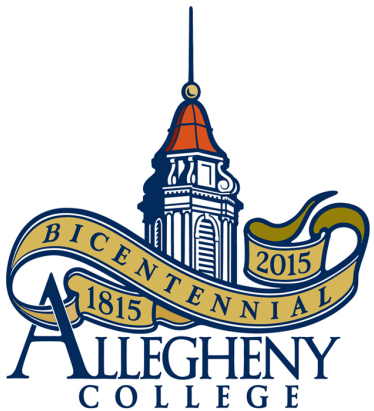
CS201 - PL'S

More on Scopes

Aravind Mohan

Allegheny College

September 20, 2021



Scope Rules

A **scope** is a program section of maximal size in which no bindings change, or at least in which no re-declarations are permitted.

In most languages with subroutines (functions, methods, procedures), we OPEN a new scope on subroutine entry:

- create bindings for new local variables,
- deactivate bindings for global variables that are re-declared (these variable are said to have a “hole” in their scope)
- make references to variables

Scope Rules

On subroutine exit:

- destroy bindings for local variables
- reactivate bindings for global variables that were deactivated

The book uses the term "**elaboration**" for the process of allocating memory and creating bindings associated with a declaration.

Elaboration Example

```
public class MyClass {
  private int a;
  public int getA() {
     return a;
  }
  public void setA(int x) {
     a = x;
  }
}
```

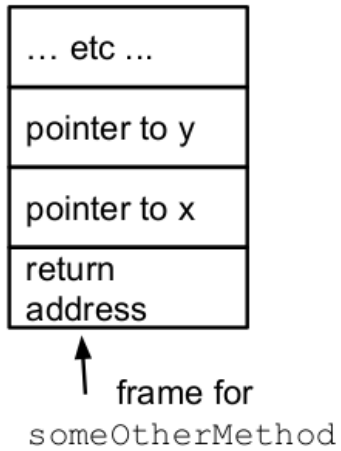
```
public void someOtherMethod() {

   MyClass x = new MyClass();

   MyClass y = new MyClass();
      …
}
```

Whenever **someOtherMethod** is invoked, a new activation record (frame) is created for **someOtherMethod** and the declarations of **x** and **y** are elaborated into locations in this frame; the names **x** and **y** are bound to these locations.

(NOTE: creation of the frame itself is an elaboration of the declaration of function **someOtherMethod**)



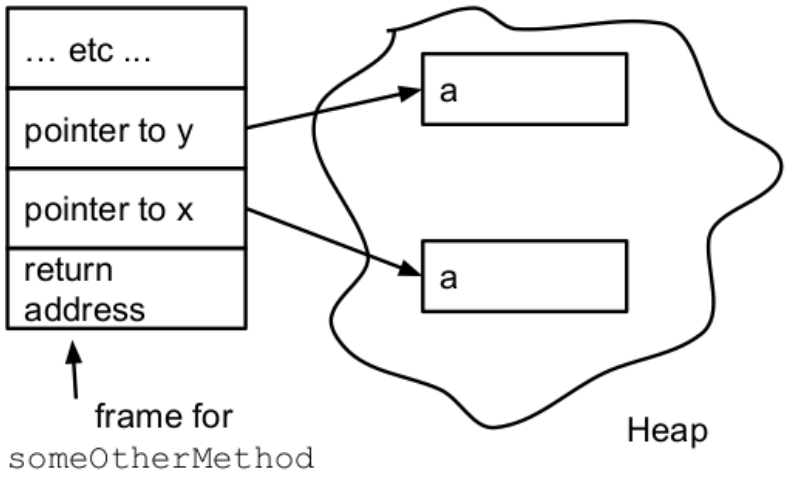
Elaboration Example

```
public class MyClass {
  private int a;
  public int getA() {
     return a;
  }
  public void setA(int x) {
     a = x;
  }
}
```

```
public void someOtherMethod() {

   MyClass x = new MyClass();

   MyClass y = new MyClass();
      ...
}
```



Furthermore, the declarations of the instance variable **a** are elaborated into memory locations in the heap and the names **x.a** and **y.a** are bound to these locations.

