CSE251: System Programming

25. Threads

Seongil Wi

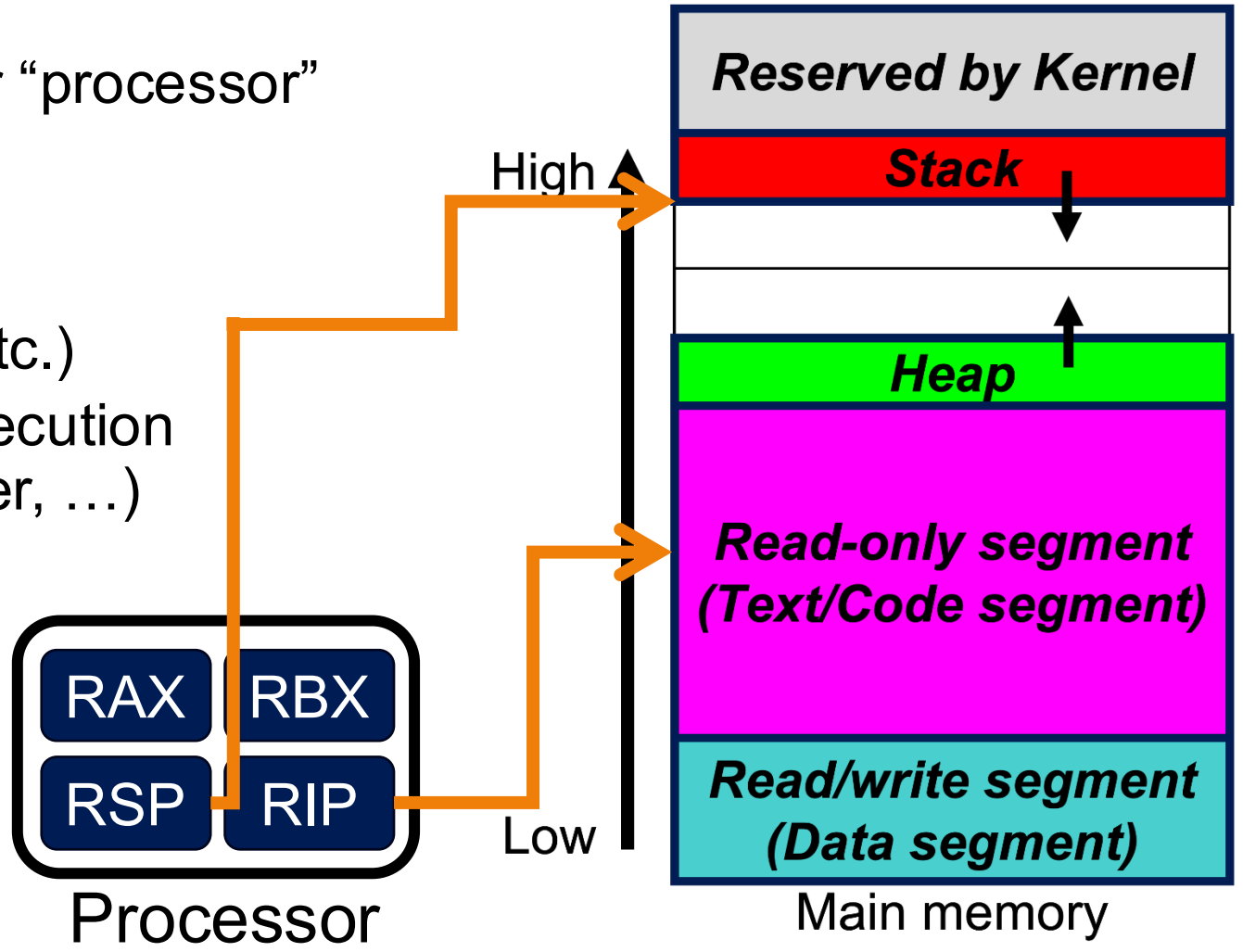
Department of Computer Science and Engineering

Recap: Process

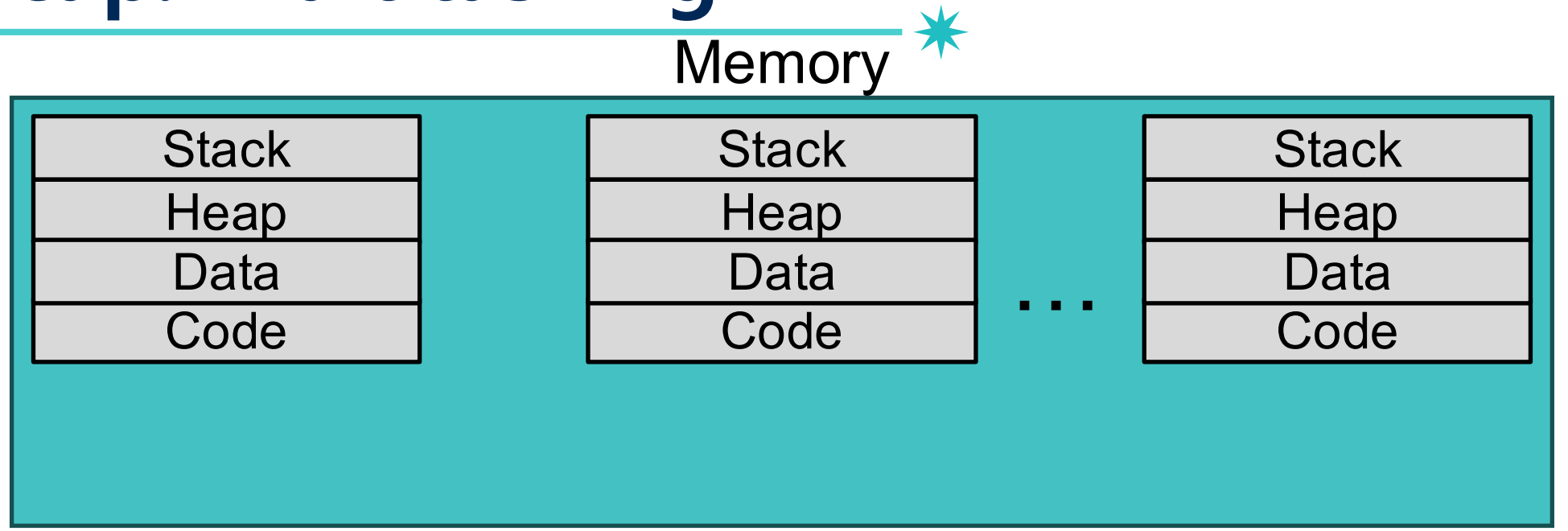
An **instance of a program** that is being executed (or is ready for execution)

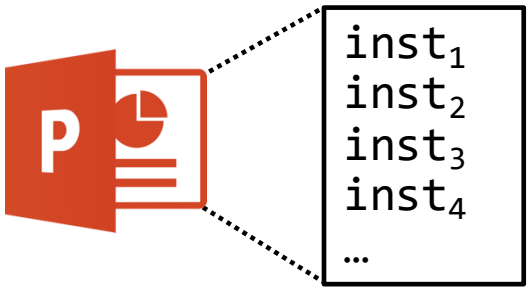
- Not the same as “program” or “processor”

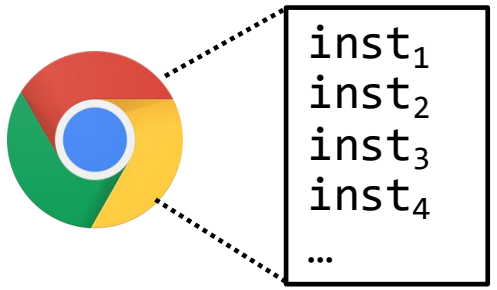
- Consists of:
  - Memory (code, heap, stack, etc.)
  - Registers used to manage execution (stack pointer, program counter, …)
  - Other resources

