CS 698L: Parallel Architecture and Programming Models

Swarnendu Biswas

Semester 2019-2020-I

CSE, IIT Kanpur

Content influenced by many excellent references, see References slide for acknowledgements.

How can we sum up all elements in an array?

```
int array[1000] = {0, 1, 34, 2, 89, -5, 67, 8, 4, 56,
                   23, 67, 0, 9, …}
```

$$sum = \sum_{i=1}^{n} array[i]$$

Comparing Implementations

Main Thread

```
long sum = 0;

for (int i =0; i < LEN; i++) {
  sum += array[i];
}
```

Comparing Implementations

Main Thread

```
long sum = 0;

for (int i =0; i < LEN; i++) {
  sum += array[i];
}
```

Main Thread

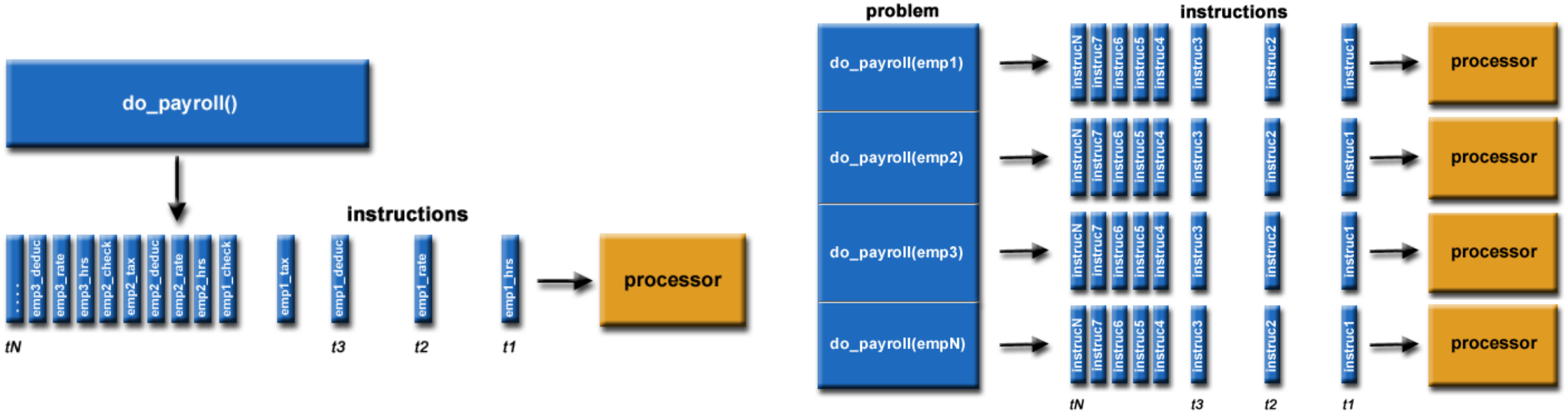
```
  Spawn n threads
  long thr_sum[n] = {0}

  for (int i = 0; i < n; i++) {
    sum += thr_sum[i];
  }

Thread i

  Compute CHUNK i of array[]
  for (int j = CHUNK_START; j + CHUNK_START <
CHUNK_END; j++) {
    thr_sum[j] += array[j];
  }
```

Serial vs Parallel Processing



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

Is it Worth the Extra Complexity?

```
1: fish  /home/swarnendu/iitk-workspace/c++-examples/src
~/i/c/src $ ./a.out
Sequential sum: 499158189 Time (ns): 2119657

Parallel sum: 499158189 Time (ns): 147934

~/i/c/src $ ./a.out
Sequential sum: 499019481 Time (ns): 2063707

Parallel sum: 499019481 Time (ns): 259234

~/i/c/src $ ./a.out
Sequential sum: 498973205 Time (ns): 2113602

Parallel sum: 498973205 Time (ns): 257328

~/i/c/src $ ./a.out
Sequential sum: 499697650 Time (ns): 2110496

Parallel sum: 499697650 Time (ns): 252351

~/i/c/src $
```

Array of unsigned ints of size 10^6, and four threads

Order of magnitude improvement

Parallel Programming Overview

Find parallelization opportunities in the problem

- Decompose the problem into parallel units

Parallel Programming Overview

Find parallelization opportunities in the problem

- Decompose the problem into parallel units

Create parallel units of execution

- Manage efficient execution of the parallel units

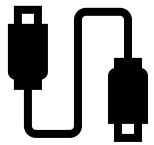
Parallel Programming Overview

Find parallelization opportunities in the problem

- Decompose the problem into parallel units

Create parallel units of execution

- Manage efficient execution of the parallel units

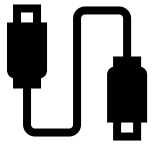


Problem may require inter-unit communication

- Communication between threads, cores, …

Inter-unit Communication

The problem logic will possibly require inter-unit communication



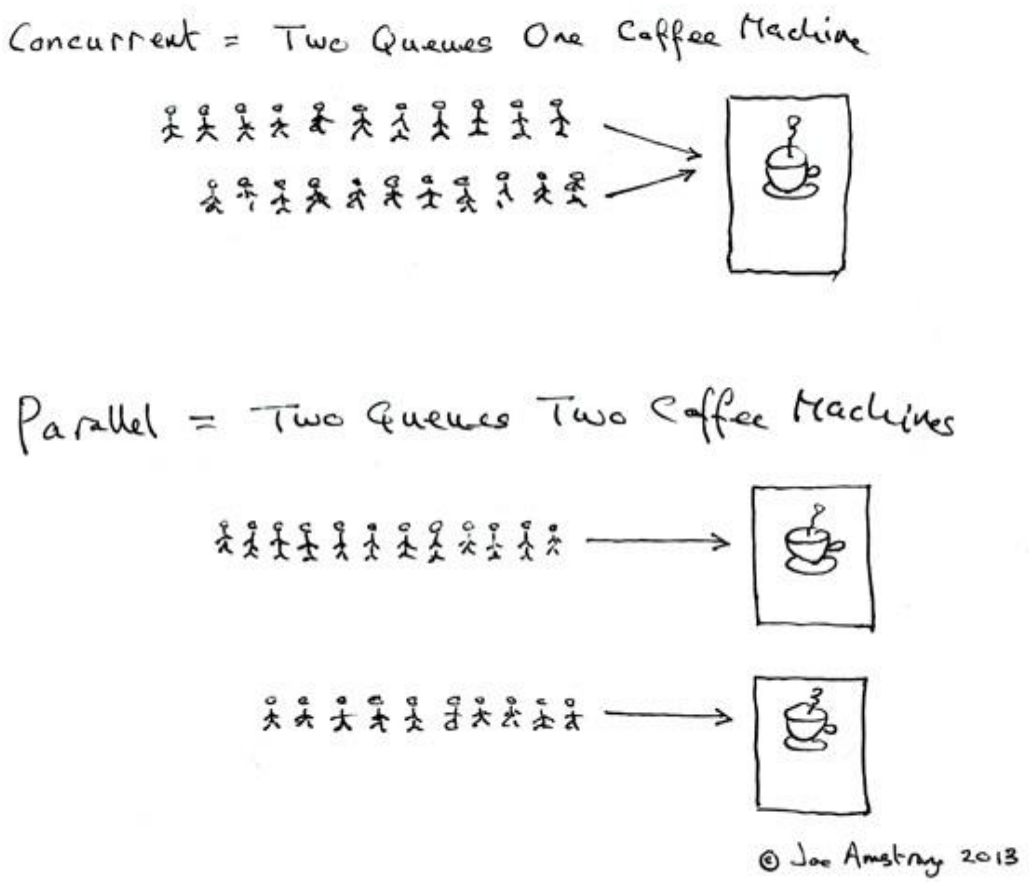
Units may be on the same processor or across processors or across nodes

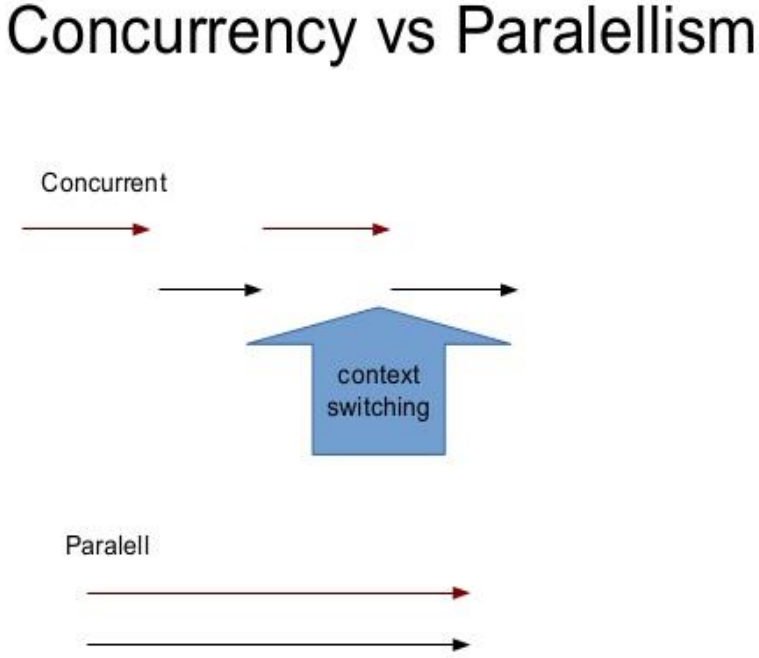
What do we communicate in sequential programs?

What do we communicate in sequential programs?

- Global variables or data structures
- Function arguments and call parameters

Parallelism vs Concurrency





Parallelism vs Concurrency

Parallel programming

- Use additional resources to speed up computation
- Performance perspective

Concurrent programming

- Correct and efficient control of access to shared resources
- Correctness perspective

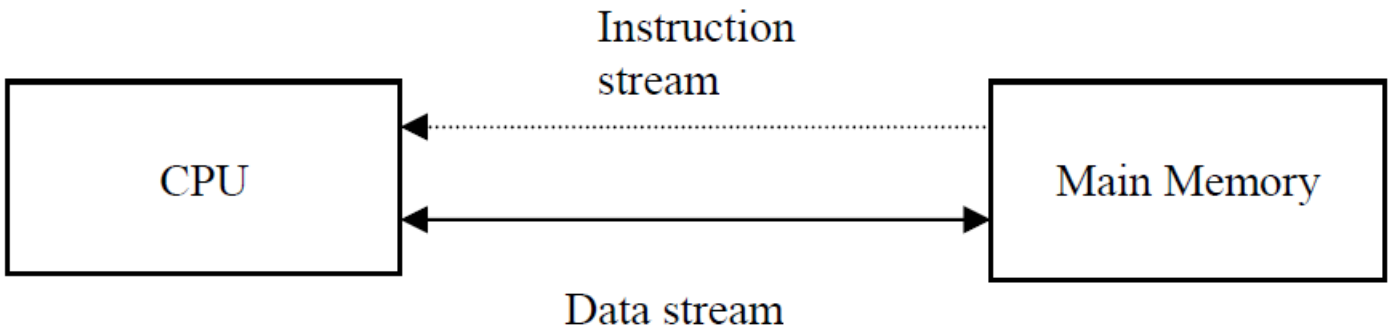
Distinction is not absolute

Parallel Architectures

Quick Overview

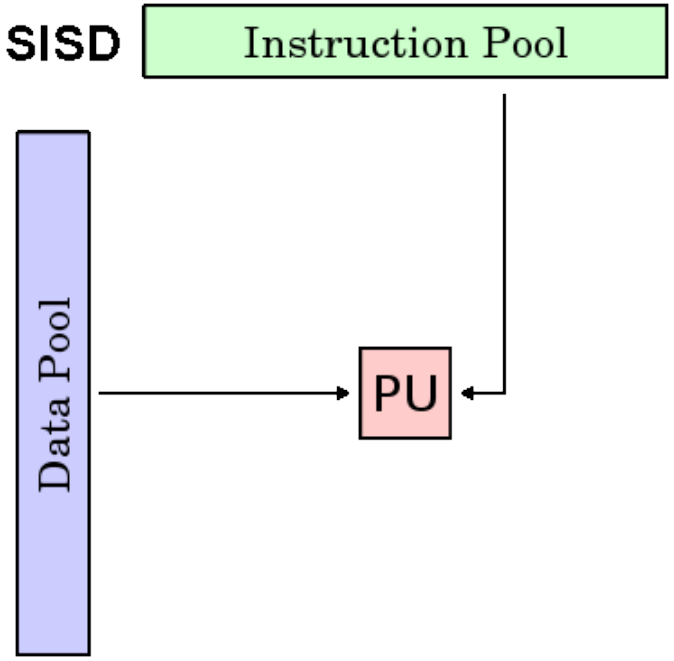
Architecture Classification

- Popular dimensions for classification
  - Instruction and data stream
  - Source of parallelism
  - Structure of the system

