Functions

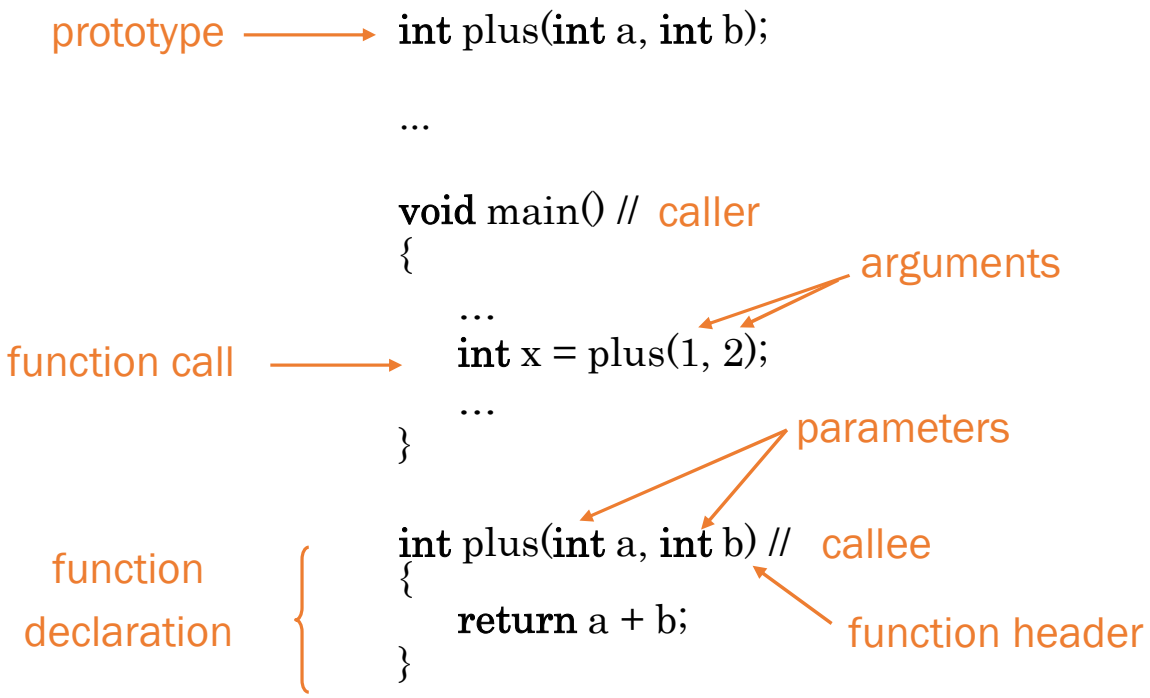
Professor: Suman Saha

Basic terminology

- Functions, subroutines (Fortran), procedures, methods
- Value-returning functions:
  - *known as "non-void functions/methods" in C/C++/Java*
  - *called from within an expression. e.g.,* x = (b*b - sqrt(4*a*c))/2*a
- Non-value-returning functions:
  - *"void functions/methods" in C/C++/Java*
  - *known as "procedures" in Ada,*
  - *"subroutines" in Fortran,*
  - *called from a separate statement. e.g.,* strcpy(s1, s2);

Terminology

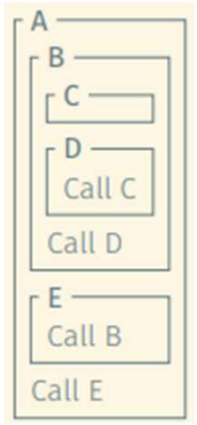
- Example in C



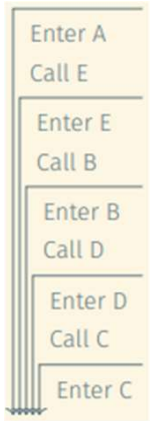
Static Chains and Dynamic Chains

We already discussed activation records or (stack) frames as a means to manage the space for local variables allocate to each subroutine call.

