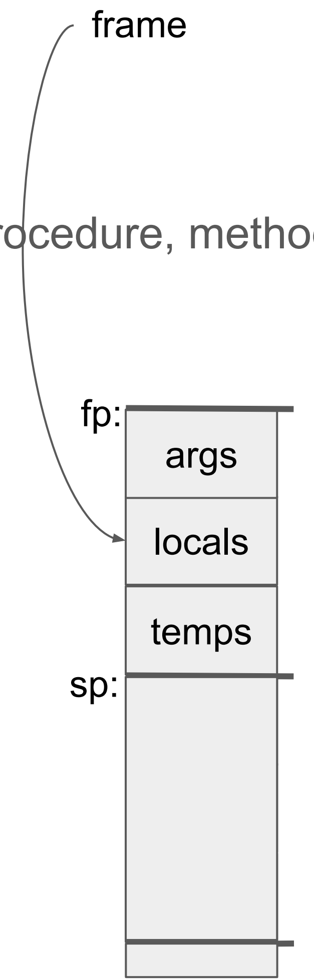
Subroutine Calls

- Subroutine: generic term for: function, procedure, method, whatever!
- If you are a good programmer, you have
  - many or few subroutines?
- Frame of Memory for a subroutine
  - input arguments are placed onto the stack
  - local variables are placed onto the stack
  - any tempories are placed onto the stack
- Where does the return value go?



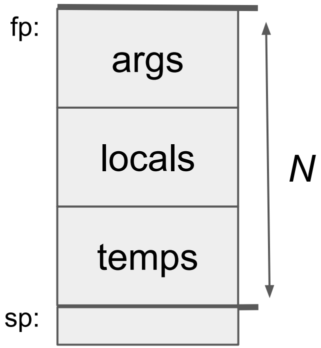

```
int f (int A1, int A2, int A3);
int g (int A1);

int f(int A1,int A2, int A3) {
    int L1;

     L1 = <some calculation>
   {
     int L3 ;

     L3 = A1 * L1 + g(A2+L2);
     L1 = L3 | A3

   }
   return (L1);

}


main() {

   y= f(a, b, c);
```

- Remember Memory is SLOW!

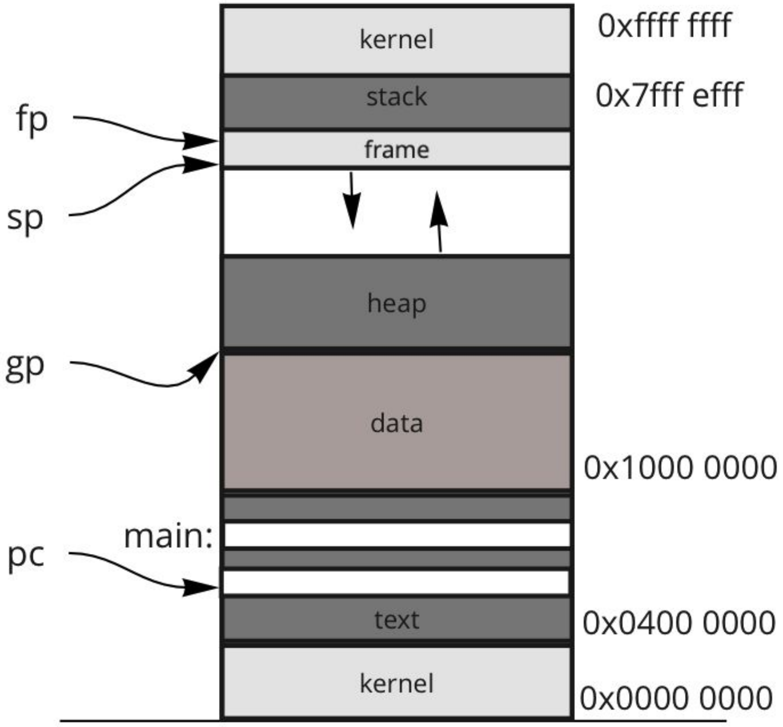
CPU General Purpose Registers: ($0 -- $31)

