HPC & Parallel Programming

Thread

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Outline

- What is thread?
  - Multiple thread application
  - Thread vs Process
  - Advantage and disadvantage of thread
- Thread in Linux
- Thread design
  - Kernel space vs User space
  - Local thread vs Global thread scheduling

Process review

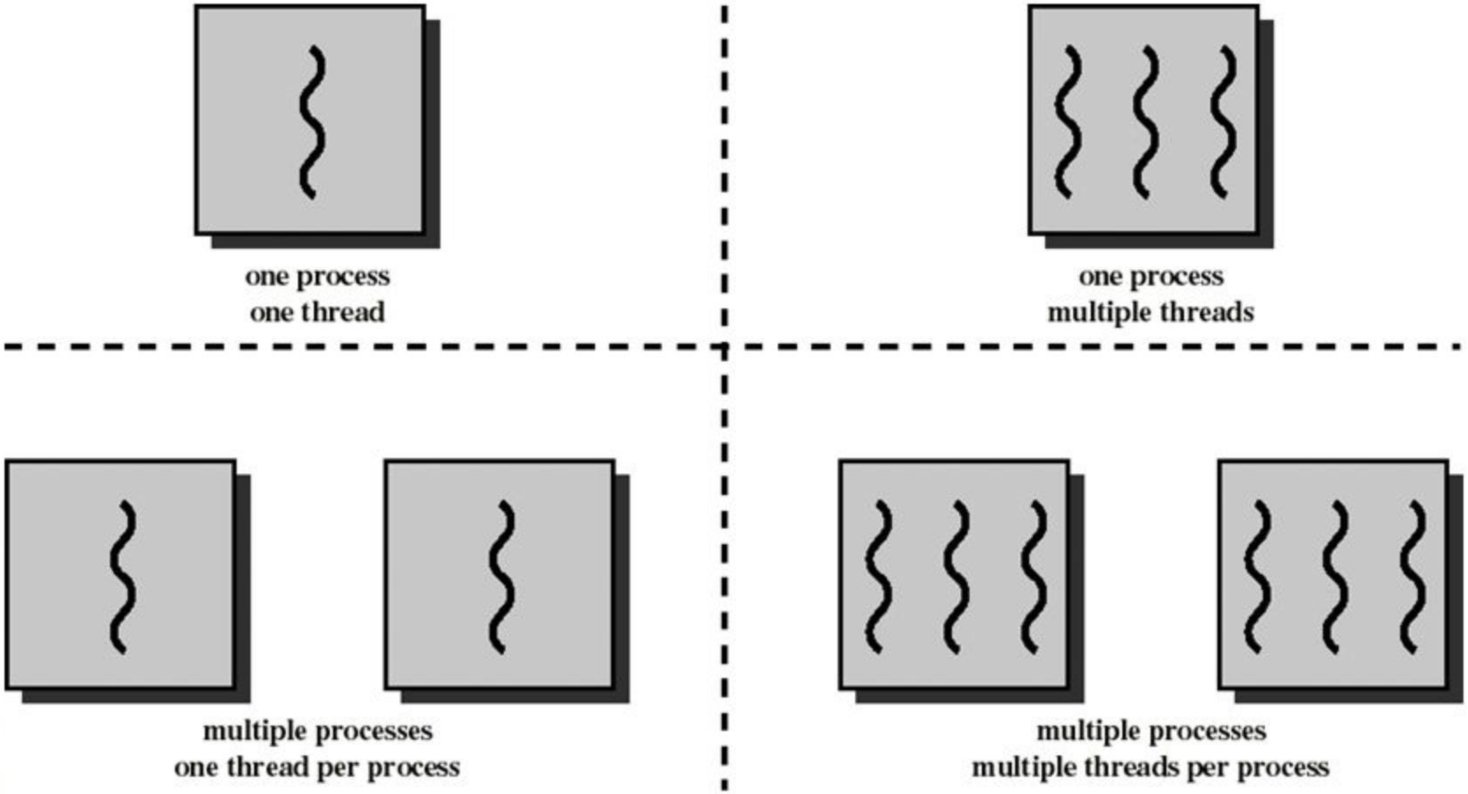
- Definition
  - An instance of a *program* running on a computer
  - An *abstraction* that supports running programs - -> cpu virtualization
  - An *execution stream* in the context of a particular *process state* - -> dynamic unit
  - A *sequential* stream of execution in its *own address space* - -> execution code line by line

What is a thread?

- Thread
  - A finer-granularity entity for execution and parallelism
  - Lightweight process
  - A program in execution without dedicated address space

- Multithreading
  - Running multiple threads within a single process

Finer-granularity entity



Process : thread = 1: N

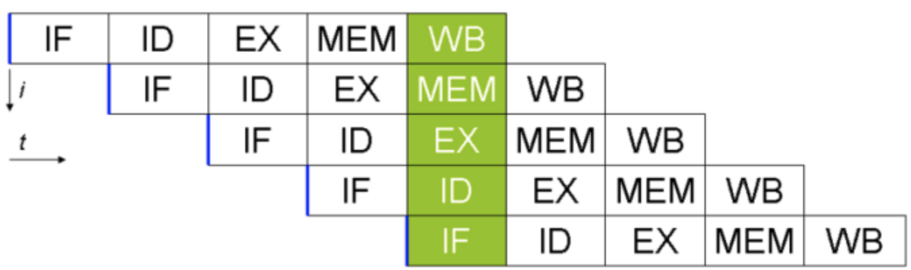
What is a thread?

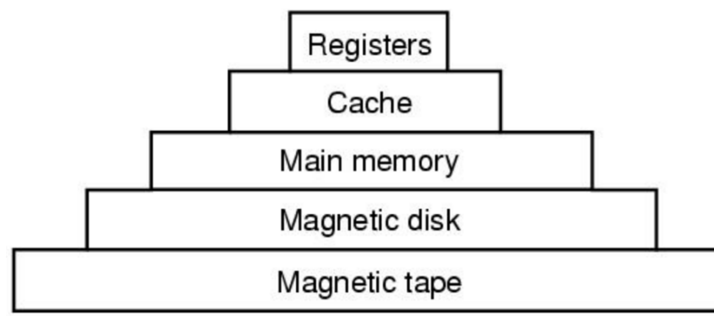
- Thread
  - A finer-granularity entity for execution and parallelism
  - Lightweight process
  - A program in execution without dedicated address space

- Multithreading
  - Running multiple threads within a single process

Process review

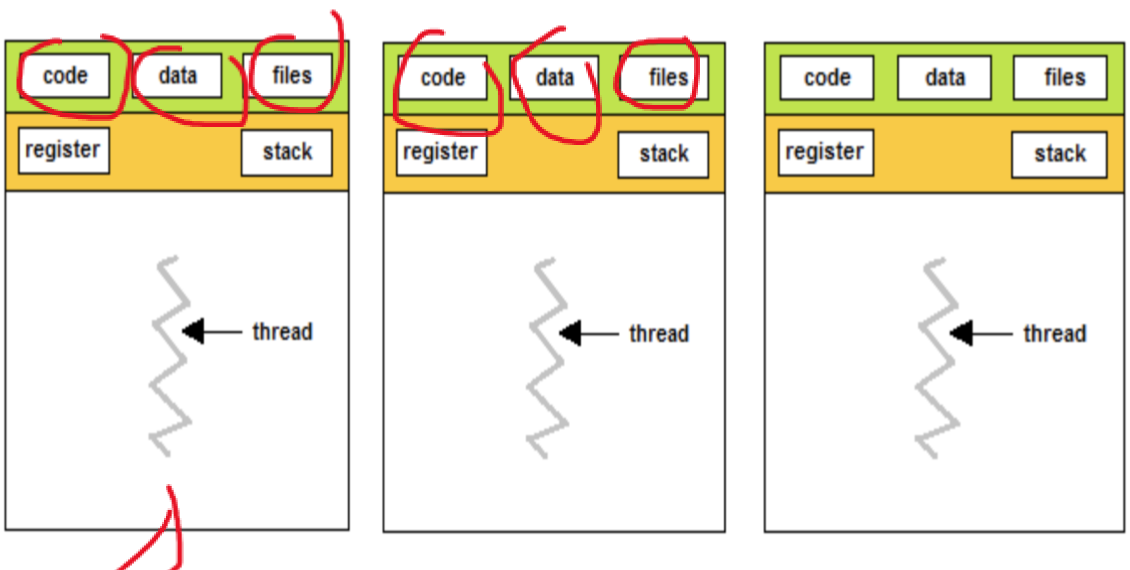
- Two parts of a process
  - Sequential execution of instructions
  - Process state
    - registers: PC (program counter), SP (stack pointer),…
    - Memory: address space, code, data, stack, heap …
    - I/O status: open files …



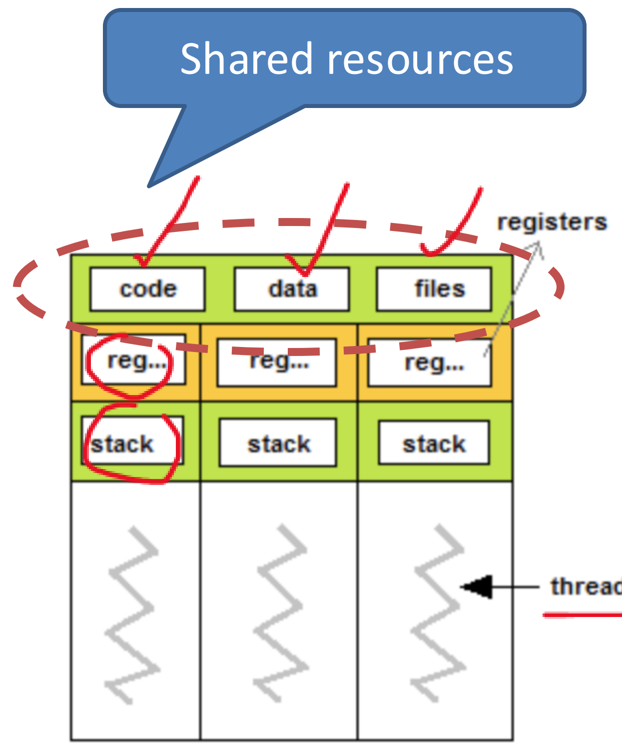


Lightweight process

Occupy more memory,
complex switching (e.g., save old data, switch to new data),
low CPU utilization (e.g., slow context switch)



Three processes



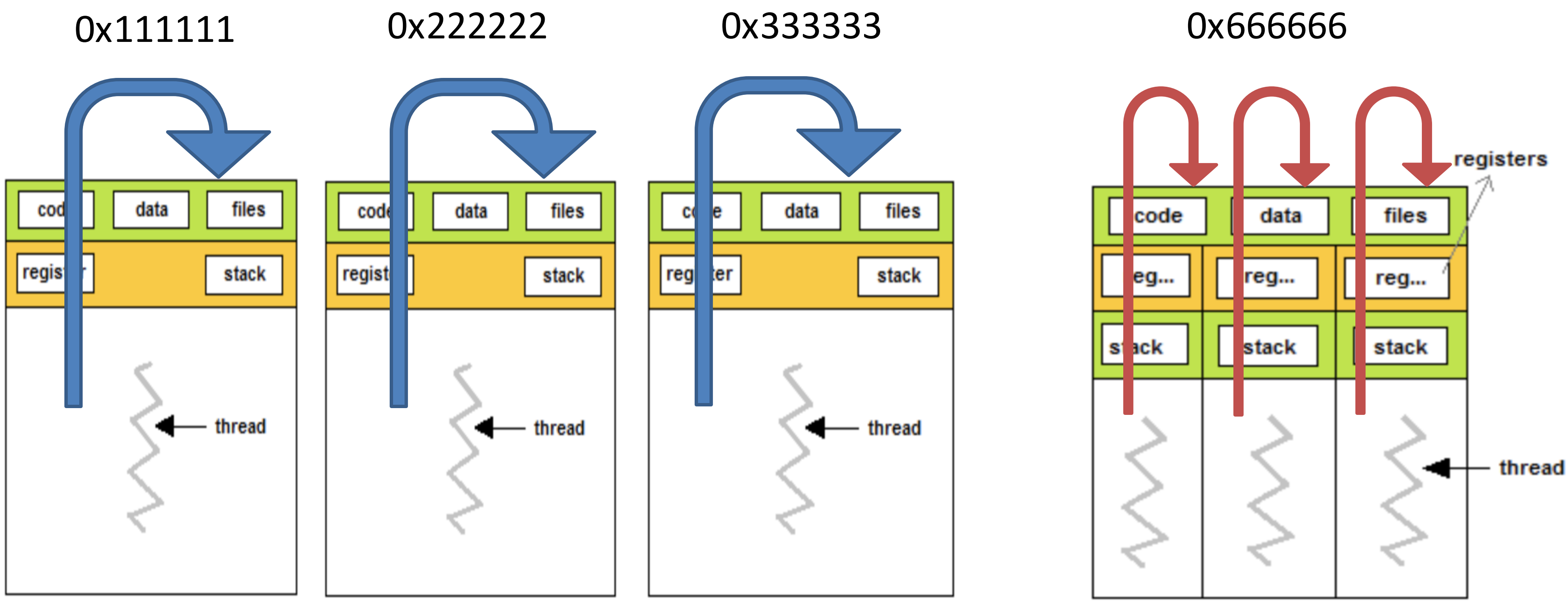
Three threads

What is a thread?

- Thread
  - A finer-granularity entity for execution and parallelism
  - Lightweight process
  - A program in execution without dedicated address space

- Multithreading
  - Running multiple threads within a single process

Dedicated address space



Three processes

Three threads

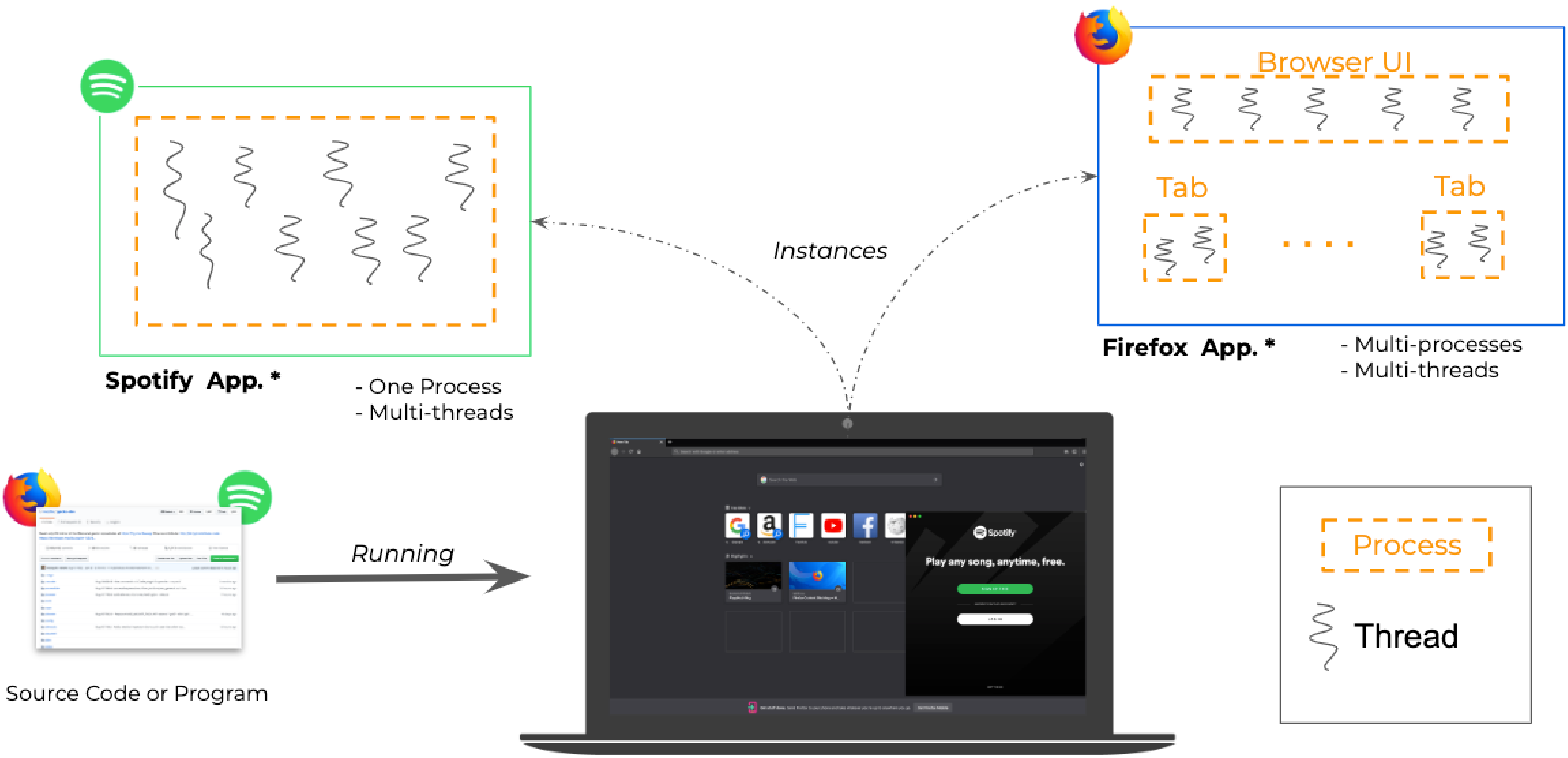
What is a thread?

