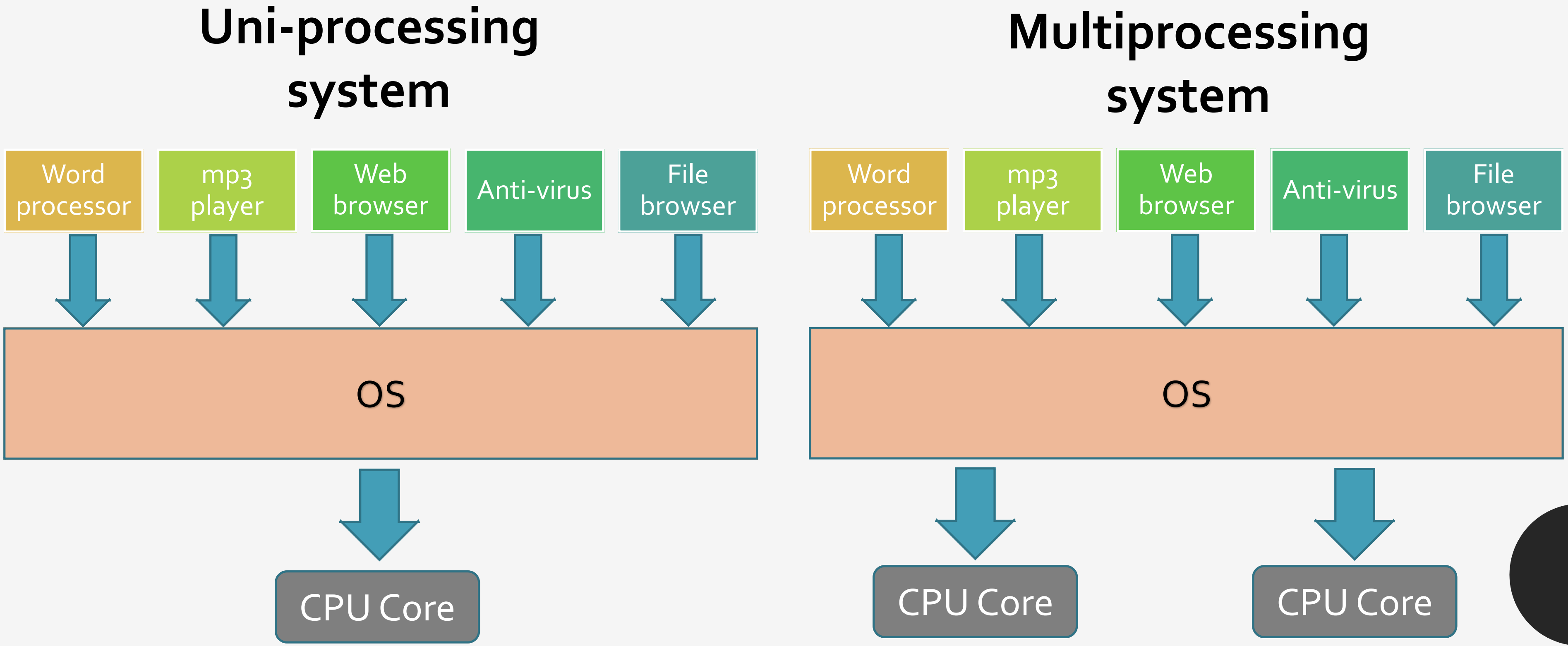
CS35L – *Winter ‘19*

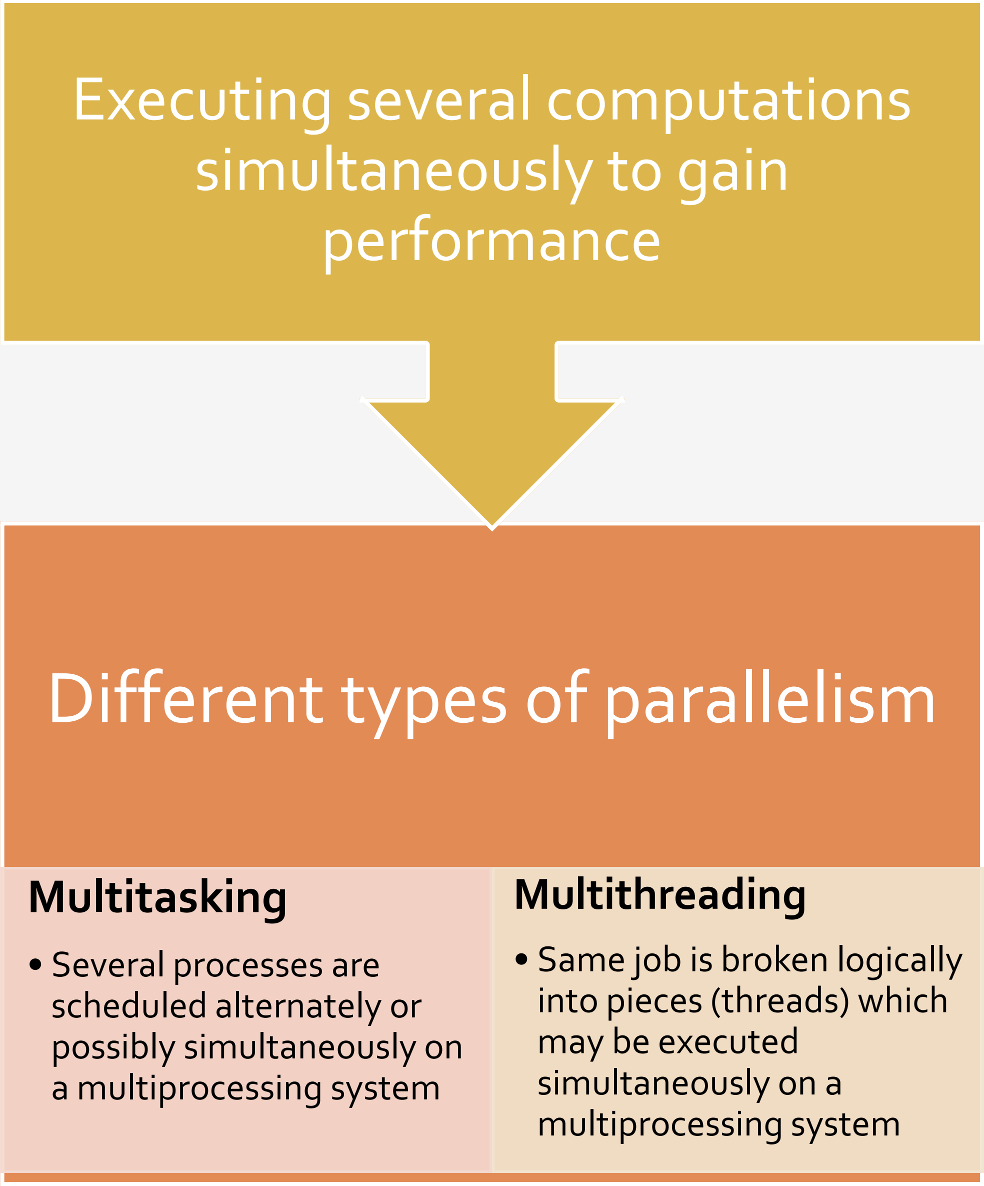
Slide set:	6.1
Slide topics:	Multithreaded performance
Assignment:	6

Multiprocessing

The use of multiple CPUs/cores to run multiple tasks simultaneously



Parallelism



Executing several computations simultaneously to gain performance

Different types of parallelism

Multitasking

- Several processes are scheduled alternately or possibly simultaneously on a multiprocessing system

Multithreading

- Same job is broken logically into pieces (threads) which may be executed simultaneously on a multiprocessing system

Threads

- A flow of instructions, path of execution within a process
- The smallest unit of processing scheduled by OS
- A process consists of at least one thread
- Multiple threads can be run on:
  - **A uniprocessor (time-sharing)**
    - Processor switches between different threads
    - Parallelism is an illusion
  - **A multiprocessor**
    - Multiple processors or cores run the threads at the same time
    - True parallelism

