Parallel Architecture and Execution

Lecture 2

January 7, 2026

Parallel Application Performance

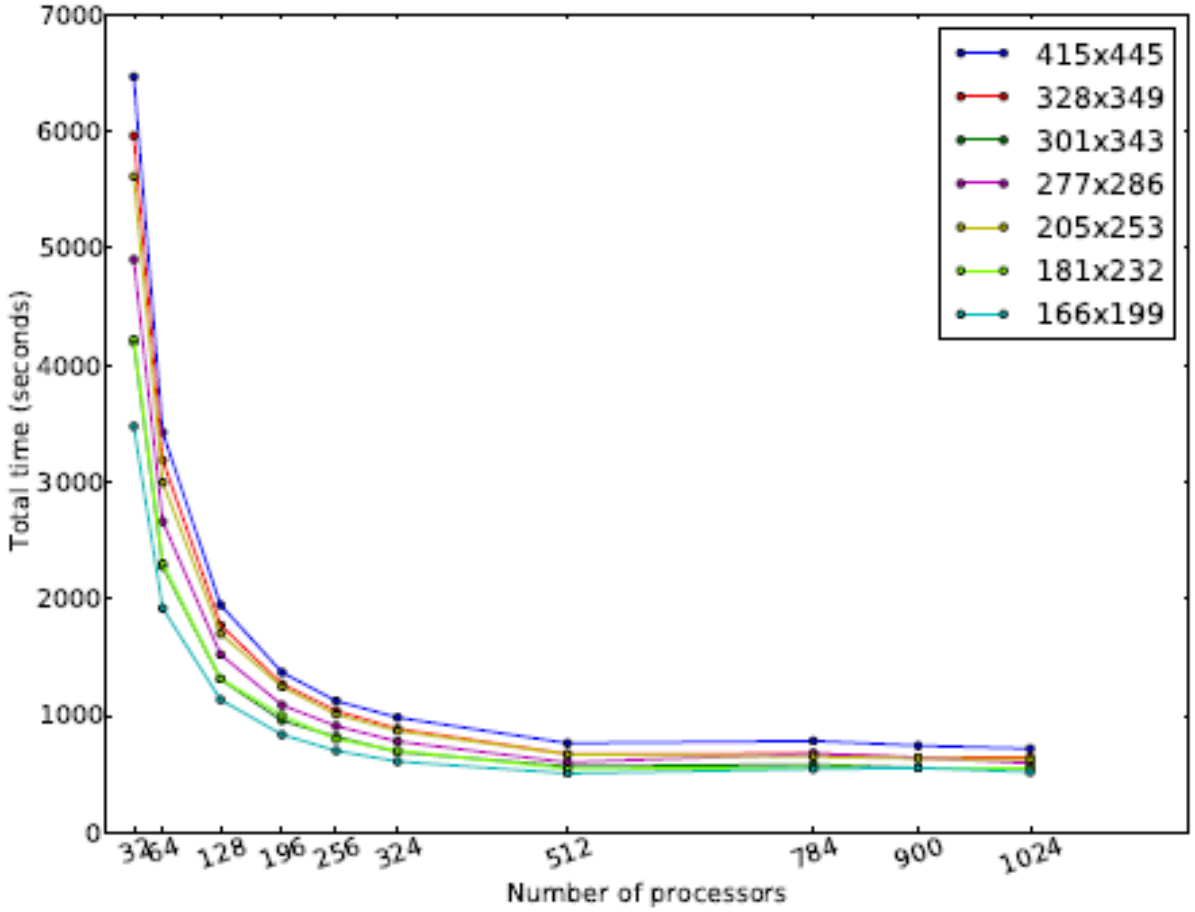
Depends on

- Parallel architecture
- Algorithm

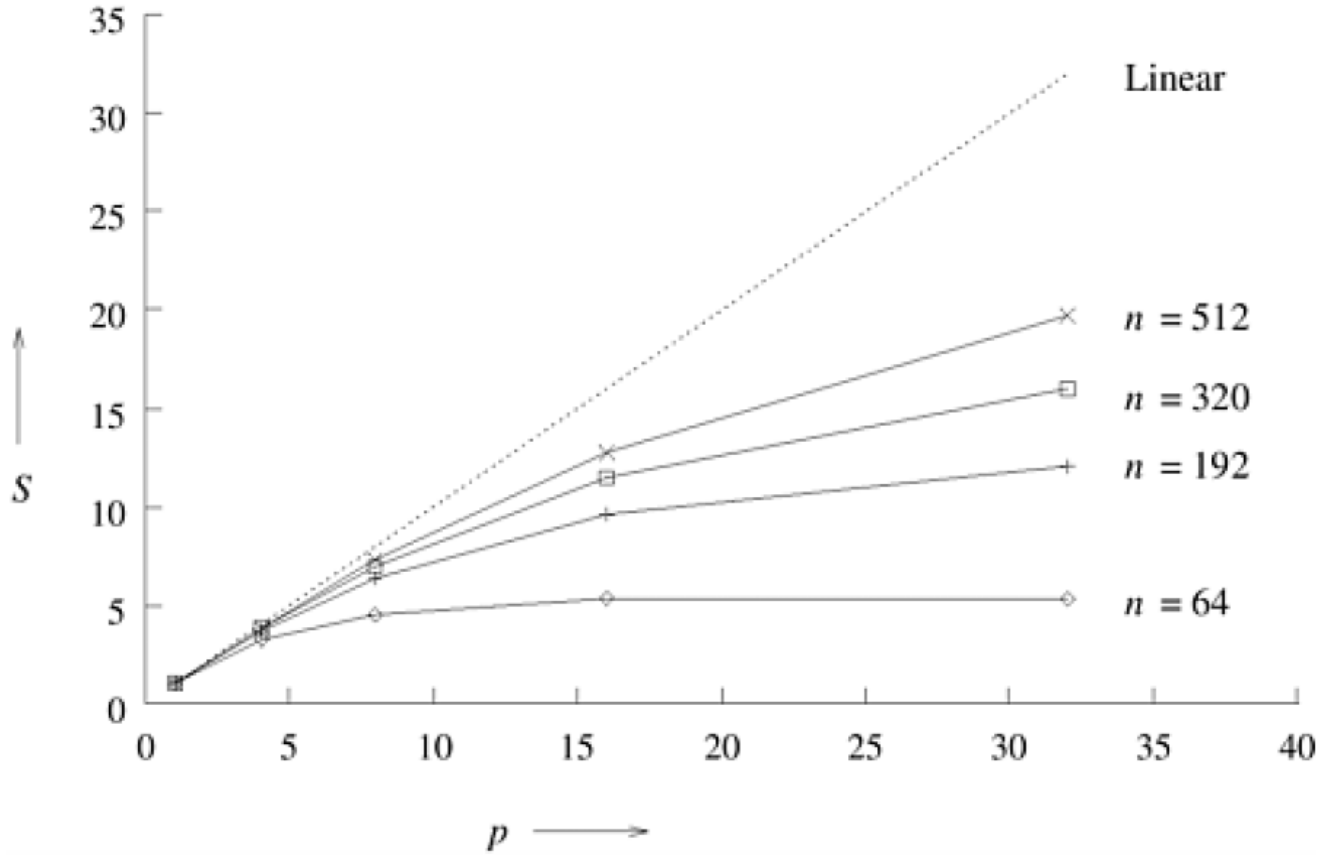
Saturation – Example 1

#Processes	Time (sec)	Speedup	Efficiency
1	0.025	1	1.00
2	0.013	1.9	0.95
4	0.010	2.5	0.63
8	0.009	2.8	0.35
12	0.007	3.6	0.30

Saturation – Example 2



Saturation – Example 3



Source: GGKK Chapter 5

Efficiency (Adding numbers)

Problem size

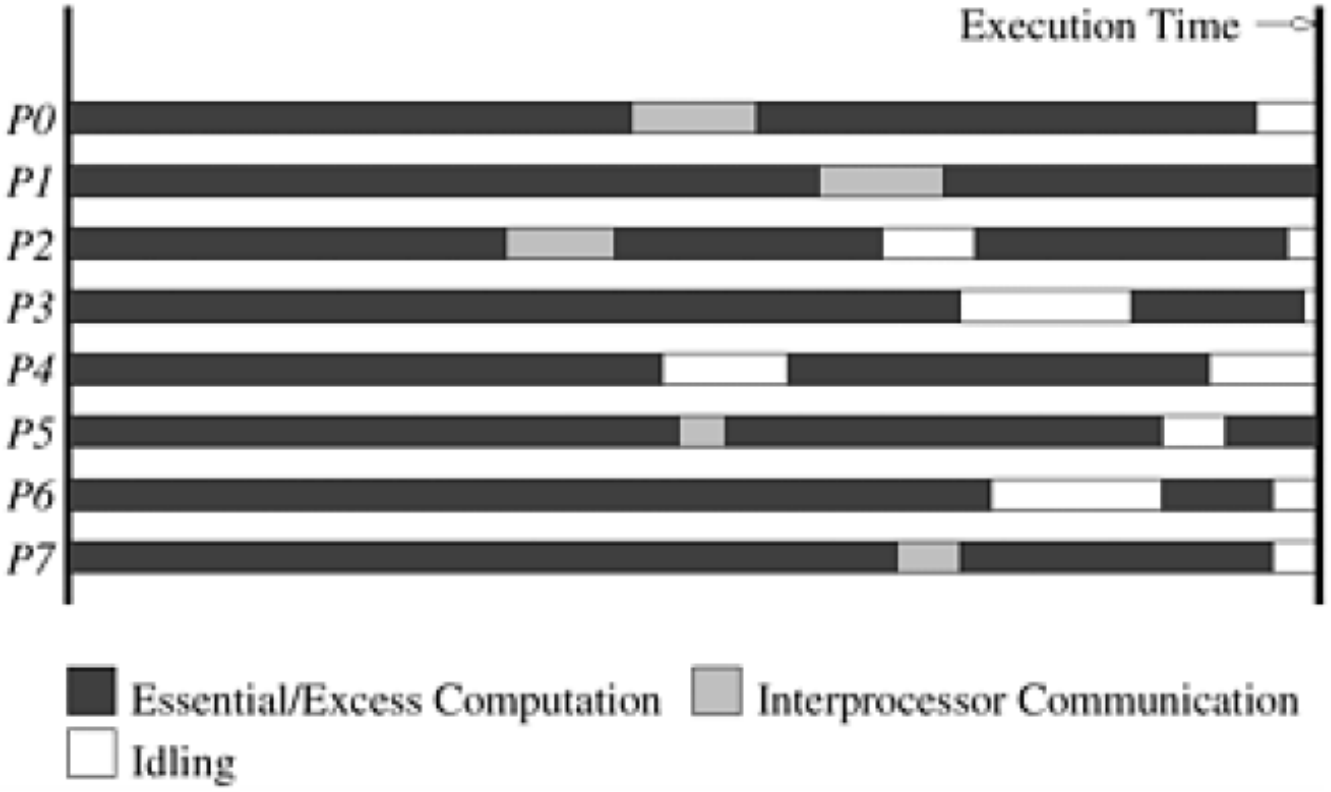
n	*p* = 1	*p* = 4	*p* = 8	*p* = 16	*p* = 32
64	1.0	***0.80***	0.57	0.33	0.17
192	1.0	0.92	***0.80***	0.60	0.38
320	1.0	0.95	0.87	0.71	0.50
512	1.0	0.97	0.91	***0.80***	0.62