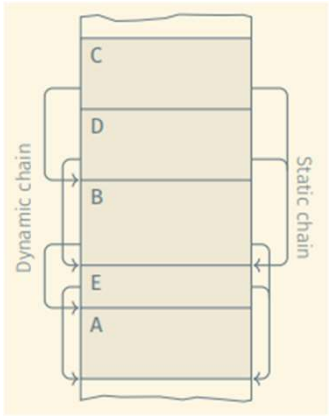
Source Code



Program Execution



Execution Stack



Inline Expansion

PennState

During compile time, the compiler replaces a subroutine call with the code of the subroutine.

Advantages:

- Avoids overhead associated with subroutine calls; faster code.
- Encourages building abstractions in the form of many small subroutines.
- Related to but cleaner than macros.

Disadvantages:

- Code bloating
- Cannot be used for recursive subroutines.
- Code profiling becomes more difficult

Parameter Passing

PennState

Notation:

f(a, b, c)	C, C++, Java, …
(f a b c)	Lisp, Scheme
a f: b fcont: c	Smalltalk, Objective C
f a b c	Haskell, shell scripts

Meaning: Execute the named subroutine with its formal arguments bound to the provided actual arguments.

Parameters VS. Arguments

PennState

Parameters (AKA formal parameters, formal arguments): names in the declaration of a function header

Arguments (AKA actual parameters, actual arguments): variables/expressions passed to a function during a function call

Parameter-Argument Matching

PennState

- Usually by number and by position
  - Suppose f has two parameters, then any call to f must have two arguments, and they must match the corresponding parameters' types.
- Exceptions
  - Python/Ada/OCaml/C++
    - arguments can have default values
    - Python example:

```
>>> def myfun(b, c=3, d="hello"):
                return b + c
>>> myfun(5,3,"hello")
>>> myfun(5,3)
>>> myfun(5)
```

Parameter-Argument Matching

PennState

- Exceptions:
  - Arguments and parameters can be linked by name
  - Python example:

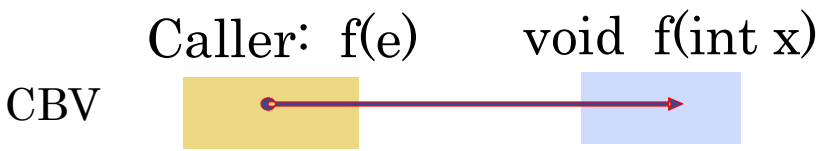
```
>>> def myfun(a, b, c):
        return a-b
>>> myfun(2, 1, 43)
1
>>> myfun(c=43, b=1, a=2)
1
>>> myfun(2, c=43, b=1)
1
```

Parameter Passing

- How are values passed between arguments and parameters?
- Different parameter passing mechanisms
  - Call by value (CBV, AKA pass by value)
  - Call by result (CBR)
  - Call by value-result (CBVR)
  - Call by reference (CBR)
  - Call by name (CBN)

Call By Value

- Compute the *value* of the argument at the time of the call and copy that value to storage for the corresponding parameter
  - Copy-in semantics
    - At start of call, argument's value is computed and copied into parameter's storage



  - Example: void f(int x) {...}; f(3+5)

