Parallel Computing

Ruini Xue

School of Computer Science and Engineering

University of Electronic Science and Technology of China

2016

Outline

- Top500
- Motivation and Introduction
- Memory Architectures
- Ways of Exploiting Parallelism
- Parallel Programming Models
  - OpenMP
  - Message Passing Interface
  - GPGPU
  - MapReduce

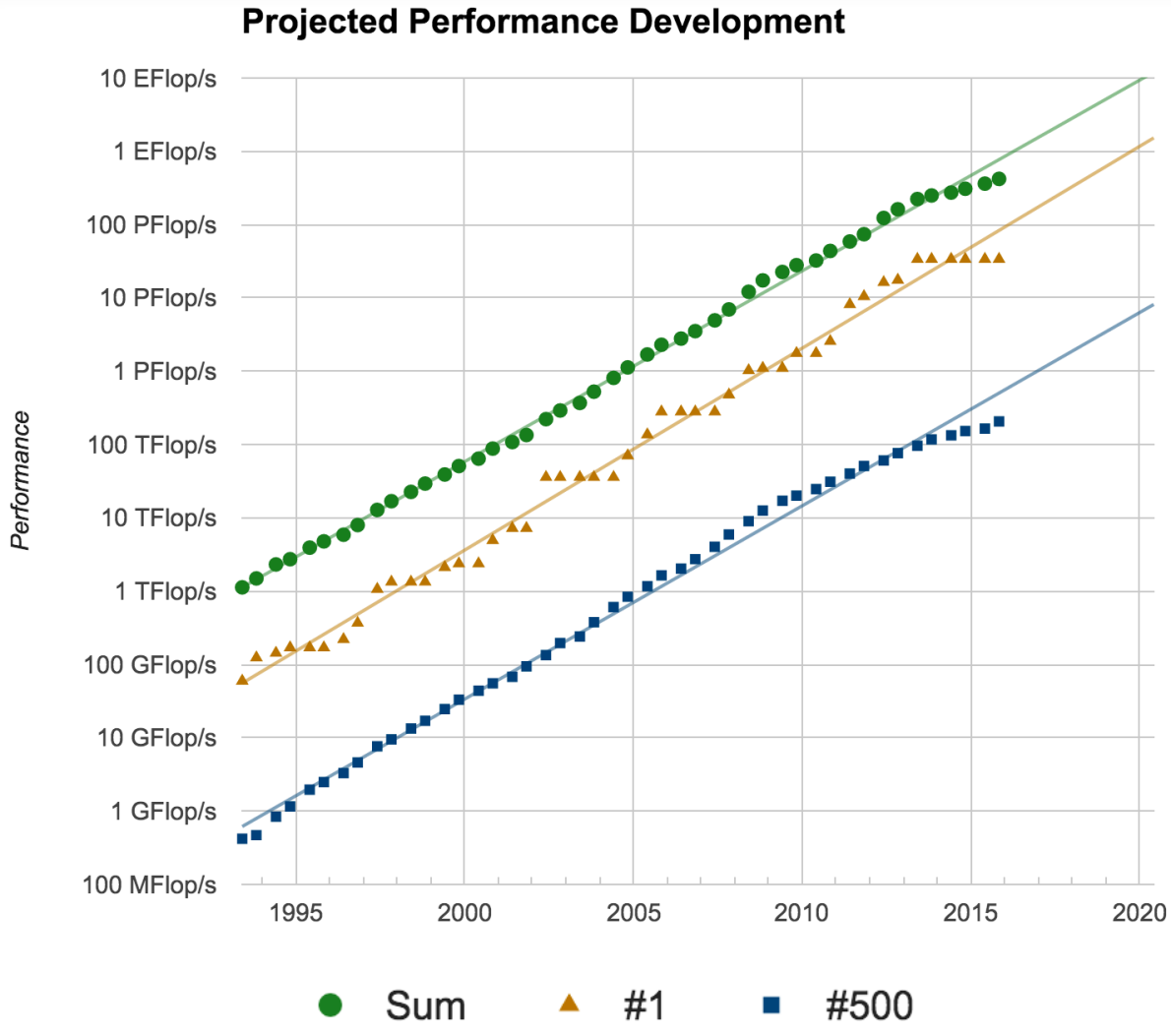
Supercomputers: a case study

- Characterized by very high speed, very large memory
- Speed measured in terms of number of floating point operations per second (FLOPS)
  - MegaFLOPS (MFLOPS) → 10^6
  - GigaFLOPS (GFLOPS) → 10^9
  - TeraFLOPS (TFLOPS) → 10^{12} (万亿)
  - PetaFLOPS (PFLOPS) → 10^{15} (千万亿)
  - Now: 10x PFLOPS → 10^{16} (万万亿)
- Memory in the order of hundreds of gigabytes or terabytes

top500.org

- The TOP500 project ranks and details the 500 most powerful (non-distributed) computer systems in the world.
- started in 1993 and publishes an updated list of the supercomputers twice a year.
- The first of these updates always coincides with the *International Supercomputing Conference (ISC)* in June, and the second is presented in November at the *ACM/IEEE Supercomputing Conference (SC).*

Performance Development



LINPACK

Original author(s)	Jack Dongarra, Jim Bunch, Cleve Moler, and Gilbert Stewart
Written in	Fortran
Type	Library
Website	www.netlib.org/linpack/

a software library for performing numerical linear algebra on digital computers.

Top 10 positions of the 46th TOP500 in November 2015

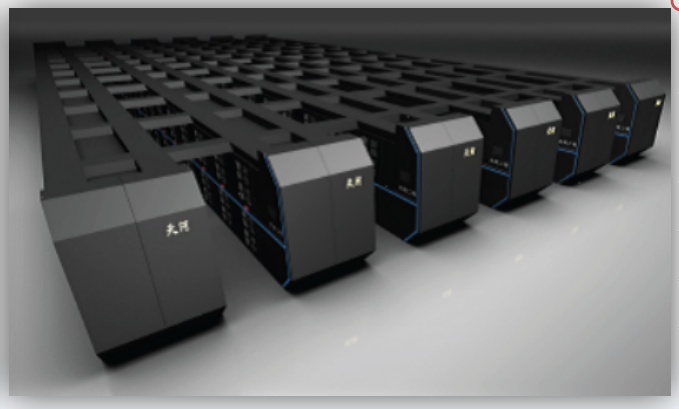
Rank	Rmax Rpeak (PFLOPS)	Name	Computer design Processor type, interconnect	Vendor	Site Country, year	Operating system
1	33.863 54.902	*Tianhe-2*	**NUDT** Xeon E5–2692 + Xeon Phi 31S1P, TH Express-2	NUDT	National Supercomputing Center in Guangzhou China, 2013	Linux (Kylin)
2	17.590 27.113	*Titan*	**Cray XK7** Opteron 6274 + Tesla K20X, Cray Gemini Interconnect	Cray Inc.	Oak Ridge National Laboratory United States, 2012	Linux (CLE, SLES based)
3	17.173 20.133	*Sequoia*	**Blue Gene/Q** PowerPC A2, Custom	IBM	Lawrence Livermore National Laboratory United States, 2013	Linux (RHEL and CNK)
4	10.510 11.280	*K computer*	**RIKEN** SPARC64 VIIIfx, Tofu	Fujitsu	RIKEN Japan, 2011	Linux
5	8.586 10.066	*Mira*	**Blue Gene/Q** PowerPC A2, Custom	IBM	Argonne National Laboratory United States, 2013	Linux (RHEL and CNK)
6	8.101 11.079	Trinity	**Cray XC40** Xeon E5-2698v3, Cray Aries Interconnect	Cray Inc.	DOE/NNSA/LANL/SNL United States, 2015	Linux (CLE)
7	6.271 7.779	*Piz Daint*	**Cray XC30** Xeon E5–2670 + Tesla K20X, Aries	Cray Inc.	Swiss National Supercomputing Centre Switzerland, 2013	Linux (CLE)
8	5.640 7.404	Hazel Hen	**Cray XC40** Xeon E5-2680v3, Cray Aries Interconnect	Cray Inc.	HLRS - Höchstleistungsrechenzentrum, Stuttgart Germany, 2015	Linux (CLE)
9	5.537 7.235	*Shaheen II*	**Cray XC40** Xeon E5–2698v3, Aries	Cray Inc.	King Abdullah University of Science and Technology Saudi Arabia, 2015	Linux (CLE)
10	5.168 8.520	*Stampede*	**PowerEdge C8220** Xeon E5–2680 + Xeon Phi, Infiniband	Dell	Texas Advanced Computing Center United States, 2013	Linux (CentOS)[13]

Systems ranked #1 since 1993

- NUDT Tianhe-2A (National Supercomputing Center of Guangzhou China, June 2013 - present)
- Cray Titan (Oak Ridge National Laboratory United States, November 2012 - June 2013)
- IBM Sequoia Blue Gene/Q (Los Alamos National Laboratory United States, June 2012 – November 2012)
- Fujitsu K computer (RIKEN Advanced Institute for Computational Science Japan, June 2011 – June 2012)
- NUDT Tianhe-1A (National Supercomputing Center of Tianjin China, November 2010 – June 2011)
- Cray Jaguar (Oak Ridge National Laboratory United States, November 2009 – November 2010)
- IBM Roadrunner (Los Alamos National Laboratory United States, June 2008 – November 2009)
- IBM Blue Gene/L (Lawrence Livermore National Laboratory United States, November 2004 – June 2008)
- NEC Earth Simulator (Earth Simulator Center Japan, June 2002 – November 2004)
- IBM ASCI White (Lawrence Livermore National Laboratory United States, November 2000 – June 2002)
- Intel ASCI Red (Sandia National Laboratories United States, June 1997 – November 2000)
- Hitachi CP-PACS (University of Tsukuba Japan, November 1996 – June 1997)
- Hitachi SR2201 (University of Tokyo Japan, June 1996 – November 1996)
- Fujitsu Numerical Wind Tunnel (National Aerospace Laboratory of Japan Japan, November 1994 – June 1996)
- Intel Paragon XP/S140 (Sandia National Laboratories United States, June 1994 – November 1994)
- Fujitsu Numerical Wind Tunnel (National Aerospace Laboratory of Japan Japan, November 1993 – June 1994)
- TMC CM-5 (Los Alamos National Laboratory United States, June 1993 – November 1993)

Tianhe-2

- 720 square meters of space
  - 2 basketball courts
- 16,000 computer nodes
- Total cores: 3,120,000
- 88 GB Memory / node
- Total memory: 1,375 TB (1.34 PB)

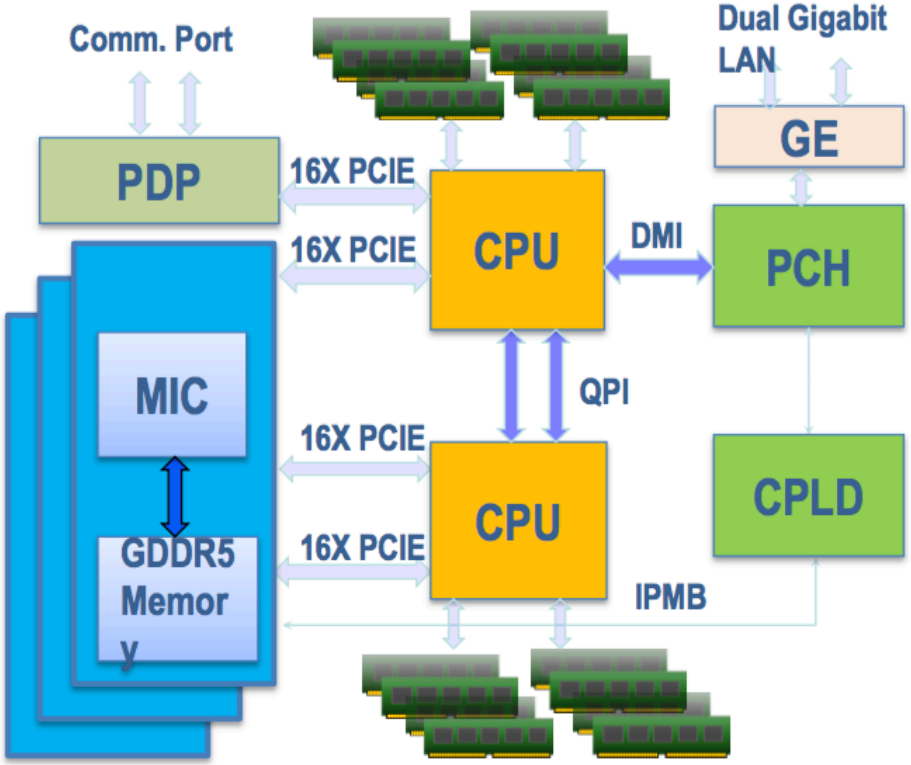


Tianhe-2

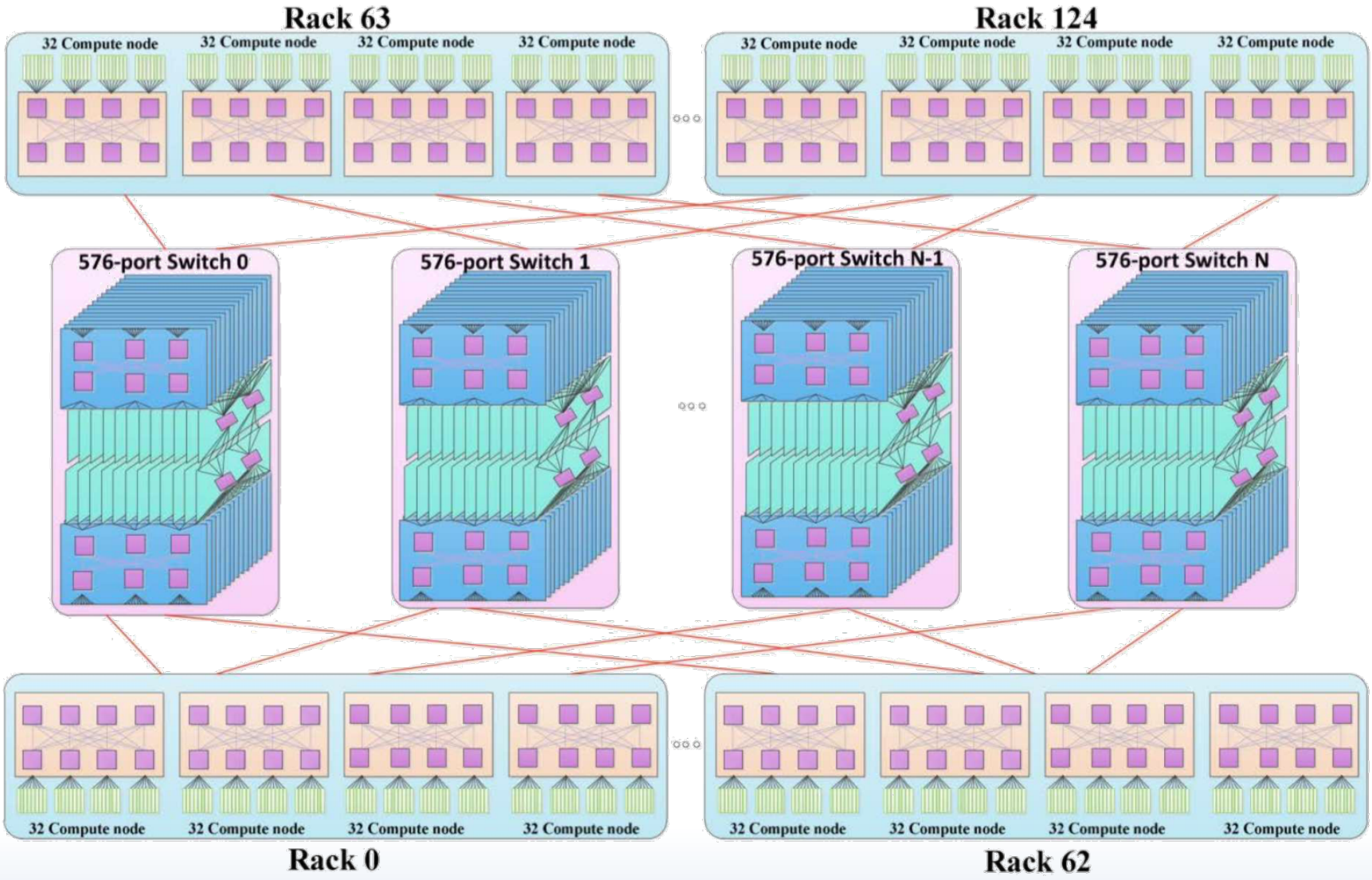
Sponsors	863 Program
Location	National Supercomputer Center, Guangzhou, China
Architecture	32,000 Intel Xeon E5-2692 12C with 2.200 GHz 48,000 Xeon Phi 31S1P
Power	17.6 MW (24 MW with cooling)
Operating system	Kylin Linux[1]
Memory	1,375 TiB (1,000 TiB CPU and 375 TiB coprocessor)[1]
Storage	12.4 PB
Speed	33.86 PFLOPS
Cost	2.4 billion Yuan (US$390 million)[2]
Purpose	Simulation, analysis, and government security applications.

Tianhe-2: computing node

- Similar ISA, different ALU
- 2 Intel Ivy Bridge CPU + 3 Intel Xeon Phi
- 16 Registered ECC DDR3 DIMMs, 64GB
- 3 PCI-E 3.0 with 16 lanes
- PDP Comm. Port
- Dual Gigabit LAN
- Peak Perf. : 3.432 Tflops



Tianhe-2: network



Supercomputing Centers @ DOE

Name	Focus

Supercomputing Centers @ NSF

Name	Field of research
National Center for Supercomputer Applications (NCSA)	Cyber-infrastructure, supercomputing, cyber-resources, cyber-environments, visualization
San Diego Supercomputer Center (SDSC)	high performance computing, grid computing, computational biology, geo-informatics, computational physics, computational chemistry, data management, scientific visualization, and computer networking.
Pittsburgh Supercomputing Center (PSC)	provides university, government, and industrial researchers with access to several of the most powerful systems for high-performance computing, communications and data-handling available to scientists and engineers nationwide for unclassified research.
Texas Advanced Computing Center (TACC)	enable discoveries that advance science and society through the application of advanced computing technologies

Motivation & Introduction

Why Parallel Computing?

Why Use Parallel Computing?

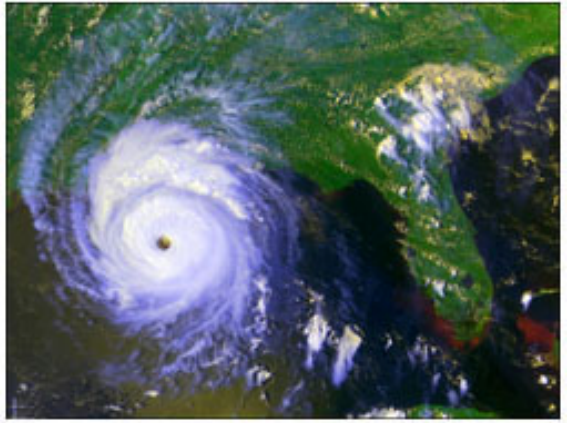
- The Real World is Massively Parallel
  - In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence.
  - Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena.

Why Use Parallel Computing?

- Solve problems faster → save time and/or money
  - Reduce turn-around time of big jobs
  - Increase responsiveness of interactive apps
- Solve larger / more complex problems
  - Increase resolution of models
  - Make model more sophisticated
  - Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory.
    - Example: "Grand Challenge Problems" requiring PetaFLOPS and PetaBytes of computing resources.
    - Example: Web search engines/databases processing millions of transactions every second

Galaxy Formation

Planetary Movments



Climate Change

Rush Hour Traffic

Plate Tectonics

Weather

Auto Assembly

Jet Construction

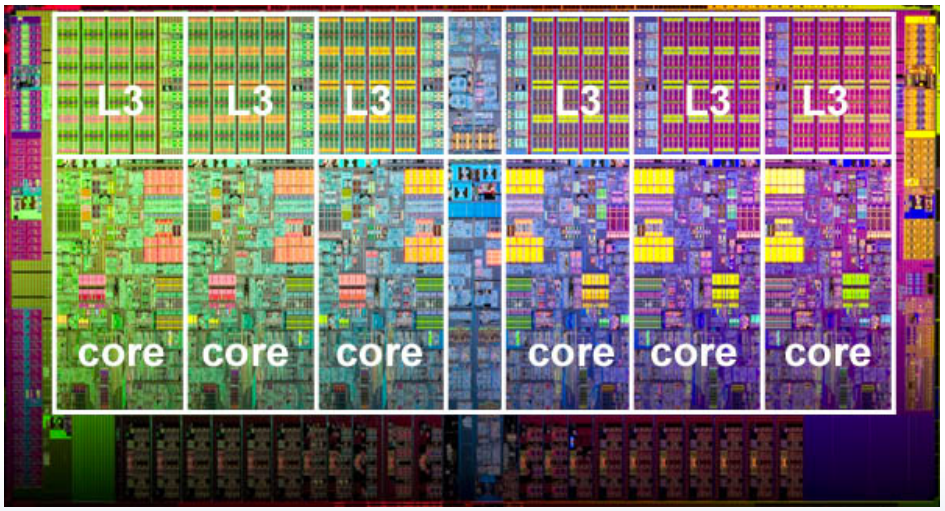
Drive-thru Lunch

Why Use Parallel Computing?

- Take advantage of non-local resources
  - Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient.
  - Example: SETI@home over 1.5 million users in nearly every country in the world (June, 2015).
  - Example: Folding@home uses over 160,000 computers globally (June, 2015)

Why Use Parallel Computing?

- Make better use of underlying parallel hardware
  - Modern computers, even laptops, are parallel in architecture with multiple processors/cores.
  - Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc.
  - In most cases, serial programs run on modern computers "waste" potential computing power.

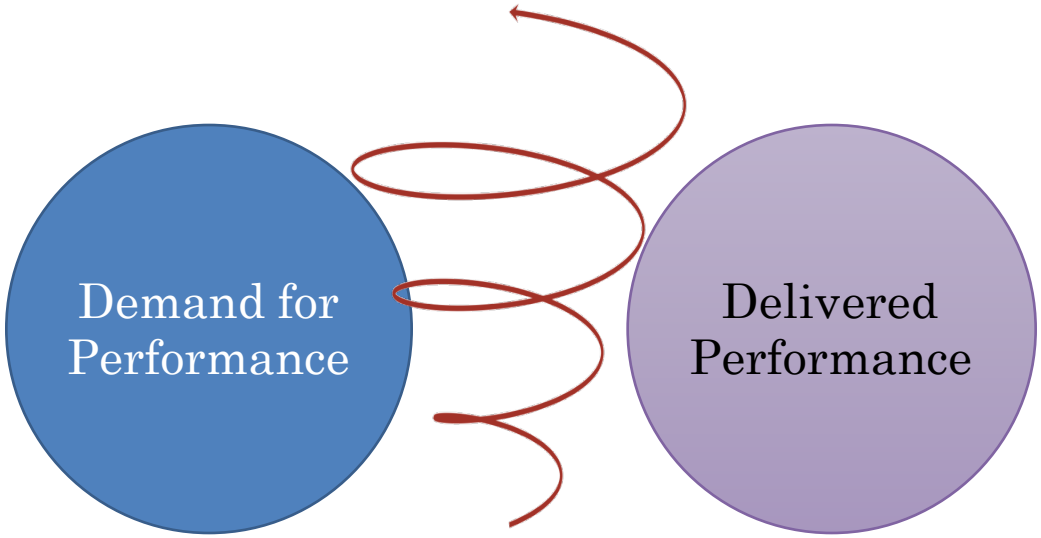


Intel Xeon processor with 6 cores and 6 L3 cache units

Actually, the most important reason is

Human Nature

better, faster, more…



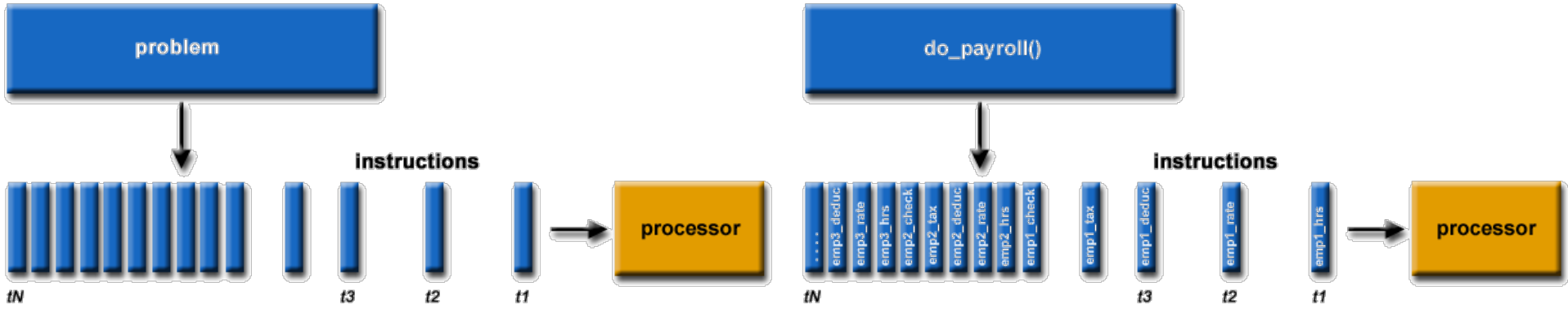
Arch. Trends: Exploiting Parallelism

- Greatest trend in VLSI generation increases in parallelism
- Up to 1985: bit level parallelism: 4-bit → 8 bit → 16-bit
  - slows after 32 bit
  - adoption of 64-bit in recent years, 128-bit far (not performance issue)
  - great inflection point when 32-bit micro and cache fit on a chip
- Mid 80s to mid 90s: instruction level parallelism
  - pipelining and simple instruction sets, + compiler advances (RISC)
  - on-chip caches and functional units => superscalar execution
  - greater sophistication: out of order execution, speculation, prediction
    - to deal with control transfer and latency problems
- 10s in 21st century: thread (core) level parallelism
  - Difficulty with thread programming
- Now: ?

