Lecture 5…

- Example 4: DAXPY
  - Row major vs Column major storage order
- Example 5: 2d Matrix multiplication
  - Loop interchange; Loop unrolling; Tiling
- More on caches
  - Direct mapped vs Set associative caches
  - Hardware cache block prefetchers
  - Hardware performance counters
- Improved loop performance
  - Vectorization

Assignment 3 (Due 2 Sept 2022)

Matrix Multiplication in C

Program: **The ijk triply nested for loop matrix multiplier from class**. Assume 256x256 float matrices (or square matrix sizes that are larger powers of 2 if feasible)

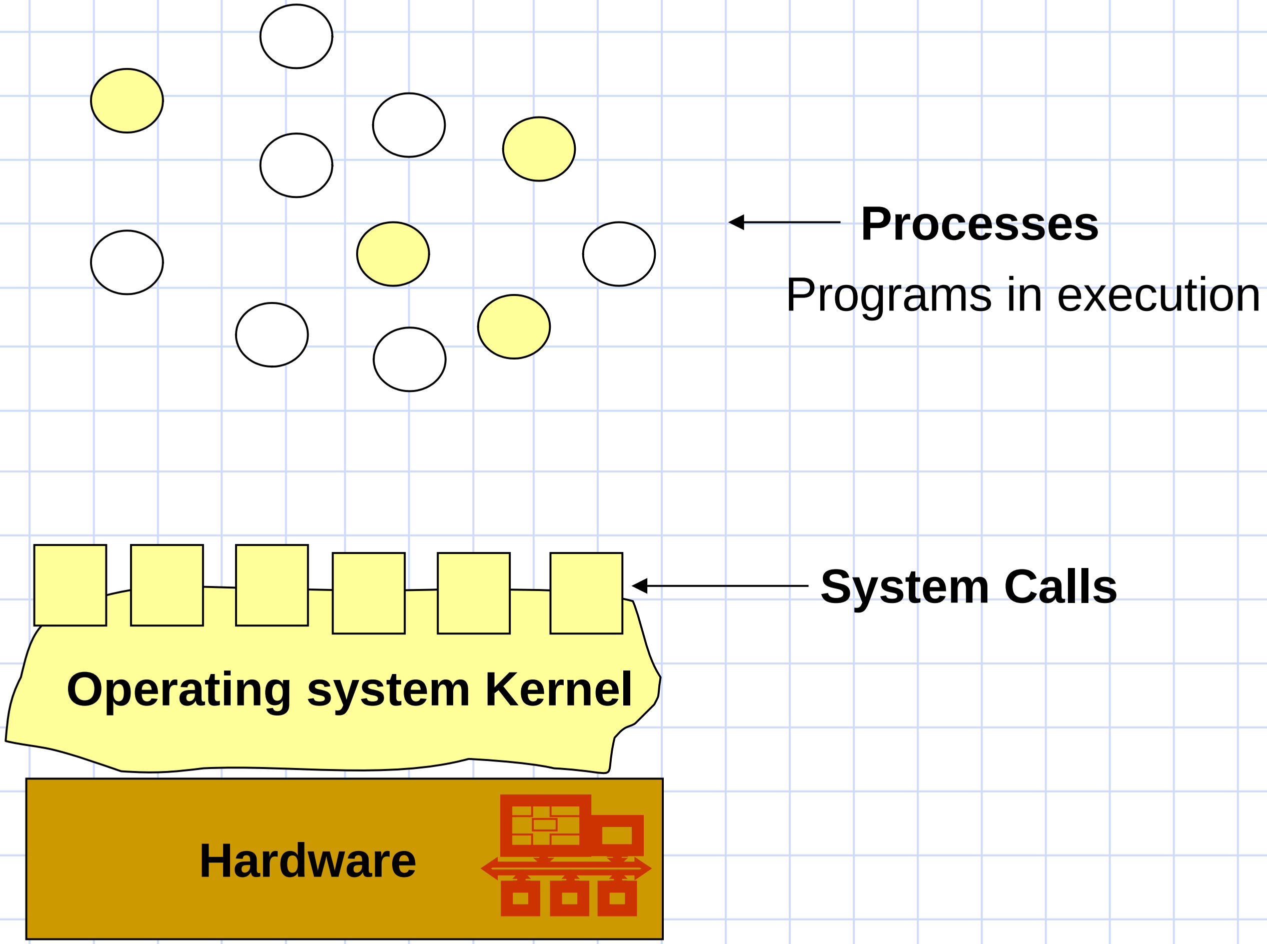
Objective: Find out which, if any, of the loop optimizations that we looked at in class (loop interchange, loop unrolling, loop tiling, vectorization) can be done using the compiler command line options available to you. Determine the combination of command line options that results in the fastest matrix multiplication.

Submit: A typed report (pdf) describing what you tried and what you discovered, including relevant details of the compiler + computer used and measured execution times

Software Organization

- The hardware resources of a computer system must be shared by the programs currently in execution on it
- Operating system: special software that manages this sharing
- Objective: Understand more about the impact of software organization on the execution of your program
  - 1. Main Memory management
  - 2. CPU time management

Hardware, Operating system, Processes



System Calls

- How a process gets the operating system to do something for it; an interface or API to operating system functionality
- When a process is executing in a system call, it is actually executing Operating System code
- Examples (Linux/UNIX)
  - Related to main memory: brk, sbrk, mmap
  - Related to CPU time: wait
  - Related to files: open, close, read, write
  - Related to processes: fork, exec, exit

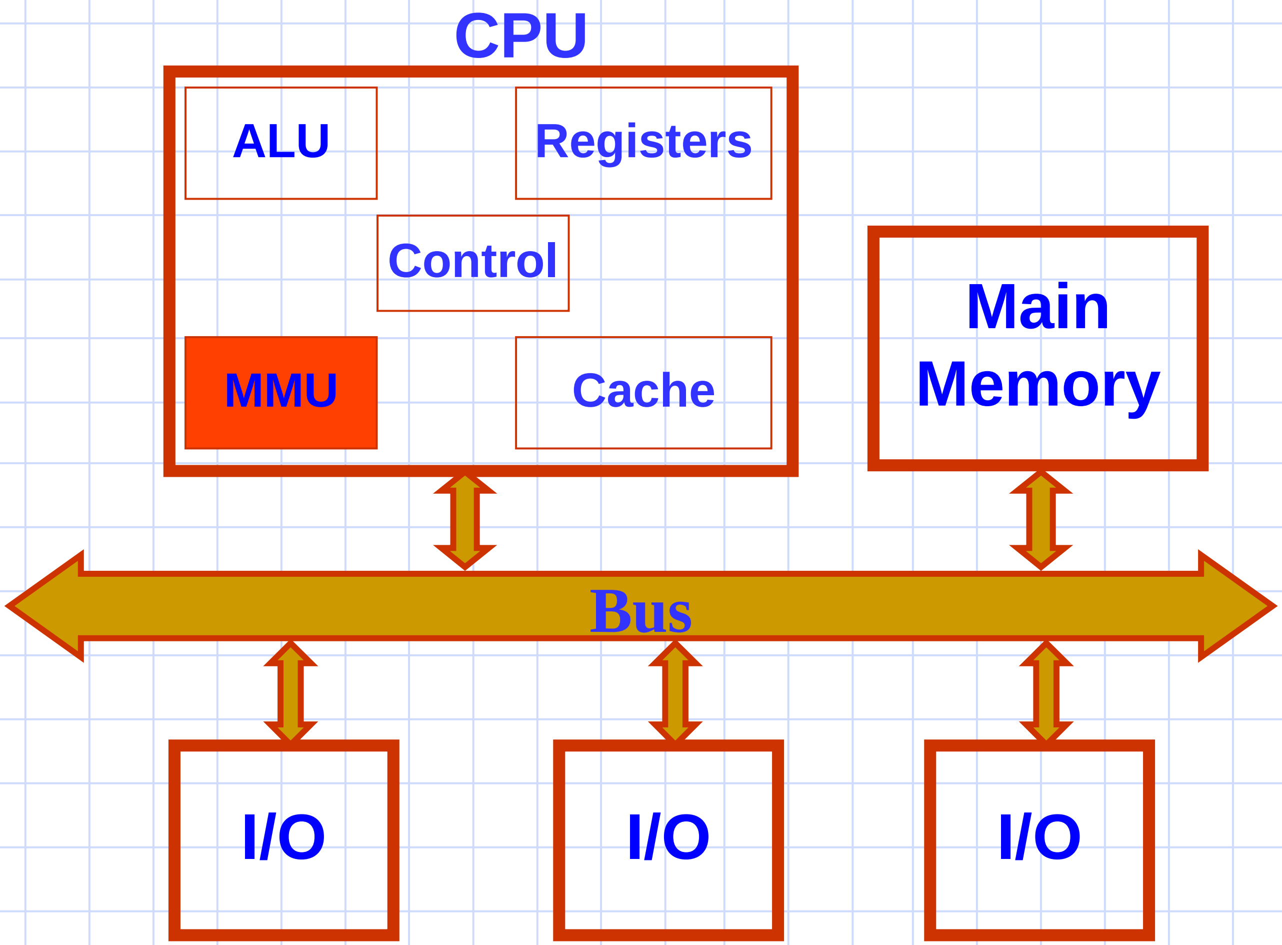
1. Memory Management

- There can be several programs concurrently executing on the machine
- These programs must be protected from each other
  - one program should not be able to access the variables of another
- This is typically done through Address Translation

