MHPC
Master in High Performance Computing

Lecture 3

Introduction to modern CPUs and memory hierarchies

Stefano Cozzini

CNR/IOM and eXact-lab srl

SISSA
Scuola Internazionale Superiore di Studi Avanzati

ICTP The Abdus Salam International Centre for Theoretical Physics

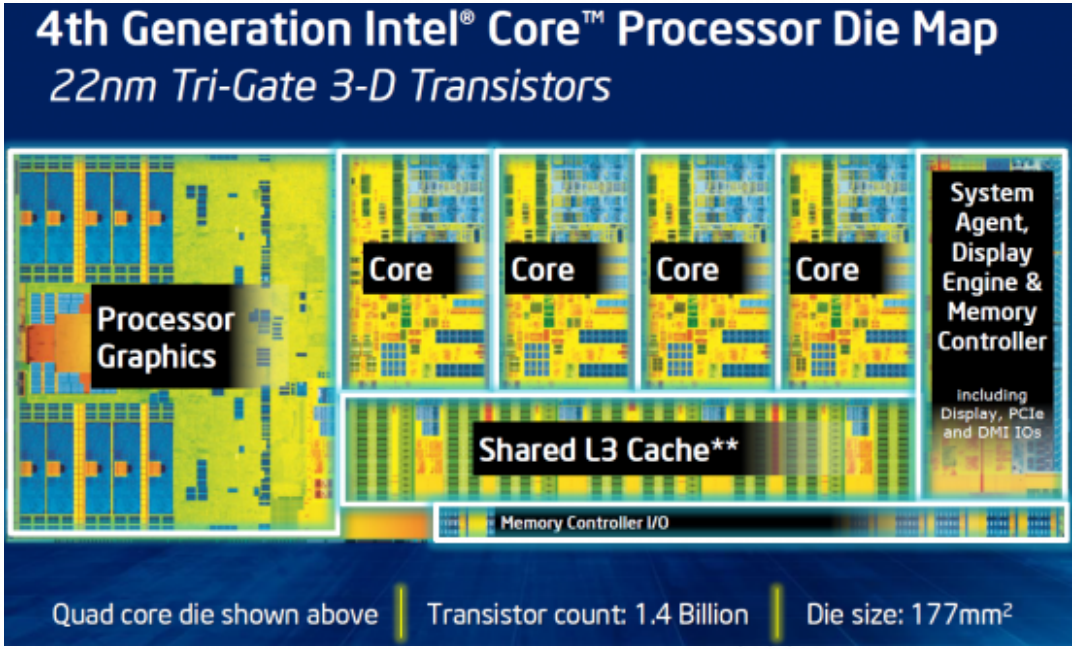
MHPC
Master in High Performance Computing

Outline

- ~~Moore law~~
- Components of a modern CPU
- Von Neumann architecture
- Modern features of CPU and cores
  - SIMD
  - Pipelines
  - superscalar processing and ILP
- Survey of modern CPUs for HPC
- Memory hierarchy and its components

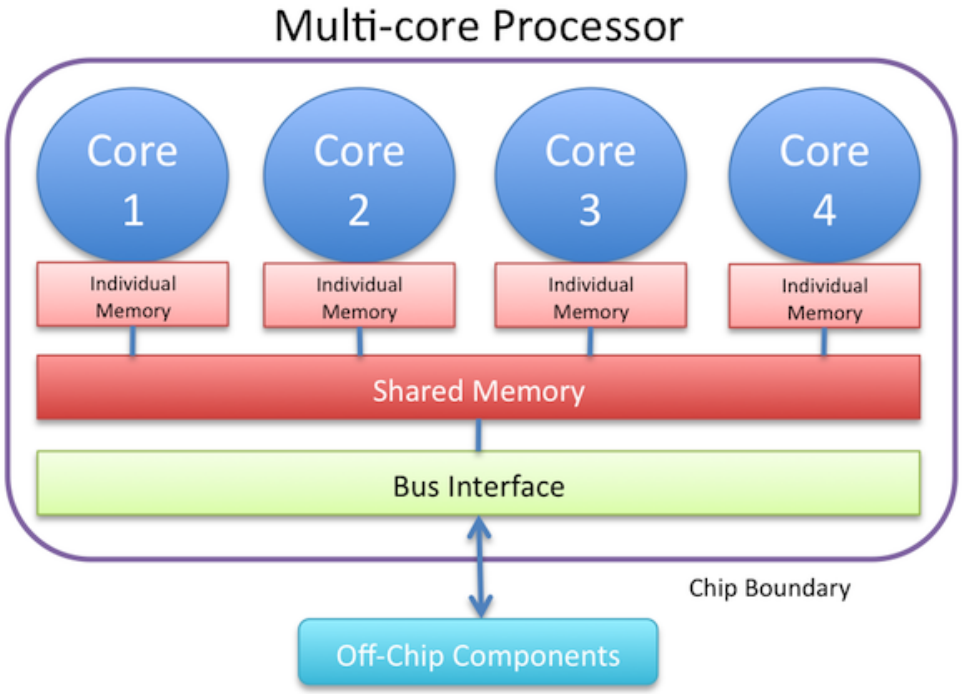
MHPC
Master in High Performance Computing

What does a CPU look like?



Modern CPU are multicore

- Because of power, heat dissipation, etc increasing tendency to actually lower clock frequency but pack more computing cores onto a chip.
- These cores will share some resources, e.g. memory, network, disk, etc but are still capable of independent calculations



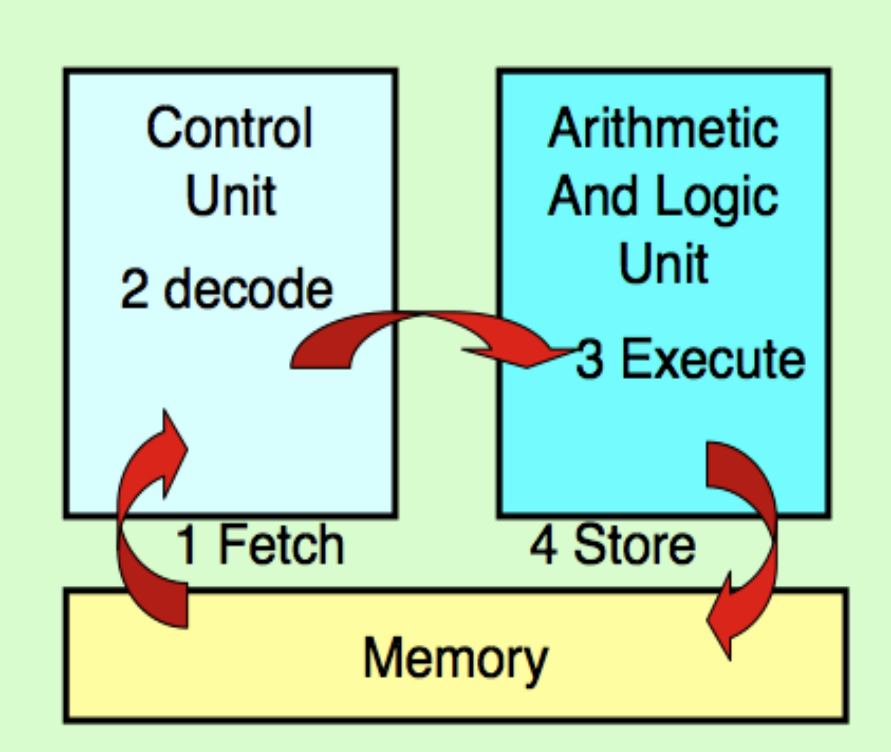
Picture from: http://www.cse.wustl.edu/~jain/cse567-11/ftp/multcore/

What is a core?

Level 0 approximation:
von Nuemann architecture

Von Nuemann architecture:

- Control Unit: processes instructions ALU: math and logic operations
- At each cycle the CPU fetches both data and a description of what operations need to be performed and stores them in registers.



Instruction Set Architecture (ISA)

- It is the boundary between SW and HW
- The deeper level accessible to the programmers
- The interface between the programmer and the microarchitecture
- Different microarchitectures can have the same ISA (binary Compatible)
- Different generation of microarchitectures can be backward compatible

MHPC
Master in High Performance Computing

Analysis of a simple program on Von Neumann architecture:

```
void store(double *a, double *b,
double *c) {
 *c = *a + *b;
}
```

```
[exact@master ~]$ gcc -O2 -S -o - frammento.c
        .file "frammento.c"
        .text
        .p2align 4,,15
.globl store
        .type   store, @function
store:
.LFB0:
        .cfi_startproc
        movsd   (%rdi), %xmm0   #load *a to mmx0
        addsd   (%rsi), %xmm0   # load b and add to *a
        movsd   %xmm0, (%rdx)  # store to C
        ret
        .cfi_endproc
.LFE0:
        .size   store, .-store
        .ident  "GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-4)"
        .section .note.GNU-stack,"",@progbits
```

SISSA ICTP

Workflow:

- Instruction fetch
- Instruction decode: determine operation and operands
- Memory fetch: Get operands from memory
- Perform operation
- Write results back
- Continue with next instruction

How to go faster ?

