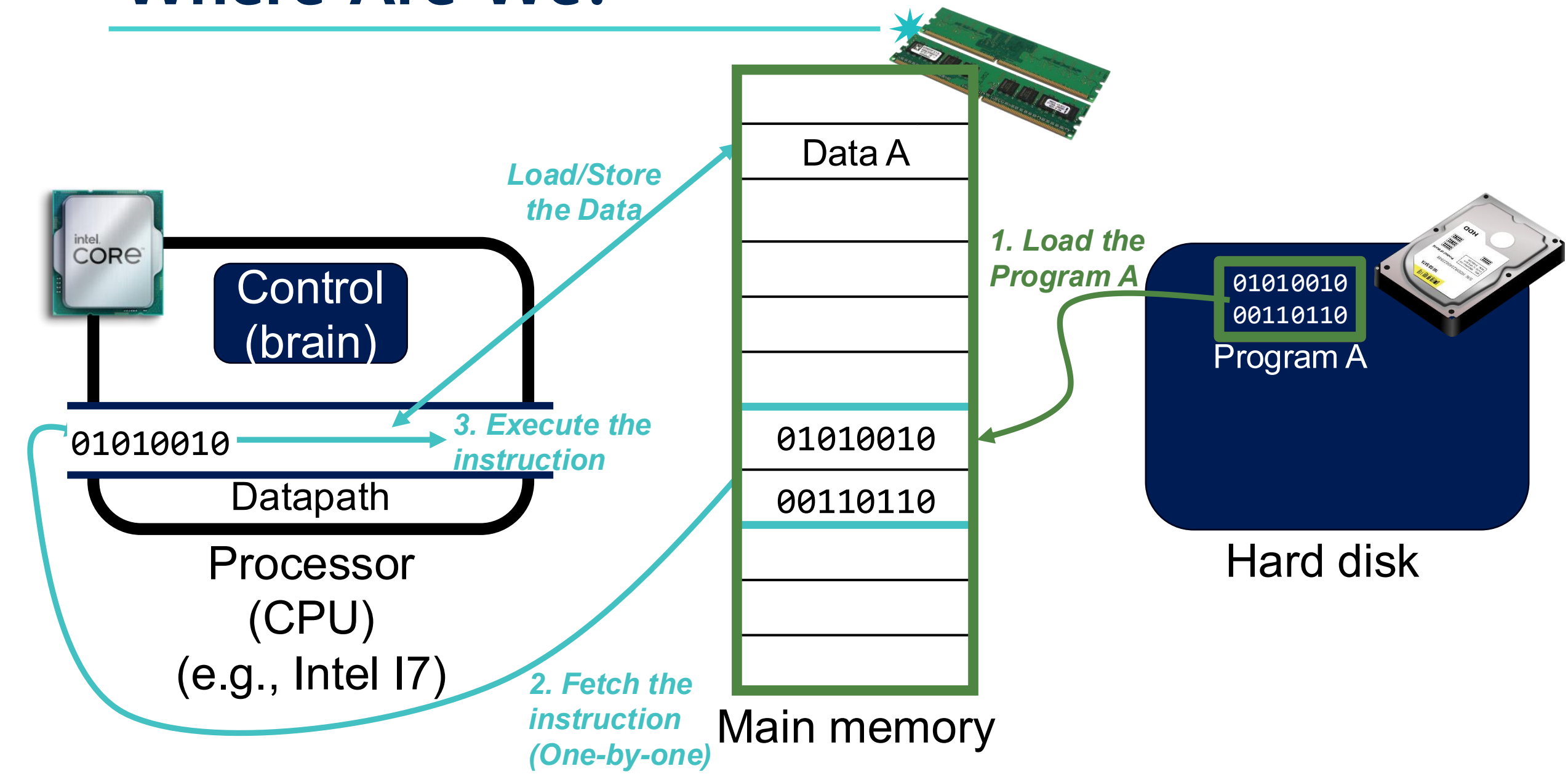
CSE251: System Programming

21. Process (1)

Seongil Wi

Department of Computer Science and Engineering

Where Are We?


Load/Store
the Data
Data A
1. Load the
Program A
01010010
00110110
Program A
Control
(brain)
01010010
3. Execute the
instruction
Datapath
01010010
00110110
Processor
(CPU)
(e.g., Intel I7)
2. Fetch the
instruction
(One-by-one)
Main memory
Hard disk

Where Are We?

How data (bit, byte, int, …) is represented in memory?

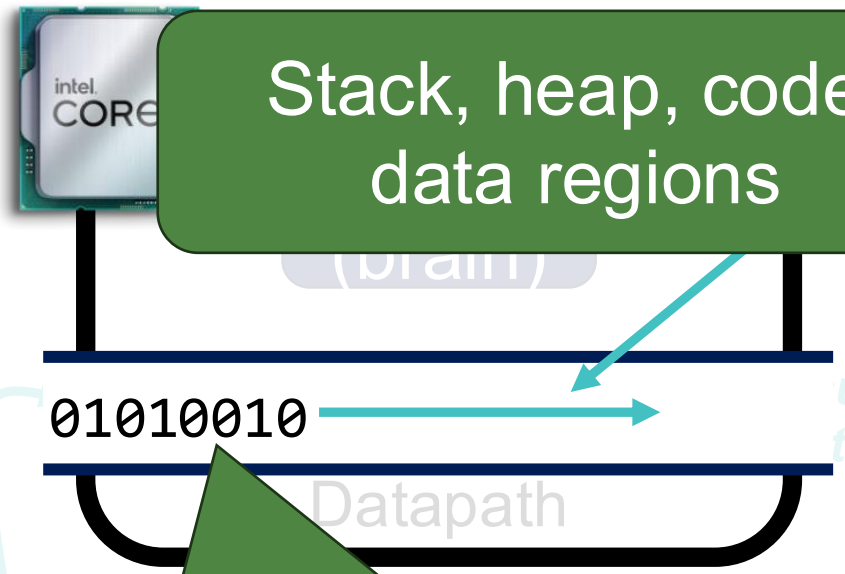
Stack, heap, code, data regions

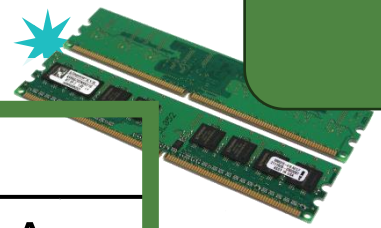
How is assembly code interpreted, and how does it interact with memory?

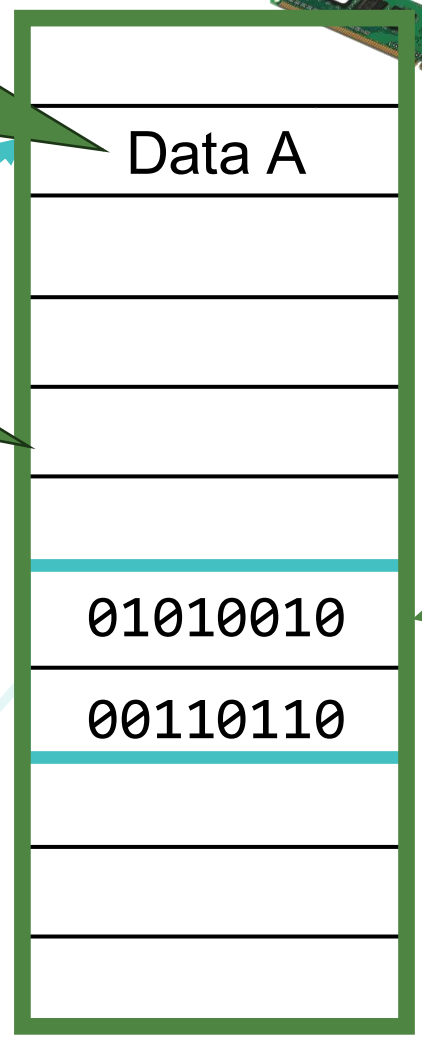
Executable and Linkable Format (ELF), Linking Process

Loading process via loader (kernel)

Memory hierachy (cache, virtual memory)





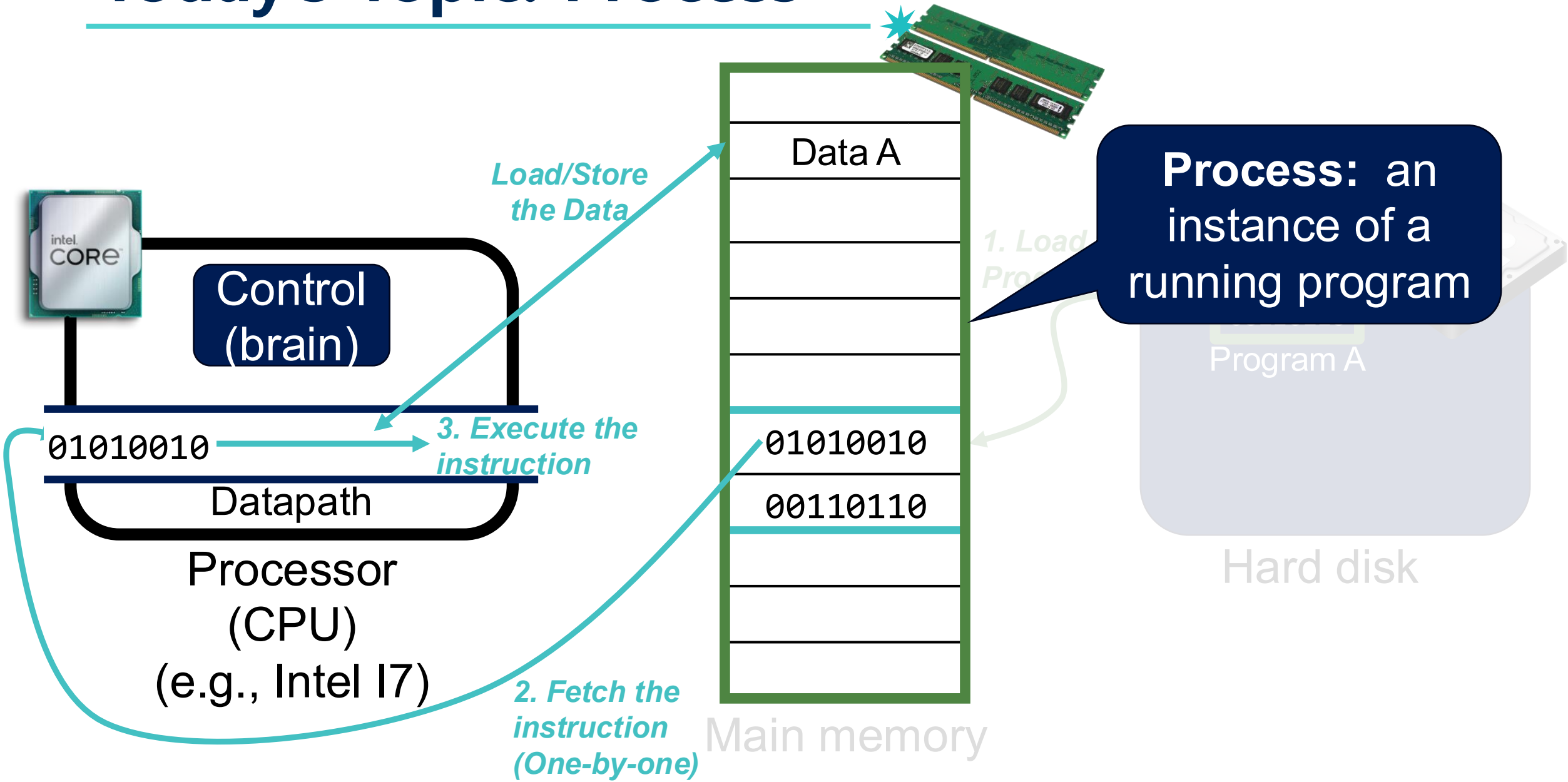


Main memory

1. Load the Program A

Hard disk

Today's Topic: Process



Processes

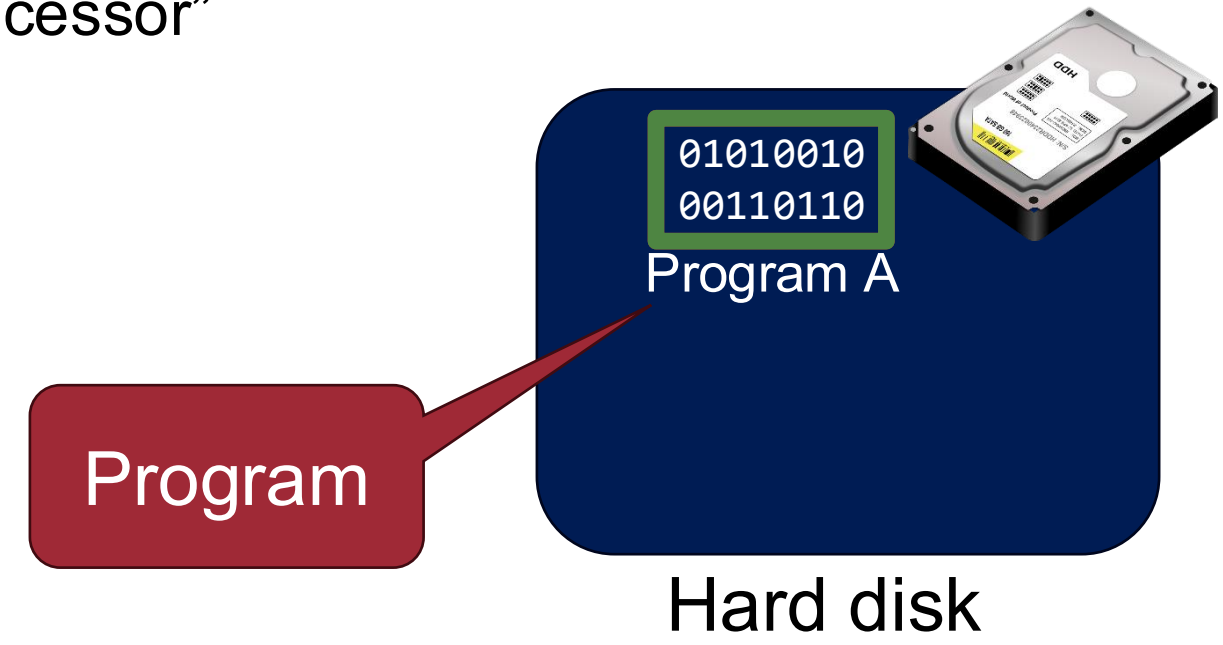
Process

An **instance of a program** that is being executed (or is ready for execution)

Process

An **instance of a program** that is being executed (or is ready for execution)

- Not the same as “program” or “processor”

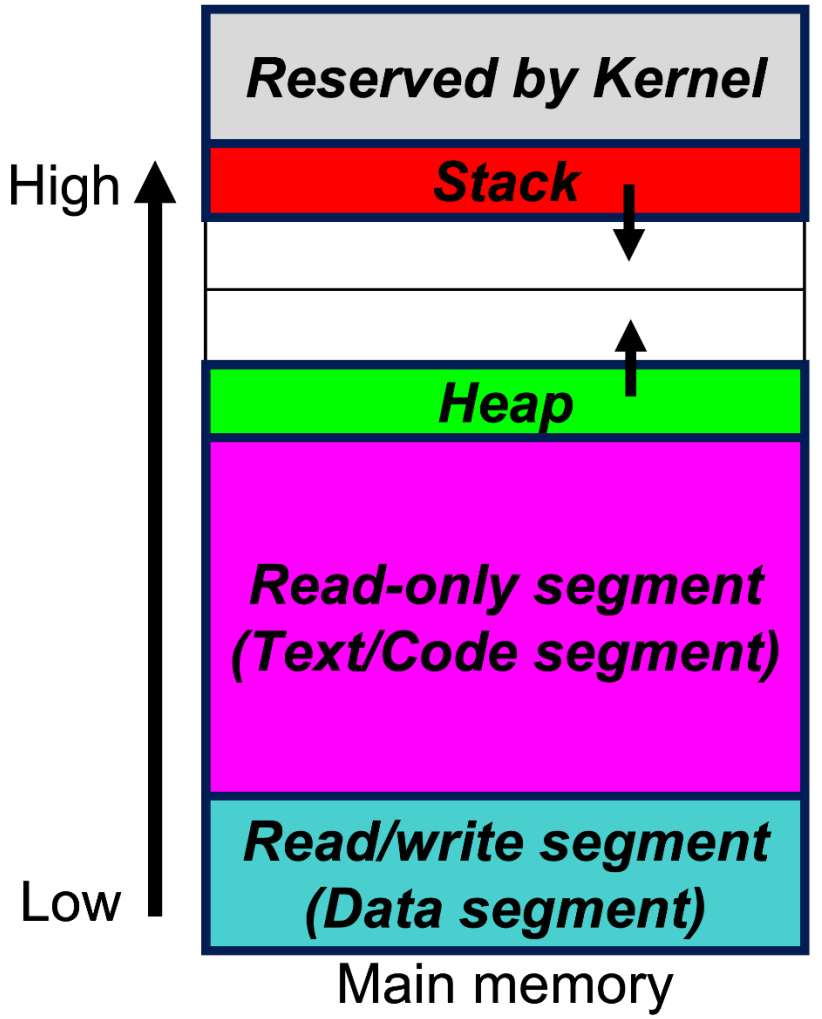


Process

An **instance of a program** that is being executed (or is ready for execution)

- Not the same as “program” or “processor”

- Consists of:
  - Memory (code, heap, stack, etc.)