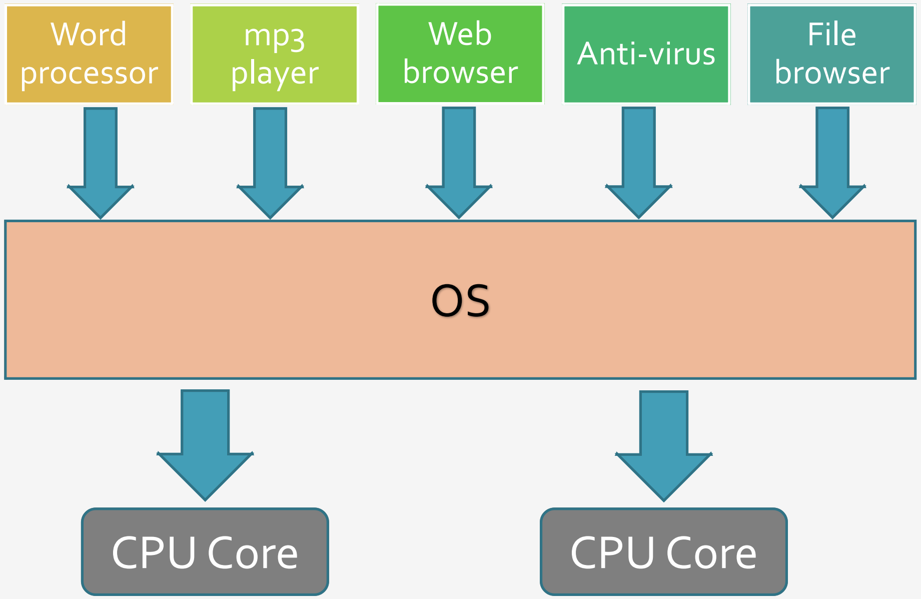
Multitasking vs. Multithreading

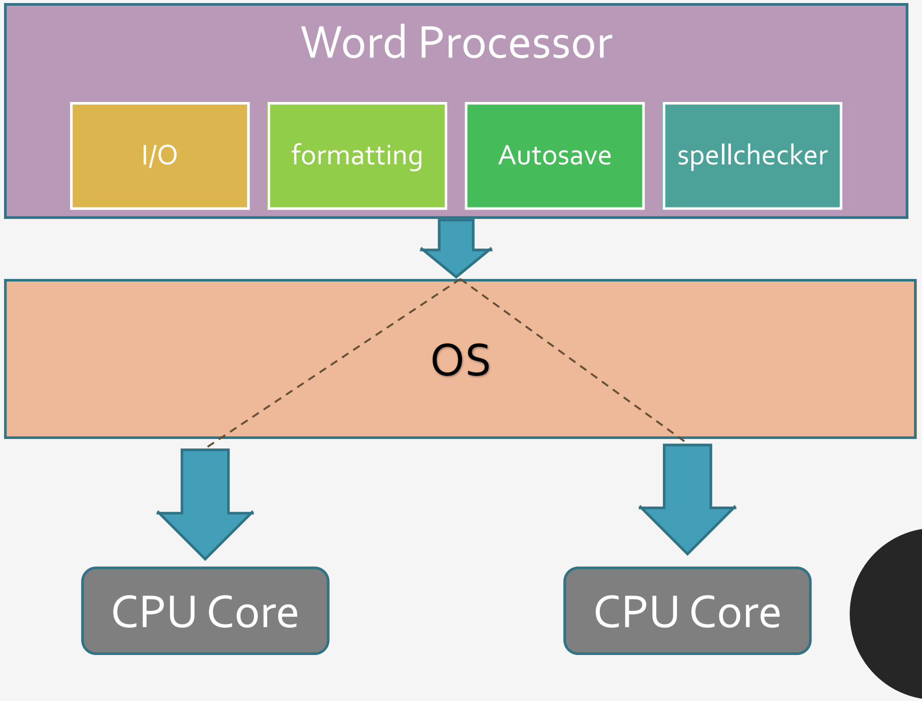
Multitasking

