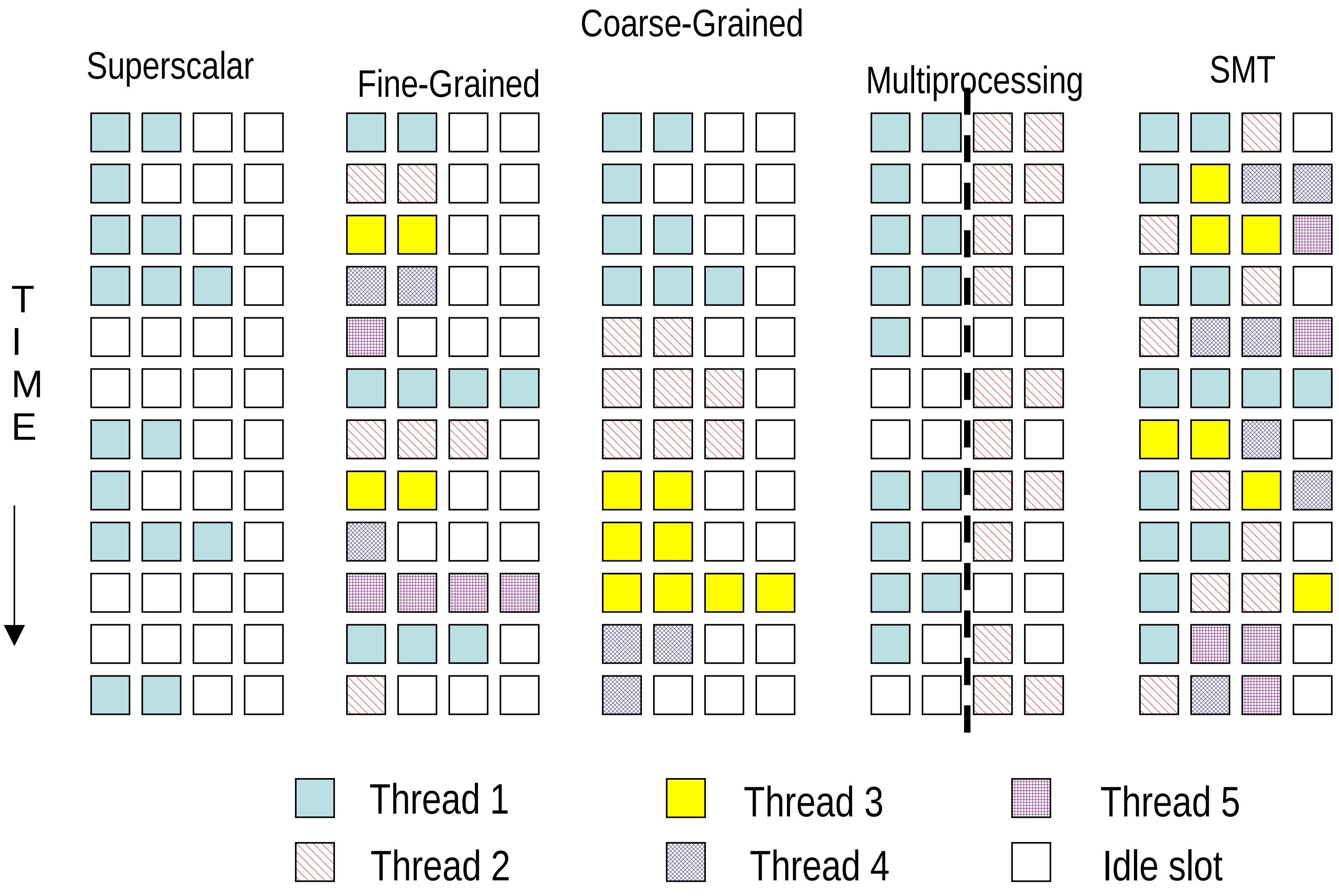
SMT:
Simultaneous Multi-threading

Computer System Architecture
IIT Tirupati
jtt@iittp.ac.in
April, 2020

Multi-threaded Exec Model



Architecture Research

- **Concept** & Potential of Simultaneous Multithreading:
  (ISCA ’95 & ISCA 25th Anniversary Anthology)
- Designing the **microarchitecture**: ISCA ’96
  - straightforward extension of out-of-order superscalars
- I-fetch **thread chooser**: ISCA ’96
  - 40% faster than round-robin
- Software-directed **register deallocation**: TPDS ’99
  - large register-file performance vs. small register file
- **Mini-threads**: HPCA ’03
  - large SMT performance vs. small SMTs
- SMT instruction **speculation**: TOCS ‘03
  - a good thread mix is the most important performance factor

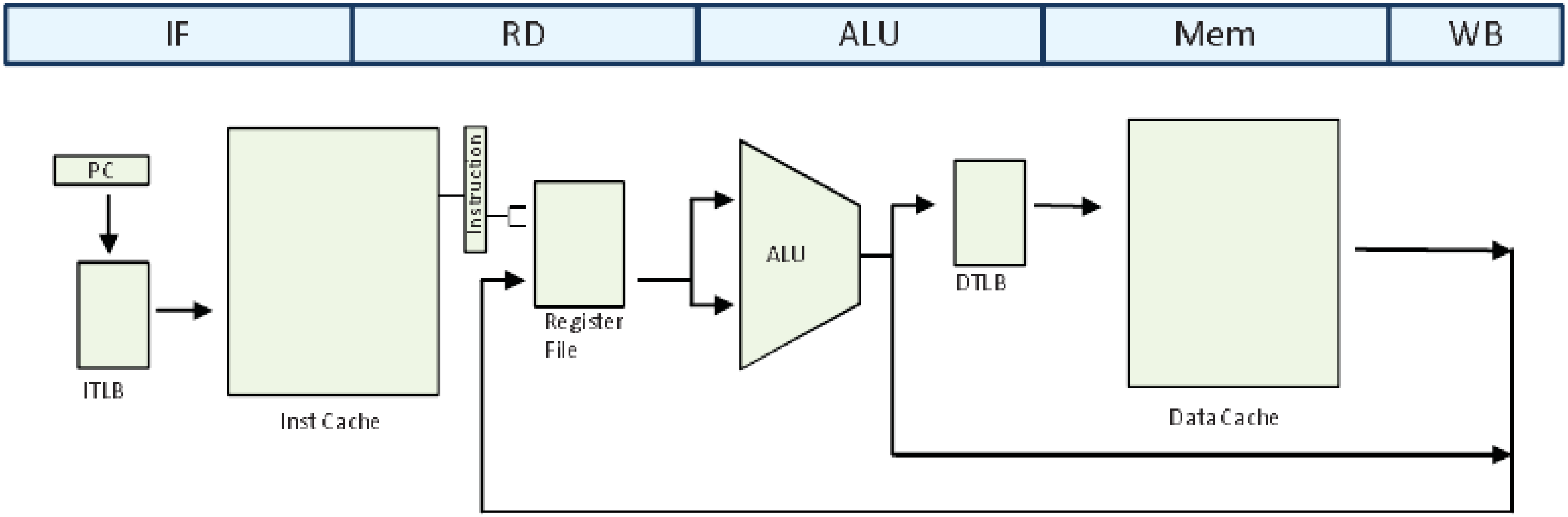
Design Challenges in SMT

- Impact of fine-grained scheduling on single thread performance? (Since SMT makes sense only with fine-grained implementation)
- Larger register file is required to hold multiple contexts
- Challenge in not affecting the clock cycle time, especially in
  - Instruction issue - more candidate instructions need to be considered
  - Instruction completion - choosing which instructions to commit may be challenging
- Ensuring that cache and TLB conflicts generated by SMT does not degrade performance

[Managing large pool of threads is the challenge!]

Implementation of CGMT

A simple pipeline architecture without multi-threading support



32 32-bit GPRs
32 co-processor zero CP0 registers for OS related work
3 special registers: PC, Hi and Lo

How to make it multi-threaded (4 threads coarse-grained)?

Implementation of CGMT

How to make it multi-threaded (4 threads coarse-grained)?

Replicate GPR0 to GPR31: GPR0 to GPR31 for **T0**, GPR0 – GPR31 for **T1**…..