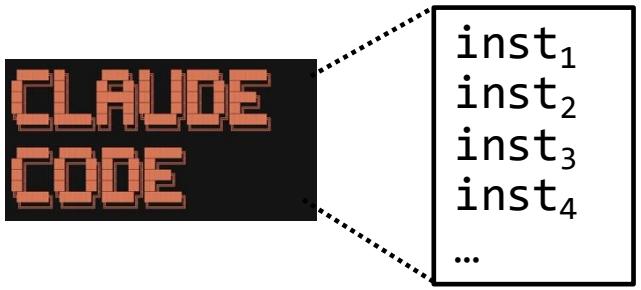


Recap: Multitasking

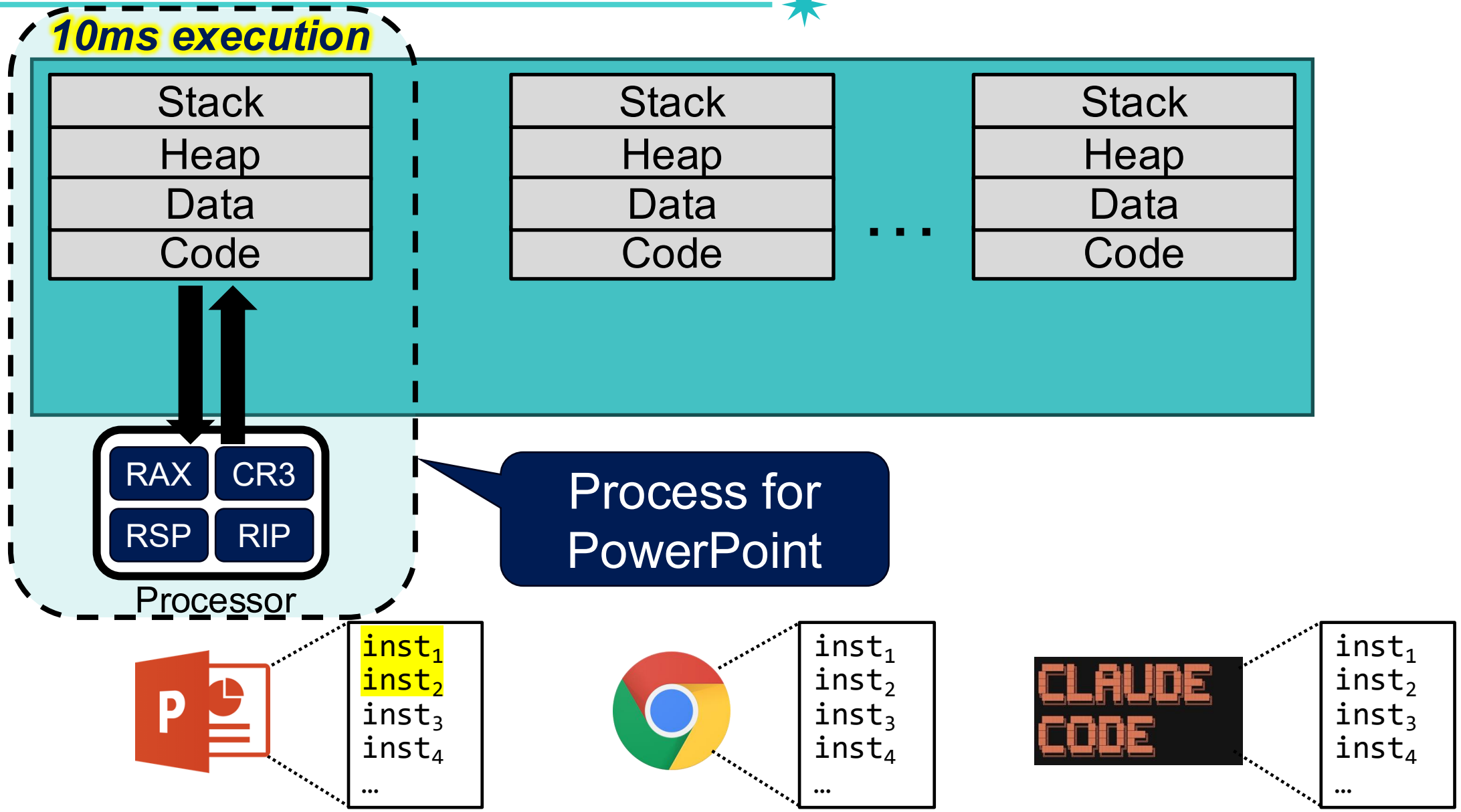




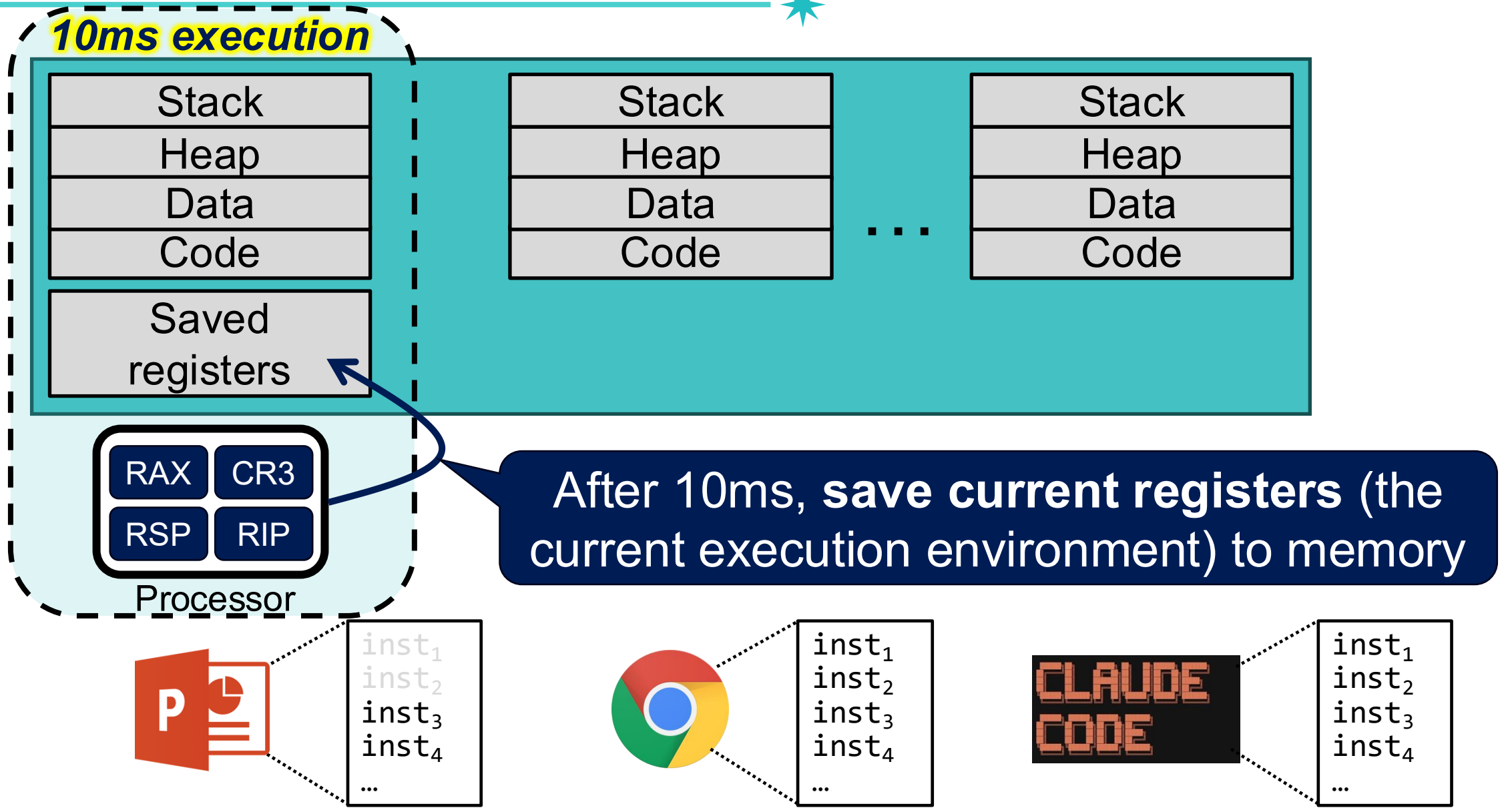




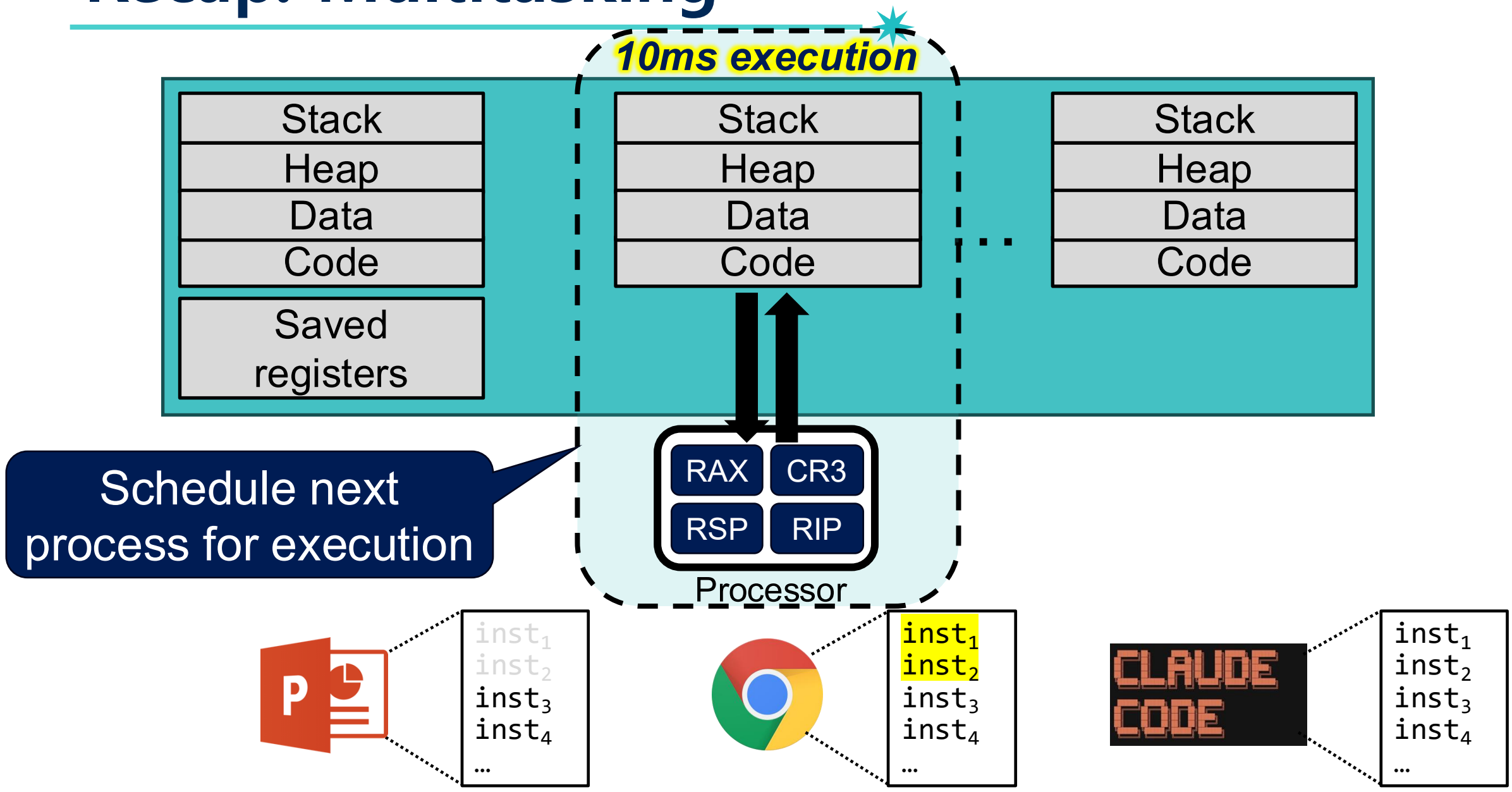
Recap: Multitasking



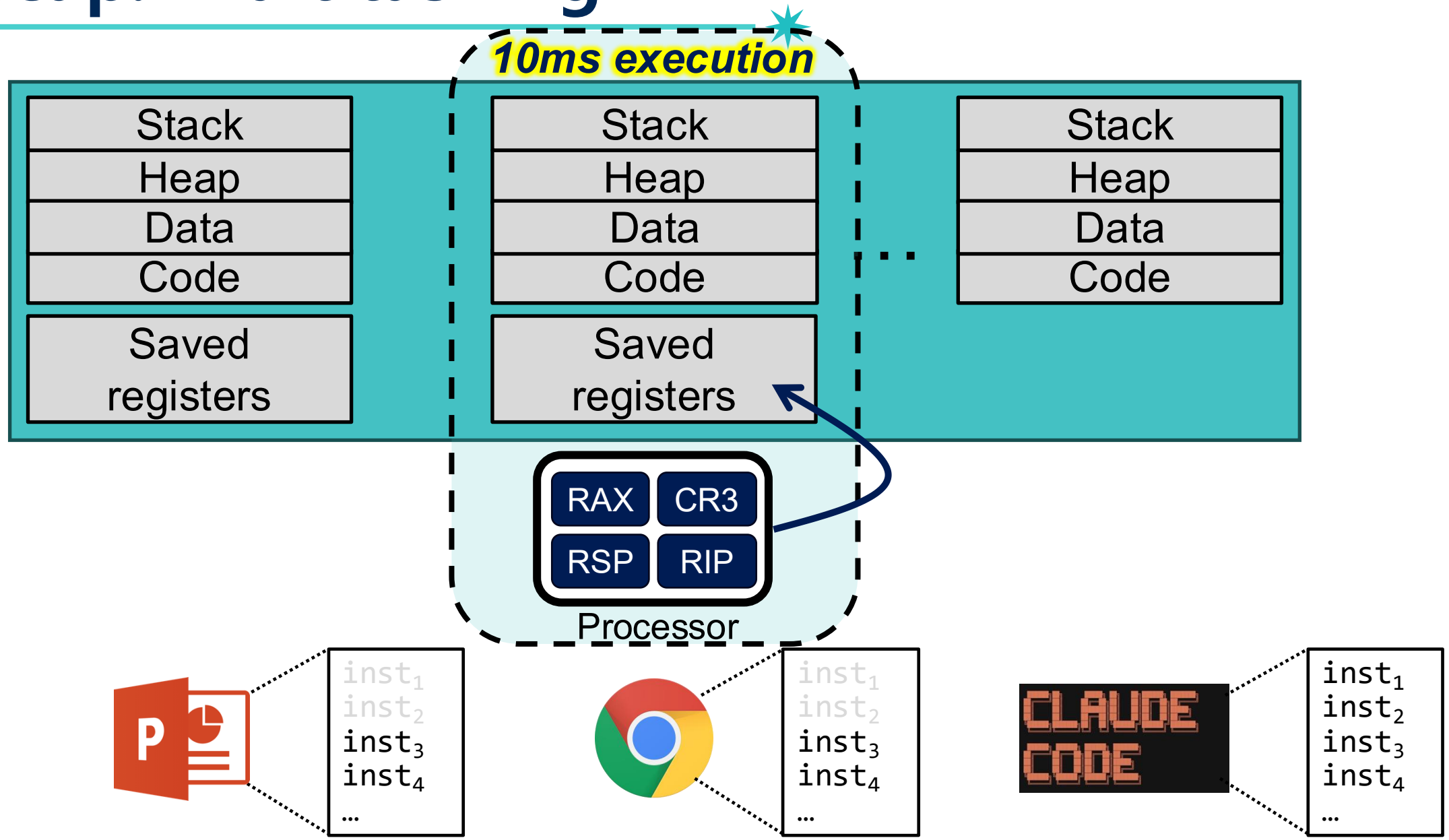
Recap: Multitasking