- $t0 - $t9: temporary registers
- $zero: holds the value 0
    - All needed literals must be stored in memory
    - But memory is slow, so keep 0 in a register
- $s0 - $7: saved temporary registers
- Subroutine Specific
    - $v0 - $v1: return values
    - $a0 - $a3: input arguments
- Special Usage:
    - $sp: stack pointer
    - $fp: frame pointer
    - $ra: return address

x = f(a, b)



Memory Organization



General Purpose Registers

Name	Register Number	Usage
`$zero`	0	constant 0 (hardware)
`$v0 - $v1`	2 - 3	subroutine return values
`$a0 - $a3`	4 - 7	subroutine arguments
`$t0 - $t7`	8 - 15	temporaries
`$s0 - $s7`	16 - 23	saved temporaries
`$t8 - $t9`	34 - 35	temporaries
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address (hardware)

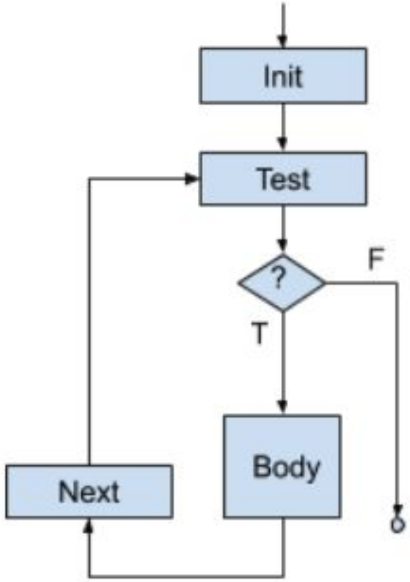
Recall: Control Flow Graph

- A graphic representation of the representation between basic blocks
- A basic block:
    - a list of instructions with
    - a single entry point (starting point)
    - a single exit point (last instruction)
- Such representations model the behavior of our code
- Recall the while loop, and other control structures

- What about subroutines calls

    (subroutine: general term for …

        methods, functions, procedures, etc.)

