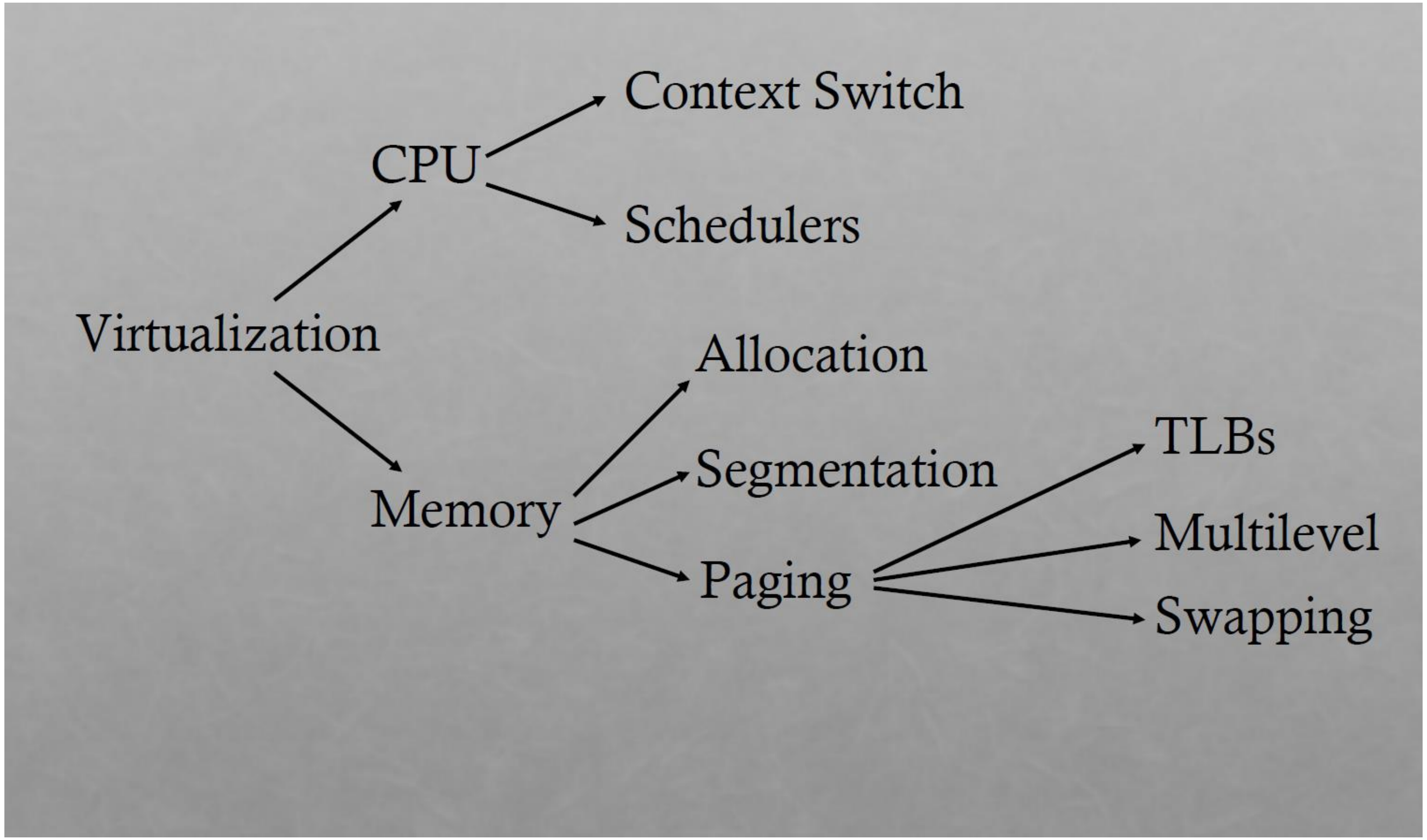
Big Picture so far…



Concurrency: Threads

Sridhar Alagar

Motivation

- Develop applications that utilizes many cores of CPU
- Build application using many processes
  - Example: Browser with one process per tab
    - Communicate using pipe or other IPC mechanisms
- Pros
  - No need to develop any new mechanisms
- Cons
  - Cumbersome programming
  - High communication overheads
  - Expensive context switch