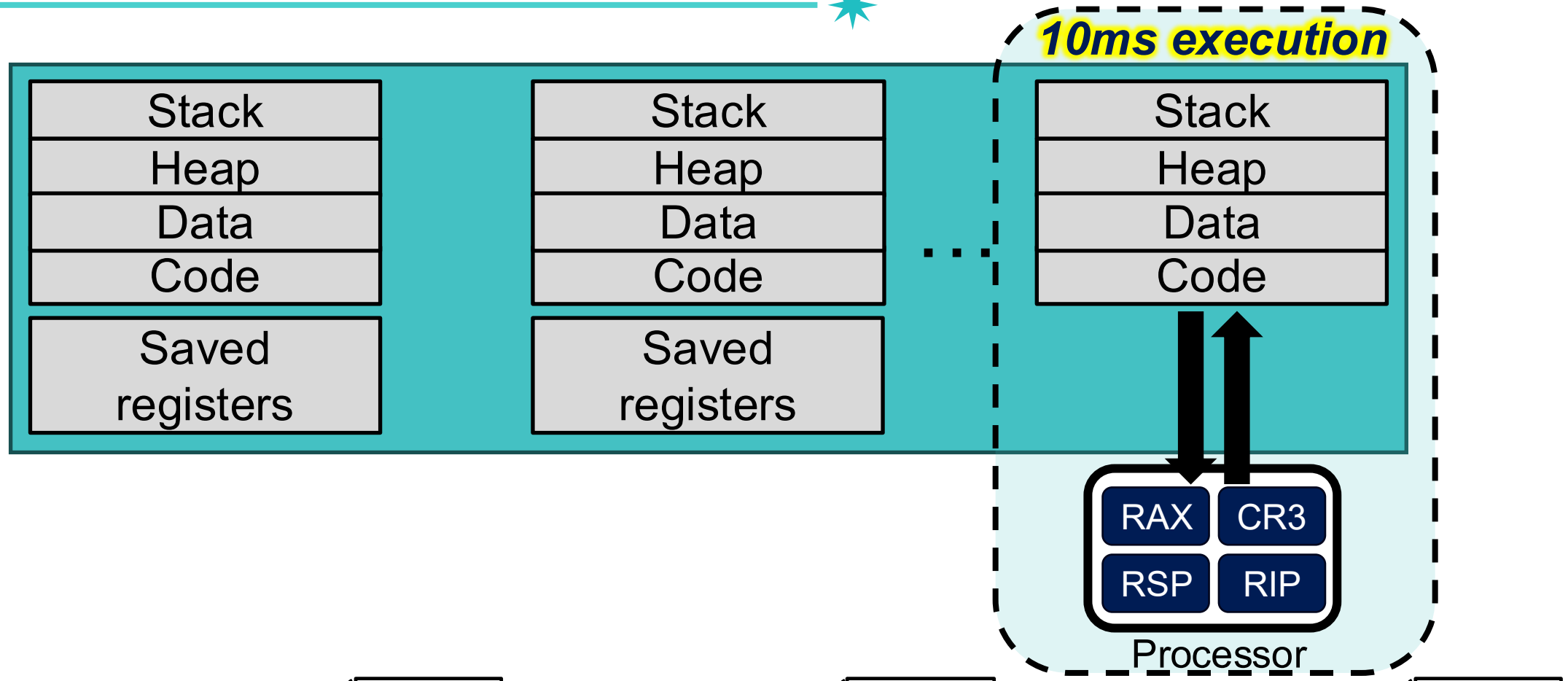
Recap: Multitasking



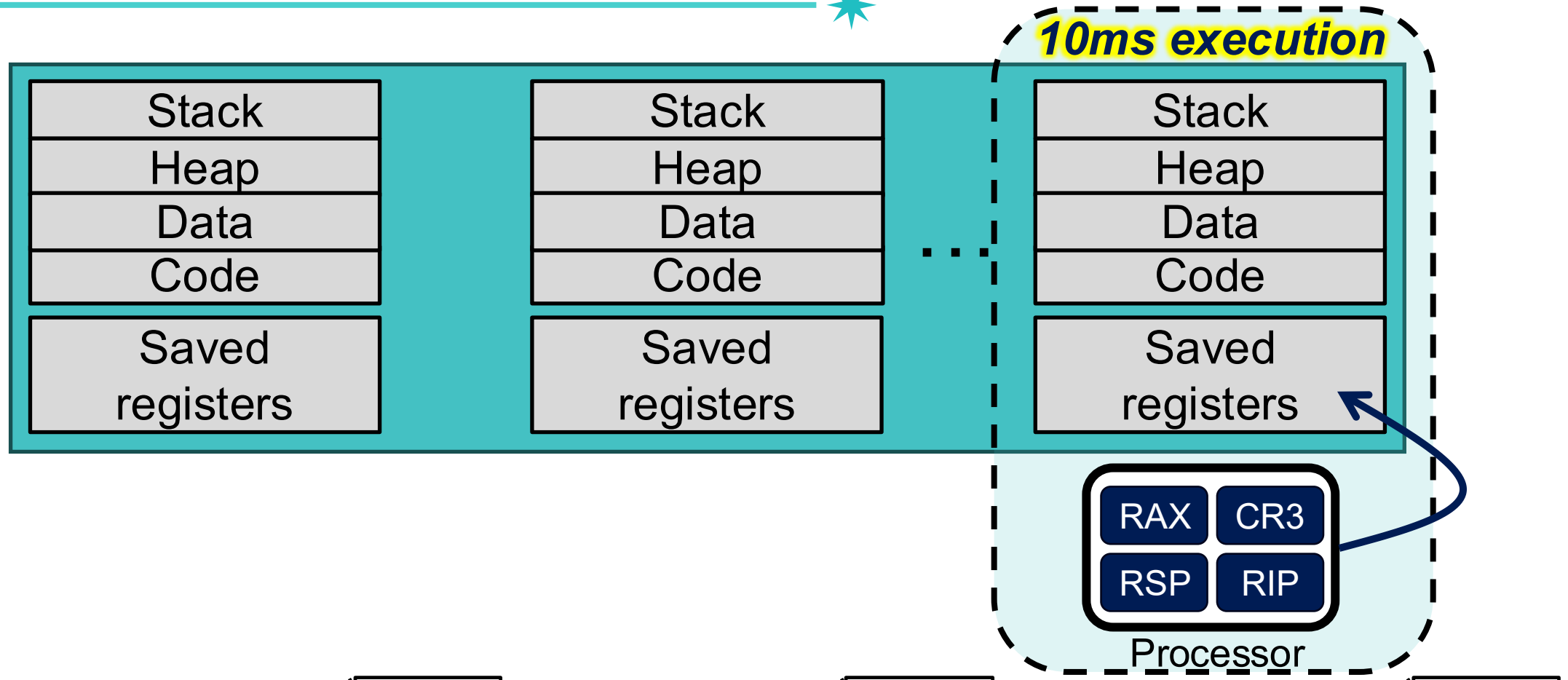
Recap: Multitasking



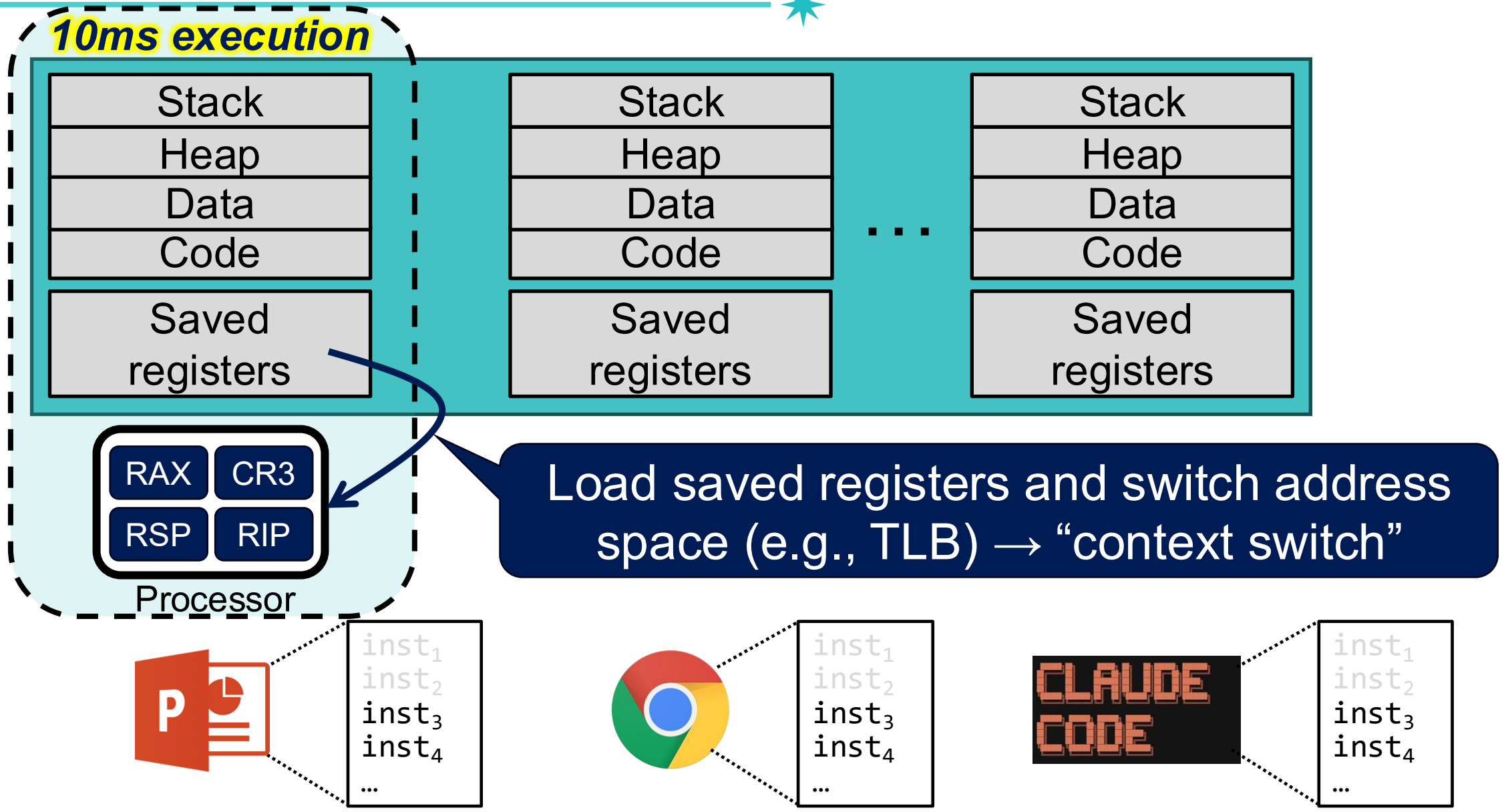
Recap: Multitasking



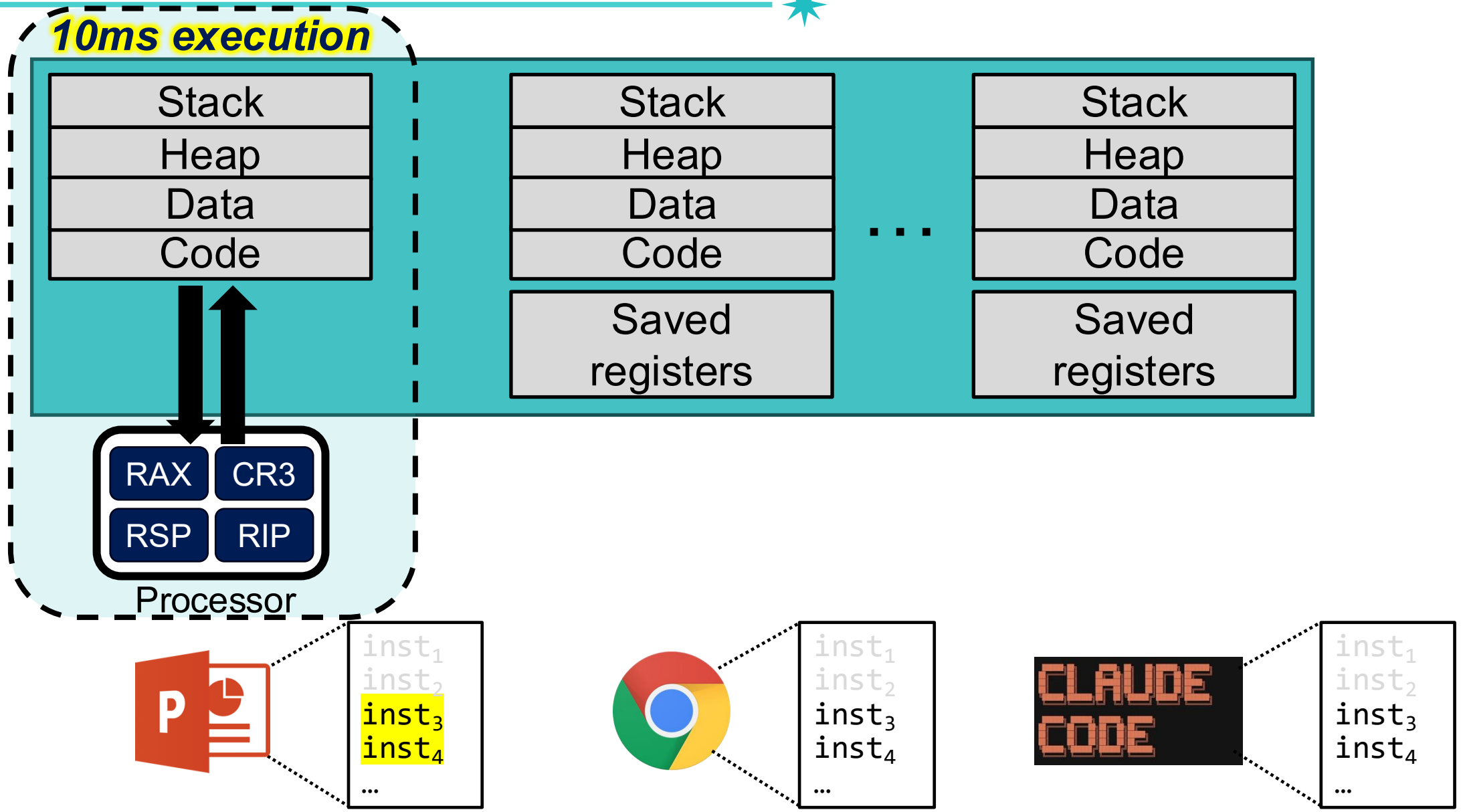
Recap: Multitasking



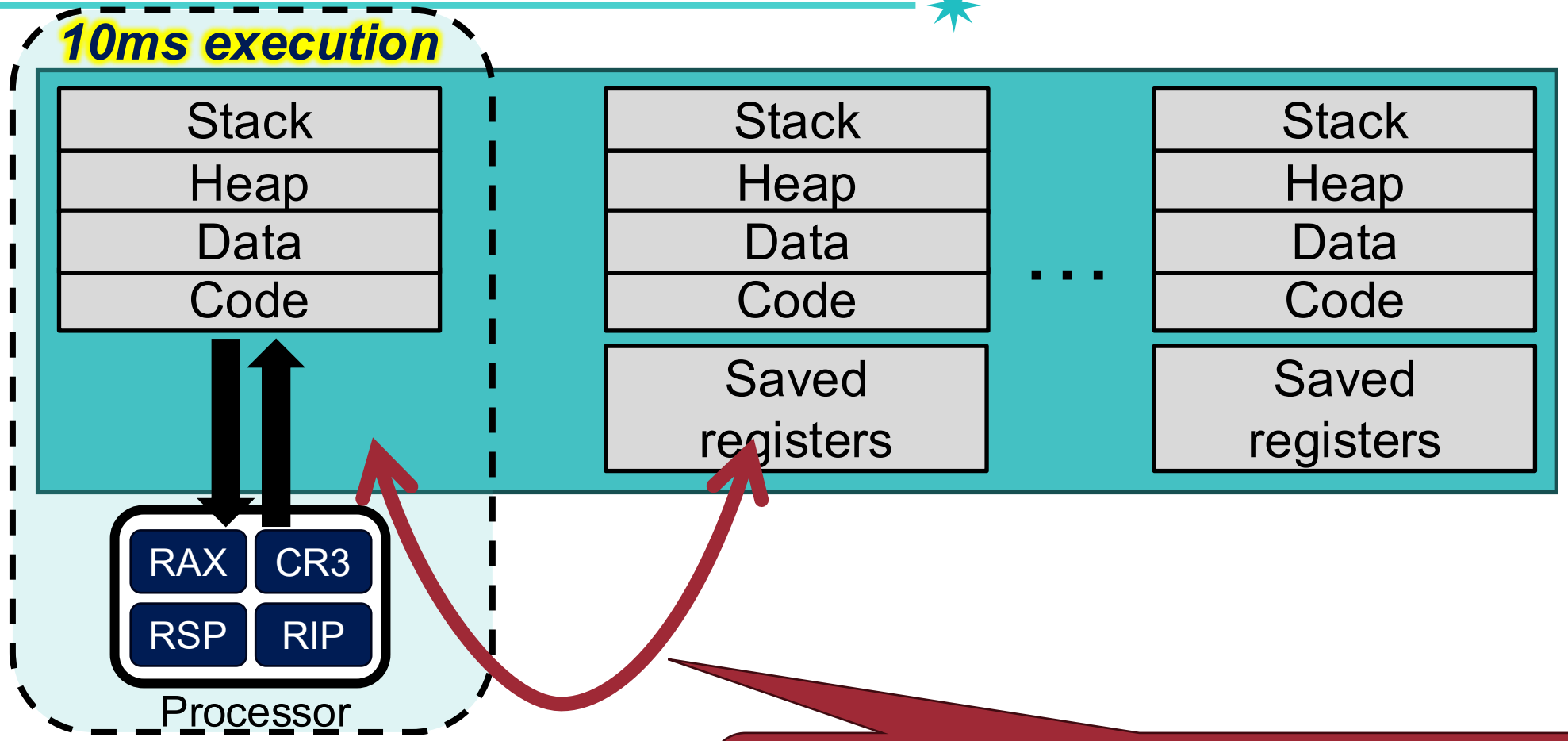