Source: GGKK Chapter 5

Limitations of Parallelization

- Overhead
  - E.g. communication
- Over-decomposition
  - Work per process/core
- Idling
  - Load imbalance
  - Synchronization
  - Serialization

Execution Profile



Execution Profile of a Hypothetical Parallel Program

Source: GGKK Chapter 5

Performance

- Sequential
  - Input size
- Parallel
  - Input size
  - Number of processing elements (PE)
  - Communication speed
  - …

Scaling Deep Learning Models

Problem size

#PEs

Solver	batch size	steps	F1 score on dev set	TPUs	Time
Baseline	512	1000k	90.395	16	81.4h
LAMB	512	1000k	91.752	16	82.8h
LAMB	1k	500k	91.761	32	43.2h
LAMB	2k	250k	91.946	64	21.4h
LAMB	4k	125k	91.137	128	693.6m
LAMB	8k	62500	91.263	256	390.5m
LAMB	16k	31250	91.345	512	200.0m
LAMB	32k	15625	91.475	1024	101.2m

SOURCE: LARGE BATCH OPTIMIZATION FOR DEEP LEARNING: TRAINING BERT IN 76 MINUTES

"The fastest known sequential algorithm for a problem may be difficult or impossible to parallelize."- GGKK

Parallelization

- Speedup
- Efficiency

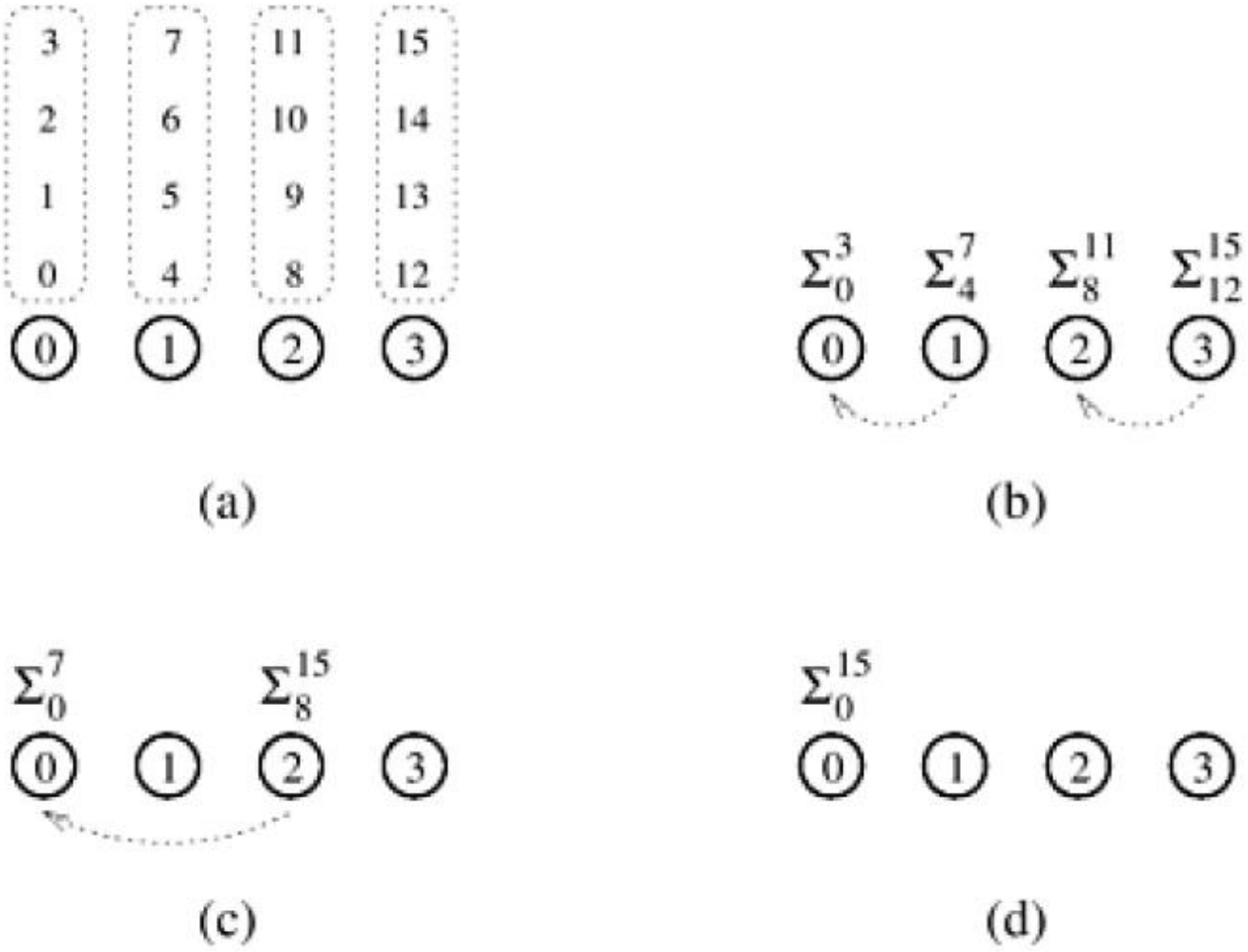
Sum of Numbers Speedup

$$\text{Speedup} = \frac{N}{\frac{N}{P} + P + P}$$

Naïve parallelization method:

- Compute in parallel
- Send partial result to one (All-to-one)
- Compute final result

Parallel Sum (Optimized)



Source: GGKK Chapter 5

Sum of Numbers Speedup (Optimized)

$S_1 = \frac{N}{\frac{N}{P} + 2P}$ $E_1 = ?$

$S_2 = \frac{N}{\frac{N}{P} + 2\,logP}$ $E_2 = \frac{1}{\frac{2PlogP}{N} + 1}$

Efficiency (Adding numbers)

Homework: Analyze the measured efficiency based on your derivation

n	*p* = 1	*p* = 4	*p* = 8	*p* = 16	*p* = 32
64	1.0	***0.80***	0.57	0.33	0.17
192	1.0	0.92	***0.80***	0.60	0.38
320	1.0	0.95	0.87	0.71	0.50
512	1.0	0.97	0.91	***0.80***	0.62

Source: GGKK Chapter 5

A Limitation of Parallel Computing

Fraction of code that is parallelizable

$$\text{Speedup } S = \frac{1}{(1-f) + f/P}$$

Amdahl's Law

Parallel Architecture

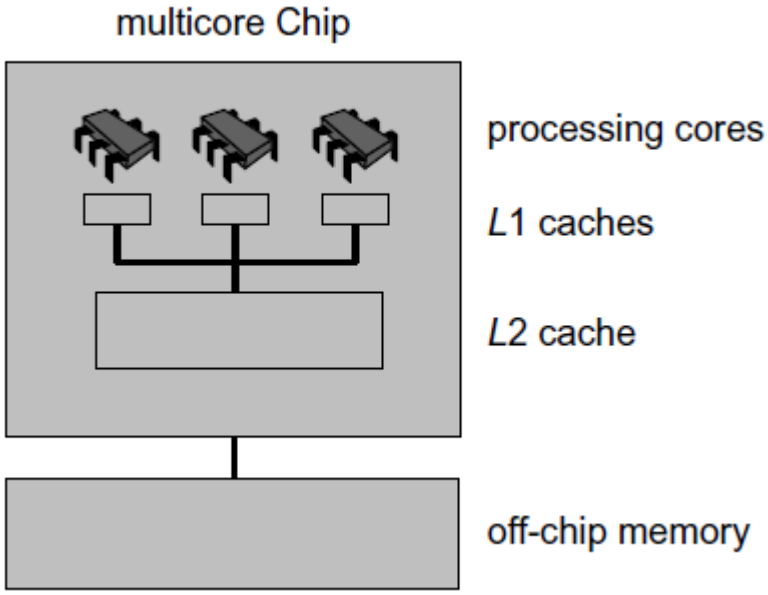
System Components

- Processor
- Memory
- Network
- Storage

NUMA

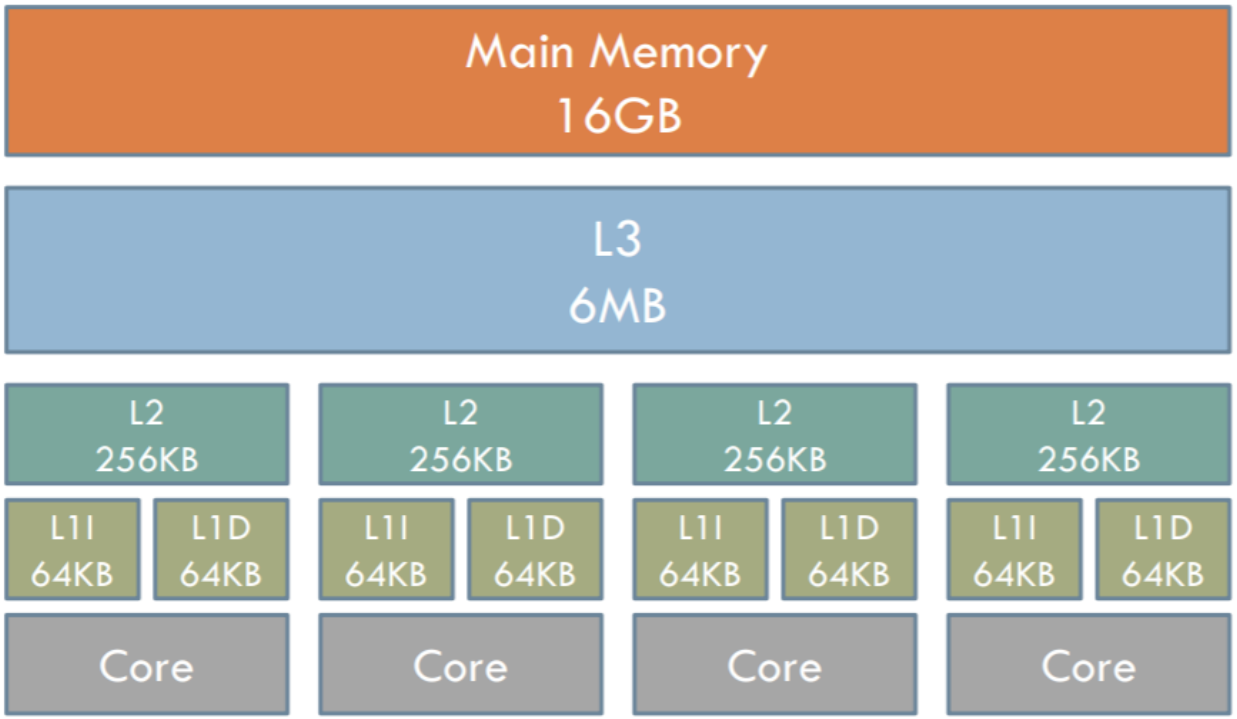
Source: https://www.sciencedirect.com/topics/computer-science/non-uniform-memory-access

Memory Hierarchy



A multicore SMP architecture
Image Source: The Art of Multiprocessor Programming – Herlihy, Shavit

