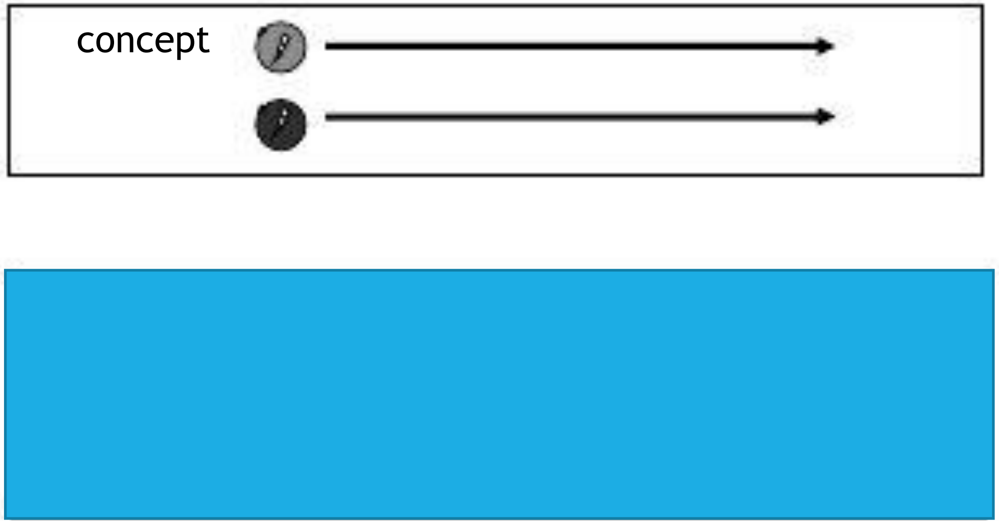
CSci 4061
Introduction to Operating Systems

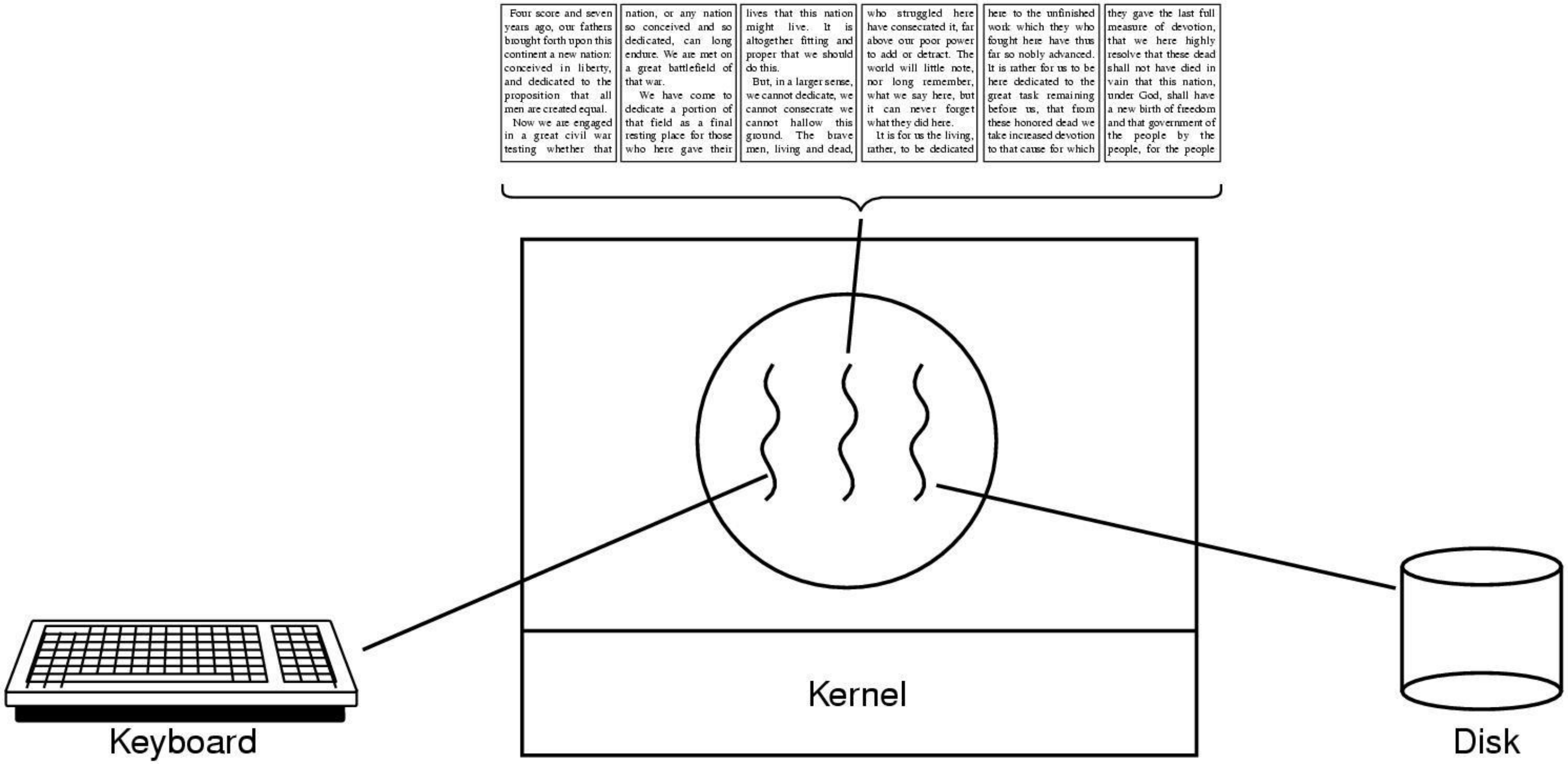
Module 5: Threads
(Implementation/Models)
Chapter 12

