Threading Concepts

Thread

From Wikipedia, the free encyclopedia

A thread in computer science is short for a **thread of execution**. Threads are a **way for a program to split itself into two or more simultaneously** (or pseudo-simultaneously) **running tasks**. Threads and processes differ from one operating system to another, but in general, the way that a thread is created and shares its resources is different from the way a process does.

Multiple threads can be executed in parallel on many computer systems. This multithreading generally occurs by **time slicing**, wherein **a single processor switches between different threads**, in which case the processing is not literally simultaneous, for the single processor is only really doing one thing at a time. **This switching can happen so fast as to give the illusion of simultaneity to an end user**. For instance, a typical PC today contains only one processor core, but you can run multiple programs at once, such as a word processor alongside an audio playback program; though the user experiences these things as simultaneous, in truth, **the processor is quickly switching back and forth between these separate processes**. On a multiprocessor system, threading can be achieved via multiprocessing, wherein **different threads and processes can run literally simultaneously on different processors**.

Threads compared with processes

Threads are distinguished from traditional multi-tasking operating system processes in that **processes are typically independent, carry considerable state information, have separate address spaces, and interact only through system-provided inter-process communication mechanisms**. **Multiple threads, on the other hand, typically share the state information of a single process, and share memory and other resources directly**. Context switching between threads in the same process is typically faster than context switching between processes. Systems like Windows NT and OS/2 are said to have "cheap" threads and "expensive" processes; in other operating systems there is not so great a difference.

Multithreading is a popular programming and execution model that allows multiple threads to exist within the context of a single process, sharing the process' resources but able to execute independently. The threaded programming model provides developers with a useful abstraction of concurrent execution. However, perhaps the most interesting application of the technology is when it is applied to a single process to enable parallel execution on a multiprocessor system.

This advantage of a multi-threaded program allows it to operate faster on computer systems that have multiple CPUs, CPUs with multiple cores, or across a cluster of machines. This is because the threads of the program naturally lend themselves to truly concurrent execution. In such a case, the programmer needs to be careful to avoid race conditions, and other non-intuitive behaviors. In order for data to be correctly manipulated, threads will often need to rendezvous in time in order to process the data in the correct order.

Threads compared with processes

Process: an instance of a program running on a computer.

- Resource ownership.
- Virtual address space to hold process image including program, data, stack, and attributes.
- Execution of a process follows a path though the program.
- **Process switch** — an expensive operation due to the need to save the control data and register contents.

Thread: a dispatchable unit of work within a process.

- Interruptible: processor can turn to another thread.
- All threads within a process share code and data segments.
- **Thread switch**:
- Switching processor between threads within the same process.
- Typically less costly than process switch.

Parallelism

ILP

Parallelism at the machine-instruction level.

• The processor can re-order and pipeline instructions, do aggressive branch prediction, etc.

• Superscalar processors have multiple execution units working in parallel.

• Challenge to find enough instructions that can be executed concurrently.

• Out-of-order execution => instructions are sent to execution units based on instruction dependencies rather than program order.

• Instruction-level parallelism enabled rapid increases in processor speeds over the last 15 years.

TLP

This is parallelism on a more coarser scale.

• Instruction stream divided into smaller streams (threads) to be executed in parallel.

• Thread: has own instructions and data.

• May be part of a parallel program or independent programs.

• Each thread has all state (instructions, data, PC, etc.) needed to execute.

• Wide variety of multithreading designs.

• Single-core superscalar processors cannot fully exploit TLP.

• Multi-core architectures are the next step in processor evolution:

•explicitly exploiting TLP.

• Use multiple instruction streams to improve the throughput of computers that run many programs .

• TLP are more cost-effective to exploit than ILP.

- **Pipelining**

  Implementation Technique whereby multiple instructions are overlapped in execution

  Limited by the dependencies between instructions

  Effected by stalls and effective CPI is greater than 1

- **Instruction Level Parallelism**

  It refers to techniques to increase the number of instructions executed in each clock cycle.

  Exists whenever the machine instructions that make up a program are insensitive to the order in which they are executed if dependencies does not exist, they may be executed.

Instruction-Level Parallelism (ILP)

Concepts involved:

- pipelining
- dynamic scheduling
- dynamic branch prediction
- multiple-issue architectures: superscalar, VLIW
- hardware-based speculation
- compiler techniques and software approaches

Bottom line: There just aren't enough instructions that can actually be executed in parallel!

- instruction issue: limit on maximum issue count
- branch prediction: imperfect
- # registers: finite
- functional units: limited in number
- data dependencies: hard to detect dependencies via memory

Solution

Two Approaches:

- multiple CPU's: each executing a distinct process
  - "Multiprocessors" or "Parallel Architectures"
- single CPU: executing multiple processes ("threads")
  - "Multi-threading" or "Thread-level parallelism"

- **Thread level parallelism**
  - Chip Multi Processing
    - Two processors, each with full set of execution and architectural resources, reside on a single die.
  - Time Slice Multi Threading
    - single processor to execute multiple threads by switching between them
  - Switch on Event Multi Threading
    - switch threads on long latency events such as cache misses

- **Thread level parallelism** (cont..)

  - Simultaneous Multi Threading (SMT)

    Multiple threads can execute on a single processor without switching.

    The threads execute simultaneously and make much better use of the resources.

    It maximizes the performance vs. transistor count and power consumption.

Why Multicore?

Limitations of single core architectures:
- High power consumption due to high clock rates (2-3% power
- Increase per 1% performance increase).
- Heat generation.
- Limited parallelism (Instruction Level Parallelism only).
- Design time and complexity increased due to complex methods to increase ILP.

Many new applications are multithreaded.
- Ex. Multimedia applications.

General trend in computer architecture (shift towards more parallelism).

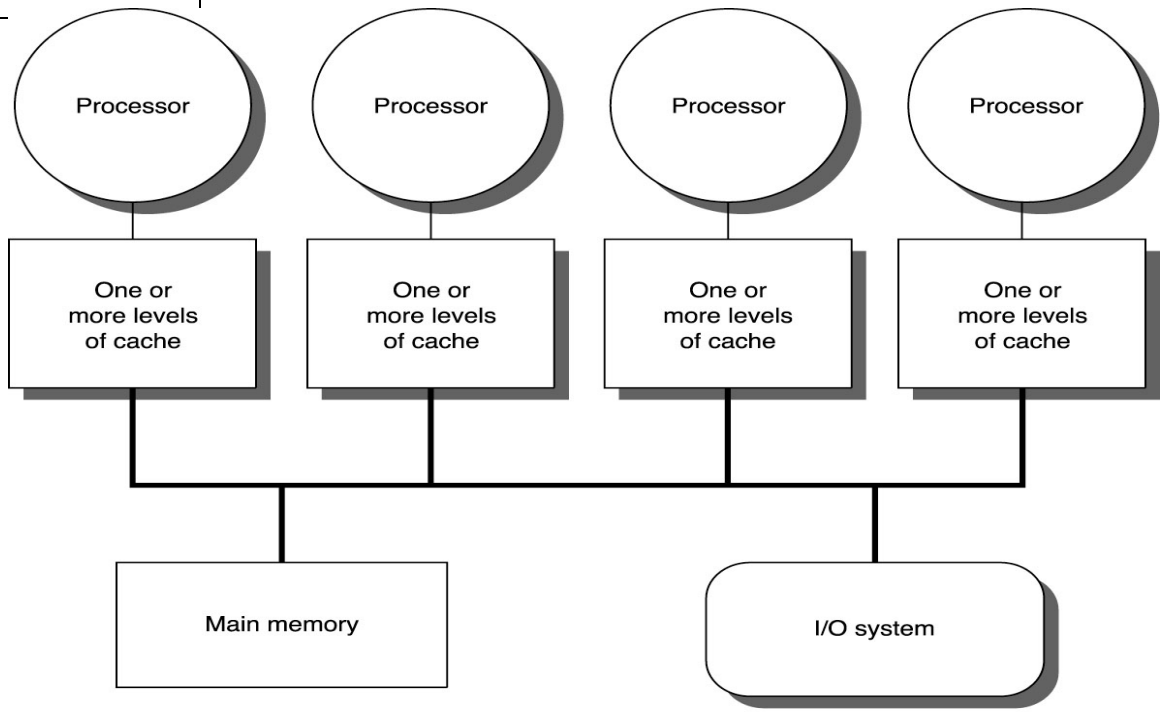
Much faster cache coherency circuits.

Smaller in physical size than SMP.

