CSCI-GA.3033-012

Multicore Processors: Architecture & Programming

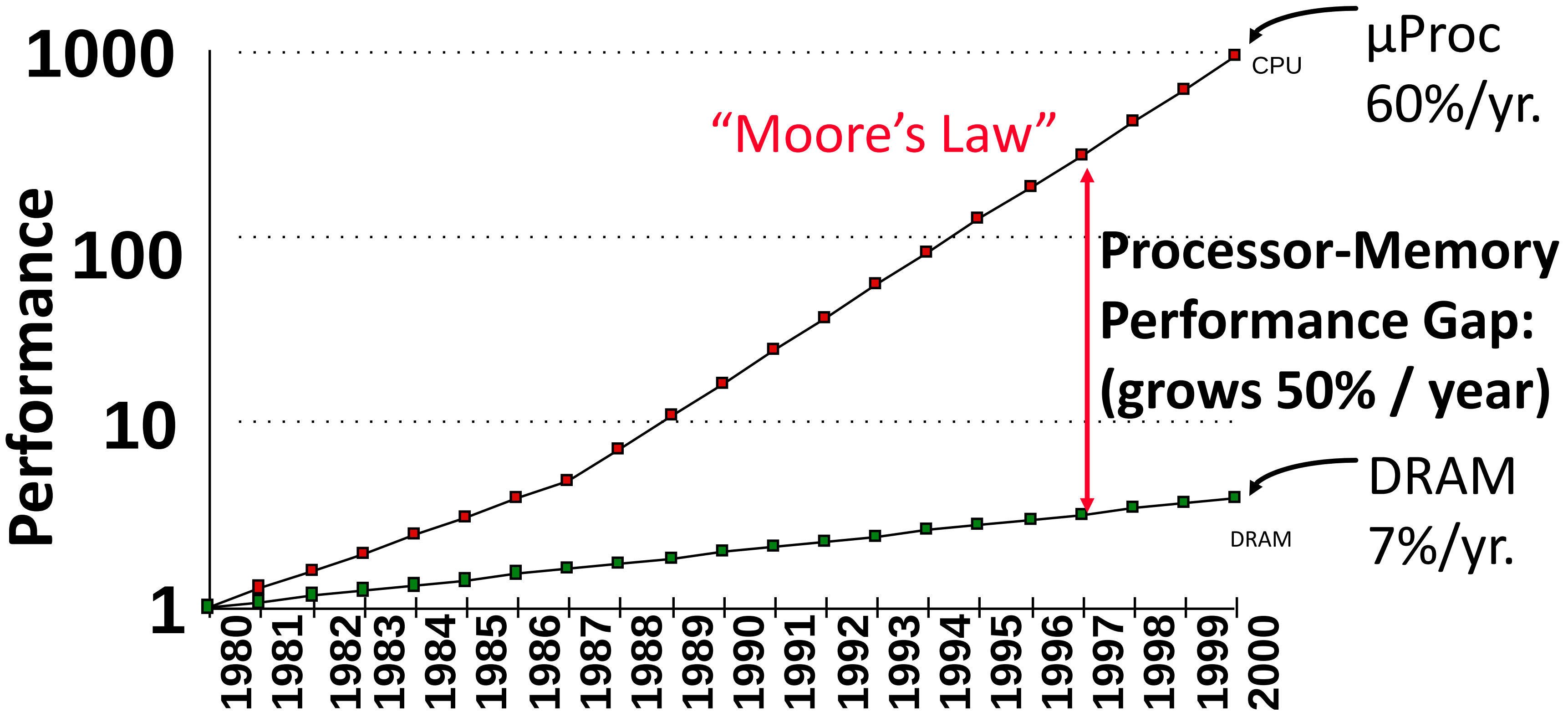
Lecture 3: The Memory System … You Can't Ignore it!

