CPEG 852 — Advanced Topics in Computing Systems
Program Execution Models (PXMs)

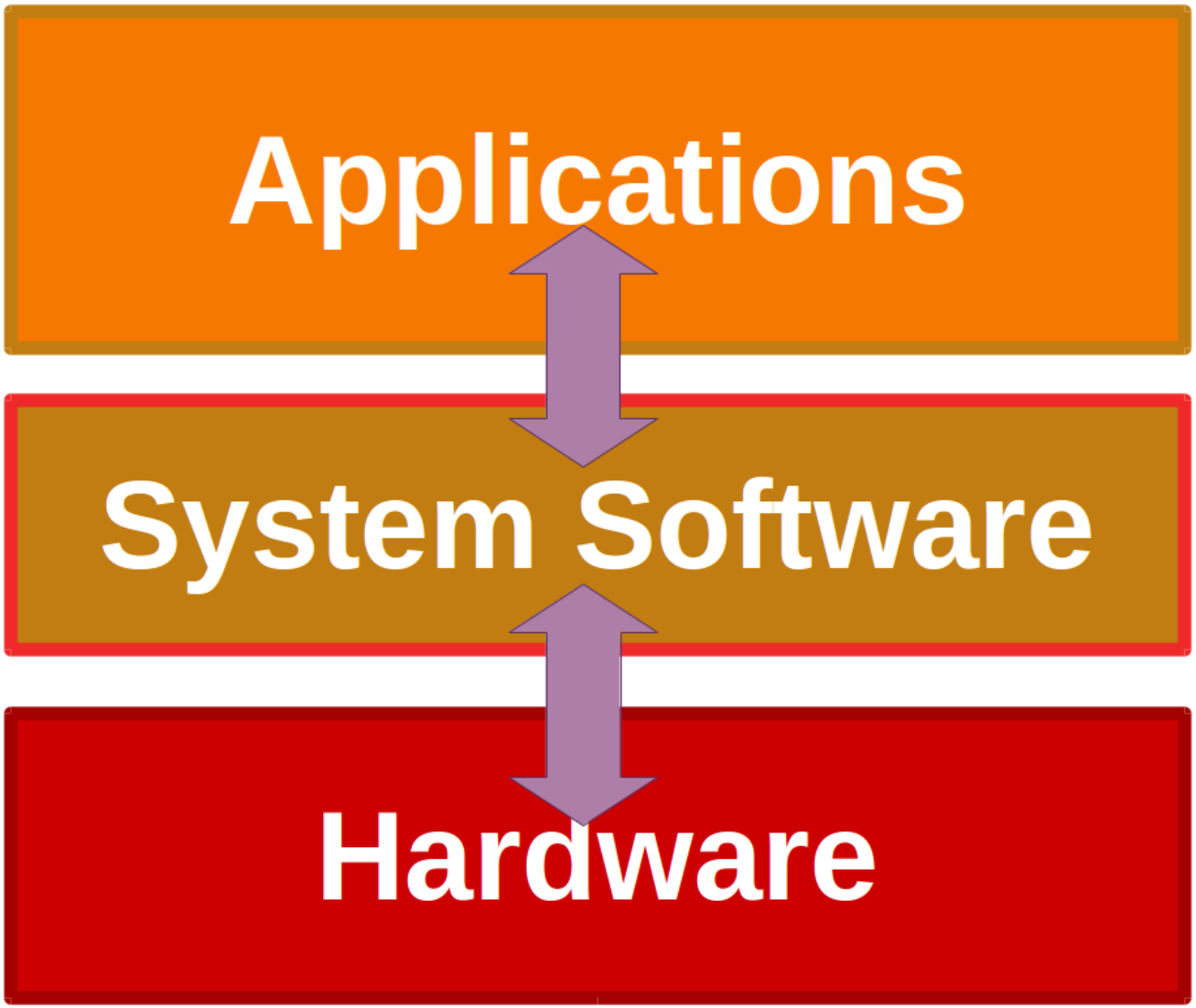
Stéphane Zuckerman

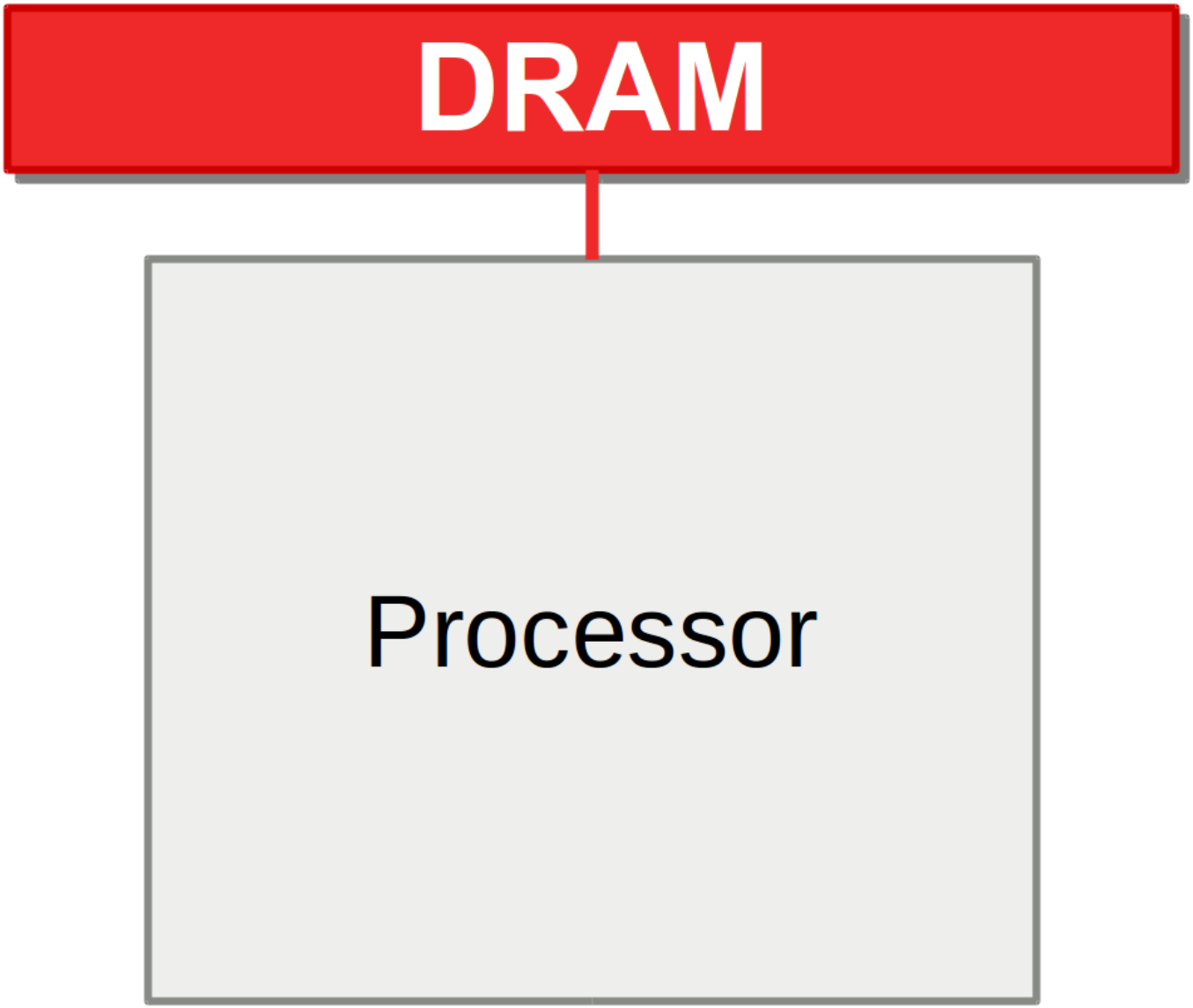
Computer Architecture & Parallel Systems Laboratory
Electrical & Computer Engineering Dept.
University of Delaware
140 Evans Hall Newark,DE 19716, United States
szuckerm@udel.edu

