Lecture 16 – Parallel Architecture and Algorithms and OpenMP

Prof. Brendan Kochunas

NERS/ENGR 570 - Methods and Practice of Scientific Computing (F22)

COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Outline

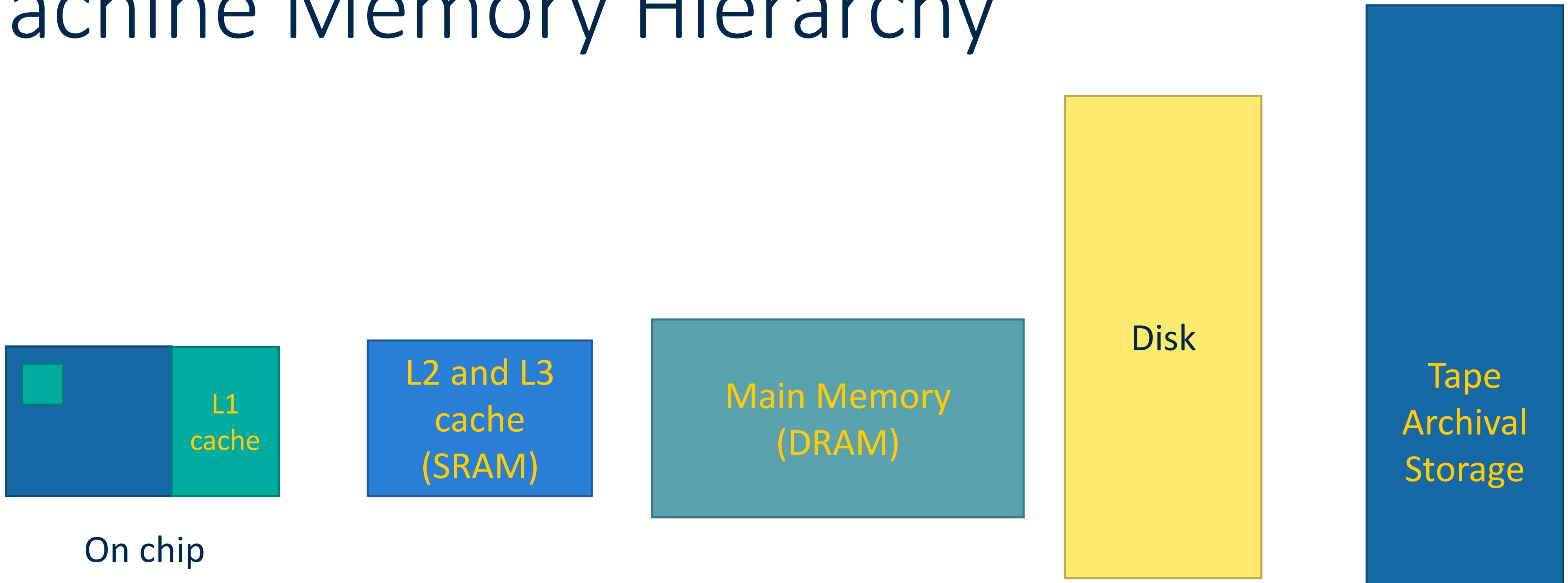
- Overview of Parallel Architectures
- General Types of Parallel Algorithms
- Shared Memory Execution Model
- Hands on Introduction to OpenMP

Learning Objectives: By the end of Today’s Lecture you should be able to

- (*Knowledge*) describe the difference of shared and distributed parallel computing
- (*Knowledge*) list a couple algorithmic models for parallel programming
- (*Knowledge*) Describe how shared memory programs run on a computer

