Arquitectura de Computadores

TEMA 3

Lanzamiento múltiple, Límites de ILP, Multithreading

DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

Curso 2012-2013

Contenidos

- Introducción: CPI < 1
- Lanzamiento múltiple de instrucciones: Superescalar, VLIW
- Superescalar simple
- VLIW
- Superescalar con planificación dinámica
- Límites de ILP
- Ejemplo: Implementaciones X86
- Thread Level Parallelism y Multithreading

- Bibliografía
  - Capítulo 3 y Apéndice H de [HePa12]
  - Capítulos 6 y 7 de [SiFK97]

Más ILP: Lanzamiento múltiple

❑ Introducción

- ¿ Por que limitar a una instrucción por ciclo?
- Objetivo: CPI < 1
- Lanzar y ejecutar ***simultáneamente*** múltiples instrucciones por ciclo
- ¿Tenemos recursos?
  - Más área de silicio disponible
  - Técnicas para resolver las dependencias de datos (*planificación*)
  - Técnicas para resolver las dependencias de control (*especulación)*

Más ILP: Lanzamiento múltiple

❑ Alternativas

- ❑ Procesador Superescalar con planificación estática
- ❑ Procesador Superescalar con planificación dinámica+(especulación)
- ❑ Procesadores VLIW (very long instruction processor)

- ✔ **Superescalar**
  - ✔ Lanza de 1 a 8 instrucciones por ciclo
  - ✔ Reglas de ejecución
    - o Ejecución en orden-planificación estática
    - o Ejecución fuera de orden-planificación dinámica
- ✔ **VLIW**
  - ✔ Numero fijo de instrucciones por ciclo
  - ✔ Planificadas estáticamente por el compilador
  - ✔ EPIC (Explicitly Parallel Instruction Computing) Intel/HP

Más ILP: Lanzamiento múltiple

❑ Alternativas

Tipo	Forma del "issue"	Detección de riesgos	Planificación	Ejemplos
Superescalar estático	Dinámico	HW	estática	Embeded MIPS, ARM
Superescalar dinámico	Dinámico	HW	dinámica	ninguno
Superescalar especulativo	Dinámico	HW	Dinámica con especulación	P4, Core2, Power5, 7 SparcVI, VII
VLIW	Estático	Básicamente SW	estática	TI C6x Itanium

Más ILP: Lanzamiento múltiple

❑ SUPERESCALAR

Grado 2
2 Vías

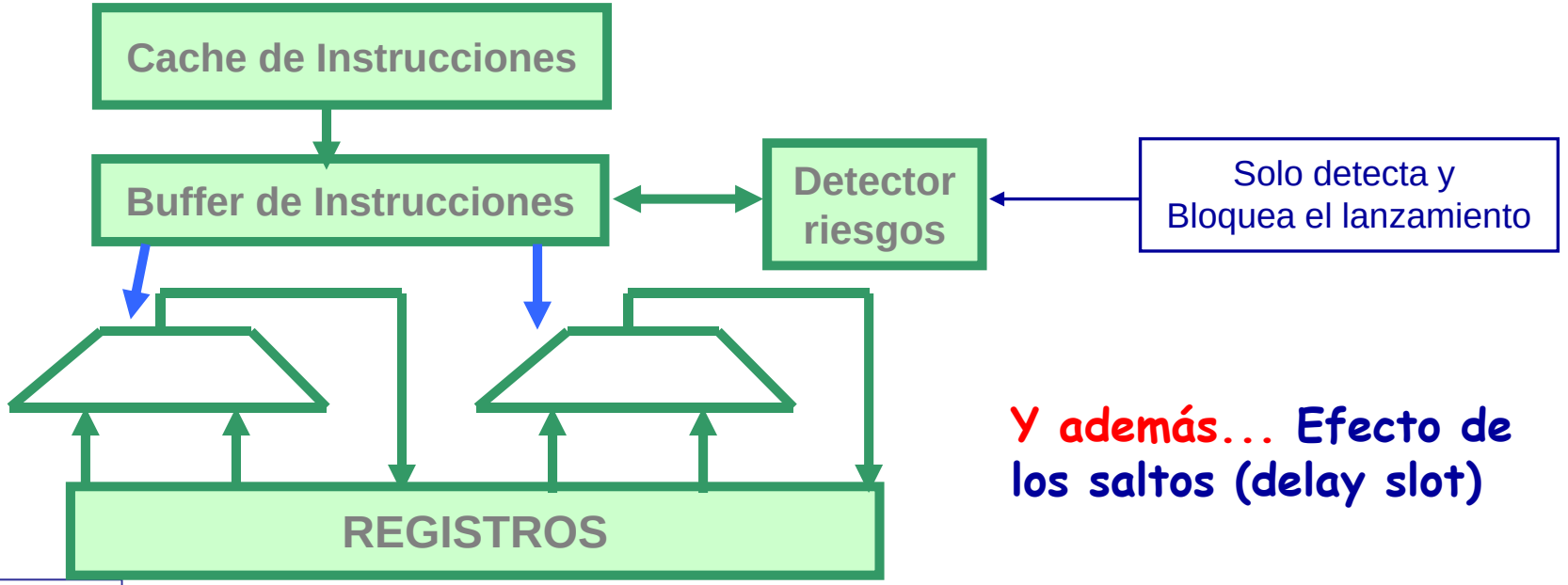
Instruction	1	2	3	4	5	6	7
i	IF	ID	EX	MEM	WB		
i+1	IF	ID	EX	MEM	WB		
i+2		IF	ID	EX	MEM	WB	
i+3		IF	ID	EX	MEM	WB	
i+4			IF	ID	EX	MEM	WB
i+5			IF	ID	EX	MEM	WB

- **Duplicar todos los recursos:**
  - Puertas bloque de Registros
  - Fus
  - Puertas de memoria,..
  - Control
- **Ideal CPI= 0.5 se reduce por:**
  - Saltos
  - LOADs
  - Dependencias verdaderas LDE
- **Necesita:**
  - Predicción sofisticada
  - Tratamiento de LOAD; Cargas especulativas, técnicas de prebúsqueda
- **Más presión sobre la memoria**
- **Efecto incremental de los riesgos**
- **Se puede reducir complejidad con limitaciones (Un acceso a memoria por ciclo)**

Más ILP: Lanzamiento múltiple

❑ SUPERESCALAR Simple (estático, en orden)

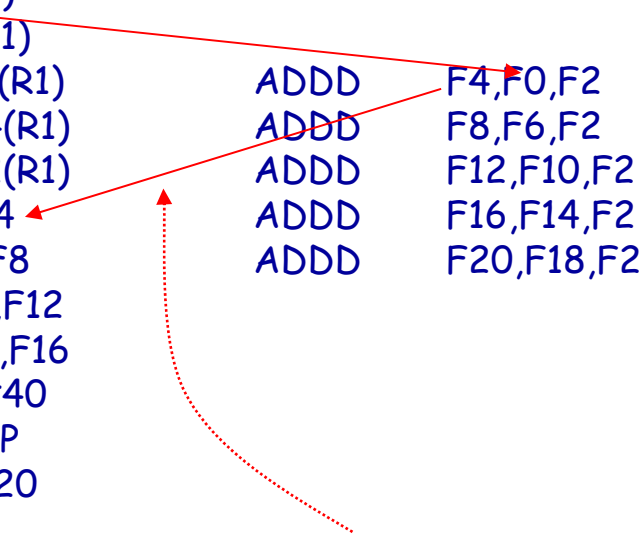
- Regla de lanzamiento: Una instrucción FP (2ª)+ una instrucción de cualquier otro tipo (1ª)
- Buscar y decodificar dos instrucciones por ciclo (64 bits)
  - Ordenamiento y decodificación
  - Se analizan en orden. Sólo se lanza la 2ªsi se ha lanzado la 1ª (conflictos)
- Unidades funcionales segmentadas (una ope. por ciclo) ó múltiples (división, raíz), más puertas en el bloque de registros
- Lanzamiento simple, recursos no conflictivos (diferentes reg y UF,..), excepto
  - Conflictos de recursos; load, store, move FP → más puertas en el bloque de reg.
  - Conflictos de datos LDE → más distancia entre instrucciones.



Y además... Efecto de los saltos (delay slot)

❑SUPERESCALAR Simple (estático, en orden)

	Instrucción entera		Instrucción FP		Ciclo
Loop:	LD	F0,0(R1)			1
	LD	F6,-8(R1)			2
	LD	F10,-16(R1)	ADDD	F4,F0,F2	3
	LD	F14,-24(R1)	ADDD	F8,F6,F2	4
	LD	F18,-32(R1)	ADDD	F12,F10,F2	5
	SD	0(R1),F4	ADDD	F16,F14,F2	6
	SD	-8(R1),F8	ADDD	F20,F18,F2	7
	SD	-16(R1),F12			8
	SD	-24(R1),F16			9
	SUBI	R1,R1,#40			10
	BNEZ	R1,LOOP			11
	SD	8(R1),F20			12



- **Desarrollo para ejecución superescalar: se desarrolla una iteración más.**
  12 ciclos, 2.4 ciclos por iteración

- **El código máquina está compacto en la memoria**

Más ILP: Lanzamiento múltiple

❑ SUPERESCALAR Simple (estático, en orden)

➢ *Ventajas*

- No modifica código. Compatibilidad binaria
- No riesgos en ejecución

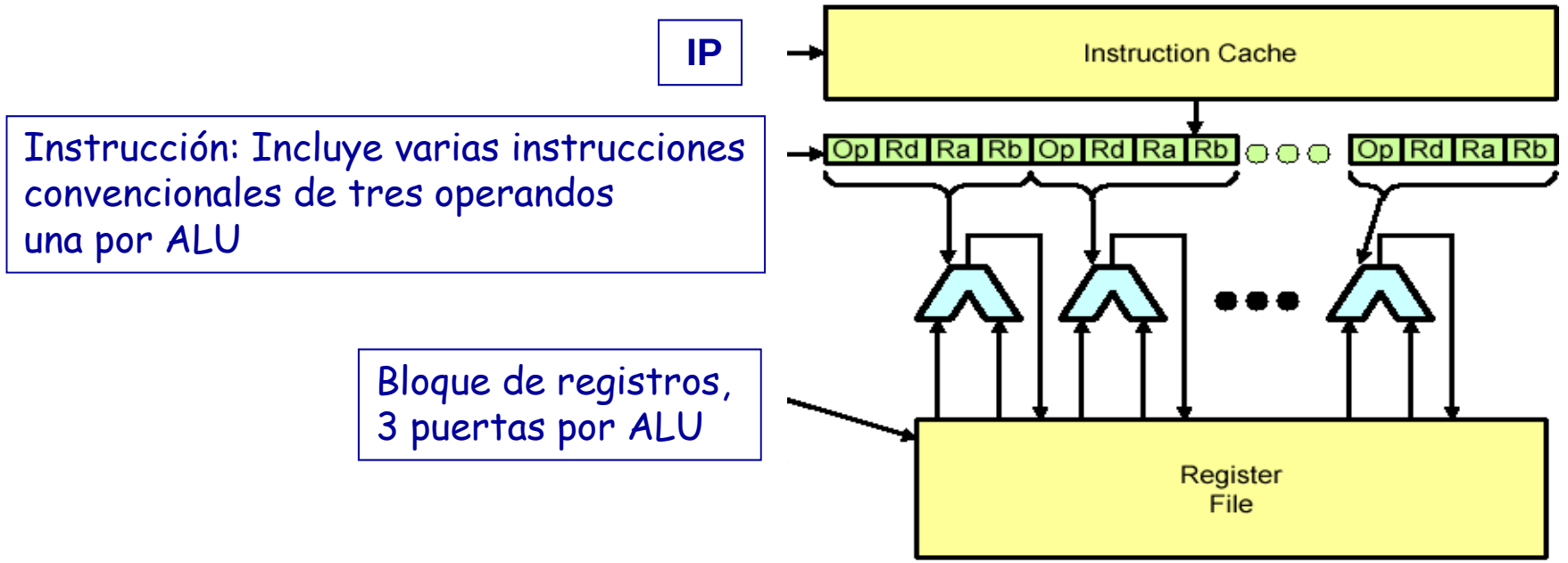
➢ *Desventajas*

- Mezcla de instrucciones. Solo obtiene CPI de 0.5 en programas con 50 % de FP
- Bloqueos en el lanzamiento
- Planificación fija: No puede adaptarse a cambios en ejecución (Fallos de cache)
- Los códigos deben de ser replanificados para cada nueva implementación (eficiencia)

Más ILP: Lanzamiento múltiple

❑ VLIW

- El análisis de dependencias en tiempo de compilación
- Muchas operaciones por instrucción (IA64 packet, Transmeta molecula)
- Todas las operaciones de una instrucción se ejecutan en paralelo
- Instrucciones con muchos bits
- Muchas operaciones vacías (NOP)

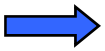


Más ILP: Lanzamiento múltiple

❑ VLIW Ejemplo Tema 2

```
LOOP    LD      F0,0(R1)
        ADDD    F4,F0,F2
        SD      0(R1),F4
        SUBI    R1,R1,#8
        BNEZ    R1,LOOP
```

- Aplicar técnicas conocidas para minimizar paradas
  - Unrolling
  - Renombrado de registros



```
LOOP:   LD      F0, 0(R1)
        LD      F6, -8(R1)
        LD      F10, -16(R1)
        LD      F14,-24(R1)
        ADDD    F4, F0, F2
        ADDD    F8, F6, F2
        ADDD    F12, F10, F2
        ADDD    F16, F14, F2
        SD      0(R1), F4
        SD      -8(R1), F8
        SUBI    R1, R1, #32
        SD      16(R1), F12
        BNEZ    R1, LOOP
        SD      8(R1), F16;   8-32 = -24
```

- Latencias de uso: LD a ADD 1 ciclo, ADD a SD 2 ciclos
- Opción: desarrollar 4 iteraciones y planificar: 14 ciclos, 3.5 ciclos por iteración

Más ILP: Lanzamiento múltiple

❑ VLIW

Loop unrolling en VLIW (desarrollo 7 iteraciones)

```
LOOP:   LD F0,0(R1)       ; F0 = array element
        ADDD F4,F0,F2     ; add scalar in F2
        SD 0(R1),F4       ; store result
        SUBI R1,R1,#8     ; decrement pointer
        BNEZ R1, LOOP     ; branch if R1!=0
```

Mem ref 1	Mem ref 2	FP op	FP op	Int op/branch
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,-24(R1)			
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2	
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2	
		ADDD F20,F18,F2	ADDD F24,F22,F2	
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2		
SD -16(R1),F12	SD -24(R1),F16			SUBI R1,R1,#56
SD 24(R1),F20	SD 16(R1),F24			
SD 8(R1),F28				BNEZ R1, LOOP

- ✓ **7 iteraciones en 9 ciclos: 1.3 ciclos por iteración**
- ✓ **23 operaciones en 45 slots (~50% de ocupación)**
- ✓ **Muchos registros necesarios**

Más ILP: Lanzamiento múltiple

❑VLIW

VENTAJAS

- Hardware de control muy simple
  - No detecta dependencias
  - Lógica de lanzamiento simple
- Puede explotar paralelismo a todo lo largo del programa

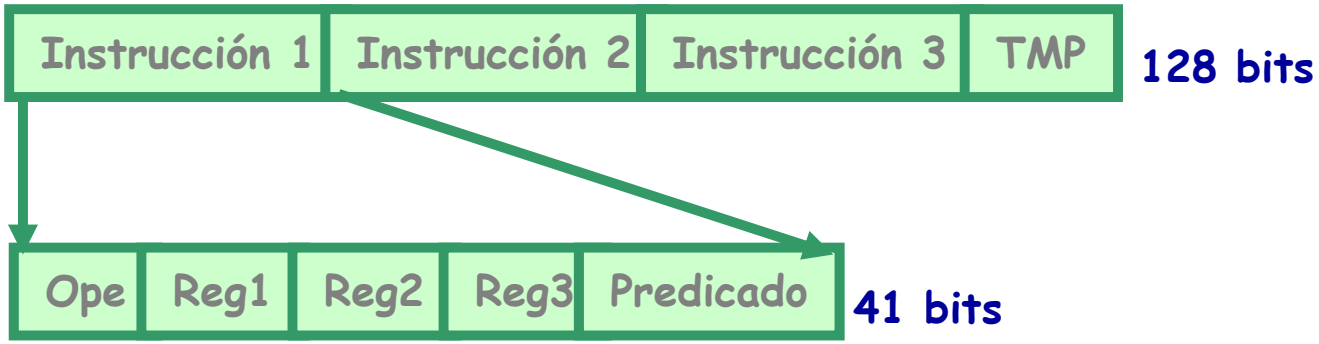
DESVENTAJAS

- Planificación estática; Muy sensible a fallos de cache
- Necesita desenrollado muy agresivo
- Bloque de registros muy complejo en área y tiempo de acceso
- Muchas NOP
  - Poca densidad de código
  - Capacidad y AB de la cache de instrucciones
- Compilador muy complejo
- No binario compatible
- Operación síncrona para todas las operaciones de una instrucción

Más ILP: Lanzamiento múltiple

❑EPIC: Explicitly Parallel Instruction Computing (IA64)

- Instrucciones de 128 bits
  - Operaciones de tres operandos
  - TMP codifica dependencias entre las operaciones
  - 128 registros enteros (64bits), 128 registros FP (82bits)
  - Ejecución predicada. 64 registros de predicado de 1 bit
  - Cargas especulativas
  - Hw para chequeo de dependencias



Primera implementación Itanium (2001), 6 operaciones por ciclo, 10 etapas, 800Mhz

Segunda implementación Itanium2 (2005), 6 operaciones por ciclo, 8 etapas, 1,66Ghz

Más ILP: Lanzamiento múltiple

❑SUPERESCALAR con Planificación Dinámica.Fuera de orden

➢ Un Diseño Simple

- Estaciones de reserva separadas para enteros (+reg) y PF (+reg)
- Lanzar dos instrucciones en orden (ciclo de lanzamiento: partir en dos subciclos)
- Solo FP load causan dependencias entre instrucciones enteras y PF
  - Reemplazar buffer de load con cola. Las lecturas se hacen en orden
  - Ejecución Load: "check" dirección en cola de escritura para evitar LDE
  - Ejecución Store: "check" dirección en cola de lecturas para evitar EDL

➢ Rendimiento del procesador

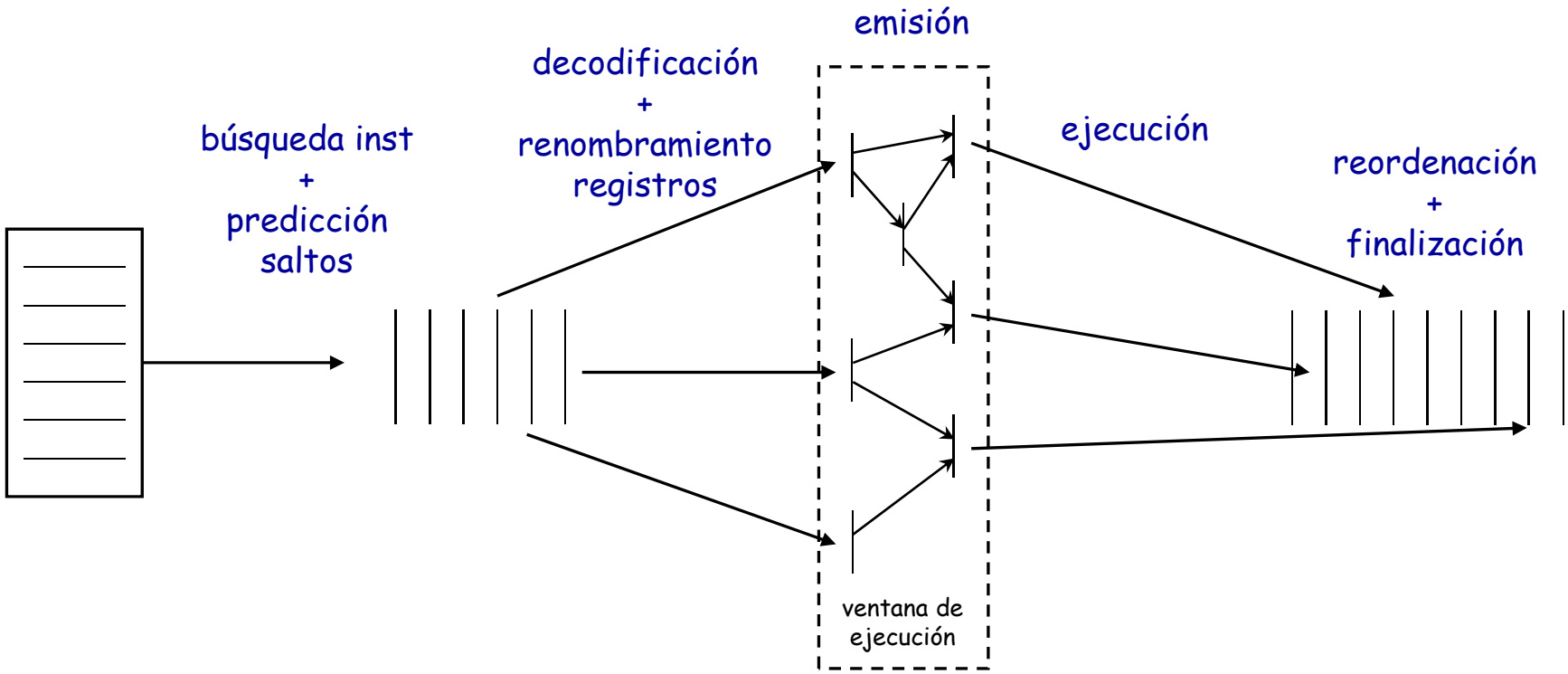
Iteración no.	Instrucción	Lanzada	Comienza Ejecución (número de ciclo)	Escribe resultado	
1	LD F0,0(R1)	1	2	4	
1	ADDD F4,F0,F2	1	5	8	⬅
1	SD 0(R1),F4	2	9		
1	SUBI R1,R1,#8	3	4	5	
1	BNEZ R1,LOOP	4	6		
2	LD F0,0(R1)	5	6	8	⬅
2	ADDD F4,F0,F2	5	9	12	
2	SD 0(R1),F4	6	13		
2	SUBI R1,R1,#8	7	8	9	
2	BNEZ R1,LOOP	8	10		

4 ciclos por iteración

Más ILP: Lanzamiento múltiple

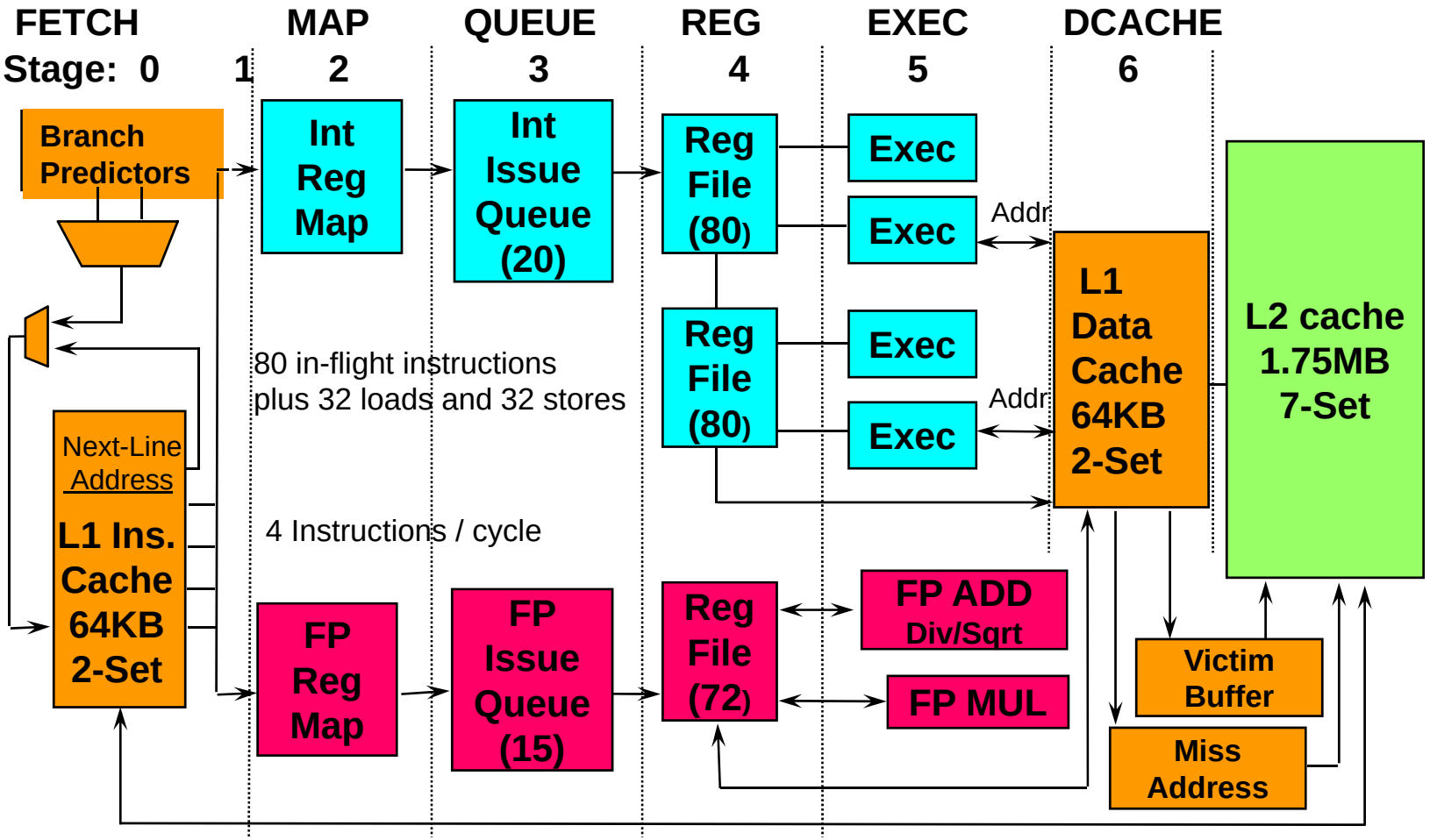
❑ SUPERESCALAR con Planificación Dinámica y Especulación

Ejecución fuera de orden. Finalización en orden



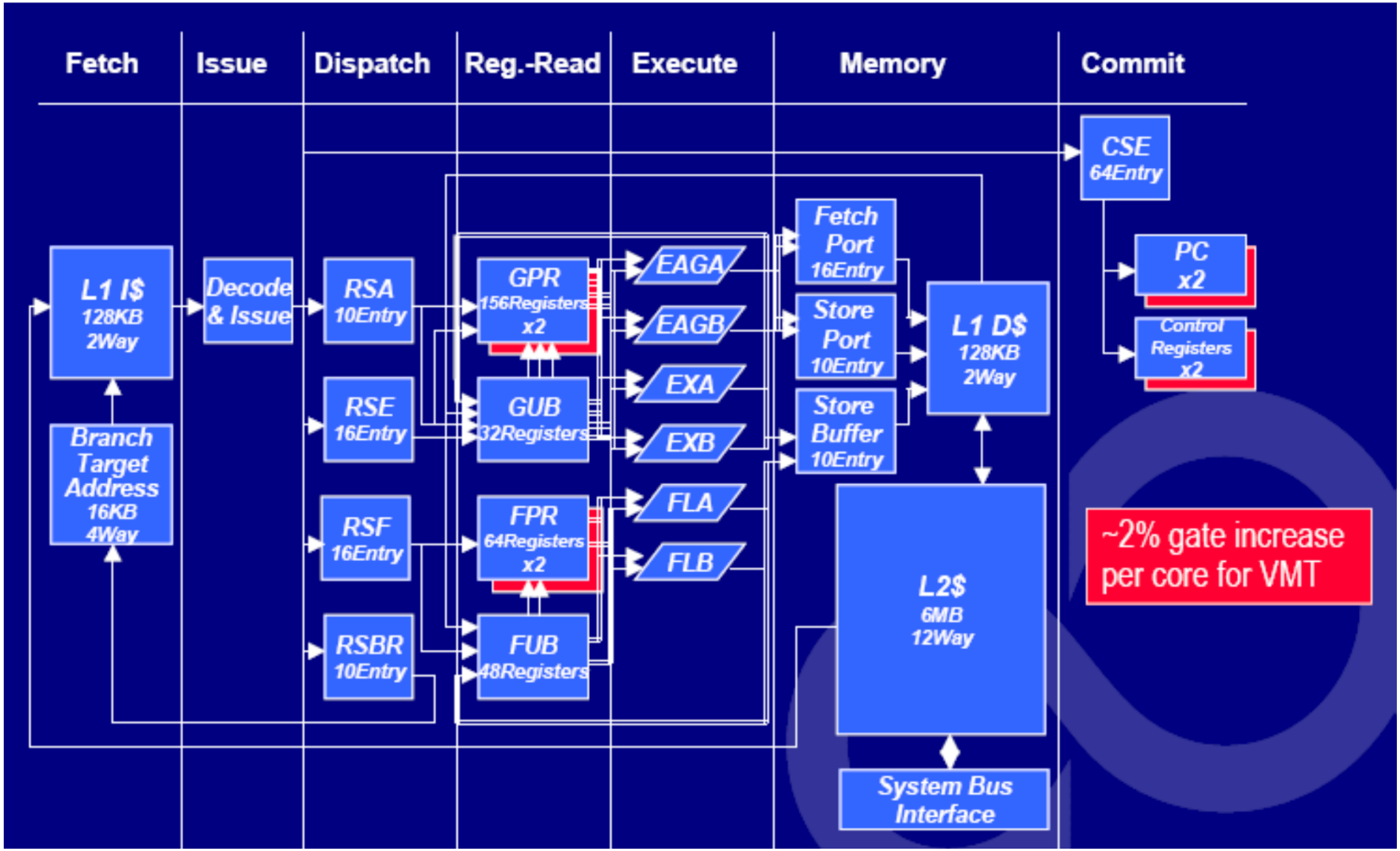
programa estático | **flujo dinámico instrucciones** | **planificación dinámica y ejecución fuera de orden** | **escritura en orden**

❑ EV7 ALPHA 21364 Core (2003)



Más ILP: Lanzamiento múltiple

- SPARC64 VI (2006/7)



Límites del ILP

- El lanzamiento múltiple permite mejorar el rendimiento sin afectar al modelo de programación
- En los últimos años se ha mantenido el mismo ancho superescalar que tenían los diseños del 1995
- La diferencia entre rendimiento pico y rendimiento obtenido crece

- ¿Cuanto ILP hay en las aplicaciones?

