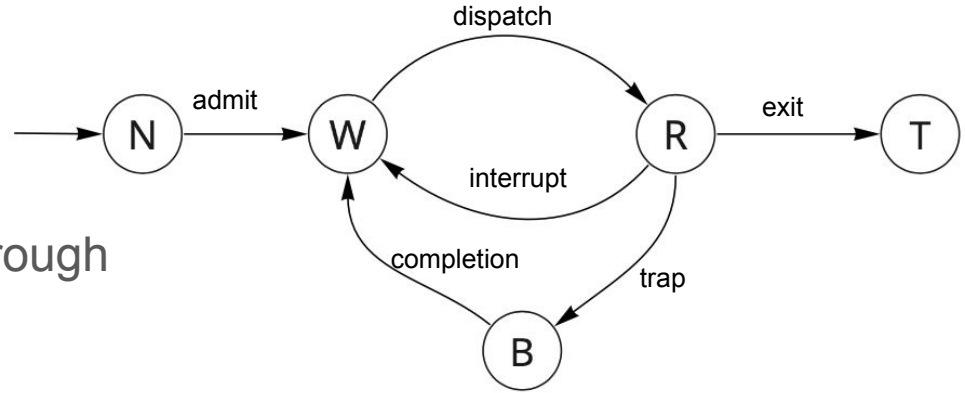
Process Status Diagram

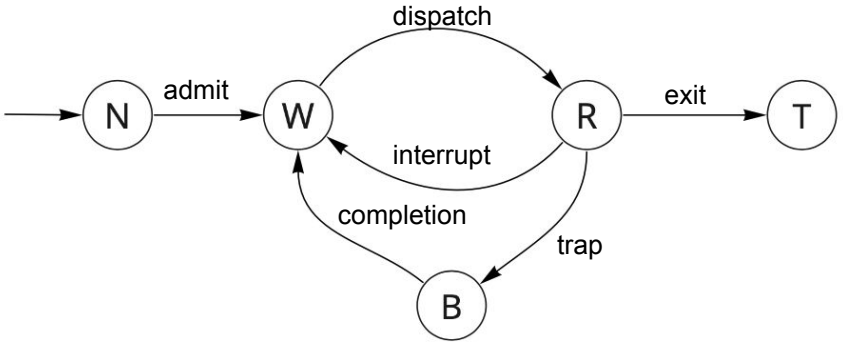


- Control of the computer moves through a well-defined cycle

- At any point in time, a single process is in control
  - loosely speaking process is equivalent to a program

- Transitions:
  - admit: A request is made to allow your program to content for control
  - dispatch: Your program is given control
  - exit: Your program asserts that it is done
  - interrupt: The OS seizes control
  - trap: Your program (implicitly or explicitly) requests a service to be performed
  - completion: The request is satisfied

Execution of your program



1. Invoke the program:
2. Wait to use the CPU
3. Execute for as long as you can -- Until
   - (Exit) You are done
   - (Interrupt) You get interrupted by some outside force
   - (Trap) You need help because you made an error or you requested it
4. If you were interrupted, goto Step 2
5. If you trap, and then goto Step 2
   - recover from the error, or
   - obtain the requested server

Driving your Car from LA to Vegas

Interrupts and Traps: (results in the kernel seizing control)

- Interrupts are asynchronous events
  - such events occur outside of your process/program
  - such events may or may not be associated with your program
  - Examples:
    - data has arrived on the NIC
    - a disk request for a different process has been completed
- Traps are synchronous events
  - such events occur inside of your process/program
  - some events are error conditions, e.g.,
    - division by zero
    - invalid or illegal memory access
  - some events are requests, e.g.,
    - read/write from a file
    - create a child process
- Exits are a specific type of trap that results in a different flow through the PSD

➢ For speed, traps are to be avoided!

Reading a block of 10 bytes!

- Java Example:

```
byte header[10];
stdin = new Scanner(System.in);
for( i = 0; i < 10 ; i++ ) {
    header[i] = stdin.nextByte();
}
```

- The Scanner class only handles primitive types
- We are not in a position to reimplement it.
- The OS knows nothing about my Java class
- Consequently, this results in 10 systems calls

- Equivalent C Example

```
byte header[10];
for( i = 0; i < 10 ; i++ ) {
    header[i] = (byte) getchar();
}
```

A More Efficient Approach

- Via 'read' system call:

```
byte header[10];

read(STDIN_FILENO, (void *) &header, 10);
```

- Read does not care what it is reading
- This results in 1 system calls
- But we need to understand pointers: * and &
- Moreover we need to cast our variables