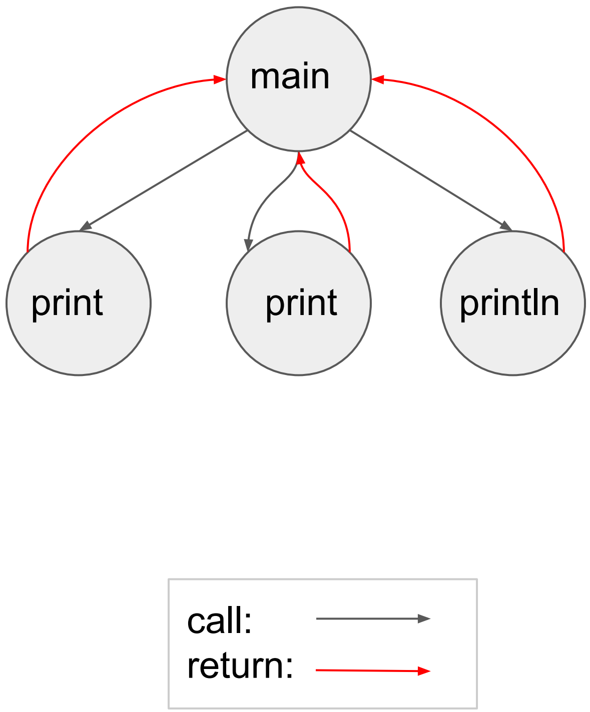


While Loop

Call Graph

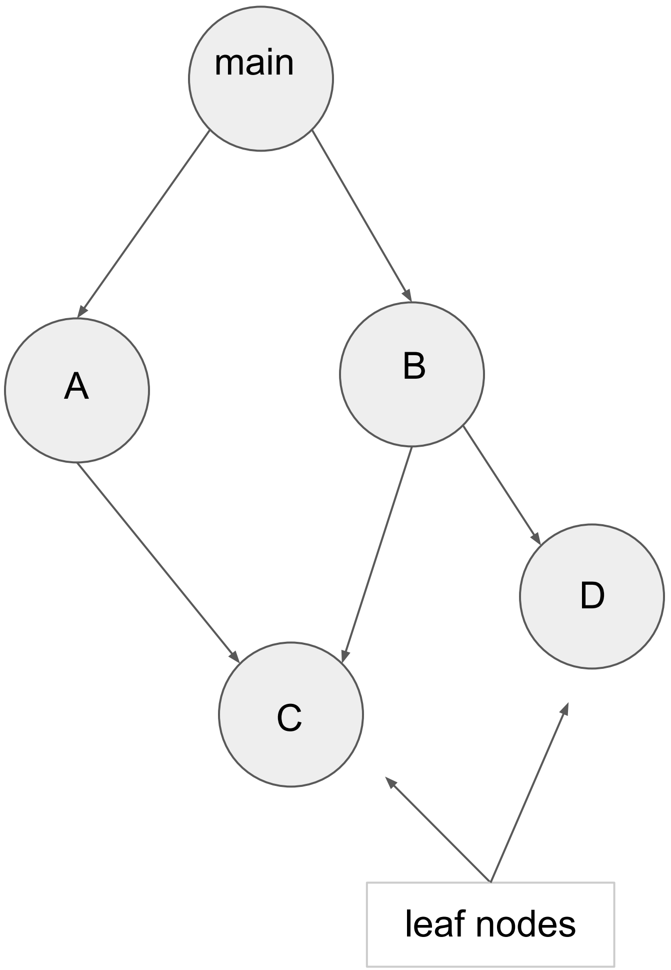
- a control flow graph depicting the relationships between subroutines
- Call Graph for the "Hello World" program

```
class HelloWorld
{
   public static void main(String args[])
   {
      System.out.print("Hello ");
      System.out.print("World");
      System.out.println("");
   }
}
```



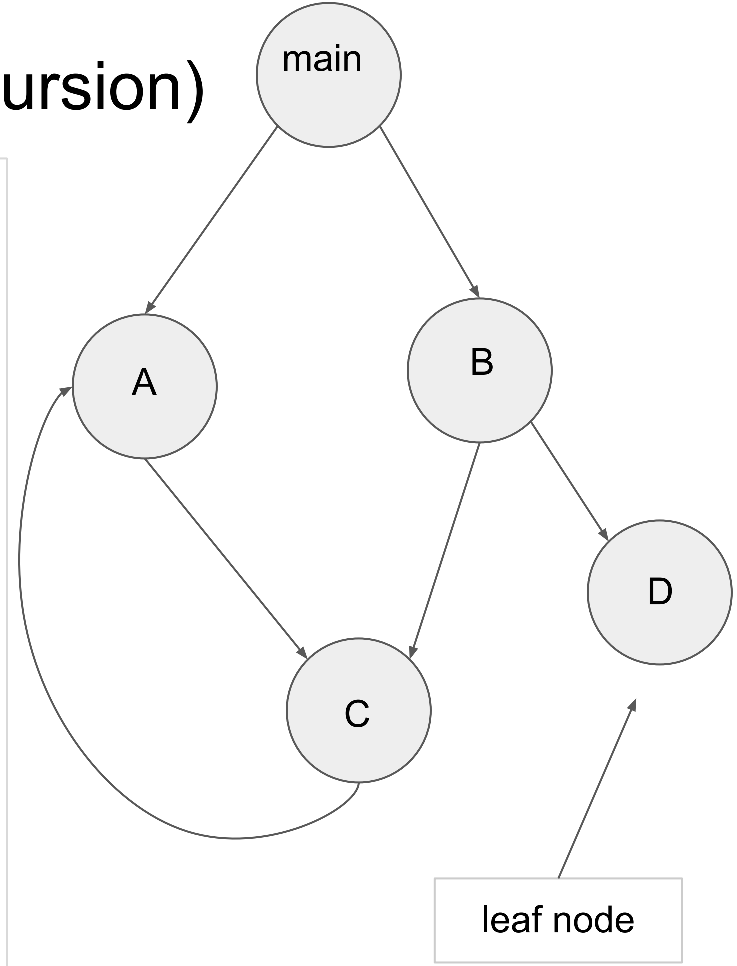
Call Graph II

```
public static void A(void) {
    int x = 5;
    C();
}
public static void B(void) {
    C();
    D();
}
public static void C(void) {
    ;
}
public static void D(void) {
    ;
}

public static void main(String args[])
   {
      A();
      B();
   }
}
```



