ICAPlants [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], from Wikimedia Commons

Process Description and Control

Week 3

(part one)

COMP 4735
Rafael Roman Otero

What’s an OS?
(small detour)

What’s an OS?

“**a program** that controls the execution of application programs, and acts interface between applications and the computer hardware”

--- Stallings

What’s an OS?

“**a program** that controls the execution of application programs, and acts interface between applications and the computer hardware”

--- Stallings

“It is hard to pin down what an operating system is other than saying **it is the software that runs in kernel mode**—and even that is not always true.

--- Tanenbaum

What’s an OS?

The OS as an extended machine (Tanenbaum)

- The architecture of most computers at the machine-language level is **primitive** and **awkward to program**, especially for I/O.

What’s an OS?

The OS as an extended machine (Tanenbaum)

- Say you want to do bare-metal **USB programming.**

RC-508

Documentation needed:

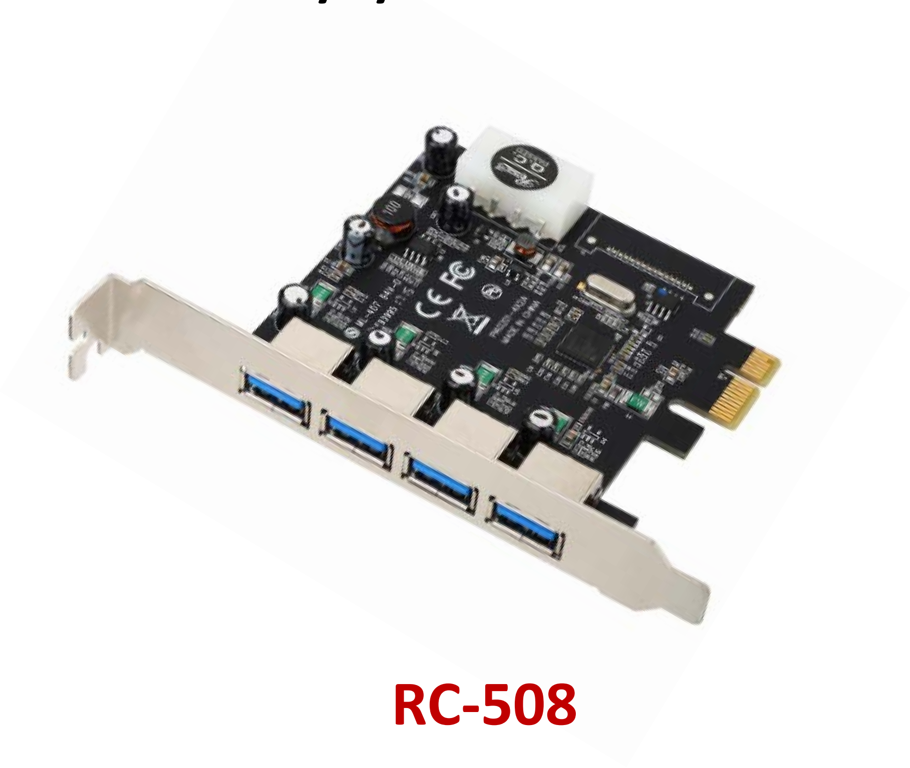
- USB protocol Book (300 pages) so you can make yourself a semi-expert on USB protocol.
- RC-508 Datasheet (a few hundred pages as well)
- The manual of the chip that is mounted on the RC-508 (the one doing low-level USB stuff)
- A book on PCI protocol (300 pages), or else how will you write the PCI drivers that you need to talk to the device?
- Your computer’s chipset manual (100 pages)
- X86 developers manual (2k pages)

(+ 10 months of trying to get the thing to do something)

What's an OS?

The OS as an extended machine (Tanenbaum)

- Say you want to do bare-metal **USB programming.**



RC-508

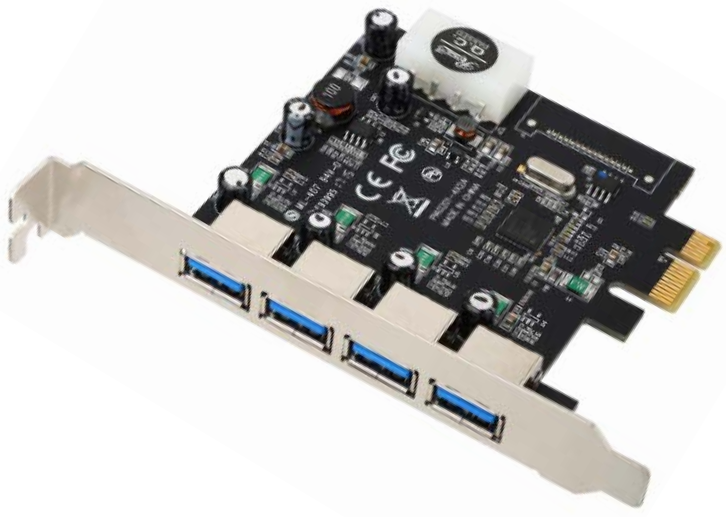
It's ridiculous how complex low-level I/O programming is.

You can't possibly expect a computer to be useful at all when you have to do everything yourself

What's an OS?

The OS as an extended machine (Tanenbaum)

- Say you want to do bare-metal **USB programming.**



RC-508

Instead an OS provides you and the whole world with:

a) **PCI Drivers**
b) **USB Drivers**
c) Additional **abstractions** like "files", "drives", "plug and play" so you can print the complain letter you wrote to the Queen and continue with your life.

What's an OS?

The OS as an extended machine (Tanenbaum)

- **In general**, CPUs, memories, disks, and I/O devices are very **complex** and present **difficult**, **awkward**, **idiosyncratic**, and **inconsistent interfaces**.

What’s an OS?

The OS as an extended machine (Tanenbaum)

- **In general**, CPUs, memories, disks, and I/O devices are very **complex** and present **difficult**, **awkward**, **idiosyncratic**, and **inconsistent interfaces**.

It’s the OS task to hide the hardware and present programs (and their programmers) with **nice**, **clean**, **elegant**, **consistent**, **abstractions to work with instead**.

What's an OS?

The OS as an extended machine (Tanenbaum)

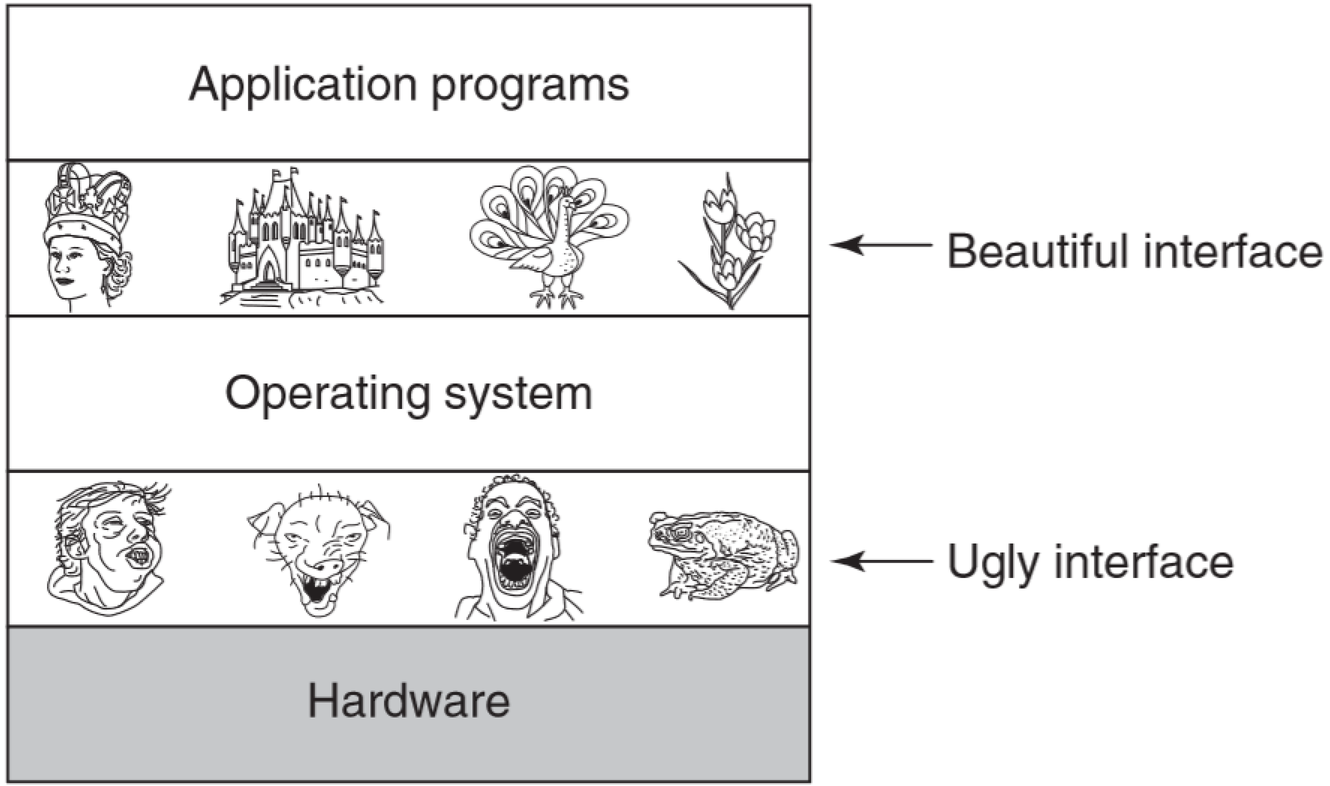


Figure 1-2. Operating systems turn ugly hardware into beautiful abstractions.

What’s an OS?

The OS as a resource manager (Tanenbaum)

- An alternative, bottom-up, view holds that the operating system is there to **manage all the pieces of a complex system**.

What’s an OS?

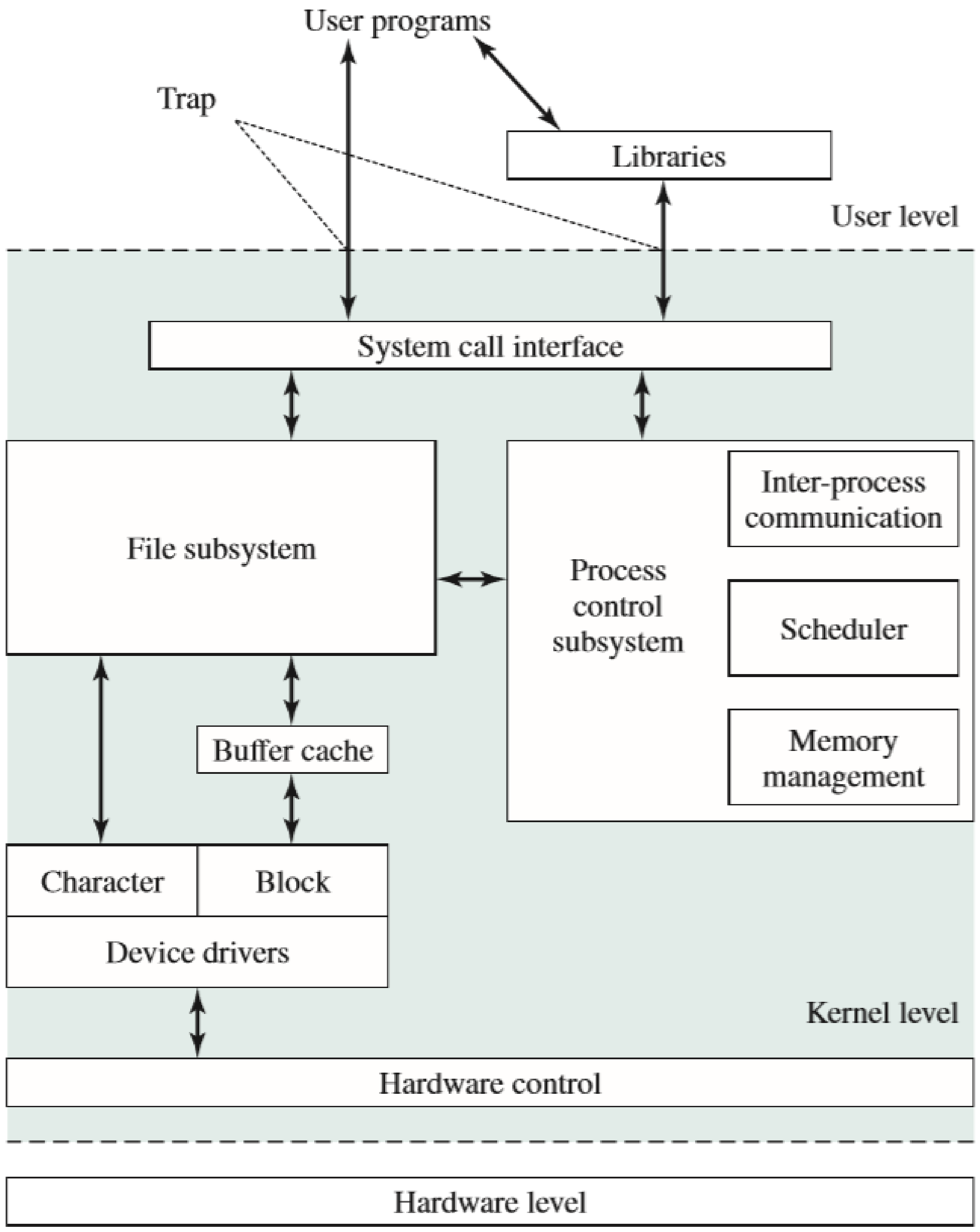
The OS as a resource manager (Tanenbaum)

- An alternative, bottom-up, view holds that the operating system is there to **manage all the pieces of a complex system**.
- In this view, the OS’s job is to provide for an **orderly and controlled allocation** of the processors, memories, and I/O devices among processes.

What’s an OS?

Traditional *nix architecture

Stalling’s



What’s an OS?

Resources on the Web

- The mind behind Linux | Linus Torvalds
- Old School Sean - The MINIX operating system

What’s an OS?

Book reference

Section 2.1, 2.8

Process

Process

What is a process?

Process

“A program in execution”

“An instance of a program running on a computer”

--- Stallings

“A process is basically a program in execution”

--- Tanenbaum

Process

Definition revolves around **the task**, not the program.

"The agent that carries out the <u>task described in a program</u> is called a process."

--- Haldar & Aravind

"An executable that was brought into 'existence' by the OS"

--- My definition

Process

Each process is associated with an **address space.**

What's an address space?