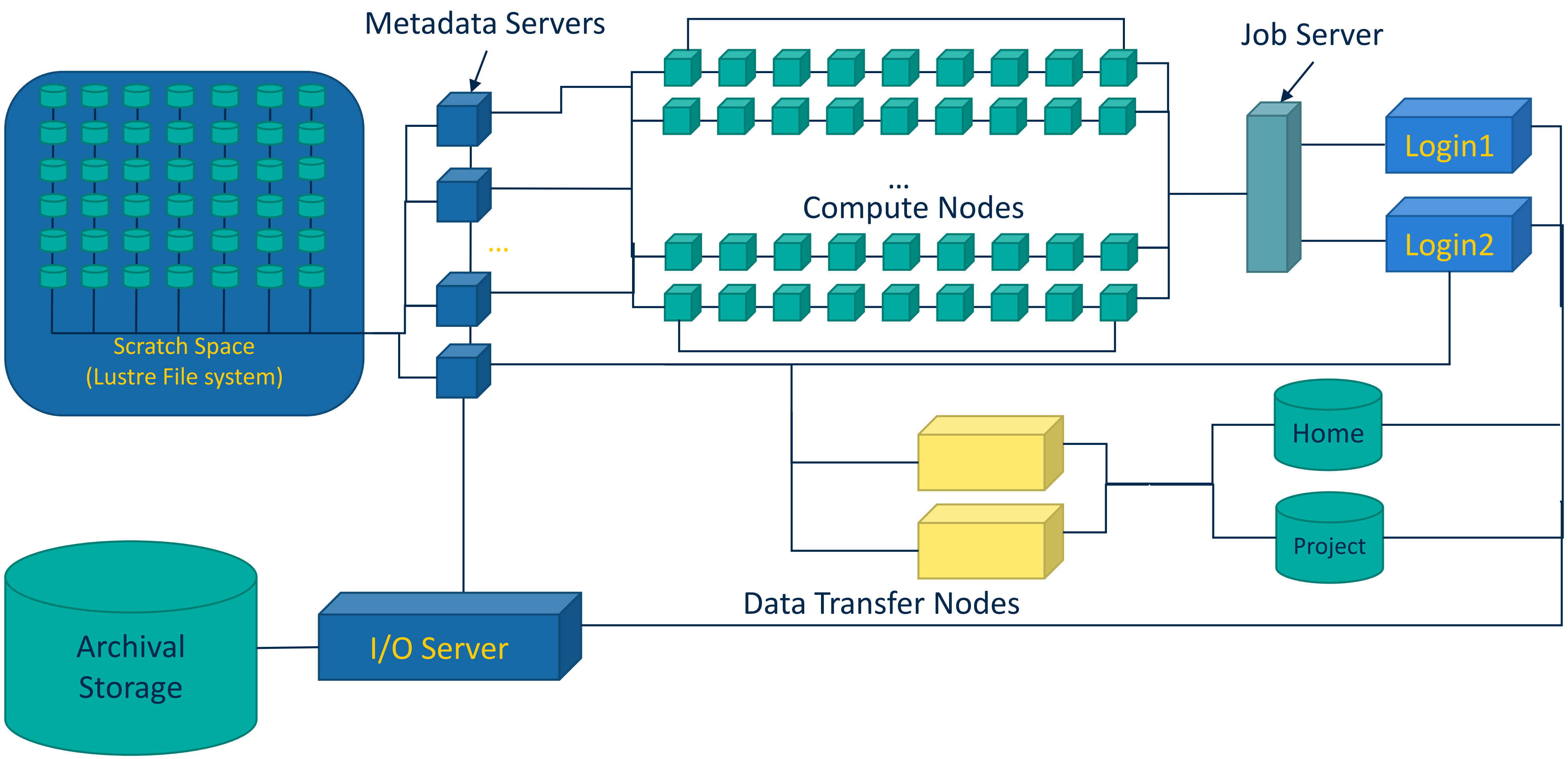
Parallel Architecture

Serial Machine Memory Hierarchy

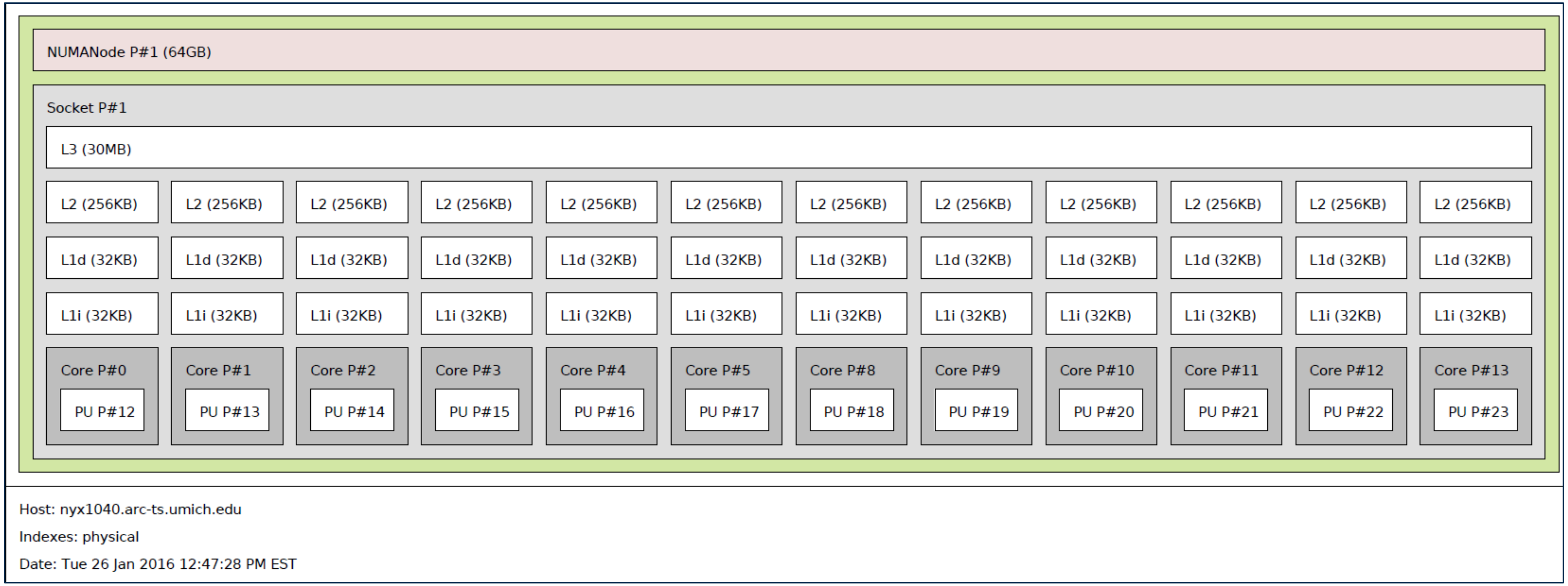


	Register	**L1**	**L2**	**L3**	**DRAM**	**Disk**	**Tape**
Size	< 1 KB	~1KB	1 MB	10's MB	1-100's GB	TB	PB
Speed	< 1ns	<1 ns	~1 ns	~1-10 ns	10-100 ns	10 ms	~10s