Process level

Serial Computing

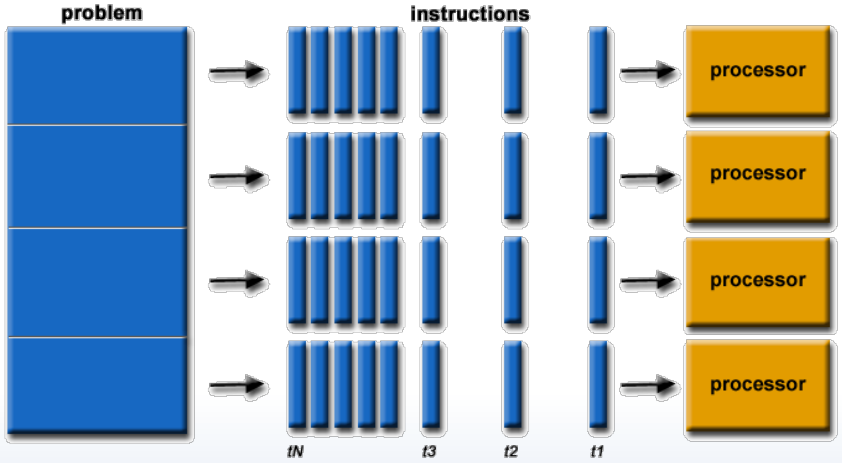
- Traditionally, software has been written for serial computation:
  - A problem is broken into a discrete series of instructions
  - Instructions are executed sequentially one after another
  - Executed on a single processor

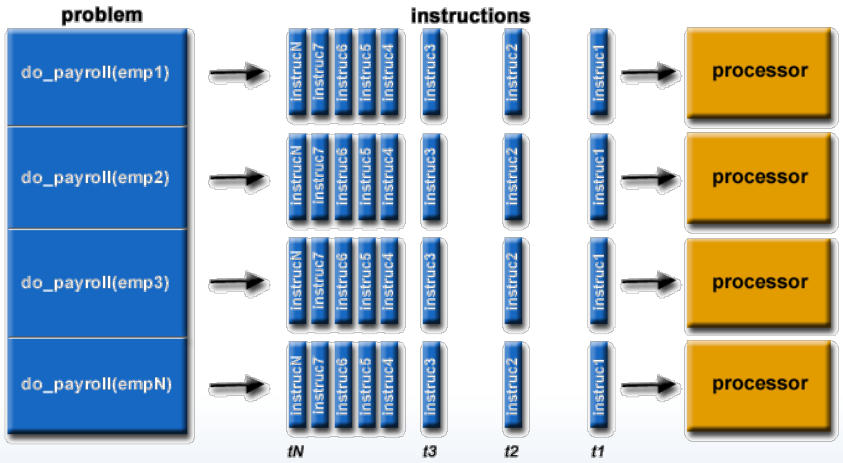


From: LLNL Parallel Computing Tutorial

Parallel Computing

- the simultaneous use of multiple compute resources to solve a computational problem
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
  - Instructions from each part execute simultaneously on different processors
  - An overall control/coordination mechanism is employed



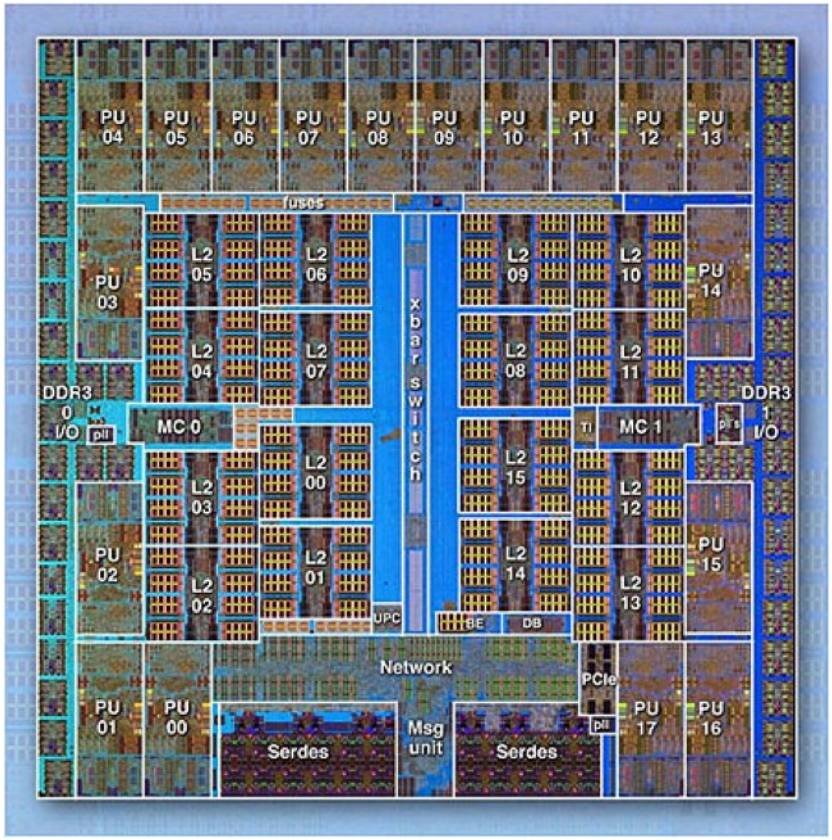


Parallel Computing

- The computational problem should be able to
  - Be broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Be solved in less time with multiple compute resources than with a single compute resource.
- BSP: Bulk Synchronous Parallel
- The compute resources are typically
  - A single computer with multiple processors/cores
  - An arbitrary number of such computers connected by a network

Parallel Computer

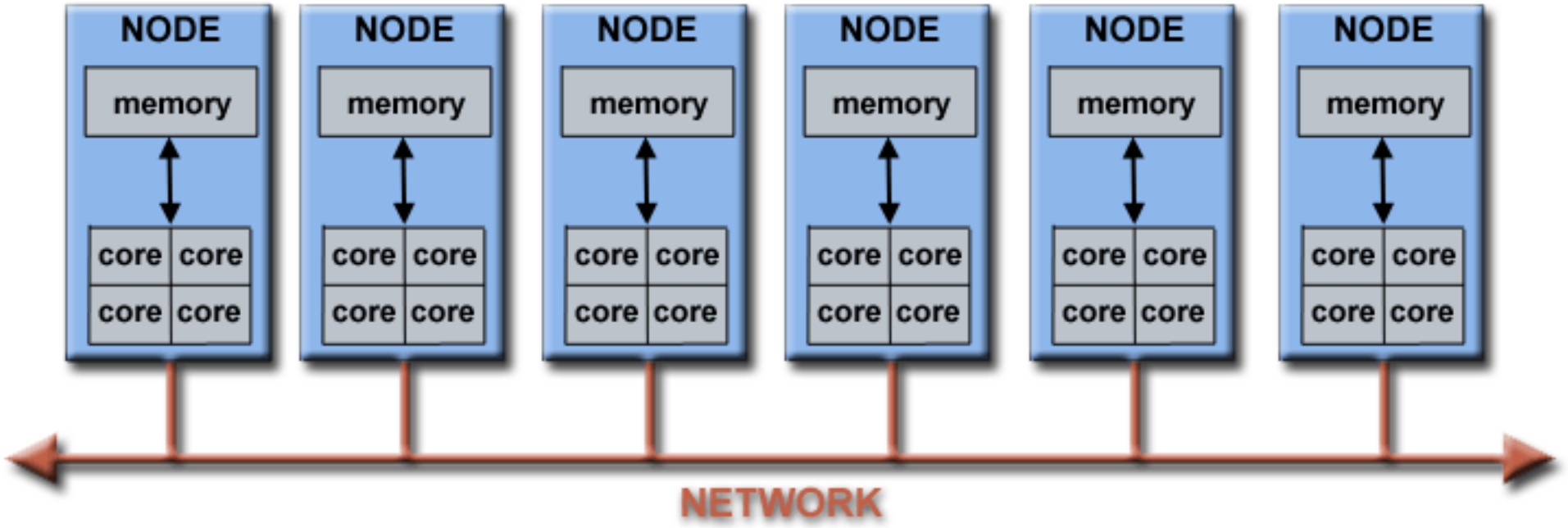
- A collection of processing elements that cooperate to solve large problems fast
- Virtually all stand-alone computers today are parallel from a hardware perspective:
  - Multiple functional units (L1/L2 cache, branch, prefetch, decode, floating-point, GPU, etc.)
  - Multiple execution units/cores
  - Multiple hardware threads



IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

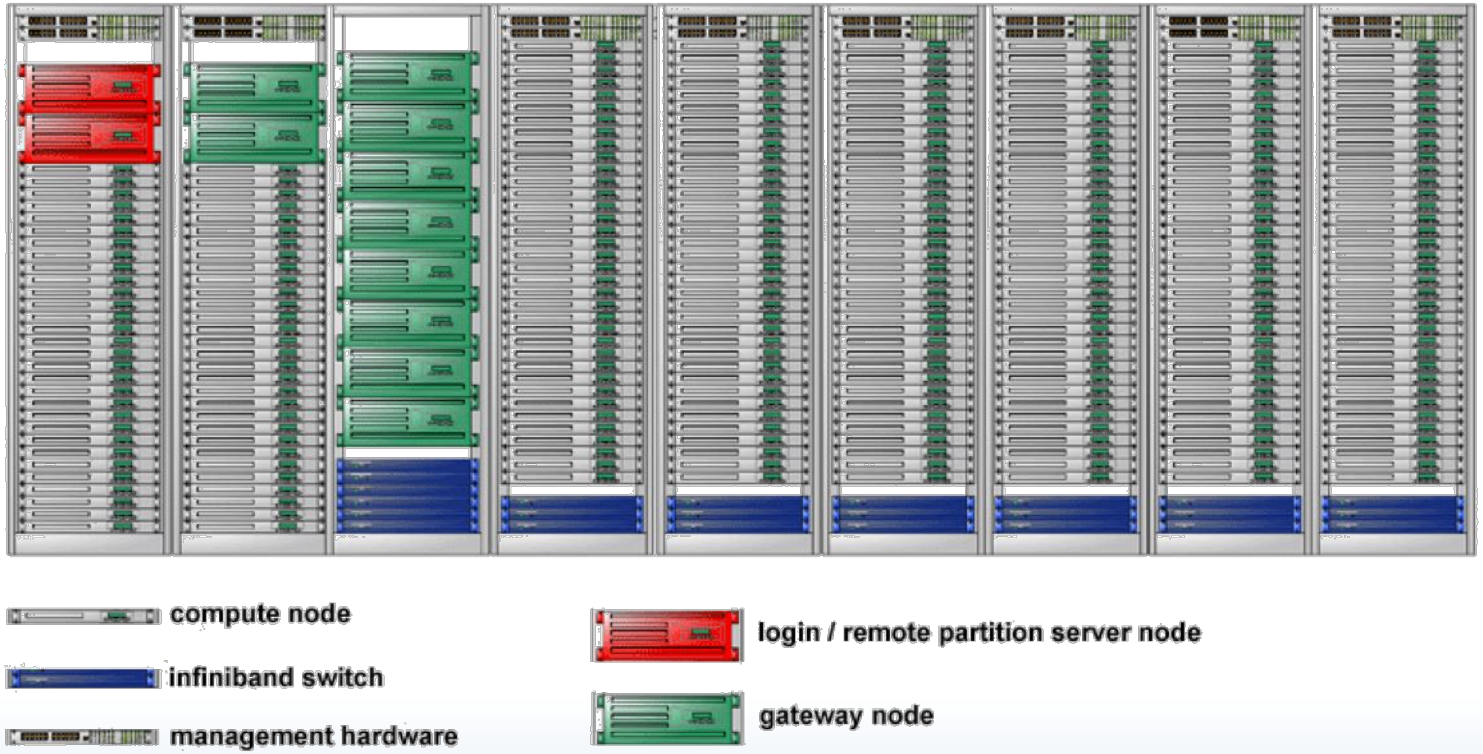
Parallel Computer

- Networks connect multiple stand-alone computers (nodes) to make larger parallel computer clusters.



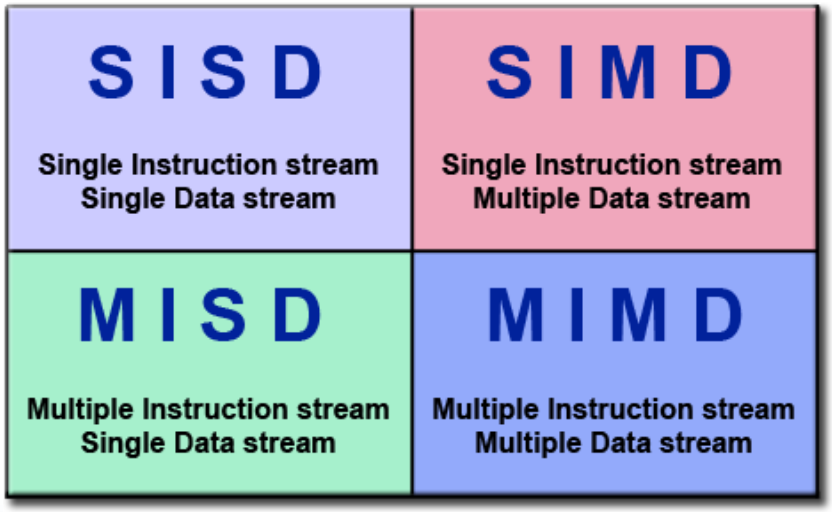
Example: LLNL Cluster

- Each compute node is a multi-processor parallel computer in itself
- Multiple compute nodes are networked together with an Infiniband network
- Special purpose nodes, also multi-processor, are used for other purposes



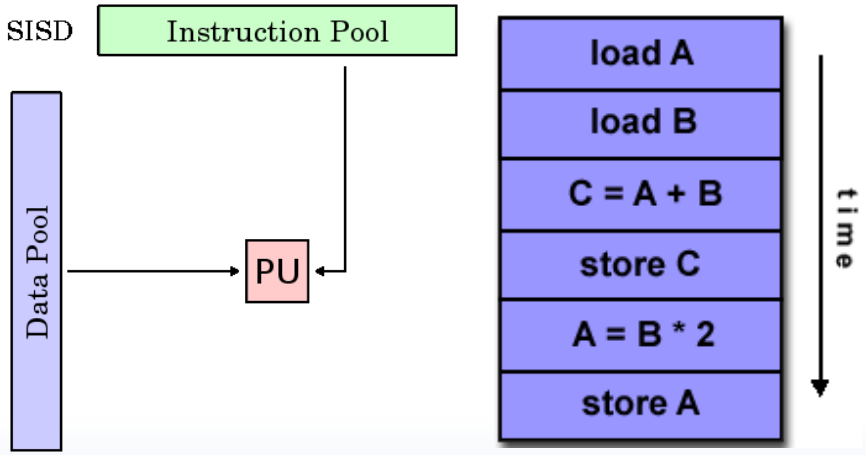
Arch. Classes

- Flynn's Taxonomy
  - distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of *Instruction Stream* and *Data Stream*.
  - Each of these dimensions can have only one of two possible states: *Single* or *Multiple*.



Single Instruction, Single Data (SISD)

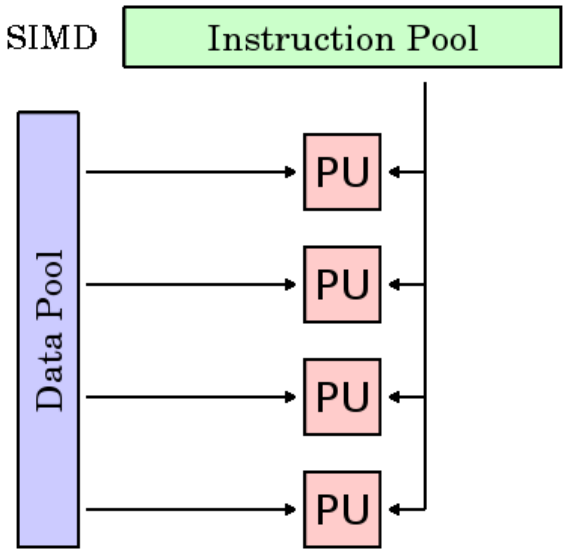
- A serial (non-parallel) computer
- Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle
- Single Data: Only one data stream is being used as input during any one clock cycle
- Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.

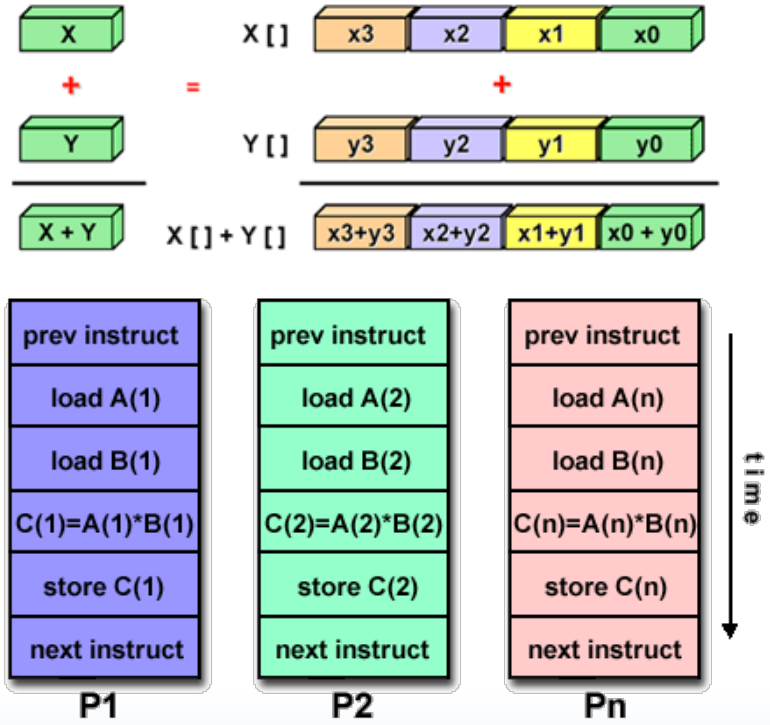


CRAY 1

Single Instruction, Multiple Data (SIMD)

- Single Instruction: all processing units execute the same instruction at any given clock cycle
- Multiple Data: each processing unit can operate on a different data element

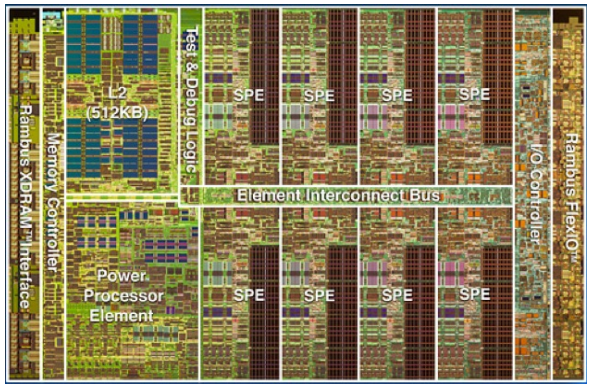




SIMD

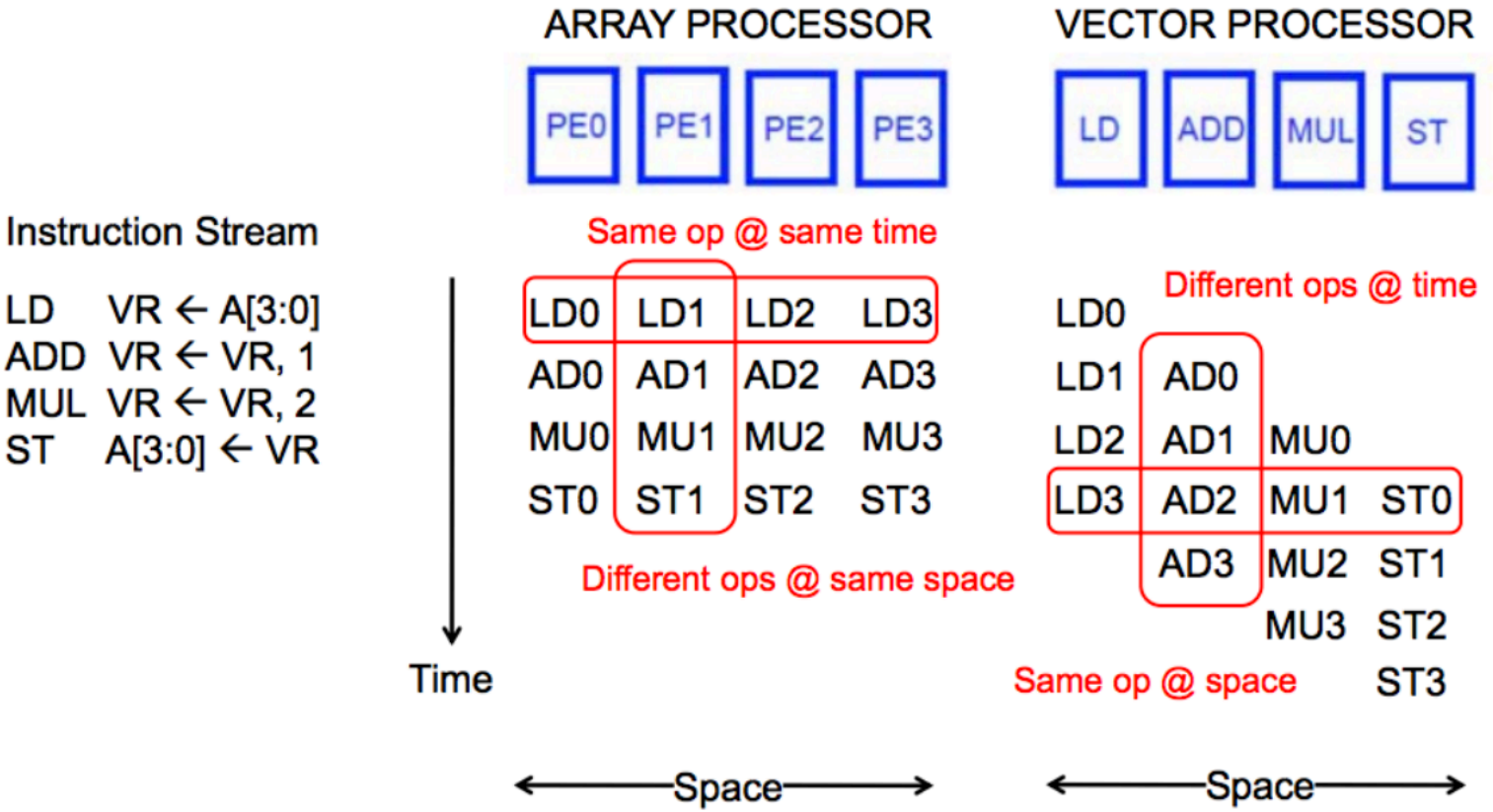
- Two varieties
  - Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV
  - Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
- Image/Video processing, MMX

Cray Y-MP



IBM Cell Processor

SIMD Array vs. Vector



Cray Inc.

- an American supercomputer manufacturer headquartered in Seattle, Washington.
- The most famous supercomputer manufacturer since its birth.

Seymour Cray
(1925-1996)

It seems impossible to exaggerate the effect he had on the industry; many of the things that high performance computers now do routinely were at the farthest edge of credibility when Seymour envisioned them."

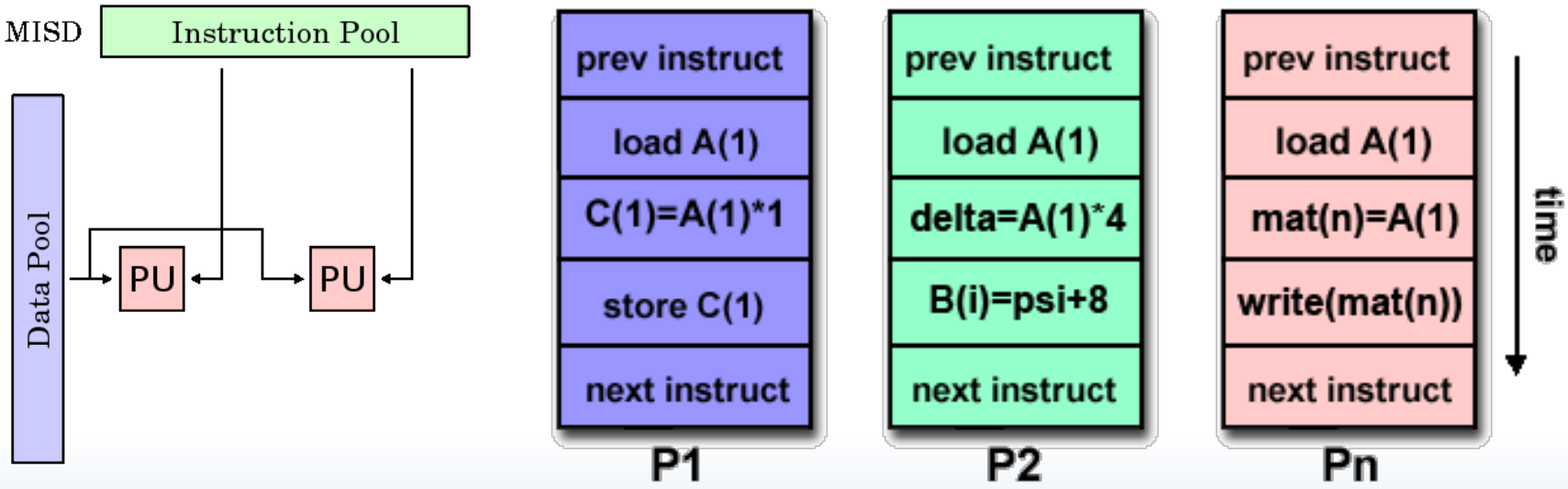
——Joel Birnbaum (HP CTO)

CRAY

Type	Public
Traded as	NASDAQ: CRAY
Founded	1972 as Cray Research, Inc.
Founder	Seymour Cray
Headquarters	Seattle, Washington United States
Key people	Peter J. Ungaro
Products	Supercomputers
Revenue	▲ US$ 561.6 million (2014)[1]
Operating income	▼ US$ 9.2 million (2014)[1]
Net income	▼ US$ 62.3 million (2014)[1]
Total assets	▲ US$ 651.4 million (2014)[1] ▲ US$ 603.366 million (2013) [2] ▲ US$ 510.314 million (2012) [2]
Total equity	▲ US$ 453.9 million (2014)[1]
Number of employees	1,148 (Dec 2014)[1]
Website	cray.com

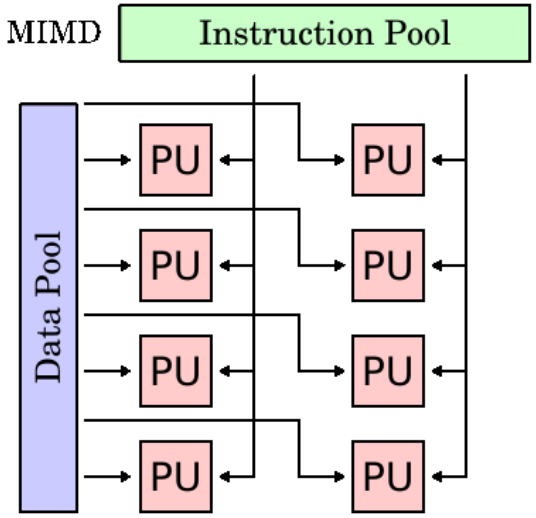
Multiple Instruction, Single Data (MISD)

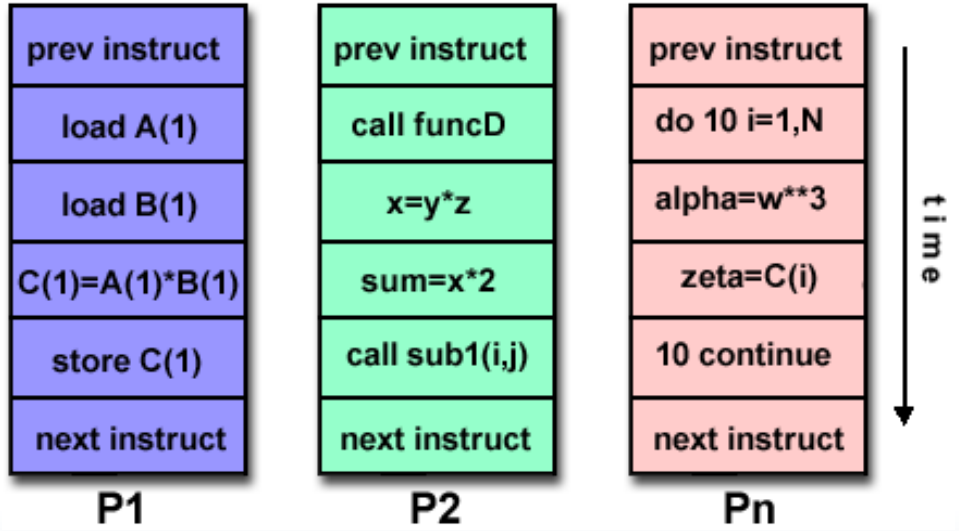
- Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
- Single Data: A single data stream is fed into multiple processing units.
- Few (if any) actual examples of this class of parallel computer have ever existed



Multiple Instruction, Multiple Data (MIMD)

- Multiple Instruction: Every processor may be executing a different instruction stream
- Multiple Data: Every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic





MIMD

- Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
- Examples: most current supercomputers, networked parallel computer clusters and “grids”, multi-processor SMP computers, multi-core PCs.

