Computer Systems Design

Lesson 15

Basic communications optimization. Shared memory integration. Cache coherence

Alexander Antonov, Assoc. Prof., ITMO University

Hangzhou, 2025

ITMO

Outline the lesson

- Basic communication models
- Shared memory integration
- Cache coherence
- Memory consistency models
- ISA extension interfaces

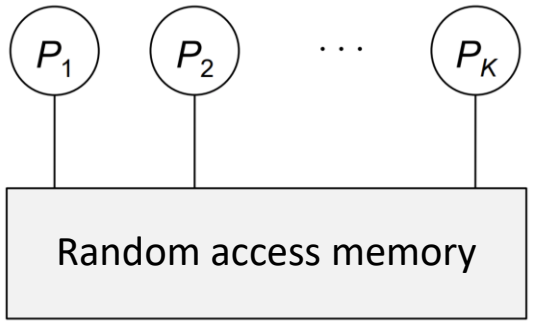
Basic communication models in parallel architectures

- Shared memory

  ***PRAM** – **P**arallel **R**andom **A**ccess **M**achine*

  Natural extension of traditional RAM machine

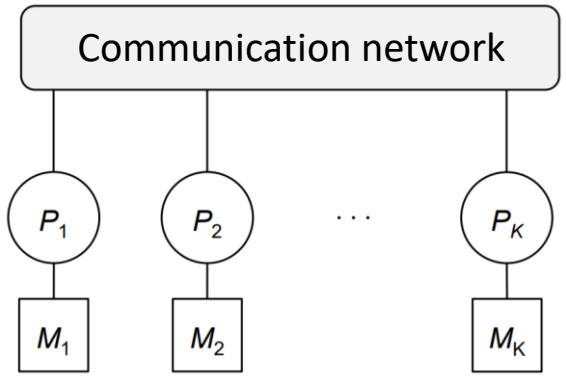
  New data is instantly (and implicitly) accessible for everyone, but memory bus can be a bottleneck



- Message passing

  Memory is distributed, computations are done locally and asynchronously, synchronization is done on demand

  Needs explicit data transfer, but more scalable



Shared memory integration

High-performance cores typically **master the interconnect (DMA – Direct Memory Access):**

- *CPUs*
- *accelerators: GPU, FPGA, …*
- *high-speed I/O controllers (e.g. Ethernet)*

Access locality may make ***caching*** effective

For correct synchronization, memory behaviour rules have to be specified:

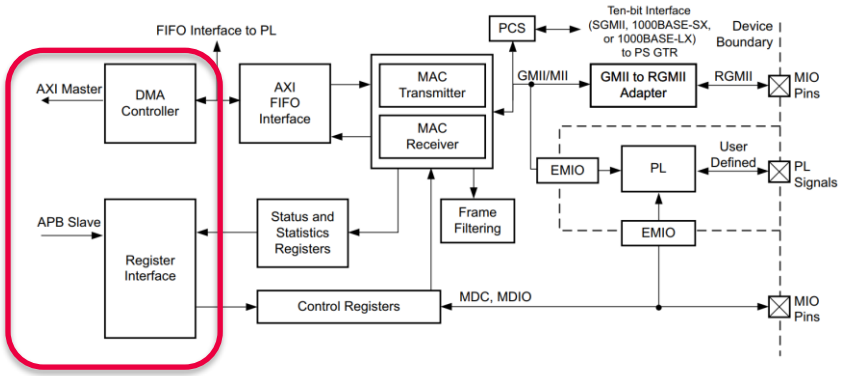
- consistency

  *ordering of accesses to **various addresses** from **single master***

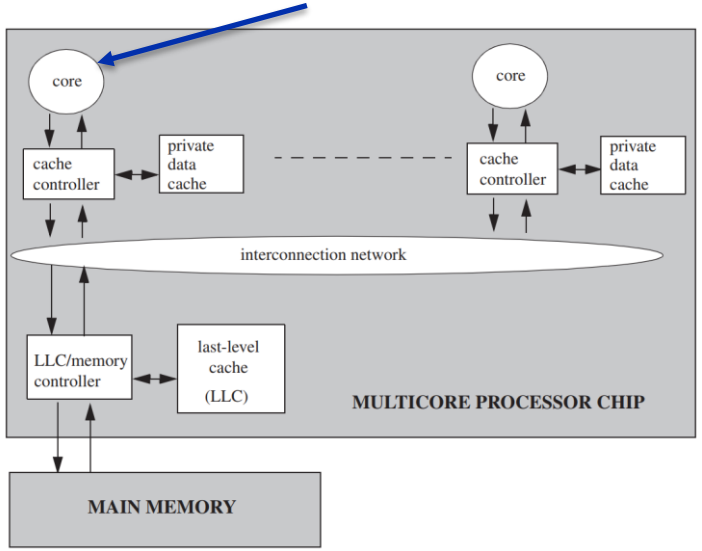
- coherence

  *ordering of accesses to **single address** from **multiple masters: read/write history** should be the **same** for all distributed cores*

Ethernet controller in Xilinx Zynq:



CPUs, accelerators, I/O

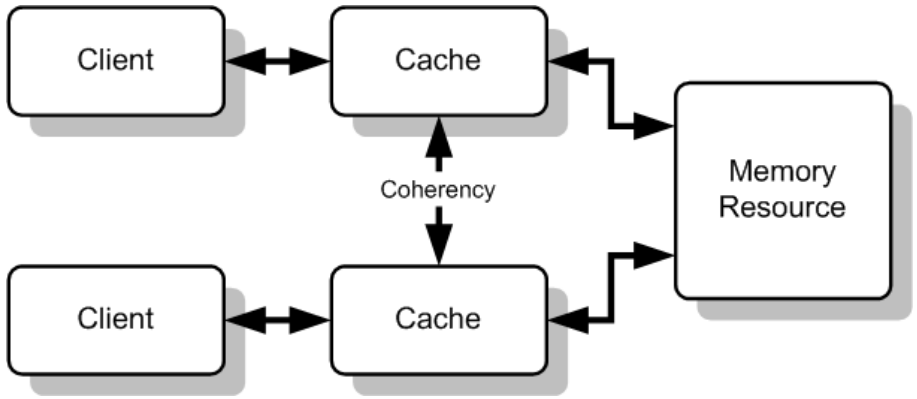


Cache coherency formulation

Can be formulated as the following invariants:

