Workshop on
High Performance Computing (HPC08)
School of Physics, IPM
February 16-21, 2008

Introduction to HPC

Stefano Cozzini
CNR/INFM Democritos and SISSA/eLab

Agenda

- Introduction: what is e-science ?
- High Performance Computing:
  - introduction/ concepts /definitions
- Understanding parallel programming: some ideas
  - Speedup: the effectiveness of parallelism
  - Limits to parallel performance
  - Modern Serial processor and parallelism
- Parallel Machines
- Clusters:
  - definition and some other funny things
- Grid and all the rest
- Wrap-up

Agenda

- Introduction: what is e-science ?
- High Performance Computing:
  - introduction/ concepts /definitions
- Understanding parallel programming: some ideas
  - Speedup: the effectiveness of parallelism
  - Limits to parallel performance
  - Modern Serial processor and parallelism
- Parallel Machines
- Clusters:
  - definition and some other funny things
- Grid and all the rest
- Wrap-up

in search of E-science

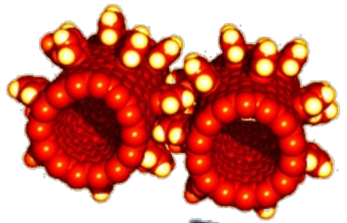
- What is meant by e-Science? In the future, e-Science will refer to the large scale science that will increasingly be carried out through distributed global collaborations enabled by the Internet [from http://www.nesc.ac.uk/nesc/define.html]
- The term e-Science (or eScience) is used to describe computationally intensive science that is carried out in highly distributed network environments

e-science is a buzzword

- Buzzwords are typically intended to impress one's audience with the pretense of knowledge.
- 2006: tools for computational physics:
    - 200 application 1000 Euro sponsors
- 2007: HPC tools for e-Science:
    - 400 applications
    - 6000 Euro + hardware from sponsors + books donation...

e-science=computationally intensive science

- Science is becoming increasingly digital and needs to deal with increasing amounts of data and computing power
- Simulations get ever more detailed
  - Nanotechnology – design of new materials from the molecular scale
  - Modelling and predicting complex systems (weather forecasting, river floods, earthquake)
  - Decoding the human genome
- Experimental Science uses ever more sophisticated sensors to make precise measurements
  - → Need high statistics
  - → Huge amounts of data
  - → Serves user communities around the world



e-science= new approach to do science

- New tools&methods
  - distribute collaborations
  - pooling of resources geographically distributed (GRID Computing)
  - powerful and modern hardware/software (High Performance Computing)
  - IT- skilled computational scientists

Agenda

- Introduction: what is e-science ?
- High Performance Computing:
  - introduction/ concepts /definitions
- Understanding parallel programming: some ideas
  - Speedup: the effectiveness of parallelism
  - Limits to parallel performance
  - Modern Serial processor and parallelism
- Parallel Machines
- Clusters:
  - definition and some other funny things
- Grid and all the rest
- Wrap-up

High Performance Computing (HPC)

- performance is everything (well, almost everything):
- I want ...
    - my calculation run faster and faster...
- it ranges from your laptop to the cutting-edge supercomputers
- it is not only on hardware but involves software and people as well

How to run application faster ?

- There are 3 ways to improve performance:
  - Work Harder
  - Work Smarter
  - Get Help
- Computer Analogy
  - Using faster hardware
  - Optimized algorithms and techniques used to solve computational tasks
  - Multiple computers to solve a particular task

see optimization lecture

See MPI tutorial

Units of High Performance Computing:

- Processor speed:

  Floats: floating point operation/ second

  - Mega flops / Gigaflops / Teraflops / Petaflops
- Network speed:

  bits : bit /second transmitted
  10Mbit/100Mbit/1000Mbit=1Gbit and now also 10Gb
- Size unit: byte
  - kbyte/Mbyte ----> caches/RAM
  - Gigabite -----> RAM/hard disks
  - Terabyte -----> Disks/SAN ...
  - Petabyte ------> SAN

Agenda

- Introduction: what is e-science ?
- High Performance Computing:
  - introduction/ concepts /definitions
- Understanding parallel programming: some ideas
  - Speedup: the effectiveness of parallelism
  - Limits to parallel performance
  - Modern Serial processor and parallelism
- Parallel Machines
- Clusters:
  - definition and some other funny things
- Grid and all the rest
- Wrap-up

defining parallel computing

- Parallel computing is the simultaneous execution of the same task (split up and specially adapted) on multiple processors in order to obtain results faster.
- The process of solving a problem usually can be divided into smaller tasks, which may be carried out simultaneously with some coordination.
  [from wikipedia]

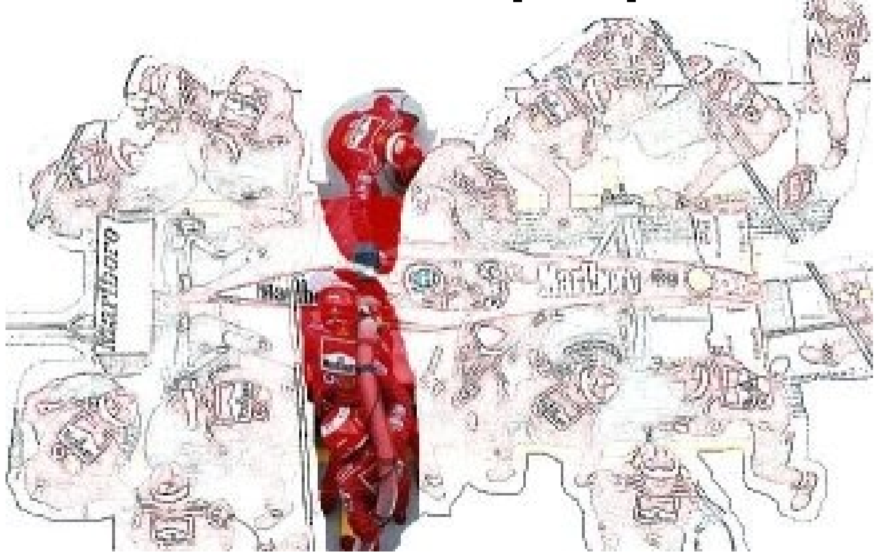
high performance problem example:

picture from http://www.f1nutter.co.uk/tech/pitstop.php

analysis of the parallel solution:

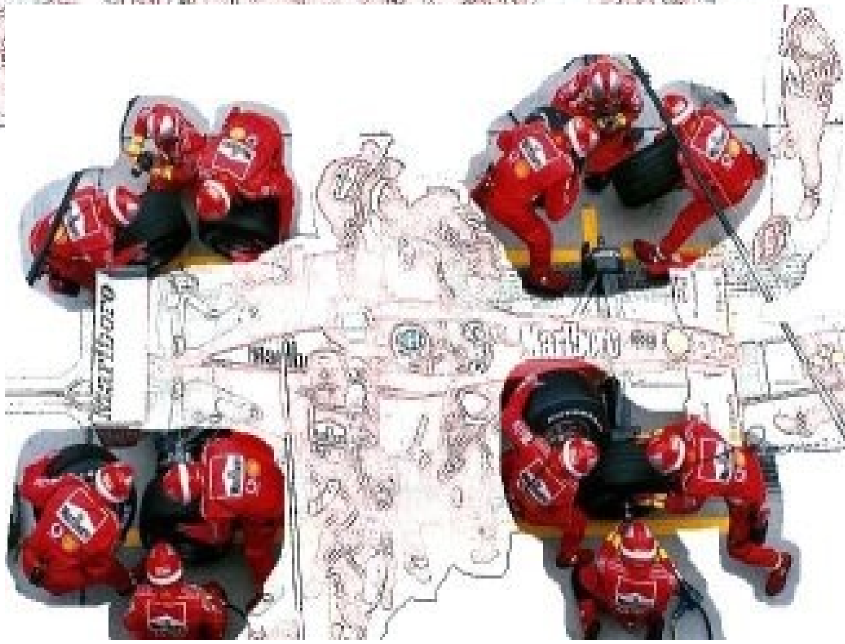
FUNCTIONAL PARTITIONING

different people are executing different tasks

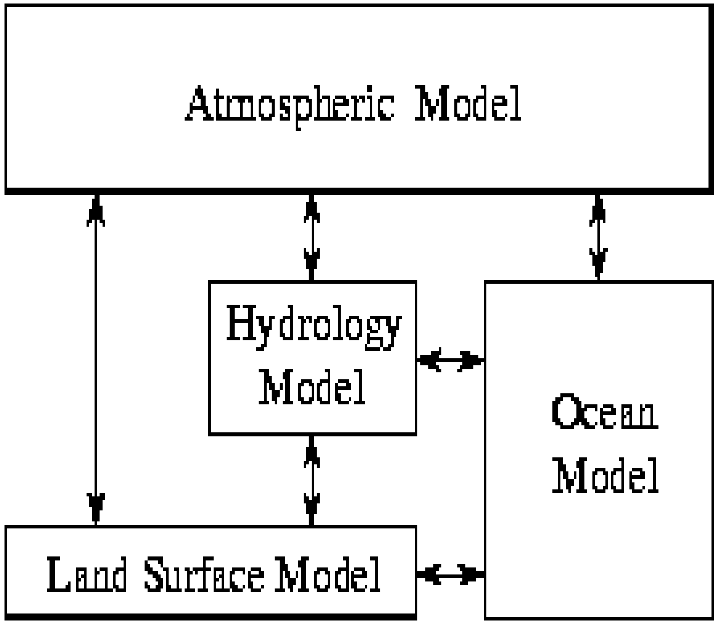


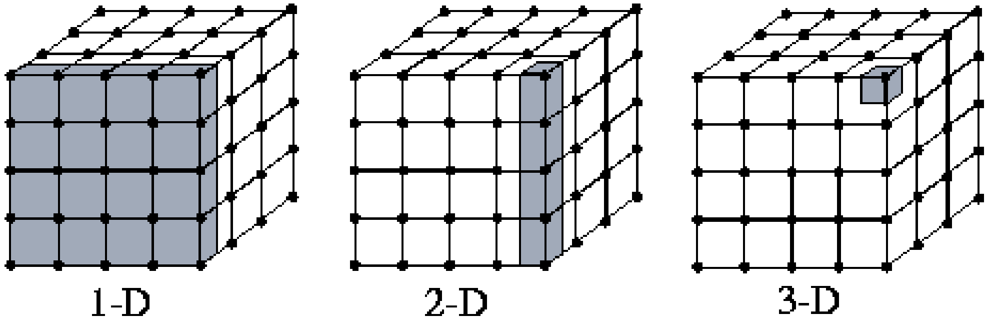
DOMAIN DECOMPOSITION

different people are solving the same global task but on smaller subset



Parallel computing techniques





- DOMAIN DECOMPOSITION

- FUNCTIONAL PARTITIONING

EFFICIENT SOLUTION TO THE PROBLEM

picture from the on-line book:
http://www-unix.mcs.anl.gov/dbpp/

Principles of Parallel Computing

- Speedup, efficiency, and Amdahl’s Law
- Finding and exploiting parallelism
- Finding and exploiting data locality
- Load balancing
- Coordination and synchronization
- Performance modeling

All of these things make parallel programming more difficult than sequential programming.

