HPC on linux clusters Nodes and Networks hardware

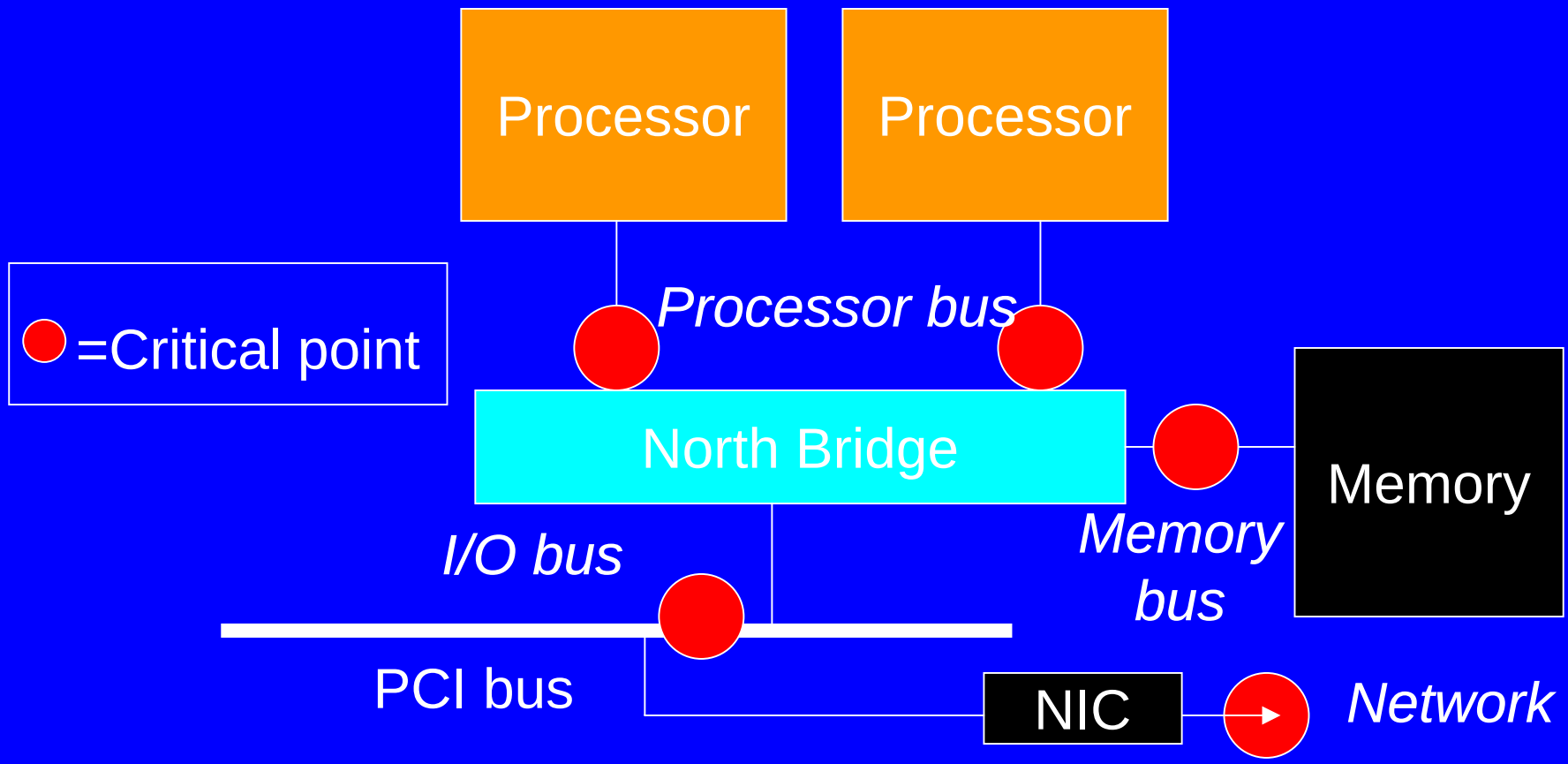
Roberto Innocente

inno@sissa.it

Overview

- Nodes:
  - CPU
  - Processor Bus
  - I/O bus
- Networks
  - boards
  - switches

Typical Node Architecture



Computer families/1

- RISC (Reduced Instruction Set Computer)
  - small set of simple instructions(Mips,alpha)
  - also called Load/Store architecture because operations are done in registers and only load/store instructions can access memory
  - instructions are typically hardwired and require few cycles
  - instructions have fixed length so that is easy to parse them
- CISC (Complex Instruction Set Computer)
  - large set of complex instructions (VAX, x86)
  - many instructions can have operands in memory
  - variable length instructions
  - many instructions require many cycles to complete

Computer families/2

It is very difficult to optimize computers with CISC instruction sets, because it is difficult to predict what will be the effect of the instructions.

For this reason today high performance processors have a RISC core even if the external instruction set is CISC.

Starting with the Pentium Pro, Intel x86 processors in fact translate x86 instruction into 1 or more RISC microops (uops).

Micro architecture

- Superscalar
- OOO (Out Of Order) execution
- Pipelining: super/hyper pipelining
- Branch prediction/speculative execution

Superscalar

It’s a CPU having multiple functional execution units and able to dispatch multiple instruction per cycle (double issue, quad issue ,...).

Pentium 4 has 7 distinct functional units:

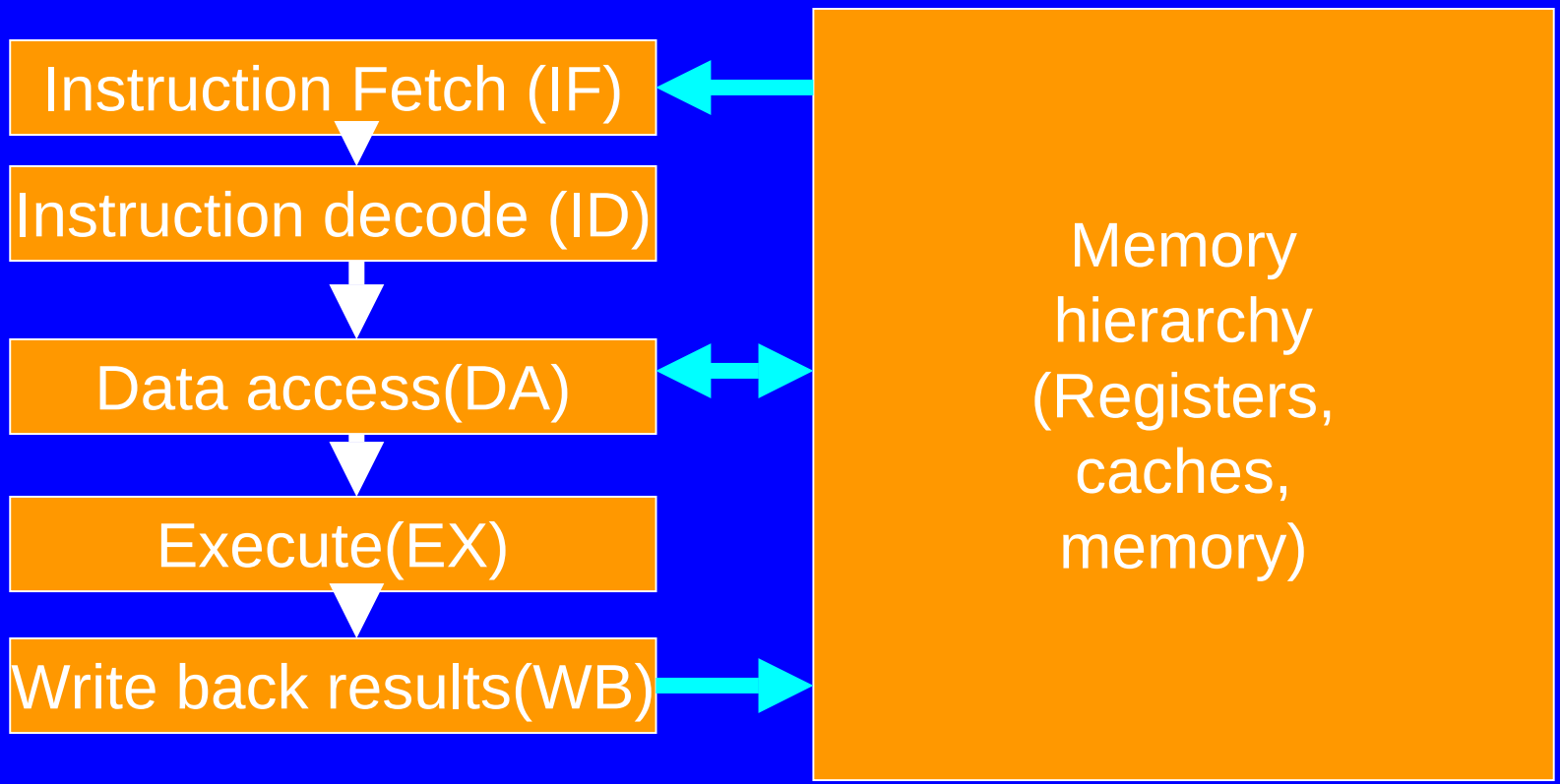
Load, Store, 2 x double speed ALUs, normal speed ALU, FP, FP Move. It can issue up to 6 uops per cycle (it has 4 dispatch ports but the 2 simple ALUs are double speed).

The Athlon has 9 functional units.

Pipelining/1

- It is the division of the work necessary to execute instructions in stages to allow more instructions in execution at the same time (at different stages)
- Previous generation architectures had 5/6 stages
- Now there are 10/20 stages, Intel called this *superpipelining* or *hyperpipelining*

Pipelining/2



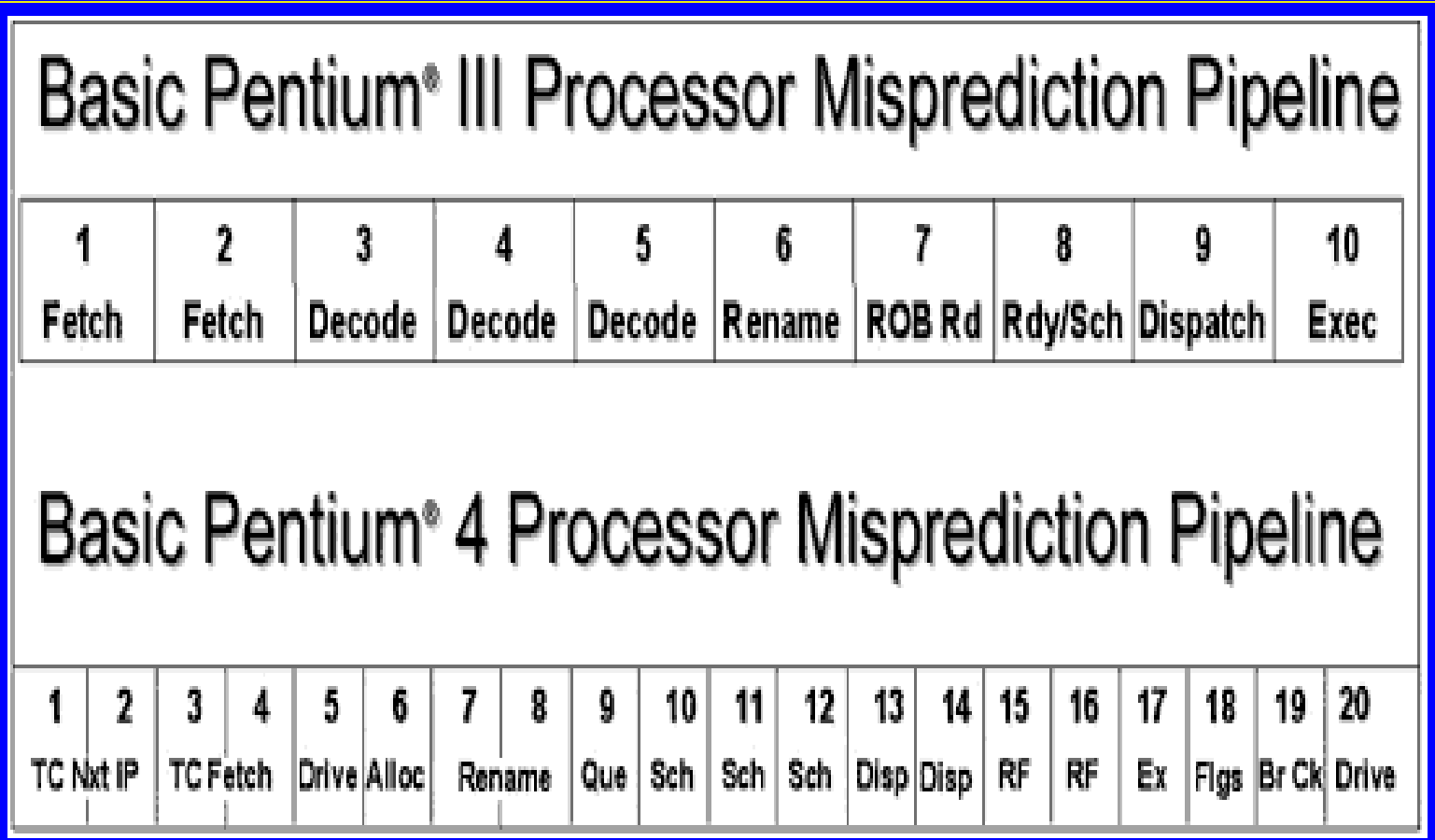
Pipelining/3

Clock cycle

	1	2	3	4	**5**	6	7	8	9	10
Instr 1	IF	ID	DA	EX	**WB**					
Instr 2		IF	ID	DA	**EX**	WB				
Instr 3			IF	ID	**DA**	EX	WB			
Instr 4				IF	**ID**	DA	EX	WB		
Instr 5					**IF**	ID	DA	EX	WB	

On the 5th cycle there are 5 instr. simultaneously executing

P3/P4 superpipelining



picture from Intel

Branch prediction Speculative execution

To avoid pipeline starvation, the most probable branch of a conditional jump (for which the condition register is not yet ready) is guessed (*branch prediction*) and the following instructions are executed (*speculative execution*) and their result is stored in hidden registers.

If later the condition turns out as predicted we say the instruction has to be *retired* and the hidden registers renamed to the real registers, otherwise we had a *misprediction* and the pipe following the branch is cleared.

x86 uarchitectures

Intel:

- P6 uarchitecture
- NetBurst

AMD:

- Thunderbird
- Palomino

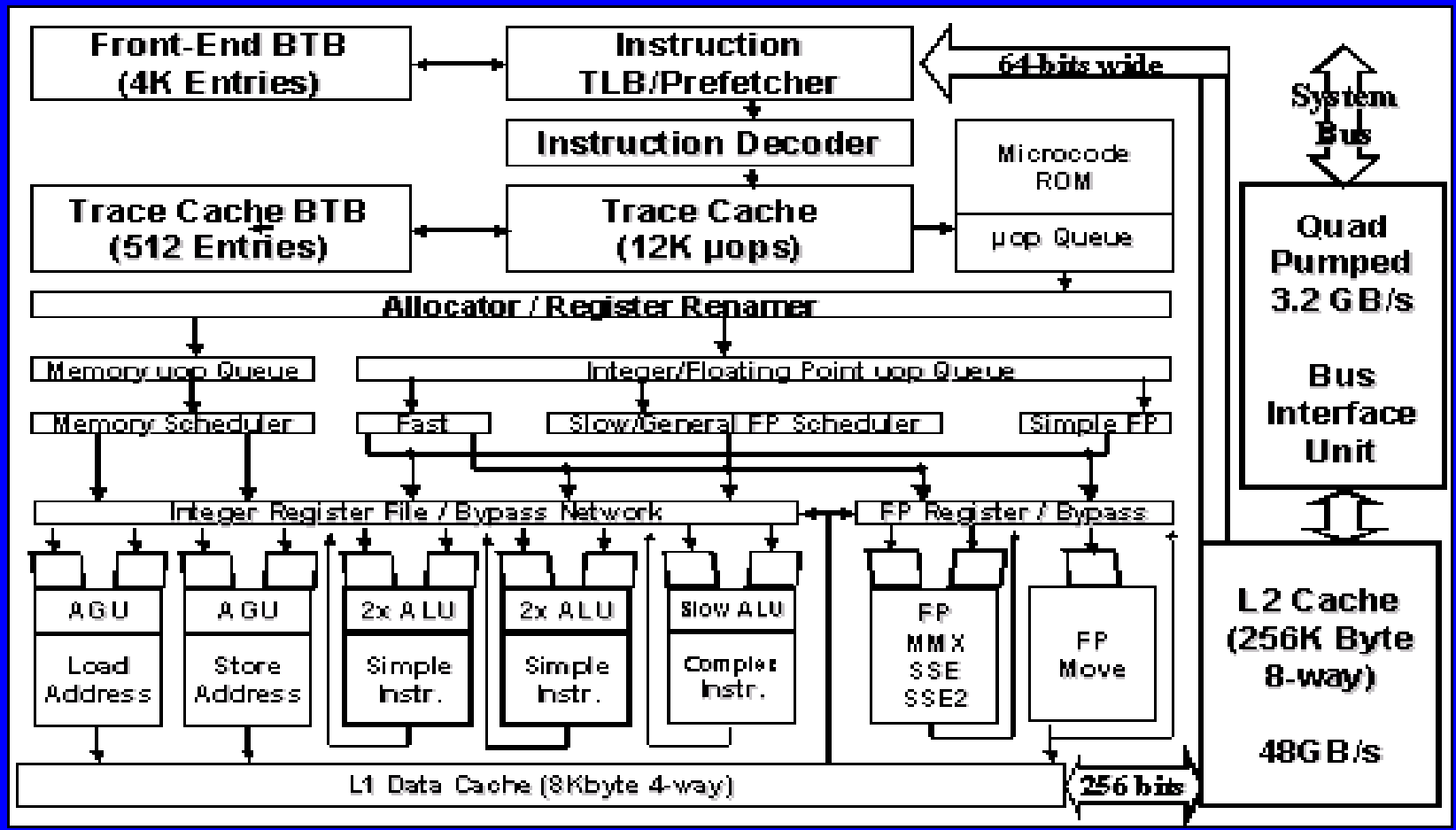
Intel x86 family

Pentium III	Pentium 4	
– Katmai		0.25 u
– Coppermine • 500 Mhz – 1.13 Ghz	– Willamette • 1.4-2.0 Ghz	0.18 u
– Tualatin • 1.13 – 1.33 Ghz	– Northwood • 2.0–2.2 Ghz	0.13 u

