Lecture 16 – Parallel Architecture and Algorithms

Prof. Brendan Kochunas

NERS/ENGR 570 - Methods and Practice of Scientific Computing (F20)

COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Outline

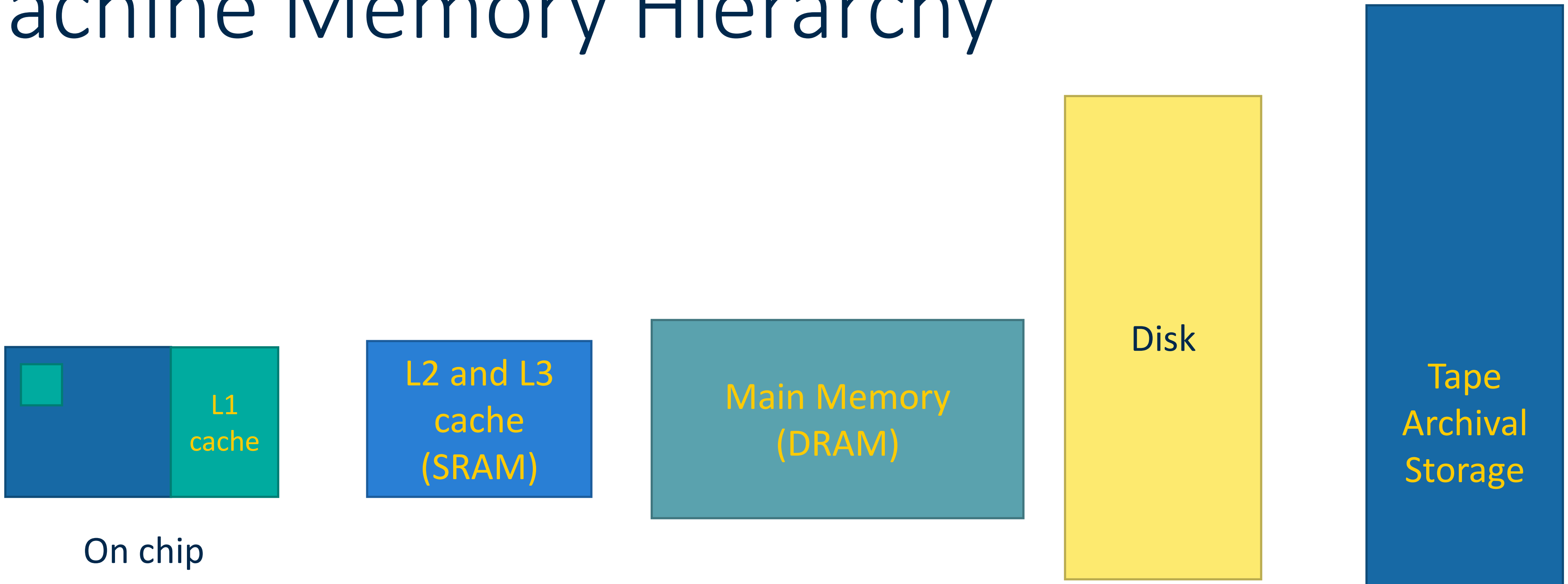
- Overview of Parallel Architectures
- General Types of Parallel Algorithms
- Parallel Algorithm “Ingredients”
- Shared Memory Execution Model
- (Time permitting) Introduction to OpenMP