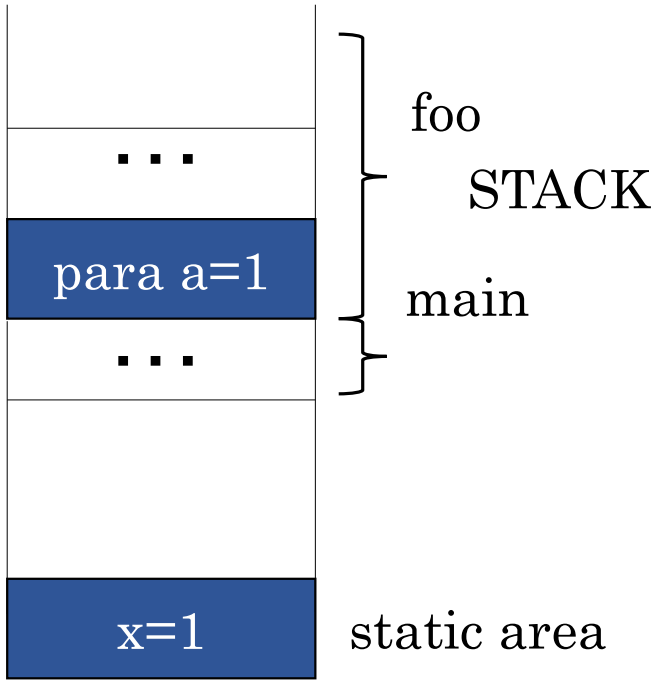
Call By Value: Storage Allocation

PennState

1. At start of the call, arguments are evaluated to their values
2. Storage allocated for parameters on AR
3. Argument values copied to storage for parameters AR
4. AR destroyed when callee returns

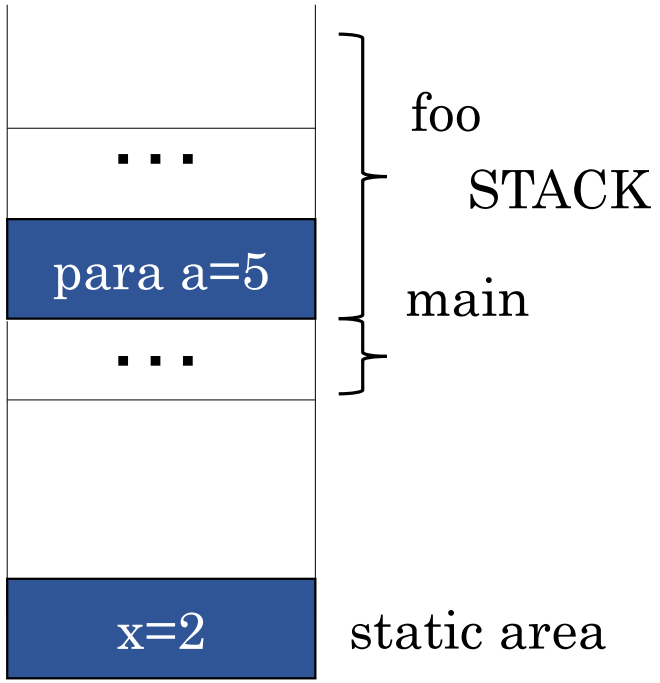
Call By Value

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    print(foo(x)); //result?
    print(x); //result?
}
```



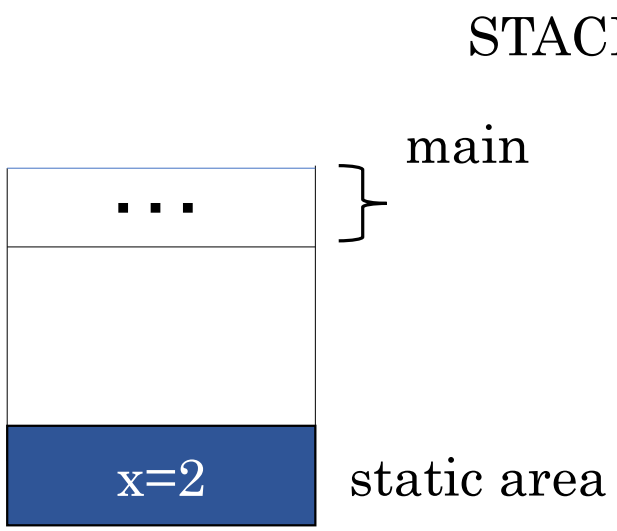
Call By Value

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    print(foo(x)); //result?
    print(x); //result?
}
```



Call By Value

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    print(foo(x)); //result?
    print(x); //result?
}
```



Call By Value

PennState

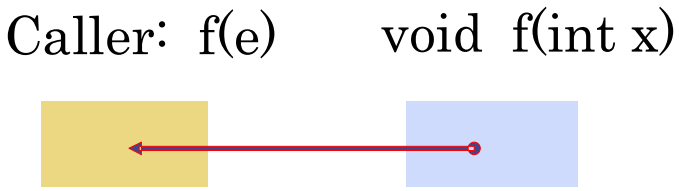
- Arguments and parameters have separate storage
  - Their values may diverge
  - Call by value doesn't allow the callee to modify an argument's value.
- Does the following C program swap the values of arguments?