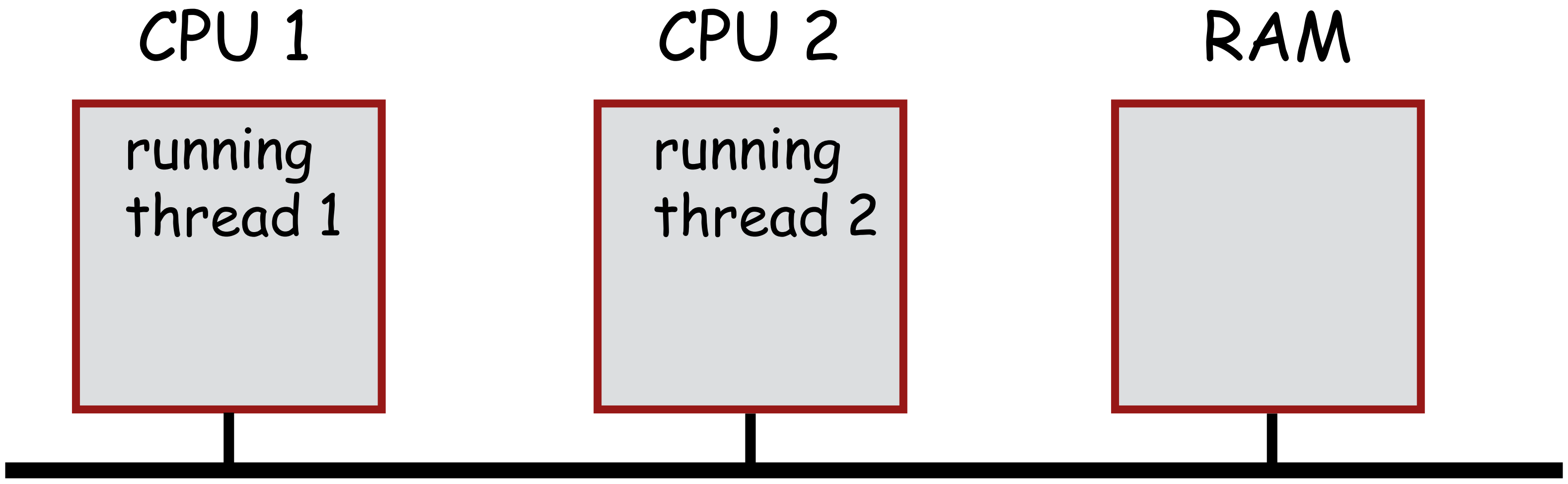
New Abstraction: Threads

- Threads are like processes except that
  - threads of the same process share the same address space

- Threads are like procedures within a process except that they can be run in parallel

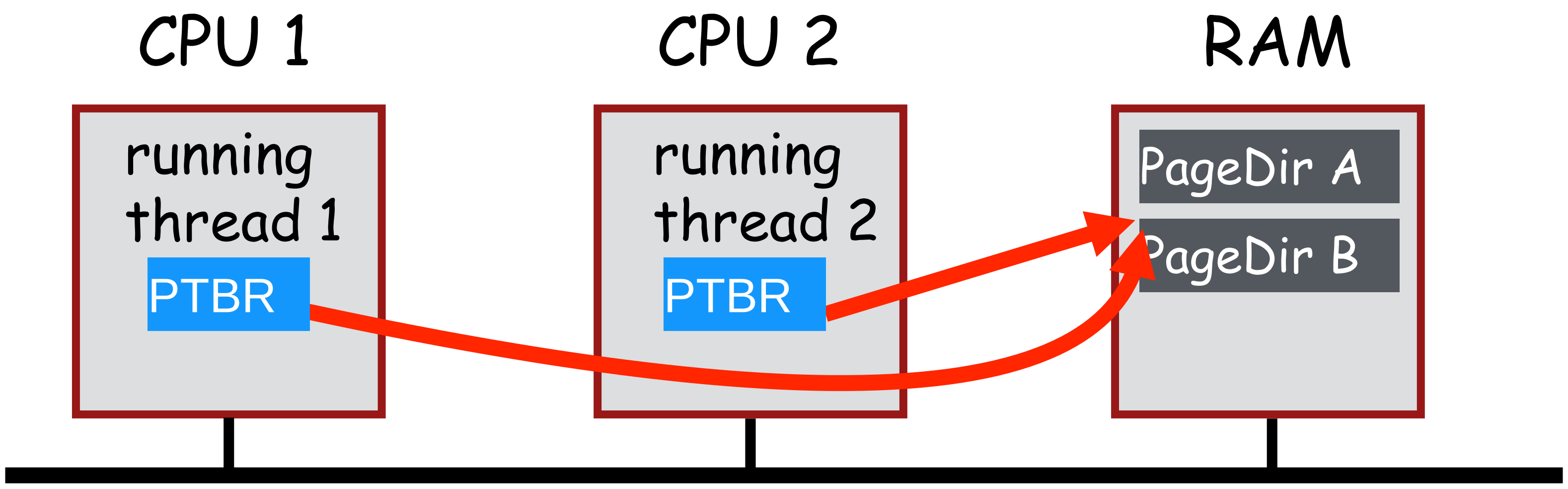
- Divide large tasks among multiple cooperating threads
  - Communicate through shared variables in shared address space

What state do threads share?