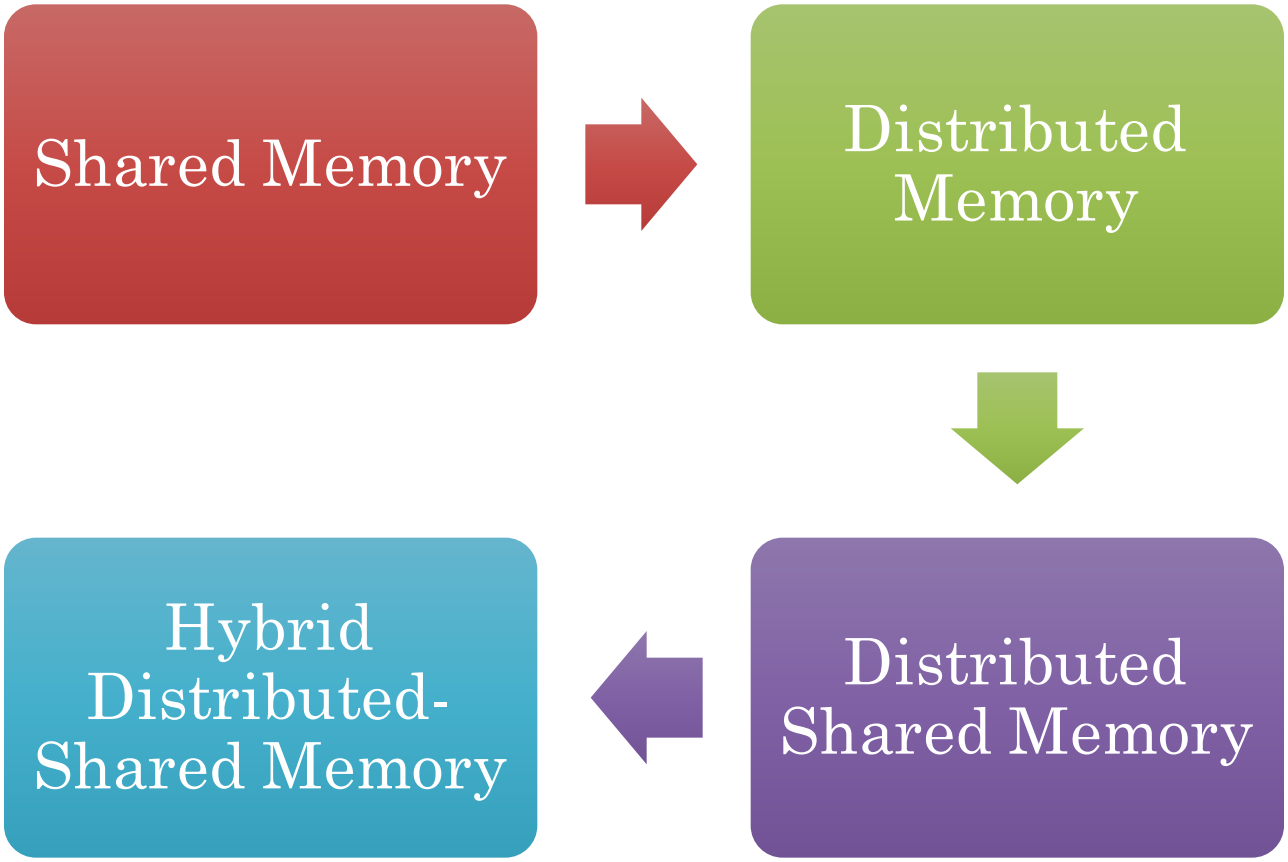
HP/Compaq Alphaserver

IBM BG/L

Cray XT-3

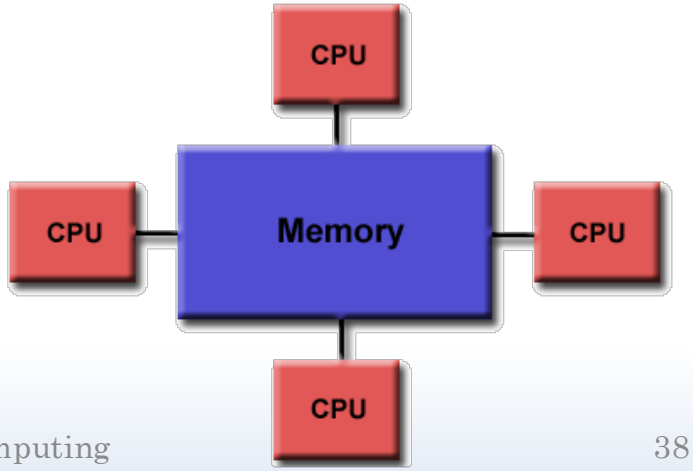
Parallel Computer Memory Architectures

Four Types


Shared Memory
Distributed Memory
Distributed Shared Memory
Hybrid Distributed-Shared Memory

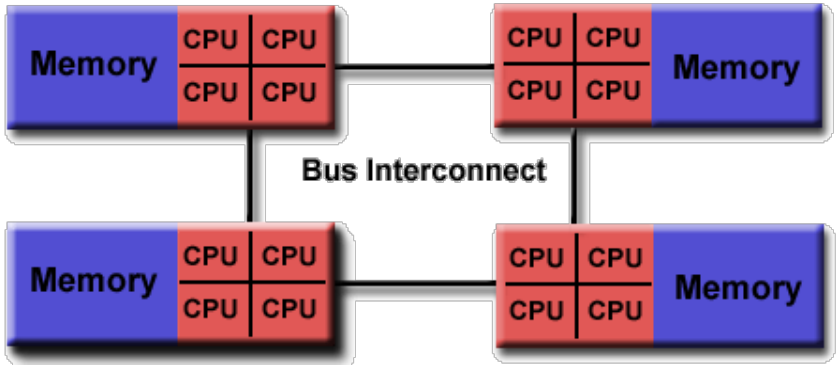
Shared Memory

- All processors share the same ***memory address (and inherently, address space)***
- Uniform Memory Access (UMA)
  - Symmetric Multiprocessor (SMP); typical multi-core CPU
  - Identical processors, equal access times to memory
  - Sometimes called CC-UMA - Cache Coherent UMA.
    - if one processor updates a location in shared memory, all the other processors know about the update → accomplished at the hardware level.



Non-Uniform Memory Access (NUMA)

- Often made by physically linking two or more SMPs
  - One SMP can directly access memory of another SMP
  - Not all processors have equal access time to all memories
  - Memory access across link is slower
  - If cache coherency is maintained, then may also be called CC-NUMA

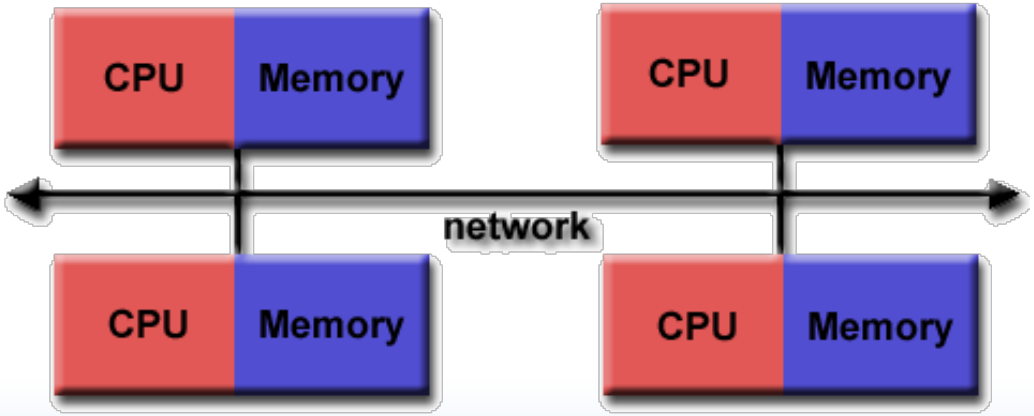


Shared Memory

- Advantages
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
- Disadvantages
  - lack of scalability between memory and CPUs.
    - Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
  - Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

Distributed Memory

- Local private memory for each processor → memory addresses in one processor do not map to another processor → *no concept of global address space*
  - computational tasks can only operate on local data (memory)
  - require a communication network to connect inter-processor memory.
  - local update → cache coherency does not apply.

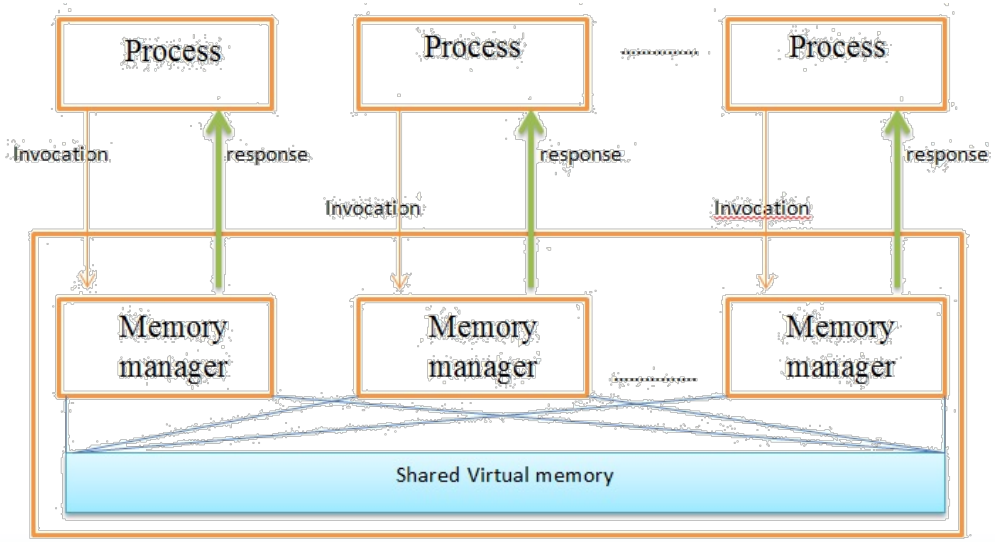


Distributed Memory

- Advantages
  - Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately.
  - Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- Disadvantages
  - The programmer is responsible for many of the details associated with data communication between processors.
  - It may be difficult to map existing data structures, based on global memory, to this memory organization.
  - Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.

Distributed Shared Memory (DSM)

- A form of memory architecture, the ***physically separate*** memories can be addressed as one ***logically shared address space***.
- it is shared memory, NUMA.
- Programming model is the same as shared memory.



DSM——Advantages

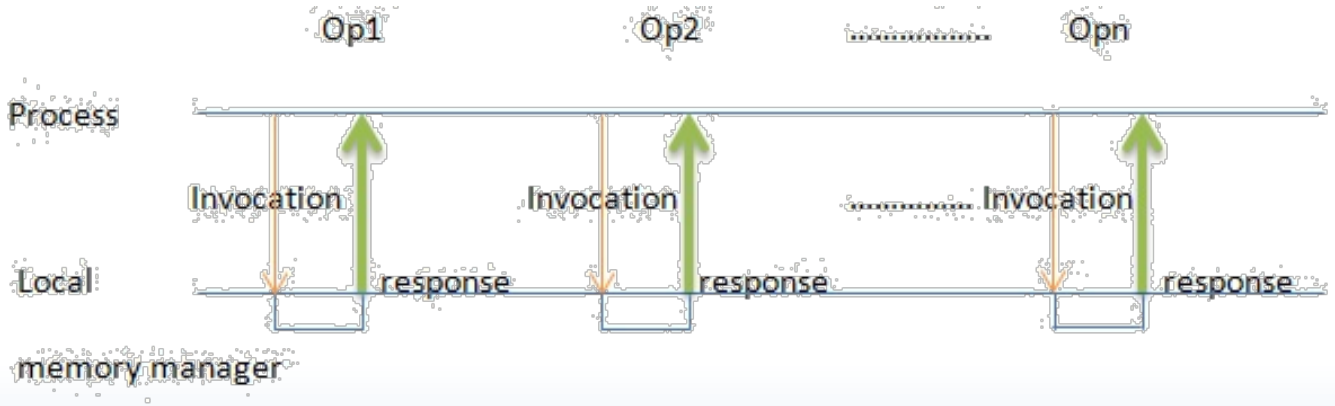
- Hides the message passing
- Can handle complex and large data bases without replication or sending the data to processes
- DSM provides large virtual memory space
- DSM programs portable as they use common programming interface
- Shields programmer from sending or receive primitives
- DSM can (possibly) improve performance by speeding up data access.

DSM——Issues

- Could cause a performance penalty
  - Like framework, difficult to optimize
- Should provide for protection against simultaneous access to shared data such as lock.
- Reduce delays → Data is replicated or cached
- Semantics for concurrent access must be clearly specified
- Distributed Shared Memory is controlled by memory management software, Operating system, language run-time system
- Granularity: how much data is transferred in a single operation?

Memory Coherence

- Suppose we have *n* processes and *Mi* memory operations for each process *Pi*, and all the operations are executed sequentially → *how many possible interleavings?*
- MC defines *which of these are correct and which ones are wrong*.
- Your solution?



Memory Model

- A memory model provides a formal specification of the effect of read and write operations on the memory system and describes how memory appears to the programmer.
- Memory model affects:
  - Programmability (easy-of-programming)
  - Performance (optimizations that it allows, out-of-order)
  - Portability (moving software across different systems)

Why it matters?

- x = y = 0
- No reordering
  - For thread 2, r1 = 2 → r2 = 1
- If thread 1 allows reordering,
  - For thread 2, r1 =2, r2 = 0

Thread 1	Thread 2
x = 1;	r1 = y;
y = 2;	r2 = x;

Strict Consistency

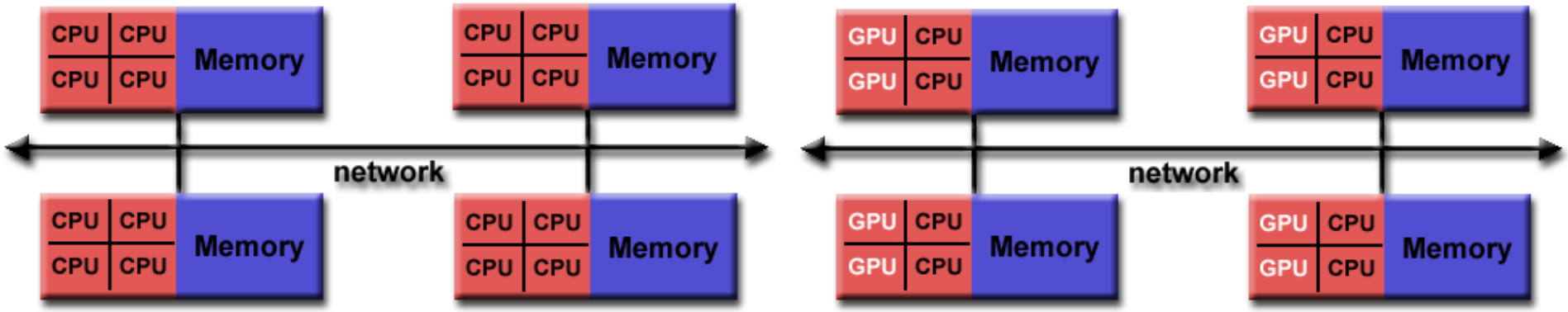
- Each operation is stamped with a global wall-clock time
- Rules:
  - *Rule 1: Each read gets the latest written value.*
  - *Rule 2: All operations at one CPU are executed in order of their time stamps.*
- Maintain cache coherence by broadcasting writes to all processors, which can then either update or invalidate their local caches.
- Too strict (too much overhead) to implement, so we need solutions for real-world.

Sequential Consistency

- If all operations of the processors were executed in sequential order, the result of any execution would be the same and each processor's operation appears in this sequence in the local program order.
- Weaker than strict consistency (no real-time force)
- Rules
  - Rule 1: Each machine's own ops appear in order
  - Rule 2: All machines see results according to total order. (*may interleave*)
- *What is that? Intuition.*

Hybrid Distributed-Shared Memory

- DSM is the one of the hybrid solutions
- Shared + distributed memory : the mainstream for HPC/SC
  - Almost all top500 machines.
- Two schemes.



HDSM (not formal notation)

- Whatever is common to both shared and distributed memory architectures.
- Increased scalability is an important advantage.
- Increased programmer complexity is an important disadvantage.
  - Who has programming experience with GPGPU? CUDA, or OpenCL?

Parallel vs Distributed Computing

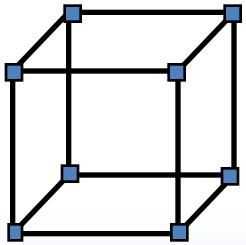
- Parallel computing splits a single application up into tasks that are executed at the same time and is more like a top-down approach
- Parallel computing is about decomposition
  - how to divide a computation into smaller parts which may potentially be executed in parallel
- Parallel computing consider how to reach a maximum degree of concurrency
  - Scientific computing

Message passing vs DSM

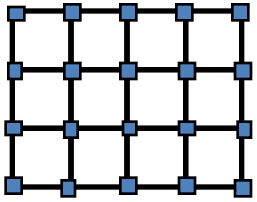
Message passing	DSM
Variables have to be marshalled	Variables are shared directly
Cost of communication is obvious	Cost of communication is invisible
Processes are protected by having private address space	Processes could cause error by altering data
Processes should execute at the same time	Executing the processes may happen with non-overlapping lifetimes

Interconnect Method

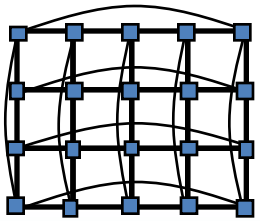
- There is no single way to connect bunch of processors
- The manner in which the nodes are connected - Network & Topology
- Best choice would be a fully connected network (every processor to every other). Unfeasible for cost and scaling reasons : Instead, processors are arranged in some variation of a grid, torus, tree, bus, mesh or hypercube.



3-d hypercube



2-d mesh



2-d torus

Ways of Exploiting Parallelism

Types of parallelism

- Control parallelism (Algorithmic parallelism):
  - Different portions (or subroutines/functions) can execute independently and concurrently
  - E.g. matrix multiplication
- Data parallelism
  - Data can be split up into multiple chunks and processed independently and concurrently
  - Most scientific applications exhibit data parallelism
  - E.g., sum()

Exploit Parallelism

- Domain decomposition
- Task decomposition
- Pipelining

Domain Decomposition

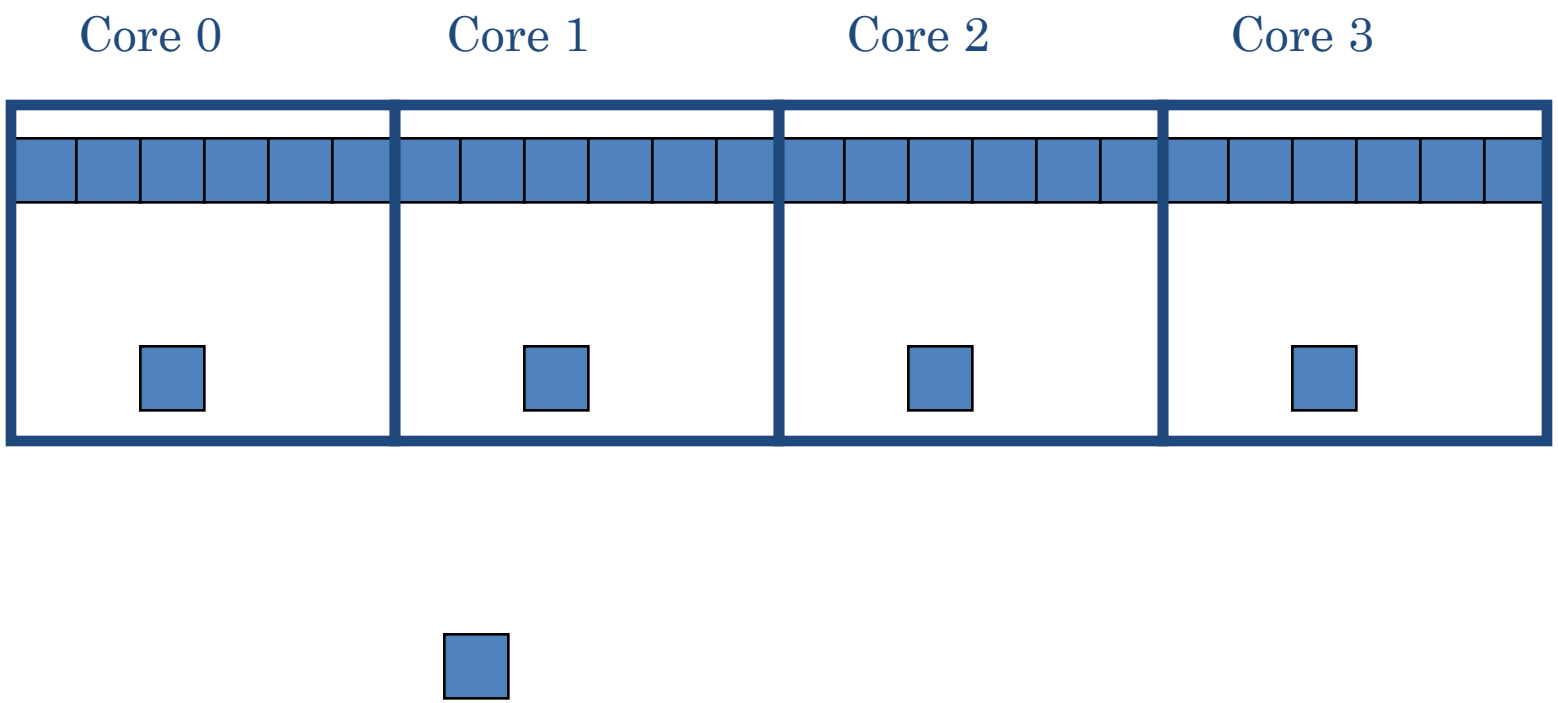
- First, decide how data elements should be divided among cores
- Second, decide which tasks each core should be doing
- Example: Vector addition

