

CONCURRENCY: THREADS

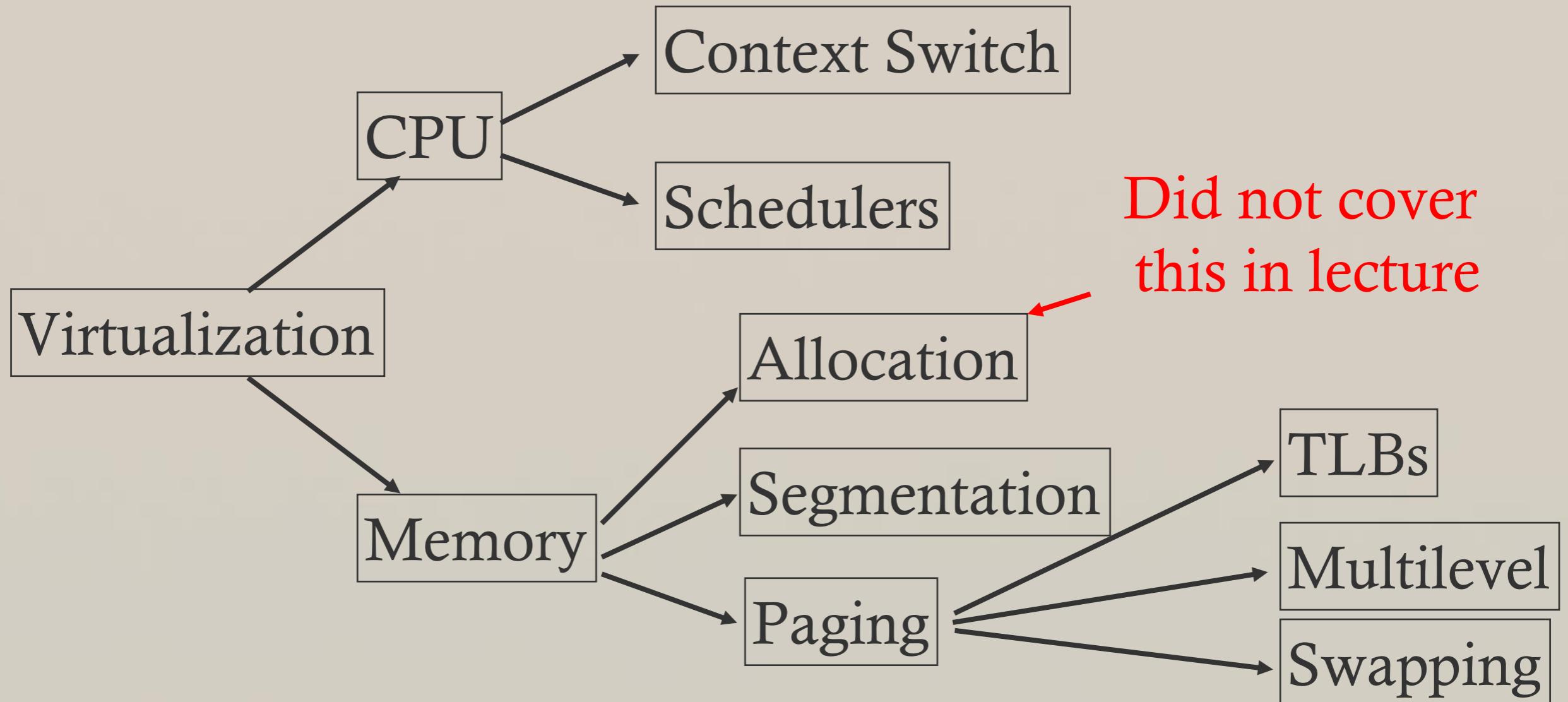
Questions answered in this lecture:

Why is concurrency useful?

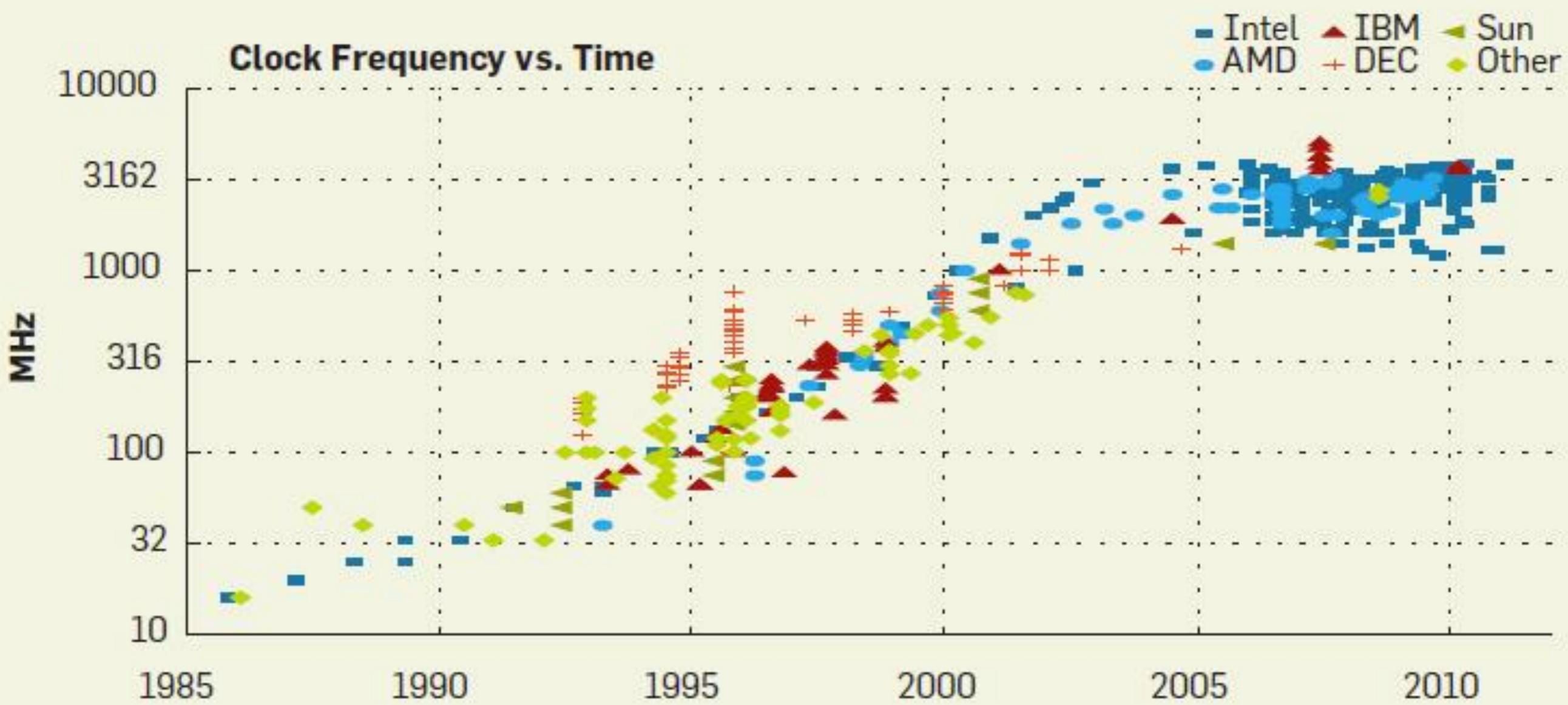
What is a thread and how does it differ from processes?