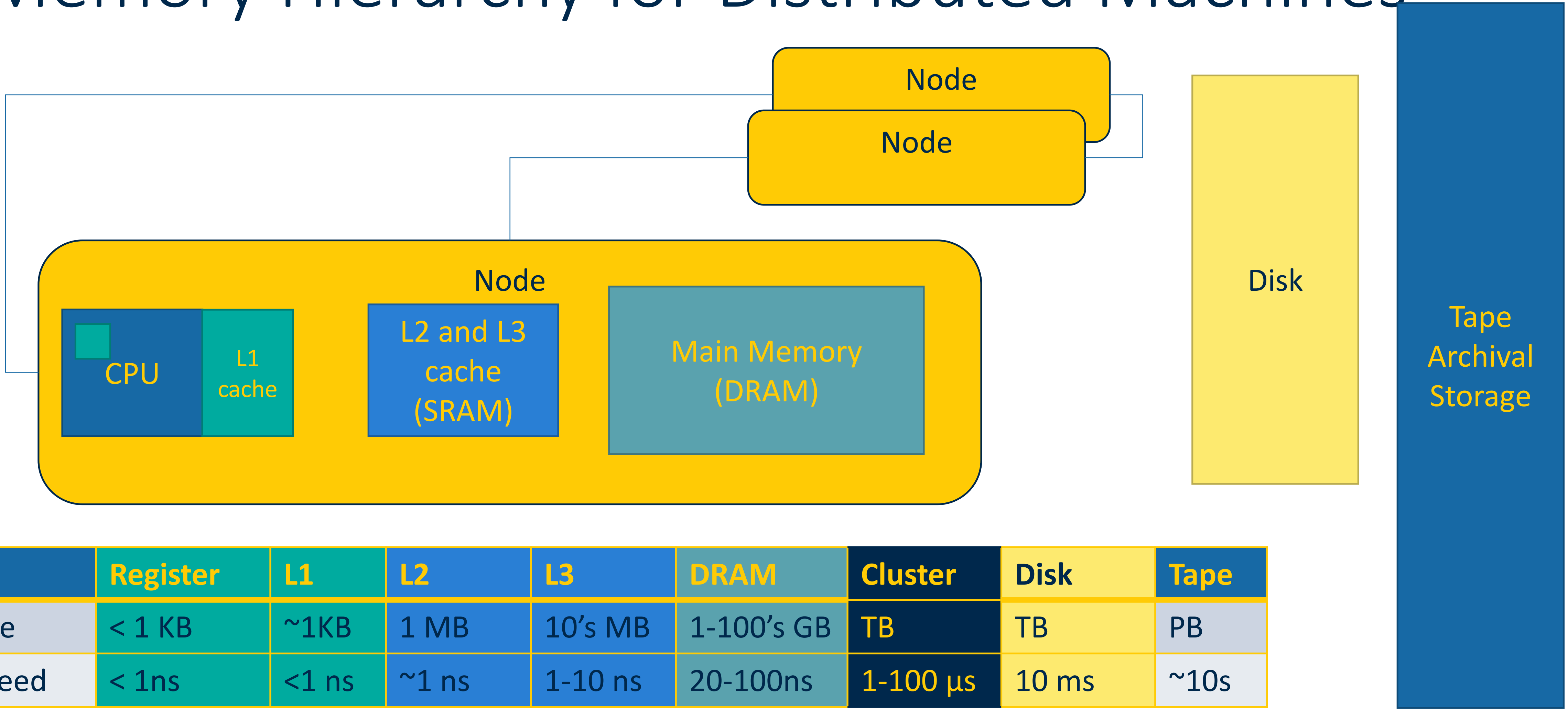
Contemporary HPC Platforms



Note about node hardware



Memory Hierarchy for Distributed Machines

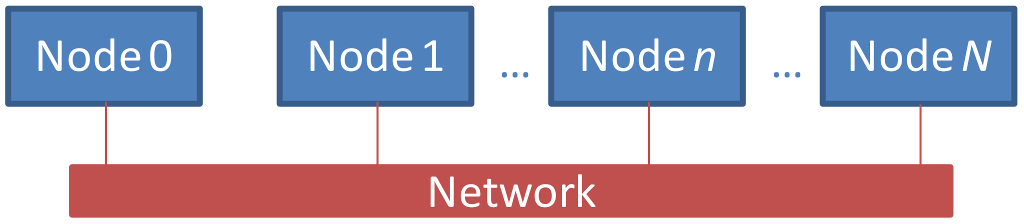


	Register	L1	L2	L3	DRAM	Cluster	Disk	Tape
Size	< 1 KB	~1KB	1 MB	10's MB	1-100's GB	TB	TB	PB
Speed	< 1ns	<1 ns	~1 ns	1-10 ns	20-100ns	1-100 µs	10 ms	~10s