Recap: Multitasking



Recap: Multitasking

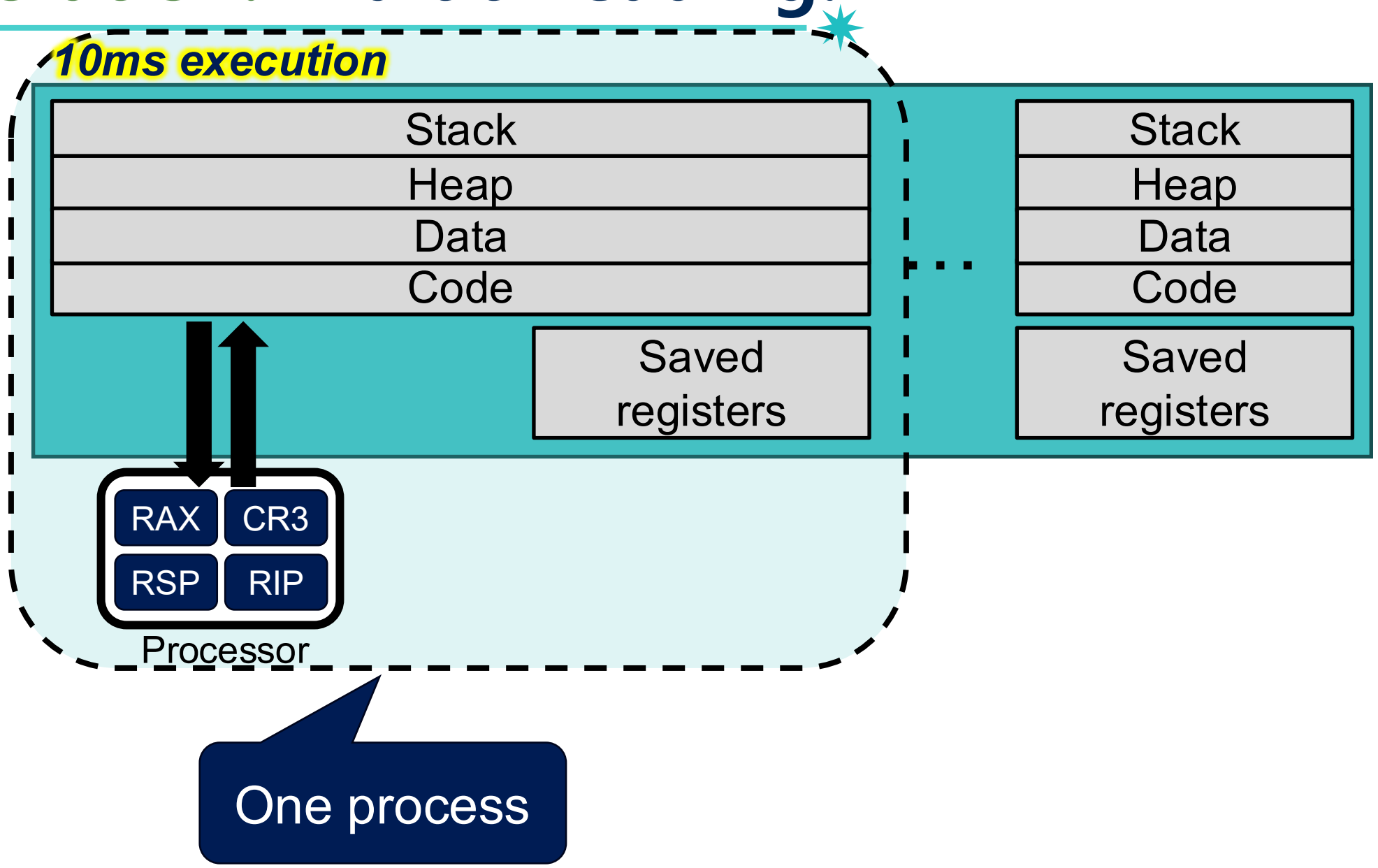


Main Limitation of Multitasking

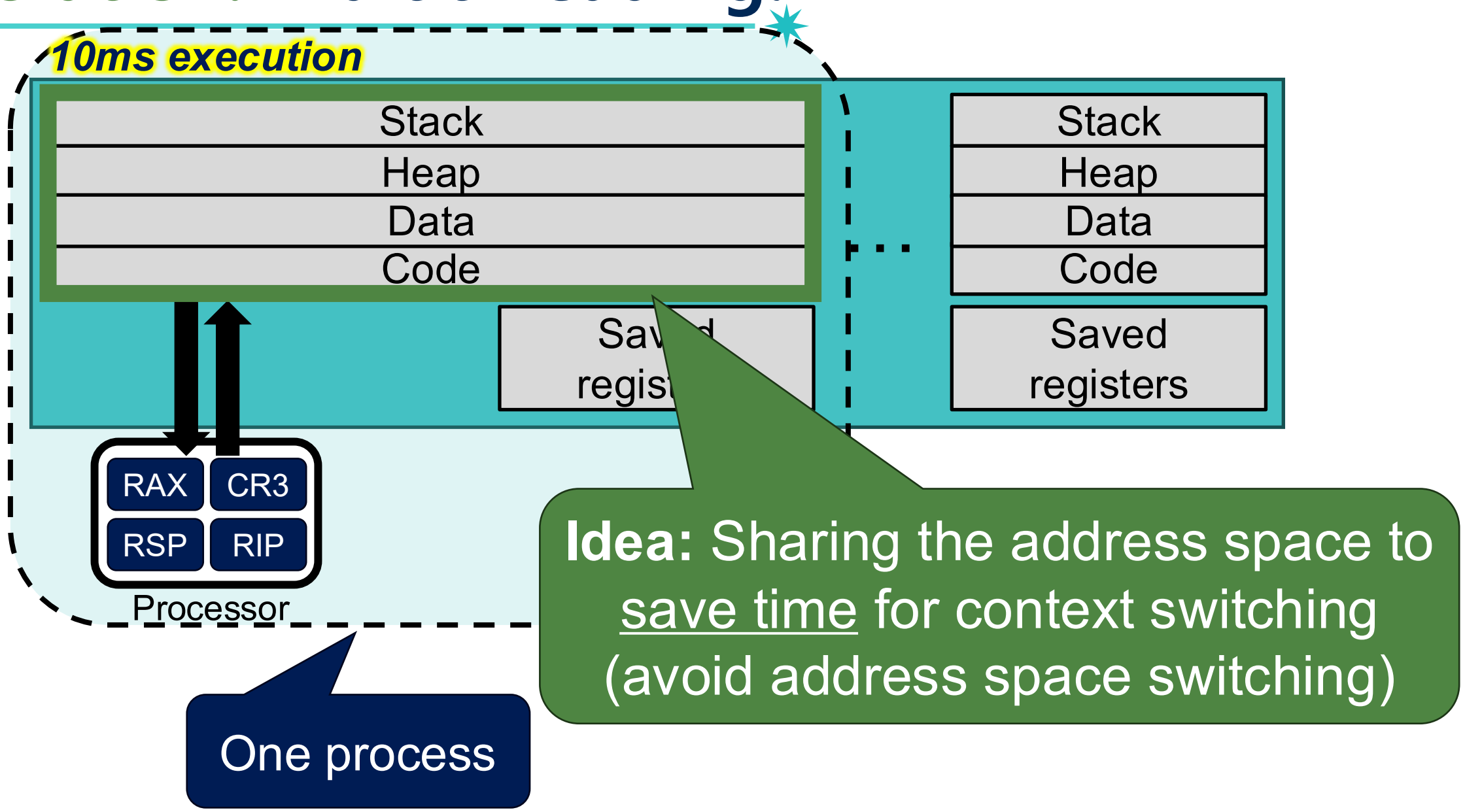


Context switching takes a lot of time!

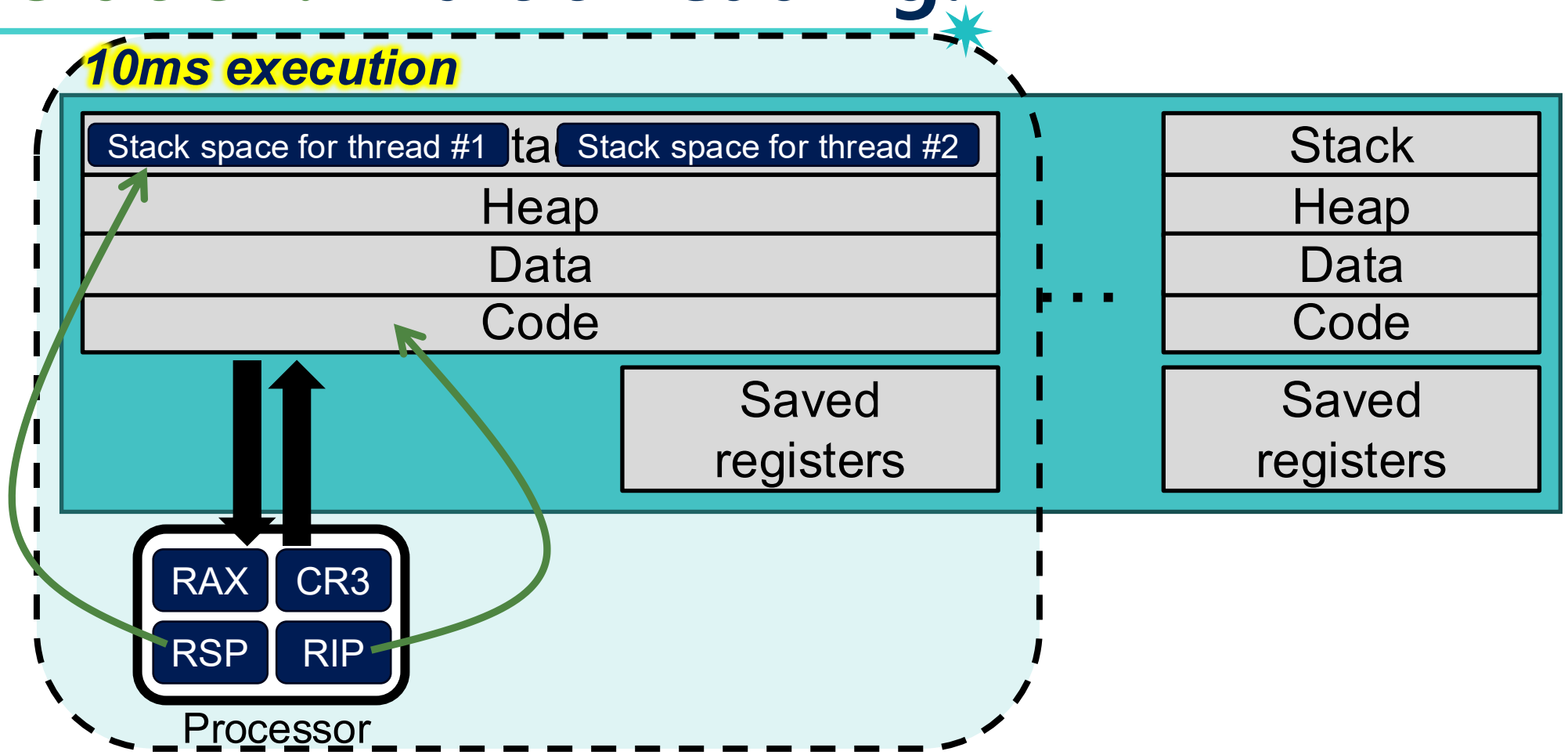
Solution: Multithreading!