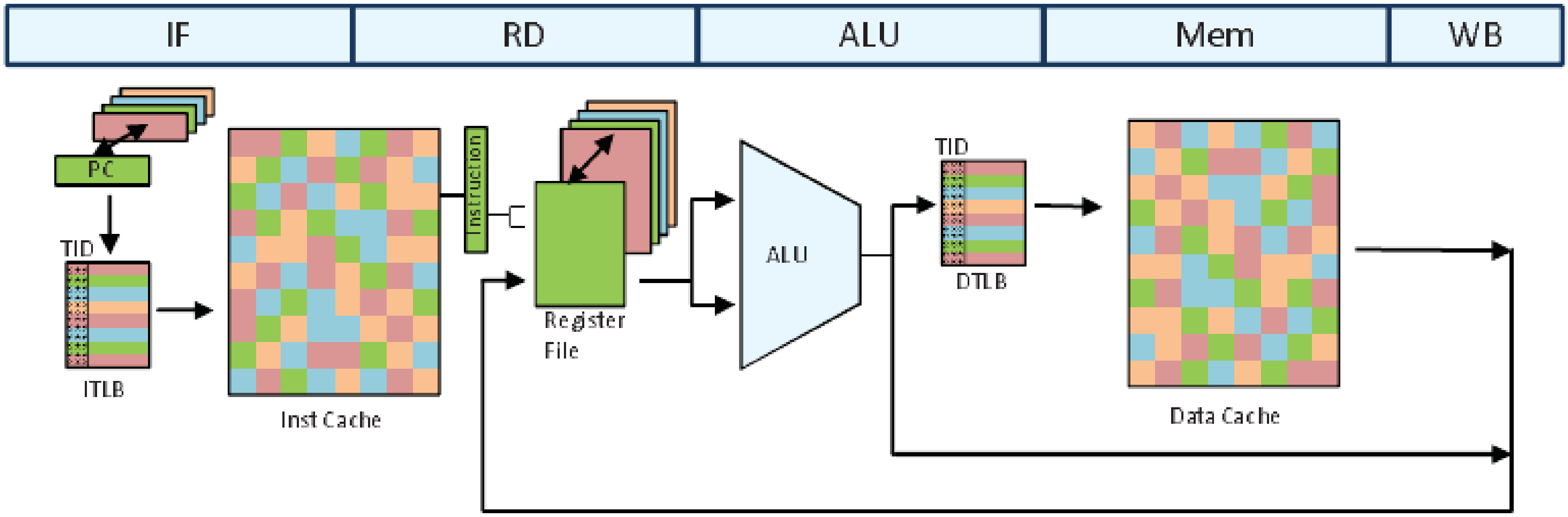
Replicate Hi, Lo and PC:

In case of PC, PC0 – PC3 would get access via active PC.

Thread ID register: A new 2 bit register to store Hardware Thread ID (HTID)

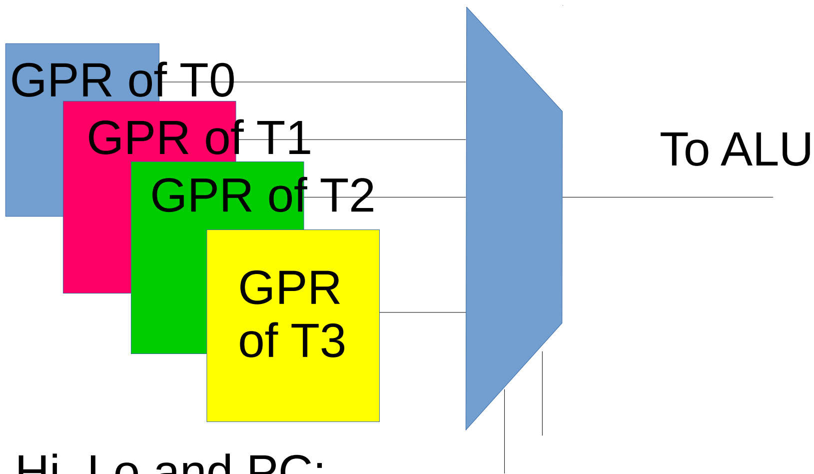
Need additional registers for context switch:

Valid Vector (VV) and Waiting vector(WV)



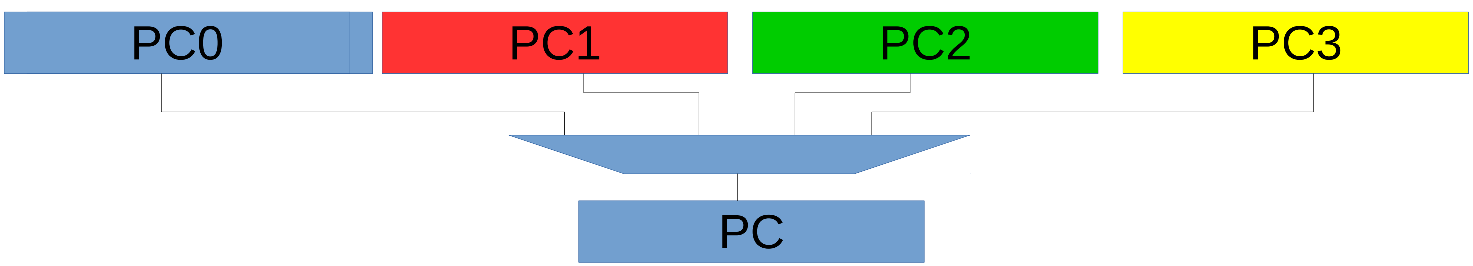
Implementation of CGMT

Replicate GPR0 to GPR31:



Replicate Hi, Lo and PC:

In case of PC, PC0 – PC3 would get access via active PC.



Thread ID register: A new 2 bit register to store Hardware Thread ID (HTID)

Need additional registers for context switch:

Valid Vector (VV) and Waiting vector(WV)

Implementation of CGMT

Thread ID register: A new 2 bit register to store Hardware Thread ID (HTID)

HTID

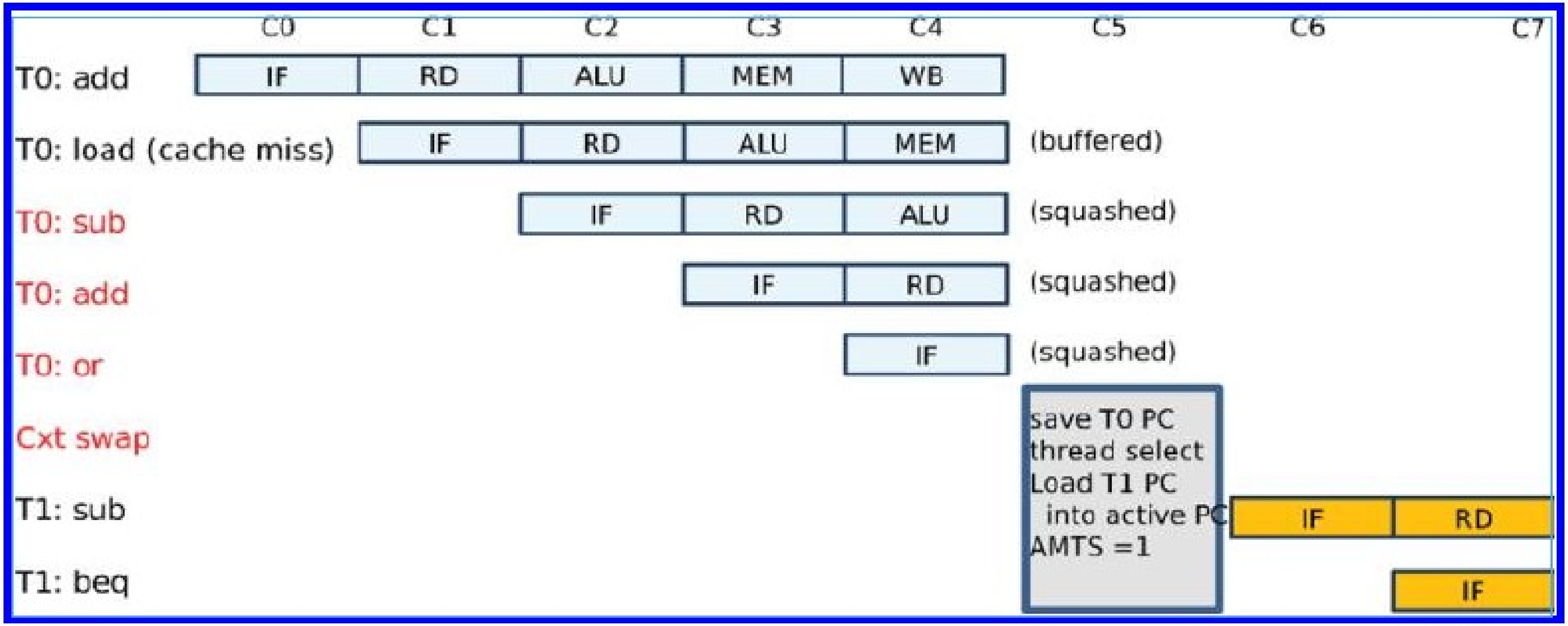
0 0 → Thread 0
0 1 → Thread 1
1 0 → Thread 2
1 1 → Thread 3

Need additional registers for context switch:
Valid Vector (VV) and Waiting vector(WV)

