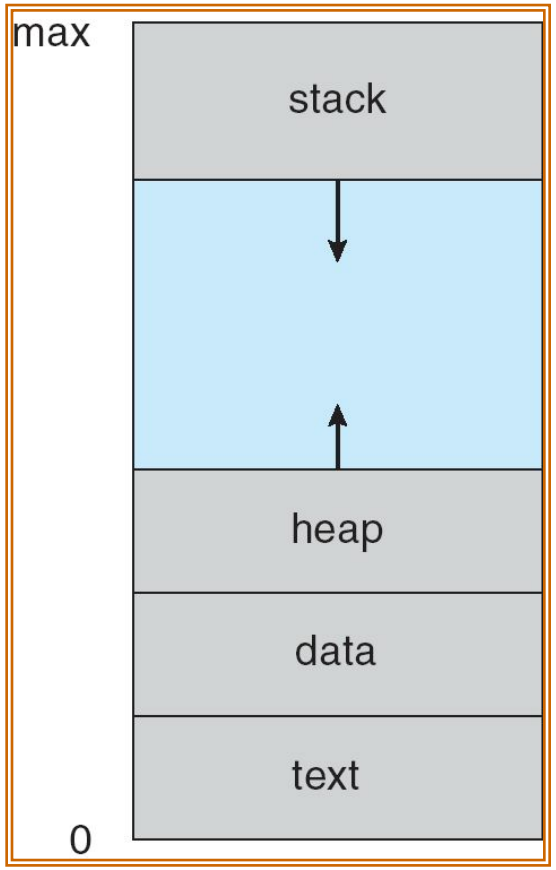
Procesos / Hilos

Procesos

- Un *proceso* es un programa en ejecución
- Se suele decir que el programa es la entidad pasiva y el proceso la activa
- Los libros de texto usan los términos **proceso** y **tarea** para referirse normalmente a lo mismo
- Un proceso es la unidad de ejecución más pequeña planificable



Ejemplo de Stack

```
int main(int argc, char *argv[]) {
    hacer_algo(3); // 3 veces
}

void hacer_algo(int tope) {
    if(tope > 0) {
        hacer_algo(tope – 1);
    }
}
```

hacer_algo-> tope = 0

hacer_algo-> tope = 1

hacer_algo-> tope = 2

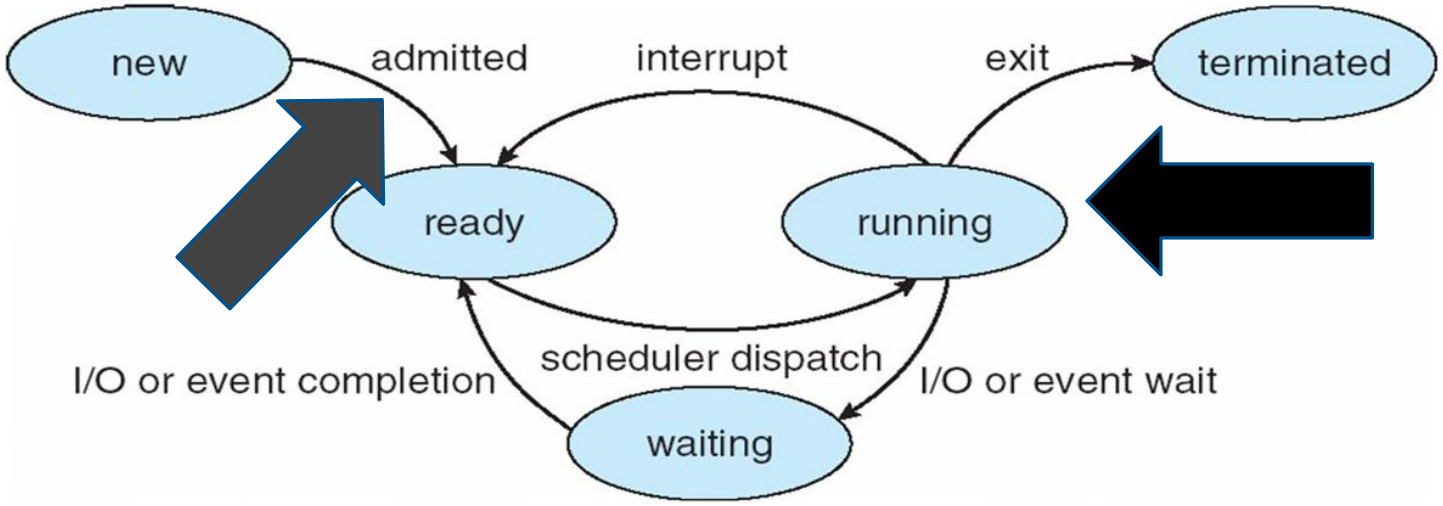
hacer_algo-> tope = 3

main -> argc = 0, argv = []

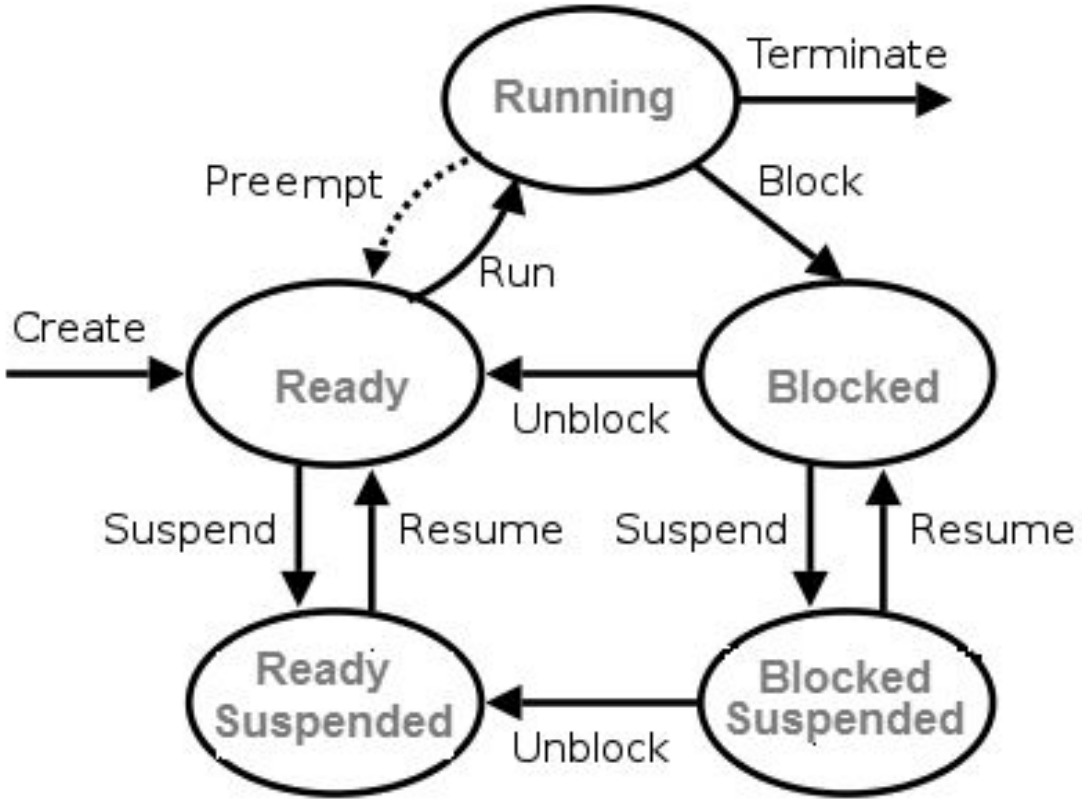
Estados

Conforme se ejecuta un proceso cambia su *estado*

- **nuevo:** El proceso se está creando
- **en ejecución:** Se están ejecutando sus instrucciones
- **en espera:** Está esperando que ocurra algún evento
- **listo:** Está esperando que le asignen la CPU
- **terminado:** Ha terminado su ejecución