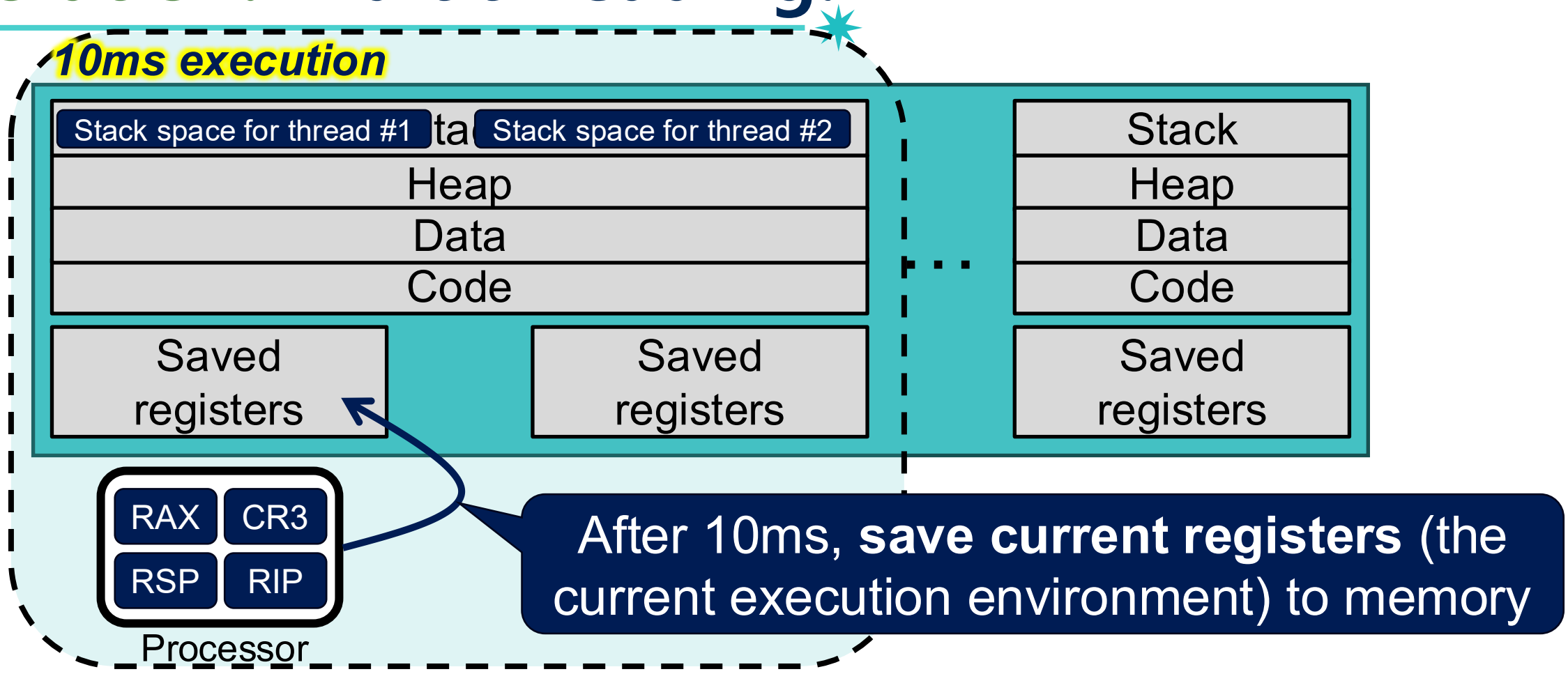
Solution: Multithreading!



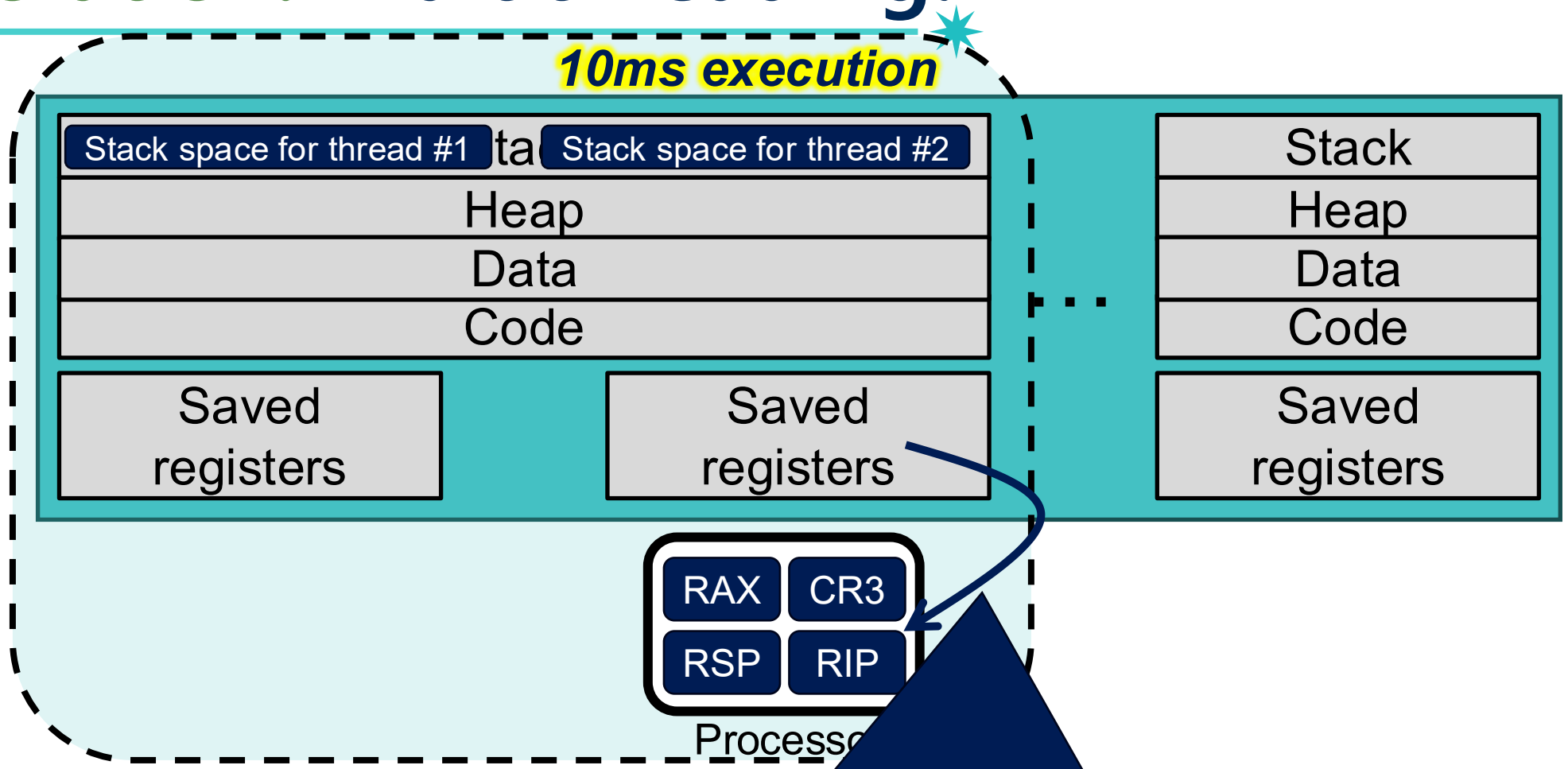
Solution: Multithreading!



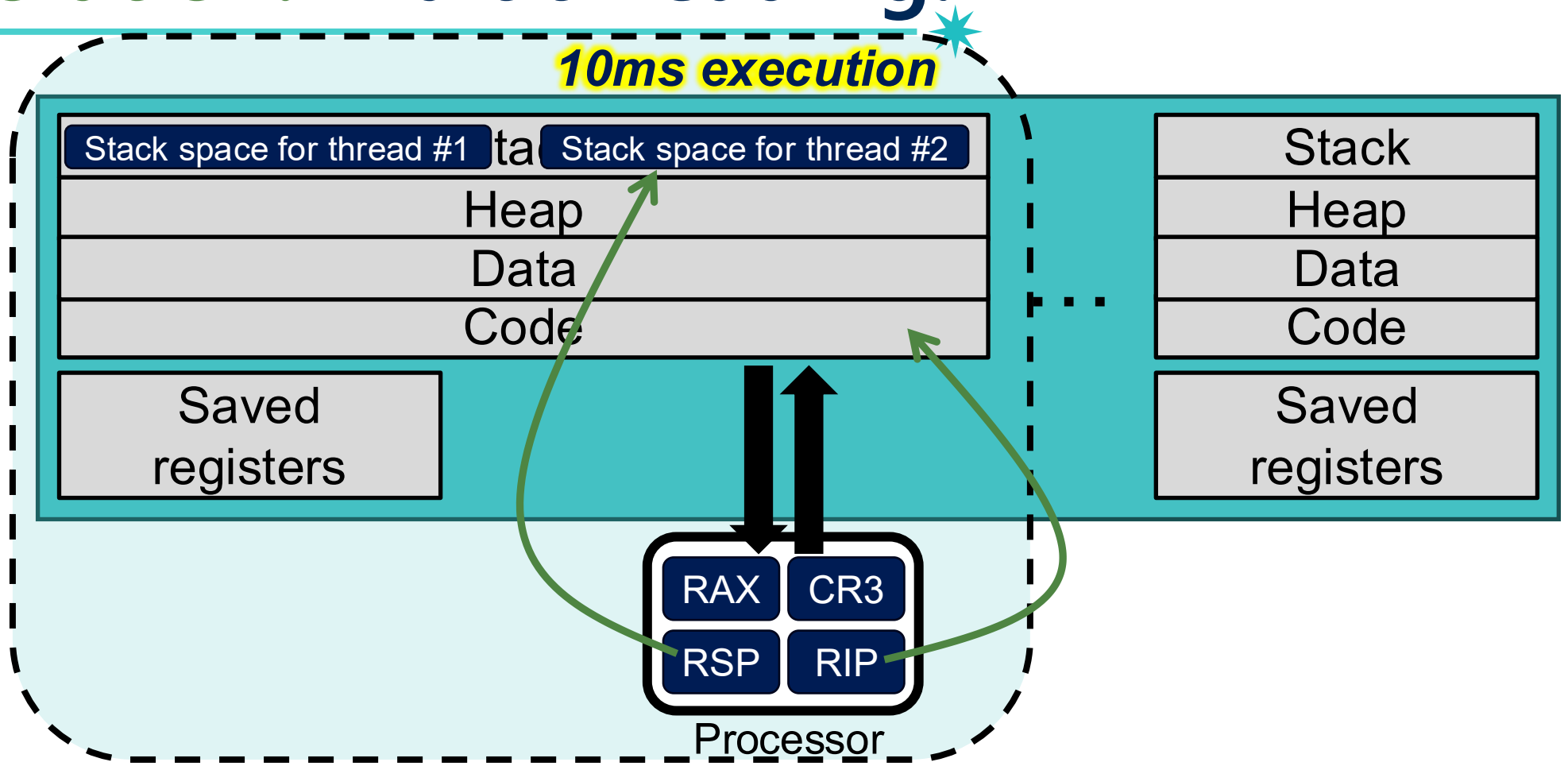
Solution: Multithreading!



