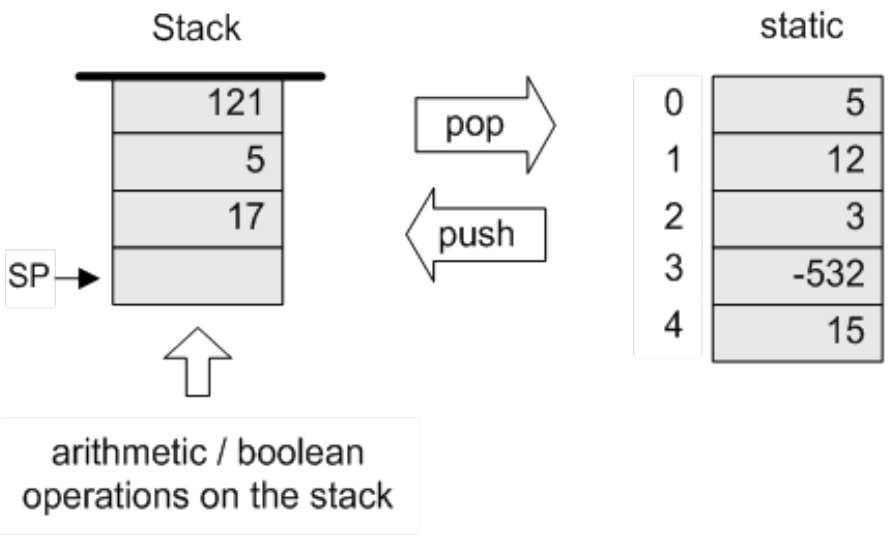
- All operations are done on a stack
- Data is saved in several separate *memory segments*
- All the memory segments behave the same
- One of the memory segments m is called `static`, and we will use it (as an arbitrary example) in the following examples:



