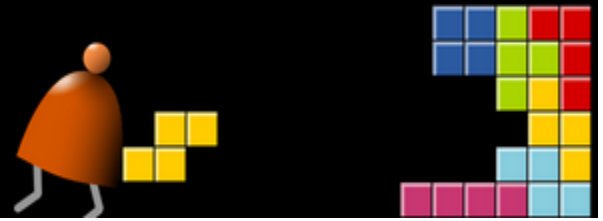


Nand to Tetris, Part II: Software

Slide deck for the "Part II: Software" chapter of the book

The Elements of Computing Systems (2nd edition)

By Noam Nisan and Shimon Schocken

MIT Press

Hello, World Below

High-level program (Jack)

```
// First example in Programming 101
class Main {
    function void main() {
        do Output.printString("Hello World!");
        return;
    }
}
```

Issues

- How does the computer execute this program?
- How does the program write on the screen?
- How are classes and methods realized?
- How is function-call-and-return handled?
- What is the role of the operating system?
- How is memory allocated to objects and arrays?

Add your questions to the list…

Q: How can high-level programmers ignore all these issues?

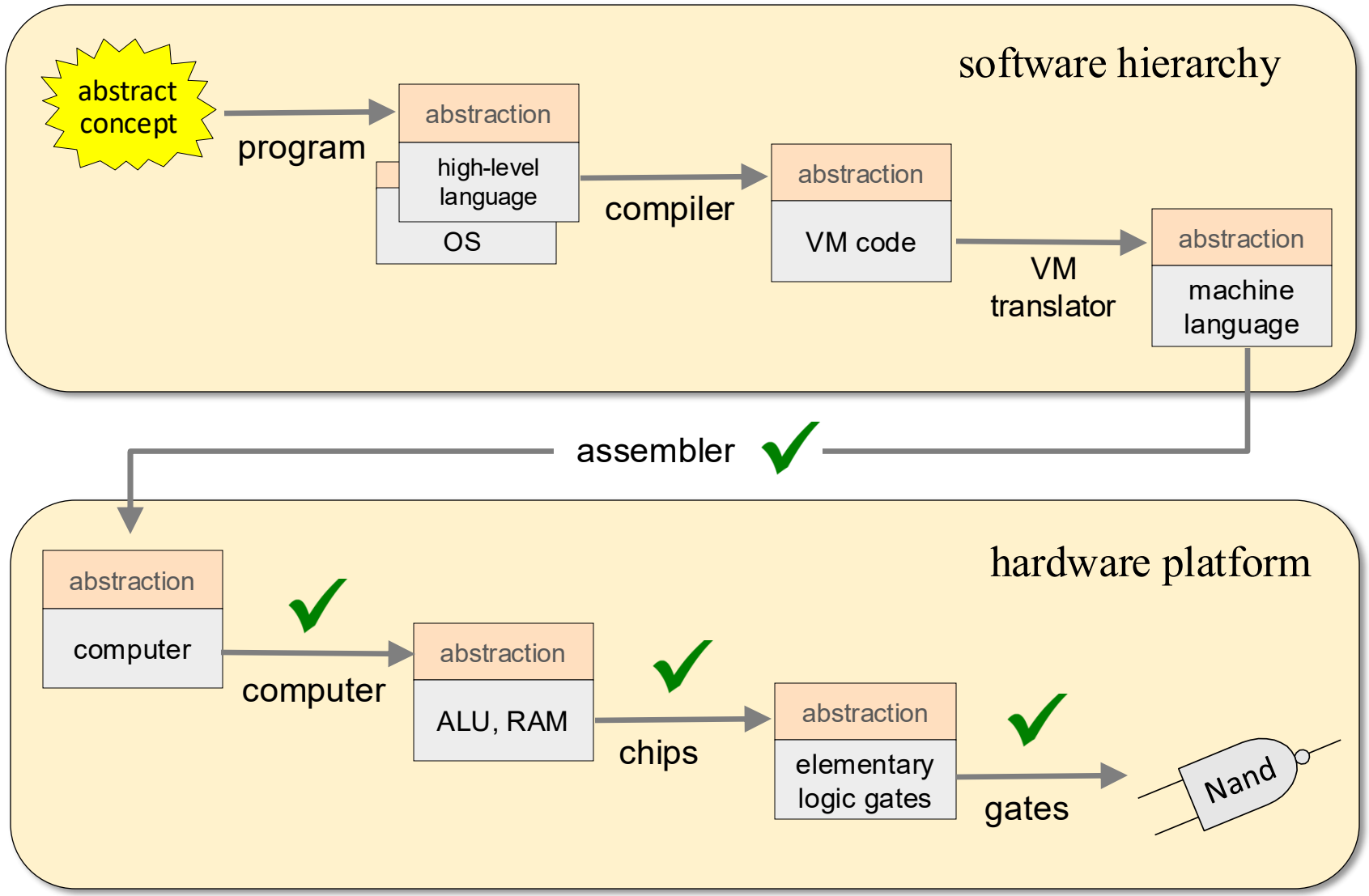
A: They treat the high-level program as an ***abstraction***

