Review

- Conditional Constructs
  - If Statement
  - If-Else Statement

- Iteration Constructs
  - While Statement
  - Do-While Statement
  - For Statement
  - Break and Continue Statements
  - Switch Statement

Functions in C – Grammar

Lecture 28-1

Hyung-Sin Kim

SNU Graduate School of Data Science

You already know what functions are.

Today we focus more on

*(1) slightly different **grammar** in C*
*(2) what happens in the **memory** (LIFO stack)*
when you call a function in C

Functions in C – Grammar

- Definition
  - Header: Function name, parameter list including **data types**, and return value's **data type**
  - Any variable declared in the body is **local** to the function

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

int main(void) {
    int number;
    int answer;
    printf("Input a number: ");
    scanf("%d", &number);
    answer = Factorial(number);
    printf("The factorial of %d is %d\n", number, answer);
}

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

Functions in C – Grammar

- Definition
  - Header: Function name, parameter list including **data types**, and return value's **data type**
  - Any variable declared in the body is **local** to the function
- Call
  - The caller must transmit proper arguments (number, **data type**) to the callee
  - Calling is possible only when the callee is **declared** before (not necessarily **defined**)

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

int main(void) {
    int number;
    int answer;
    printf("Input a number: ");
    scanf("%d", &number);
    answer = Factorial(number);
    printf("The factorial of %d is %d\n", number, answer);
}

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

Functions in C – Grammar

- Definition
  - Header: Function name, parameter list including **data types**, and return value's **data type**
  - Any variable declared in the body is **local** to the function
  - Return value goes back to the caller
    - If there is no return, return type is **void**
- Call
  - The caller must transmit proper arguments (number, **data type**) to the callee
  - Calling is possible only when the callee is **declared** before (not necessarily **defined**)

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

int main(void) {
    int number;
    int answer;
    printf("Input a number: ");
    scanf("%d", &number);
    answer = Factorial(number);
    printf("The factorial of %d is %d\n", number, answer);
}

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

Functions in C – Grammar

- Definition
  - Header: Function name, parameter list including **data types**, and return value's **data type**
  - Any variable declared in the body is **local** to the function
  - Return value goes back to the caller
    - If there is no return, return type is **void**
- Call
  - The caller must transmit proper arguments (number, **data type**) to the callee
  - Calling is possible only when the callee is **declared** before (not necessarily **defined**)
- **Declaration** (function prototype)
  - Informs the compiler about relevant properties
    - Name, data type of return value, type of input arguments, and **semicolon**!

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

int main(void) {
    int number;
    int answer;
    printf("Input a number: ");
    scanf("%d", &number);
    answer = Factorial(number);
    printf("The factorial of %d is %d\n", number, answer);
}

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

Functions in C – Memory Operation Overview

Lecture 28-2

Hyung-Sin Kim

SNU Graduate School of Data Science

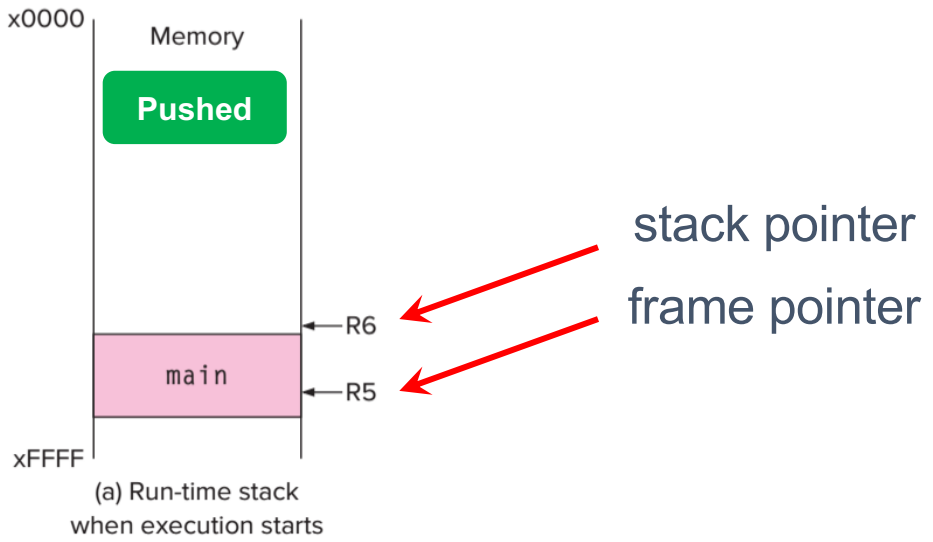
A Function Example in C

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);    // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);    // Watt calls Volt
    return w;
}

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

Memory Operation – Overview

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);   // Watt calls Volt
    return w;
}

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

Main is started!

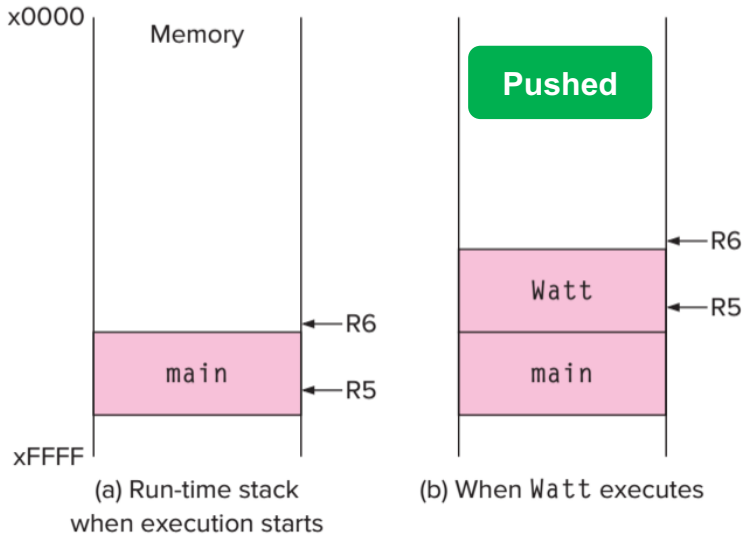


Memory Operation – Overview

```
int main(void) {
    int a;
    int b;
    b = Watt(a);     // main calls Watt first
    b = Volt(a, b);    // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);    // Watt calls Volt
    return w;
}

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

Main is started!

Main calls Watt!



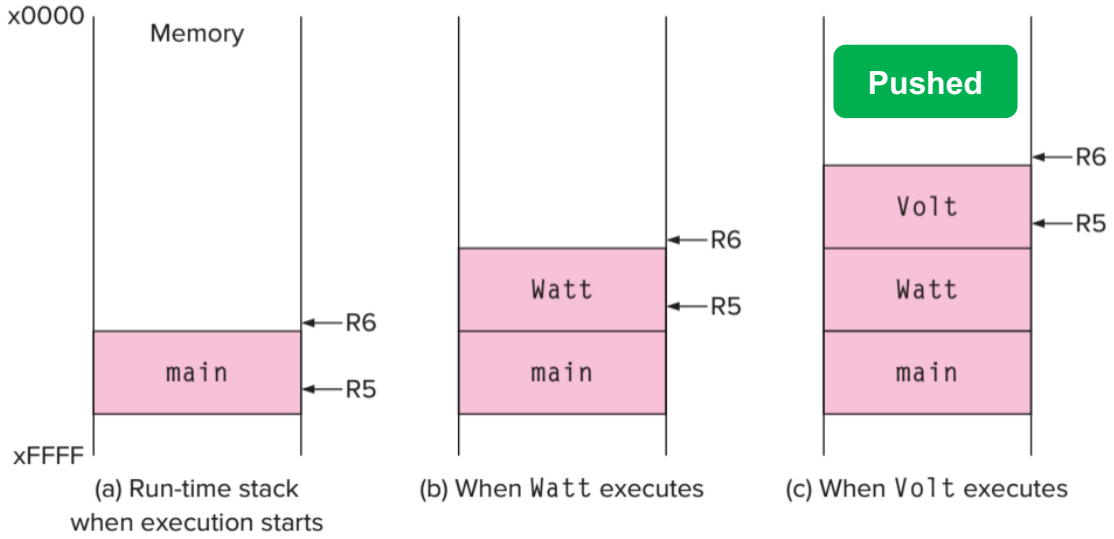
Memory Operation – Overview

```
int main(void) {
    int a;
    int b;
    b = Watt(a);     // main calls Watt first
    b = Volt(a, b);    // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);   // Watt calls Volt
    return w;
}

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

Main is started!

Main calls Watt!

Watt calls Volt!



