The Abdus Salam International Centre for Theoretical Physics

Modern Computer Architectures

Ivan Girotto – igirotto@ictp.it

Information & Communication Technology Section (ICTS)

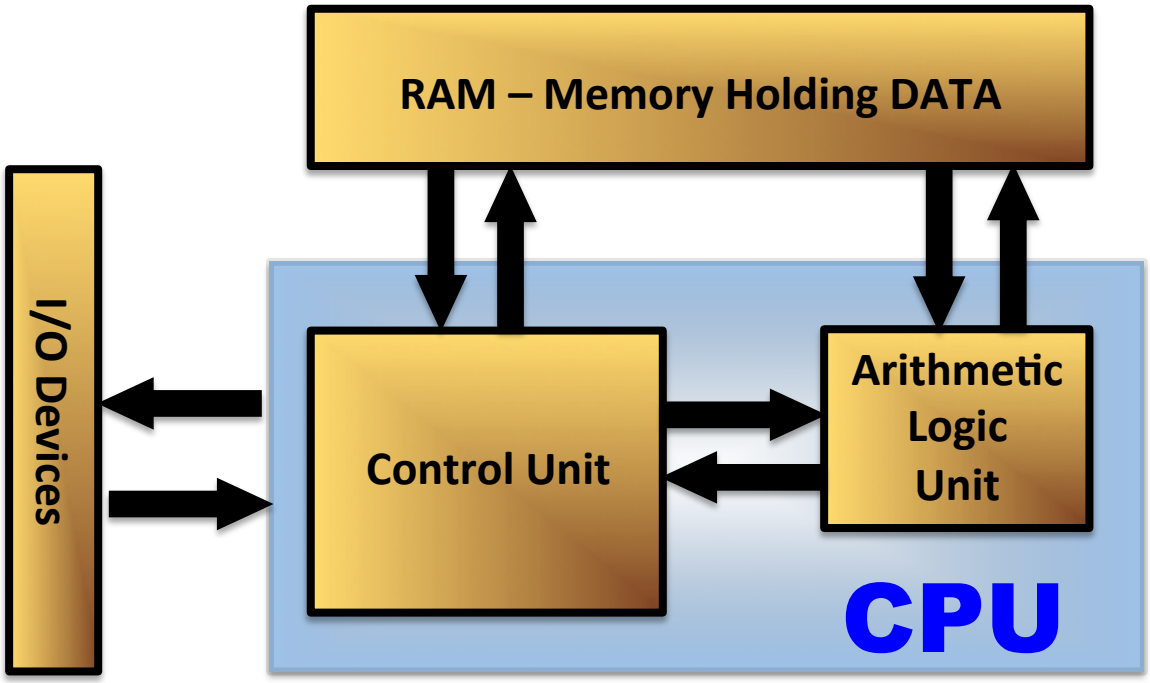
International Centre for Theoretical Physics (ICTP)

Performance Metrics

- When all CPU component work at maximum speed that is called *peak of performance*
  - Tech-spec normally describe the theoretical peak
  - Benchmarks measure the real peak
  - Applications show the real performance value
- CPU performance is measured as:
  - Floating point operations per seconds FLOP/s
- The real performance is in many cases mostly related to the memory bandwidth (Bytes/s) and the exploitation of the parallelism within the CPU

