The Data-Flow model of Computation in the Multi-core era

Skevos (Paraskevas) Evripidou
Department of Computer Science,
University of Cyprus
skevos@cs.ucy.ac.cy

Historical Overview

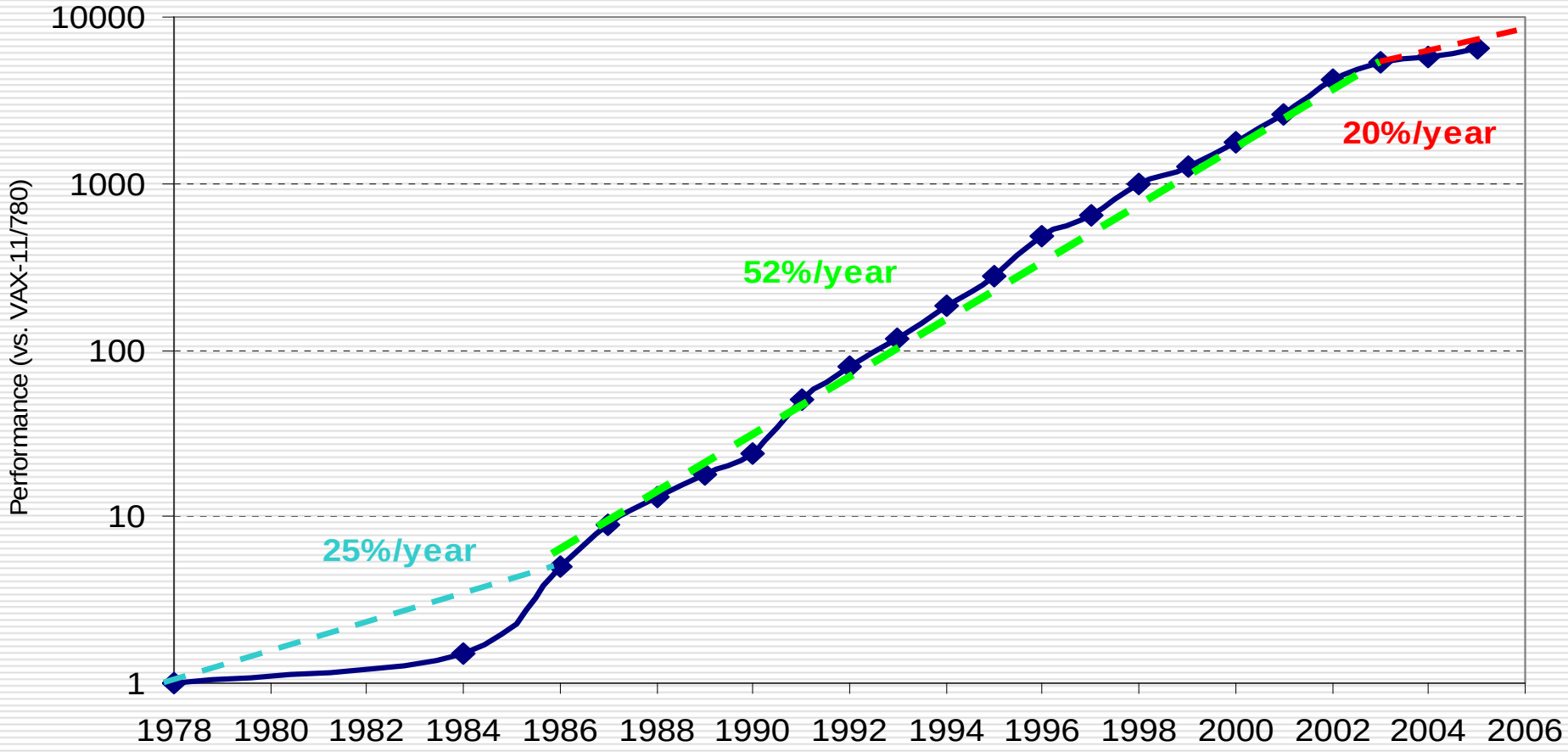
- Since the advent of Digital computers, in the early 1940's, the computer architecture field has been dominated by the sequential model of execution.
  - von Neumann model of execution
- Since the 1960's proponents of Parallel Processing have being predicting the end of sequential computing and the swift to parallel processing.
  - Michael Flyn develop his classification of Parallel system because he believed that Parallel processing was going mainstream after the ILLIAC IV development- Personal communication
- Chip designers have been using the power granted to them by Moore's Law to postpone the shift indefinitely.
- The Revenge of the Parallel Processing Nerds: At the dawn of the new millennium the sequential computing had a head on collision with the Memory Wall.
  - The problem most of them are not around anymore (retire) or have switch field

Use of excessive force

- The trend in the 90's was to build high-end microprocessors with
  - Large Cache and Multiple Issue/Superscalar to Tolerate Memory Latency (Memory Wall)
  - Exploit ILP (through increased complexity)
    - Out of Order Execution (OOE)
      - Deconstruct the Sequential program with hardware assisted implicit synchronization.
      - Ad Hoc Data-Flow the hard and very costly way. (Restricted dataflow)
- Contributed to the rise of the Power and Heat Walls

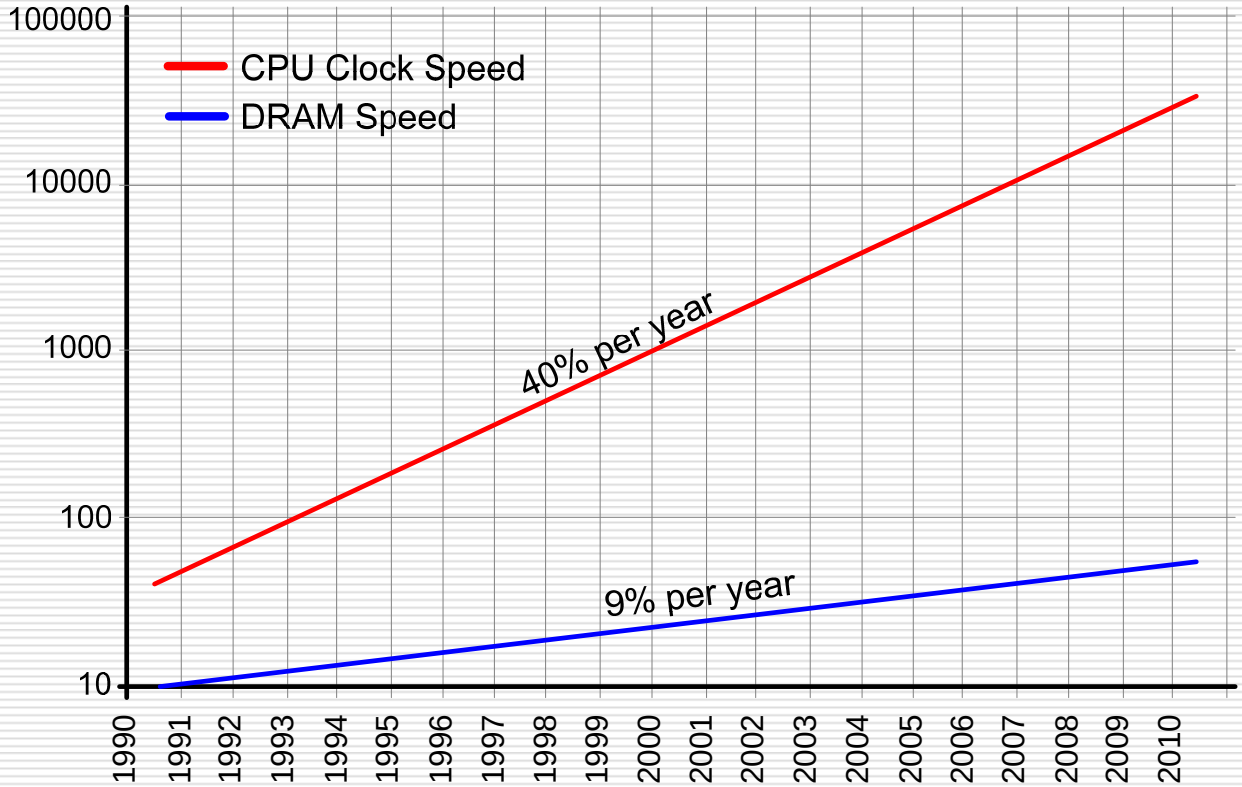
Arvind Keynote speech ISCA 2005,
: RAMPS project -- Dave Patterson

Uniprocessor Performance (SPECint)



- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: 20%/year 2002 to 2005**

CPU-Memory Speed



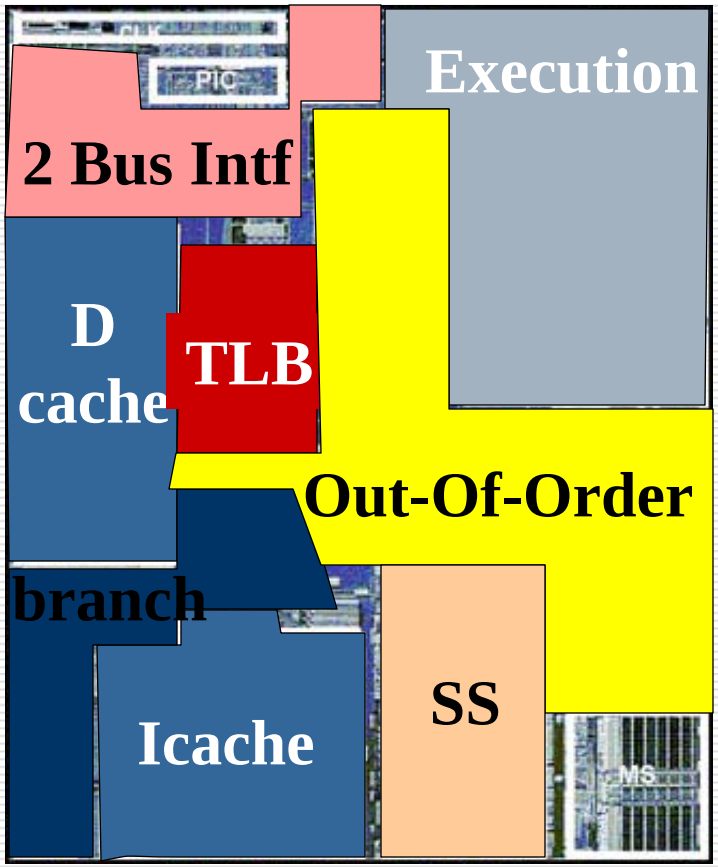
Overcoming the memory wall:

- 8KB L1 (Intel 486, 1989)
- On-board L2 (Pentium Pro, 1995)
- On-package L2 (Pentium II, 1997)
- On-die L2 (Pentium III, 1999)

Source: J. Patterson, “Modern Microprocessors”, www.pattosoft.com.au/Articles/ModernProcessors

Where have all the transistors gone?

- Superscalar (multiple instructions per clock cycle)
- **3 levels of cache**
- **Branch prediction (predict outcome of decisions)**
- **Out-of-order execution (executing instructions in different order than programmer wrote them)**



Intel Pentium III (10M transistors)

Source: J. Patterson, "The future of Microprocessors", NAE presentation 2001

A view of the Research Labs at INTEL

- "A major breakthrough in Boosting IPC is the introduction of out-of-order execution, where instruction execution depends on Data-Flow, not on the Program counter"
- "out-of-order execution involves dependency analysis and instruction scheduling, therefore its takes longer time (more pipe stages) to process an instruction in an out-of-order microprocessor. "
- "With deeper pipe, an out-of-order microprocessor suffers more from branch misprediction"
- "Needles to say, an out-of-order microprocessor especially a wide-issue one is much more complex and power hungry."

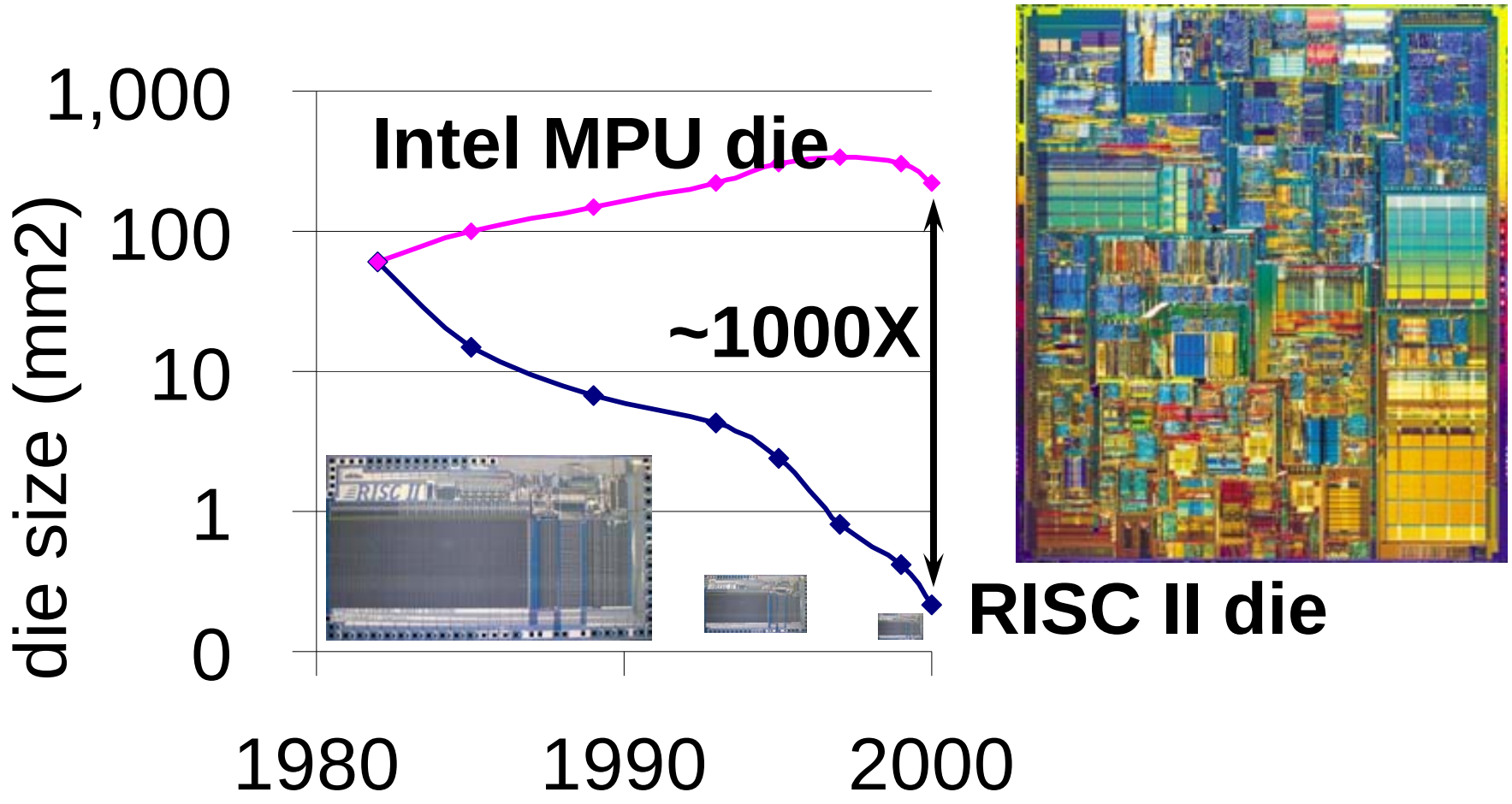
Quotations from
Ronny Ronen et al, Coming Challenges in Microarchitevcture
and Architecture. Proceedings of the IEEE, vol 89, No 3. March 2001

The Products of Intel: P4 --2000-2006

- 1.3 GHz - 3.8 GHz
- 20 Pipeline stages vs 10 for P3
- At the launch of the P4, Intel stated NetBurst was expected to scale to 10 GHz (over several fabrication process generations).
- In 2005/6 Intel shifted development away from P4 (NetBurst) to focus on the cooler running Pentium M architecture.

- In March 2006, Intel announced the Intel Core microarchitecture, which puts greater emphasis on energy efficiency and performance per clock.

Moore's Law vs. Common Sense?



