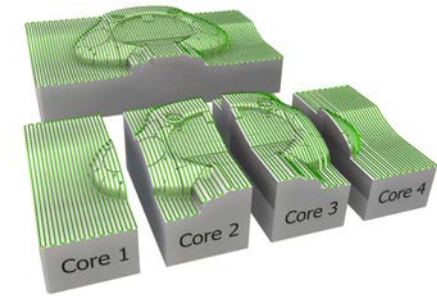




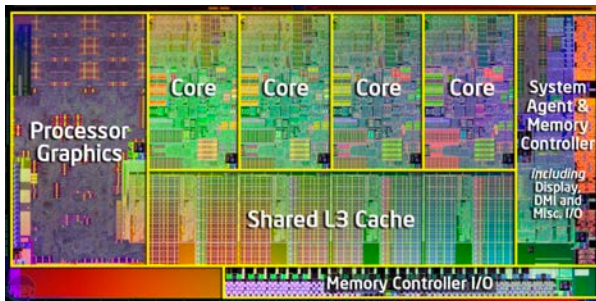
© Can Stock Photo - csp7049649



Welcome to CSE 160!

Introduction to parallel computation

Scott B. Baden



Welcome to Parallel Computation!

- Your instructor is *Scott B. Baden*
 - Office hours week 1: **Thursday after class**
 - baden+160@eng.ucsd.edu
- Your TAs – veterans of CSE 260
 - *Jingjing Xie*
 - *Karthikeyan Vasuki Balasubramaniam*
- Your Tutors – veterans of CSE 160
 - *John Hwang*
 - *Ryan Lin*
- Lab/office hours: After class (this Thursday)
- Section (attend 1 each week)
 - Wednesdays **4:00 to 4:50 pm**
 - Fridays **12:00 to 12:50 pm**
 - Bring your laptop

About me

- PhD at UC Berkeley
(High Performance Computing)
- Undergrad: Duke University
- 26th year at UCSD

```
AI ITS.853. DDT.535.
12. USERS
MEM ERRS 0.28506038E-1/HOUR
SCOTT$U
TO SEE SYSTEM MESSAGES, DO ":MSG$<CR>"
MSG$ JOB?
:TELNET
557) .SLEEP 11, $P
HELLO
STANFORD 6.15/X 04-13-74

.L NET.GUE
JOB 32 STANFORD 6.15/X 04-13-74
SUNDAY 14-APR-74 1735
32 JOBS LOGGED IN. 5 RUNNING.

EXIT
.R BASIC
NEW OR OLD--NW\WE\W PERFCT

READY
10 PRINT "RANGE"
10 PRINT "RANGE";
5 PRINT "RANGE";
7 INPUT A1,A2
10 FOR I=A1 TO A2
15 N=1
17 P=1
20 FOR F=2 TO SQR(I)
30 IF FIX(I/F)<>I/F THEN 50
40 N=N+F
42IF I/F<>F THEN 47
45 GOTO50
47N=N+I/F
50 NEXT F
60 IF N=1 THEN 65
62GOTO 70
65 PRINT I;"IS PERFECT"

91 IF N/G<>G THEN 95
RUNNH

RANGE 71,8\899\9999
1 IS PERFECT
6 IS PERFECT
28 IS PERFECT
220 AND 284 ARE FRIENDLY
496 IS PERFECT
1184 AND 1210 ARE FRIENDLY
2620 AND 2924 ARE FRIENDLY
5020 AND 5564 ARE FRIENDLY
6232 AND 6368 ARE FRIENDLY
8128 IS PERFECT
10744 AND 10856 ARE FRIENDLY
12285 AND 14595 ARE FRIENDLY
17296 AND 18416 ARE FRIENDLY

tCtC
tZ
(TYPEIN;) 557) .SLEEP 11, $$U
AI ITS 853 CONSOLE 4 FREE. 21:24:26
```



My Background

- I have been programing since 1971
HP Programmable calculators, Minicomputers, Supercomputers; Basic+, Algol/W, APL, Fortran, C/C++, Lisp, Matlab, CUDA, threads, Supercomputers,...
- I am an active coder, for research and teaching
- My research: techniques and tools that transform source code to change some aspect of performance for large scale applications in science and engineering
- We run parallel computations on up to 98,000 processors!