Types of Parallel Algorithms

Distributed Memory Parallelism

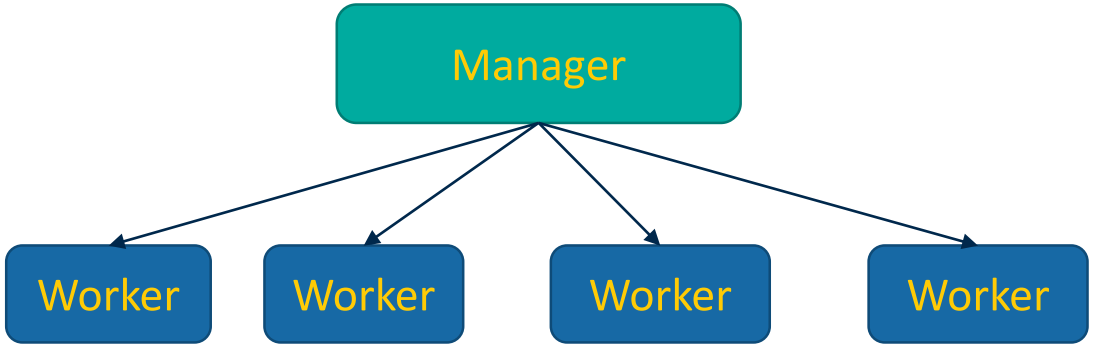
- Each process has its own memory.
  - Data between processes must be explicitly communicated.
- Usually more difficult to convert serial programs to distributed memory execution models
- Generally much easier to design software from ground up to run with distributed memory
- Common programming models
  - MPI
  - Unified Parallel C (UPC), Fortran Co-arrays



Typical Algorithms for Distributed Memory Parallelism

Manager/Worker

- Master usually does more variety of work (e.g. I/O)
- Master controls execution of workers. Sends workload to workers



Bulk Synchronous

- Periodic synchronization
- Large workloads on processors between synchronization.

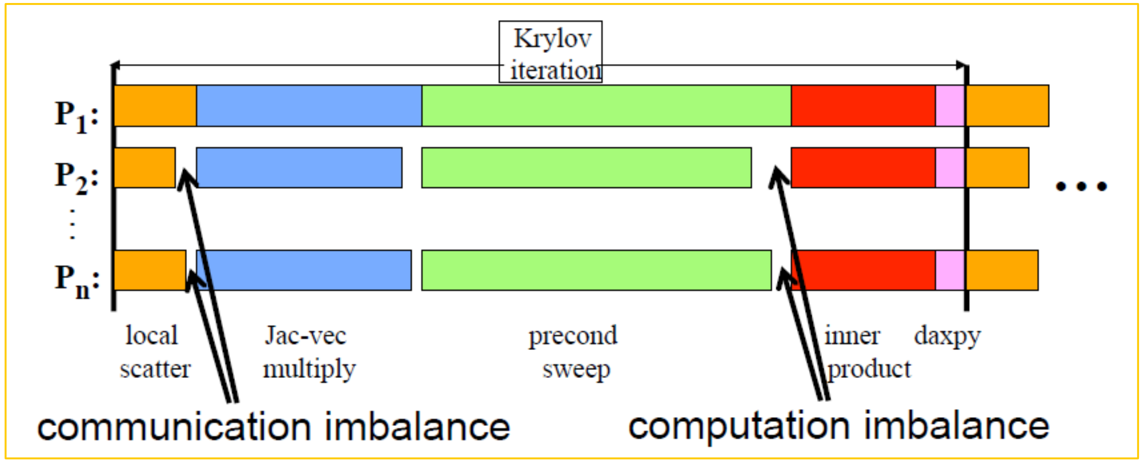
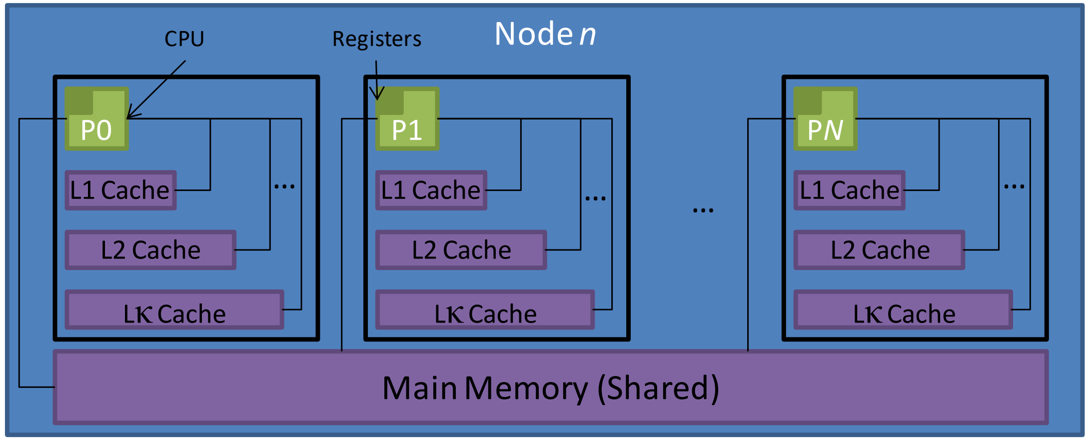


Figure from: D. Keyes, “Algorithmic Adaptations to Extreme Scale Computing”, ATPESC Workshop Presentation, (2013).