- Scaled 32-bit, 5-stage RISC II 1/1000th of current MPU, die size or transistors (1/4 mm²)

Source: J. Patterson, "The future of Microprocessors", NAE presentation 2001

Switch to Multi-core chips

- The switch did not address the cause of the problem but it was just an engineering work-around.
- Similar very-complex and power hungry cores at lower frequencies.
- Still most of transistors are used to overcome the major limitations of the Control flow model: Intolerance to Memory Latencies

Multi-core chips and the Concurrency Challenge

- Old Challenges: the inability of the sequential model to tolerate long latencies.
  - Techniques used to tackle this problem, such OOE and large caches, increase complexity and power consumption.
- New Challenges: Concurrency is now *the* major issue for success
  - Extending the sequential model with concurrent constructs is an ad hoc solution
  - Revisit alternative models that are naturally parallel

- Data-Flow is a formal and elegant model for handling concurrency
  - Functional/Side-effect free
    - Easy programmability
  - An operation is scheduled for execution only after all its input data have been produced.
    - Tolerance to Memory, Synchronization, and Network latencies
  - The Optimized Sisal compiler was the best parallelizing compiler of its time

Data-flow 101

- Tolerance to Memory and communication Latencies.
  - Instructions are executed after their Input data are ready!
  - This can be optimized to mean present in the faster level of the Memory hierarchy
  - Immunity to the Power Wall
- Tolerance to synchronization latencies
  - No need for Barriers, Busy-waits etc
  - Data-Flow semantics taken care of these
- Data-Flow execution is functional
  - Observes the single assignment semantics
  - No need for exclusive access, locks etc.
  - No Side –effects
  - Easier to parallelise since only true data-decencies exits in a Data-flow graph

Data-flow Architectures

- Proposed in the 70s (Most people credit Jack Dennis of MIT as the "father" of Data-Flow)
  - Asynchrony: Execution is driven by data availability.
  - Functional: No side effects.
- Implementation: Provide "Context-switch" support at the instruction level
- Data-flow programs are represented as graphs:
  - The nodes (actors) are the instructions of the program
  - The arcs carry data from producer to consumer actor
- Enabling rule: an instruction is enabled (i.e. executable) if all operands are available.
- An instruction can be fired (i.e. executed) only after it becomes enabled.

Dynamic Data-Flow (DDF)/Tagged Token DF (TTDF)

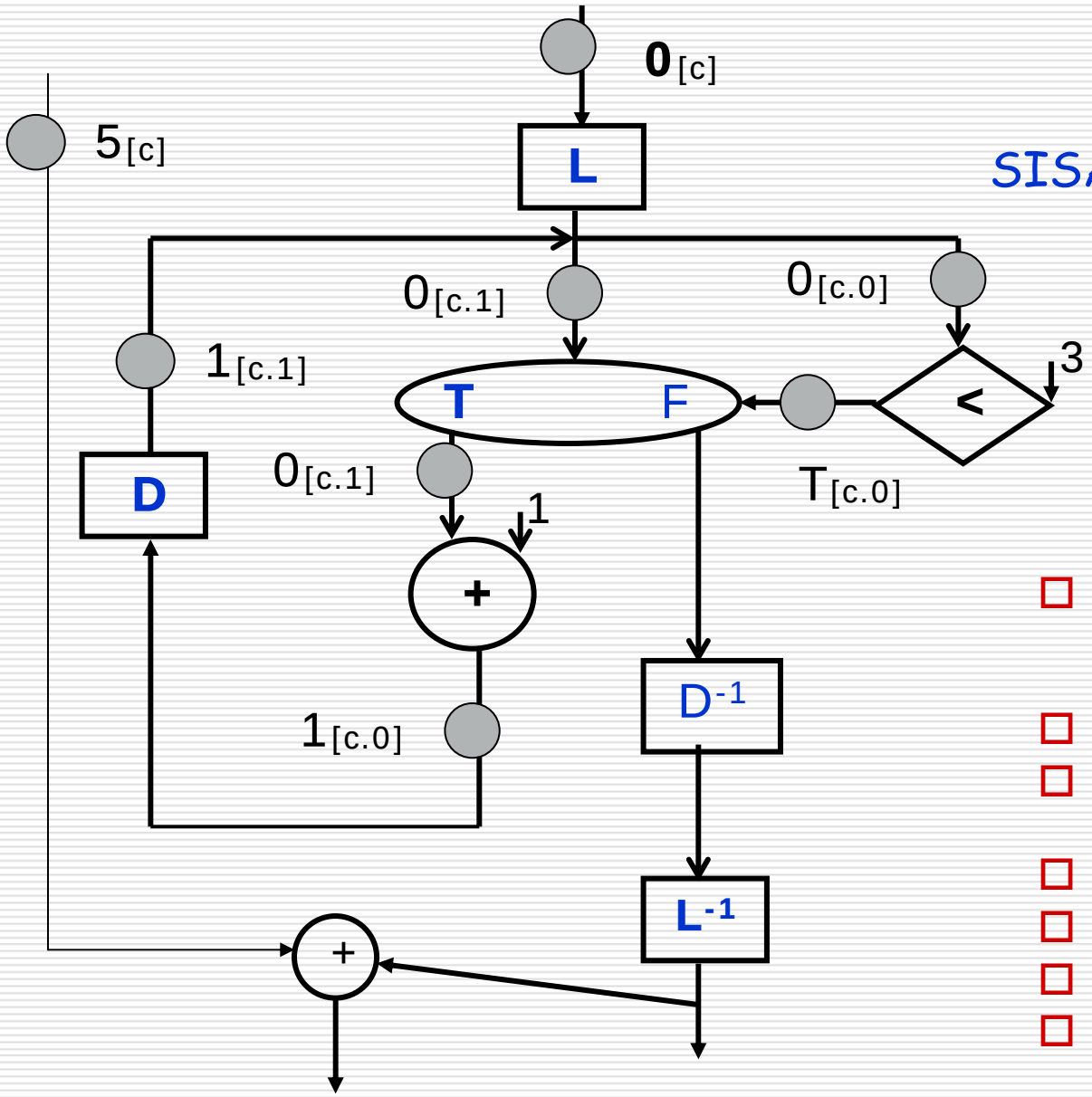
- Developed independently by Gurd & Watson at the University of Manchester and Arvind at UCI and MIT
- Each loop iteration or subprogram invocation can execute in parallel as a separate instance of a reentrant subgraph.
- Each token has a tag: The address of the instruction for which the particular data value is destined and context information
- $V_{[c.s.i]}$ c: context, s: inst. pointer and i: Iter. identifier,
- Each arc can be viewed as a bag that may contain an arbitrary number of tokens with different tags.
- The enabling and firing rule is now:

  A node is enabled and fired as soon as tokens with identical tags are present on all input arcs.

Iteration in DDF: U-interpreter

Y:= 5 + for i in 1,3
returns value of i
end for

SISAL code!



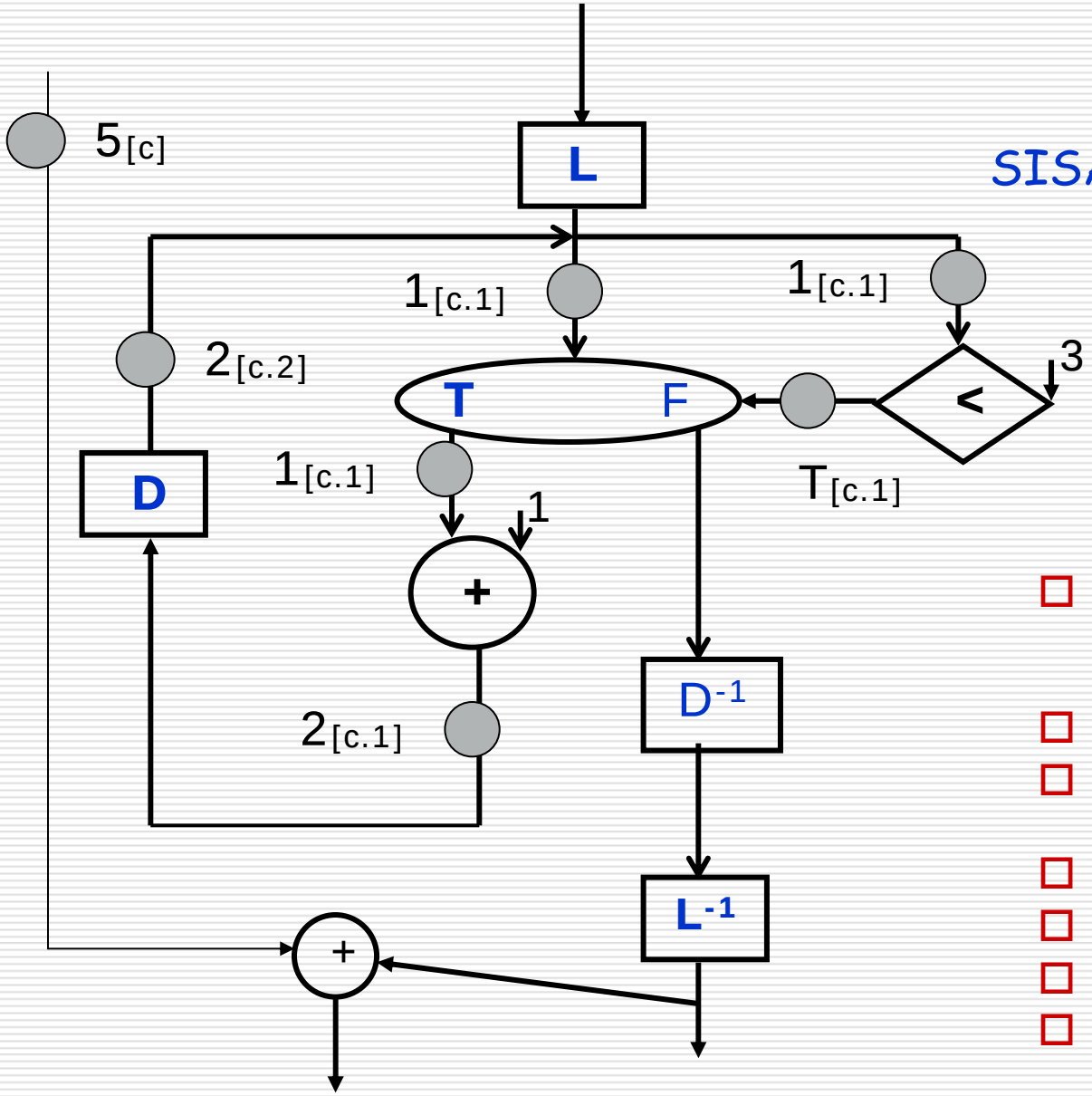
Iteration 1

- U-Intpreter: Special Actors for Tag Manipulation
- L: add a loop context
- L^{-1}: Restores orig. Context
- D: Inc. loop identifier.
- D^{-1}: Resets loop identif.
- A: Function call
- A^{-1}: return from function call

Iteration in DDF: U-interpreter

Y:= 5 + for i in 1,3
returns value of i
end for

SISAL code!



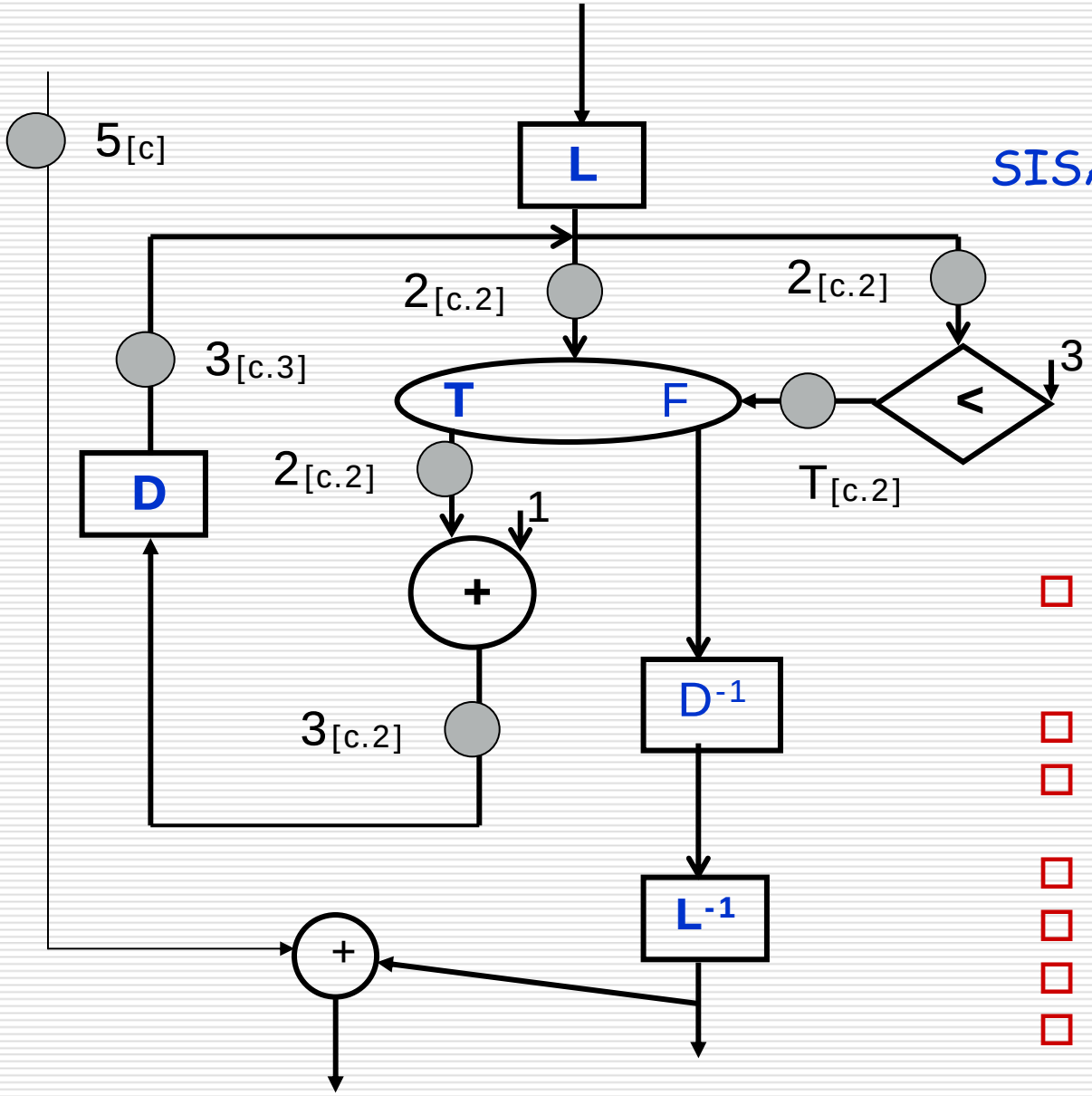
Iteration 2

- U-Intpreter: Special Actors for Tag Manipulation
- L: add a loop context
- L^{-1}: Restores orig. Context
- D: Inc. loop identifier.
- D^{-1}: Resets loop identif.
- A: Function call
- A^{-1}: return from function call

Iteration in DDF: U-interpreter

Y:= 5 + for i in 1,3
returns value of i
end for

SISAL code!



