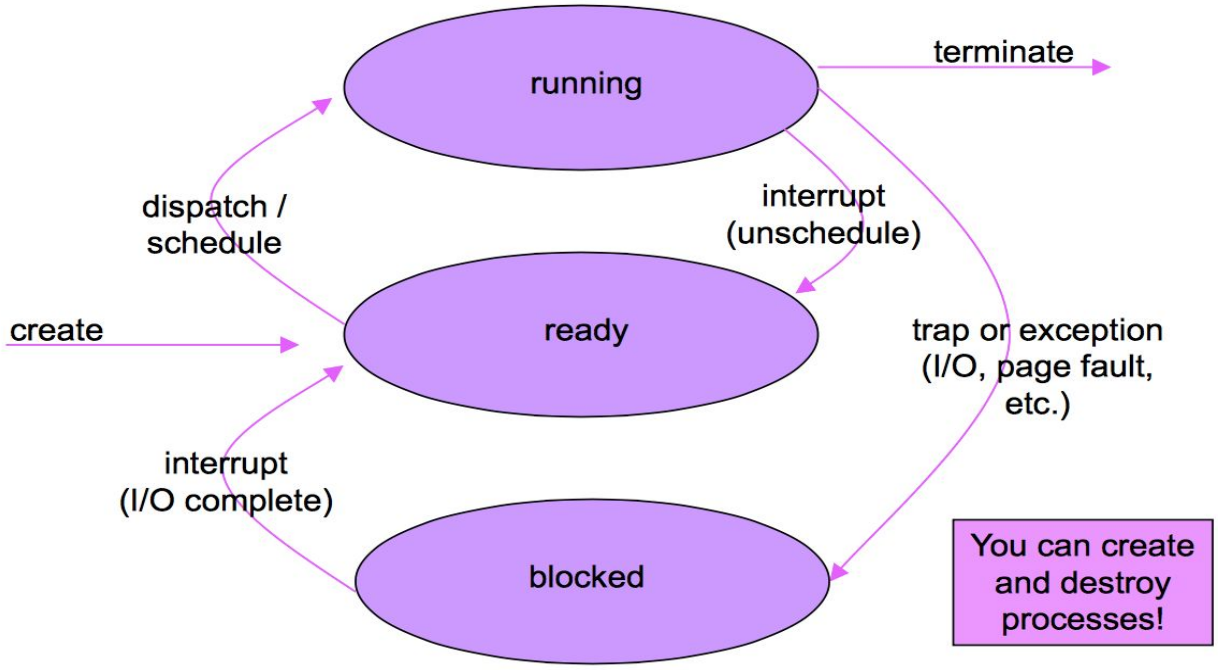
Process states and state transitions



Process:

An instance of a program being executed.

Process->Program is similar to Object->Class

Process Control Block or **PCB**:

Data structure containing metadata of a process.

Interrupt:

Hardware device requests service. Asynchronous.

Examples: User Input (keyboard), Moving Mouse, I/O Device Ready

Exception:

Program does something unexpected. Synchronous.

Examples: Divide by Zero, Page Fault, General Protection Fault

Trap:

Program requires OS assistance (System call). Synchronous.

Examples: Fork, Read, Write

JOS PCB:

```
struct Env {
      struct Trapframe env_tf; // Saved registers
      struct Env *env_link; // Next free Env
      envid_t env_id;  // Unique environment identifier
      envid_t env_parent_id; // env_id of this env's parent
      enum EnvType env_type; // Indicates special system environments
      unsigned env_status; // Status of the environment
      uint32_t env_runs; // Number of times environment has run
      pde_t *env_pgdir; // Kernel virtual address of page dir
};
```

Questions:

Is the kernel a process? Why or why not?

When an infinite loop is being executed in a user process, why doesn't the entire system freeze?

Describe what happens when a user process executes the following code. (assume the program did not modify signal handlers in any way)

```
int x = *((int *) 0);
```

What are some additions to the JOS PCB that would be worth adding? (Linux PCB has over 95 fields)

How is it ensured that a process can only access it's own memory?