Two Threads Sharing a CPU



What may cause a switch?

Threads Example: editor



Four score and seven years ago, our fathers brought forth upon this continent a new nation: conceived in liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield of that war.

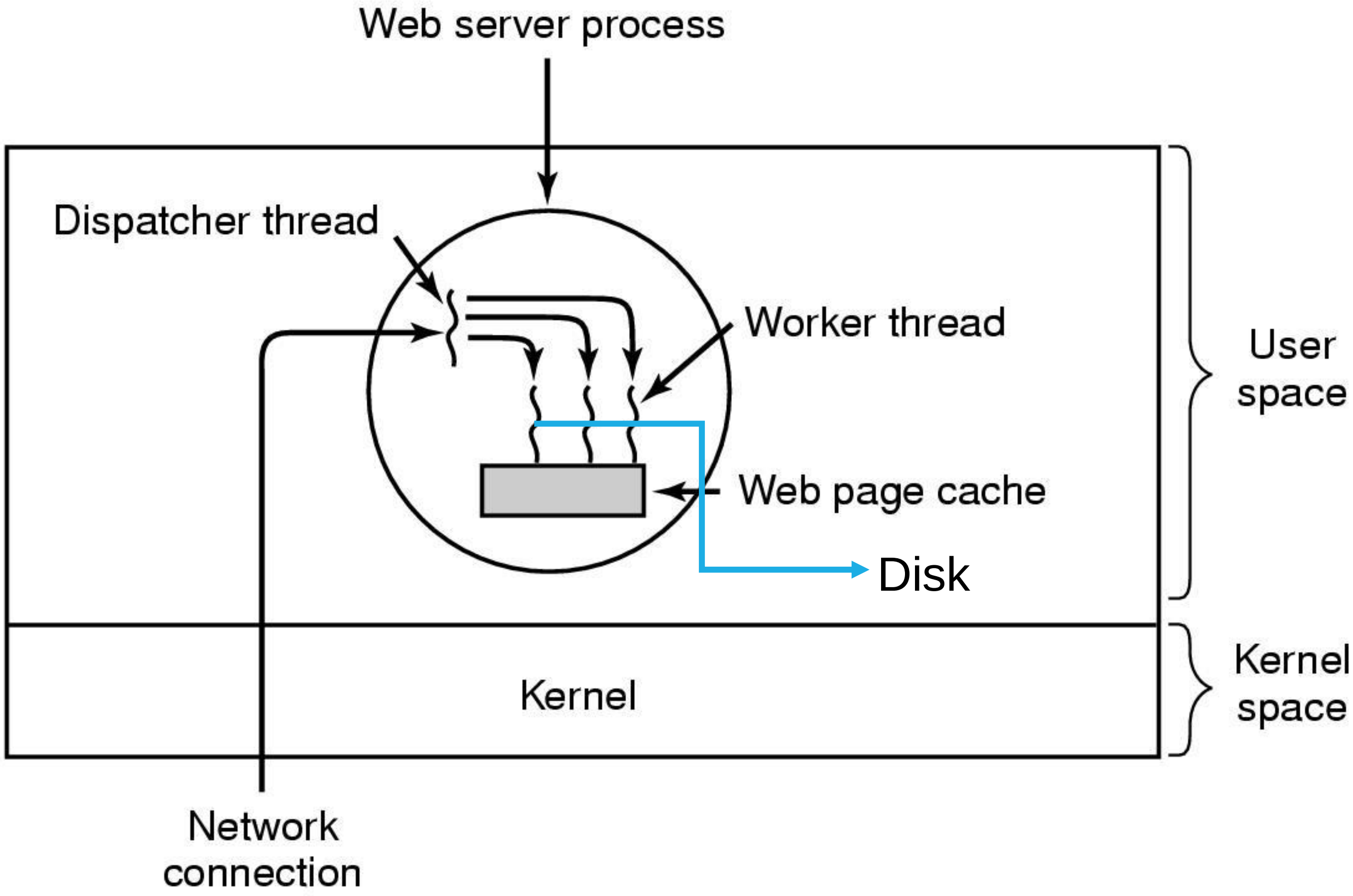
We have come to dedicate a portion of that field as a final resting place for those who here gave their lives that this nation might live. It is altogether fitting and proper that we should do this.

