Outlook for Parallel Computing in the Electric Power Industry

Siddhartha Kumar Khaitan

Electrical and Computer Engineering

Iowa State University

skhaitan@iastate.edu

PSERC Webinar

March 19, 2013

Presentation Overview

- Acceleration using high performance (parallel) computing (HPC) techniques
  - Basics of HPC and parallelization paradigms (shared memory and distributed computing)
  - Parallelization approach (task-level and data-level) and parallel solvers
  - Applications of HPC (Dynamic security assessment)
- Addressing resource-usage efficiency in HPC using task-scheduling techniques
  - Static and dynamic techniques (Master-Slave, Work-Stealing, etc.)
- State-of-the-art HPC languages and their unique features
  - C/C++ (MPI, Cilk, OpenMP, Pthread, Hybrid)
  - D, JAVA, GO, CHAPEL, X10
  - *Which one suits your needs?*
- Porting legacy code on HPC platforms
- TDPSS: A Scalable Time Domain Power System Simulator for Dynamic Security Assessment
- Results

The Importance of Using HPC

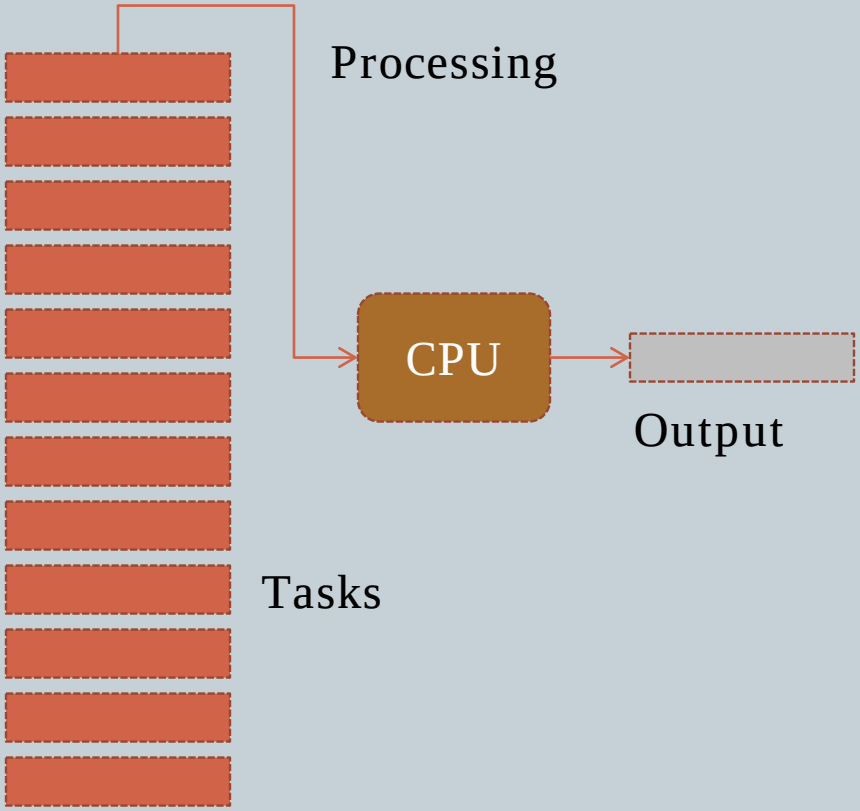
- Most modern day applications are extremely data- and/or compute-intensive.
- Example 1: Consider *N*-*k* contingency analysis with *N*= 12000, *k* =1, *k*= 2 and 10 seconds/contingency.

  =>Serial execution ~33 hrs and ~23 yrs !
- Example 2: PJM does security assessment for 3,000 contingencies in 15 minutes with 40 processors
- Example 3:YouTube serves 100 million videos/day!
- Example 4: Every month 3 billion photos are uploaded to Facebook!

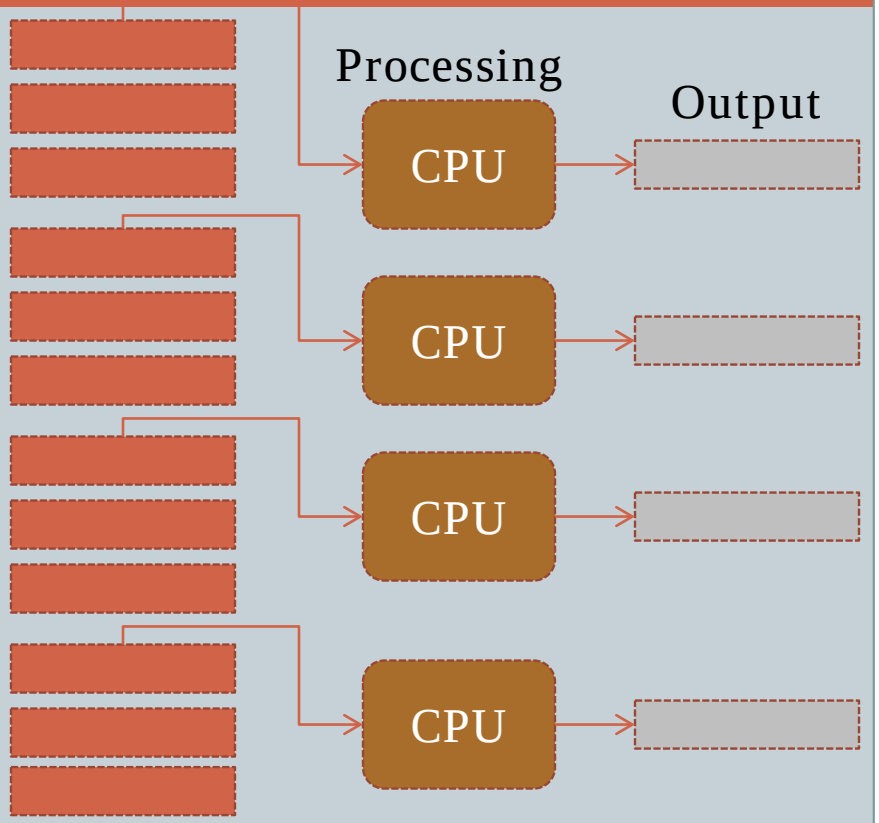
- Parallel computing (HPC) is an essential computation paradigm for today's applications.

Serial vs. Parallel Computing

Serial Computing



Parallel Computing



Parallel computing provides much higher throughput!

Benefits of Parallel Computing

- By Moore's law, transistors/chip are increasing, but due to power limits, it is very difficult to make a single processor faster.

- Parallel computing provides saving of time/money
- Ability to solve larger problems in same time
- Ability to distribute problems over large number of processors, which may be situated remotely

Parallelism is the future of computing!

Requirements for Parallel Computing

Hardware

- Parallel hardware for performing parallel computations e.g. multi-core CPU, GPU (graphics processing unit) or FPGA (field-programmable gate array).

Software

- Significant part of the computation should be parallelizable to get good speedups (Amdahl's law)
- Minimal communication and synchronization
- Scheduling algorithms
- Specialized parallel programming languages

Examples of HPC Applications

- Bioinformatics
- Particle physics
- Aerospace
- Defense
- Telecommunication
- Power systems
  - ISO New England –for robust unit commitment evaluation
  - GE Energy – for improving PSLF simulation performance and capability
  - Hydro Quebec – uses the platform provided by OPAL RT technologies for operation and design
  - LLNL (Lawrence Livermore National Laboratory) - for research
  - PNNL (Pacific Northwest National Laboratory) - to enhance energy infrastructure and operations
- Walmart, FedEx, Motorola, Whirlpool, Portland Cement Association, etc.

HPC is available through **cloud** (e.g. Amazon aws.amazon.com/hpc-applications/), Penguin Computing (penguincomputing.com), IBM, etc. HPC Systems are provided by IBM, Intel, etc., and installed by Penguin Computing, Dell, etc.

- Here we focus on dynamic security assessment in power systems.

Power System Security Assessment (SA)

- SA is important for avoiding overloads, voltage instability, transient instability, cascading outages, and blackouts
- The service cost of one hour of downtime in credit card authorization is $2,600,000!
- To avoid it, contingency analysis is performed.
- Analyzing a large number of contingencies requires high computation power
- Parallelization and HPC techniques necessary to get high throughput.

Issues in Parallel Computing Programming

- Memory Architectures (Shared, Distributed, Hybrid)
- Programming Models (Shared, Distributed, Hybrid)
- Type of Parallelization (Data/Task)
- Load Balancing
- Synchronization/Communication
- Programming Languages
- Memory per Core
- Latency/Bandwidth
- Cost (Price of parallel processors (servers up to 16 cores) ranges between $400 to $4,700)
  - http://www.cpu-world.com/Price_Compare/Server_CPU_prices_%28latest%29.html
  - http://www.newegg.com/Processors-Servers/SubCategory/ID-727

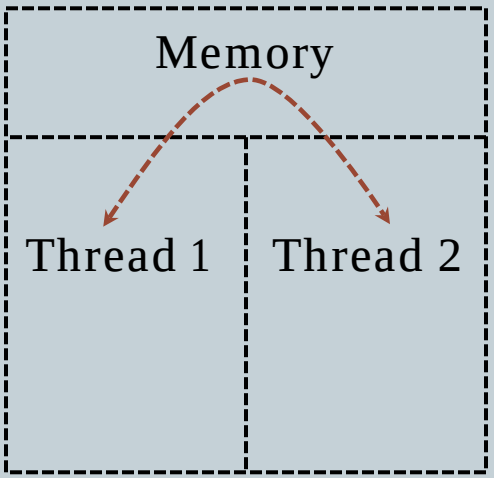
Parallelization Paradigms

How parallelization is implemented

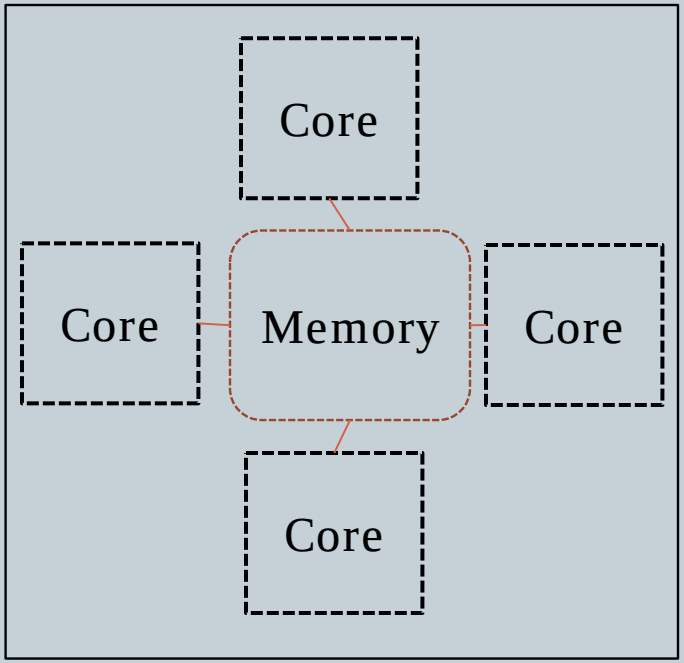
- Shared memory
  - Different processors/threads share main memory
- Distributed memory (distributed computing)
  - Each processor has its own memory
- Hybrid approach
- PGAS (Partitioned Global Address Space)

Shared Memory Computing

- Different cores/threads share memory
- Example: multithreading in languages such as Java, D, OpenMP, Go, Cilk.



