Hardware:CPU

By Mohit Kumar

CPU:Concepts

- Background introduces CPU-related terminology, basic models of CPUs, and key CPU performance concepts.
- Architecture introduces processor and kernel scheduler architecture.
- Methodology describes performance analysis methodologies, both observational and experimental.
- Analysis describes CPU performance analysis tools on Linux- and Solaris-based systems, including profiling, tracing, and visualizations.
- Tuning includes examples of tunable parameters.

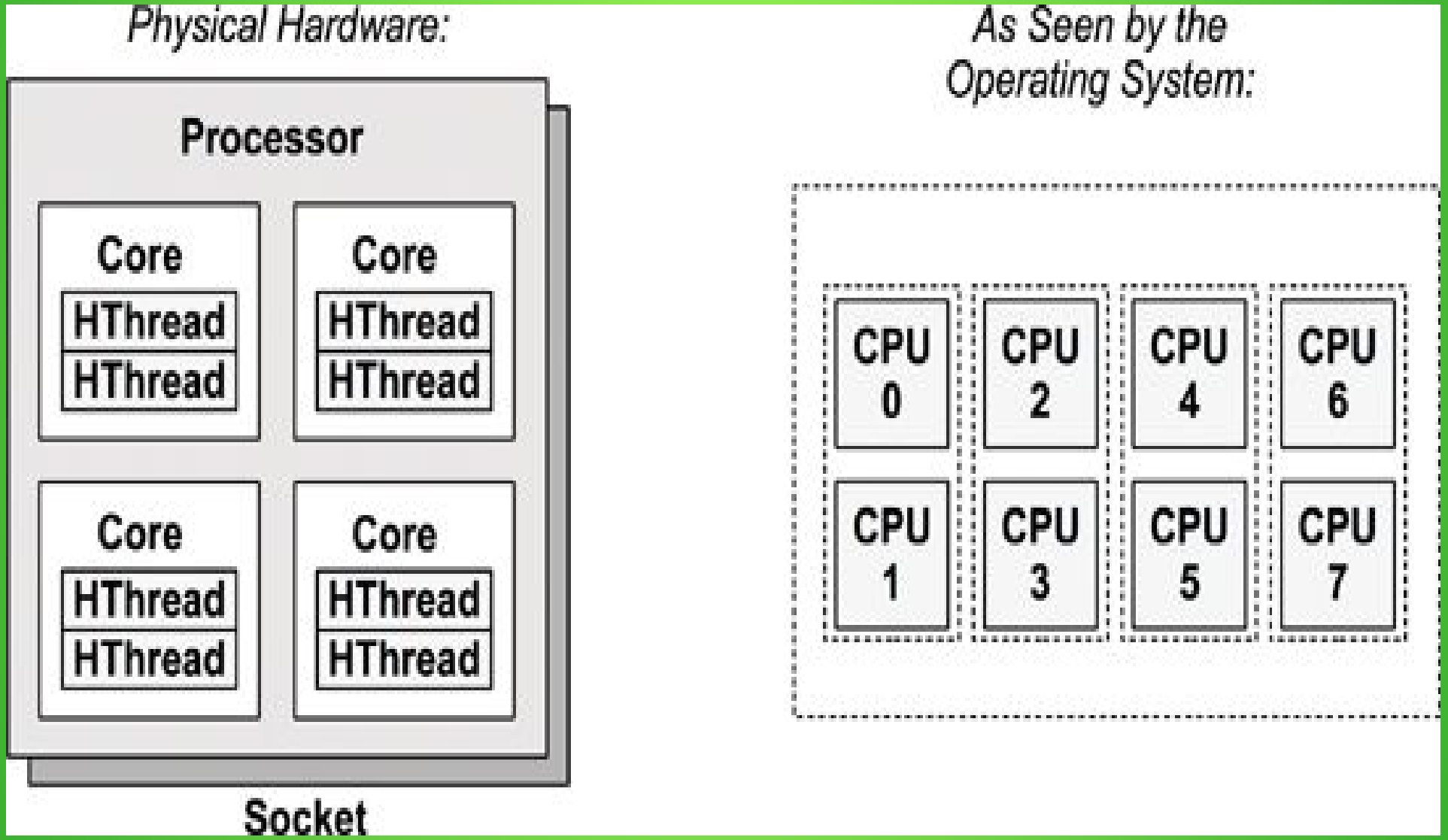
CPU:Terminology

- **Processor**: the physical chip that plugs into a socket on the system or processor board and contains one or more CPUs implemented as cores or hardware threads.
- **Core**: an independent CPU instance on a multicore processor. The use of cores is a way to scale processors, called chip-level multiprocessing (CMP).
- **Hardware thread**: a CPU architecture that supports executing multiple threads in parallel on a single core (including Intel's Hyper-Threading Technology), where each thread is an independent CPU instance. One name for this scaling approach is multithreading.
- **CPU instruction**: a single CPU operation, from its instruction set. There are instructions for arithmetic operations, memory I/O, and control logic.

CPU:Terminology

- **Logical CPU**: also called a virtual processor, an operating system CPU instance (a schedulable CPU entity). This may be implemented by the processor as a hardware thread (in which case it may also be called a virtual core), a core, or a single-core processor.
  - It is also sometimes called a virtual CPU; however, that term is more commonly used to refer to virtual CPU instances provided by a virtualization technology.
- **Scheduler**: the kernel subsystem that assigns threads to run on CPUs.
- **Run queue**

CPU:Architecture



CPU:Architecture:Clock rate

- Clock Rate
  - The clock is a digital signal that drives all processor logic. Each CPU instruction may take one or more cycles of the clock (called CPU cycles) to execute. CPUs execute at a particular clock rate; for example, a 5 GHz CPU performs 5 billion clock cycles per second.

CPU:Architecture:Clock rate:scaling:cpufreq-info

```
mohit@emptyminnd:~/Work/JINT$ cpufreq-info
cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: acpi-cpufreq
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 10.0 us.
  hardware limits: 780 MHz - 2.00 GHz
  available frequency steps: 2.00 GHz, 2.00 GHz, 1.90 GHz, 1.80 GHz, 1.70 GHz, 1.60 GHz, 1.50 GHz, 1.40 GHz, 1.30 GHz, 1.20 GHz, 1.10 GHz, 1000 MHz, 900
 MHz, 800 MHz, 780 MHz
  available cpufreq governors: conservative, ondemand, userspace, powersave, performance
  current policy: frequency should be within 780 MHz and 1.40 GHz.
                  The governor "ondemand" may decide which speed to use
                  within this range.
  current CPU frequency is 780 MHz.
  cpufreq stats: 2.00 GHz:0.00%, 2.00 GHz:0.00%, 1.90 GHz:0.00%, 1.80 GHz:0.00%, 1.70 GHz:0.00%, 1.60 GHz:0.00%, 1.50 GHz:0.00%, 1.40 GHz:10.72%, 1.30 G
Hz:0.76%, 1.20 GHz:0.73%, 1.10 GHz:0.79%, 1000 MHz:0.87%, 900 MHz:0.99%, 800 MHz:0.21%, 780 MHz:84.94%  (232844)
```

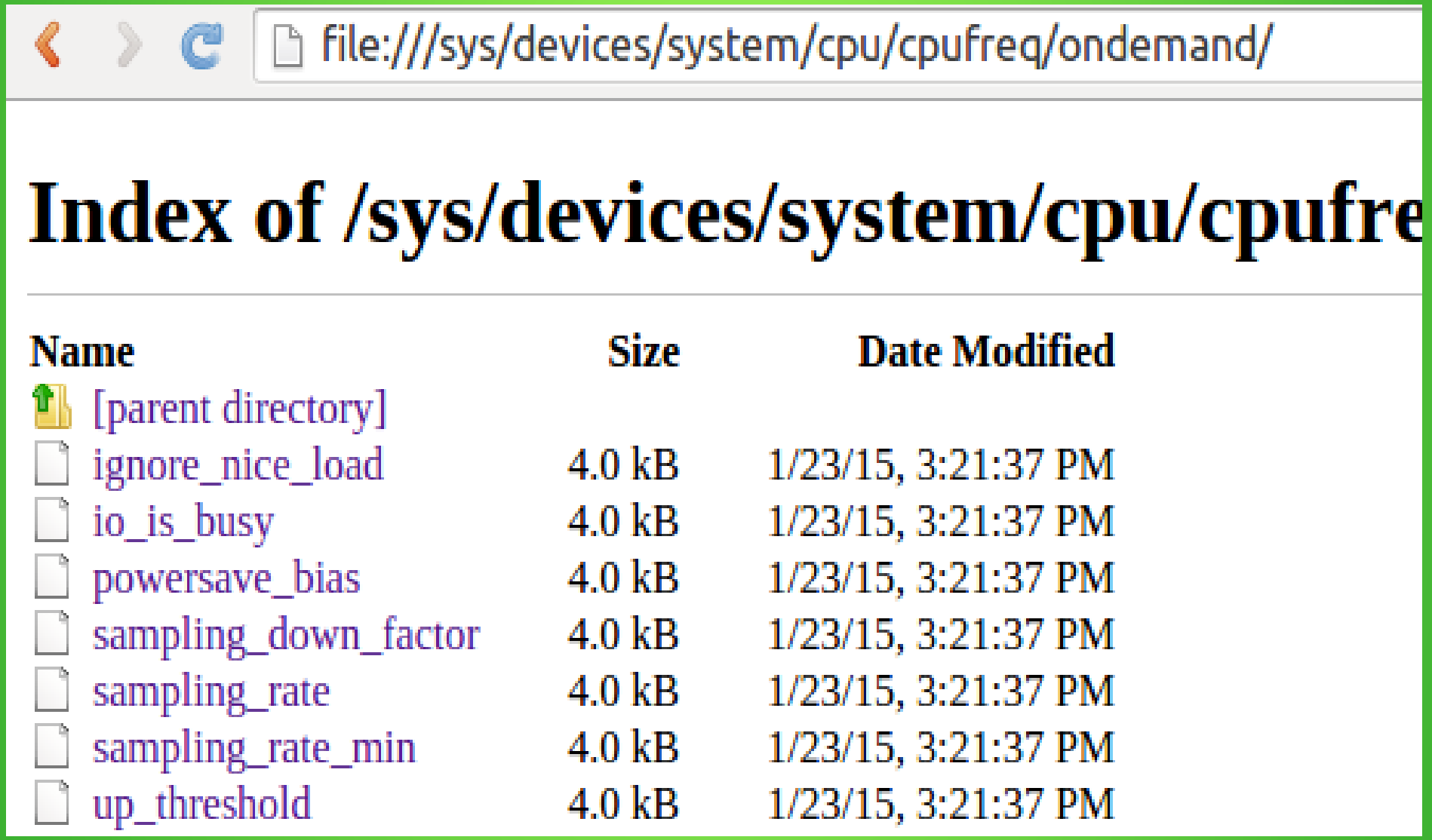
CPU:Architecture:Clock rate:scaling:cpufreq-info

```
mohit@emptyminnd:~/Work/JINT$ sudo cat /sys/devices/system/cpu/*/cpufreq/cpuinfo_cur_freq
2001000
2001000
2001000
1400000
mohit@emptyminnd:~/Work/JINT$ sudo cat /sys/devices/system/cpu/*/cpufreq/cpuinfo_cur_freq
2001000
2001000
1400000
2001000
mohit@emptyminnd:~/Work/JINT$ sudo cat /sys/devices/system/cpu/*/cpufreq/cpuinfo_cur_freq
2001000
2001000
2001000
1400000
mohit@emptyminnd:~/Work/JINT$ sudo cat /sys/devices/system/cpu/*/cpufreq/cpuinfo_cur_freq
2001000
2001000
2001000
2001000
```

CPU:Architecture:Clock rate:scaling:cpufreq-set -c 3 -f 2.50 GHz

```
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
root@nomind:~# cpufreq-set -c 7 -f 2.50GHz
root@nomind:~# cpufreq-info | grep current
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 2.50 GHz (asserted by call to hardware).
root@nomind:~# cpufreq-set -c 6 -f 2.50GHz
root@nomind:~# cpufreq-info | grep current
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 800 MHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 2.50 GHz (asserted by call to hardware).
  current policy: frequency should be within 800 MHz and 2.50 GHz.
  current CPU frequency is 2.50 GHz (asserted by call to hardware).
```

CPU:Architecture:Clock rate:scaling:sysfs(but prefer frequtils instead)



CPU:Architecture:Instruction

- CPUs execute instructions chosen from their instruction set. An instruction includes the following steps, each processed by a component of the CPU called a functional unit:
  - Instruction fetch
  - Instruction decode
  - Execute
  - Memory access
  - Register write-back

CPU:Architecture:InstructionPipelining

- Instruction Pipeline
  - The instruction pipeline is a CPU architecture that can execute multiple instructions in parallel, by executing different components of different instructions at the same time. It is similar to a factory assembly line, where stages of production can be executed in parallel, increasing throughput.

CPU:Architecture:InstructionWidth(DLP)

- Instruction Width
  - But we can go faster still. Multiple functional units can be included of the same type, so that even more instructions can make forward progress with each clock cycle.
  - This CPU architecture is called superscalar and is typically used with pipelining to achieve a high instruction throughput.