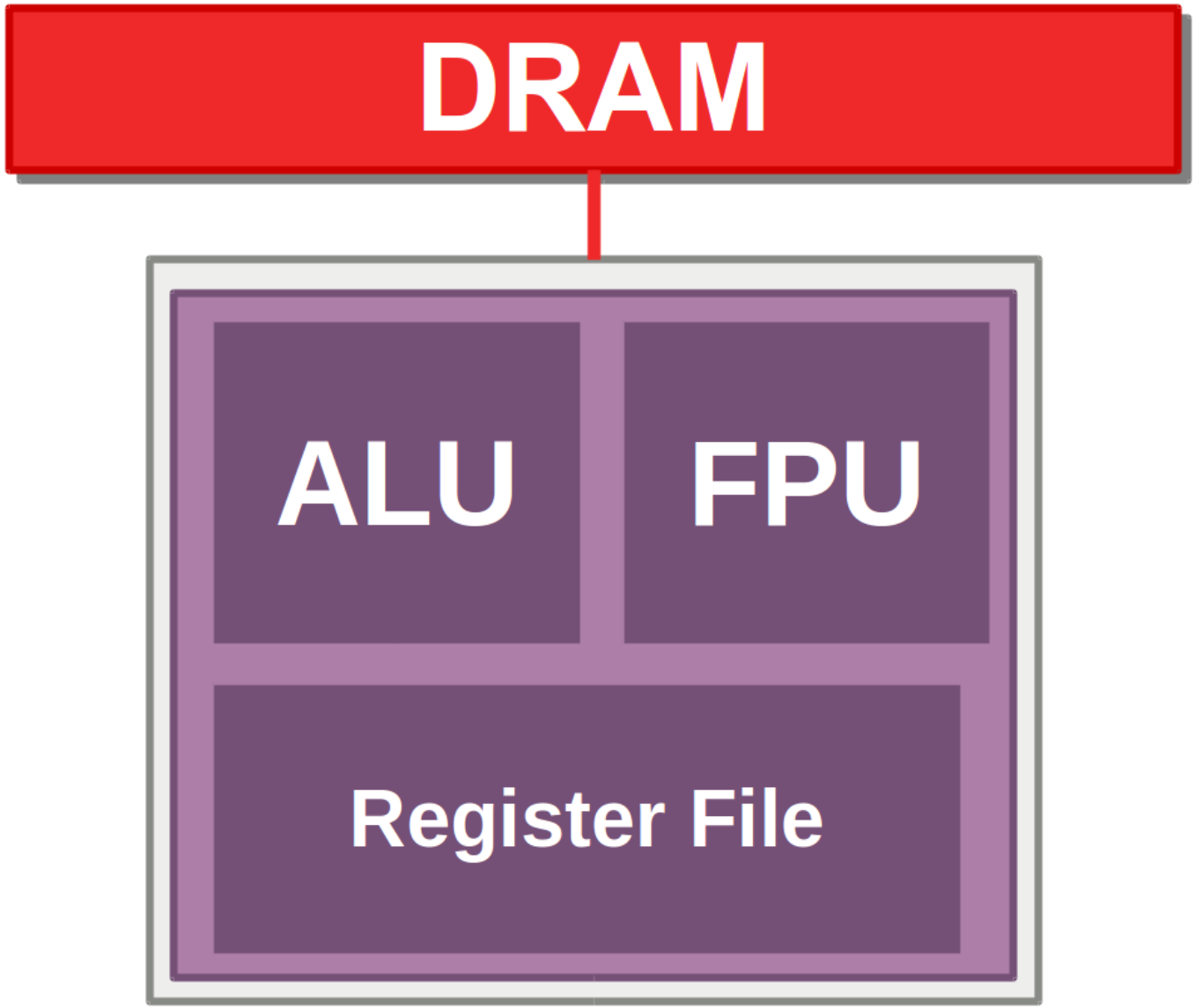
September 1, 2015

Outline

Outline

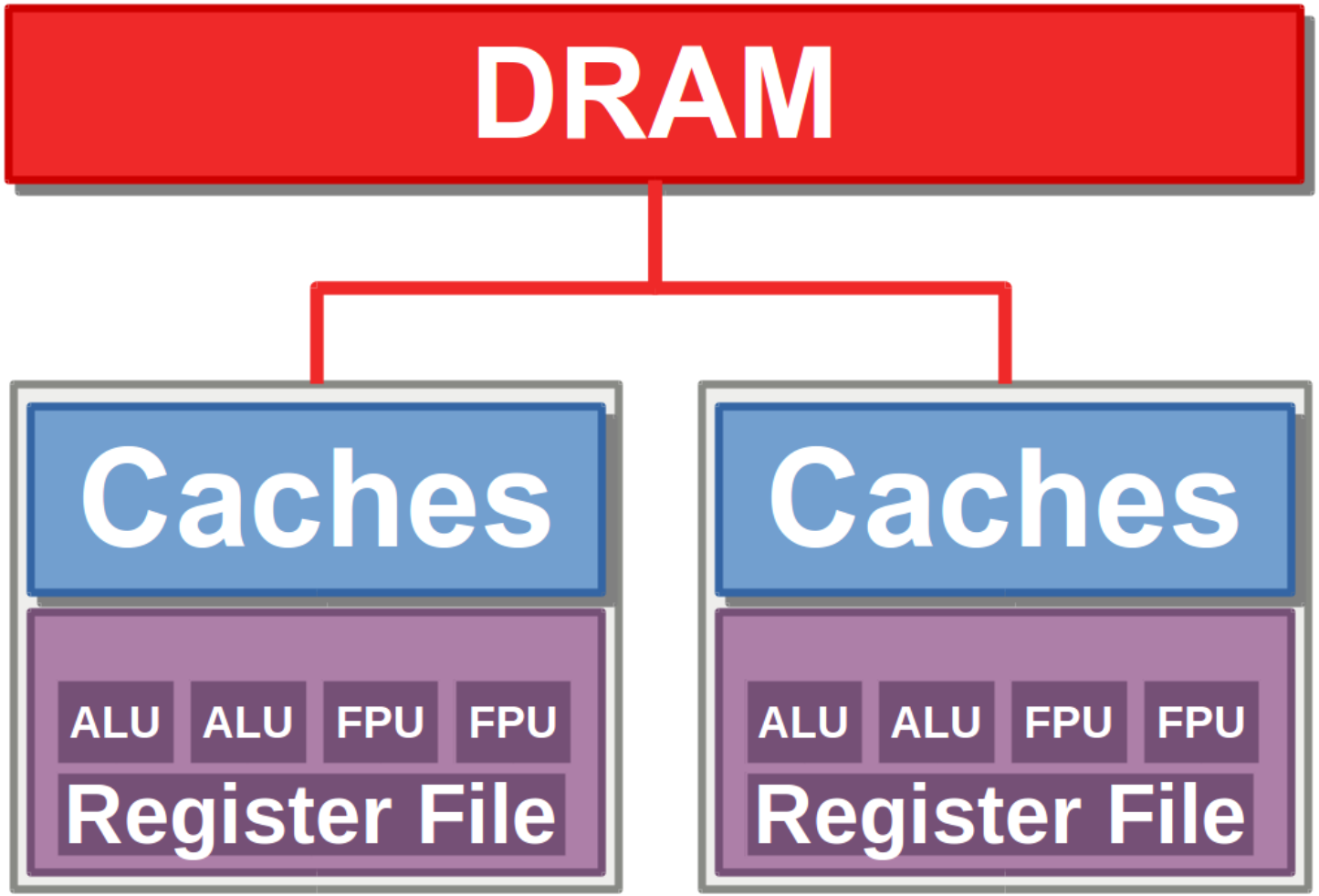

Applications
System Software
Hardware


DRAM
Processor


DRAM
ALU
FPU
Register File

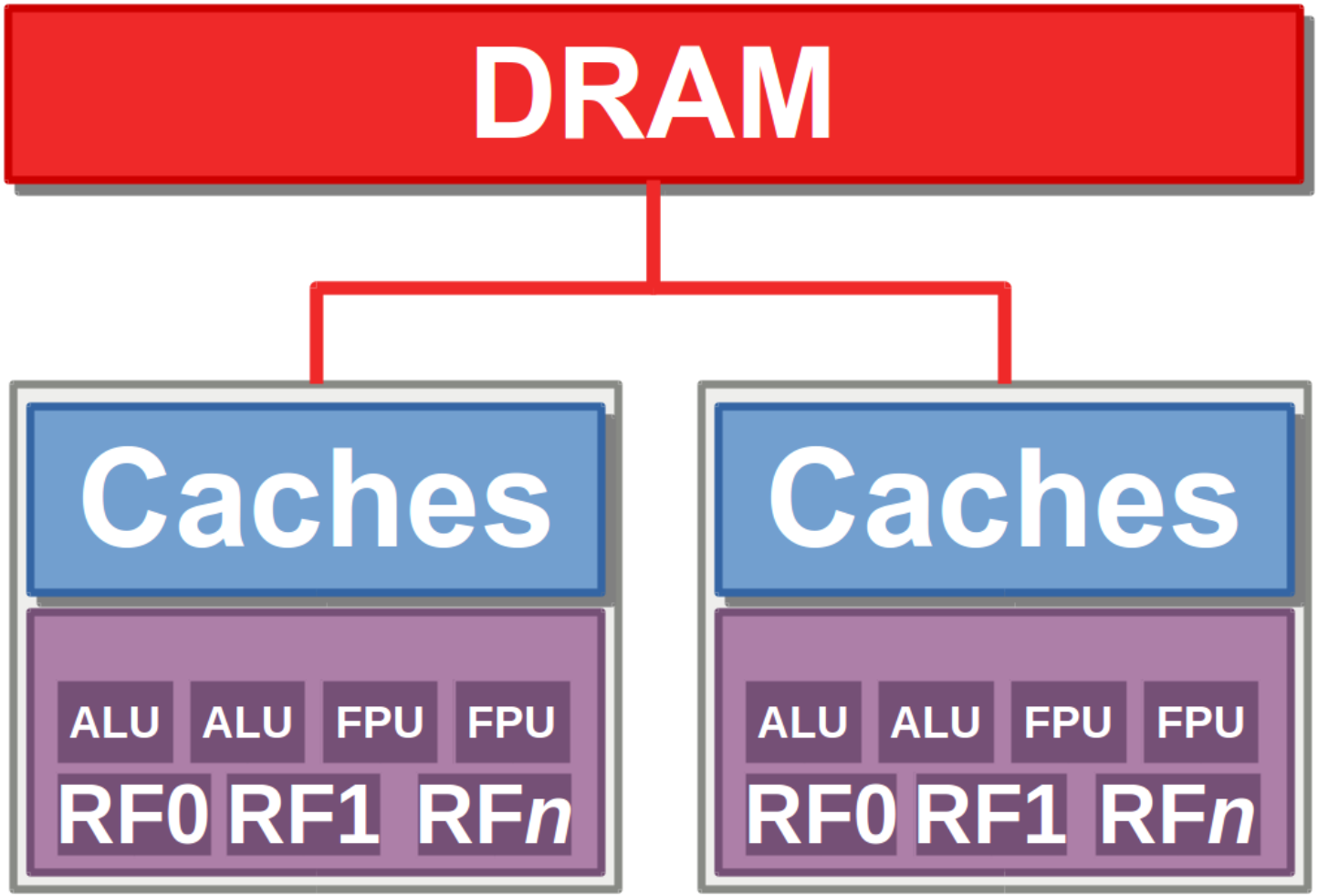
Modern High-Performance Computer Systems

Hardware



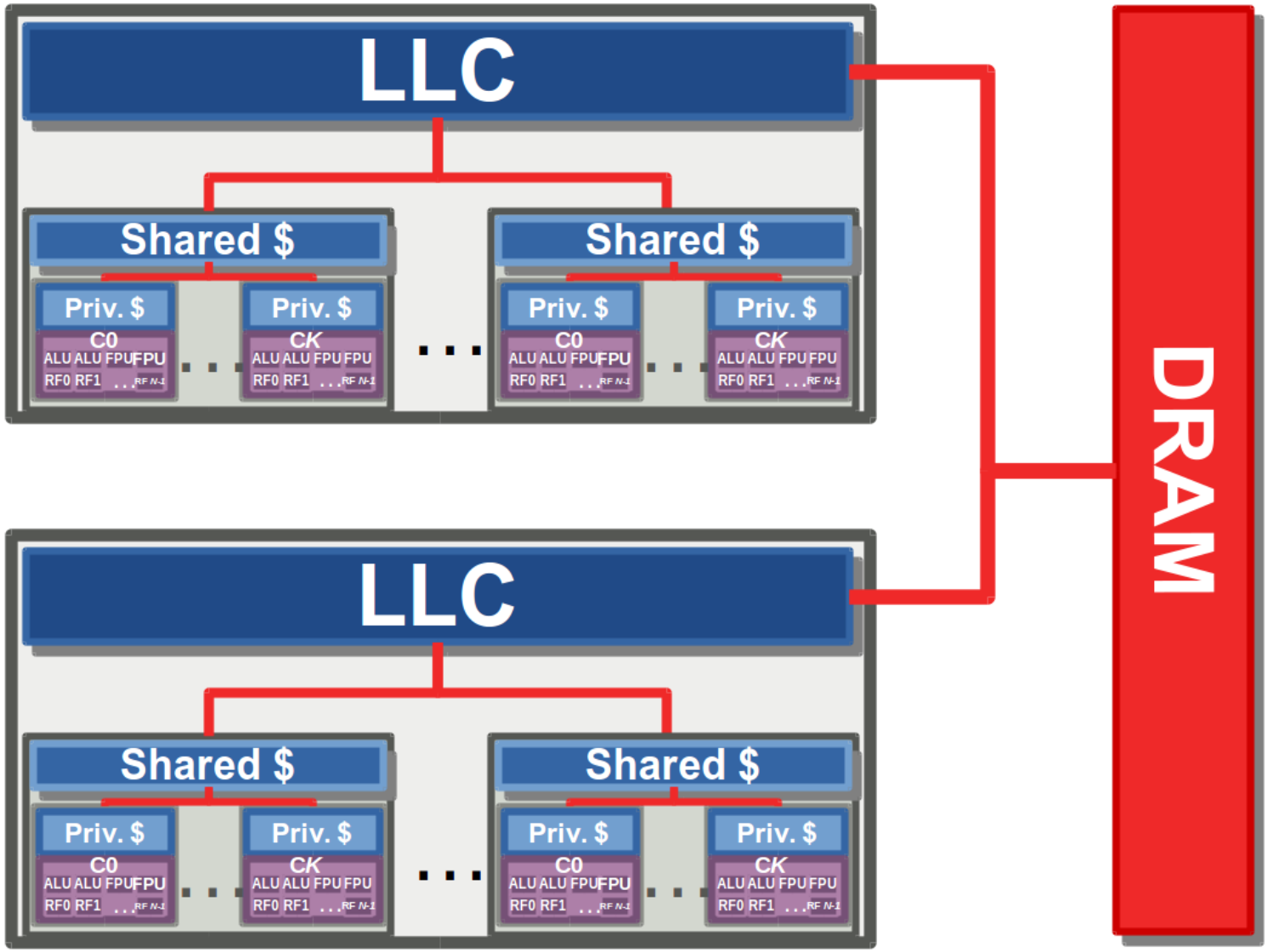
Modern High-Performance Computer Systems