Reading

- Two required texts <http://goo.gl/SH98DC>
 - *An Introduction to Parallel Programming*, by Peter Pacheco, Morgan Kaufmann, 2011
 - *C++ Concurrency in Action: Practical Multithreading*, by Anthony Williams, Manning Publications, 2012
 - **Lecture slides are no substitute for reading the texts!**
- Complete the assigned readings before class
readings → pre-class quizzes → in class problems → exams
- All announcements will be made on-line
 - Course home page
<http://cseweb.ucsd.edu/classes/wi16/cse160-a>
 - Piazza (Announcement, Q&A)
 - Moodle (pre-class quizzes & grades only)
 - Register your clicker today!

Background

- Pre-requisite: CSE 100
- Comfortable with C/C++ programming
- If you took Operating Systems (CSE 120), you should be familiar with threads, synchronization, mutexes
- If you took Computer Architecture (CSE 141) you should be familiar with memory hierarchies, including caches
- We will cover these topics sufficiently to level the playing field

Course Requirements

- 4 Programming assignments (45%)
 - Multithreading with C++11 + performance programming
 - Assignments shall be done in teams of 2
- Exams (35%)
 - 1 Midterm (15%) + Final (20%)
 - `midterm = (final > midterm) ? final : midterm`
- On-line pre-class quizzes (10%)
- Class participation
 - Respond to 75% of clicker questions and you've participated in a lecture
 - No cell phone usage unless previously authorized. Other devices may be used for note-taking only

Cell phones?!? Not in class unless invited!



"It keeps me from looking at my phone every two seconds."

Policies

- Academic Integrity
 - Do you own work
 - Plagiarism and cheating will not be tolerated
- You are required to complete an Academic Integrity Scholarship Agreement (part of A0)

Programming Labs

- Bang cluster
- Ieng6
- Make sure your accounts work
- Software
 - C++11 threads
 - We will use Gnu 4.8.4
- Extension students:
Add CSE 160 to your list of courses
<https://sdacs.ucsd.edu/~icc/exadd.php>

Class presentation technique

- I will assume that you've read the assigned readings before class
- Consider the slides as talking points, class discussions driven by *your* interest
- Learning is not a passive process
- Class participation is important to keep the lecture active
- Different lecture modalities
 - The 2 minute pause
 - In class problem solving

The 2 minute pause



- Opportunity **in class** to improve your understanding, to make sure you “got” it
 - By trying to explain to someone else
 - Getting your mind actively working on it
- The process
 - I pose a question
 - You discuss with 1-2 neighbors
 - Important Goal: understand why the answer is correct
 - After most seem to be done
 - I’ll ask for quiet
 - A few will share what their group talked about
 - Good answers are those where you were wrong, then realized...
 - Or ask a question!



Please pay attention and quickly return to “lecture mode”
so we can keep moving!

Group Discussion #1

What is your Background?

- C/C++ Java Fortran?
- TLB misses
- Multithreading
- MPI
- RPC
- C++11 Async
- CUDA, OpenCL, GPUs
- Abstract base class



$$\nabla \cdot u = 0$$

$$\frac{D\rho}{Dt} + \rho(\nabla \cdot v) = 0$$

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$$

The rest of the lecture

- Introduction to parallel computation

What is parallel processing ?

- Compute on *simultaneously executing physical resources*
- Improve some aspect of performance
 - Reduce time to solution: multiple cores are faster than 1
 - Capability: Tackle a larger problem, more accurately
- Multiple processor cores co-operate to process a related set of tasks – *tightly coupled*
- What about distributed processing?
 - Less tightly coupled, unreliable communication and computation, changing resource availability
- Contrast *concurrency* with parallelism
 - Correctness is the goal, e.g. data base transactions
 - Ensure that shared resources are used appropriately

Group Discussion #2

Have you written a parallel program?