Multicore: Memory Organization

Centralized, shared-memory multiprocessor:

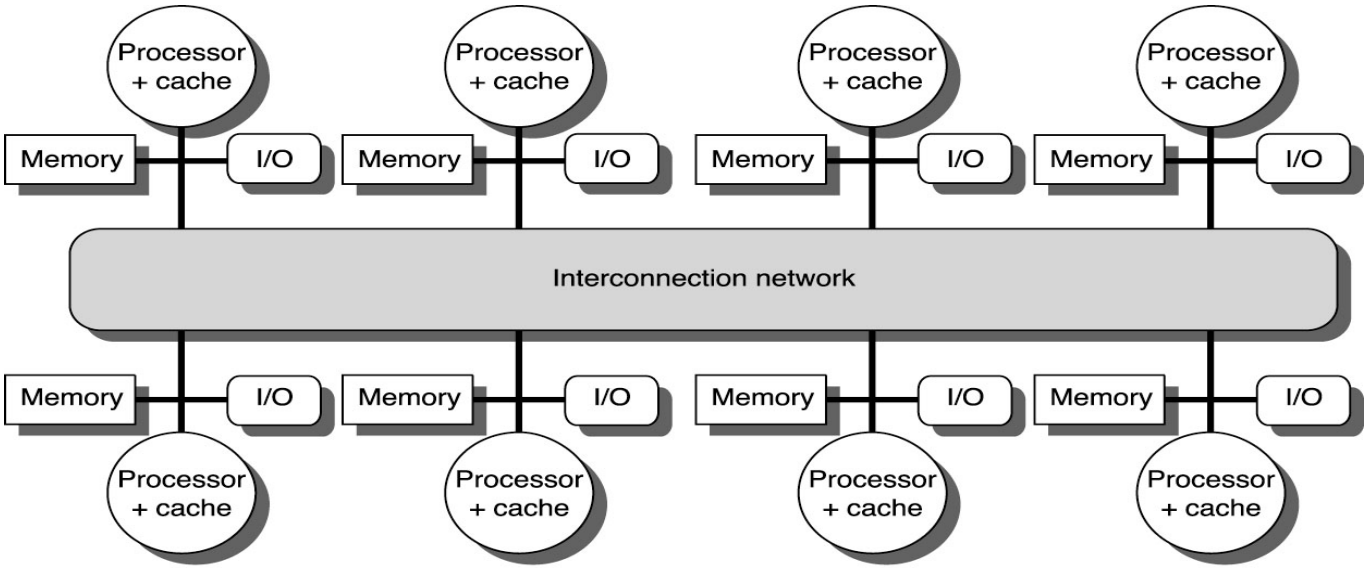
- usually few processors
- share single memory & bus
- use large caches



Multicore: Memory Organization

Distributed-memory multiprocessor:

- can support large processor counts
  - cost-effective way to scale memory bandwidth
  - works well if most accesses are to local memory node
- requires interconnection network
  - communication between processors becomes more complicated, slower



Multicore: Hybrid Organization

- Use distributed-memory organization at top level
- Each node itself may be a shared-memory multiprocessor (2-8 processors)

Multicore: Communication Mechanisms

- Shared-Memory Communication
  - around for a long time, so well understood and standardized
    - memory-mapped
  - ease of programming when communication patterns are complex or dynamically varying
  - better use of bandwidth when items are small
  - ***Problem:*** cache coherence harder
    - use "Snoopy" and other protocols

- Message-Passing Communication
  - simpler hardware because keeping caches coherent is easier
  - communication is explicit, simpler to understand
    - focuses programmer attention on communication
  - synchronization: naturally associated with communication
    - fewer errors due to incorrect synchronization

Multithreading approaches

Interleaved (fine-grained)

- Processor deals with several thread contexts at a time.
- Switching thread at each instruction or clock cycle (hardware need).
- The interleaving is usually round-robin
- If a thread is blocked, it is skipped.
- Hide latency of both short and long pipeline stalls.
- it slows down the execution of individual threads – but better throughput

Blocked (coarse-grained)

- Thread executed until an event causes delay (e.g., switch only on a level-2 cache miss)
- Relieves need to have very fast thread switching.
- No slow down for ready-to-go threads.
- shorter stalls are ignored, and there may be plenty of those
- every switch involves emptying and restarting the instruction pipeline

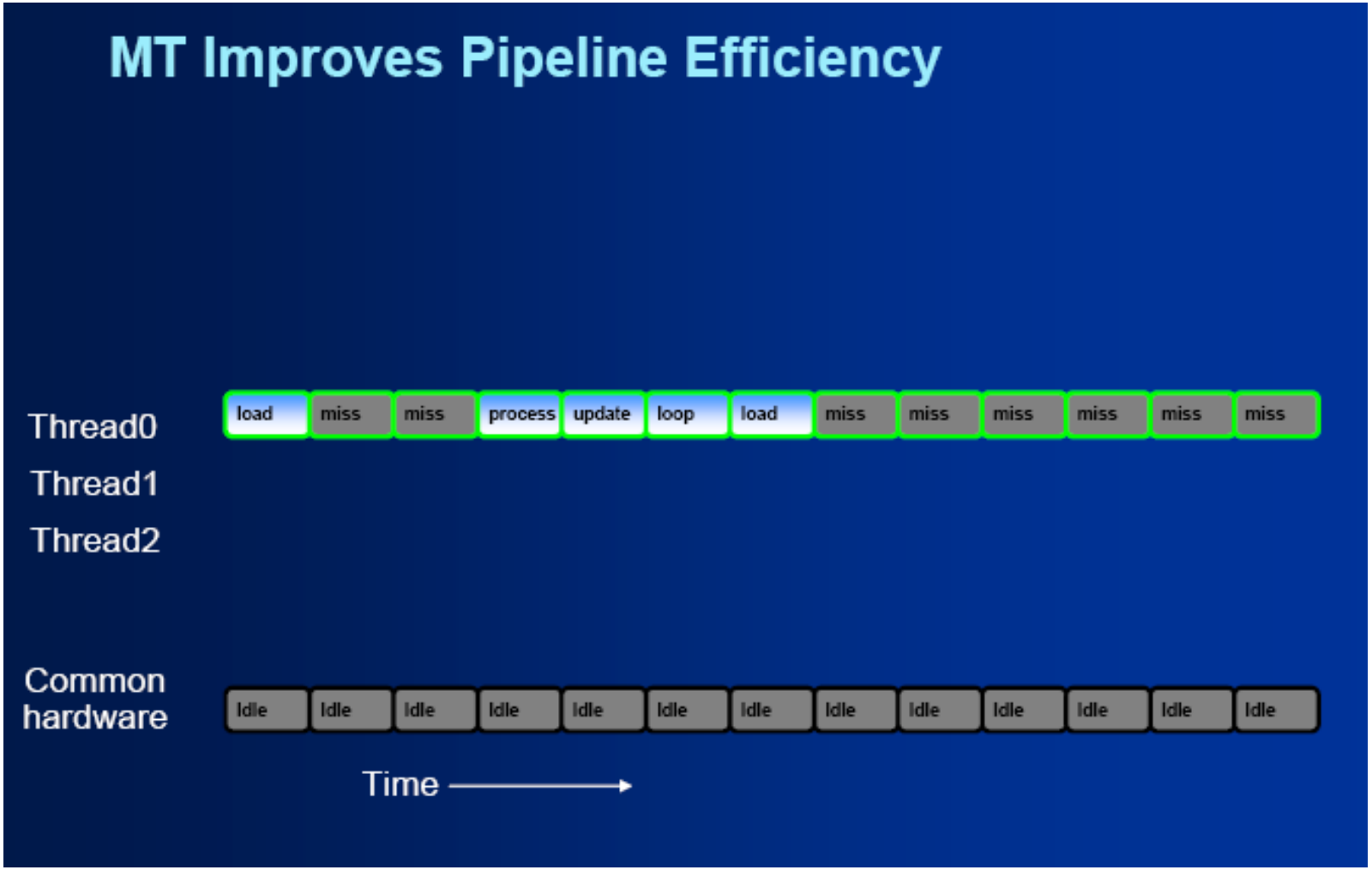
Simultaneous (SMT)

- Instructions simultaneously issued from multiple threads to execution units of superscalar processor.
- multiple functional units
- register renaming and dynamic scheduling

