Multicore-Manycore Programming

Architectures, Compilers and OS/Runtimes

Fabrice Rastello, INRIA Grenoble Rhône-Alpes

Jean-François Méhaut, UJF-CEA/LIG, Grenoble

Agenda

- **Multi-Cores Architectures (JFM)**
  - Introduction
  - Memore Hierarchy
  - Instruction Level Parallelism
  - Simultaneous Multithreading (SMT)

- **Compilation, SSA (FR)**
  - Introduction to SSA
  - Properties and flavors
  - Register Allocations
  - SSA extension

- **Operating Systems for Multi-Cores (JFM)**
  - Multithreading (POSIX Threads, OpenMP,...)
  - Synchronization (Locks/semaphores, Transactional Memory,...)

- **Runtimes and multi-cores (JFM)**

What is Parallel Computing?

- **Parallel computing: using multiple processors/cores in parallel to solve problems more quickly than with a single processor/core**

- **Examples of parallel machines:**
  - **A cluster computer that contains multiple PCs combined together with a high speed network**
  - **A shared memory multiprocessor (SMP*) by connecting multiple processors to a single memory system**
  - **A Chip Multi-Processor (CMP) contains multiple processors (called cores) on a single chip**

- **Concurrent execution comes from desire for performance; unlike the inherent concurrency in a multi-user distributed system**

- *** Technically, SMP stands for "Symmetric Multi-Processor"**

Why Parallel Computing Now?

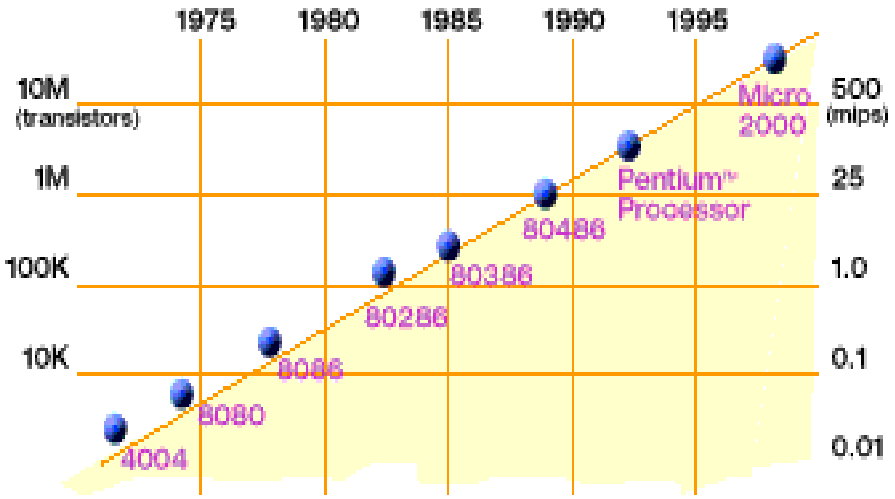
- Researchers have been using parallel computing for decades:
  - Mostly used in computational science and engineering
  - Problems too large to solve on one computer; use 100s or 1000s

- Many companies in the 80s/90s “bet” on parallel computing and failed
  - Computers got faster too quickly for there to be a large market

- Why are Universities adding courses?
  - Because the entire computing industry has bet on parallelism
  - There is a desperate need for parallel programmers

- Let’s see why…

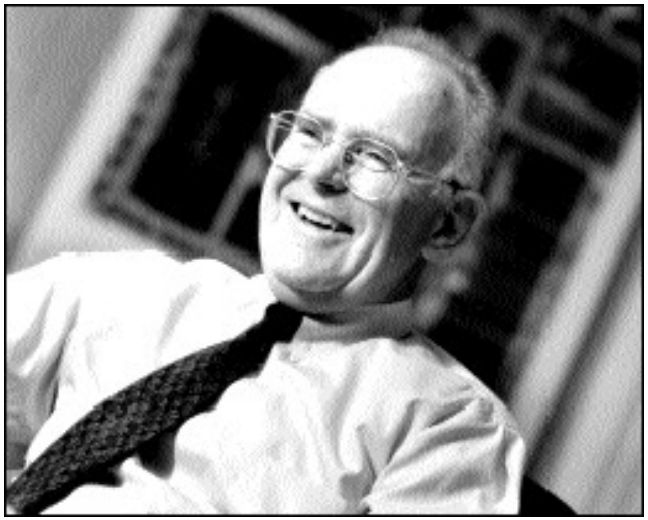
Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years

Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.

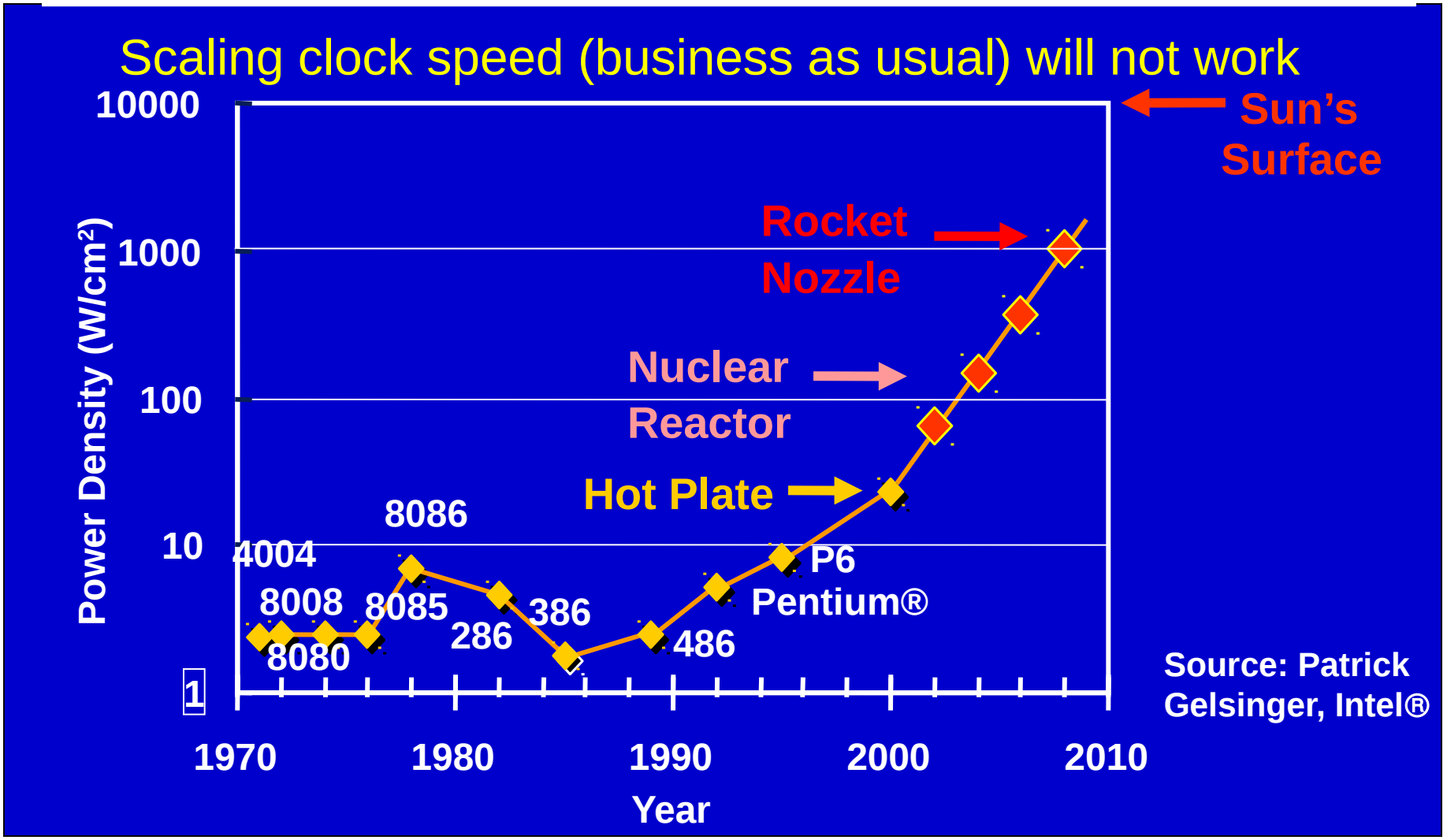


Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Limit #1: Power density

Can soon put more transistors on a chip than can afford to turn on.

-- Patterson '07



Parallelism Saves Power

- **Exploit explicit parallelism for reducing power**

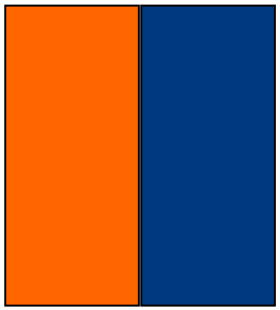
Power = $(C * V^2 * F)/4$ Performance = $(Cores * F)*1$

Capacitance Voltage Frequency

- **Using additional cores**
  - Increase density (= more transistors = more capacitance)
  - Can increase cores (2x) and performance (2x)
  - Or increase cores (2x), but decrease frequency (1/2): same performance at ¼ the power
- **Additional benefits**
  - Small/simple cores → more predictable performance

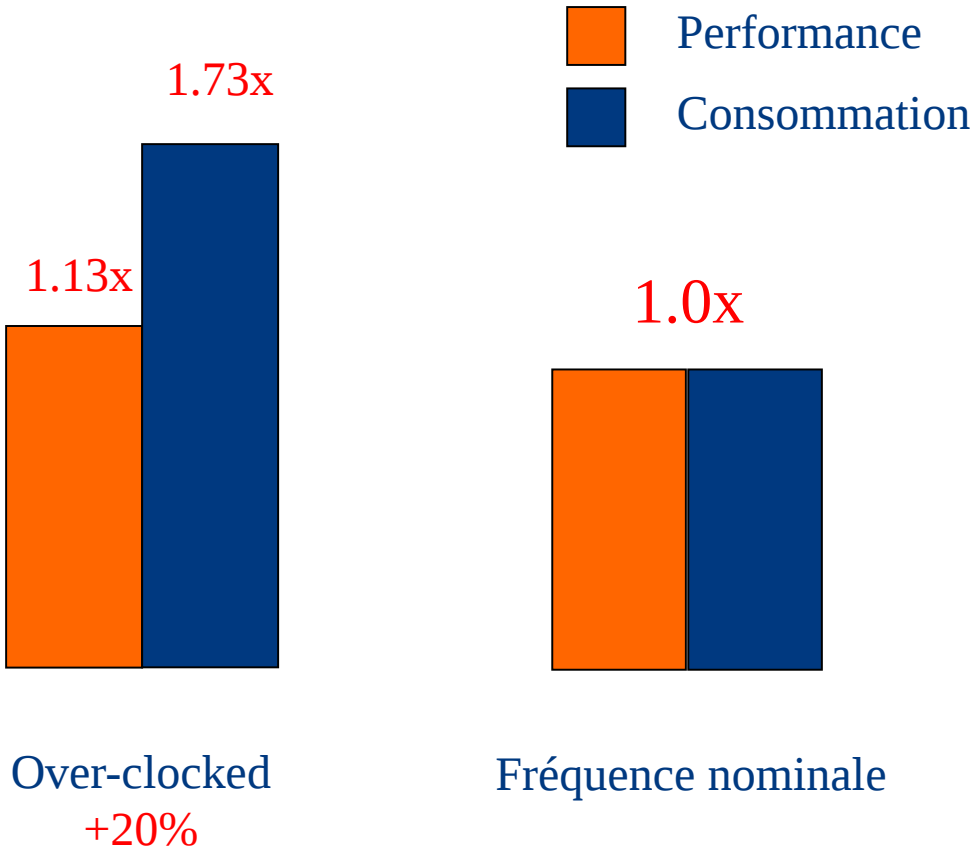
Why Multicore Processors?

1.0x

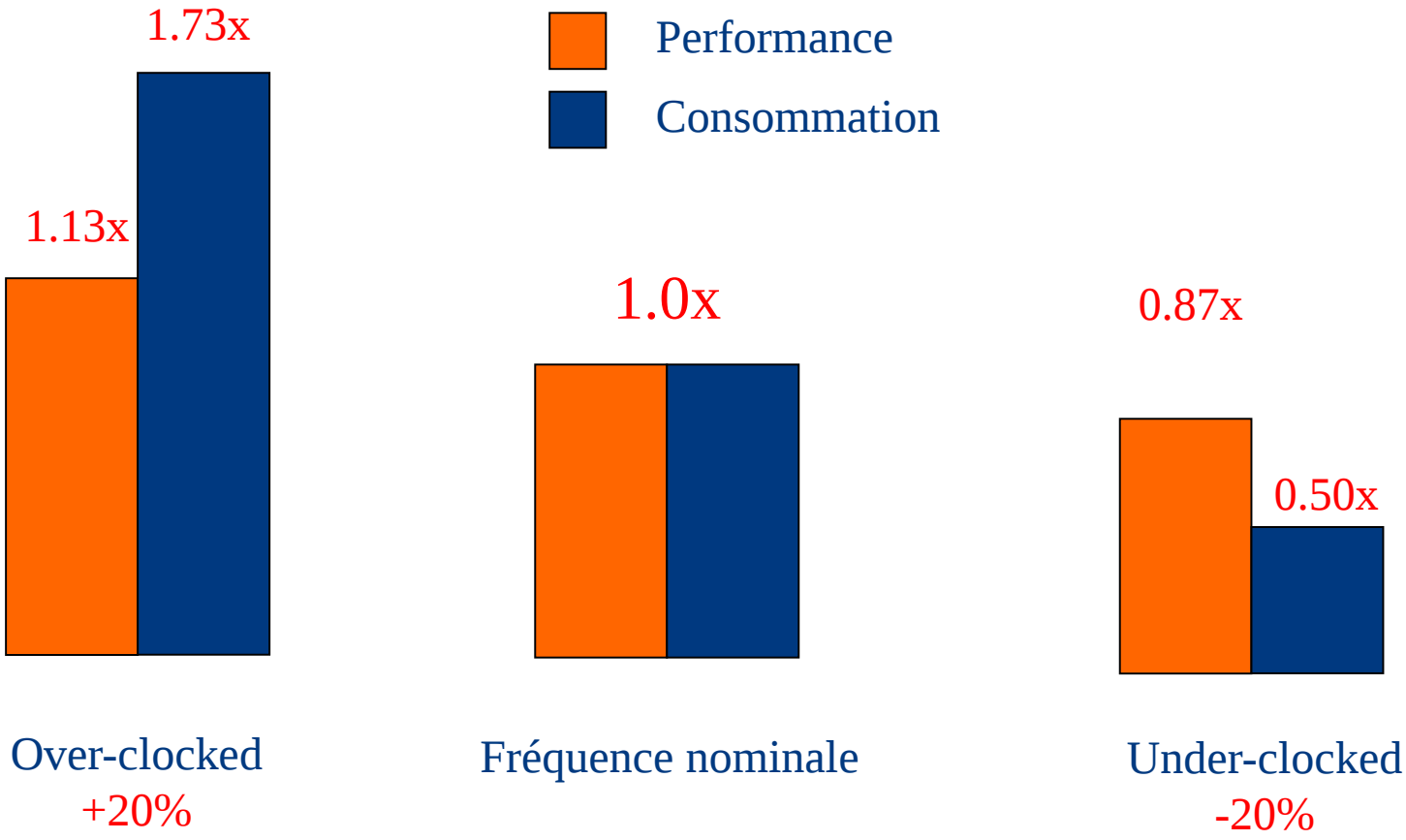


Fréquence nominale

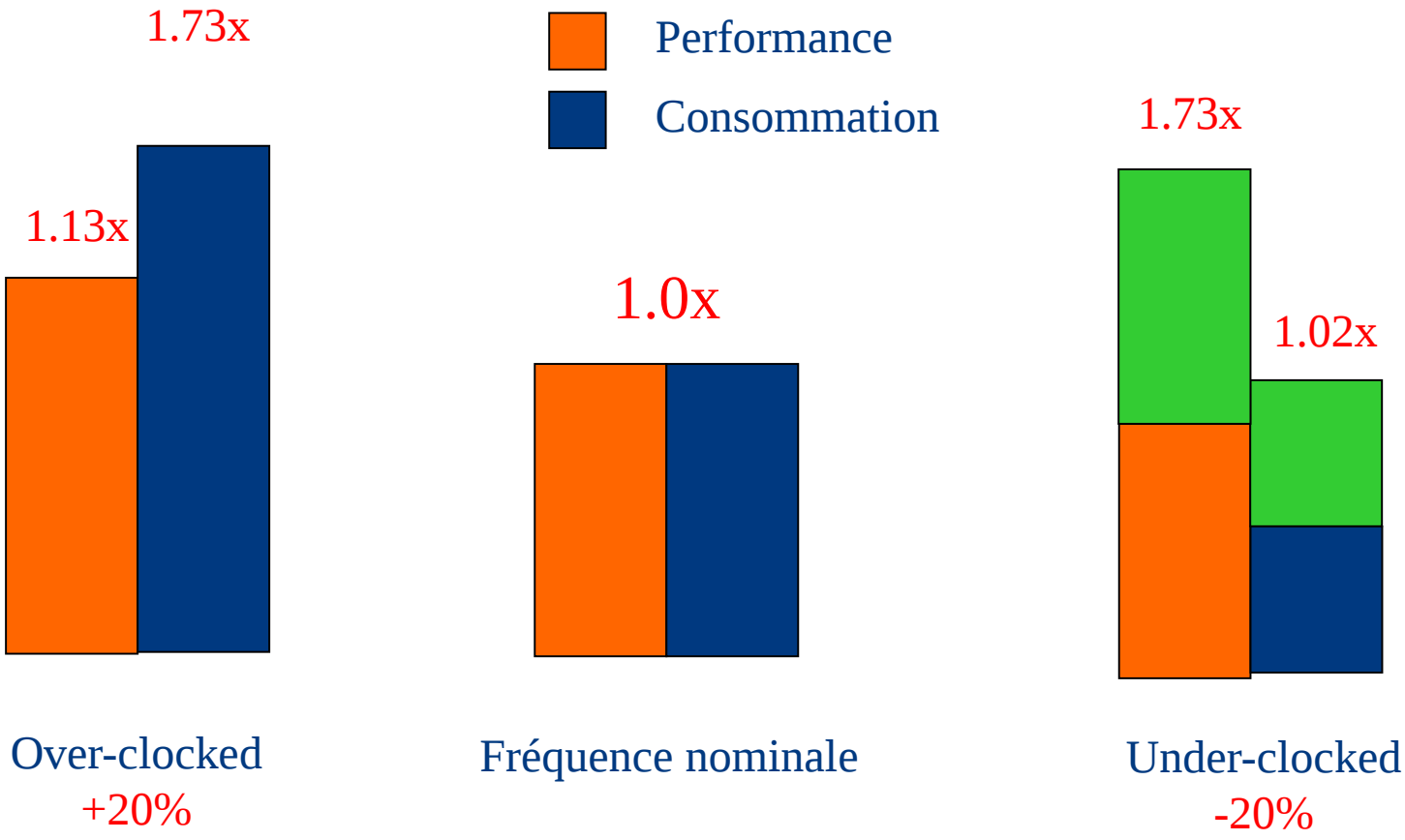
Over-clocking



Under-clocking

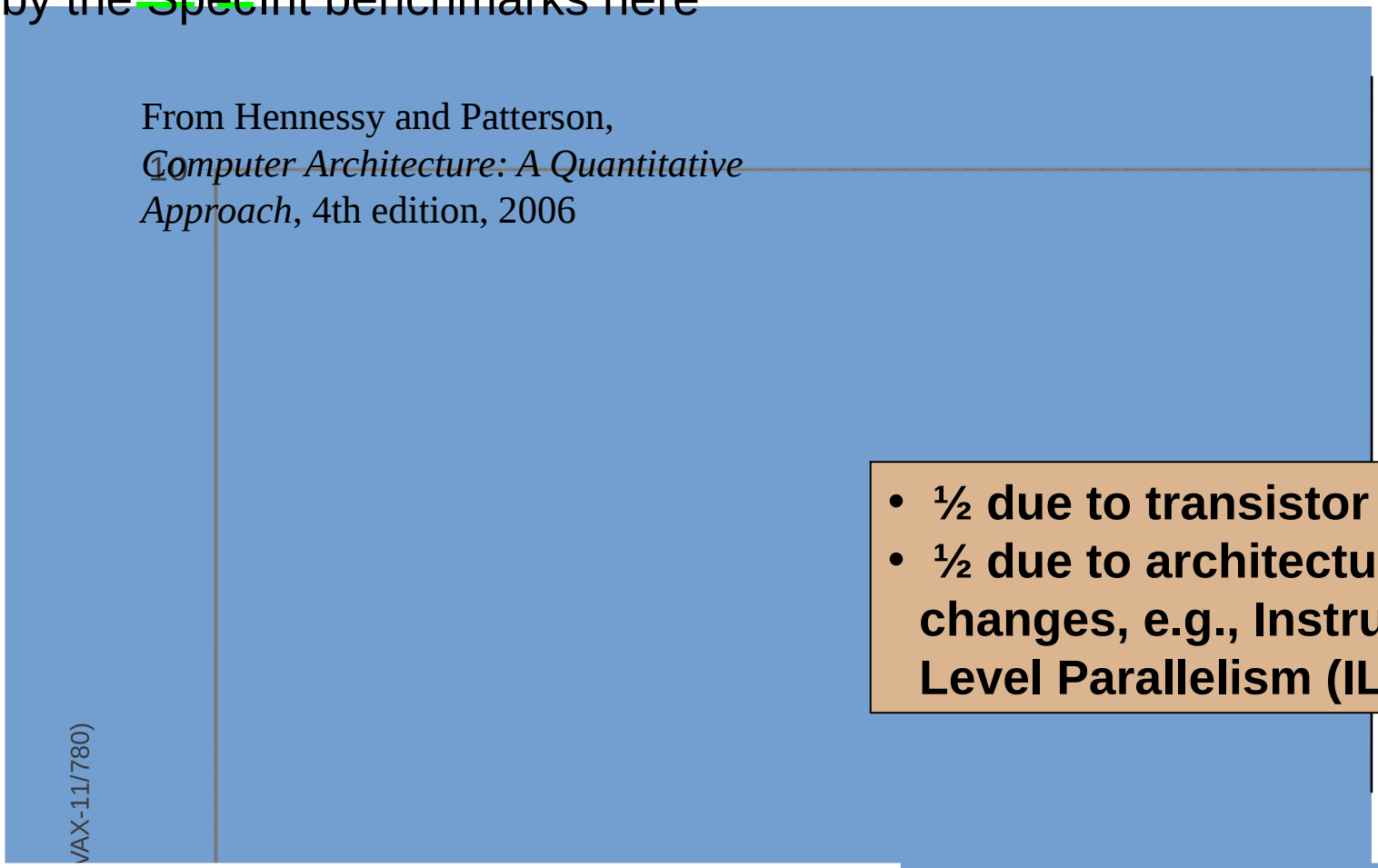


Multi-Core : Energy-Efficient Performance



Limit #2: Hidden Parallelism Tapped Out

Application performance was increasing by 52% per year as measured by the SpecInt benchmarks here



- **½ due to transistor density**
- **½ due to architecture changes, e.g., Instruction Level Parallelism (ILP)**

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

Limit #2: Hidden Parallelism Tapped Out

- **Superscalar (SS) designs were the state of the art; many forms of parallelism not visible to programmer**
  - **multiple instruction issue**
  - **dynamic scheduling: hardware discovers parallelism between instructions**
  - **speculative execution: look past predicted branches**
  - **non-blocking caches: multiple outstanding memory ops**
- **You may have heard of these in your architecture courses, but you haven't needed to know about them to write software (C/C++, Java,...)**
- **Unfortunately, these sources have been used up**