- Thread
  - A finer-granularity entity for execution and parallelism
  - Lightweight process
  - A program in execution without dedicated address space

- Multithreading
  - Running multiple threads within a single process

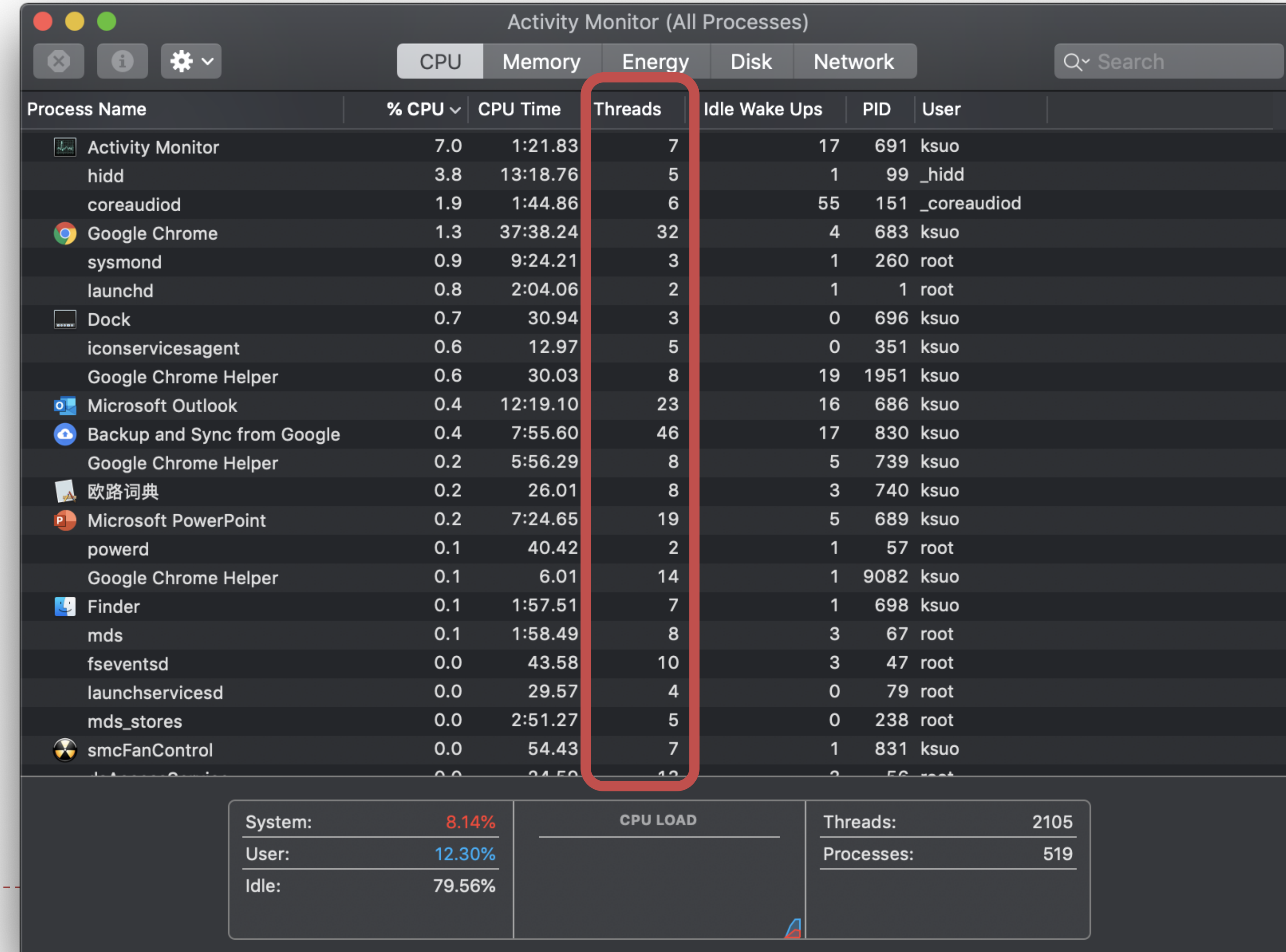
What is a thread? Multithreading apps

Programs, Apps, Processes & Threads



** this image may not reflect the reality for the show-cased apps*

What is a thread? Multithreading apps



Activity Monitor (All Processes)

CPU | Memory | Energy | Disk | Network

Search

Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User
Activity Monitor	7.0	1:21.83	7	17	691	ksuo
hidd	3.8	13:18.76	5	1	99	_hidd
coreaudiod	1.9	1:44.86	6	55	151	_coreaudiod
Google Chrome	1.3	37:38.24	32	4	683	ksuo
sysmond	0.9	9:24.21	3	1	260	root
launchd	0.8	2:04.06	2	1	1	root
Dock	0.7	30.94	3	0	696	ksuo
iconservicesagent	0.6	12.97	5	0	351	ksuo
Google Chrome Helper	0.6	30.03	8	19	1951	ksuo
Microsoft Outlook	0.4	12:19.10	23	16	686	ksuo
Backup and Sync from Google	0.4	7:55.60	46	17	830	ksuo
Google Chrome Helper	0.2	5:56.29	8	5	739	ksuo
欧路词典	0.2	26.01	8	3	740	ksuo
Microsoft PowerPoint	0.2	7:24.65	19	5	689	ksuo
powerd	0.1	40.42	2	1	57	root
Google Chrome Helper	0.1	6.01	14	1	9082	ksuo
Finder	0.1	1:57.51	7	1	698	ksuo
mds	0.1	1:58.49	8	3	67	root
fseventsd	0.0	43.58	10	3	47	root
launchservicesd	0.0	29.57	4	0	79	root
mds_stores	0.0	2:51.27	5	0	238	root
smcFanControl	0.0	54.43	7	1	831	ksuo

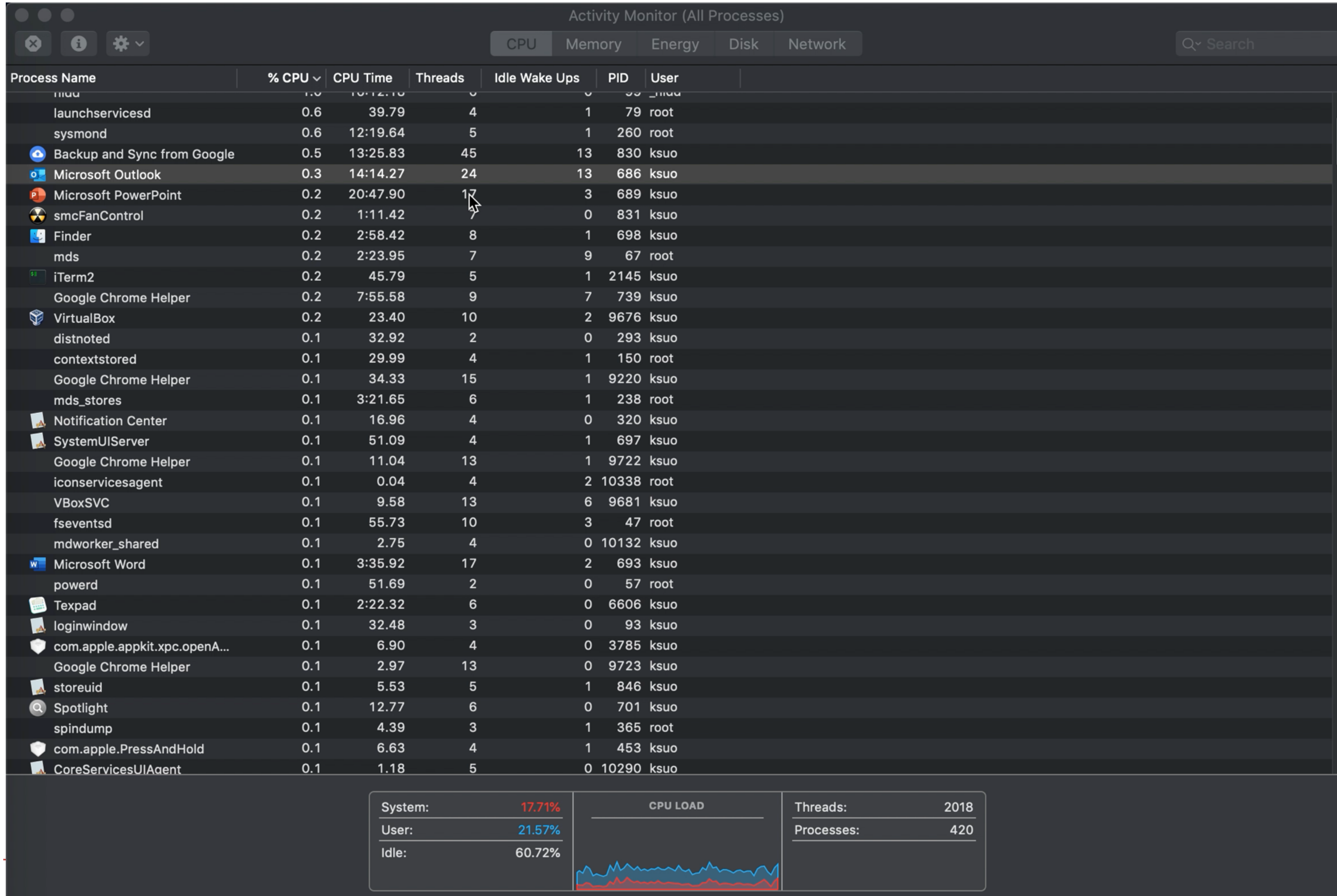
System:	8.14%
User:	12.30%
Idle:	79.56%

CPU LOAD

Threads:	2105
Processes:	519

Multithreading apps

https://youtu.be/Me0mL44TxW0


Activity Monitor (All Processes)

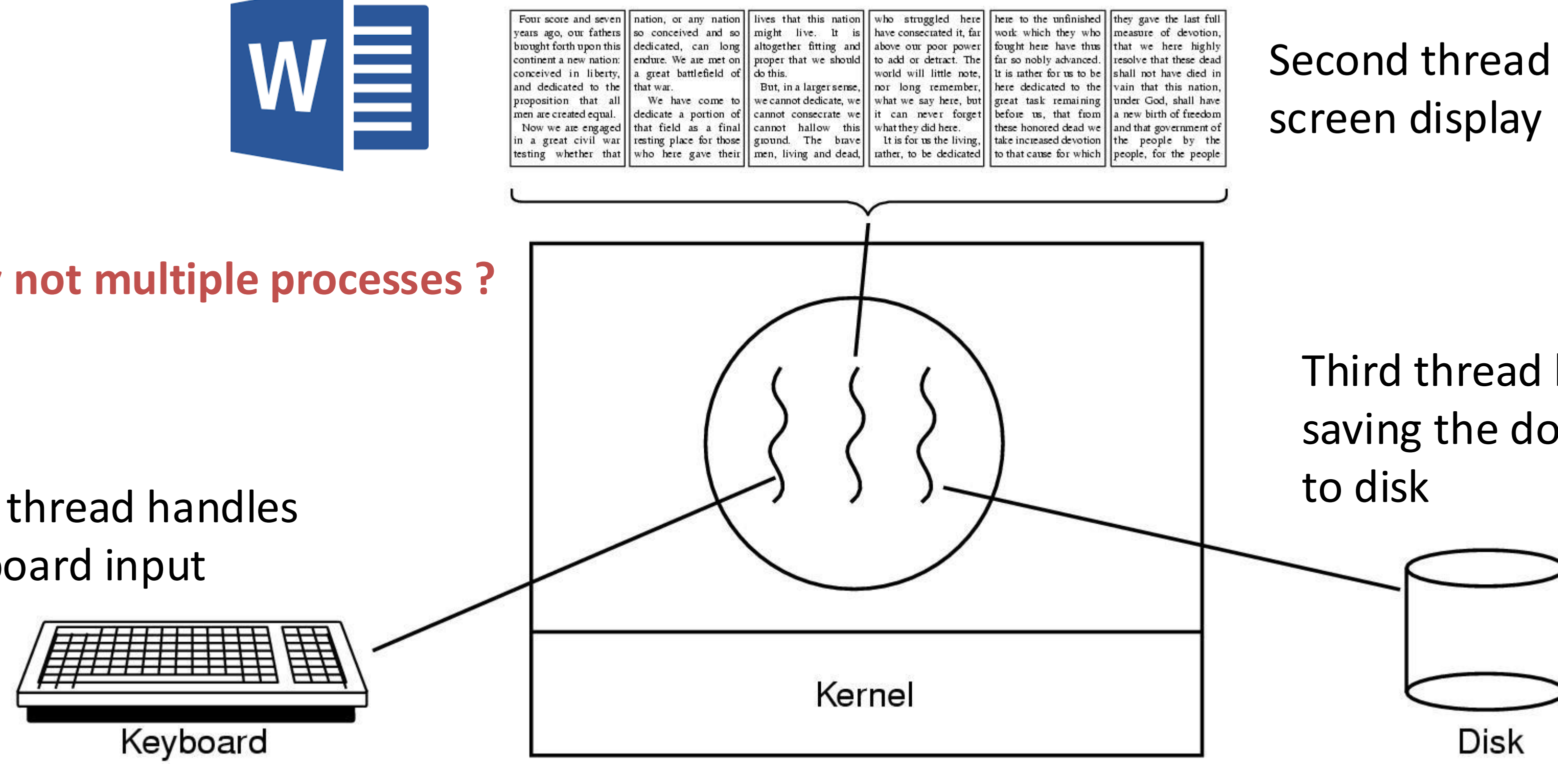
CPU Memory Energy Disk Network

Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User
launchservicesd	0.6	39.79	4	1	79	root
sysmond	0.6	12:19.64	5	1	260	root
Backup and Sync from Google	0.5	13:25.83	45	13	830	ksuo
Microsoft Outlook	0.3	14:14.27	24	13	686	ksuo
Microsoft PowerPoint	0.2	20:47.90	17	3	689	ksuo
smcFanControl	0.2	1:11.42	7	0	831	ksuo
Finder	0.2	2:58.42	8	1	698	ksuo
mds	0.2	2:23.95	7	9	67	root
iTerm2	0.2	45.79	5	1	2145	ksuo
Google Chrome Helper	0.2	7:55.58	9	7	739	ksuo
VirtualBox	0.2	23.40	10	2	9676	ksuo
distnoted	0.1	32.92	2	0	293	ksuo
contextstored	0.1	29.99	4	1	150	root
Google Chrome Helper	0.1	34.33	15	1	9220	ksuo
mds_stores	0.1	3:21.65	6	1	238	root
Notification Center	0.1	16.96	4	0	320	ksuo
SystemUIServer	0.1	51.09	4	1	697	ksuo
Google Chrome Helper	0.1	11.04	13	1	9722	ksuo
iconservicesagent	0.1	0.04	4	2	10338	root
VBoxSVC	0.1	9.58	13	6	9681	ksuo
fseventsd	0.1	55.73	10	3	47	root
mdworker_shared	0.1	2.75	4	0	10132	ksuo
Microsoft Word	0.1	3:35.92	17	2	693	ksuo
powerd	0.1	51.69	2	0	57	root
Texpad	0.1	2:22.32	6	0	6606	ksuo
loginwindow	0.1	32.48	3	0	93	ksuo
com.apple.appkit.xpc.openA...	0.1	6.90	4	0	3785	ksuo
Google Chrome Helper	0.1	2.97	13	0	9723	ksuo
storeuid	0.1	5.53	5	1	846	ksuo
Spotlight	0.1	12.77	6	0	701	ksuo
spindump	0.1	4.39	3	1	365	root
com.apple.PressAndHold	0.1	6.63	4	1	453	ksuo
CoreServicesUIAgent	0.1	1.18	5	0	10290	ksuo

System:	17.71%	CPU LOAD	Threads:	2018
User:	21.57%		Processes:	420
Idle:	60.72%			

Example 1: A word process with three threads

Why not multiple processes ?