Idea of Address Translation

- Each program is compiled to use addresses in the range 0 .. MaxAddress (e.g., 0 .. 4G-1)
  - This is its own (personal) notion of what memory looks like
  - So, these addresses are not actual Main Memory addresses; they are called Virtual Addresses
  - The Processor generates such virtual addresses to access instructions or data in Main Memory
- Before an access request from the Processor reaches Main Memory, this virtual address must be translated into an actual Main Memory address
  - Terminology: virtual address, physical address
- Memory Management Unit (MMU): The hardware that does the address translation

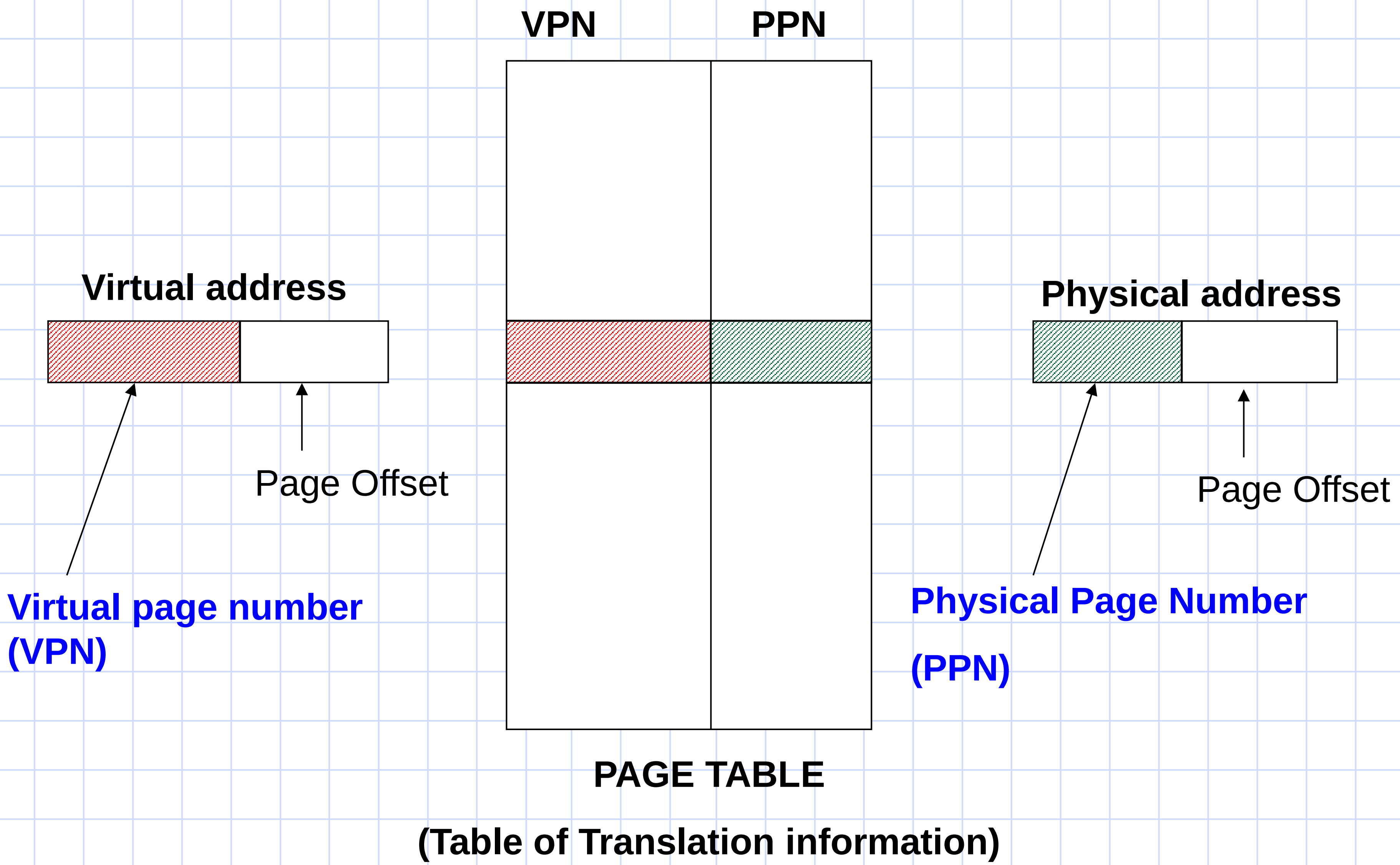
Recall Basic Computer Organization



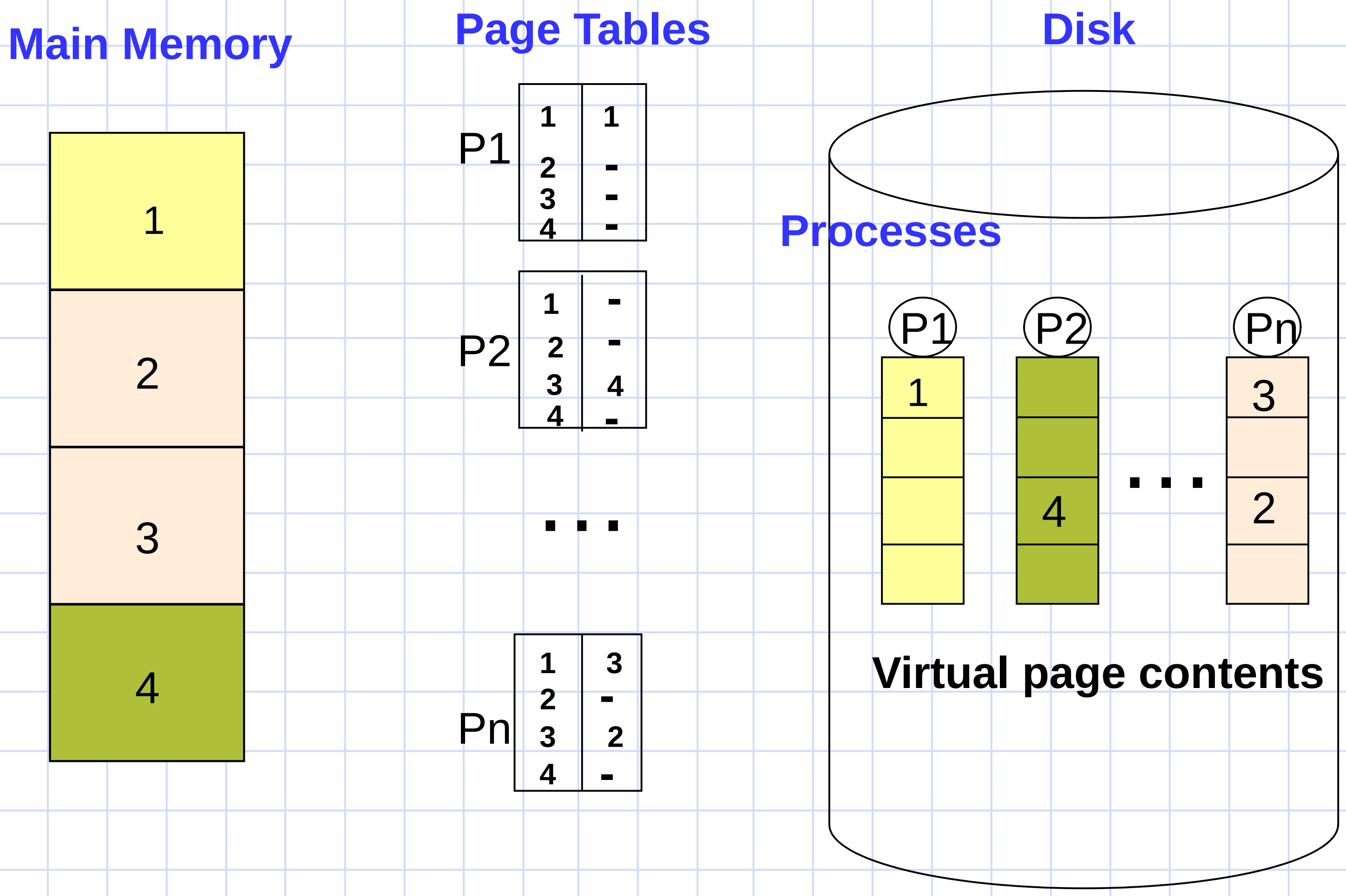
Memory Management

- Example: Paged Virtual Memory
- To translate a virtual address to the corresponding physical (Main Memory) address, a table of translation information is needed
- As with caches, translations are managed not on individual byte basis but at larger granularity
- Page: fixed size unit of memory (contiguous memory locations) for which a single piece of translation information is maintained
  - e.g., 4 KB

Virtual Address Translation


VPN
PPN
Virtual address
Page Offset
Virtual page number (VPN)
Physical address
Page Offset
Physical Page Number (PPN)
PAGE TABLE
(Table of Translation information)

What’s happening…



Impact on our cache analysis?