V V	0	1	1	0
W V	0	1	0	0

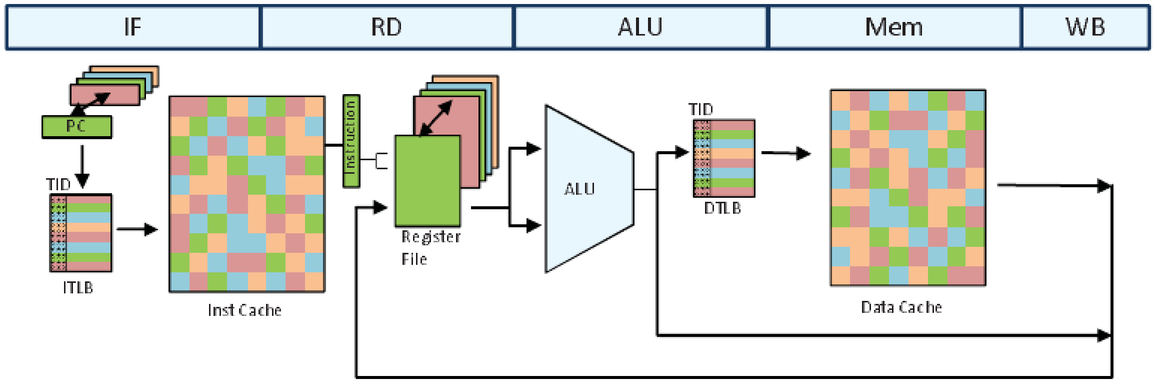
The Operation of CGMT

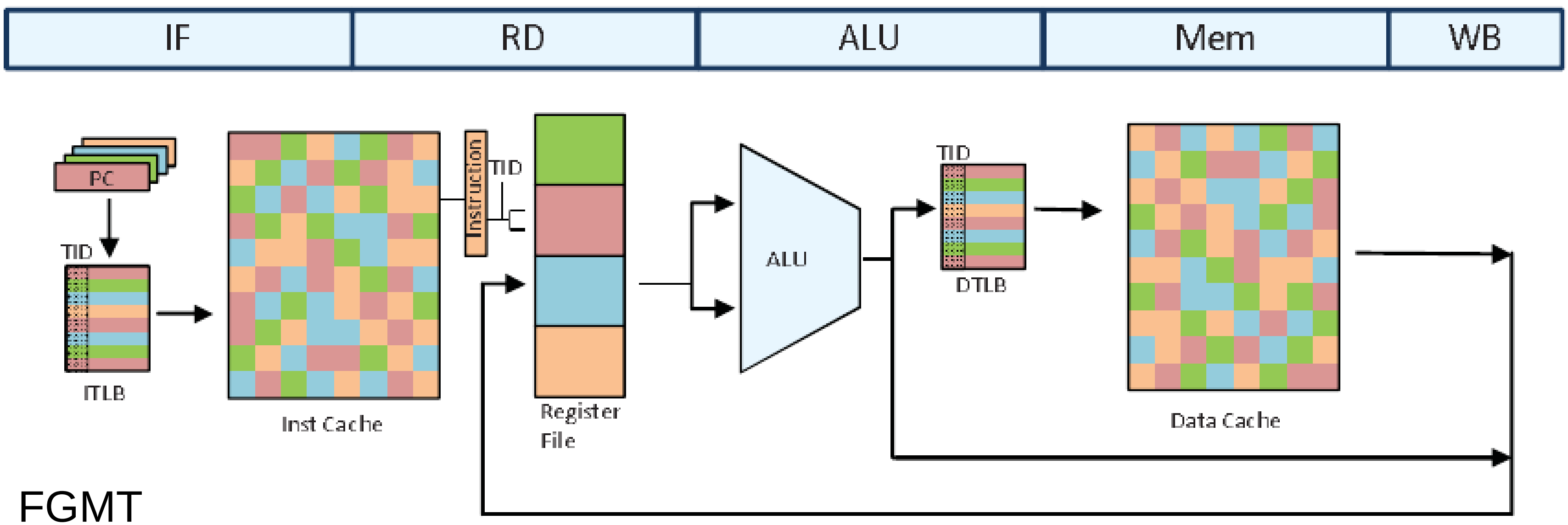
Execution map of of two threads with context switching.
Five stage pipeline with CGMT
The reason for context switch is Cache miss