Speedup

- The *speedup* of a parallel application is

  Speedup(p) = Time(1)/Time(p)

- Where
  - Time(1) = execution time for a single processor
  - Time(p) = execution time using p parallel processors
- If Speedup(p) = p we have *perfect speedup* (also called *linear scaling*)
- speedup compares an application with itself on one and on p processors
- more useful to compare
  - The execution time of the best serial application on 1 processor

versus

  - The execution time of best parallel algorithm on p processors

Efficiency

- The *parallel efficiency* of an application is defined as

  Efficiency(p) = Speedup(p)/p

  - Efficiency(p) <= 1
  - For perfect speedup Efficiency (p) = 1
- We will rarely have perfect speedup.
  - Lack of perfect parallelism in the application or algorithm
  - Imperfect load balancing (some processors have more work)
  - Cost of communication
  - Cost of contention for resources, e.g., memory bus, I/O
  - Synchronization time
- Understanding why an application is not scaling linearly will help finding ways improving the applications performance on parallel computers.

Superlinear Speedup

Question: can we find "*superlinear*" speedup, that is

$$\text{Speedup}(p) > p \quad ?$$

- Choosing a bad "baseline" for T(1)
  - Old serial code has not been updated with optimizations
  - Avoid this, and always specify what your baseline is
- Shrinking the problem size per processor
  - May allow it to fit in small fast memory (cache)
- Application is not deterministic
  - Amount of work varies depending on execution order
  - Search algorithms have this characteristic

Amdahl's Law

- Suppose only part of an application runs in parallel
- Amdahl's law
  - Let s be the fraction of work done serially,
  - So (1-s) is fraction done in parallel
  - What is the maximum speedup for P processors?

$$\text{Speedup}(p) = T(1)/T(p)$$

$$T(p) = \underbrace{(1-s)*T(1)/p}_{\text{assumes perfect speedup for parallel part}} + s*T(1)$$

$$= T(1)*((1-s) + p*s)/p$$

$$\boxed{\text{Speedup}(p) = p/(1 + (p-1)*s)}$$

Even if the parallel part speeds up perfectly, we may be limited by the sequential portion of code.

Amdahl's law(2)

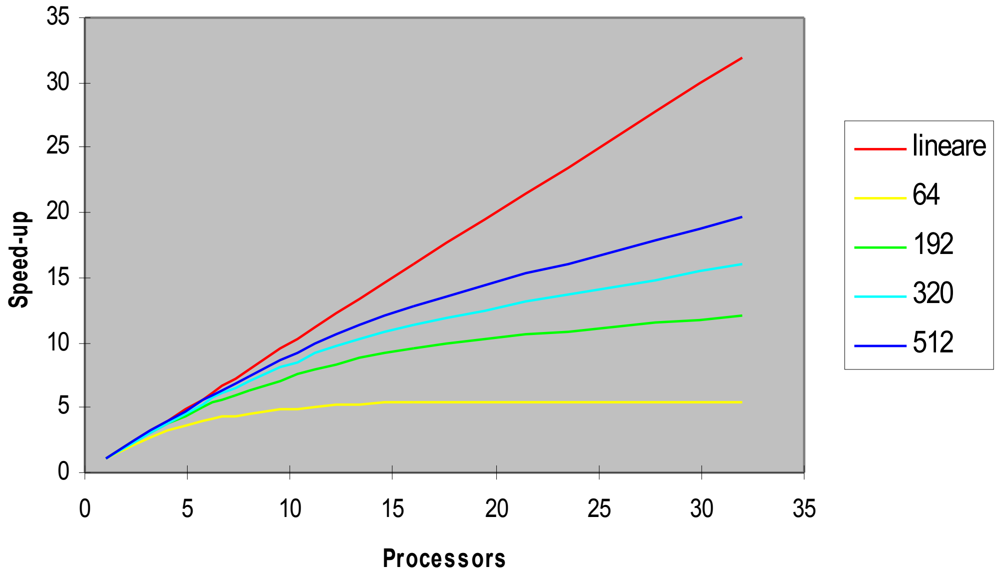
- Which fraction of serial code is it allowed ?

>	2	4	8	32	64	256	512	1024
5%	1.91	3.48	5.93	12.55	15.42	18.62	19.28	19.63
2%	1.94	3.67	6.61	16.58	22.15	29.60	31.35	32.31
1%	1.99	3.88	7.48	24.43	39.29	72.11	83.80	91.18

What about Scalability ???

Problem scaling..

- Amdahl’s Law is relevant only if serial fraction is indipendent of problem size, which is rarely true
- Fortunately “The proportion of the computations that are sequential (non parallel) normally decreases as the problem size increases ” (a.k.a. Gustafon’s Law)



Scaled Speedup

- Speedup improves as the problem size grows
  - Among other things, the Amdahl effect is smaller
- Consider
  - scaling the problem size with the number of processors (add problem size parameter, n)
  - for problem in which running time scales linearly with the problem size: T(1,n) = T(1)*n
  - let n=p (problem size on p processors increases by p)

ScaledSpeedup(p,n) = T(1,n)/T(p,n)

T(p,n) = (1-s)*n*T(1,1)/p +s*T(1,1)

= (1-s)*T(1,1) + s*T(1,1)=T(1,1)

assumes serial work does not grow with n

ScaledSpeedup(p,n) = n = p

Scaled Efficiency

- Previous definition of *parallel efficiency* was

  Efficiency(p) = Speedup(p)/p

- We often want to scale problem size with the number of processors, but scaled speedup can be tricky
  - Previous definition depended on a linear work in problem size
- May use alternate definition of efficiency that depends on a notion of throughput or rate, R(p):
  - Floating point operations per second
  - Transactions per second
  - Strings matches per second
- Then

  Efficiency(p) = R(p)/(R(1)*p)

- May use a different problem size for R(1) and R(p)

Three Definitions of Efficiency: Summary

- People use the word “efficiency” in many ways
- Performance relative to advertised machine peak

  Flop/s in application / Max Flops/s on the machine

  - Integer, string, logical or other operations could be used, but they should be a machine-level instruction
- Efficiency of a fixed problem size

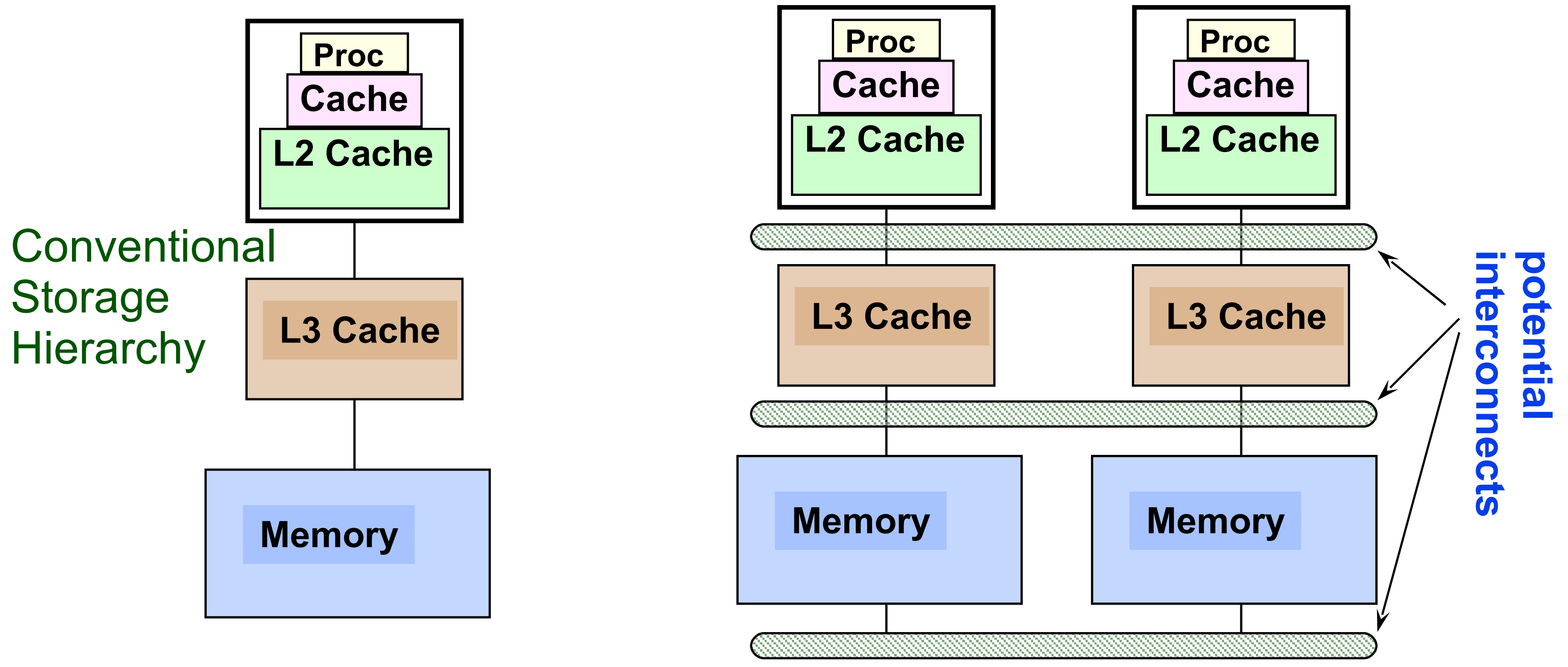
  Efficiency(p) = Speedup(p)/p
- Efficiency of a scaled problem size

  Efficiency(p) = R(p)/(R(1)*p)
- All of these may be useful in some context
- Always make it clear what you are measuring

Overhead of Parallelism

- Given enough parallel work, this is the most significant barrier to getting desired speedup.
- Parallelism overheads include:
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - extra (redundant) computation
- Each of these can be in the range of milliseconds (= millions of arithmetic ops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work.