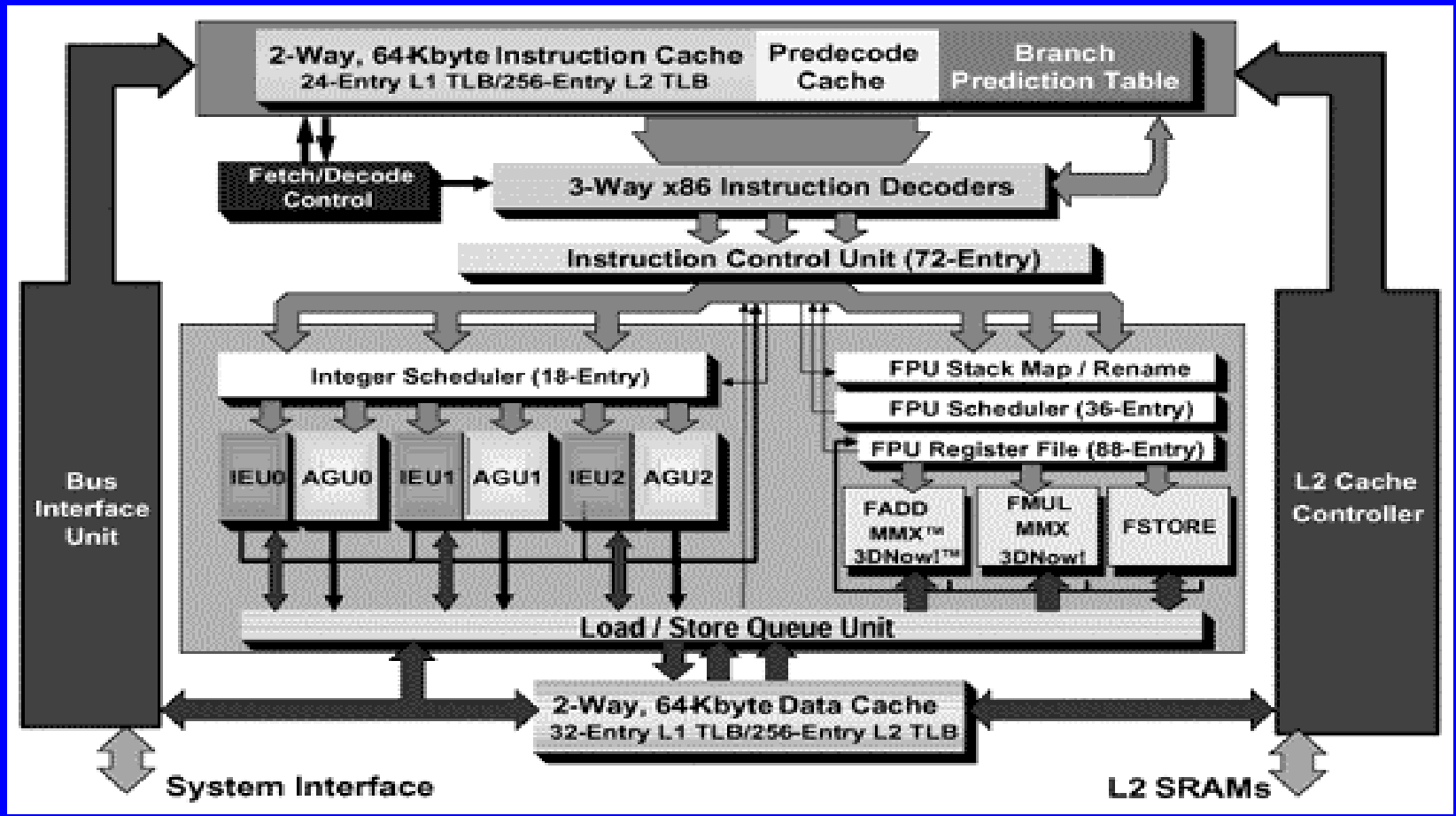
AMD Athlon family

- K7 Athlon 1999 0.25 technology w/3DNow!
- K75 Athlon 0.18 techology
- Thunderbird (Athlon 3) 0.18 u technology(on die L2 cache)
- Palomino (Athlon 4) 0.18 u Quantispeed uarchitecture (SSE support) 15 stages pipeline

Pentium 4 uarchitecture



Athlon uarchitecture



x86 Architecture extensions/1

x86 architecture was conceived in 1978. Since then it underwent many architecture extensions.

- Intel MMX (Matrix Math eXtensions):
  - introduced in 1997 supported by all current processors
- Intel SSE (Streaming SIMD Extensions):
  - introduced on the Pentium III in 1999
- Intel SSE2 (Streaming SIMD Extensions 2):
  - introduced on the Pentium 4 in Dec 2000

x86 Architecture extensions/2

- AMD 3DNow! :
  - introduced in 1998 (extends MMX)
- AMD 3DNow!+ (or 3DNow! Professional, or 3DNow! Athlon):
  - introduced with the Athlon (includes SSE)

x86 architecture extensions/3

The so called *feature set* can be obtained using the assembly instruction

```
cpuid
```

that was introduced with the Pentium and returns information about the processor in the processor's registers: processor family, model, revision, features supported, size and structure of the internal caches.

On linux the kernel uses this instrucion at startup and many of these info are available typing:

```
cat /proc/cpuinfo
```

x86 architecture extensions/4

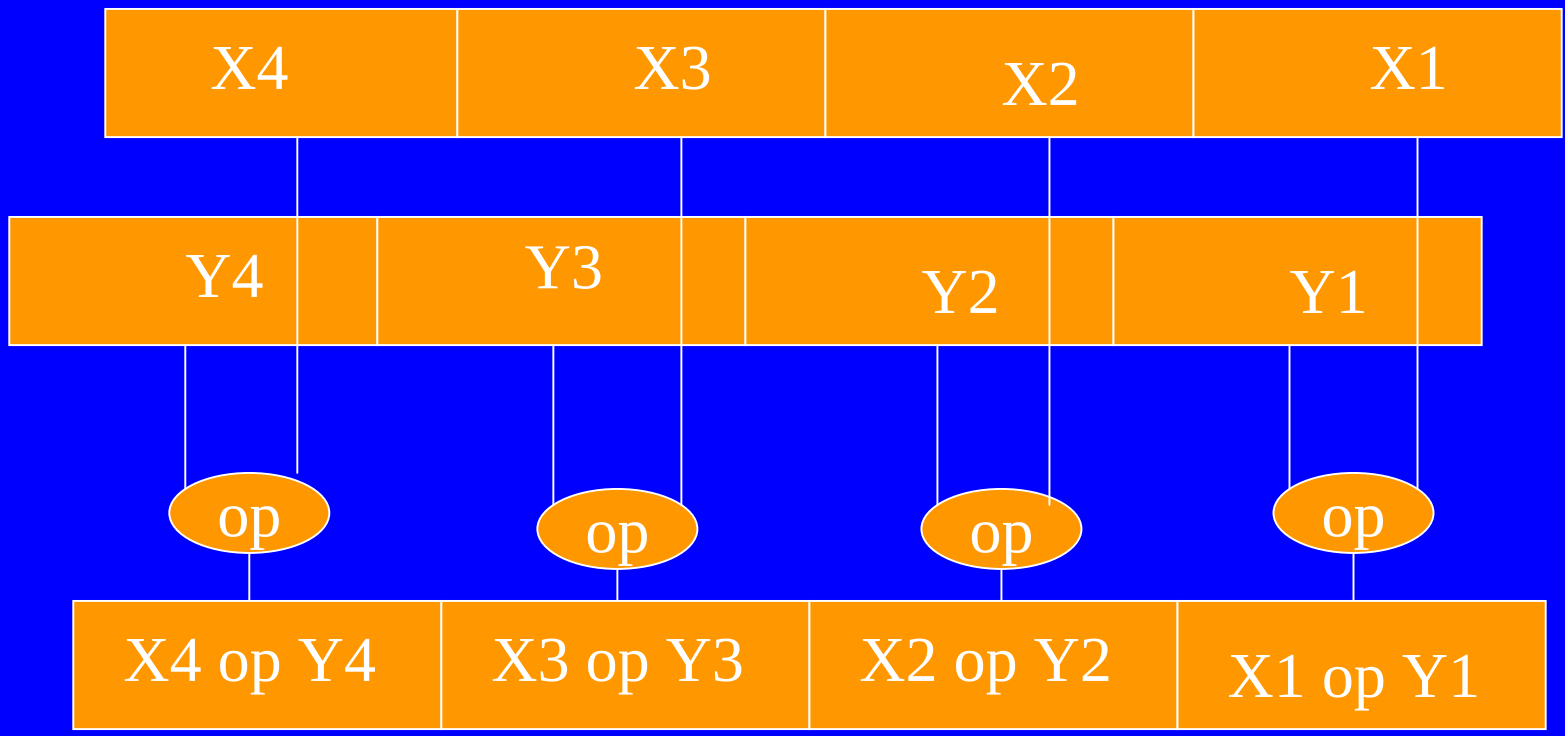
Part of a typical output can be :

```
vendor_id    : GenuineIntel
cpu family   : 6
model        : 8
model name   : Pentium III(Coppermine)
cache size   : 256 KB
flags        : fpu pae tsc mtrr pse36 mmx
               sse
```

SIMD technology

- A way to increase processor performance is to group together equal instructions on different data (Data Level Parallelism: DLW)
- SIMD (Single Instruction Multiple Data) comes from Flynn taxonomy (1966)
- Intel proposed its :
  - **SWAR** (SIMD Within A Register)

typical SIMD operation



MMX

- Adds 8 64-bits new registers :
  - MM0 – MM7
- MMX allows computations on packed bytes, word or doubleword integers contained in the MM registers (the MM registers overlap the FP registers !)
- Not useful for scientific computing

SSE

- Adds 8 128-bits registers :
  - XMM0 – XMM7
- SSE allows computations on operands that contain 4 Single Precision Floating Points either in memory or in the XMM registers
- Very limited use for scientific computing, because of lack of precision

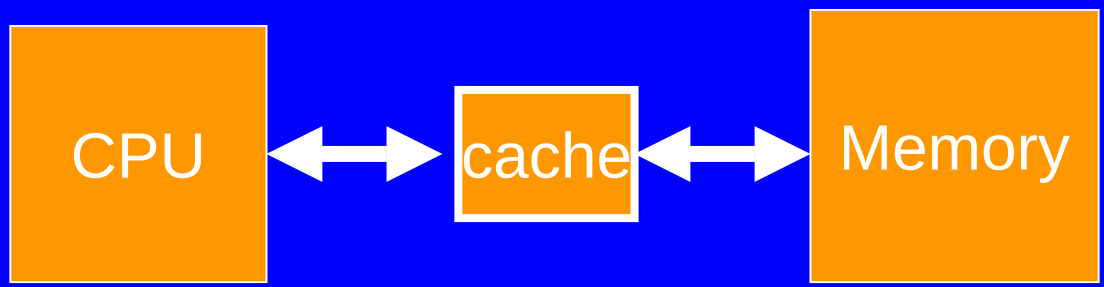
SSE2

- Works with operands either in memory or in the XMM registers
- Allows operations on packed Double Precision Floating Points or 128-bit integers
- Using a linear algebra kernel with SSE2 instructions matrix multiplication can achieve 1.8 Gflops on a P4 at 1.4 Ghz : http://hpc.sissa.it/p4/

Cache memory

Cache=a place where you can safely hide something

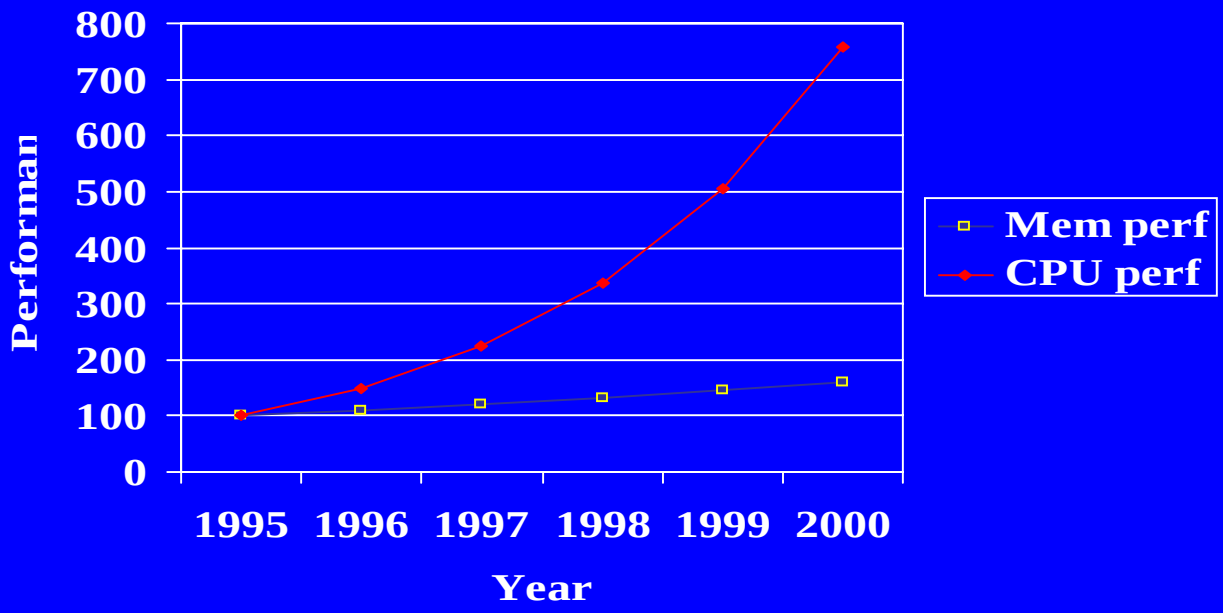
It is a high-speed memory that holds a small subset of main memory that is in frequent use.



Cache memory/1

- Processor /memory gap is the motivation :
  - 1980 no cache at all
  - 1995 2 level cache
  - 2002 ? 3 level cache ?

Cache memory/2



CPU/DRAM gap :
50 %/year

1975 cpu cycle 1 usec, SRAM access 1 usec
2001 cpu cycle 0.5 ns, DRAM access 50 ns

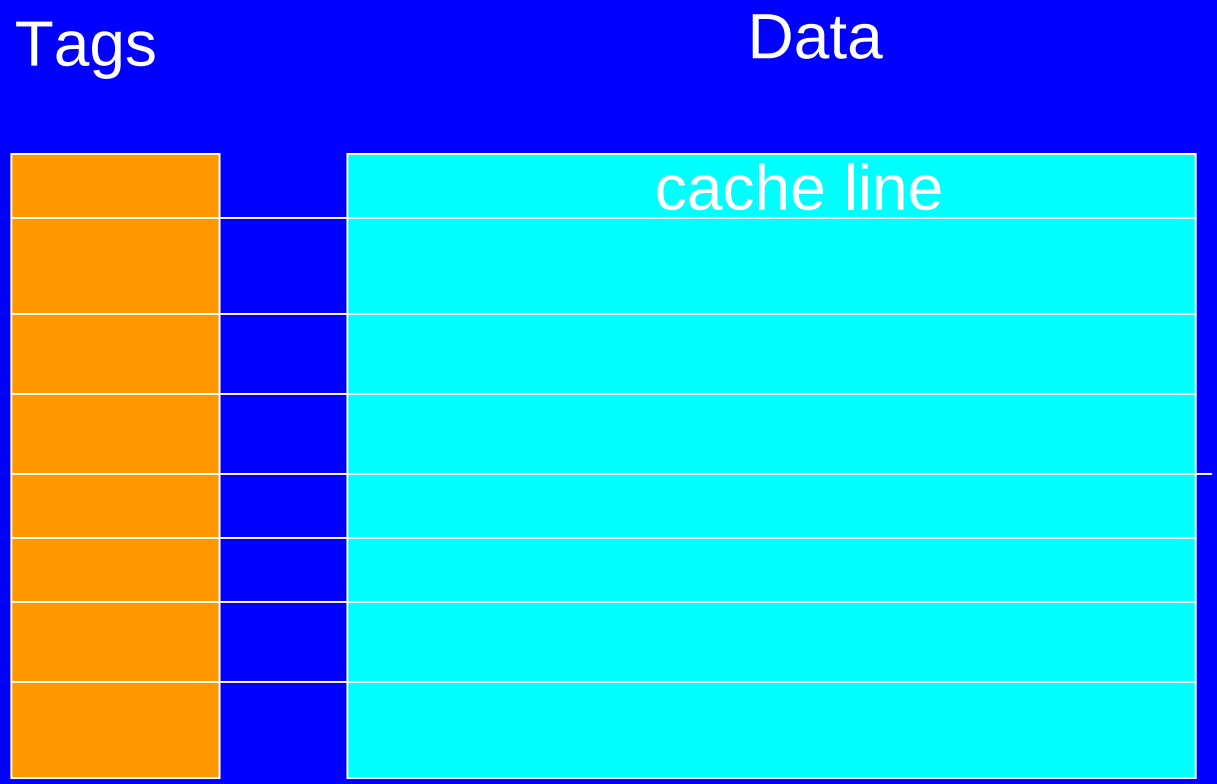
Cache memory/3

As the cache contains only part of the main memory, we need to identify which portions of the main memory are cached. This is done by *tagging* the data that is stored in the cache.