Performance Comparison

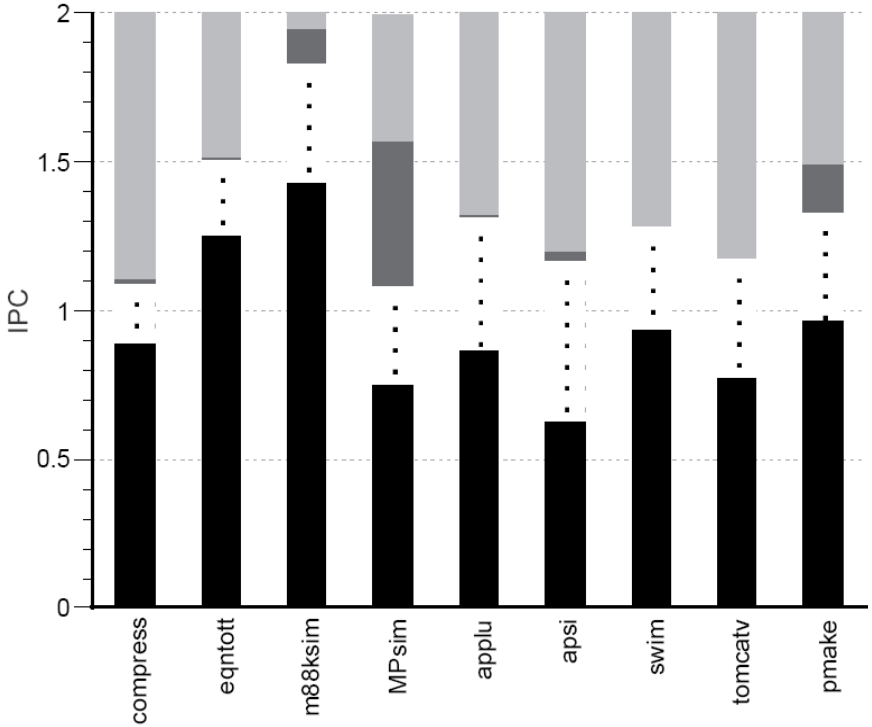


Figure 4. IPC Breakdown for a single 2-issue

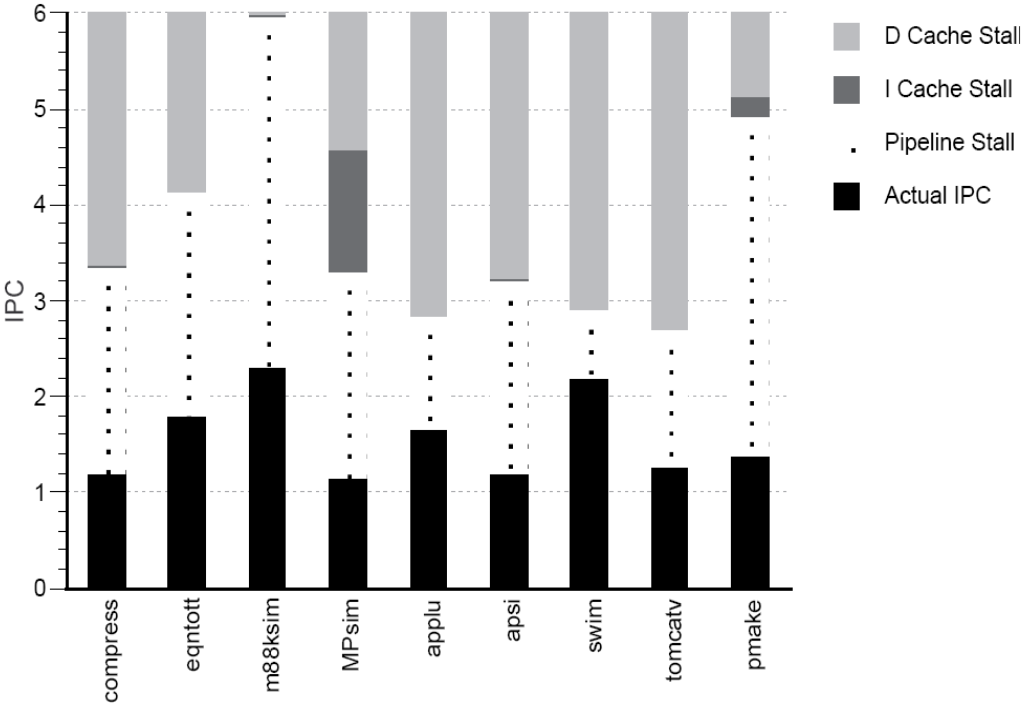
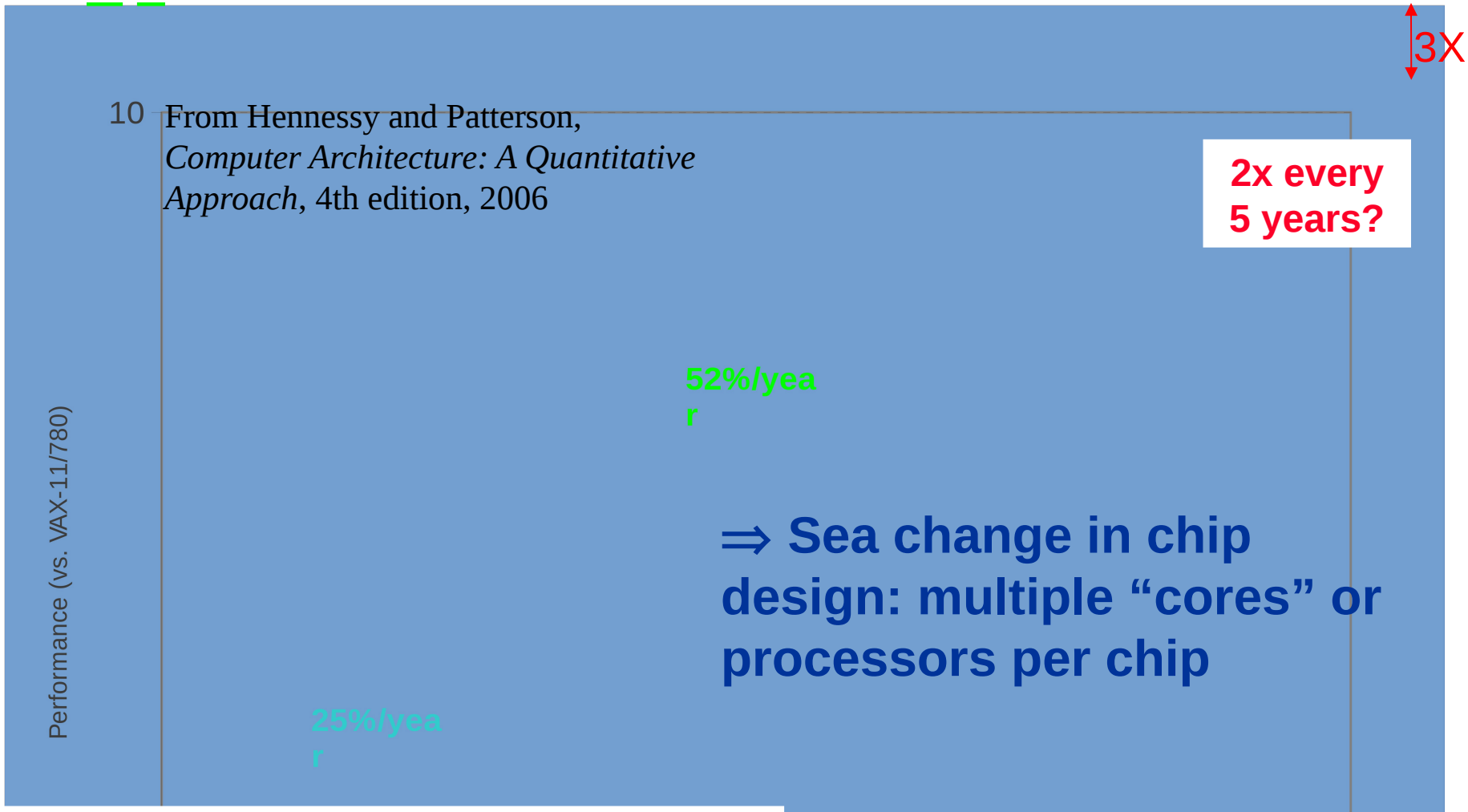


Figure 5. IPC Breakdown for the 6-issue processor.

- **Measure of success for hidden parallelism is Instructions Per Cycle (IPC)**
- **The 6-issue has higher IPC than 2-issue, but far less than 3x**
- **Reasons are: waiting for memory (D and I-cache stalls) and dependencies (pipeline stalls)**

Graphs from: Olukotun et al, ASPLOS, 1996

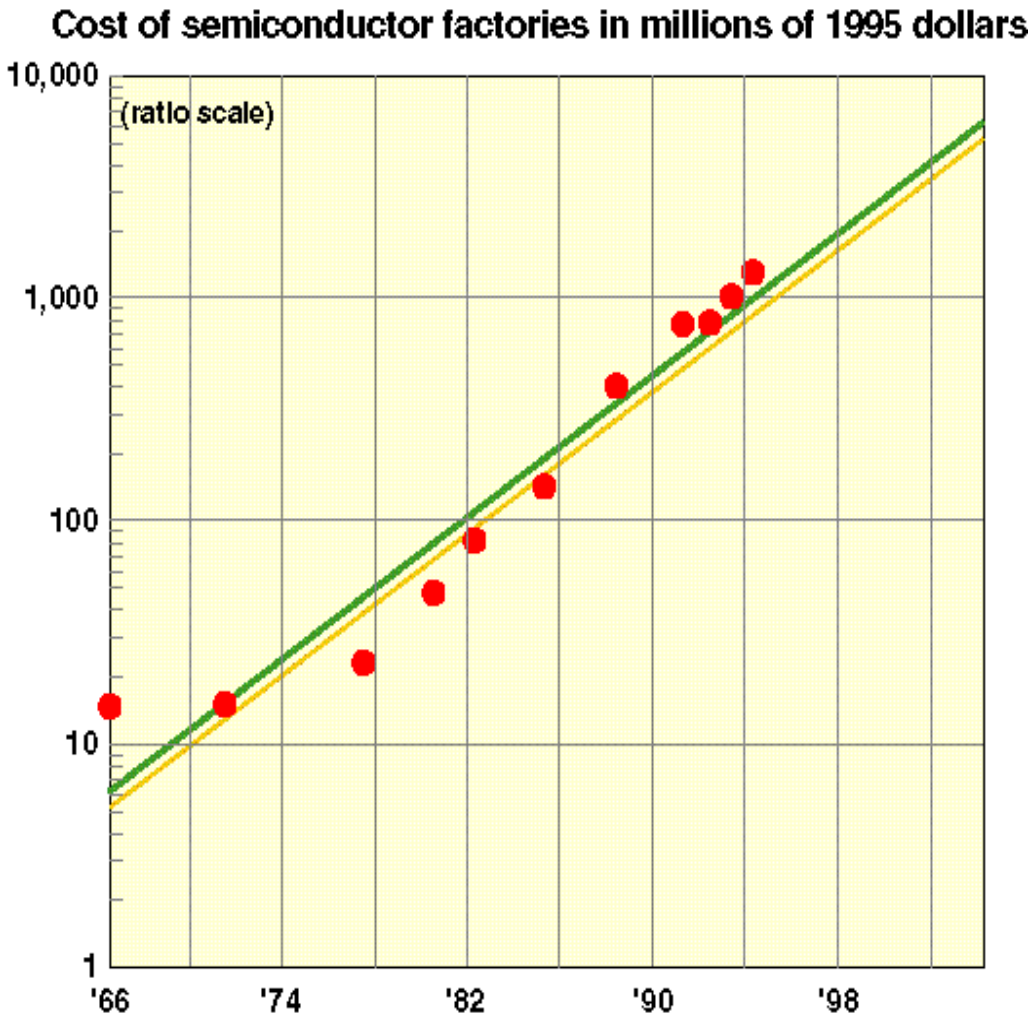
Uniprocessor Performance (SPECint) Today



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

Limit #3: Chip Yield

Manufacturing costs and yield problems limit use of density

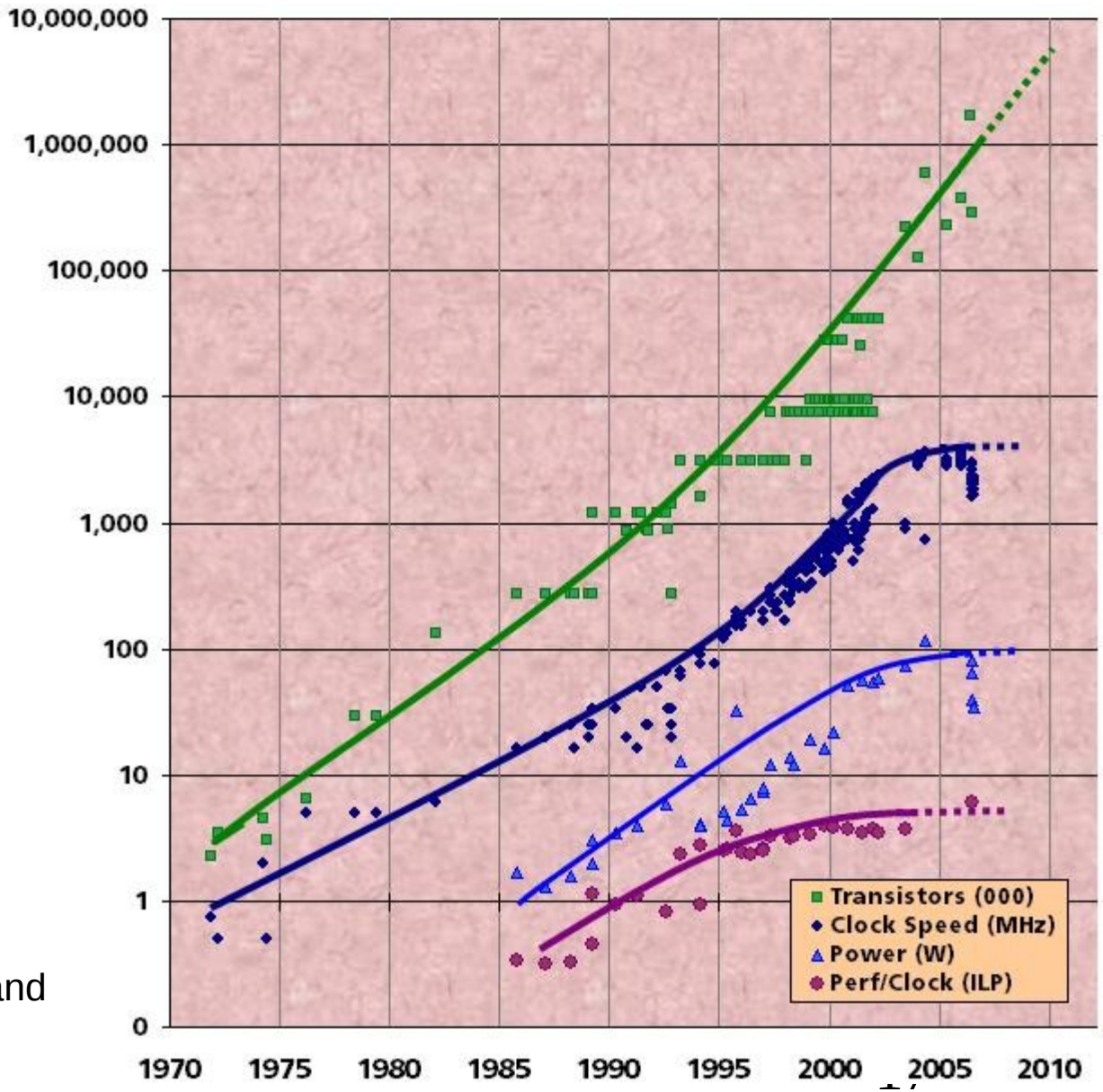


- **Moore's (Rock's) 2nd law: fabrication costs go up**
- **Yield (% usable chips) drops**

- **Parallelism can help**
  - **More smaller, simpler processors are easier to design and validate**
  - **Can use partially working chips:**
  - **E.g., Cell processor (PS3) is sold with 7 out of 8 "on" to improve yield**

Revolution is Happening Now

- **Chip density is continuing increase ~2x every 2 years**
  - **Clock speed is not**
  - **Number of processor cores may double instead**
- **There is little or no hidden parallelism (ILP) to be found**
- **Parallelism must be exposed to and managed by software**



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)

Multicore in Products

- **"We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"**

  Paul Otellini, President, Intel (2005)

- **All microprocessor companies switch to MP (2X CPUs / 2 yrs)**

Manufacturer/Year	AMD/'12	Intel/'13	IBM/'13	Sun/'07
Processors/chip	16	8	12	8
Threads/Processor	1	16	8	16
Threads/chip	16	32	96	128

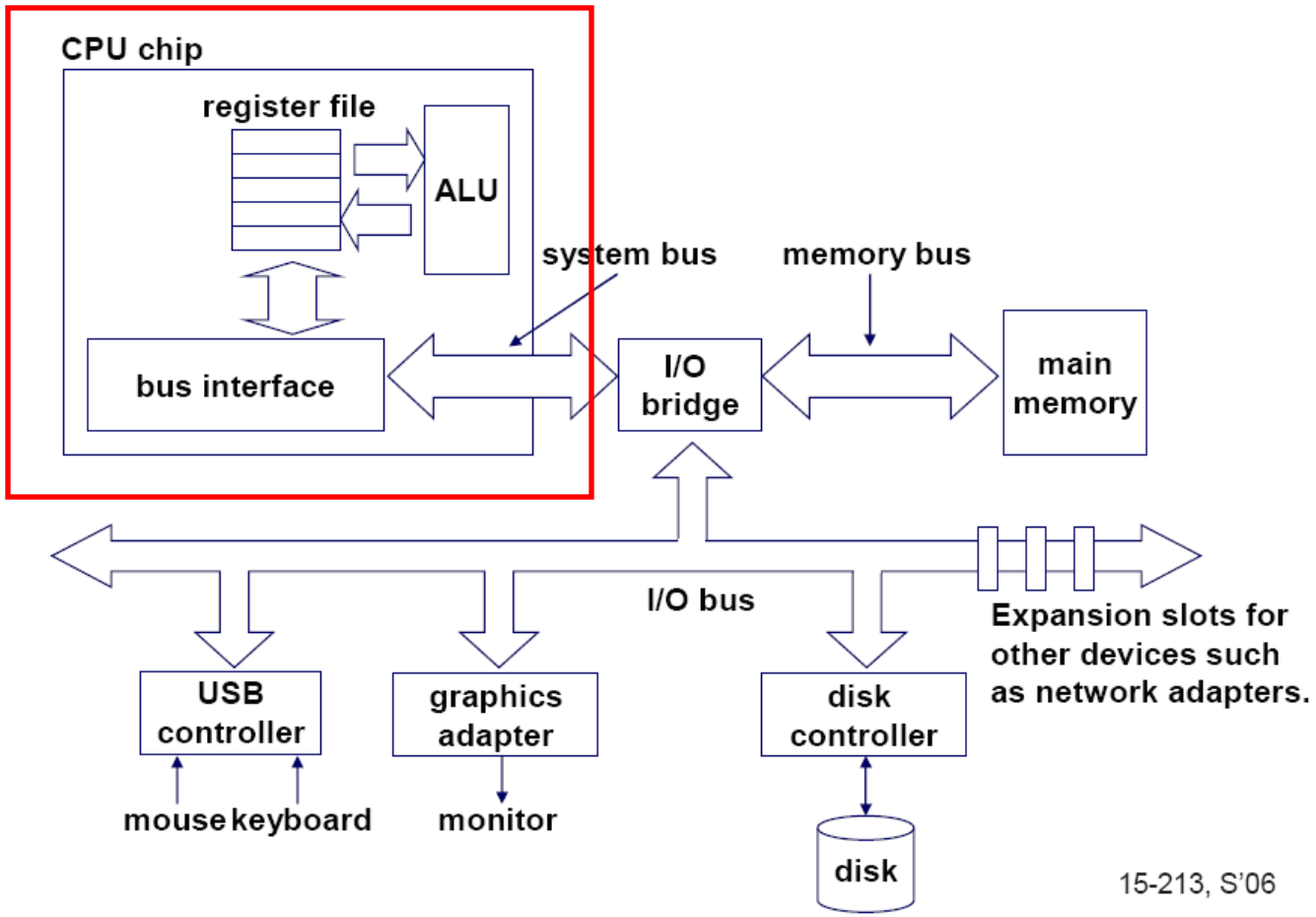
ne time,

MPPA has 256 cores, for 7 Watts

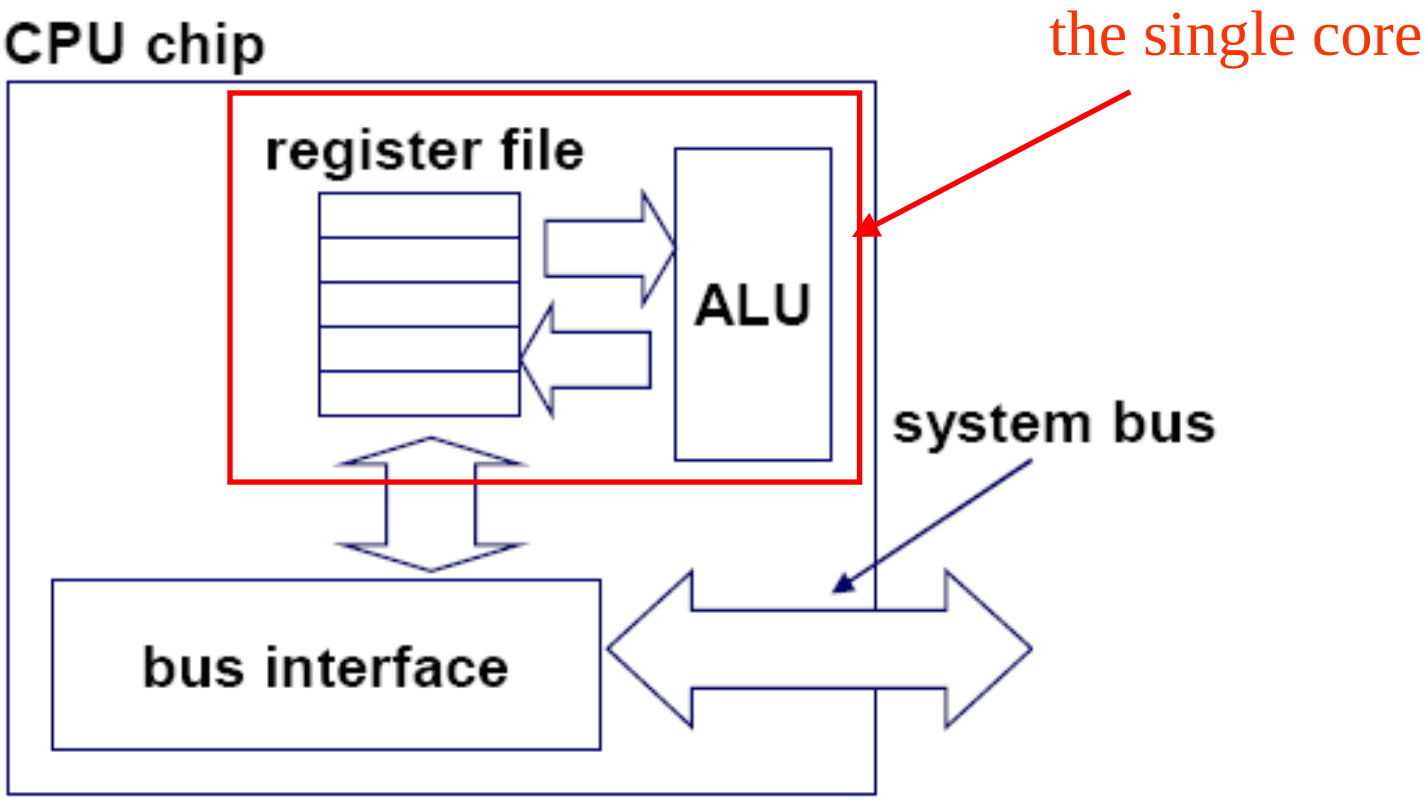
Vidia Graphics Processing Unit (Kepler) has 2880 cores