Memory Access Times



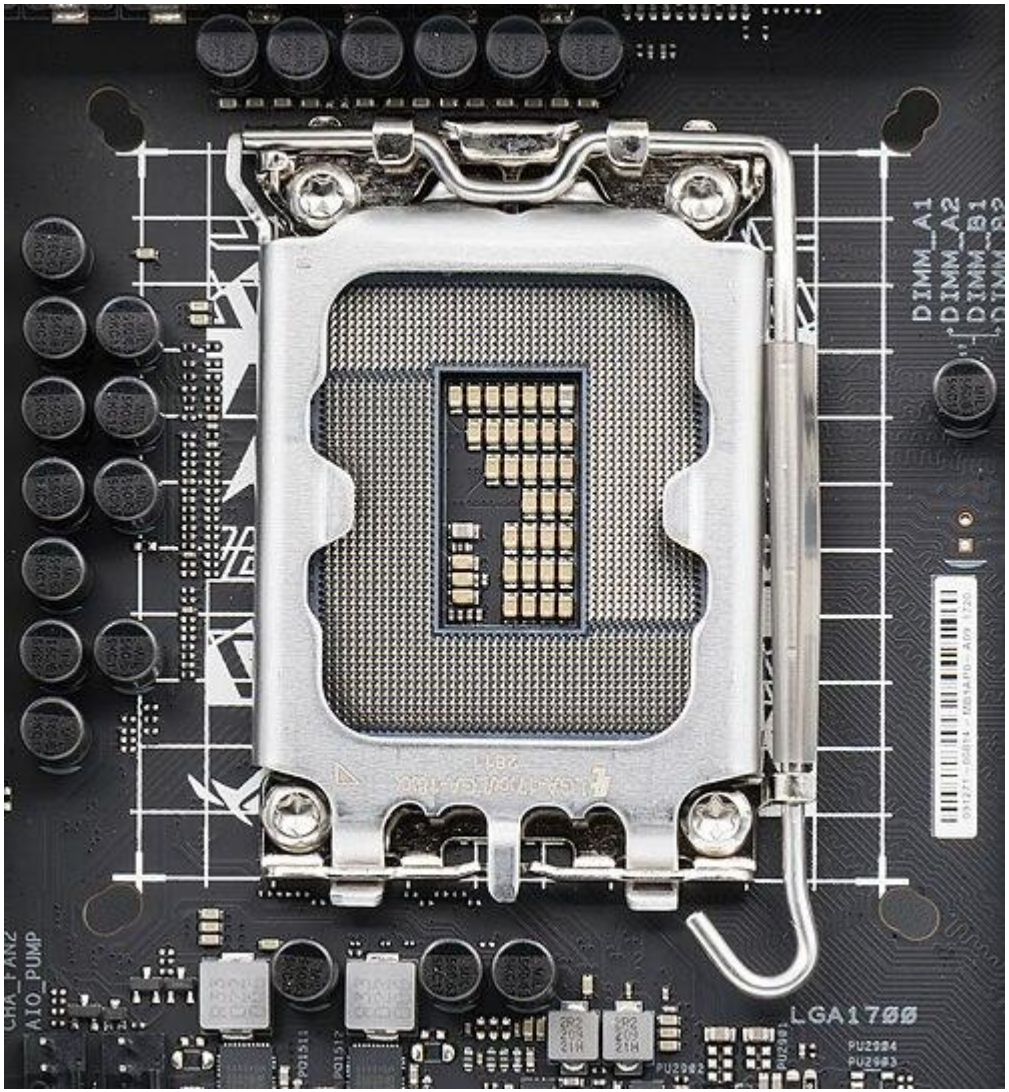
150 cycles, 21GB/s

26-31 cycles, 96GB/s
(24GB/s/core)

8 cycles, 48GB/s

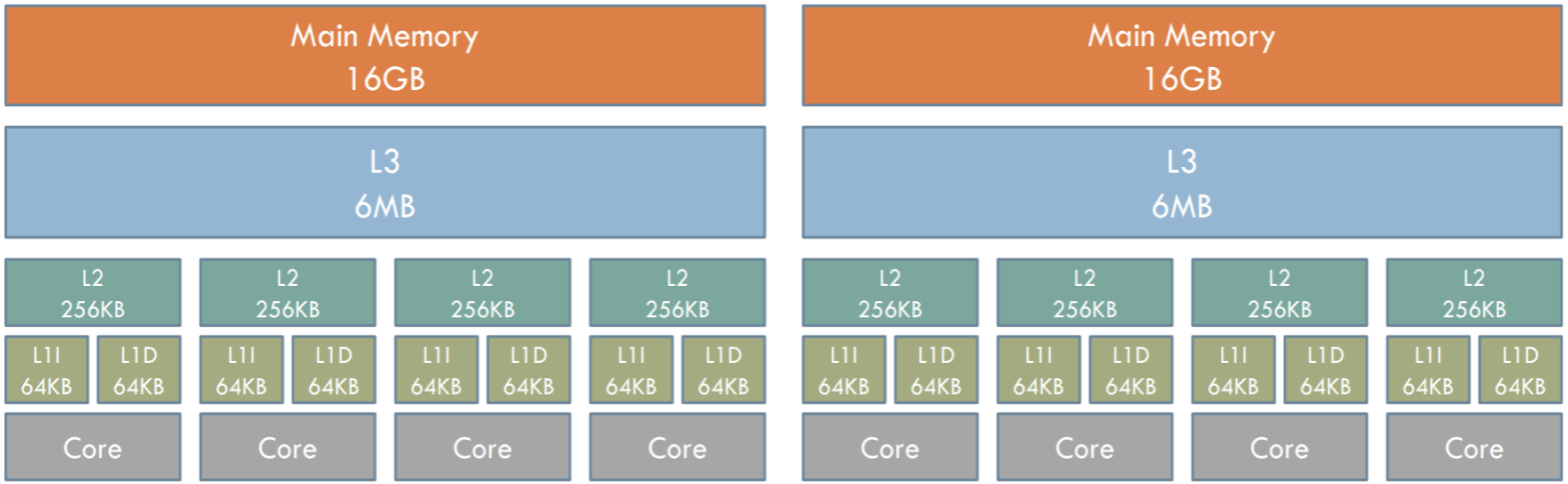
4 cycles, 72GB/s

Source: MIT CSAIL



Intel LGA1851 Socket (Wikipedia)

Effective Memory Access Times



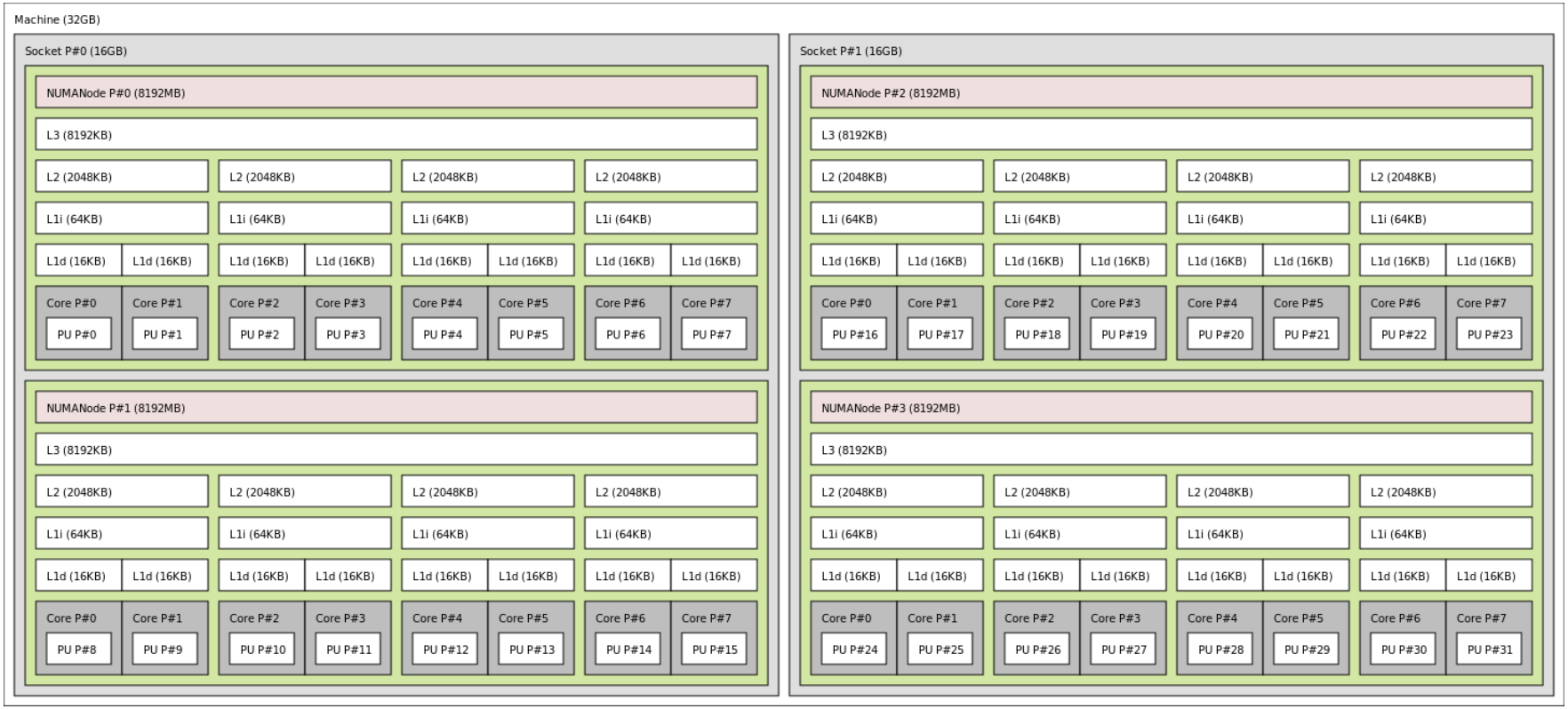
Processor vs. Memory

"While clock rates of high-end processors have increased at roughly 40% per year over the last decade, DRAM access times have only improved at the rate of roughly 10% per year over this interval."