Hardware



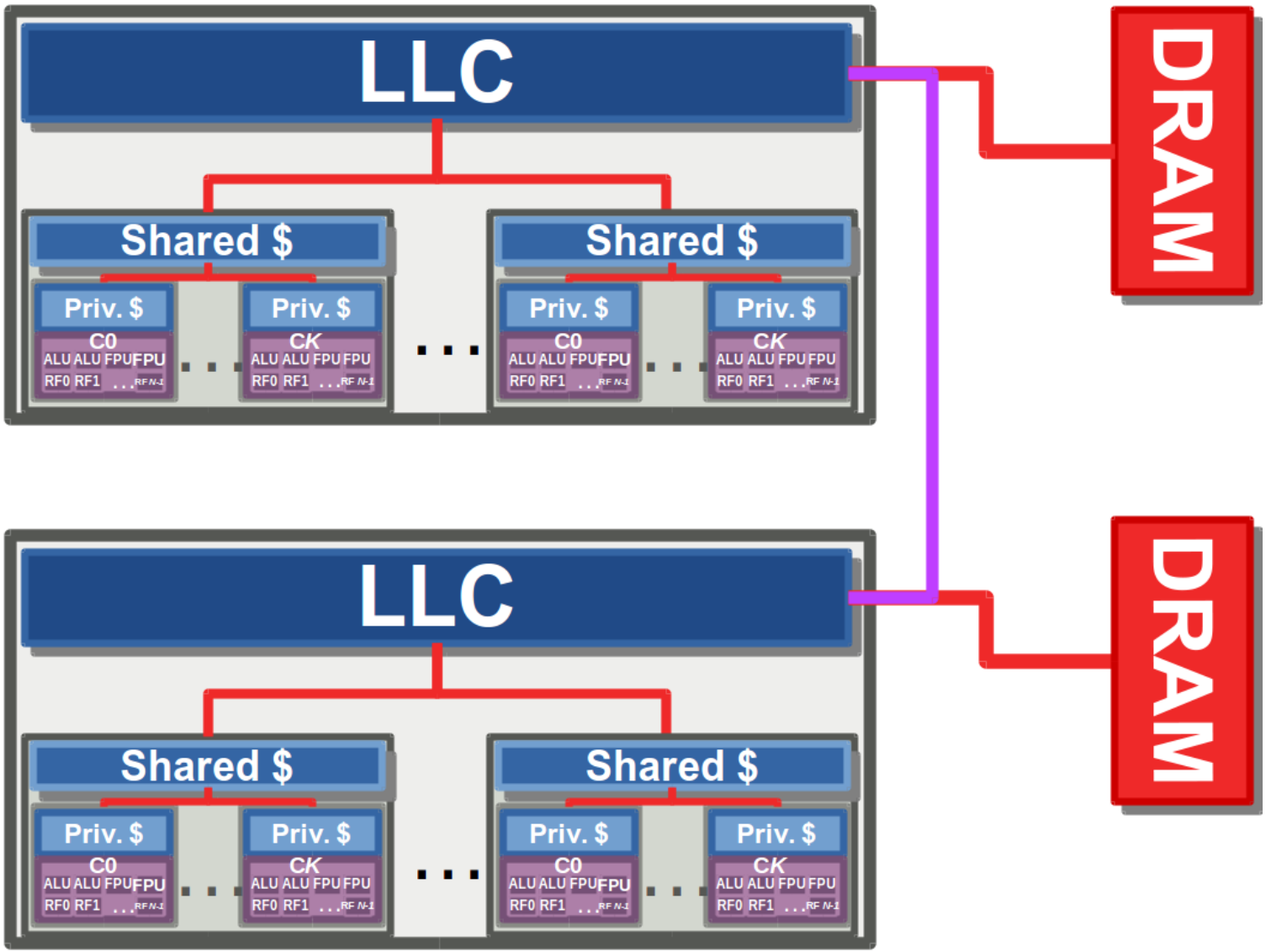
Modern High-Performance Computer Systems

Hardware



Modern High-Performance Computer Systems

Hardware



HPC Systems: a High-Level View

- One or more compute nodes.They are connected through fast interconnection networks, *e.g.*, 10Gbps Ethernet, or Infiniband.
- Each node contains one or more chips, connected through some kind of fast on-board interconnect.
- Chips contain **billions** of transistors. They use them to provide various ways to parallelize program executions in different ways.
- Possibly additional accelerators/co-processors.

Micro-Processors Overview

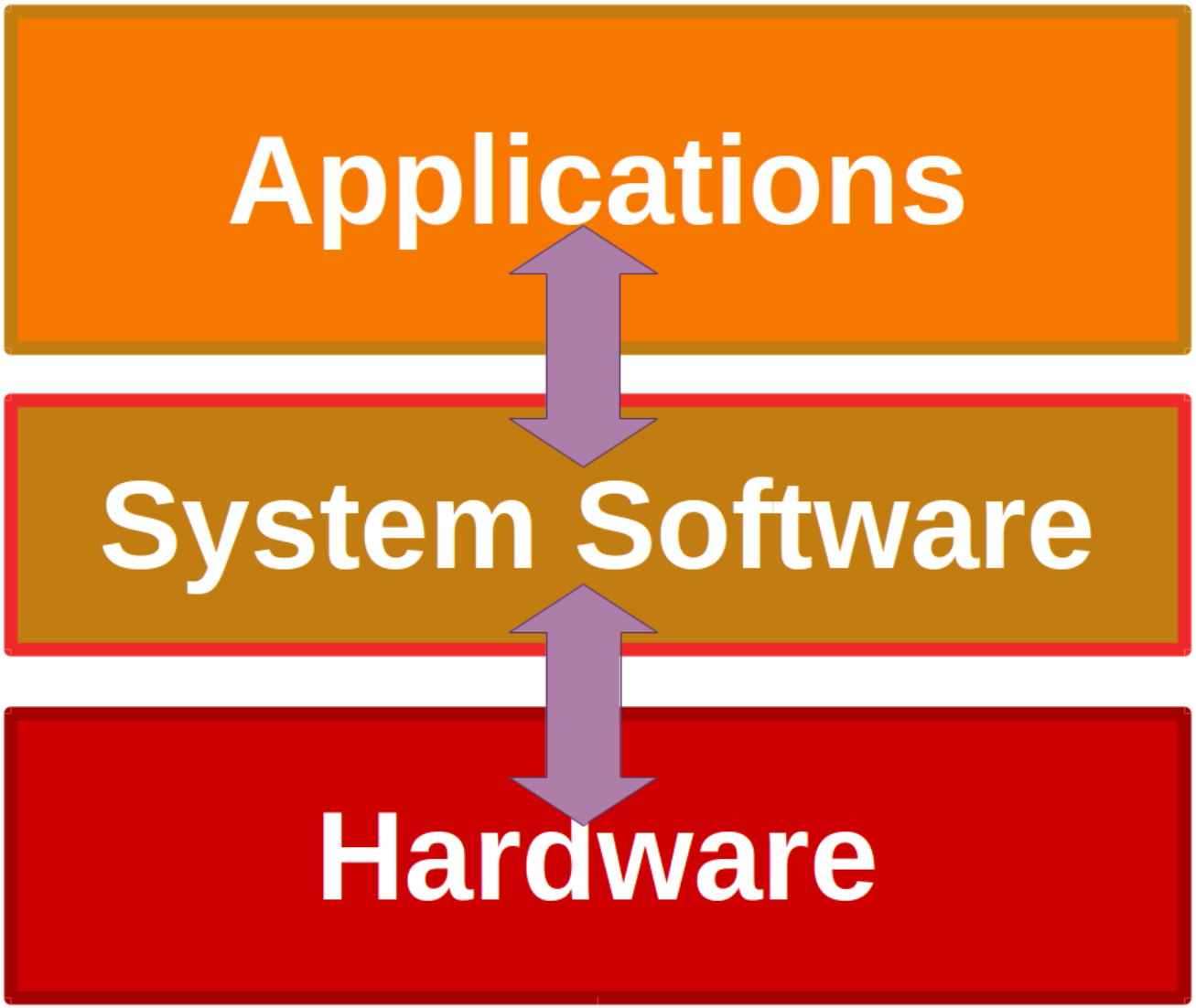
- Chips contain **billions** of transistors. They use them to provide various ways to parallelize program executions in different ways.
  - At the instruction level (ILP)
    - From a micro-architectural point of view: super-scalar and VLIW
    - By leverage vector registers and instructions (*e.g.*, SSE/AVX for x86, or AltiVec for PowerPC/POWER)
  - At the thread level (TLP)
    - Micro-processors chips may embed multiple threads (CG-MT, FG-MT, SMT)
    - Micro-processors chips may embed multiple cores (CMP)

Types of Accelerators

- AMD or nVidia GPUs
  - Each GPU board embeds **thousands** of hardware threads
  - They are great at "embarrasingly" parallel tasks, but tend to be bad at handling branching
- Intel Xeon Phi
  - Made up of revamped 1st generation Intel Pentium (P54C), with additional vector instruction engines (SSE/AVX)
  - Currently: Up to 61 cores / 244 hardware threads
    - 60 cores / 240 threads are usable, as the last core is used for the OS
  - However: The Xeon Phi requires as much care as a regular GPU to efficiently optimize code for high-performance.
- FPGA boards
  - Used to provide some application-specific hardware acceleration
  - May be reconfigured to fit another application
  - But: reconfiguring an FPGA is **slow**.