- ***Single Writer – Multiple Readers***
  Several masters cannot write in the same location simultaneously

- Cached copy of memory location always contains the ***latest*** version of data



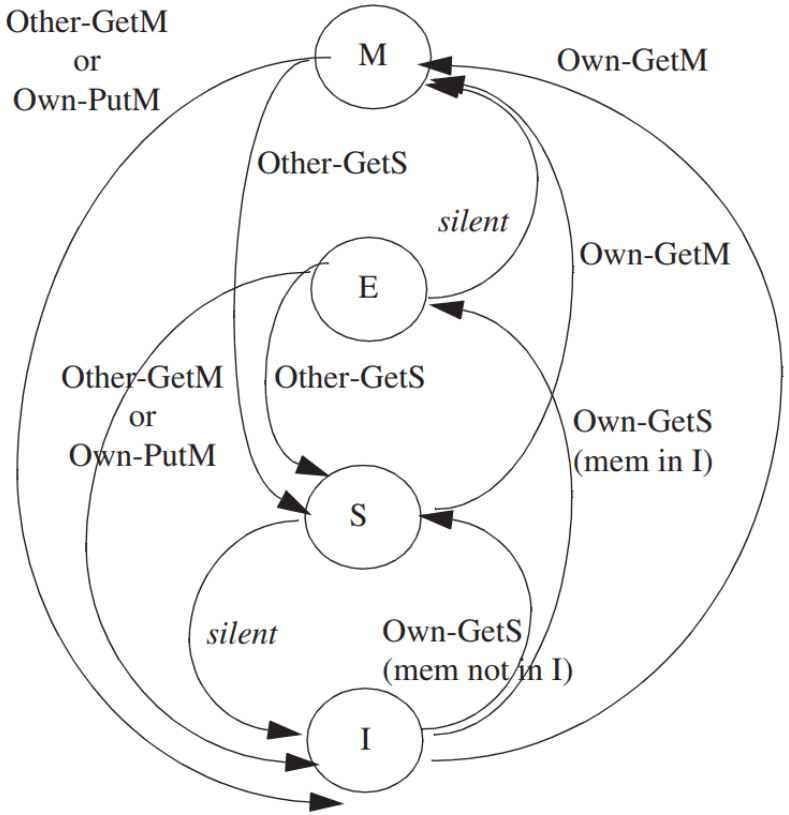
Daniel J. Sorin, Mark D. Hill, David A. Wood, A Primer on Memory Consistency and Cache Coherence, Morgan and Claypool Publishers, 2011

Classic cache coherence: MESI (Illinois) protocol (1984)

Each cache block can be in one of the following states:

- M (Modified)
- E (Exclusive)
- S (Shared)
- I (Invalid)

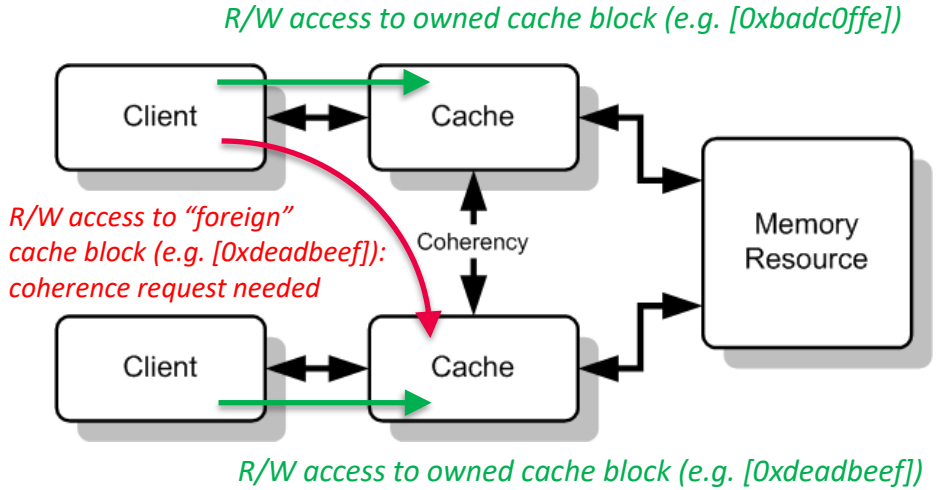
New states can be added:
*e.g. M**O**ESI (AMD), MESI**F** (Intel), etc.*



ITMO

MESI cache coherence: masters' operation

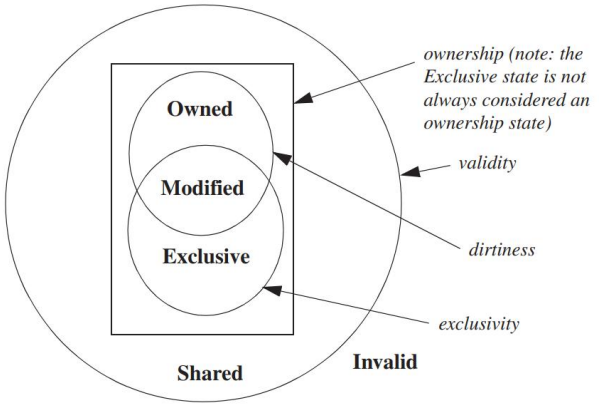
- Cores ***read/write different*** cache blocks ***asynchronously and in parallel***
- Cache block can be distributed for ***simultaneous reading***
- ***Writing*** possible only ***after exclusive ownership request granted***
- ***Update history for every individual cache block is seen the same for all cores***



Coherent caching: attempt to combine advantages of both:

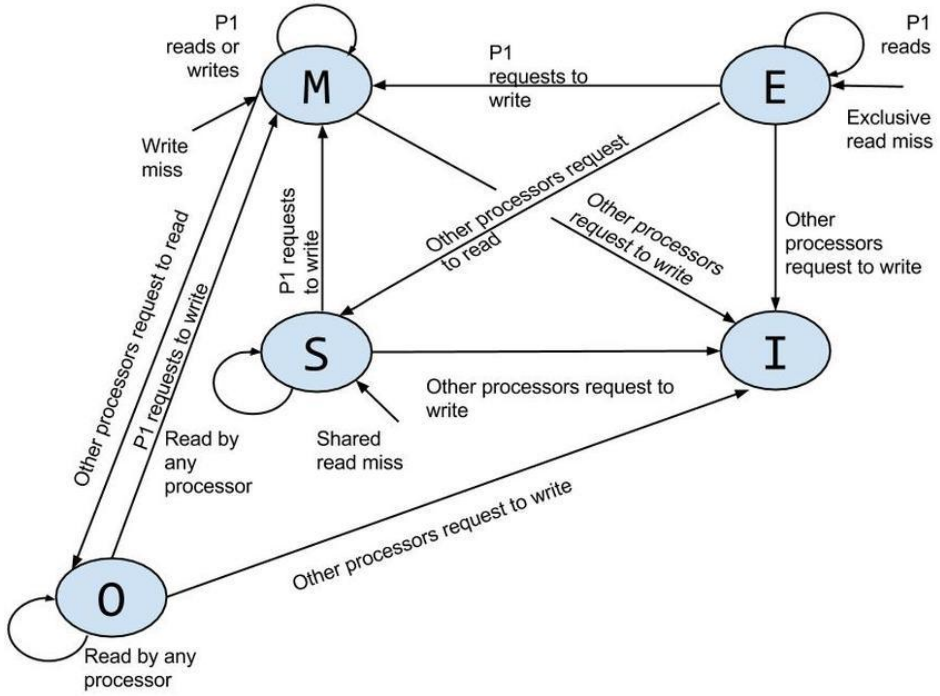
- **shared memory model (convenient programming)**
- **… and message passing (reduces unnecessary transfers)**

MOESI protocol (Sweazey/Smith, 1986)



Two states are differentiated by ownership:

- O (Owned) – cache line is potentially dirty, memory is potentially stale, this controller manages this line, others may have it in S state in AMD processors core can write data and broadcast it to other processors (write-update protocol)
- S (Shared) – line is read-only, either other core is owner or main memory holds actual data



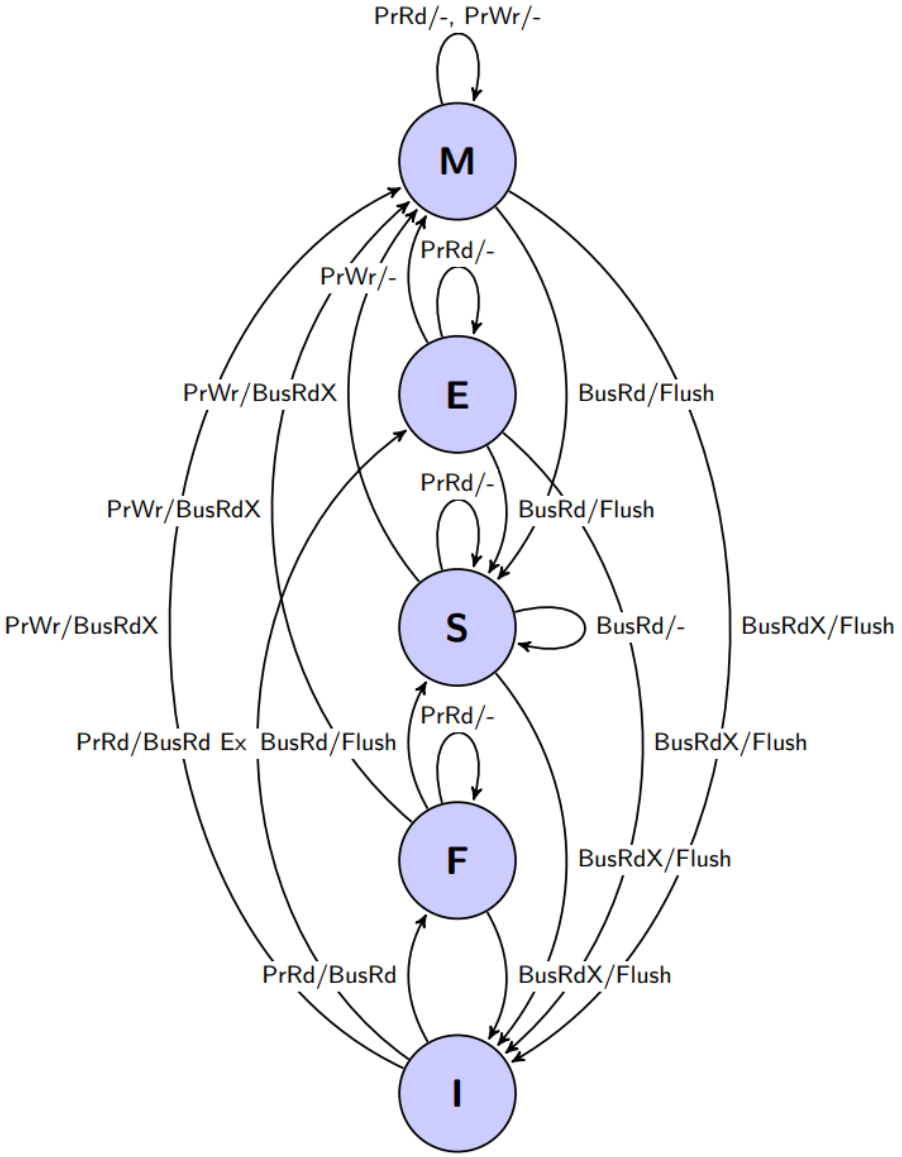
P. Sweazey and A. J. Smith. A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus. In Proceedings of the 13th Annual International Symposium on Computer Architecture, pp. 414–423, June 1986.