Domain Decomposition

Find the largest element of an array

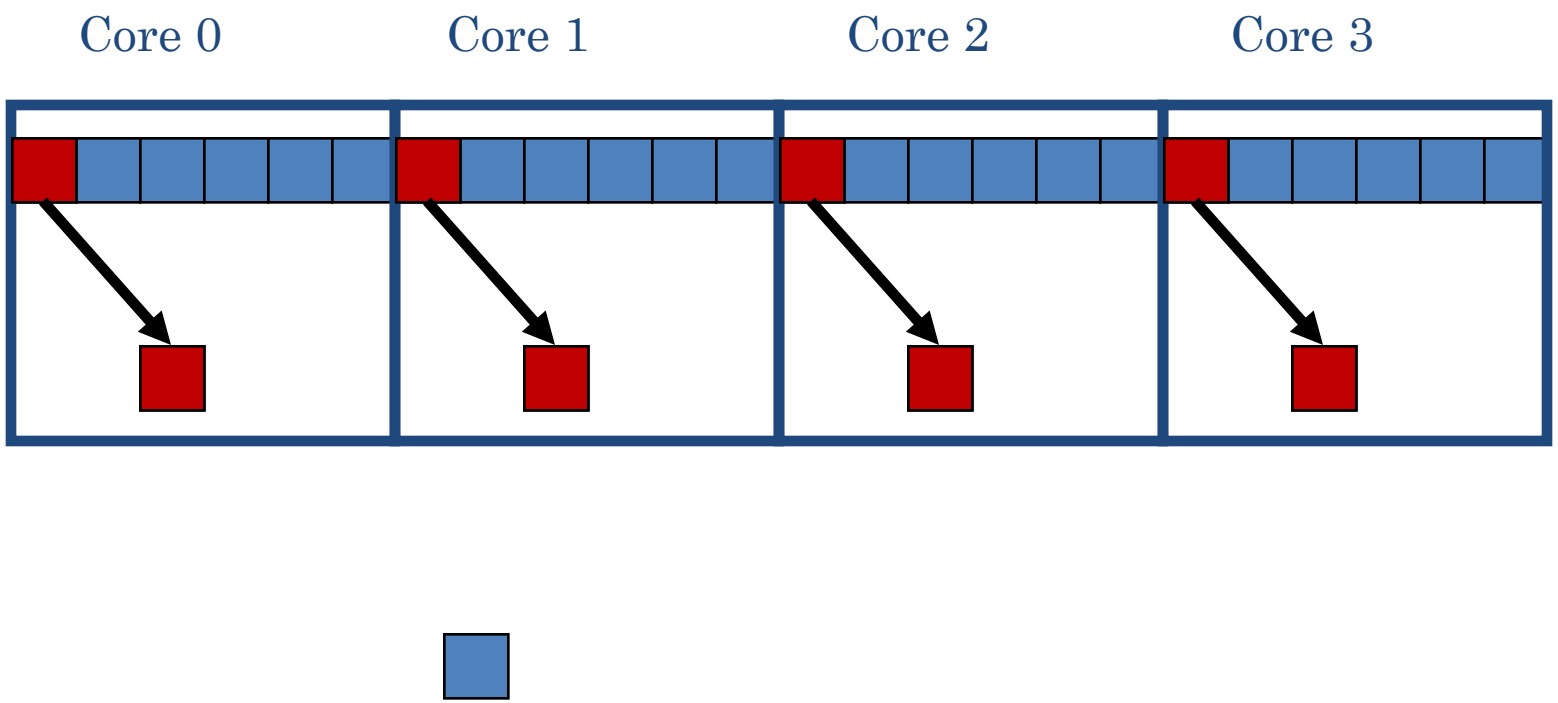
Domain Decomposition

Find the largest element of an array



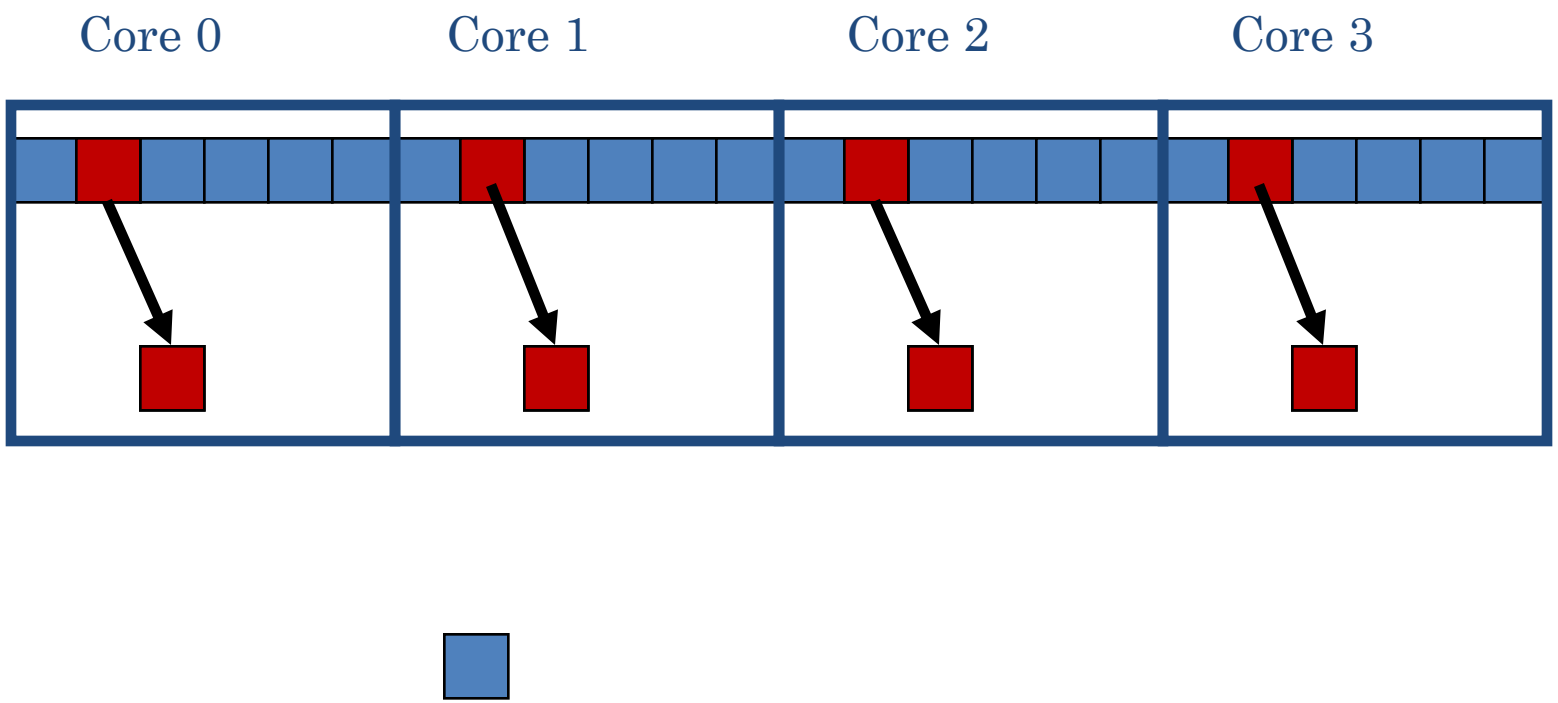
Domain Decomposition

Find the largest element of an array



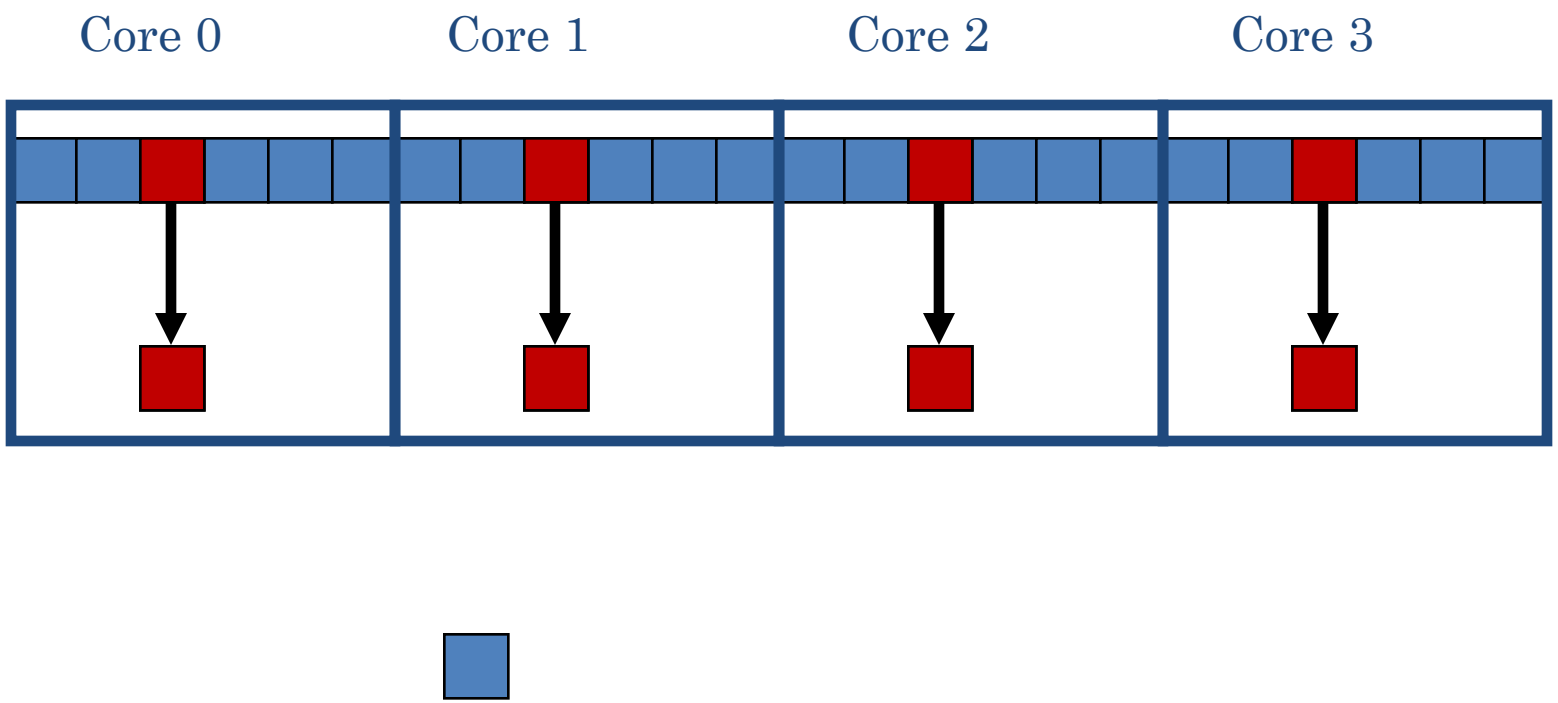
Domain Decomposition

Find the largest element of an array



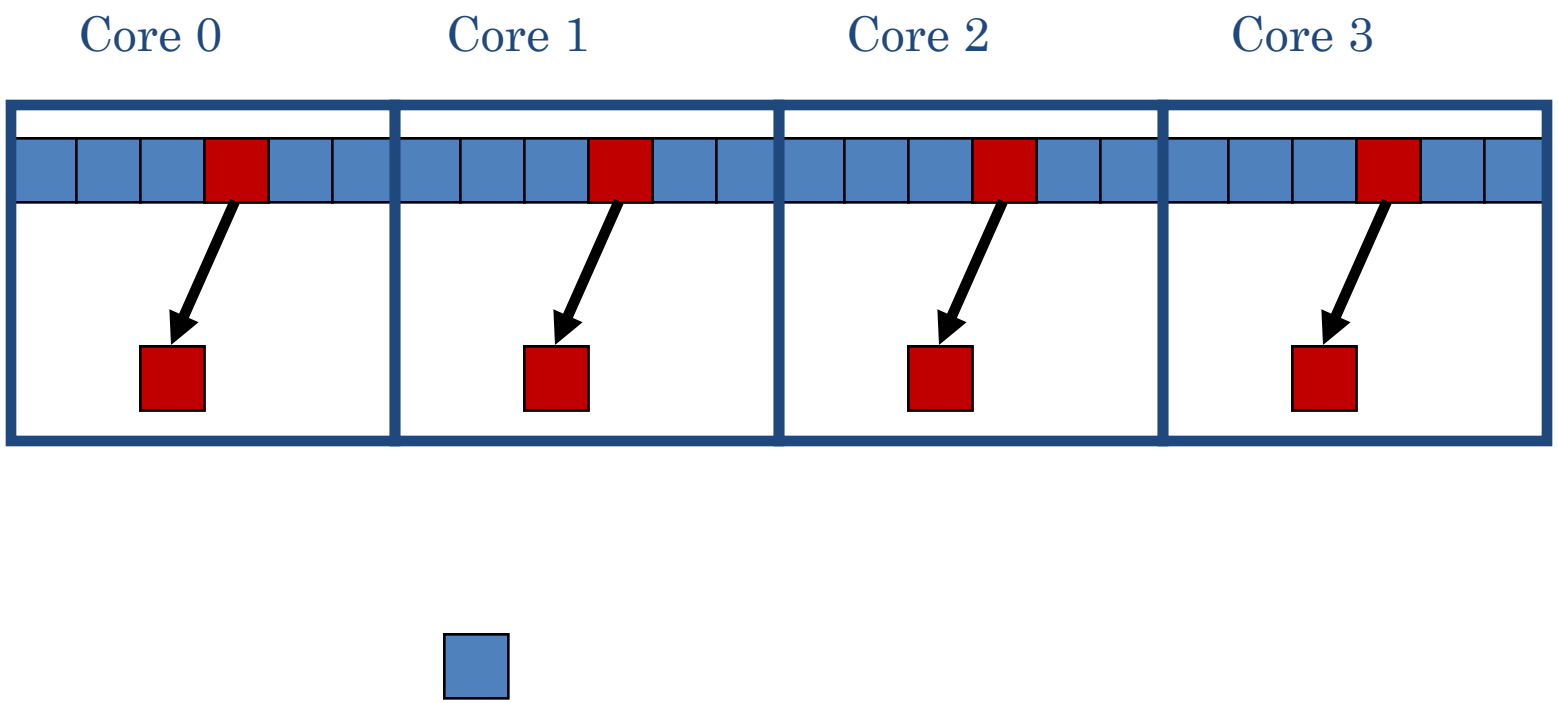
Domain Decomposition

Find the largest element of an array



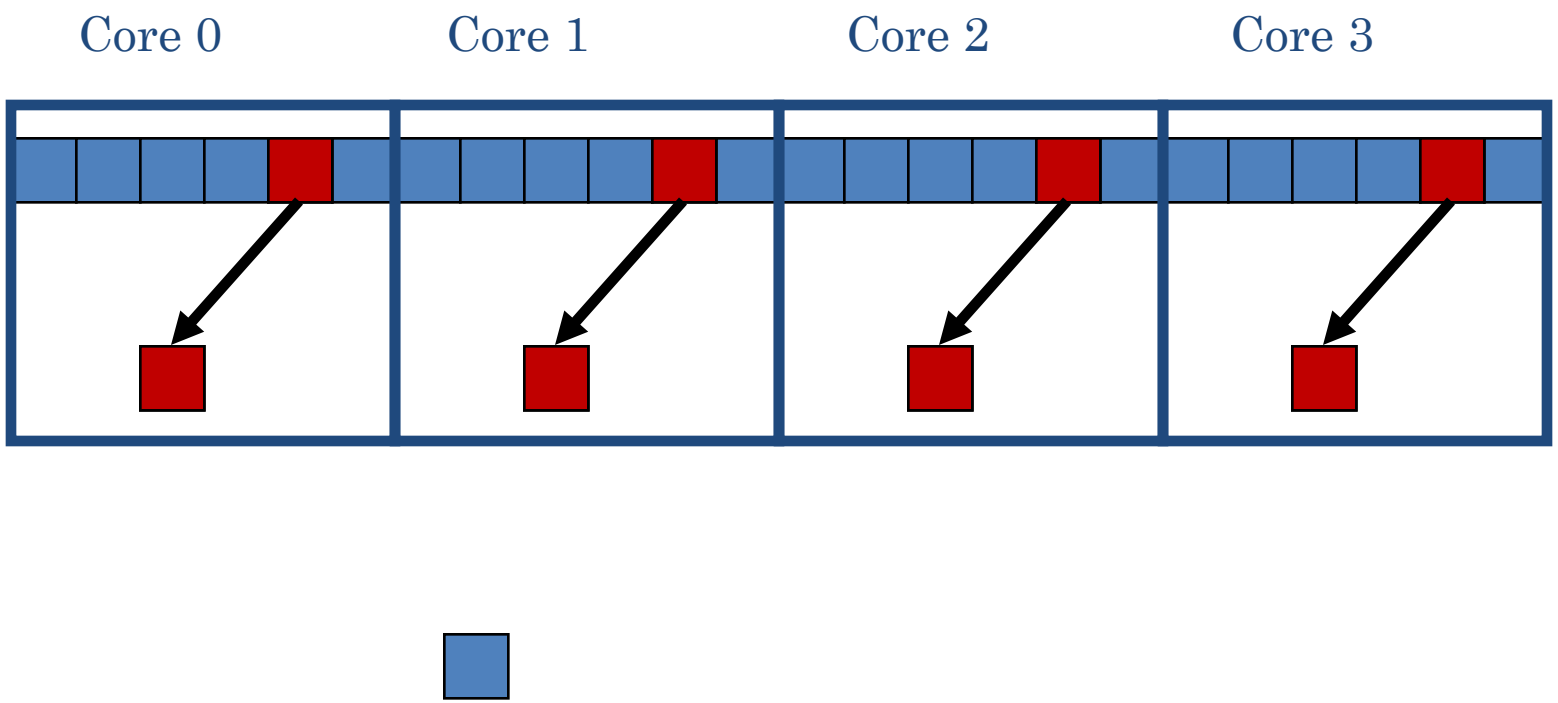
Domain Decomposition

Find the largest element of an array



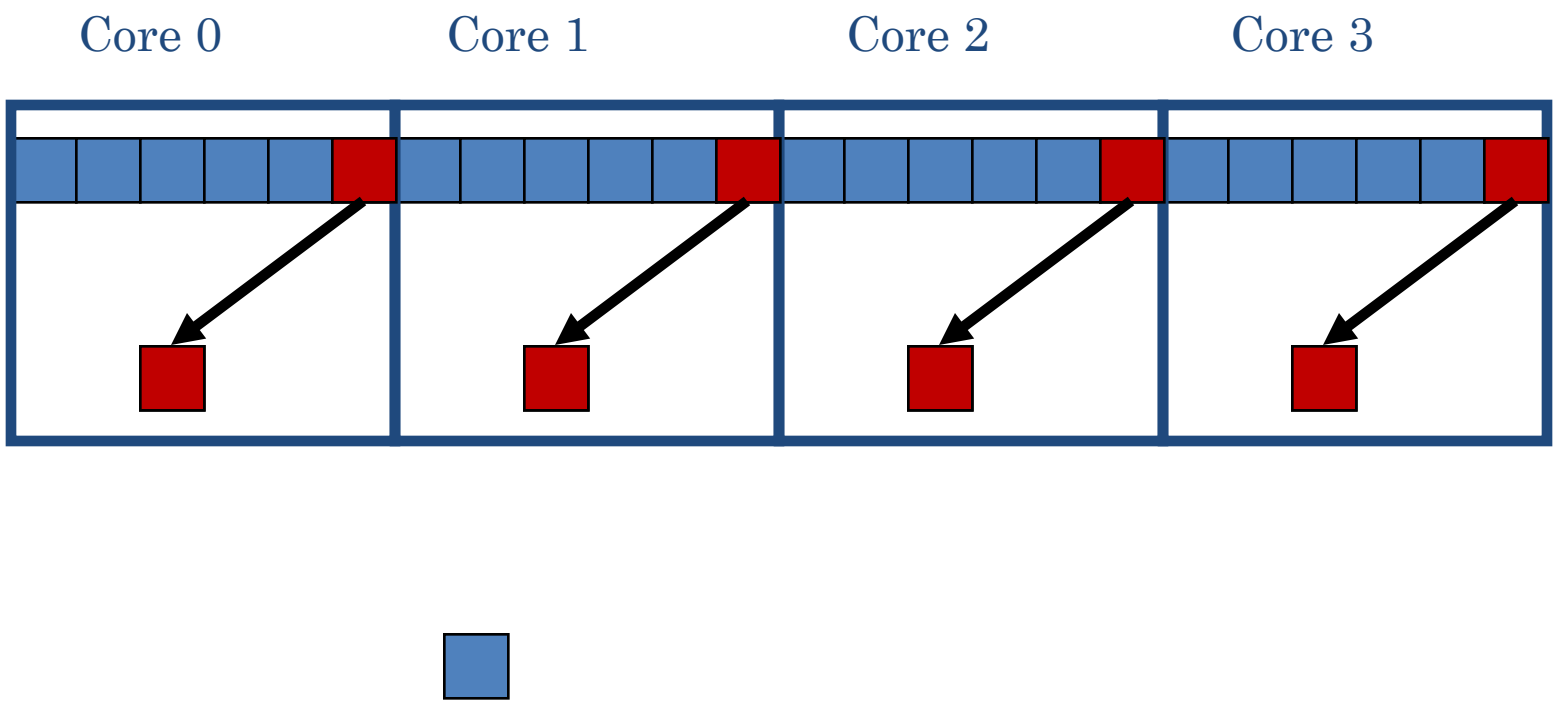
Domain Decomposition

Find the largest element of an array



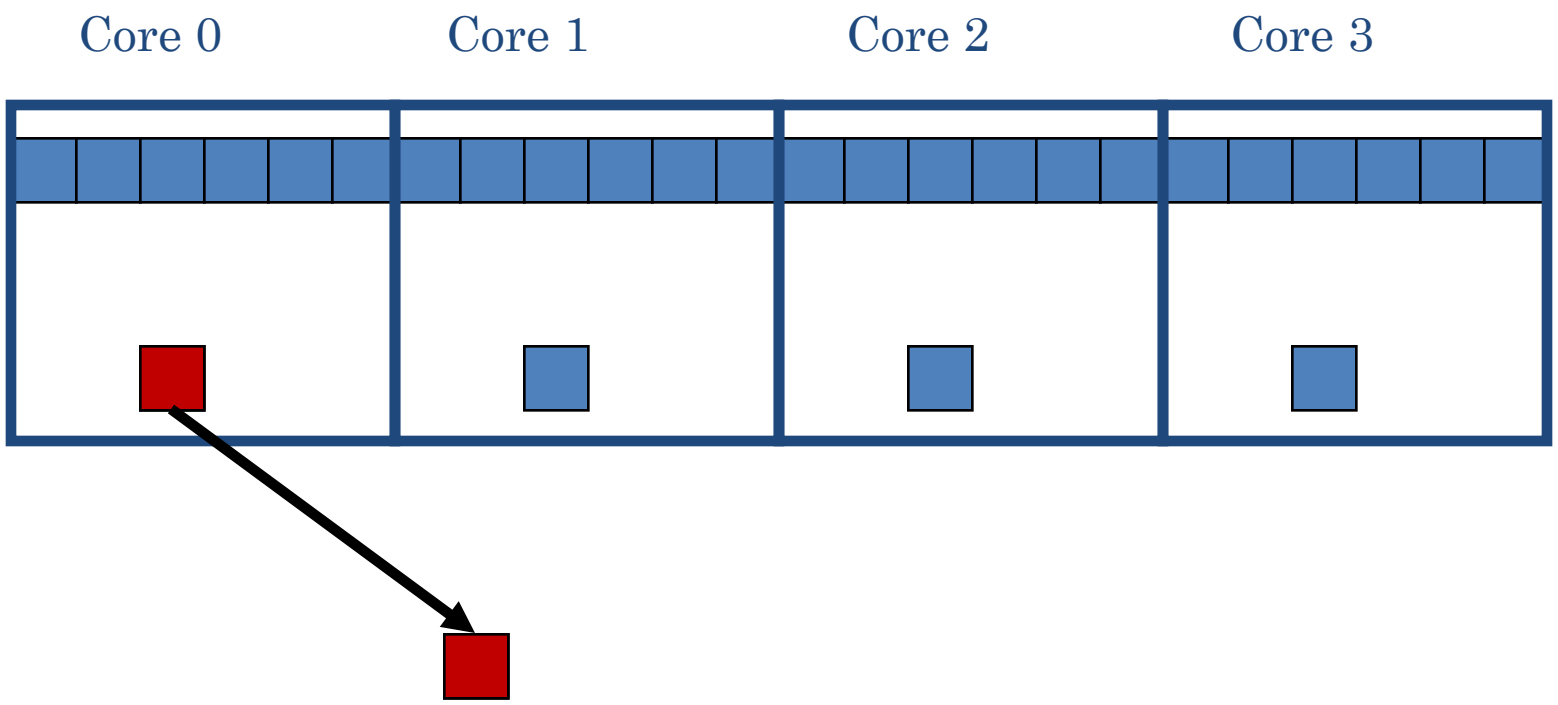
Domain Decomposition

Find the largest element of an array



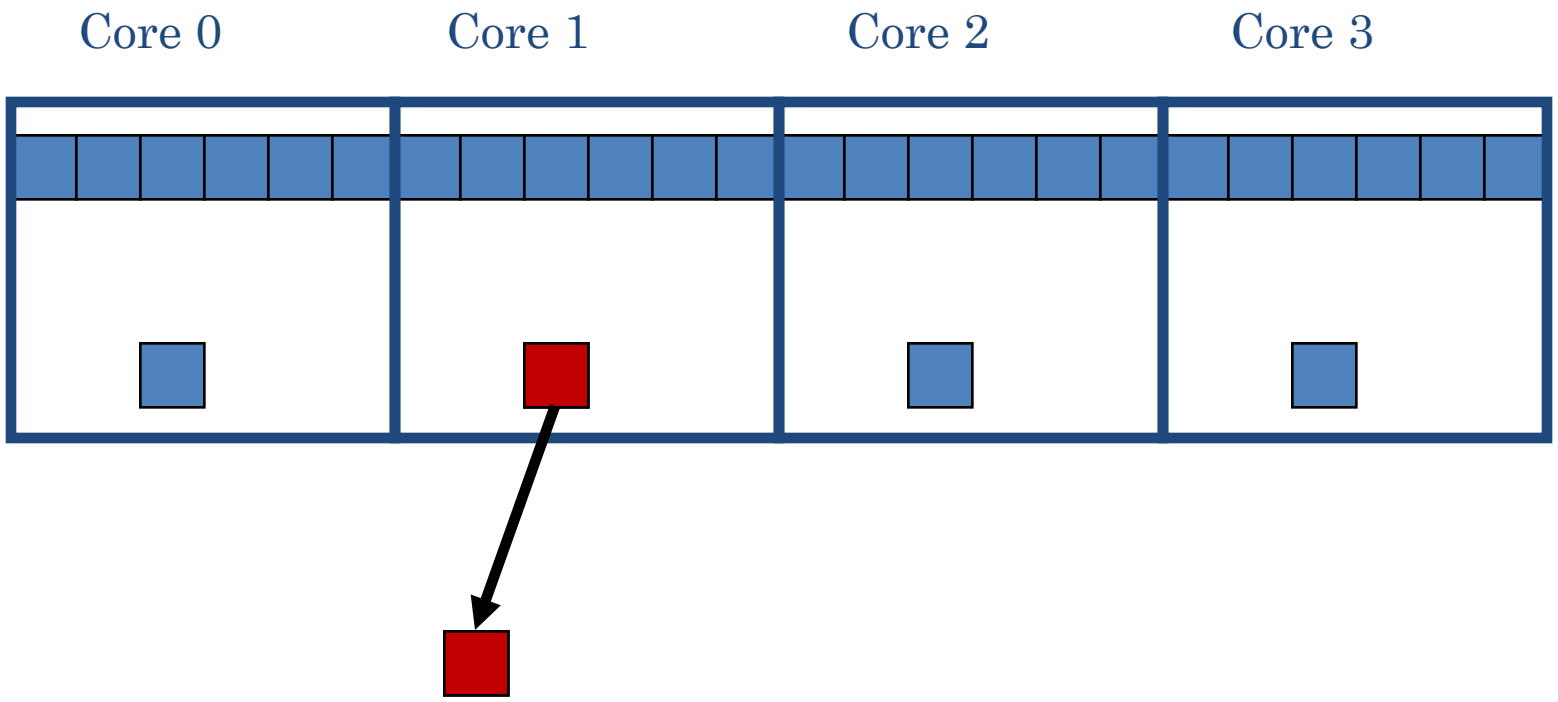