Locality and Parallelism



- Large memories are slower; fast memories are small.
- Storage hierarchies are designed to fast <u>on average.</u>
- Parallel processors, collectively, have large, fast memories -- the slow accesses to “remote” data we call “communication”.
- Algorithm should do most work on local data.

Load Imbalance

- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase).
  - unequal size tasks.
- Examples of the latter
  - adapting to “interesting parts of a domain”.
  - tree-structured computations.
  - fundamentally unstructured problems.
- Algorithm needs to balance load
  - but techniques the balance load often reduce locality

Idealized Uniprocessor Model

- Processor names bytes, words, etc. in its address space
  - These represent integers, floats, pointers, arrays, etc.
  - Exist in the program stack, static region, or heap
- Operations include
  - Read and write (given an address/pointer)
  - Arithmetic and other logical operations
- Order specified by program
  - Read returns the most recently written data
  - Compiler and architecture translate high level expressions into “obvious” lower level instructions
  - Hardware executes instructions in order specified by compiler
- Cost
  - Each operations has roughly the same cost

    (read, write, add, multiply, etc.)

Uniprocessors in the Real World

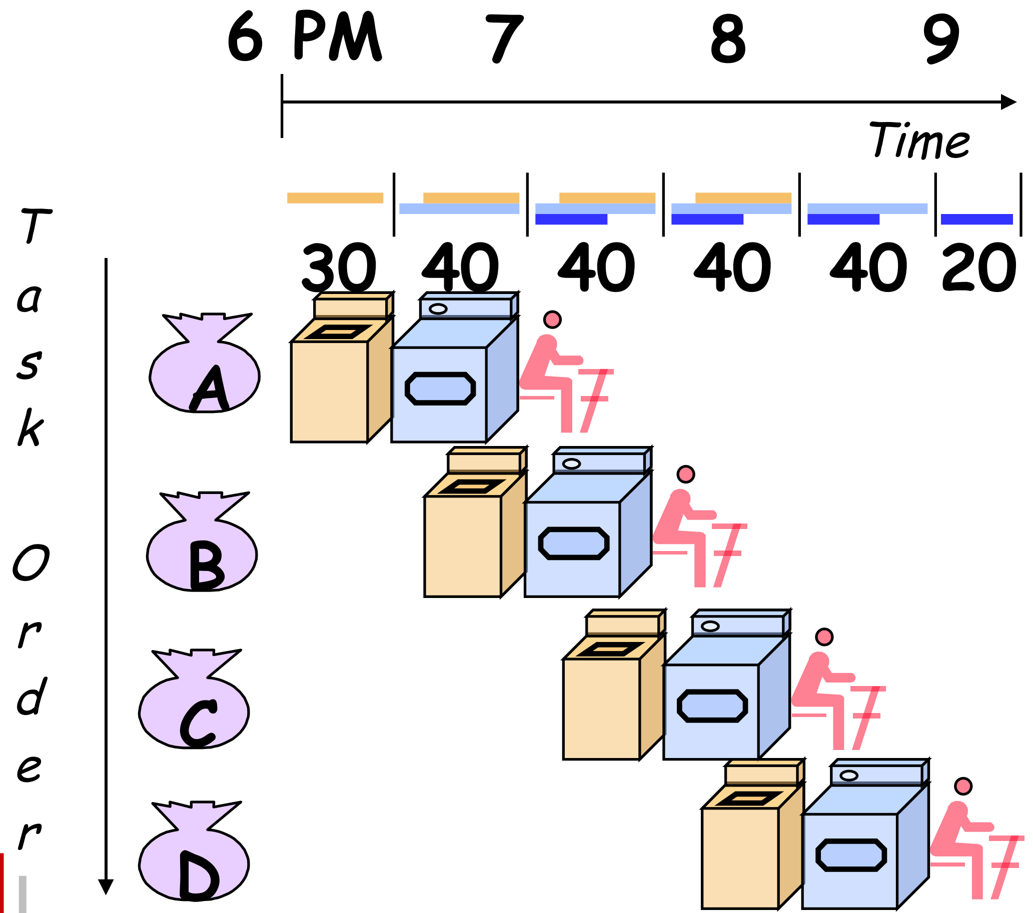
- Real processors have
  - registers and caches
    - small amounts of fast memory
    - store values of recently used or nearby data
    - different memory ops can have very different costs
  - parallelism
    - multiple “functional units” that can run in parallel
    - different orders, instruction mixes have different costs
  - pipelining
    - a form of parallelism, like an assembly line in a factory
- Why is this your problem?

In theory, compilers understand all of this and can optimize your program; in practice they don’t.

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
- Pipelining helps throughput, but not latency
- Pipeline rate limited by slowest pipeline stage
- Potential speedup = Number pipe stages
- Time to "fill" pipeline and time to "drain" it reduces speedup

Limits to Instruction Level Parallelism (ILP)

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).
- The hardware and compiler will try to reduce these:
  - Reordering instructions, multiple issue, dynamic branch prediction, speculative execution…
- You can also enable parallelism by careful coding

Lessons

- Actual performance of a simple program can be a complicated function of the architecture
  - Slight changes in the architecture or program change the performance significantly
  - To write fast programs, need to consider architecture

Agenda

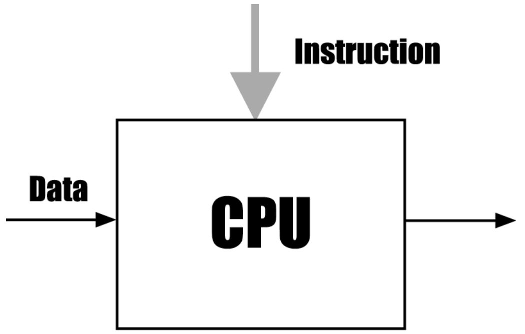
- Introduction: what is e-science ?
- High Performance Computing:
  - introduction/ concepts /definitions
- Understanding parallel programming: some ideas
  - Speedup: the effectiveness of parallelism
  - Limits to parallel performance
  - Modern Serial processor and parallelism
- Parallel Machines
- Clusters:
  - definition and some other funny things
- Grid and all the rest
- Wrap-up

Parallel computers

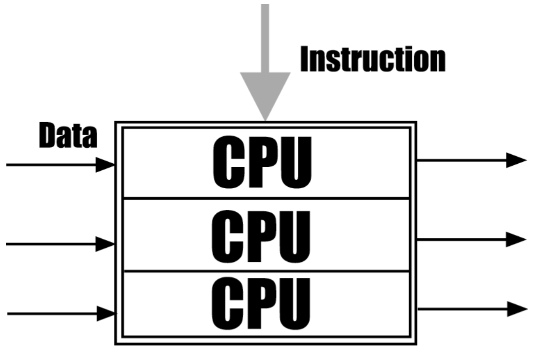
- Tons of different machines !
- Flynn Taxonomy (1966): helps (?) us in classifying them:
  - Data Stream
  - Instruction Stream

Name	Instruction stream	Data stream
SISD	Single	Single
SIMD	Single	Multiple
MIMD	Multiple	Multiple
MISD	Multiple	Single

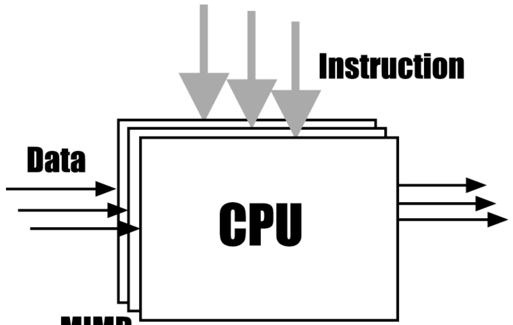
Flynn Taxonomy (graphical view)



SISD
(Single instruction stream
single data stream)



SIMD
(Single instruction stream
multipe data stream)



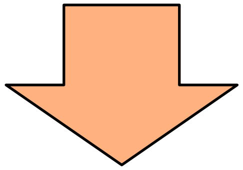
MIMD
(Multipe instruction stream
mingle data stream)

Another important question:

- MEMORY: The simplest and most useful way to classify modern parallel computers is by their memory model:
  - SHARED MEMORY
  - DISTRIBUTED MEMORY

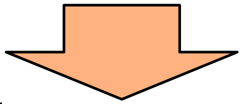
Shared vs Distributed ?

- Distributed Memory each processor has it’s own local memory. Must do message passing to exchange data between processors

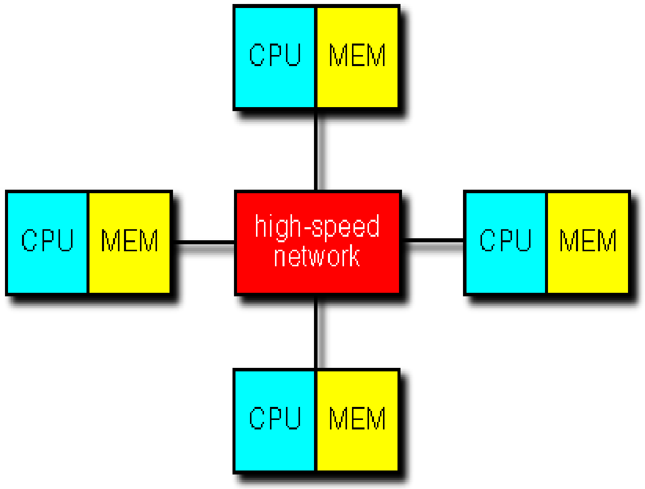


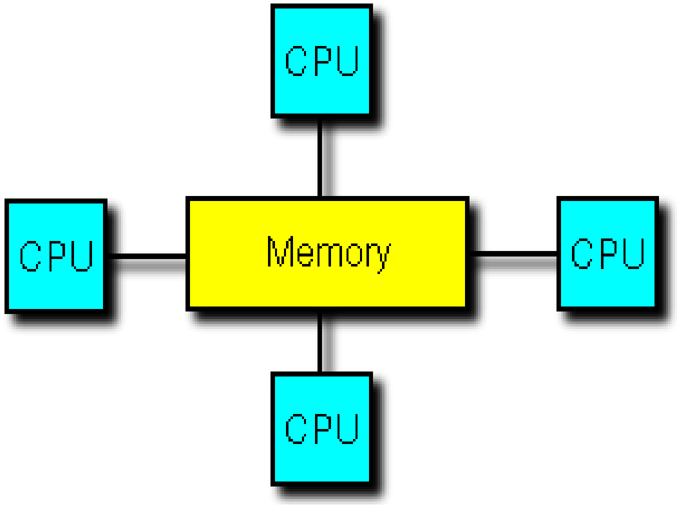
  - **multicomputers**

- Shared Memory
  - single address space. All processors have access to a pool of shared memory.