Memory Operation – Overview

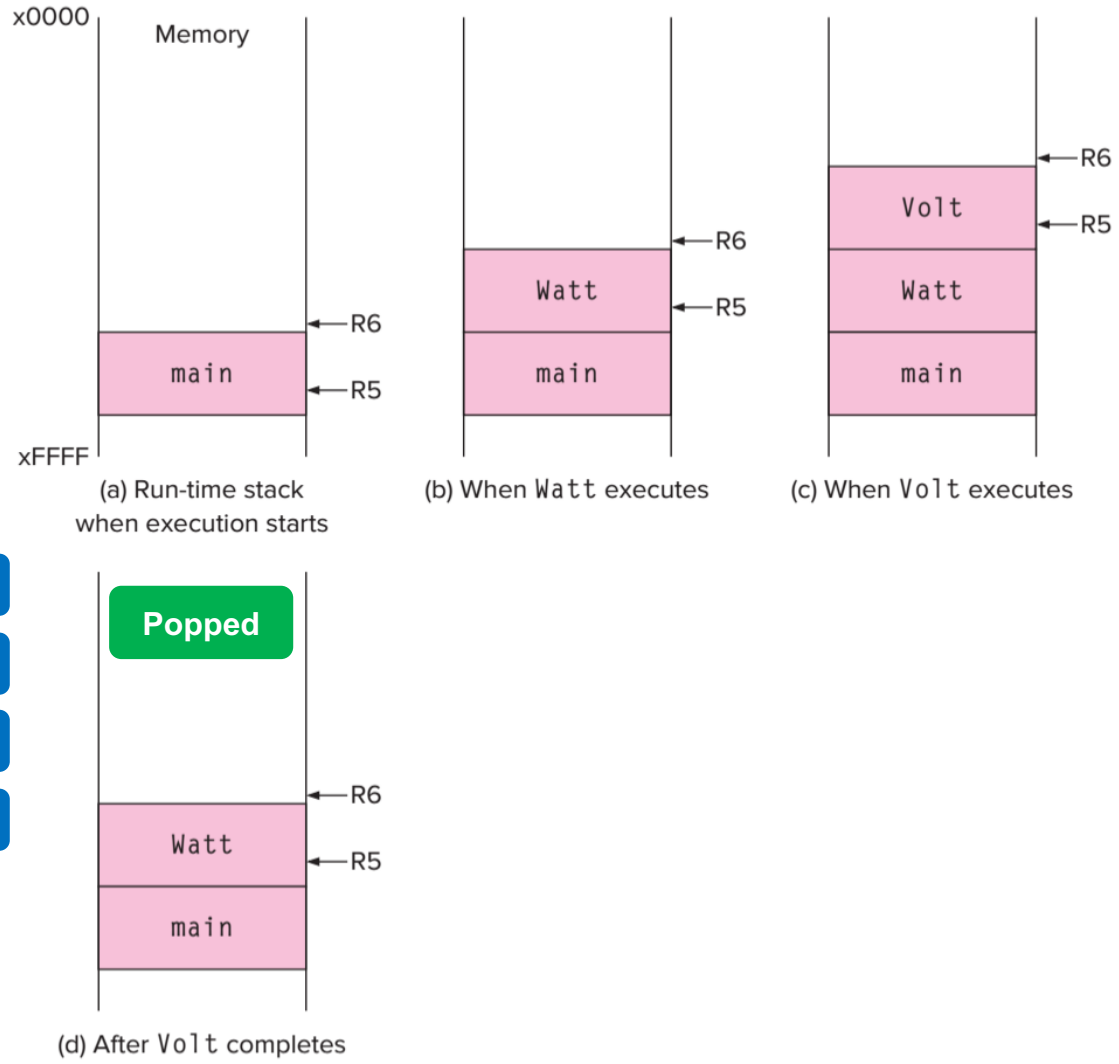
```
int main(void) {
    int a;
    int b;
    b = Watt(a);     // main calls Watt first
    b = Volt(a, b);    // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);    // Watt calls Volt
    return w;
}

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

Main is started!

Main calls Watt!

Watt calls Volt!

Volt (called by Watt) is terminated!



Memory Operation – Overview

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);    // Watt calls Volt
    return w;
}

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

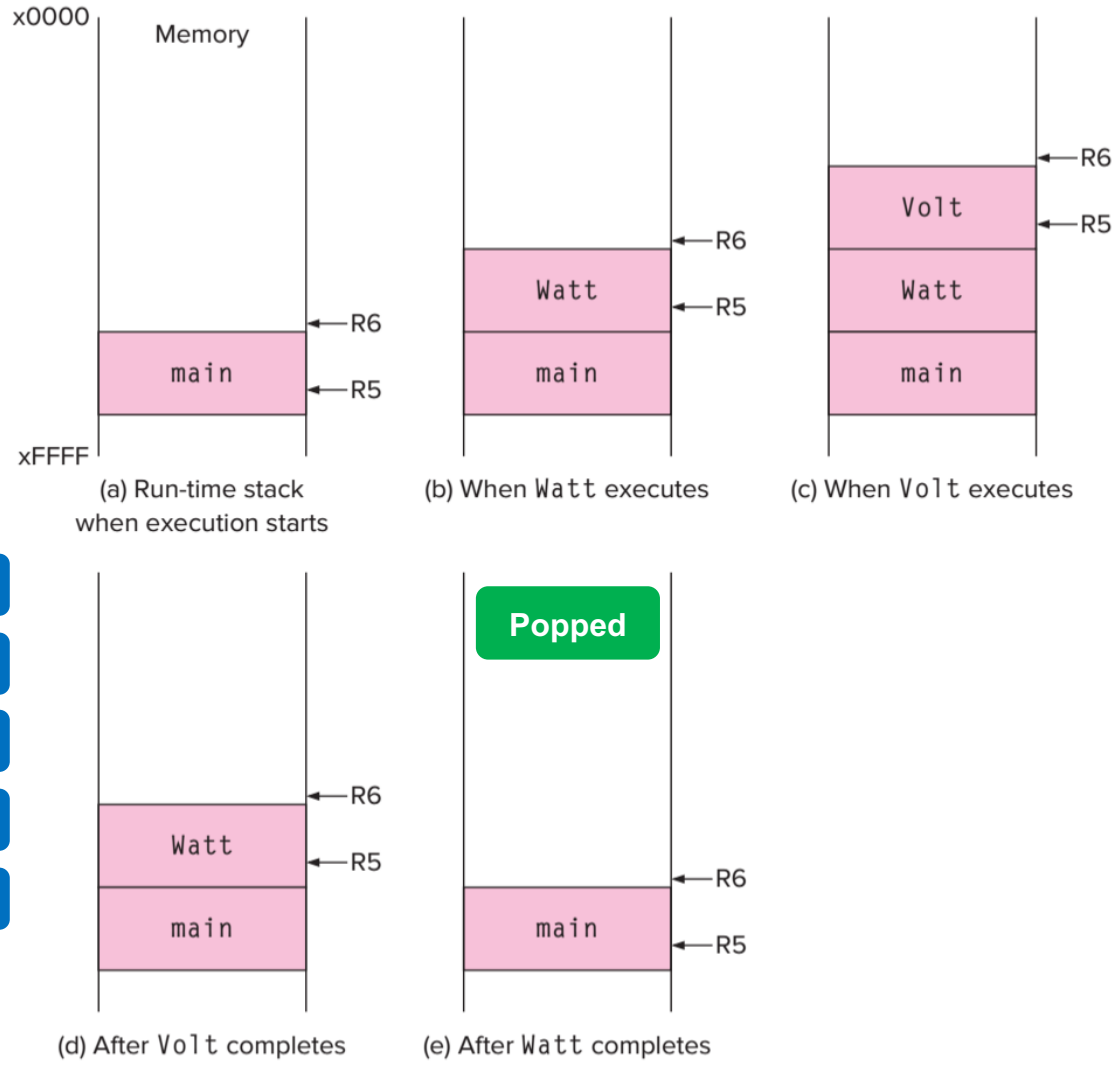
Main is started!

Main calls Watt!

Watt calls Volt!

Volt (called by Watt) is terminated!

Watt (called by main) is terminated!



Memory Operation – Overview

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);    // Watt calls Volt
    return w;
}

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

Main is started!

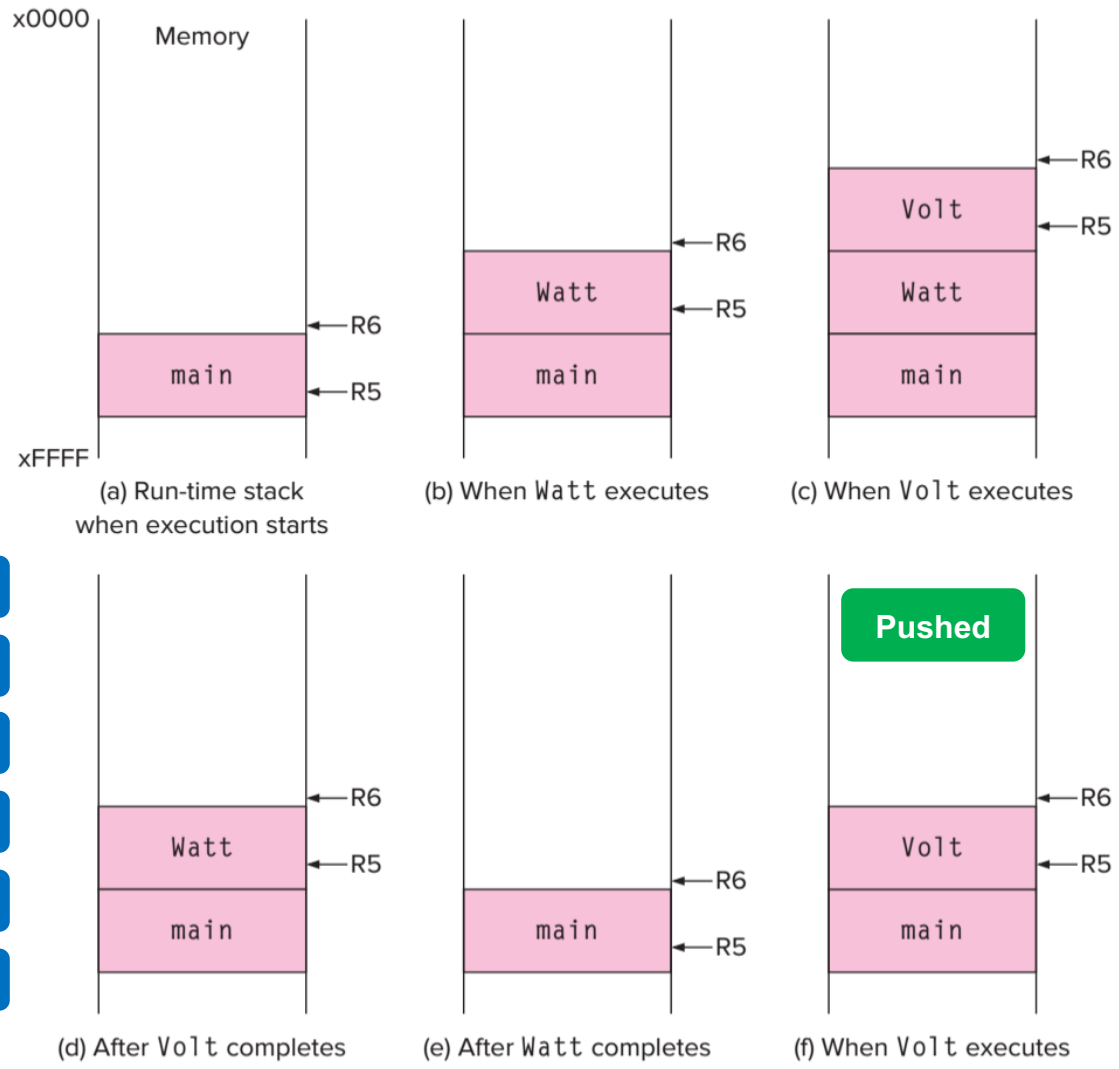
Main calls Watt!

Watt calls Volt!

Volt (called by Watt) is terminated!

Watt (called by main) is terminated!

Main calls Volt!



Computing Bootcamp

Functions in C – Memory Operation Details

Lecture 28-3

Hyung-Sin Kim

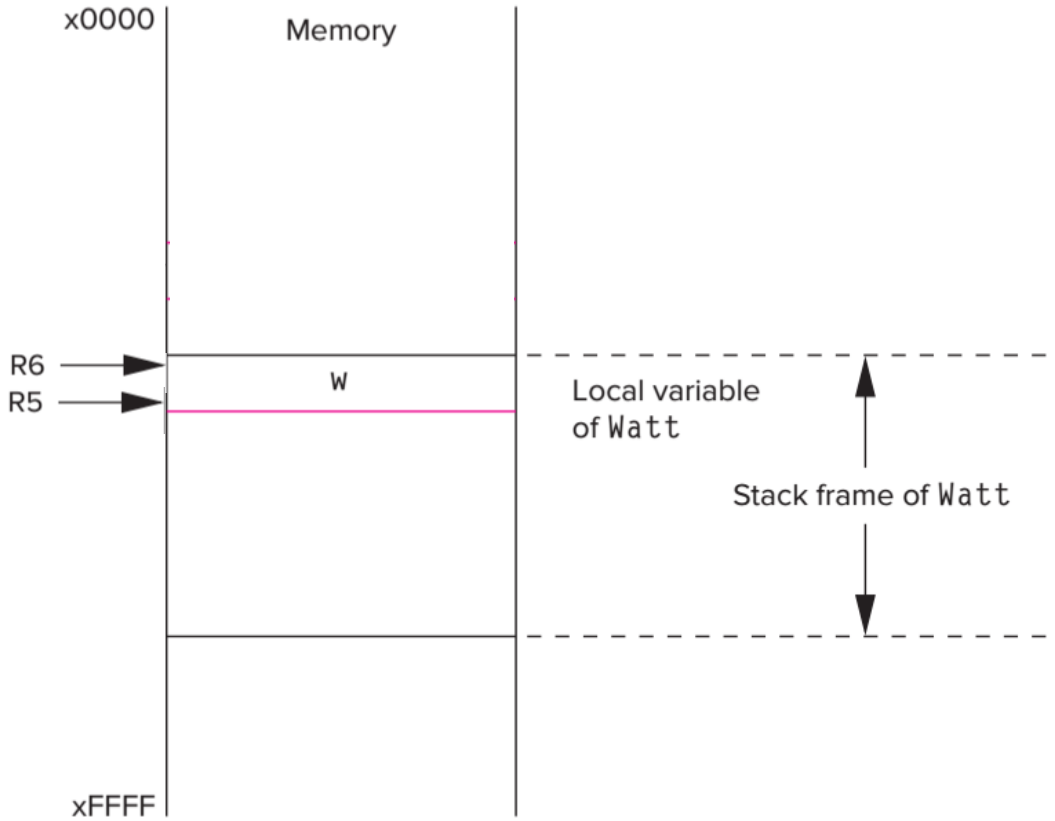
SNU Graduate School of Data Science

Let's take a deeper look into
each stage of a function's lifetime

(1) Calling (Passing arguments to the function)
(2) Start Callee (Reserve stack for the function)
(3) End Callee (Return)
(4) Return to Caller

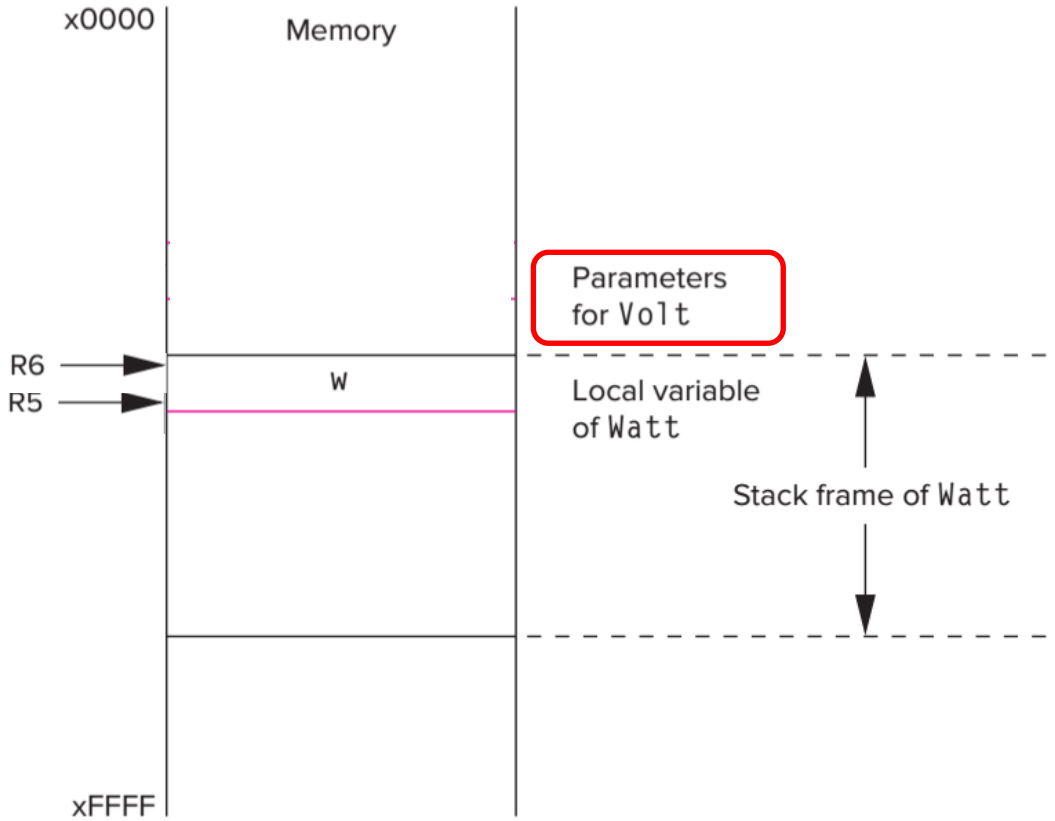
Detailed Memory Operation – Calling

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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



Detailed Memory Operation – Calling

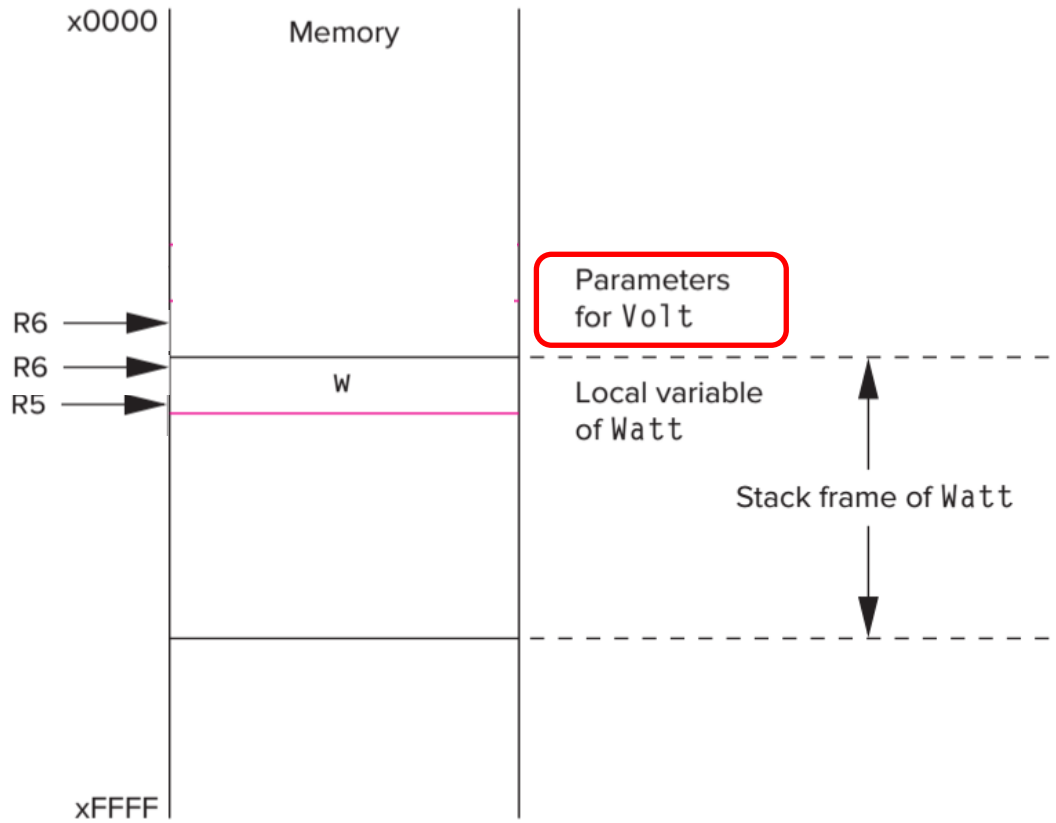
```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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



