Scuola Superiore
Sant'Anna
di Studi Universitari e di Perfezionamento

UNIVERSITÀ DI PISA

Master Program (Laurea Magistrale) in Computer Science and Networking

High Performance Computing Systems and Enabling Platforms

Marco Vanneschi

1. Prerequisites Revisited

Contents

1. System structuring by levels
2. Firmware machine level
3. Assembler machine level, CPU, performance parameters
4. Memory Hierarchies and Caching
5. Input-Ouput

Scuola Superiore Sant'Anna
di Studi Universitari e di Perfezionamento

Università di Pisa

Master Program (Laurea Magistrale) in Computer Science and Networking

High Performance Computing Systems and Enabling Platforms

Marco Vanneschi

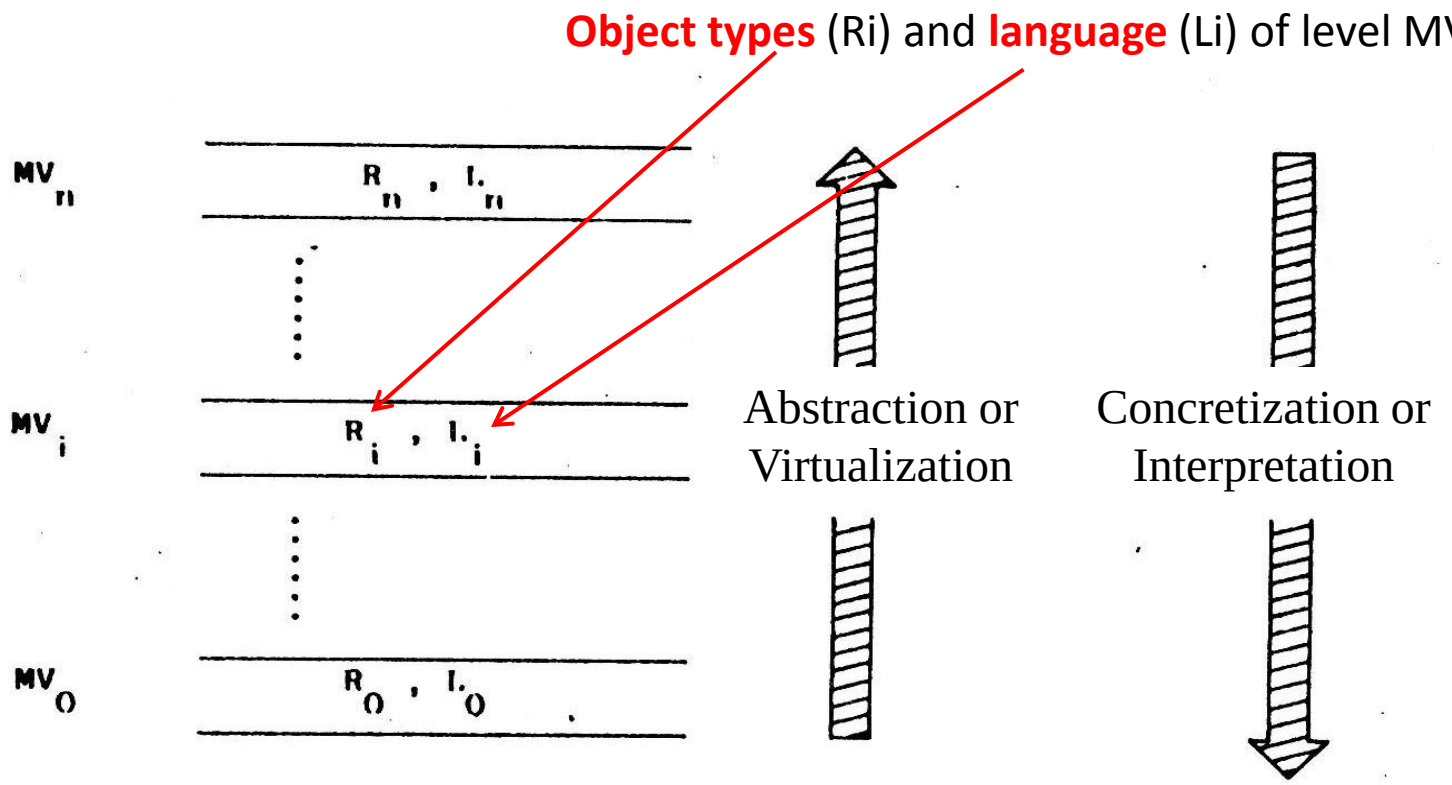
1. Prerequisites Revisited