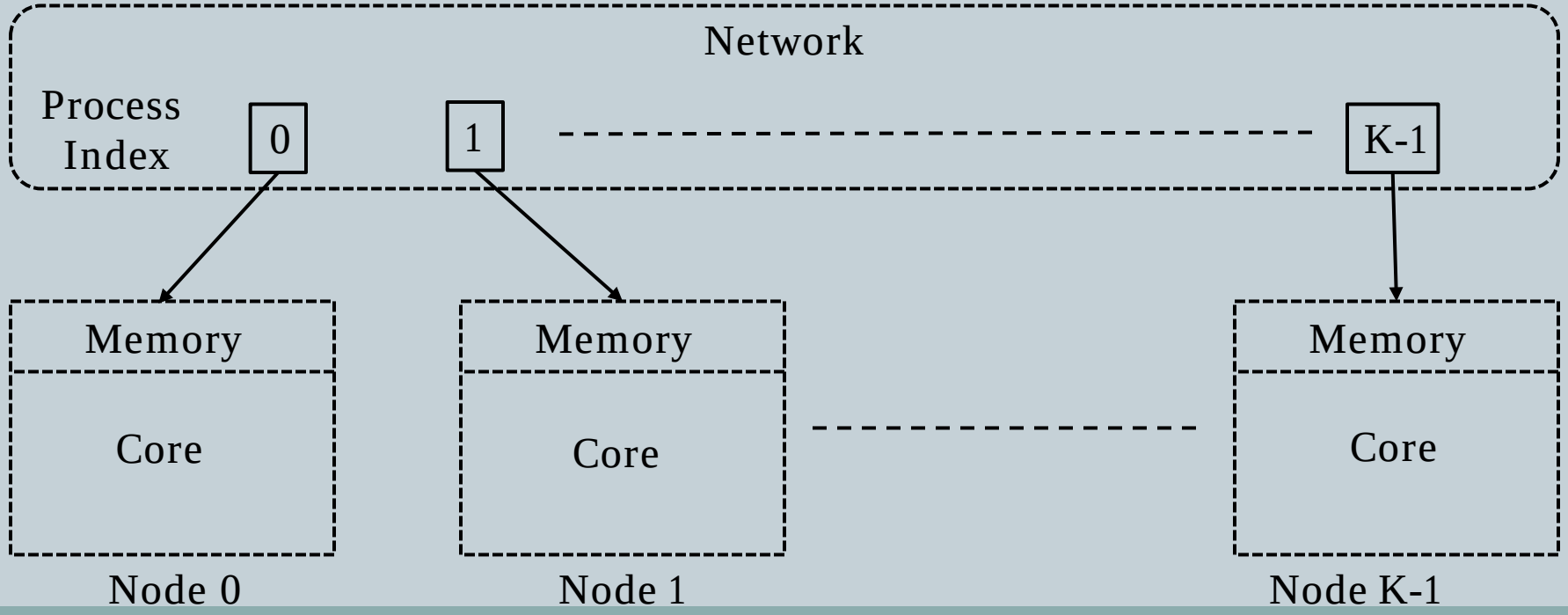
Threads share memory



Cores share memory

Distributed Computing

- Different processors use different memory spaces and communicate with each other through messages
- Example: MPI (Message Passing Interface).

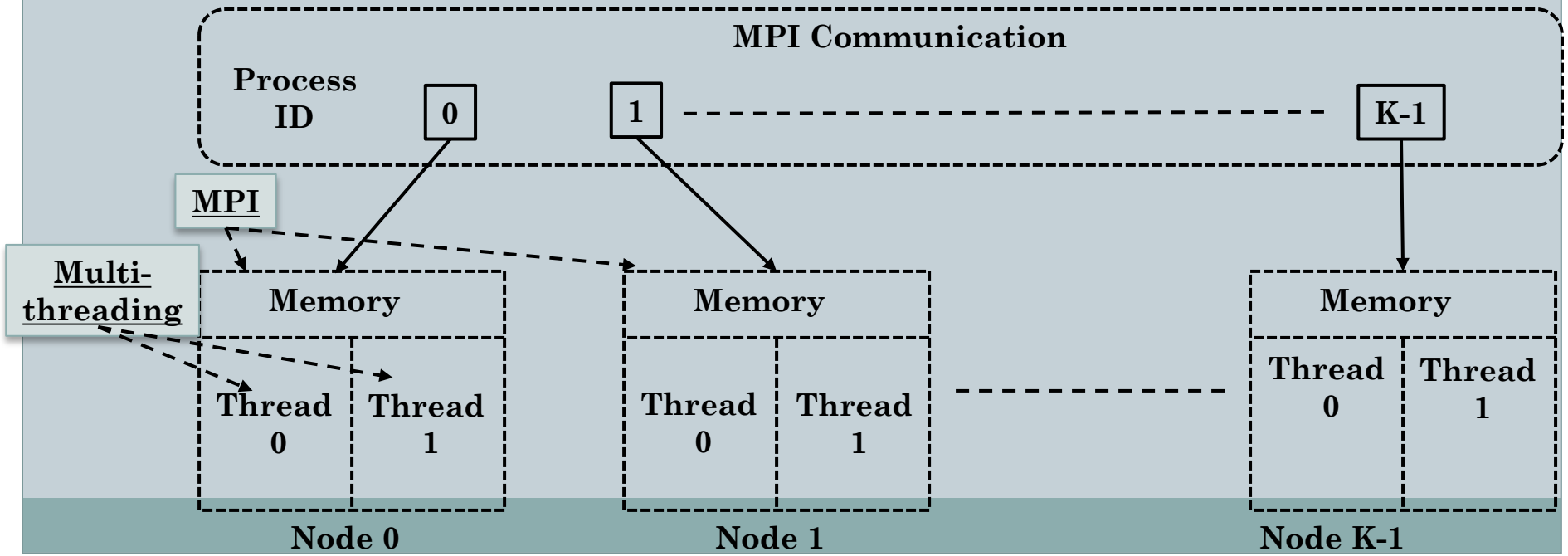


A Comparison

Distributed Computing	Shared Memory Computing
• Easier to scale to tens or thousands of processors (e.g. supercomputer). • Sharing is through explicit communication • Latency between communication nodes is a prime concern	• Difficult to scale to large number of cores. • Maintaining integrity of shared data is challenging. Need of locks, mutex, etc. • Low latency of data sharing

Hybrid Approach

- Shared memory computing on single processor and distributed computing across processors.
- Example: multithreading in single processor, with MPI across processors



PGAS (Partitioned Global Address Space) Memory Architecture

- Shared memory approach does not scale well beyond tens of cores while distributed memory approach with message passing incurs overhead of communication
- PGAS assumes a global memory address space that is logically partitioned
- A portion of the memory is local to each process or thread.
- Portions of the shared memory space may have an affinity for a particular process, thereby exploiting locality of reference.

A Comparison

	# of Threads	# of Memories	Non-local Access Supported
Serial	1	1	N/A
Shared (OpenMP)	p	1	N/A
Distributed (MPI)	p	p	No. Message passing reqd.
PGAS	p	q	Yes

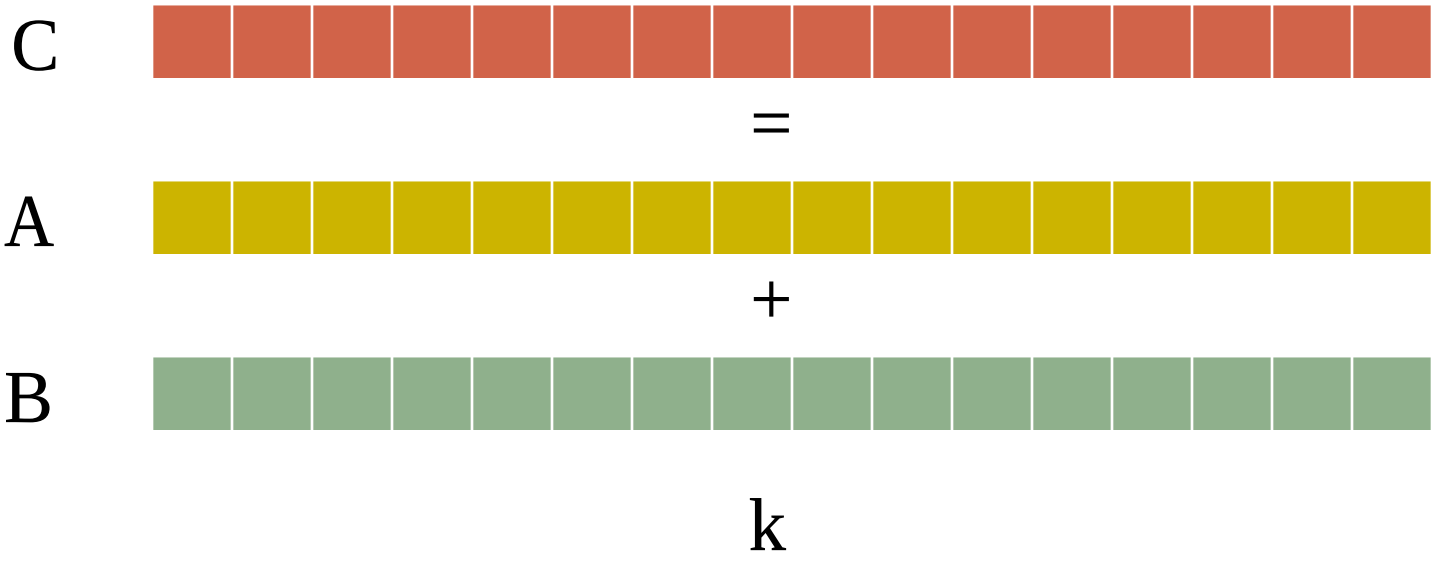
A thread can access the memory at other process without message passing!

