Arquitectura de Computadores

Capítulo 4.
Técnicas Avanzadas en Paralelismo. Parte 1.

## Escuela Politécnica Superior

Universidad Autónoma de Madrid

Profesores:
G130 y 131: Gustavo Sutter
G135 y 139: Iván González Martínez
G132: Francisco Javier Gómez Arribas
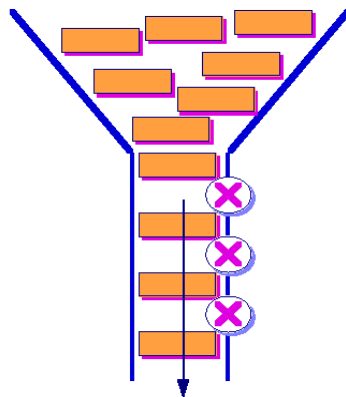
Processor Architectures

- Scalar processor : Pipelining, ILP

- Superescalar processor:

    - Multiple issue

    - Dynamic scheduling

- VLIW processor:

    - Static multiple issue

- MultiThreading

- Simultaneous Multithreading

- Multicore

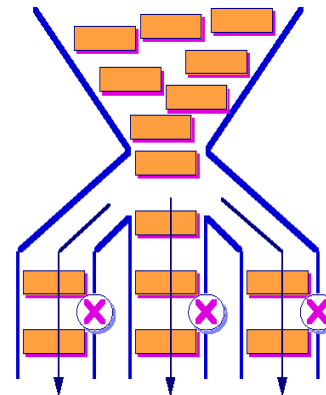# Increasing the Processor Performance

- Scalar
- Pipeline
- Superscalar
- Very Long Instruction Word (VLIW)
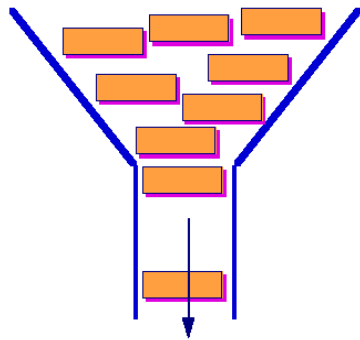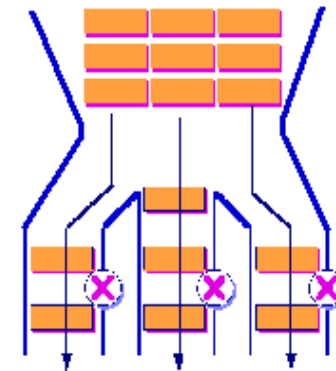
Scalar   Pipelined   Superscalar   VLIW

Instruction-Level Parallelism (ILP)

- Pipelining: executing multiple instructions in parallel

- To increase ILP
  - Deeper pipeline
    - Less work per stage $\Rightarrow$ shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages $\Rightarrow$ multiple pipelines
    - Start multiple instructions per clock cycle
    - CPI < 1, so use Instructions Per Cycle (IPC)
    - E.g., 4GHz 4-way multiple-issue
      - 16 BIPS, peak CPI = 0.25, peak IPC = 4
    - But dependencies reduce this in practice

Multiple Issue

- Static multiple issue
  - Compiler groups instructions to be issued together
  - Packages them into "issue slots"
  - Compiler detects and avoids hazards

- Dynamic multiple issue
  - CPU examines instruction stream and chooses instructions to issue each cycle
  - Compiler can help by reordering instructions
  - CPU resolves hazards using advanced techniques at runtime

Static Multiple Issue

- Compiler groups instructions into "issue packets"
  - Group of instructions that can be issued on a single cycle
  - Determined by pipeline resources required

- Think of an issue packet as a very long instruction
  - Specifies multiple concurrent operations
  - $\Rightarrow$ Very Long Instruction Word (VLIW)

Scheduling Static Multiple Issue

- Compiler must remove some/all hazards
  - Reorder instructions into issue packets
  - No dependencies with a packet
  - Possibly some dependencies between packets
    - Varies between ISAs; compiler must know!
  - Pad with *nop* if necessary