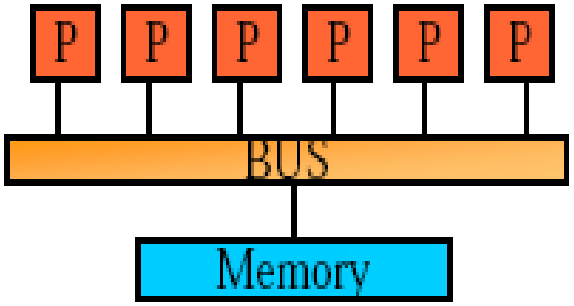
- **Multiprocessors (MPs)**



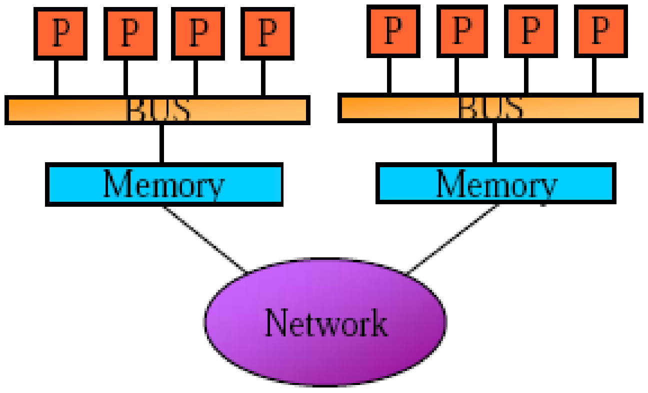


Shared Memory: UMA vs NUMA

- *Uniform memory access (UMA):* Each processor has uniform access to memory. Also known as symmetric multiprocessors (SMP)



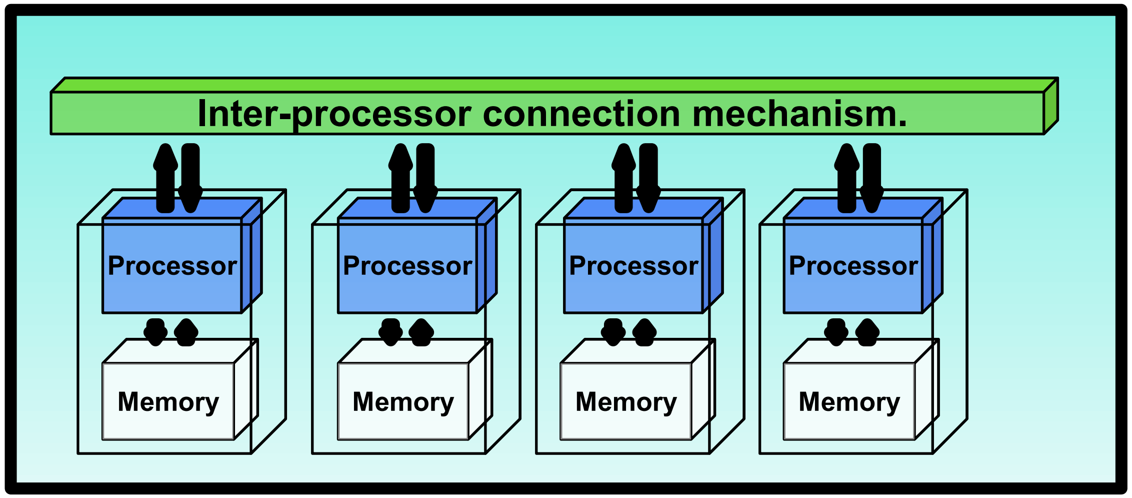
- *Non-uniform memory access (NUMA):* Time for s memory access depends on location of data. Local access is faster than non-local access.



Distributed memory architecture: Clusters !

- Subject: Re: [Beowulf] about concept of Beowulf clusters Date: Thu, 24 Feb 2005 19:41:22 -0500 (EST) From: Donald Becker <becker@scyld.com>

CLUSTER: independent machines combined into a unified system through software and networking



Agenda

- Introduction: what is e-science ?
- High Performance Computing:
  - introduction/ concepts /definitions
- Understanding parallel programming: some ideas
  - Speedup: the effectiveness of parallelism
  - Limits to parallel performance
  - Modern Serial processor and parallelism
- Parallel Machines
- Clusters:
  - definition and some other funny things
- Grid and all the rest
- Wrap-up

Beowulf Clusters !

- Subject: Re: [Beowulf] about concept of beowulf clusters Date: Thu, 24 Feb 2005 19:41:22 -0500 (EST) From: Donald Becker <becker@scyld.com>
- The Beowulf definition is commodity machines connected by a private cluster network running an open source software infrastructure for scalable performance computing
- this means:

commodity machines: exclude custom built hardware e.g. a single Altix is not a Beowulf cluster (or even a cluster by the strict definition)

connected by a cluster network: These machines are dedicated to being a cluster, at least temporarily. This excludes cycle scavenging from NOWs and wide area grids.

running an open source infrastructure The core elements of the system are open source and verifiable

for scalable performance computing The goal is to scale up performance over many dimensions, rather than simulate a single more reliable machine e.g. fail-over. Ideally a cluster incrementally scales both up and down, rather than being a fixed size.

The Cluster revolution in HPC

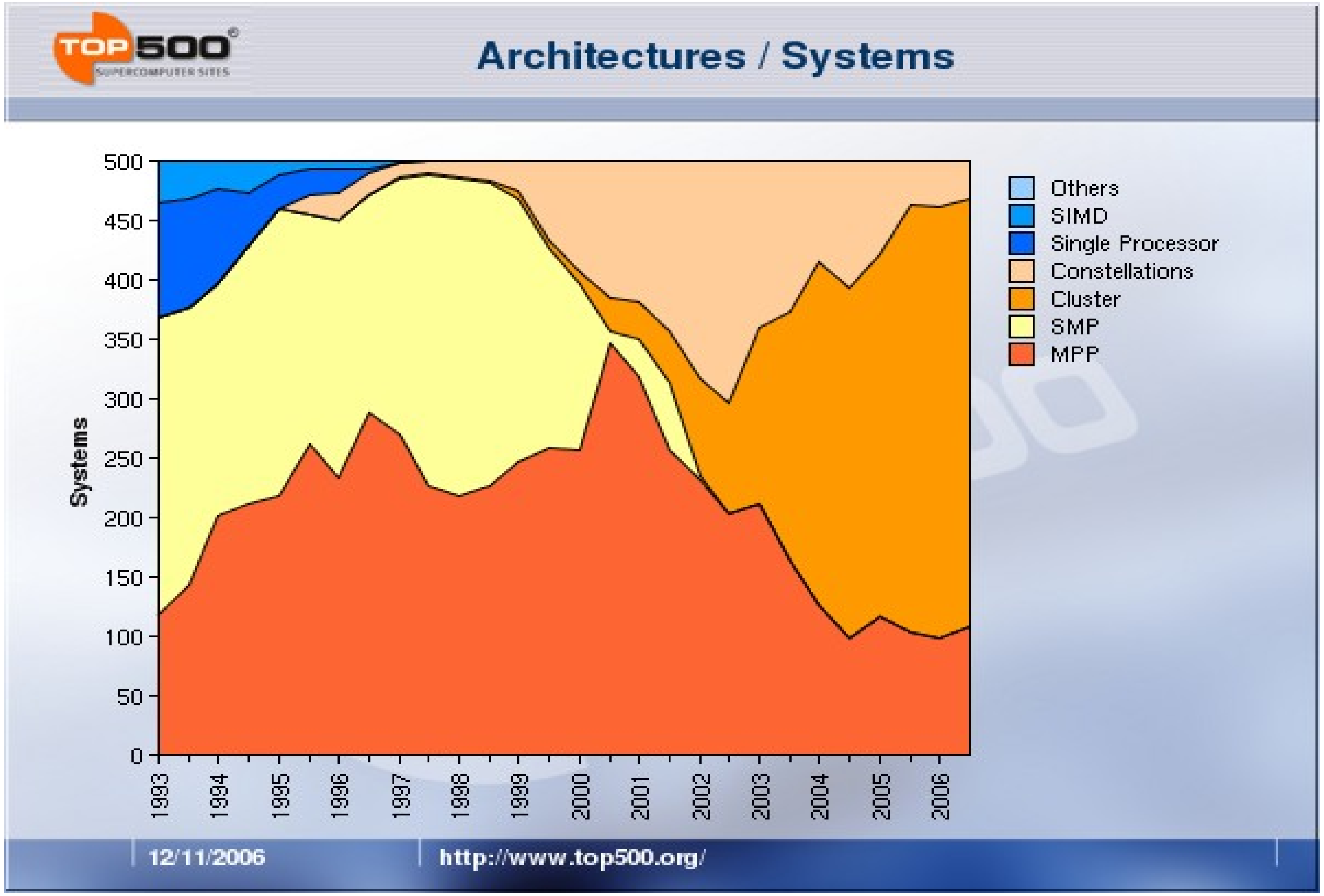
- The adoption of clusters, virtually exploded since the introduction of the first Beowulf cluster in 1994.
- The attraction lies
  - in the (potentially) low cost of both hardware and software
  - the control that builders/users have over their system.
- The problem lies:
  - you should be an expert to build and run efficiently your clusters
  - not always the problem you have fit into a cluster solution (even if this is cheap!)

really a cluster revolution ?