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

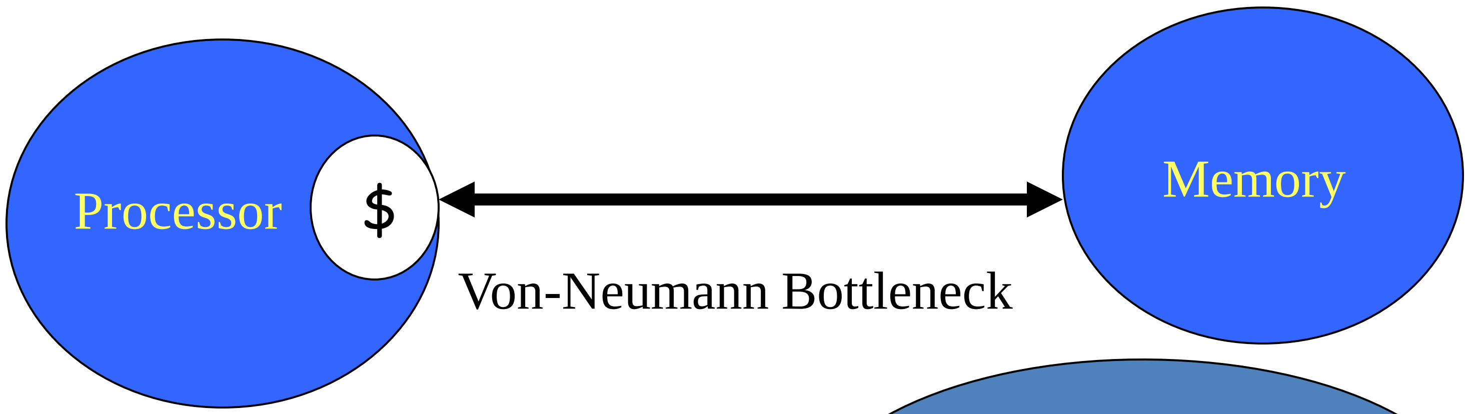
Computer Technology

- Memory
  - DRAM capacity: 2x / 2 years (since '96); 64x size improvement in last decade.
- Processor
  - Speed 2x / 1.5 years (since '85); 100X performance in last decade.
- Disk
  - Capacity: 2x / 1 year (since '97) 250X size in last decade.

Memory Wall



Most of the single core performance loss is on the memory system!


Processor
$
Memory
Von-Neumann Bottleneck

I am really sorry. Didn't know that it would be that serious.