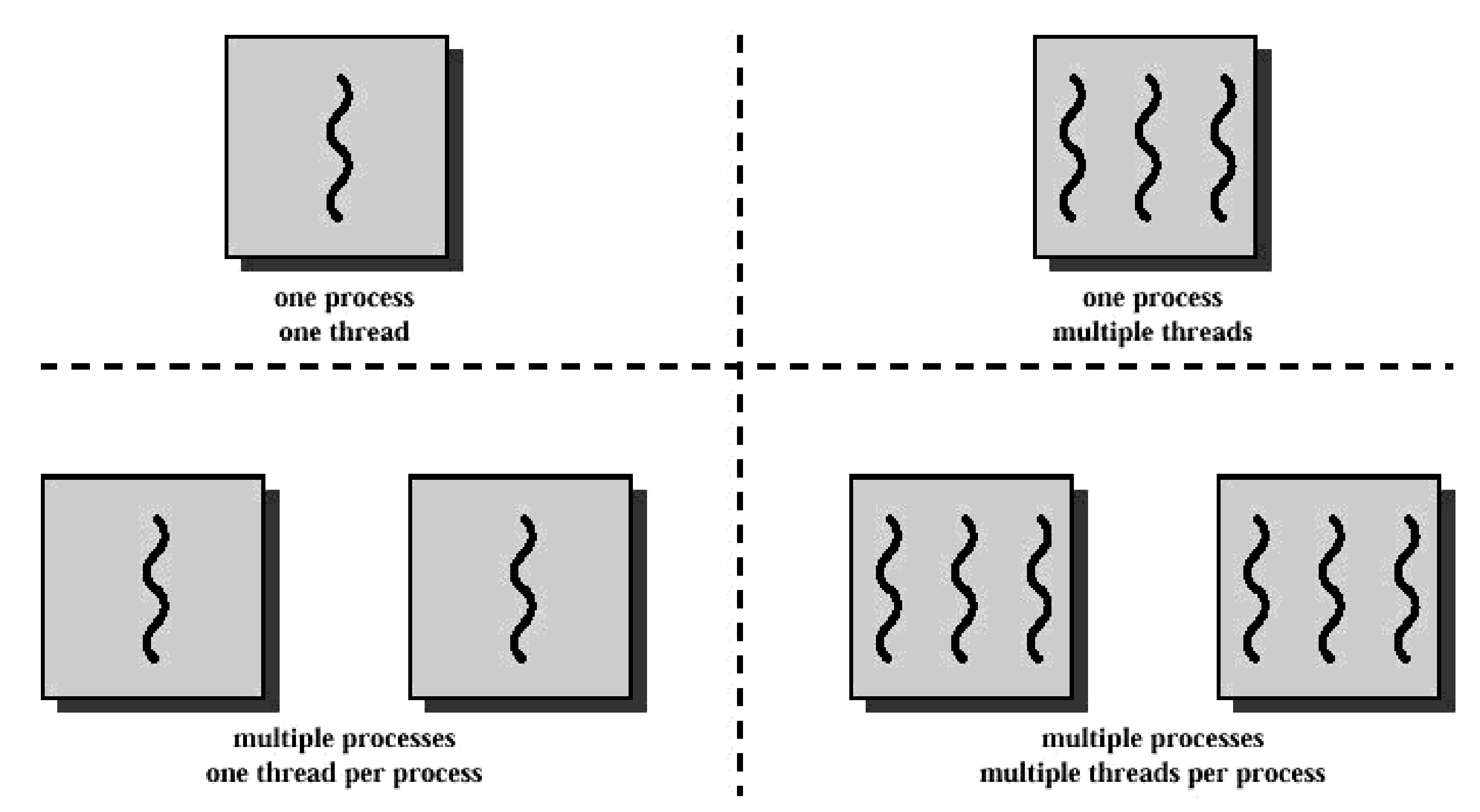
Sharing of computing resources (CPU, memory, devices, etc.) among processes