- since many years CPU designers have been trying to improve and optimize the "Classical Model":
  - increasing clock frequency (Moore's law)
  - reducing the number of cycles needed to perform a single instruction (a = c + d)
    - flow/branch prediction
  - improve the number of outputs x cycle
    - Instructions Level Parallelism (ILP)
    - Execution => in-order Vs. out-of-order

MHPC
Master in High Performance Computing

Contemporary cores are a little bit more complicated..

SISSA
ICTP

MHPC
Master in High Performance Computing

What is in a core really ?

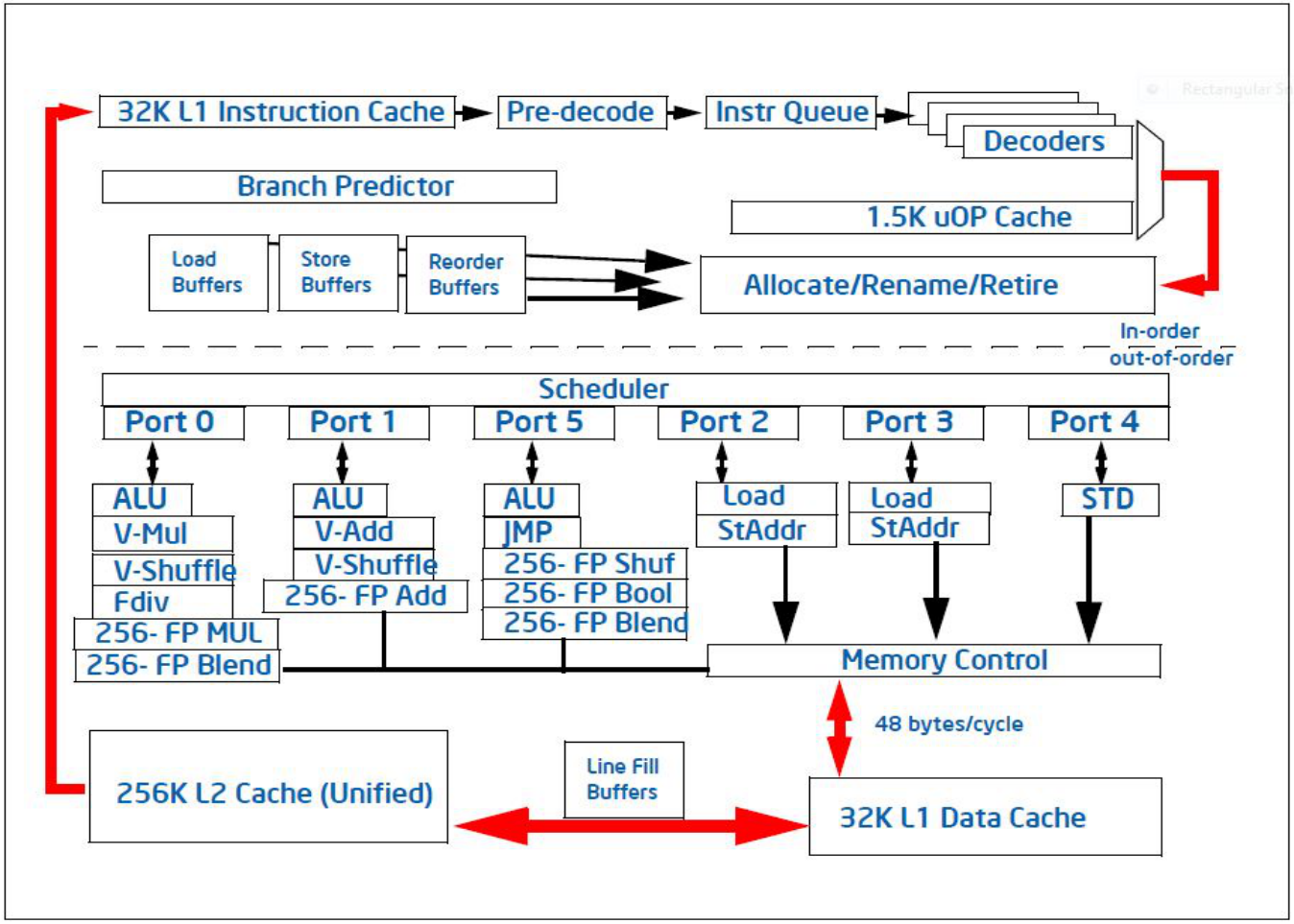


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

ICTP

Instruction handling: out of order execution.

- It is unrealistic on modern CPU to assume all the instruction are executed sequentially
- “out of order” instruction handling has been present for almost 20 years
- Processor/core is allowed to change the order of the instructions in your code.
  - Be careful on this on Floating Point operations
- Several units on modern core are dedicated to this

MHPC
Master in High Performance Computing

Instruction handling sections on Sandybridge core:

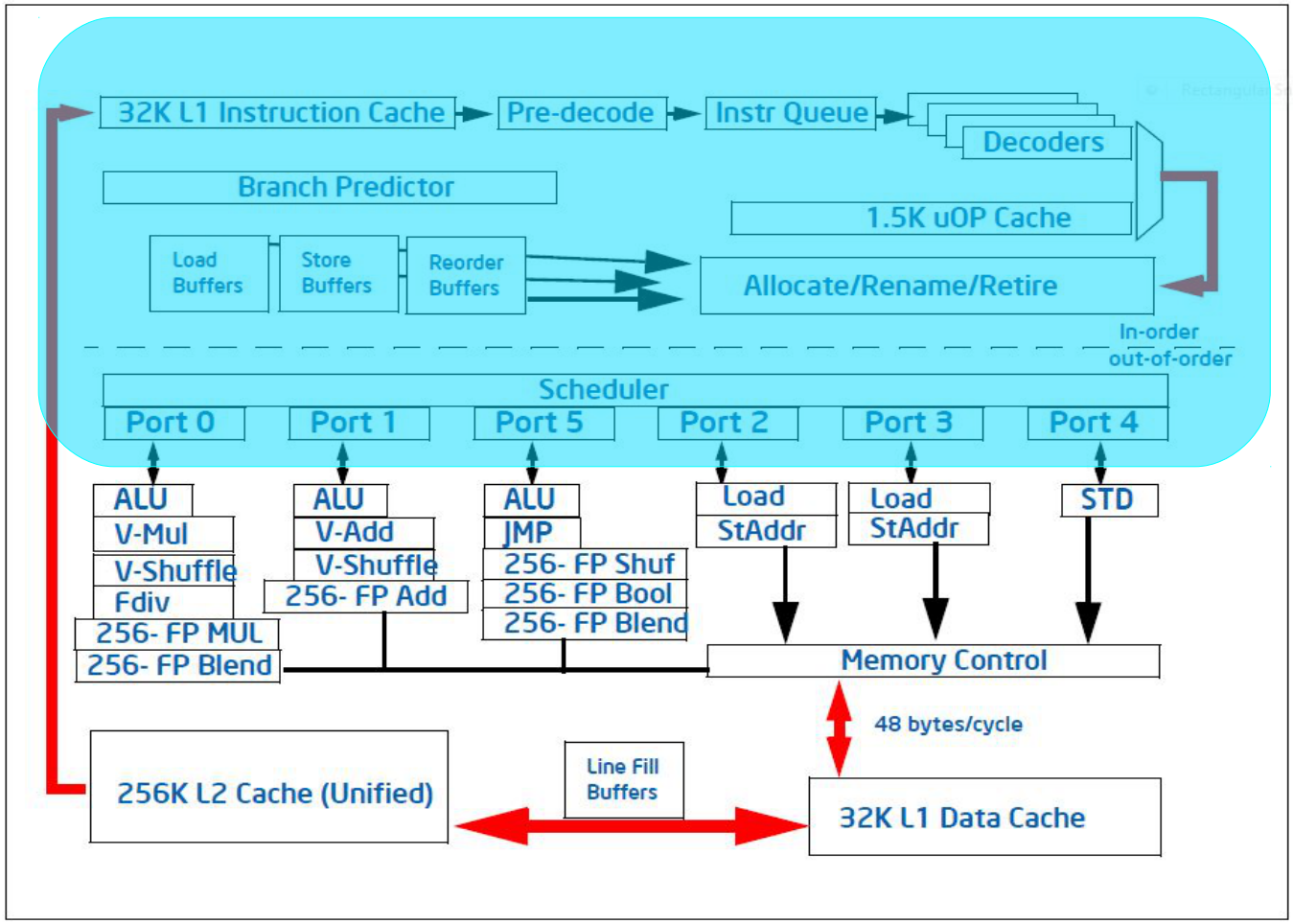


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

ICTP

Many functional unit

- Circuitry on the chip which performs a given type of operation on operands in registers is known as a functional unit.
- For HPC we are mainly interested in Floating Point Units
- Some FPUs can perform more than one operation per cycle..
  - Multiple floating point units
  - 1 Mul + 1 Add, or 1 Fused Multiply-Add (FMA)

$x \leftarrow c*x+y$

What else in the cores ?

