Simultaneous Multithreading

Multiple Issue Machines

- To decrease the CPI to less than “one”, *multiple issue machines*—they come in two flavors
- **Superscalar**: **multiple** parallel dedicated **pipelines**:
  - Issue varying number of instructions per cycle
    - either statically scheduled by compiler(in-order) and/or

      dynamically by hardware(out-of-order) (Tomasulo--# of inst issued depends upon?)
    - IBM PowerPC, Sun UltraSparc, DEC Alpha, IA32 Pentium
- **Explicit parallelism** → VLIW (Very Long Instruction Word):
  - also classified as EPIC (Explicitly Parallel Instruction Computer)
  - several operations encoded as one long instruction
  - instructions have wide template (4-16 operations)
    - e.g. IA-64 Itanium
- **Explicit parallelism** → Word-level / SIMD / Vector processors:
  - multimedia instruction sets (Intel’s MMX and SSE, Sun’s VIS, etc.)

Choices for Multiple Issue machines

Common Name	Issue structure	Hazard detection	Scheduling	Examples
Superscalar (static)	Dynamic	Hardware	Static	Sun UltraSPARC II/III
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	IBM POWER2
Superscalar (speculative)	Dynamic	Hardware	Dynamic with Speculation	PentiumIII/4, MIPS R10k, Alpha21264, IBM POWER4, HP PA8500
VLIW	Static	Software	Static	Trimedia, i860
EPIC	“mostly” static	Mostly software	Mostly static	Itanium (IA64)

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Basic approach: fetch (when possible) two instructions belonging to different classes each clock cycle
- Superscalar MIPS:

  2 instructions/cycle, 1 FP & 1 anything else

  – Fetch 64-bits/clock cycle; integer on left, FP on right

  – Issue 2 instructions in one clock cycle

Int inst	**FP** inst

	Single Issue Clock Cycle									
	1	2	3	4	5	6	7	8	9	10
i	IF	ID	EX	M	WB					
i+1		IF	ID	EX	M	WB				
i+2			IF	ID	EX	M	WB			
i+3				IF	ID	EX	M	WB		
i+4					IF	ID	EX	M	WB	
i+5						IF	ID	EX	M	WB

Multiple Issue Clock Cycle						
1	2	3	4	5	6	7
IF	ID	EX	M	WB		
IF	ID	EX	M	WB		
	IF	ID	EX	M	WB	
	IF	ID	EX	M	WB	
		IF	ID	EX	M	WB
		IF	ID	EX	M	WB

Multiple Issue Challenges

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations
  - No hazards
- If more instructions issued at same time, greater difficulty in decode and issue
  - Even 2-way scalar ⇒ examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue
- VLIW: tradeoff=larger instruction space but simpler decoding—no hardware scheduling
  - The long instruction word has room for many operations
  - By definition, all the operations the compiler puts in the long instruction word are independent ⇒ execute in parallel
  - E.g., 2 integer operations, 2 FP ops, 2 memory refs, 1 branch ⇒ 16 to 24 bits per field ⇒ 7*16 or 112 bits to 7*24 or 168 bits wide
  - Need compiling technique that schedules across several branches
  - Static issue– completely rely on compilers to schedule
    (Superscalar machines rely on hardware →dynamic issue)

Limitations to Multi-Issue Machines

- As usual, we have to deal with the **three hazards**:
  - Structural Hazards
  - Data Hazards
  - Control Hazards
- Multiple issue gives:
  - More hazards probable
  - Also larger performance hit from hazards

Summary of techniques to increase ILP

To increase ILP →Reduce Stalls

Techniques for reducing Stalls

Hardware techniques

1. Dynamic inst scheduling
   - -Scoreboard
   - -Tomasulo's

   Reduces Structural & Data Hazards
2. Dynamic Branch Predictions
3. Branch Target Buffers + Branch Target cache + Return address predictor
4. Superscalar/multiple issue
   - -Superscalar
   - -VLIW
5. Hardware based Speculation

   Reduces Control Hazards

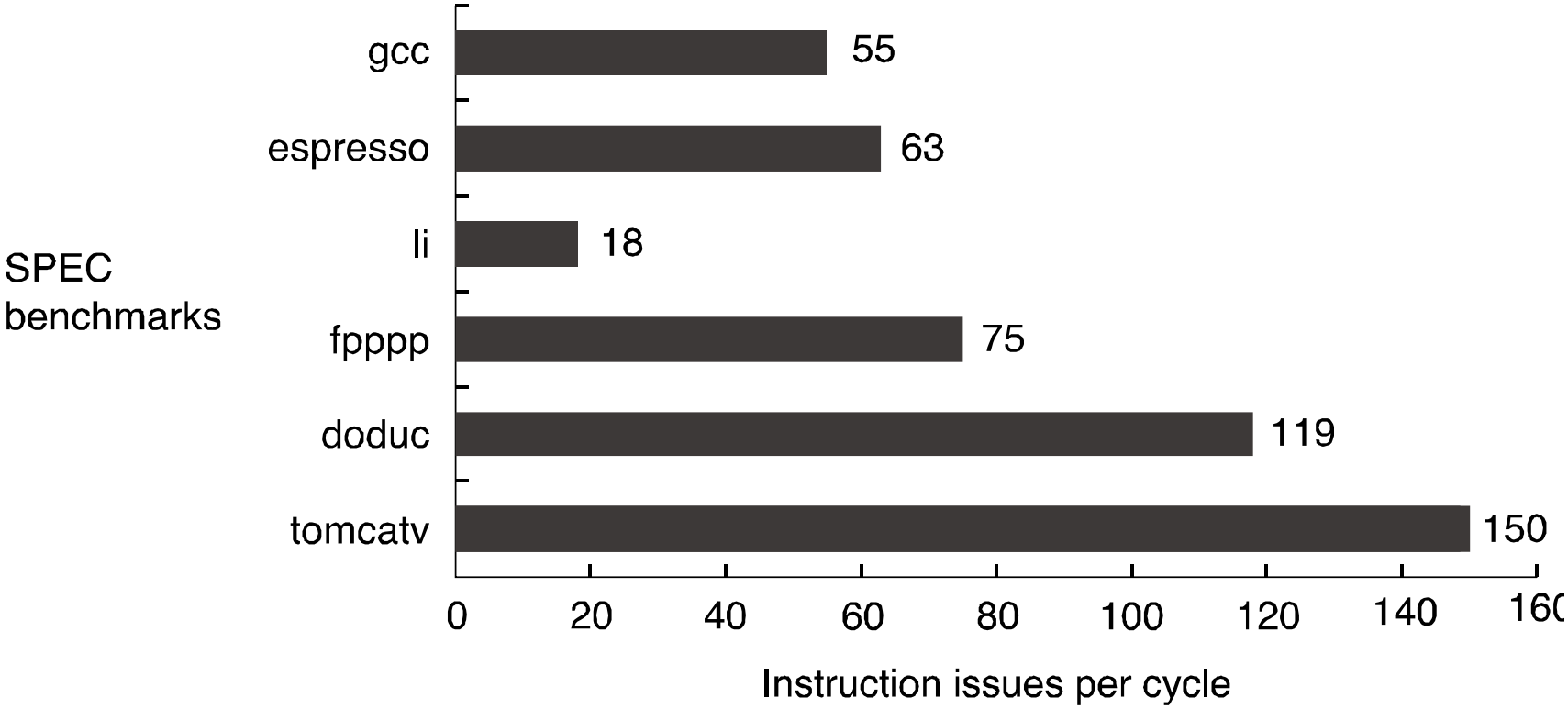
Software techniques by compiler

Summary of techniques to increase ILP

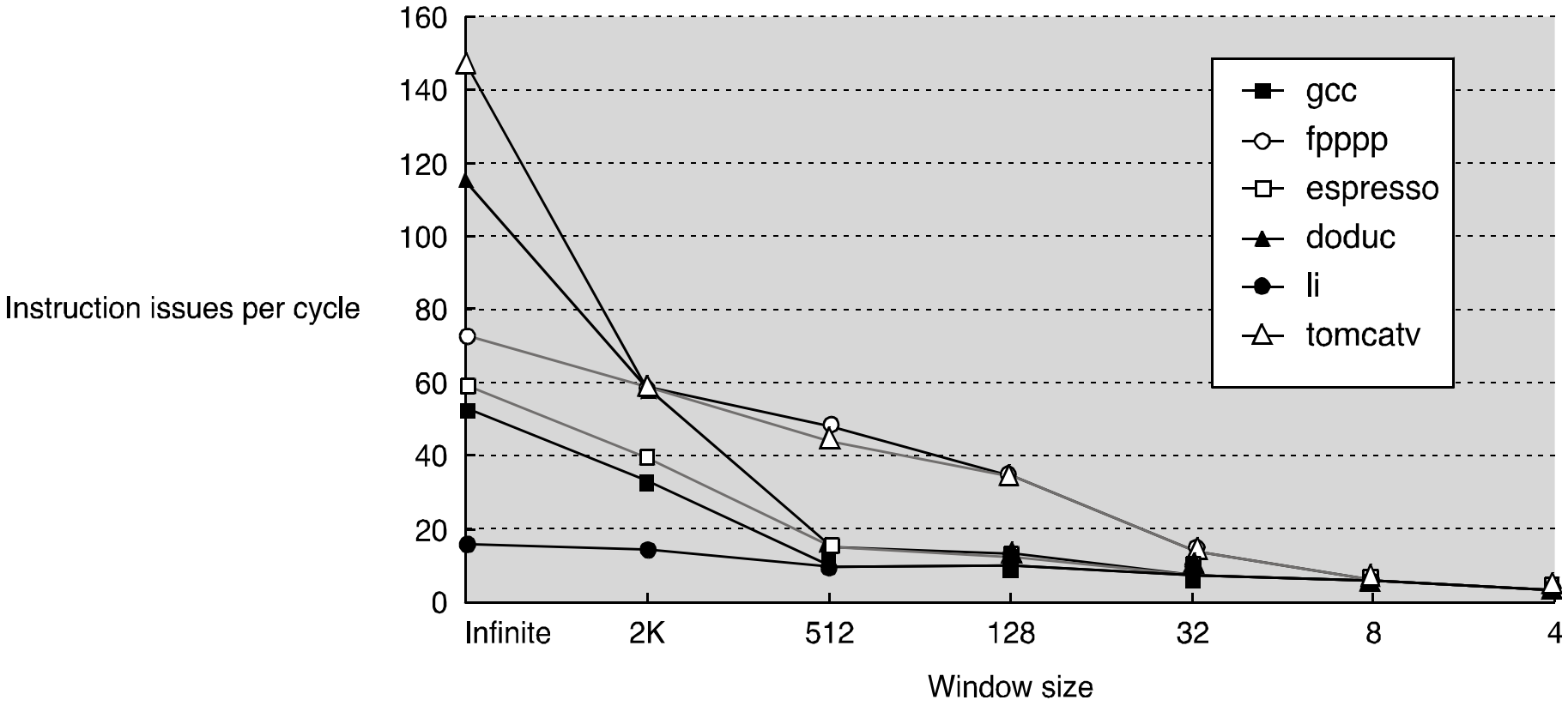
Techniques	Reduces
Forwarding and bypassing	Potential data hazard stalls
Delayed branches and simple branch scheduling	Control hazard stalls
Basic dynamic scheduling (scoreboarding)	Data hazard stalls from true dependences
Dynamic scheduling with renaming	Data hazard stalls and stalls from antidependences and output dependences
Dynamic branch prediction	Control stalls
Issuing multiple instructions per cycle	Ideal CPI
Speculation	Data hazards and control hazard stalls
Dynamic memory disambiguation	Data hazard stalls with memory
Loop unrolling	Control hazard stalls
Basic compiler pipeline scheduling	Data hazard stalls
Compiler dependence analysis	Ideal CPI, data hazard stalls
Software pipelining, trace scheduling	Ideal CPI, data hazard stalls
Compiler speculation	Ideal CPI, data, control stalls

Studies of the Limitations of ILP:

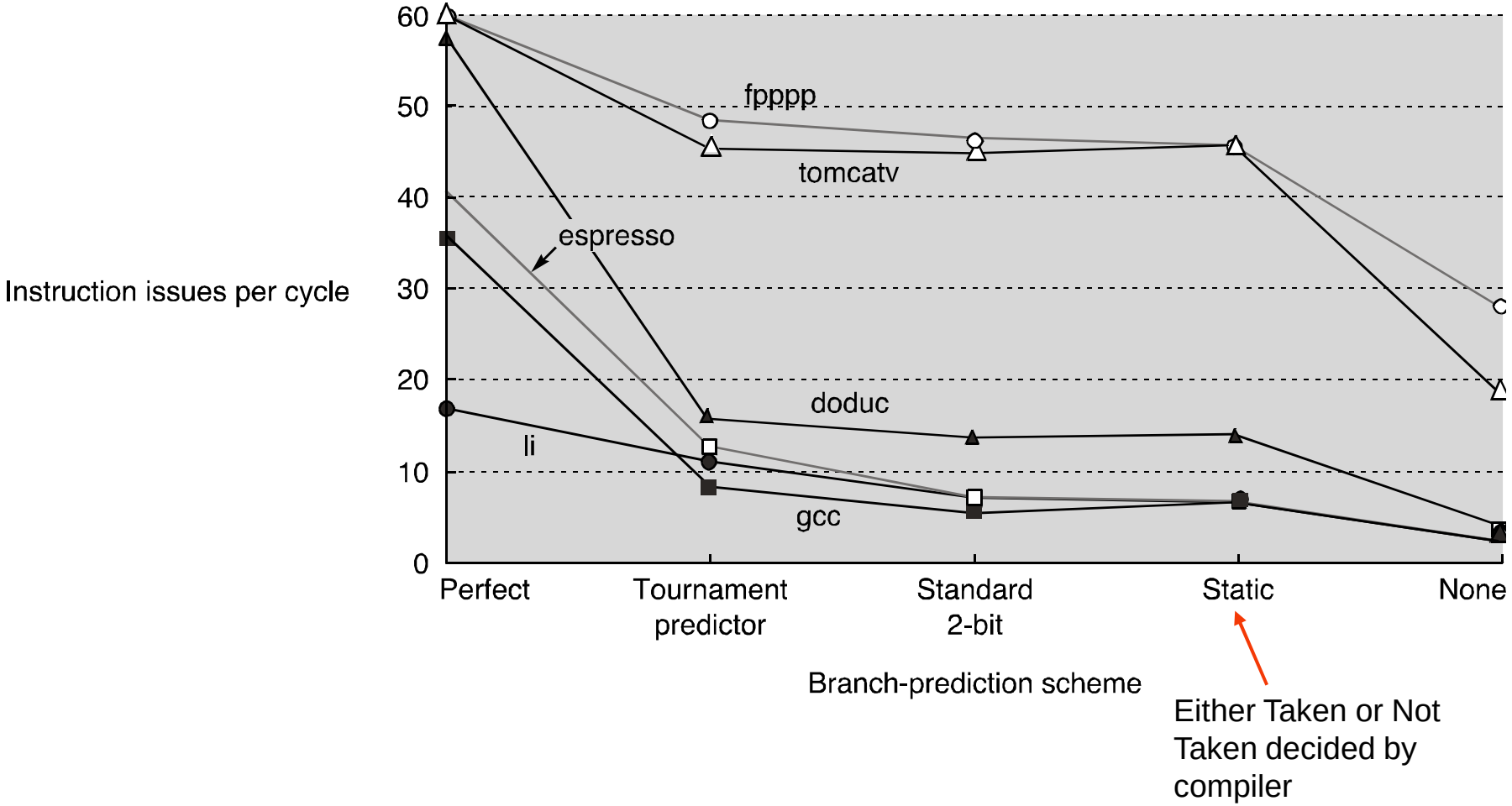
ILP available in a perfect processor



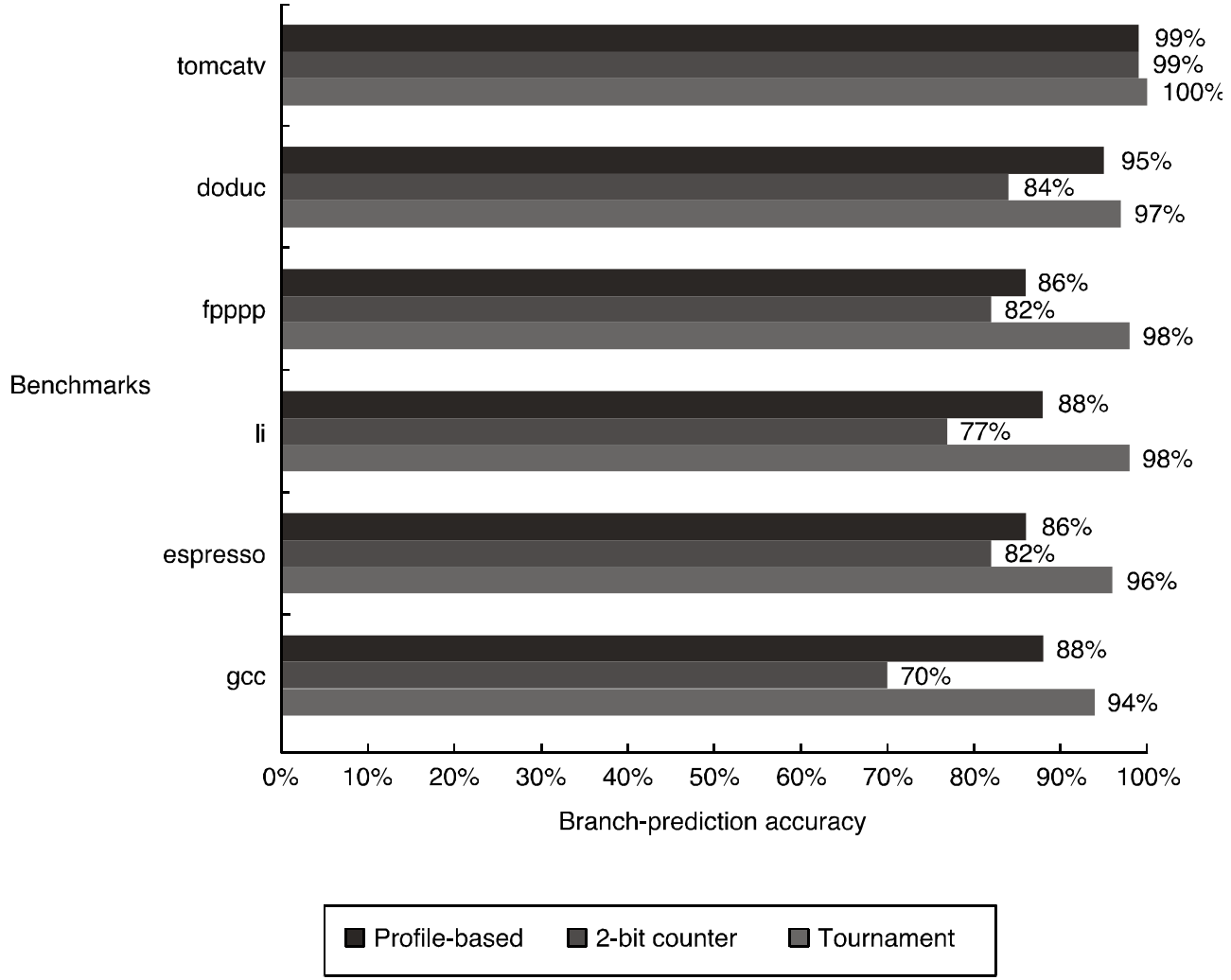
Effect of Window size on ILP



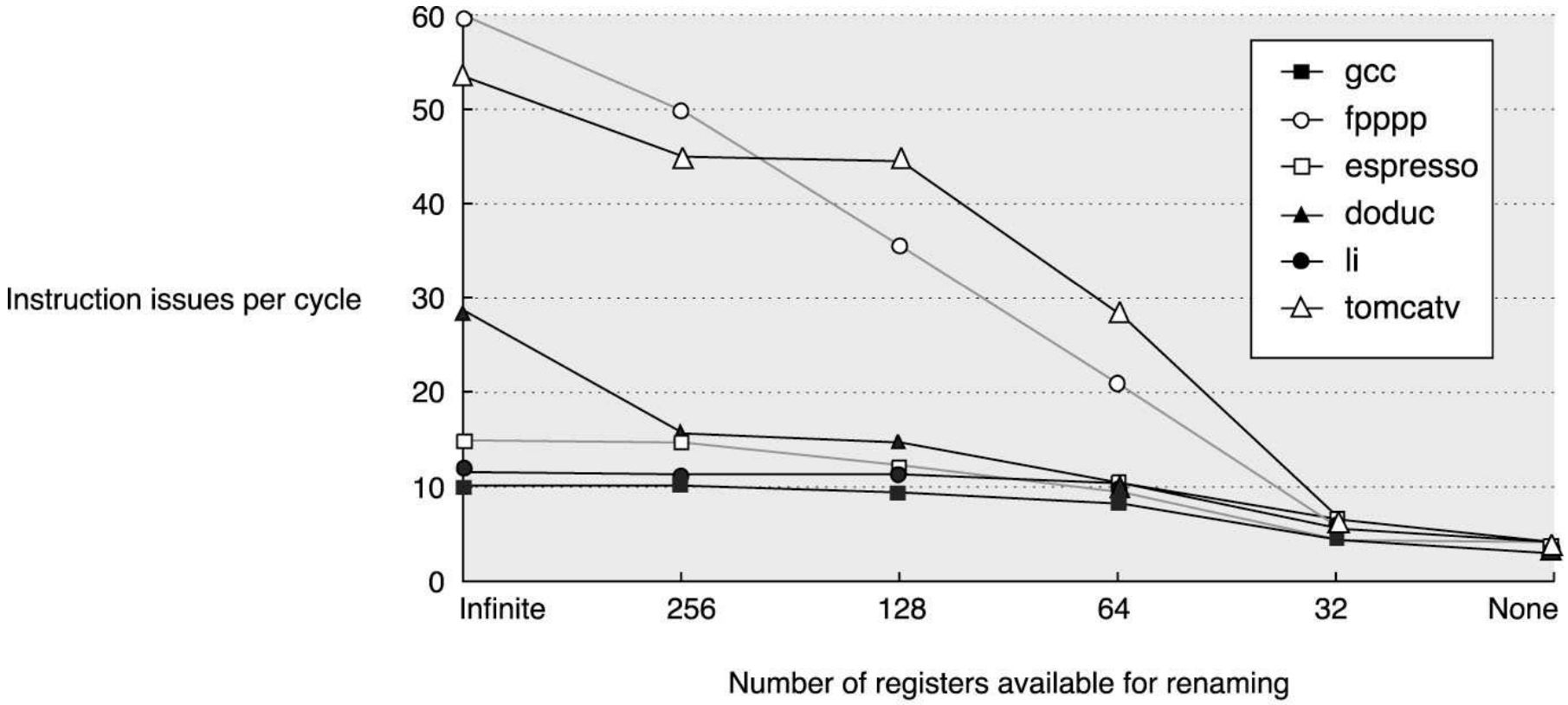
Effect of Branch Prediction Schemes



Conditional Branch Prediction accuracy



Effect of number of registers available for renaming

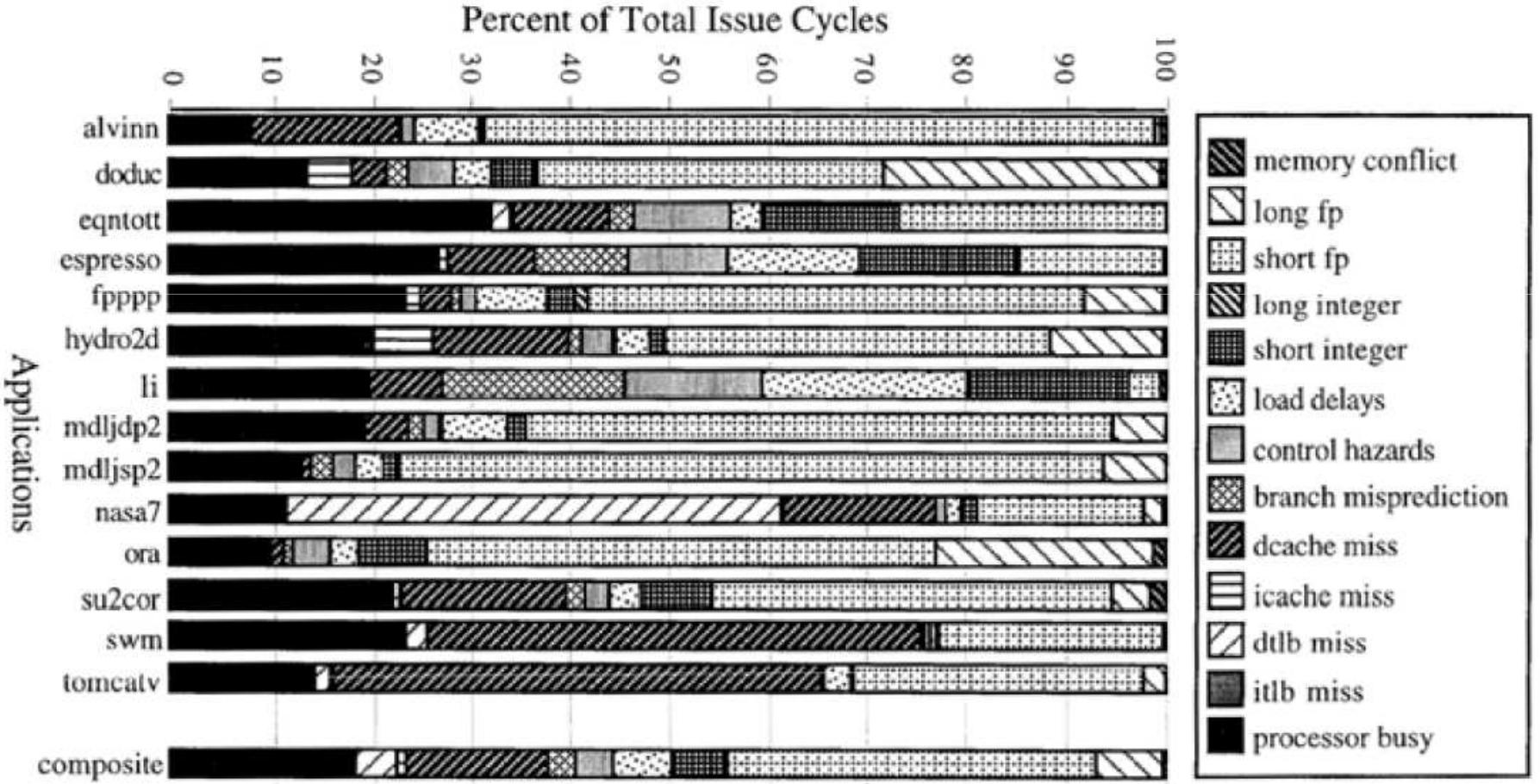


Contemporary forms of parallelism

- Instruction-level parallelism (ILP)
  - Wide-issue SuperScalar processors (SS)
    - 4 or more instruction per cycle
    - Executing a single program or thread
    - Attempts to find multiple instructions to issue each cycle.