John Von Neumann

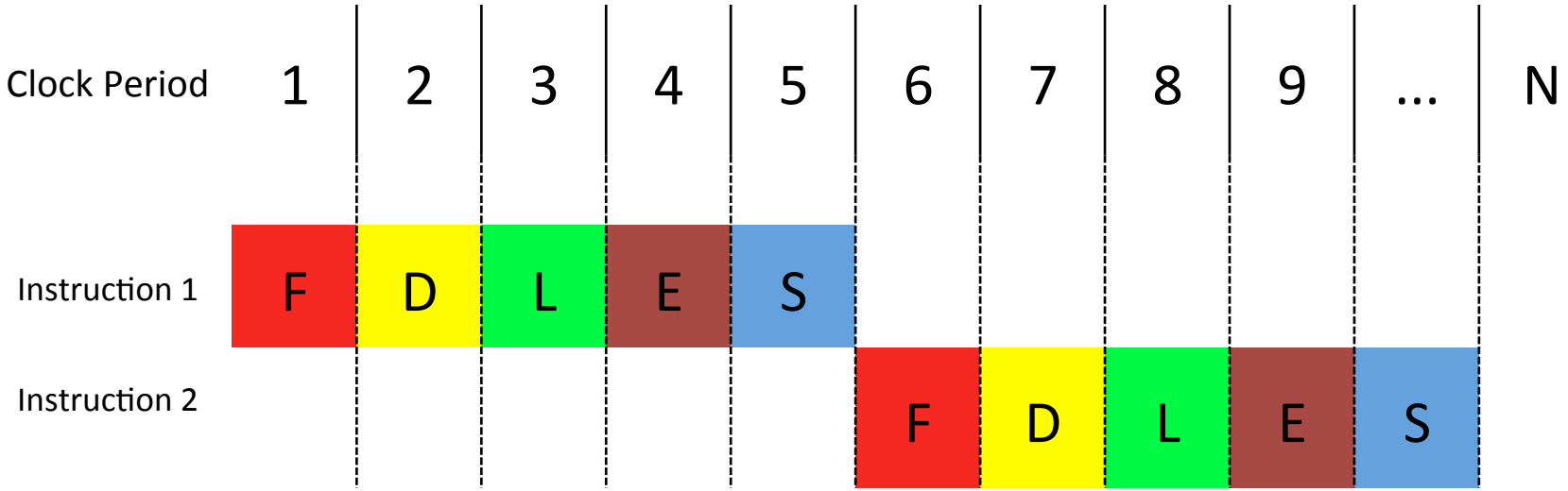
The Classical Model



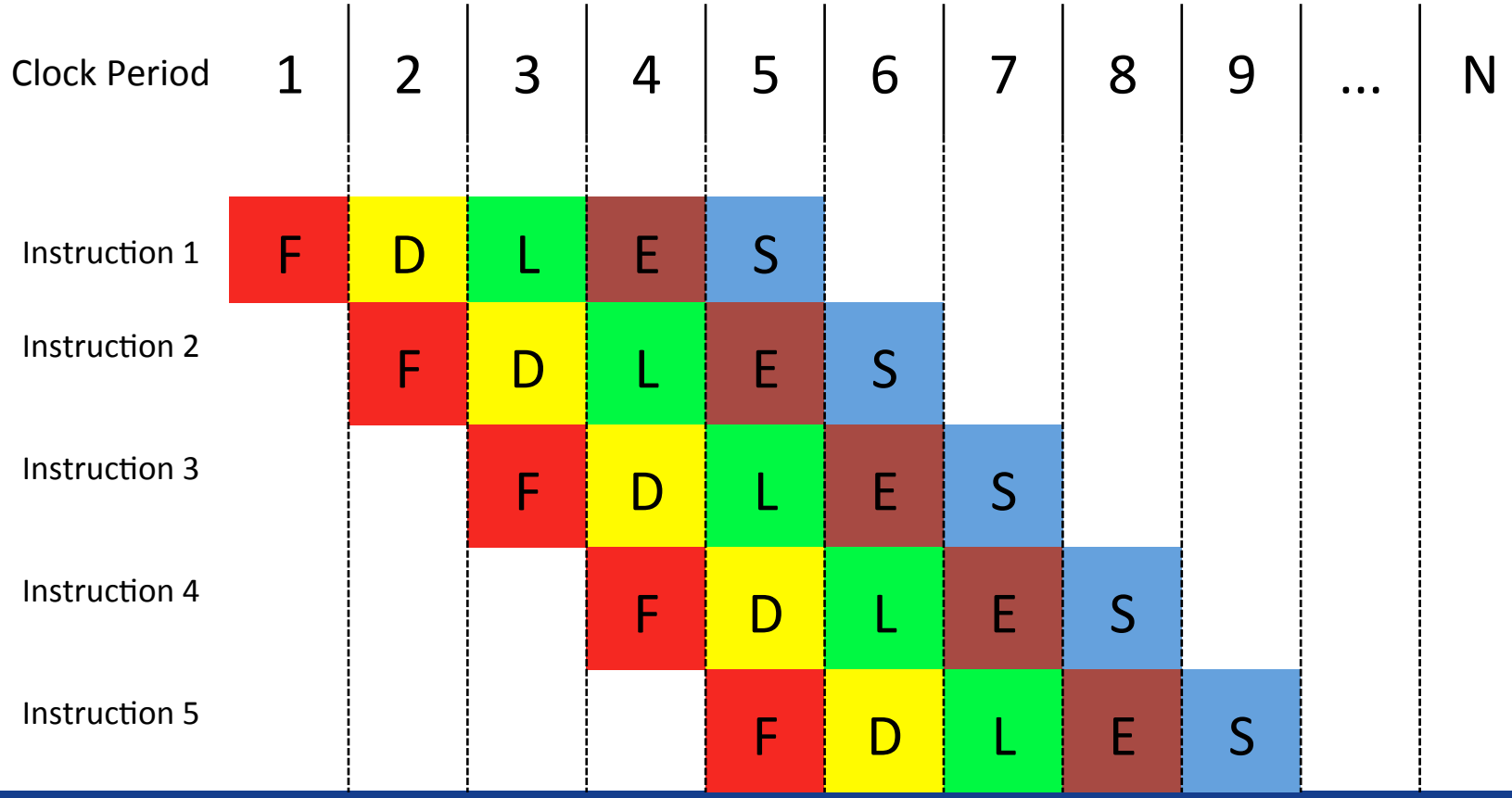
The Instruction Processing Cycle

- Fetch: read the next instruction from memory
  - 001000 00001 00010 0000000100001000
- Decode: operands and operation are decoded
  - add, $r1, $r2, 10
- Load: retrieve the data from memory to registers
- Execute: execute the instruction
  - $r1 = 4500 + 10
- Store: store the results