Second thread handles screen display

Third thread handles saving the document to disk

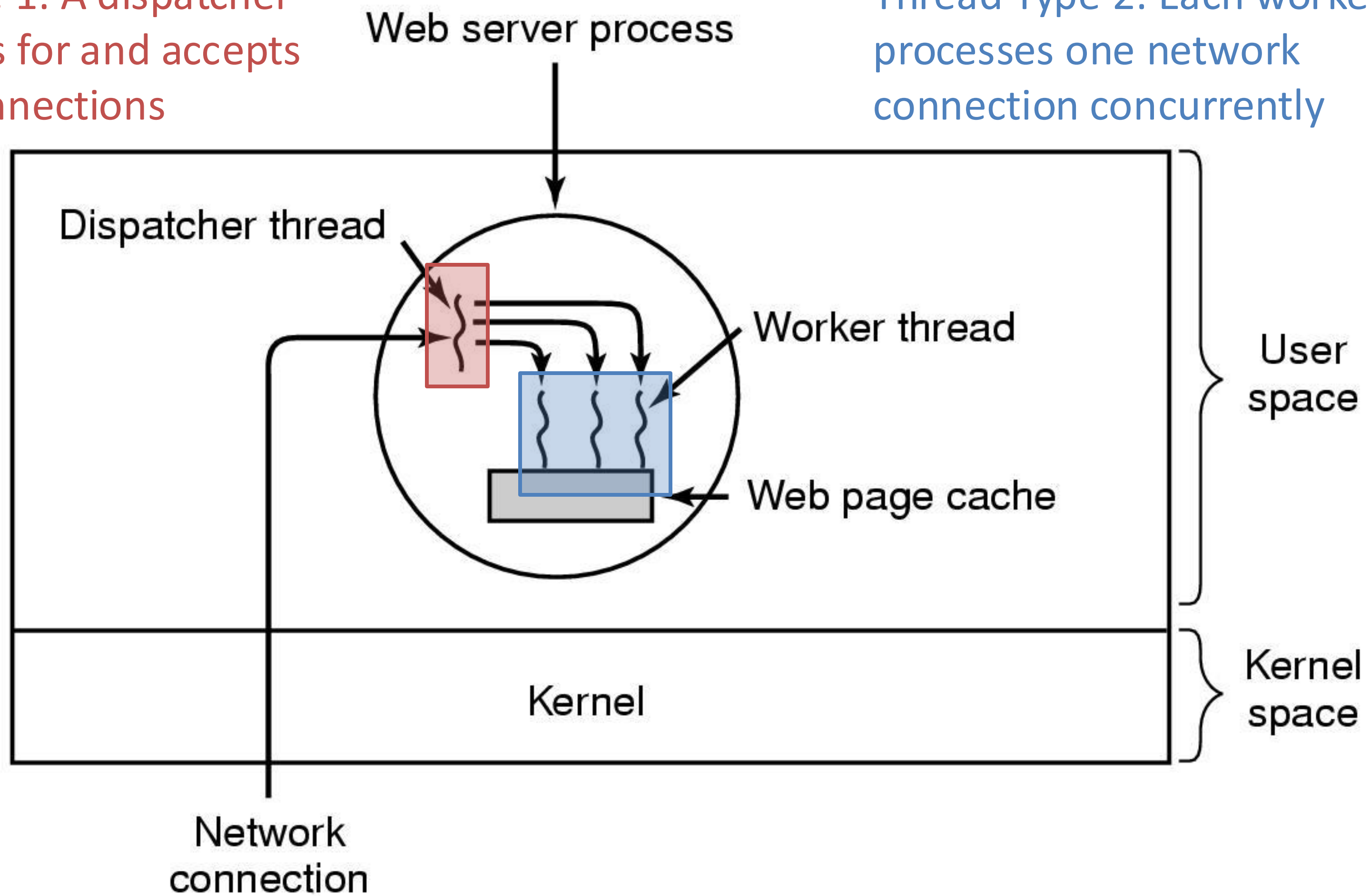
First thread handles keyboard input

A word process with three threads.

Example 2: a multi-threaded web server

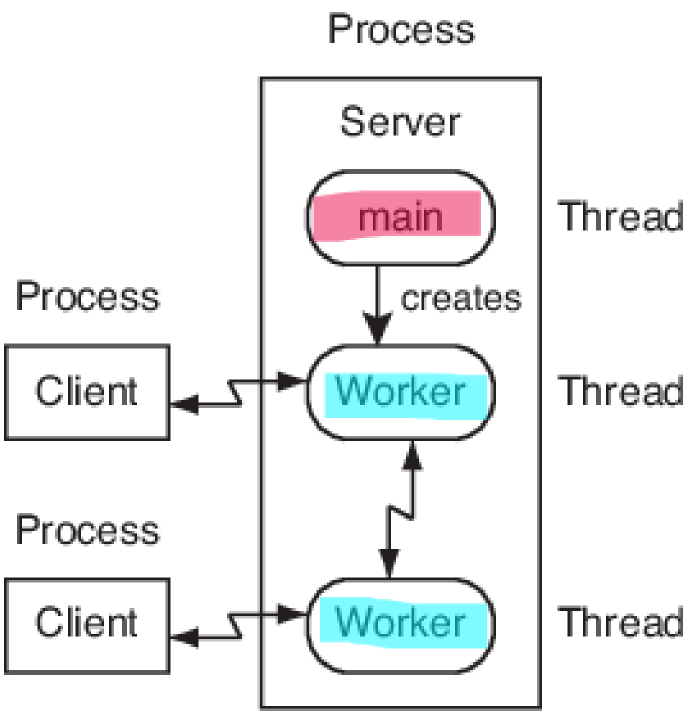
Thread Type 1: A dispatcher thread waits for and accepts network connections

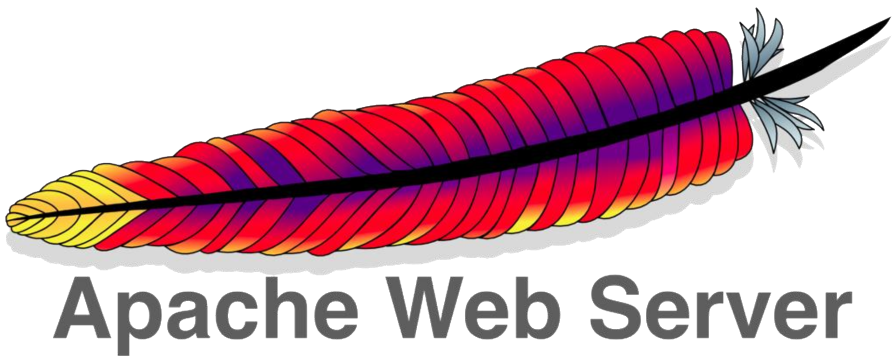
Thread Type 2: Each worker processes one network connection concurrently



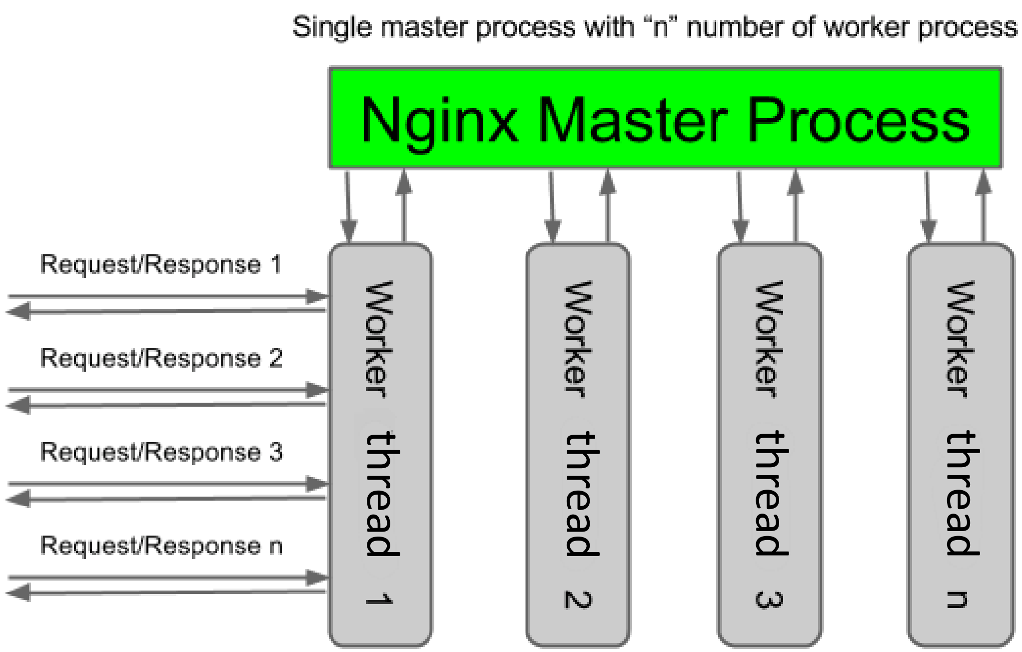
A multithreaded Web server.

Example 2: a multi-threaded web server



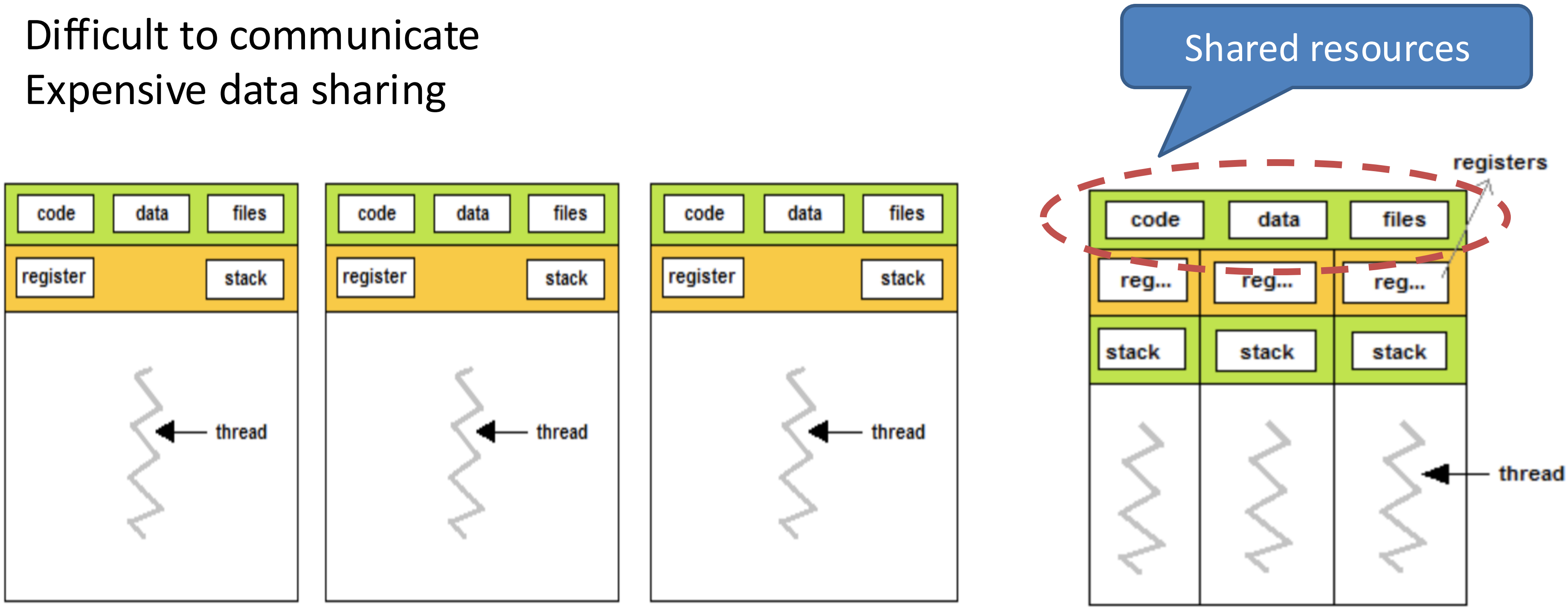


Example 2: a multi-threaded web server



Why not multiple processes ?

Occupy more memory,
complex switching
low CPU utilization
Difficult to communicate
Expensive data sharing



Three processes

Three threads

Why multiple threads

- Good example from Wikipedia: multiple threads within a single process are like multiple cooks trying to prepare the same meal together.

  - Each one is doing one thing.
  - They are probably doing different things.
  - They all share the same recipe but may be looking at different parts of it.
  - They have private state but can communicate easily.
  - They must coordinate!

Outline

- What is thread?
  - Multiple thread application
  - Thread vs Process
  - Advantage and disadvantage of thread
- Thread in Linux
- Thread design
  - Kernel space vs User space
  - Local thread vs Global thread scheduling

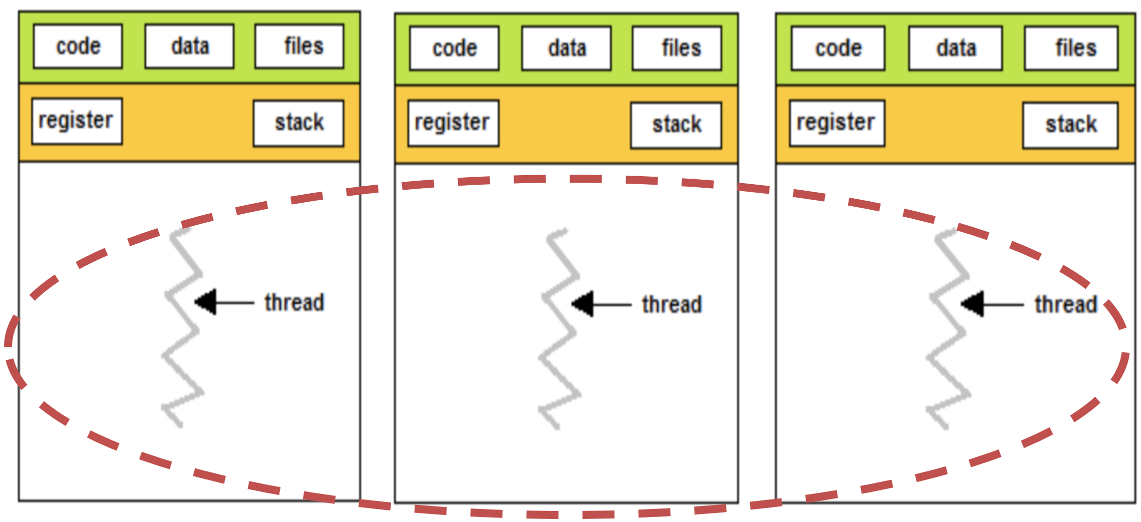
Process vs. Thread comparison

	Process	Thread
Concurrency and protection	?	?
Data structure	?	?
Performance	?	?

