3. Explicit concurrency

Multi-cores - hardware multi-threading

Advances in Computer Architecture

The shift towards multi-cores

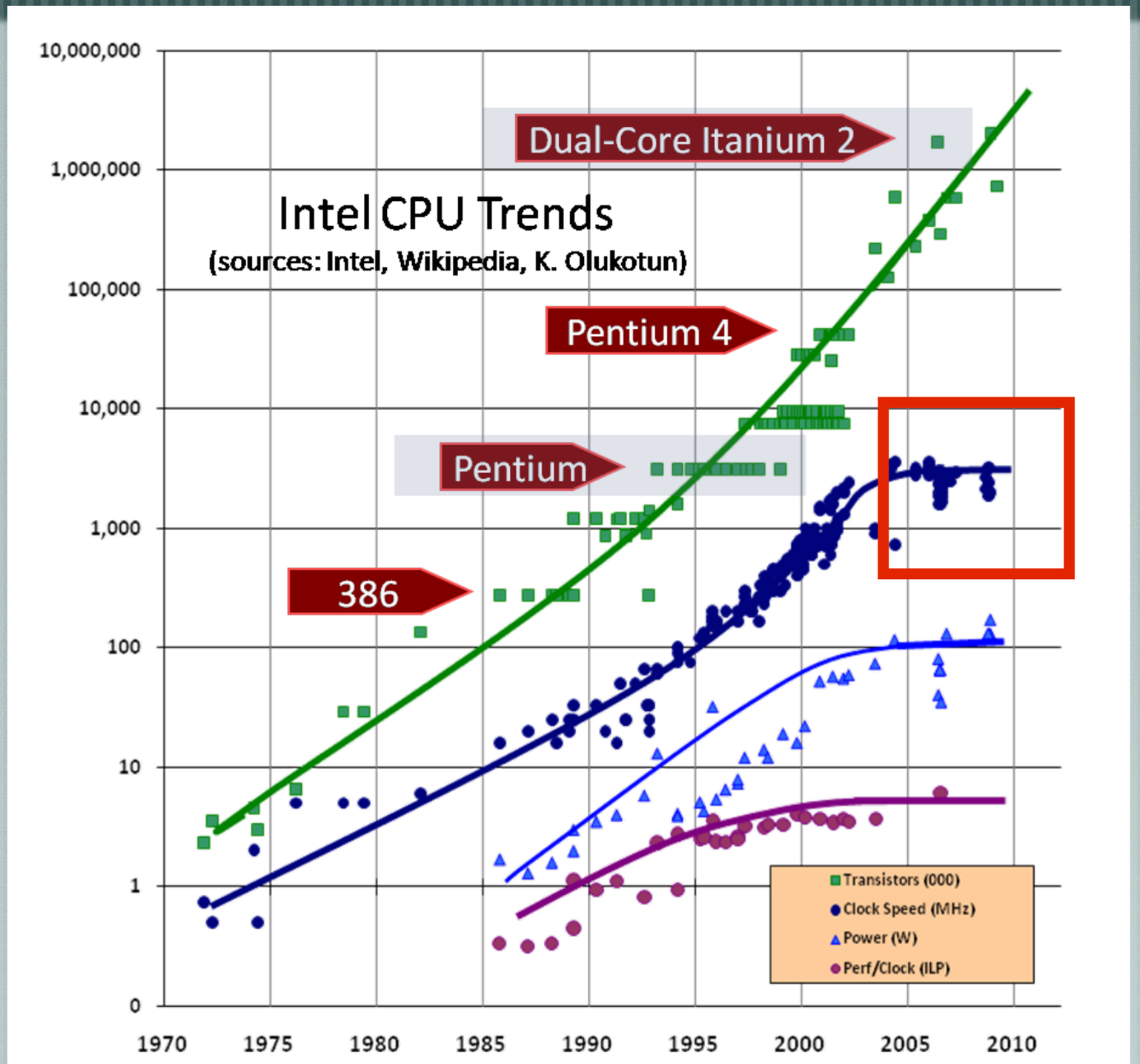
Trends

- During the mid 2000's, Intel (and previously DEC, Compaq) cancelled wide superscaler projects
  - eg. pentium Netburst and Alpha 21464
- The current trend is to develop independent microprocessors on chip – **multi-cores**
- **But why is this?**

We have hit several walls…

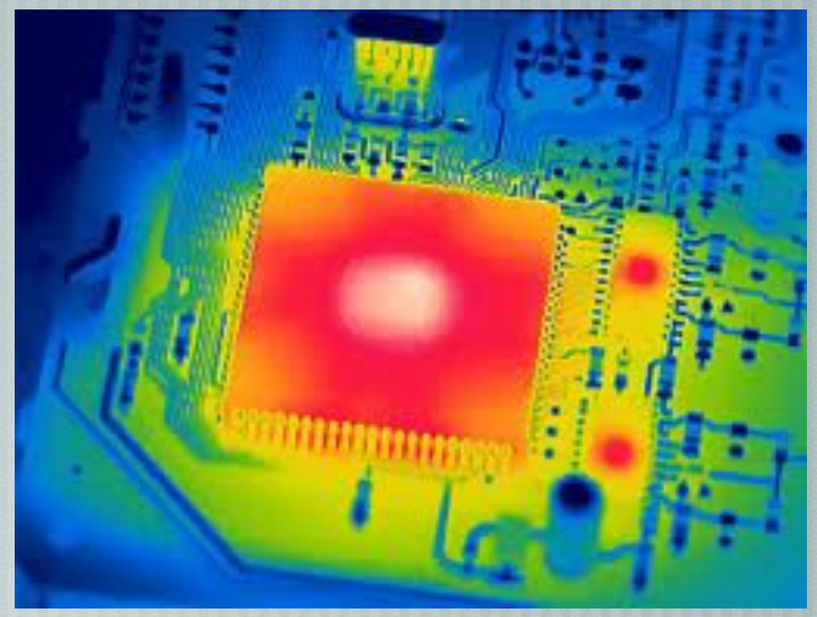
We have hit several walls…

The frequency wall

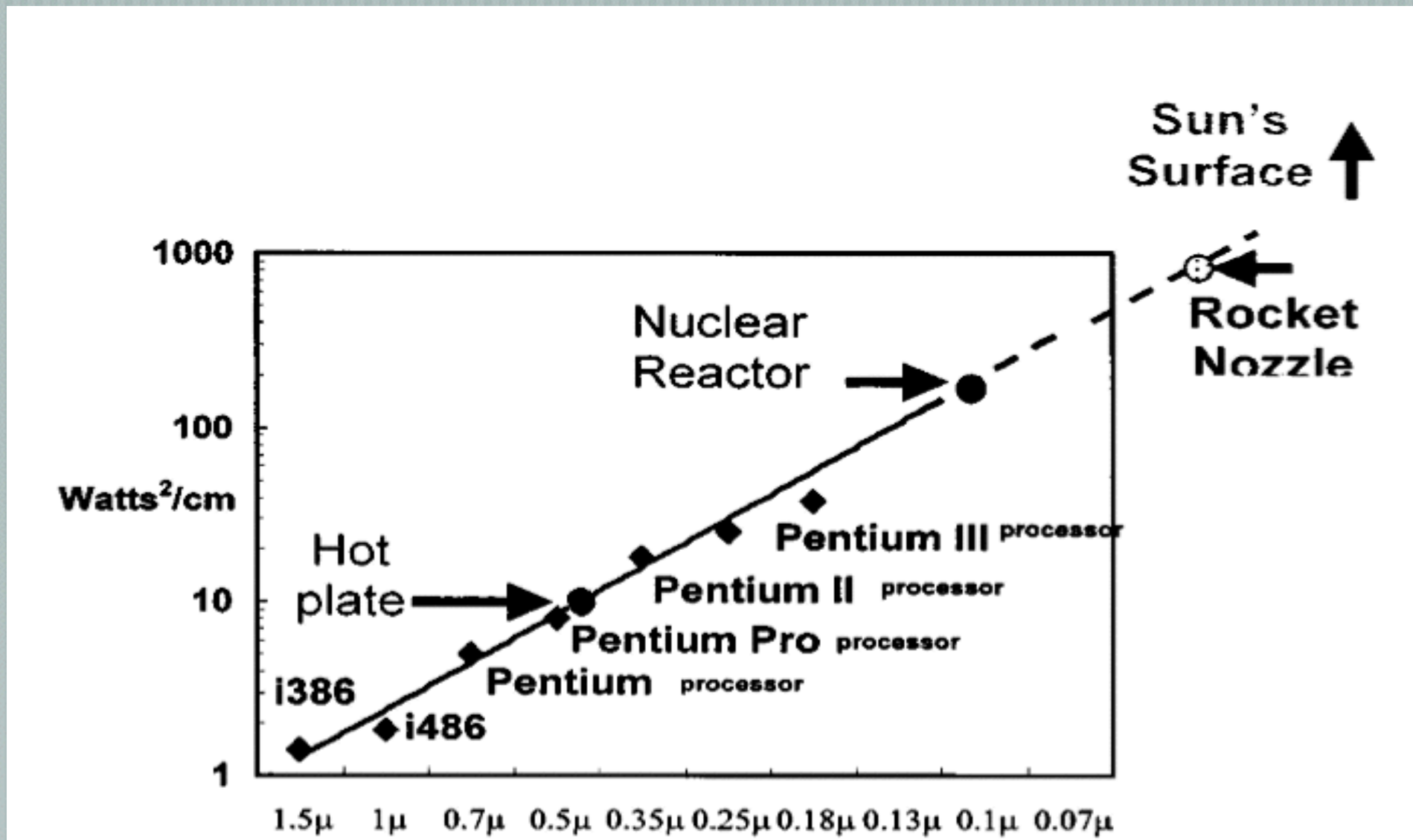


The power wall

- Power dissipated is a silicon CMOS circuit comprises several components and **the major component has been dynamic power**
- Dynamic power = $a \cdot C \cdot V^2 \cdot f$
  - a is activity, C is capacitance, V is voltage, and f is frequency
- Higher frequency requires higher voltages



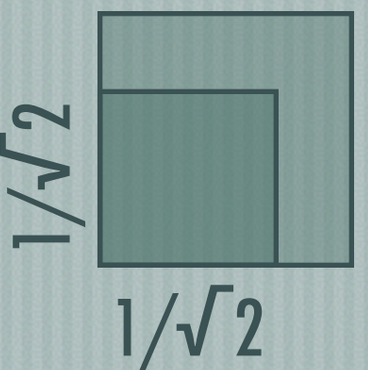
The power wall



Ronan et. al. Coming Challenges in Microarchitecture and Architecture, Proc IEEE, 89 (3) pp. 325-340, 2001

So, what happened around 2004?

Dennard Scaling



- Moore's Law: 2 times as many transistors every new technology generation, growth per dimension **k** = √2 (=1.4)
- Scaling factor for transistors: **1/k** = 0.7
- Area scales with $(1/k)^2$ = 0.5
- Voltage scales with **1/k** = 0.7
- Capacitance scales with **1/k** = 0.7
- Transistor delay scales with **1/k**
  - Frequency scales with 1/(1/k)=k=1.4

So, what happened around 2004?

Dennard Scaling

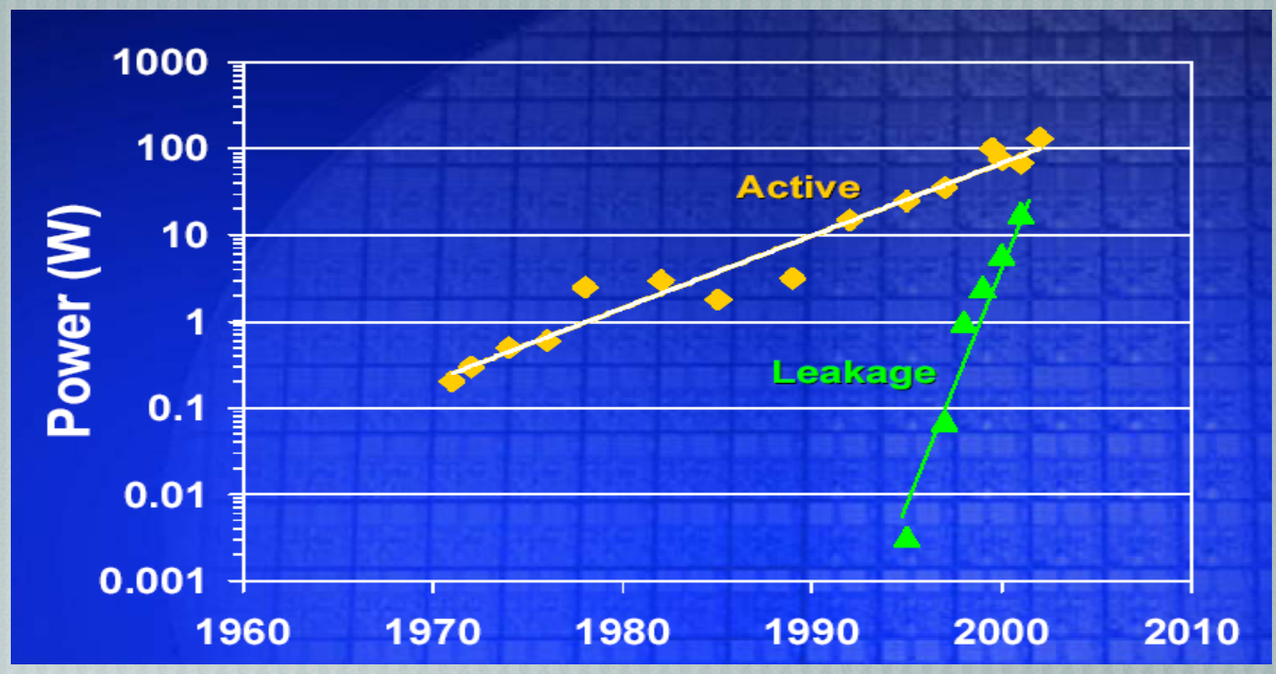
- Area (A) scales with 0.5
- Voltage (V) scales with 0.7
- Capacitance (C) scales with 0.7
- Frequency scales (F) with 1.4

$$\text{Power}_{\text{density}} \sim \frac{C \cdot V^2 \cdot f}{A}$$

$$\text{Power}_{\text{density}}\ \text{ratio} \sim (0.7 \cdot 0.7^2 \cdot 1.4) / 0.5 = 1\ (!)$$

So, what happened around 2004?

Dennard Scaling stopped around 2004!



Moore, ISSCC Keynote, 2003

- Voltage (V_{dd}) does not scale anymore
  - $V_{threshold}$ cannot decrease because of leakage power

So, what happened around 2004?

Dennard Scaling stopped around 2004!