The data in the cache is organized in *lines* that are the unit of transfer from/to the CPU and the memory.

Cache memory/4

Typical line sizes are 32 (Pentium III), 64 (Athlon), 128 bytes (Pentium 4)



Cache memory/5

Block placement algorithm :

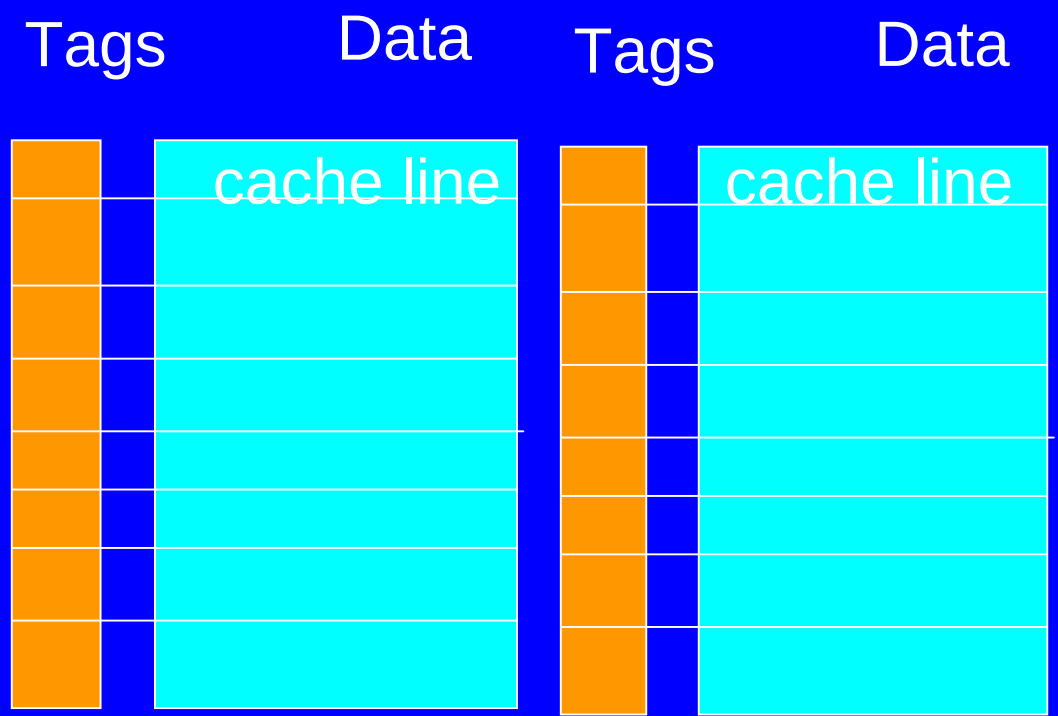
- **Direct Mapped** : a block can be placed in just one row of the cache
- **Fully Associative** : a block can be placed on any row
- **n-Way** set associative: a block can be placed on *n* places in a cache row

With direct mapping or n-way the row is determined using a simple hash function such as the least significant bits of the row address.

E.g. If the cache line is 64 bytes(bits [5:0] of address are used only to address the byte inside the row) and there are 1k rows, then bits [15:6] of the address are used to select the cache row. The tag is then the remaining most significant bits [:16] .

Cache memory/6

2-way cache : a block can be placed on any of the two positions in a row



Cache memory/7

- With fully associative or n-way caches an algorithm for block replacement needs to be implemented.
- Usually some approximation of the LRU (least recently used) algorithm is used to determine the victim line.
- With large associativity (16 or more ways) choosing at random the line is almost equally efficient

Cache memory/8

- When during a memory access we find the data in the cache we say we had a *hit*, otherwise we say we had a *miss.*
- The *hit ratio* is a very important parameter for caches in that it let us predict the **AMAT** (Average Memory Access Time)

Cache memory/9

If

- *tc* is the time to access the cache
- *tm* the time to access the data from memory
- *h* the unitary hit ratio

then:

$$t = h * tc + (1-h)*tm$$

Cache memory/10

In a *typical* case we could have :

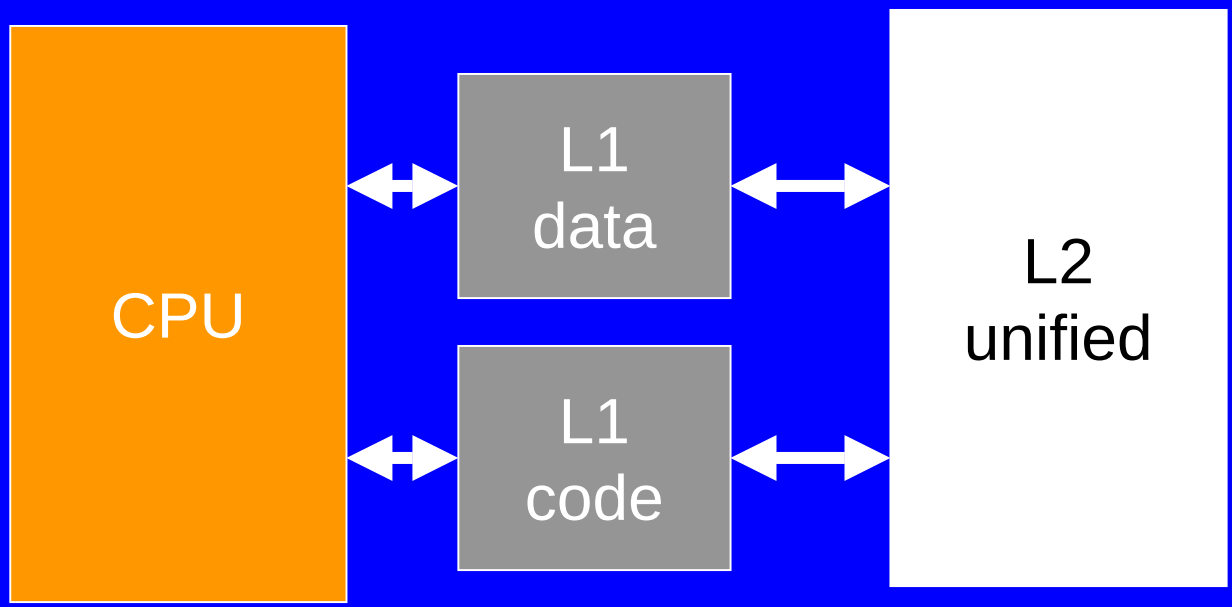
- *tc=2 ns*
- *tm=50 ns*
- *h=0.90*

then the AMAT would be :

AMAT = 0.9 * 2+0.1*50= 6.8 ns

Cache memory/11

Typical split L1/unified L2 cache organization:



Real Caches

- Pentium III : line size 32 bytes
  - split L1 – 16KB inst/16KBdata 4-way write through
  - L2 256 KB (coppermine) 8-way write back
- Pentium 4 :
  - split L1 :
    - Instruction L1 : 12 K uops 8-ways
    - Data L1 : 8 KB – 4 way/64 bytes line size/write through
  - unified L2: 256 KB 8-way /line size 128 bytes/write back (512KB on Northwood)
- Athlon : split L1 64 KB/64 KB/ line 64 bytes
  - L2 256 KB 16-ways/ 64 bytes line size

Memory performance

- stream (McCalpin) : http://www.cs.virginia.edu/stream
- memperf (T.Stricker): http://www.cs.inf.ethz.ch/CoPs/ECT

MTRR/1

Memory Type and Range Registers were introduced with the P6 uarchitecture, they should be programmed to define how the processor should behave regarding the use of the cache in different memory areas. The following types of behaviour can be defined:

UC(Uncacheable), WC(Write Combining), WT(Write through), WB(Write back), WP(write Protect)

MTRR/2

- UC=uncacheable:
  - no cache lookups,
  - reads are not performed as line reads, but as is
  - writes are posted to the write buffer and performed in order
- WC=write combining:
  - no cache lookups
  - read requests performed as is
  - a Write Combining Buffer of 32 bytes is used to buffer modifications until a different line is addressed or a serializing instruction is executed

MTRR/3

- WT=write through:
  - cache lookups are perfomed
  - reads are performed as line reads
  - writes are performed also in memory in any case (L1 cache is updated and L2 is eventually invalidated)
- WB=write back
  - cache lookups are performed
  - reads are performed as line reads
  - writes are performed on the cache line eventually reading a full line.Only when the line needs to be evicted from cache if modified, it will be written in memory
- WP=write protect
  - writes are never performed in the cache line

MTRR/4

There are 8 MTRRs for variable size areas on the P6 uarchitecture.

On Linux you can display them with:

```
cat /proc/mtrr
```

MTRR/5

A typical output of the command would be:

```
reg00: base=0x00000000 (0MB),
  size=256MB: write-back
reg01: base=0xf0000000 (3840MB),
  size=32MB:write-combining
```

the first entry is for the DRAM, the other for the graphic board framebuffer

MTRR/6

It is possible to remove and add regions using these simple commands:

```
echo ‘disable=1’ >| /proc/mtrr
echo ‘base=0xf0000000
  size=0x4000000 type=write-
  combining’ >|/proc/mtrr
```

(Suggestion: Try to use xengine while disabling and re-enabling the write-combining region of the framebuffer.)

Explicit cache control/1

Caches were introduced as an h/w only optimization and as such destined to be completely hidden to the programmers.

This is no more true, and all recent architectures have introduced some instructions for explicit cache control by the application programmer. On the x86 this was done togheter with the introduction of the SSE instructions.

Explicit cache control/2

On the Intel x86 architecture the following instructions provide explicit cache control :

- `prefetch` : these instructions load a cache line before the data is actually needed, to hide latency
- `non temporal stores:` to move data from registers to memory without writing it in the caches, when it is known data will not be re-used
- `fence:` to be sure order between prior/following memory operation is respected
- `flush` : to write back a cache line, when it is known it will not be re-used

Performance and timestamp counters/1

These counters were initially introduced on the Pentium but documented only on the PPro. They are supported also on the Athlon.

Their aim was fine tuning and profiling of the applications. They were adopted after many other processors had shown their usefulness.

Performance and timestamp counters/2

Timestamp counter (TSC):

- it is a 64 bit counter
- counts the clock cycles (processor cycles: now up to 2 Ghz=0.5ns) since reset or since programmer zeroed
- when it reaches a count of all 1s, it wraps around without generating any interrupt
- it is an x86 extension that is reported by the cpuid instruction, as such can be found on /proc/cpuinfo
- Intel assured that even on future processor it will never wrap around in less than 10 years

Performance and timestamp counters/3

- the TSC can be read with the **RDTSC** (Read TSC) instruction or the **RDMSR** (Read ModelSpecificRegister 0x10) instruction
- it is possible to zero the lower 32 bits of the TSC using the **WRMSR** (Write MSR 0x10) instruction (the upper 32 bits will be zeroed automagically by the h/w
- the RDTSC is not a serializing instruction as such does’nt avoid Out of Order execution. This can be obtained using the cpuid instruction

Performance and timestamp counters/4

The TSC is used by Linux during the startup phase to determine the clock frequency:

- the PIT (Programmable Interval Timer) is programmed to generate a 50 millisecond interval
- the elapsed clock ticks are computed reading the TSC before and after the interval

The TSC can be read by any user or only by the kernel according to a cpu flag in the CR4 register that can be set e.g. by a module.

Performance and timestamp counters/5

On the P6 uarchitecture there are 4 registers used for performance monitoring:

- 2 event select registers: PerfEvtSel0, PerfEvtSel1
- 2 performance counters 40 bits wide : PerfCtr0, PerfCtr1

You have to enable and specify the events to monitor setting the event select registers.

These registers can be accessed using the RDMSR and WRMSR instructions at kernel level (MSR 0x186,0x187, 0x193 0x194).

The performance counters can also be accessed using the RDPMC (Read Performance Monitoring Counters) at any privilege level if allowed by the CR4 flag.

Performance and timestamp counters/6

The Performance Monitoring mechanism has been vastly changed and expanded on P4 and Xeons. This new feature is called : PEBS (Precise Event-based Sampling).

There are 45 event selection control (ESCR) registers , 18 performance monitor counters and 18 counter configuration control (CCCR) MSR registers.

Now counters can produce an interrupt on overflow/underflow.

You can count *bogus* (uops non retired because of misprediction) and *nonbogus* events separately.

Performance and timestamp counters/7

A useful performance counters library to instrument the code and a tool called *rabbit* to monitor programs not instrumented with the library can be found at:

http://www.scl.ameslab.gov/Projects/Rabbit/

Another tool in perl for Pentium 4 called brink/abyss is available at

http://www.eg.brucknell.edu/~bsprunt/emon/brink_abyss.shtm

And then there is PAPI (Performance API) by Dongarra et al. at UTK:

http://icl.cs.utk.edu/projects/papi/

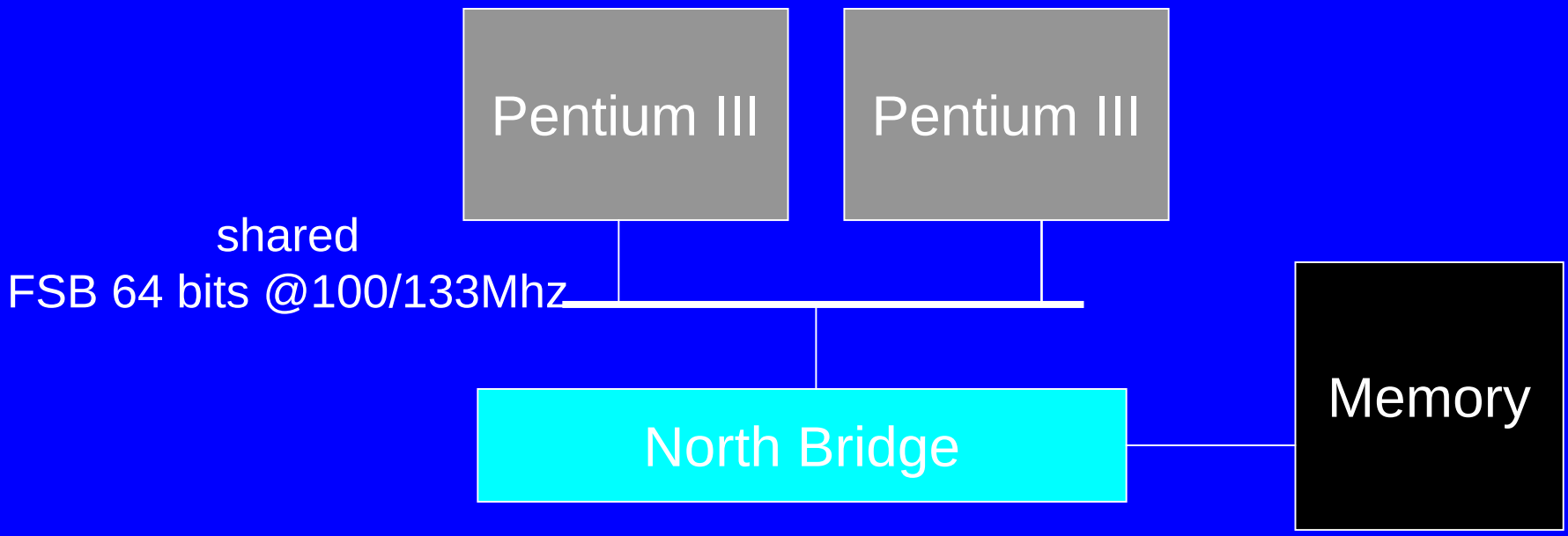
Processor bus/1

- Intel (AGTL+):
  - bus based (max 5 loads)
  - explicit in band arbitration
  - short bursts (4 data txfers)
  - 8 bytes wide(64 bits), up to 133 Mhz
- Compaq Alpha (EV6)/Athlon :
  - point to point
  - DDR double data rate (2 transfers x clock)
  - licensed by AMD for the Athlon (133 Mhz x 2)
  - source synchronous(up to 400 Mhz)
  - 8 bytes wide(64 bits)