Multithreading

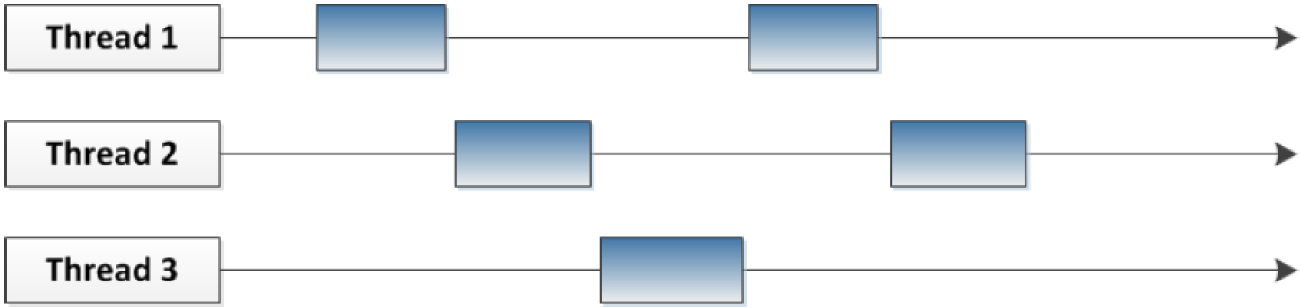
Sharing of computing resources among threads of a single process.



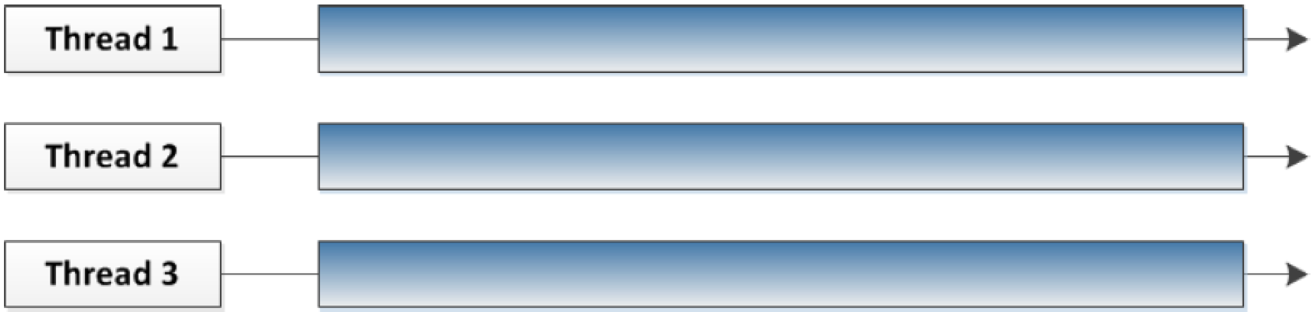



one process
one thread
one process
multiple threads
multiple processes
one thread per process
multiple processes
multiple threads per process

Multiple threads sharing a single CPU



Multiple threads on multiple CPUs



Multitasking

```
$ tr ’abc’ ‘xyz’ | sort -u |
comm -23  file1
```

- Process 1 (tr)
- Process 2 (sort)
- Process 3 (comm)

Each process has its own address space

- Area of memory (virtualized) devoted to the process

Communication between processes

- Via Pipes/System Calls

01

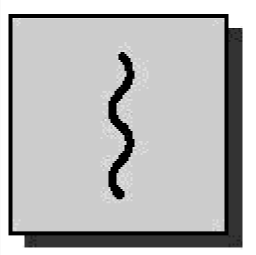
Threads share all of the process's memory except for their stacks