1.1. System structuring by levels

Structured view of computing architectures

- System structuring:
  - by ***Levels***:
    - vertical structure, hierarchy of interpreters
  - by ***Modules***:
    - horizontal structure, for each level (e.g. processes, processing units)
  - Cooperation between modules and ***Cooperation Models***
    - message passing, shared object, or both
- Each level (even the lowest ones) is associated a programming *language*
- At each level, the organization of a system is derived by, and/or is strongly related to, the *semantics of the primitives* (commands) of the associated language
  - "the hardware – software interface"

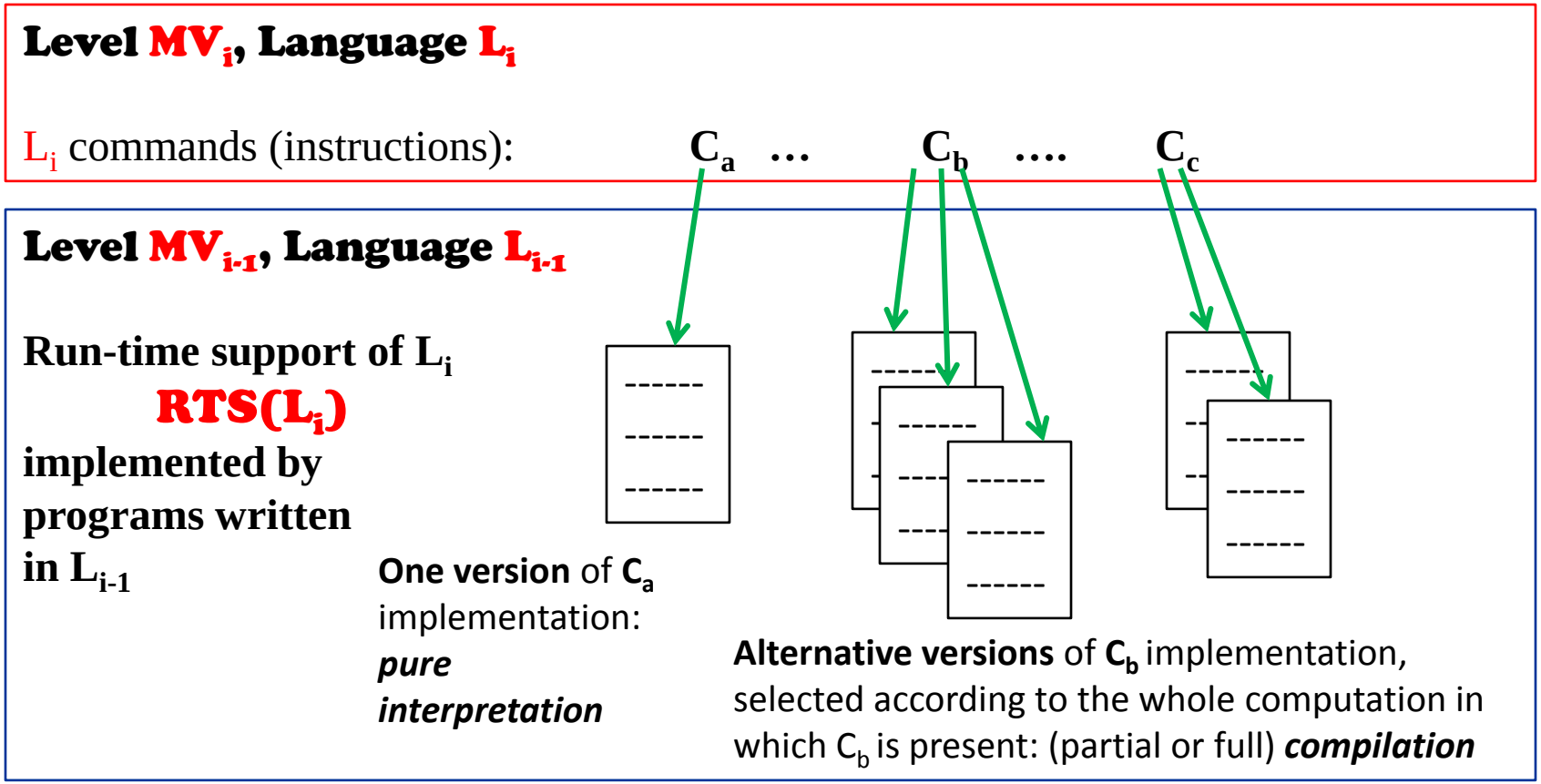
System structuring by hierarchical levels