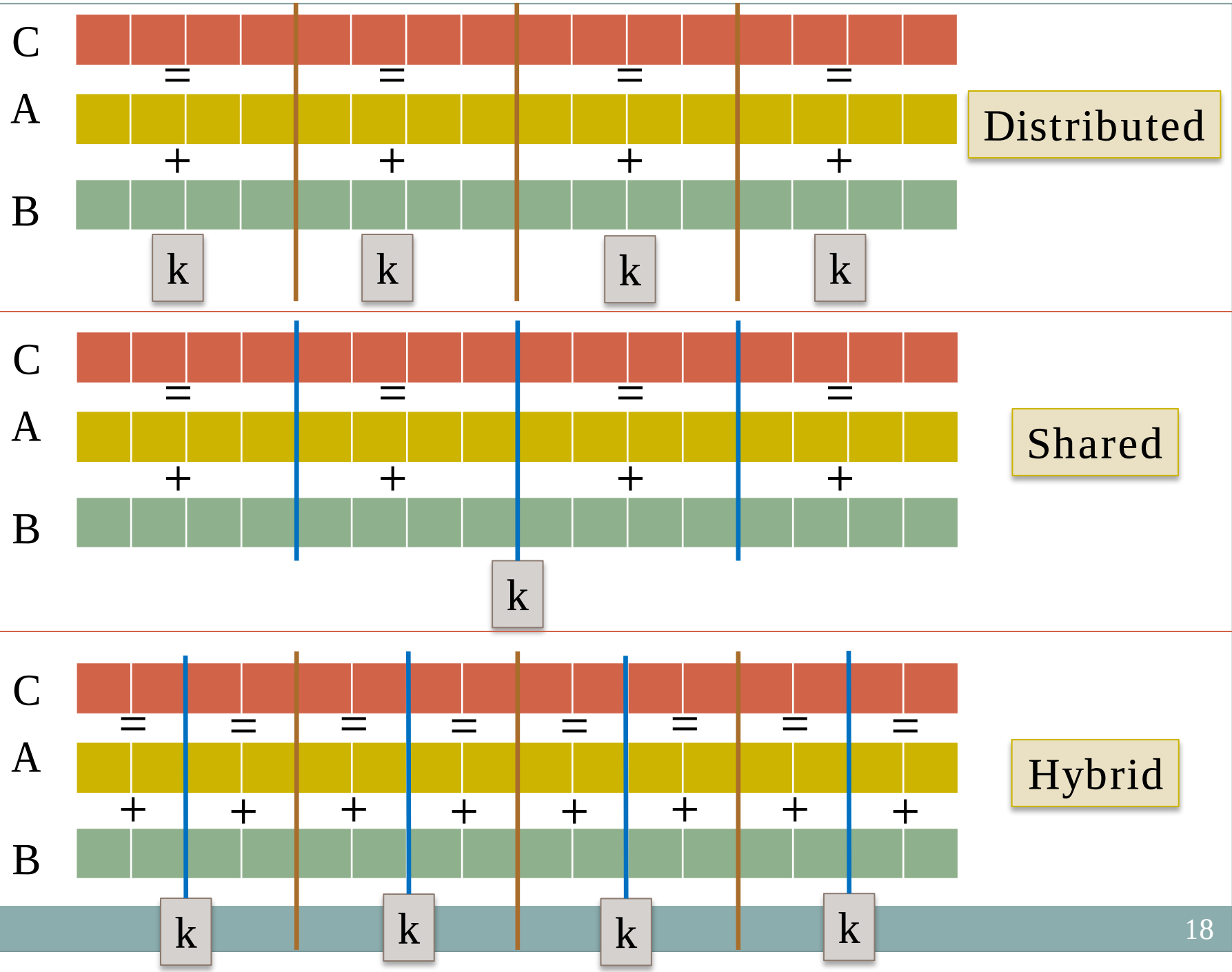
An Example for Visualization of Parallel Programming Paradigms

Assume a computation

$C = A + kB$

where k is scalar and A, B and C are vectors




C
A
B
k
k
k
k
Distributed
C
A
B
k
Shared
C
A
B
k
k
k
k
Hybrid

Parallelization Approaches

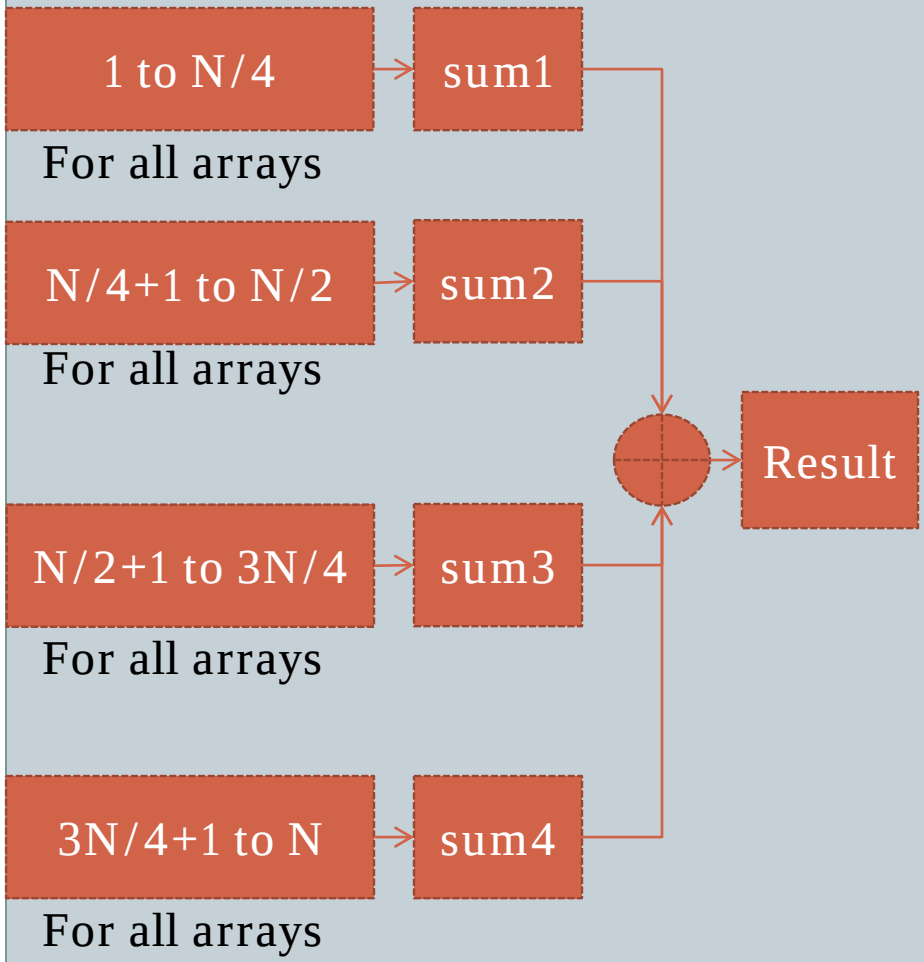
Task-Level Parallelization

- Different tasks are given to different processors.
- 10,000 contingencies and 4 processors: 2,500 contingencies to each processor.

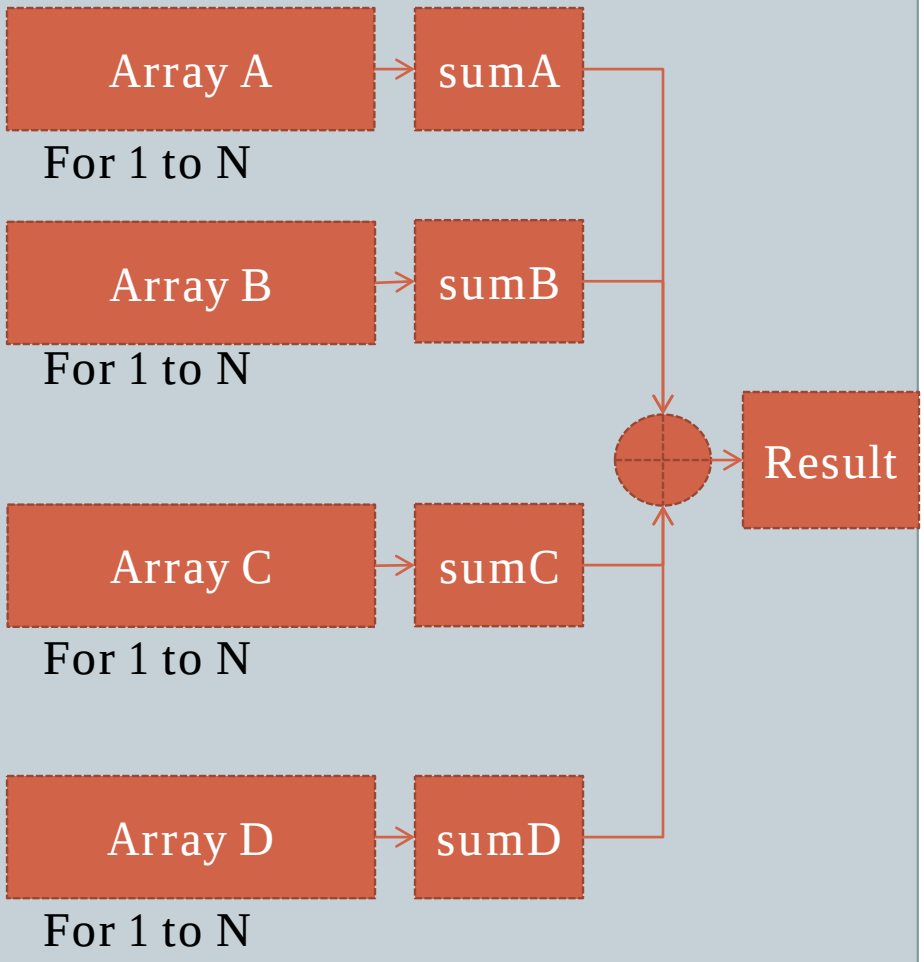
Data-Level Parallelization

- Different phases or data portions processed by different processors.
- An array of 10,000 elements and 4 processors: 2500 elements to each processor.

Example: Summing 4 Arrays (A, B, C, D) of Length N



Data Parallelism



Task Parallelism

Communication

1. Should be minimum
2. Most parallel problems require communication
3. Cost of communication should be low
4. Best if communication overlaps with computation.

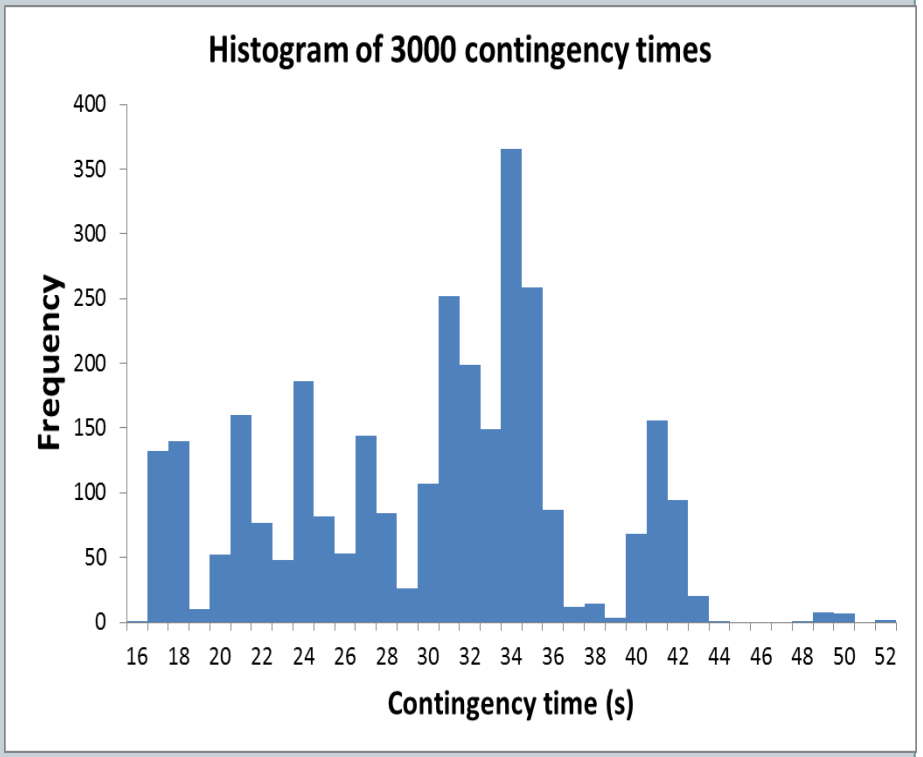
Synchronization

1. Should be minimum to allow maximum independent progress and avoid dependencies
2. Barriers and locks are used to enforce synchronization to protect shared data
3. Lack of it may lead to violation of shared data and wrong results