- Threads
- C++11 Async
- OpenCL
- CUDA
- RPC
- MPI



Why study parallel computation?

- Because parallelism is everywhere: cell phones, laptops, automobiles, etc.
- If you don't parallelism, you lose it!
 - Processors generally can't run at peak speed on 1 core
 - Many applications are underserved because they fail to use available resources fully
- But there are many details affecting performance
 - The choice of algorithm
 - The implementation
 - Performance tradeoffs
- The courses you've taken generally talked about how to do these things on 1 processing core only
- Lots of changes on multiple cores

How does parallel computing relate to other branches of computer science?

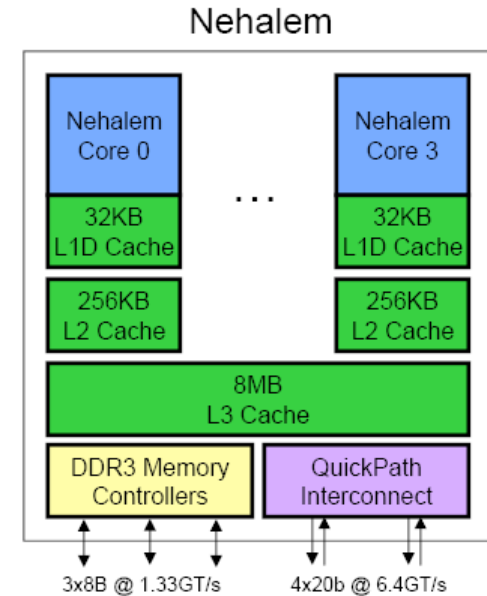
- Parallel processing generalizes problems we encounter on single processor computers
- A parallel computer is just an extension of the traditional memory hierarchy
- The need to preserve locality, which prevails in virtual memory, cache memory, and registers, also applies to a parallel computer

What you will learn in this class

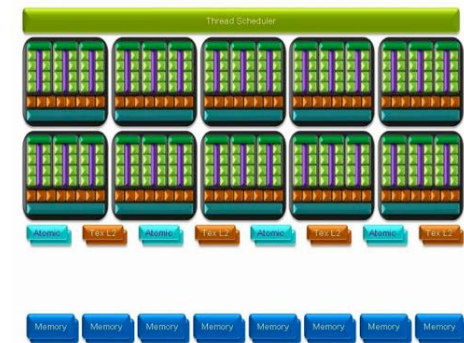
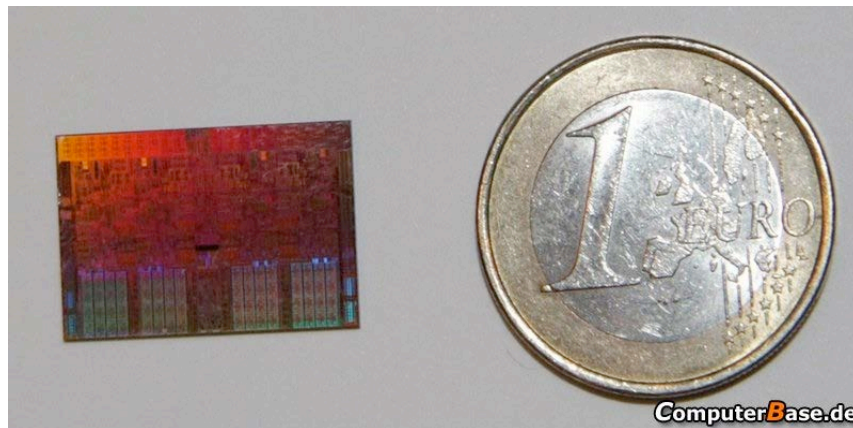
- How to solve computationally intensive problems on multicore processors effectively using threads
 - Theory and practice
 - Programming techniques, including performance programming
 - Performance tradeoffs, esp. the memory hierarchy
- CSE 160 will build on what you learned earlier in your career about programming, algorithm design and analysis