```
void swap (int a, int b) {
  int temp=a;
  a = b;
  b = temp;
}
... x =1; y = 2; swap(x,y); ...
```

- All arguments in C and Java (primitive) are passed by value.
  - But pointers can be passed to allow argument values to be modified.
  - E.g., void swap(int *a, int *b) { … }

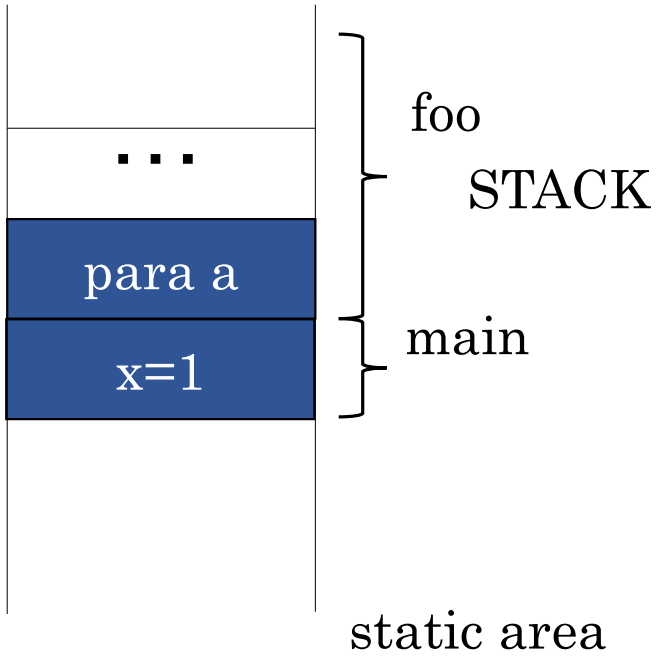
Call By Result

- Copying the final value computed for the parameter out to the argument at the end of the call
  - The local variables corresponding to the formal parameters are not set at routine activation
- Copy-out mechanism
  - before returning control to caller, final value for the parameter is copied to the argument



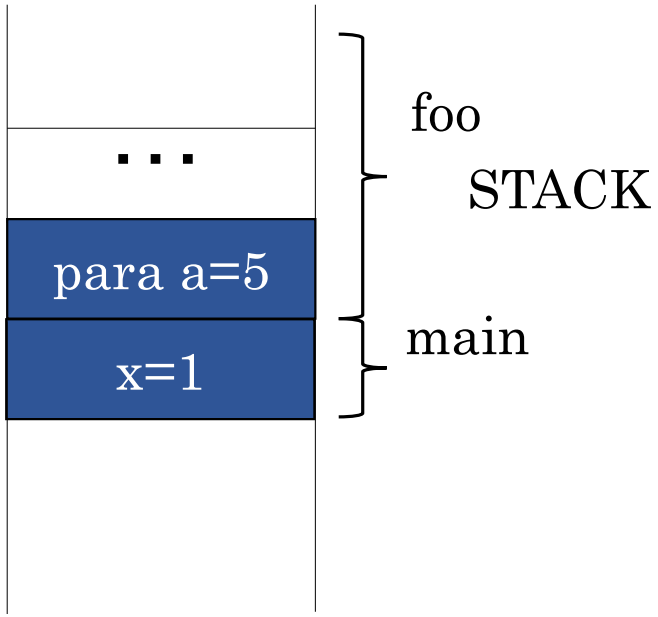
Call By Result