Task-Scheduling Techniques for Addressing Resource-Usage Efficiency in HPC

- Static scheduling technique
- Dynamic scheduling technique
  - Master-Slave scheduling (MSS)
  - Work-Stealing scheduling

Example: Variation in Contingency Simulation Time

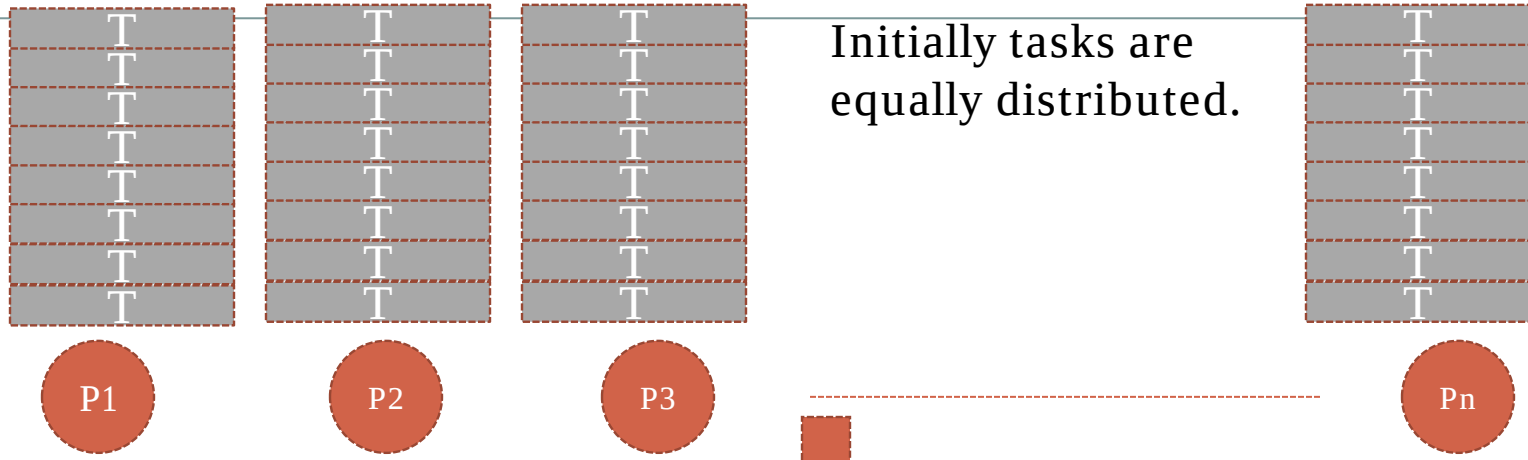


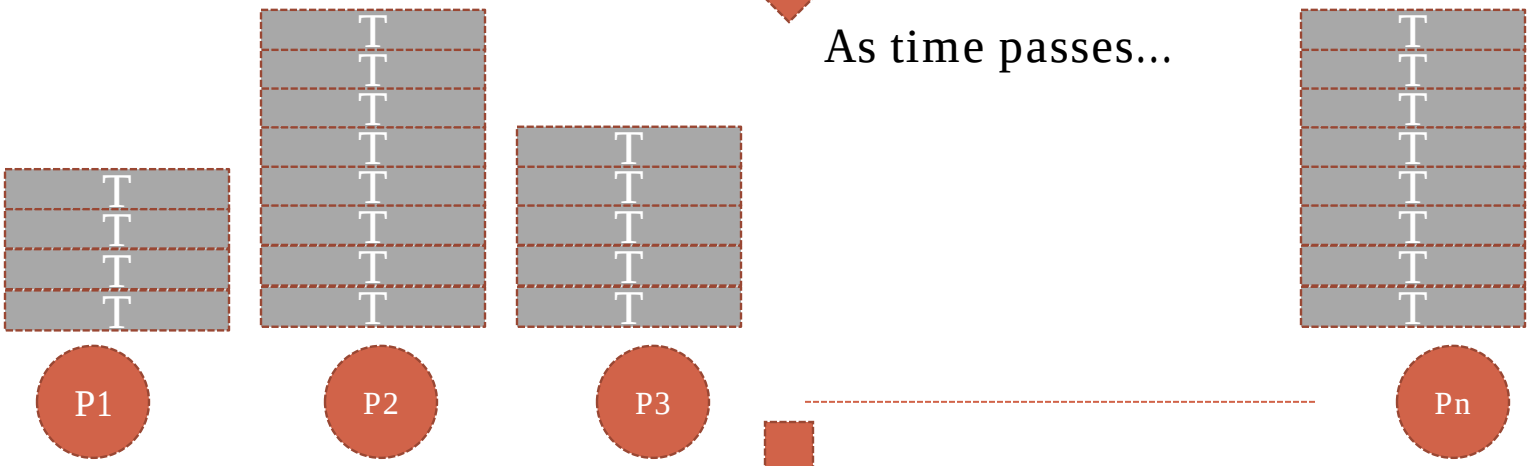
Static Scheduling

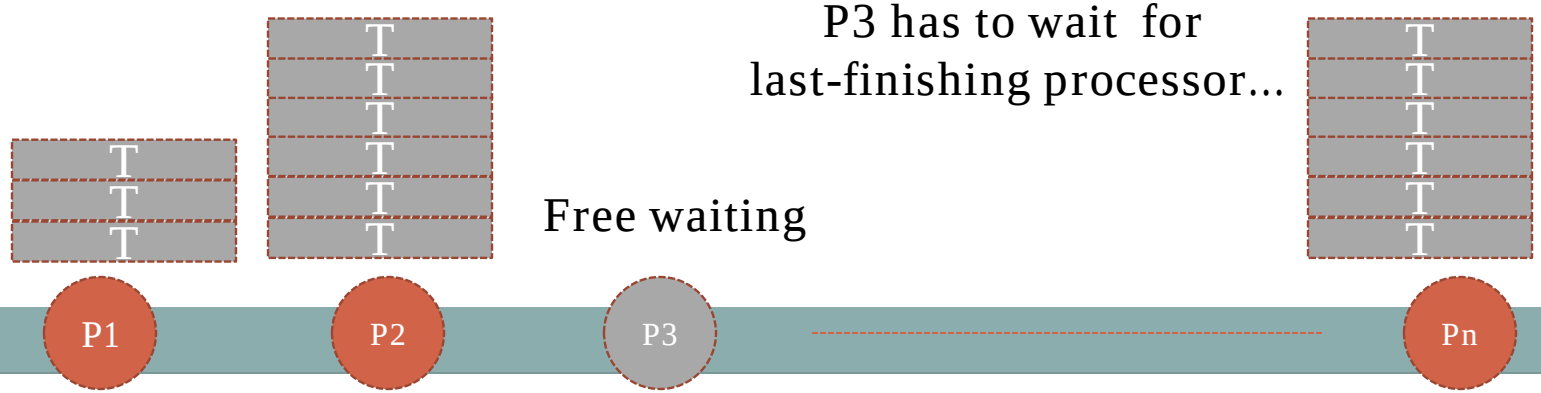
Main Idea:

- Statically assign tasks to available processors.
- The finish time of the schedule is the time when the last job finishes.

Reference: gridoptics.pnnl.gov/docs/3_Khaitan.pdf


Initially tasks are
equally distributed.
P1
P2
P3
Pn


As time passes...
P1
P2
P3
Pn

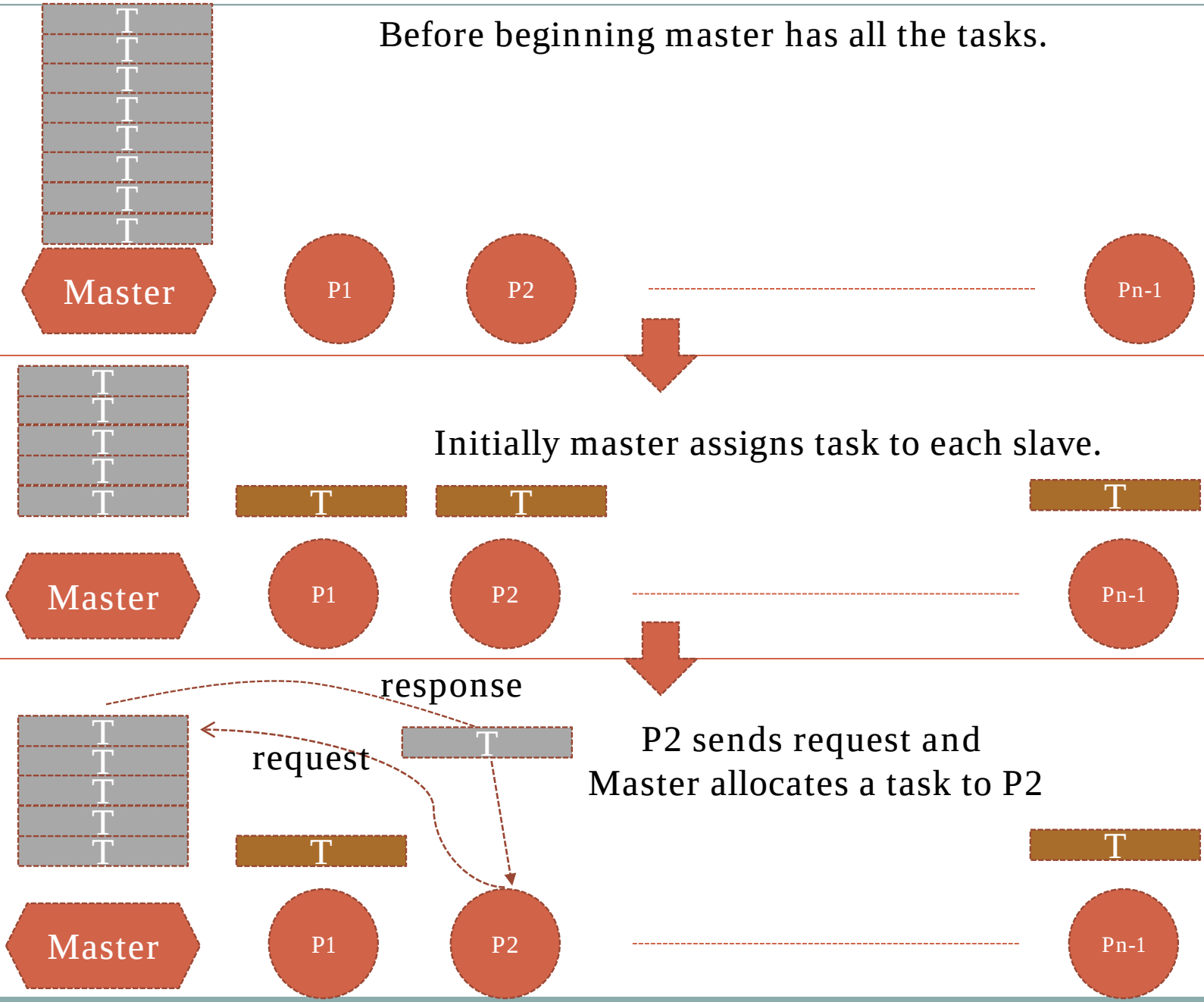

P3 has to wait for
last-finishing processor...
Free waiting
P1
P2
P3
Pn

Master-Slave Scheduling

Main Idea:

- One processor is used as the master and others as slaves.
- Master assigns tasks to each of the slaves.
- When a slave finishes a task, it requests new task from the master.
- The finish time of the schedule is the time when the last job finishes.