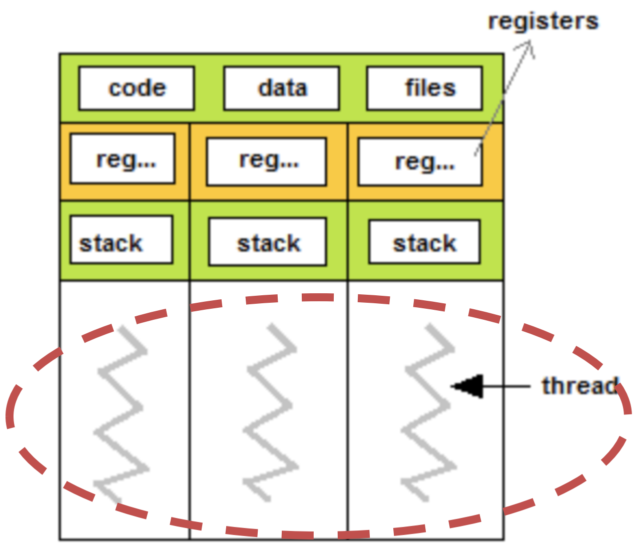
Processes vs. Threads (Concurrency and protection)

3

3



Three processes

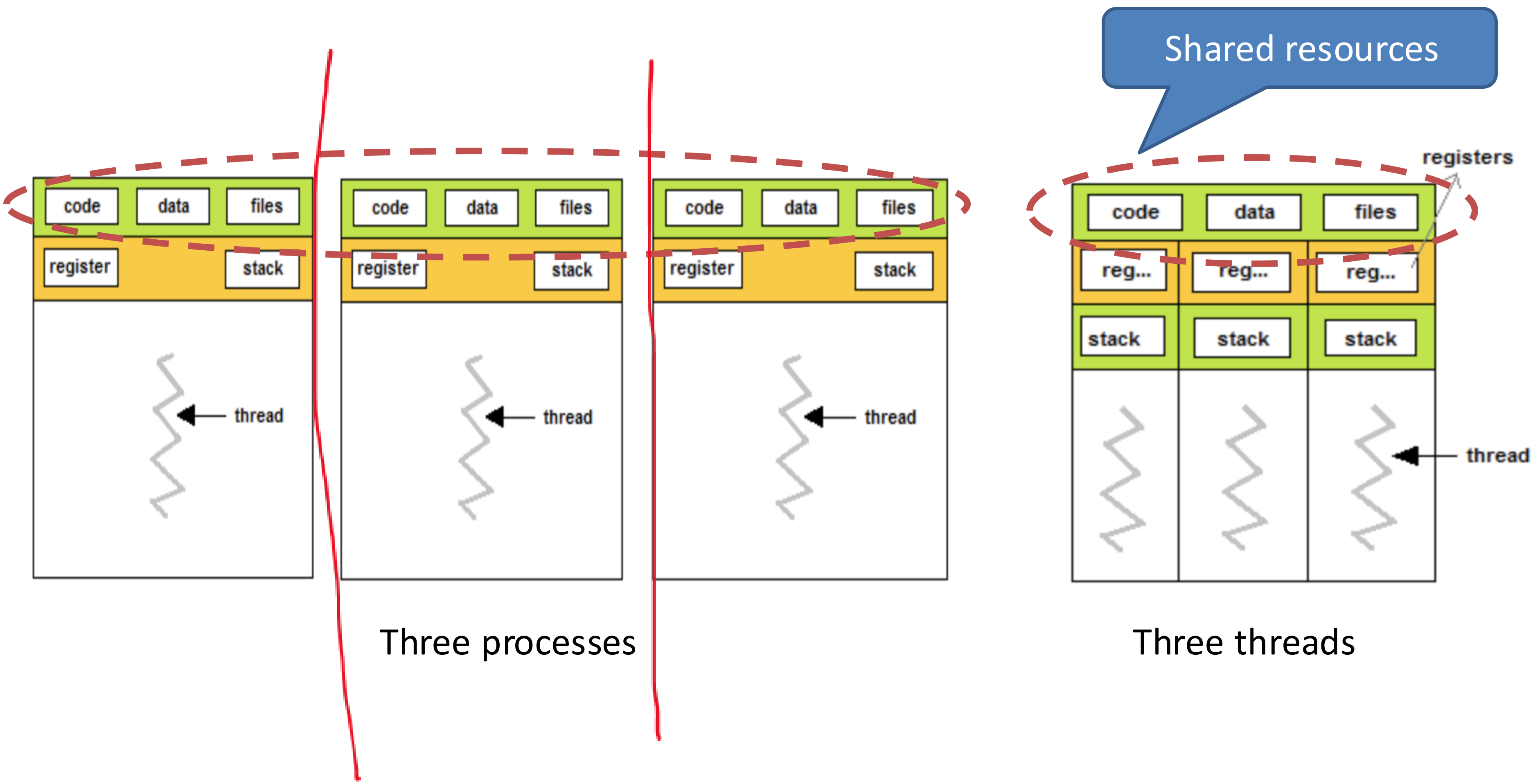


Three threads

Processes vs. Threads (Concurrency and protection)

	Process	Threads
Concurrency	Parallel execution stream of instructions ✓	Maintain parallel execution stream of instructions ✓
Protection		

Processes vs. Threads (Concurrency and protection)



Three processes

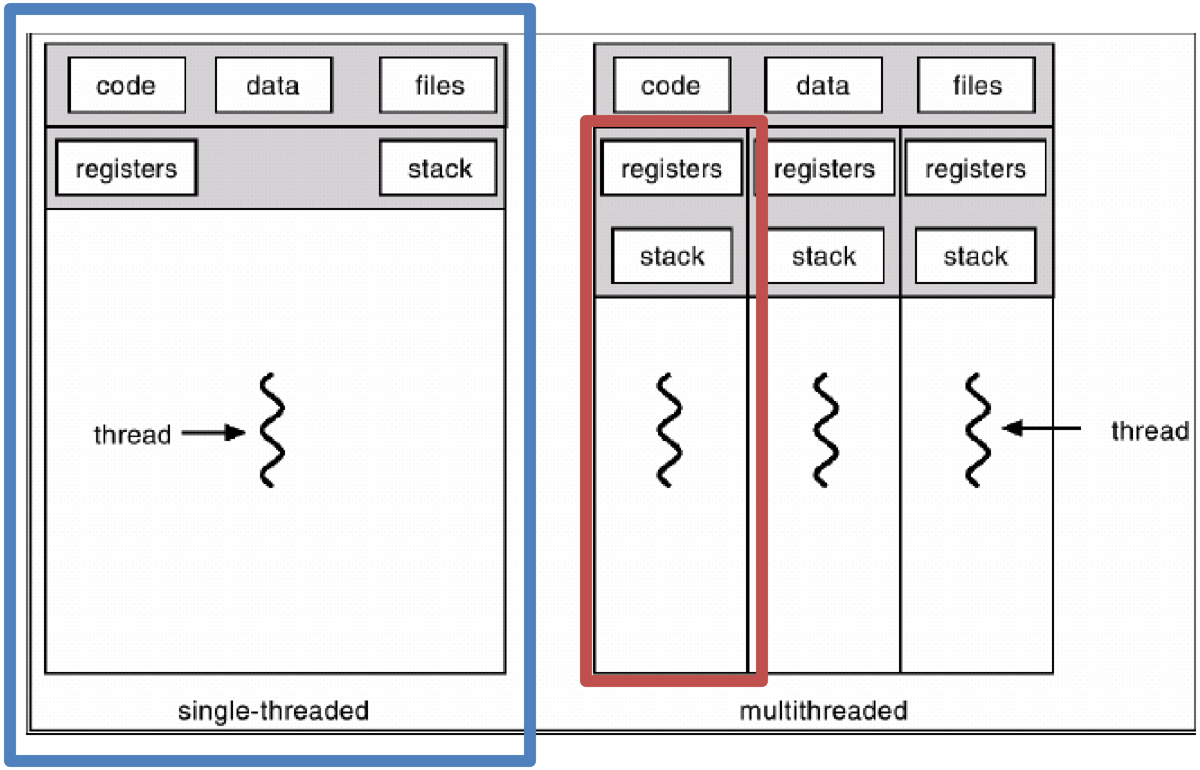
Three threads

Processes vs. Threads (Concurrency and protection)

Separate concurrency from protection

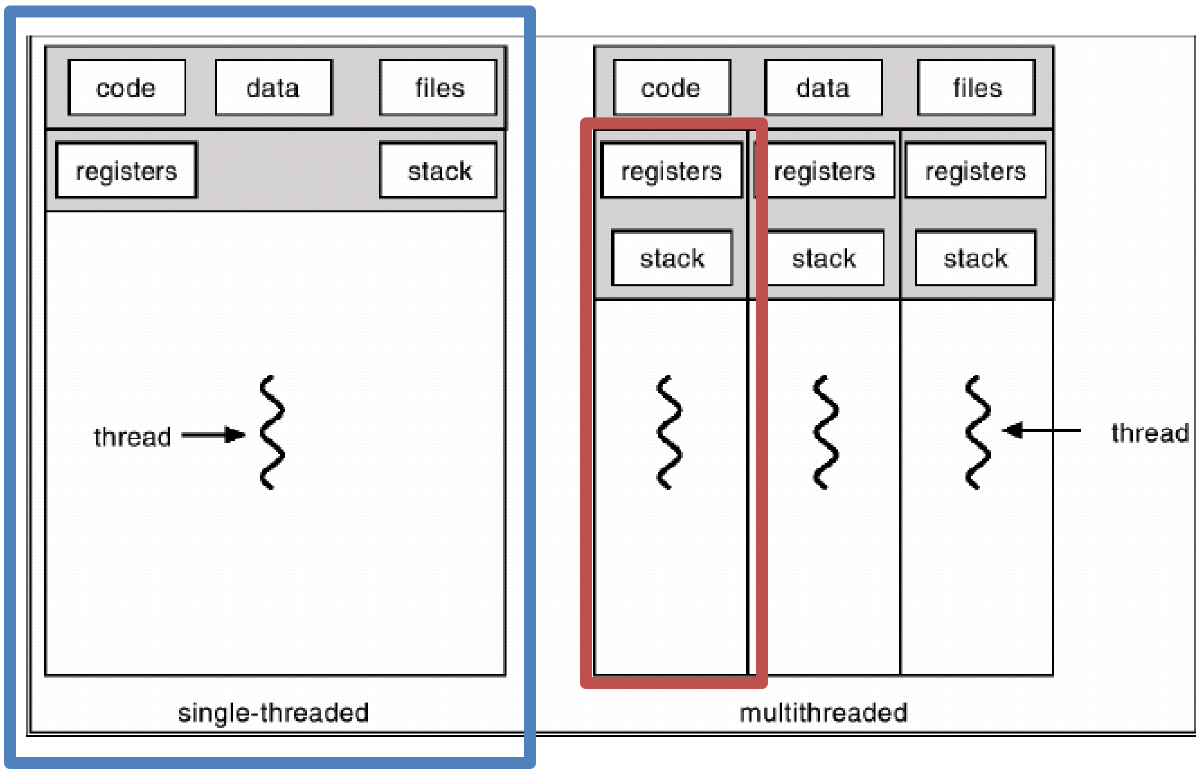
	Process	Threads
Concurrency	Parallel execution stream of instructions ✓	Maintain parallel execution stream of instructions ✓
Protection	A dedicated address space ✓	Share address space with other threads ✗

Processes vs. Threads (Data structure)



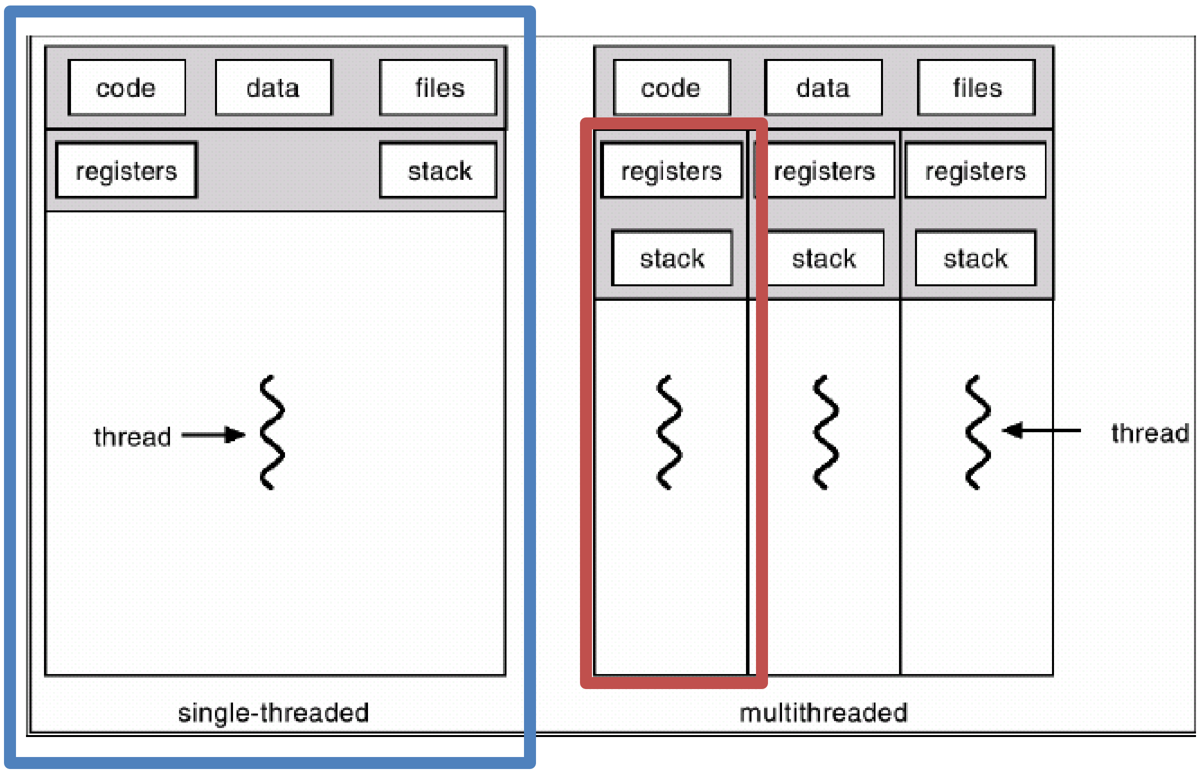
Process	Thread
Have data/code/heap	
Include at lease one thread	
Have own address space, isolated from other processes	

Processes vs. Threads (Data structure)



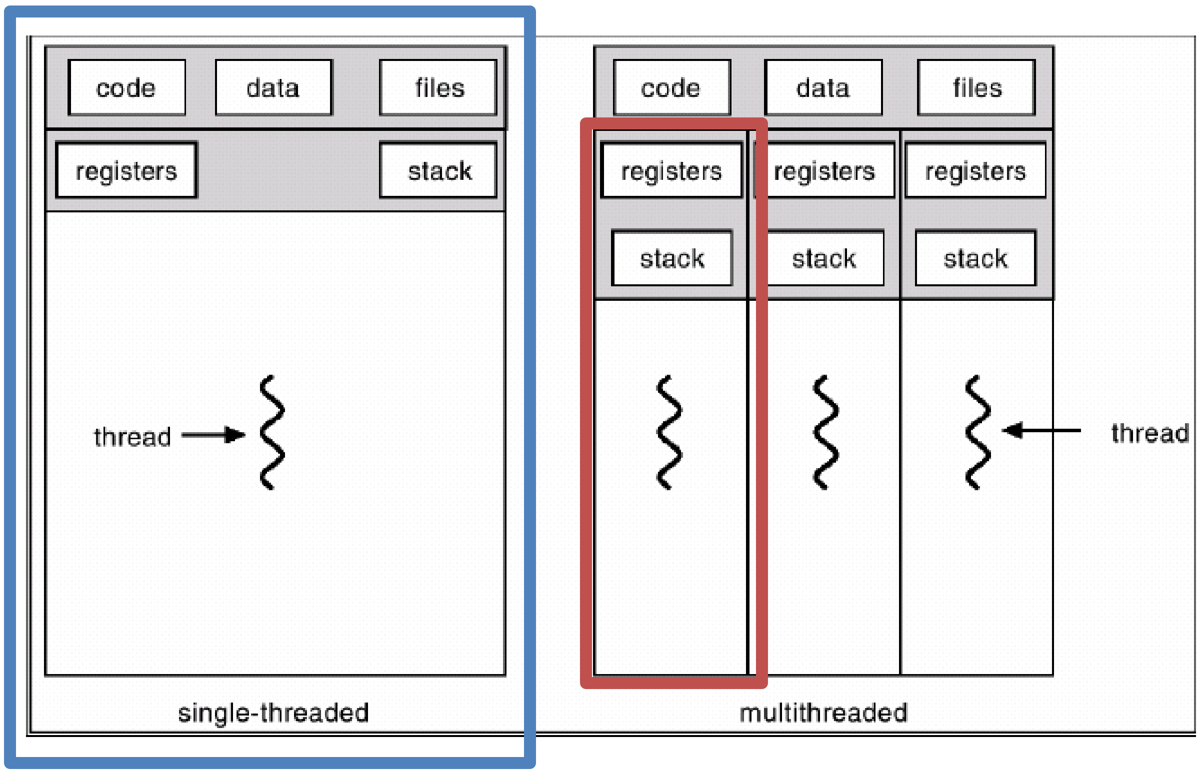
Context	Process	Thread
File pointer	*	
Stack	*	*
Memory	*	
State	*	*
Priority	*	*
I/O state	*	
Authority	*	

Processes vs. Threads (Data structure)



Context	Process	Thread
Scheduling	*	
Statistics	*	
File description	*	
Read/Write pointer	*	
Event/ Signal	*	
Registers	*	*

Processes vs. Threads (Performance)

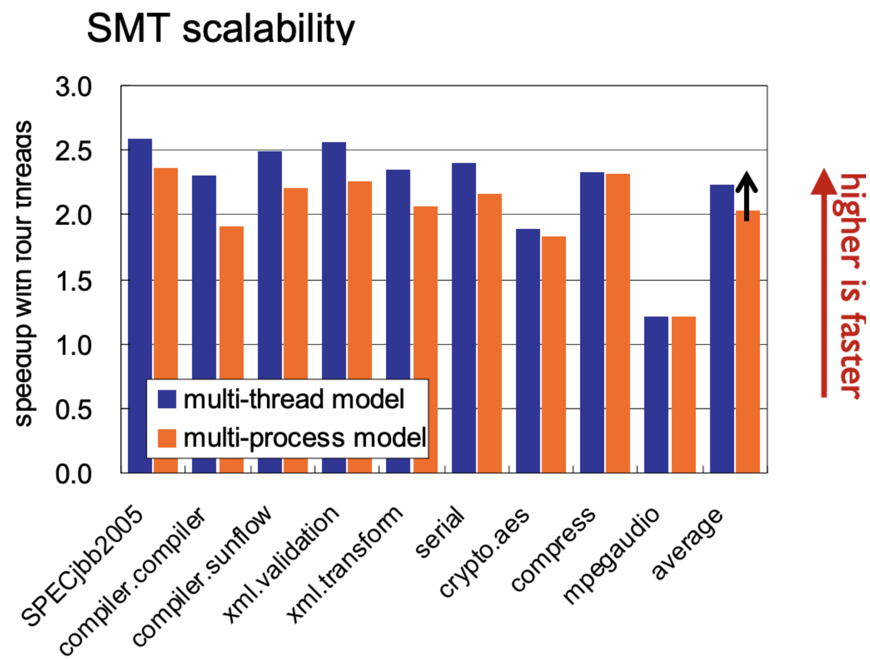


Process	Thread
Expensive to create	Inexpensive to create
Expensive context switching	Inexpensive context switching
IPC can be expensive	Efficient communication

Paper: Performance of Multi-Process and Multi-Thread Processing on Multi-core SMT Processors

- Our results showed that both models (multi-process vs. multi-thread) achieved almost comparable performance, whereas the multi-thread model achieved much better SMT scalability and higher performance.

https://pdfs.semanticscholar.org/dd54/a215131c5eacd997708c5c4612345fe989c7.pdf



multi-thread model was 9.6% faster on average

Demo

- https://github.com/tsuna/contextswitch
  - timectxsw: Benchmarks the overhead of context switching between 2 processes.
  - timetctxsw: Benchmarks the overhead of context switching between 2 threads.

