COMP 8551 Advanced Games Programming Techniqu

Borna Noureddin, Ph.D.

British Columbia Institute of Technology

Realtime Issues and Multithreading I

Overview

- Overview of multithreading
- Basic definiti
- Multithreading challenge
- Race conditions
- Mutexes

What is multithreading?

- Technique allowing application to do multiple tasks “simultaneously”
- Stream of i   a process
- Each thread   nter, set of registers, stack memory
- Virtual address space common to all threads within a process
  - Data on heap can be accessed by all threads

What is multithreading?

- Not new, but only in past decade useful on PCs, especially with multi-core processors (before tha ng systems; concurrent

- Why now?
  - emergence of SMPs in particular

What is multithreading?

- What is an SMP?
  - Multiple CPUs in a single box sharing all the resou ry and I/O
- Is an SMP m an two uniprocessor boxes?
  - Yes (roughly 20% more for a dual processor SMP)
  - Modest speedup for application on dual-processor SMP will make it worthwhile

Applications

- Multimedia
- GUIs
- Games
- Process-intensive (e.g., calculation or visualization)
- High-end rendering

Multi-threading vs. -processing

- Threads: shared memory; lightweight

- Processes: more overhead

- Usually O/S assigns threads to different processors

Multi-threading vs. -processing

- Application with multiple threads running wit process

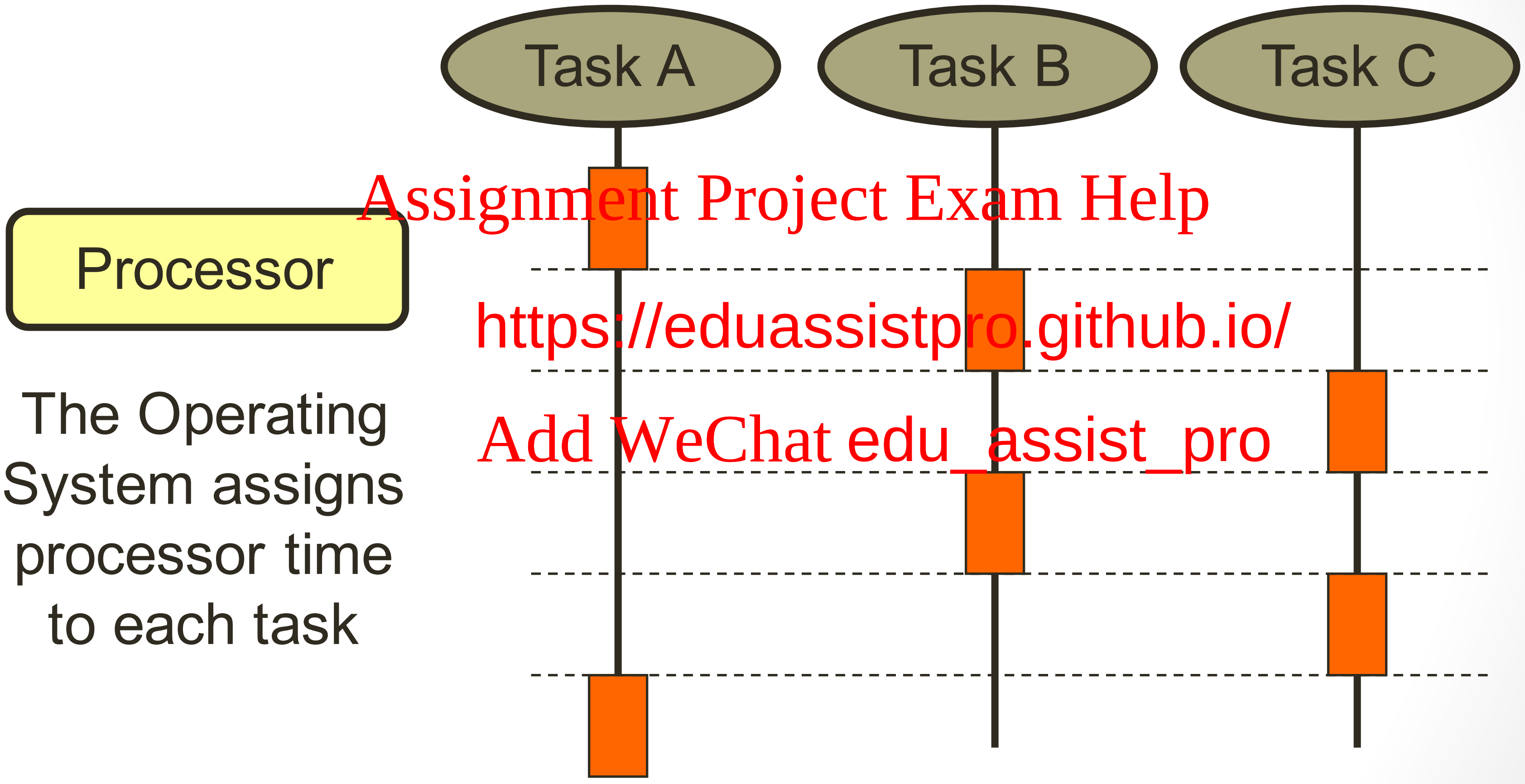
- Application organized across OS-level s

