Introduction to Computer Science II

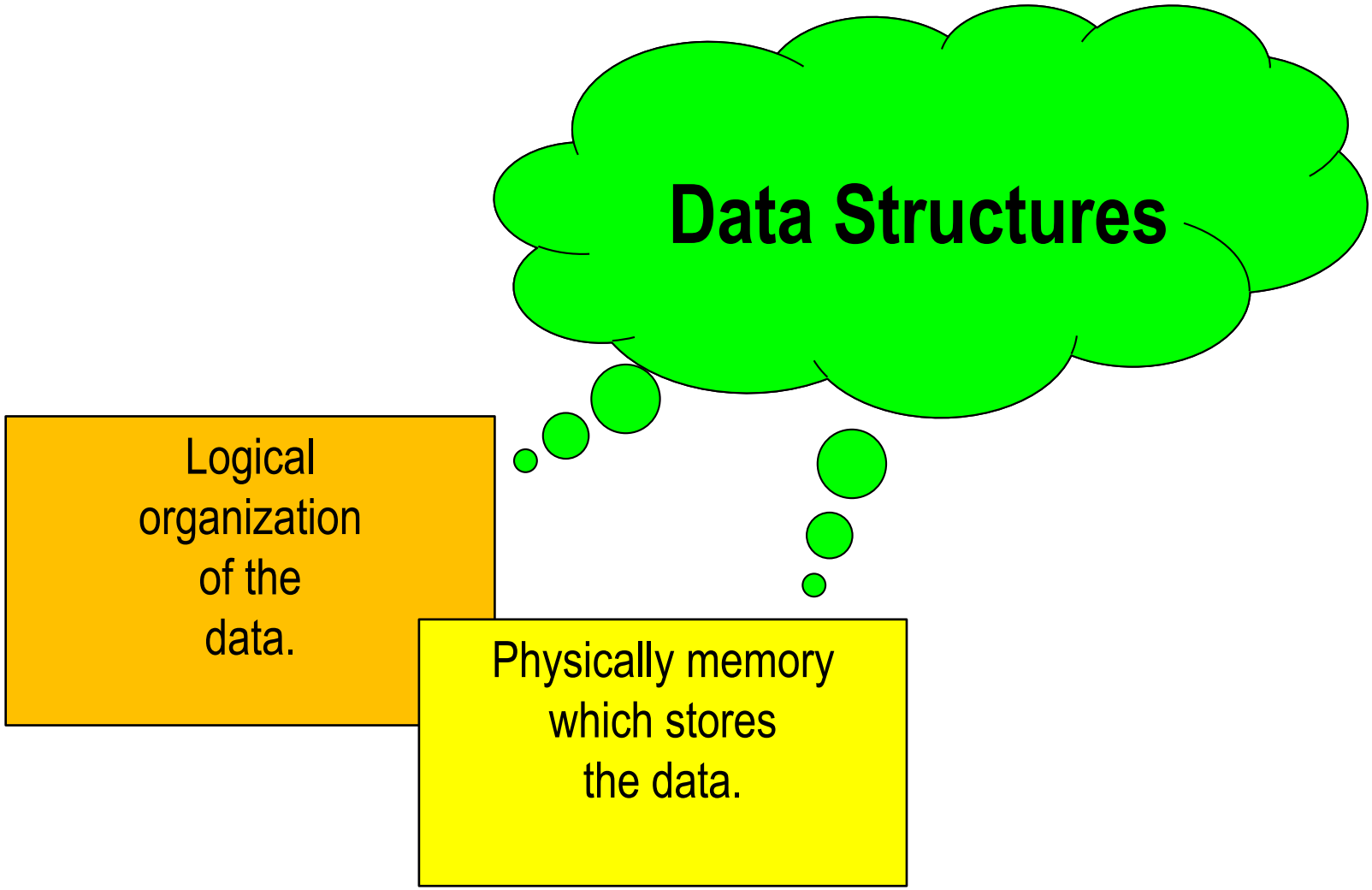
Course Overview

Computer Science 112
Boston University

Christine Papadakis-Kanaris

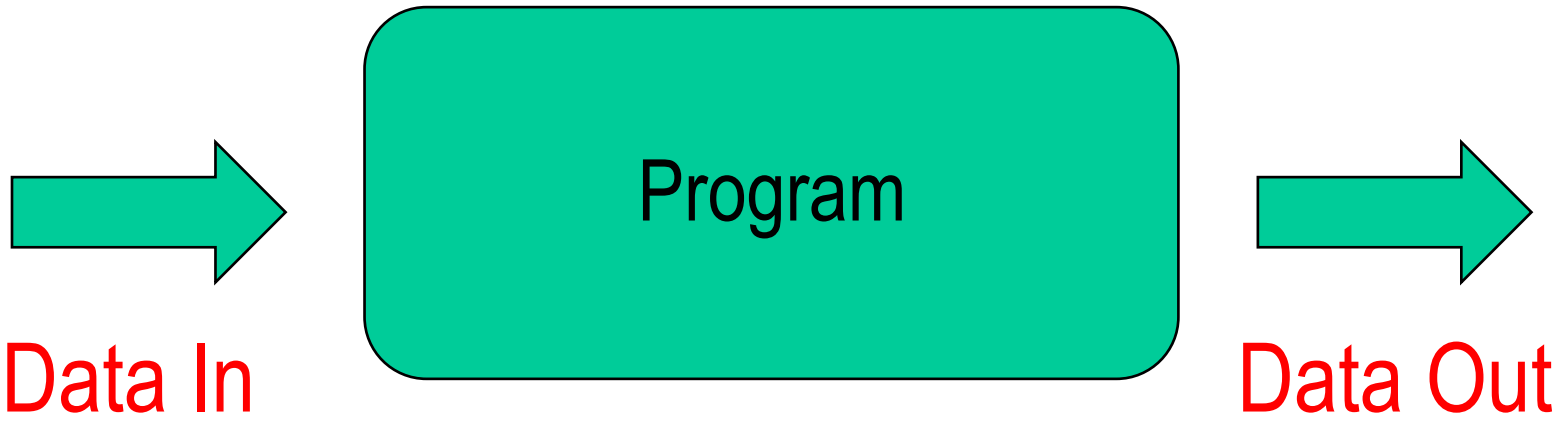
Welcome to Computer Science 112!

- We will study fundamental *data structures.*



Welcome to Computer Science 112!

- We will study fundamental *data structures.*



Welcome to Computer Science 112!

- We will study fundamental *data structures.*

Welcome to Computer Science 112!

- We will study fundamental *data structures.*

How do we store the data?

Welcome to Computer Science 112!

- We will study fundamental *data structures.*

Depends on how we want to use and access the data…

Welcome to Computer Science 112!

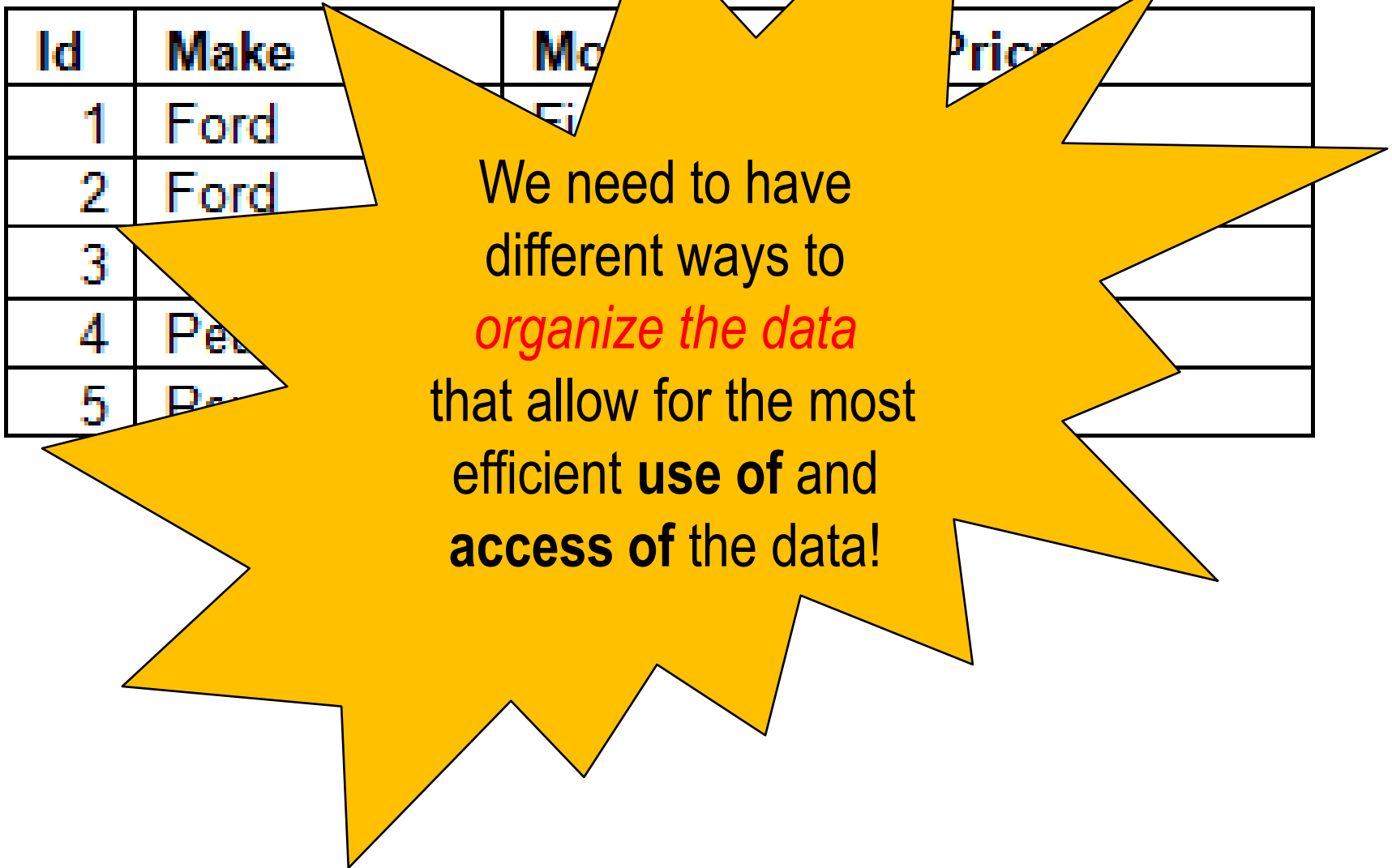
- We will study fundamental *data structures.*

Id	Make	Model	Price
1	Ford	Fiesta	*4000*
2	Ford	Focus	3000
3	VW	Golf	5000
4	Peugeot	206	*6000*
5	Peugeot	307	5500

Are we interested in the make of the vehicle? How many models per make? The most affordable vehicle?