- Thread-level parallelism (TLP)
  - Fine-grained multithreaded superscalars (FGMS)
    - Contain hardware state for several threads
    - Executing multiple threads
    - On any given cycle a processor executes instructions from one of the threads
  - Multiprocessor (MP)
    - Performance improved by adding more CPUs

Superscalar Bottlenecks

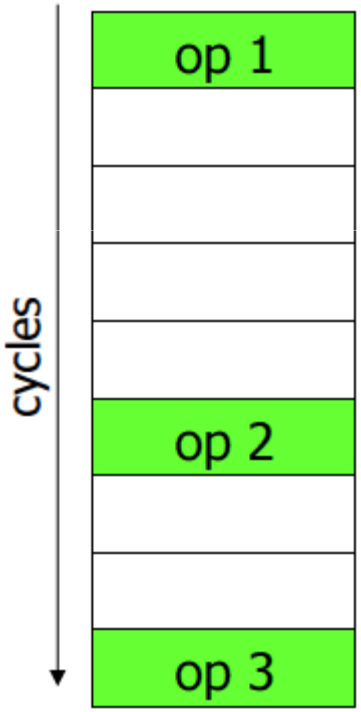


Source: Tullsen et al., © IEEE 1995

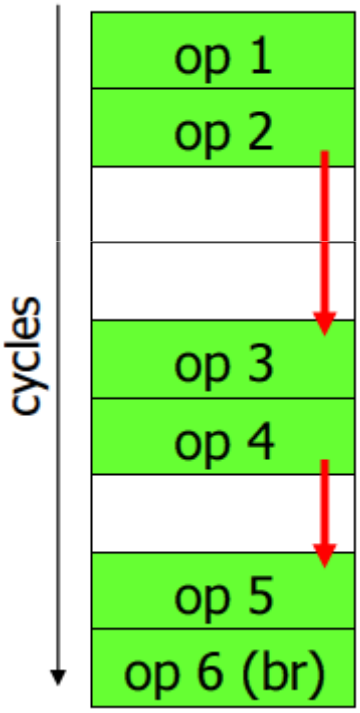
Superscalar Bottlenecks

- No dominant source of wasted issue bandwidth, therefore, no dominant solution
- No single latency-tolerating technique will produce a dramatic increase in the performance of these programs if it only attacks specific types of latencies
- Examples:
  - decrease the TLB miss rates (e.g., increase the TLB sizes)
  - larger, more associative, or faster instruction/data cache hierarchy
  - improved branch prediction scheme; lower branch misprediction penalty
  - speculative execution; more aggressive if-conversion