- Area (A) scales with 0.5
- Voltage (V) does not scale
- Capacitance (C) scales with 0.7
- Frequency scales (F) with 1.4

$$\text{Power}_{\text{density}} \sim \frac{C \cdot V^2 \cdot f}{A}$$

$$\text{Power}_{\text{density}}\ \text{ratio} \sim (0.7 \cdot 1^2 \cdot 1.4) / 0.5 = 2$$

$$\text{more general: Power}_{\text{density}}\ \text{ratio} \sim ((1/k) \cdot 1^2 \cdot k) / (1/k)^2 = k^2$$

The end of Dennard Scaling

$$\text{Power}_{\text{density}}\ \text{ratio} \sim ((1/k) \cdot 1^2 \cdot k) / (1/k)^2 = k^2$$

Keep f constant: $\text{Power}_{\text{density}}\ \text{ratio} \sim ((1/k) \cdot 1^2 \cdot 1) / (1/k)^2 = k$

Keep f constant and use k (= 1.4) times more area:

$$\text{Power}_{\text{density}}\ \text{ratio} \sim ((1/k) \cdot 1^2 \cdot 1) / (1/k) = 1$$

Scale f and use k^2 (= 2) times more area:

$$\text{Power}_{\text{density}}\ \text{ratio} \sim ((1/k) \cdot 1^2 \cdot k) / 1 = 1$$

The end of Dennard Scaling

$$\text{Power}_{\text{density}}\ \text{ratio} \sim ((1/k) \cdot 1^2 \cdot k) / (1/k)^2 = k^2$$

Keep f constant: $\text{Power}_{\text{density}}\ \text{ratio} \sim ((1/k) \cdot 1^2 \cdot 1) / (1/k)^2 = k$

Keep f constant and use k (= 1.4) times more area:

Scale f

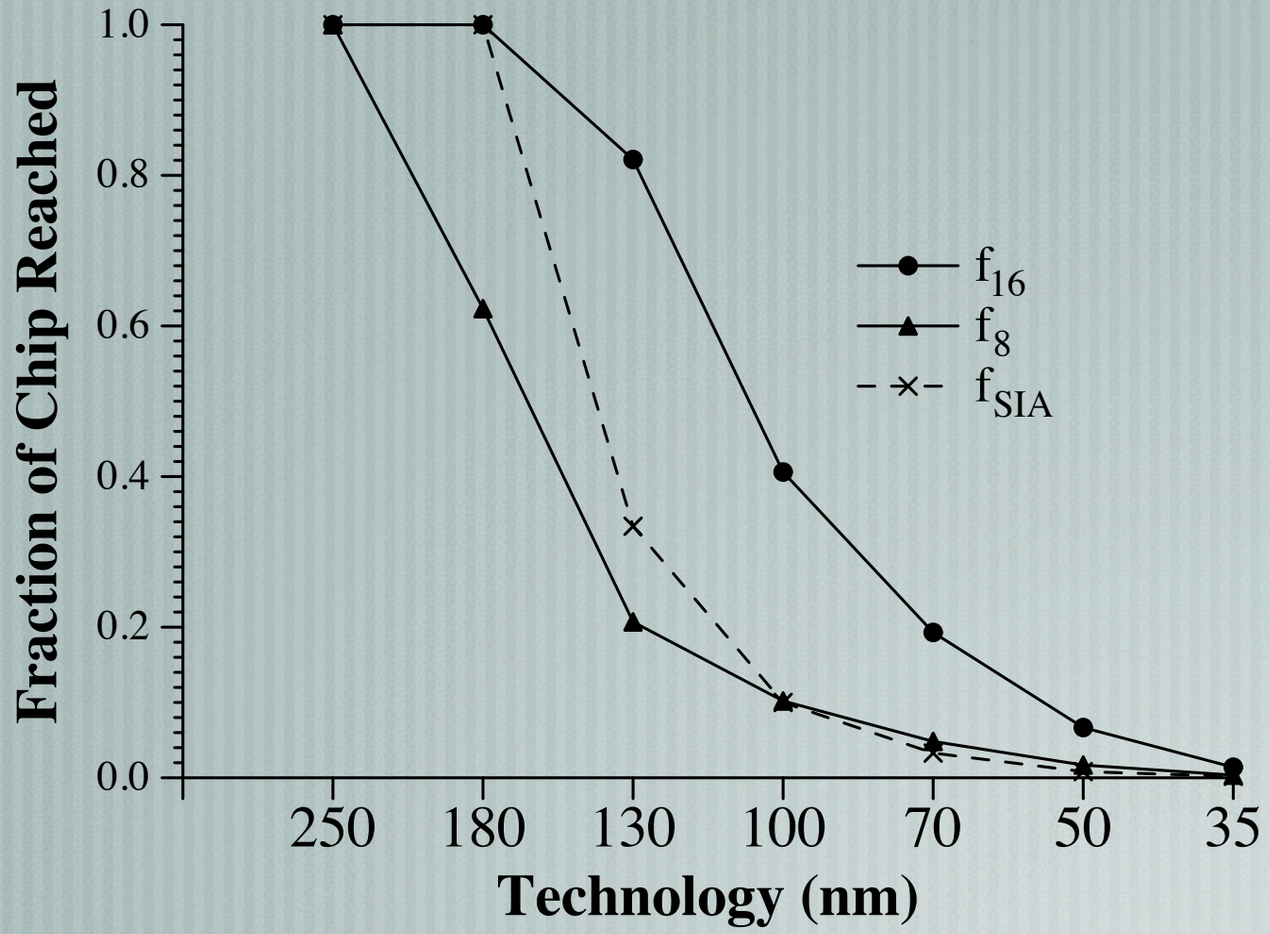
$\text{Power}_{\text{density}}\ \text{ratio} \sim ((1/k) \cdot 1^2 \cdot k) / 1 = 1$

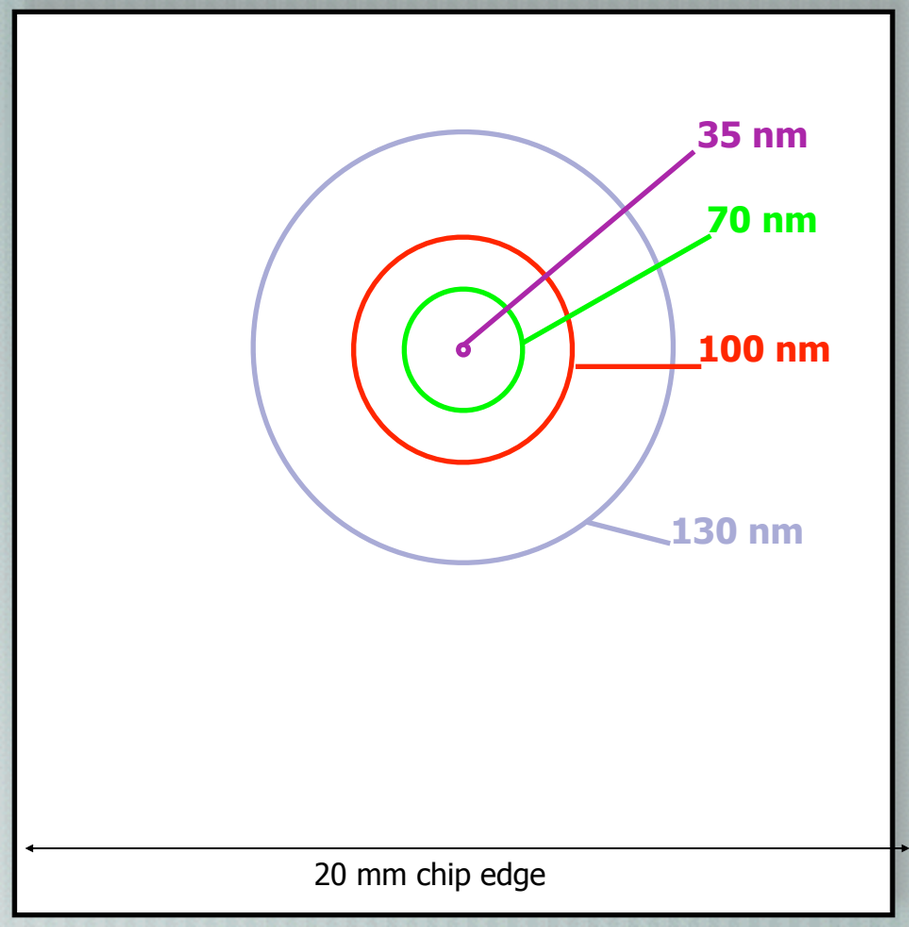
This extra, ‘unused’ area is called Dark Silicon

Some other walls we've hit...

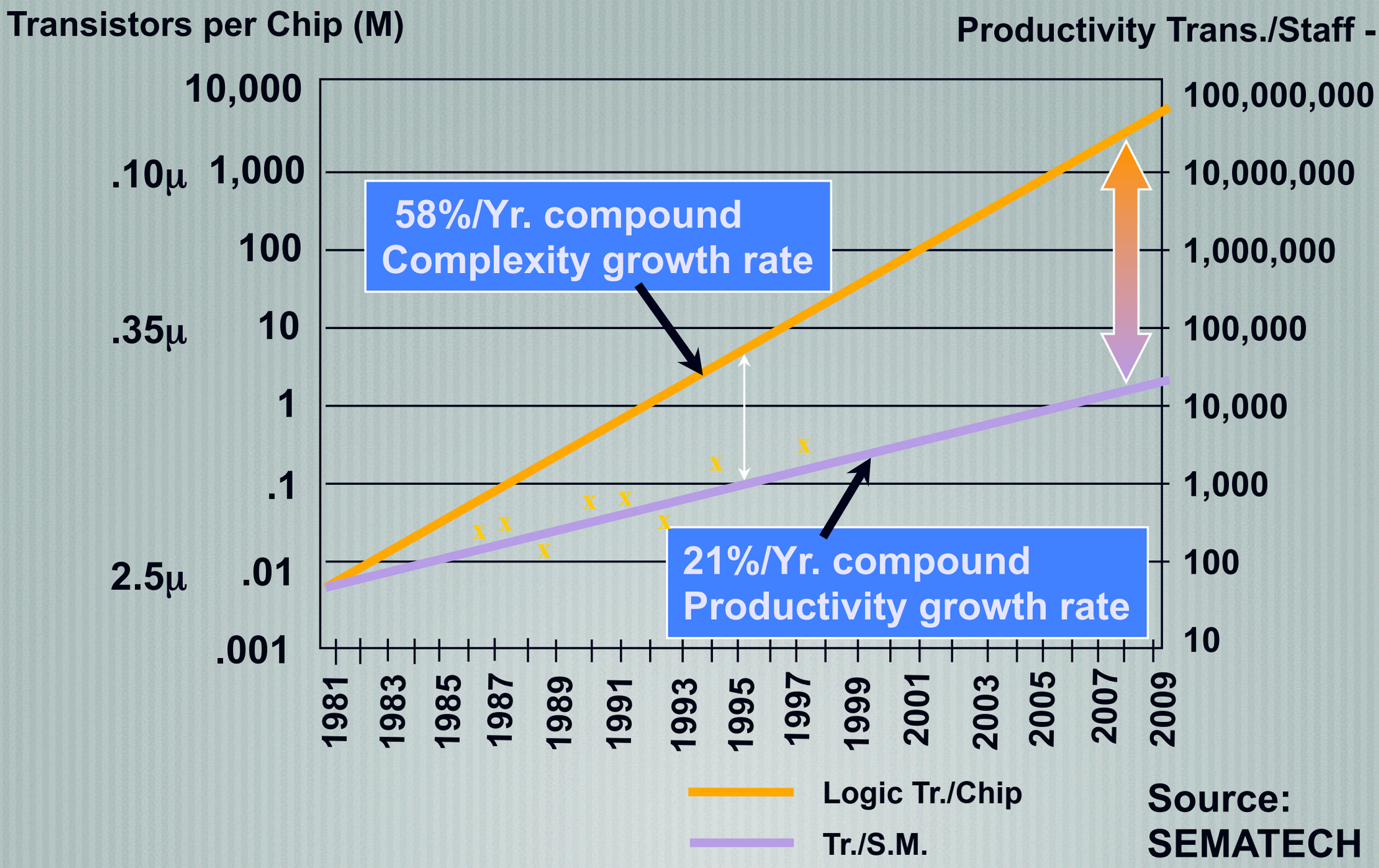
Some other walls we've hit...