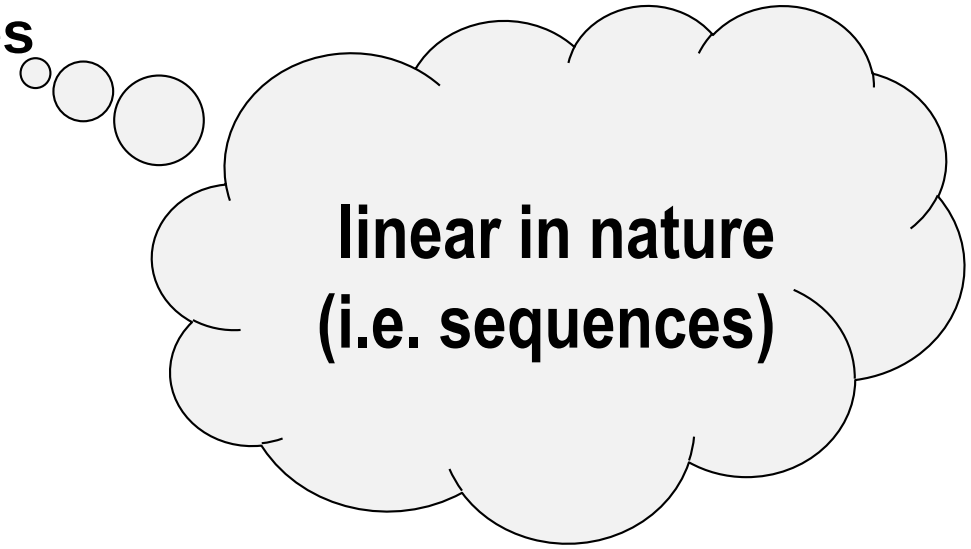
Welcome to Computer Science 112!

- We will study fundamental *data structure*



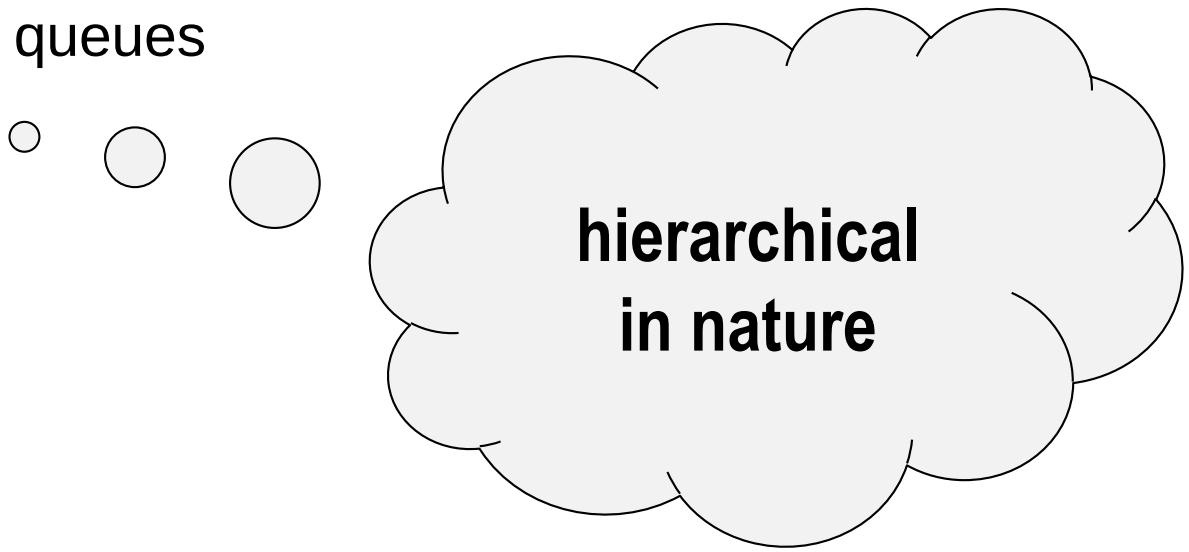
Welcome to Computer Science 112!

- We will study fundamental *data structures.*
  - ways of imposing order on a collection of information
  - **lists**, **stacks**, and **queues**
  - trees
  - hash tables



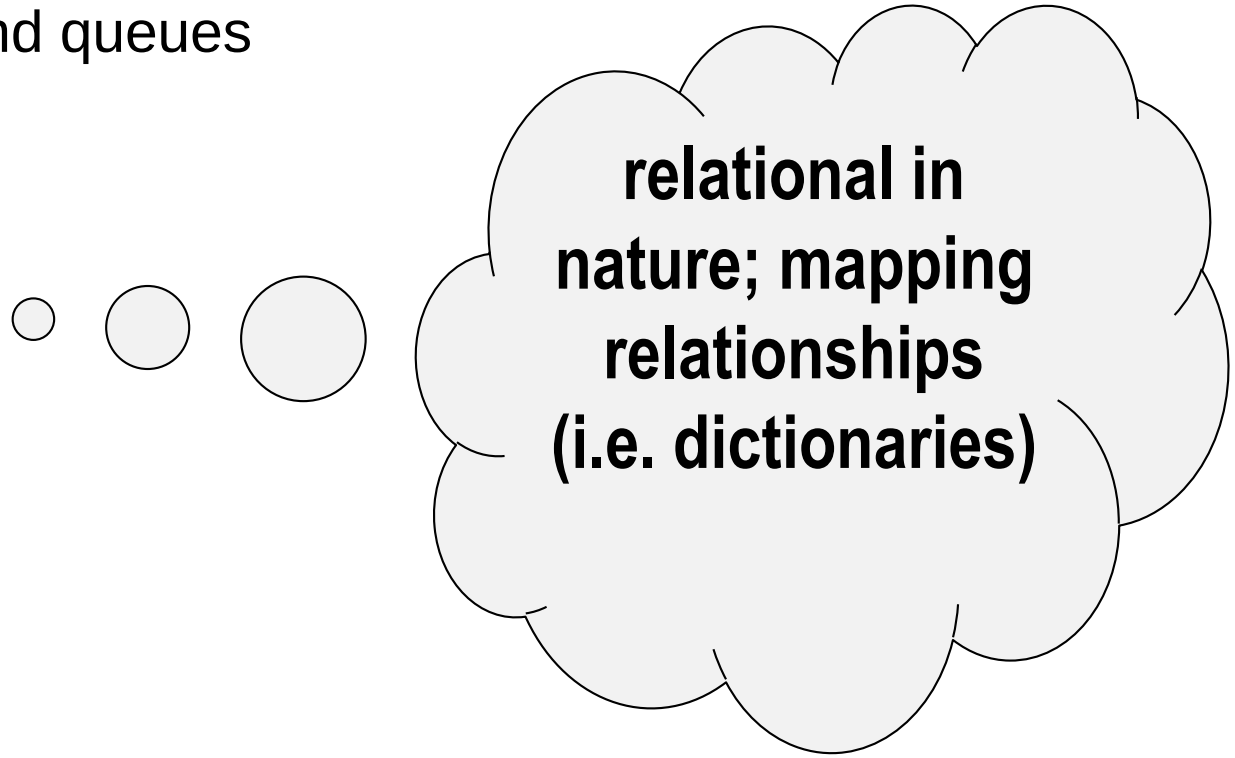
Welcome to Computer Science 112!

- We will study fundamental *data structures.*
  - ways of imposing order on a collection of information
  - lists, stacks, and queues
  - **trees**
  - hash tables



Welcome to Computer Science 112!

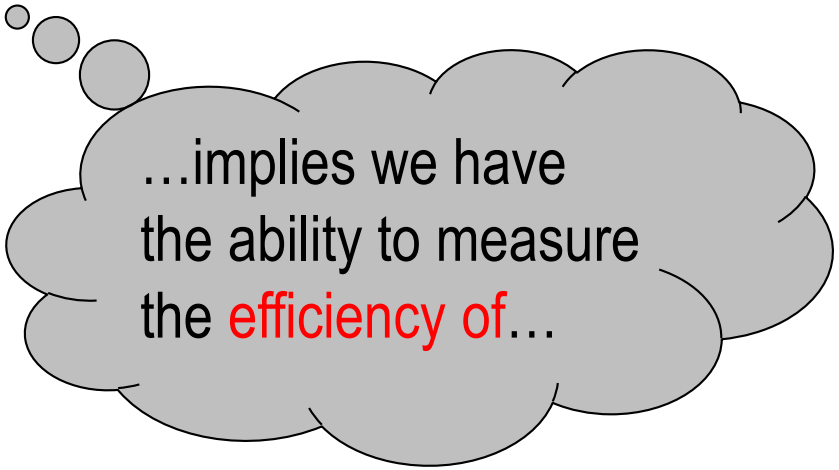
- We will study fundamental *data structures.*
  - ways of imposing order on a collection of information
  - lists, stacks, and queues
  - trees
  - **hash tables**



Welcome to Computer Science 112!

- We will study fundamental *data structures.*
  - ways of imposing order on a collection of information
  - sequences (linear): lists, stacks, and queues
  - trees (hierarchical)
  - hash tables (relational)

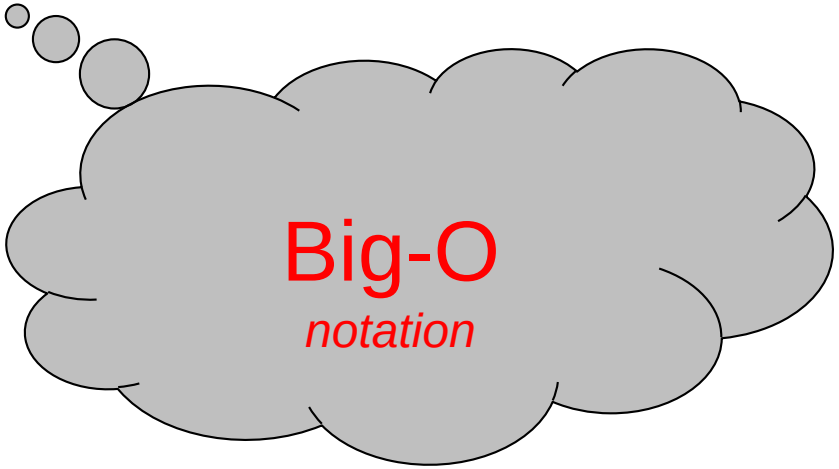
- We will also:
  - study *algorithms* related to these data structures
  - learn how to ***compare*** data structures & algorithms



Welcome to Computer Science 112!

- We will study fundamental *data structures.*
  - ways of imposing order on a collection of information
  - sequences (linear): lists, stacks, and queues
  - trees (hierarchical)
  - hash tables (relational)

- We will also:
  - study *algorithms* related to these data structures
  - learn how to ***compare*** data structures & algorithms



Welcome to Computer Science 112!

- We will study fundamental *data structures.*
  - ways of imposing order on a collection of information
  - sequences (linear): lists, stacks, and queues
  - trees (hierarchical)
  - hash tables (relational)

- We will also:
  - study *algorithms* related to these data structures
  - learn how to *compare* data structures & algorithms

- Goals:
  - learn to think more intelligently about programming problems
  - acquire a set of useful tools and techniques

- We will use the Java programming language.
  - but learning Java is ***not*** the primary focus of the course!