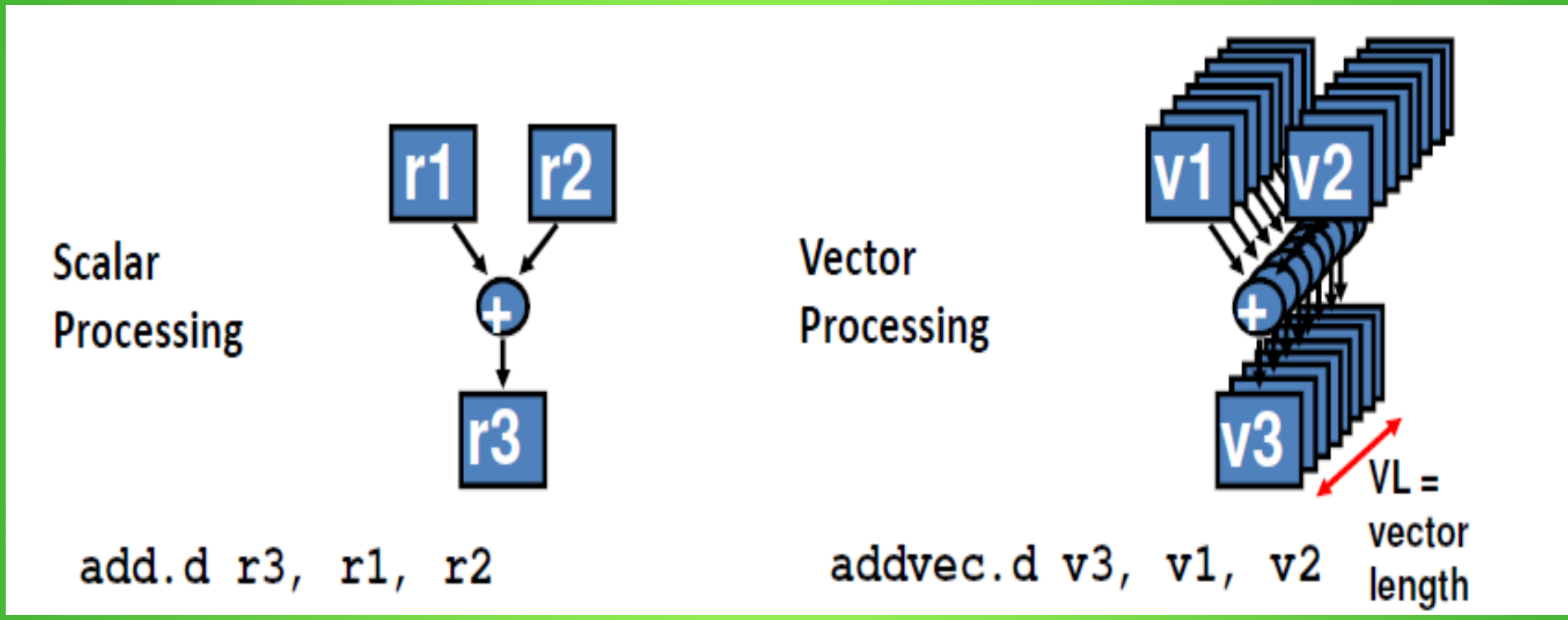
CPU:Architecture:InstructionWidth(DLP)

- Instruction Width
  - The instruction width describes the target number of instructions to process in parallel. Modern processors are 3-wide or 4-wide, meaning they can complete up to three or four instructions per cycle.

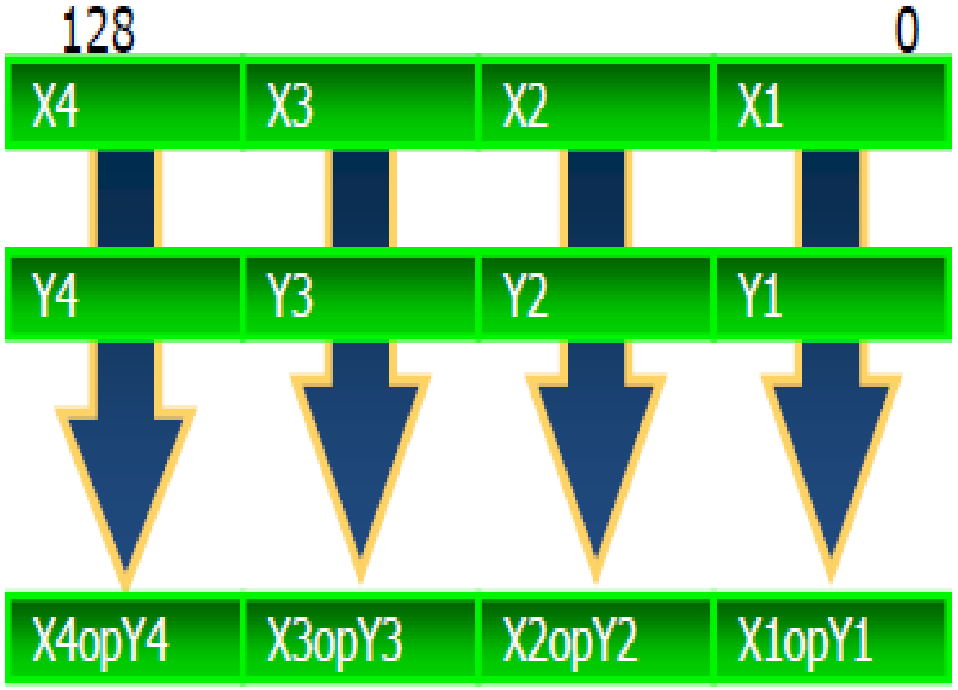
CPU:Architecture:InstructionWidth(DLP)

- Instruction Width
  - The instruction width describes the target number of instructions to process in parallel. Modern processors are 3-wide or 4-wide, meaning they can complete up to three or four instructions per cycle.

CPU:Architecture:InstructionWidth(DLP) :Vector Processing(SSE4,AVX2)



CPU:Architecture:InstructionWidth(DLP) :Vector Processing(SSE4,AVX2)



Intel® SSE

Vector size: 128bit

Data types:

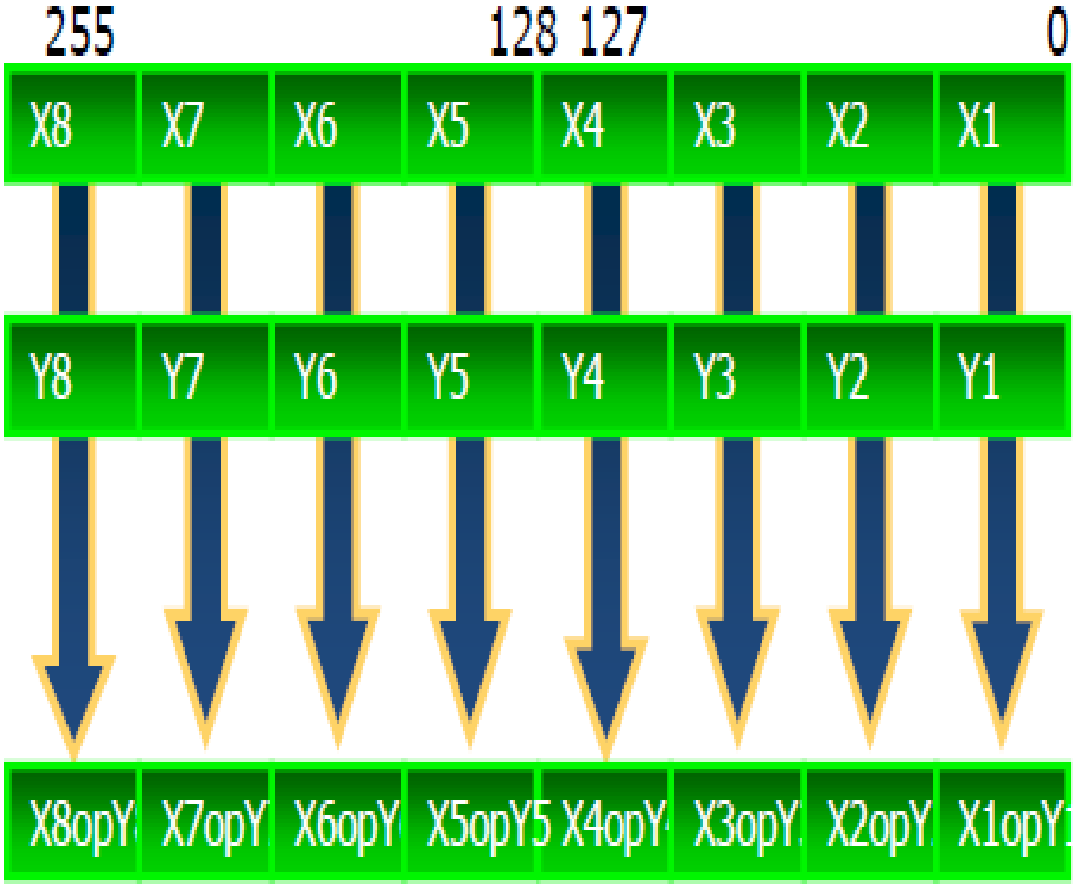
8,16,32,64 bit integers

32 and 64bit floats

VL: 2,4,8,16

Sample: Xi, Yi bit 32 int / float

CPU:Architecture:InstructionWidth(DLP) :Vector Processing(SSE4,AVX2(grafics))



Intel® AVX

Vector size: 256bit

Data types: 32 and 64 bit floats

VL: 4, 8, 16

Sample: Xi, Yi 32 bit int or float

CPU:Architecture:InstructionWidth(DLP):Vector Processing(SSE4,AVX2(grafics))

```
nohit@emptyminnd:~/Work/JINT$ cat /proc/cpuinfo | grep 'sse4\|avx2'
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acp
e1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu
2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c r
 dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acp
e1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu
2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c r
 dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acp
e1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu
2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c r
 dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acp
e1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu
2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c r
 dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid
```

CPU:Architecture:ILP

- Parallelism at the machine-instruction level
  - The processor can re-order, pipeline instructions, split them into microinstructions, do aggressive branch prediction, etc.
  - Instruction-level parallelism enabled rapid increases in processor speeds over the last 15 years

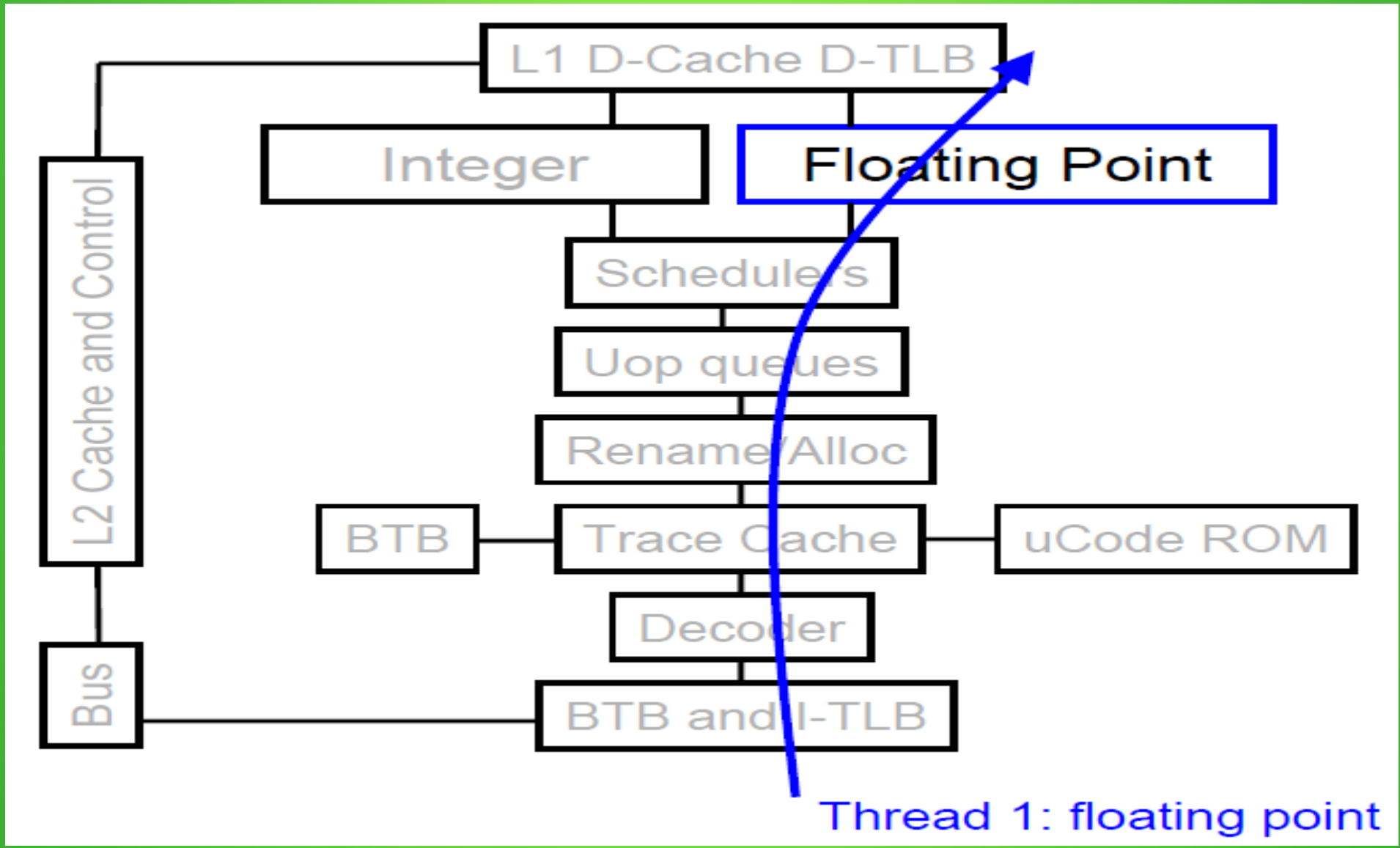
CPU:Architecture:ILP(SMT)