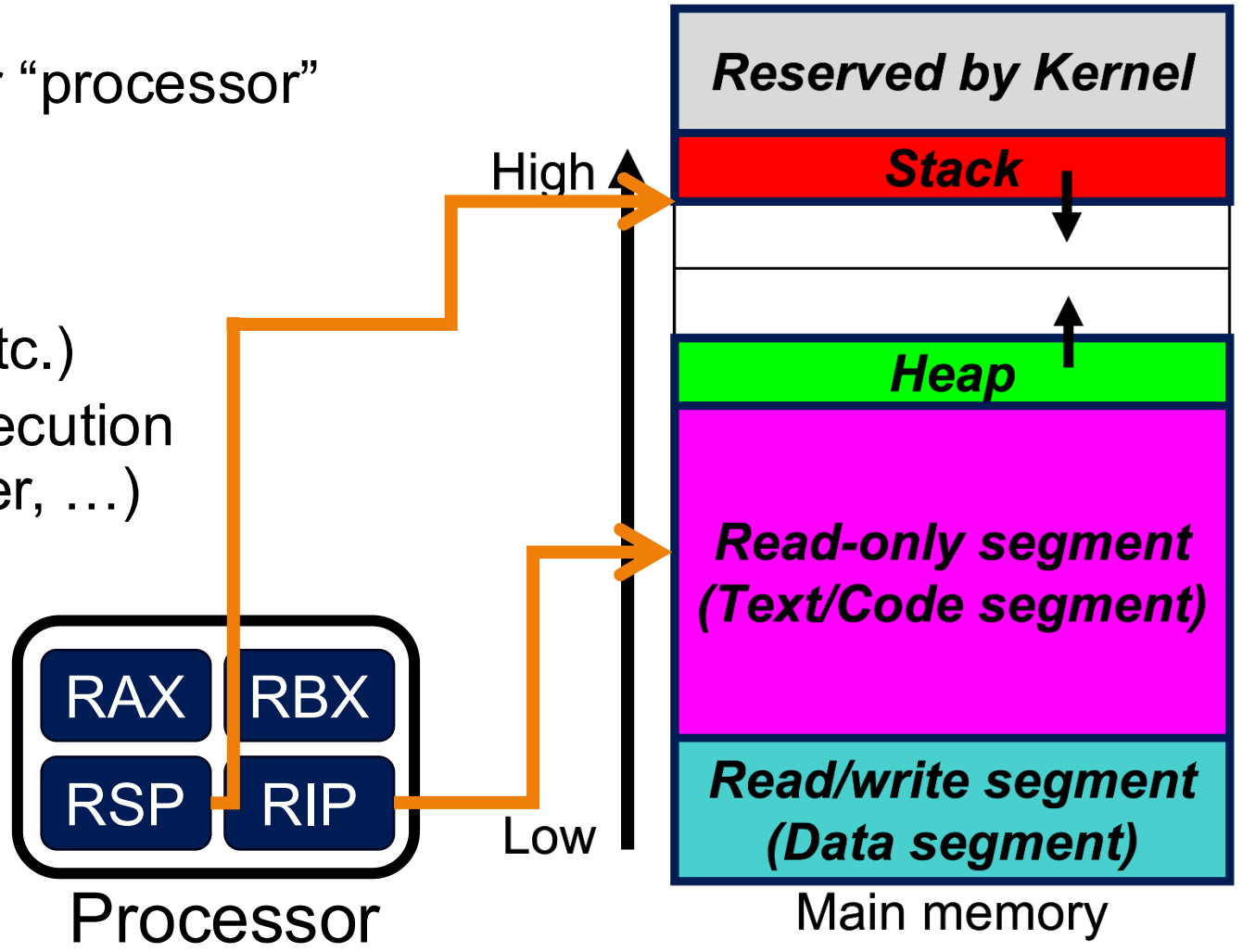


Process

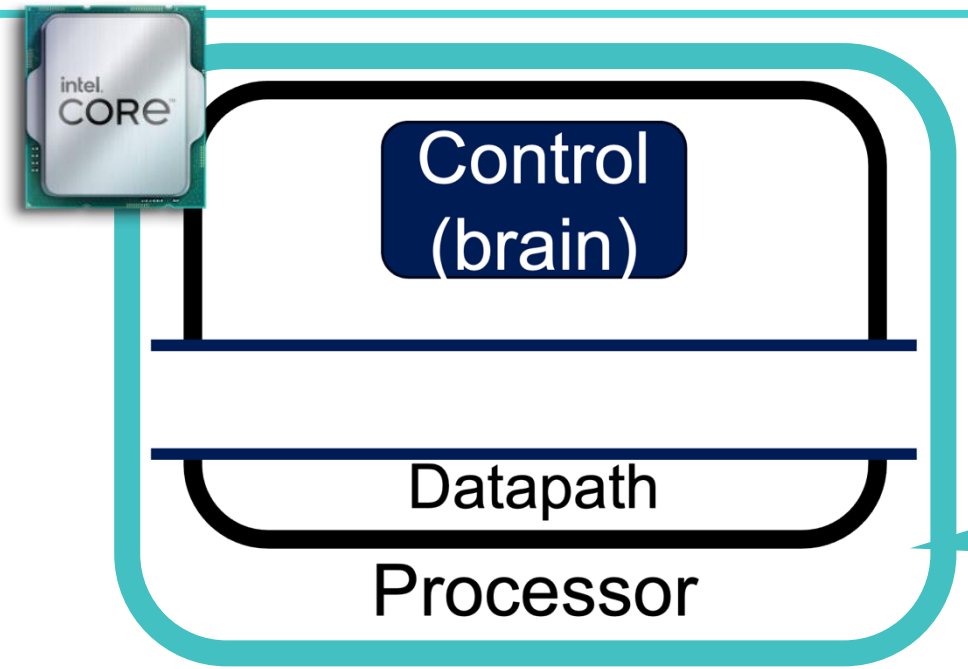
An **instance of a program** that is being executed (or is ready for execution)

- Not the same as “program” or “processor”

- Consists of:
  - Memory (code, heap, stack, etc.)
  - Registers used to manage execution (stack pointer, program counter, …)
  - Other resources

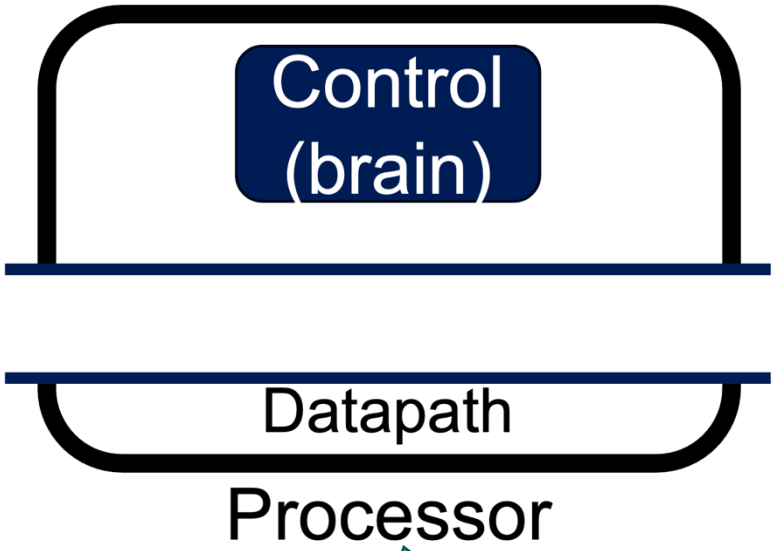


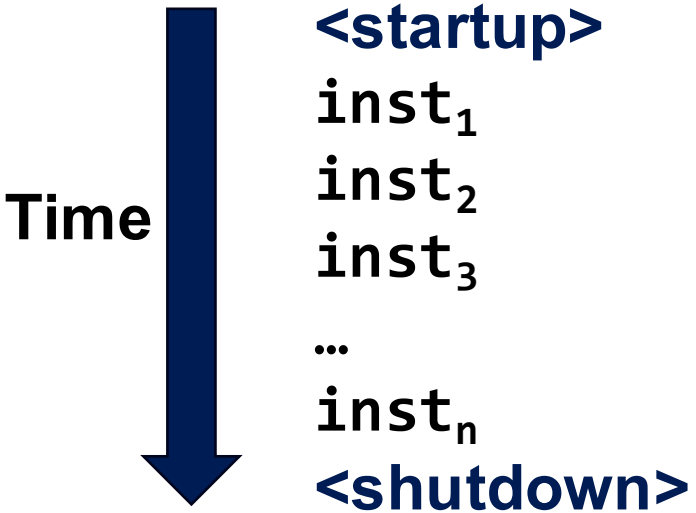
Traditional Model: Uniprocessor



Uniprocessor: one processor (core) per chip

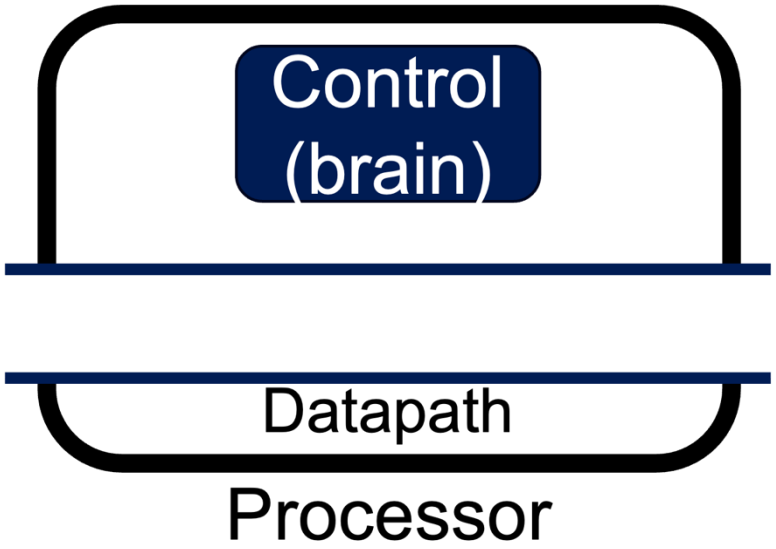
Traditional Model: Uniprocessor

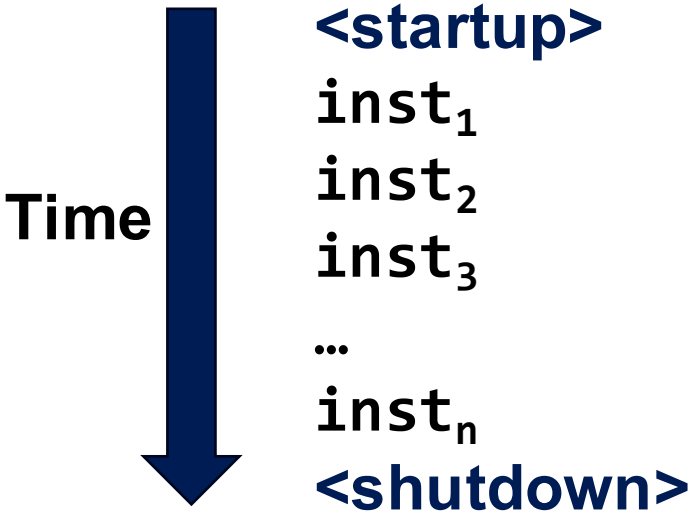




Only do one thing: simply reads and executes a sequence of instructions, one at a time

Traditional Model: Uniprocessor



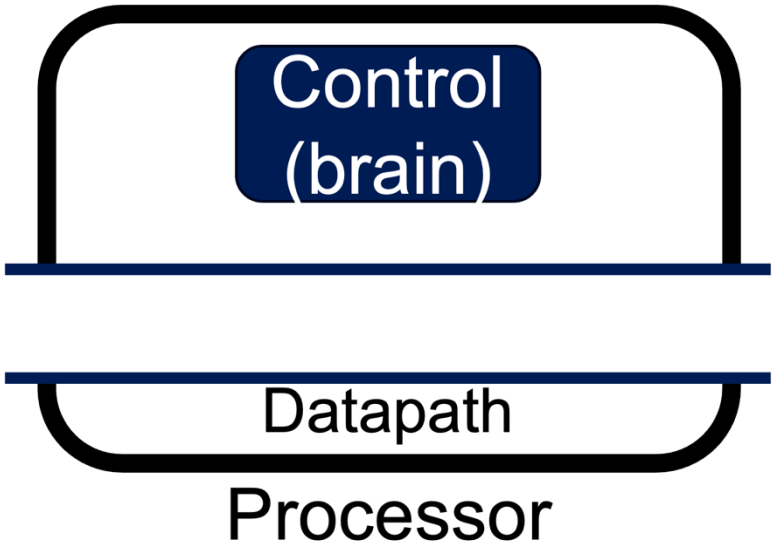


Q. How can **multiple programs run concurrently** on a single processor?

Computer runs many processes simultaneously

- ***Applications for one or more users***
  - ***Web browsers, email clients, editors, …***
- ***Background tasks***
  - ***Monitoring network & I/O devices***

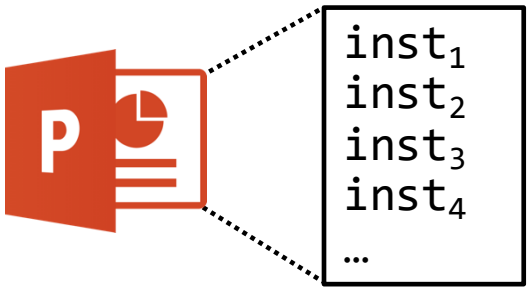
Traditional Model: Uniprocessor

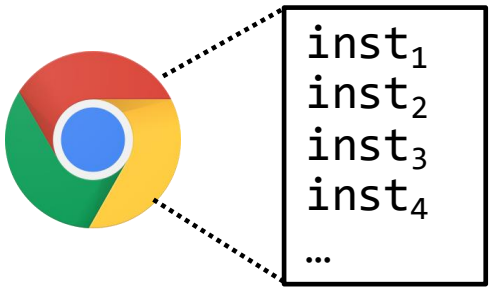


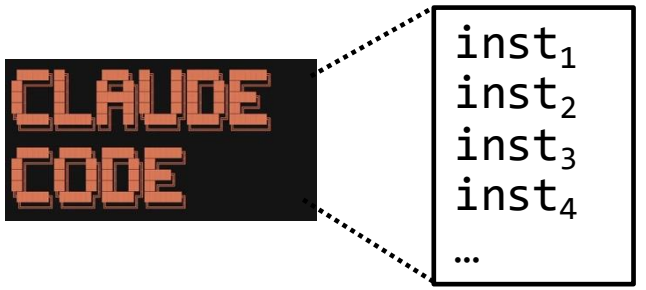
Time

<startup>
$inst_1$
$inst_2$
$inst_3$
…
$inst_n$
<shutdown>

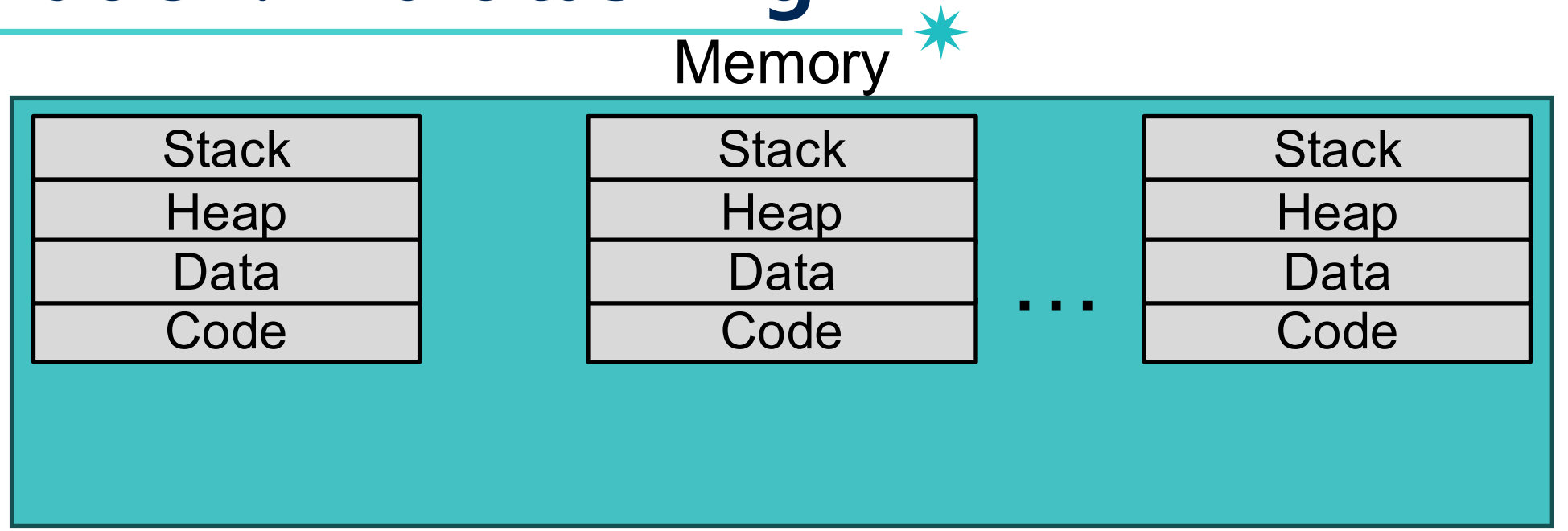
Q. How can **multiple programs run concurrently** on a single processor?