Prerequisites

- CS 111, or the equivalent
  - ideally with a B- or better
  - if not CS 111, solid coding skills in one of the following:
    - Python
    - Java
    - C++
  - comfortable with recursion
  - some exposure to object-oriented programming

- Reasonable comfort level with mathematical reasoning
  - mostly simple algebra, but need to understand the basics of logarithms (we'll review this)
  - we'll do some simple proofs

Course Materials

- Enthusiasm

Course Materials

- Enthusiasm

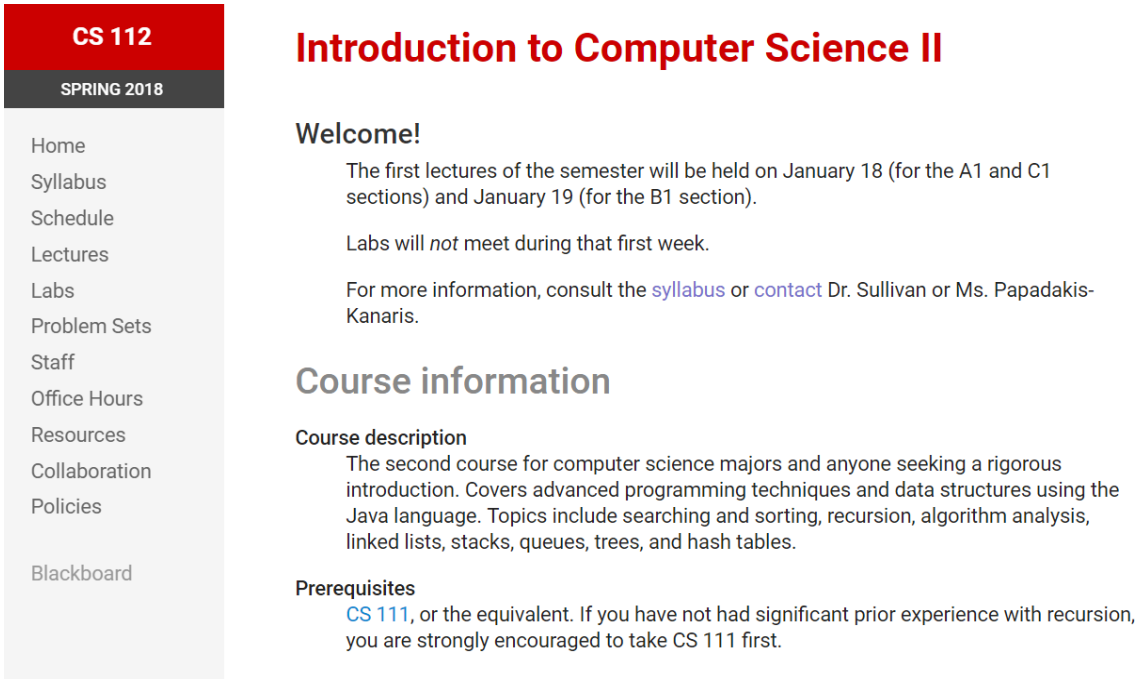
- Also bring to every lecture:
  - a notebook (loose-leaf or spiral)
  - a pen or pencil

Course Materials

- Enthusiasm

- Also bring to every lecture:
  - a notebook (loose-leaf or spiral)
  - a pen or pencil

Course Website
www.cs.bu.edu/courses/cs112



CS 112

SPRING 2018

Home
Syllabus
Schedule
Lectures
Labs
Problem Sets
Staff
Office Hours
Resources
Collaboration
Policies

Blackboard

Introduction to Computer Science II

Welcome!

The first lectures of the semester will be held on January 18 (for the A1 and C1 sections) and January 19 (for the B1 section).

Labs will *not* meet during that first week.

For more information, consult the syllabus or contact Dr. Sullivan or Ms. Papadakis-Kanaris.

Course information

Course description

The second course for computer science majors and anyone seeking a rigorous introduction. Covers advanced programming techniques and data structures using the Java language. Topics include searching and sorting, recursion, algorithm analysis, linked lists, stacks, queues, trees, and hash tables.

Prerequisites

CS 111, or the equivalent. If you have not had significant prior experience with recursion, you are strongly encouraged to take CS 111 first.

- *not* the same as the Blackboard site for the course
- use the Blackboard link to access:
  - the pre-lecture study materials } *posted by night before lecture*
  - the lecture notes – posted *after* lecture

Grading

1. Problem sets (25%)
2. Exams
   - midterm 1 & 2 (30%) - TBA
   - **final exam (40%) - TBA**

To pass the course, you must earn a passing grade on the final exam.

3. Labs, Preparation and participation (5%)
   - lecture preparation, participation and attendance
   - lab participation and attendance

Labs

- Attendance is mandatory **and begin this week**
- Will help you prepare for assignments
- Will reinforce essential skills
- **ASAP: Complete Lab 0**

Course Staff

- **Instructor(s):** Christine Papadakis-Kanaris, David Sullivan PhD
- **Teaching Fellows/Assistants (TF/TAs)**:
  - Peilun Dai
  - Xin Lu
  - Hao Yu
  - Ryan Yu
  - Ivan Izhbirdeev
- **Office hours:** http://www.cs.bu.edu/courses/cs112

Getting Started with Java

- You all have a solid foundation in a programming language.
- For most of you, that language (Python) is not the one that we'll use. Java is!
- We'll cover some of the trickier aspects of Java together
  - in lecture
  - in the labs
- You'll need to learn the rest on your own.
  - something you will do many times in your career!
  - an important skill in its own right

Fundamental Question in Computer Science

- Are computers intelligent?
- A computer is just a **black box**

A (binary) device that responds to two types of signals

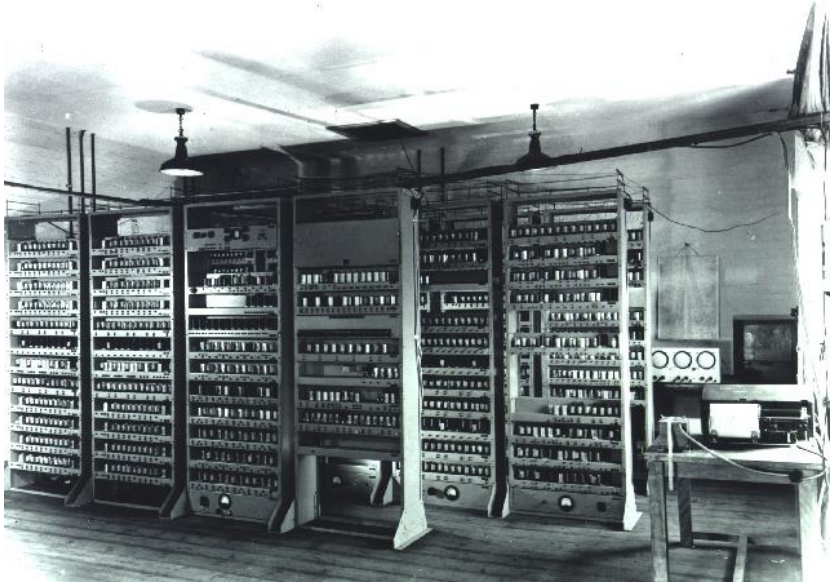
on and off!

*It is the layers of **software** executing on a computer that make computers interesting and give the illusion of intelligence!*

Importance of Software in the

Technology (R)evolution

Cost $
and
Size of



Hardware

Software

..... **1930's** 1950's 1980's 2000's 2019.. **time**

Importance of Software in the

Technology (R)evolution

Cost $
and
Size of

Hardware

Software

..... 1930's **1950's** 1980's 2000's 2019… **time**

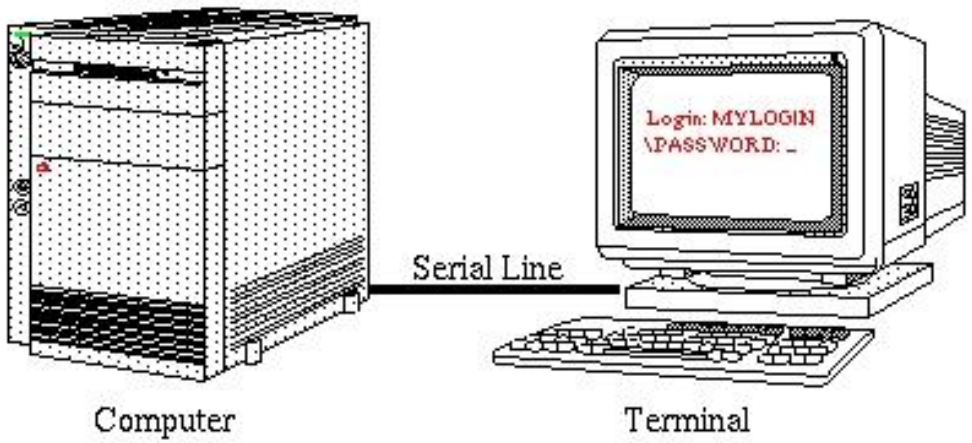
Birth of Modern computing

Importance of Software in the
Technology (R)evolution

Cost $
and
Size of

Hardware

Software



..... 1930's 1950's **1980's** 2000's 2019… **time**

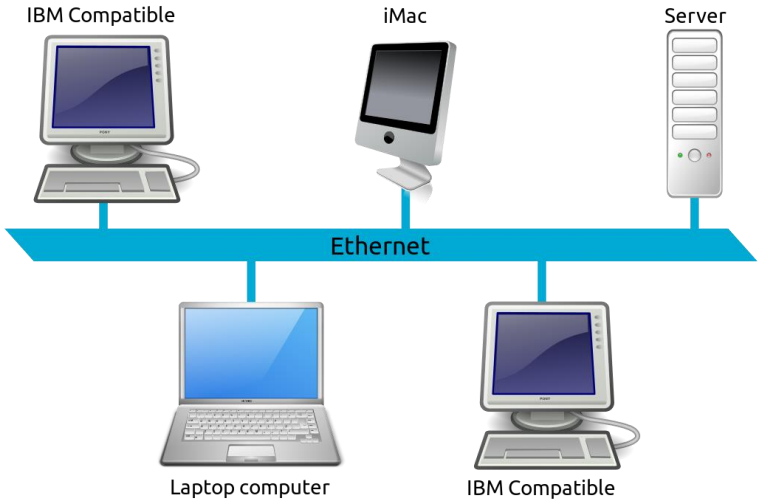
Dumb Terminals

Importance of Software in the
Technology (R)evolution

Cost $
and
Size of

Hardware

Software



..... 1930's 1950's **1980's** 2000's 2019… **time**

LAN/WAN

Importance of Software in the

Technology (R)evolution

Cost $
and
Size of

Hardware

Software

..... 1930's 1950's 1980's **2000's** 2019… **time**

Network computing &
World Wide WEB

Importance of Software in the
Technology (R)evolution

Cost $
and
Size of

Hardware

Software

..... 1930's 1950's 1980's 2000's 2019… **time**

Smart Devices

Importance of Software in the
Technology (R)evolution

Cost $
and
Size of

Hardware

Software

Nation States and
Large Corporations

Who could afford to own
purchase computers?

..... 1930's 1950's 1980's 2000's 2019… **time**

Importance of Software in the
Technology (R)evolution

Cost $
and
Size of

Hardware

Software

States and

Who could afford to own
purchase computers?