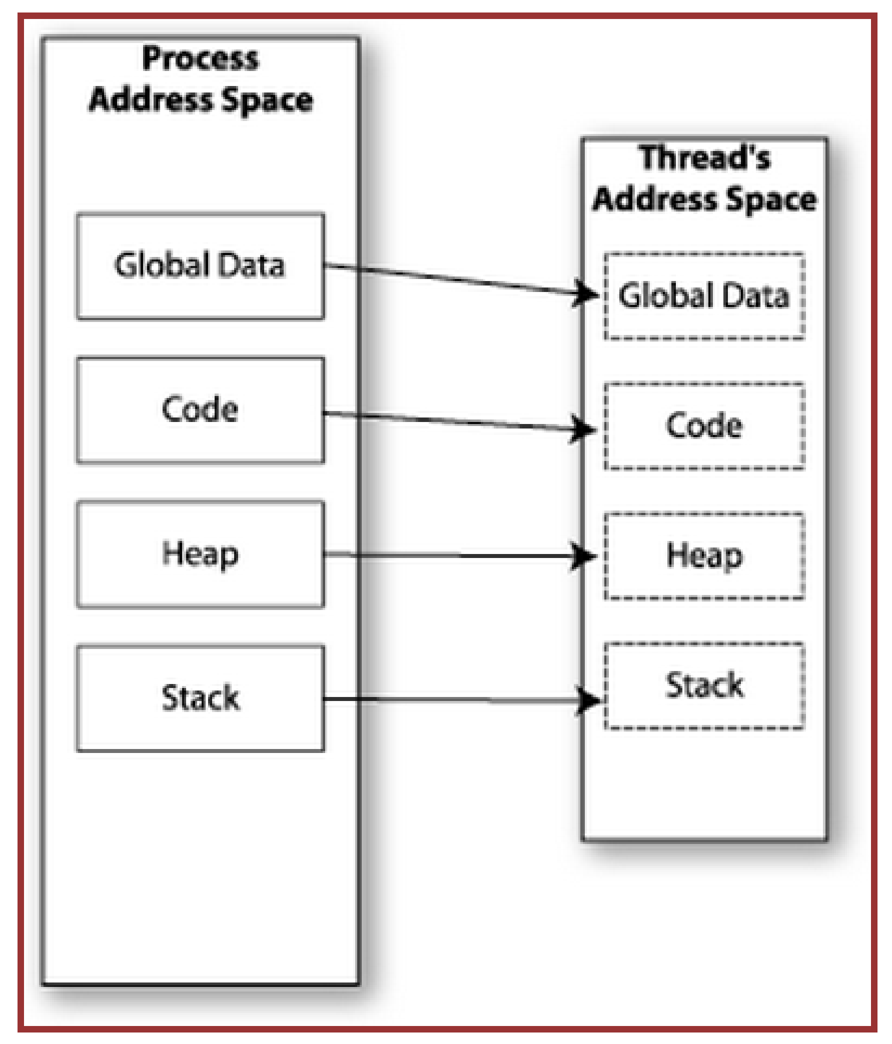
02

Data sharing requires no extra work (no system calls, pipes, etc.)