The age of the multi-core processor

- On-chip parallel computer
- IBM Power4 (2001), Intel, AMD ...
- First dual core laptops (2005-6)
- GPUs (nVidia, ATI): desktop supercomputer
- In smart phones, behind the dashboard blog.laptopmag.com/nvidia-tegrak1-unveiled
- Everyone has a parallel computer at their fingertips

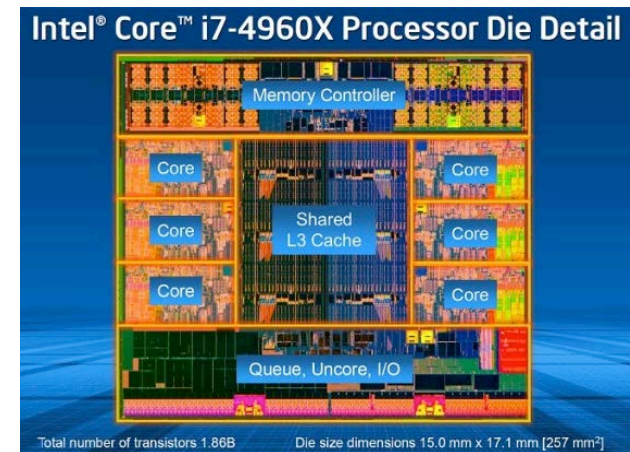
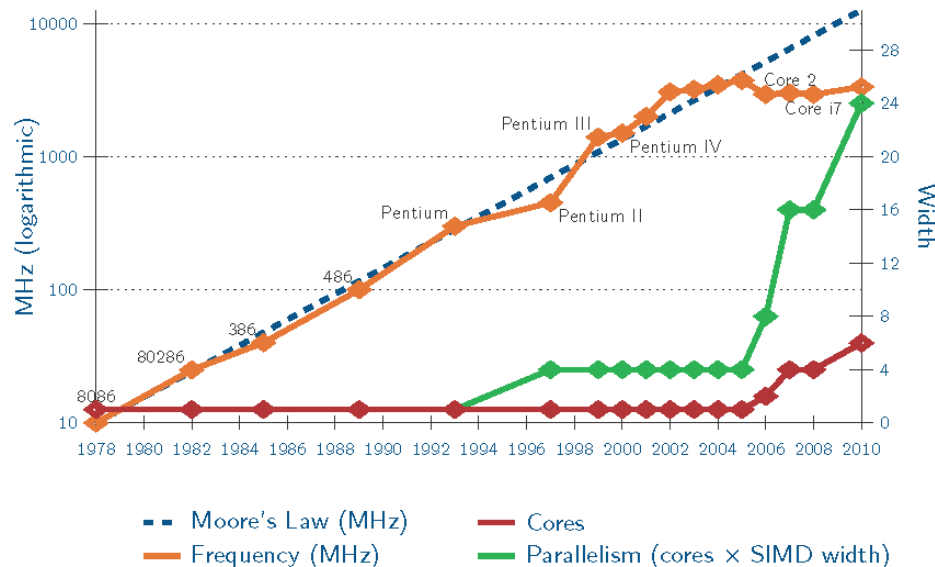


realworldtech.com



Why is parallel computation inevitable?

- Physical limitations on heat dissipation prevent further increases in clock speed
- To build a faster processor, we replicate the computational engine