Reference: gridoptics.pnnl.gov/docs/3_Khaitan.pdf



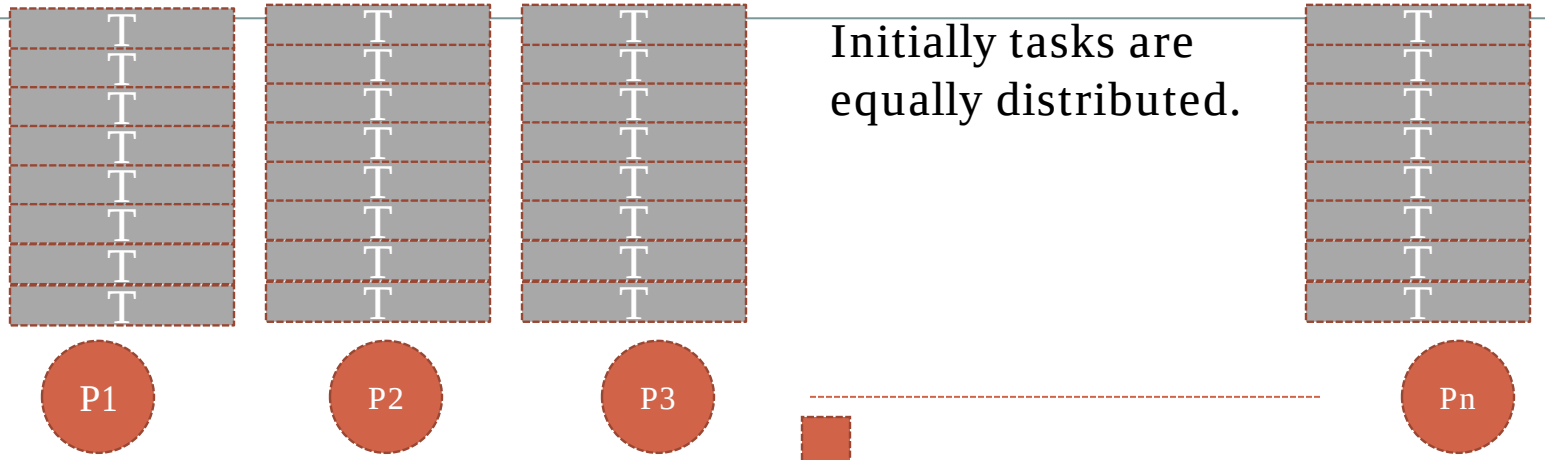
Multiple slaves may demand tasks=> contention. Locking required

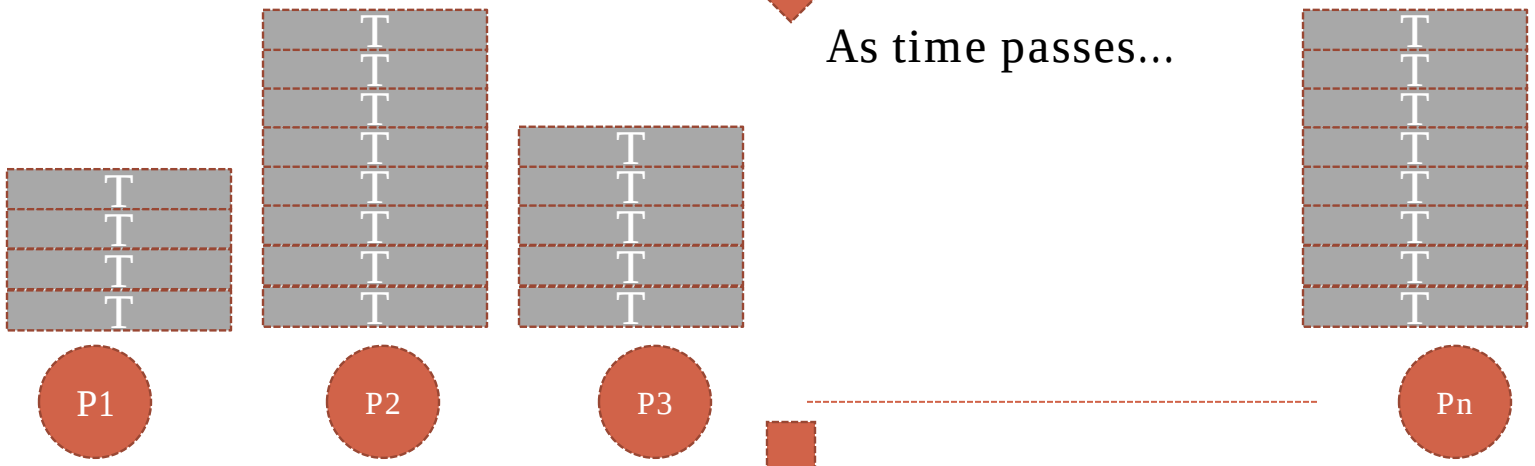
Work-Stealing Based Scheduling

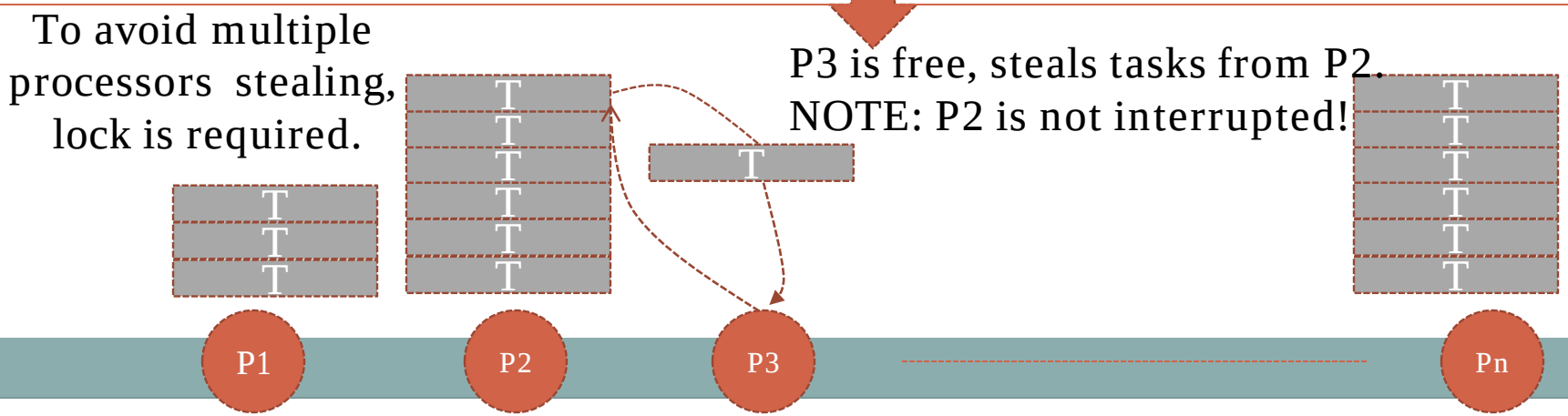
Main Idea:

- All tasks hold task-queues and start their work.
- A free node (“thief”) steals tasks from another node (“victim”), which has excess tasks.
- Uses double-ended queue: stealing request can be addressed without waiting for finishing of current task.
- Efficient in space, time and communication overhead[1].

[1] R. Blumofe,C. Leiserson, “Scheduling multithreaded computations by work stealing”, JACM 1999.
[2] gridoptics.pnnl.gov/docs/3_Khaitan.pdf


Initially tasks are equally distributed.
P1
P2
P3
Pn


As time passes...
P1
P2
P3
Pn


To avoid multiple processors stealing, lock is required.
P3 is free, steals tasks from P2.
NOTE: P2 is not interrupted!
P1
P2
P3
Pn

Static Scheduling

Advantages

- No overhead of scheduling
- Good if tasks lengths equal

Disadvantages

- Very poor load balancing in worst-case
- Free processors have to wait when done

Master-Slave Scheduling

- Overcomes limitation of static scheduling
- No communication b/w slaves => low overhead

- Master becomes busy and performs no useful work
- If multiple slaves request from master simultaneously=> contention.

Work-Stealing Scheduling

- No contention at master
- No wastage of a processor

- Each processor can communicate with any other processor: special topology required.
- Termination-detection more challenging.

State-of-the-Art HPC Languages and their Unique Features

Several languages facilitate writing parallel programs. They have different unique features and limitations:

- C/C++ (MPI, Cilk, OpenMP, Pthread)
- JAVA
- D
- GO
- CHAPEL
- X10

Which one suits your needs?

MPI

- Distributed computing
- Highly scalable, used in large clusters and supercomputers
- Open source, based on C++ or Fortran
- De facto standard in industry
- Limitations:
  - Overhead of message passing
  - Does not provide global view of memory

Code Snippet

```
MPI_Init (&argc, &argv); // starts MPI

 MPI_Comm_rank (MPI_COMM_WORLD, &rank);
            // get current process id

 MPI_Comm_size (MPI_COMM_WORLD, &size);
            // get number of processes

 printf( “From process %d of %d\n", rank, size );

 MPI_Finalize();
```

OpenMP

- Multithreaded programming
- Uses mostly compiler directives.
- Advantage of incremental programming without disturbing existing code => Easy to debug
- Very useful for parallelizing legacy code, Open source
- Easy to learn since it is based on C++ or Fortran
- Limitations:
  - Does not scale to hundreds of cores

Code Snippet

```
#pragma omp parallel for
for (int i = 0; i <= 10000; i++)
  {
   doWork(i); //all instances of doWork(i) run concurrently
  }
```

X10 (From IBM)

- Aims to improve productivity and portability of high-end computing systems
- Open source, Object-oriented
- Higher level programming model than MPI
- Supports PGAS
- Limitations:
  - Still being developed

Code Snippet

```
finish  // wait till all inside functions have finished
{
for(i =0; i< 10; i++)
{
   val ii = i;
   async  doWork (i); //all instances of doWork(i)
run concurrently
}
}
```

Comparison of Languages

	Language/Add-on Library	Paradigm	Garbage Collection	Open-Source	VM/Native
Cilk	Library	Shared	N/A	No	Native
MPI	Library	Distributed	N/A	Yes	Native
OpenMP	Library	Shared	N/A	Yes	Native
Pthread	Library	Shared	N/A	Yes	Native
Java	Language	Shared	Yes	Yes	VM
D	Language	Shared	Yes	Yes	Native
Go	Language	Both	Yes	Yes	Native
Chapel	Language	Both	No	Yes	Native
X10	Language	Both	Yes	Yes	Both possible

This is nice but… How to port *my* legacy application code to HPC?

Challenges of Porting Legacy Code to HPC

- Legacy code are generally written in C/Fortran
- They are large (e.g. millions of lines of code)
- They are complex (e.g. mathematical software)
- Rewriting the code in a new language may introduce bugs!
- Time and money overhead of porting may be huge!
- Maintaining legacy code requires technical experts

Thus, porting needs to be done carefully!