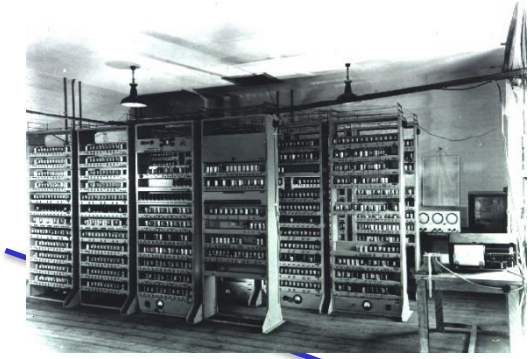
Its all about the
App!

..... 1930's 1950's 1980's 2000's 2019… **time**

Importance of Software in the

Technology (R)evolution

Cost $
and
Size of



Hardware

Software

..... | 1930's | 1950's | 1980's | 2000's | 2019… | **time**

Importance of Software in the
Technology (R)evolution

Cost $
and
Size of

Hardware

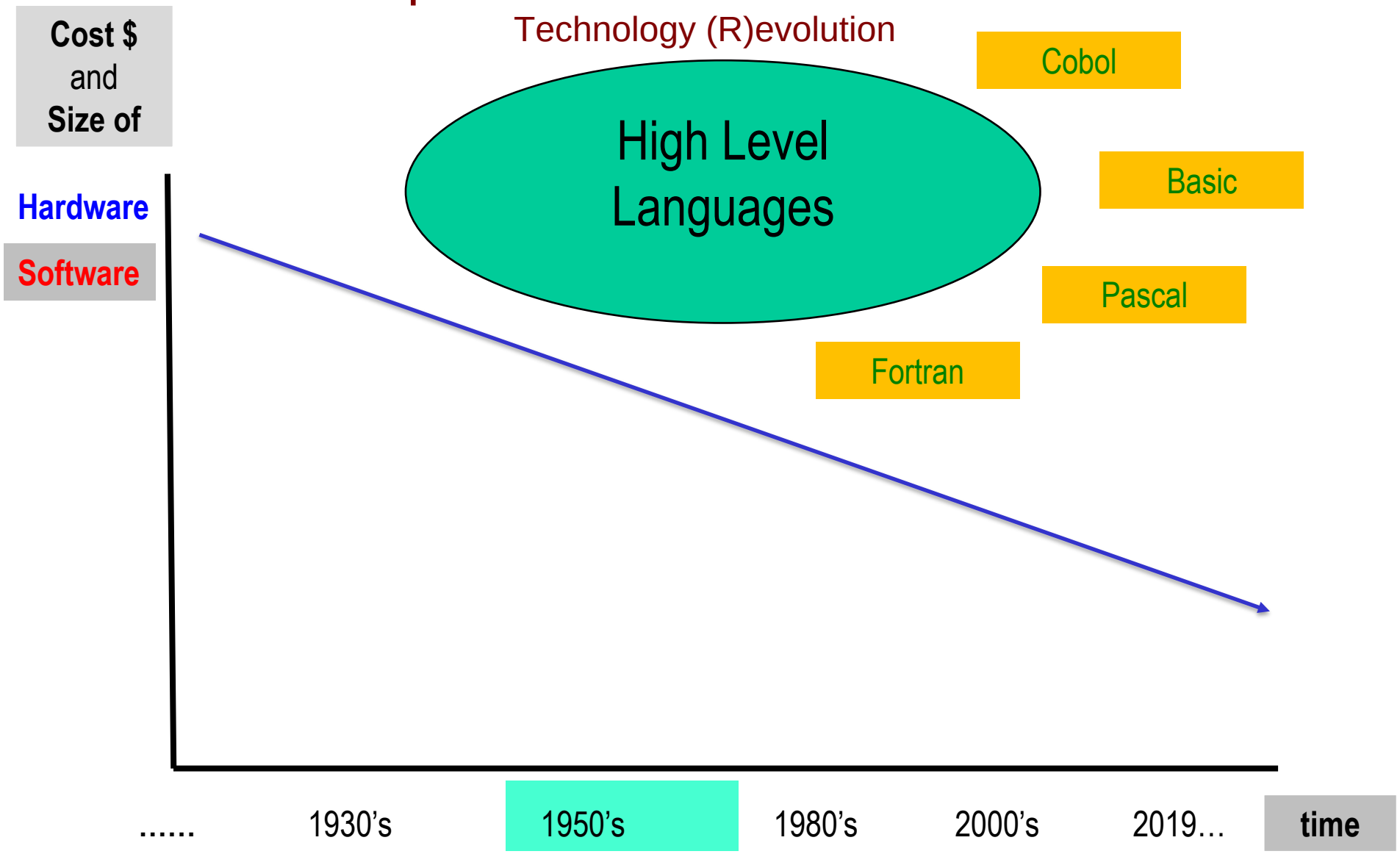
Software

Machine code	Assembly code	Description
001 1 000010	LOAD #2	Load the value 2 into the Accumulator
010 0 001101	STORE 13	Store the value of the Accumulator in memory location 13
001 1 000101	LOAD #5	Load the value 5 into the Accumulator
010 0 001110	STORE 14	Store the value of the Accumulator in memory location 14
001 0 001101	LOAD 13	Load the value of memory location 13 into the Accumulator
011 0 001110	ADD 14	Add the value of memory location 14 to the Accumulator
010 0 001111	STORE 15	Store the value of the Accumulator in memory location 15
111 0 000000	HALT	Stop execution