Signal propagation



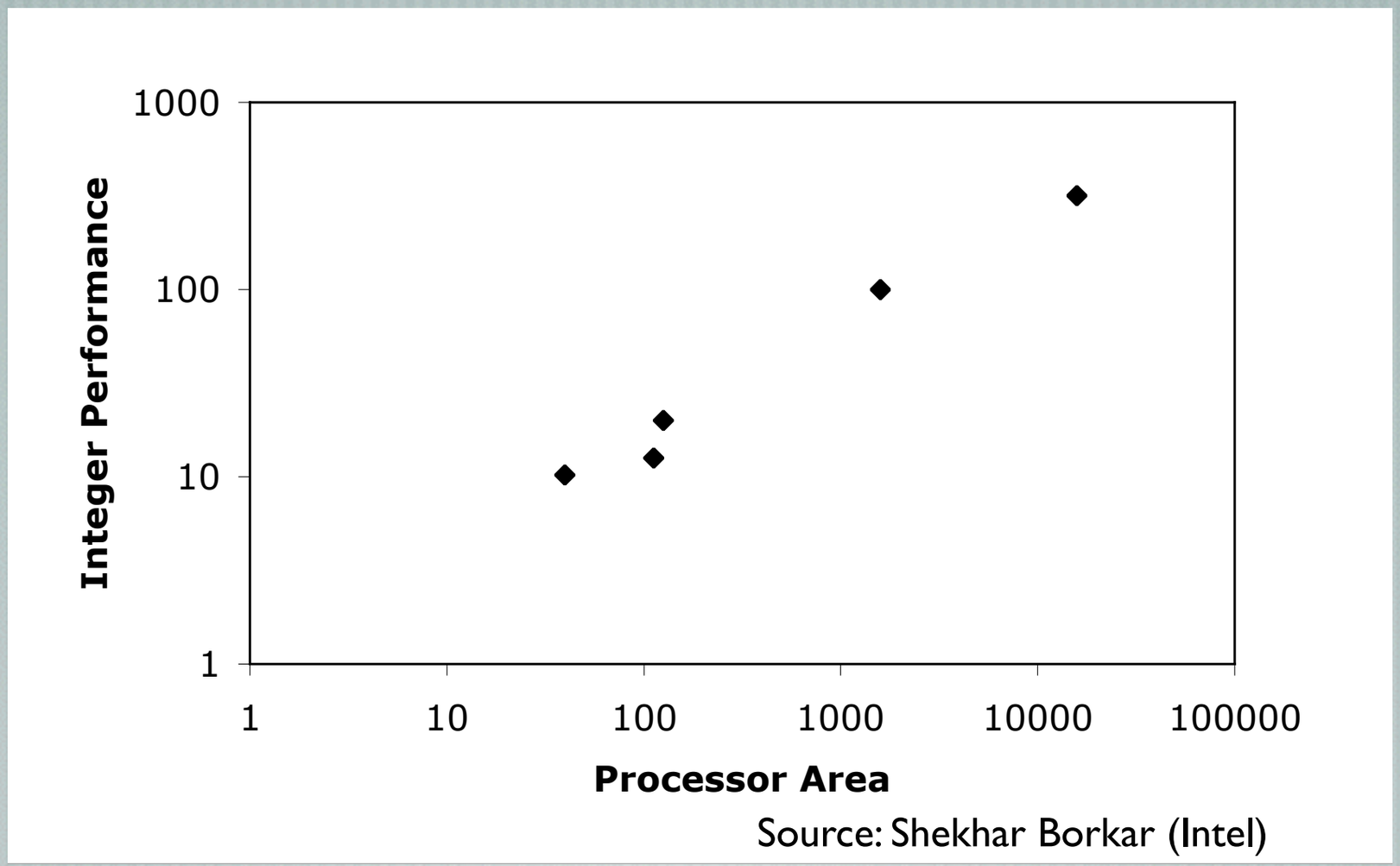


The design complexity wall



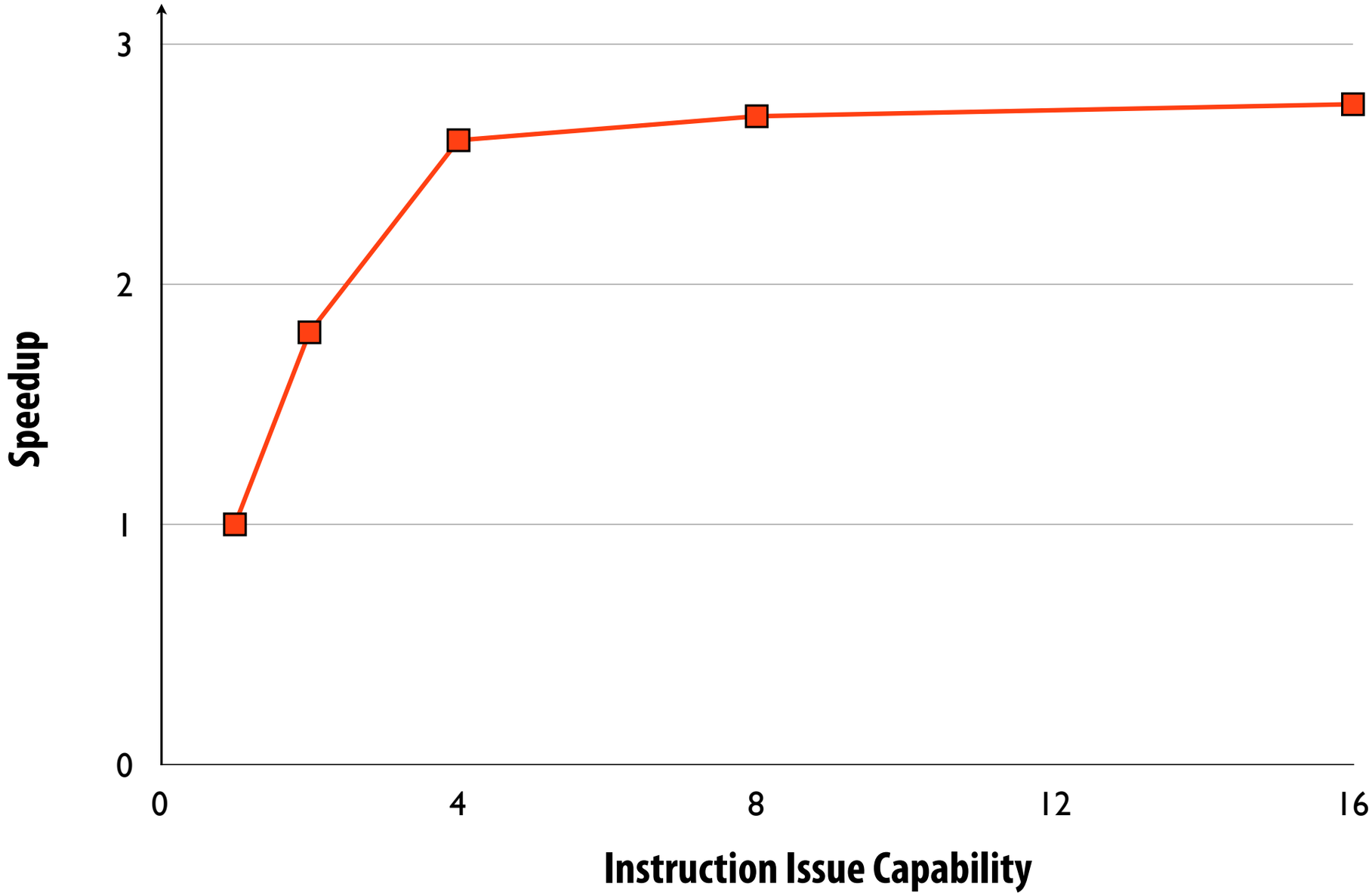
The design complexity wall

Pollack's law: single processor performance grows with the square root of area



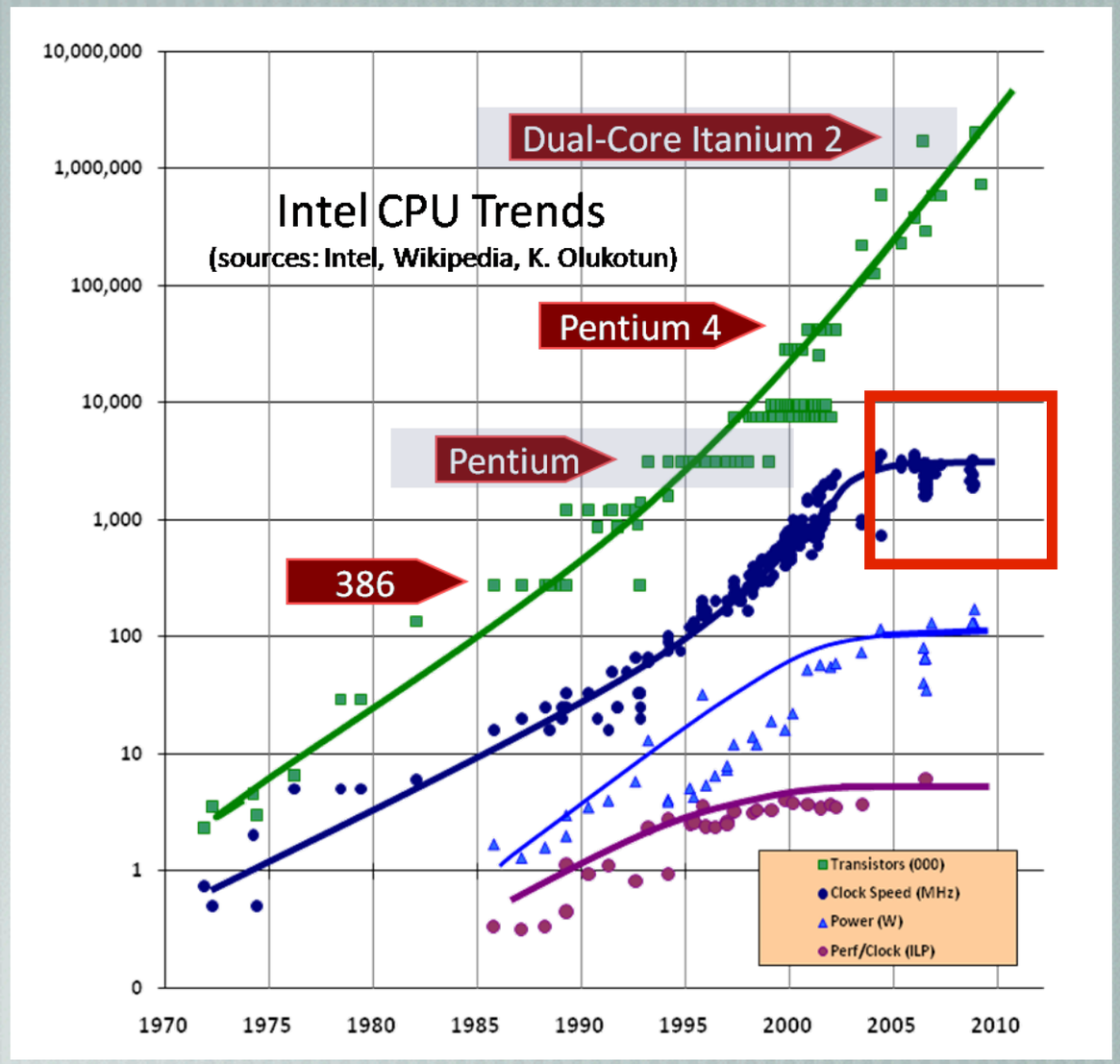
Source: Shekhar Borkar (Intel)

The ILP wall

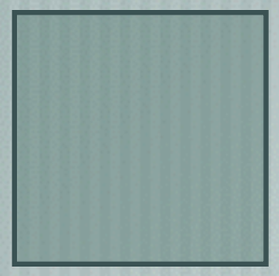


Single core performance scaling

- The improvement rate of single core performance has decreased (essentially to 0)
  - Frequency scaling limited by power (end of Dennard Scaling)
  - ILP scaling tapped out
  - Design complexity wall
- No more free lunch for software developers!



The shift to multi-core



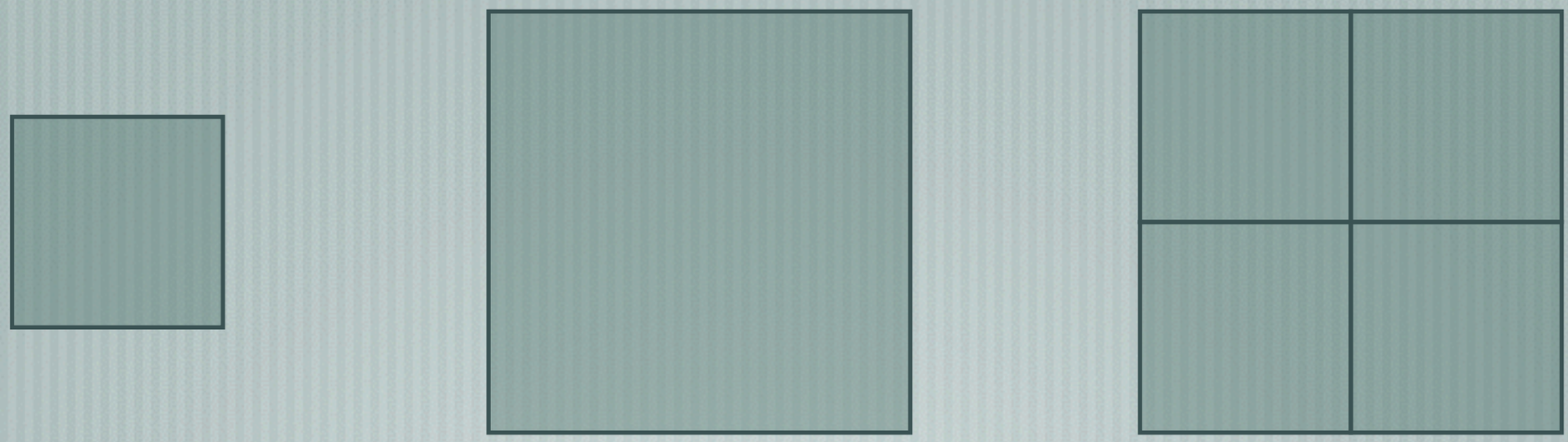
Performance 1
Power 1

The shift to multi-core

Performance 1
Power 1

Performance 2
Power 4

The shift to multi-core



Performance 1
Power 1

Performance 2
Power 4

Performance 4
Power 4

The shift to multi-core

Or: halving frequency

Performance 1
Power 1

Performance 2
Power 4

Performance 2
Power ≈1

— 1 core at f proportional to: $f \cdot V_1^2$, 4 cores at f/2 proportional to: $4 \cdot (f/2) \cdot V_2^2 = 2f \cdot V_2^2$

The shift to multi-core

Or: halving frequency

Performance 1
Power 1

Performance 2
Power 4

Performance 2
Power ≈1

1 core at f proportional to: $f \cdot V_1^2$, 4 cores at f/2 proportional to: $4 \cdot (f/2) \cdot V_2^2 = 2f \cdot V_2^2$

$V_1 > V_2$!!

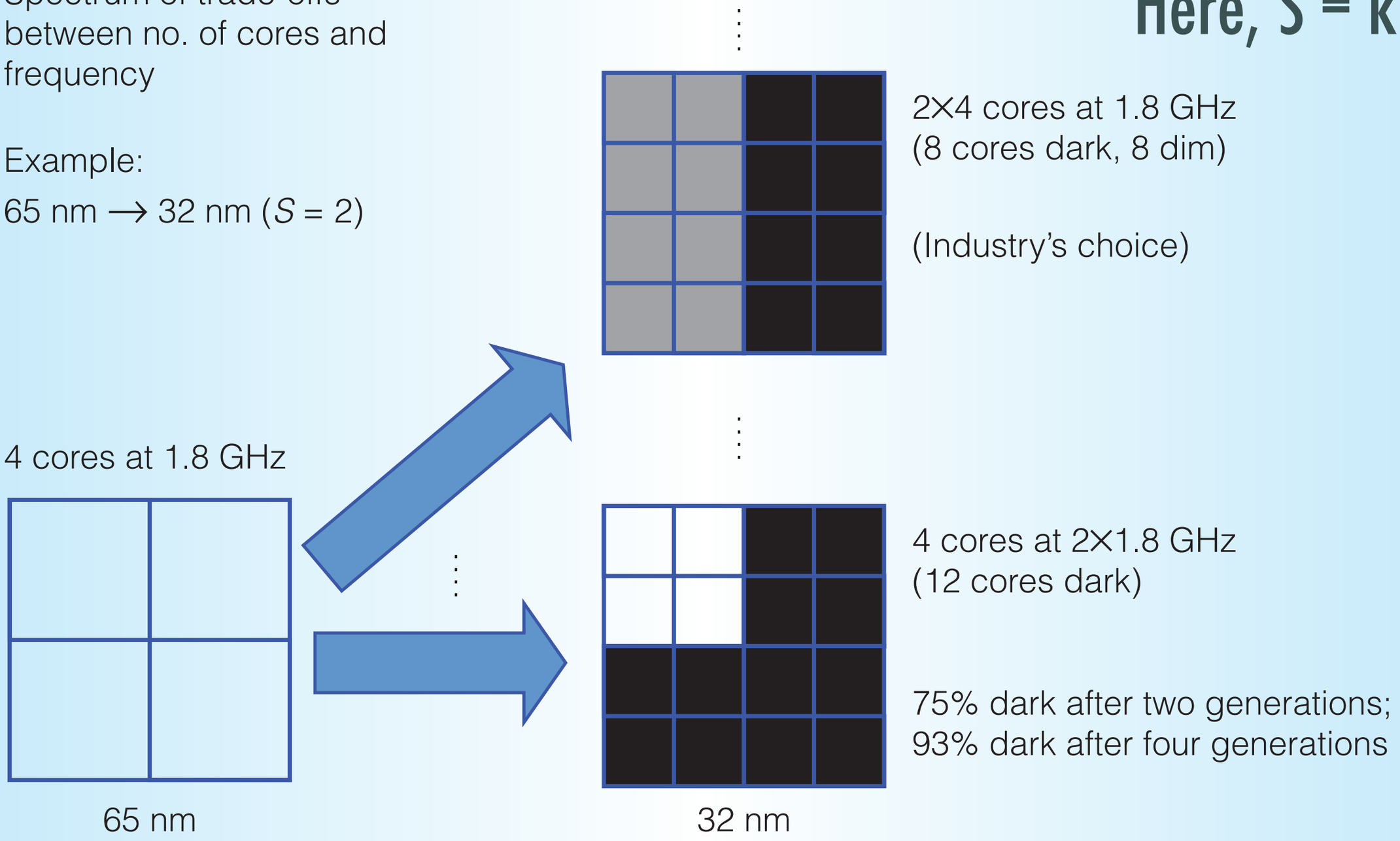
Multi-core and Dark Silicon

Spectrum of trade-offs between no. of cores and frequency

Example:

65 nm $\longrightarrow$ 32 nm (S = 2)

Here, S = k



Multi-core processors and hw multithreading

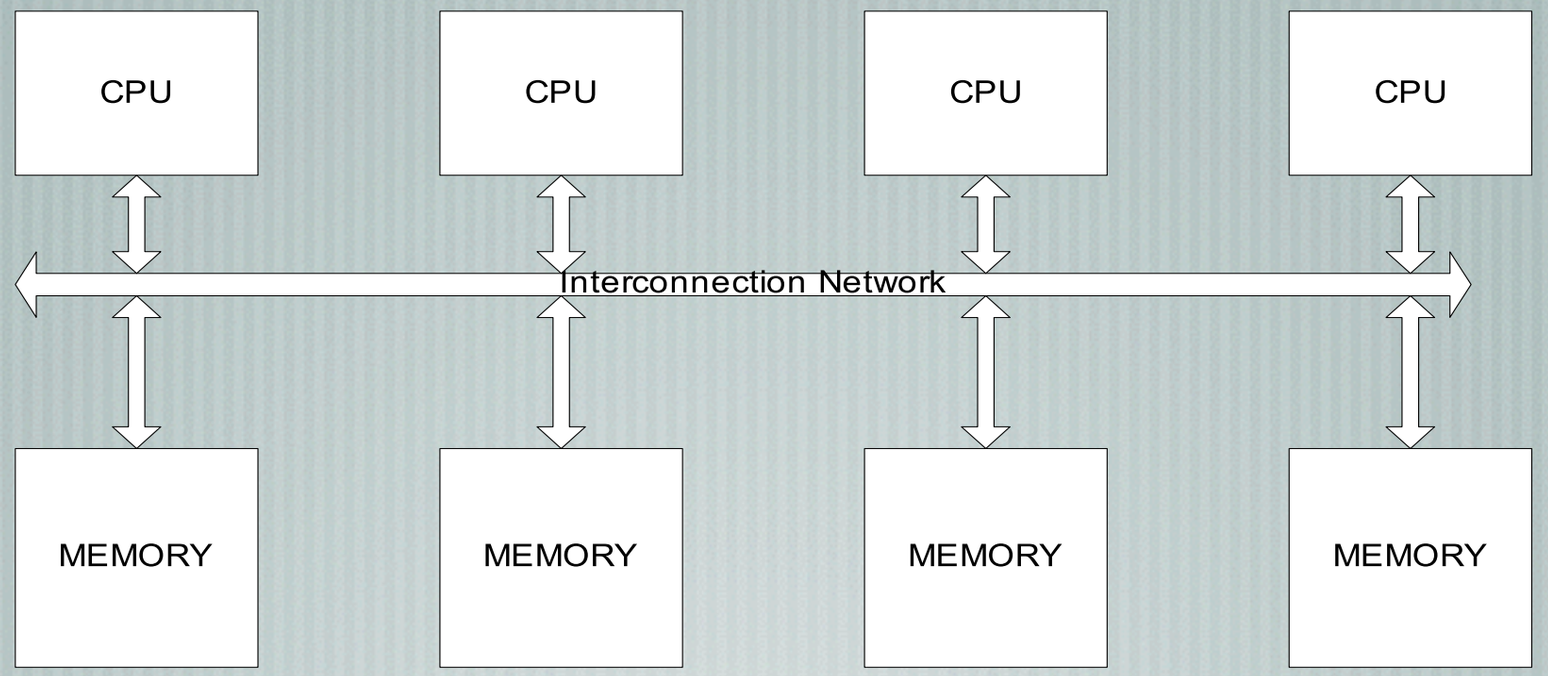
Explicit concurrency in hardware,
explicit concurrency in low-level software

Multi-cores main players

- Sun (now Oracle) was the forerunner in this field with its Niagara chips
- Intel have moved to multi-core without significantly changing their architecture
  - i7 is a 4-6 core with 14 stage speculative pipeline, Poulson IA-64 with 8 cores
  - Intel launched an experimental 48 core Single-Chip Cloud (SCC) chip
  - In 2012, Intel introduced the Xeon Phi: up to 61 cores on a chip
- IBM moved to multicore used in both games consoles & supercomputers
  e.g. Cell = 1 PPC + 8 vector cores