Planificaciones



Tenemos 3 / 4 tipos de planificaciones

- **Extra largo plazo:** La realiza el administrador
- **Largo plazo:** Se controla el grado de multiprogramación
- **Mediano plazo:** Se controlan los procesos suspendidos
- **Corto plazo:** Se controla el grado de multiprocesamiento

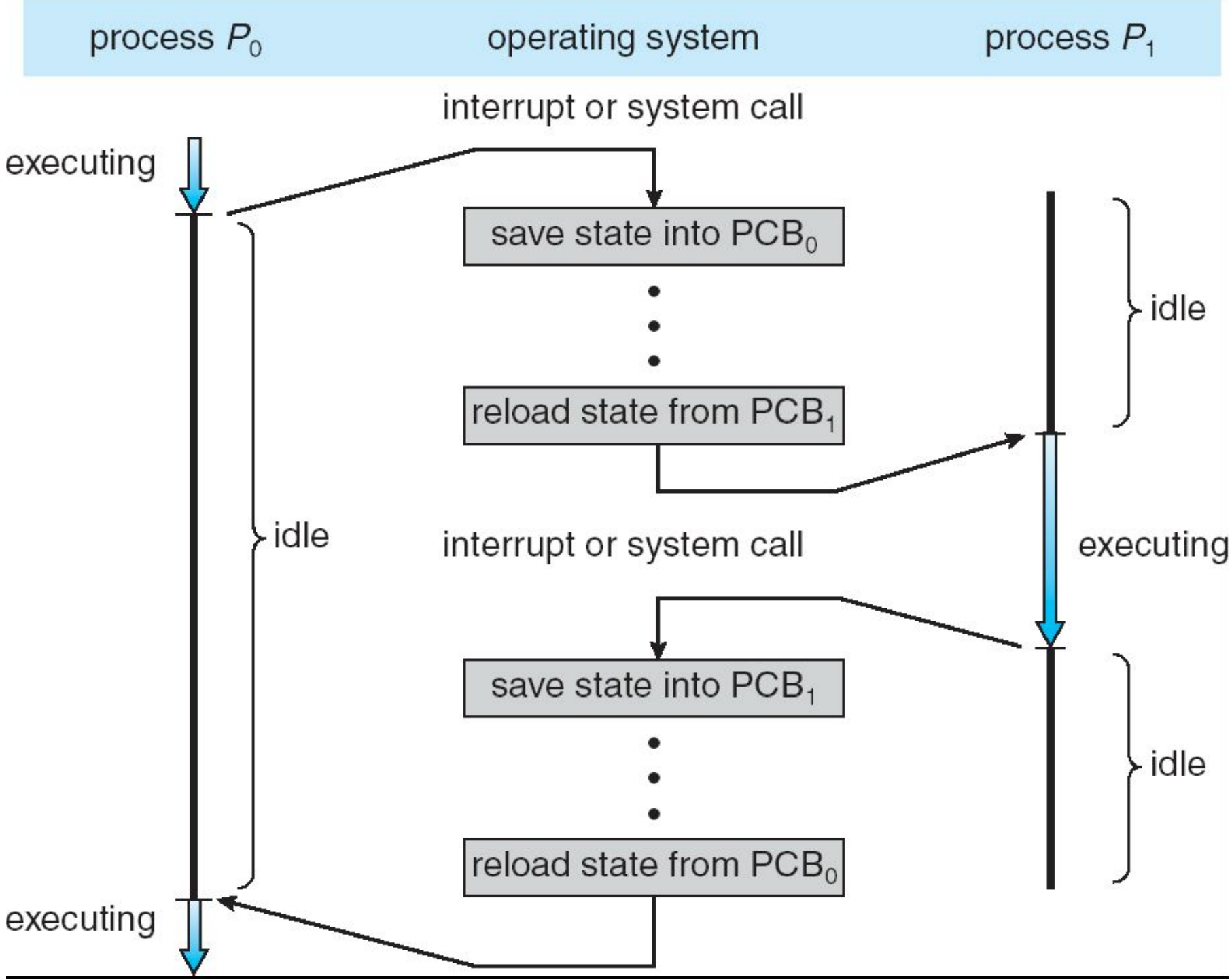
PCB (Process Control Block)

Contiene información asociada con cada proceso

- ID
- Estado del proceso
- Contador de programa
- Registros de la CPU
- Información de planificación de CPU
- Información de gestión de memoria
- Información contable
- Información de estado de E/S
- Punteros

Cambio de contexto

- Cuando se cambia el proceso que posee la CPU, el sistema debe salvar el estado del proceso anterior y cargar el estado salvado del nuevo proceso
- El tiempo que dura una conmutación de contexto es un gasto extra; el sistema no hace nada útil durante la conmutación
  - Este tiempo es Overhead
  - Se tratará de minimizar
- El tiempo requerido para la conmutación depende del soporte del procesador
- El cambio de contexto puede realizarse para cambiar de proceso, pero también para atender una interrupción, permitiendo que el proceso original continúe