Shared Memory Parallelism

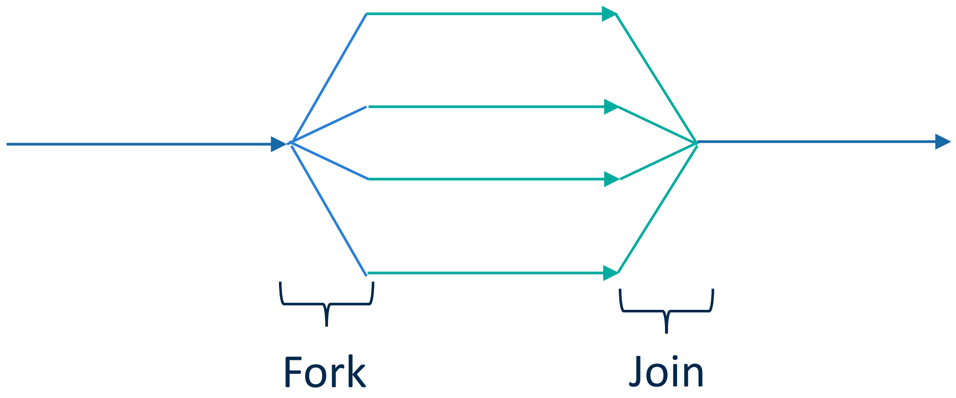
- All processes "see" the same memory.
  - Changes by one process to main memory are visible to all processes
- Usually low overhead to implement with current programming models
  - Not always easy to get good performance
- Common programming models
  - pthreads (POSIX)
  - OpenMP
  - Kokkos
  - Intel Thread Building Blocks (TBB)