- Problem addressed:The processor pipeline can get stalled:
  - Waiting for the result of a long floating point (or integer) operation
  - Waiting for data to arrive from memory
- Other execution units wait unused

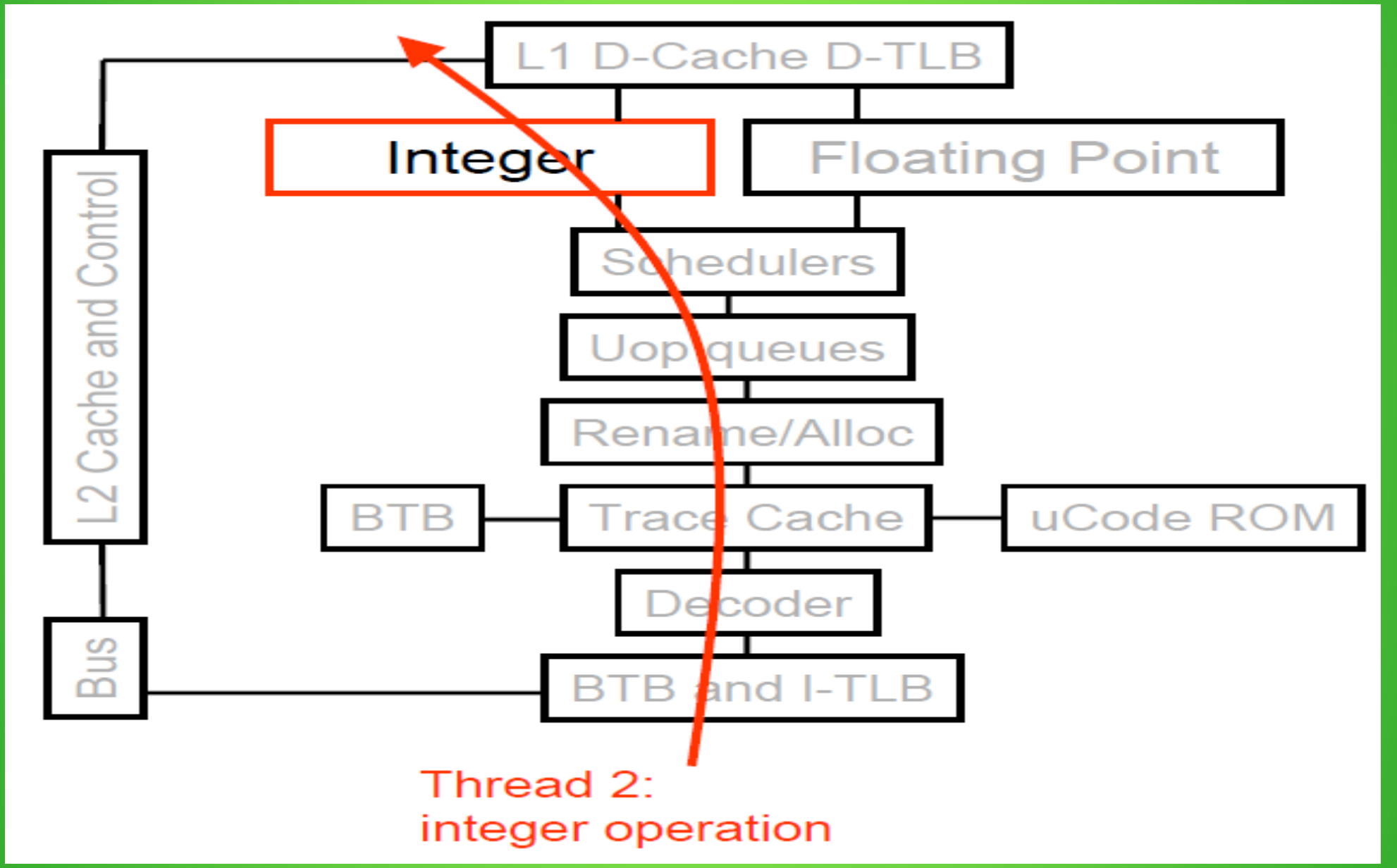
CPU:Architecture:ILP(SMT)

- Permits multiple independent threads to execute SIMULTANEOUSLY on the SAME core
  - Weaving together multiple “threads” on the same core
  - Example: if one thread is waiting for a floating point operation to complete, another thread can use the integer units

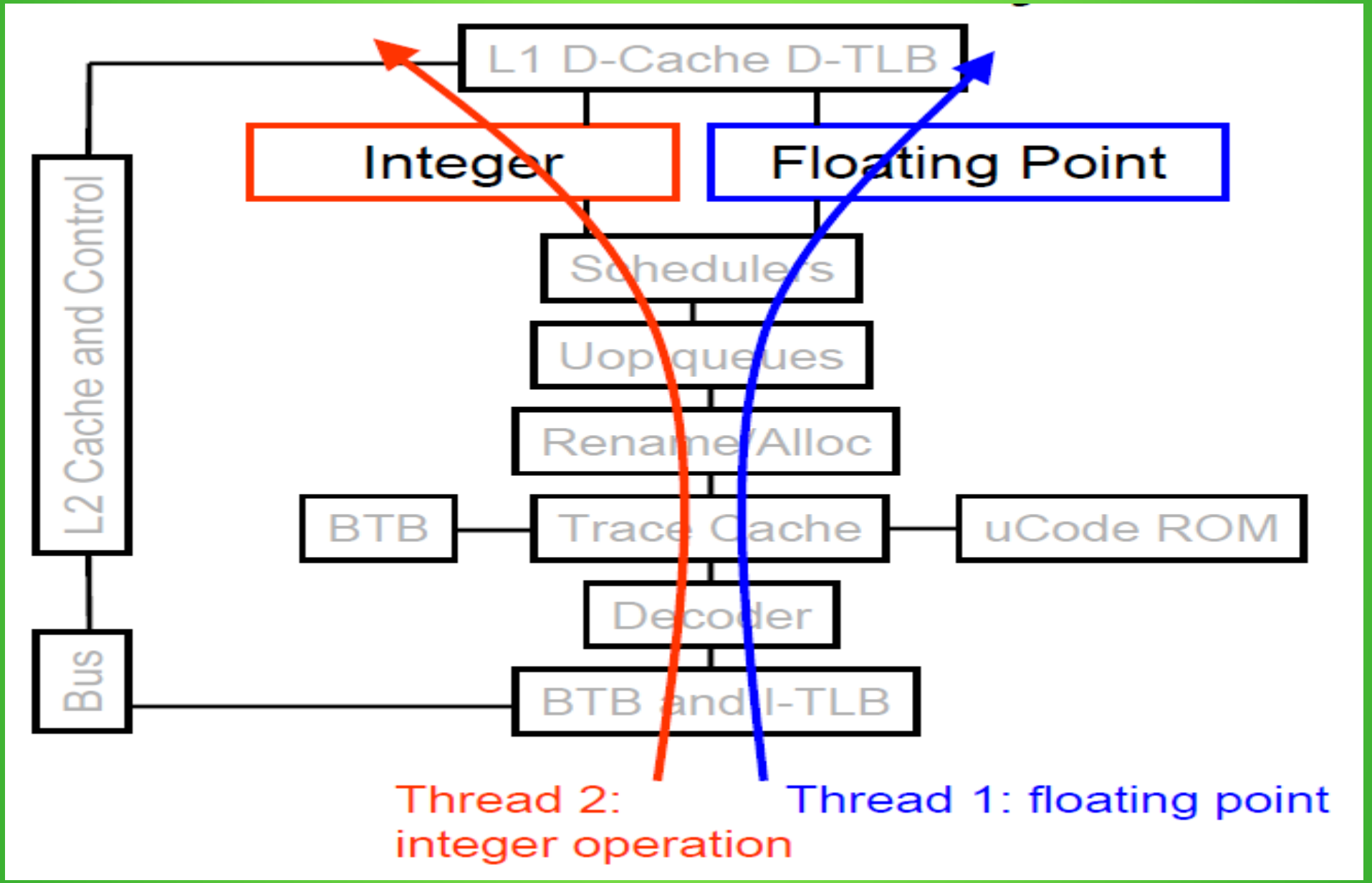
CPU:Architecture:ILP(SMT)



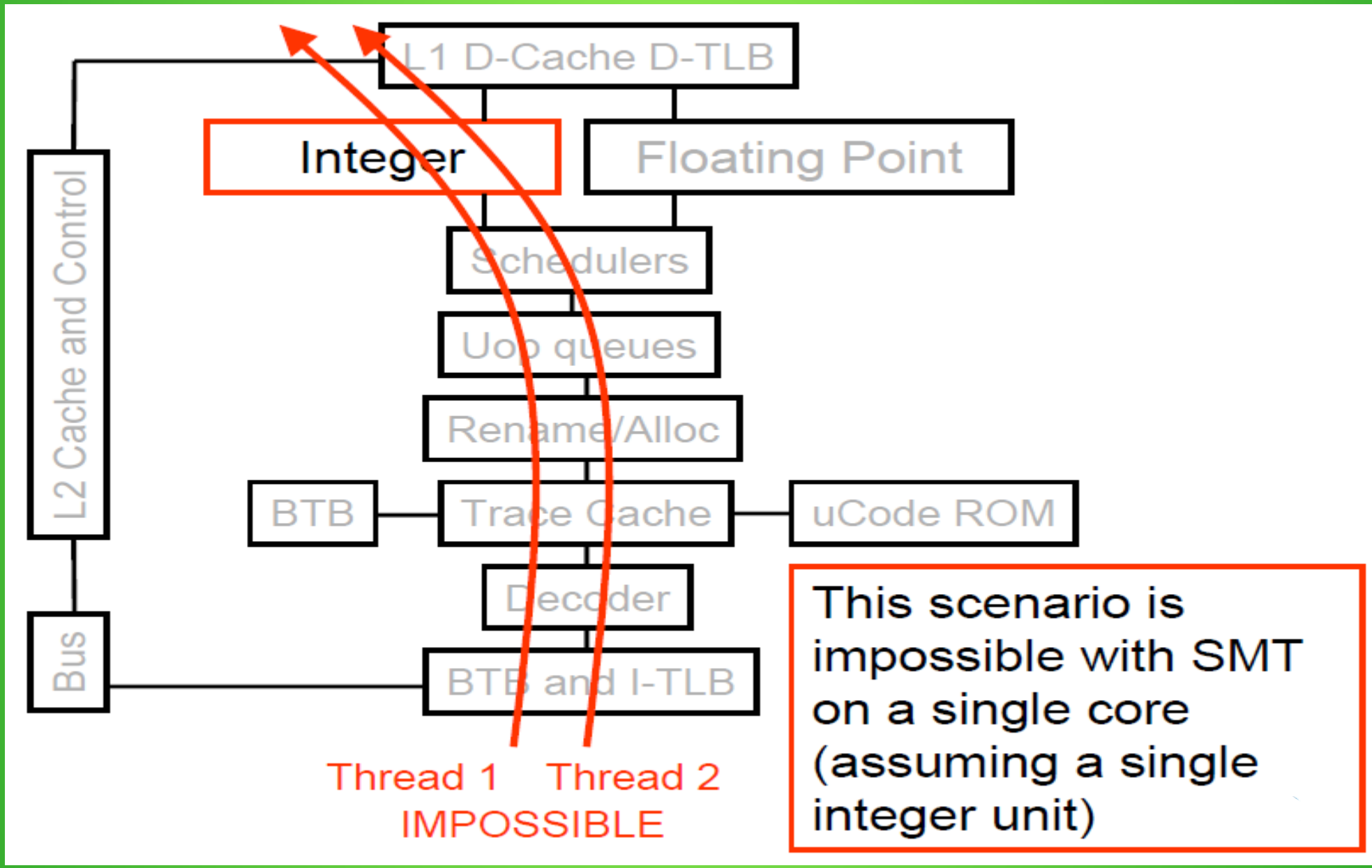
CPU:Architecture:ILP(SMT)



CPU:Architecture:ILP(SMT)