Dynamic Multiple Issue

- "Superscalar" processors

- CPU decides whether to issue 0,1, 2, … each cycle

  - Avoiding structural and data hazards

- Avoids the need for compiler scheduling

  - Though it may still help

  - Code semantics ensured by the CPU

Dynamic Pipeline Scheduling

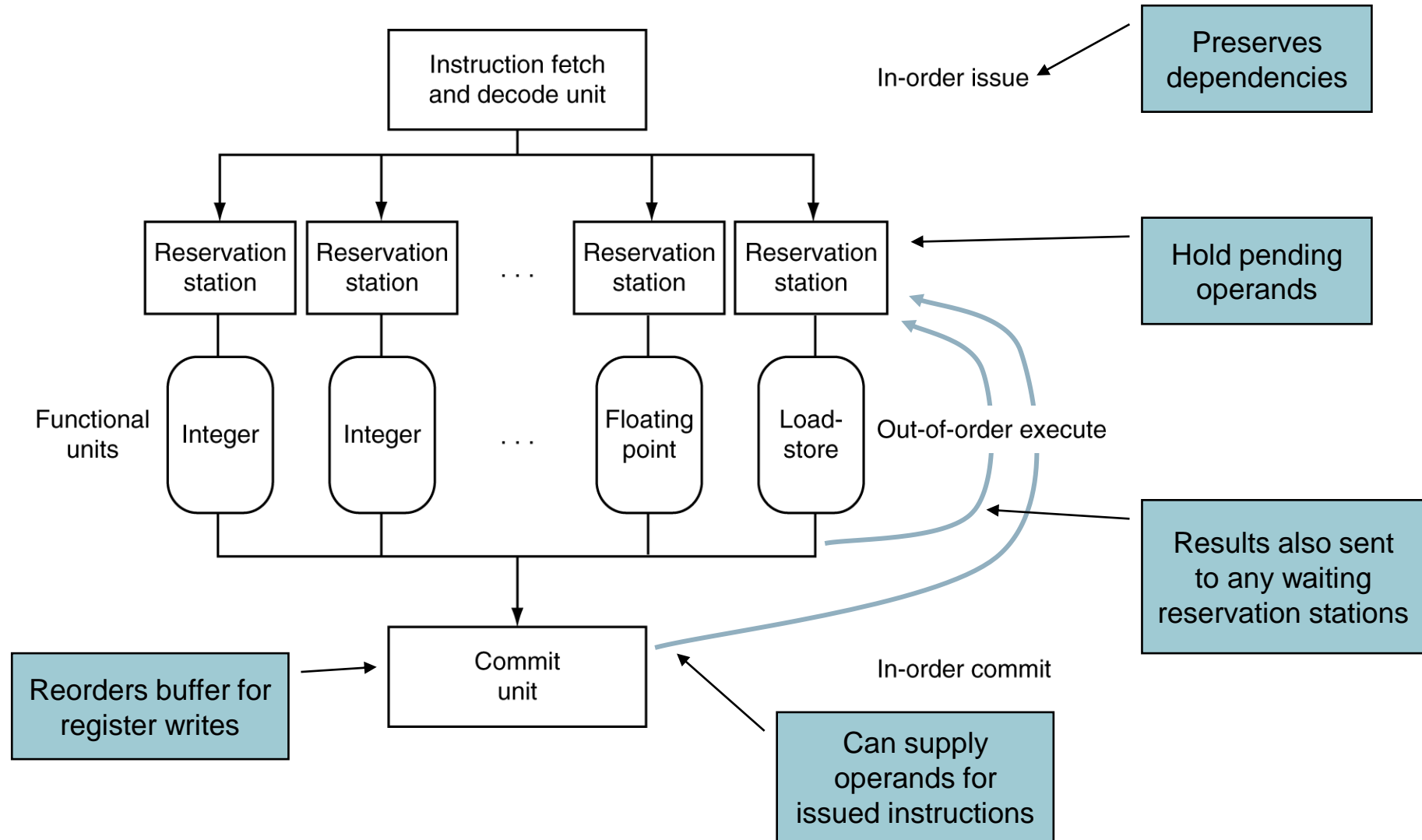- Allow the CPU to execute instructions out of order to avoid stalls
    - But commit result to registers in order
- Example