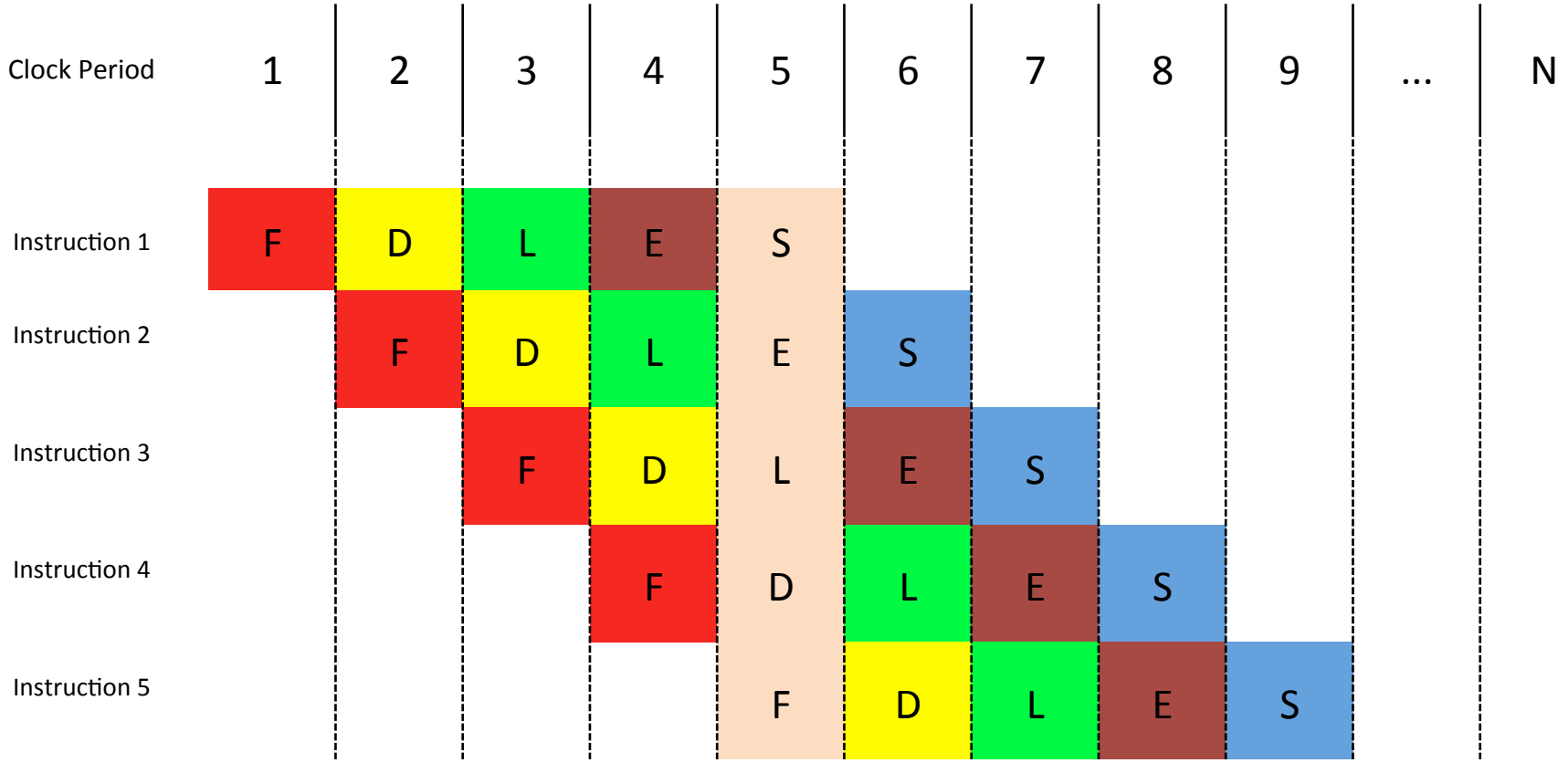
Sequential Processing



Pipelining

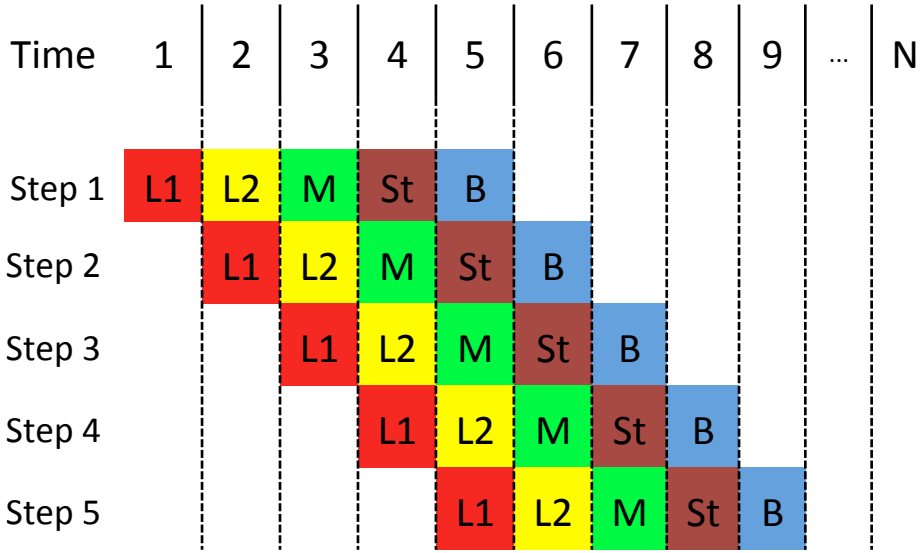

Clock Period
1
2
3
4
5
6
7
8
9
...
N
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
F
D
L
E
S

Pipelining


Clock Period
1
2
3
4
5
6
7
8
9
...
N
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
F
D
L
E
S