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

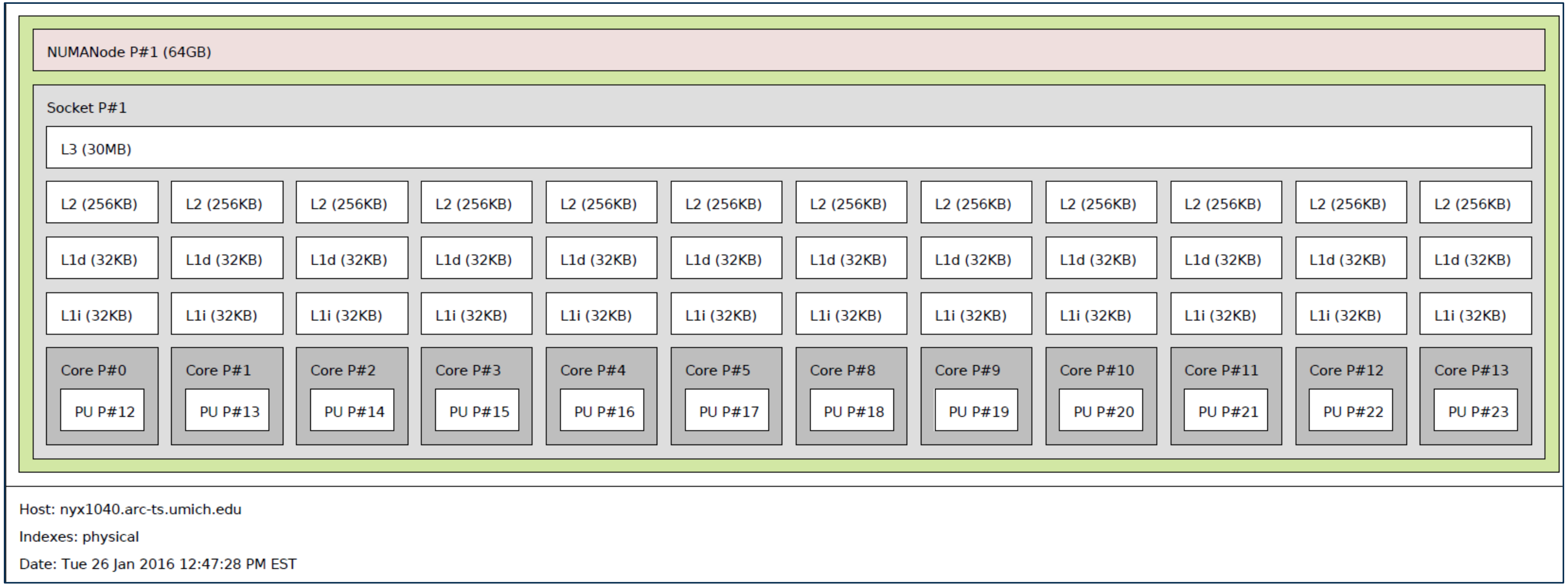
Parallel Architecture

Serial Machine Memory Hierarchy

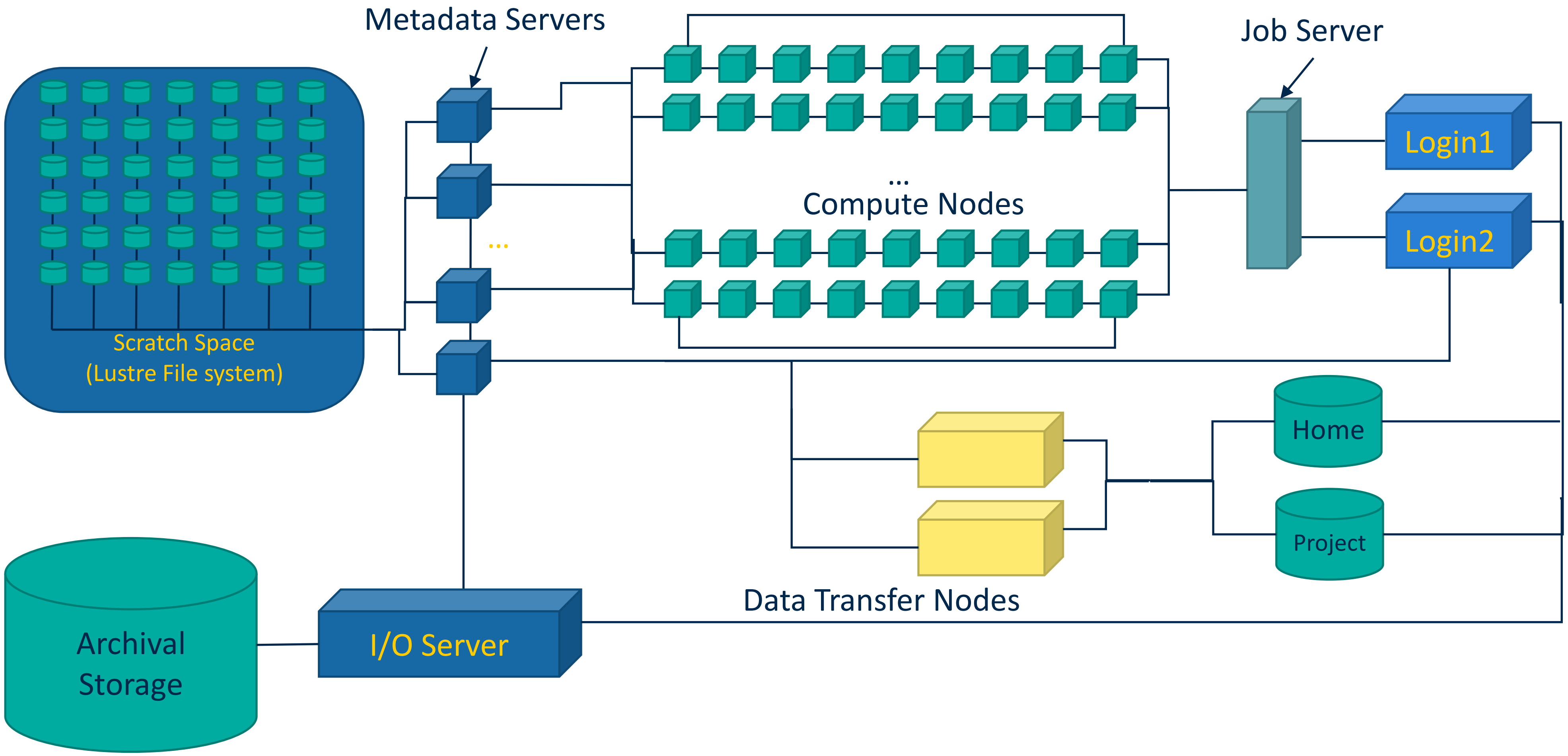


	Register	L1	L2	L3	DRAM	Disk	Tape
Size	< 1 KB	~1KB	1 MB	10's MB	1-100's GB	TB	PB
Speed	< 1ns	<1 ns	~1 ns	~1-10 ns	10-100 ns	10 ms	~10s

Note about node hardware



Contemporary HPC Platforms



Cores, Processors, and Nodes OH MY!

- We have been narrowly focused on a “single core”.
  - Many processors are “multi-core”
  - Common for motherboards to have multiple “sockets” or processors
  - A node has one motherboard, with multiple processors, and each processor has multiple cores
- Other terminology
  - Symmetric Multi-processor (SMP)
  - Non-Uniform Memory Access (NUMA)