- ¿Necesitamos nuevos mecanismos HW/SW para explotarlo?
  - Extensiones multimedia:
    - Intel MMX,SSE,SSE2,SSE3, SSE4
    - Motorola Altivec, Sparc, SGI, HP

Límites del ILP

- ¿Cuanto ILP hay en las aplicaciones?

- Supongamos un procesador superescalar fuera de orden con especulación y con recursos ilimitados
  - Infinitos registros para renombrado
  - Predicción perfecta de saltos
  - Caches perfectas
  - Lanzamiento no limitado
  - Desambiguación de memoria perfecta

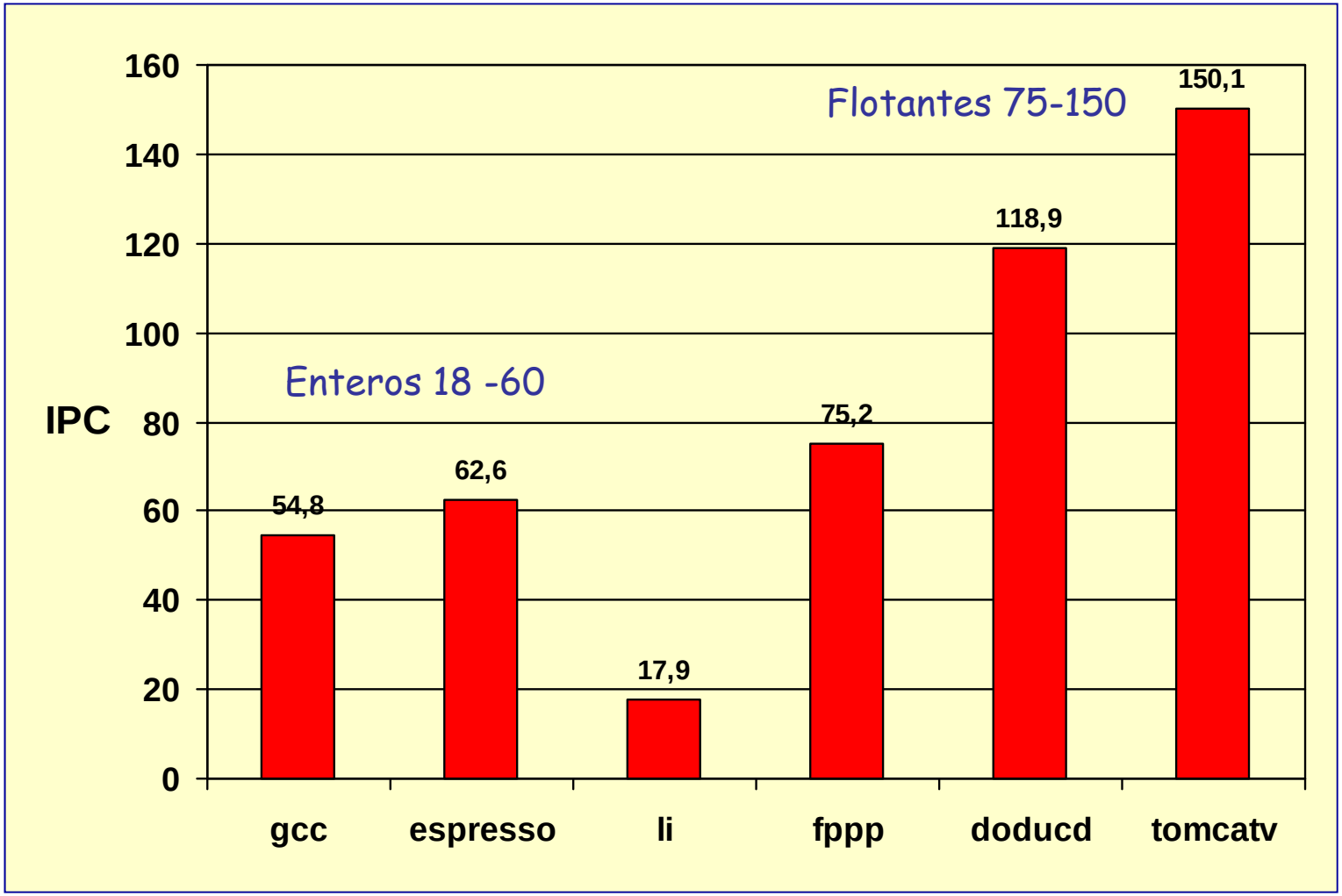
Límites del ILP

- Modelo versus procesador real

	Modelo	Power 5
Instrucciones lanzadas por ciclo	Infinito	4
Ventana de instrucciones	Infinita	200
Registros para renombrado	Infinitos	48 integer + 40 Fl. Pt.
Predicción de saltos	Perfecta	2% to 6% de fallos de predicción (Tournament Branch Predictor)
Cache	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3 (off-chip)
Análisis de Memory Alias	Perfecto	

Límites del ILP

❑ Límite superior: Recursos ilimitados



Algunas aplicaciones tienen poco paralelismo (Li)

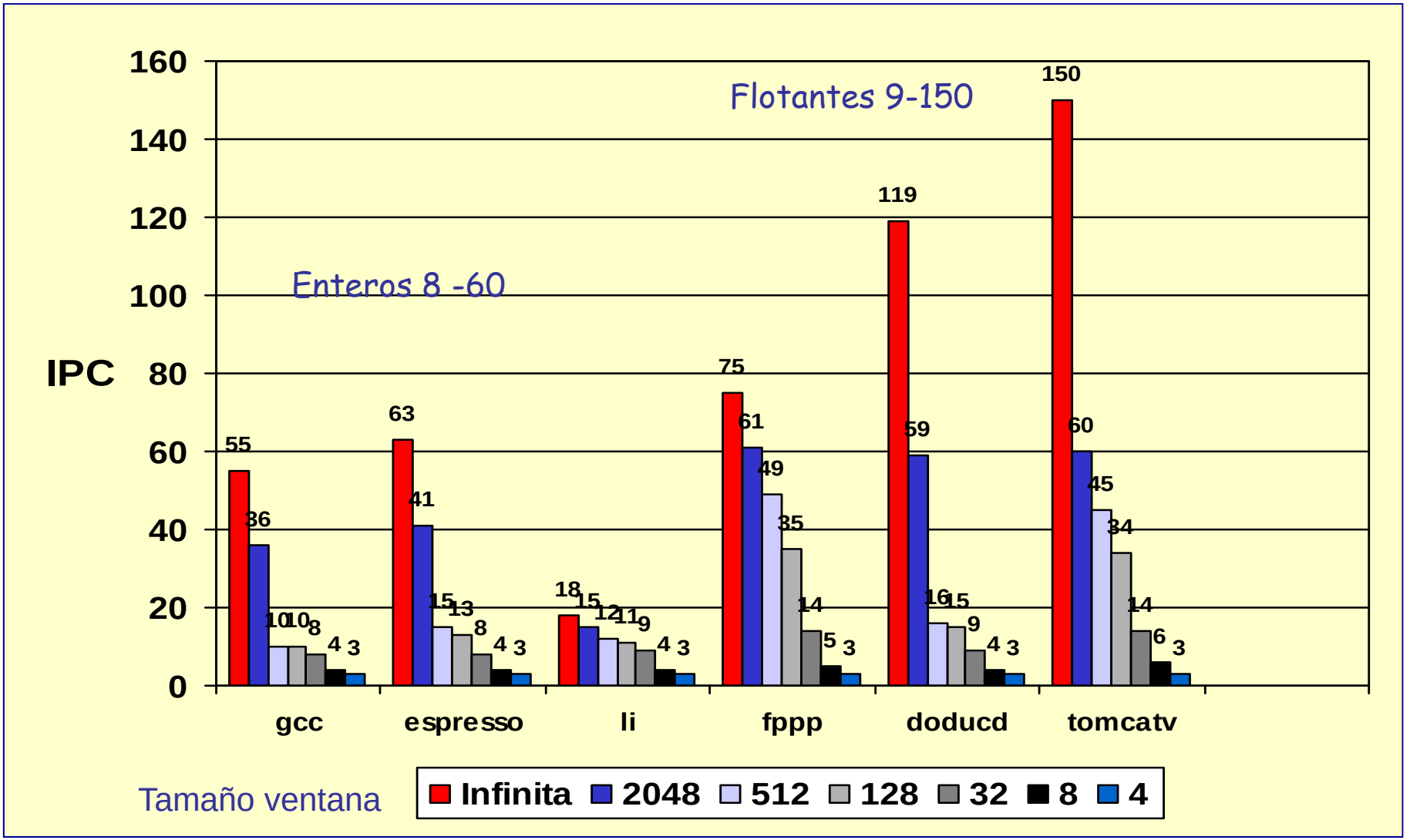
Límites del ILP

- Modelo versus procesador real

	Nuevo Modelo	Modelo	Power 5
Instrucciones lanzadas por ciclo	Infinitas	Infinitas	4
Ventana de instrucciones	Infinito vs. 2K, 512, 128, 32, 8,4	Infinita	200
Registros para renombrado	Infinitos	Infinitos	48 enteros + 40 Fl. Pt.
Predicción de saltos	Perfecta	Perfecta	Tournament
Cache	Perfecta	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	Perfecto	Perfecto	

Límites del ILP

- HW más real: Impacto del tamaño de la ventana de instrucciones



Límites del ILP

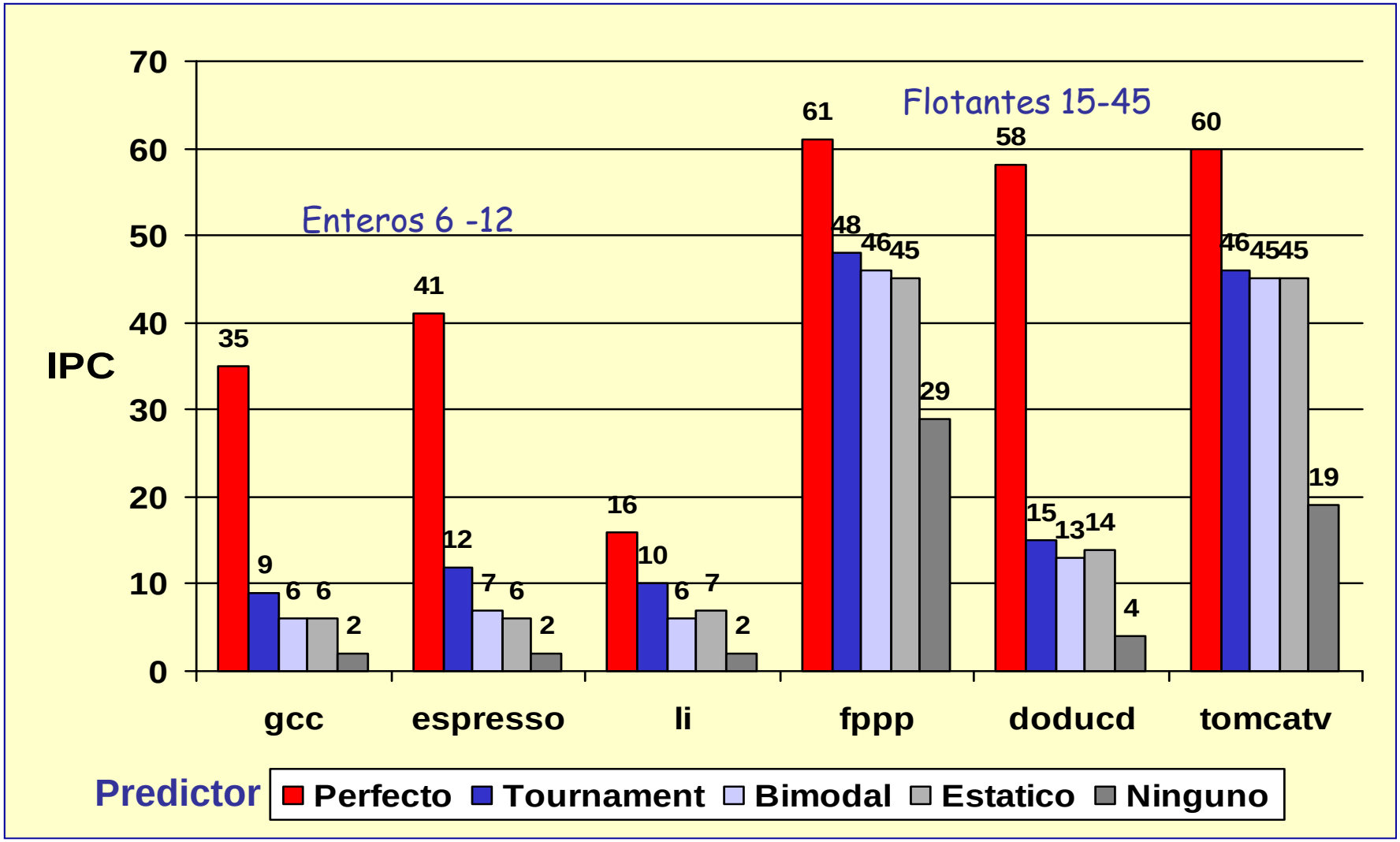
- Modelo versus procesador real

	Nuevo Modelo	Modelo	Power 5
Instrucciones lanzadas por ciclo	64 sin restricciones	Infinitas	4
Ventana de instrucciones	2048	Infinita	200
Registros para renombrado	Infinitos	Infinitos	48 enteros + 40 Fl. Pt.
Predicción de saltos	Perfecto vs. 8K Tournament vs. 512 2-bit vs. profile vs. ninguno	Perfecta	Tournament
Cache	Perfecta	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	Perfecto	Perfecto	

Límites del ILP

❑ HW más real: Impacto de los saltos

Ventana de 2048 y lanza 64 por ciclo



Límites del ILP

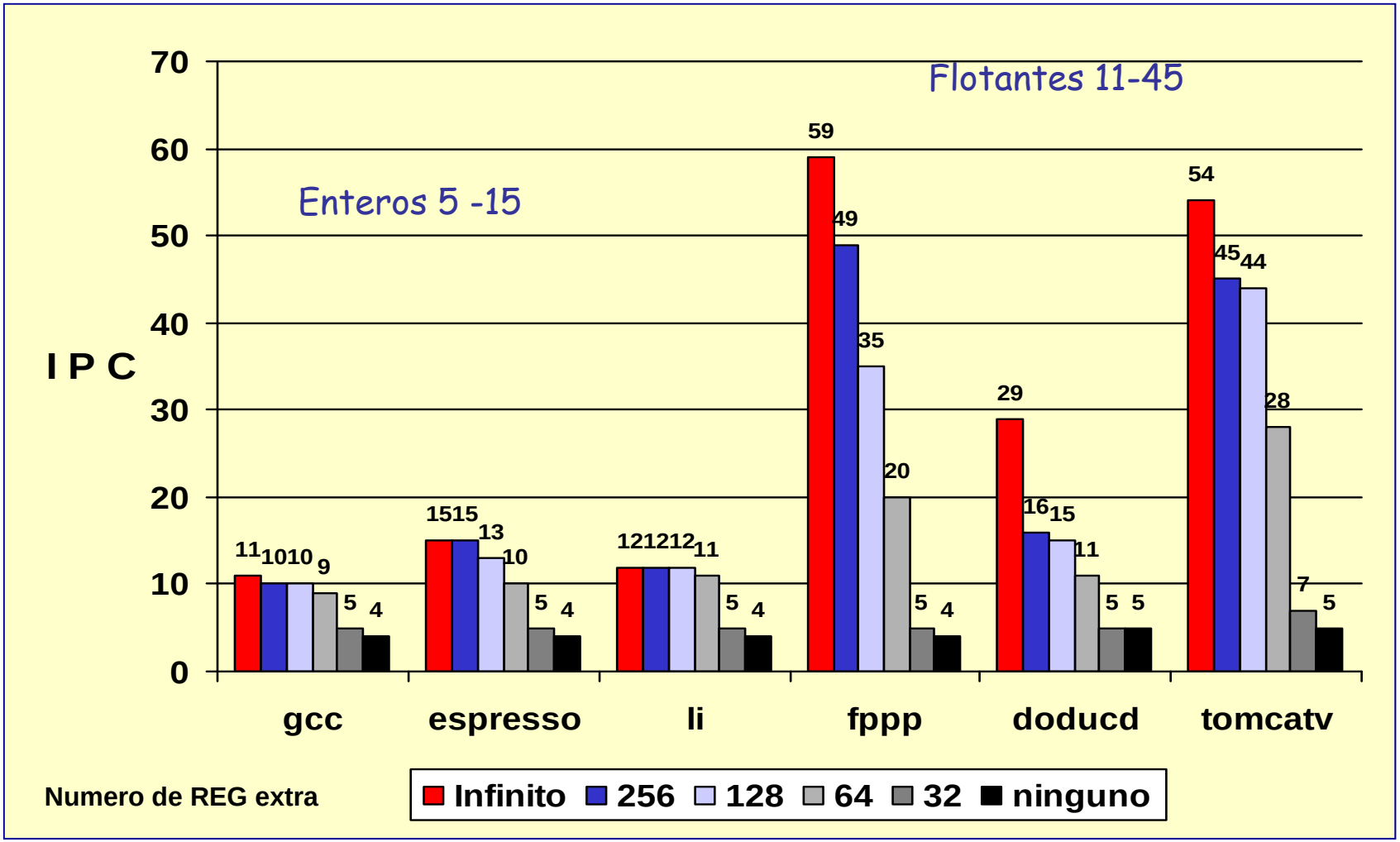
- Modelo versus procesador real

	Nuevo Modelo	Modelo	Power 5
Instrucciones lanzadas por ciclo	64 sin restricciones	Infinitas	4
Ventana de instrucciones	2048	Infinita	200
Registros para renombrado	Infinito v. 256, 128, 64, 32, ninguno	Infinitos	48 enteros + 40 Fl. Pt.
Predicción de saltos	8K Tournament (hibrido)	Perfecta	Tournament
Cache	Perfecta	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	Perfecto	Perfecto	

Límites del ILP

❑ HW más real: Impacto del numero de registros

Ventana de 2048 y lanza 64 por ciclo, predictor híbrido 8K entradas



Límites del ILP

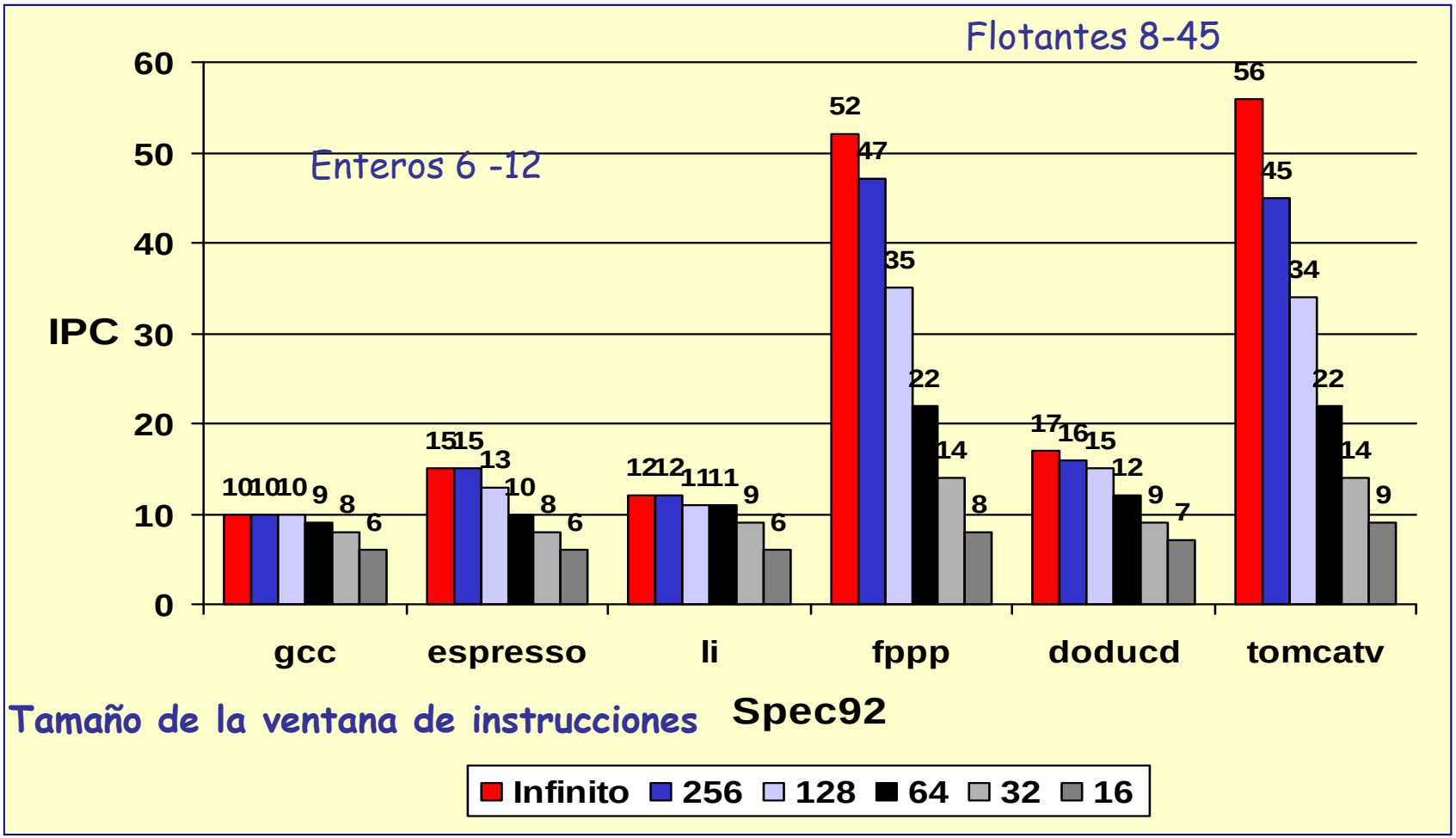
- Modelo versus procesador real

	Nuevo Modelo	Modelo	Power 5
Instrucciones lanzadas por ciclo	64 (sin restricciones)	Infinitas	4
Ventana de instrucciones	Infinito vs. 256, 128, 32, 16	Infinita	200
Registros para renombrado	64 Int + 64 FP	Infinitos	48 enteros + 40 Fl. Pt.
Predicción de saltos	1K 2-bit	Perfecta	Tournament
Cache	Perfecto	Perfecta	64KI, 32KD, 1.92MB L2, 36 MB L3
Análisis de Memory Alias	HW disambiguation	Perfecto	

Límites del ILP

- **HW realizable**

  64 instrucciones por ciclo sin restricciones, predictor híbrido, predictor de retornos de 16 entradas, Desambiguación perfecta, 64 registros enteros y 64 Fp extra



Un ejemplo: Implementaciones X86 P6, Netburst(Pentium4)...

P6

- P6 (implementacion de la arquitectura IA-32 usada en Pentium Pro, II, III)

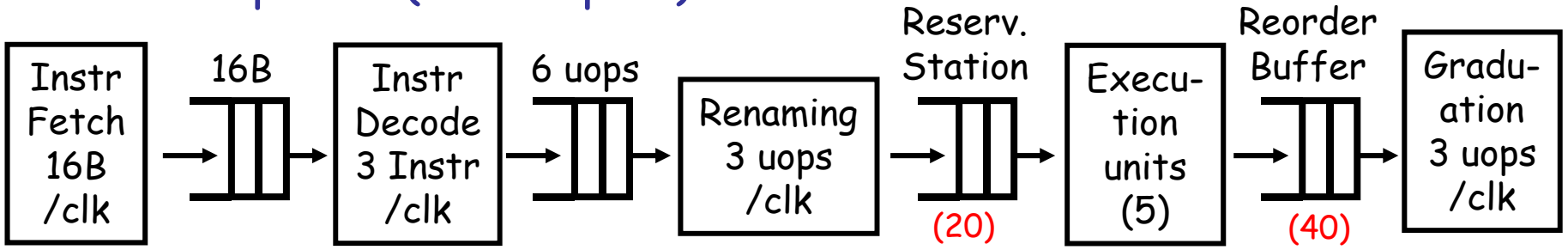
Modelo	Año	Clock	L1	L2
Pentium Pro	1995	100-200Mhz	8+8 KB	126-1024KB
Pentium II	1998	233-450Mhz	16+16 KB	256-512KB
Celeron	1999	500-900Mhz	16+16 KB	128KB
Pentium III	1999	450-1100Mhz	16+16 KB	256-512KB
PentiumIII Xeon	1999	700-900Mhz	16+16 KB	1MB-2MB

¿ Como segmentar un ISA con instrucciones entre 1 y 17 bytes?

