Parallel Programming Models

- Introduction to Programming Models
  - Parallel Execution Models
    - Models for Communication
    - Models for Synchronization
    - Memory Consistency Models
    - Runtime systems
  - Productivity
  - Performance
  - Portability

PPM – Implicit Synchronization

Implicit Synchronization

- Hidden in communication operations (e.g., two-sided communication)
- Data Dependence Graph (DDG)
  - PPL synchronizes where necessary to enforce the dependences
  - E.g., STAPL
- Distributed Termination Detection
  - When implemented as background algorithm (e.g., in Charm++)
- Improved productivity
  - Less bugs from race conditions, deadlocks …
- E.g., STAPL, Charm++, MPI-1 and GASNet (to a certain extent)

PPM – Explicit Synchronization

Explicit Synchronization

- Critical section / locks
  - One thread allowed to execute the guarded code at a time
- Condition variables / blocking semaphores
  - Producer/consumer synchronization
  - Introduces order in the execution
- Monitors / counting semaphores
  - Shared resources management
- Barrier / Fence (global synchronization)
  - Threads of execution wait until all reach the same point
- E.g., Pthreads, TBB, OpenMP

148

Parallel Programming Models

- Introduction to Programming Models
  - Parallel Execution Models
    - Models for Communication
    - Models for Synchronization
    - Memory Consistency Models
    - Runtime systems
  - Productivity
  - Performance
  - Portability

149

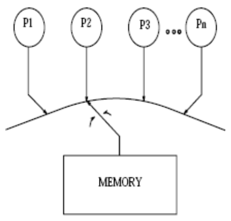
PPM – Memory Consistency

Introduction to Memory Consistency

– Specification of the effect of Read and Write operations on the memory

– Usual user assumption: Sequential Consistency

Definition: [A multiprocessor system is sequentially consistent if] the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

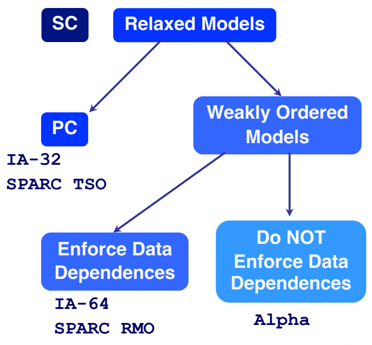


PPM – Memory Consistency

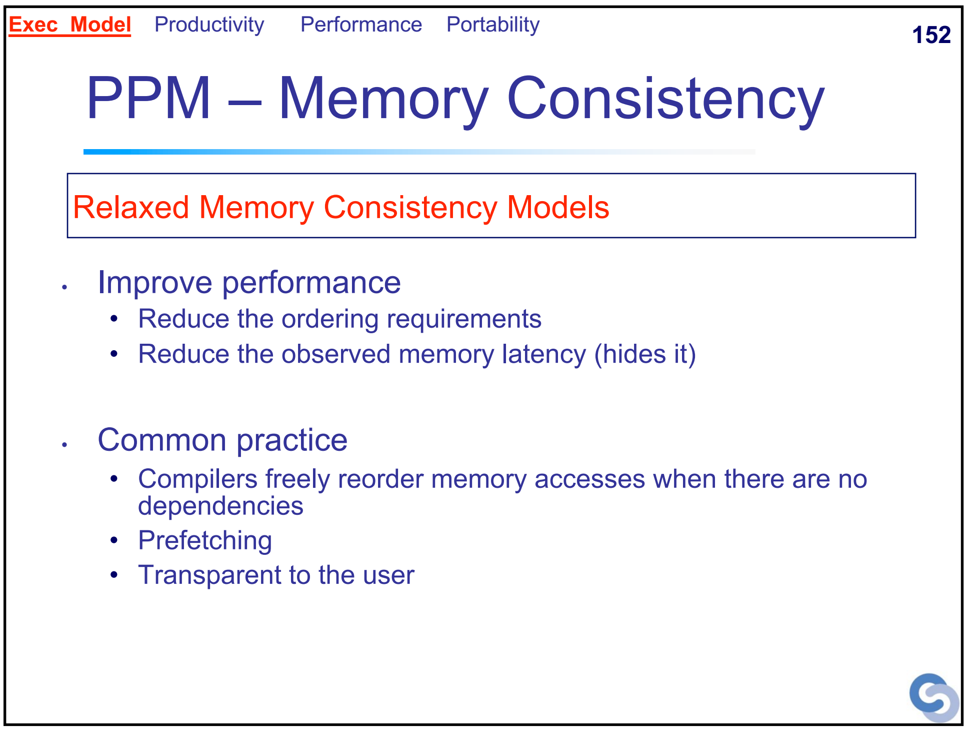
Introduction to Memory Consistency

Sequential Consistency: Don't assume it !

- Sequential Consistency (SC)
  - •**MIPS/SGI**
  - •**HP PA-RISC**
- Processor Consistency (PC)
  - •*Relax write→read dependencies*
  - •**Intel x86 (IA-32)**
  - •**Sun TSO (Total Store Order)**
- Relaxed Consistency (RC)
  - •*Relax all dependencies, but add fences*
  - •**DEC Alpha**
  - •**IBM PowerPC**
  - •**Intel IPF (IA-64)**
  - •**Sun RMO (Relaxed Memory Order)**