Porting Legacy Application to HPC

- **Method 1**: Direct code integration and interfacing with parallelization routine.

- **Method 2**: Using program binary as a task in the parallelization routine.

We now compare their relative advantages...

A Comparison

Direct code integration

- Initialization & finishing are done only once
- May not be possible on some platforms
- Requires much more effort to implement, esp. for large codes
- Summary: Efficient but requires large effort

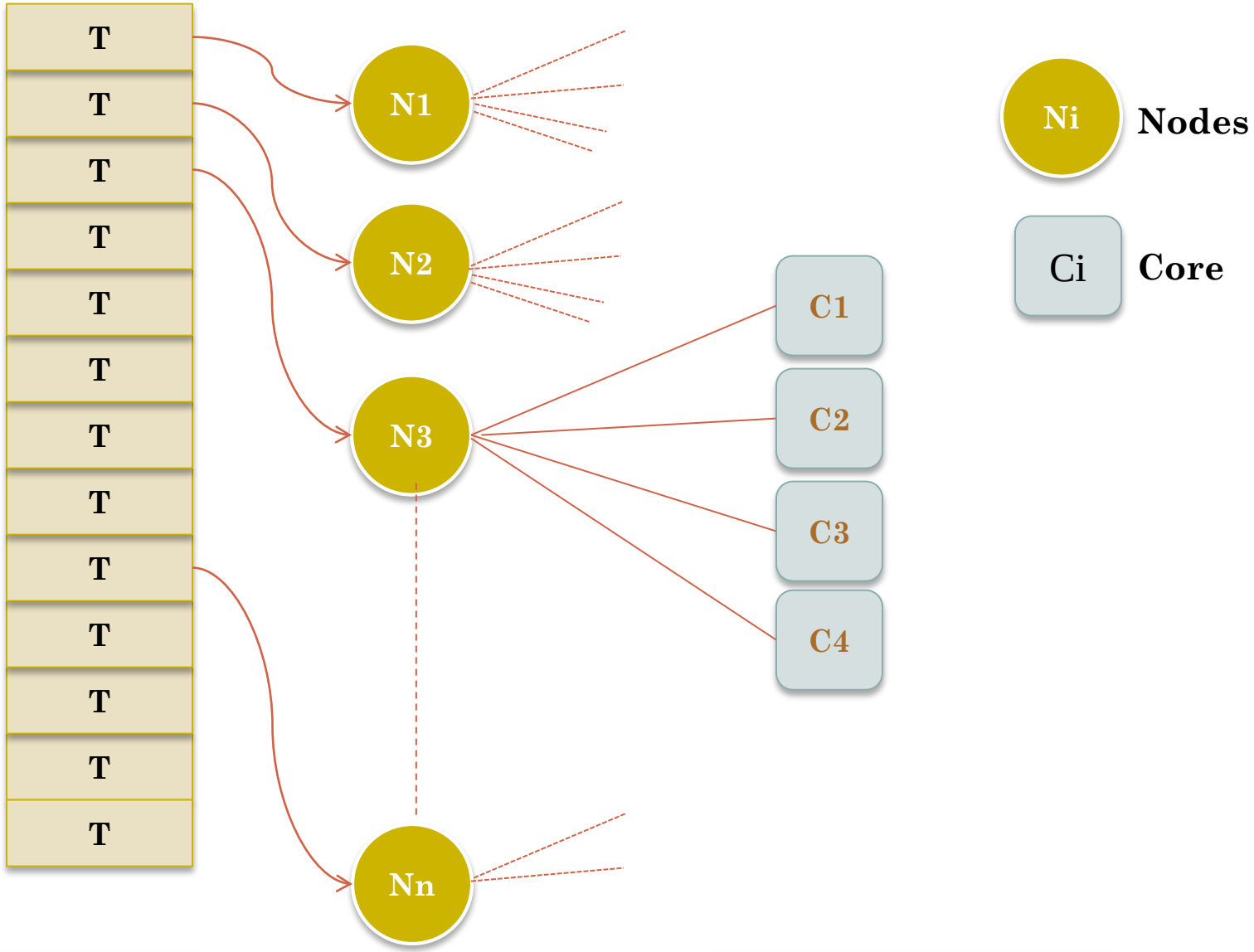
Using program binary

- Initialization & finishing are done each time
- Possible on most platforms
- Easy to implement
- Summary: Less efficient but easy to implement

TDPSS: A Scalable Time Domain Power System Simulator for Dynamic Security Assessment

- A research grade simulator for steady state and dynamic contingency analysis
  - Provides models for different power system components
  - Provides different numerical solvers
  - Designed with object oriented programming
  - Validated against commercial software packages
  - Allows easy exploration!
- Parallelization
  - For the solution of a single contingency (Data Parallelism)
  - Across multiple contingencies (Task Parallelism)

Hierarchical Design of the Parallel TDPSS Simulator

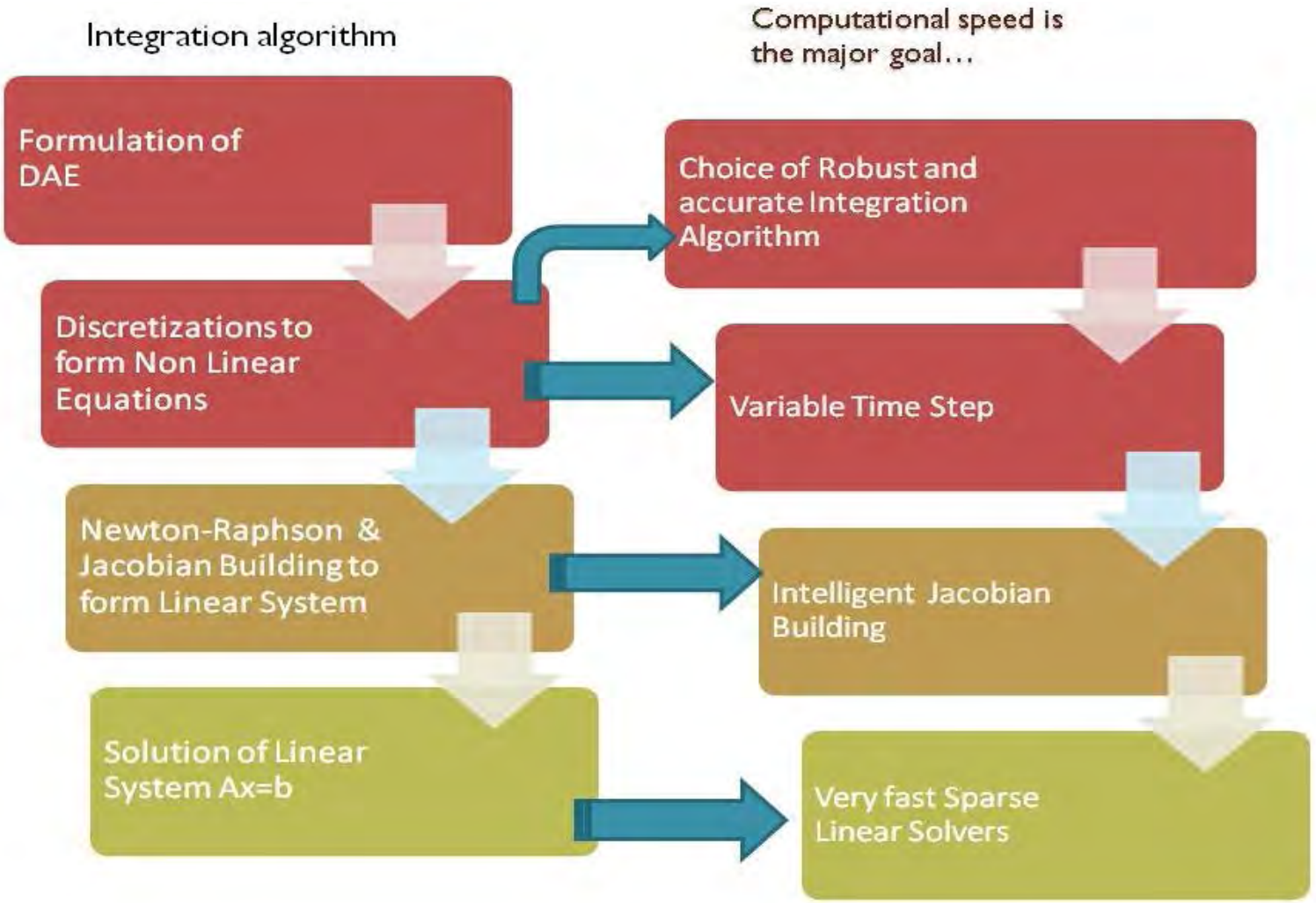


Contingency List

Contingency Parallelization

Parallel DAE Solution

Computational Steps in Time Domain Simulation



Results for Different HPC Characteristics

- Memory Architectures (shared, distributed, hybrid)
- Programming Models (shared, MPI)
- Type of Parallelization (Data/task)
- Load Balancing
- Synchronization/Communication
- Programming Languages

Data Parallelism

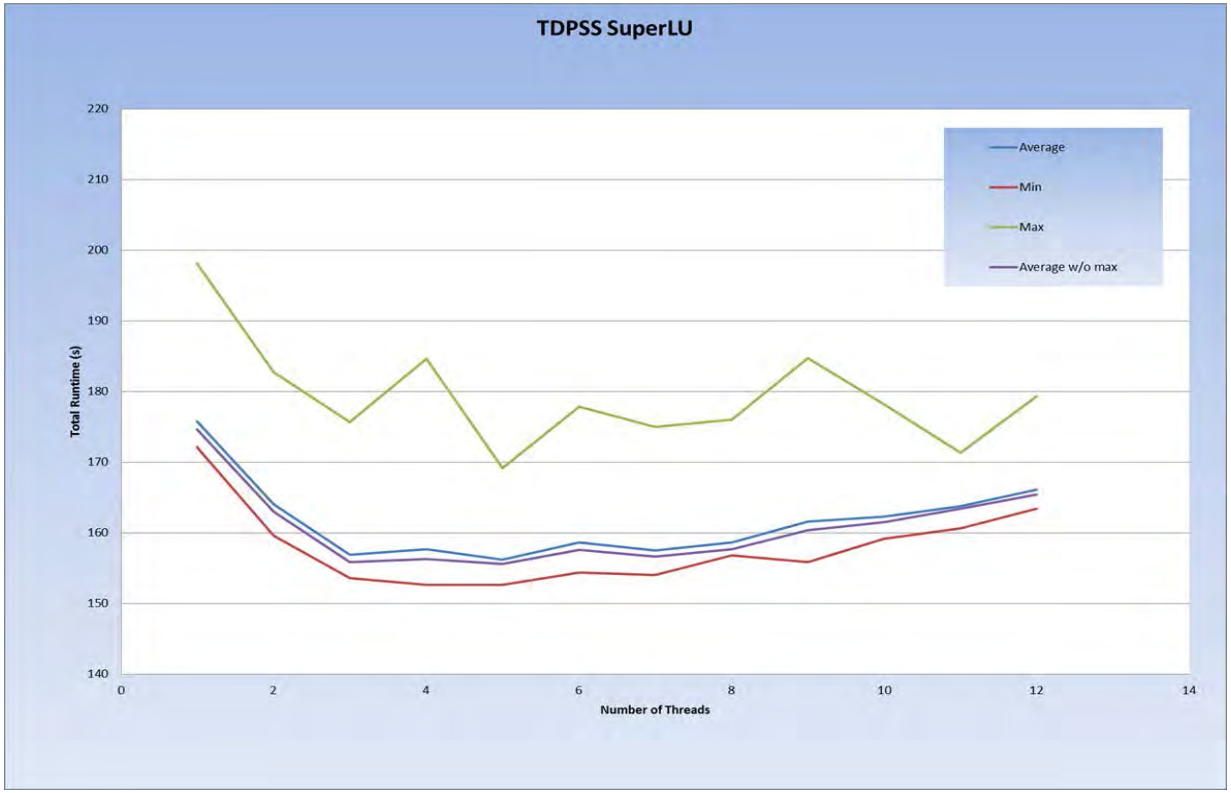
Parallel linear solvers and integration methods are key.