Heap Allocation (Dynamic allocation)

Example (Java):

```
int values[ ];
System.out.print("How big is the array? ");
int n = scan.nextInt();
values = new int[n];
```

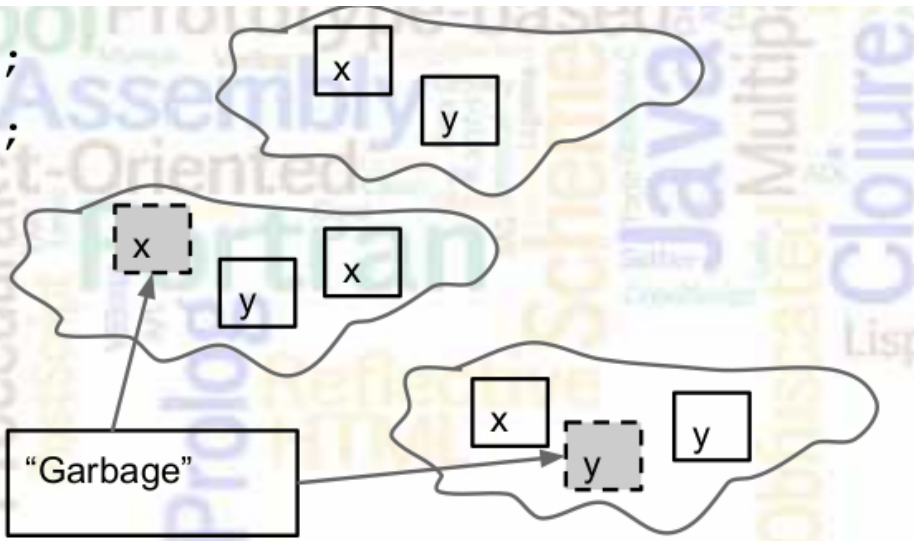
- No way to know at compile time how much space will be needed for the array “values” – determined at run time.
- So... how can we know how much memory to save on the stack?
- We must allocate it dynamically from a special memory area called the **heap**.

Heap Allocation

Stack grows and shrinks (“push” and “pop”); easy to generate code for this at compile time.

Heap: no pattern–no “last-in, first-out” or similar rule:

```
MyClass x = new MyClass();
MyClass y = new MyClass();
if (count == 10)
  x = new MyClass();
else
  y = new MyClass();
```