- **The Intel Xeon Phi 7100 has 61 cores**

Single-core computer



Single-Core CPU Chip

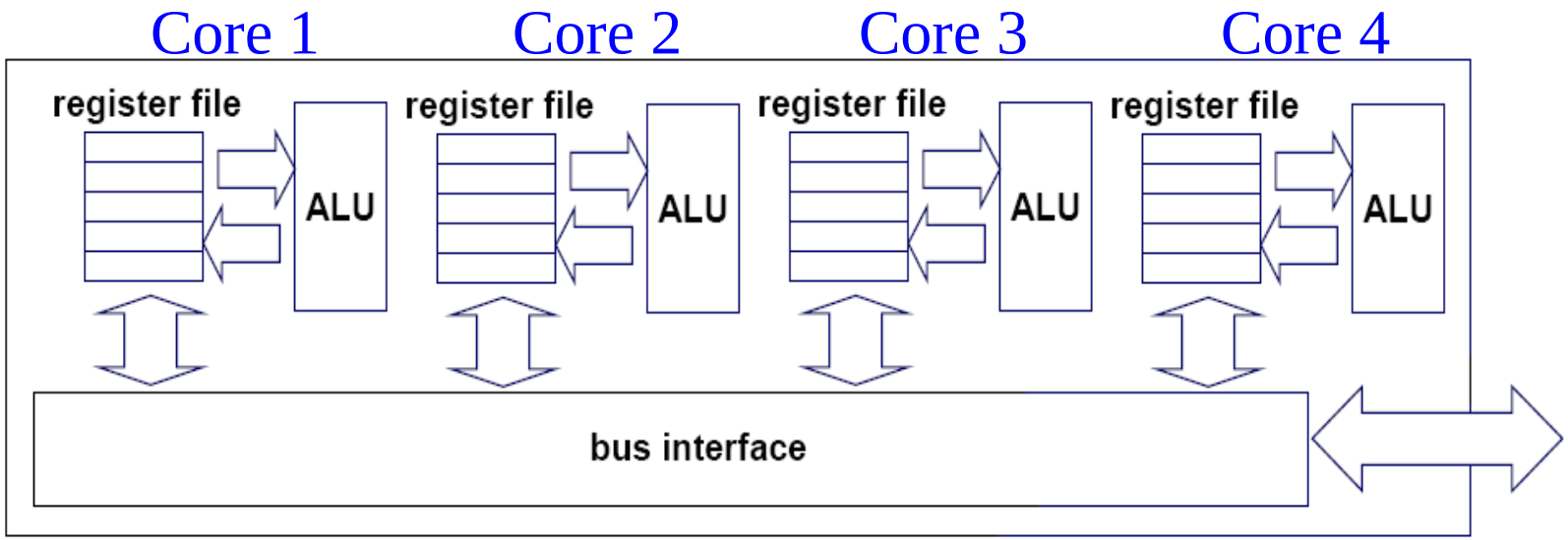


Intel Nehalem Register File

- 16 64-bit general purpose registers
- 6 16-bit segment register
- 64-bit instruction pointer register
- 64-bit RFLAGS register
- 8 64-bit MMX registers
- 16 128-bit XMM registers
- 32-bit MXCSR register
- 60 Performance Registers

Multi-Core Architectures

- **This lecture is about a new trend in computer architecture: Replicate multiple processor cores on a single die.**



Multi-core CPU chip

Performance Registers (1)

- Level 1 data cache misses
- Level 1 instruction cache misses
- Level 2 data cache misses
- Level 2 instruction cache misses
- Level 3 data cache misses
- Level 3 instruction cache misses
- Level 1 cache misses
- Level 2 cache misses
- Level 3 cache misses
- Requests for a snoop
- Requests for exclusive access to shared cache line
- Requests for exclusive access to clean cache line
- Requests for cache line invalidation
- Requests for cache line intervention
- Level 3 load misses
- Level 3 store misses
- Cycles branch units are idle
- Cycles integer units are idle
- Cycles floating point units are idle
- Cycles load/store units are idle
- Data translation lookaside buffer misses
- Instruction translation lookaside buffer misses
- Total translation lookaside buffer misses

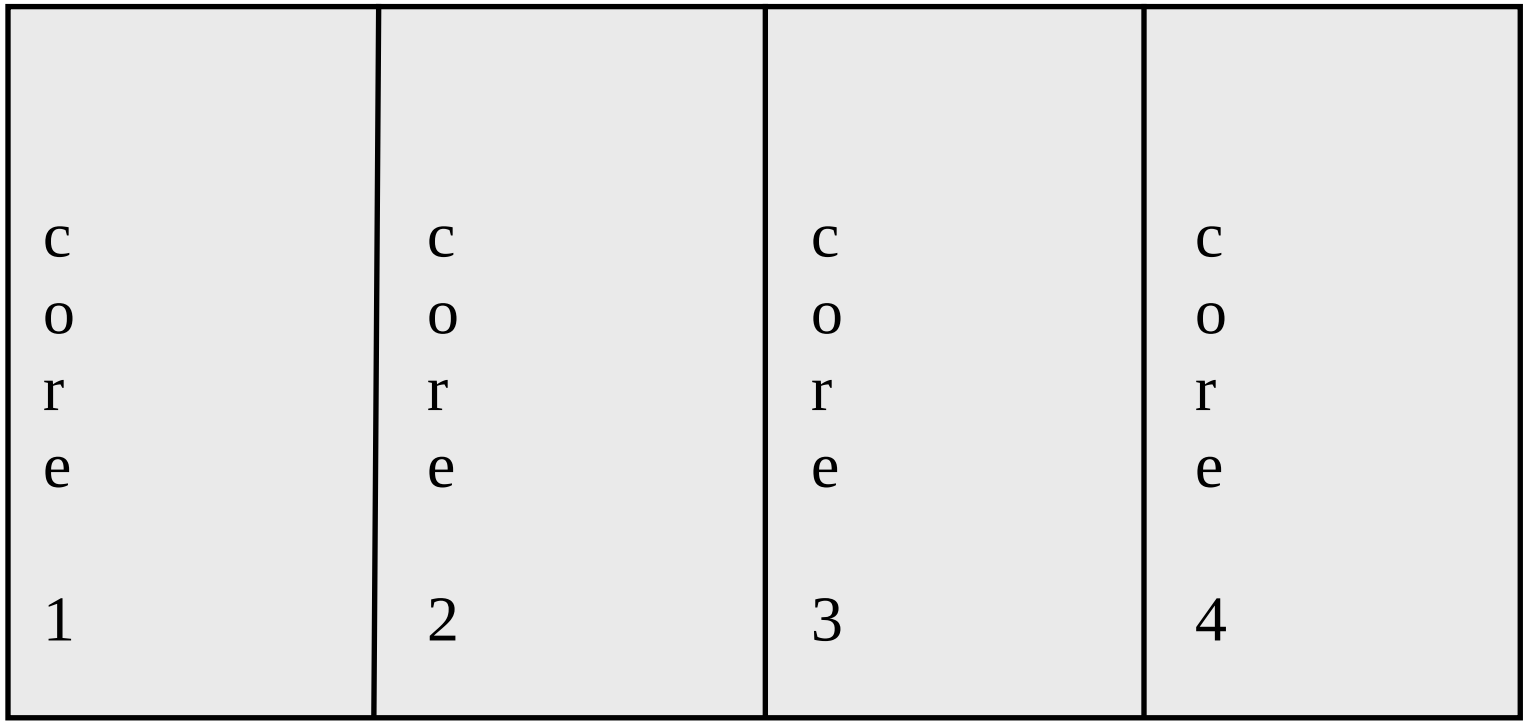
s

Performance Registers (2)

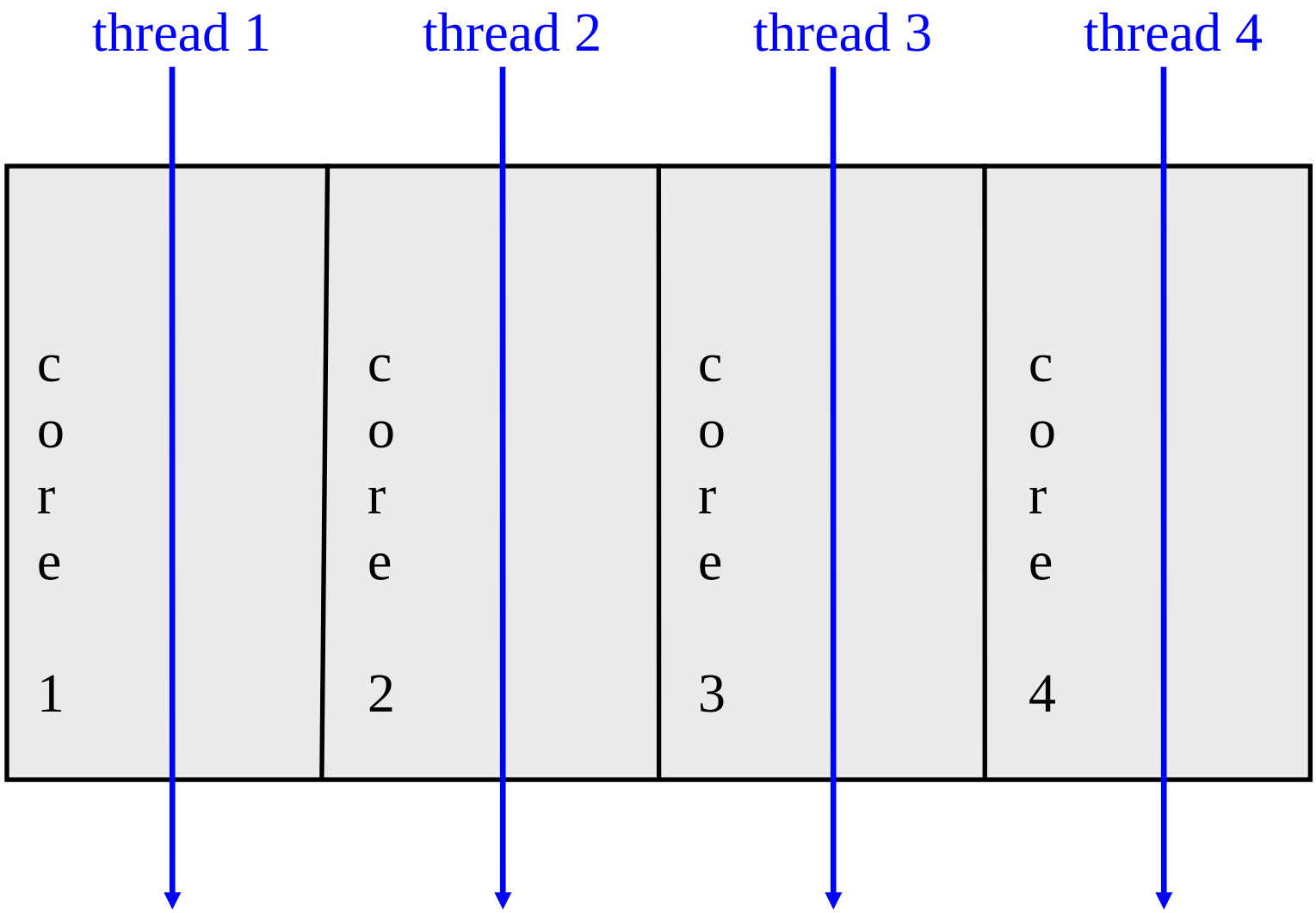
- Level 1 load misses
- Level 1 store misses
- Level 2 load misses
- Level 2 store misses
- Branch target address cache misses
- Data prefetch cache misses
- Level 3 data cache hits
- Translation lookaside buffer shootdowns
- Failed store conditional instructions
- Successful store conditional instructions
- Total store conditional instructions
- Cycles Stalled Waiting for memory accesses
- Cycles Stalled Waiting for memory Reads
- Cycles Stalled Waiting for memory writes
- Cycles with no instruction issue
- Cycles with maximum instruction issue
- Cycles with no instructions completed
- Cycles with maximum instructions completed
- Hardware interrupts
- Floating point operations; optimized to count scaled single precision vector operations
- Floating point operations; optimized to count scaled double precision vector operations
- Single precision vector/SIMD instructions
- Double precision vector/SIMD instructions

Multi-Core CPU chip

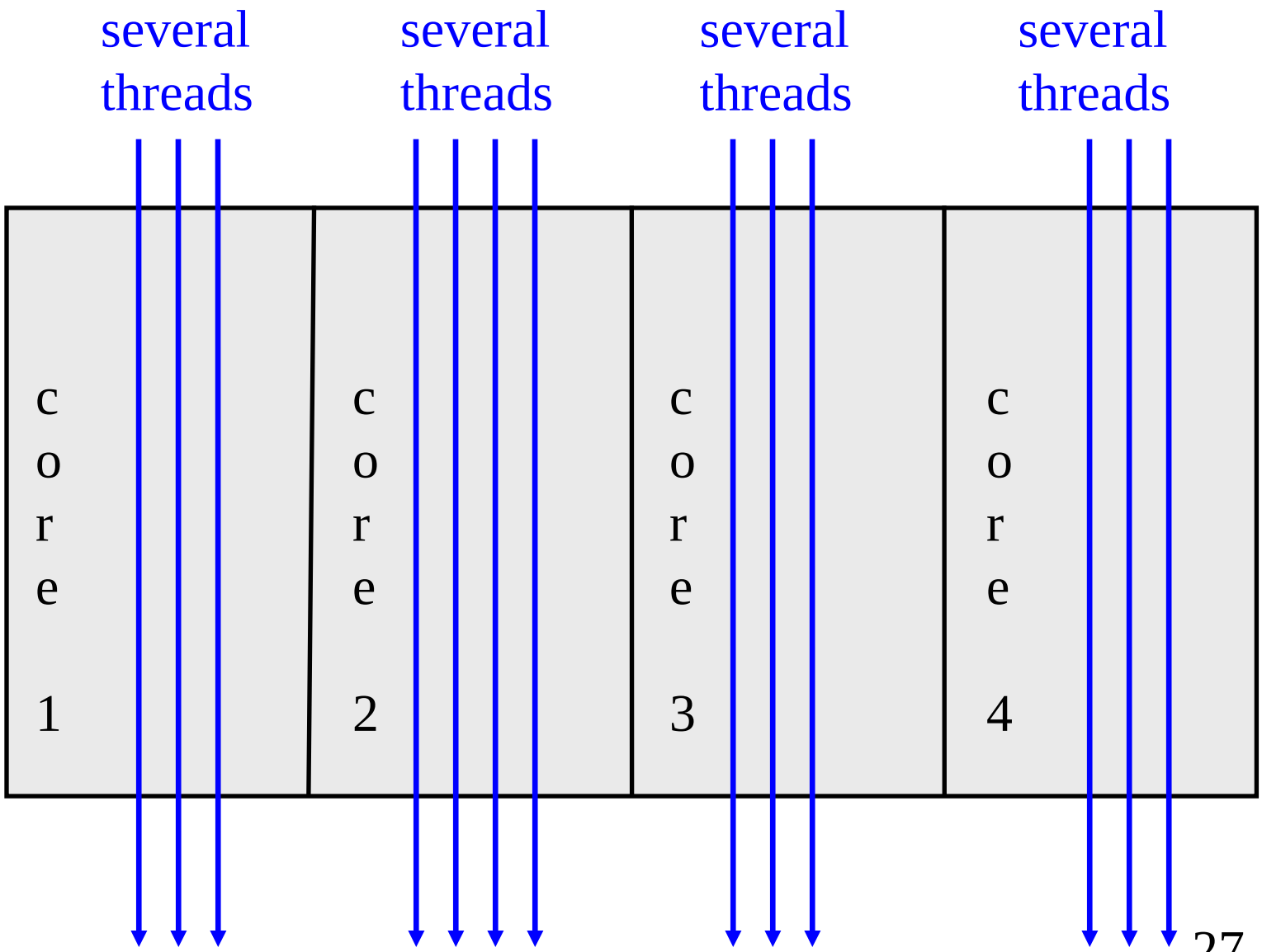
- **The cores fit on a single processor socket**
- **Also called CMP (Chip Multi-Processor)**



The cores run in parallel



Within each core, threads are time-sliced (just like on a uniprocessor)



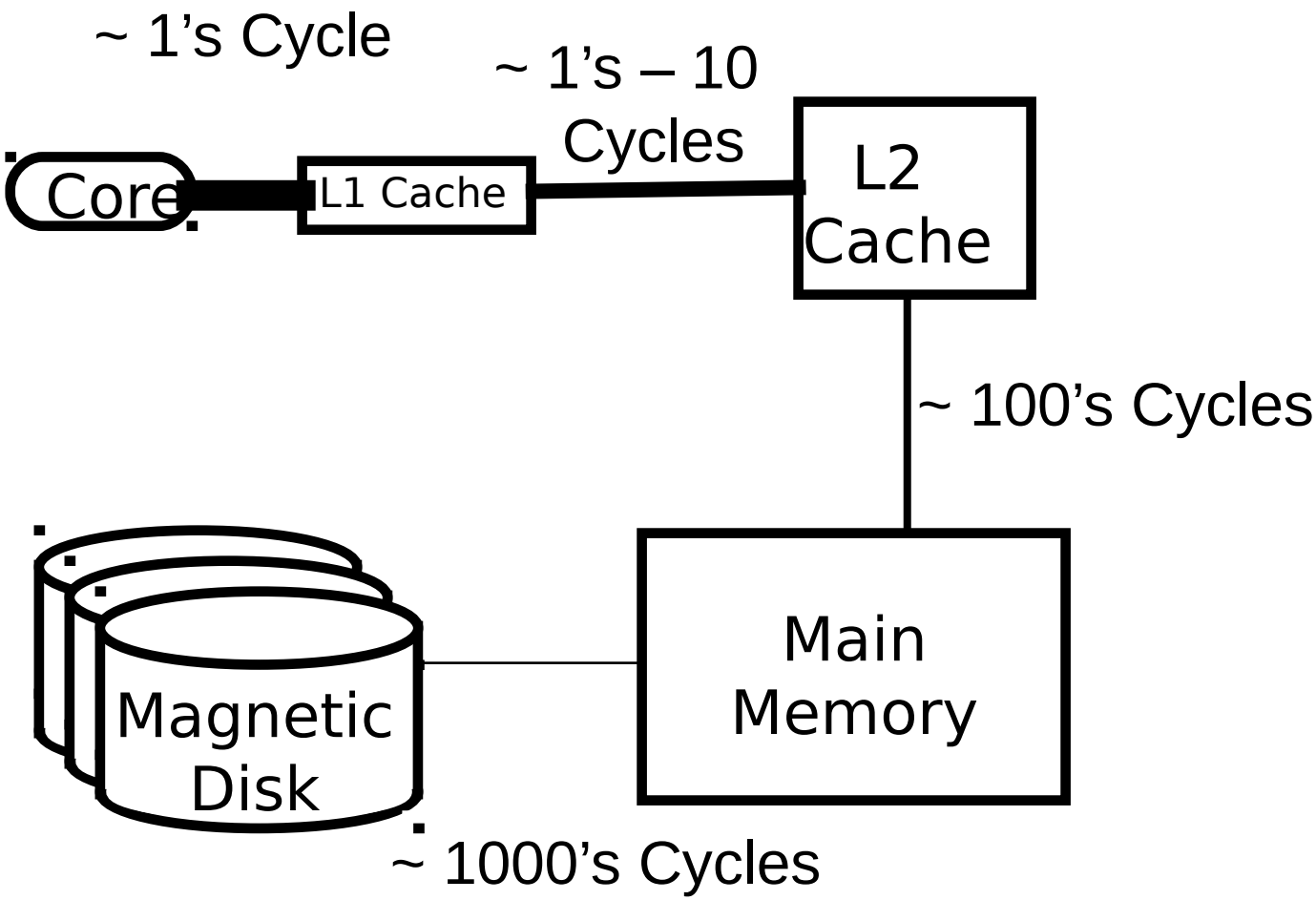
Interactions with the Operating System

- **OS perceives each core as a separate processor**
- **OS scheduler maps threads/processes to different cores**
- **Most major OS support multi-core today: Windows, Linux, Mac OS X, …**

Memory Hierarchy

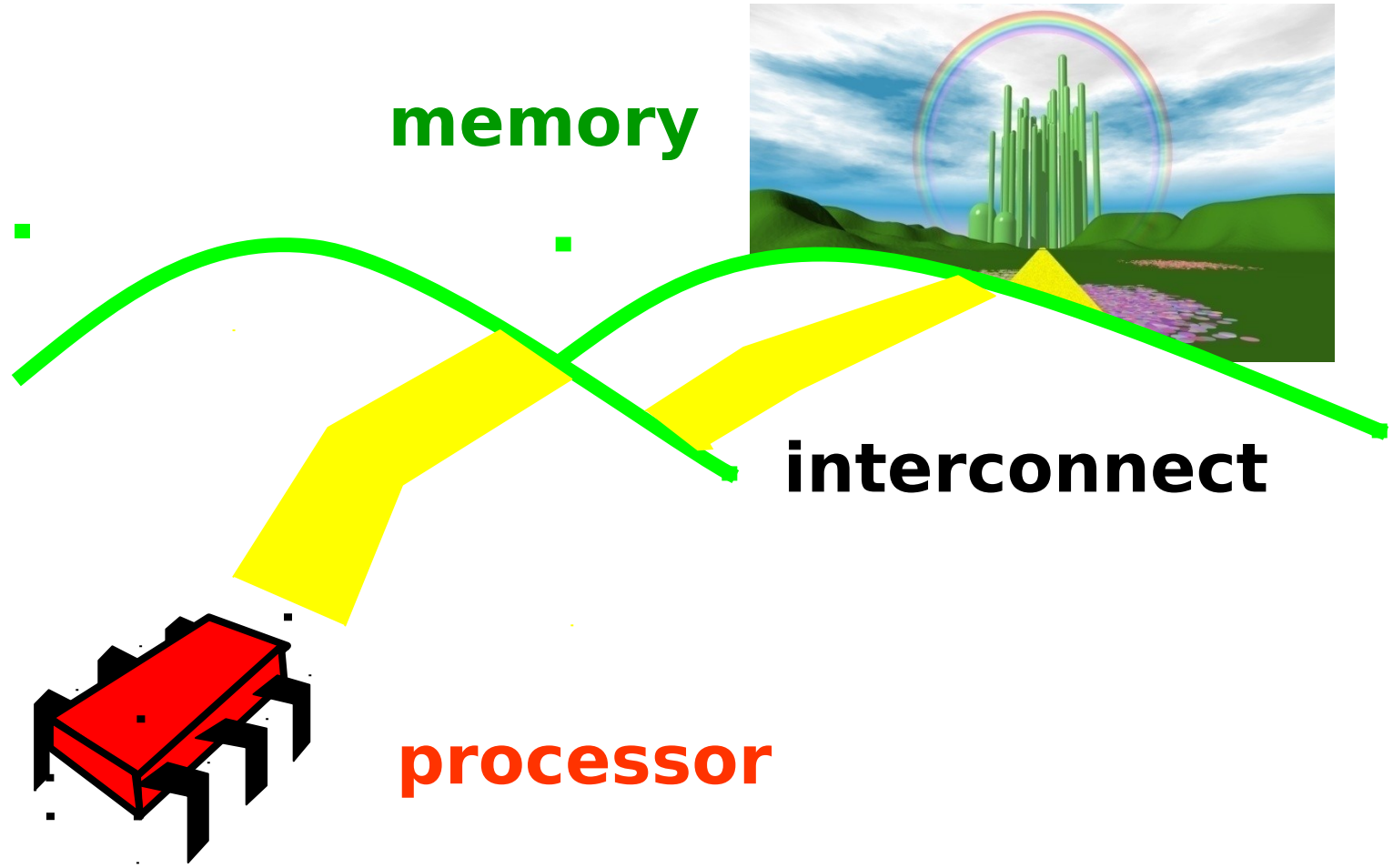
Key Observations

- Access to L1 cache is on order of 1 cycle
- Access to L2 on order of 1 to 10 cycles
- Access to Main memory ~ 100's cycles
- Access to Disk ~ 1000's cycles