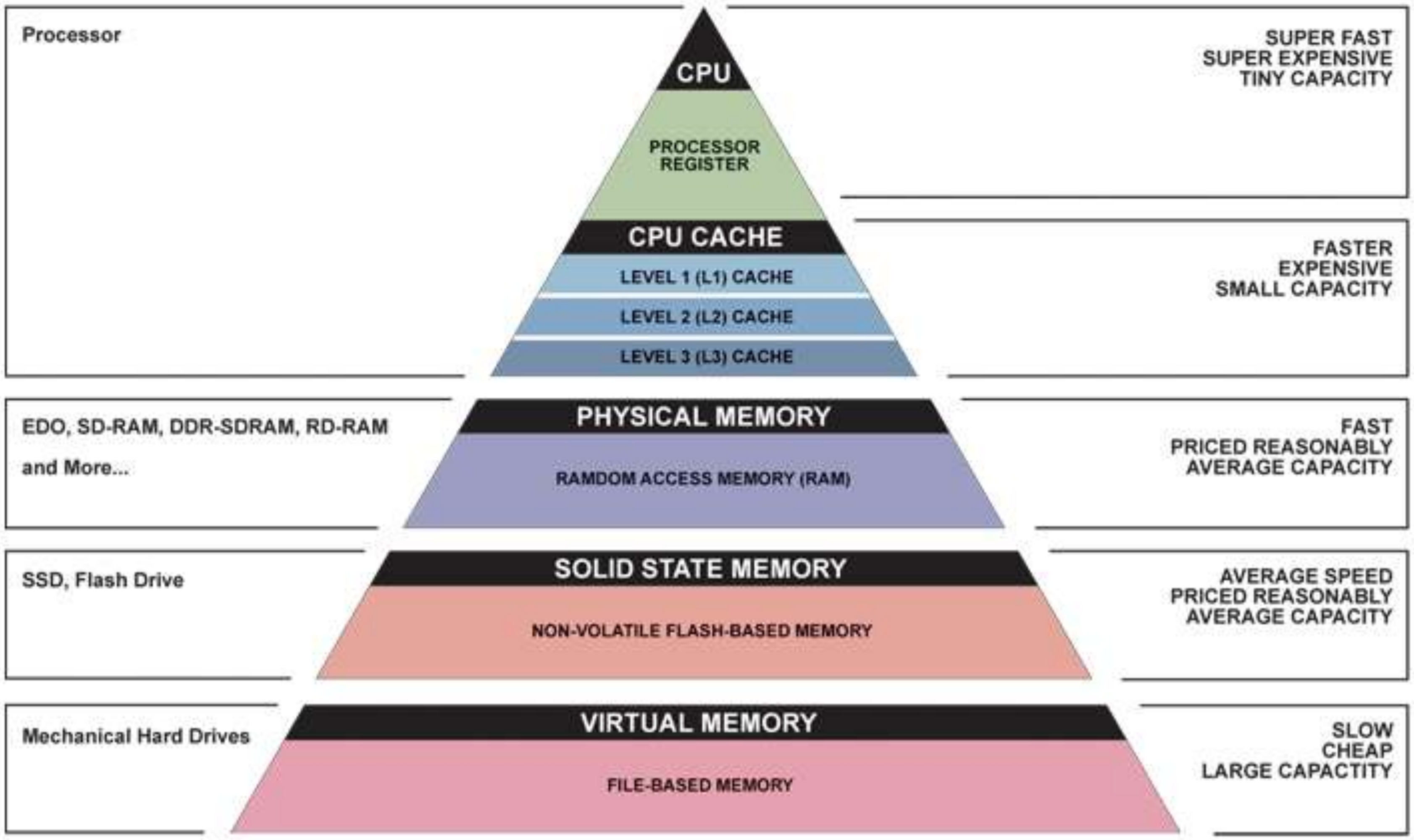
Two Main Program Characteristics

- **Temporal locality**
  - I used X
  - Most probably I will use it again soon
- **Spatial locality**
  - I used item number M
  - Most probably I will need item M+1 soon

Cache Analogy

- Hungry! must eat!
  - Option 1: go to refrigerator
    - Found → eat!
    - Latency = 1 minute
  - Option 2: go to store
    - Found → purchase, take home, eat!
    - Latency = 20-30 minutes
  - Option 3: grow food!
    - Plant, wait … wait … wait … , harvest, eat!
    - Latency = ~250,000 minutes (~ 6 months)

Storage Hierarchy Technology



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Why Memory Wall?

- RAMs not optimized for speed but for density (till now at least!)
- Off-chip bandwidth
- Increasing number of on-chip cores
  - Need to be fed with instructions and data
  - Big pressure on buses, memory ports, …