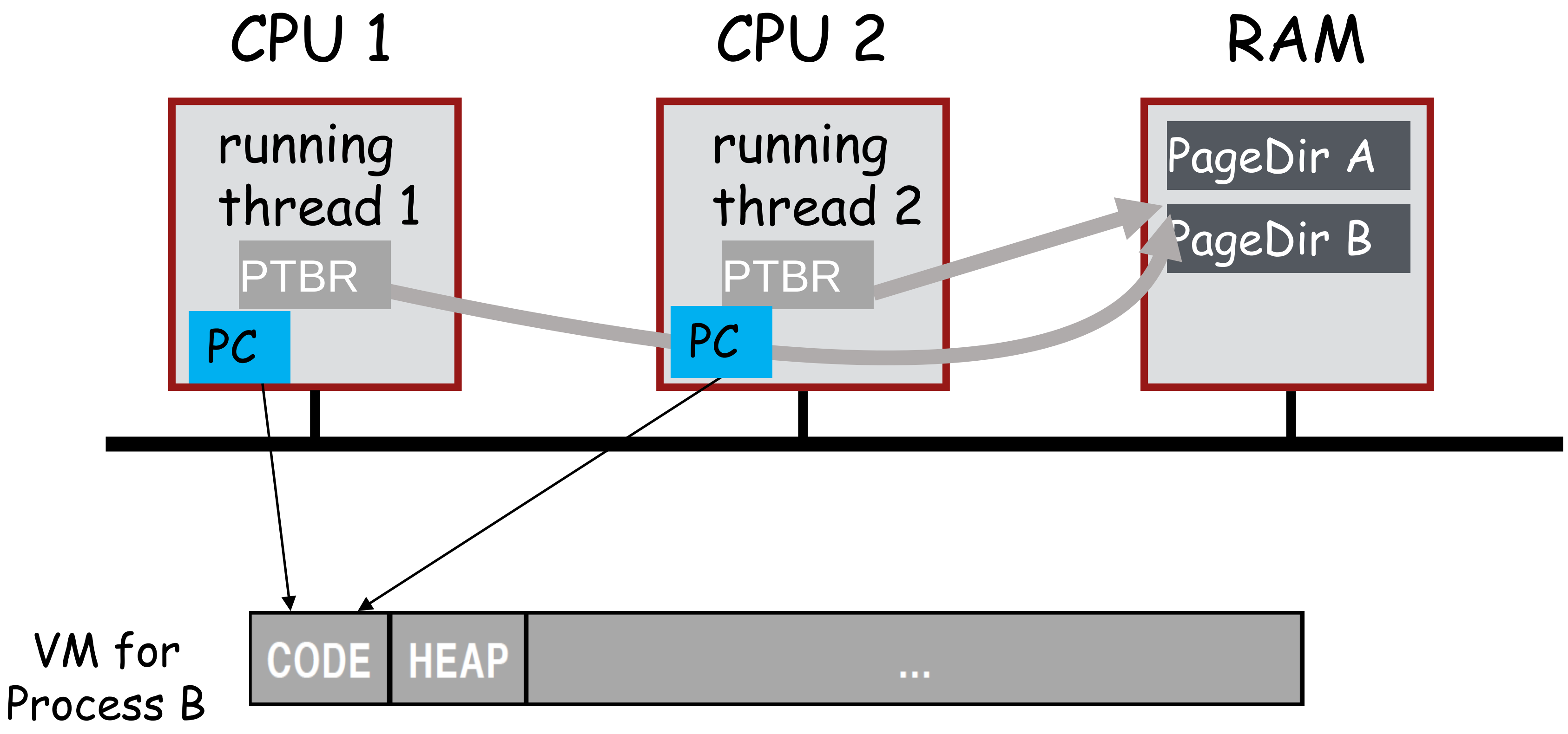


Address space

Share Address Space

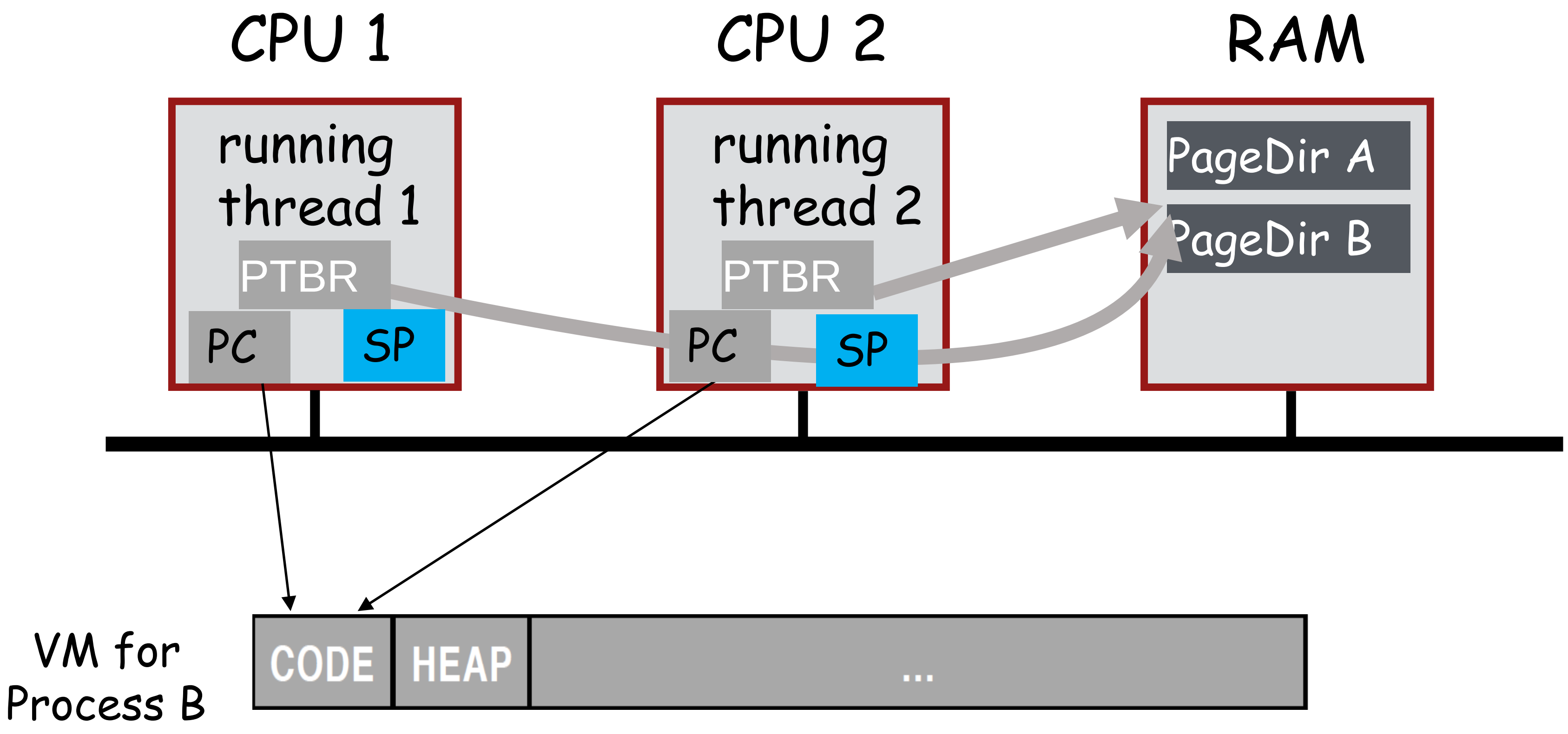


Instruction Pointer?