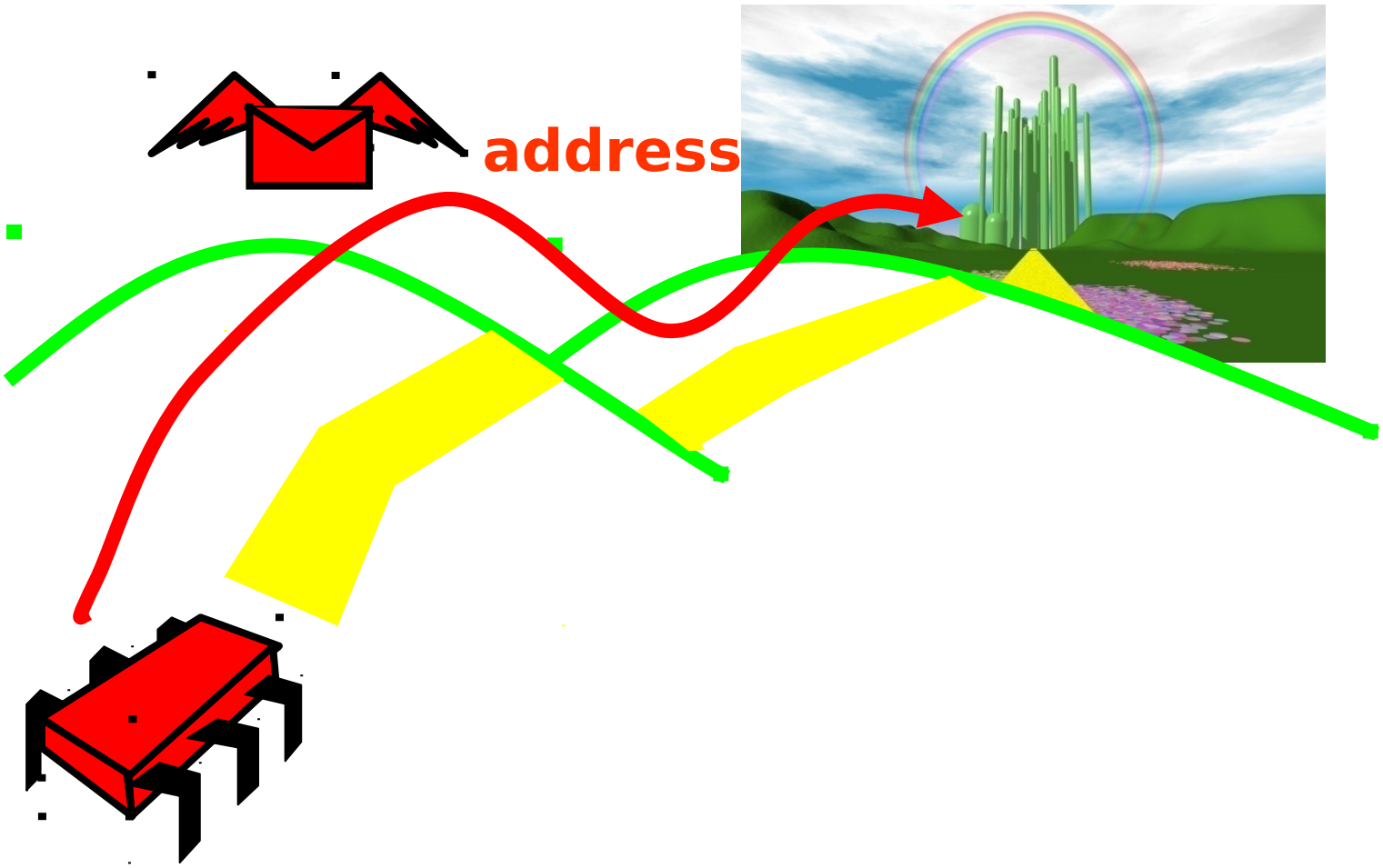


Connecting lines **thickness** depict bandwidth: Bytes/Second

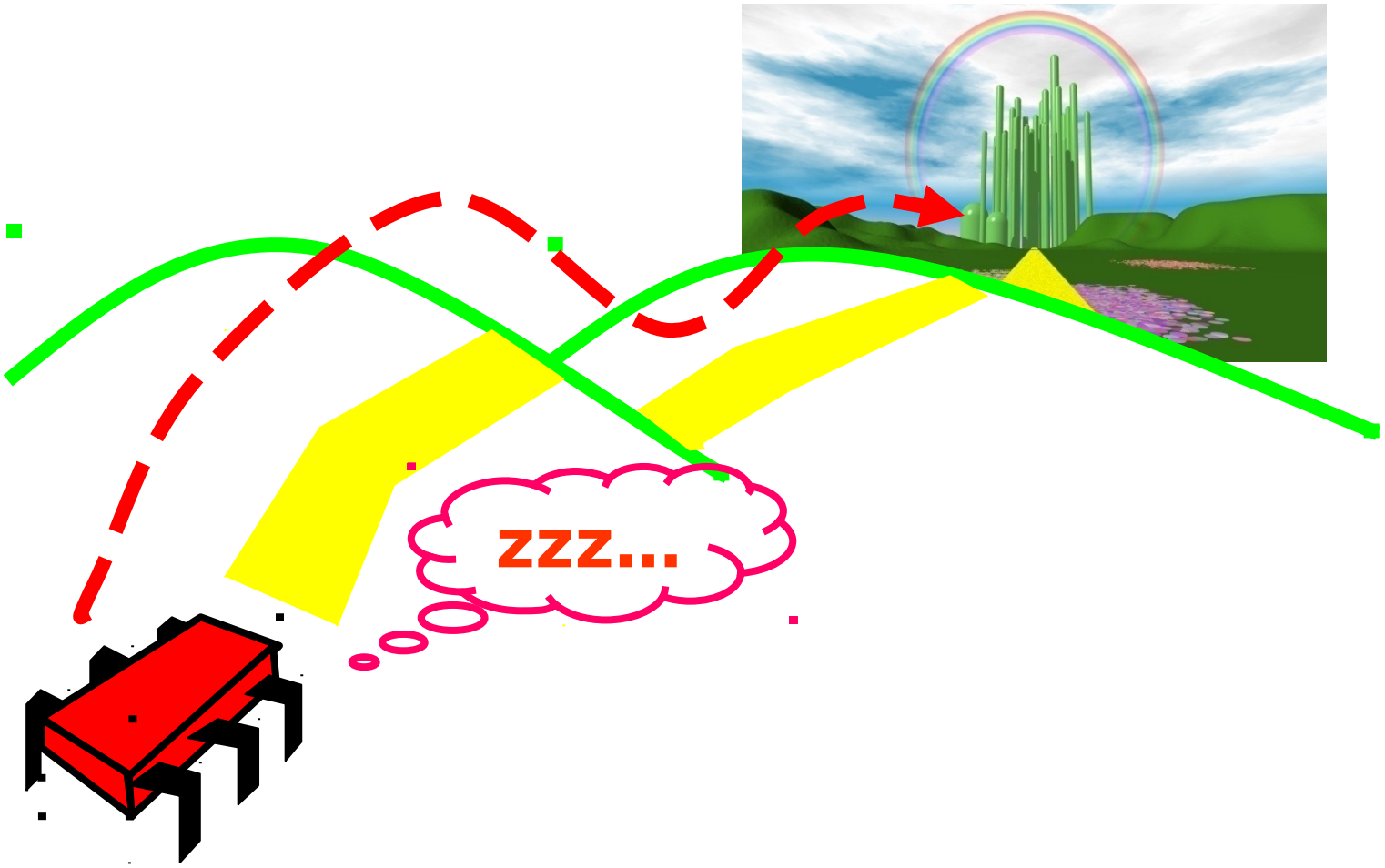
Processor and Memory are Far Apart



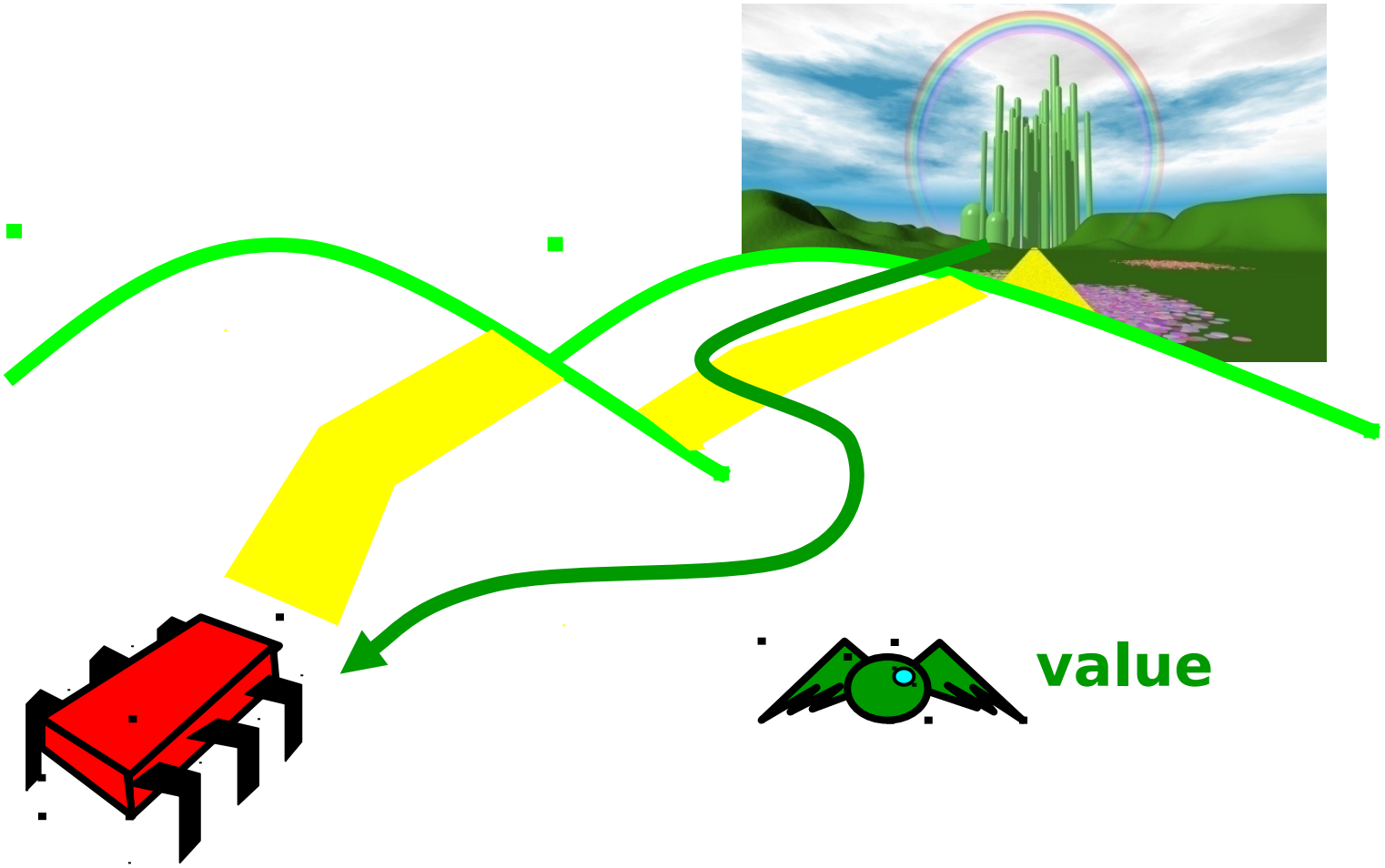
Reading from Memory



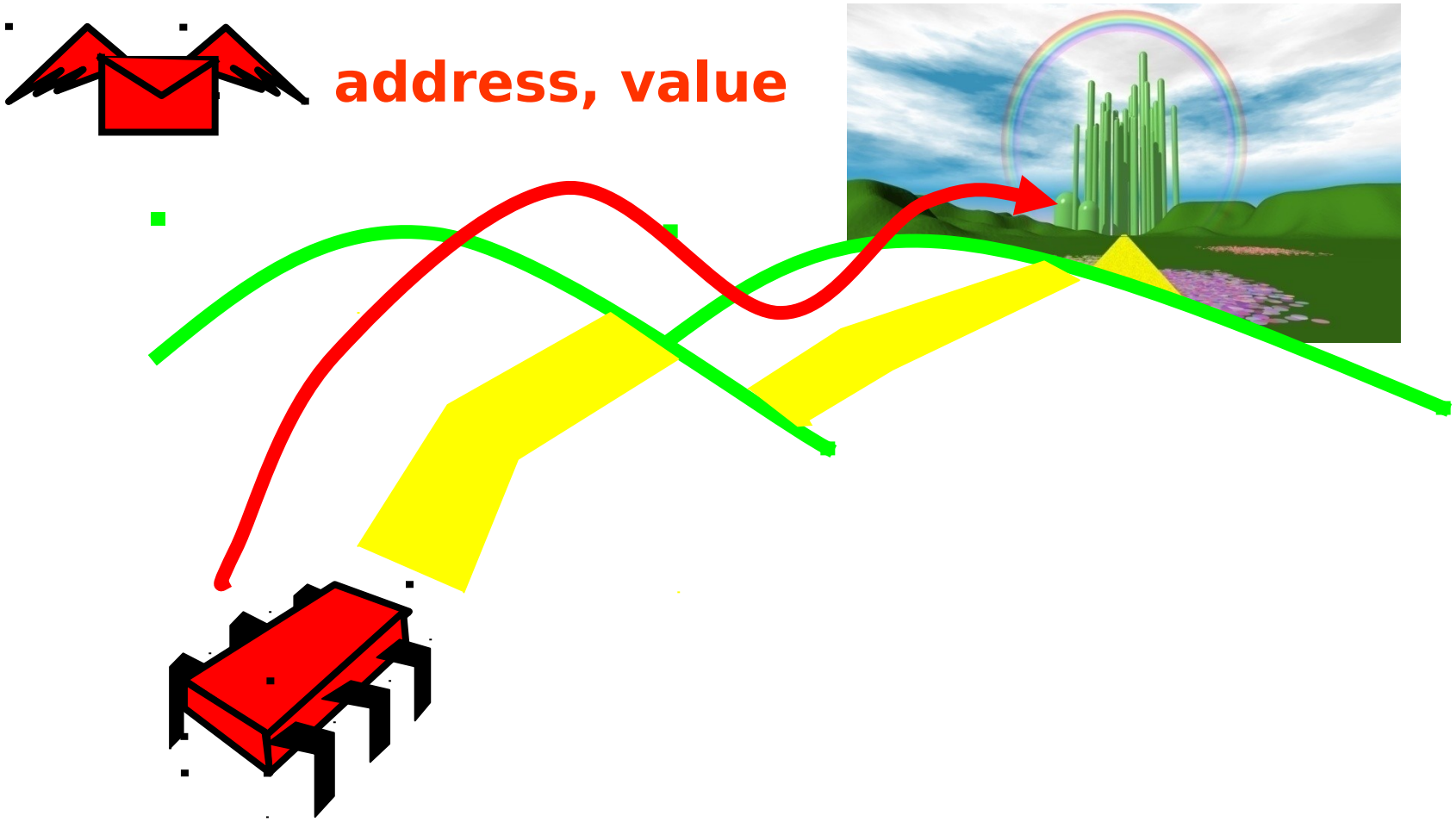
Reading from Memory



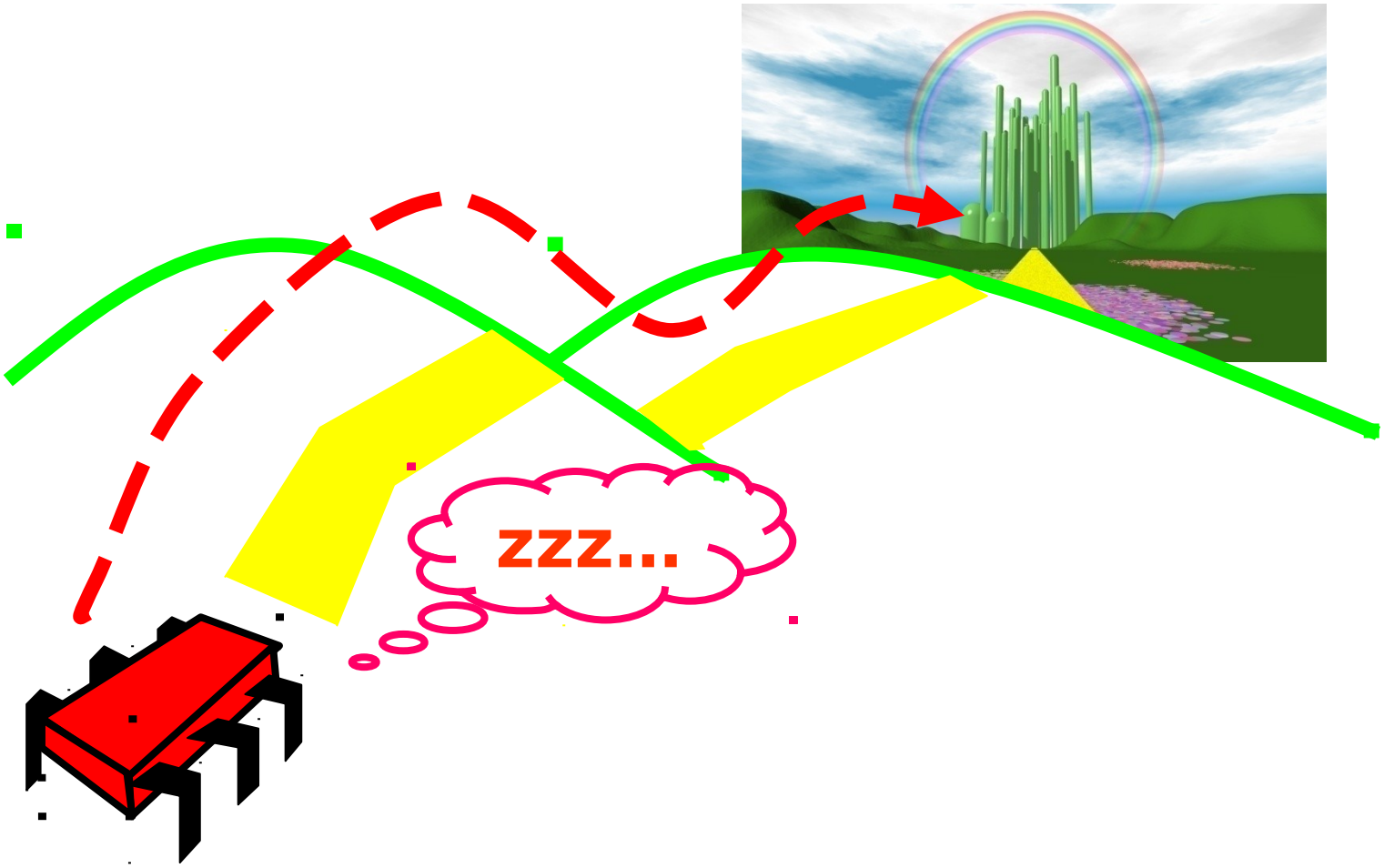
Reading from Memory



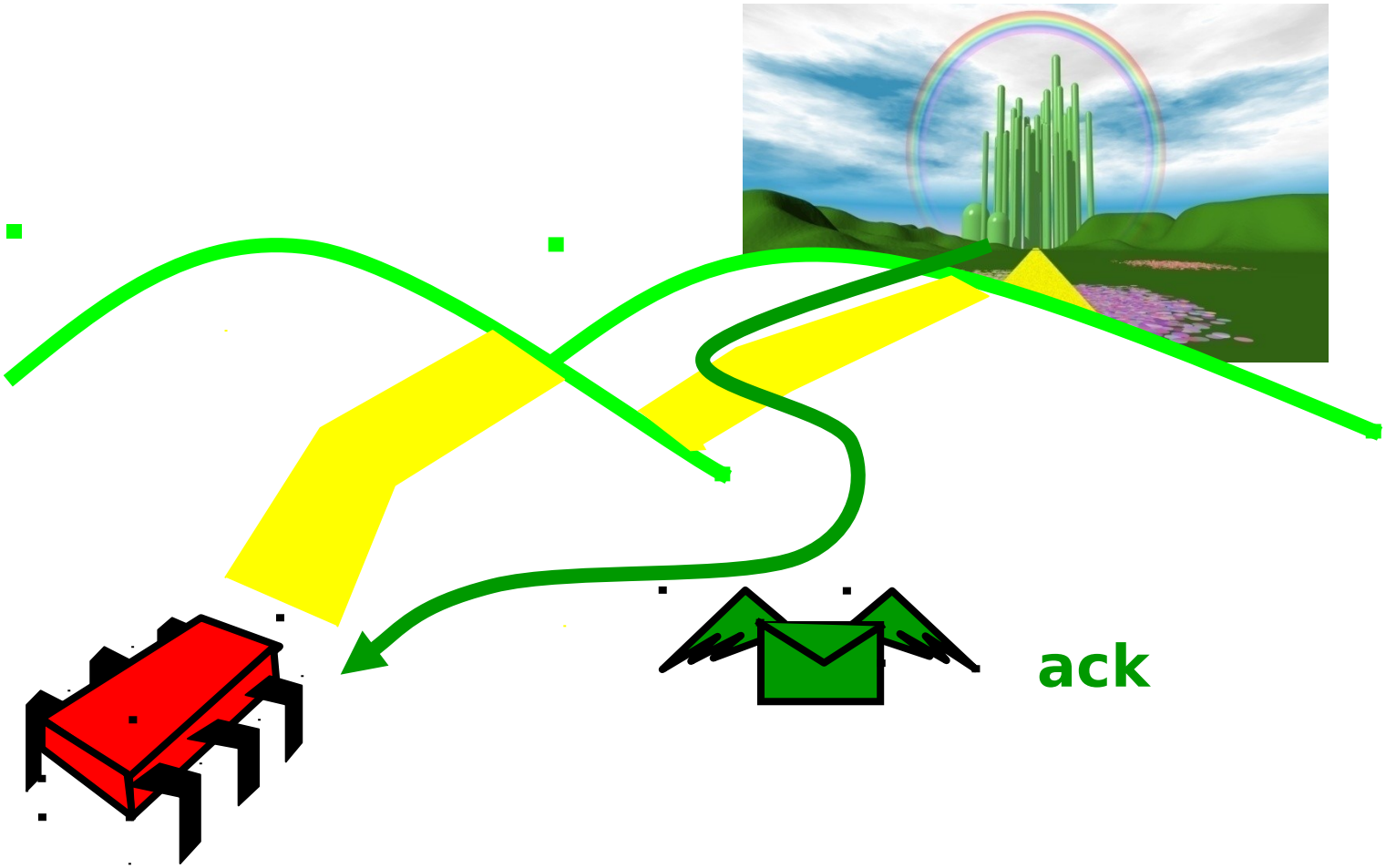
Writing to Memory



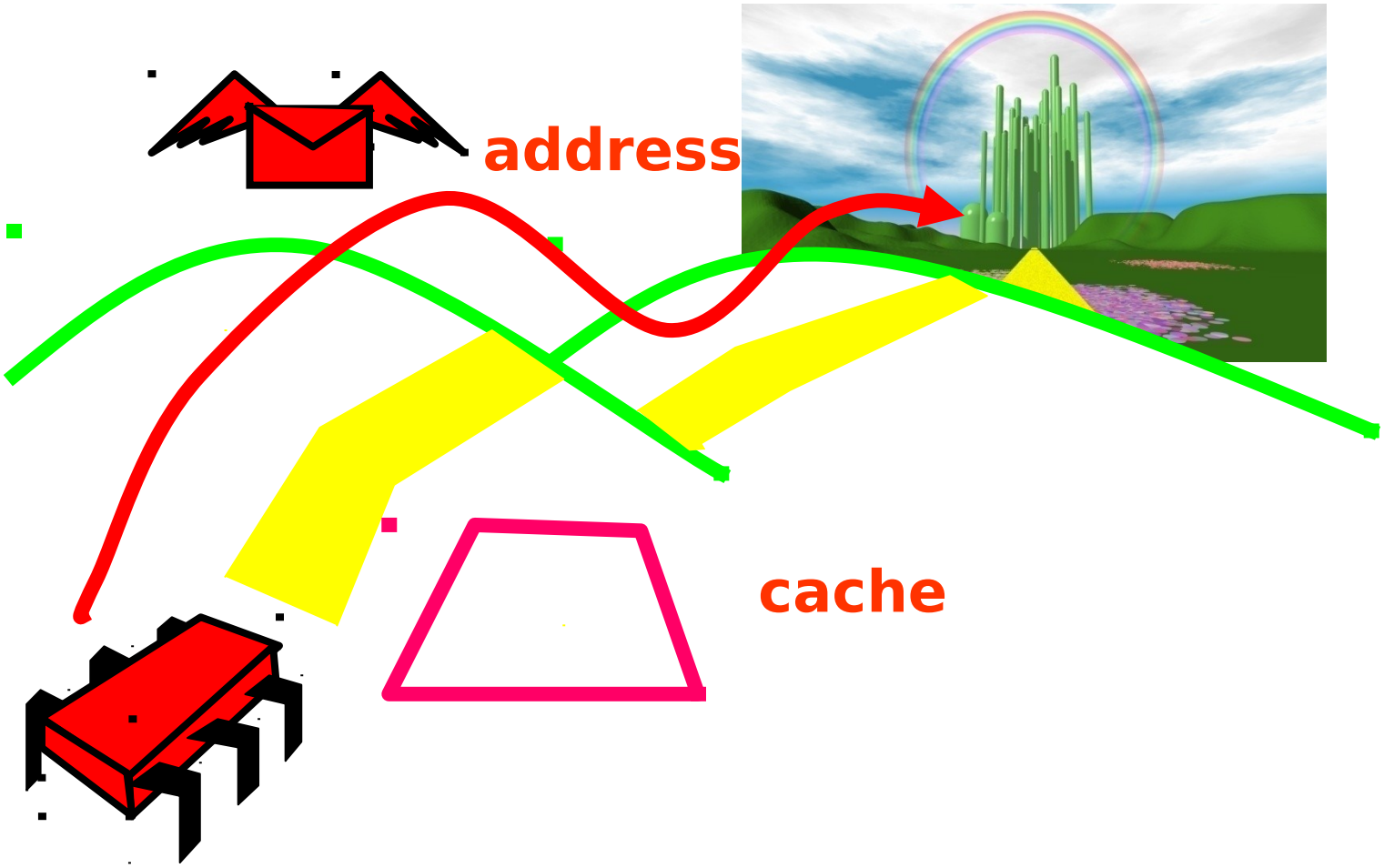
Writing to Memory



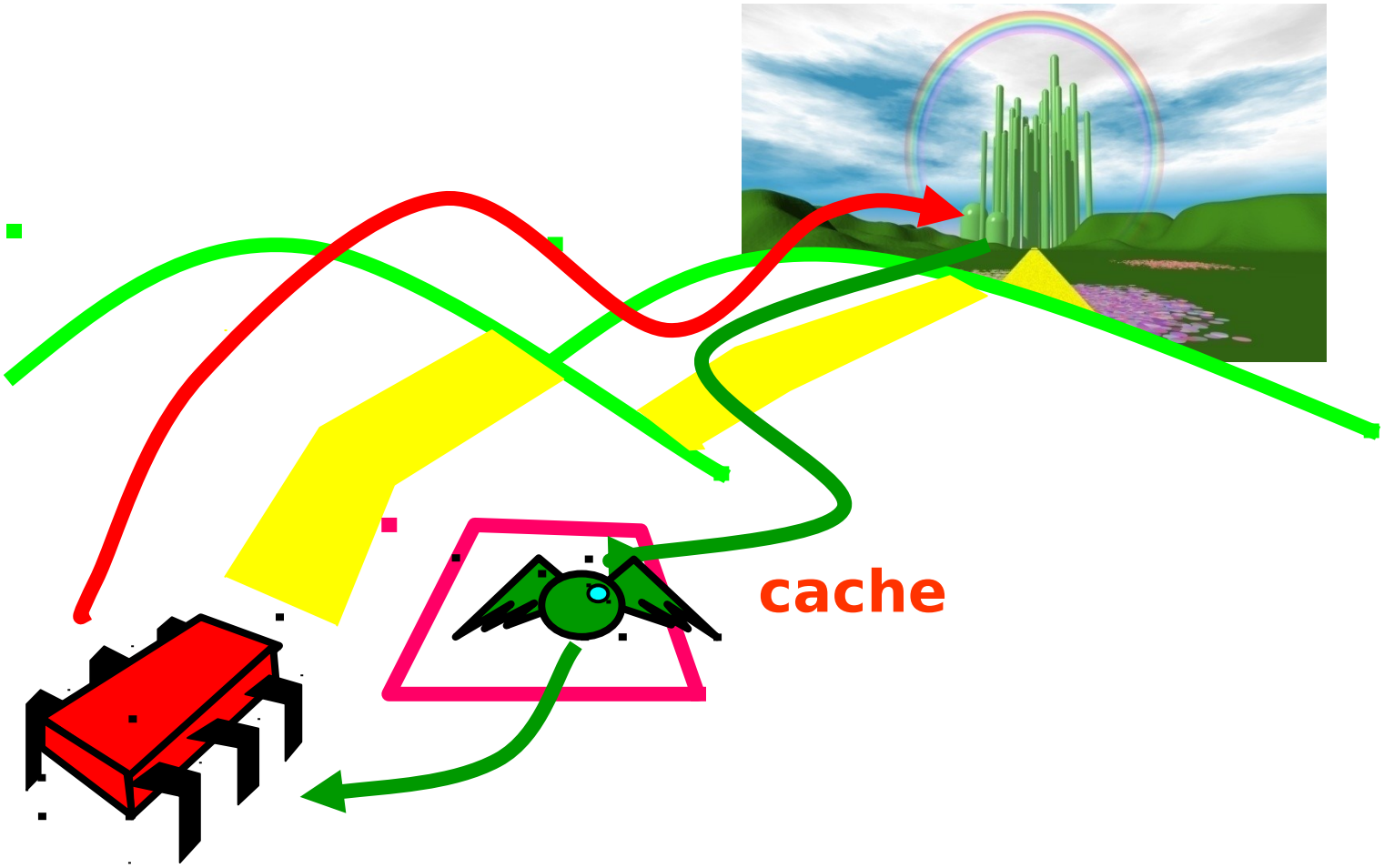
Writing to Memory



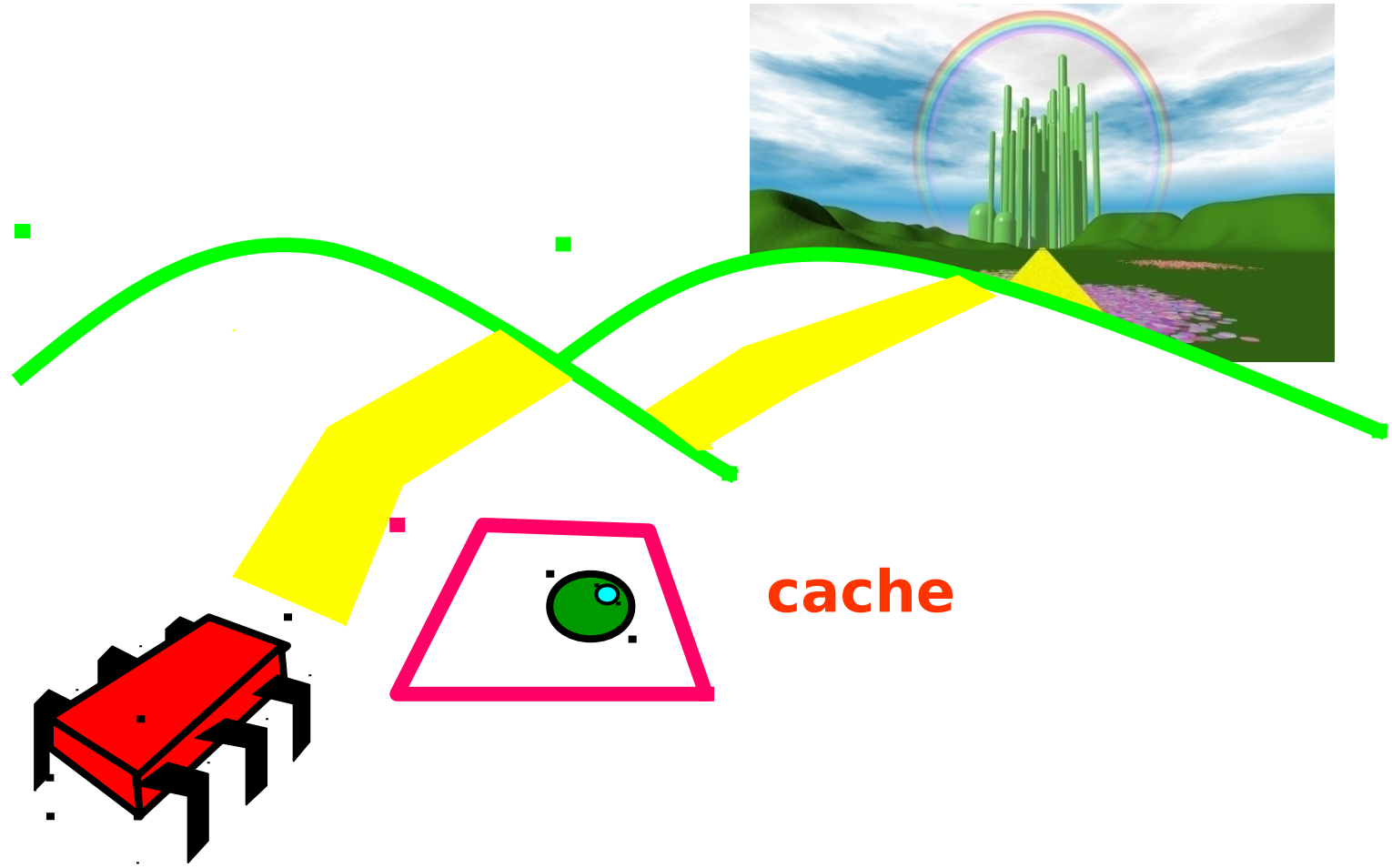
Cache: Reading from Memory



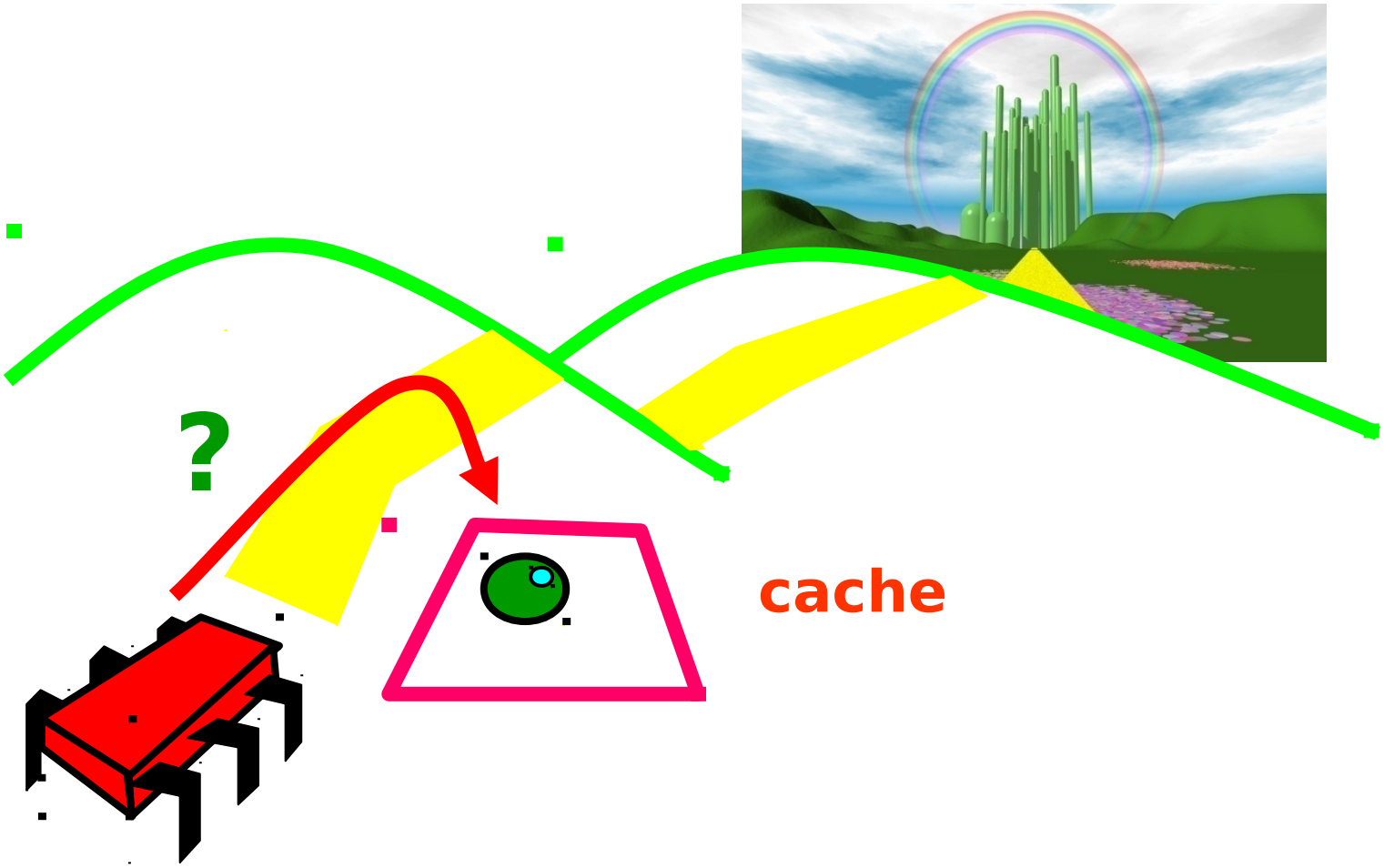
Cache: Reading from Memory



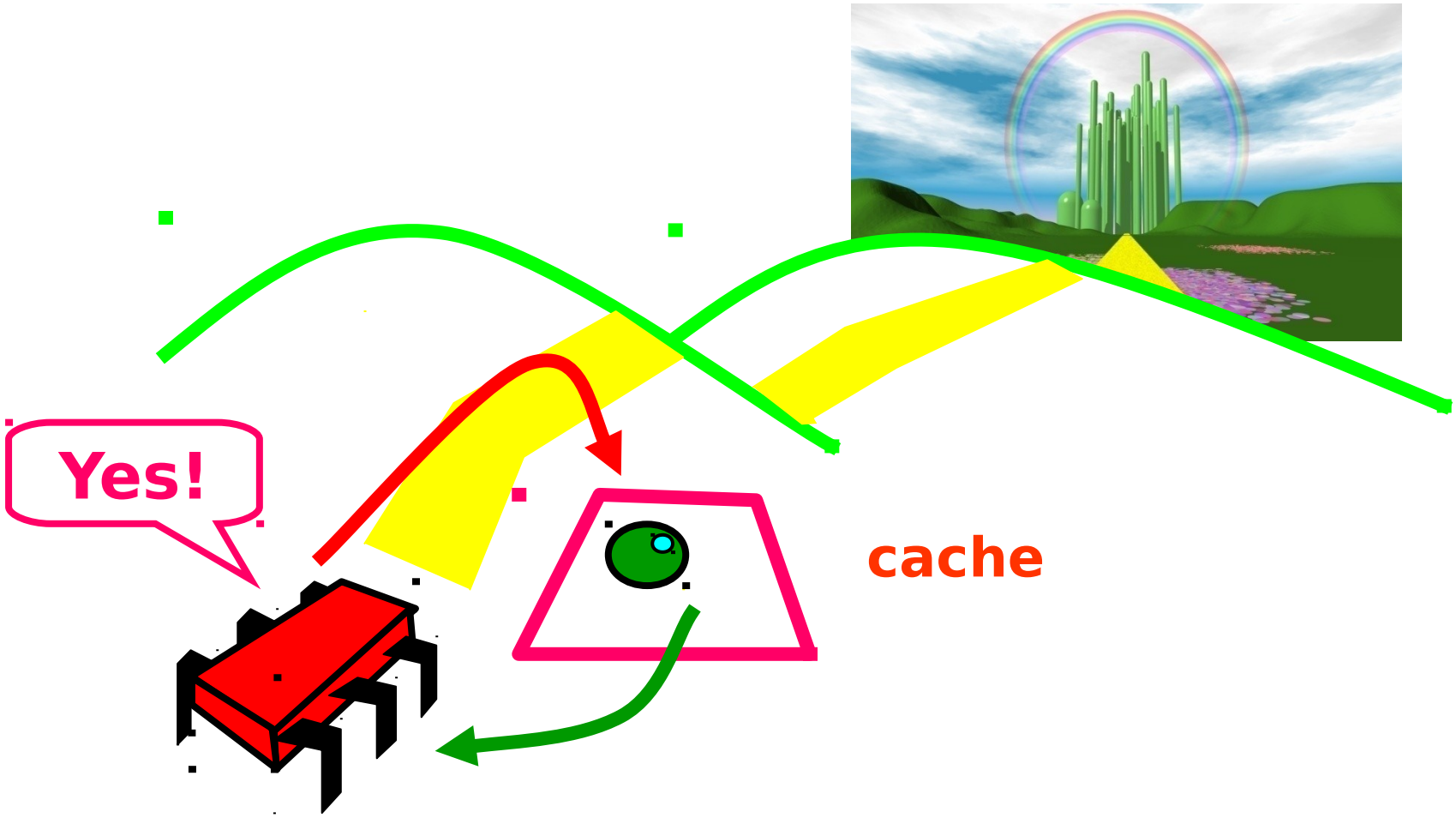
Cache: Reading from Memory



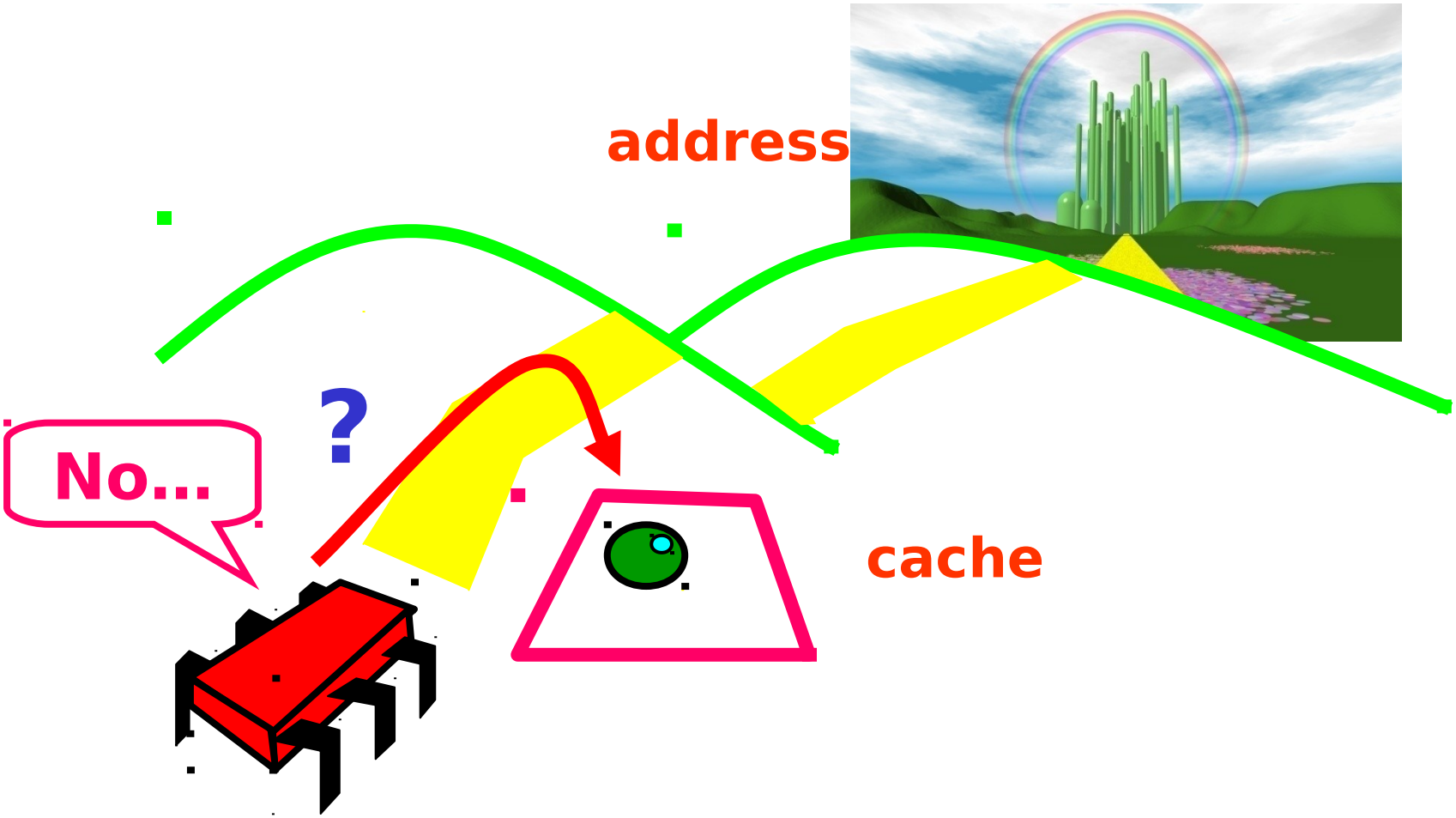
Cache Hit



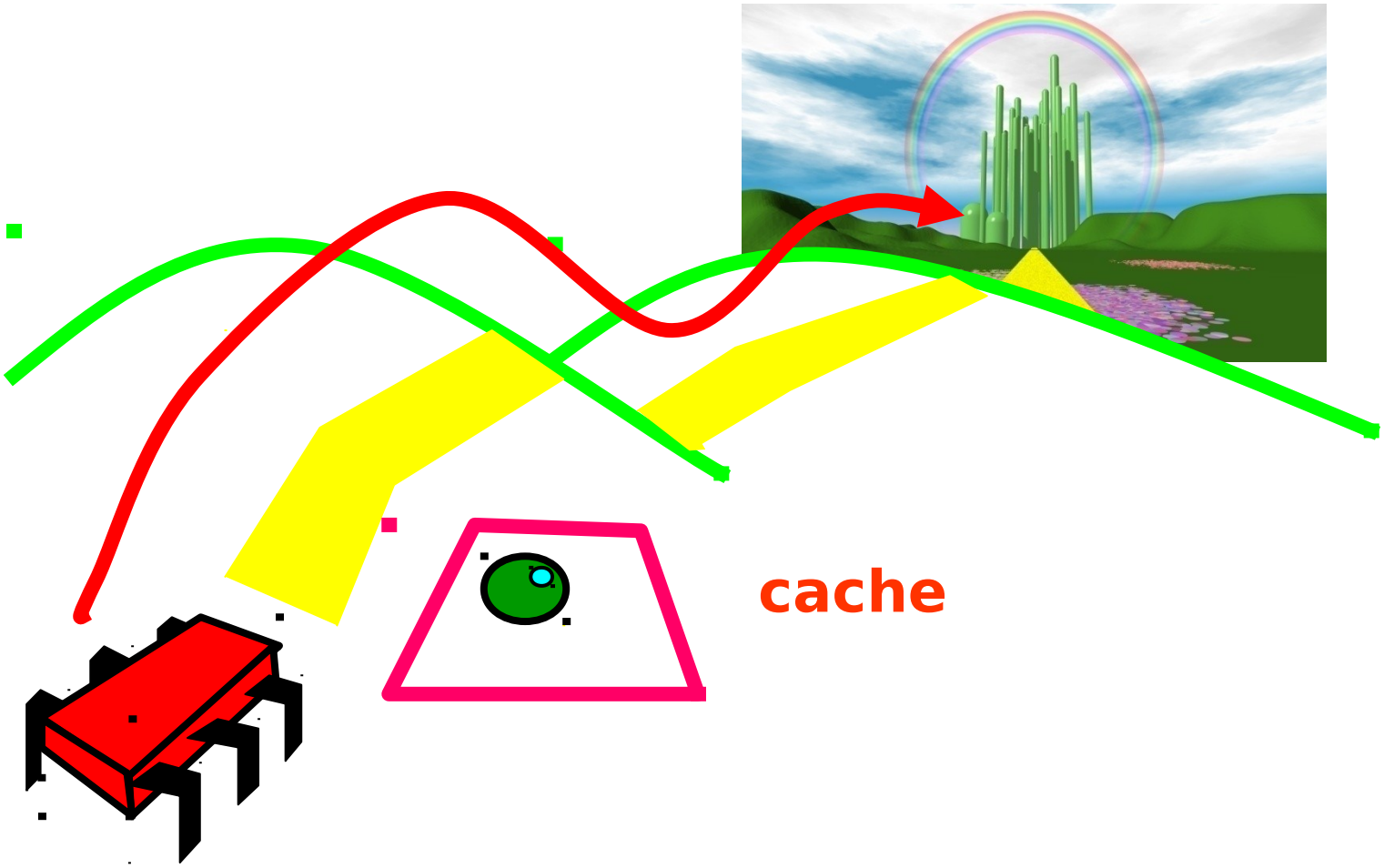
Cache Hit



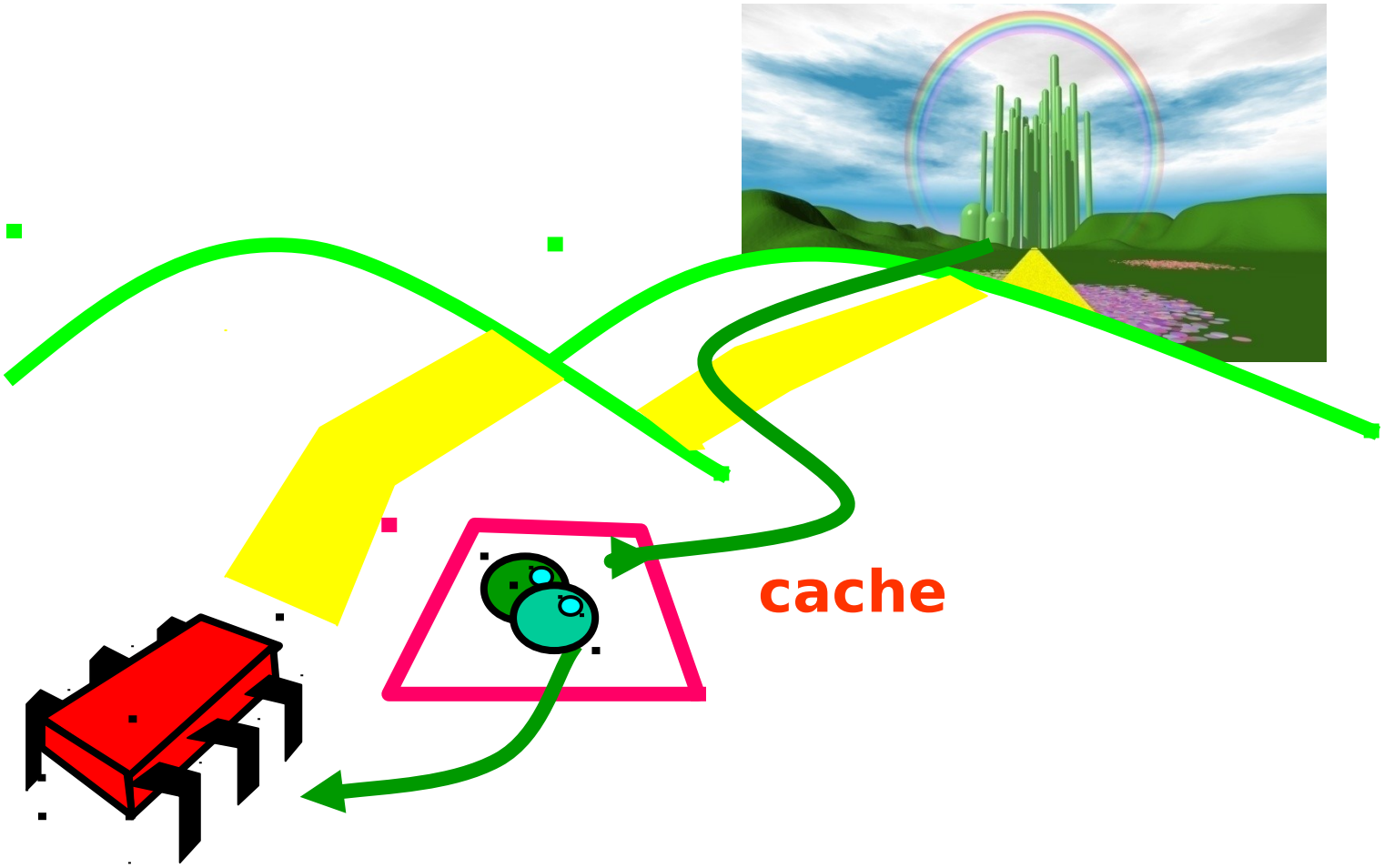
Cache Miss



Cache Miss



Cache Miss



Instruction Level Parallelism (ILP)

- **Parallelism at the machine-instruction level**
- **The processor can re-order, pipeline instructions, split them into microinstructions, do aggressive branch prediction, etc.**
- **Instruction-level parallelism enabled rapid increases in processor speeds over the last 15 years**