Implementation of FGMT

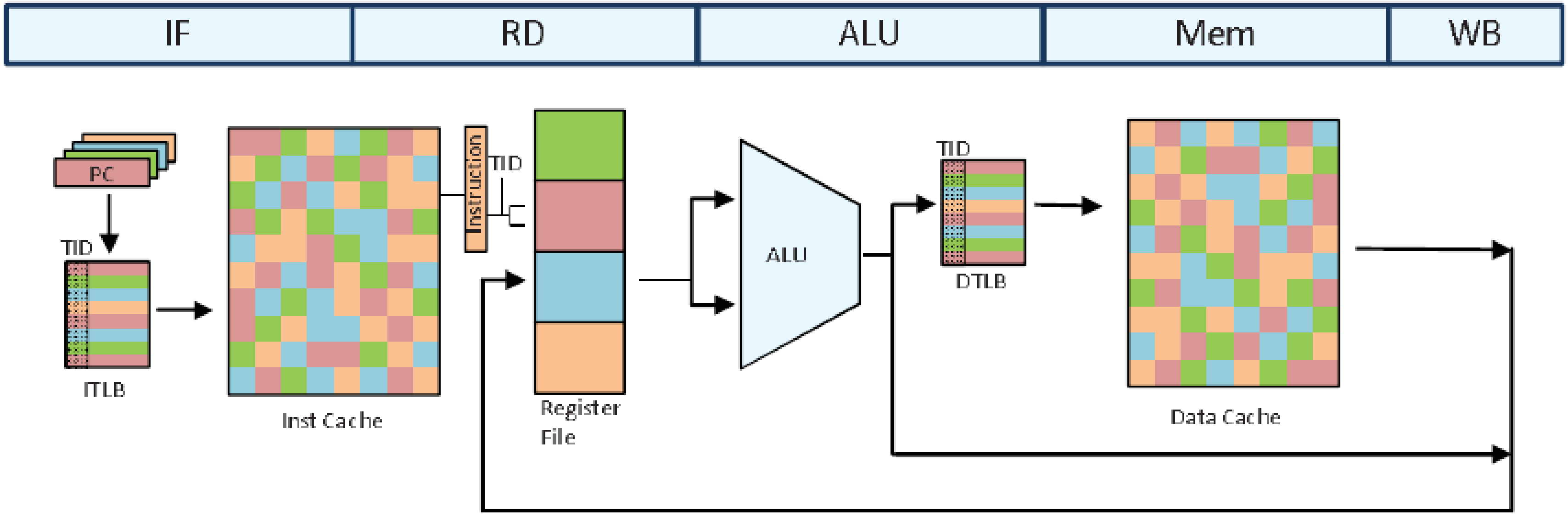
CGMT





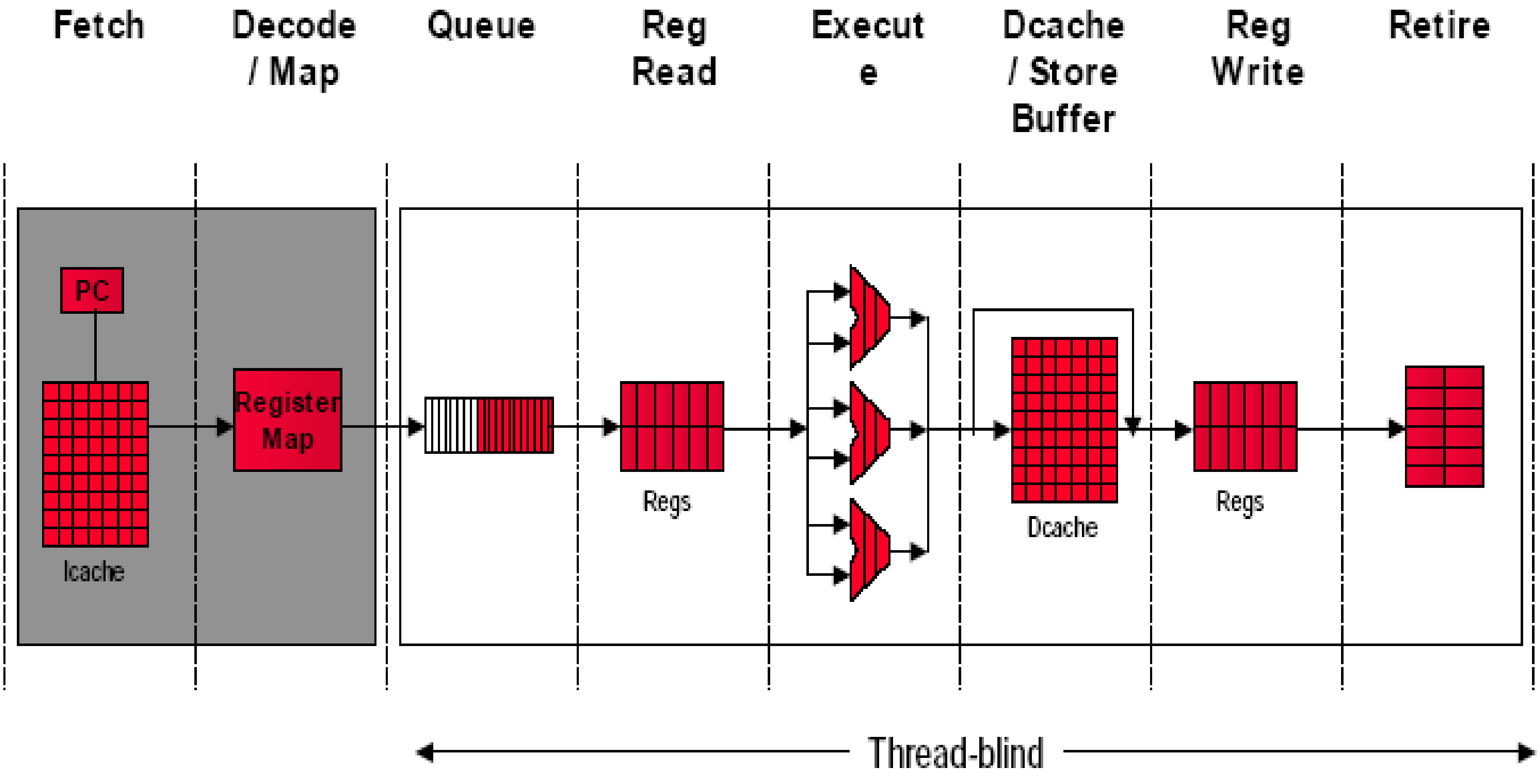
FGMT

Implementation of FGMT

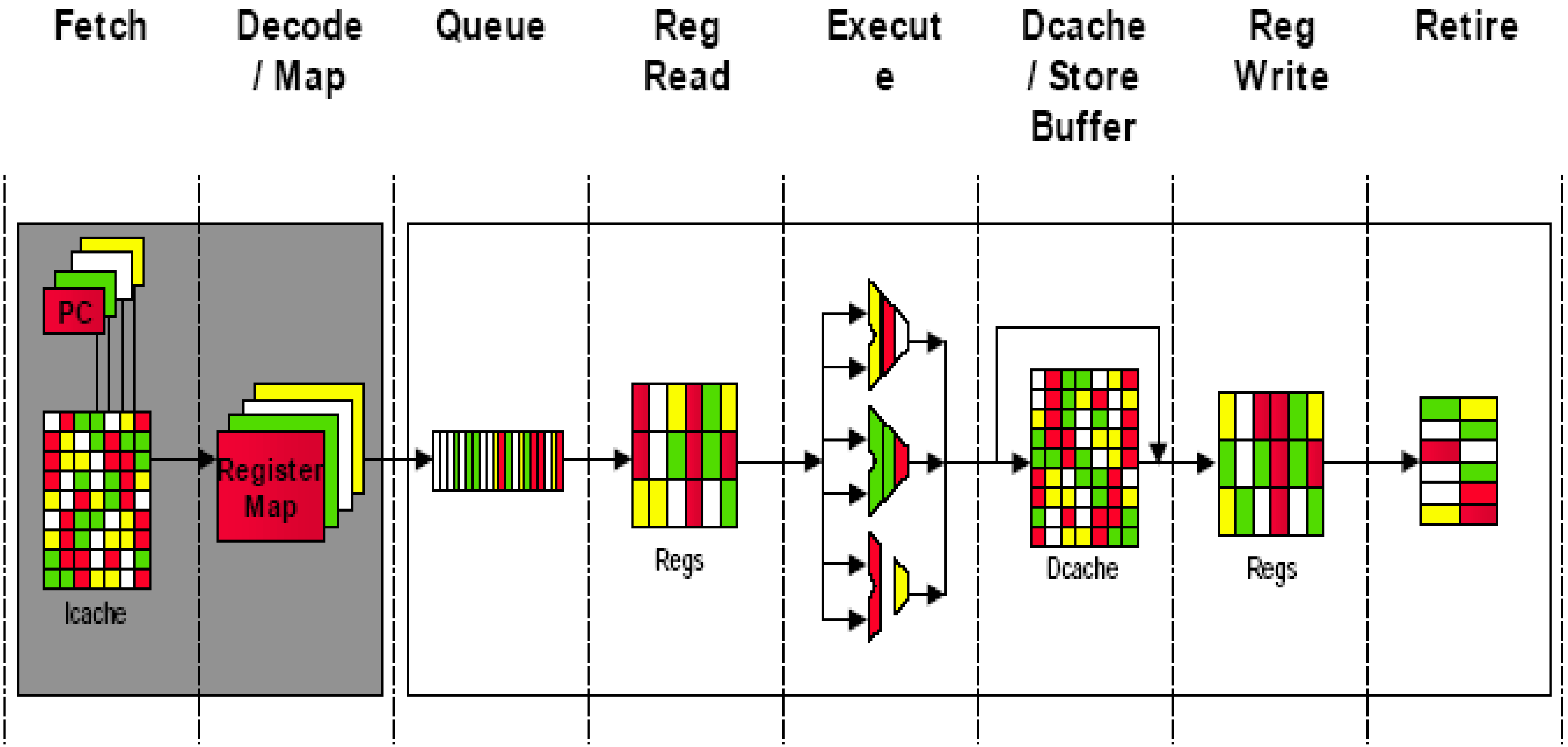


Instruction are fetched from each PC and TID will be appended to it.
The register file is expanded to accommodated all the threads.

Basic Out-of-order Pipeline



SMT Pipeline



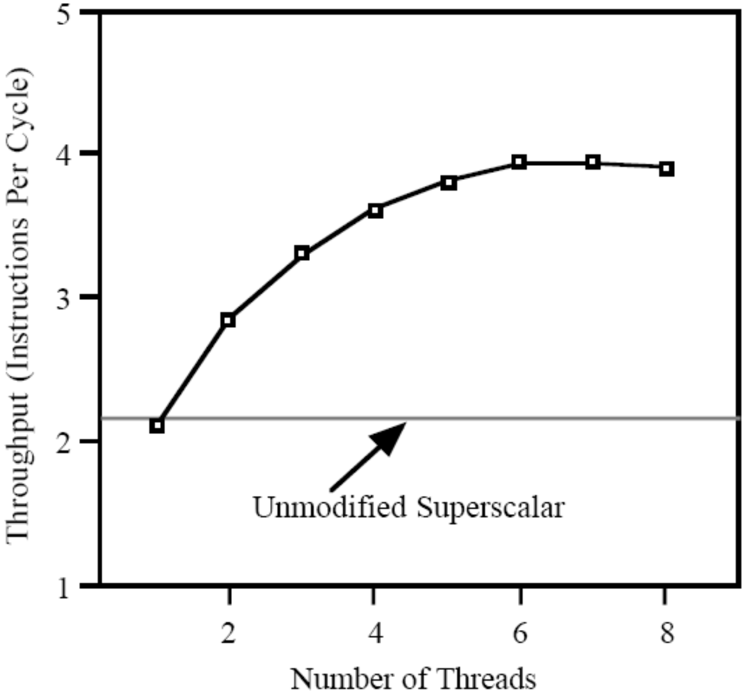
Implementing SMT

Can use as is most hardware on current out-or-order processors

Out-of-order renaming & instruction scheduling mechanisms

- physical register pool model
- renaming hardware eliminates false dependences both within a thread (just like a superscalar) & between threads
- map thread-specific architectural registers onto a pool of thread-independent physical registers
- operands are thereafter called by their physical names
- an instruction is issued when its operands become available & a functional unit is free
- instruction scheduler need not consider thread IDs when dispatching instructions to functional units
  (unless threads have different priorities)

SMT Performance



Metric	Number of Threads 1	4	8
out-of-registers (% of cycles)	3%	7%	3%
I cache miss rate	2.5%	7.8%	14.1%
-misses per thousand instructions	6	17	29
D cache miss rate	3.1%	6.5%	11.3%
-misses per thousand instructions	12	25	43
L2 cache miss rate	17.6%	15.0%	12.5%
-misses per thousand instructions	3	5	9
L3 cache miss rate	55.1%	33.6%	45.4%
-misses per thousand instructions	1	3	4
branch misprediction rate	5.0%	7.4%	9.1%
jump misprediction rate	2.2%	6.4%	12.9%
integer IQ-full (% of cycles)	7%	10%	9%
fp IQ-full (% of cycles)	14%	9%	3%
avg (combined) queue population	25	25	27
wrong-path instructions fetched	24%	7%	7%
wrong-path instructions issued	9%	4%	3%

Tullsen '96

From Superscalar to SMT

Per-thread hardware

- small stuff
- all part of current out-of-order processors
- none endangers the cycle time
- other per-thread processor state, e.g.,
  - program counters
  - return stacks
  - thread identifiers, e.g., with BTB entries, TLB entries
- per-thread bookkeeping for
  - instruction retirement
  - instruction queue flush

This is why there is only a 10% increase to Alpha 21464 chip area.