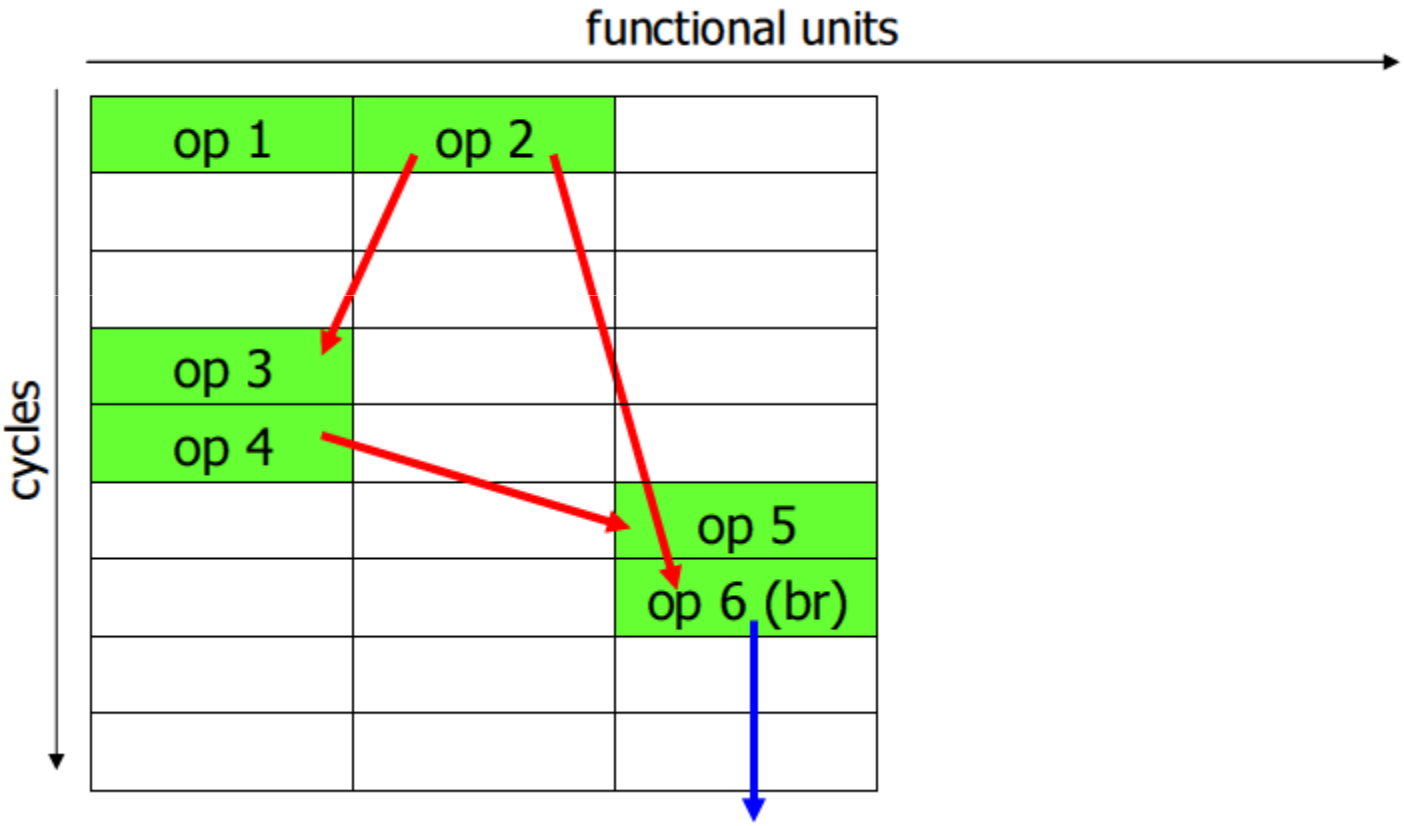
Simple Sequential Processor



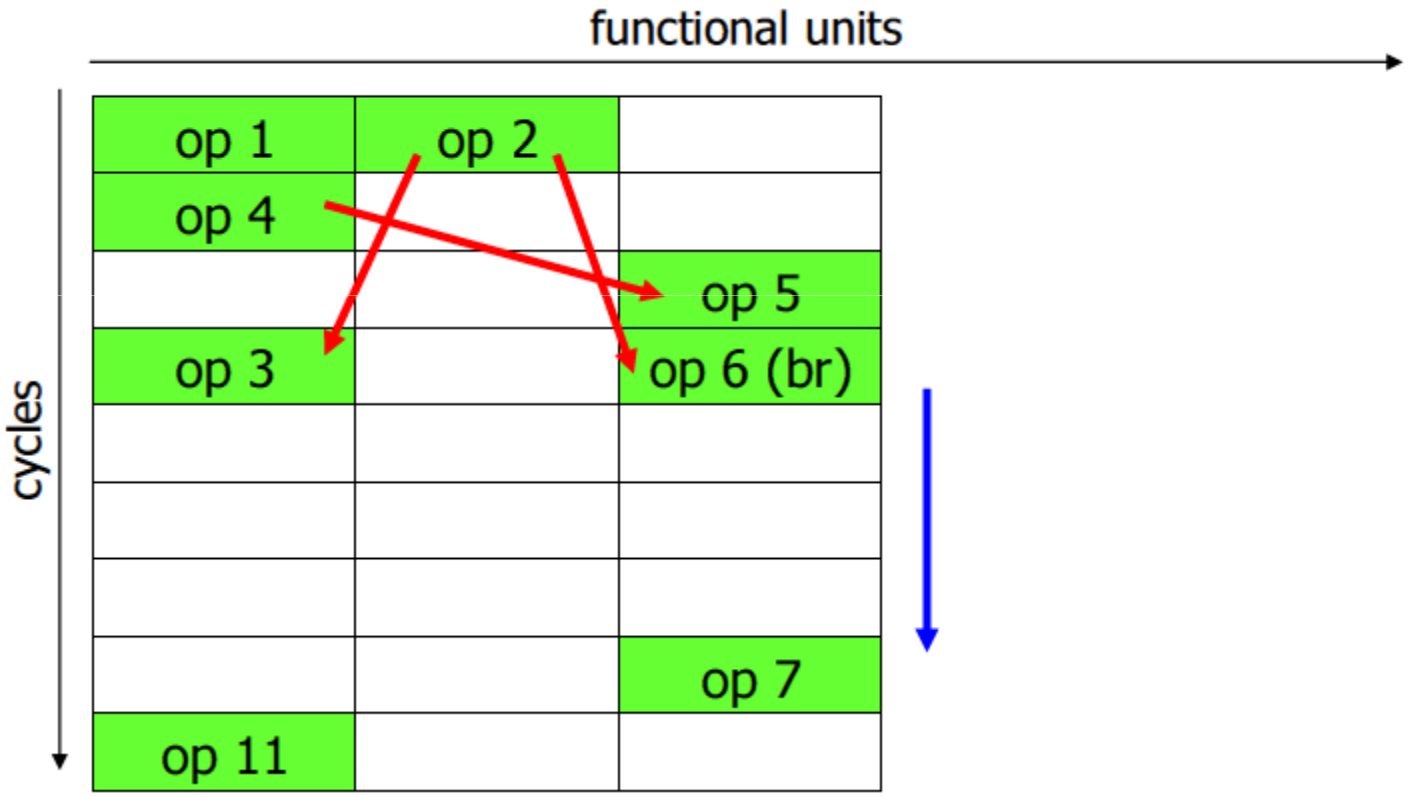
Pipelined Processor



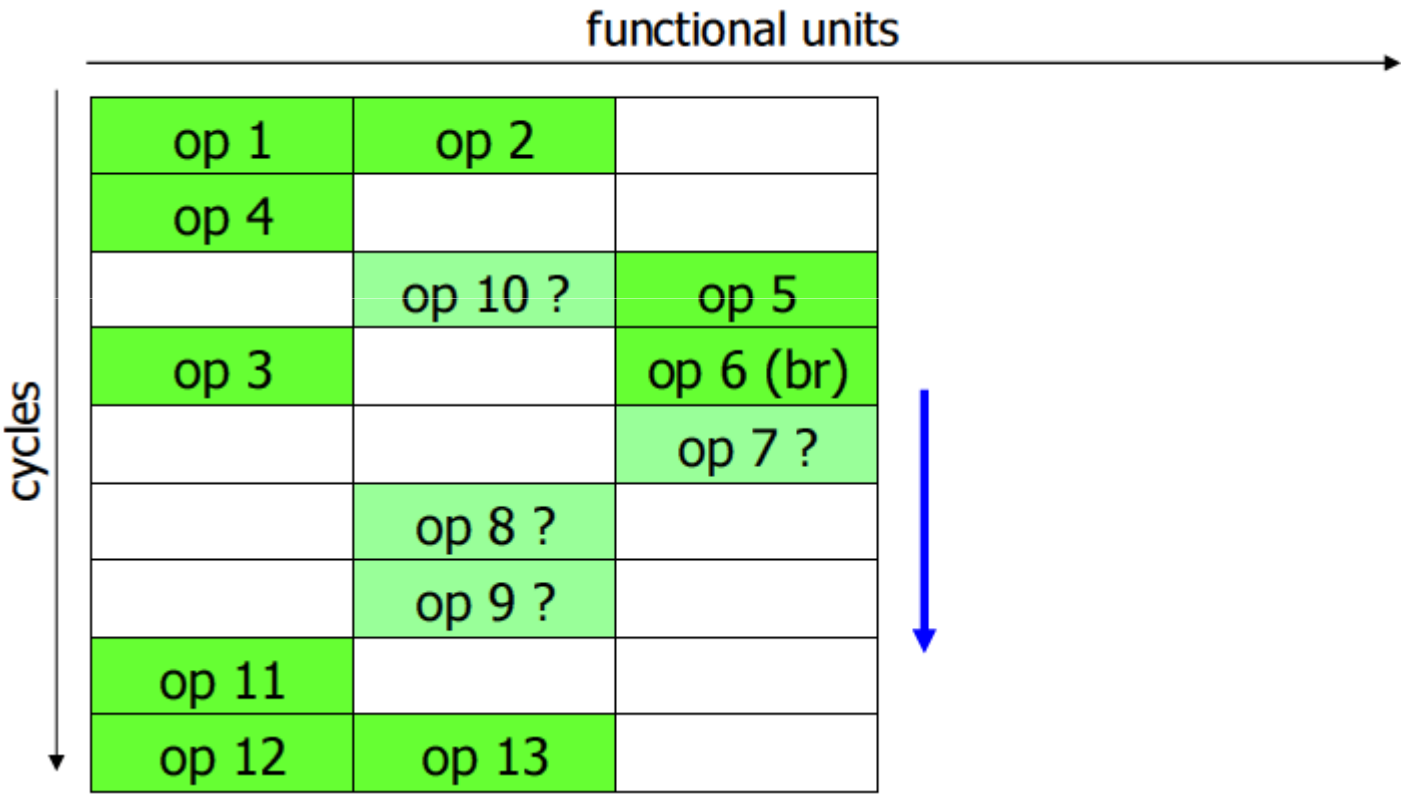
Superscalar Processor



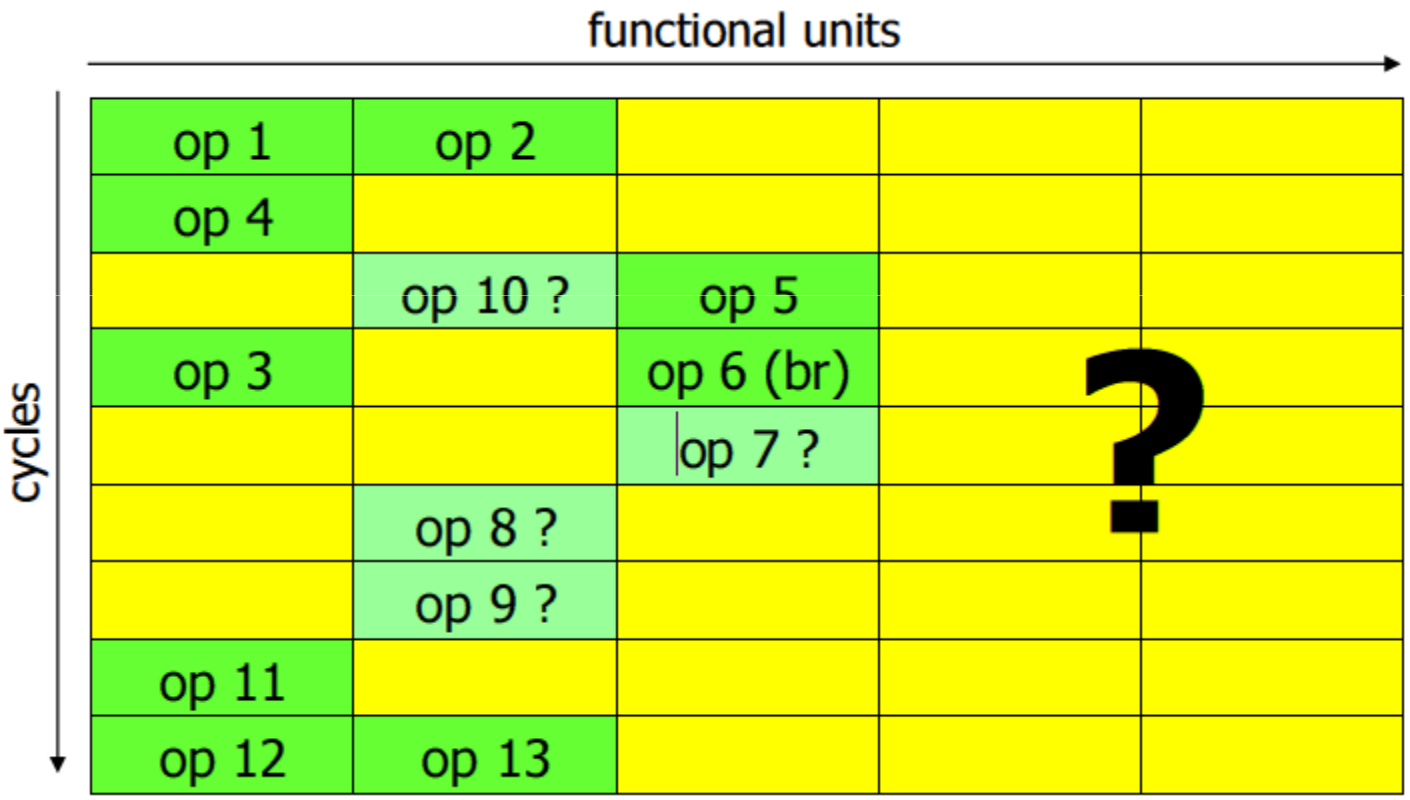
OOO Superscalar Processor



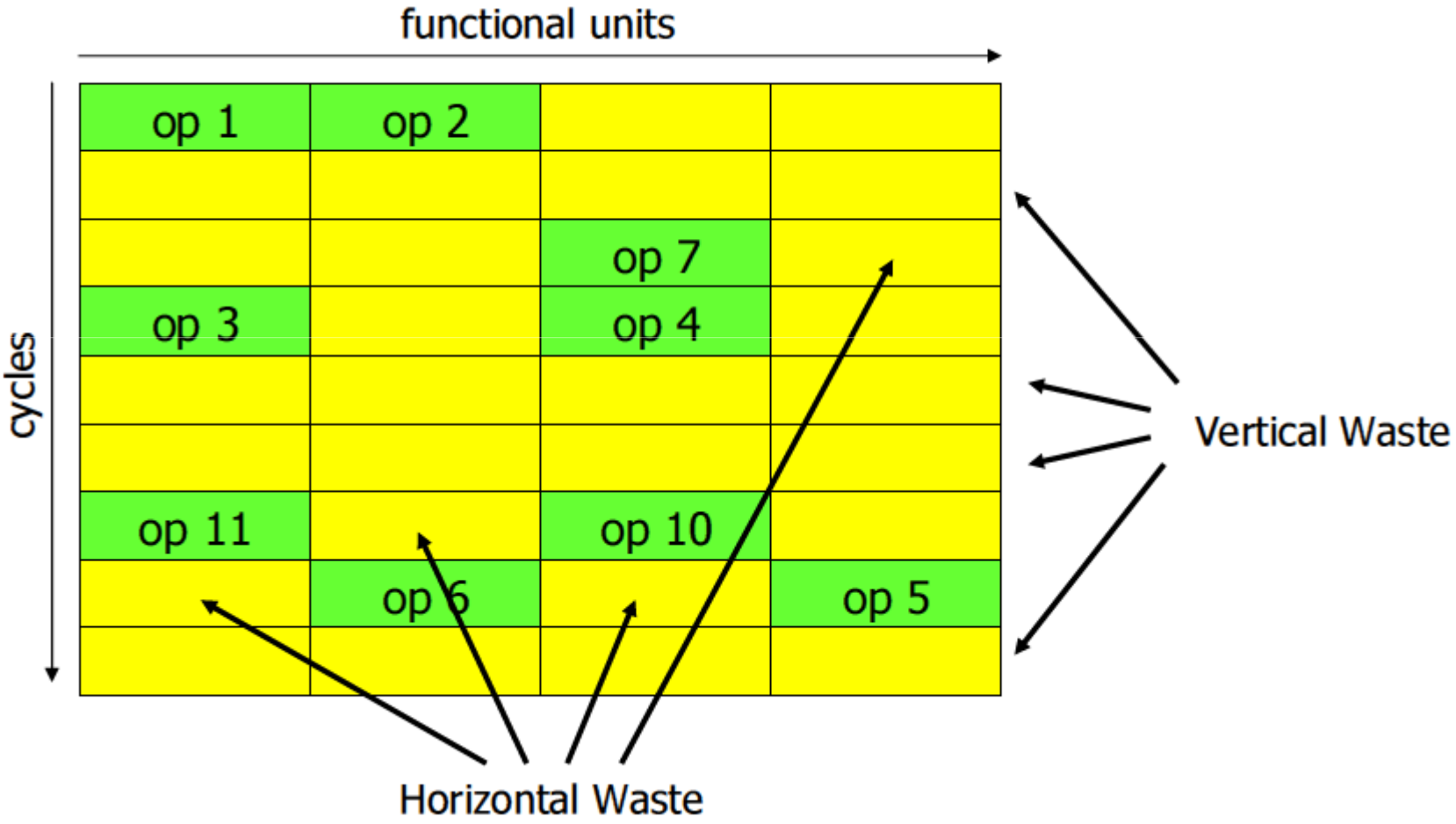
Speculative Execution



Limits of ILP



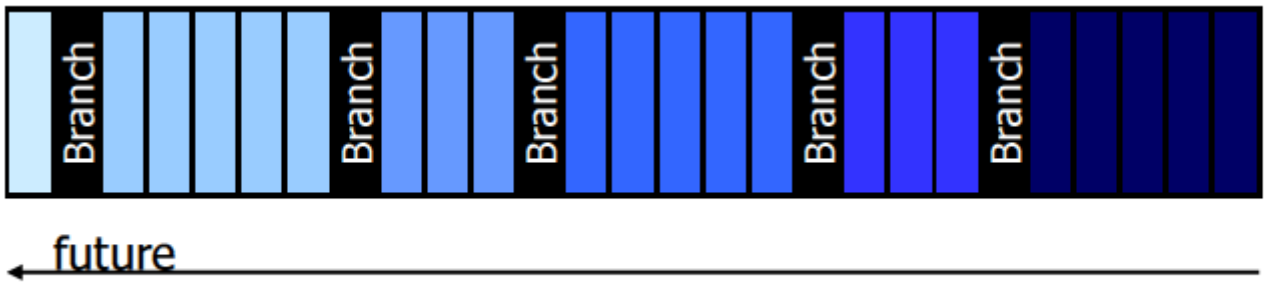
Horizontal and Vertical Waste



Multithreading

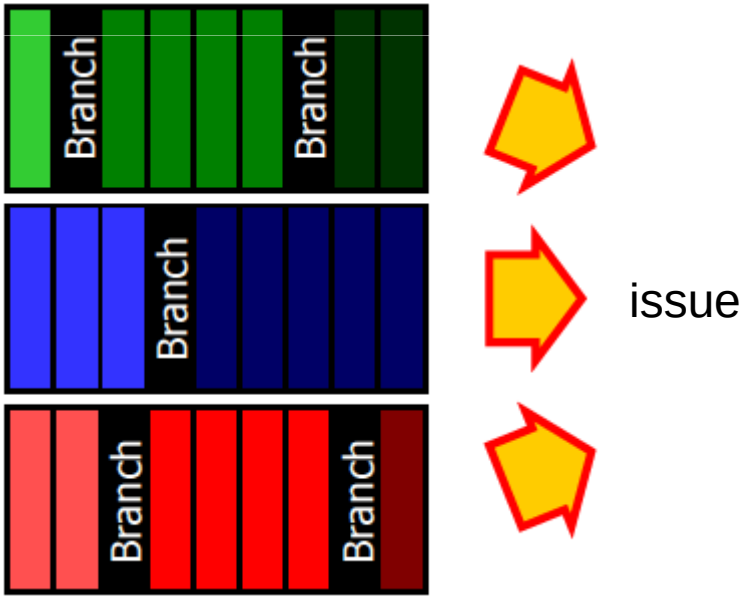
- Key idea
  - ***Issue multiple instructions from multiple threads each cycle***
- Somewhere in between implicit and explicit parallelism
- Features
  - Fully exploits thread-level parallelism and instruction-level parallelism
  - Potentially better performance for
    - A suitable mix of independent programs
    - Programs that are parallelizable
  - These scenarios are likely to happen for both servers (e.g. a Web server serving several independent requests) and desktop computers.
- Possibly still transparent to the programmer/compiler