Linear solvers	Parallelism		Integrator		
	Multithread on dual-core machine	Distributed on HPC cluster	Trapezoidal	IDAS	Explicit
KLU			√	√	√
UMFPACK			√	√	√
SuperLu	√	√	√	√	√
WSMP	√	√	√	√	√
Pardiso	√	√	√	√	√
MUMPS		√	√	√	√

Initial Results with threaded SuperLU_MT

SuperLu 2.0 with pthreads

Ansel Linux at LLNL w/ 1 node w/ 2 x 6 core Intel Xeon



Clear benefit from SuperLU_MT with 2-5 threads; problem size too small for using more

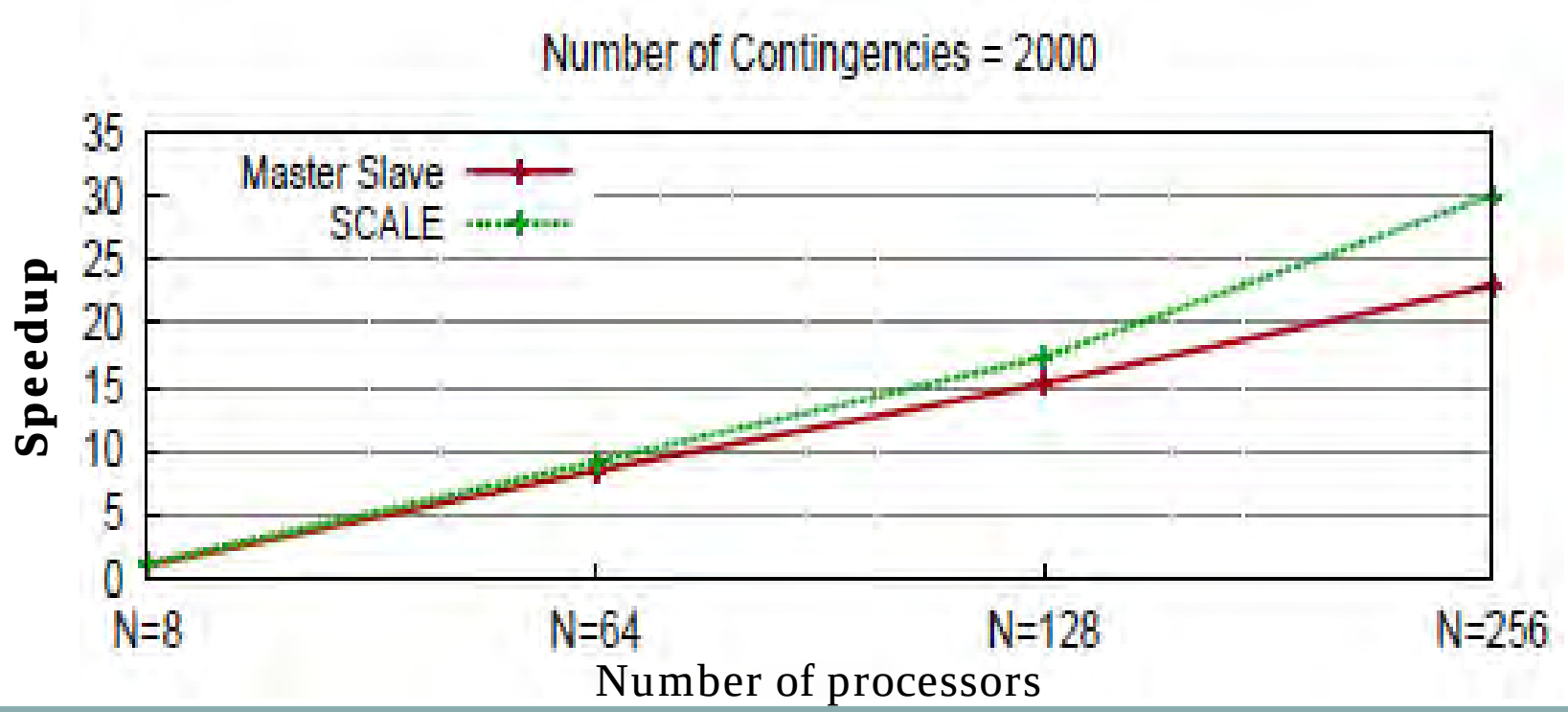
Task Parallelism (Parallelism by Contingency)

- Intra-contingency parallelism (Linear Solver)
- Distributed computing (Cystorm Super Computer)
  - Distributed (MPI)
  - Hybrid (MPI+Ptheads)
  - PGAS (X10)
- Shared memory multicore computing
  - Multithreading
    - Schedulers with direct code integration (X10)
    - Schedulers using binary (JAVA, X10, Chapel, D, Go)
  - Scheduler using Linux system calls and binary

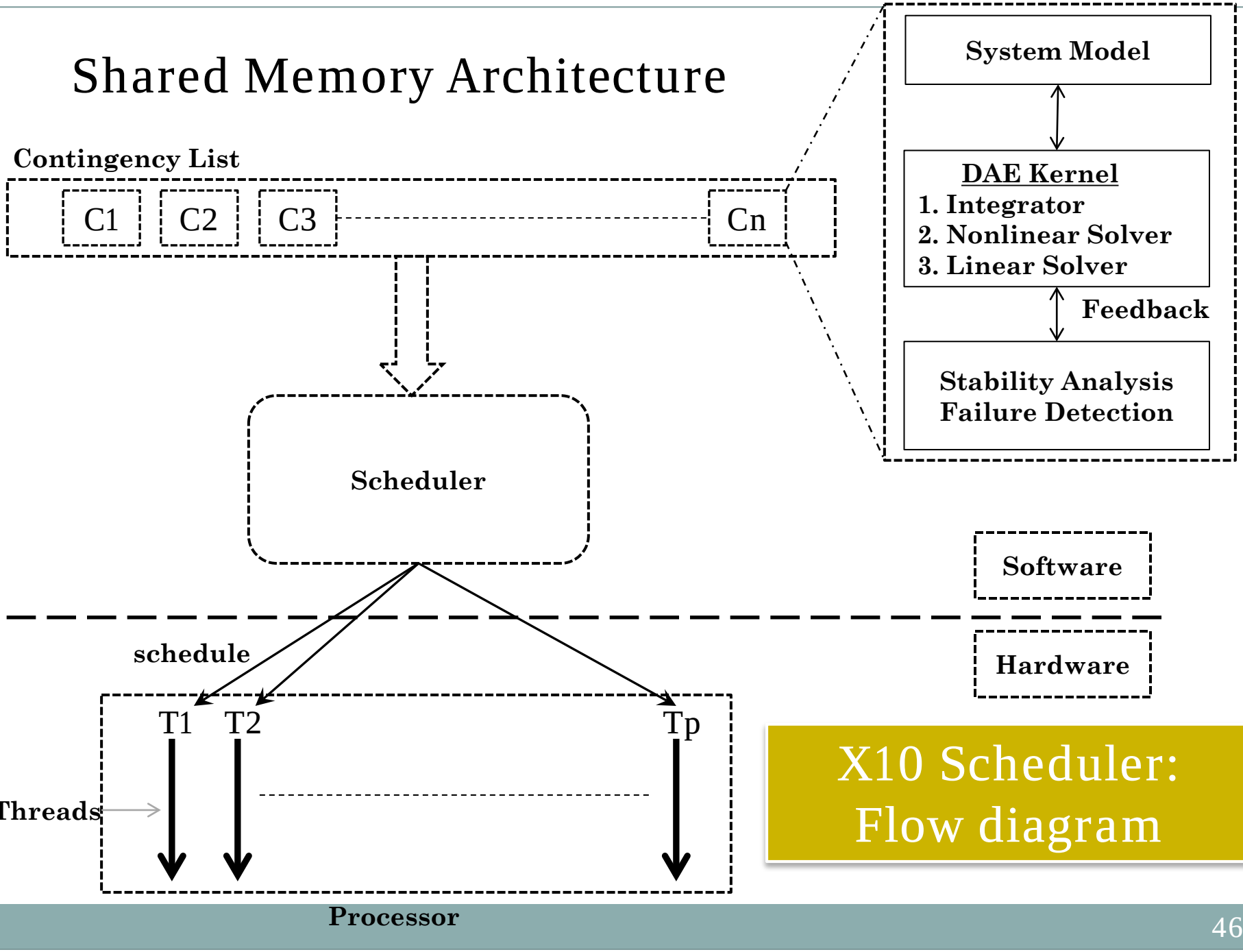
Distributed Memory Architecture

Master-Slave (MPI) and Work-Stealing (MPI+Multithreaded)
Weak scaling w.r.t. 8 processors - Speedup vs. #processors

Speedup is defined as S(C, P) =T(C, 8)/T(C,P)