Domain Decomposition

Find the largest element of an array



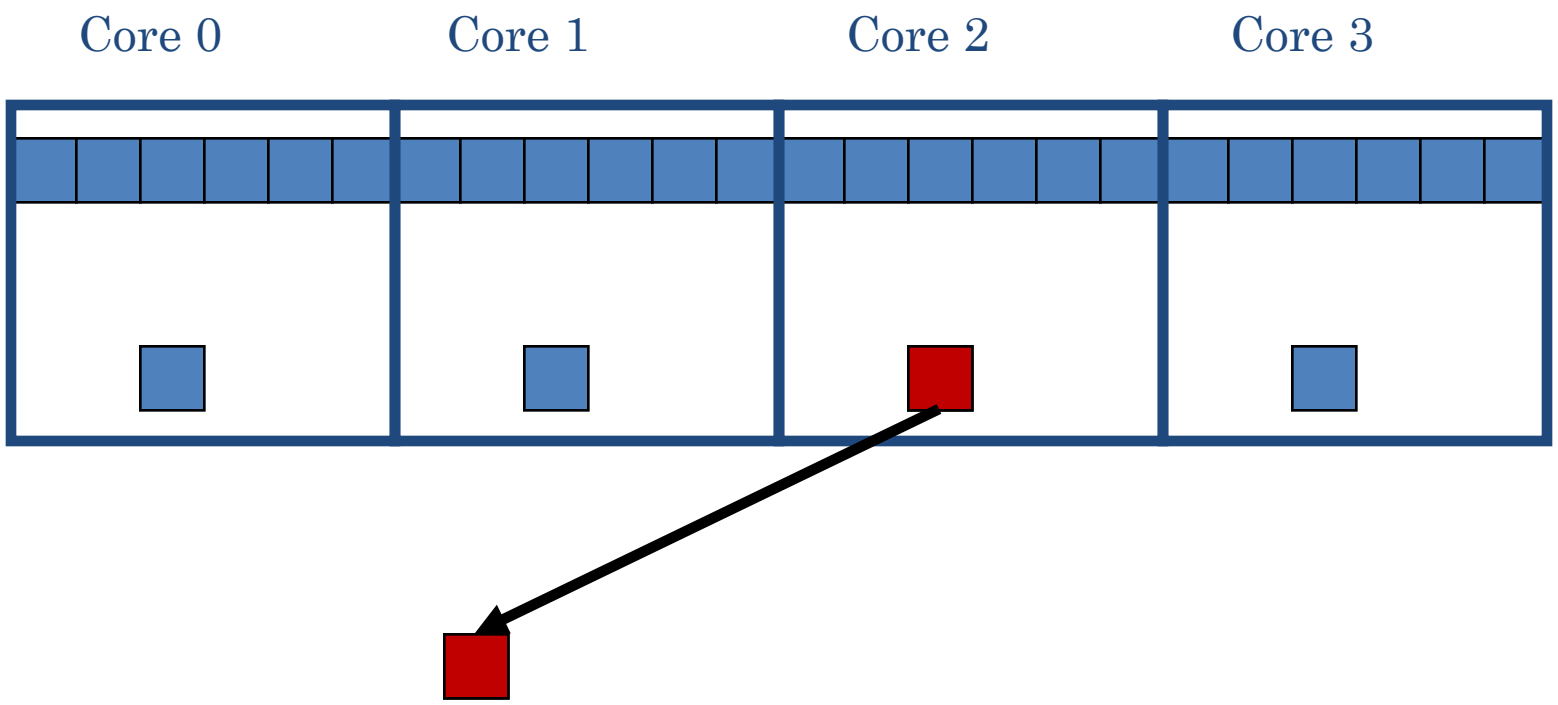
Domain Decomposition

Find the largest element of an array



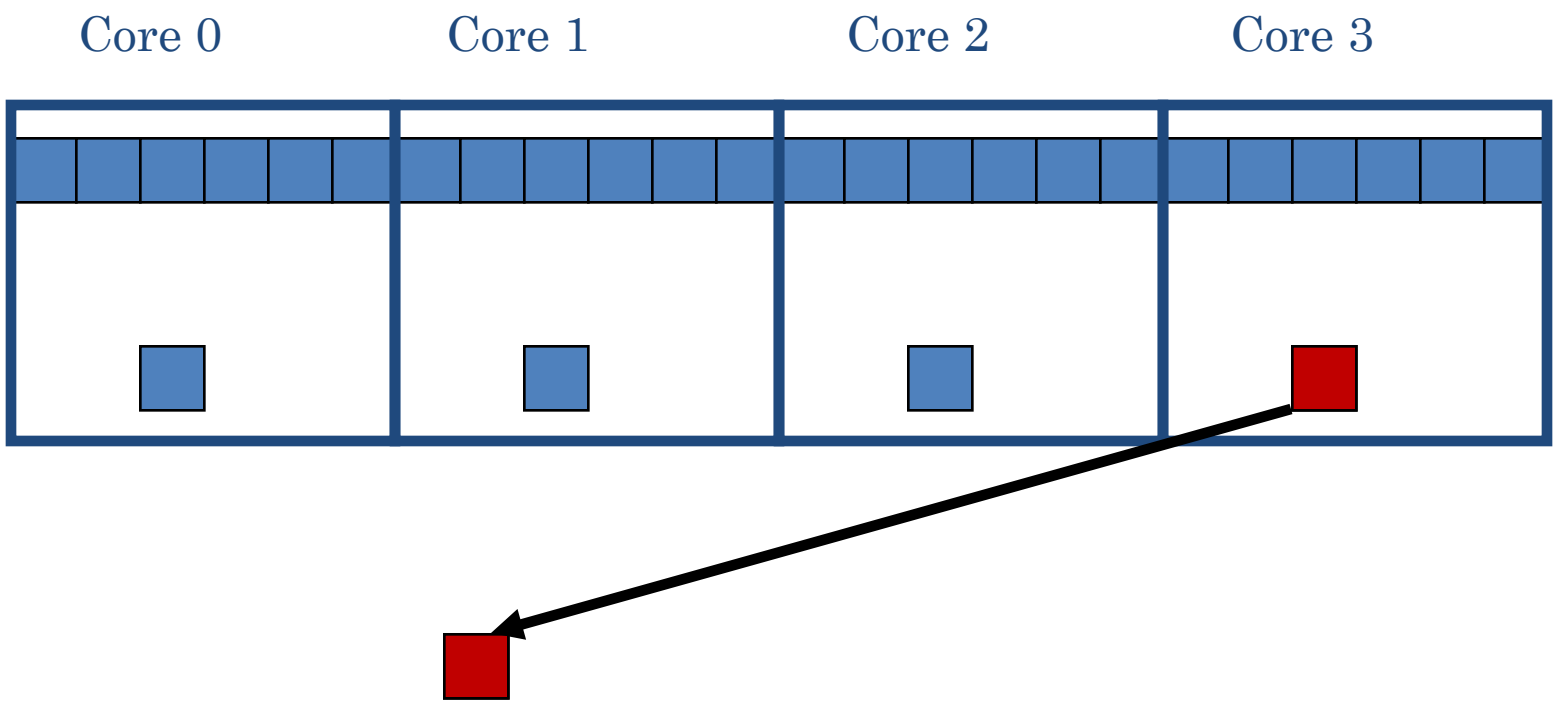
Domain Decomposition

Find the largest element of an array



Domain Decomposition

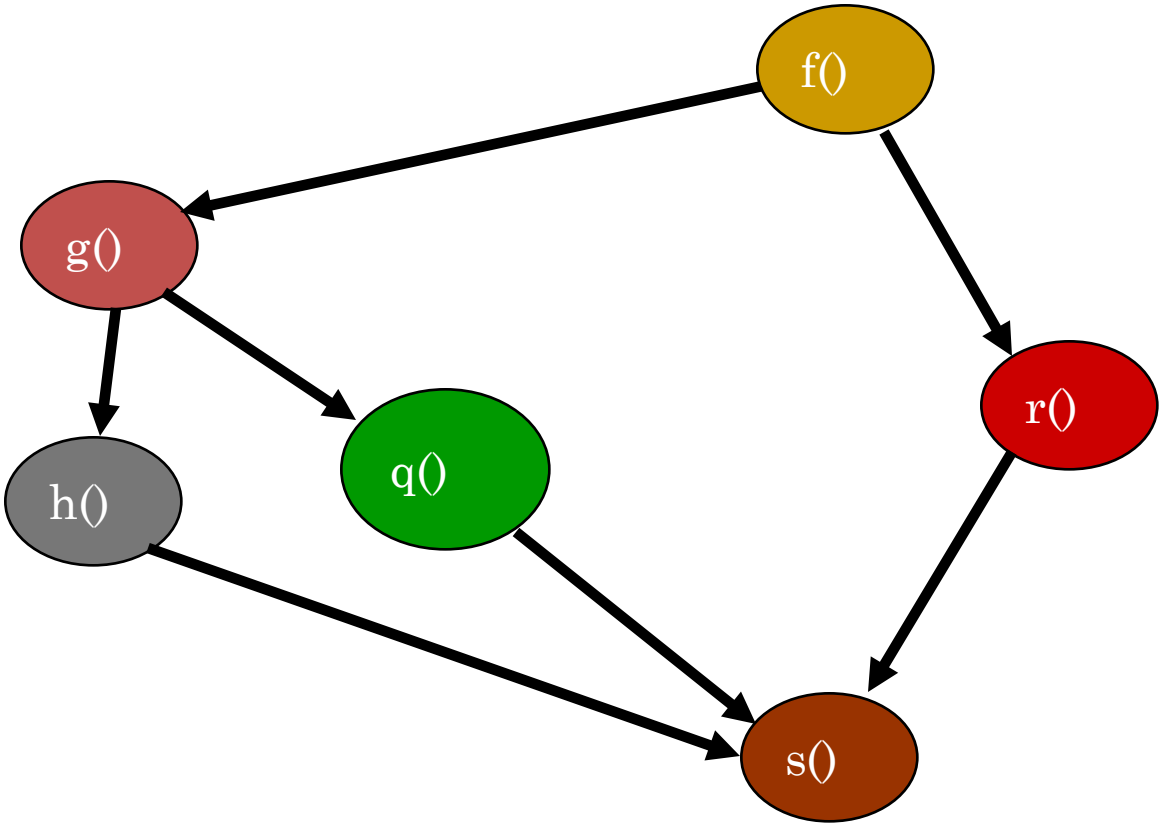
Find the largest element of an array



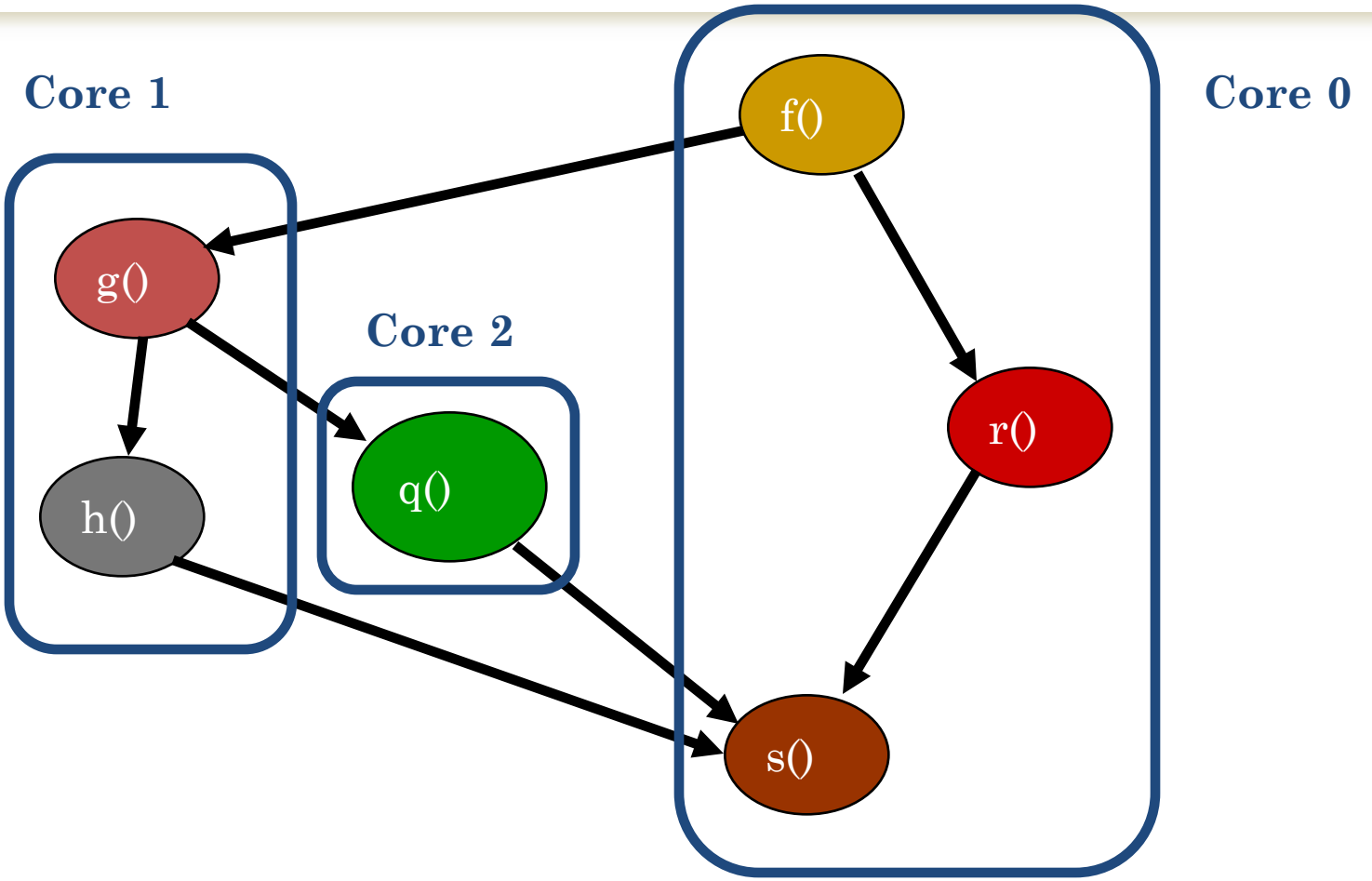
Task (Functional) Decomposition

- First, divide problem into independent tasks
- Second, decide which data elements are going to be accessed (read and/or written) by which tasks
- Example: Event-handler for GUI

Task Decomposition

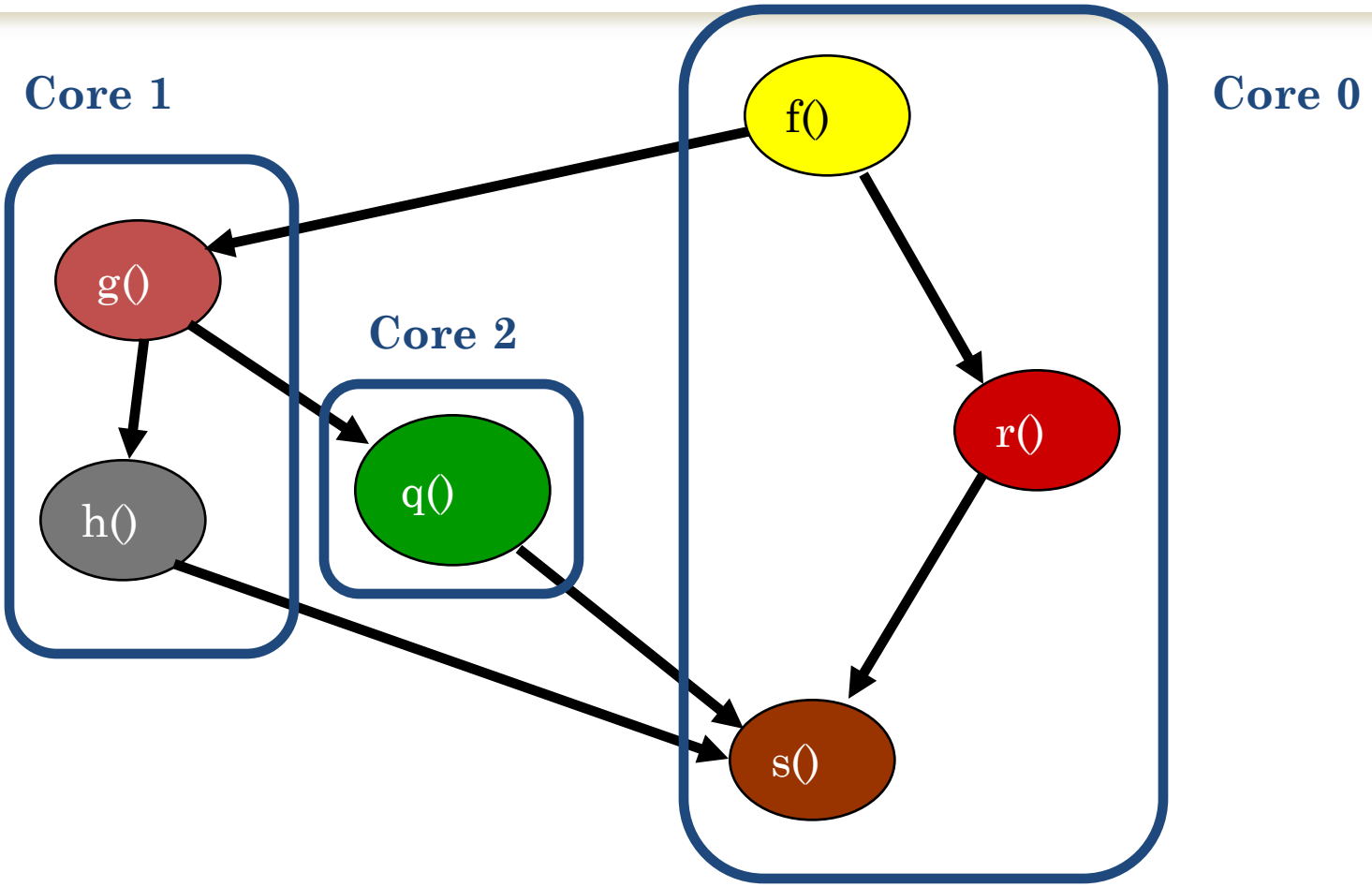


Task Decomposition

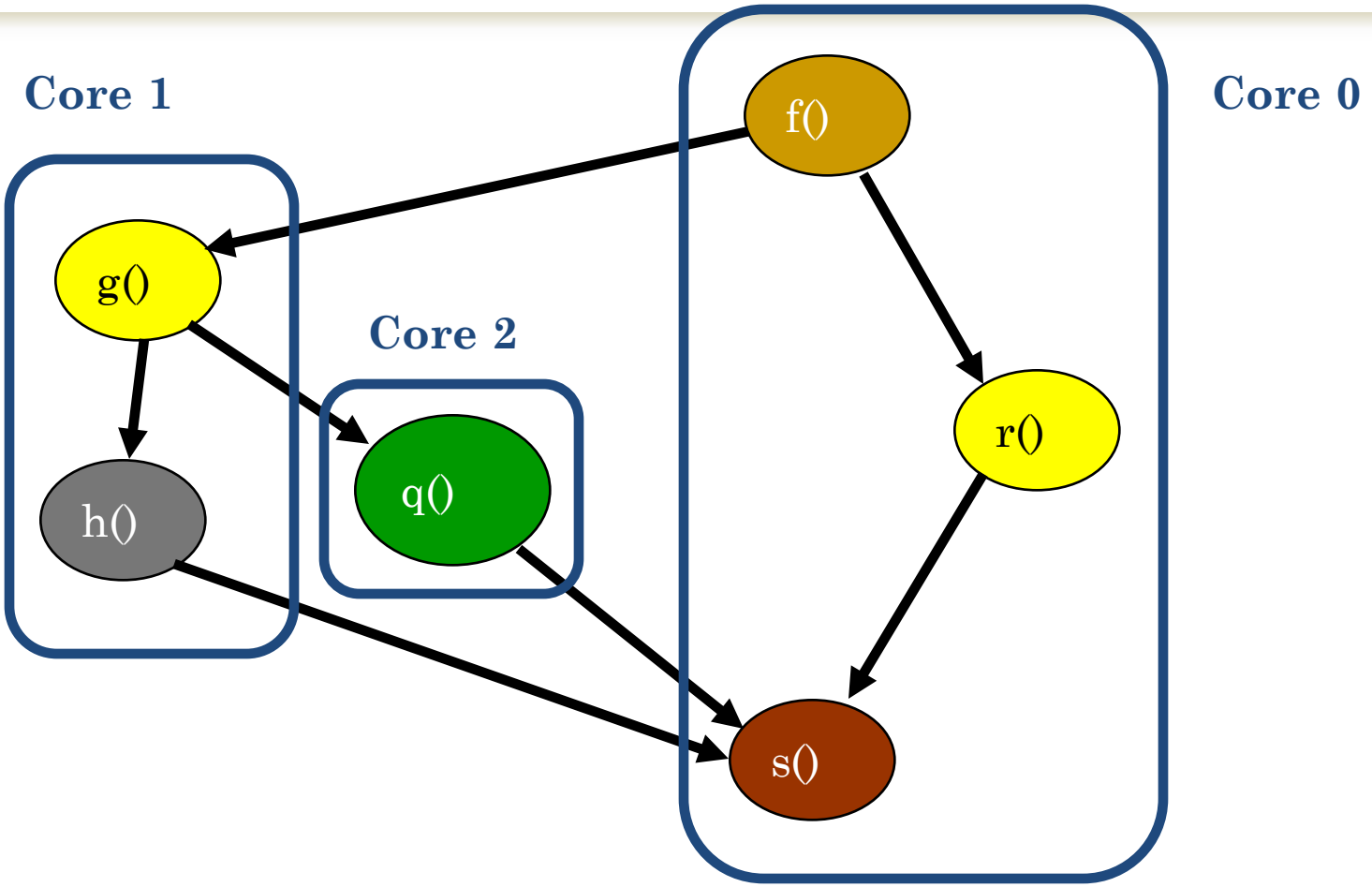


Why is there no point in assigning “f”, “r”, and “s” to different CPUs?