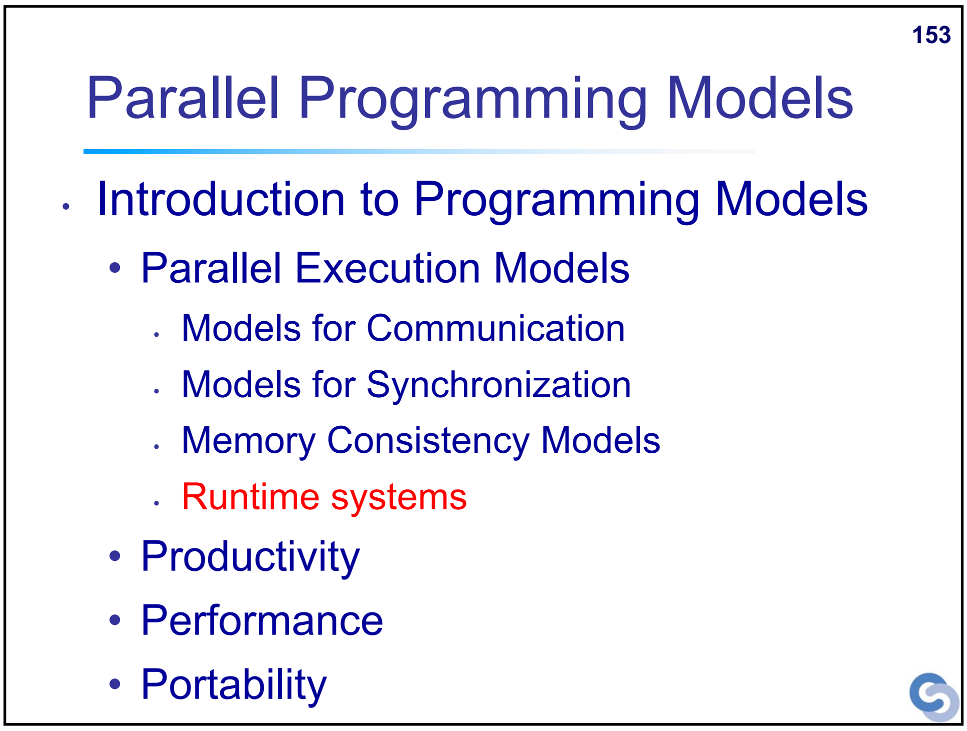
Material from: Hill, M. D. Revisiting "Multiprocessors Should Support Simple Memory Consistency Models", http://www.cs.wisc.edu/multifacet/papers/dagstuhl03_memory_consistency.ppt



PPM – Memory Consistency

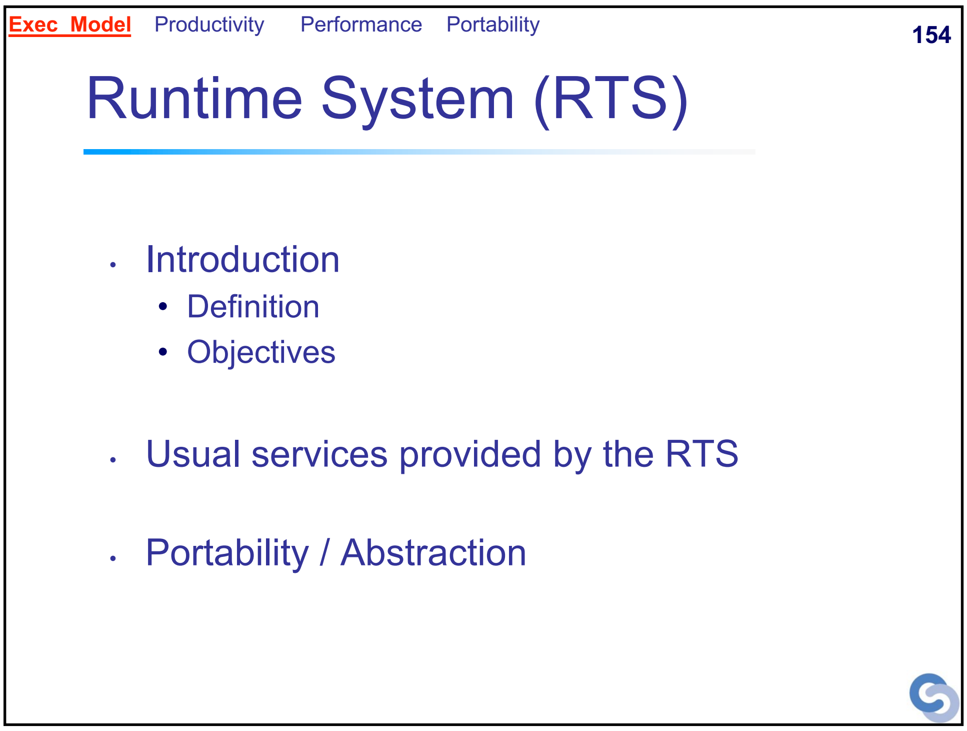
Relaxed Memory Consistency Models

- Improve performance
  - Reduce the ordering requirements
  - Reduce the observed memory latency (hides it)
- Common practice
  - Compilers freely reorder memory accesses when there are no dependencies
  - Prefetching
  - Transparent to the user



Parallel Programming Models

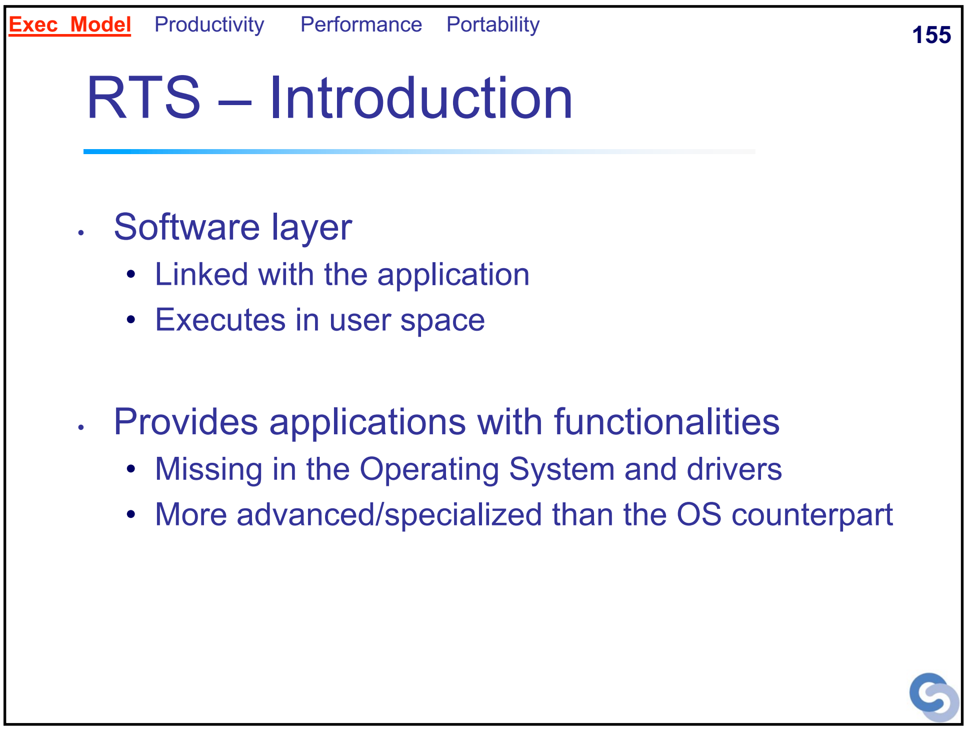
- Introduction to Programming Models
  - Parallel Execution Models
    - Models for Communication
    - Models for Synchronization
    - Memory Consistency Models
    - Runtime systems
  - Productivity
  - Performance
  - Portability