$ gcc xxx.c -o xxx.o -pthread
$ ./xxx.o

```
fish /home/NewUsersDir/ksuo/ksuo/contextswitch
ksuo@ccsegpu4 ~/contextswitch (master)> ./timectxsw.o
2000000 process context switches in 5001406293ns (2500.7ns/ctxsw)
ksuo@ccsegpu4 ~/contextswitch (master)> ./timetctxsw.o
2000000  thread context switches in 3958269002ns (1979.1ns/ctxsw)
ksuo@ccsegpu4 ~/contextswitch (master)>
```

Outline

- What is thread?
  - Multiple thread application
  - Thread vs Process
  - Advantage and disadvantage of thread
- Thread in Linux
- Thread design
  - Kernel space vs User space
  - Local thread vs Global thread scheduling

Advantages of Threads

- Efficient creation
  - Only create the thread context
- Express concurrency
  - Lightweight, better performance, higher scalability
- Efficient communication
  - Communication can be carried out via shared data objects within the shared address space

Disadvantages of Threads

- Shared data - -> Security
  - Global variables are shared between threads.
  - Accidental data changes can cause errors.
- Lack of robustness
  - Crash in one thread will crash the entire process.
- Some library functions may not be thread-safe
  - Library Functions that return pointers to static internal memory. E.g. gethostbyname()

Test

- The following statements about threads, which ones are correct (ABCDE).
  - A. The thread is introduced to improve the execution efficiency of the system and reduce the idle time of the processor and the scheduling switching time
  - B. Thread is the basic unit independently scheduled by the system
  - C. The thread itself basically does not own system resources, but it can share all the resources owned by the process with other threads belonging to the same process
  - D. Thread is also called lightweight process
  - E. Multiple threads in the same process can execute concurrently

Test

- A process can contain multiple threads, each thread (A).
  - A. Shares process virtual address space
  - B. The address space of each thread is completely independent
  - C. is the unit of resource allocation
  - D. Shares stack

Test

- In the following description, (D) is not a feature of multi-threaded systems.
  - A. Use threads to perform matrix multiplication operations in parallel
  - B. Web server uses thread to request http service
  - C. GUI-based debugger uses different threads to process user input, calculation, tracking and other operations
  - D. The keyboard driver creates a thread for each running process in response to the corresponding keyboard input

Test

- The following description of the comparison between processes and threads (D) is wrong:
  - A) The time overhead of concurrent execution of threads is less than that of processes
  - B) Inter-thread communication is simpler than process
  - C) Threads have fewer resources than processes
  - D) Thread is more stable than process

Outline

- What is thread?
  - Multiple thread application
  - Thread vs Process
  - Advantage and disadvantage of thread
- Thread in Linux
- Thread design
  - Kernel space vs User space
  - Local thread vs Global thread scheduling

Processes vs. Threads in Linux

- On Mon, 5 Aug 1996, Peter P. Eiserloh wrote:
  We need to keep a clear the concept of threads. Too many people seem to confuse a thread with a process. The following discussion does not reflect the current state of linux, but rather is an attempt to stay at a high level discussion.

- http://lkml.iu.edu/hypermail/linux/kernel/9608/0191.html
- The way Linux thinks about this (and the way I want things to work) is that there _is_ no such thing as a “process” or a “thread”. There is only the totality of the COE (called "task" by Linux).

Thread creation: clone(), not fork()

- Create new threads in Linux
  - Use `clone()` to create threads instead of using fork()
  - `clone()` is usually not called directly but from some threading libraries, such as pthread.
  - http://man7.org/linux/man-pages/man2/clone.2.html

```
int main(int argc, char *argv[])
{
        char *stack;                    /* Start of stack buffer */
        char *stackTop;                 /* End of stack buffer */
        pid_t pid;
        struct utsname uts;

        if (argc < 2) {
                fprintf(stderr, "Usage: %s <child-hostname>\n", argv[0]);
                exit(EXIT_SUCCESS);
        }

        /* Allocate stack for child */

        stack = malloc(STACK_SIZE);
        if (stack == NULL)
                errExit("malloc");
        stackTop = stack + STACK_SIZE;  /* Assume stack grows downward */

        /* Create child that has its own UTS namespace;
           child commences execution in childFunc() */

        pid = clone(childFunc, stackTop, CLONE_NEWUTS | SIGCHLD, argv[1]);
        if (pid == -1)
                errExit("clone");
        printf("clone() returned %ld\n", (long) pid);

        /* Parent falls through to here */
```

Clone system call

- No. 56, https://filippo.io/linux-syscall-table/

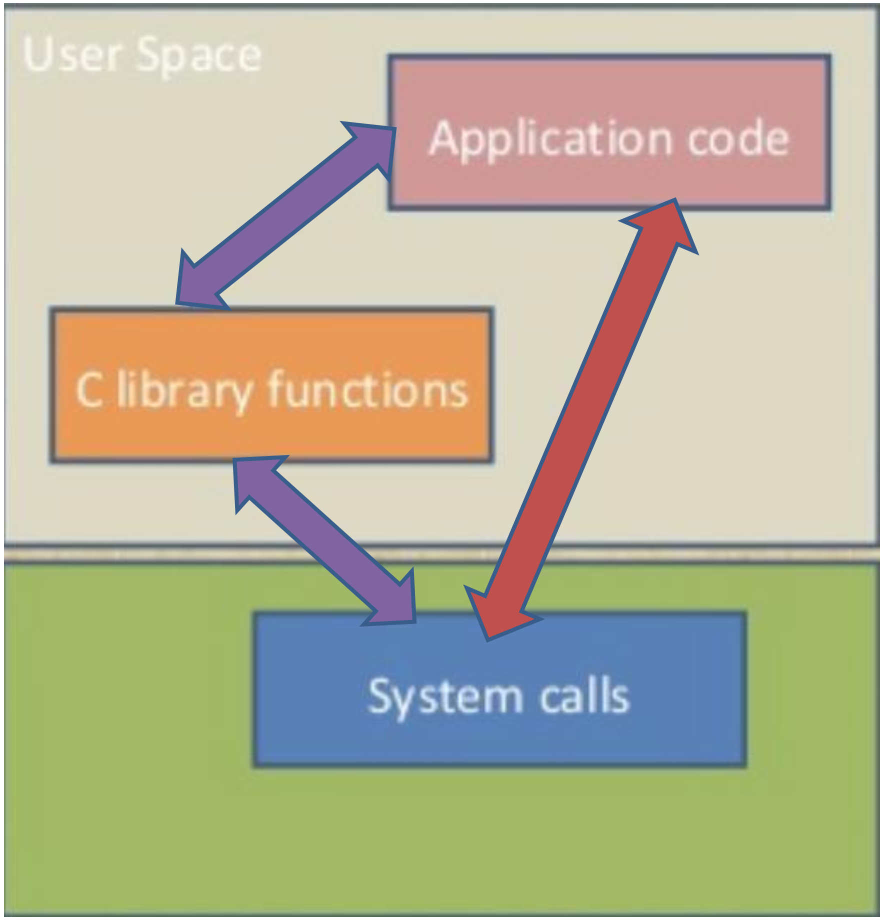
filippo.io

51	getsockname	sys_getsockname	net/socket.c
52	getpeername	sys_getpeername	net/socket.c
53	socketpair	sys_socketpair	net/socket.c
54	setsockopt	sys_setsockopt	net/socket.c
55	getsockopt	sys_getsockopt	net/socket.c
56	clone	stub_clone	kernel/fork.c
57	fork	stub_fork	kernel/fork.c
58	vfork	stub_vfork	kernel/fork.c
59	execve	stub_execve	fs/exec.c
60	exit	sys_exit	kernel/exit.c
61	wait4	sys_wait4	kernel/exit.c
62	kill	sys_kill	kernel/signal.c

Clone system call

Pthread_create()
in the user space

Clone()
in the kernel



Pthread Creation Example

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void *myThread1(void)
{
        int i;
        for (i=0; i<3; i++)
        {
                printf("This is the 1st pthread.\n");
                sleep(1);
        }
}

int main()
{
        int ret=0;
        pthread_t id1;

        printf("This is main thread!\n");

        ret = pthread_create(&id1, NULL, (void*)myThread1, NULL);
        if (ret)
        {
                printf("Create pthread error!/n");
                return 1;
        }

        pthread_exit(NULL);

        return 0;
}
```

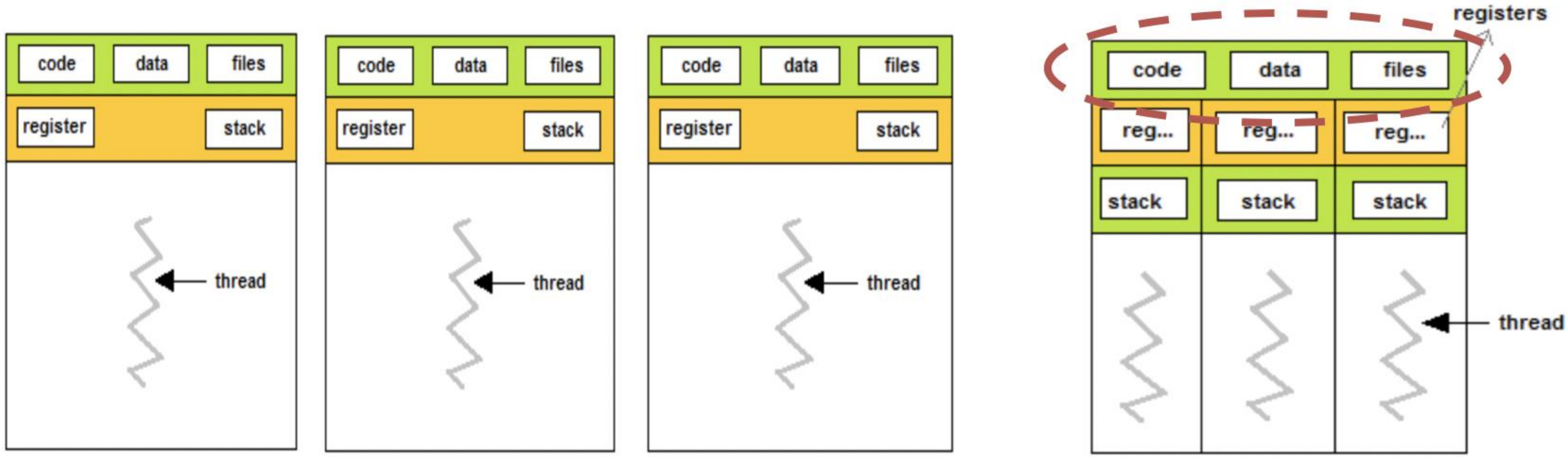
https://github.com/kevinsuo/CS7172/blob/master/pthread_create.c

