RECAP

B649

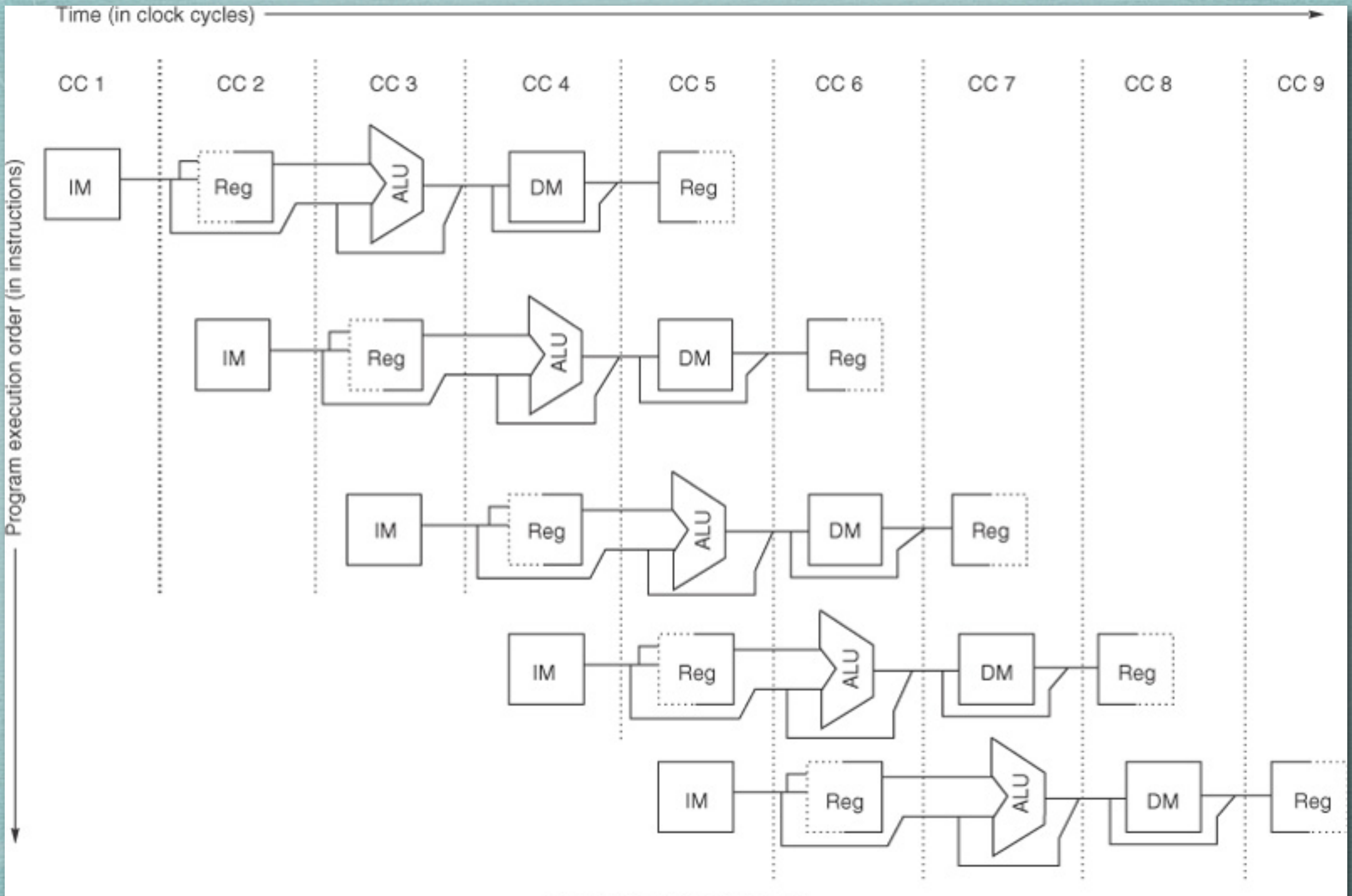
Parallel Architectures and Programming

RECAP

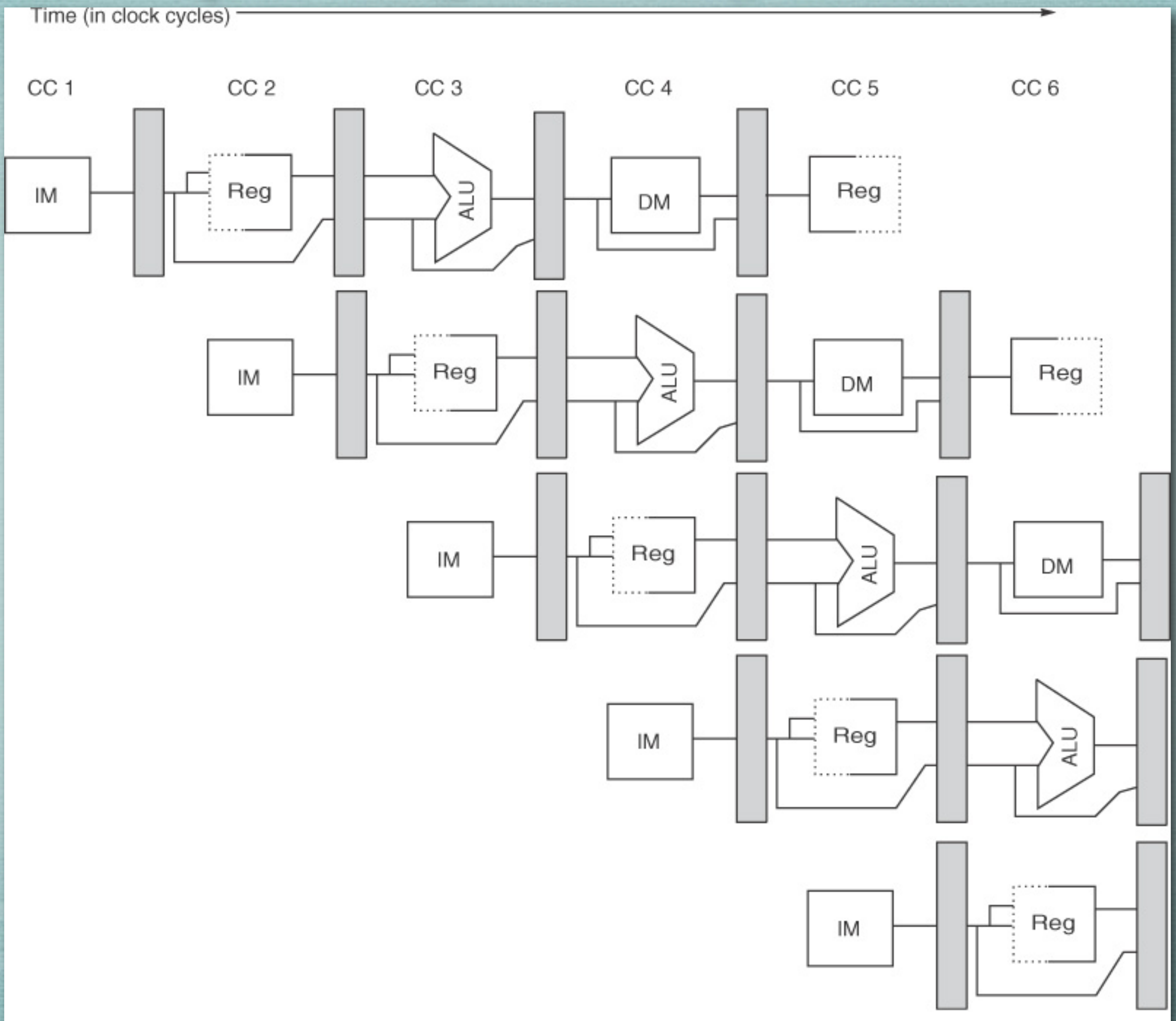
Recap

- ILP
- Exploiting ILP
- Dynamic scheduling
- Thread-level Parallelism
- Memory Hierarchy
- Other topics through student presentations
- Virtual Machines