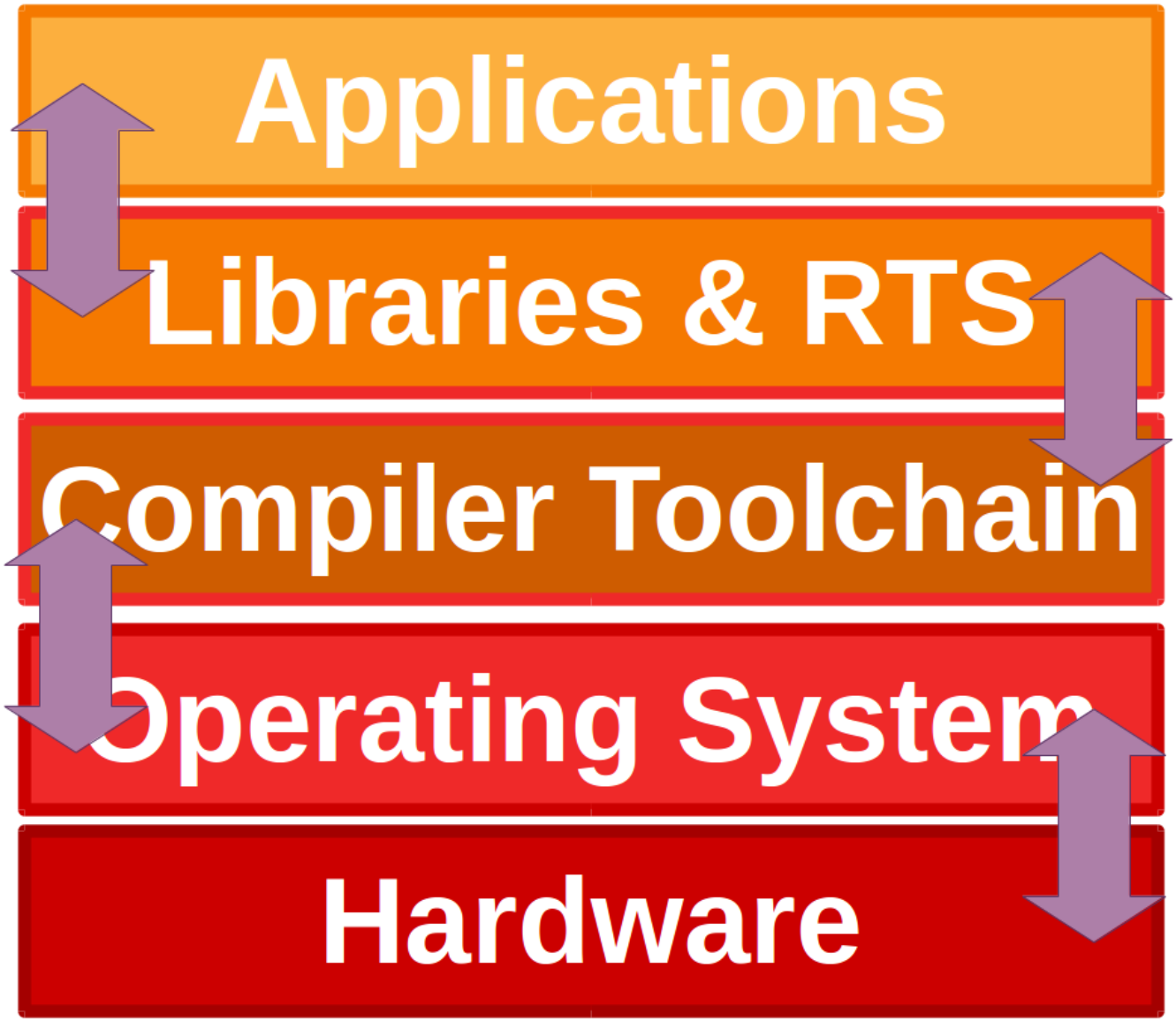
Modern High-Performance Computer Systems

Hardware & Software



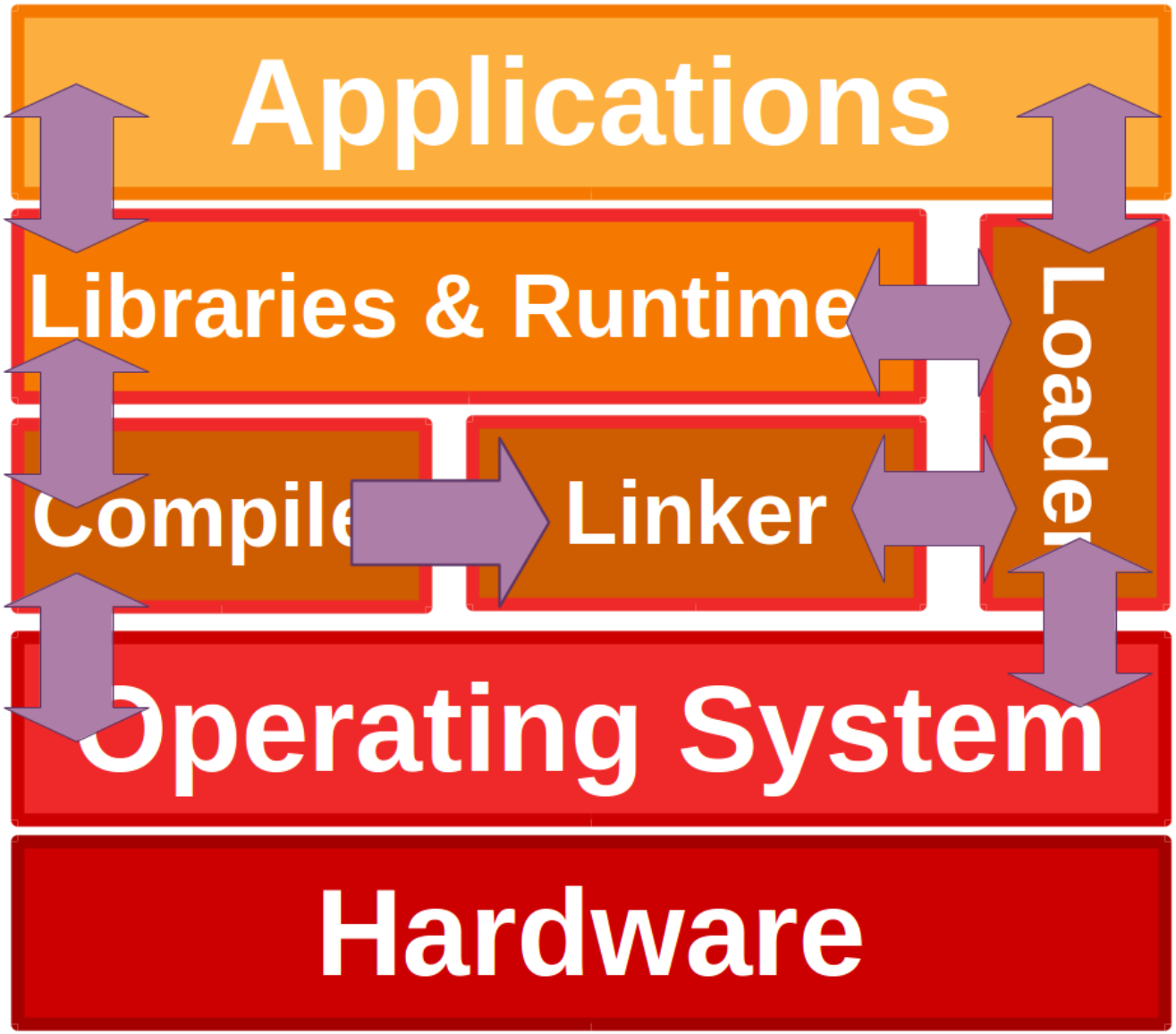
Modern High-Performance Computer Systems

Hardware & Software



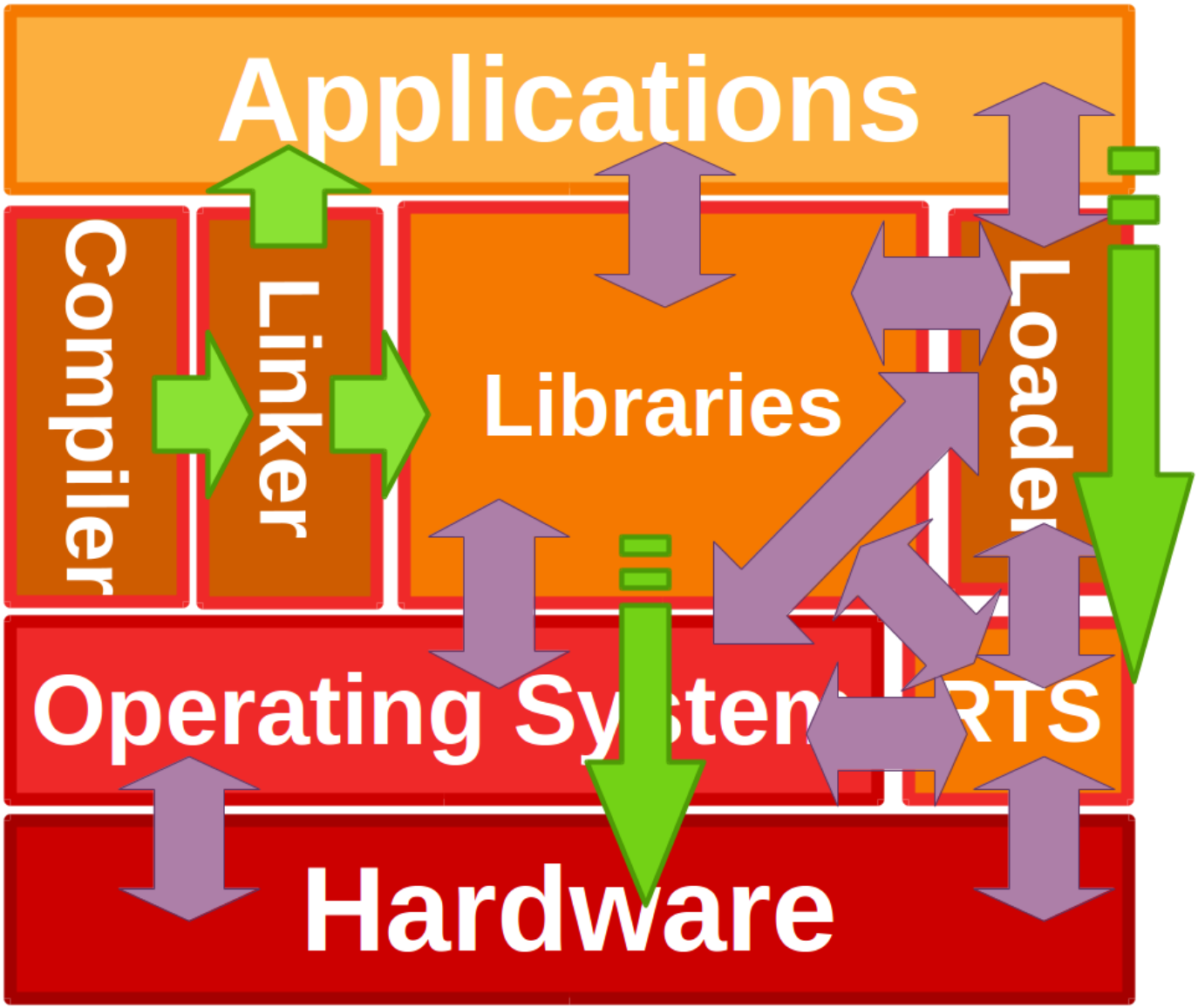
Modern High-Performance Computer Systems

Hardware & Software



Modern High-Performance Computer Systems

Hardware & Software



HPC Systems Software: Overview

Writing bare-metal code is doable, but extremely complex. As a result, a whole software ecosystem has been developed over the years.

- The operating system
- The compiler toolchain
- Various runtime systems
- Other tools which bridge everything mentioned above

The Operating System (OS)

It provides an interface between the hardware/software resources and the user.

- Handles resource management: I/Os, computing resource allocation, memory management, *etc.*
- In recent years, OSes have been heavily modified to take into account CMP and HW MT technologies
- OSes have to be **fair**: processes may have higher priorities, but eventually, every process must have a chance to run on the hardware.

Components of a High-Performance Computer System III

The Software

Compiler Toolchain

- Input: a program written in a programming language
- Usual input: a program written in a high-level language
  - For some definition of "high-level:" technically, assembly language is high-level compared to machine code
- Output: a program written in a different language
- Usual output: a program written in a low-level language
  - for some definition of "low-level"
- The compiler must know about the underlying OS, so that it can produce binaries and libraries that can be launched by it.
- One example of direct interaction between the compiler and the OS: complying to a specific Application Binary Interface (ABI) format

Components of a High-Performance Computer System IV

The Software

Run-Time Systems (RTS)

Runtime systems are very diversified, and are meant to be specialized in certain programming tasks. For example:

- MPI, the Message Passing Interface, provides an efficient way to send and receive data in a point-to-point or collective manner over a network
- PTHREADS allow a programmer to create several threads of execution which can all share the same memory
- Some runtime environments simply provide a way to manage memory using garbage collection
- RTSes **do not have to be fair:** they have a specific set of goals to achieve, within the frame provided by the underlying OS (if it exists).
- *etc.*

Other Vital System Software

- Linkers take several object files produced by a compiler, and *link* them in a single executable entity, or a single library of functions.
- Loaders are given binaries to execute. They substitute symbolic addresses contained within the binary and replace them with the true address of functions to call.