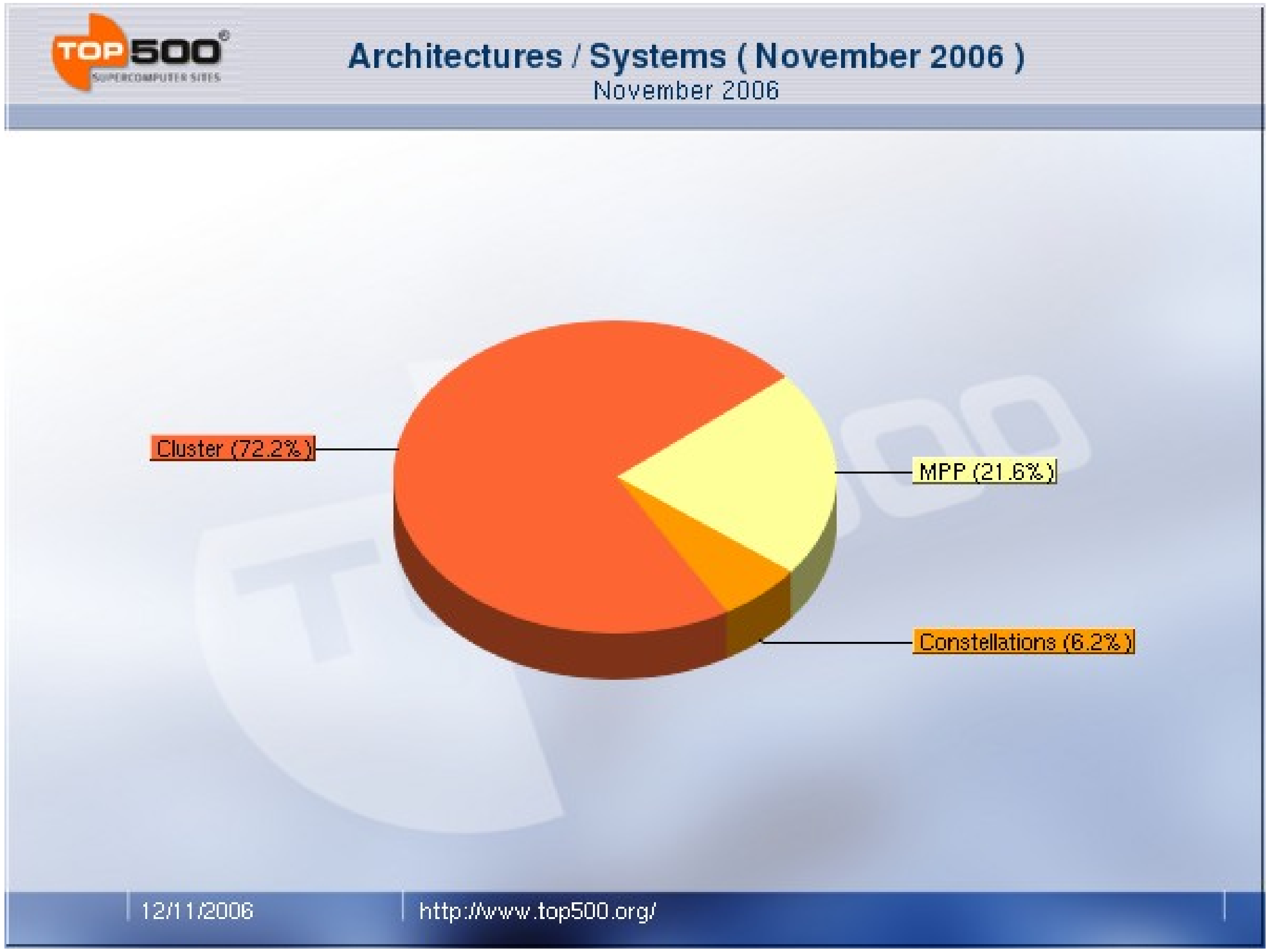
Let us check the Top500 list

- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP Ax=b, dense problem
- Updated twice a year
  - SC‘xy in the States in November
  - Meeting in Germany in June
- All data available from www.top500.org

architectures over last 13 years

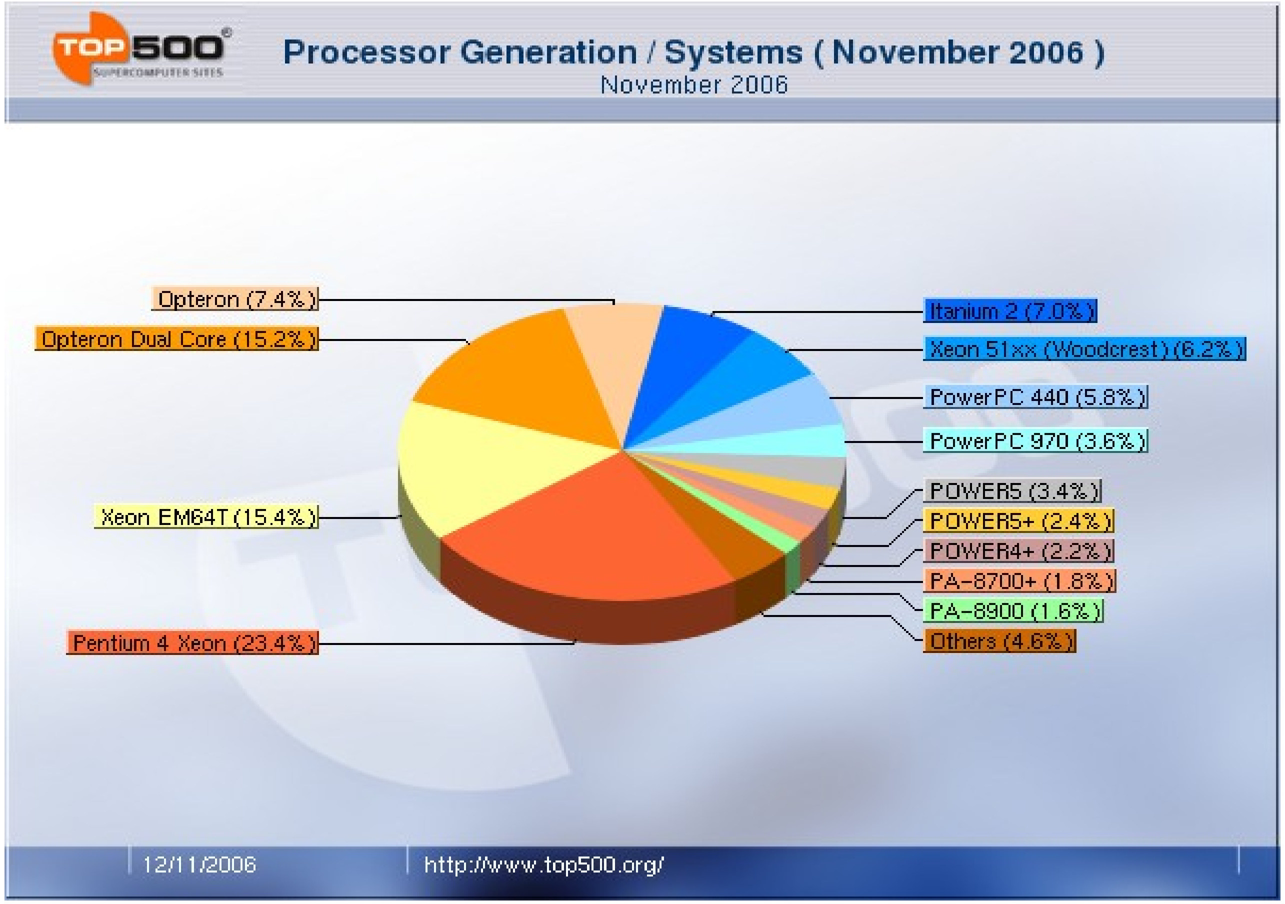


current snapshot about architecture



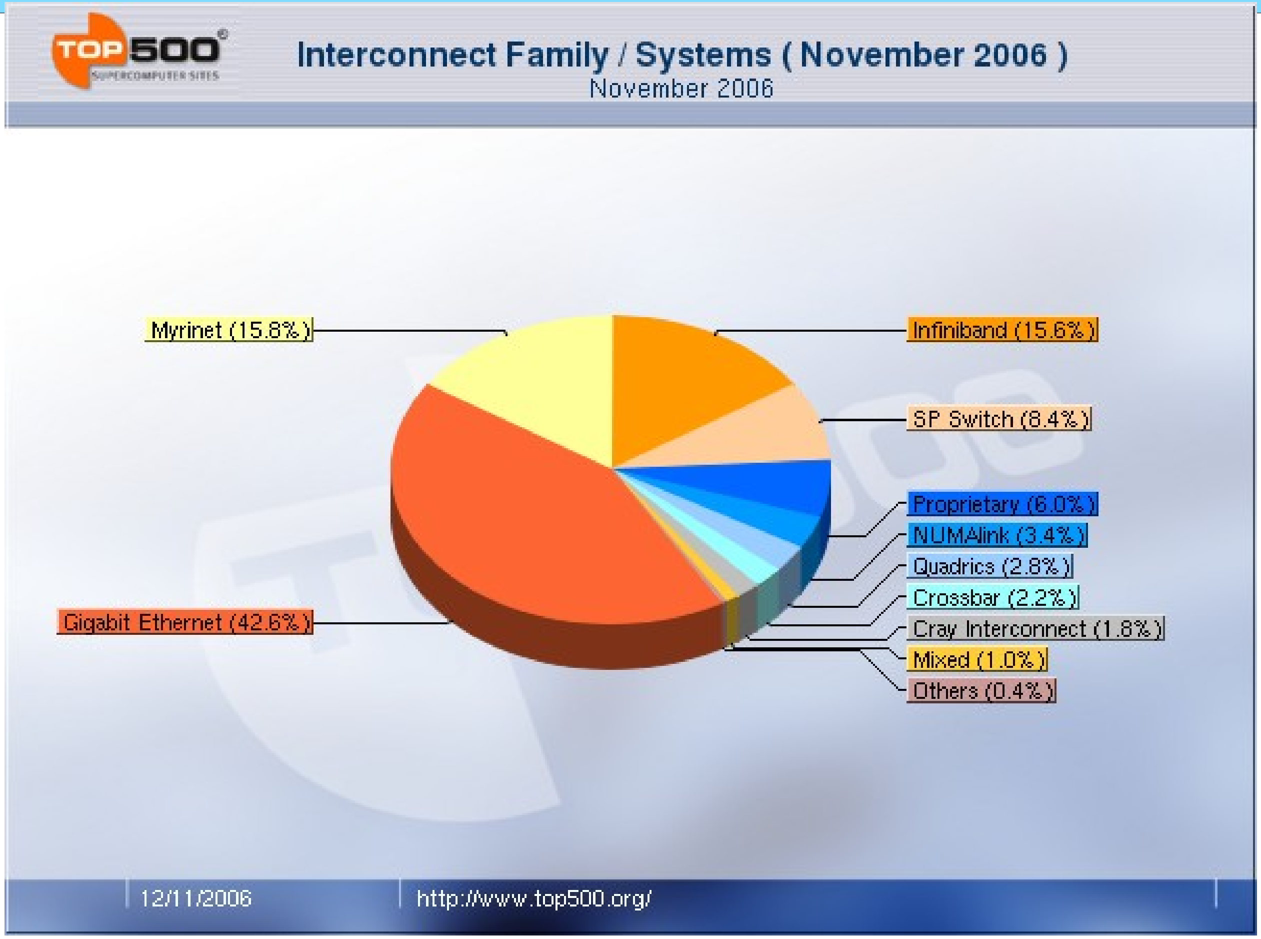
Elements of a Beowulf cluster (1)

