University at Buffalo
The State University of New York

cse@buffalo

CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

**Tuesday adjustments: 11:00 AM – 1:00 PM on 10/11, 11/1 and 12/6*

ANNOUNCEMENTS

ANNOUNCEMENTS

Undergraduate TA office hours will be ramped up to meet demand.

This week:

Kira - Tuesday at 5:00
Corwyn - Wednesday at 2:00
Steven - Thursday at 1:00

See “UTA Office Hours” table here:

www.cse.buffalo.edu/faculty/alphonce/cse115/people.php

(a very early) EXAM 1 NOTICE

DATE: Tuesday October 4

TIME: 8:45 PM – 9:45 PM

LOCATION: various rooms in NSC

specific room/seat assignments to come

COVERAGE:

lecture material up to and including 9/23 (this week)

lab material up to and including lab 3 (next week)

readings: all assigned up to and including 3.2

HAVE A CONFLICT?

I will ask for documentation 9/26 – 9/30

BRING: your UB card

NO ELECTRONICS: cell phone, calculator, etc.

ELECTRONICS:

off & away

ROADMAP

Last time

- variables
- expression evaluation
- object diagrams

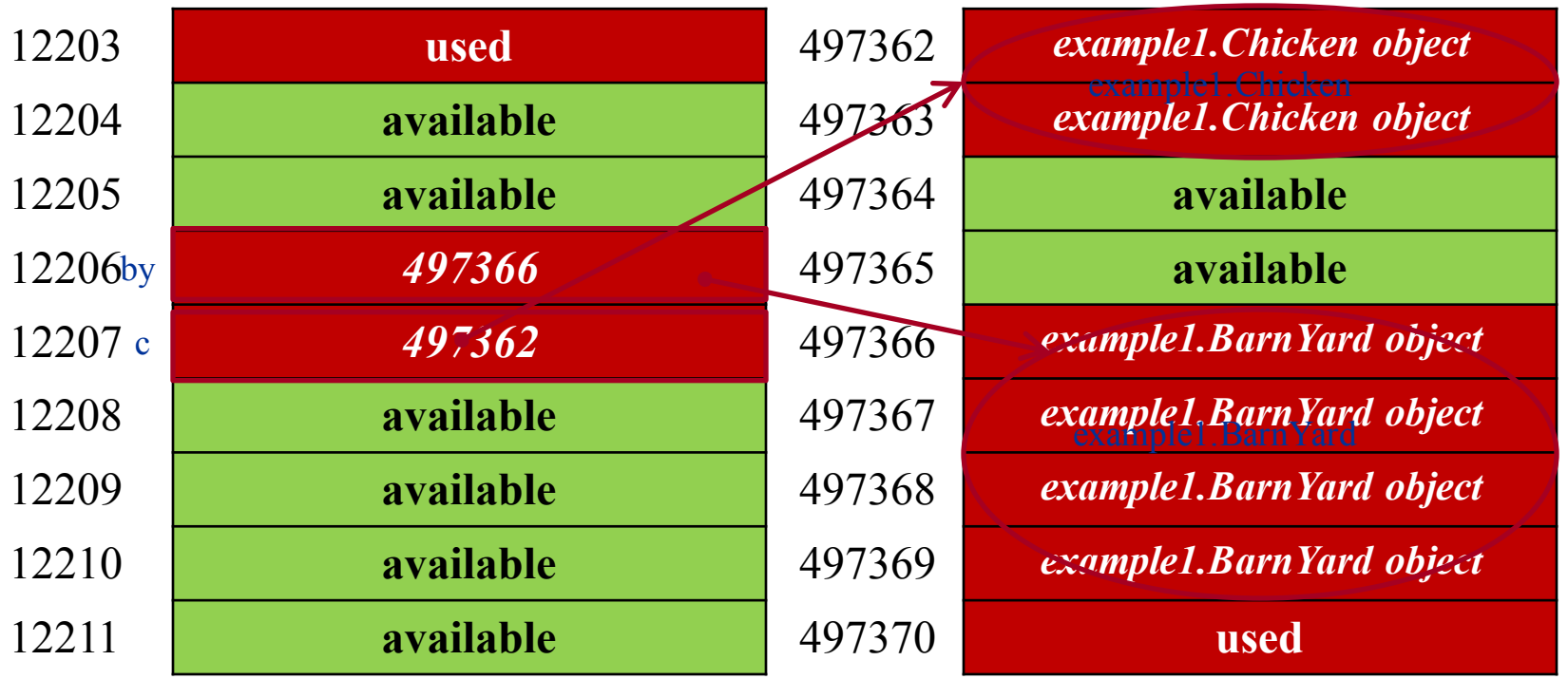
Today

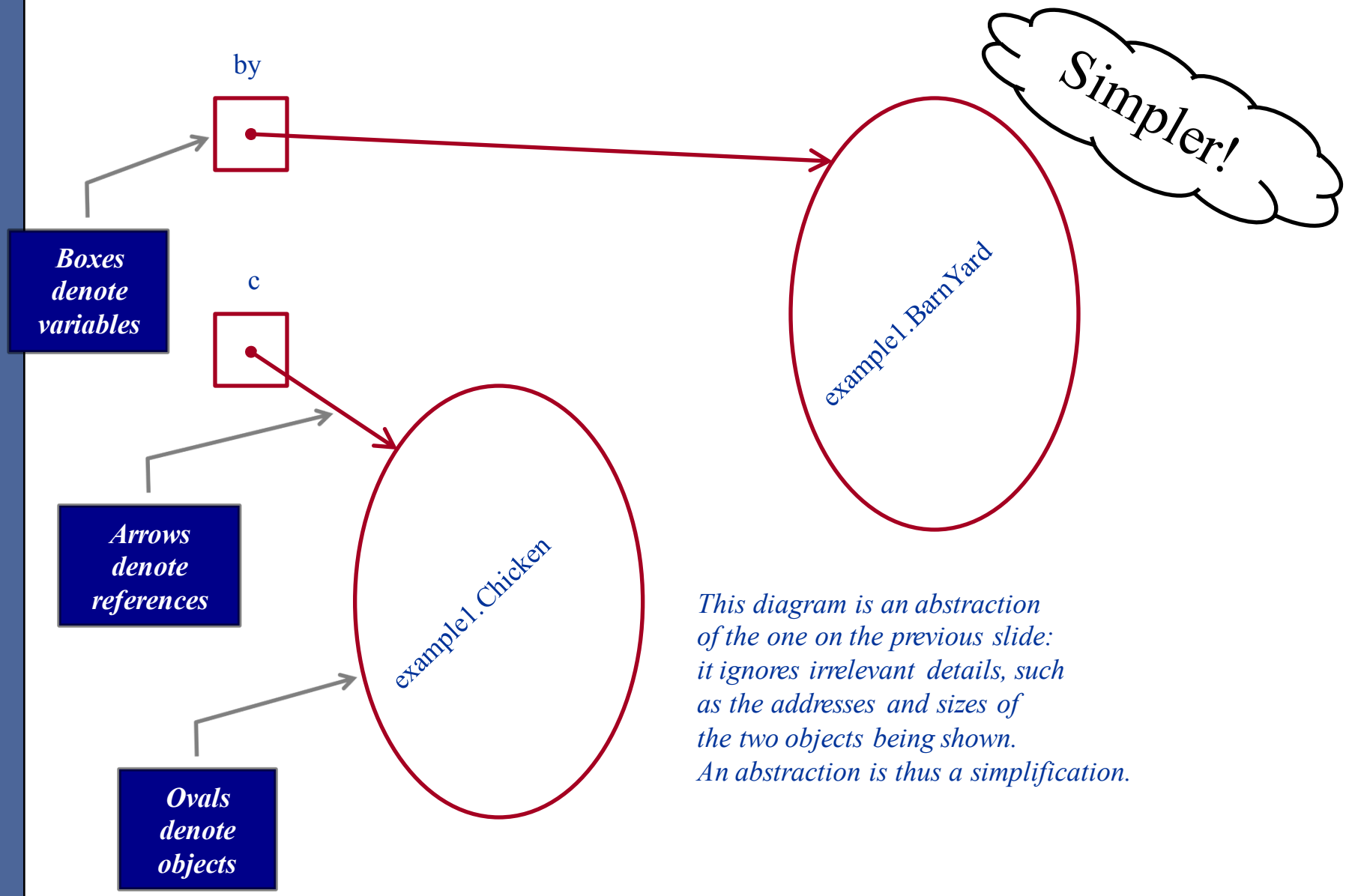
- class definitions in detail (terminology review)
- variable scope & lifetime
- method definitions