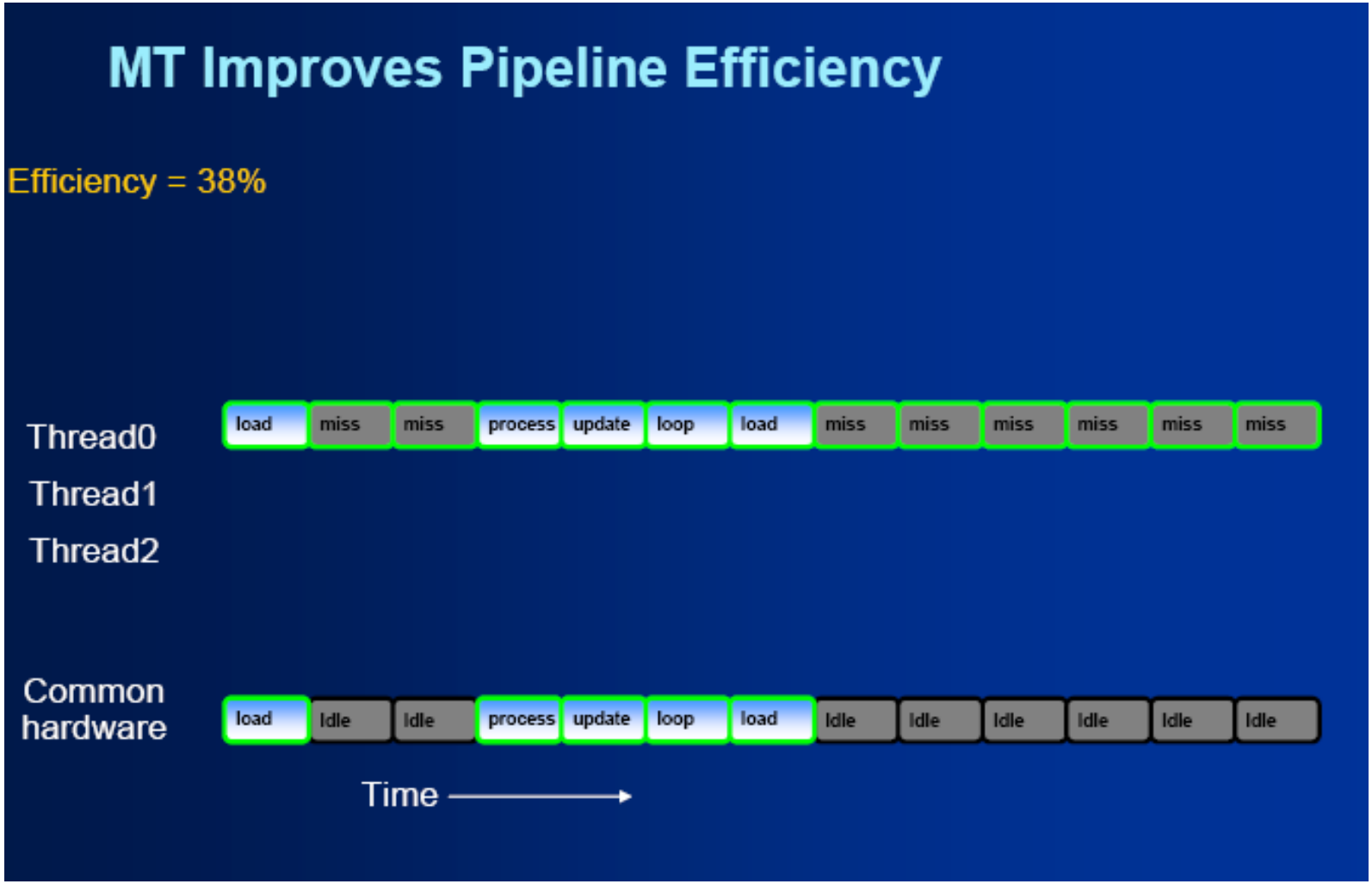
Chip multiprocessing

- Each processor handles separate threads.