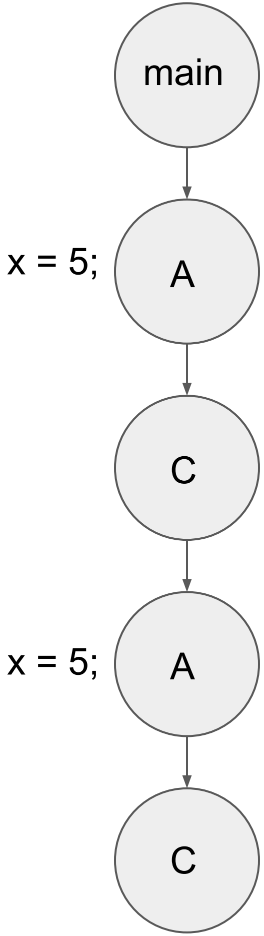
Call Graph with a Loop (Recursion)

```
public static void A(void) {
    int x = 5;
    C();
}
public static void B(void) {
    C();
    D();
}
public static void C(void) {
   A();
}
public static void D(void) {
    ;
}

public static void main(String args[])
   {
      A();
      B();
   }
}
```

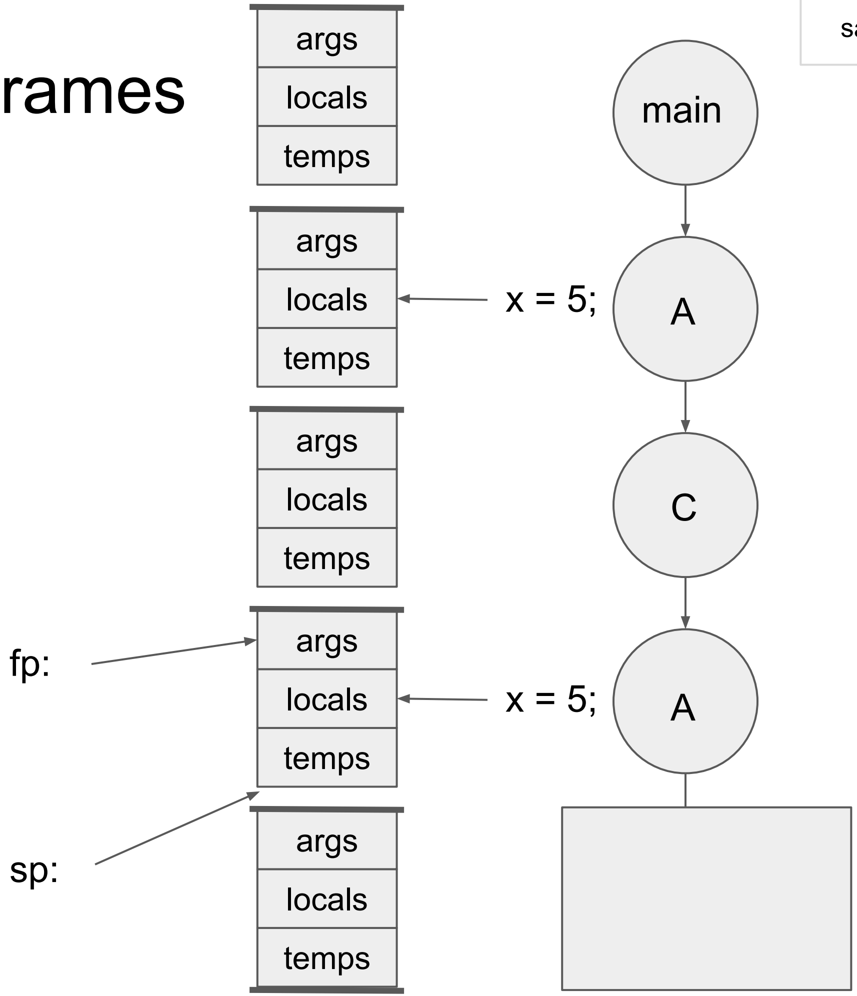


Dynamic Call Graph (Runtime)