Processor bus/2

- Intel Netburst (Pentium 4):
  - source synchronous (like EV6)
  - 8 bytes wide
  - 100 Mhz clock
  - quad data rate for data transfers (4 x 100 Mhz x 8 bytes= 3.2 GB/s, just in theory)
  - double data rate for address transfer
  - max 128 bytes(a P4 cache line)in a transaction (4 data transfer cycles)

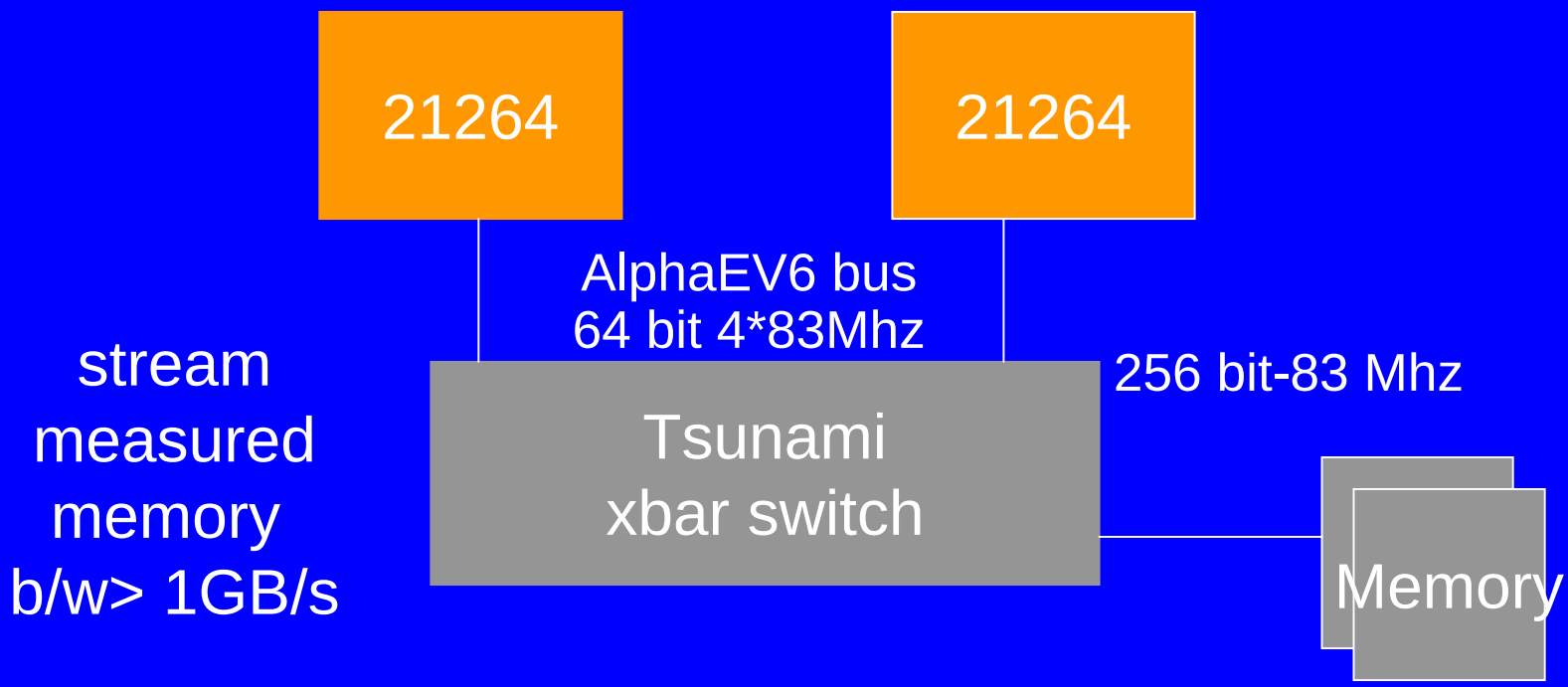
Intel IA32 node



Intel PIII/P4 processor bus

- Bus phases :
  - Arbitration: 2 or 3 clks
  - Request phase: 2 clks packet A, packet B (size)
  - Error phase: 2 clks, check parity on pkts,drive AERR
  - Snoop phase: variable length 1 ...
  - Response phase: 2 clk
  - Data phase : up to 32 bytes (4 clks, 1 cache line), 128 bytes with quad data rate on Pentium 4
- 13 clks to txfer 32 bytes, or 128 bytes on P4

Alpha node


21264
21264
AlphaEV6 bus
64 bit 4*83Mhz
stream
measured
memory
b/w> 1GB/s
Tsunami
xbar switch
256 bit-83 Mhz
Memory

Alpha/Athlon EV6 bus

- 3 high speed channels :
  - Unidirectional processor request channel
  - Unidirectional snoop channel
  - 72-bit data channel (ECC)
- source synchronous
- up to 400 Mhz (4 x 100 Mhz: quad pumped, Athlon :133Mhz x 2 ddr)

Pentium 4 (Willamette)

- 1.4-2.2 Ghz with 256(512 Northwood)KB L2 cache (SKT 423)/(SKT478)
- Processor bus at 100 Mhz but Quad pumped (2x address rate/4x data rate)
- With PC800 RDRAM at 1.4 Ghz stream gives ~ 1.5 GB/s memory bandwidth
- L2 Cache line is 128 bytes (1 bus transaction: 4x4x8)

SMPs

Symmetrical Multi Processing denotes a multiprocessing architecture in which there is no master CPU, but all CPUs co-operate.

- processor bus arbitration
- cache coherency/consistency
- atomic RMW operations
- mp interrupts

Intel MP
Processor bus arbitration

- As we have seen there is a special phase for each bus transaction devoted to arbitration (it takes 2 or 3 cycles)
- At startup each processor is assigned a cluster number from 0 to 3 and a processor number from 0 to 3 (the Intel MP specification covers up to 16 processors)
- In the arbitration phase each processor knows who had the bus during last transaction (the *rotating ID*) and who is requesting it. Then each processor knows who will own the current transaction because they will gain the bus according to the fixed sequence 0-1-2-3.

Cache coherency

Coherence: all processor should see the same data.

This is a problem because each processor can have its own copy of the data in its cache.

There are essentially 2 ways to assure cache coherency:

- *directory based :* there is a central directory that keeps track of the shared regions (ccNUMA: Sun Enterprise, SGI)
- *snooping :* all the caches monitor the bus to determine if they have a copy of the data requested (UMA: Intel MP)

Cache consistency

- *Consistency:* mantain the order in which writes are executed

Writes are serialized by the processor bus.

Snooping

Two protocols can be used by caches to snoop the bus :

- *write-invalidate:* when a cache hears on the bus a write request for one of his lines from another bus agent then it invalidates its copy (Intel MP)
- *write-update:* when a cache hears on the bus a write request for one of his lines from another agent it reloads the line

Intel MP snooping

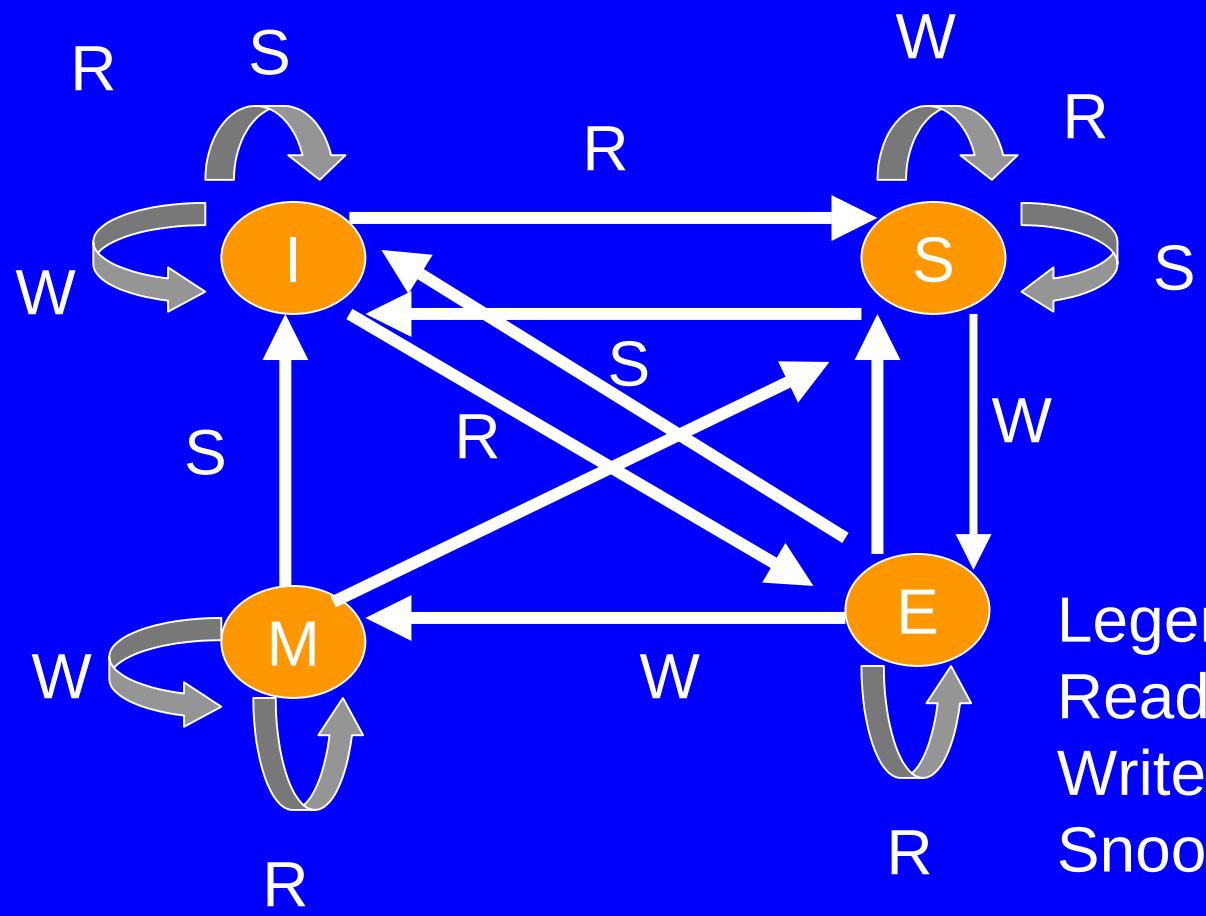
- If a memory transaction (memory read and invalidate/read code/read data/write cache line/write) was not cancelled by an error, then the error phase of the bus is followed by the snoop phase(2 or more cycles)
- In this phase the caches will check if they have a copy of the line line
- if it is a read and a processor has a modified copy then it will supply its copy that is also written to memory
- if it is a write and a processor has a modified copy then the memory will store first the modified line and then will merge the write data

MESI protocol

Each cache line can be in 1 of the 4 states:

- Invalid (I): the line is not valid and should not be used
- Exclusive(E): the line is valid, is the same as main memory and no other processor has a copy of it, it can be read and written in cache w/o problems
- Shared(S): the line is valid, the same as memory, one or more other processors have a copy of it, it can be read from memory, it should be written-through (even if declared write back!)
- Modified(M): the line is valid, has been updated by the local processor, no other cache has a copy, it can be read and written in cache

MESI states



Legenda:
Read access
Write access
Snoop

L2/L1 coherence

- An easy way to keep coherent the 2 levels of caches is to require inclusion (L1 subset of L2)
- Otherwise each cache can perform its snooping

Atomic Read Modify Write

The x86 instruction set has the possibility to prefix some instructions with a LOCK prefix:

- bit test and modify
- exchange
- increment, decrement, not,add,sub,and,or

These will cause the processor to assert the LOCK# bus signal for the duration of the read and write.

The processor automatically asserts the LOCK# signal during execution of an XCHG , during a task switch, while reading a segment descriptor

Intel MP interrupts

Intel has introduced an I/O APIC (*Advanced Programmable Interrupt controller*) which replaces the old 8259A.

- each processor of an SMP has its own integrated *local APIC*
- an *Interrupt Controller Communication* (ICC) bus connects an external *I/O APIC* (front-end) to these local APICs (on the Pentium 4 this communication happens on the processor bus)
- externel IRQ lines are connected to the I/O APIC that acts as a router
- the I/O APIC can dispatch interrupts to a fixed processor or to the one executing lowest priority activites (the priority table has to be updated by the kernel at each context switch)

PCI Bus

Standard PCI in use today is 32 bits at 33 Mhz, just sufficient for 1 Gb/s

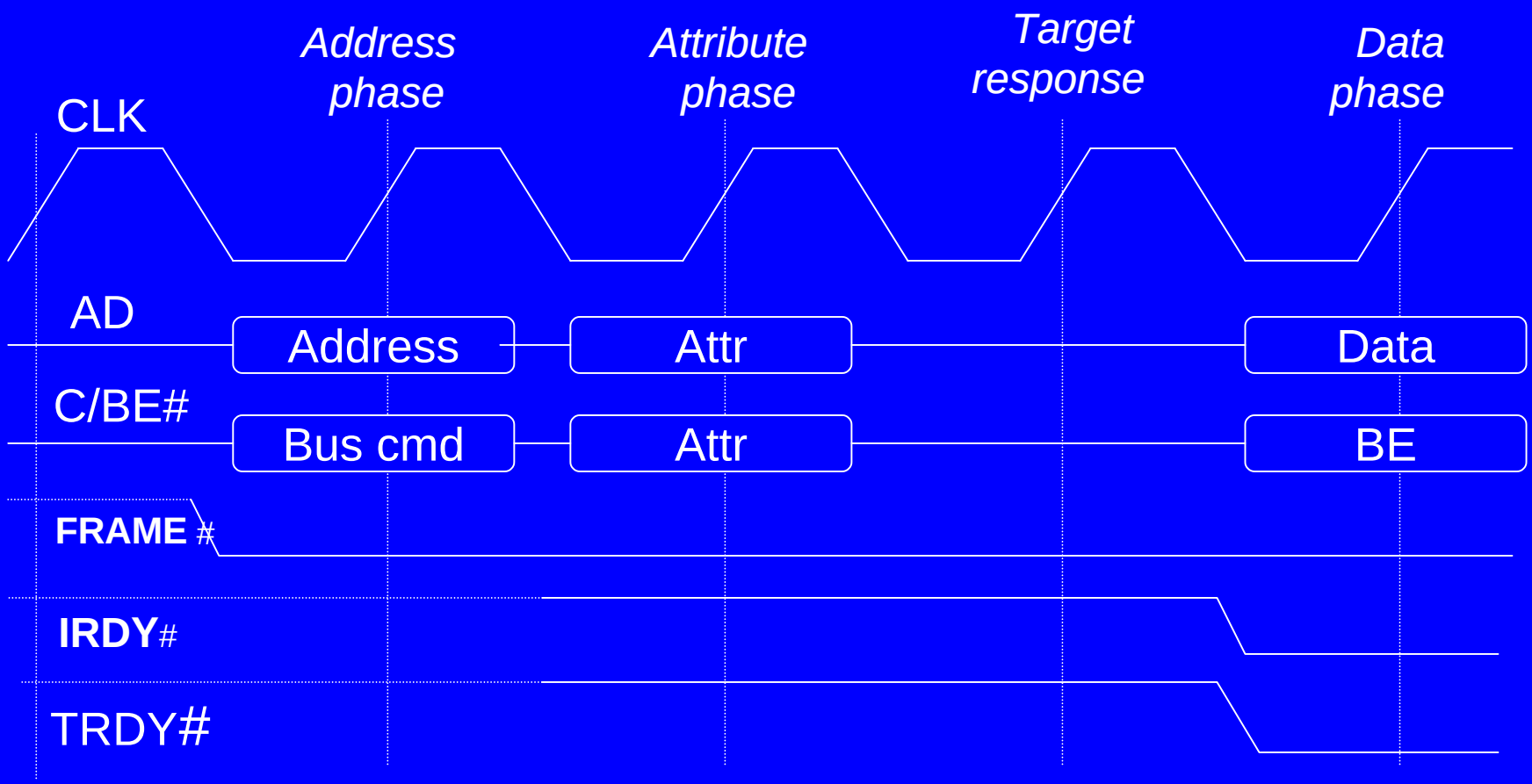
- PCI32/33 4 bytes@33Mhz=132MBytes/s (on i440BX,...)
- PCI64/33 8 bytes@33Mhz=264Mbytes/s
- PCI64/66 8 bytes@66Mhz=528Mbytes/s (on i840)
- PCI-X 8 bytes@133Mhz=1056Mbytes/s

PCI-X will implement split transactions

PCI efficiency

- Multimaster bus but arbitration is performed out of band
- Multiplexed but in burst mode (implicit addressing) only start address is txmitted
- Fairness guaranteed by MLT (Maximum Latency Timer)
- 3 / 4 cycles overhead on 64 data txfers < 5 %
- on Linux use : *lspci –vvv* to look at the setup of the boards

PCI 2.2/X timing diagram



common chipsets PCI performance

Chipset	Read MB/s	Write MB/s
Serverworks Serverset III LE	455	512
Titan (Alpha)	407	488
Intel 460GX (Itanium)	372	399
Serverset III HE	315	372
Serverworks champion II HE	309	372
Serverworks HE	303	372
AMD760MPX	315	328
Intel 860	227	315
Tsunami (alpha)	205	248

Memory buses

- SDRAM 8 bytes wide (64 bits): these memories are pipelined DRAM. Industry found for them much more appealing to indicate clock frequency than access times. Number of clock cycles to wait for access is written in a small rom on the module(SPD)
  - PC-100 PC-133
  - DDR PC-200, DDR 266 QDR on the horizon
- RDRAM 2 bytes wide(16 bits) these memories have a completely new signaling technology. Their bus should be terminated at both ends (coninuity module required if slot not used!)
  - RDRAM 600/800/1066 double data rate at 300,400,533 Mhz

Interconnects /1

Today we speak of *interconnection networks* (or *interconnects*) referring to processor-memory, I/O controllers-I/O devices, processor-processor, computer-computer communications. Up to the beginning of the ’90 many of these applications were using a very simple interconnection: the *multidrop bus*.