What makes the abstraction work?

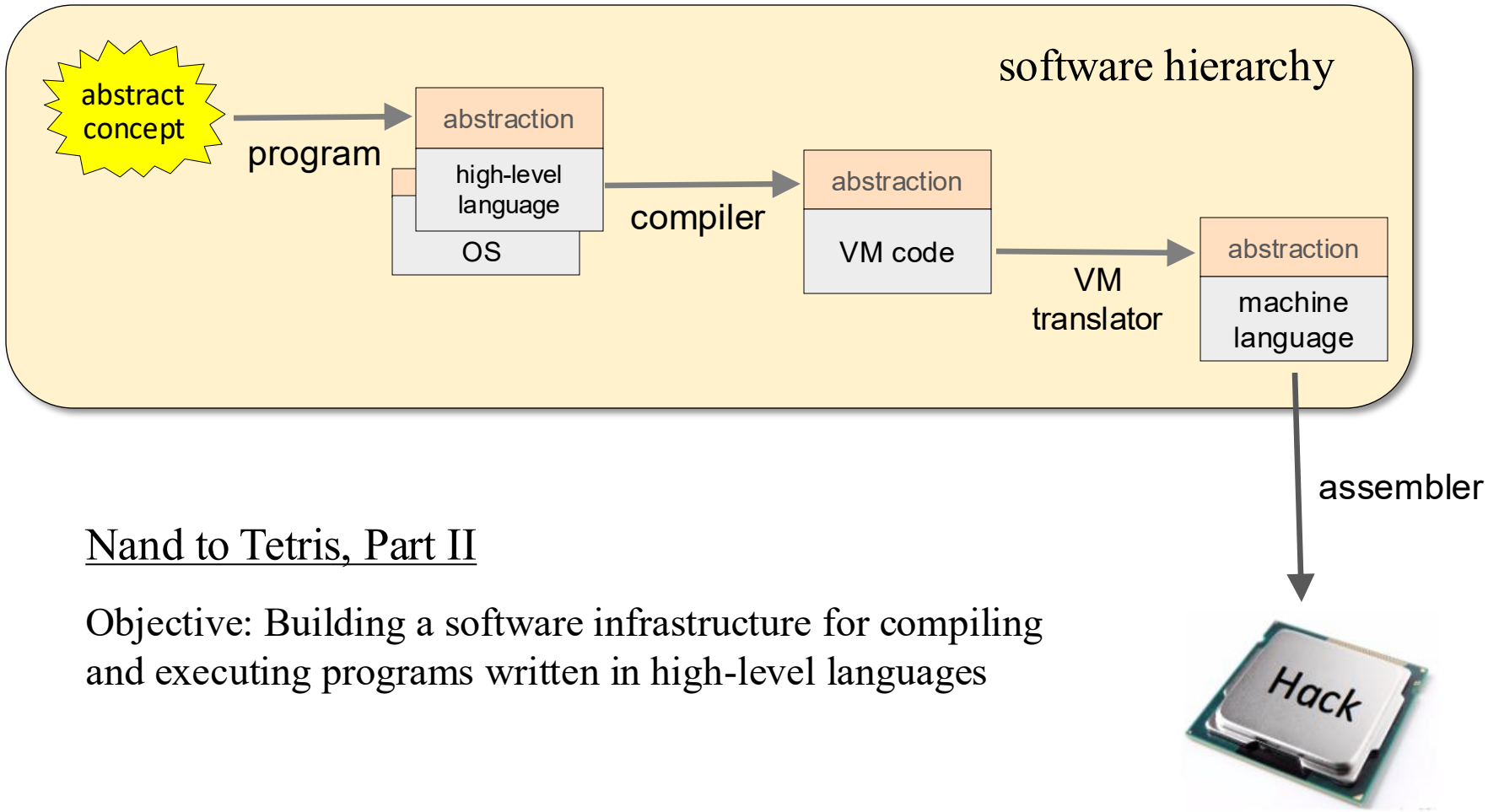
- Assembler
- Virtual machine
- Compiler
- Operating system