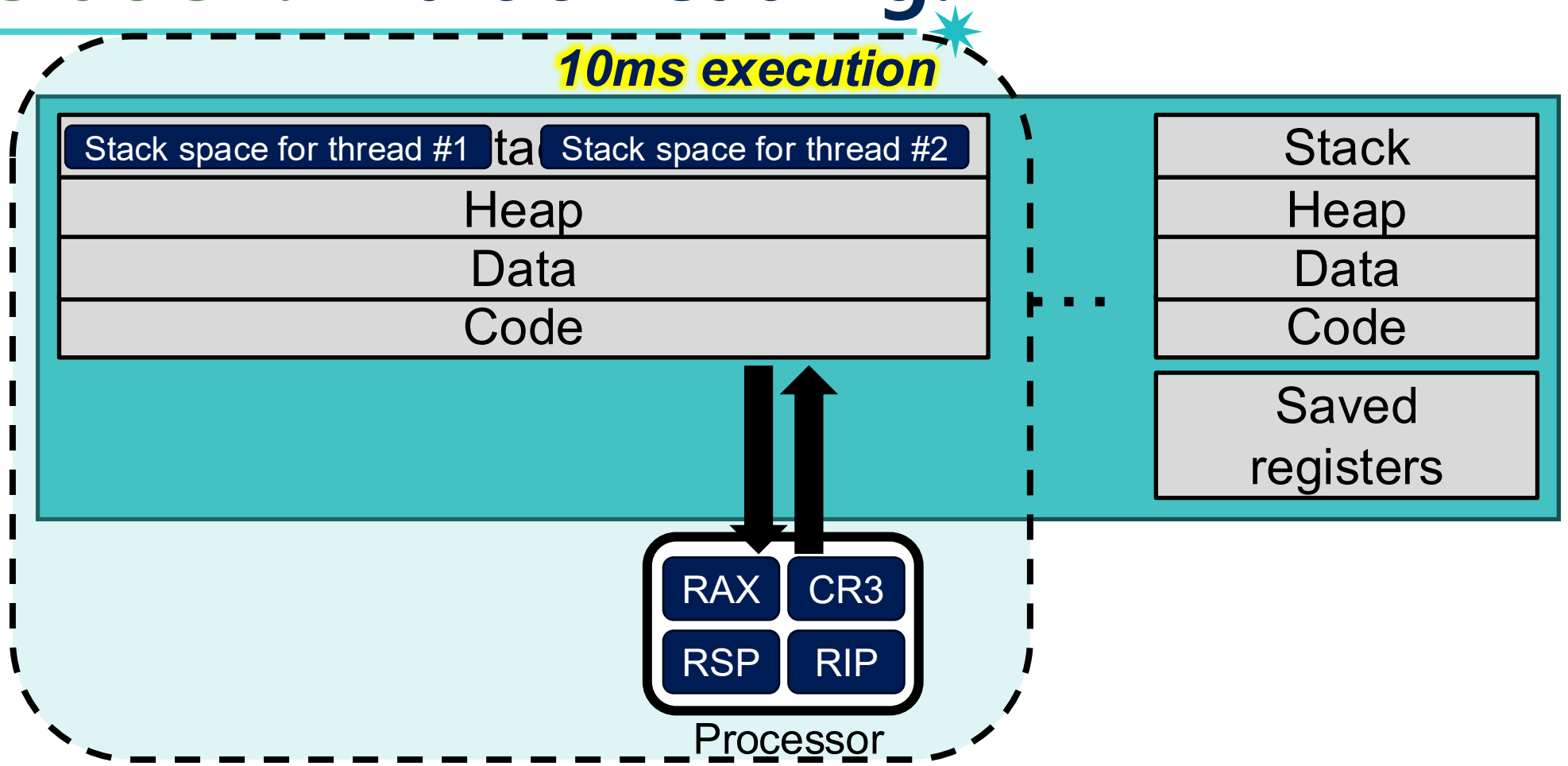
Solution: Multithreading!



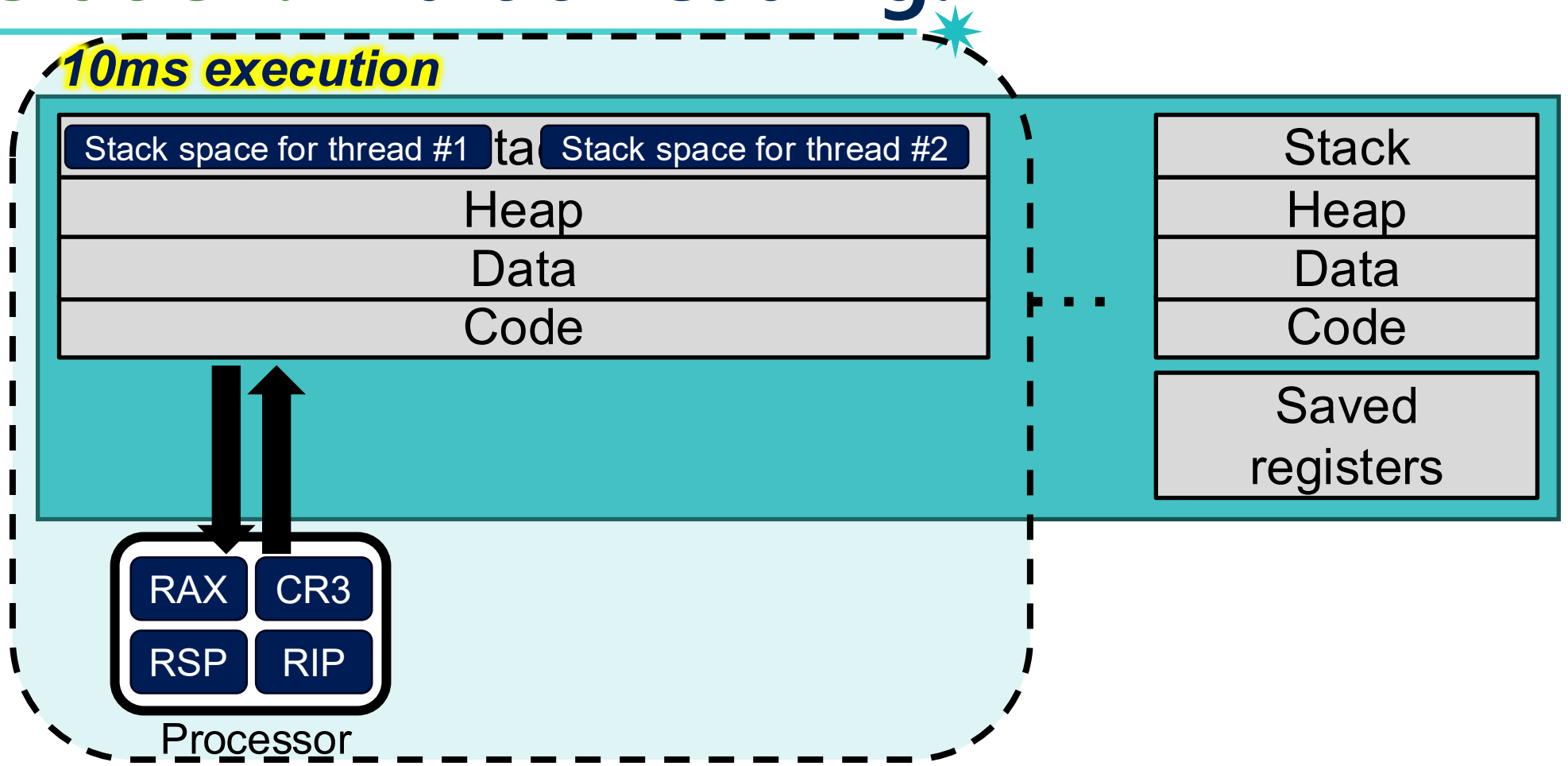
Solution: Multithreading!



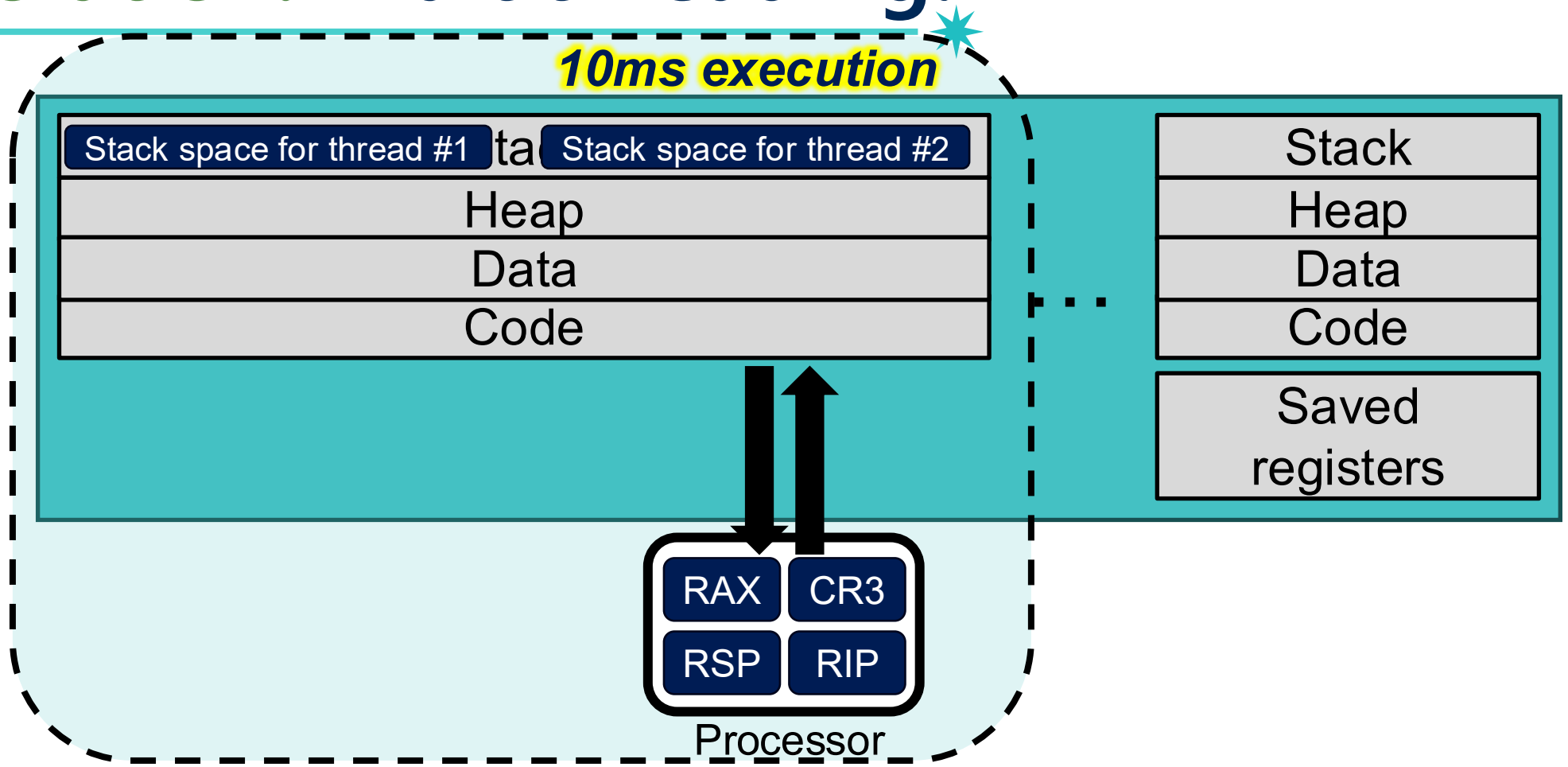
Solution: Multithreading!



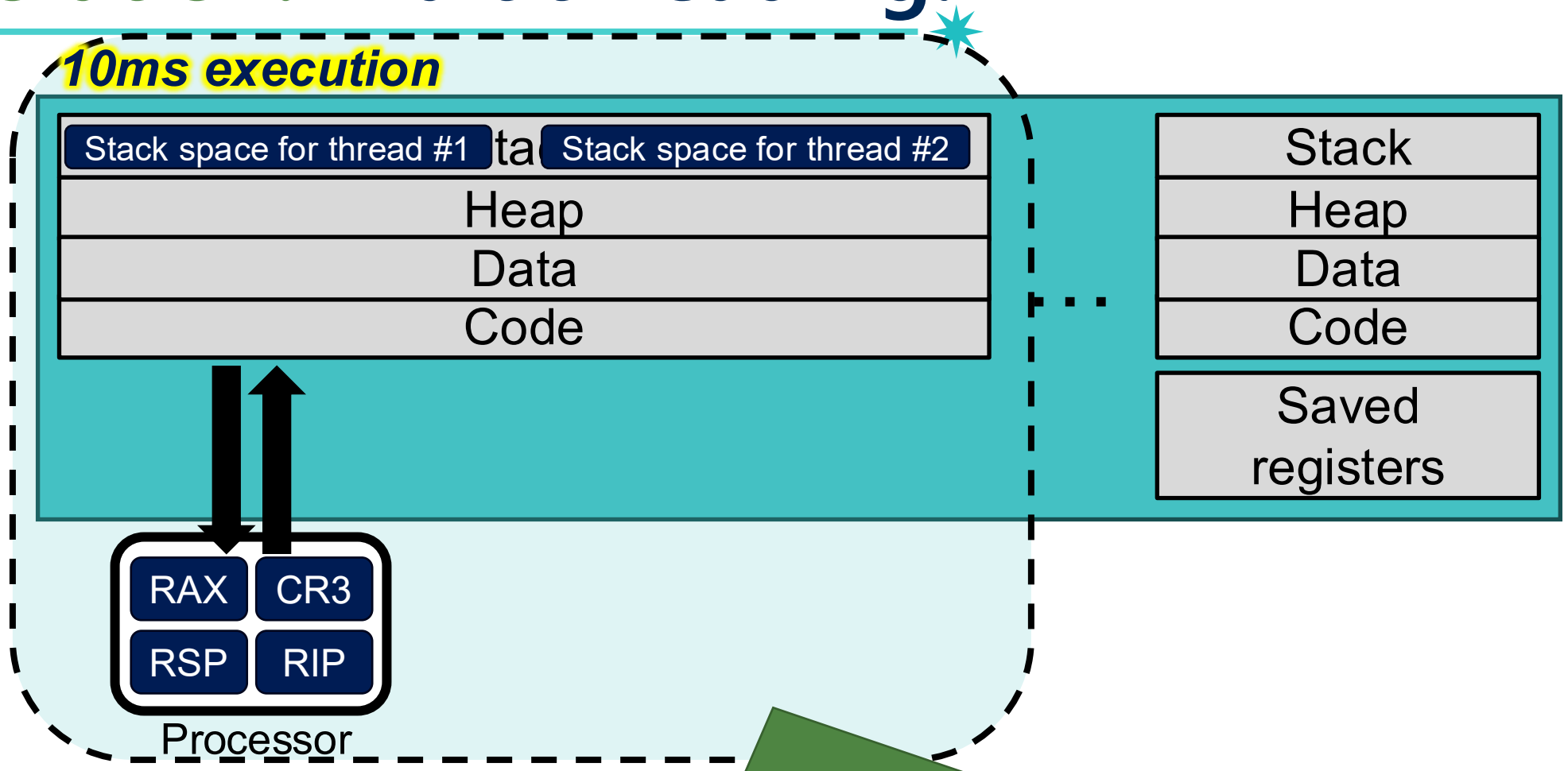
Solution: Multithreading!



