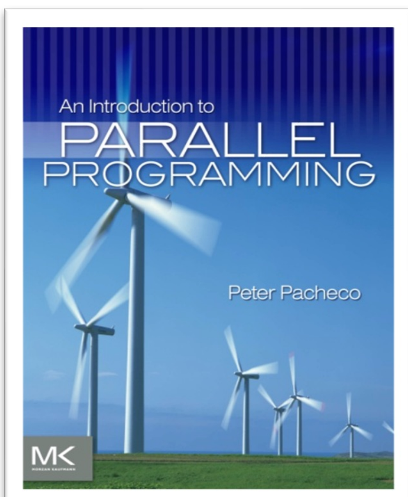
Introduction to Parallel Programming

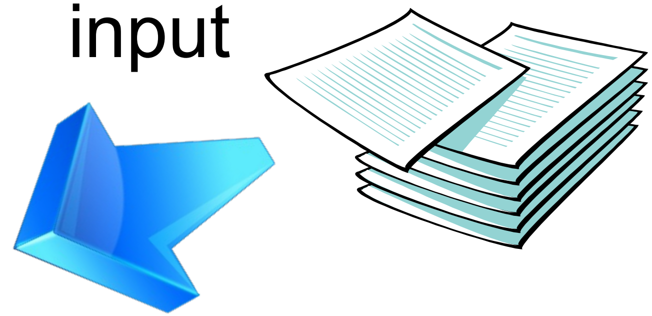
Center for Institutional Research Computing



Slides for the book "An introduction to Parallel Programming", by Peter Pacheco (available from the publisher website): [*http://booksite.elsevier.com/9780123742605/*](http://booksite.elsevier.com/9780123742605/)

Copyright © 2010, Elsevier Inc. All rights Reserved

Serial hardware and software



Computer runs one program at a time.

Copyright © 2010, Elsevier Inc. All rights Reserved

Why we need to write parallel programs

- Running multiple instances of a serial program often is not very useful.
  - Have the same program run 100 times
  - Have 100 computers run the same program 1 time

Copyright © 2010, Elsevier Inc. All rights Reserved

Why we need to write parallel programs

- Running multiple instances of a serial program often is not very useful.
  - Have the same program run 100 times
  - Have 100 computers run the same program 1 time
- What you really want is to make the overall process finish faster.

Copyright © 2010, Elsevier Inc. All rights Reserved

How do we write parallel programs?

- Partition the workload and let CPU cores work in parallel
  - Task parallelism
    - Partition various tasks used in solving the problem among the cores.

Copyright © 2010, Elsevier Inc. All rights Reserved

How do we write parallel programs?

- Partition the workload and let CPU cores work in parallel
  - Task parallelism
    - Partition various tasks used in solving the problem among the cores.
  - Data parallelism
    - Partition the data used in solving the problem among the cores.

Copyright © 2010, Elsevier Inc. All rights Reserved

How do we write parallel programs?

- Partition the workload and let CPU cores work in parallel
  - Task parallelism
    - Partition various tasks used in solving the problem among the cores.
  - Data parallelism
    - Partition the data used in solving the problem among the cores.
    - Each core carries out similar operations on it’s part of the data.

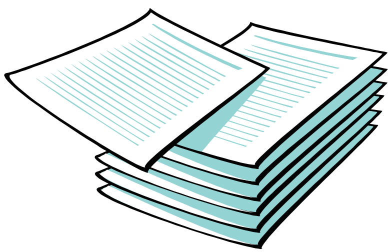
Copyright © 2010, Elsevier Inc. All rights Reserved

Professor P

Grade an exam:

300 exam papers

15 questions each

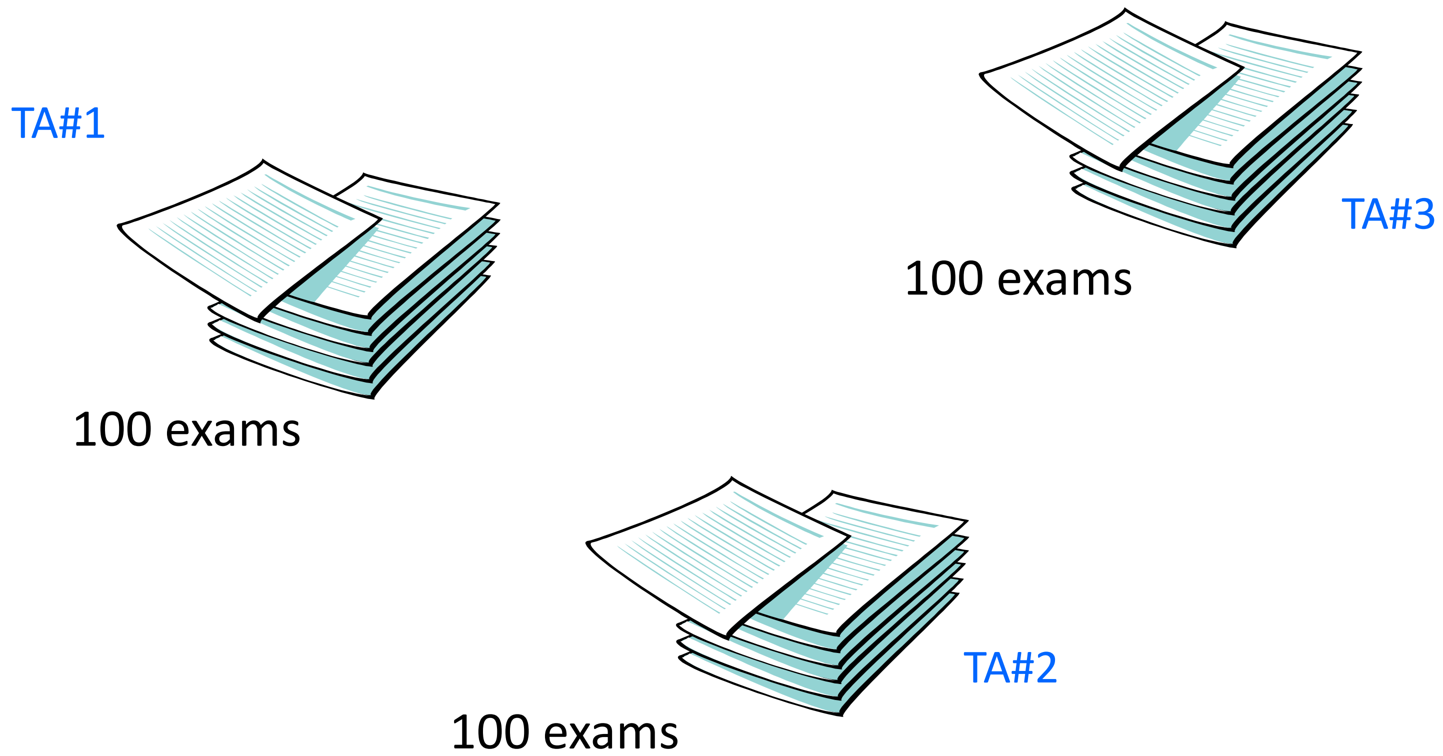


Copyright © 2010, Elsevier
Inc. All rights Reserved

Professor P's grading assistants

Copyright © 2010, Elsevier
Inc. All rights Reserved

Division of work – data parallelism



Copyright © 2010, Elsevier
Inc. All rights Reserved

Division of work – task parallelism

TA#1

Questions 1 - 5

or Questions 1 - 7

TA#3

Questions 11 - 15

or Questions 12 - 15

TA#2

Questions 6 - 10

or Questions 8 - 11

Partitioning strategy:
- either by number
- Or by workload

Copyright © 2010, Elsevier Inc. All rights Reserved

Coordination

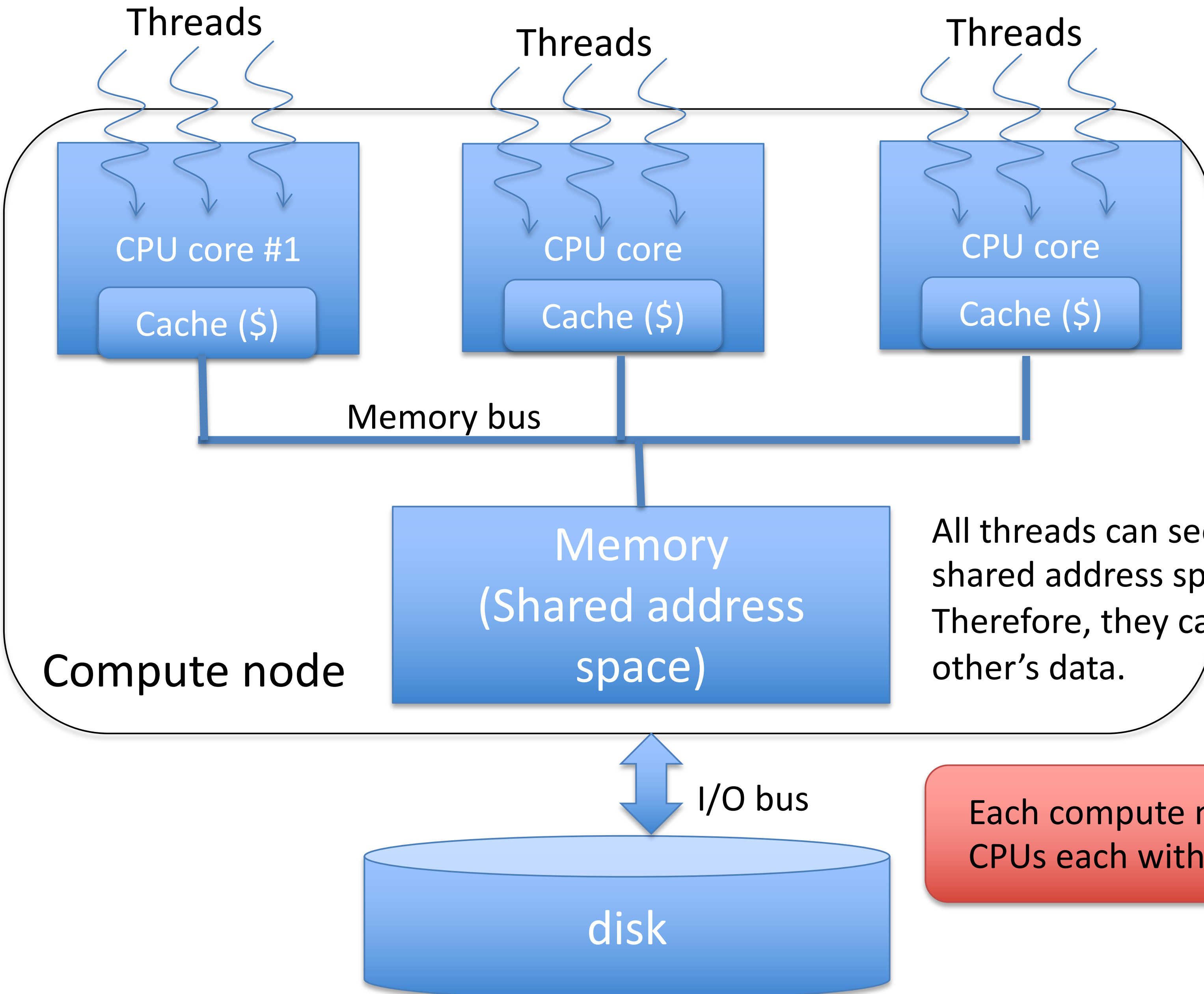
- Cores usually need to coordinate their work.