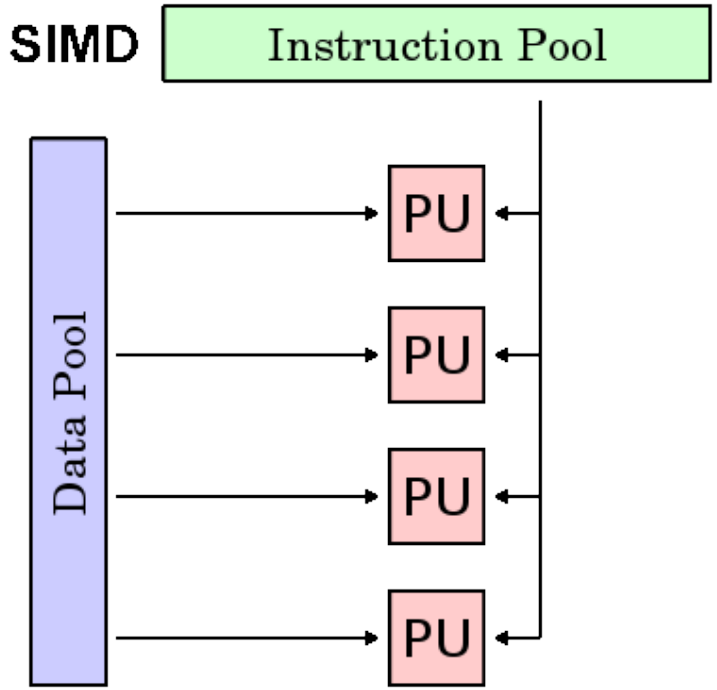


Flynn's Taxonomy

- Single Instruction Single Data

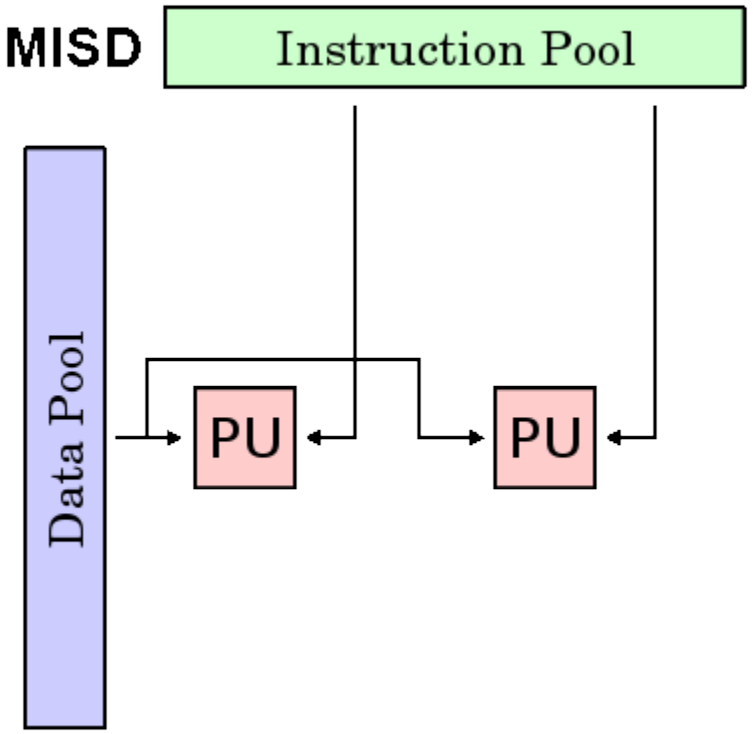


- Single Instruction Multiple Data

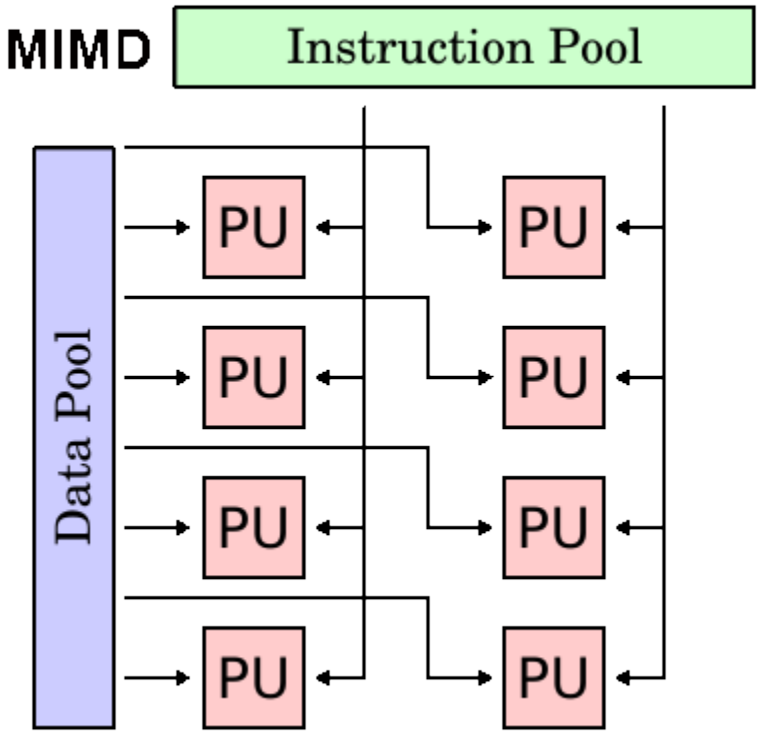


Flynn's Taxonomy

- Multiple Instructions Single Data



- Multiple Instructions Multiple Data



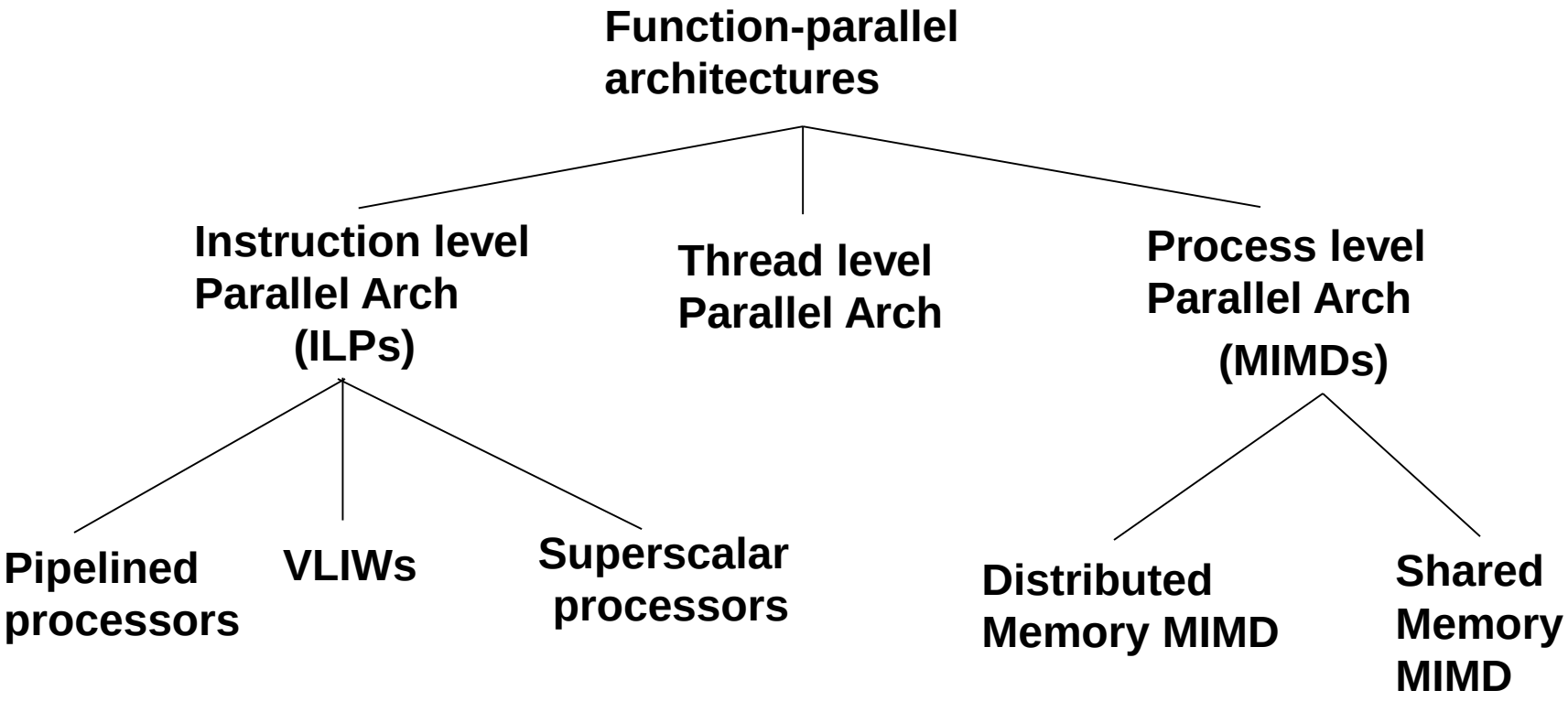
Sources of Parallelism in Hardware

- Instruction-Level Parallelism (ILP)
  - Pipelining, out-of-order execution, Superscalar, VLIW, …
- Data parallelism
  - Increase amount of data to be operated on at same time
- Processor and resource parallelism
  - Increase units, memory bandwidth, …

Source of Parallelism

Data	Control/Function
• Vector processors, systolic arrays, and SIMD	• Pipelined, superscalar, VLIW processors • Shared-memory systems, distributed memory systems

Control Parallel Architectures



Xin Yuang - Parallel Computer Architecture Classification. UFL.