Superscalaring

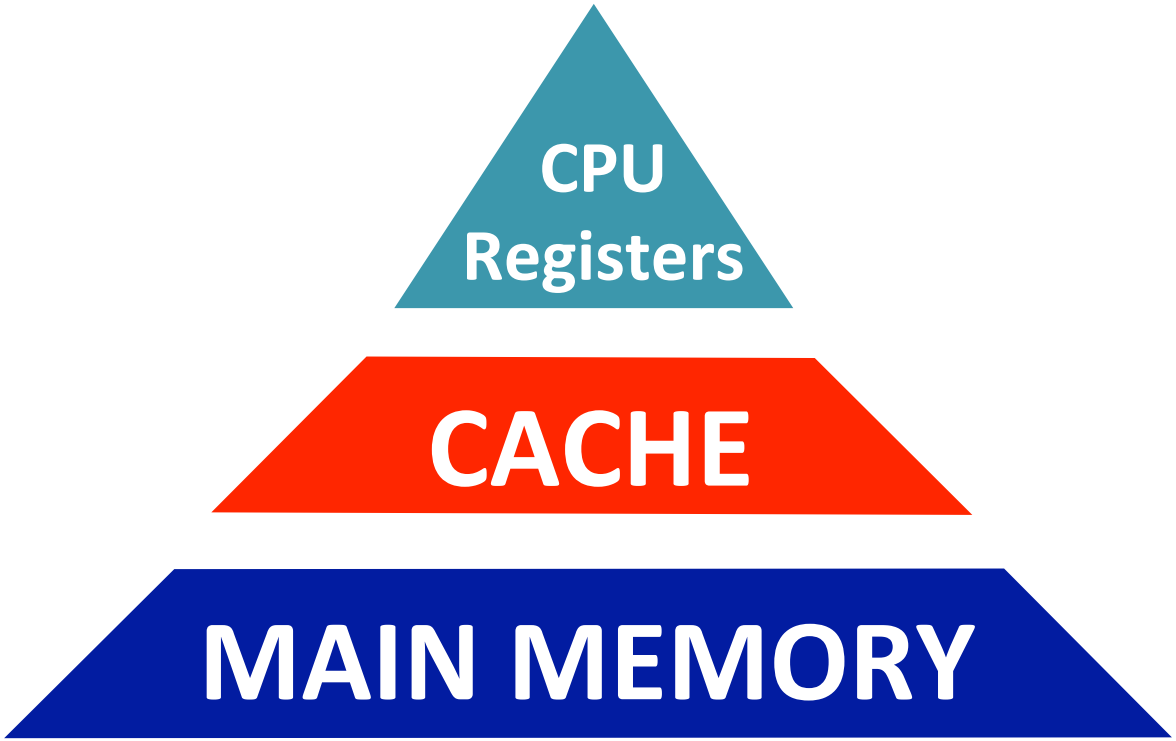
Loops and Pipeline

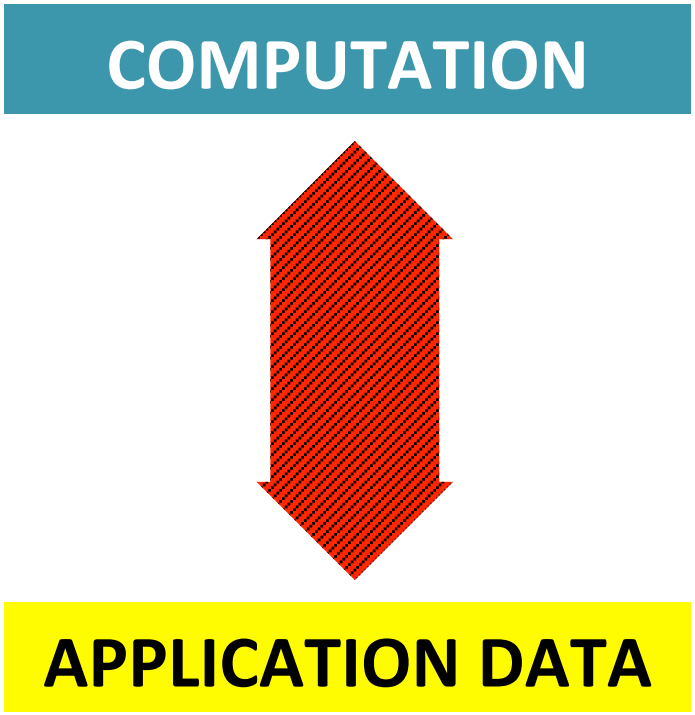



```
for( i = 0; i < N; i += 1 )
{
    A[i] = s * A[i]
}
```

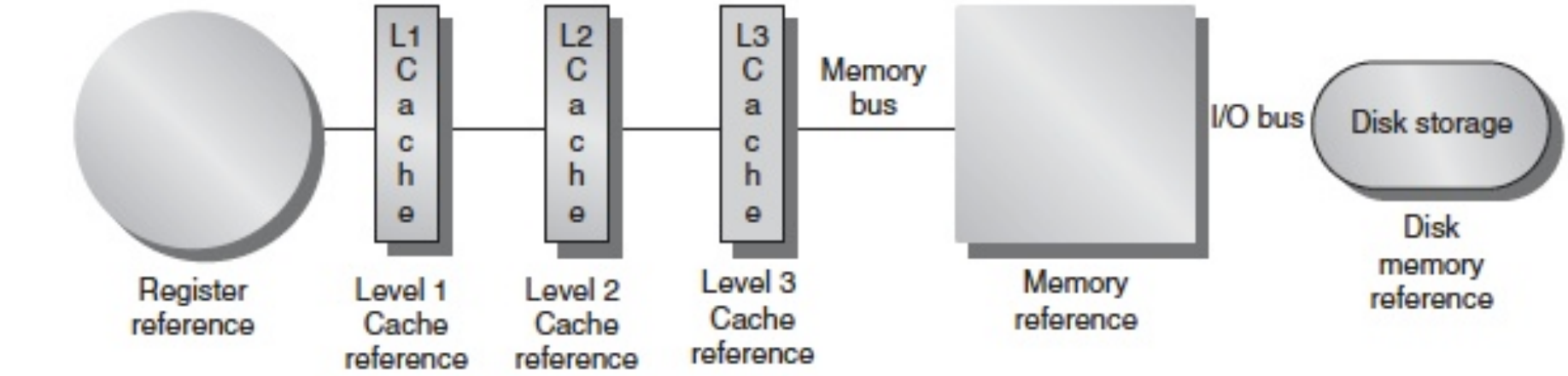
```
Loop:   load r1, A(i)
        load r2, s
        mult r3, r2, r1
        store A(i), r3
        branch => loop
```

The CPU Memory Hierarchy





The CPU Memory Hierarchy



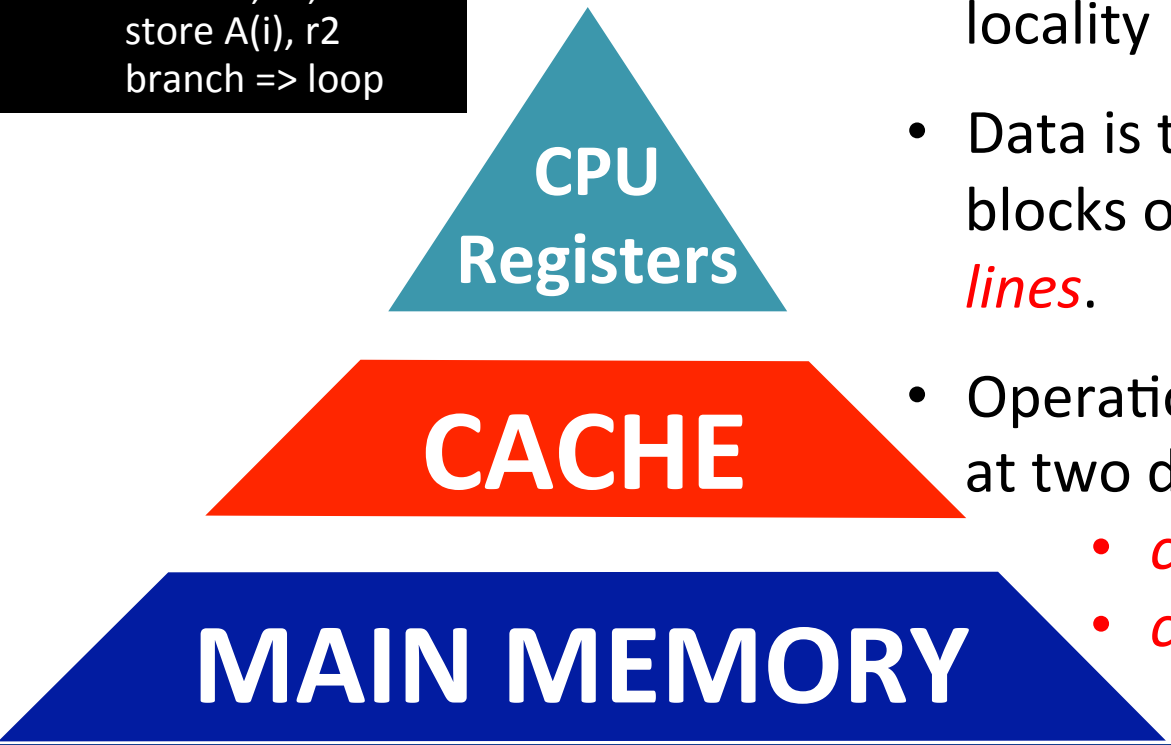
(a) Memory hierarchy for server

Cache Memory

- Expensive (SRAM) high-speed memory
- Relatively low-capacity in regards to RAM
- Cache Memory are for Instructions (i.e., L1I) and for Data (i.e., L1D)
- Modern CPU are designed with several levels of cache memories