Detailed Memory Operation – Calling

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

(1) Decrement R6 and Push 10

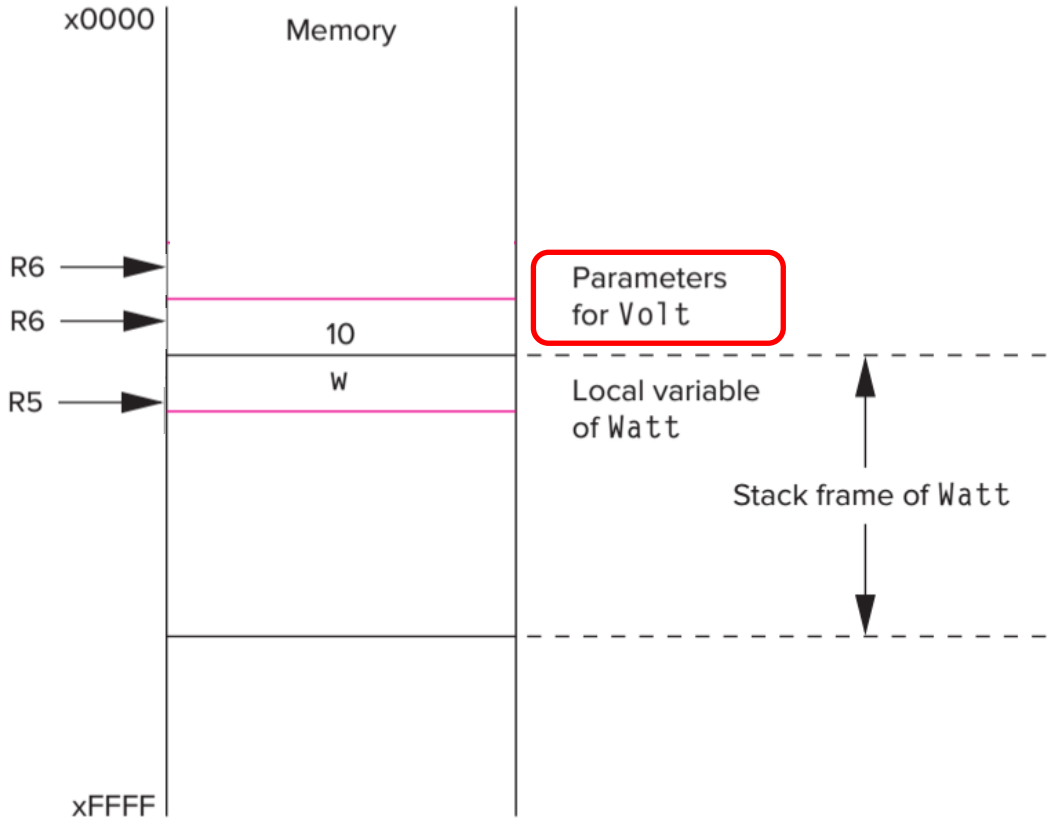


Detailed Memory Operation – Calling

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

(1) Decrement R6 and Push 10

(2) Decrement R6 and Push w value



Detailed Memory Operation – Calling

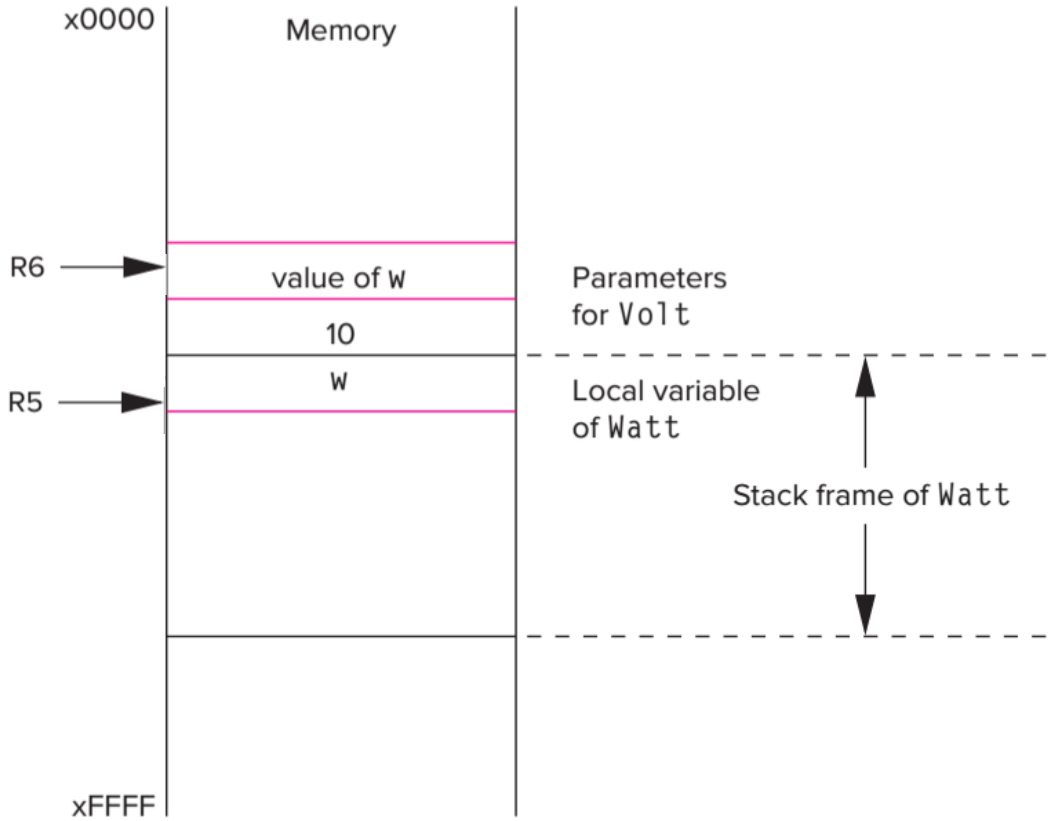
```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

PC

(1) Decrement R6 and Push 10

(2) Decrement R6 and Push w value

(3) Pass the control to Volt



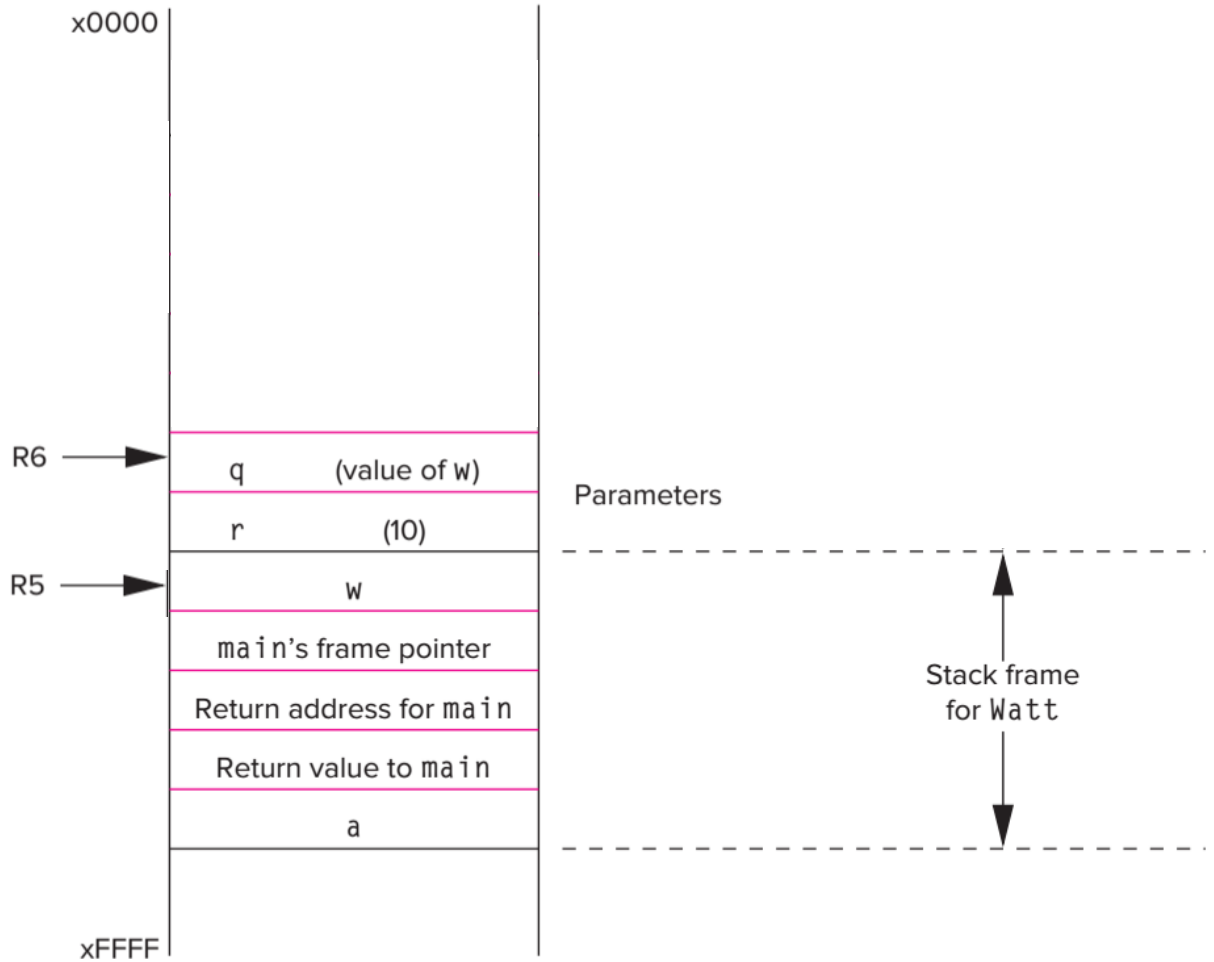
Let's take a deeper look into
each stage of a function's lifetime

(1) Calling (Passing arguments to the function)
(2) Start Callee (Reserve stack for the function)
(3) End Callee (Return)
(4) Return to Caller