Multithreading



- Enlarge the "**width**" of the the instruction window *rather than its depth*

→ fetch from ***multiple threads***



Hardware Multithreading

- Processor must be aware of *several independent states*, one per each thread:
  - Program Counter
  - Register File (and Flags)
  - (Memory)
- We need either multiple resources in the processor or a fast way to switch across states
  - E.g. several physical Register Files OR fast copy to/from memory of a Register File image

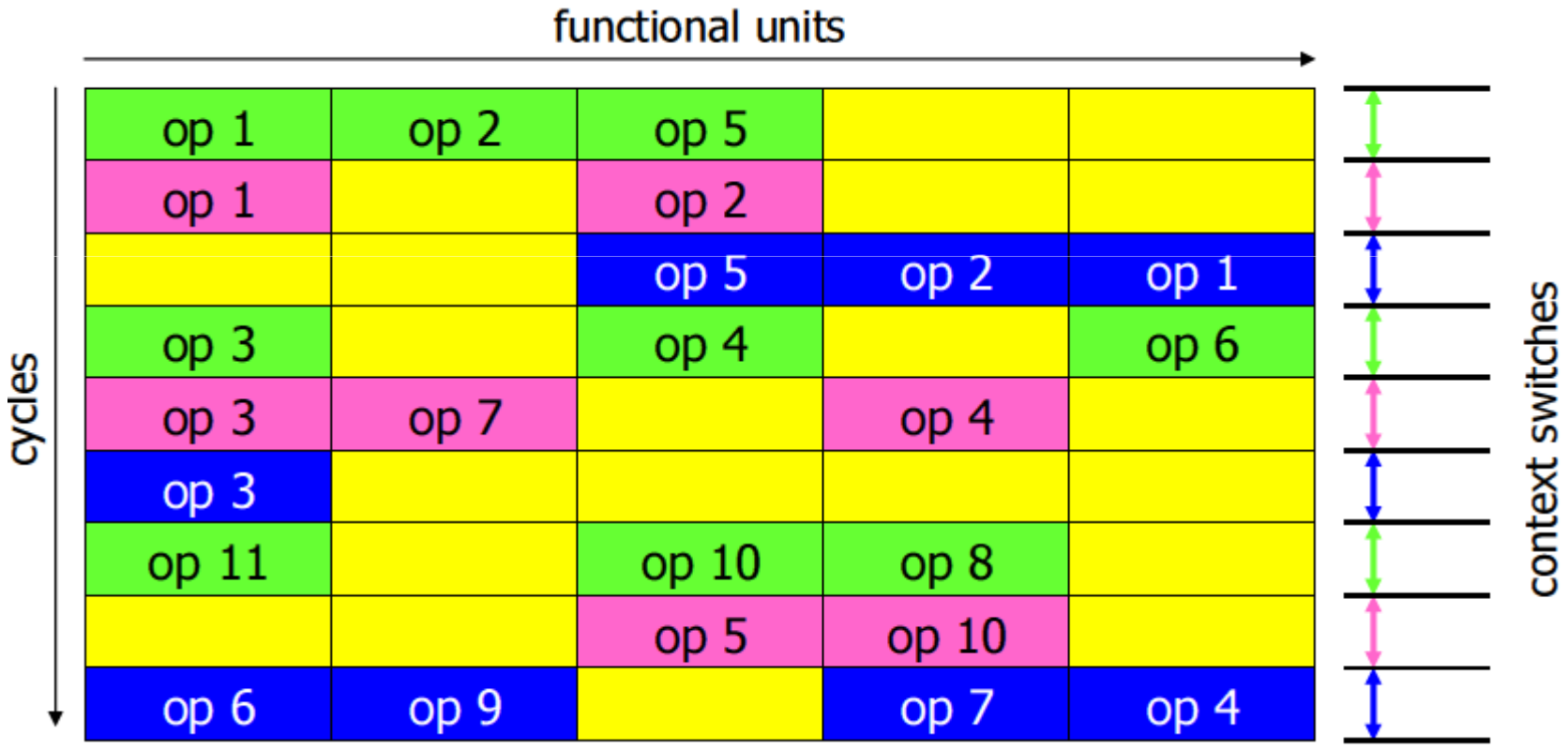
- Early hardware solutions include the so-called *Barrel processors* (including even CDC6000!) and more recent architectures such as HEP, TERA, MASA, Alewife

Hardware Multithreading

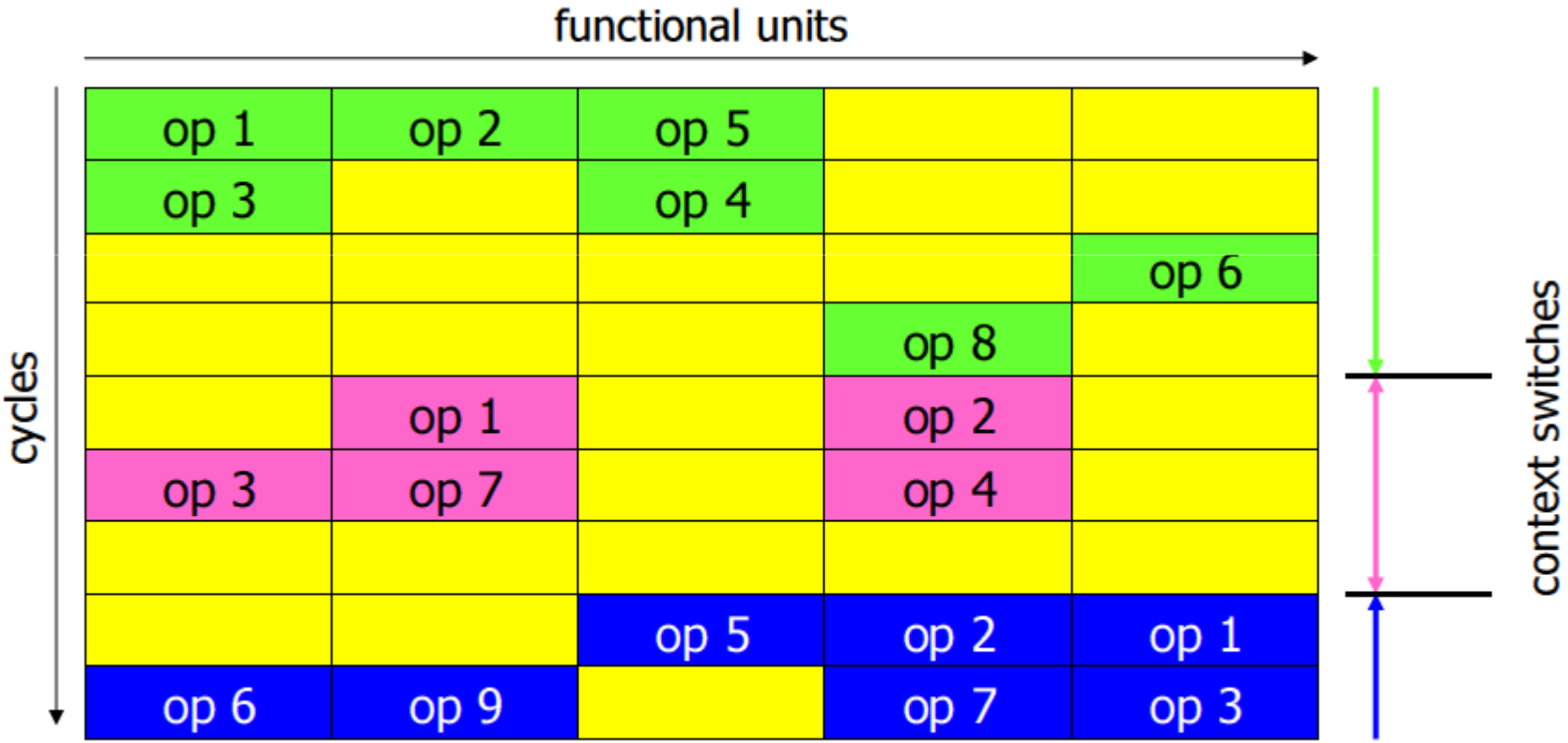
- **Fine-grained** (cycle-by-cycle) multithreading:
  - Round-robin selection between a set of threads

- **Coarse-grained** (block) multithreading: Keep executing a thread until something happens
  - Long latency instruction found
  - Some indication of scheduling difficulties
  - Maximum number of cycles per thread executed

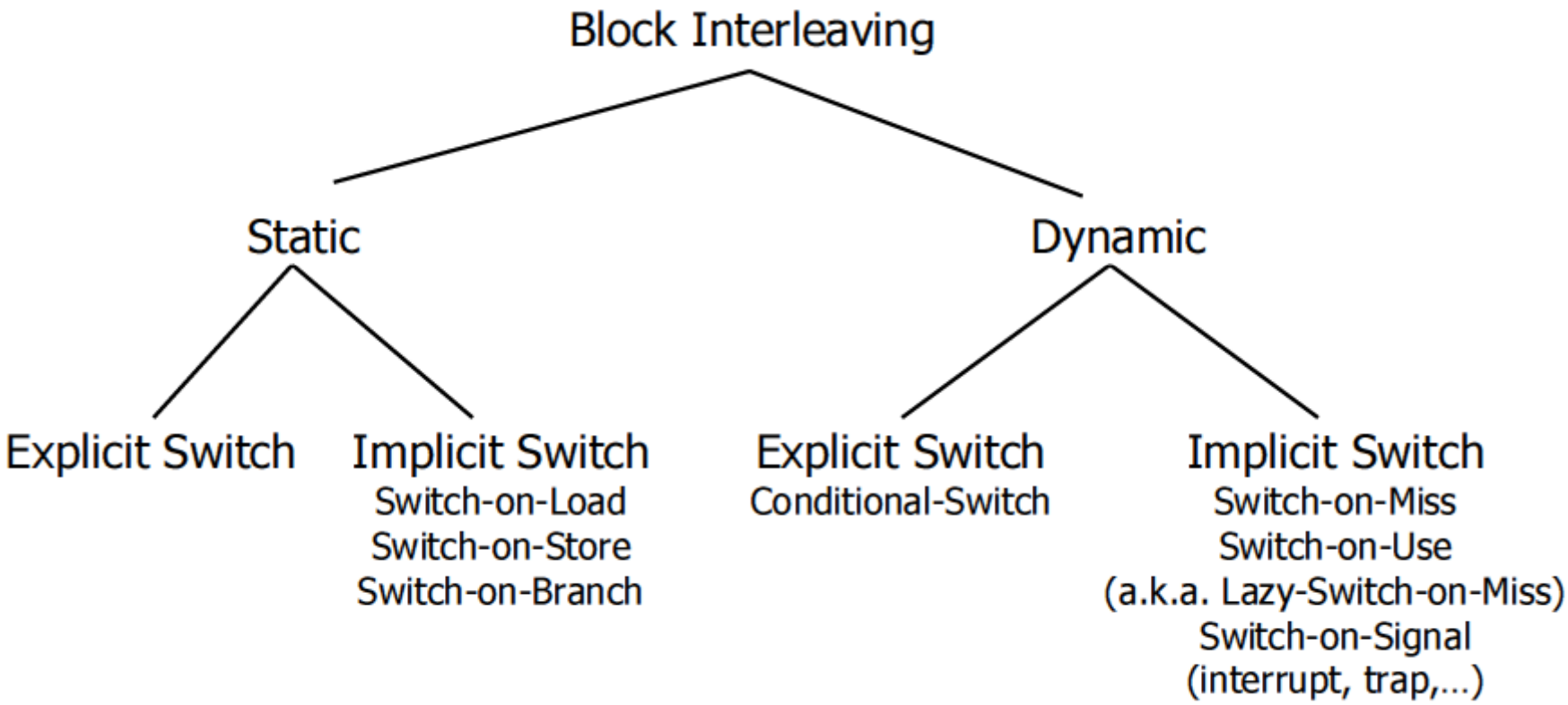
Fine-grained multithreading



Coarse-grained multithreading



Coarse-grained multithreading



Pros&Cons of fine-grained MT

- Null time to switch context
  - Multiple Register Files
- No need for forwarding paths if supported threads are more than pipeline depth!
  - Simpler hardware
- Fills well *short* vertical waste