Multi-threading vs. -processing

- More “light weight” form of concurrenc
  - context pe
  - lifetime, context switching and synchronization costs lower

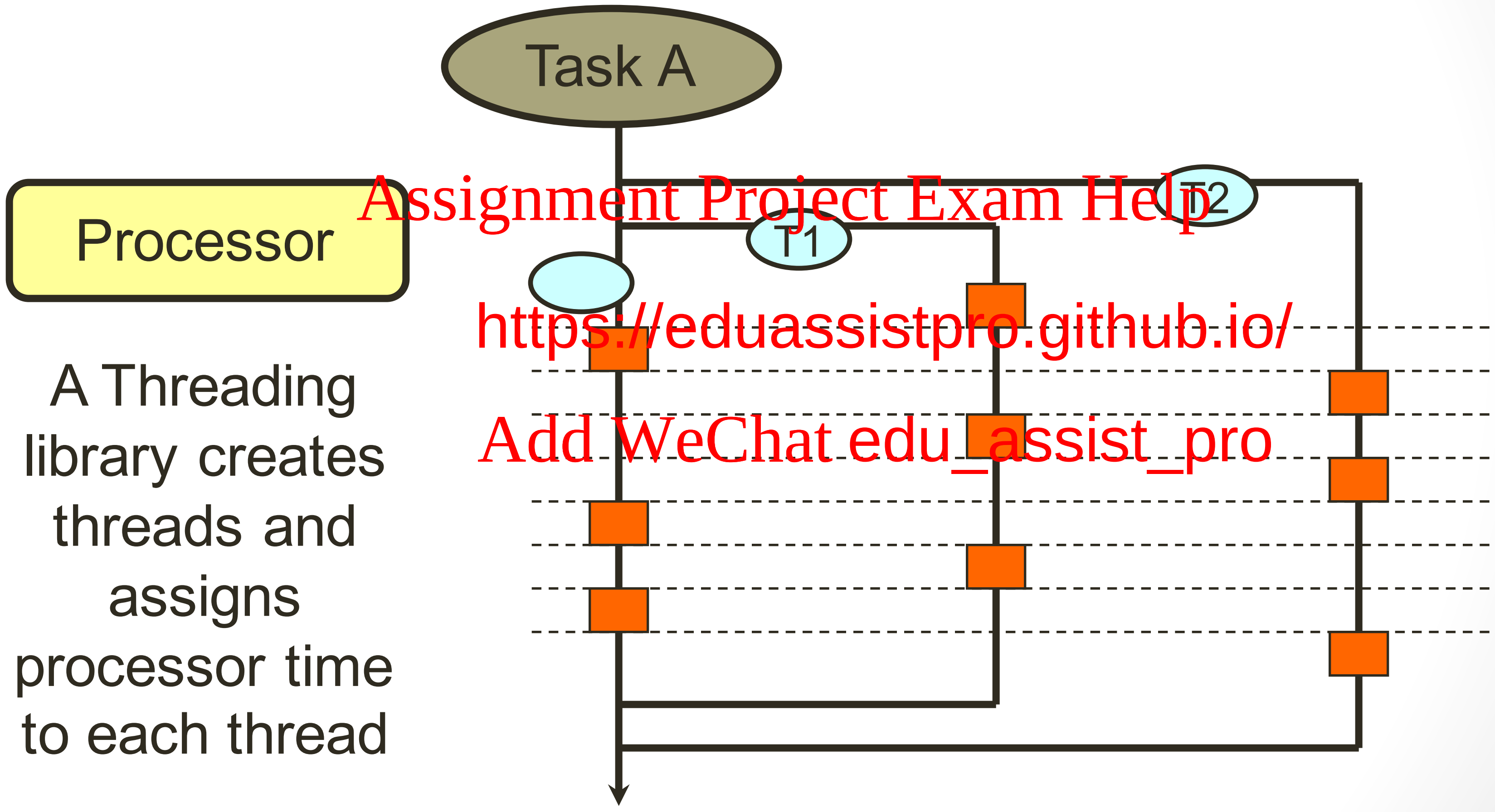
- Processes are insulated from each
  OS
  n one cannot
  wn another
  - individual processes may run as different users and have different permissions