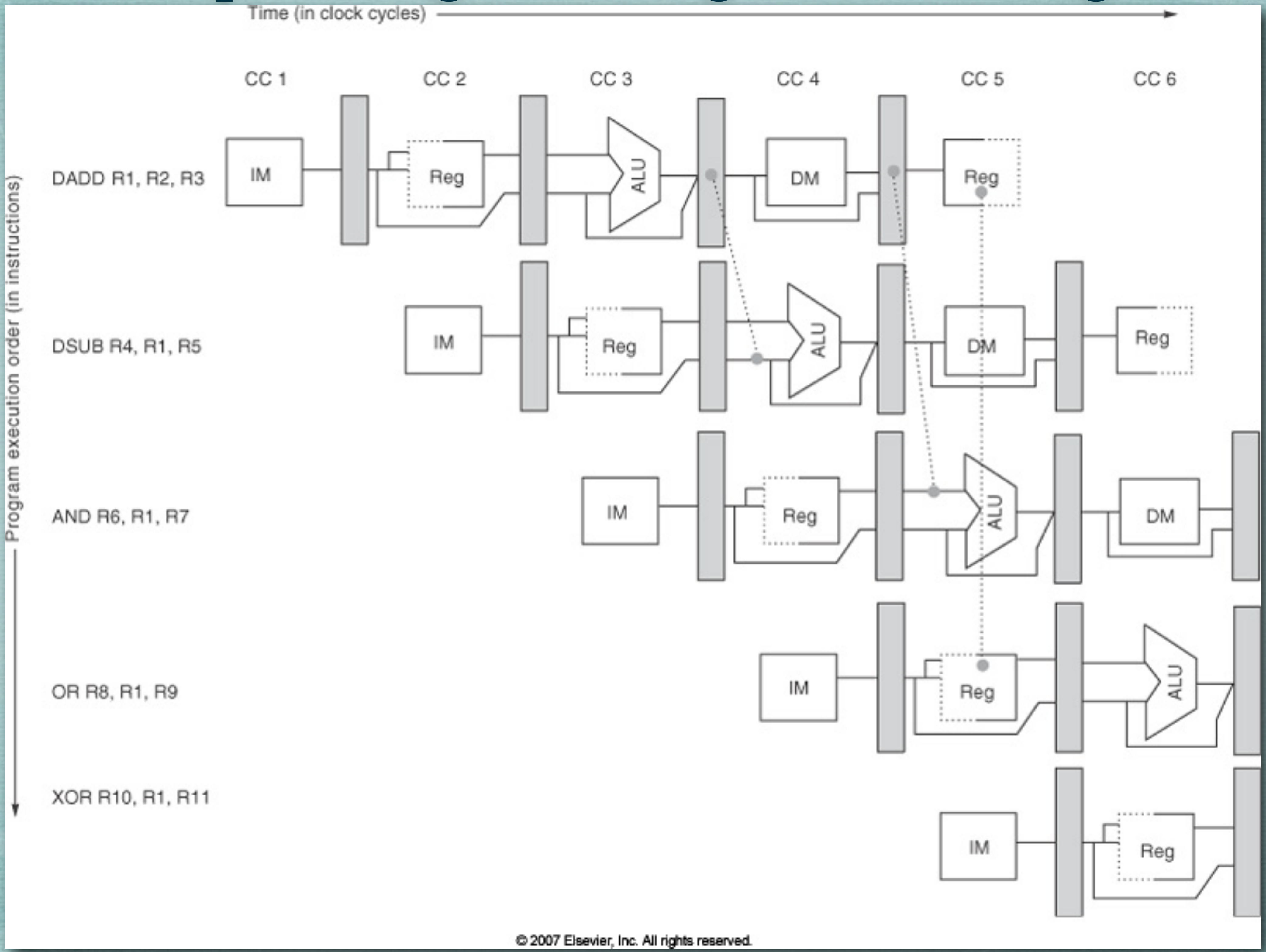
ILP: Pipelining



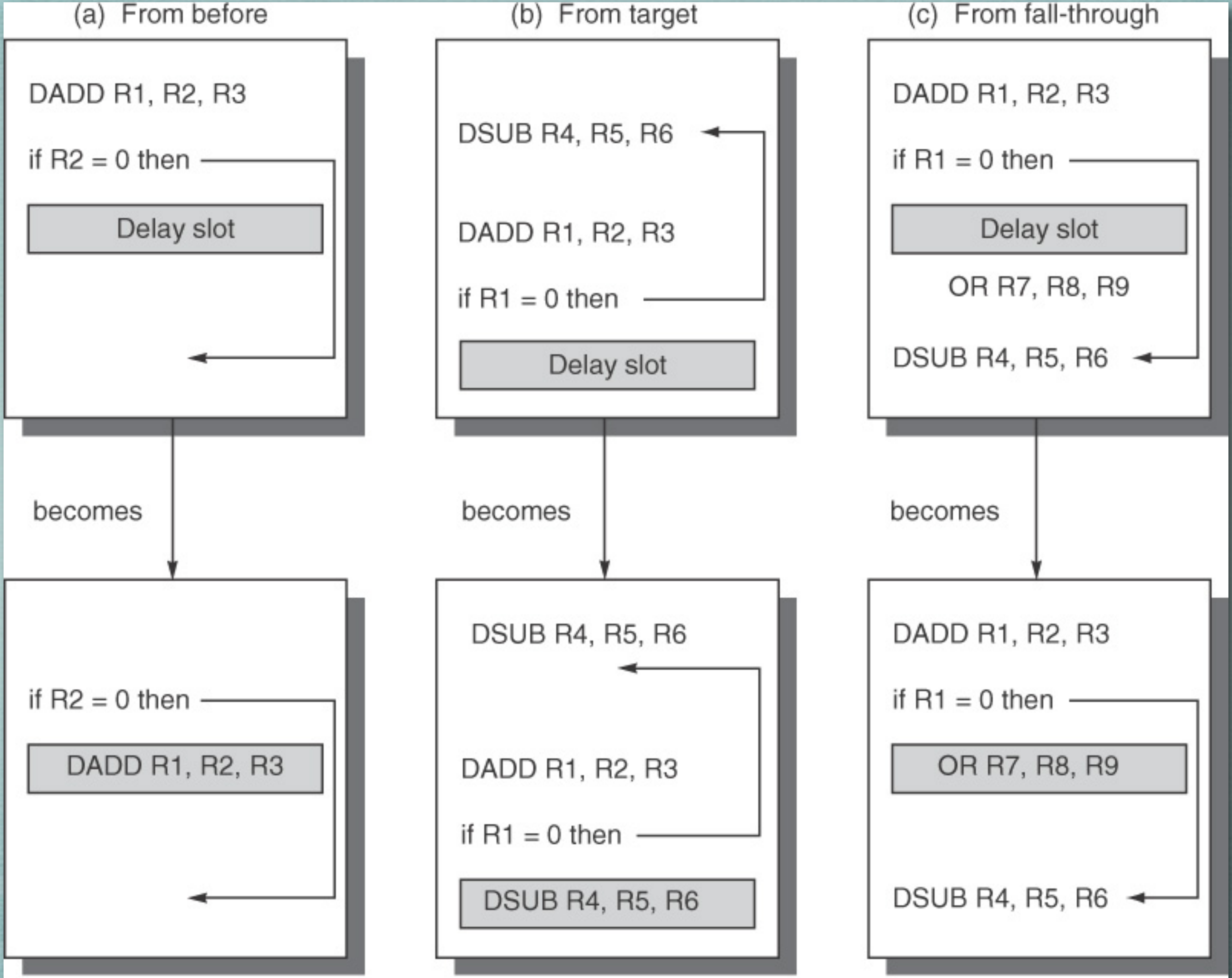
Pipelining: Adding Latches



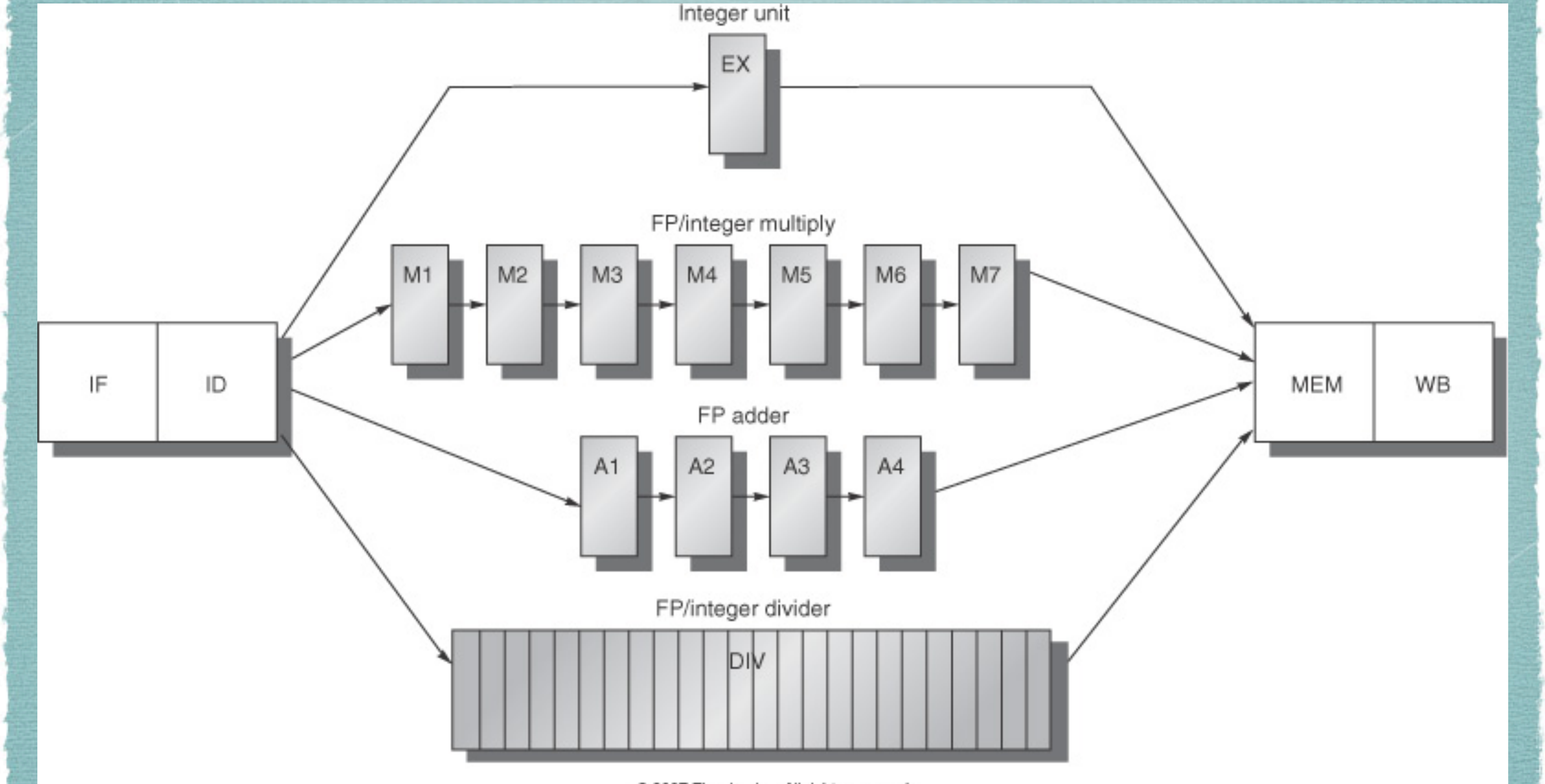
Pipelining: Adding Forwarding



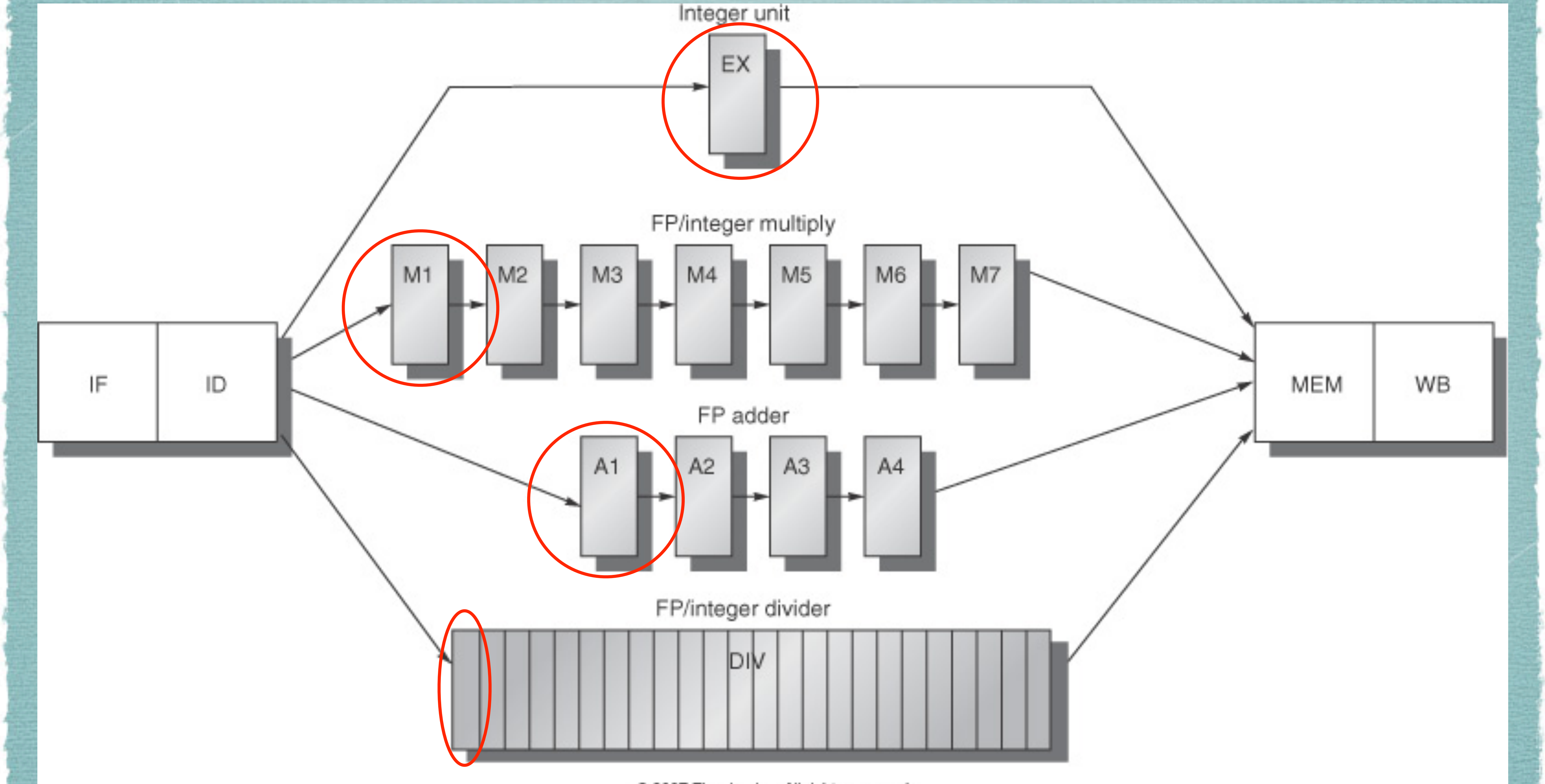
Pipelining: Adding Branch Delay Slots