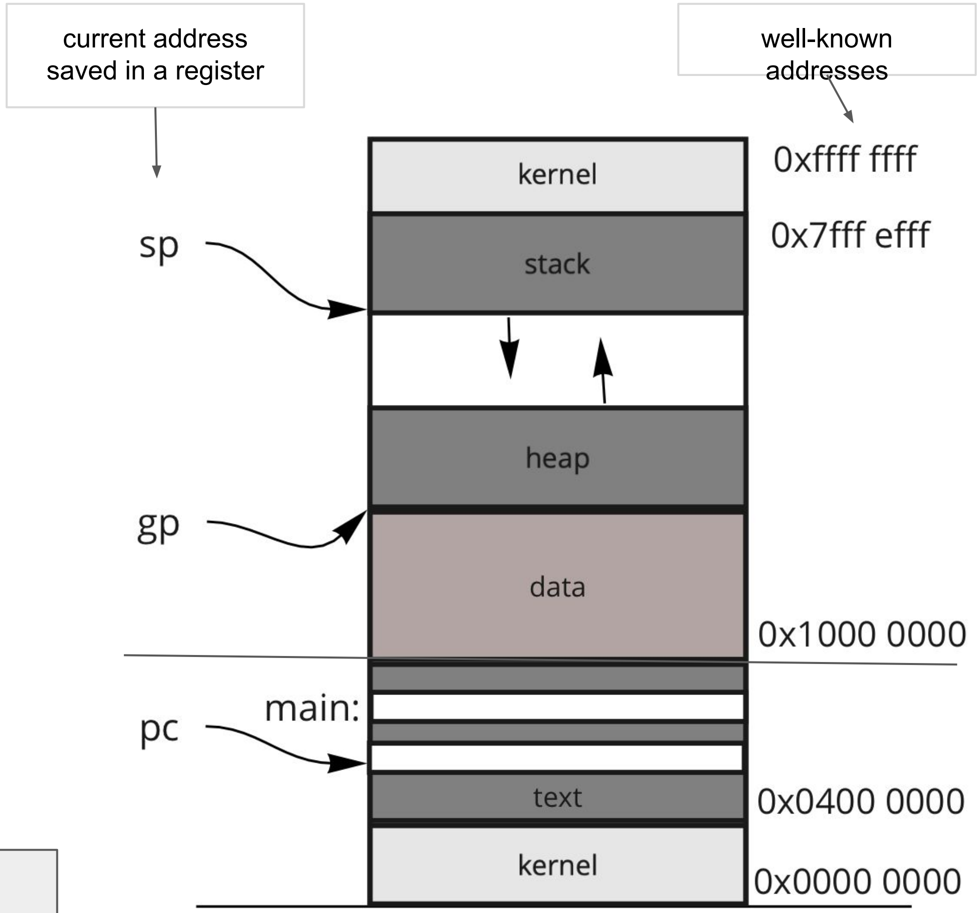
```
public ~~static~~ void A(void) {
    static int x = 5;
    C();
}
public static void B(void) {
    C();
    D();
}
public static void C(void) {
    A();
}
public static void D(void) {
    ;
}

public static void main(String args[])
   {
      A();
      B();
   }
}
```



Frames



args
locals
temps
args
locals
temps
args
locals
temps
fp:
args
locals
temps
sp:
args
locals
temps
main
x = 5;
A
C
x = 5;
A



current address saved in a register
well-known addresses
kernel
0xffff ffff
sp
stack
0x7fff efff
heap
gp
data
0x1000 0000
main:
pc
text
0x0400 0000
kernel
0x0000 0000

Three Address Code (TAC)

- A generic assembly language in which <u>all</u> instructions have <u>at most</u> three addresses

- An address references either
  - a register location
  - a memory location

- Immediate values are stored in a location within memory

Examples:
- a = y + x
- a = y
- a = x + 2
- b = d * 2 + y
  - t0 = d * 2
  - t1 = t0 + y
  - b = t1

Assumption: the assembly language is for a register-based machine, with an infinite number of registers.