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing

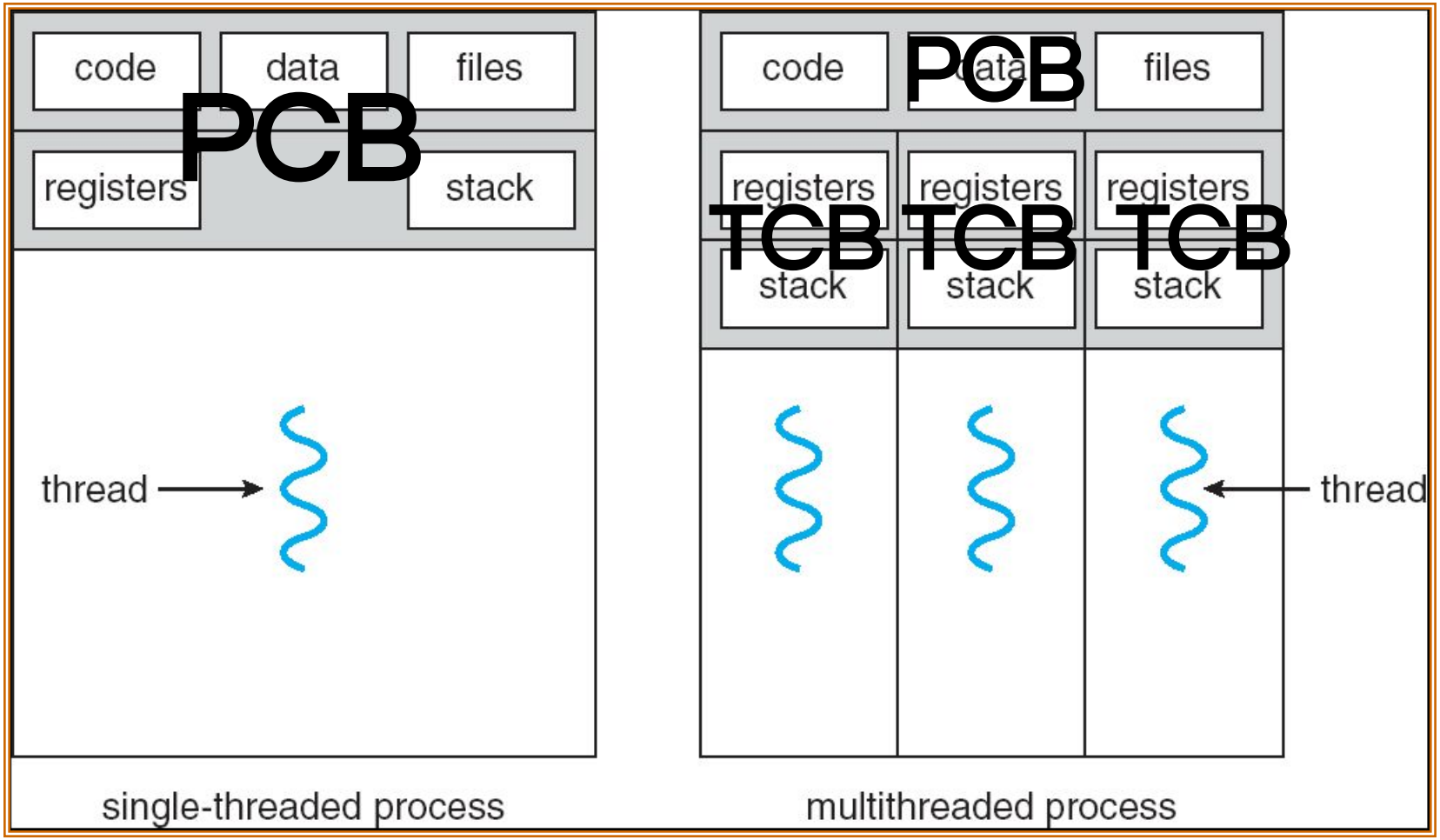
Cambios de contexto

- 1 Process switch  2 Context switch (se guarda el ctx de un proceso y se restaura el de otro)
- 2 Context switch  1 Process switch (Se puede elegir al mismo proceso, ejecutar una syscall, atender una interrupción)
- 1 Mode switch  1 Context switch (paso de ejecutar un proceso de usuario a ejecutar el SO o viceversa)
- 1 Context switch  1 Mode switch (puede ocurrir una interrupción cuando ya estoy atendiendo una)

Hilos

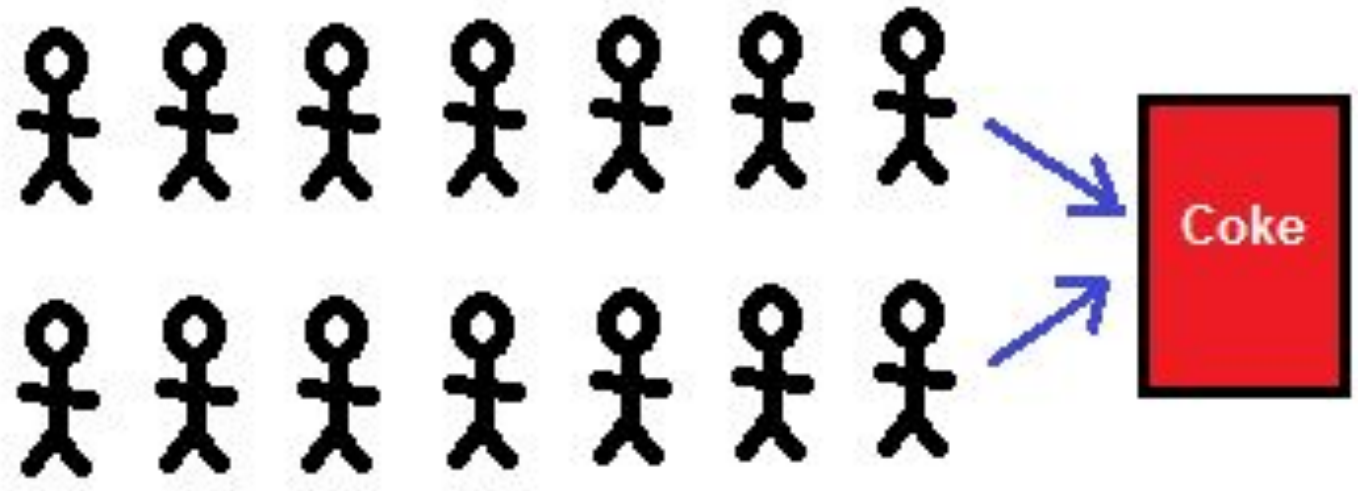
- Un **hilo** es una unidad básica de utilización de la CPU y consiste en un juego de registros y un espacio de pila. Es también conocido como **proceso ligero.**
- Comparte el **código, los datos y los recursos** con sus hilos pares.
- Un **proceso** está formado por **uno o más** hilos de ejecución.
- Permiten paralelismo dentro de un proceso o aplicación.
- Al compartir recursos, pueden comunicarse sin usar ningún mecanismo de comunicación inter-proceso del SO.
- No hay protección entre hilos. Un hilo puede escribir en la pila de otro hilo del mismo proceso.

Hilos

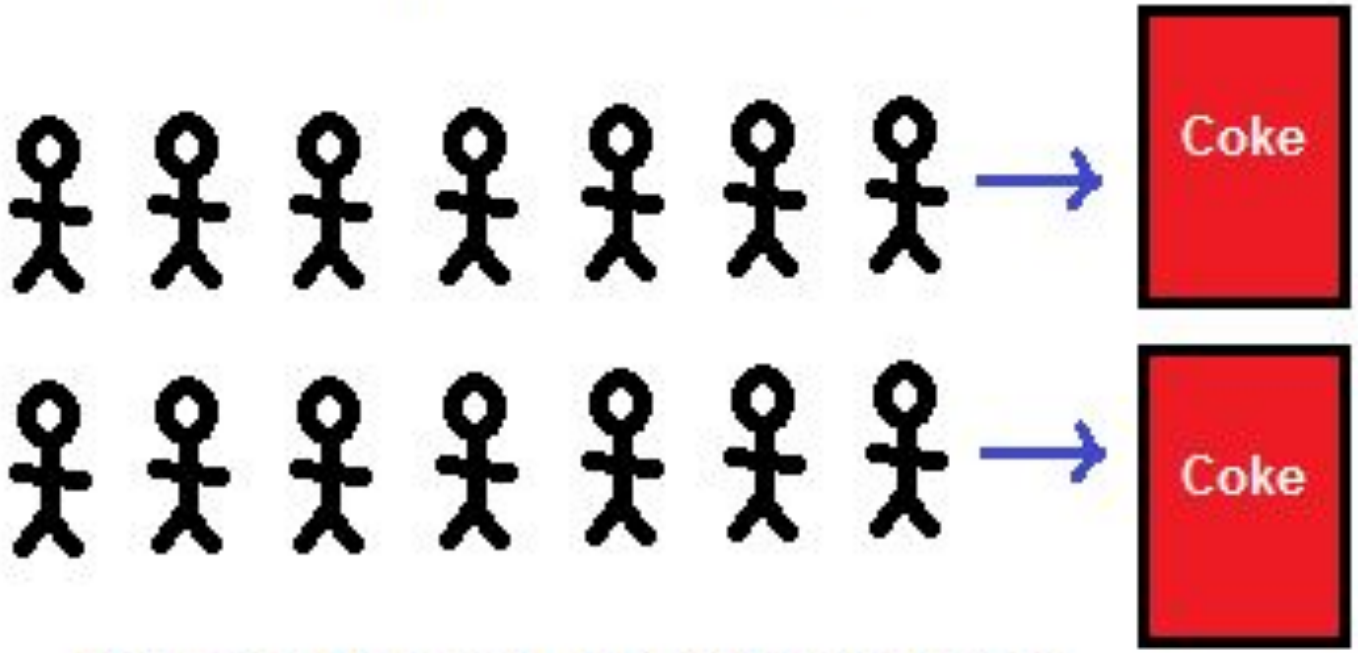

code
data
files
PCB
registers
stack
thread
single-threaded process
code
PCB
files
registers
registers
registers
TCB
TCB
TCB
stack
stack
stack
thread
multithreaded process

Diferencias con procesos

- Permiten la **comunicación** privada entre varios hilos del mismo proceso, **sin intervención del S.O**.
- **Mayor eficiencia en el cambio** de un Hilo a otro, que de un Proceso a otro.
- **Mayor eficiencia en la creación** de un Hilo que en la creación de un Proceso Hijo.
- Un Proceso Multihilo puede recuperarse de la “muerte” de un Hilo, pues conoce los efectos de esta, y toma su espacio de memoria (excepto para el main).
- Cuando un Proceso “muere” todos sus Hilos también, pues los recursos de Proceso son tomados por el Sistema Operativo.