Implementing SMT

Thread-shared hardware:

- fetch buffers
- branch prediction structures
- instruction queues
- functional units
- active list
- all caches & TLBs
- MSHRs
- store buffers

This is why there is little single-thread performance degradation (~1.5%).

Design Challenges in SMT- Fetch

- Most expensive resources
  - Cache port
  - Limited to accessing the contiguous memory locations
  - Less likely that multiple thread from contiguous or even spatially local addresses
- Either provide dedicated fetch stage per thread
- Or time share a single port in fine grain or coarse grain manner
- Cost of dual porting cache is quite high
  - Time sharing is feasible solution

Design Challenges in SMT- Fetch

- Other expensive resource is Branch Predictor
  - Multi-porting branch predictor is equivalent to halving its effective size
  - Time sharing makes more sense

- Certain element of BP rely on serial semantics and may not perform well for multi-thread
  - Return address stack rely on FIFO behaviour
  - Global BHR may not perform well
  - BHR needs to be replicated

Inter-thread Cache Interference

- Because they share the cache, so more threads, lower hit-rate. (spatial locality gets affected).
- Two reasons why this is not a significant problem:
  1. The L1 Cache miss can almost be entirely covered by the 4-way set associative L2 cache.
  2. Out-of-order execution, write buffering and the use of multiple threads allow SMT to hide the small increases of additional memory latency.

0.1% speed up without interthread cache miss.

Increase in Memory Requirement

- More threads are used, more memory references per cycle.
- Bank conflicts in L1 cache account for the most part of the memory accesses.
- It is avoidable:
  1. For longer cache line: gains due to better spatial locality out-weighted the costs of L1 bank contention
  2. 3.4% speedup if no interthread contentions.

Fetch Policies

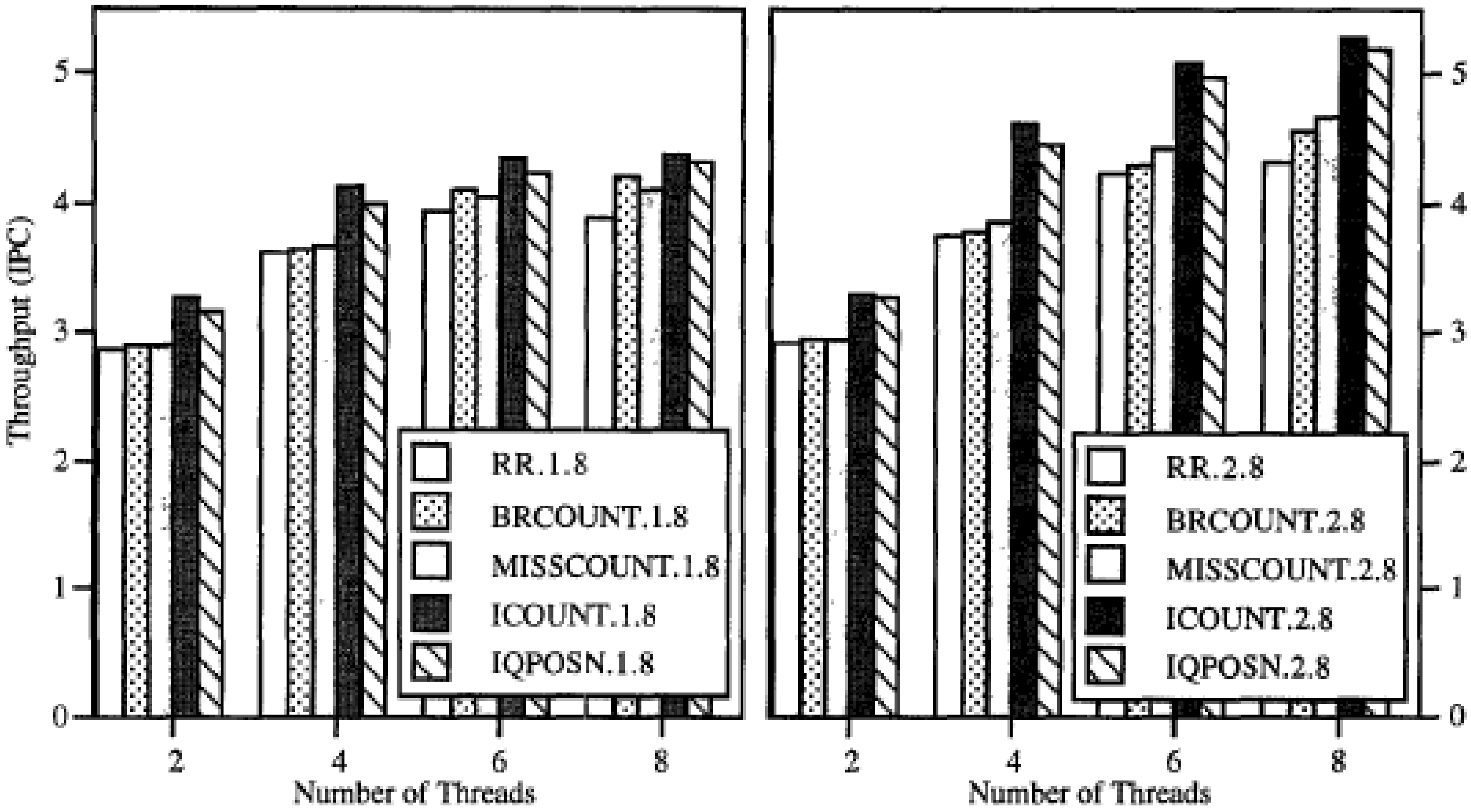
- Basic: Round-robin: RR.2.8 fetching scheme, i.e., in each cycle, two times 8 instructions are fetched in round-robin policy from two different 2 threads,
  - superior to different other schemes like RR.1.8, RR.4.2, and RR.2.4

- Other fetch policies:
  - BRCOUNT scheme gives highest priority to those threads that are least likely to be on a wrong path,
  - MISSCOUNT scheme gives priority to the threads that have the fewest outstanding D-cache misses
  - IQPOSN policy gives lowest priority to the oldest instructions by penalizing those threads with instructions closest to the head of either the integer or the floating-point queue
  - ICOUNT feedback technique gives highest fetch priority to the threads with the fewest instructions in the decode, renaming, and queue pipeline stages

Fetch Policies

Throughput comparison of Fetch Policy!

ICOUNT performs better



Dean Tullsen, 1996

Fetch Policies

- The ICOUNT policy proved as superior!
- The ICOUNT.2.8 fetching strategy reached a IPC of about 5.4 (the RR.2.8 reached about 4.2 only).
- Most interesting is that neither mispredicted branches nor blocking due to cache misses, but a mix of both and perhaps some other effects showed as the best fetching strategy.

- Simultaneous multithreading has been evaluated with
  - SPEC95,
  - database workloads,
  - and multimedia workloads.
- Both achieving roughly a 3-fold IPC increase with an eight-threaded SMT over a single-threaded superscalar with similar resources.