} Projects 6-12

Nand to Tetris Roadmap: Part I



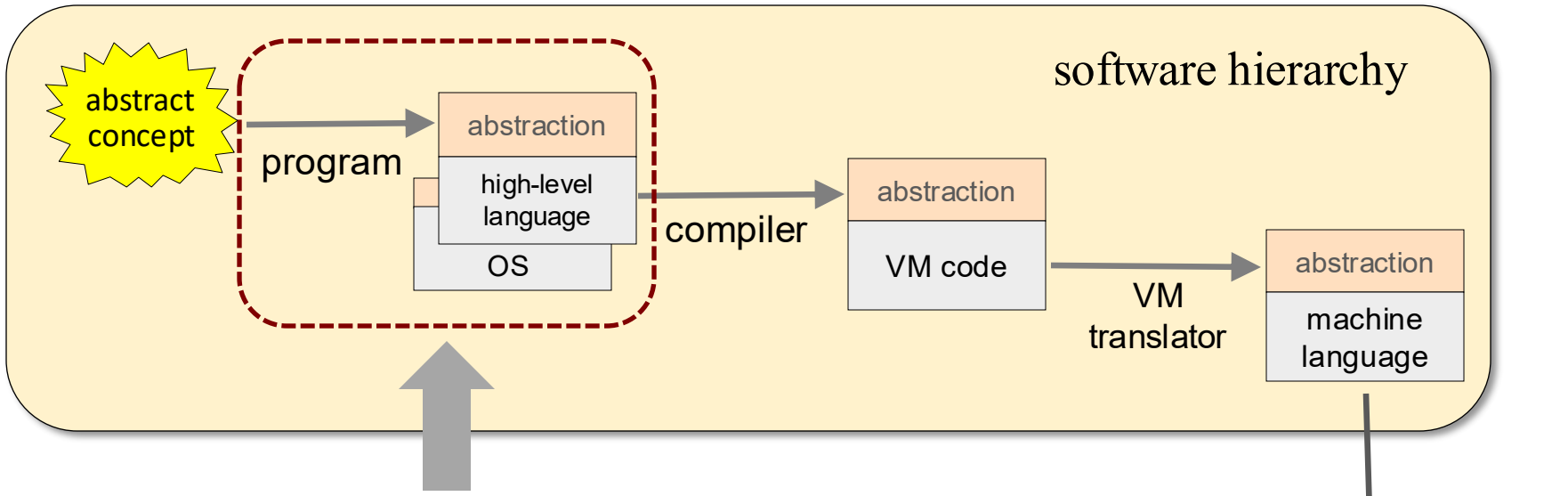
Nand to Tetris Roadmap: Part II



Nand to Tetris, Part II

Objective: Building a software infrastructure for compiling and executing programs written in high-level languages

Nand to Tetris Roadmap: Part II



Nand to Tetris, Part II

Objective: Building a software infrastructure for compiling and executing programs written in high-level languages

In this course we'll focus one such language: *Jack*

Jack: A simple, Java-like, object-based language.