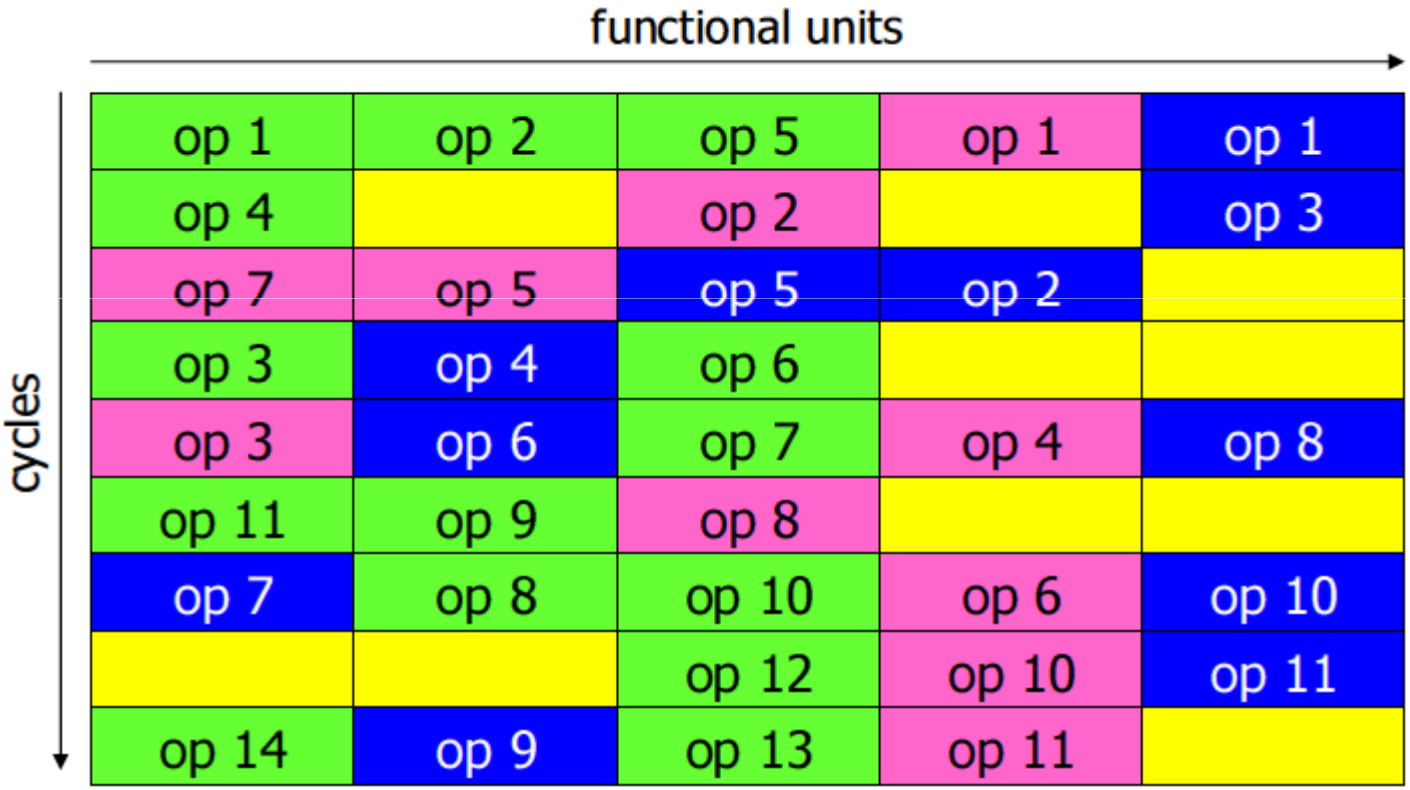
- Fills much less well *long* vertical waste
- Does not reduce significantly horizontal waste (per thread, the instruction window is not much different…)
- Significant deterioration of single thread job

Pros&Cons of coarse-grained MT

- More time allowable for context switch
- Fills very well *long* vertical waste (other threads come in)

- Fills poorly *short* vertical waste (if not sufficient to switch context)
- Does not reduce almost at all horizontal waste
- Scheduling of threads not self-evident:
  - What happens of thread #2 if thread #1 executes perfectly well and leaves no gap?
  - Explicit techniques require ISA modifications → Bad…

Simultaneous Multithreading (SMT)



Thread scheduling

- Allow a *preferred thread* for maintaining single-thread performance

- Prioritised scheduling
  - Thread #0 schedules freely
  - Thread #1 is allowed to use #0 empty slots
  - Thread #2 is allowed to use #0 and #1 empty slots, etc.

- Fair scheduling
  - All threads compete for resources
  - If several threads want the same resource, round-robin assignment

Hardware support for SMT

- Fits well on top of an *ordinary superscalar* processor organization
- Multiple program counters (= threads) and a policy for the fetch units to decide which threads to fetch
- Multiple or larger register file(s) with at least as many registers as logical registers for all threads
- Multiple instruction retirement (e.g., per thread squashing)

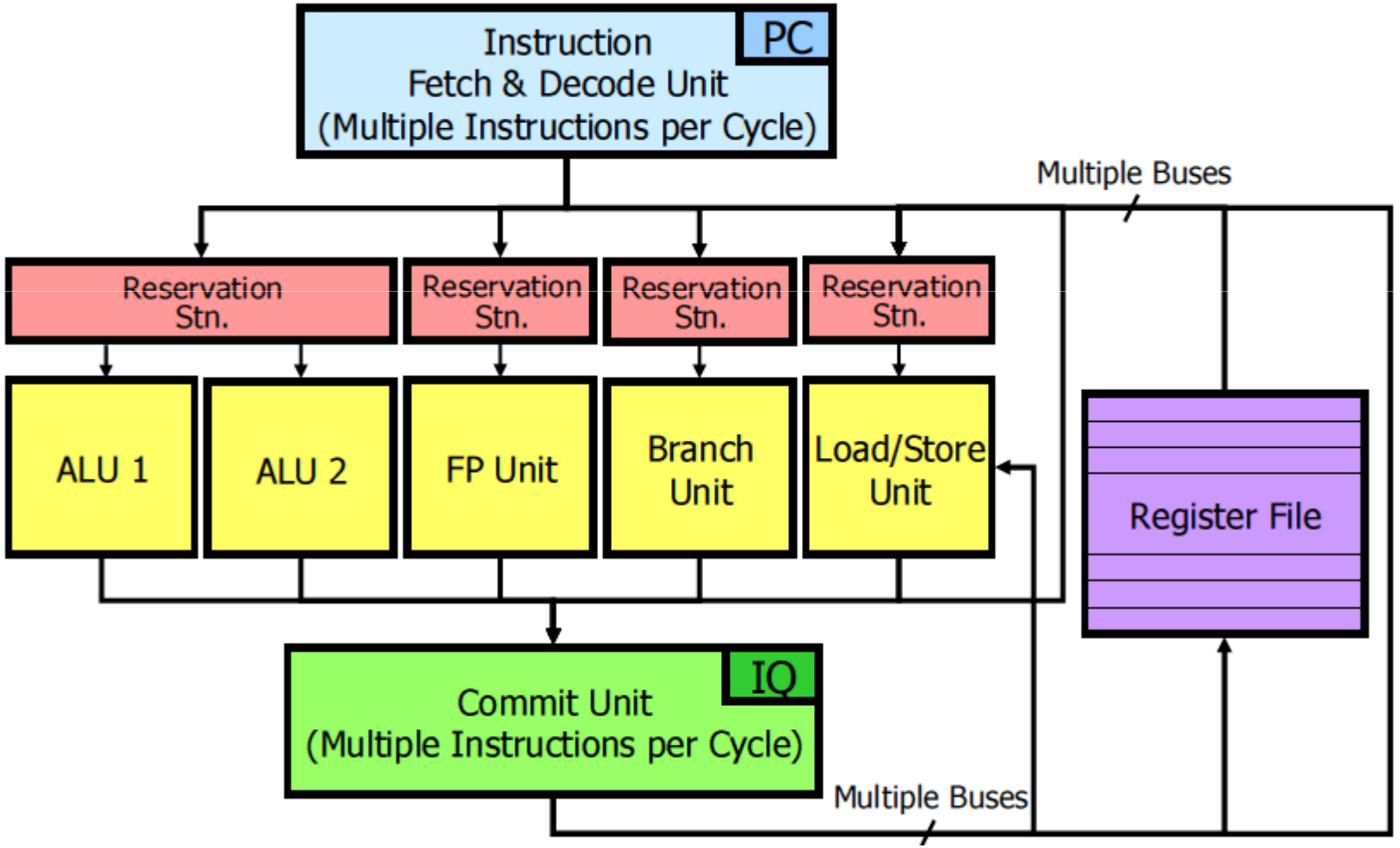
  → **No changes needed in the execution path**

  and also:
- Thread-aware branch predictors (BTBs, etc.)
- per-thread Return Address Stacks

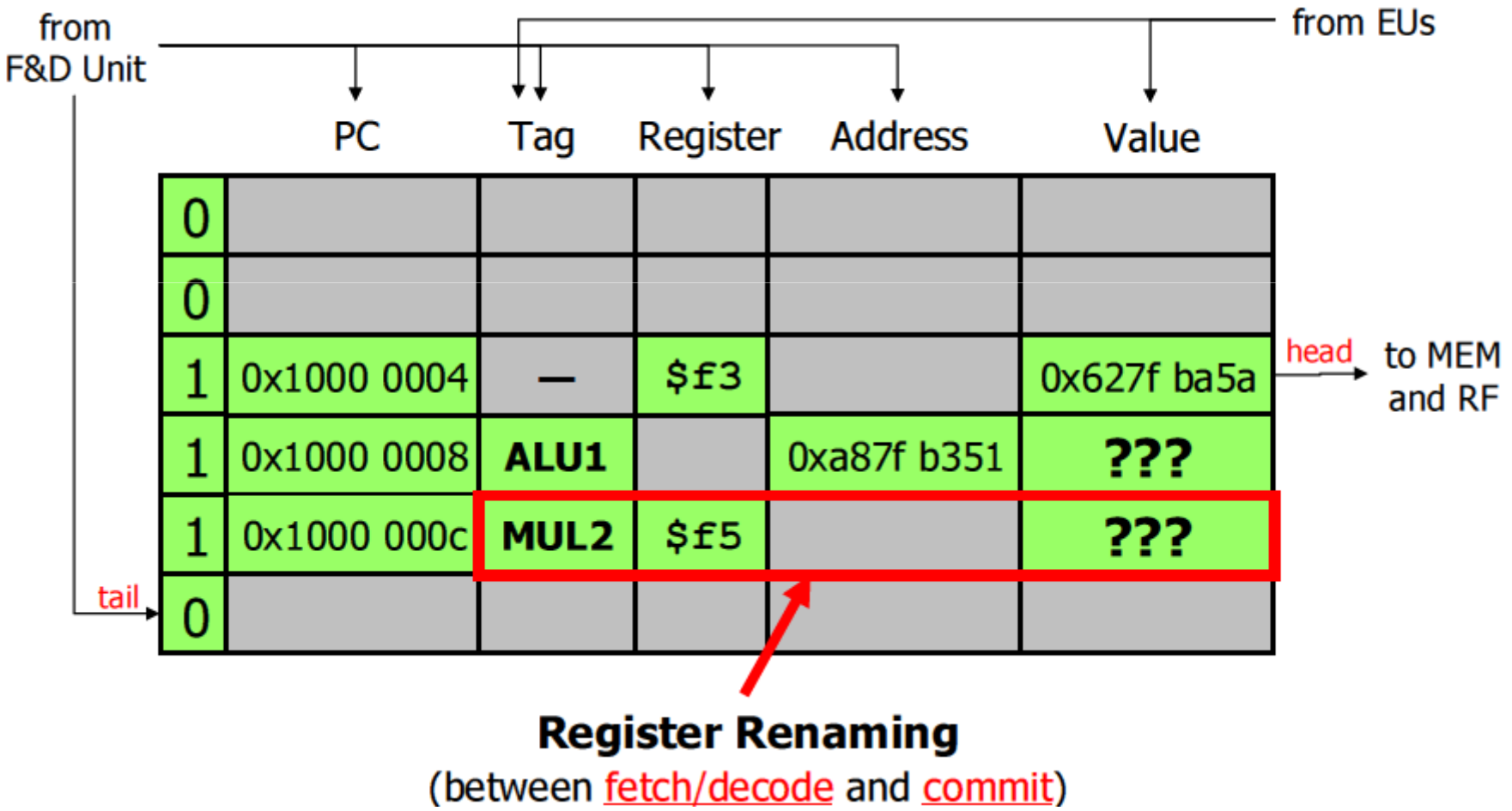
Hardware support for SMT

- Complication of instruction commit
  - We want istructions from separate threads to be allowed to commit independently
  - Use *logically separate* ReorderBuffers
- Dealing with larger register files needed to hold multiple contexts → potentially slower hardware
- Cache “conflicts” between threads may cause some performance degradation in memory access