Extending the Basic Pipeline

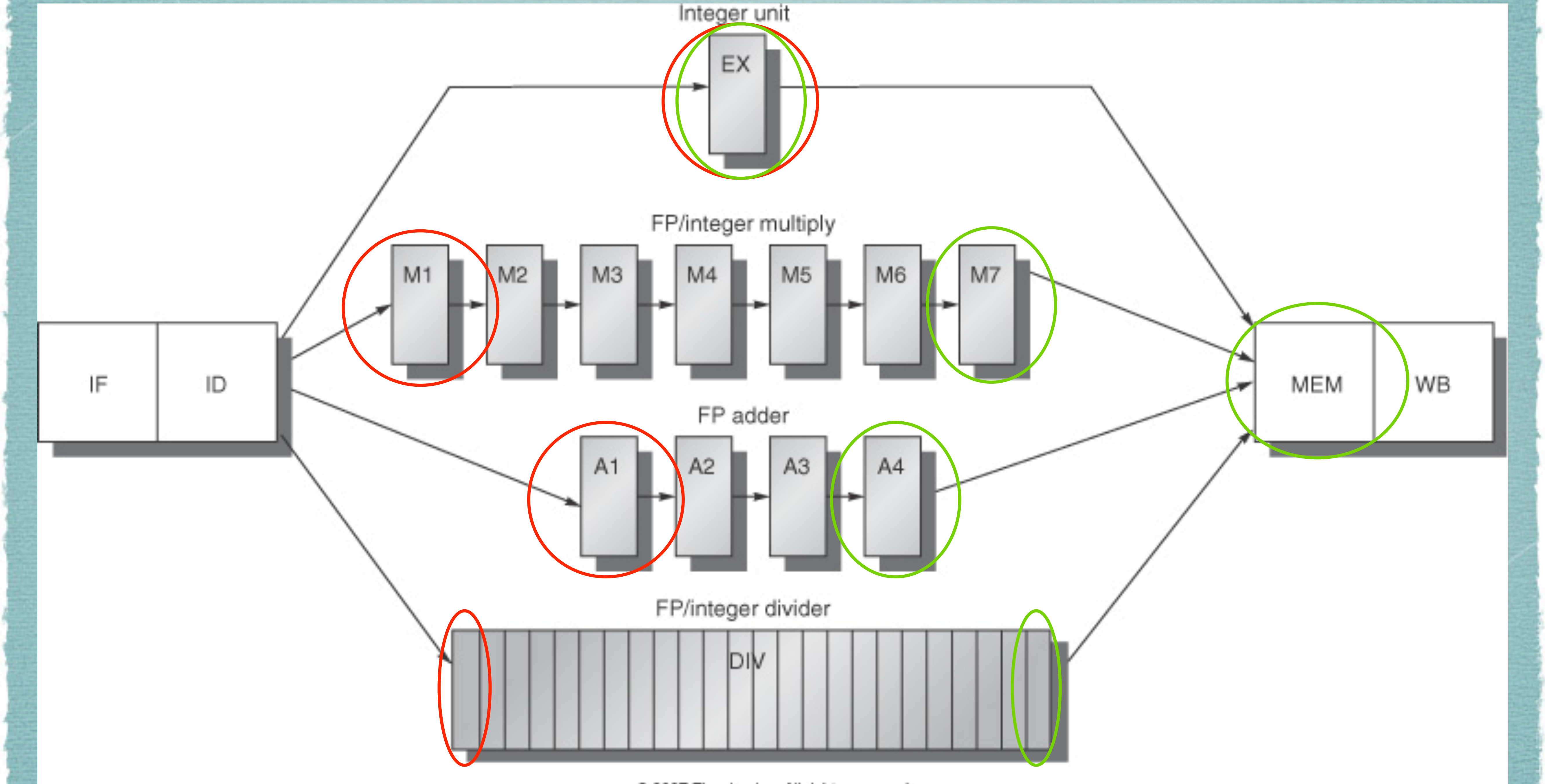


Extending the Basic Pipeline



© 2007 Elsevier, Inc. All rights reserved.

Extending the Basic Pipeline

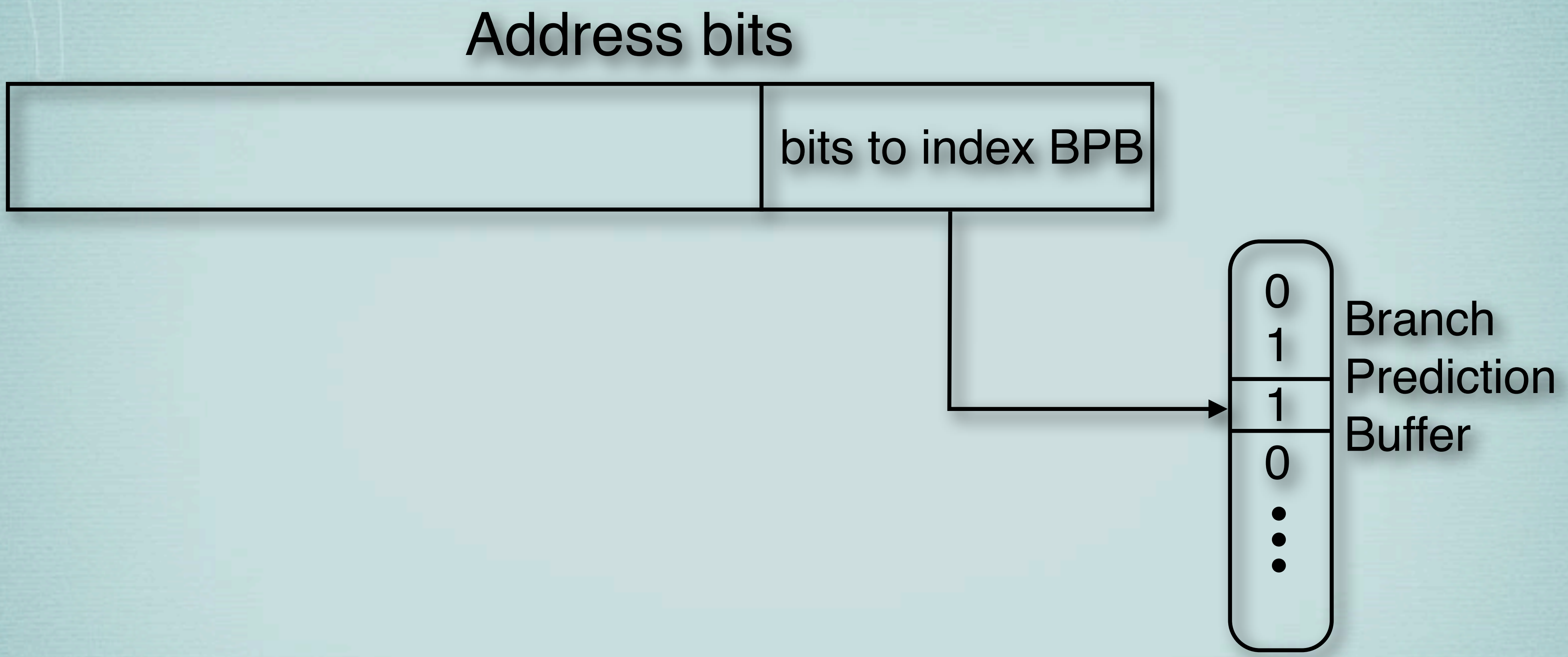


© 2007 Elsevier, Inc. All rights reserved.