Concurrent: 2 queues, 1 vending machine

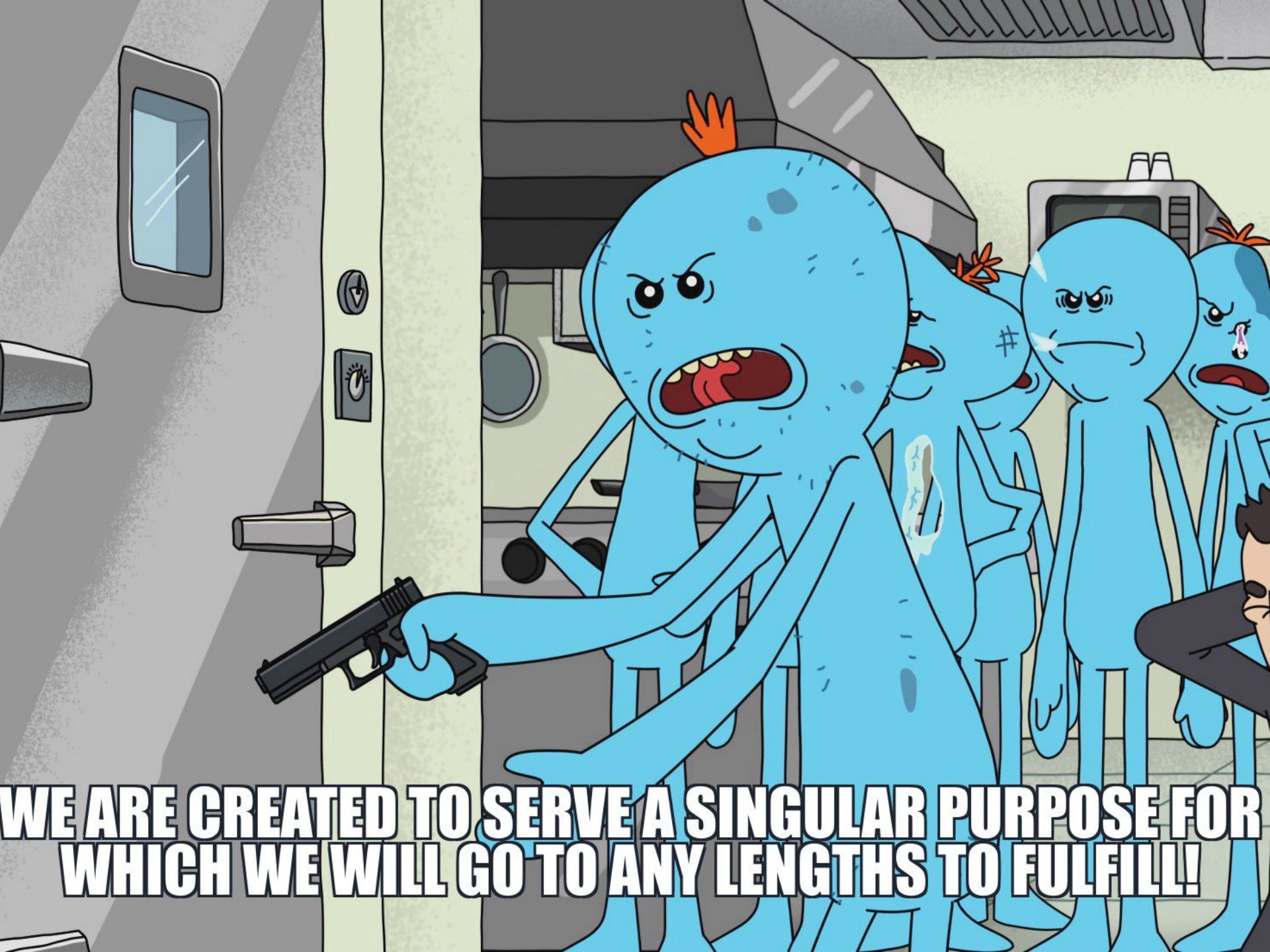


Parallel: 2 queues, 2 vending machines

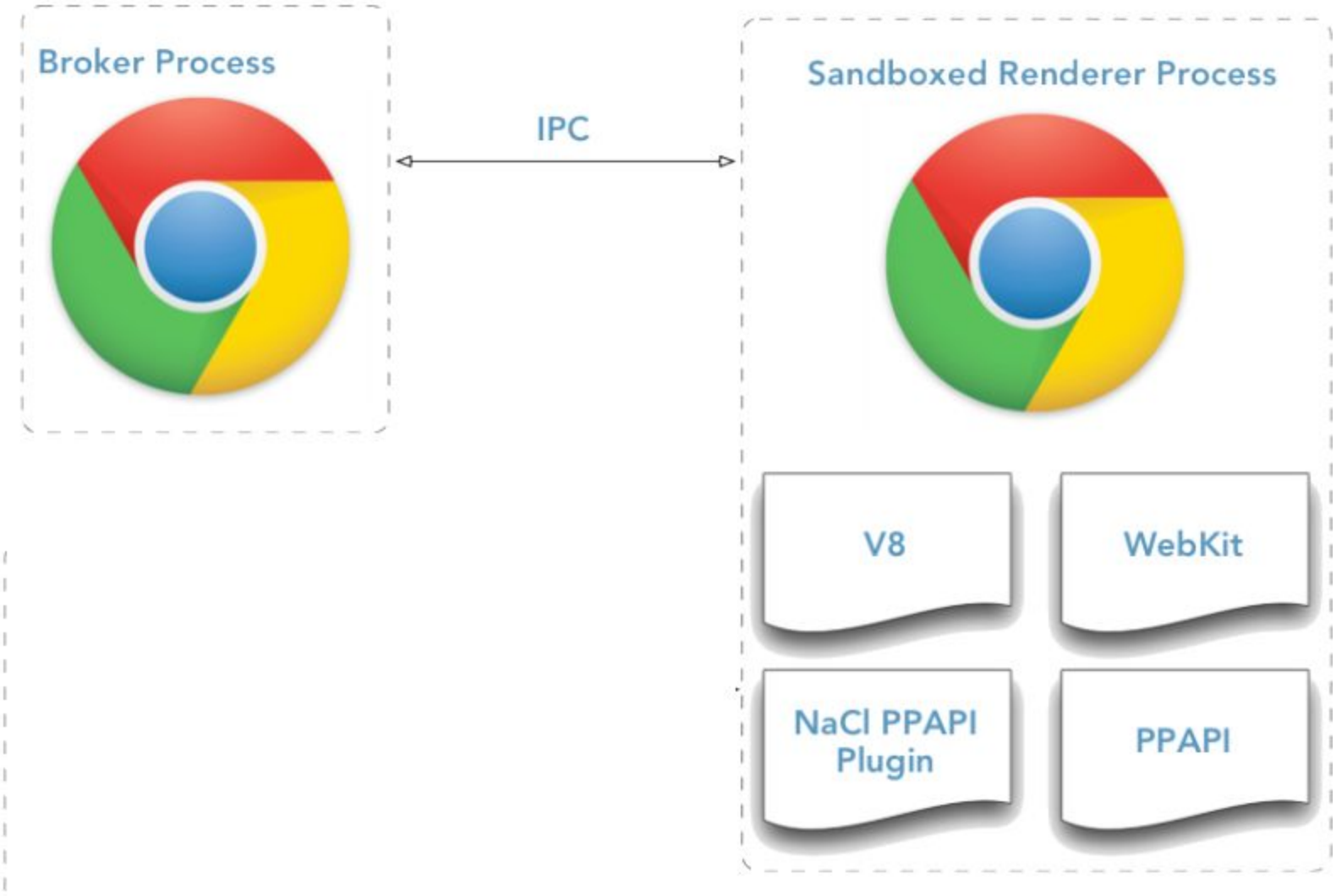
KLT - ULT

- Los KLTs son llamados “Hilos a nivel de Kernel” o "nativos".
- El SO conoce de su existencia y controla su ejecución

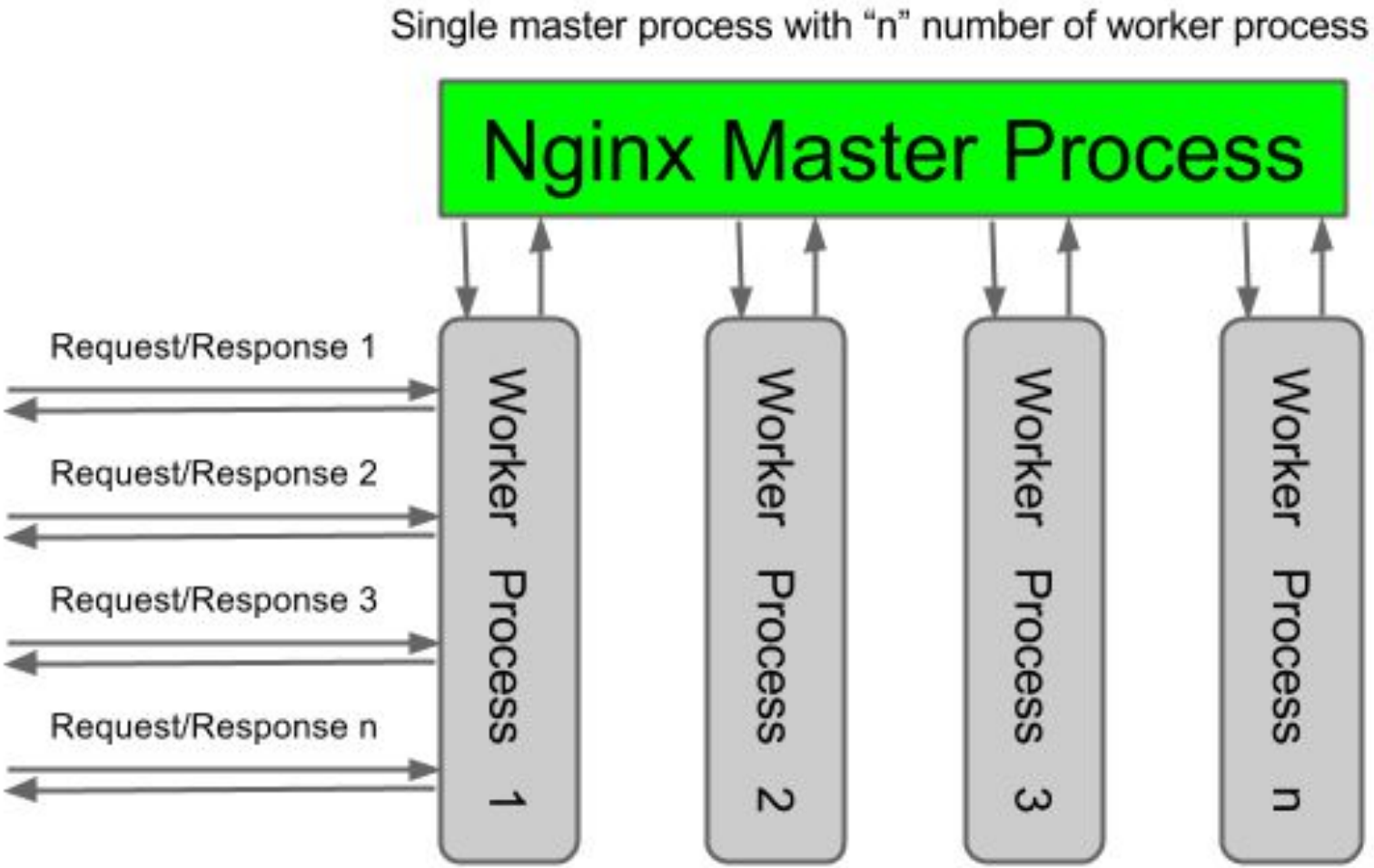
- Los ULTs son llamados “Hilos a nivel de usuario” o "Green Threads".
- Su gestión es realizada por bibliotecas en modo usuario, y por lo tanto, el SO no sabe de la existencia de estos hilos.
- Características:
  - Realizan la conmutación de contexto aún más rápidamente.
  - Cuando uno de ellos realiza una operación bloqueante (ej. E/S), sus hilos pares no pueden continuar.
  - No permiten paralelismo entre hilos pares.


WE ARE CREATED TO SERVE A SINGULAR PURPOSE FOR
WHICH WE WILL GO TO ANY LENGTHS TO FULFILL!