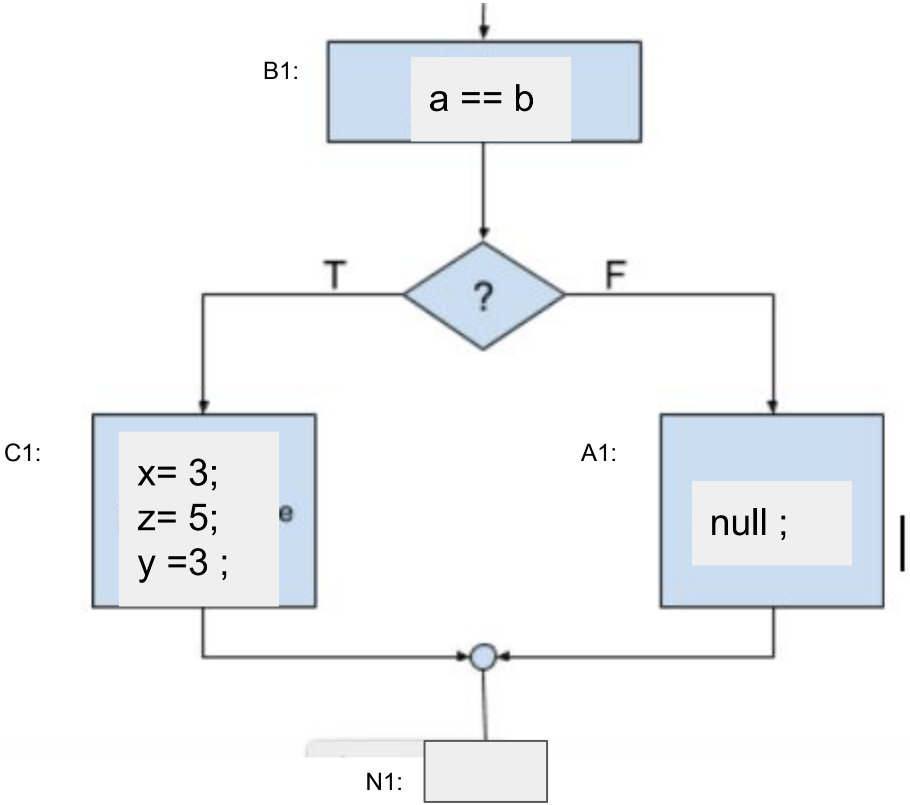
Basic Blocks

```
label:
    x = 3;
    z = 5;
    y = 3;
    goto ?
```

- A number of instructions in which there is
  - a single entry point (via a label), and
  - a single exit point (via a goto)
- All programs can be broken down into a set of basic blocks
- A control flow graph determines which a basic block is executed.
- Standard control flow graphs
  - if-the-else and all other variants (e.g., switch)
  - while, do-while and all other variants
  - for loop and all other variants
  - call-return

Code Flow: If-the-else

```
B1:  a == b

     if true goto C1   ;
     goto A1

C1:
     x = 3
     z = 5
     y = 3
    goto N1

A1:
   goto N1;

N1:
```




```
if ( a == b ) {
    x = 3;
    z = 5;
    y = 3;
} else {
   null ;
}
```

Control Flow: Loops

```
for ( i = 0; i< 10 ; i++ ) {
   a = x * 2
}
```

```
x = 5;
while ( x < 10 ) {
   a = x * 2;

    x ++;
}
```

```
x = 5;
until ( x <= 10 ) {
   a = x * 2;

    x ++;
}
```

```
x = 5;
{
    a = x * 2;

    x ++;
} while ( x < 10)
```