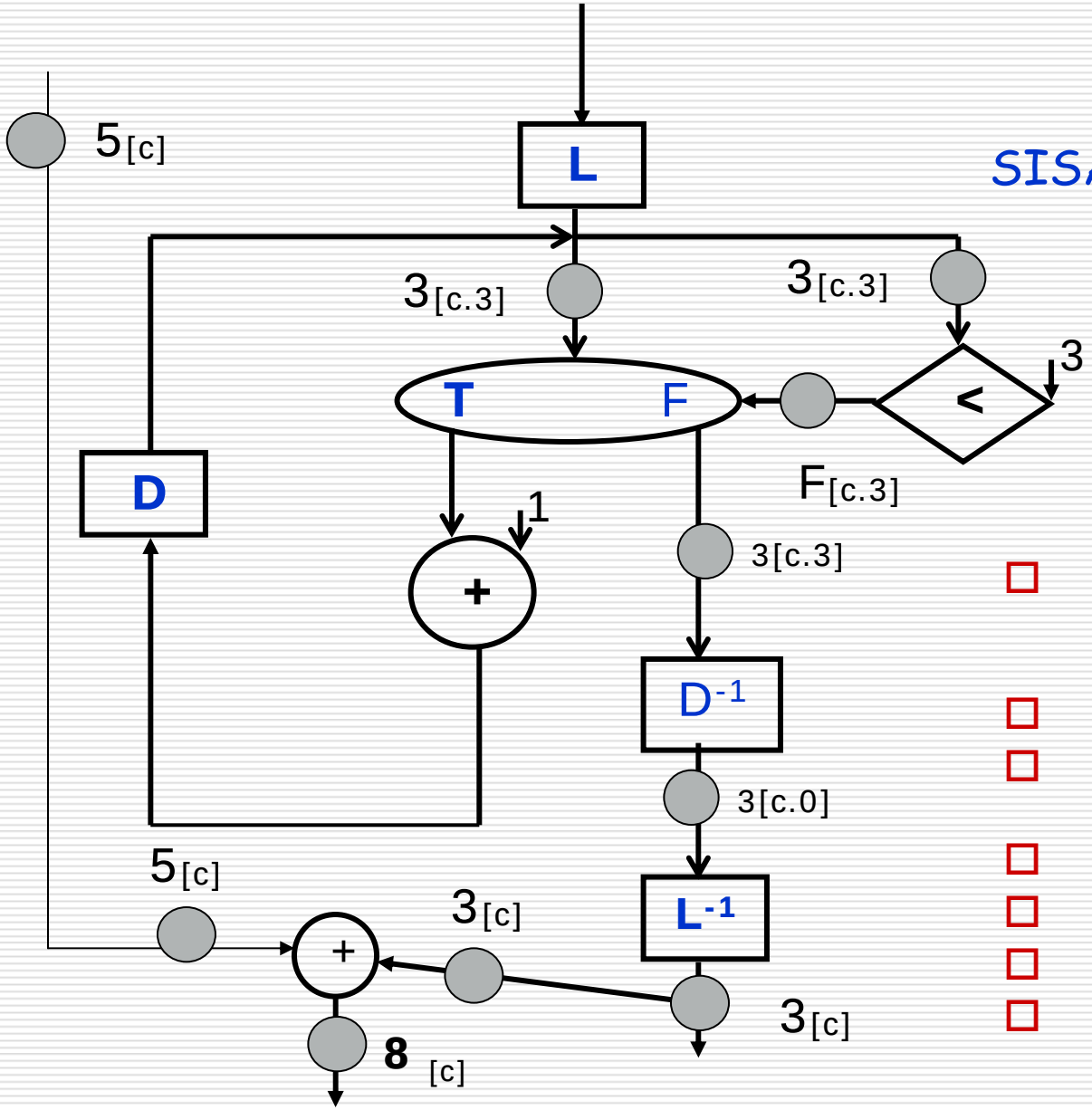
Iteration 3

- U-Intpreter: Special Actors for Tag Manipulation
- L: add a loop context
- L^{-1}: Restores orig. Context
- D: Inc. loop identifier.
- D^{-1}: Resets loop identif.
- A: Function call
- A^{-1}: return from function call

Iteration in DDF: U-interpreter

Y:= 5 + for i in 1,2
returns value of i
end for

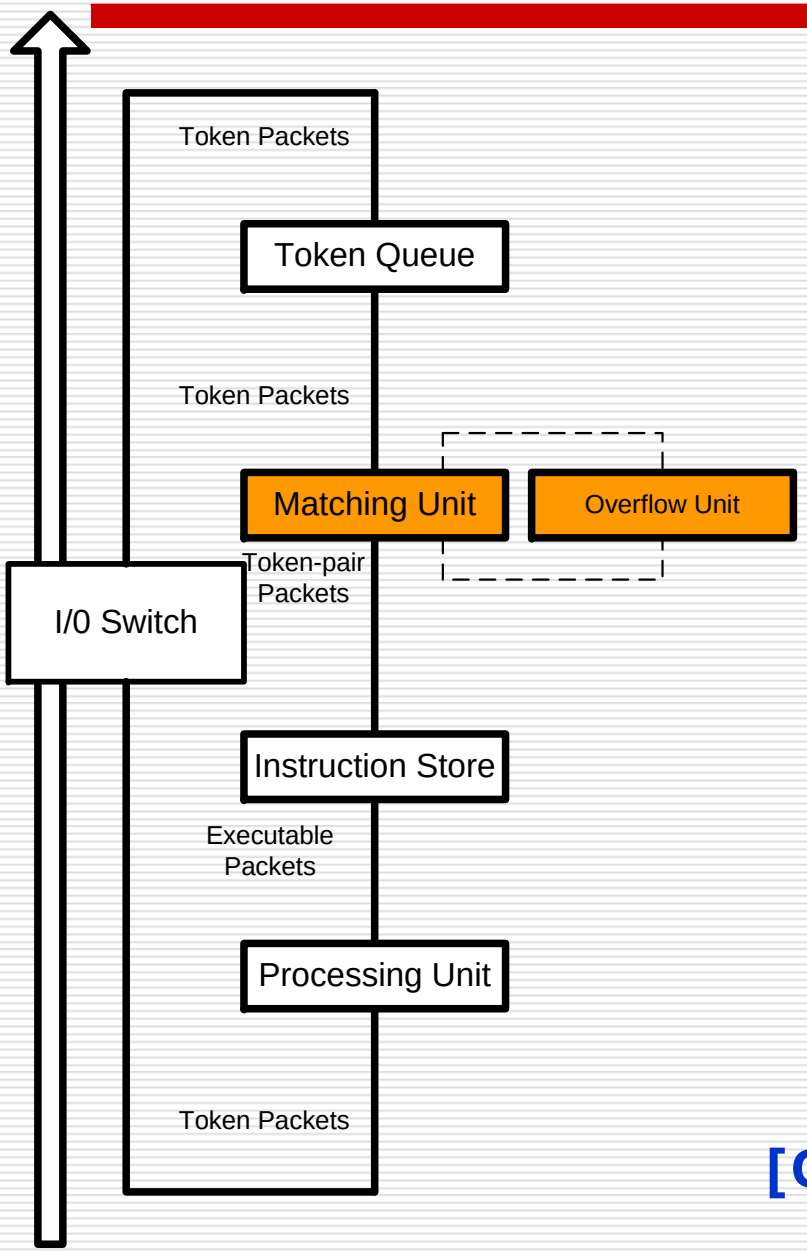
SISAL code!



Loop Exit

- U-Intpreter: Special Actors for Tag Manipulation
- L: add a loop context
- L^{-1}: Restores orig. Context
- D: Inc. loop identifier.
- D^{-1}: Resets loop identif.
- A: Function call
- A^{-1}: return from function call

Manchester Dataflow Machine [Gurd & Watson 1979]



- Operational 1981
- Performance of 1.2 MIPS
- Matching Unit 1M tokens
- Parallel Hashing: mapping of incoming tag to a set of 8 slots
- Associative matching at the Slot

Program	Vax 11/780	Dataflow 1 FU	Dataflow 12 FUs
RSIM/1	0.04	1.36	0.16
RSIM/1	0.10	8.26	0.89
RSIM/1	0.08	6.12	0.68
RSIM/1	0.28	8.67	0.88

[Gurd, Kirkham & Watson 1985]

Limitations of TTDF machines

1. Implementation of Waiting-Matching Store.
   - Associate memory is ideal but unfeasible
   - Hashing techniques are not fast enough to be a single pipeline stage.
   - Amount of parallelism is unpredictable, might fill up the Waiting-Matching store and cause deadlock.
     - Overflow is possible but complicated.
2. Unbounded size of the activity names.
3. Different types of Stores (Matching store, Program store,Token Queue) made it difficult for memory management.
4. Poor performance with sequential code

[Arvind, Bic, Ungerer 1991]

Monsoon Speed Up Results

September, 1992

Boon Ang, Derek Chiou, Jamey Hicks

	speed up				critical path (millions of cycles)			
	1pe	2pe	4pe	8pe	1pe	2pe	4pe	8pe
Matrix Multiply 500 x 500	1.00	1.99	3.90	7.74	1057	531	271	137
Paraffins n=22	1.00	1.99	3.92	7.25	322	162	82	44
GAMTEB-2C 40 K particles	1.00	1.95	3.81	7.35	590	303	155	80
SIMPLE-100 100 iters	1.00	1.86	3.45	6.27	4681	2518	1355	747

Could not have asked for more

Slide from Arvind's Keynote speech at ISCA 2005

Dynamic Data-Flow (DDF) Summary

- Elegant solution: parallel processing with implicit synchronization
- It can exploit the ultimate amount of parallelism.
  - Loop throttling to limit the amount of parallelism!
- Immunity to high communication and memory latencies

- Throughout the years innovative Data-Flow prototypes showed very good relative performance
  - In absolute performance they did not fare well when compared to commercial offerings of the same era.
- Difficult to benefit directly from efficient constructs and building blocks of the von Neumann model
- If you cannot beat them Join-Them

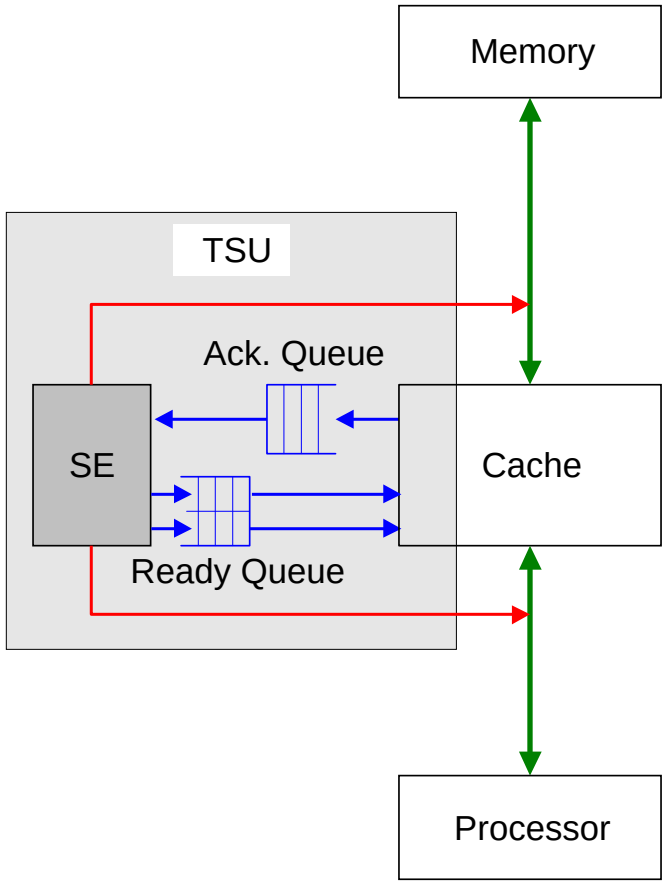
Our view

- The Computer Science Community has resisted the move to a parallel model of execution such as Data-Flow because it did not have to do it!
  - Control flow was good enough for everyone to keep its job.
- The switch to Multi-core has brought concurrency to the mainstream.
- Now the basic building block, the microprocessor, has to exploit concurrency:
  - Option1: continue doing it in ad hoc manner
  - Option2: Good time to reconsider alternative models such a data-flow
- In the near term the more likely "winner" will be systems that can utilize as much as possible from the existing State of Art know-How

Data-Driven Multithreading overview

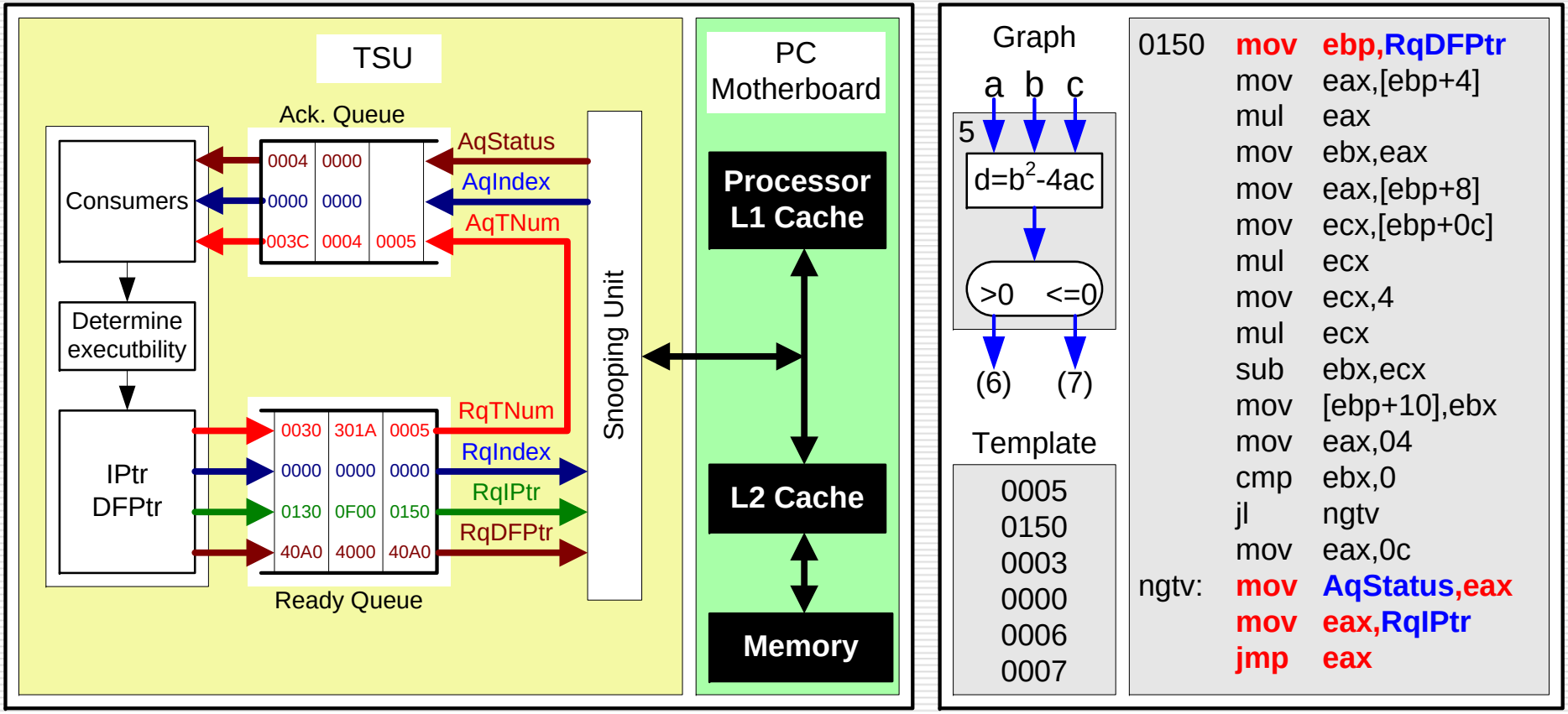
- Compiler driven thread generation
  - Data-Driven scheduling of Threads
  - Sequential execution within a thread
- Non Blocking--Threads execute to completion
- Can be Implemented efficiently with conventional microprocessors with the addition of memory mapped hardware unit: Thread Synchronization Unit (TSU)
- CacheFlow: Data-Driven perfecting improves drastically the hit ratio of the cache and at the same time requires much smaller cache memories.
  - Reduces space and power consumption,
  - Reducing further the effect of long memory latencies.