Modern Classification

Uniprocessor

- Scalar processor
- Vector processor
- SIMD

Multiprocessor

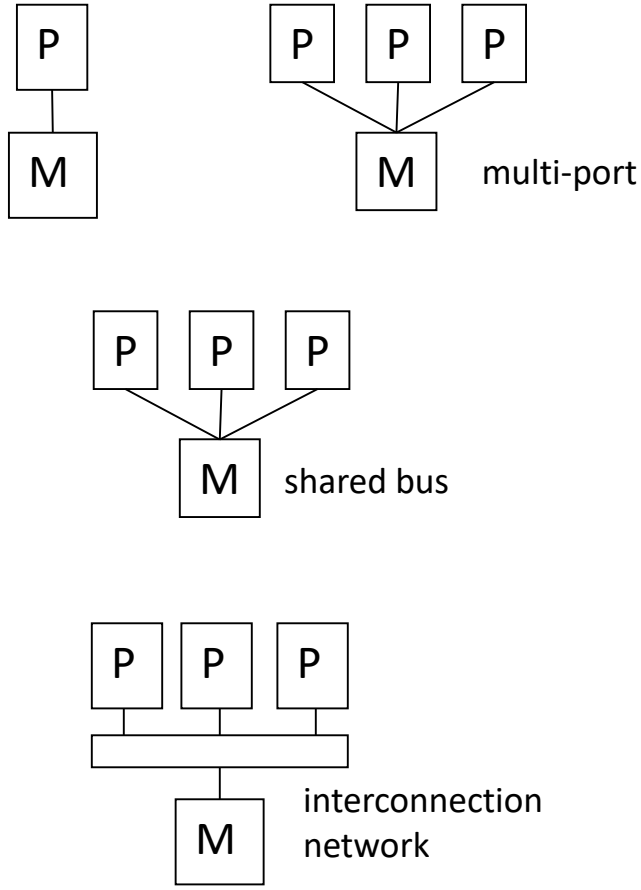
- Symmetric multiprocessors (SMP)
- Distributed memory multiprocessor
- SMP clusters
  - Shared memory addressing within node
  - Message passing between nodes

Performance Metrics of Parallel Architectures

- MIPS – million instructions per second
- MFLOPS – million floating point operations per second
- Which is a better metric?

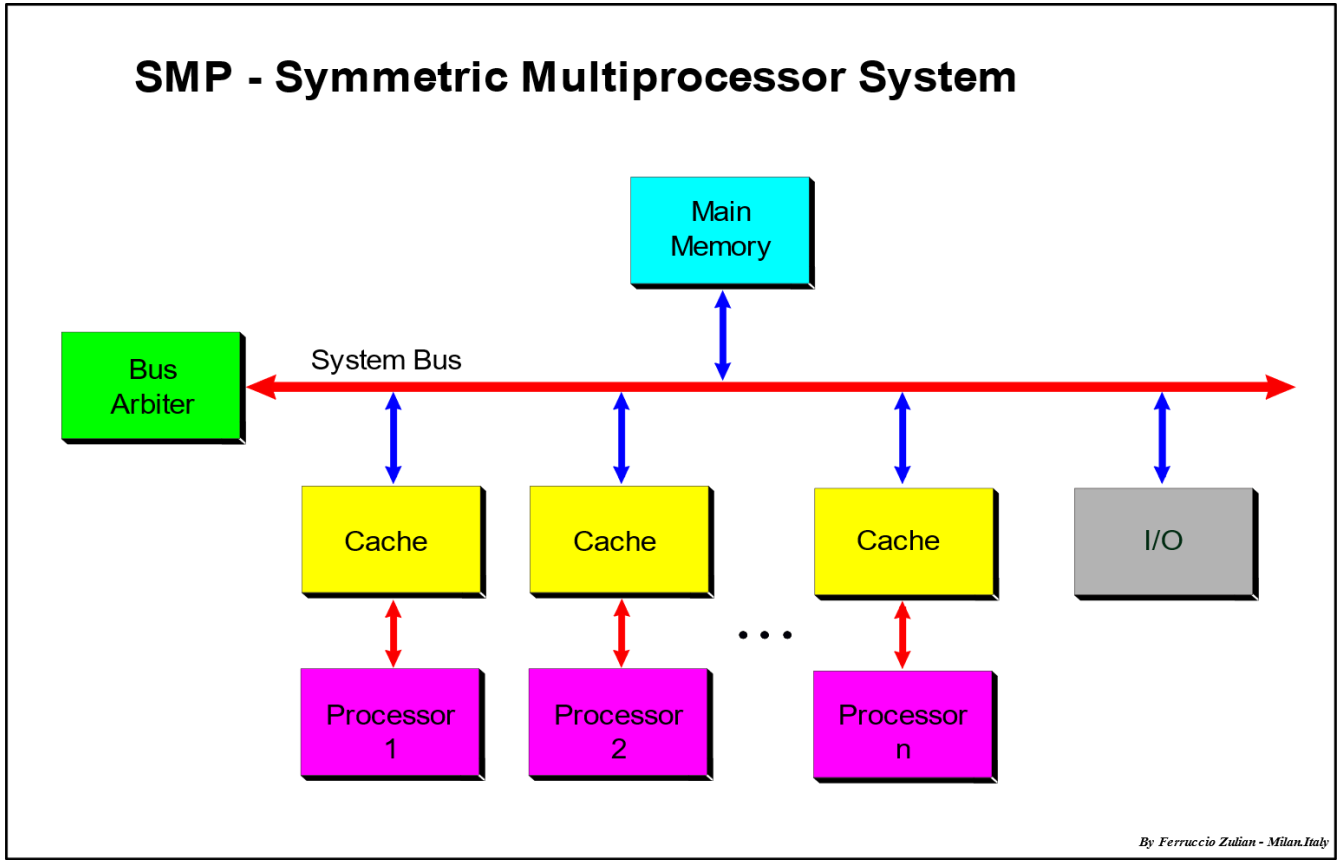
Shared Memory Architecture

- Single address space shared by multiple cores
- Communication is implicit through memory instructions (i.e., loads and stores)
- Can share data efficiently



Implementing Shared Memory

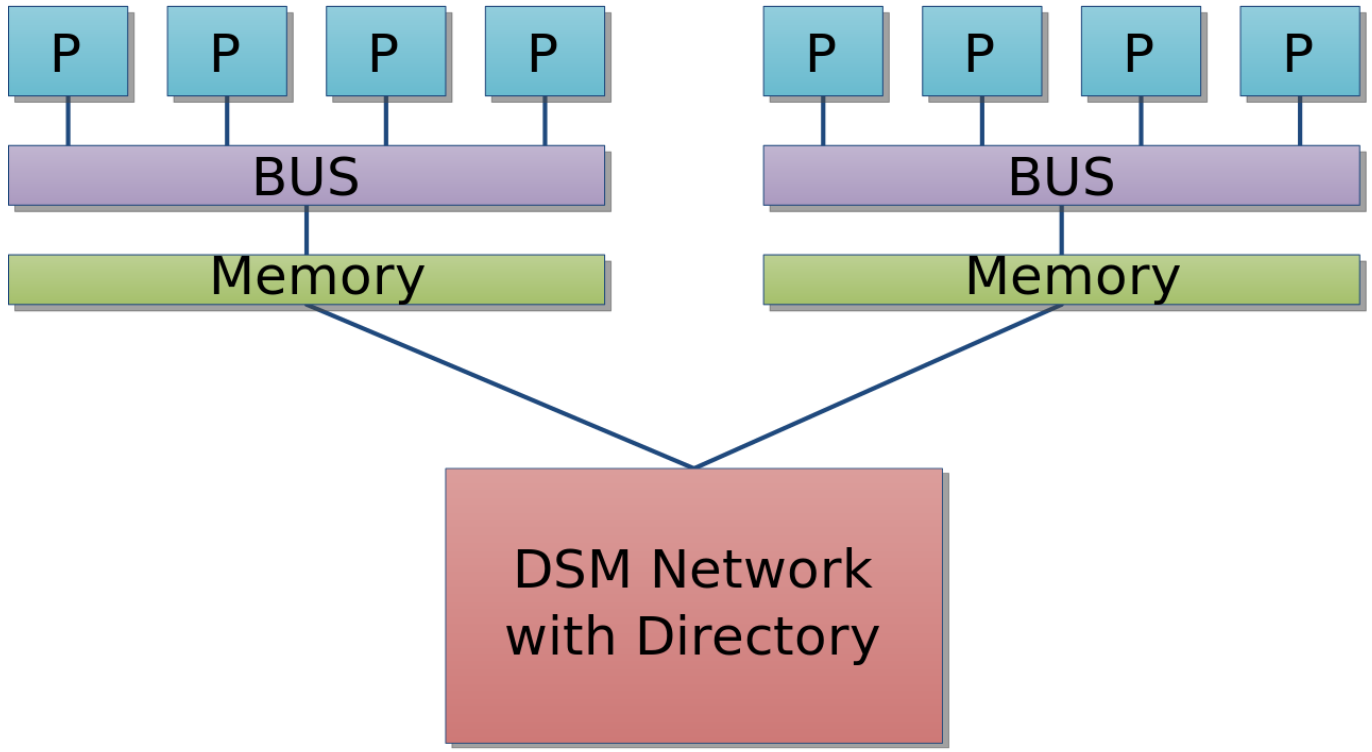
- Uniform memory access (UMA)
- Interconnection network used in the UMA can be a single bus, multiple buses, or a crossbar switch



Wikipedia.

Implementing Shared Memory

- Non-uniform memory access
- Memory access time depends on the distance from the core



Challenges with Shared Memory

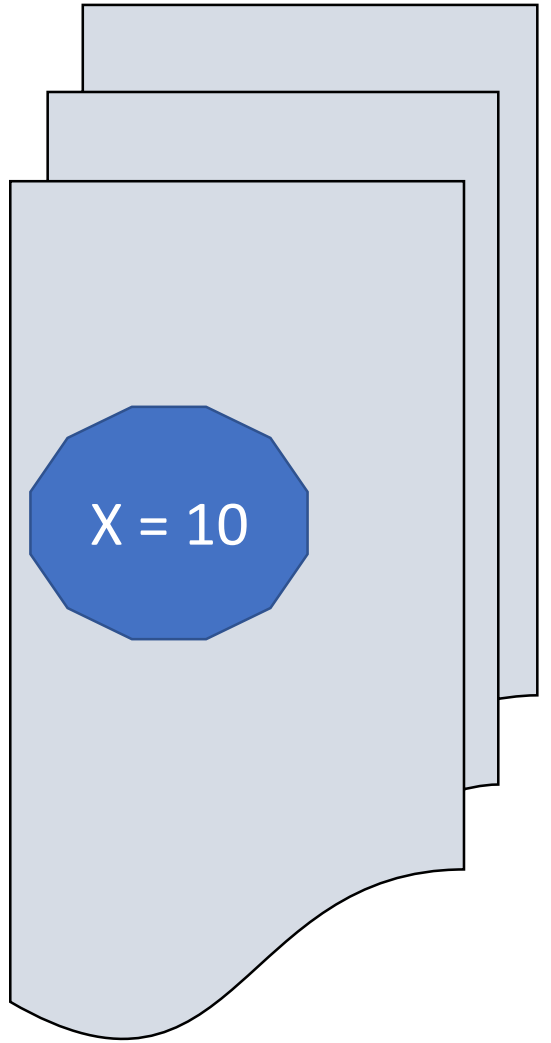
- Caches play key role in SMP performance
  - Reduce average data access time, reduce interconnect bandwidth
- However, private caches create problem of data coherence
  - Copies of a variable can be present in multiple caches