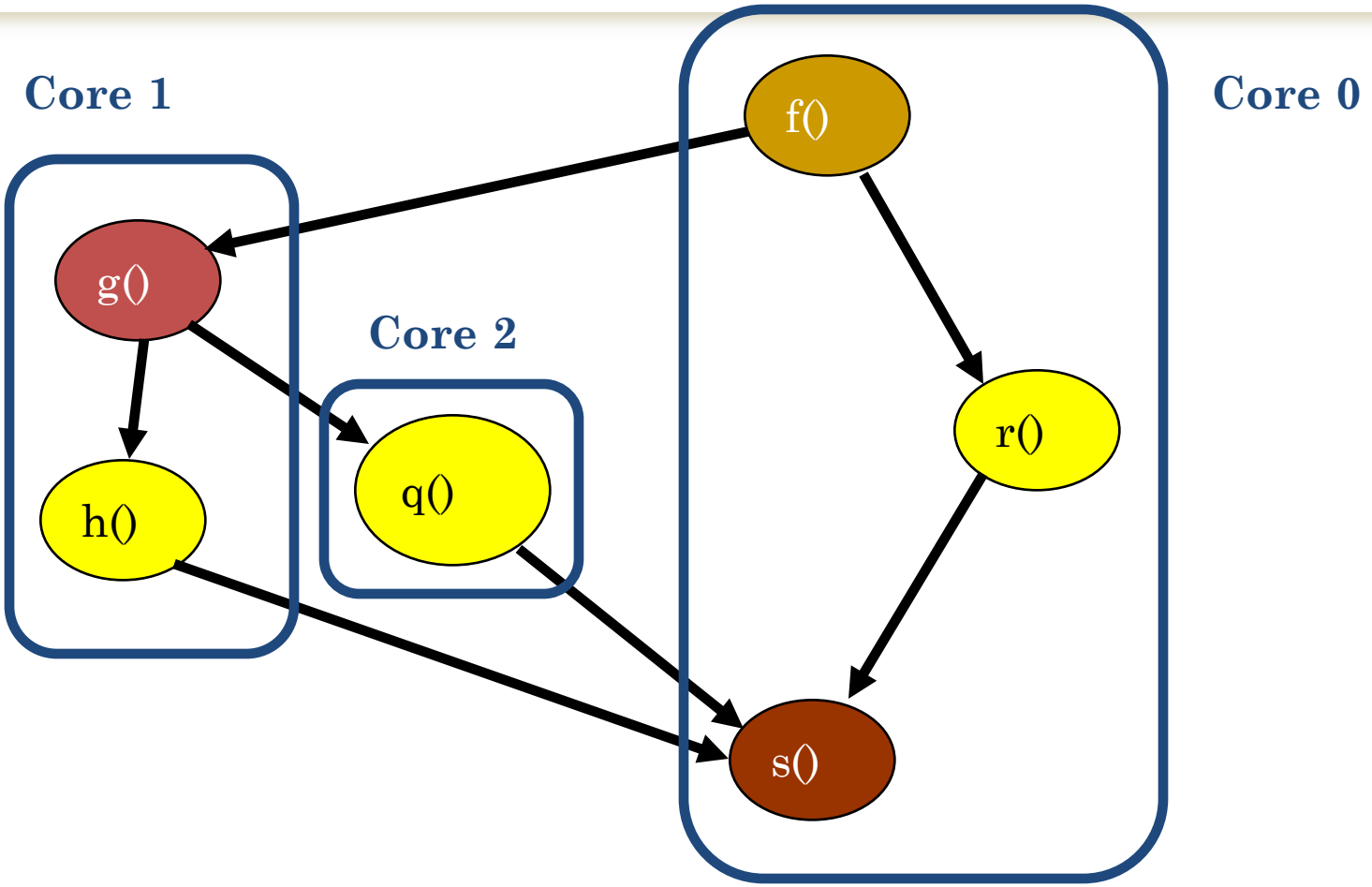
Task Decomposition



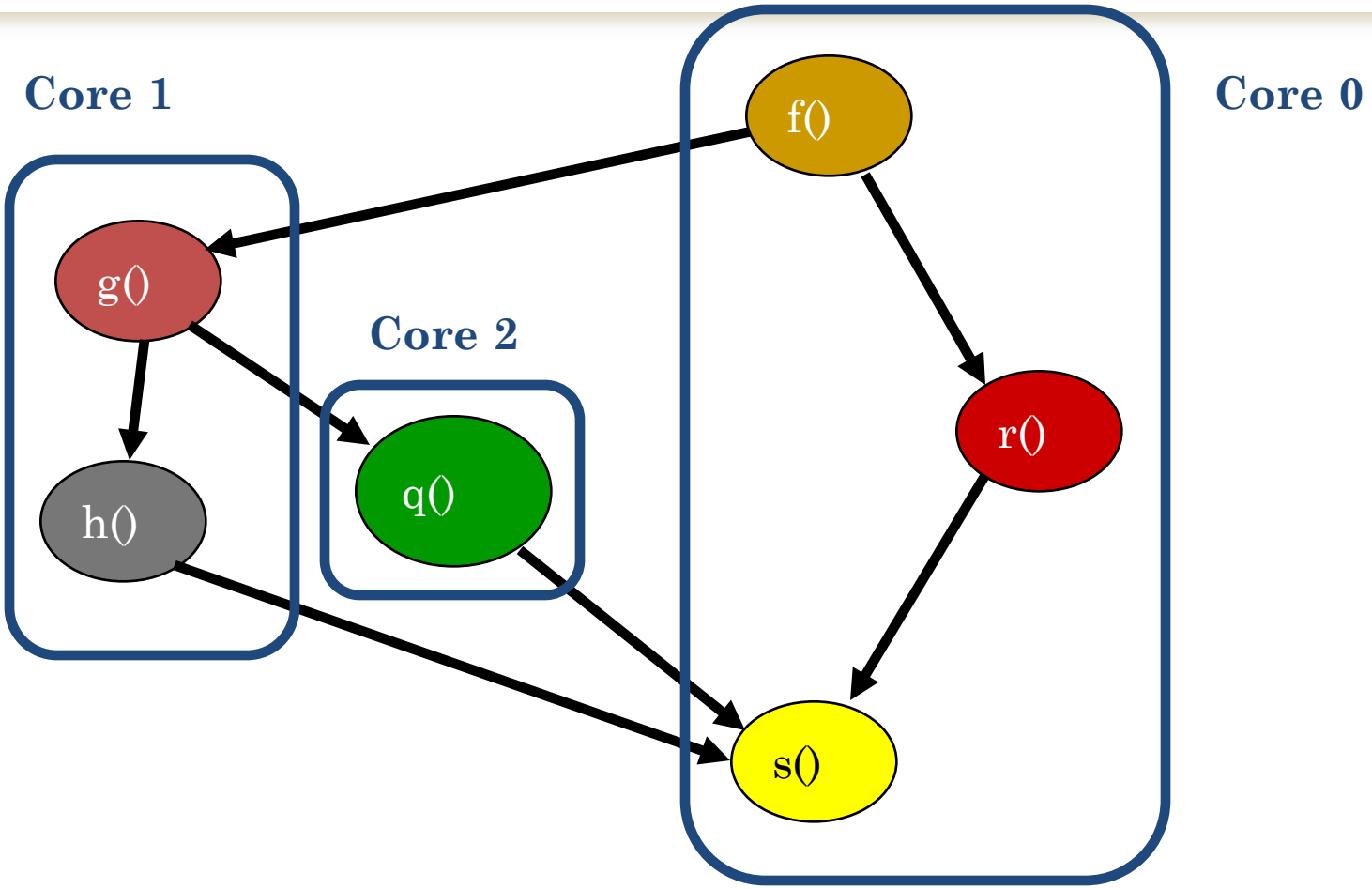
Task Decomposition



Task Decomposition

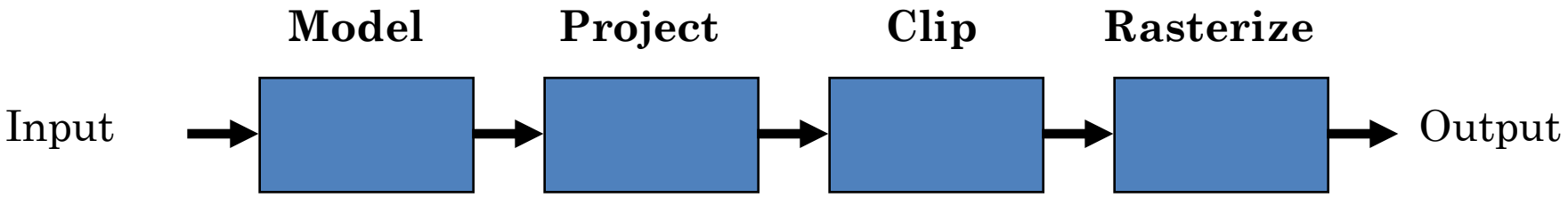


Task Decomposition

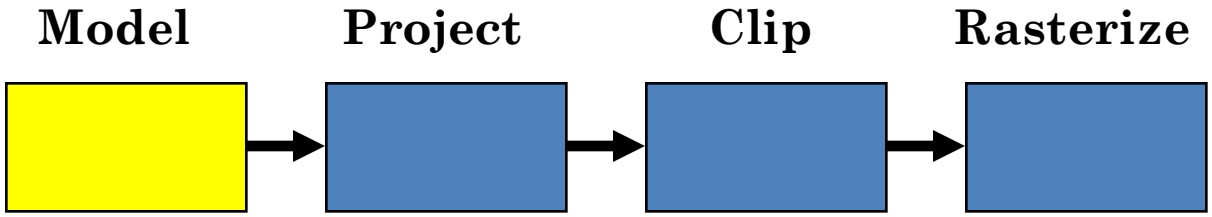


Pipelining

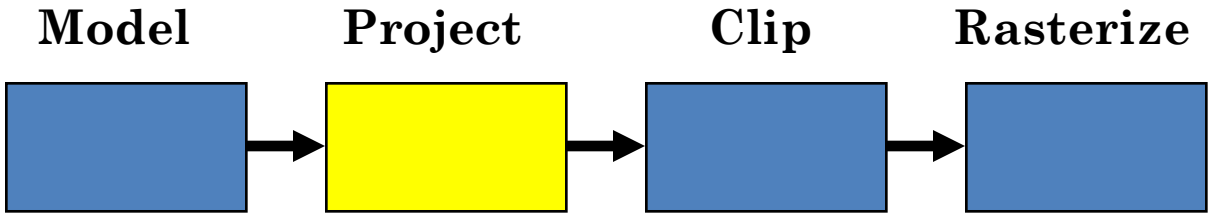
- Special kind of task decomposition
- "Assembly line" parallelism
- Example: 3D rendering in computer graphics
- The throughput is limited by the slowest stage



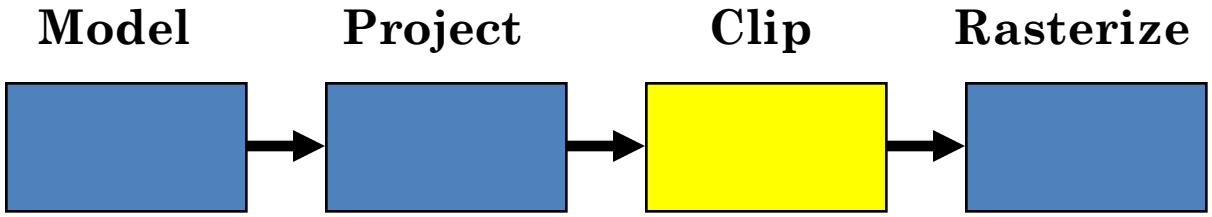
Processing One Data Set (Step 1)



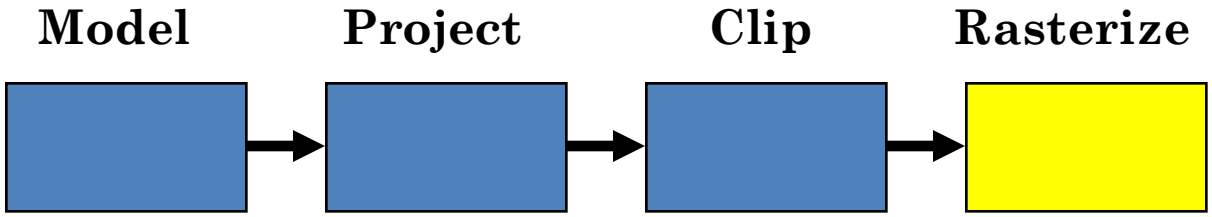
Processing One Data Set (Step 2)



Processing One Data Set (Step 3)

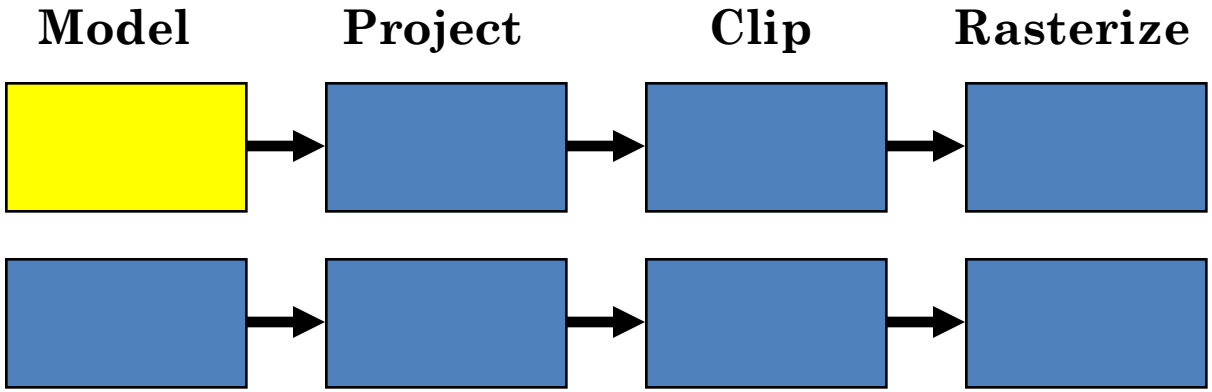


Processing One Data Set (Step 4)

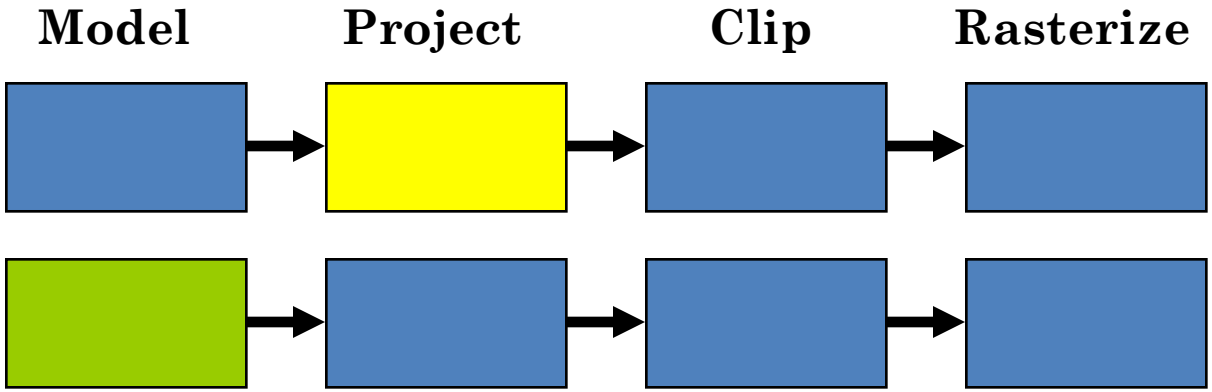


The pipeline processes 1 data set in 4 steps

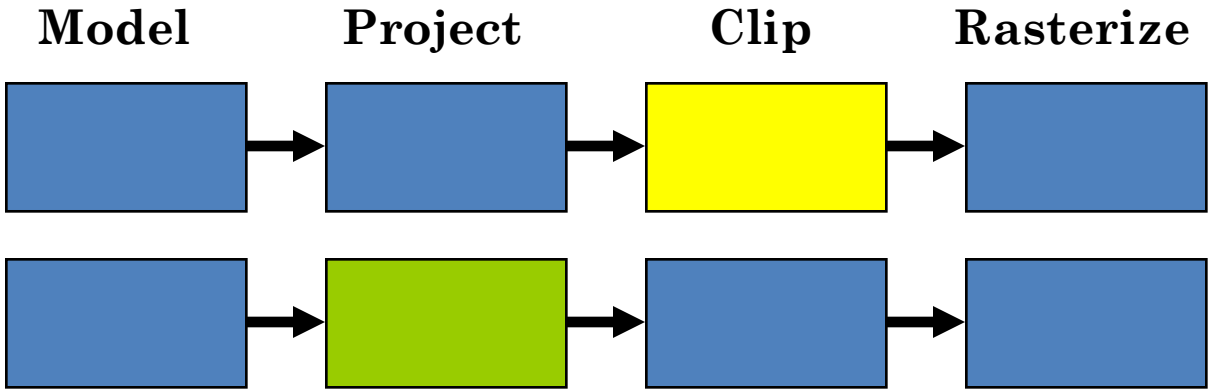
Processing Two Data Sets (Step 1)



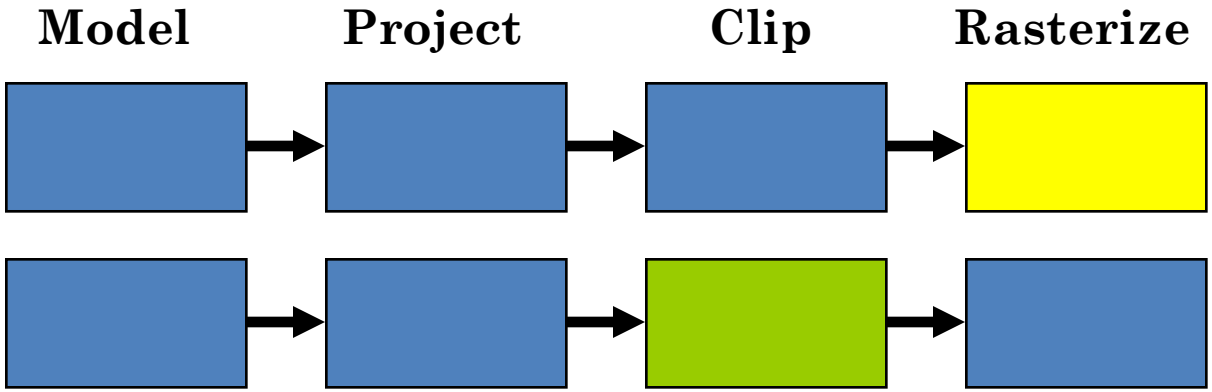
Processing Two Data Sets (Time 2)



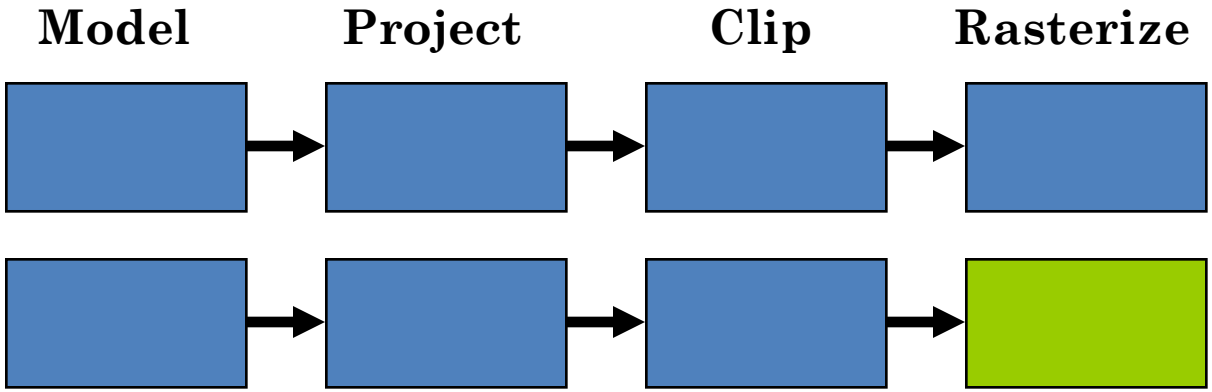
Processing Two Data Sets (Step 3)



Processing Two Data Sets (Step 4)

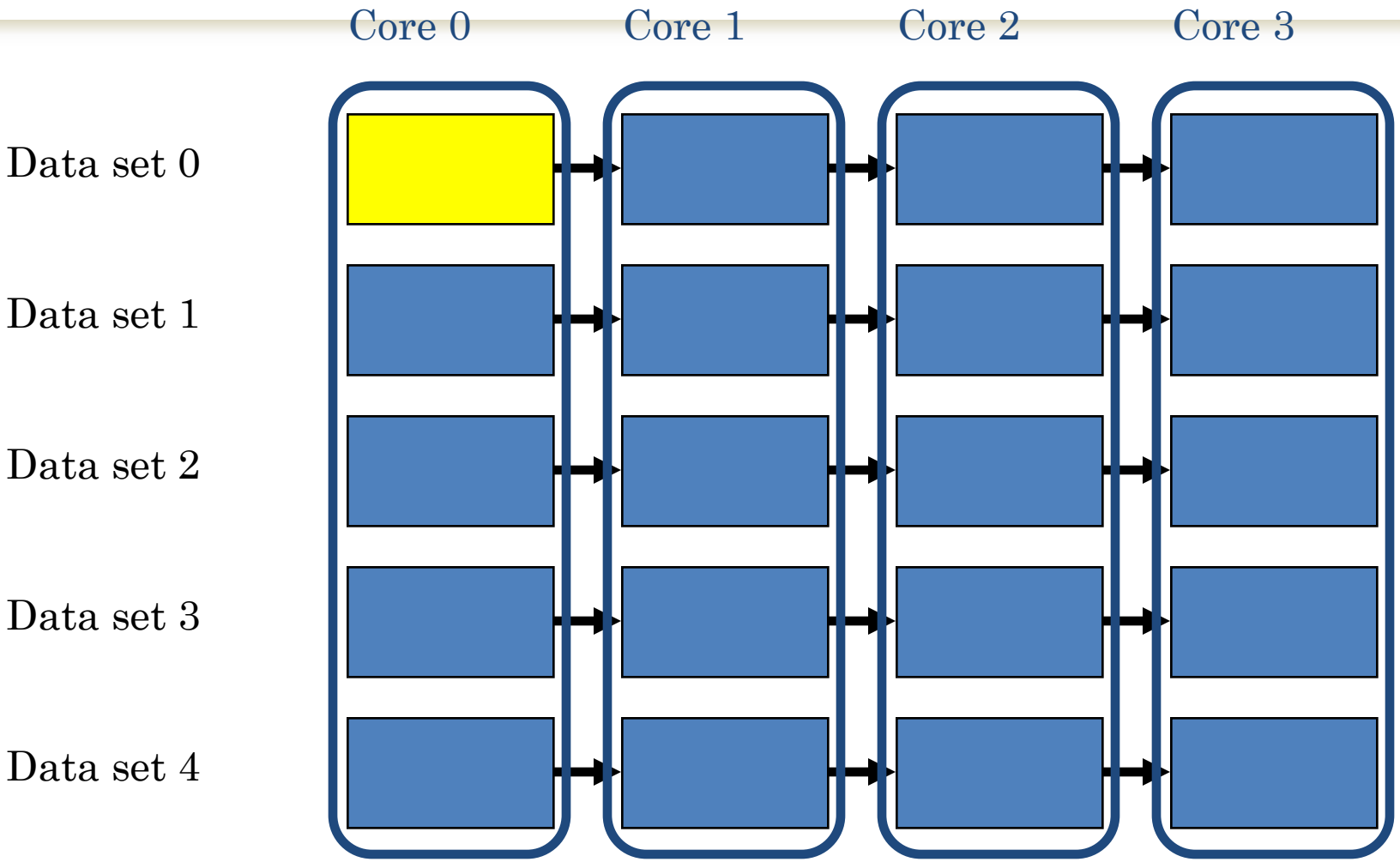


Processing Two Data Sets (Step 5)

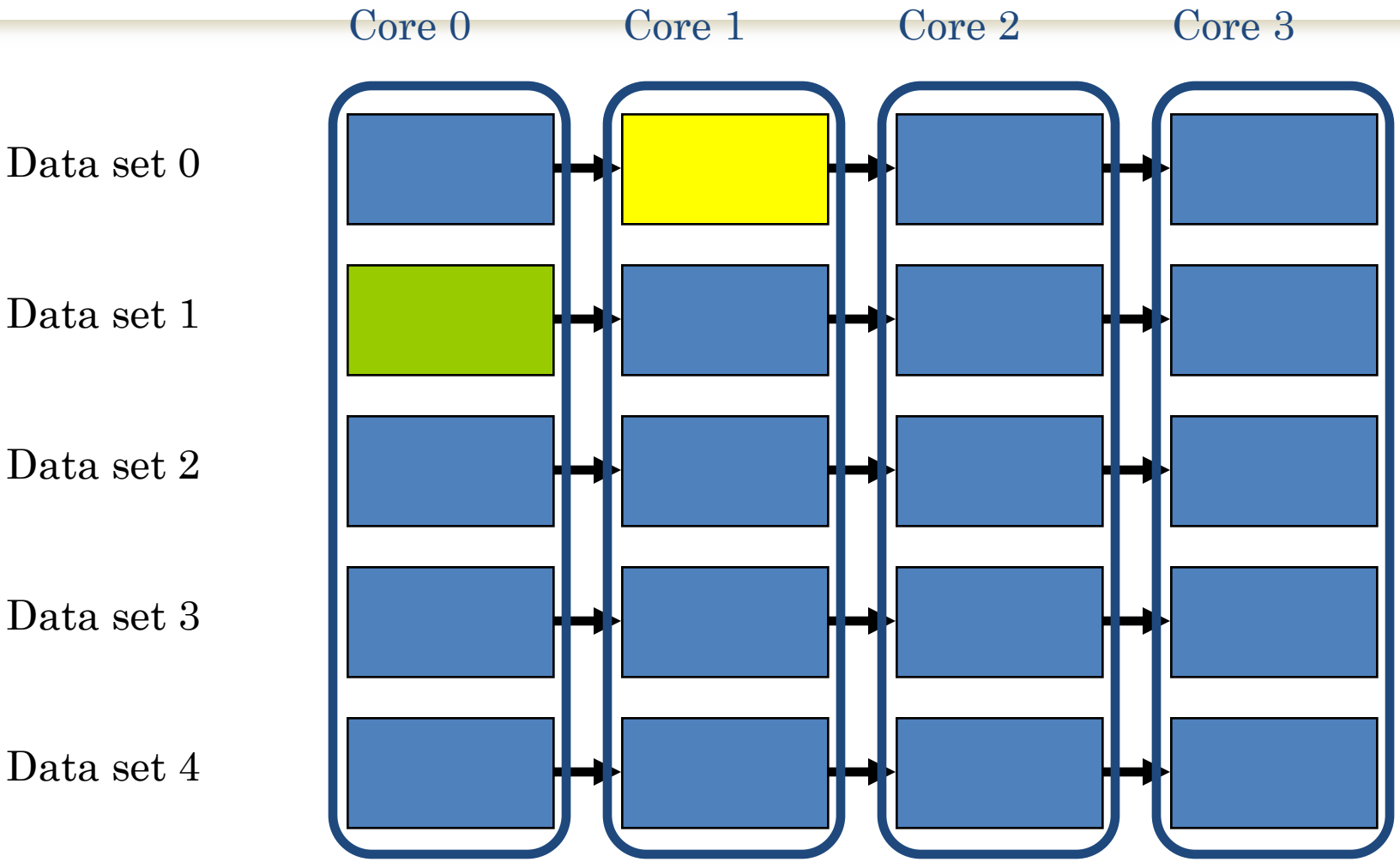


The pipeline processes 2 data sets in 5 steps

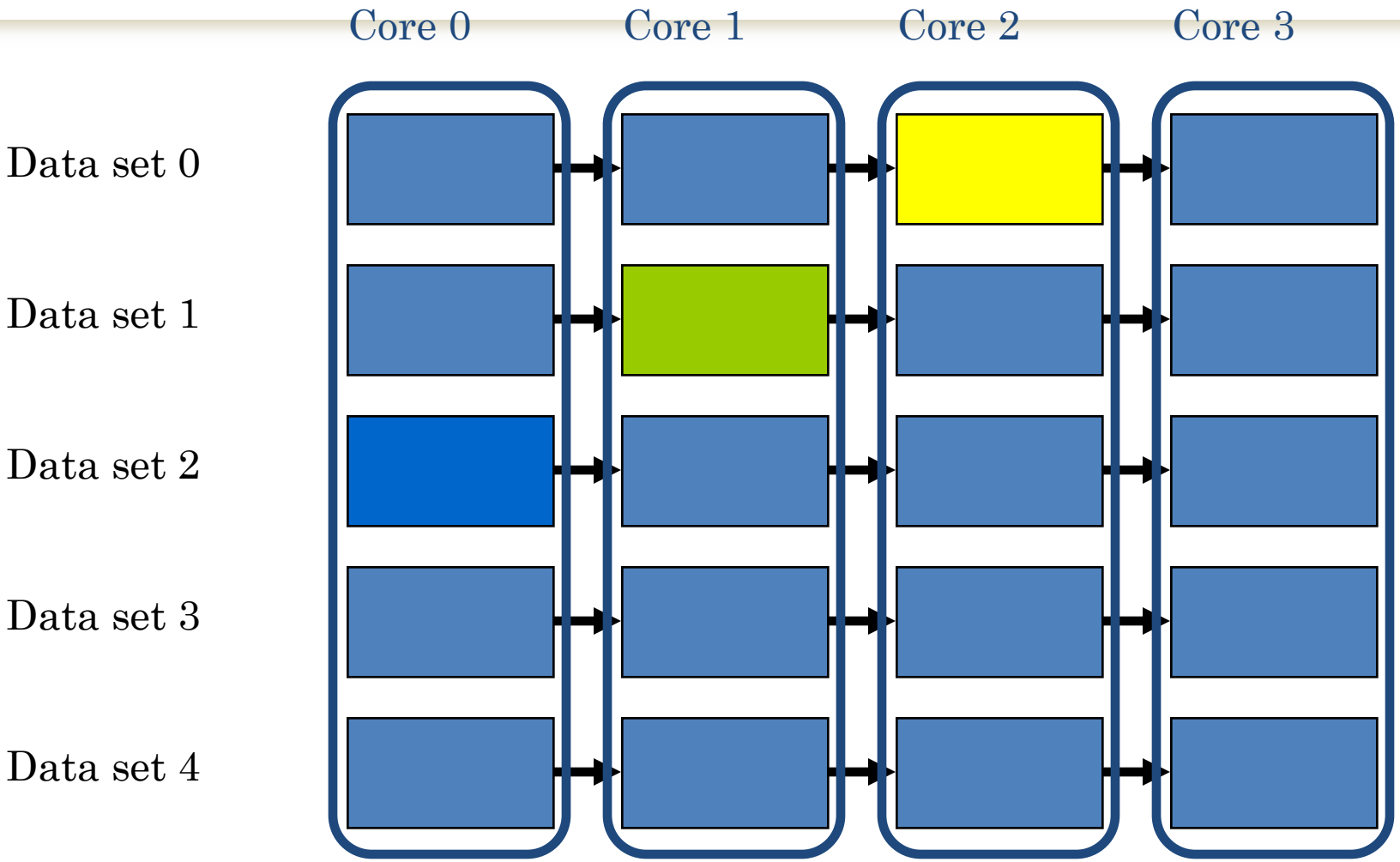
Pipelining Five Data Sets (Step 1)