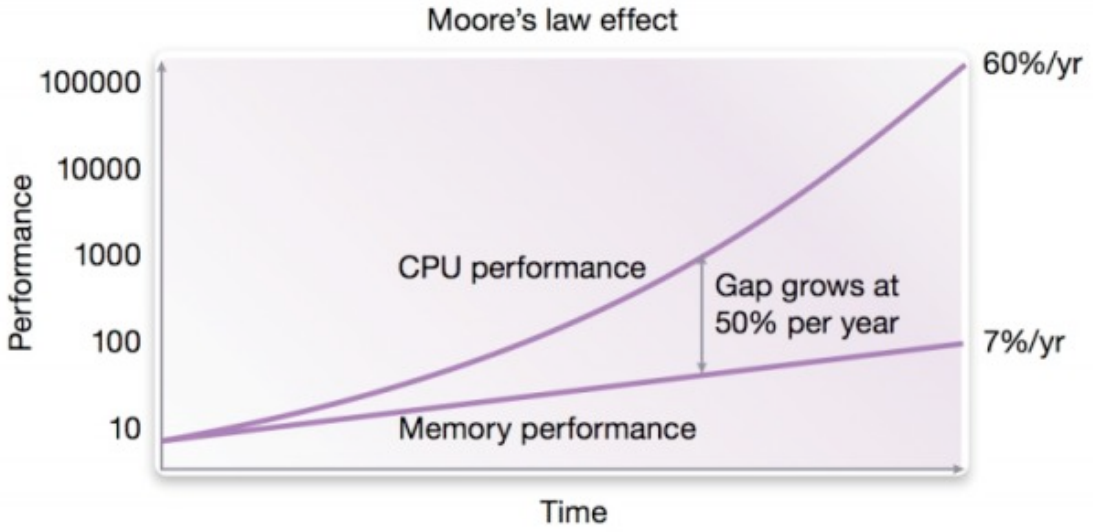
Cache Memory

```
Loop: load r1, A(i)
      load r2, s
      mult r3, r2, r1
      store A(i), r2
      branch => loop
```



- Designed for temporal/spatial locality
- Data is transferred to cache in blocks of fixed size, called *cache lines*.
- Operation of LOAD/STORE can lead at two different scenario:
  - *cache hit*
  - *cache miss*