Multi-core organization

- Most multi-cores typically are symmetric, i.e. have an UMA organization



Uniform Memory Architecture

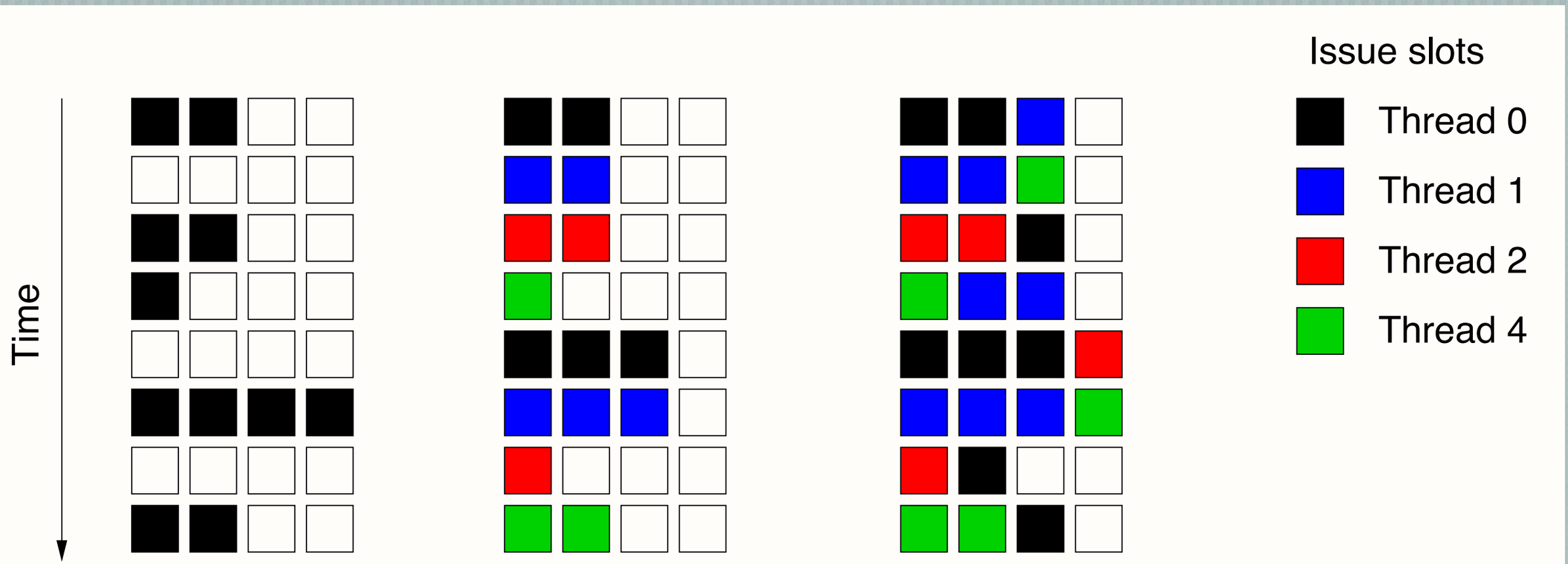
Multi-core often implies multi-threading per core

- Larger number of cores implies **larger average distance, hence latency**, between cores and cores/memory
- In turn, this implies larger mandatory off-core communication overheads for single threads
- To maximize utilization and throughput, cores should **fetch instructions from independent threads to tolerate latencies**
- This must be possible at the finest grain (individual loads and stores), hence the need for **hardware thread scheduling** in the fetch/issue stage

Hardware multi-threading

- Requires replication of hardware resources
  - Each thread uses its own PC and often its own register file
- **Interleaved** (or temporal) **multi-threading**
  - Each clock, core chooses from which thread one or more instructions are issued
- **Simultaneous multi-threading** (SMT)
  - Each clock, core chooses instructions from multiple threads (extension of superscalar design)

Two types of multi-threading



Multi-thread main players

- Sun/Oracle again with Niagara chips - 8 threads/core
- Intel recycled the SMT plans of 21464 as “HyperThreading”, found in P4 and again in Core i7 Nehalem, 2 threads/core
- Also found again in Itanium 2, 2 threads/core
- IBM POWER8: 8 threads/core
- Two main strategies for scheduling hw threads: **control flow** scheduling and **dataflow** scheduling

Control flow scheduling

- In control flow scheduling threads are identified for scheduling using **control flow triggers**
  - e.g. cache miss on a load
  - branches
- Threads are selected for execution from ready threads (e.g. round robin scheduling)
- On a trigger, e.g. branch or cache miss, the thread is suspended until resolution – e.g. Niagara, Itanium 2

Dataflow scheduling

- In dataflow scheduling threads are scheduled when **data to complete the instruction is available**
- Need a mechanism to suspend a thread on reading data (called "matching store," e.g. registers or memory)
- Dataflow i-structure does this: it includes synchronisation bits and holds either data or a handle to suspended thread(s)
- e.g. Transputer and Delencor HEP

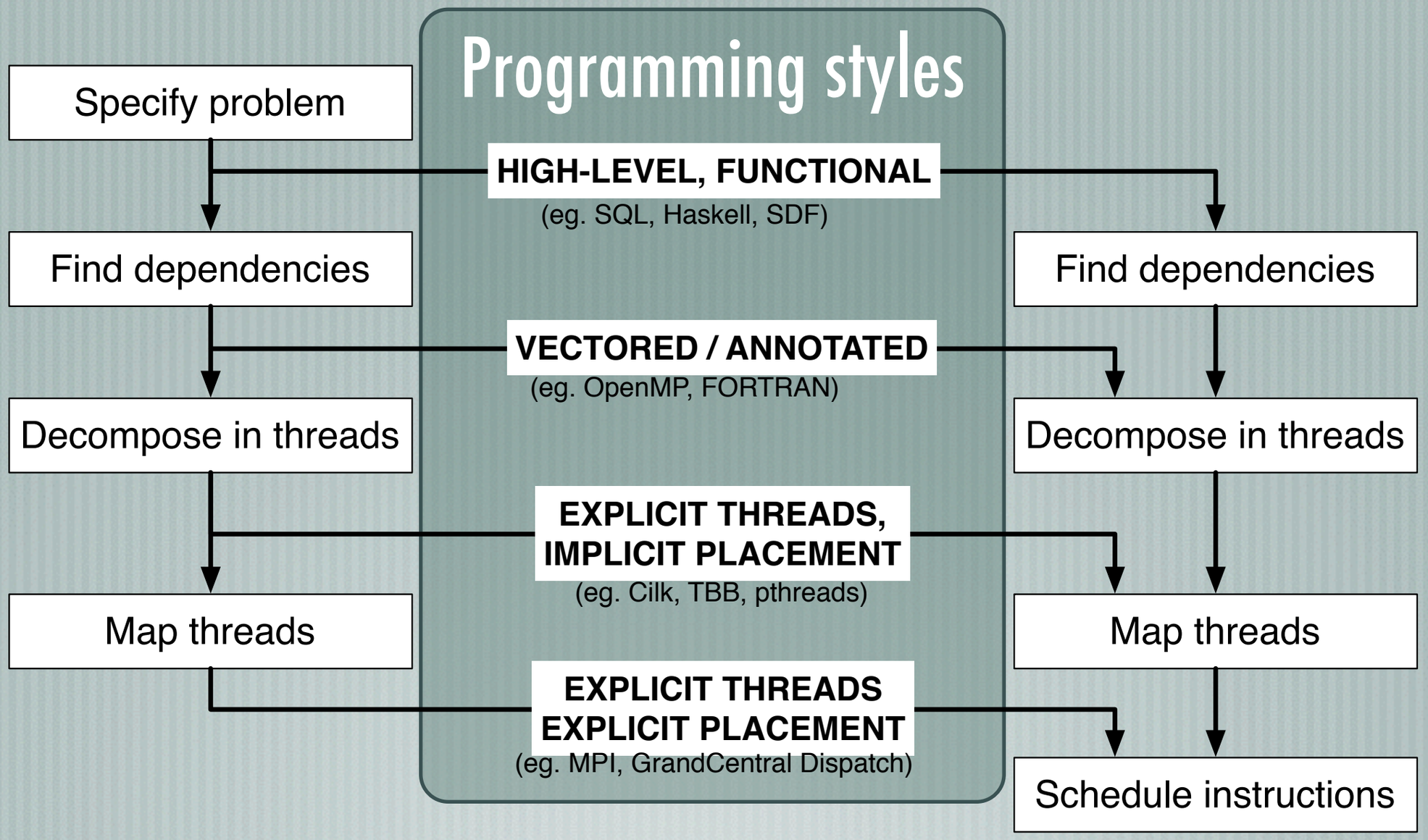
Programming issues

- Multiple cores and multiple threads per core **appear as different processors to software**, each with their own instruction stream (program counter sequence)
- Major departure from the "simple" Turing/Von Neumann model, convergence with parallel programming of HPC
- Explicit hardware concurrency requires **parallel machine models** to abstract the hardware, which in turn entail **concurrent programming models**

Programming models

- 3 phases to program an explicit concurrent chip:
  - **decompose** problem into concurrent sub-problems
  - **express** sub-problems as communicating threads
  - **map** threads onto chip components
- Different programming environments automate these tasks in different ways

Who's in charge of explicit concurrency?



Task of programmer

Task of software stack

(compiler + run-time system)

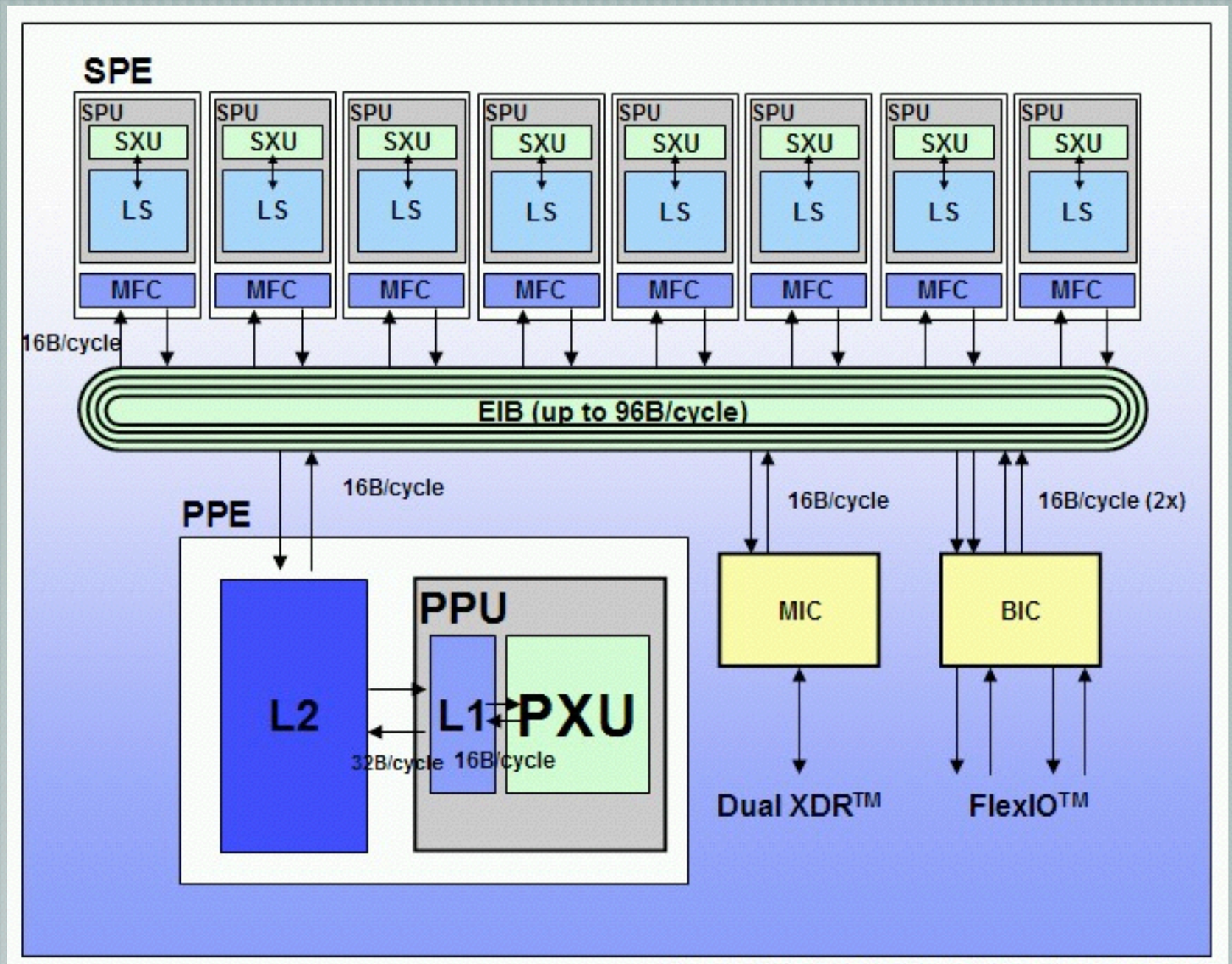
Programming issues (revisited)

- Parallel programming models at each level of abstraction come in two flavors: **implicit vs. explicit communication**
- **Implicit communication** based on shared memory or distributed software cache protocols, **which do not scale**
- **Explicit communication** leaves the program in charge of scalability, but is **more difficult to program**
- These issues are revisited in the Concurrent Programming course

Example

multi- and many-cores

IBM PowerXCell 8i

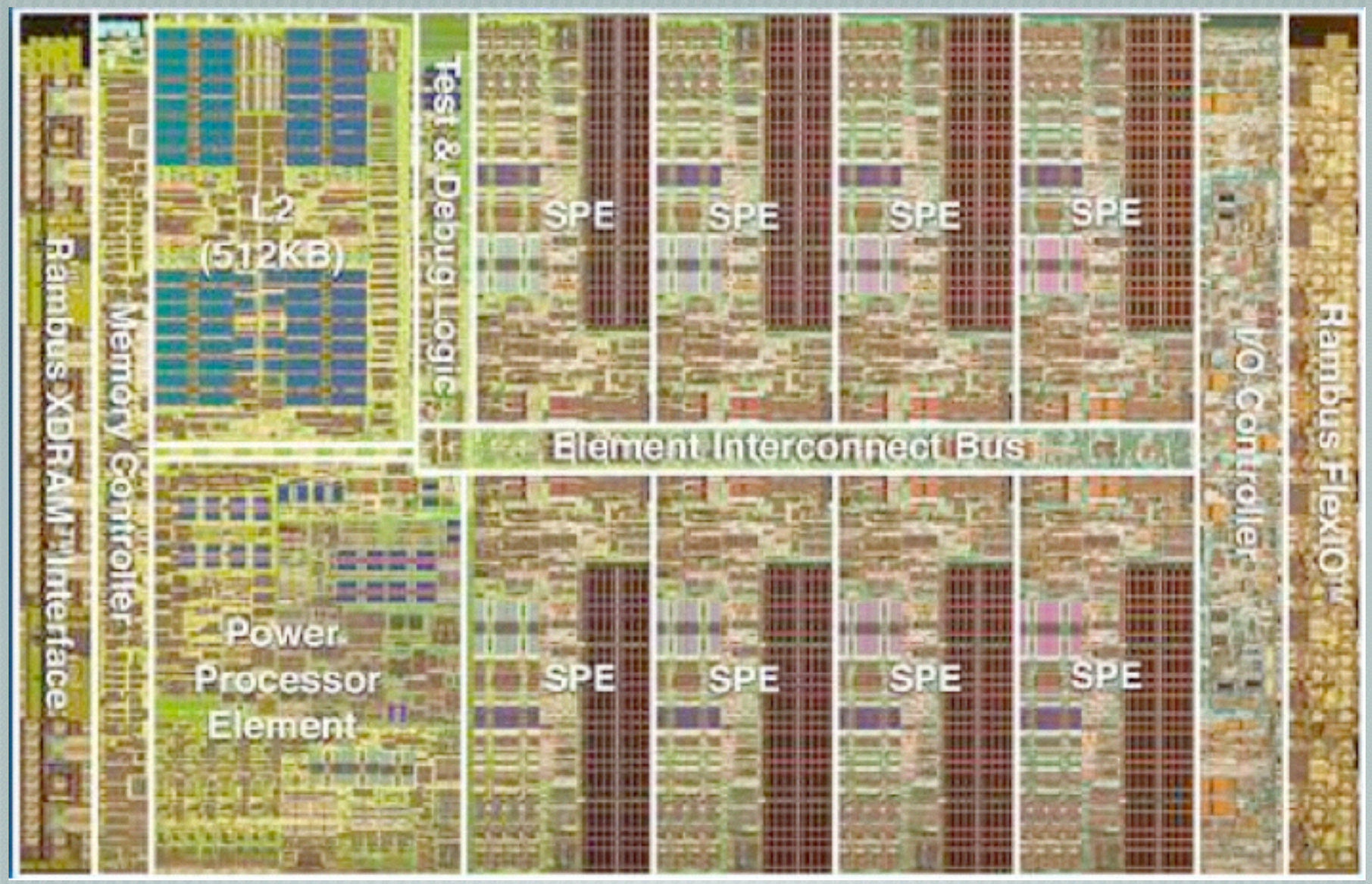


Source: M. Gschwind et al., Hot Chips-17, August 2005

IBM PowerXCell 8i

- 1 Power Processor Element (PPE)
  - Derived from IBM Power5 architecture
- 8 Synergistic Processor Elements (SPEs)– SIMD processors
  - 128 bit vector unit supporting variable precision integer & double precision FP
- 1 Element Interconnect Bus (EIB) - a fast multiple ring network
- Direct Memory Access controller
- DDR-2 memory interface (originally Rambus XDR)
- 65nm technology 3.2 GHz frequency