Sequence of Operations

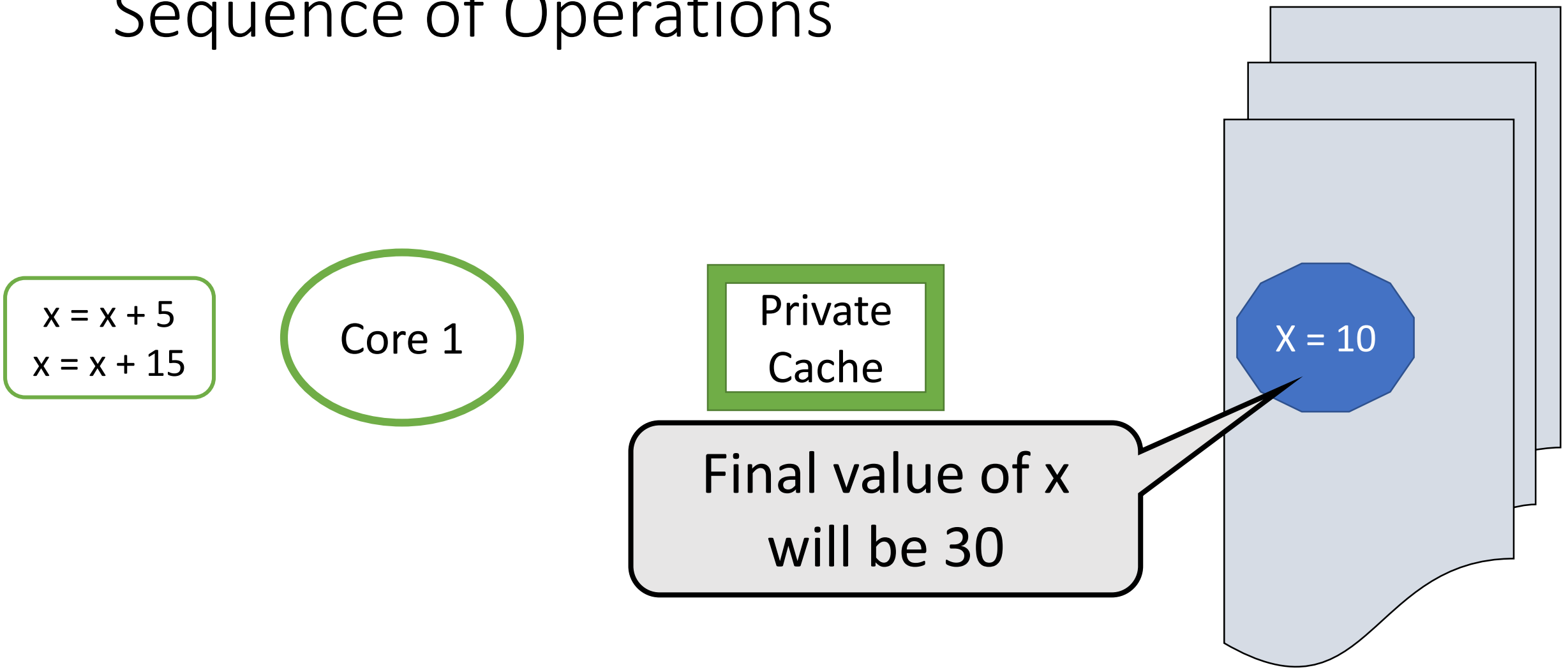
x = x + 5
x = x + 15

Core 1

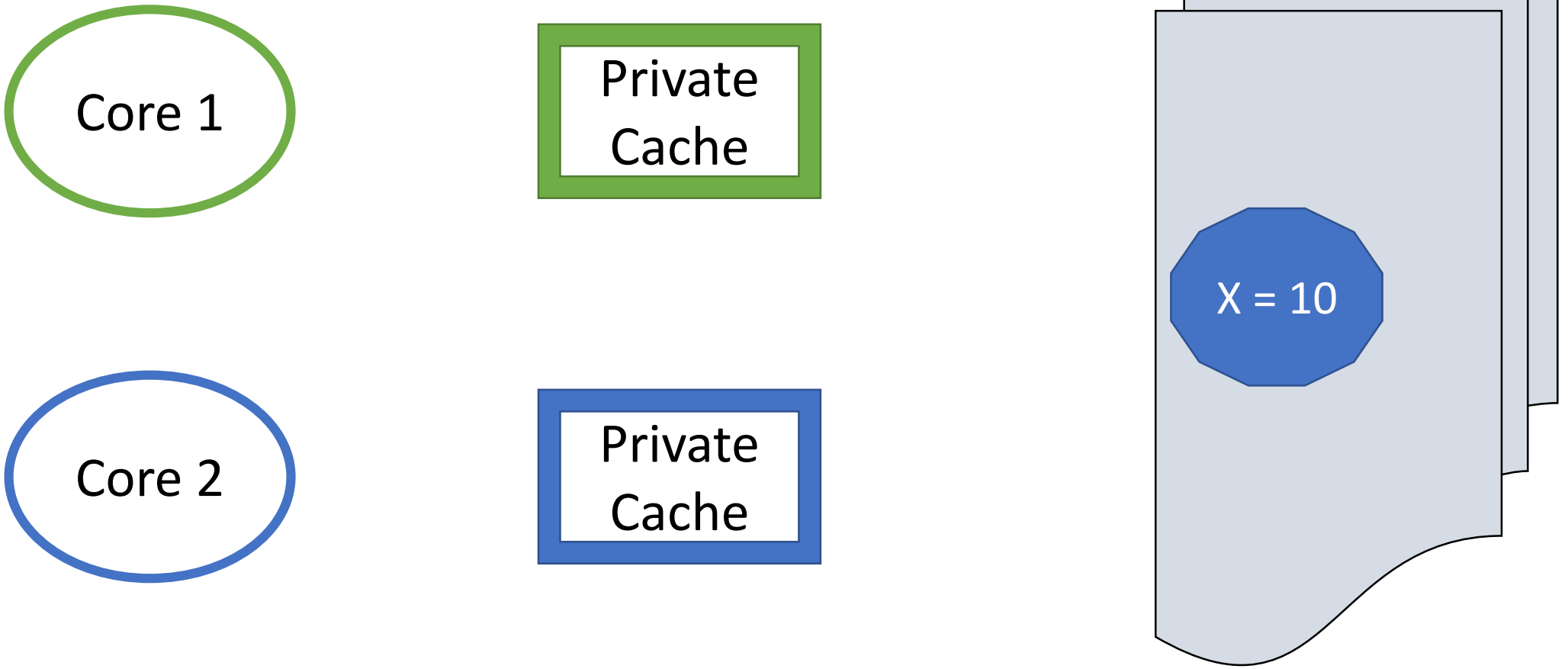
Private Cache



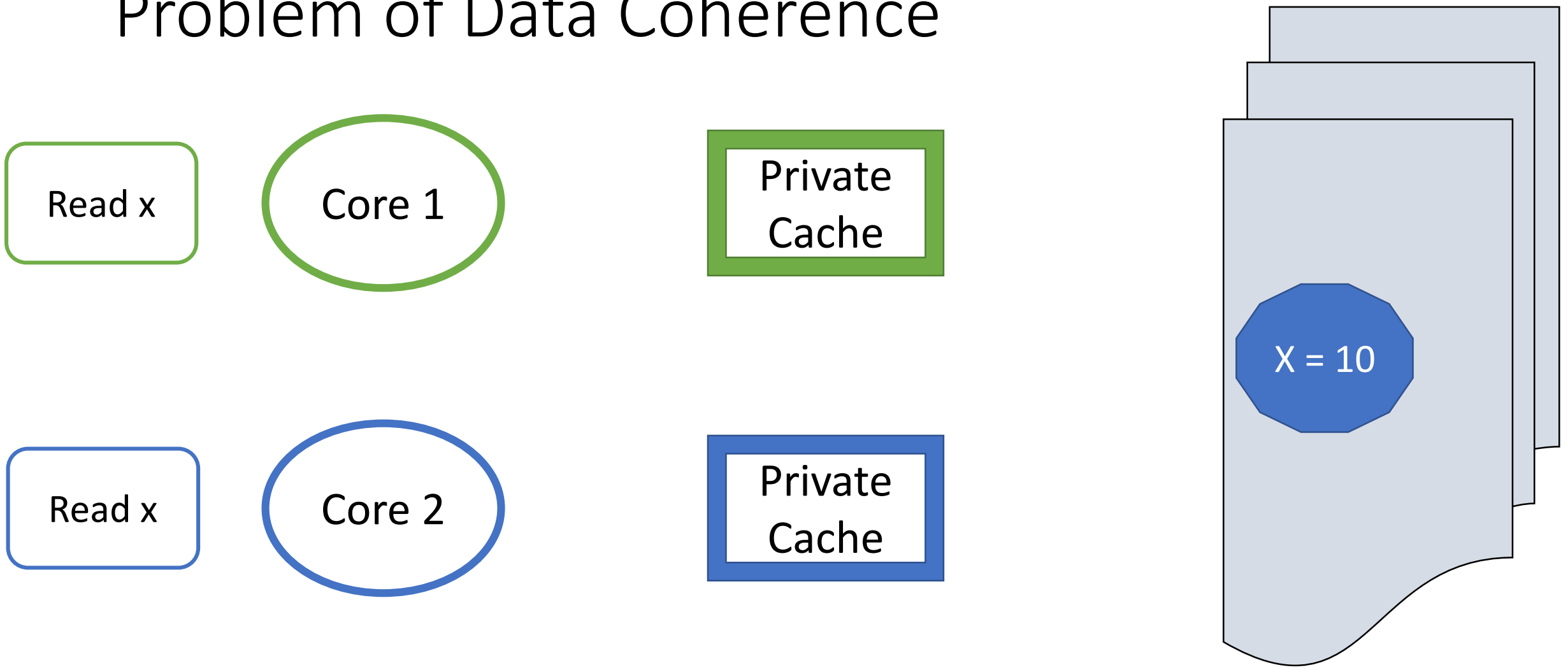
Sequence of Operations



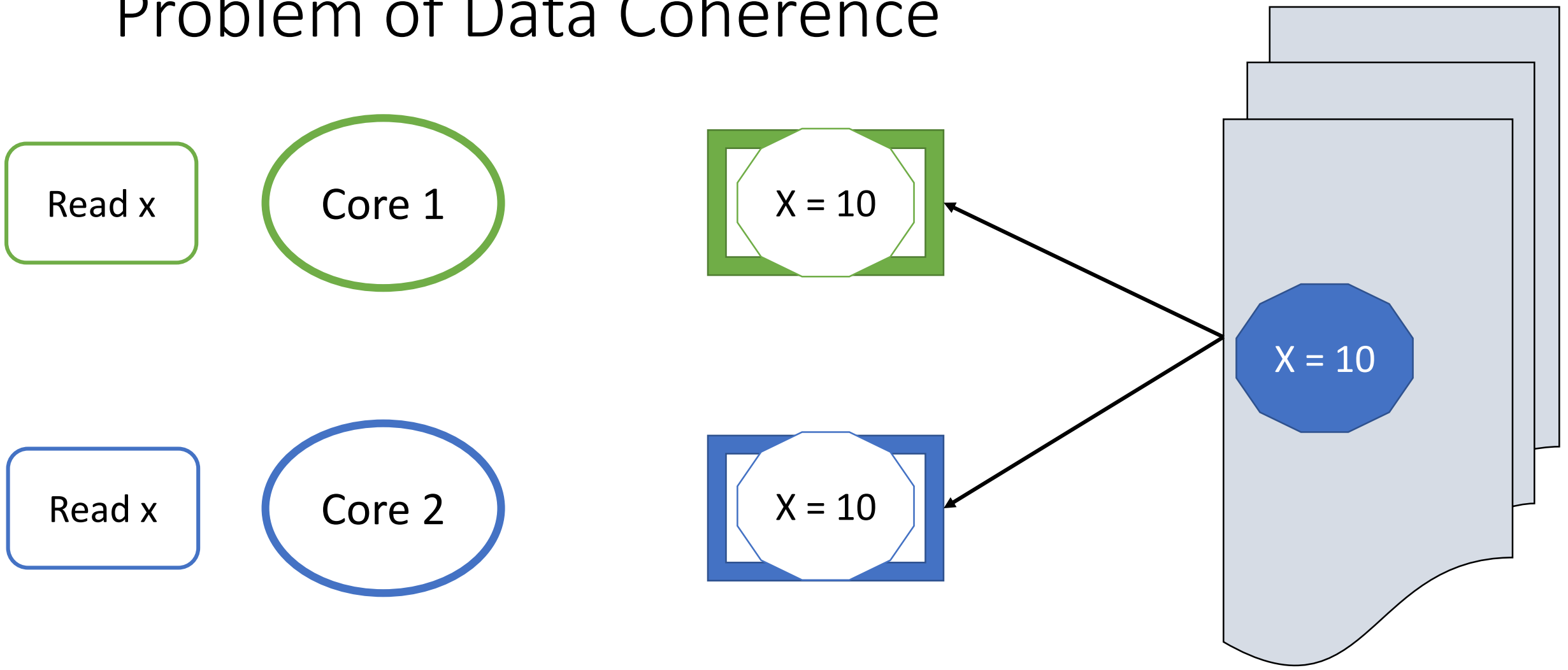
Problem of Data Coherence



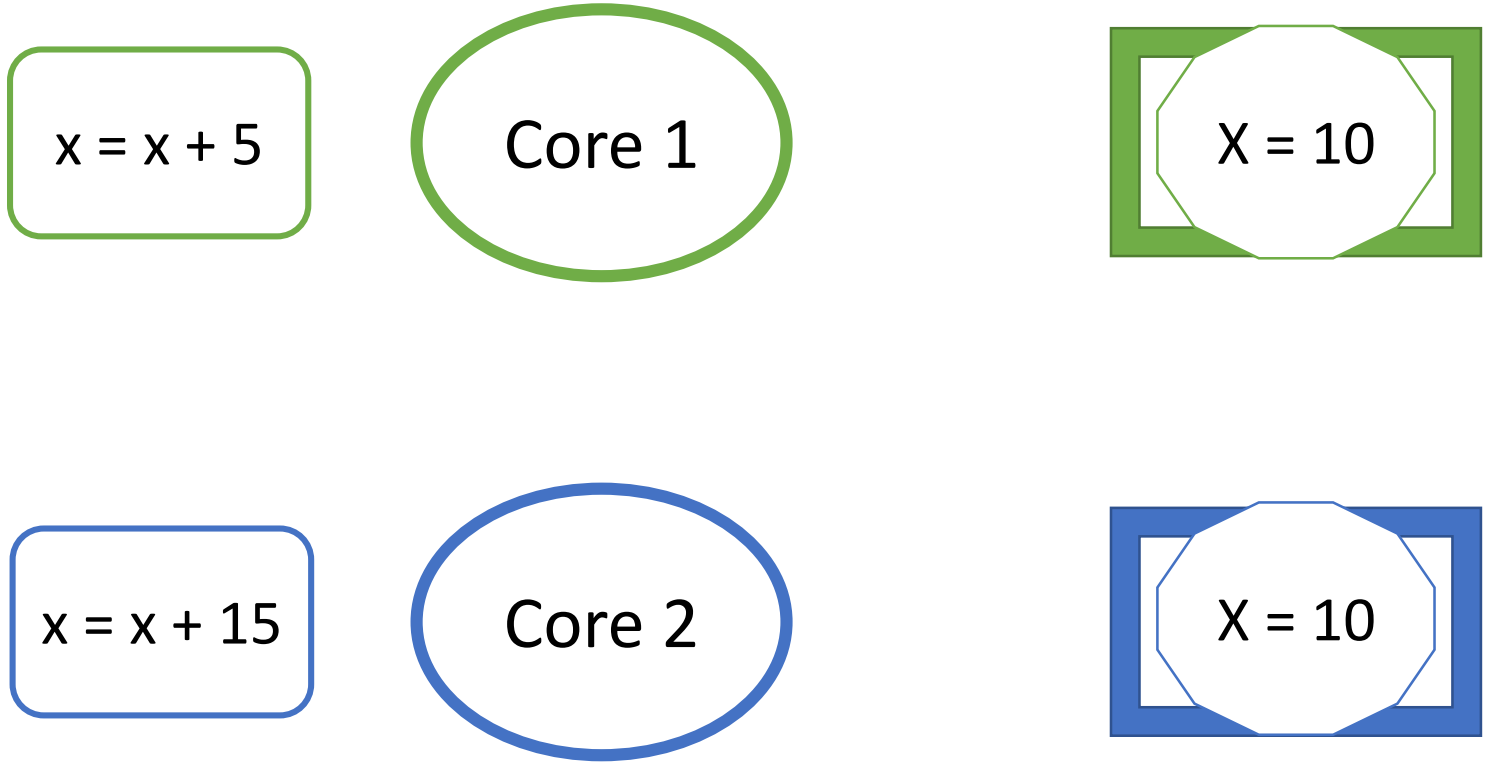
Problem of Data Coherence

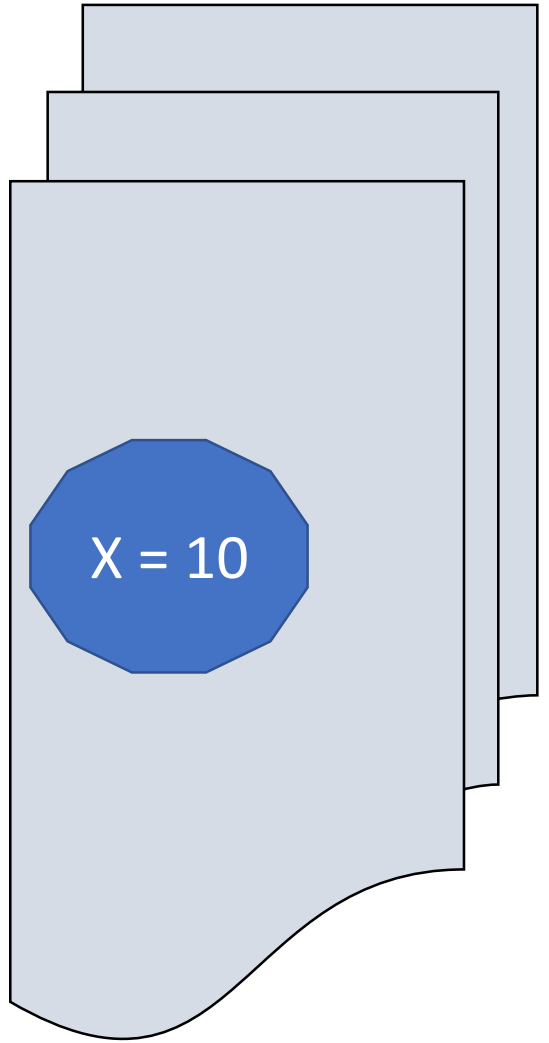


Problem of Data Coherence