However, how does one describe the *execution* of a program in such a complex system?

Outline

What Are Program Execution Models?

PXMs: A "Horizontal" Definition

A program execution model specifies the interface between a given set of hardware features and resources, and the software that tells them what to do.

What Are Program Execution Models?

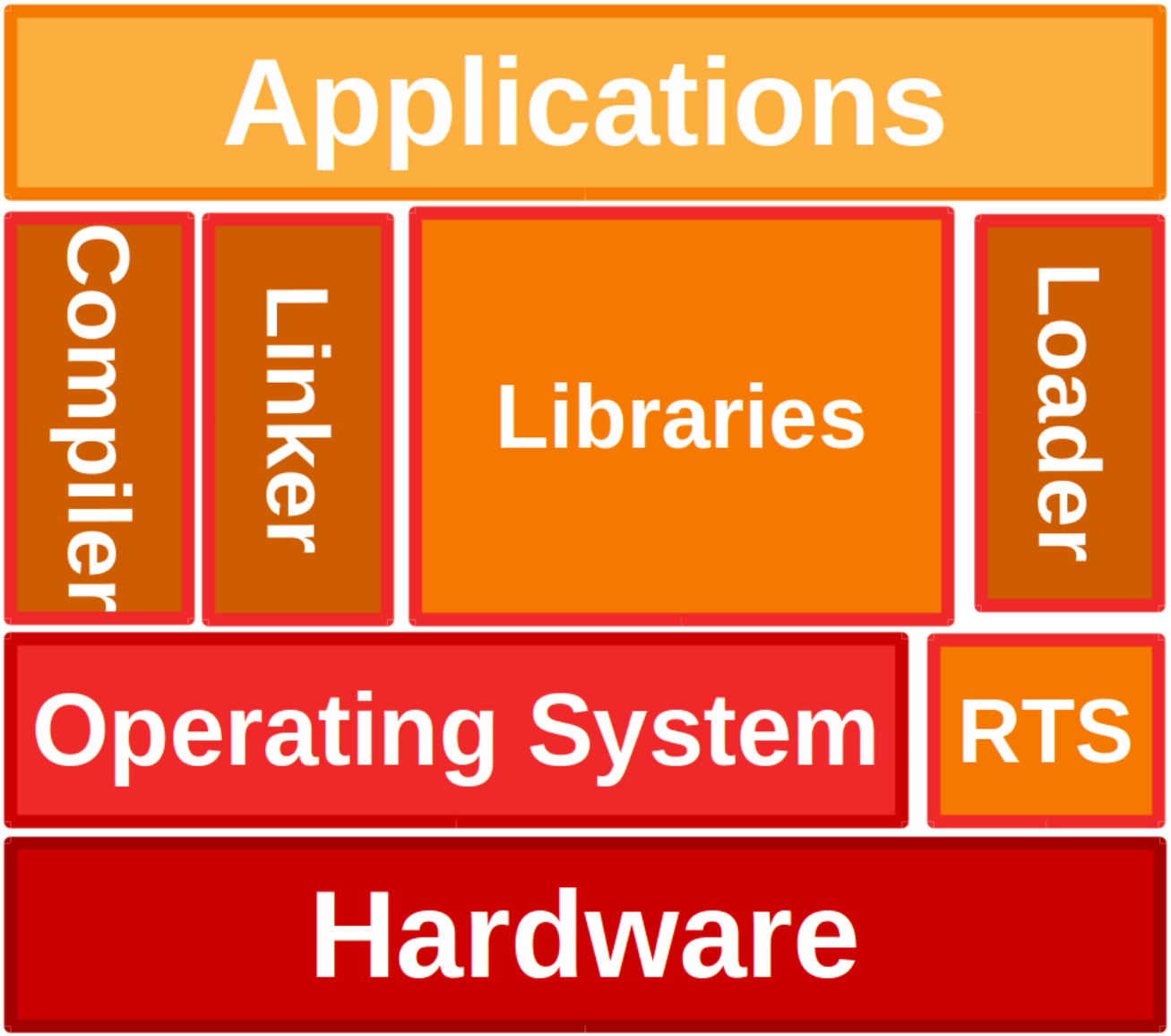
PXMs: A "Horizontal" Definition

A program execution model specifies the interface between a given set of hardware features and resources, and the software that tells them what to do.

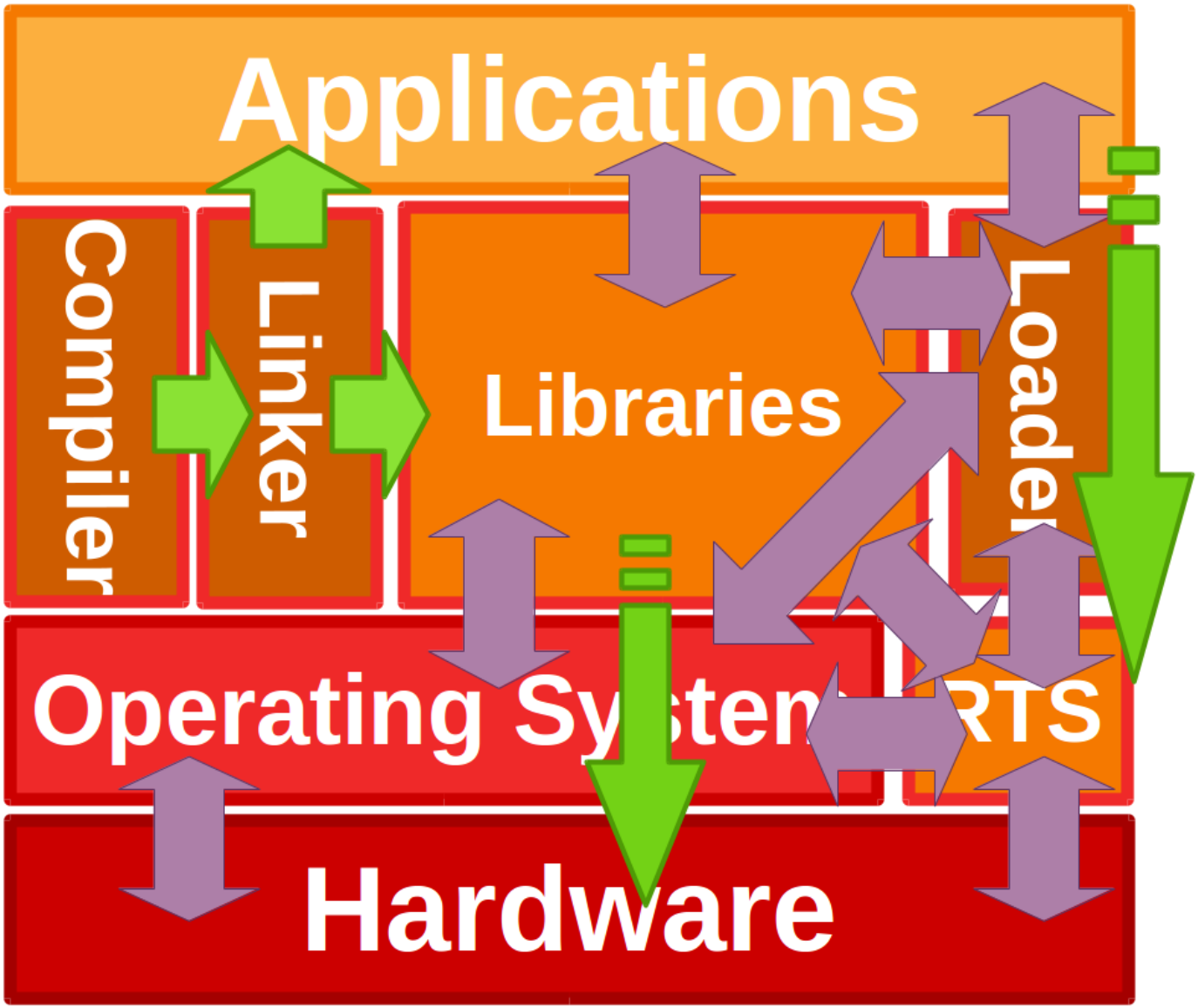
PXMs: A "Vertical" Definition

A program execution model specifies the way programs behave across all layers that compose a computer system: Hardware, software, and even user-related interactions.

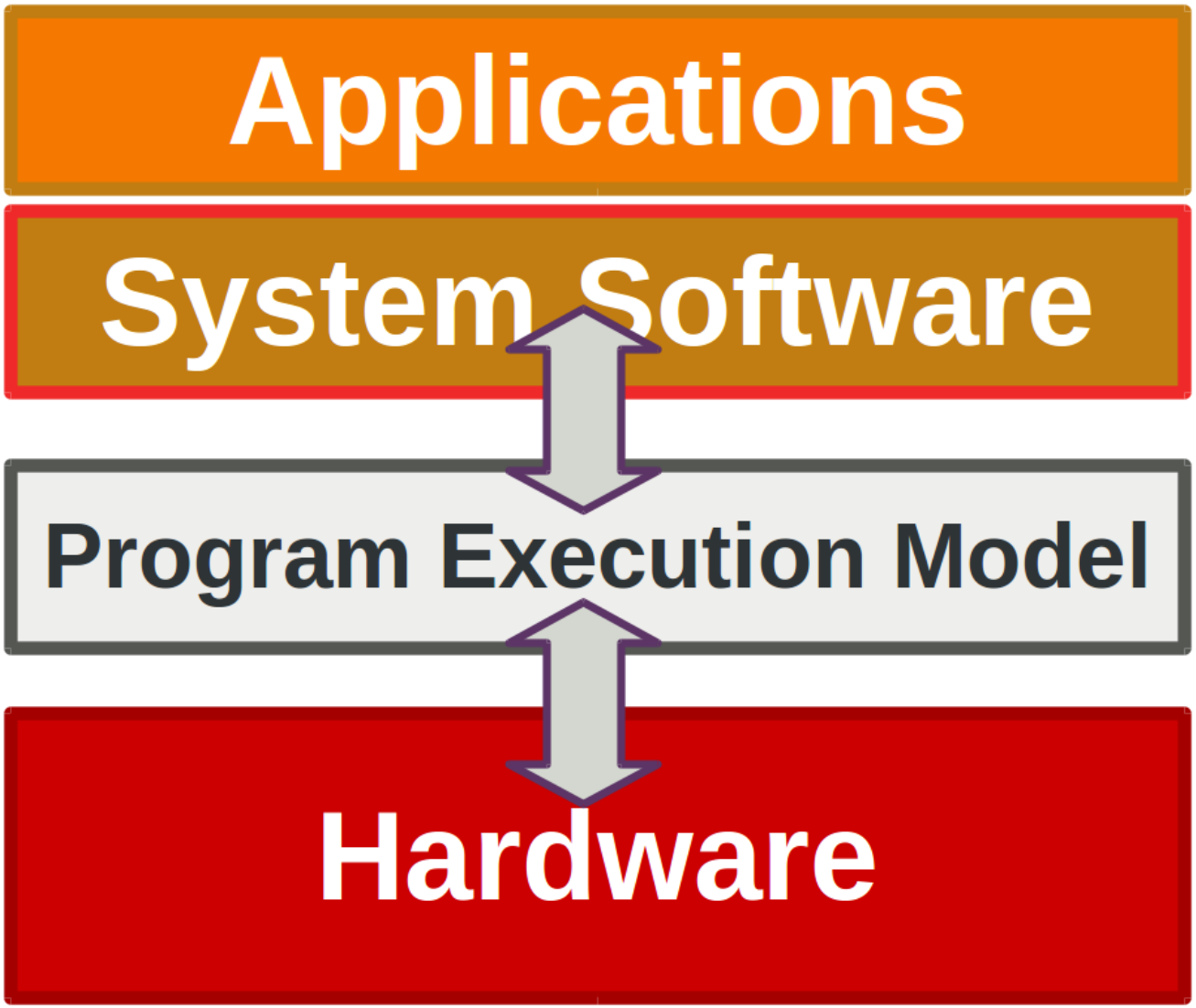
What Are Program Execution Models?