Runtime System (RTS)

- Introduction
  - Definition
  - Objectives
- Usual services provided by the RTS
- Portability / Abstraction

154

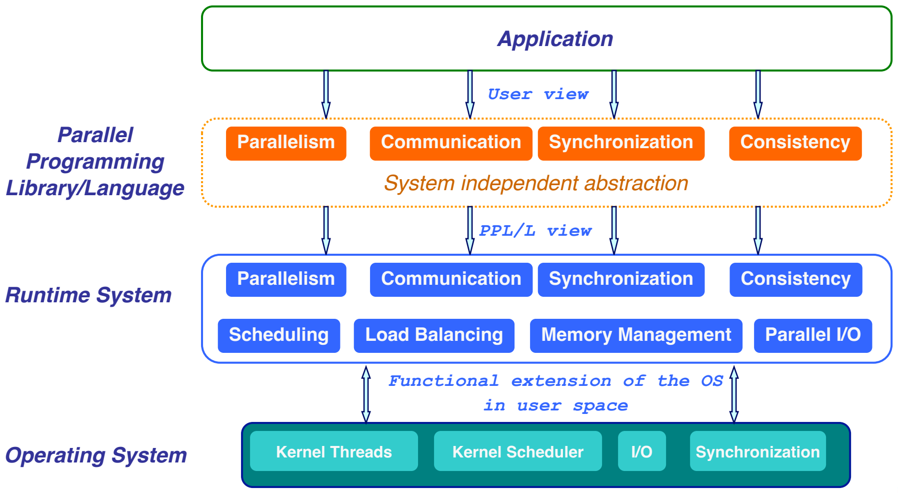


RTS – Introduction

- Software layer
  - Linked with the application
  - Executes in user space
- Provides applications with functionalities
  - Missing in the Operating System and drivers
  - More advanced/specialized than the OS counterpart

155

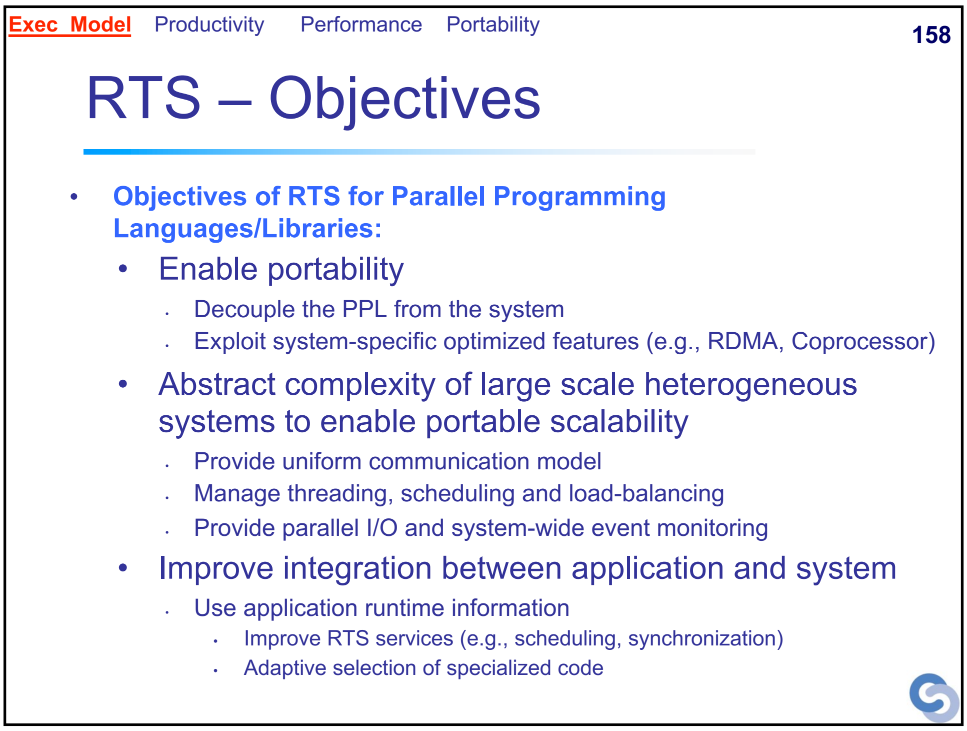
Parallel Execution Model



RTS – Definition*

- **Functional extension of the Operating System in user space**
  - No precise definition available
  - Fuzzy functional boundary between RTS and OS
    - Services are often a refined or extended version of the OS
    - Functional redundancy with OS services
      - Avoid entering Kernel space
      - Provide reentrancy
      - E.g., threading, synchronization, scheduling …
  - Widely variable set of provided services
    - No minimum requirements
    - No limit on the amount of functionality