Process

Each process is associated with an **address space.** This address space contains:

- **The process' __________ ?**
- **The process' __________ ?**
- **The process' __________ ?**

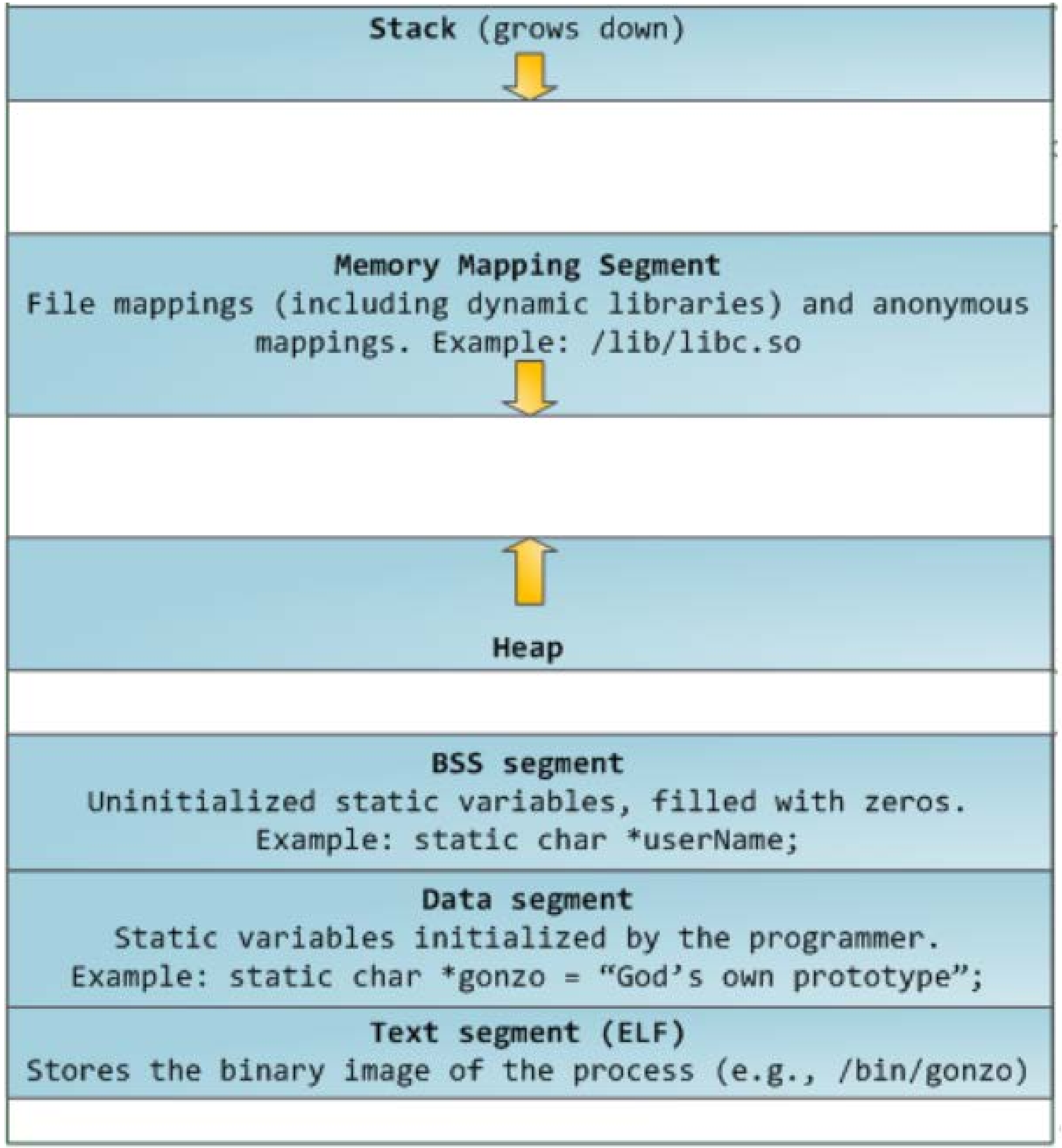
Process

Each process is associated with an **address space.** This address space contains:

- **The process' __________ ?**
- **The process' __________ ?**
- **The process' __________ ?**

Process

- **A process address' space (in more detail)**



Up to some number (e.g. 0xFFFFFFFF)

From some number (e.g. 0x0000000)

https://manybutfinite.com/post/anatomy-of-a-program-in-memory/

ubuntu@ip-172-31-13-223: ~

```
ubuntu@ip-172-31-13-223:~$ htop
```

htop is a system monitor for Linux

```
CPU[                                        0.0%]   Tasks: 29, 24 thr; 1 running
Mem[|||||||||||||||||||||||||         89.9M/984M]   Load average: 0.00 0.00 0.00
Swp[                                       0K/0K]   Uptime: 01:03:13

 PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
1626 ubuntu      20   0 32324  4320  3616 R  0.7  0.4  0:00.02 htop
   1 root        20   0  156M  9088  6796 S  0.0  0.9  0:02.33 /sbin/init
 401 root        19  -1 94844 12256 11568 S  0.0  1.2  0:00.57 /lib/systemd/systemd-journald
 415 root        20   0 97708  1948  1772 S  0.0  0.2  0:00.00 /sbin/lvmetad -f
 427 root        20   0 43560  4644  3184 S  0.0  0.5  0:00.26 /lib/systemd/systemd-udevd
 483 systemd-t   20   0  138M  3236  2720 S  0.0  0.3  0:00.00 /lib/systemd/systemd-timesyncd
 454 systemd-t   20   0  138M  3236  2720 S  0.0  0.3  0:00.01 /lib/systemd/systemd-timesyncd
 570 systemd-n   20   0 80016  5340  4748 S  0.0  0.5  0:00.01 /lib/systemd/systemd-networkd
 588 systemd-r   20   0 70612  5444  4892 S  0.0  0.5  0:00.02 /lib/systemd/systemd-resolved
 711 root        20   0 25376   292    60 S  0.0  0.0  0:00.06 /sbin/iscsid
 712 root        10 -10 25880  5272  4044 S  0.0  0.5  0:00.00 /sbin/iscsid
 727 daemon      20   0 28332  2544  2332 S  0.0  0.3  0:00.00 /usr/sbin/atd -f
 735 root        20   0 31748  3232  2944 S  0.0  0.3  0:00.00 /usr/sbin/cron -f
```

X stands fo executable, so that’s probably
The text segment

```
ubuntu@ip-172-31-13-223:~$ sudo cat /proc/735/maps
563396682000-56339668d000 r-xp 00000000 ca:01 13048                      /usr/sbin/cron
56339688c000-56339688d000 r--p 0000a000 ca:01 13048                      /usr/sbin/cron
56339688d000-56339688e000 rw-p 0000b000 ca:01 13048                      /usr/sbin/cron
563398153000-563398174000 rw-p 00000000 00:00 0                          [heap]
7f1416525000-7f1416530000 r-xp 00000000 ca:01 2089                       /lib/x86_64-linux-gnu/libnss_files-2
7f1416530000-7f141672f000 ---p 0000b000 ca:01 2089                       /lib/x86_64-linux-gnu/libnss_files-2
7f141672f000-7f1416730000 r--p 0000a000 ca:01 2089                       /lib/x86_64-linux-gnu/libnss_files-2
7f1416730000-7f1416731000 rw-p 0000b000 ca:01 2089                       /lib/x86_64-linux-gnu/libnss_files-2
7f1416731000-7f1416737000 rw-p 00000000 00:00 0
7f1416737000-7f141674e000 r-xp 00000000 ca:01 2086                       /lib/x86_64-linux-gnu/libnsl-2.27.so
```

```
7ffd7ba0a000-7ffd7ba2b000 rw-p 00000000 00:00 0                          [stack]
7ffd7ba91000-7ffd7ba94000 r--p 00000000 00:00 0                          [vvar]
7ffd7ba94000-7ffd7ba96000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
ubuntu@ip-172-31-13-223:~$
```

Process

Each process is also associated with a **set of resources**:

- **Registers**
- **File descriptors (used to access files, IO, pipes, sockets)**
- **CPU number, etc**

Process

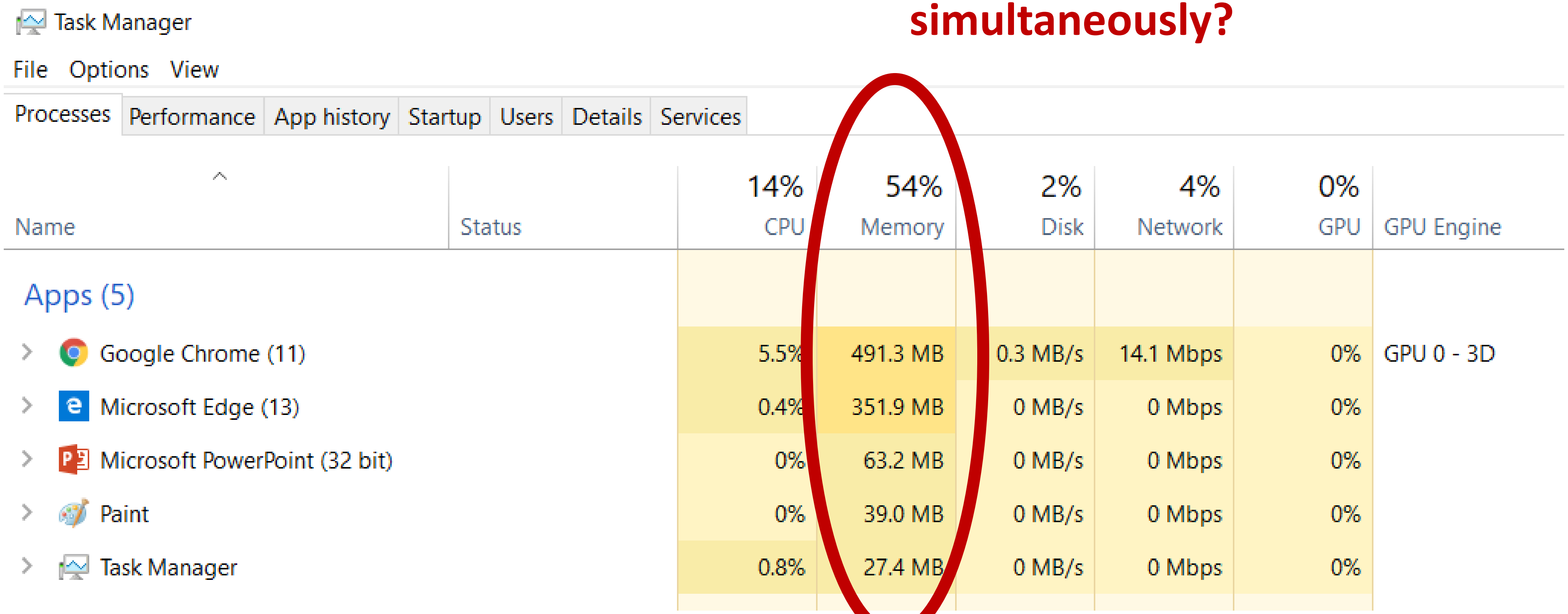
Processes exists in the context of **multiprogramming** or **multitasking** operating systems.

What’s this?

Process

Processes exists in the context of **multiprogramming** or **multitasking** operating systems.

How can several processes be in memory simultaneously?


Task Manager

File Options View

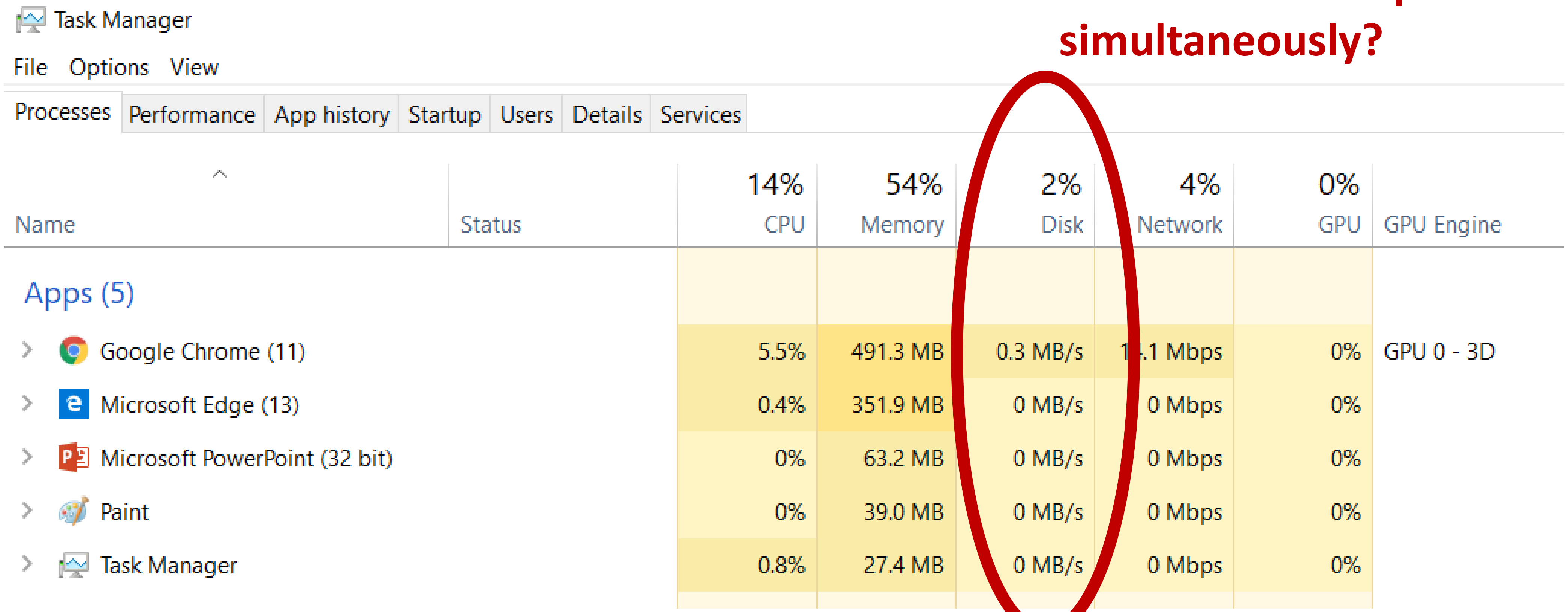
Processes | Performance | App history | Startup | Users | Details | Services

Name	Status	14% CPU	54% Memory	2% Disk	4% Network	0% GPU	GPU Engine
Apps (5)							
Google Chrome (11)		5.5%	491.3 MB	0.3 MB/s	14.1 Mbps	0%	GPU 0 - 3D
Microsoft Edge (13)		0.4%	351.9 MB	0 MB/s	0 Mbps	0%	
Microsoft PowerPoint (32 bit)		0%	63.2 MB	0 MB/s	0 Mbps	0%	
Paint		0%	39.0 MB	0 MB/s	0 Mbps	0%	
Task Manager		0.8%	27.4 MB	0 MB/s	0 Mbps	0%	

Process

Processes exists in the context of **multiprogramming** or **multitasking** operating systems.

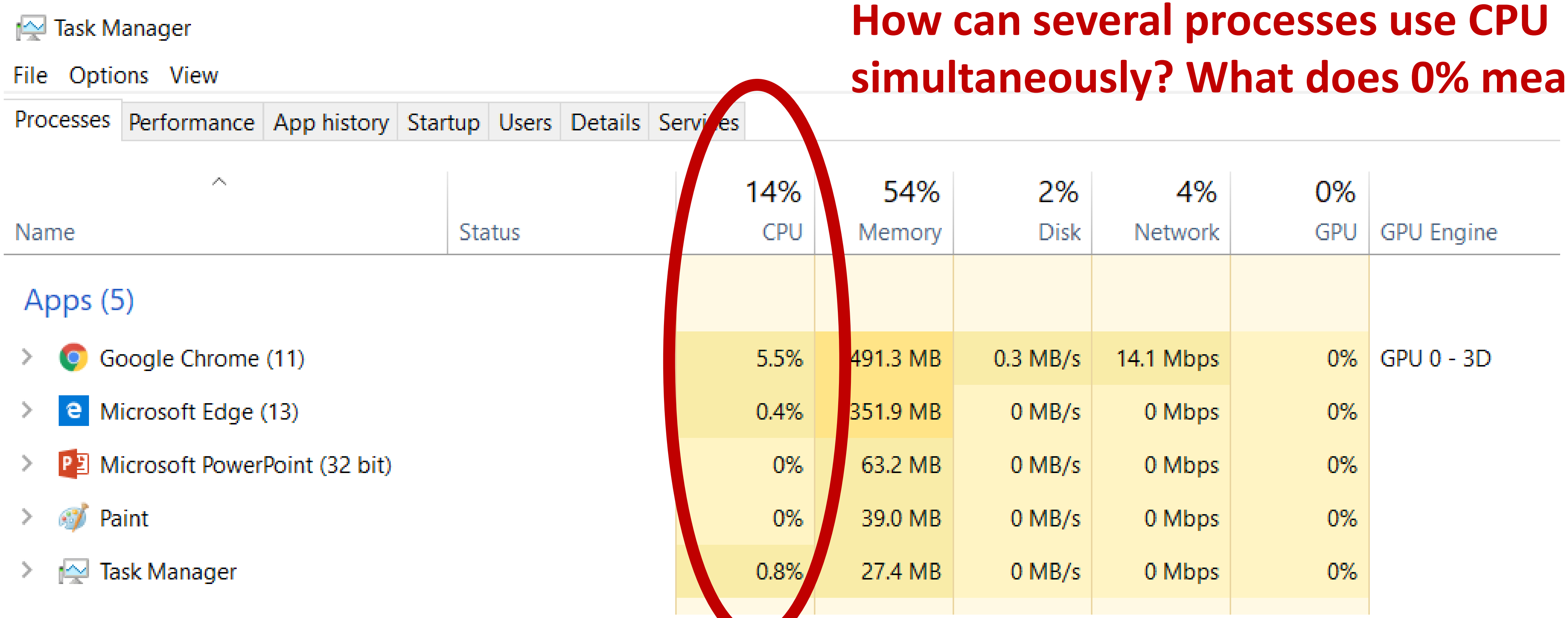
How can several processes use disk simultaneously?



Process

Processes exists in the context of **multiprogramming** or **multitasking** operating systems.

How can several processes use CPU simultaneously? What does 0% mean?