Coming up

- class relationships

REVIEW


12203
used
12204
available
12205
available
12206 by
497366
12207 c
497362
12208
available
12209
available
12210
available
12211
available
497362
example1.Chicken object
497363
example1.Chicken object
497364
available
497365
available
497366
example1.BarnYard object
497367
example1.BarnYard object
497368
example1.BarnYard object
497369
example1.BarnYard object
497370
used



This diagram is an abstraction of the one on the previous slide: it ignores irrelevant details, such as the addresses and sizes of the two objects being shown. An abstraction is thus a simplification.

MOVING ON

Our first class definition!

Here's a minimal class definition. We will label and discuss each part of it in detail next class. For now we identify the major parts:

```
package lab2;

public class Farm {
     public Farm() {
      }
}
```

Syntax

Package declaration is shown in green:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

package is a reserved word:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

lab2 is the name of the package – you choose this

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

A semicolon ; marks the end of the declaration:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

The class definition is shown in green:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

The class definition consists of a header . . .

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

. . . and a body:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

The class header consists of an access control modifier . . .

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

. . . the reserved word class . . .

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

. . . and a class name:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

The class body begins with an opening brace '{' . . .

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

. . . and ends with the matching closing brace '}':

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

In this example, the body consists of a single constructor definition:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

The constructor definitions consists of a header . . .

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

. . . and a body:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

The constructor header consists of an access control modifier . . .

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

. . . the constructor name (which is the same as the class name) . . .

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

. . . and a parameter list:

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

Syntax

The constructor body begins with an opening brace '{' . . .

```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

. . . and ends with the matching closing brace '}':

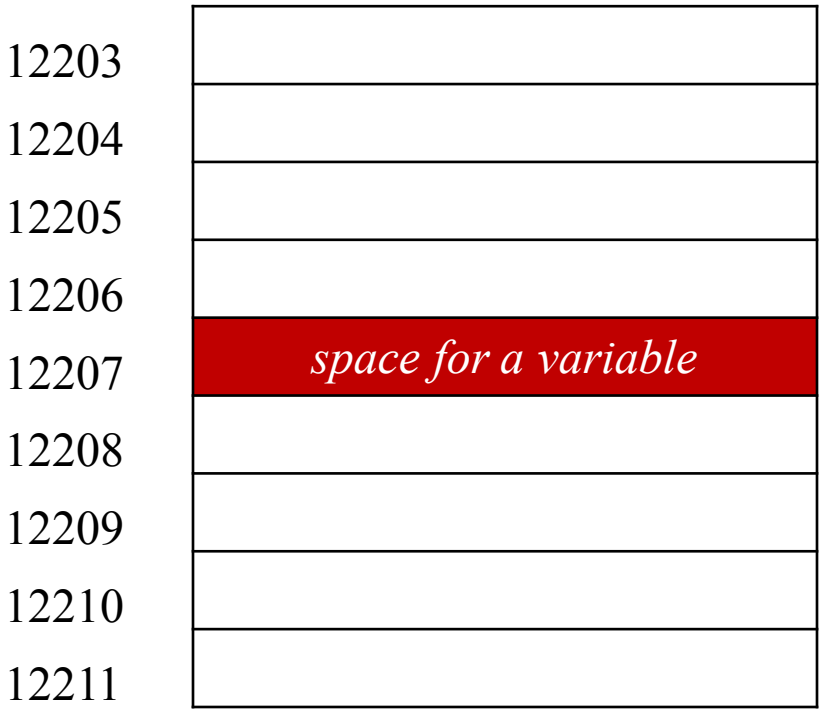
```
package lab2;