IBM PowerXCell 8i



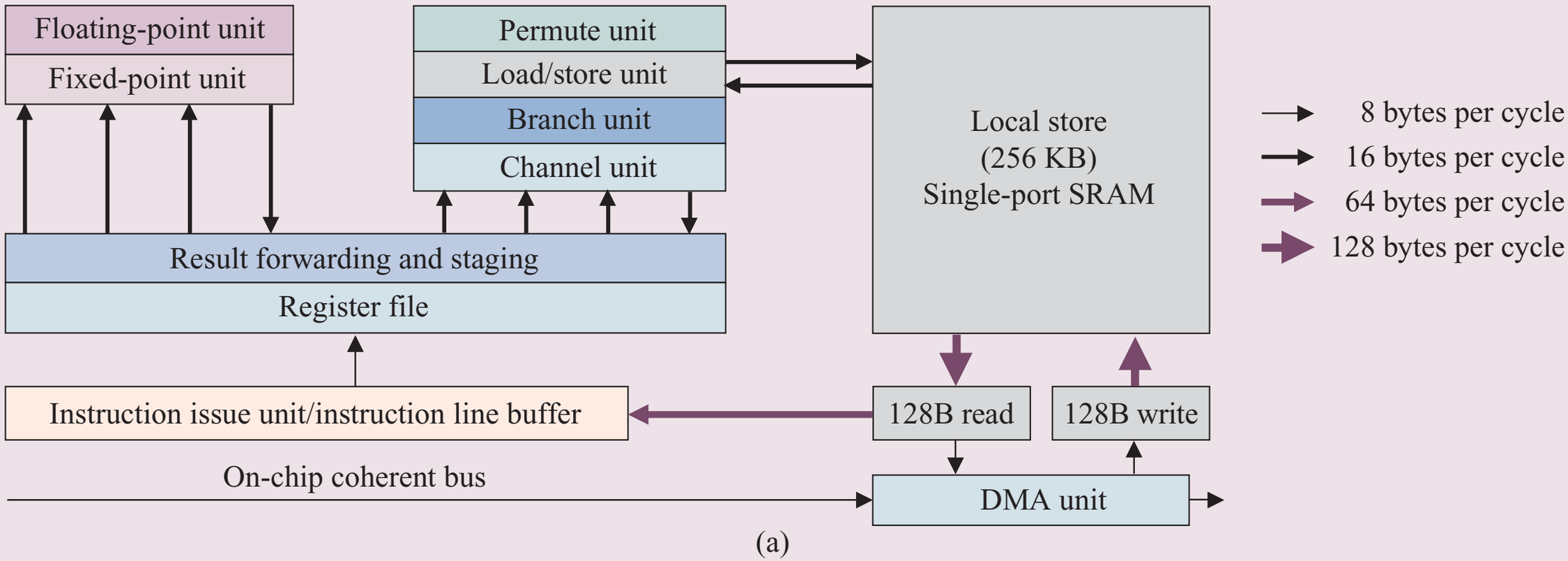
Cell's PPE

- 64 bit RISC processor, PowerPC ISA
- 32/32 KByte L1 I- and D-caches
- 512KB L2 cache
- 64GB/s load-store bandwidth
- In-order execution, 2-way issue - 2 hardware threads
- Optionally equipped with AltiVec SIMD extensions

Cell's SPE

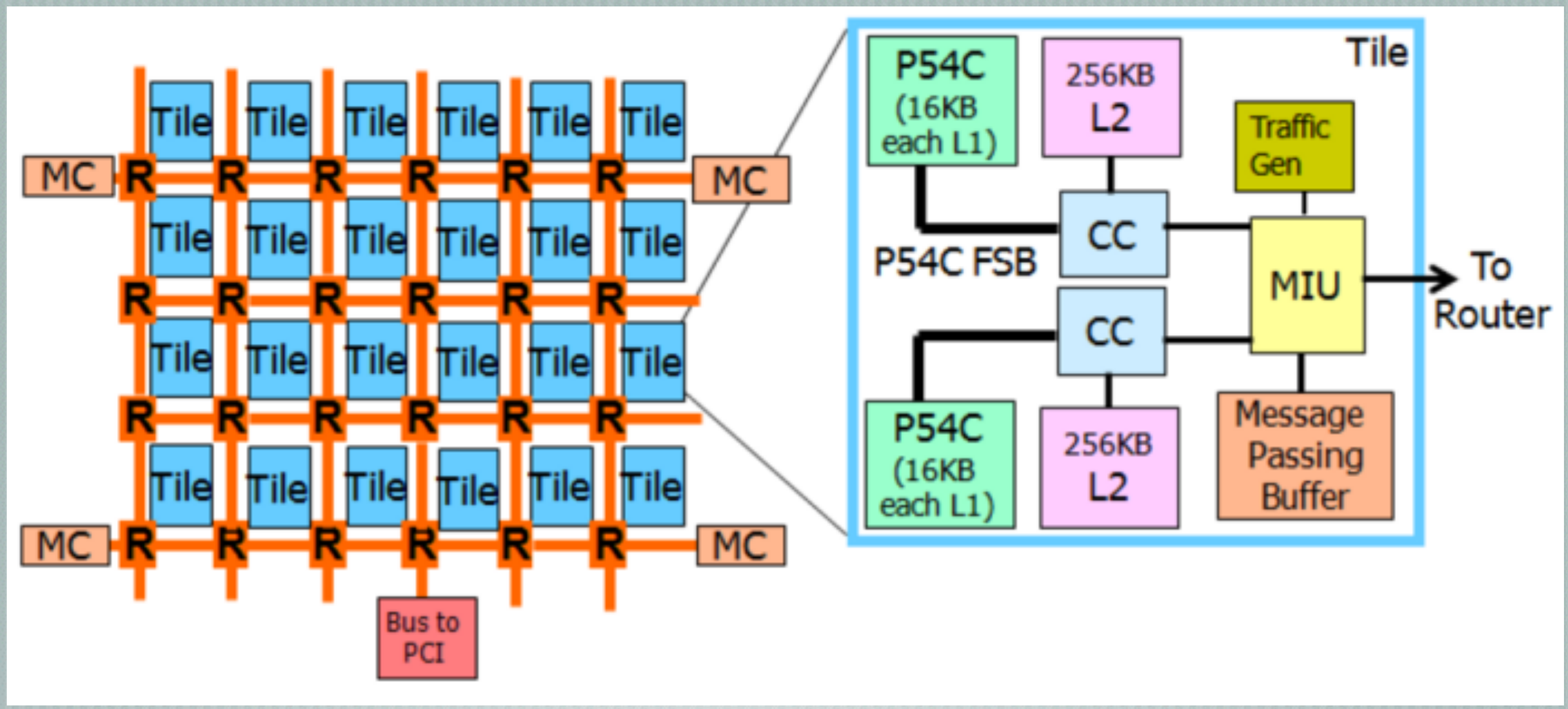
- Independent processors each runs an application thread
  - has its own 256KB private local store
  - has DMA access to coherent shared memory of PPE
- It is a SIMD vector processor with an Altivec-like ISA
  - 128 by 128 bit registers used as 16 x 8 bit, 8 x 16 bit, 4 x 32bit, 2 x 64bit
  - 4 single precision FP units (latest version supports 2 x DP)
  - 4 integer units
  - Dual issue - 8 x 32 bit operations per cycle
  - max 25.6 GFLOP/s with single precision FP

Cell's SPE


Floating-point unit
Fixed-point unit
Permute unit
Load/store unit
Branch unit
Channel unit
Local store
(256 KB)
Single-port SRAM
8 bytes per cycle
16 bytes per cycle
64 bytes per cycle
128 bytes per cycle
Result forwarding and staging
Register file
Instruction issue unit/instruction line buffer
128B read
128B write
On-chip coherent bus
DMA unit

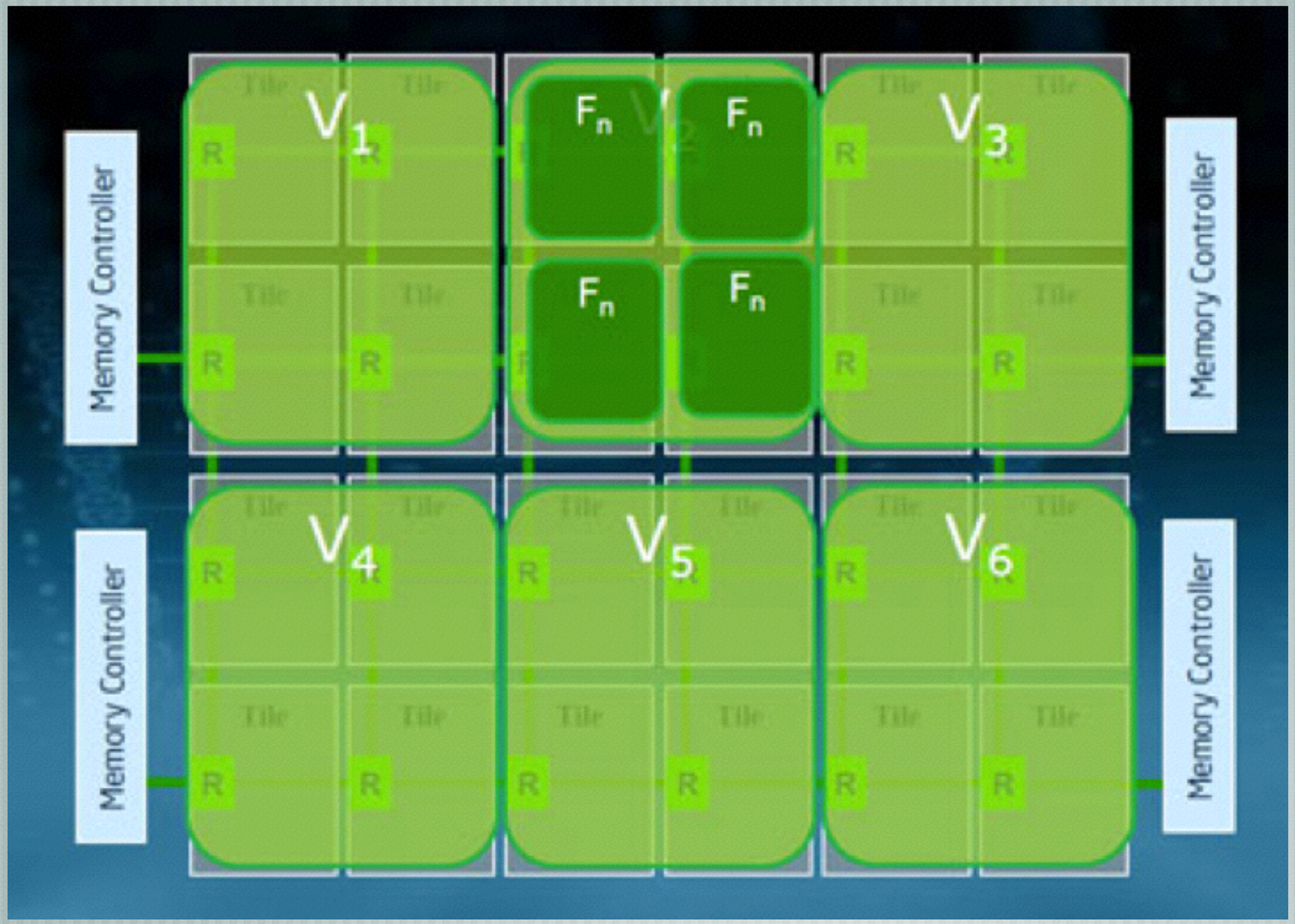
(a)