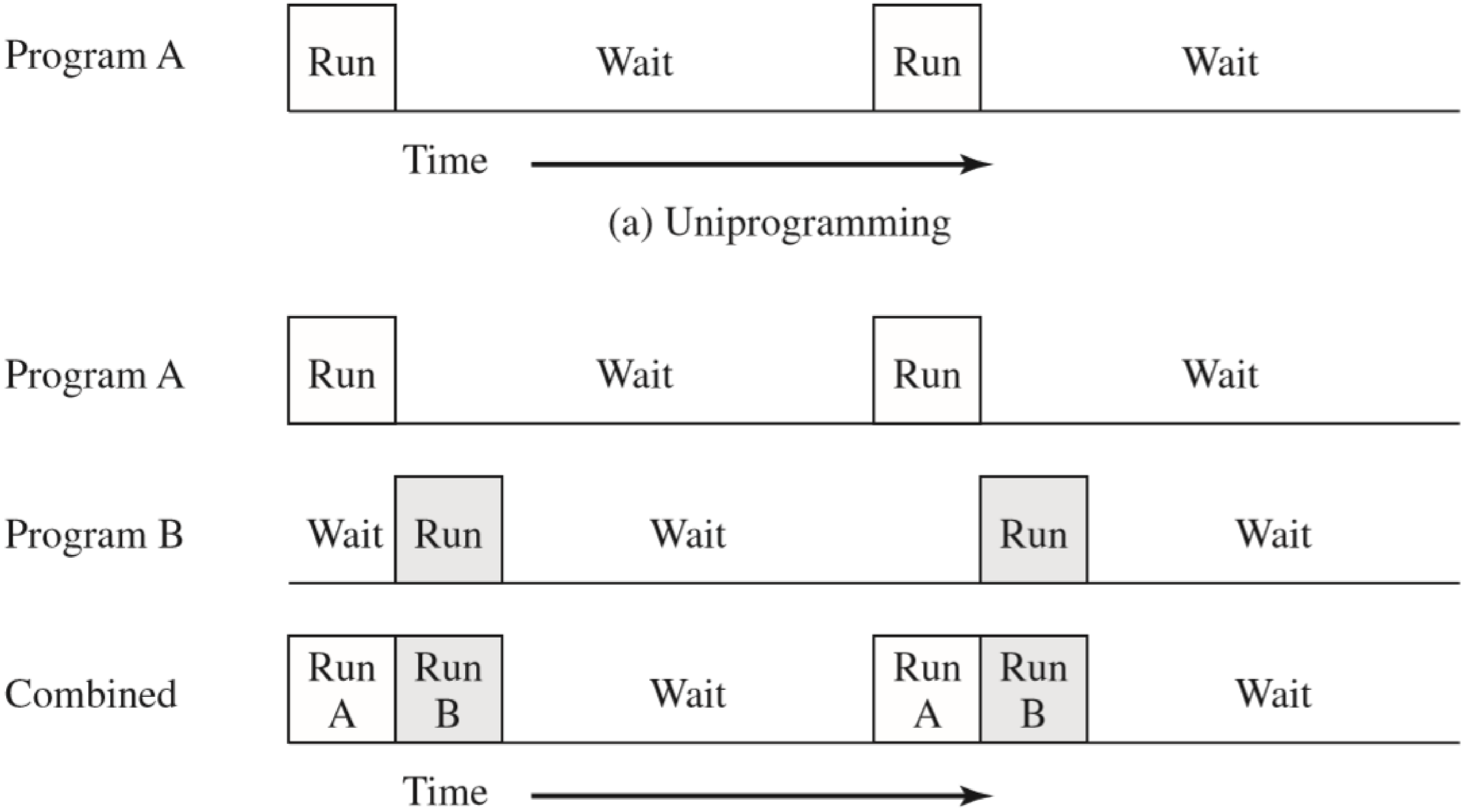

Task Manager

File Options View

Processes | Performance | App history | Startup | Users | Details | Services

Name	Status	14% CPU	54% Memory	2% Disk	4% Network	0% GPU	GPU Engine
Apps (5)							
Google Chrome (11)		5.5%	491.3 MB	0.3 MB/s	14.1 Mbps	0%	GPU 0 - 3D
Microsoft Edge (13)		0.4%	351.9 MB	0 MB/s	0 Mbps	0%	
Microsoft PowerPoint (32 bit)		0%	63.2 MB	0 MB/s	0 Mbps	0%	
Paint		0%	39.0 MB	0 MB/s	0 Mbps	0%	
Task Manager		0.8%	27.4 MB	0 MB/s	0 Mbps	0%	

Process

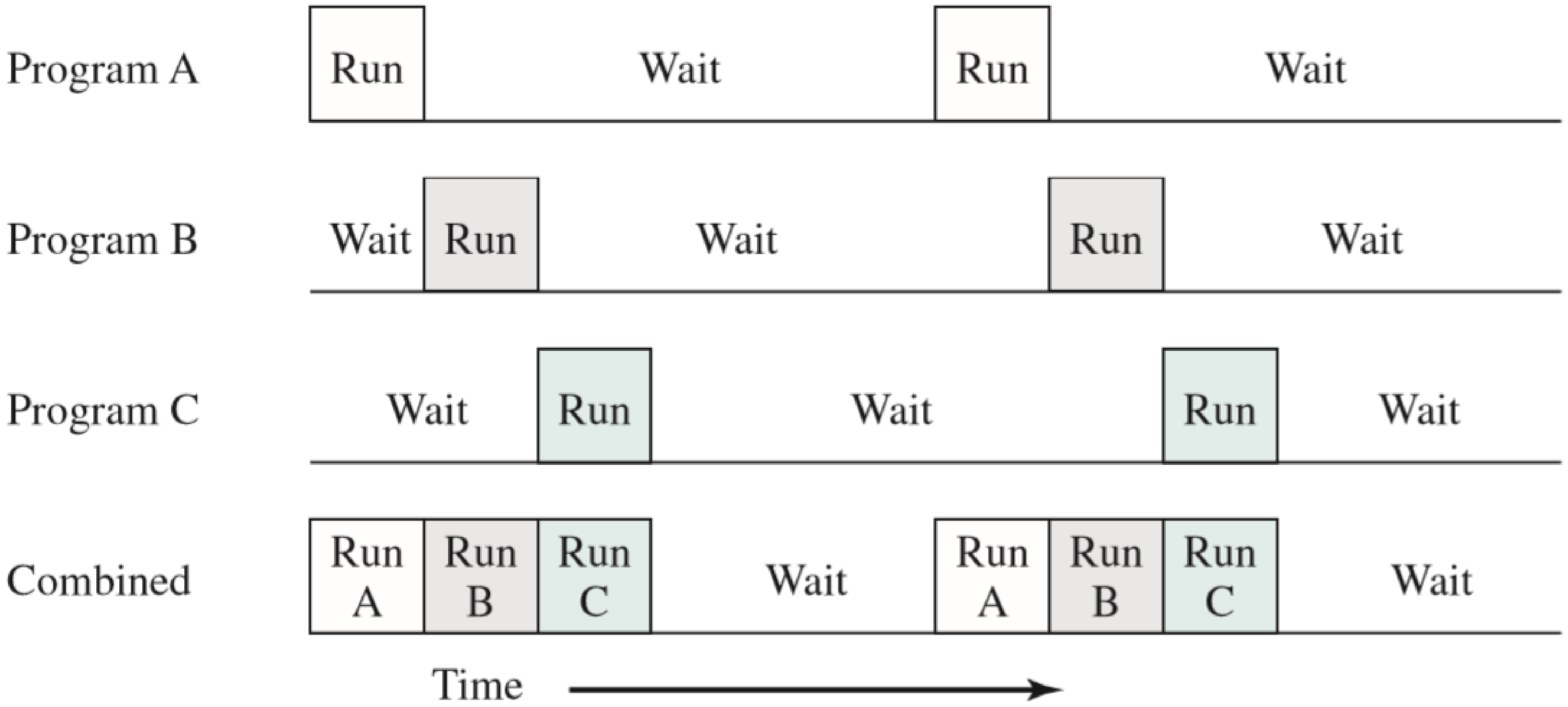


(a) Uniprogramming

(b) Multiprogramming with two programs

Stallings'

Process



(c) Multiprogramming with three programs

Stallings'

Process

- In a multiprogramming system, the kernel periodically grants **short periods of CPU access** to processes.
- When the kernel takes the CPU from a process, we say the process is **suspended**.

Process

- In a multiprogramming system, the kernel periodically grants **short periods of CPU access** to processes.
- When the kernel takes the CPU from a process, we say the process is **suspended**.

How long typically?

When does this happen?
(Hint: there's a few occasions)

Process

Once a process is suspended, it needs **certain information** to resume execution. So the Kernel keeps a **process table** (or **process control block**)

Process

Once a process is suspended, it needs **certain information** to resume execution. So the Kernel keeps a **process table** (or **process control block**)

- ID, name
- Process state (suspended, running, waiting for IO, etc)
- Priority
- Stack Pointer
- Program Counter **<- - Maybe in some implementations**
- File descriptors
- Address space
- Current CPU

torvalds / linux

Watch 6,618 | Star 68,296 | Fork 24,614

Code | Pull requests 248 | Projects 0 | Insights

Branch: master

linux / include / linux / sched.h

Find file | Copy path

torvalds Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net e874644 4 days ago

```c
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
	/*
	 * For reasons of header soup (see current_threa
	 * must be the first element of task_struct.
	 */
	struct thread_info		thread_info;
#endif
	/* -1 unrunnable, 0 runnable, >0 stopped: */
	volatile long			state;
```

```c
	void				*stack;
	atomic_t			usage;
	/* Per task flags (PF_*), defined further below:
	unsigned int			flags;
	unsigned int			ptrace;

#ifdef CONFIG_SMP
	struct llist_node		wake_entry;
	int				on_cpu;
```

```c
CONFIG_THREAD_INFO_IN_TASK
	/* Current CPU: */
	unsigned int			cpu;
```

```c
	int				on_rq;

	int				prio;
	int				static_prio;
	int				normal_prio;
	unsigned int			rt_priority;

```

Process

Max number of threads in Linux?

Process

Max number of threads in Linux?

man7.org > Linux > man-pages

0X3ff…f = 1,073 Millions

/proc/sys/kernel/threads-max (since Linux 2.3.11)

This file specifies the system-wide limit on the number of threads (tasks) that can be created on the system.

Since Linux 4.1, the value that can be written to *threads-max* is bounded. The minimum value that can be written is 20. The maximum value that can be written is given by the constant **FUTEX_TID_MASK** (0x3fffffff). If a value outside of this range is written to *threads-max*, the error **EINVAL** occurs.

The value written is checked against the available RAM pages. If the thread structures would occupy too much (more than 1/8th) of the available RAM pages, *threads-max* is reduced accordingly.

Process

Resources on the Web

- Linux: Processes

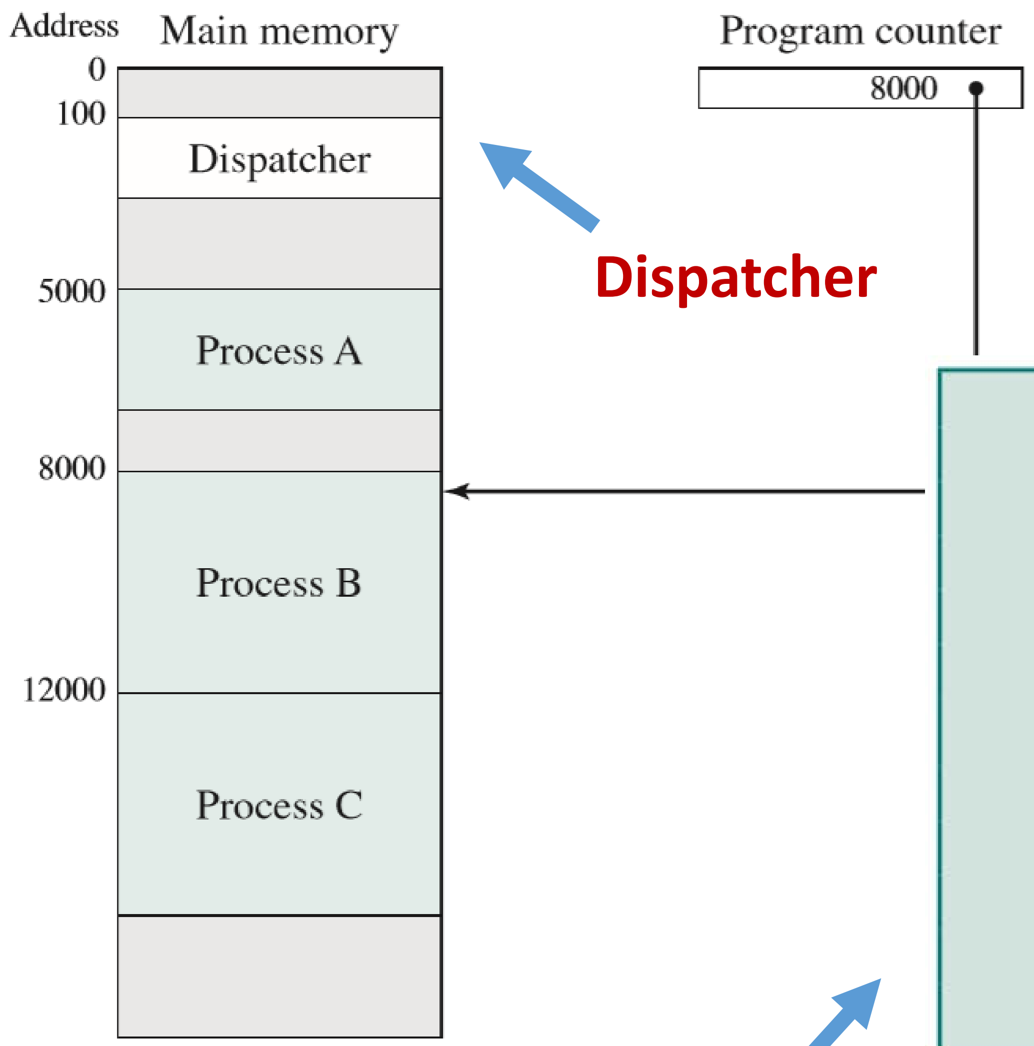
Process

Book reference

Section 3.1 and 2.2 (for multiprogramming definition)

The Life of a Process
(process states)

Consider the current state of a computer, with three process A, B and C executing concurrently



Dispatcher

(a) Trace of process A	(b) Trace of process B	(c) Trace of process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of process A
8000 = Starting address of program of process B
12000 = Starting address of program of process C

Figure 3.3 **Traces of Processes of Figure 3.2**

Sample trace of Instructions being executed

Stallings'

1	5000	
2	5001	
3	5002	A
4	5003	
5	5004	
6	5005	
---------------------Time-out		
7	100	
8	101	
9	102	
10	]103	Dispatcher
11	]104	
12	105	
13	8000	
14	8001	B
15	8002	
16	8003	
---------------------I/O request		
17	100	
18	101	
19	102	Dispatcher
20	103	
21	104	
22	105	
23	12000	
24	12001	C
25	12002	
26	12003	
27	12004	
28	12005	
---------------------Time-out		
29	100	
30	101	
31	102	Dispatcher
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	A
38	5009	
39	5010	
40	5011	
---------------------Time-out		
41	100	
42	101	
43	102	Dispatcher
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	C
50	12009	
51	12010	
52	12011	
---------------------Time-out		

Sample concurrent execution for a single core

Stallings'

The Life of a process

- Newly created processes begin in a **new state.** It’s process table was created, but not yet loaded into memory.

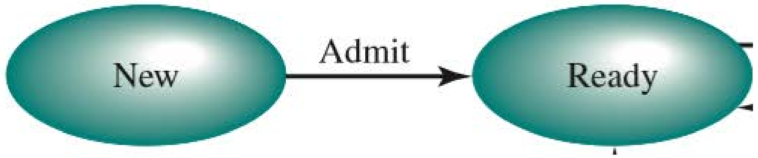
  (at this point they’re **not eligible to get picked by the dispatcher**).

Stallings’

The Life of a process

- Once loaded to memory the move to a ready queue, and their state is now in **ready state**.

  (at this point **they are eligible to get picked by the dispatcher**).



Stallings'

The Life of a process

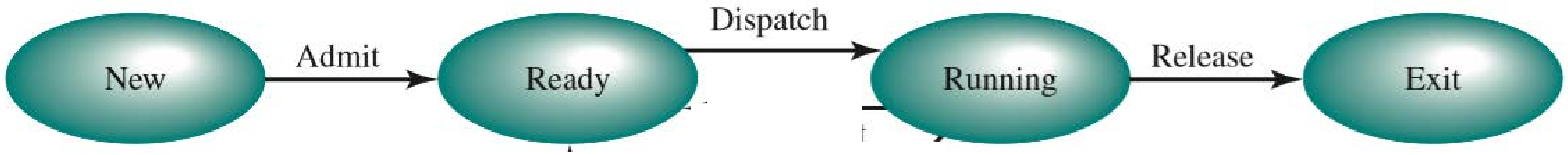
- When their turn comes, they'll be moved to **running state**, and the dispatcher will give them control of the CPU.