Design Challenges in SMT- Decode

- Primary tasks
  - Identify source operands and destination
  - Resolve dependency
- Instructions from different threads are not dependent
- Trade-off Single thread performance

Design Challenges in SMT- Rename

- Allocate physical register
- Map AR to PR
- Makes sense to share logic which maintain the free list of registers
- AR numbers are disjoint across the threads, hence can be partitioned
  - High bandwidth al low cost than multi-porting
- Limits the single thread performance

Design Challenges in SMT- Issue

- Tomasulo’s algorithm
- Wakeup and select
- Clearly improve the performance
- Selection
  - Dependent on the instruction from multiple threads
- Wakeup
  - Limited to intra-thread interaction
  - Make sense to partition the issue window
- Limit the performance of single thread

Design Challenges in SMT- Execute

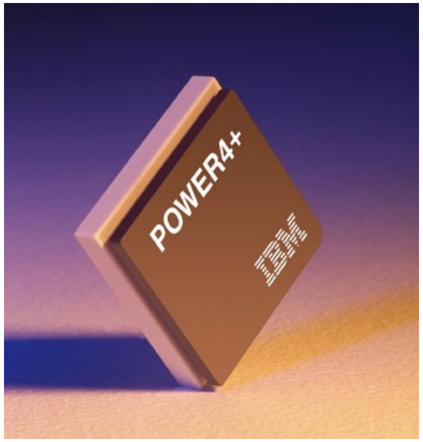
- Clearly improve the performance
- Bypass network
- Memory
  - Separate LS queue

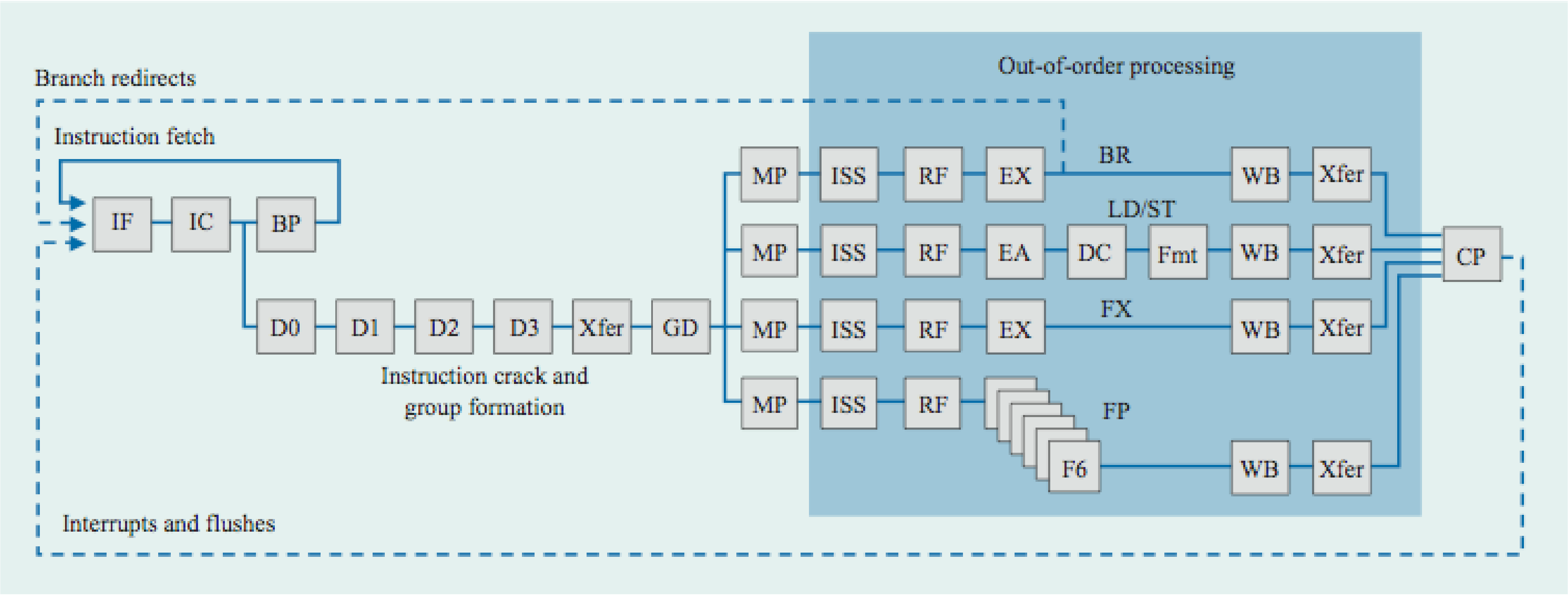
Multi-threading Processors

- Intel Hyperthreding (HT)
  - Dual threads
  - Pentium 4, XEON
- Sun CoolThreads
  - UltraSPARC T1
  - 4-threads per core
- IBM
  - POWER5

IBM POWER4

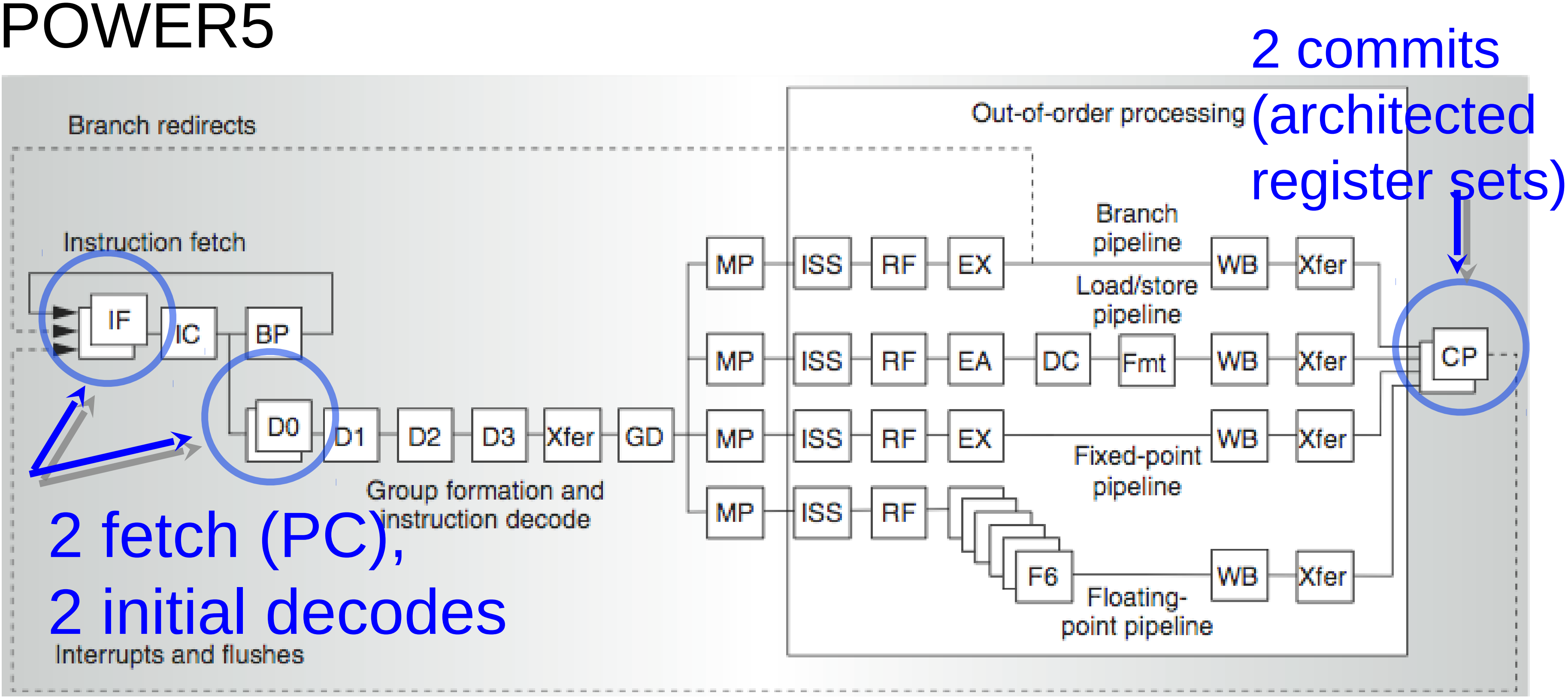
Single-threaded predecessor to POWER5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.