Intel Single-Chip Cloud



48 P54C cores (Pentium I), mesh interconnect, no cache coherency in hardware

Intel Single-Chip Cloud

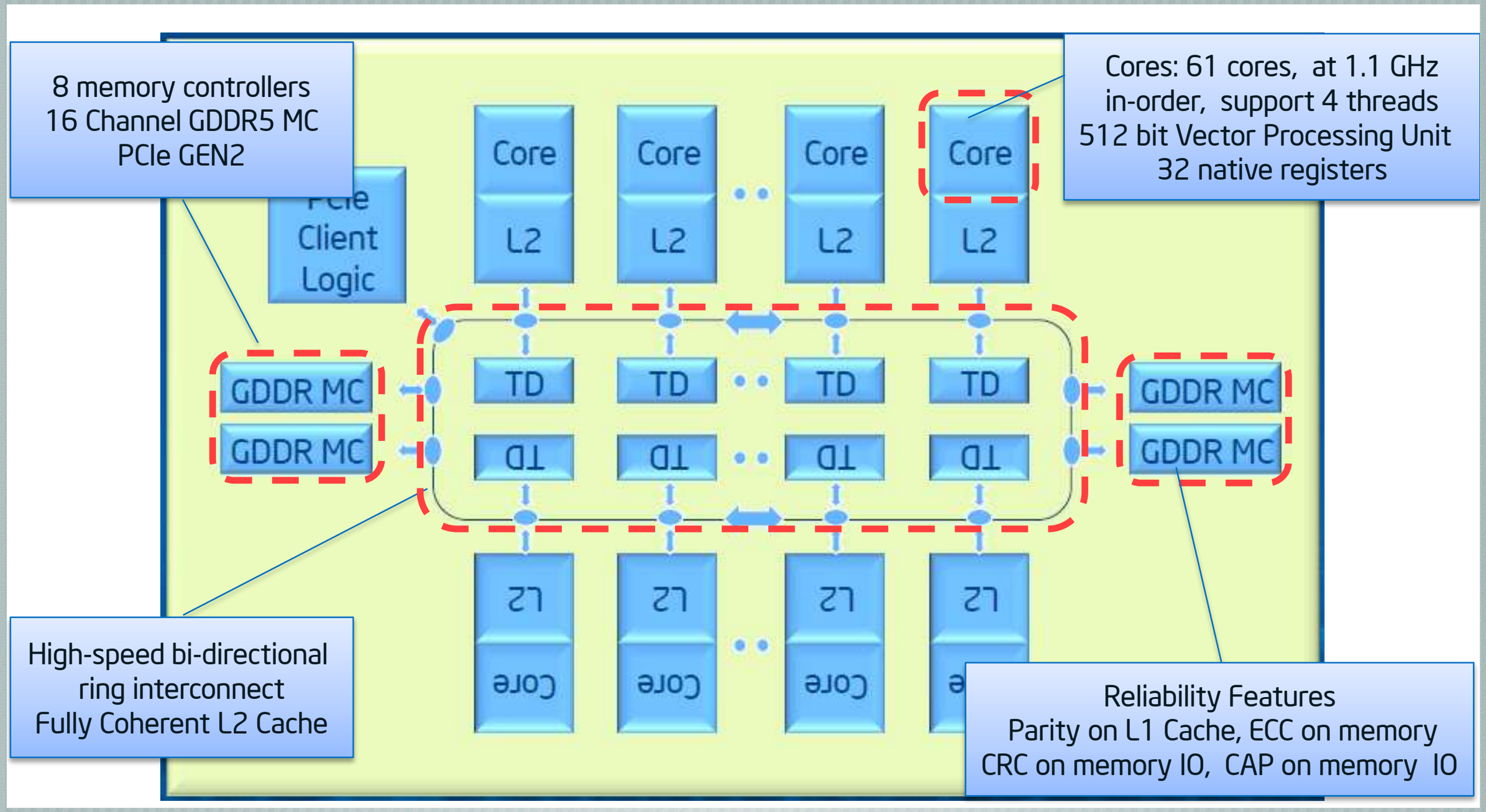


8 voltage islands

28 frequency islands

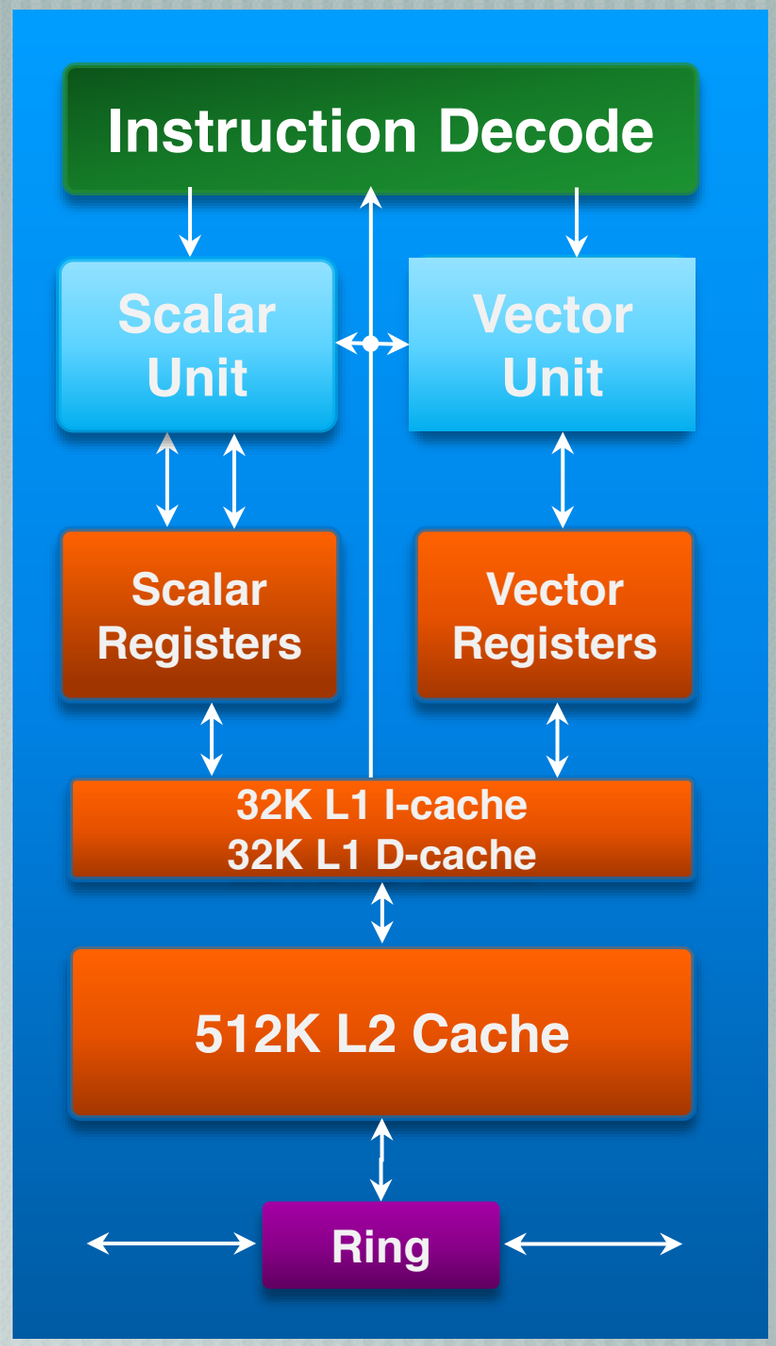
Independent V/F for I/O and memory

Intel Xeon Phi



Intel Xeon Phi core

- Two pipelines
  - Scalar unit based on Pentium
  - Dual issue with scalar instructions
- SIMD Vector processing unit
- 4 HW threads per core
  - Cannot issue instructions back-to-back from same thread
  - Need minimum of 2 threads to keep pipeline filled

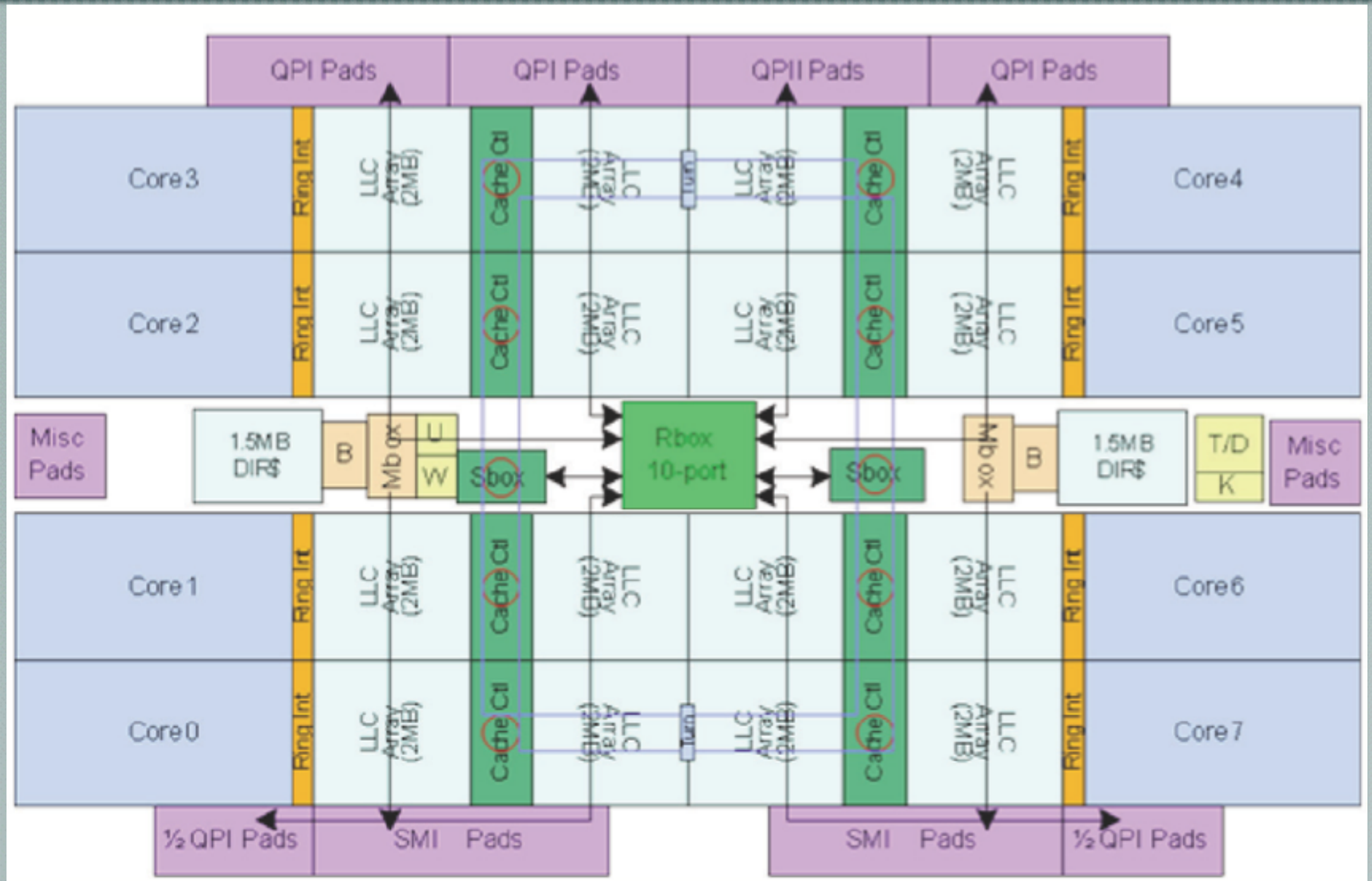


IA-64 Montecito... Poulson

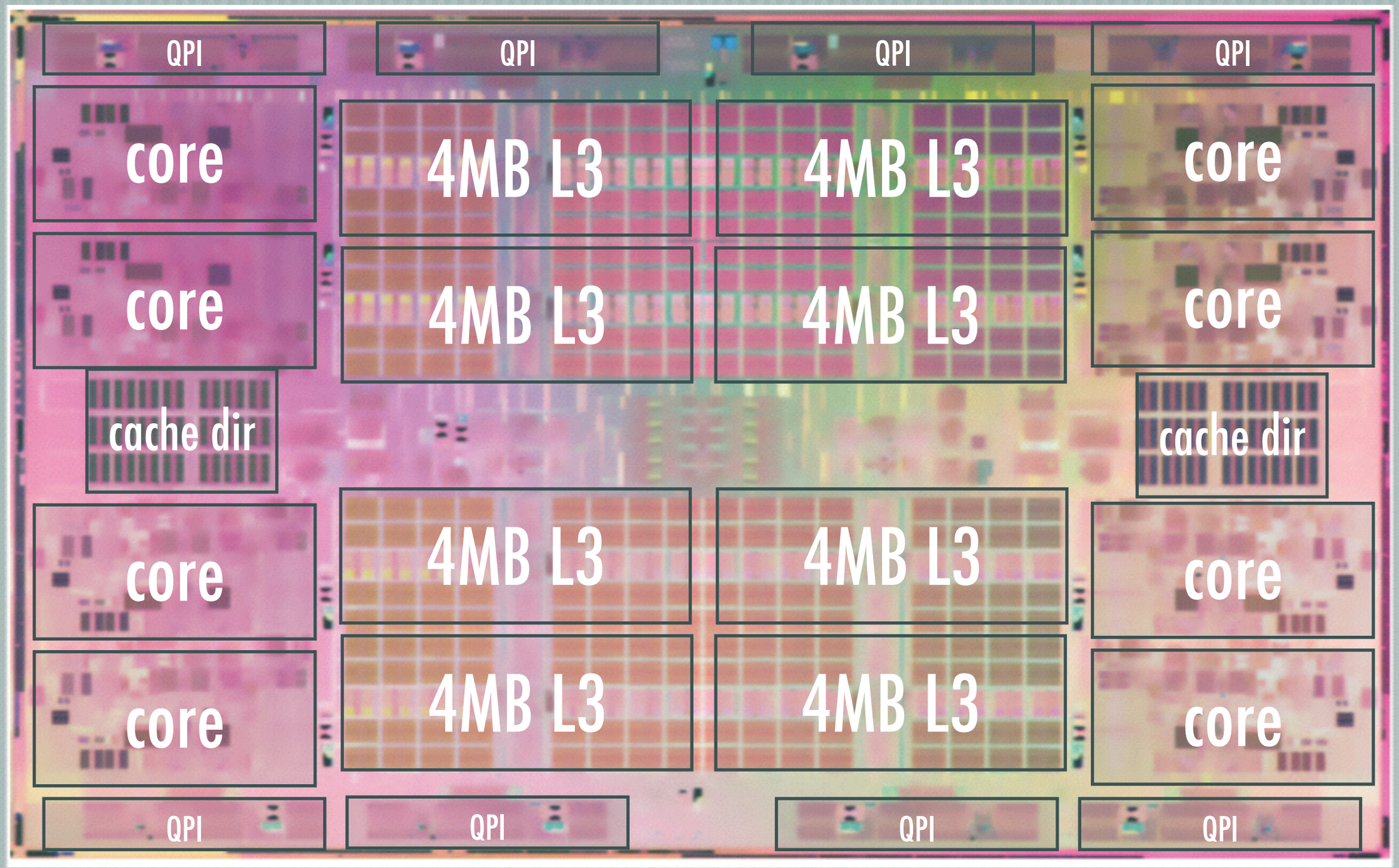
Both multi-core McKinley (Itanium 2)

Montecito (2006)	**Poulson (2012)**
1.72 billion transistors	3.1 billion transistors
90nm - 596mm^2	32nm - 544mm^2
75-104W	15-170W
2 cores, 1.4-1.6GHz core clock	8 cores, 1.6-1.85GHz core clock
6-way issue per core, 12-way total	6-way issue per core, 48-way total
6-24MB L3 on chip (2 x 3-12MB)	32MB L3 on chip (8 x 4MB)
16/16 KB L1, 1MB/256KB L2	16/16 KB L1, 512KB / 256KB L2
21GB/s FSB bandwidth	700GB/s system bandwidth (est.)

IA-64 Poulson



IA-64 Poulson



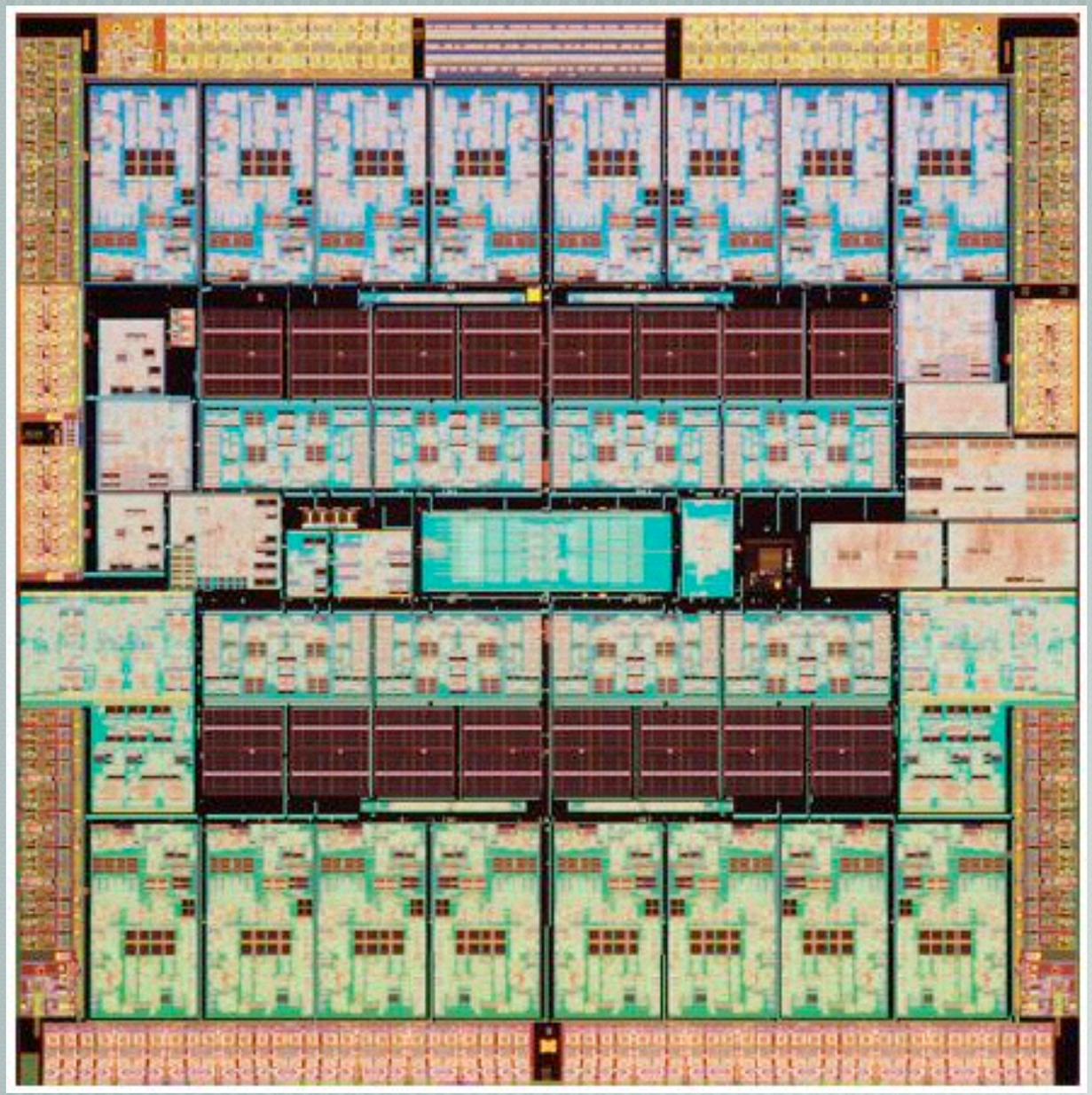
SPARC Niagara

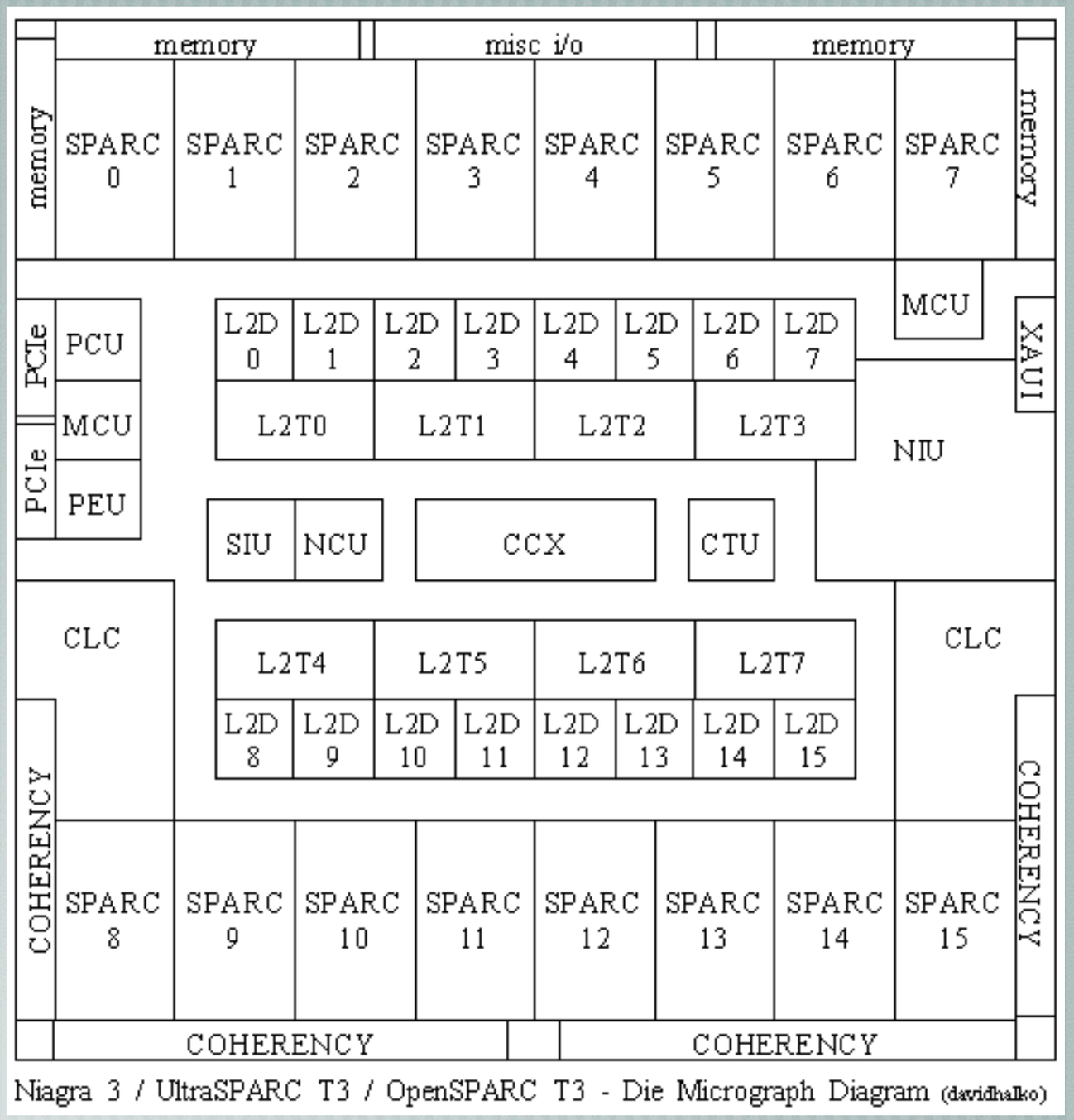
T1/2/3/4/5

Niagara in a nutshell

- Departure from the beaten road of sequential performance: focus on multi-cores and multi-threading
- Niagara T1 (2005): 8 cores, 4 threads/core, 1-1.4GHz
- Niagara T2 (2007): 8 cores, 8 threads/core, 1.2-1.6GHz
- Niagara T3 (2009): 16 cores, 8 threads/core, 1.67GHz
- 2 single-issue in-order pipelines / core, 4 threads per pipeline