High-level language

Example: write a Jack program that

- Constructs two 2D points, `p1` and `p2`
- Computes and prints `p3 = p1 + p2`
- Computes and prints `distance(p1,p3)`

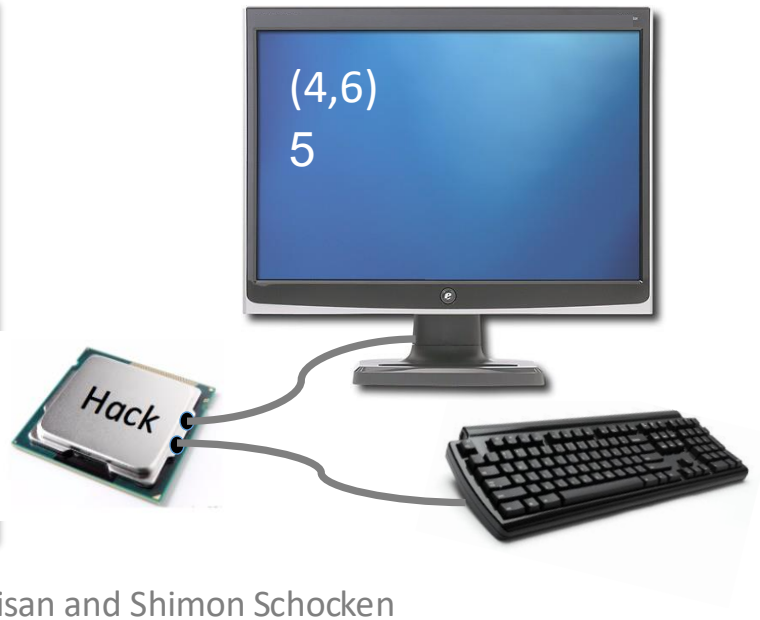
Using a typical object-oriented `Point` class.

```
/** Constructs and manipulates Point objects */
class Main {
  function void main() {
     var Point p1, p2, p3;
     let p1 = Point.new(1,2);
     let p2 = Point.new(3,4);
     let p3 = p1.plus(p2);
     do p3.print();
     do Output.println();
     do Output.printInt(p1.distance(p3));
     return;
  }
}
```

Written in the Jack language

`Point` class (skeletal)

```
/** Represents a 2D point */
class Point {
    /** Constructs a new point with the given coordinates */
    constructor Point new(int ax, int ay)
    /** Returns the point which is this point plus the other point */
    method Point plus(Point other)
    /** Euclidian distance between this and the other point */
    method int distance(Point other)
    /** Prints this point as "(x,y)" */
    method void print()
    ...
}
```