..... 1930's 1950's 1980's 2000's 2019… **time**

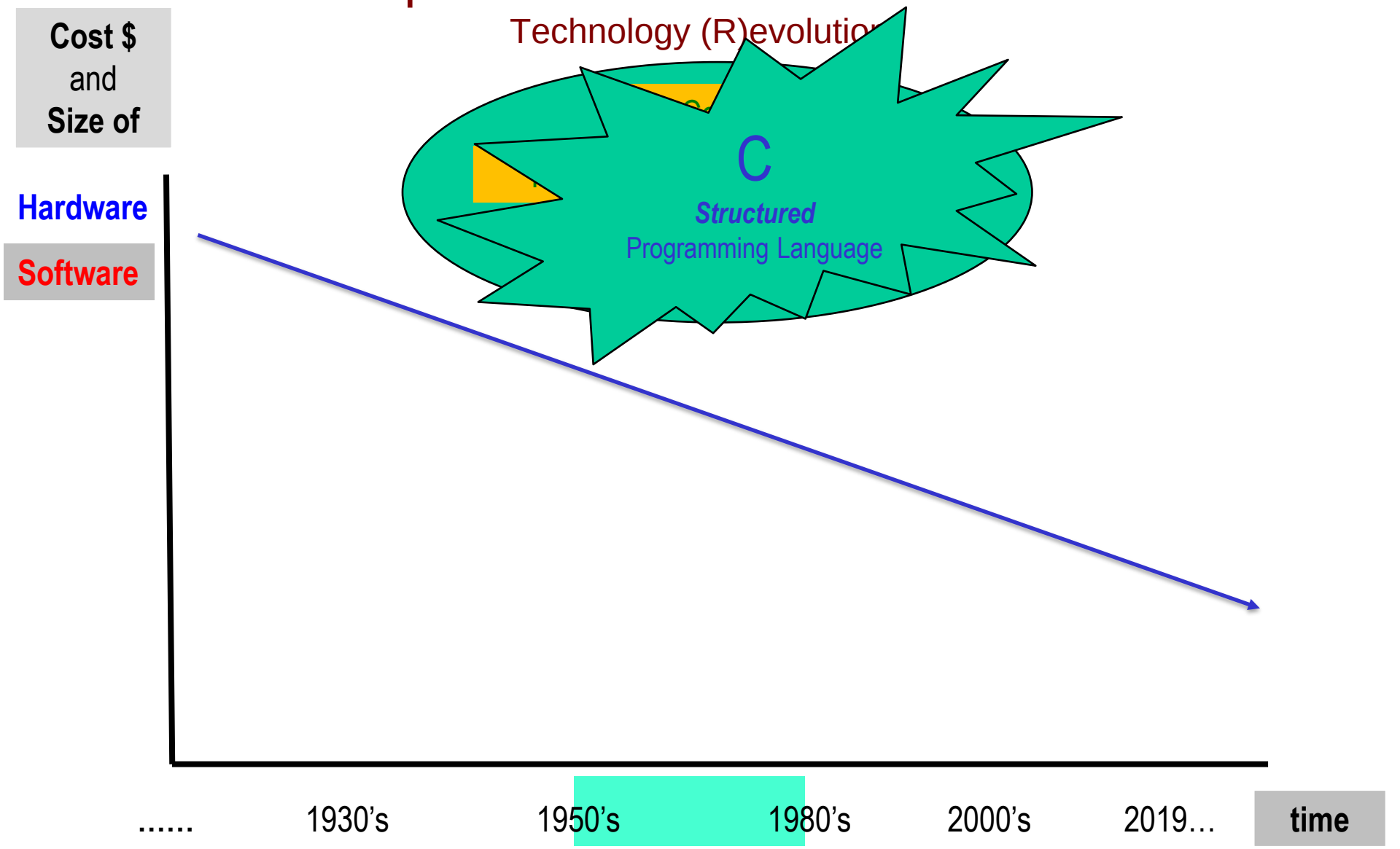
Importance of Software in the

Technology (R)evolution

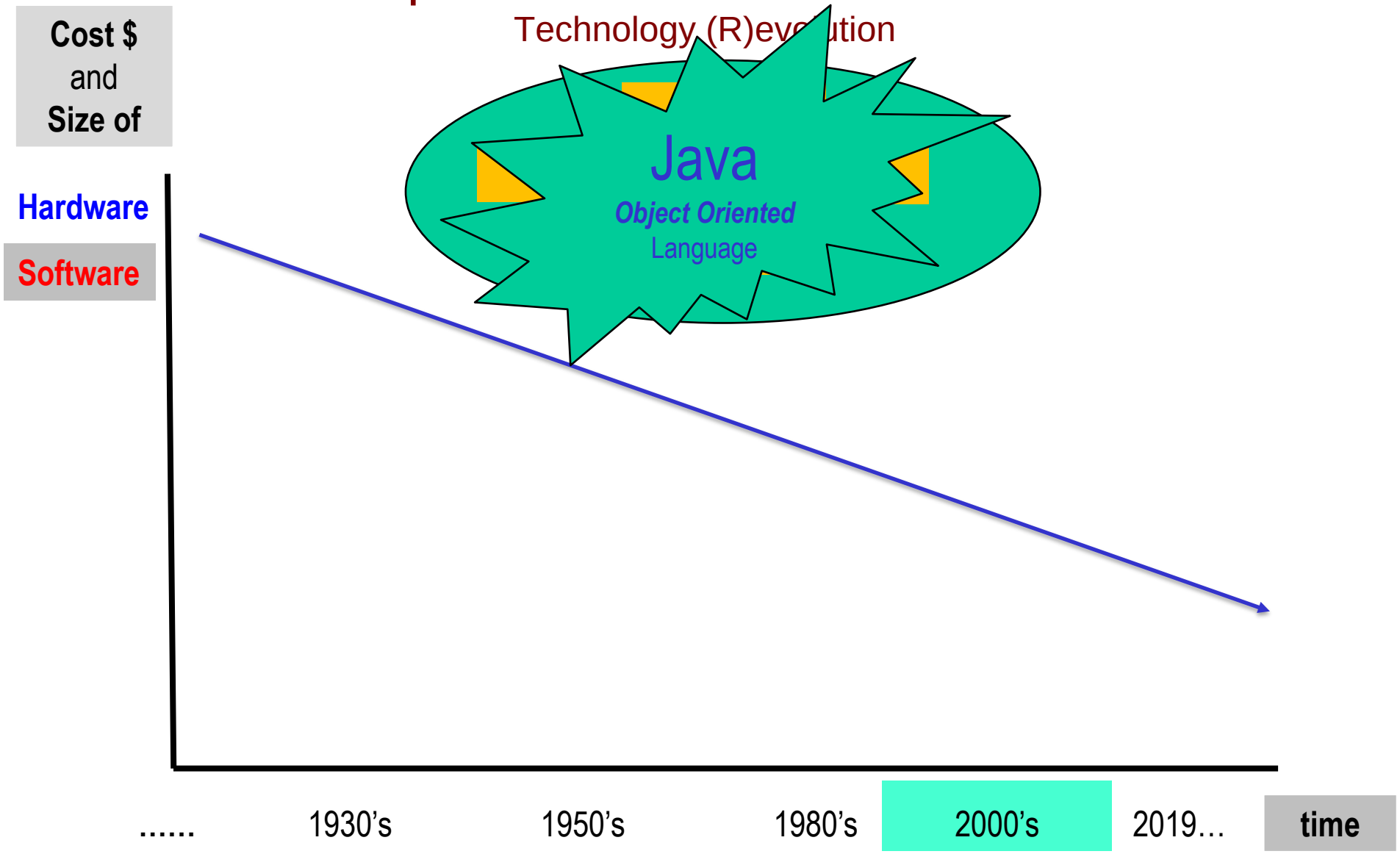


Importance of Software in the

Technology (R)evolutio

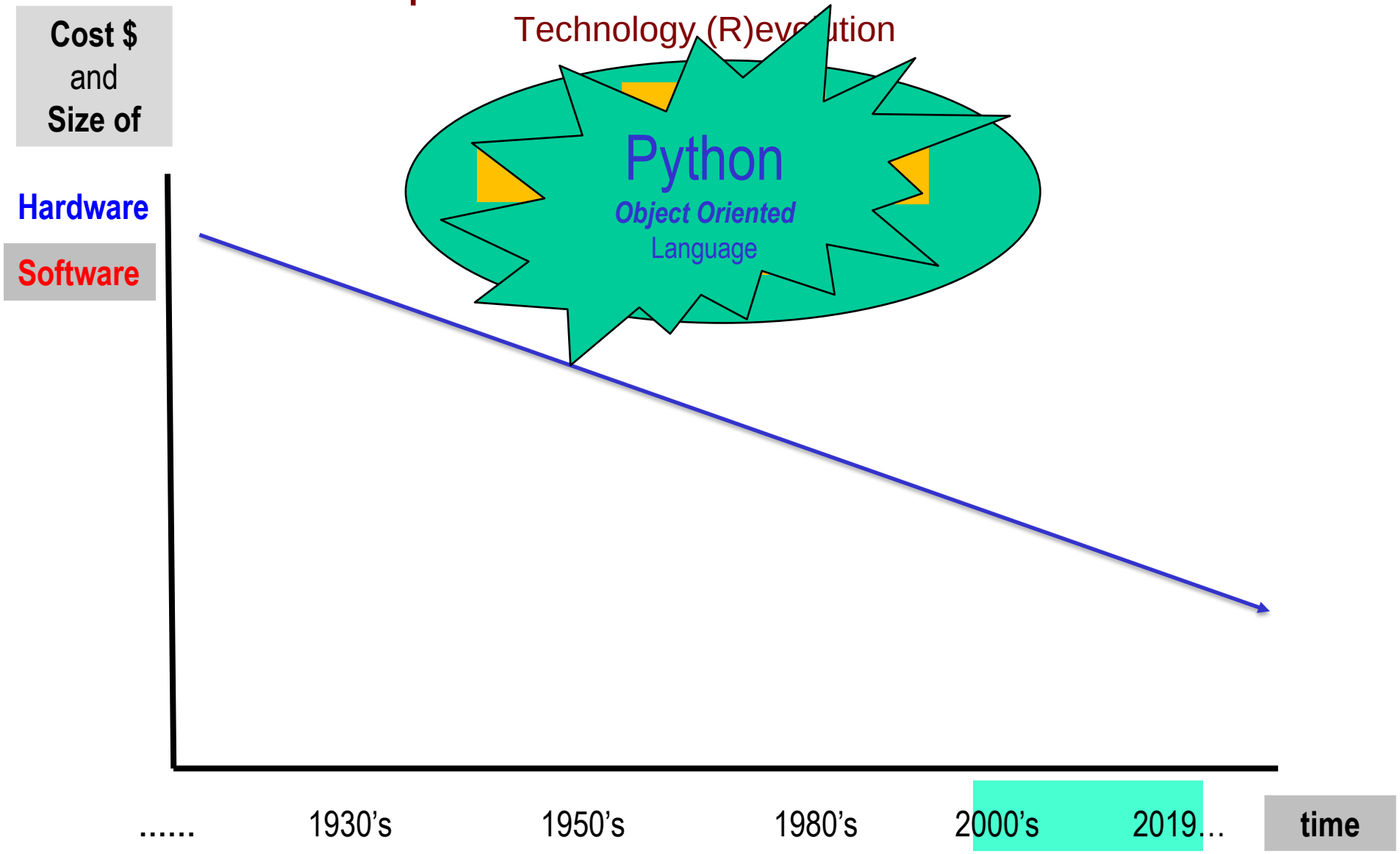


Importance of Software in the Technology (R)evolution



Importance of Software in the

Technology (R)evolution



Importance of Software in the

Technology (R)evolution

Cost $
and
Size of

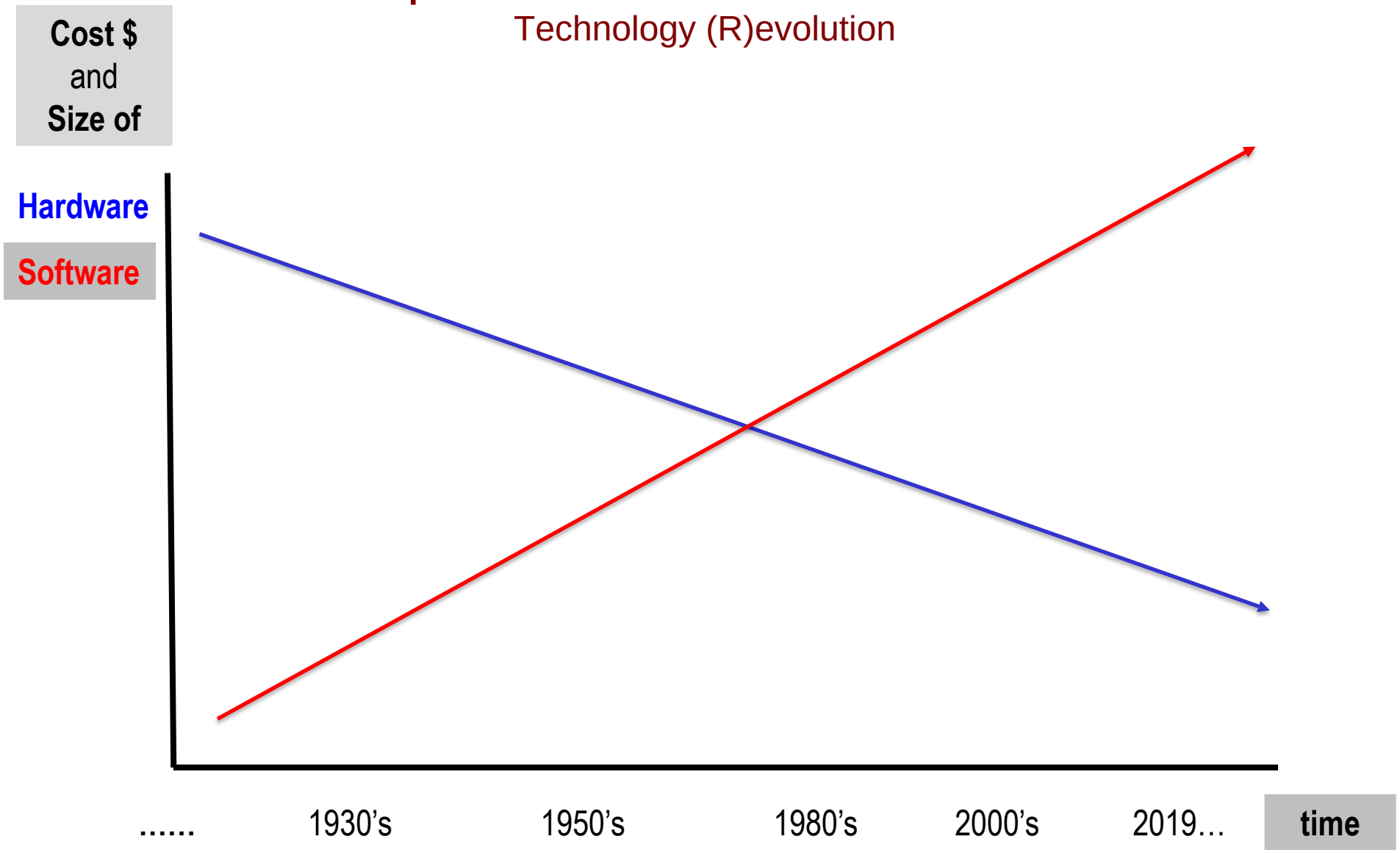
Hardware

Software

..... 1930's 1950's 1980's 2000's 2019… **time**

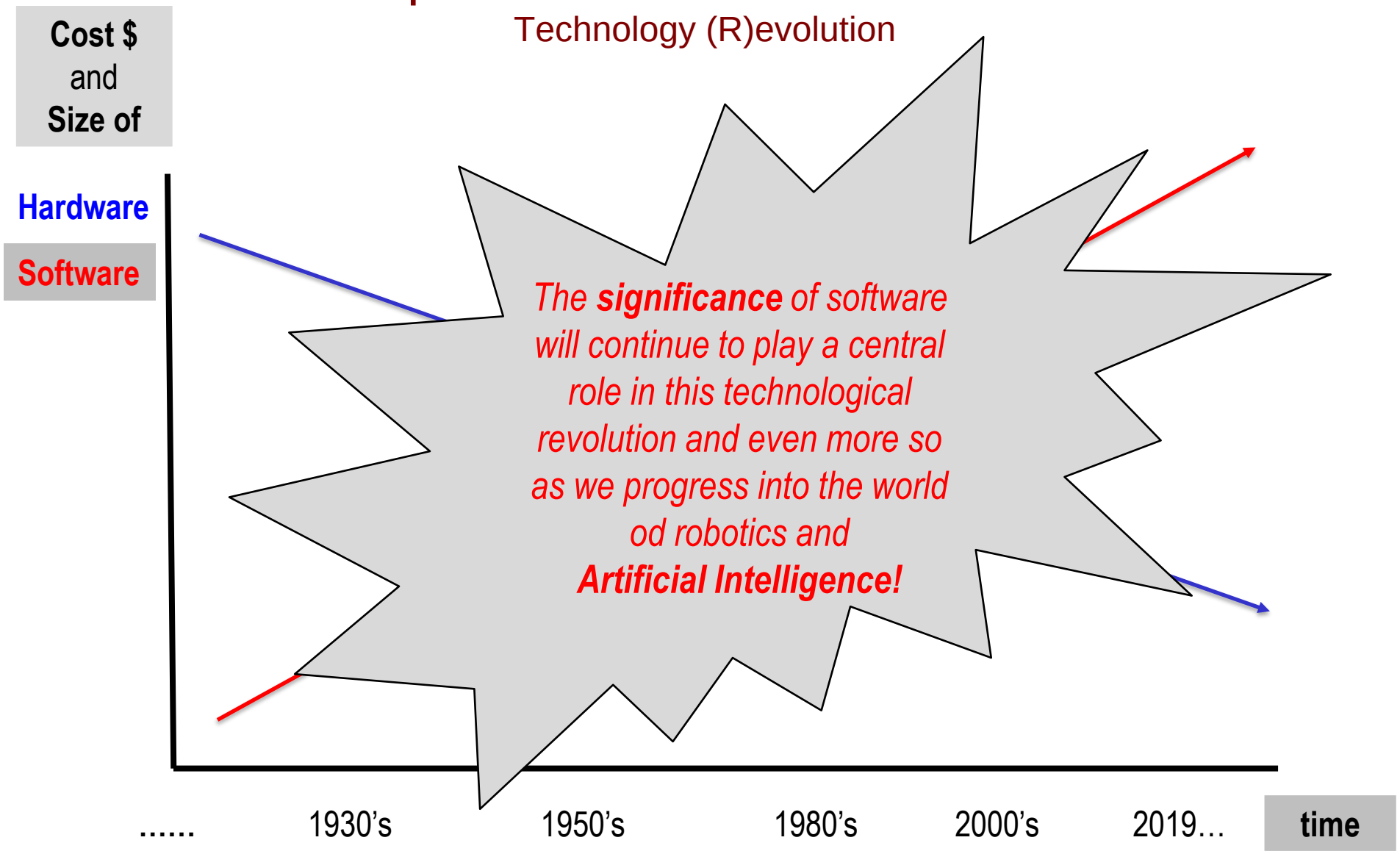
Importance of Software in the

Technology (R)evolution



Importance of Software in the