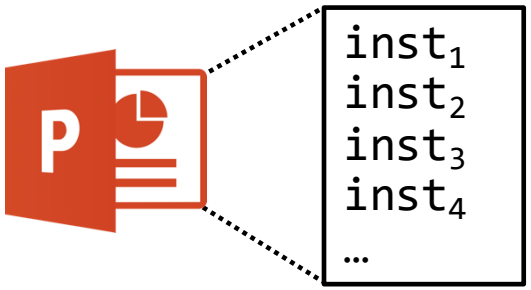


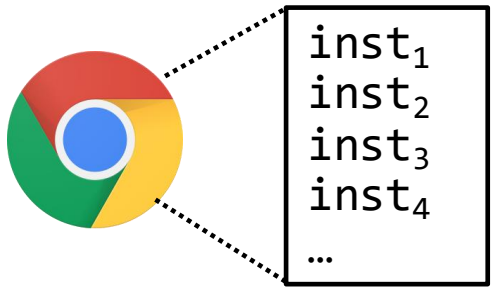


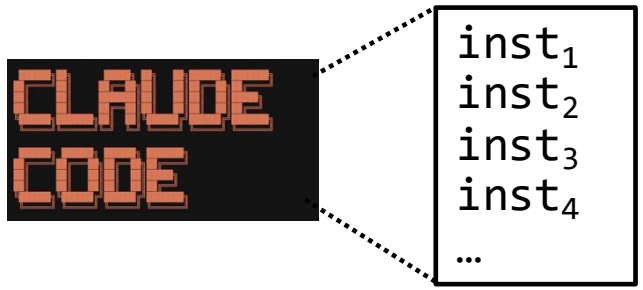


Solution: Multitasking

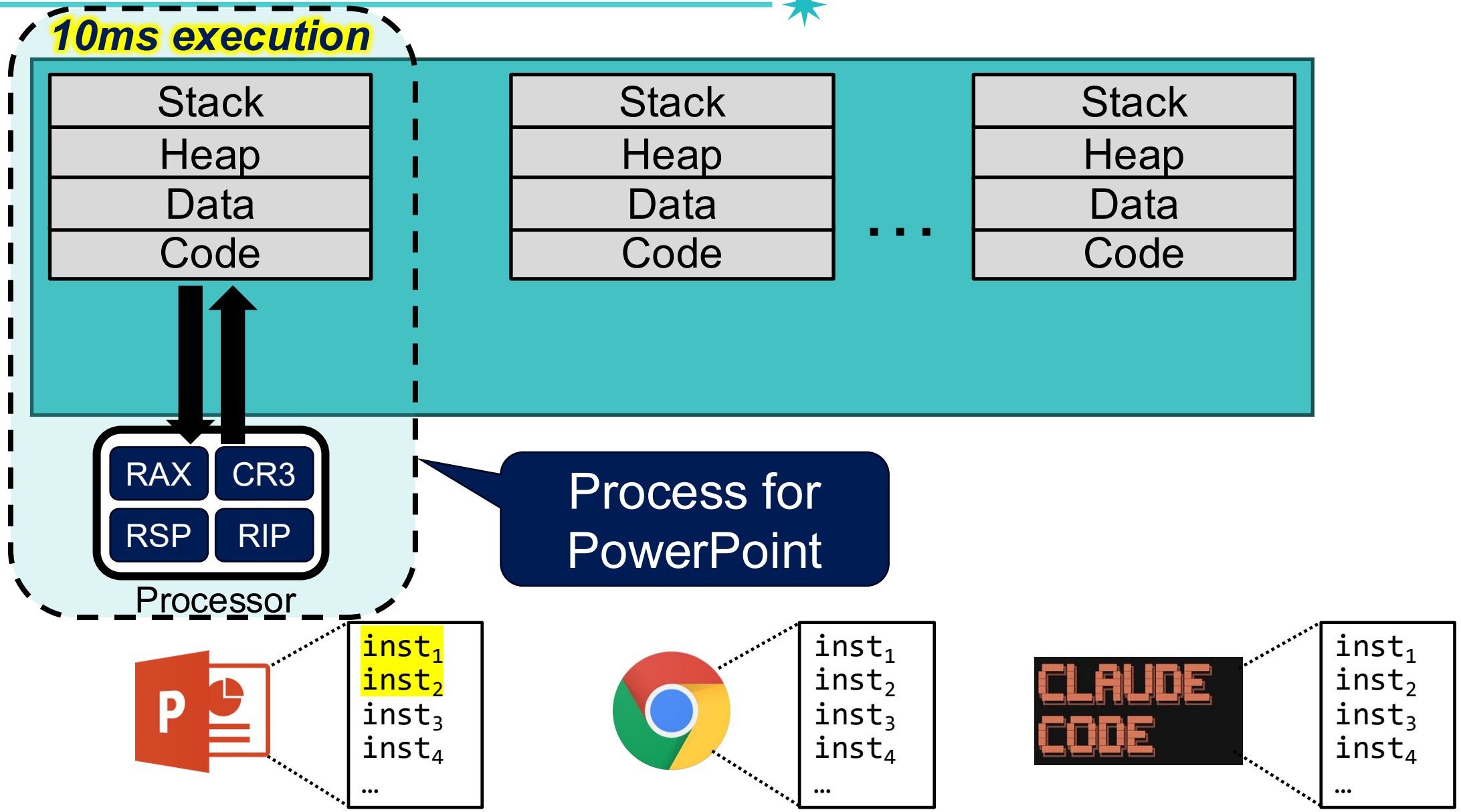




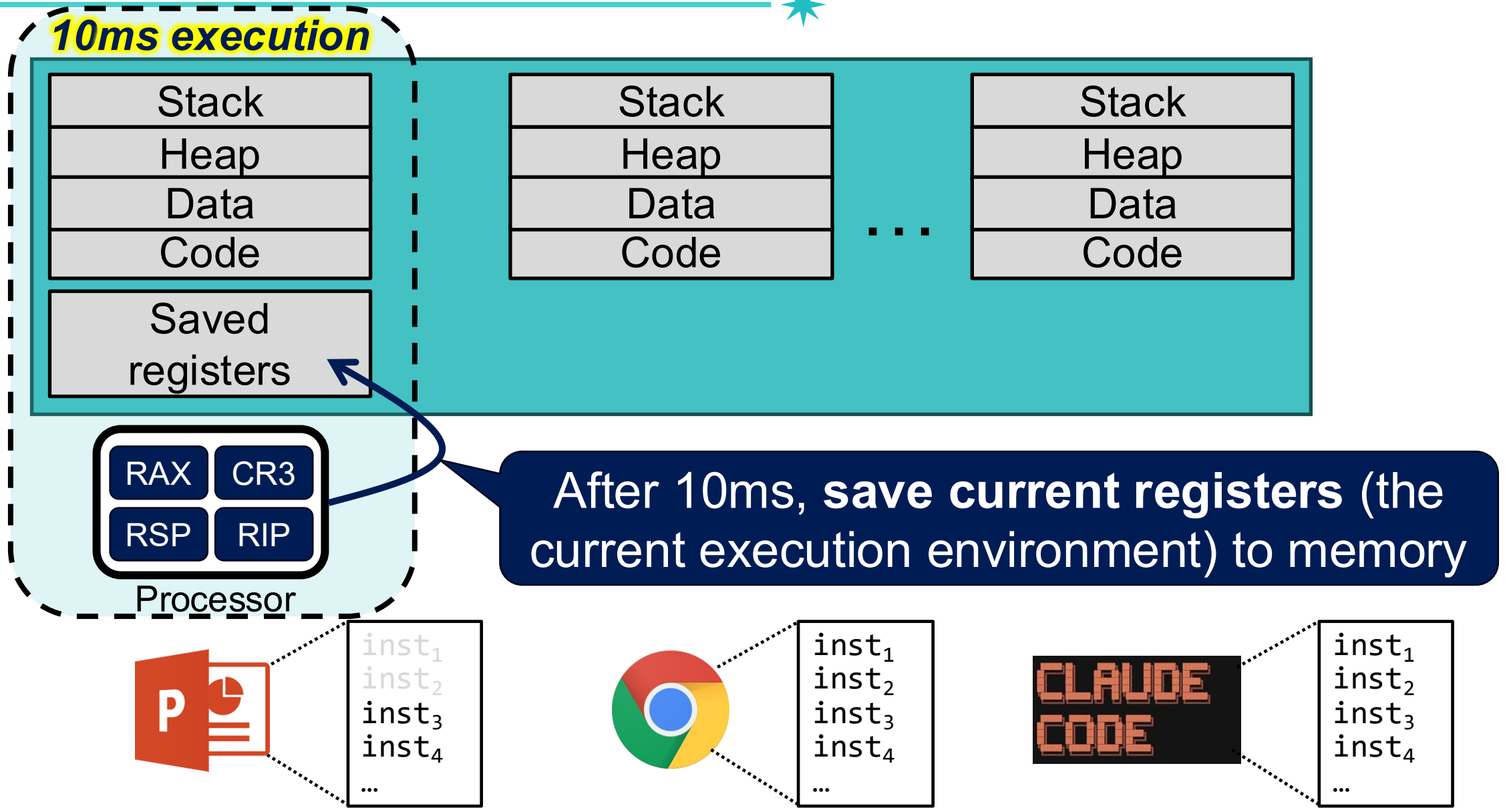




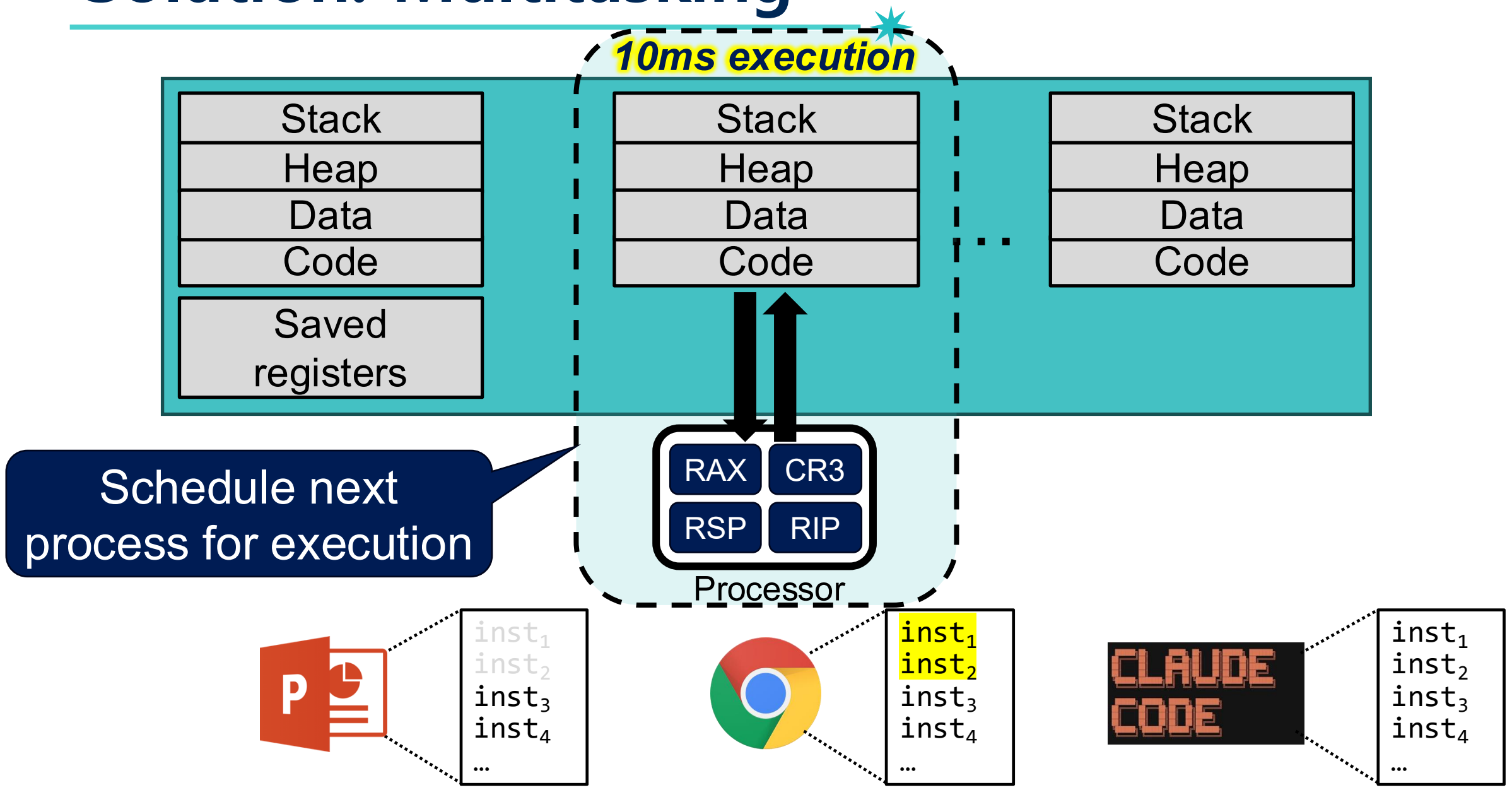
Solution: Multitasking



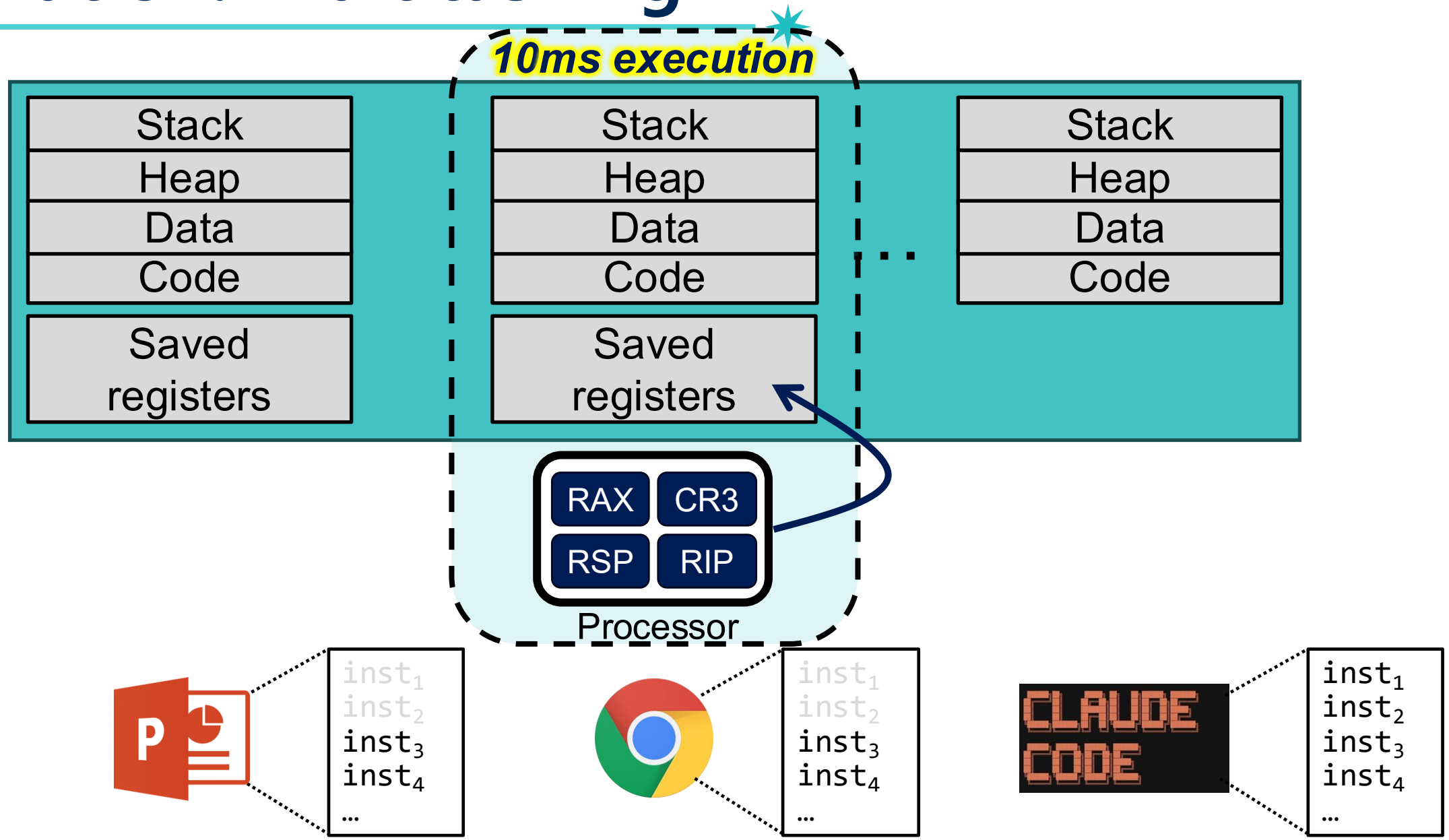
Solution: Multitasking



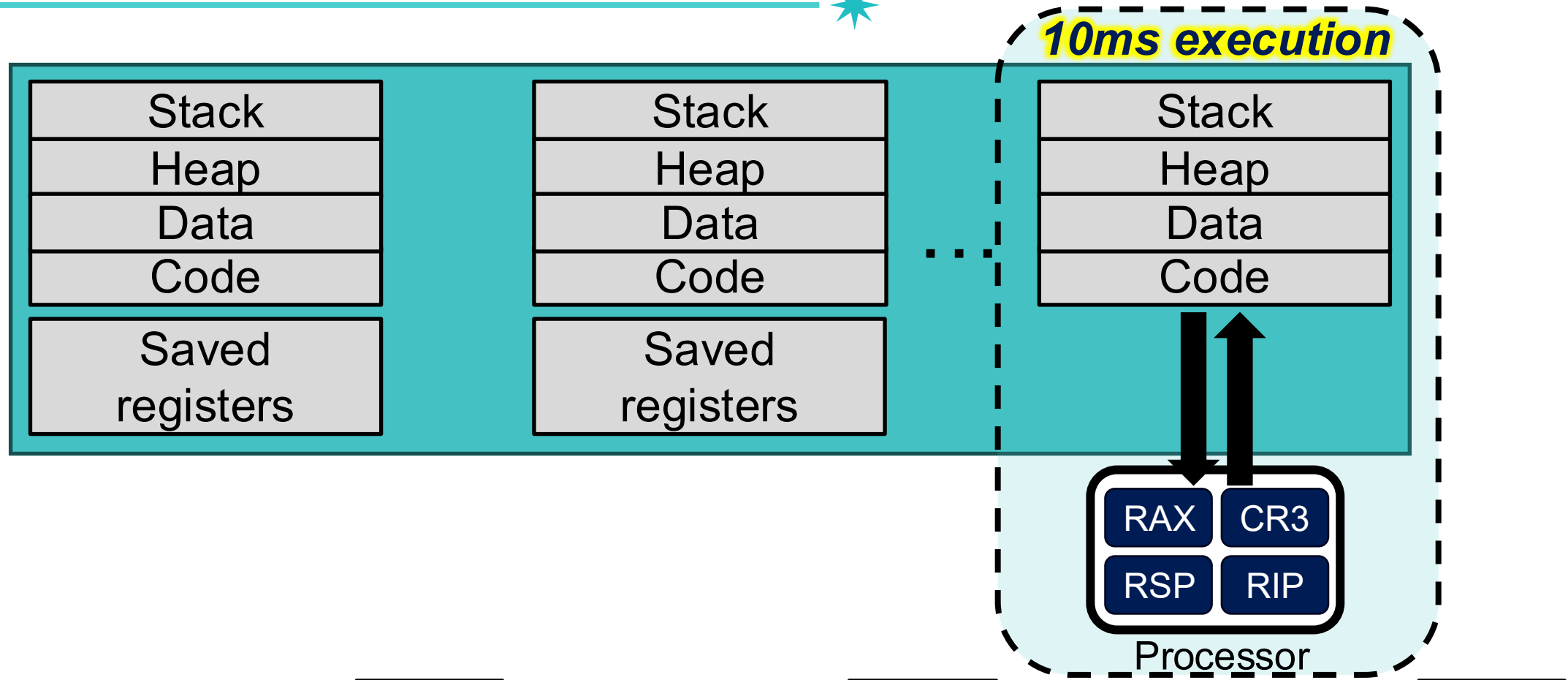
Solution: Multitasking



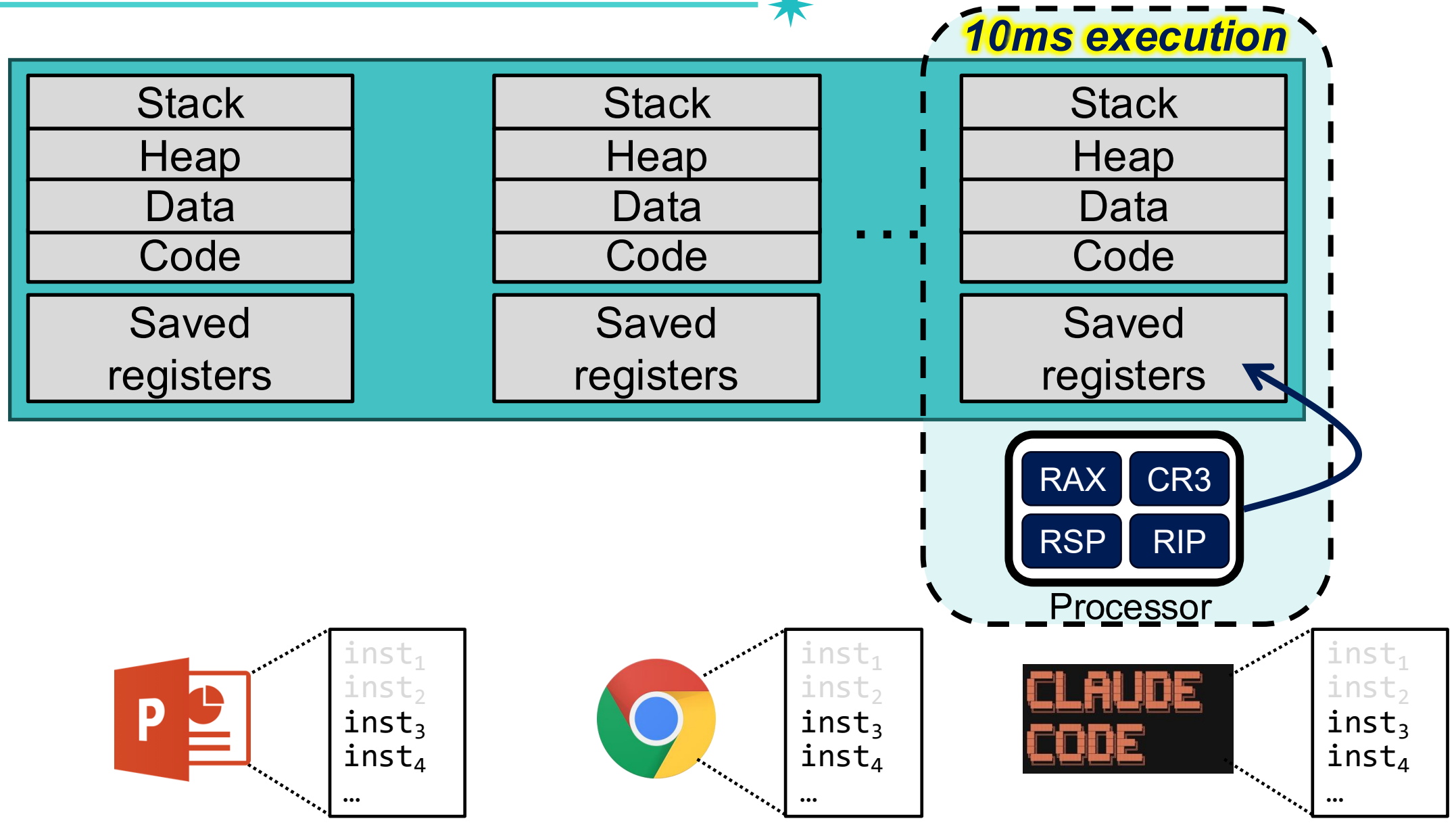
Solution: Multitasking



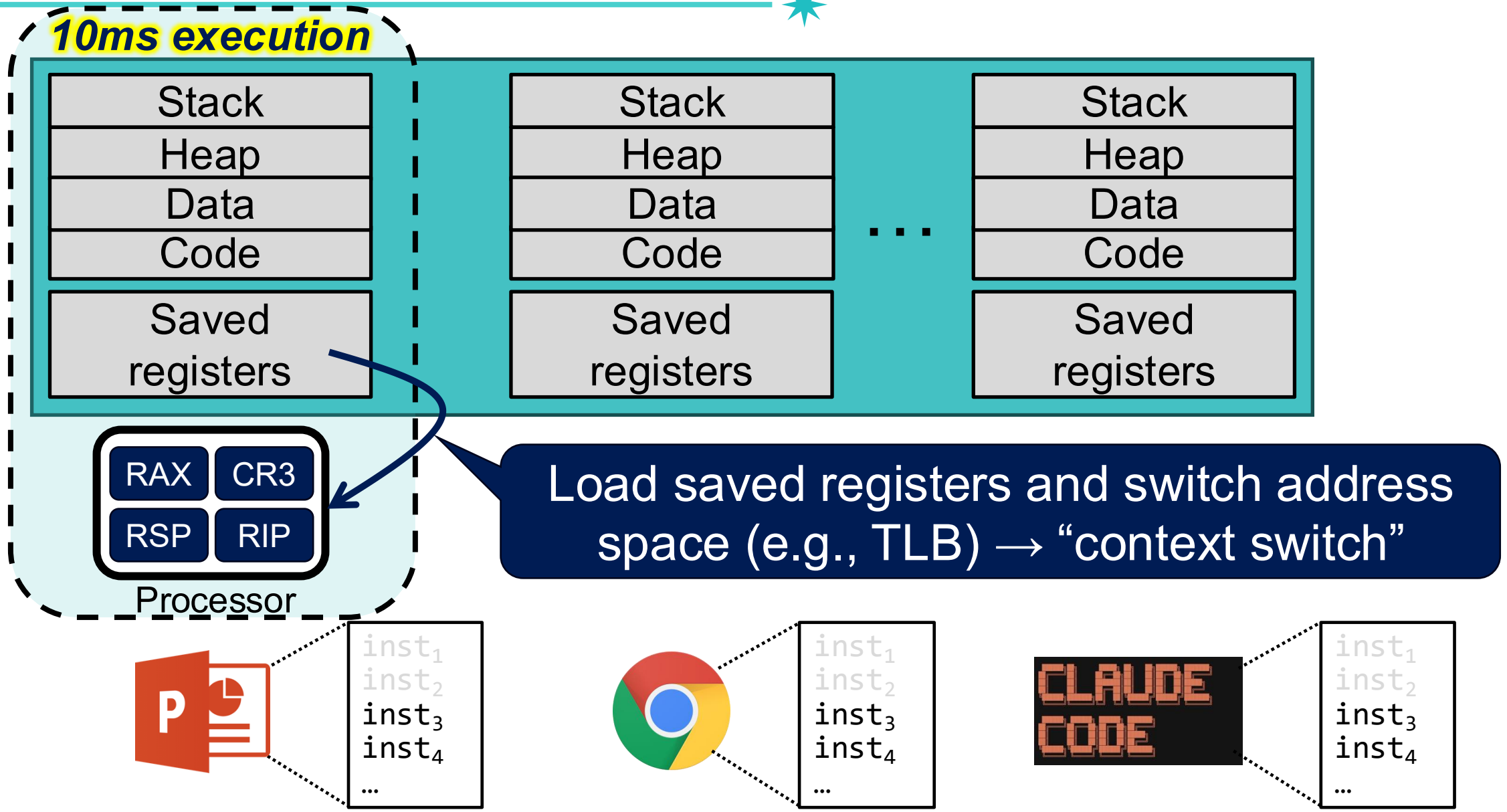
Solution: Multitasking