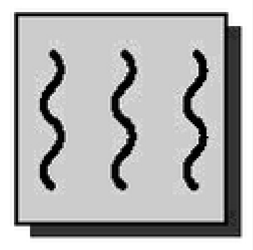
Multithreading

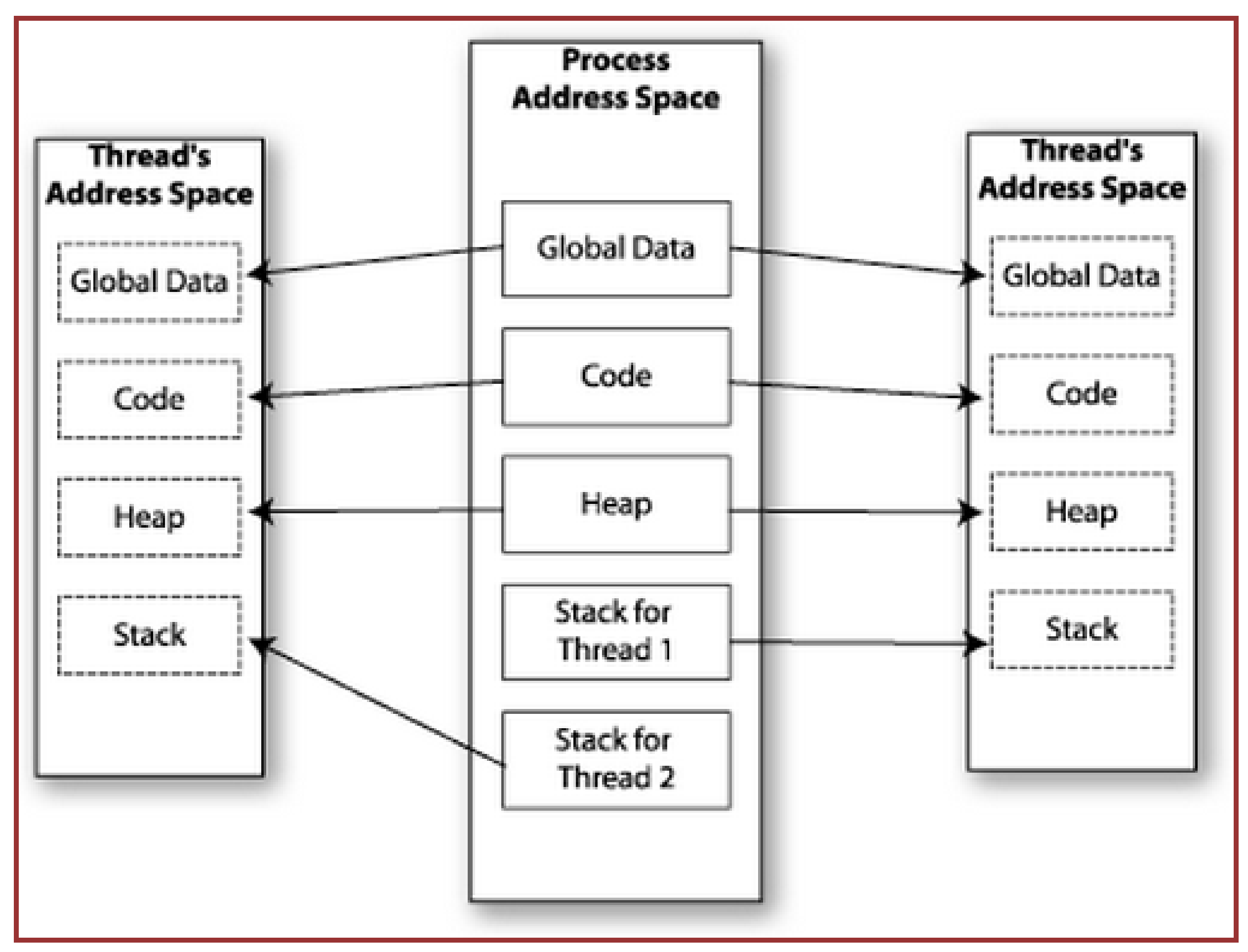
Memory Layout: Single-Threaded Program





Memory Layout: Multithreaded Program



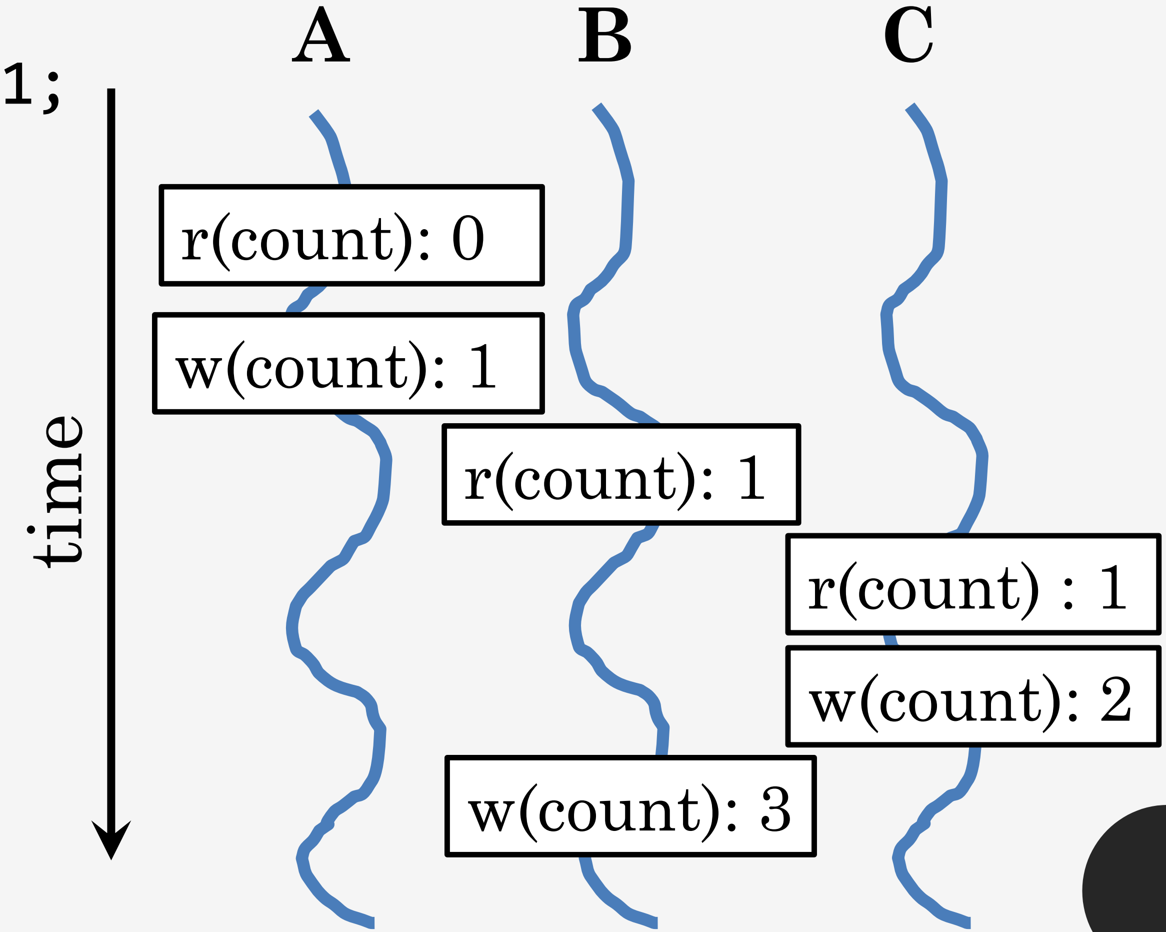


Shared Memory

- Makes multithreaded programming
  - **Powerful**
    - can easily access data and share it among threads
  - **More efficient**
    - No need for system calls when sharing data
    - Thread creation and destruction less expensive than process creation and destruction
  - **Non-trivial**
    - Have to prevent several threads from accessing and changing the same shared data at the same time (synchronization)

Race Condition

```
int count = 0;
void increment()
{
    count = count + 1;
}
```



r(count): 0

w(count): 1

r(count): 1

r(count) : 1

w(count): 2

w(count): 3

Result depends on order of execution

Synchronization needed

Multithreading & Multitasking: A Comparison

Multithreading

- Threads share the same address space
- Light-weight creation/destruction
- Easy inter-thread communication
- An error in one thread can bring down all threads in process

Multitasking

- Processes are insulated from each other
- Expensive creation/destruction
- Expensive IPC (interprocess communication)
- An error in one process cannot bring down another process

Lab 6

- Evaluate the performance of multithreaded sort
- Add /usr/local/cs/bin to PATH
  - `$ export PATH=/usr/local/cs/bin:$PATH`
- Generate a file containing 2^24 random single-precision floating point numbers, one per line with no white space
  - `/dev/urandom: pseudo-random number generator`
- Disk quota exceeded
  - http://www.seasnet.ucla.edu/seasnet-account-quotas/

Lab 6

- `head`
  - Write the first few lines of a file to output
  - Use –N to select number of lines
- `od`
  - Write the contents of its input files to standard output in a user-specified format
  - Options
    - `-t :` select output format
    - `-N <count>:` Format no more than *count* bytes of input
- `sed, tr`
  - Remove address, delete spaces, add newlines between each float

Lab 6

- use `time -p` to time the command `sort -g` on the data you generated
- Send output to */dev/null*
- Run `sort` with the `--parallel` option and the

  `-g` option: compare by general numeric value

  - Use `time` command to record the real, user and system time when running sort with 1, 2, 4, and 8 threads
    - $ `time -p sort -g file_name > /dev/null` (1 thread)
    - $ `time -p sort -g --parallel=[2, 4, or 8] file_name > /dev/null`
  - Record the times and steps in log.txt