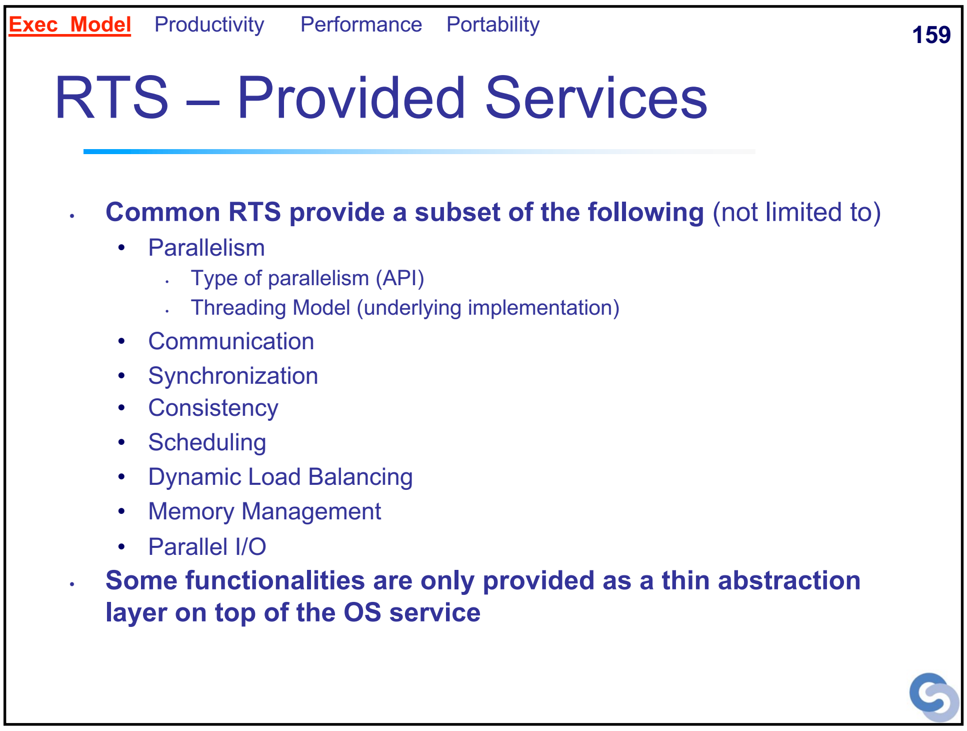
*Non-formal, short definition



RTS – Objectives

- **Objectives of RTS for Parallel Programming Languages/Libraries:**
  - Enable portability
    - Decouple the PPL from the system
    - Exploit system-specific optimized features (e.g., RDMA, Coprocessor)
  - Abstract complexity of large scale heterogeneous systems to enable portable scalability
    - Provide uniform communication model
    - Manage threading, scheduling and load-balancing
    - Provide parallel I/O and system-wide event monitoring
  - Improve integration between application and system
    - Use application runtime information
      - Improve RTS services (e.g., scheduling, synchronization)
      - Adaptive selection of specialized code

158

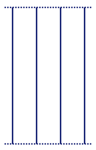


RTS – Provided Services

- **Common RTS provide a subset of the following** (not limited to)
  - Parallelism
    - Type of parallelism (API)
    - Threading Model (underlying implementation)
  - Communication
  - Synchronization
  - Consistency
  - Scheduling
  - Dynamic Load Balancing
  - Memory Management
  - Parallel I/O
- **Some functionalities are only provided as a thin abstraction layer on top of the OS service**

159

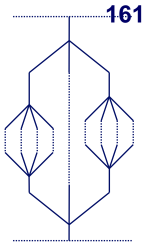
RTS – Flat Parallelism



Parallelism types – Flat Parallelism

- All threads of execution have the same status
  - No parent/child relationship
- Threads are active during the whole execution
- Usually constant number of threads of execution
- Well adapted for problems with large granularity
- Difficult to achieve load-balance for non-embarrassingly parallel applications
- E.g. **MPI**

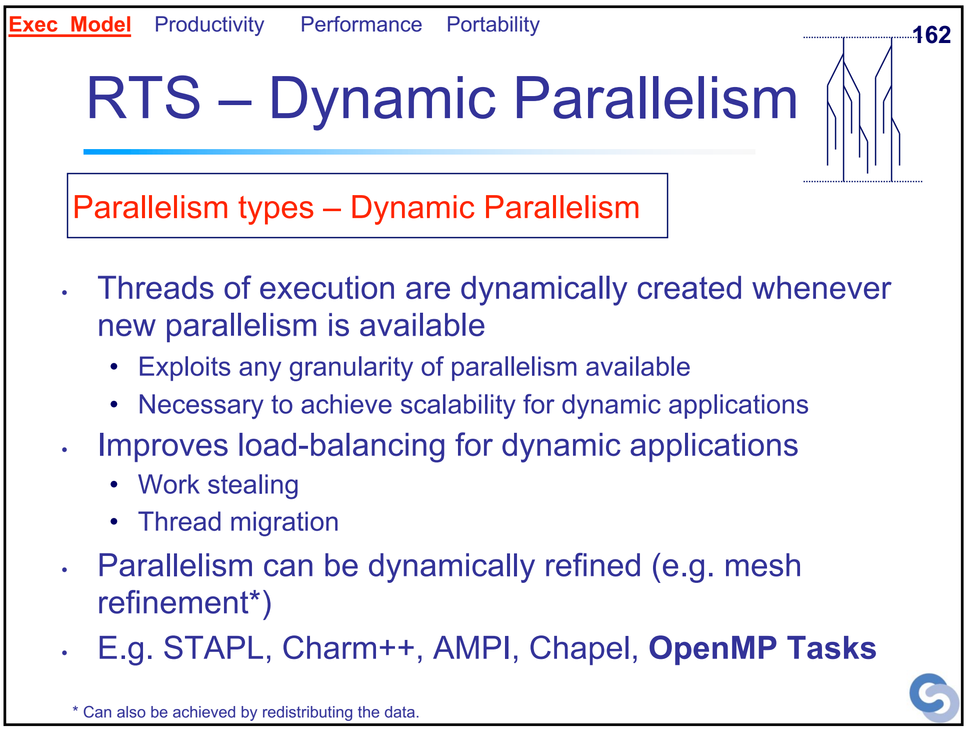
RTS – Nested Parallelism



Parallelism types – Nested Parallelism

- Parallelism is hierarchal
  - Threads of execution can spawn new threads to execute their task
  - Exploits multiple levels of parallelism (e.g. nested parallel loops)
- Good affinity with heterogeneous architectures (e.g. clusters of SMPs)*
  - Allows the exploitation of different levels of granularity
- Natural fit for composed parallel data structures*
  - E.g. `p_vector< p_list< Type > >`
- E.g. **OpenMP**, Cilk, TBB

* Also for dynamic parallelism.