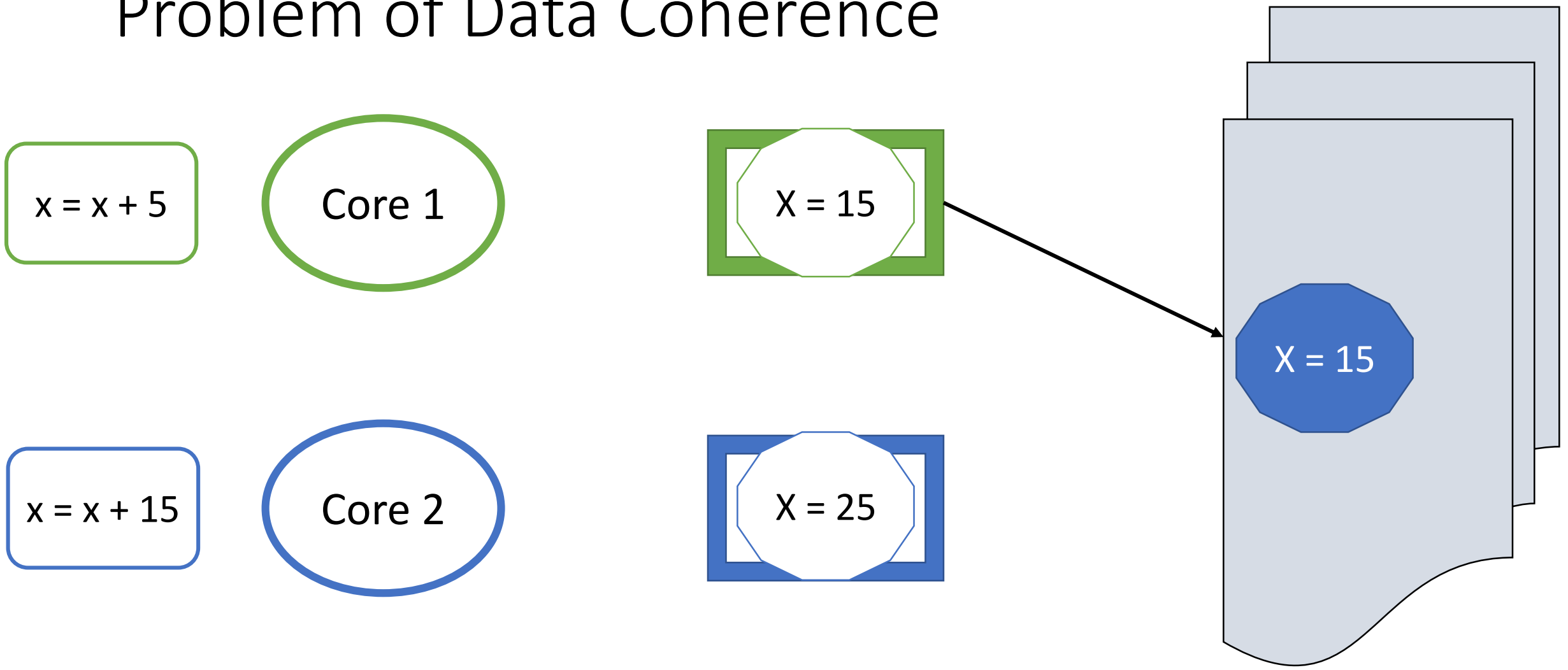
Problem of Data Coherence





Problem of Data Coherence

Problem of Data Coherence



Problem of Data Coherence

Challenges with Shared Memory

- Caches play key role in SMP performance
  - Reduce average data access time, reduce interconnect bandwidth
- Private caches create data coherence problem
  - Copies of a variable can be present in multiple caches
- Need support for cache coherence