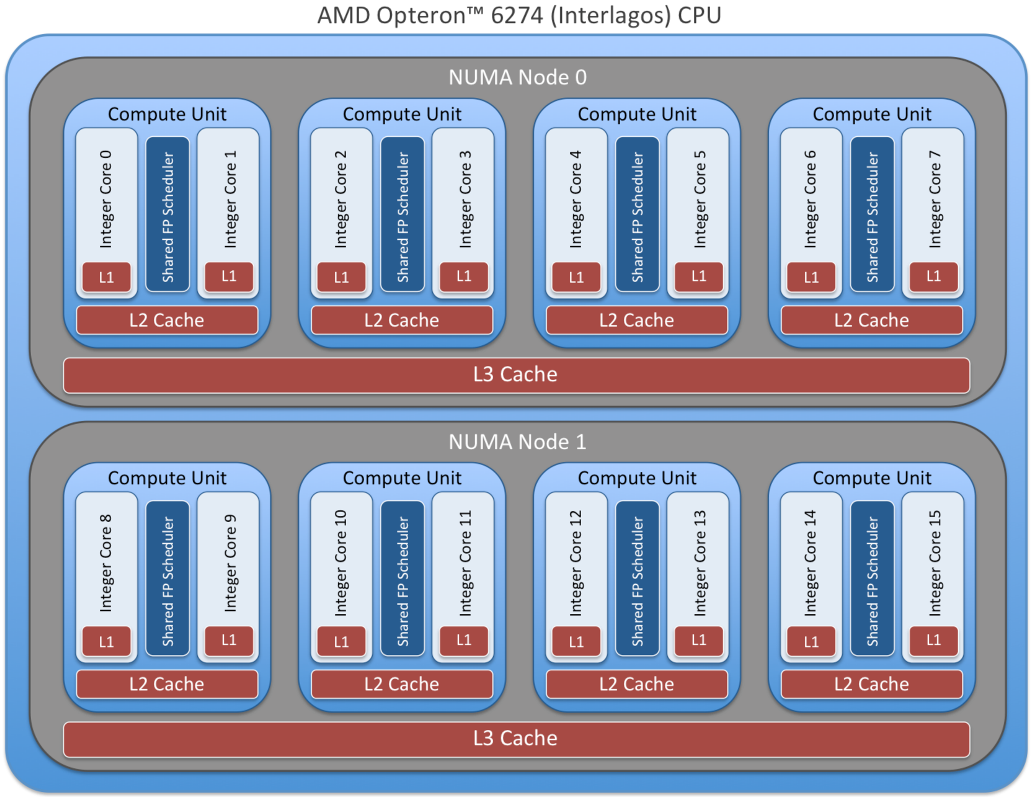
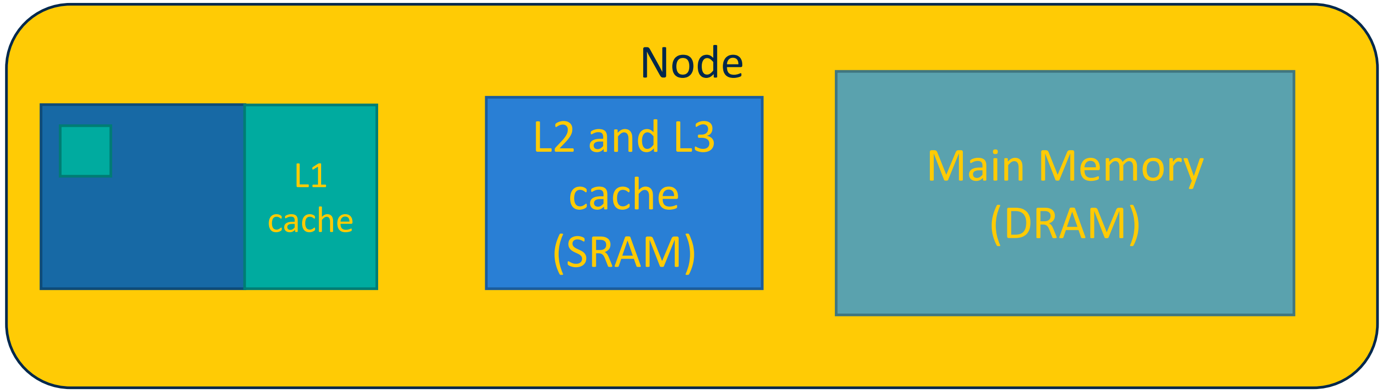
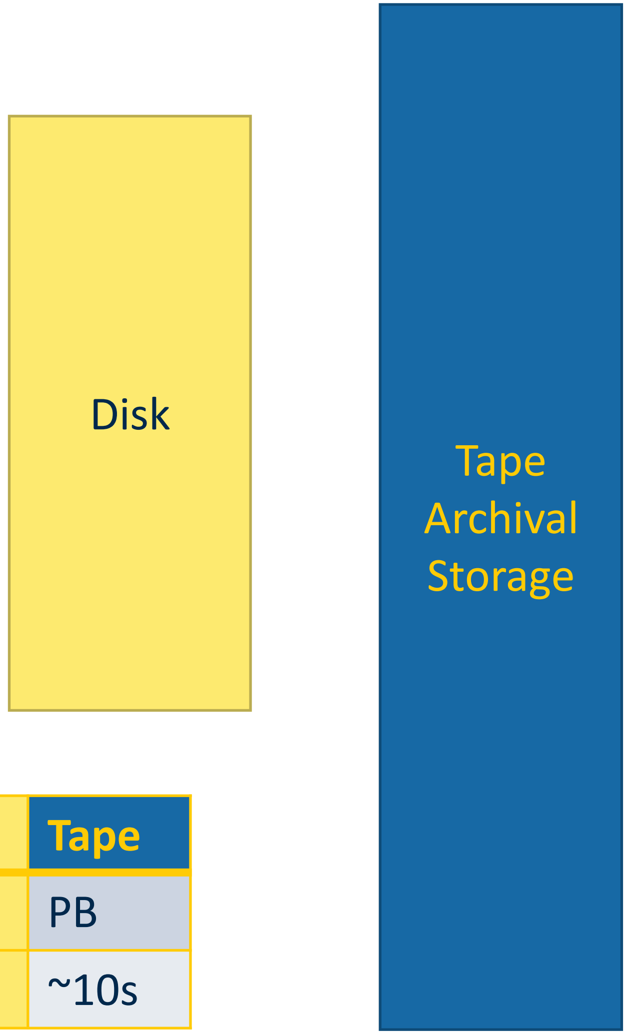