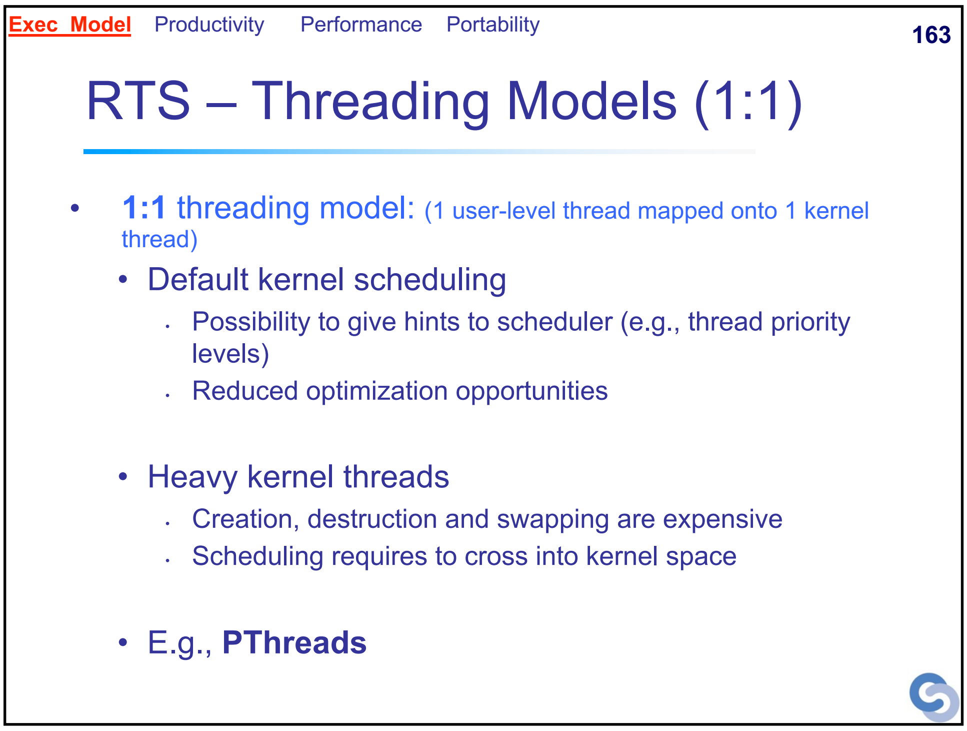
RTS – Dynamic Parallelism

Parallelism types – Dynamic Parallelism

- Threads of execution are dynamically created whenever new parallelism is available
  - Exploits any granularity of parallelism available
  - Necessary to achieve scalability for dynamic applications
- Improves load-balancing for dynamic applications
  - Work stealing
  - Thread migration
- Parallelism can be dynamically refined (e.g. mesh refinement*)
- E.g. STAPL, Charm++, AMPI, Chapel, **OpenMP Tasks**

* Can also be achieved by redistributing the data.

162



RTS – Threading Models (1:1)

- **1:1** threading model: (1 user-level thread mapped onto 1 kernel thread)
  - Default kernel scheduling
    - Possibility to give hints to scheduler (e.g., thread priority levels)
    - Reduced optimization opportunities
  - Heavy kernel threads
    - Creation, destruction and swapping are expensive
    - Scheduling requires to cross into kernel space
  - E.g., **PThreads**

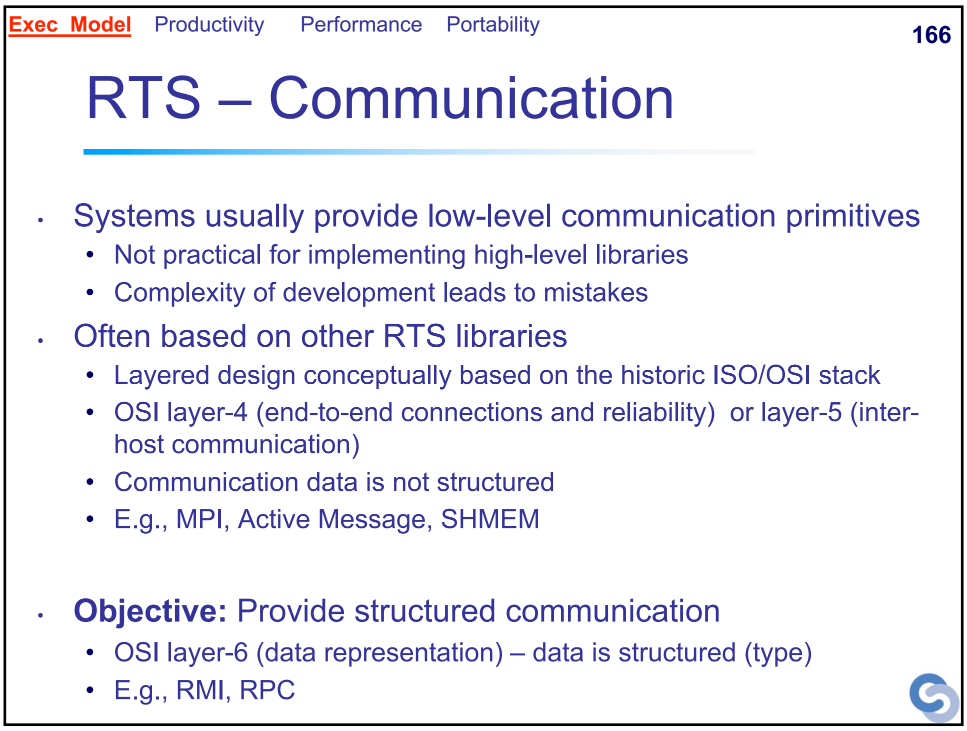
163

RTS – Threading Models (M:1)

- **M:1** threading model: (M user-level threads mapped onto 1 kernel thread)
  - Customizable scheduling
    - Enables scheduler-based optimizations (e.g. priority scheduling, good affinity with latency hiding schemes)
  - Light user-level threads
    - Lesser threading cost
      - User-level thread scheduling requires no kernel trap
  - Problem: no effective parallelism
    - User-level threads' execution serialized on 1 kernel thread
    - Often poor integration with the OS (little or no communication)
    - E.g., GNU Portable Threads

RTS – Threading Models (M:N)