- El P6 traduce las instrucciones originales IA-32 a microoperaciones de 72 bits (similar al DLX)
- Cada instrucción original se traduce a una secuencia de1 a 4 microoperaciones. Pero ...
  - La instrucciones más complejas son traducidas por una secuencia almacenada en una memoria ROM(8Kx72)(microprograma)
- Tiene un pipeline de 14 etapas
- Ejecución fuera de orden especulativa, con renombrado de registros y ROB

P6

❑ P6 Pipeline (14 etapas)



8 etapas para fetch, decodificación y issue en orden

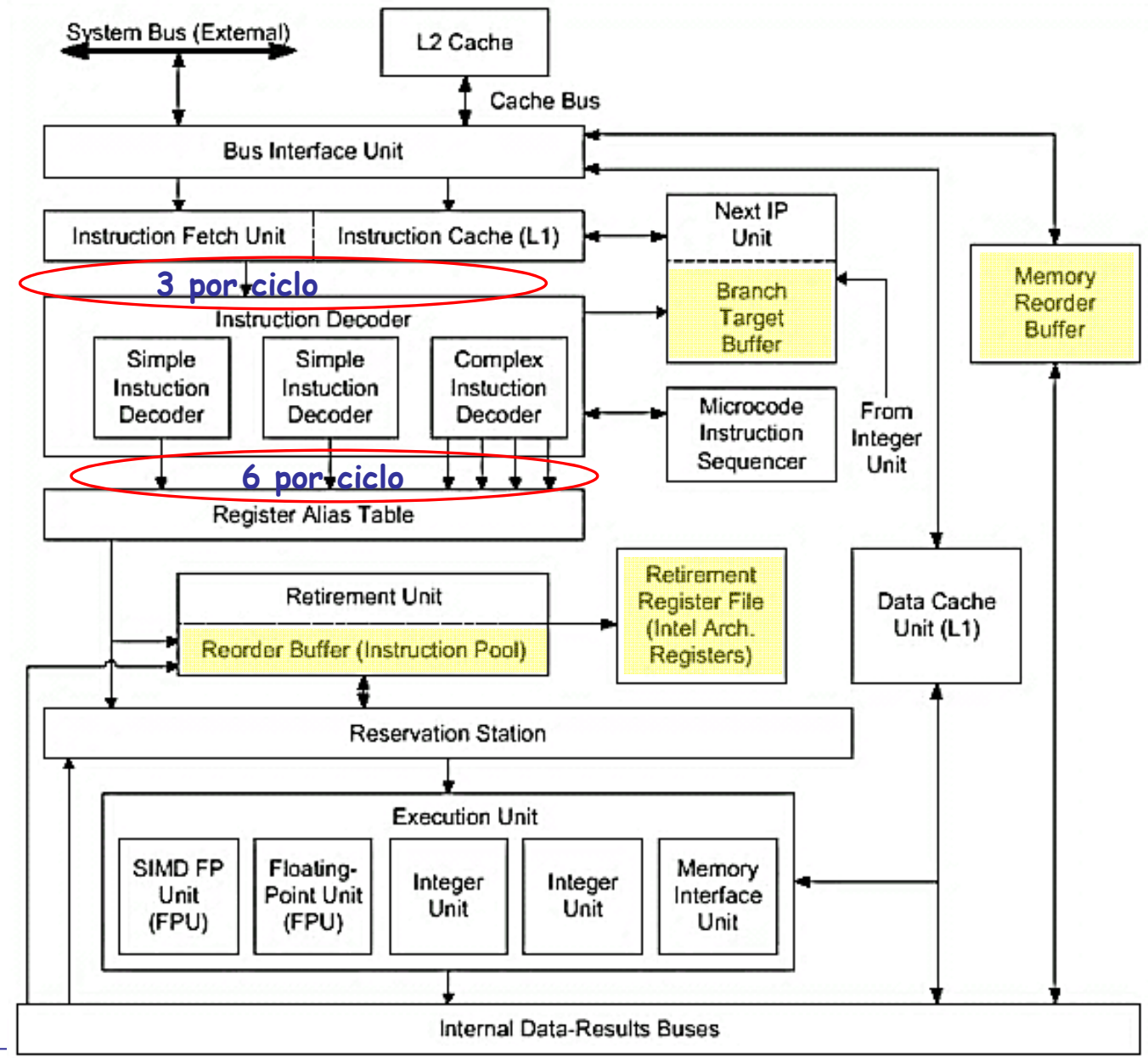
- 1 ciclo para determinar la longitud de la instrucción 80x86 in + 2 más para generar las microoperaciones

❑ 3 etapas para ejecución fuera de orden en una de 5 unidades funcionales

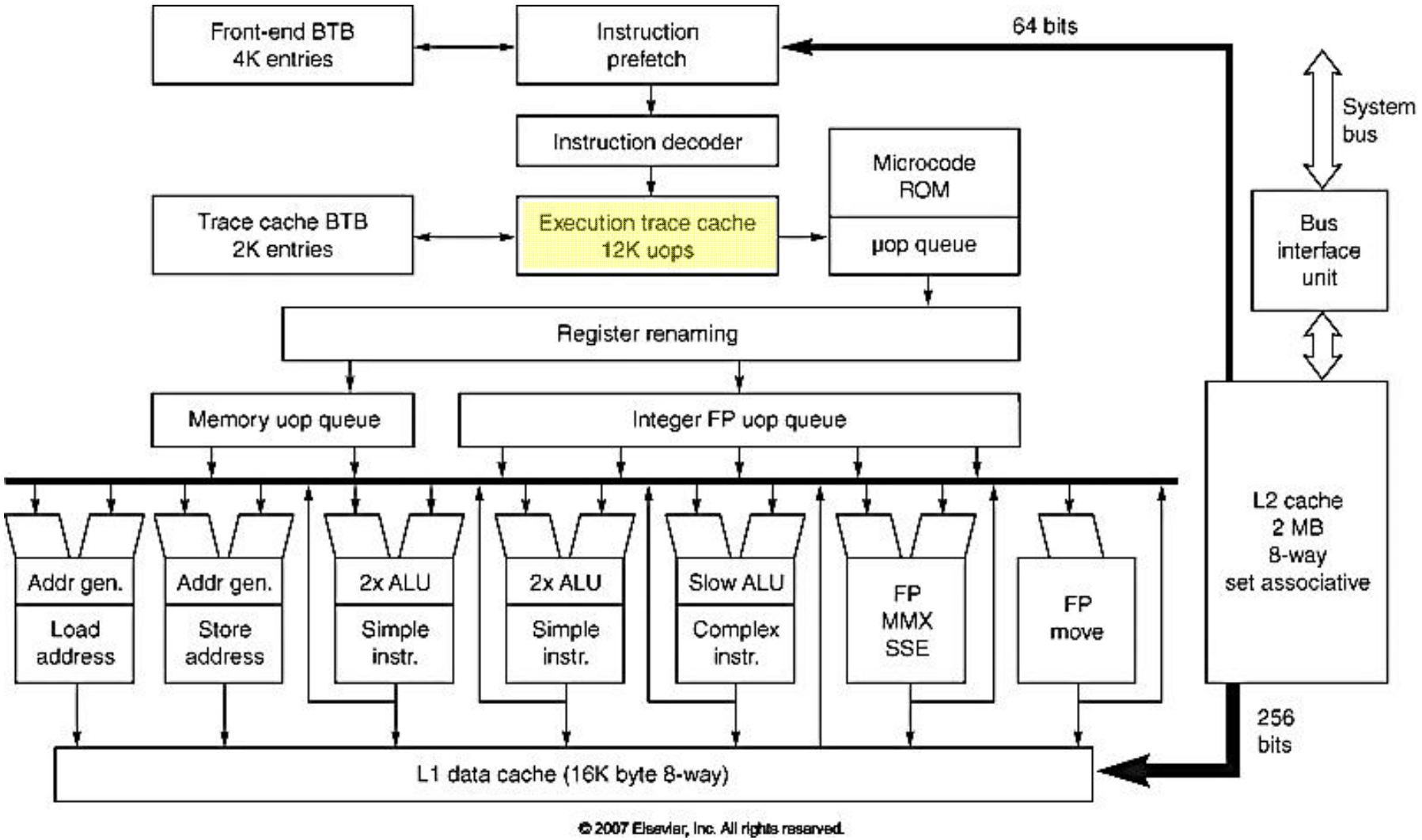
❑ 3 etapas para la finalización de la instrucción (commit)

Parameter	80x86	microops
Max. instructions issued/clock	3	6
Max. instr. complete exec./clock		5
Max. instr. commited/clock		3
Window (Instrs in reorder buffer)	40	
Number of reservations stations	20	
Number of rename registers	40	
No. integer functional units (FUs)	2	
No. floating point FUs	1	
No. SIMD Fl. Pt. FUs	1	
No. memory Fus	1 load + 1 store	

P6


System Bus (External)
L2 Cache
Cache Bus
Bus Interface Unit
Instruction Fetch Unit
Instruction Cache (L1)
Next IP Unit
Branch Target Buffer
3 por ciclo
Instruction Decoder
Simple Instuction Decoder
Simple Instuction Decoder
Complex Instuction Decoder
Microcode Instruction Sequencer
From Integer Unit
Memory Reorder Buffer
6 por ciclo
Register Alias Table
Retirement Unit
Reorder Buffer (Instruction Pool)
Retirement Register File (Intel Arch. Registers)
Data Cache Unit (L1)
Reservation Station
Execution Unit
SIMD FP Unit (FPU)
Floating-Point Unit (FPU)
Integer Unit
Integer Unit
Memory Interface Unit
Internal Data-Results Buses

Pentium 4 Microarchitecture



- BTB = Branch Target Buffer (branch predictor)
- I-TLB = Instruction TLB, Trace Cache = Instruction cache
- RF = Register File; AGU = Address Generation Unit
- "Double pumped ALU" means ALU clock rate 2X => 2X ALU F.U.s

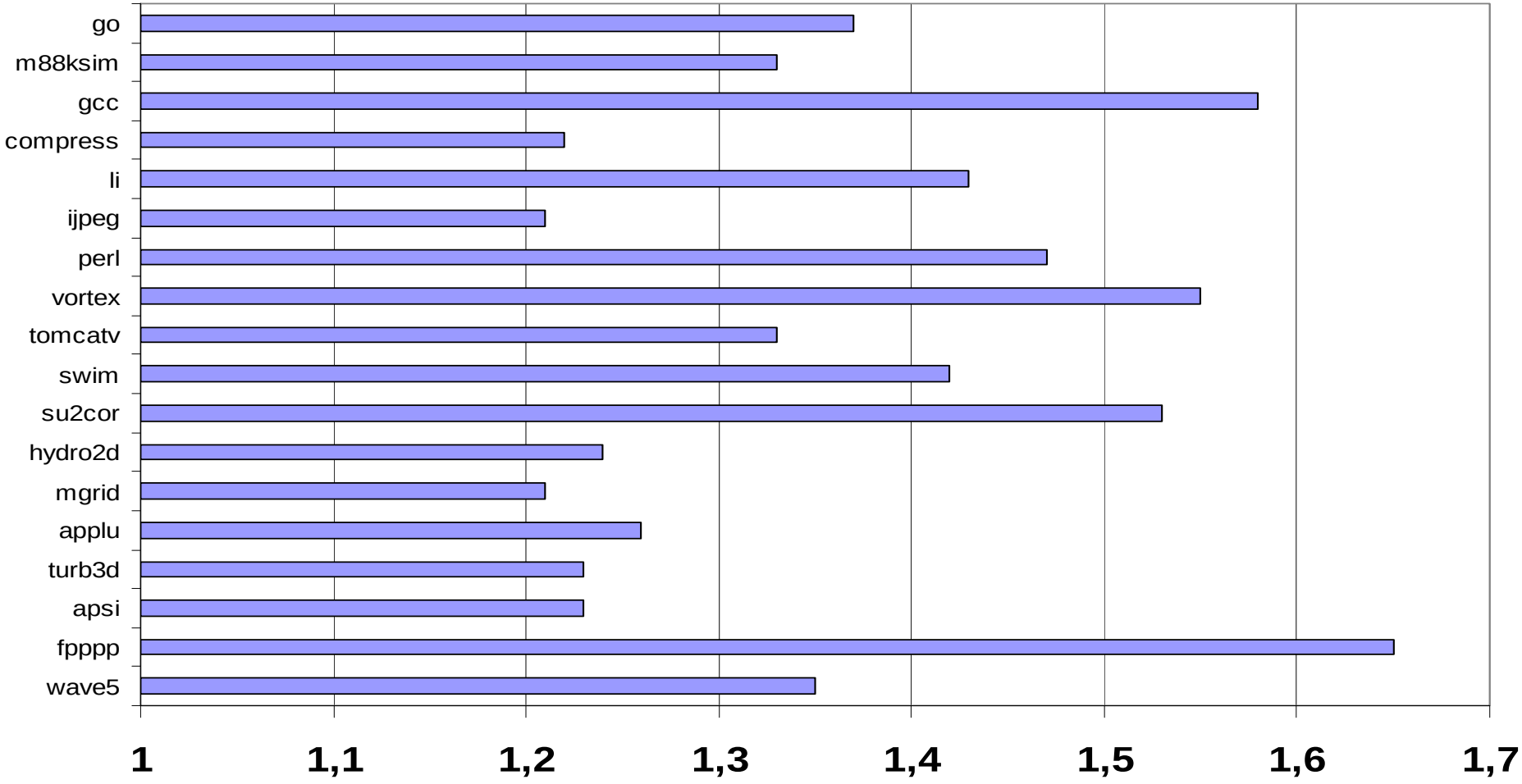
Pentium 4

Intel Netburst Microarchitecture

- Traduce instrucciones 80x86 a micro-ops (como P6)
- P4 tiene mejor predicción de saltos (x8) y más FU (7 versus 5)
- La Cache de instrucciones almacena micro-operaciones vs. 80x86 instrucciones. "trace cache" (TC), BTB TC 2K entradas
  - En caso de acierto elimina decodificación
- Nuevo bus de memoria: 400(800) MHz vs. 133 MHz (RamBus, DDR, SDRAM) (Bus@1066 Mhz)
- Caches
  - Pentium III: L1I 16KB, L1D 16KB, L2 256 KB
  - Pentium 4: L1I 12K uops, L1D 16 KB 8-way, L2 2MB 8-way
- Clock :
  - Pentium III 1 GHz v. Pentium 4 1.5 GHz (3.8 Ghz)
  - 14 etapas en pipeline vs. 24 etapas en pipeline (31 etapas)
- Instrucciones Multimedia: 128 bits vs. 64 bits => 144 instrucciones nuevas.
- Usa RAMBUS DRAM
  - Más AB y misma latencia que SDRAM. Costo 2X-3X vs. SDRAM
- ALUs operan al doble del clock para operaciones simples
- Registros de renombrado: 40 vs. 128; Ventana: 40 v. 126
- BTB: 512 vs. 4096 entradas. Mejora 30% la tasa de malas predicciones

Netburst(Pentium 4) Rendimiento

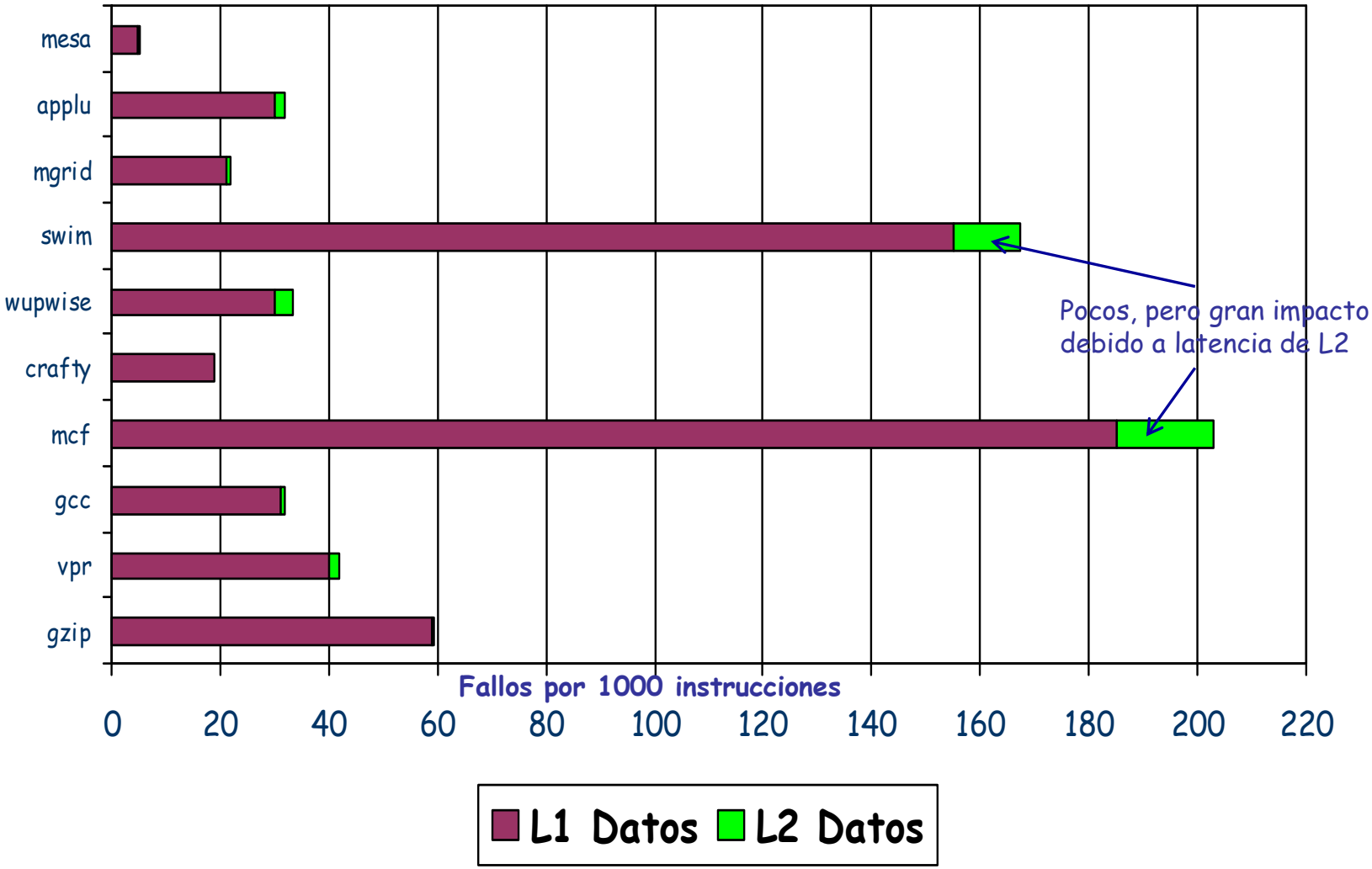
Relación u-operaciones s/x86 instrucciones



1.2 to 1.6 uops per IA-32 instruction: 1.36 avg. (**1.37 integer**)

Netburst(Pentium4) Rendimiento

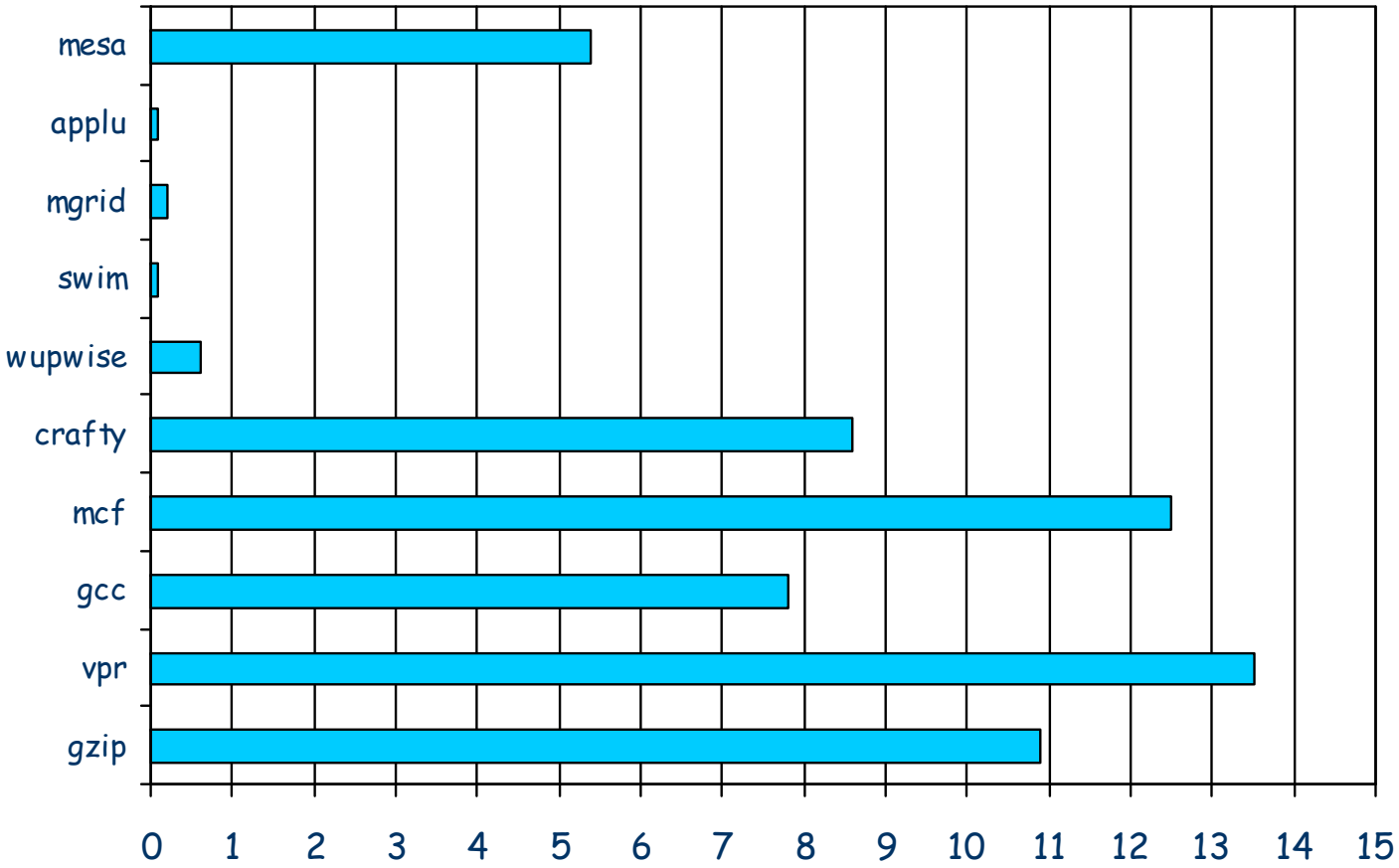
Fallos de cache de datos



De 10 a 200 fallos por cada mil instrucciones

Netburst(Pentium4) Rendimiento

Comportamiento del predictor: Dos niveles, con información global y local 4K entradas

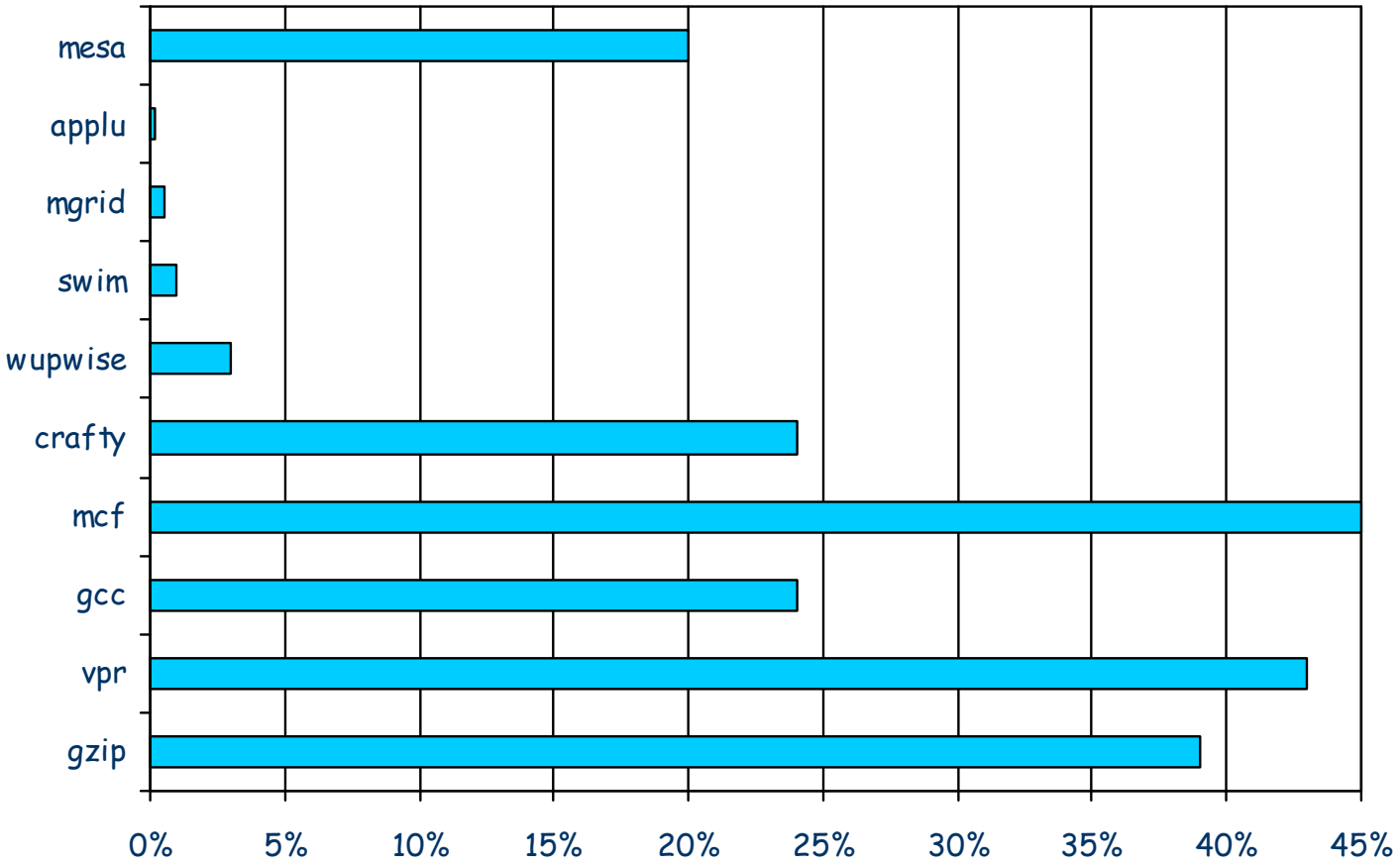


Fallos de predicción por cada mil instrucciones, 2% en FP, 6% en INT

Netburst(Pentium4) Rendimiento

Especulación: porcentaje de μ-operaciones que no finalizan (commit)