Thread Synchronization Unit (TSU)



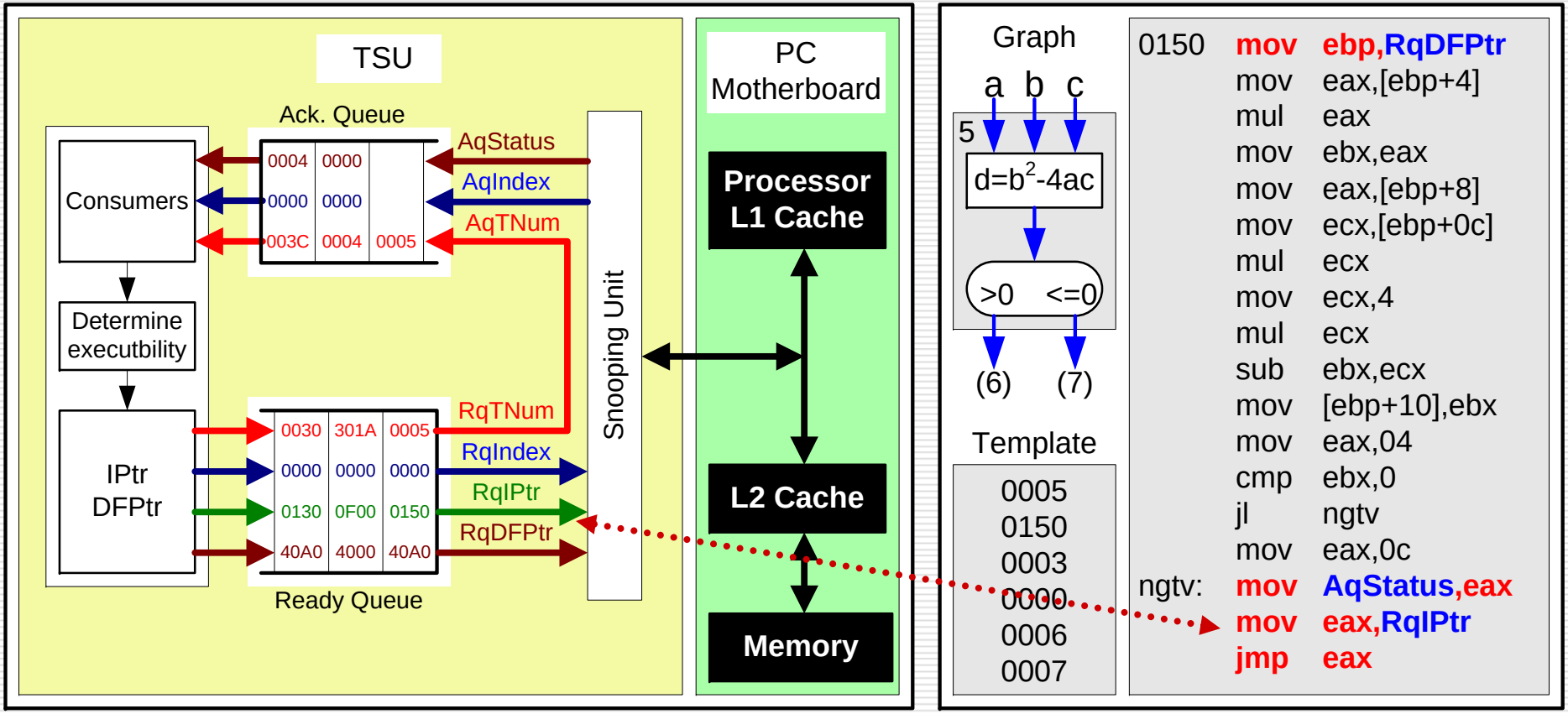
- Use a commercial microprocessor for the computational Engine
- Implement the functionality of the Synchronization Engine and the two queues in an extra module.
- The TSU integrates the function of the SE, the RQ and AQ.
- The processor does not have any knowledge about the presence of the TSU.
- Five addresses are reserved for the communication of the processor and the TSU.
- The TSU uses snooping to intercept these addresses and process them accordingly.

The TSU/Processor Interface



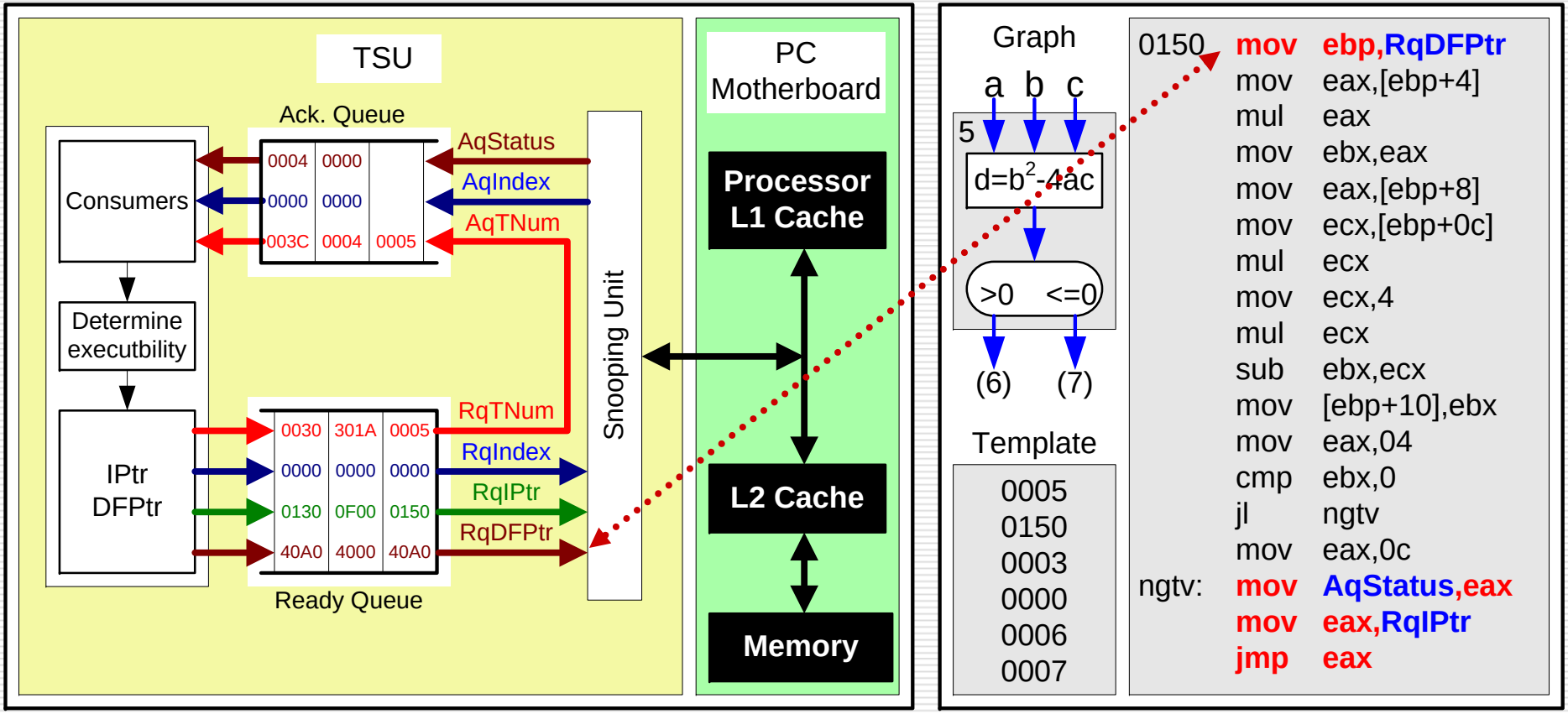
The TSU communicates with the CPU through five non-cachable memory addresses. These can also be I/O addresses.

The TSU/Processor Interface



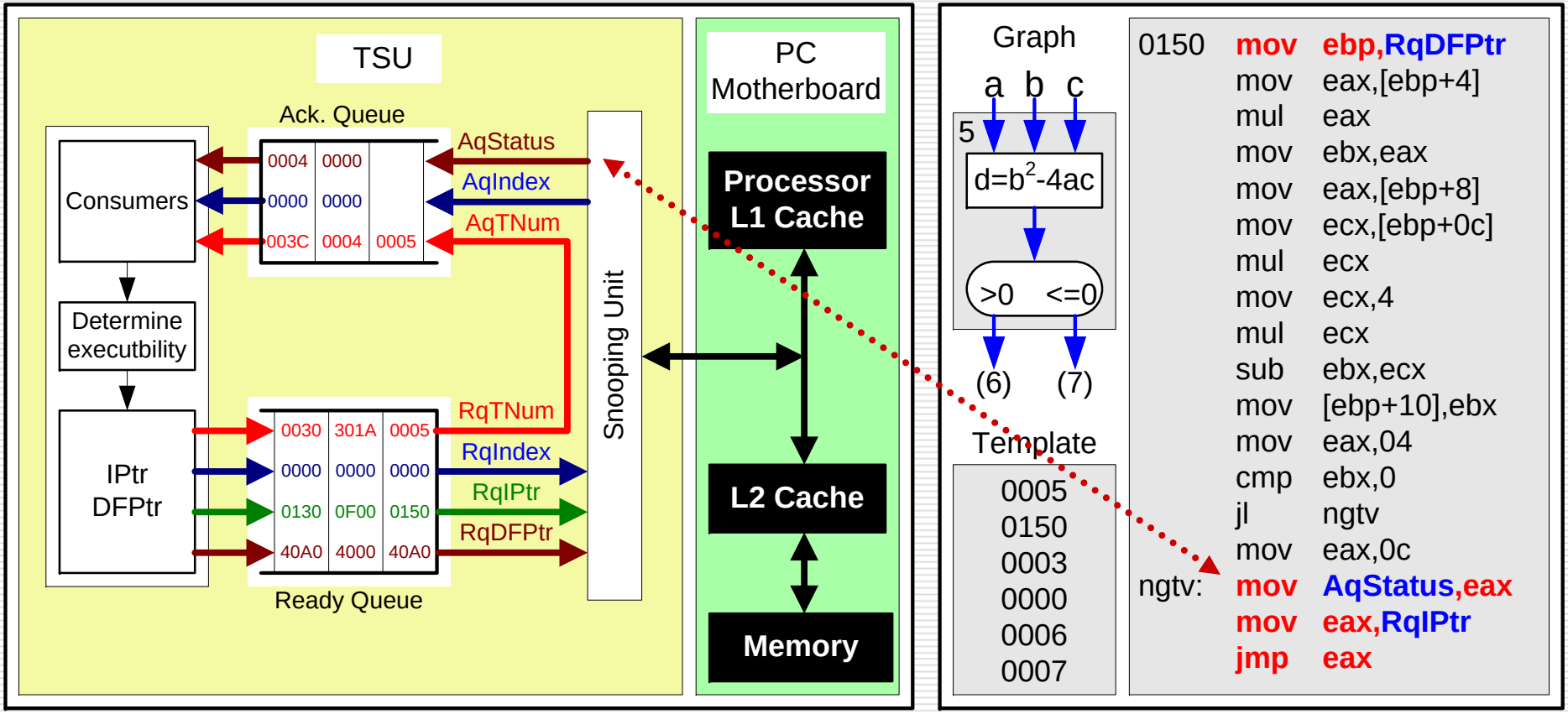
The RqIPtr used to provide to the CPU the address of the next thread to be executed.

The TSU/Processor Interface



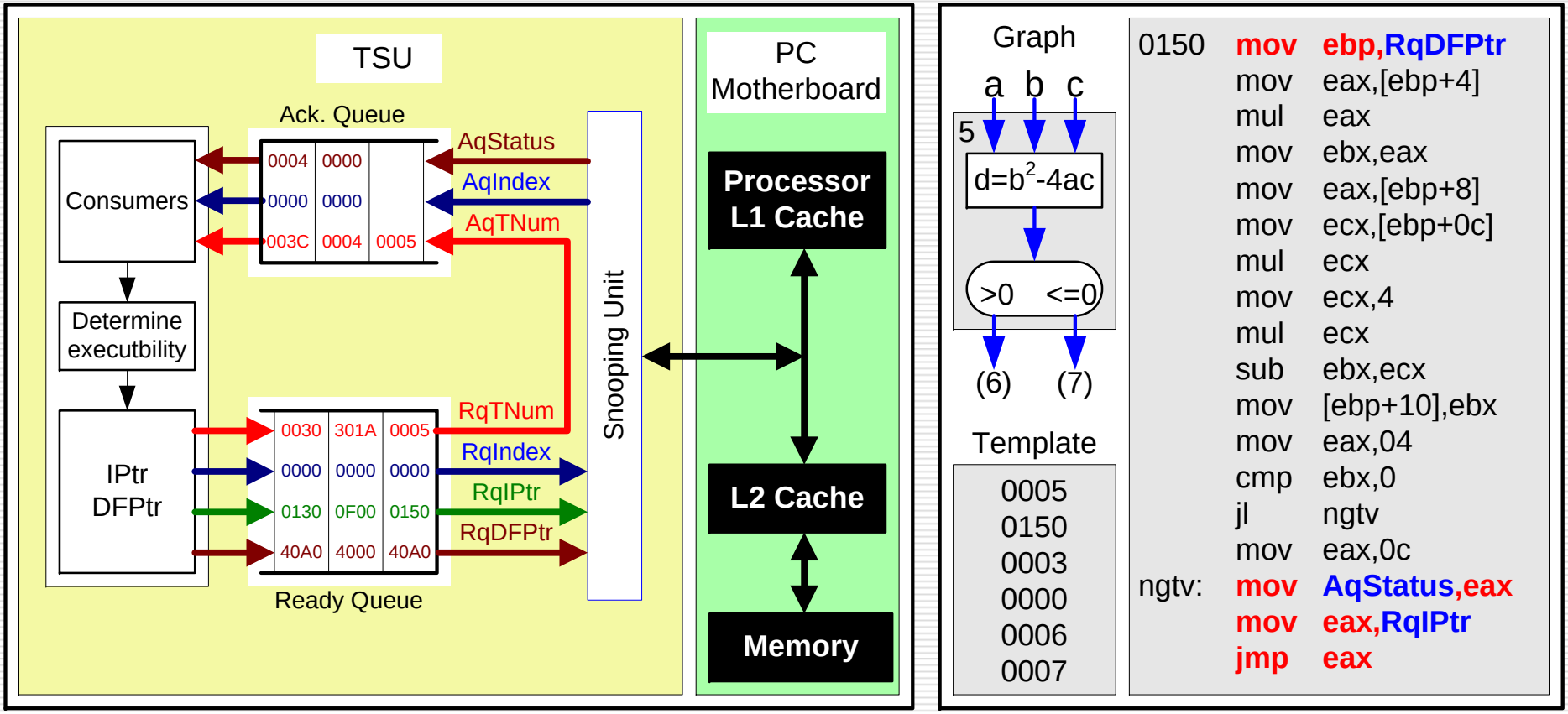
The RqDFPtr used to provide to the CPU the address of the data frame of the thread.

The TSU/Processor Interface



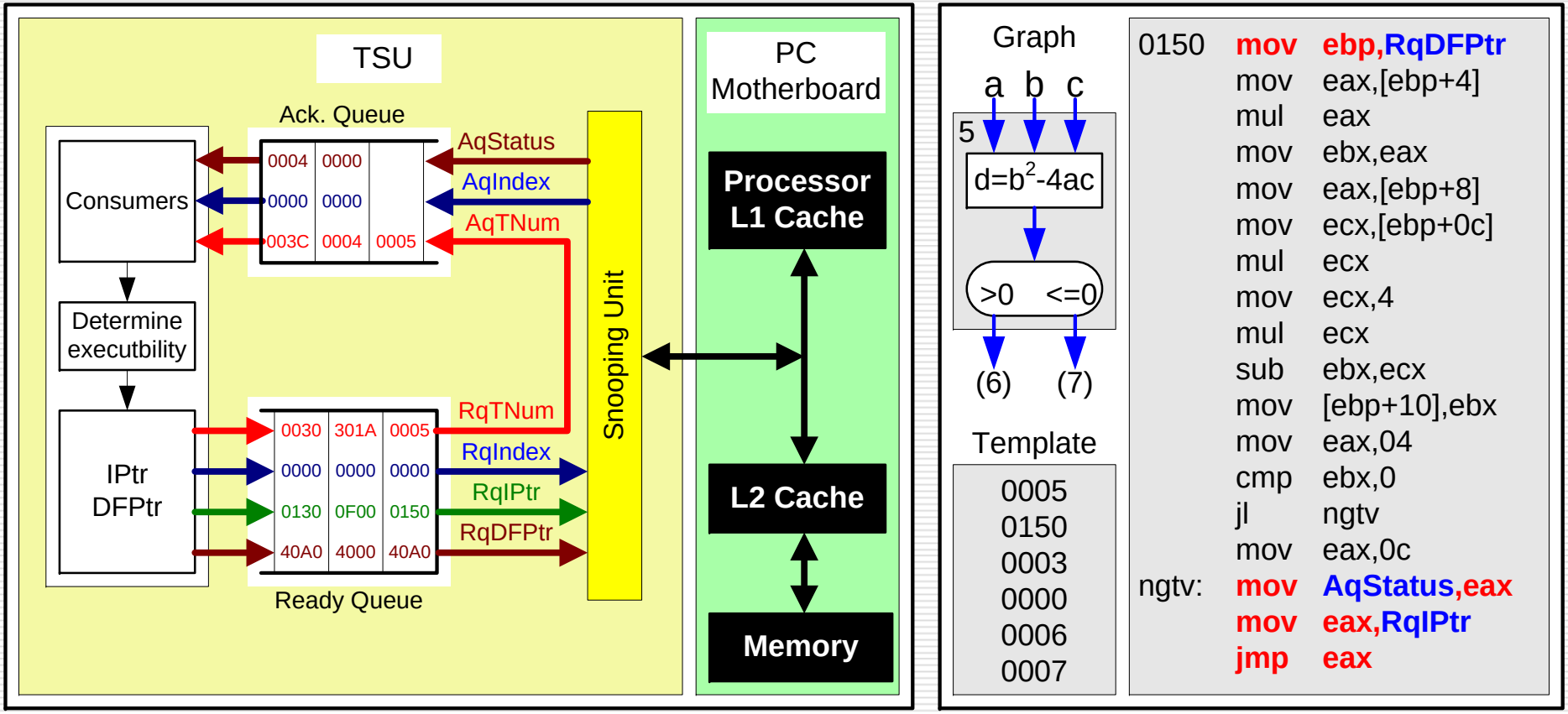
The AqStatus used by the CPU to provide to the TSU Information about the status of the completed thread.