High-level language

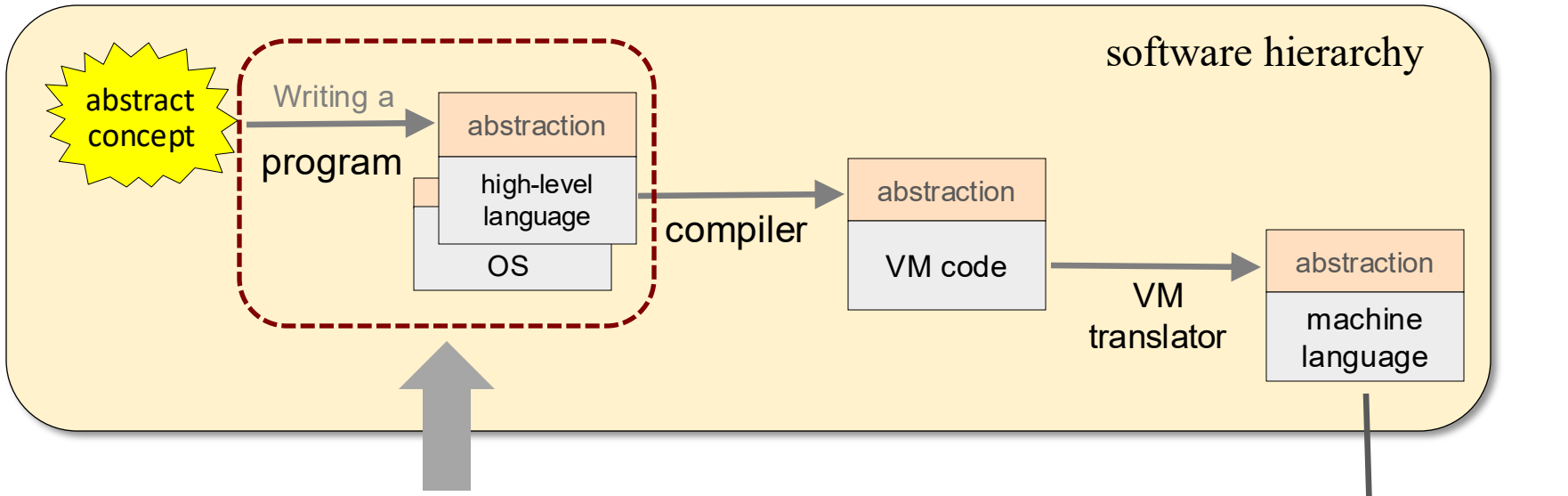
```
/** Represents a 2D point.
    File name: Point.jack. */
class Point {

    // The coordinates of this point
    field int x, y

    // The number of Point objects constructed so far:
    static int pointCount;

    /** Constructs a two-dimensional point and
        initializes it with the given coordinates. */
    constructor Point new(int ax, int ay) {
        let x = ax;
        let y = ay;
        let pointCount = pointCount + 1;
        return this;
    }

    /** Returns the x coordinate of this point. */
    method int getx() { return x; }

    /** Returns the y coordinate of this point. */
    method int gety() { return y; }

    /** Returns the number of points constructed so far. */
    function int getPointCount() {
        return pointCount;
    }

    // Class declaration continues on top right.
```

```
    /** Returns a point which is this
        point plus the other point. */
    method Point plus(Point other) {
        return Point.new(x + other.getx(),
                         y + other.gety());
    }

    /** Returns the Euclidean distance between
        this and the other point. */
    method int distance(Point other) {
        var int dx, dy;
        let dx = x - other.getx();
        let dy = y - other.gety();
        return Math.sqrt((dx*dx) + (dy*dy));
    }

    /** Prints this point as "(x,y)" */
    method void print() {
        do Output.printString("(");
        do Output.printInt(x);
        do Output.printString(",");
        do Output.printInt(y);
        do Output.printString(")");
        return;
    }

} // End of Point class declaration.
```

Jack is a typical object-based, high-level language

Details, later.

Nand to Tetris Roadmap: Part II



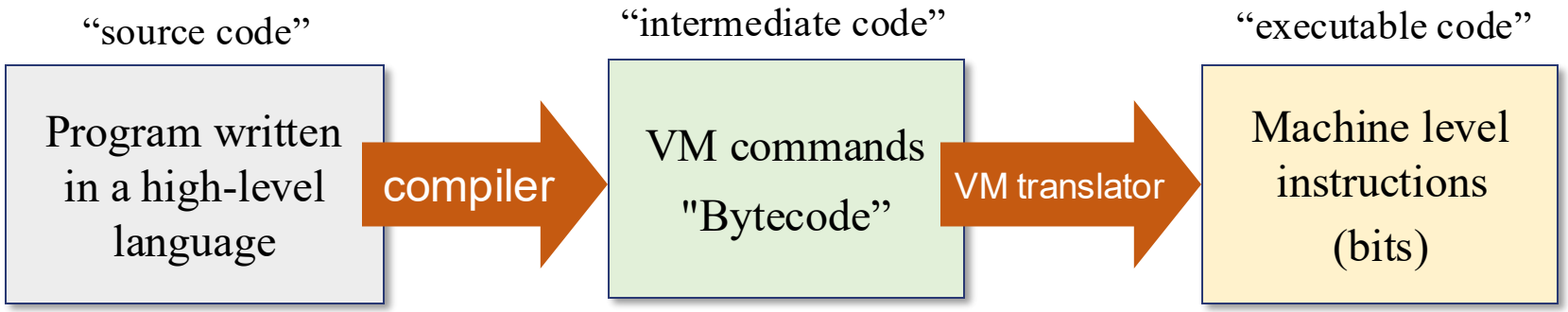
How is this high-level language implemented?