- Introduction to Parallel Computing by Ananth Grama et al. (GGKK)

NUMA Nodes

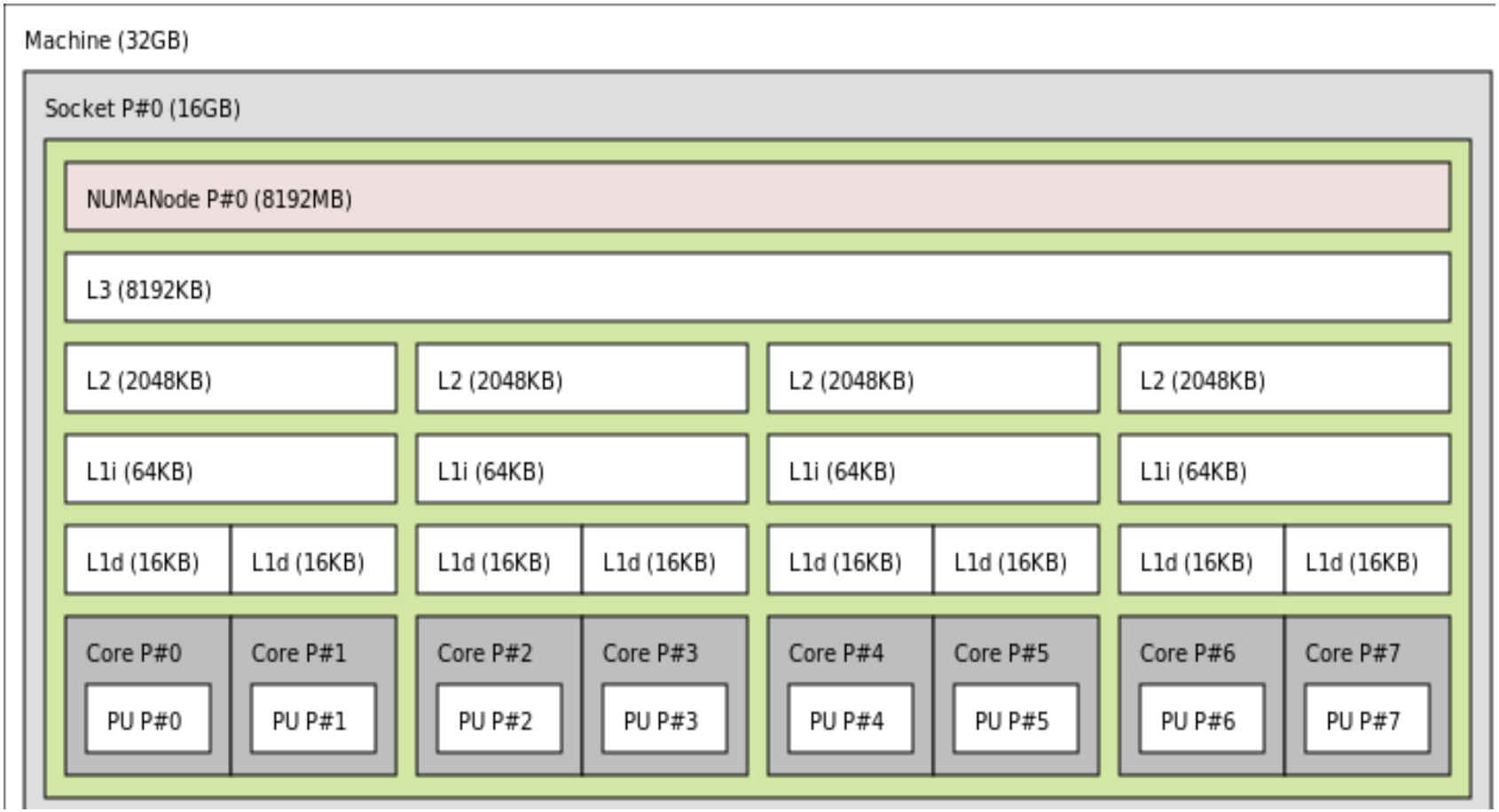
Utility: **lstopo** (hwloc package)



AMD Bulldozer Memory Topology (Source: Wikipedia)

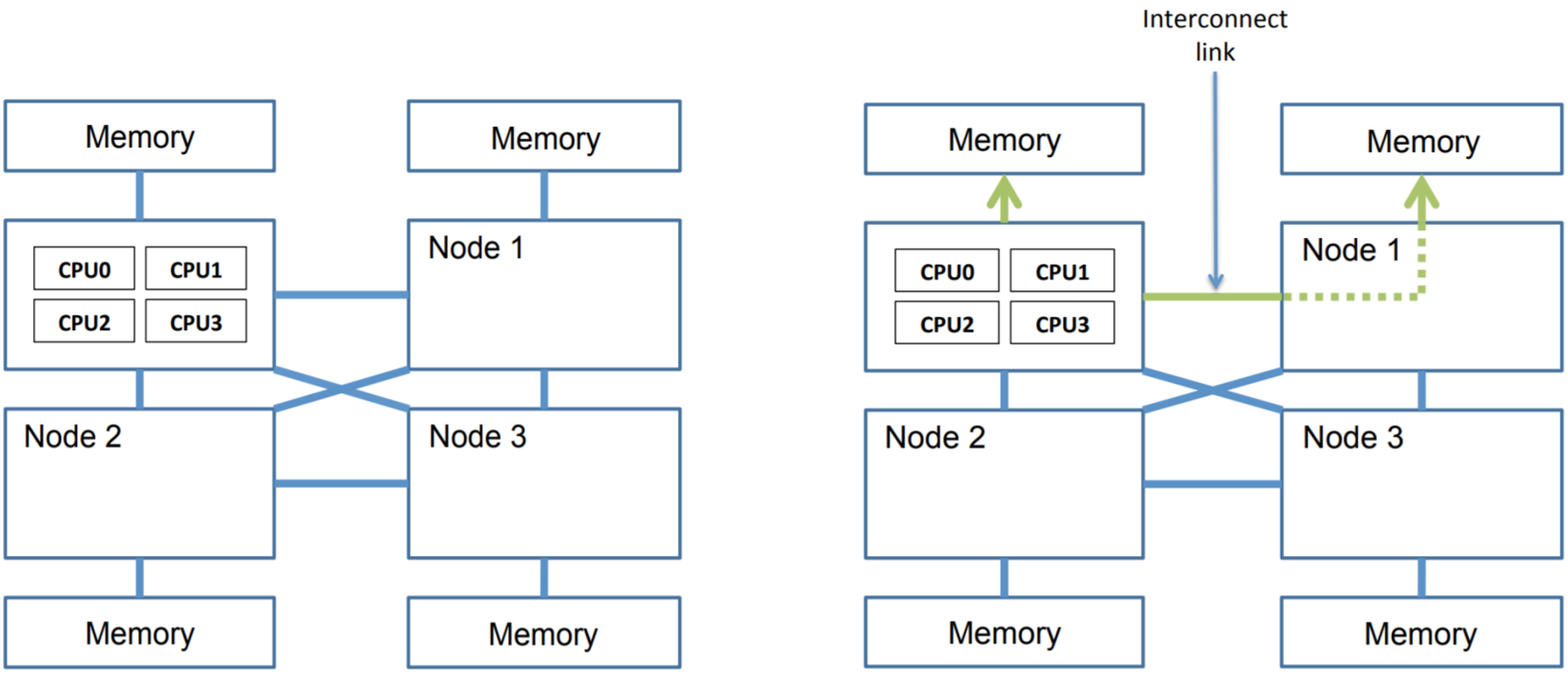
NUMA Node (Zoomed)

Utility: lstopo (hwloc package)



AMD Bulldozer Memory Topology (Source: Wikipedia)

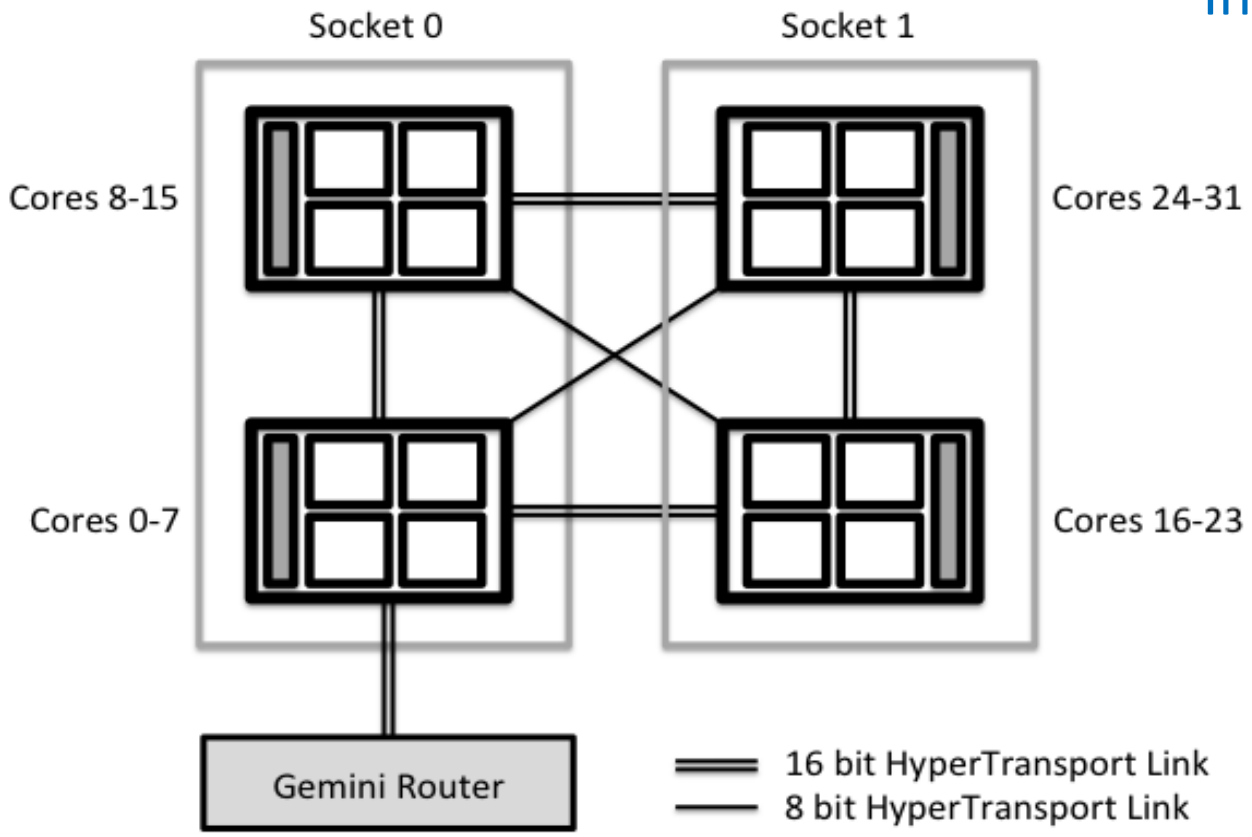
Memory Placement



Lepers et al., “Thread and Memory Placement on NUMA Systems: Asymmetry Matters”, USENIX ATC 2015.