Now all high performance communications are performed by point-to-point interconnection networks. This depends on the demand for higher performance and the impossibility to scale for the bus.

bandwidth : speed in Mb/s of the network including overhead bits

txmission time: time to pass through the net (size in bits with overhead)/bandwidth

time of flight: time for a bit to traverse the net sender to receiver

sender overhead: time to prepare the message

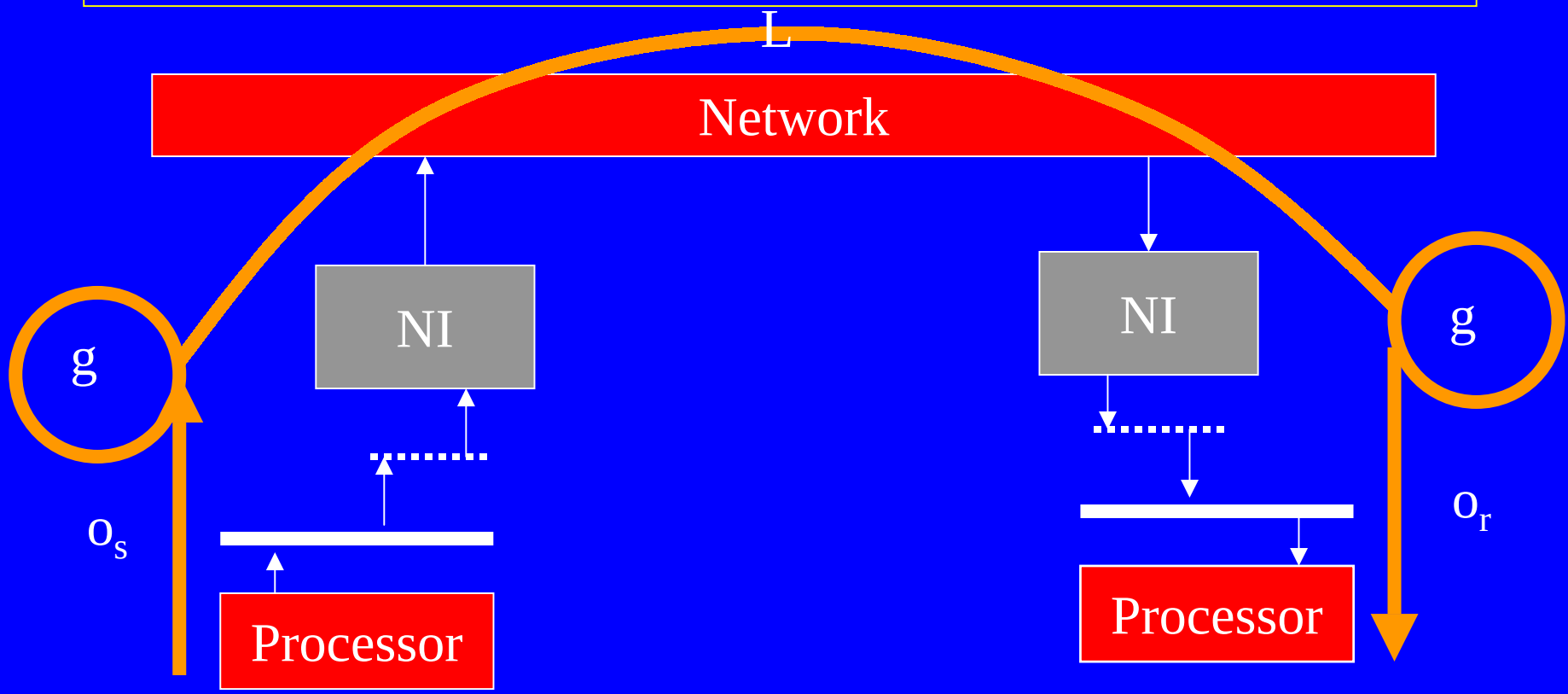
receiver overhead: time to get the message from the network

LogP metrics (Culler)

This metric was introduced to characterize a distributed system with its most important parameters,a bit outdated, but still useful.(e.g..does'nt take into account pipelining)

- L = Latency: time data is on flight between the 2 nodes
- o = overhead: time during which the processor is engaged in sending or receiving
- g = gap : minimum time interval between consecutive message txmissions(or receptions)
- P = # of Processors

LogP diagram



time=o_s+L+o_r

Interconnects /2

- single mode fiber (sonet 10 gb/s for some km)
- multimode fiber (1 gb/s 300 m: GigE)
- coaxial cable (800 mb/s 1 km: CATV)
- twisted pair (1 gb/s 100 m: GigE)

Interconnects /3

- shared / switched media:
  - bus (shared communication):
    - coaxial
    - pcb trace
    - backplane
  - switch (1-1 connection):
    - point to point

Interconnects /4

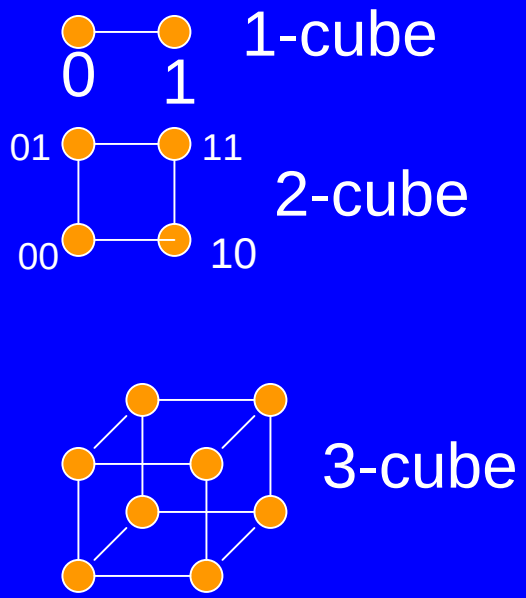
Network Topology:

- now not so important as it has been considered in the past
- clearly the best network would be the complete graph: every node connected directly with all other nodes, but this requires n(n-1)/2 connections
- issues: routing distance, diameter, avg distance
- the major issue is the number of wires, delay and bandwidth
  - **ring** : nodes are numbered 0:n-1, each node has 2 connections one with the preceding and one with the following node (modulo n)
  - **crossbar**
  - **hypercube**
  - **mesh/grid**
  - **star:** there is a central switch to which all nodes are connected
  - **torus**
  - **omega**

Interconnects /5

Hypercube:

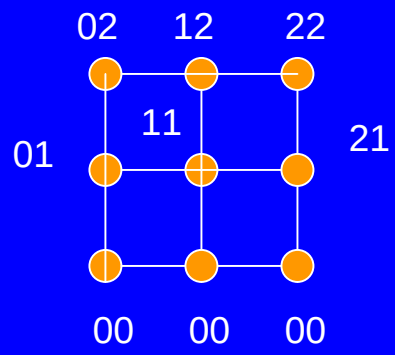
- A k-cube has 2**k nodes, each node is labelled by a k-dim binary coordinate
- 2 nodes differing in only 1 dim are connected
- there are at most k hops between 2 nodes, and 2**k * k wires



Interconnects /6

Mesh:

- an extension of hypercube
- nodes are given a k-dim coordinate (in the 0:N-1 range)
- nodes that differ by 1 in only 1 coordinate are connected
- a k-dim mesh has N**k nodes
- at most there are kN hops between nodes and wires are ~ kN**k

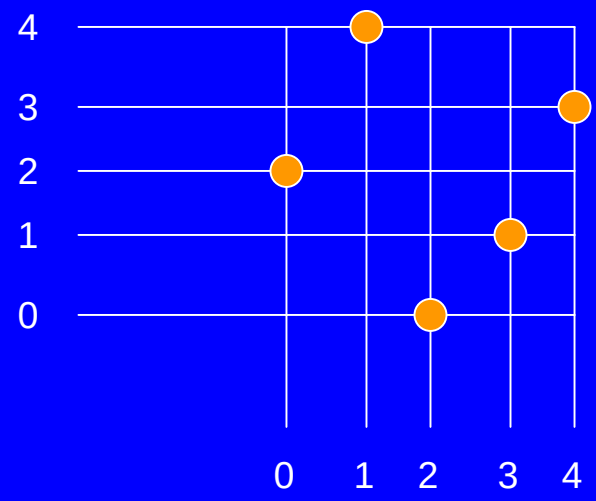


2-dim 3x3 mesh

Interconnects /7

Crossbar (xbar):

- typical telephone network technology
- organized by rows and columns (NxN)
- requires N**2 switches
- any permutation w/o blocking blocking
- in the picture the i-row is the sender and the i-col is the receiver of node i



5x5 xbar switch

Interconnects /8

- Bisection width :

	bisection	ports/switch
bus	1	
ring	2	3
star	32	64
grid/mesh	8	5
2-d torus	16	5
hypercube	32	7
fully connected	1024	64

Interconnects /9

- Connection/connectionless:
  - circuit switched: the telephone network is a typical example of circuit switching, ATM
  - packet switched: ethernet, X25
- routing
  - source based routing (SBR): myrinet
  - virtual circuit: ATM, phone net
  - destination based: multicast routing

Interconnects /10

- switch mechanism (buffer management):
  - store and forward
    - each msg is received completely by a switch before being forwarded to the next
    - pros: easy to design because there is no handling of partial msgs
    - cons: long msg latency, requires large buffer
  - cut-through (virtual cut-through)
    - msg forwarded to the next node as soon as its header arrives
    - if the next node cannot receive the msg then the full msg needs to be stored
  - wormhole routing
    - the same as cut-trough except that the msg can be buffered togheter by multiple successive switches
    - the msg is like a worm crawling through a worm-hole

Interconnects /11

- congestion control
  - packet discarding: typically used by switches, routers on IP stack
  - flow control
    - window /credit based: MAN, myrinet gm protocol
    - start/stop: xon/xoff serial protocol

Bisection /1

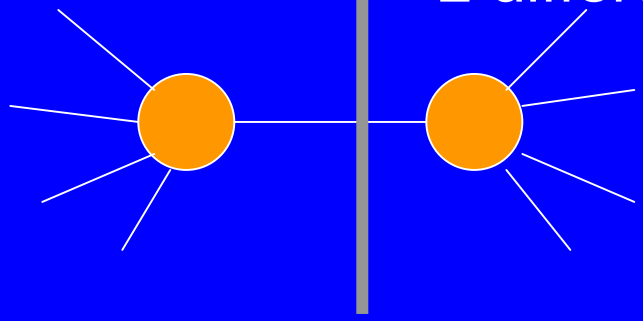
Bisection width : given an N nodes net, divided the nodes in two sets of N/2 nodes, the number of links that go from one set to the other.

An important parameter is the *minimum bisection width*: the minimum number of links whatever cut you use to bisect the nodes.

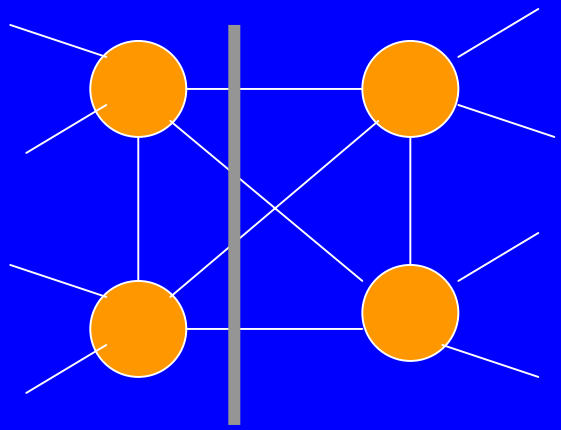
An upper bound on the *minimum bisection* is N/2 because whatever the topology would be you can always find a cut across half of the node links. If a network has minimum bisection of N/2 we say it has *full bisection*.

Bisection /2

2 different 8 nodes nets



Bisection cut:
1 link



Bisection cut:
4 links

Bisection /3

It can be difficult to find the min bisection.

For full bisection networks, it can be shown that if a network is *re-arrangeable* (it can route any permutation w/o blocking) then it has full bisection.

From	0	1	2	3	4	5	6
To	3	0	4	5	2	6	1

NIC Interconnection point

(from D.Culler)

	Controller	Special uproc	General uproc
Register	TMC CM-5		
Memory	T3E annex	Meiko CS-2	Intel Paragon
Graphics Bus	HP Medusa		
I/O Bus	Many ether cards	Myrinet, 3ComGbe	SP2, Fore ATM cards

Ethernet history

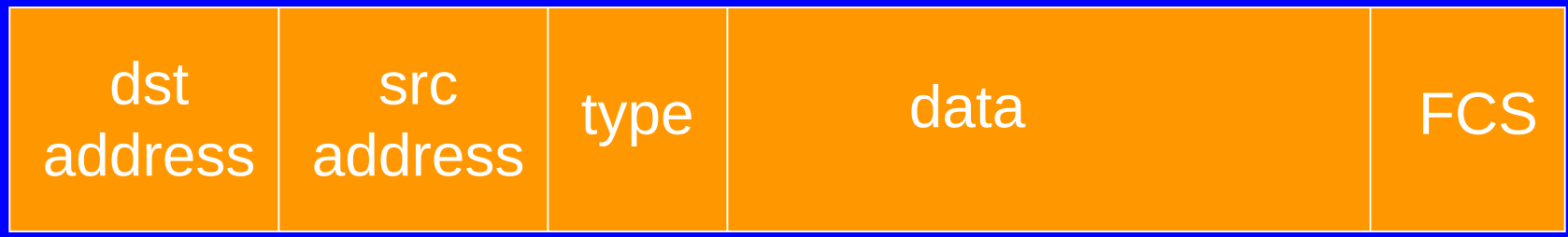
- 1976 Metcalfe invented a 1 Mb/s ether
- 1980 Ethernet DIX (Digital, Intel, Xerox) standard
- 1989 Synoptics invented twisted pair Ethernet
- 1995 Fast Ethernet
- 1998 Gigabit Ethernet
- 200x 10 Gigabit Ethernet

Ethernet

- 10 Mb/s
- Fast Ethernet
- Gigabit Ethernet

Ethernet Frames

- 6 bytes dst address
- 6 bytes src address :3 bytes vendor codes, 3 bytes serial #
- 2 bytes ethernet type: ip, ...
- data from 46 up to 1500 bytes
- 4 bytes FCS (checksum)



VLAN

Virtual LAN :

- IEEE 802.1Q extension to the ethernet std
- frames can be tagged with 4 more bytes (so that now an ethernet frame can be up to 1522 bytes):
  - TPID tagged protocol identifier, 2 bytes with value 0x8100
  - TCI Tag control information: specifies priority and Virtual LAN (12 bits) this frame belongs
  - remaining bytes with standard content

Hubs

- Repeaters: layer 1
  - like a bus they just repeat all the data across all the connections (not very useful for clusters)
- Switches (bridges) layer 2
  - they filter traffic and repeat only necessary traffic on connections (very cheap switches can easily switch at full speed many Fast Ethernet connections)
- Routers : layer 3

Ethernet flow/control

- With large switches it is necessary to be able to limit the flow of packets to avoid packet discarding following an overflow
- Vendors had tried to implement non standard mechanism to overcome the problem
- One of this mechanism that could be used for half/duplex links is sending carrier bursts to simulate a busy channel
- This mechanism does'nt apply to full/duplex links
- MAC control protocol : to support flow control on f/d links
  - a new ethernet type = 0x8808, for frames of 46 bytes (min length)
  - first 2 bytes are the opcode other are the parameters
  - opcode = 0x0001 PAUSE stop txmitting
  - dst address = 01-80-c2-00-00-01 (an address not forwarded by bridges)
  - following 2 bytes represent the number of 512 bit times to stop txmission

Auto Negotiation

- On twisted pairs each station transmits a series of pulses (quite different from data) to signal link active. These pulses are called **Normal Link Pulses** (NLP)
- A set of **Fast Link Pulses** (FLP) is used to code station capabilities.These FLPs bits code the set of station capabilities
- There are some special repeaters that connect 10mb/s links together and 100mb/s together and then the two sets to ports of a mixed speed bridge
- **Parallel detection**: when autonegotiation is not implemented, this mechanism tries to discover speed used from NLP pulses (**will not detect duplex links**).