    ```
    lw     $t0, 20($s2)
    addu   $t1, $t0, $t2
    sub    $s4, $s4, $t3
    slti   $t5, $s4, 20
    ```
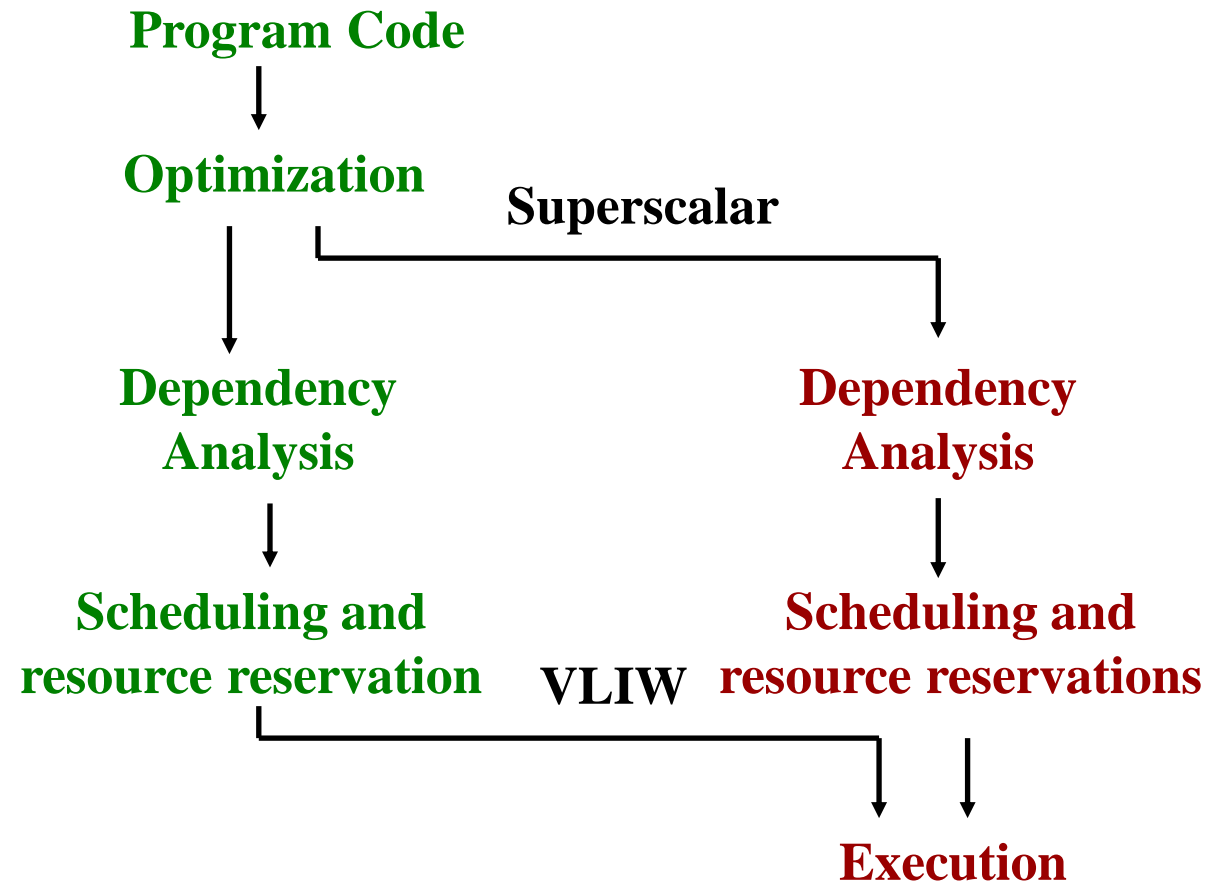
    - Can start sub while addu is waiting for lw
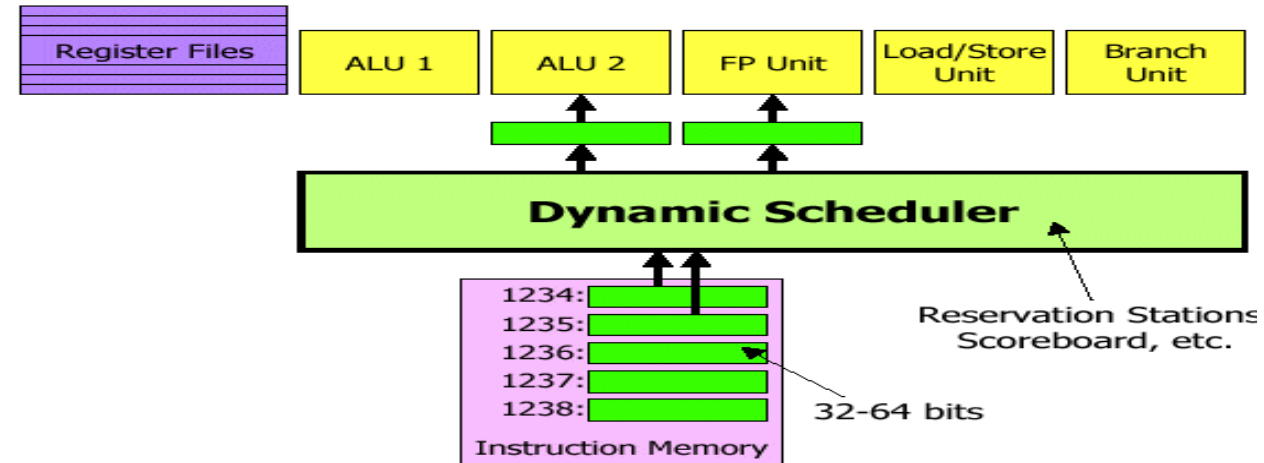
Dynamically Scheduled CPU

Instruction fetch and decode unit

In-order issue

Preserves dependencies

Reservation station — Reservation station — . . . — Reservation station — Reservation station