What can go wrong if scheduling of critical sections is not atomic?

FIRST PART OF COURSE



MOTIVATION FOR CONCURRENCY



MOTIVATION

CPU Trend: Same speed, but multiple cores

Goal: Write applications that fully utilize many cores

Option 1: Build apps from many communicating **processes**

- Example: Chrome (process per tab)
- Communicate via pipe() or similar

Pros?

- Don't need new abstractions; good for security

Cons?

- Cumbersome programming
- High communication overheads
- Expensive context switching (why expensive?)

CONCURRENCY: OPTION 2

New abstraction: **thread**

Threads are like processes, except:
multiple threads of same process share an address space

Divide large task across several cooperative threads

Communicate through shared address space

COMMON PROGRAMMING MODELS

Multi-threaded programs tend to be structured as:

- **Producer/consumer**

Multiple producer threads create data (or work) that is handled by one of the multiple consumer threads

- **Pipeline**

Task is divided into series of subtasks, each of which is handled in series by a different thread

- **Defer work with background thread**

One thread performs non-critical work in the background (when CPU idle)

CPU 1

running
thread 1

CPU 2

running
thread 2

RAM



What state do threads share?

CPU 1

running
thread 1

CPU 2

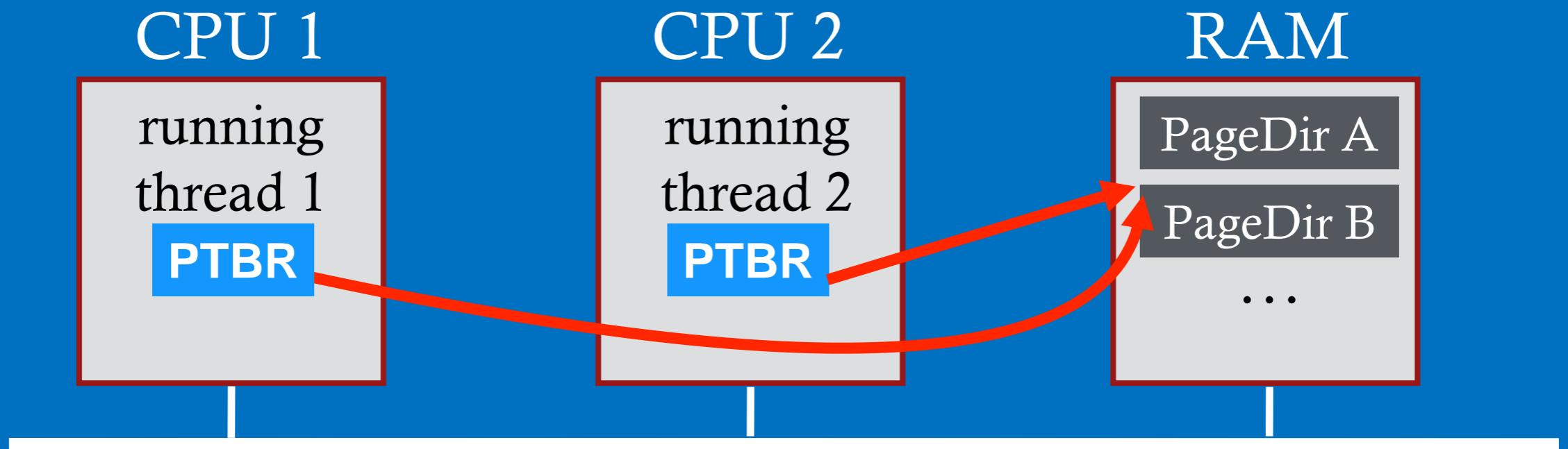
running
thread 2

RAM

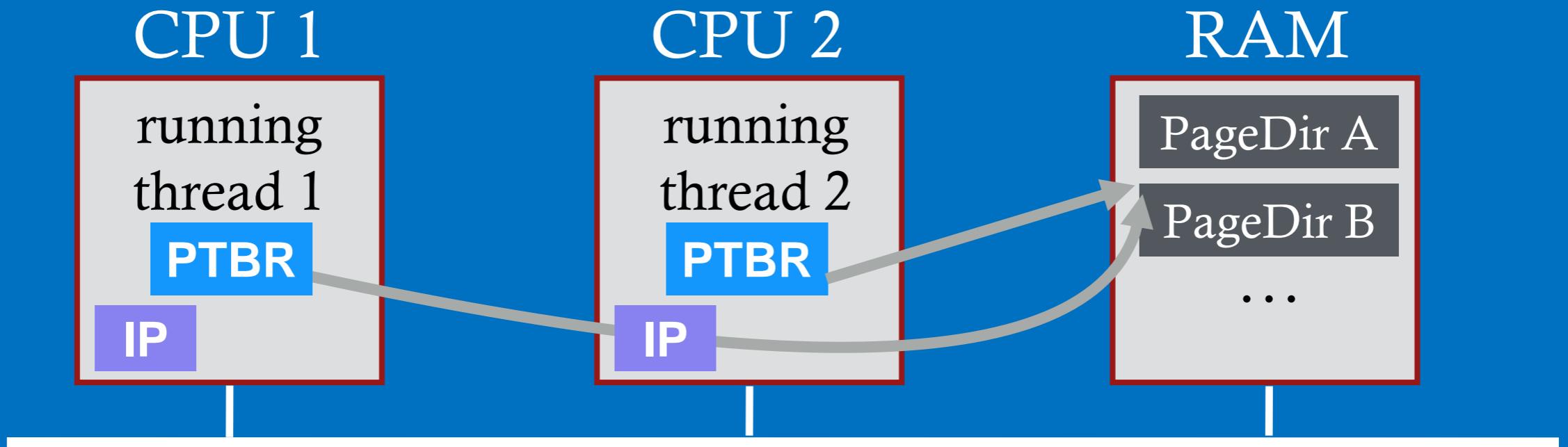
PageDir A
PageDir B

...

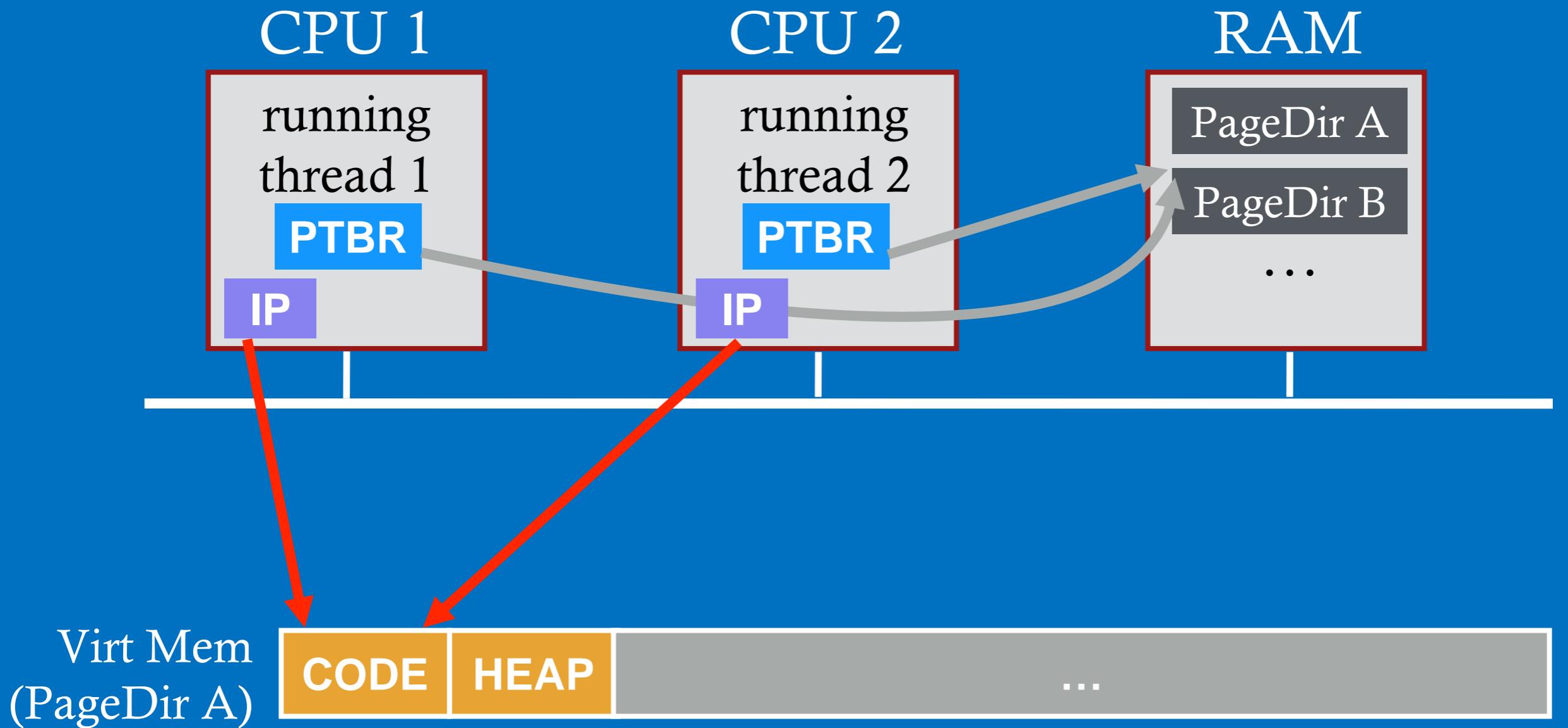
What state do threads share?