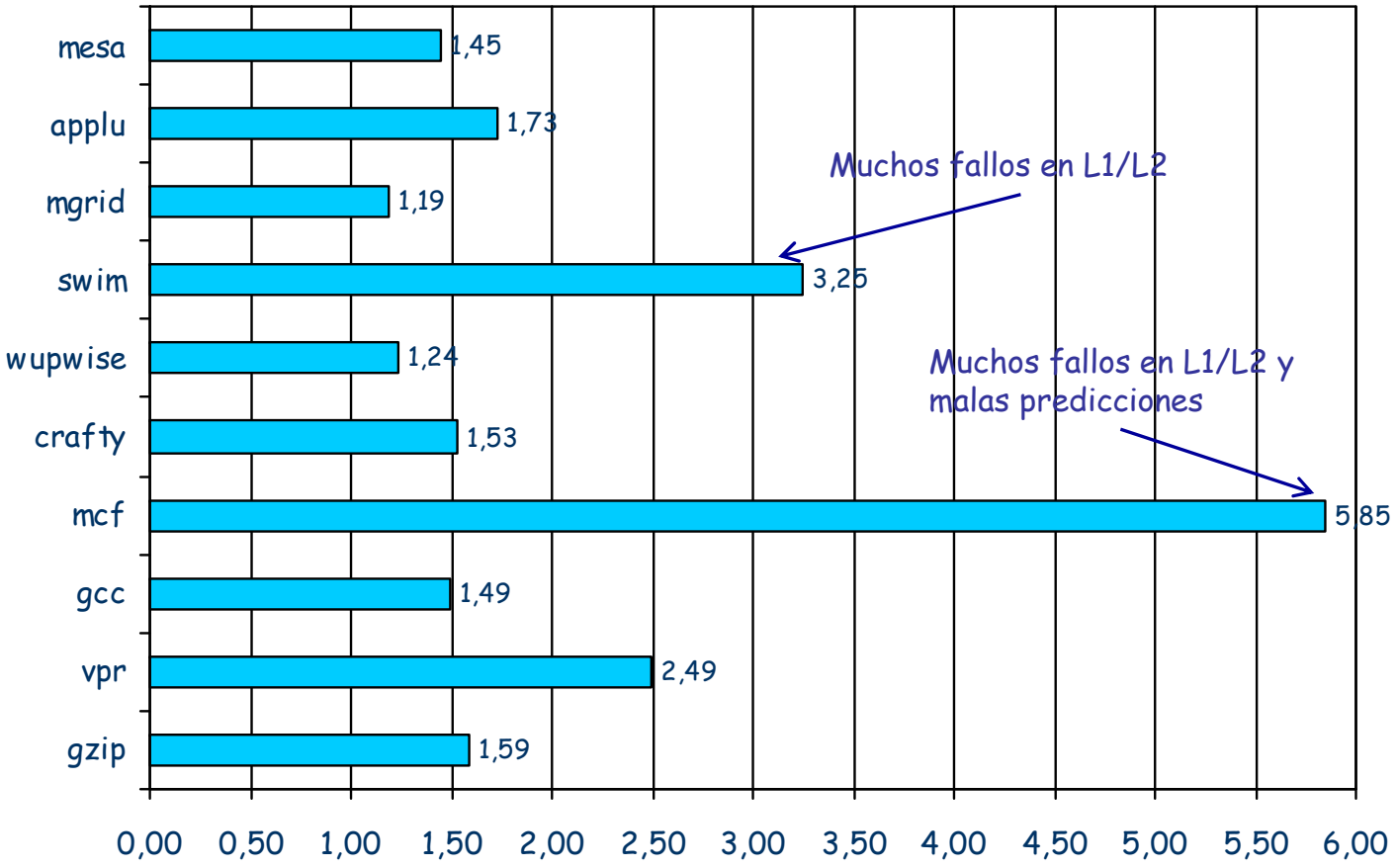
Muy correlacionada con gráfica anterior



Del 1% al 45 % de las instrucciones no finalizan

Netburst(Pentium4) Rendimiento

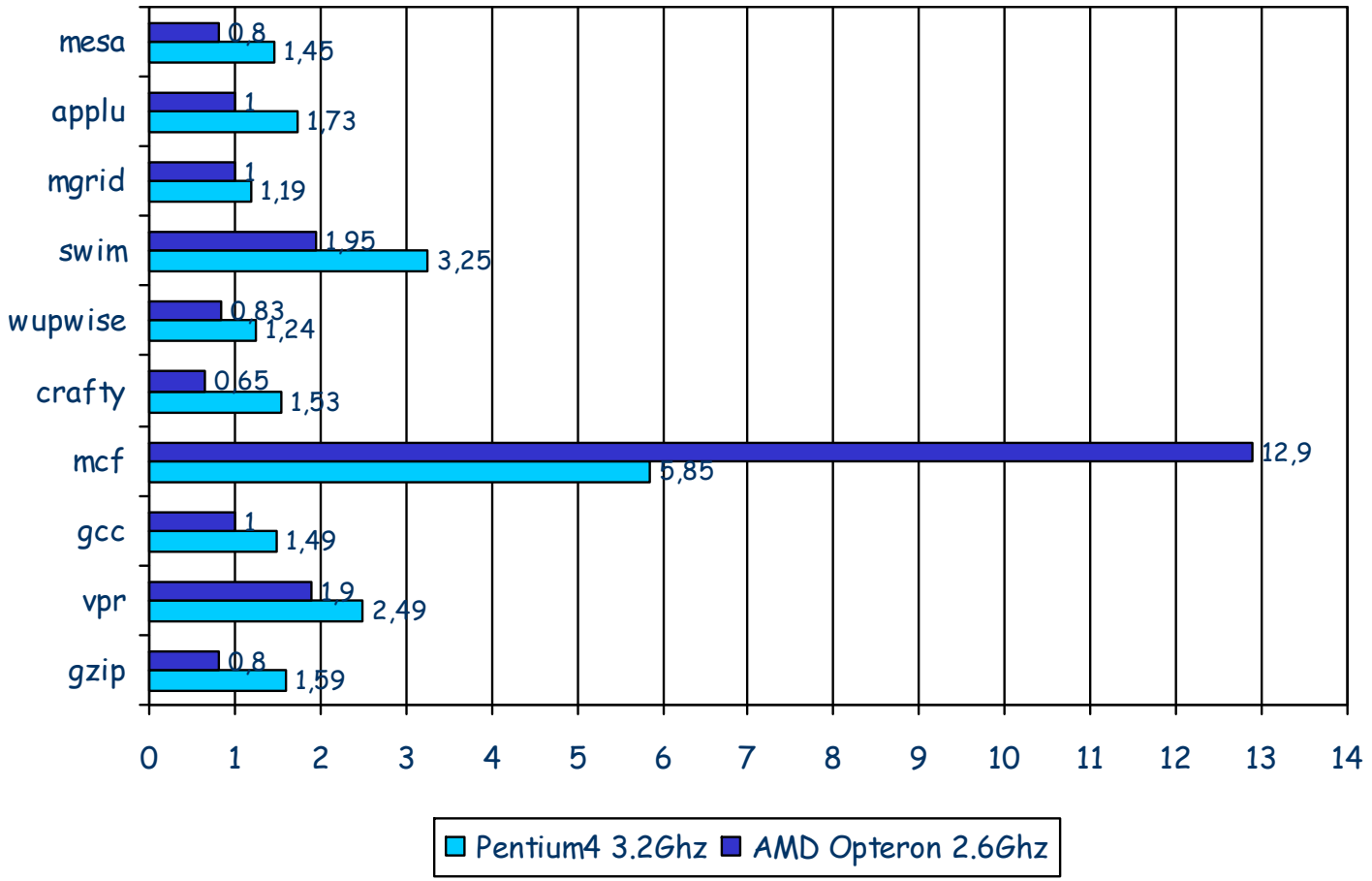
CPI real obtenido



Netburst(Pentium4) versus AMD

Diferencia fundamental; Pentium pipe profundo para soportar alta frecuencia

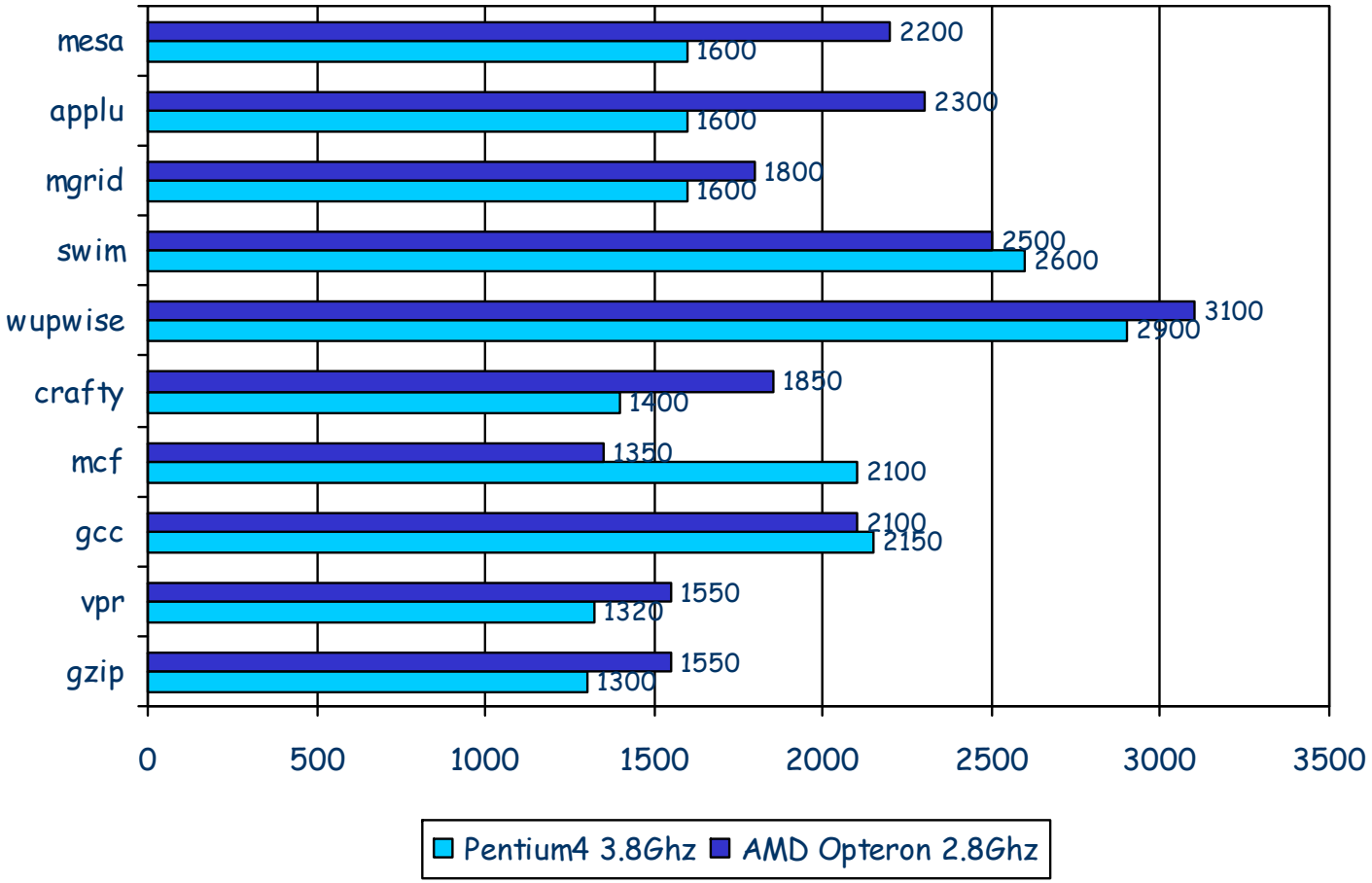
CPI real obtenido



AMD CPI menor en un factor de 1,27. ¿Se puede compensar con la mayor frecuencia?

Netburst(Pentium4) versus AMD

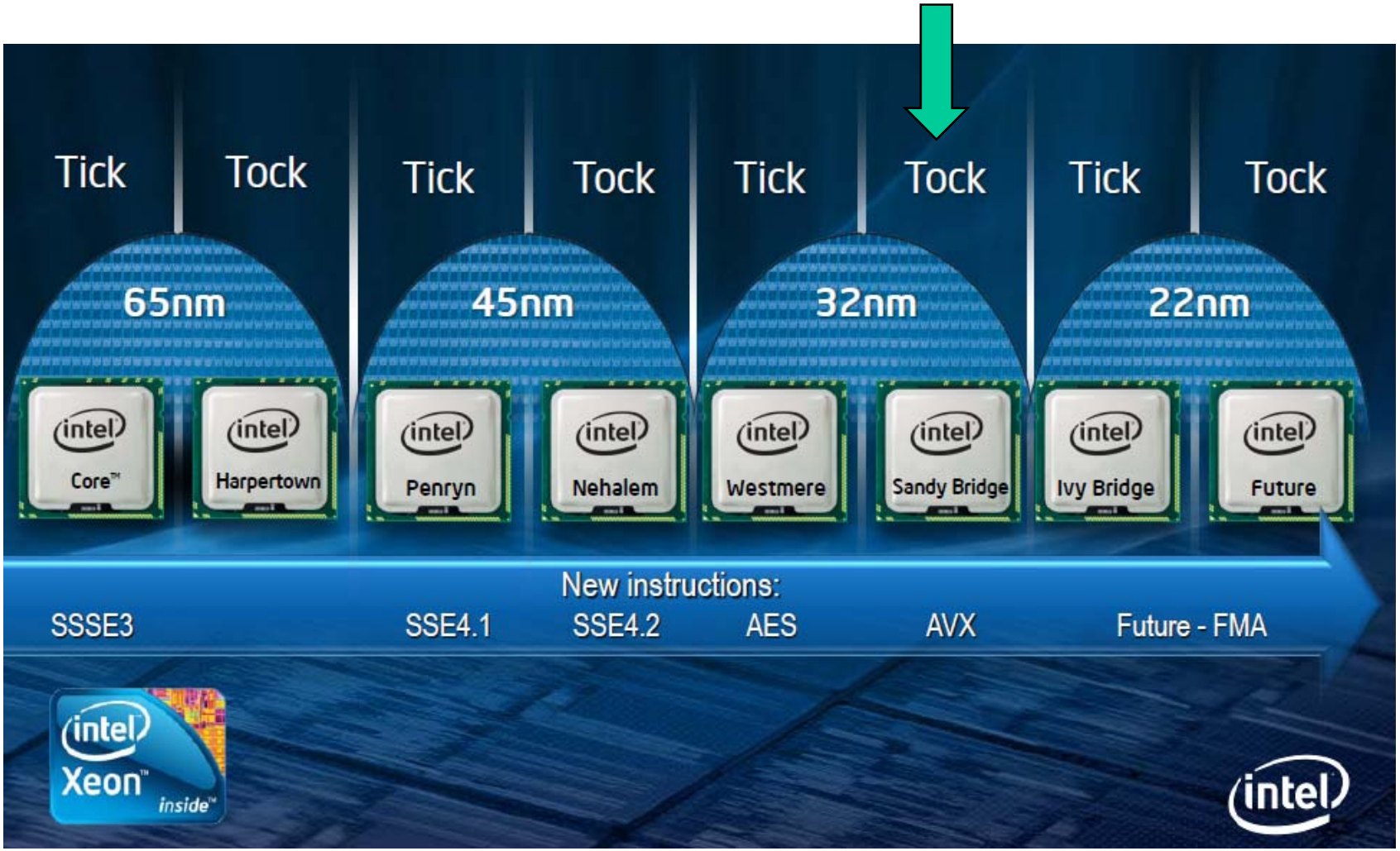
Rendimiento en SPEC2000



AMD 1,08 más rendimiento. La mayor frecuencia no compensa el mayor CPI

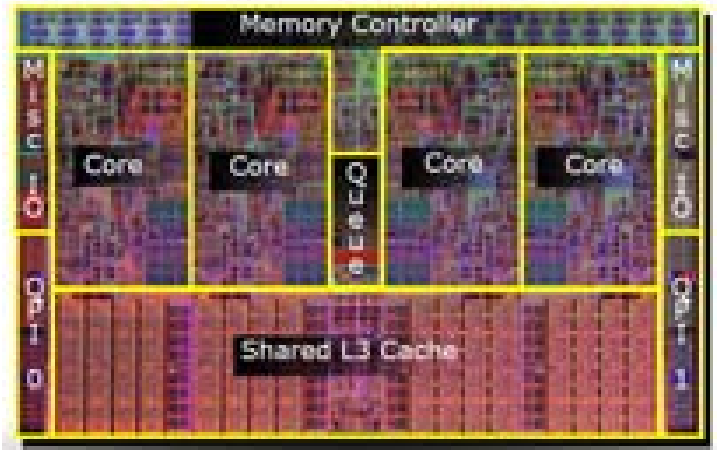
Intel Dual Core, Core2, Quad, Core i…

Tick-Tock



Tock 2008/09 (Nehalem)

Core i7 nov 2008
Nuevo core, 2 Threads SMT, 2 a 8 cores, 740MT, 3.2Ghz, 140w
L2 256KB propietaria, L3 8MB compartida todos los cores (inclusiva)
128 Mi "on fly" (96 Penryn)
2 Level TLB y 2 level BTB, Mejoras en prefech y Load/stores
QuickPath 25GB/s, "on chip" controlador de memoria
40% más rendimiento

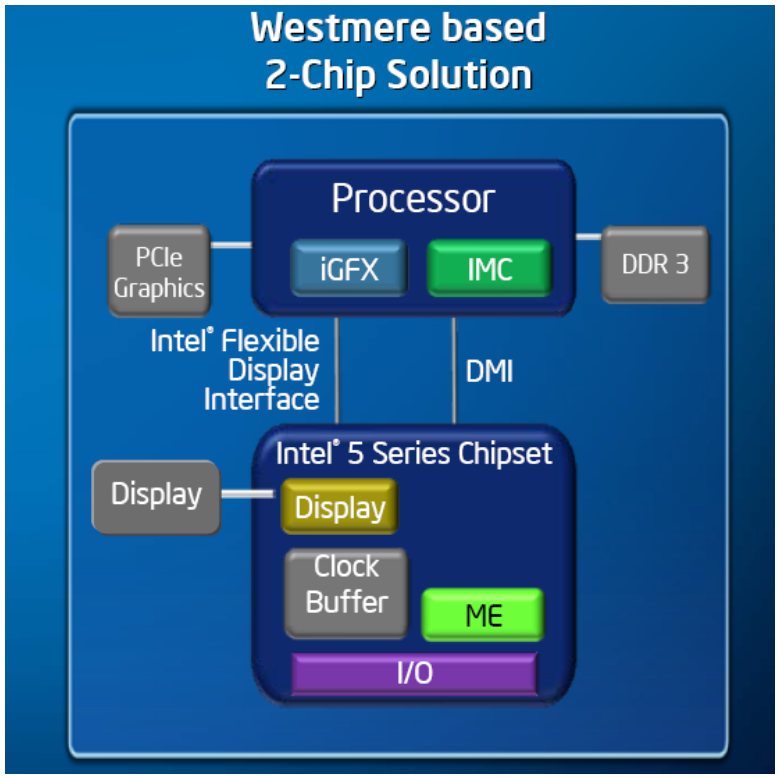


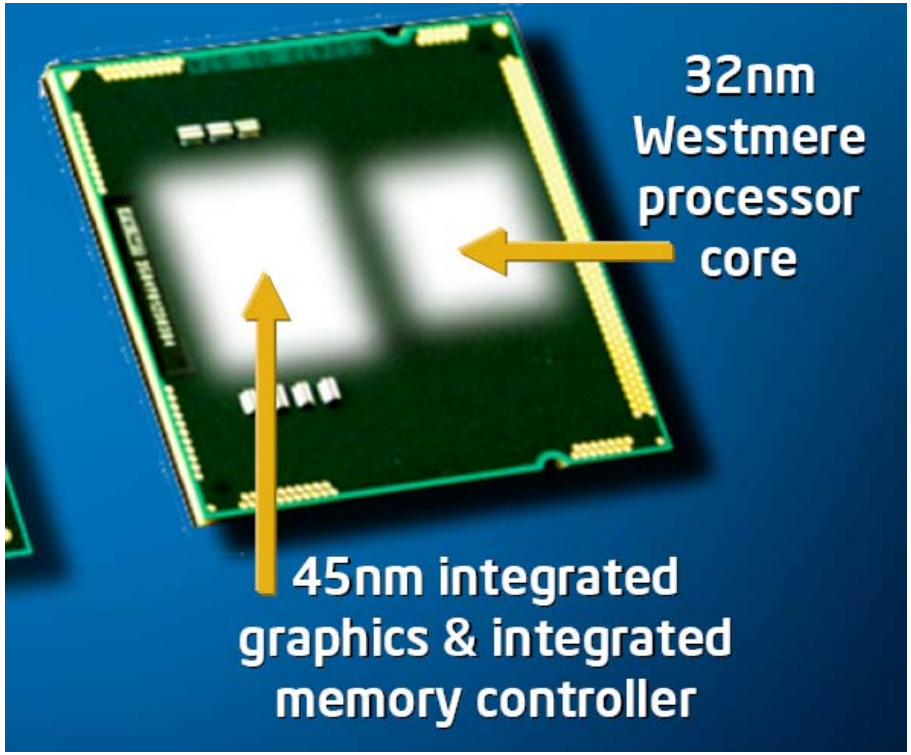
Más **X86-Everywhere**

Atom x86, 24mm^2,160mw, 2threads, 2cores
SoC x86, network, comunicaciones,…

Tick 2009/10 (Westmere)

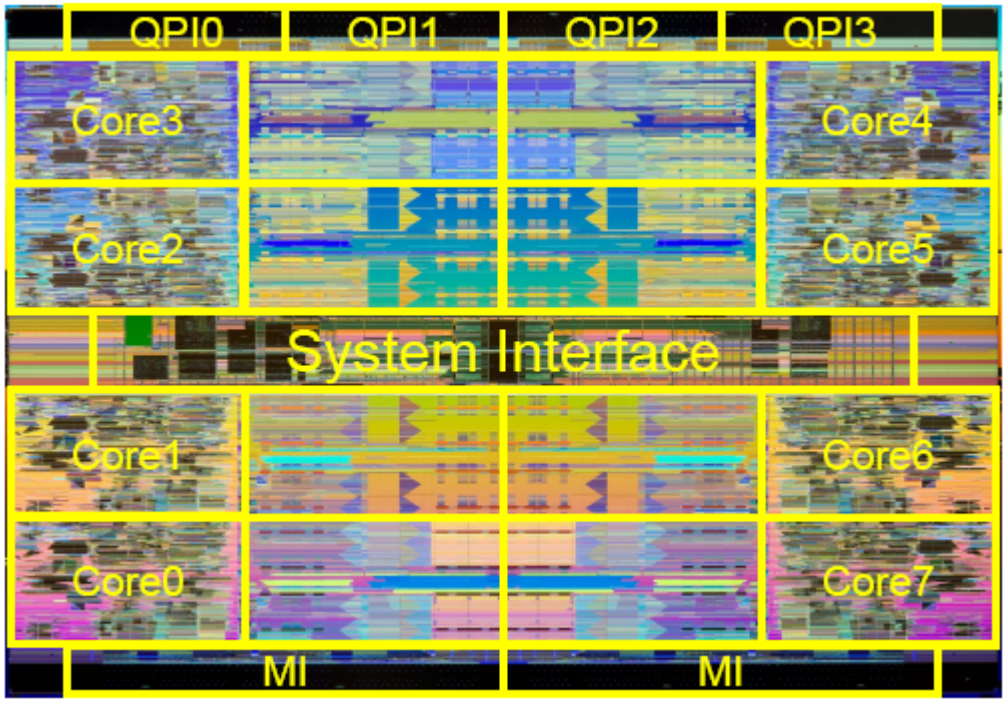
Westmere core i5 e i3
Nuevos desktop y mobile chips (clarkdale y arrandale)
2 cores / 4theads, 4-3 MB L3, 7 nuevas instrucciones AES