The TSU/Processor Interface



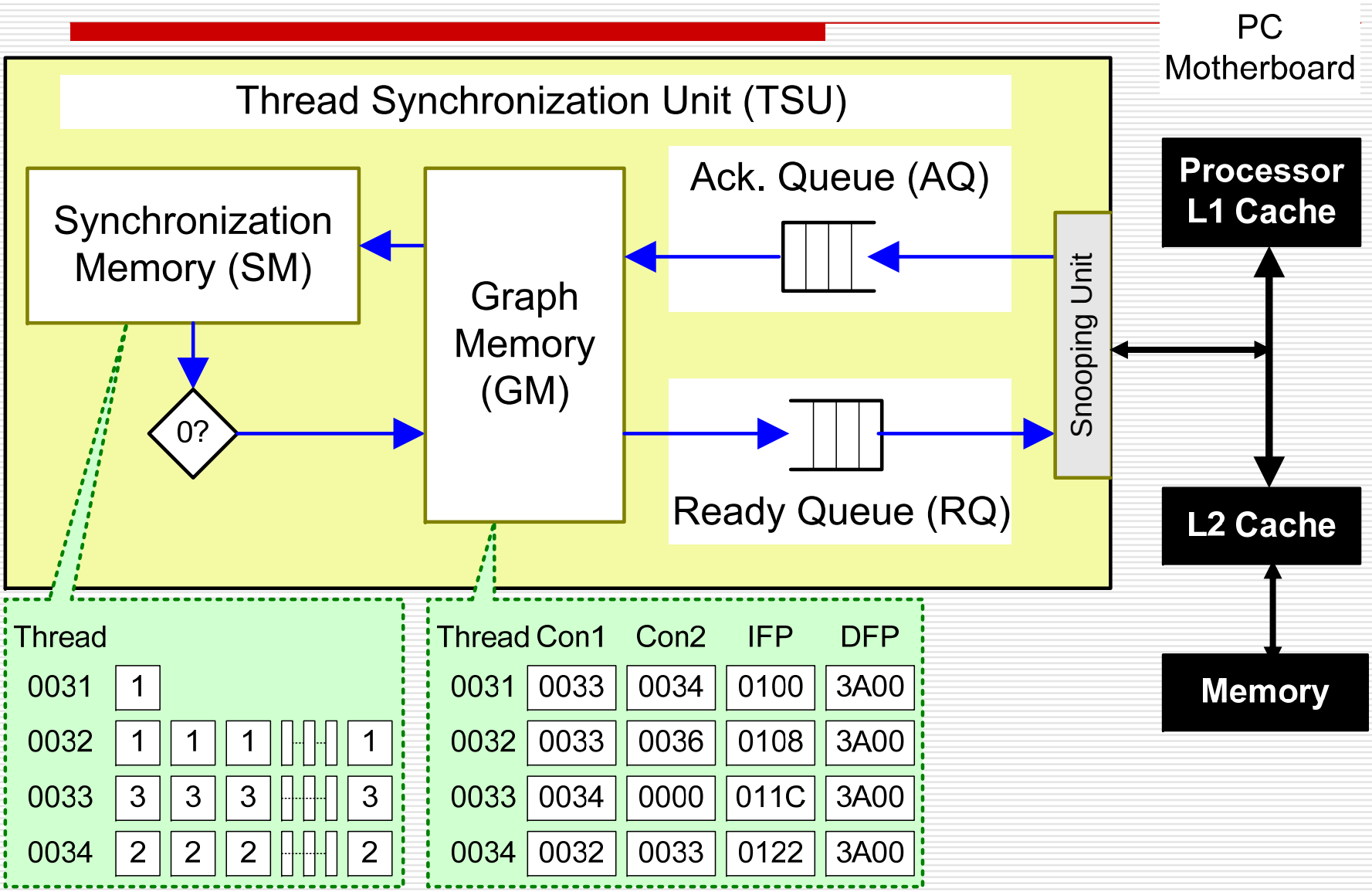
The AqIndex/RqIndex used by the TSU/CPU to provide the iteration index of the thread.

The TSU/Processor Interface

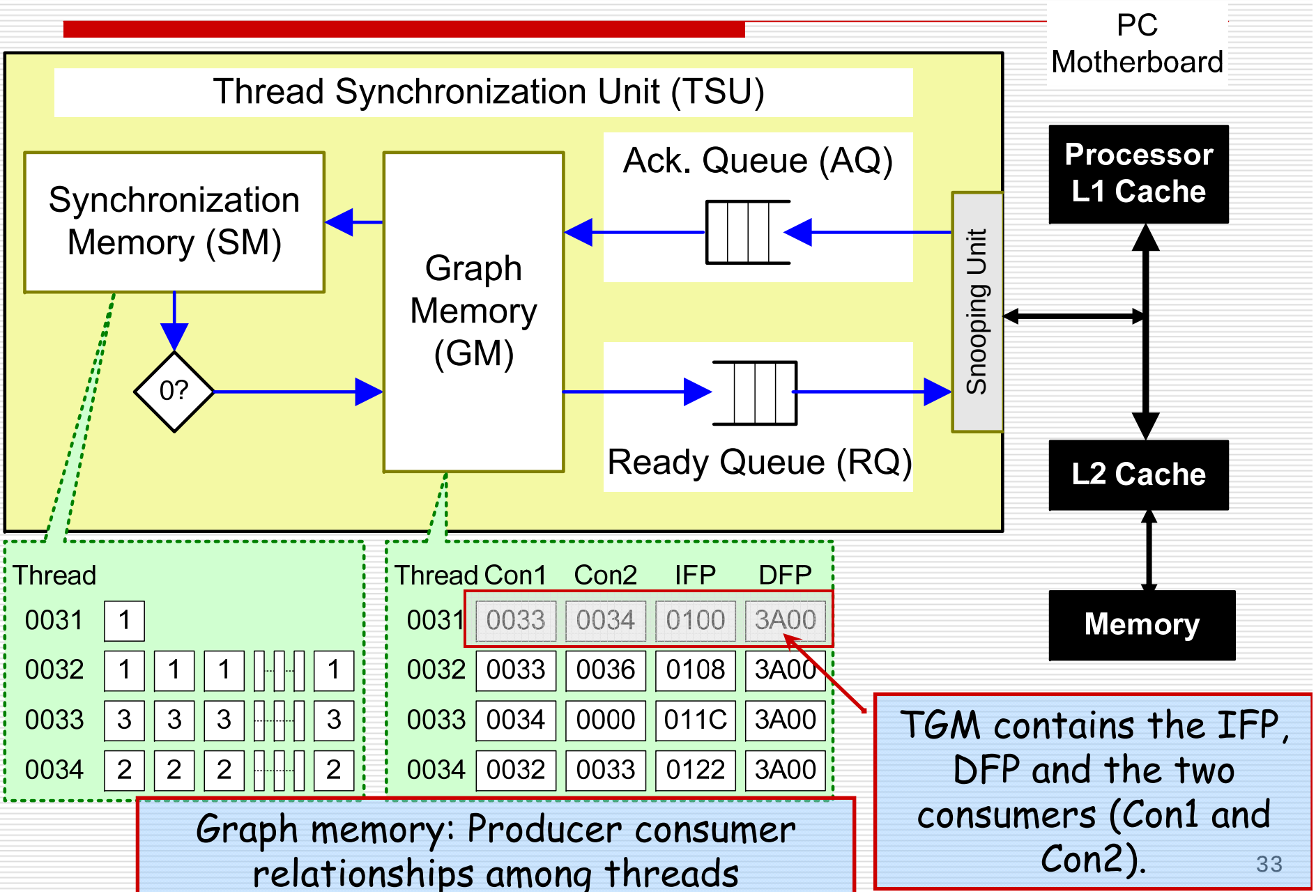


These addresses are intercepted by the Snooping Unit and forwarded to TSU for further processing.