- "Onion like" structure
- Hierarchy of **Virtual Machines** (MV)
- Hierarchy of **Interpreters**: commands of MVi language are interpreted by programs at level MVj, where $j < i$ (often: $j = i - 1$)

Compilation and interpretation

The implementation of some levels can exploit optimizations through a *static analysis* and *compilation* process



Very simple example of optimizations at compile time

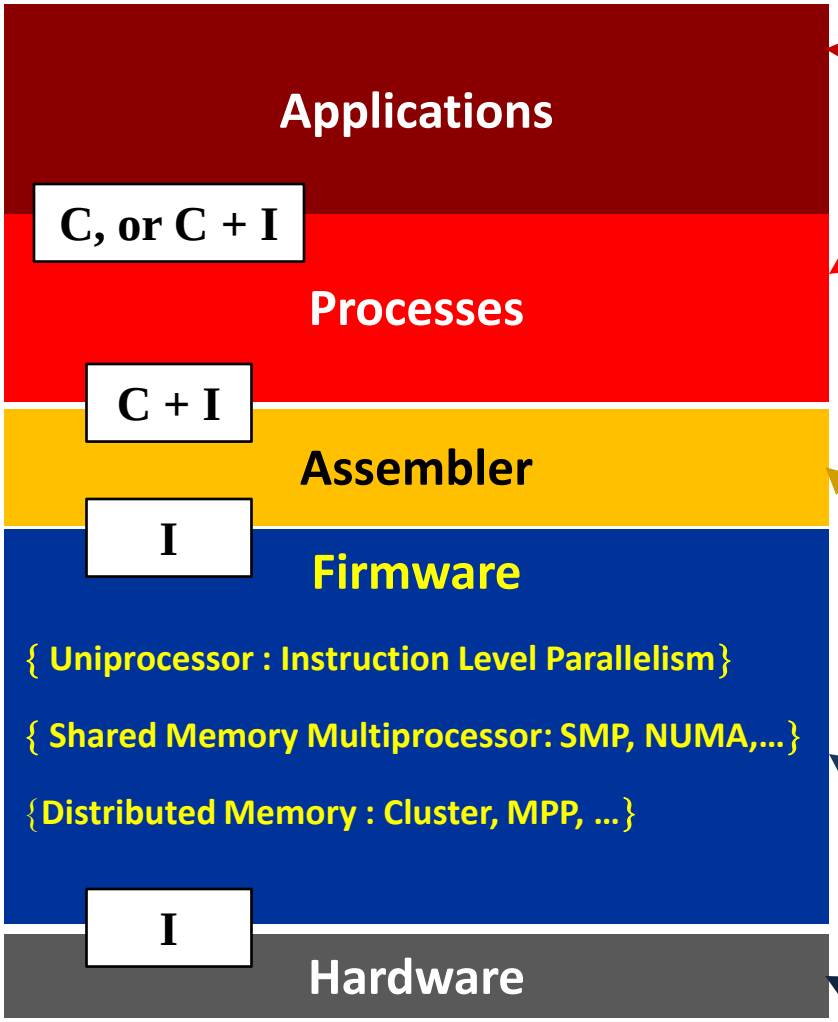
```
int A[N], B[N], X[N];
        for (i = 0; i < N; i++)
                X[i] = A[i] * B[i] + X[i]
```

```
int A[N], B[N]; int x = 0;
        for (i = 0; i < N; i++)
                x = A[i] * B[i] + x
```