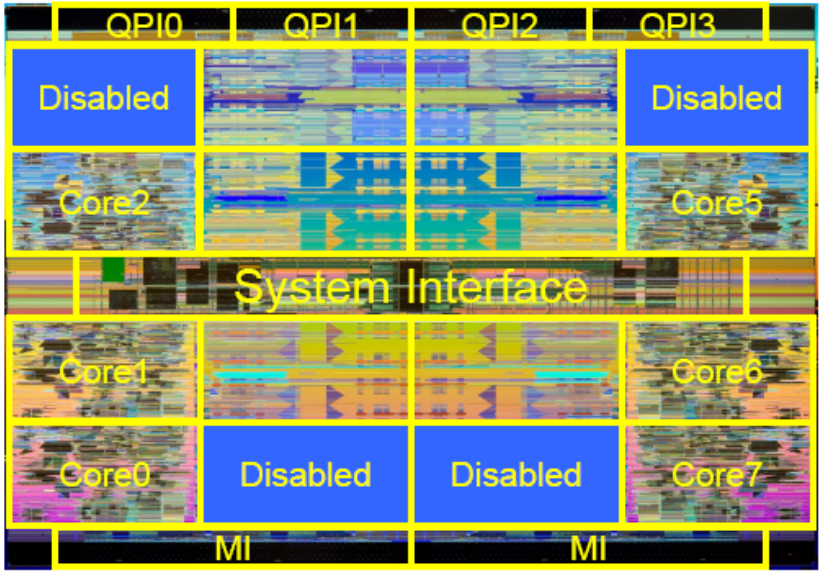


Intel Roadmap…

Nehalem EX 8 cores /16 Theads, 45 nm, 2300 M Trans

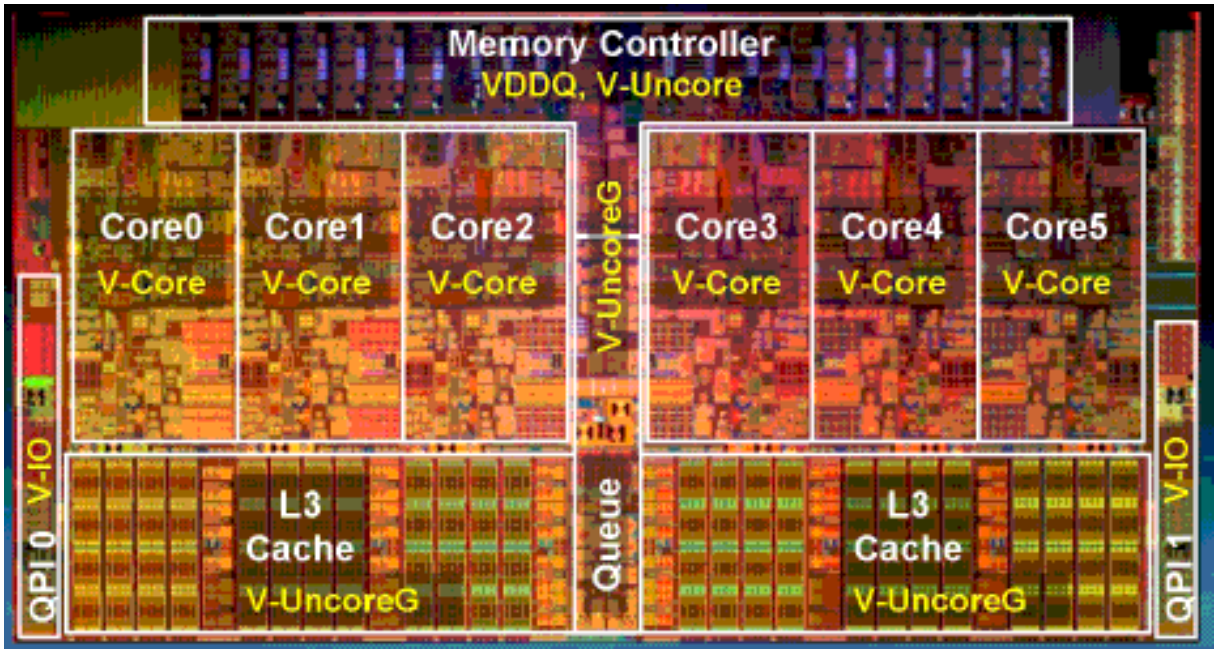


24 MB L3, 130w
4QPI links (25.6GB/s), 2 MC
Nodo 8 socket , 128 threads



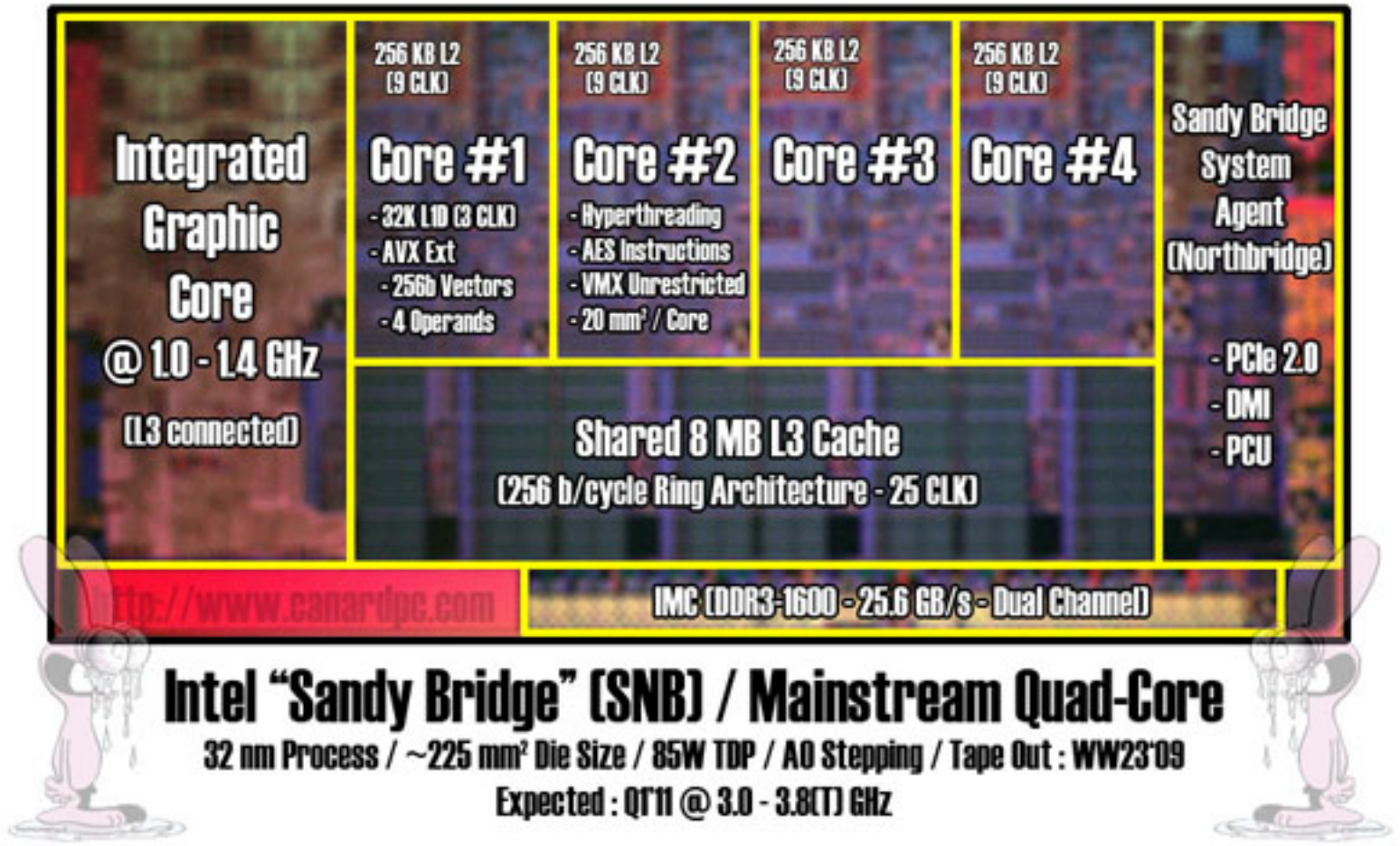
Dos cores y 2 slice de cache deshabilitados.

Westmere EP 2010

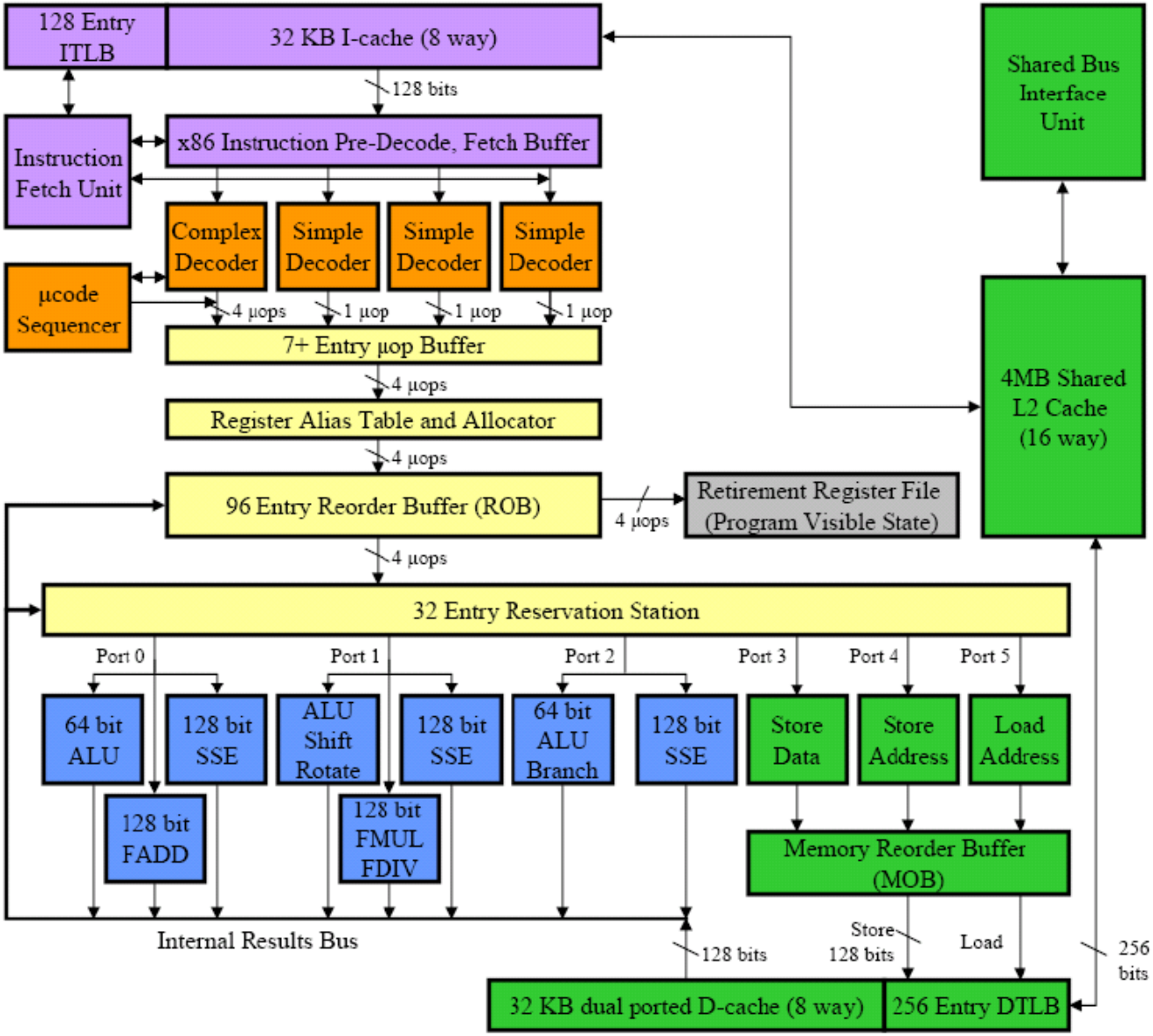


6 cores/ 12 threads , 32nm, 233mm2, 130w, 12Mb L3
Gestión mas eficiente del power (core, uncore)
Westmere EX 2011, 10 cores/20 threads

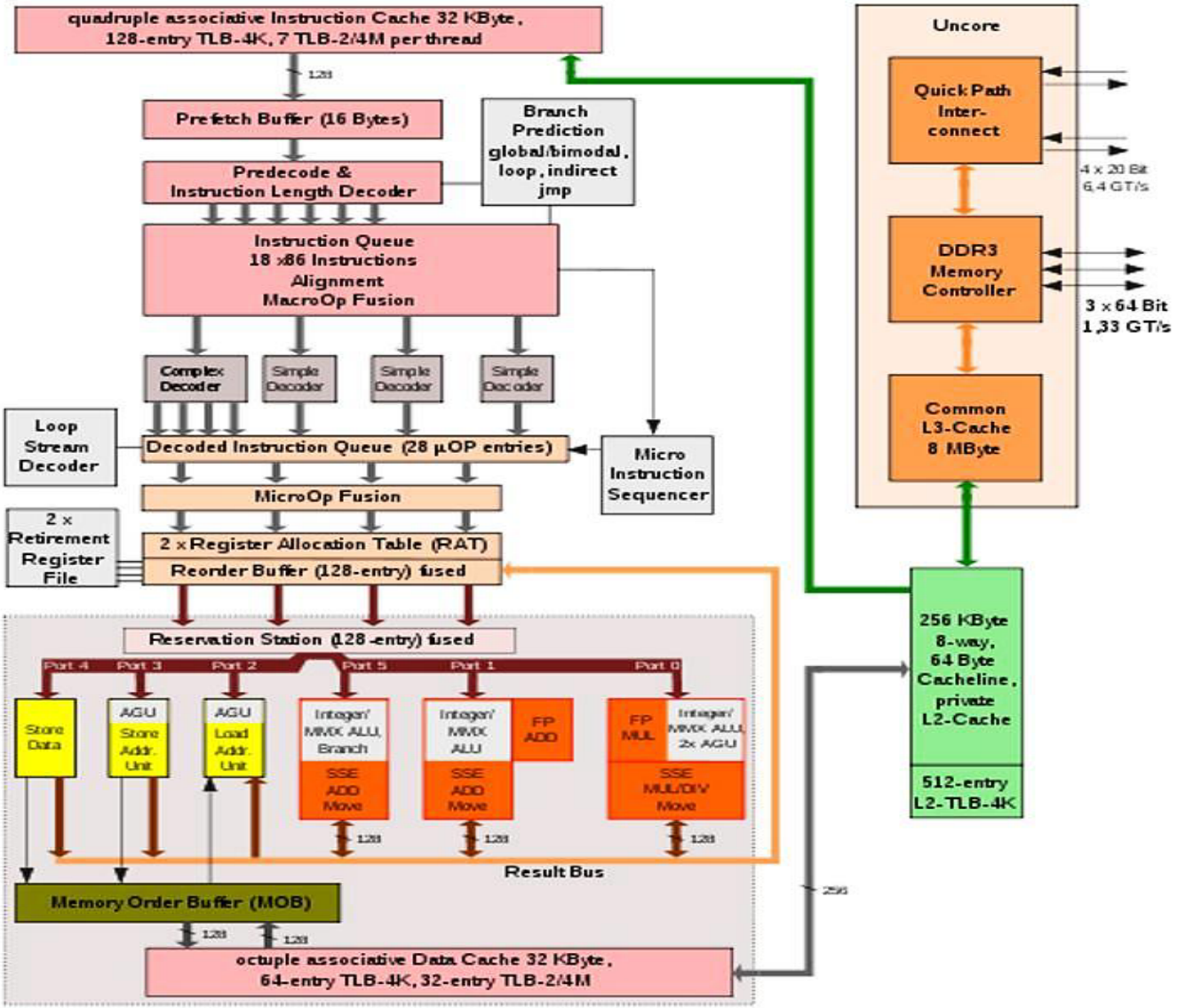
Tock 2010/2011 (Sandy Bridge)



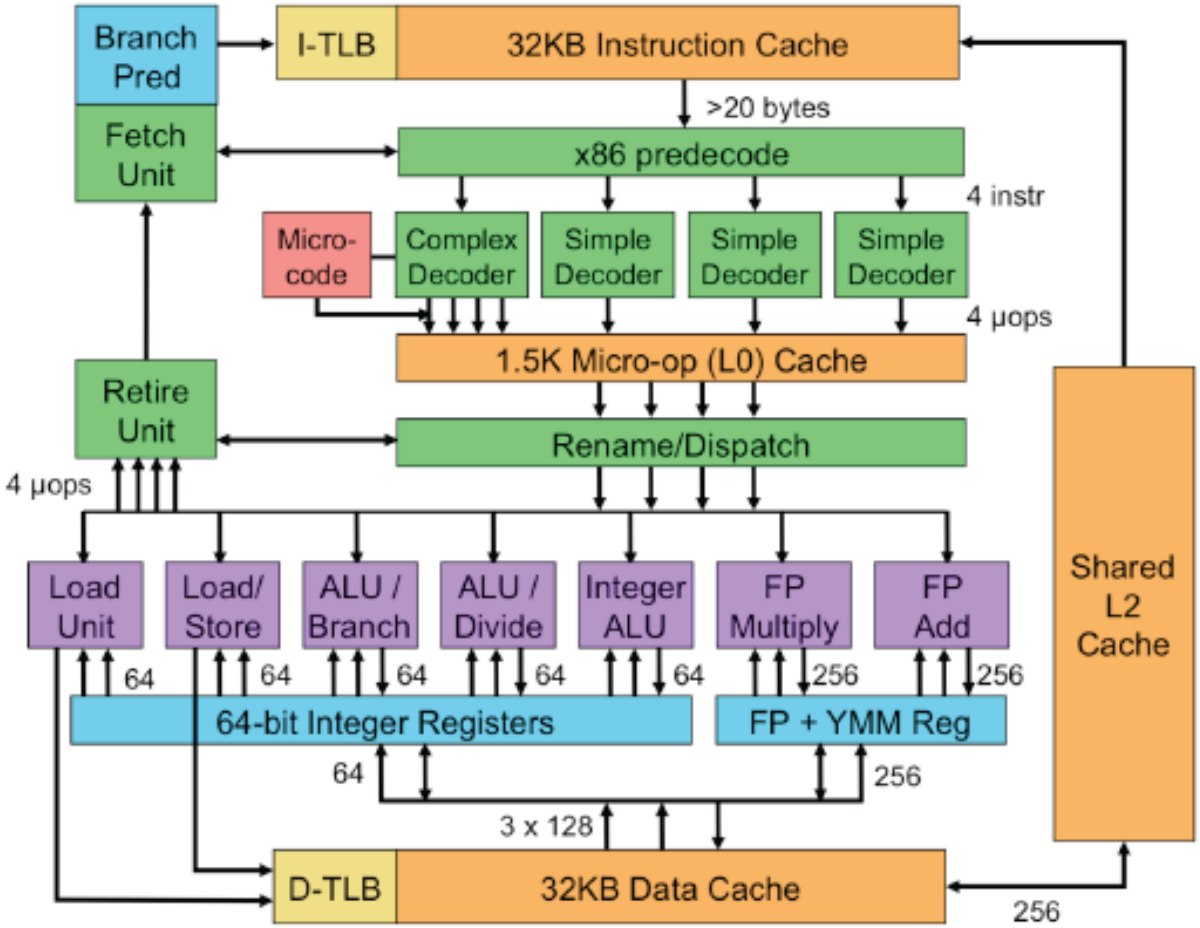
Core Microarchitecture

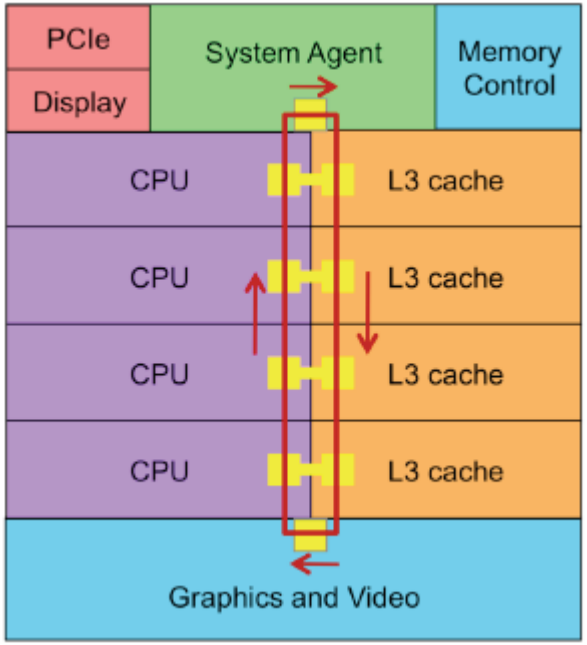


Nahalem Microarchitecture



Sandy Bridge Microarchitecture





ROB 168 (128)
Cache L0 , 5K de instrucciones
2 Load y 1 Store por ciclo

Límites del ILP

- **¿Como superar los limites de este estudio?**
  - Mejoras en los compiladores e ISAs
  - Eliminar riesgos EDL y EDE en la memoria
  - Eliminar riesgos LDE en registros y memoria. Predicción
  - Buscar otros paralelismos (thread)

Límites del ILP

❑ Buscar paralelismo de más de un thread

- Hay mucho paralelismo en algunas aplicaciones (Bases de datos, códigos científicos)
- Thread Level Parallelism
  - Thread: proceso con sus propias instrucciones y datos
    - Cada thread puede ser parte de un programa paralelo de múltiples procesos, o un programa independiente.
    - Cada thread tiene todo el estado (instrucciones, datos, PC, register state, …) necesario para permitir su ejecución
    - Arquitecturas(multiprocesadores, MultiThreading y multi/many cores)
- Data Level Parallelism: Operaciones idénticas sobre grandes volúmenes de datos (extensiones multimedia y arquitecturas vectoriales

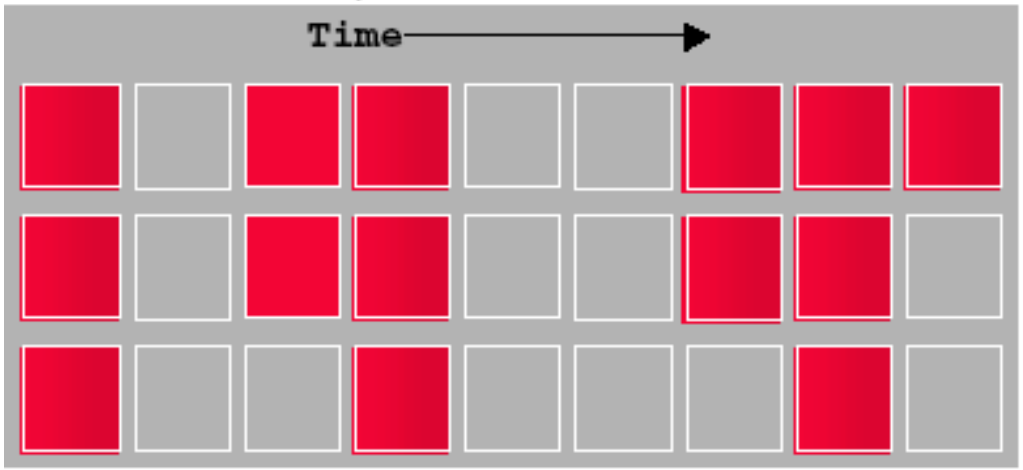
Thread Level Parallelism (TLP) versus ILP

- ILP explota paralelismo implícito dentro de un segmento de código lineal o un bucle
- TLP representa el uso de múltiples thread que son inherentemente paralelos.
- Objetivo: Usar múltiples streams de instrucciones para mejorar:
  - Throughput de computadores que ejecutan muchos programas diferentes.
  - Reducir el tiempo de ejecución de un programa multi-threaded
- TLP puede ser más eficaz en coste que ILP

Multithreading

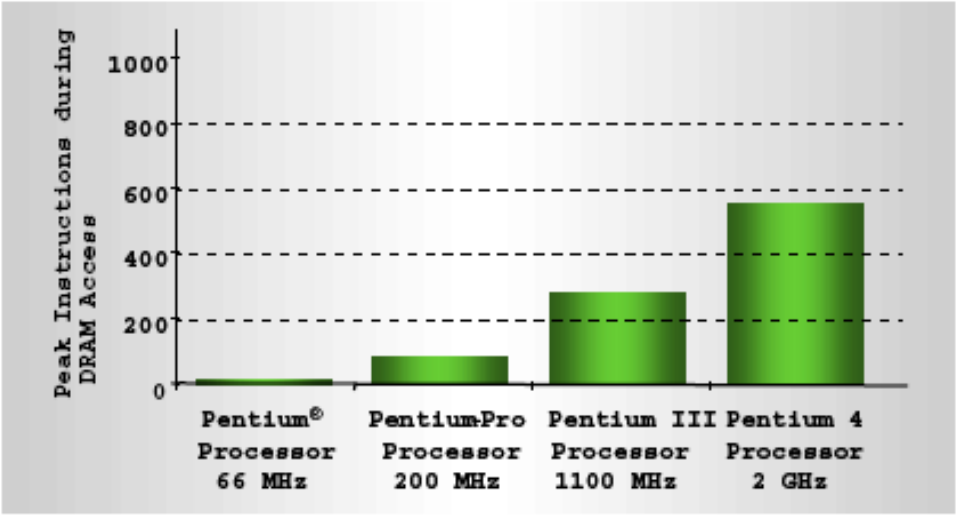
¿ Por que multithreading ?

- Procesador superescalar

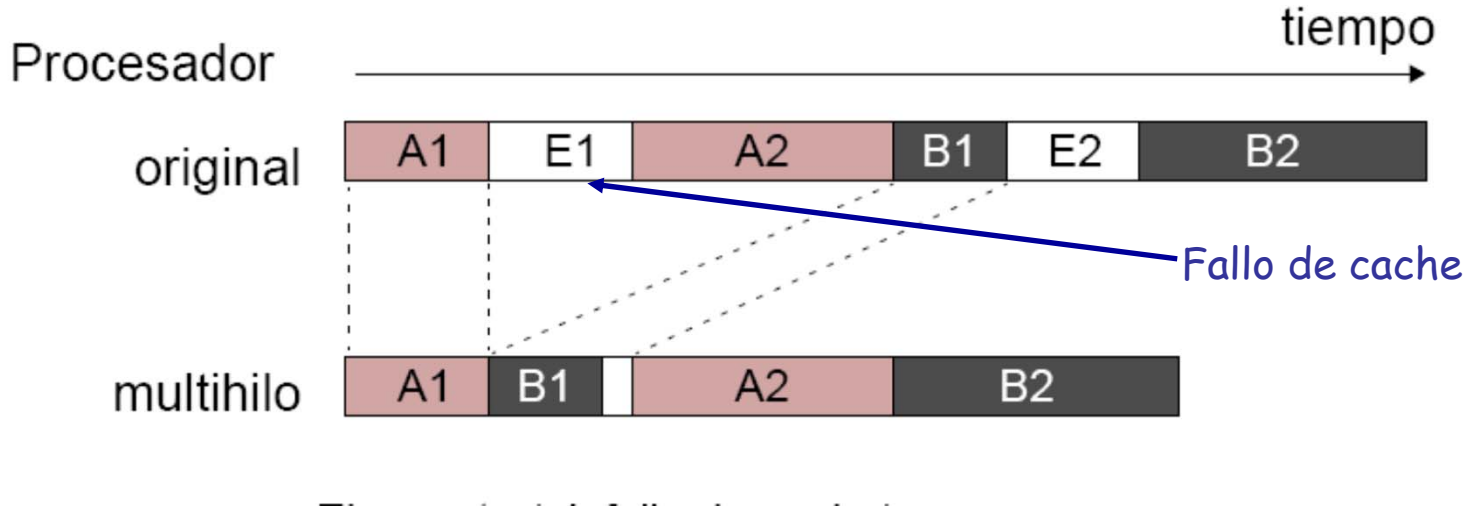


- La latencia de memoria crece.

¿ Como soportarla?



❑ Multithreading



- Incrementar el trabajo procesado por unidad de tiempo
- Si los hilos son del mismo trabajo se reduce el tiempo de ejecución

La técnica multithreading no es ideal y se producen pérdidas de rendimiento.
Por ejemplo, un programa puede ver incrementado su tiempo de ejecución aunque el número de programas ejecutados por unidad de tiempo sea mayor cuando se utiliza multithreading.