Connect Multiple Compute Nodes

Intraconnect



Source: hector.ac.uk

Parallel Programming Models

- Shared memory
- Distributed memory

Shared Memory

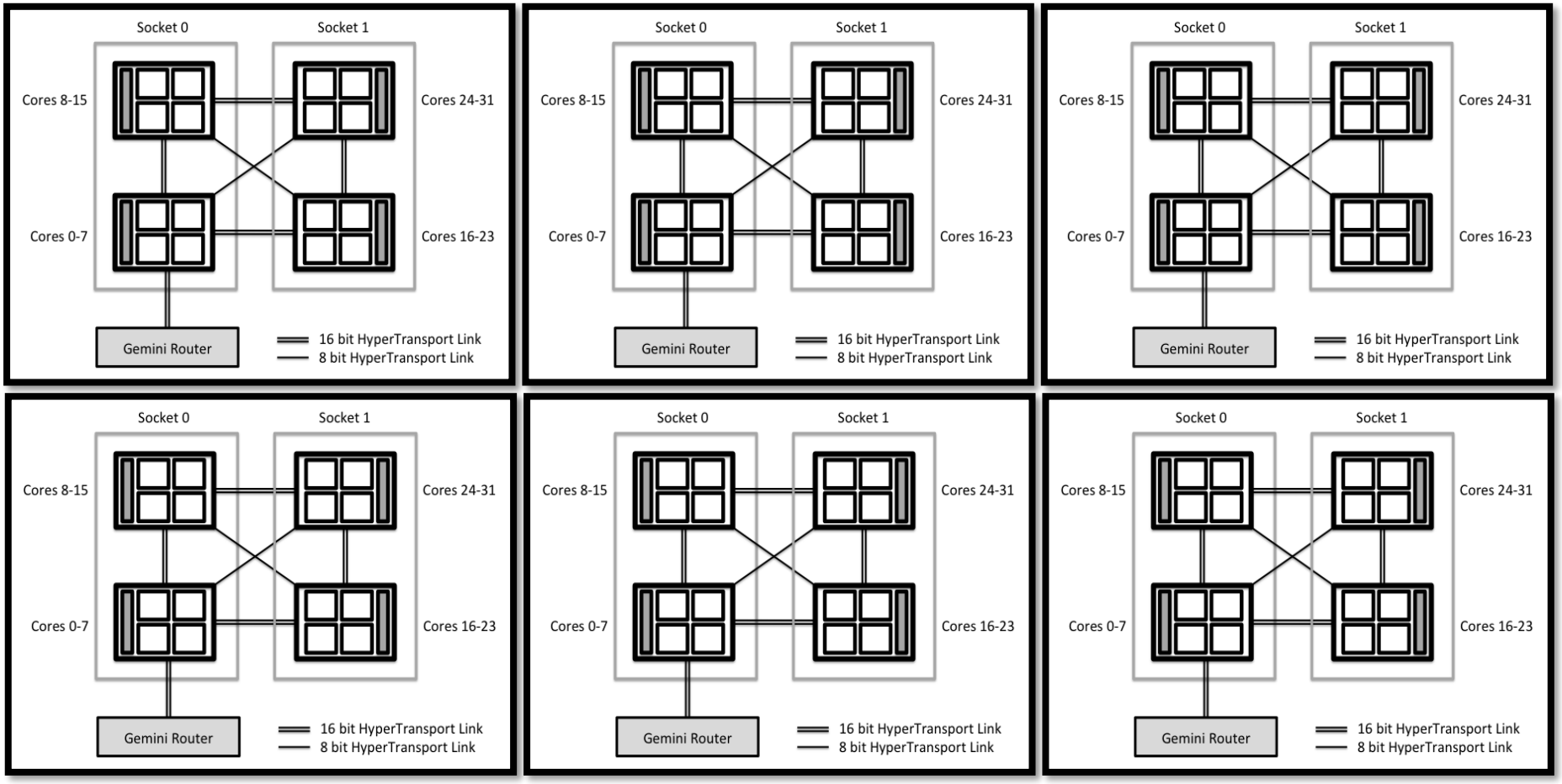
- Shared address space
- Time taken to access certain memory words is longer (NUMA)
- Need to worry about concurrent access
- Programming paradigms – Pthreads, OpenMP

Intel Processors (Latest)

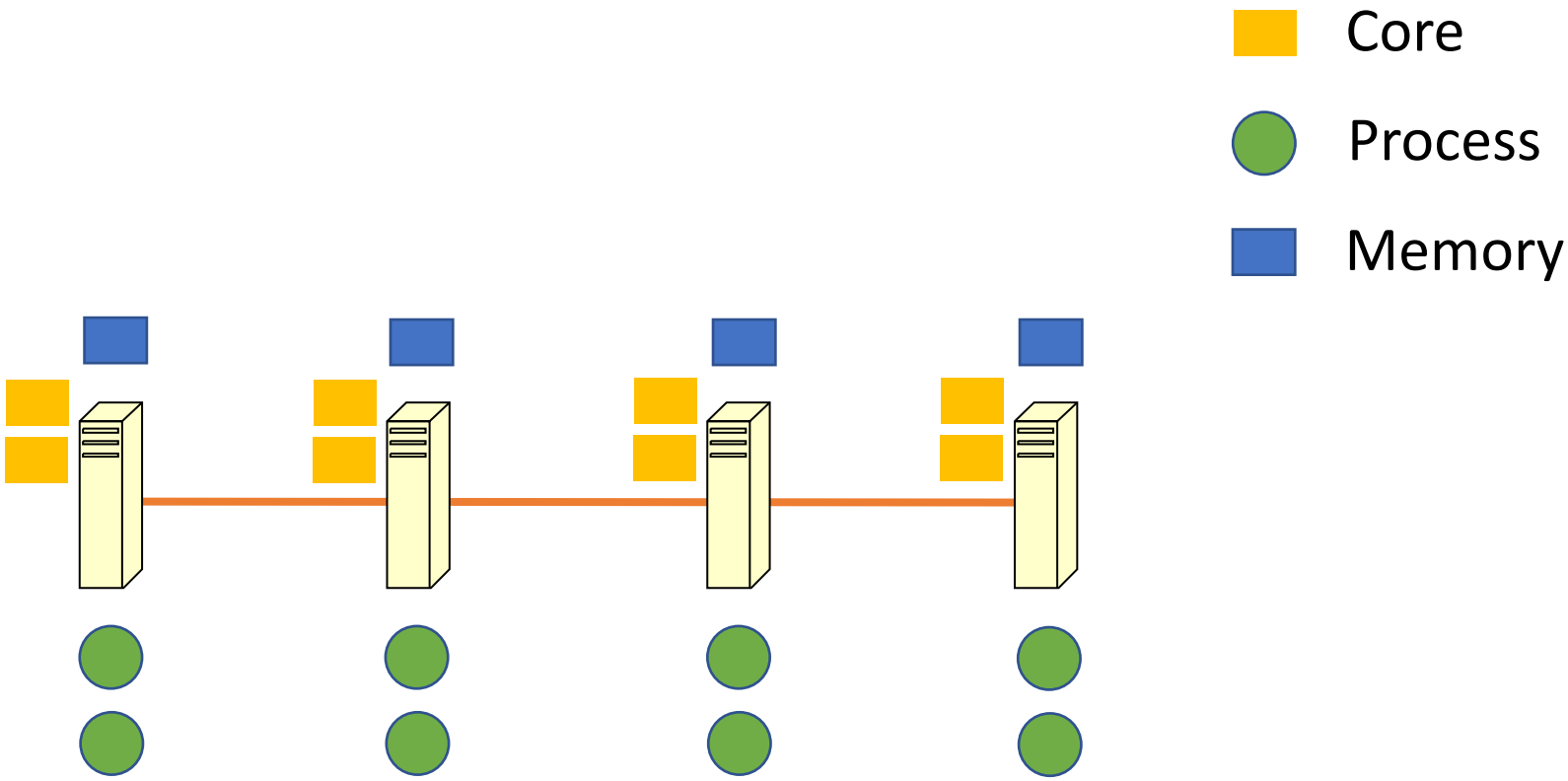
Product Name	Launch Date	Total Cores	Max Turbo Frequency	Processor Base Frequency	Cache
Intel® Xeon® Silver 4510T Processor (30M Cache, 2.00 GHz)	Q4'23	12	3.7 GHz	2.00 GHz	30 MB
Intel® Xeon® Platinum 8571N Processor (300M Cache, 2.40 GHz)	Q4'23	52	4 GHz	2.4 GHz	300 MB
Intel® Xeon® Gold 6530 Processor (160M Cache, 2.10 GHz)	Q4'23	32	4 GHz	2.1 GHz	160 MB
Intel® Xeon® Platinum 8593Q Processor (320M Cache, 2.20 GHz)	Q4'23	64	3.9 GHz	2.2 GHz	320 MB
Intel® Xeon® Platinum 8558 Processor (260M Cache, 2.10 GHz)	Q4'23	48	4 GHz	2.1 GHz	260 MB
Intel® Xeon® Platinum 8558P Processor (260M Cache, 2.70 GHz)	Q4'23	48	4 GHz	2.7 GHz	260 MB
Intel® Xeon® Platinum 8592+ Processor (320M Cache, 1.90 GHz)	Q4'23	64	3.9 GHz	1.9 GHz	320 MB
Intel® Xeon® Gold 6554S Processor (180M Cache, 2.20 GHz)	Q4'23	36	4 GHz	2.2 GHz	180 MB
Intel® Xeon® Platinum 8558U Processor (260M Cache, 2.00 GHz)	Q4'23	48	4 GHz	2 GHz	260 MB

https://www.intel.com/content/www/us/en/products/docs/processors/xeon/5th-gen-xeon-scalable-processors.html

Cluster of Compute Nodes



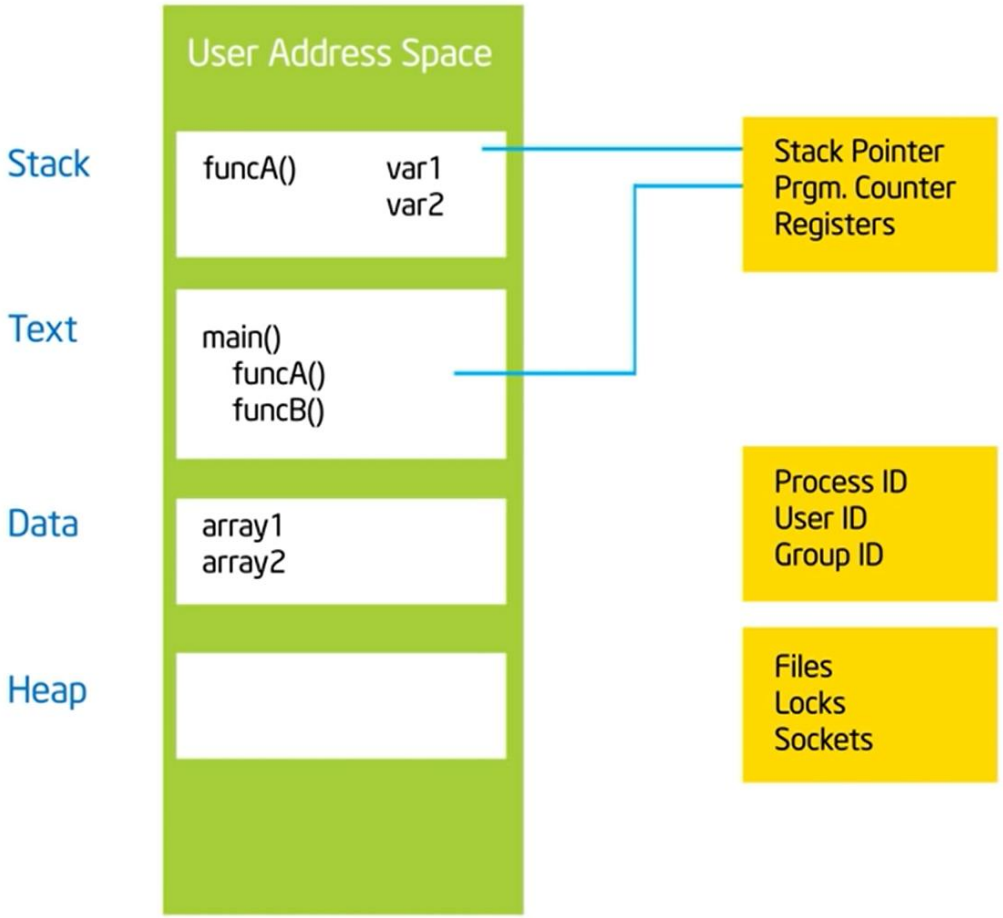
Our Parallel World



Distributed memory programming

- Distinct address space
- Explicit communication

Process



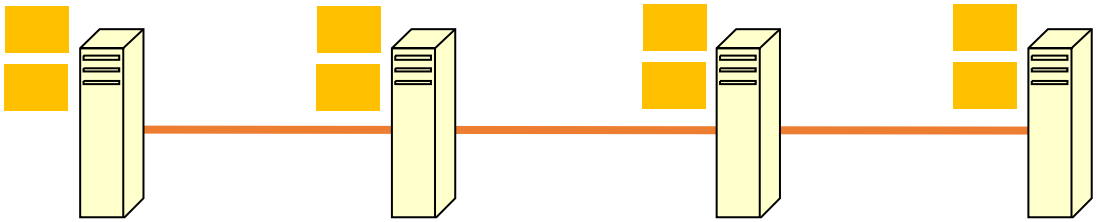
Process:

* An instance of a program execution.
* The execution context of a running program... i.e. the resources associated with a program's execution.