- Apparently similar program structures
- **A static analysis of the programs (data types manipulated inside the *for* loop) allows the compiler to understand important differences and to introduce optimizations**
- *First example*: at *i-th* iteration of ***for*** command, a memory-read and a memory-write operations of ***X[i]*** must be executed
- *Second example*: a temporary variable for ***x*** is initialized and allocated in a CPU Register (General Register), and only the exit of ***for*** commmand ***x*** value is written in memory
  - 2N – 1 memory accesses are saved
  - what about the effect of caching in the first example ?

Typical Levels

C = compilation
I = interpretation



- Sequential or parallel applications

- Implementation of applications as collection of processes (threads).
- *Concurrent language*.
- **Run-time support** of concurrent language is implemented at the lower levels.
- *Also:* Operating system services.

- Assembly language: Risc *vs* Cisc
- Intermediate level: *does it favour optimizations ?*
- Could it be eliminated ?

- *The true architectural level.*
- Microcoded interpretation of assembler instructions.
- This level is fundamental (it cannot be eliminated).

- Physical resources: combinatorial and sequential circuit components, physical links

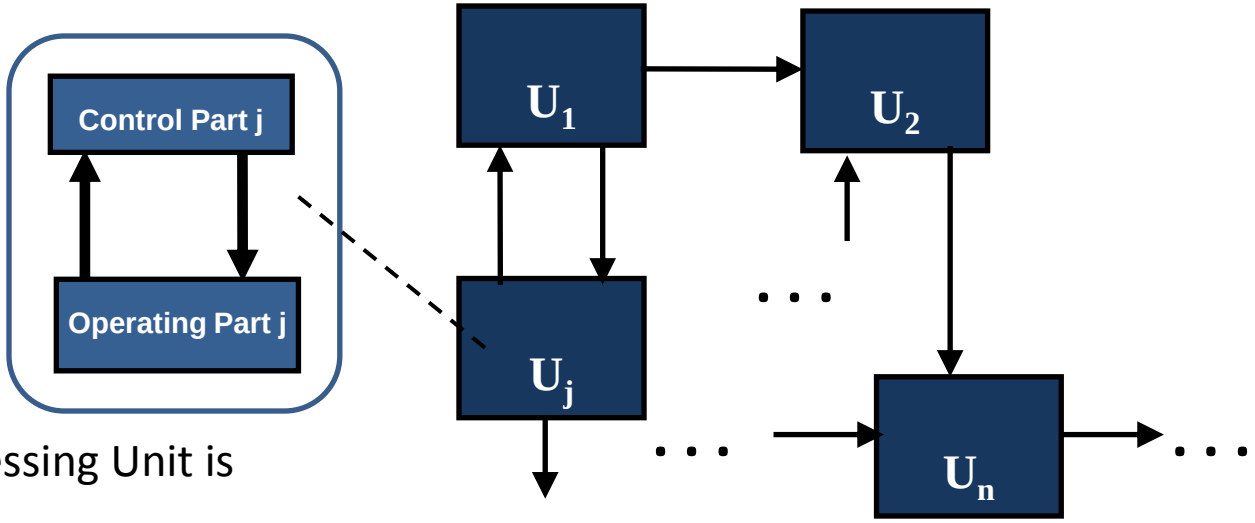
Examples

C-like application language

- Compilation of the majority of sequential code and data structures
  - Intensive optimizations according to the assembler – firmware architecture
    - Memory hierarchies, Instruction Level Parallelism, co-processors
- Interpretation of dynamic memory allocation and related data structures
- Interpretation of interprocess communication primitives
- Interpretation of invocations to linked services (OS, networking protocols, and so on)