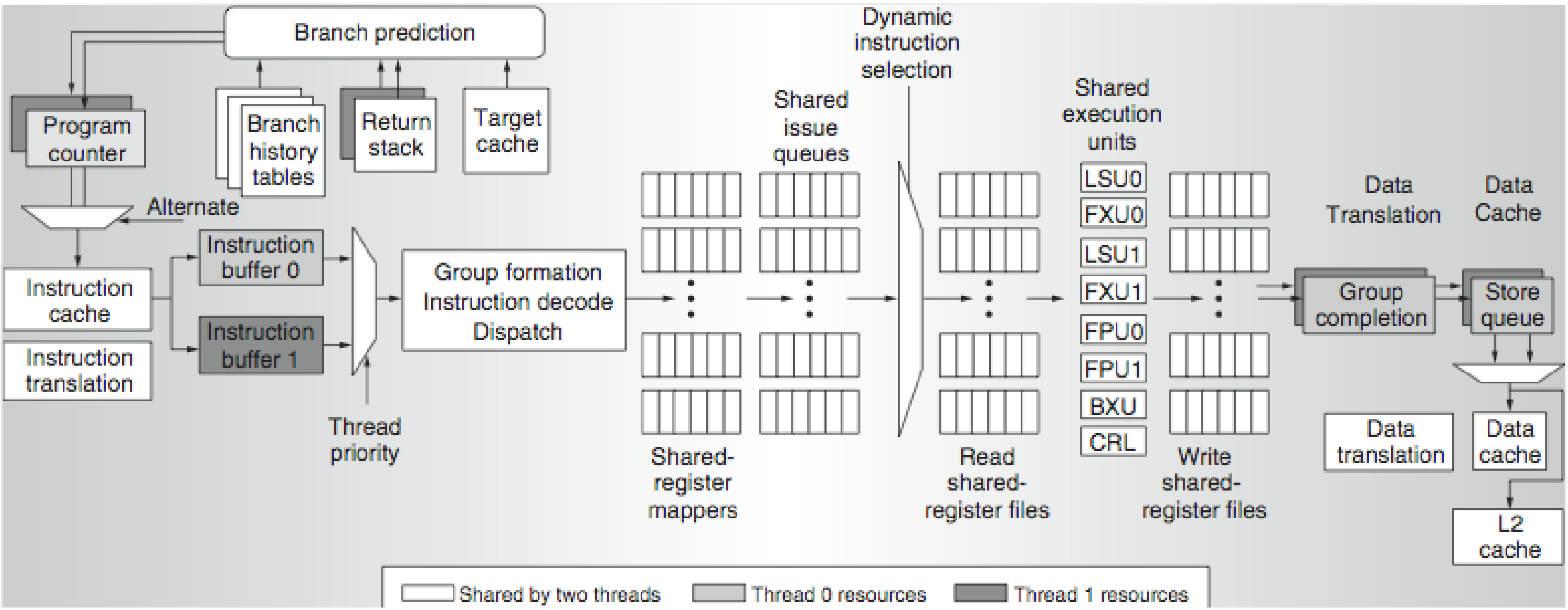


IBM POWER5

POWER5



POWER5 Data Flow



Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

Changes in POWER5 to Support SMT

- Increased associativity of L1 instruction cache and the instruction address translation buffers
- Added per thread load and store queues
- Increased size of the L2 and L3 caches
- Added separate instruction prefetch and buffering per thread
- Increased the number of virtual registers from 152 to 240
- Increased the size of several issue queues
- The POWER5 core is about 24% larger than the POWER4 core because of the addition of SMT support

IBM Power5

Table 1 Workloads selected for the study.

Workload	*Computation type*	*SMT gain (%)*
Sentence passing	Integer	41.2
Data compression	Integer	38.6
Programming language	Integer	26.3
3D Multi-grid Solver	Floating-point	21.6
Circuit Routing	Integer	19.8
Seismic Wave Simulation	Floating-point	15.3
Object-oriented Database	Integer	12.5
Neural Network	Floating-point	11.2

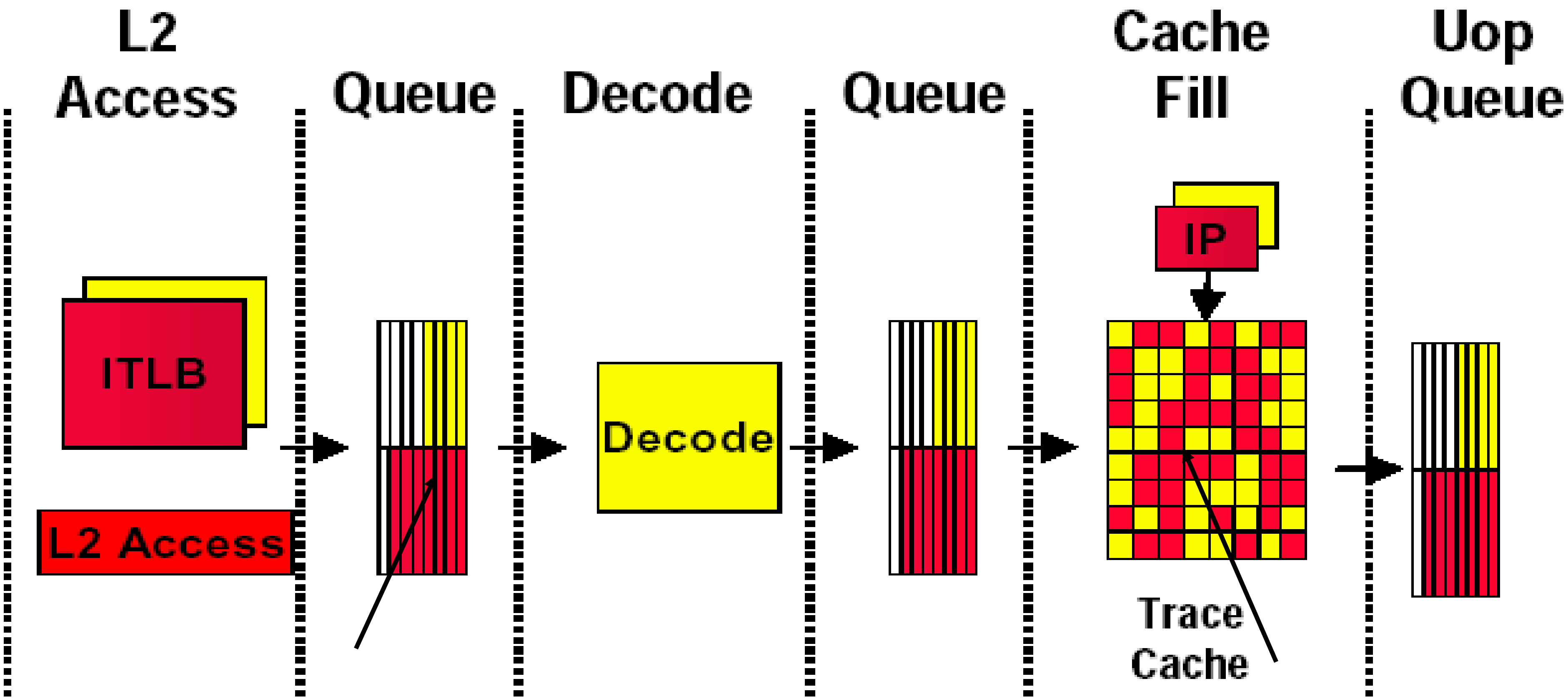
http://www.research.ibm.com/journal/rd/494/mathis.pdf

Pentium-4 Hyperthreading (2002)

- First commercial SMT design (2-way SMT)
  - Hyperthreading == SMT
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - No logical processor can use all entries in queues when two threads are active
- Processor running only one active software thread runs at approximately same speed with or without hyperthreading