- On modern CPU/Cores there are many other stuff:
  - Superscalar execution
  - Pipelined functional units
  - Floating point instruction set extensions

MHPC
Master in High Performance Computing

Superscalar CPU architecture

- The ability to issue/execute more than one instruction at the same time
- aka Instruction level parallelism (ILP)
- On sandybridge:
  - Six issue ports/ execution units that can execute Instruction simultaneously
  - e.g. Floating Point-Add (Port 1) and Floating-Point Mult (Port 0)

SISSA ICTP

Superscalarity on Sandybridge

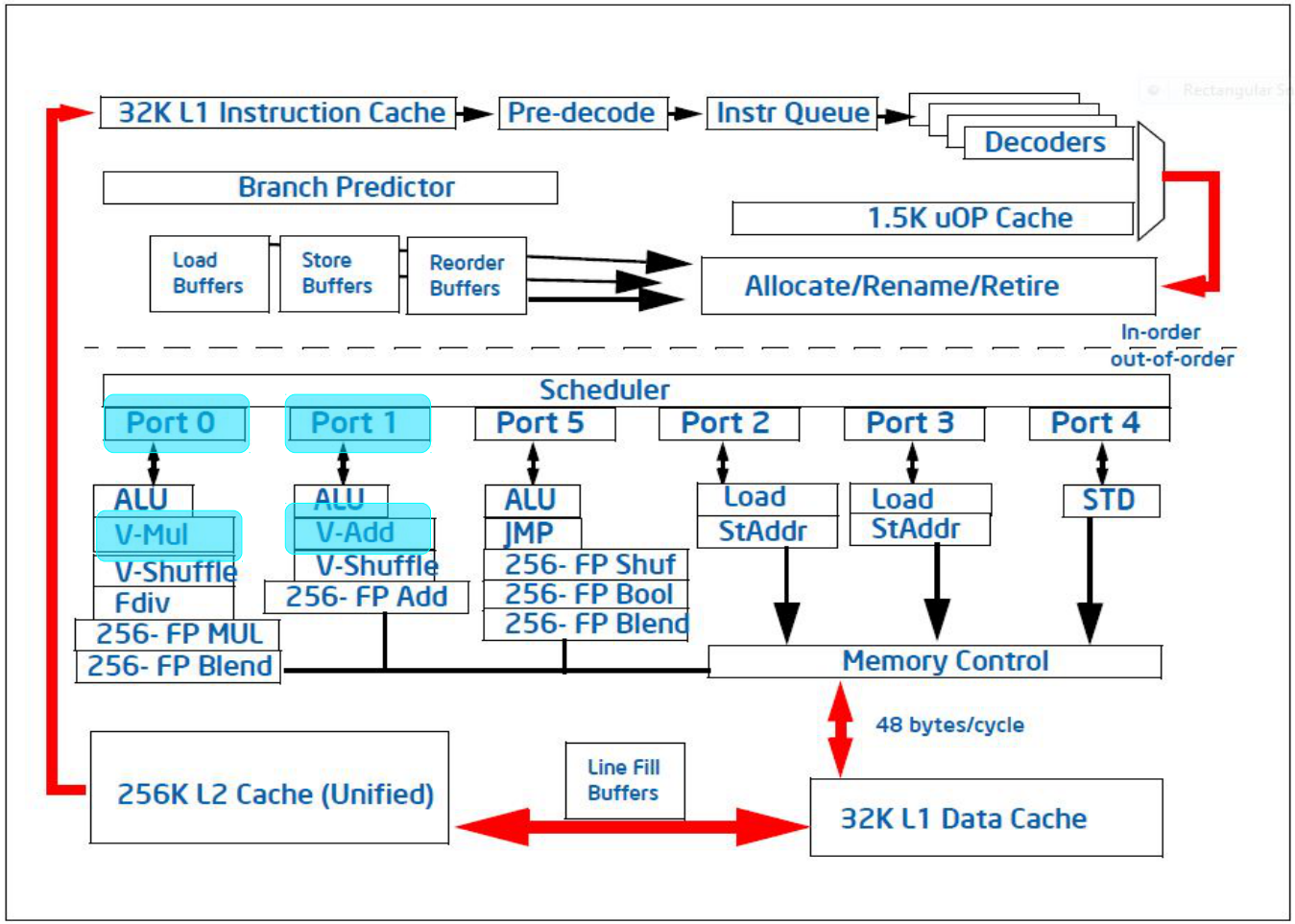


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

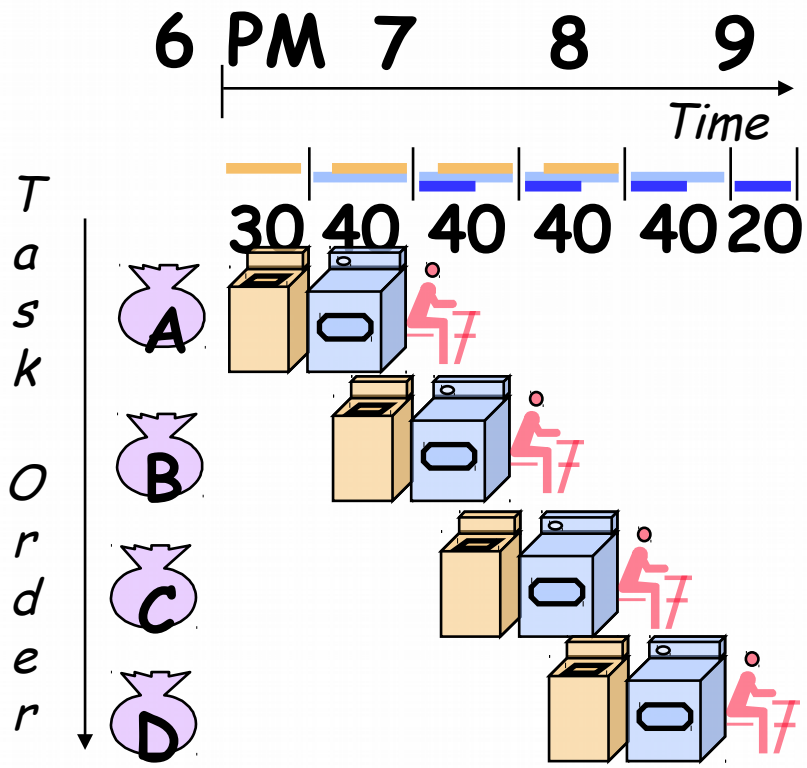
Pipelined Functional Units

- Most integer and floating point functional units are pipelined, meaning that they can have multiple independent executions of the same instruction placed in a queue.
- The idea is that after an initial startup latency, the functional unit should be able to generate one result every clock period (CP).
- Each stage of a pipelined operation can be working simultaneously on different sets of operands.
- For each stage of the pipeline there is a dedicated HW

MHPC
Master in High Performance Computing

What is Pipelining?

