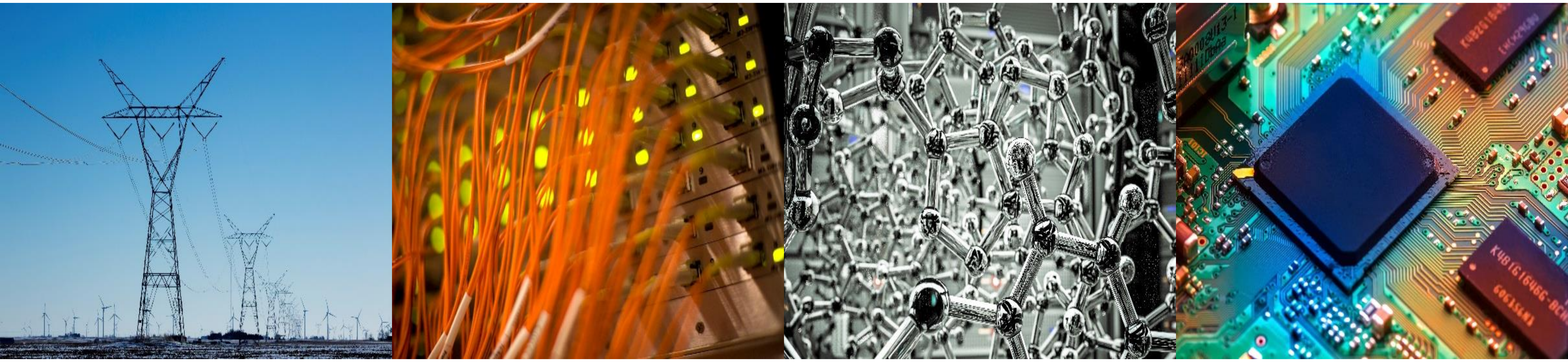
ECE 220 Computer Systems & Programming

Lecture 8 – Run-Time Stack

September 19, 2024



- MT1 conflict request deadline is Sunday, 9/22
- HKN review session will be held on Sunday, 9/22, 12:30pm-3pm in ECEB 1002