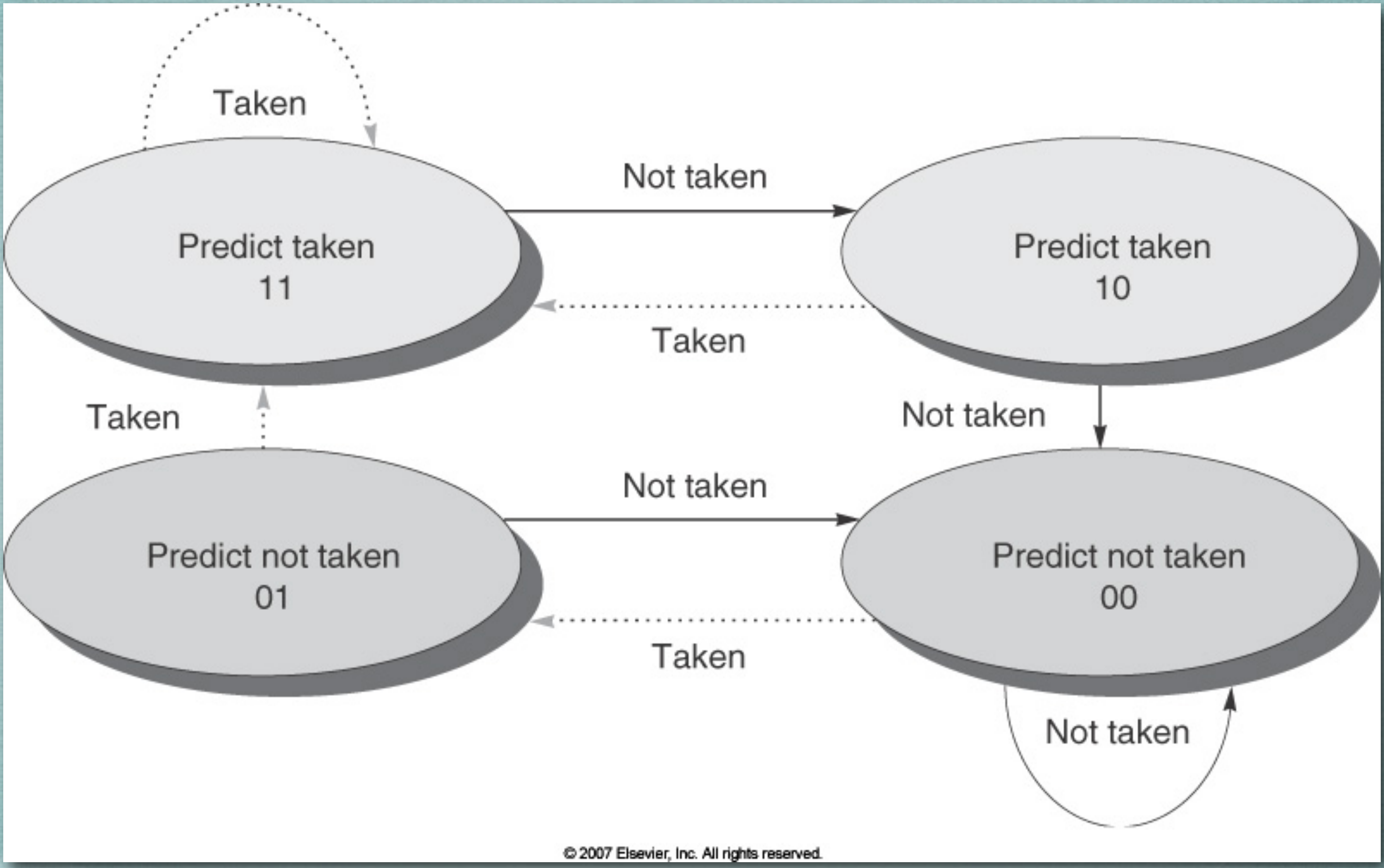
Exploiting ILP Through Compiler Techniques

- Loop unrolling
- Making use of branch delayed slots
- Static branch prediction
- Loop fusion
- Unroll and jam
- ...

Dynamic Branch Prediction

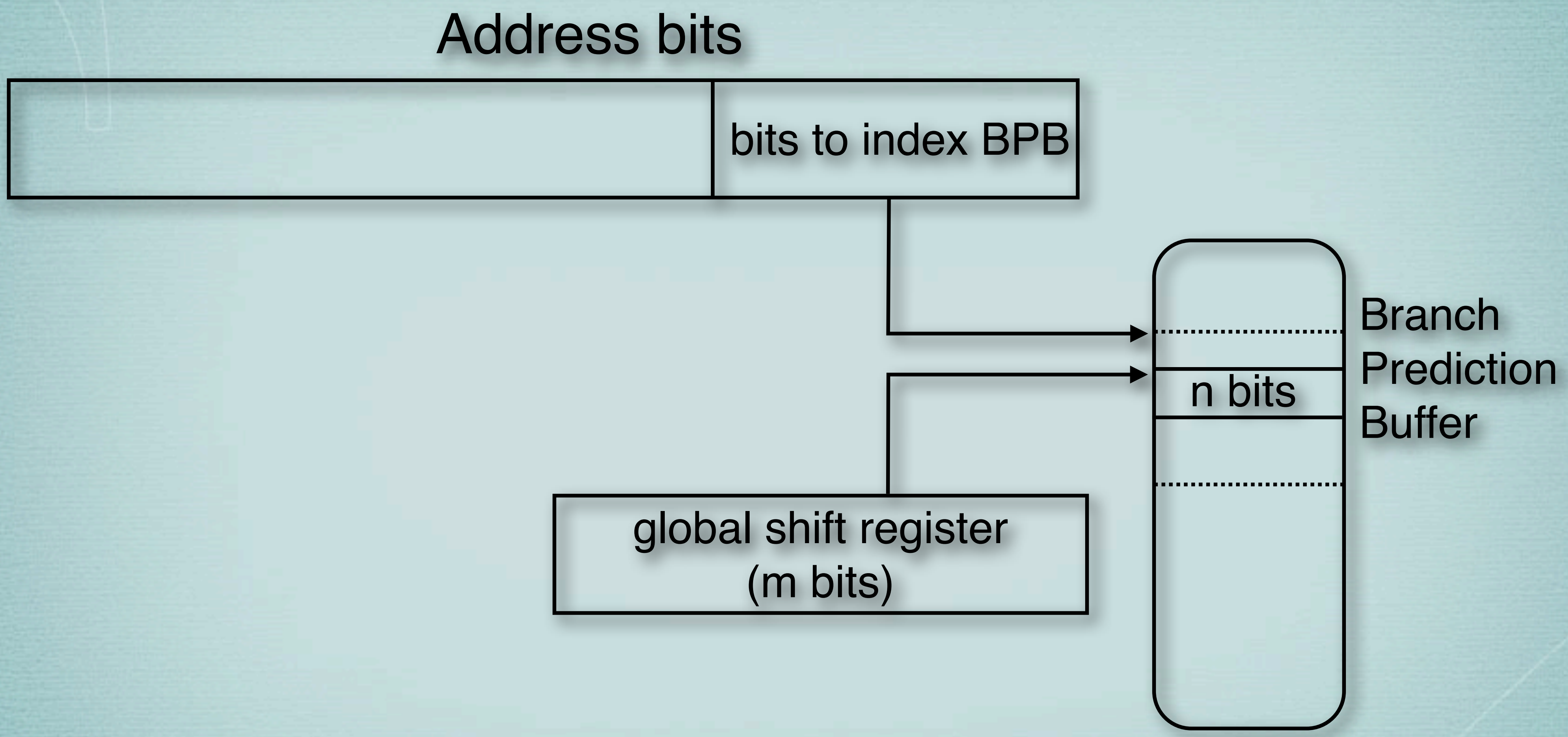


Two-bit Branch Predictor



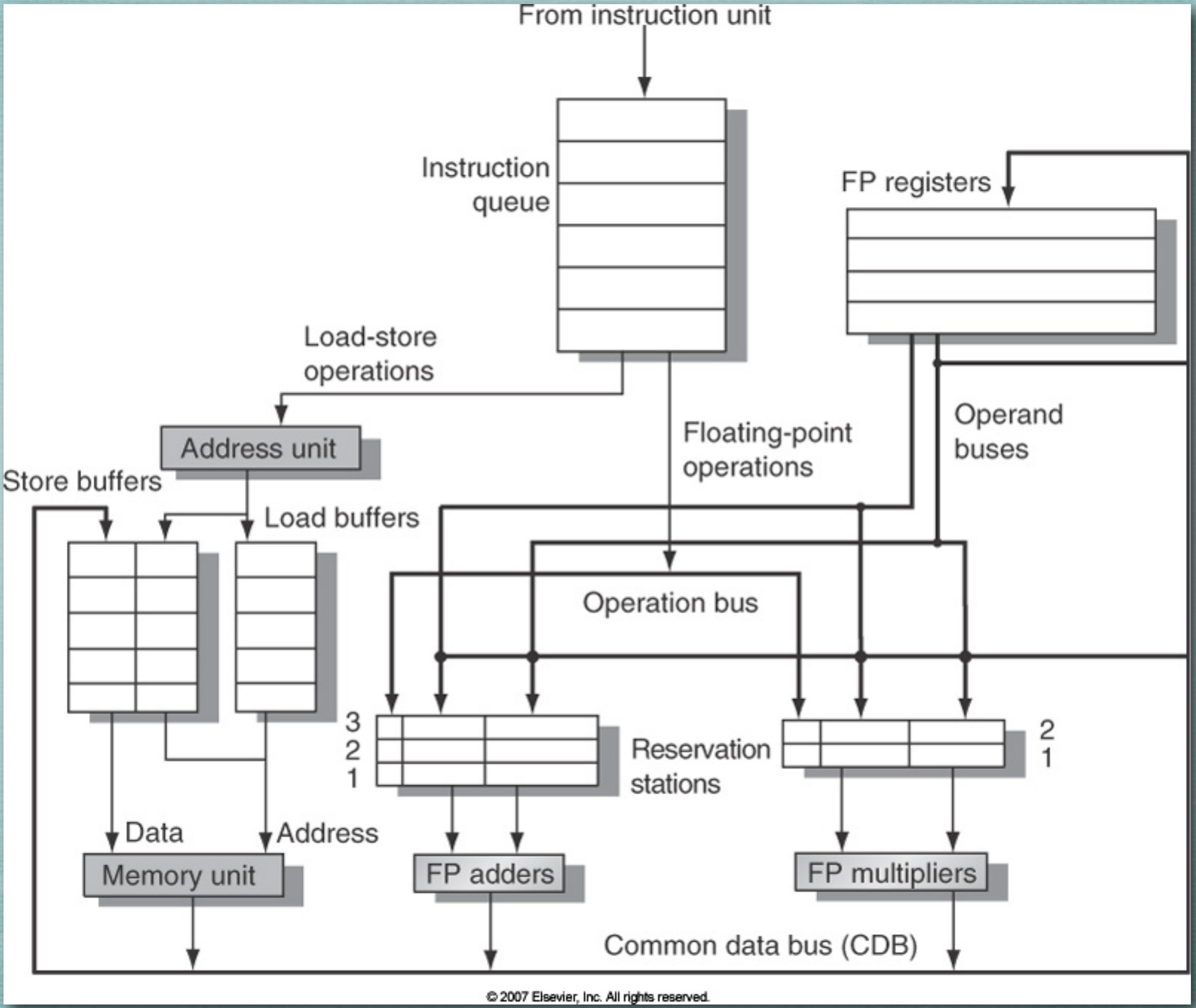
© 2007 Elsevier, Inc. All rights reserved.