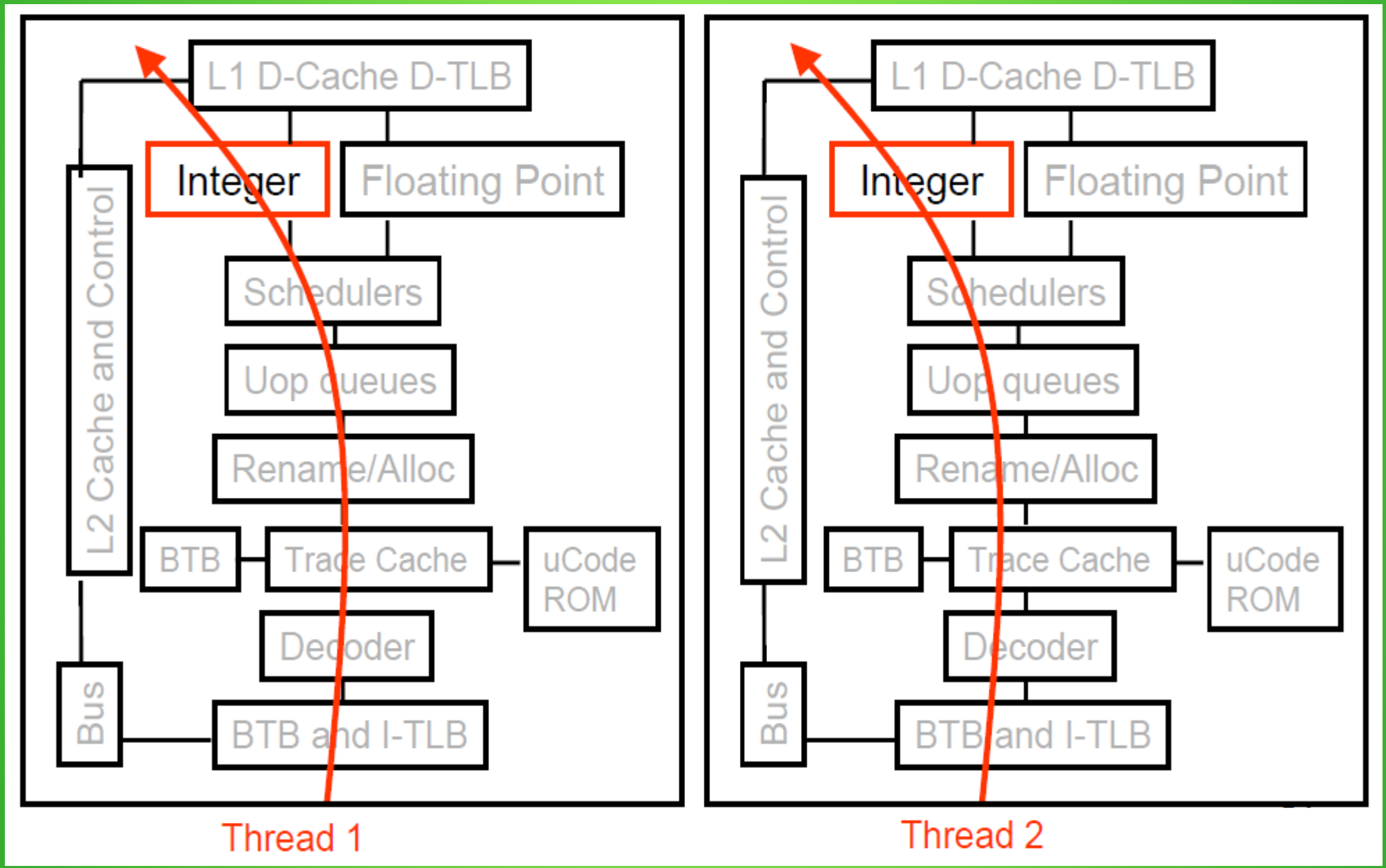
CPU:Architecture:ILP(SMT)



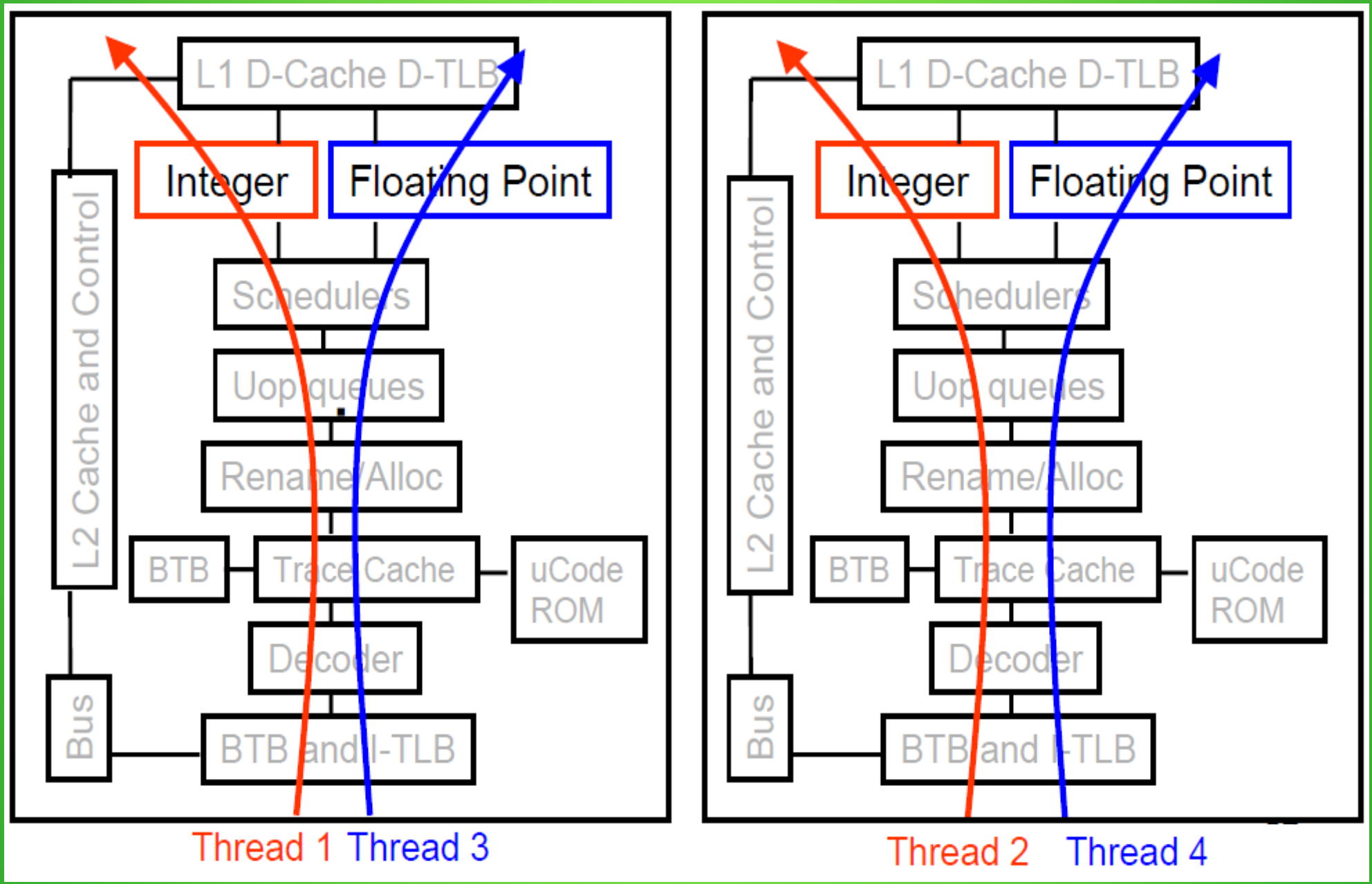
CPU:Architecture:ILP(SMT)

- SMT not a “true” parallel processor Enables better threading (e.g. up to 30%)
  - OS and applications perceive each simultaneous thread as a separate “virtual processor”
  - The chip has only a single copy of each resource
  - Compare to multi-core: each core has its own copy of resources

CPU:Architecture:ILP(SMT)



CPU:Architecture:ILP(SMT)



CPU:Architecture:TLP

- This is parallelism on a more coarser scale
  - Server can serve each client in a separate thread (Web server, database server)
  - A computer game can do AI, graphics, and physics in three separate threads
  - Single-core superscalar processors cannot fully exploit TLP
  - Multi-core architectures are the next step in processor evolution: explicitly exploiting TLP

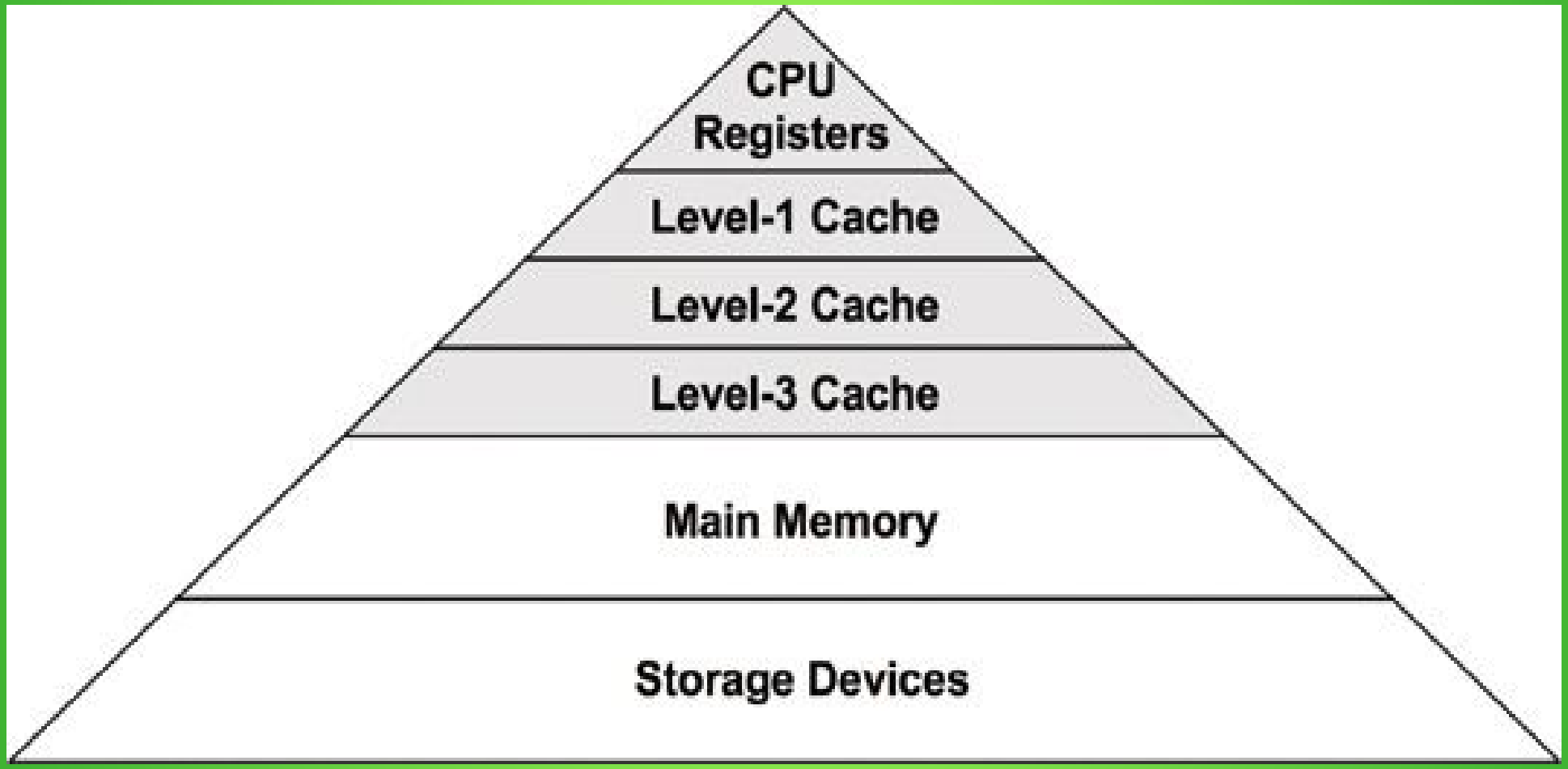
CPU:Measurements:IPC/PCI

- CPI, IPC
  - Cycles per instruction (CPI) is an important high-level metric for describing where a CPU is spending its clock cycles and for understanding the nature of CPU utilization. This metric may also be expressed as instructions per cycle (IPC), the inverse of CPI.
  - A high CPI indicates that CPUs are often stalled, typically for memory access. A low CPI indicates that CPUs are often not stalled and have a high instruction throughput. These metrics suggest where performance tuning efforts may be best spent.