Solution: Multithreading!

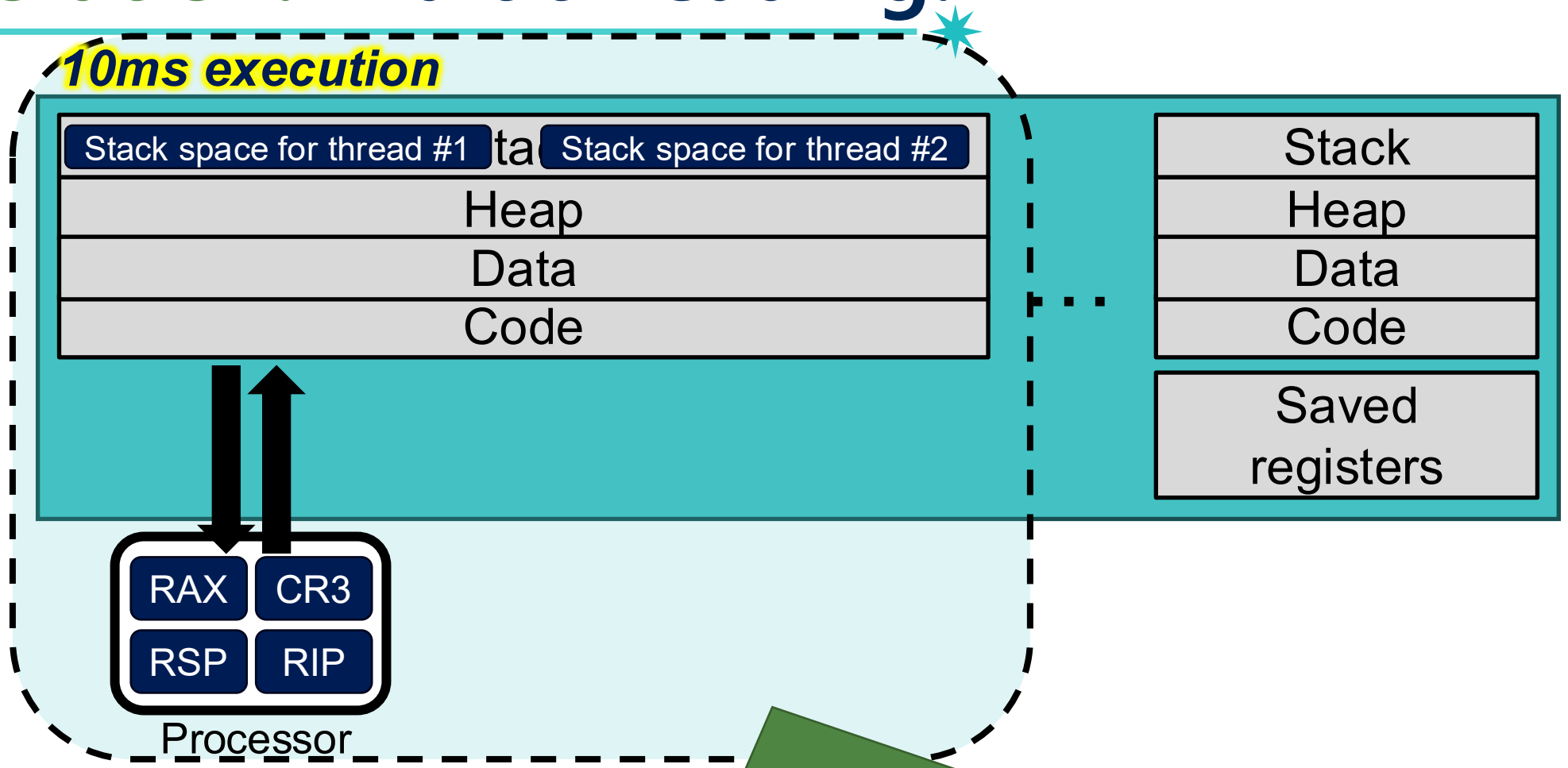


Solution: Multithreading!



Illusion! Two execution flows (= ***threads***) can run concurrently (within a single process)

Solution: Multithreading!



Multithreading!

Threads

Thread

- An execution flow within **a process**

Thread

- An execution flow within **a process**
- Threads **share the same address space (save time!)** but have their own execution context

Thread

- An execution flow within **a process**
- Threads **share the same address space (save time!)** but have their own execution context

- For example, …

Chrome
(Single process)