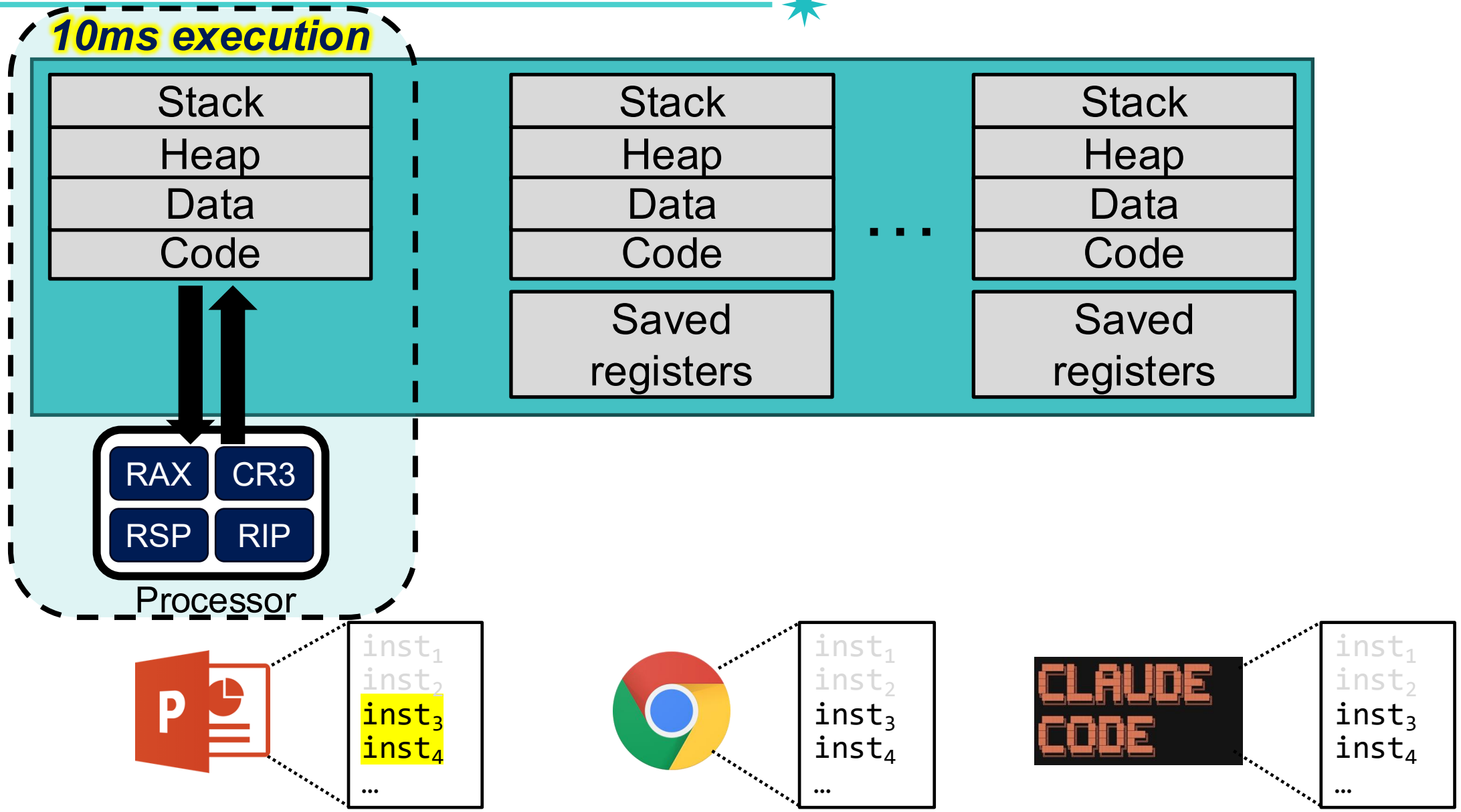
Solution: Multitasking



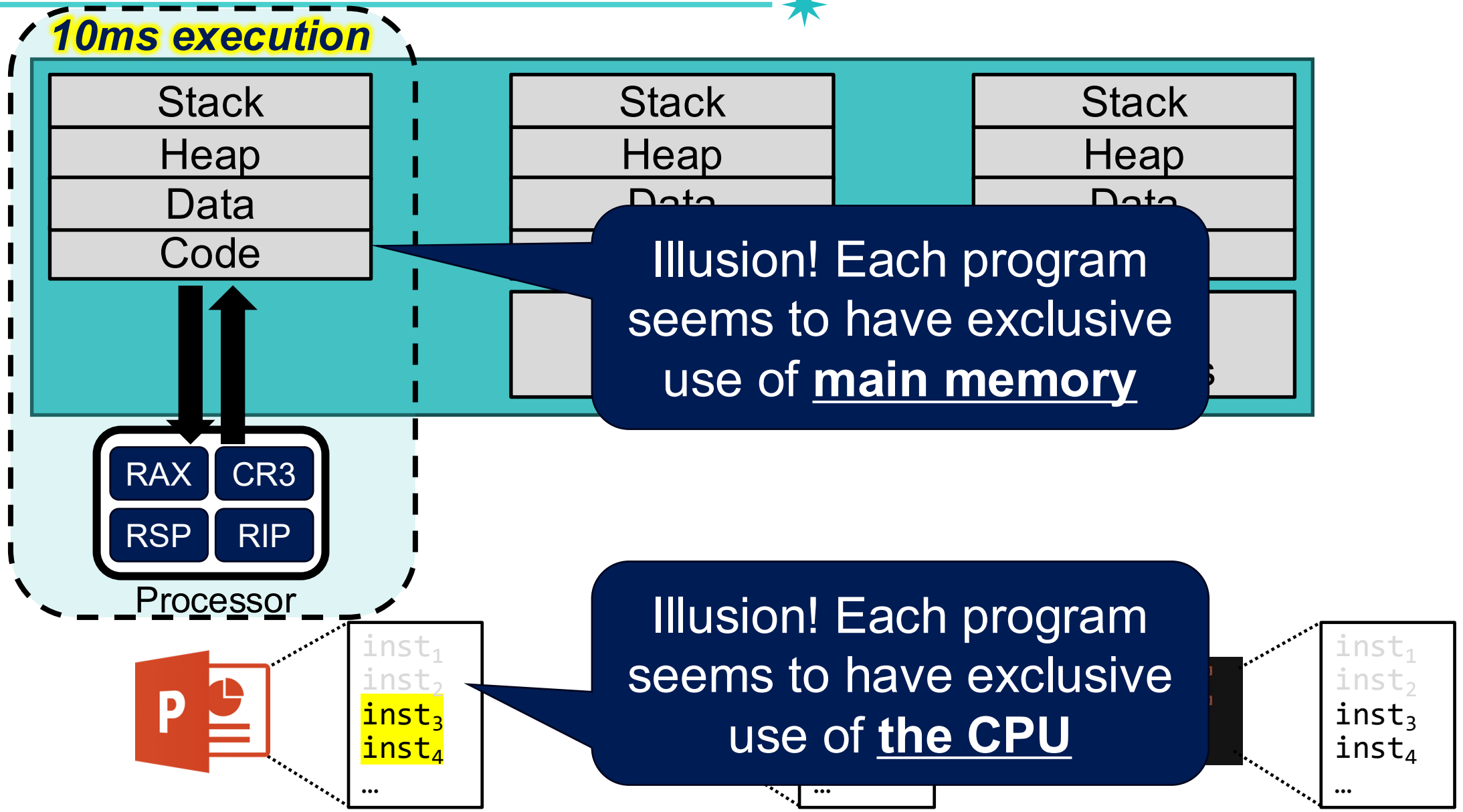
Solution: Multitasking



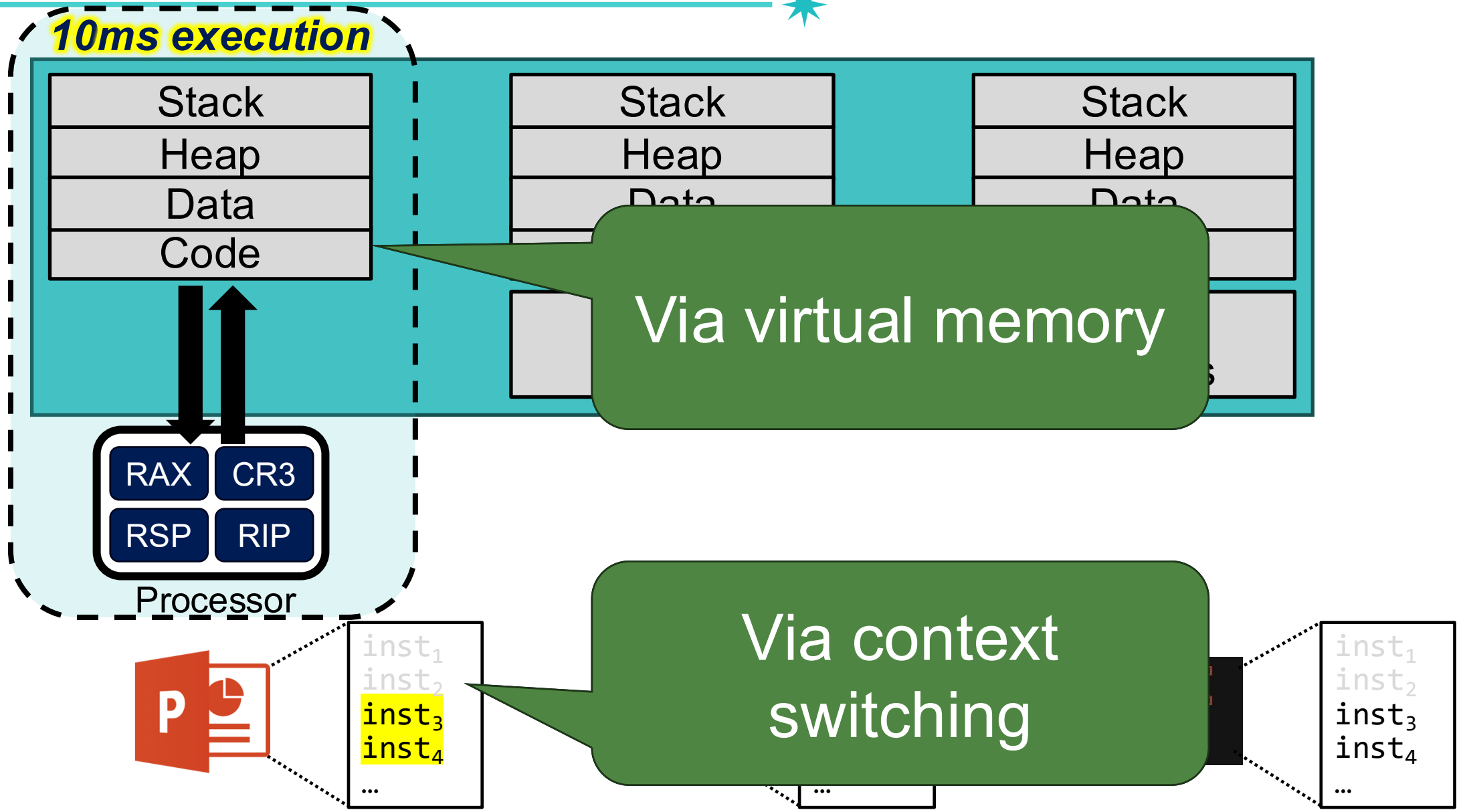
Solution: Multitasking



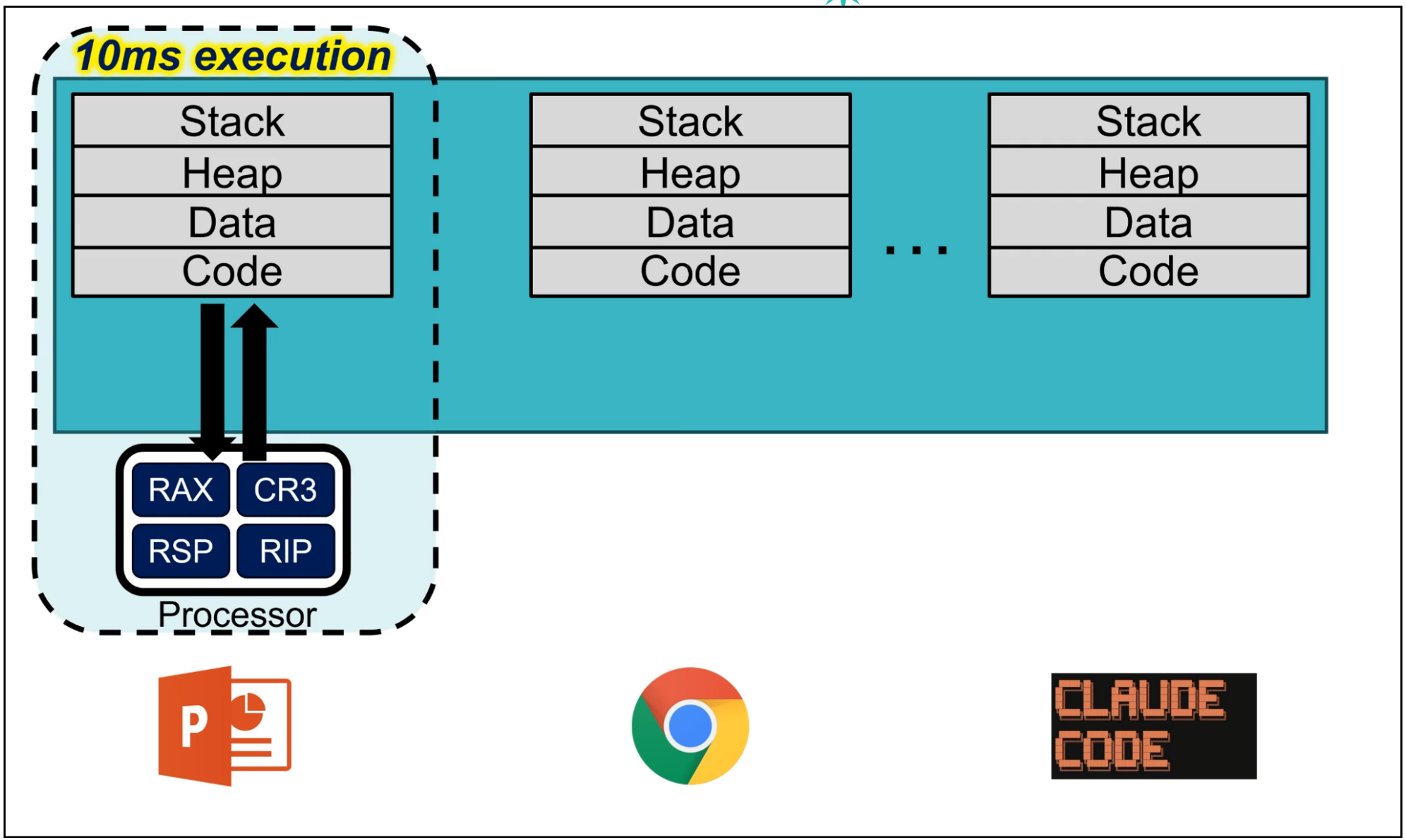
Solution: Multitasking



Solution: Multitasking

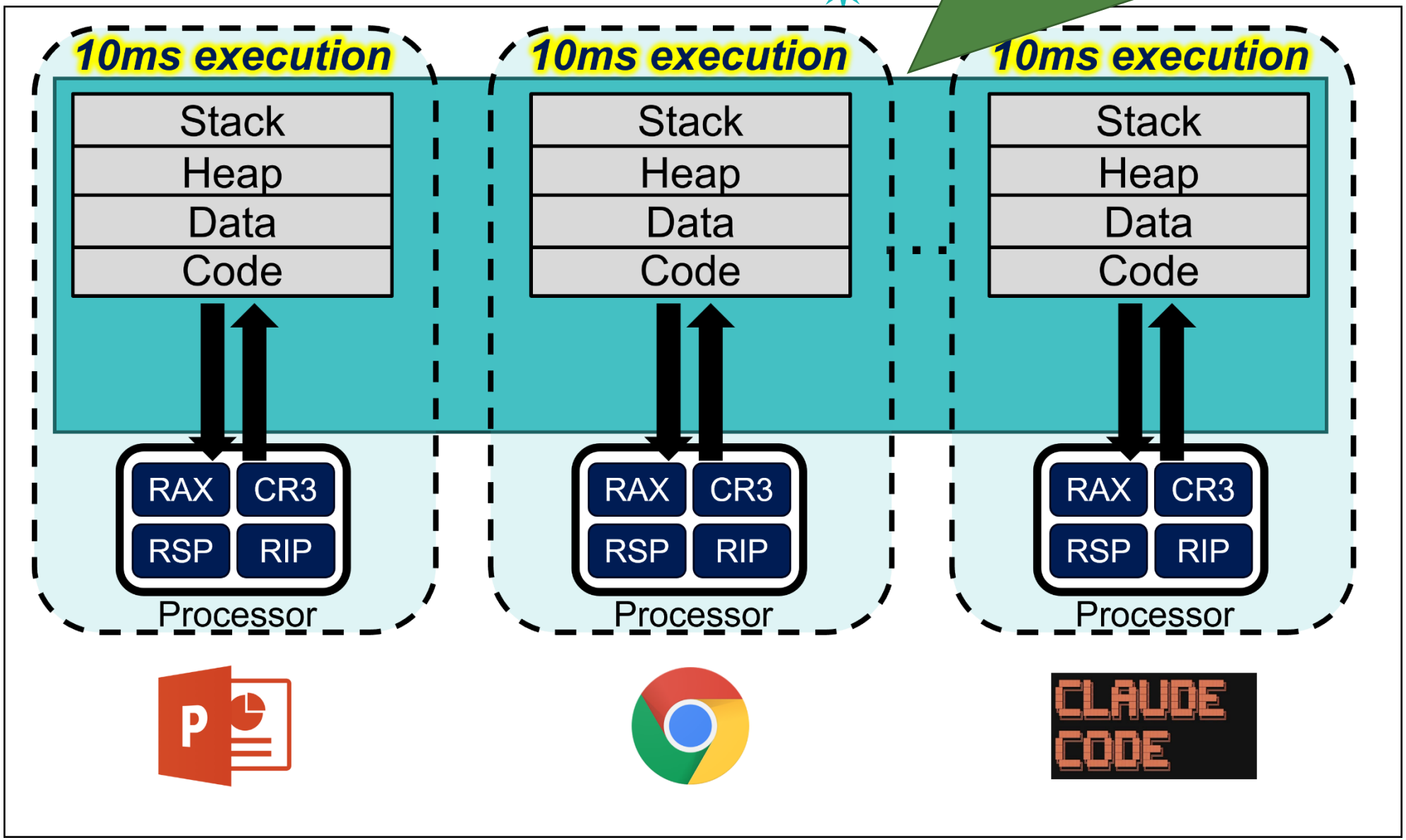


Illusion Demo Video



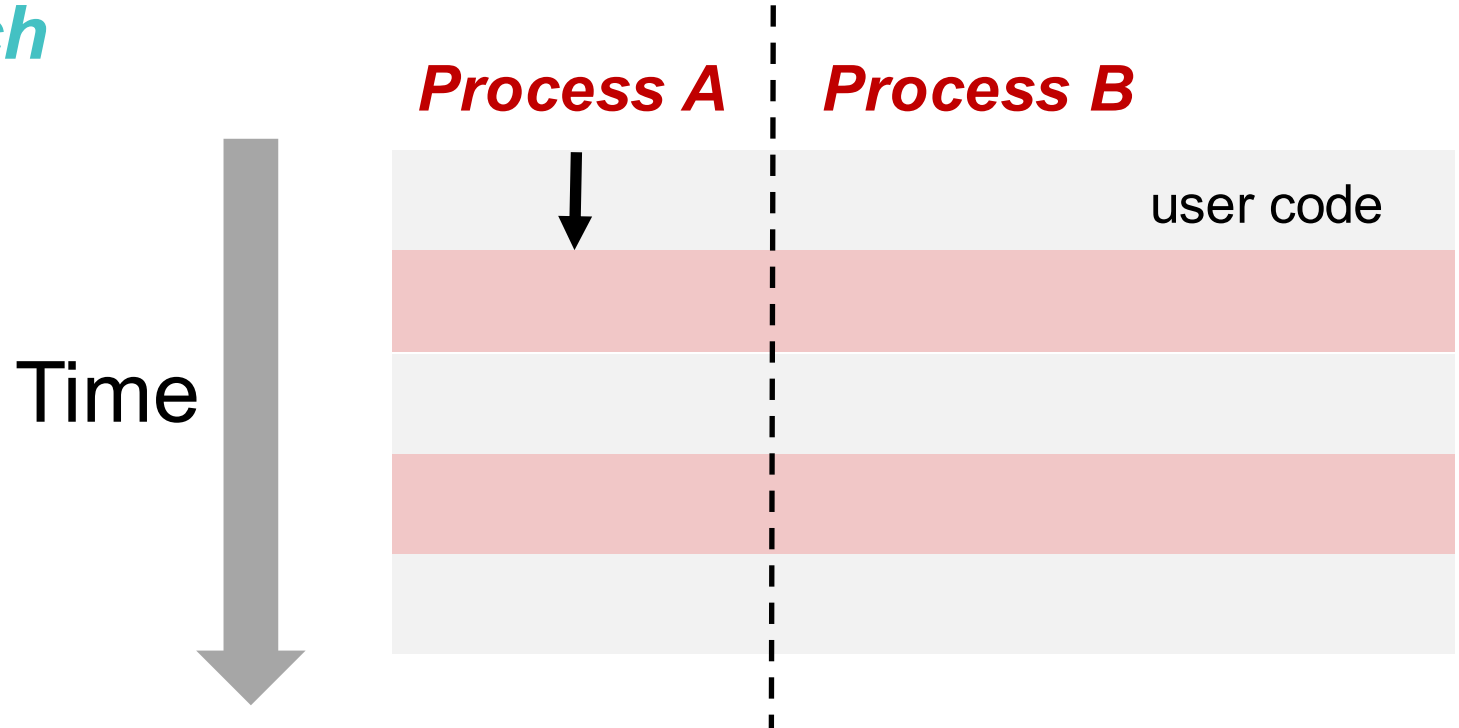
Illusion Logical View

Through context switching, the OS makes a **single CPU** appear as **multiple logical CPUs, one for each process**



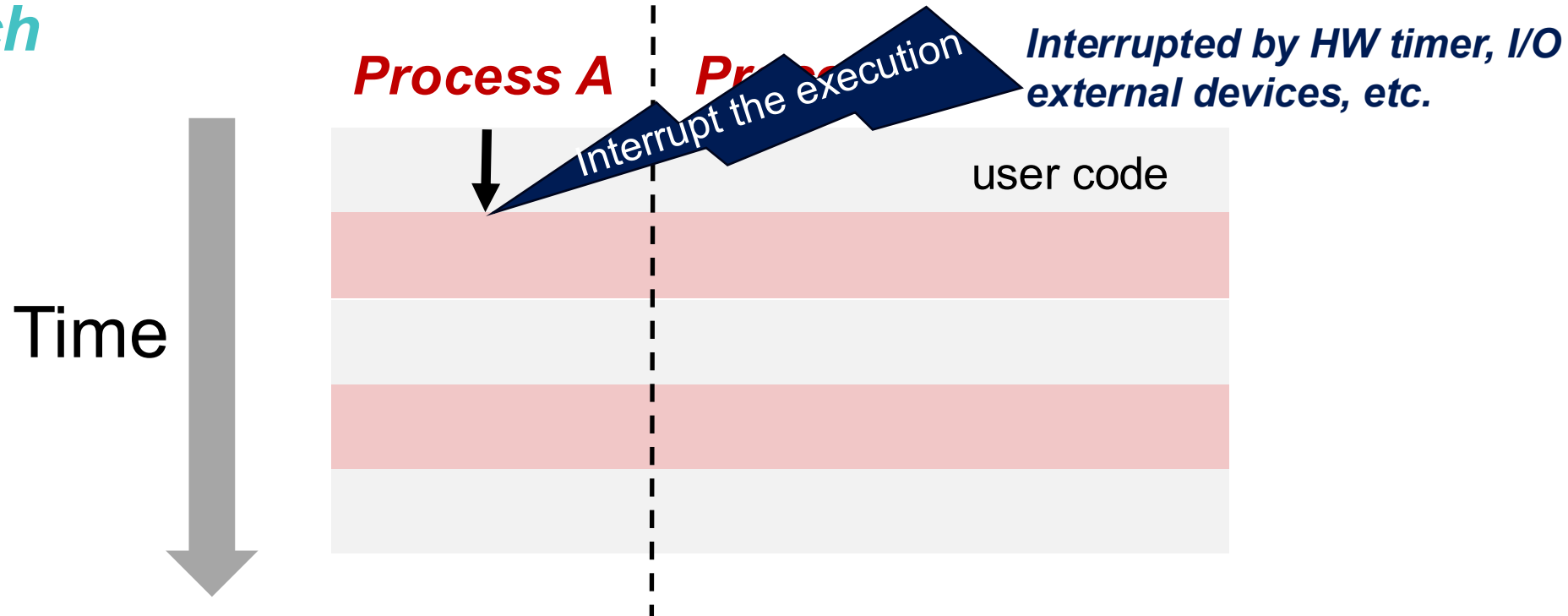
Context Switching