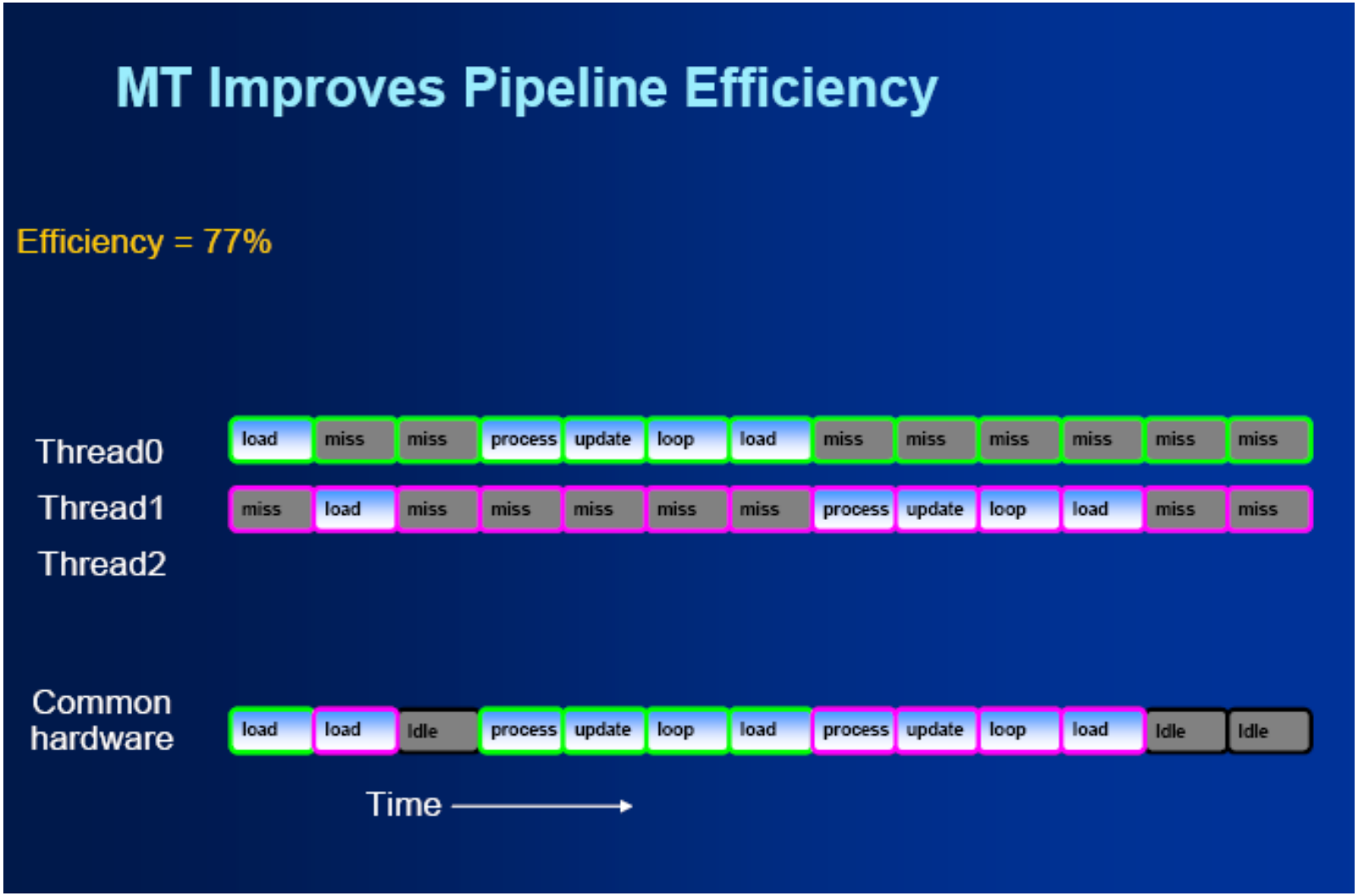
Multithreading Example



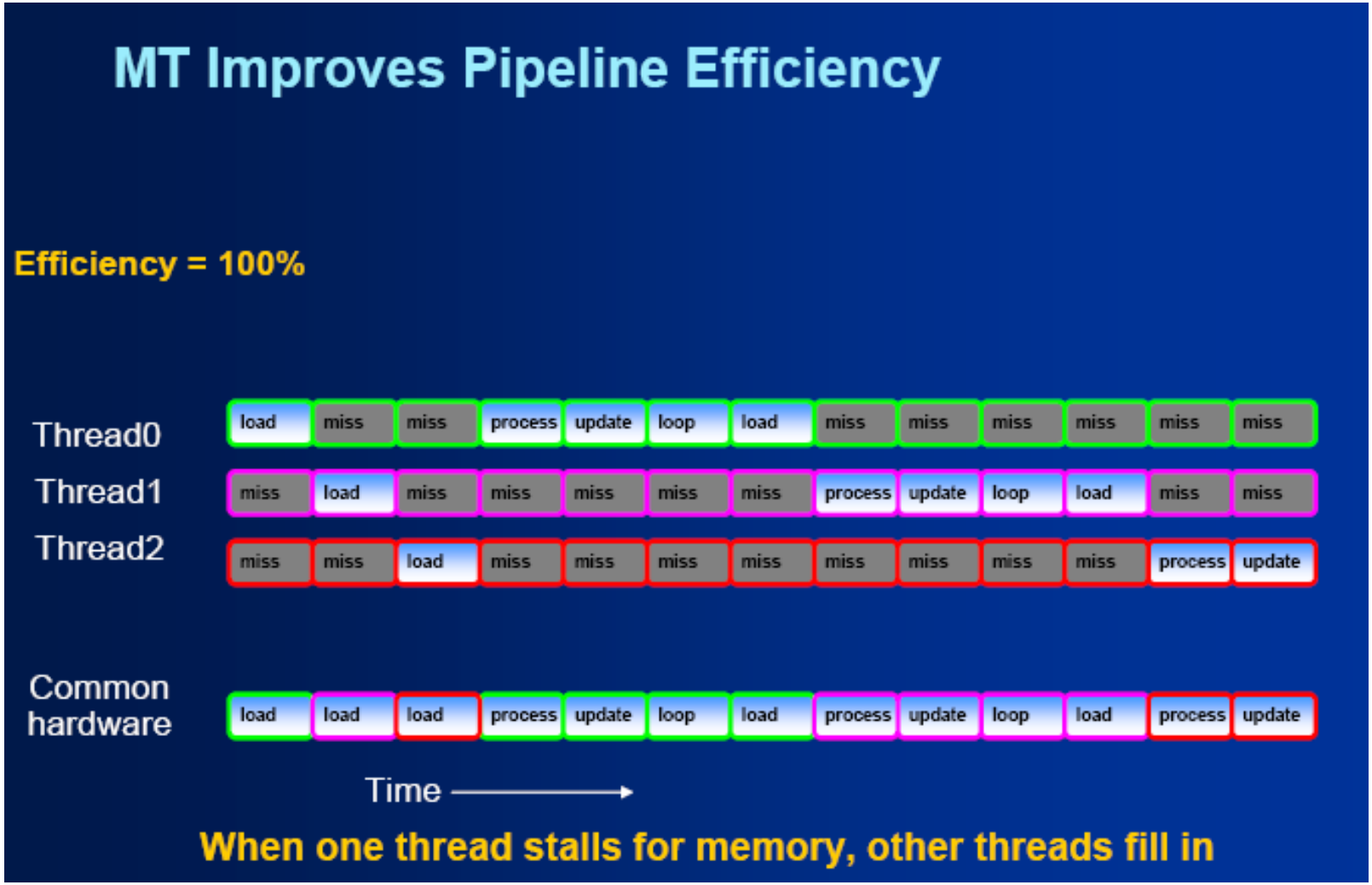
Multithreading Example



Multithreading Example



Multithreading Example



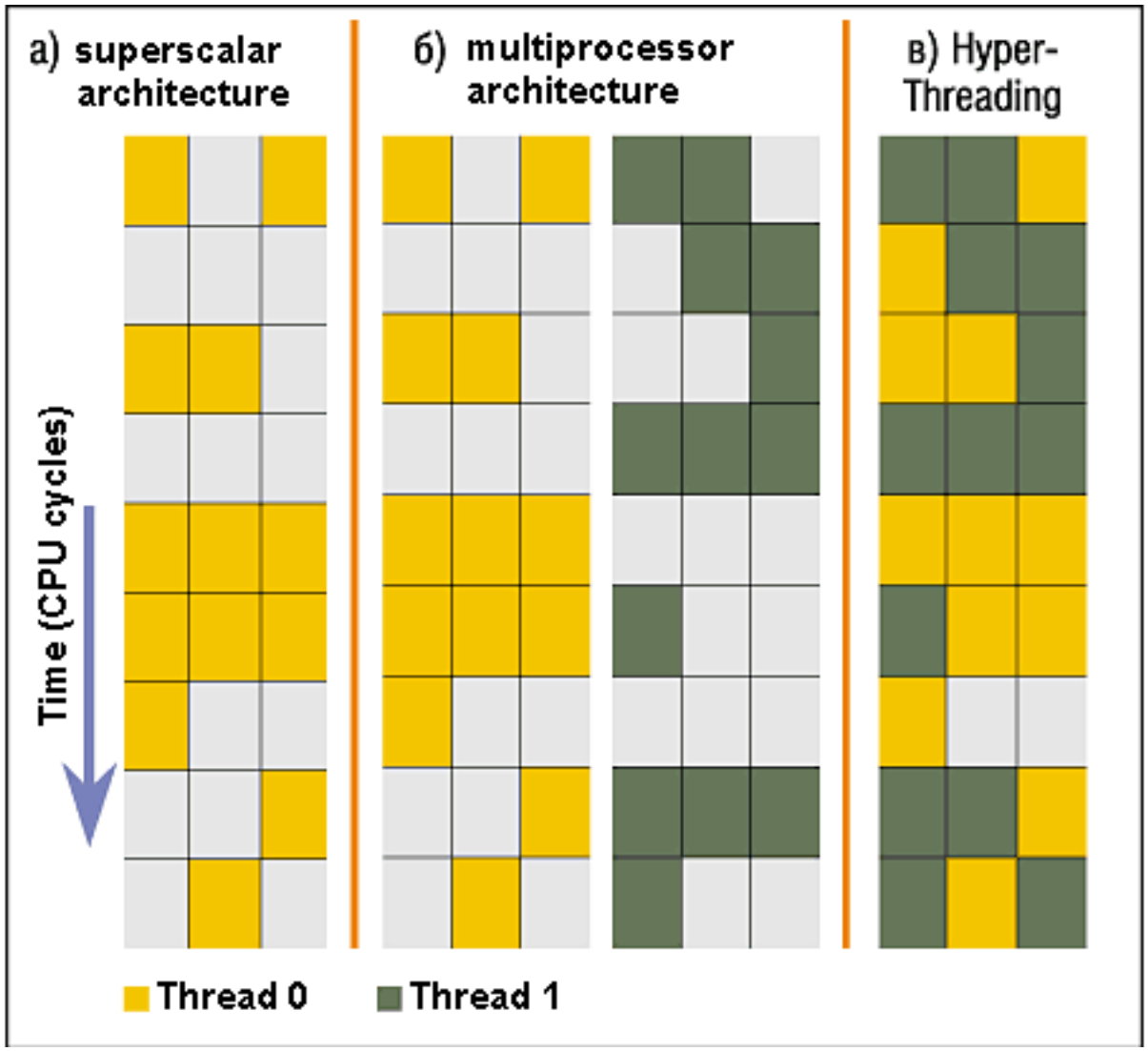
Definitions

- Process – a program, made up of threads.
- Context – all the information that completely describes the process's current state of execution.
- Thread – A thread is a single,sequential flow of control within a program.
- Context switch – saving the current process's context, flushing the CPU and loading the next process's context.

Hyper-Threading Technology

- Hyper-Threading Technology brings the simultaneous multi-threading approach to the Intel architecture.
- Hyper-Threading Technology makes a single physical processor appear as two or more logical processors
- Hyper-Threading Technology first invented by Intel Corp.
- Hyper-Threading Technology provides thread-level-parallelism (TLP) on each processor resulting in increased utilization of processor and execution resources.
- Each logical processor maintain one copy of the architecture state

Threading Examples



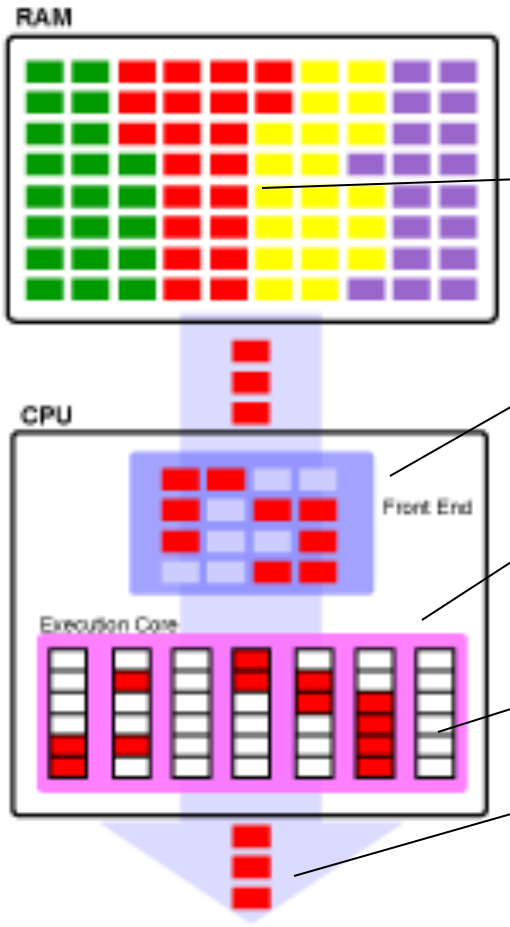
Scalar Processor Approaches

Single-threaded scalar
Simple pipeline and no multithreading.

Interleaved multithreaded scalar
Easiest multithreading to implement.
Switch threads at each clock cycle.
Pipeline stages kept close to fully occupied.
Hardware needs to switch thread context between cycles.