- The processor does not generate physical (Main Memory) addresses
  - It generates virtual addresses
- L1 caches do cache directory lookup largely using virtual addresses
- Address translation might only be required when there is a cache miss

Steps in Memory Access

- Processor generates a virtual memory address
  - MMU: translates virtual address into a physical memory address
  - Cache: looks it up in cache directory
- The typical data reference will be translated by the MMU and hit in the L1 data cache
- What can go wrong?
  - TLB miss
    - TLB (Translation Lookaside Buffer) is a cache of Page Table entries, used by the MMU
  - Cache miss
  - Page fault

2. CPU Time Management

- There can be several programs concurrently executing on the machine
  - Process: “program in execution”
- The Operating System manages the CPU, allowing one process to run on the CPU at a time
  - OS keeps track of information about all processes
  - It uses this information to switch them into and out of the CPU

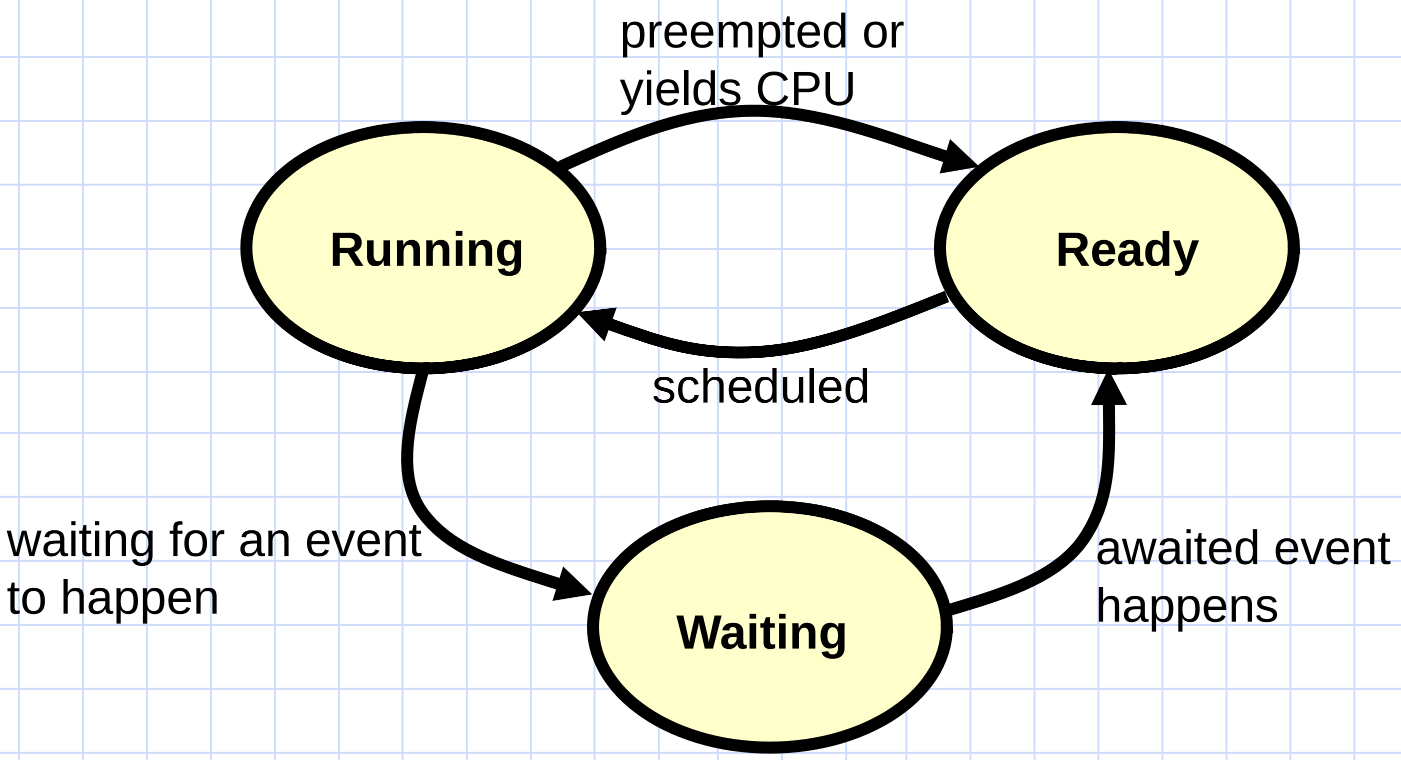
Information about a process

- Instructions and Data
  - Text, data, stack, heap
- Hardware information
  - Register values
- Other information
  - Process id, parent id, user id
  - CPU time used by process so far
  - Memory management info: Page table
  - File related info: Open files, file pointers

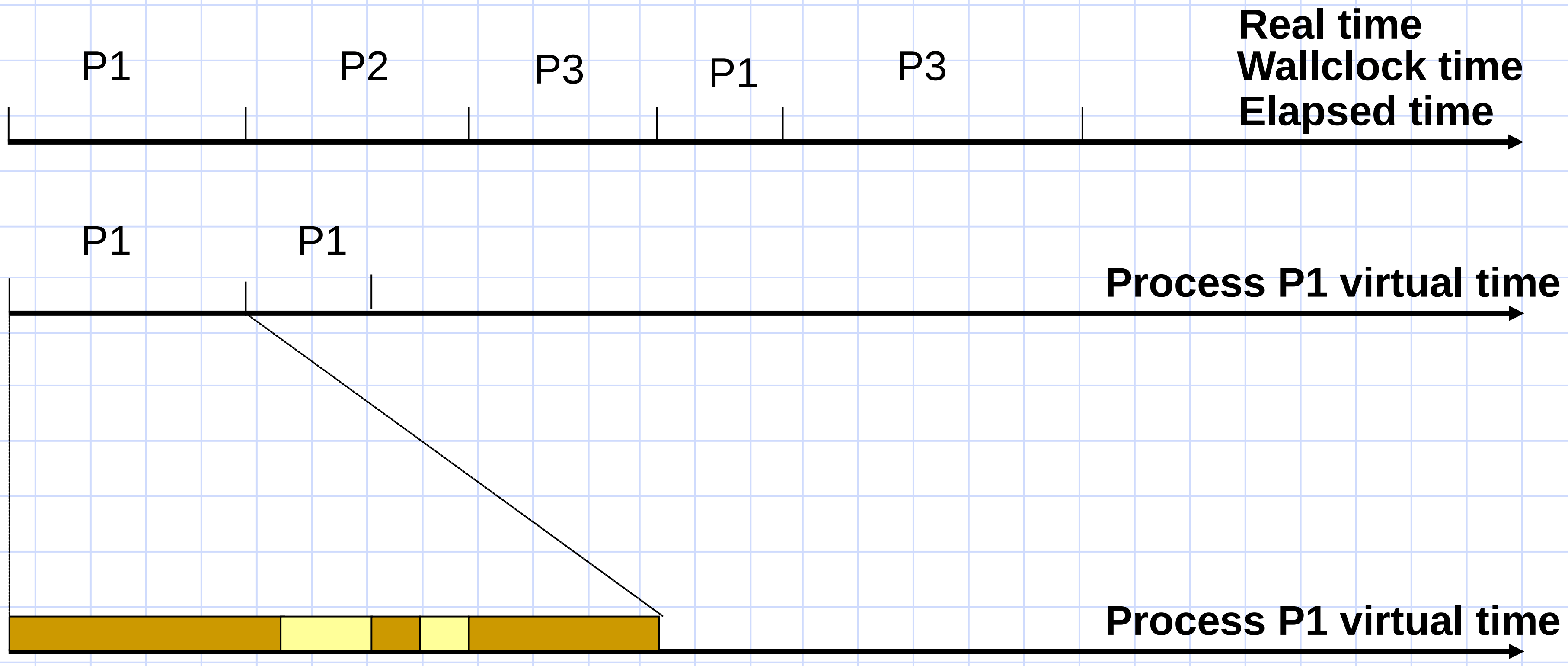
Process Management

- What should OS do when a process does something that will involve a long wait?
  - e.g., file read/write operation, page fault
  - Make another process run on the CPU
- Which process?
  - Decided by OS process scheduling policy
    - Possible objectives: Minimize average program execution time; Fairness