```
pi@raspberrypi ~/Downloads> ./test.o
This is main thread!
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
```

Clone() in the kernel

Fork vs clone

- Fork:
  - **All resources** in PCB are **copied** from parent process to child process
- Clone:
  - The resources in PCB are **partly copied** from one thread to another thread (the copied context is different in vfork)

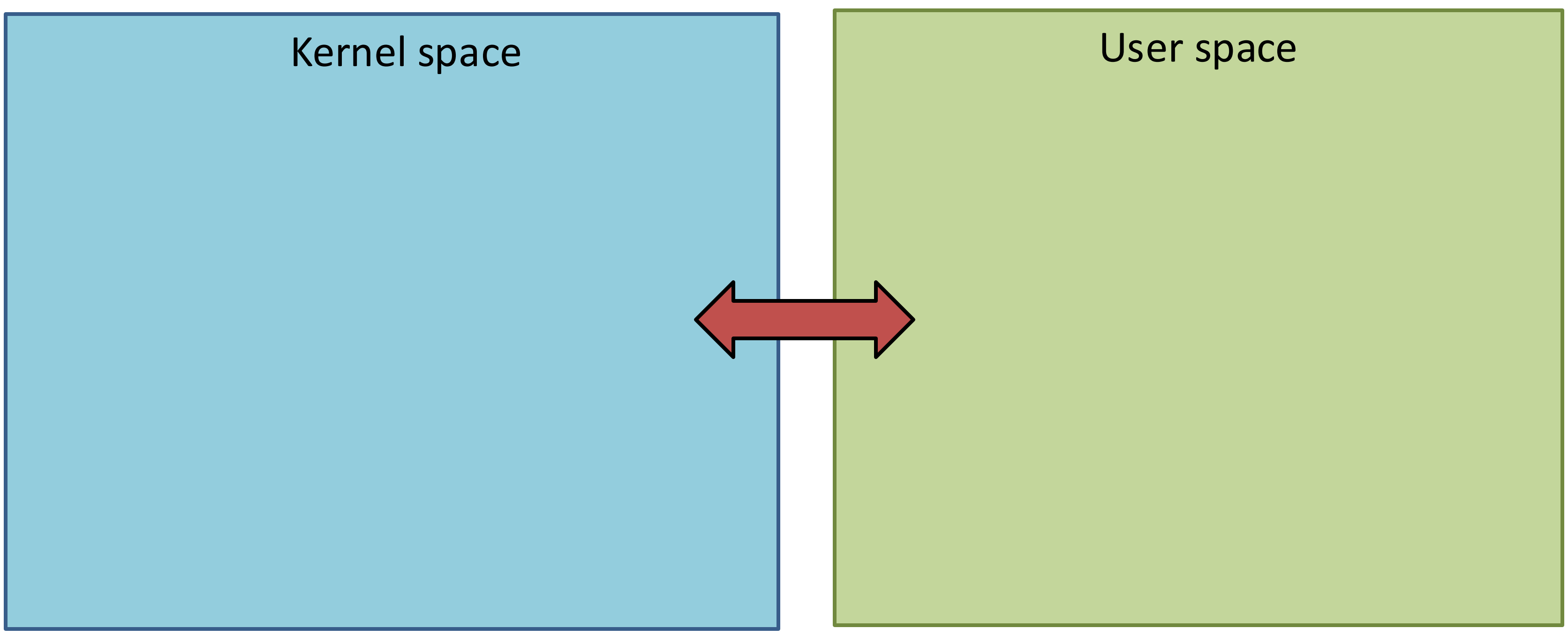


Outline

- What is thread?
  - Multiple thread application
  - Thread vs Process
  - Advantage and disadvantage of thread
- Thread in Linux
- Thread design
  - Kernel space vs User space
  - Local thread vs Global thread scheduling

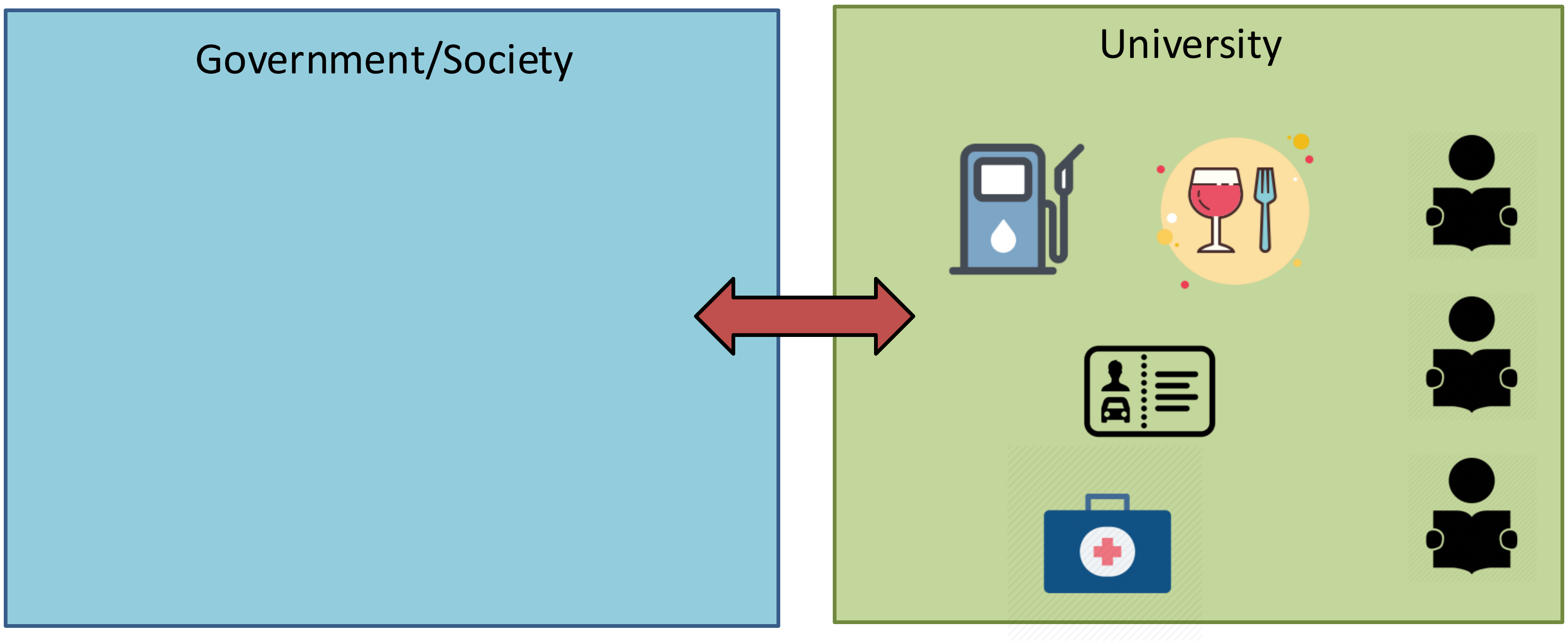
Thread design

- When you have multiple threads, where to put them?



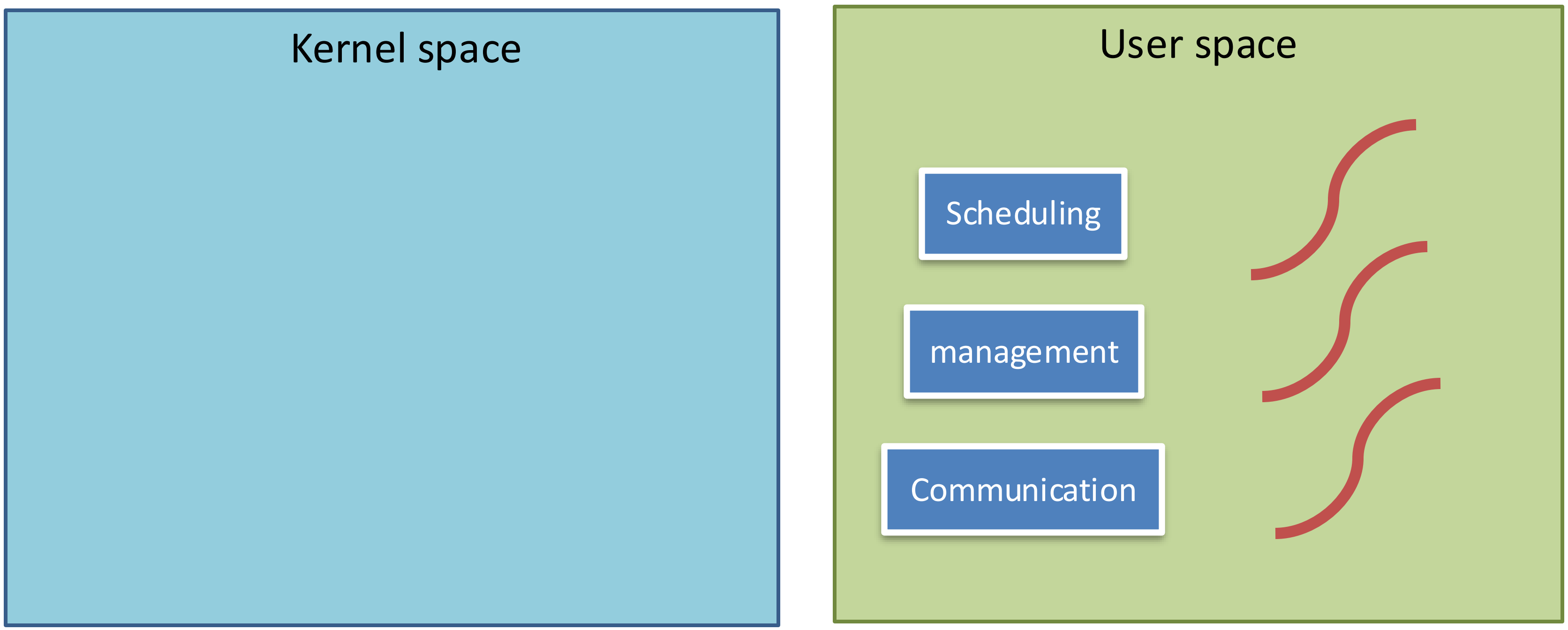
A comparison

- Services in or out of campus



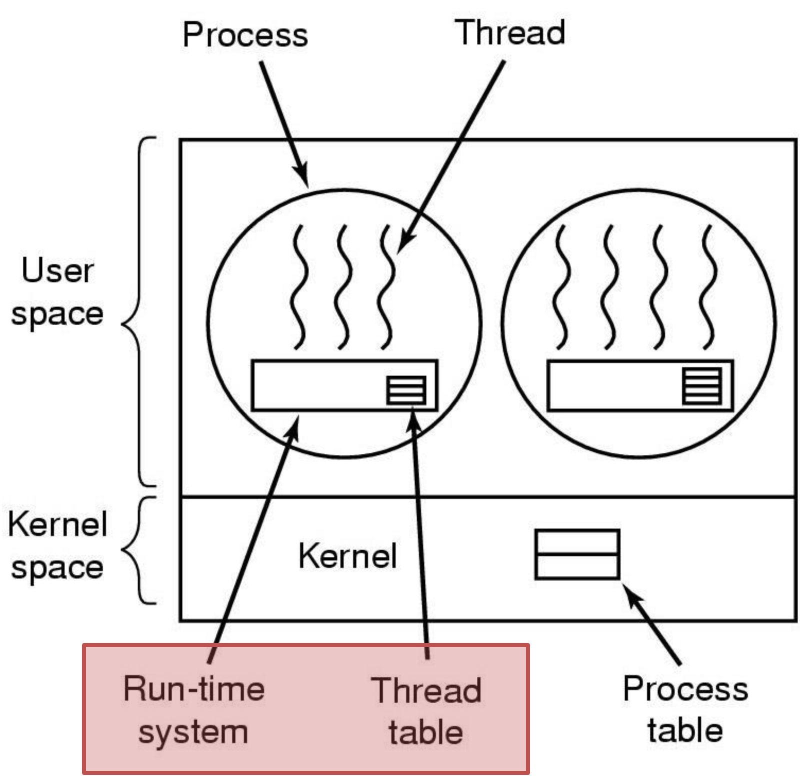
User-level Thread

- When you have multiple threads, where to put them?



User-level Thread

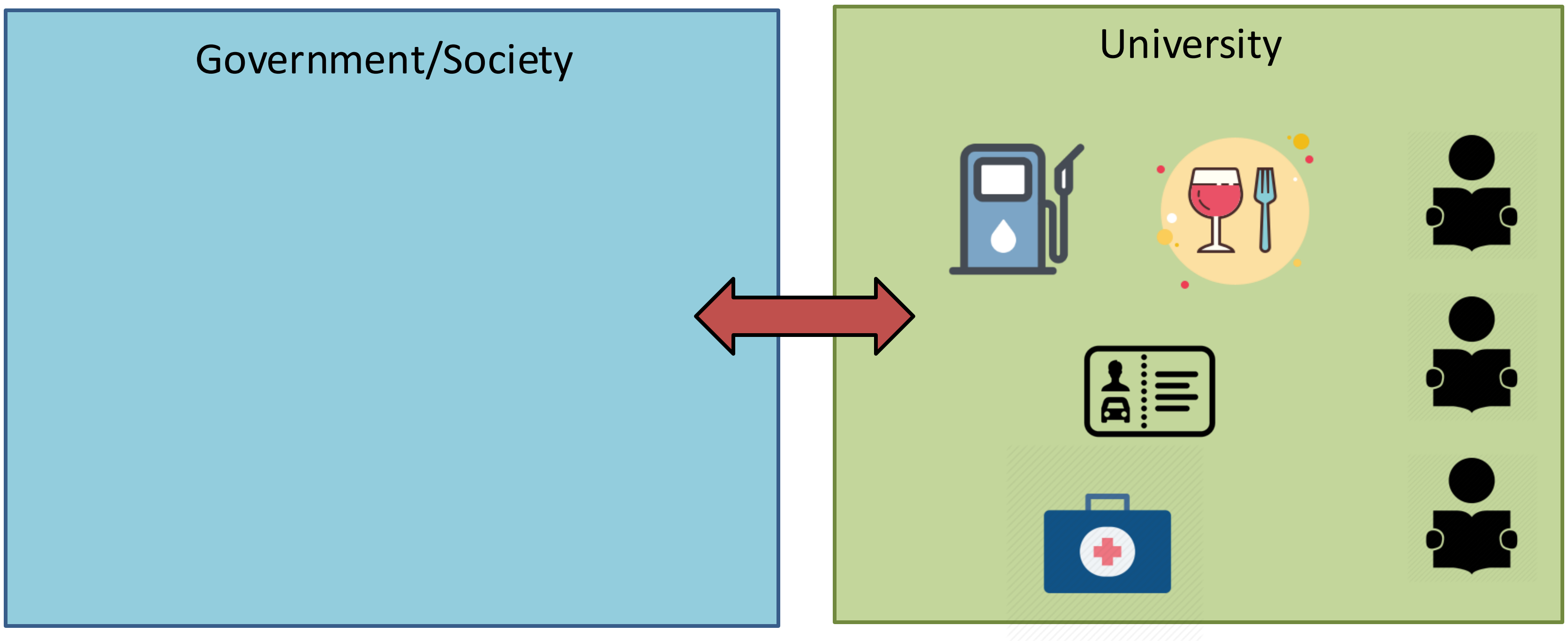
- User-level threads: the kernel knows nothing about them and managed by applications



A user-level threads package

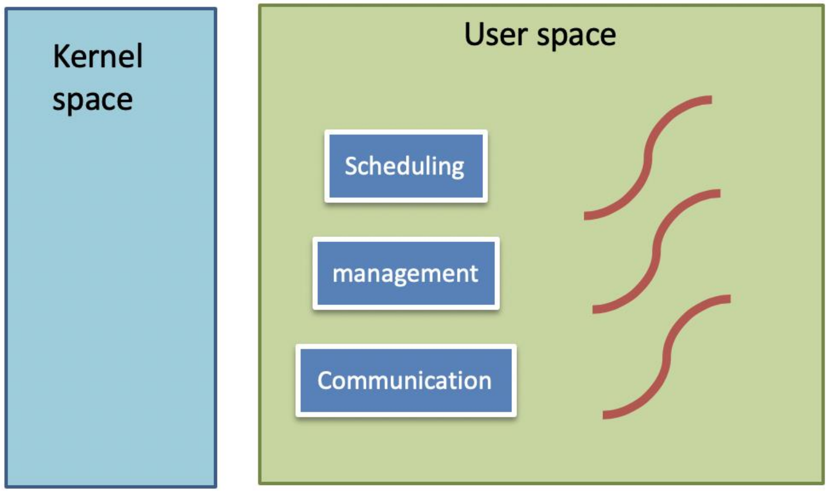
Discussion

- Advantages?



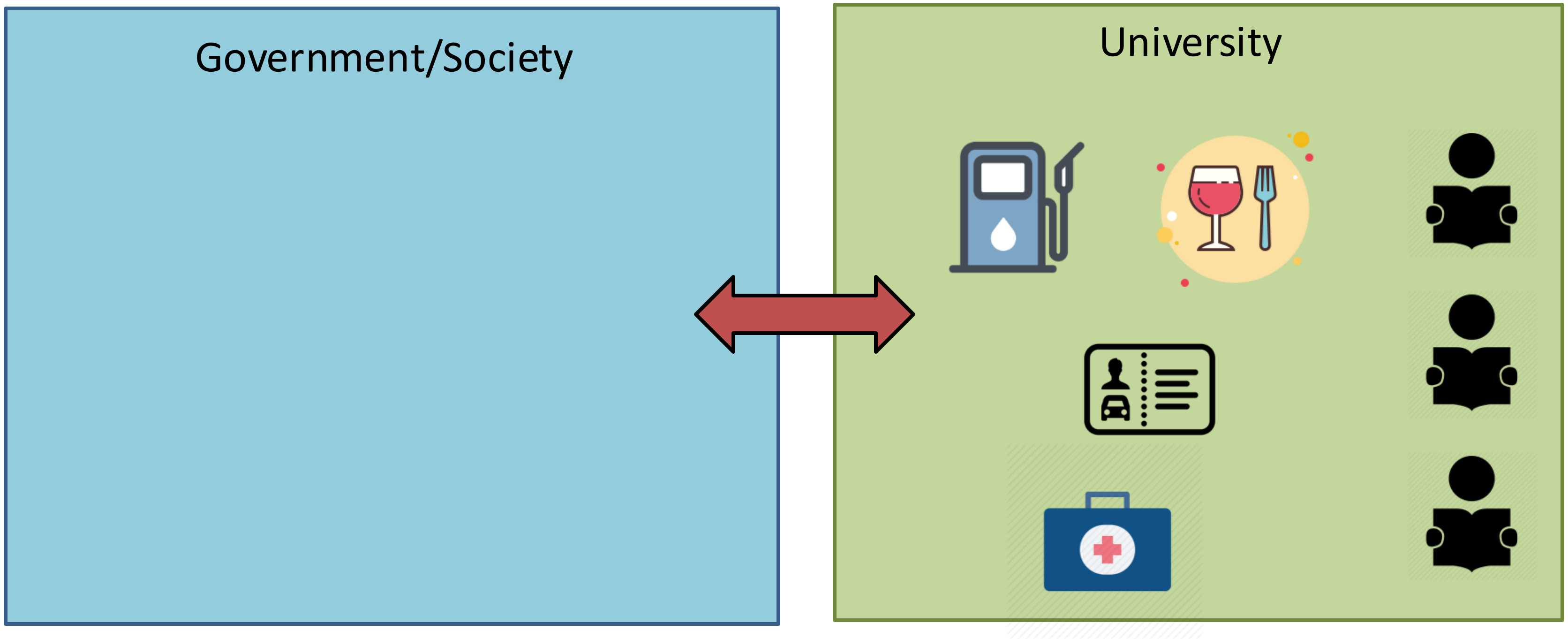
User-level Thread - Discussions

- Advantages
  - No OS thread-support need
  - Lightweight: thread switching vs. process switching
    - Less memory
    - Faster context switch
    - Easier to communicate and data sharing
    - Management in the user space



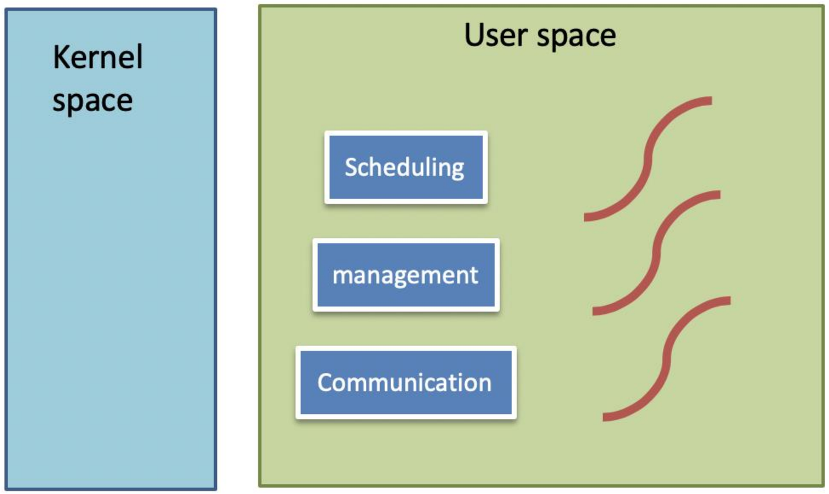
Discussion

- Disadvantages?



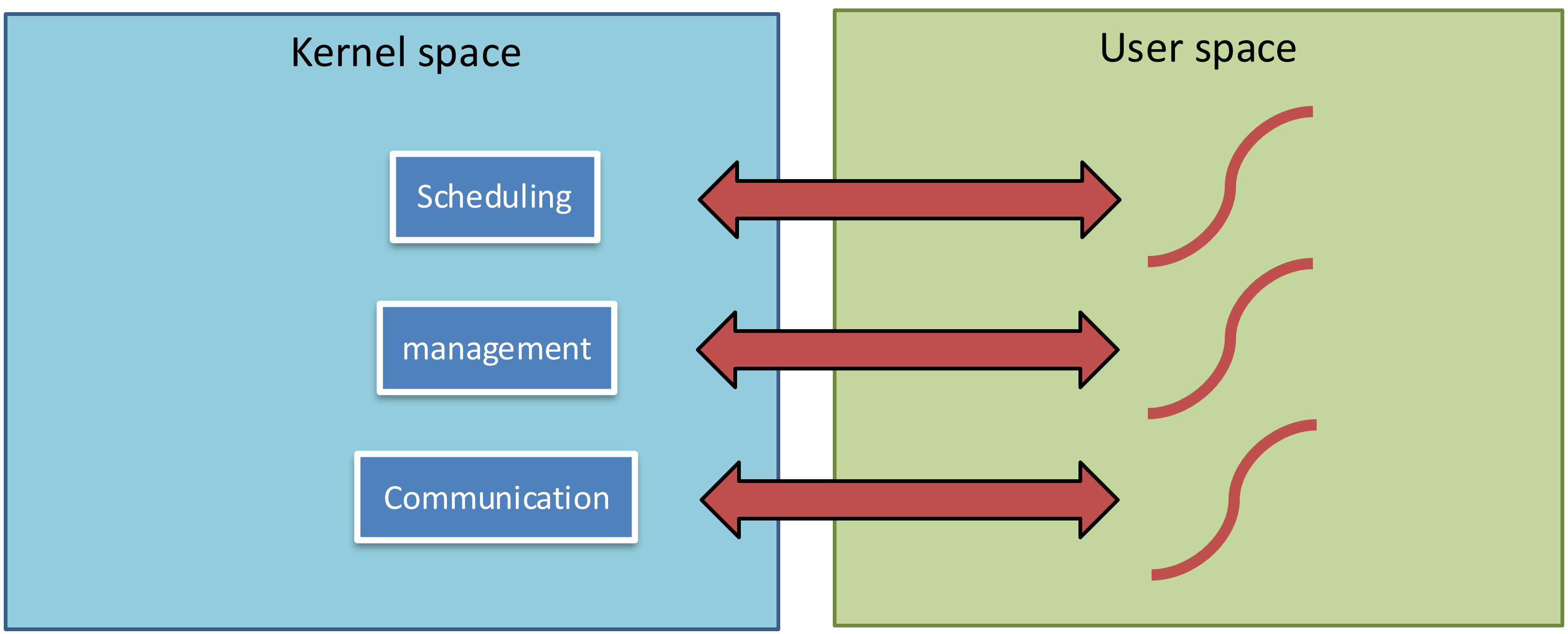
User-level Thread - Discussions

- Disadvantages (less kernel support, need to implement all logic by application itself)
  - Scheduling:
    - How blocking system calls implemented? Called by a thread?
    - How to change blocking system calls to non-blocking?
  - Memory management:
    - How to deal with page faults?
  - Interrupt
    - How to stop a thread from running forever? No clock interrupts



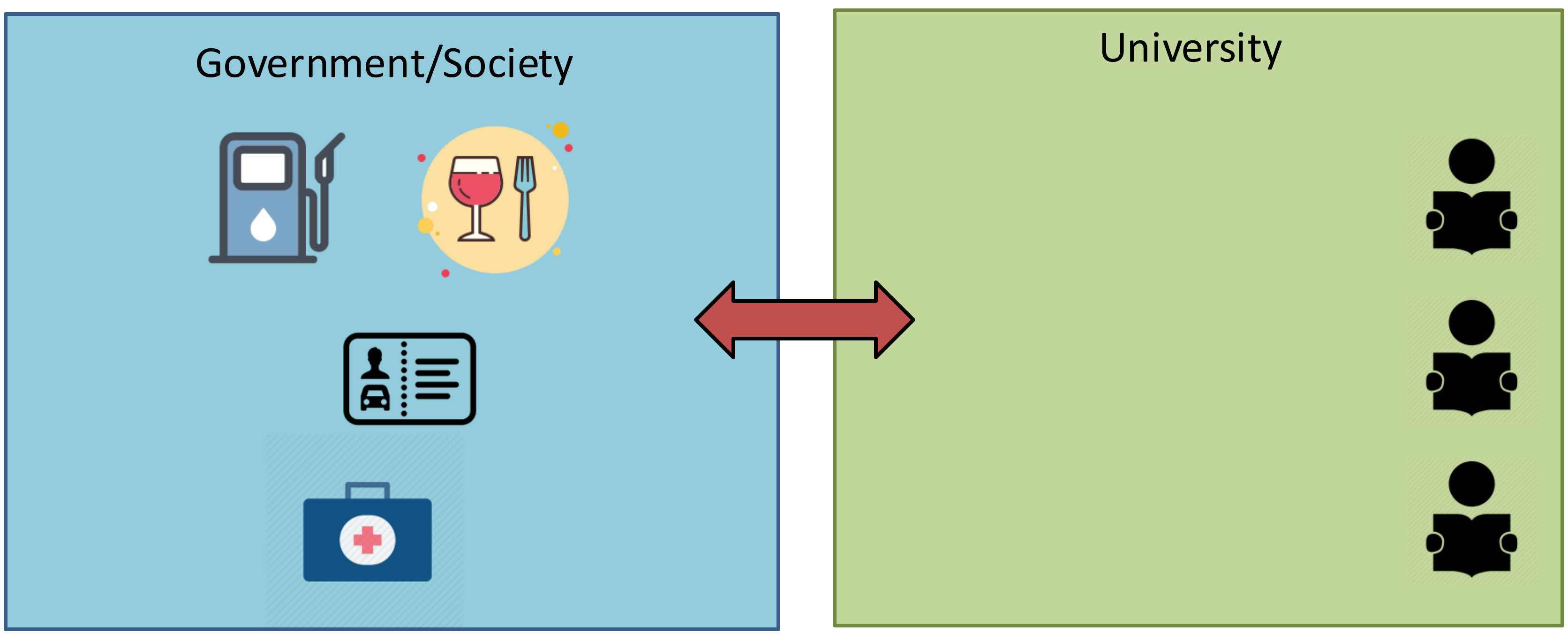