But, in a larger sense, we cannot dedicate, we cannot consecrate we cannot hallow this ground. The brave men, living and dead, who struggled here have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember, what we say here, but it can never forget what they did here.

It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us, that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion, that we here highly resolve that these dead shall not have died in vain that this nation, under God, shall have a new birth of freedom and that government of the people by the people, for the people

When one blocks, another can run …

Thread example: Web server…

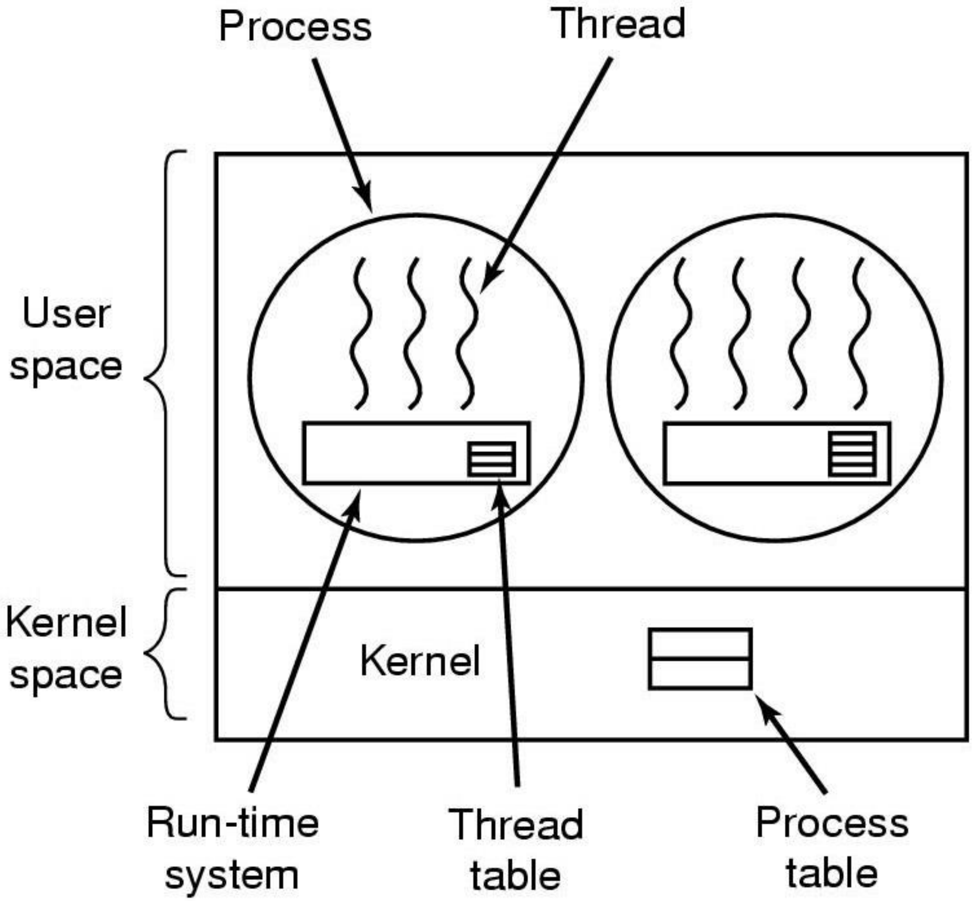


When one blocks, another can run …