public class Farm {
        public Farm() {
        }
}
```

VARIABLES (more detail)

A variable is:

A variable exists at a storage location in memory.

For example, location 12207:



A variable has:

- a name → in the HLL (Java)
- a location → in memory
- a type → representation scheme/size
- a value → contents
- a scope
- a lifetime

} We'll discuss these next

A variable has:

a name → determined by declaration

a location

a type → determined by declaration

a value

a scope

a lifetime

A variable has:

a name

a location

a type

a value → determined by assignment

a scope

a lifetime

SCOPE

(no, not the mouthwash…)

Variable scope

The *scope* of a variable is the part of a program where a variable declaration is in effect.

Variables declared in different ways have different scope:

- local variables
- instance variables

Scope of a local variable

A variable declared within a constructor (or a method) is called a *local variable*.

The *scope* of a local variable is from the point of the declaration to the end of the brace-delimited block containing the declaration.

Scope of 'c'

```
package lab2;

public class Farm {
        public Farm() {
                example1.Terrarium t;
                t = new example1.Terrarium();
                example1.Chicken c;
                c = new example1.Chicken();
                t.addChicken(c);
                c.start();
        }
}
```

Declaration

End of block containing declaration

Scope of an instance variable

A variable declared within a class but outside of any method is called an *instance variable*.

The *scope* of an instance variable is the entire class body.

Scope of '_tail'

```
package code;

public class Dog {

    private Tail _tail;