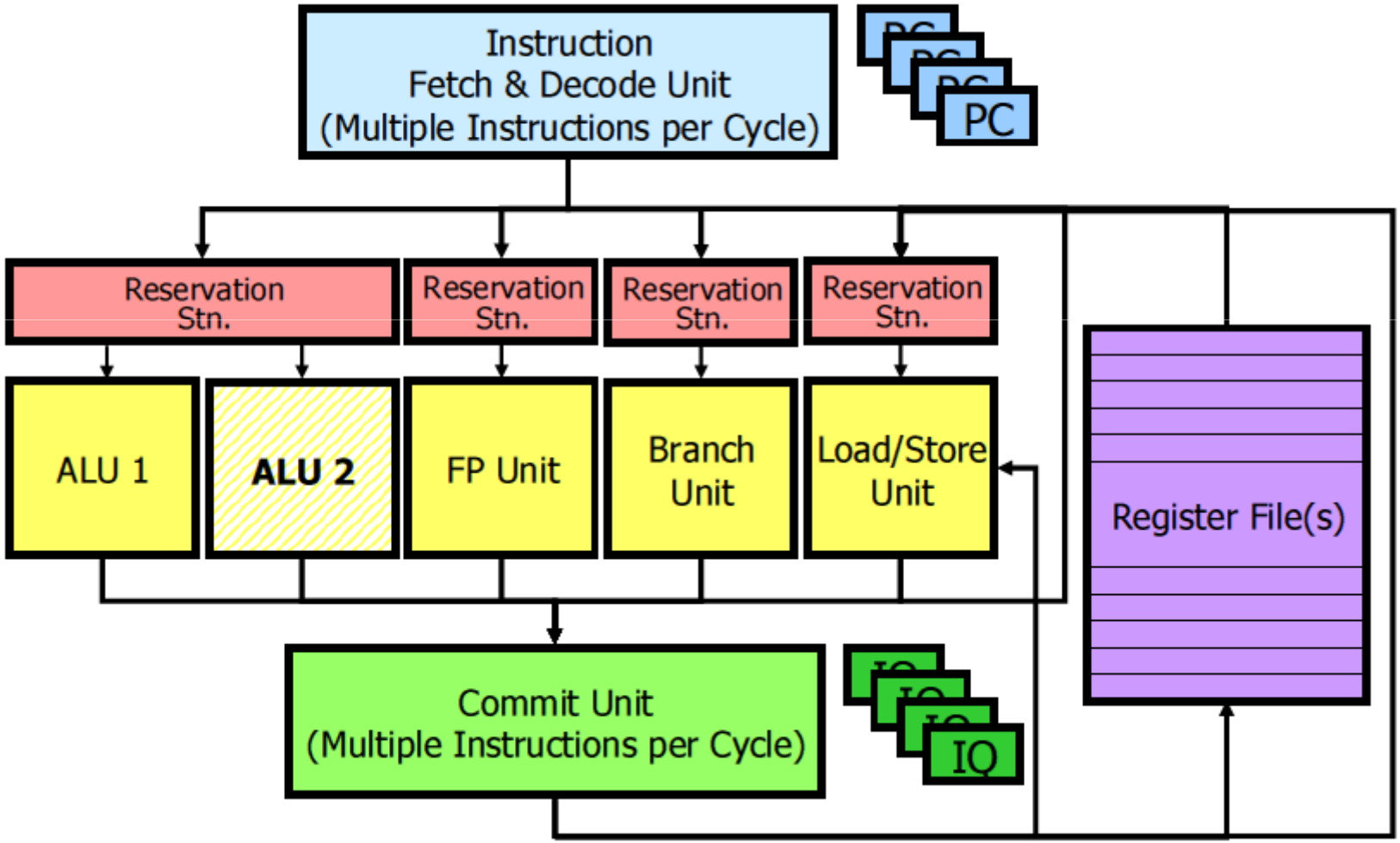
Base superscalar organization



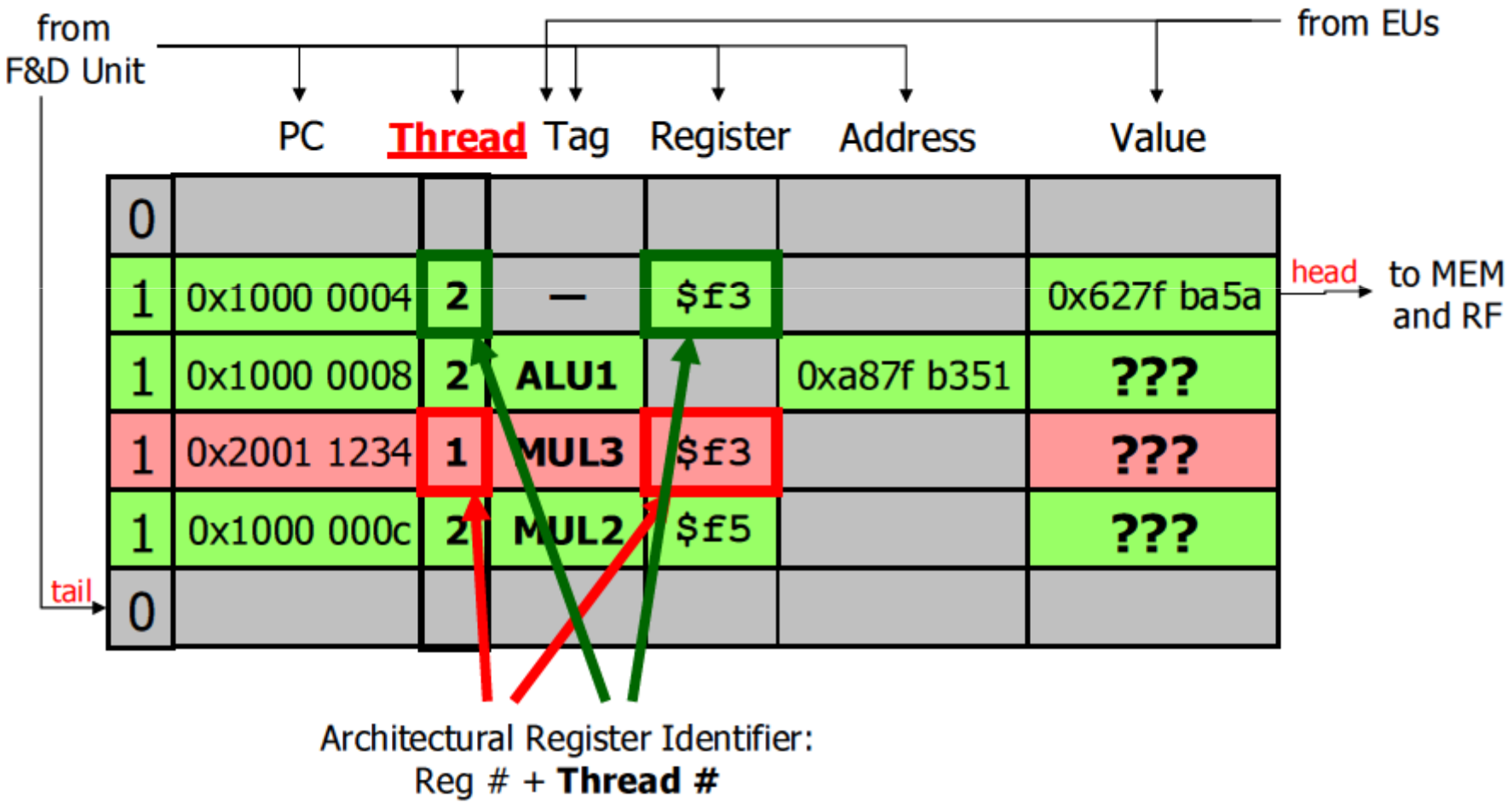
Base reorder buffer



Superscalar processor with SMT

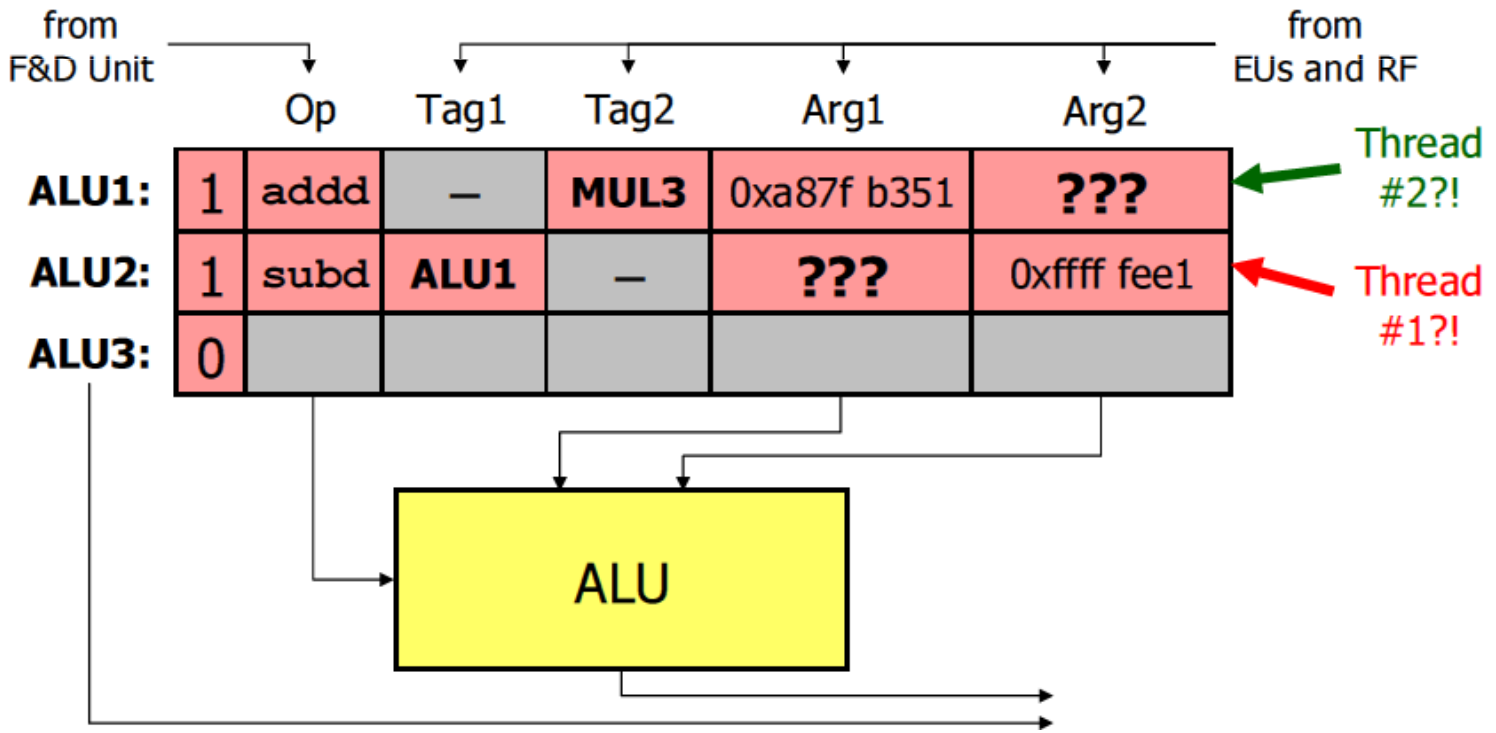


Reorder buffer with SMT



Reservation stations with SMT

- Reservation stations do not need to know which thread an instruction belongs to
- Operand sources are renamed—physical regs, tags, etc.