Firmware level

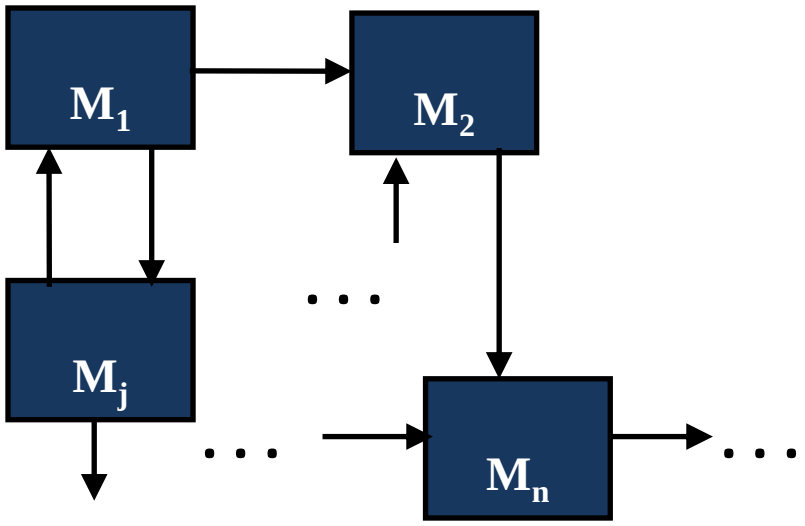
- At this level, the system is viewed as a collection of cooperating modules called PROCESSING UNITS (simply: units).



- Each Processing Unit is
  - autonomous
    - has its own control, i.e. it has self-control capability, i.e. it is an **active computational entity**
  - described by a sequential program, called microprogram.
- Cooperation is realized through COMMUNICATIONS
  - Communication channels *implemented* by physical links and a firmware protocol.
- Parallelism between units.