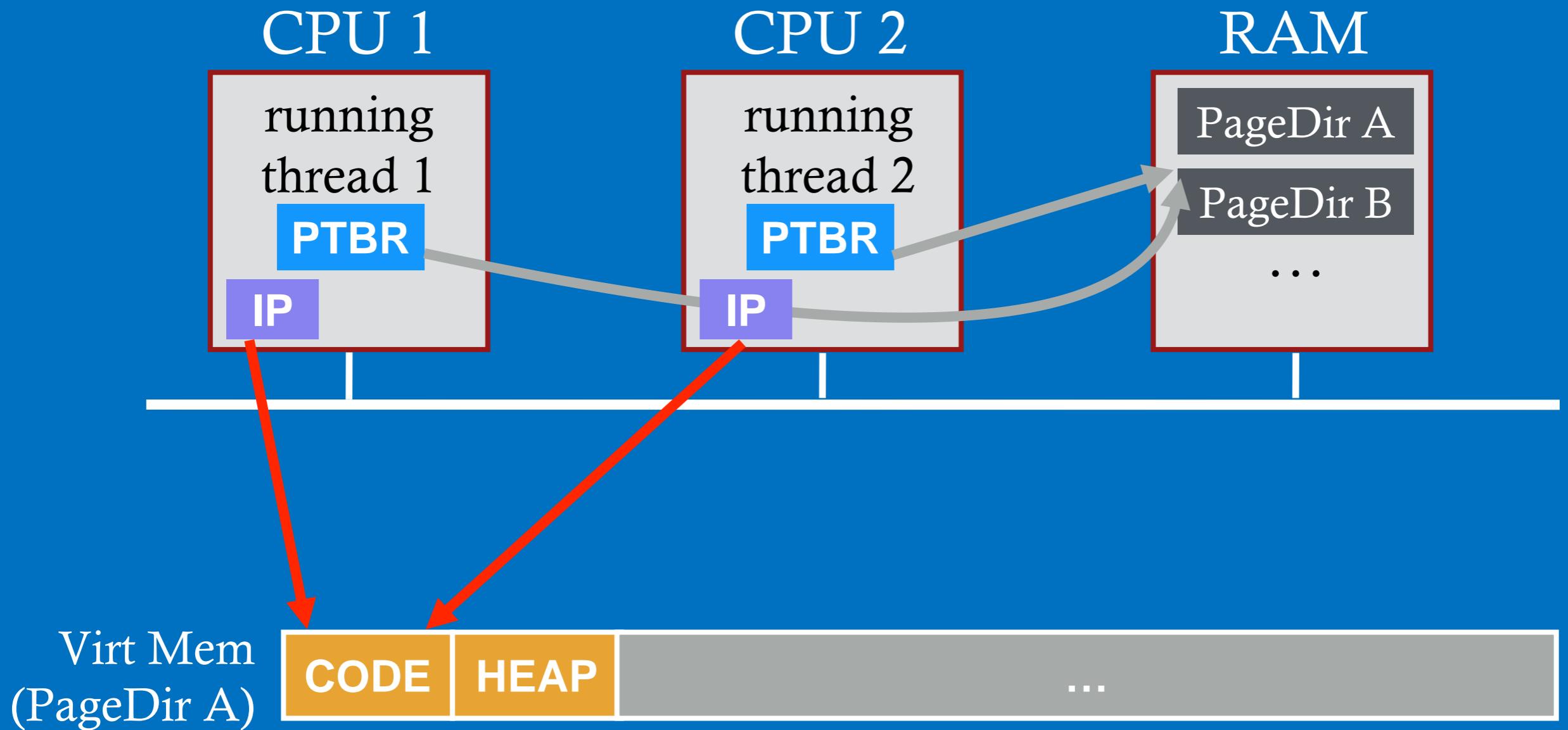
They share the page table of their host process



Do threads share Instruction Pointer?