Process State Transition Diagram



Timelines



🟧 : Running in User Mode: Executing instructions of the user program

🟨 : Running in System Mode: Executing instructions of OS related activity

Timing

Timing: measuring the time spent in specific parts of your program

- Examples of `parts': Functions, loops, …
- Recall: Different kinds of time that can be measured (real/wallclock/elapsed vs virtual/CPU)

1. Decide
   - which time you are interested in measuring
   - at what granularity
2. Find out what mechanisms are available and their granularity of measurement

time command

Usage: % time a.out

Example: % time ls

```
0.00user  0.002sys  0:0.003elapsed
```

CPU time in user mode, CPU time in system mode, Elapsed time

Example: % time man csh

```
0.268user  0.032sys  0:15.486elapsed
```

gettimeofday( )

Reports elapsed time since
00:00 GMT 1 January 1970

```
#include <sys/time.h>

int gettimeofday(struct timeval *tv, struct timezone *tz);

struct timeval {
    long tv_sec;   /* seconds */
    long tv_usec; /* microseconds */
};
```

Usage: Insert calls to gettimeofday in your C program

Using gettimeofday( )

Your C program

```
struct timeval before, after;

gettimeofday(&before);
          /
          / region of program you want to time
          /
gettimeofday(&after);

printf (“%d\n”, after.tv_sec – before.tv_sec );
```

High resolution, real timers

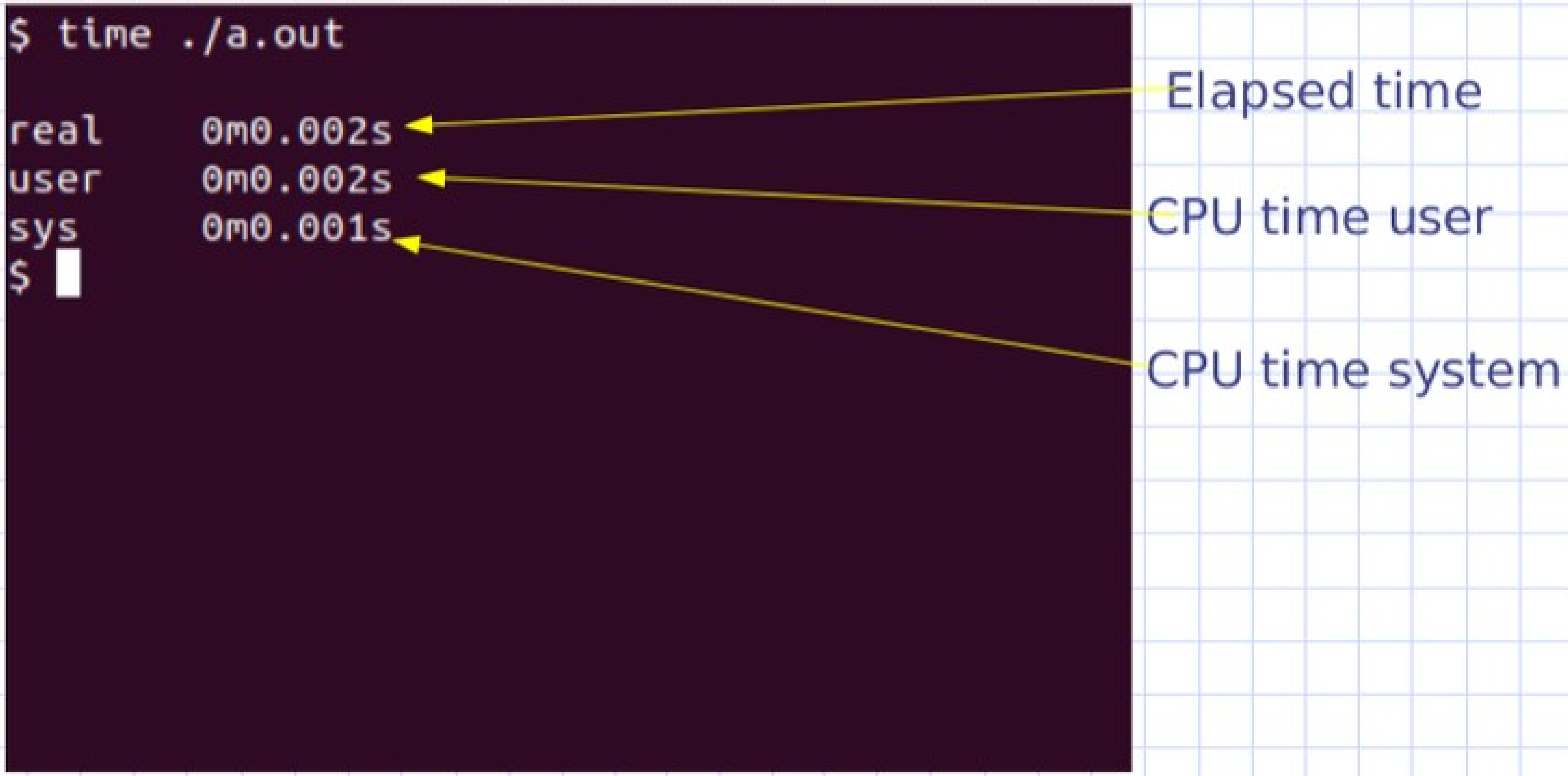
- Most modern processors provide a hardware cycle counting mechanism
  1. A special purpose register that is incremented every clock cycle
  2. An instruction to read the value in that register
- Example: Intel® time stamp counter and rdtsc instruction

Issues in Measuring Execution Time

- Elapsed time ?? CPU time
- CPU time user ?? CPU time user + system
- Resolution? sec/msec/usec/nsec
- Timing mechanism? time/cron/...
- Experimental conditions
- Repetitions

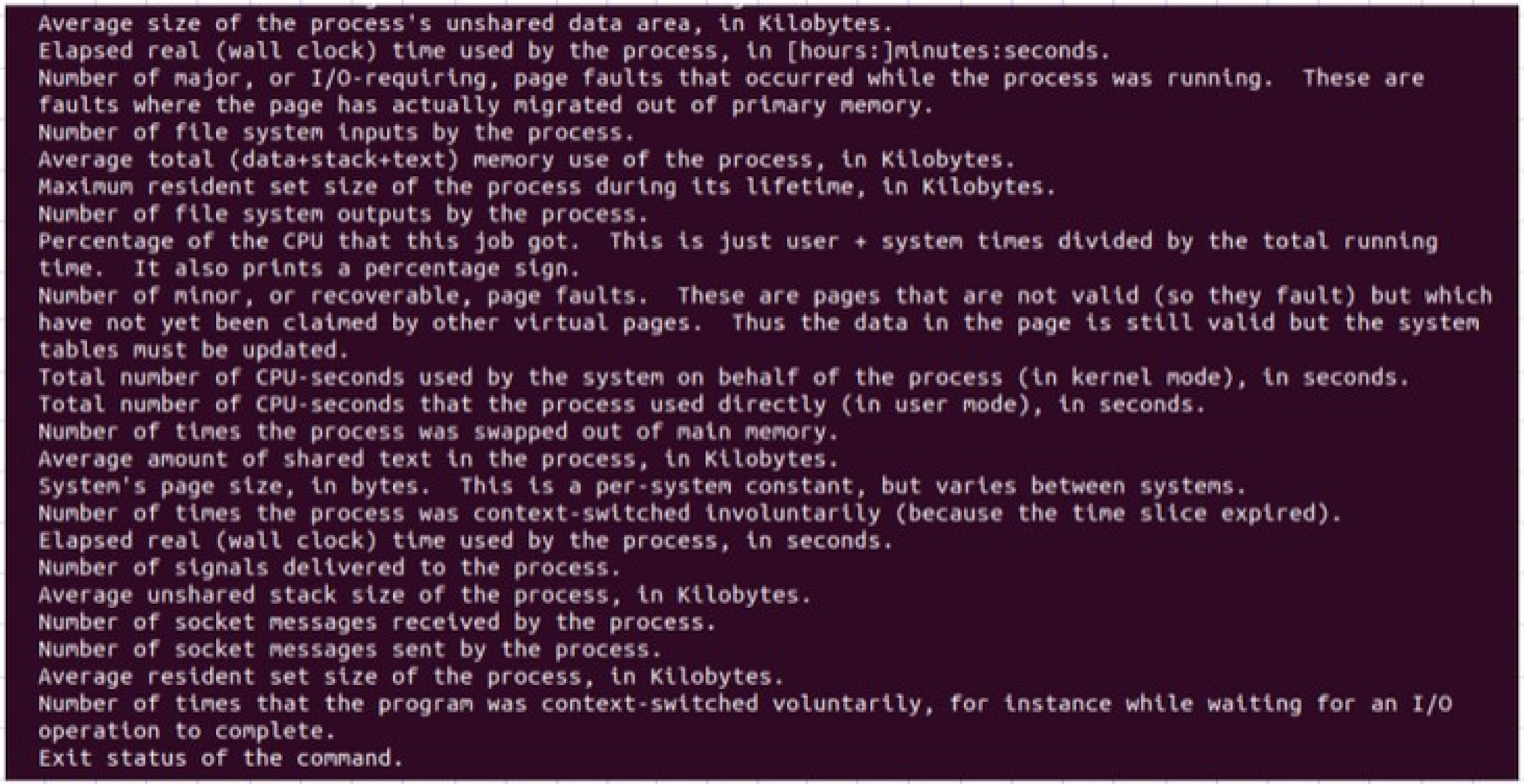
time: Command execution

$ time ./a.out



```
$ time ./a.out

real	0m0.002s
user	0m0.002s
sys	0m0.001s
$ 
```