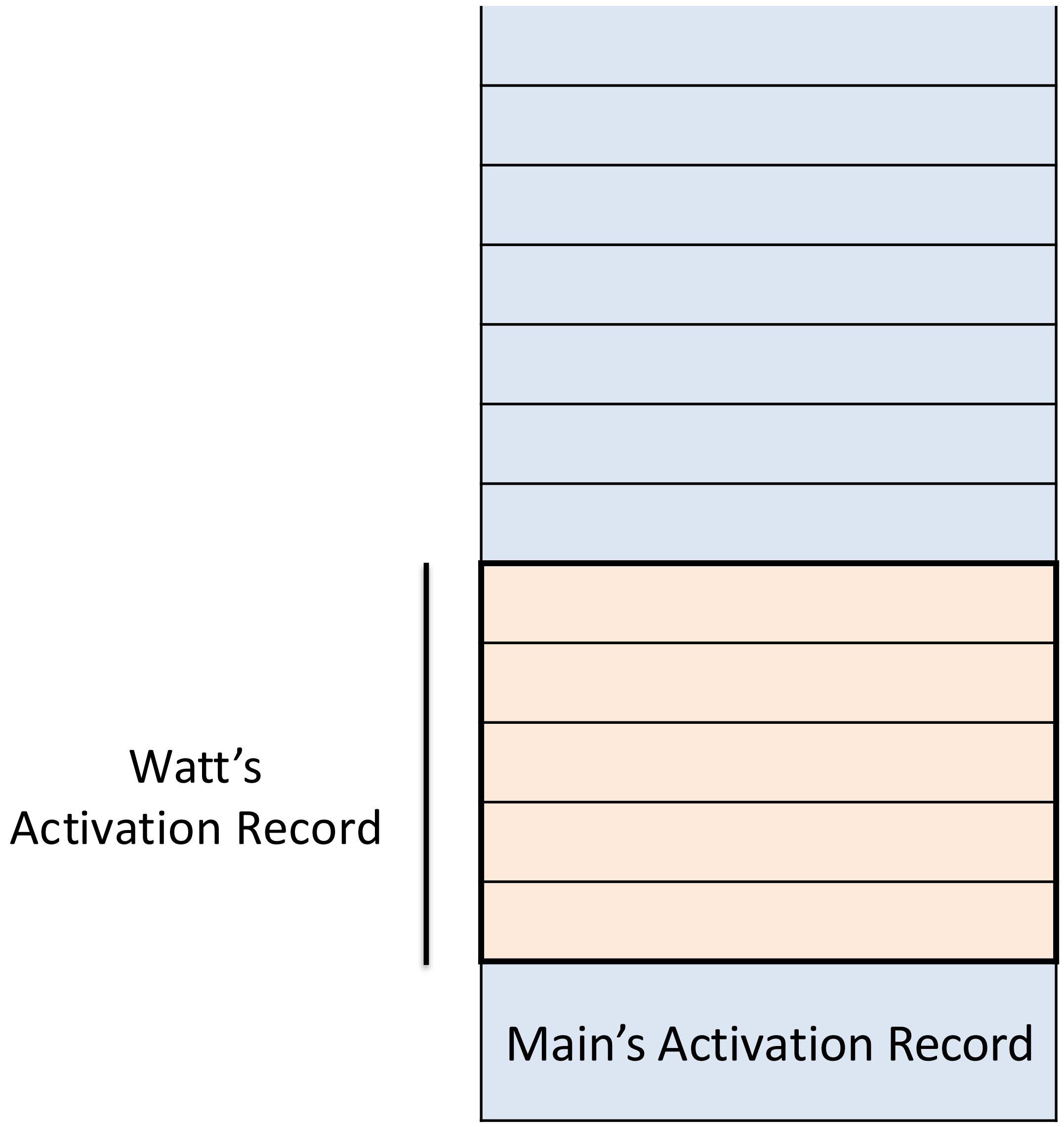
I ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

Lecture 7 Review: Run-Time Stack and Activation Record

```
int main(){
        int a;
        int b;
        …
        b = Watt(a);
        b = Volta(a,b);
        return 0;
}

int Watt(int a){
        int w;
        …
        w = Volta(w,10);
        return w;
}

int Volta(int q, int r){
        int k;
        int m;
        …
        return k;
}
```