Typical Algorithms in Shared Memory Parallelism

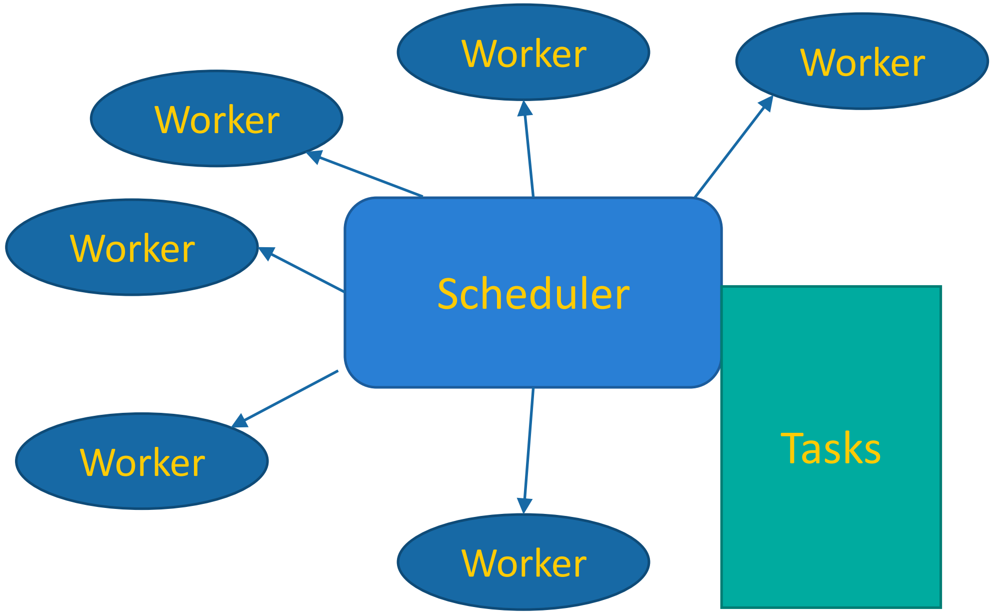
Fork/Join

- Simple loop parallelization



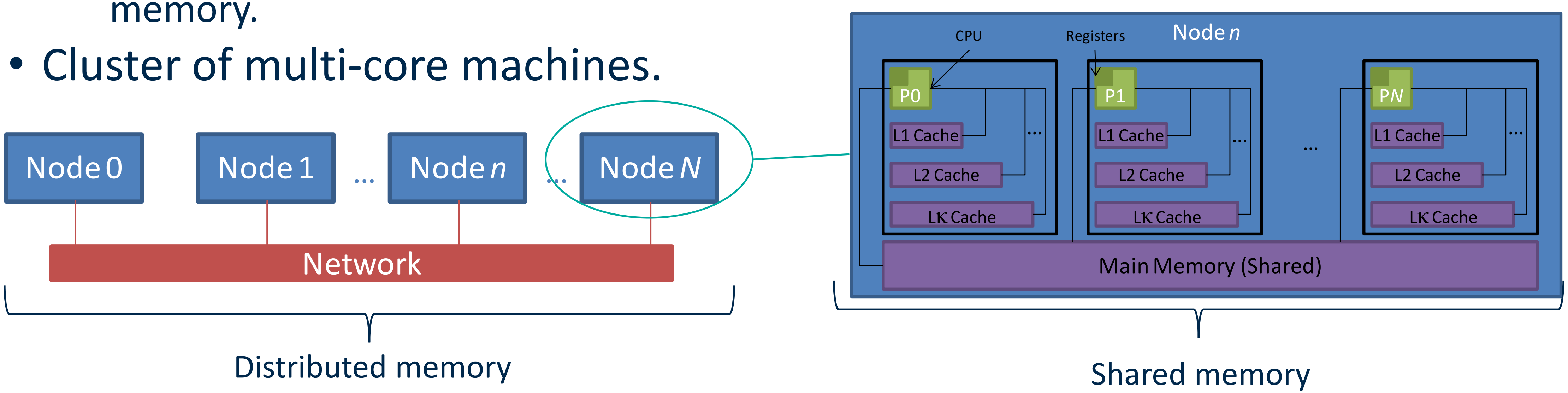
Pool of Tasks

- Tasks and work assignment are usually dynamic.



Hybrid Parallelism

- You guessed it, combines distributed and shared memory.
- This is representative of most modern compute clusters.
  - But remember these machines are configured to be able to run flexibly as either purely distributed, hybrid, or (if the programming model exists) purely shared memory.
- Cluster of multi-core machines.