- Processes are managed by a shared chunk of memory-resident OS code called the **kernel**
  - Important: the kernel is not a separate process, but rather runs as part of some existing process
- Control flow passes from one process to another via a ***context switch***



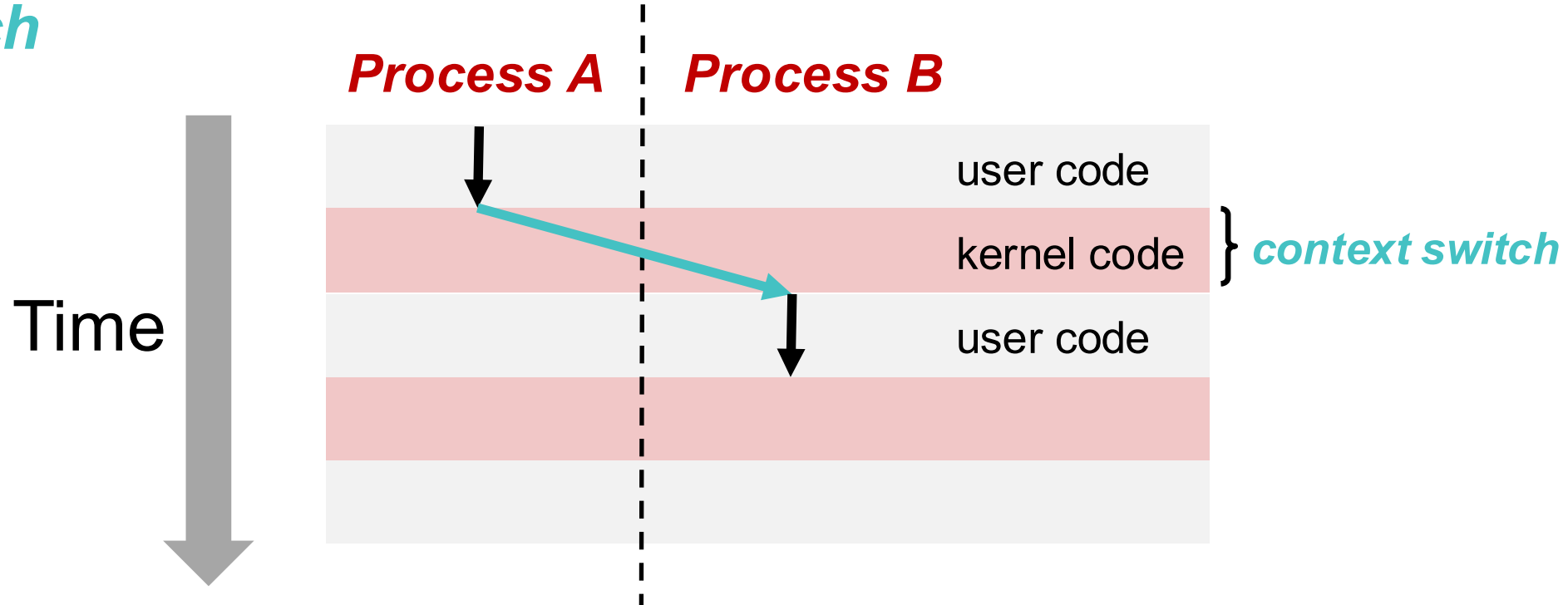
Context Switching

- Processes are managed by a shared chunk of memory-resident OS code called the **kernel**
  - Important: the kernel is not a separate process, but rather runs as part of some existing process
- Control flow passes from one process to another via a ***context switch***