The Multi-Tasking Concept



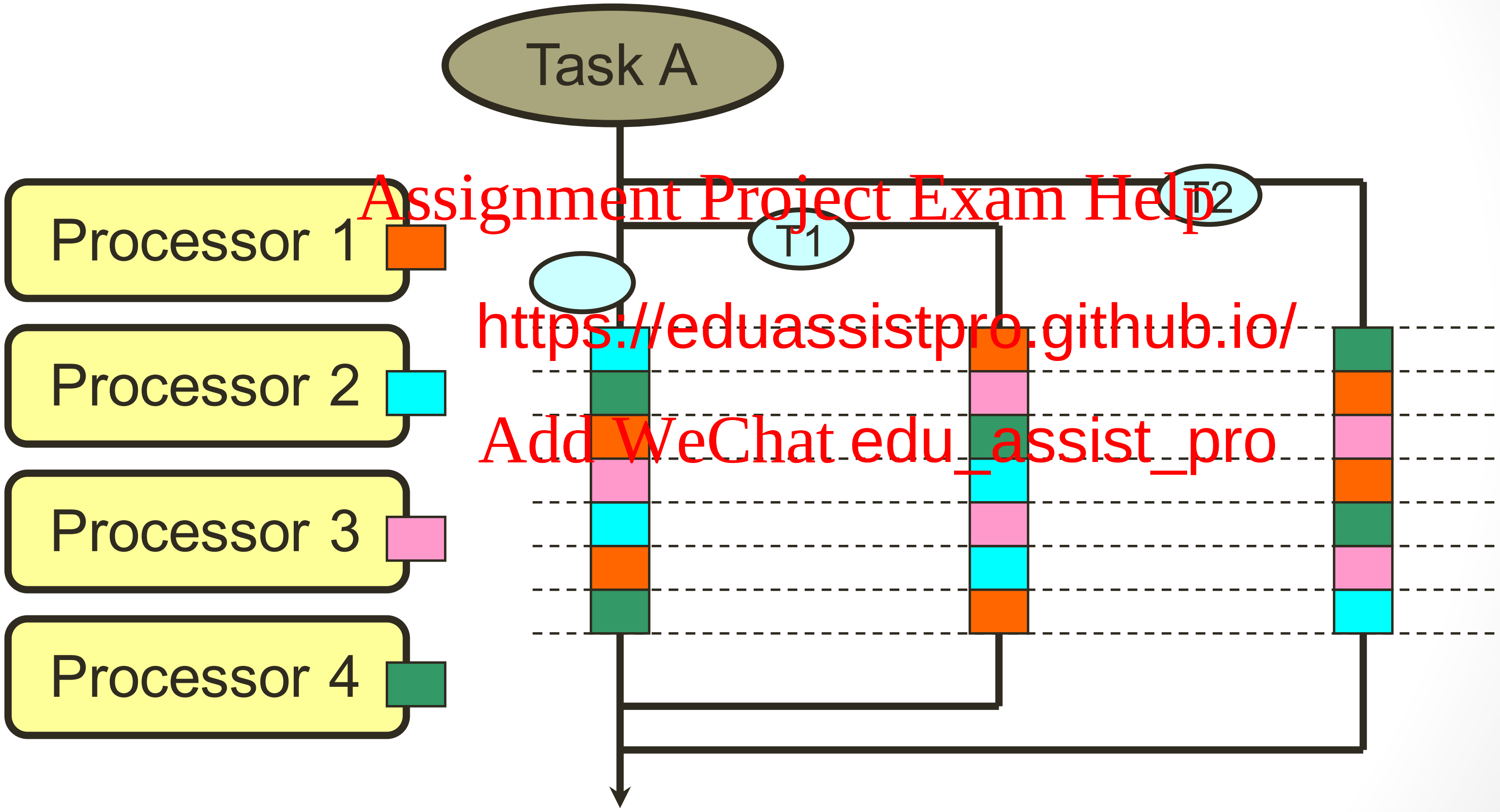
The Operating System assigns processor time to each task