Kernel-level Thread

- When you have multiple threads, where to put them?



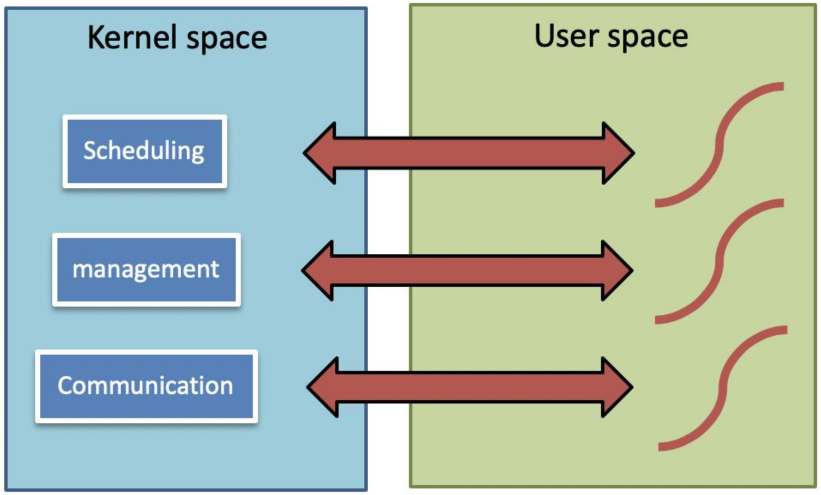
Discussion

- Advantages?



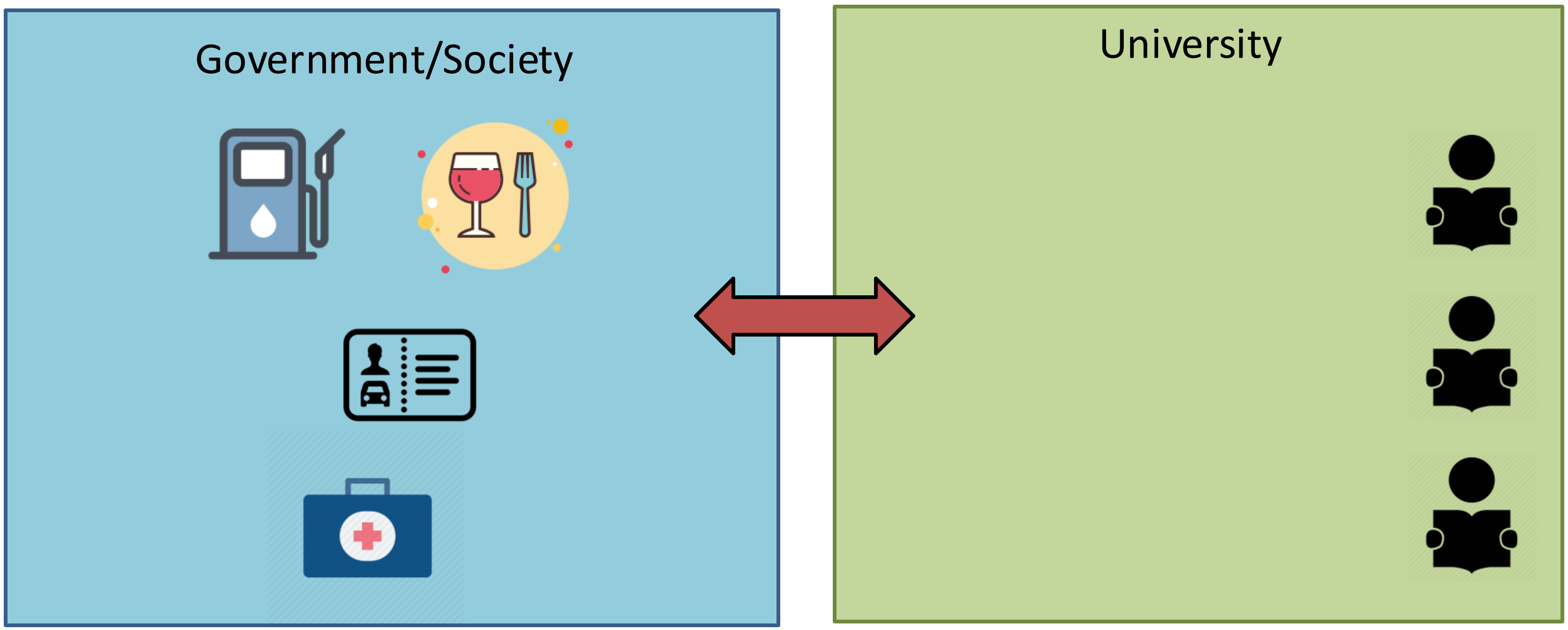
Implementing Threads in the Kernel

- Kernel-level threads: managed by the kernel
  - Threads known to OS, use OS service directly
    - Scheduling, memory management, storage, I/O, etc.



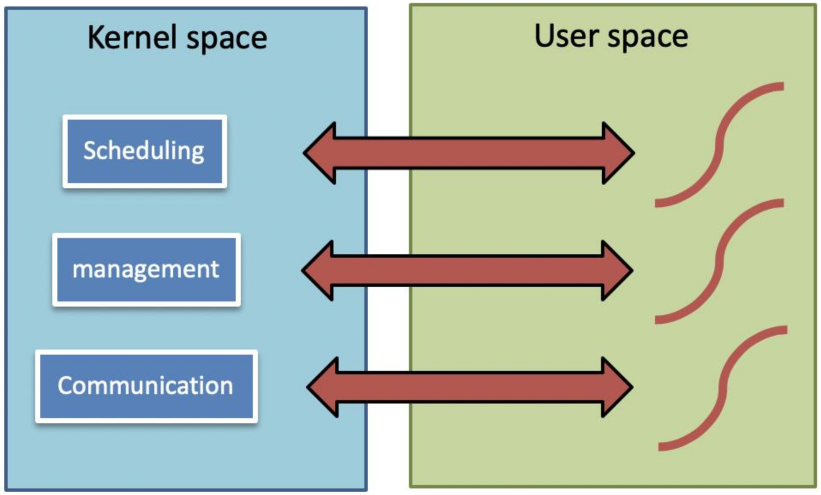
Discussion

- Disadvantages?



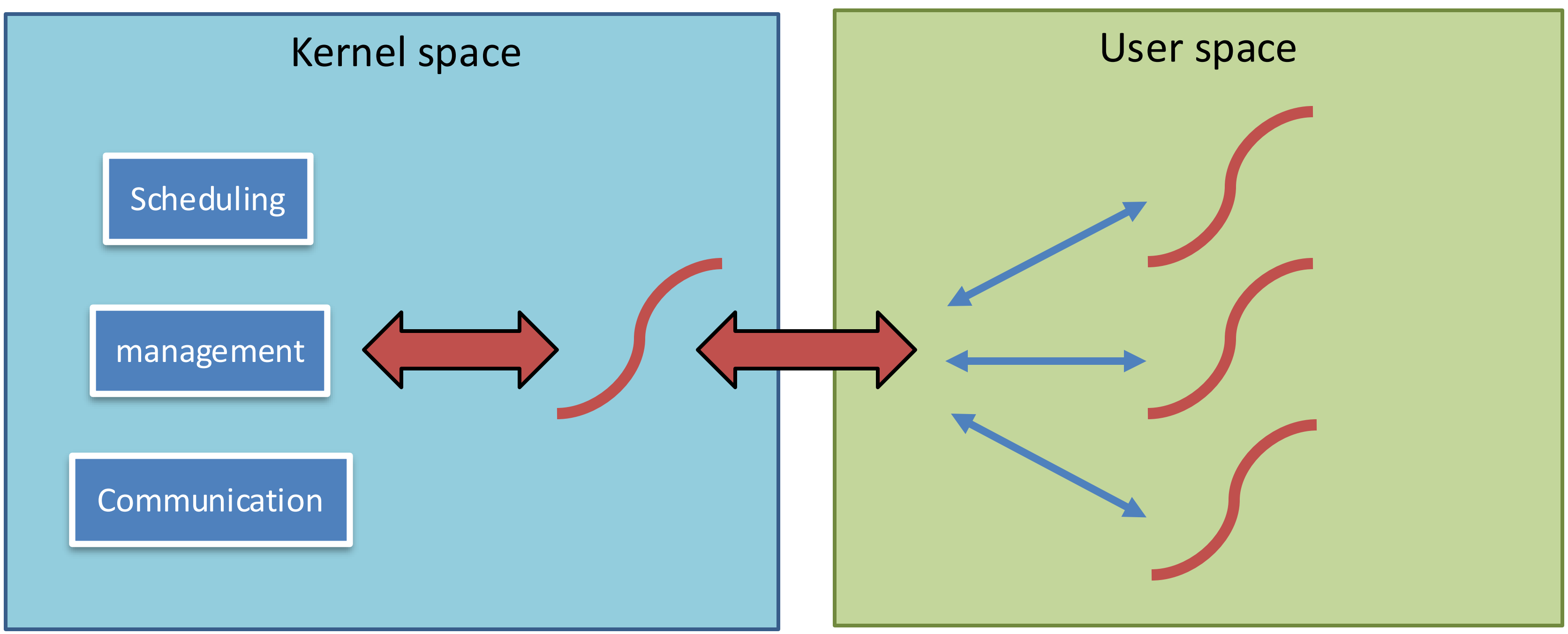
Implementing Threads in the Kernel

- Kernel-level threads: managed by the kernel
  - Threads known to OS, use OS service directly
    - Scheduling, memory management, storage, I/O, etc.
  - Slow
    - Trap into the kernel mode
  - Expensive to create and switch
    - Create memory inside the kernel
    - Context switch between the kernel and user space



Threads in Hybrid-Space

- When you have multiple threads, where to put them?



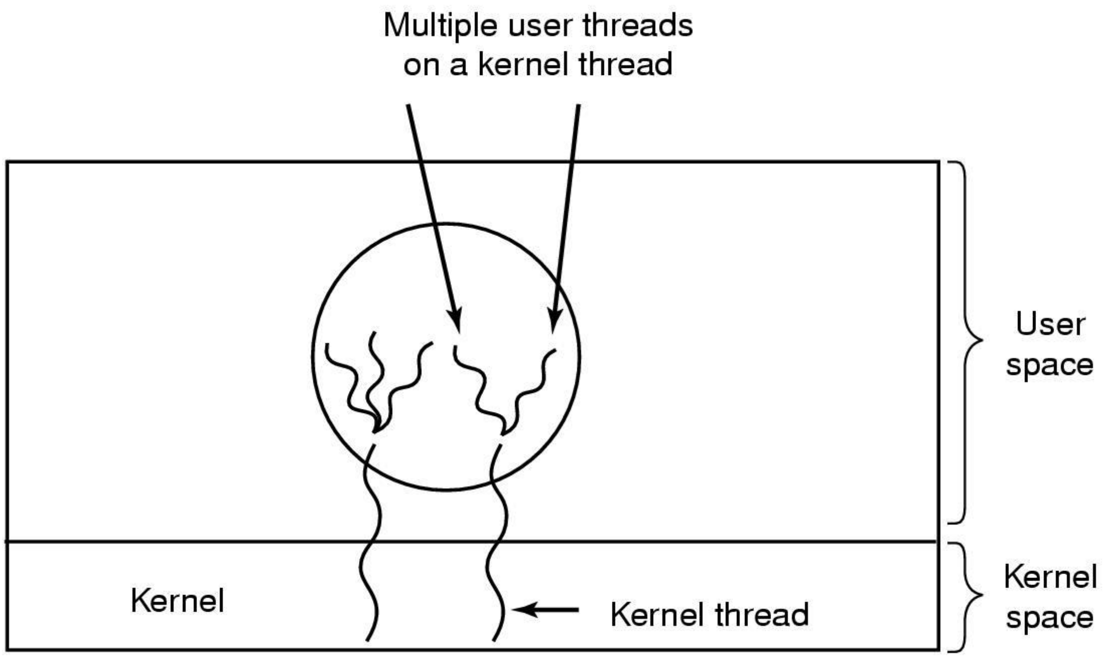
Hybrid Implementations

- Use kernel-level threads and then *multiplex* user-level threads onto some or all of the kernel-level threads

- Multiplexing user-level threads onto kernel-level threads

- Enjoy the benefits of user and kernel level threads

- Too complex !



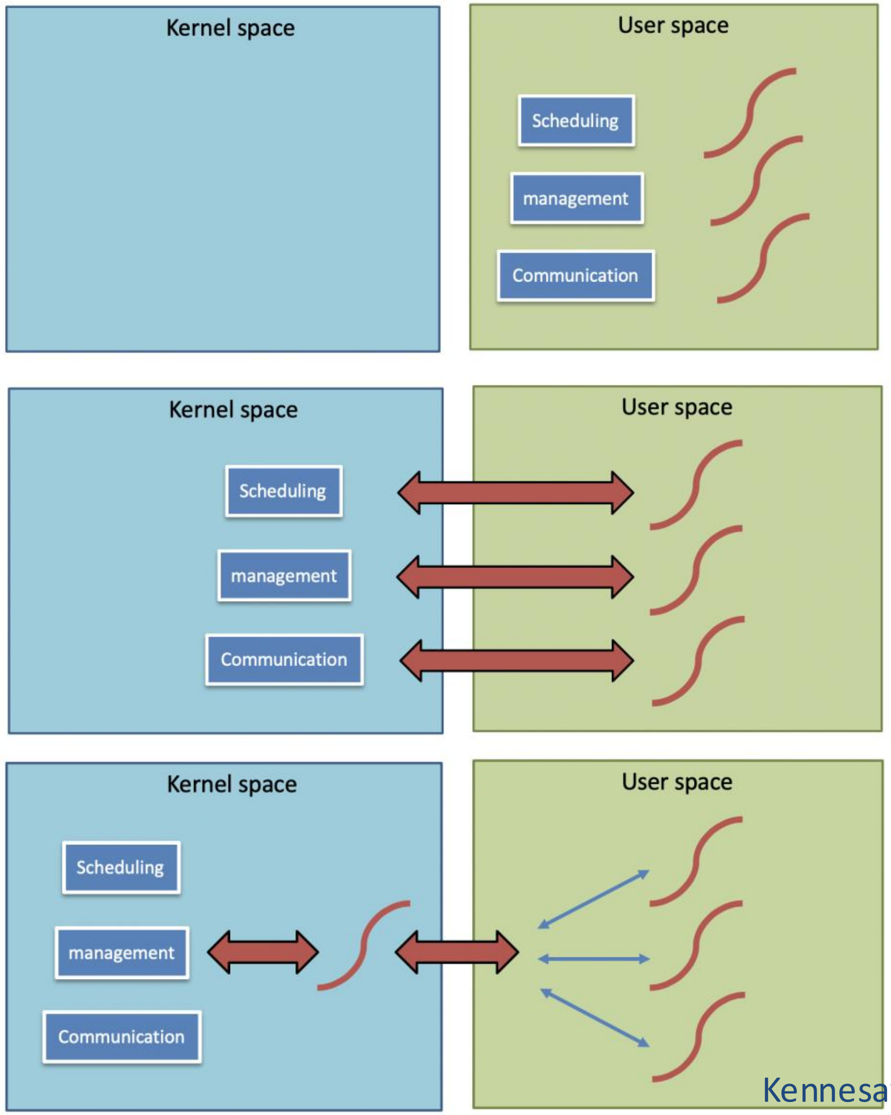
Multiplexing user-level threads onto kernel- level threads

Thread in user level/kernel level/hybrid

user

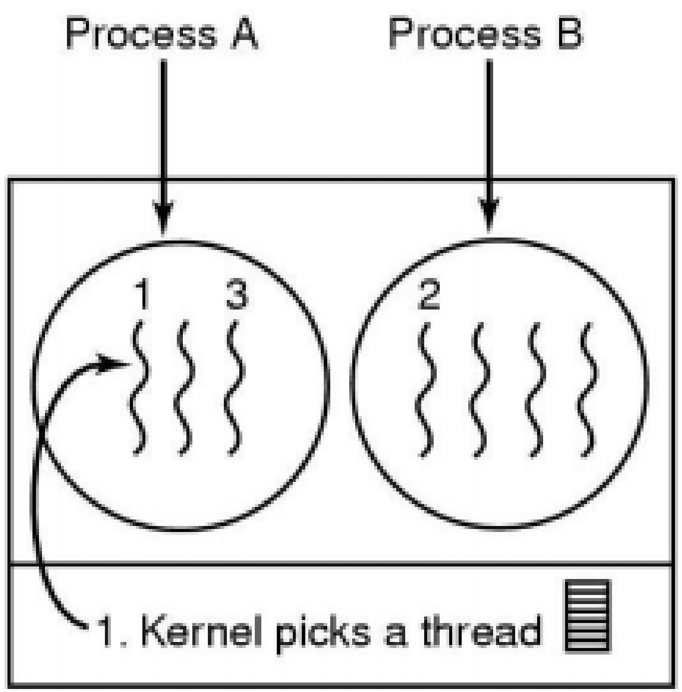
kernel

hybrid



Thread design

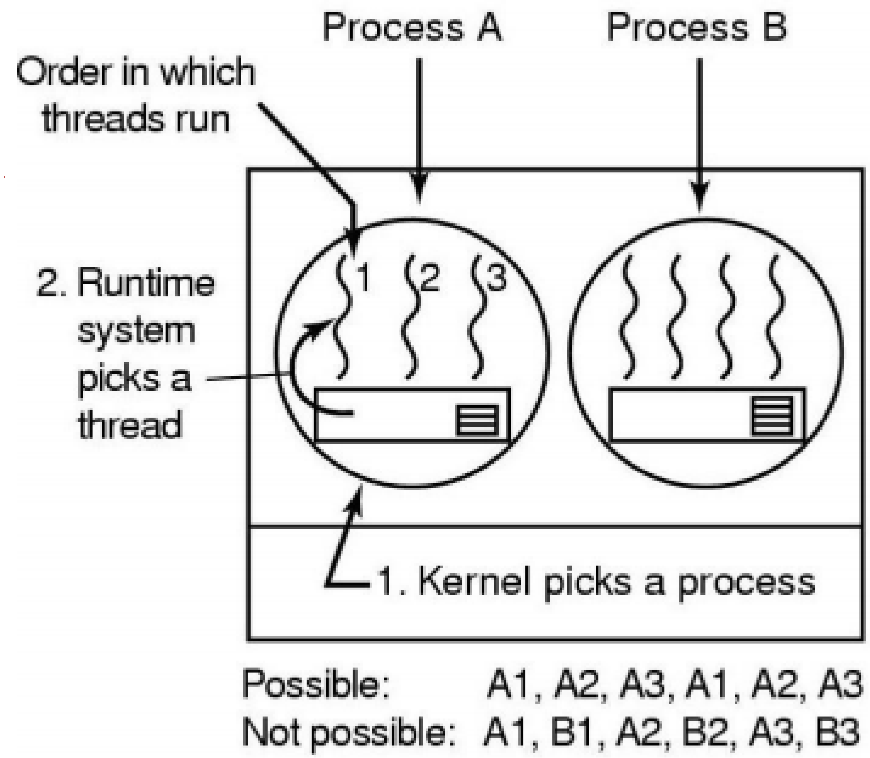
- When you have multiple threads, by what order should we execute them? - -> Scheduling



Possible:	A1, A2, A3, A1, A2, A3
Also possible:	A1, B1, A2, B2, A3, B3

Local Thread Scheduling

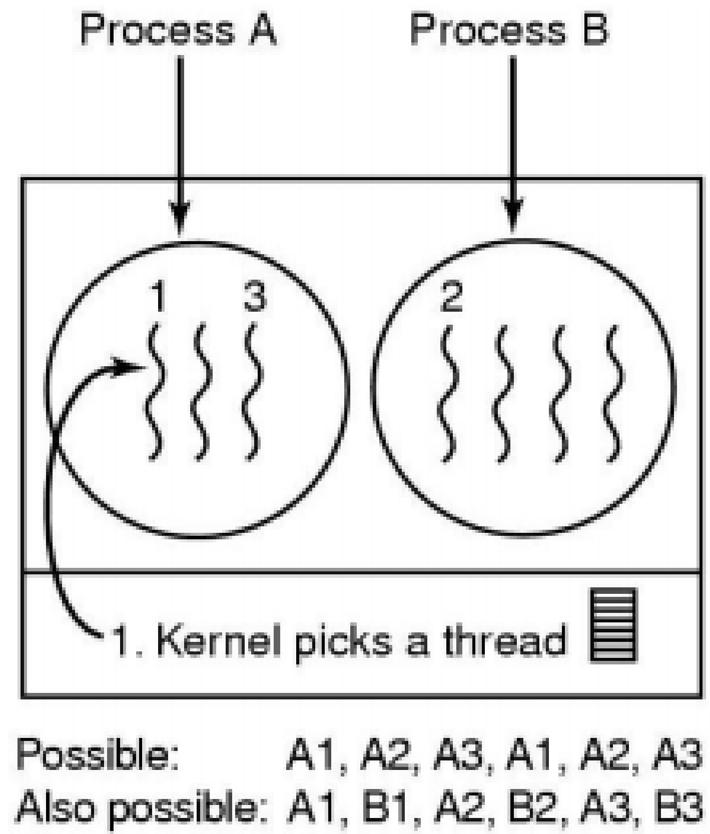
- Next thread is picked from among the threads belonging to the **current process**
- Each process gets a timeslice from kernel.
- Then the timeslice is divided up among the threads within the current process
- Scheduling decision requires only **local knowledge** of threads within the current process. It can be implemented by using **user or kernel level threads**