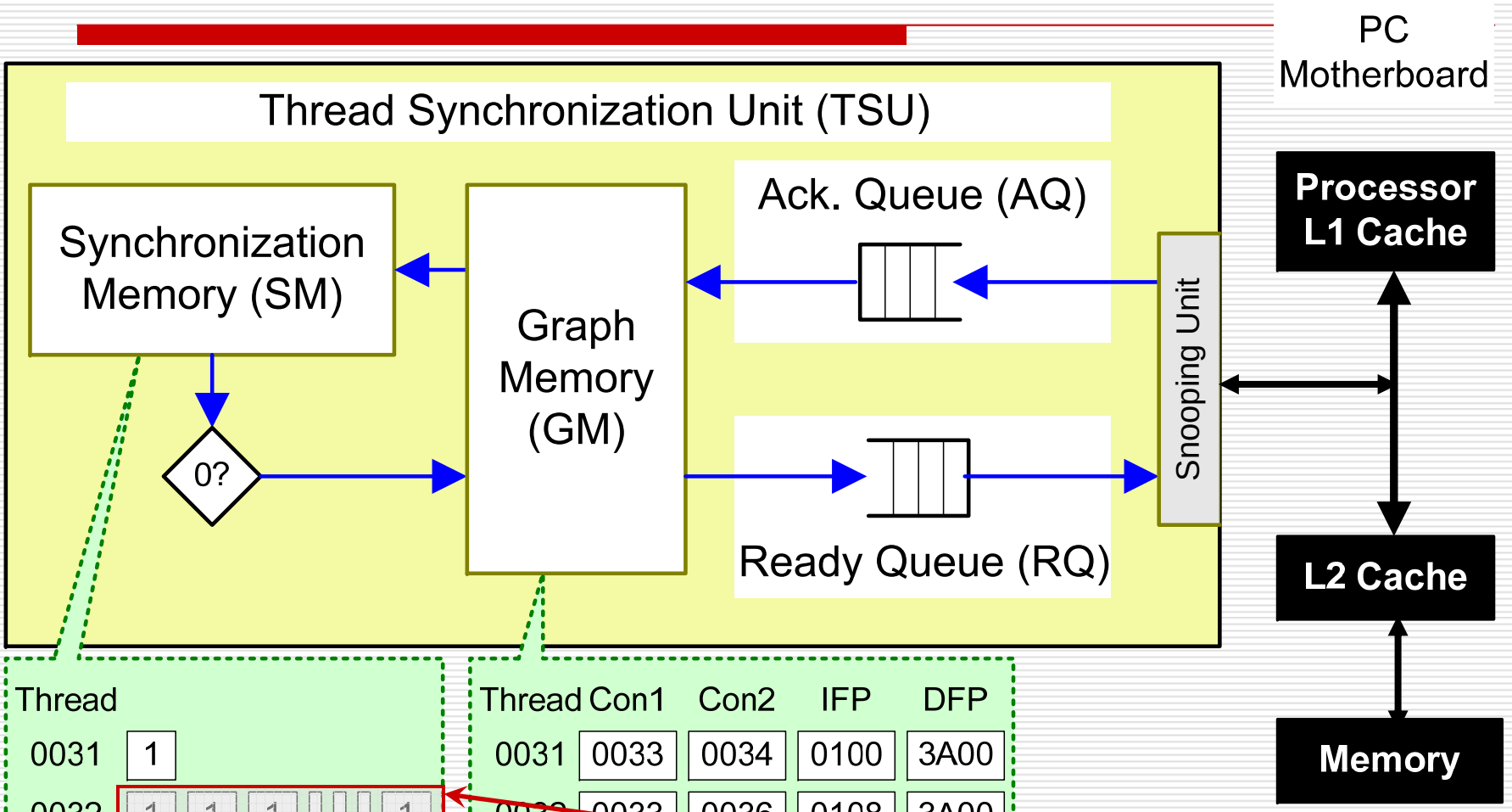
Data-Driven Multithreading Execution



Data-Driven Multithreading Execution



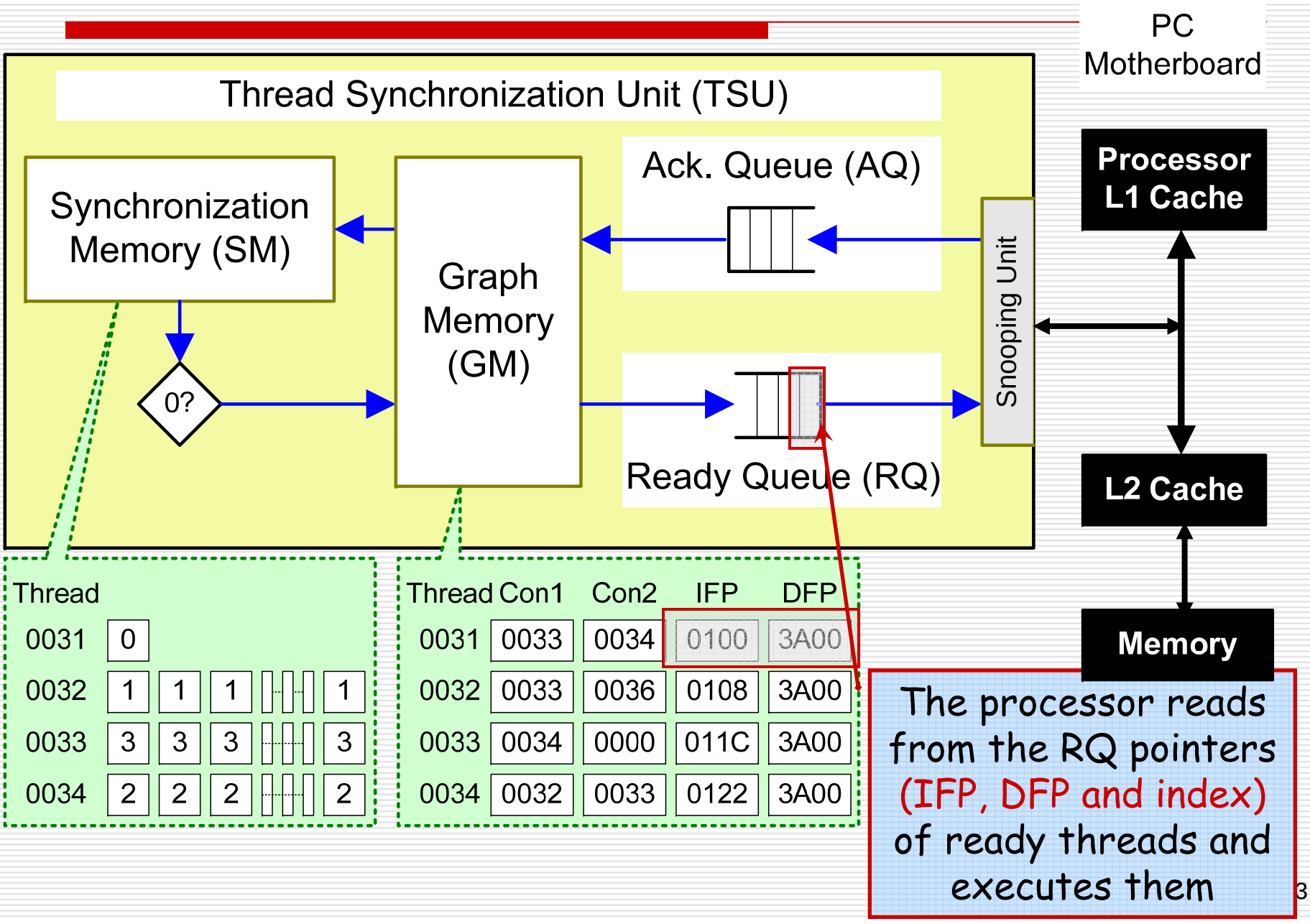
Data-Driven Multithreading Execution



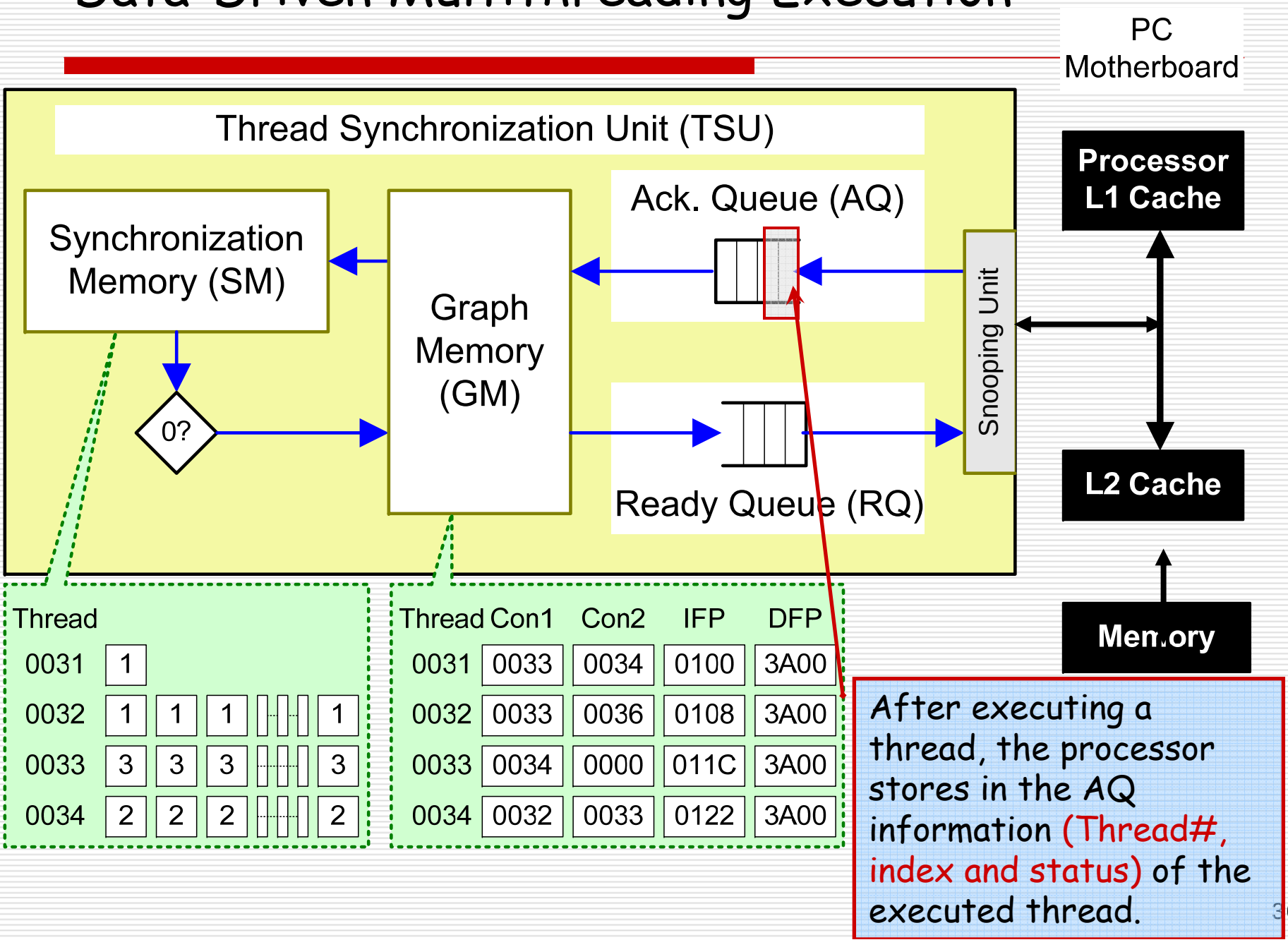
The SM contains the Ready Counts. One value for each loop iteration.