Cache Memory: Yesterday

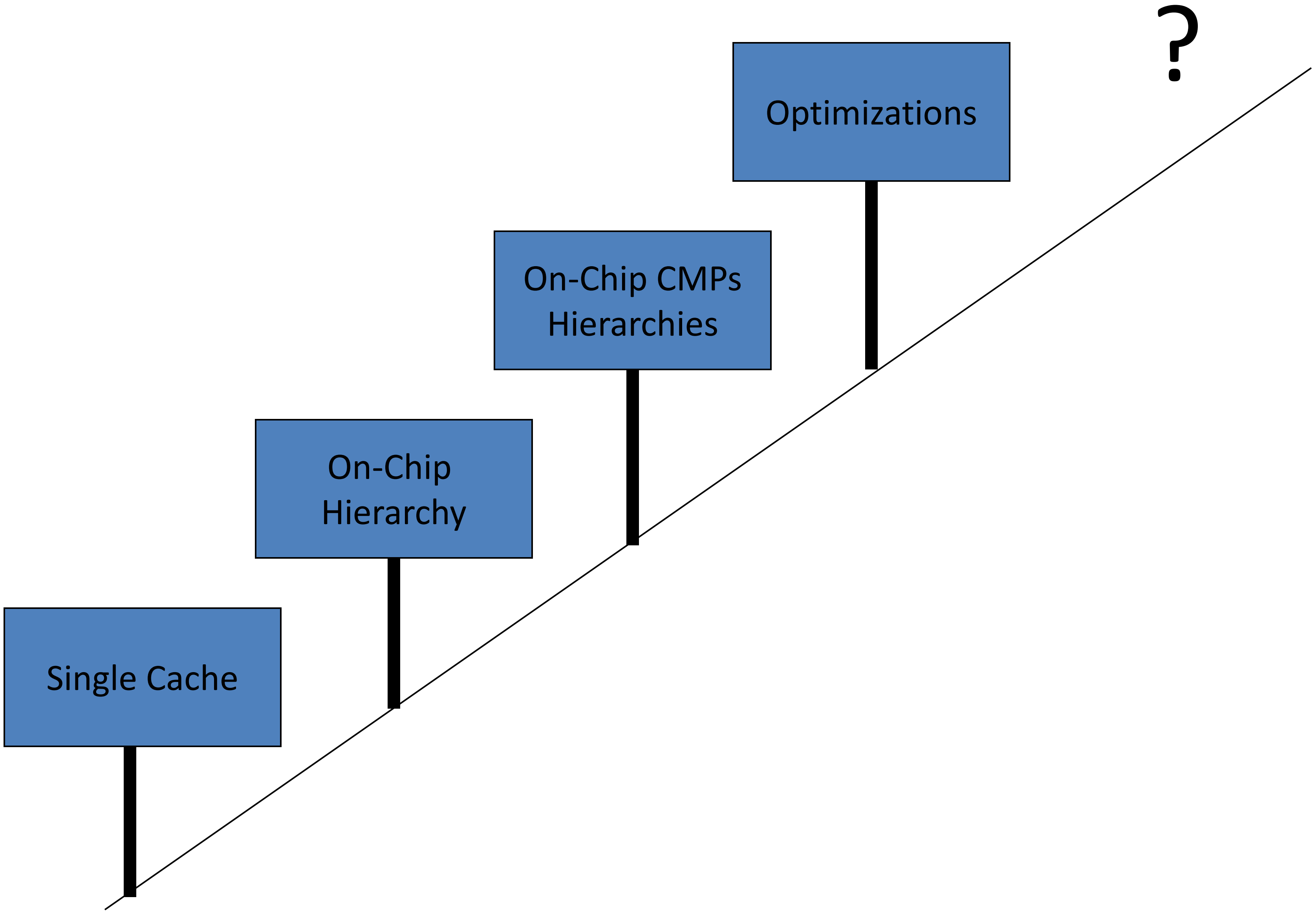
- Processor-Memory gap not very wide
- Simple cache (one or two levels)
- Inclusive
- Small size and associativity

Cache Memory: Today

- Wider Processor-Memory gap
- Two or three levels of cache hierarchy
- Larger size and associativity
- Inclusion property revisited
- Coherency
- Many optimizations
  - Dealing with static power
  - Dealing with soft-errors
  - Prefetching
  - …

Cache Memory: Tomorrow

- Very wide processor-memory gap
- Multiple cache hierarchies (multi-core)
- On/Off chip bandwidths become bottleneck
- Scalability problem
- Technological constraints
  - Power
  - Variability
  - …


Single Cache
On-Chip Hierarchy
On-Chip CMPs Hierarchies
Optimizations
?