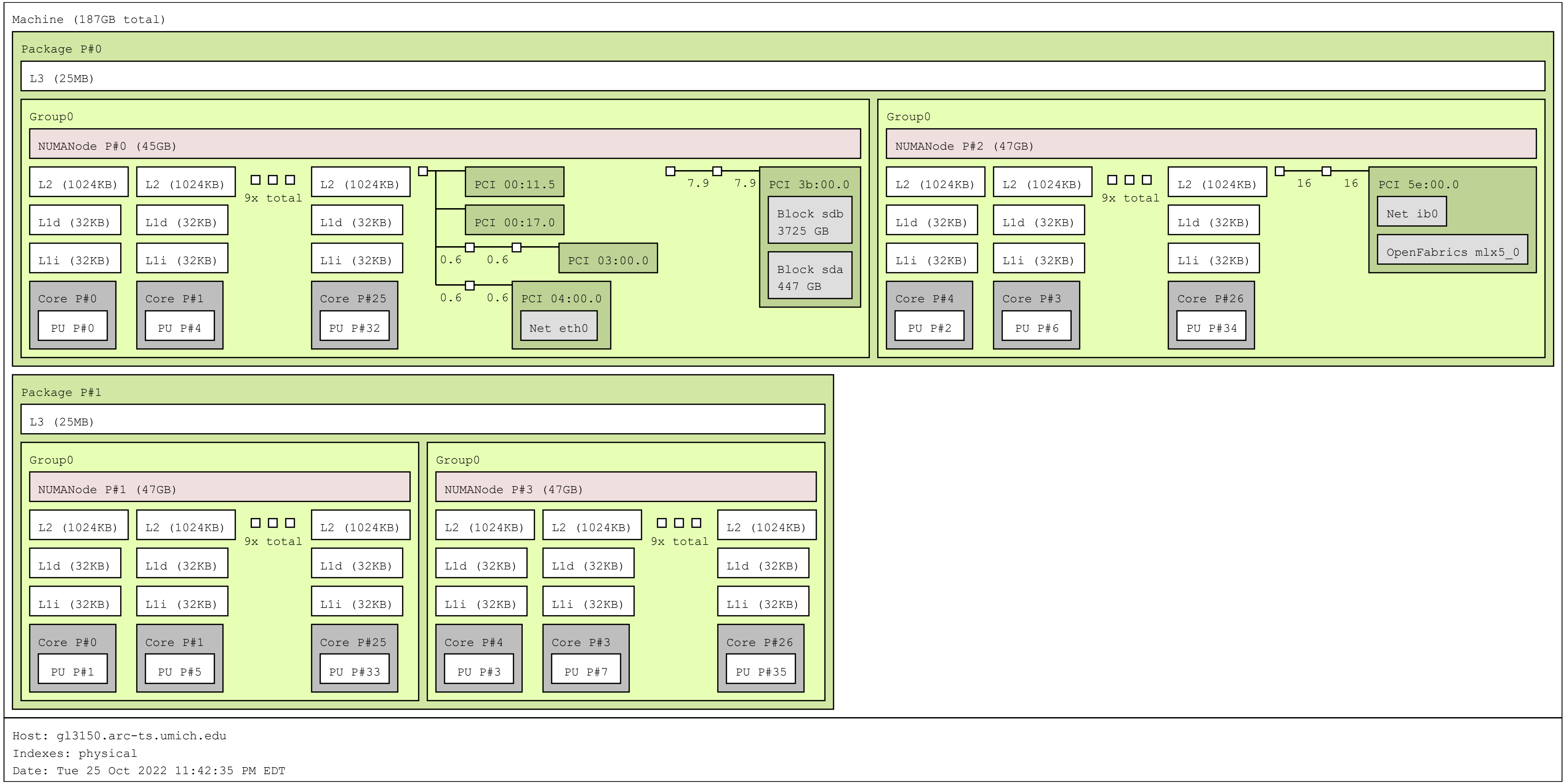


A few closing points

- Distributed memory algorithms and shared memory algorithms are not necessarily mutually exclusive
  - e.g. your code may make use of some combination of these
- There are other types of algorithms, but these are the “most common”
- Generally, parallel algorithms typically require some definition of how the memory is treated between the parallel processes
  - This can be abstracted away from the hardware.

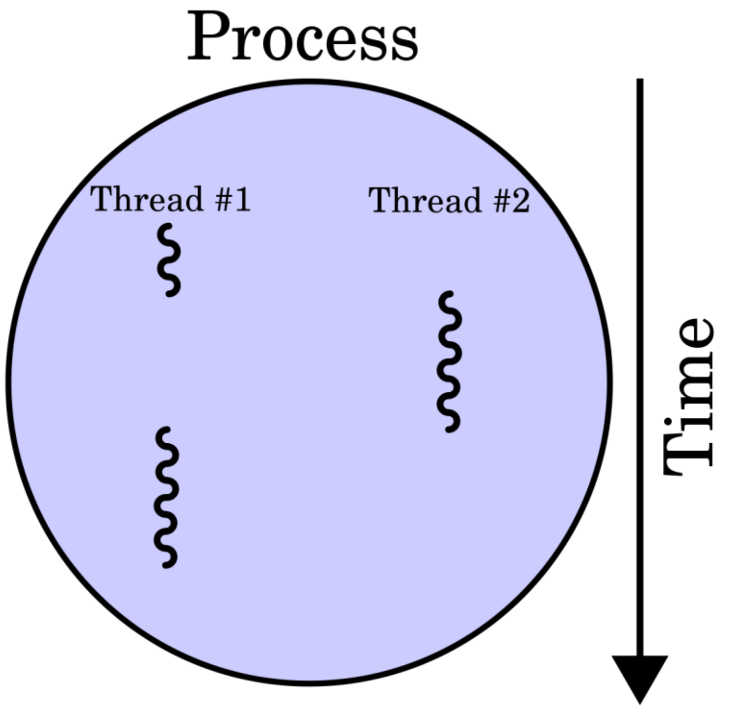
Shared Memory Execution Model

Great Lakes Node Architecture (Extent of hardware to consider with OpenMP)



Concept of a Thread

- Ability for the hardware/operating system to execute multiple processes *concurrently*
  - Typically process = thread
  - In multi-threading a process can have multiple threads
  - Usage of "process" and "thread" is confusing
- In Linux the `top` command (short for table of processes) lists all processes
  - These are basically threads
- Bottom line is that *a thread is a software entity*, not a hardware entity



Thread Affinity

- Affinity - association of thread (software) with core (hardware)
  - This is not guaranteed.
  - By default OS and OpenMP runtime library control this.
- Threads can “drift” from core to core during execution
- Fortunately, thread affinity can be controlled

OpenMP in 2 slides

Then we do some examples

Programming Shared Memory Parallelism with OpenMP

- Most of the constructs in OpenMP are ***compiler directives***.
  - C/C++ `#pragma omp <construct> [<clause> [<clause>] ...]`
  - Fortran `!$OMP <construct> [<clause> [<clause>] ...]`
- Examples
  - `#pragma omp parallel private(x)`
  - `!$OMP parallel private(x)`
- Function interface declarations and compile time constants and types in either:
  - `#include <omp.h>`
  - `USE OMP_LIB`
- Most OpenMP constructions apply to a “structured block”.
  - Structured block: a block of one or more statements with one point of entry at the top and one point of exit at the bottom
  - Examples: in C/C++ anything inside “{}”; in Fortran its loops, subroutines, functions, etc.

Enabling OpenMP

Switches for compiling and linking

Compiler	Flag
GNU gcc/g++/gfortran	-fopenmp
PGI pgcc/pgf90	-mp
Intel (Windows) icl/ifort	/Qopenmp
Intel (Linux/OSX) icc/icpc/ifort	-fopenmp
IBM xlc/xlcxx/xlf77/xlf90/xlf95/xlf2003	-qsmp
NAG nagfor	-openmp
Cray	-h omp

Examples

Hello World Example

C/C++

Serial

```
#include <stdio.h>

int main ()
{




  printf("Hello World \n");


}
```