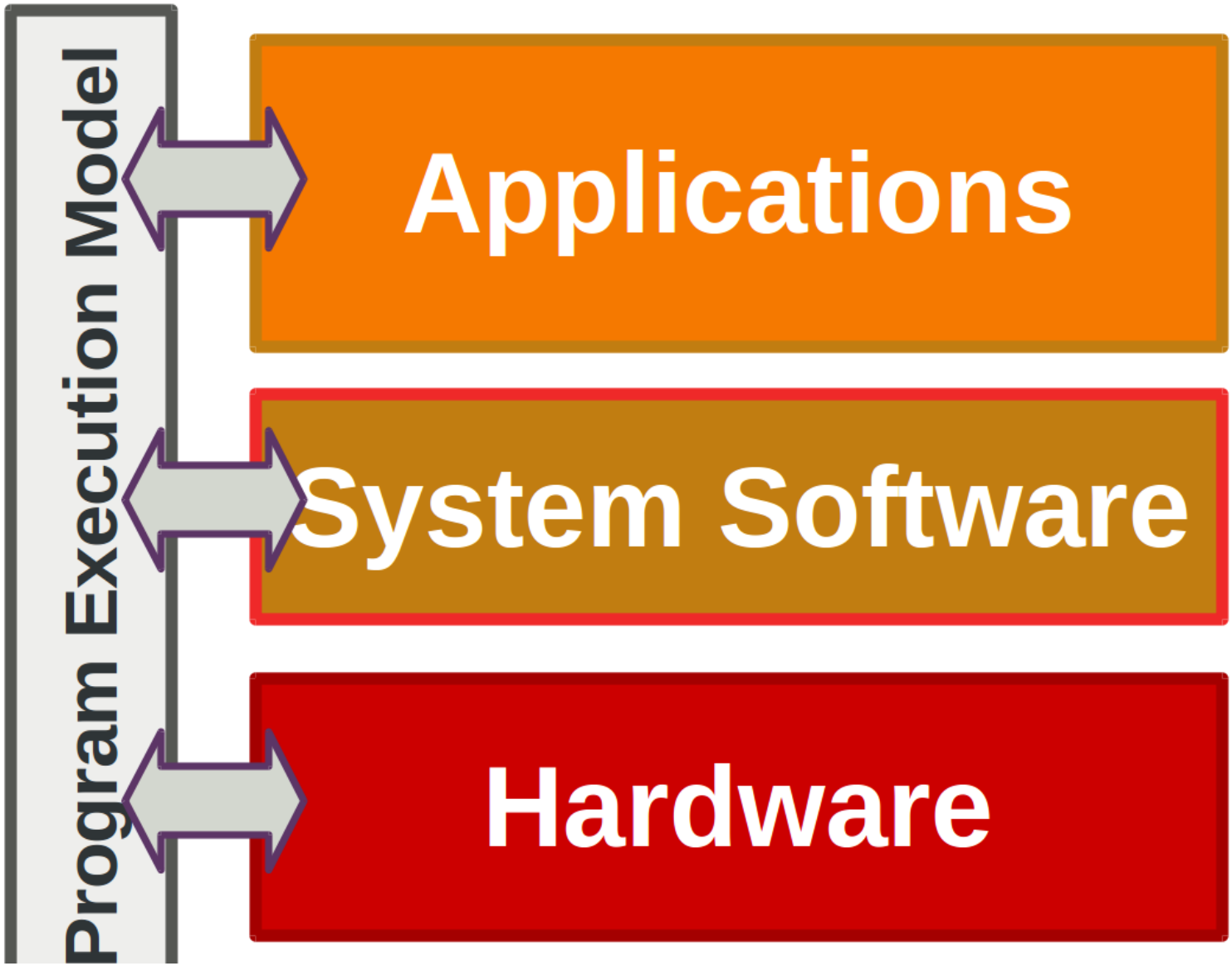
What Are Program Execution Models?



What Are Program Execution Models?



What Are Program Execution Models?



Components of a Program Execution Model

PXMs: Components

- A threading (or concurrency) model
- A memory model
- A synchronization model

Components of a Program Execution Model

The Threading Model

The threading model describes how to create concurrency. In particular, it specifies

- How to create a new thread, or a group of threads
- How to end an existing thread, or a group of threads
- How and when to schedule existing threads
- How and when to suspend and resume a thread

Components of a Program Execution Model

The Memory Model

Traditionally, memory models cover two aspects.

- The addressing mode. This includes:
  - **Shared vs. private memory.** Example: Threads vs. processes in UNIX
  - **Shared memory vs. distributed memory.** Example: PThreads vs. MPI
  - **Physically** vs. *logically* shared memory, *a.k.a.* Distributed Shared Memory (DSM)
  - **Physical vs. virtual memory**
    - Embedded systems tend to access memory directly through physical addresses
    - More general-purpose systems tend to rely on virtual memory
- The memory consistency model, *i.e.*:
  - If a memory location is concurrently accessed by two or more memory operations in a parallel system, what is the observed order?
  - What are the rules to alter memory ordering?
  - Can multiple values pertaining to the same memory location be observed by different actors?

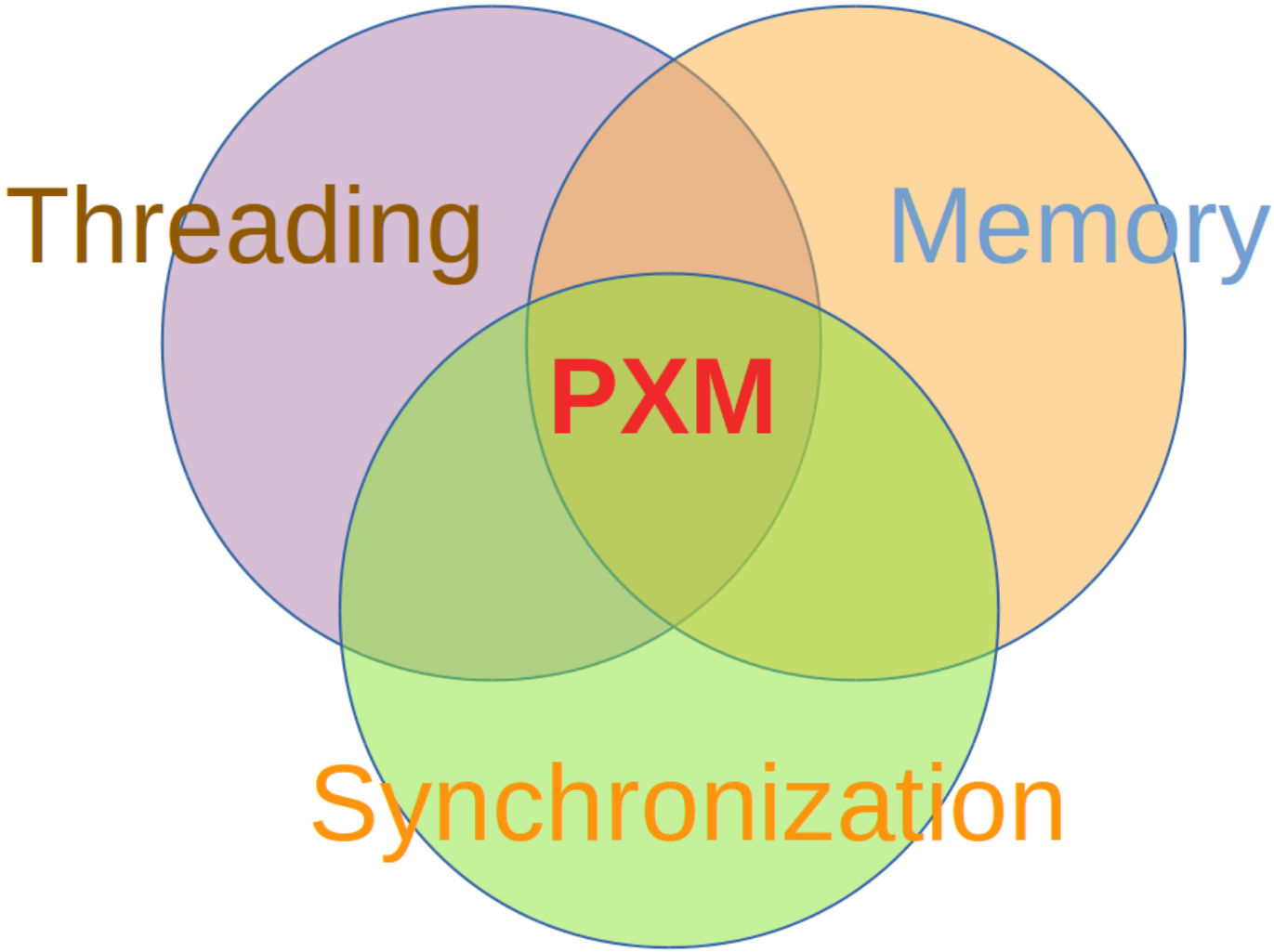
Components of a Program Execution Model

The Synchronization Model

It determines how concurrent and parallel threads can coordinate with each other. Some examples:

- When to be woken up after an event was triggered
- Whether some data transfer is blocking or non-blocking
- Acquiring some locking mechanism and thus guarantee atomic access to a region of code
- How to force a group of threads or processes to wait for each other before continuing the program's execution; *etc.*