Stallings'

The Life of a process

- While in execution, a process terminate execution, thus moving to an **exit state**.

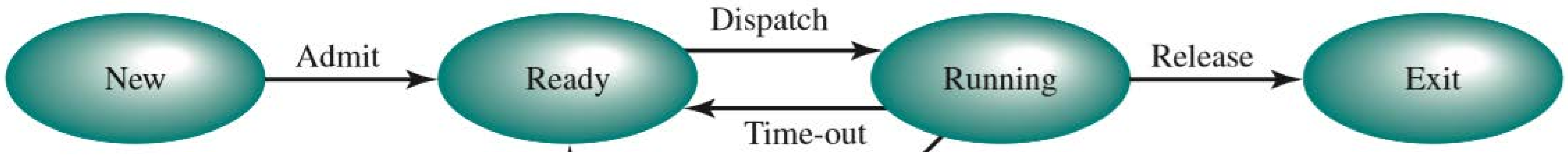
  (Processes in exit state have been removed from the ready queue)



Stallings'

The Life of a process

- While in execution, a process could also get placed back to the ready queue and into **ready state**, when a tick occurs.

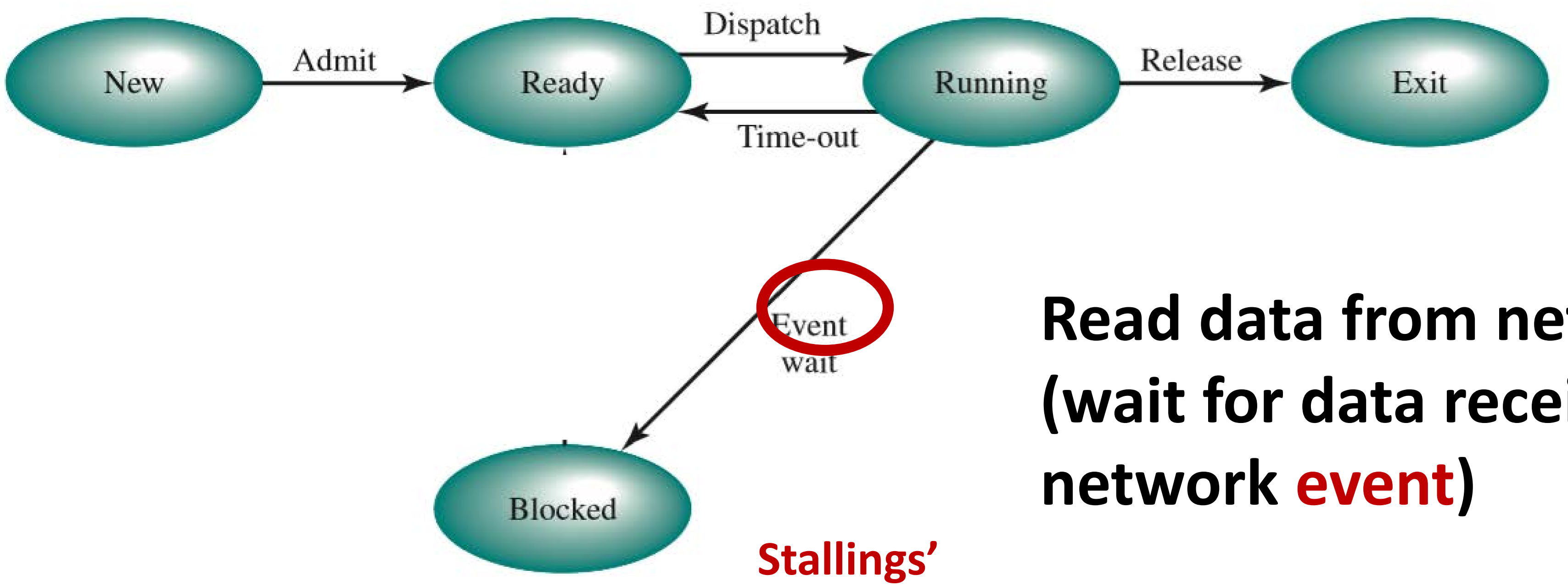


Stallings'

The Life of a process

- While in execution, a process could also request for IO (say *System.in.readLine()*) and get into a **Blocked state**.

(when blocked, processes wait in the blocked queue)

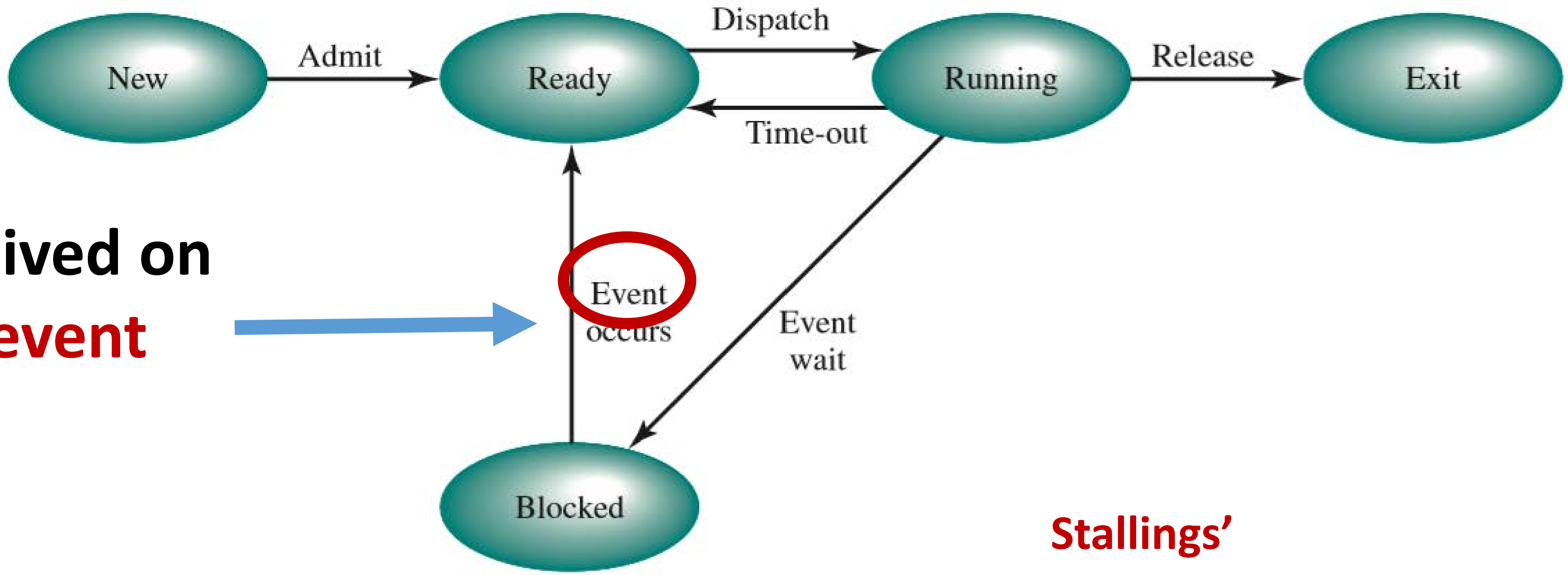


Stallings'

Read data from network (wait for data received on network event)

The Life of a process

- A blocked process gets unblocked once the IO it's waiting for arrives, and goes back to **ready state**, and on the ready queue.

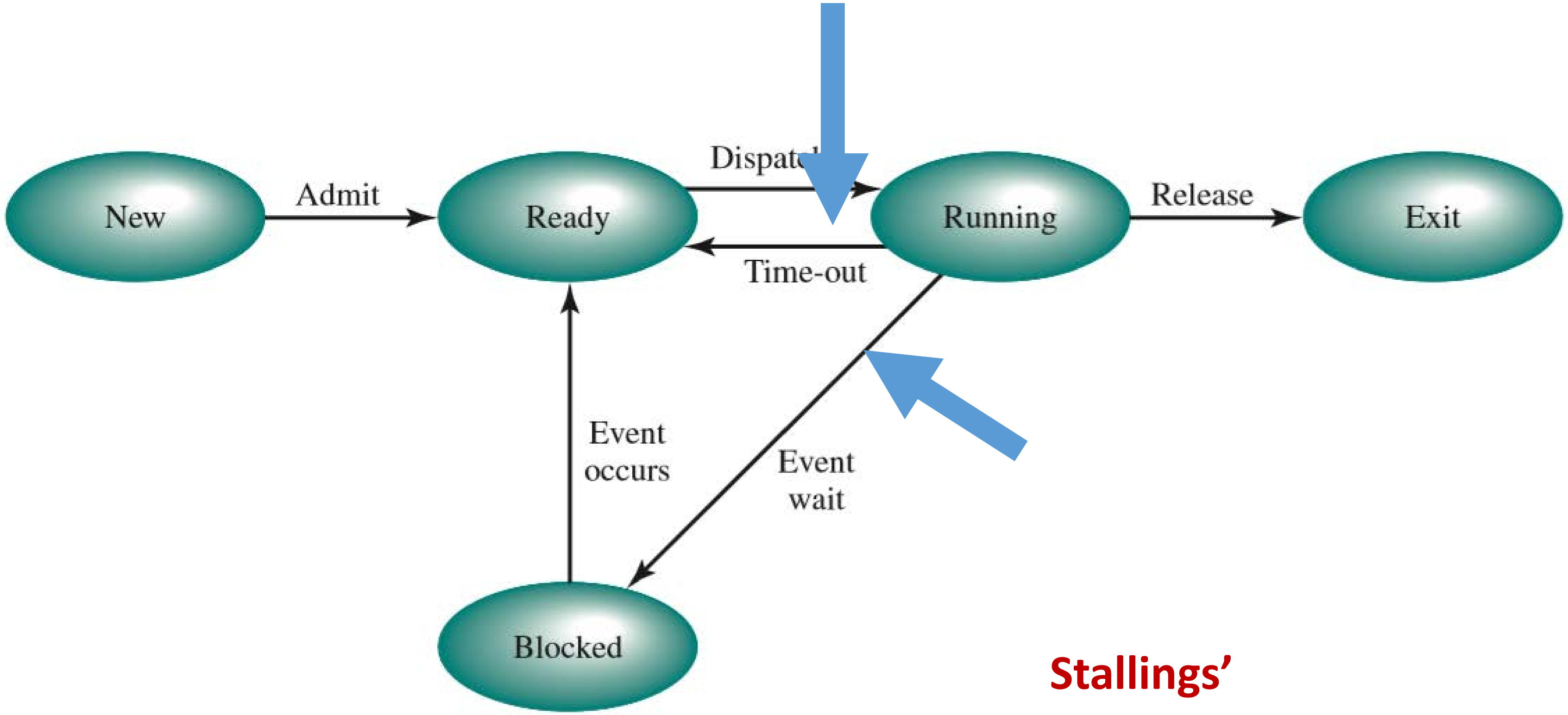


Data received on network event

Stallings'

The Life of a process

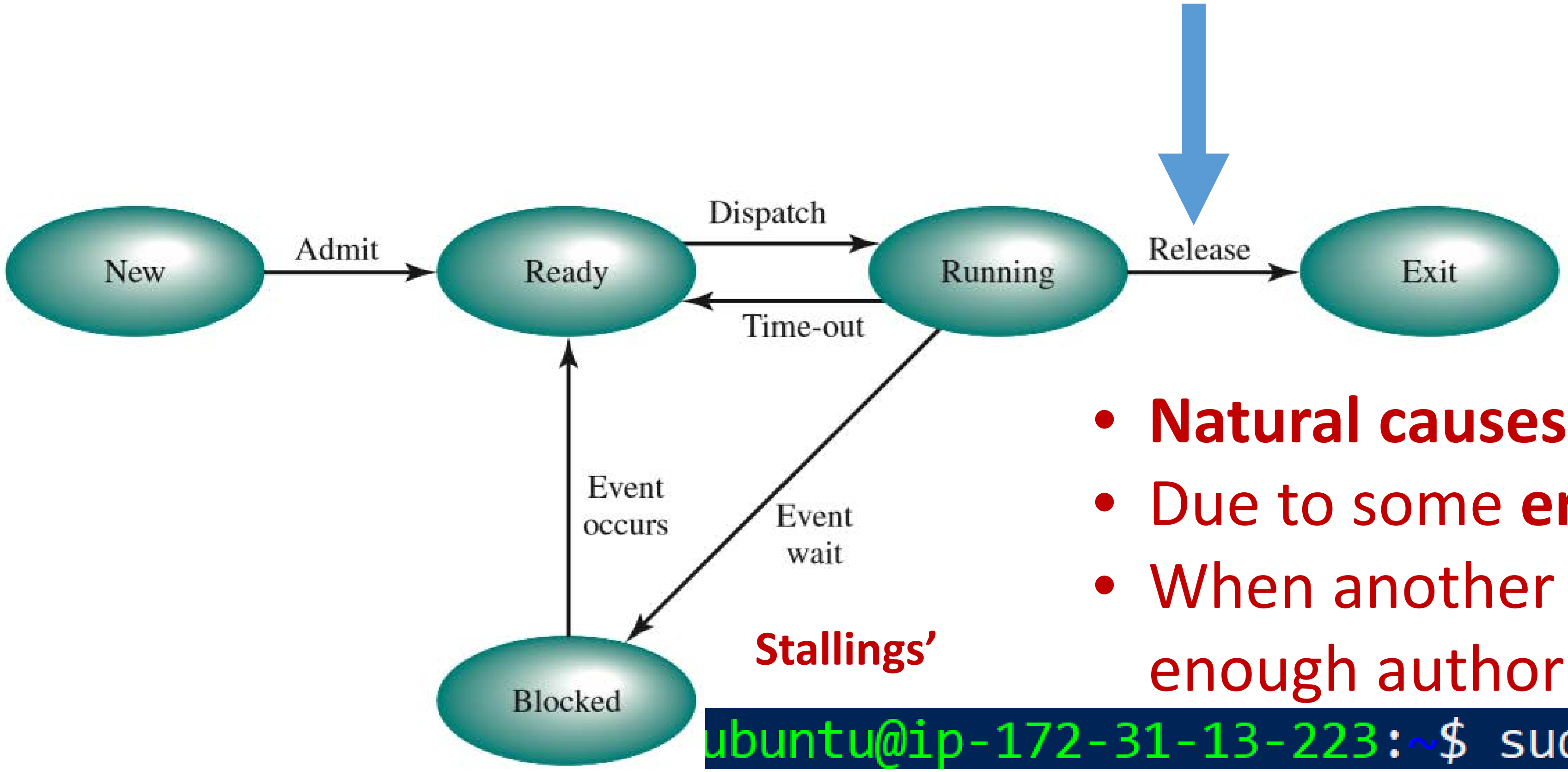
- A process can get **pre-empted (being robbed of the CPU)** for two reasons:



Stallings'

The Life of a process

- A process can get **terminated (being robbed of its glorious life as process)** for a few reasons:



Stallings'

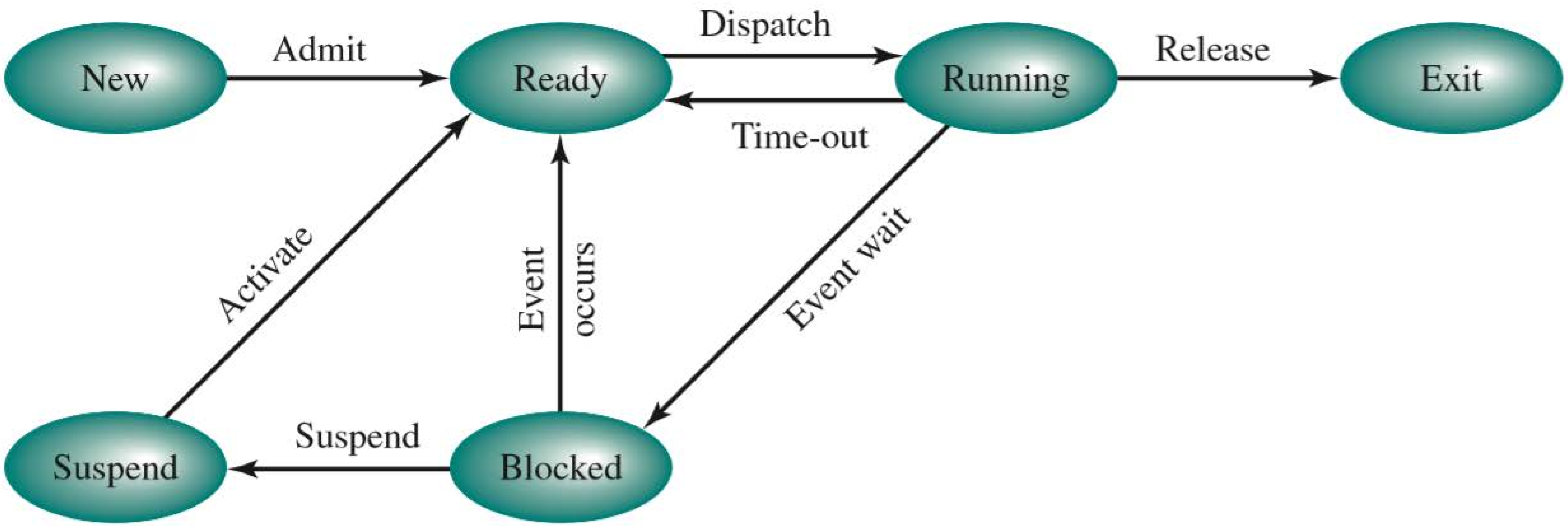
- **Natural causes**
- Due to some **error/violation**
- When another process with enough authority **kills it**.

```
ubuntu@ip-172-31-13-223:~$ sudo kill -9 727
```

The Life of a process

Activity (5mins).