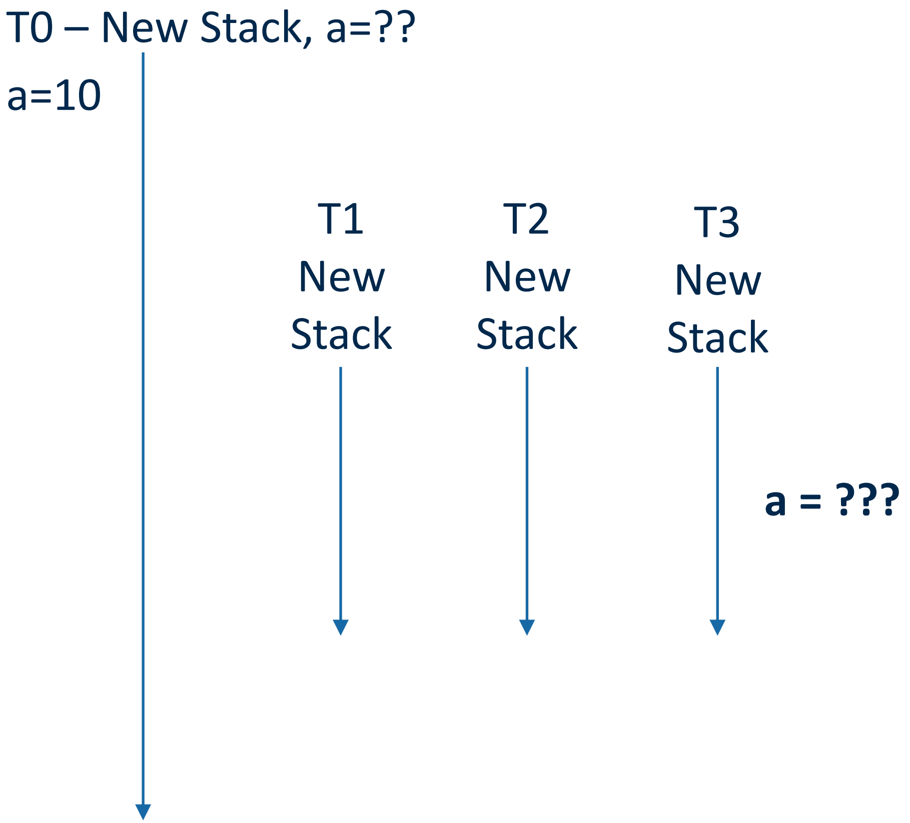
Threaded

```
#include <stdio.h>
#include <omp.h>
int main ()
{
  omp_set_num_threads(4);
   #pragma omp parallel
   {
      int id = omp_get_thread_num();
      printf("Hello World from thread = %d", id);
      printf(" with %d threads\n",omp_get_num_threads());
   }
}
```

Data Environment

Consider the following scenario

```
int a;
a=10
omp_set_thread_num(4);
#pragma omp parallel
{
  int id = omp_get_thread_num();
  printf("On thread = %d, a=%d\n", id, a);
}
```



Data Environment Default Behavior

- Most variables are shared
  - Actual behavior depends on how/where variable is defined

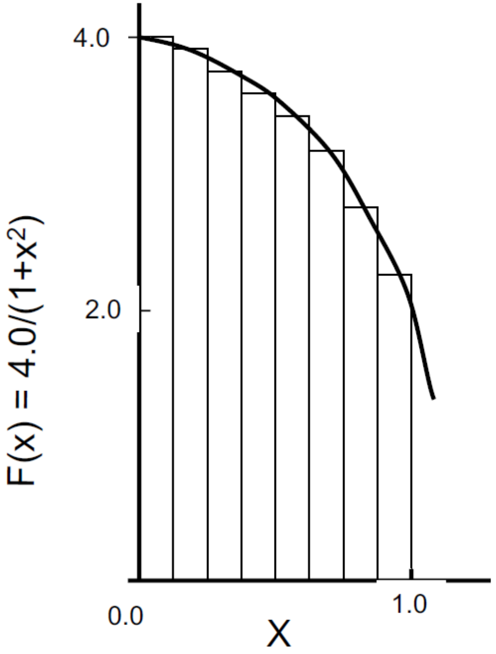
- Global variables default to SHARED
  - In Fortran: COMMON blocks, variables with SAVE attribute, and module variables, dynamically allocated arrays
  - In C/C++: file scope variables, static variables, and dynamically allocated memory
- Default private variables include
  - Stack variables and automatic variables
- Default behavior can be declared explicitly with default clause
  - `default(none|shared|private)`

Controlling Data Environment

- When declaring new parallel sections, OpenMP provides clauses for defining the data environment.
  - `shared` – variable retains one copy in memory, threads do not duplicate anything
  - `private` – specify which variables are private amongst threads
    - Creates local copies of variables. Variables have typical automatic definitions of serial code (e.g. declared but not defined). Note fixed sized arrays are duplicated!
- Special cases
  - `firstprivate` – create local copies and initialize all of them to their state just before the parallel construct. Note this duplicates all arrays!
  - `lastprivate` – variable is set equal to the private version of whichever thread executes the final iteration of for-loop or last section of sections construct.

Calculate π

Numerical Integration of π



$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

Approximate as a summation of rectangles (midpoint rule)

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Each rectangle has width Δx and height $F(x_i)$ at the middle of the interval i.