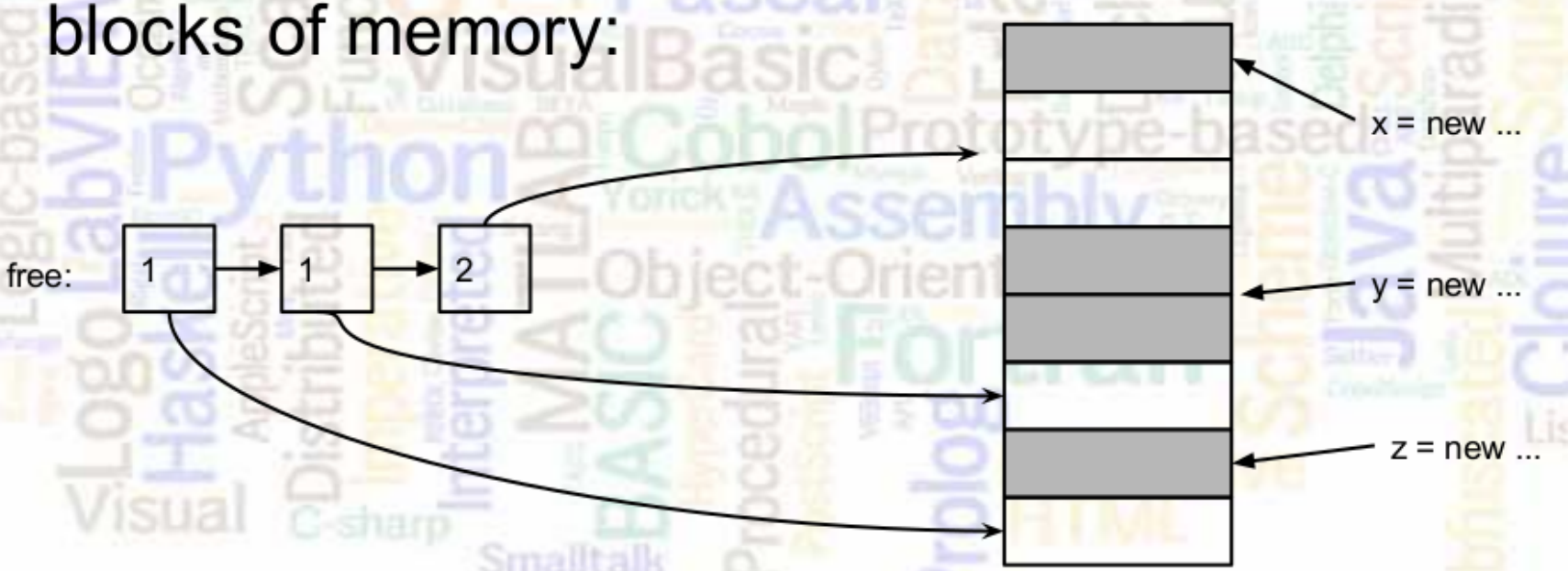


Heap Allocation

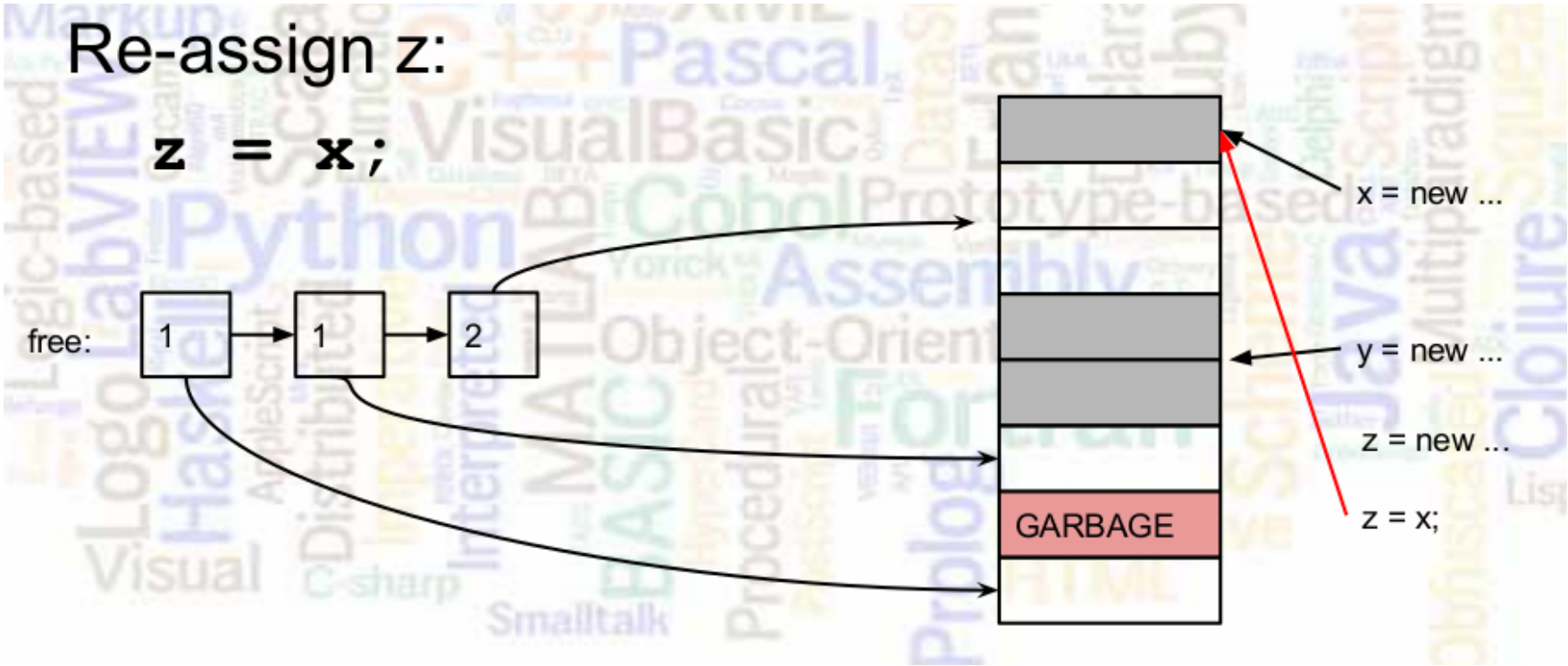
- Any time we use the “new” operator in Java, we allocate space from the heap.
- In C, use of the malloc function allocates from the heap.
- Harder to maintain than a stack; many techniques used.