    public Dog() {
        _tail = new Tail();
    }

}
```

Start of block containing declaration (class definition)

Declaration

End of block containing declaration (class definition)

LIFETIME

(sorry, no pun here)

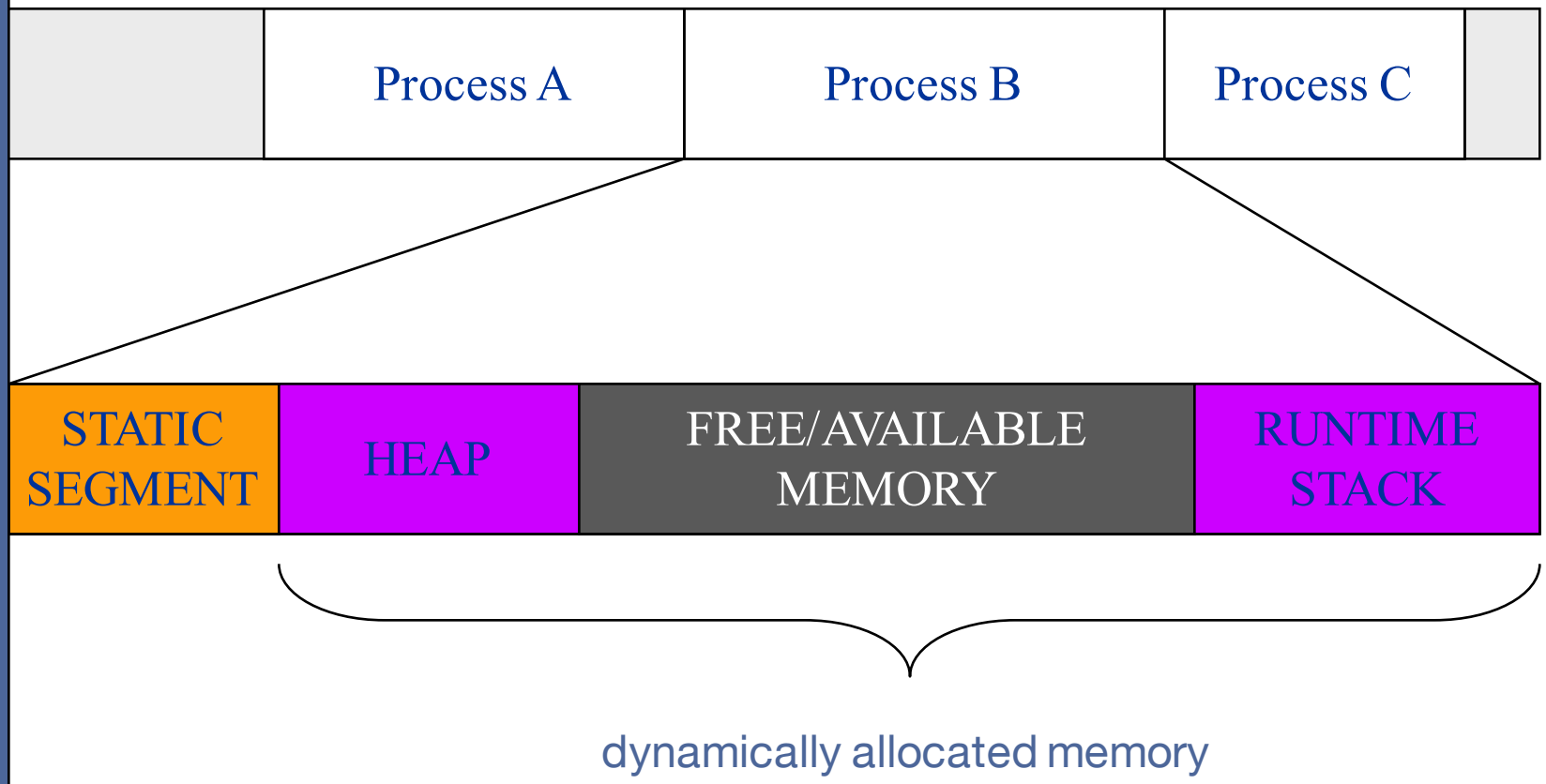
Lifetime of a variable

The *lifetime* of a variable is the period of time during execution of a program that the variable exists in memory. This is a dynamic property (one relating to runtime).

Variables declared in different ways have different lifetimes:

- local variables
- instance variables

Memory organization



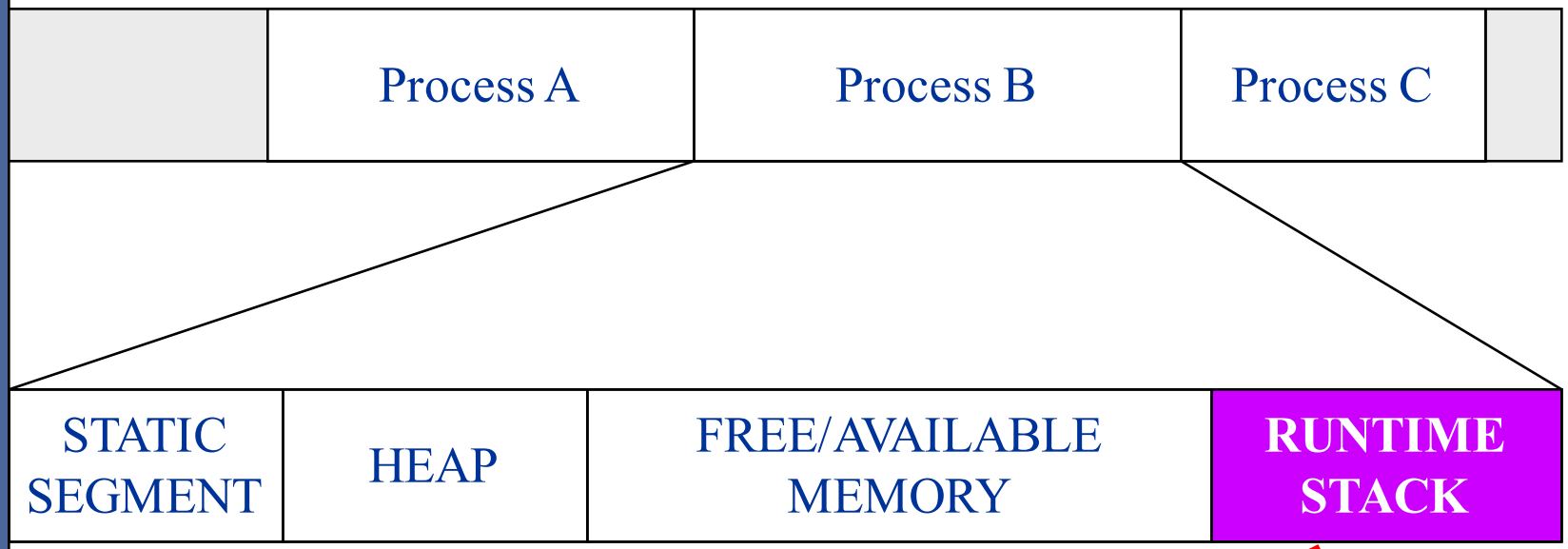
Lifetime of a local variable

A local variable comes into existence when a method is called, and disappears when the method is completed.

Space for a local variable is allocated in a special region of memory, called the *runtime stack*.

All the local variables of a method are allocated space in the same area, called a *stack frame* (or *invocation record*).