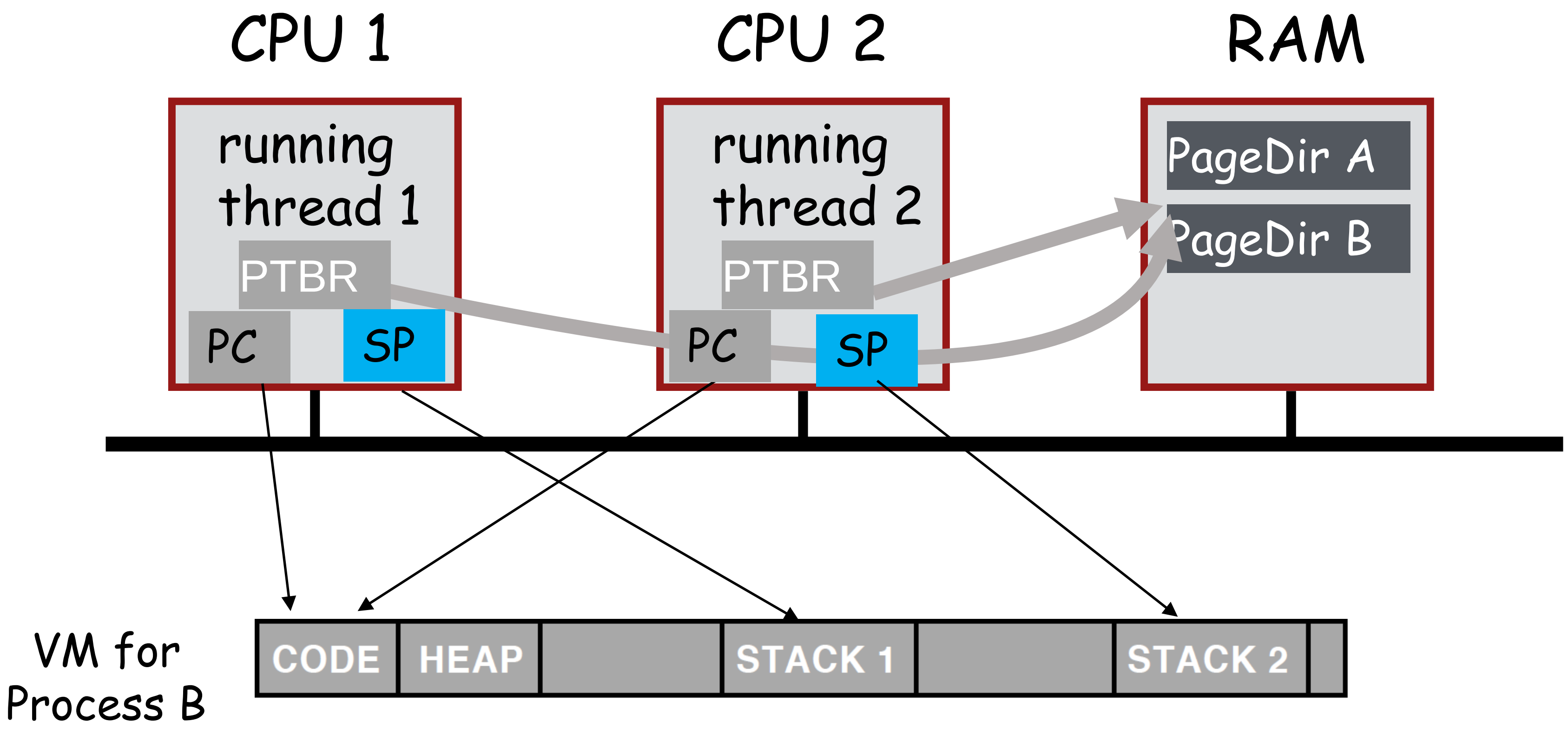
Share code; but each executing independently different parts of the code

Stack Pointer?