Linux manual: time (1)

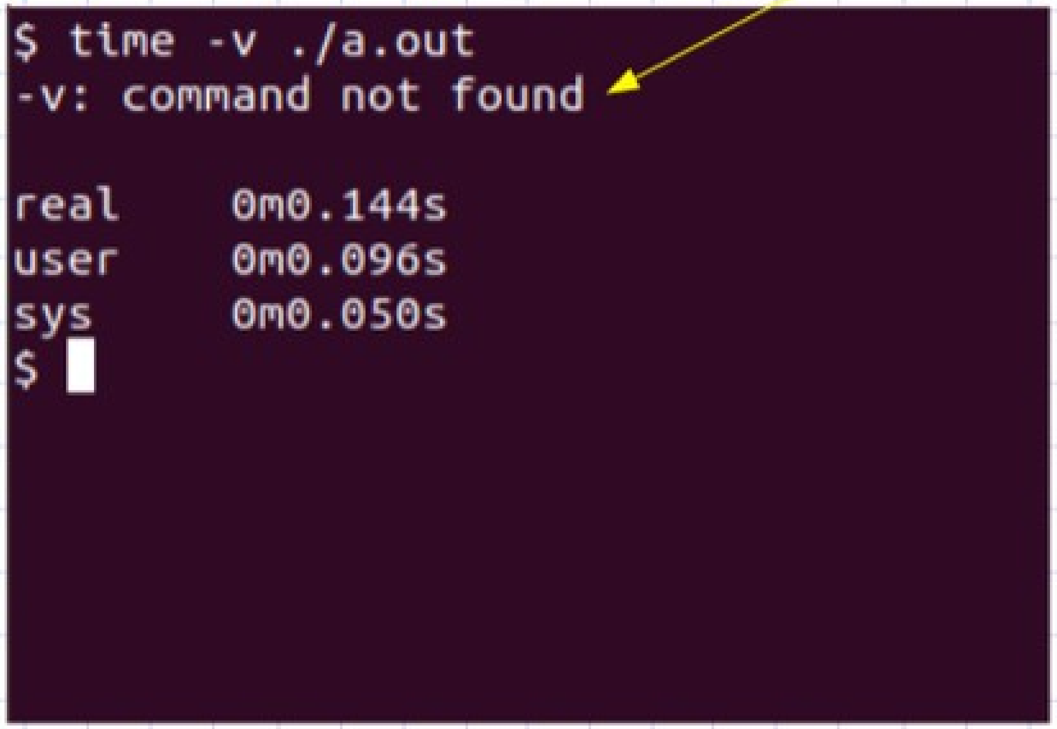


```
Average size of the process's unshared data area, in Kilobytes.
Elapsed real (wall clock) time used by the process, in [hours:]minutes:seconds.
Number of major, or I/O-requiring, page faults that occurred while the process was running.  These are
faults where the page has actually migrated out of primary memory.
Number of file system inputs by the process.
Average total (data+stack+text) memory use of the process, in Kilobytes.
Maximum resident set size of the process during its lifetime, in Kilobytes.
Number of file system outputs by the process.
Percentage of the CPU that this job got.  This is just user + system times divided by the total running
time.  It also prints a percentage sign.
Number of minor, or recoverable, page faults.  These are pages that are not valid (so they fault) but which
have not yet been claimed by other virtual pages.  Thus the data in the page is still valid but the system
tables must be updated.
Total number of CPU-seconds used by the system on behalf of the process (in kernel mode), in seconds.
Total number of CPU-seconds that the process used directly (in user mode), in seconds.
Number of times the process was swapped out of main memory.
Average amount of shared text in the process, in Kilobytes.
System's page size, in bytes.  This is a per-system constant, but varies between systems.
Number of times the process was context-switched involuntarily (because the time slice expired).
Elapsed real (wall clock) time used by the process, in seconds.
Number of signals delivered to the process.
Average unshared stack size of the process, in Kilobytes.
Number of socket messages received by the process.
Number of socket messages sent by the process.
Average resident set size of the process, in Kilobytes.
Number of times that the program was context-switched voluntarily, for instance while waiting for an I/O
operation to complete.
Exit status of the command.
```

time -v: Command execution

$ time -v ./a.out

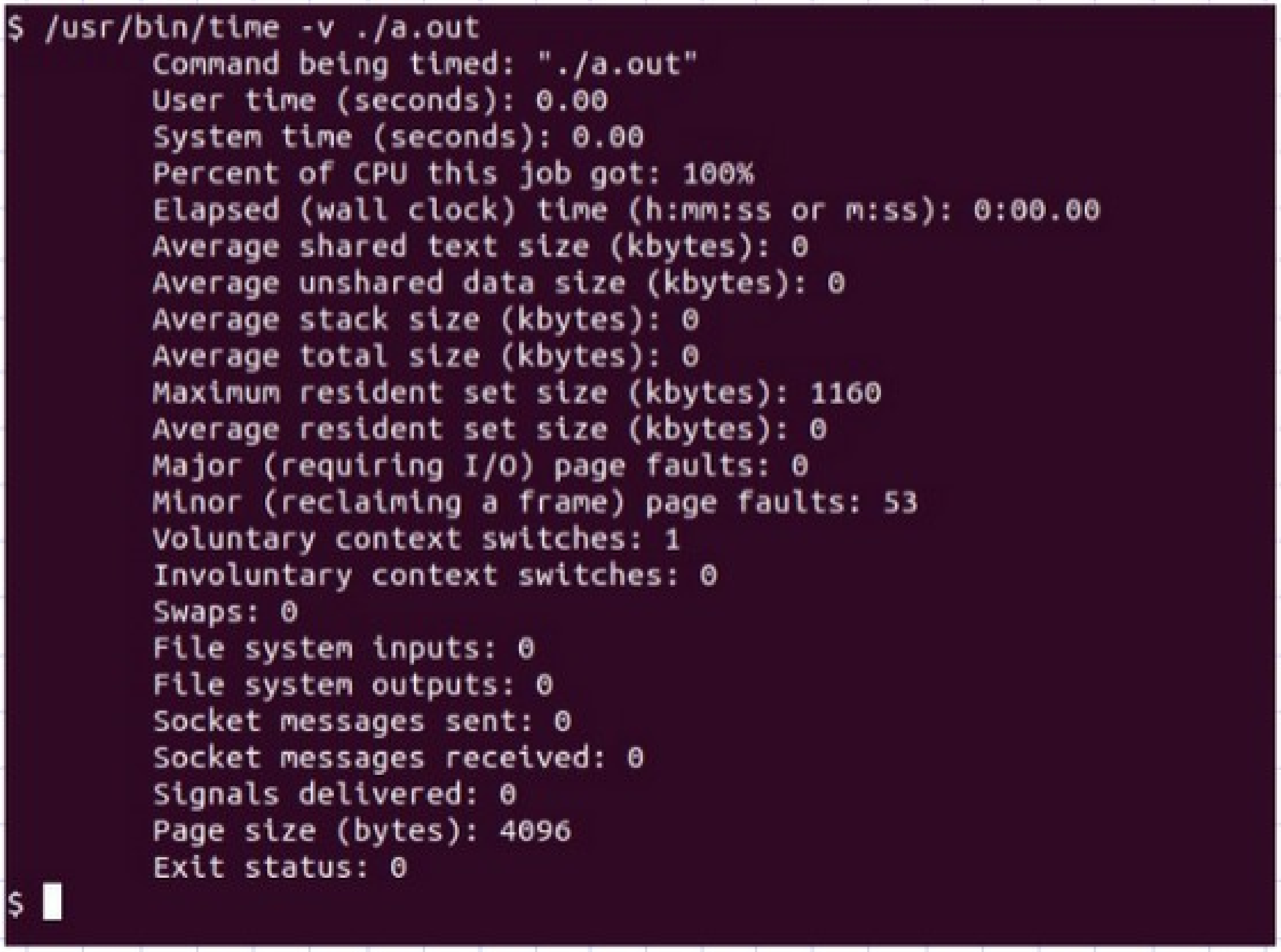
Why? man bash



```
$ time -v ./a.out
-v: command not found

real	0m0.144s
user	0m0.096s
sys	0m0.050s
$ 
```