❑ Ejecución Mulithreaded

- Multithreading: múltiples threads comparten los recursos del procesador
  - El procesador debe mantener el estado de cada thread e.g., una copia de bloque de registros, un PC separado, tablas de páginas separadas.
  - La memoria compartida ya soporta múltiples procesos.
  - HW para conmutación de thread muy rápido. Mucho mas rápido que entre procesos.
- ¿Cuándo conmutar?
  - Cada ciclo conmutar de thread (grano fino)
  - Cuando un thread debe parar (por ejemplo fallo de cache)
- HEP (1978), Alewife , M-Machine , Tera-Computer

Multithreading de Grano Fino

- Conmuta entre threads en cada instrucción, entrelazando la ejecución de los diferentes thread.
- Generalmente en modo "round-robin", los threads bloqueados se saltan
- La CPU debe ser capaz de conmutar de thread cada ciclo.
- Ventaja; puede ocultar stalls de alta y baja latencia, cuando un thread esta bloqueado los otros usan los recursos.
- Desventaja; retarda la ejecución de cada thread individual, ya que un thread sin stall es retrasado por reparto de recursos (ciclos) entre threads
- Ejemplo Niagara y Niagara 2 (SUN)

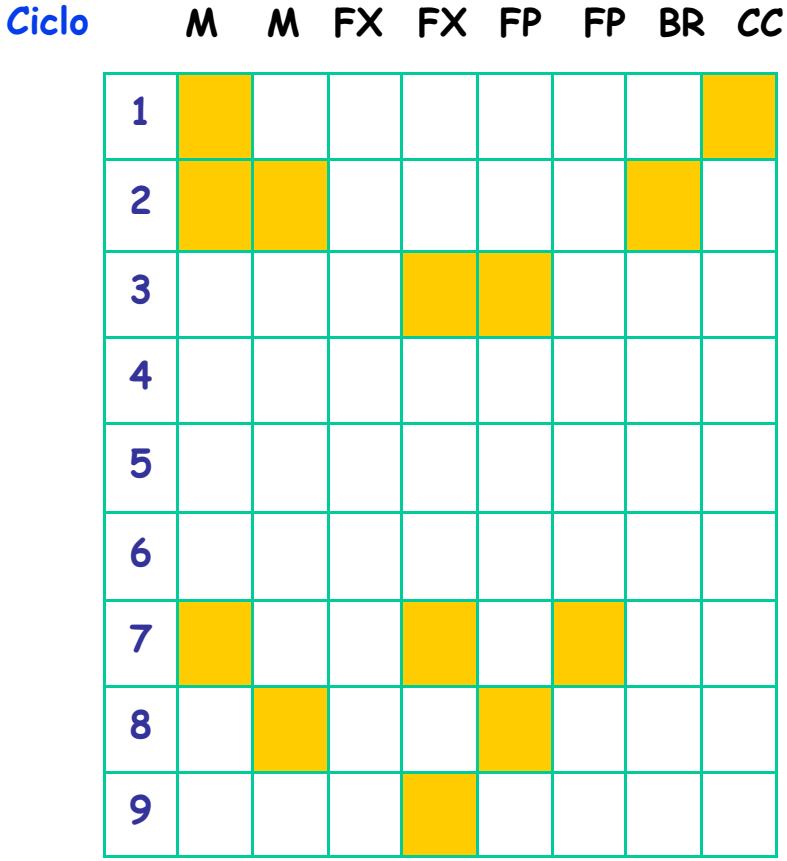
Multithreading Grano Grueso

- Conmuta entre threads solo en caso de largos stalls, como fallos de cache L2
- Ventajas
  - No necesita conmutación entre thread muy rápida.
  - No retarda cada thread, la conmutación solo se produce cuando un thread no puede avanzar.
- Desventajas; no elimina perdidas por stalls cortos. La conmutación es costosa en ciclos.
  - Como CPU lanza instrucciones de un nuevo thread, el pipeline debe ser vaciado.
  - El nuevo thread debe llenar el pipe antes de que las instrucciones empiecen a completarse.
- Ejemplos; IBM AS/400, Montecito (Itanium 9000), Sparc64 VI

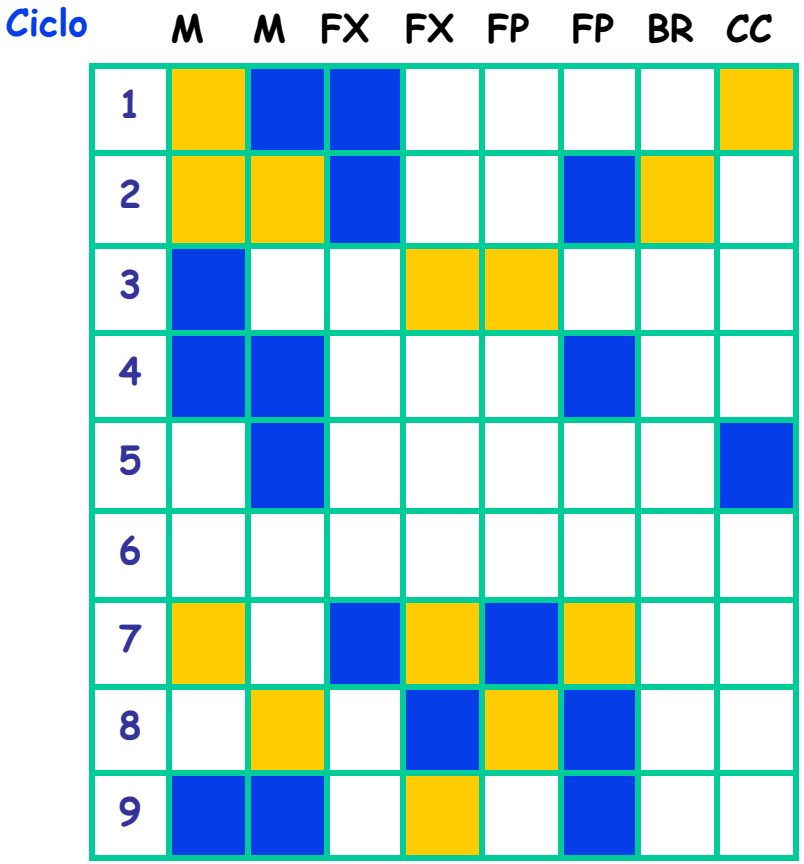
❑ Simultaneous Multi-threading

Motivación: Recursos no usados en un procesador superescalar

Un thread, 8 unidades



Dos threads, 8 unidades



M = Load/Store, FX = Fixed Point, FP = Floating Point, BR = Branch, CC = Condition Codes

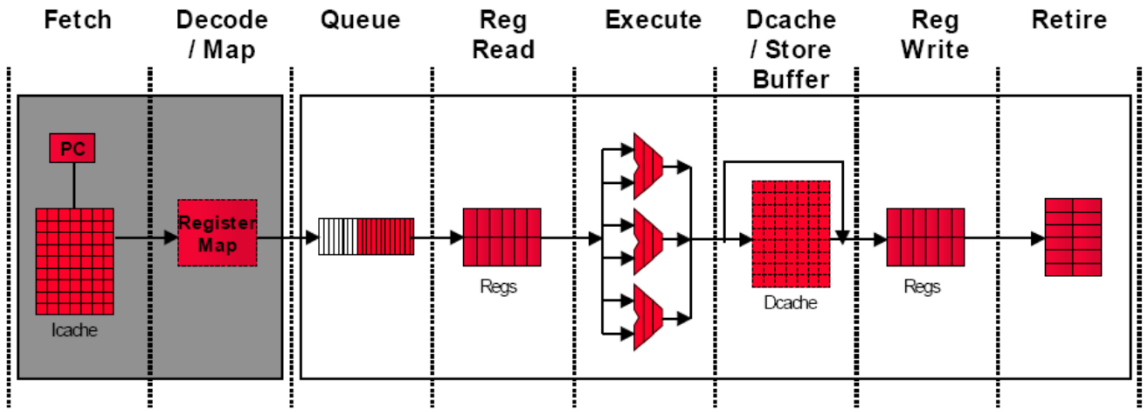
Simultaneous Multithreading (SMT)

- Simultaneous multithreading (SMT): dentro de un procesador superescalar fuera de orden ya hay mecanismos Hw para soportar la ejecución de más de un thread
  - Gran numero de registros físicos donde poder mapear los registros arquitectónicos de los diferentes threads
  - El renombrado de registros proporciona un identificador único para los operandos de una instrucción, por tanto instrucciones de diferentes thread se pueden mezclar sin confundir sus operados
  - La ejecución fuera de orden permite una utilización eficaz de los recursos.
- Solo necesitamos sumar una tabla de renombrado por thread y PC separados
  - Commit independiente se soporta con un ROB por thread (Lógico o físico)
- Ojo conflictos en la jerarquía de memoria
- Ejemplos; Pentium4, Power5 y 6, Nehalem (2008)

SMT Multithreading

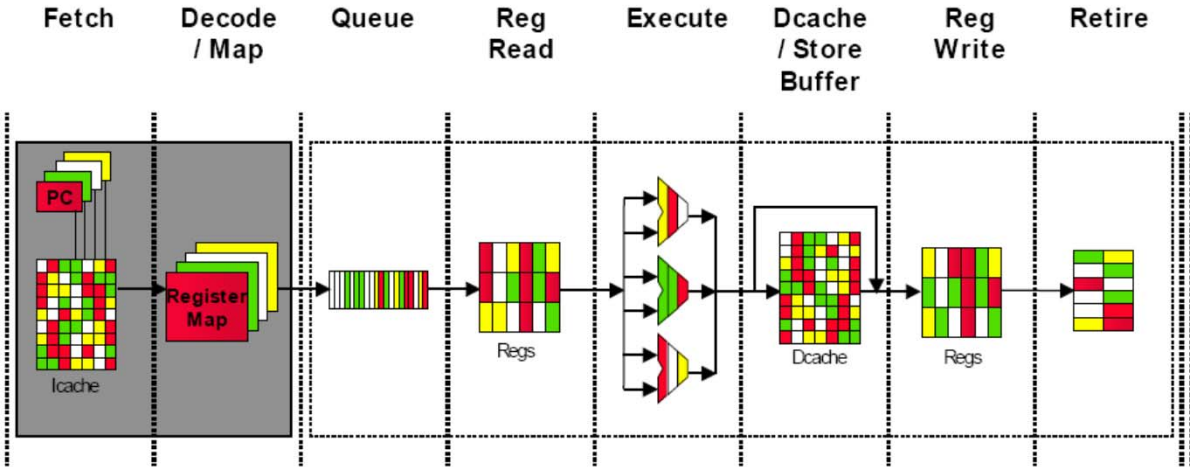
❑ Simultaneous multithreading

- ❑ Un solo hilo: un flujo de instrucciones



✓ todos los recursos utilizados por un hilo

- ❑ Multihilo: varios flujos de instrucciones

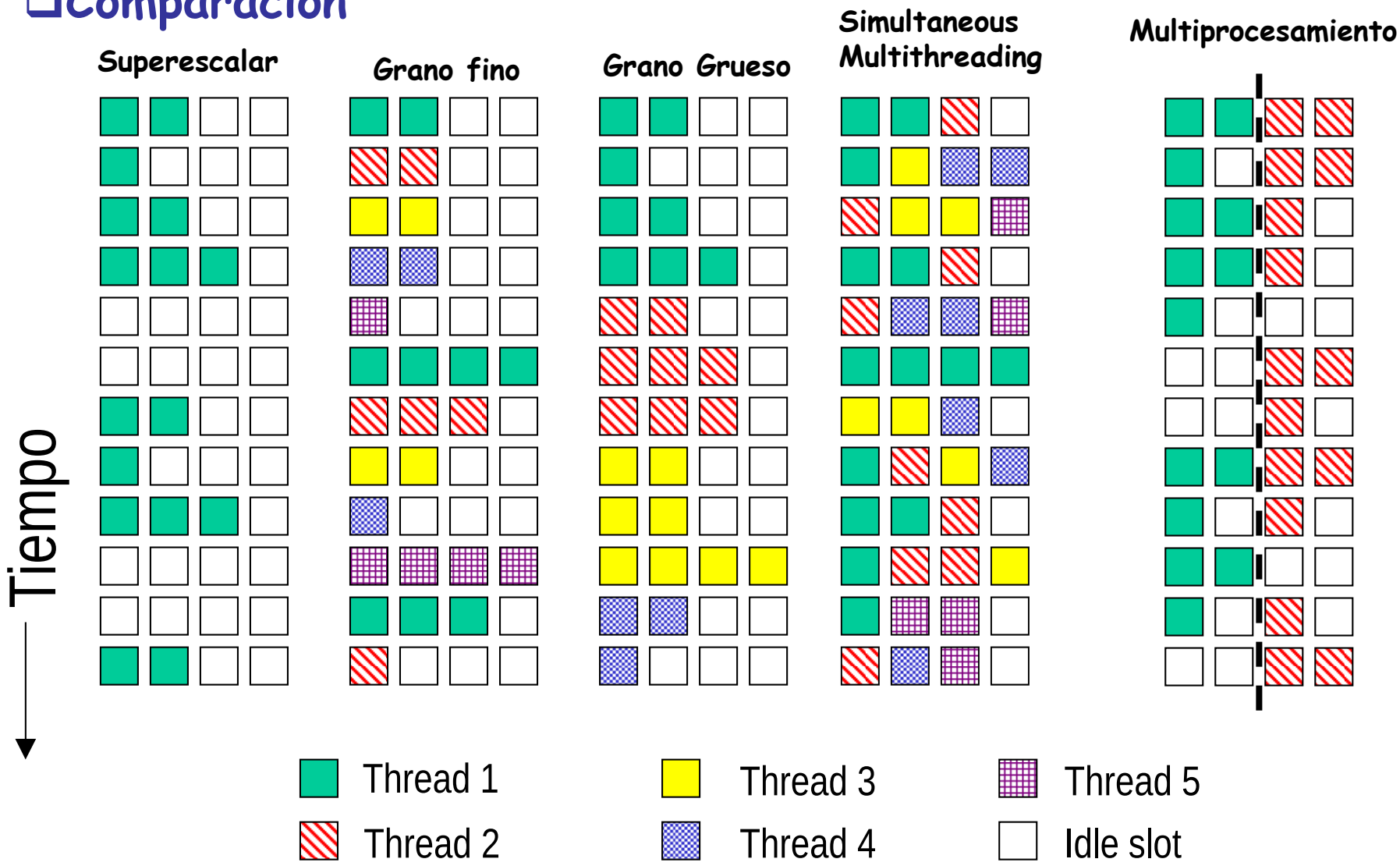


✓ recursos para distinguir el estado de los hilos

✓ los otros recursos se pueden compartir

Multithreading

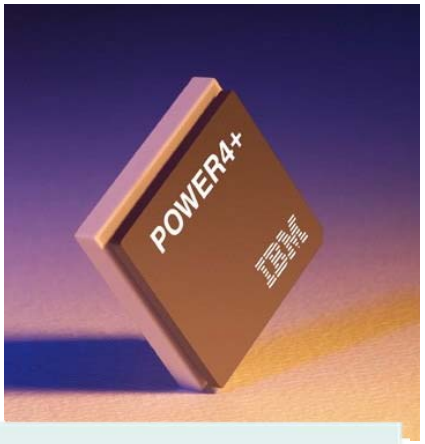
❑Comparación

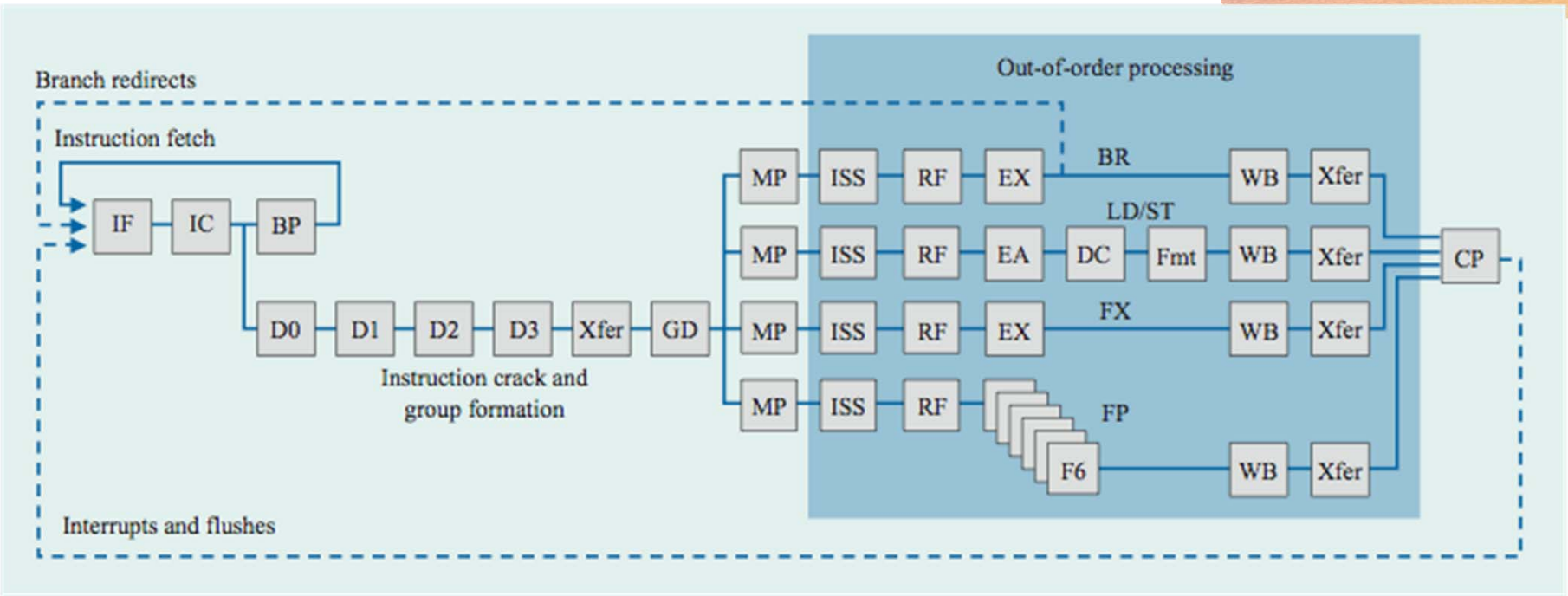


SMT Multithreading

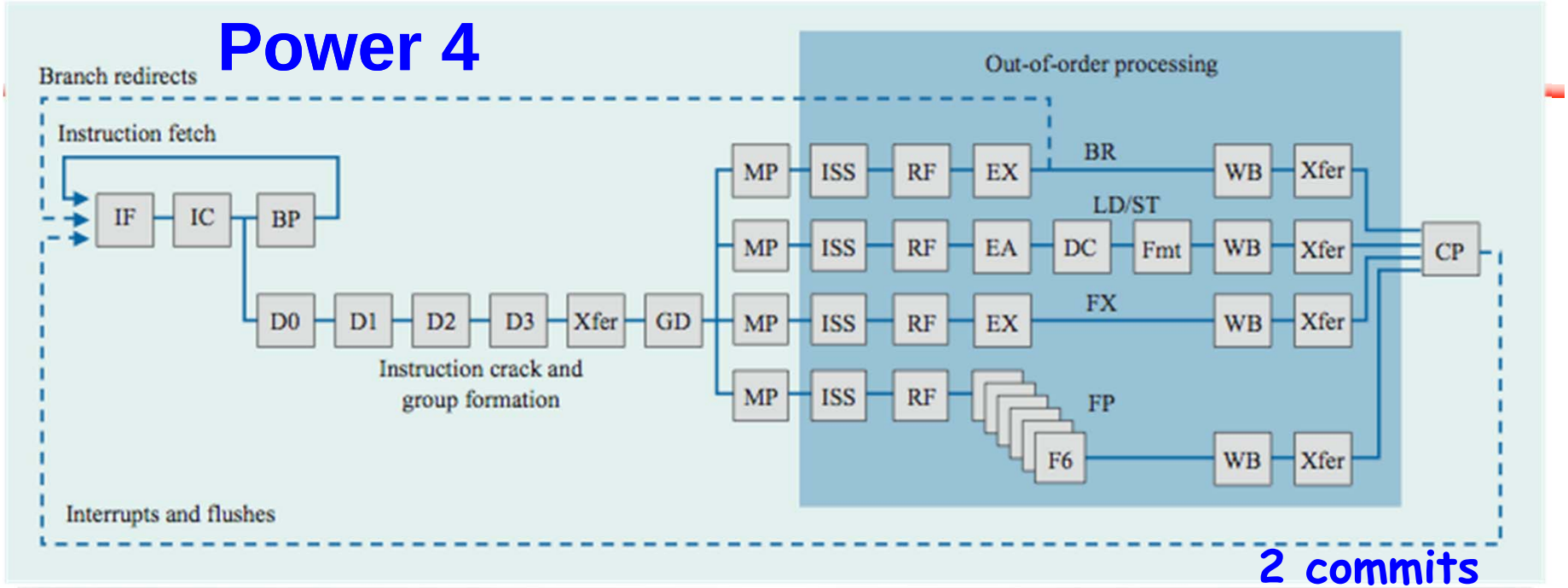
- Power 4 (IBM 2000)

Predecesor Single-threaded del Power 5.
8 unidades de ejecución fuera de orden



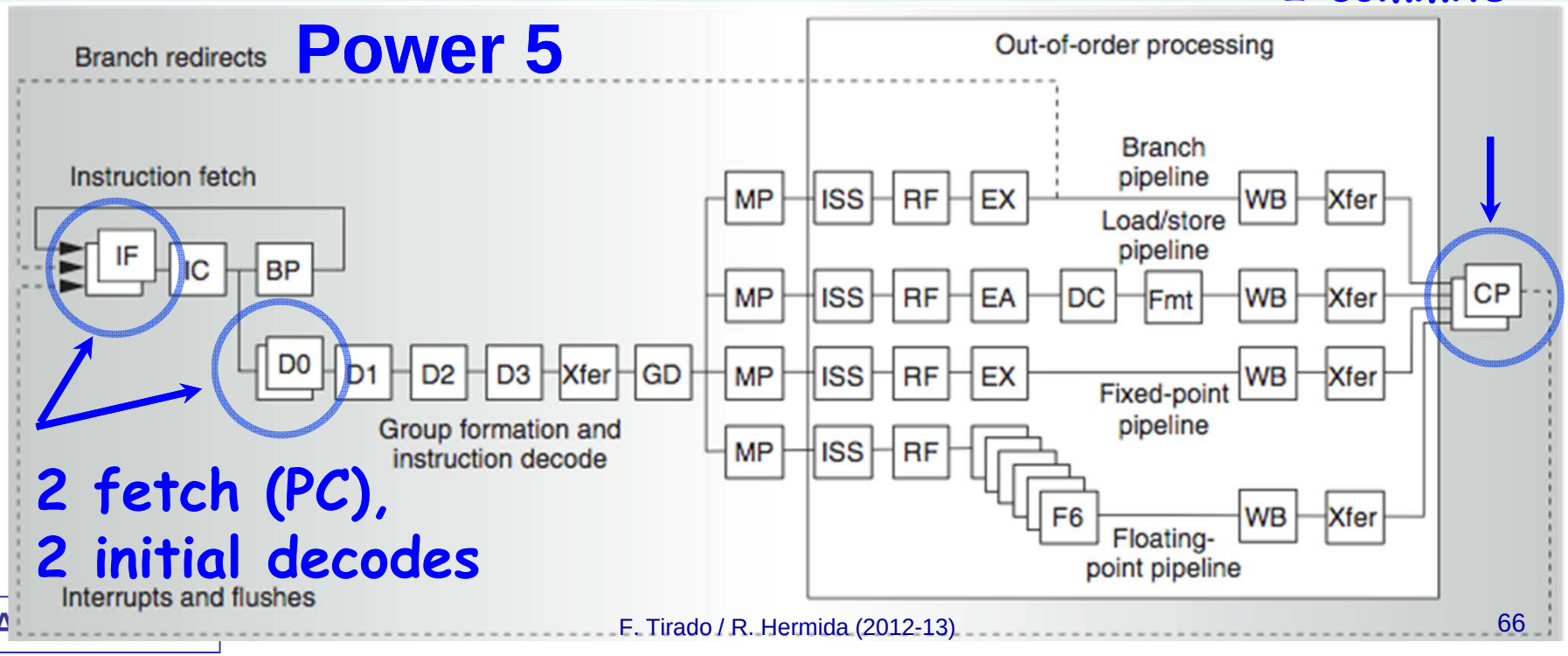


Power 4



2 commits

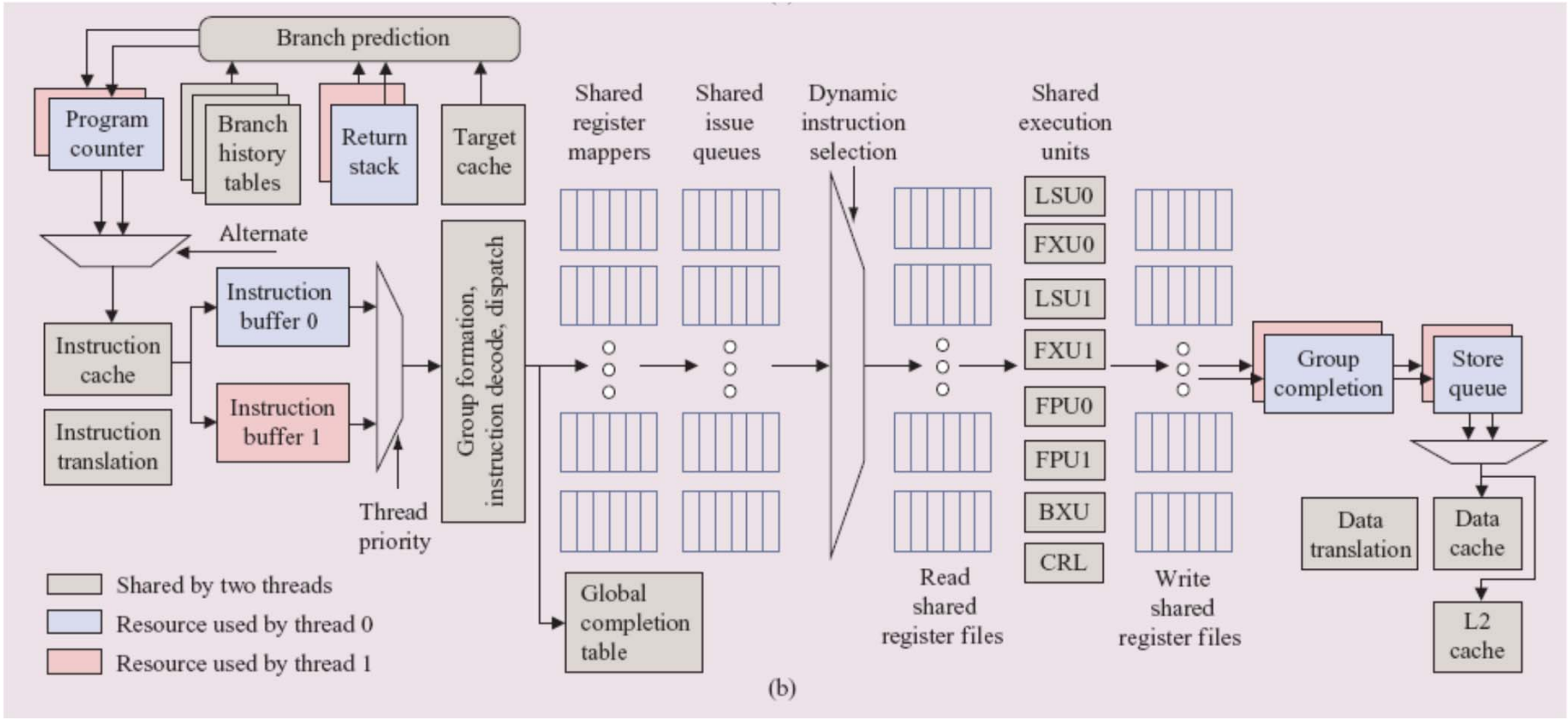
Power 5



2 fetch (PC),
2 initial decodes

SMT Multithreading

❑ Power 5 (IBM 2005)



¿Por qué sólo 2 threads? Con 4, los recursos compartidos (registros físicos , cache, AB a memoria) son un cuello de botella.