The Multi-Threading Concept



A Threading library creates threads and assigns processor time to each thread

Multi-Threading in Multi-Processors



Definitions

Physical CPU: Actual CPU/processor on the motherboard. Single core: same number of physical

Logical CPU: A separate line. HyperThreaded: two logical CPUs per core, multi-core processor: one logical CPU per core per processor.

Definitions

Atomic Operation: Operation in code to be executed by one thread at a time, typically t egrity:

```
intThreadI
while (int                                mumIterations &&
       intThreadIteration
   // Do some work
   intSharedVariable++;
   intThreadIterations++;
}
```

will not work correctly if other threads try updating `intSharedVariable`: this block of code must be atomic operation.

Definitions

Block: Thread (process) is in such a state that all other threads must wait until it is finished to c E.g., any thread tryin `dVariable` will block until atomic o is completed.

Lock: System for restricting access to resource to other threads (other threads will block until lock is released).

Main challenge

Shared resources

- Locking data

- Size of atomic operations
- Testing/debugging is much harder

Race conditions

- Behaviour of code depends on interleaving of multiple threads –fundamental problem with multi-t [illegible] ing
- Single-thre [illegible] think about lines of code right in fron [illegible] can assume data will not “magically” change between statements
- Multi-threaded code: non-local data can change unexpectedly due to actions of another thread

Race conditions

- Can result in high-level logical fault in your program
- May even pi -level abstraction:
  - cannot even assume t C++ statements execute atomically (may compile to multiple assembly instructions)
  - cannot guarantee outcome of `foo += 1;` if `foo` is non-local and may be accessed from multiple threads

Race conditions

```
int sharedCounter = 50;

void* workerThread(void*)
{
    while(sh
    {
        doSomeWork();
        --sharedCounter;
    }
}
```

Race conditions

- Start a number of threads, all executing workerThread()
- Just one thr will be executed th f times (whatever sharedCoun out at).
- Multiple threads: doSomeWork() will most likely be executed too many times.
  - we do not test and update sharedCounter as an atomic operation!

Race conditions

- Solution: use a mutex to synchronize threads with respect to the test and update

- That is, we need to defi ical section” in which we both test and update the sharedCounter.

Mutexes

- A locking primitive used to ensure that only one thread at a time has access to a resource
- An OS-level synchronizat tive that can be used to ensure a section of code can only be executed by one thread at a time

Mutexes

- Two states: locked and unlocked
- Locked: any further attempt to lock it will block (callin spended)
- Unlocked: if aiting, one of these will be resume lock the mutex
  - mutex may only be unlocked by the thread that locked it

Mutexes

- If we have resource we need to share between threads:
  - Associate
  - Use mutex ce access
  - Ensure our code locks fore using resource, and unlocks it after it is finished
- Will prevent race conditions related to multiple threads simultaneously accessing that resource

Mutexes

Additional Reading

http://randu.org/tutorials/threads/

http://www.codeproject.com/Articles/14746/Multithreading-Tutorial

http://www.compu eadingTut1.htm

https://katyscode.wordpress.com/2 troduction-to-multi-threaded-multi-core-and-parallel-programming-concepts/

https://scalibq.wordpress.com/2012/06/01/multi-core-and-multi-threading/

Review

- Overview of multithreading
- Basic definiti
- Multithreading challenge
- Race conditions
- Mutexes