time (1): Command execution

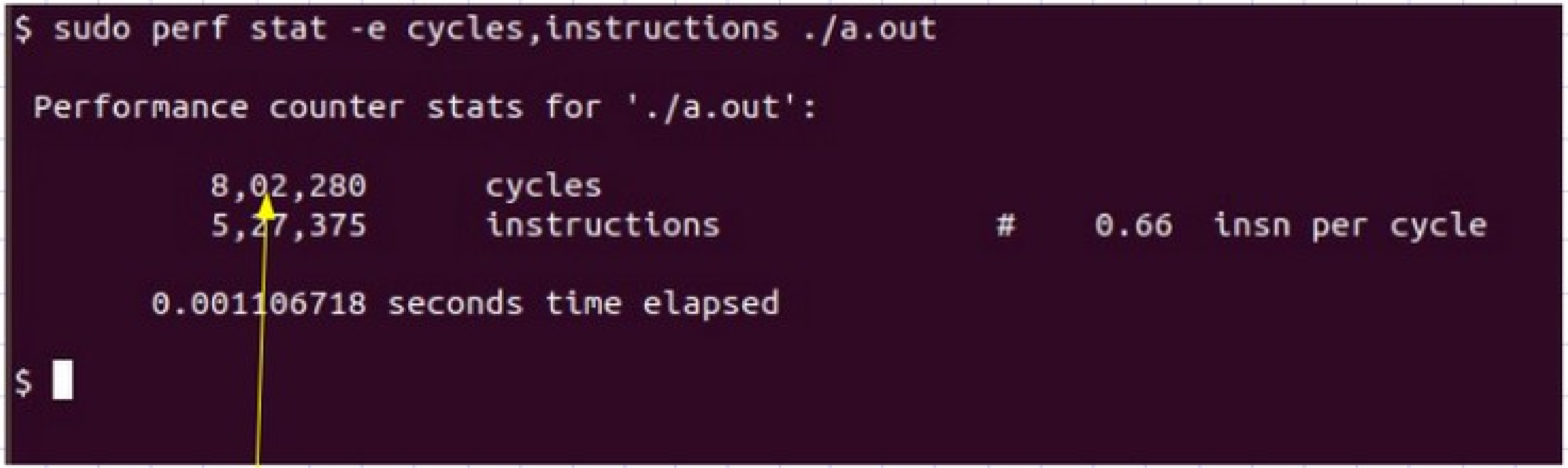
$ /usr/bin/time -v ./a.out



```
$ /usr/bin/time -v ./a.out
        Command being timed: "./a.out"
        User time (seconds): 0.00
        System time (seconds): 0.00
        Percent of CPU this job got: 100%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 1160
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 53
        Voluntary context switches: 1
        Involuntary context switches: 0
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
$
```

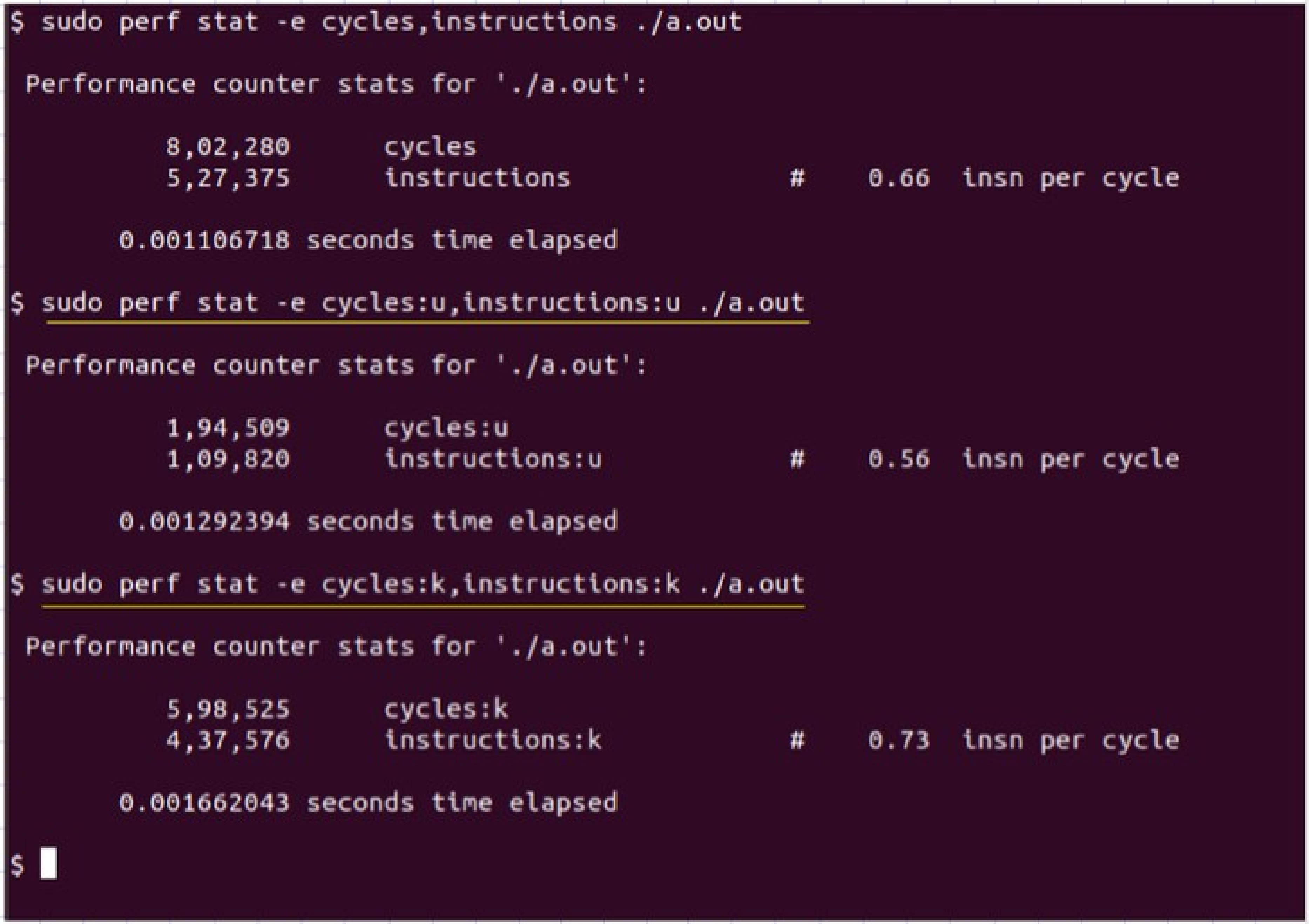
perf: Command execution

$ sudo perf stat -e cycles ./a.out



```
$ sudo perf stat -e cycles,instructions ./a.out

 Performance counter stats for './a.out':

         8,02,280      cycles
         5,27,375      instructions              #    0.66  insn per cycle

      0.001106718 seconds time elapsed

$ 
```