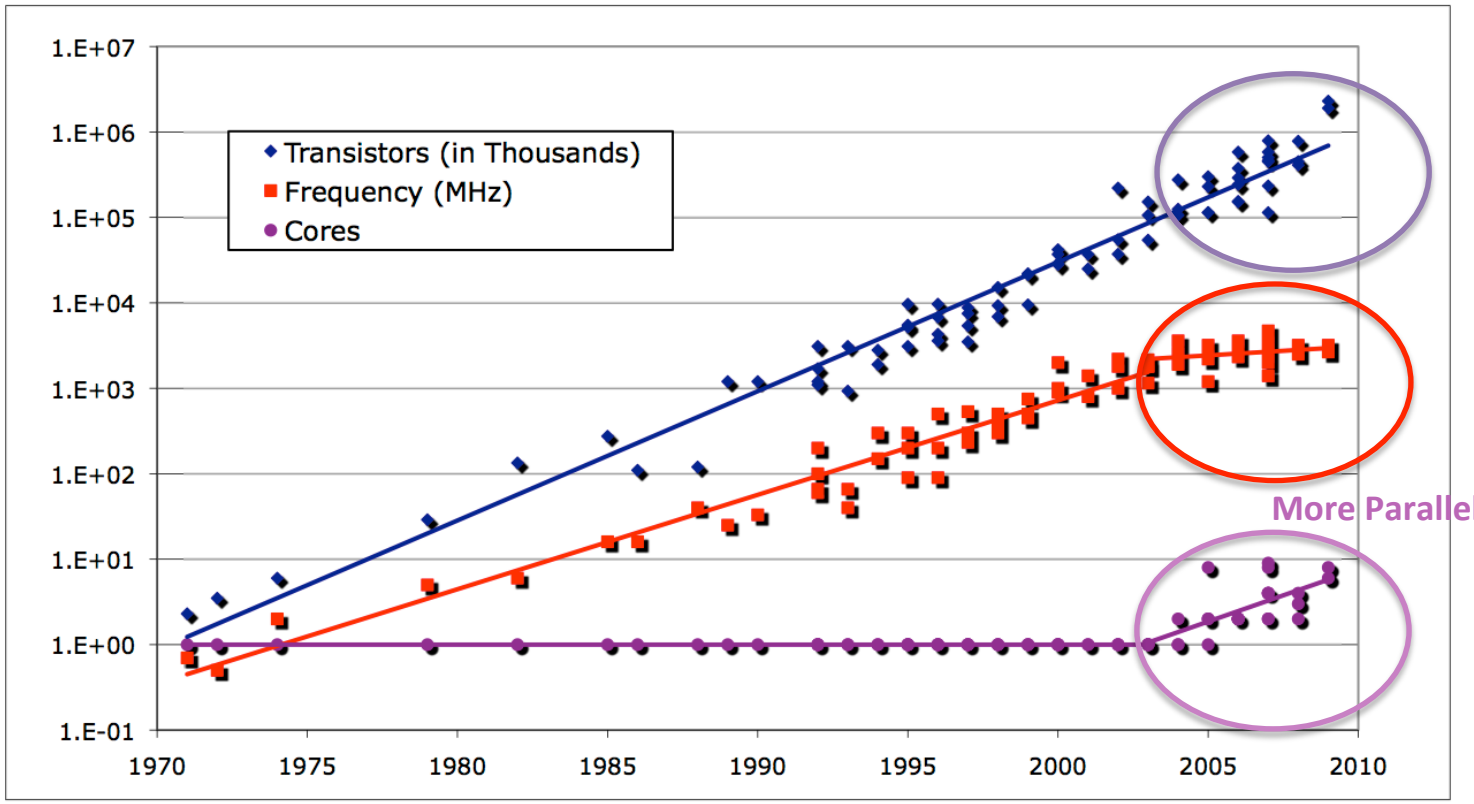
The CPU Memory Hierarchy



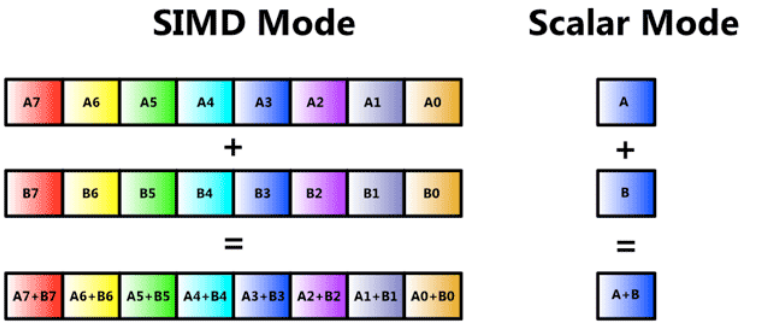
COMPUTATION

APPLICATION DATA

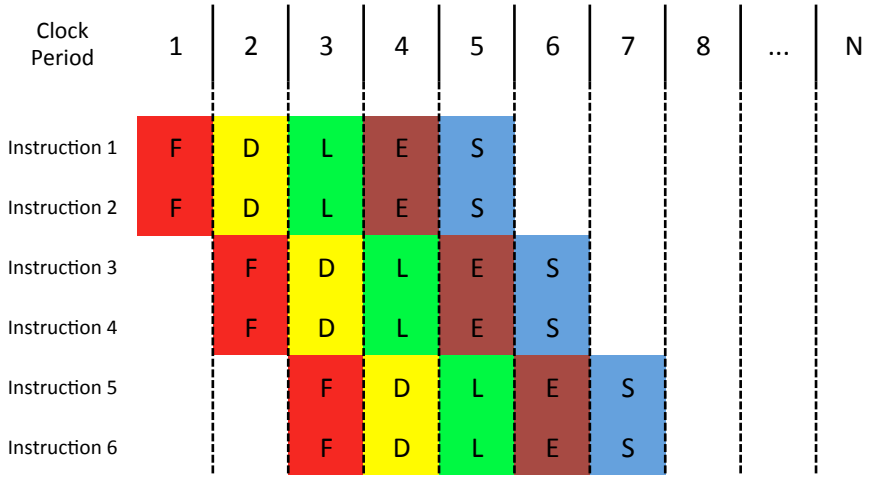
HPC Trend and Moore's Law



To the Extreme - Parallel Inside

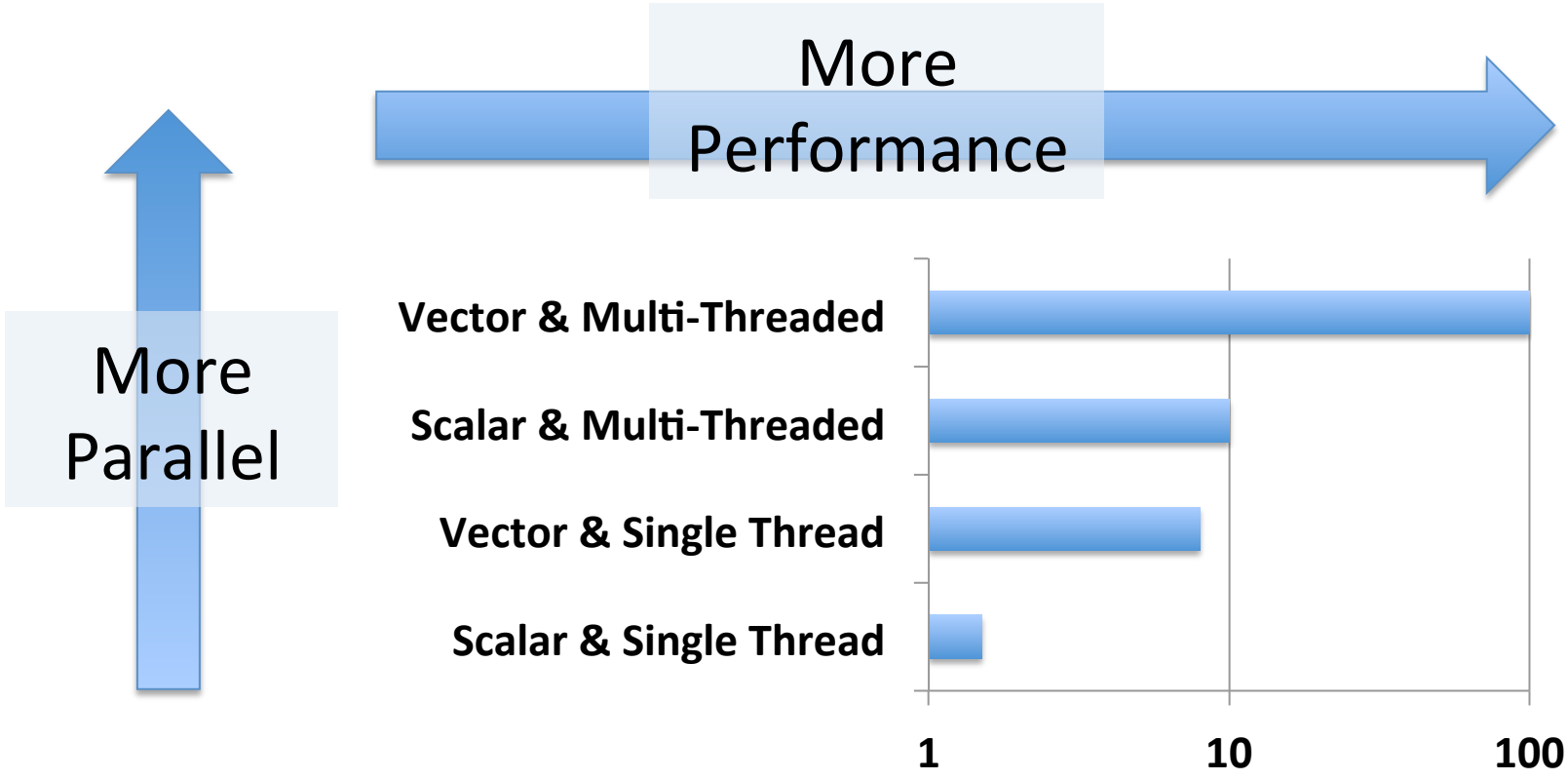


Vector Units for processing multiple data in //



Pipelined/Superscalar design: multiple functional units operate concurrently

Threading and Vectorization



Few basic rules for optimized codes

- Do less work!!
  - Elimination of common sub-expressions
- Avoid expensive operations
  - Reduce your math to cheap operations
  - Avoid branches
- Think as a the compiler works
  - Enhance the compiler