Read page 144 from the book and explain this new “suspend” state.



torvalds / linux

<> Code | Pull requests 248 | Projects 0 | Insig

Branch: master ▾ linux / include / linux / sched.h

torvalds Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem

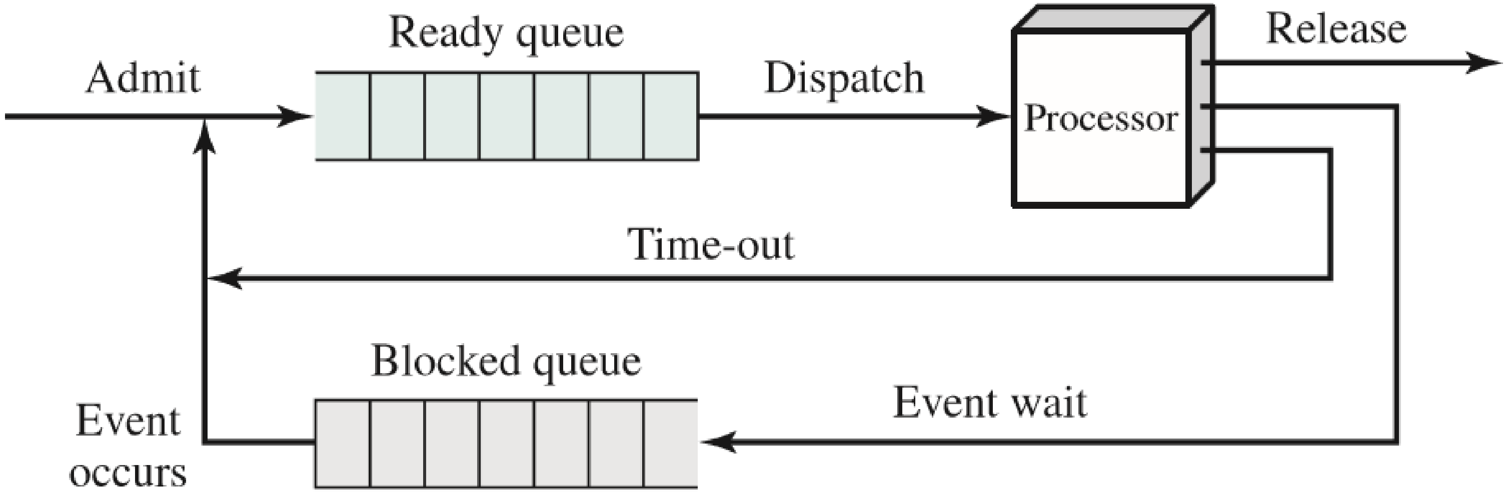
```c
/* Used in tsk->state: */
#define TASK_RUNNING                    0x0000
#define TASK_INTERRUPTIBLE              0x0001
#define TASK_UNINTERRUPTIBLE            0x0002
#define __TASK_STOPPED                  0x0004
#define __TASK_TRACED                   0x0008
/* Used in tsk->exit_state: */
#define EXIT_DEAD                       0x0010
#define EXIT_ZOMBIE                     0x0020
#define EXIT_TRACE                      (EXIT_ZOMBIE | EXIT_DEAD
```

```c
/* Used in tsk->state again: */
#define TASK_PARKED                     0x0040
#define TASK_DEAD                       0x0080
#define TASK_WAKEKILL                   0x0100
#define TASK_WAKING                     0x0200
#define TASK_NOLOAD                     0x0400
#define TASK_NEW                        0x0800
#define TASK_STATE_MAX                  0x1000
/* Convenience macros for the sake of set_current_s
#define TASK_KILLABLE                   (TASK_WAKEK
#define TASK_STOPPED                    (TASK_WAKEK
#define TASK_TRACED                     (TASK_WAKEK

#define TASK_IDLE                       (TASK_UNINT
```

The Life of a process

The **ready queue** and **blocked queue**

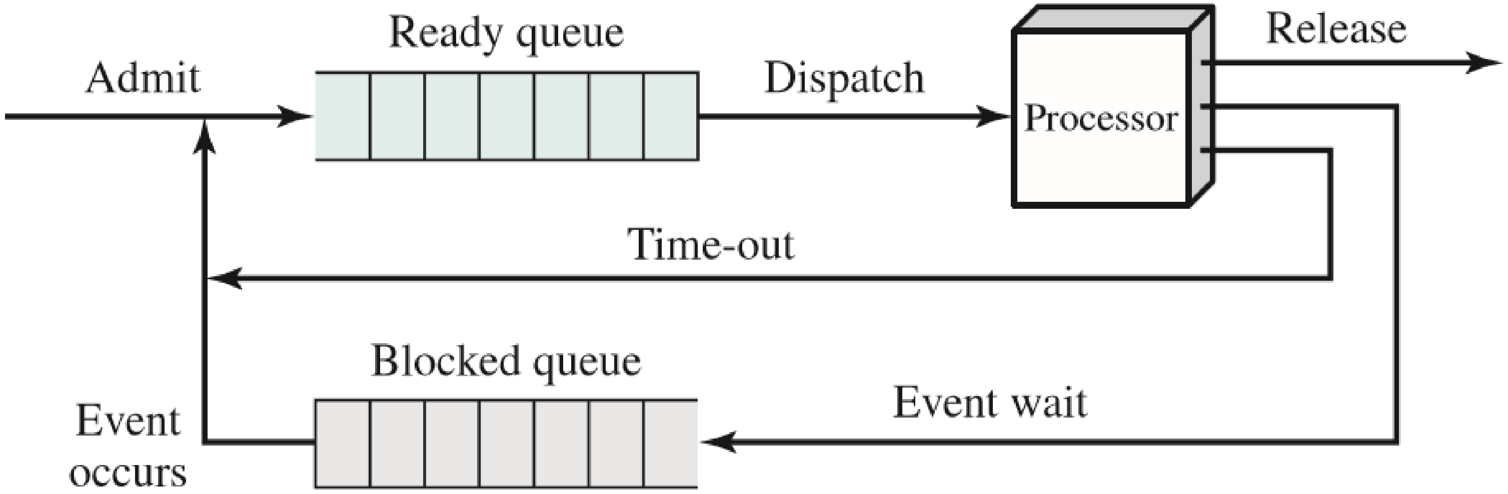


(a) Single blocked queue

Stallings'

The Life of a process

The **ready queue** and **blocked queue**



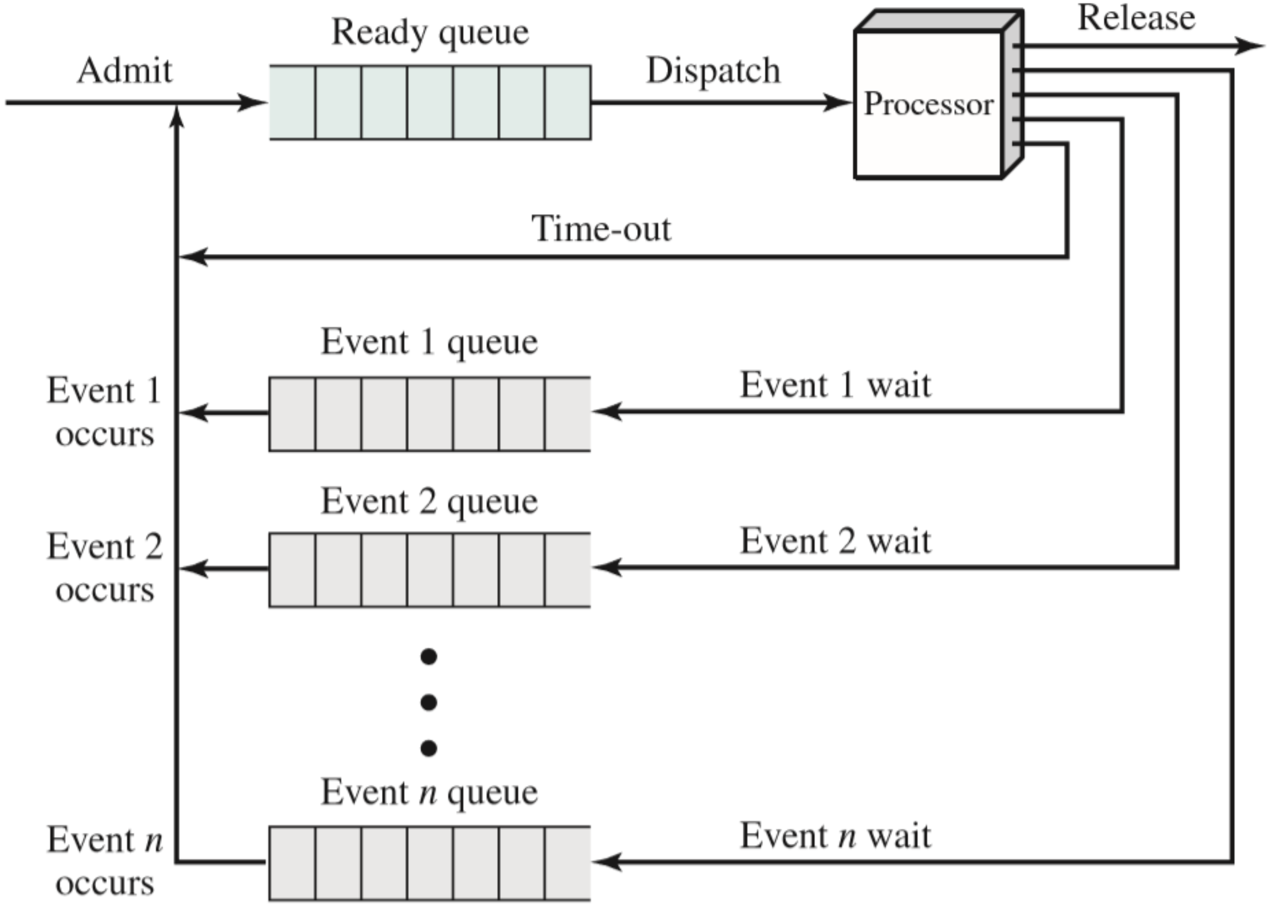
(a) Single blocked queue

Stallings'

What's the problem with having a single blocked queue?
Hint: O(n)

The Life of a process

The **ready queue** and **blocked queue**



Stallings'

Solution:

Multiple Blocked Queues

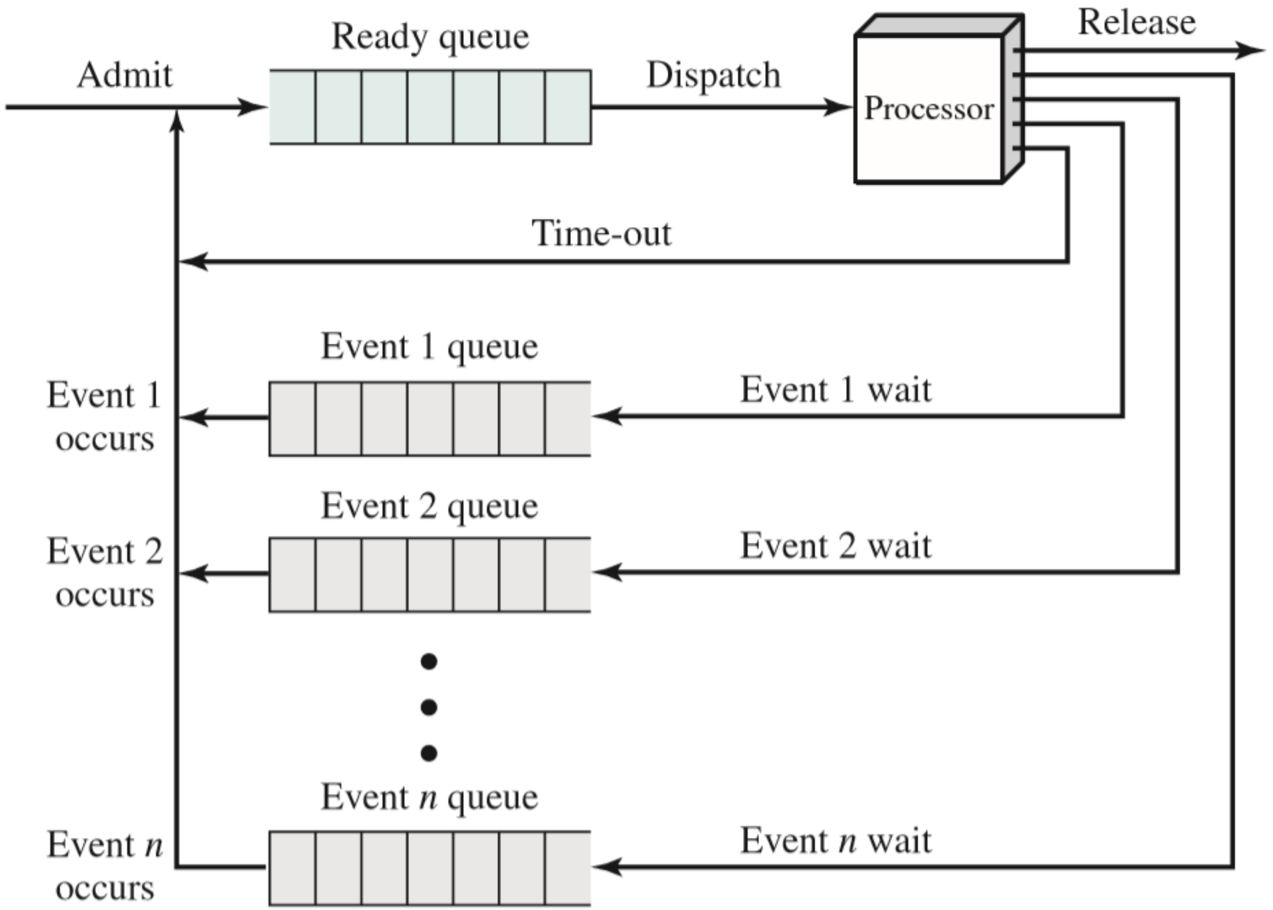
When an *event n* occurs
The whole *event n queue*
Is moved to the ready queue

The Life of a process

The **ready queue** and **blocked queue**

What exactly can an event be?