General n-bit Correlating Branch Predictors



Use Branch Target Buffers (BTBs) for caching branch targets

Dynamic Scheduling: Tomasulo's Approach



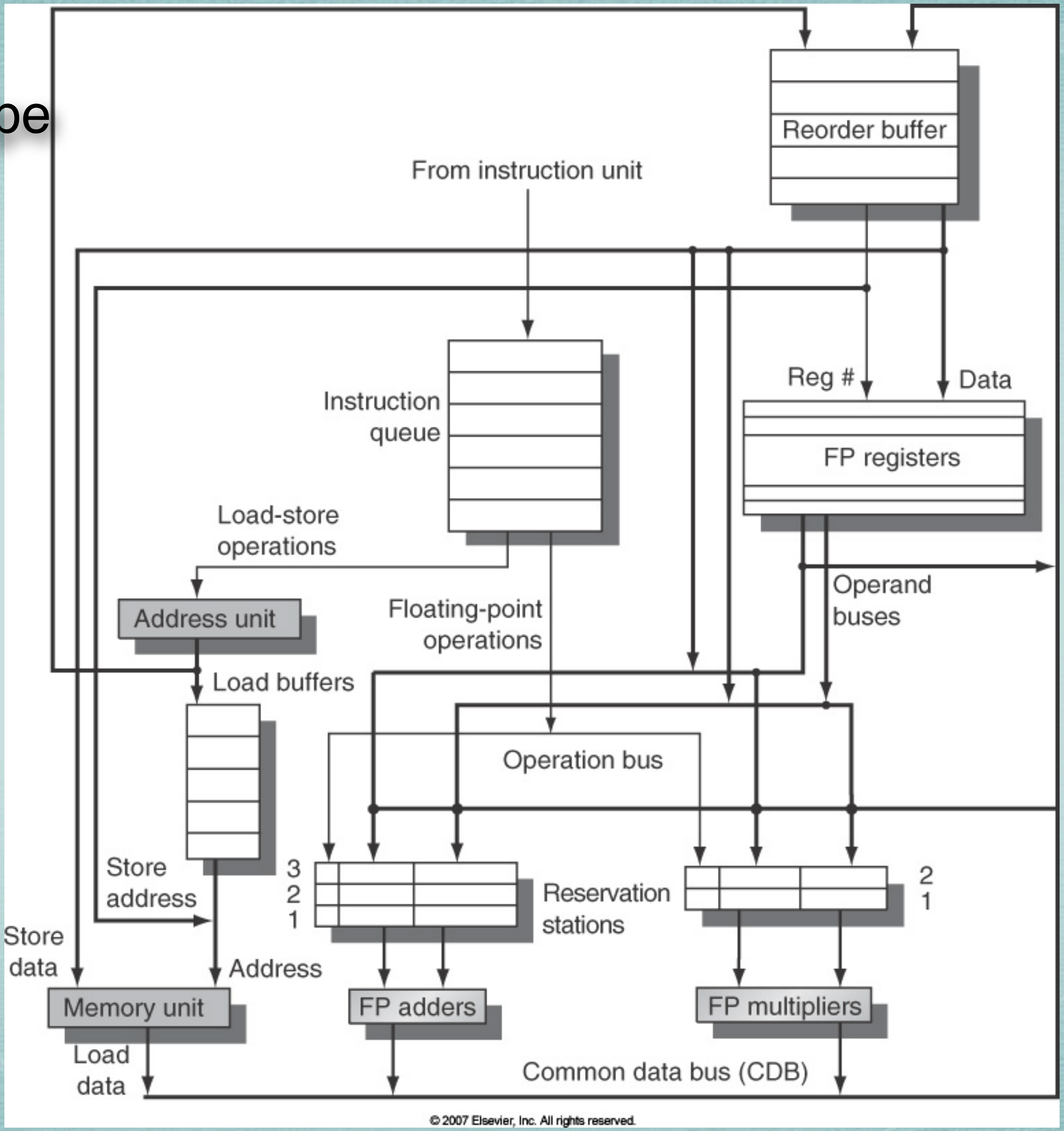
© 2007 Elsevier, Inc. All rights reserved.

Tomasulo's Approach: Observations

- RAW hazards handled by waiting for operands
- WAR and WAW hazards handled by register renaming
  - ⋆ only WAR and WAW hazards between instructions currently in the pipeline are handled; is this a problem?
  - ⋆ larger number of hidden names reduces name dependences
- CDB implements forwarding

Tomasulo's Approach + Speculation

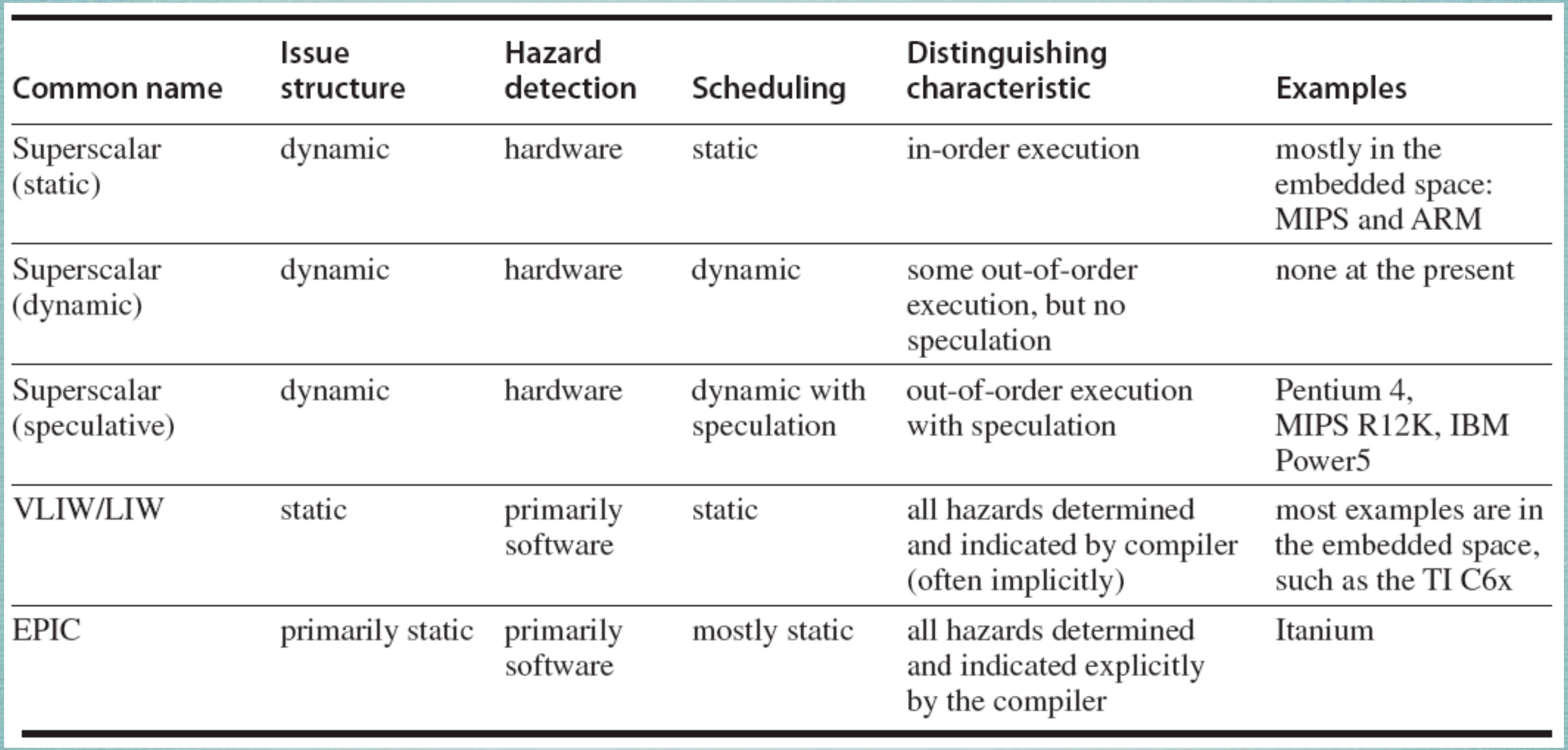
Fields in ROB
1. Instruction type
2. Destination
3. Value
4. Ready



Observations on Speculation

- Speculation enables precise exception handling
  - ⋆ defer exception handling until instruction ready to commit
- Branches are critical to performance
  - ⋆ prediction accuracy
  - ⋆ latency of misprediction detection
  - ⋆ misprediction recovery time
- Must avoid hazards through memory
  - ⋆ WAR and WAW already taken care of (how?)
  - ⋆ for RAW
    - ✳ don't allow load to proceed if an active ROB entry has Destination field matching with A field of load
    - ✳ maintain program order for effective address computation (why?)