Instruction Level Parallelism (ILP)

Programme séquentiel

N'y aurait-il pas des instructions indépendante qui pourraient être exécutées en parallèle?

Comment générer de l'ILP?

- Pipe-line du processeur
  - Recouvrement d'exécution d'instructions
  - Limité par la divisibilité de l'instruction
- Superscalaire
  - Plusieurs unités fonctionnelles
  - Limité par le parallélisme intrinsèque du programme seq.

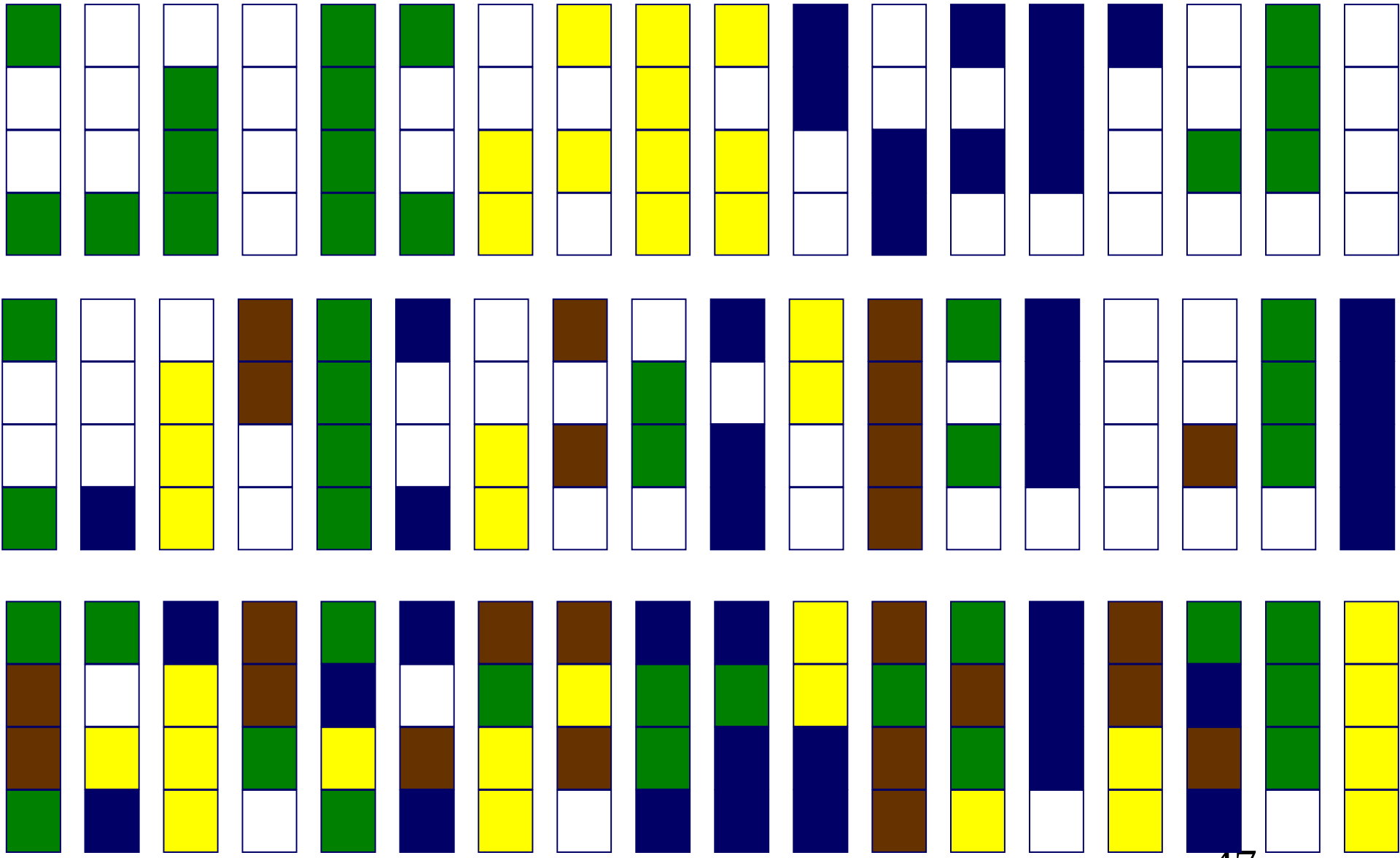
Comment accroître l'ILP?

- Prédictions sur les branchements conditionnels
- Réordonnancer les instructions (Out of Order Execution)

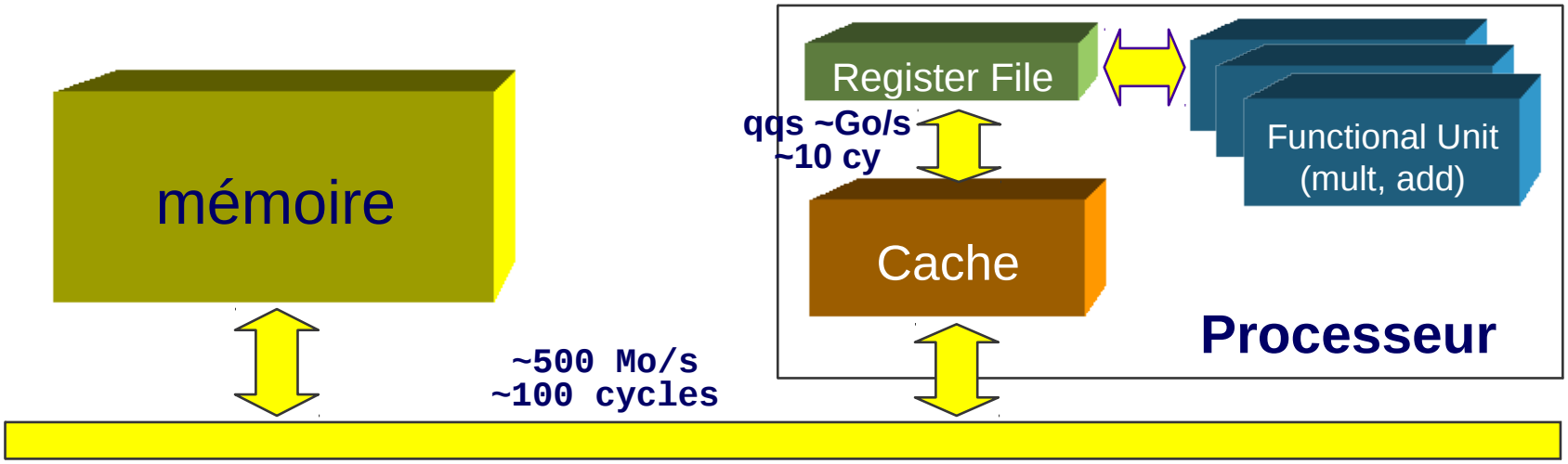
Recherche

- Domaines de l'Architecture-compilation

Simultaneous Multithreading (SMT), Hyperthreading



Scalar Architecture



Reduced Instruction Set (RISC) Architecture:

Les instructions load/store font référence à la mémoire

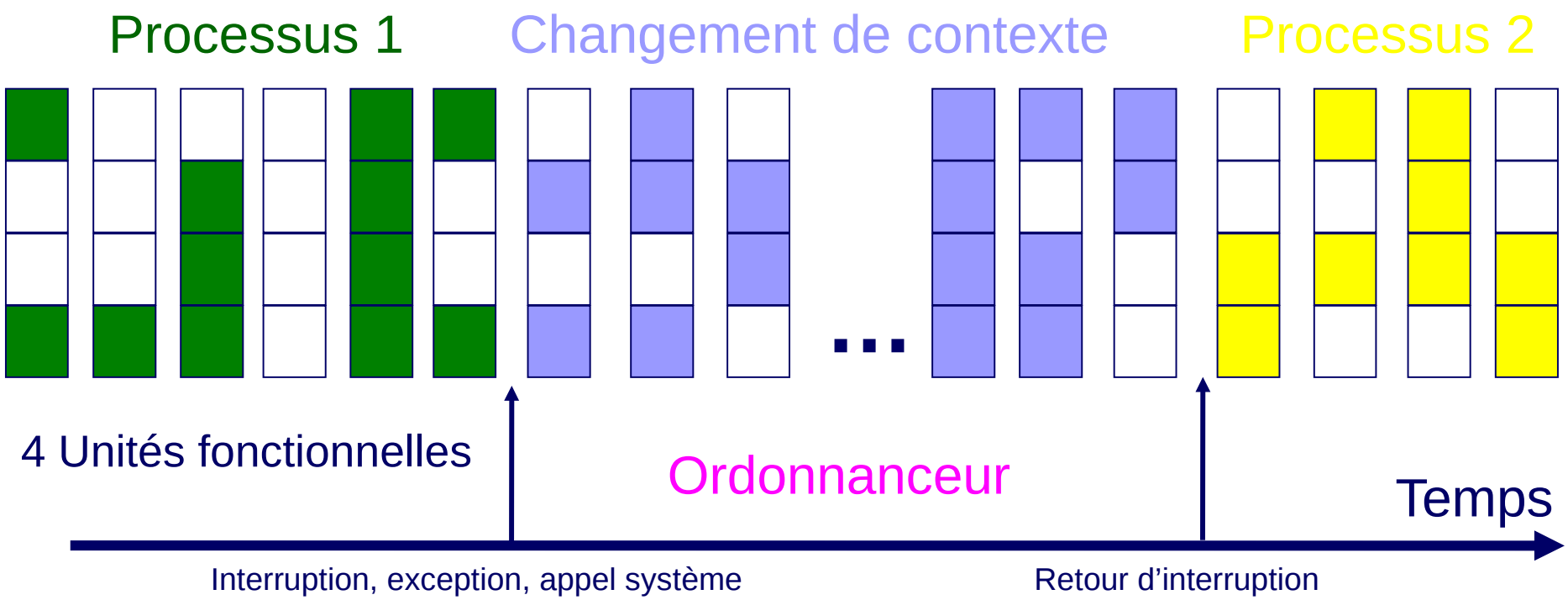
Les unités fonctionnelles travaillent sur des données stockées dans les registres

Hiérarchie mémoire dans une architecture scalaire :

Les éléments utilisés récemment sont copiés dans le cache,

Les accès au cache sont plus rapides que les accès à la mémoire.

Scalar Processor



Réduire le temps des changements de contexte (cases grisées-bleues)