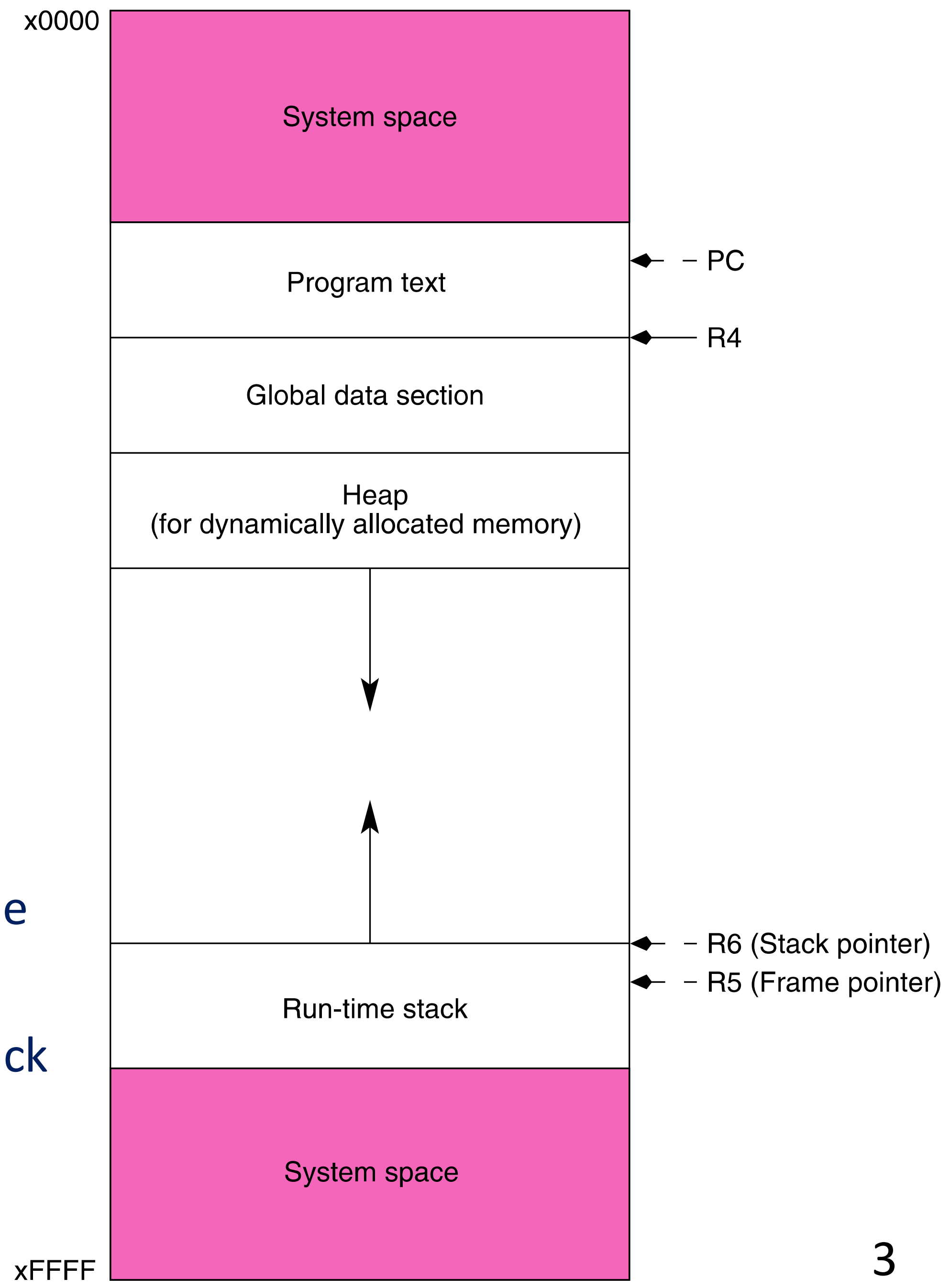


ECE ILLINOIS

Space for Variables

1. Global data section
   (global/static variables)
2. Run-time stack
   (local variables)
3. Heap
   (dynamically allocated variables)

R4 (**global pointer**) points to the first global variable
R5 (**frame pointer**) points to first local variable
R6 (**stack pointer**) points to the top of run-time stack



Symbol Table

- It contains name, type, location (as an offset), and scope.

```
int inGlobal;
int outGlobal;

int dummy(int in1, int in2);

int main(){
    int x,y,z;
    …

}

int dummy(int in1, int in2){
    int a,b,c;
    …
}
```

Name	Type	Location (as an offset)	Scope
inGlobal	int	0	global
outGlobal	int	1	global
x	int	0	main
y	int	-1	
z	int	-2	
a	int		
b	int		
c	int		

Stack Built-up and Tear-down

Caller function	**1. caller setup** (push callee's arguments onto stack)
	2. pass control to callee (invoke function)
Callee function	**3. callee setup** (push bookkeeping info and local variables onto stack)
	4. execute function
	5. callee teardown (pop local variables, caller's frame pointer, and return address from stack)
	6. return to caller
Caller function	**7. caller teardown** (pop callee's return value and arguments from stack)

Run-Time Stack Example

```
#include <stdio.h>
int Factorial(int n);

int main() {
   int number;
   int answer;
   …
   answer = Factorial(number);
   …
   return 0;
}

int Factorial(int n) {
   int i, result = 1;

   for(i=1; i<=n; i++){
      result = result*i;
   }
   return result;
}
```

x3FF7	
x3FF8	
x3FF9	
x3FFA	
x3FFB	
x3FFC	
x3FFD	
x3FFE	
x3FFF	answer
x4000	number