Challenges with Shared Memory

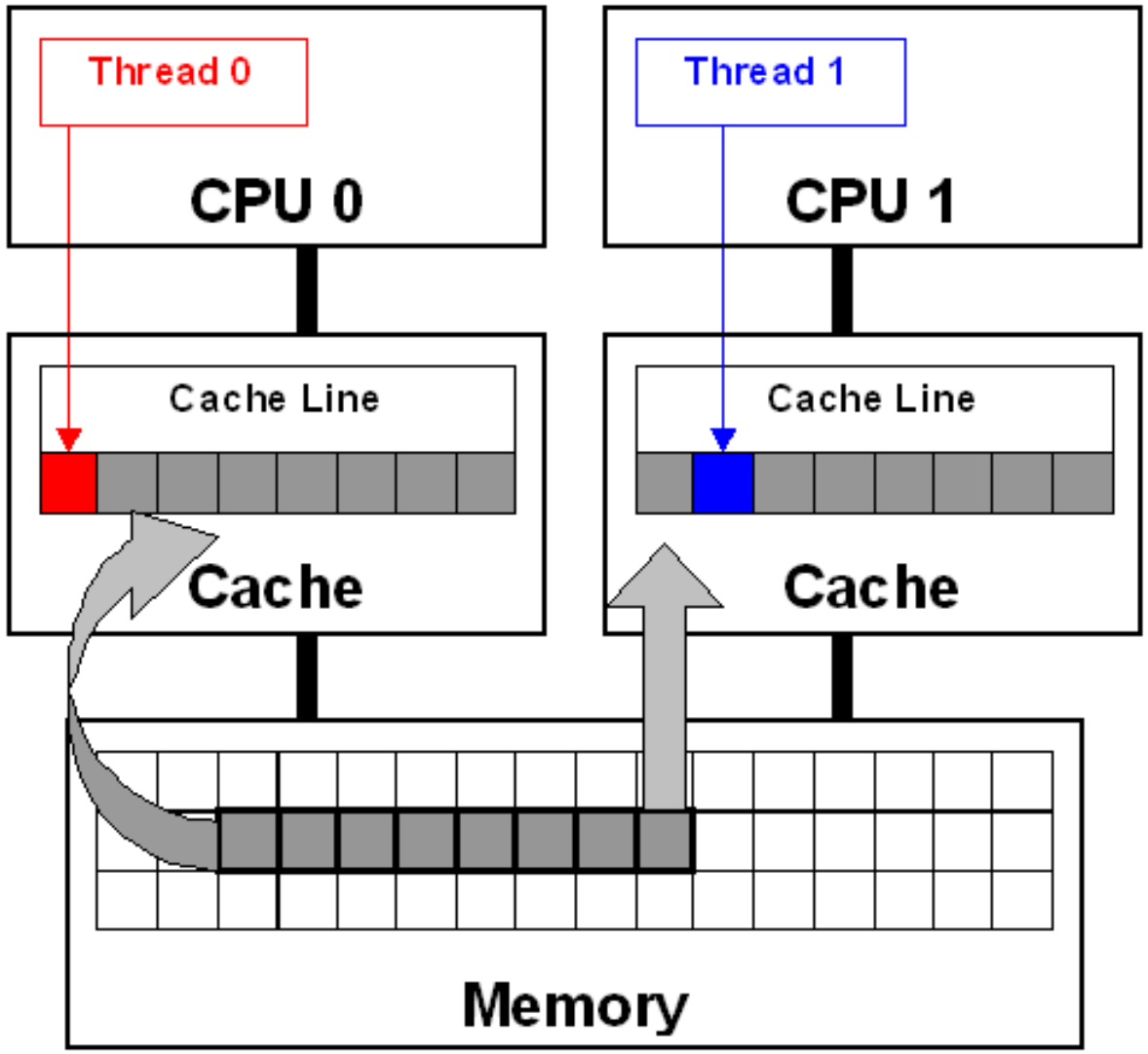
- Access conflicts - several threads can try to access the same shared location
  - Data race is the accesses are not correctly synchronized and one the accesses is a write
  - Synchronization is not cheap
  - Programmer responsible for synchronized accesses to memory

- Coherence operations can become a bottleneck
  - Takes time and effort in keeping shared-memory locations consistent
  - Traffic due to data and cache/memory management
  - Lack of scalability

  - Other performance hazards – false sharing

False Sharing

What is going on here?

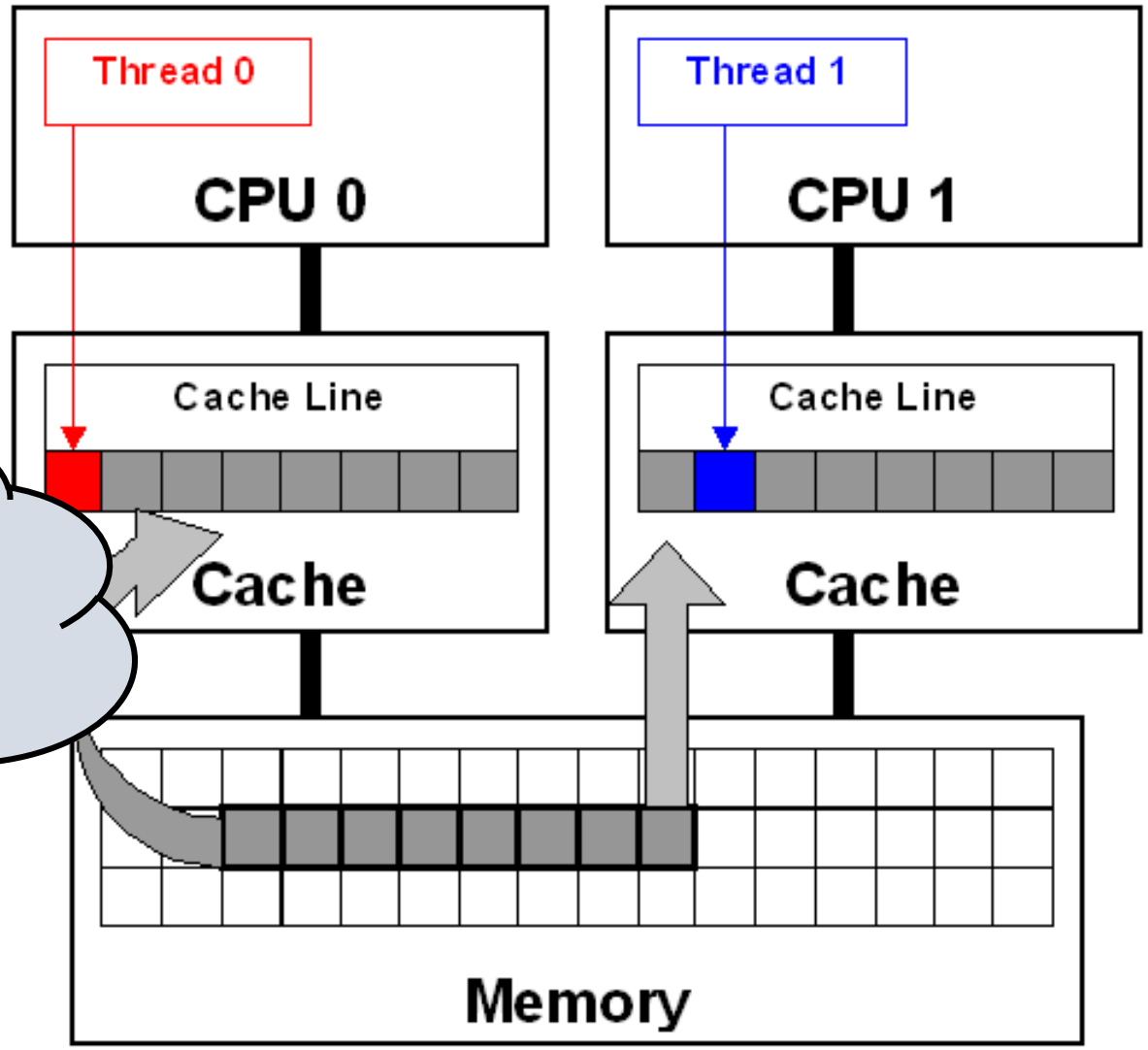


Intel. Avoiding and Identifying False Sharing Among Threads.

False Sharing

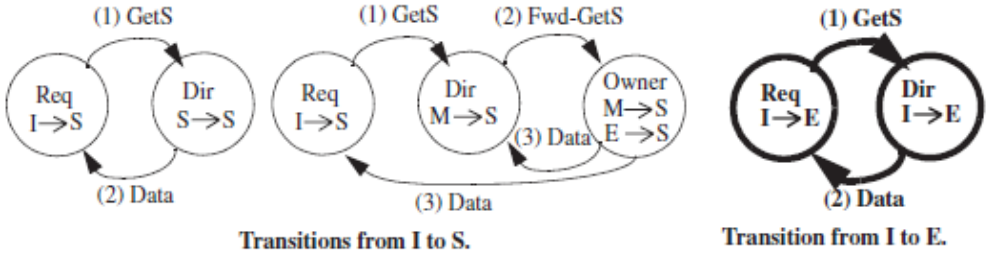
What is going on here?

Does anyone remember the MESI protocol?



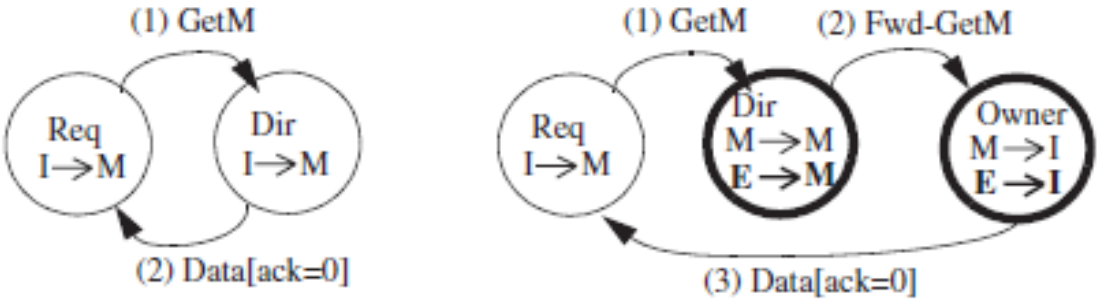
Intel. Avoiding and Identifying False Sharing Among Threads.

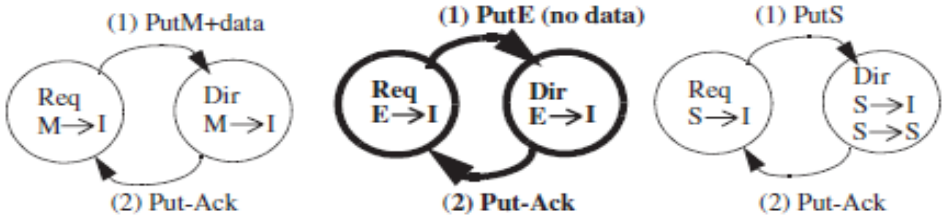
State Transitions in MESI



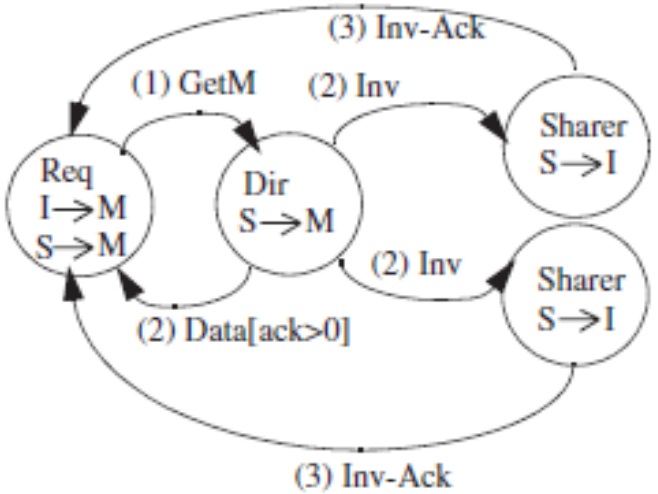
Transitions from I to S.

Transition from I to E.