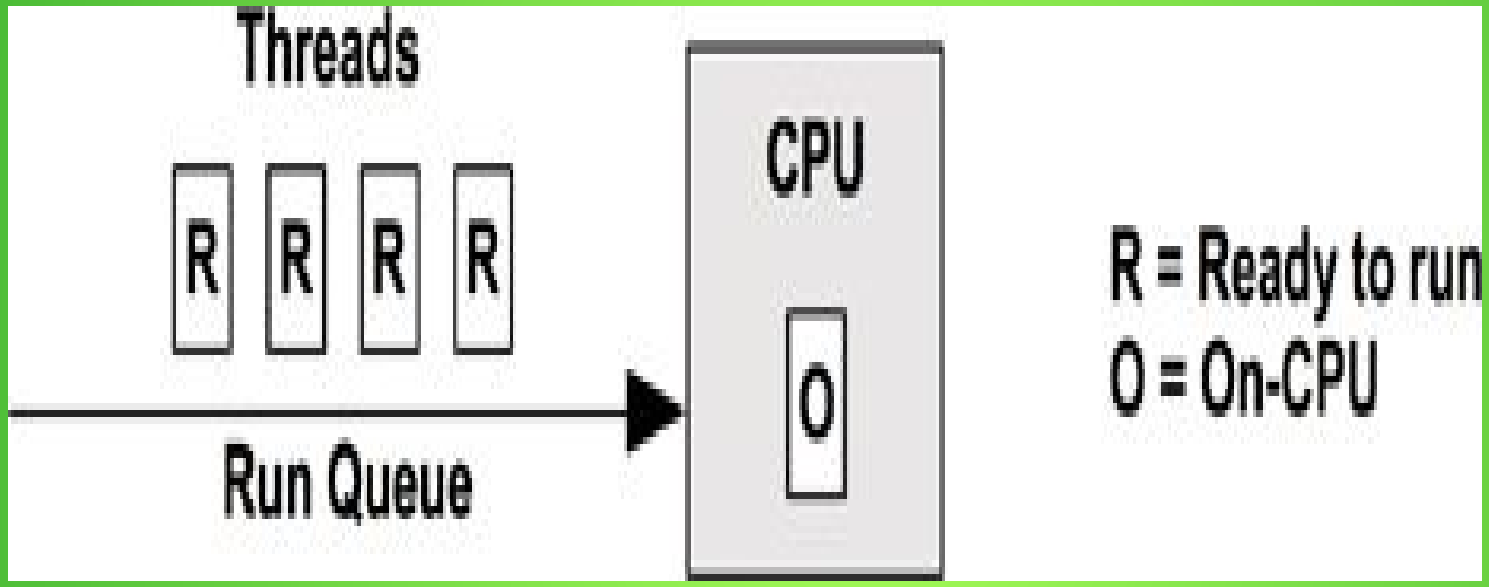
CPU:Measurements:IPC/PCI(Single Threaded TM being 4 with sse4 and int)

```
mohit@emptyminnd:~/Work/JINT$ perf stat java -Xmx4g -classpath $WORK_HOME/HARDW
esspatterns.MemoryAccessPatterns 3
0 - 25.06ns RANDOM_HEAP_WALK
1 - 25.49ns RANDOM_HEAP_WALK
2 - 26.06ns RANDOM_HEAP_WALK
3 - 22.82ns RANDOM_HEAP_WALK
4 - 22.12ns RANDOM_HEAP_WALK

 Performance counter stats for 'java -Xmx4g -classpath /home/mohit/Work/JINT/HA
accesspatterns.MemoryAccessPatterns 3':

      37580.307476 task-clock (msec)         #    1.001 CPUs utilized
             1,533 context-switches          #    0.041 K/sec
                42 cpu-migrations            #    0.001 K/sec
           1,29,420 page-faults              #    0.003 M/sec
   41,04,04,09,703 cycles                    #    1.092 GHz
   <not supported> stalled-cycles-frontend
   <not supported> stalled-cycles-backend
   11,77,80,41,247 instructions              #    0.29  insns per cycle
    1,37,79,82,011 branches                  #   36.668 M/sec
         65,51,444 branch-misses             #    0.48% of all branches

      37.528876130 seconds time elapsed

mohit@emptyminnd:~/Work/JINT$
```