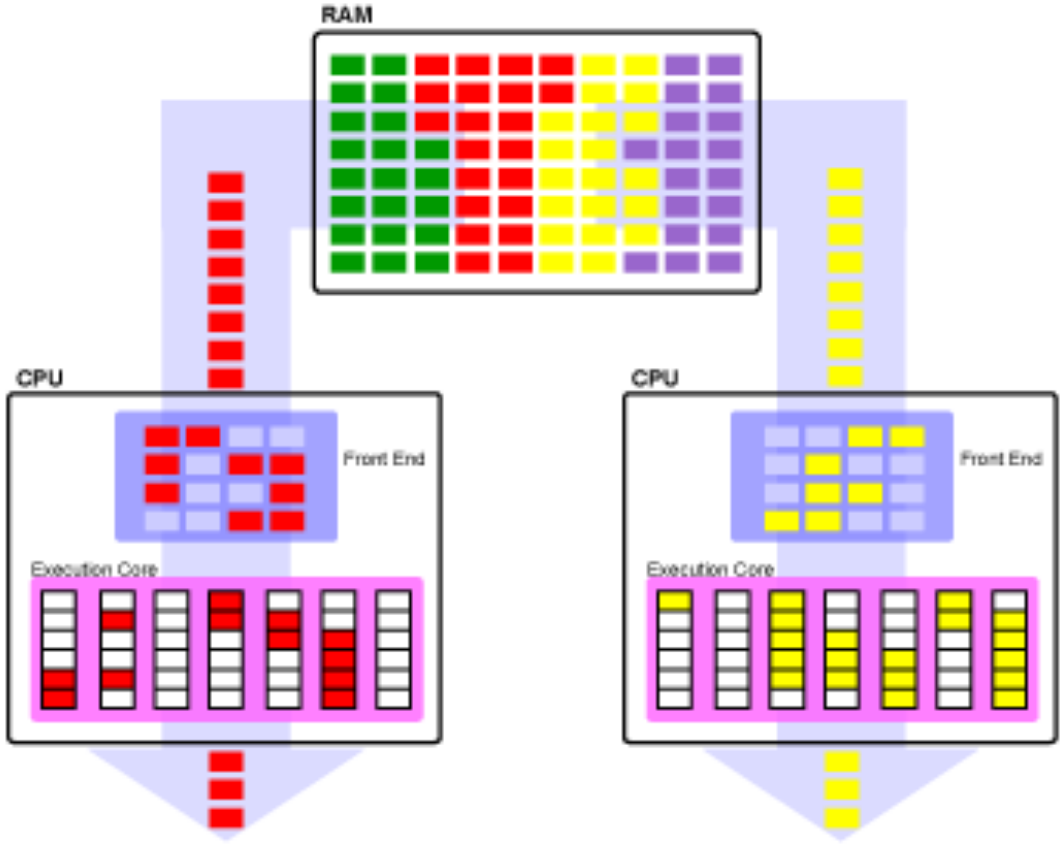
Blocked multithreaded scalar
Thread executed until latency event occurs to stop the pipeline.
Processor switches to another thread

Single threaded CPU



- 4 Colors indicate 4 different processes.
- Front end can issue 4 instructions per clock cycle to exec core.
- 7 columns (in exec core) represent functional units of the pipeline. 6 rows represent pipeline stages.
- Only red instructions are actually being executed.
- White spaces = pipeline bubbles – missed opportunity for work.
- CPU can issue up to 4 instructions per clock cycle but on most cycles it issues 2, once it issues 3.

Single threaded SMP



- Uses two CPU's that each share a process – this cuts down on # context switches & provides more CPU time to each process.

- Red and yellow processes are executing simultaneously – one on each processor. More time available for execution & less time waiting in memory for a new time slice.

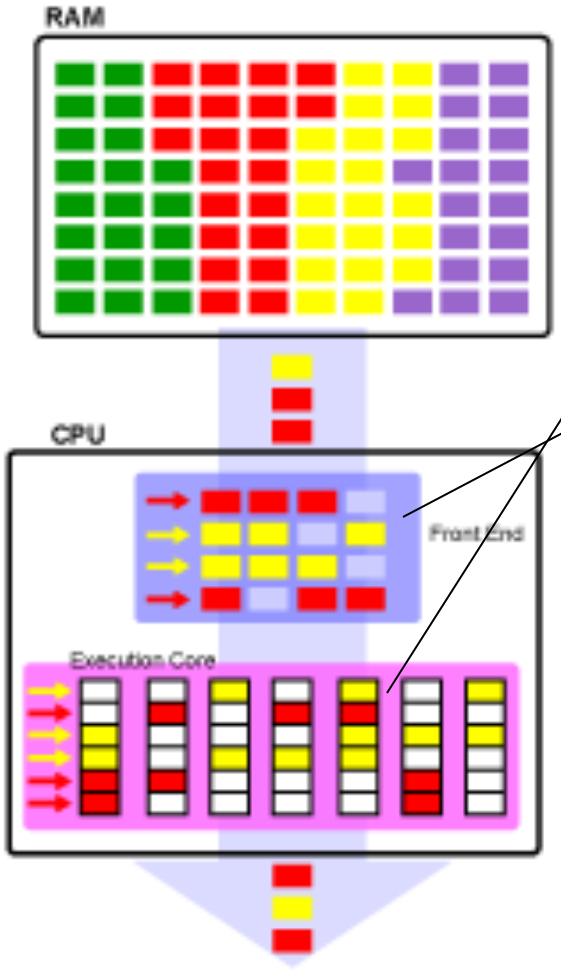
- White spaces indicate wasted execution slots.

Definition

- Super-threading = time slice multi-threading

A single CPU is capable of executing more than 1 thread at a time.

Super-threaded CPU

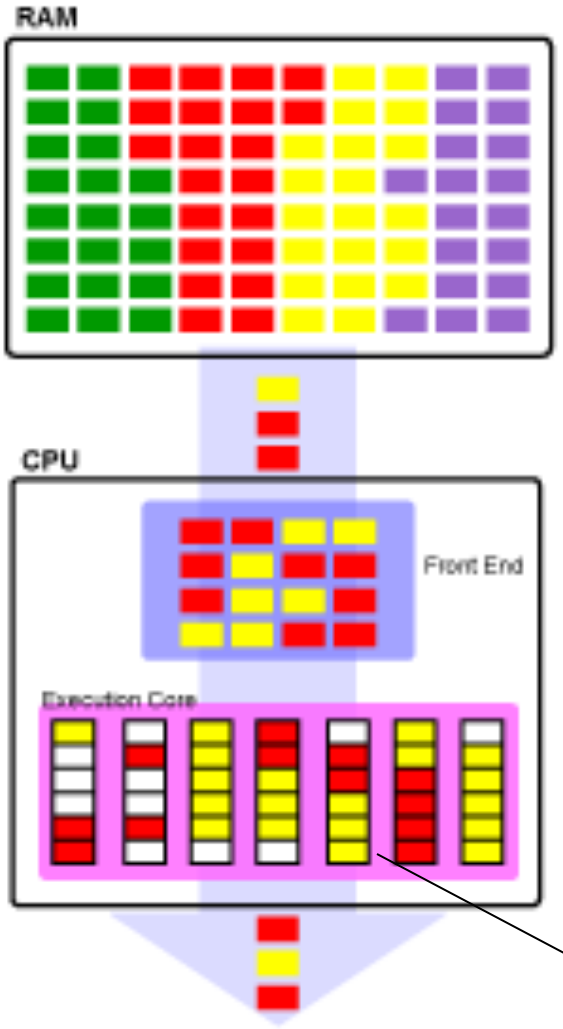


- Fewer wasted execution slots.
- Each of 6 pipeline stages can contain instructions for only 1 thread.
- 4 instructions per clock can be issued to any 4 of the 7 functional unit pipelines.
in the execution core (all 4 instructions must come from the same thread in 1 CPU clock cycle).
- Hides memory access latencies but if scheduler can find only 2 instructions to execute, other 2 issue slots will be unused.

Note:

Mult-ithreaded processors can help alleviate some of the latency problems brought on by DRAM memory's slowness relative to the CPU. For instance, consider the case of a multithreaded processor executing two threads, red and yellow. If the red thread requests data from main memory and this data isn't present in the cache, then this thread could stall for many CPU cycles while waiting for the data to arrive. In the meantime, however, the processor could execute the yellow thread while the red one is stalled, thereby keeping the pipeline full and getting useful work out of what would otherwise be dead cycles.

Hyper-threaded CPU = SMT



- Simultaneous multi-threading (SMT) = hyper-threading (same as super-threading except That all issued instructions need not be from the same thread.

- Red & yellow threads combined to fit together on a single hyper-threaded processor. Acts like 2 CPU's in one.

- A SMT processor is split into 2 or more logical processors - threads can be scheduled to execute on any logical processor.

- HT Allows scheduling logic max. flexibility to fill execution slots.

- HT uses less resources and has less wasted slots.