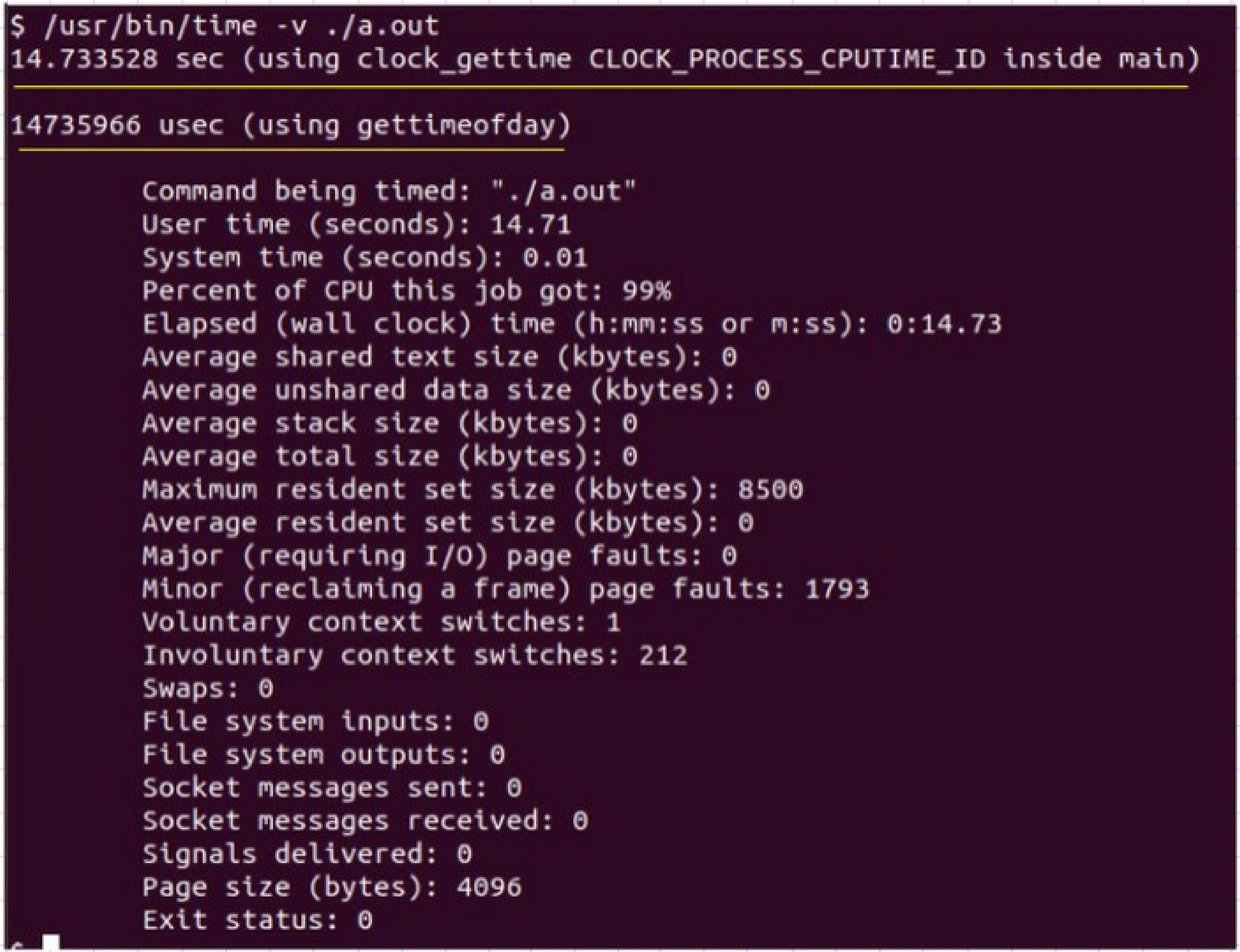
At 2.4GHz, 8,02,280 cycles is 0.00033 sec



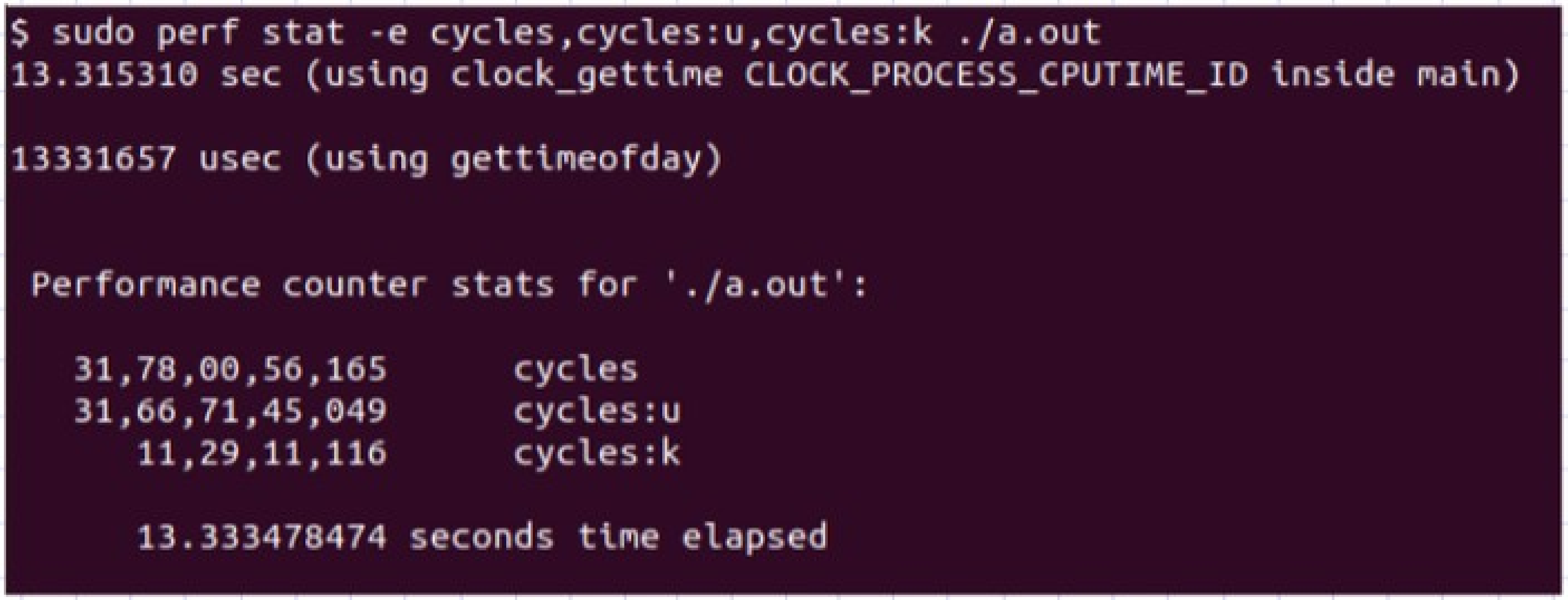
```
$ sudo perf stat -e cycles,instructions ./a.out

 Performance counter stats for './a.out':

          8,02,280      cycles
          5,27,375      instructions                     #    0.66  insn per cycle

       0.001106718 seconds time elapsed

$ sudo perf stat -e cycles:u,instructions:u ./a.out

 Performance counter stats for './a.out':

          1,94,509      cycles:u
          1,09,820      instructions:u                   #    0.56  insn per cycle

       0.001292394 seconds time elapsed

$ sudo perf stat -e cycles:k,instructions:k ./a.out

 Performance counter stats for './a.out':

          5,98,525      cycles:k
          4,37,576      instructions:k                   #    0.73  insn per cycle

       0.001662043 seconds time elapsed

$ 
```

Timing part of a program

- Example: Only the time for doing matrix multiplication
  - Initialize operand matrices
  - Do multiplication using unoptimized triple loop
  - Do multiplication using your faster method
  - Compare the 2 outputs and certify correctness
- Linux:
  - clock_gettime
  - gettimeofday