Hold pending operands

Functional units

Integer — Integer — . . . — Floating point — Load-store

Out-of-order execute

Results also sent to any waiting reservation stations

Reorders buffer for register writes

Commit unit

In-order commit

Can supply operands for issued instructions

Superscalar vs VLIW

Different division of tasks between the SOFTWARE & HARDWARE

Program Code

↓

Optimization

Superscalar

Dependency Analysis   **Dependency Analysis**

↓   ↓

Scheduling and resource reservation   VLIW   **Scheduling and resource reservations**

↓

Execution

Superscalar vs VLIW
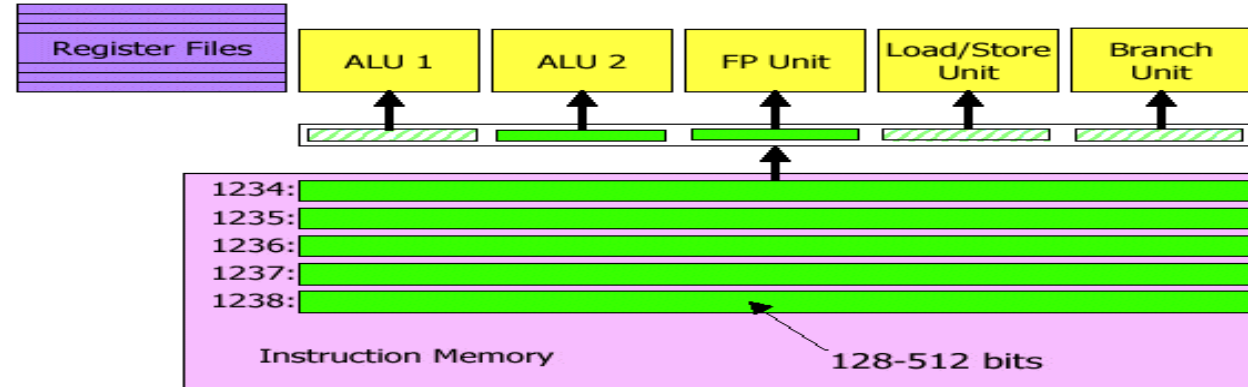
Why Do Dynamic Scheduling?

- Why not just let the compiler schedule code?

- Not all stalls are predictable

  - e.g., cache misses

- Can't always schedule around branches

  - Branch outcome is dynamically determined

- Different implementations of an ISA have different latencies and hazards

Does Multiple Issue Work?

- Yes, but not as much as we'd like

- Programs have real dependencies that limit ILP

- Some dependencies are hard to eliminate
    - e.g., pointer aliasing

- Some parallelism is hard to expose
    - Limited window size during instruction issue

- Memory delays and limited bandwidth
    - Hard to keep pipelines full

- Speculation can help if done well

Multithreading

- Performing multiple threads of execution in parallel
  - Replicate registers, PC, etc.
  - Fast switching between threads
- Fine-grain multithreading
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
- Coarse-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)