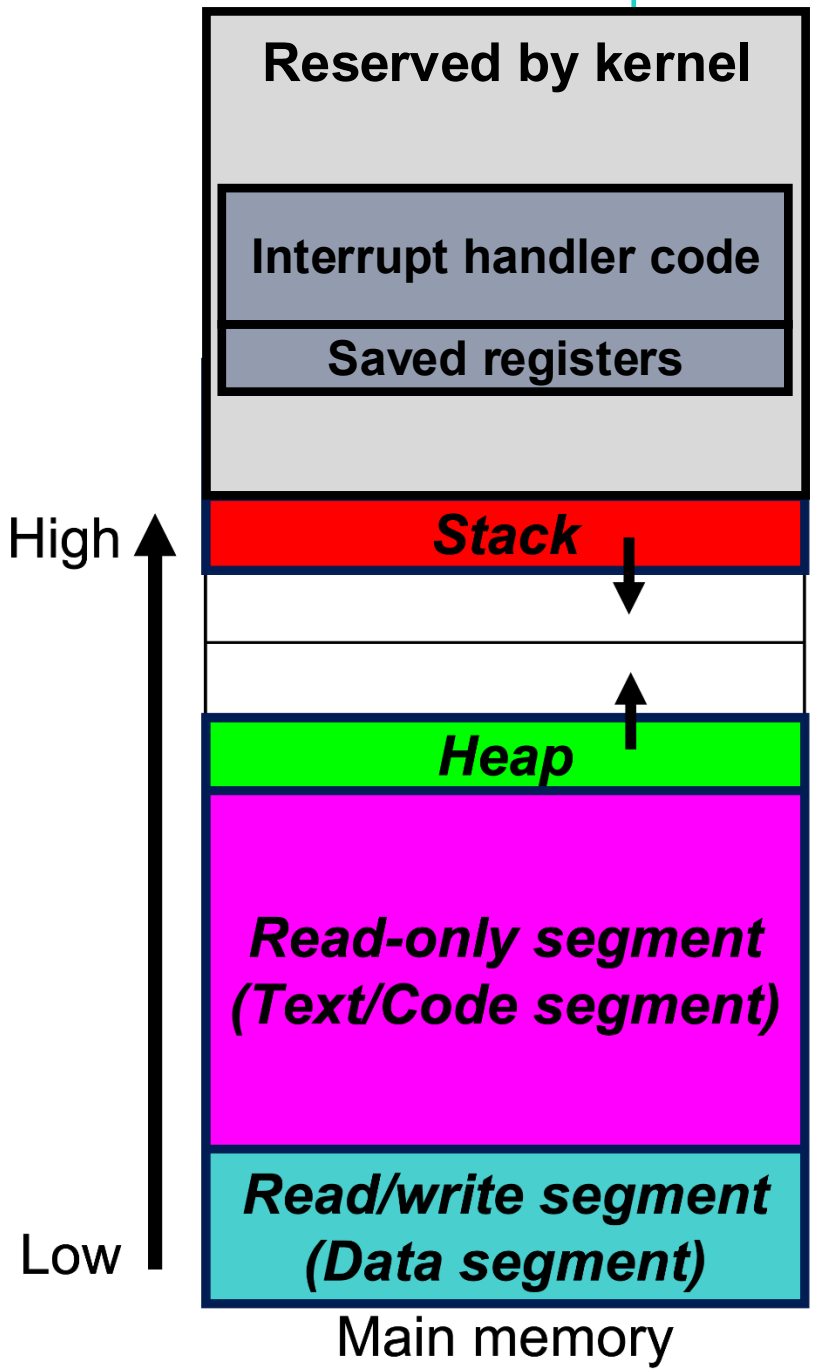


Context Switching

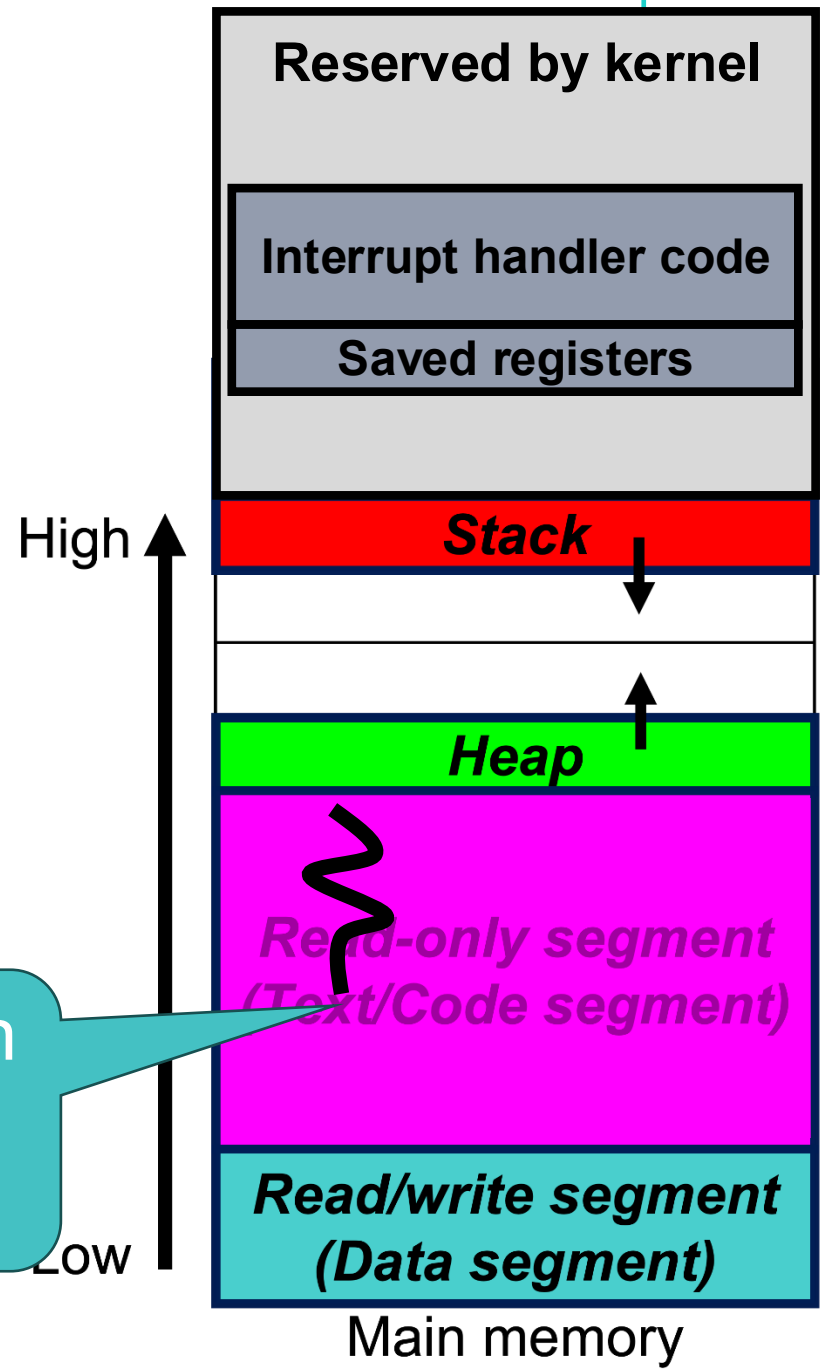
- Processes are managed by a shared chunk of memory-resident OS code called the **kernel**
  - Important: the kernel is not a separate process, but rather runs as part of some existing process
- Control flow passes from one process to another via a ***context switch***



Context Switching

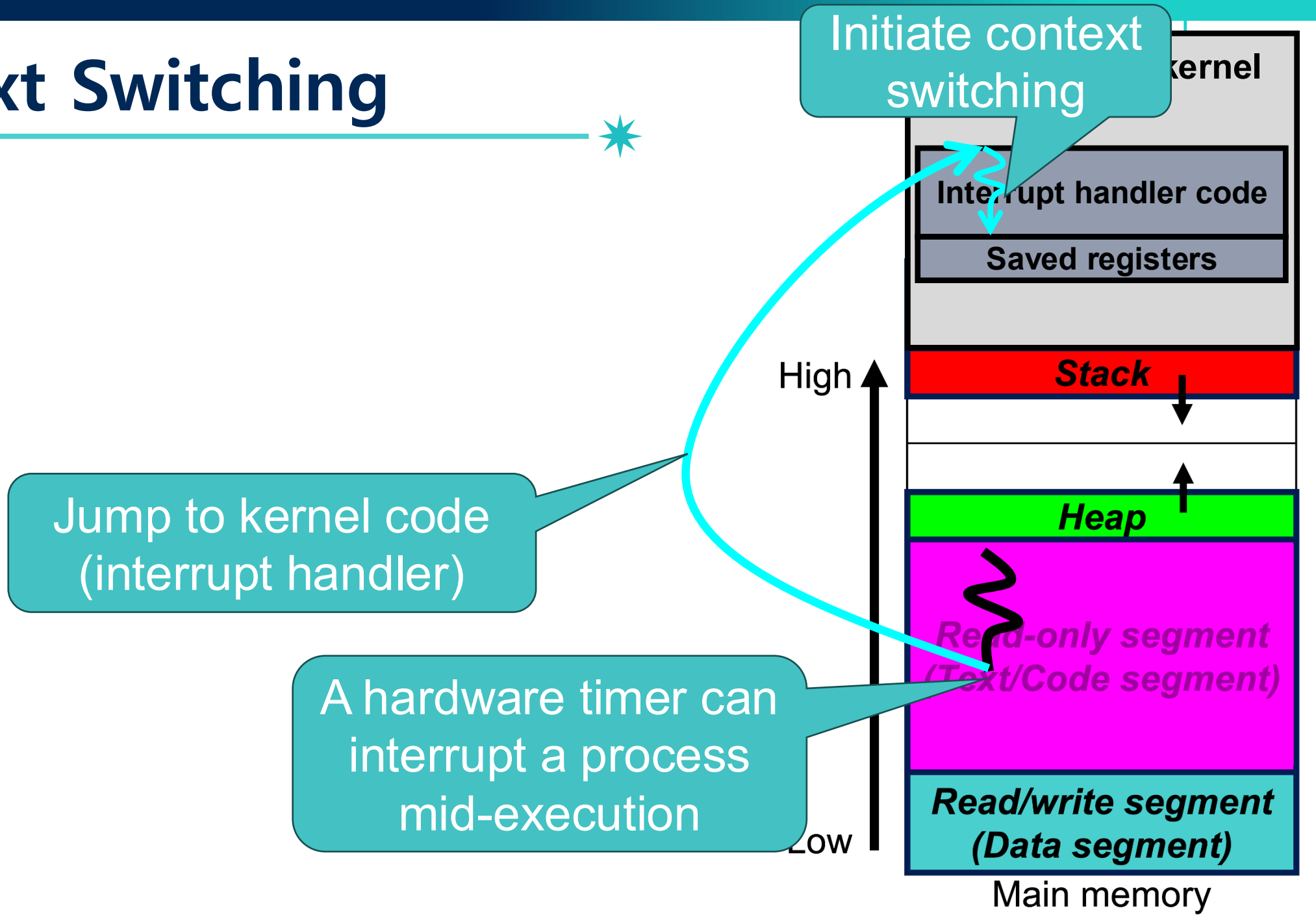


Context Switching



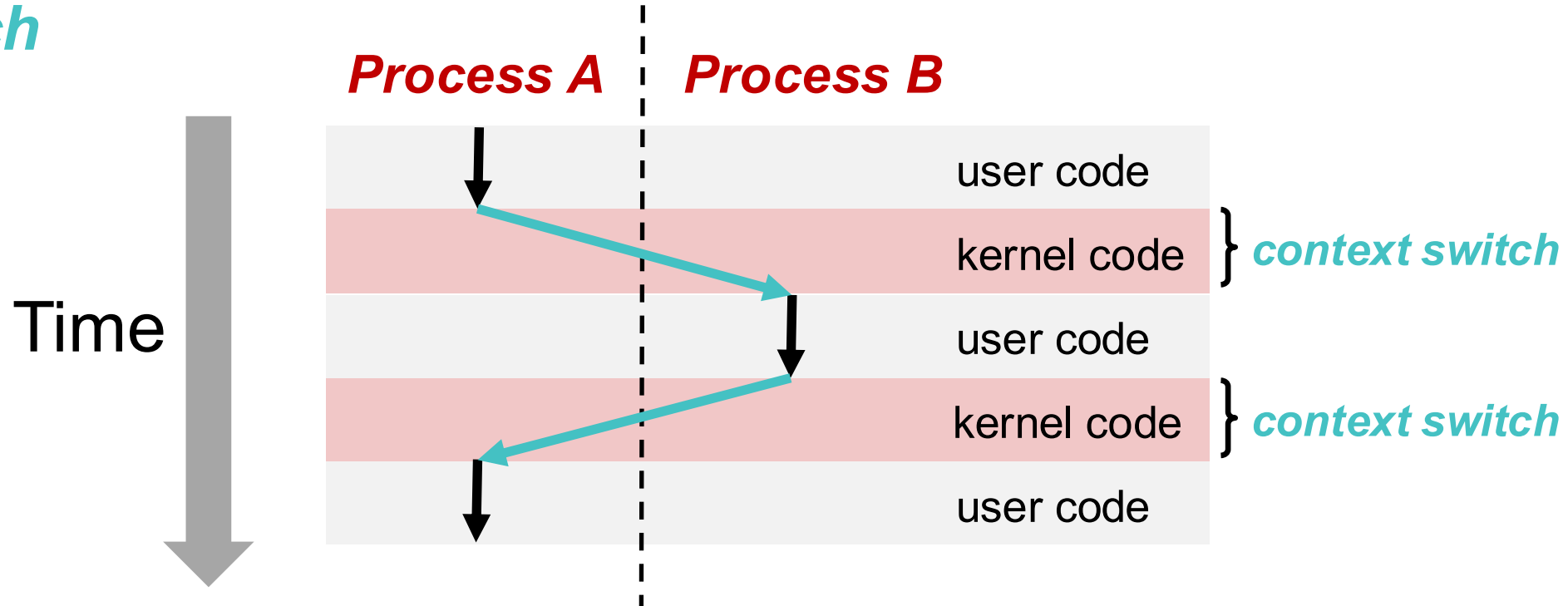
A hardware timer can interrupt a process mid-execution

Context Switching



Context Switching

- Processes are managed by a shared chunk of memory-resident OS code called the **kernel**
  - Important: the kernel is not a separate process, but rather runs as part of some existing process
- Control flow passes from one process to another via a ***context switch***



Multitasking

- A **single processor** executes multiple processes **concurrently**
  - Process executions interleaved through context switching
  - Address spaces managed by virtual memory system
  - Register values for non-executing processes saved in memory

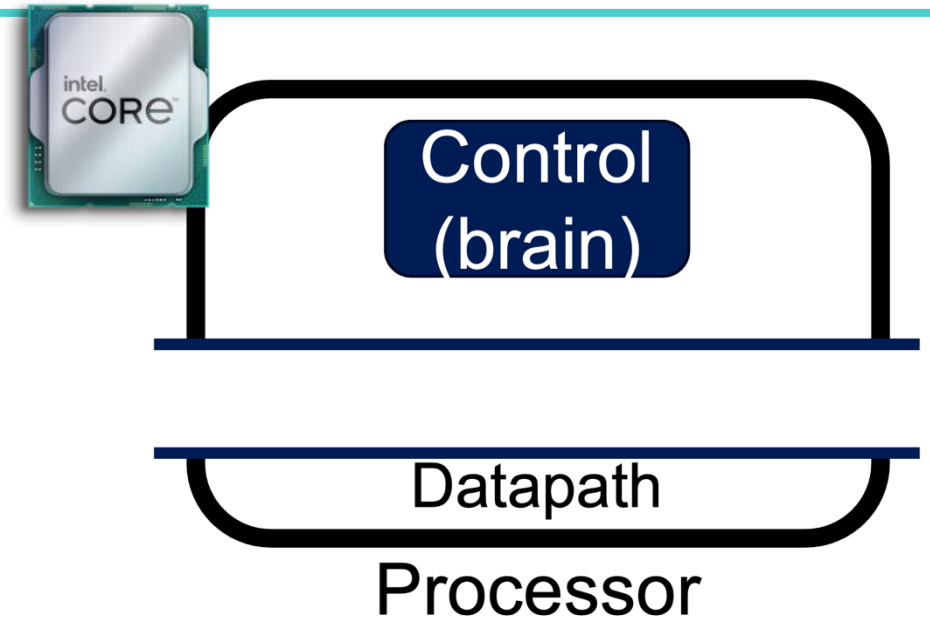
Summary: Process

An **instance of a program** that is being executed (or is ready for execution)

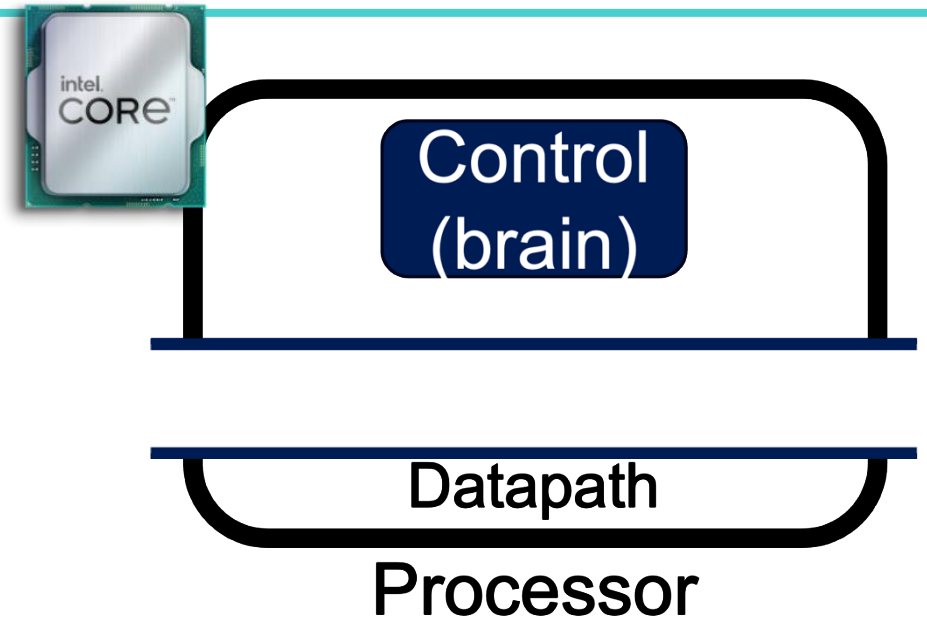
- Not the same as “program” or “processor”

- Provides each program with two key abstractions:
  - ***Private address space***
    - Each program seems to have exclusive use of main memory
    - Provided by kernel mechanism called **virtual memory**
  - ***Logical control flow***
    - Each program seems to have exclusive use of the CPU
    - Provided by kernel mechanism called **context switching**