100s On-Chip Cores

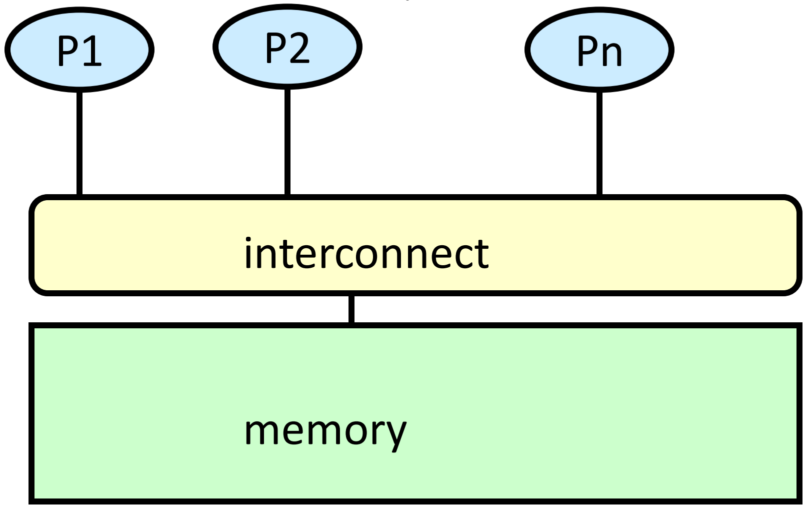
- Will soon be technologically possible
- Near-future usage:
  - Massively parallel applications
    - Multithreading
- In the long run
  - Day to day use
    - Hybrid multithreading + multiprogramming

From Single Core to Multicore

- Currently mostly shared memory
  - This can change in the future
  - The “sharing” can be logical only (i.e. distributed shared memory)
- A new set of complications, in addition to what we already have ☹
  - Coherence
  - Consistency

Shared Memory Mutlicore

- Uniform
  - Uniform Cache Access
  - Uniform Memory Access
- Non-Uniform
  - Non-Uniform Cache Access
  - Non-Uniform Memory Access



Memory Model

- **Intuitive:** The reading and address returns the most recent write to that address.
- This is what we find in uniprocessors
- For multicore, we call this: sequential consistency
  - Much harder and tricky to achieve
  - This is why we need coherence