Illustration of Titan Compute Node

Memory Hierarchy for Distributed Machines



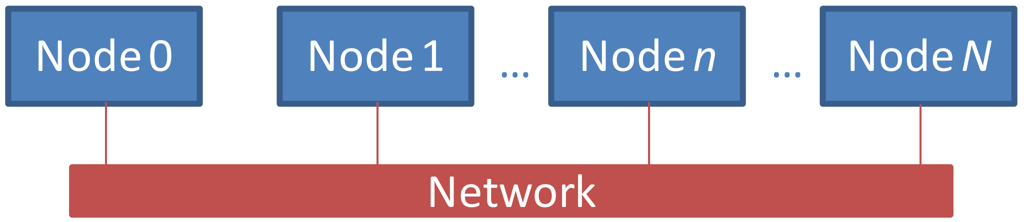


	Register	L1	L2	L3	DRAM	Cluster	Disk	Tape
Size	< 1 KB	~1KB	1 MB	10's MB	1-100's GB	TB	TB	PB
Speed	< 1ns	<1 ns	~1 ns	1-10 ns	20-100ns	1-100 µs	10 ms	~10s

Types of Parallel Algorithms

Distributed Memory Parallelism

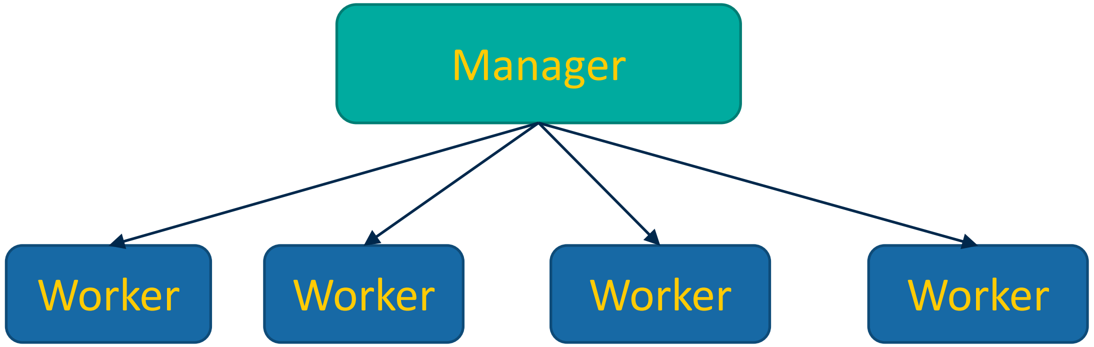
- Each process has its own memory.
  - Data between processes must be explicitly communicated.
- Usually more difficult to convert serial programs to distributed memory execution models
- Generally much easier to design software from ground up to run with distributed memory
- Common programming models
  - 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.

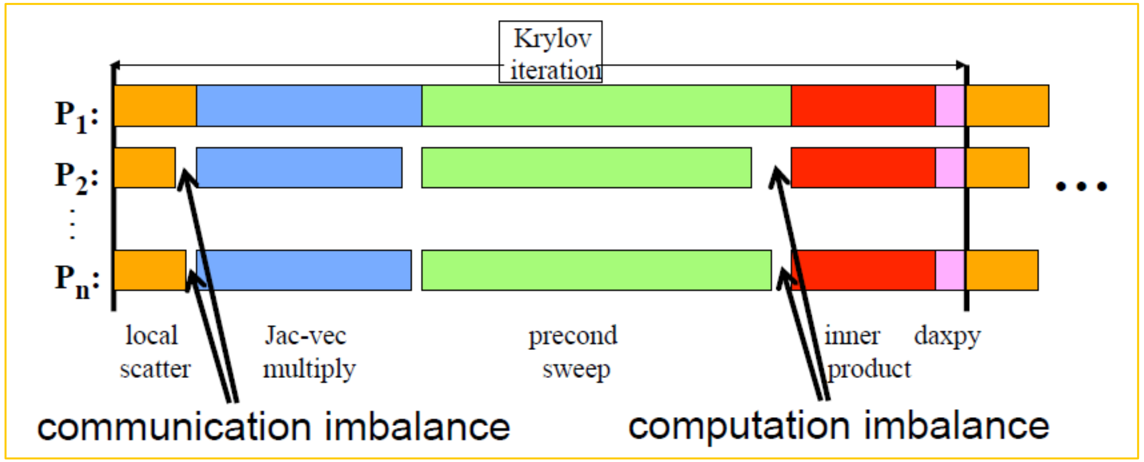
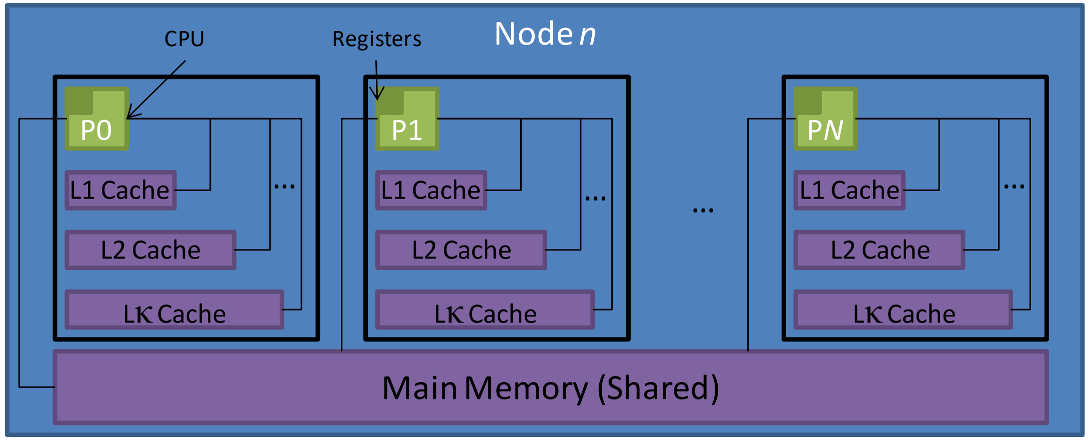


Figure from: D. Keyes, “Algorithmic Adaptations to Extreme Scale Computing”, ATPESC Workshop Presentation, (2013).