Dave Patterson's Laundry example: 4 people doing laundry
wash (30 min) + dry (40 min) + fold (20 min)



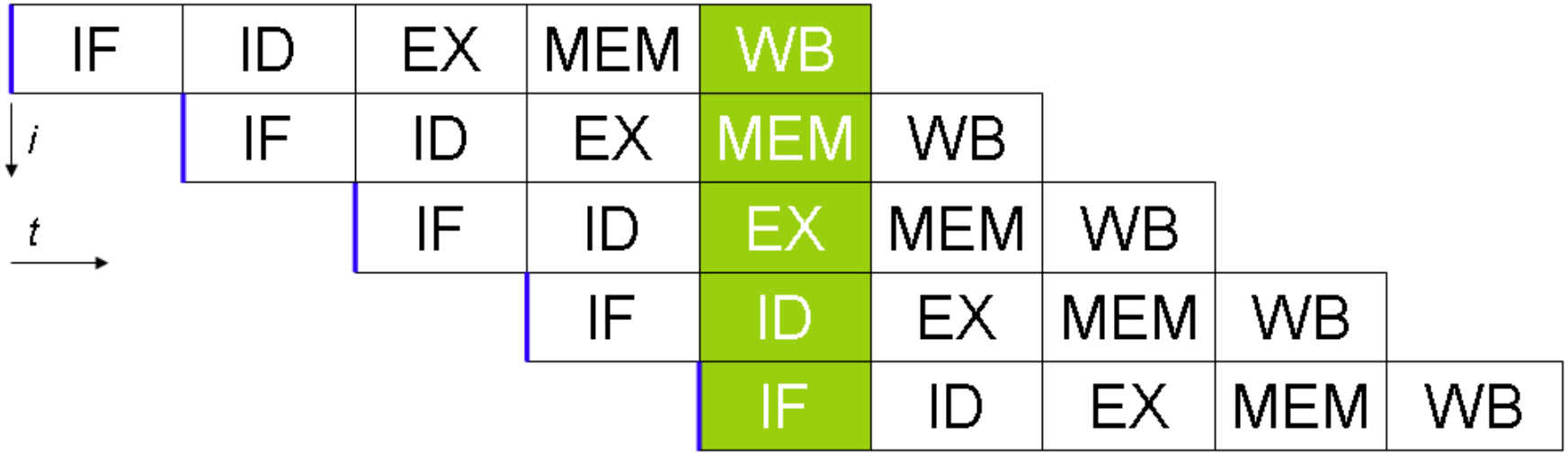
In this example:

Sequential execution takes 4 * 90min = 6 hours

Pipelined execution takes 30+4*40+20 = 3.3 hours

SISSA
ICTP

The standard RISC 5 stage pipeline



IF = Instruction Fetch
ID = Instruction Decode
EX = Execute,
MEM = Memory access
WB = Register write back

Pipeline concepts/jargon

- Wind-up phase: time it takes to load the pipeline (a.k.a. latency)
- Wind-down phase: time it takes to drain the pipeline
- Segments=stages=steps of the pipeline
- Speedup: *Tseq/Tpipe*
- Throughput = *N/Tpipe*
- Pipelining helps throughput, but not latency
- Pipeline rate limited by slowest pipeline stage

Pipeline analysis: $N_{1/2}$

Given a pipeline with *s* segments and *N* operations:

- T_{seq} (the time without pipelining) = sN
- $T_{pipe} = s+N-1$

  → Speed-up= $s*N/s+N-1$

  Throughput= $N/T_{pipe} = 1/(1 + (s-1)/N)$
- for N large enough
  - *Speedup ~ s*
  - *Throughput ~ 1*

    i.e. 1 result per clock cycle
- With *N* operations, actual rate is $N/(s+N)$
- This is half of the asymptotic rate if $s=N$

Asymptotic behavior of pipeline throughput

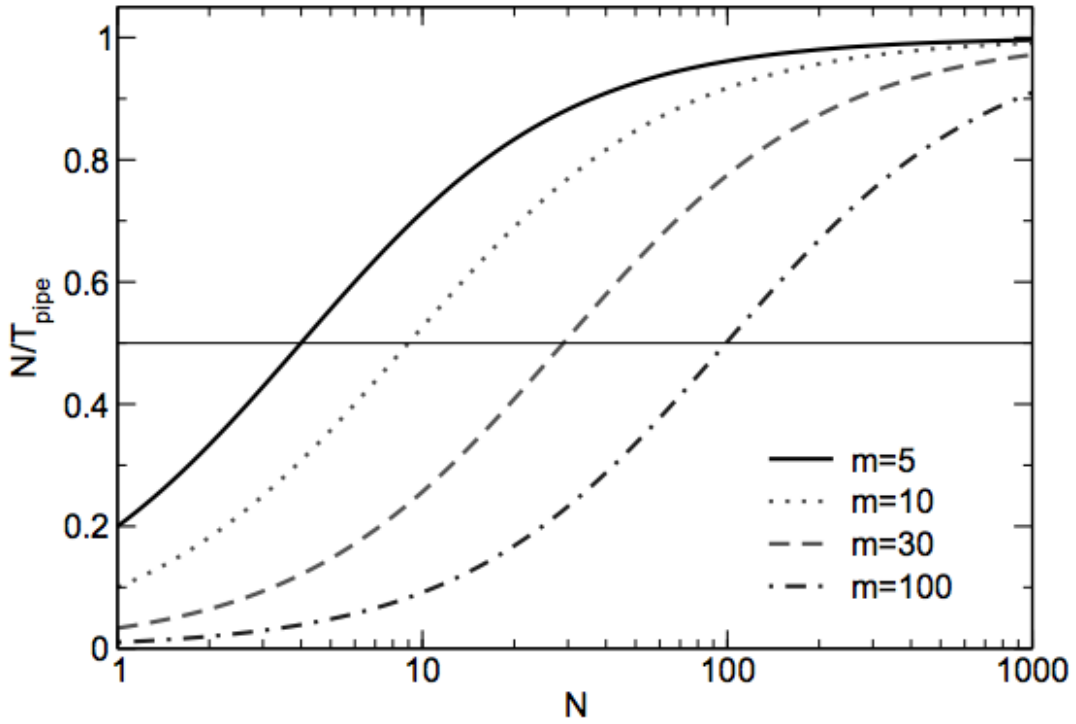


Figure 1.6: Pipeline throughput as a function of the number of independent operations. *m* is the pipeline depth.

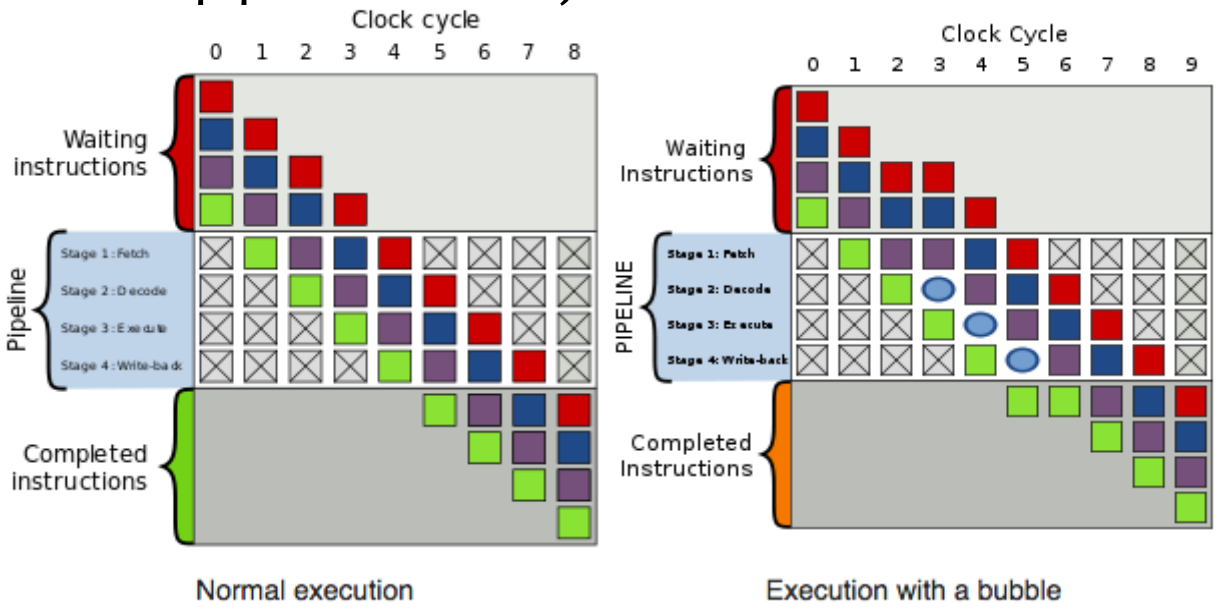
MHPC
Master in High Performance Computing

Pipeline drawbacks

Standard lengths of modern processor: from 10 to 35

- Not efficient at all if loops are short and tight

- Not efficient if there are jumps (branches) or stalling in instructions (the so called pipeline bubble)



ICTP

Branch Prediction

- The “instruction pipeline” is all of the processing steps (also called segments) that an instruction must pass through to be “executed”.
- Higher frequency machines have a larger number of segments.
- Branches are points in the instruction stream where the execution may jump to another location, instead of executing the next instruction.
- For repeated branch points (within loops), instead of waiting for the loop to branch route outcome, it is predicted.