- Compiler
- OS
- Virtual machine
- Assembler

Compilation

One tier

Two tier



Compilation

One tier

Pros

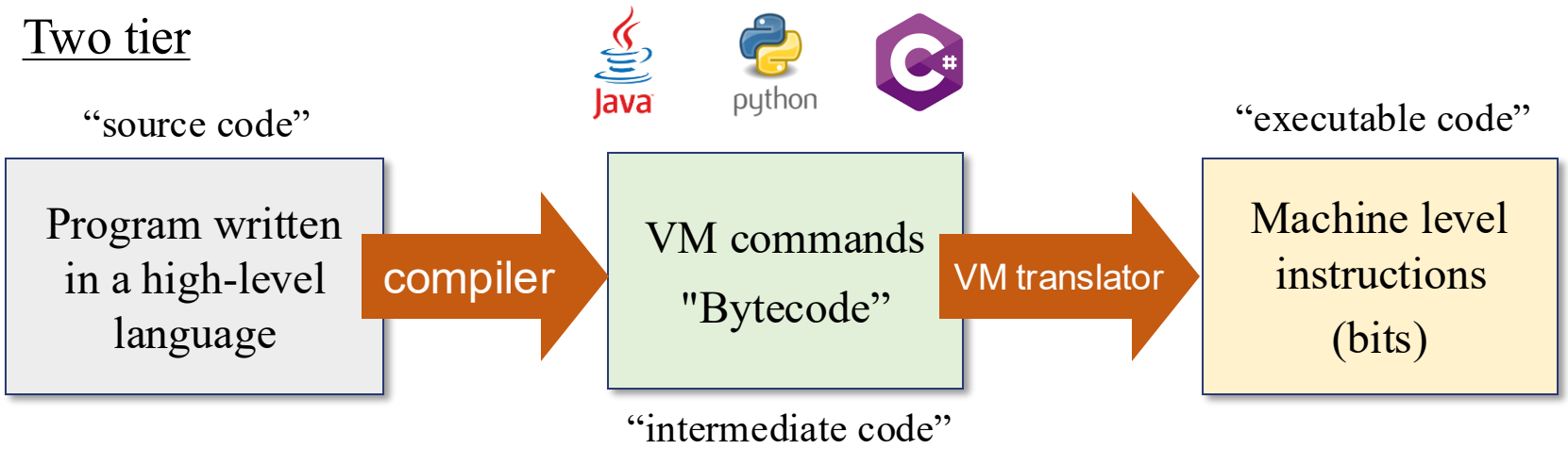
1-tier compilers generate efficient, target-specific code

Cons

Requires multiple compilers / compilations, one for each target hardware platform.