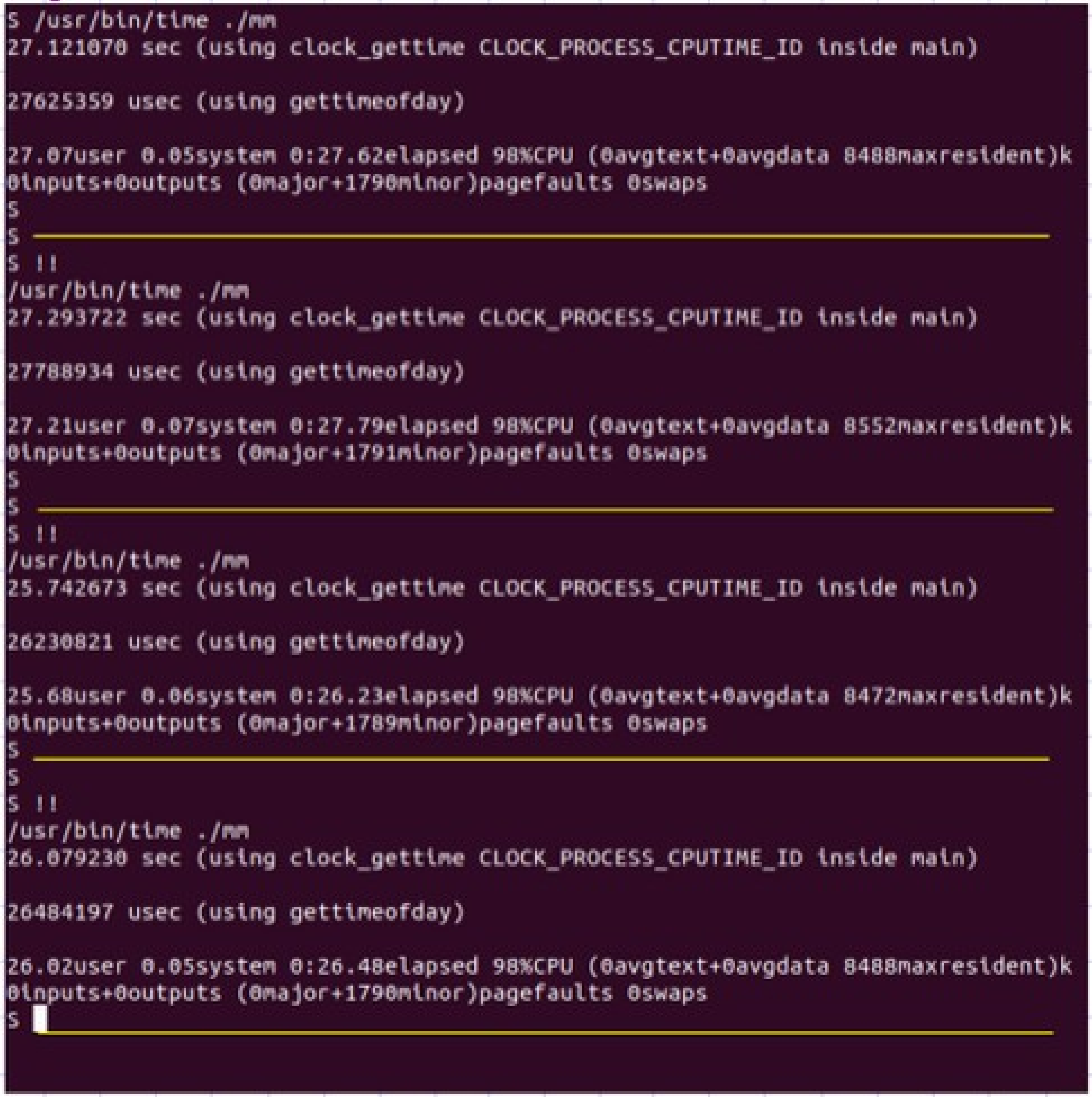


```
$ /usr/bin/time -v ./a.out
14.733528 sec (using clock_gettime CLOCK_PROCESS_CPUTIME_ID inside main)

14735966 usec (using gettimeofday)

        Command being timed: "./a.out"
        User time (seconds): 14.71
        System time (seconds): 0.01
        Percent of CPU this job got: 99%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:14.73
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 8500
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 1793
        Voluntary context switches: 1
        Involuntary context switches: 212
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
$
```



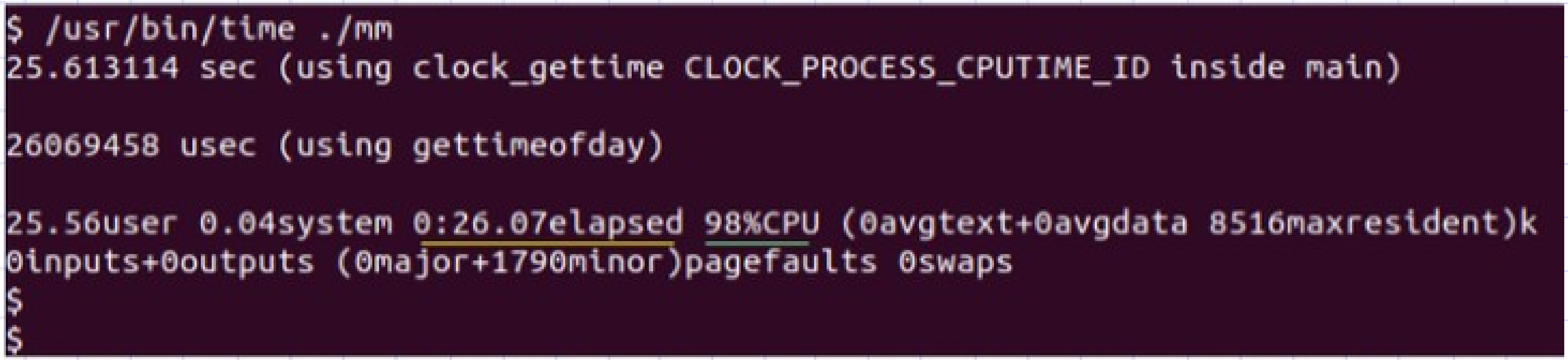
```
$ sudo perf stat -e cycles,cycles:u,cycles:k ./a.out
13.315310 sec (using clock_gettime CLOCK_PROCESS_CPUTIME_ID inside main)

13331657 usec (using gettimeofday)


 Performance counter stats for './a.out':

   31,78,00,56,165      cycles
   31,66,71,45,049      cycles:u
      11,29,11,116      cycles:k

      13.333478474 seconds time elapsed
```

Repetitions



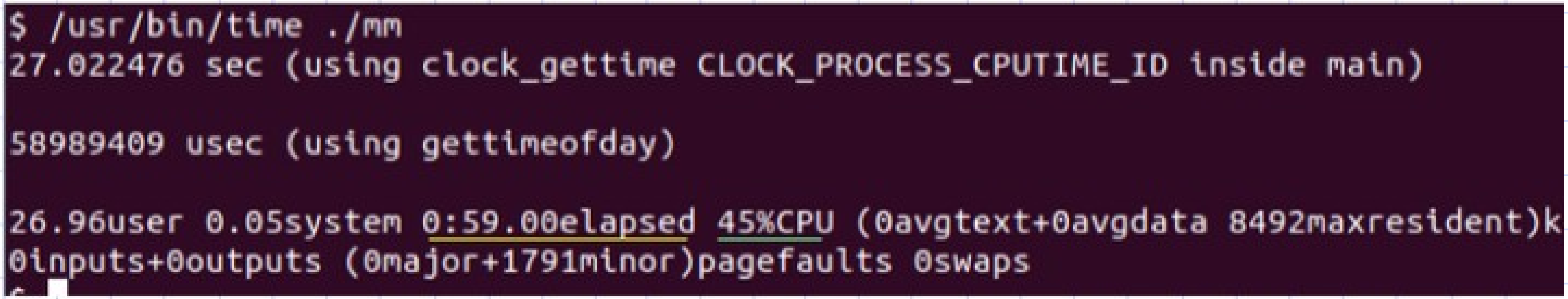
```
$ /usr/bin/time ./mm
27.121070 sec (using clock_gettime CLOCK_PROCESS_CPUTIME_ID inside main)

27625359 usec (using gettimeofday)

27.07user 0.05system 0:27.62elapsed 98%CPU (0avgtext+0avgdata 8488maxresident)k
0inputs+0outputs (0major+1790minor)pagefaults 0swaps
$
$
$ !!
/usr/bin/time ./mm
27.293722 sec (using clock_gettime CLOCK_PROCESS_CPUTIME_ID inside main)

27788934 usec (using gettimeofday)

27.21user 0.07system 0:27.79elapsed 98%CPU (0avgtext+0avgdata 8552maxresident)k
0inputs+0outputs (0major+1791minor)pagefaults 0swaps
$
$
$ !!
/usr/bin/time ./mm
25.742673 sec (using clock_gettime CLOCK_PROCESS_CPUTIME_ID inside main)

26230821 usec (using gettimeofday)

25.68user 0.06system 0:26.23elapsed 98%CPU (0avgtext+0avgdata 8472maxresident)k
0inputs+0outputs (0major+1789minor)pagefaults 0swaps
$
$
$ !!
/usr/bin/time ./mm
26.079230 sec (using clock_gettime CLOCK_PROCESS_CPUTIME_ID inside main)

26484197 usec (using gettimeofday)

26.02user 0.05system 0:26.48elapsed 98%CPU (0avgtext+0avgdata 8488maxresident)k
0inputs+0outputs (0major+1790minor)pagefaults 0swaps
$
```

Measurement conditions



```
$ /usr/bin/time ./mm
25.613114 sec (using clock_gettime CLOCK_PROCESS_CPUTIME_ID inside main)

26069458 usec (using gettimeofday)

25.56user 0.04system 0:26.07elapsed 98%CPU (0avgtext+0avgdata 8516maxresident)k
0inputs+0outputs (0major+1790minor)pagefaults 0swaps
$
$
```

With 3 other programs running



```
$ /usr/bin/time ./mm
27.022476 sec (using clock_gettime CLOCK_PROCESS_CPUTIME_ID inside main)

58989409 usec (using gettimeofday)

26.96user 0.05system 0:59.00elapsed 45%CPU (0avgtext+0avgdata 8492maxresident)k
0inputs+0outputs (0major+1791minor)pagefaults 0swaps
$
```

Guidelines

- Decide Elapsed time / CPU time
- Decide CPU time user / CPU time user+system
- Decide resolution (nsec, usec, msec, sec, ..)
- Decide timing mechanism (time, chrone, ..)
- Measurement conditions: as light a load as possible
- Repetitions: a few, under similar measurement conditions, as far as possible
  - report average