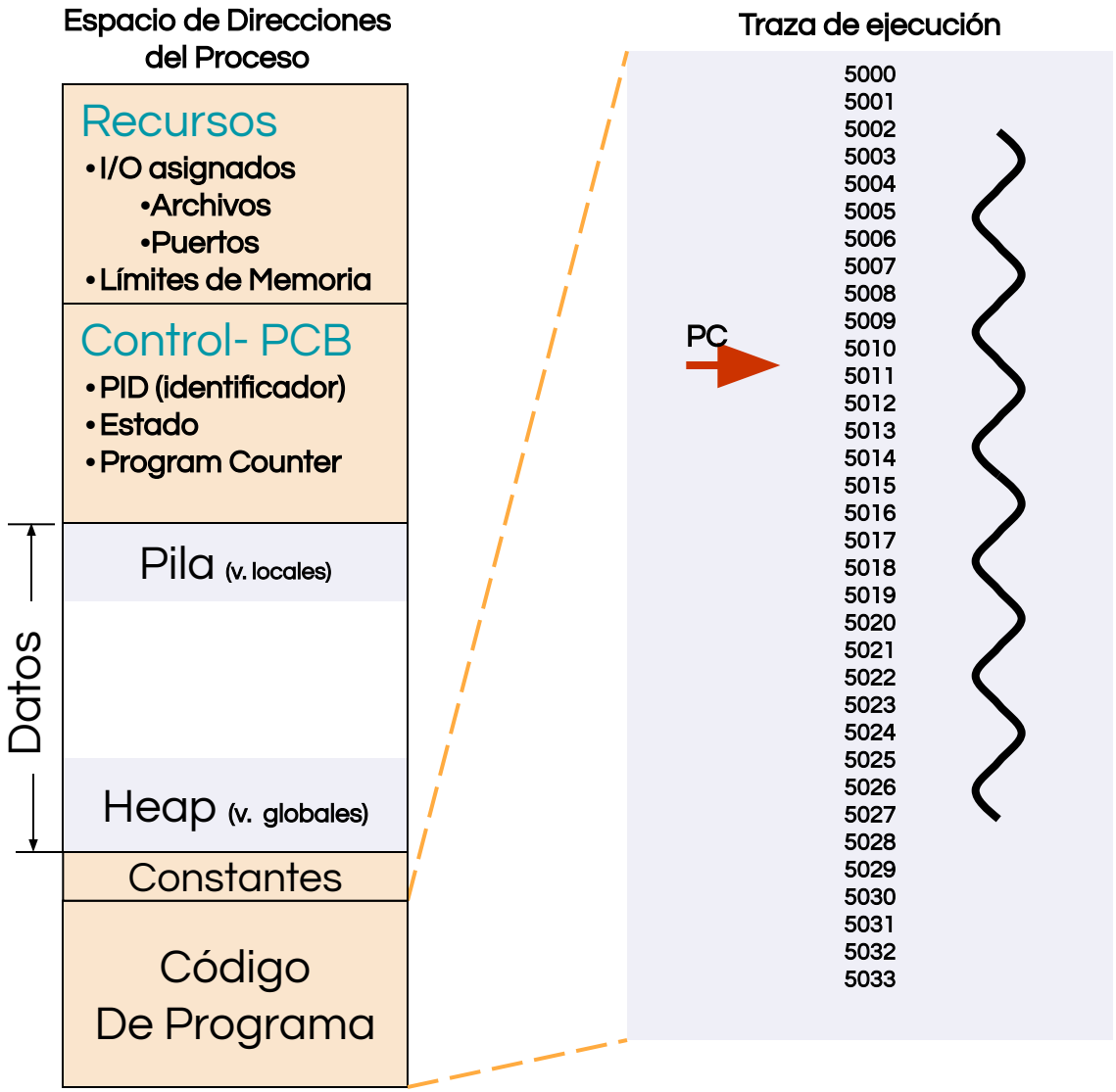
Ejemplos


Broker Process
IPC
Sandboxed Renderer Process
V8
WebKit
NaCl PPAPI Plugin
PPAPI

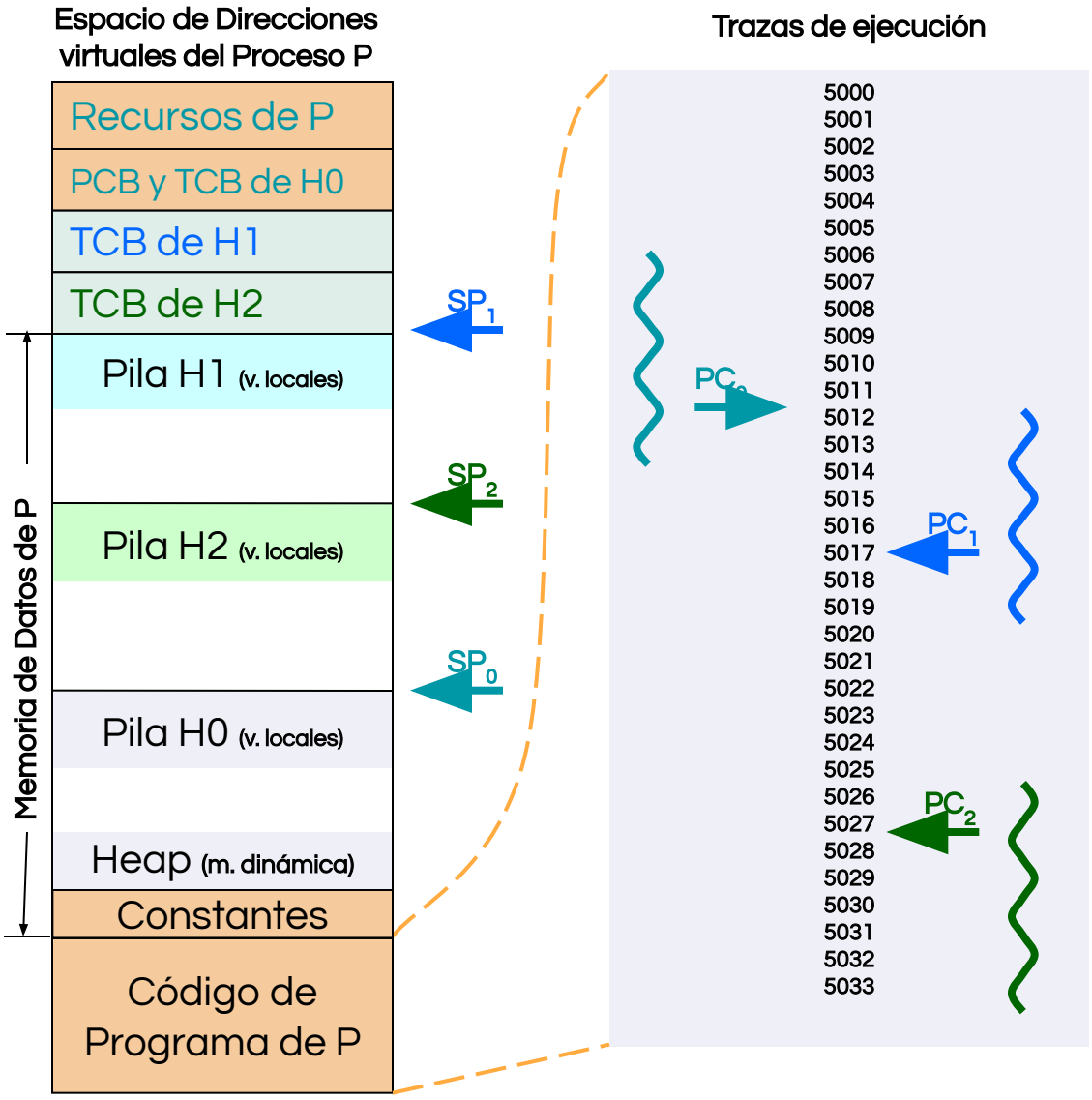
Ejemplos


Single master process with “n” number of worker process
Nginx Master Process
Request/Response 1
Request/Response 2
Request/Response 3
Request/Response n
Worker Process 1
Worker Process 2
Worker Process 3
Worker Process n

Ejecución típica



Ejecución con hilos



- H1 y H2 son Hilos creados por el Proceso P.
- El Proceso P define el espacio de direcciones.
- El Proceso P es dueño de los recursos que pueden usar y compartir H1 y H2.
- Los Hilos H1 y H2 comparten el espacio de direcciones.
- El Hilo H0 es el Hilo "por defecto" de P, creado para la función **`main()`** del Programa.

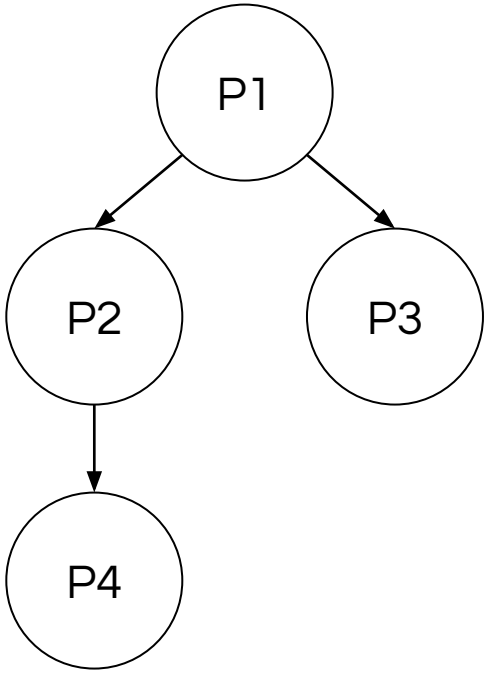
Dinámica de estados

El estado del Proceso **P** es la combinación de los estados de sus Hilos.