Accroître l'utilisation des unités d'exécution (cases blanches)

Lightweight Process/Thread

Objectifs

- Mener plusieurs activités indépendantes au sein d'un processus
- Exploitation des architectures SMP
- Améliorer l'utilisation du processeurs (context-switch)

Exemples

- Simulations
- Serveurs de fichiers
- Systèmes d'exploitation (!)

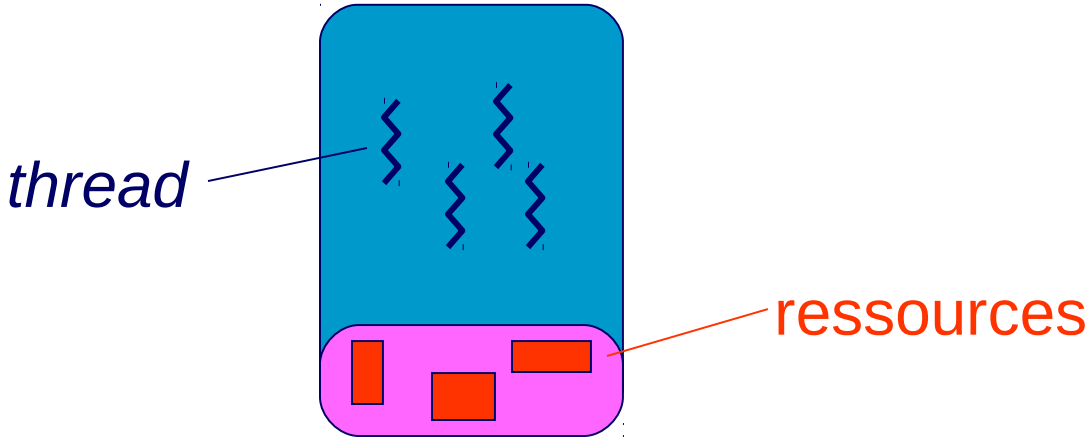
Solution sans l'aide du multithreading

- Automate à états finis implanté « à la main »
  (sauvegardes d'états)

Lightweight Process/Thread (2)

Principe

Détacher flot d'exécution et ressources



Introduits dans divers langages & systèmes

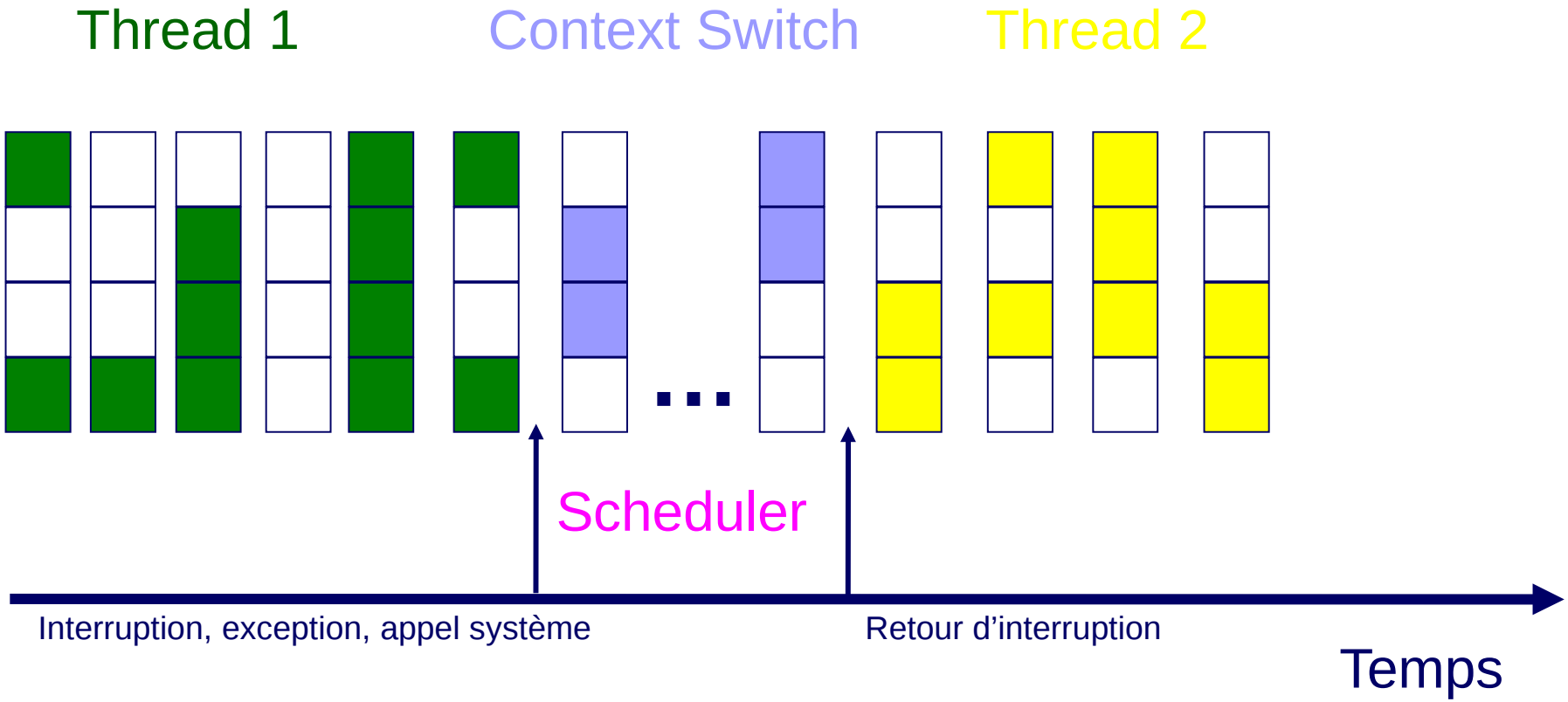
Programmation concurrente

Recouvrement des E/S

Exploitation des architectures SMP

Thread and Superscalar Processor

Processeur SuperScalaire



Evolution des Architectures

Phase 1 : Processeurs Multithreadés

Modification de l'architecture du processeur

- Incorporer au processeur deux (ou plus) jeux de registres pour les contextes des threads
  - Registres généraux
  - Program Counter (PC), registre d'instruction
  - Process Status Word (PSW), registre d'état
- A tout instant, un thread et son contexte sont actifs
- Changement du contexte courant instantané
  - Appel système, IT
  - Défaut de cache