SMT Multithreading

❑ Power 5

Balanceo de la carga dinámica

1- Monitorizar

cola de fallos (load en L2)

entradas ocupadas en ROB (GCT)

2 – Quitarle recursos;

Reducir prioridad del hilo,

Inhibir decodificación(L2 miss)

Eliminar instrucciones desde emisión y parar decodificación

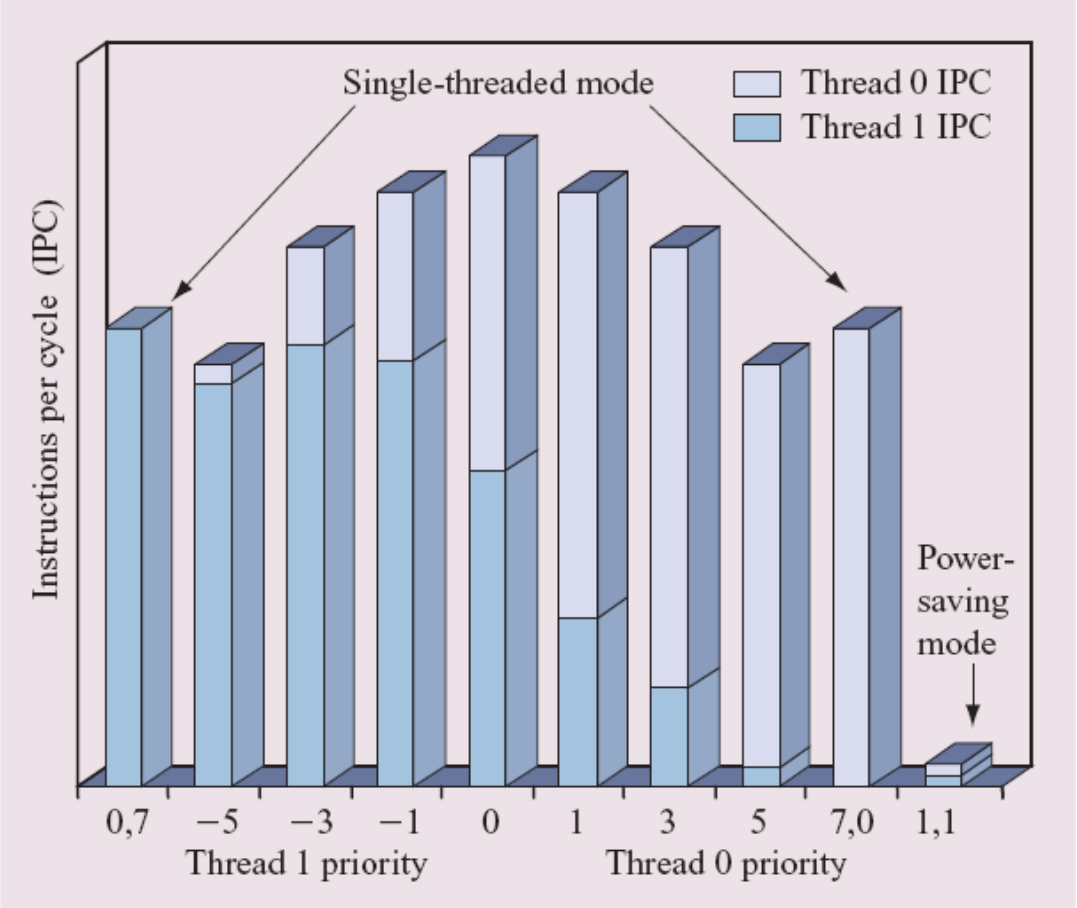
·3- Ajustar prioridad del hilo (hardware/software)

Baja en espera activa

Alta en tiempo real

8 niveles de prioridad

Da mas ciclos de decodificación al de mas prioridad

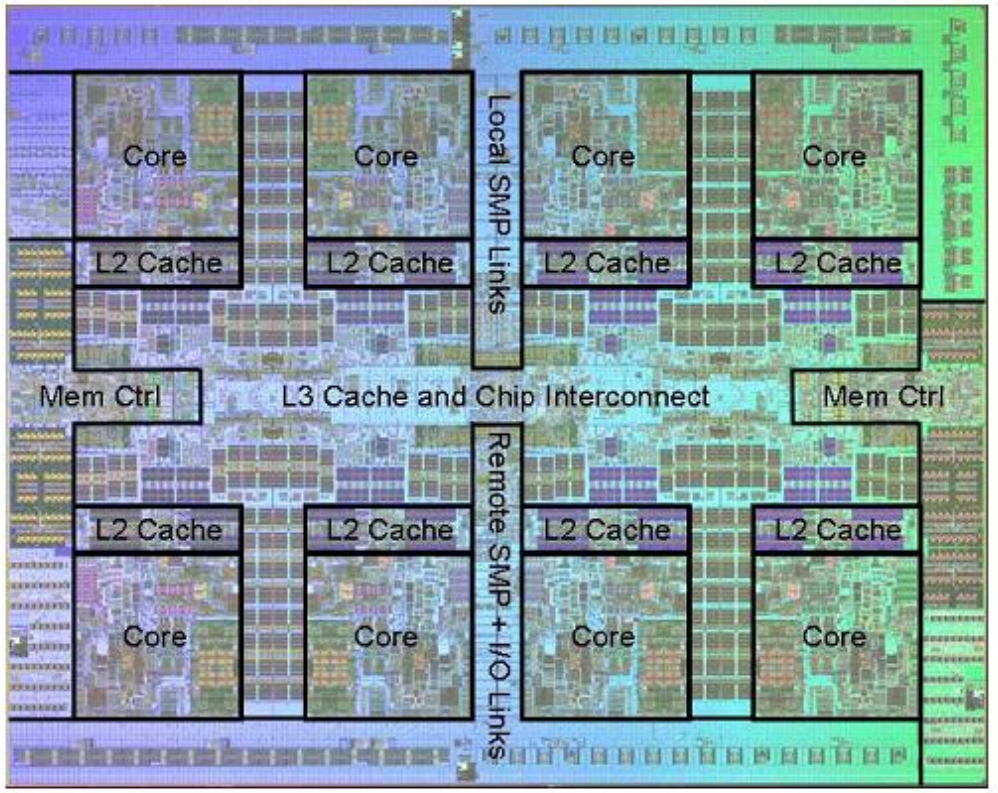


SMT Multithreading

- **Cambios en Power 5 para soportar SMT**
  - Incrementar asociatividad de la L1 de instrucciones y del TLB
  - Una cola de load/stores por thread
  - Incremento de tamaño de la L2 (1.92 vs. 1.44 MB) y L3
  - Un buffer de prebúsqueda separado por thread
  - Incrementar el numero de registros físicos de 152 a 240
  - Incrementar el tamaño de la colas de emisión
  - El Power5 core es 24% mayor que el del Power4 para soportar SMT
  - Más consumo, pero soportado con DVS

POWER7 Processor Chip

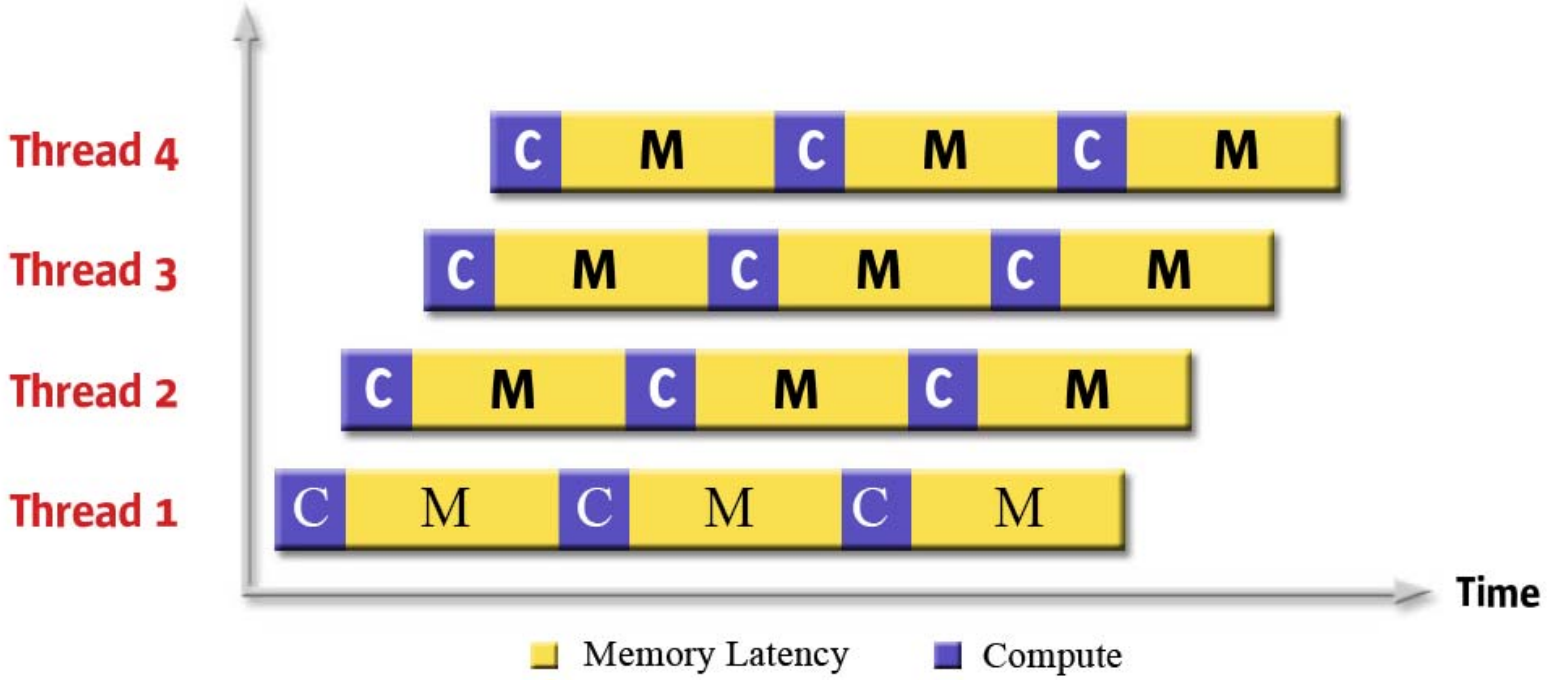
- 567mm² Technology: 45nm lithography, Cu, SOI, eDRAM
- 1.2B transistors
  - Equivalent function of 2.7B
  - eDRAM efficiency
- Eight processor cores
  - 12 execution units per core
  - 4 Way SMT per core
  - 32 Threads per chip
  - 256KB L2 per core
- 32MB on chip eDRAM shared L3
- Dual DDR3 Memory Controllers
  - 100GB/s Memory bandwidth per chip sustained
- Scalability up to 32 Sockets
  - 360GB/s SMP bandwidth/chip
  - 20,000 coherent operations in flight
- Advanced pre-fetching Data and Instruction
- Binary Compatibility with POWER6



IBM Blue Water NCSA Illinois 2011

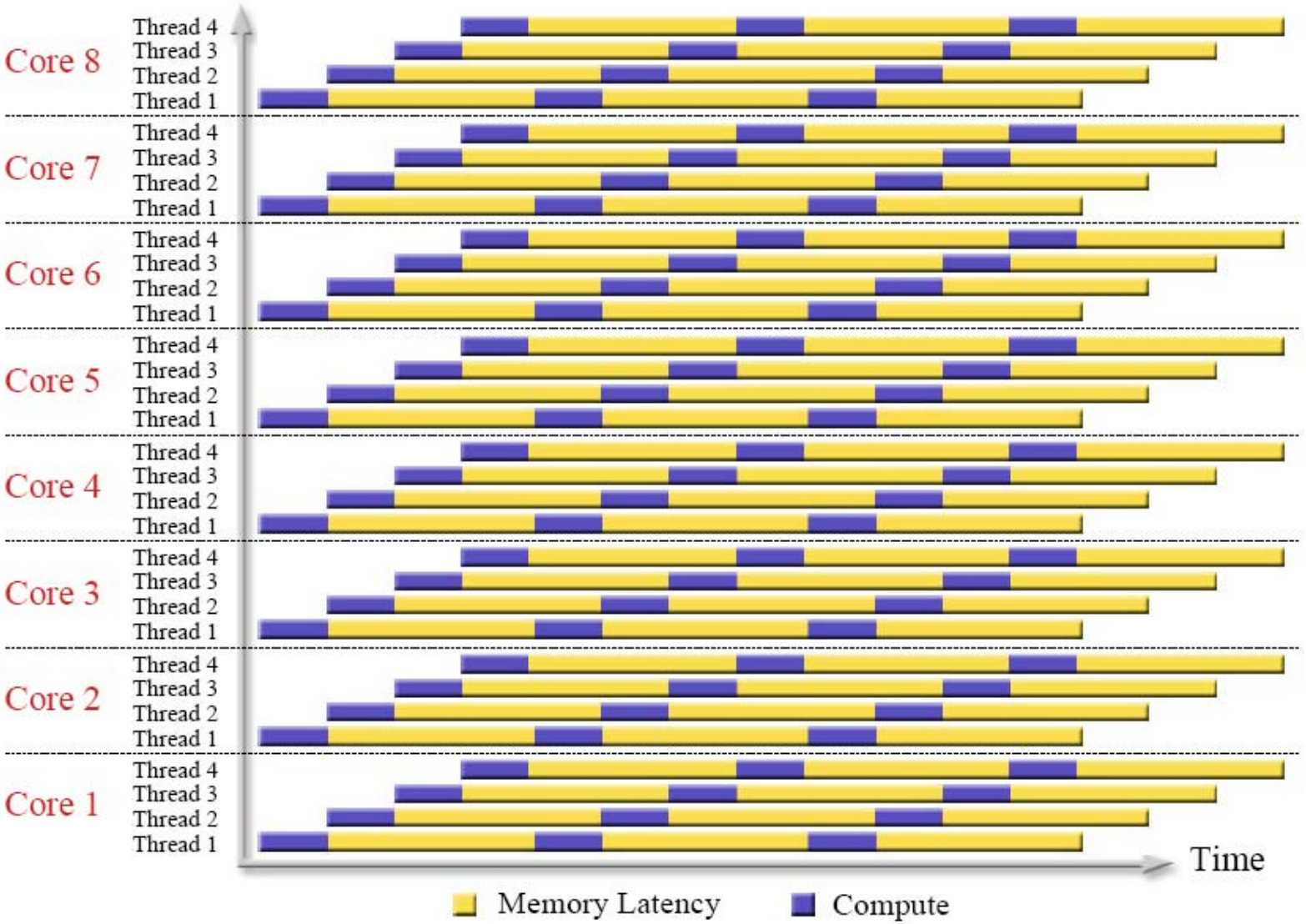
❑ Niagara (SUN 2005)

❑ Tolerar (soportar) la latencia de memoria mediante hilos concurrentes

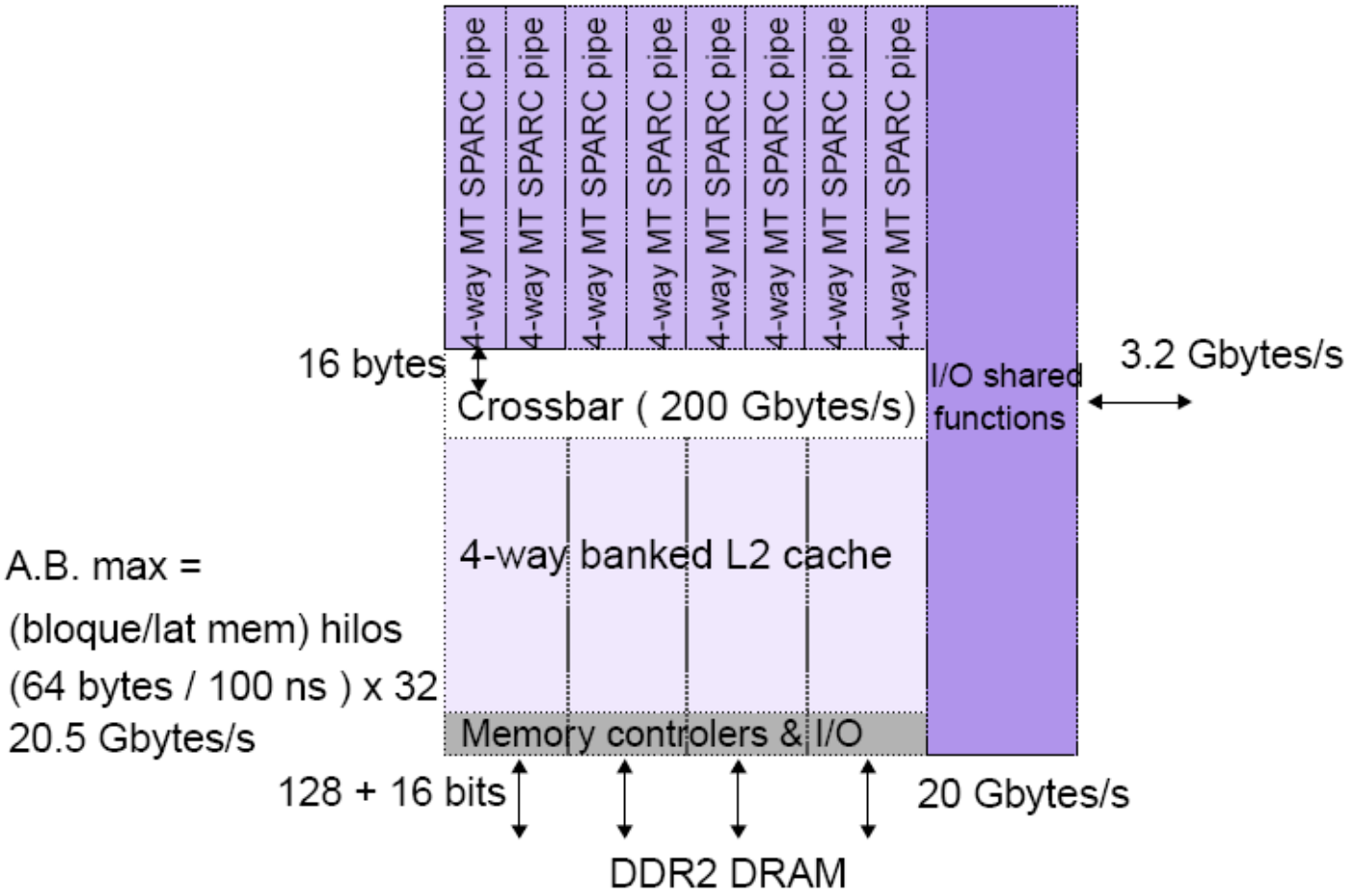


✓ Incrementa la utilización del procesador
✓ Es necesario un gran ancho de banda
- 4 accesos concurrentes a memoria

❑ Niagara: Múltiples cores-múltiples thread

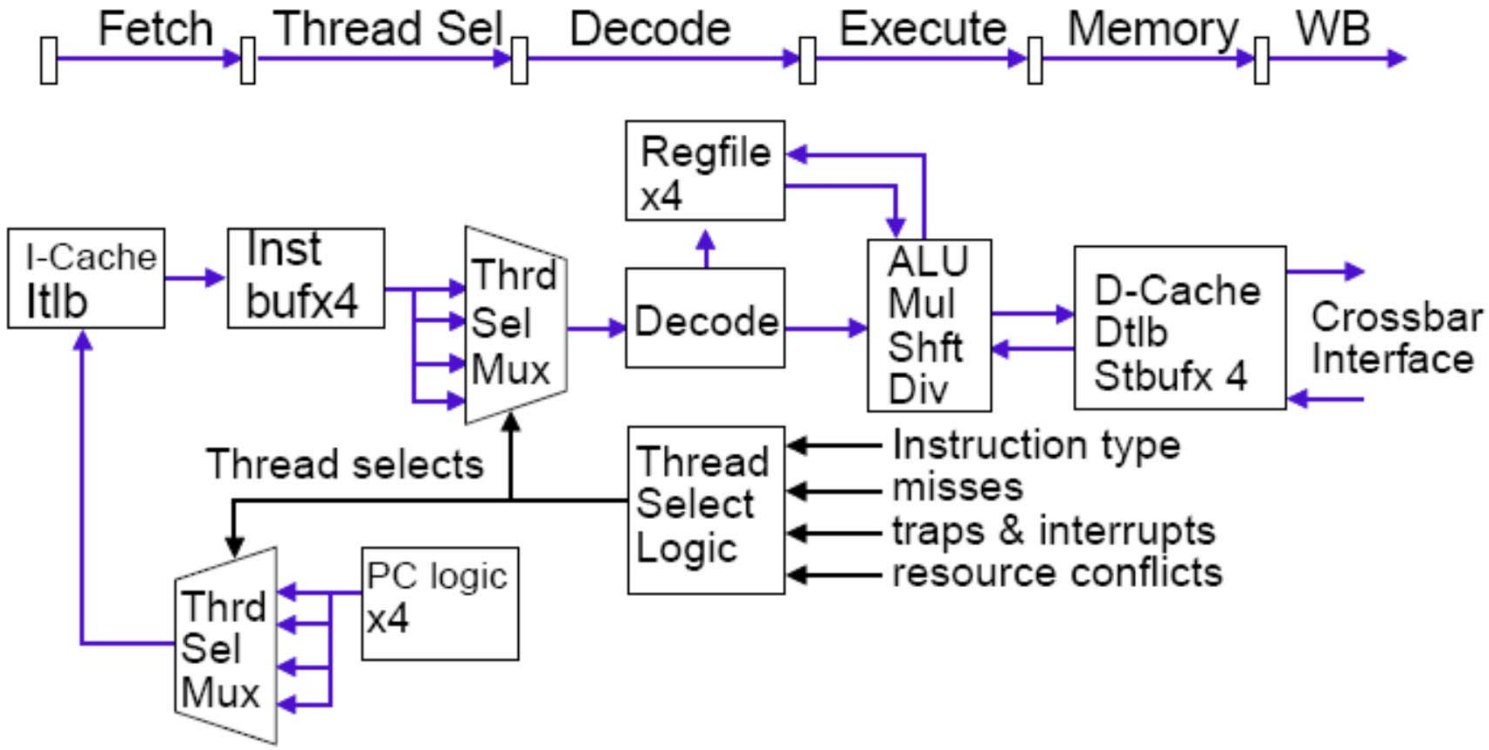


Niagara UltraSparc T1



1Ghz, 1 instrucción por ciclo, 4 thread por core, 60 vatios
I-L1 16Kb(4-Way) / D-L1 8Kb (4-Way), escritura directa / L2, 3Mb(12-Way)
Crossbar no bloqueante, No predictor de saltos, no FP (1 por chip)

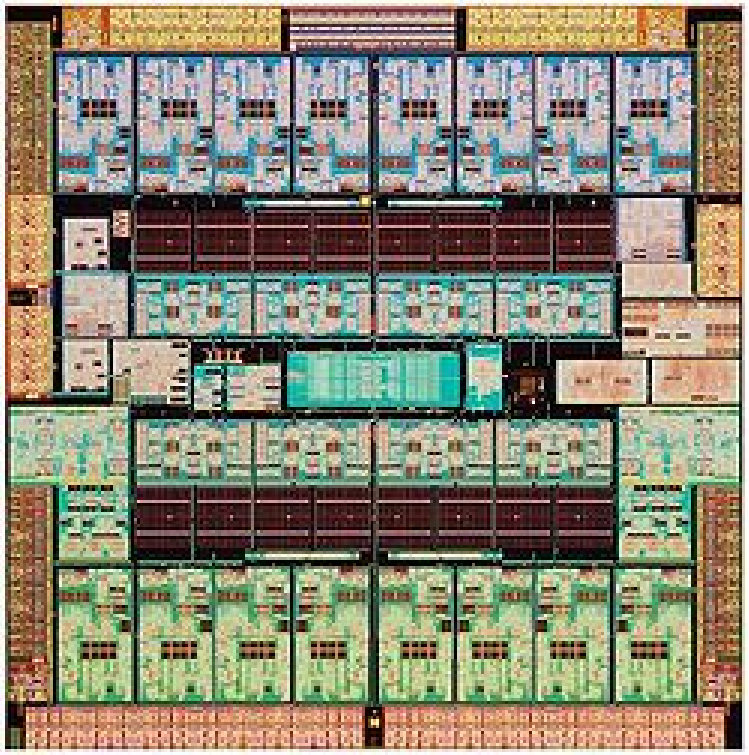
Niagara UltraSparcT1

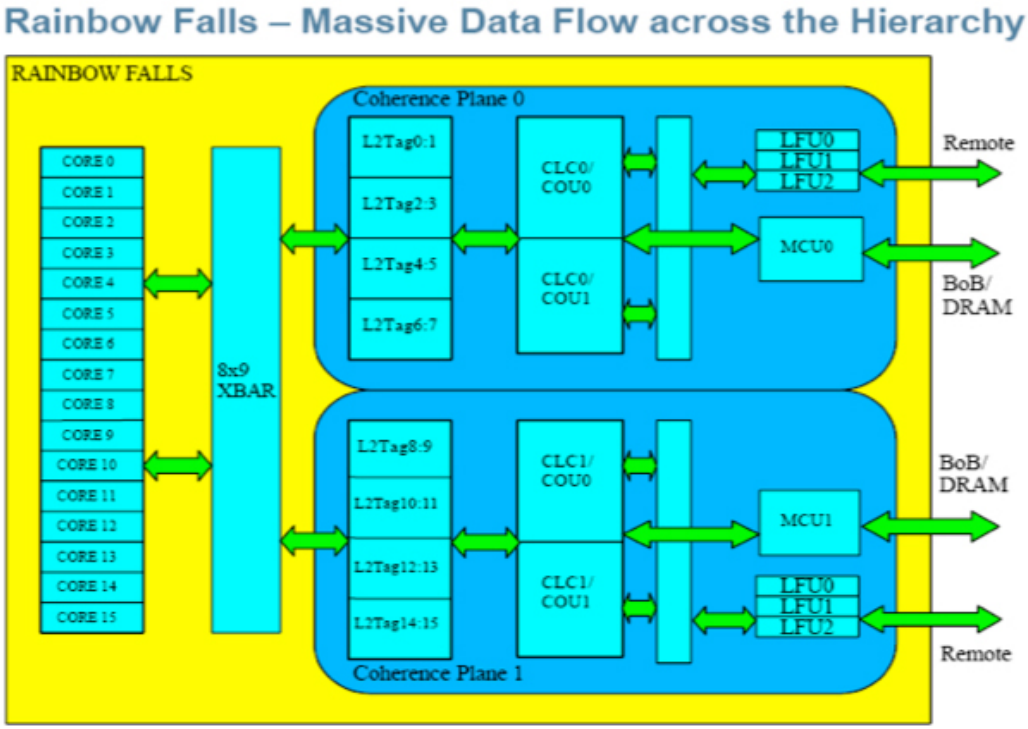


- 6 etapas
- 4 thread independientes
- algunos recursos x4: Banco de registros, contador de programa, store buffer, buffer de instrucciones
- el control de selección de hilo determina el hilo en cada ciclo de reloj
  - ✓cada ciclo elige un hilo