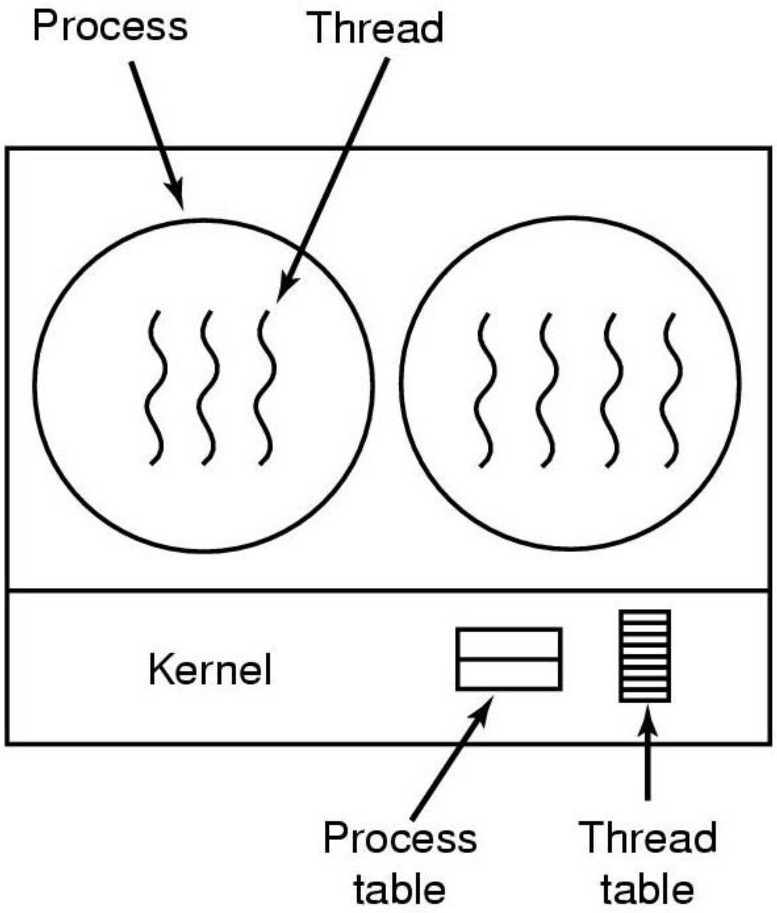
Thread Overview

- Thread advantages
  - modularity, **concurrency**
  - sharing, cheap
- Sharing is a double edged sword
  - race conditions, failure
- **Implementation models**
  - **user, kernel, hybrid**
- **Programming models**
  - **dispatcher and team**

Implementing Threads in User Space



Implementing Threads in the Kernel



A threads package managed by the kernel

User vs. Kernel Threads

- User thread advantages
  - no thread system calls! **--cheaper**
  - more scalable
  - more portable
  - custom control and scheduling
  - blocking is a big problem!

```
dispatcher (…) {
while (TRUE){
 get_next_request (&req);
 handoff_work (&req, &buf);
}
```