Processeur IBM PowerPC RS 64

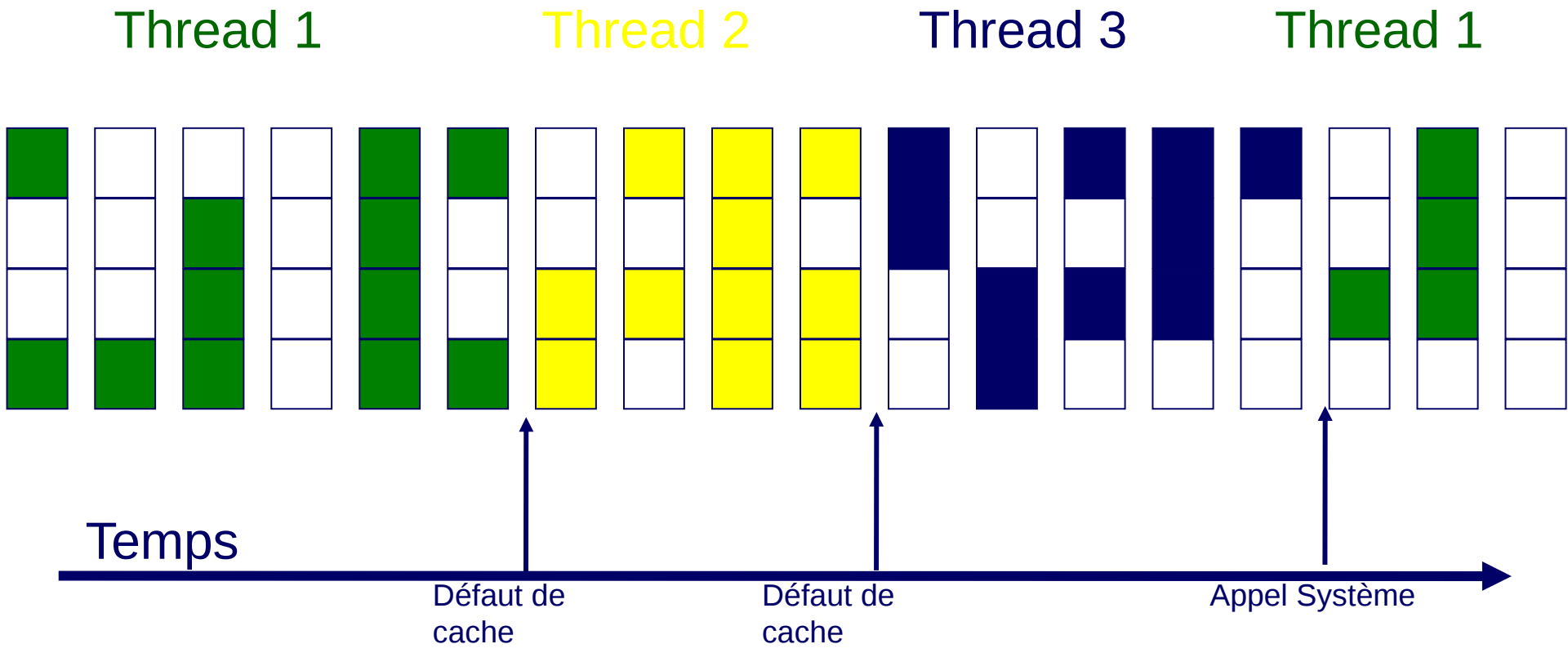
- Recherche, non commercialisé

Processeur Intel Xeon Hyperthreading

- Serveur Bi-processeur Xeon Nehalem Hyperthreadé
  - Vue de Linux ou Windows, 4 processeurs

Phase 1 : Processeurs Multithreadés (2)

Multithread à Gros Grain (Coarse-Grained Multi-threaded)



Phase 2 : Processeurs Multithreadés

Modification de l'architecture du processeur

Incorporer au processeur N jeux de registres pour les contextes des threads

- Registres généraux
- Program Counter (PC), registre d'instruction
- Process Status Word (PSW), registre d'état

A tout instant, un thread et son contexte sont actifs

Changement de contexte à chaque cycle

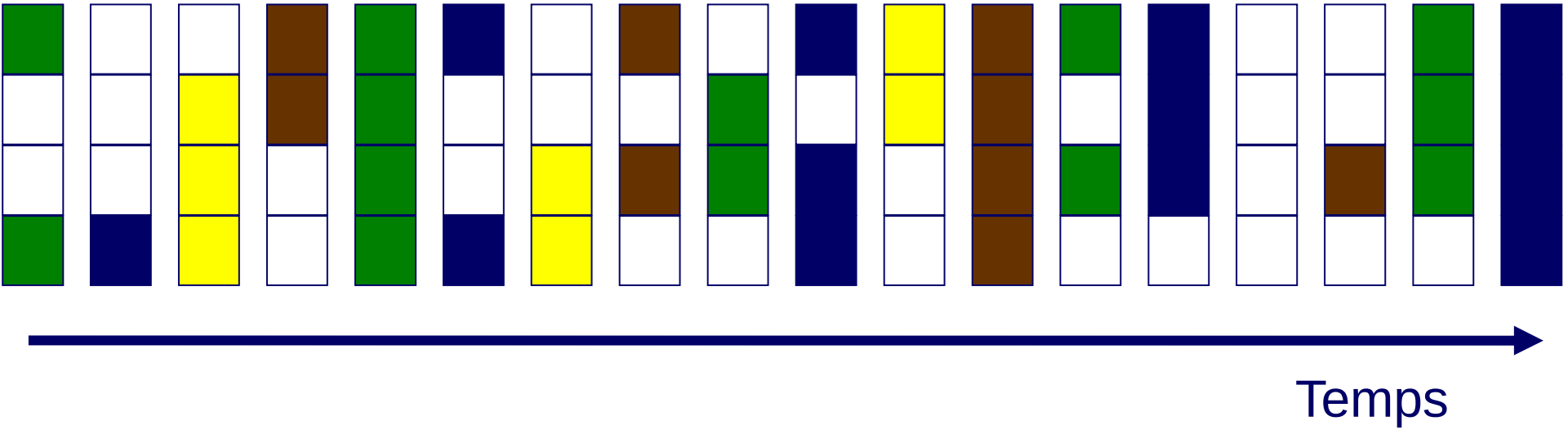
- Chaque thread dispose de son ratio du processeur (1/N)

Processeur TERA

Phase 2 : Processeurs Multithreadés (2)

Multithread à Grain Fin (Fine-Grained Multi-threaded)

4 registres de threads : ¼ temps processeur par thread



Temps

Thread 1 Thread 2 Thread 3 Thread 4

Phase 3 : Processeurs Multithreadés

Modification de l'architecture du processeur

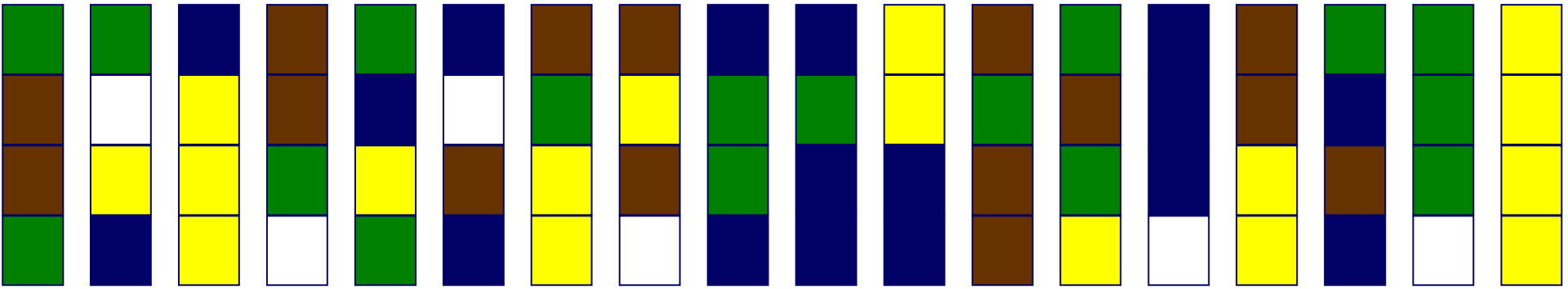
- Incorporer au processeur N jeux de registres pour les contextes des threads
  - Registres généraux
  - Program Counter (PC), registre d'instruction
  - Process Status Word (PSW), registre d'état
- A un instant donné, les unités du processeurs peuvent être partagées entre plusieurs threads

Eviter de stresser les mêmes ressources

- Conflit d'accès à certaines unités d'exécution

Processeurs ALPHA EV8, CRAY-TERA

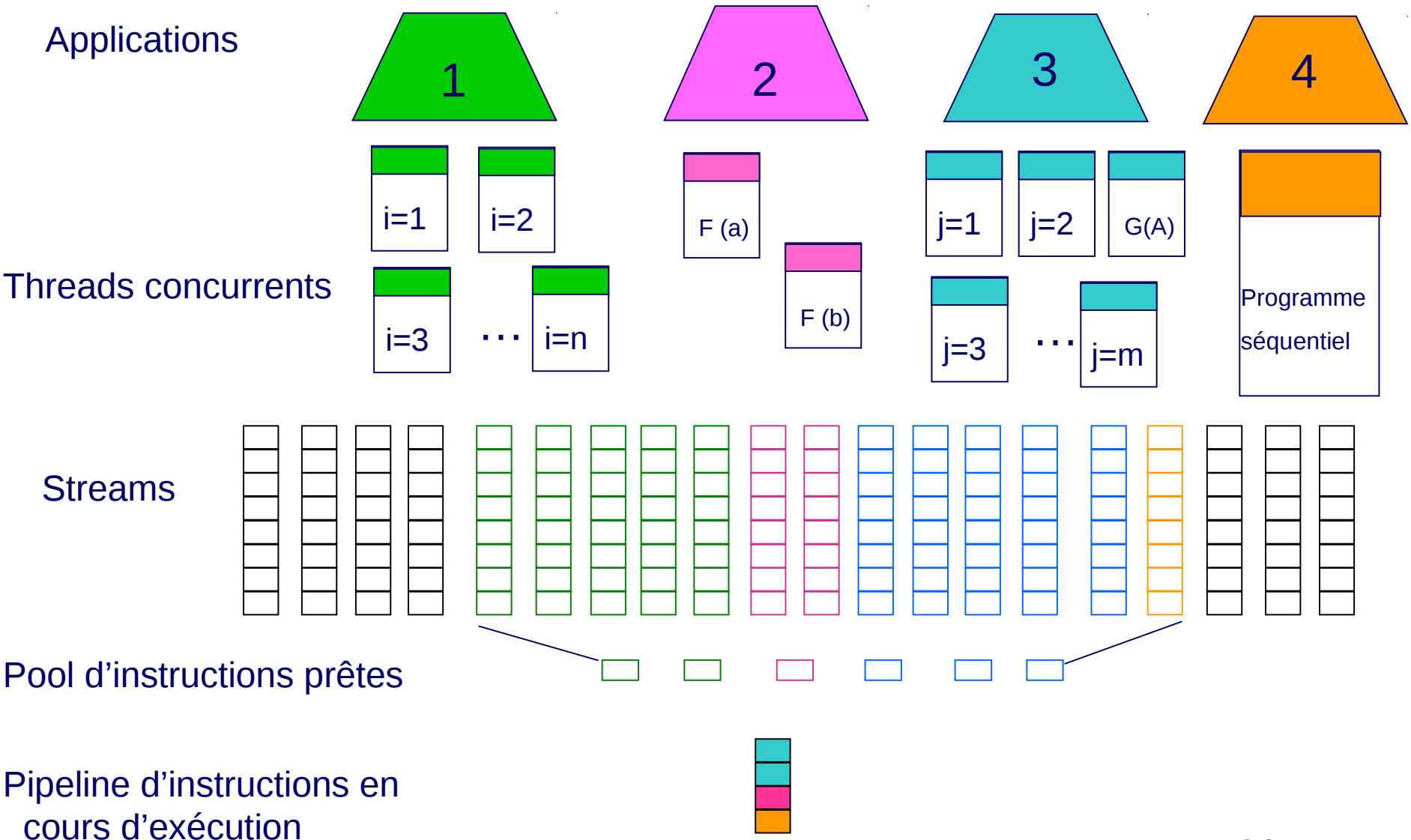
Phase 3 : Processeurs Multithreadés (3)



Thread 1 Thread 2 Thread 3 Thread 4

Temps

Processeur Cray-TERA

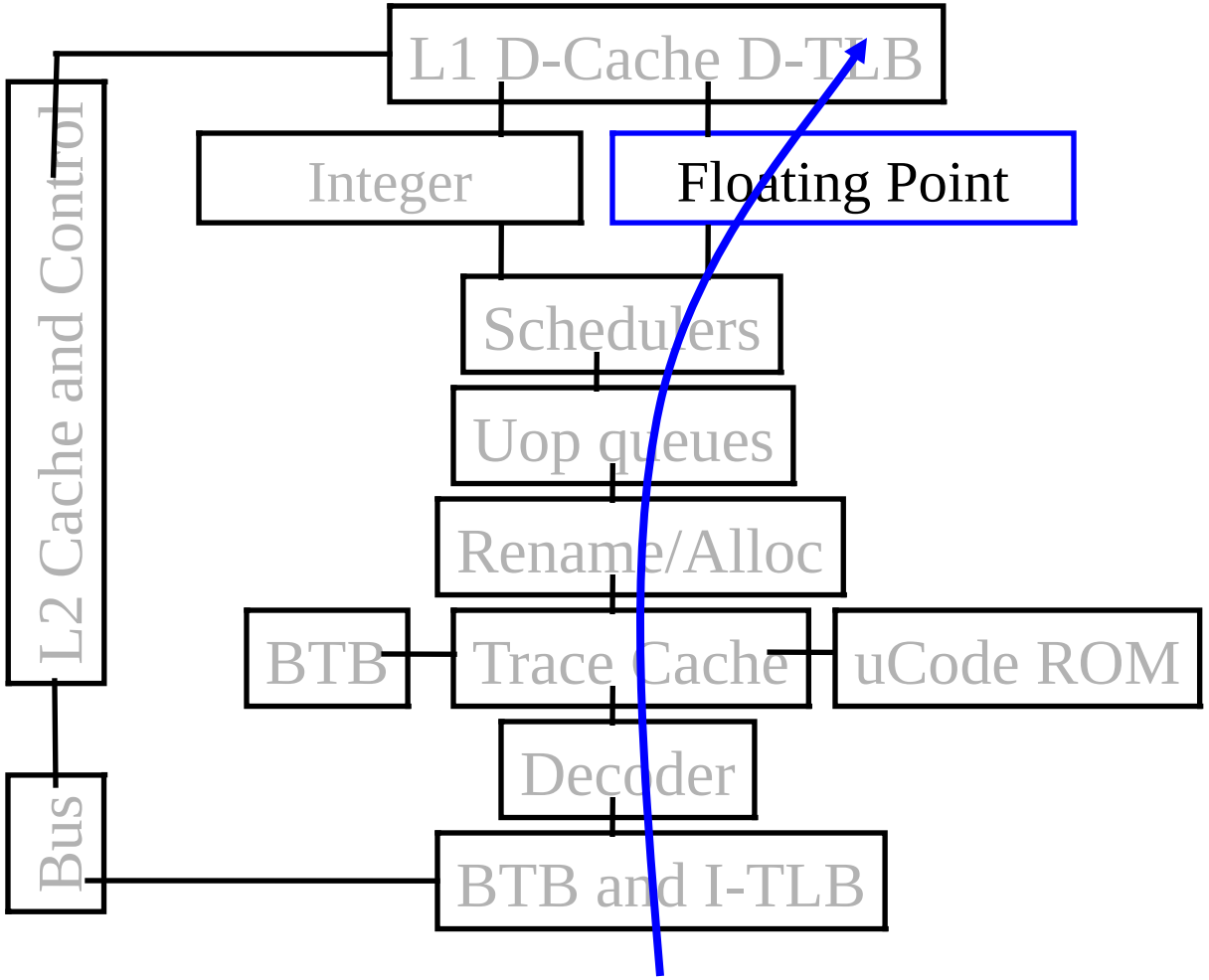


Simultaneous Multithreading (SMT)

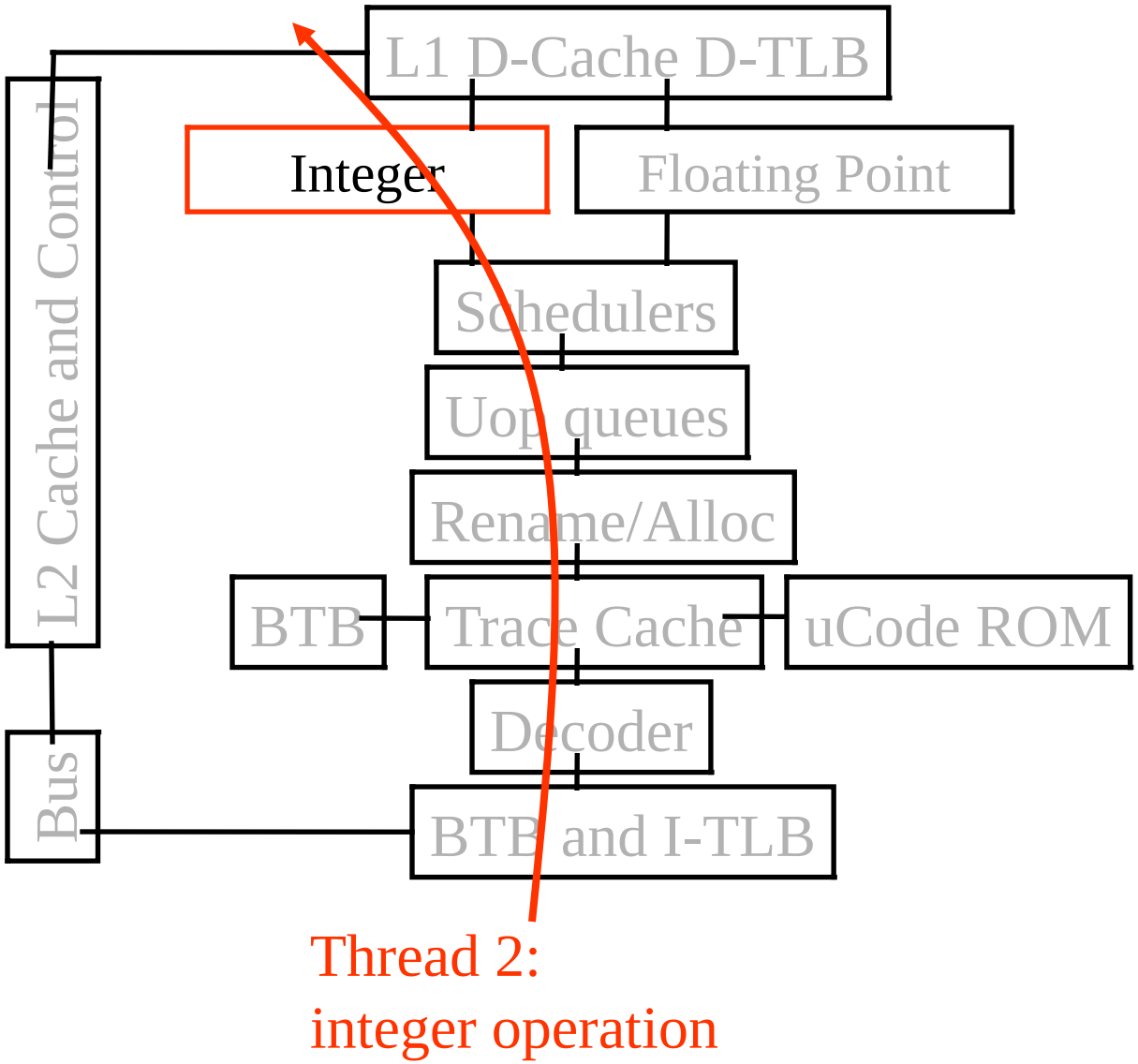
- **Permits multiple independent threads to execute SIMULTANEOUSLY on the SAME core**

- **Weaving together multiple "threads" on the same core:**
  - **Exemple: If one thread is waiting for a floating point operation to complete, another thread can use the integer units**

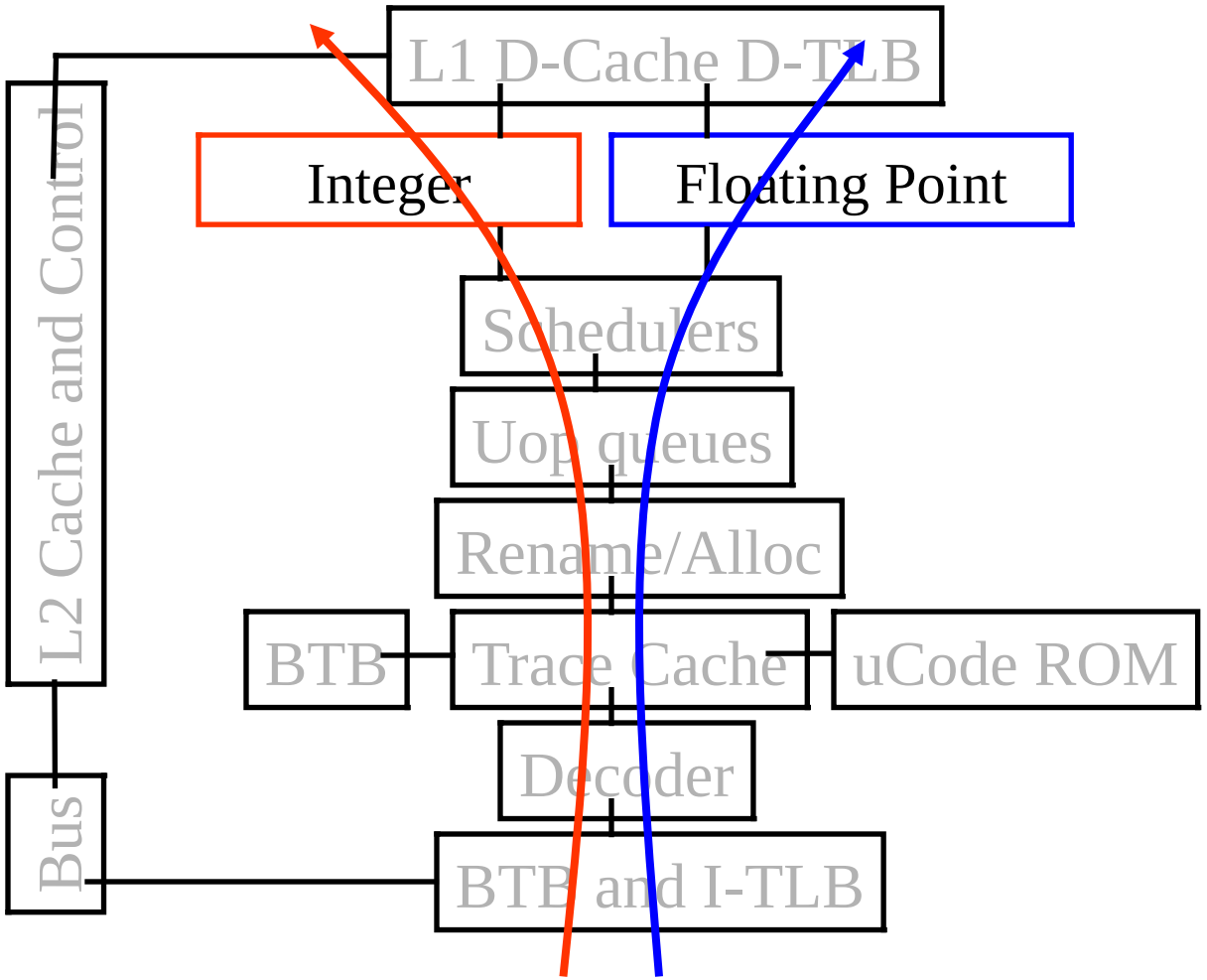
Without SMT, only a single thread can run at any given time