Ready count of a thread: number of producers

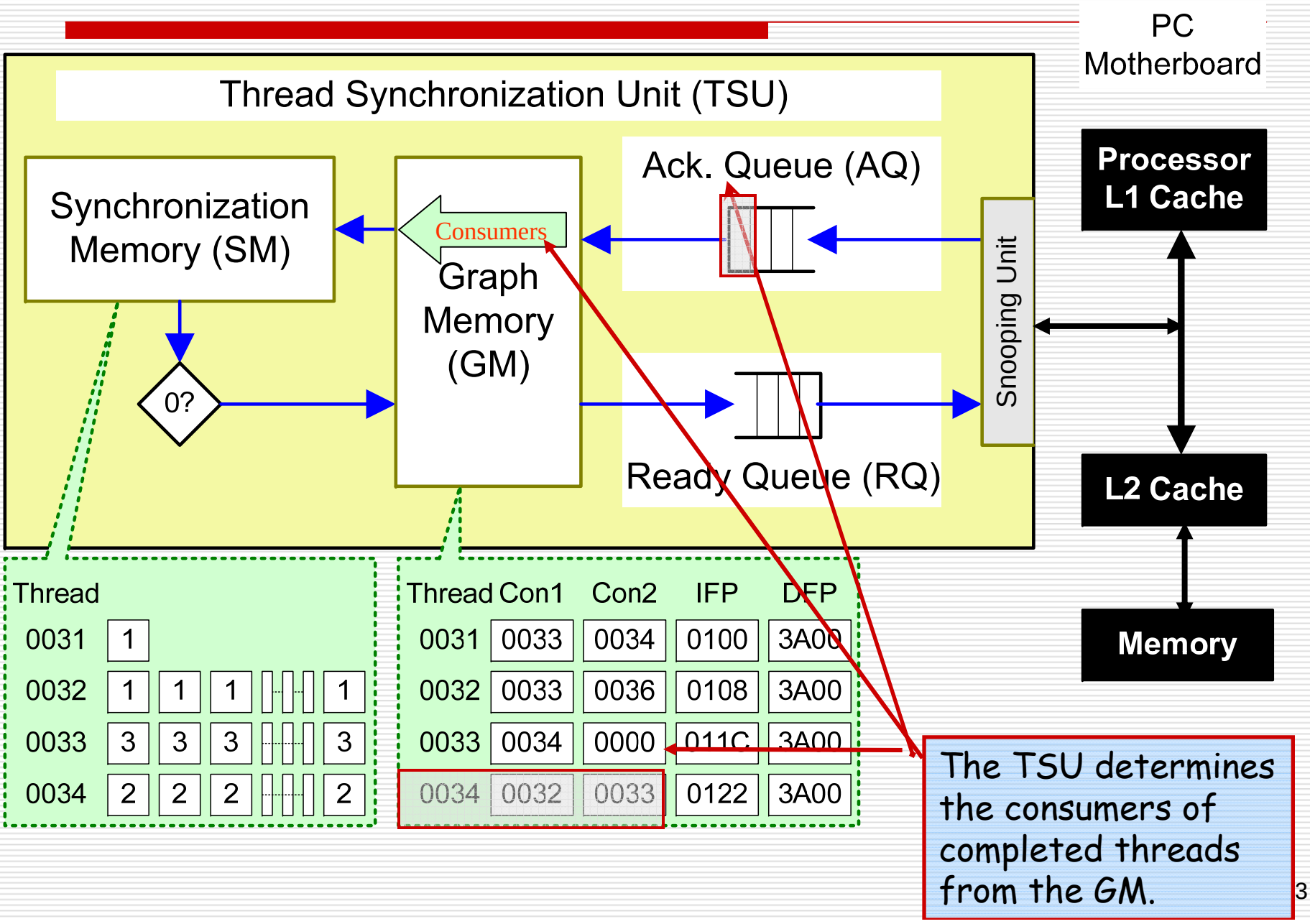
Data-Driven Multithreading Execution



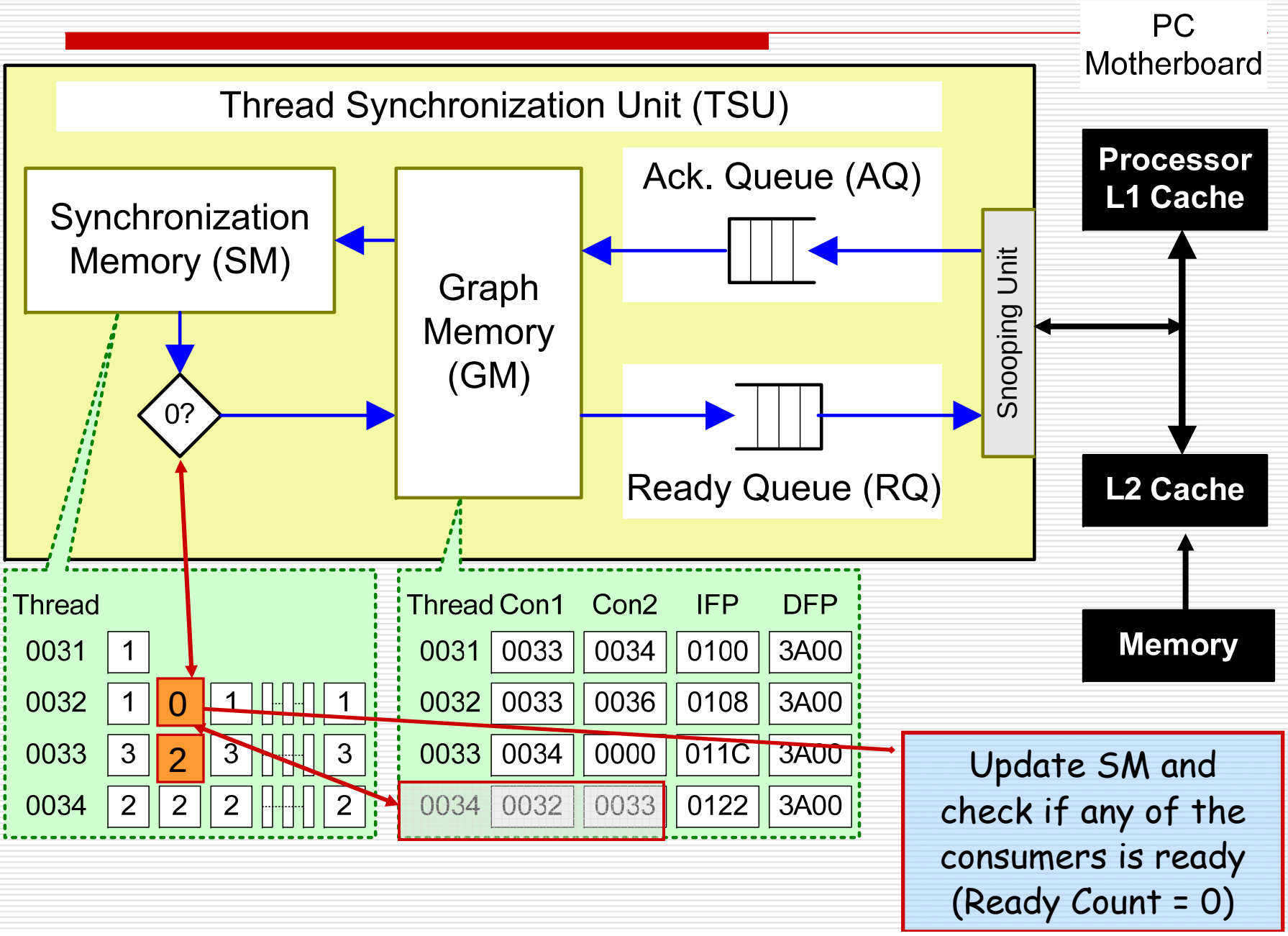
Data-Driven Multithreading Execution



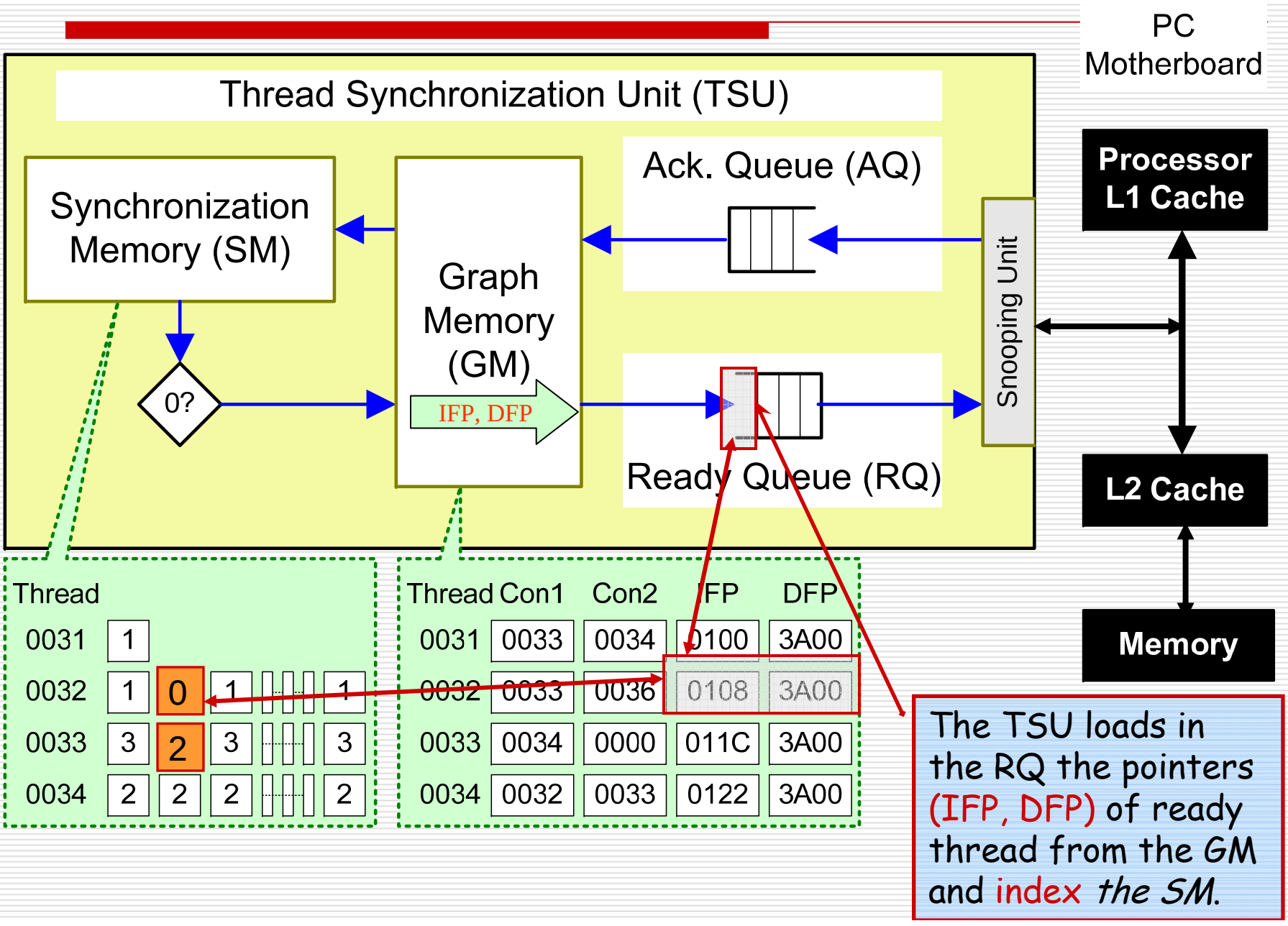
Data-Driven Multithreading Execution



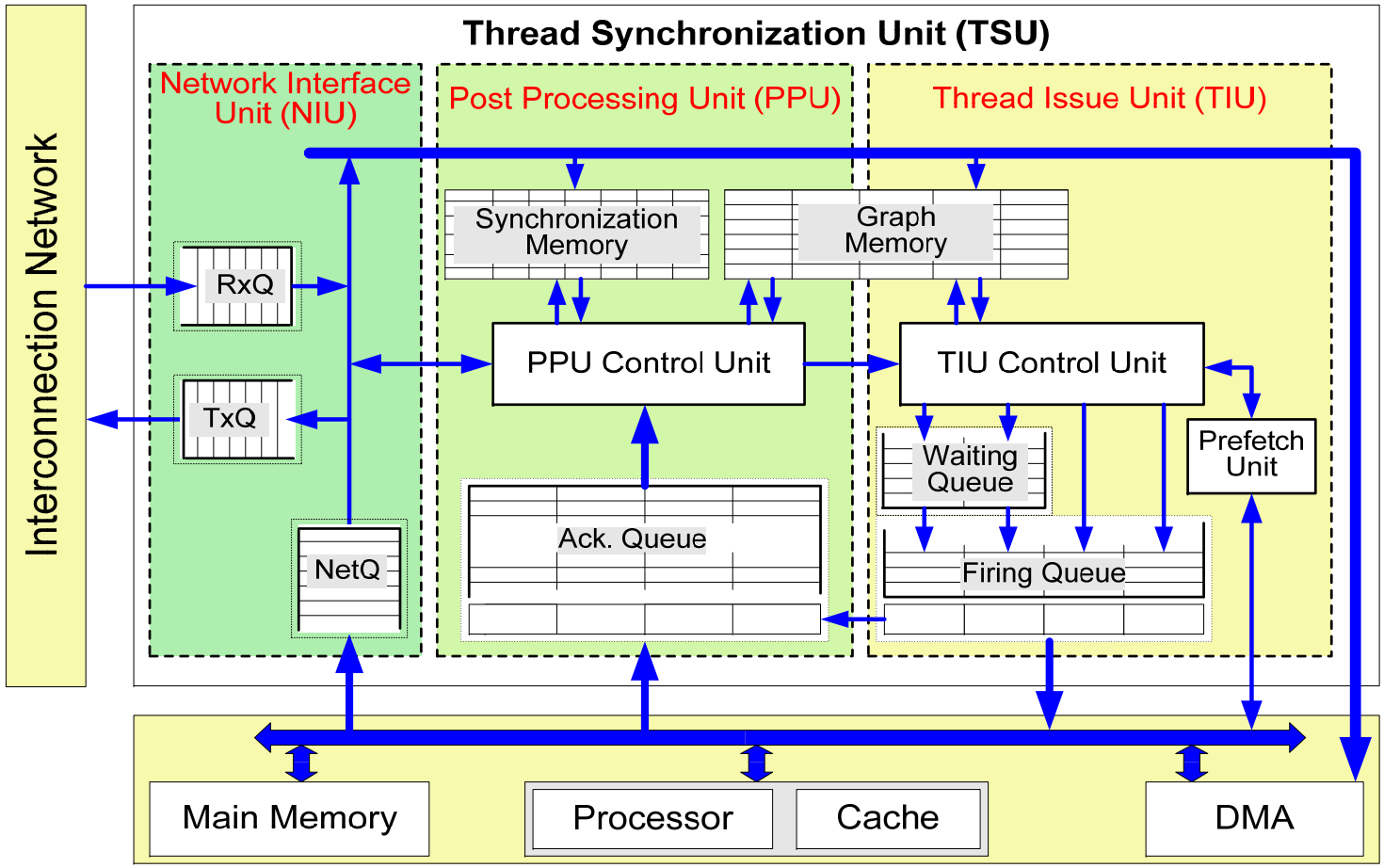
Data-Driven Multithreading Execution



Data-Driven Multithreading Execution

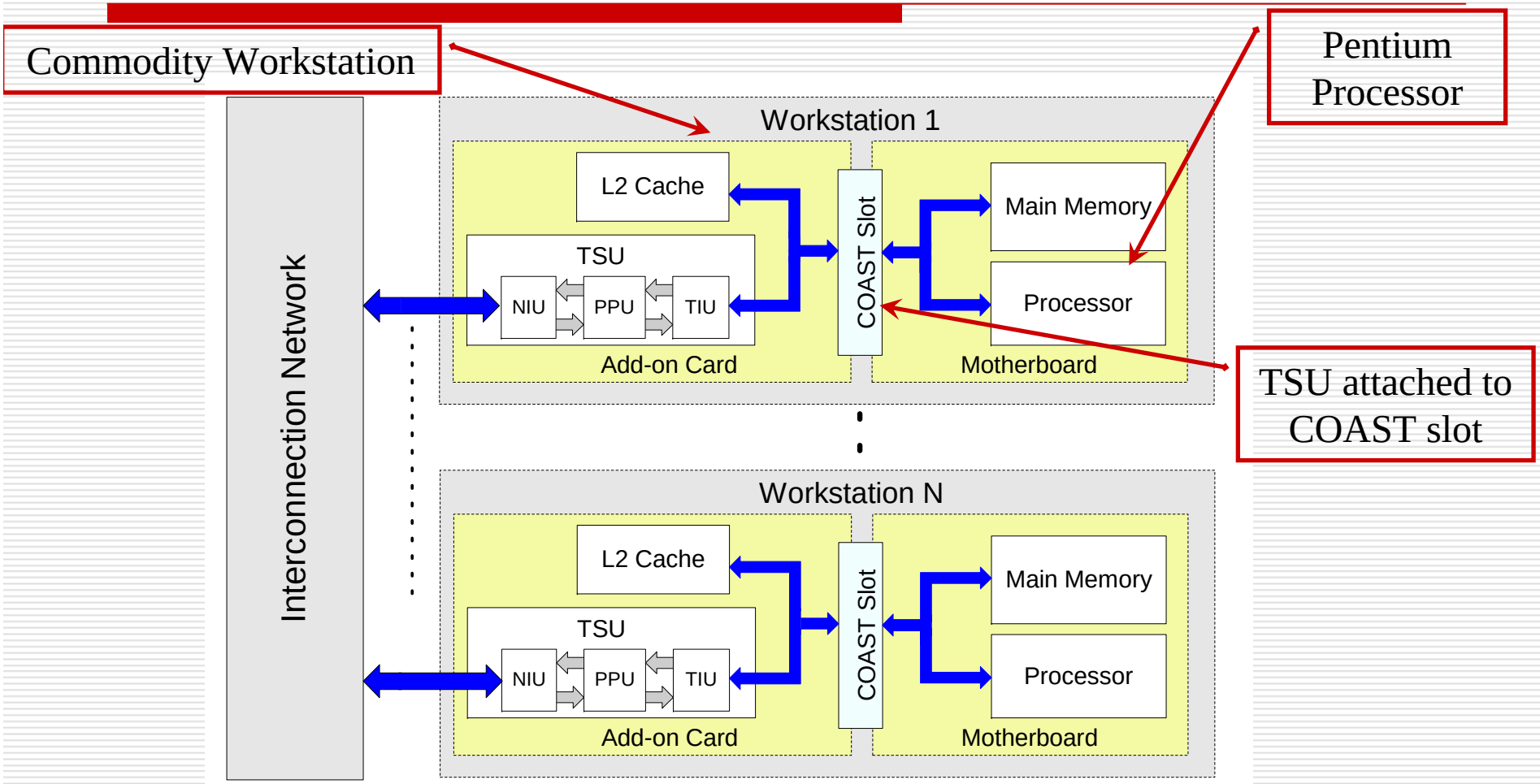


Thread Synchronization Unit (TSU)



- Three Units
  - Thread Issue Unit
  - Post Processing Unit
  - Network Interface Unit
- Units are Decoupled/asynchronous
- Communicate via Queues

D²NOW with Pentium Workstations (2006)

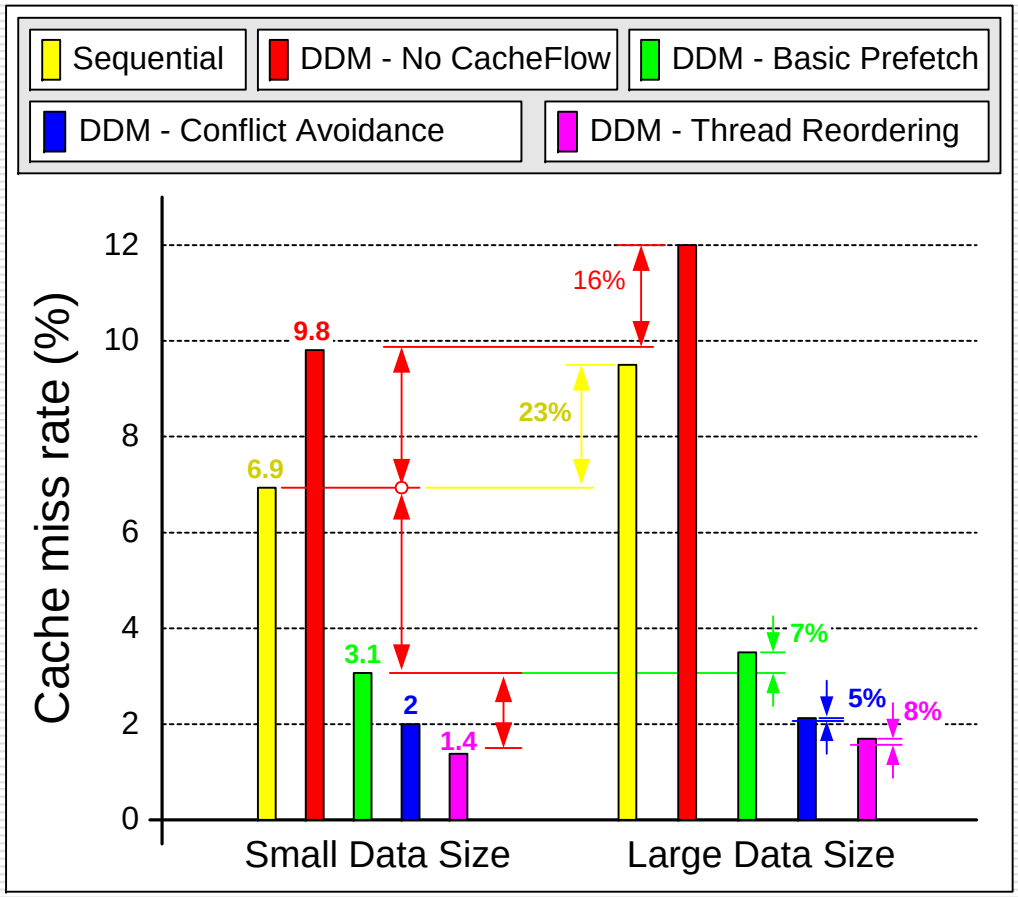


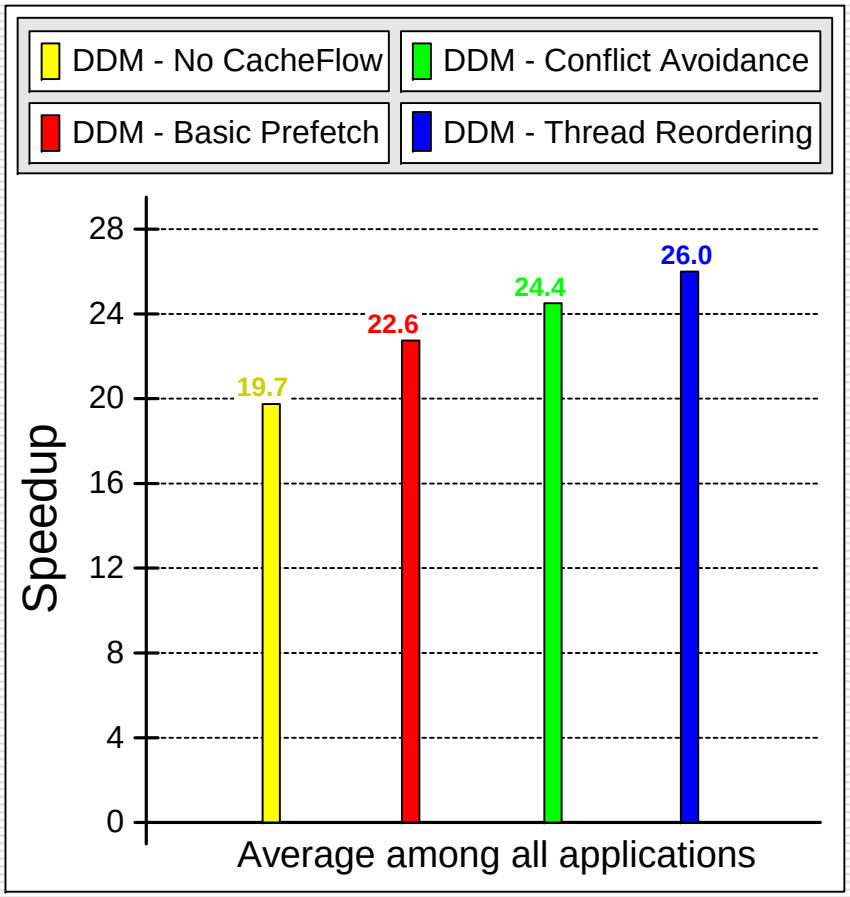
- TSU could go on the COAST slot
  - TSU and Cache with dual ported Tag bits
- TSU in the Co-processor slot, Use MESI instructions for cacheflow

CacheFlow: A Cache Management Policy for Data-Driven Multithreading

- Motivation
  - The Firing Queue of a DDM machine determines the order in which the threads are executed
  - Each Thread has a pointer to its Data
  - Future memory accesses are known!
- CacheFlow reduces cache misses by Prefetching blocks that are needed by the threads that enter the RQ (basic implementation)
  - Optimization 1: False Conflict avoidance: Not scheduling threads that could cause false cache conflicts
    - False cache conflicts: prefetching displaces data prefetch for other threads waiting in the FQ
  - Optimization 2: Thread Reordering Reordering the sequence of execution of ready threads to exploit locality.

Effect of CacheFlow on Cache Miss Rate and Speedup





- DDM increases miss rate
- Problem data size increases miss rate
- CacheFlow reduces miss rate (lower than the sequential)
- Minimal increase in miss rate when CacheFlow is employed
- Optimizations reduce further miss rate

- Significant speedup improvement for all applications
- Average speedup increased from 19.7 to 26.0

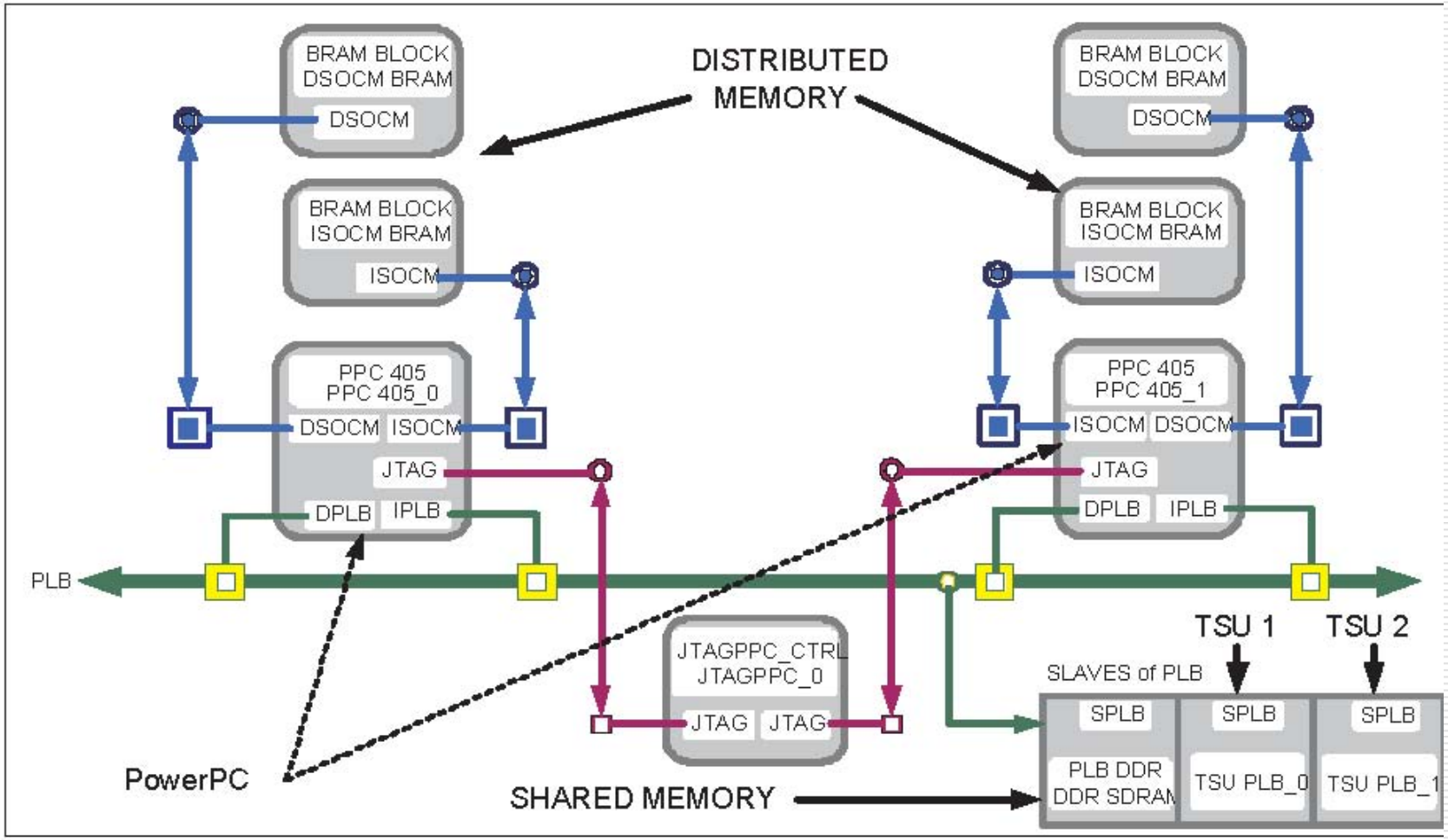
Experimental Results Summary

- Thread granularity: impact on
  - DDM overheads, locality, TSU latency, pipeline performance
  - Increasing from 1 to 8 increases performance by 20%
    - **12.8 to 16.8 (on 32-node system)**
- Communication assist optimizations: impact on
  - CPU communication overheads
  - Lead to 22% increase in speeedup (19.7 speedup 32-nodes)
  - Increase in communication latency by 500%
    - **in 13.4% (2.8-23%) average speedup reduction**
- But DDM could destroy locality and increase cache misses
- CacheFlow: hardware prefetching to
  - Completely eliminates extra misses due to DDM
    - **Serial: 6.9** **DDM: 9.8** **DDM w Cacheflow: 1.4**
      - further reduces cache misses
  - Overall speedup increased from **19.7 to 26.0 (32-nodes)**

TFux (Thread Flux) 2nd DDM implementation 2008

- TFlux developed a complete platform:
  - definition of DDM compiler directives,
  - a preprocessor tool to generate source code that includes the application as well
  - Kernel that provides runtime support and scheduling code,
- Enables compatibility of DDM codes on a variety of commodity multi-core systems (x86, Sparc, possibly on anything that yuns Linux and supports C ☺)
- Hardware simulations using Simics.