MESIF protocol (Intel, 2001)

Designed for ccNUMA systems, forerunner of QPI

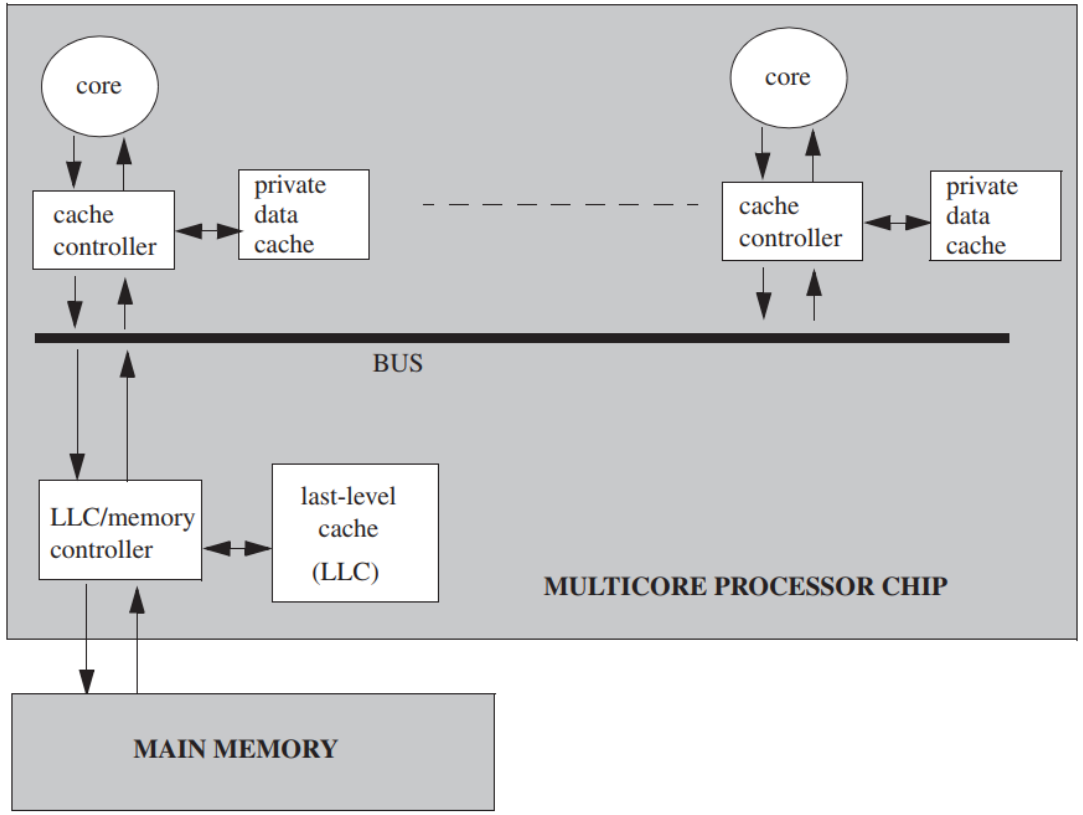
MESI S state is split by the latest access:

- F (Forward) – similar to S, but this core is responsible for answering requests. New read line goes to F state.
- S (Shared) – similar to MESI S state, but this core doesn't answer requests. F line requested by other core goes to S state.



Cache controllers communication: snooping-based

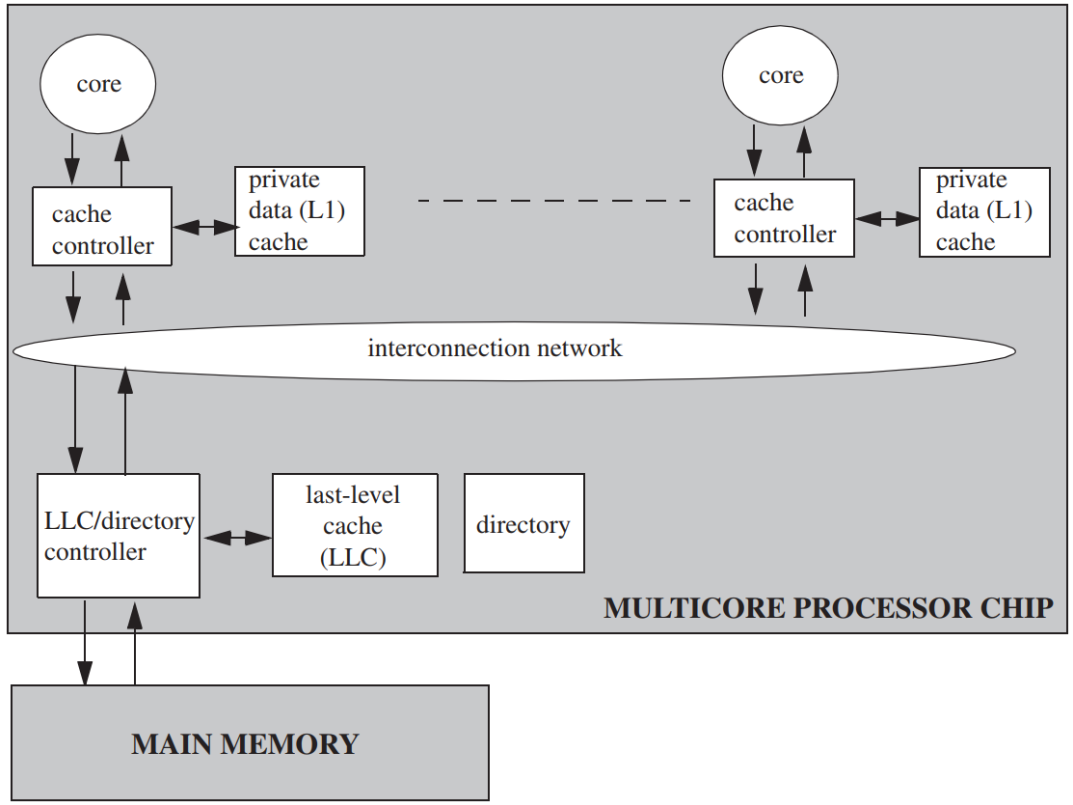
- Cache management is ***distributed***: each controller manages its cache
- Coherence traffic (*snoop requests*) is ***broadcasted***
- Ordering point: bus
- ***Low latency***, but ***doesn't scale*** (broadcasts)