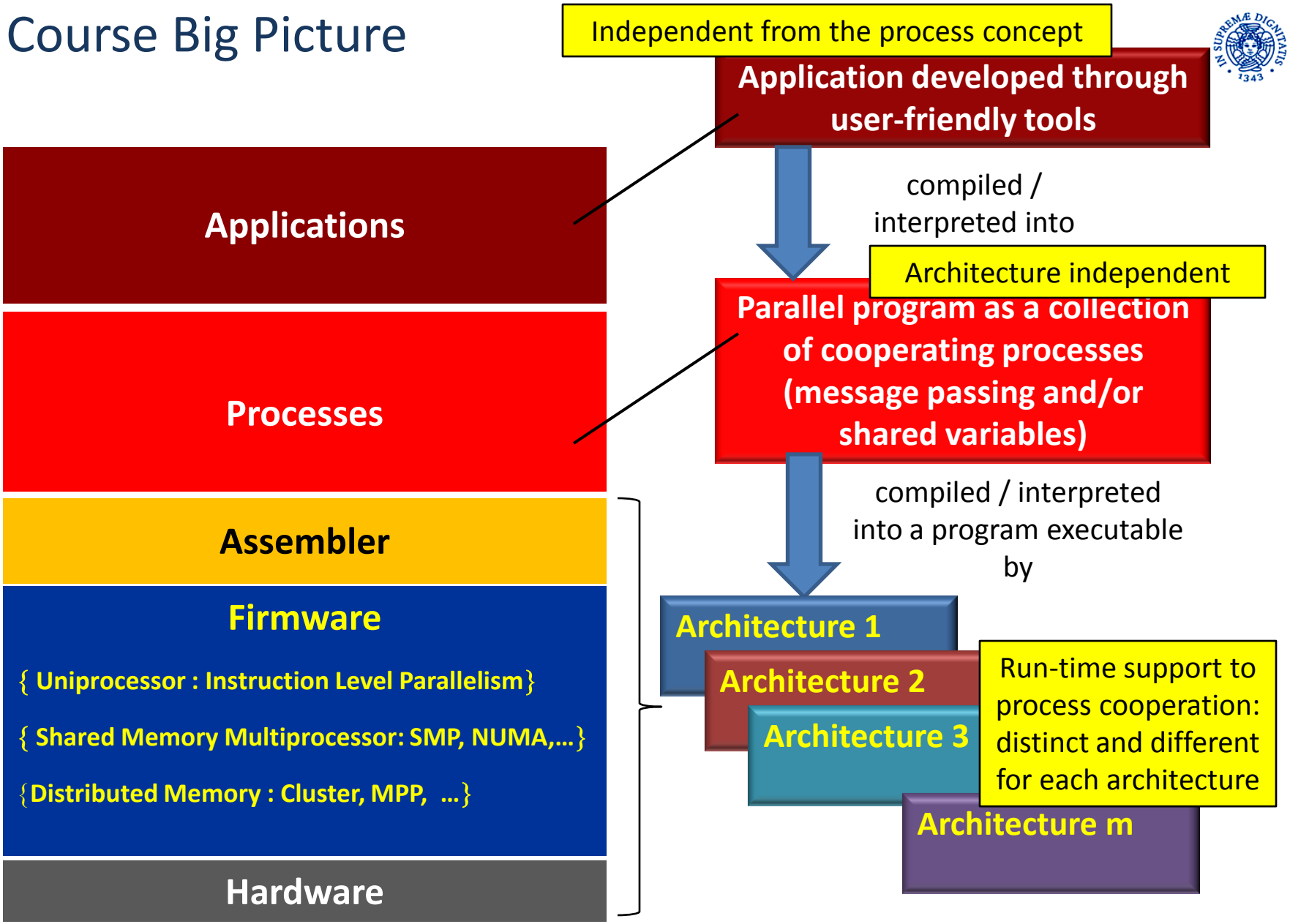
Modules at different levels

- The same definition of Processing Unit extends to **Modules** at any level:



- Each Module is
  - autonomous
    - has its own control, i.e. it has self-control capability, i.e. it is an **active computational entity**
  - described by a sequential program, e.g. a process or a thread at the Process Level.
- **Cooperation** is realized through COMMUNICATIONS and/or SHARED OBJECTS
  - depending on the level: at some levels *both* cooperation model are feasible (Process), in other cases only communication is a feasible module in a primitive manner (Firmware).
- Parallelism between Modules.