```
int foo (int a) {
    a = 5;
}
void main() {
    int x = 1
    foo(x);
    print(x); //result?
}
```



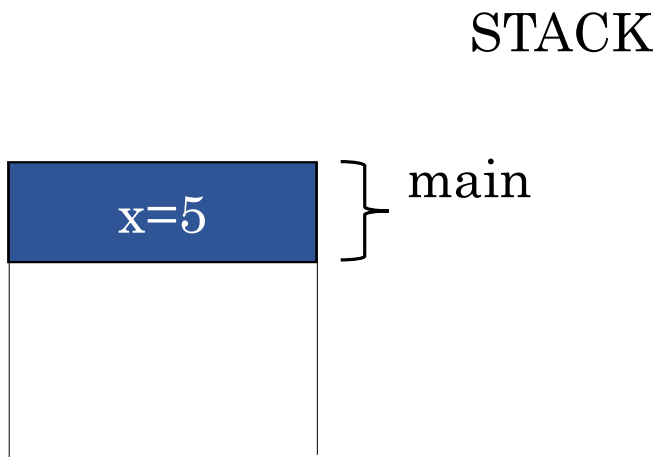
Call By Result

```
int foo (int a) {
    a = 5;
}
void main() {
    int x = 1
    foo(x);
    print(x); //result?
}
```



Call By Result

```
int foo (int a) {
    a = 5;
}
void main() {
    int x = 1
    foo(x);
    print(x); //result?
}
```



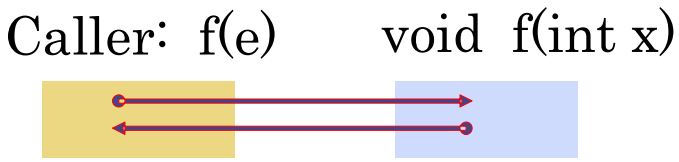
Call By Result

- Notes:
  - The parameter is initialized by the callee
  - The callee doesn't care about the initial value of the argument
  - Used as a mechanism for the callee to return a value

Call By Value-Result

PennState

- Two steps:
  1. Copying the argument's value into the parameter at the beginning of the call
  2. Copying the computed result back to the corresponding argument at the end of the call
- Copy-in and copy-out
  - at start of call, argument's value is computed and is copied into parameter's storage
  - before returning control to caller, final value for the parameter is copied to the argument



Example: Call By Value-Result

PennState

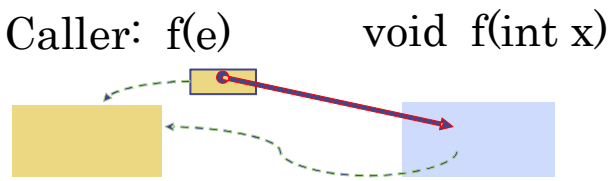
```
SUBROUTINE F(NA, NB)
    NA = NA + 3
    NB = NB + 5
    END

NX = 10
NY = 25
CALL F(NX, NY)
PRINT *, NX, NY // Result 13 30
NX = 10
CALL F(NX, NX)
PRINT *, NX // Result 15
END
```

Call By Reference

- Compute the *address* of the argument at the time of the call and assign it to the parameter
  - During function execution, the argument and the parameter are alias
  - Argument must have an address (l-value in C/C++)
- C++ example

```
int h=10;
void B(int &w) {
        int i;
        i = 2*w;
        w = w+1
}
… B(h) …
```



Call-by-Value-Result vs. Call-by-Reference

PennState

```
program foo;
var x: int;
        procedure p(y: int);
        begin
                y := y + 1;
                y := y * x;
        end
begin
        x := 2;
        p(x);
        print(x);
end
```

	Call-by-Value-Result		Call-by-Reference	
	x	y	x	y
Entry to p	2	2	2	2
After y: y+1	2	3	3	3
At p's return	6	6	9	9

Exercise: Call By Value

PennState

```
void sub(int x) {
    x = x + 5;
}
a = 10;
sub(a);
print(a);
```

A. 15
B. 10
C. 20

Exercise: Call By Value-Result

PennState