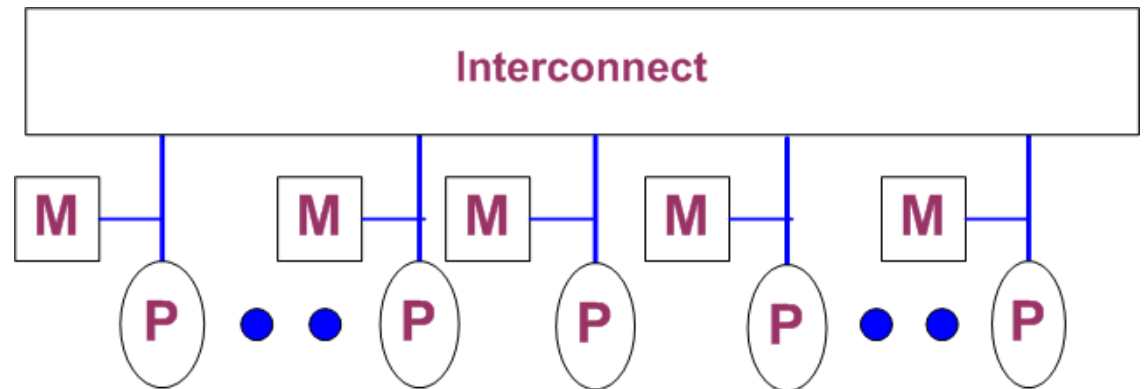
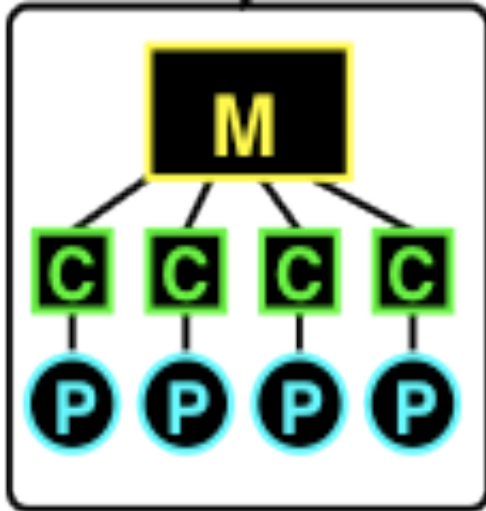


<http://www.neowin.net/>

Christopher Dyken, SINTEF

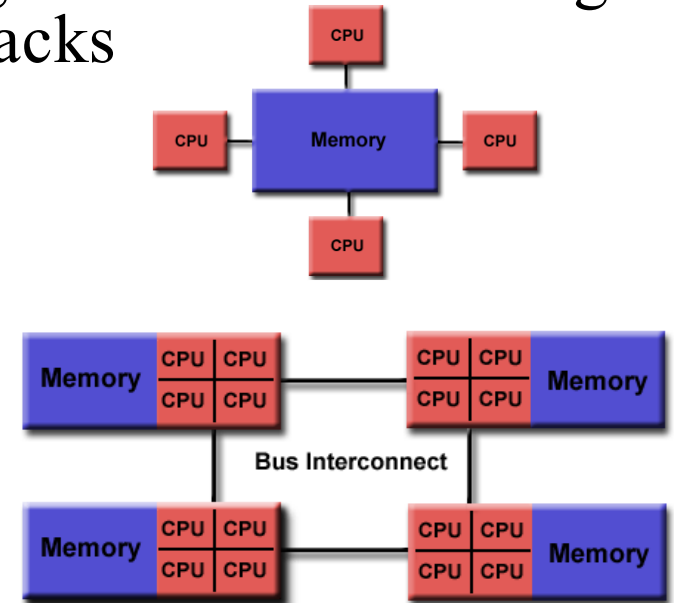
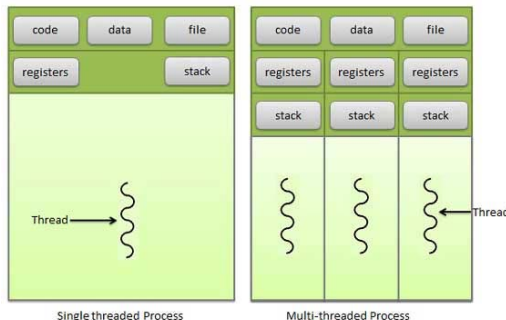
The anatomy of a multi-core processor

- MIMD
 - Each core runs an independent instruction stream
- All share the global memory
- 2 types, depends on uniformity of memory access times
 - **UMA:** *Uniform* Memory Access time
Also called a Symmetric Multiprocessor (SMP)
 - **NUMA:** *Non-Uniform* Memory Access time