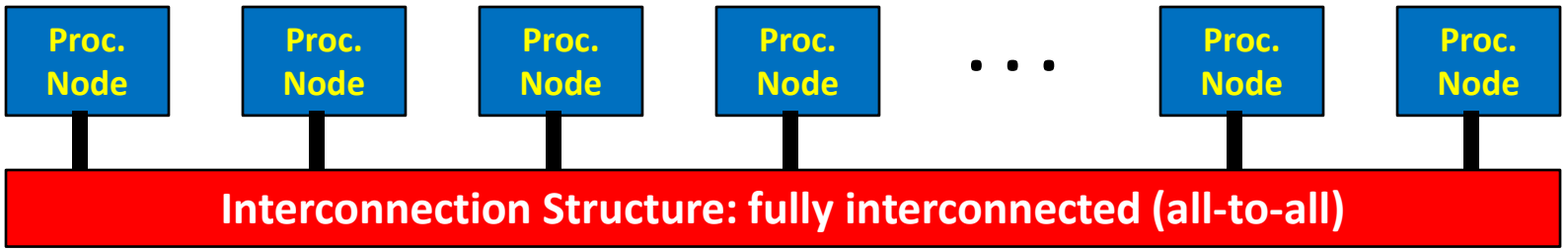
Course Big Picture



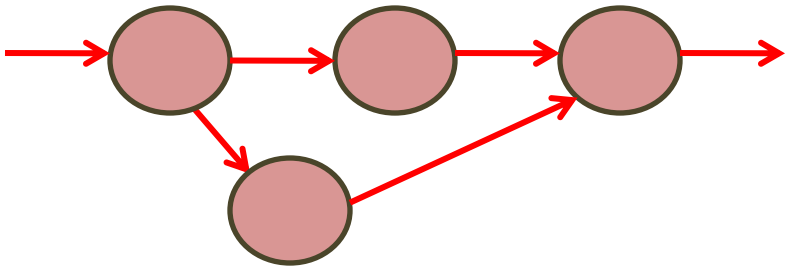
Cost models and abstract architectures

- Performance parameters and *cost models*
  - for each level, a cost model to evaluate the system performance properties
    - Service time, bandwidth, efficiency, scalability, latency, response time, …, mean time between failures, …, power consumption, …
- Static *vs* dynamic techniques for performance optimization
  - the importance of **compiler technology**
  - **abstract architecture** *vs* **physical/concrete architecture**
    - *abstract architecture*: a semplified view of the concrete one, able to describe the essential performance properties
  - **relationship between the abstract architecture and the cost model**
  - in order to perform optimizations, *the compiler "sees" the abstract architecture*
    - often, the compiler *simulates* the execution *on* the abstract architecture

Example of abstract architecture

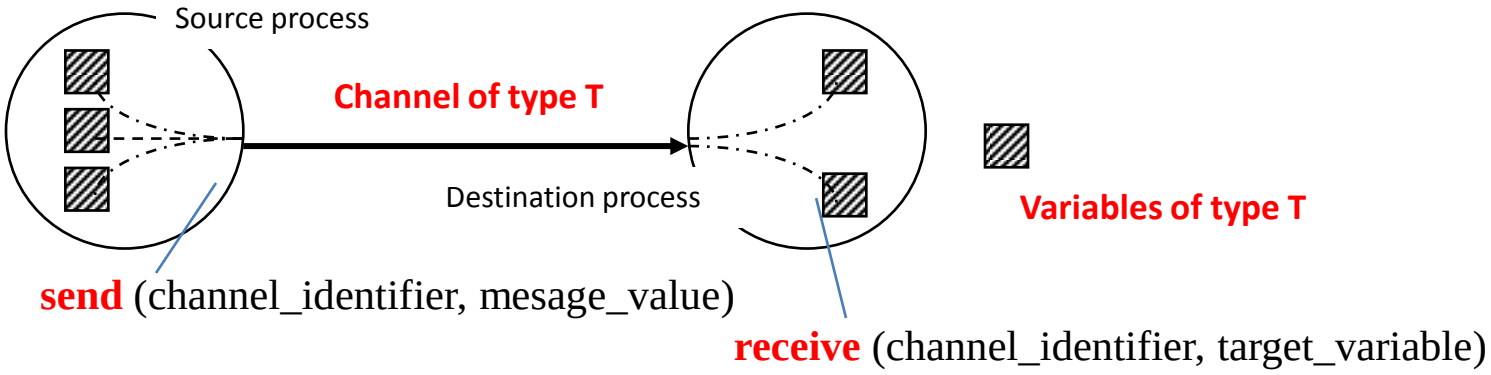


- **Processing Node**= (CPU, memory hierarchy, I/O)
  - Same characteristics of the concrete architecture node
- Parallel program allocation onto the Abstract Architecture: **one process per node**
  - Interprocess communication channels: one-to-one correspondence with the Abstarct Architecture interconnection network channels



Process Graph for the parallel program = **Abstract Architecture Graph** (same topology)