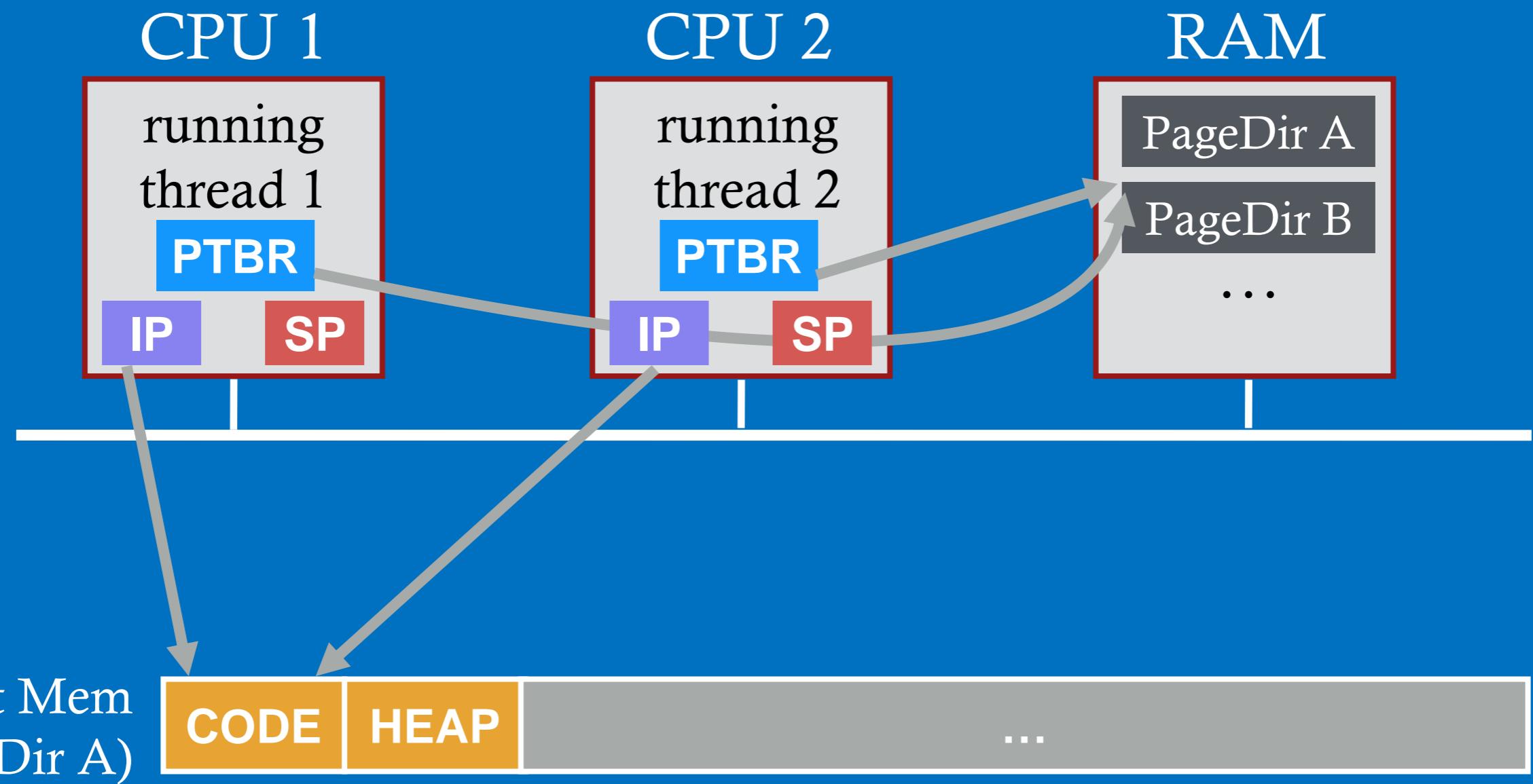


Show process address space and how a thread fits in

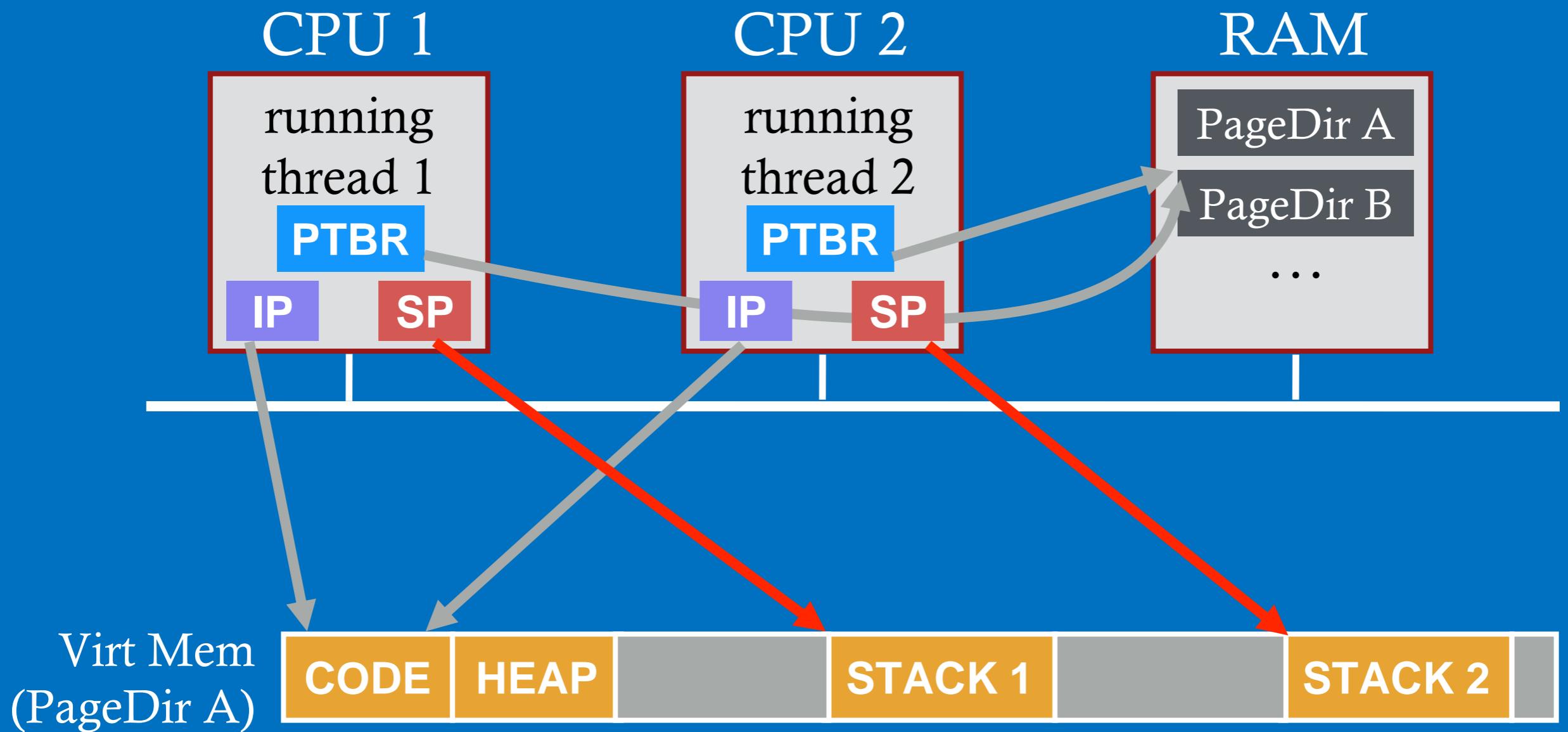


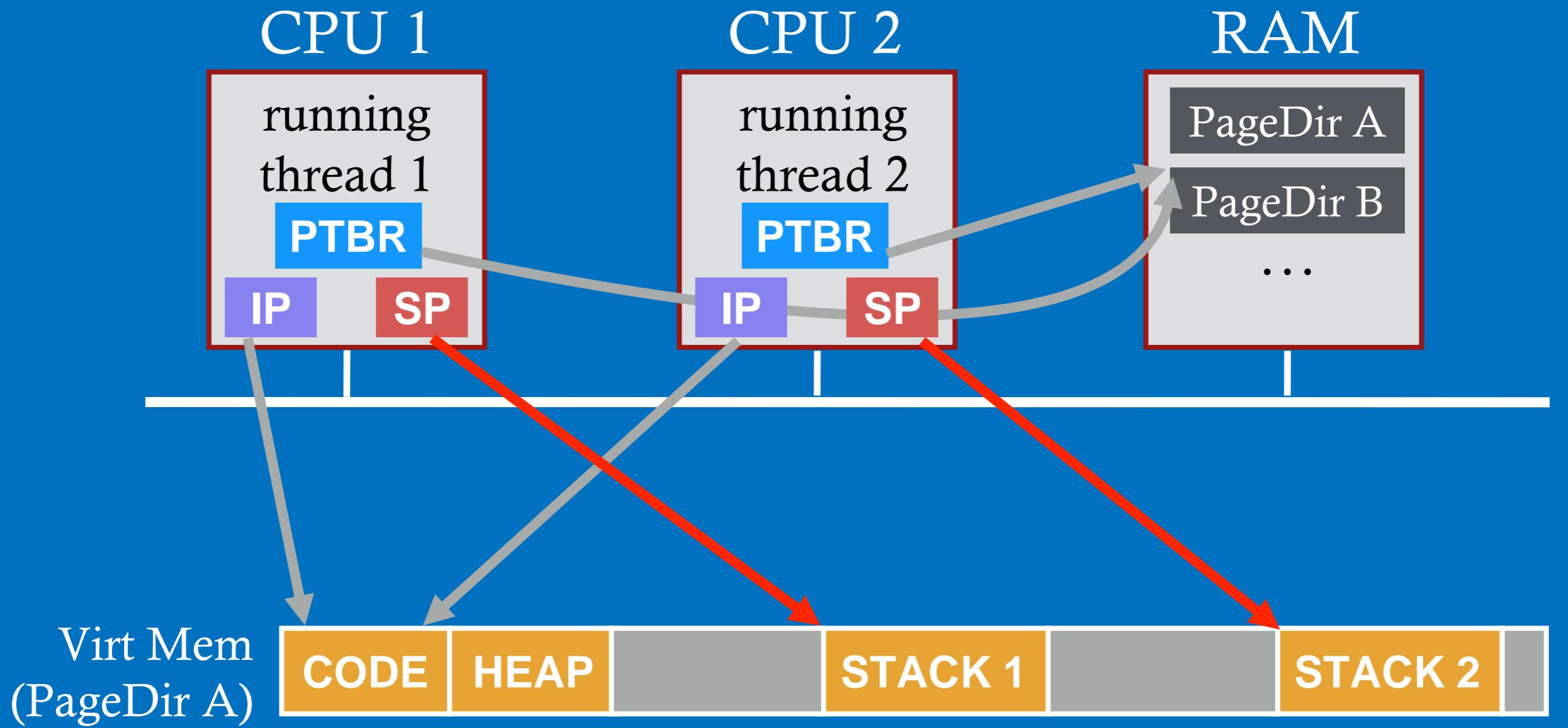
Share code, but each thread may be executing different code at the same time

→ Different Instruction Pointers



Do threads share stack pointer?





threads executing different functions need different stacks

THREAD VS. PROCESS

Multiple threads within a single process share:

- Process ID (PID)
- Address space
 - Code (instructions)
 - Most data (heap)
- Open file descriptors
- Current working directory
- User and group id

Each thread has its own

- Thread ID (TID)
- Set of registers, including Program counter and Stack pointer
- Stack for local variables and return addresses
(in same address space)

THREAD API

Variety of thread systems exist

- POSIX Pthreads

Common thread operations

- Create
- Exit
- Join (instead of wait() for processes)

OS SUPPORT: APPROACH 1

User-level threads: Many-to-one thread mapping

- Implemented by user-level runtime libraries
 - Create, schedule, synchronize threads at user-level
- OS is not aware of user-level threads
 - OS thinks each process contains only a single thread of control

Advantages

- Does not require OS support; Portable
- Can tune scheduling policy to meet application demands
- Lower overhead thread operations since no system call

Disadvantages?

- Cannot leverage multiprocessors
- Entire process blocks when one thread blocks

OS SUPPORT: APPROACH 2

Kernel-level threads: One-to-one thread mapping

- OS provides each user-level thread with a kernel thread
- Each kernel thread scheduled independently
- Thread operations (creation, scheduling, synchronization) performed by OS

Advantages

- Each kernel-level thread can run in parallel on a multiprocessor
- When one thread blocks, other threads from process can be scheduled