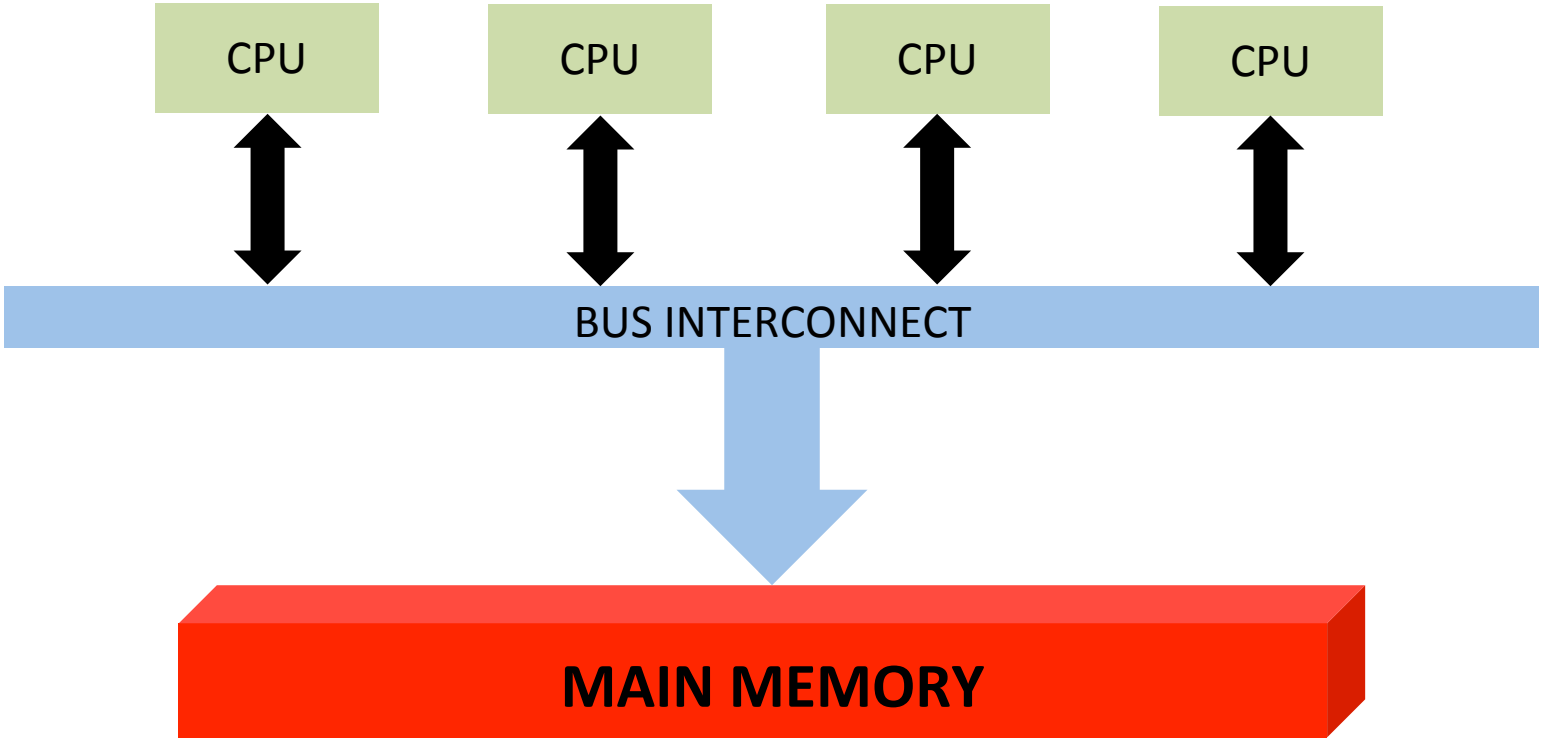
Compiler Optimizations

- Most compilers provides different levels of code optimizations. The compiler option is usually in -O[N] format.
- N goes from 0 to 4/5 levels. Each level introduces new levels of optimization
- “man [compiler command]” provides you a detailed report on all possible options. You can always add/reduce optimization options.

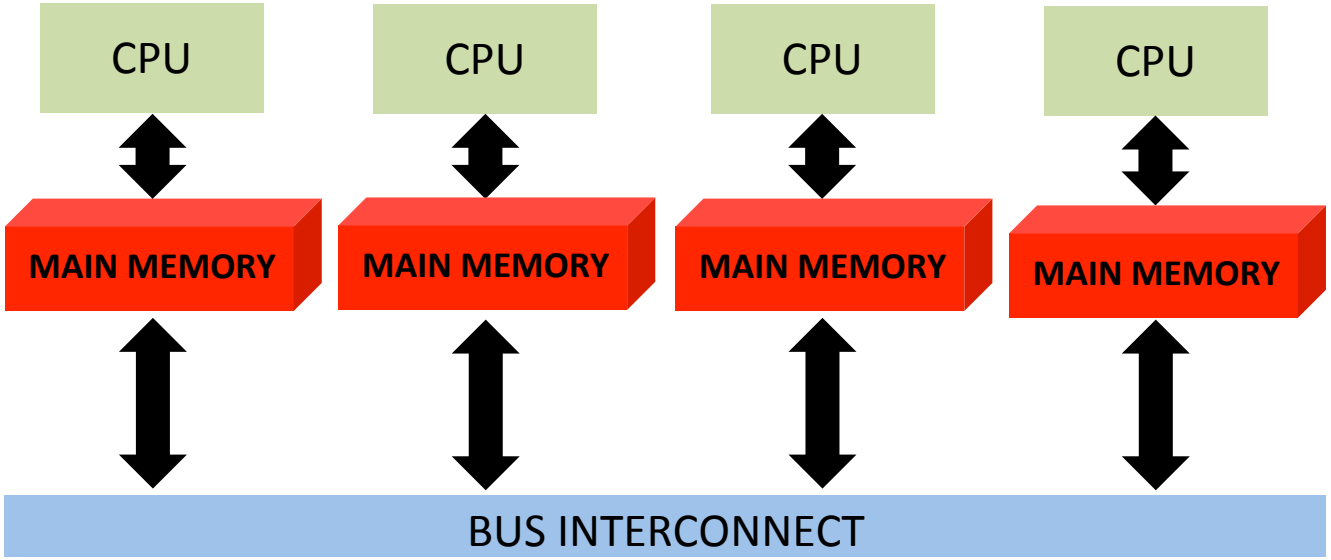
Expensive Operations

- Fast (0.5x-1x): add, subtract, multiply
- Medium (5-10x): divide, modulus, sqrt()
- Slow (20-50x): most transcendental functions
- Very slow (>100x): power (xy for real *x* and *y)*
- Often **only** the fastest operations are pipelined, so code will be the fastest when using only multiply/add
  - BLAS (= Basic Linear Algebra Subroutines)
  - LAPACK (Linear Algebra Package)