Transition from M or E or S to I



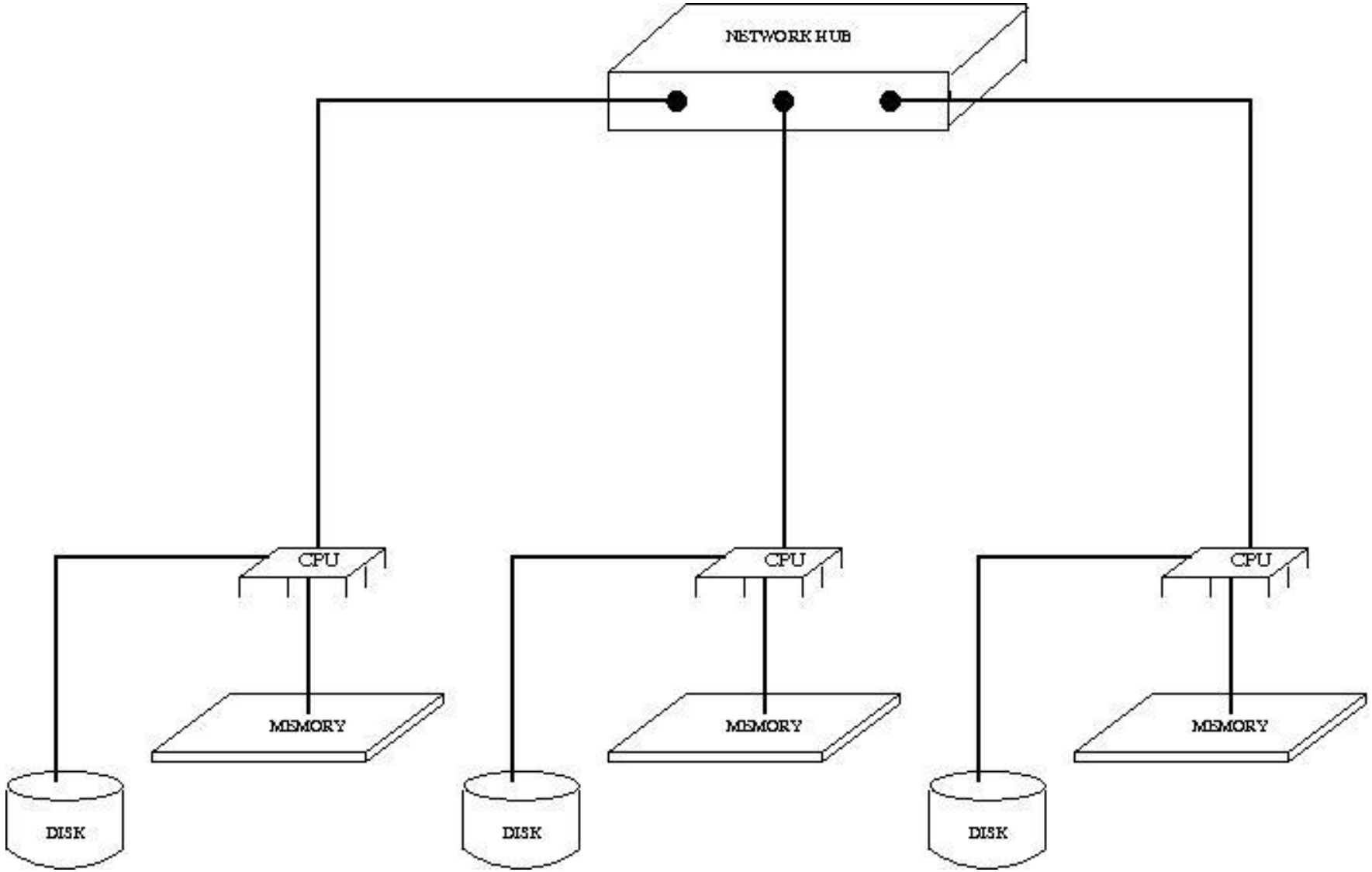
The only sharer might be the requestor, in which case no Invalidation messages are sent and the Data message from the Dir to Req has an AckCount of zero.

Transitions from I or S to M. Transition from E to M is silent.

A Primer on Memory Consistency and Cache Coherence.

Distributed Memory Architecture

- Each processor has its own private memory
  - Physically separated memory address space
- Processor must communicate to access non-local data
  - Also called message passing architecture
- Requires interconnection network for communication
  - Interconnection network topology is a key design factor, determines how the system scales
  - Need high bandwidth for communication



Wikipedia.

Advantages of Distributed Memory

- Memory scales with the number of processors
- Can quickly access your own memory without the need for global coherence
- Can use off-the-shelf components

Be clear with uses!

Parallel computing

- Multiple tasks in a program cooperate to solve a problem efficiently

Concurrent programming

- Multiple tasks in a program can be in progress at the same time

Distributed computing

- A program needs to cooperate with other programs to solve a problem

Yonghong Yan.

Parallel Programming Models

Parallel Programming Models

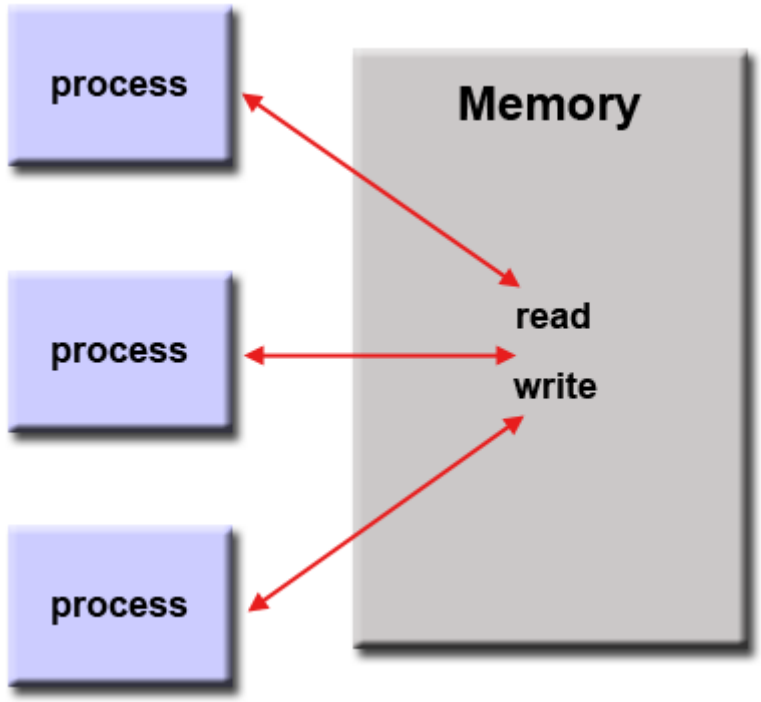
- An abstraction of parallel computer architectures
- Building block to design algorithms and write programs
- Dimensions
  - Performance – how efficiently can programs run
  - Productivity – how easy is it to develop programs

Parallel Programming Models

- Shared-memory
- Distributed memory
- Data parallel (PGAS)
- Single program multiple data (SPMD)
- Multiple program multiple data (MPMD)
- Hybrid

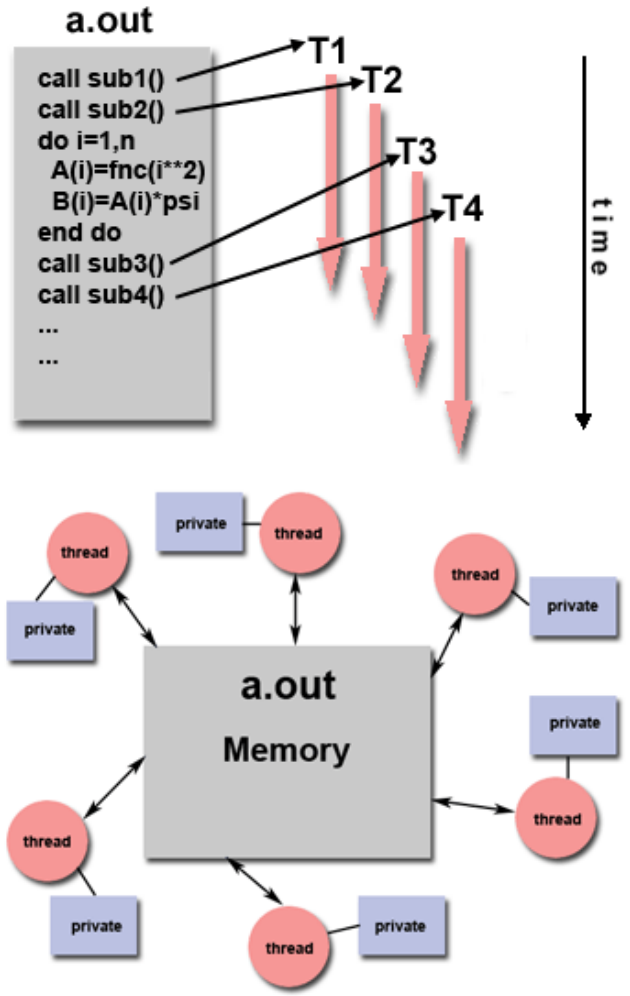
Shared Memory without Threads

- Processes share a common address space
  - Notion of ownership of data is missing, complicating matters
- Unix-like systems provide support via functions like `shm_open()`, `shmget()`, and `shmctl()`



Shared Memory with Threads

- A single process can be composed of multiple worker threads
- Threads are software analog of cores
  - Each thread has its own PC, SP, registers, etc
  - All threads share the process heap and the global data structures



Shared Memory with Threads

- All threads access the shared address space
  - Threads also have a private memory
- Synchronization is required to access shared resources
  - Can otherwise lead to pernicious bugs

- Runtime system schedules threads to cores
  - Concurrent units of execution
  - If there are more threads than cores, the runtime will time-slice threads on to the cores

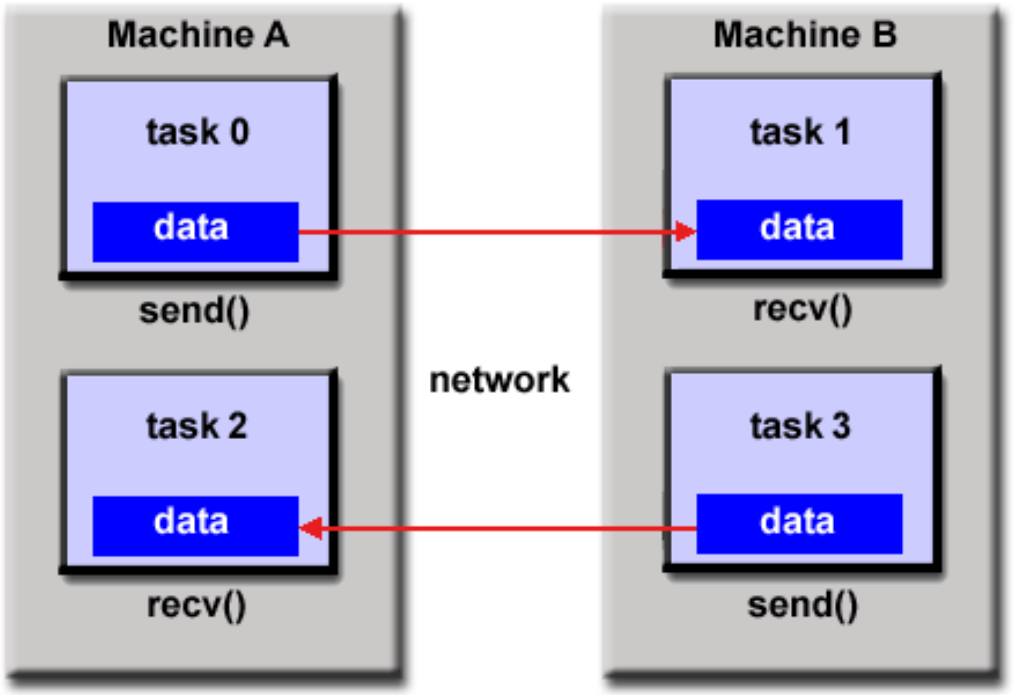
Distributed Memory

- The problem size may not fit on a single machine
  - Graph analytics
  - Obvious step: Go distributed!