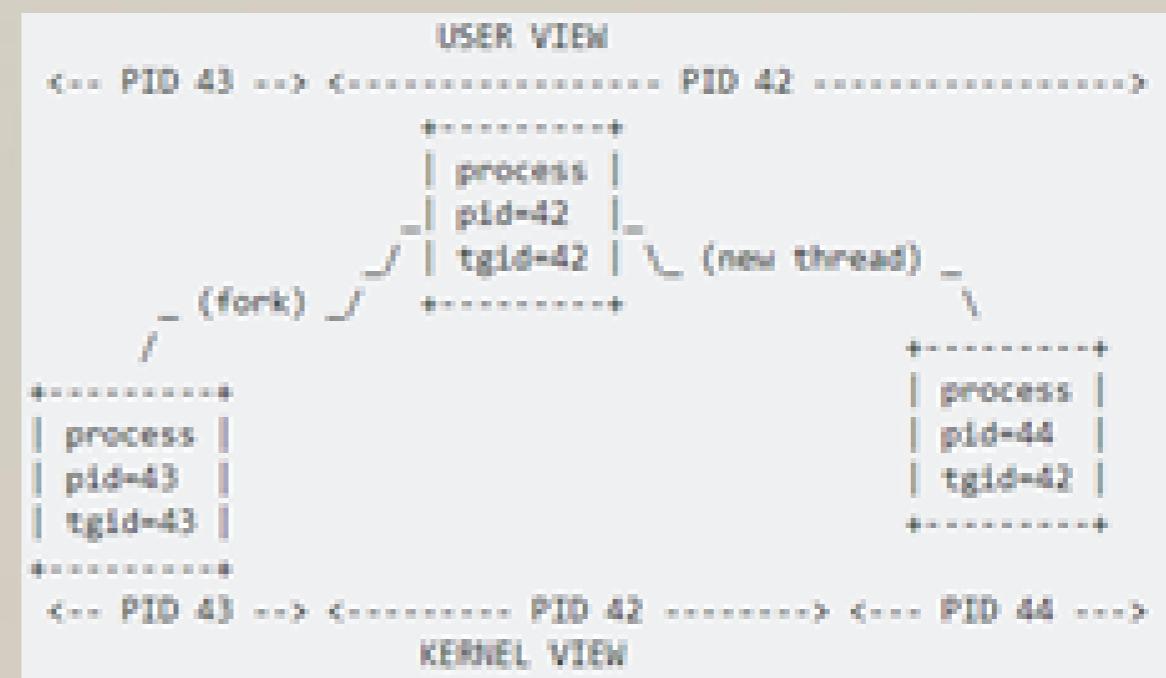
Disadvantages

- Higher overhead for thread operations
- OS must scale well with increasing number of threads

ASIDE: LINUX PROCESSES AND THREADS

Linux treats threads as processes

- Parent process pid==tgid
- Launched threads have new pid and **Parents** tgid
- So scheduler sees different pids (for scheduling)
- But same tgid means don't swap memory if swapping to same tgid

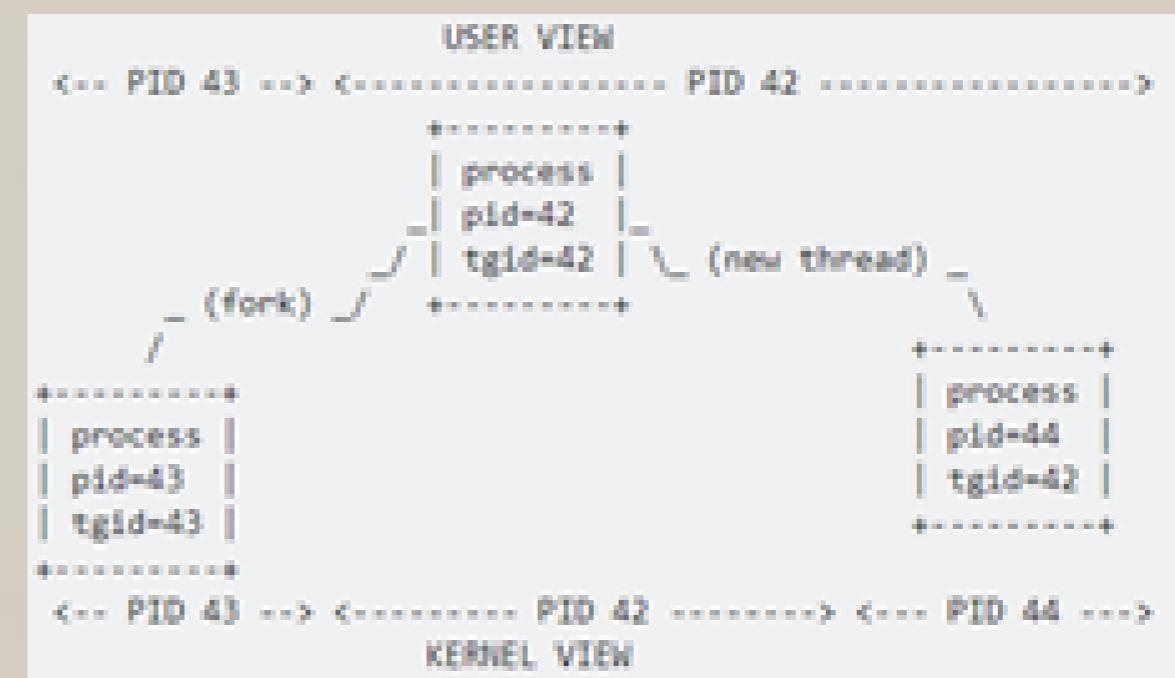


ASIDE: HTOP

Also means you can just show processes (not their internal threads) by displaying tgid only

Show how HTOP tracks processes and threads

- F2 to setup columns (PID, TID)
- Show tree view (Display options->to see parent process)



DEMO: BASIC THREADS

THREAD SCHEDULE #1

balance = balance + 1; balance at 0x9cd4

State:

0x9cd4: 100

%eax: ?