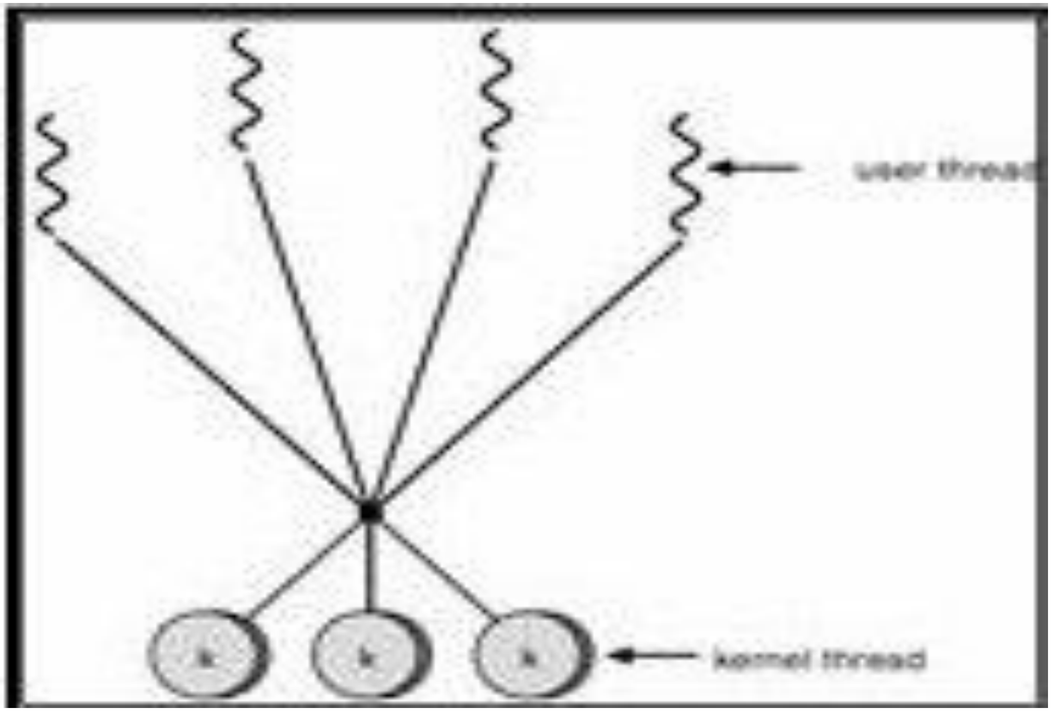
```
worker (…) {
wait_for_work (&buf, &req);
look_for_page_in_cache (&req, &answer);
if (page_not_in_cache (&answer) {
  read_page_from_disk (&req, &answer);
  put_page_in_cache (&req, &answer);
}
return (&answer);
```

Kernel Threads

- Advantages
  - thread can block: OS can pick another from same process
  - can exploit multiprocessors
- Hyper-threading
  - hardware support for threads!

Hybrid Thread Models

- One-To-One (Linux, WinXY, Posix)
  - each user-level thread maps to a kernel thread
  - if on a multicore system

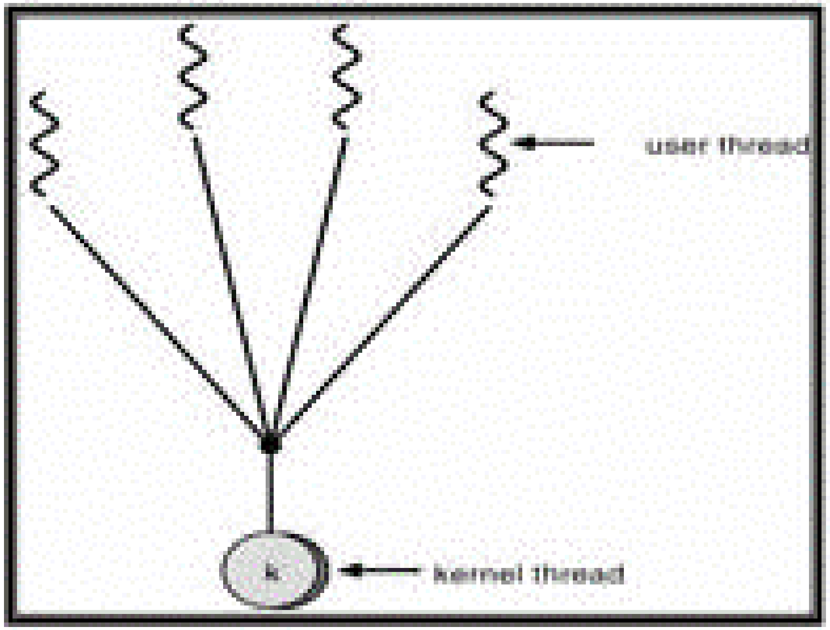


User thread

Kernel thread

Hybrid Thread Models

- Many-to-One (Java)
  - Many user-level threads mapped to a single kernel thread



How do I program them?