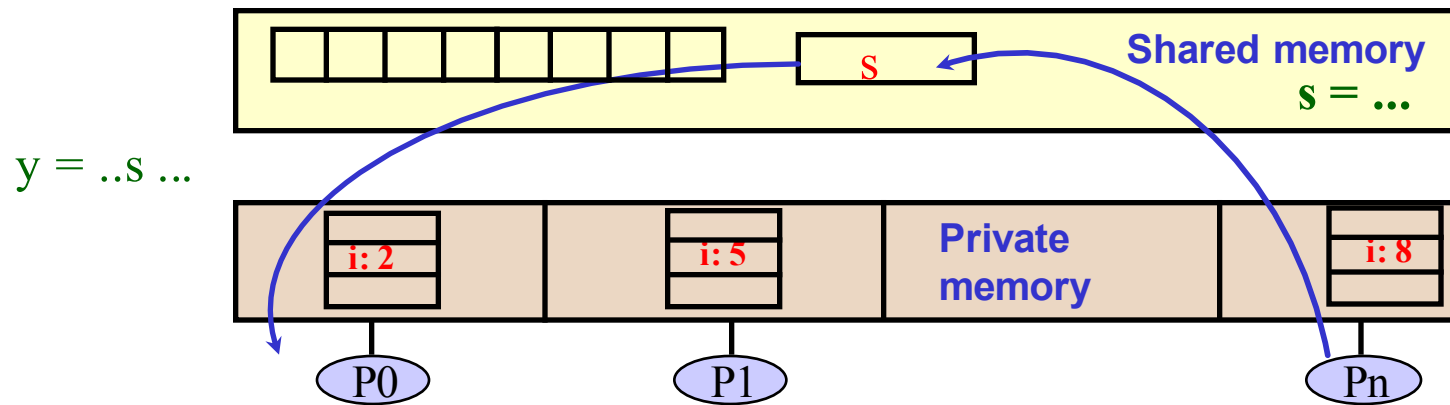
Multithreading

- How do we explain how the program runs on the hardware?
 - On shared memory, a natural programming model is called *multithreading*
 - Programs execute as a set of *threads*
 - Threads are usually assigned to different physical cores
 - Each thread runs the same code as an independent instruction stream
- Same Program Multiple Data programming model = “SPMD”**
- Threads communicate implicitly through shared memory (e.g. the heap), but have their own private stacks
 - They coordinate (synchronize) via shared variables



What is a thread?

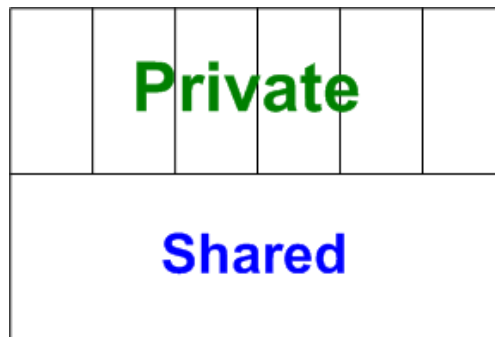
- A thread is similar to a procedure call with notable differences
- The control flow changes
 - A procedure call is “synchronous;” return indicates completion
 - A spawned thread executes asynchronously until it completes, and hence a return doesn’t indicate completion
- A new storage class: shared data
 - Synchronization may be needed when updating shared state (*thread safety*)