That depends on IO drivers
(so many possibilities)



Stallings'

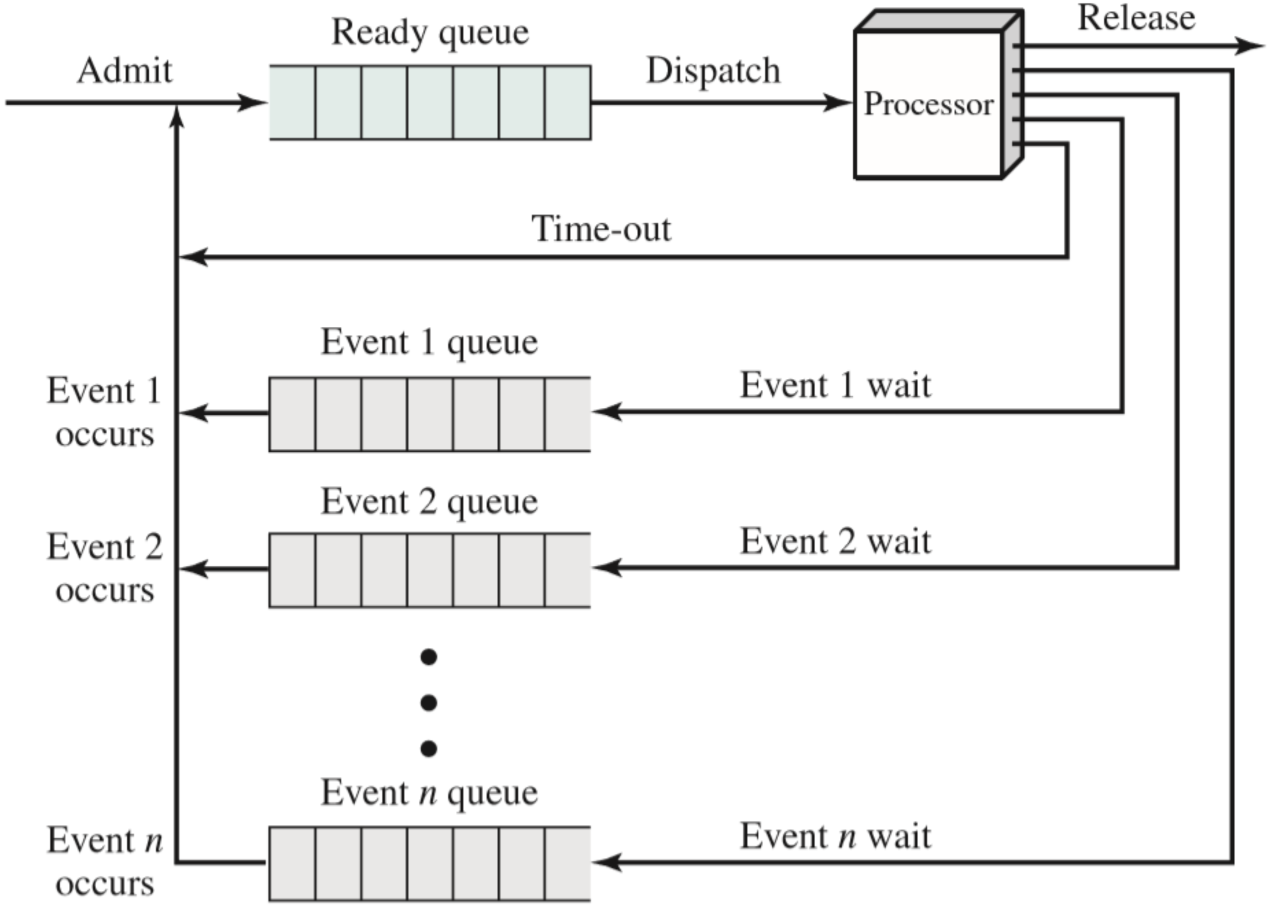
Solution:
Multiple Blocked Queues

When an *event n* occurs
The whole *event n queue*
Is moved to the ready queue

The Life of a process

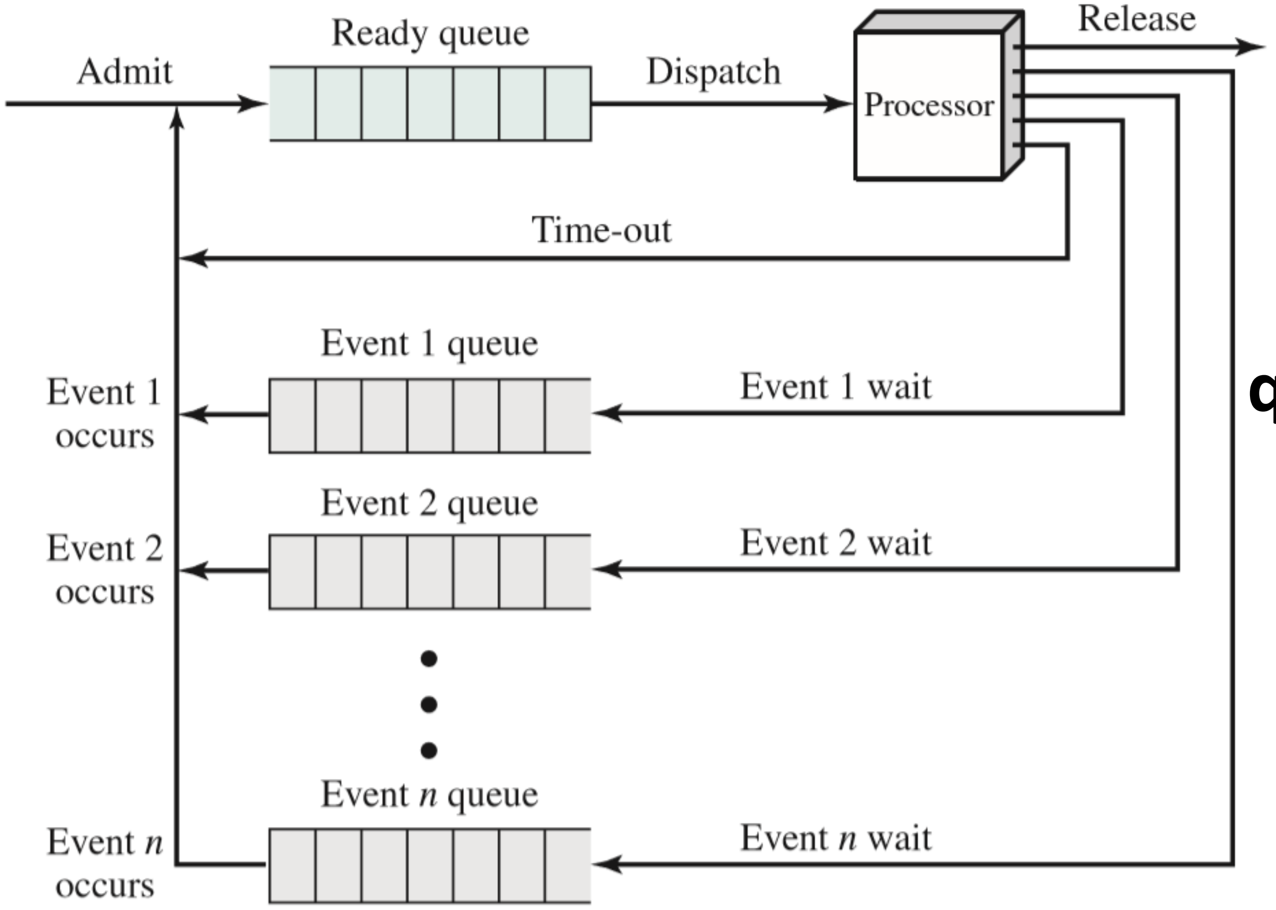
The **ready queue** and **blocked queue**

The Life of a process



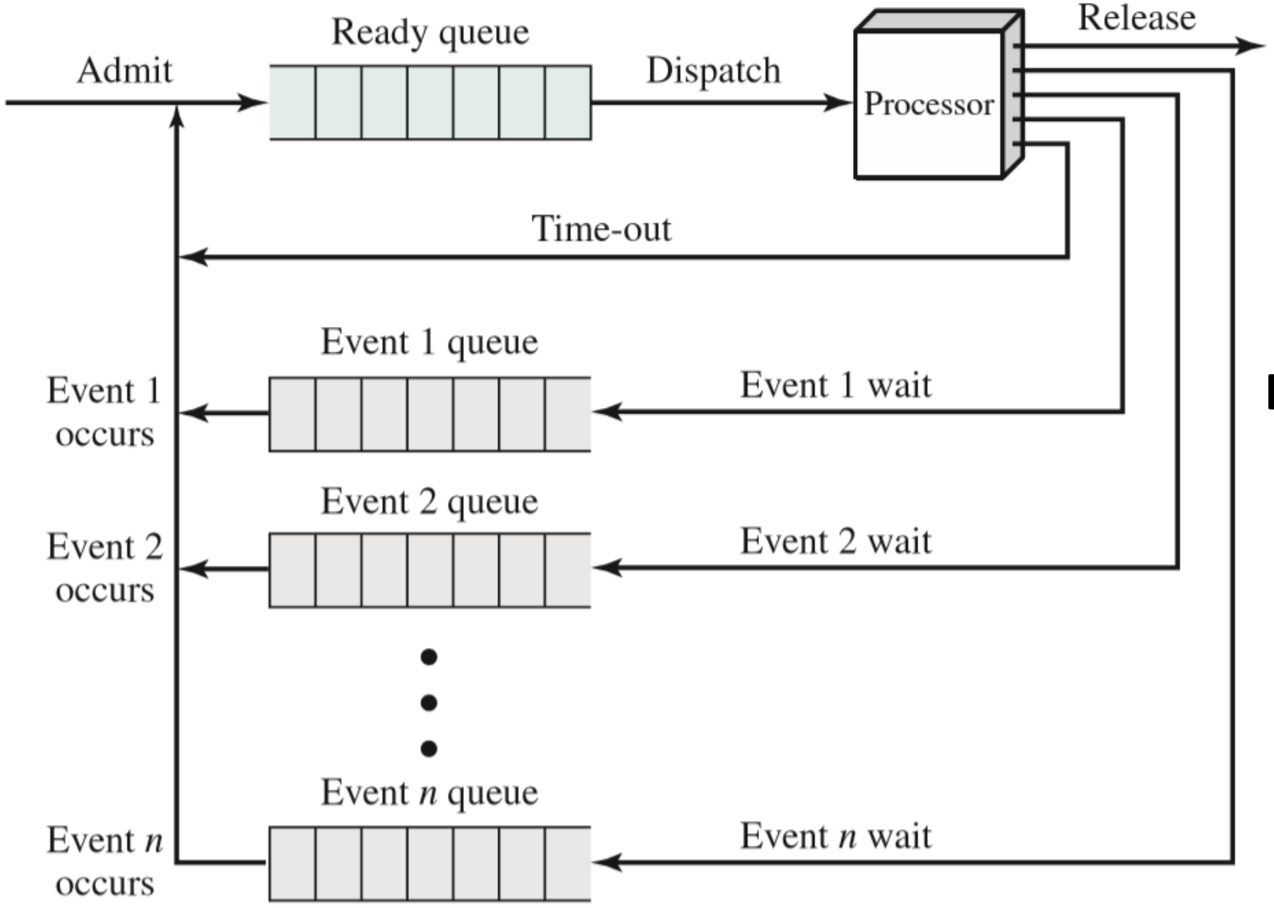
Example. Let's say a **user space process** makes a system call to requests a file to the **file system driver.**

The Life of a process



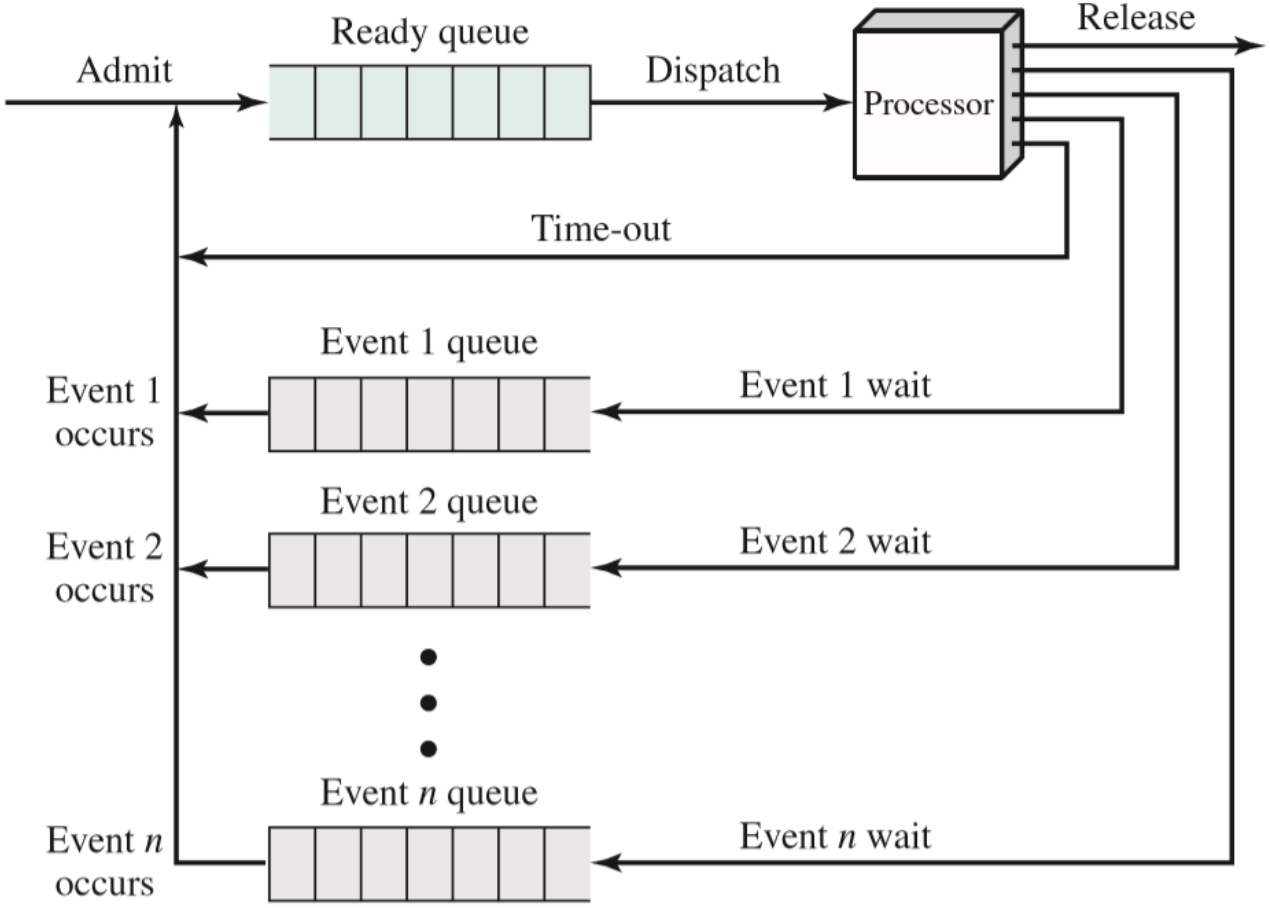
The file system driver will **create a queue** (for that file request) and **put the user process to sleep**

The Life of a process



The file system driver will **then requests the disk driver a sector in disk** where the file contents are stored; and will set up **a second wait queue for that sector in disk.**

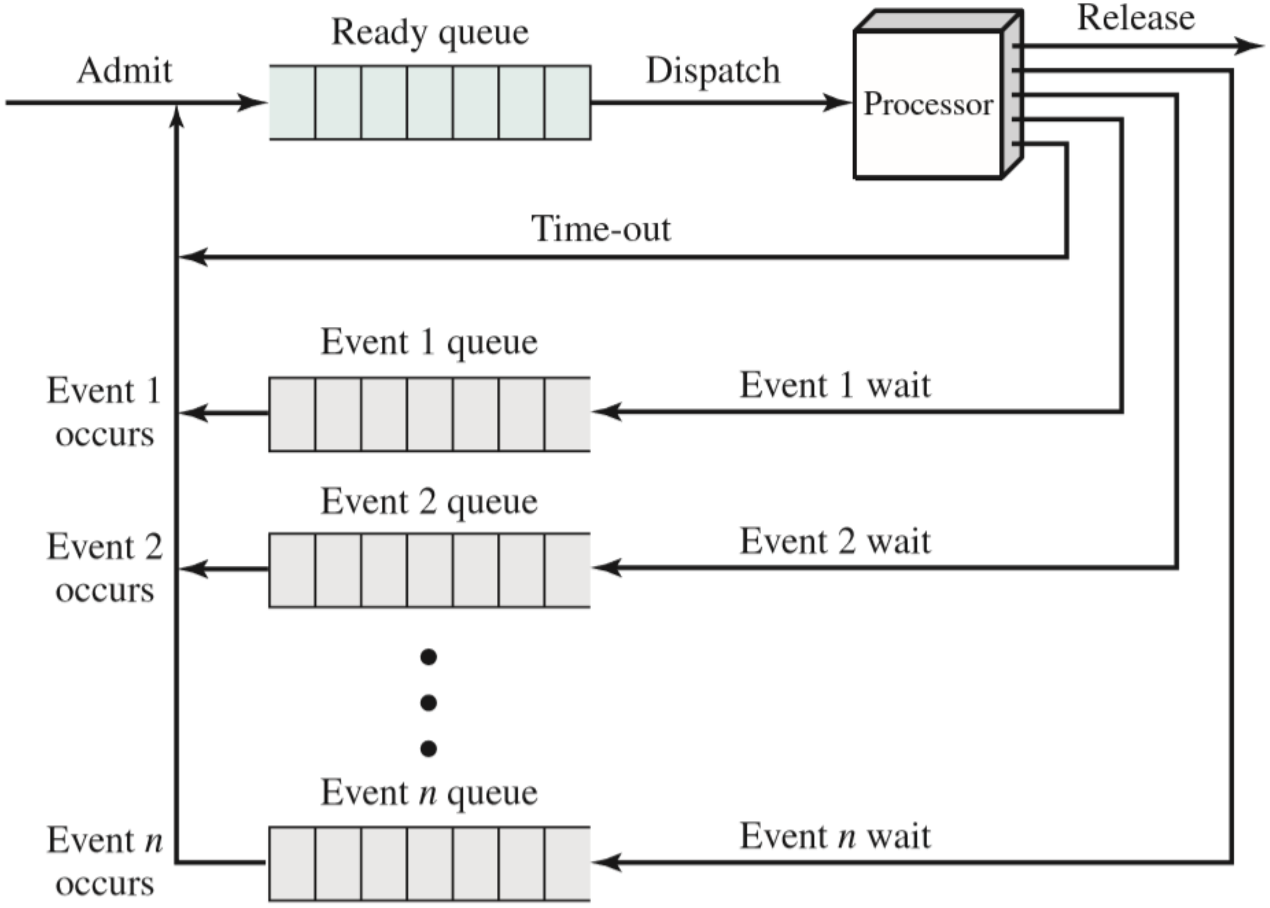
The Life of a process



There's some serious magic Going on here

In this case, the second queue is **associated with a piece of code** (a callback), not really a process.