Sequential Consistency Model

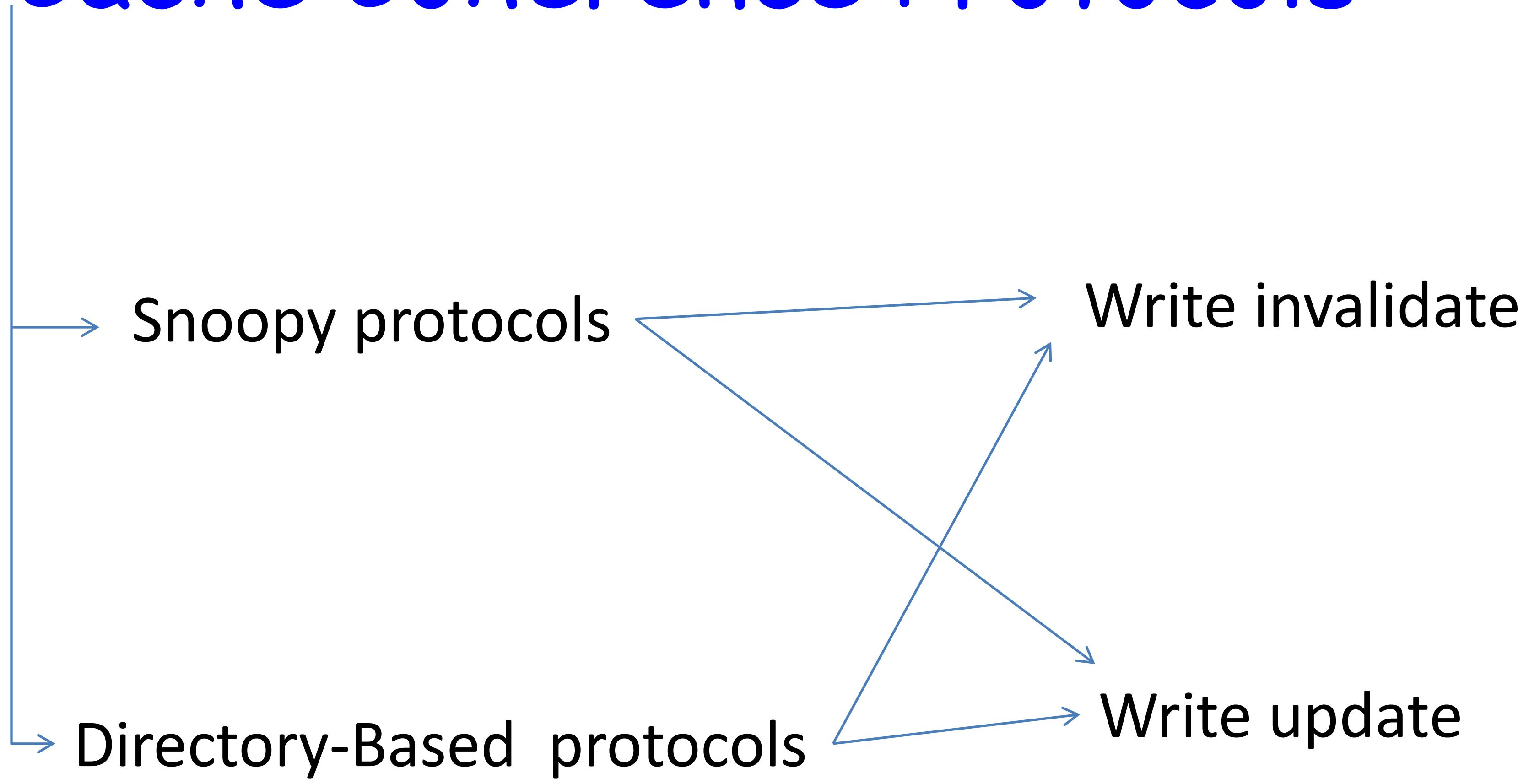
- Example:
  - P1 writes data=1, then writes flag=1
  - P2 waits until flag=1, then reads data