Pentium-4 Hyperthreading

Front End



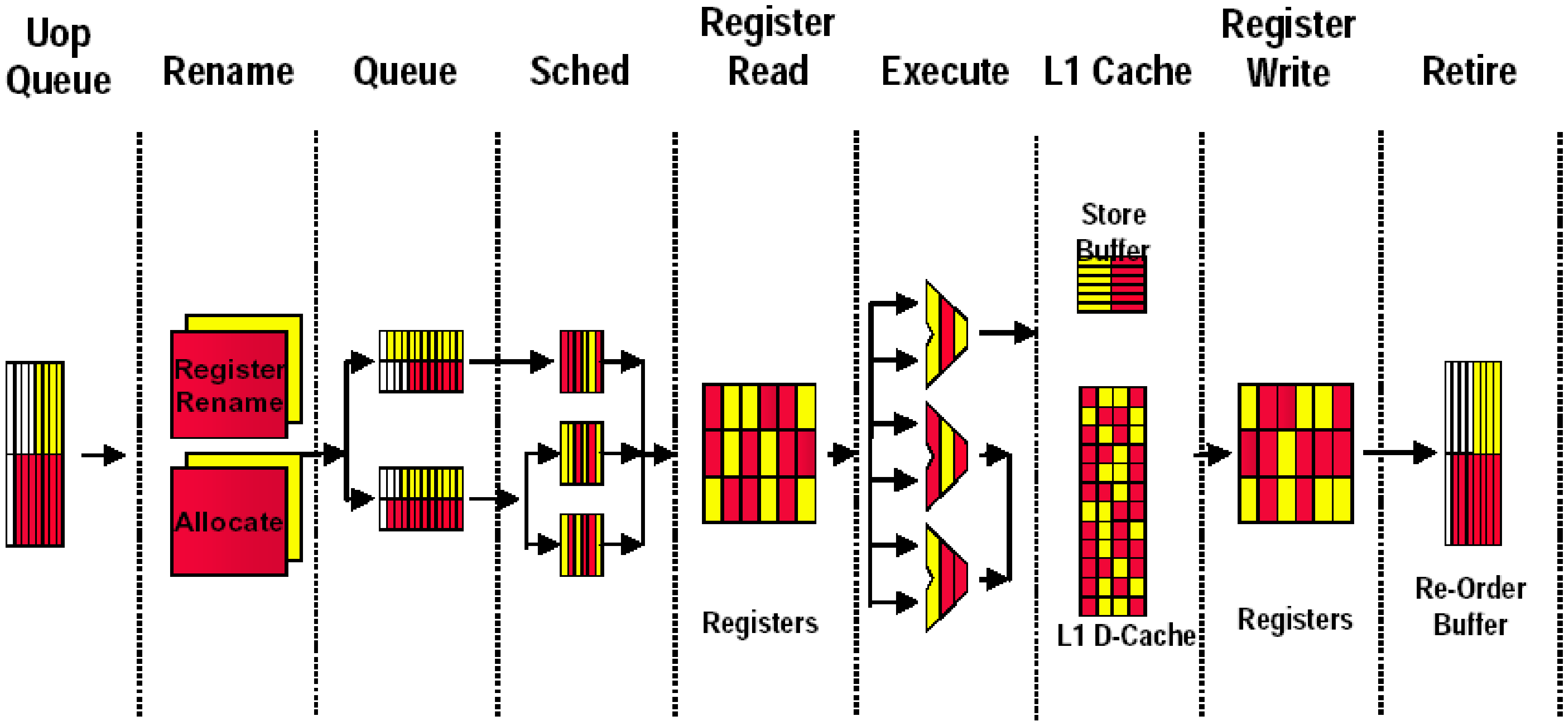
Resource divided between logical CPUs

Resource shared between logical CPUs

[Intel Technology Journal, Q1 2002]

Pentium-4 Hyperthreading

Execution Pipeline



[Intel Technology Journal, Q1 2002]

Initial Performance of SMT

- P4 Extreme Edition SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium 4 is dual threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on P4 each of 26 SPEC benchmarks paired with every other (26^2 runs) speed-ups from 0.90 to 1.58; average was 1.20
- POWER5, 8 processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate
- POWER5 running 2 copies of each app speedup between 0.89 and 1.41
  - Most gained some
  - FP apps had most cache conflicts and least gains

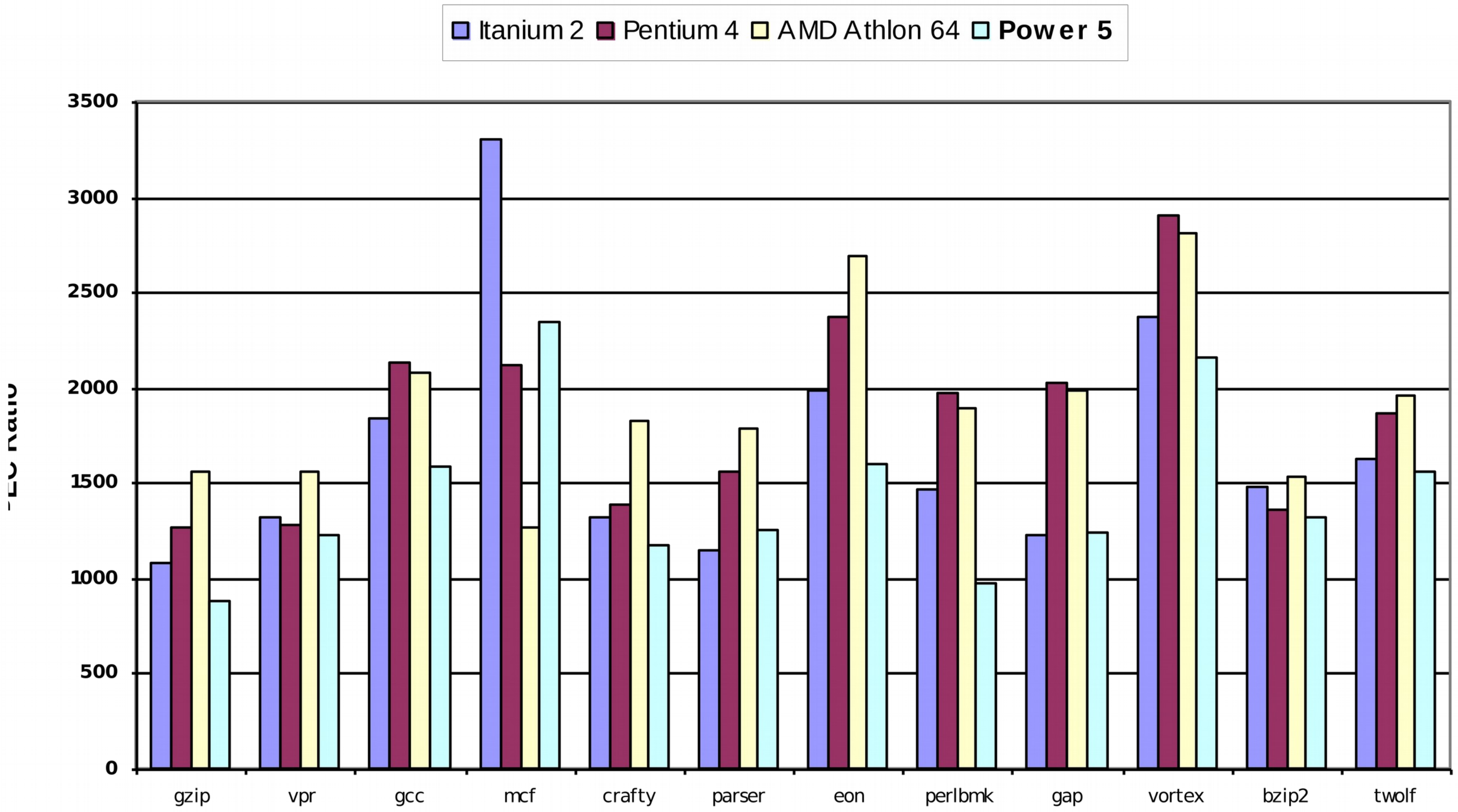
Initial Performance of SMT

- P4 Extreme Edition SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium 4 is dual threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on P4 each of 26 SPEC benchmarks paired with every other (26^2 runs) speed-ups from 0.90 to 1.58; average was 1.20
- POWER5, 8 processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate
- POWER5 running 2 copies of each app speedup between 0.89 and 1.41
  - Most gained some
  - FP apps had most cache conflicts and least gains

Head to Head ILP competition

Processor	Micro architecture	Fetch / Issue / Execute	FU	Clock Rate (GHz)	Transis-tors Die size	Power
Intel Pentium 4 Extreme	Speculative dynamically scheduled; deeply pipelined; SMT	3/3/4	7 int. 1 FP	3.8	125 M 122 mm^2	115 W
AMD Athlon 64 FX-57	Speculative dynamically scheduled	3/3/4	6 int. 3 FP	2.8	114 M 115 mm^2	104 W
IBM POWER5 (1 CPU only)	Speculative dynamically scheduled; SMT; 2 CPU cores/chip	8/4/8	6 int. 2 FP	1.9	200 M 300 mm^2 (est.)	80W (est.)
Intel Itanium 2	Statically scheduled VLIW-style	6/5/11	9 int. 2 FP	1.6	592 M 423 mm^2	130 W

Performance on SPECint2000



No Silver Bullet for ILP

- No obvious over all leader in performance
- The AMD Athlon leads on SPECInt performance followed by the P4, Itanium 2, and POWER5
- Itanium 2 and POWER5, which perform similarly on SPECFP, clearly dominate the Athlon and P4 on SPECFP
- Itanium 2 is the most inefficient processor both for FP and integer code for all but one efficiency measure (SPECFP/Watt)
- Athlon and P4 both make good use of transistors and area in terms of efficiency
- IBM POWER5 is the most effective user of energy on SPECfp and essentially tied on SPECint

Limits to ILP

- Doubling issue rates above today's 3-6 instructions per clock, say to 6 to 12 instructions, probably requires a processor to
  - issue 3 or 4 data memory accesses per cycle,
  - resolve 2 or 3 branches per cycle,
  - rename and access more than 20 registers per cycle, and
  - fetch 12 to 24 instructions per cycle.
- The complexities of implementing these capabilities is likely to mean sacrifices in the maximum clock rate
  - E.g, widest issue processor is the Itanium 2, but it also has the slowest clock rate, despite the fact that it consumes the most power!

Limits to ILP

- Most techniques for increasing performance increase power consumption
- The key question is whether a technique is *energy efficient*: does it increase power consumption faster than it increases performance?
- Multiple issue processors techniques all are energy inefficient:
  1. Issuing multiple instructions incurs some overhead in logic that grows faster than the issue rate grows
  2. Growing gap between peak issue rates and sustained performance
- Number of transistors switching = f(peak issue rate), and performance = f(sustained rate),
  growing gap between peak and sustained performance
  increasing energy per unit of performance

Next Lecture

Multi-core/Multi-processor