Detailed Memory Operation – Start Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);   // Watt calls Volt
    return w;
}

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

(1) Parameters are already in the stack

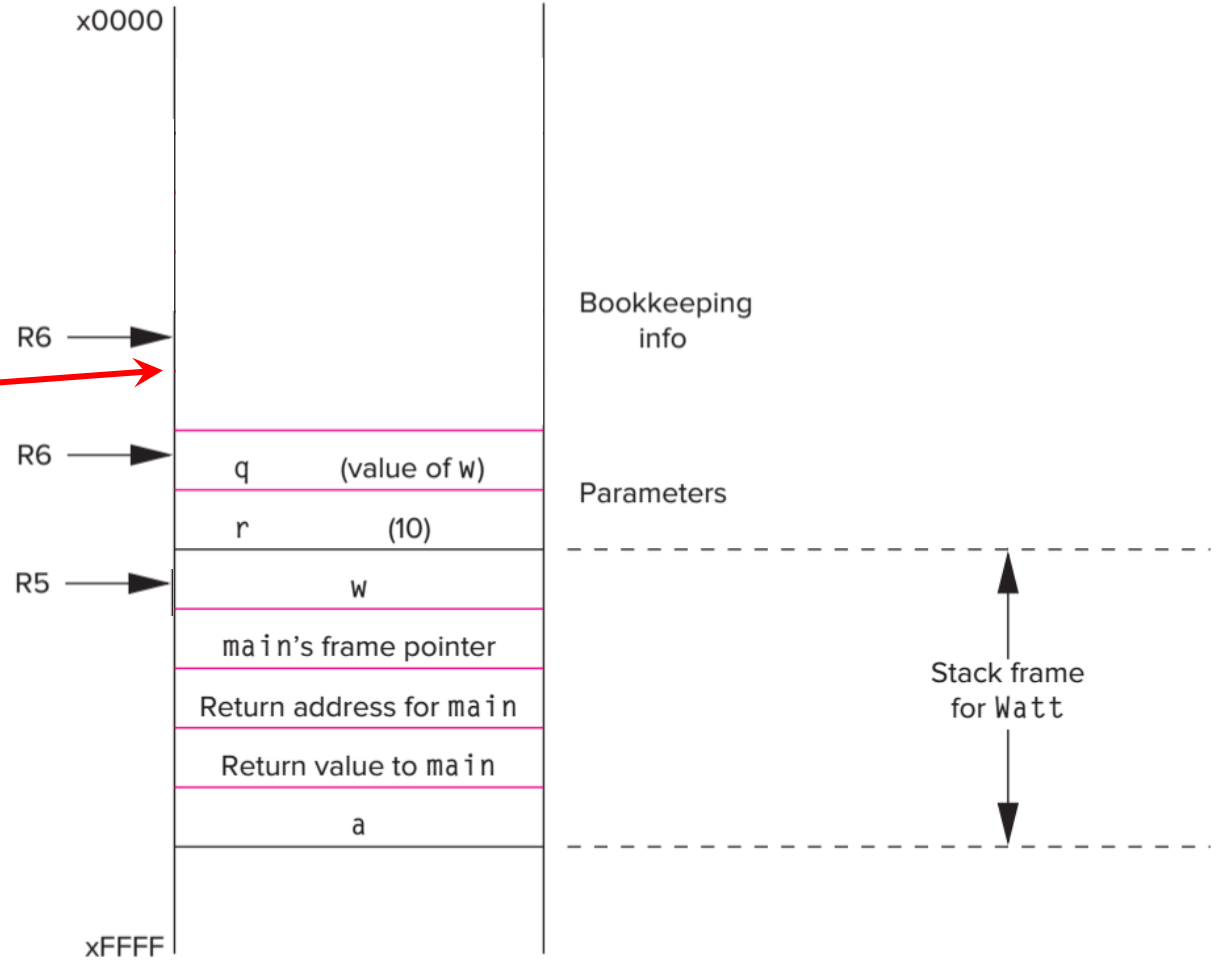


Detailed Memory Operation – Start Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

(1) Parameters are already in the stack

(2) Reserve memory for the return value and store return address



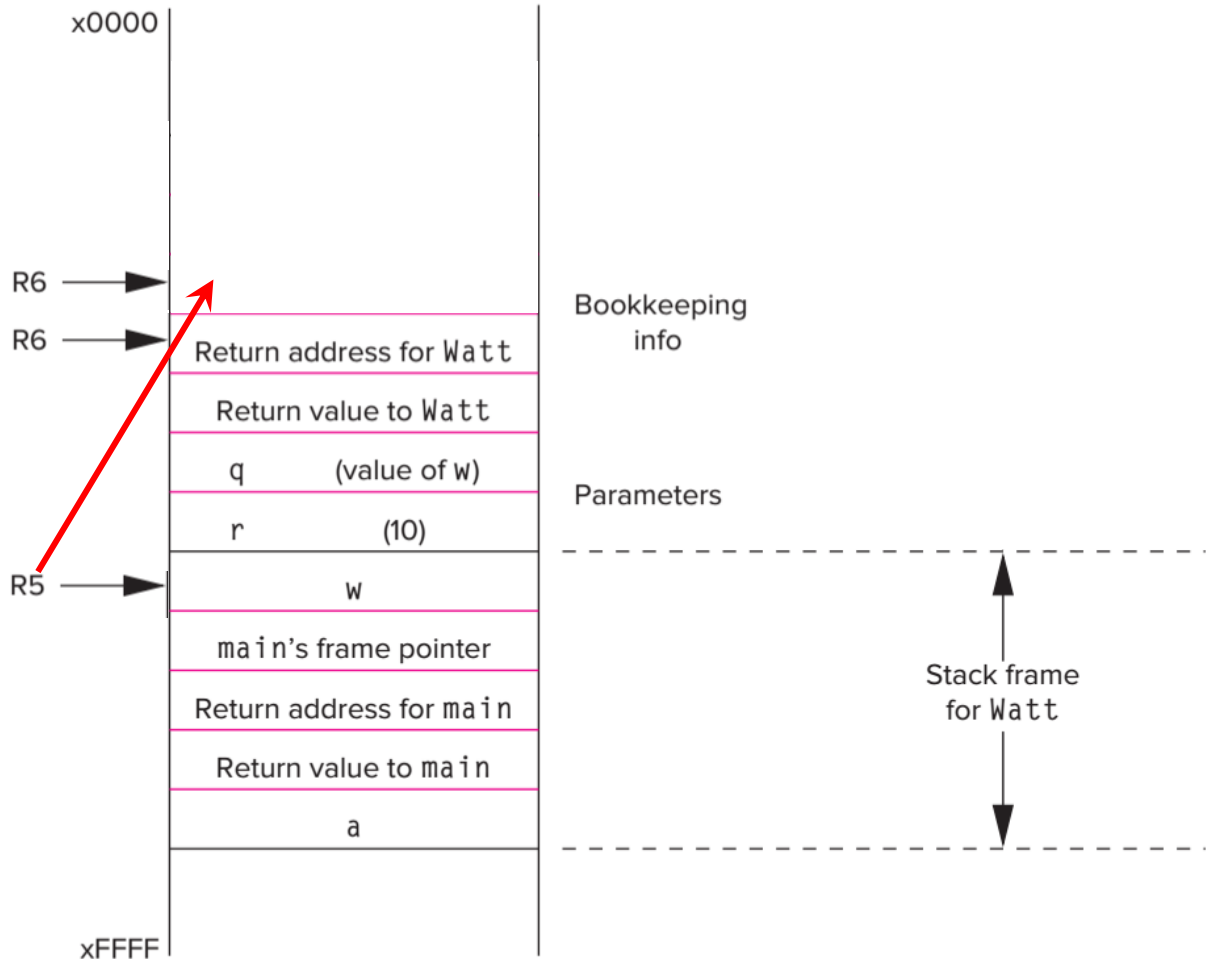
Detailed Memory Operation – Start Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

(1) Parameters are already in the stack

(2) Reserve memory for the return value and store return address

(3) Decrease R6 and copy Watt's frame pointer in R5 to where R6 points at



Detailed Memory Operation – Start Callee

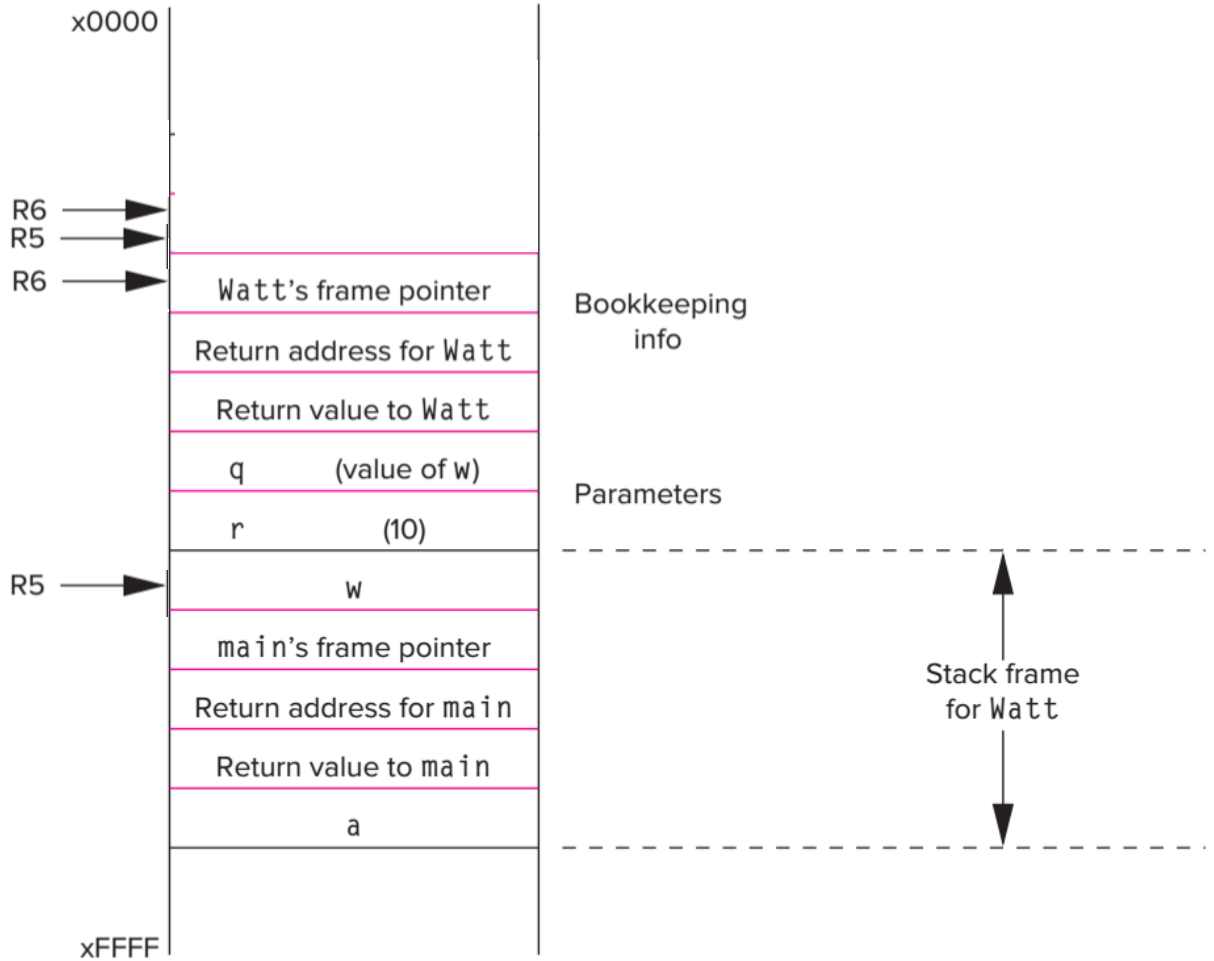
```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