```
void sub(int x) {
    x = x + 5;
}
a = 10;
sub(a);
print(a);
```

A. 15
B. 10
C. 20

Call By Name

PennState

- Textually substitute the argument for each occurrence of the parameter in the function's body
  - without computing the value of the argument first
  - can be viewed as macro expansion
  - originally used in Algol 60
- C macros
  - #define SQUARE(x) ((x) * (x))

Call-by-Value vs. Call-by-Name

PennState

```
def square(x, y) = x * x
```

// Call-by-Value

```
square(2*2, 2+3) // function call
square(4, 2+3) // reduction 1
square(4, 5) // reduction 2
4 * 4  // reduction 3
16  // reduction 4
```

// Call-by-Name

```
square(2*2, 2+3) // function call
(2*2) * (2*2) // reduction 1
4 * (2*2) // reduction 2
4 * 4  // reduction 3
16  // reduction 4
```

Call By Name

- Not many languages use it
  - Examples: Algol 60
- Haskell uses call by need, a variant of call by name.
  - Call-by-need or lazy evaluation is an evaluation strategy where an expression isn't evaluated until its first use i.e to postpone the evaluation till its demanded.
- Compilers for these languages, have relied heavily on thunks, with the added feature that the thunks save their initial result so that they can avoid recalculating it

Considerations when choosing parameter-passing mechanisms

- minimize access to data
  - use pass-by-value if no data need be returned
  - use pass-by-result if no data need be sent to callee
  - Call by reference or value-result otherwise
- only use pass-by-reference when needed
  - can accidentally change the value of the parameter
- large arrays/objects usually pass-by-reference
  - avoids copying the entire array

Parameter passing in major languages: C

PennState

- Essentially pass-by-value
- Pass-by-reference
  - simulated using pointer values
  - pointer notation
    - int *p
      - declares p to be a pointer to an int
    - &x
      - provides the address of variable x
    - *p
      - dereferences a pointer
      - get the value of the variable pointed at by p

```
int* pc, c;
c = 5;
pc = &c;
```

Parameter passing in major languages: C++

PennState

- same as C, but
- also has Call by reference
  - these are like pointers, but implicitly dereferenced
  - true pass-by-reference
  - ```
    void swap(int& a, int& b) {
       int temp = a;
       a = b;
       b = temp;
    }
    swap(x,y);
    ```
  - note, int &a, int &b can also be used
    ```

Parameter passing in major languages: Java

PennState

- Pass-by-value
- primitive data types: values are copied
- object and array parameters
  - Pass references values
  - still Call by value, but it's the reference values being passed

Example: Why Java is Call By Value?