Implementation of SMT

- Instruction scheduling not more complex
- Register File datapaths not more complex (but much larger register file!)
- Instruction Fetch Throughput is attainable even without more fetch bandwidth
- Unmodified cache and branch predictors are appropropriate also for SMT
- SMT achieves better results than aggressive superscalar

Implementation of SMT

- **Static** fetch solutions: Round-robin
  - Each cycle 8 instructions from 1 thread
  - Each cycle 4 instructions from 2 threads, 2 from 4,…
  - Each cycle 8 instructions from 2 threads, and forward as many as possible from #1 then when long latency instruction in #1 pick rest from #2
- **Dynamic** fecth solutions: Check execution queues!
  - Favour threads with minimal # of in-flight branches
  - Favour threads with minimal # of outstanding misses
  - Favour threads with minimal # of in-flight instructions
  - Favour threads with instructions far from queue head

Implementation of SMT

- Issue policy is not exactly the same as in superscalars…
  - In superscalar: oldest is the best (least speculation, more dependent ones waiting, etc.)
  - In SMT not so clear: branch-speculation level and optimism (cache-hit speculation) vary across threads
- One can think of many selection strategies:
  - Oldest first
  - Cache-hit speculated last
  - Branch speculated last
  - Branches first…

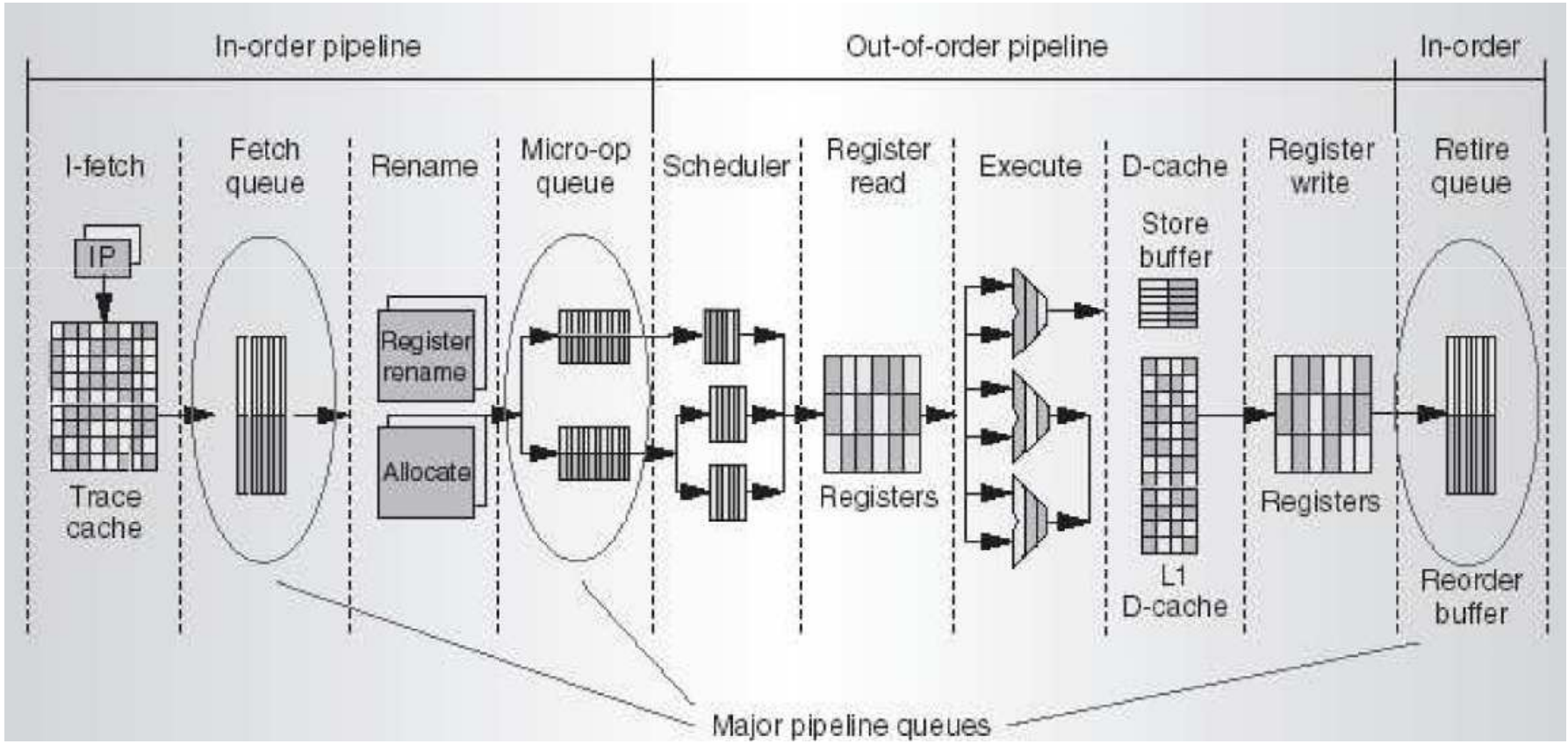
Commercial solutions with SMT

- Compaq Alpha 21464 (EV8)
  - 4T SMT
  - Project killed June 2001
- Intel Pentium IV (Xeon)
  - 2T SMT
  - Introduced in 2002
  - 10-30% gains expected
- SUN Ultra III
  - 2-core CMP, 4T SMT

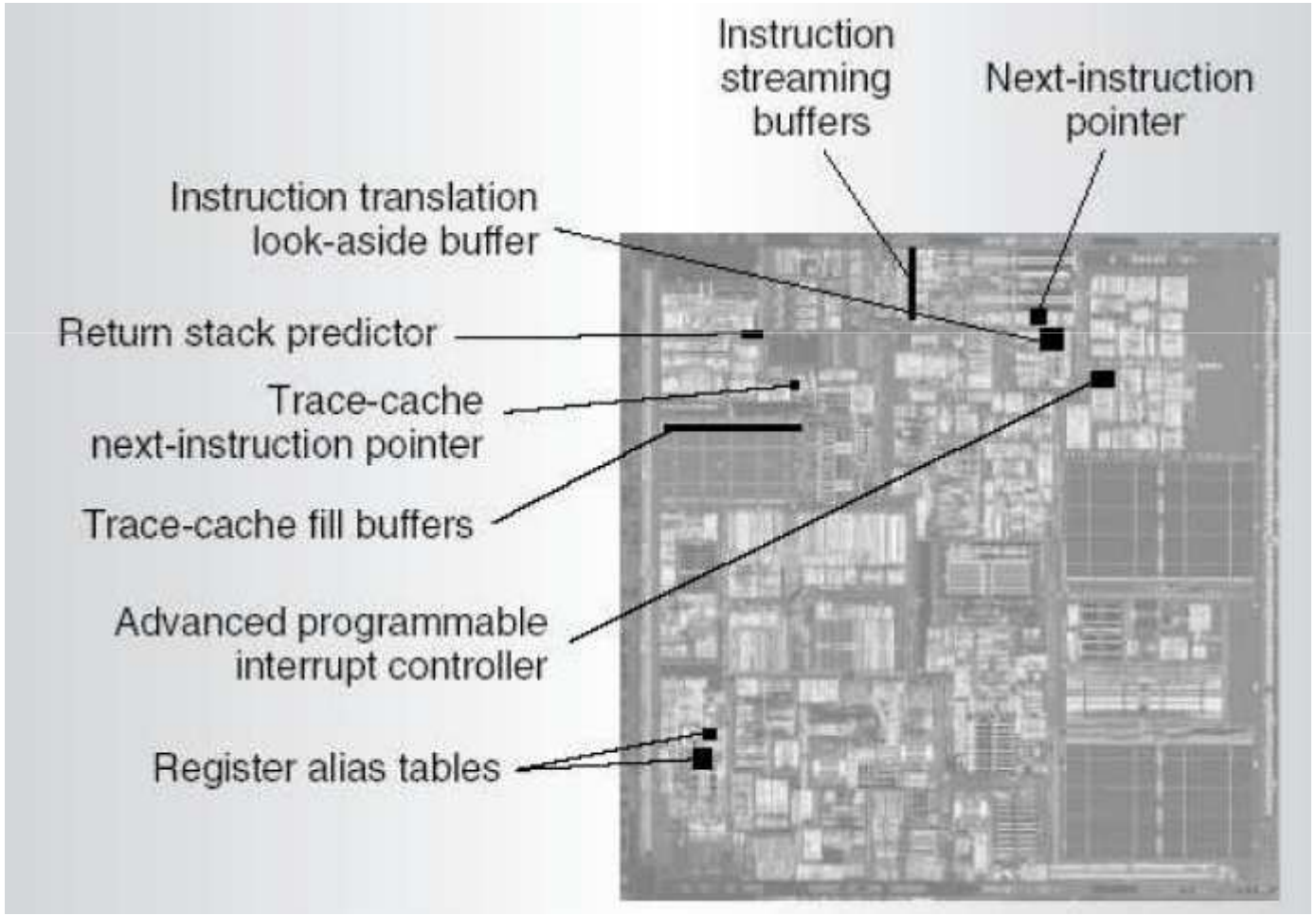
Intel P4 Xeon HyperThreading

- From a software perspective, OSs and user programs can schedule processes or threads to logical processors as they would in a multiprocessor system
- The duplicated resources are primarily:
  - Next–Instruction Pointer
  - Instruction Stream Buffers
  - Instruction translation look–aside buffer
  - Return stack predictor
  - Trace–cache plus local next–instruction pointer
  - Trace–cache fill buffers
  - Advanced Programmable Interrupt Controller (APIC)
  - Register Alias Tables

Intel P4 Xeon HyperThreading

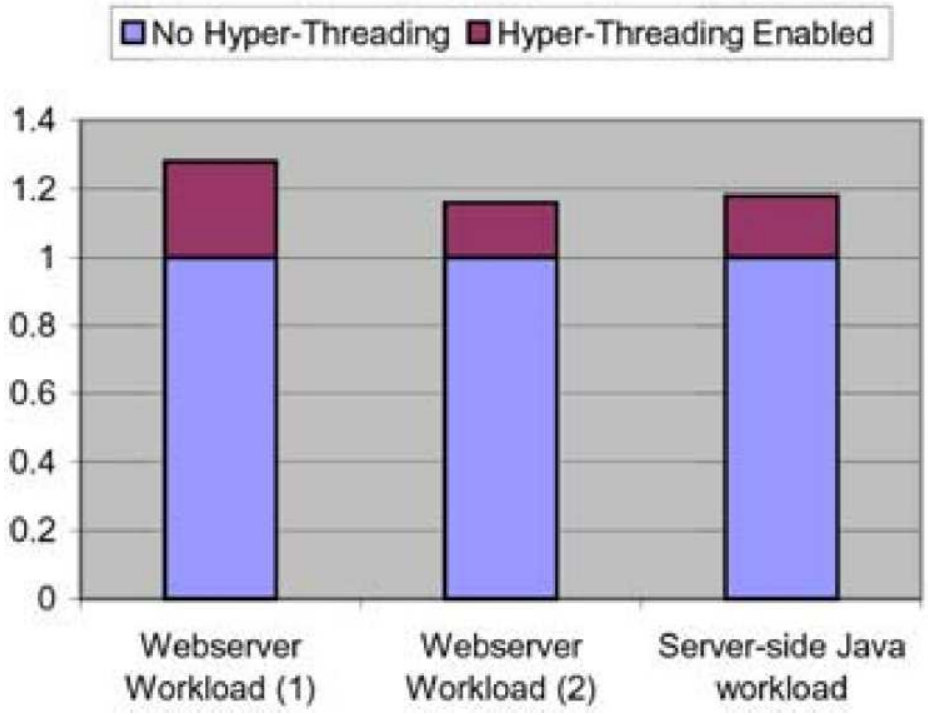


Intel P4 Xeon HyperThreading



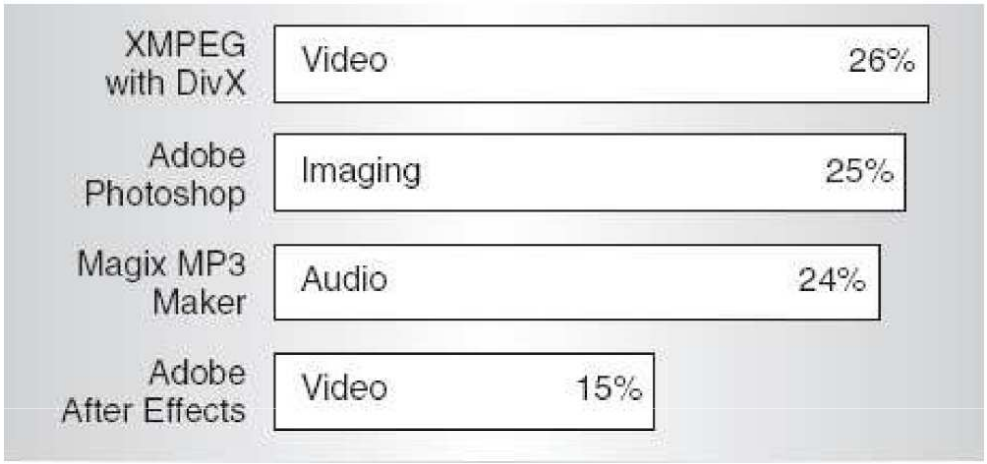
Intel Xeon Hyper-Threading

- Minimum additional cost: SMT = approx. 5% area
- No impact on single-thread performance
  - Recombine partitioned resources
- Fair behaviour with 2 threads



Intel P4 Xeon HyperThreading

- Performance results
  - Multithread benchmarks
  - 15% to 26% performance boost



  - Multitask benchmarks
  - 15% to 27% performance boost