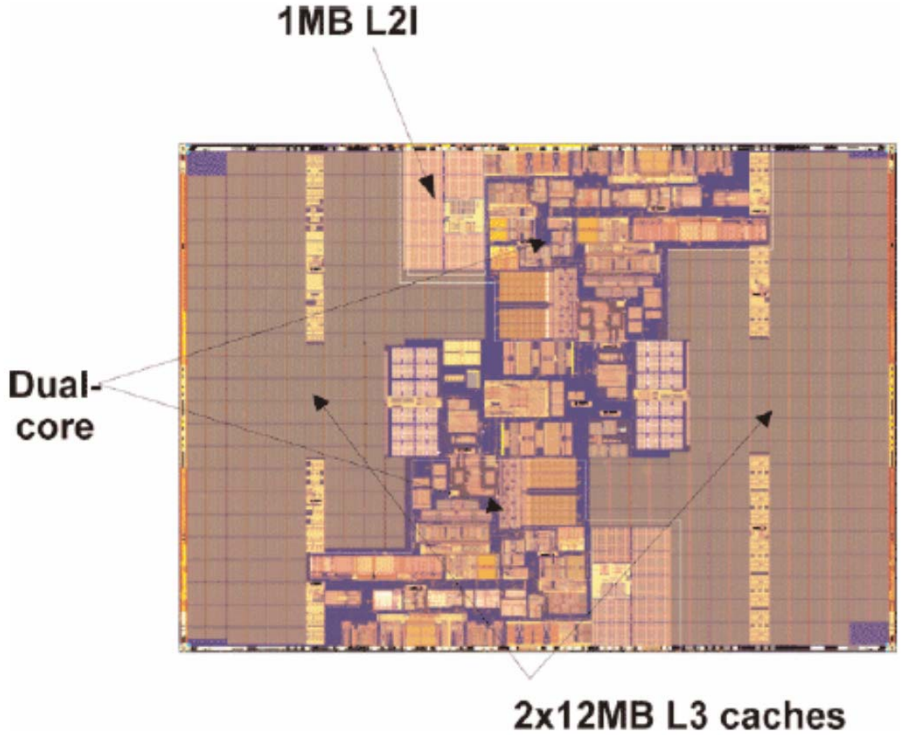
3 Generación Rainbow Falls (UltraSparc T3)

16 cores/ 8 threads. 128 threads por chip. 2Ghz. 40nm, 1000MTrans, 377mm^2
Nodos de 4 socket, 512 threads
16 bancos de L2 compartidos via crossbar (8x9). Coherencia





❑ Itanium2 9000



Itanium2 9000- Montecito

1720 Mtrans, 90 nm, 595 mm^2, 104 W
1,6 Ghz, 2 cores y 2 threads/core
Cache L2 separada
- 1 MByte L2 Icache
- 256KB L2 Dcache

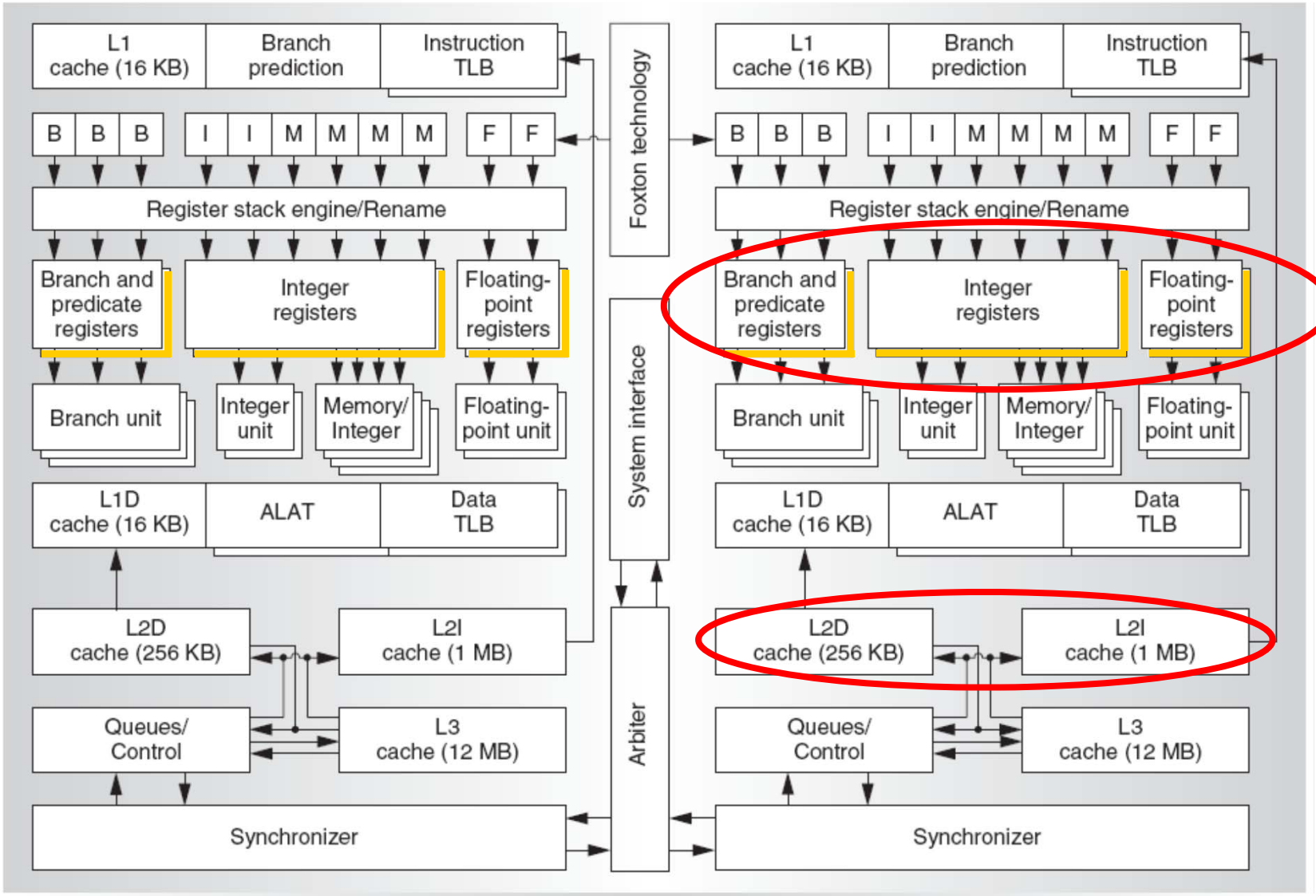
L3 mas grande
- L3 crece a 12 Mbyte por core (24MB)
- Mantiene latencia Itanium® 2

Colas y Control
- Más L3 y L2 victim buffers
- Mejor control de las colas

10,66 GBps AB con memoria

VLIW EPIC-IA 64+Multithreading grano grueso

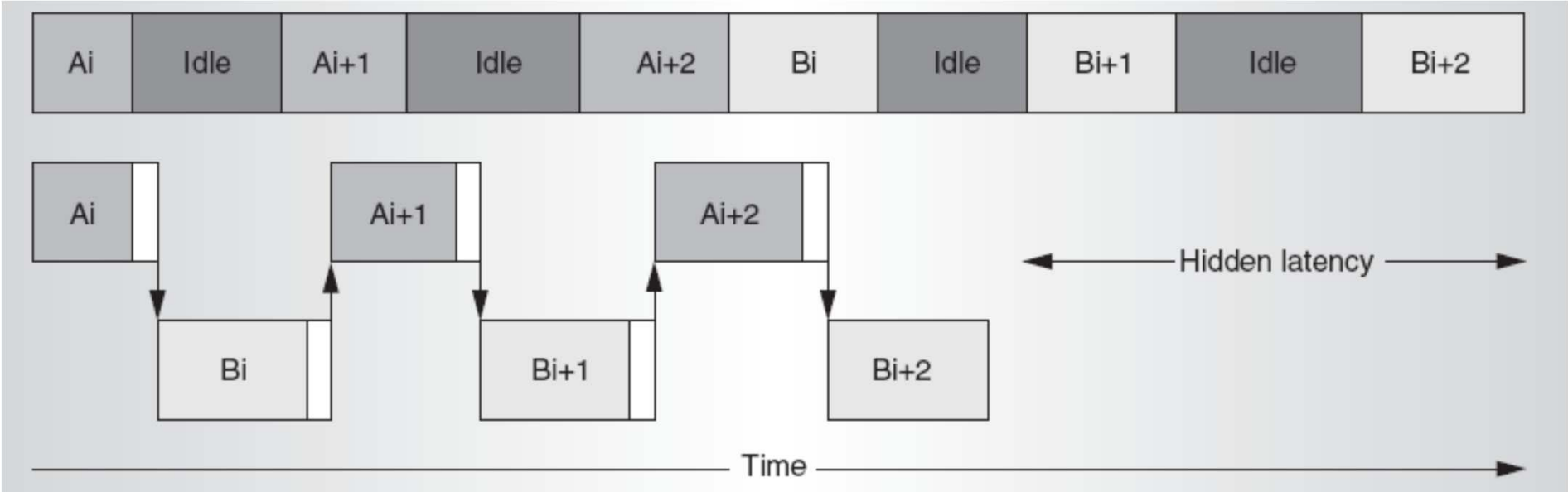
- Itanium2 9000



VLIW EPIC-IA 64 +Multithreading grano grueso

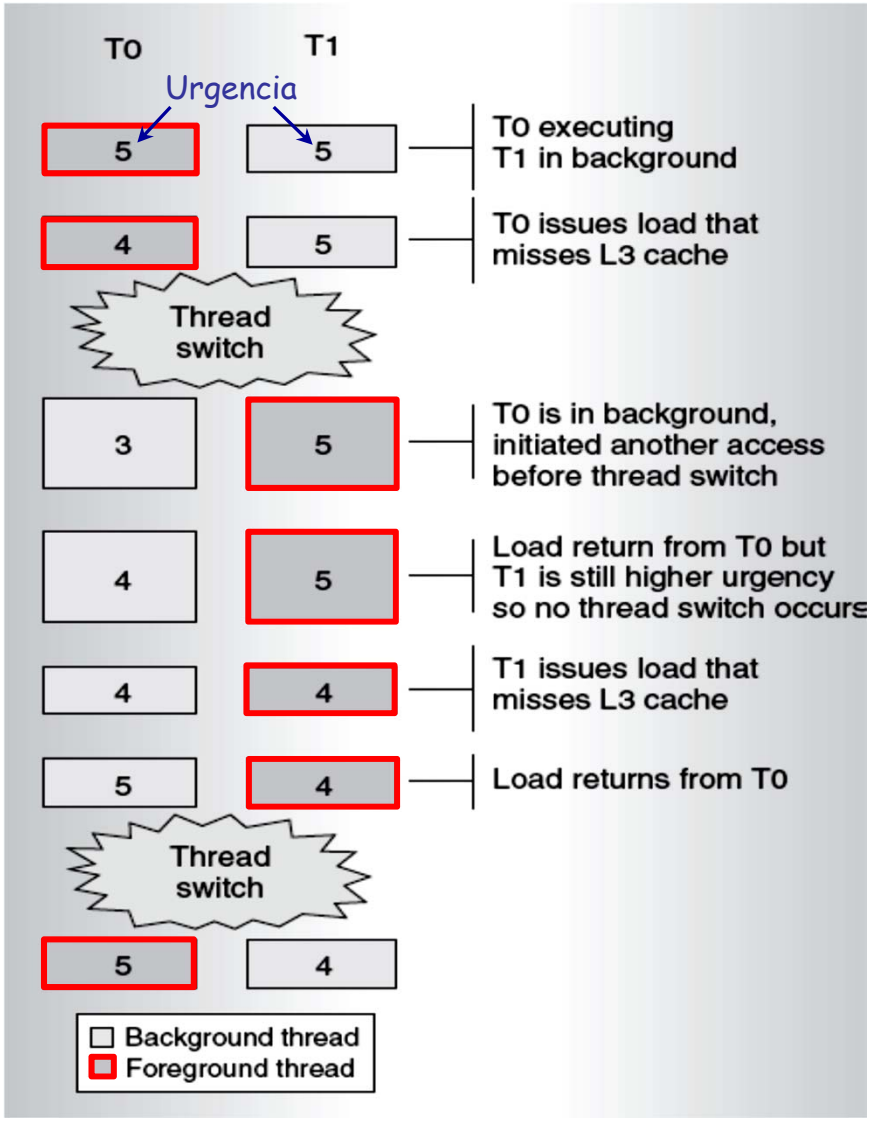
❑ Itanium2 9000 Multi-Threading

- Dinámicamente asigna recursos en función del el uso efectivo a realizar.
  - Un evento de alta latencia determina la cesión de recursos por parte del thread activo.



VLIW EPIC-IA 64+Multithreading grano grueso

❑ Montecito Multi-Threading



Conmutación de Threads

- Eventos de alta latencia producen un "stall" de ejecución
  - L3 misses o accesos a datos no "cacheable"
  - Agotar el "time slice" asignado a un thread

- Prioridad permite la conmutación de thread

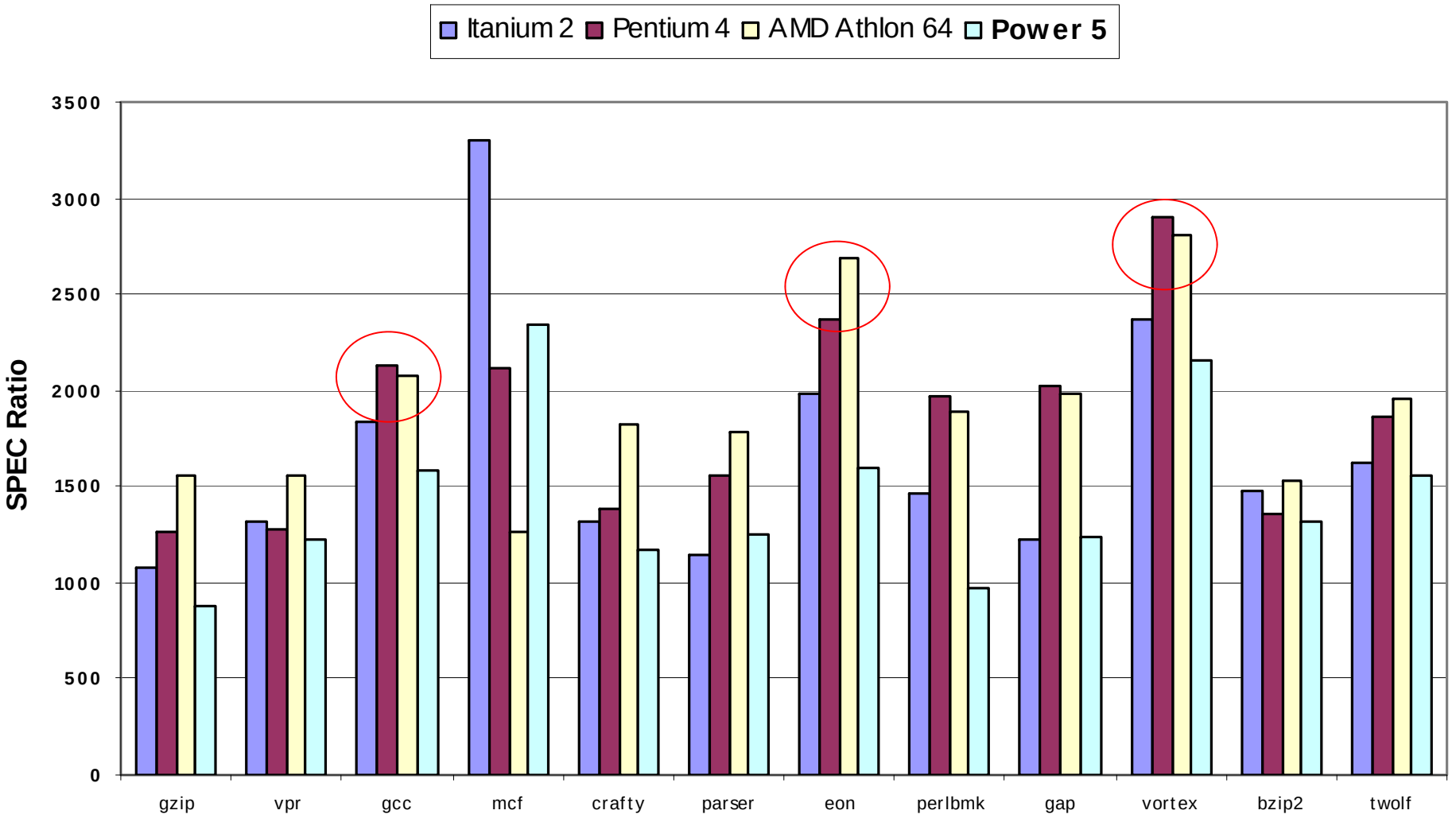
Rendimiento

¿ Quien es mejor?

Procesador	Microarchitectura	Fetch / Issue / Execute	FU	Clock Rate (GHz)	Transis -tores Die size	Power
Intel Pentium 4 Extreme	Especulativo con planificación dinámica; Pipe profundo; SMT	3/3/4	7 int. 1 FP	3.8	125 M 122 mm^2	115 W
AMD Athlon 64 FX-57	Especulativo con planificación dinámica.	3/3/4	6 int. 3 FP	2.8	114 M 115 mm^2	104 W
IBM Power5 (1 CPU only)	Especulativo con planificación dinámica; 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	Planificación estática VLIW-style	6/5/11	9 int. 2 FP	1.6	592 M 423 mm^2	130 W

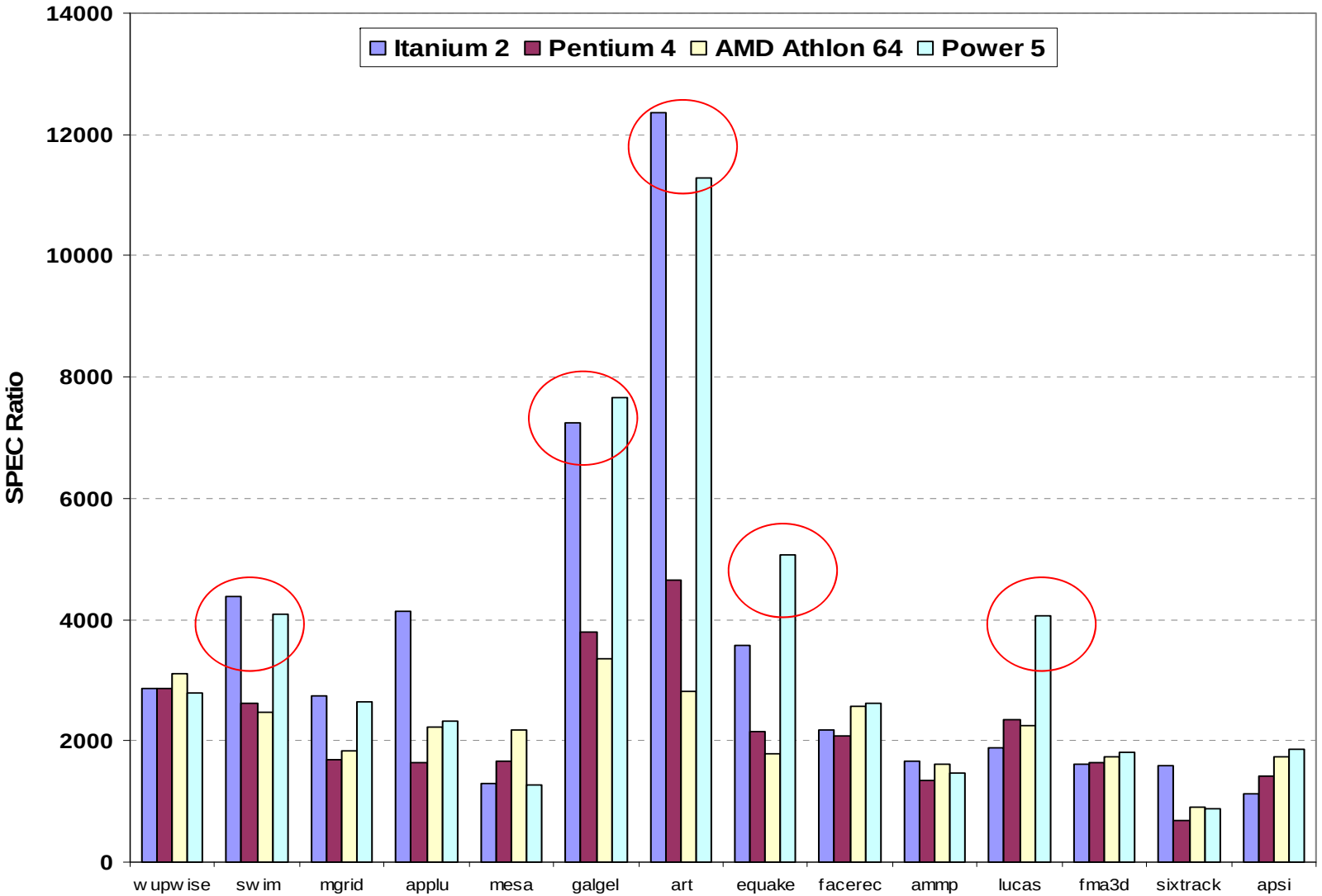
Rendimiento

SPECint2000



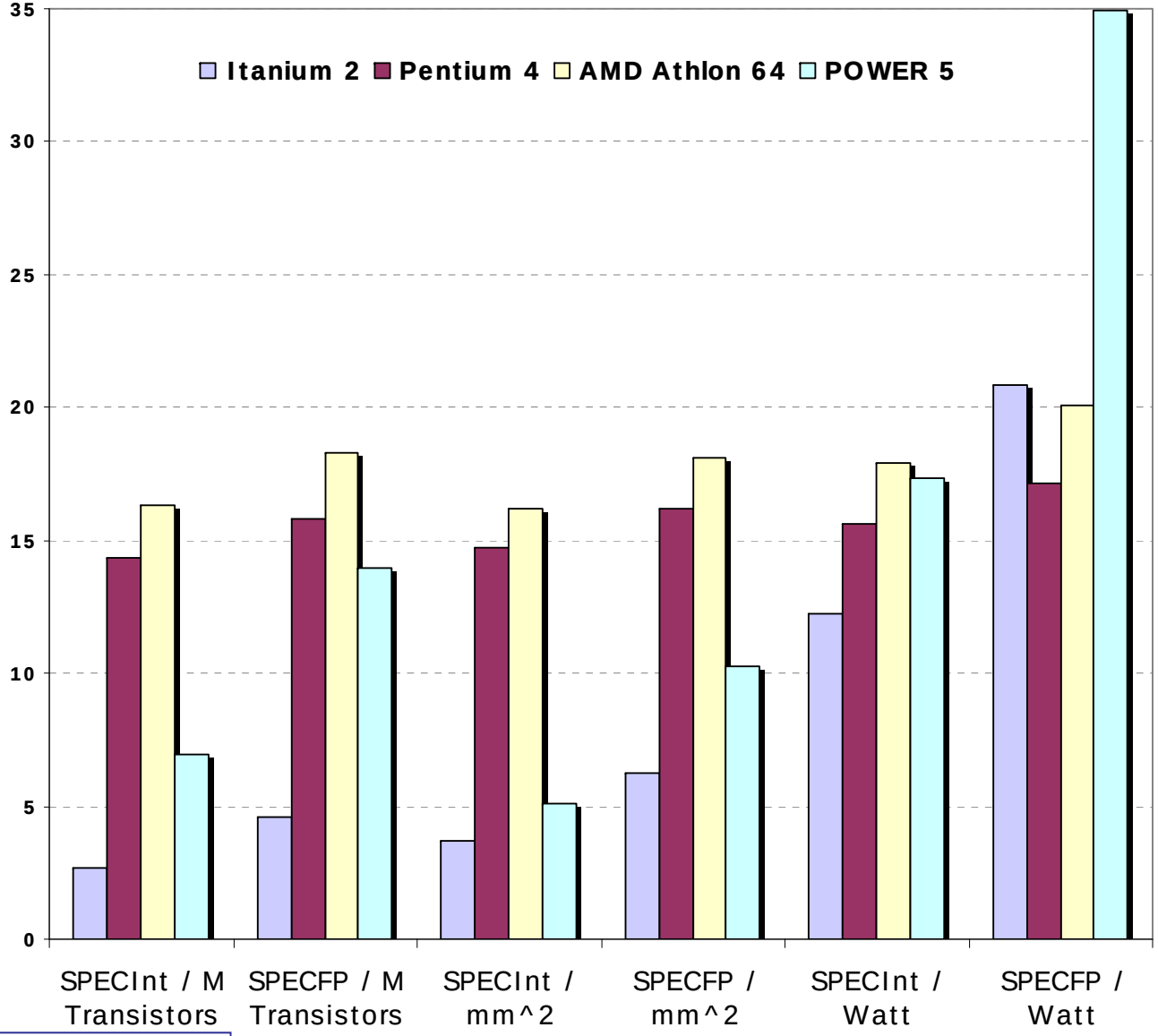
Rendimiento

- SPECfp2000



Rendimiento

❑ Rendimiento normalizado: Eficiencia



Rank	Itanium 2	PentIum 4	Athlon	Power 5
Int/Trans	4	2	1	3
FP/Trans	4	2	1	3
Int/area	4	2	1	3
FP/area	4	2	1	3
Int/Watt	4	3	1	2
FP/Watt	2	4	3	1

Rendimiento Conclusiones

- No hay un claro ganador en todos los aspectos
- El AMD Athlon gana en SPECInt seguido por el Pentium4, Itanium 2, y Power5
- Itanium 2 y Power5, tienen similares rendimientos en SPECFP, dominan claramente al Athlon y Pentium 4
- Itanium 2 es el procesador menos eficiente en todas las medidas menos en SPECFP/Watt.
- Athlon y Pentium 4 usan bien los transistores y el área en términos de eficacia
- IBM Power5 es el mas eficaz en el uso de la energía sobre los SPECFP y algo menos sobre SPECINT

Conclusiones -Limites del ILP

- Doblar en ancho de emisión (issue rates) sobre los valores actuales 3-6 instrucciones por ciclo, a digamos 6 a 12 instrucciones requiere en el procesador
  - de 3 a 4 accesos a cache de datos por ciclo,
  - Predecir-resolver de 2 a 3 saltos por ciclo,
  - Renombrar y acceder a mas de 20 registros por ciclo,
  - Buscar de la cache de instrucciones de 12 a 24 instrucciones por ciclo.
- La complejidad de implementar estas capacidades implica al menos sacrificar la duración del ciclo e incrementa de forma muy importante el consumo.

Conclusiones -Limites del ILP

- La mayoría de la técnicas que incrementan rendimiento incrementan también el consumo.
- Una técnica es *eficiente en energía* si incrementa más el rendimiento que el consumo.
- Todas las técnicas de emisión múltiple son poco eficientes desde el punto de vista de la energía.

Conclusiones -Limites del ILP

- La arquitectura Itanium no representa un paso adelante en el incremento el ILP, eliminado los problemas de complejidad y consumo.
- En lugar de seguir explotando el ILP, los diseñadores se han focalizado sobre multiprocesadores en un chip (CMP, multicores,..)
- En el 2000, IBM abrió el campo con el 1º multiprocesador en un chip, el Power4, que contenía 2 procesadores Power3 y una cache L2 compartida. A partir de este punto todos los demás fabricantes han seguido el mismo camino. (Intel, AMD, Sun, Fujitsu,..).
- El balance entre ILP y TLP a nivel de chip no es todavía claro, y probablemente será muy dependiente del tipo de aplicaciones a que se dedique el sistema.