PXMs in One Picture



Basic Components of the von Neumann Model

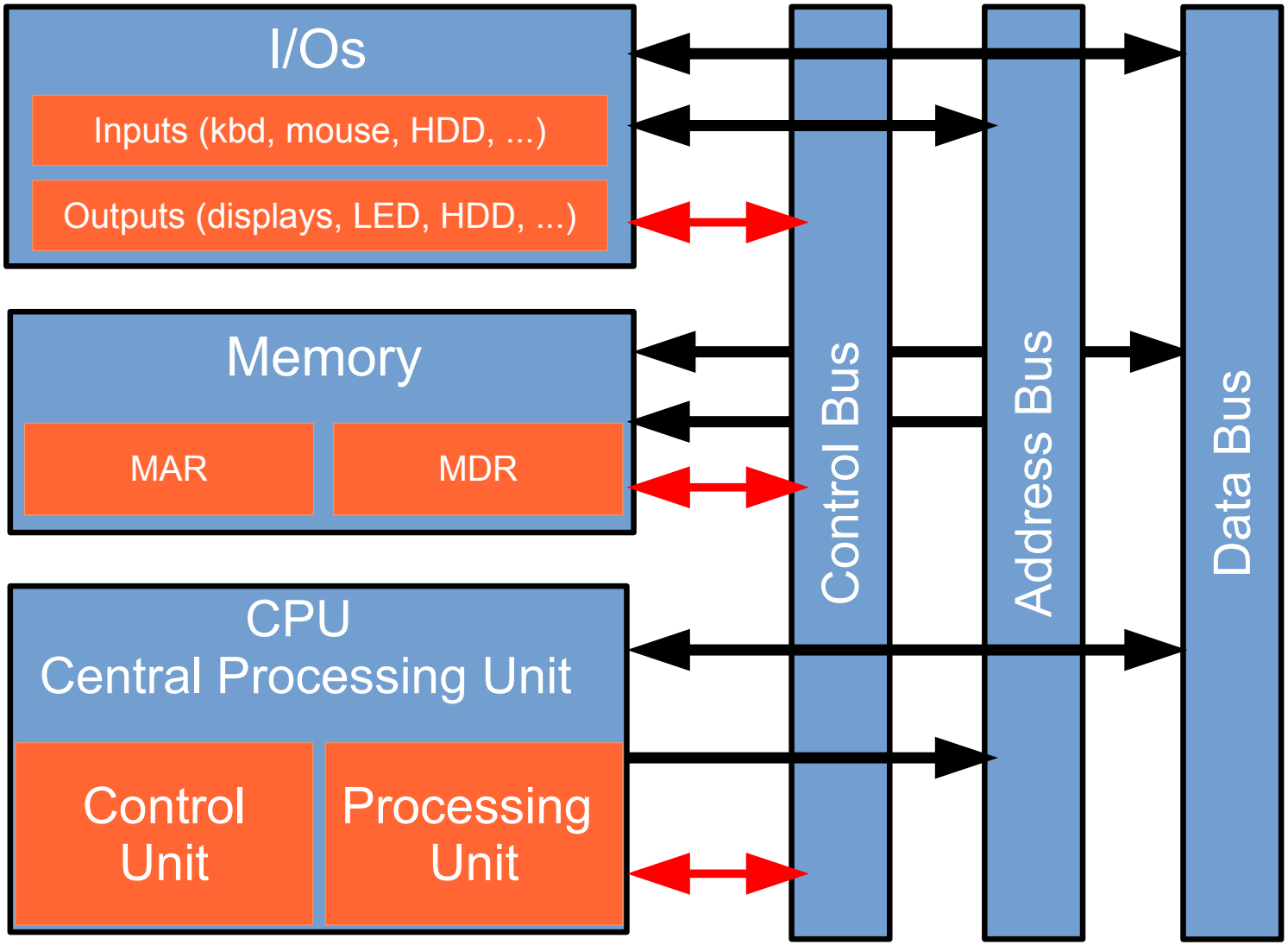
The von Neumann model relies on an abstract machine composed of:

- A control unit
- A central processing unit
- A memory unit
- An inputs/outputs "hub"
- Data, address, and control buses

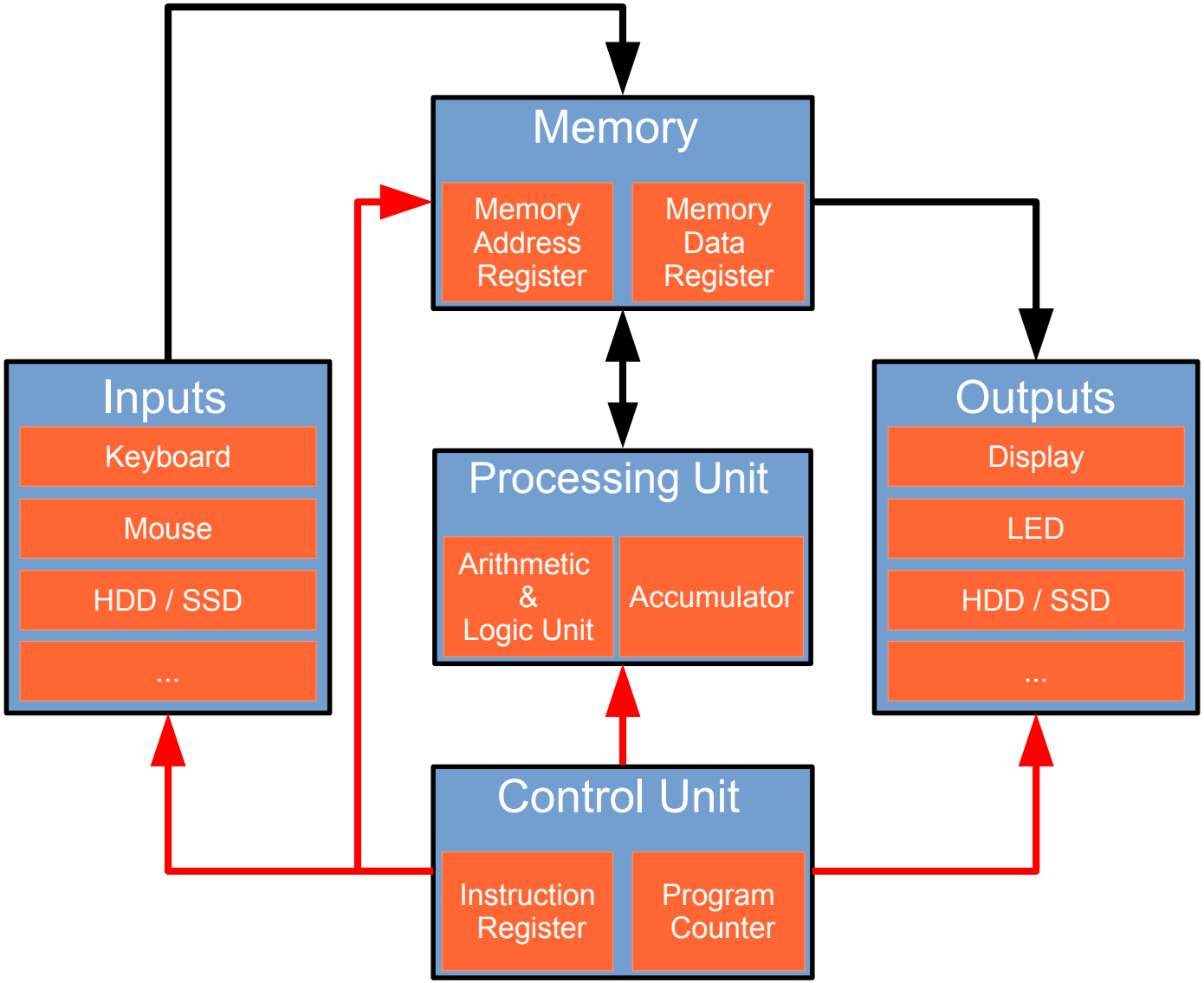
Execution Rules

1. The program counter (PC) contains the address of the next instruction to execute.
2. Automatically increment the PC by the size of the instruction currently executing.
3. Branching instructions may overwrite the PC with arbitrary values.

The Von Neumann Model



The Von Neumann Model



Extending the Von Neumann Model for Parallelism

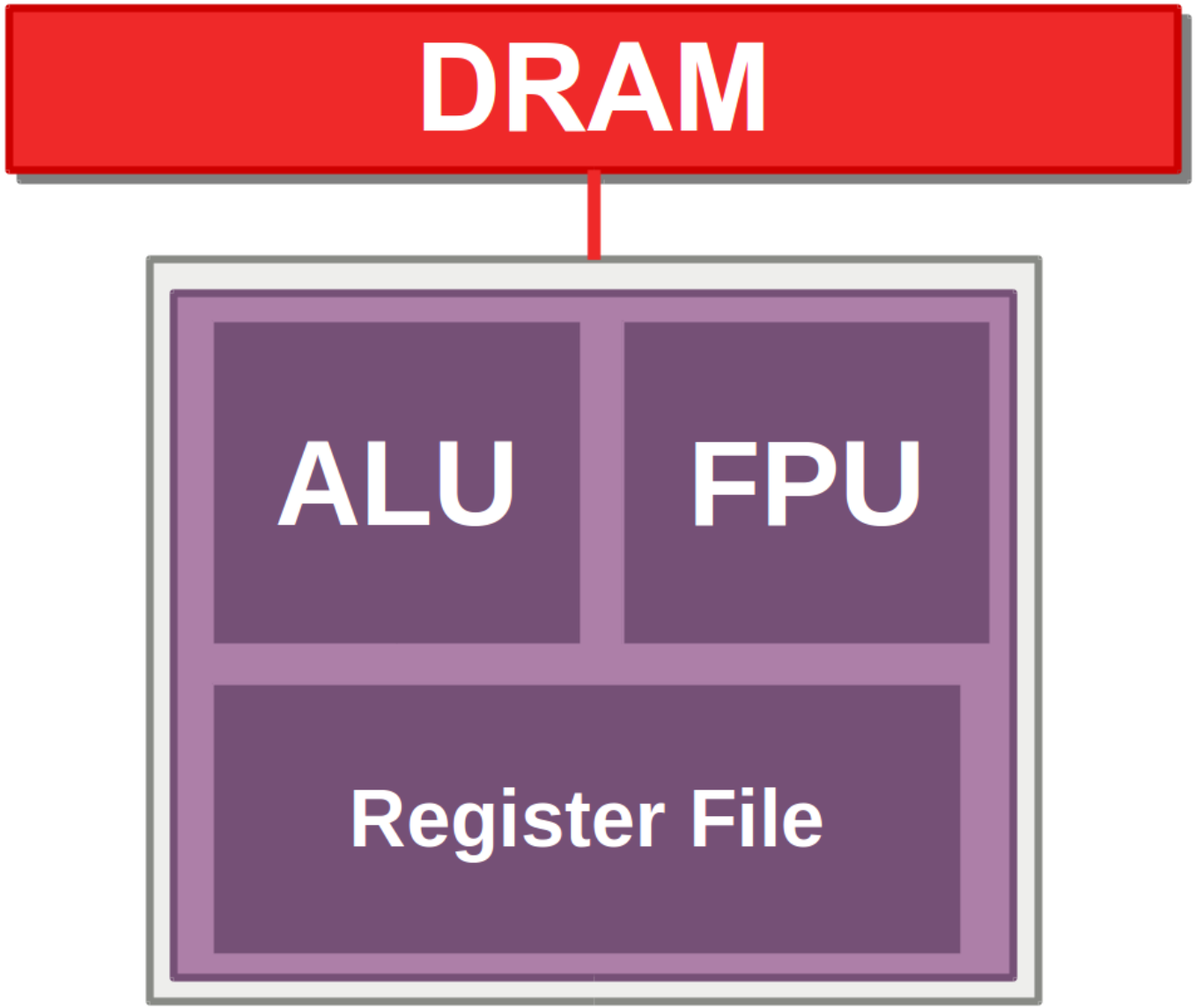
- Need to duplicate the program counters (PCs)
- What happens to memory accesses?
- What happens to inter-core/CPU synchronization?