CLICKERS OUT

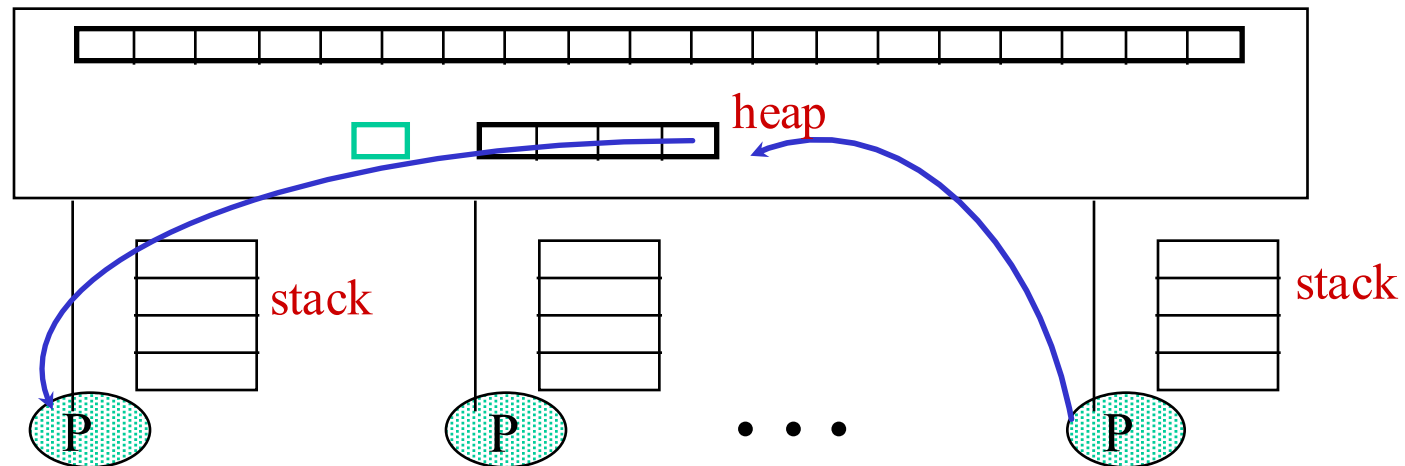
Which of these storage classes can never be shared among threads?

- A. Globals declared outside any function
- B. Local automatic storage
- C. Heap storage
- D. Class members (variables)
- E. B & C



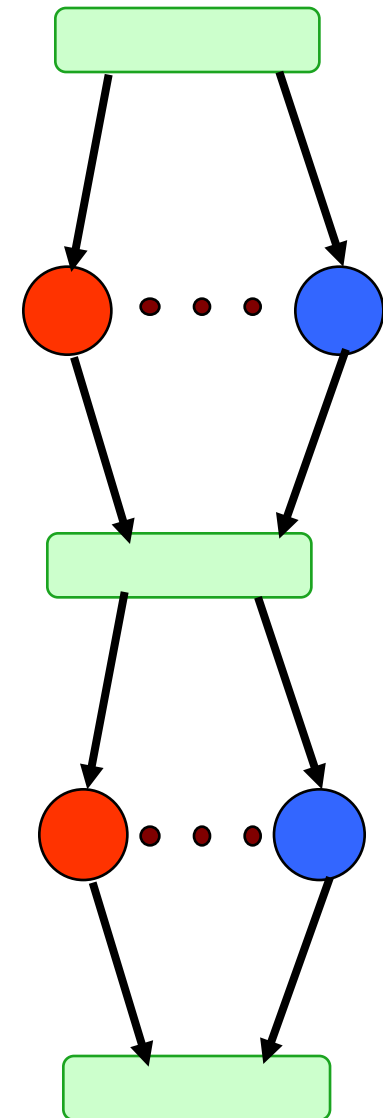
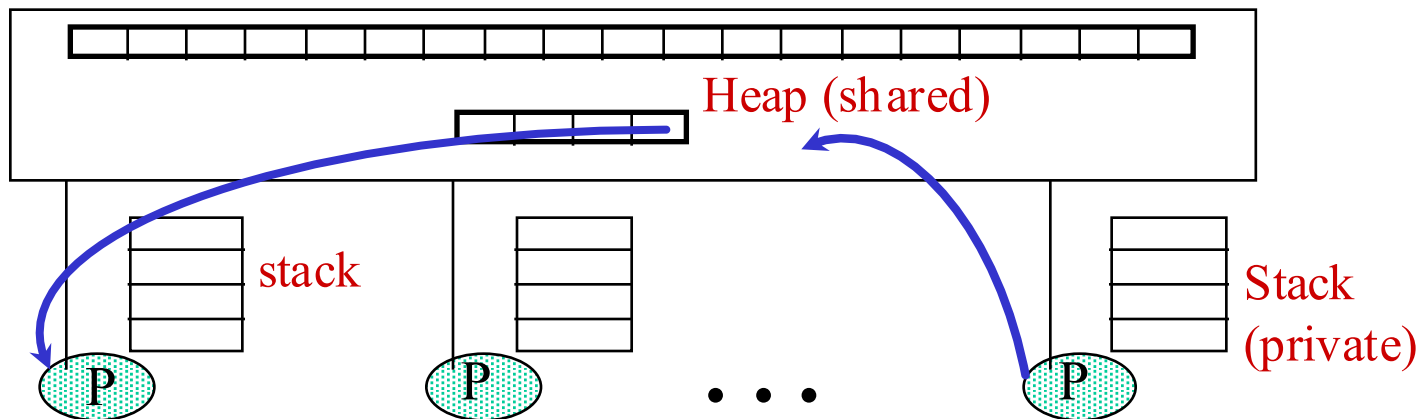
Why threads?