Compilation

Two tier



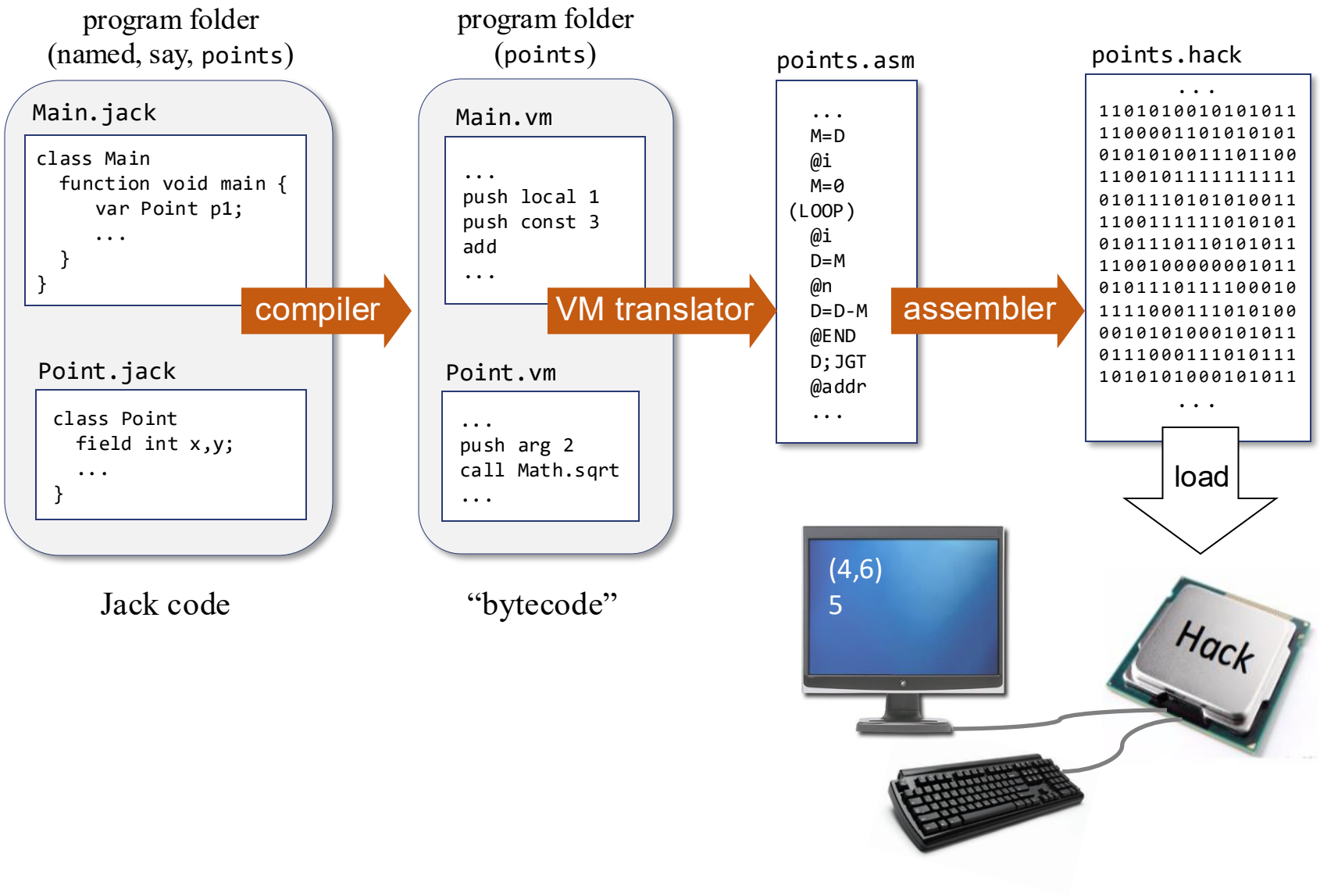
Pros

Compile once, run everywhere

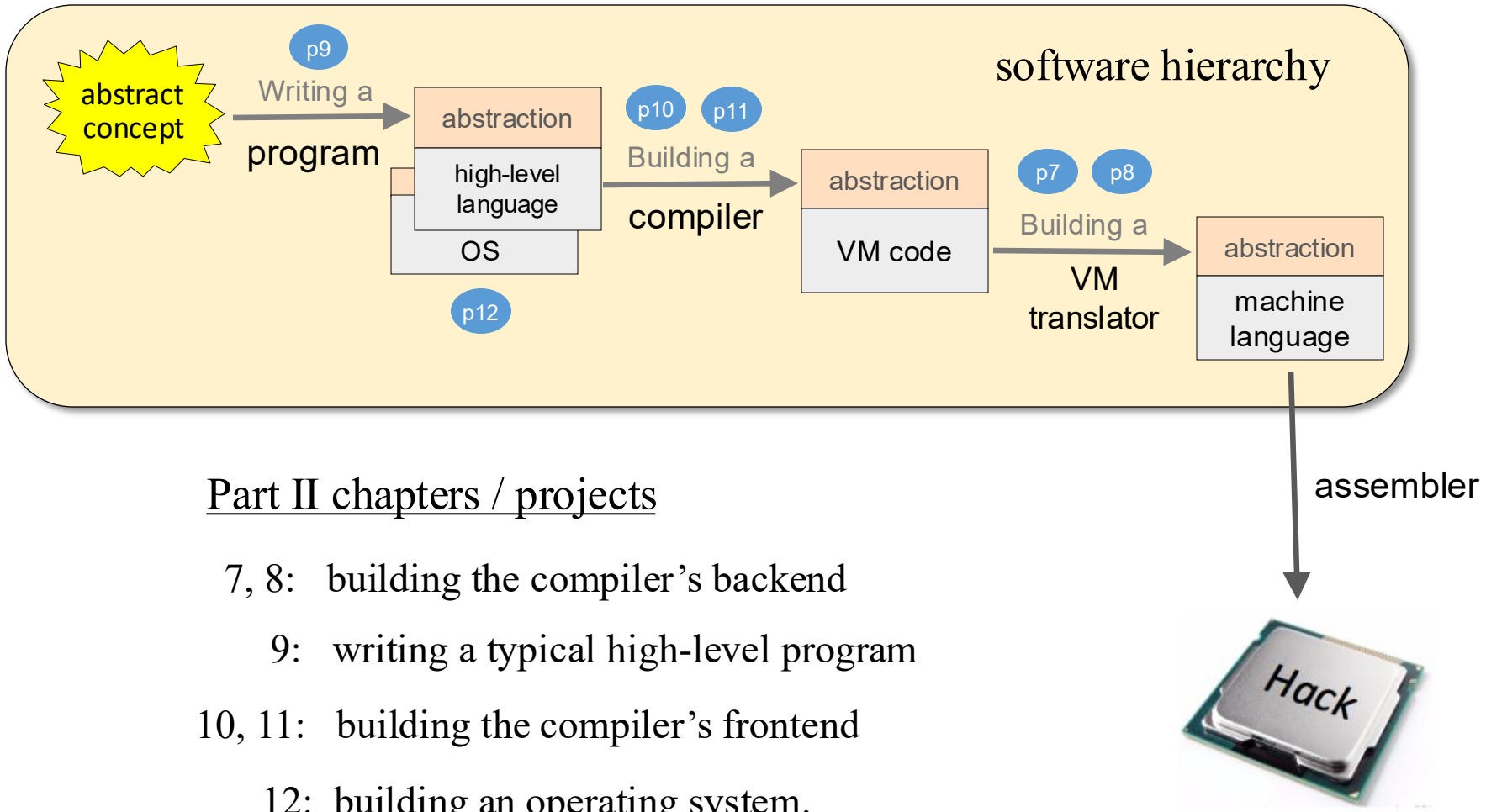
Cons

- Requires multiple VM translators, one for each target hardware platform
  (But: VM translators are much simpler than compilers)
- Generates less efficient machine language code
  (But: For numerous apps, the inefficiency delta is insignificant).

Compilation on the Hack/Jack platform



Nand to Tetris Roadmap: Part II



Part II chapters / projects

7, 8: building the compiler's backend

9: writing a typical high-level program

10, 11: building the compiler's frontend

12: building an operating system.