Multiple Issue Processor Types

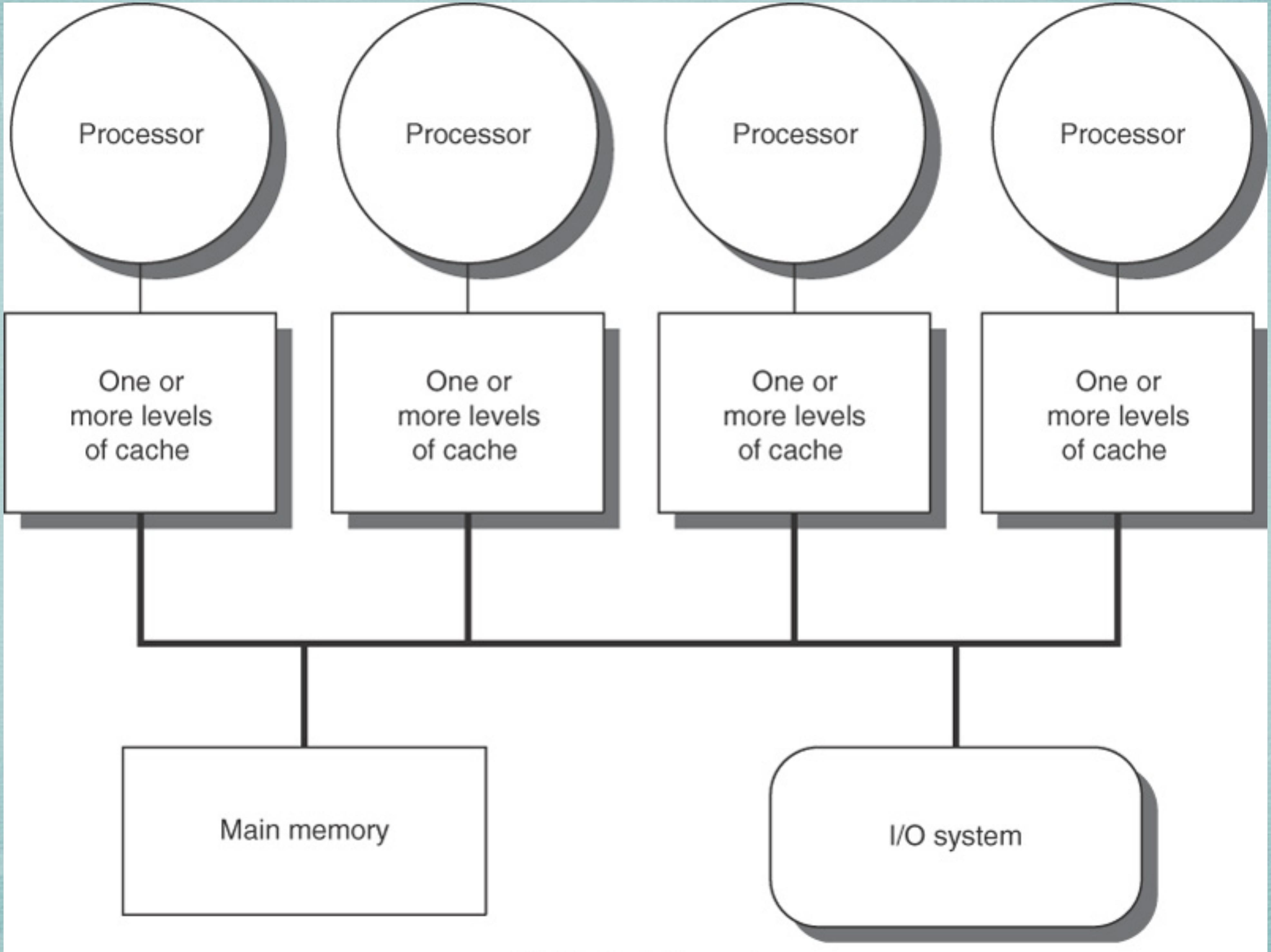


Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	dynamic	hardware	static	in-order execution	mostly in the embedded space: MIPS and ARM
Superscalar (dynamic)	dynamic	hardware	dynamic	some out-of-order execution, but no speculation	none at the present
Superscalar (speculative)	dynamic	hardware	dynamic with speculation	out-of-order execution with speculation	Pentium 4, MIPS R12K, IBM Power5
VLIW/LIW	static	primarily software	static	all hazards determined and indicated by compiler (often implicitly)	most examples are in the embedded space, such as the TI C6x
EPIC	primarily static	primarily software	mostly static	all hazards determined and indicated explicitly by the compiler	Itanium

Dyn. Scheduling+Multiple Issue+Speculation

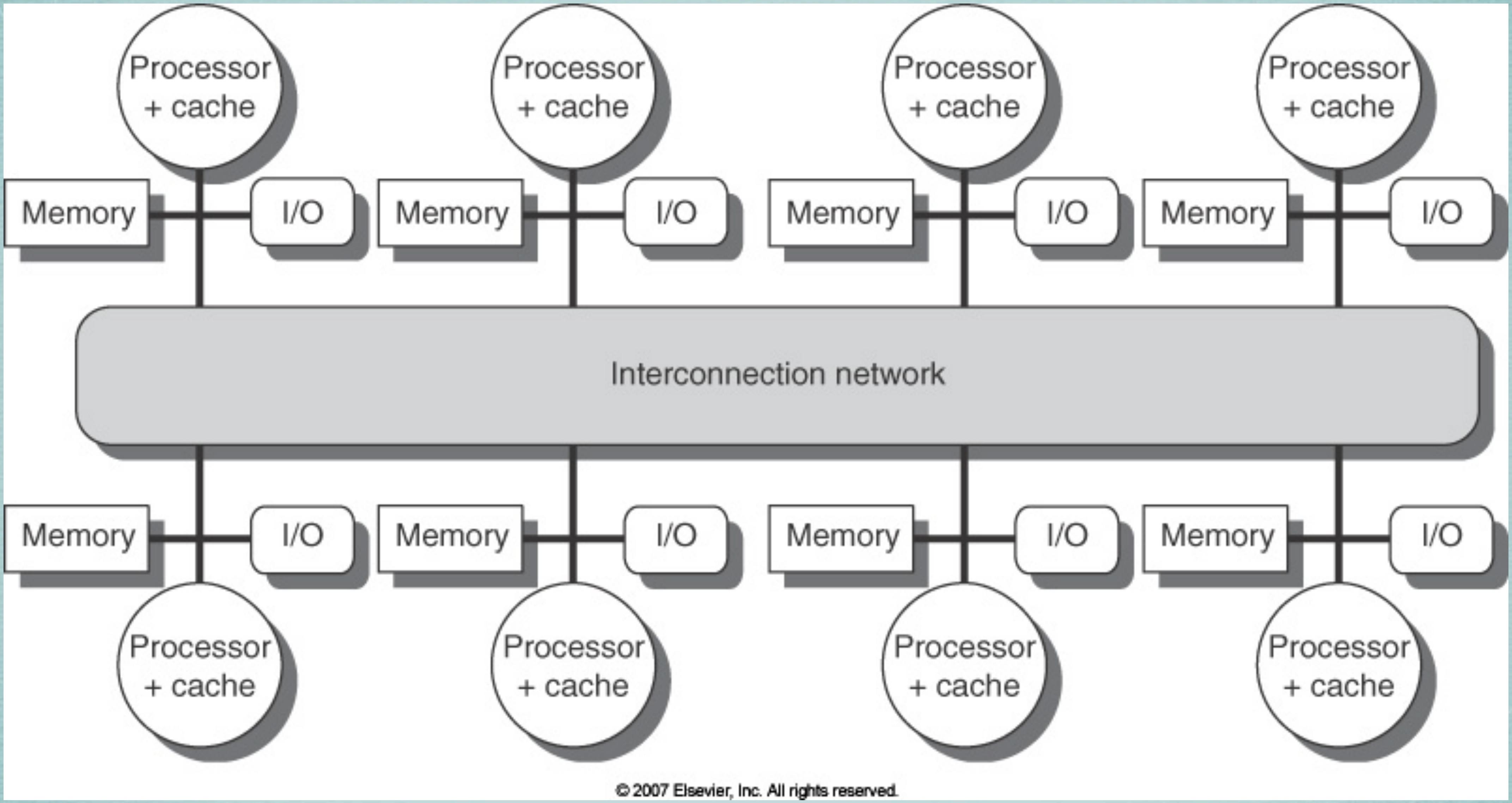
- Design parameters
  - ⋆ two-way issue (two instruction issues per cycle)
  - ⋆ pipelined and separate integer and FP functional units
  - ⋆ dynamic scheduling, but not out-of-order issue
  - ⋆ speculative execution
- Task per issue: assign reservation station and update pipeline control tables (i.e., control signals)
- Two possible techniques
  - ⋆ do the task in half a clock cycle
  - ⋆ build wider logic to issue any pair of instructions together
- Modern processors use both (4 or more way superscalar)

Shared-Memory Multiprocessors

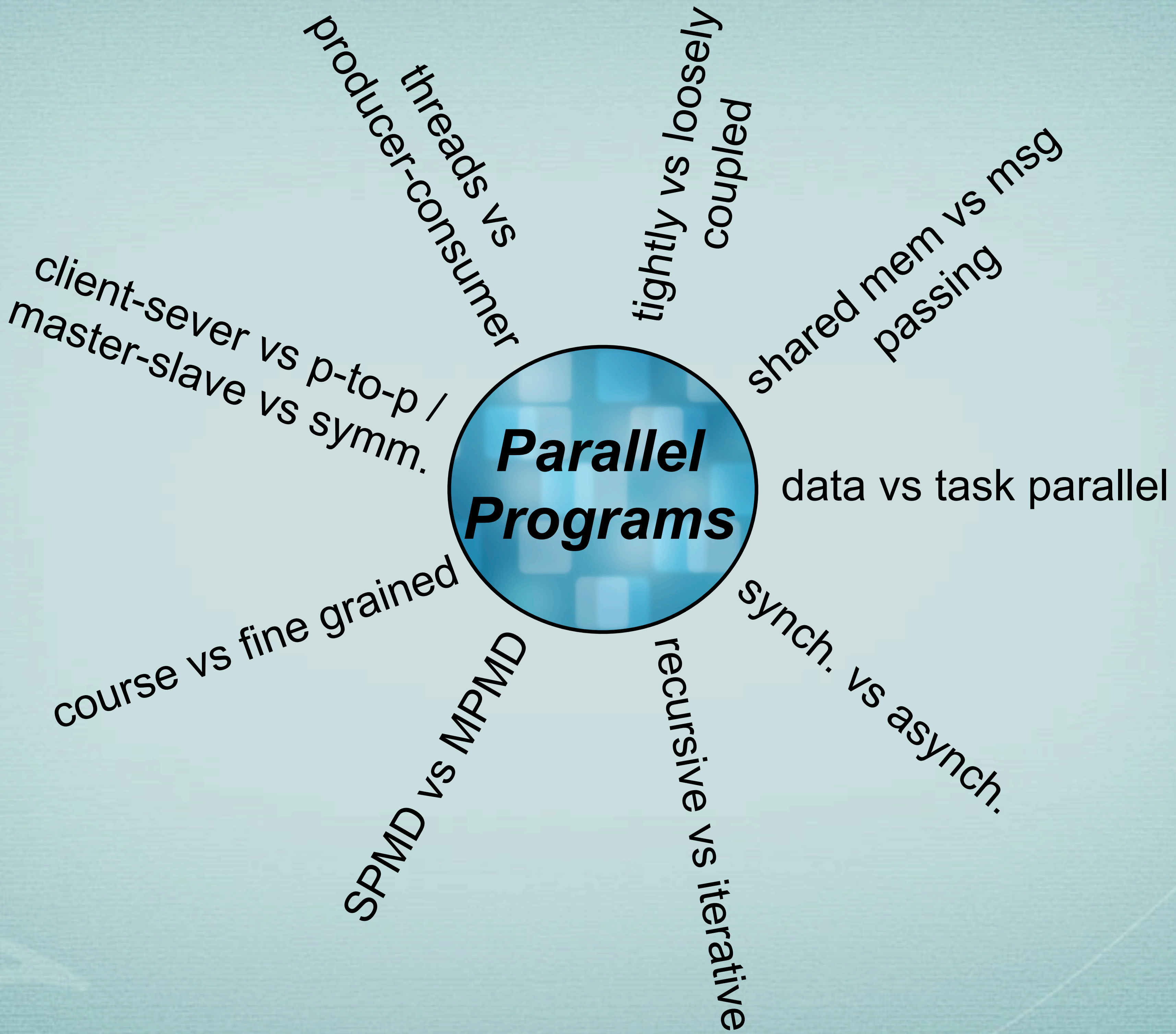

Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
I/O system

© 2007 Elsevier, Inc. All rights reserved.