GigE

Peculiarities :

- Flow control necessary on full duplex
- 1000base-T uses all 4-pairs of cat 5 cable in both directions: 250 Mhz*4 , with echo cancellation(10 and 100base-T used only 2 pairs)
- 1000base-SX on multimode fiber uses a laser (usually a VCSEL). Some multimode fibers have a discontinuity in the refraction index for some microns just in the middle of the fiber. This is not important with a LED launch where the light fills the core but is essential with a laser launch. An *offset patch cord* (a small length of single mode fiber that moves away from the center the spot) has been proposed as solution.

Network (Physical Layer)

Current technology is at ~1/2 Gb/s(GbE,Myrinet), is there room for improvement?

- Ethernet 2.5 Gb/s ..10 Gb/s
- Infiniband 2.5Gb/s – 30 Gb/s
- SONET OC192 10 Gb/s
- GSN(Hippi) 6.4 Gb/s

A lot of the improvements in the optical arena are coming from the use in the last years of the low cost VCSELs(Vertical Cavity Surface Emitting Laser)

Network (Physical Layer)

Two technologies have provided room for vast improvements at the net physical layer in the last decade :

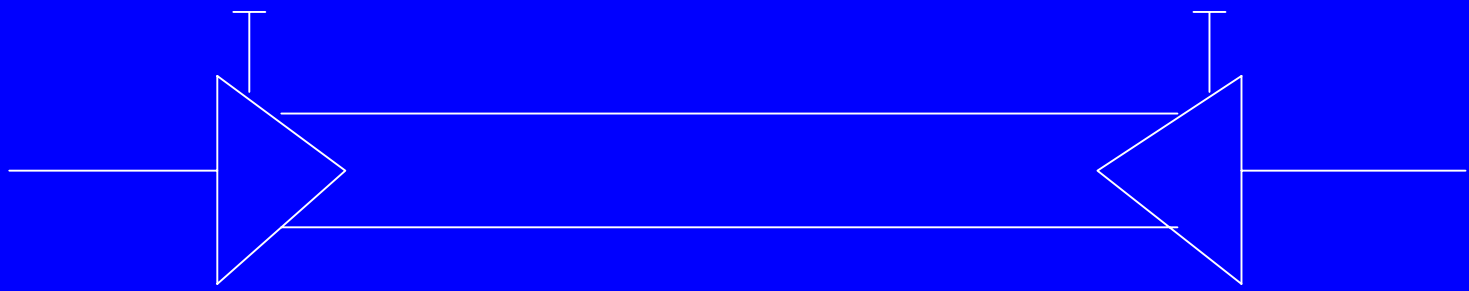
- LVDS(Low voltage differential signaling) on copper
- VCSEL (Vertical Cavity Surface Emitting Lasers) on fibers

LVDS/1

- Low Voltage Differential Signaling (ANSI/TIA/EIA 644-1995)
- Defines only the electrical characteristics of drivers and receivers
- Transmission media can be copper cable or PCB traces

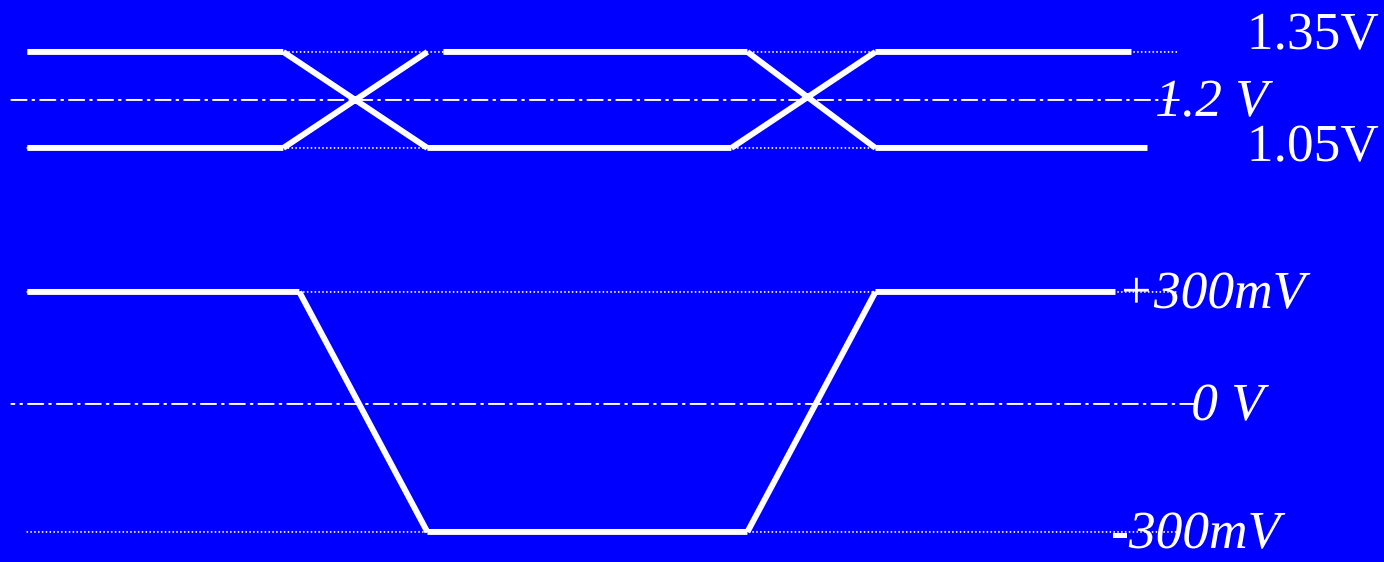
LVDS/2

- Differential :
  - Instead of measuring the voltage Vref+U between a signal line and a common GROUND, a pair is used and Vref+U and Vref–U are transmitted on the 2 wires
  - In this way the transmission is immune to Common Mode Noise (the electrical noise induced in the same way on the 2 wires: EMI,..)



LVDS/3

- Low Voltage:
  - voltage swing is just 300 mV, with a driver offset of +1.2V
  - receivers are able to detect signals as low as 20 mV,in the 0 to 2.4 V (supporting +/- 1 V of noise)



LVDS/4

- It consumes only 1 mW(330mV swing constant current): GTL would consume 40 mW(1V swing) and TTL much more
- consumer chips for connecting displays (OpenLDI DS90C031) are already txmitting 6 Gb/s
- The low slew rate (300mV in 333 ps is only 0.9V/ns) minimize xtalk and distortion

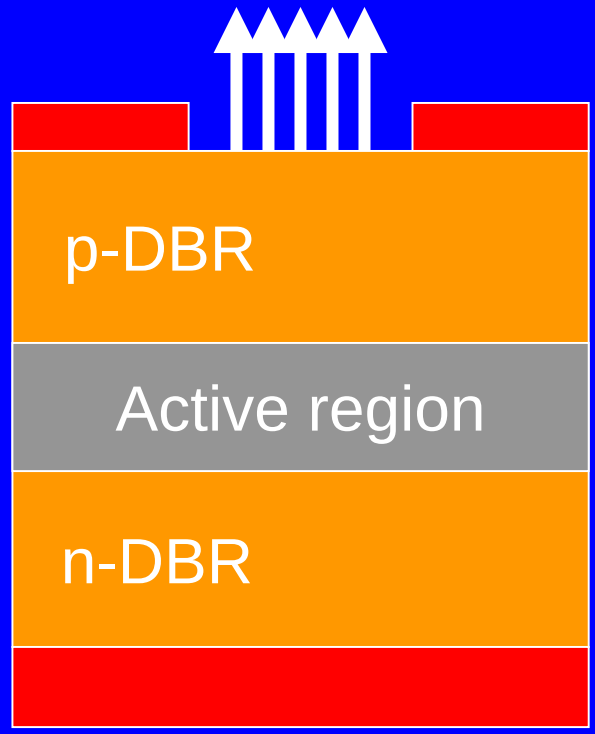
VCSEL/1

VCSELs: Vertical Cavity Surface Emitting Lasers:

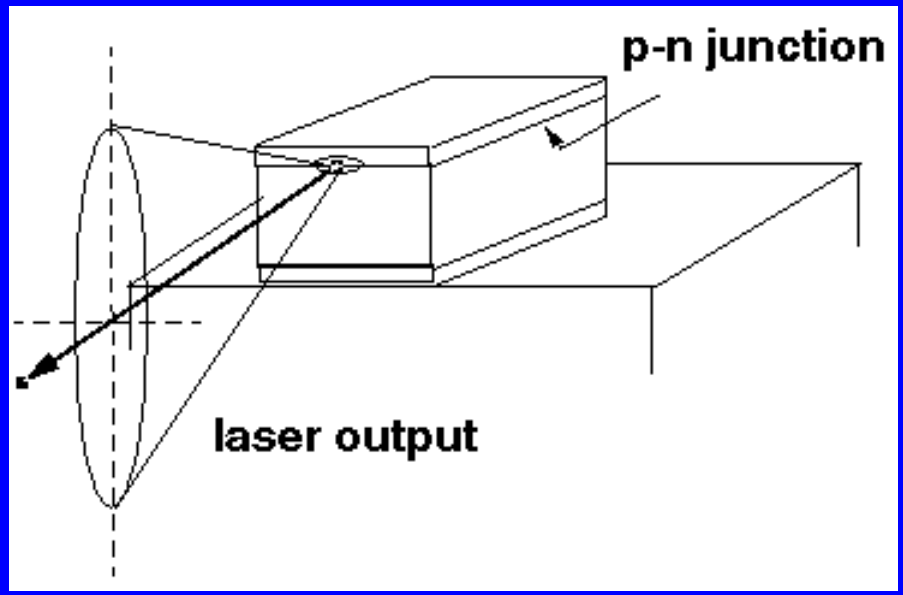
- the laser cavity is vertical to the semiconductor wafer
- the light comes out from the surface of the wafer

Distributed Bragg Reflectors(DBR):

- 20/30 pairs of semiconductor layers

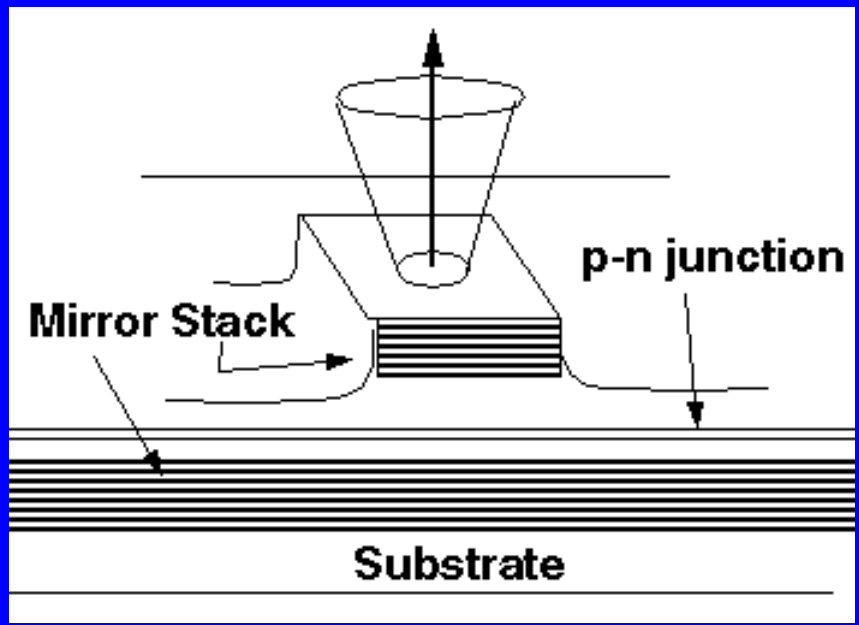


VCSEL/2-EEL (Edge Emitting)



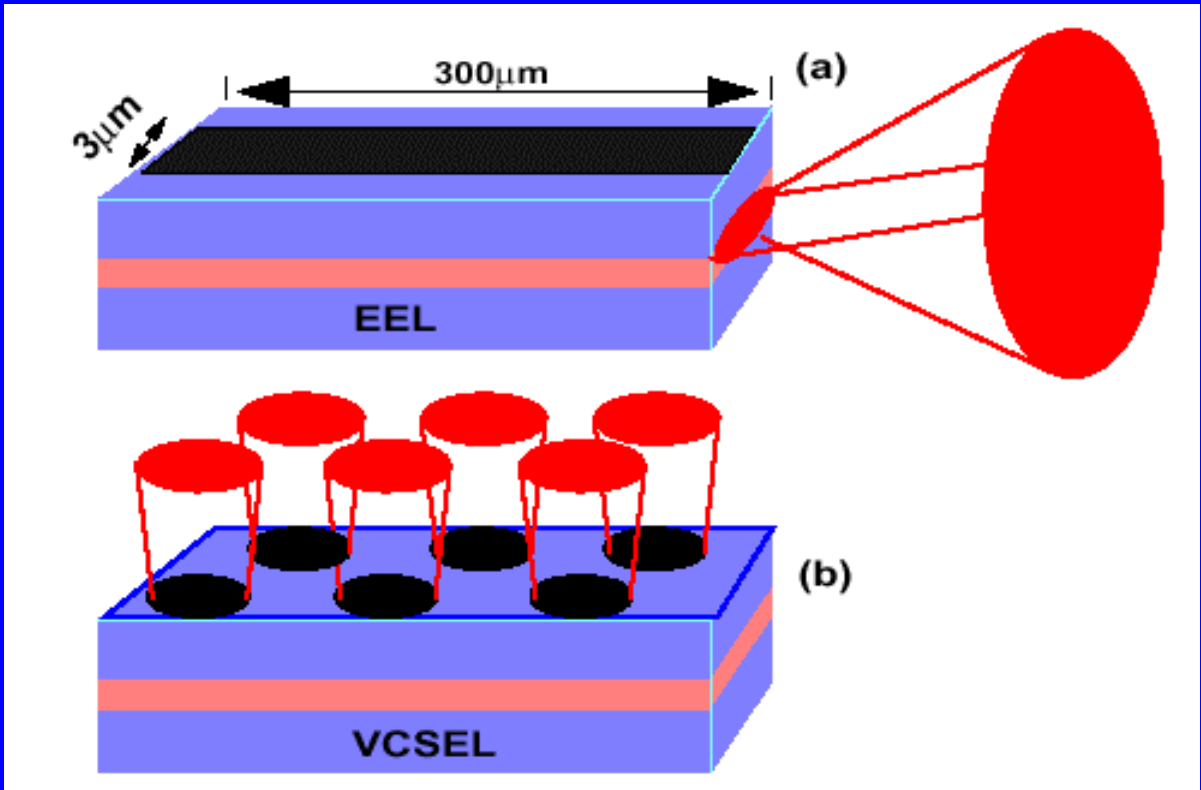
picture from Honeywell

VCSEL/3- Surface Emission



picture from Honeywell

VCSEL/4



pict from Honeywell

VCSEL/5

At costs similar to LEDs have the characteristics of lasers.