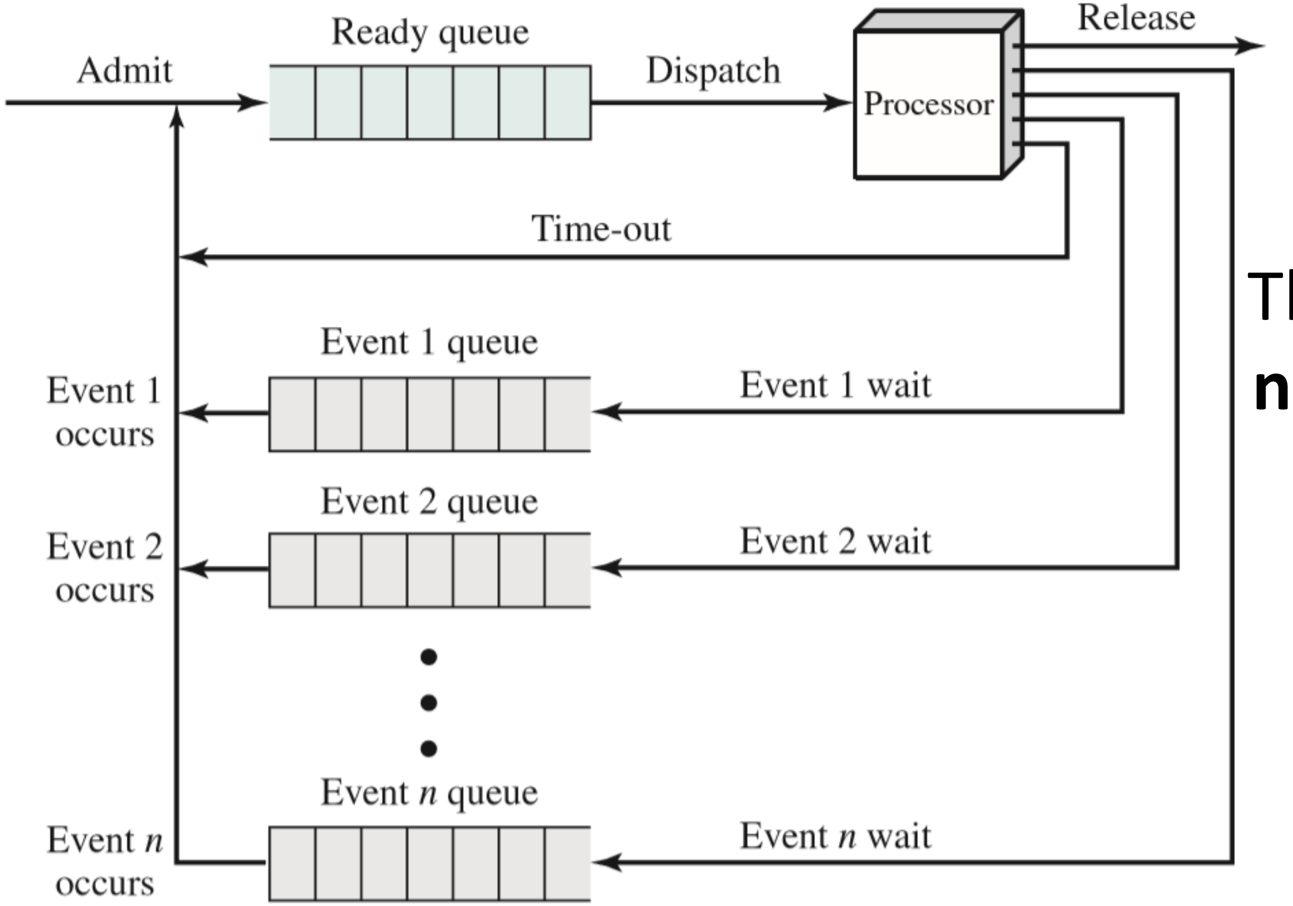
The Life of a process



If more requests arrive asking for the **same file**, they can be "placed"/"associated" in the same **wait queue**.

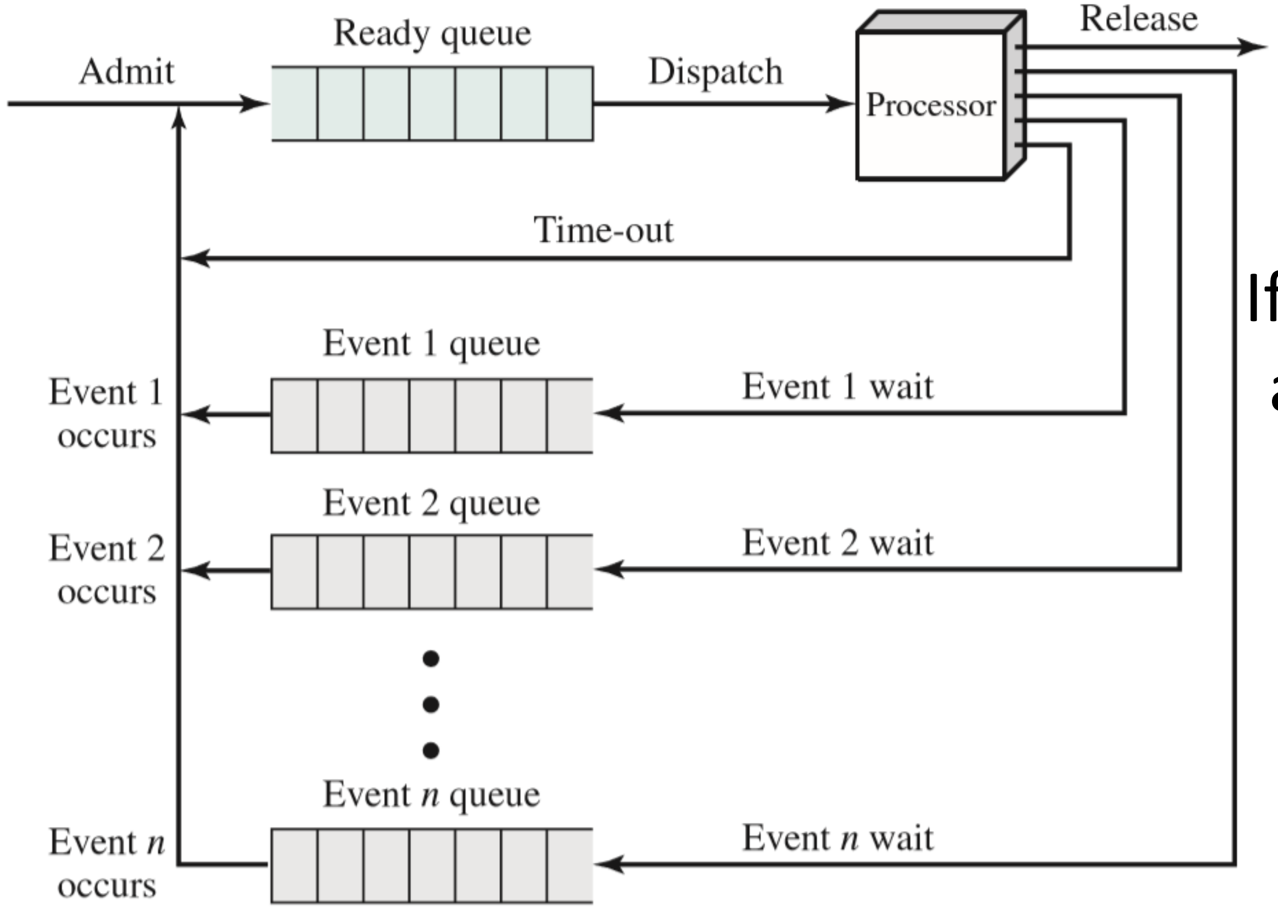
E.g.
Multiple receivers of the same file can Be handled within the same callback

The Life of a process



The disk driver will then **request the needed sector from disk** and create a **third queue**, and associate a callback with it.

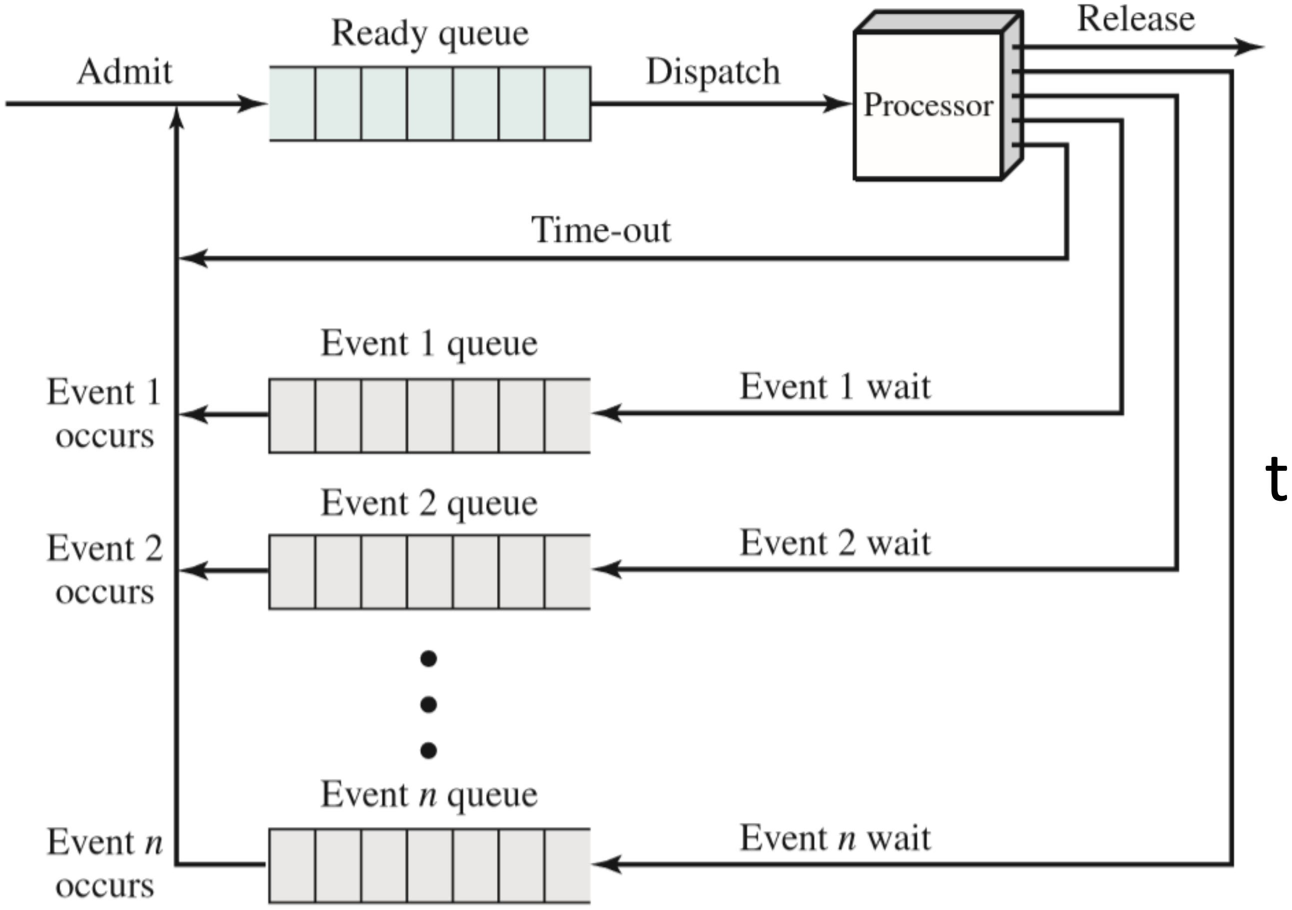
The Life of a process



If more requests for the **same sector** arrive. They can be associated with the same queue.

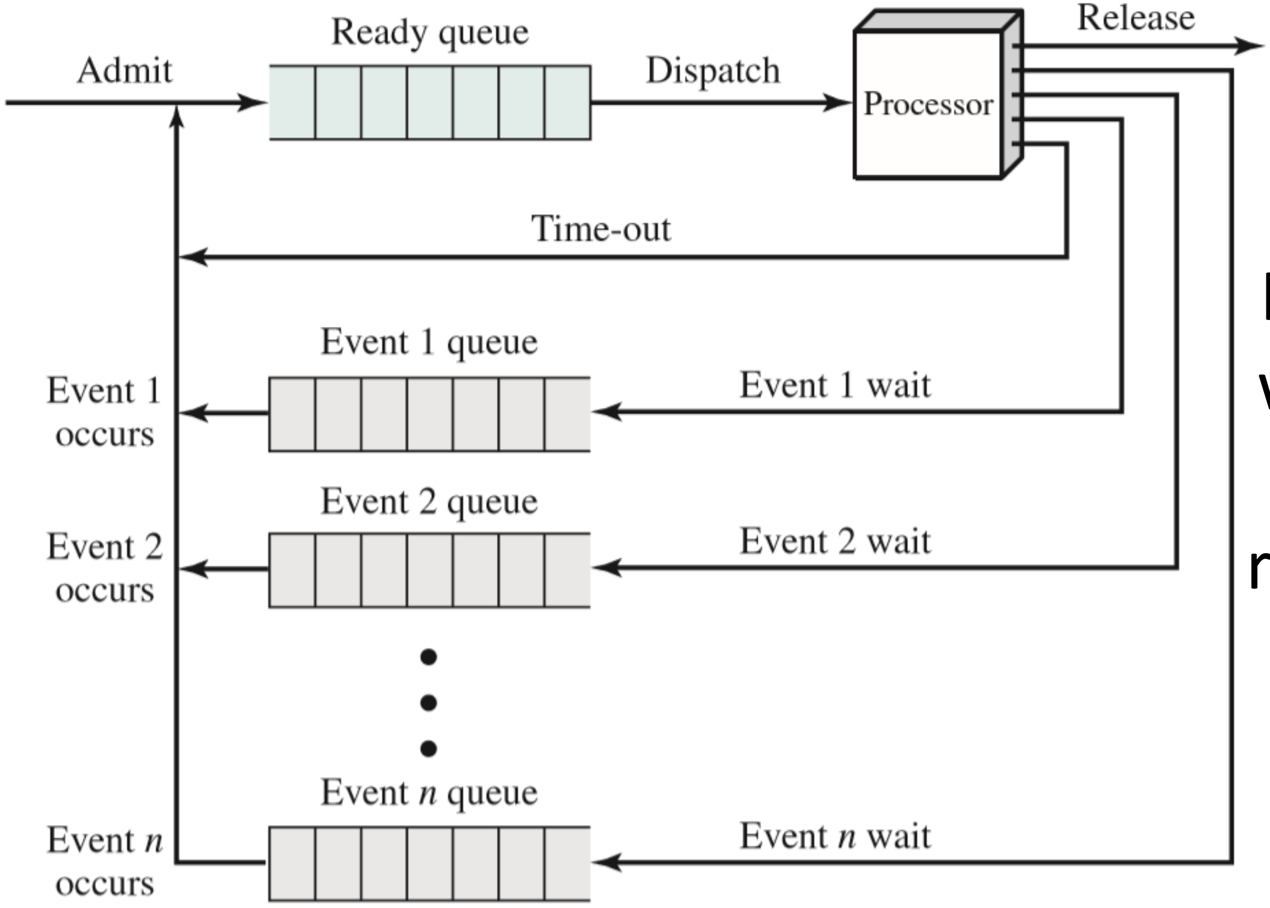
E.g.
Multiple receivers of the same file can Be handled within the same callback