For example, say process timeslice may be N ms, and each thread within the process runs for N/3 ms per cycle

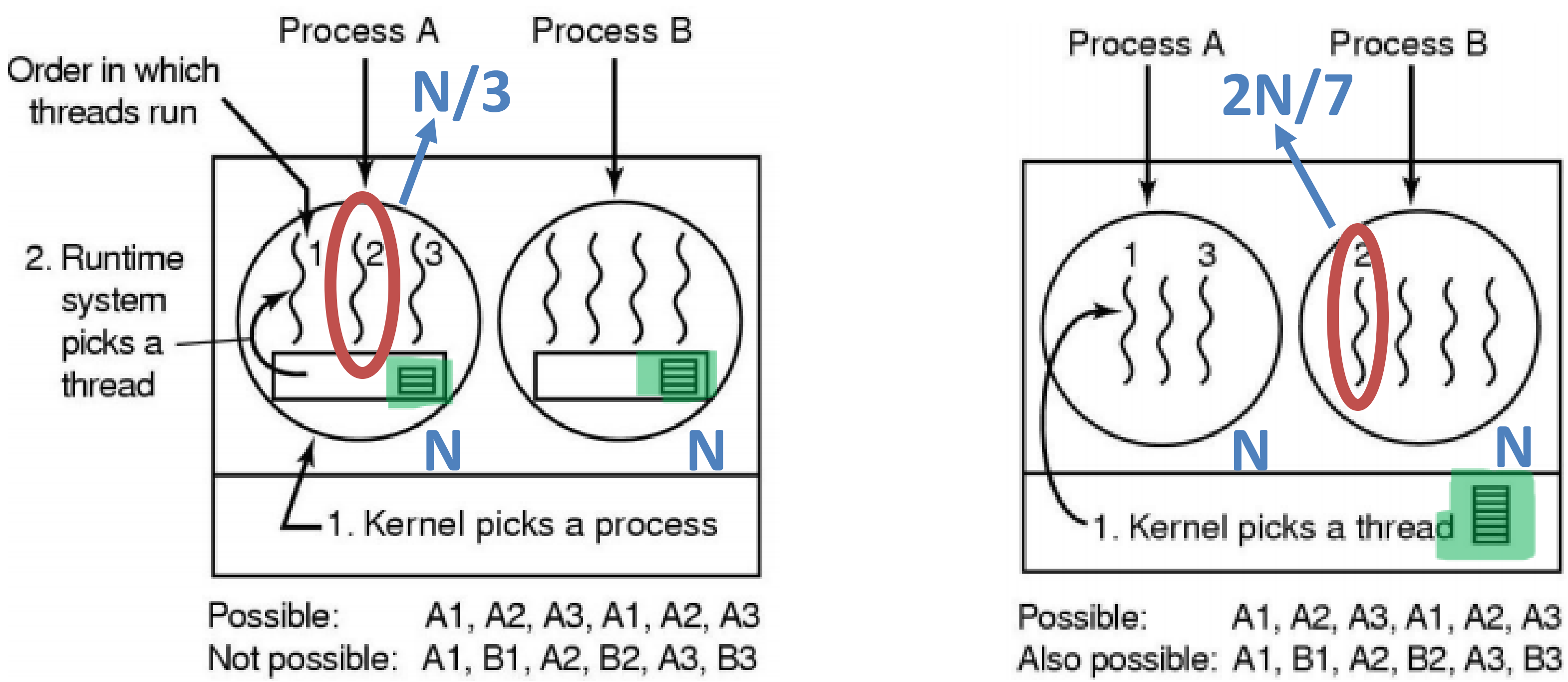
Global Thread scheduling

- Next thread to be scheduled is picked up from **ANY process** in the system.
  - Not just the current process
- Timeslice is allocated at the granularity of threads
  - No notion of per-process timeslice
- Global scheduling can be implemented **only with kernel-level threads**
  - Picking the next thread requires global knowledge of threads in all processes.



For example, say process timeslice may be N ms, For example each thread runs for 2N/7 ms per cycle

Local Thread Scheduling vs Global Thread scheduling



1, next thread

2, time slice for each thread

3, implementation: user/kernel level vs. only kernel level thread

Multiple threads → Pandora's Concurrency Box

- Multiple threads → Concurrency
- The illusion of concurrency is both **powerful** and **useful**:
  - It helps us think about how to structure our applications - -> multiple threads apps
  - It hides latencies caused by hardware devices - -> accelerate execution
- Unfortunately, concurrency also creates **problems**:
  - **Coordination**: how do we enable efficient communication between the multiple threads involved in performing a single task?
  - **Correctness**: how do we ensure that shared information remains consistent when being accessed by multiple threads concurrently?

Concurrency and multi-threads

- Unless precisely synchronized, threads may:
  - Be run in **any order**,
  - Be stopped and restarted at **any time**,
  - Remain stopped for **arbitrary lengths of time**.

Problems

- Generally these are **good things** - - the operating system is responsible for how to allocate resources.

Talk about CPU scheduling, memory management, lock, synchronization in future talks

Conclusion

- What is thread?
  - Multiple thread application
  - Thread vs Process
  - Advantage and disadvantage of thread
- Thread in Linux
- Thread design
  - Kernel space vs User space
  - Local thread vs Global thread scheduling