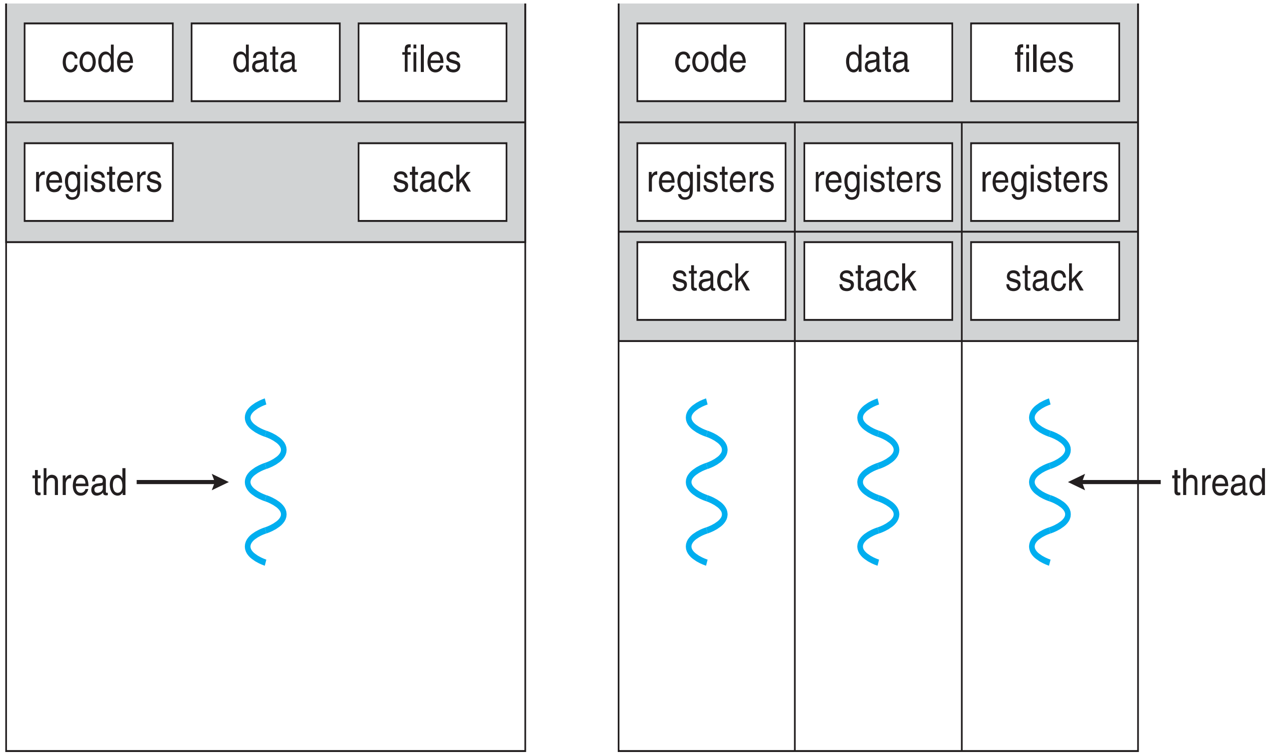
They execute different functions (instances)

Stack Pointer?



They execute different functions (instances)