- Processes are “heavy weight” objects scheduled by the OS
 - Protected address space, open files, and other state
- A thread AKA a lightweight process (LWP)
 - Threads share the address space and open files of the parent, but have their own stack
 - Reduced management overheads, e.g. thread creation
 - Kernel scheduler multiplexes threads



Parallel control flow

- Parallel program
 - Start with a single root thread
 - Fork-join parallelism to create concurrently executing threads
 - Threads communicate via shared memory
- A spawned thread executes asynchronously until it completes
- Threads may or may not execute on different processors



What forms of control flow do we have in a serial program?

- A. Function Call
- B. Iteration
- C. Conditionals (if-then-else)
- D. Switch statements
- E. All of the above

Multithreading in Practice

- C++11
- POSIX Threads “standard” (pthreads):
IEEE POSIX 1003.1c-1995
 - Low level interface
 - Beware of non-standard features
- OpenMP – program annotations
- Java threads not used in high performance computation
- Parallel programming languages
 - Co-array FORTRAN
 - UPC

C++11 Threads

- Via `<thread>`, C++ supports a threading interface similar to pthreads, though a bit more user friendly
- *Async* is a higher level interface suitable for certain kinds of applications
- New memory model
- Atomic template
- Requires C++11 compliant compiler, gnu 4.7+, etc.

Hello world with <Threads>

```
#include <thread>
void Hello(int TID) {
    cout << "Hello from thread " << TID << endl;
}

int main(int argc, char *argv[ ]){
    thread *thrds = new thread[NT];

    // Spawn threads
    for(int t=0;t<NT;t++){
        thrds[t] = thread(Hello, t );
    }

    // Join threads
    for(int t=0;t<NT;t++)
        thrds[t].join();
}
```

```
$ ./hello_th 3
Hello from thread 0
Hello from thread 1
Hello from thread 2
$ ./hello_th 3
Hello from thread 1
Hello from thread 0
Hello from thread 2
$ ./hello_th 4
Running with 4 threads
Hello from thread 0
Hello from thread 3
Hello from thread Hello from
thread 21
```

\$PUB/Examples//Threads/Hello-Th

PUB = /share/class/public/cse160-wi16

Steps in writing multithreaded code

- We write a *thread function* that gets called each time we spawn a new thread
- *Spawn* threads by constructing objects of class Thread (in the C++ library)
- Each thread runs on a separate processing core (If more threads than cores, the threads share cores)
- Threads share memory, declare shared variables outside the scope of any functions
- Divide up the computation fairly among the threads
- *Join* threads so we know when they are done

Summary of today's lecture

- The goal of parallel processing is to improve some aspect of performance
- The multicore processor has multiple processing cores sharing memory, the consequence of technological factors
- We will employ multithreading in this course to “parallelize” applications
- We will use the C++ threads library to manage multithreading

Next Time

- Multithreading
- Be sure your clicker is registered
- By Friday at 6pm:
do Assignment #0
cseweb.ucsd.edu/classes/wi16/cse160-a/HW/A0.html
- Establish that you can login to
bang and ieng6
cseweb.ucsd.edu/classes/wi16/cse160-a/lab.html