Simultaneous Multi-Threading [Eggers, et al.]
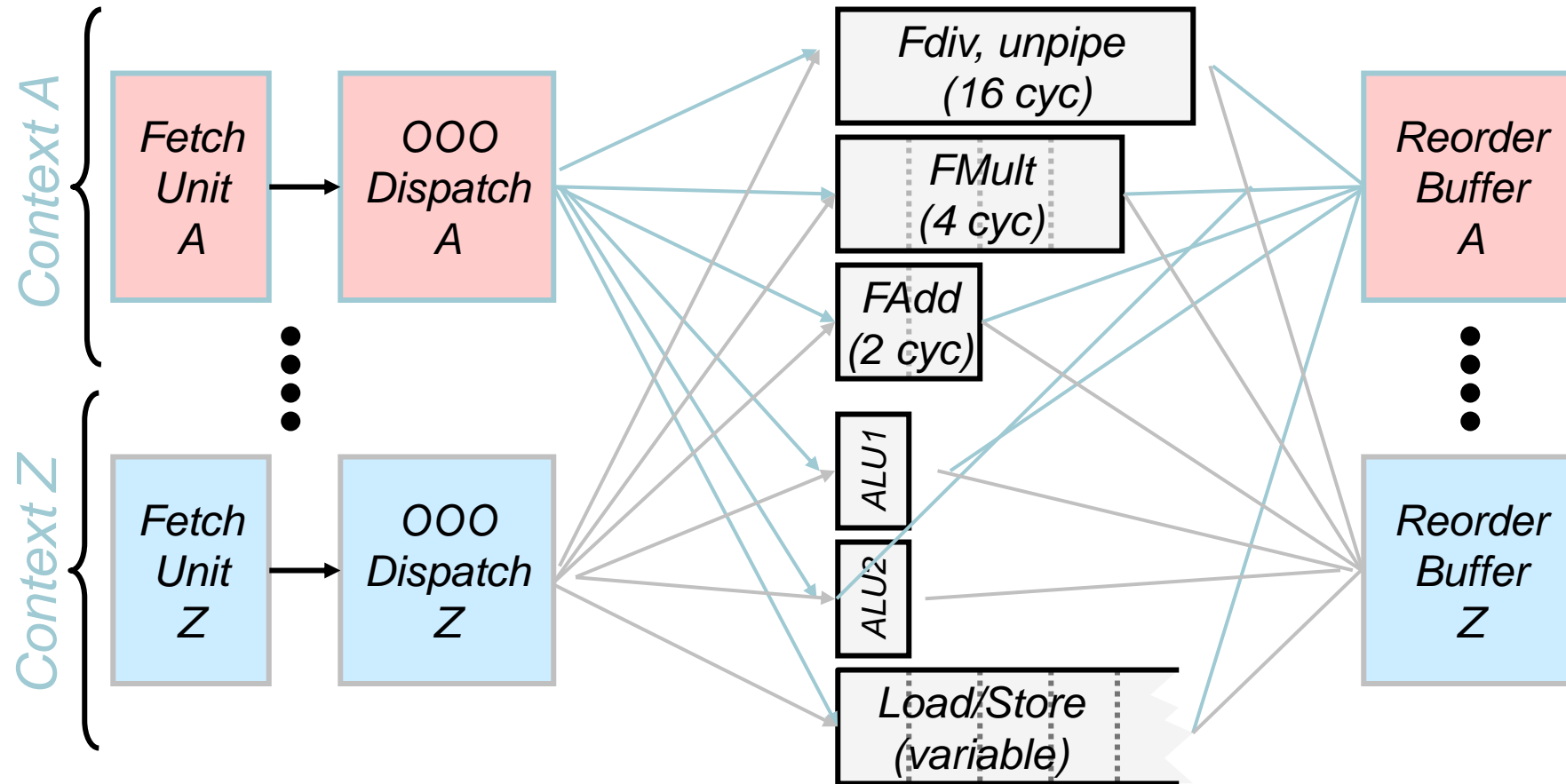
- Se comparten dinámicamente unidades funcionales entre múltiple threads
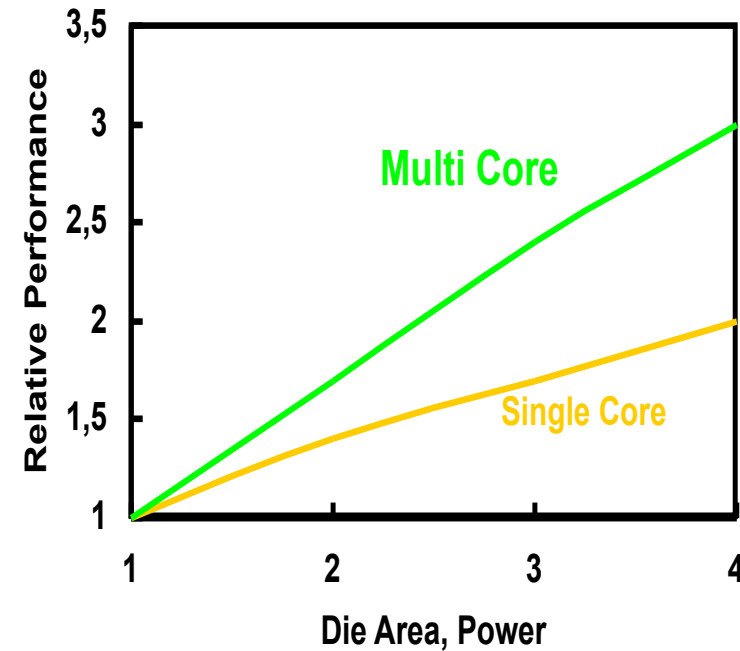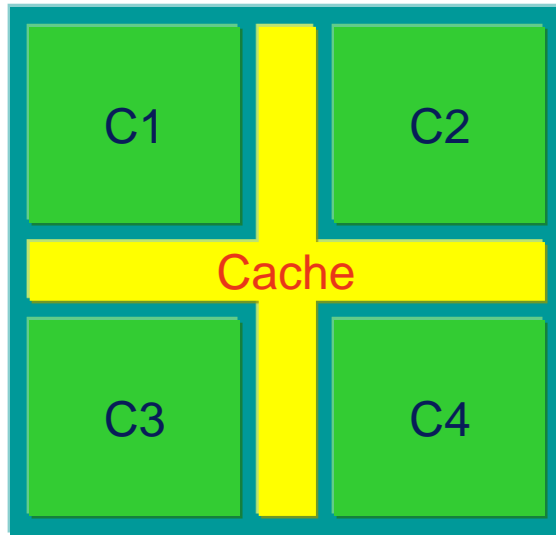 ⇒ *mayor utilización* ⇒ *mayor rendimiento*

Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HT
  - Two threads: duplicated registers, shared function units and caches

TLP: Thread Level Parallelism

Superscalar Fine-Grained Coarse-Grained Multiprocessing Simultaneous Multithreading

Time (processor cycle)

Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Idle slot

Multicore Processors

- Multi-core, each core Multi-threaded
- Shared cache and front side bus
- Each core has different Vdd & Freq

Multi-Core

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

- **Many new applications are multi-threaded, suitable for multi-core.**
    - Ex. Multimedia applications.