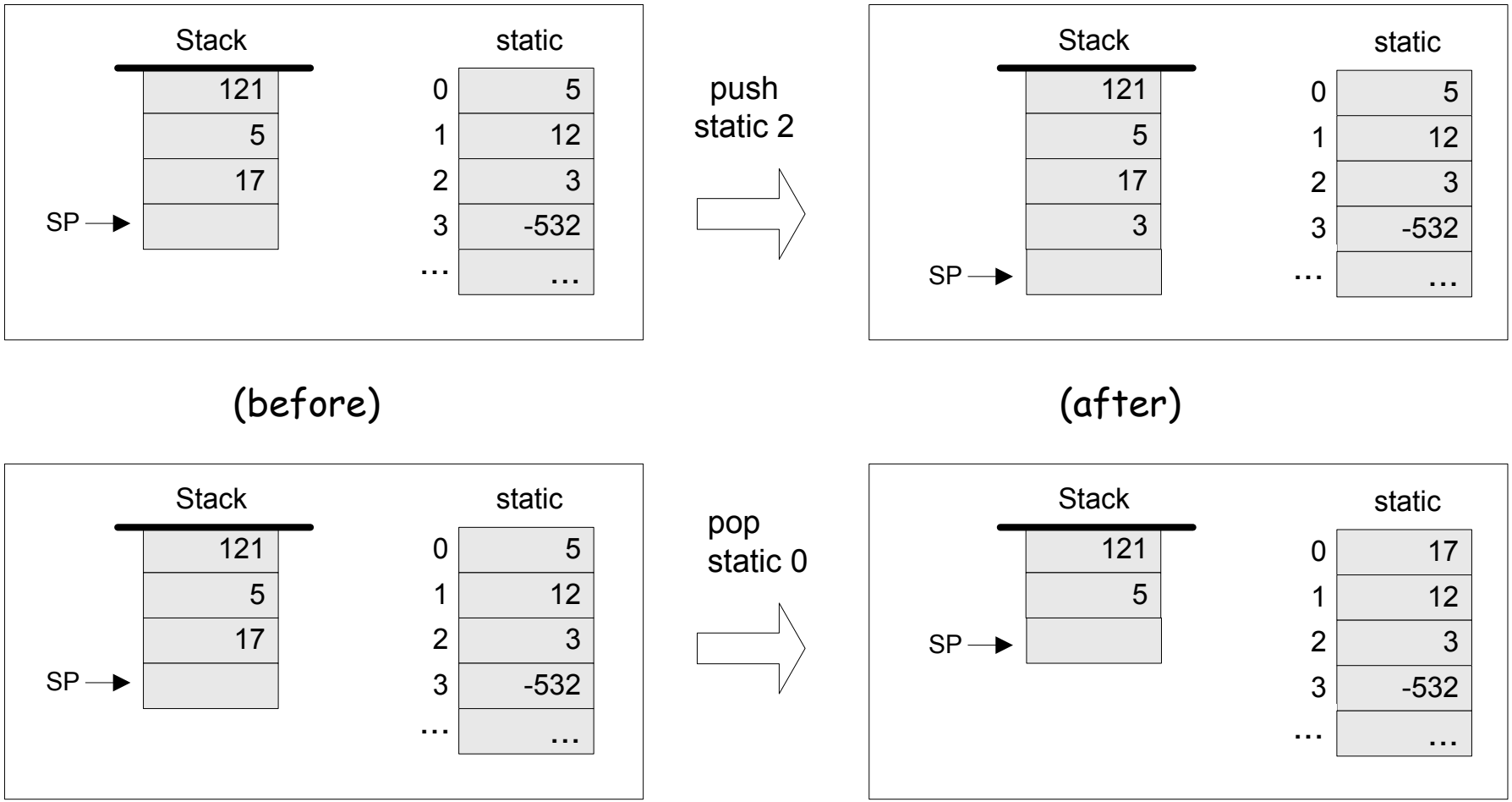
Data types

Our VM model features a single 16-bit data type that can be used as:

- an integer value (16-bit 2's complement: `-32768`, ... , `32767`)
- a Boolean value (`0` and `-1`, standing for `true` and `false`)
- a pointer (memory address)



Memory access operations



The stack:

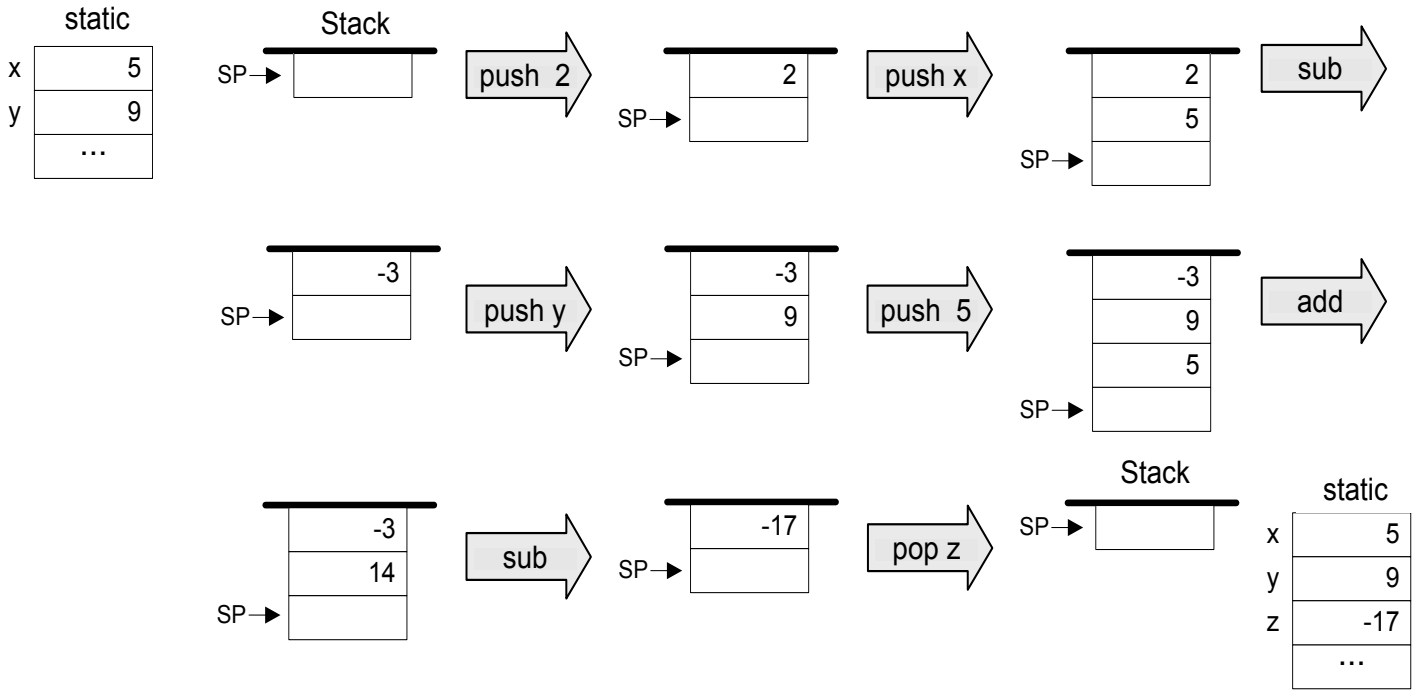
- A classical LIFO data structure
- Elegant and powerful
- Several hardware / software implementation options.