The Life of a process



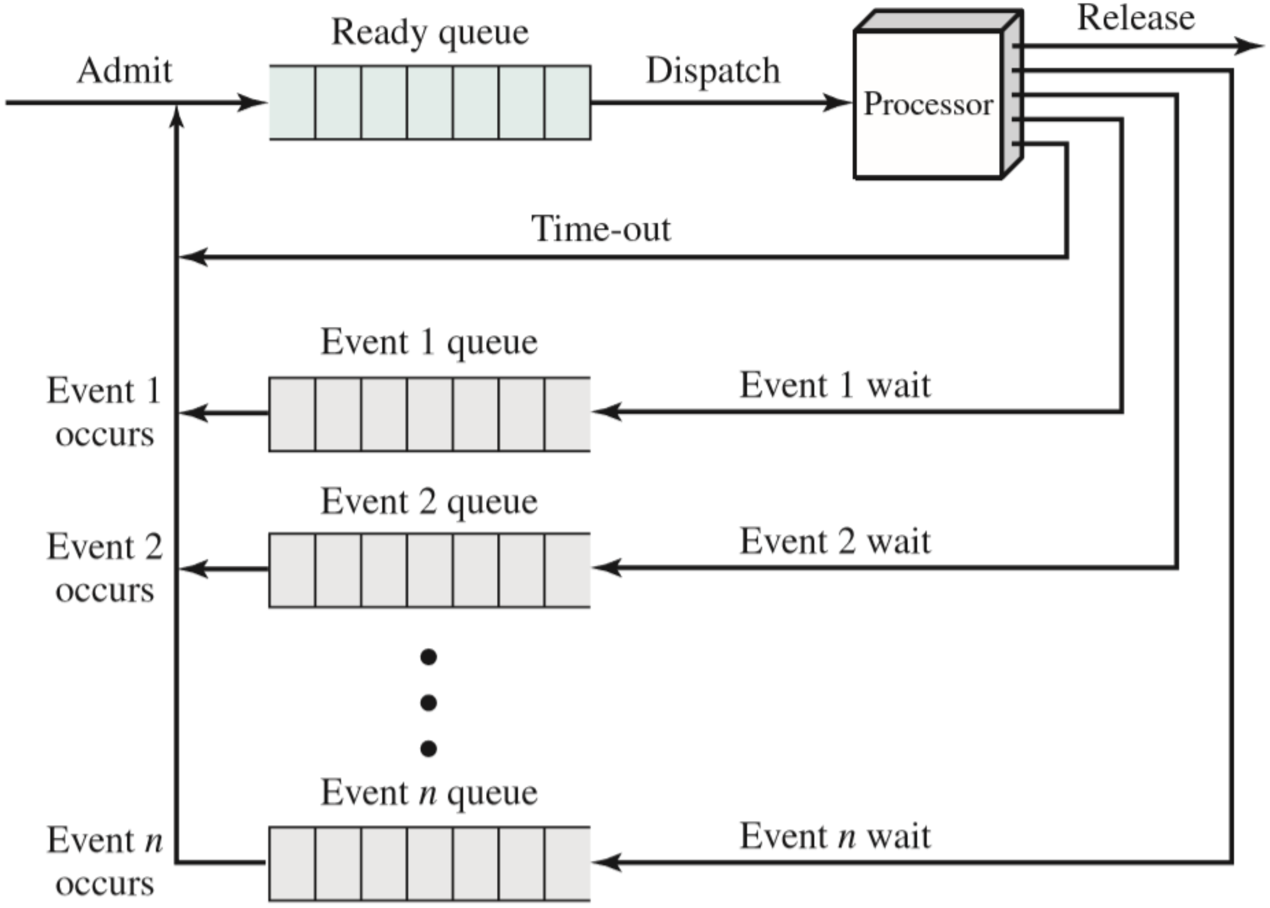
Eventually when the DMA has finished fetching the sector, it'll trigger a **disk interrupt (part of the disk driver)**.

The Life of a process



In the disk interrupt, the disk driver will **“wake up” the code waiting on the third event queue** (which remember is more disk driver code).

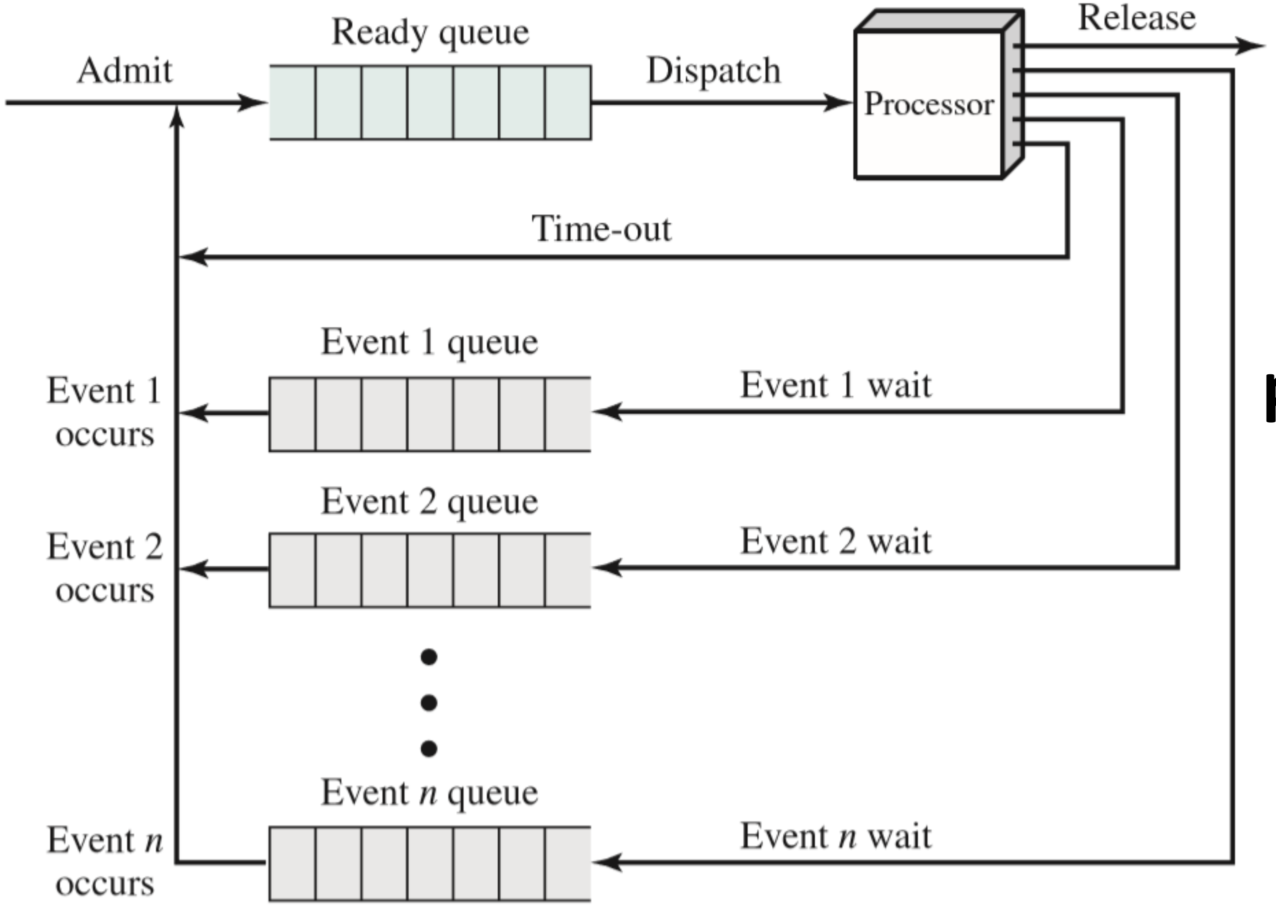
The Life of a process



From the **disk driver** and Associated with queue 3

This code will **“wake up” code waiting on the second queue**.

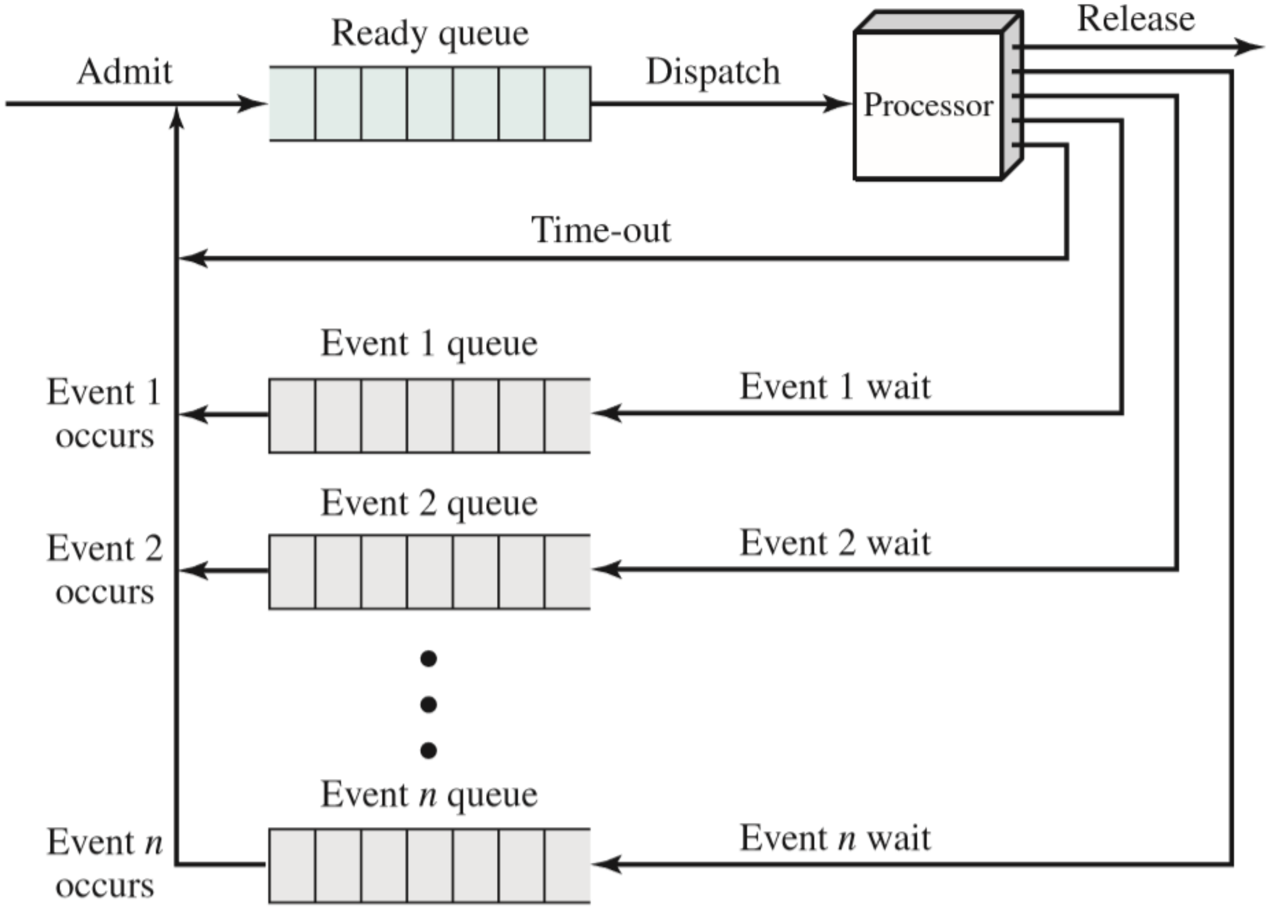
The Life of a process



From the **file driver** and Associated with queue 2

This code will **"wake up" the user process waiting on the first queue**.

The Life of a process

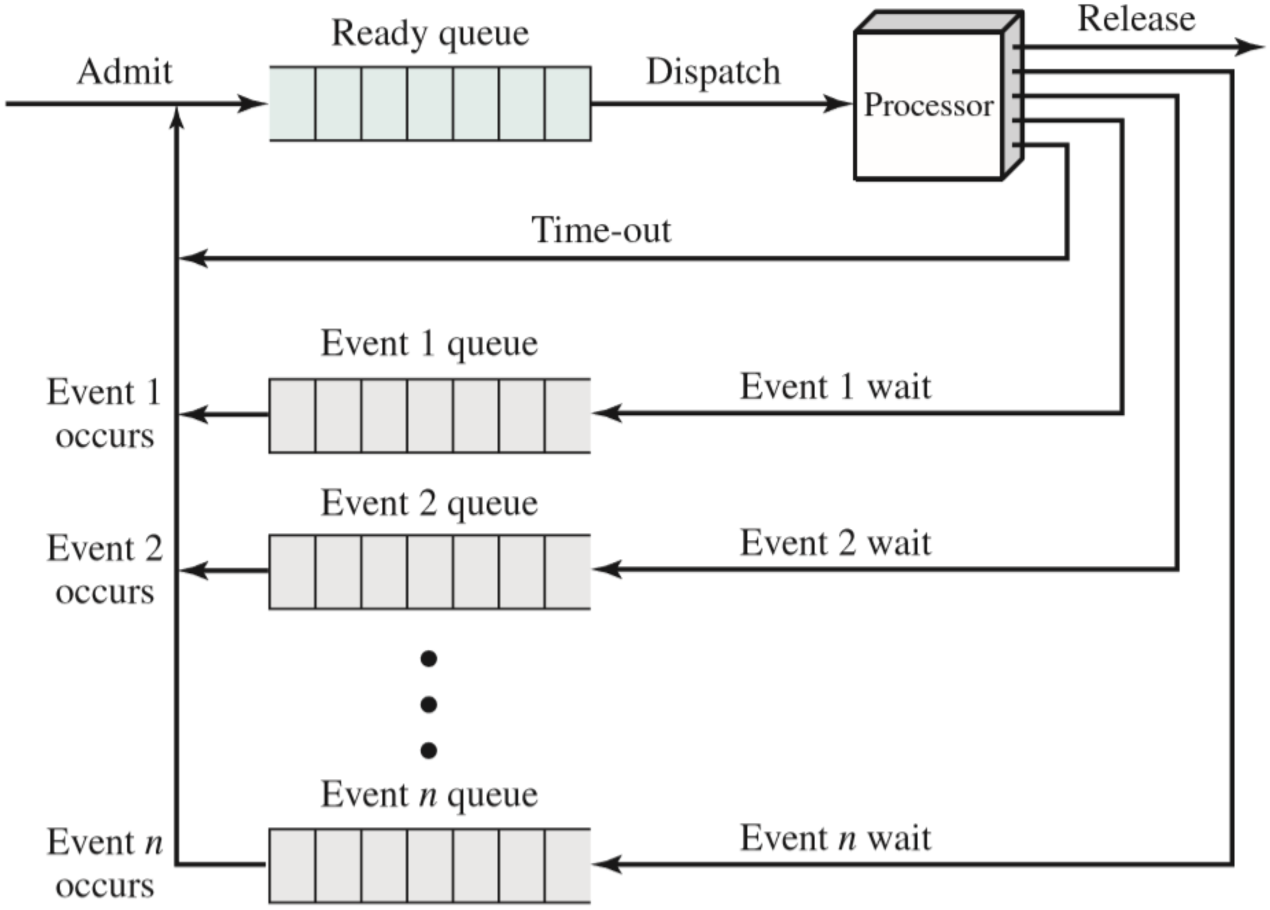


Additional notes:
Linux used to have the

"When an event n occurs
the whole event n queue
Is moved to the ready queue"

semantics, but then in 1999 a study found that **Linux had some Performance issues**.

The Life of a process



Part of the problem was the *Thundering herd problem*.

Short 5min activity where you find out what is it, and what it has to do with the

"When an event n occurs
the whole event n queue
Is moved to the ready queue"

semantics; and how it was solved. (hint: exclusive waiting, prepare_to_wait_exclusive())

The Life of a process

Book reference

Section 3.2