Single and Multithreaded Processes



single-threaded process

multithreaded process

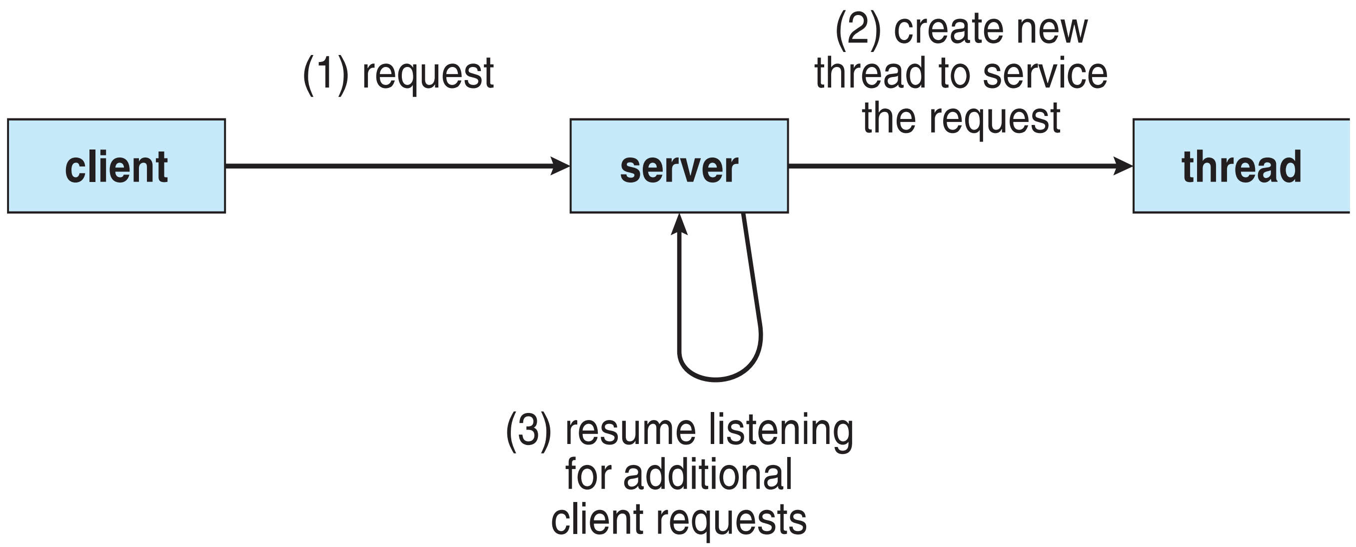
Threads vs Process

- Multiple threads within a same process share
  - Process ID
  - Address space
    - Code
    - Data (heap too)
  - Open file descriptors
  - Current Working directory
  - User and Group ID
- Each thread has its own
  - Thread ID
  - Registers including PC, and SP
  - Stack (in the same address space)

Programming Models

- Multiple tasks within the application can be implemented by separate threads.
- A word processor application uses several threads to
  - Get data from keyboard (foreground)
  - Spell checking and grammar (back ground)
  - Load images from file
  - Take a back up
- Another example is client server architecture

Multithreaded Server Architecture

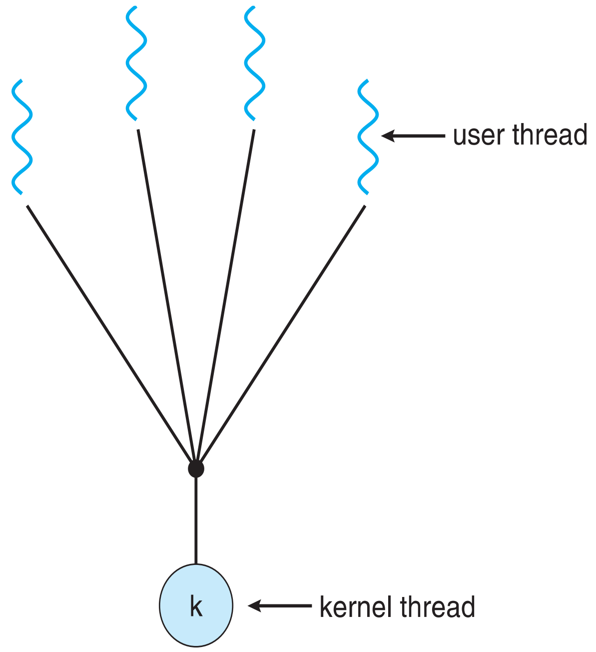


Benefits

- Responsiveness
  - one thread may continue to execute while part of process is blocked, especially important for user interfaces
- Resource Sharing
  - threads share resources(code, memory, files) of process; easier than shared memory or message passing
- Economy
  - cheaper than process creation, thread switching lower overhead than context switching
-

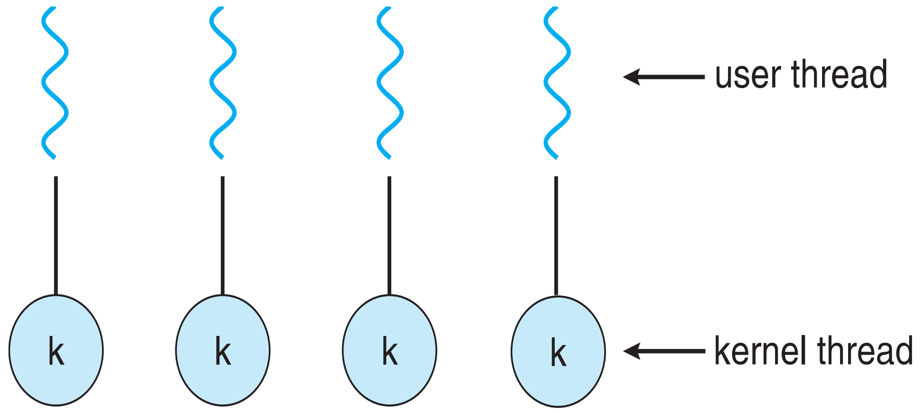
OS Support: Many-to-One

- Many user-level threads mapped to single kernel thread
- Implemented by the user level runtime libraries
  - Create, schedule, synchronize at user level
- Kernel not aware of user level threads
  - Thinks each process contain single thread