Extending the Von Neumann Model for Parallelism

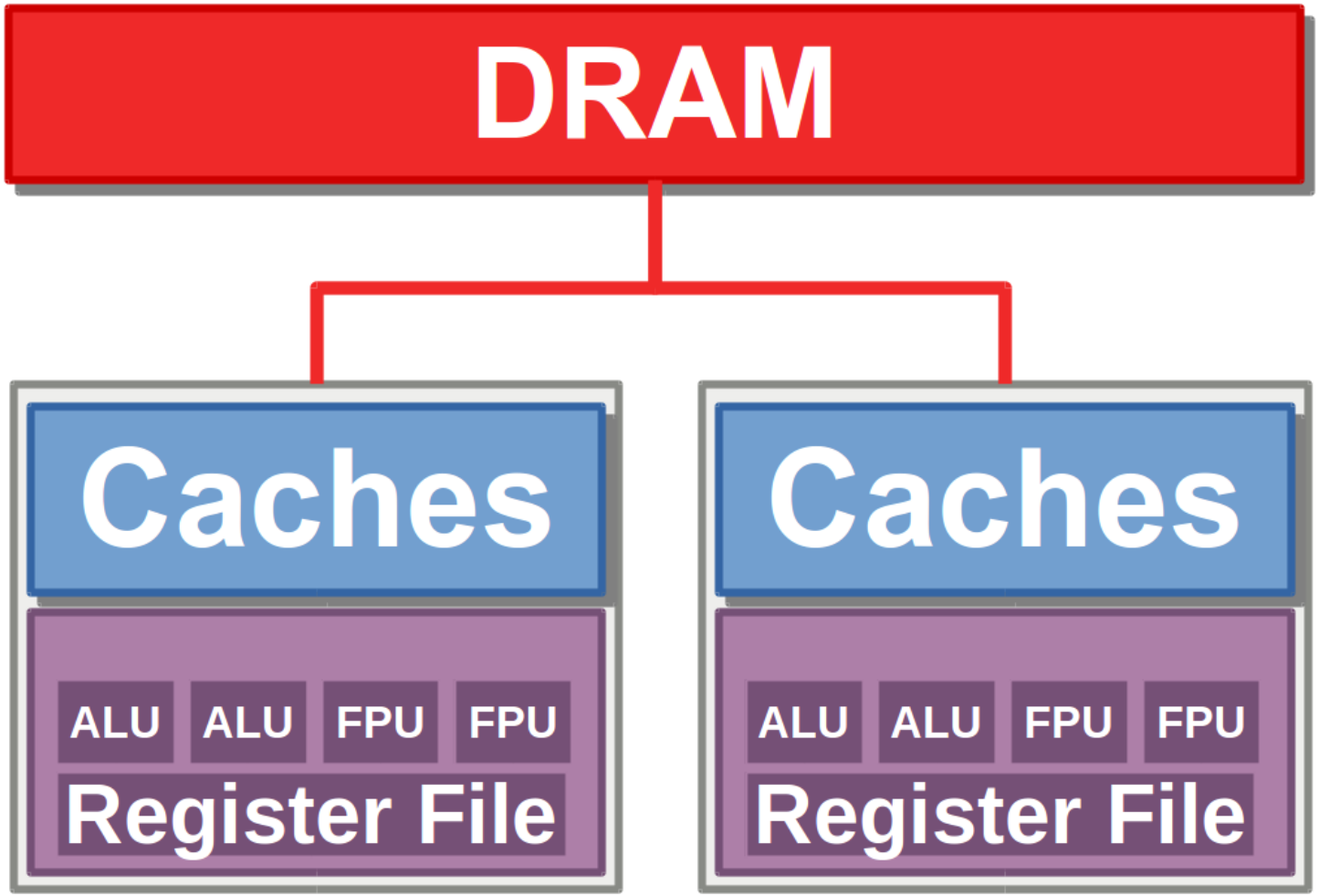
- Need to duplicate the program counters (PCs)
- What happens to memory accesses?
- What happens to inter-core/CPU synchronization?

No two parallel systems are alike when it comes to answer those questions.

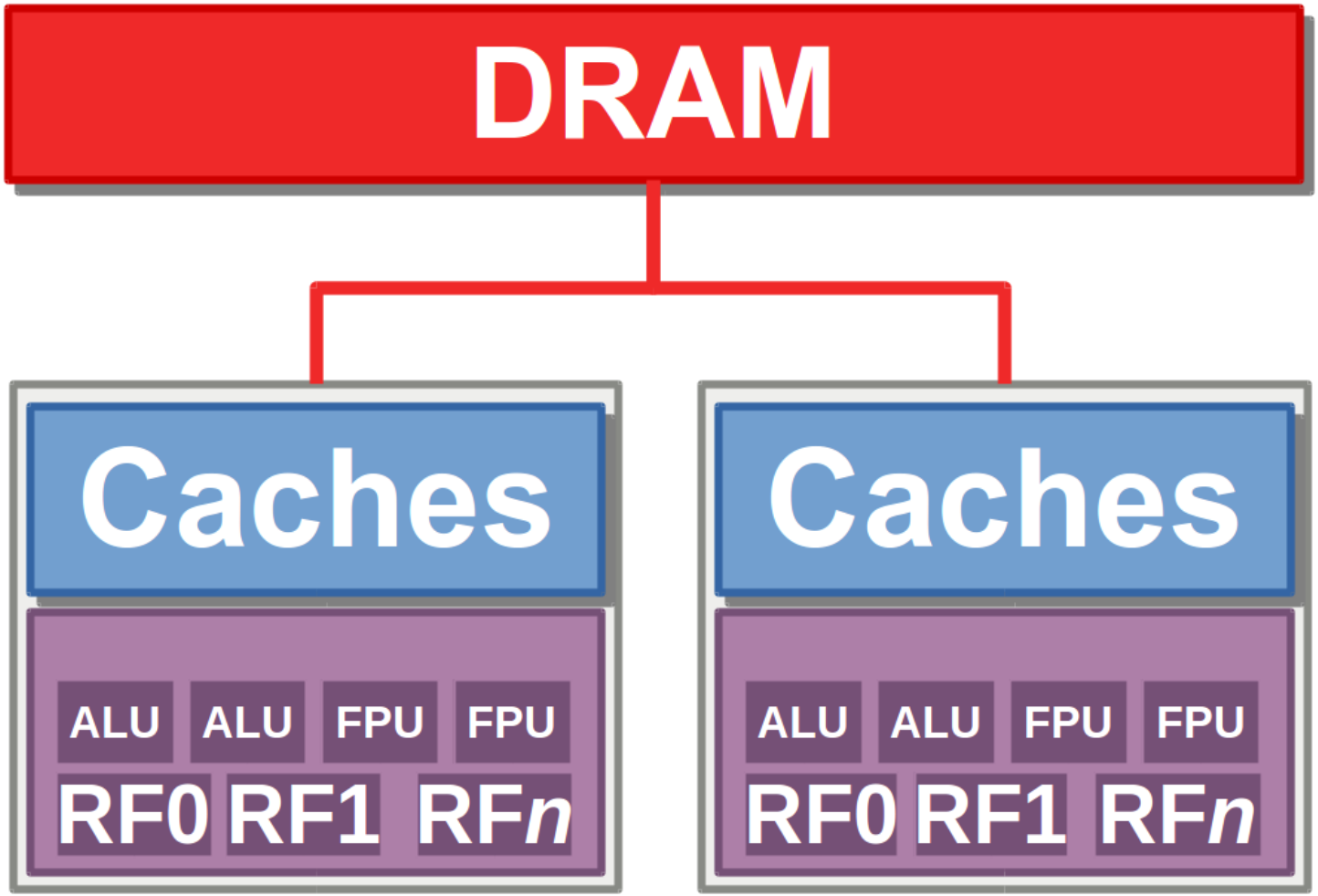
Extending The Von Neumann Model for Parallelism



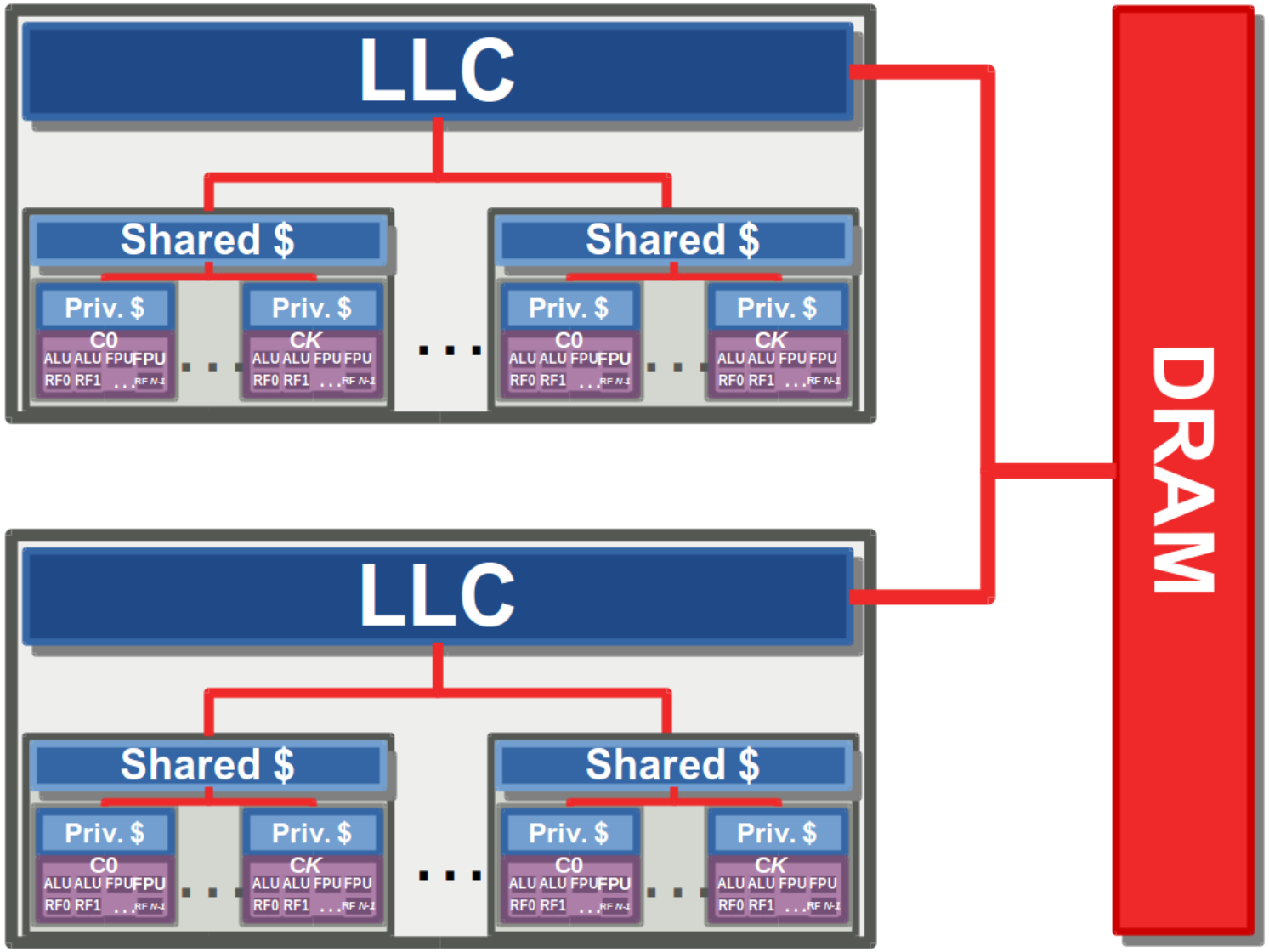
Extending The Von Neumann Model for Parallelism



Extending The Von Neumann Model for Parallelism



Extending The Von Neumann Model for Parallelism



Extending The Von Neumann Model for Parallelism