- Communication – one or more cores send their current partial sums to another core.
  - E.g., ML algorithms, PageRank
- Load balancing – share the work evenly among the cores so that one is not heavily loaded.
- Synchronization – because each core works at its own pace, make sure cores do not get too far ahead of the rest.

Copyright © 2010, Elsevier Inc. All rights Reserved

Memory

- Two major classes of parallel programming models:
  - Shared Memory
  - Distributed Memory

Shared Memory Architecture



All threads can see a single shared address space. Therefore, they can see each other's data.

Each compute node has 1-2 CPUs each with 10-14 cores

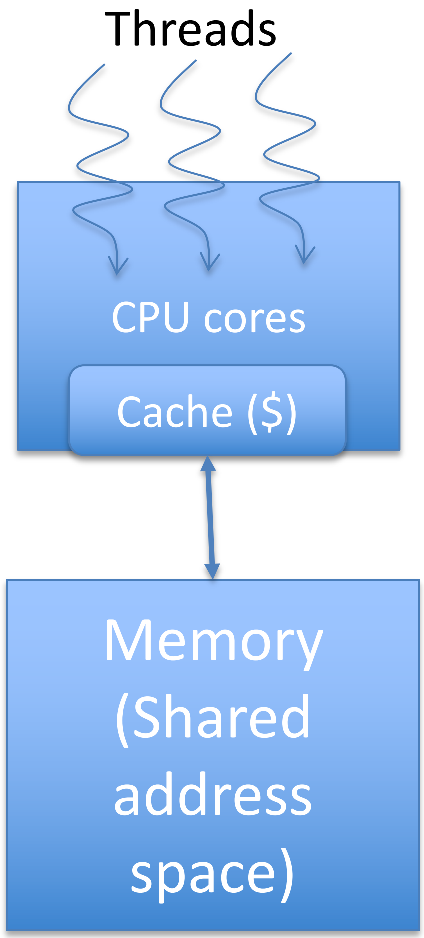
Multi-Threading (for shared memory architectures)