Thread Sharing of pipeline Resources

- Major Sharing Schemes are-
  - Partition
  - Threshold
  - Full Sharing

Partition

- Each logical processor uses half the resources
- Simple and low in complexity
- Ensures fairness and progress
- Good for *major pipeline queues*

Threshold

- Puts a threshold on number of resource entries a logical processor can use.
- Limits maximum resource usage
- For small structures where resource utilization in burst and time of utilization is short, uniform and predictable
- Eg- Processor Scheduler

Full Sharing

- Most flexible mechanism for resource sharing, do not limit the maximum uses for resource usage for a logical processor
- Good for large structures in which working set sizes are variable and there is no fear of starvation
- Eg: All Processor caches are shared
  - Some applications benefit from a shared cache because they share code and data, minimizing redundant data in the caches

Implementing Hyper-threading - Xeon

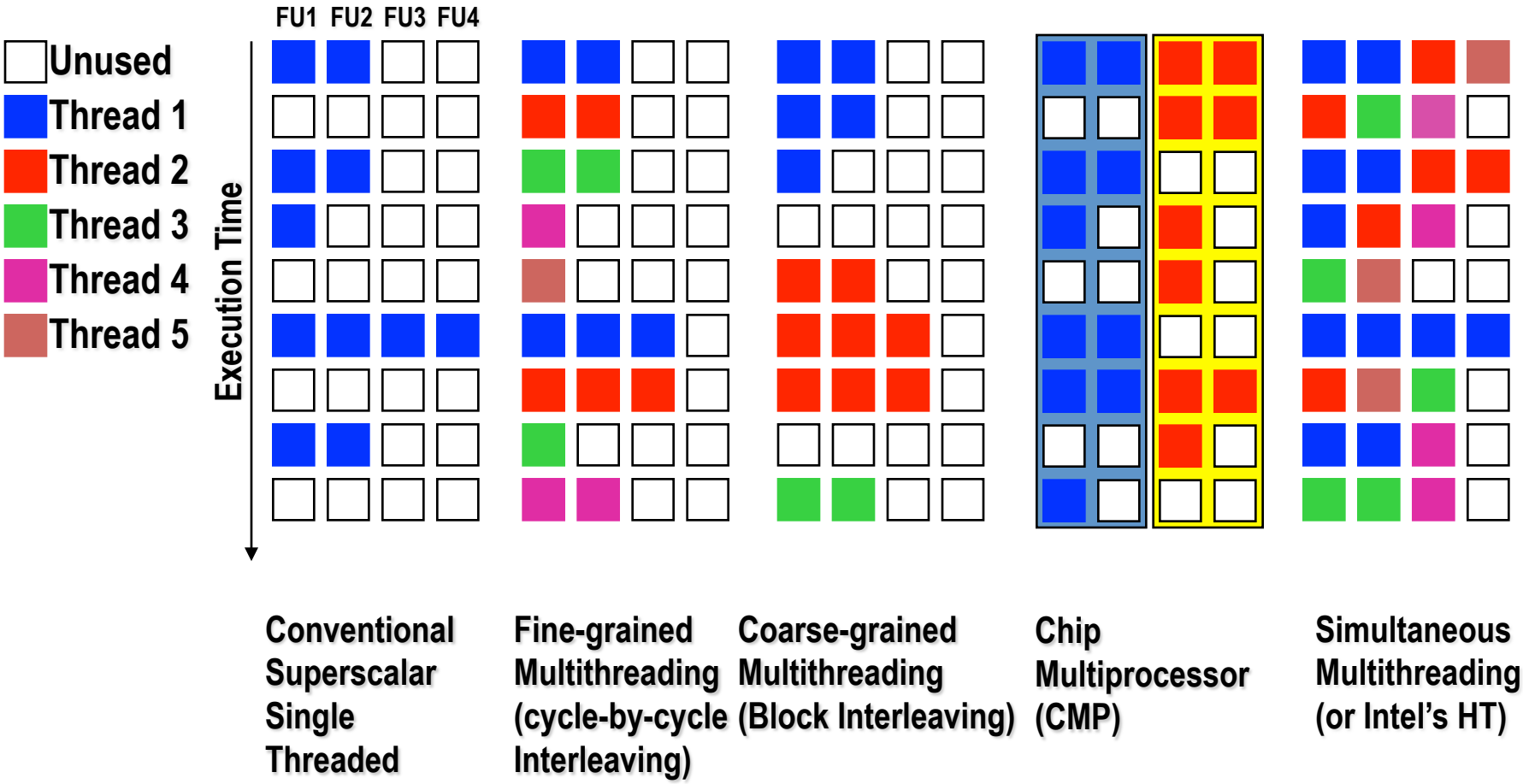
- Hyper-threading hardware adds only 5% to Xeon die area.
- Xeon can execute at most 2 threads in parallel on 2 logical processors by maintaining 2 independent thread contexts.
- Xeon does this by dividing processor resources into 3 types:

  1) Replicated (for each logical processor)– Reg. Rename logic, IP, ITLB, return stack predictor, other regs.

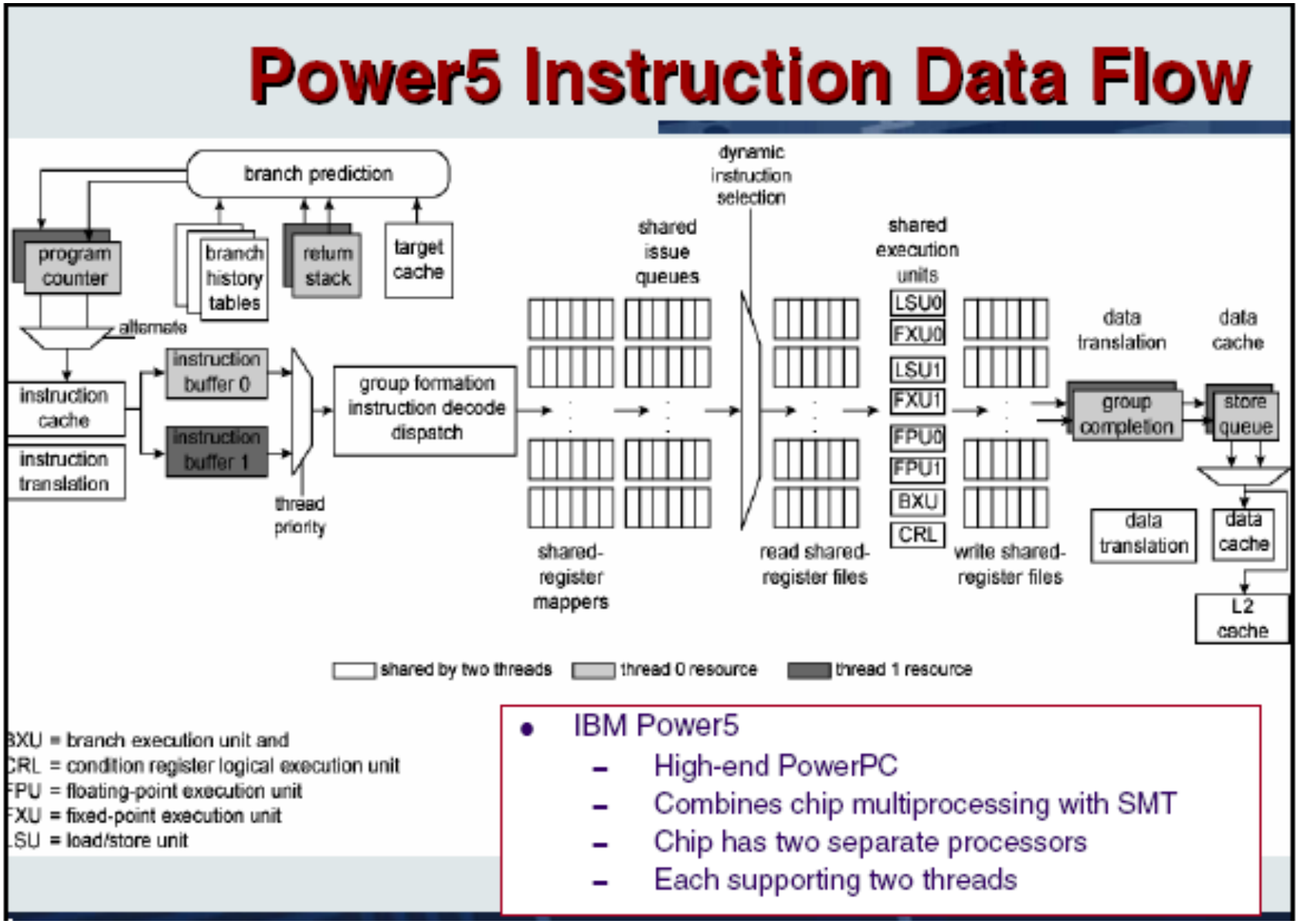
  2) Partitioned – ROB's, load/store buffers, queues (u-op, etc.)

  3) Shared – caches (trace, L1, L2, L3), microarchitectural regs, execution units.