ITMO

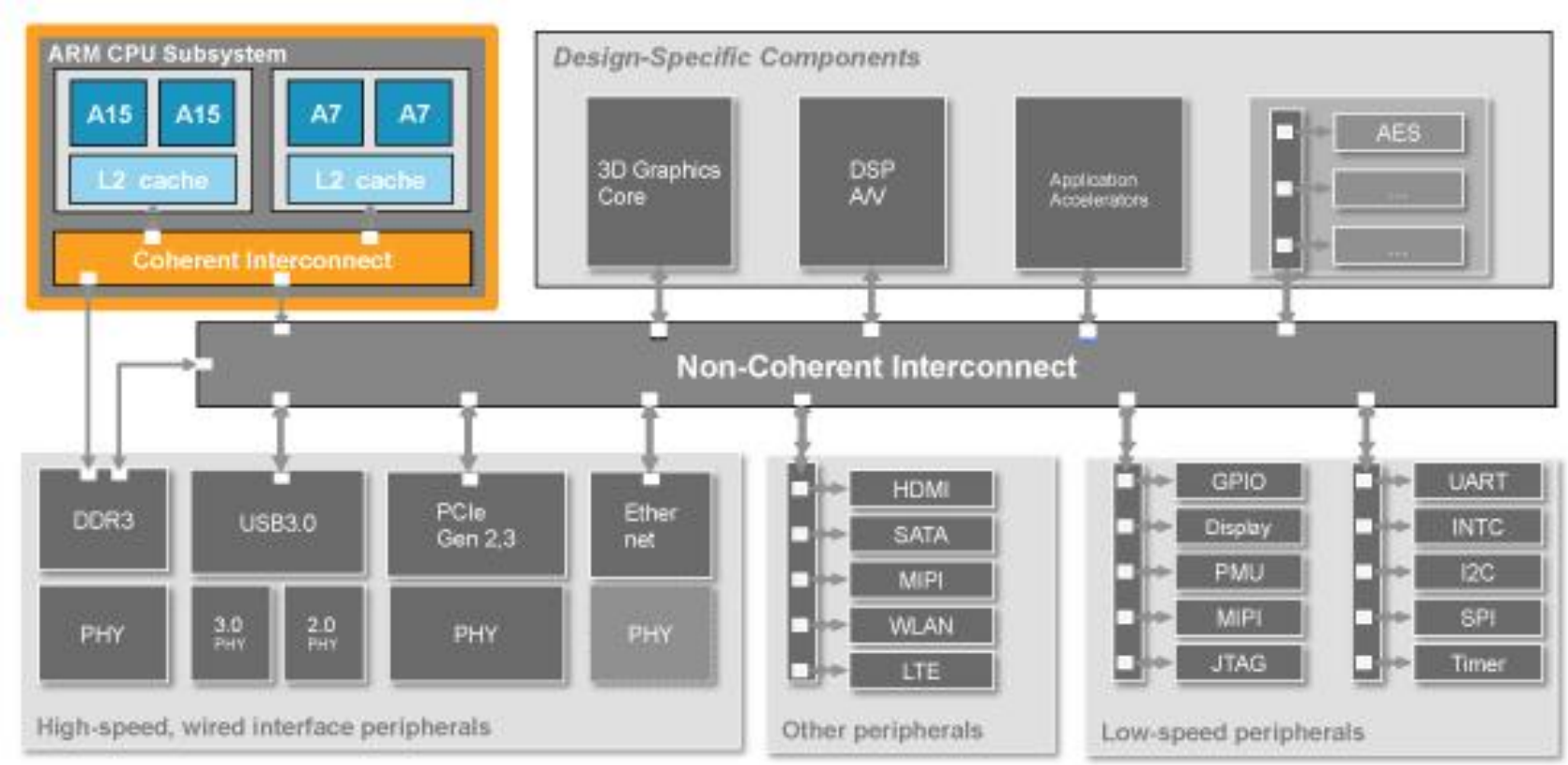
Cache controllers communication: directory-based

- Cache management is ***centralized***: directory manages cache states
- Coherence traffic (*snoop requests*) ***is not broadcasted***
- Ordering point: directory
- ***Bigger latency***, but ***scales better*** (no broadcasts)



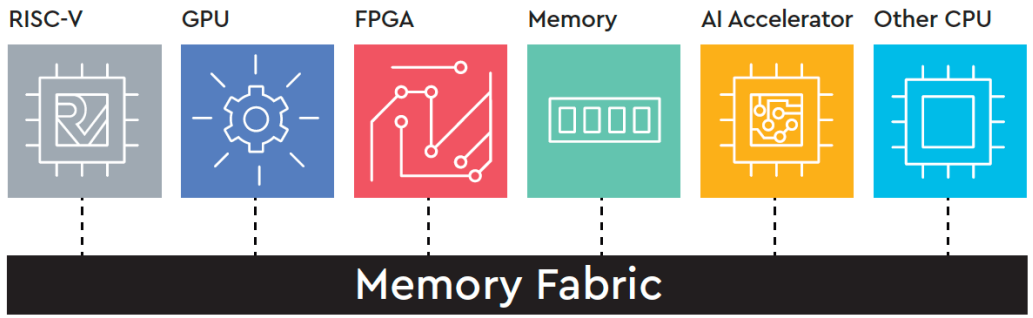
Intermediate solutions (mixture of snooping and directories) can be considered