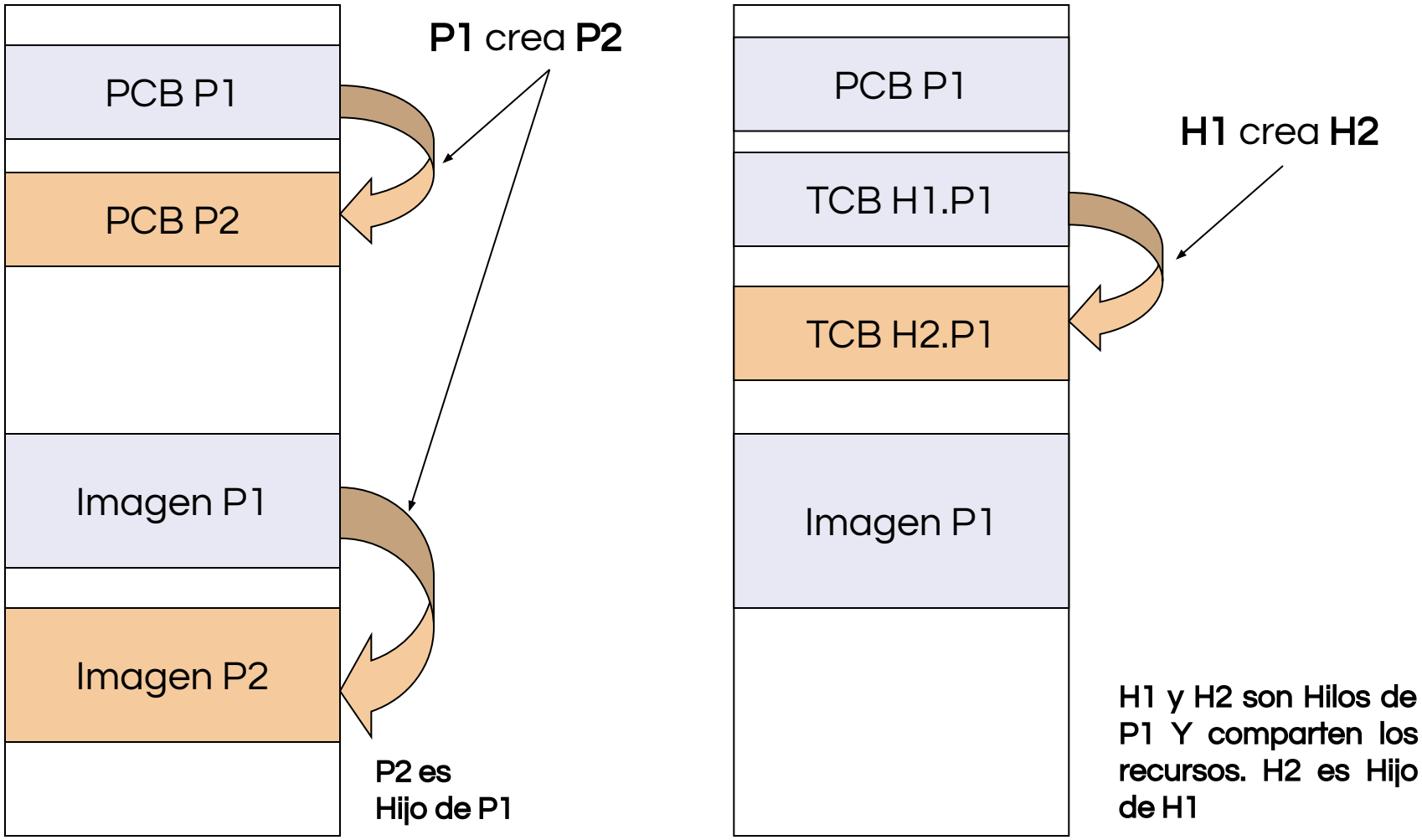
- Cuando cualquiera de los Hilos está en estado "**Ejecutando**", el estado de **P** será "**Ejecutando**".
- Si ningún Hilo está en "**Ejecutando**", si alguno está en "**Listo**", el estado de **P** será "**Listo**".
- El estado de **P** es "Bloqueado" sólo si todos sus Hilos están en estado "**Bloqueado**".

Procesos Hijos

- Un Proceso puede crear otro Proceso, iniciando una tarea mientras termina otra.
- En la figura, P2 y P3 son Hijos de P1, y P4 es hijo de P2.
- Jerarquía entre Procesos:
  - Padre e Hijo se pueden ejecutar concurrentemente (quizás en paralelo).
  - El Padre hace su trabajo y debe esperar a que el Hijo termine para terminar.
- Padre e Hijo pueden compartir todos los recursos, una parte de ellos, o ninguno.
- La estructura de Memoria de un Proceso Hijo es un duplicado de la estructura del Padre (espacio de direcciones).