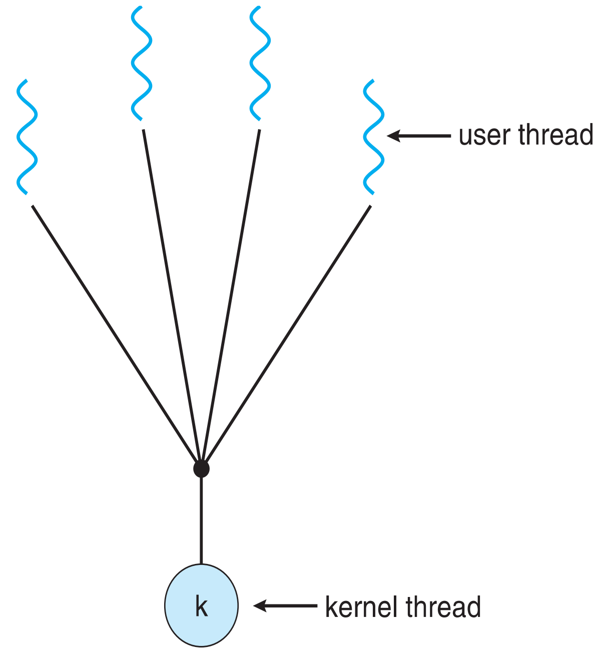
OS support: One-to-One

- Each user-level thread mapped to a kernel thread by the OS
- Each kernel thread scheduled independently
- Thread operation performed by the OS



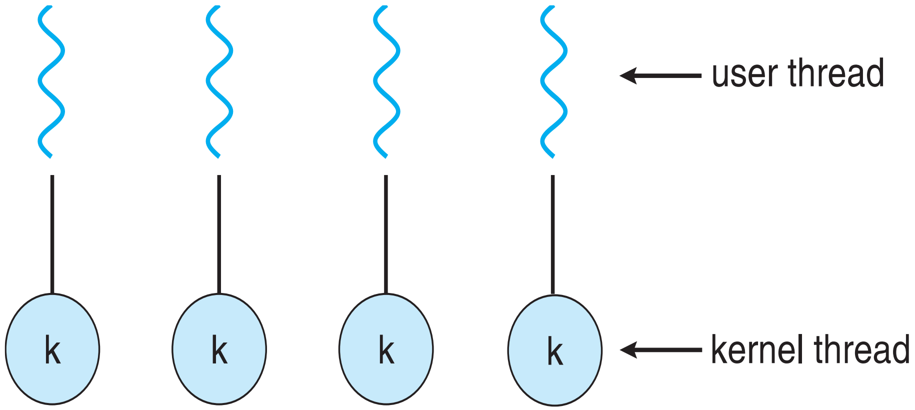
OS Support: Many-to-One

- Pros
  - Does not require OS support; Portable
  - Lower overhead thread operation since no system call
- Cons
  - One thread blocking causes all to block
  - Multiple threads cannot run in parallel on muticore system



OS support: One-to-One

- No blocking of other threads
- Can in run in parallel on a multiprocessor
- Higher overhead
- OS should be scalable or limit # threads
- Examples
  - Windows
  - Linux



Thread API

- Thread library provides the API for creating and managing threads
- Several exists
  - POSIX Pthreads
- Common thread operations:
  - Create
  - Exit
  - Join (instead of wait for process)

Thread Creation

```
#include <pthread.h>

int
pthread_create(      pthread_t*        thread,
               const pthread_attr_t*  attr,
                     void*  (*start_routine)(void*),
                     void*             arg);
```

Exit a thread

```
int
pthread_exit(void *value_ptr);
```

- `value_ptr`: A pointer to the return value

Wait for a thread to complete

```
int
pthread_join(pthread_t thread, void **value_ptr);
```

- `thread`: Specify which thread to wait for
- `value_ptr`: A pointer to the return value

Thread Schedule #1

`counter = counter + 1; counter at 0x9cd4`

State:
0x9cd4: 100
%eax: ?
%rip = 0x195

process control blocks:

Thread 1	Thread 2
%eax: ?	%eax: ?
%rip: 0x195	%rip: 0x195

T1

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4A

Thread Schedule #1

State:
0x9cd4: 100
%eax: 100
%rip = 0x19a

process control blocks:

	Thread 1	Thread 2
	%eax: ? %rip: 0x195	%eax: ? %rip: 0x195

T1

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

Thread Schedule #1

State:
0x9cd4: 100
%eax: 101
%rip = 0x19d

process control blocks:

	Thread 1	Thread 2
%eax	?	?
%rip	0x195	0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T1

Thread Schedule #1

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

	Thread 1	Thread 2
%eax	?	?
%rip	0x195	0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T1

Thread Schedule #1

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: ?	%eax: ?
%rip: 0x195	%rip: 0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T1

Thread Context Switch

Thread Schedule #1

State:
0x9cd4: 101
%eax: ?
%rip = 0x195

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: ? %rip: 0x195

T2 →

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

Thread Schedule #1

State:
0x9cd4: 101
%eax: 101
%rip = 0x19a

process control blocks:

	Thread 1	Thread 2
%eax	101	?
%rip	0x1a2	0x195

T2

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

Thread Schedule #1

State:
0x9cd4: 101
%eax: 102
%rip = 0x19d

process control blocks:

	Thread 1	Thread 2
	%eax: 101 %rip: 0x1a2	%eax: ? %rip: 0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T2

Thread Schedule #1

State:
0x9cd4: 102
%eax: 102
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: ? %rip: 0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T2

Thread Schedule #1

State:
0x9cd4: 102
%eax: 102
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: ? %rip: 0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T2 ➔

Desired Result!