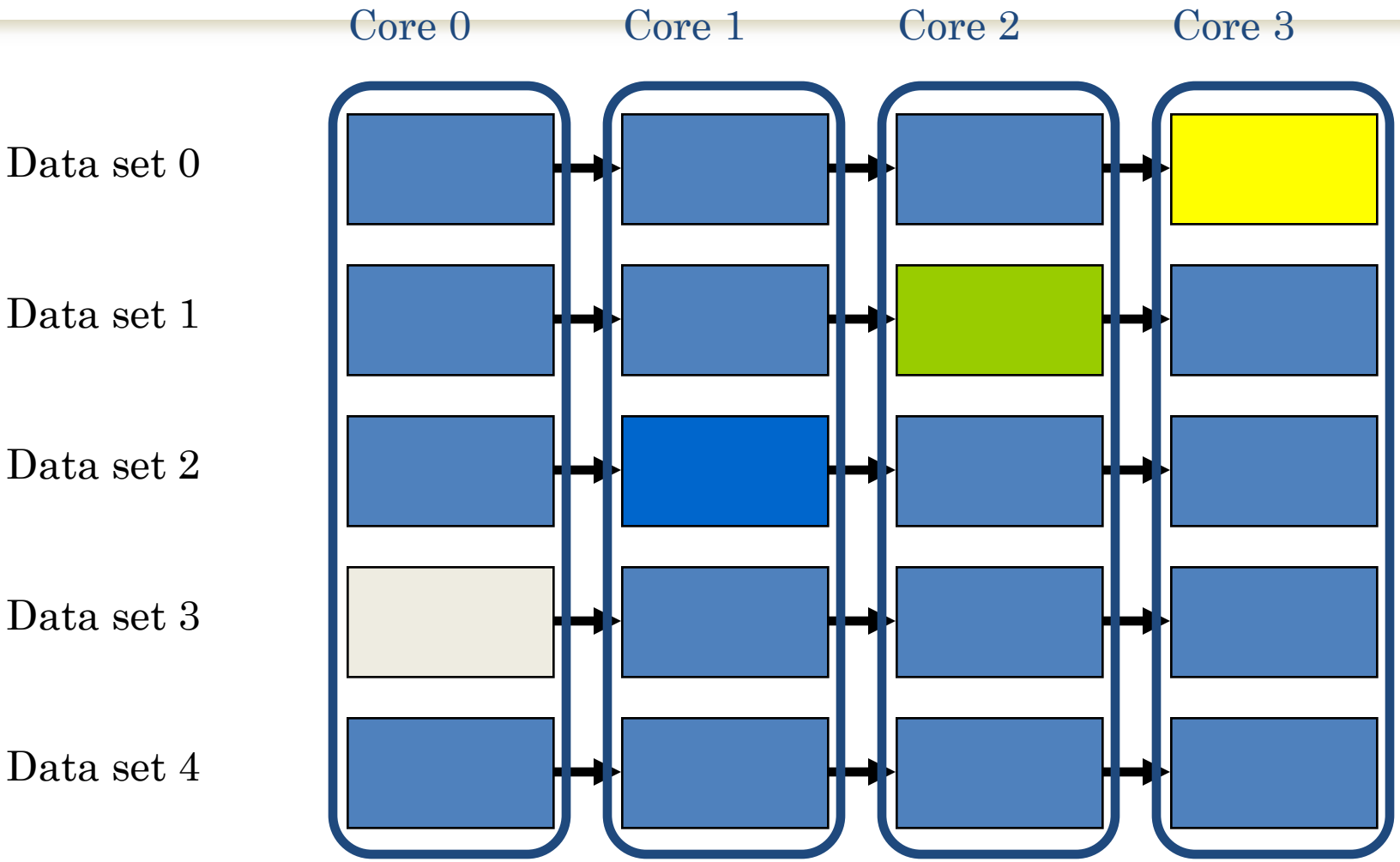
Pipelining Five Data Sets (Step 2)



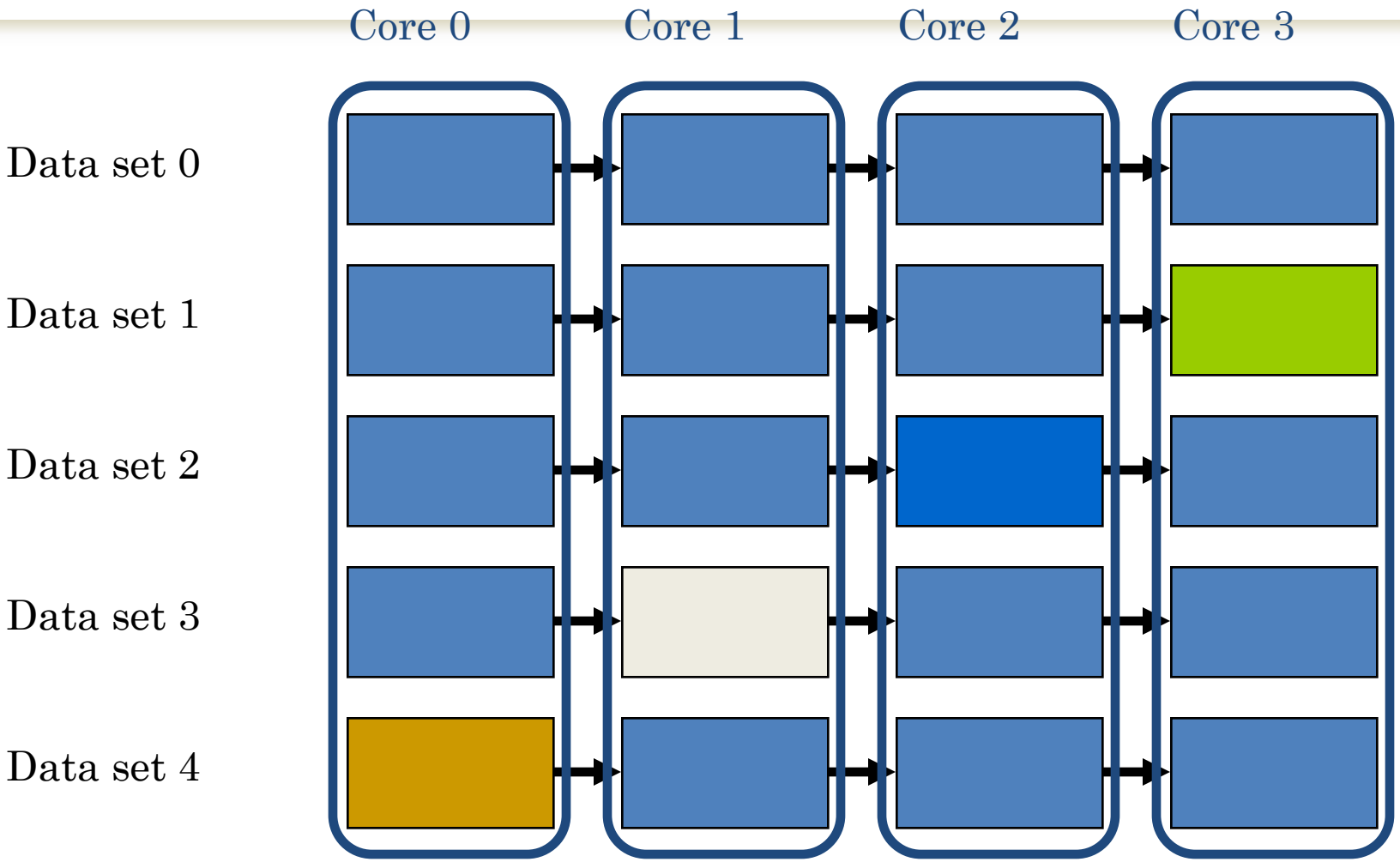
Pipelining Five Data Sets (Step 3)



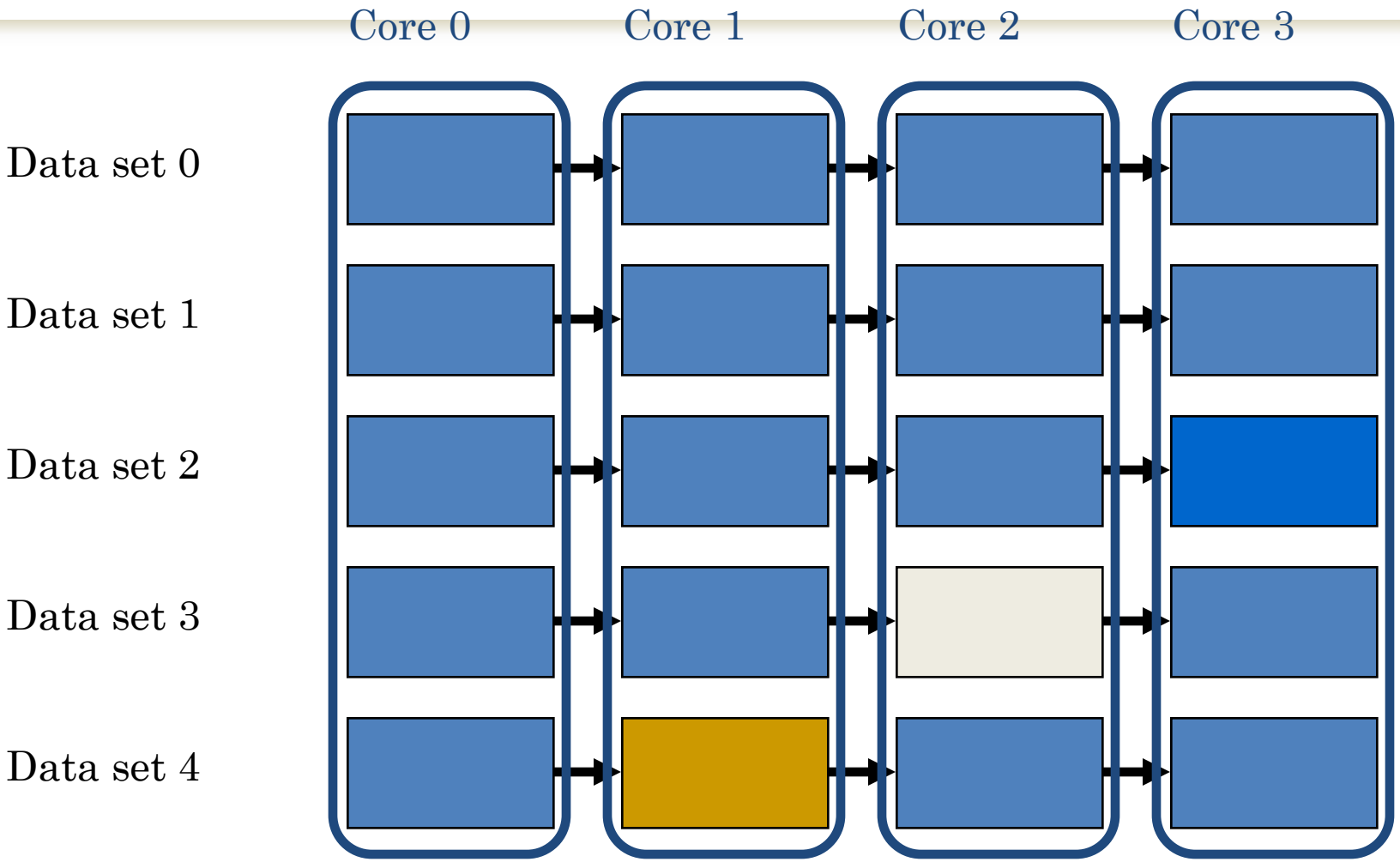
Pipelining Five Data Sets (Step 4)



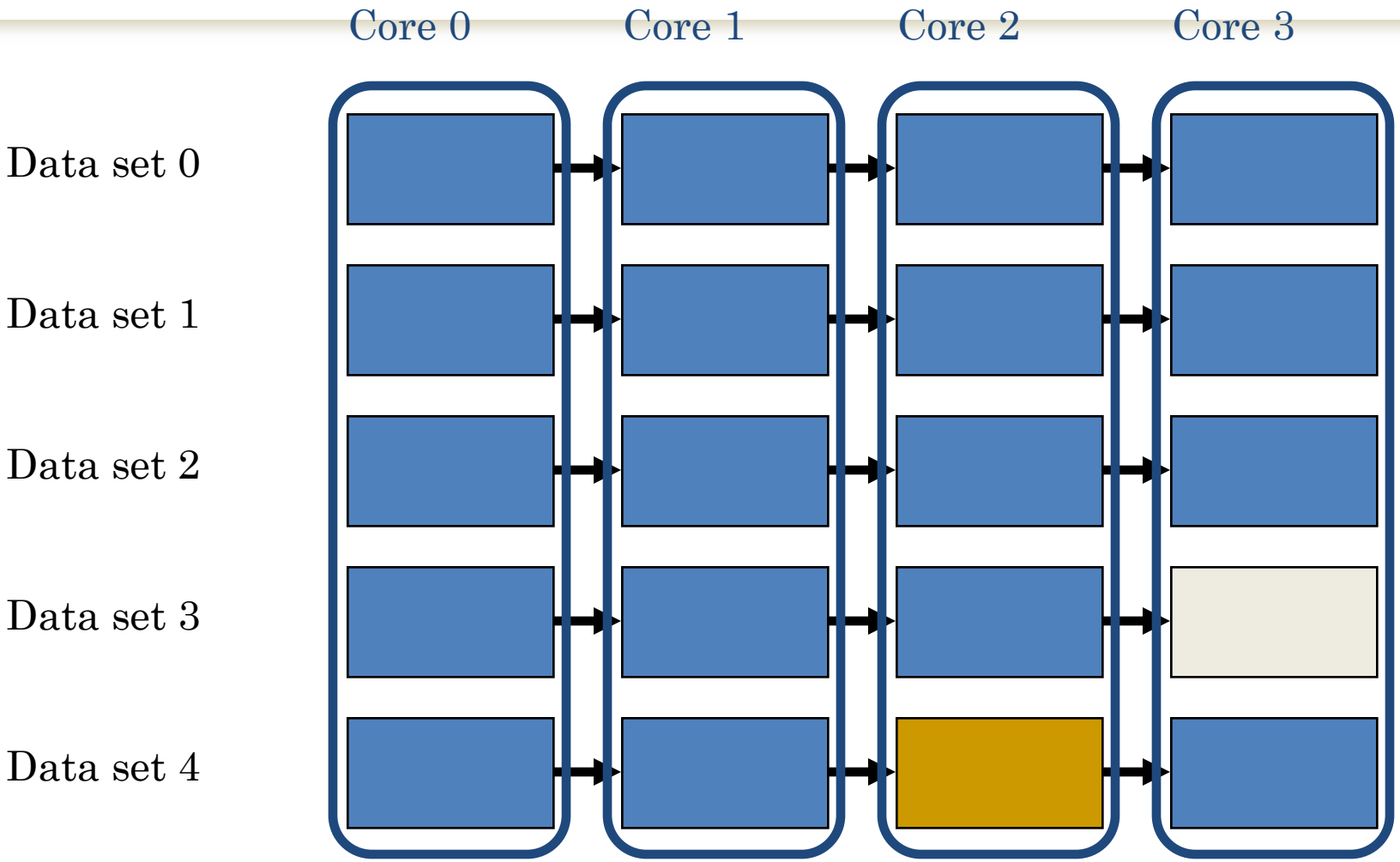
Pipelining Five Data Sets (Step 5)



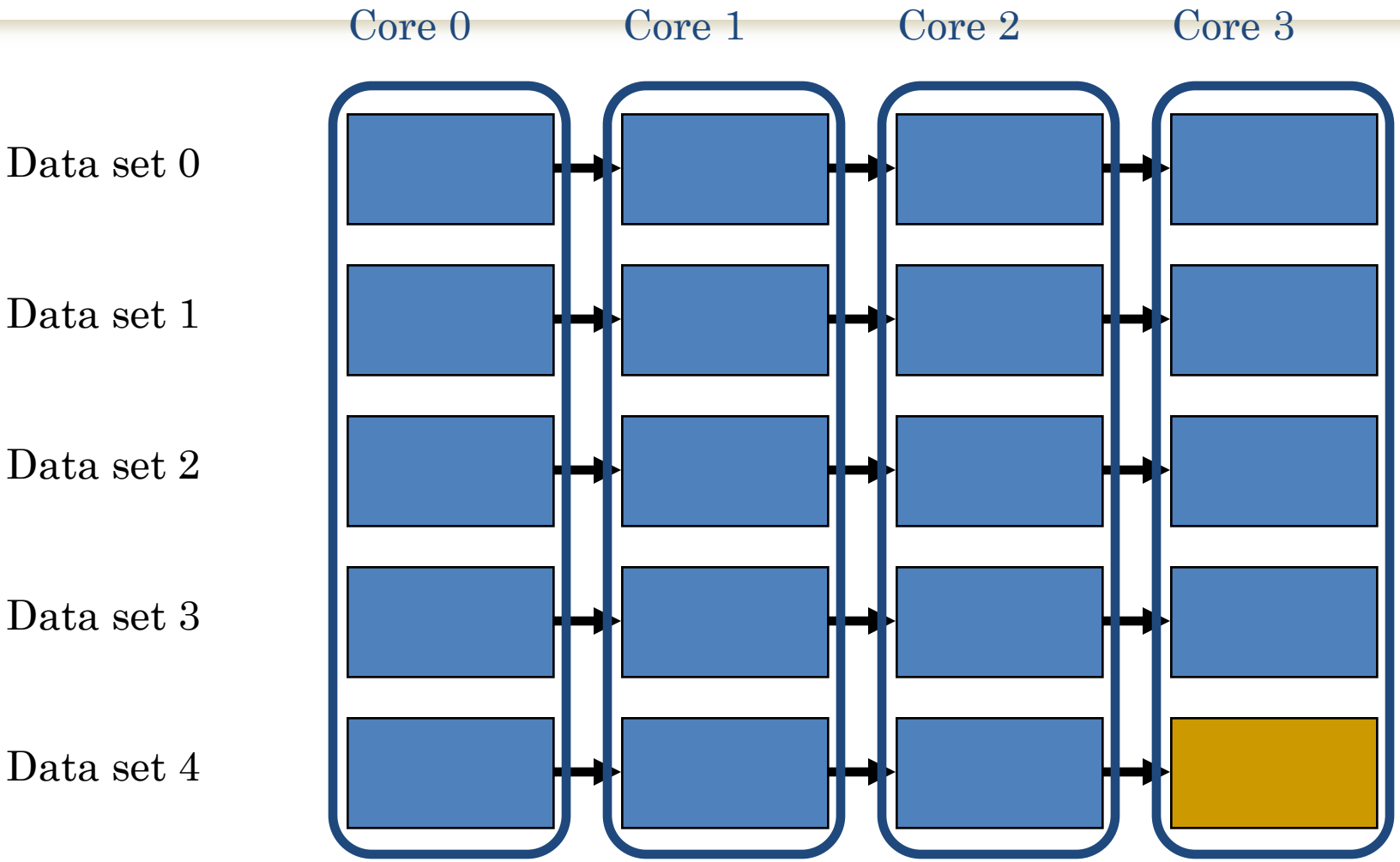
Pipelining Five Data Sets (Step 6)



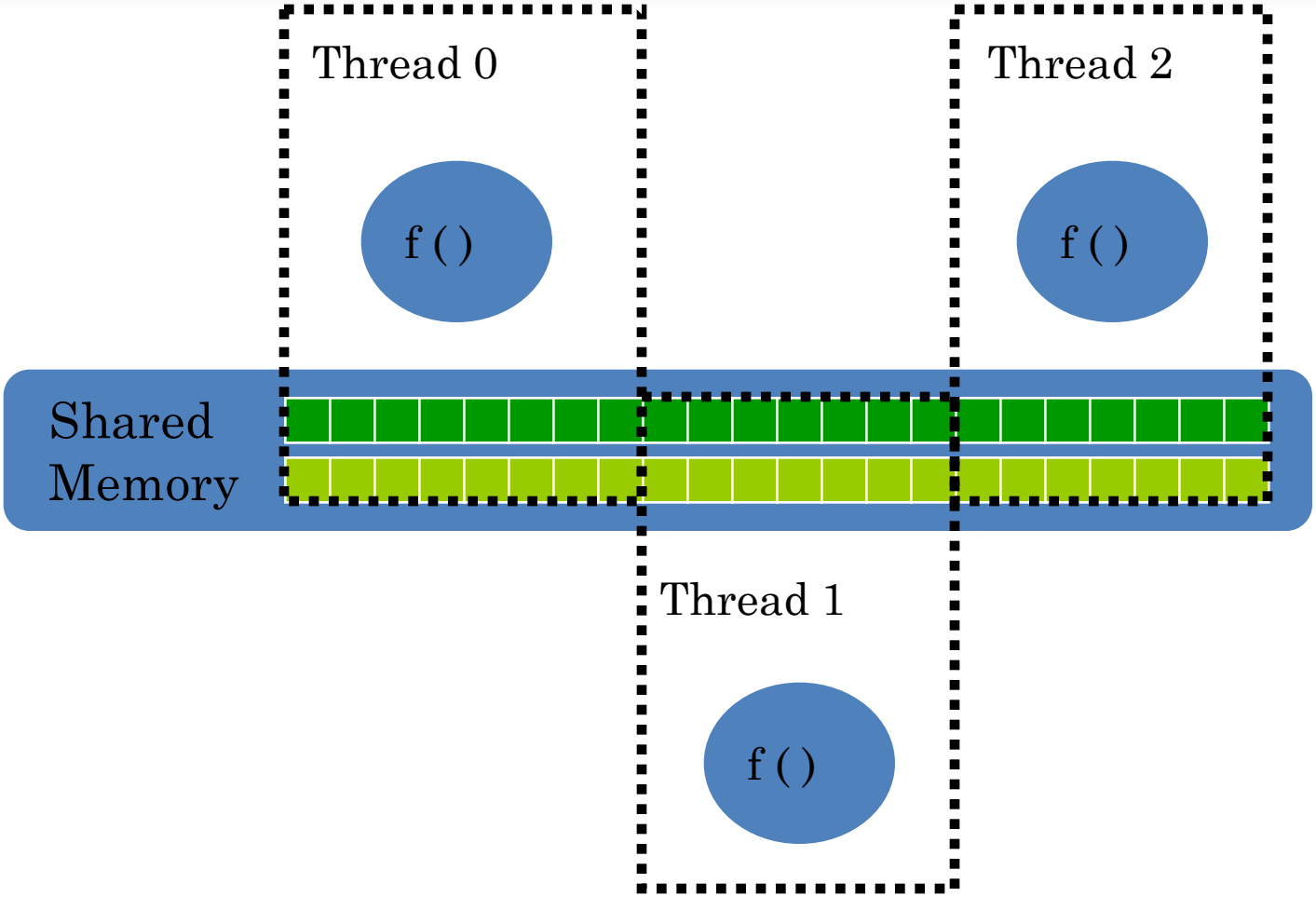
Pipelining Five Data Sets (Step 7)



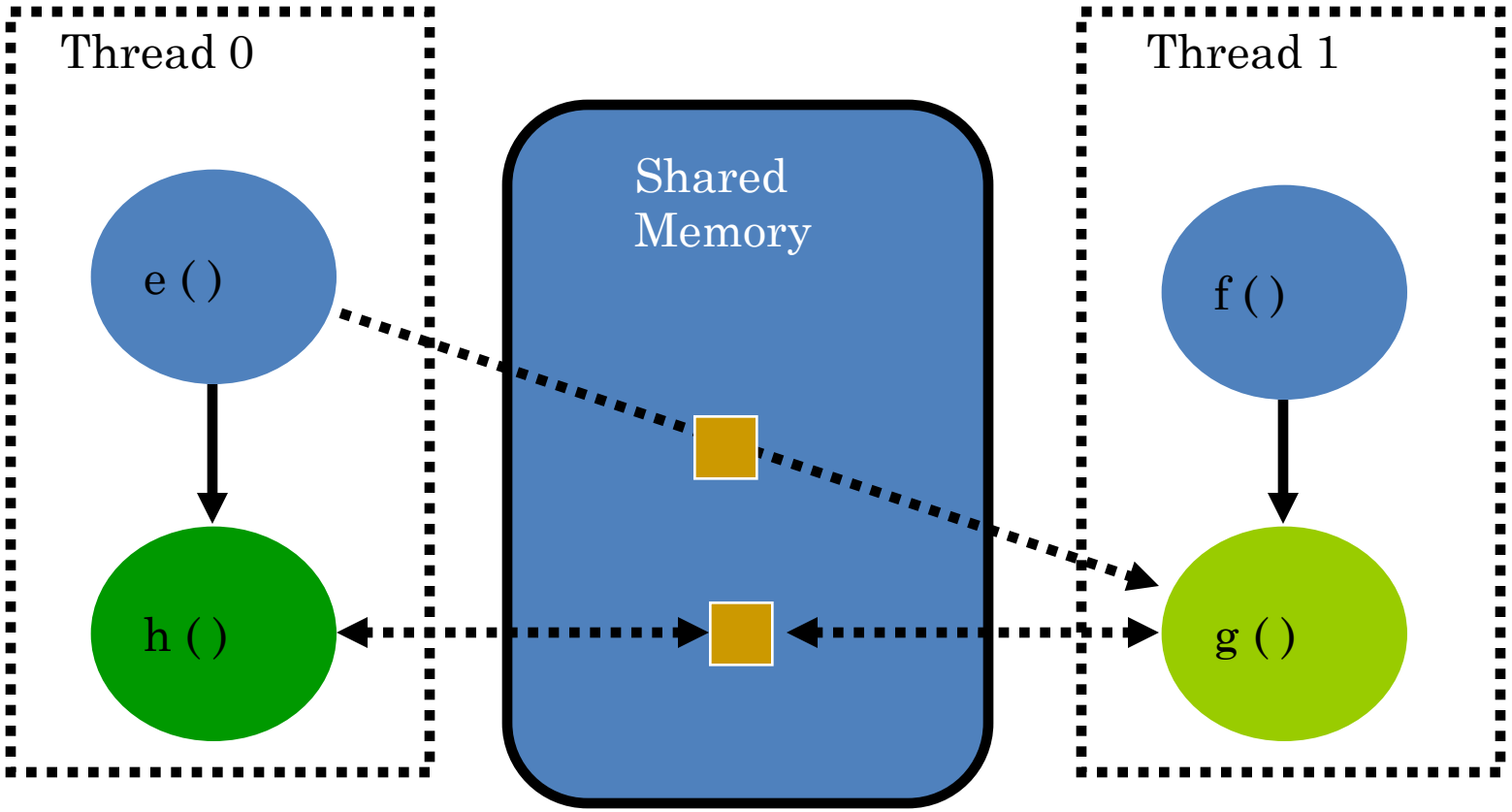
Pipelining Five Data Sets (Step 8)