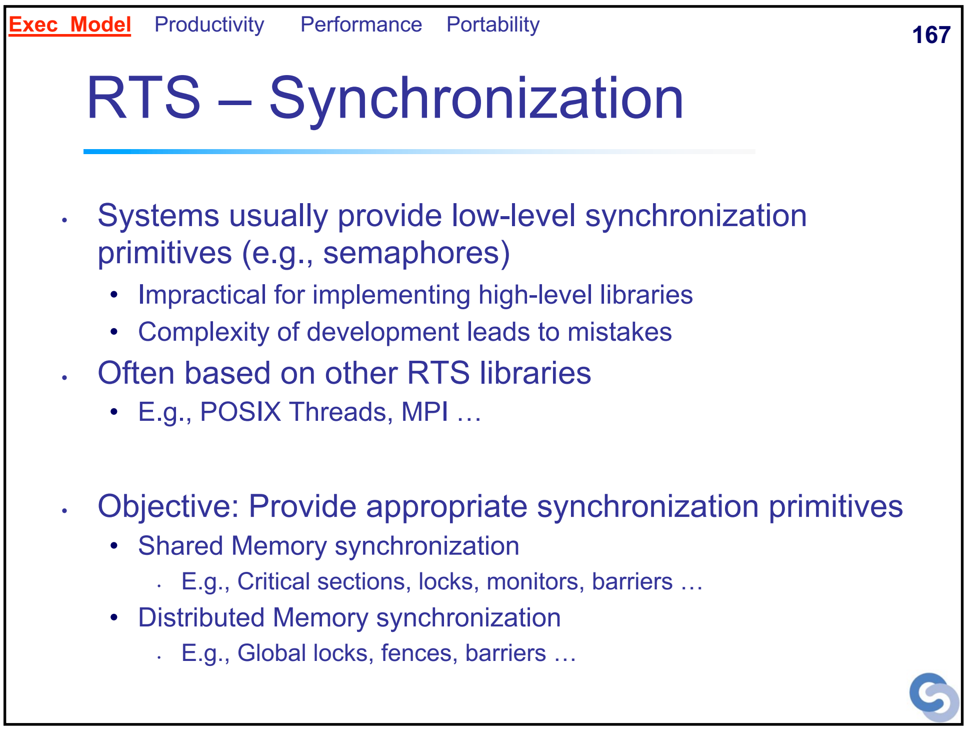
- **M:N** threading model: (M user-level threads mapped onto N kernel threads)
  - Customizable scheduling
    - Enables scheduler-based optimizations (e.g. priority scheduling, better support for relaxing the consistency model …)
  - Light user-level threads
    - Lesser threading cost
      - Can match N with the number of available hardware threads : no kernel-thread swapping, no preemption, no kernel over-scheduling …
      - User-level thread scheduling requires no kernel trap
    - Perfect and free load balancing within the node
      - User-level threads are cooperatively scheduled on the available kernel threads (they migrate freely).
  - E.g., PM2/Marcel



RTS – Communication

- Systems usually provide low-level communication primitives
  - Not practical for implementing high-level libraries
  - Complexity of development leads to mistakes
- Often based on other RTS libraries
  - Layered design conceptually based on the historic ISO/OSI stack
  - OSI layer-4 (end-to-end connections and reliability) or layer-5 (inter-host communication)
  - Communication data is not structured
  - E.g., MPI, Active Message, SHMEM
- **Objective:** Provide structured communication
  - OSI layer-6 (data representation) – data is structured (type)
  - E.g., RMI, RPC

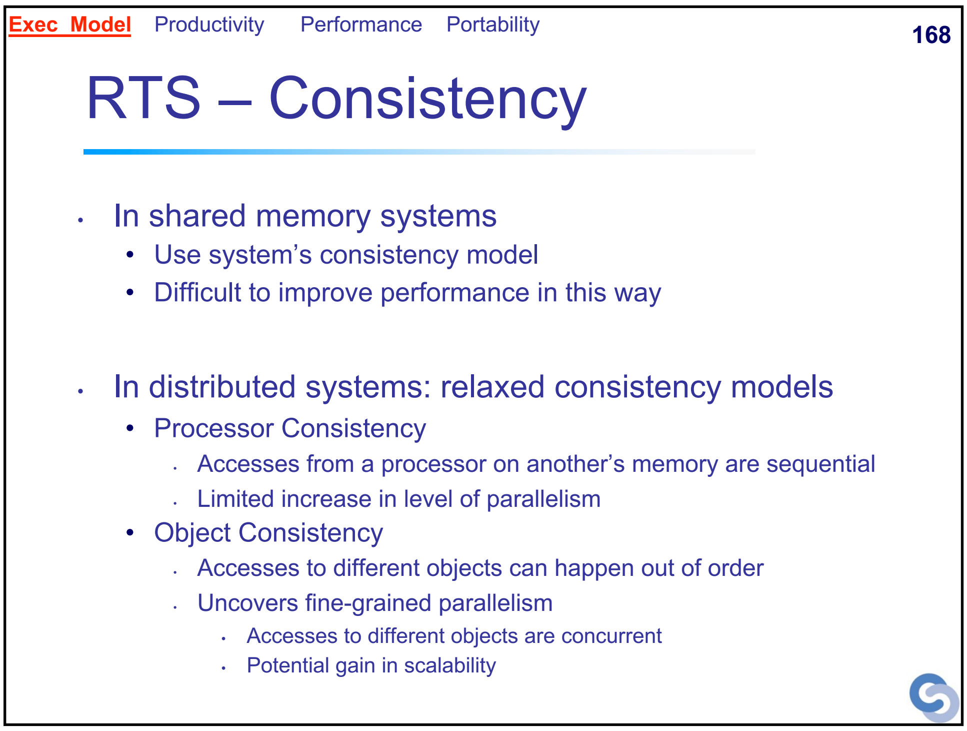
166



RTS – Synchronization

- Systems usually provide low-level synchronization primitives (e.g., semaphores)
  - Impractical for implementing high-level libraries
  - Complexity of development leads to mistakes
- Often based on other RTS libraries
  - E.g., POSIX Threads, MPI …
- Objective: Provide appropriate synchronization primitives
  - Shared Memory synchronization
    - E.g., Critical sections, locks, monitors, barriers …
  - Distributed Memory synchronization
    - E.g., Global locks, fences, barriers …