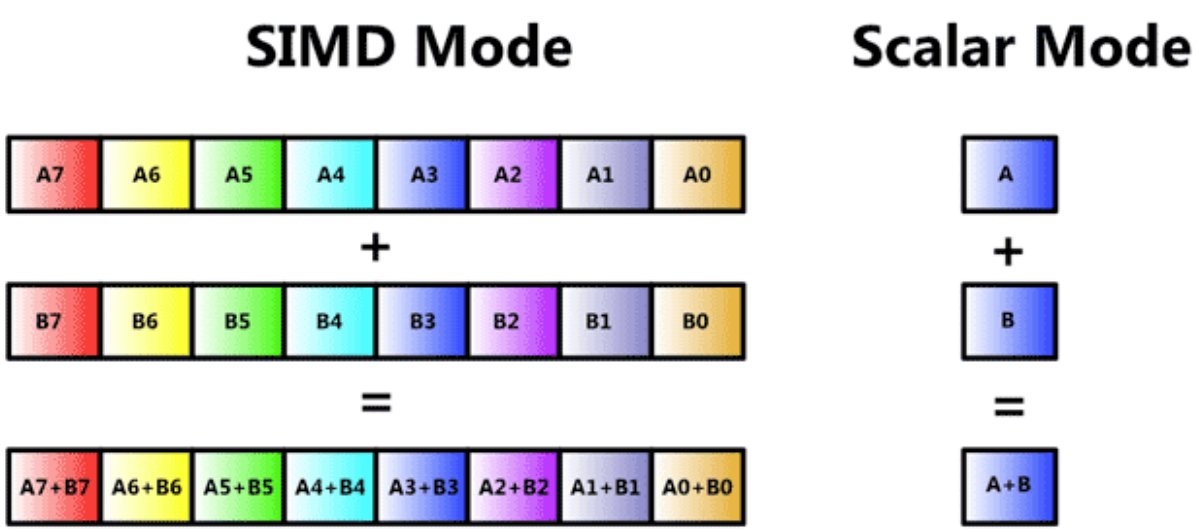
Floating Point Instruction Set Extensions

- additional floating point instructions beyond the usual floating point add and multiply instructions:
  - Square root instruction --usually not pipelined!
    - AMD Opteron / Intel Xeon
  - SIMD (a.k.a. vector) floating point instructions
    - AMD Opteron/ Intel Xeon
- Combined floating point multiply/add (MADD) instruction
  - AMD Opteron ("Barcelona" and after, using SIMD)
  - Intel Xeon ("Woodcrest" and after, using SIMD)

Instruction Set Extensions

- Intel
  - MMX (Matrix Math eXtensions)
  - SSE (Streaming SIMD Extensions)
  - SSE2 / SSE4.2
  - AVX AVX-512 (Advanced vector eXtension)
    - From sandybridge processor on
- AMD
  - 3DNow!
  - AMD 3DNow!+ (or 3DNow! Professional, or 3DNow! Athlon) ...
- To check what you have on your machine:
  - cat /proc/cpuinfo
  - to enable them: use appropriate compiler flag..

SIMD approach:



The AVX Instruction Set

Latest version of Intel and GNU compiler are in most cases able to perform auto-vectorization on simple loop constructs and with simplified data-structures

Machine-dependent vector intrinsic that use a function call syntax to emit machine code (to be used at large when compilers not work as expected)

Memory alignment is required

It is available as compiler option on both Intel and GNU compiler:

-mavx (GNU) | -xAVX (INTEL)

Intel SIMD evolution

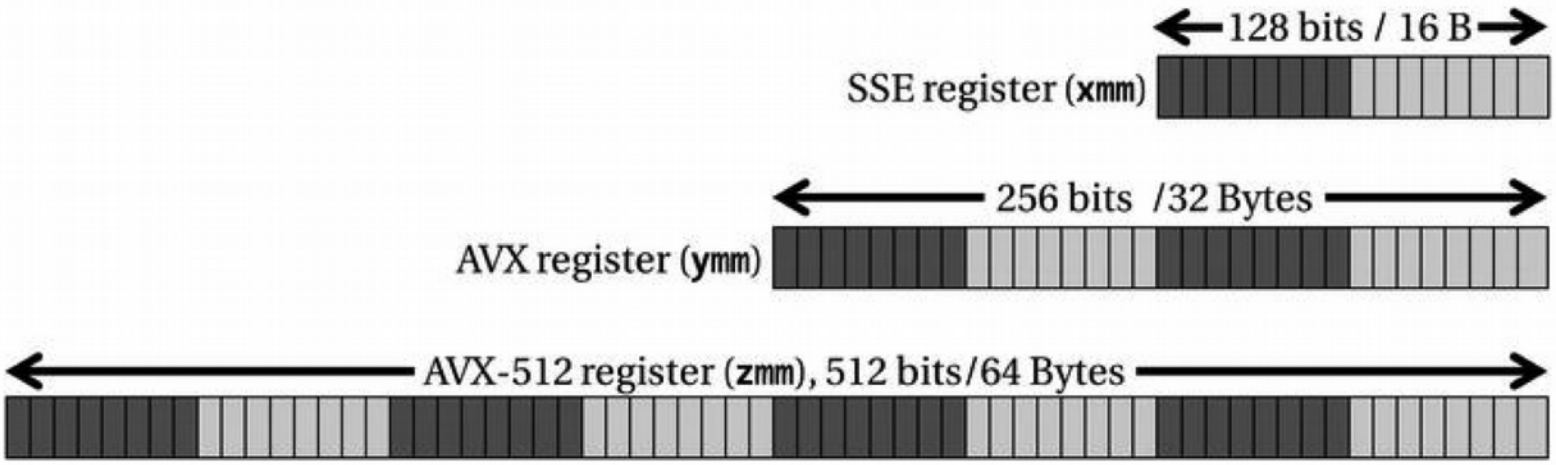


Figure 2-9. SSE, AVX, and AVX-512 vector registers with packed 64-bit numbers

MHPC
Master in High Performance Computing

Number crunching on CPU: what do we count ?

- Rate of [million/billions of] floating point operations per second ([M|G]flops) FLOPs/S
- Theoretical peak performance:

determined by counting the number of floating-point additions and multiplications that can be completed during a period of time, usually the cycle time of the machine

FLOPS=Clock-rate*Number_of_FP_ operation*Number_of_cores

SISSA ICTP

MHPC
Master in High Performance Computing

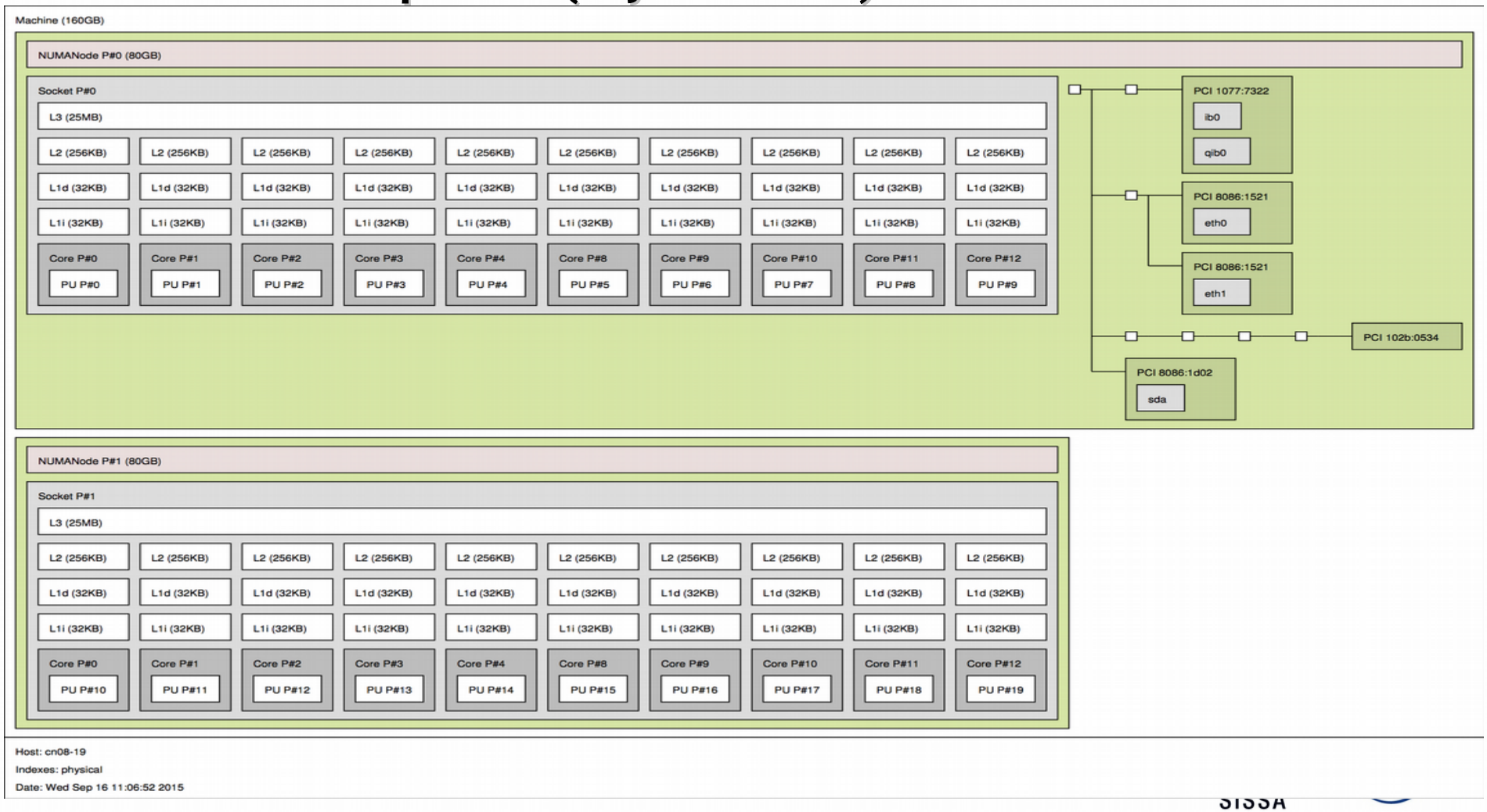
Exercise: compute Theoretical Peak performance for your laptop/desktop:

- Identify the CPU
- Identify the frequency
- Identify the number of floating point for cycle
- Identify how many cores
- Put all together in one single number

SISSA
ICTP

MHPC
Master in High Performance Computing

A modern node picture (Ulysses-node)



MHPC
Master in High Performance Computing

Peak performance on ulysses node:

- Two Sockets
- 10 Cores per Socket @ 2.6 GHz
- Two-way superscalar with respect to floating-point
  - AVX Mult on Port 0, AVX Add on Port 1
- 256b AVX
    - Four double-precision operations with a single instruction

2 x 10 x 2.6 GHz x 2 instr/cycle x 4 ops/instr = ????? Gflop/s

SISSA ICTP

MHPC
Master in High Performance Computing

What CPU for HPC ?

Processor Generation	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
Intel Xeon E5 (Broadwell)	246	49.2	306,000,700	482,350,558	13,002,624
Intel Xeon E5 (Haswell)	119	23.8	168,402,842	273,284,090	8,920,138
Xeon Gold	35	7	78,141,598	127,714,633	1,668,020
Intel Xeon E5 (IvyBridge)	20	4	96,164,654	150,344,962	6,530,040
Intel Xeon Phi	19	3.8	110,722,839	222,455,609	4,956,500
Xeon Platinum	15	3	55,781,560	89,381,831	1,229,744
Power BQC	11	2.2	37,387,541	43,830,478	3,424,256
Intel Xeon E5 (SandyBridge)	10	2	13,825,898	17,420,061	972,396
Xeon 5600-series (Westmere-EP)	5	1	8,295,600	16,463,946	514,948
SPARC64 XIfx	5	1	10,422,200	11,530,310	354,384
Intel Xeon E7 (Broadwell)	3	0.6	2,833,215	4,463,616	128,320
IBM POWER9	3	0.6	194,928,000	308,338,395	3,874,464
Opteron 6200 Series "Interlagos"	2	0.4	18,757,000	28,617,830	711,168
Xeon Silver	1	0.2	1,471,000	3,000,000	33,000
POWER7	1	0.2	1,587,000	1,931,625	62,944
SPARC64 VIIIfx	1	0.2	10,510,000	11,280,384	705,024
Sunway	1	0.2	93,014,594	125,435,904	10,649,600
ShenWei	1	0.2	795,900	1,070,160	137,200
Intel Xeon E7 (Haswell-Ex)	1	0.2	822,723	1,495,859	42,496
Xeon 5500-series (Nehalem-EX)	1	0.2	1,050,000	1,254,550	138,368

Some considerations

- Intel is dominating
- AMD is disappearing (or already disappeared)
- IBM is coming back with Power8/9 architecture
- ARM processor still to come

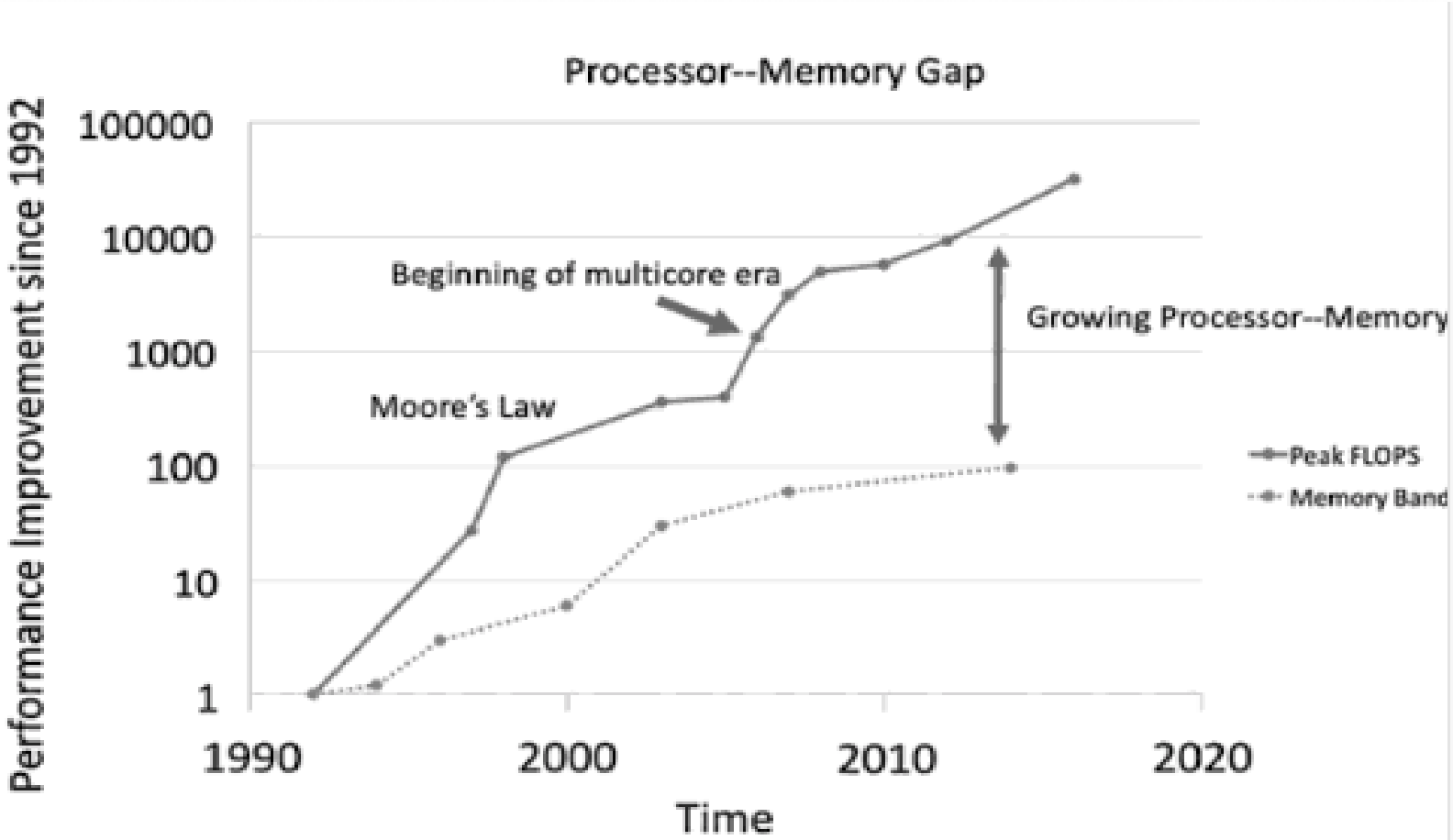
MHPC
Master in High Performance Computing

Final notes on CPU/cores..

- Modern CPU have a high degree of parallelism some time hidden to the user
- In order to optimize on them you should be aware of this (see next lectures)

SISSA
ICTP

The memory wall



MHPC
Master in High Performance Computing

The memory wall: some notes

- Memory latency is a barrier to performance improvements
- Current architectures have ever growing caches to improve the "average memory reference" time to fetch or write instructions or data
- Memory Wall: due to latency and limited communication bandwidth beyond chip boundaries.
- From 1992 to nowadays, CPU speed improved by 4 orders of magnitude while memory access speed only improved by two orders !