- Distributed computing model
  - Launch multiple processes on multiple systems
  - Processes carry out work
  - Processes may communicate through message passing
  - Processes coordinate either through message passing or synchronization

Distributed Memory

- Also called message passing programming model
- Set of tasks that use local memory for computation
  - Exchange data via communication
- MPI is a popular runtime



Challenges with Distributed Memory

Often, communication turns out to be the primary bottleneck

- How do you partition the data between different nodes?
- Network topology is very important for scalability
  - Non-uniform memory access

Since communication is explicit, therefore it **excludes** race conditions

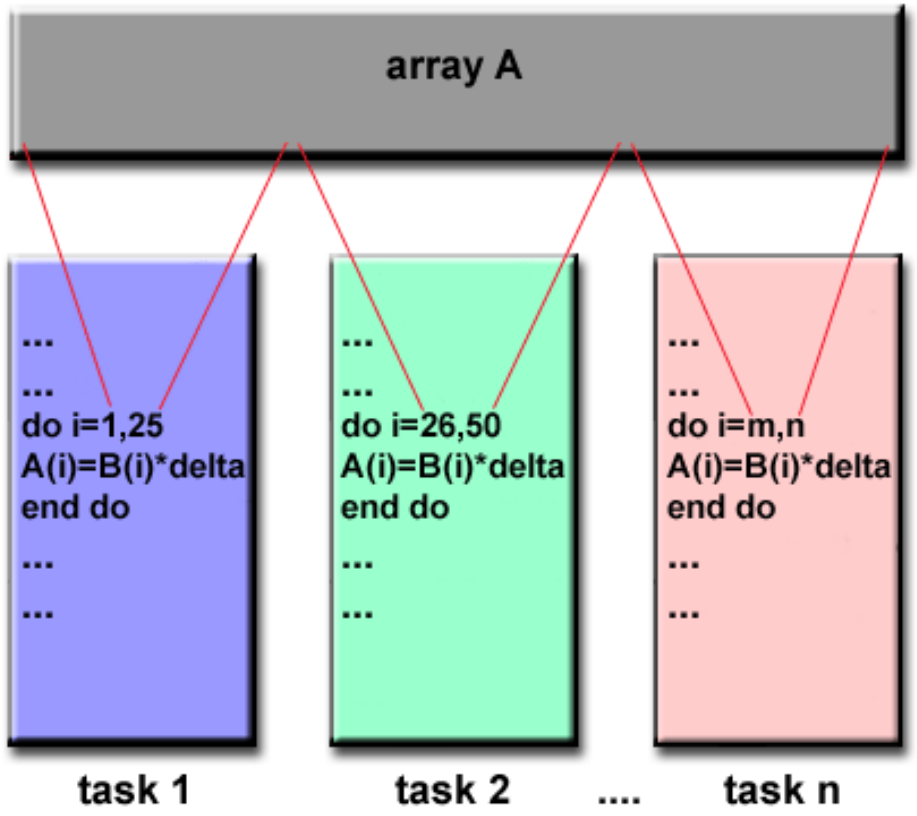
- Programmer's responsibility to synchronize between tasks

Shared Memory vs Distributed Memory Programming

Shared Memory	Distributed Memory
• Communication is implicit	• Communication via explicit messages
• Explicit synchronization	• Synchronization implicit via messages
• Requires hardware support for coherence	• Requires support for in-node coherence and network communication
• Lower development effort to begin with	• Higher development effort to begin with

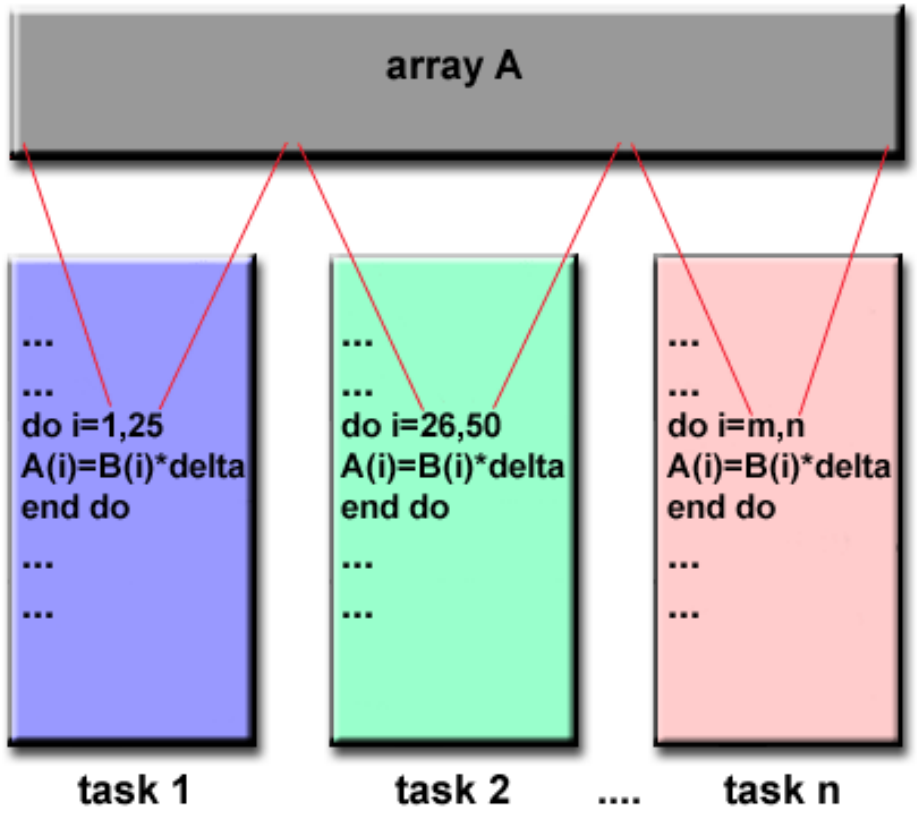
Data Parallel

- Also known as the partitioned global address space (PGAS)
  - Address space is global, and partitioned for tasks
  - Tasks operate on their own partition
  - Can have locality of reference
- Implementations
  - Unified Parallel C (UPC)
  - X10 from IBM
  - Chapel



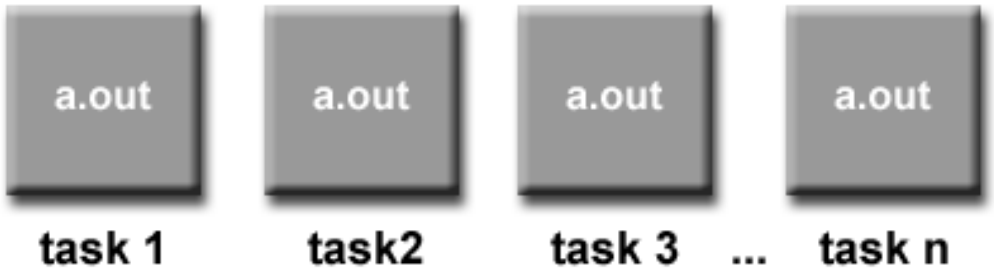
Data Parallel

- No library calls for communication
- Variables can name memory locations on other machines
  - Assume y points to a remote location
  - The following is equivalent to a send/receive
  - x = *y



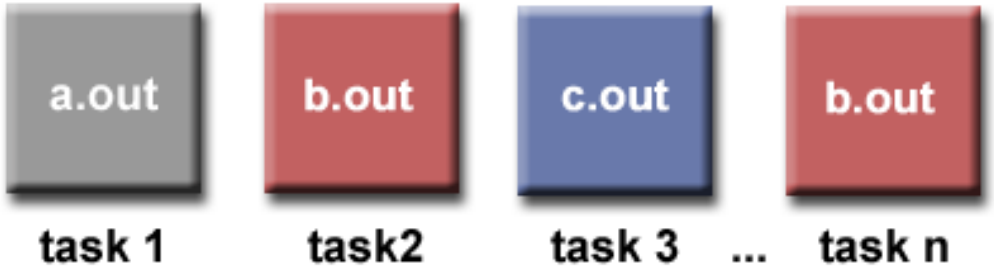
Single Program Multiple Data (SPMD)

- Tasks execute the same copy of the program on different data
  - Data parallel
  - Can be threads or message passing interface



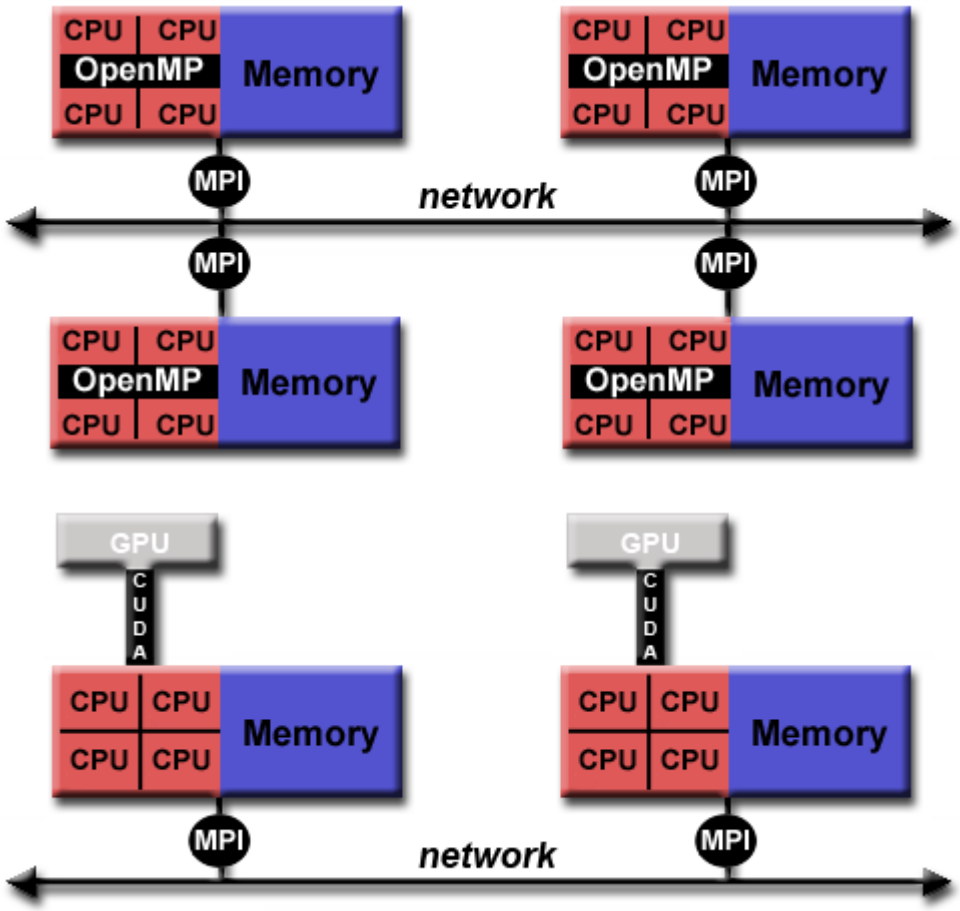
Multiple Program Multiple Data (MPMD)

- Tasks may execute different programs with different data



Hybrid

- Combine more than one of the other models
- Examples
  - Combine shared-memory on local nodes, and exchange data over networks
  - Use GPUs for compute kernels with CUDA for exchange between host and device, and MPI for inter-node communication



References

- James Demmel and Katherine Yelick – CS 267: Shared Memory Programming: Threads and OpenMP
- Keshav Pingali – CS 377P: Programming Shared-memory Machines, UT Austin.
- Blaise Barney, LLNL. Introduction to Parallel Computing, https://computing.llnl.gov/tutorials/parallel_comp/