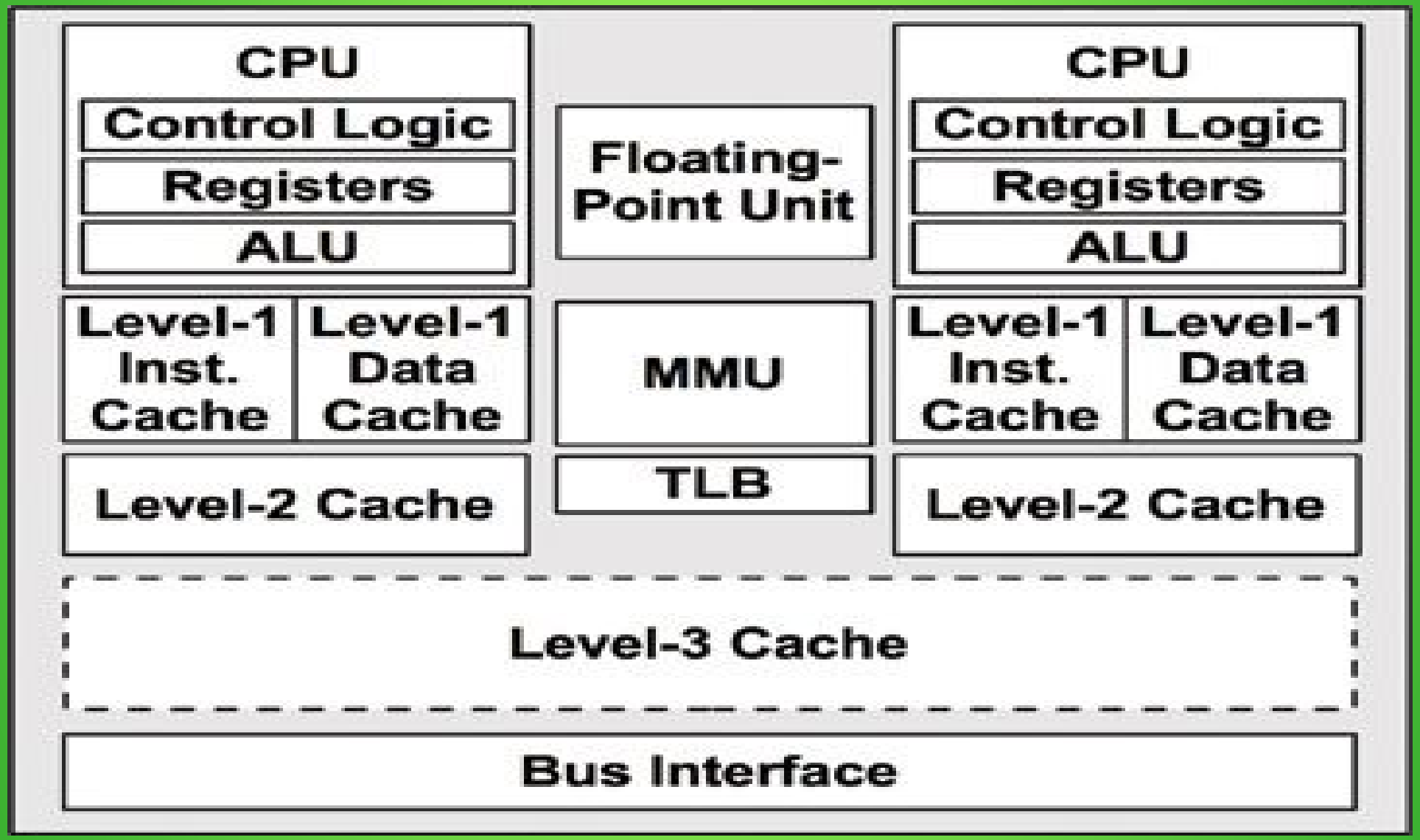
CPU:Architecture:cache



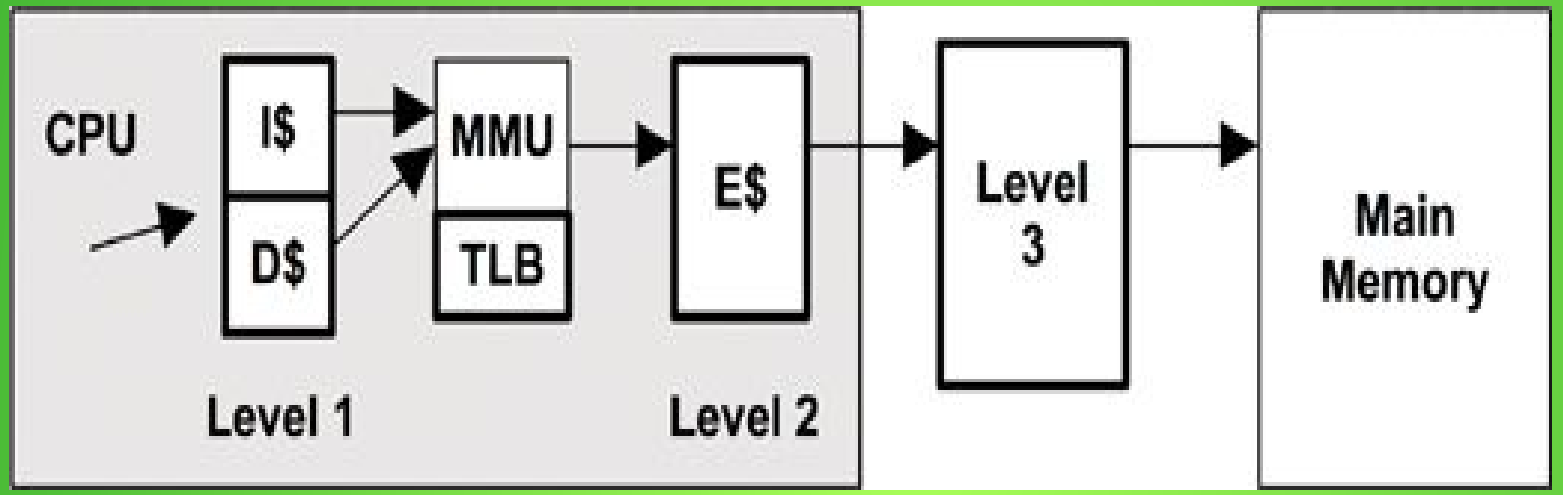
CPU:Architecture:run queue



CPU:Architecture:generic



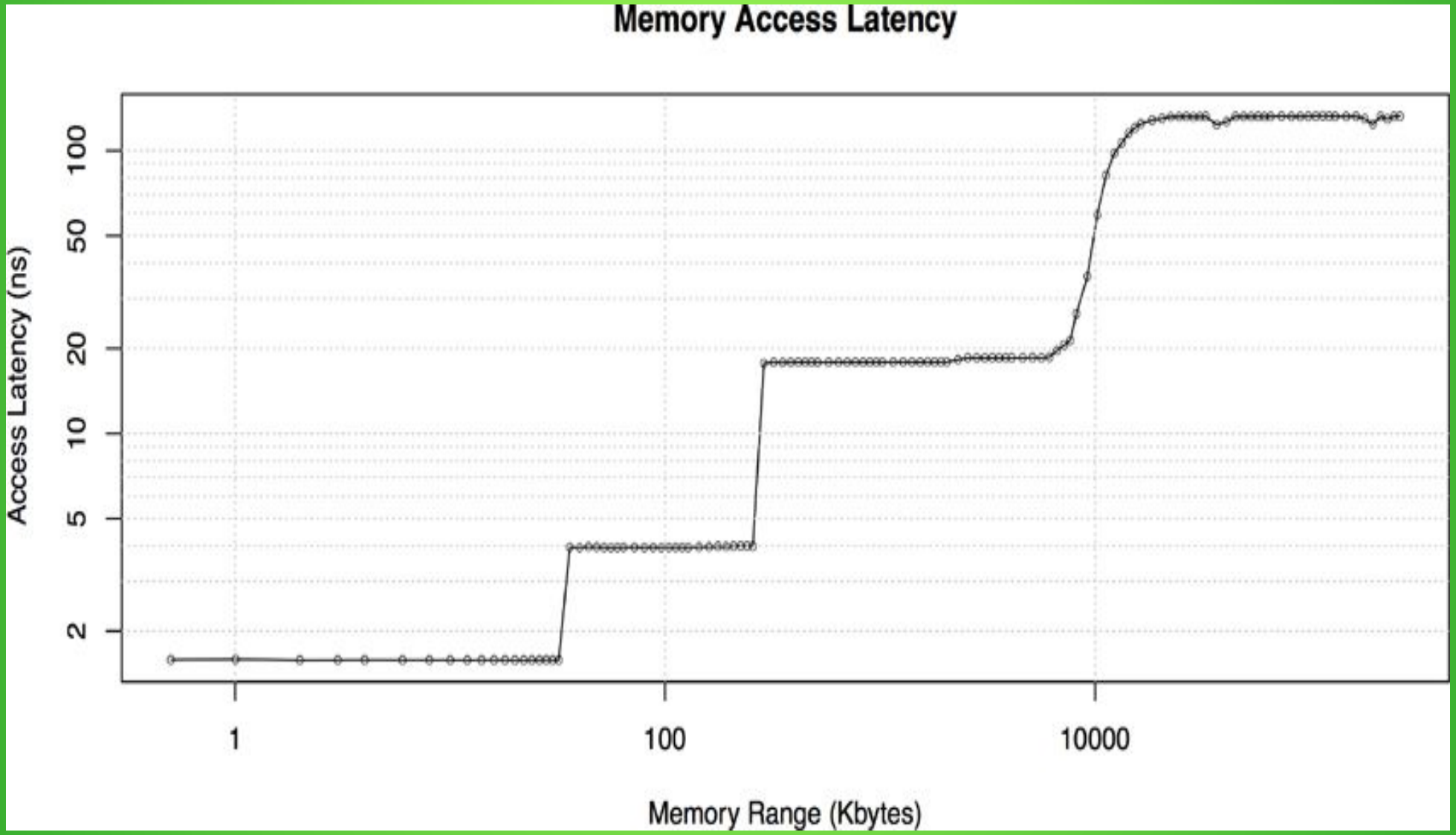
CPU:Architecture:cache access



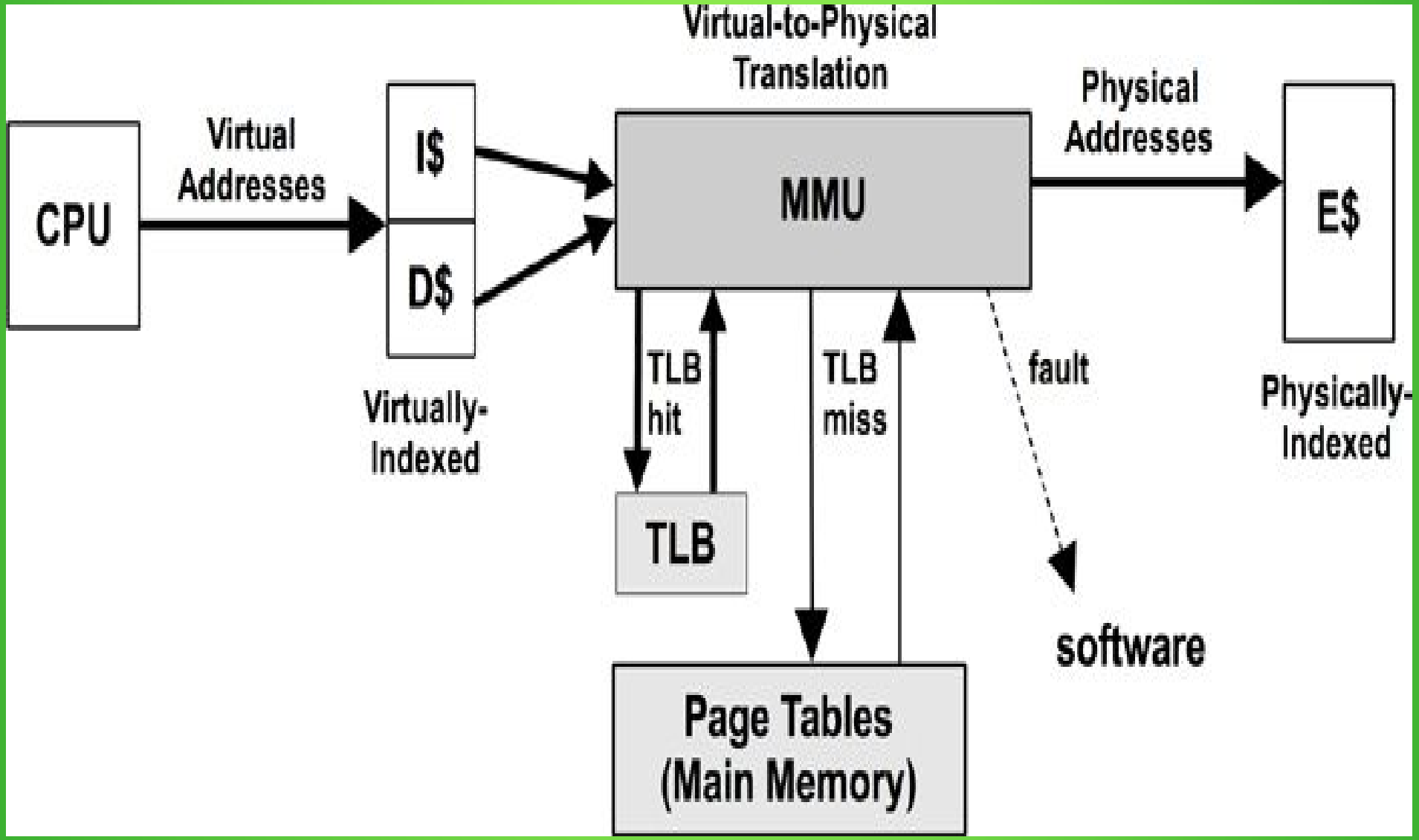
CPU:Architecture:cache sizes(Intel)

Processor	Date	Max Clock	Transistors	Data Bus	Level 1	Level 2	Level 3
8086	1978	8 MHz	29 K	16-bit	—		
Intel 286	1982	12.5 MHz	134 K	16-bit	—		
Intel 386 DX	1985	20 MHz	275 K	32-bit	—	—	—
Intel 486 DX	1989	25 MHz	1.2 M	32-bit	8 KB	—	—
Pentium	1993	60 MHz	3.1 M	64-bit	16 KB	—	—
Pentium Pro	1995	200 MHz	5.5 M	64-bit	16 KB	256/512 KB	—
Pentium II	1997	266 MHz	7 M	64-bit	32 KB	256/512 KB	—
Pentium III	1999	500 MHz	8.2 M	64-bit	32 KB	512 KB	—
Intel Xeon	2001	1.7 GHz	42 M	64-bit	8 KB	512 KB	—
Pentium M	2003	1.6 GHz	77 M	64-bit	64 KB	1 MB	—
Intel Xeon MP	2005	3.33 GHz	675 M	64-bit	16 KB	1 MB	8 MB
Intel Xeon 7410	2006	3.4 GHz	1.3 B	64-bit	64 KB	2 x 1 MB	16 MB
Intel Xeon 7460	2008	2.67 GHz	1.9 B	64-bit	64 KB	3 x 3 MB	16 MB
Intel Xeon 7560	2010	2.26 GHz	2.3 B	64-bit	64 KB	256 KB	24 MB
Intel Xeon E7-8870	2011	2.4 GHz	2.2 B	64-bit	64 KB	256 KB	30 MB

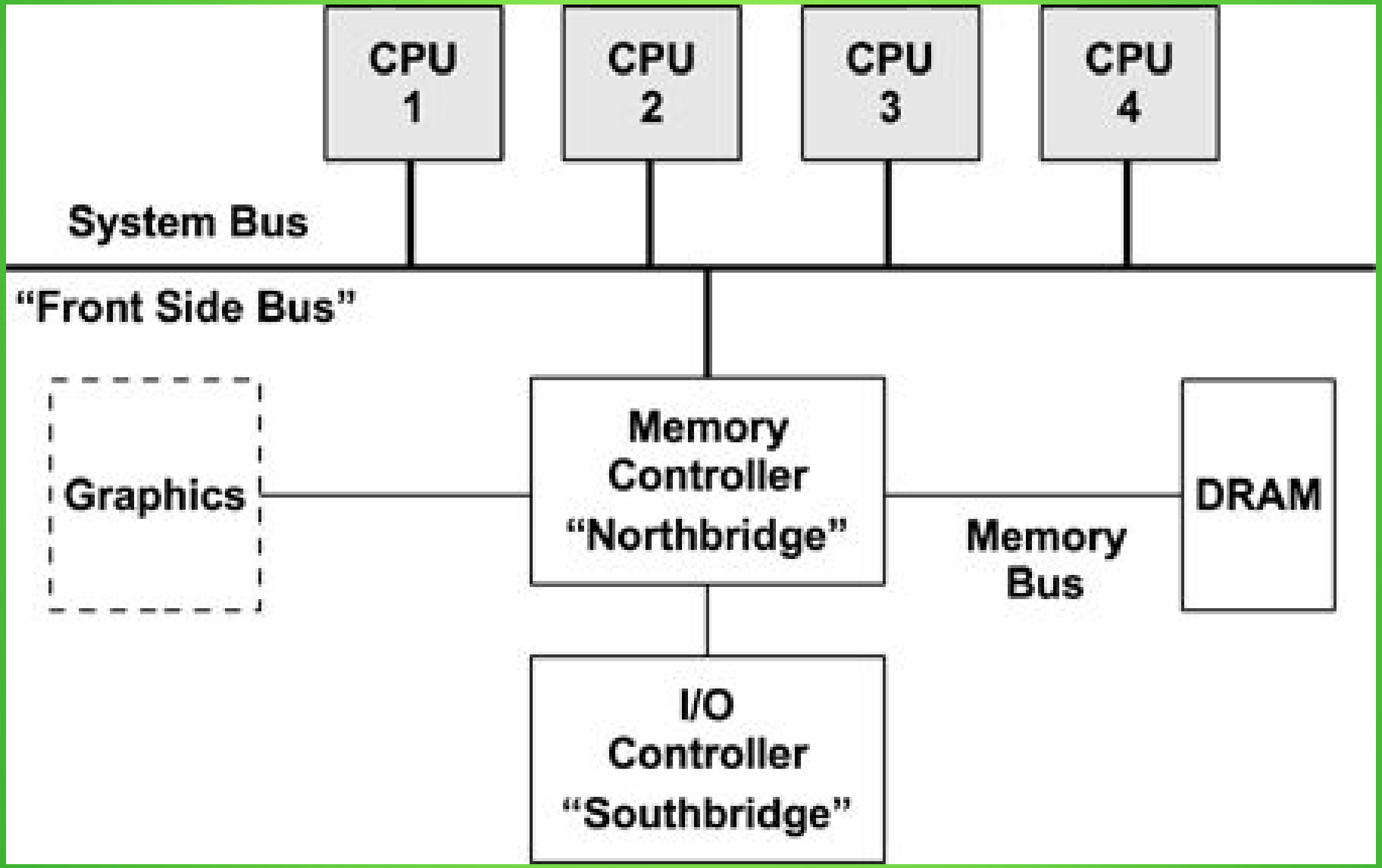
CPU:benchmark:lmbench



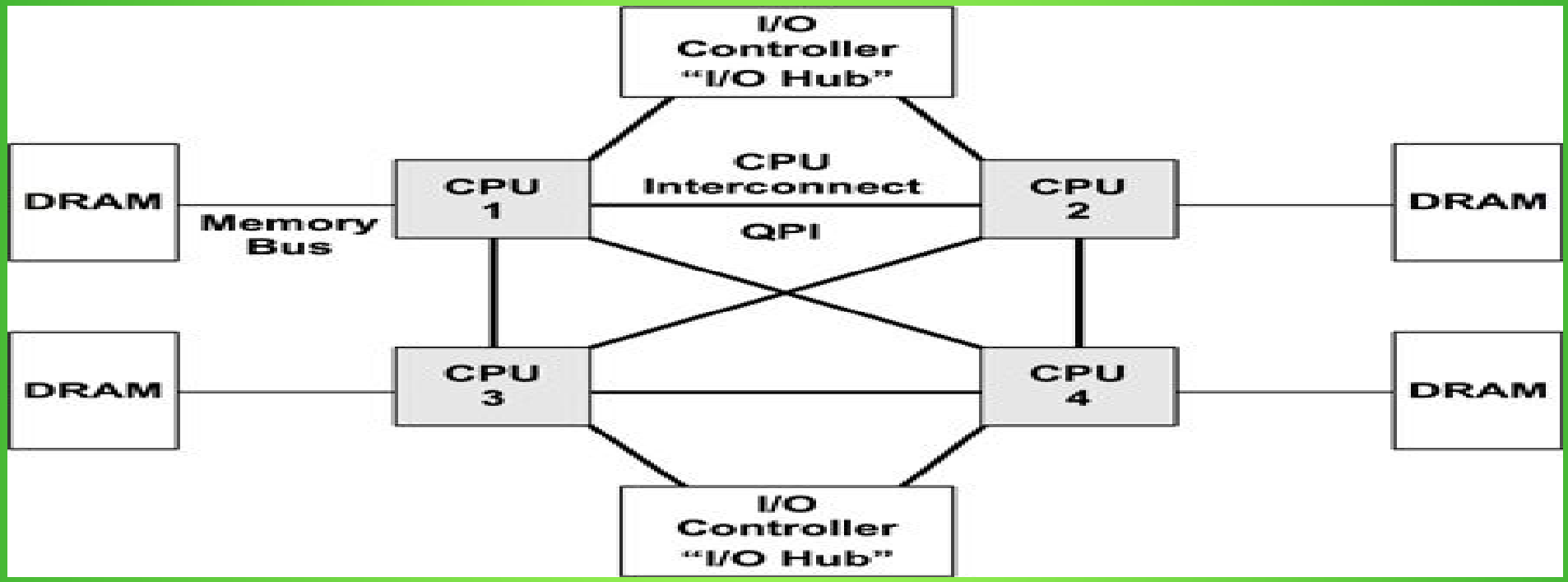
CPU:MMU


Virtual-to-Physical
Translation
CPU
Virtual
Addresses
I$
D$
MMU
Physical
Addresses
E$
Virtually-
Indexed
TLB
hit
TLB
miss
fault
Physically-
Indexed
TLB
software
Page Tables
(Main Memory)

CPU:Interconnect:FSB

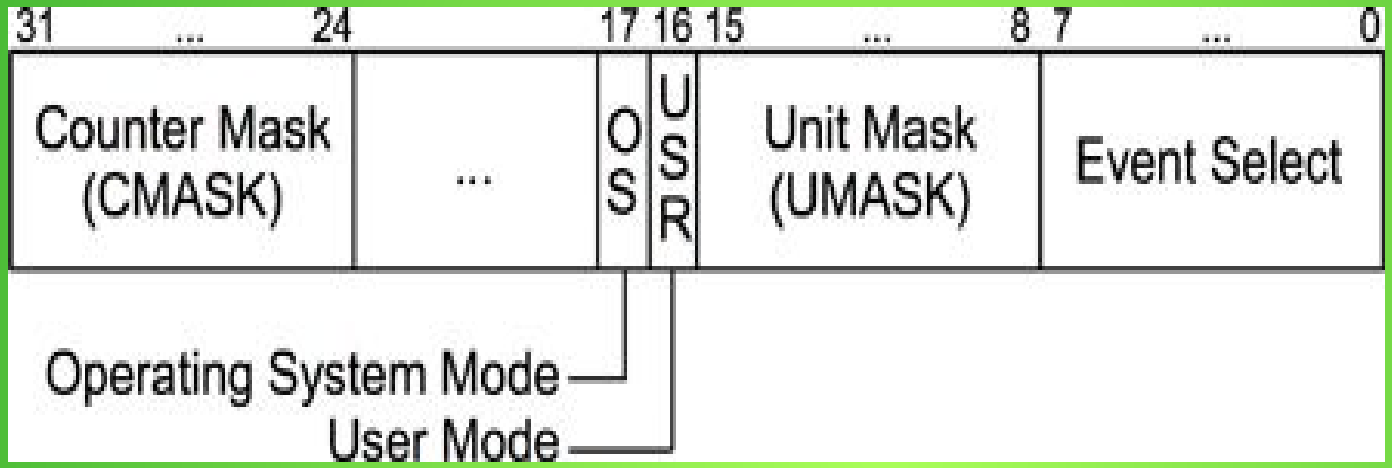


CPU:Interconnect:QPI



Intel	Transfer Rate	Width	Bandwidth
FSB (2007)	1.6 GT/s	8 bytes	12.8 Gbytes/s
QPI (2008)	6.4 GT/s	2 bytes	25.6 Gbytes/s

CPU:CPC:Perf tools are based on these.