Memory organization



Local variables are stored on the runtime stack. Each method invocation (call) results in an invocation record (stack frame) being added to the top of the stack. When a method exits, its invocation record is removed from the top of the stack.

Lifetime of an instance variable

Instance variables are created when a class is instantiated.

- 'new' allocates memory from the heap

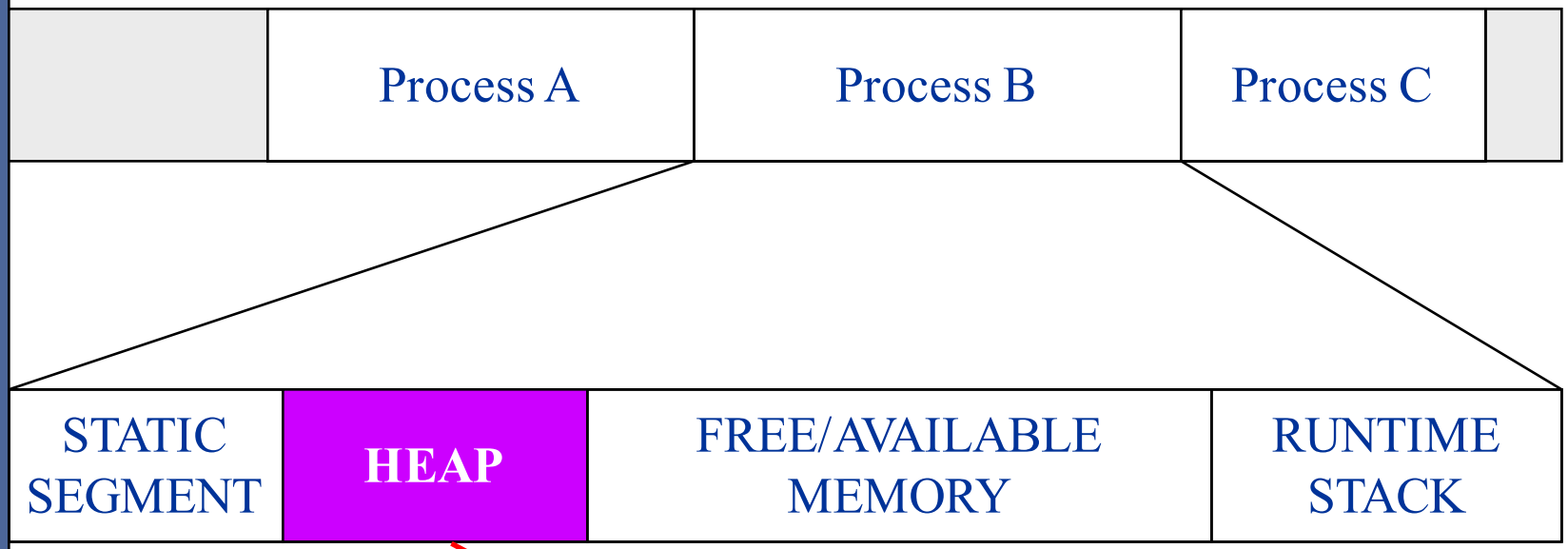
Each object has its own set of instance variables.

- the variables are the constituents of an object
- instance variables therefore exist on the heap

Instance variables persist as long as their objects persist

- as far as we know right now, objects persist until the end of the runtime of the program.

Memory organization



All memory allocated by 'new' comes from the heap.

Objects are allocated space by 'new', and their representations (which contain their instance variables) therefore exist on the heap.