C to LC-3 Conversion with Run-Time Stack (RTS)

;; main program

; main code omitted here for simplicity

; assume R6 pointing to answer and R5 pointing to number on the RTS at this moment

; 1. Caller setup *(push callee's argument onto the RTS)*

```
; push number
```

; 2. Caller pass control to callee

; 7. Caller teardown *(pop callee's return value and argument from the RTS)*

```
; load return value at top of stack (R6)

; perform assignment: answer = Fact(number)

; pop return value and argument
```

```
FACTORIAL
        ; 3. Callee setup (push bookkeeping info & local variables onto the RTS)
        ; reserve space for return value

        ; push return address (R7)


        ; push caller's frame pointer (R5)


        ; set new frame pointer

        ; push local variables




        ; 4. Execute function (function logic omitted here for simplicity)
        …
```

; 5. Callee teardown *(pop local variables, C.F.P., and return addr from the RTS)*

```
; copy result into return value


; pop local variables

; pop caller’s frame pointer (into R5)


; pop return address (into R7)
```

; 6. Return to caller *(R6 should be pointing to return value when returning to caller)*

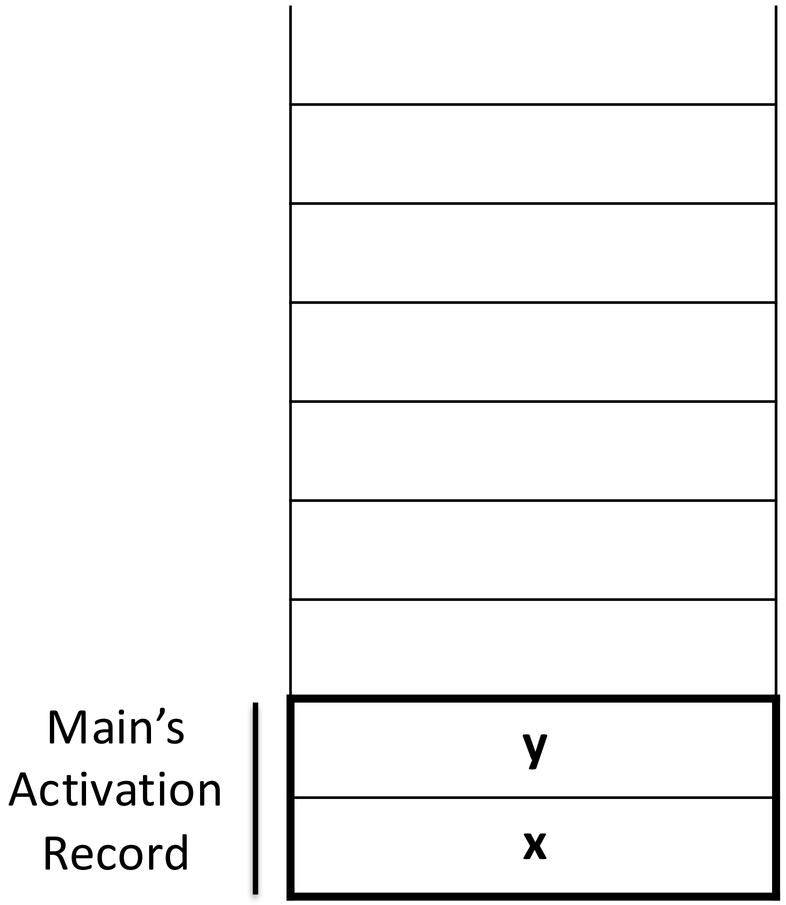
Swap Function

- Analyze the code below based on what we have learned so far about the Run-Time Stack. *Will the values of **x** and **y** be swapped in main after calling swap?*

```
void swap(int x, int y);

int main(){
        int x = 2;
        int y = 3;
        swap(x,y);
        return 0;
}

void swap(int x, int y){
        int temp;
        temp = x;
        x = y;
        y = temp;
}
```


Main's Activation Record
y
x