- Threads are contained within processes
  - One process => multiple threads

- All threads of a process share the same address space (in memory).

- Threads have the capability to run concurrently (executing different instructions and accessing different pieces of data at the same time)

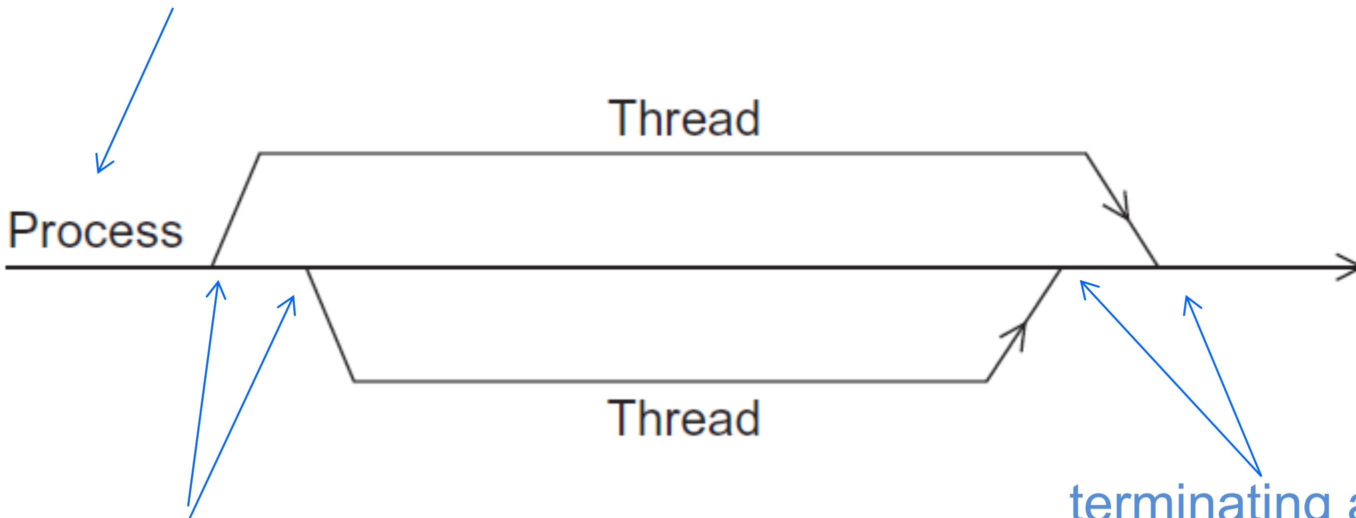
- But if the resource is occupied by another thread, they form a queue and wait.
  - For maximum throughput, it is ideal to map each thread to a unique/distinct core