PennState

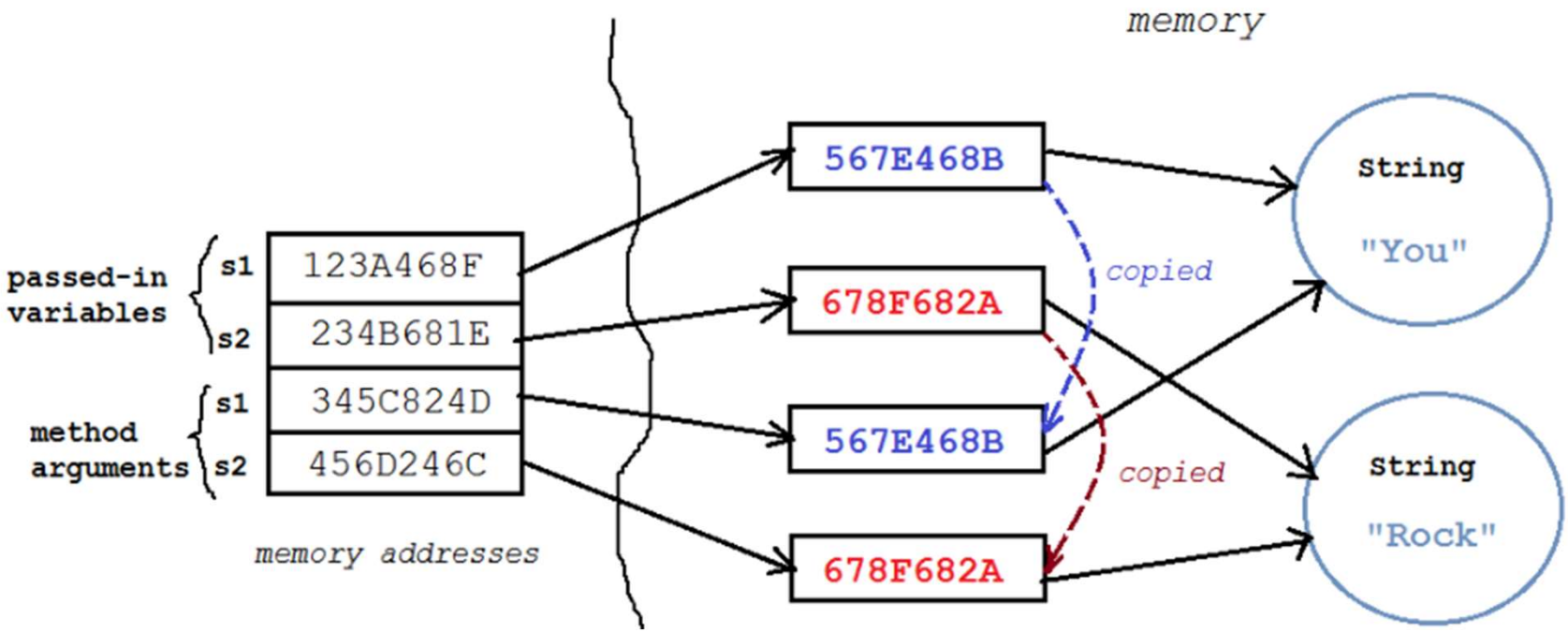
```java
/**
 * Impossible Swap function in Java
 * @author www.codejava.net
 */
public class SwapObject {
    public static void swap(String s1, String s2) {
        String temp = s1;
        s1 = s2;
        s2 = temp;
        System.out.println("s1(1) = " + s1);
        System.out.println("s2(1) = " + s2);
    }
    public static void main(String[] args) {
        String s1 = "You";
        String s2 = "Rock";
        swap(s1, s2);
        System.out.println("s1(2) = " + s1);
        System.out.println("s2(2) = " + s2);
    }
}
```

```
s1(1) = Rock
s2(1) = You
s1(2) = You
s2(2) = Rock
```