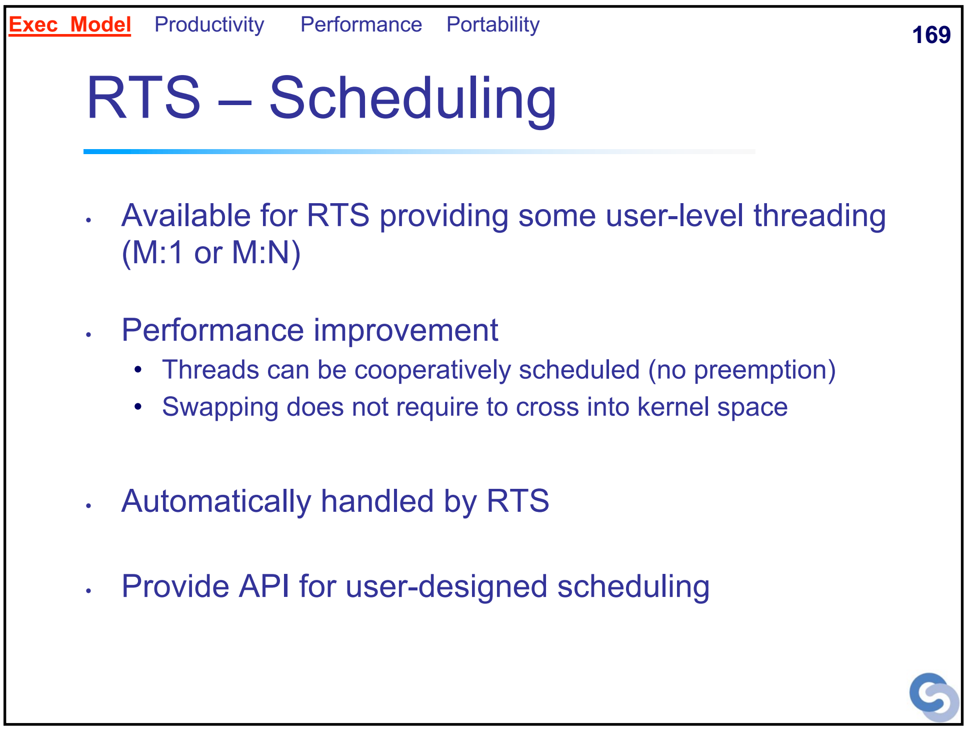
167



RTS – Consistency

- In shared memory systems
  - Use system's consistency model
  - Difficult to improve performance in this way
- In distributed systems: relaxed consistency models
  - Processor Consistency
    - Accesses from a processor on another's memory are sequential
    - Limited increase in level of parallelism
  - Object Consistency
    - Accesses to different objects can happen out of order
    - Uncovers fine-grained parallelism
      - Accesses to different objects are concurrent
      - Potential gain in scalability

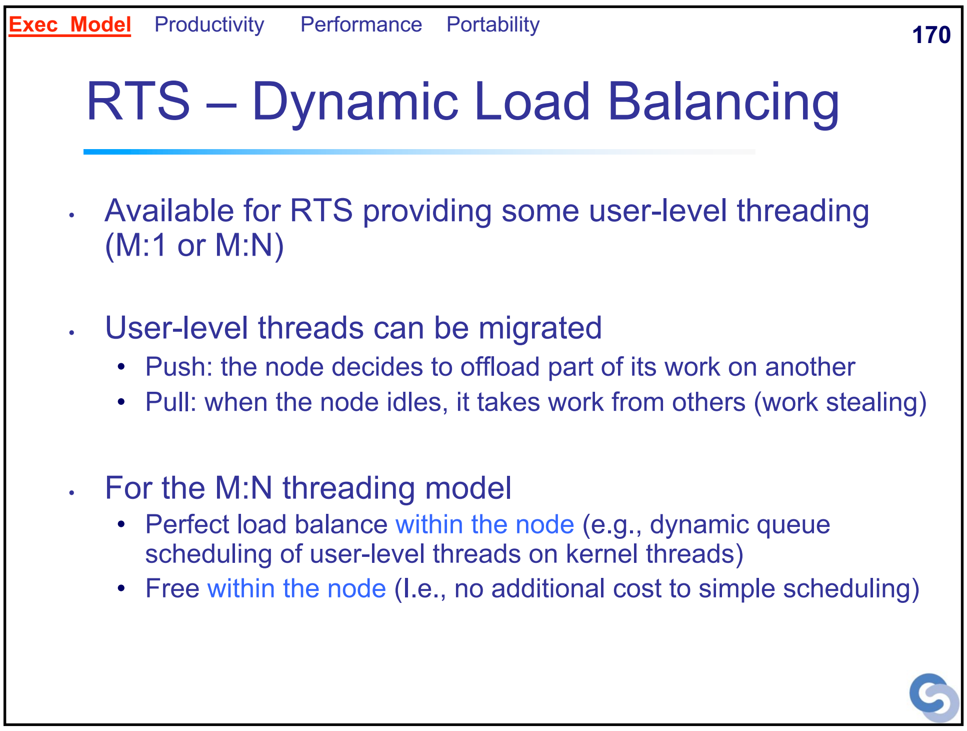
168



RTS – Scheduling

- Available for RTS providing some user-level threading (M:1 or M:N)
- Performance improvement
  - Threads can be cooperatively scheduled (no preemption)
  - Swapping does not require to cross into kernel space
- Automatically handled by RTS
- Provide API for user-designed scheduling

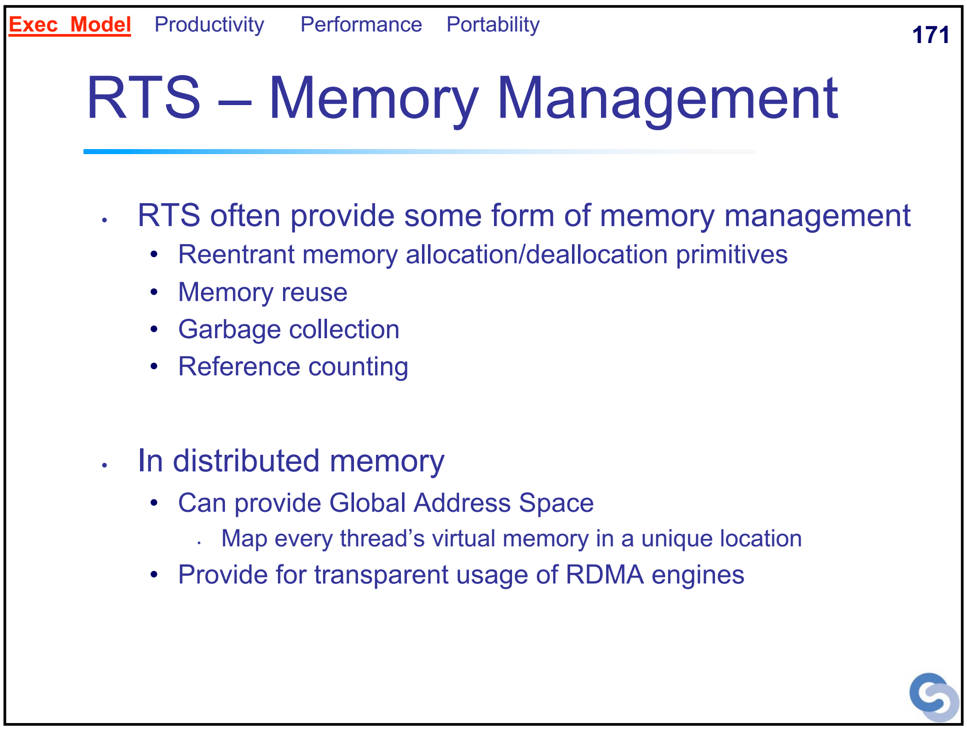
169



RTS – Dynamic Load Balancing

- Available for RTS providing some user-level threading (M:1 or M:N)
- User-level threads can be migrated
  - Push: the node decides to offload part of its work on another
  - Pull: when the node idles, it takes work from others (work stealing)
- For the M:N threading model
  - Perfect load balance within the node (e.g., dynamic queue scheduling of user-level threads on kernel threads)
  - Free within the node (I.e., no additional cost to simple scheduling)