Technology (R)evolution

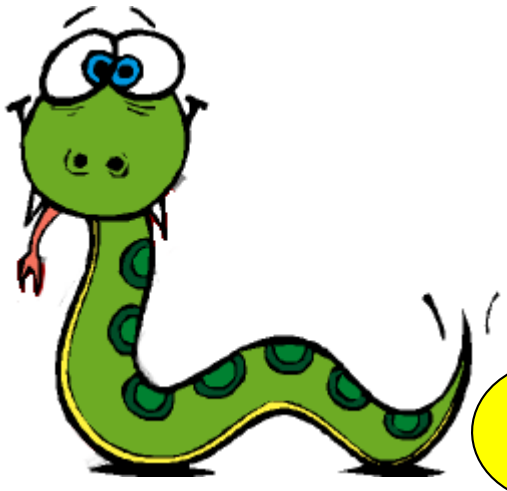


Software and Intelligence:
Program Intelligence

- Senses ⇨ Input/Ouput (I/O)
- Memory
  - Short term ⇨ Data Types and Variables, **and Data structures**
  - Long term ⇨ Files
- Reason or Logic ⇨ Conditional Statements
  - We use reason and logic to make decisions
- Repetition ⇨ Loops
  - Once we learn to do something, we can do it over and over!
- Optimization ⇨ Functions, OO Programming
  - Once we learn how to do something, we try to do it better!

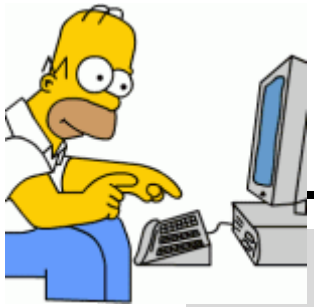
Python, Java Compare/Contrast Language Overview

Interpreted language implementation.



Compiled language implementation, *sort of*...