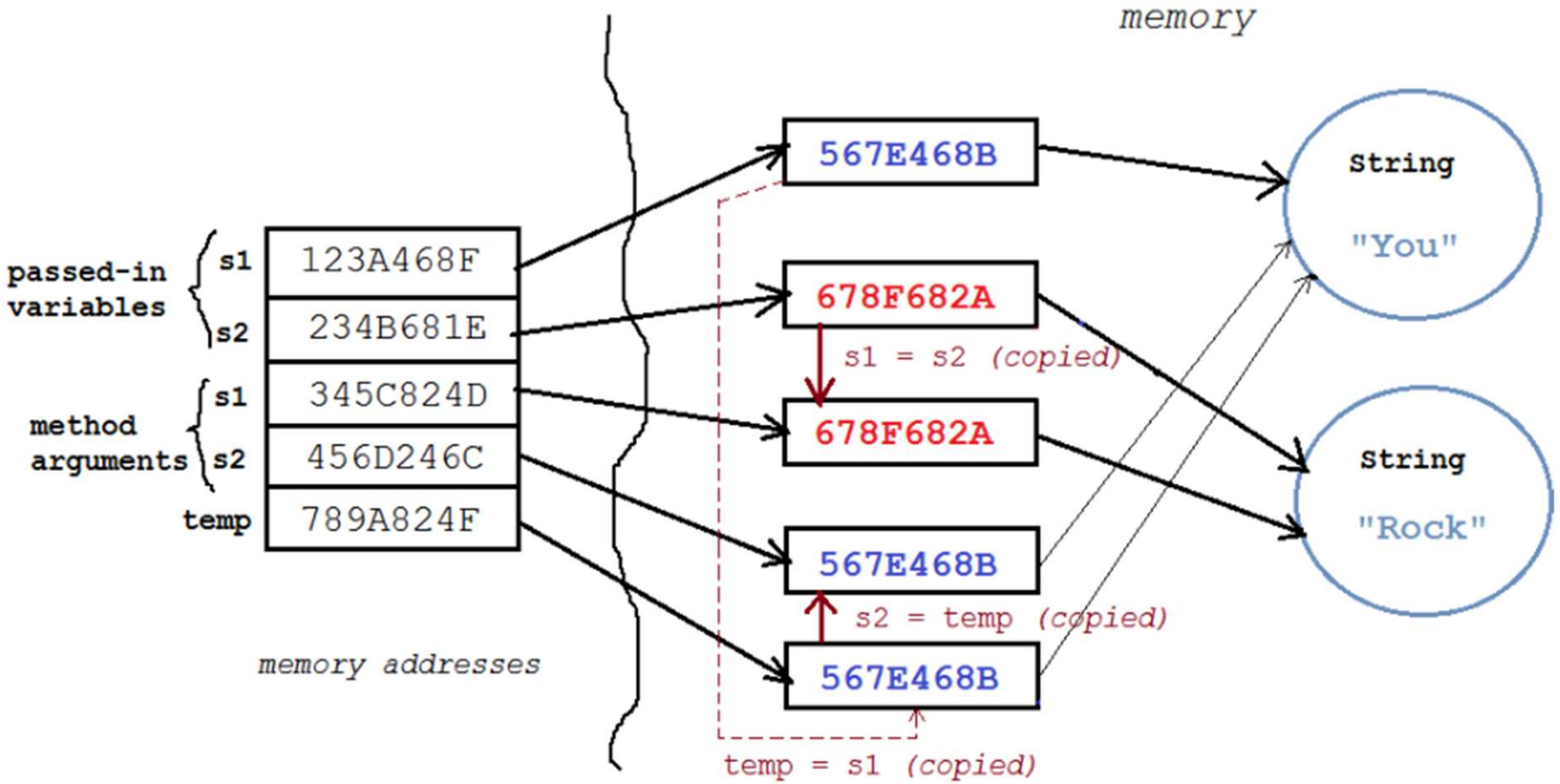
Example: Why Java is Call By Value?

PennState

Here's the memory state right before the method execution:



Example: Why Java is Call By Value?



Parameter passing in major languages: Ada

- Can specify each parameter as:
  - in: can be read but not modified
    - implementation can be Call by value or Call by reference depending on size of parameter
  - out: cannot be read until value set by the callee function
    - initial value is never used
    - argument get the final value
    - implementation can be Call by result or Call by reference depending on the size of parameter
  - in out: can be read and modified
    - implementation either Call by value-result or Call by reference

Ada Example

PennState

```
procedure shift(a:out integer, b:in out integer, c:in integer) is
begin
  a := b; b := c;
end shift;
```

Top Hat

PennState

Exercise

PennState

```
int p=5, q=6;
void foo(int b, int c) {
    b = 2 * c;
    p = p + c;
    c = 1 + p;
    q = q * 2;
    print(b + c);
}
main() {
    foo(p, q);
    print p, q;
}
```

Passed by value	Passed by reference
p = 5, 11 q = 6, 12 b = 5, 12 c = 6, 12	p = 5, 12, 18 q = 6, 19, 38 b = 5, 12, 18 c = 6, 19, 38
Print: 24 11 12	Print: 56 18 38

Passed by value Result	Passed by name
p = 5, 11, 12 q = 6, 12, 12 b = 5, 12 c = 6, 12	p = 5, 12, 18 q = 6, 19, 38 b = c =
Print: 24 12 12	Print: 56 18 38

Reading

PennState

Reading

- Chapter: 9.1, 9.2, and 9.3 (Michael Scott Book)