General Thread Operations

- `Create/Fork`
  - Allocate memory for stack, perform bookkeeping
  - Parent thread creates child threads
  - Associates function to execute
  - Returns an id

- `Destroy/Cancel`
  - Release memory (or recycle), perform bookkeeping

- `Suspend` (e.g. Sleep) ->blocked
  - `Resume`->unblock, `Yield`->deschedule

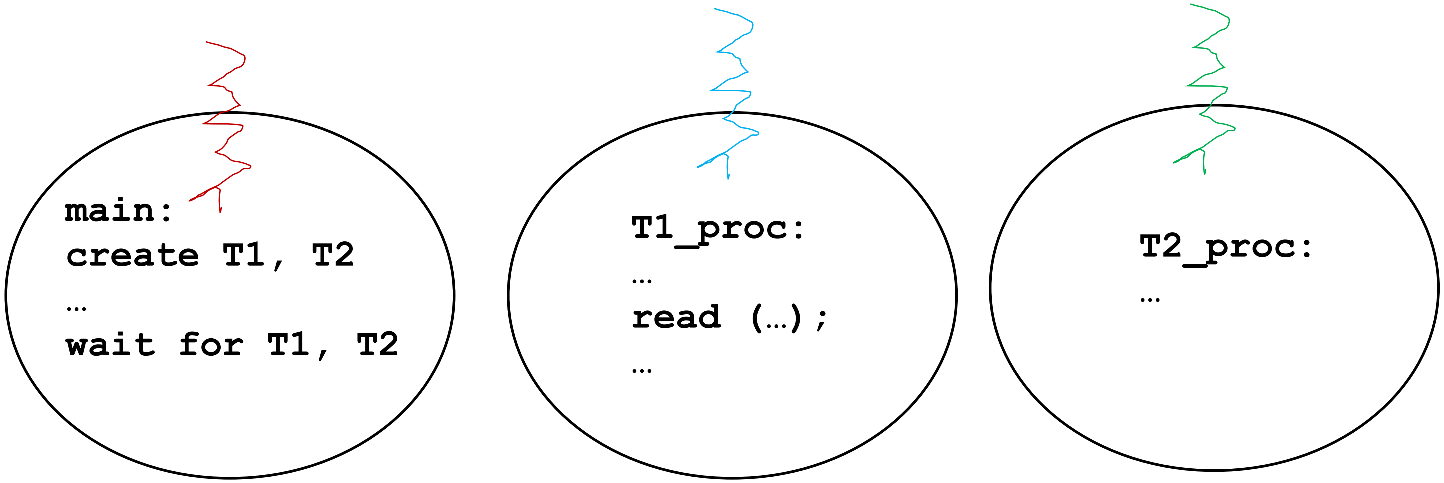
- Wait/`Join`
  - Wait for something, e.g. child finishing

Inside Threads

- A thread contains
  - pc
  - sp
  - registers
  - child threads
  - state

- What about open files?

Threads in Action



```
main:
create T1, T2
…
wait for T1, T2
```

```
T1_proc:
…
read (…);
…
```

```
T2_proc:
…
```

- `main` starts executing creates T1 and T2
- `main` blocks at `wait`
- Switches to T1 or T2, say T1
  - blocks at `read`
- Switches to T2

Thread Models

- **Dispatcher-worker (master-slave)**
  - a master process/thread receives request for work
  - generates/dispatches a thread to service work request
    - e.g. threaded server

- Two options:
  - master can create a thread on “as needed basis” *pop-up*
  - master can keep a thread pool
    - may reduce perceived latency of creating threads to service request
    - issues?

Thread pop up vs. pool

- Pop up
    - ^ latency of serving request
    - can size the # of threads to workload
    - what to do when workload is very high?

- Thread pool
    - how to size the pool?
    - too many threads may waste resources increase context switching
    - too few threads increase latency
    - could make the pool dynamic!

Thread Models (cont’d)

- **Team**
  - a collection of peer threads working on some part of a problem together
  - identical threads (parallelism):
    - parallel program running on shared-memory multiprocessor, multi-core
    - *n* threads are created and each are given a share of the problem

e.g. scale element of a matrix

  - different threads (concurrency):
    - editor example

Thought Question

- On a uniprocessor
    - Threaded matrix multiply program
    - NxN matrix (N is large) and sitting in memory
    - Create 4 threads each responsible for ¼ of the matrix multiply operations
    - Time the 4-threaded version and compare with a single threaded version
        - The 4-threaded version does worse---WHY?

Remember

- System may time-slice your thread

- You should assume that a thread could be switched at any time … your program should still work
  - A program that fails 1 out of 10^100 runs is buggy

- This will make your code much more portable

Thread Safety

- Two or more threads call code that has the potential for race conditions.
- Deal with it.
- Linux man pages will tell you if a syscall is thread-safe…or not