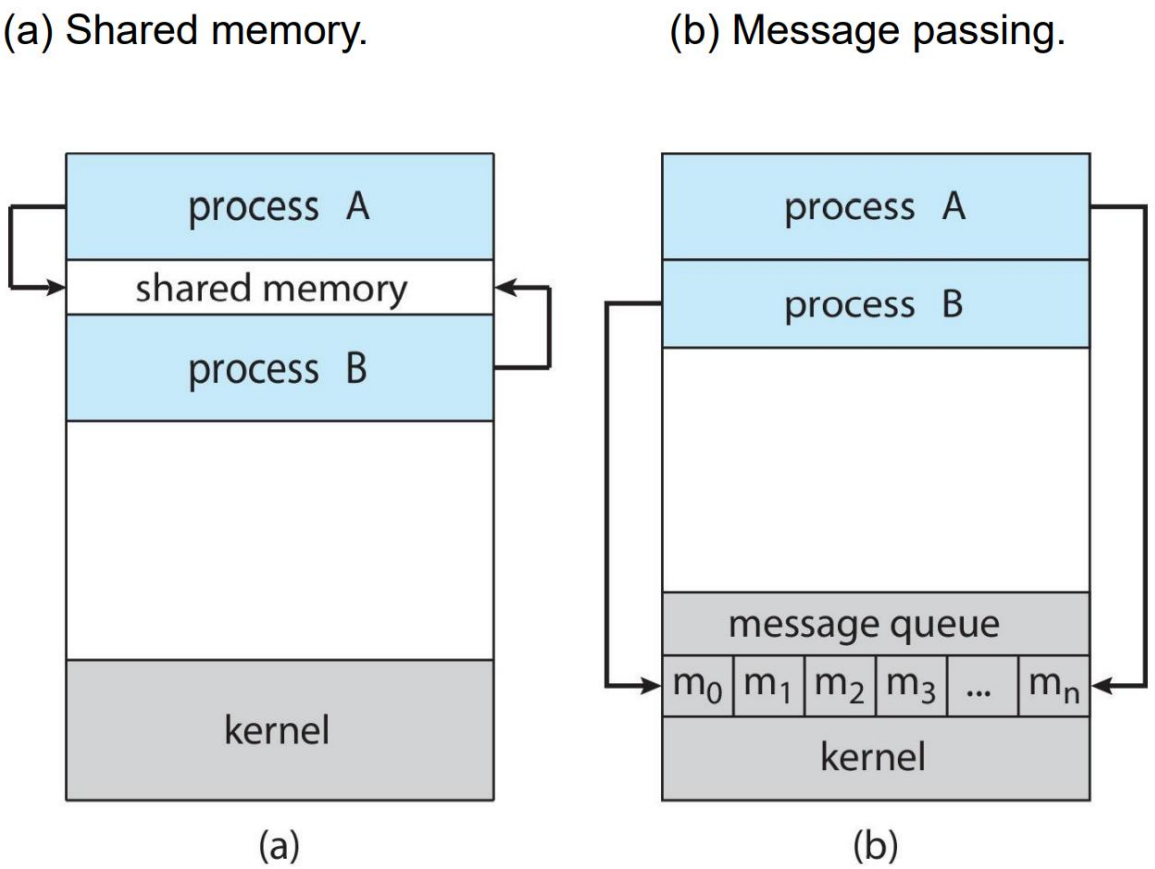
Adapted from Tim Mattson's slides

Message Passing

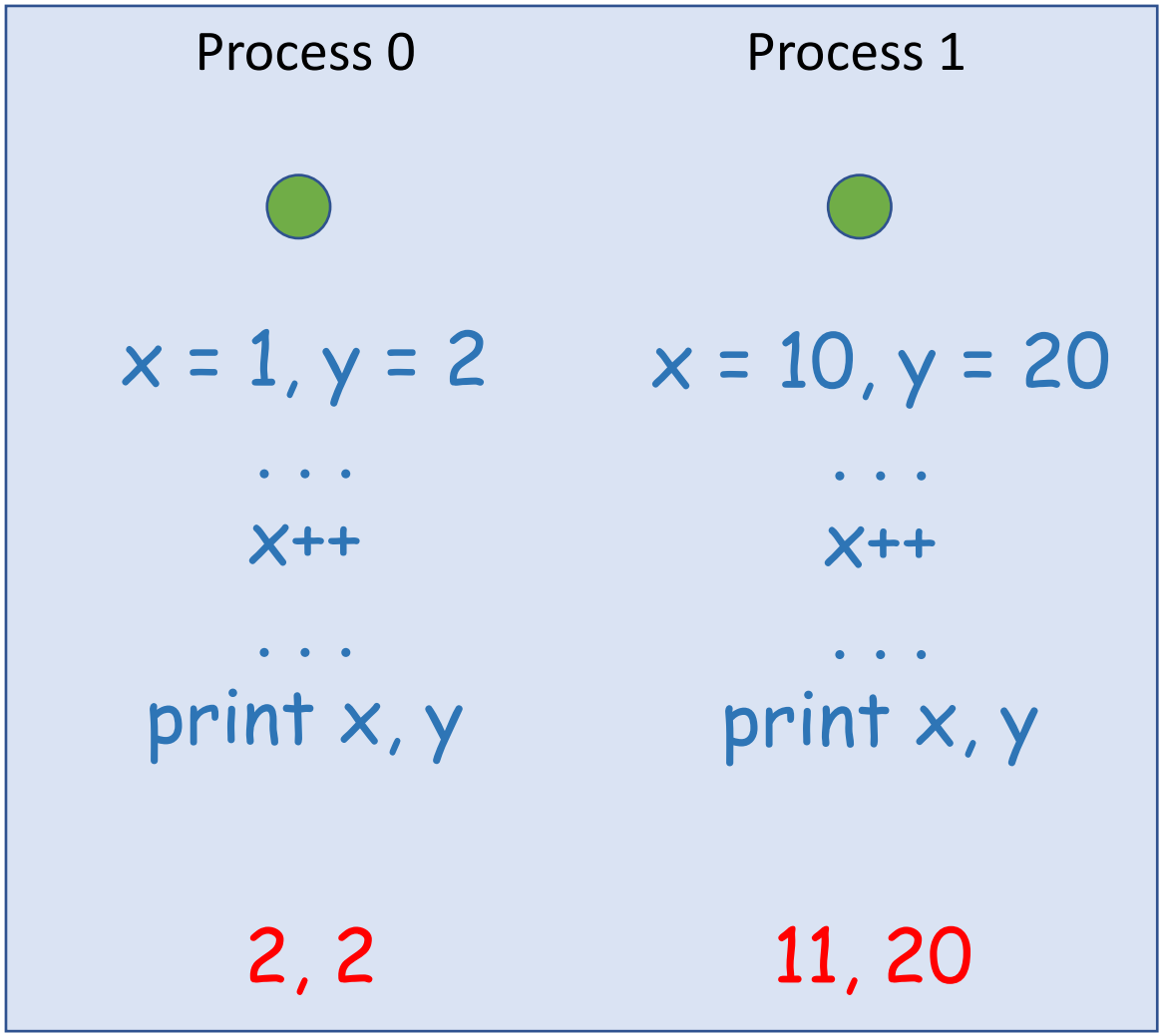
- Distinct address space per process
- Multiple processing nodes
- Basic operations are send and receive