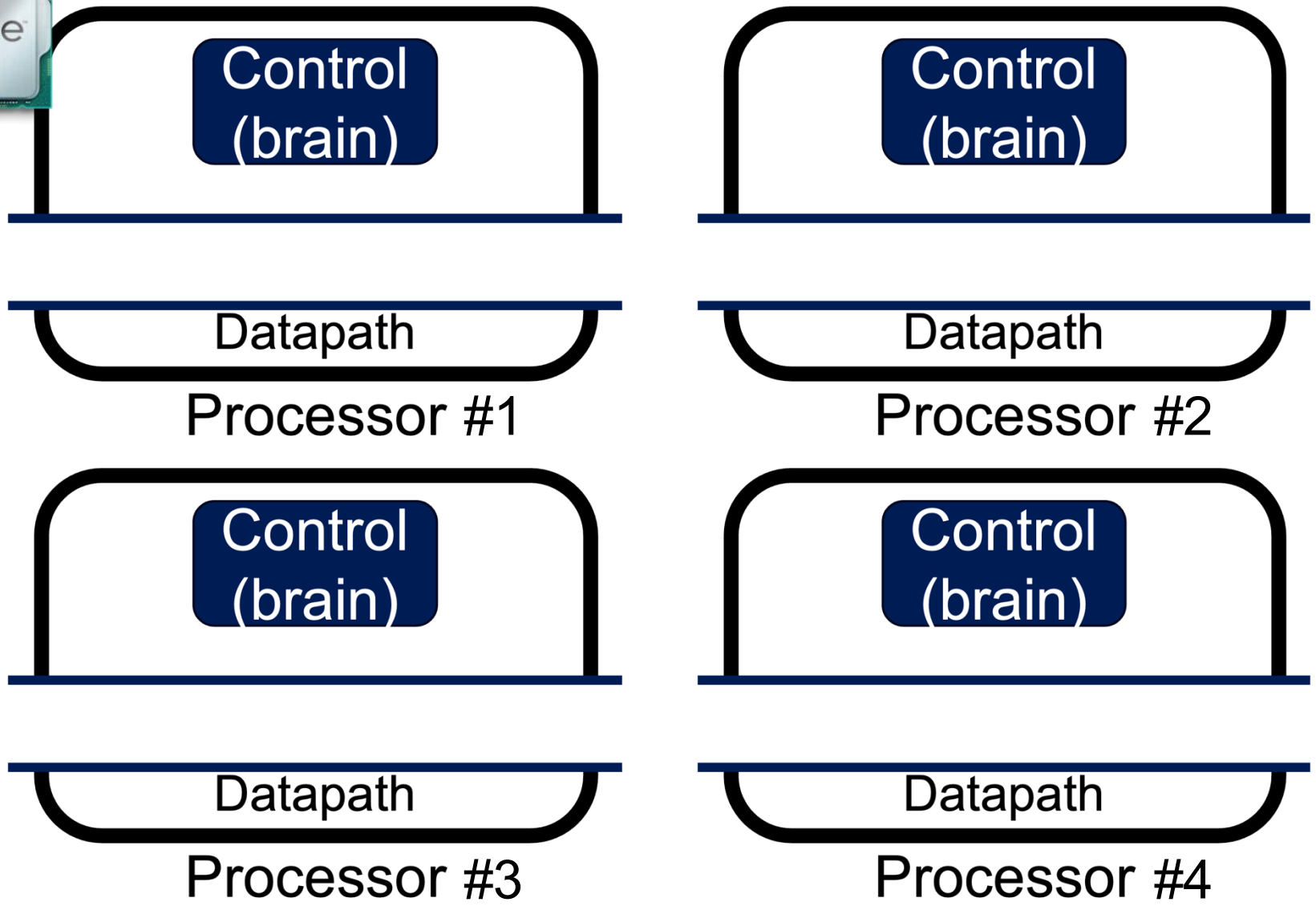
Nowadays: Uniprocessor to Multiprocessors



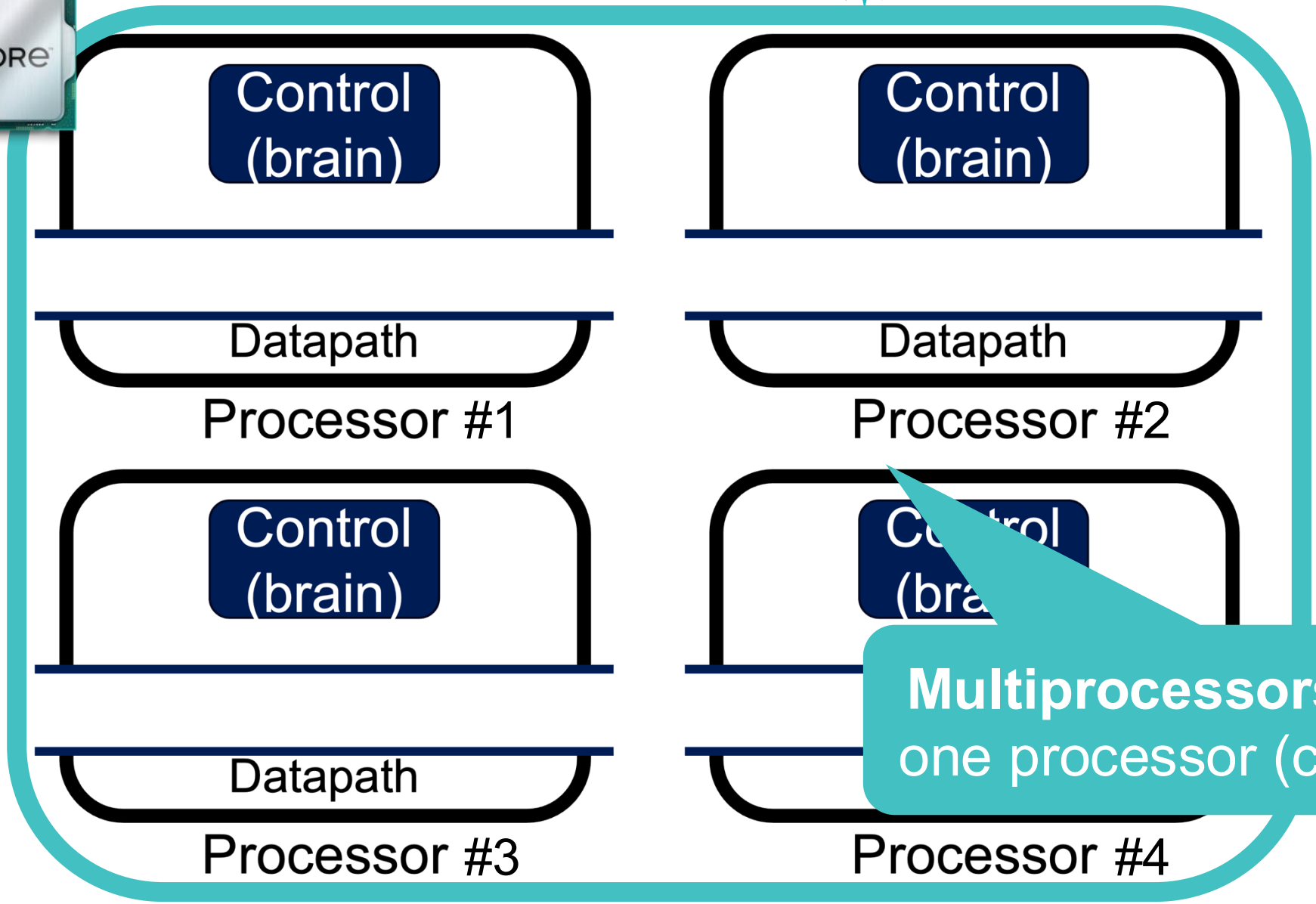
Nowadays: Uniprocessor to Multiprocessors



Nowadays: Uniprocessor to Multiprocessors

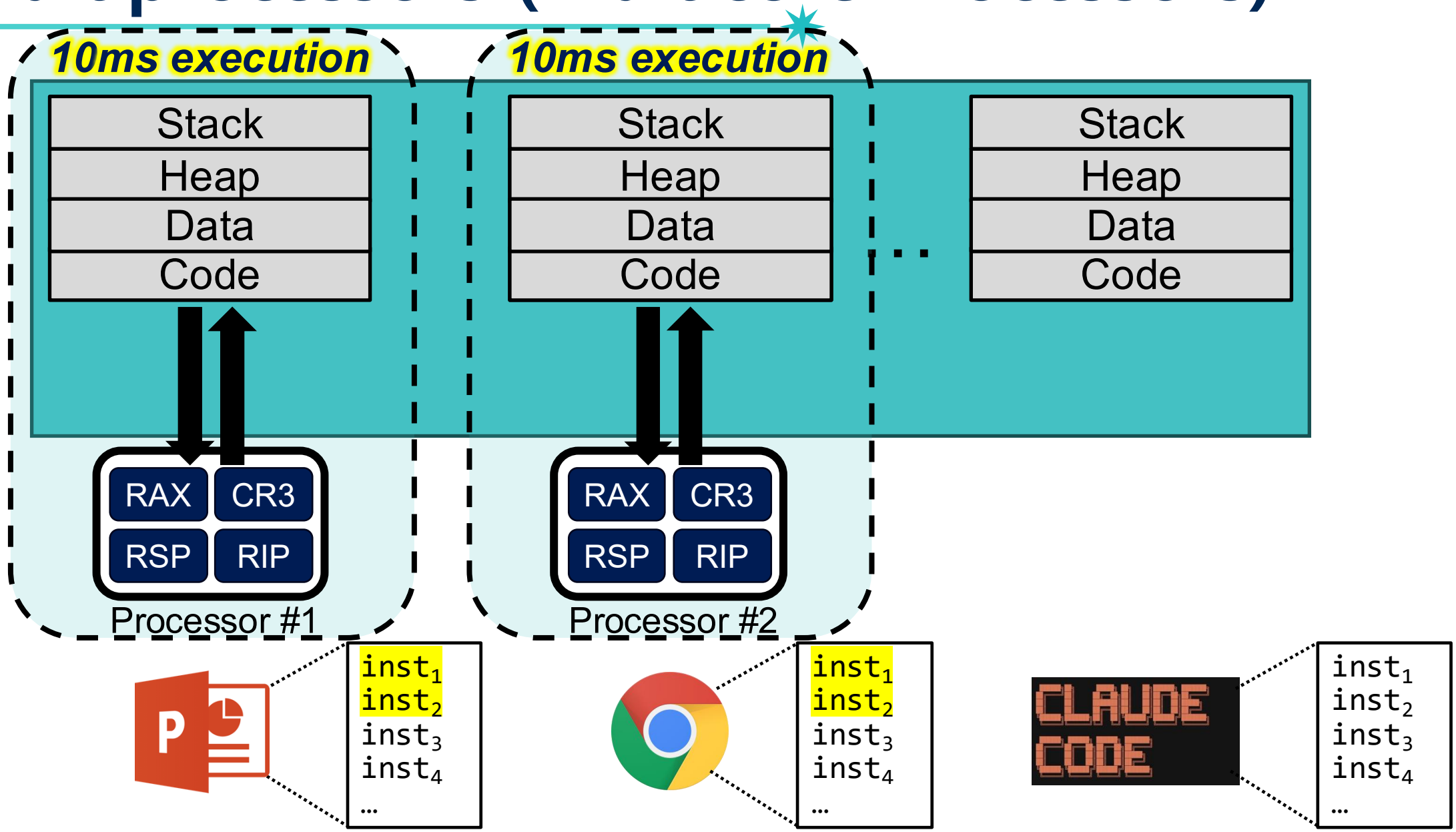


Nowadays: Uniprocessor to Multiprocessors

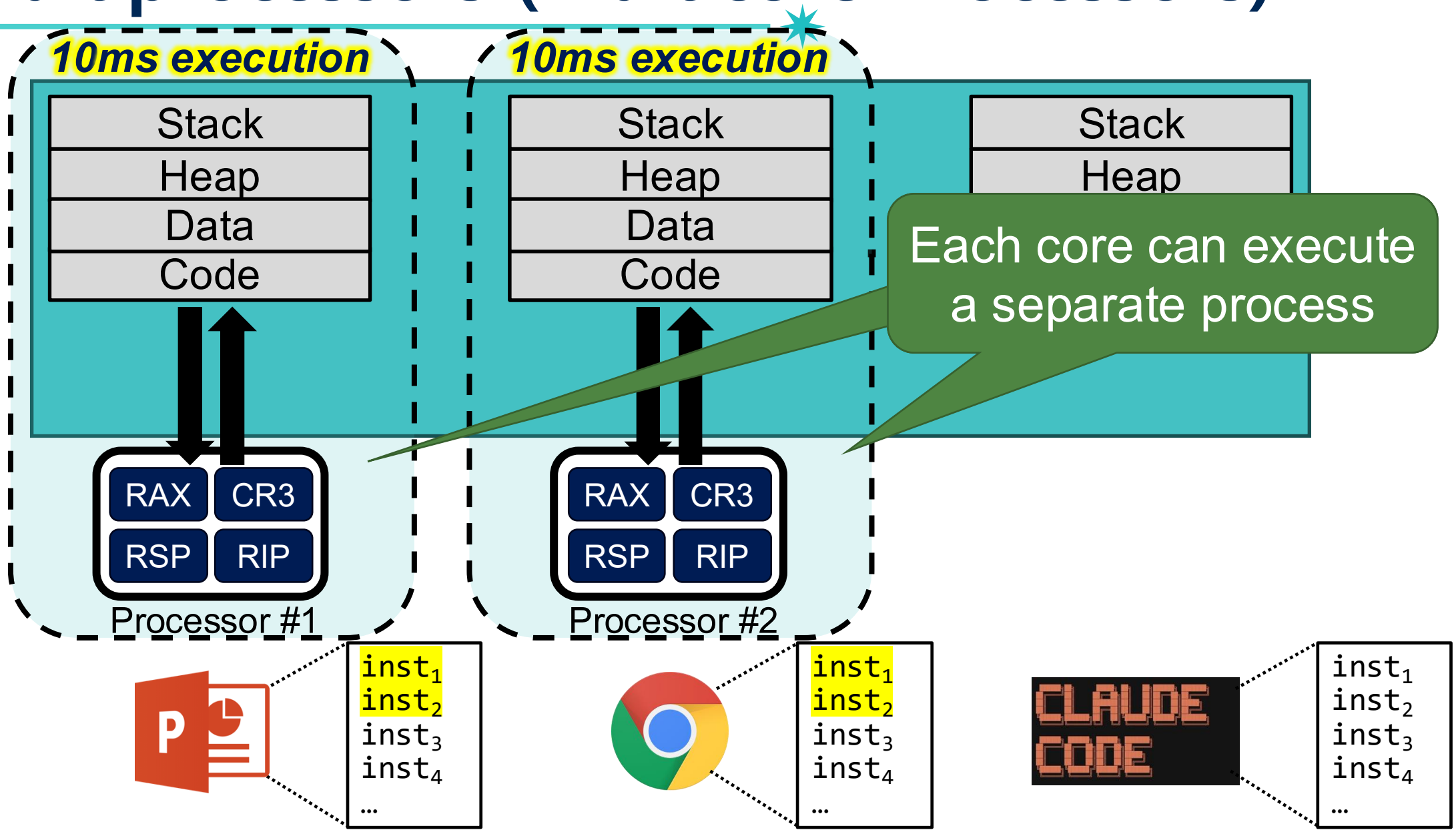


Multiprocessors: more than one processor (core) per chip

Multiprocessors (Multicore Processors)



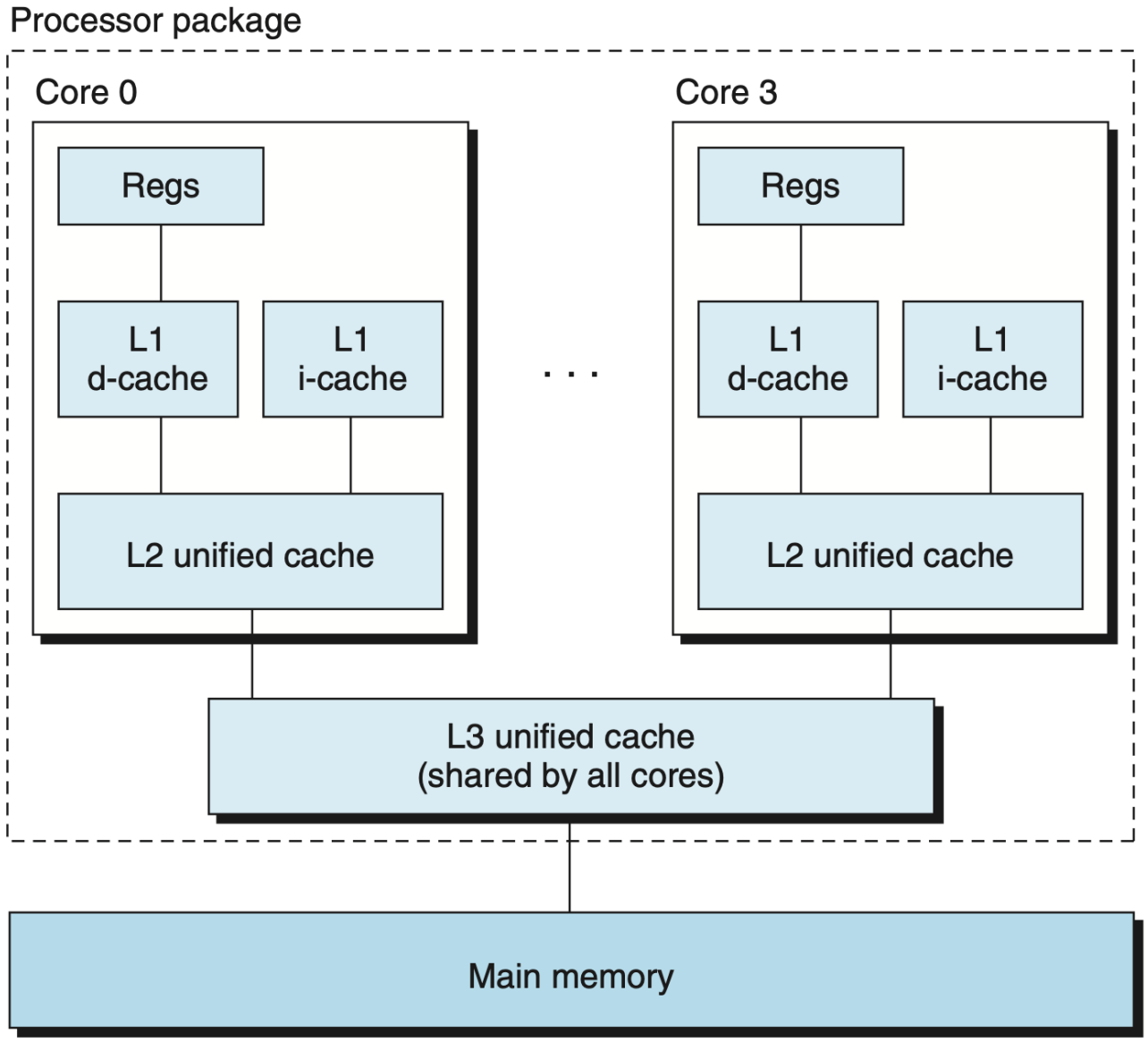
Multiprocessors (Multicore Processors)