Heap Allocation

Simplest way to maintain heap--"free list" of blocks of memory:



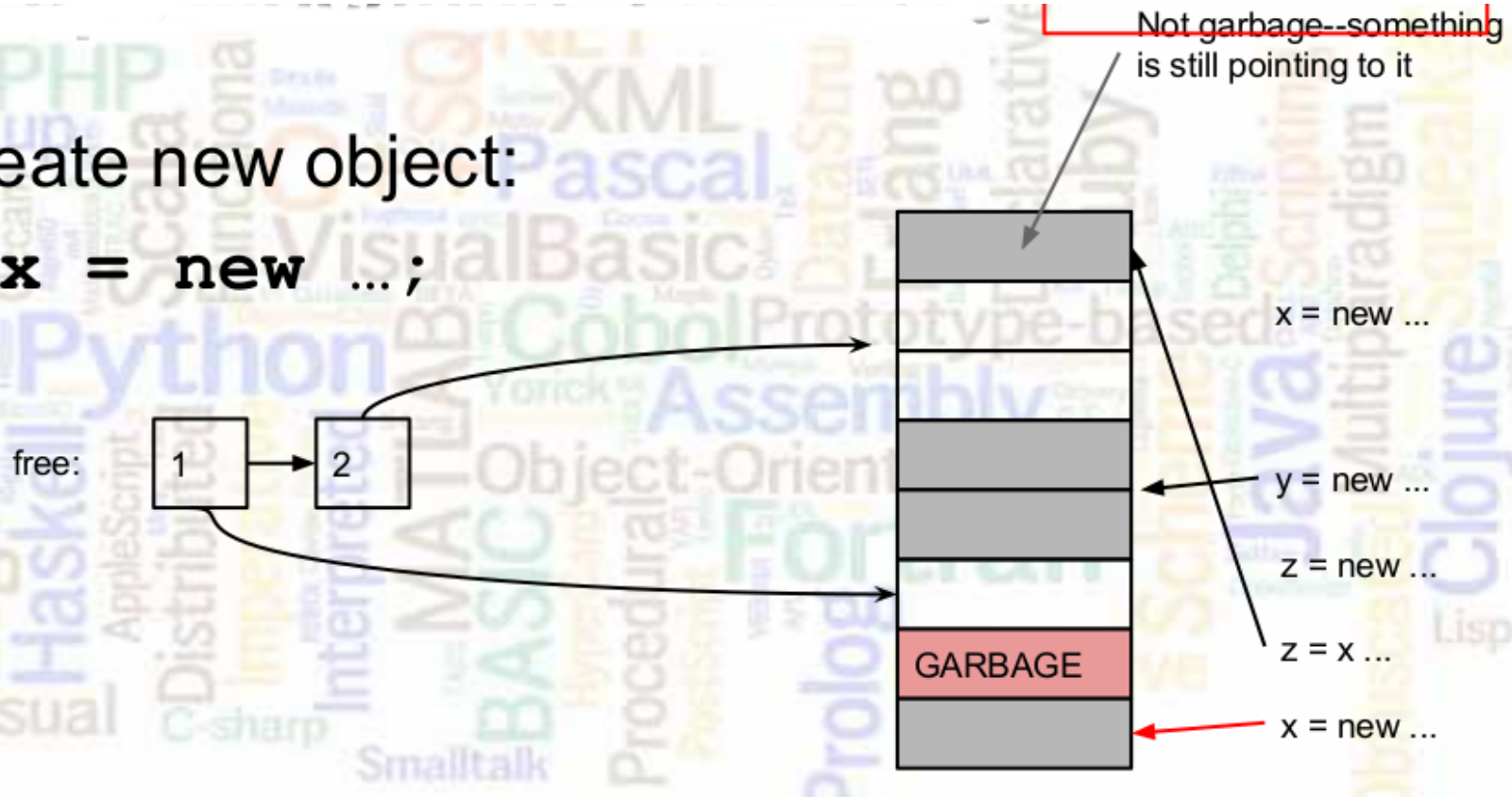
Heap Allocation



Heap Allocation

Create new object:

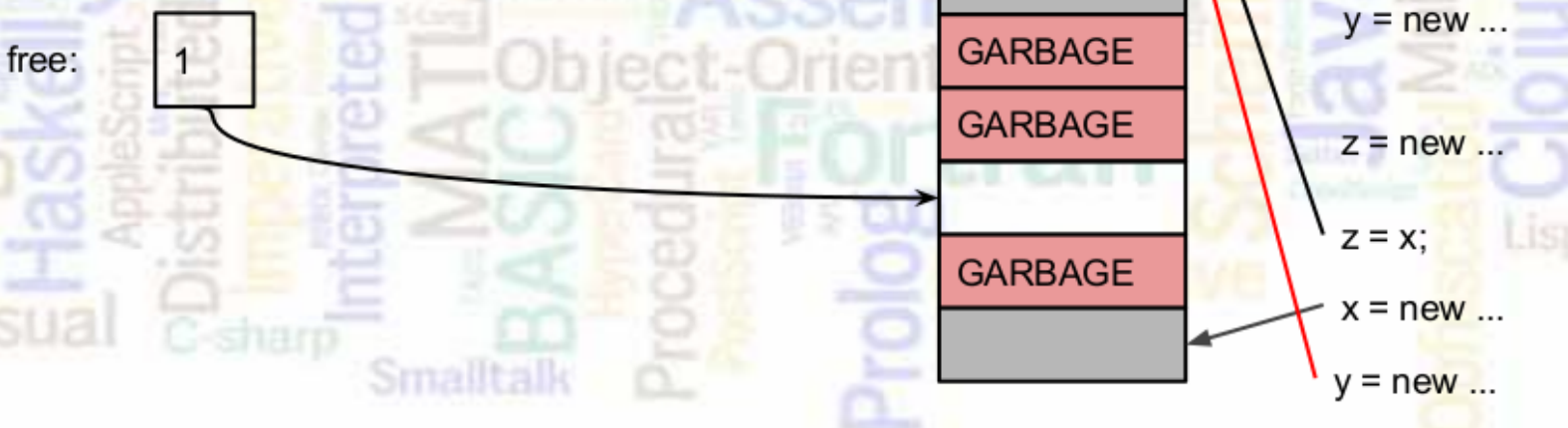
```
x = new ...;
```



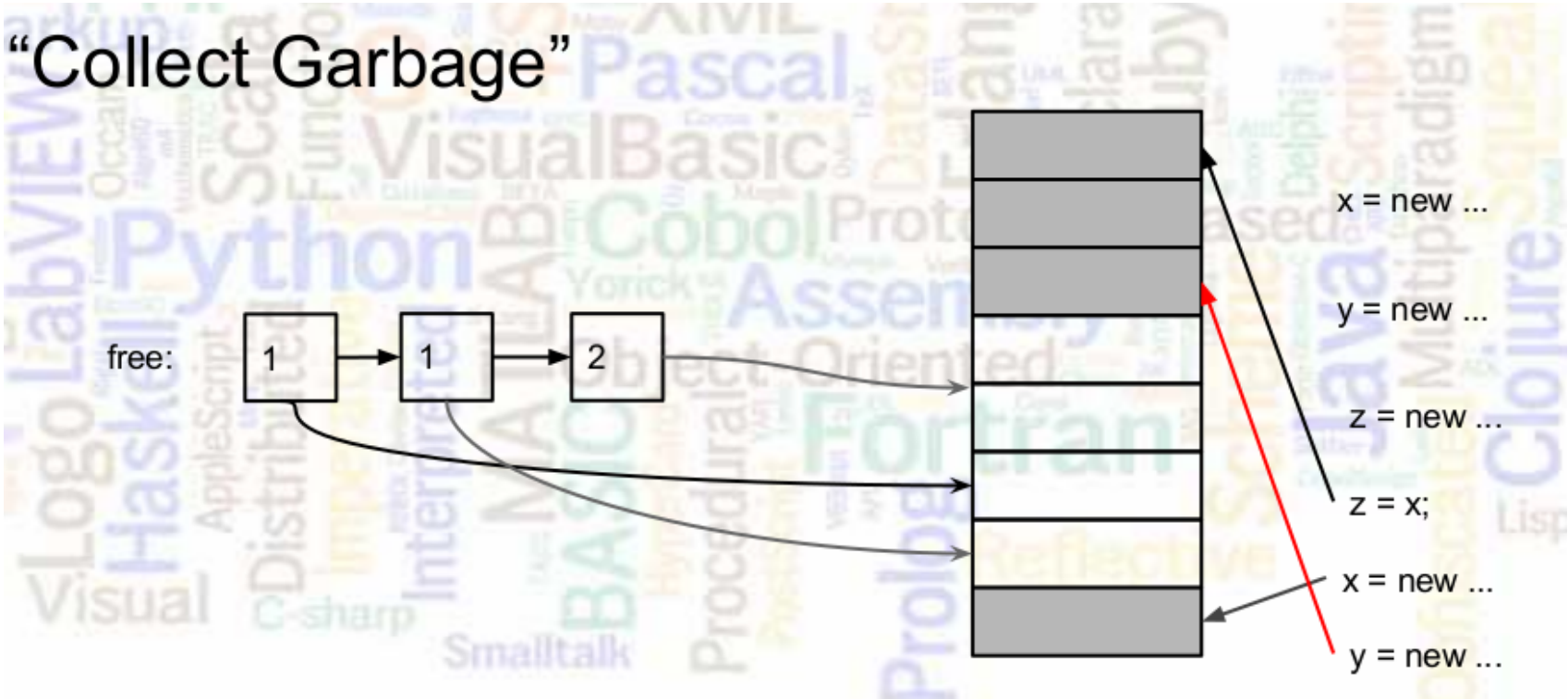
Heap Allocation

Create new object:

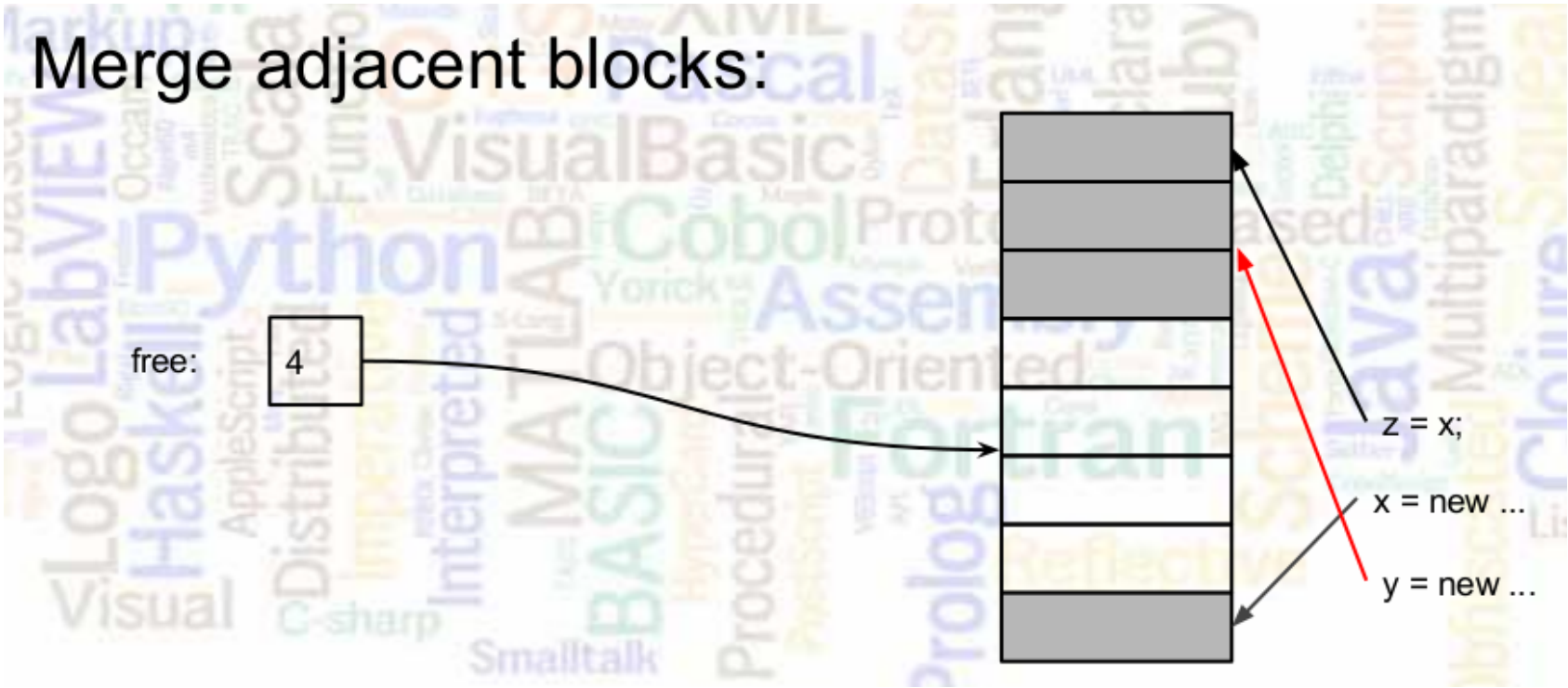
```
y = new ...;
```



Heap Allocation



Heap Allocation



Which Block to Use from Free List?

"First fit": take the first block in the free list that is big enough to satisfy the requested amount:

Request for block of size 1:

"Best fit": take the smallest block in the free list that is big enough to satisfy the request:

Request for block of size 1:

This is an expansion of material on page 119

Summary: Names, Scopes and Bindings

Binding

is an association between an attribute and an entity, such as between a variable and its type or value, or between an operation and a symbol.
static vs. dynamic

Example: $count = count + 5;$

Summary: Names, Scopes and Bindings

Example: $count = count + 5;$
Some of the bindings and their binding times:

- The type of count is bound at compile time.
- The set of possible values of count is bound at compiler design time.
- The meaning of the operator symbol + is bound at compile time, when the types of its operands have been determined.
- The internal representation of the literal 5 is bound at compiler design time.
- The value of count is bound at execution time with this statement.

Example: Static vs. Dynamic Scoping

```
function big() {
function sub1() {
var x = 7;
sub2();
}
function sub2() {
var y = x;
}
var x = 3;
sub1();
}
```

Under static scoping, the reference to the variable x in sub2 is to the x declared in the procedure big.

Example: Static vs. Dynamic Scoping

```
function big() {
 function sub1() {
 var x = 7;
 }
 function sub2() {
 var y = x;
 var z = 3;
 }
 var x = 3;
}
```

Under dynamic scoping, the meaning of x may reference the variable from either declaration of x, depending on the calling sequence.

Summary: Names, Scopes and Bindings

How is scope implemented at execution time?

- Pointers on the activation record stack refer to surrounding scope.
- Lexical: “static link”.
- Dynamic: “dynamic link”.

Example: JavaScript

Go to `http://goo.gl/hcrqmE` for a working version of Figure 3.5 (in JavaScript).

Ways Around “Hole in Scope”

Some languages allow access to scopes that are ”hidden” by new declarations. E.g., Java:

```
public class MyClass {
    private int x;
    // This creates a hole in the scope
    // of the instance variable x
    public void myMethod(int x) {
        x = 10; // Parameter x
        this.x = 20; // instance variable x
        ...
```

C++ has a similar construct.

Reading Assignment

PLP Chapter 03

Questions

Do you have any questions from this class discussion?