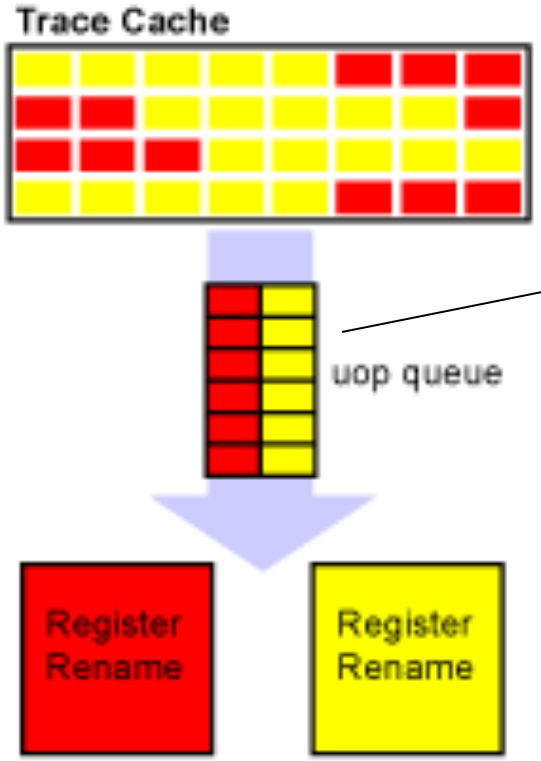
SMT Concept vs. Other Alternatives



Resources


Power5 Instruction Data Flow
branch prediction
program counter
branch history tables
return stack
target cache
alternate
instruction cache
instruction translation
instruction buffer 0
instruction buffer 1
thread priority
group formation instruction decode dispatch
shared issue queues
dynamic instruction selection
shared execution units
LSU0
FXU0
LSU1
FXU1
FPU0
FPU1
BXU
CRL
shared-register mappers
read shared-register files
write shared-register files
data translation
data cache
group completion
store queue
data translation
data cache
L2 cache
shared by two threads
thread 0 resource
thread 1 resource
BXU = branch execution unit and
CRL = condition register logical execution unit
FPU = floating-point execution unit
FXU = fixed-point execution unit
LSU = load/store unit
IBM Power5
High-end PowerPC
Combines chip multiprocessing with SMT
Chip has two separate processors
Each supporting two threads

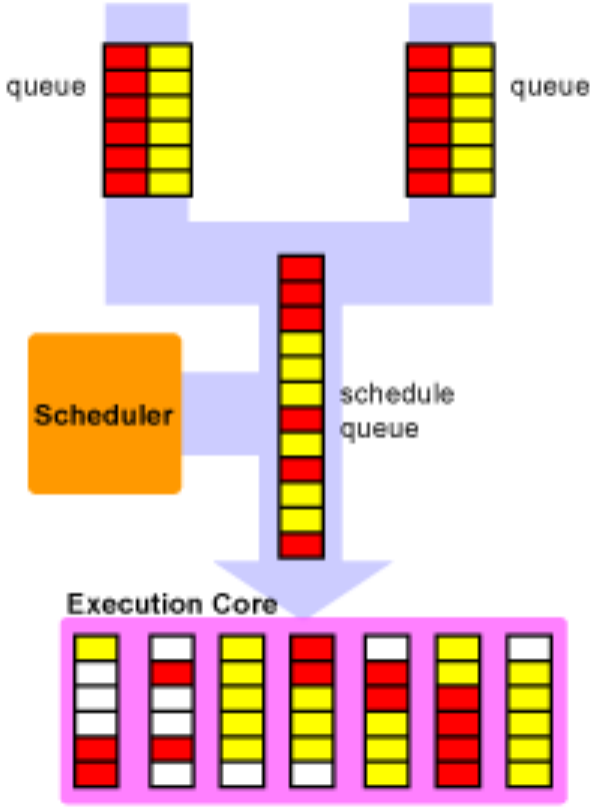
Statically partitioned queue



In a scheduling queue with 12 entries, instead of assigning entries 0 through 5 to logical processor 0 and entries 6 through 11 to logical processor 1, the queue allows any logical processor to use any entry but it places a limit on the number of entries that any one logical processor can use. So in the case of a 12-entry scheduling queue, each logical processor can use no more than six of the entries.

- Each queue is split in half, with half its entries designated for sole use by one Logical processor

Dynamically partitioned Queue



From the point of view of each logical processor and thread, this kind of dynamic partitioning has the same effect as fixed partitioning: it confines each LP to half of queue. However, from the point of view of the physical processor, there's a crucial difference between the two types of partitioning. The scheduling logic, like the register file and the execution units, is a shared resource, a part of the Xeon's microarchitecture that is SMT-unaware. The scheduler has no idea that it's scheduling code from multiple threads. It simply looks at each instruction in the scheduling queue on a case-by-case basis, evaluates the instruction's dependencies, compares the instruction's needs to the physical processor's currently available execution resources, and then schedules the instruction for execution. To return to the example from our hyper-threading diagram, the scheduler may issue one red instruction and two yellow to the execution core on one cycle, and then three red and one yellow on the next cycle. So while the scheduling queue is itself aware of the differences between instructions from one thread and the other, the scheduler in pulling instructions from the queue sees the entire queue as holding a single instruction stream.

- Xeon's scheduling queue (1 of 3) is dynamically partitioned – any logical processor can use any entry but places a limit

Shared resources – Pro's & Con's

- The more resources that are shared, the more efficient HT can be.
  Xeon has 128 microarchitectural floating point regs (FPRs).

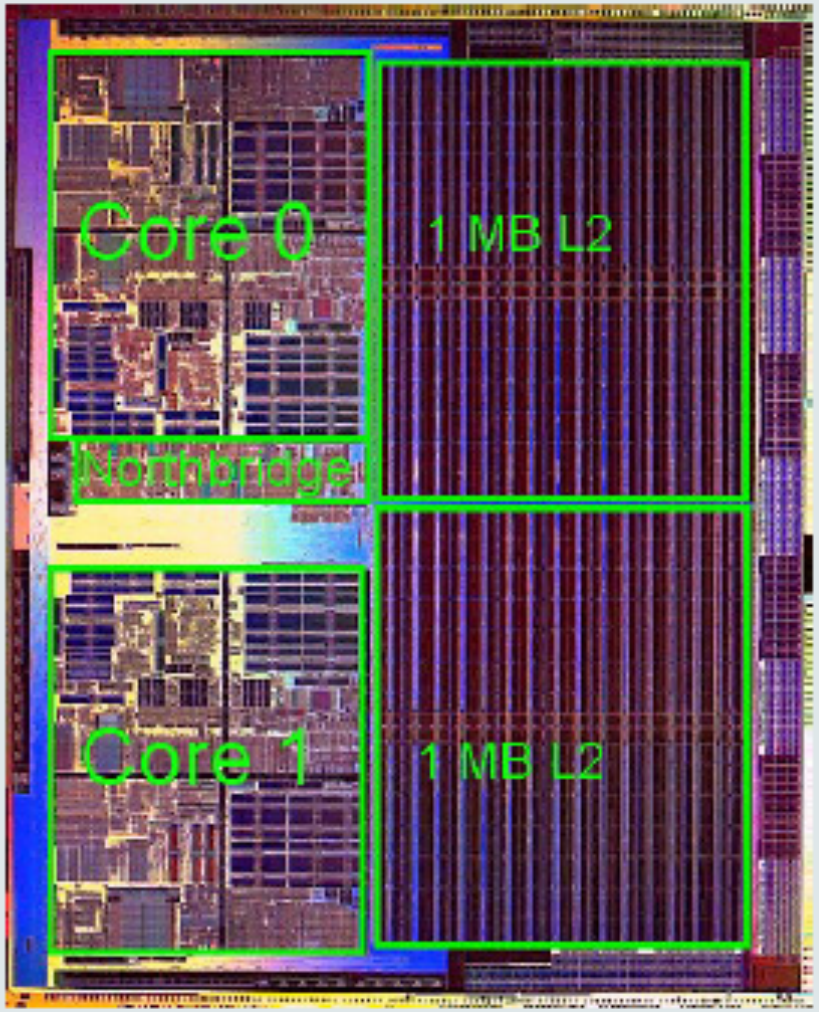
- Problem with sharing resources - If one thread monopolizes a crucial resource, the other thread will stall.

- Caches are shared by logical processors so coherency problems are minimized. Logical processors don't have to snoop other caches to see if they have a current copy.

- Cache conflicts may occur. Since caches are not SMT aware, any one thread can Monopolize the entire cache. The CPU will continue to run both threads. This can Cause thrashing as data for each thread is swapped in & out until bus & cache Bandwidth are maxed out.