FPGA based implementation of DDM (Xilinx VirtexPro II) 2008



DDM-VM
(Data-Driven Multithreading Virtual Machine), 3rd DDM impl.

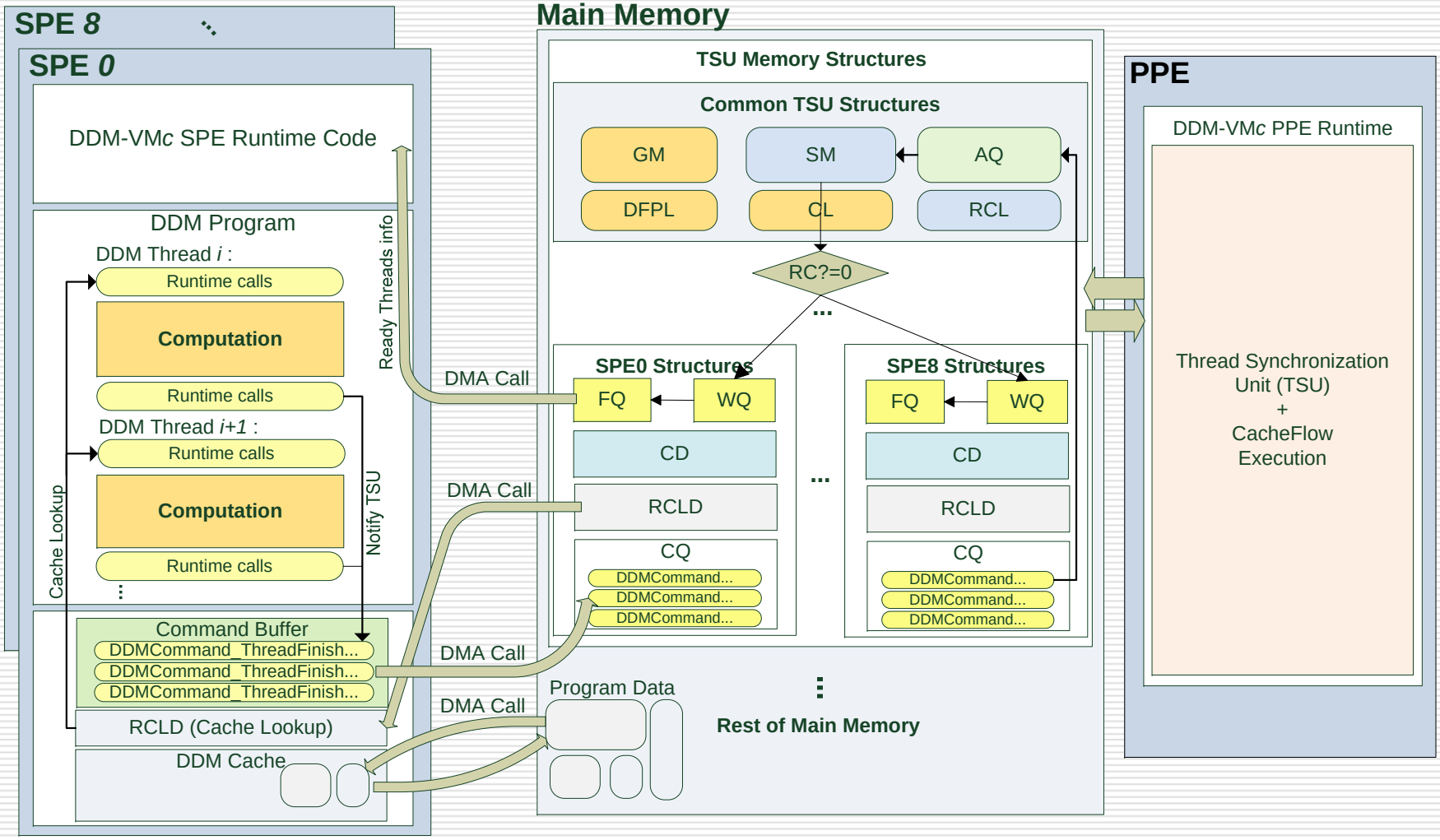
- DDM-VM virtualizes DDM execution across heterogeneous and homogeneous multi-core system including distributed ones.
- DDM-VMc, is the branch of DDM-VM targeting high-performance heterogeneous multi-cores.
  - The Cell is a representative example of such systems.
- DDM-VMs, Symmetric multi-core systems

- Allow us to compare with ("similar") state of the art parallel processing approaches/systems

The DDM-VMc

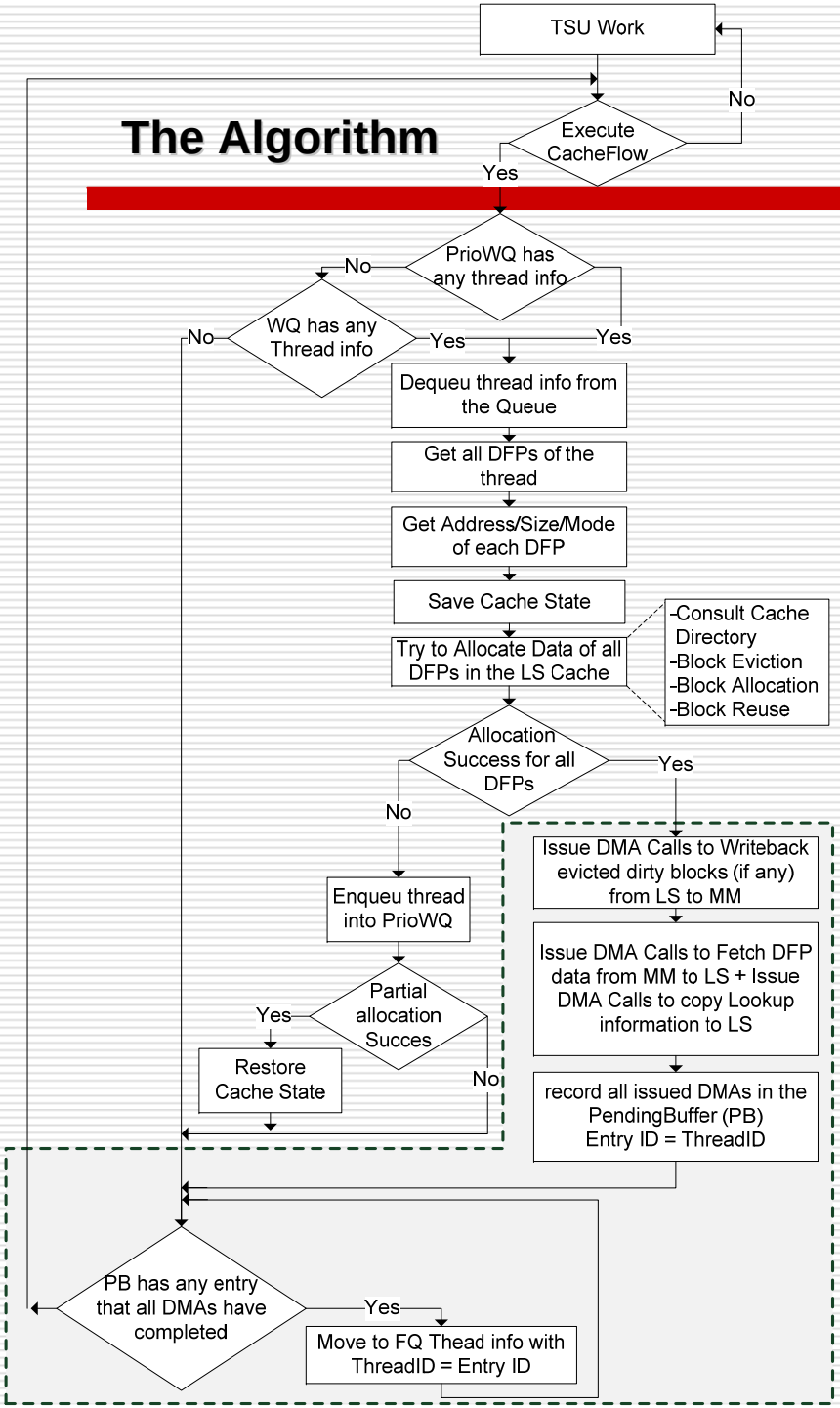
- The Cell provides a high a computational power on a single chip (204 GFLOPs) but programming it is not a trivial process
- Utilizing the DDM model of execution DDM-VMc leverages the latency tolerance and distributed concurrency of the Dataflow model with the efficient execution of the sequential model to program the Cell processor:
  - It virtualizes the parallel resources of the Cell and the low-level details of memory management, scheduling, synchronization and execution instantiation
  - It schedules threads dynamically at run-time and manages the memory hierarchy transparently using CacheFlow
  - It interleaves the scheduling of threads & management data with the execution of the threads, thus tolerating memory & synch. latencies

The DDM-VMc Architecture



CacheFlow on the Cell

The Algorithm



- CacheFlow an automated and efficient memory management for the Cell
- A portion of the LS of each SPE is pre-allocated and divided into cache blocks.
- A Cache Directory (CD) keeps track of the blocks state.
- CacheFlow maintains consistency with DF synchronization
- Move data in before starts execution and write back after finishing
- Exploits explicit locality

Speedup comparison of DDV-VMc vs CELLSs (BSC) and Sequoia (Stanford) (1 to 6 SPE on Sony Playstation)

DDM-VMc vs CEllSs for Matrix Multiplication

Fine grain (512x512)	Medium Grain (1024x1024)	Coarser grain (2048x2048)
31-149%	22-35%	20-24%

DDM-VMc vs CEllSs for Cholesky

Fine grain (512x512)	Medium Grain (1024x1024)	Coarser grain (2048x2048)
60-455%	34-174%	19-26%

DDM-VMc vs Sequoia Matrix Multiplication

Fine grain (512x512)	Medium Grain (1024x1024)	Coarser grain (2048x2048)
67-93%	90-122%	93-113%

DDM-VMc vs Sequoia for ConV2D

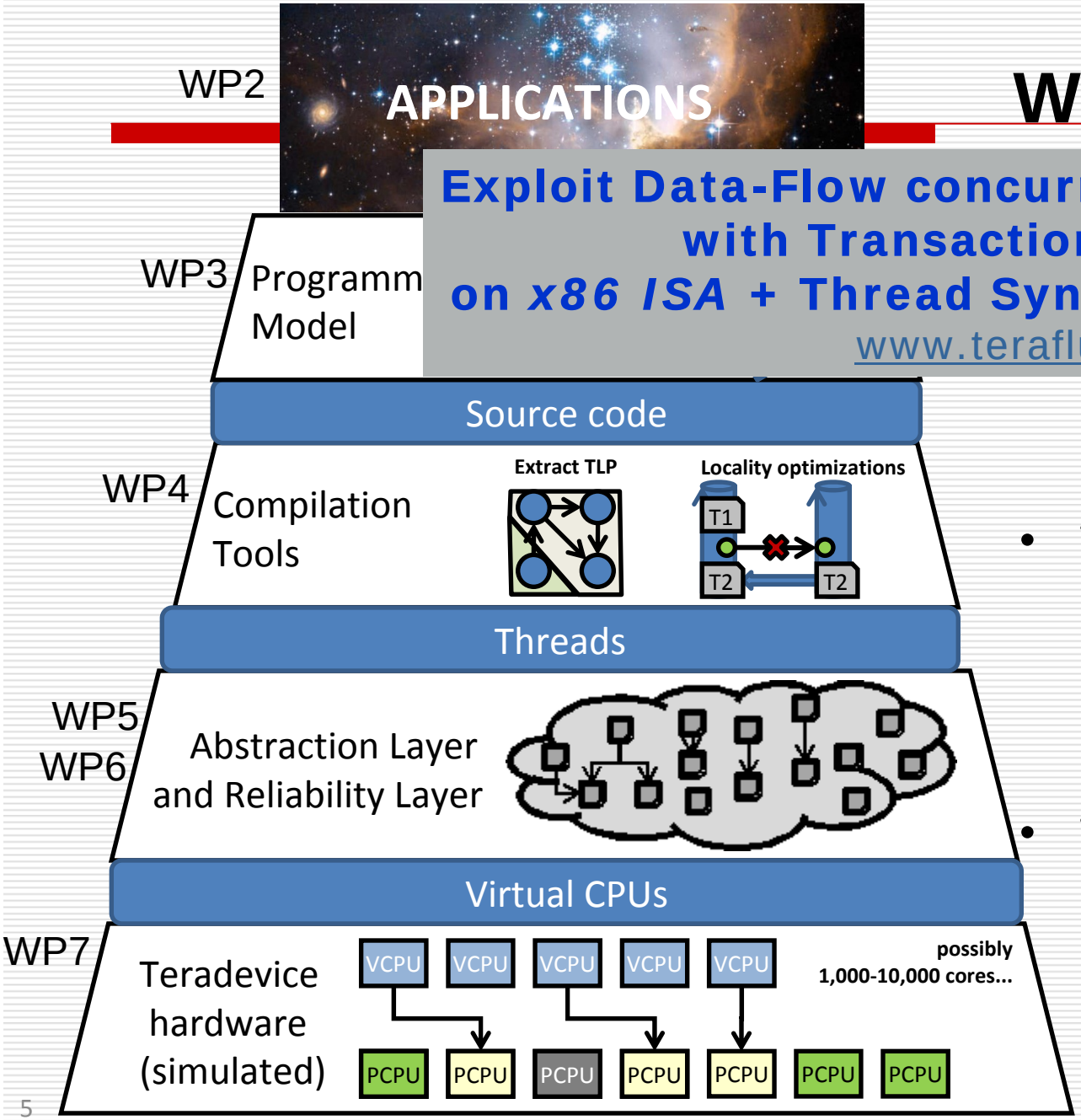
Fine grain (512x512)	Medium Grain (1024x1024)	Coarser grain (2048x2048)
16-70%	17-37%	20-41%

TERAFLUX

What is about

Exploit Data-Flow concurrency and combine it with Transactional Memory on *x86 ISA* + Thread Synchronization Unit. …

www.teraflux.eu



complexity of design, reliability

- TERAFLUX context:
  - High performance computing and applications (not necessarily embedded)
- TERAFLUX scope:
  - Exploiting a less exploited path (DATA-FLOW) at each level of abstraction

Our proposal B2B plus DF support at the Thread level

- Back to Basic:
  - Just a simple Fast execution Pipeline
- Hardware support for Data-Driven Threaded execution
- Deterministic Pre-fetching into the "cache"