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)



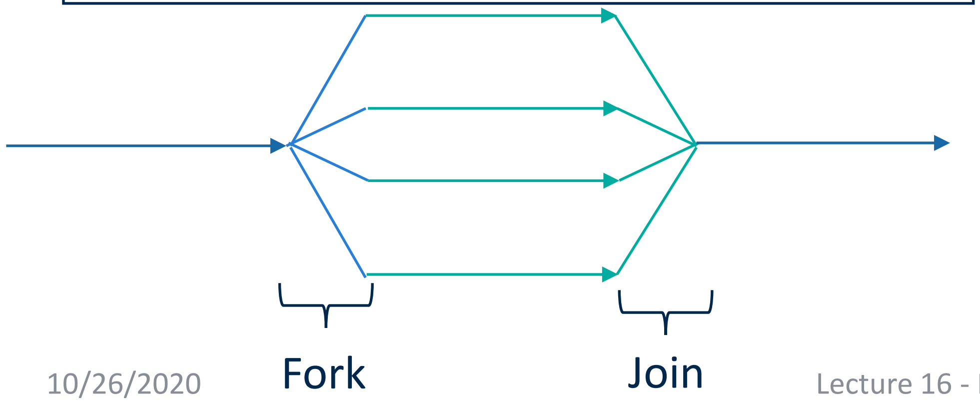
Typical Algorithms in Shared Memory Parallelism

Fork/Join

- Simple loop parallelization

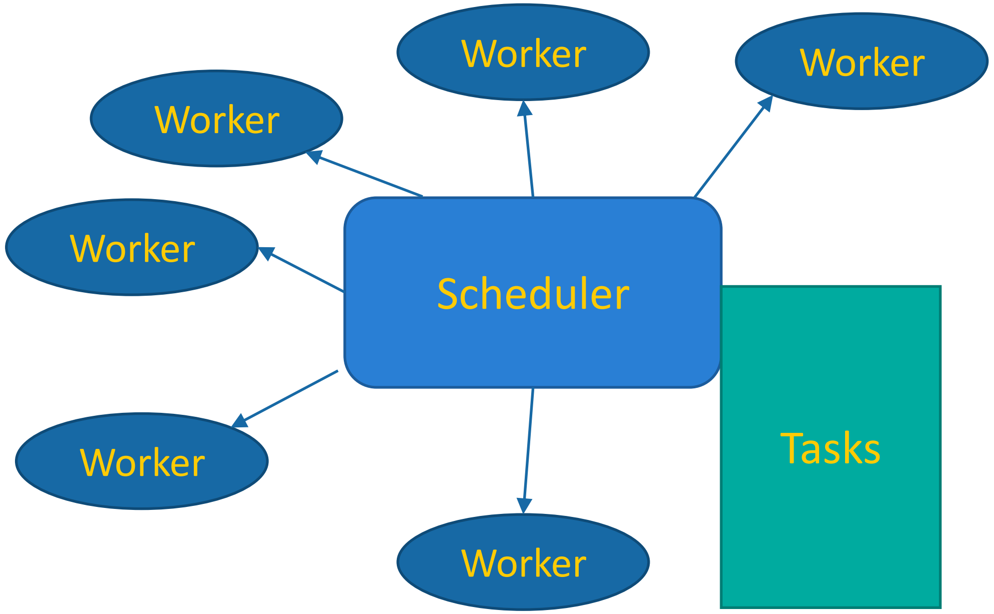
```
!Serial Section

DO PARALLEL i=1,n !implied fork
  !some operations for i
END DO PARALLEL !implied join
```



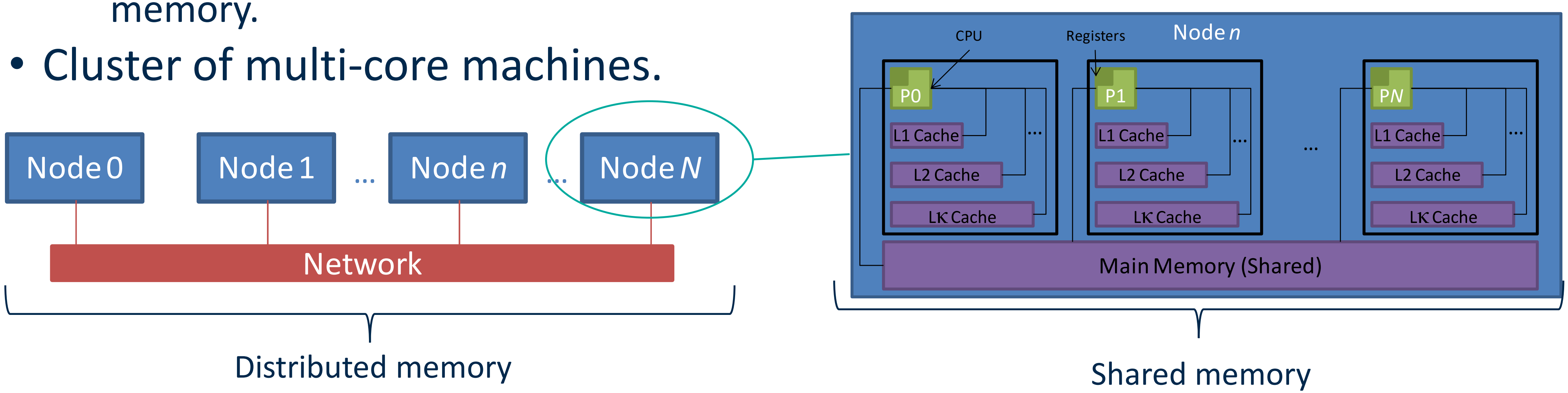
Pool of Tasks

- 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.