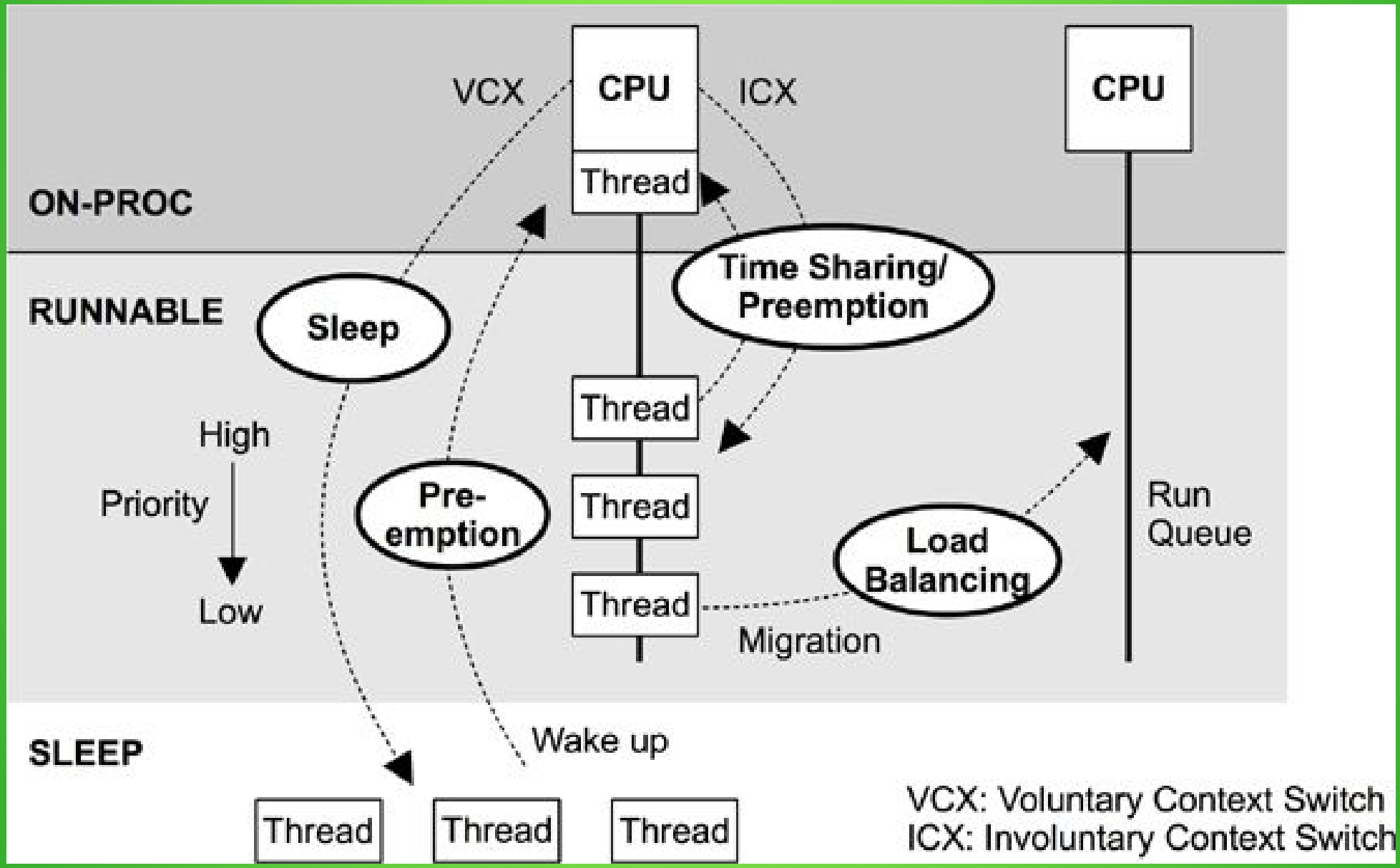
CPU:CPC:Perf tools are based on these.

Event Select	UMASK	Unit	Name	Description
0x43	0x00	data cache	DATA_MEM_ REFS	All loads from any memory type. All stores to any memory type. Each part of a split is counted separately. . . . Does not include I/O accesses or other nonmemory accesses.
0x48	0x00	data cache	DCU_MISS_ OUTSTANDING	Weighted number of cycles while a DCU miss is outstanding, incremented by the number of outstanding cache misses at any particular time. Cacheable read requests only are considered. . . .
0x80	0x00	instruction fetch unit	IFU_IFETCH	Number of instruction fetches, both cacheable and noncacheable, including UC (uncacheable) fetches.
0x28	0x0F	L2 cache	L2_IFETCH	Number of L2 instruction fetches. . . .
0xC1	0x00	floating-point unit	FLOPS	Number of computational floating-point operations retired. . . .
0x7E	0x00	external bus logic	BUS_SNOOP_ STALL	Number of clock cycles during which the bus is snoop stalled.
0xC0	0x00	instruction decoding and retirement	INST_RETIRED	Number of instructions retired.
0xC8	0x00	interrupts	HW_INT_RX	Number of hardware interrupts received.
0xC5	0x00	branches	BR_MISS_PRED_ RETIRED	Number of mispredicted branches retired.
0xA2	0x00	stalls	RESOURCE_ STALLS	Incremented by one during every cycle for which there is a resource-related stall. . . .
0x79	0x00	clocks	CPU_CLK_ UNHALTED	Number of cycles during which the processor is not halted.

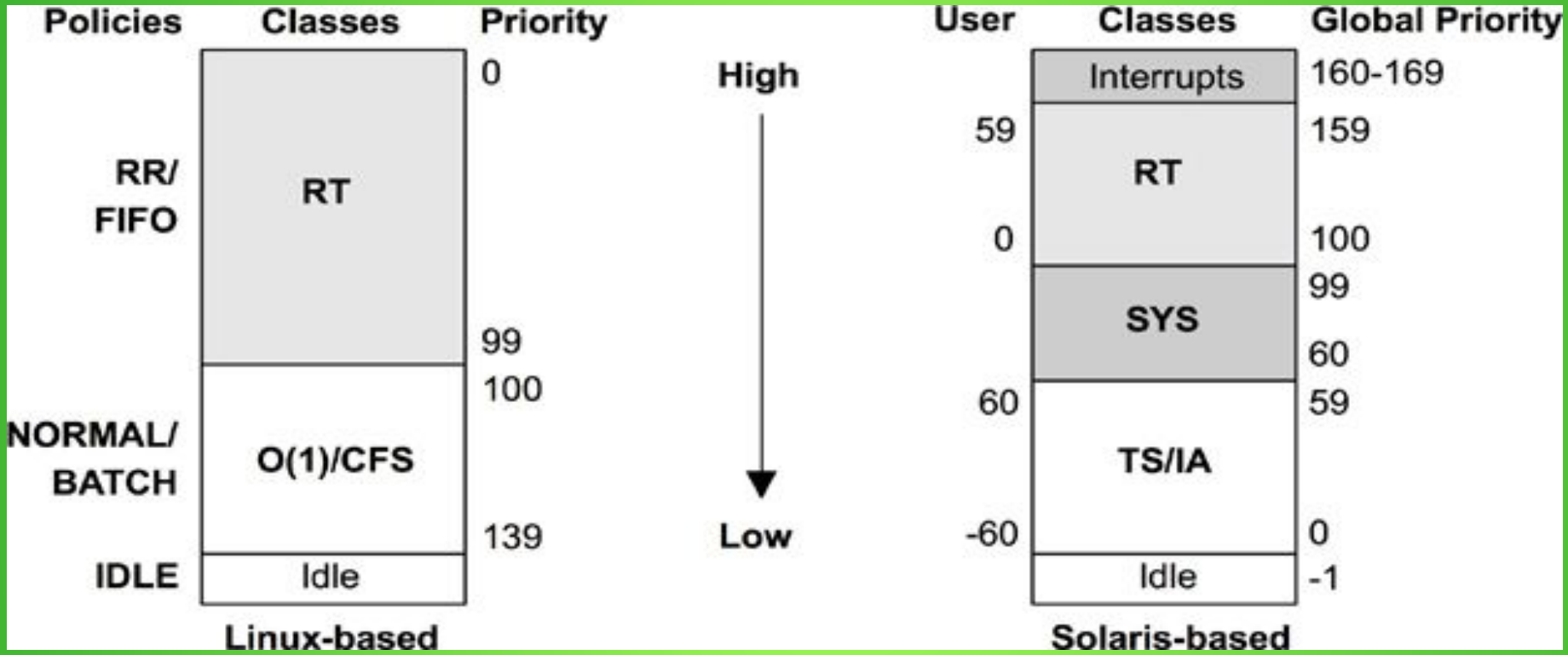
CPU:CPC:perf list

```
mohit@emptyminnd:~/Work/JINT$ perf list

List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  cache-references                                   [Hardware event]
  cache-misses                                       [Hardware event]
  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  ref-cycles                                         [Hardware event]

  cpu-clock                                          [Software event]
  task-clock                                         [Software event]
  page-faults OR faults                              [Software event]
  context-switches OR cs                             [Software event]
  cpu-migrations OR migrations                       [Software event]
  minor-faults                                       [Software event]
  major-faults                                       [Software event]
  alignment-faults                                   [Software event]
  emulation-faults                                   [Software event]
  dummy                                              [Software event]

  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-dcache-store-misses                             [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
  LLC-prefetches                                     [Hardware cache event]
  dTLB-loads                                         [Hardware cache event]
```

CPU:OS(*nix):scheduler



CPU:OS(*nix):scheduler:priority



CPU:profiling:perf record -F 99 -p <PID> sleep 10

```
17.74%  acroread  libglib-2.0.so.0.4002.0  [.] 0x00000000000351e8
12.85%  acroread  acroread                 [.] _start
 9.84%  acroread  [kernel.kallsyms]        [k] fetch_task_cputime
 9.41%  acroread  libpthread-2.19.so       [.] __pthread_mutex_unlock_usercnt
 7.01%  acroread  [kernel.kallsyms]        [k] aa_revalidate_sk
 6.18%  acroread  libc-2.19.so             [.] 0x000000000012d31e
 5.50%  acroread  acroread                 [.] 0x0000000000a07bc5
 5.49%  acroread  libglib-2.0.so.0.4002.0  [.] g_source_set_priority
 5.40%  acroread  EScript.api              [.] 0x000000000011e20c
 4.58%  acroread  libxcb.so.1.1.0          [.] 0x0000000000009760
 4.40%  acroread  libpthread-2.19.so       [.] pthread_mutex_lock
 3.90%  acroread  libstdc++.so.6.0.19      [.] 0x0000000000046e40
 3.90%  acroread  [kernel.kallsyms]        [k] update_curr
 3.51%  acroread  libglib-2.0.so.0.4002.0  [.] g_mutex_lock
 0.29%  acroread  [vdso]                   [.] 0x0000000000000431
 0.01%  acroread  [kernel.kallsyms]        [k] update_blocked_averages
 0.00%  acroread  [kernel.kallsyms]        [k] native_write_msr_safe
```

CPU:Priority

- Unix has always provided a **nice()** system call for adjusting process priority, which sets a nice-ness value.
- Positive nice values result in lower process priority (nicer), and negative values—which can be set only by the superuser (root)—result in higher priority.