SISSA
ICTP

Memory Hierarchy

In modern computer system same data is stored in several storage devices during processing

The storage devices can be described & ranked by their speed and "distance" from the CPU

There is thus a *hierarchy of memory objects*

Programming a machine with memory hierarchy requires optimization for that memory structure.

MHPC
Master in High Performance Computing

Memory Hierarchies

- Memory is divided into different levels:
  - Registers
  - Caches
  - Main Memory
- Memory is accessed through the hierarchy
  - registers where possible
  - ... then the caches
  - ... then main memory

SISSA
ICTP

Memory hierarchy

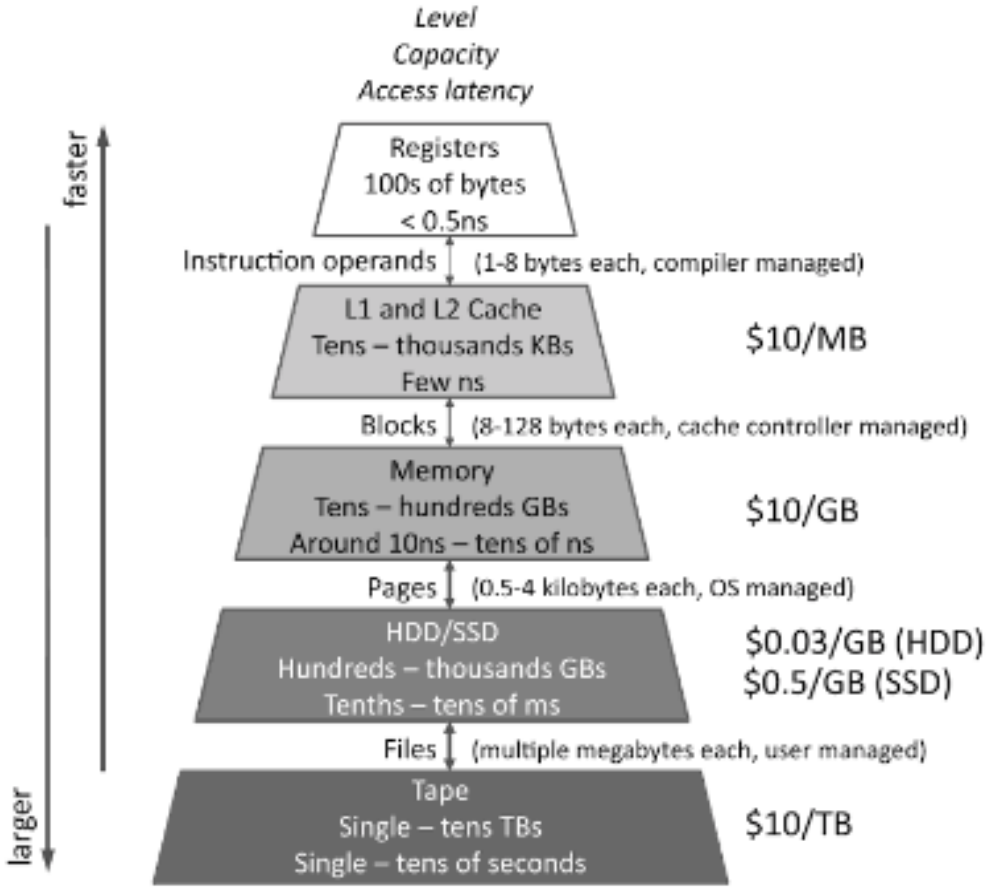


FIGURE 6.7

Latency and Bandwidth

- The two most important terms related to performance for memory subsystems and for networks are:
  - Latency
    - How long does it take to retrieve a word of memory?
    - Units are generally nanoseconds (milliseconds for network latency) or clock periods (CP).
    - Sometimes addresses are predictable: compiler will schedule the fetch. Predictable code is good!
  - Bandwith
    - What data rate can be sustained once the message is started?
    - Units are B/sec (MB/sec, GB/sec, etc.)

Registers

- Highest bandwidth, lowest latency memory that a modern processor can access
  - built into the CPU
  - often a scarce resource
  - not RAM
- Processors instructions operate on registers directly
  - have assembly language names names like:
    - eax, ebx, ecx, etc.
  - sample instruction:

```
addl   %eax, %edx
```

- Separate instructions and registers for floating-point operations

Note: some of SIMD tools have they own register !

Data Caches

- Between the CPU Registers and main memory
- L1 Cache: Data cache closest to registers
- L2 Cache: Secondary data cache, stores both data and instructions
  - Data from L2 has to go through L1 to registers
  - L2 is 10 to 100 times larger than L1
- L3 cache, ~10x larger than L2

Cache line

- The smallest unit of data transferred between main memory and the caches (or between levels of cache; every cache has its own line size)
- *N* sequentially-stored, multi-byte words (usually N=8 or 16).
- If you request one word on a cache line, you get the whole line
  - make sure to use the other items, you've paid for them in bandwidth
  - Sequential access good, "strided" access ok, random access bad

Main Memory

- Cheapest form of RAM
- Also the slowest
  - lowest bandwidth
  - highest latency
- Unfortunately most of our data lives out here

MHPC
Master in High Performance Computing

Multi-core chips

- Cores may have separate L1/L2, shared L2/L3 cache
  - Depends on CPU model
  - Hybrid shared/distributed model
- Cache coherency problem: conflicting access to duplicated cache lines.

(more on this when we discuss multicore)

SISSA ICTP

Cache and register access

- Access is transparent to the programmer
  - data is in a register or in cache or in memory
  - Loaded from the highest level where it's found
  - processor/cache controller/MMU hides cache access from the programmer
- …but you can influence it:
  - Access *x* (that puts it in L1), access 100k of data, access *x* again: it will probably be gone from cache
  - If you use an element twice, don't wait too long
  - If you loop over data, try to take chunks of less than cache size
  - C declare register variable, only suggestion

Register use

- y[i] can be kept in register
- Declaration is only suggestion to the compiler
- Compiler can usually figure this out itself

```
for (i=0; i<m; i++) {
  for (j=0; j<n; j++) {
    y[i] = y[i]+a[i][j]*x[j];
  }
}
```

```
register double s;
for (i=0; i<m; i++) {
  s = 0.;
  for (j=0; j<n; j++) {
    s = s+a[i][j]*x[j];
  }
  y[i] = s;
}
```

Hits, Misses, Thrashing

- Cache hit
  - location referenced is found in the cache
- Cache miss
  - location referenced is not found in cache
  - triggers access to the next higher cache or memory
- Cache thrashing
  - Two data elements can be mapped to the same cache line: loading the second "evicts" the first
  - Now what if this code is in a loop? "thrashing": really bad for performance

MHPC
Master in High Performance Computing

Cache Mapping

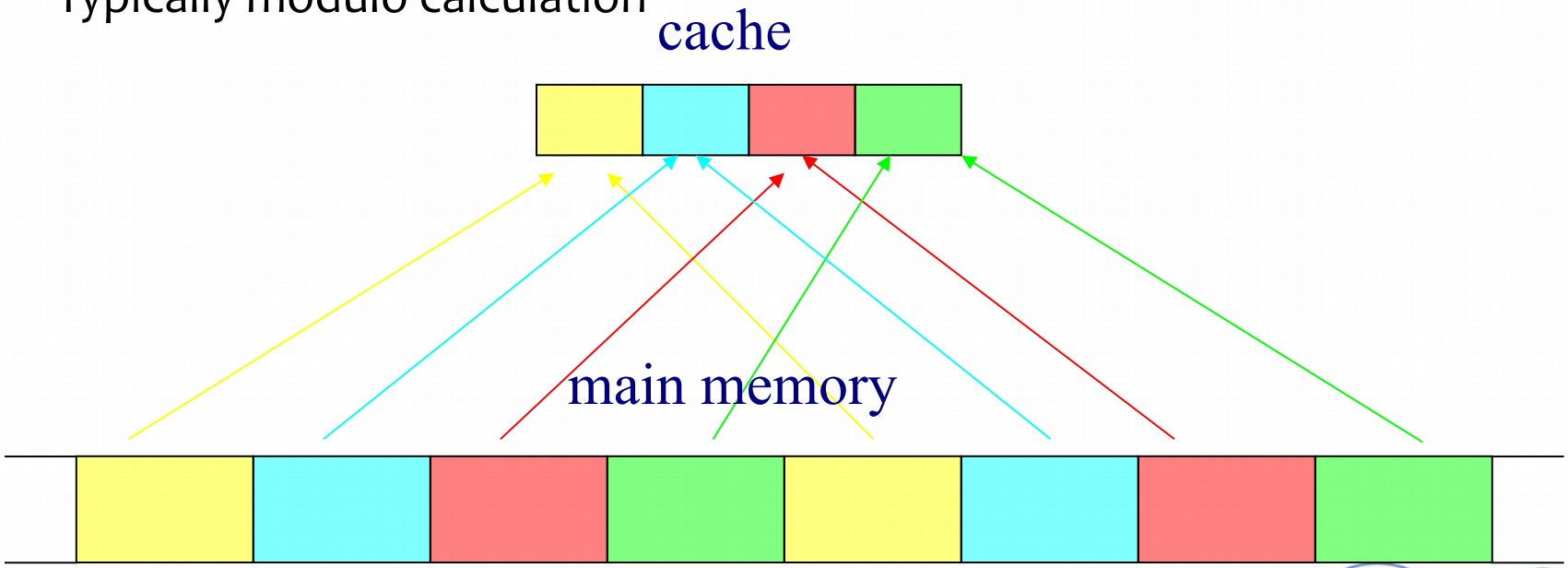
- Because each memory level is smaller than the next-closer level, data must be mapped
- Types of mapping
  - Direct
  - Set associative
  - Fully associative