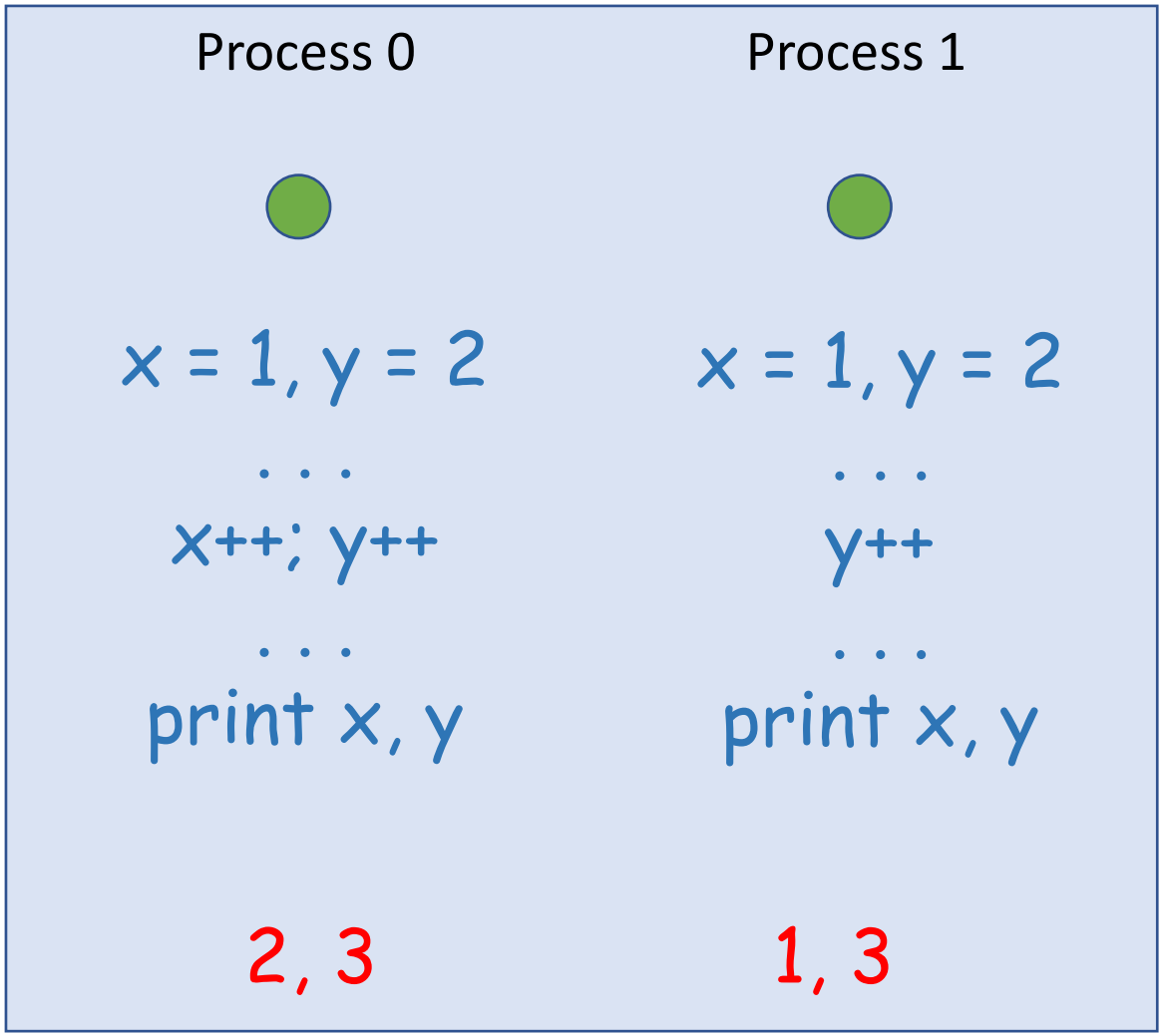
Interprocess Communication

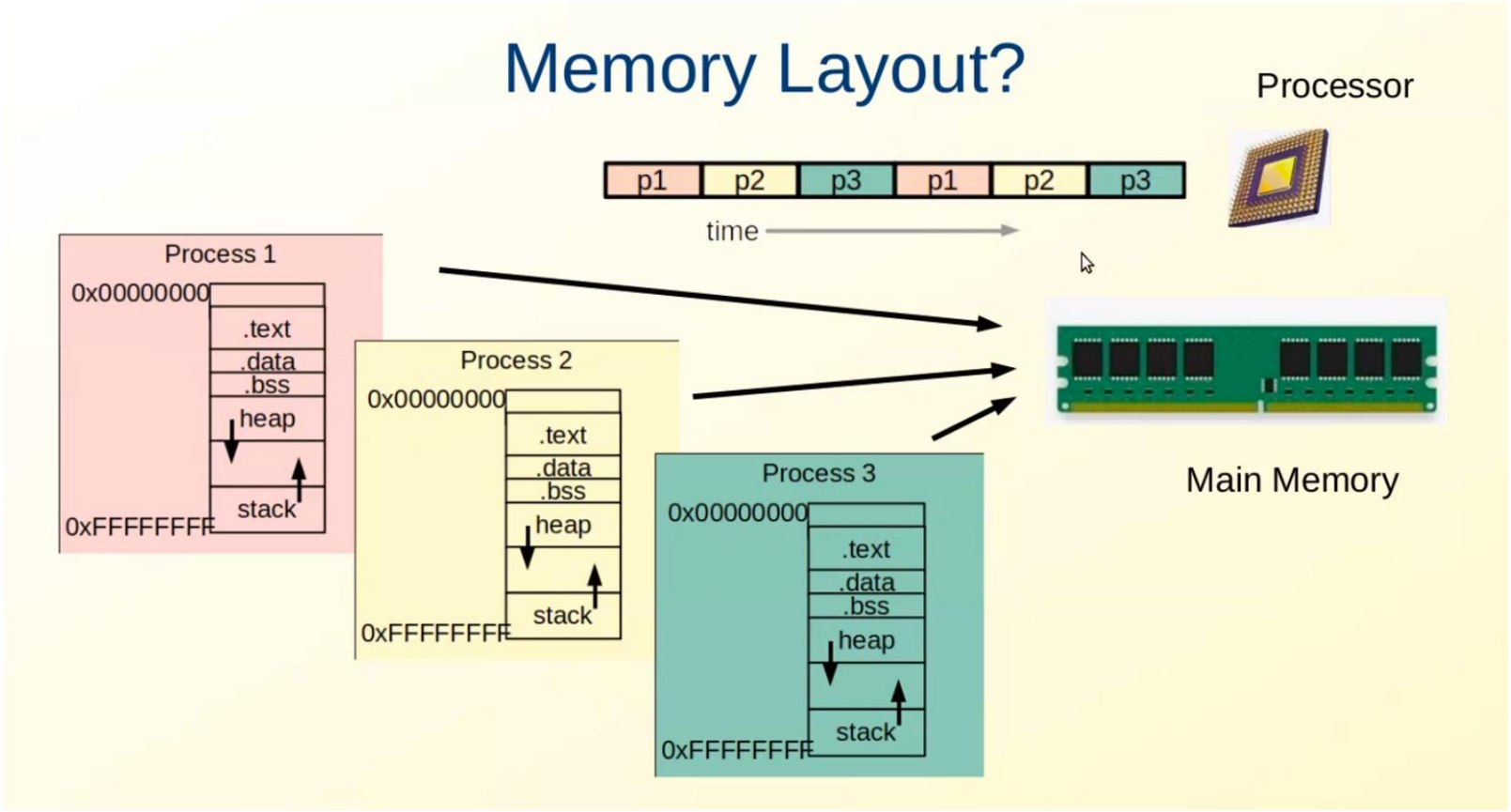


Distinct Process Address Space



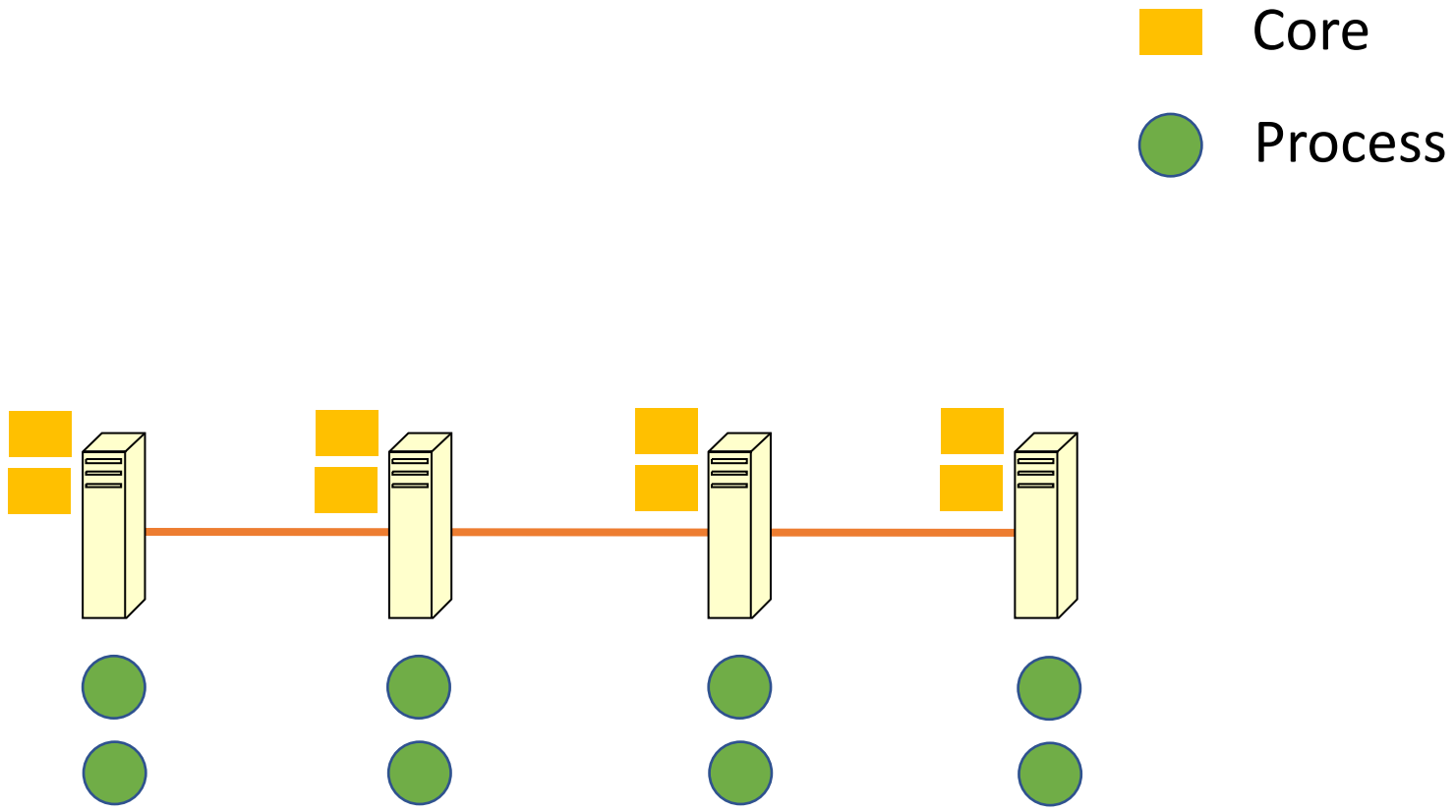
Distinct Process Address Space





Adapted from Neha Karanjkar's slides

Our Parallel World



NO centralized server/master

Message Passing Interface

Message Passing Interface (MPI)

- Efforts began in 1991 by Jack Dongarra, Tony Hey, and David W. Walker.
- Standard for message passing in a distributed memory environment
- MPI Forum in 1993
  - Version 1.0: 1994
  - Version 2.0: 1997
  - Version 3.0: 2012
  - Version 4.0: 2021
  - Version 5.0 (under discussion)

MPI Implementations

"The MPI standard includes point-to-point message-passing, collective communications, group and communicator concepts, process topologies, environmental management, process creation and management, one-sided communications, extended collective operations, external interfaces, I/O, some miscellaneous topics, and a profiling interface." – MPI report

- MPICH (ANL)
- MVAPICH (OSU)
- OpenMPI
- Intel MPI
- Cray MPI

Programming Environment

- Shell scripts (e.g. bash)
- ssh basics
  - E.g. ssh –X
  - …
- Mostly in C/C++
- Compilation, Makefiles, ...
- Linux environment variables
  - PATH
  - LD_LIBRARY_PATH
  - …

MPI Installation – Laptop

- Linux or Linux VM on Windows
  - apt/snap/yum/brew
- Windows
  - No support

- https://www.mpich.org/documentation/guides/

MPI

- Standard for message passing
- Explicit communications
- Medium programming complexity
- Requires communication scope

Simple MPI Code

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  // initialize MPI
  MPI_Init (&argc, &argv);

  printf ("Hello, world!\n");

  // done with MPI
  MPI_Finalize();
}

~
~
```

MPI Code Execution Steps

- Compile
  - mpicc -o program.x program.c
- Execute
  - mpirun -np 1 ./program.x (mpiexec -np 1 ./program.x)
    - Runs 1 process on the launch/login node
  - mpirun -np 6 ./program.x
    - Runs 6 processes on the launch/login node

Output – Hello World

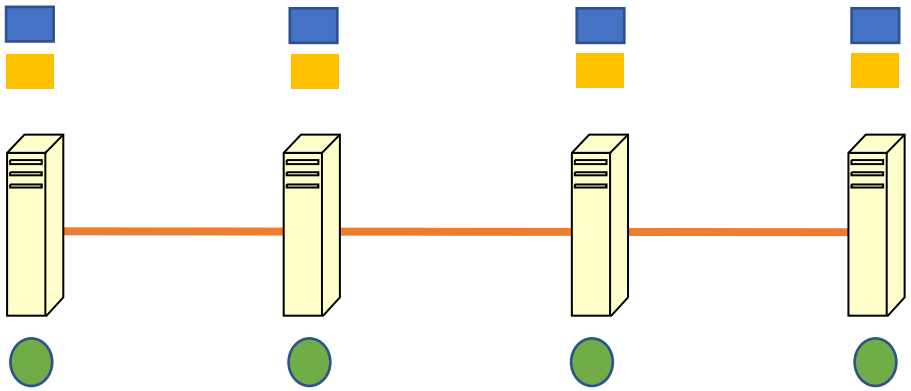
$ mpirun –np 20 ./program.x

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!Hello, world!
Hello, world!
Hello, world!
Hello, world!

Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

Parallel Execution