Example: cache coherent cluster inside multicore processor

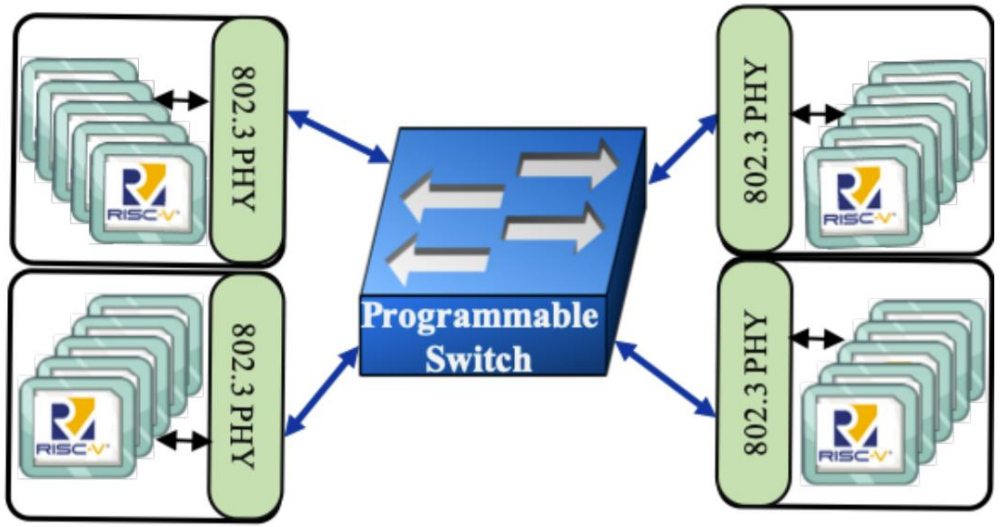


https://www.chipestimate.com/Multicore-ARM-SoCs-Face-Cache-Coherency-Dilemma/Cadence/Technical-Article/2012/10/02

Example: cache coherency over Ethernet



Data is the center of the architecture
No established hierarchy – CPU doesn't 'own' the GPU or the Memory
Preserved Cache Coherency over the Network



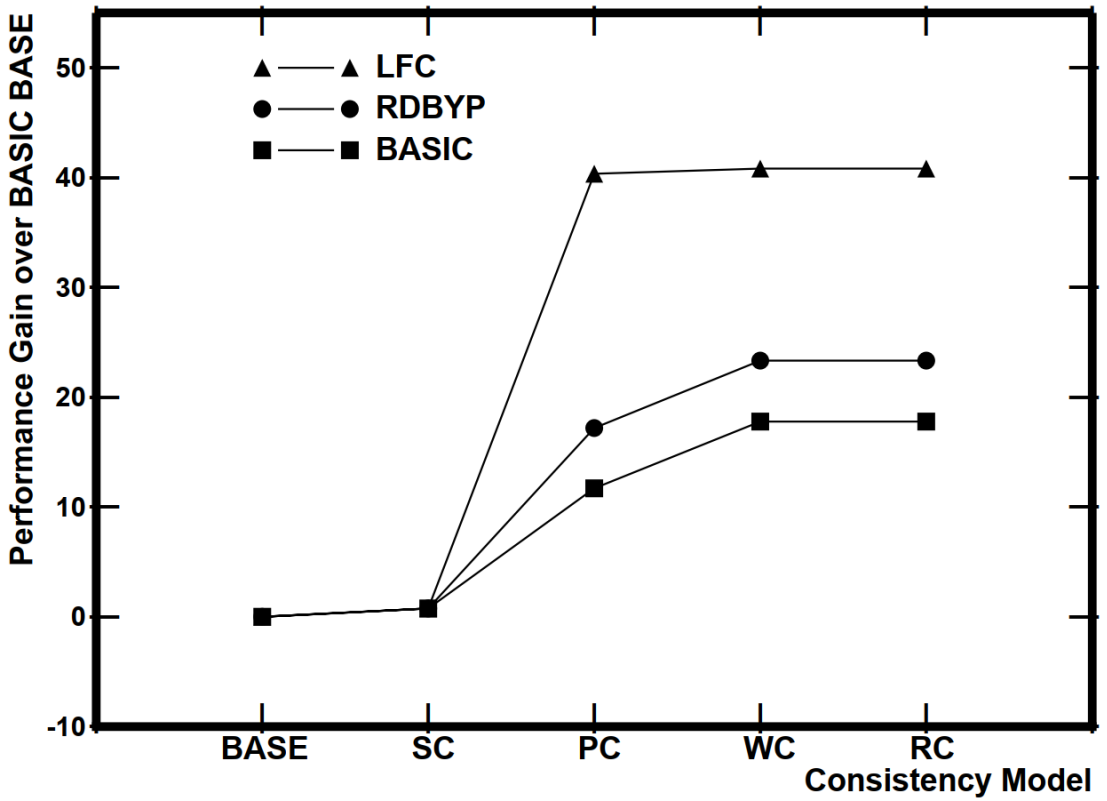
Marjan Radi et al. OmniXtend: Direct to Caches Over Commodity Fabric. 2019 IEEE Symposium on High-Performance Interconnects (HOTI)

ITMO

Memory consistency models

Models specifying possible read and write reorderings:

- Base
  Without buffering and pipelining
- SC (Sequential Consistency)
  store buffers added,
  transactions don't reorder
- PC (Processor Consistency)/TSO
  reads can bypass writes
- WC (Weak Consistency)
  writes reordering
- RC (Release Consistency)
  the weakest, all transactions can buffer, pipeline and reorder



Read prioritization over writes gives significant performance boost (RAW dependencies less block computations)!

K. Gharachorloo, A. Gupta, J. Hennessy, Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors, ACM SIGARCH Computer Architecture News 19(2), 1997

RISC-V memory consistency models

- **RVWMO** (**R**ISC-V **W**eak **M**emory **O**rdering) – default memory model, relaxed