Symmetric Multiprocessors (SMP)



Modern NUMA Multicores



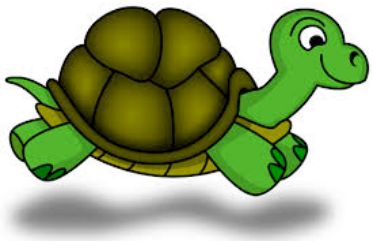
The "dimensions of performance"

- Vectors
- Instruction Pipelining
- Instruction Level Parallelism (ILP)
- Hardware threading
- Clock frequency
- Multi-core
- Multi-socket
- Multi-node

Micro-parallelism: gain in throughput and in time-to-solution

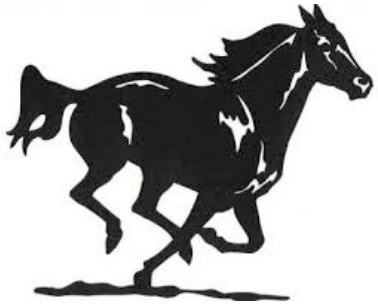
Gain in memory footprint and time-to-solution but not in throughput

Libraries



Highly-portable
(included into some
Linux distributions)

netlib



Freely available
Open Source Optimized

FFTW MAGMA ATLAS

OpenBLAS PLASMA

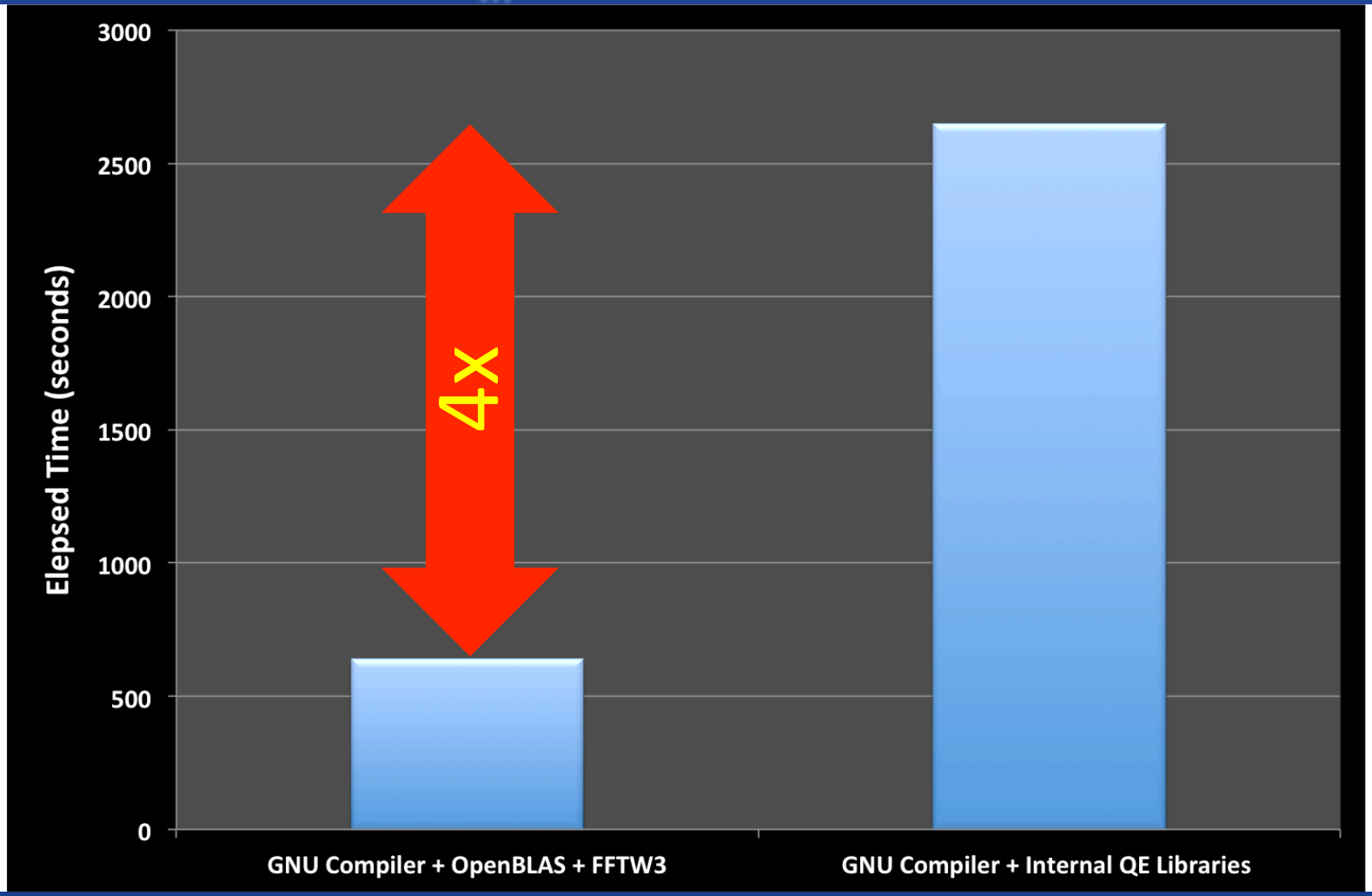
Third-Party Highly-Optimized

AMD Core Math Library ACML

Intel Math Kernel Library

nag Results Matter. Trust NAG.

IBM ESSL


3000
2500
2000
1500
1000
500
0
Elepsed Time (seconds)
4x
GNU Compiler + OpenBLAS + FFTW3
GNU Compiler + Internal QE Libraries

Conclusion

- Handle the complexity of current CPU technology is a non-trivial task
- Moore's law continues, but leads to multi-core, larger caches and higher integration => Performance increase now mostly through better algorithms, optimization, vectorization, and parallelization
- Bottleneck has transitioned from CPU speed to memory access and efficient data structures
- Make use of optimized software (i.e., libraries)
- Write cleaned and structured code to enhance the compiler and perform possible optimizations
- Best optimization requires often a writing/rewriting for reducing the complexity and/or help the compiler to work efficiently