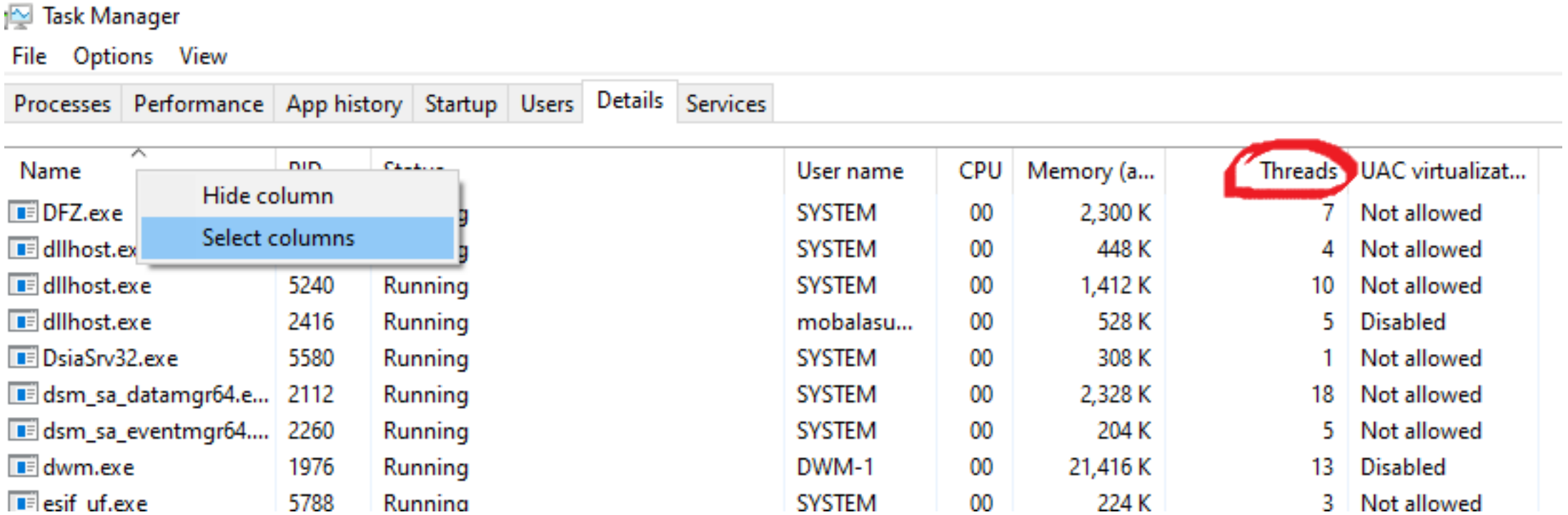
Thread #1: Play video

Thread #2: Load image

Thread #3: Download file

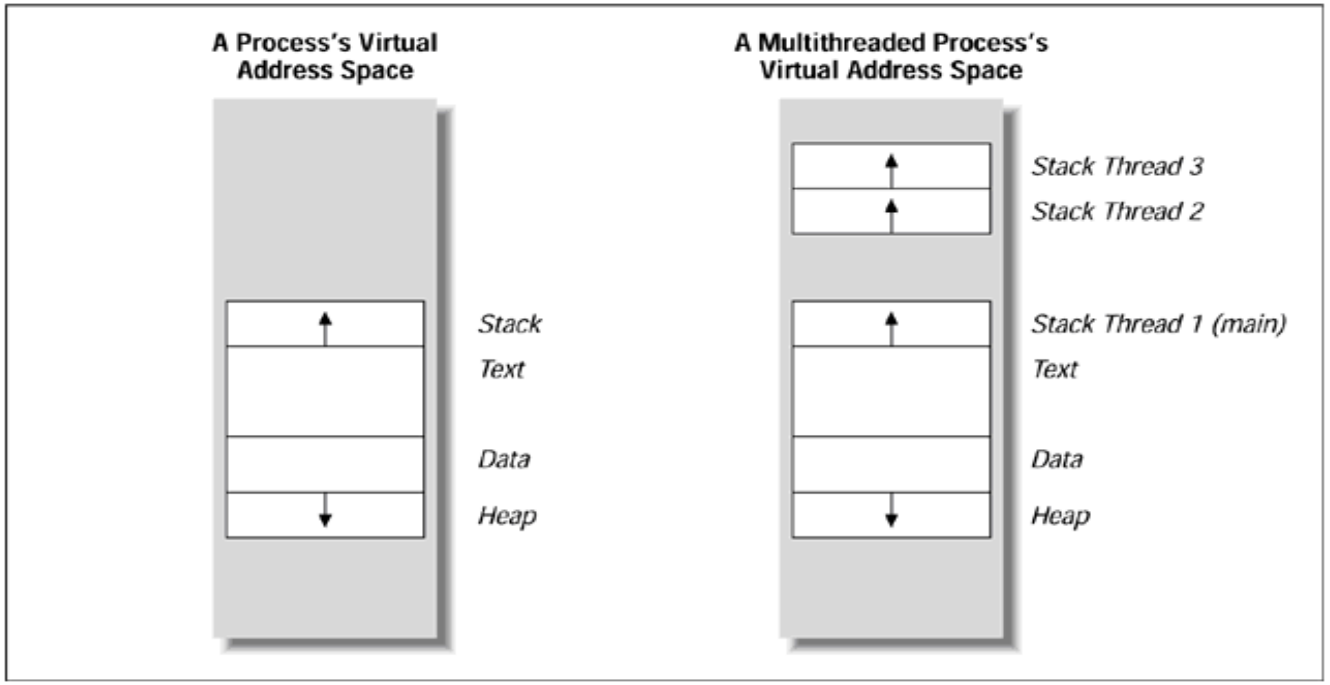
A Process with Multiple Threads

- **Multiple threads** can be associated with **a process**



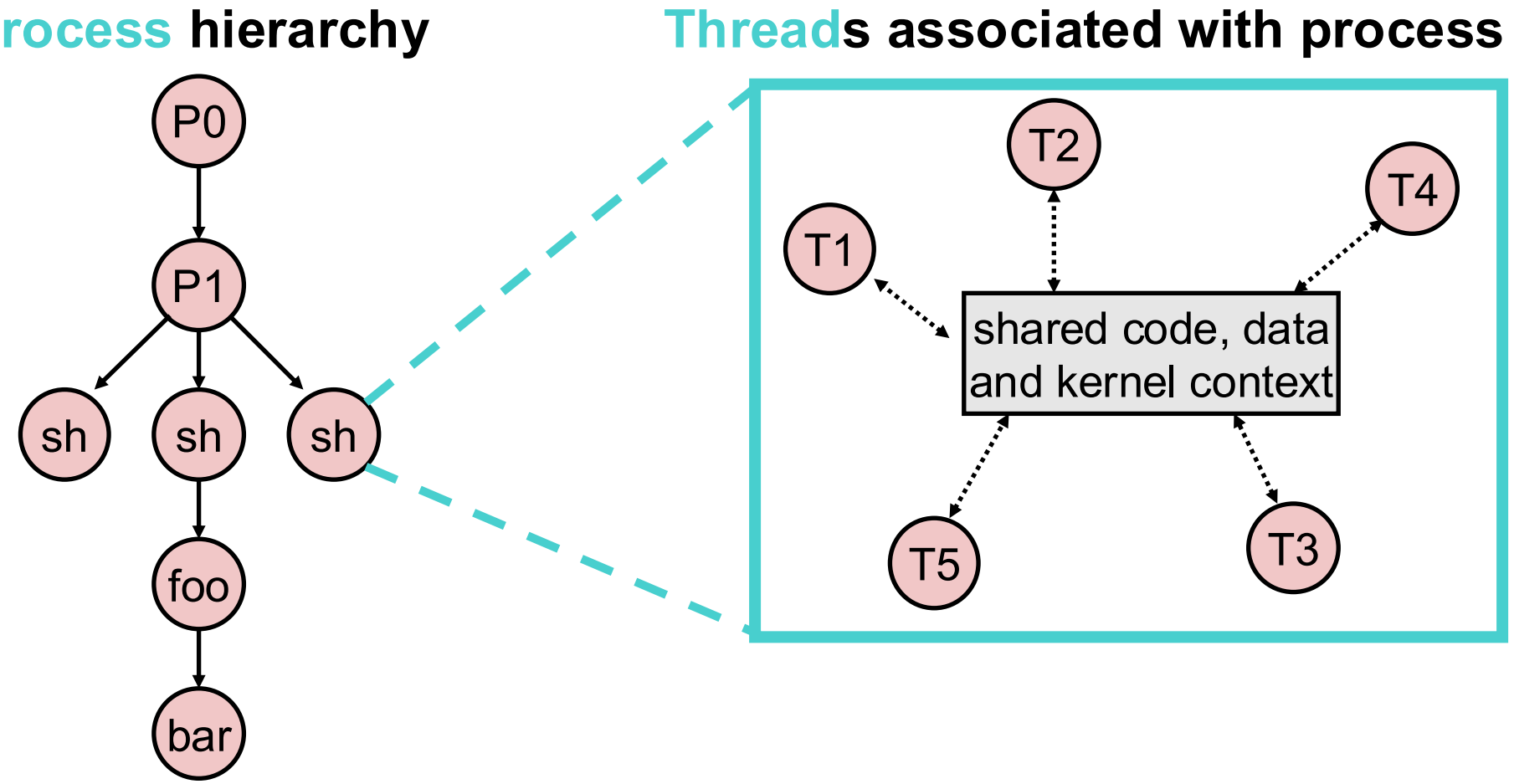
A Process with Multiple Threads

- **Multiple threads** can be associated with **a process**
  - Each thread has its own
    - logical control flow (with different RIP)
    - stack space (with different RSP, RBP) (but not protected from other threads – because they share the same address space!)
    - thread id (TID)
  - Threads share the same
    - code, data, and kernel context



Logical View of Threads

- Threads associated with process form a pool of peers
  - Unlike processes which form a tree hierarchy



★ Threads vs. Processes

- How threads and processes are **similar**
  - Each has its own logical control flow
  - Each can run concurrently with others (possibly on different cores)
  - Each is context switched

- How threads and processes are **different**
  - Threads share all code and data (except local stacks)
    - Processes do not
  - Threads are somewhat less expensive than processes
    - Process control (creating and reaping) twice as expensive as thread control
    - Linux numbers:
      - ✓ ~20K cycles to create and reap a process
      - ✓ ~10K cycles (or less) to create and reap a thread

Thread Programming

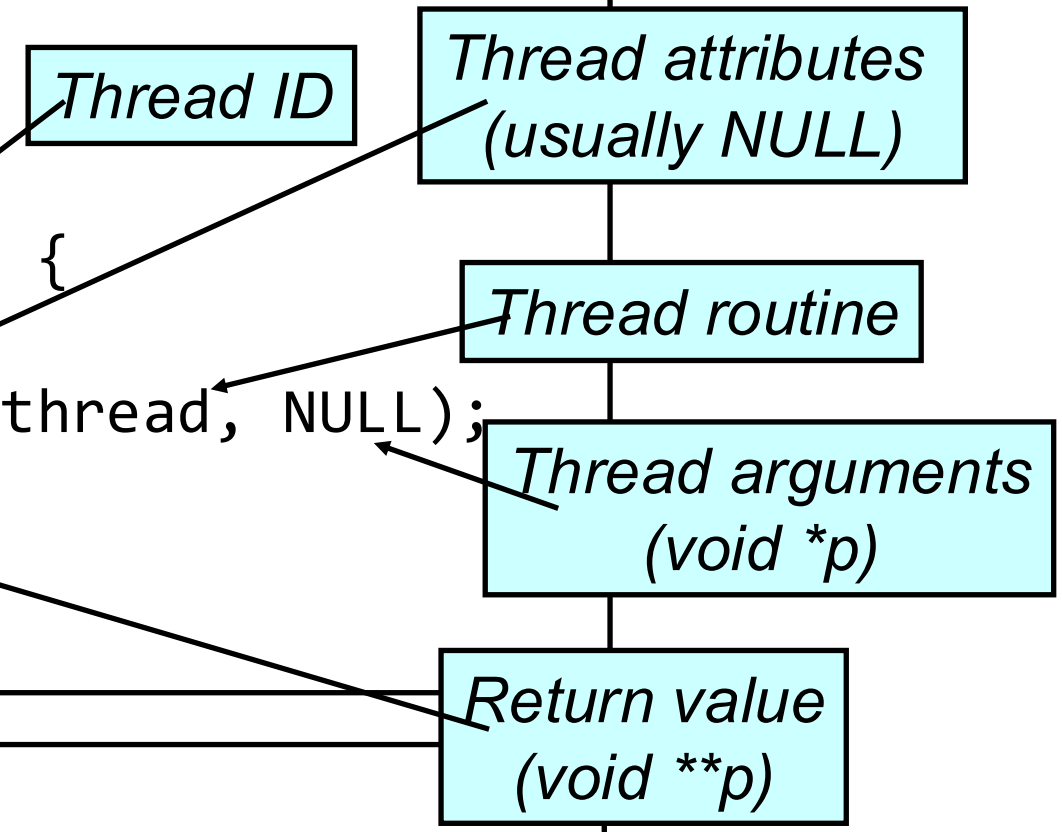
Posix Threads (pthreads) Interface

- *pthreads*: Standard interface for ~60 functions that manipulate threads from C programs
  - **Creating and reaping threads**
    - `pthread_create()`
    - `pthread_join()`
  - **Determining your thread ID**
    - `pthread_self()`
  - **Terminating threads**
    - `pthread_cancel()`
    - `exipthread_exit()`
    - `exit()` [terminates all threads]
    - `return` [terminates current thread]
  - **Synchronizing access to shared variables**
    - `pthread_mutex_init`
    - `pthread_mutex_[un]lock`

The pthreads "hello, world" Program

```
#include <pthread.h>
void *thread(void *vargp);

int main(int argc, char** argv) {
    pthread_t tid;
    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, NULL);
    return 0;
}
```



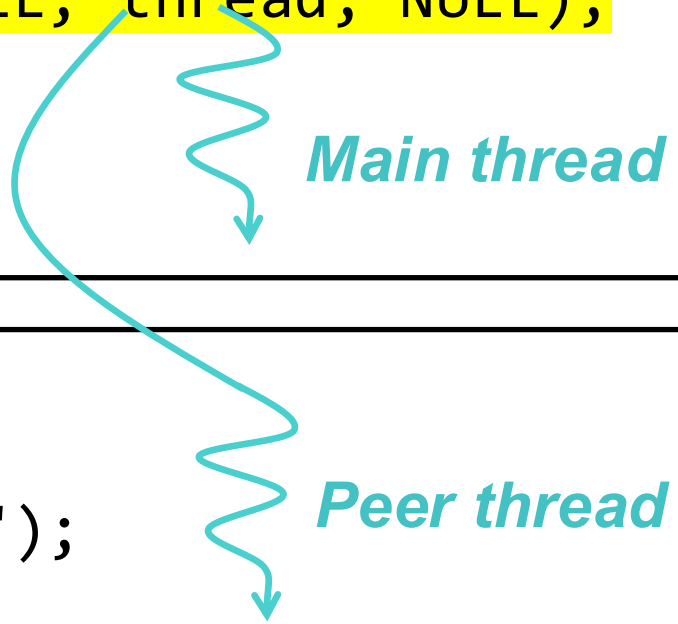

```
/* thread routine */
void *thread(void *vargp) {
    printf("Hello, world!\n");
    return NULL;
}
```

The pthreads "hello, world" Program


```
#include <pthread.h>
void *thread(void *vargp);

int main(int argc, char** argv) {
    pthread_t tid;
    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, NULL);
    return 0;
}
```

```
/* thread routine */
void *thread(void *vargp) {
    printf("Hello, world!\n");
    return NULL;
}
```



The pthreads "hello, world" Program

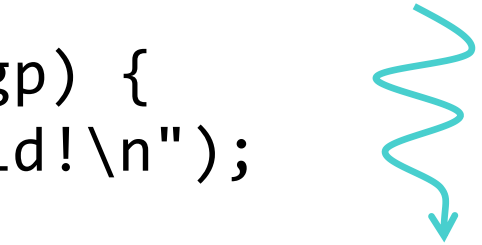

```
#include <pthread.h>
void *thread(void *vargp);

int main(int argc, char** argv) {
    pthread_t tid;
    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, NULL);
    return 0;
}
```

Main thread **waits** for peer thread to terminate

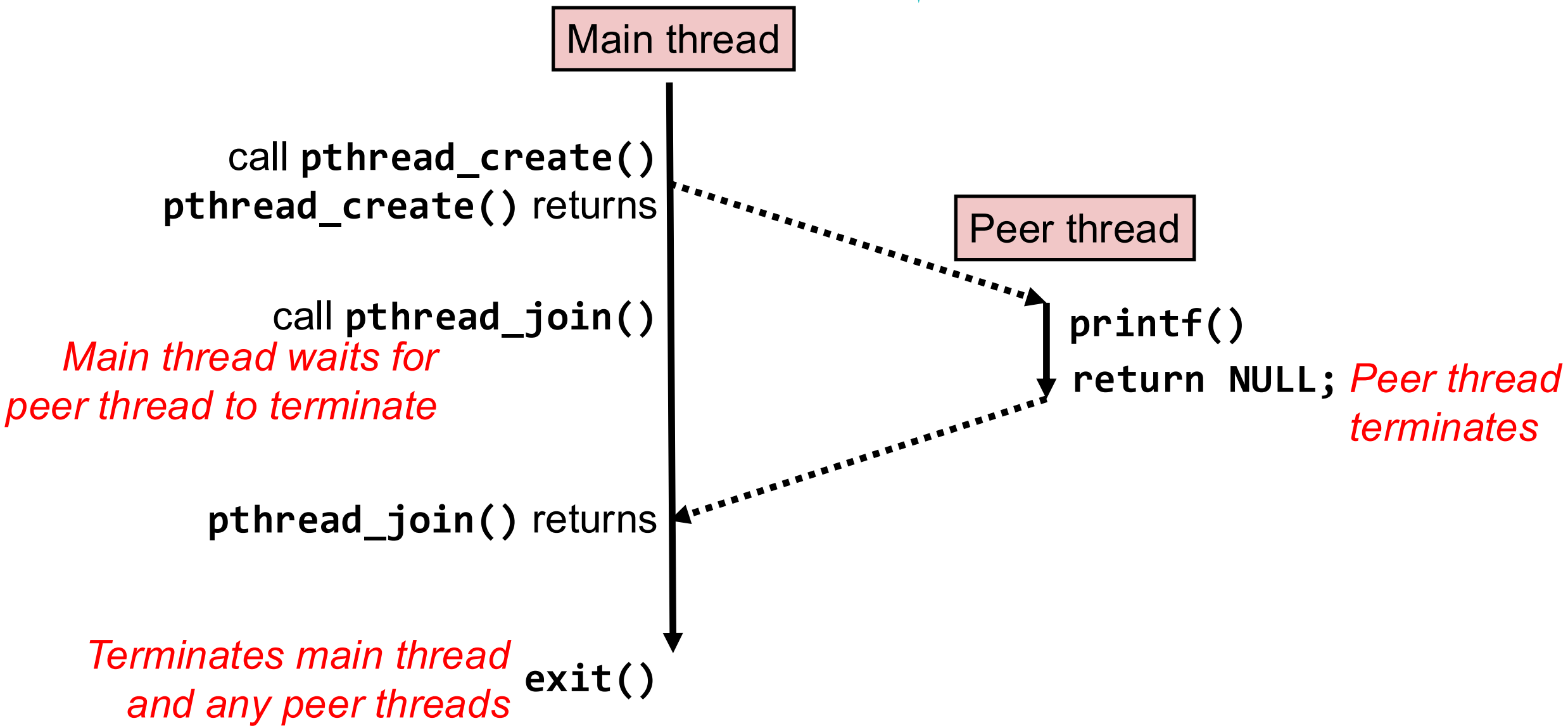
Main thread

```
/* thread routine */
void *thread(void *vargp) {
    printf("Hello, world!\n");
    return NULL;
}
```



Peer thread

Execution of Threaded "hello, world"



Question?