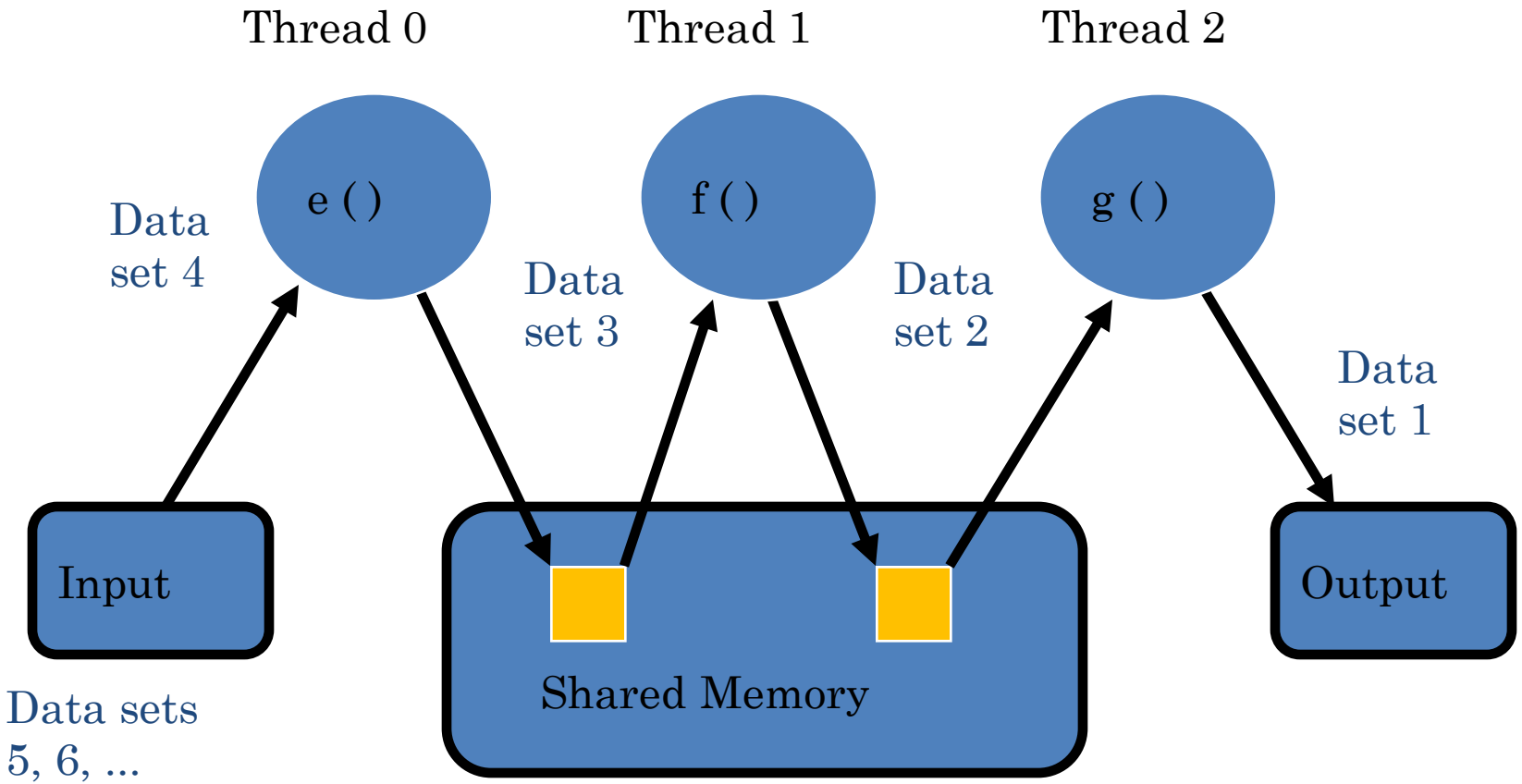
Domain Decomposition Using Threads



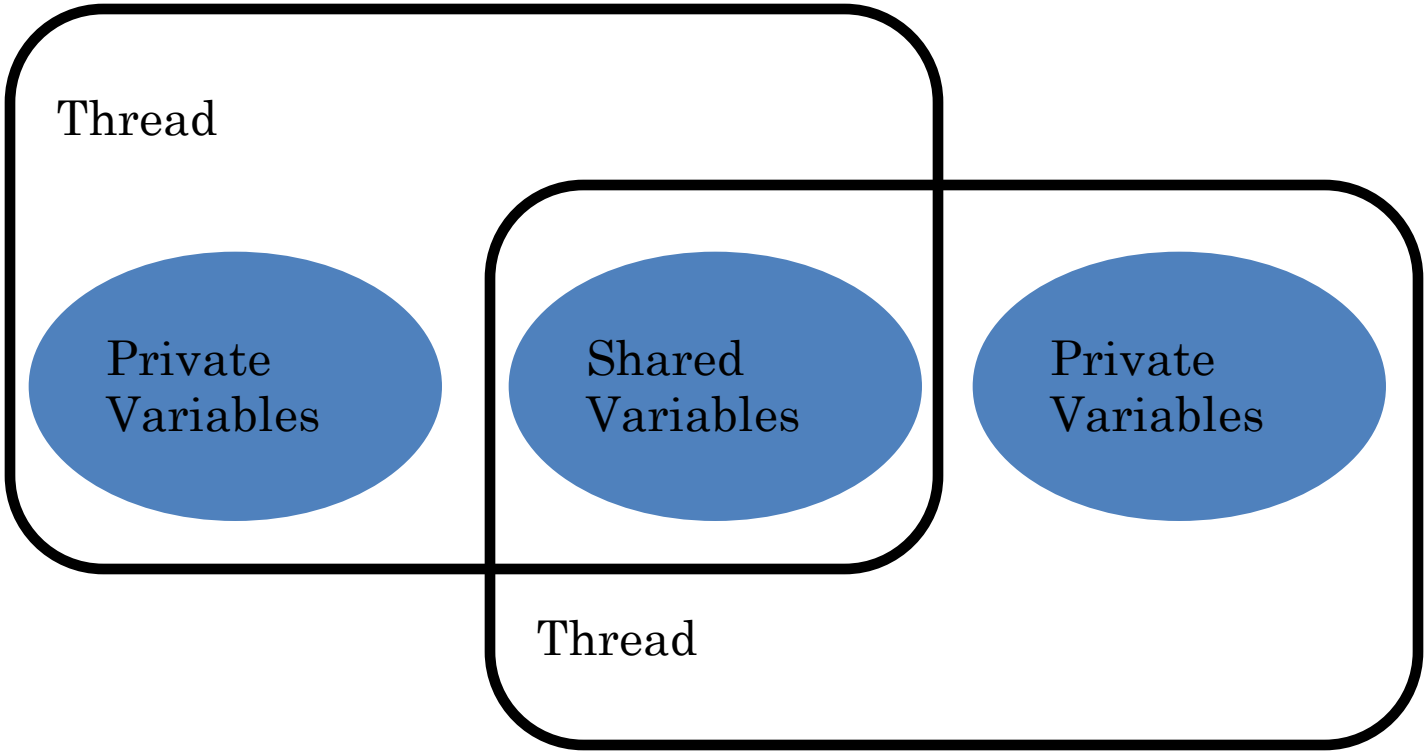
Task Decomposition Using Threads



Pipelining Using Threads



Shared versus Private Variables



Limitations of Parallel Computing

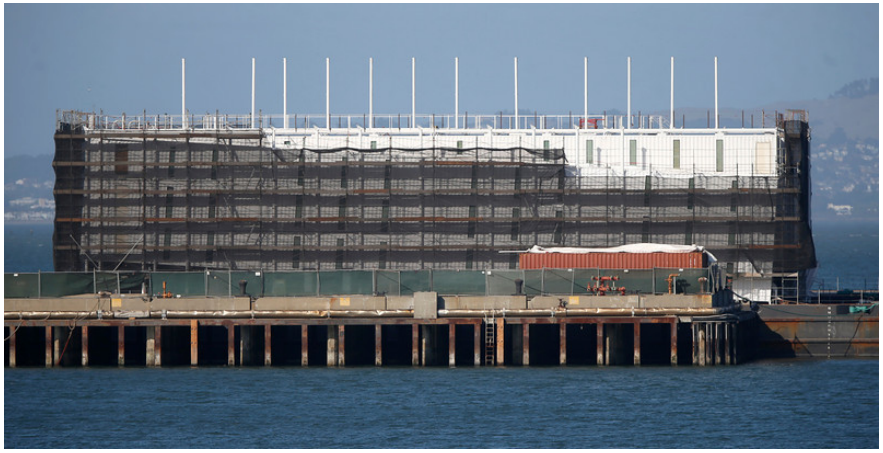
- Programming so many nodes concurrently remains a major barrier for most applications
  - Source code should be known & parallisable
  - Scalable algorithm development is not an easy task
  - User has to worry about message passing, synchronization and scheduling of his job
- Fortunately lot of free MPI codes and even parallel solvers are now available
- Debug is really HARD
- Still there is large gap between technology & usage as parallel tools are not so user friendly

Issues in building large clusters

- Scalability of interconnection network
- Scalability of software components
  - Communication Libraries
  - I/O Subsystem
  - Cluster Management Tools
  - Applications
- Installation and Management Procedures
- Troubleshooting Procedures

Issues in operating large clusters

- Space Management
  - Node form factor
  - Layout of the nodes
  - Cable routing and weight
- Power Management
- Cooling arrangements



Hamina, Finland

Tianhe-2

Sponsors	863 Program
Location	National Supercomputer Center, Guangzhou, China
Architecture	32,000 Intel Xeon E5-2692 12C with 2.200 GHz 48,000 Xeon Phi 31S1P
Power	17.6 MW (24 MW with cooling)
Operating system	Kylin Linux[1]

https://www.google.com/green/efficiency/datacenters/

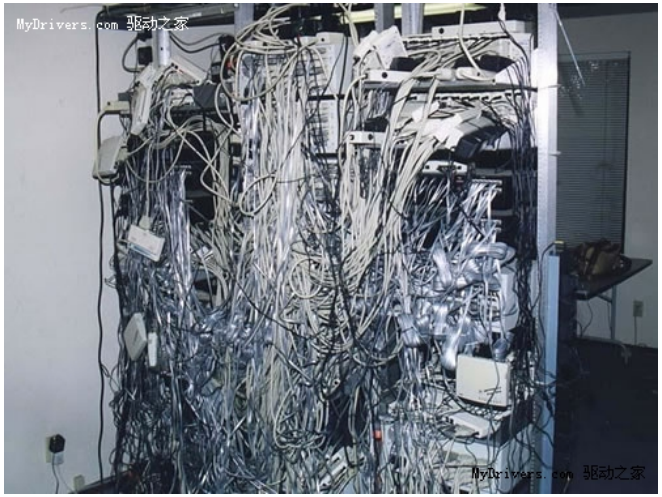
So, Green500.org

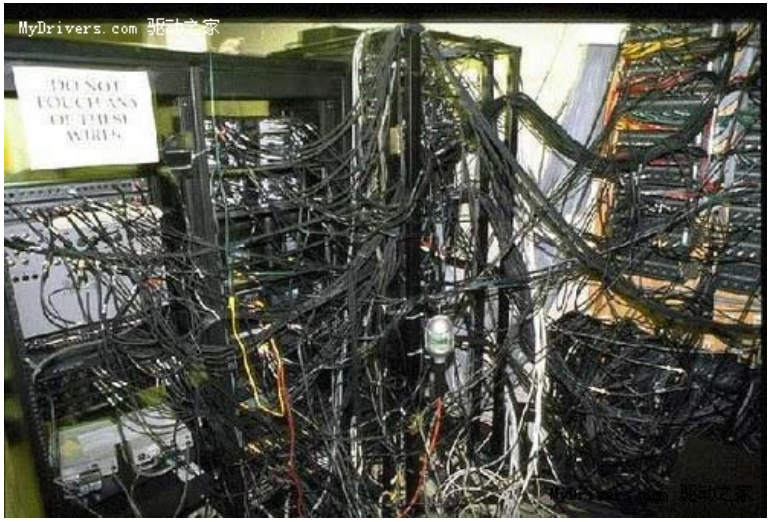
- performance per watt: measure the energy efficiency of a particular computer architecture or computer hardware.

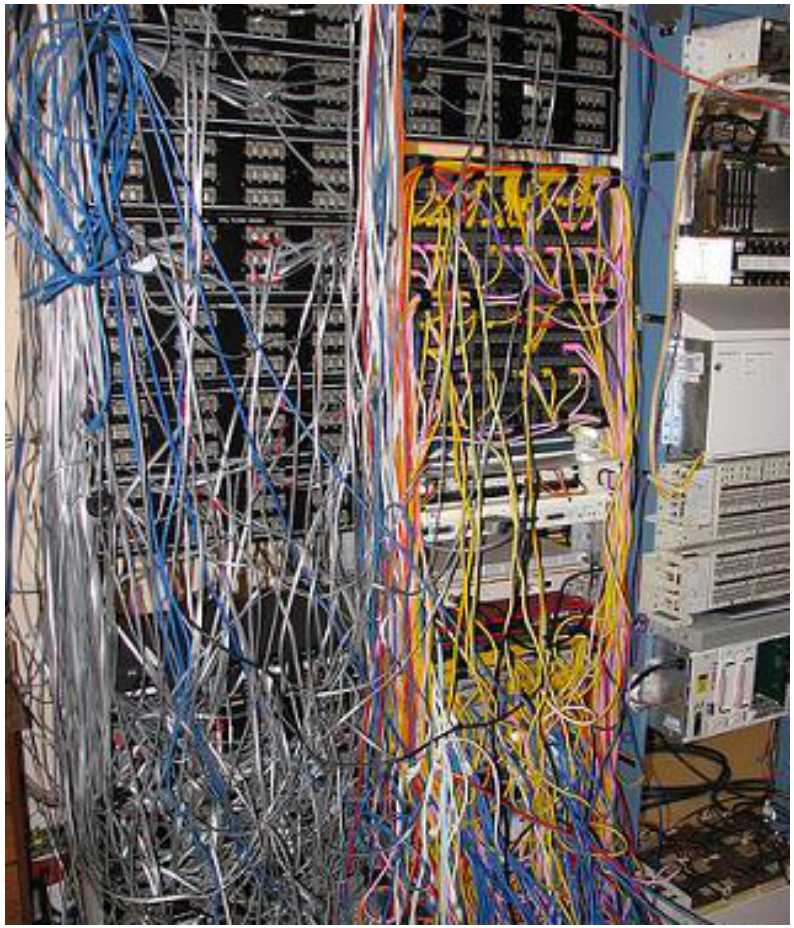
Listed below are the June 2015 The Green500's energy-efficient supercomputers ranked from 1 to 10.

Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	7,031.58	RIKEN	Shoubu - ExaScaler-1.4 80Brick, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband FDR, PEZY-SC	50.32
2	6,842.31	High Energy Accelerator Research Organization /KEK	Suiren Blue - ExaScaler-1.4 16Brick, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband, PEZY-SC	28.25
3	6,217.04	High Energy Accelerator Research Organization /KEK	Suiren - ExaScaler 32U256SC Cluster, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, PEZY-SC	32.59
4	5,271.81	GSI Helmholtz Center	ASUS ESC4000 FDR/G2S, Intel Xeon E5-2690v2 10C 3GHz, Infiniband FDR, AMD FirePro S9150	57.15
5	4,257.88	GSIC Center, Tokyo Institute of Technology	TSUBAME-KFC - LX 1U-4GPU/104Re-1G Cluster, Intel Xeon E5-2620v2 6C 2.100GHz, Infiniband FDR, NVIDIA K20x	39.83
6	4,112.11	Stanford Research Computing Center	XStream - Cray CS-Storm, Intel Xeon E5-2680v2 10C 2.8GHz, Infiniband FDR, Nvidia K80	190.00
7	3,962.73	Cray Inc.	Storm1 - Cray CS-Storm, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, Nvidia K40m	44.54
8	3,631.70	Cambridge University	Wilkes - Dell T620 Cluster, Intel Xeon E5-2630v2 6C 2.600GHz, Infiniband FDR, NVIDIA K20	52.62
9	3,614.71	TU Dresden, ZIH	Taurus GPUs - Bull bullx R400, Xeon E5-2680v3 12C 2.5GHz, Infiniband FDR, Nvidia K80	58.01
10	3,543.32	Financial Institution	iDataPlex DX360M4, Intel Xeon E5-2680v2 10C 2.800GHz, Infiniband, NVIDIA K20x	54.60

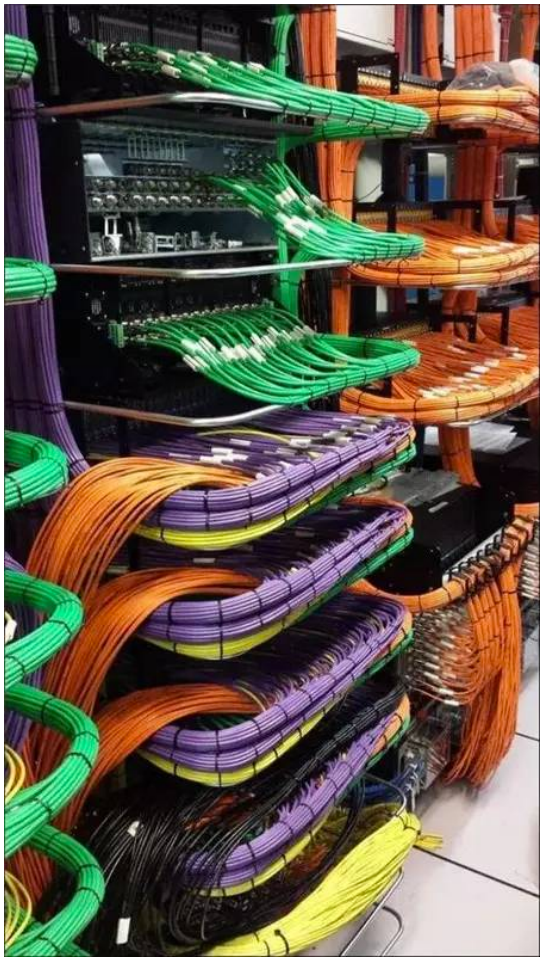
Wires



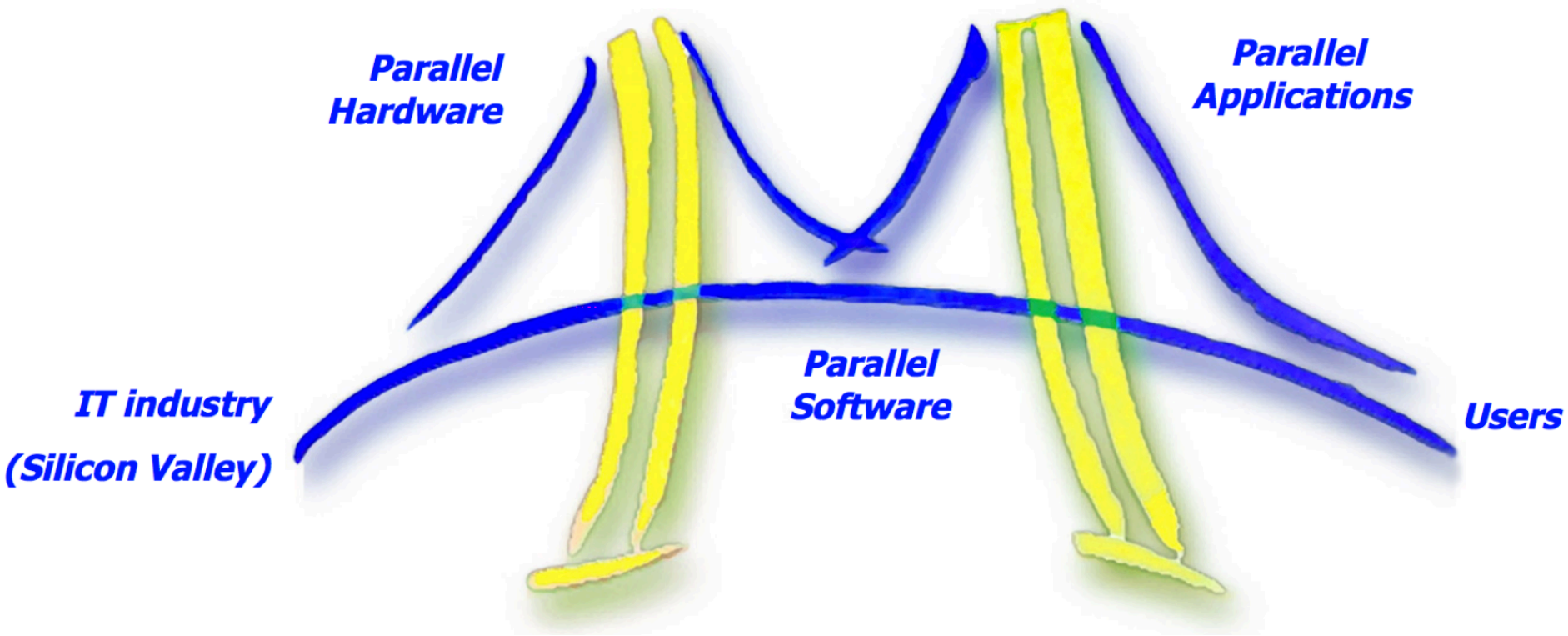




Wires



Berkeley View



A Lot of Questions to Answer

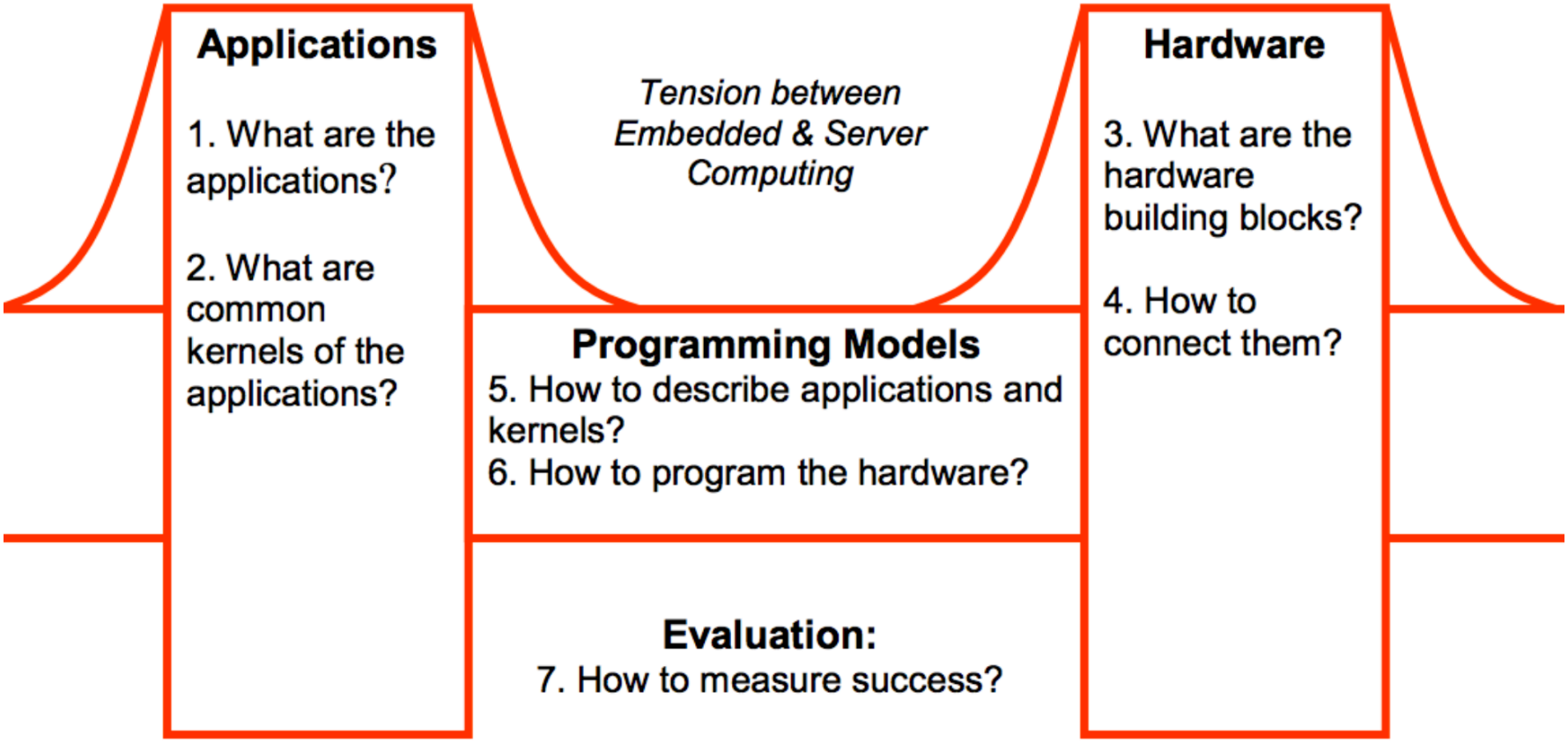


Figure 1. A view from Berkeley: seven critical questions for 21st Century parallel computing. (This figure is inspired by a view of the Golden Gate Bridge from Berkeley.)

From "The Landscape of Parallel Computing Research: A View from Berkeley"

HPC Benchmark Classics

- Linpack or HPL: Parallel LU factorization for solution of linear equations
- NPB version 1: Mainly classic HPC solver kernels
  - MG: Multigrid
  - CG: Conjugate Gradient
  - FT: Fast Fourier Transform
  - IS: Integer sort
  - EP: Embarrassingly Parallel
  - BT: Block Tridiagonal
  - SP: Scalar Pentadiagonal
  - LU: Lower-Upper symmetric Gauss Seidel

13 Berkeley Dwarfs

- Dense Linear Algebra
- Sparse Linear Algebra
- Spectral Methods
- N-Body Methods
- Structured Grids
- Unstructured Grids
- MapReduce
- Combinational Logic
- Graph Traversal
- Dynamic Programming
- Backtrack and Branch-and-Bound
- Graphical Models
- Finite State Machines

First 6 of these correspond to Colella's original.
Monte Carlo dropped.
N-body methods are a subset of Particle in Colella.

Note a little inconsistent in that MapReduce is a programming model and spectral method is a numerical method.
Need multiple facets!

References

- Parallel Programming Principle and Practice (http://grid.hust.edu.cn/courses/parallel/)
- Introduction to Parallel Computing (https://computing.llnl.gov/tutorials/parallel_comp/)
- Intel® Software Academic Program (https://software.intel.com/en-us/courseware/hpc)
- IDF Series Talks