The Beowulf definition is commodity machines connected by a private cluster network running an open source software infrastructure for scalable performance computing



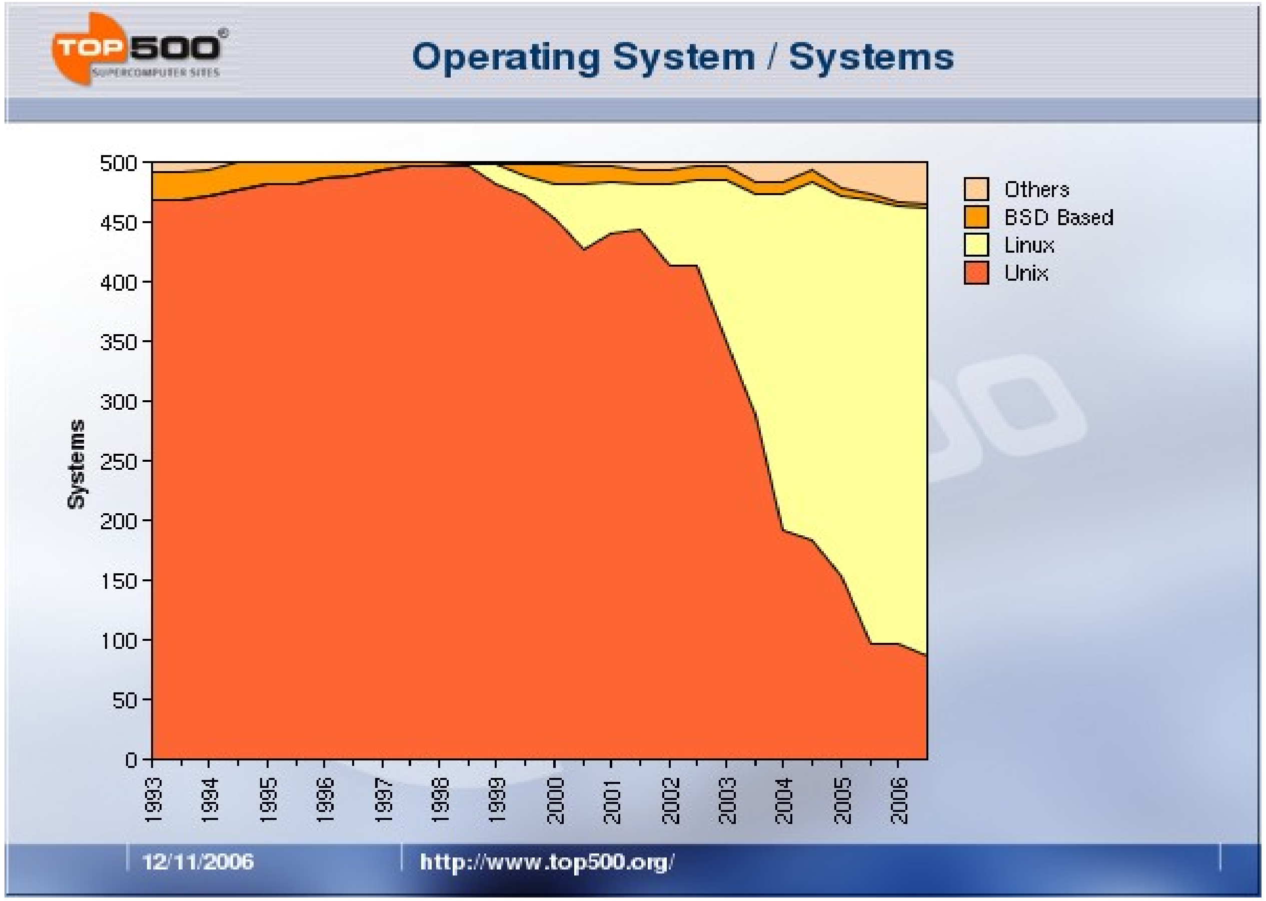
Elements of a Beowulf cluster (2)

The Beowulf definition is commodity machines connected by a private cluster network running an open source software infrastructure for scalable performance computing



Elements of a Beowulf cluster (3)

The Beowulf definition is commodity machines connected by a private cluster network running an open source software infrastructure for scalable performance computing



Elements of an HPC infrastructure

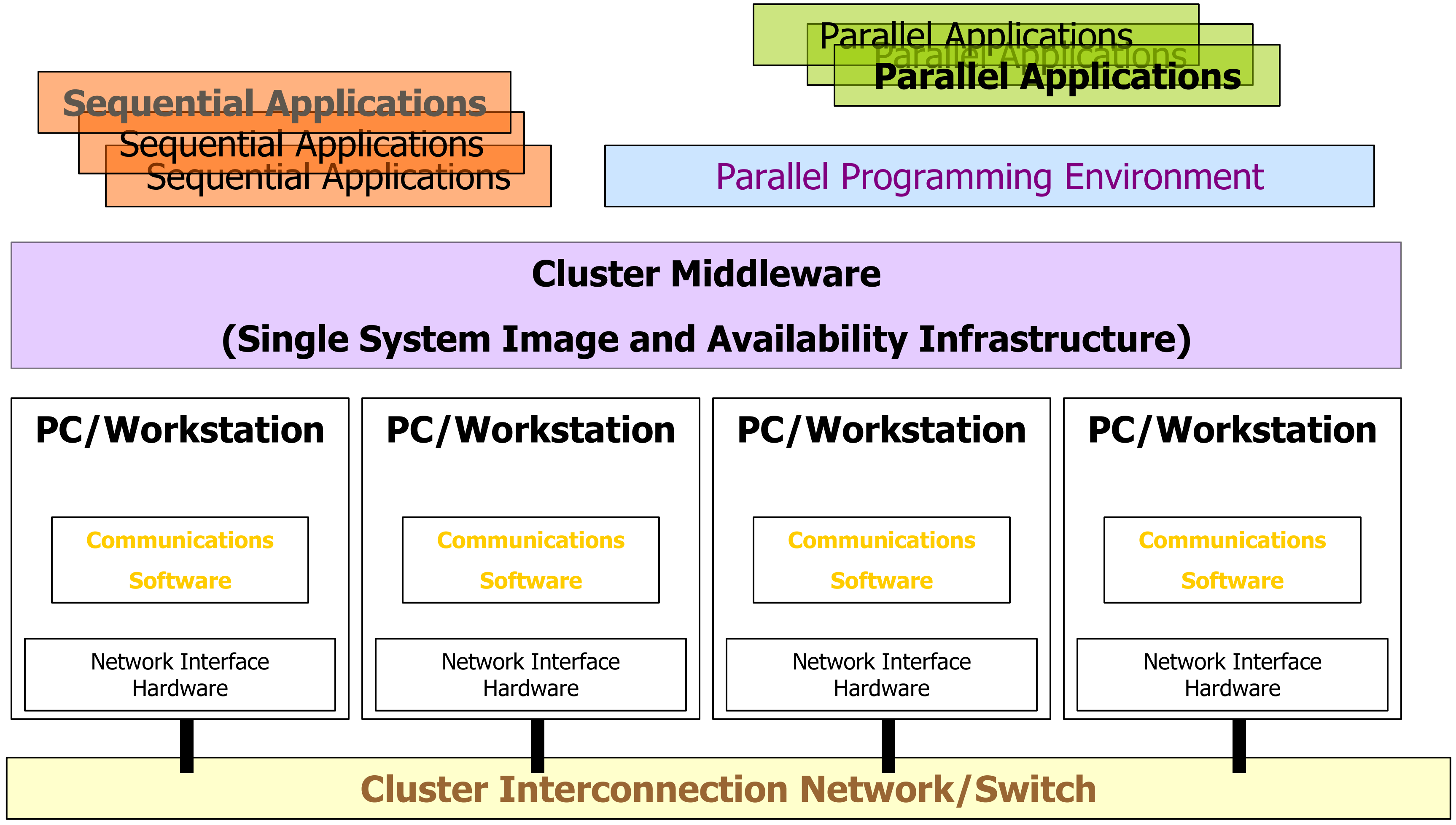
- Hardware
  - The basic bricks
- Software
  - To make hardware usable
- People
  - installers/sys adm. /planners/ users etc..
- Problems to be solved
  - Any action in building such an infrastructure should be motivated by real needs

Building your own HPC infrastructure

- HPC infrastructure was extremely expensive a few years ago
  - based on supercomputers
- Open source software + commodity off the shelf hardware provides now tools to build low cost HPC infrastructure
  - based on clusters