Parallel Algorithm "Ingredients"

Parallel Algorithm Ingredients

- What is the *programming model*? (distributed, shared, both)
  - If distributed, what is the communication model?
- What should the *granularity* of the parallelism be?
- How are you going to *decompose* the problem in parallel?
- How are you going *partition* the problem to obtain a balanced decomposition?
- Can all this be done once for a single simulation?
- What synchronizations are required?

Coarse Grained vs. Fine Grained

Coarse Grained

- Divide work into large tasks
  - Example: executing several functions
- Coarse grained parallelism usually has better strong scaling than fine-grained parallelism.
  - Although smaller limits to the maximum parallelism
- More susceptible to load imbalance.

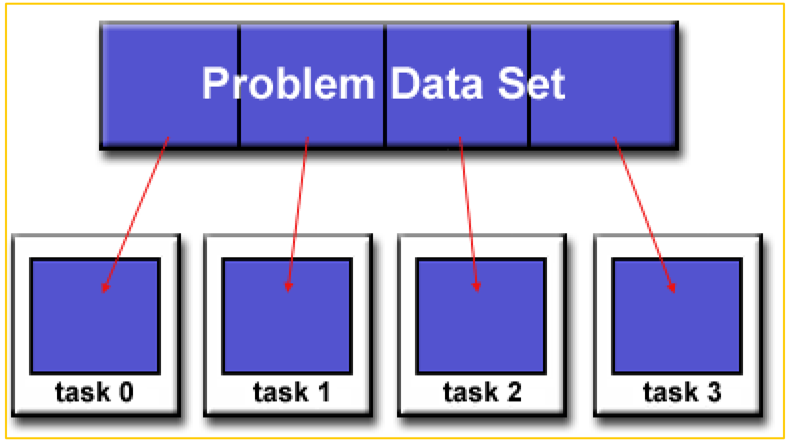
Fine Grained

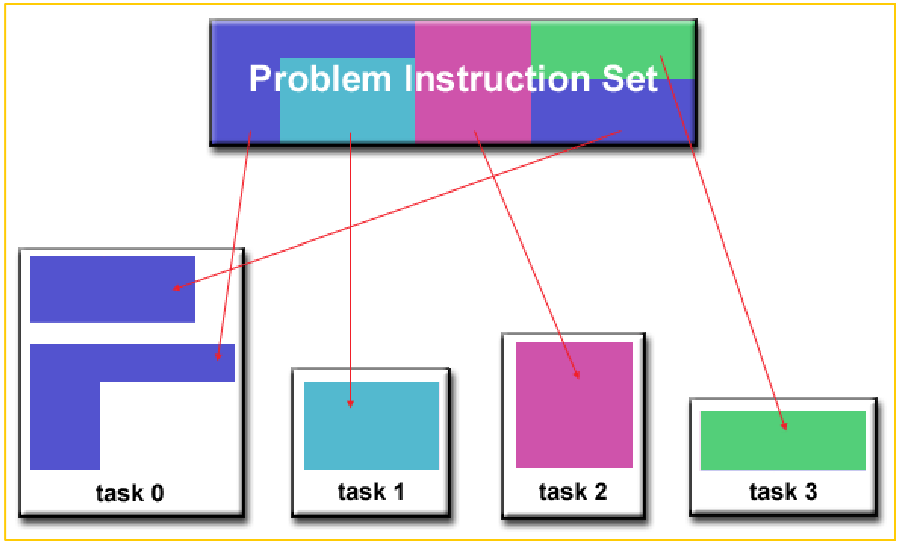
- Divide work into many small tasks
  - Example: iterations of a loop
- Usually has good load balance
- Difficult to hide overhead from parallelism
- Works well for things like SIMD & vector computing

Algorithm & Hardware will ultimately determine which is better. However, coarse-grained will usually be better

Decomposition

- What is being divided into parallel work?
- Most typical is domain decomposition
  - Divide up part of your equation “phase space”
    - Phase space = dependent variables of unknown (e.g. Cartesian space)
  - Slightly different is data decomposition
    - e.g. decompose a matrix in parallel
      - Matrix is usually a discretization of the phase space(s)
- Also have functional decomposition
  - Decompose by computation or operation
    - e.g. fluid on one process, solid on another for convective/conductive heat transfer

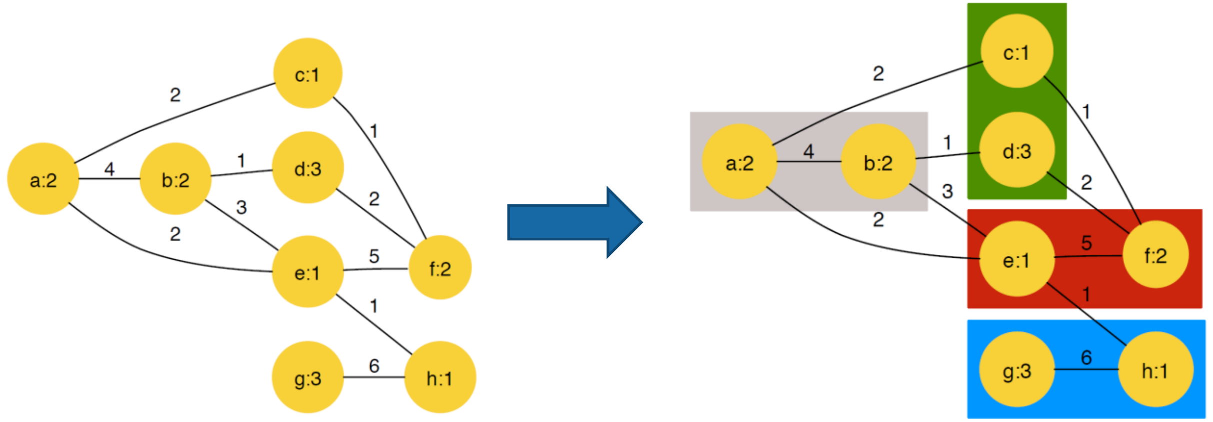




Figures from: https://computing.llnl.gov/tutorials/parallel_comp/

Partitioning

- How do you decompose the problem in parallel?
  - Example: Matrix partitioning
- In general this is a much harder problem.
  - Especially for the general case.
    - Involves a lot of graph theory