(1) Parameters are already in the stack

(2) Reserve memory for the return value and store return address

(3) Decrease R6 and copy Watt's frame pointer in R5 to where R6 points at

(4) Decrease R6 and move the frame point (R5) to R6



Detailed Memory Operation – Start Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

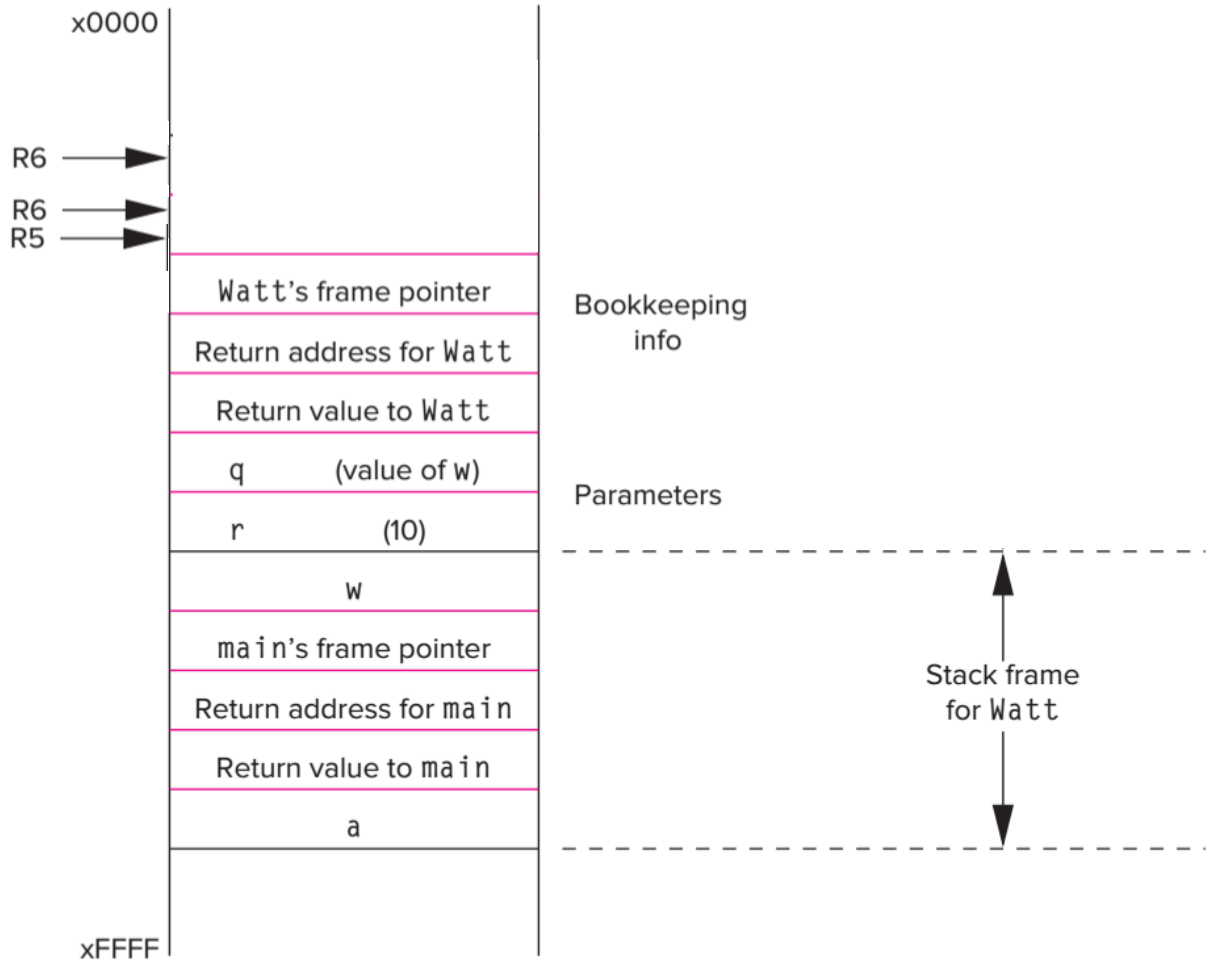
(1) Parameters are already in the stack

(2) Reserve memory for the return value and store return address

(3) Decrease R6 and copy Watt's frame pointer in R5 to where R6 points at

(4) Decrease R6 and move the frame point (R5) to R6

(5) Reserve memory for Volt's local variables



Detailed Memory Operation – Start Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);   // Watt calls Volt
    return w;
}