%rip = 0x195

process
control
blocks:

Thread 1

%eax: ?
%rip: 0x195

Thread 2

%eax: ?
%rip: 0x195

T1 →

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

THREAD SCHEDULE #1

State:

0x9cd4: 100

%eax: 100

%rip = 0x19a

process
control
blocks:

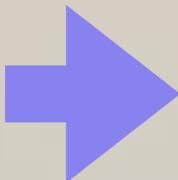
Thread 1

%eax: ?
%rip: 0x195

Thread 2

%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

%eax: ?
%rip: 0x195

Thread 2

%eax: ?
%rip: 0x195

T1 →

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

THREAD SCHEDULE #1

State:

0x9cd4: 101

%eax: 101

%rip = 0x1a2

process
control
blocks:

Thread 1

%eax: ?
%rip: 0x195

Thread 2

%eax: ?
%rip: 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

%eax: ?
%rip: 0x195

Thread 2

%eax: ?
%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

%eax: 101
%rip: 0x1a2

Thread 2

%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

%eax: 101
%rip: 0x1a2

Thread 2

%eax: ?
%rip: 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:

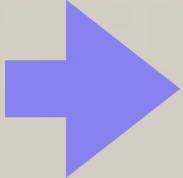
Thread 1

%eax: 101
%rip: 0x1a2

Thread 2

%eax: ?
%rip: 0x195

T2



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

THREAD SCHEDULE #1

State:

0x9cd4: 102

%eax: 102

%rip = 0x1a2

process
control
blocks:

Thread 1

%eax: 101
%rip: 0x1a2

Thread 2

%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

%eax: 101
%rip: 0x1a2

Thread 2

%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

thread
control
blocks:

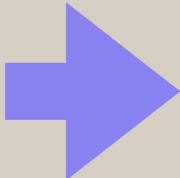
Thread 1

%eax: ?
%rip: 0x195

Thread 2

%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

thread
control
blocks:

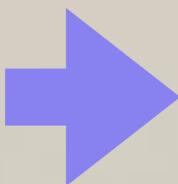
Thread 1

%eax: ?
%rip: 0x195

Thread 2

%eax: ?
%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

thread
control
blocks:

Thread 1

%eax: ?
%rip: 0x195

Thread 2

%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

thread
control
blocks:

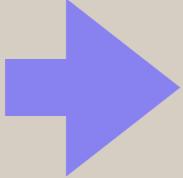
Thread 1

%eax: 101
%rip: 0x19d

Thread 2

%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

thread
control
blocks:

Thread 1

%eax: 101
%rip: 0x19d

Thread 2

%eax: ?
%rip: 0x195

T2 →

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

THREAD SCHEDULE #2

State:

0x9cd4: 100

%eax: 101

%rip = 0x19d

thread
control
blocks:

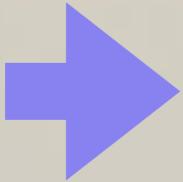
Thread 1

%eax: 101
%rip: 0x19d

Thread 2

%eax: ?
%rip: 0x195

T2



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

THREAD SCHEDULE #2

State:

0x9cd4: 101

%eax: 101

%rip = 0x1a2

thread
control
blocks:

Thread 1

%eax: 101
%rip: 0x19d

Thread 2

%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

thread
control
blocks:

Thread 1

%eax: 101
%rip: 0x19d

Thread 2

%eax: ?
%rip: 0x195

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

T2 →

Thread Context Switch

THREAD SCHEDULE #2

State:

0x9cd4: 101

%eax: 101

%rip = 0x19d

thread
control
blocks:

Thread 1

%eax: 101
%rip: 0x19d

Thread 2

%eax: 101
%rip: 0x1a2

T1 →

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

Thread Context Switch

THREAD SCHEDULE #2

State:

0x9cd4: 101

%eax: 101

%rip = 0x19d

thread
control
blocks:

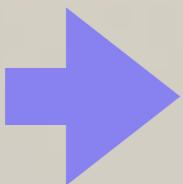
Thread 1

%eax: 101
%rip: 0x19d

Thread 2

%eax: 101
%rip: 0x1a2

T1



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

THREAD SCHEDULE #2

State:

0x9cd4: 101

%eax: 101

%rip = 0x1a2

thread
control
blocks:

Thread 1

%eax: 101
%rip: 0x1a2

Thread 2

%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

thread
control
blocks:

Thread 1

%eax: 101
%rip: 0x1a2

Thread 2

%eax: 101
%rip: 0x1a2

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

T1 →

WRONG Result! Final value of balance is 101

TIMELINE VIEW

Thread 1

```
mov 0x123, %eax
```

```
add %0x1, %eax
```

```
mov %eax, 0x123
```

Thread 2

```
mov 0x123, %eax
```

```
add %0x2, %eax
```

```
mov %eax, 0x123
```

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

TIMELINE VIEW

Thread 1

mov 0x123, %eax

add %0x1, %eax

mov %eax, 0x123

How much is added?

Thread 2

mov 0x123, %eax

add %0x2, %eax

mov %eax, 0x123

2: incorrect!

TIMELINE VIEW

Thread 1

mov 0x123, %eax

add %0x1, %eax

mov %eax, 0x123

Thread 2

mov 0x123, %eax

add %0x2, %eax

mov %eax, 0x123

How much is added?

1: incorrect!

TIMELINE VIEW

Thread 1

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

Thread 2

```
mov 0x123, %eax  
add %0x2, %eax  
mov %eax, 0x123
```

How much is added?

3: correct!

TIMELINE VIEW

Thread 1

mov 0x123, %eax

add %0x1, %eax

mov %eax, 0x123

Thread 2

mov 0x123, %eax

add %0x2, %eax

mov %eax, 0x123

How much is added?

2: incorrect!

NON-DETERMINISM

Concurrency leads to non-deterministic results

- Not 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

More general:

Need mutual exclusion for critical sections

- if process A is in critical section C, process B can't (okay if other processes do unrelated work)

SYNCHRONIZATION

Build higher-level synchronization primitives in OS

- Operations that ensure correct ordering of instructions across threads

Motivation: Build them once and get them right

Monitors Locks Semaphores
Condition Variables

Loads Stores Test&Set
Disable Interrupts

CONCLUSIONS

Concurrency is needed to obtain high performance by utilizing multiple cores

Threads are multiple execution streams within a single process or address space (share PID and address space, own registers and stack)

Context switches within a critical section can lead to non-deterministic bugs (race conditions)

Use locks to provide mutual exclusion