Multiprocessors (Multicore Processors)

- Multiple CPUs on single chip
- Share main memory (and some caches)
- Each can execute a separate process
  - Scheduling of processors onto cores done by kernel
- This is called “Parallelism”

Recall: Intel Core i7 Cache Hierarchy



Multiprocessing Example

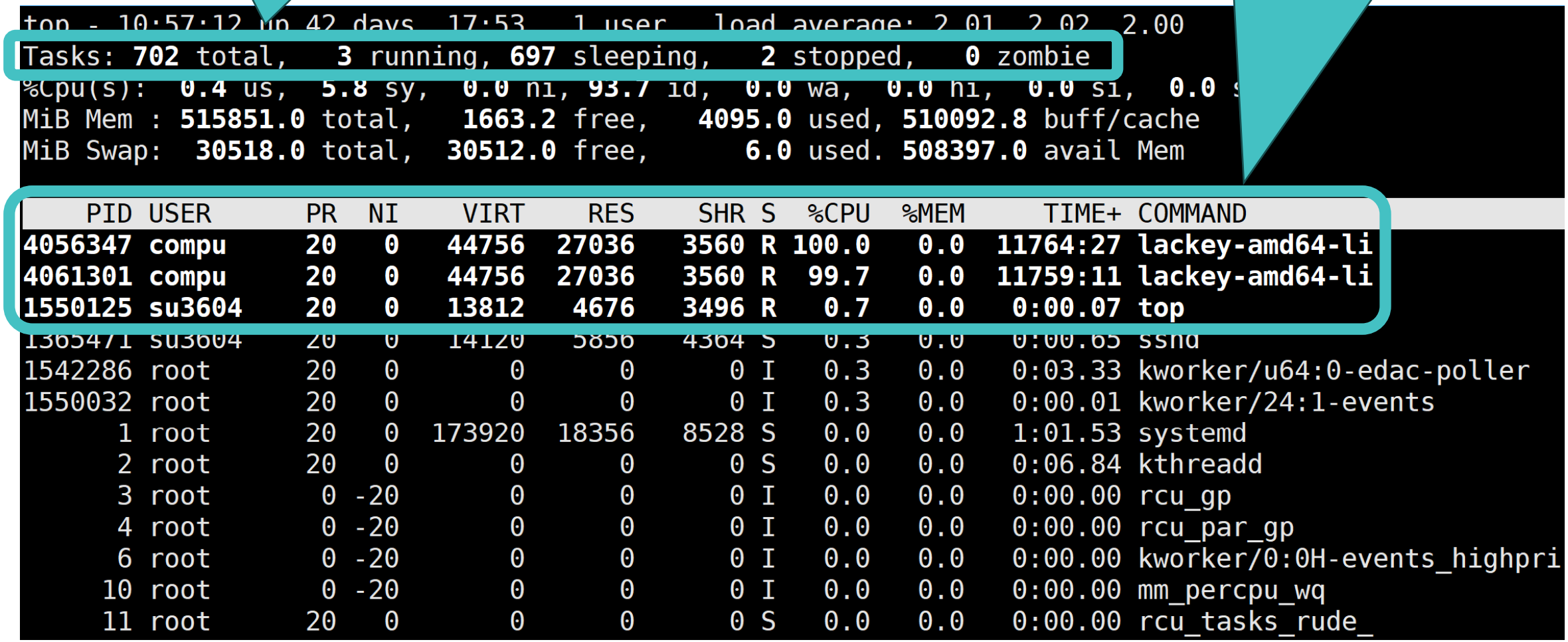
- Running “top” on your environment!

```
top - 10:57:12 up 42 days, 17:53,  1 user,  load average: 2.01, 2.02, 2.00
Tasks: 702 total,   3 running, 697 sleeping,   2 stopped,   0 zombie
%Cpu(s):  0.4 us,  5.8 sy,  0.0 ni, 93.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 515851.0 total,   1663.2 free,   4095.0 used, 510092.8 buff/cache
MiB Swap:  30518.0 total,  30512.0 free,      6.0 used. 508397.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
4056347 compu     20   0   44756  27036   3560 R 100.0   0.0  11764:27 lackey-amd64-li
4061301 compu     20   0   44756  27036   3560 R  99.7   0.0  11759:11 lackey-amd64-li
1550125 su3604    20   0   13812   4676   3496 R   0.7   0.0   0:00.07 top
1365471 su3604    20   0   14120   5856   4364 S   0.3   0.0   0:00.65 sshd
1542286 root      20   0       0      0      0 I   0.3   0.0   0:03.33 kworker/u64:0-edac-poller
1550032 root      20   0       0      0      0 I   0.3   0.0   0:00.01 kworker/24:1-events
      1 root      20   0  173920  18356   8528 S   0.0   0.0   1:01.53 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:06.84 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_rude_
```

essing Examp

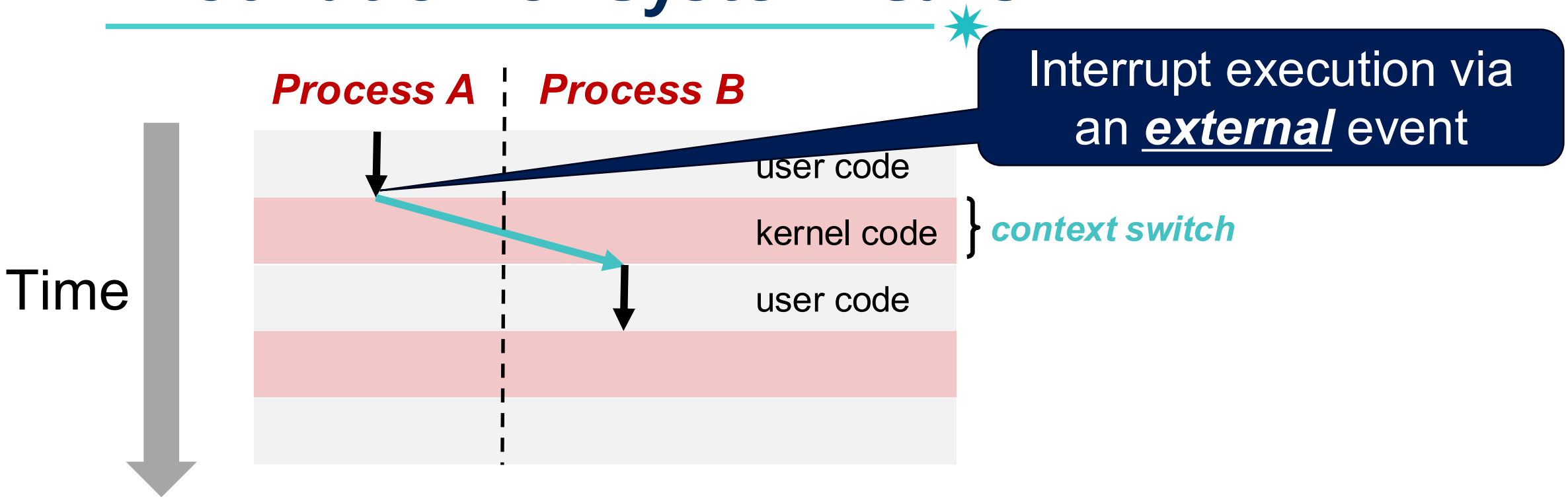
Running “top” on your environment:



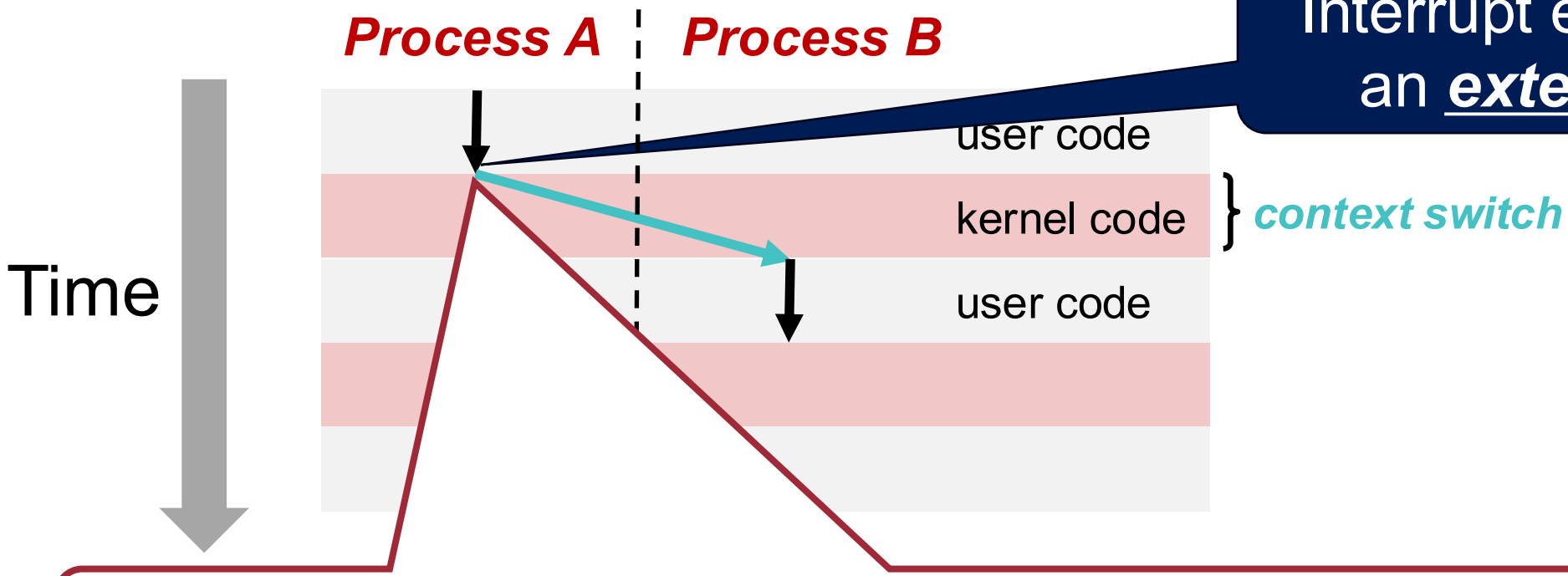
702 “tasks”, 3 of which are active

Each process is identified by Process ID (PID), user account, and command line

Motivation of System Calls



Motivation of System Calls



Interrupt execution via an ***external*** event

Sometimes, the program ***itself*** needs help from the kernel!
(e.g., print something to the screen, read/write a file, create a new process, …)

Q. How can a user program *intentionally* request a privileged operation? ➡ **System Calls!**

System Calls

System Calls

- A request to the kernel to perform a **privileged operation**

System Calls

- A request to the kernel to perform a **privileged operation**

- Whenever a program wants to cause an **effect outside its own process**, it must ask the kernel for help
- It can be called *directly from your program* or *indirectly through a library* (e.g., `printf()`)

For example,

- Read/write files
- Get current time
- Allocate RAM (sbrk)
- Create new processes

Invoke a System Call

- We can invoke a system call by executing a special instruction
  - `int 0x80` on x86 Linux
  - `syscall` on x86-64 and MIPS Linux
  - `svc` on ARM Linux
  - `ecall` on RISC-V Linux

Linux Syscall Convention

Instruction	System Call Num.	Arguments	Return
syscall	rax	rdi, rsi, rdx, r10, r8, r9	rax

```
$ cat /usr/include/asm/unistd_64.h

#ifndef _ASM_UNISTD_64_H
#define _ASM_UNISTD_64_H

#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
...
```

For example, the `printf()` function contains the following assembly code:

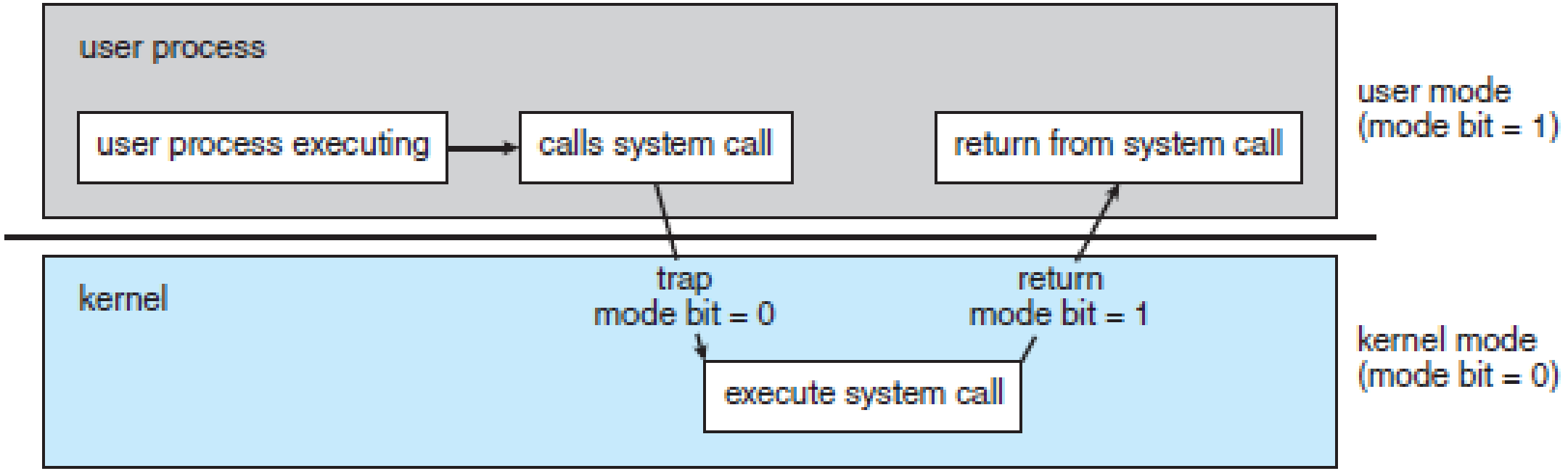
```
printf:
    ...
    mov  rax, 1
    syscall
    ...
```

All the System Calls

accept
accept4
acct
add_key
adjtimex
bind
bpf
brk
capget
capset
chdir
chroot
clock_adjtime
clock_getres
clock_gettime
clock_nanosleep
clock_settime
clone
clone3
close
close_range
connect
copy_file_range
delete_module
dup
dup3
epoll_create1
epoll_ctl
epoll_pwait
epoll_pwait2
eventfd2
execve
execveat
exit
exit_group
faccessat
faccessat2
fallocate
fanotify_init
fanotify_mark
fchdir
fchmod
fchmodat
fchown
fchownat
fdatasync
fgetxattr
finit_module
flistxattr
flock
fremovexattr
fsconfig
fsetxattr
fsmount
fsopen
fspick
fsync
futex
futex_waitv
get_mempolicy
get_robust_list
getcpu
getcwd
getdents64
getegid
geteuid
getgid
getgroups
getitimer
getpeername
getpgid
getpid
getppid
getpriority
getrandom
getresgid
getresuid
getrlimit
getrusage
getsid
getsockname
getsockopt
gettid
gettimeofday
getuid
getxattr
init_module
inotify_add_watch
inotify_init1
inotify_rm_watch
io_cancel
io_destroy
io_getevents
io_pgetevents
io_setup
io_submit
io_uring_enter
io_uring_register
io_uring_setup
ioctl
ioprio_get
ioprio_set
kcmp
kexec_file_load
kexec_load
keyctl
kill
landlock_add_rule
landlock_create_ruleset
landlock_restrict_self
lgetxattr
linkat
listen
listxattr
llistxattr
lookup_dcookie
lremovexattr
lsetxattr
madvise
mbind
membarrier
memfd_create
memfd_secret
migrate_pages
mincore
mkdirat
mknodat
mlock
mlock2
mlockall
mount
mount_setattr
move_mount
move_pages
mprotect
mq_getsetattr
mq_notify
mq_open
mq_timedreceive
mq_timedsend
mq_unlink
mremap
msgctl
msgget
msgrcv
msgsnd
msync
munlock
munlockall
munmap
name_to_handle_at
nanosleep
nfsservctl
open_by_handle_at
open_tree
openat
openat2
perf_event_open
personality
pidfd_getfd
pidfd_open
pidfd_send_signal
pipe2
pivot_root
pkey_alloc
pkey_free
pkey_mprotect
ppoll
prctl
pread64
preadv
preadv2
prlimit64
process_madvise
process_mrelease
process_vm_readv
process_vm_writev
pselect6
ptrace
pwrite64
pwritev
pwritev2
quotactl
quotactl_fd
read
readahead
readlinkat
readv
reboot
recvfrom
recvmmsg
recvmsg
remap_file_pages
removexattr
renameat
renameat2
request_key
restart_syscall
rseq
rt_sigaction
rt_sigpending
rt_sigprocmask
rt_sigqueueinfo
rt_sigreturn
rt_sigsuspend
rt_sigtimedwait
rt_tgsigqueueinfo
sched_get_priority_max
sched_get_priority_min
sched_getaffinity
sched_getattr
sched_getparam
sched_getscheduler
sched_rr_get_interval
sched_setaffinity
sched_setattr
sched_setparam
sched_setscheduler
sched_yield
seccomp
semctl
semget
semop
semtimedop
sendmmsg
sendmsg
sendto
set_mempolicy
set_mempolicy_home_node
set_robust_list
set_tid_address
setdomainname
setfsgid
setfsuid
setgid
setgroups
sethostname
setitimer
setns
setpgid
setpriority
setregid
setresgid
setresuid
setreuid
setrlimit
setsid
setsockopt
settimeofday
setuid
setxattr
shmat
shmctl
shmdt
shmget
shutdown
sigaltstack
signalfd4
socket
socketpair
splice
statx
swapoff
swapon
symlinkat
sync
sync_file_range
sync_file_range2
syncfs
sysinfo
syslog
tee
tgkill
timer_create
timer_delete
timer_getoverrun
timer_gettime
timer_settime
timerfd_create
timerfd_gettime
timerfd_settime
times
tkill
umask
umount2
uname
unlinkat
unshare
userfaultfd
utimensat
vhangup
vmsplice
wait4
waitid
write
writev

System Call

- The programmatic way in which a computer program requests a service from the operating system



Question?