Direct Mapped Caches

- If the cache size is N_c and it is divided into k lines, then each cache line Is N_c/k in size
- If the main memory size is N_m, memory is then divided into $N_m/(N_c/k)$ blocks that are mapped into each of the k cache lines
- Means that each cache line is associated with particular regions of memory

Direct Mapped Caches

Direct mapped cache: A block from main memory can go in exactly one place in the cache. This is called direct mapped because there is direct mapping from any block address in memory to a single location in the cache. Typically modulo calculation



The problem with Direct Mapping

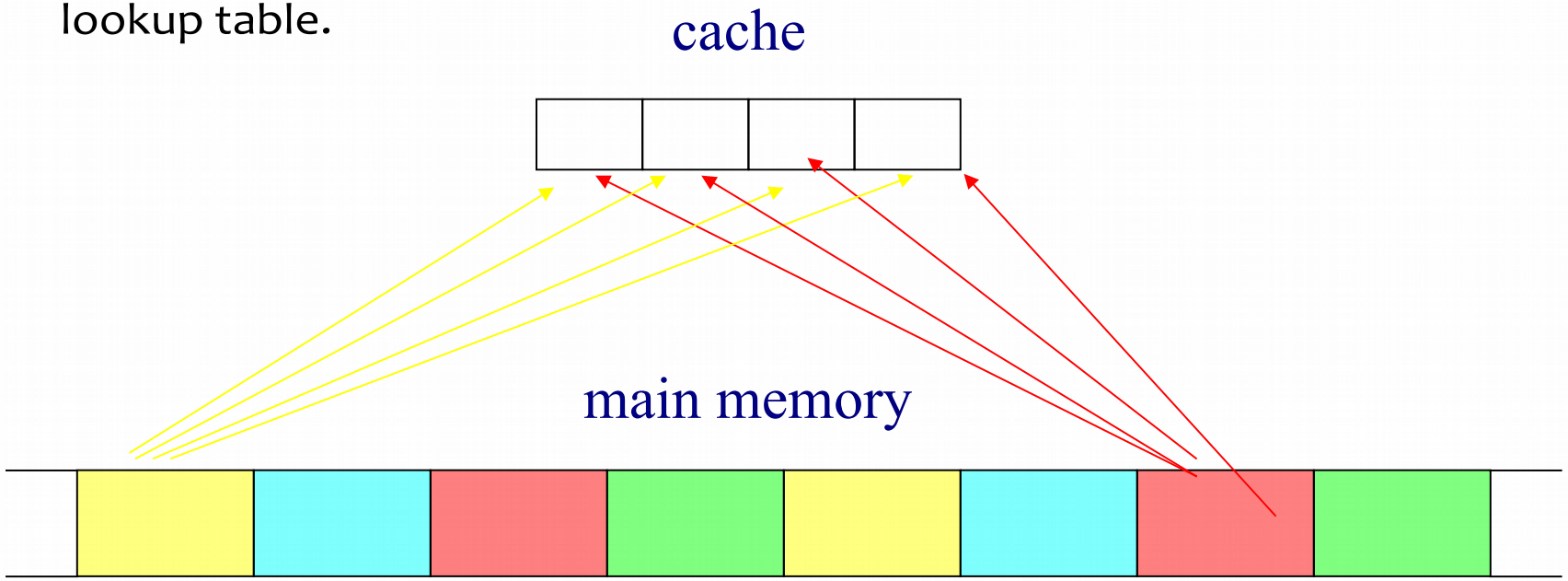
```
double a[8192],b[8192];
for (i=0; i<n; i++) {
  a[i] = b[i]
}
```

Example: cache size 64k=2^{16} byte = 8192 words

- a[0] and b[0] are mapped to the same cache location
- Cache line is 4 words
- Thrashing:
  - b[0]..b[3] loaded to cache, to register
  - a[0]..a[3] loaded, gets new value, *kicks b[0]..b[3] out of cache*
  - b[1] requested, so b[0]..b[3] loaded again
  - a[1] requested, loaded, *kicks b[0..3] out again*

Fully Associative Caches

Fully associative cache : A block from main memory can be placed in any location in the cache. This is called fully associative because a block in main memory may be associated with any entry in the cache. Requires lookup table.

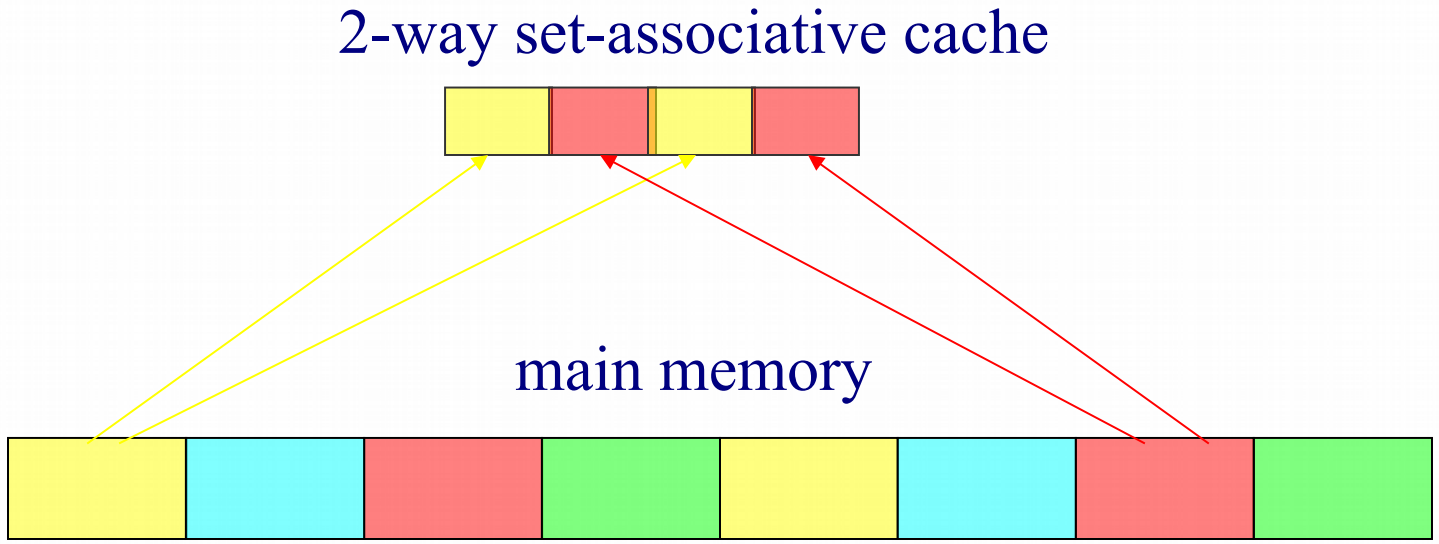


Fully Associative Caches

- Ideal situation
- Any memory location can be associated with any cache line
- Cost prohibitive

Set Associative Caches

Set associative cache : The middle range of designs between direct mapped cache and fully associative cache is called set-associative cache. In a n-way set-associative cache a block from main memory can go into n (n at least 2) locations in the cache.



Set Associative Caches

- Direct-mapped caches are 1-way set-associative caches
- For a k-way set-associative cache, each memory region can be associated with *k* cache lines
- Fully associative is k-way with k the number of cache lines

Last element: TLB

- Translation Look-aside Buffer
- Translates between logical space that each program has and actual memory addresses
- Memory organized in ‘small pages’, a few Kbyte in size
- Memory requests go through the TLB, normally very fast
- Pages that are not tracked through the TLB can be found through the ‘page table’: much slower
- => jumping between more pages than the TLB can track has a performance penalty.