GREAT CHANCE FOR
LOW BUDGET INSTITUTIONS

Cluster Computer Architecture



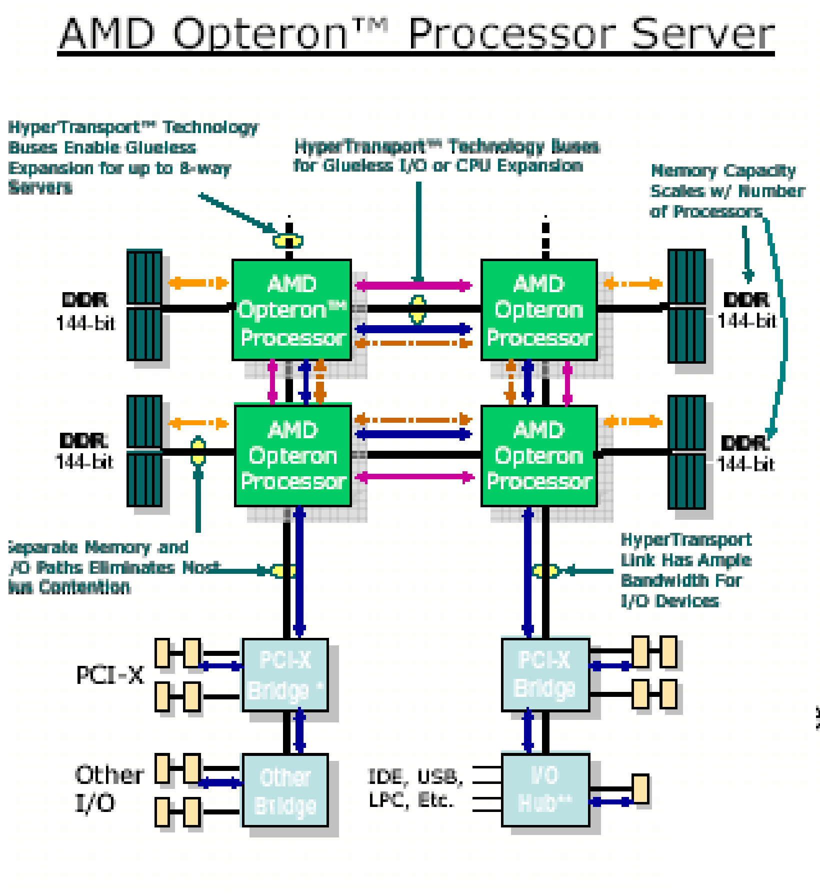
Hardware bricks for clusters

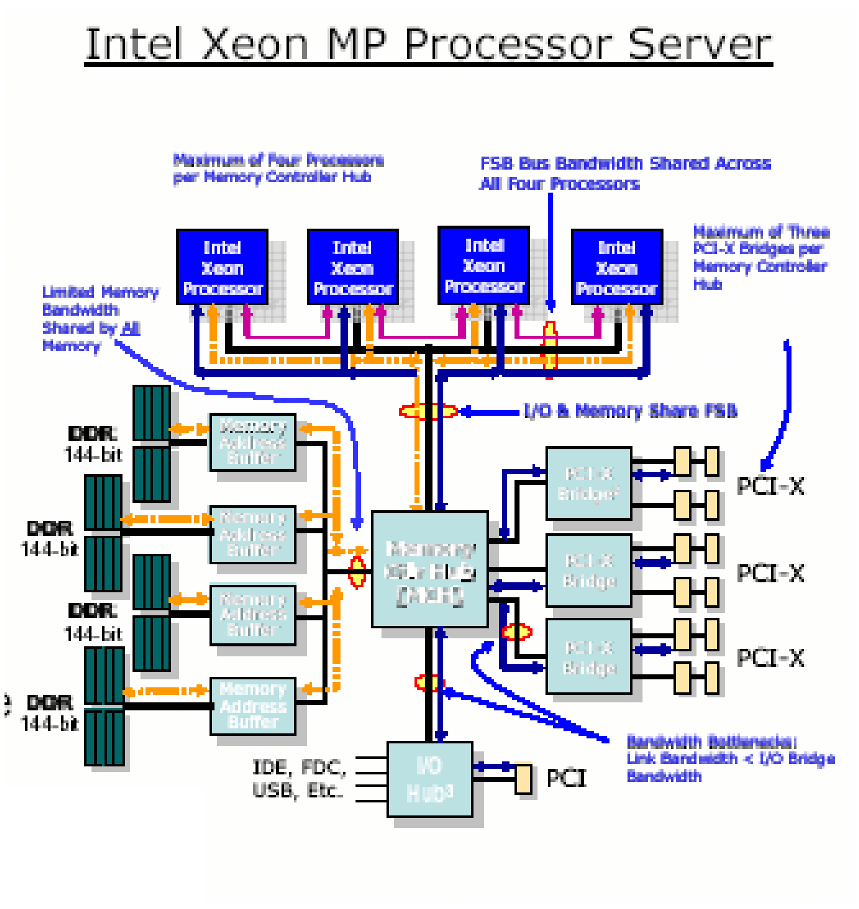
- Commodity 64 bit CPUs:
  - AMD
  - INTEL
- Networks
  - Standard (commodity)
    - Gigabit
  - High speed (not really commodities)
    - Myrinet
    - Qsnet
    - Scali/Dolphin
    - Infiniband

Processors at 64 bit

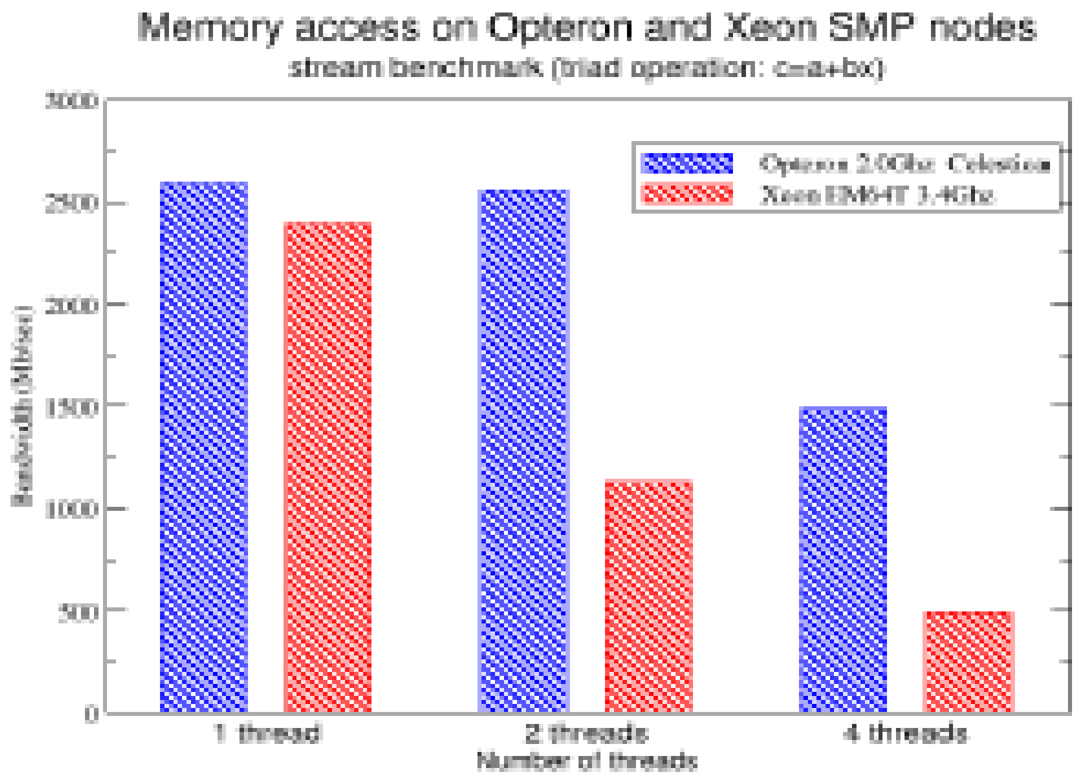
- What does it mean 64 bit ?
  - Memory address space and disk files:
    $2^{64-1} >> 2^{32-1}$
  - I/O bandwidth: 64 =32 x2 double the size !
  - Nothing to do with precision !

AMD/Intel XEON comparison


AMD Opteron™ Processor Server
HyperTransport™ Technology Buses Enable Glueless Expansion for up to 8-way Servers
HyperTransport™ Technology Buses for Glueless I/O or CPU Expansion
Memory Capacity Scales w/ Number of Processors
DDR 144-bit
AMD Opteron™ Processor
AMD Opteron Processor
Separate Memory and I/O Paths Eliminates Most Bus Contention
HyperTransport Link Has Ample Bandwidth For I/O Devices
PCI-X
PCI-X Bridge *
PCI-X Bridge
Other I/O
Other Bridge
IDE, USB, LPC, Etc.
I/O Hub**


Intel Xeon MP Processor Server
Maximum of Four Processors per Memory Controller Hub
FSB Bus Bandwidth Shared Across All Four Processors
Intel Xeon Processor
Maximum of Three PCI-X Bridges per Memory Controller Hub
Limited Memory Bandwidth Shared by All Memory
I/O & Memory Share FSB
DDR 144-bit
Memory Address Buffer
Memory Controller Hub (MCH)
PCI-X Bridge
PCI-X
Bandwidth Bottlenecks: Link Bandwidth < I/O Bridge Bandwidth
IDE, FDC, USB, Etc.
I/O Hub³
PCI

AMD/Intel XEON comparison

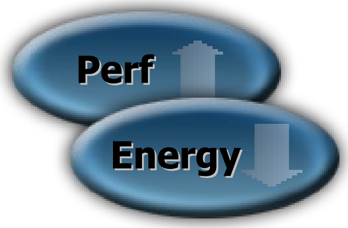


Which hardware for HPC nodes ?

- MULTIPROCESSORS machines
  - dual processor quite common/inexpensive
  - Quad processor available..

Multiple, externally visible processors on a single die where the processors have independent control-flow, separate internal state and no critical resource sharing

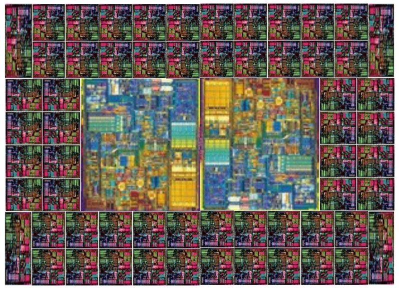
- MULTICORE !
  - Quad core for AMD
  - quad core for Intel

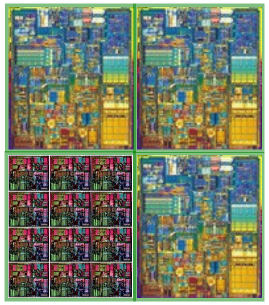


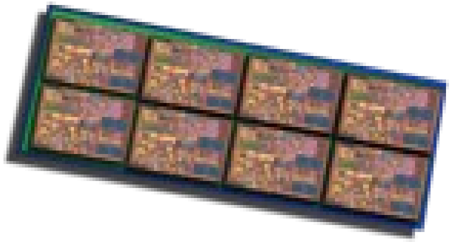
Evolutionary configurable architecture

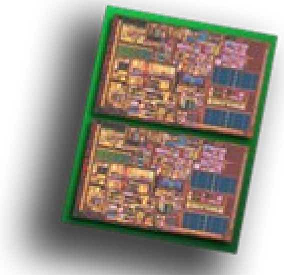
Large, Scalar cores for high single-thread performance