Distributed-Memory Multiprocessors



Other Ways to Categorize Parallel Programming



Parallel Programs

threads vs producer-consumer

tightly vs loosely coupled

shared mem vs msg passing

data vs task parallel

synch. vs asynch.

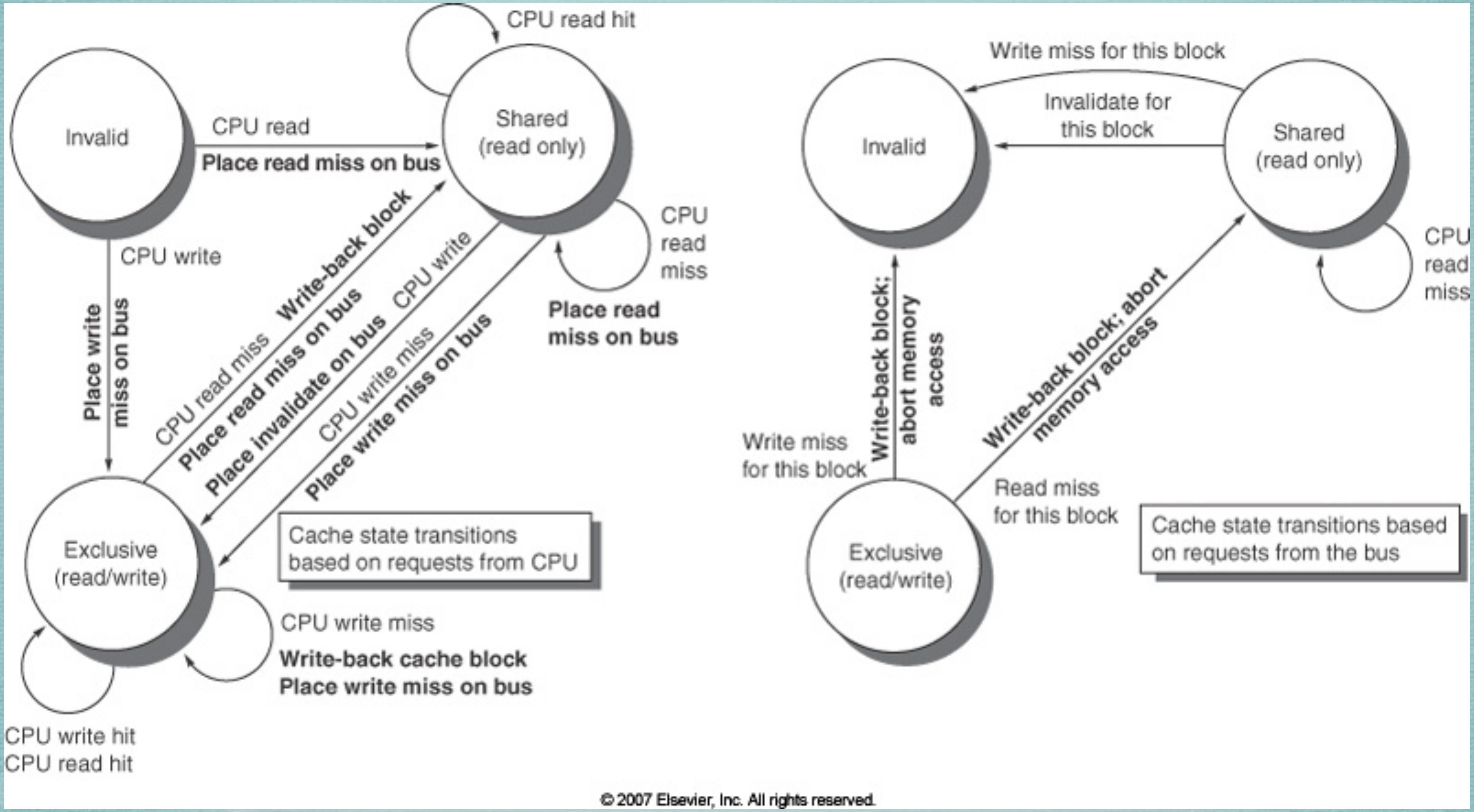
recursive vs iterative

SPMD vs MPMD

course vs fine grained

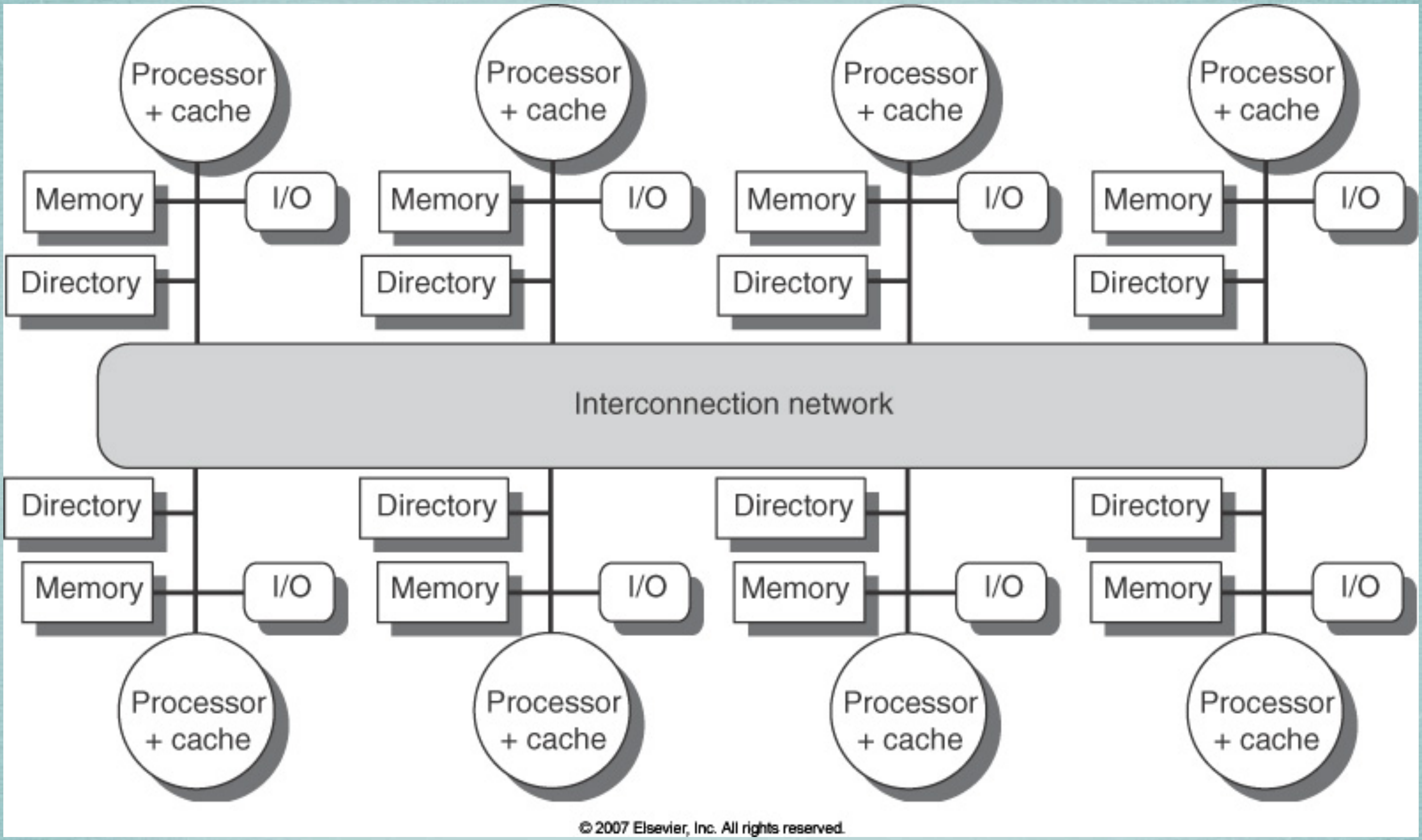
client-sever vs p-to-p / master-slave vs symm.