CPU:Priority:java

```
mohit@emptyminnd:~/Work/JINT$ java -XX:+PrintFlagsFinal | grep Priority
     intx CompilerThreadPriority                    = -1                  {product}
     intx DefaultThreadPriority                     = -1                  {product}
     intx JavaPriority10_To_OSPriority              = -1                  {product}
     intx JavaPriority1_To_OSPriority               = -1                  {product}
     intx JavaPriority2_To_OSPriority               = -1                  {product}
     intx JavaPriority3_To_OSPriority               = -1                  {product}
     intx JavaPriority4_To_OSPriority               = -1                  {product}
     intx JavaPriority5_To_OSPriority               = -1                  {product}
     intx JavaPriority6_To_OSPriority               = -1                  {product}
     intx JavaPriority7_To_OSPriority               = -1                  {product}
     intx JavaPriority8_To_OSPriority               = -1                  {product}
     intx JavaPriority9_To_OSPriority               = -1                  {product}
     intx ThreadPriorityPolicy                      = 0                   {product}
     bool ThreadPriorityVerbose                     = false               {product}
     intx VMThreadPriority                          = -1                  {product}
```

CPU:Binding(Affinity)

- There are generally two ways this is performed:
  - **Process binding**: configuring a process to run only on a single CPU, or only on one CPU from a defined set.
  - **Exclusive CPU sets**: partitioning a set of CPUs that can be used only by the process(es) assigned to them. This can improve CPU cache further, as when the process is idle other processes cannot use the CPUs, leaving the caches warm.

CPU:Binding(Affinity):binding:linux

```
$ taskset -pc 7-10 10790
pid 10790's current affinity list: 0-15
pid 10790's new affinity list: 7-10
```

CPU:Binding(Affinity):binding:linux

```
$ taskset -pc 7-10 10790
pid 10790's current affinity list: 0-15
pid 10790's new affinity list: 7-10
```

CPU:Binding(Affinity):binding:Solaris

```
$ taskset -pc 7-10 10790
pid 10790's current affinity list: 0-15
pid 10790's new affinity list: 7-10
```

CPU:Binding(Affinity):exclusive:linux

```
# mkdir /dev/cpuset
# mount -t cpuset cpuset /dev/cpuset
# cd /dev/cpuset
# mkdir prodset              # create a cpuset called "prodset"
# cd prodset
# echo 7-10 > cpus           # assign CPUs 7-10
# echo 1 > cpu_exclusive     # make prodset exclusive
# echo 1159 > tasks          # assign PID 1159 to prodset
```

CPU:Binding(Affinity):exclusive:solaris:psrset

CPU:Binding(Affinity):linux:Irqs and Kernel Modules

```
Affinity for ISR and Kernel Modules
grep wlan /proc/interrupts
cat /proc/irq/18/smp_affinity
echo 1 >/proc/irq/<irqnum>/smp_affinity
```

CPU:Measurements:vmstat

```
$ vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
15  0   2852 46686812 279456 1401196    0    0     0     0    0    0  0  0 100  0
16  0   2852 46685192 279456 1401196    0    0     0     0 2136 36607 56 33 11  0
15  0   2852 46685952 279456 1401196    0    0     0    56 2150 36905 54 35 11  0
15  0   2852 46685960 279456 1401196    0    0     0     0 2173 36645 54 33 13  0
[...]
```

CPU:Measurements:vmstat

- r: run-queue length—the total number of runnable threads(on **Linux this is running + runnables**)
  - man page currently describes it as something else—“the number of processes waiting for run time”
- b: Threads waiting on IO
- us: user-time
- sy: system-time (kernel)
- id: idle
- wa: wait I/O, which measures CPU idle when threads are blocked on disk I/O
- st: stolen , which for virtualized environments shows CPU time spent servicing other tenants

CPU:Measurements:mpstat

```
$ mpstat -P ALL 1
02:47:49   CPU    %usr   %nice    %sys %iowait    %irq   %soft %steal %guest  %idle
02:47:50   all   54.37    0.00   33.12    0.00    0.00    0.00    0.00    0.00  12.50
02:47:50     0   22.00    0.00   57.00    0.00    0.00    0.00    0.00    0.00  21.00
02:47:50     1   19.00    0.00   65.00    0.00    0.00    0.00    0.00    0.00  16.00
02:47:50     2   24.00    0.00   52.00    0.00    0.00    0.00    0.00    0.00  24.00
02:47:50     3  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   0.00
02:47:50     4  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   0.00
02:47:50     5  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   0.00
02:47:50     6  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   0.00
02:47:50     7   16.00    0.00   63.00    0.00    0.00    0.00    0.00    0.00  21.00
02:47:50     8  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   0.00
02:47:50     9   11.00    0.00   53.00    0.00    0.00    0.00    0.00    0.00  36.00
02:47:50    10  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   0.00
02:47:50    11   28.00    0.00   61.00    0.00    0.00    0.00    0.00    0.00  11.00
02:47:50    12   20.00    0.00   63.00    0.00    0.00    0.00    0.00    0.00  17.00
02:47:50    13   12.00    0.00   56.00    0.00    0.00    0.00    0.00    0.00  32.00
02:47:50    14   18.00    0.00   60.00    0.00    0.00    0.00    0.00    0.00  22.00
02:47:50    15  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   0.00
[...]
```

CPU:Measurements:mpstat

- CPU: logical CPU ID, or all for summary
  - %usr: user-time
  - %nice: user-time for processes with a nice’d priority
  - %sys: system-time (kernel)
  - %iowait: I/O wait
  - %irq: hardware interrupt CPU usage
  - %soft: software interrupt CPU usage
  - %steal: time spent servicing other tenants
  - %guest: CPU time spent in guest virtual mac

CPU:Measurements:sar

- The system activity reporter, sar(1), can be used to observe current activity and can be configured to archive and report historical statistics.

Option	Metrics	Description
`-P ALL`	`%user %nice %system %iowait %steal %idle`	per-CPU utilization
`-u`	`%user %nice %system %iowait %steal %idle`	CPU utilization
`-q`	`runq-sz`	CPU run-queue size
`-B`	`pgpgin/s pgpgout/s fault/s majflt/s pgfree/s pgscank/s pgscand/s pgsteal/s %vmeff`	paging statistics
`-H`	`hbhugfree hbhugused`	huge pages
`-r`	`kbmemfree kbmemused kbbuffers kbcached kbcommit %commit kbactive kbinact`	memory utilization
`-R`	`frpg/s bufpg/s campg/s`	memory statistics

CPU:Measurements:sar

-S	kbswpfree kbswpused kbswpcad	swap utilization
-W	pswpin/s pswpout/s	swapping statistics
-v	dentused file-nr inode-nr	kernel tables
-d	tps rd_sec/s wr_sec/s avgrq-sz avgqu-sz await svctm %util	disk statistics
-n DEV	rxpck/s txpck/s rxkB/s txkB/s	network interface statistics
-n EDEV	rxerr/s txerr/s coll/s rxdrop/s txdrop/s rxfifo/s txfifo/s	network interface errors
-n IP	irec/s fwddgm/s orq/s	IP statistics
-n EIP	idisc/s odisc/s	IP errors
-n TCP	active/s passive/s iseg/s oseg/s	TCP statistics
-n ETCP	atmptf/s retrans/s	TCP errors
-n SOCK	totsck ip-frag tcp-tw	socket statistics

CPU:Measurements:ps

```
$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  23772  1948 ?        Ss    2012   0:04 /sbin/init
root         2  0.0  0.0      0     0 ?        S     2012   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S     2012   0:26 [ksoftirqd/0]
root         4  0.0  0.0      0     0 ?        S     2012   0:00 [migration/0]
root         5  0.0  0.0      0     0 ?        S     2012   0:00 [watchdog/0]
[...]
web      11715 11.3  0.0 632700 11540 pts/0    Sl    01:36   0:27 node indexer.js
web      11721 96.5  0.1 638116 52108 pts/1    Rl+   01:37   3:33 node proxy.js
[...]
```

CPU:Measurements:linux:top(expensive and can miss short lived processes. Try atop)

```
$ top
top - 01:38:11 up 63 days,  1:17,  2 users,  load average: 1.57, 1.81, 1.77
Tasks: 256 total,   2 running, 254 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.0%us,  3.6%sy,  0.0%ni, 94.2%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  49548744k total, 16746572k used, 32802172k free,   182900k buffers
Swap: 100663292k total,        0k used, 100663292k free, 14925240k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
11721 web       20   0  623m  50m 4984 R   93  0.1   0:59.50 node
11715 web       20   0  619m  20m 4916 S   25  0.0   0:07.52 node
   10 root      20   0     0    0    0 S    1  0.0 248:52.56 ksoftirqd/2
   51 root      20   0     0    0    0 S    0  0.0   0:35.66 events/0
11724 admin     20   0 19412 1444  960 R    0  0.0   0:00.07 top
    1 root      20   0 23772 1948 1296 S    0  0.0   0:04.35 init
```

CPU:Measurements:solaris:prstat

```
$ prstat
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 21722 101        23G   20G cpu0    59    0  72:23:41 2.6% beam.smp/594
 21495 root      321M  304M sleep    1    0   2:57:41 0.9% node/5
 20721 root      345M  328M sleep    1    0   2:49:53 0.8% node/5
 20861 root      348M  331M sleep    1    0   2:57:07 0.7% node/6
 15354 root      172M  156M cpu9     1    0   0:31:42 0.7% node/5
 21738 root      179M  143M sleep    1    0   2:37:48 0.7% node/4
 20385 root      196M  174M sleep    1    0   2:26:28 0.6% node/4
 23186 root      172M  149M sleep    1    0   0:10:56 0.6% node/4
 18513 root      174M  138M cpu13    1    0   2:36:43 0.6% node/4
 21067 root      187M  162M sleep    1    0   2:28:40 0.5% node/4
 19634 root      193M  170M sleep    1    0   2:29:36 0.5% node/4
 10163 root      113M  109M sleep    1    0  12:31:09 0.4% node/3
 12699 root      199M  177M sleep    1    0   1:56:10 0.4% node/4
 37088 root     1069M 1056M sleep   59    0  38:31:19 0.3% qemu-system-x86/4
 10347 root       67M   64M sleep    1    0  11:57:17 0.3% node/3
Total: 390 processes, 1758 lwps, load averages: 3.89, 3.99, 4.31
```

CPU:Measurements:linux:pidstat

```
$ pidstat 1
Linux 2.6.35-32-server (dev7)   11/12/12        _x86_64_        (16 CPU)

22:24:42          PID    %usr %system  %guest    %CPU   CPU  Command
22:24:43         7814    0.00    1.98    0.00    1.98     3  tar
22:24:43         7815   97.03    2.97    0.00  100.00    11  gzip

22:24:43          PID    %usr %system  %guest    %CPU   CPU  Command
22:24:44          448    0.00    1.00    0.00    1.00     0  kjournald
22:24:44         7814    0.00    2.00    0.00    2.00     3  tar
22:24:44         7815   97.00    3.00    0.00  100.00    11  gzip
22:24:44         7816    0.00    2.00    0.00    2.00     2  pidstat
[...]
```

CPU:Measurements:linux:time

```
$ time cksum Fedora-16-x86_64-Live-Desktop.iso
560560652 633339904 Fedora-16-x86_64-Live-Desktop.iso

real    0m5.105s
user    0m2.810s
sys     0m0.300s
$ time cksum Fedora-16-x86_64-Live-Desktop.iso
560560652 633339904 Fedora-16-x86_64-Live-Desktop.iso

real    0m2.474s
user    0m2.340s
sys     0m0.130s
```

CPU:Measurements:linux:time

```
$ /usr/bin/time -v cp fileA fileB
        Command being timed: "cp fileA fileB"
        User time (seconds): 0.00
        System time (seconds): 0.26
        Percent of CPU this job got: 24%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.08
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 3792
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 294
        Voluntary context switches: 1082
        Involuntary context switches: 1
        Swaps: 0
        File system inputs: 275432
        File system outputs: 275432
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

CPU:KernelProfiling

```
# Sample on-CPU functions for the specified command, at 99 Hertz:
perf record -F 99 command

# Sample on-CPU functions for the specified PID, at 99 Hertz, until Ctrl-C:
perf record -F 99 -p PID

# Sample on-CPU functions for the specified PID, at 99 Hertz, for 10 seconds:
perf record -F 99 -p PID sleep 10

# Sample CPU stack traces for the specified PID, at 99 Hertz, for 10 seconds:
perf record -F 99 -p PID -g -- sleep 10

# Sample CPU stack traces for the PID, using dwarf to unwind stacks, at 99 Hertz, for 10 seconds:
perf record -F 99 -p PID -g dwarf sleep 10

# Sample CPU stack traces for the entire system, at 99 Hertz, for 10 seconds:
perf record -F 99 -ag -- sleep 10

# Sample CPU stack traces for the entire system, with dwarf stacks, at 99 Hertz, for 10 seconds:
perf record -F 99 -ag dwarf sleep 10

# Sample CPU stack traces, once every 10,000 Level 1 data cache misses, for 5 seconds:
perf record -e L1-dcache-load-misses -c 10000 -ag -- sleep 5

# Sample CPU stack traces, once every 100 last level cache misses, for 5 seconds:
perf record -e LLC-load-misses -c 100 -ag -- sleep 5
```

CPU:Bios Tuning

- Processor Options (BIOS Tuning)
  - Processors typically provide settings to enable, disable, and tune processor-level features. On x86 systems, these are typically accessed via the BIOS settings menu at boot time.
  - The settings usually provide maximum performance by default and don’t need to be adjusted. Bearing in mind that, for production use, Turbo Boost should be enabled for slightly faster performance).