EELs:

- it's difficult to produce a geometry w/o problems cutting the chips
- it's not possible to know in advance if the chip it's good or not

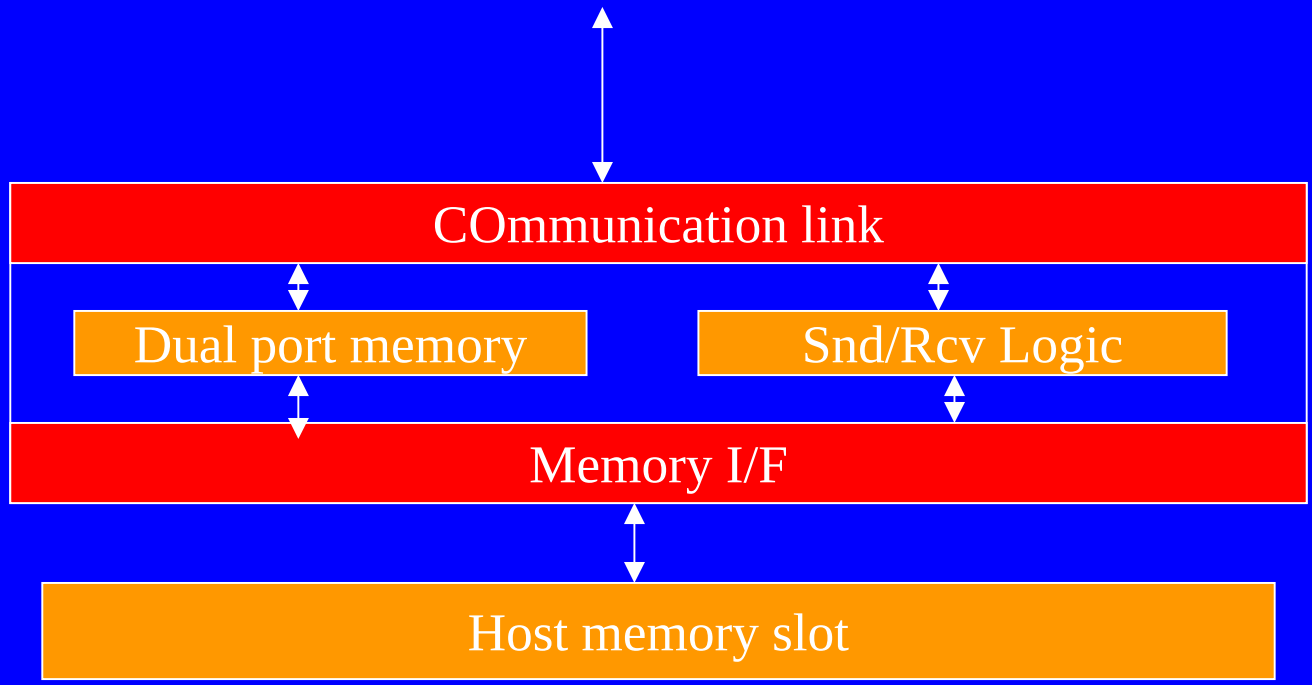
VCSELs:

- they can be produced and tested with standard I.C. procedures
- arrays of 10 or 1000 can be produced easily

MINI
(Memory Integrated Network Interface)

- MINI *(R.Minnich,D.Burns)* IEEE Micro 1995: ATM interface on a SIMM slot
- MEMOnet/DIMMnet: (*Tanabe et al. 2000)* a NIC on a PC-133 DIMM slot
- Estimated performance of DIMMnet-1 :
  - *short msg latency: 250ns*
  - *long msgs b/w: 300-450 MB/s (depending on processor and chipset)*

MINI/2


COmmunication link
Dual port memory
Snd/Rcv Logic
Memory I/F
Host memory slot

Infiniband/1

- It represents the convergence of 2 separate proposal:
  - NGIO (NextGenerationIO:Intel,Microsoft,Sun)
  - FutureIO (Compaq,IBM,HP)
- Infiniband: Compaq, Dell, HP, IBM, Intel, Microsoft, Sun

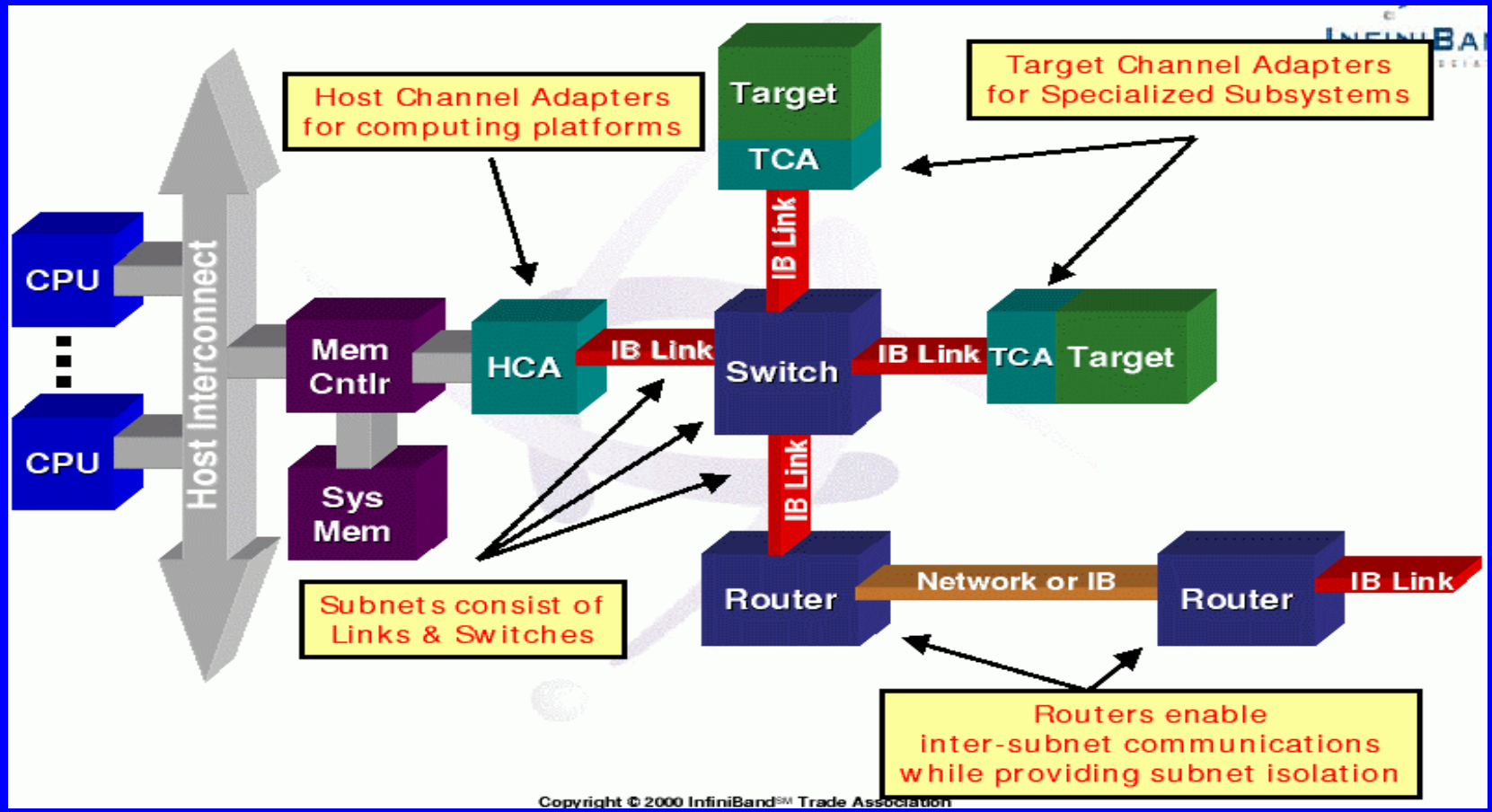
Infiniband/2

- Std channel at 2.5 Gb/s (Copper LVDS or FIber) :
  - 1x width 2.5 Gb/s
  - 4x width 10 Gb/s
  - 12x width 30 Gb/s

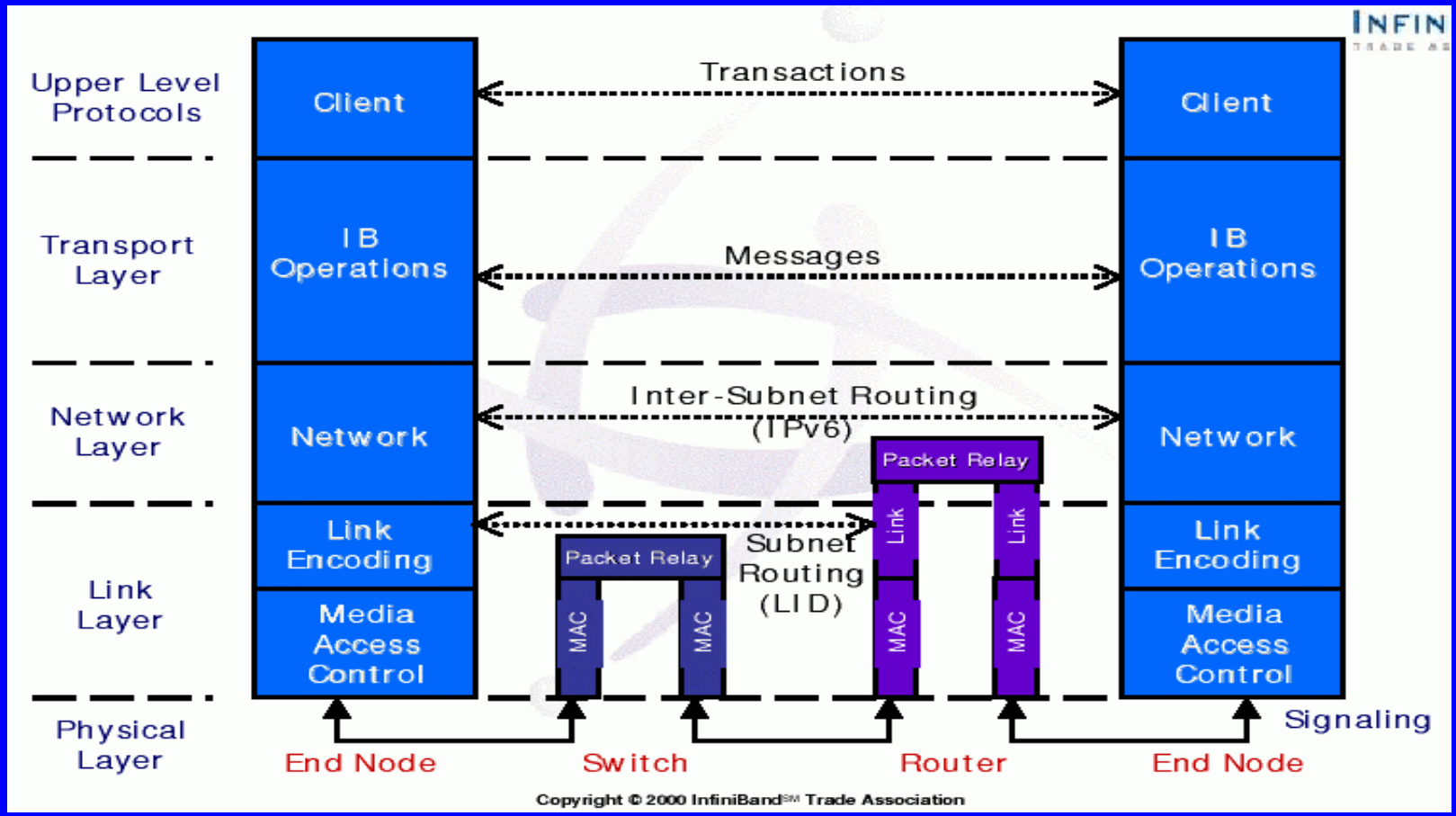
Infiniband/3

- Highlights:
  - point to point switched interconnect
  - channel based message passing
  - computer room interconnect, diameter < 100-300 m
  - one connection for all I/O : ipc, storage I/O, network I/O
  - up to thousands of nodes

Infiniband/4

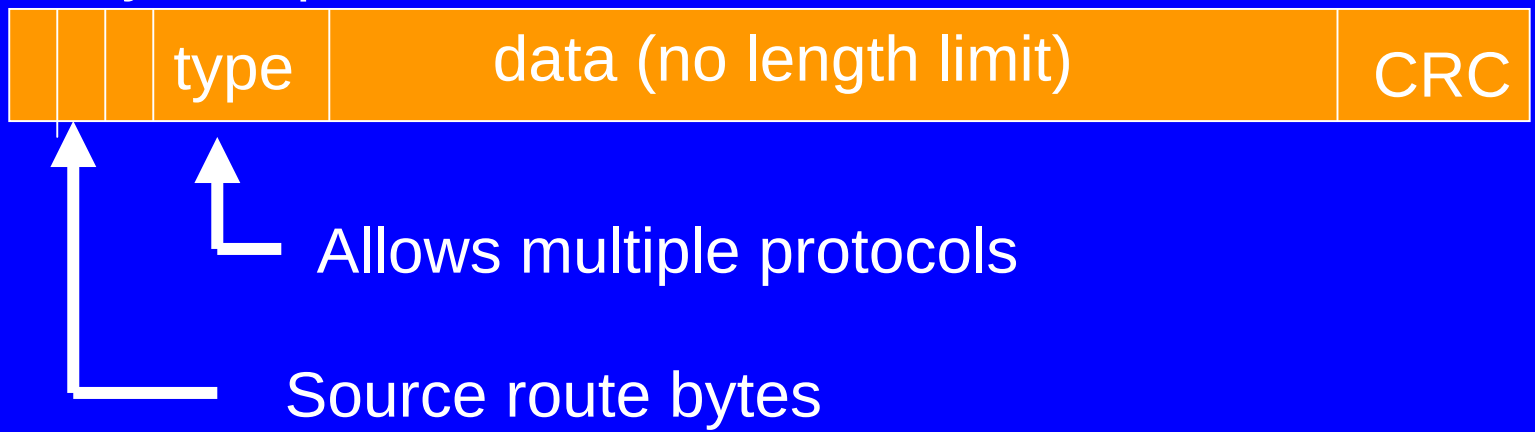


Infiniband/5

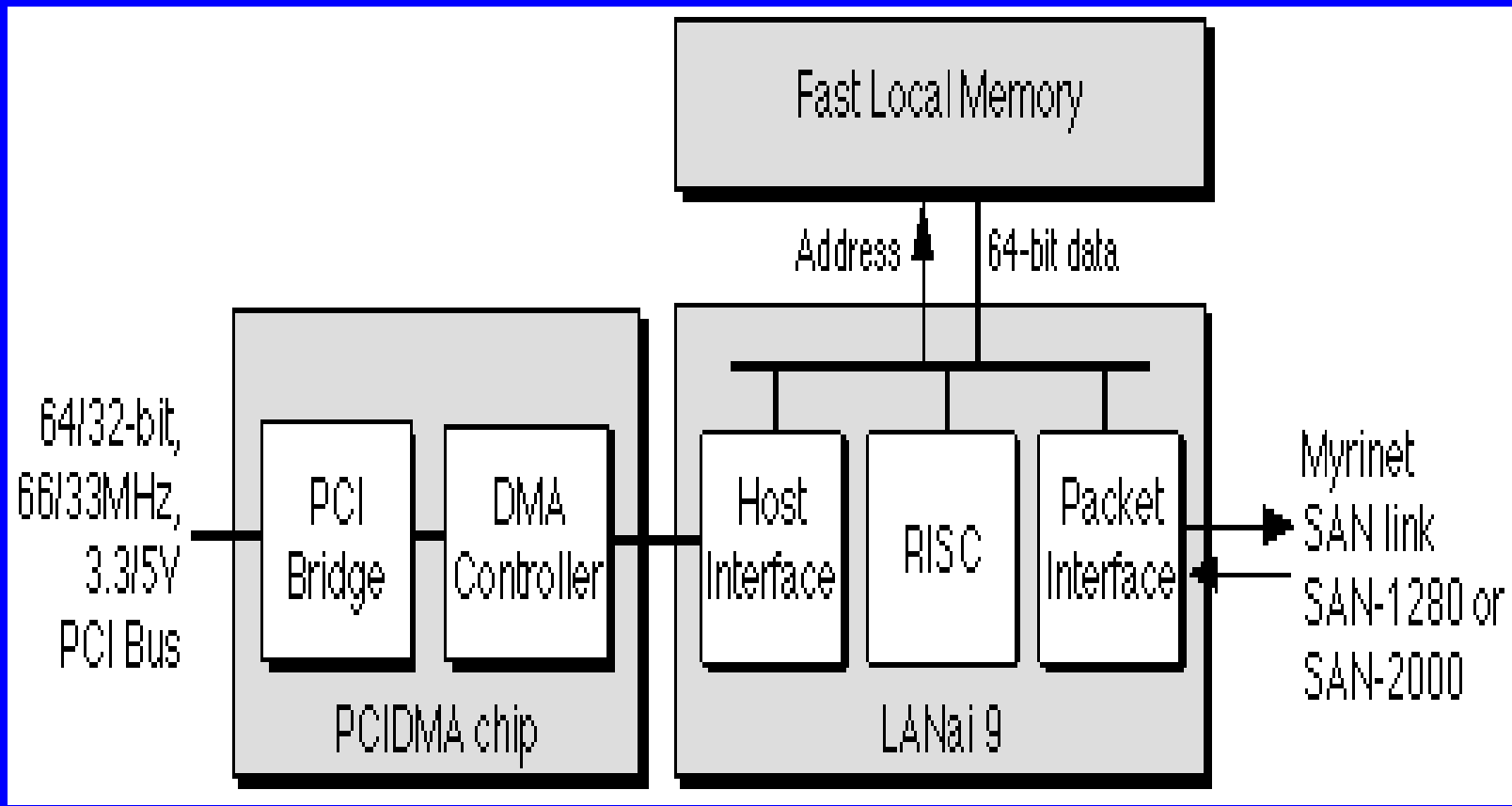


Myrinet/1

- comes from a USC/ISI research project : ATOMIC
- flow contro and error control on all links
- cut-through xbar switches, intelligent boards
- Myrinet packet format :



Myrinet/2


Fast Local Memory
Address
64-bit data
64/32-bit,
66/33MHz,
3.3/5V
PCI Bus
PCI
Bridge
DMA
Controller
PCIDMA chip
Host
Interface
RISC
Packet
Interface
LANai 9
Myrinet
SAN link
SAN-1280 or
SAN-2000

Myrinet/3

Every node can discover the topology with probe packets (*mapper*).

In this way it gets a routing table that can be used to compute headers for any destination.

There is 1 byte for each hop traversed, the first byte with routing information is removed by each switch on the path.

Clos networks

Named after Charles Clos who introduced them in 1953.