Compiled vs. Interpreted



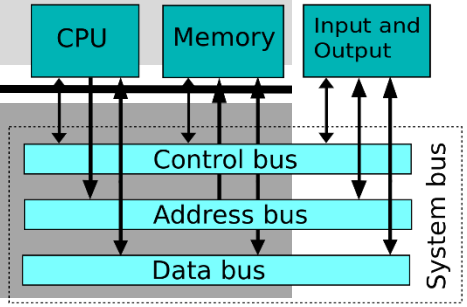
Programmer

High Level Languages

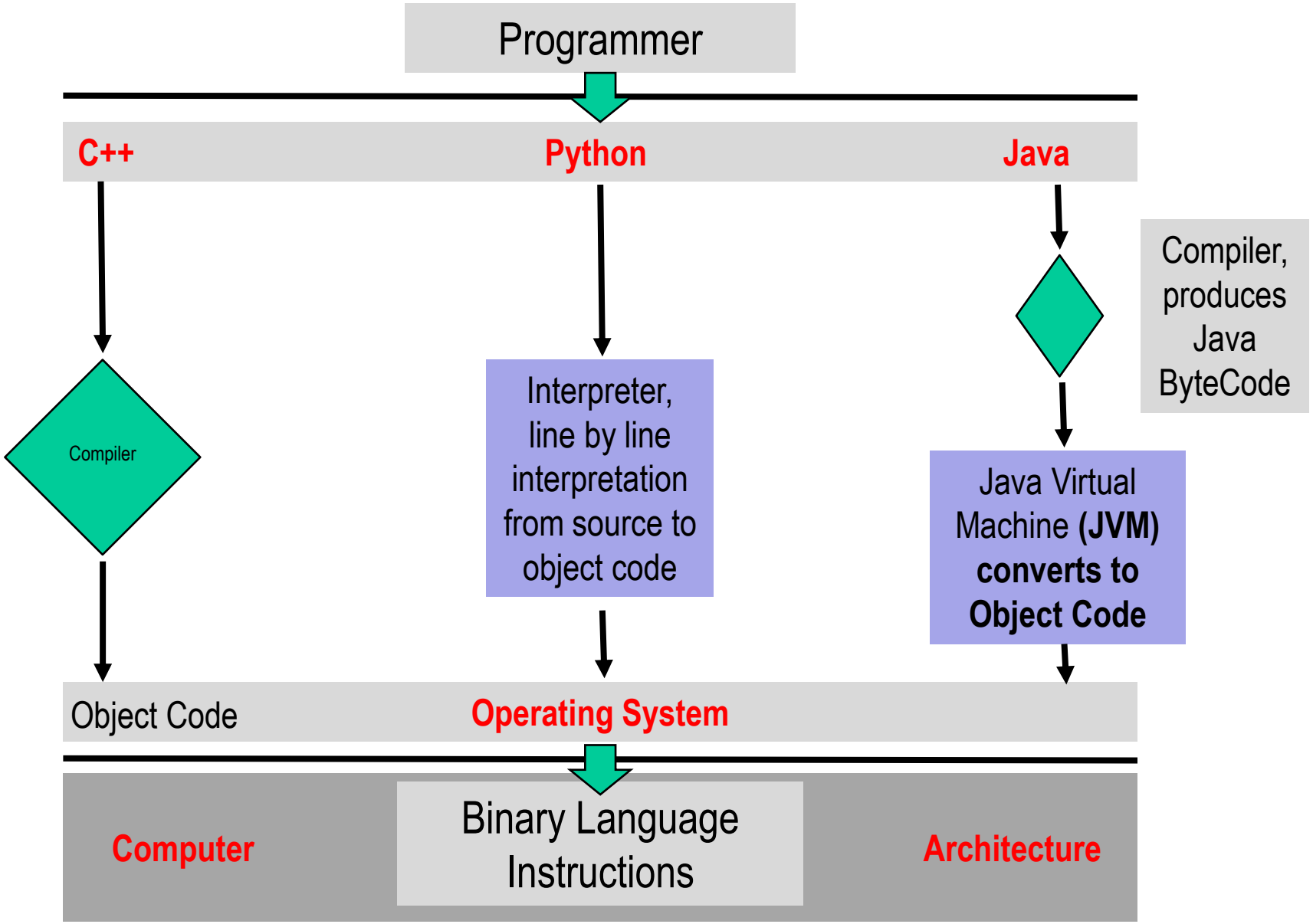
Operating System

Computer Architecture

Binary Language

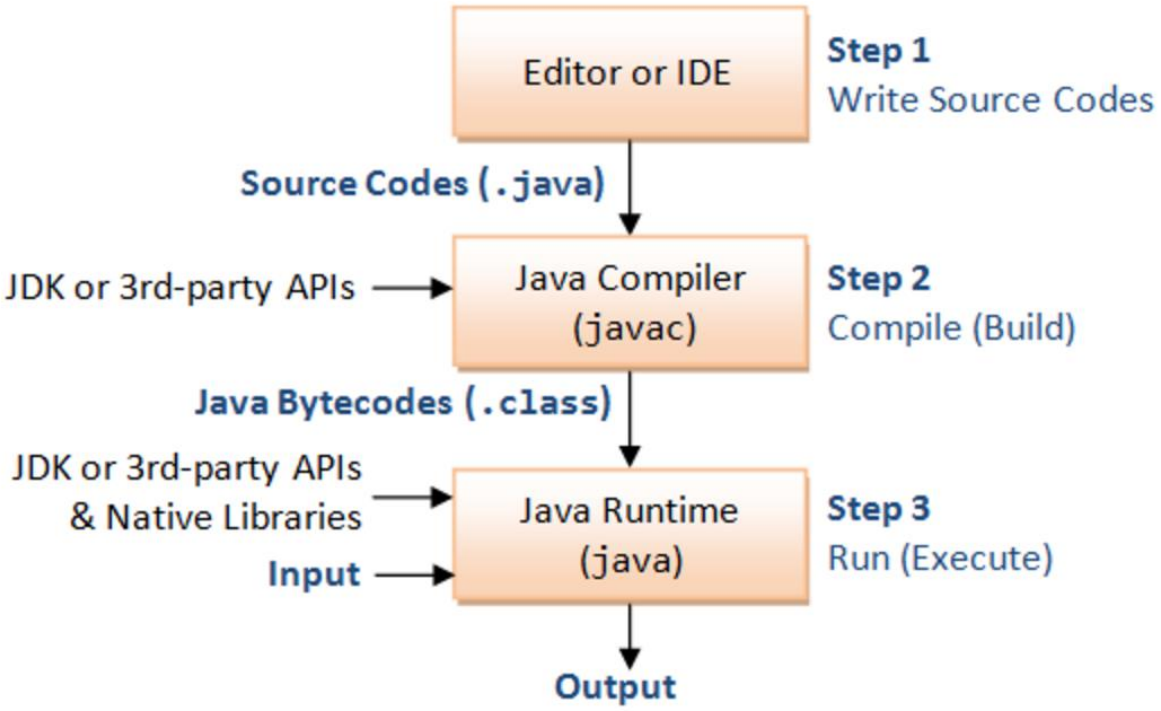


Compiled vs. Interpreted



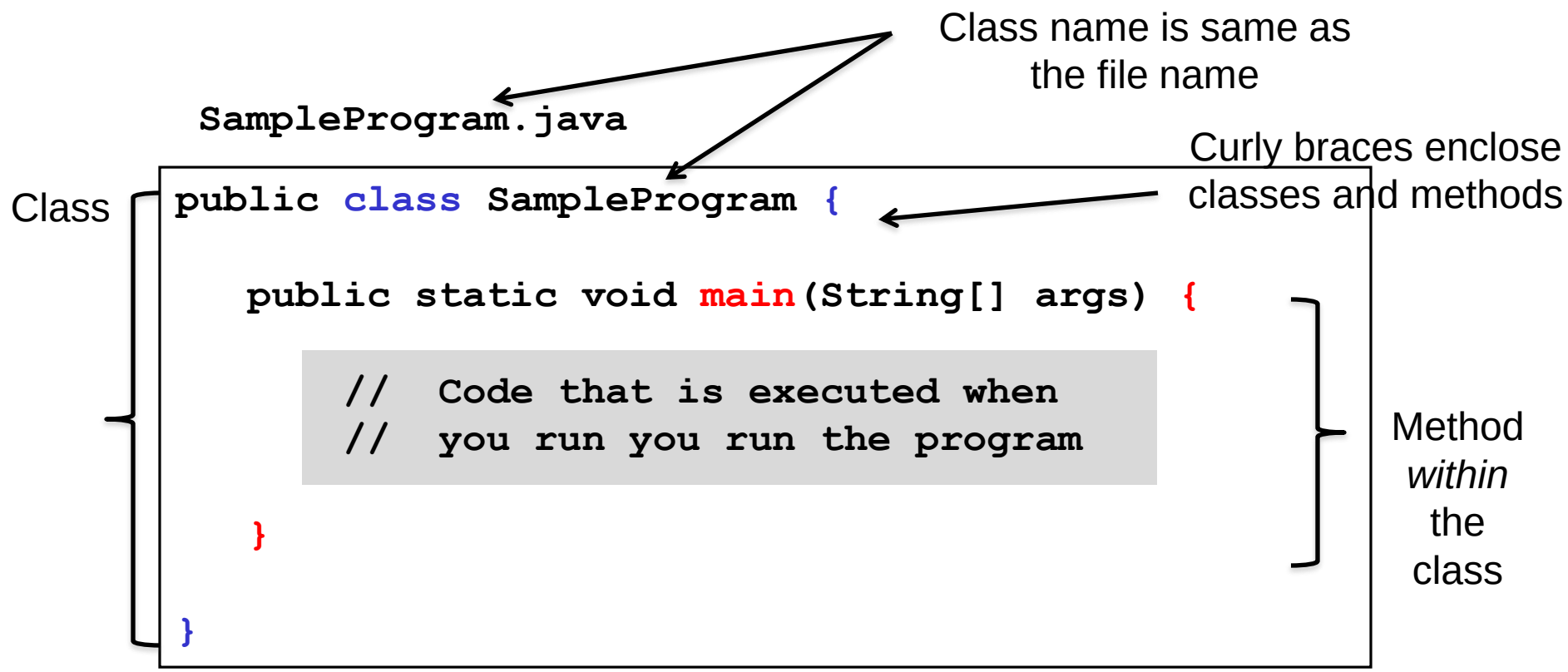
Compilation vs Interpretation

- Java is an example of a language which is compiled; before executing any code, your program must be transformed into a lower-level code called byte-code. The byte-code is then passed to the Java Virtual Machine (JVM), which runs the program and produces output:



Java Basic Program Structure

- Java programs are organized as classes stored in files with the ".java" extension, and with code written inside methods delimited by curly braces; each program must have a method called main, which contains the code that will be executed when you run your program:



Java Comments

Java comments: **block** and **line**

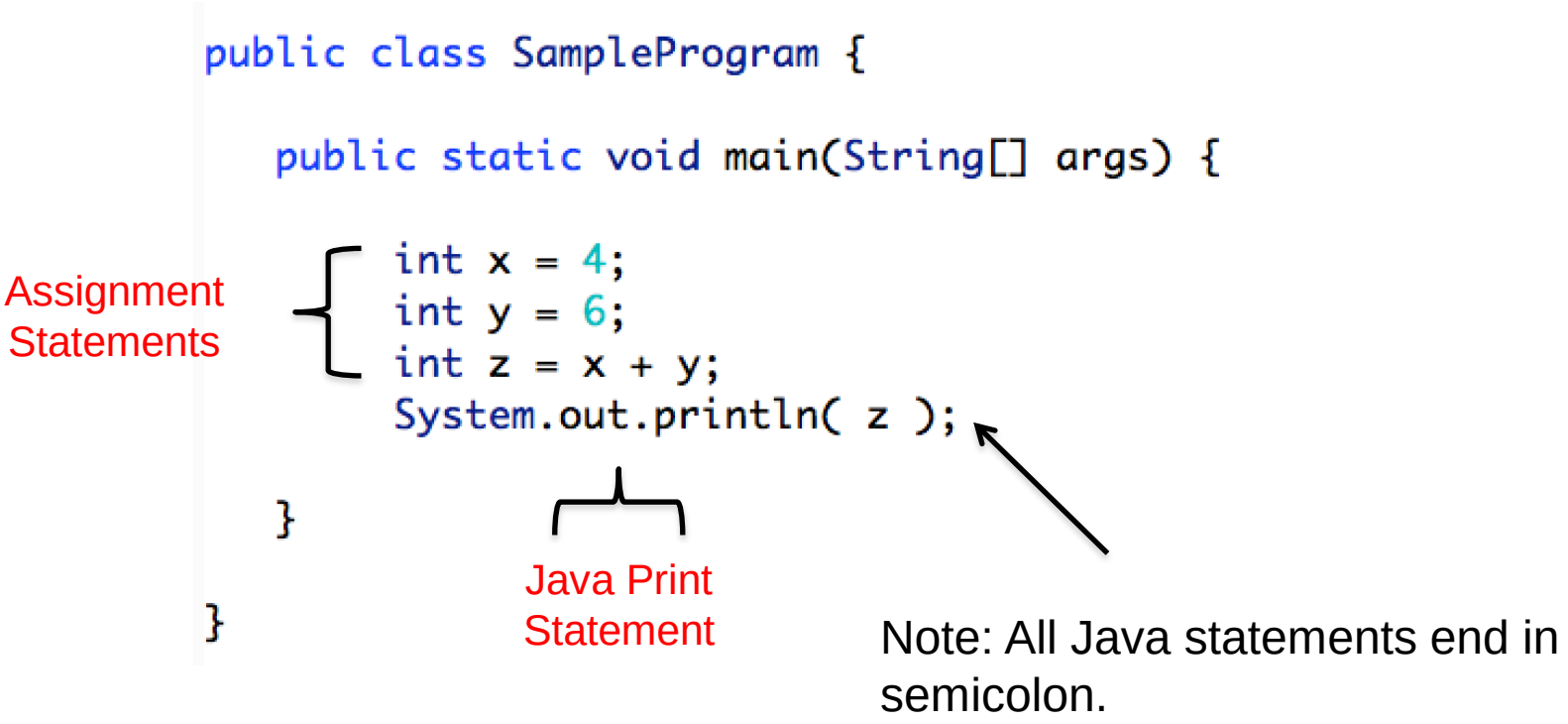
/* …. */ and //

```
/*
   File: Statistics.java
   Author: Wayne Snyder
   Date: January 23rd, 2015
   Purpose: This is a solution for
 */

// The following is a library whic
// reading input from the user in
// libraries (such as Math) are al
// as Scanner) you need to explici
// must occur before your class de
```

Java Statements

- In Java, we compute by executing statements, which have an effect on the state of the program: they assign a value to a variable, or print a value out, or otherwise change something in the system.



Java Statements

- It is often useful to understand the effect of a sequence of assignment statements by *tracing the values of the variables*, which change after each statement. The collection of all values is the state of the program:

	a	b	t
`int a, b;`	*undefined*	*undefined*	
`a = 1234;`	1234	*undefined*	
`b = 99;`	1234	99	
`int t = a;`	1234	99	1234
`a = b;`	99	99	1234
`b = t;`	99	1234	1234

Java Statements

- It is often useful to understand the effect
statements by *tracing the values of the va*
statement. The collection of all values is t

Often referred to as a
paper and pencil
trace of your program.

	a	b	t
`int a, b;`	*undefined*	*undefined*	
`a = 1234;`	1234	*undefined*	
`b = 99;`	1234	99	
`int t = a;`	1234	99	1234
`a = b;`	99	99	1234
`b = t;`	99	1234	1234

Java Values and Types

- A Data Type (or just Type) is a collection of values and associated operations.
- Java is a Strongly-Typed language supporting many different *types of data*:

Java Primitive Data Types				
Type	**Values**	**Default**	**Size**	**Range**
byte	signed integers	0	8 bits	-128 to 127
short	signed integers	0	16 bits	-32768 to 32767
int	signed integers	0	32 bits	-2147483648 to 2147483647
long	signed integers	0	64 bits	-9223372036854775808 to 9223372036854775807
float	IEEE 754 floating point	0.0	32 bits	+/-1.4E-45 to +/-3.4028235E+38, +/-infinity, +/-0, NAN
double	IEEE 754 floating point	0.0	64 bits	+/-4.9E-324 to +/-1.7976931348623157E+308, +/-infinity, +/-0, NaN
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF
boolean	`true, false`	`false`	1 bit used in 32 bit integer	NA

String “hi there” “”

Java Values and Types

In CS 112 we will only use the following types:

type	set of values	literal values	operations
`char`	characters	`'A'` `'@'`	compare
`String`	sequences of characters	`"Hello World"` `"126 is fun"`	concatenate
`int`	integers	`17` `12345`	add, subtract, multiply, divide
`double`	floating-point numbers	`3.1415` `6.022e23`	add, subtract, multiply, divide
`boolean`	truth values	`true` `false`	and, or, not

Java Values and Types

- **Python** is “weakly typed”: values have types but variables do not; variables are just names to reference any value you want and can be reused for other values; errors occur when variables which have not yet been assigned values are used:

```
In [123]: X = 5                          // assign an integer value to variable X

In [124]: X
Out[124]: 5

In [125]: X = 4.5                        // re-assign an floating point value to variable X

In [126]: X = "hi"                       // re-assign a string value to variable X

In [127]: X
Out[127]: 'hi'

In [128]: Z                              // variables only come into existence
Traceback (most recent call last):       // when an assignment is made

  File "<ipython-input-128-41ff0912a07f>", line 1, in <module>
    Z

NameError: name 'Z' is not defined
```

Java Values and Types

Java is strongly-typed in that

All variables must be declared with a type before being used and can then only be used for that type of value:

```
int x;                      // declare x to be int
x = 4;                      // assign value to x
System.out.print(x);
double y = 3.4;             // combine declaration and assignment
double z = x + y;
System.out.print(z);
```

Java Values and Types

- The philosophy of strongly-typed languages is that specifying types makes programmers more careful about variables, and bugs and errors can be found during compilation, not when the program is running.
- Values can be converted from one type to another implicitly or explicitly:

Widening Conversions (implicit):

Narrow types (less information) ⟶ Wider types (more information)

Example: `int` ⟶ `double`

```
double  x;
x  = 4;   // 4 is widened to 4.0 and then assigned
```

No error!