Cost model for interprocess communication



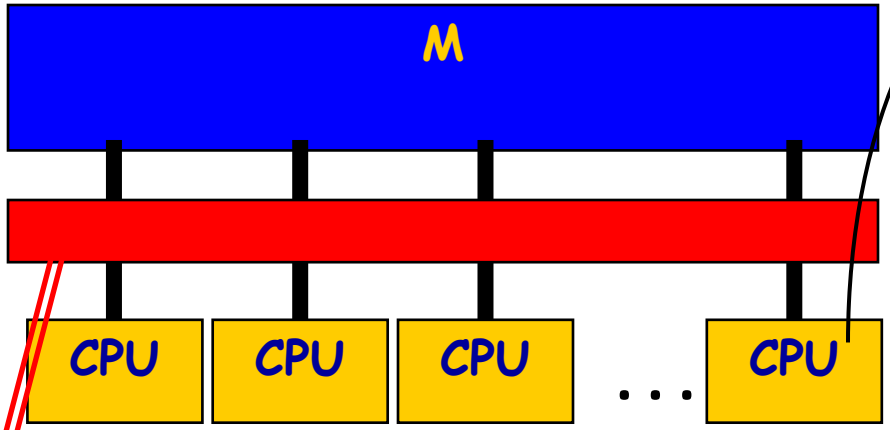
$$T_{send} = T_{setup} + L * T_{transm}$$

- T_{send} = **Average latency of interprocess communication**
  - delay needed for copying a message_value into the target_variable
- **L** = Message length
- T_{setup}, T_{transm}: known parameters, **evaluated for the concrete architecture**
- *Moreover,* the cost model must include the characteristics of possible **overlapping** of communication and internal calculation

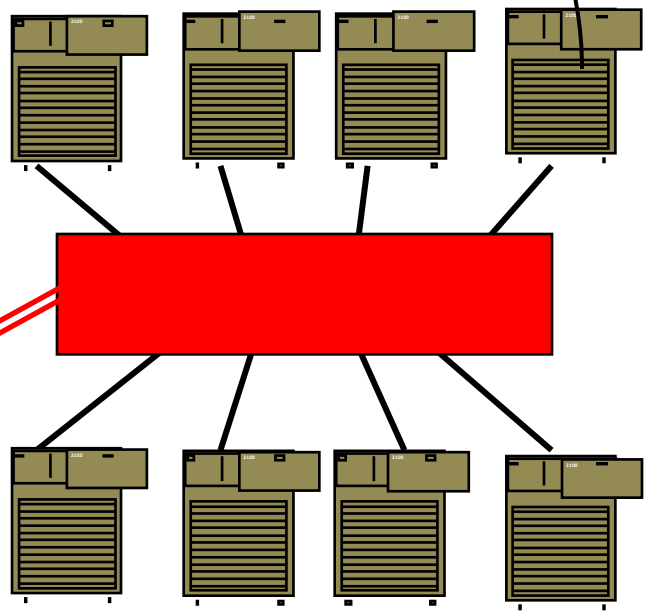
Parameters T_{setup}, T_{transm} evaluated as functions of several characteristics of the concrete architecture

Memory access time, interconnection network routing and flow-control strategies, CPU cost model, and so on

Instruction Level Parallelism CPU (pipeline, superscalar, multithreading, …)

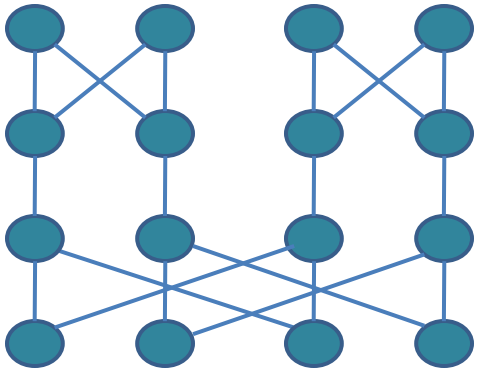


Shared memory multiprocessor



Distributed memory multicomputer: PC cluster, Farm, Data Centre, …

"Limited degree" Interconnection Network ("one-to-few")



Abstract architectures and cost models