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

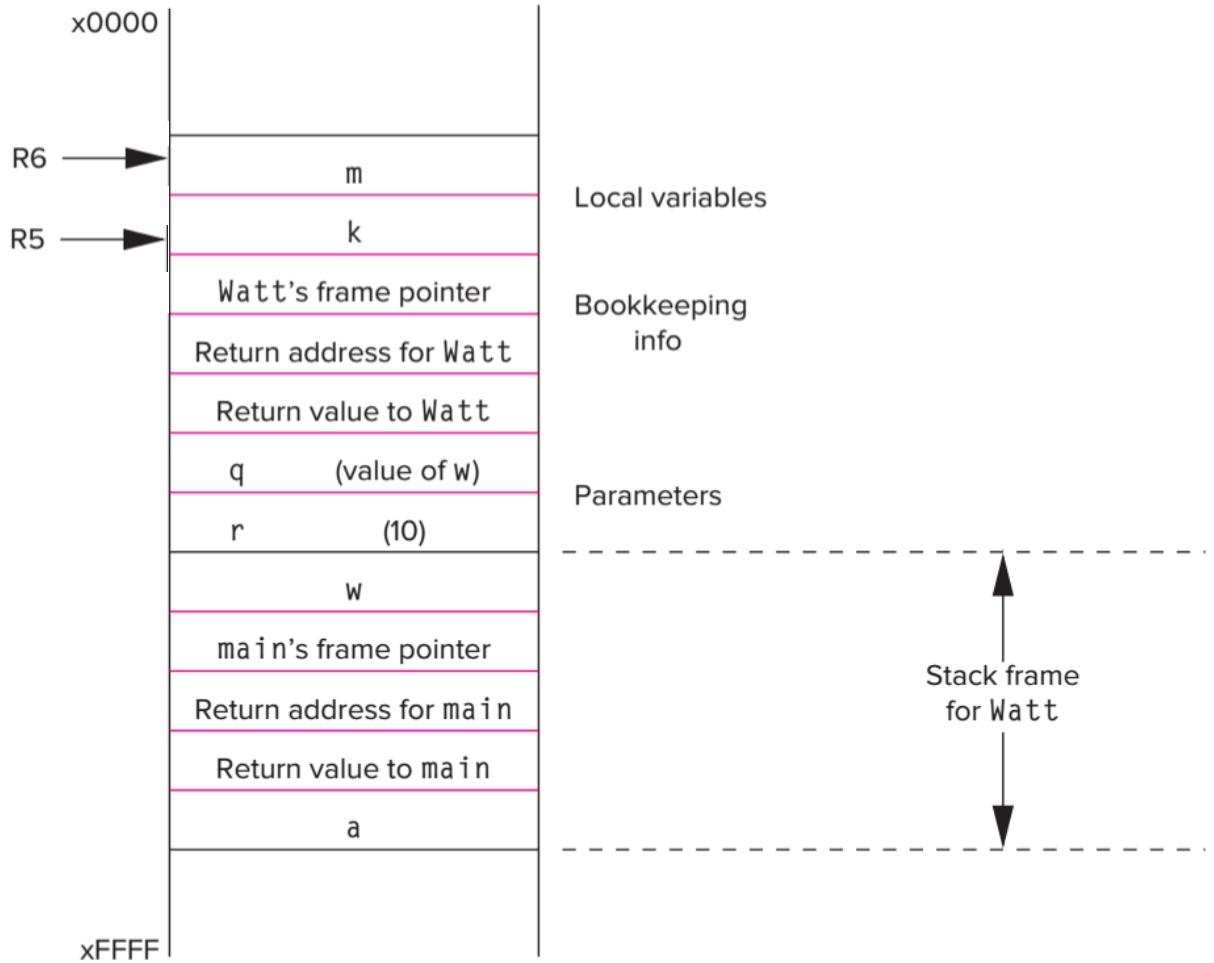
(1) Parameters are already in the stack

(2) Reserve memory for the return value and store return address

(3) Decrease R6 and copy Watt's frame pointer in R5 to where R6 points at

(4) Decrease R6 and move the frame point (R5) to R6

(5) Reserve memory for Volt's local variables



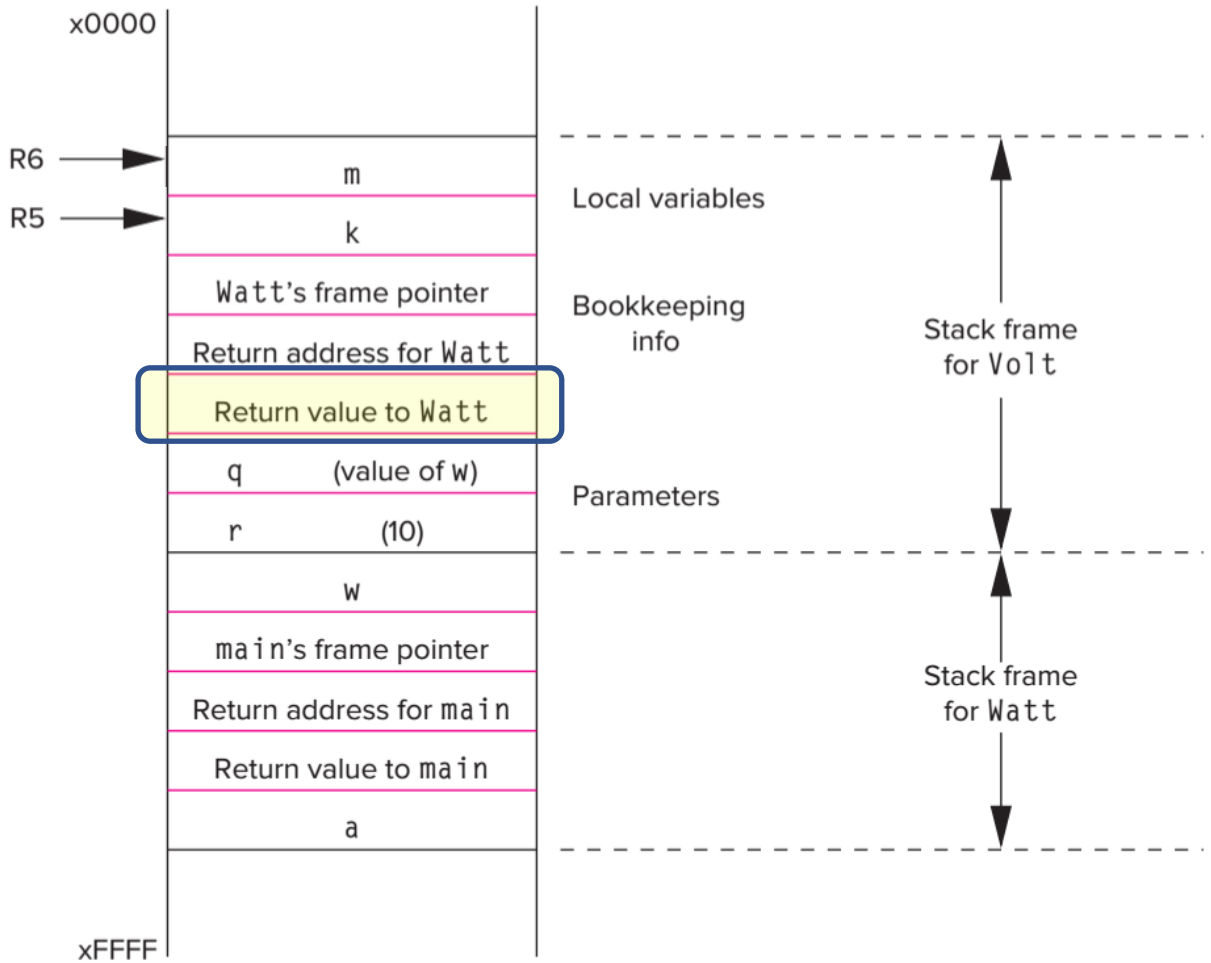
Let's take a deeper look into
each stage of a function's lifetime

(1) Calling (Passing arguments to the function)
(2) Start Callee (Reserve stack for the function)
(3) End Callee (Return)
(4) Return to Caller

Detailed Memory Operation – End Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);    // Watt calls Volt
    return w;
}

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

(1) Store the return value in R5+3

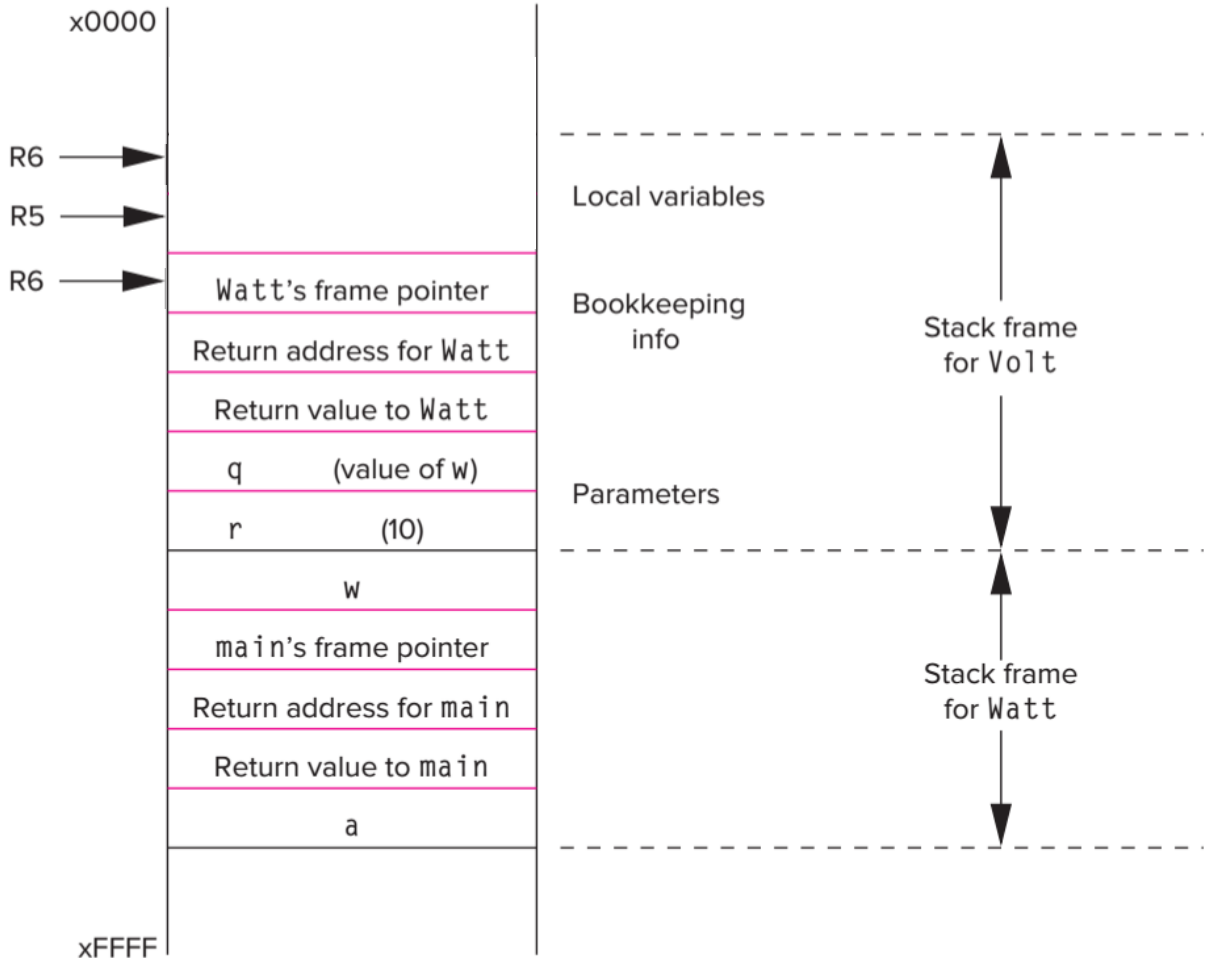


Detailed Memory Operation – End Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);   // Watt calls Volt
    return w;
}

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

(1) Store the return value in R5+3

(2) Pop the local variables (increase R6 by 2)



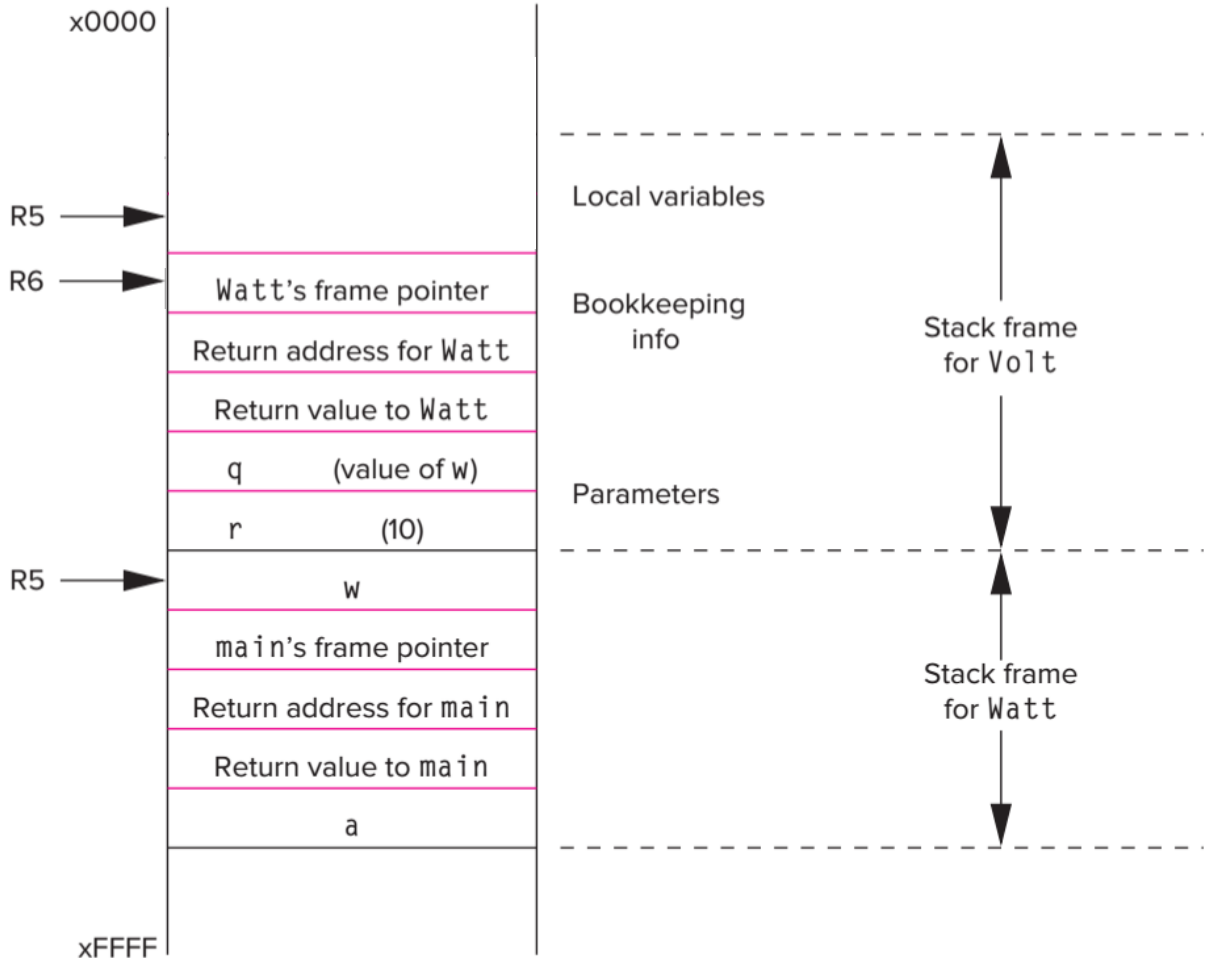
Detailed Memory Operation – End Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);    // Watt calls Volt
    return w;
}