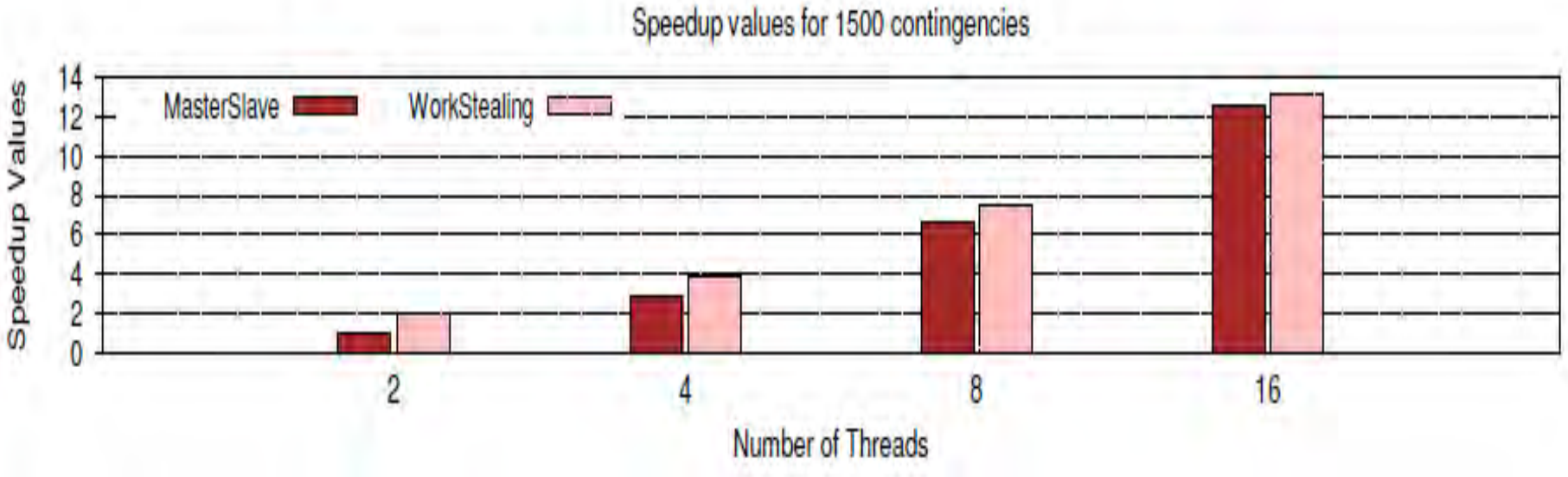
Shared Memory Architecture



X10 Scheduler: Flow diagram

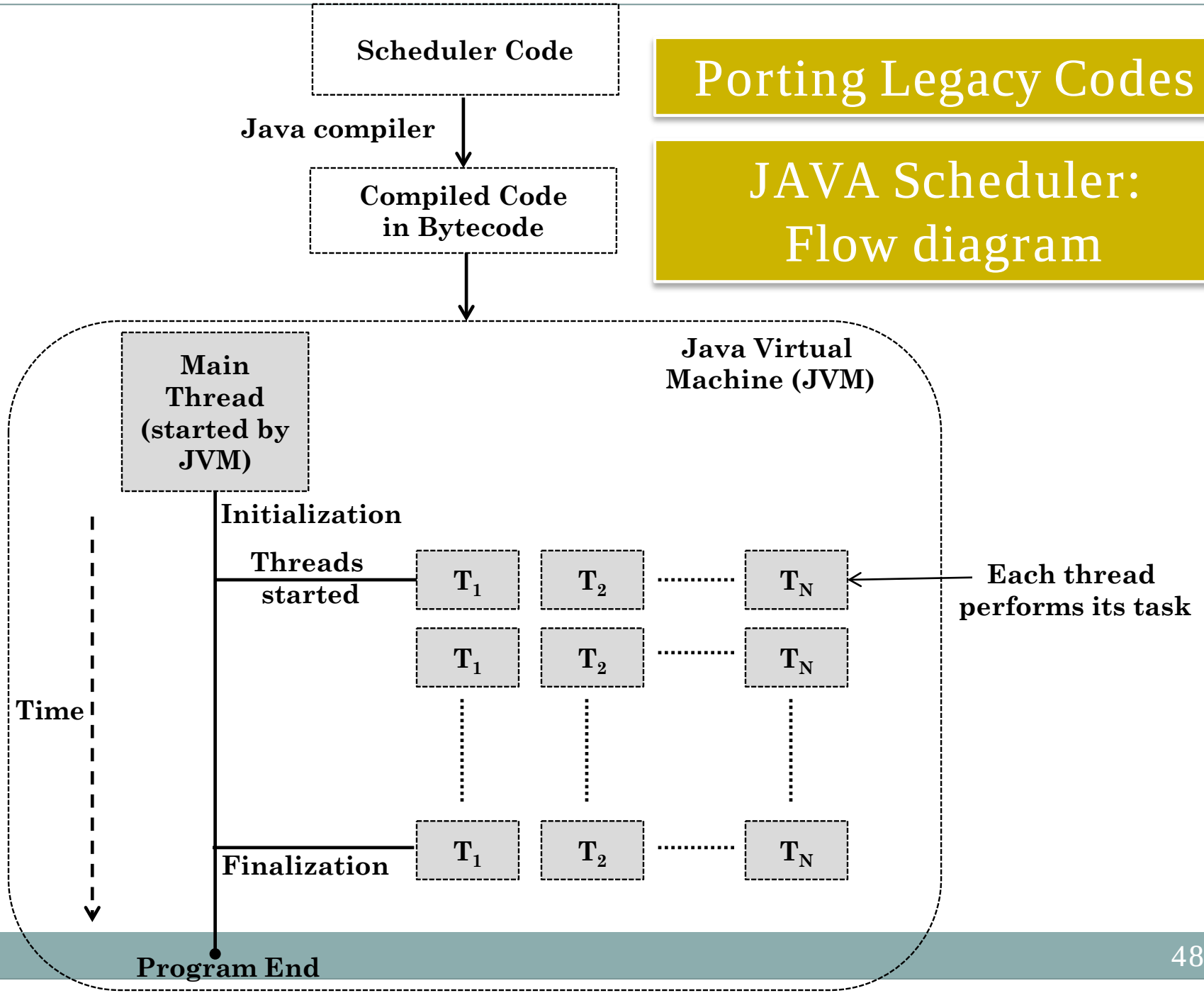
Shared Memory Architecture

X10: Master-Slave and Work-Stealing (Speedup Over Serial Execution)

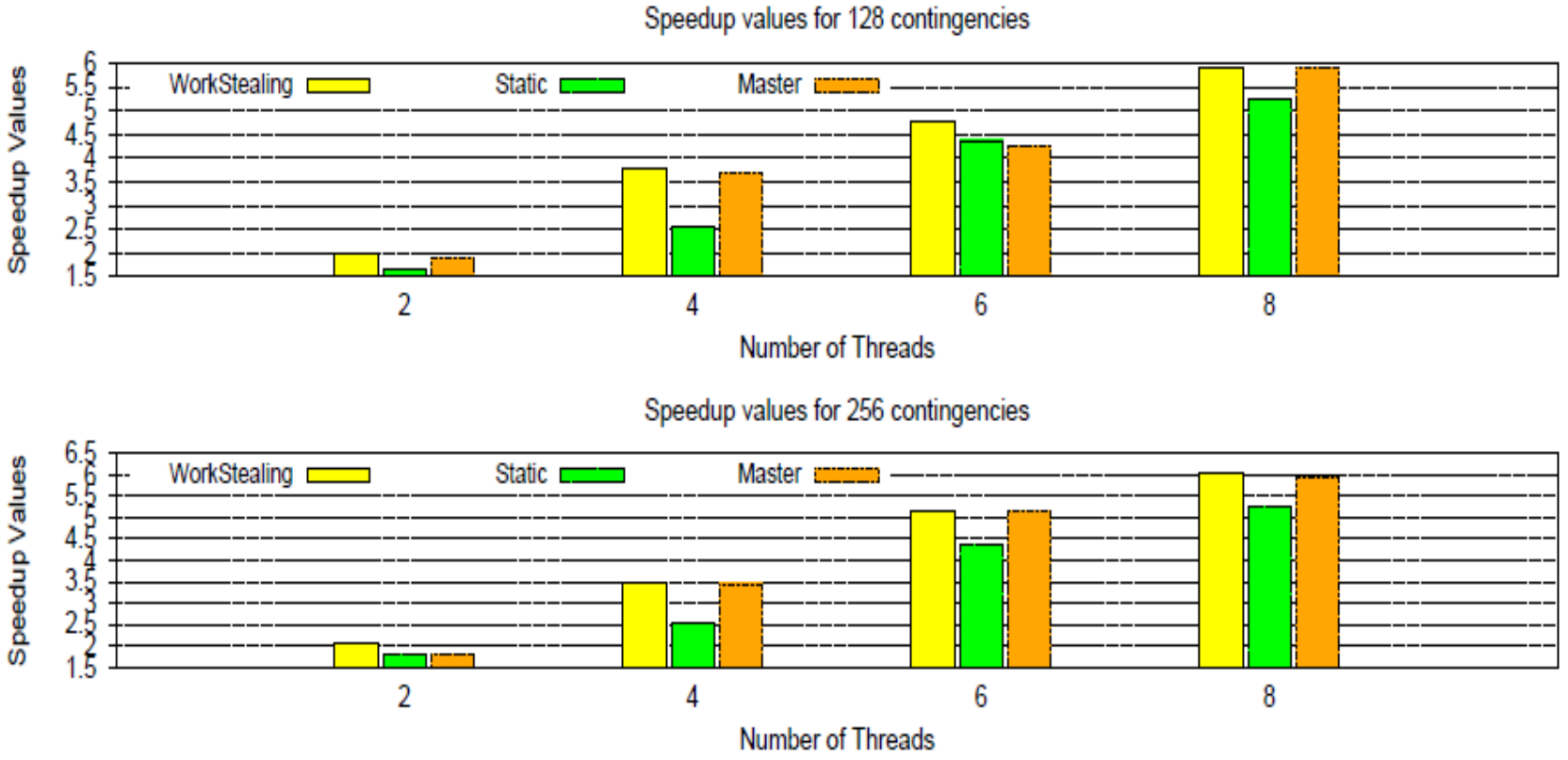


Porting Legacy Codes

JAVA Scheduler: Flow diagram



JAVA: Master-Slave, Static and Work-Stealing (Simulation Time in Seconds)



Summary

- Parallel programming is a promising approach for accelerating computation-intensive application.
- Effective scheduling techniques are important for achieving load-balancing.
- Choice of programming language is important for efficient implementation on different architectures.
- Accelerating legacy code requires special approaches.
- HPC can significantly improve computational speed for security assessment.

Selected Publications

- **Khaitan, S.** and McCalley, J. D., "Dynamic Load Balancing and Scheduling for Parallel Power System Dynamic Contingency Analysis" *High Performance Computing in Power and Energy Systems*, Springer-Verlag Inc., 2012.
- **Khaitan, S.** and McCalley, J. D., "High Performance Computing for Power System Dynamic Simulation" *High Performance Computing in Power and Energy Systems*, Springer-Verlag 2012.
- **Khaitan, S.** and McCalley, J. D., "Parallelization and Load Balancing Techniques for HPC" *Encyclopedia of Business Analytics and Optimization*, 2014 (to appear).
- **Khaitan, S**. and McCalley, J. "EmPower: An Efficient Load Balancing Approach For Massive Dynamic Contingency Analysis in Power Systems" in in *2nd HiPCNA-PG*, SC12, 2012.
- **Khaitan, S**. and McCalley, J. "TDPSS: A Scalable Time Domain Power System Simulator For Dynamic Security Assessment" in *2nd HiPCNA-PG*, SC12, 2012.
- **Khaitan, S**. and McCalley, J. "Parallelizing Power System Contingency Analysis Using D Programming Language" in IEEE PES-GM 2013.

Under Review

- **Khaitan, S.**, McCalley, J. D., and Somani, A. "Proactive Task Scheduling and Stealing in Master-Slave Based Load Balancing for Parallel Contingency Analysis", Submitted.
- **Khaitan, S**. and McCalley, J. D. "IMPACT: A Constraint-Aware Scheduling and Load-Balancing Technique for Parallel Contingency Analysis", Submitted.
- **Khaitan, S.** and McCalley, J. D., "MASTER: A JAVA Based Multithreaded Work-Stealing Technique for Parallel Contingency Analysis in Power Systems", Submitted.

Questions and comments are welcome

Thank you very much!

Siddhartha K. Khaitan
skhaitan@iastate.edu