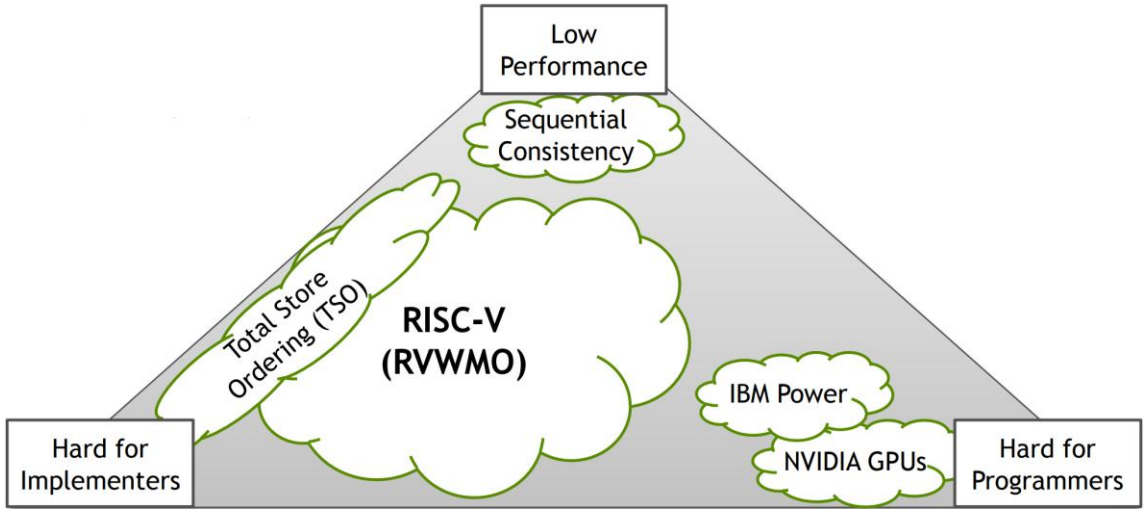
  Memory accesses can be reordered freely, memory barrier instructions are used for explicit sync

- **RVTSO** (**R**ISC-V **T**otal **S**tore **O**rdering) – additional optional memory model, stronger

  Only store -> load reordering can be observed

  Defined as "Ztso" standard ISA extension

  Positioned mostly for running legacy code (for x86, SPARC, etc.)



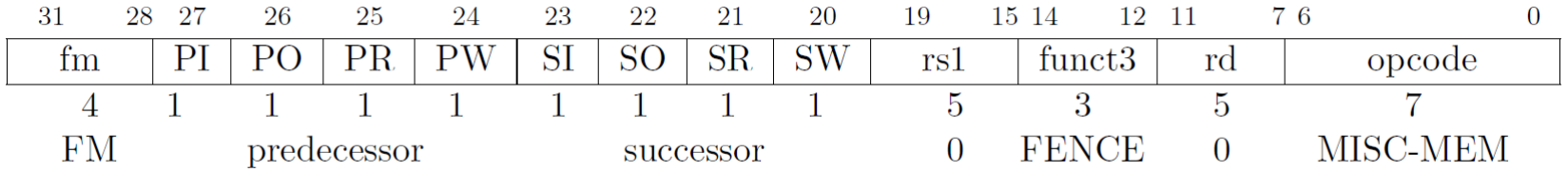
Picture: Dan Lustig. RISC-V Memory Consistency Model Tutorial. May 7, 2018

RISC-V: memory barriers

FENCE – memory barrier: instruction is used to order device I/O and memory accesses as viewed by other RISC-V harts and external devices or coprocessors.

Any combination of predecessor and successor operations can be explicitly ordered

- **Predecessor set** – operations preceding FENCE
  - **PR/PW** – preceding memory reads and writes cannot happen after FENCE
  - **PO/PI** - preceding I/O reads and writes cannot happen after FENCE
- **Successor set** – operations after FENCE
  - **SR/SW** – memory reads and writes after FENCE cannot precede FENCE
  - **SO/SI** – I/O reads and writes after FENCE cannot precede FENCE

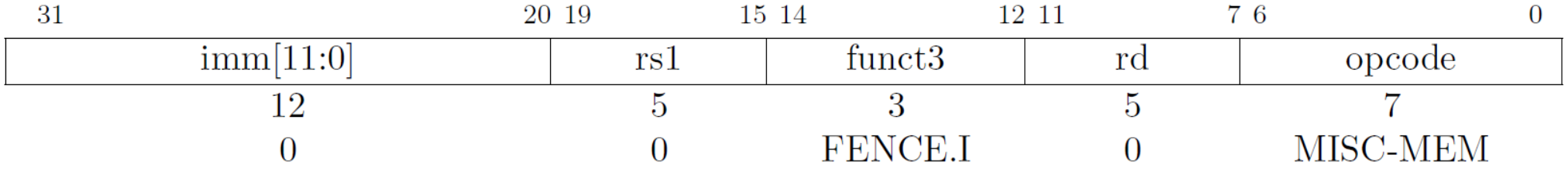


RISC-V: instruction memory barrier

FENCE.I – instruction for explicit synchronization between writes to instruction memory and instruction fetches on the same hart (for self-modifying code).

Standard mechanism to ensure that stores visible to a hart will also be visible to its instruction fetches

instruction fetch postponed until FENCE.I and/or "incorrect" (outdated) instructions discarded