Figures from: R. Vuduc, "Graph Partitioning," Lecture in CSE/CS 8803, Georgia Institute of Technology, April 2008

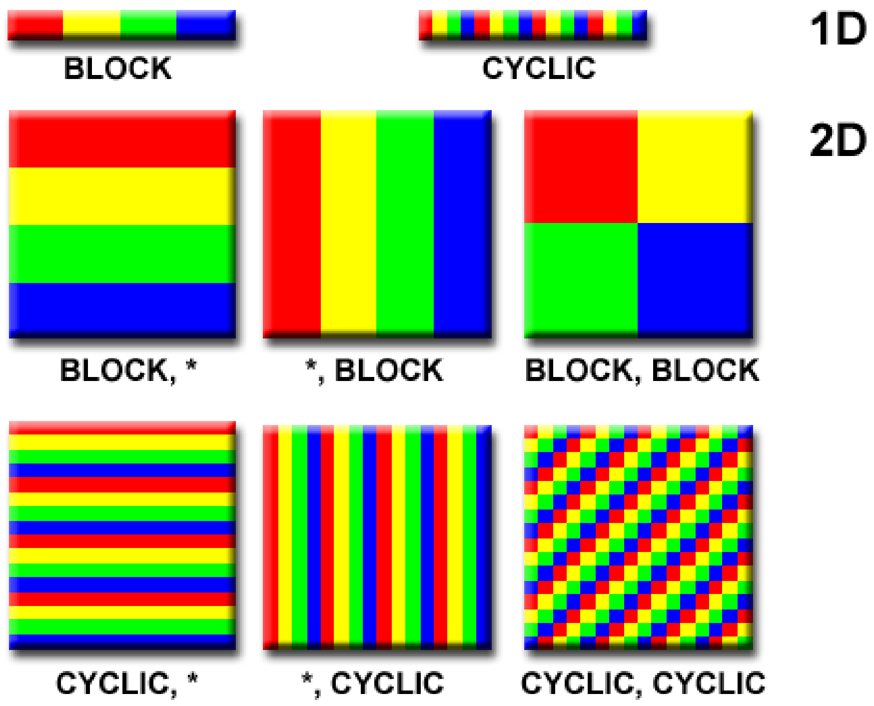


Figure from: https://computing.llnl.gov/tutorials/parallel_comp/

- Libraries exist to do this for us: METIS & ParMETIS

Dynamic vs. Static

Static

- Determine decomposition and partitioning once up-front prior to execution.
- Execute without changing number of processors or decomposition or partitioning
  - Fork/Join is not considered dynamic if the number of threads always the same
- More likely you will encounter this case

Dynamic

- Necessary to achieve better performance if computation load changes during run time.
- Change number of processors during run time.
- Change partitioning during run time.

Synchronization

- Generally, best to avoid as much as possible
  - In practice, never completely avoidable.
- In shared memory parallelism this includes the fork and join operations.
- Synchronization usually occurs whenever you encounter an integral.
  - More generally it occurs with "reduction" operations.
  - In a reduction operation you reduce parallel data to a single process
    - E.g. computing a sum, finding a max, computing a product, logical operators
- In distributed memory parallelism (more specifically MPI), it is any collective operation (not just reduce)
- Critically important to be aware of collective operations