Another schedule

Thread Schedule #2

State:
0x9cd4: 100
%eax: ?
%rip = 0x195

process control blocks:

Thread 1	Thread 2
%eax: ? %rip: 0x195	%eax: ? %rip: 0x195

T1

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

Thread Schedule #2

State:
0x9cd4: 100
%eax: 100
%rip = 0x19a

process control blocks:

Thread 1	Thread 2
%eax: ?	%eax: ?
%rip: 0x195	%rip: 0x195

T1 →

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

Thread Schedule #2

State:
0x9cd4: 100
%eax: 101
%rip = 0x19d

process control blocks:

	Thread 1	Thread 2
	%eax: ? %rip: 0x195	%eax: ? %rip: 0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T1 →

Thread Context Switch

Thread Schedule #2

State:
0x9cd4: 100
%eax: ?
%rip = 0x195

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x19d	%eax: ? %rip: 0x195

T2 →

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

Thread Schedule #2

State:
0x9cd4: 100
%eax: 100
%rip = 0x19a

process control blocks:

	Thread 1	Thread 2
%eax	101	?
%rip	0x19d	0x195

T2 →

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

Thread Schedule #2

State:
0x9cd4: 100
%eax: 101
%rip = 0x19d

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x19d	%eax: ? %rip: 0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- T2 → 0x19d mov %eax, 0x9cd4

Thread Schedule #2

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x19d	%eax: ? %rip: 0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4A

T2 →

Thread Schedule #2

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x19d	%eax: ? %rip: 0x195

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- T2 → 0x19d mov %eax, 0x9cd4

Thread Context Switch

Thread Schedule #2

State:
0x9cd4: 101
%eax: 101
%rip = 0x19d

process control blocks:	Thread 1	Thread 2
	%eax: 101 %rip: 0x19d	%eax: 101 %rip: 0x1a2

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- T1 → 0x19d mov %eax, 0x9cd4

Thread Schedule #2

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: 101 %rip: 0x1a2

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T1

Thread Schedule #2

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: 101 %rip: 0x1a2

- 0x195 mov 0x9cd4, %eax
- 0x19a add $0x1, %eax
- 0x19d mov %eax, 0x9cd4

T1

WRONG Result! Final value of counter is 101

Timeline View

Thread 1	**Thread 2**
mov 0x123, %eax	
add %0x1, %eax	
mov %eax, 0x123	
	mov 0x123, %eax
	add %0x2, %eax
	mov %eax, 0x123

How much is added to shared variable? 3: correct!

Timeline View

Thread 1	**Thread 2**
mov 0x123, %eax	
add %0x1, %eax	
	mov 0x123, %eax
mov %eax, 0x123	
	add %0x2, %eax
	mov %eax, 0x123

How much is added?

2: incorrect!

Timeline View

Thread 1	Thread 2
	mov 0x123, %eax
mov 0x123, %eax	
	add %0x2, %eax
add %0x1, %eax	
	mov %eax, 0x123
mov %eax, 0x123	

How much is added? 1: incorrect!

Timeline View

Thread 1	**Thread 2**
	mov 0x123, %eax
	add %0x2, %eax
	mov %eax, 0x123
mov 0x123, %eax	
add %0x1, %eax	
mov %eax, 0x123	

How much is added? 3: correct!

Timeline View

Thread 1	**Thread 2**
	mov 0x123, %eax
	add %0x2, %eax
mov 0x123, %eax	
add %0x1, %eax	
mov %eax, 0x123	
	mov %eax, 0x123

How much is added?

2: incorrect!

Non-Determinism

- Concurrency leads to non-deterministic results
  - Non deterministic result: different results even with same inputs
  - race conditions
- Whether bug manifests depends on CPU schedule!
- Passing tests means little
- How to program: imagine scheduler is malicious
  - Assume scheduler will pick bad ordering at some point...

What do we want?

- Want 3 instructions to execute as an uninterruptable group
- That is, we want them to be atomic

```
mov 0x123, %eax
add %0x1, %eax
mov %eax, 0x123
```

critical section

- Need mutual exclusion for critical sections
- if process A is in critical section, process B can't be in CS
  (okay if other processes do unrelated work)

Solution using Locks

- Allocate and Initialize
  - `Pthread_mutex_t mylock;`
  - `Pthread_mutex_init(&mylock, NULL);`

- Acquire: `pthread_mutex_lock(&mylock)`
  - Acquire exclusion access to lock;
  - Wait if lock is not available (some other process in critical section)
  - Spin or block (relinquish CPU) while waiting

- Release: `pthread_mutex_unlock(&mylock)`
  - Release exclusive access to lock; let another process enter critical section

Disclaimer

- Some of the materials in this lecture slides are from the lecture slides by Prof. Arpaci, Prof. Youjip, and other educators. Thanks to all of them.