Example 2: `char` ⟶ `int`

```
int  x;
x  = 'a';   // 'a' is converted to its Unicode
            //  value 65 and assigned to x
```

Java Values and Types

- The philosophy of strongly-typed languages is that specifying types makes programmers more careful about variables, and bugs and errors can be found during compilation, not when the program is running.
- Values can be converted from one type to another implicitly or explicitly:

Narrowing Conversions (you must specify a cast or else get an error):

Wider types (more information) ⟶ Narrower types (less information)

Example: double ⟶ int

```
int  x;
x  = 4.5;
```

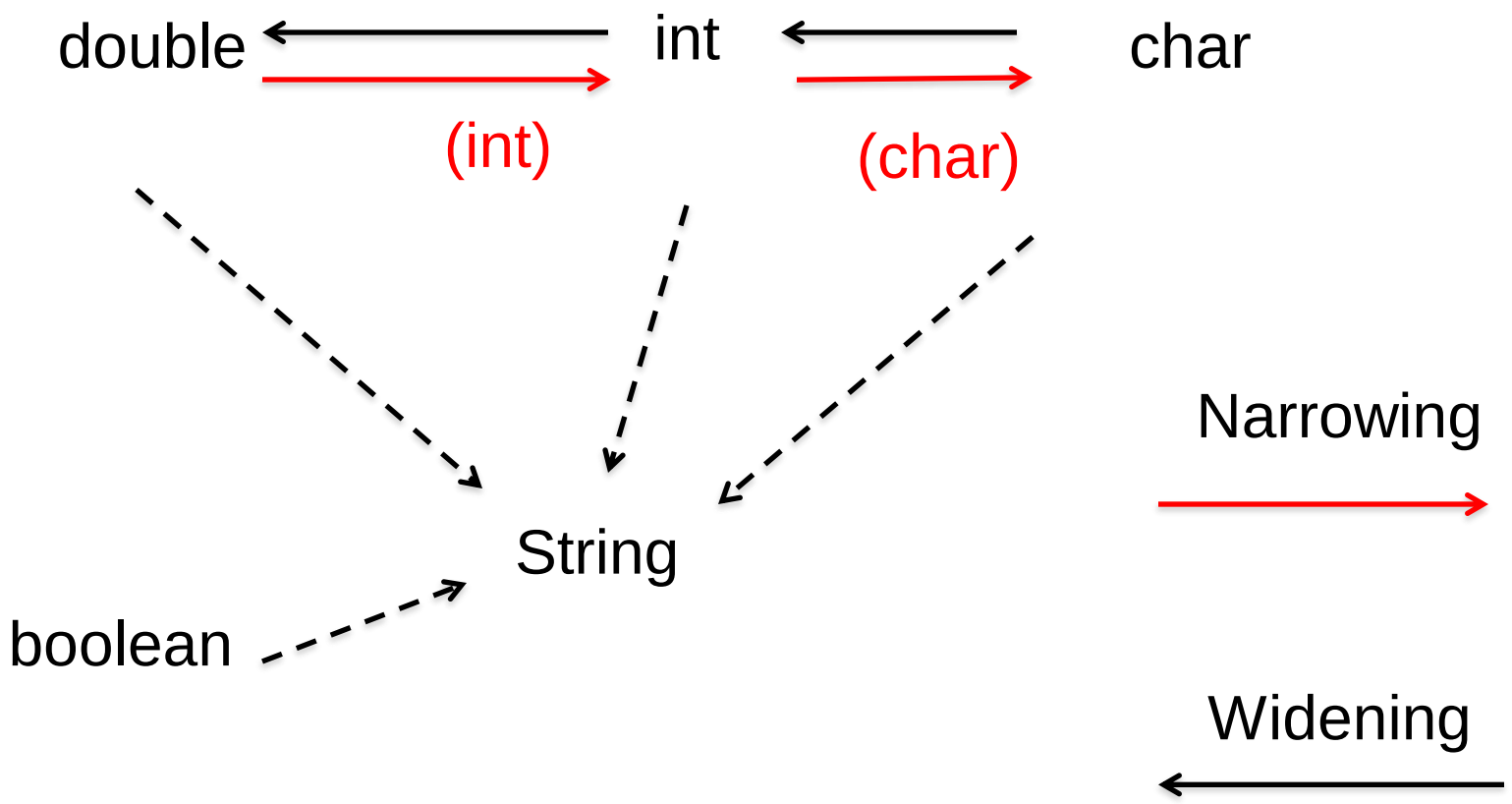
Error!

Must explicitly tell Java to truncate the double value to an integer:

```
int   x;
x   =  (int)  4.5;      // x assigned 4
```

Cast

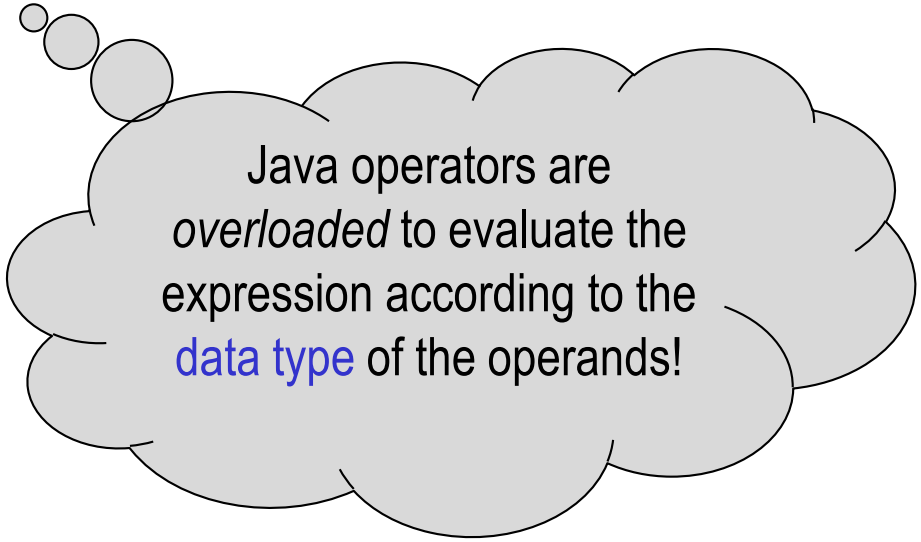
Java Values and Types



Java Operators

The operators are almost exactly the same as in Python:

+	addition	+=
-	subtraction	-=
*	multiplication	*=
%	modulus	%=
==	equals	
!=	not equal	
<	less	
<=	less or equal	
>	greater	
>=	greater or equal	



Java Operators

- When Java evaluates an overloaded operator, it automatically performs *widening* conversions as necessary to *make the operands fit the operator*:

Example: + is overloaded – it works for two ints or two doubles….

```
public static void main(String[] args) {

    int x = 4;

    double y = 2.3;

    System.out.println( ( x + x ) );     // prints:   8

    System.out.println( ( y + y ) );     // prints:   4.6

    System.out.println( ( x + y ) );     // prints:   6.3

}
```

All the arithmetic operators in java are overloaded for int and double.

Widening Conversion:

4 + 2.3

4.0 + 2.3 => 6.3

Result is the wider type!

Java Operators

Division is overloaded, therefore behaves differently for ints and doubles…..

Java: division operator is “overloaded”:

/ returns an int if both operands are ints, otherwise returns double:

5 / 2 => 2

5.0 / 2.0 => 2.5

5.0 / 2 => 2.5

5 / (double) 2 => 2.5

In both cases, the 2 is widened to 2.0!

Java Operators

The logical operators in Java look different (although they work exactly the same):

Python:	Java:
not	!
and	&&
or	||

Note that in both languages, and *and* or are lazy:

(false && X) => false (without evaluating X)
(true || X) => true (without evaluating X)

Example:

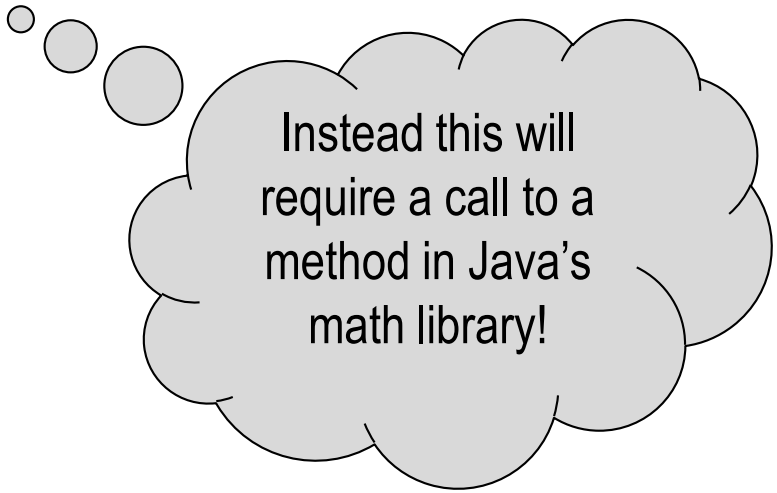
```
( (4 < 6) && ( 5 >= 5) )  =>  true      // both < and >= are evaluated
( (7 < 6) && ( 5 >= 5) )  =>  false     // only < needs to be evaluated
```

Java Operators

There is NO exponentiation operator in Java:

Python:

x ** 2 x squared



Java Operators

- Java has several useful increment and decrement operators which Python lacks; these can be used as statements OR expressions:

Statements:

```
++x;    x++ ;        // same as x = x + 1     or     x += 1
--x;    x-- ;        //  same as x = x – 1    or     x -=  1
```

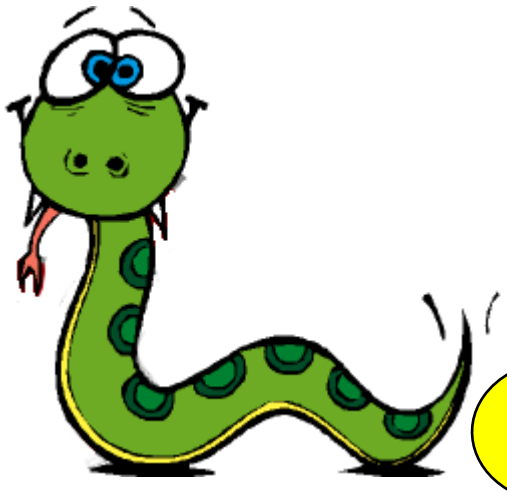
Expressions:

```
++x        has the value AFTER adding 1
x++        has the value BEFORE adding 1
```

	x	y	z
int x = 4;	4	undef	undef
int y = ++x;	5	5	undef
int z = x++ ;	6	5	5

Python, Java Compare/Contrast Language Overview

Interpreted language implementation.



Compiled language implementation, *sort of*...

For next class…

- Familiarize yourself with the Course Website.
- Make sure you have a good understanding of **course policies**, **requirements** and **expectations**.
- Review (and complete) Lab0.
- Make sure you have access to the course Blackboard page.
- Go through pre-lecture material for Thursday's class!