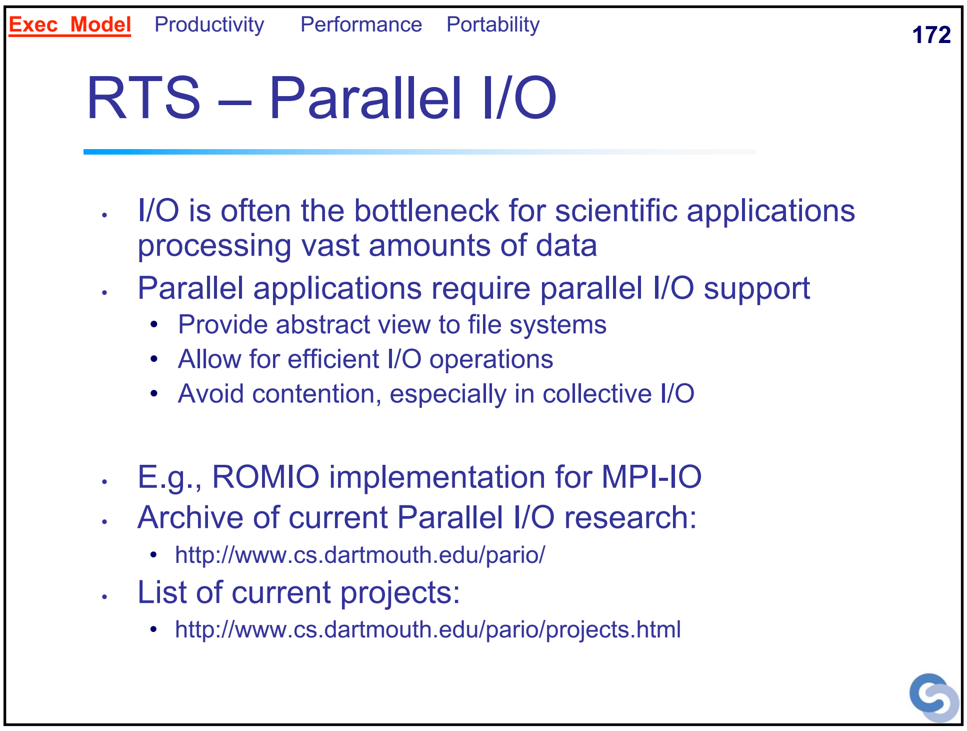
170



RTS – Memory Management

- RTS often provide some form of memory management
  - Reentrant memory allocation/deallocation primitives
  - Memory reuse
  - Garbage collection
  - Reference counting
- In distributed memory
  - Can provide Global Address Space
    - Map every thread's virtual memory in a unique location
  - Provide for transparent usage of RDMA engines

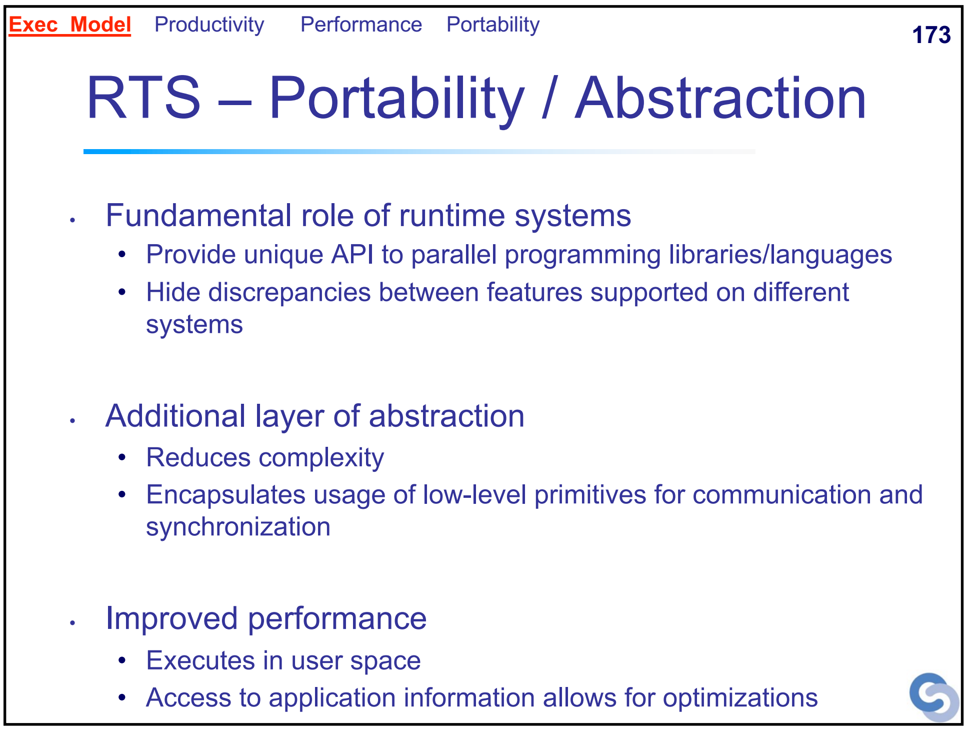
171



RTS – Parallel I/O

- I/O is often the bottleneck for scientific applications processing vast amounts of data
- Parallel applications require parallel I/O support
  - Provide abstract view to file systems
  - Allow for efficient I/O operations
  - Avoid contention, especially in collective I/O
- E.g., ROMIO implementation for MPI-IO
- Archive of current Parallel I/O research:
  - http://www.cs.dartmouth.edu/pario/
- List of current projects:
  - http://www.cs.dartmouth.edu/pario/projects.html

172



RTS – Portability / Abstraction

- Fundamental role of runtime systems
  - Provide unique API to parallel programming libraries/languages
  - Hide discrepancies between features supported on different systems
- Additional layer of abstraction
  - Reduces complexity
  - Encapsulates usage of low-level primitives for communication and synchronization
- Improved performance
  - Executes in user space
  - Access to application information allows for optimizations

173