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

(1) Store the return value in R5+3

(2) Pop the local variables (increase R6 by 2)

(3) Restore Watt's frame pointer (Push R6's value to R5) and pop



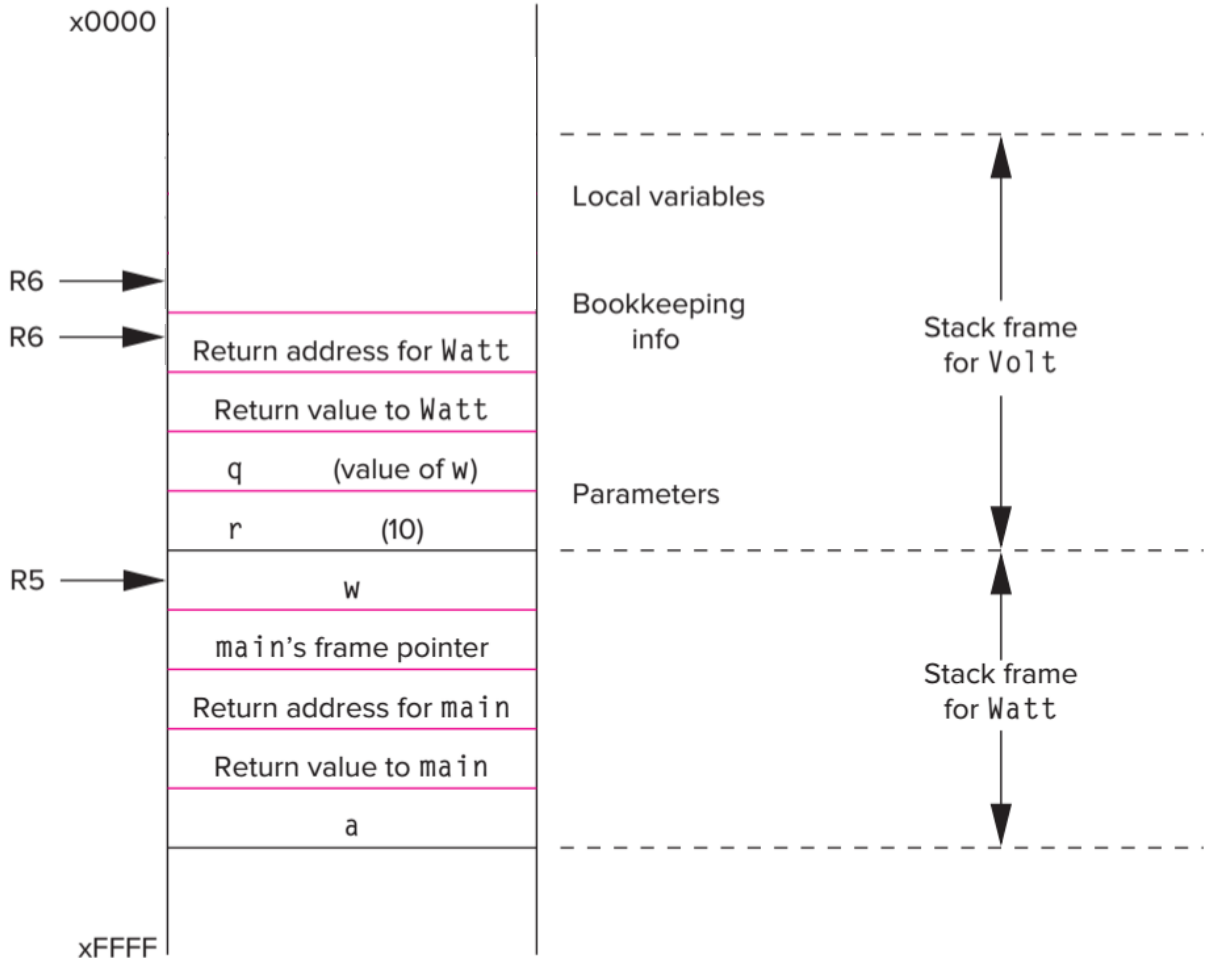
Detailed Memory Operation – End Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);   // Watt calls Volt
    return w;
}

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

(1) Store the return value in R5+3

(2) Pop the local variables (increase R6 by 2)