Illustrations of collective operations

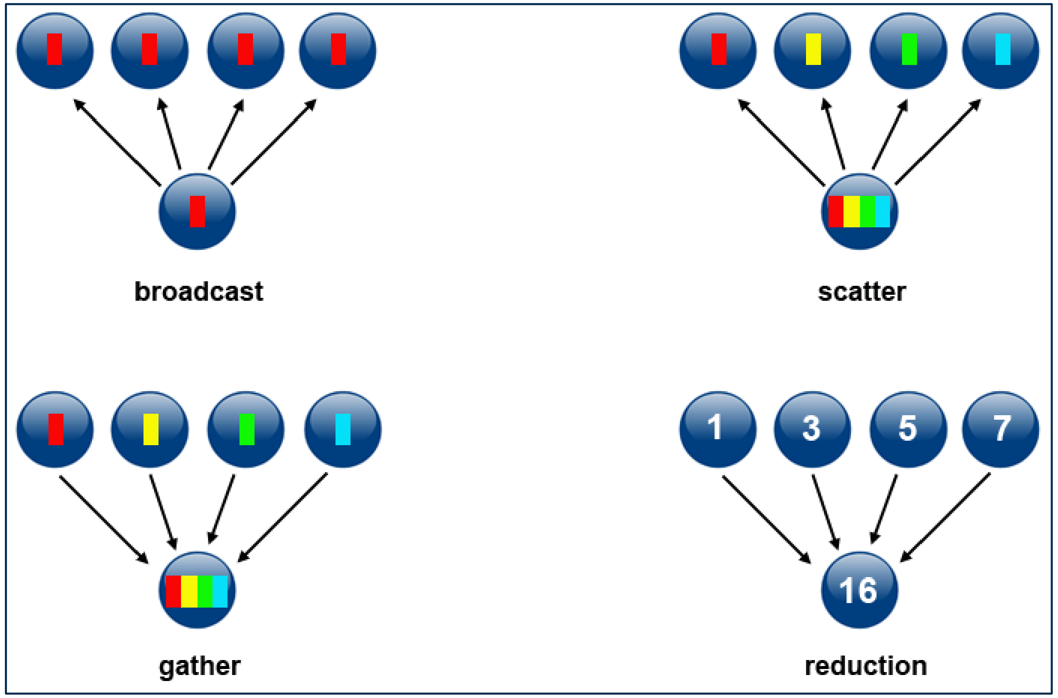
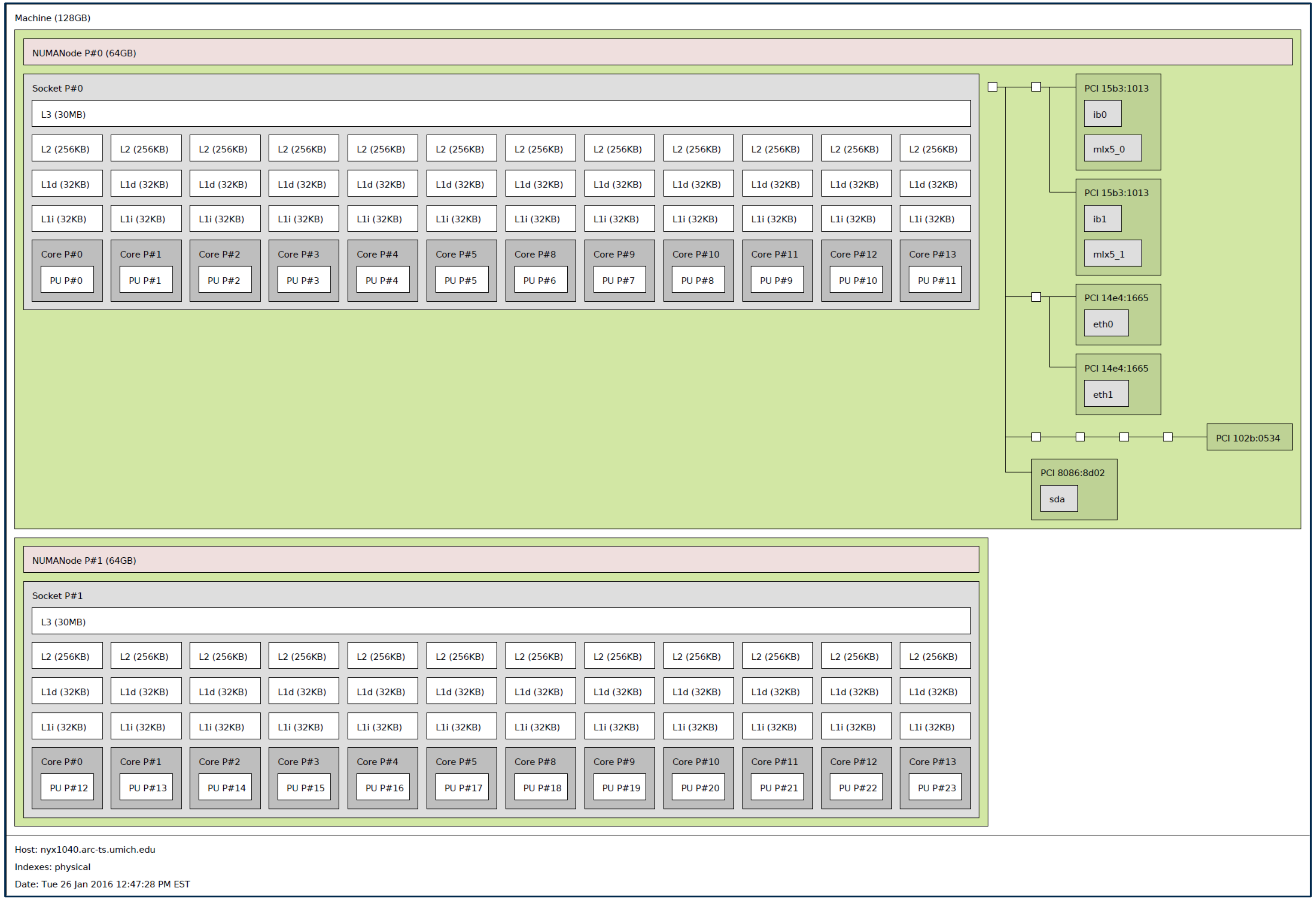


Figure from: https://computing.llnl.gov/tutorials/parallel_comp/

Shared Memory Execution Model

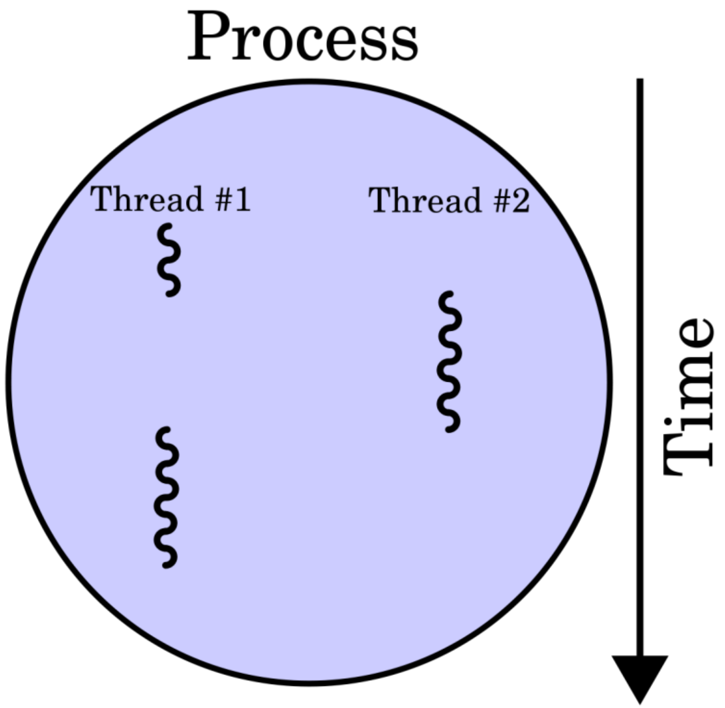
Flux node Architecture

(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

Programming Shared Memory Parallelism

- This is what we'll cover Wednesday in Lecture 17.