Evaluation of arithmetic expressions

VM code (example)

```
// z=(2-x)-(y+5)
push 2
push x
sub
push y
push 5
add
sub
pop z
```

(suppose that
x refers to static 0,
y refers to static 1, and
z refers to static 2)

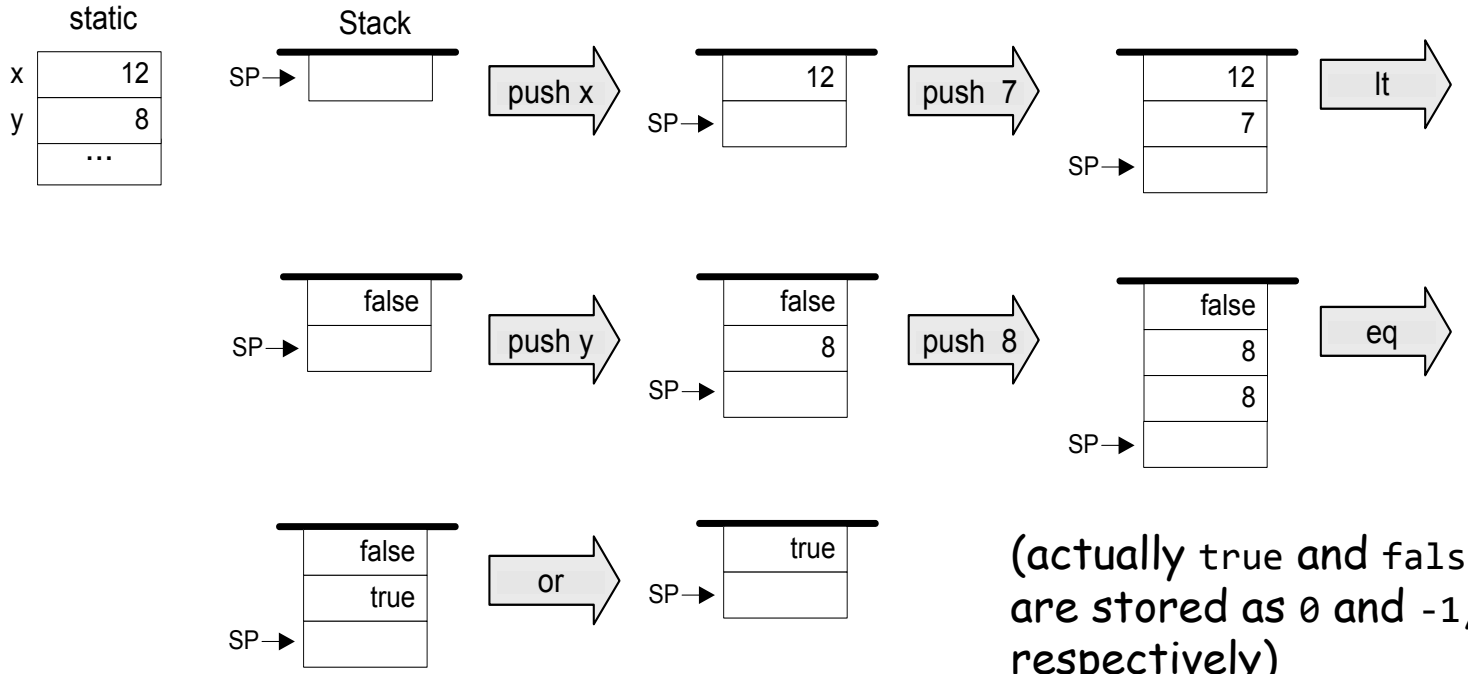


Evaluation of Boolean expressions

VM code (example)

```
// (x<7) or (y=8)
push x
push 7
lt
push y
push 8
eq
or
```

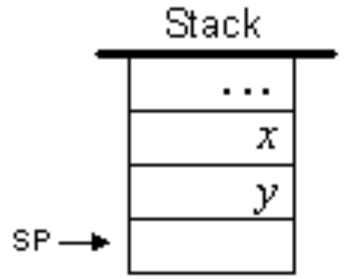
(suppose that
x refers to static 0, and
y refers to static 1)



(actually `true` and `false` are stored as `0` and `-1`, respectively)

Arithmetic and Boolean commands in the VM language (wrap-up)

Command	Return value (after popping the operand/s)	Comment
`add`	$x + y$	Integer addition (2's complement)
`sub`	$x - y$	Integer subtraction (2's complement)
`neg`	$-y$	Arithmetic negation (2's complement)
`eq`	true if $x = y$ and false otherwise	Equality
`gt`	true if $x > y$ and false otherwise	Greater than
`lt`	true if $x < y$ and false otherwise	Less than
`and`	x And y	Bit-wise
`or`	x Or y	Bit-wise
`not`	Not y	Bit-wise



Stack Machine Examples

- Worksheet lecture 12, Question 2
  - Translating expressions into Hack Virtual Machine code

Next lecture

- Virtual machine memory model
- Translating VM code into assembly language
- Keep working on your next assignment.
- Questions?