(3) Restore Watt's frame pointer (Push R6's value to R5) and pop



Detailed Memory Operation – End Callee

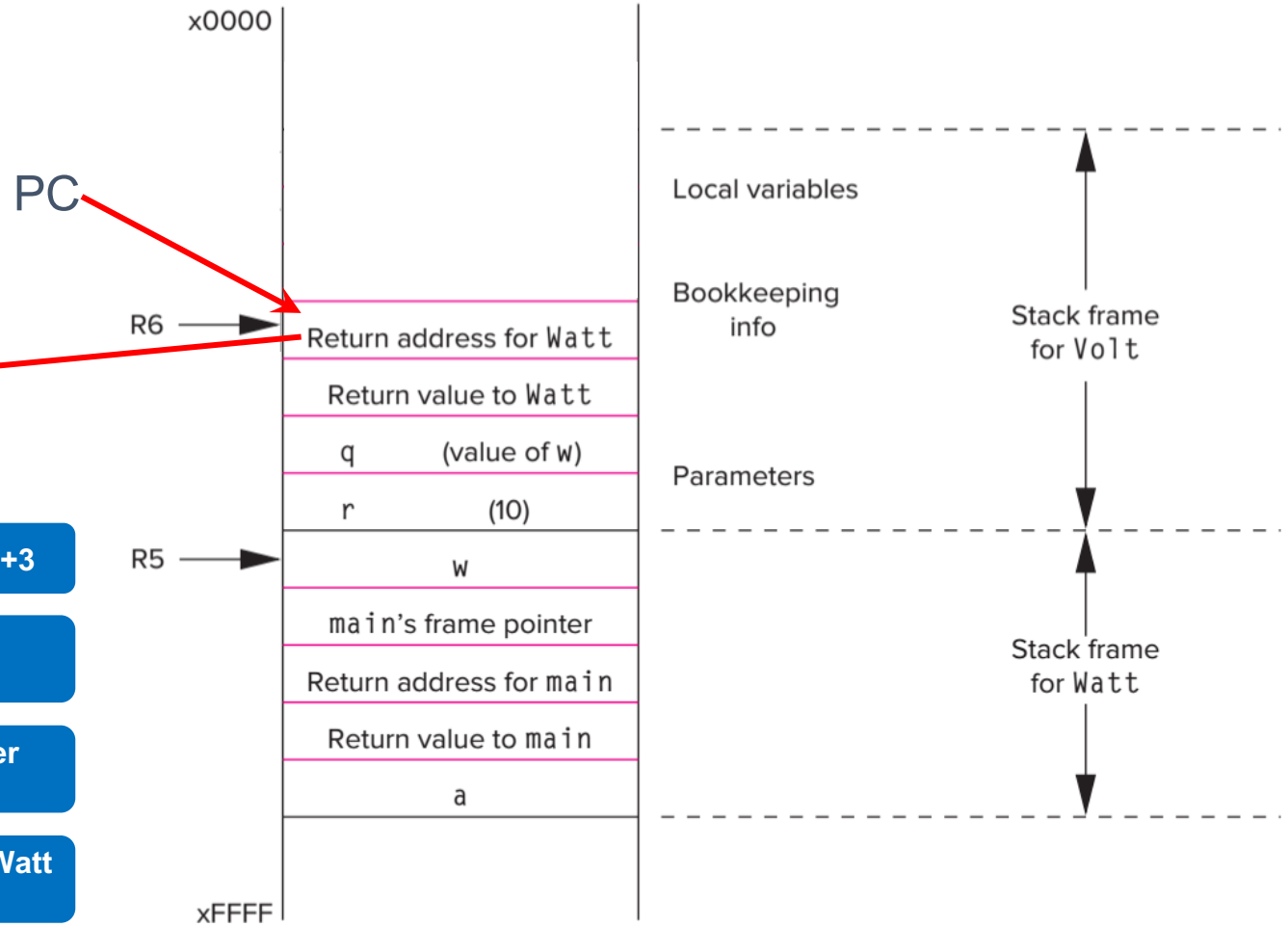
```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

(1) Store the return value in R5+3

(2) Pop the local variables (increase R6 by 2)

(3) Restore Watt's frame pointer (Push R6's value to R5)

(4) Transfer the control to Watt (PC = Return value) and pop



Detailed Memory Operation – End Callee

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

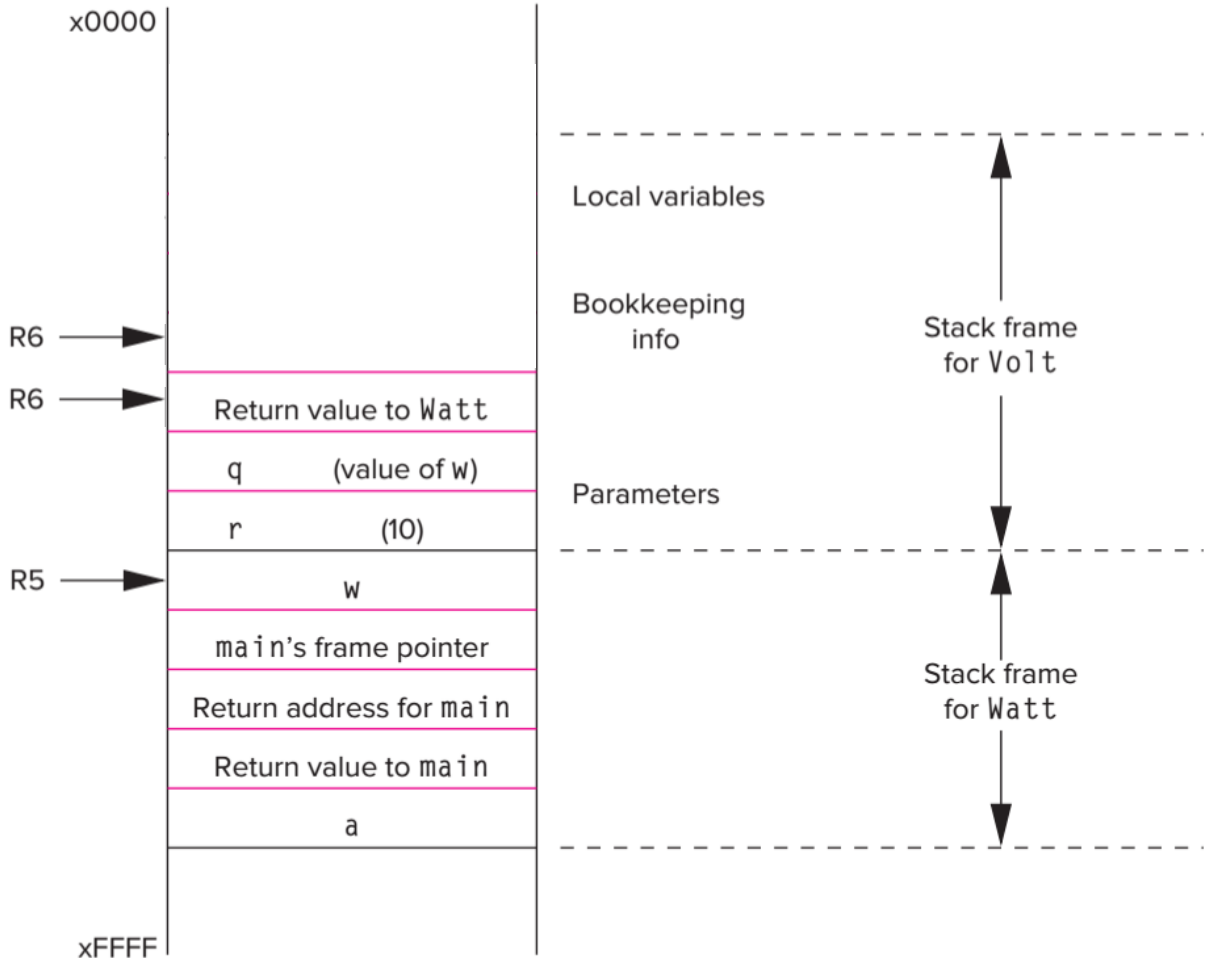
PC

(1) Store the return value in R5+3

(2) Pop the local variables (increase R6 by 2)

(3) Restore Watt's frame pointer (Push R6's value to R5)

(4) Transfer the control to Watt (PC = Return value) and pop



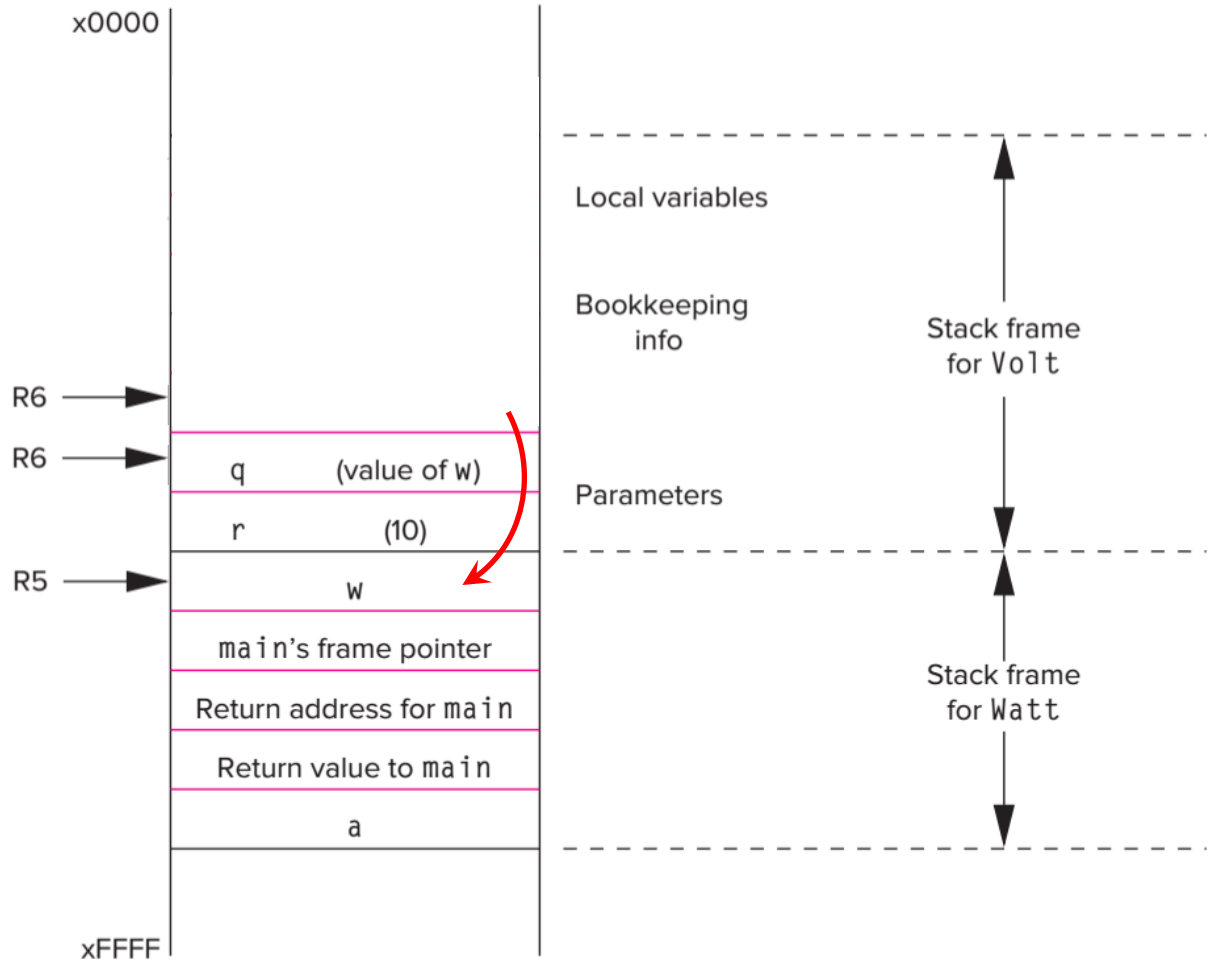
Let's take a deeper look into
each stage of a function's lifetime

(1) Calling (Passing arguments to the function)
(2) Start Callee (Reserve stack for the function)
(3) End Callee (Return)
(4) Return to Caller

Detailed Memory Operation – Return to Caller

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

(1) Load the return value (R6) to w and pop

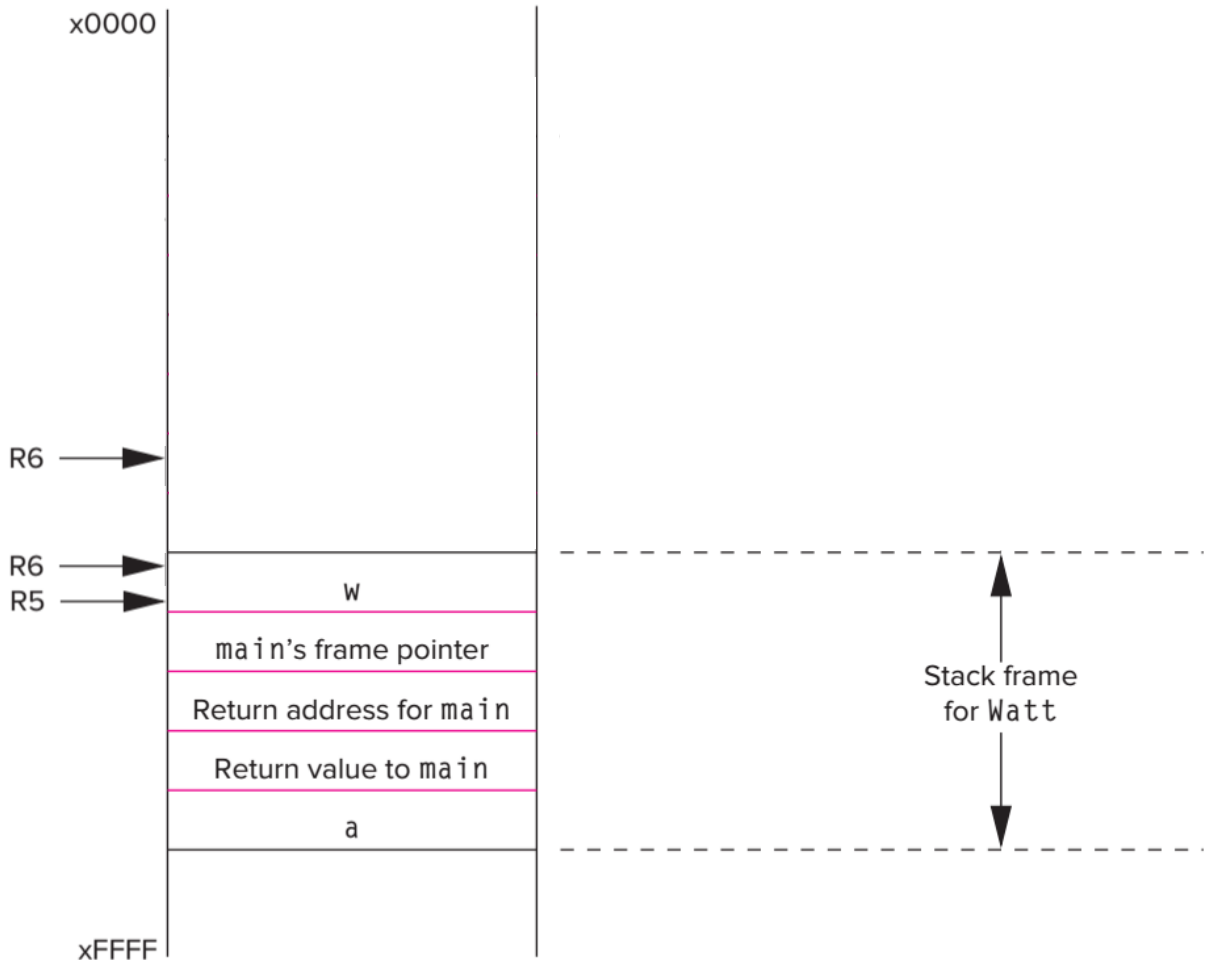


Detailed Memory Operation – Return to Caller

```
int main(void) {
    int a;
    int b;
    b = Watt(a);      // main calls Watt first
    b = Volt(a, b);   // then calls Volt
}

int Watt(int a) {
    int w;
    w = Volt(w, 10);  // Watt calls Volt
    return w;
}

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

(1) Load the return value (R6) to w and pop

(2) Pop the arguments



Summary

- Compared to Python, a few more things need to be considered to play with functions in C
  - You need to write **data types** of its parameters and return value explicitly in the function header
  - You can **declare** a function using its prototype before its full definition

- C uses **stack** to manage memory operation of a function call
  - We have seen why using stack (LIFO data structure) makes sense
  - We have seen how all the information is pushed and popped correctly using only the two pointers (stack pointer and frame pointer)

Functions in C – Examples

Lecture 28-4

Hyung-Sin Kim


SNU Graduate School of Data Science

Problem

- Given two integers a, b **recursively** find its greatest common divisor (GCD)
- Ex)
  - Input: $a = 10, b = 45$
  - Output: 5

  - Input: $a = 128, b = 96$
  - Output: 32

Thanks!