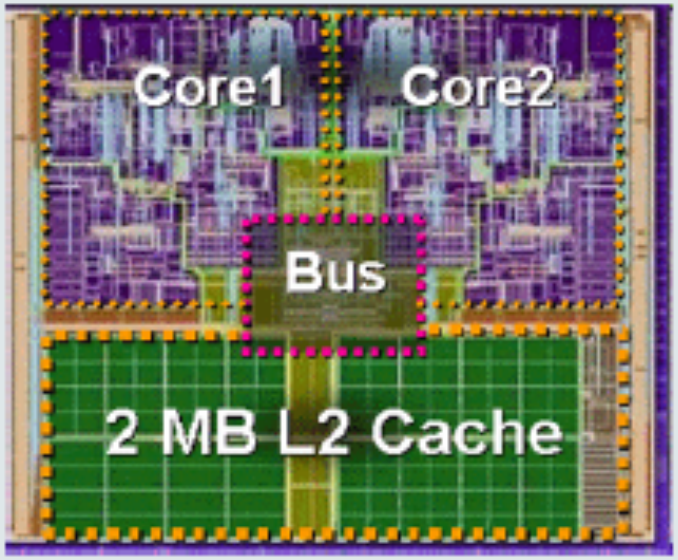
Multicore CPUs - AMD

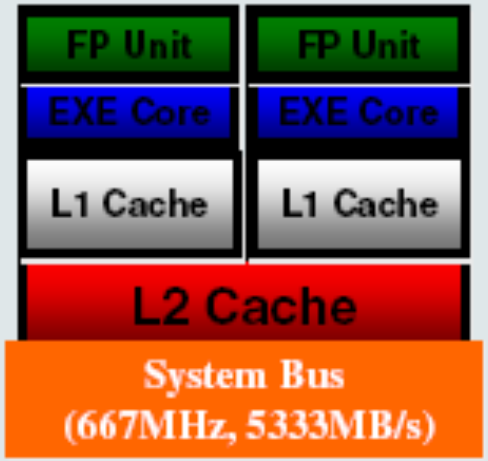
- Dual-core AMD Opteron™ processor is 199mm^2 in 90nm
- Single-core AMD Opteron processor is 193mm^2 in 130nm



Multicore CPUs - Intel

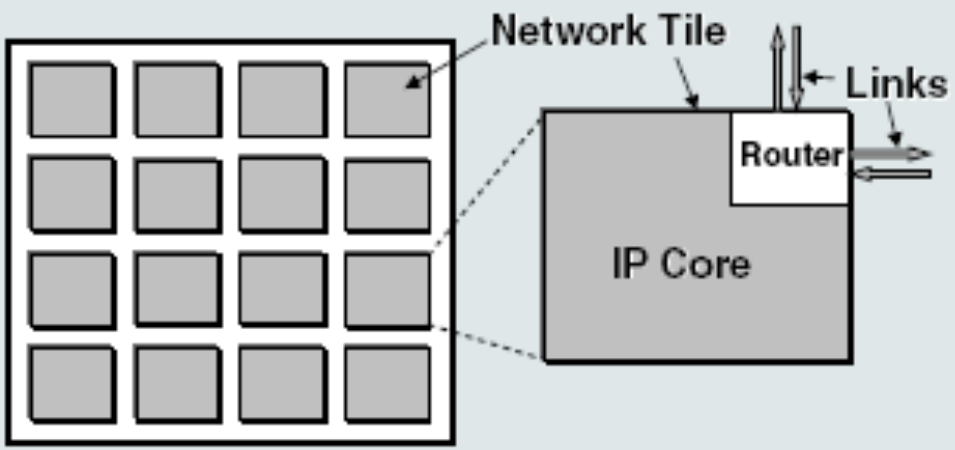
- Two identical physical cores in a package.
- Each with its own execution resources.
- Each with its own L1 cache.
- Both cores share the L2 cache.

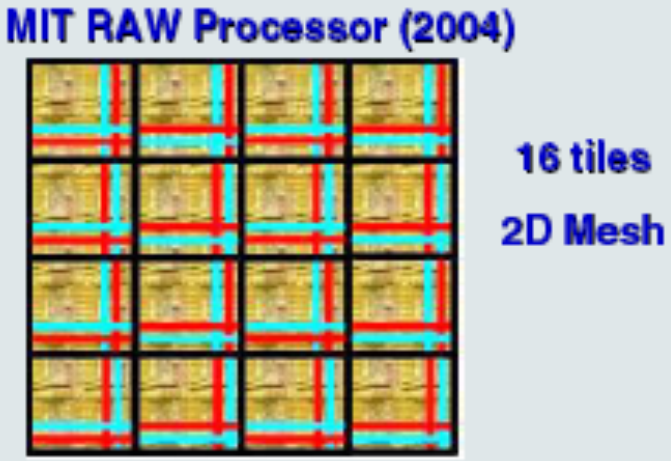




Multicore CPUs – Future – Network on chip

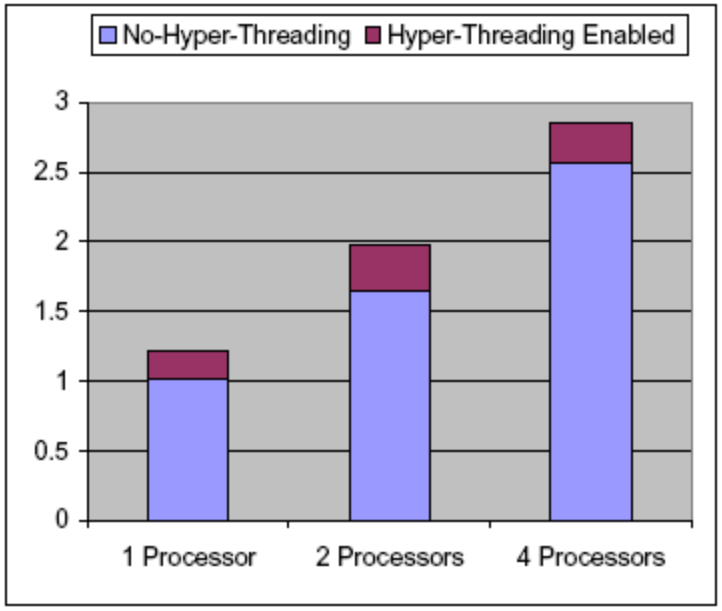
- Create an tiled architecture that scales to 100's-1000's of processors on the same chip.
- Replace shared buses with on-chip packet switched networks.
- Tiles connected via on-die interconnect fabric that routes packets.
- Structured wiring allows high-performance circuits use to maximize BW.
- Fabric power is now a large portion of chip budget.



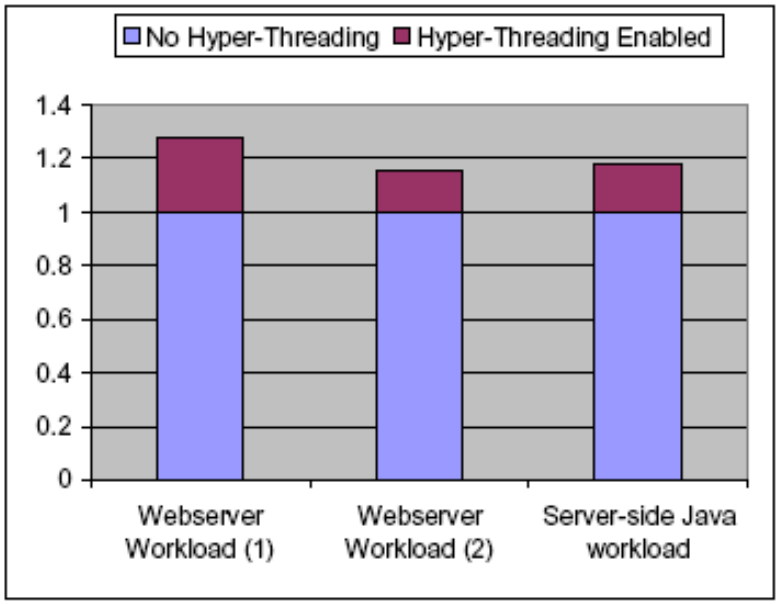


Business Benefits of Hyper-Threading Technology

- Higher transaction rates for e-Businesses
- Improved reaction and response times for end-users and customers.
- Increased number of users that a server system can support
- Handle increased server workloads
- Compatibility with existing server applications and operating systems



Performance increases from Hyper-Threading Technology on an OLTP workload



Web server benchmark performance

Conclusion

- Symmetric multi-threaded aware OSes (Windows NT, Linux, etc.)
  Know how to intelligently schedule processes between both processors
  So that resource contention is minimized.

- SMT increases efficiency in these systems.

- All new processor designs from top microprocessor vendors will implement SMT and be capable of SMP.

- Intel's Hyper-Threading Technology brings the concept of simultaneous multi-threading to the Intel Architecture.

- It will become increasingly important going forward as it adds a new technique for obtaining additional performance for lower transistor and power costs.

- The goal was to implement the technology at minimum cost while ensuring forward progress on logical processors, even if the other is stalled, and to deliver full performance even when there is only one active logical processor.