Without SMT, only a single thread can run at any given time



SMT processor: both threads can run concurrently



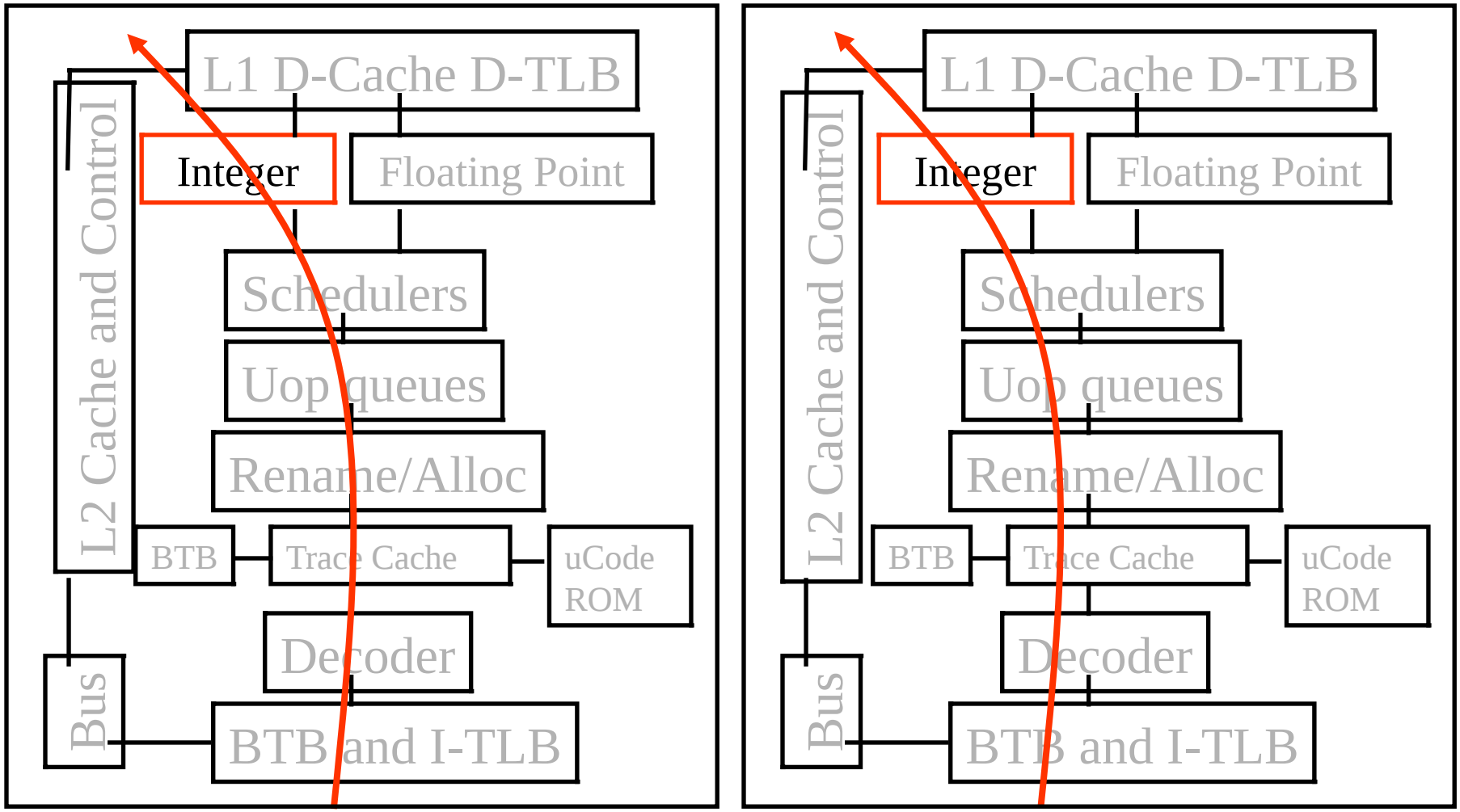
Thread 2: integer operation

Thread 1: floating point

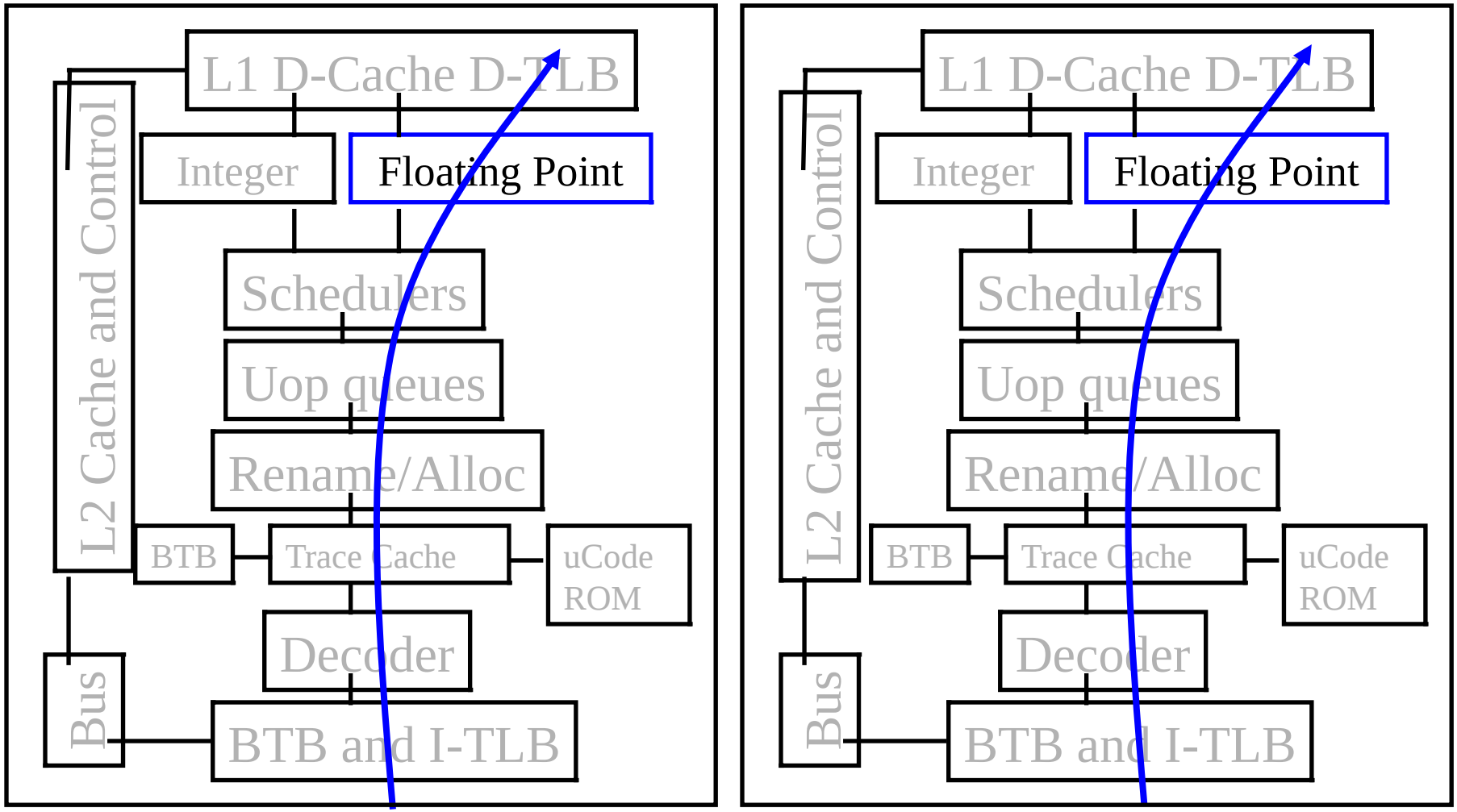
SMT not a « true » parallel processor

- Enables better threading (e.g. up to 30%)
- OS and applications perceives each simultaneous thread as a separate “virtual processor”
- The chip has only a single copy of each resource (functional units, pipeline,...)
- Compare to multi-core: each core has its own copy of resources

Multi-Core : threads can run on separate cores



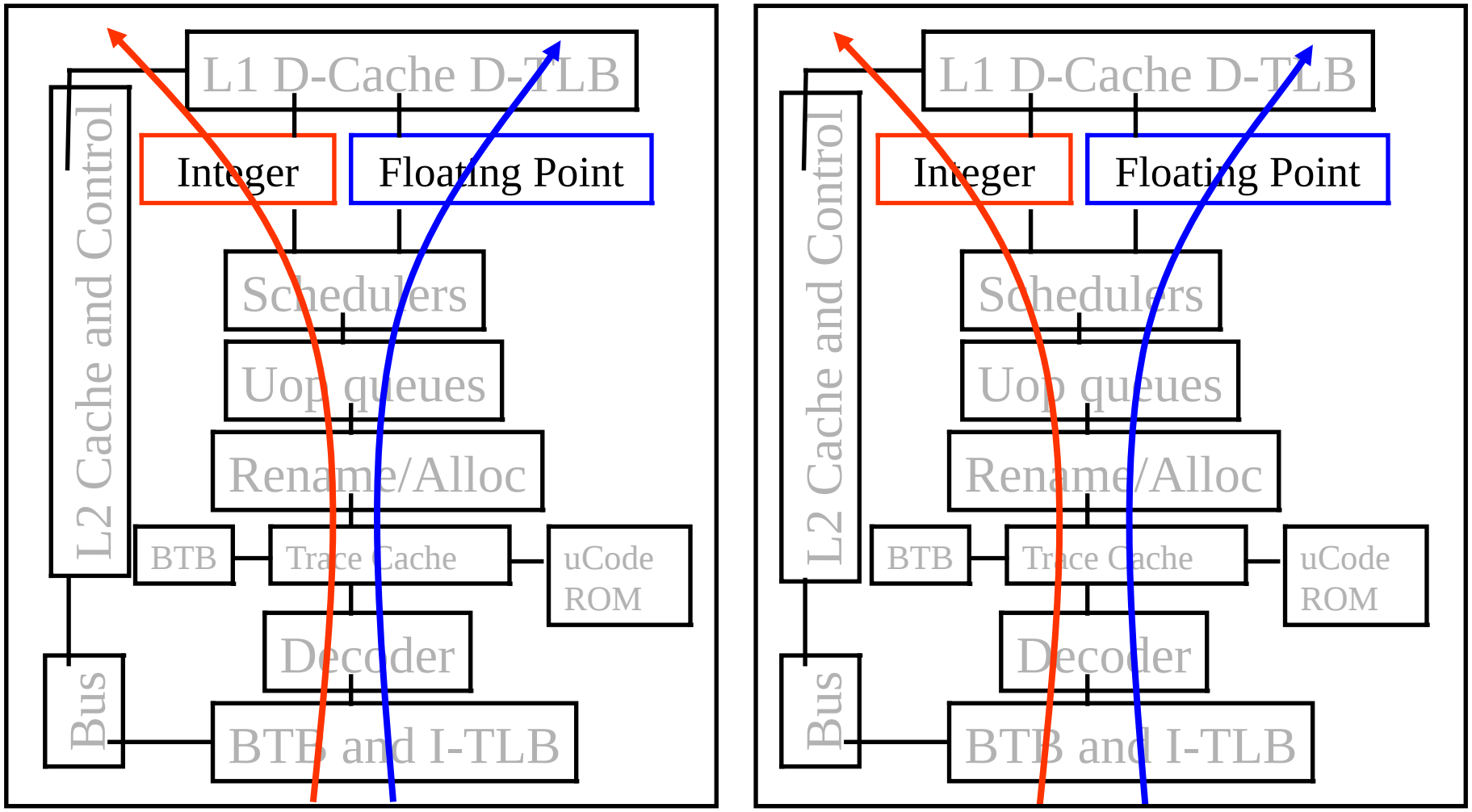
Multi-Core : threads can run on separate cores



Combining Multi-core and SMT

- Cores can be SMT-enabled (or not)
- The different combinations
  - Single-core, non-SMT (standard uniprocessor)
  - Single-core, with SMT
  - Multi-core, non-SMT (ARM, AMD,...)
  - Multi-core, with SMT (Intel, IBM,...)
- The number of SMT threads: 2, 4 or sometimes 8 Simultaneous Threads
- Intel called them “Hyper Threads” (HT technology)

SMT Dual-core : all four threads can run concurrently



Comparison : multi-core vs SMT

- **Multi-core**
  - **Since there are core, each is smaller and not as powerful (but also easier to design and manufacture)**
  - **However, great with thread-level parallelism**

- **SMT**
  - **Can have one large and fast superscalar core**
  - **Great performance on a single thread**
  - **Mostly still only exploit instruction level parallelism**

The memory hierarchy for threading

- **If simultaneous multithreading only:**
  - **All caches shared**
- **Multi-core chips**
  - **L1 private caches**
  - **L2 private caches in some architectures and shared in other**
  - **L3 shared caches (Last Level Cache on Intel)**
- **Memory is always shared**

Private vs shared caches

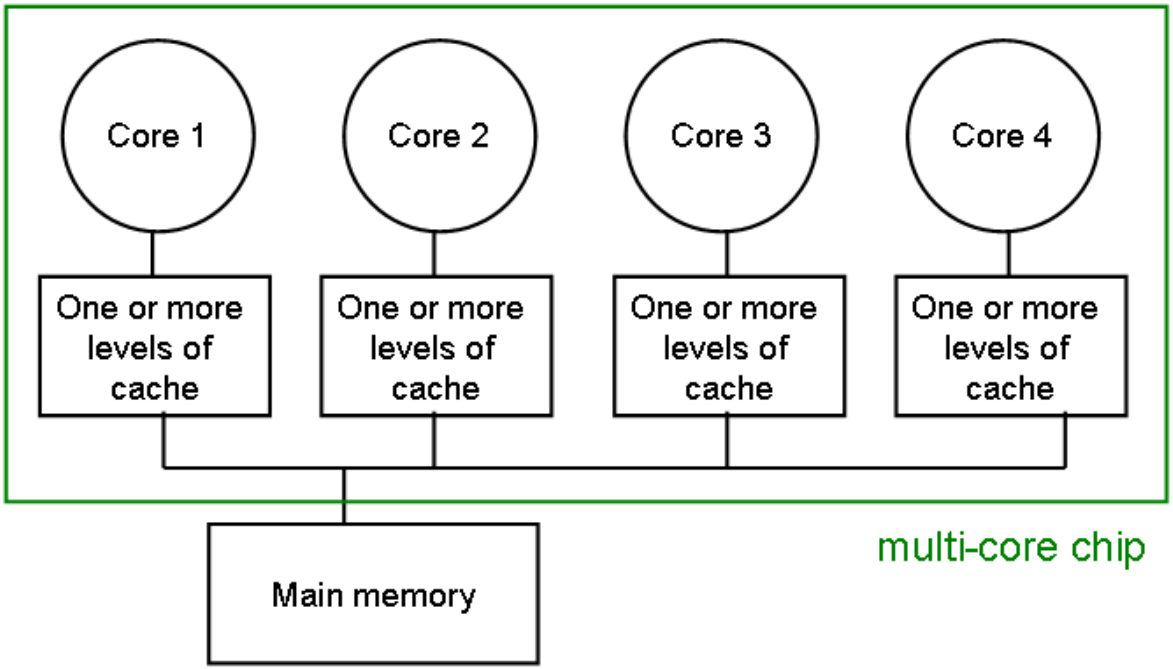
- **Advantages of private caches**
  - **They are closer to core, so faster access**
  - **No contention**
- **Advantages of shared caches**
  - **Threads on different cores can share the same cache data**
  - **More cache space available if a single (or few) intensive computing thread runs on the system**

The cache coherence problem

- **Since we have private caches**
  - **How to keep the data consistent across the caches**

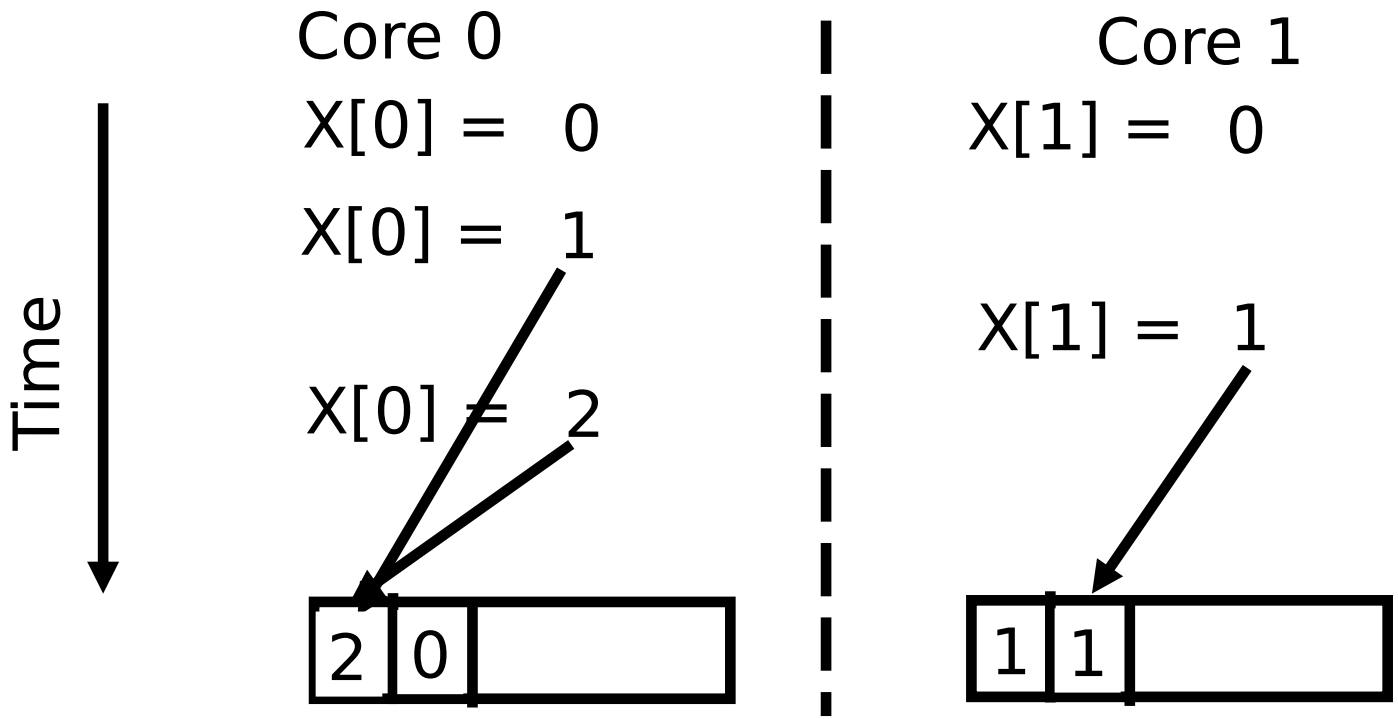
- **Each core should perceive the memory as a monolithic array, shared by all cores**

MESI cache Coherence Protocol



False sharing

- **Performance issue in programs where core may write to different memory addresses BUT in the same cache lines**
  - **Know as Ping-Ponging – Cache line is shipped between cores**



- False sharing is not an issue for shared caches
- It is an issue in private cache

Why Parallelism ?

- These arguments are no long theoretical
- All major processor vendors are producing multicore chips
  - Every machine is a parallel machine
  - All programmers will be parallel programmers???
- New software model
  - Want a new feature? Hide the “cost” by speeding up the code first
  - All programmers will be performance programmers???
- Some may eventually be hidden in libraries, compilers, and high level languages
  - But a lot of work is needed to get there
- Big open questions:
  - What will be the killer apps for multicore machines
  - How should the chips be designed, and how will they be programmed?

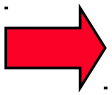
Outline

- Why powerful computers must be parallel processors
  - all
  - Including your laptop
- Why writing (fast) parallel programs is hard
- Principles of parallel computing performance
- Structure of the course

Why writing (fast) parallel programs is hard

Principles of Parallel Computing

- **Finding enough parallelism (Amdahl's Law)**
- **Granularity**
- **Locality**
- **Load balance**
- **Coordination and synchronization**
- **Performance modeling**



All of these things makes parallel programming even harder than sequential programming.

Finding Enough Parallelism

- **Suppose only part of an application seems parallel**
- **Amdahl's law**
  - **let s be the fraction of work done sequentially, so (1-s) is fraction parallelizable**
  - **P = number of processors**

$$\text{Speedup}(P) = \text{Time}(1)/\text{Time}(P)$$

$$<= 1/(s + (1-s)/P)$$

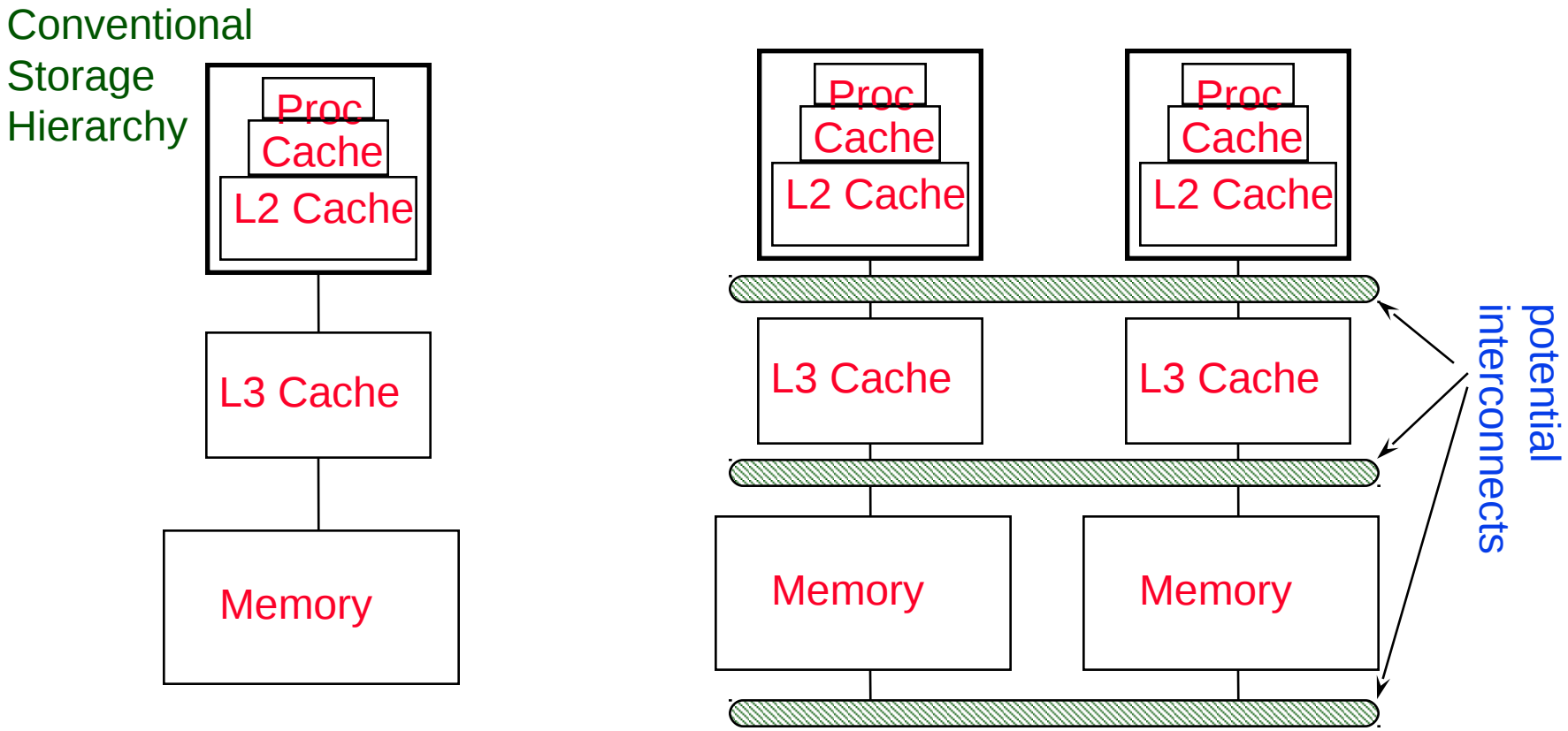
$$<= 1/s$$

- Even if the parallel part speeds up perfectly performance is limited by the sequential part

Overhead of Parallelism

- Given enough parallel work, this is the biggest barrier to getting desired speedup
- Parallelism overheads include:
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - extra (redundant) computation
- Each of these can be in the range of milliseconds (=millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (I.e. large granularity), but not so large that there is not enough parallel work

Locality and Parallelism



- **Large memories are slow, fast memories are small**
- **Storage hierarchies are large and fast <u>on average</u>**
- **Parallel processors, collectively, have large, fast cache**
  - **the slow accesses to "remote" data we call "communication"**
- **Algorithm should do most work on local data**

Load Imbalance

- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase)
  - unequal size tasks
- Examples of the latter
  - adapting to “interesting parts of a domain”
  - tree-structured computations
  - fundamentally unstructured problems
- Algorithm needs to balance load

Reading Materials

- **Programming Many-Core Chips, Andras Vajda (Ericson), Springer, 2011**
- **Ian Foster's book, "Designing and Building Parallel Programming".**
  - http://www-unix.mcs.anl.gov/dbpp/