If P2 reads flag	Then P2 may read data
0	1
0	0
1	1

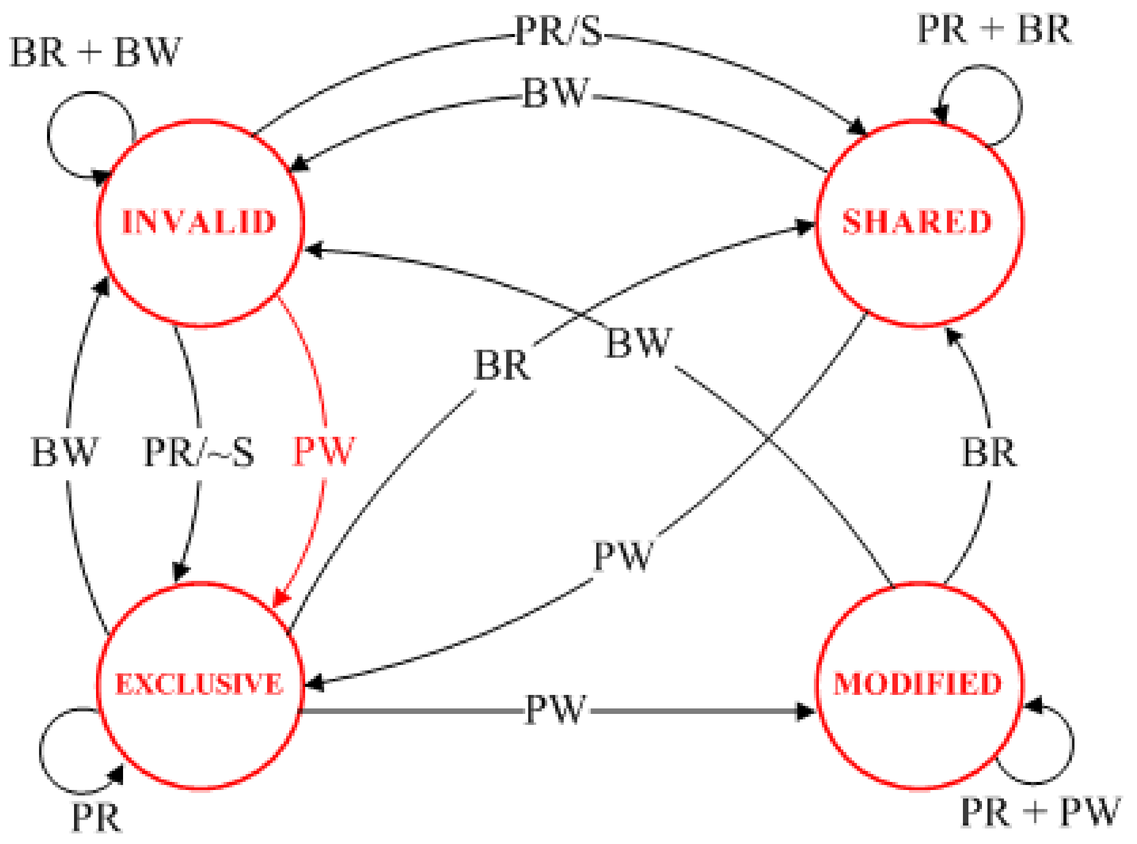
Ensuring Consistency: Coherence Protocol

- Cache coherence needed in multicore processors to ensure consistency
- A memory system is coherent if:
  - P writes to X; no other processor writes to X; P reads X and receives the value previously written by P
  - P1 writes to X; no other processor writes to X; sufficient time elapses; P2 reads X and receives value written by P1
  - Two writes to the same location by two processors are seen in the same order by all processors – write serialization
  - The memory consistency model defines "time elapsed" before the effect of a processor is seen by others

Cache Coherence Protocols



Example: MESI Protocol



PR = processor read
PW = processor write
S/~S = shared/NOT shared

BR = observed bus read
BW = observed bus write

The Future In Technology

- **Traditional**
  - SRAM
  - DRAM
  - Hard drives
- **New**
  - eDRAM
  - Flash
  - Solid-State Drive
- **Even Newer**

  (disruptive technology?)
  - M-RAM
  - STT-RAM
  - PCM
  - - ...

+

- 3D Stacking
- Photonic interconnection

As A Programmer

- A parallel programmer is also a performance programmer: know your hardware.
- Your program does not execute on a vacuum.
- In theory, compilers understand memory hierarchy and can optimize your program;
  - In practice they don't!!
- Even if compiler optimizes one algorithm, it won't know about a different algorithm that might be a much better match to the processor

As A Programmer

- You don't see the cache
  - But you feel it
- You see the disk and memory
  - So you can explicitly manage them

Required concurrency = Bandwidth * Latency

Cost
(over-simplification though)

Physics

As A Programmer: Tools In Your Box

- Tiling
- Number of threads you spawn at any given time
- Thread granularity
- User thread scheduling
- Locality (both types)
- What is your performance metric?
  - Throughput
  - Latency
  - Bandwidth-delay product
- Best performance for the a specific configuration Vs Scalabiluty

Conclusions

- The memory is a major bottleneck of performance.
- Actual performance of program can be a complicated function of the architecture
  - Slight changes in the architecture or program change the performance significantly