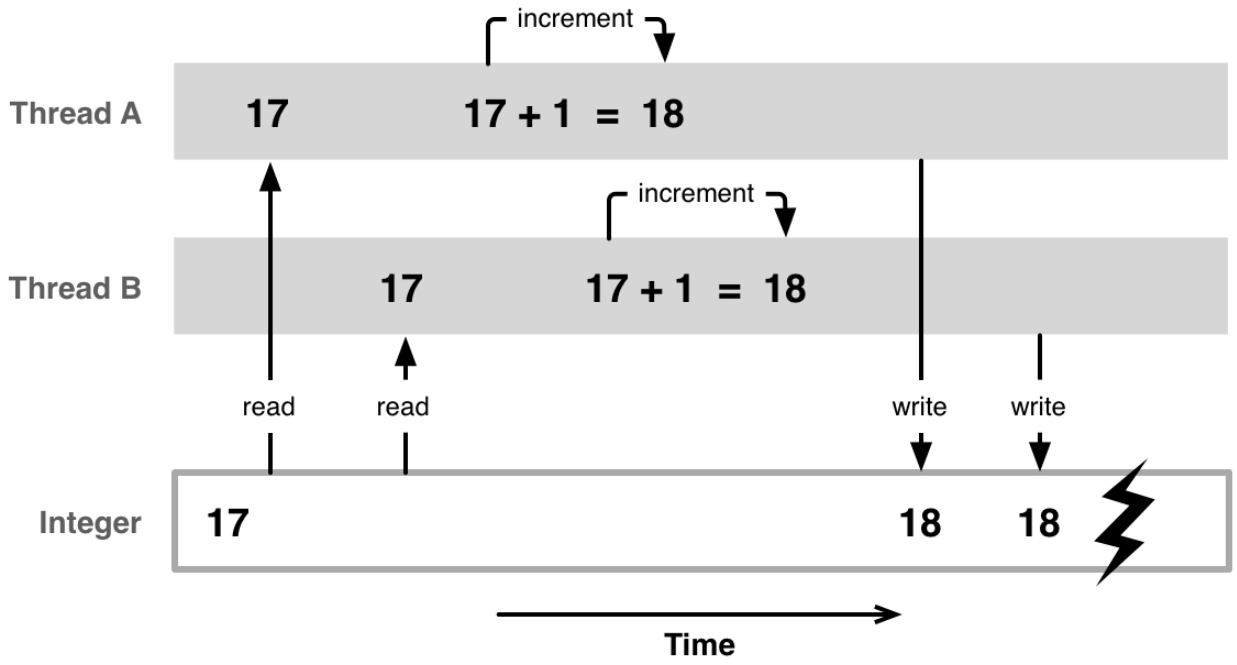


ITMO

Interprocess synchronization: race conditions

Problem: two threads (or cores) do counter increment. Both read value, then write updated values (one increment instead on two)

Root cause: divided (***non-atomic***) read-update-write cycles



Solution: implement indivisible (atomic) operations in platform

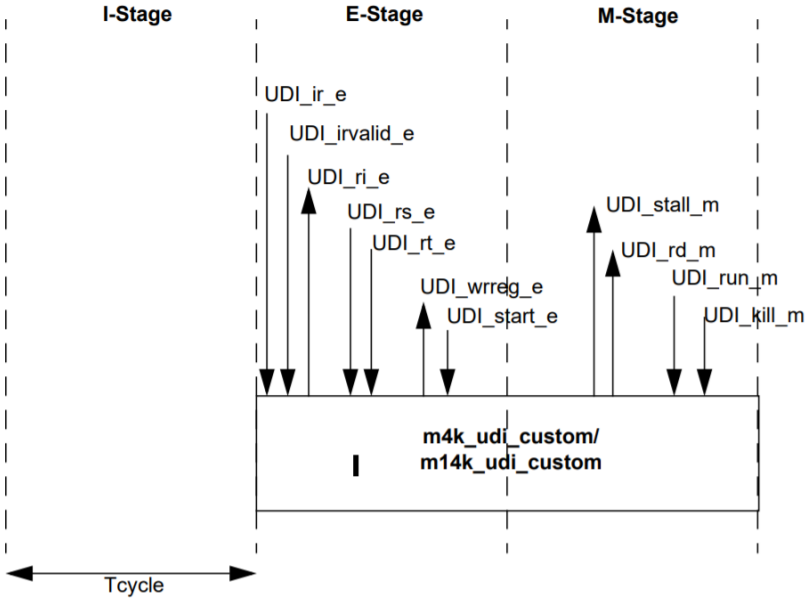
ITMO

RISC-V: atomics

- **LR (load-reserved)** – loads data word to register and registers ***reservation set*** (bytes comprising the word, released on any writing)
- **SC (store-conditional)** – stores bytes if reservation set valid, writes status to register

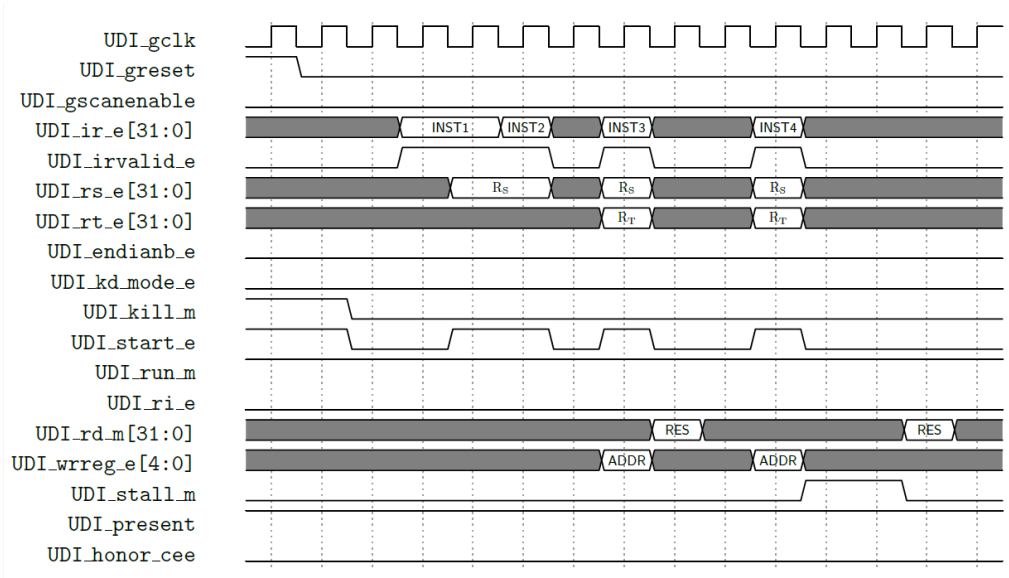
  *LR/SC pairs are useful for **lock-free data structures** programming*
- **AMOx** – atomic read, modification, write
  - **AMOSWAP** – swap between sources
  - **AMOADD** – addition
  - **AMOAND** – bitwise and
  - **AMOOR** – bitwise or
  - **AMOXOR** – bitwise xor
  - **AMOMAX** – return maximum
  - **AMOMIN** – return minimum

Processor ISA extension interfaces



These coprocessors are ***synchronized with CPU pipeline*** and have direct access to ***CPU register file***

Some processor IP cores support integration of ***custom tightly integrated coprocessors***



Pictures: CorExtend® Instruction Integrator's Guide for M4K®/4KE®/4KS™ and M14K™ Family Cores, MIPS Technologies, Inc., 2011

Thank you for the lesson!

Alexander Antonov, Assoc. Prof., antonov@itmo.ru

Hangzhou, 2025