```
$ ps (check using
this command on each
host)

parallel.x process
will be running on
each host
```

SIMD



Core
Process
Memory

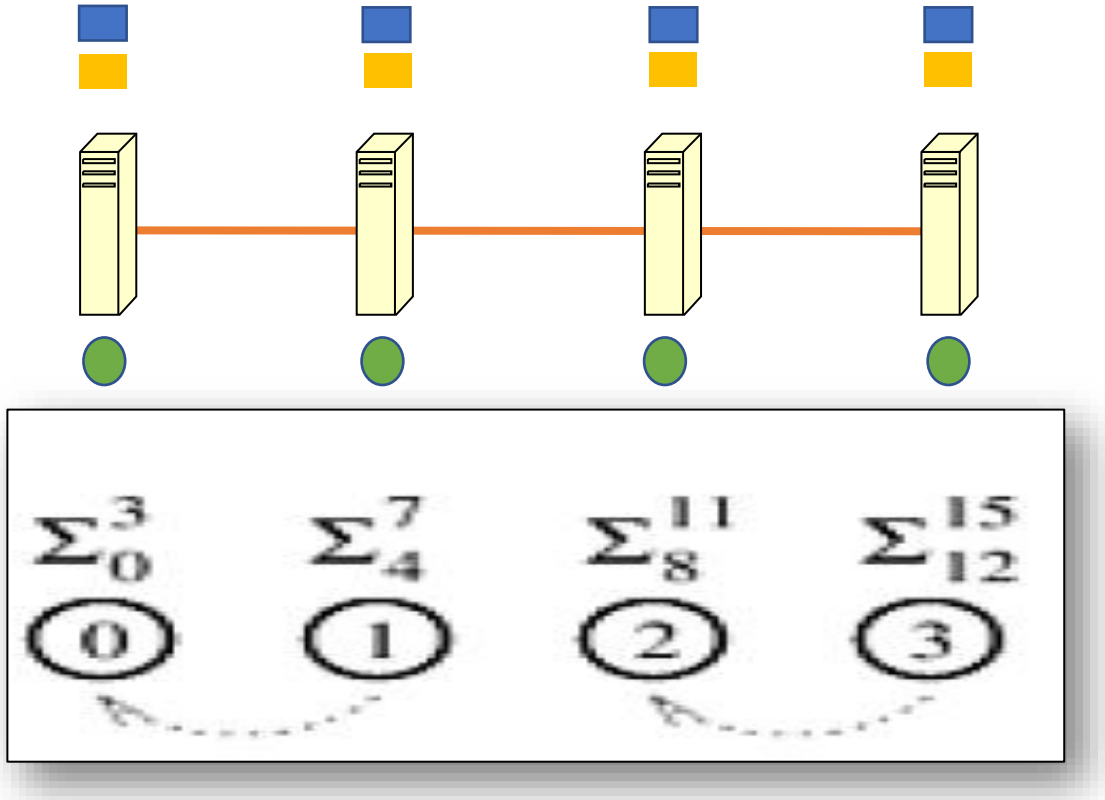
Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …

parallel.x parallel.x parallel.x parallel.x

mpirun –np 4 –f hostfile ./parallel.x

Parallel Execution

SIMD



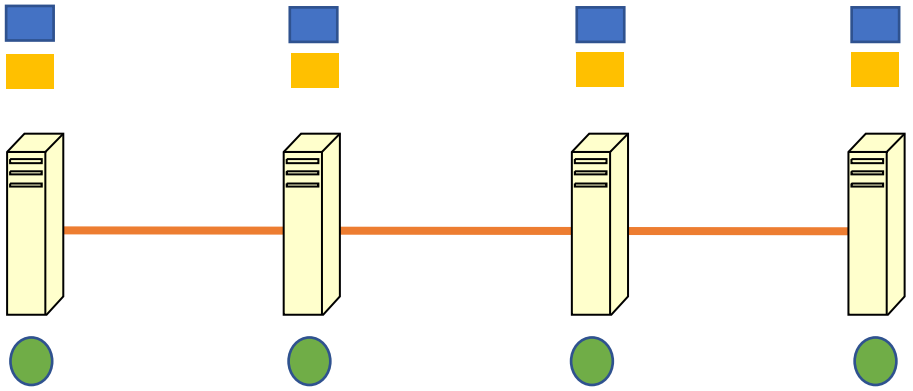
parallelsum .x | parallelsum .x | parallelsum .x | parallelsum .x

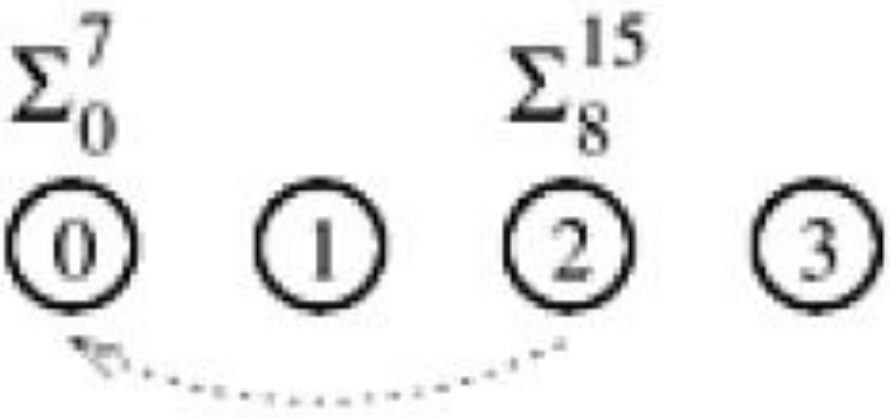
Step 1: Divide array (N) into P processes
Step 2: Local computation
Step 3: Communication rounds (log P)

mpirun –np 4 –f hostfile ./parallelsum.x

Parallel Execution

SIMD

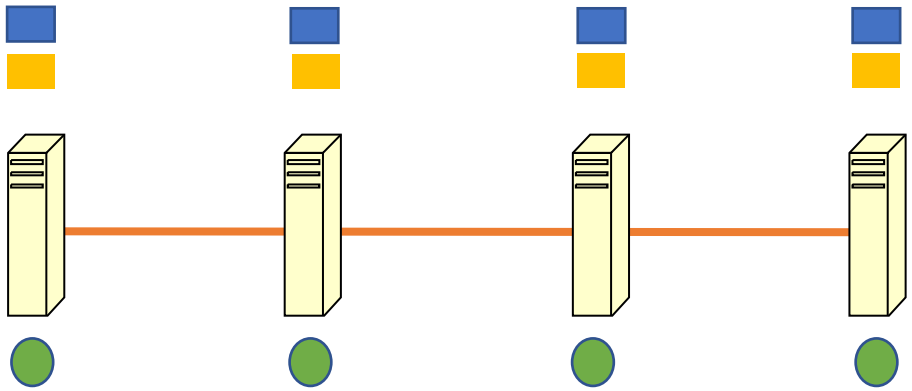




Step 1: Divide array (N) into P processes
Step 2: Local computation
Step 3: Communication rounds (log P)

Parallel Execution

```
$ ps (on each host)

parallel.x
parallel.x
```



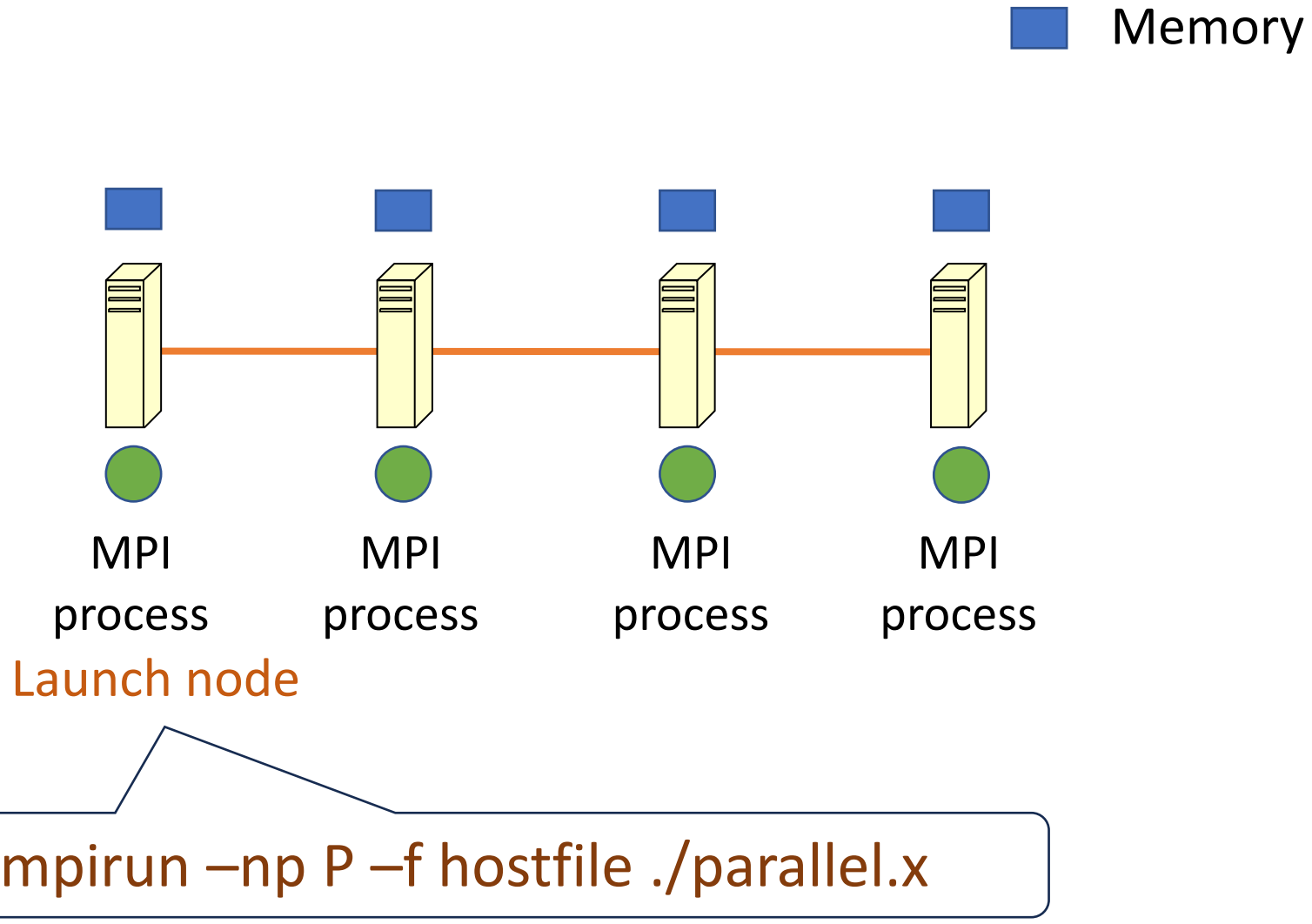
Core
Process
Memory

SIMD

Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …	Instruction 1 Instruction 2 …
parallel.x parallel.x	parallel.x parallel.x	parallel.x parallel.x	parallel.x parallel.x

mpirun –np 8 –f hostfile ./parallel.x

Process Launch



MPI Installation – BYO Cluster

Install MPICH 4.3.2 (https://www.mpich.org/static/downloads/4.3.2/) in a directory

- Download mpich-4.3.2.tar.gz
- Follow installation instructions from https://www.mpich.org/static/downloads/4.3.2/mpich-4.3.2-installguide.pdf
- Use the same installation path on all systems (e.g. /home/test)
- Verify after installation that `which mpirun` from any node points to your installation
- Create a user name and enable passwordless ssh (ssh-keygen)

MPICH Installation

- apt-get install mpich
- brew install mpich

- Windows
  - Install Linux VM

Doubts?

https://forms.gle/CuBWQgLrg1YjNNuS9

Assignment Groups

https://forms.gle/QQUMV3Ft73GbxdZQ7