Niagara T3 floorplan

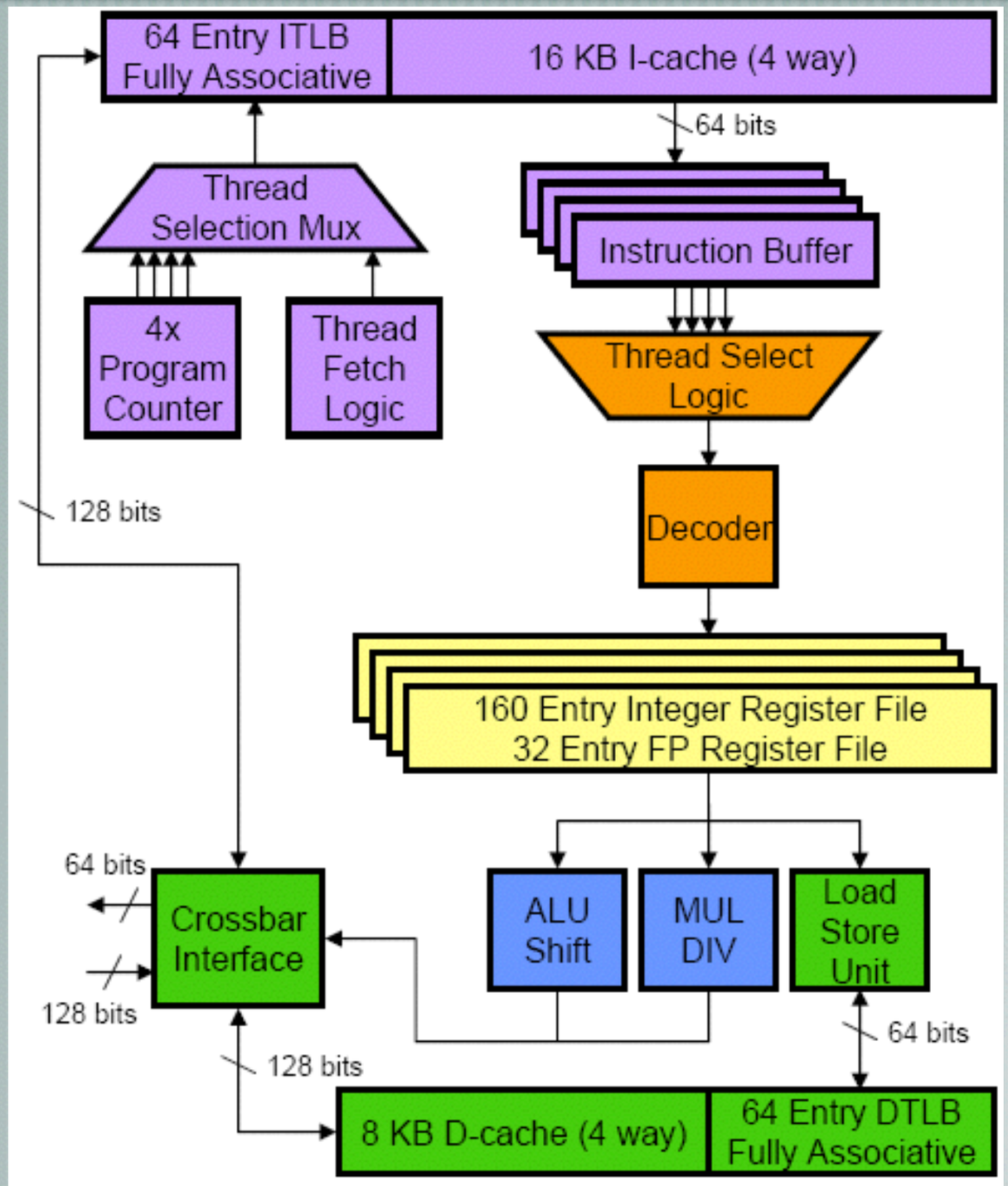




Niagara landmarks

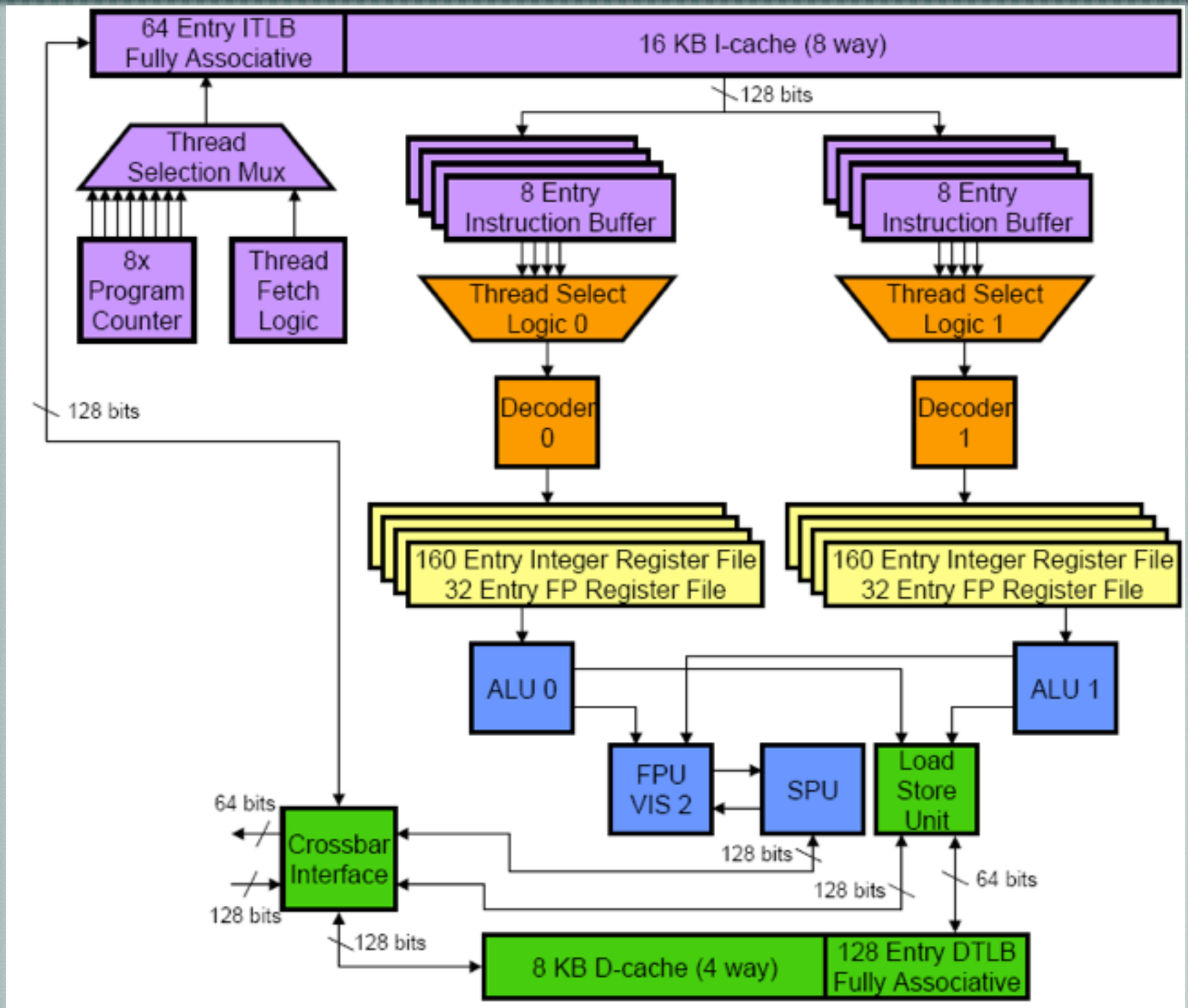
- Single shared L2 cache, cross-bar for full coherency
  Scalability problems with larger number of cores / larger cache
  (see next chapter)
- Explicit concurrency:
  - each core can issue 2 instructions per cycle to 2 pipelines (from T2 onward) which share IF, load/store and FPU
  - with 16 cores, ILP = 32 instructions per cycle
  - virtual concurrency 4 threads per pipeline
- this allows for flexibility in instruction scheduling, select stage issues from available threads

Niagara T1 pipeline



Source: RealWorldTech

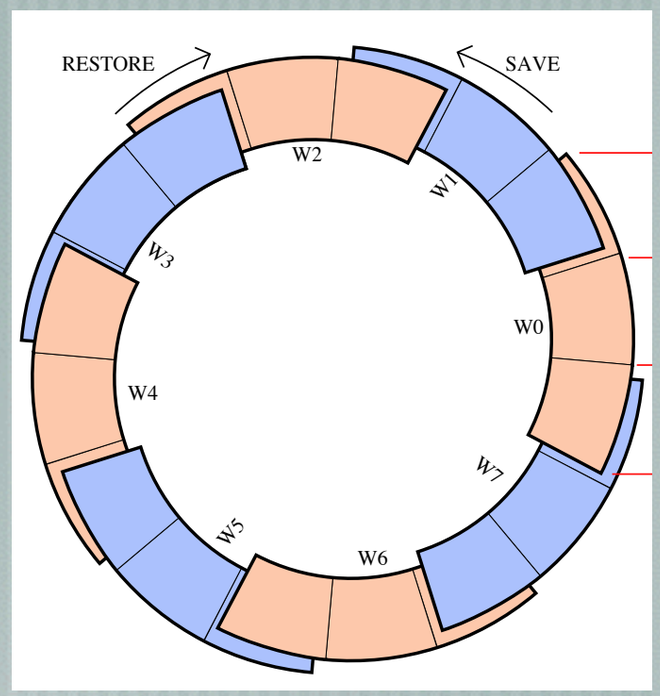
Niagara T2 pipeline



Source: RealWorldTech

Niagara register files

- SPARC ISA supports register windows
  - for n overlapping windows register file comprises 16 + n*16 registers, where each window has 8 global, 8 local, 8 input and 8 output registers
    - output of one window is the input to the next



- Niagara provides 4 independent thread contexts per pipeline
  - 8 per core in two groups (strands 0..3 and 4..7), each file has 8 register windows – 160 registers per thread giving a total of 1152 registers per core
- Each register file has **5 ports**, uses "3D addressing" to exploit the fact that only one window per thread is active at a time - **this design is scalable**

Niagara thread scheduling

- How are SPARC threads defined?
  - threads are defined by OS call setting up a thread, its stack and its PC using a system mode instr.
- How scheduled?
  - Active threads are scheduled on an LRU basis for fairness
    threads become inactive on branch instructions and when stalled waiting for memory
  - Thread scheduling assumes an L1 cache hit
- Thread management costs:
  - creation – performed in software, so relatively high cost but can be reduced using thread pooling
  - scheduling – zero cycle thread switching: new threads are selected for execution on every cycle
  - synchronisation – depends on where test and set address resides in memory hierarchy

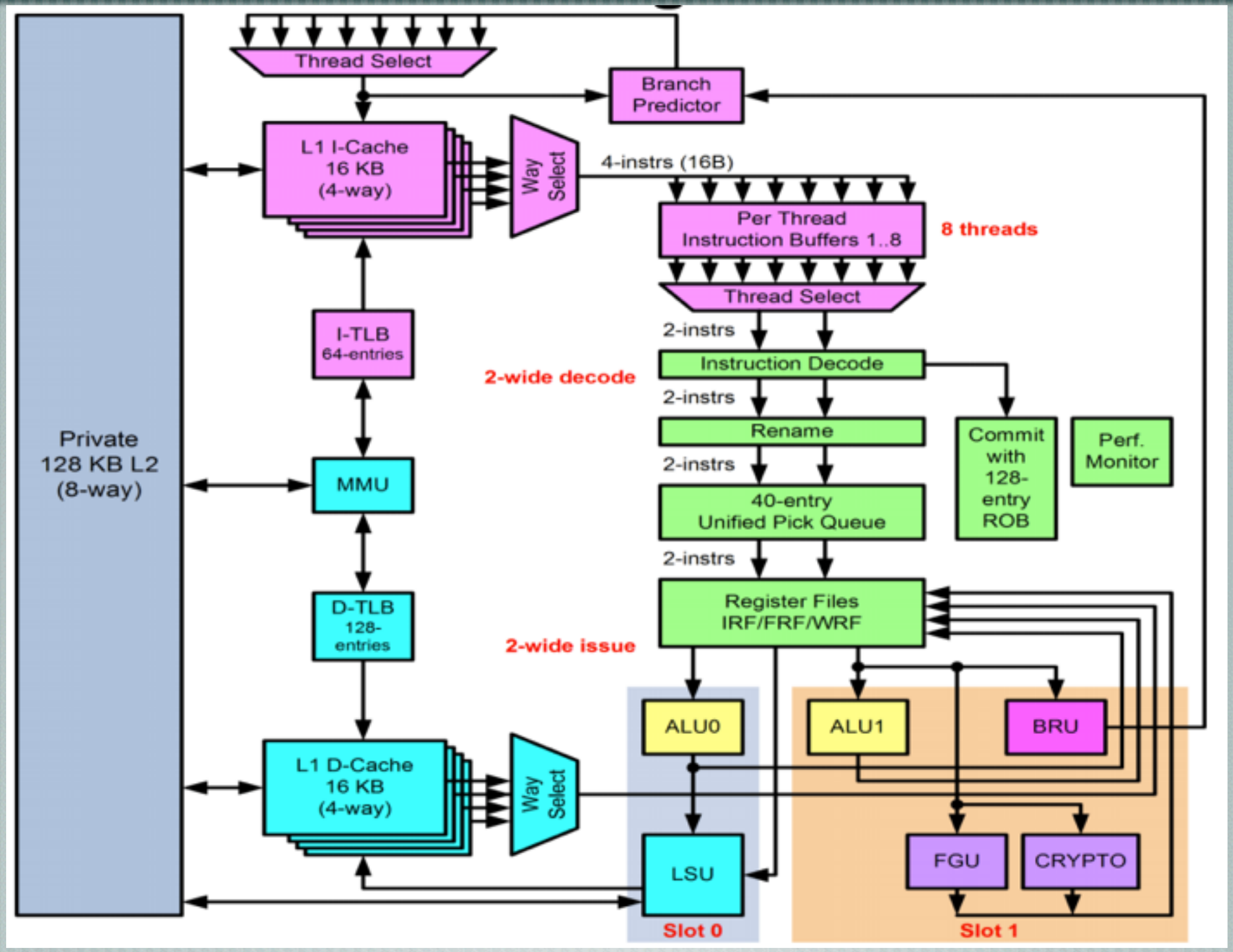
Niagara memory (T3)