Write Invalidate Cache Coherence Protocol for Write-Back Caches

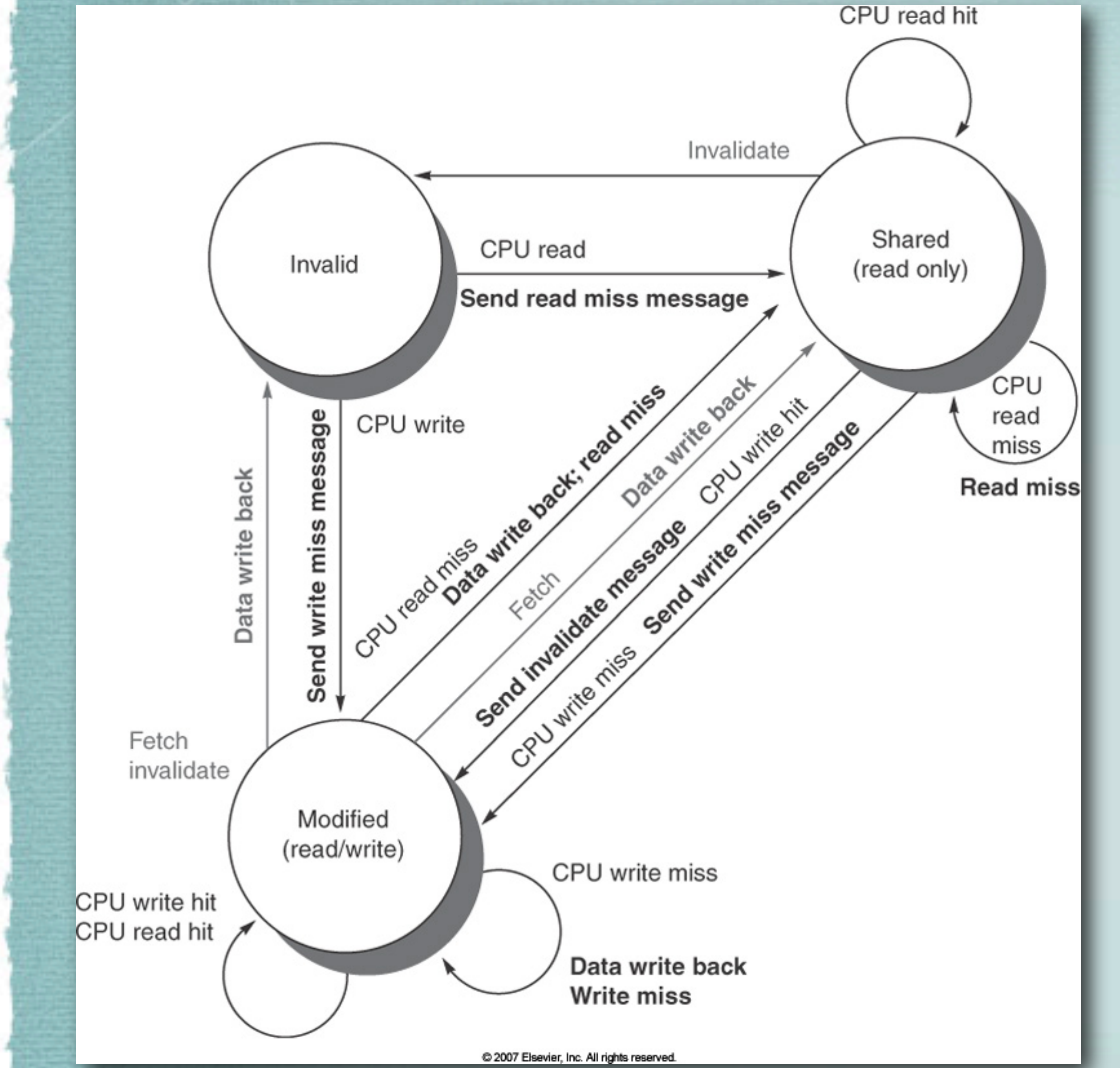


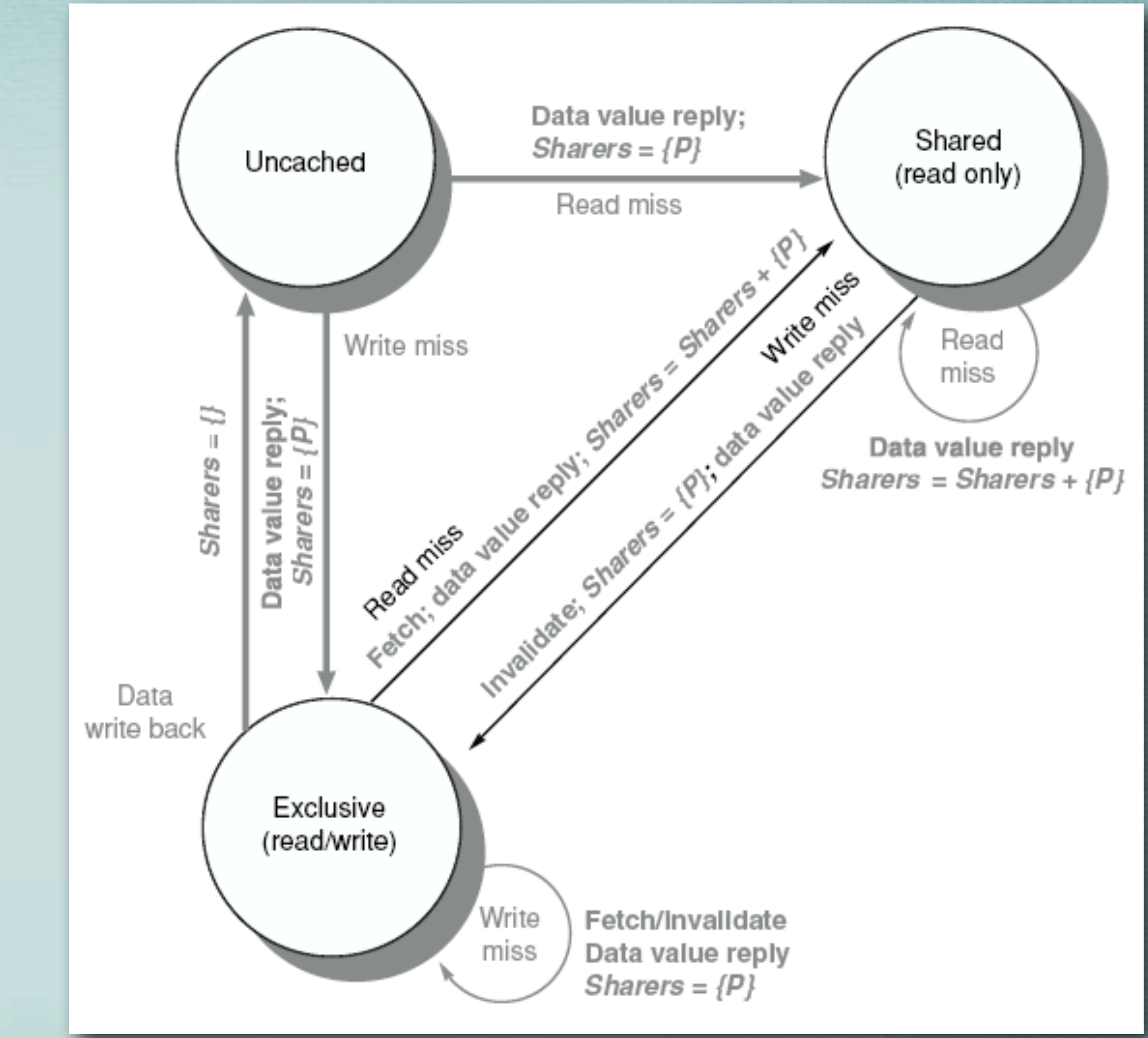
© 2007 Elsevier, Inc. All rights reserved.

Distributed Memory+Directories



Directory-Based Cache Coherence





Other Topics

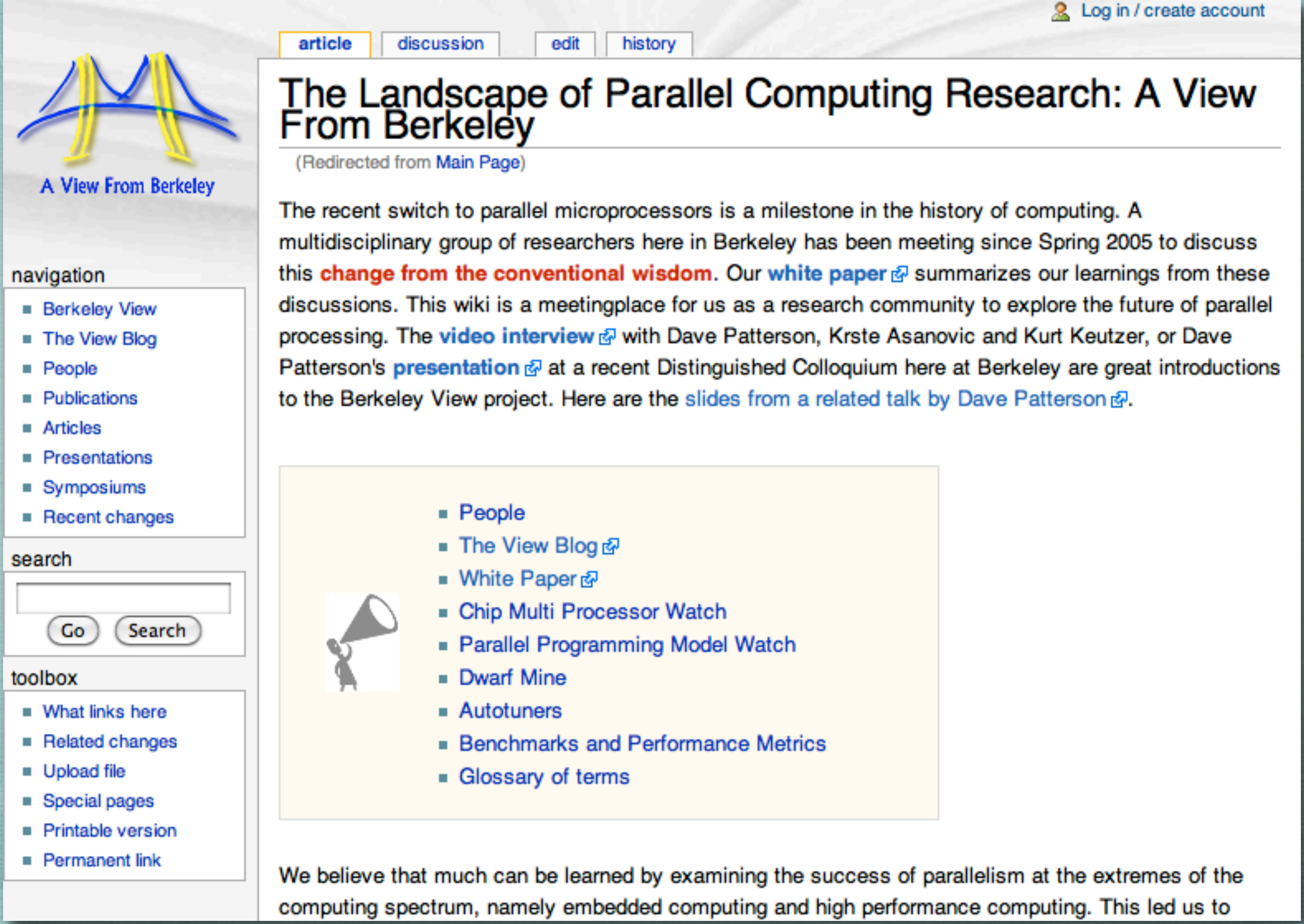
- x86 assembly programming
- VLIW / EPIC
- Vector processors
- Embedded systems
- Scientific applications
- GPUs and GPGPUs
- CUDA and OpenCL
- Interconnection networks
- Virtualization

WHAT'S NEXT?

Future

- Continued importance of parallel programming
    - challenge: how to program multiprocessors
    - role of programming languages and compilers
- Convergence or specialization?
    - “standardization” of general purpose architecture
    - migration of “special-purpose” CPUs for general use

Landscape of Parallel Computing Research: A View from Berkeley



The Landscape of Parallel Computing Research: A View From Berkeley

(Redirected from Main Page)

The recent switch to parallel microprocessors is a milestone in the history of computing. A multidisciplinary group of researchers here in Berkeley has been meeting since Spring 2005 to discuss this **change from the conventional wisdom**. Our **white paper** summarizes our learnings from these discussions. This wiki is a meetingplace for us as a research community to explore the future of parallel processing. The **video interview** with Dave Patterson, Krste Asanovic and Kurt Keutzer, or Dave Patterson's **presentation** at a recent Distinguished Colloquium here at Berkeley are great introductions to the Berkeley View project. Here are the slides from a related talk by Dave Patterson.

- People
- The View Blog
- White Paper
- Chip Multi Processor Watch
- Parallel Programming Model Watch
- Dwarf Mine
- Autotuners
- Benchmarks and Performance Metrics
- Glossary of terms

We believe that much can be learned by examining the success of parallelism at the extremes of the computing spectrum, namely embedded computing and high performance computing. This led us to