Copyright © 2010, Elsevier Inc. All rights Reserved

A process and two threads

the "master" thread

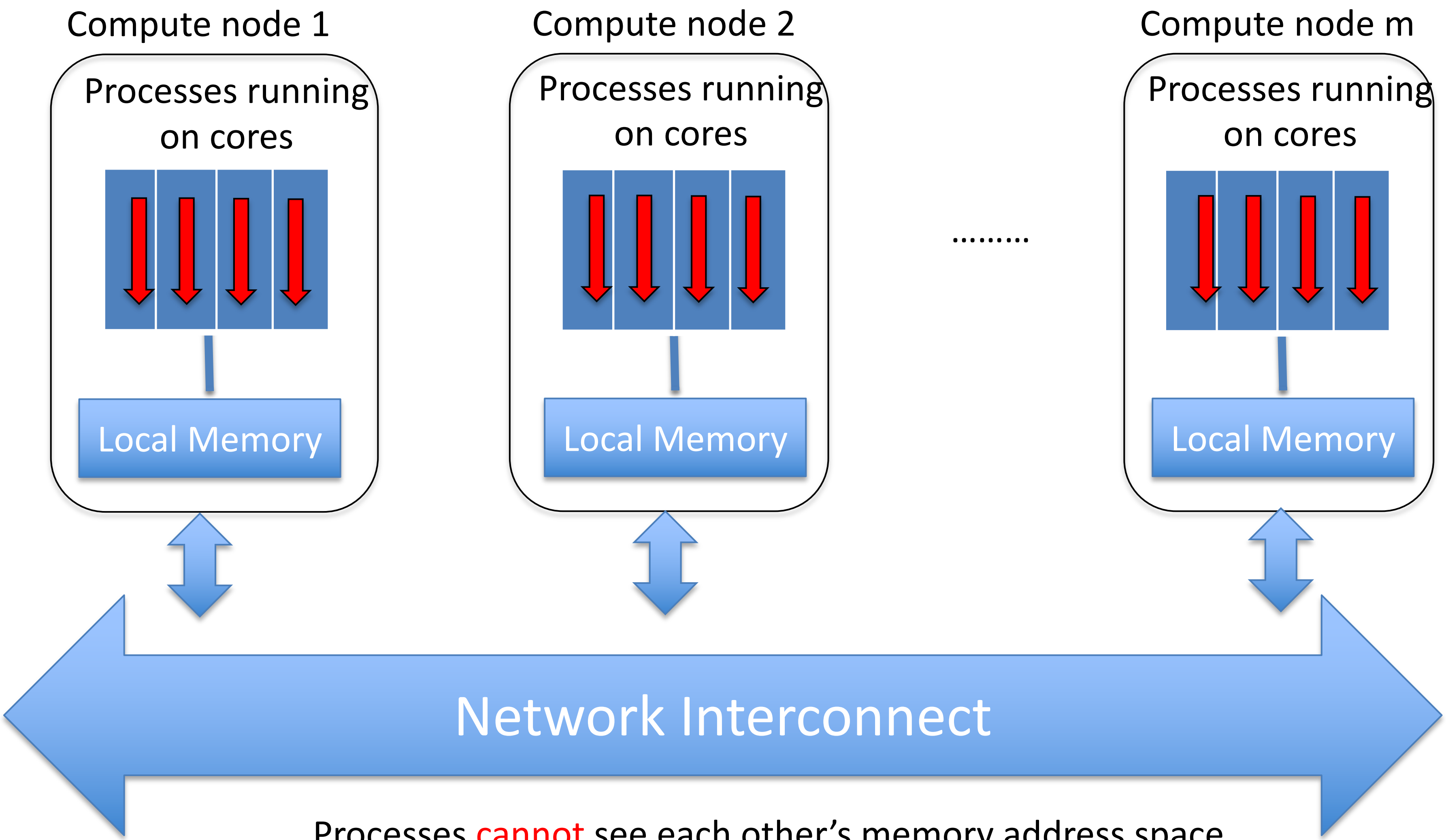


starting a thread
Is called ***forking***

terminating a thread
Is called ***joining***

Copyright © 2010, Elsevier
Inc. All rights Reserved

Distributed Memory Architecture



Processes cannot see each other's memory address space.
They have to send inter-process messages (using MPI).

Distributed Memory System

- Clusters (most popular)
  - A collection of commodity systems.
  - Connected by a commodity interconnection network.
- Nodes of a cluster are individual computers joined by a communication network.

How to change your program to a parallel program? Foster's methodology

1. Partitioning: divide the computation to be performed and the data operated on by the computation into small tasks.

   The focus here should be on identifying tasks that can be executed in parallel.

Copyright © 2010, Elsevier Inc. All rights Reserved

Foster’s methodology

2. Communication: determine what communication needs to be carried out among the tasks identified in the previous step.

Copyright © 2010, Elsevier Inc. All rights Reserved

Foster’s methodology

3. Aggregation: combine tasks and communications identified in the first step into larger tasks.

   For example, if task A must be executed before task B can be executed, it may make sense to aggregate them into a single composite task.

Copyright © 2010, Elsevier Inc. All rights Reserved

Foster’ s methodology

4. Mapping: assign the composite tasks identified in the previous step to processes/threads.

   This should be done so that communication is minimized, and each process/thread gets roughly the same amount of work.

Copyright © 2010, Elsevier Inc. All rights Reserved