Creación de hijos vs hilos



Creación de procesos

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
    int pid;
    pid = fork();
    if (pid < 0) {
        fprintf(stderr, "Falló");
        exit(-1);
    } else if (pid == 0) {
        /* Inicio de código de
   Proceso Hijo */
        execlp("/bin/ls","ls",NULL
   );
        …
        /* Fin de código de
   Proceso Hijo */
    }  else {
        /* Inicio de código de
   Proceso Padre */
        …
        /* Esperar término del
   Hijo */
        wait(NULL);
        printf("Child
   Complete");
        exit(0);
        …
        /* Fin de código de
   Proceso Padre */
     }
}
```

Si no nos creen, miren Redis: https://github.com/antirez/redis/blob/5.0.8/src/rdb.c#L1337

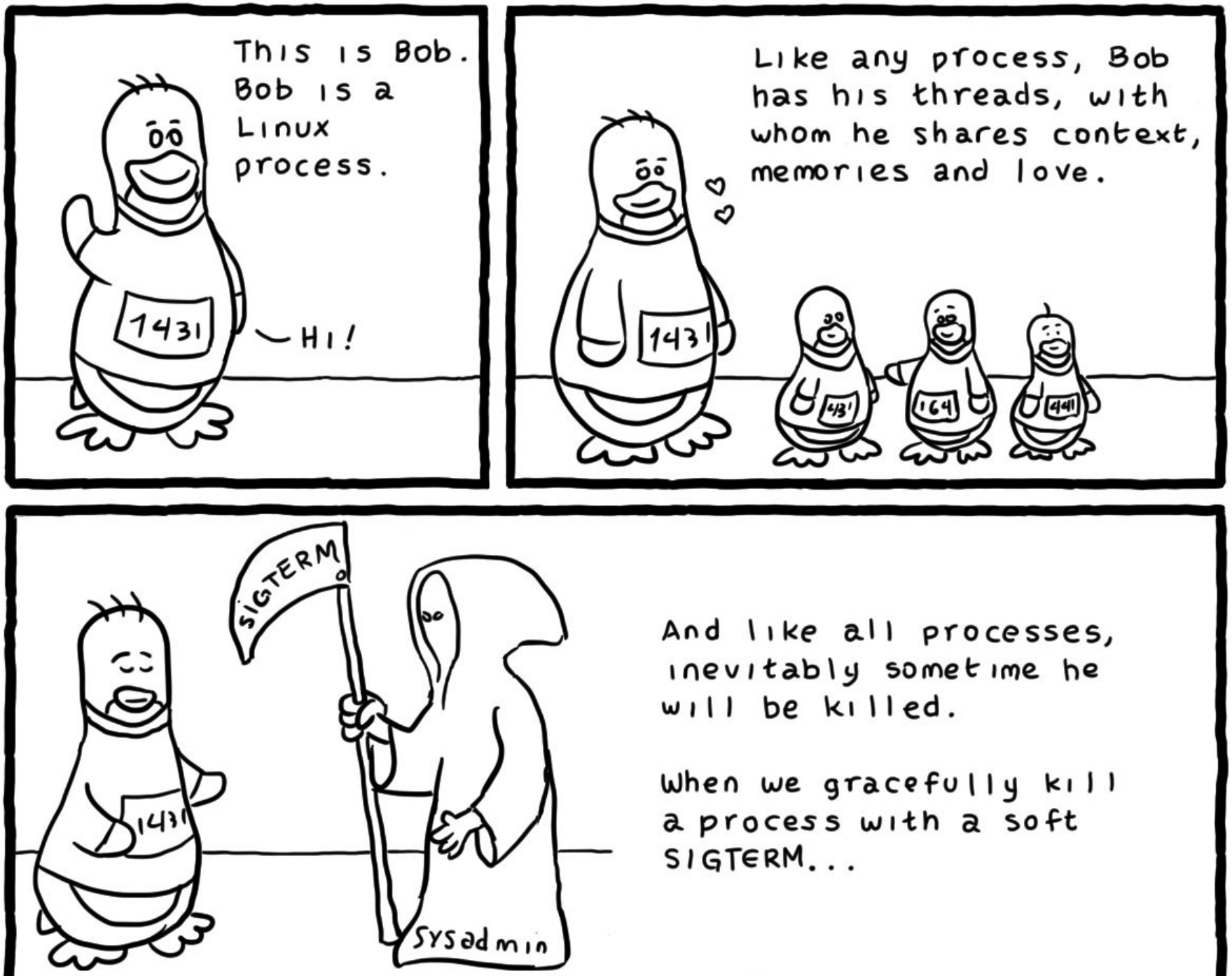
Creación de hilos


```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function( void *ptr );

main() {
     pthread_t hilo1, hilo2;
     char *message1 = "Hilo 1";
     char *message2 = "HIlo 2";
    int iret1, iret2;

     iret1 = pthread_create( &hilo1, NULL, print_message_function, (void*)
    message1);
     iret2 = pthread_create( &hilo2, NULL, print_message_function, (void*)
    message2);
     pthread_join( hilo1, NULL);
     pthread_join( hilo2, NULL);

     printf("Hilo 1 devuelve: %d\n",iret1);
     printf("Hilo 2 devuelve: %d\n",iret2);
     exit(0);
}
```

```
void *print_message_function(void *ptr) {
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}
```

Terminación de procesos

- La última operación de un proceso es una llamada al SO indicando que lo elimine (**exit**)
  - Se envía al padre información de salida (via **wait**)
  - Los recursos usados por el proceso son liberados
- Un proceso padre puede terminar la ejecución de sus hijos (**abort**)
  - El hijo se ha excedido en el uso de recursos asignados
  - La tarea que realiza el hijo no es ya necesaria
  - El padre va a terminar
    - Algunos SOs no permiten que un hijo siga si su padre termina. Todos los hijos son terminados – *terminación en cascada*
    - Otros transfieren el hijo al padre (del padre)


This is Bob. Bob is a Linux process.
1431
Hi!
Like any process, Bob has his threads, with whom he shares context, memories and love.
1431
431
164
4411
SIGTERM
1431
Sysadmin
And like all processes, inevitably sometime he will be killed.
When we gracefully kill a process with a soft SIGTERM...

...we give him the chance to talk with his kids about it. So, the kids finish their tasks...

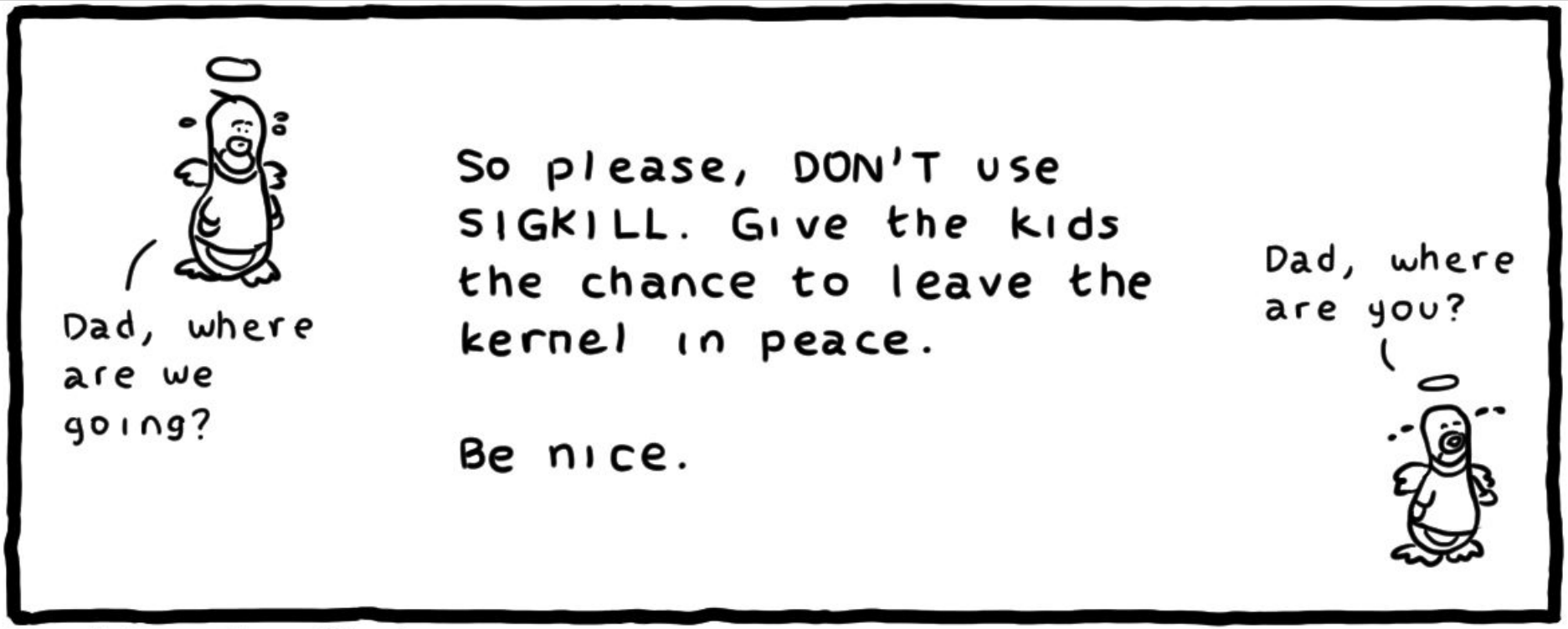
...and say goodbye to each other.

That's a process life!

On the other hand, when we brutally kill a process with a SIGKILL, we prevent them from finishing their job and say goodbye...

...and this is so SAD!

DAD!!


Dad, where are we going?
So please, DON'T use SIGKILL. Give the kids the chance to leave the kernel in peace.
Be nice.
Dad, where are you?
Daniel Stori {turnoff.us}

¿Preguntas?