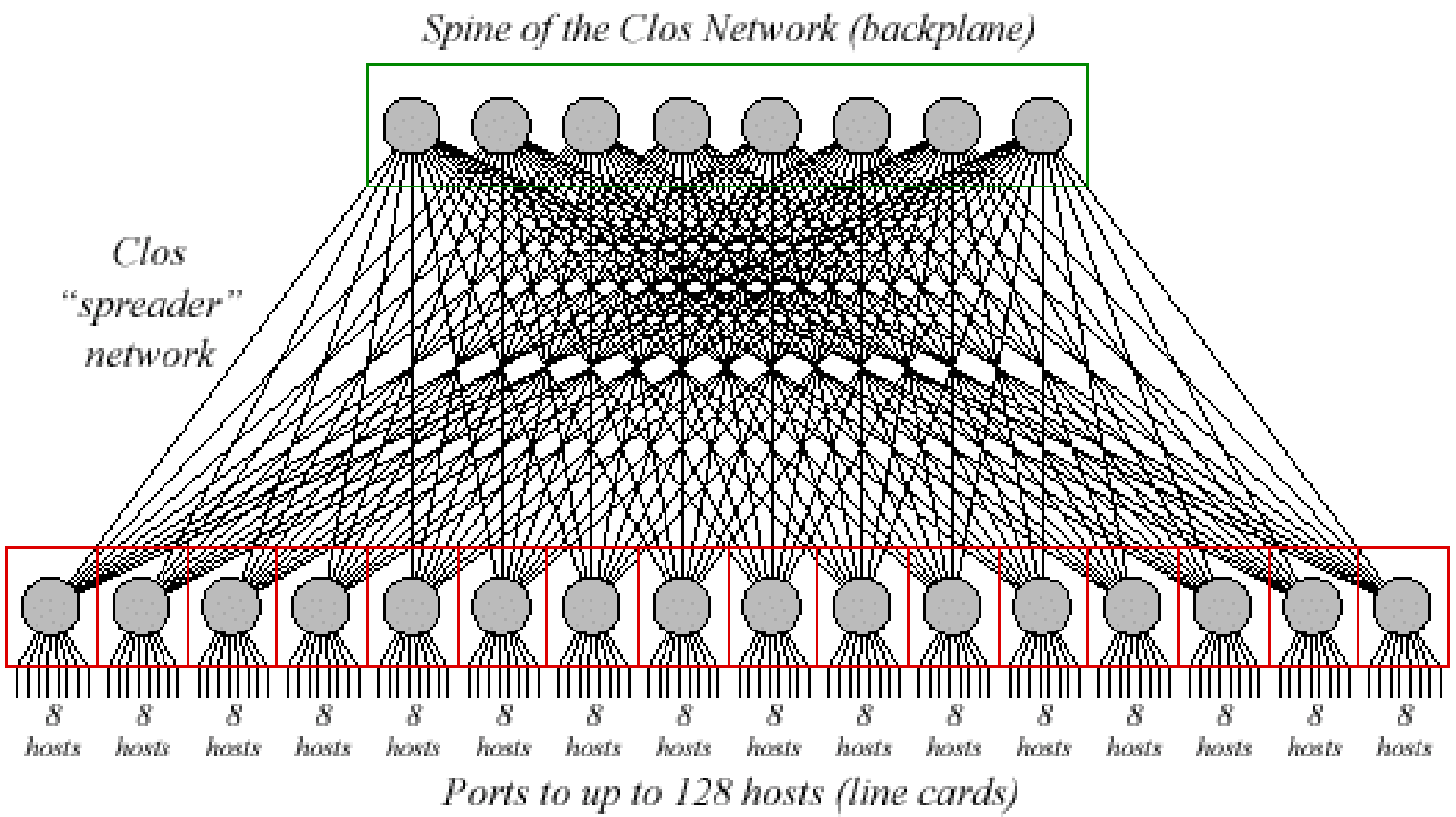
These networks have full bisection.

Large Myrinet clusters are frequently arranged as a Clos network.

Myricom base block is their 16 port xbar switch.

From this brick it’s possible to build 128 nodes/3 level (16+8switches) and 1024 nodes 5 level networks.

Clos networks/2



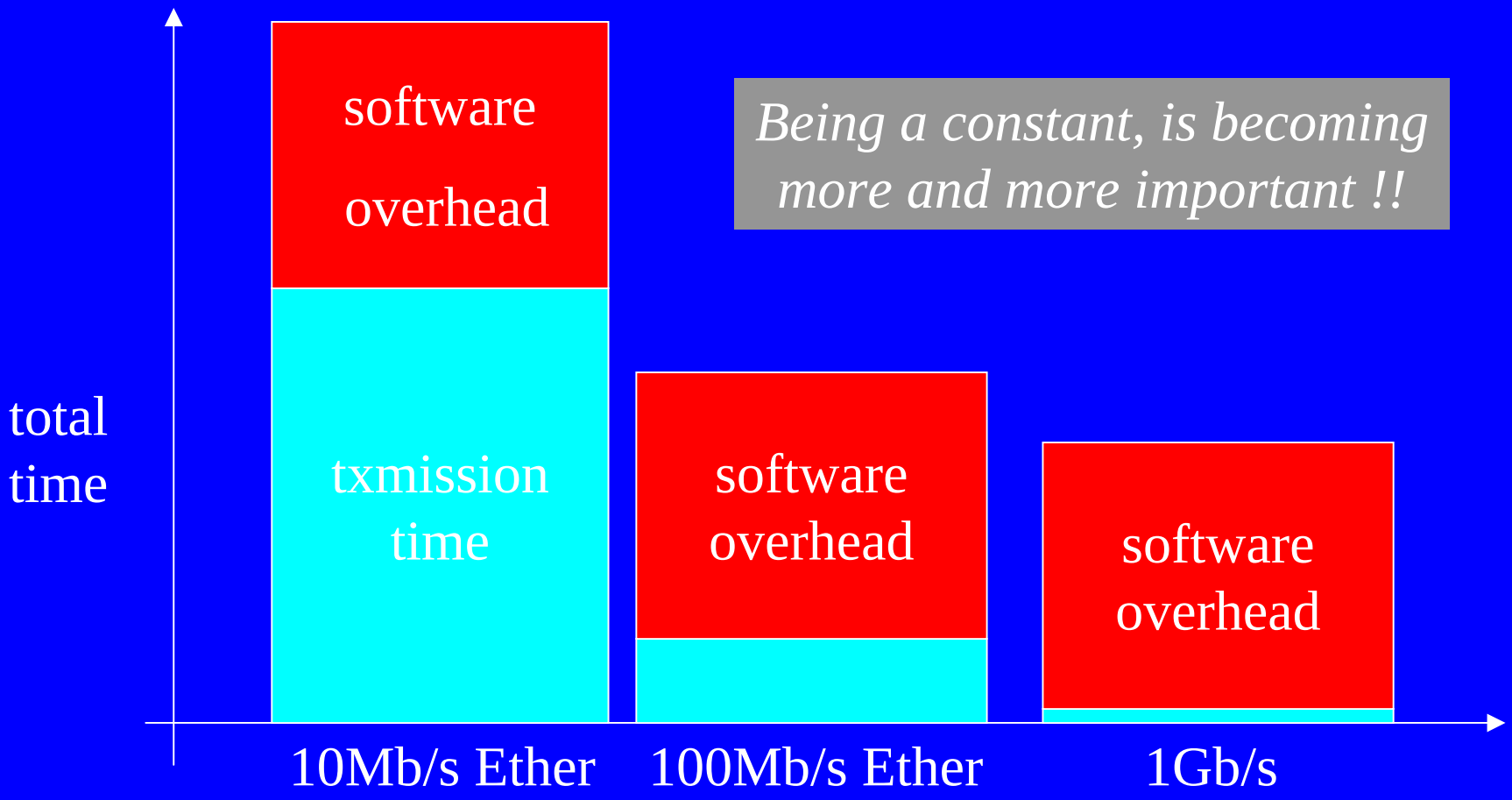
from myri.com

Software

Despite great advances in network technology(2-3
orders of magnitude), much communication s/w remained almost unchanged for many years (e.g.BSD networking).

There is a lot of ongoing research on this theme and very different solutions are proposed(zero-copy, page remapping,VIA,...)

Software overhead



Being a constant, is becoming more and more important !!

Zero Copy Research

High speed networks, I/O systems and memory have comparable bandwidths -> it is essential to avoid any unnecessary copy of data !

- Shared memory between user/kernel:
  - Fbufs*(Druschel,1993)*
  - I/O-Lite *(Druschel,1999)*
- Page remapping with copy on write *(Chu,1996)*
- Blast: hardware splits headers from data*(Carter,O'Malley,1990)*
- Ulni (User-level Network Interface): implementation of communication s/w inside libraries in user space

OS bypass – User level networking

- Active Messages (AM) – *von Eicken, Culler (1992)*
- U-Net –*von Eicken, Basu, Vogels (1995)*
- PM – *Tezuka, Hori, Ishikawa, Sato (1997)*
- Illinois FastMessages (FM) – *Pakin, Karamcheti, Chien (1997)*
- Virtual Interface Architecture (VIA) – *Compaq,Intel,Microsoft (1998)*

Active Messages (AM)

- *1-sided* communication paradigm(no receive op)
- each message as soon as received triggers a *receive handler* that acts as a separate thread (in current implementations it is sender based)

FastMessages (FM)

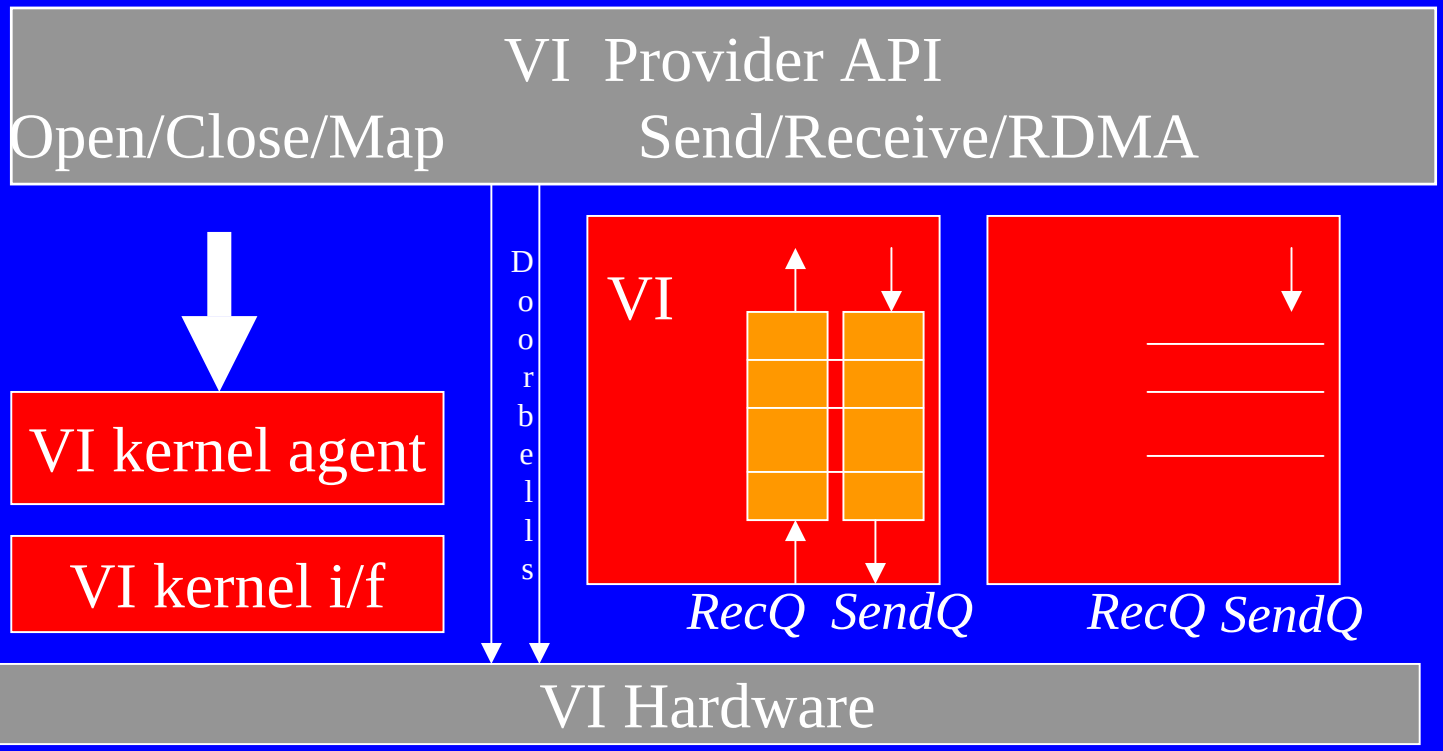
- FM_send(*dest,handler,buf,size*)
  sends a long message
- FM_send_4(*dest,handler,i0,i1,i2,i3*)
  sends a 4 words msg (reg to reg)
- FM_extract()
  process a received msg

VIA/1

Wanting to get at the industrial level the advantages obtained from the various User Level Networking (ULN) initiatives, Compaq, Intel and Microsoft proposed an industry standard called VIA (*Virtual Interface Architecture*). This proposal specifies an API (Application Programming Interface).

In this proposal network reads and writes are done bypassing the OS, while open/close/map are done with the kernel intervention. It requires *memory registering*.

VIA/2


VI Provider API
Open/Close/Map
Send/Receive/RDMA
VI kernel agent
VI kernel i/f
Doorbells
VI
RecQ SendQ
RecQ SendQ
VI Hardware

VIA/3

- VIA is better suited to network cards implementing advanced mechanism like *doorbells* (a queue of transactions in the address space of the memory card, that are remembered by the card)
- Anyway it can also be implemented completely in s/w, despite less efficiently (look at the M-VIA UCB project, and MVICH)

Software layering

Use of abstraction layers has promoted generality, but maybe it can be harmful to efficiency

A typical read/write on a tcp socket passes through:

- VFS(Virtual File System) layer
- BSD socket layer
- Inet socket layer

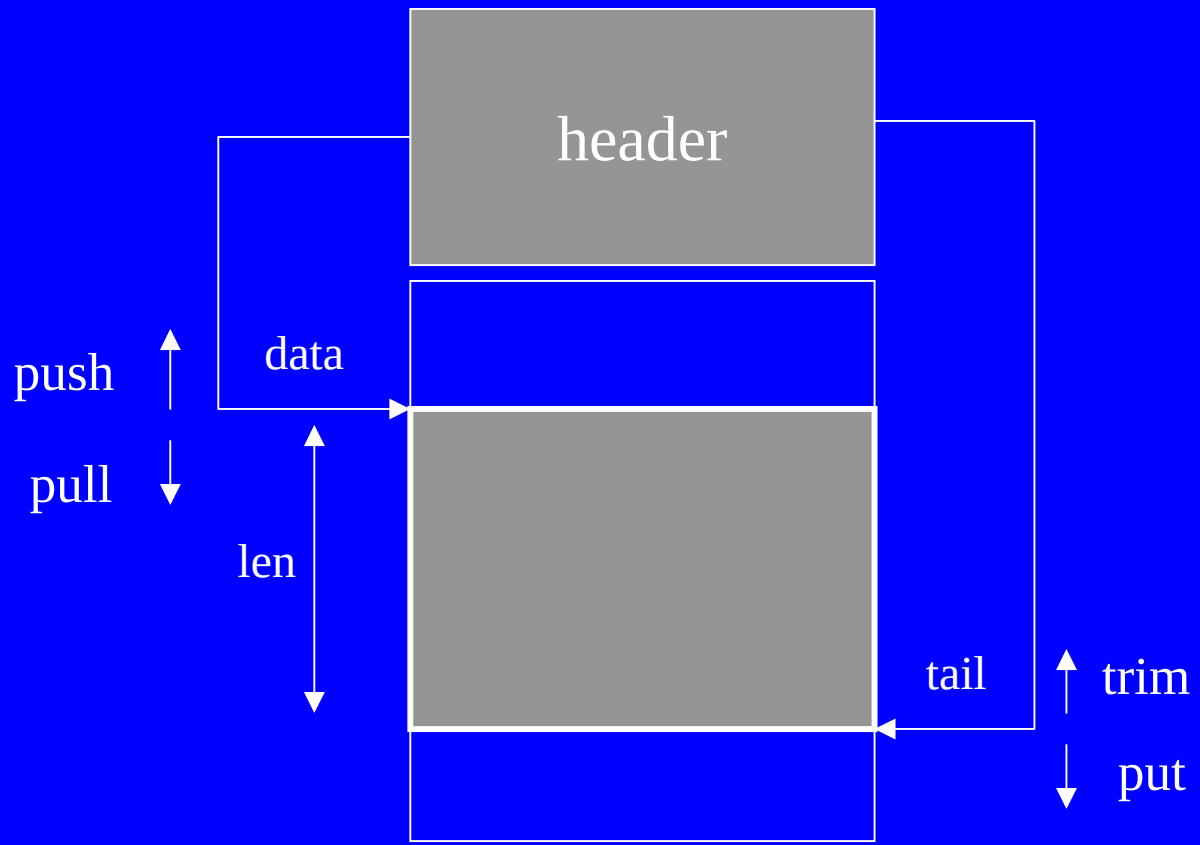
Network layering considered harmful ?

Is the successful network layering approach to networking harmful to today high speed network performance ?

- 7 layers ISO/OSI model
- 4 layers TCP/IP

Yes, if it implies data copying between layers, no if layering is just an abstraction

Linux Socket buffers (sk_buff)



This is the Linux way to avoid copying between network layers, does'nt avoid copies between kernel/user spaces and for frag/defrag-mentation

Memory Management

Linux 2.4 kiobuff

Bibliography

- Patterson/Hennessy: Computer Architecture – A Quantitative Approach
- Hwang – Advanced Computer Architecture
- Schimmel – Unix Systems for Modern Architectures