- L1 shared between 2 pipelines
- L2 shared between all cores
  - It has 16 banks with two X-bar switches between groups of 8 cores
  - Switch is approximately 5% of core area
  - Reads at 180 Gbytes/s, writes at 90 Gbytes/s
- L2 cache 6 MByte, 64 Byte lines - 16-way set associative
- Memory interfaces 4 x DDR 3, fully buffered
- **Memory system designed for throughput**

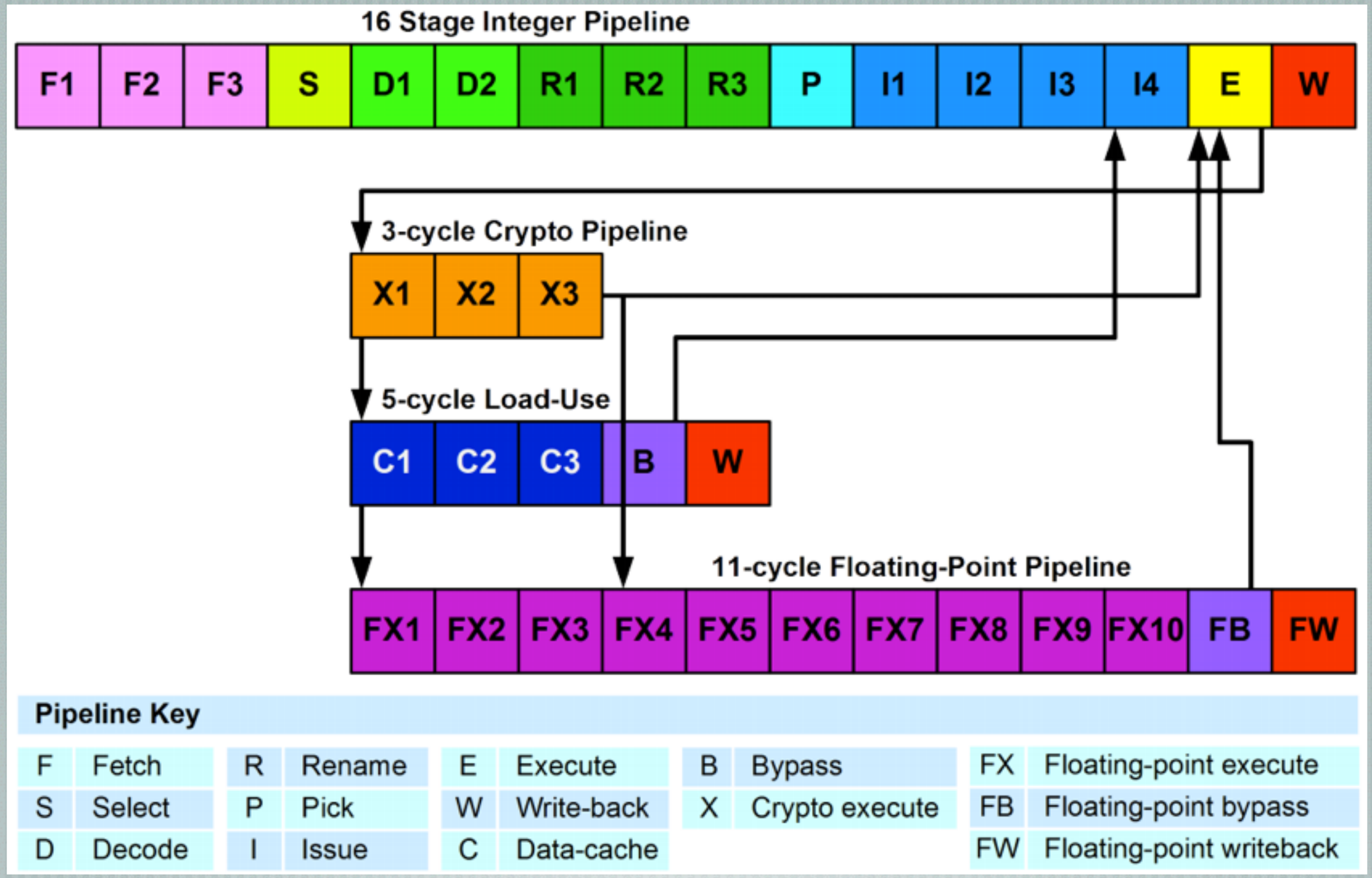
Niagara T4 - yet different

- Departure from the T1/T2/T3: focuses again on sequential performance
- Introduces OoO issue and branch prediction
- The extra logic per core is compensated by fewer cores (8)
  - T5 brings the number of cores back to 16
- Introduces a "Work Register File" for storage after register renaming

Niagara T4 pipeline



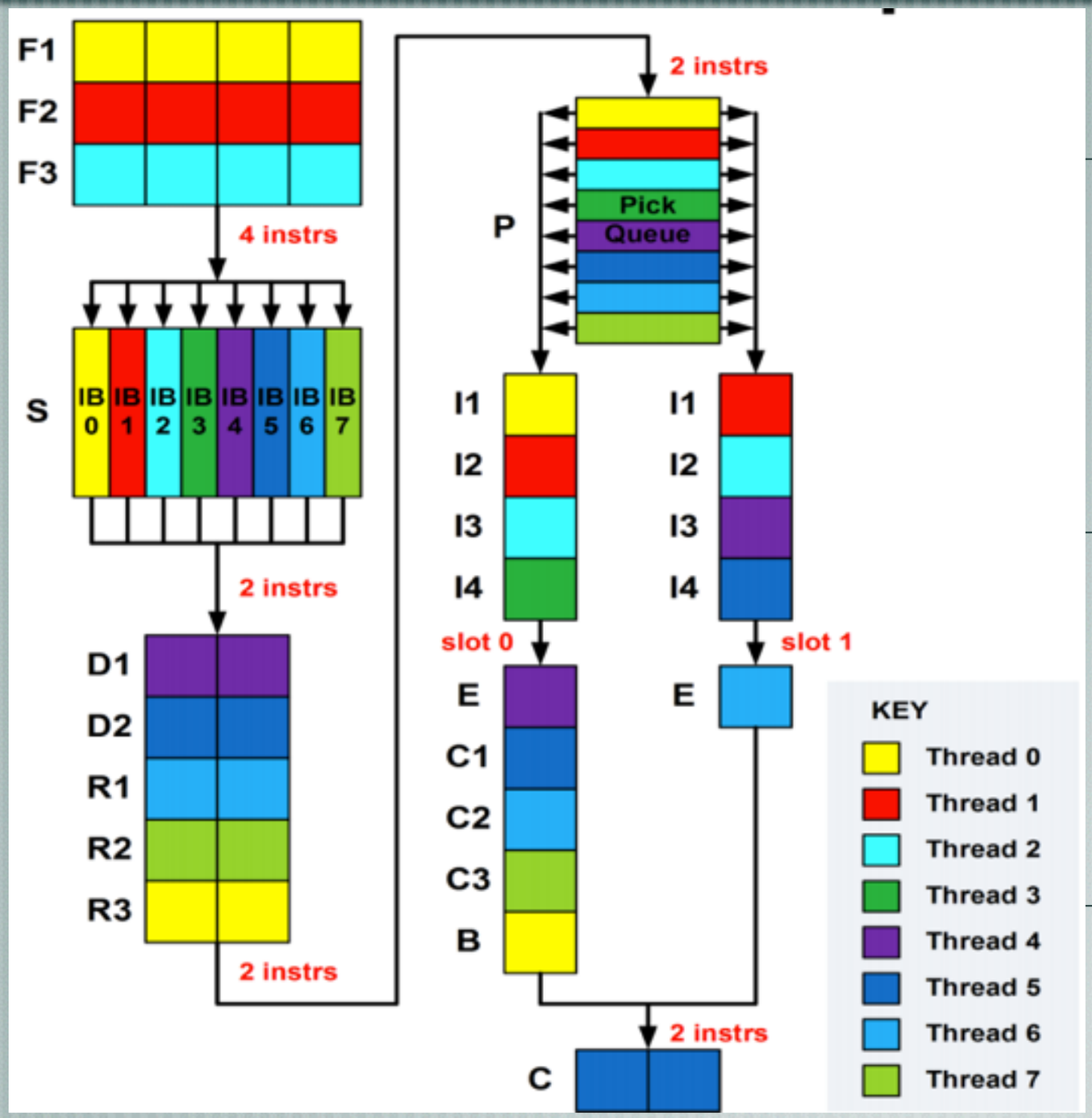
Niagara T4 pipeline



Pipeline Key

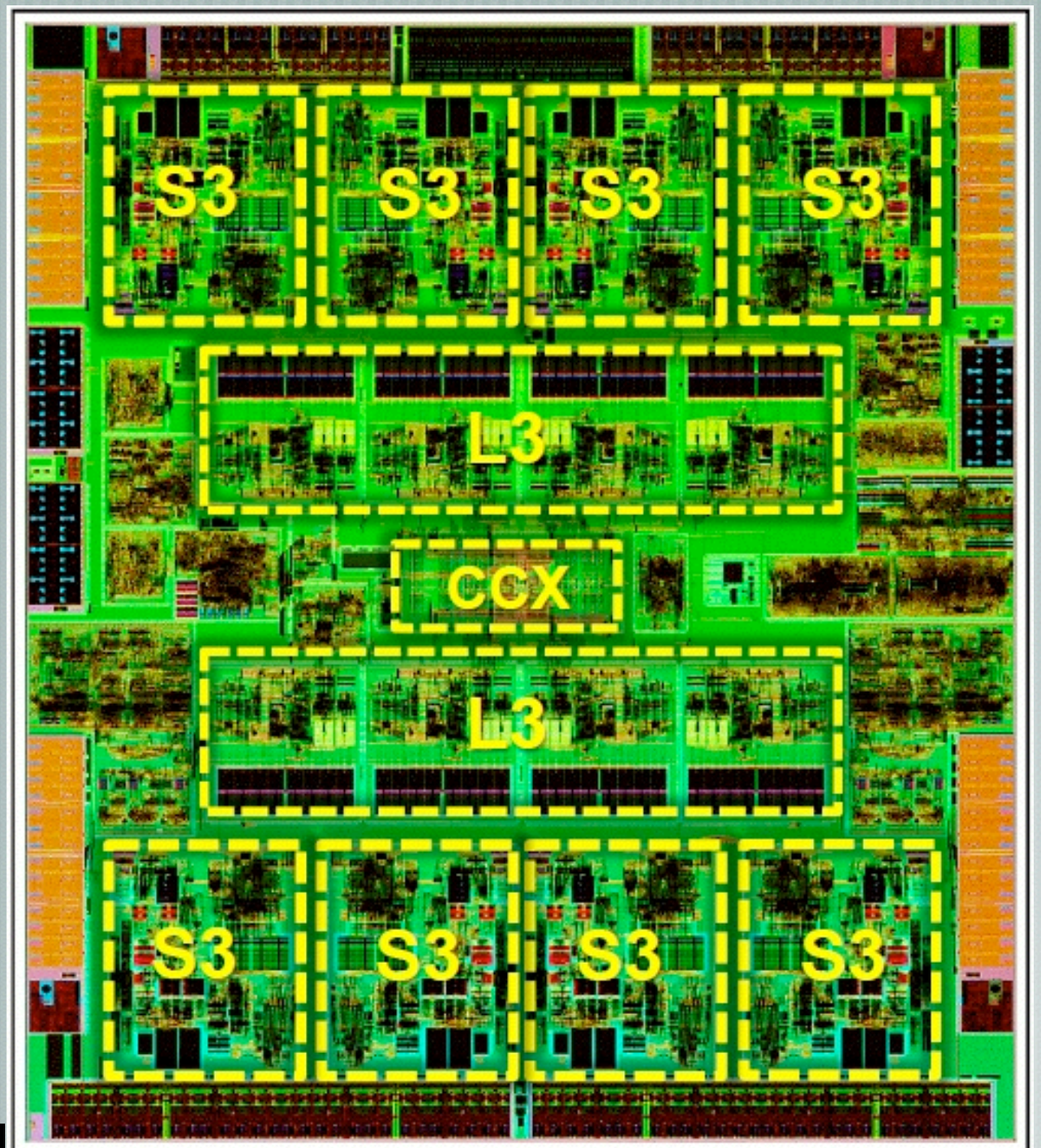
F	Fetch	R	Rename	E	Execute	B	Bypass	FX	Floating-point execute
S	Select	P	Pick	W	Write-back	X	Crypto execute	FB	Floating-point bypass
D	Decode	I	Issue	C	Data-cache			FW	Floating-point writeback

Niagara T4 pipeline



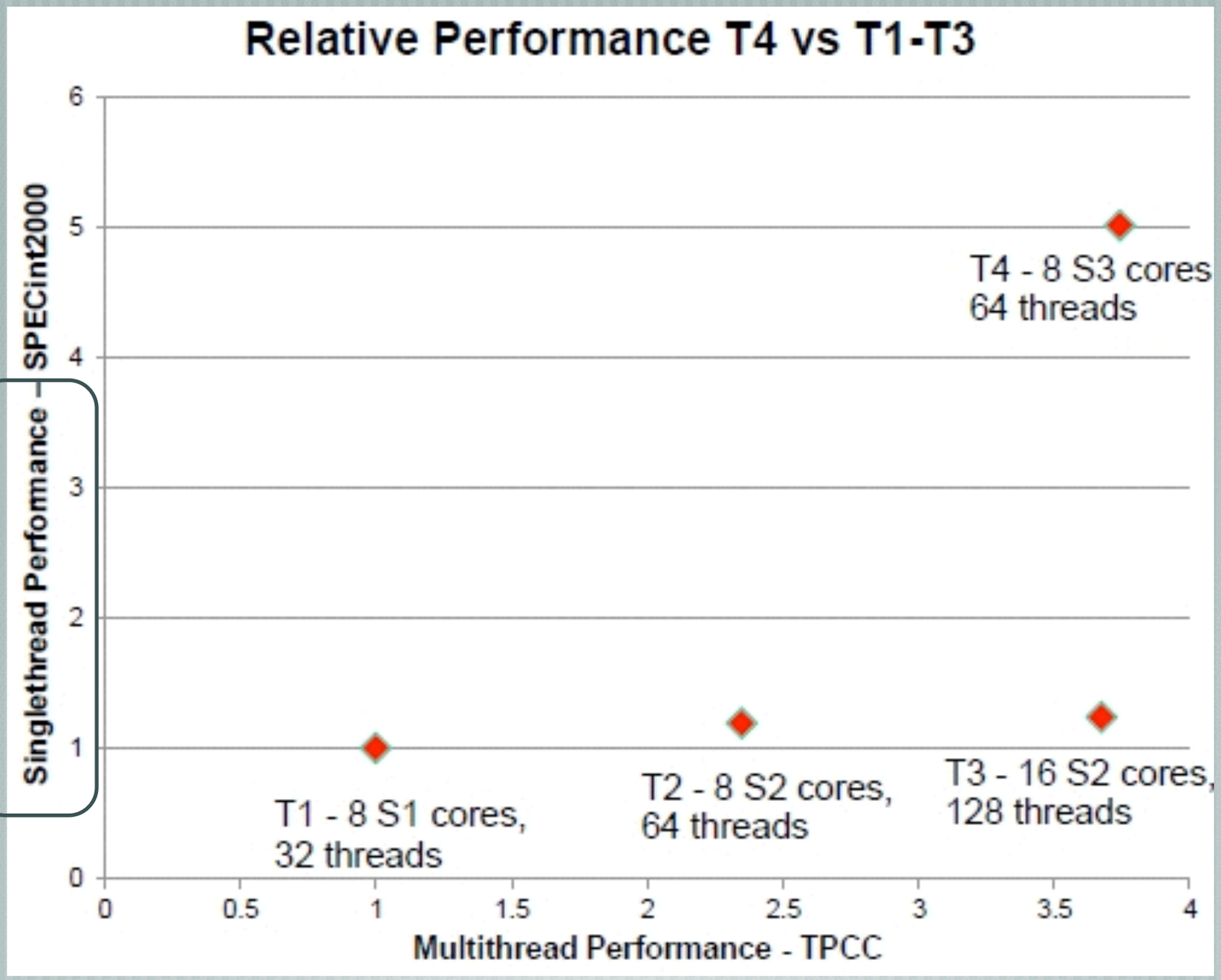
- Before pick: only 1 thread per stage
- Pick to commit: multiple threads per stage
- Commit: 1 thread per stage

Niagara T4



- Telltale signs that sequential performance matters again:
  - new 128KB L2 cache per core, shared L3
  - OoO/BP logic
  - Higher frequency (up to 3GHz)

Niagara T4 sequential performance



Niagara power usage

Chip	TDP	Nominal	Technology	Parallelism
T1		72W	378 mm^2, 90nm	8 cores, 32 threads
T2	123W	95W	342mm^2, 65nm	8 cores, 64 threads
T3	139W	75W	371mm^2, 40nm	16 cores, 128 threads
T4	240W	103W	403mm^2, 40nm	8 cores, 64 threads
T5	?	?	478mm^2, 28nm	16 cores, 128 threads