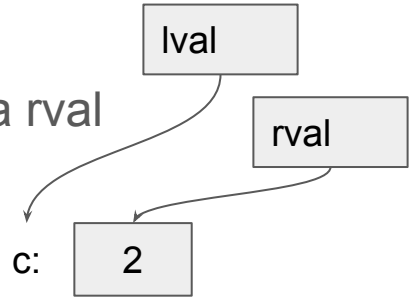
- Java Example

```
byte header[10];
stdin = new Scanner(System.in);
for( i = 0; i < 10 ; i++ ) {
    header[i] = stdin.nextByte();
}
```

Variables: Names and Values

- We all know what variables are, right!
- A variable has two values: a lval and a rval
- Consider the following assignment:

```
c = c + 1;
```



- The lval of c is the address in memory
- The rval of c is the value located at that address in memory

- A pointer is just a variable in which the rval is an address

```
int  * p  = & c;
```

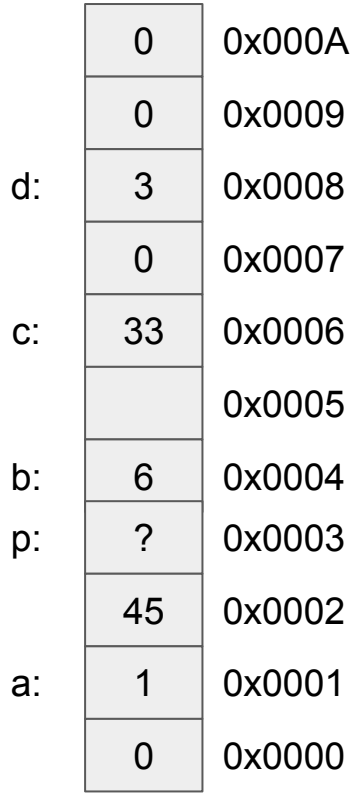
p: **&c**

- Hold that thought!

mem["c"] = 33

Memory

- We all know what an array is right!
  - Memory is just an array of integers (from 0..255):
    - mem[index] = value
- Do you know what an associative array is?
  - It's just an array that stores both the lval and rval of a variable:
    - array["name"]=value;
  - You use "name" to lookup the appropriate index
- Consider the memory to the left

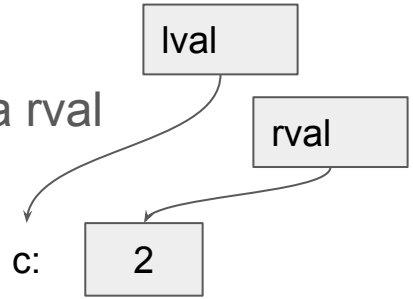


- Update the memory that I have created for this class
  - Find your name, update the associated value to be equal to your index.
  - That is to say, if your name is steven execute the following statement
    - steven = &steven;

Variables: Names and Values

- We all know what variables are, right!
- A variable has two values: a lval and a rval
- Consider the following assignment:

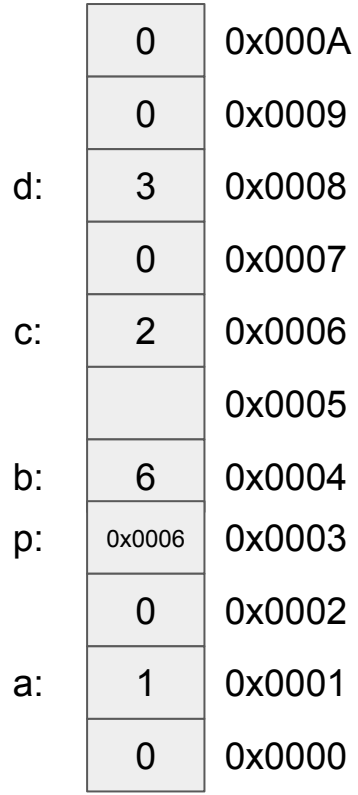
```
c = c + 1;
```



- The lval of c is the address in memory
- The rval of c is the value located at that address in memory

- A pointer is just a variable in which the rval is an address

```
int * p = & c;
```



- ~~Hold that thought!~~

- What is the address c? What is the value of p? What is the address of p?

Back to the read system call

```
int retval;
int fd;
fd = open("/home/steve/filename",
O_RDONLY);
```

- You need to allocate a buffer, a block of memory.

  ```
  byte buffer[8];
  int * p = &buffer;
  ```

- Make a read request to the OS, providing:
  - the identifier of the file to read
  - the location of the buffer
  - the number of bytes to read

  ```
  retval = read(fd, &buffer, 8);
  ```

- What are the values passed to read?
- Value of retval informs what happened.
  - retval == -1: error
  - retval == 0: end of file
  - retval <= 8: number of bytes read
- Cast the code
  - retval = read(fd, (void *) &buffer, 8);