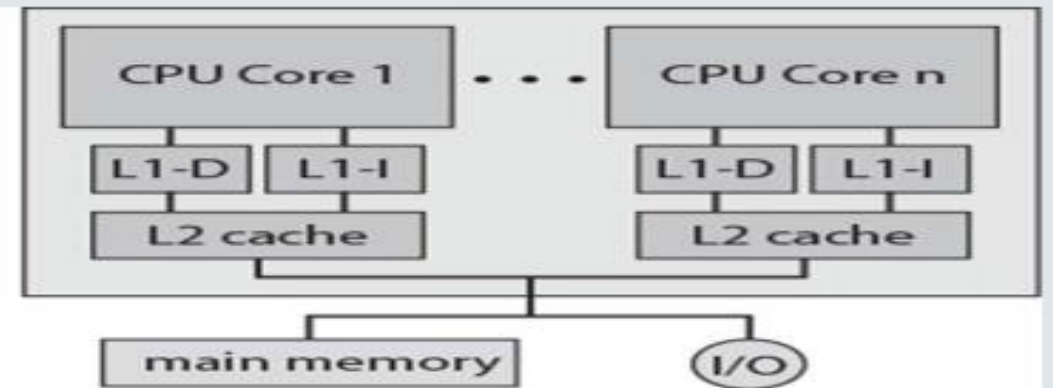- **Integration of multiple processor cores on a single chip**
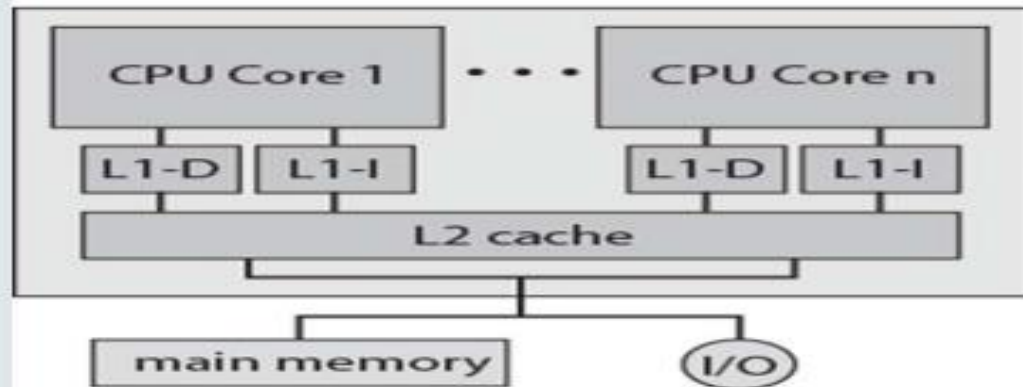
A simple Multi-Core model
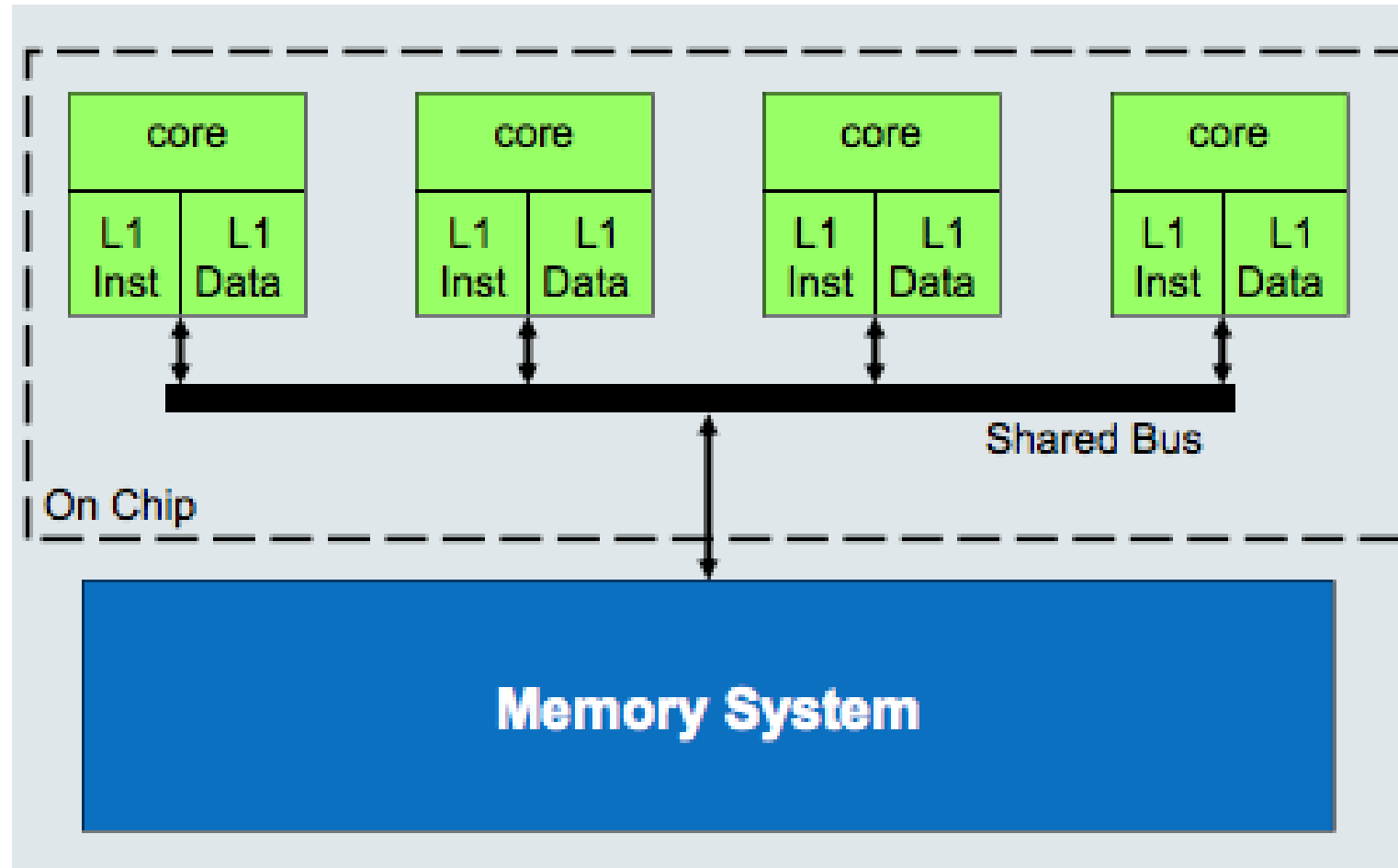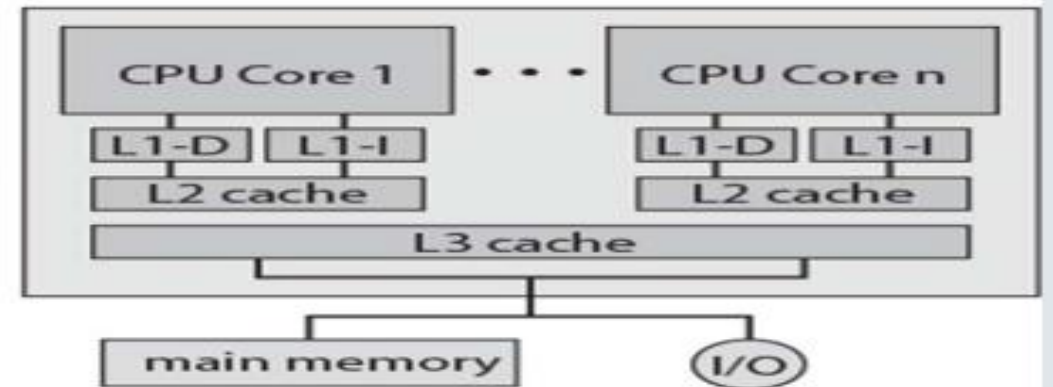
Alternative models
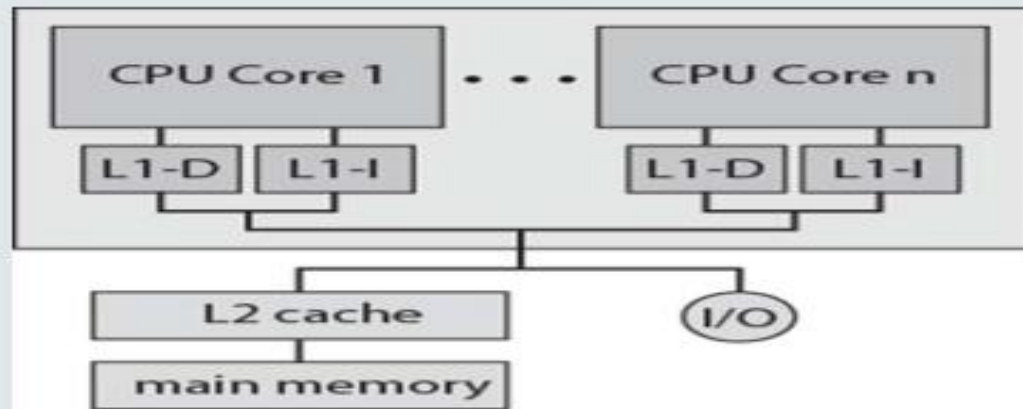
(a) Dedicated L1 cache
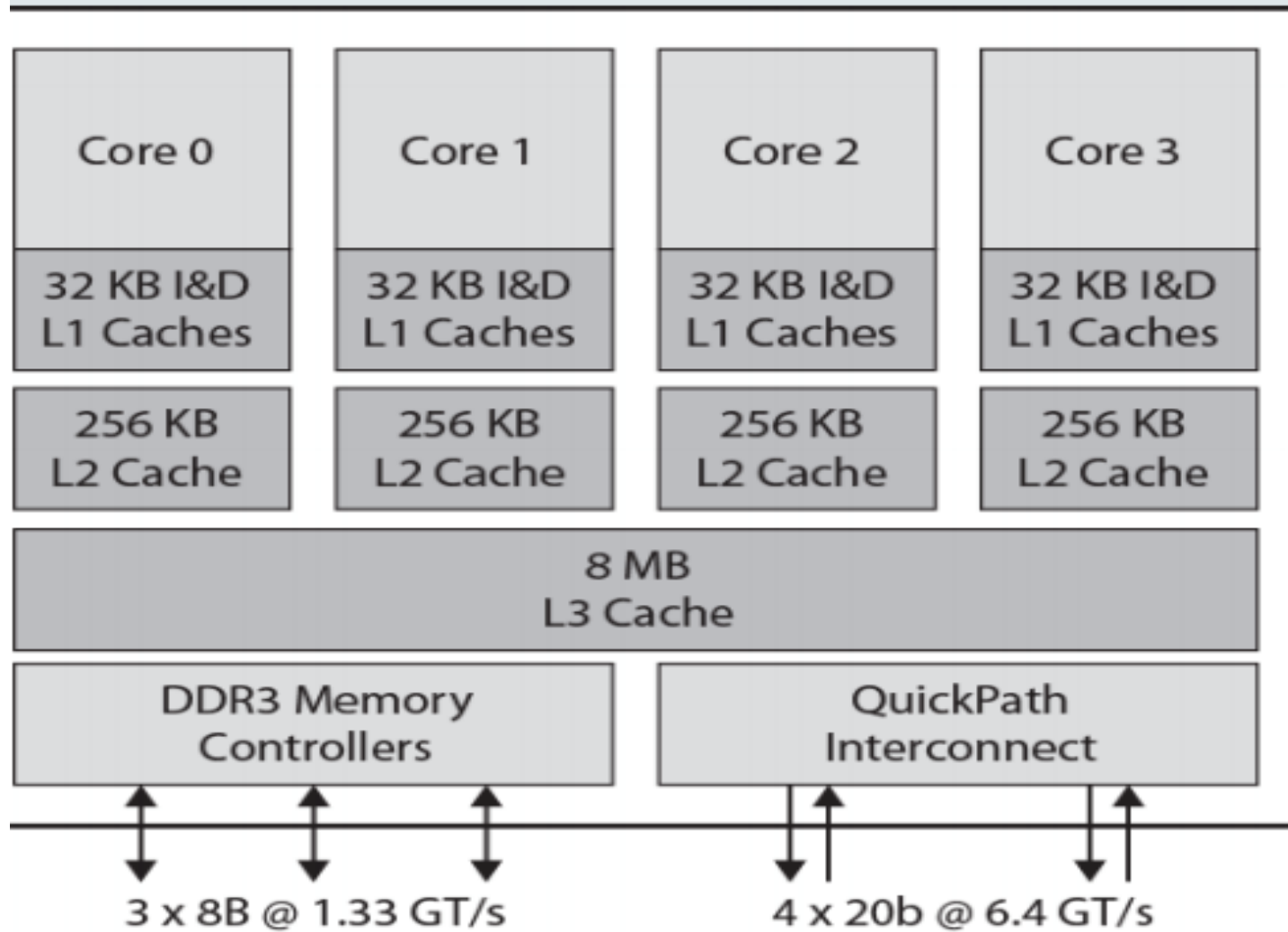ARM11 MP-Core

(b) Dedicated L2 cache
AMD Opteron

(c) Shared L2 cache
Intel Core Duo

(d) Shared L3 cache
Intel Core i7 and AMD Ryzen 7

Intel Core i7

- Four (or six) identical x86 processors.
- Each with its own L1 split, and L2 unified caches.
- Shared on-chip L3 cache to speed up inter-processor communication
- High speed link between processor chips.
- Up to 4 GHz clock frequency.

Programming for Multi-Core

There must be many threads or processes:

- Multiple applications running on the same machine.
  - Multi-tasking is becoming very common.
  - OS software tends to run many threads as a part of its normal operation.
- OS scheduler should map threads to different cores:
  - To balance the work load; or
  - To avoid hot spots due to heat generation.