Scalar plus many core for highly threaded workloads





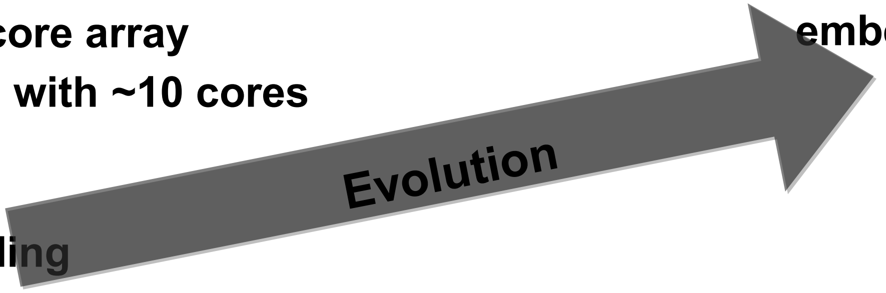




Many-core array
- CMP with 10s-100s low power cores
- Scalar cores
- Capable of TFLOPS+
- Full System-on-Chip
- Servers, workstations, embedded…

Multi-core array
- CMP with ~10 cores

Dual core
- Symmetric multithreading



Single core vs dual core:

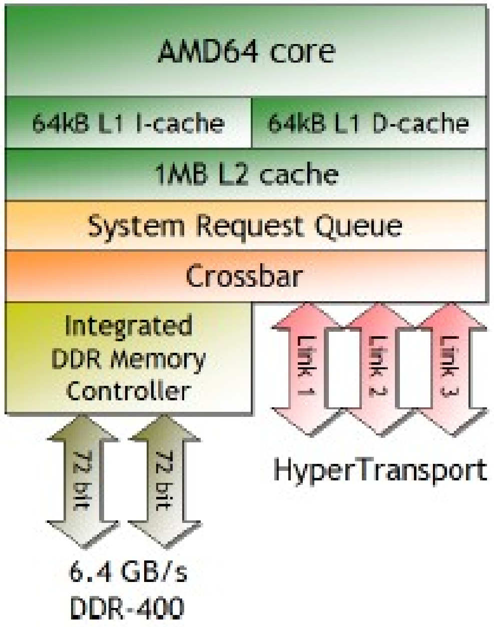


Figure 1: Single core AMD64 block diagram

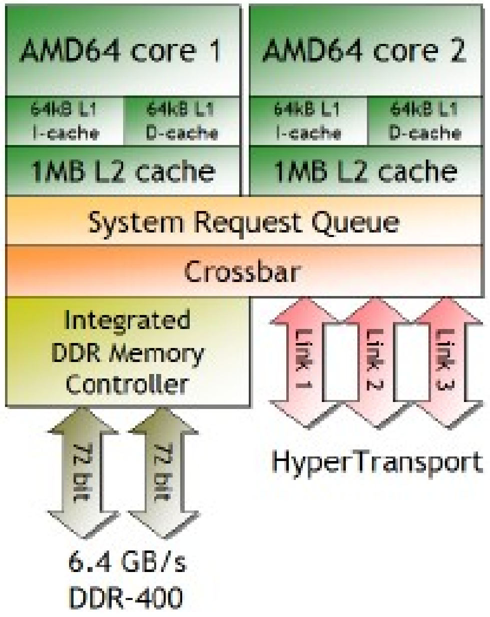
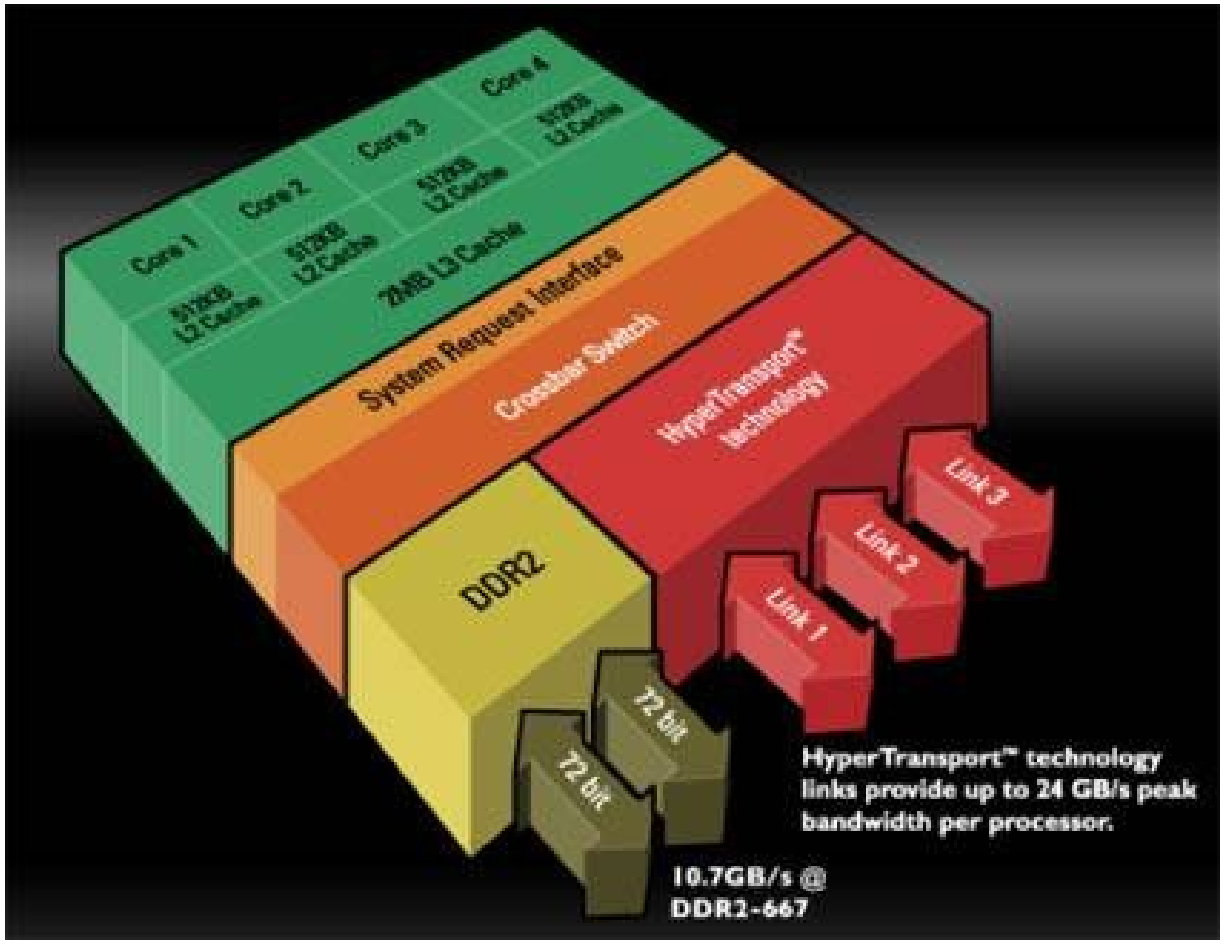
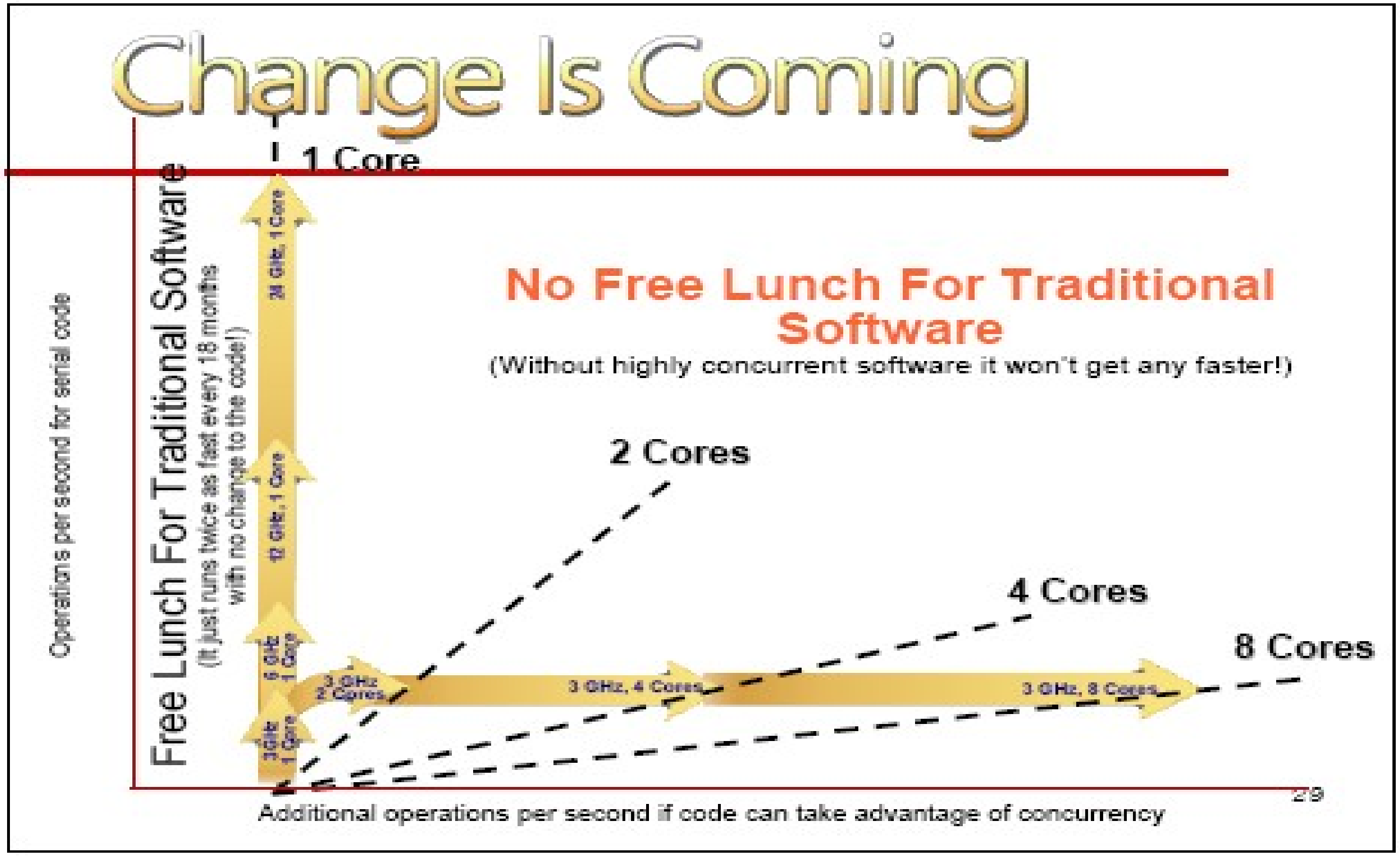


Figure 2: Dual core AMD64 block diagram

Barcelona quad core architecture



single Core VS Multiple core (from J.Dongarra talk)



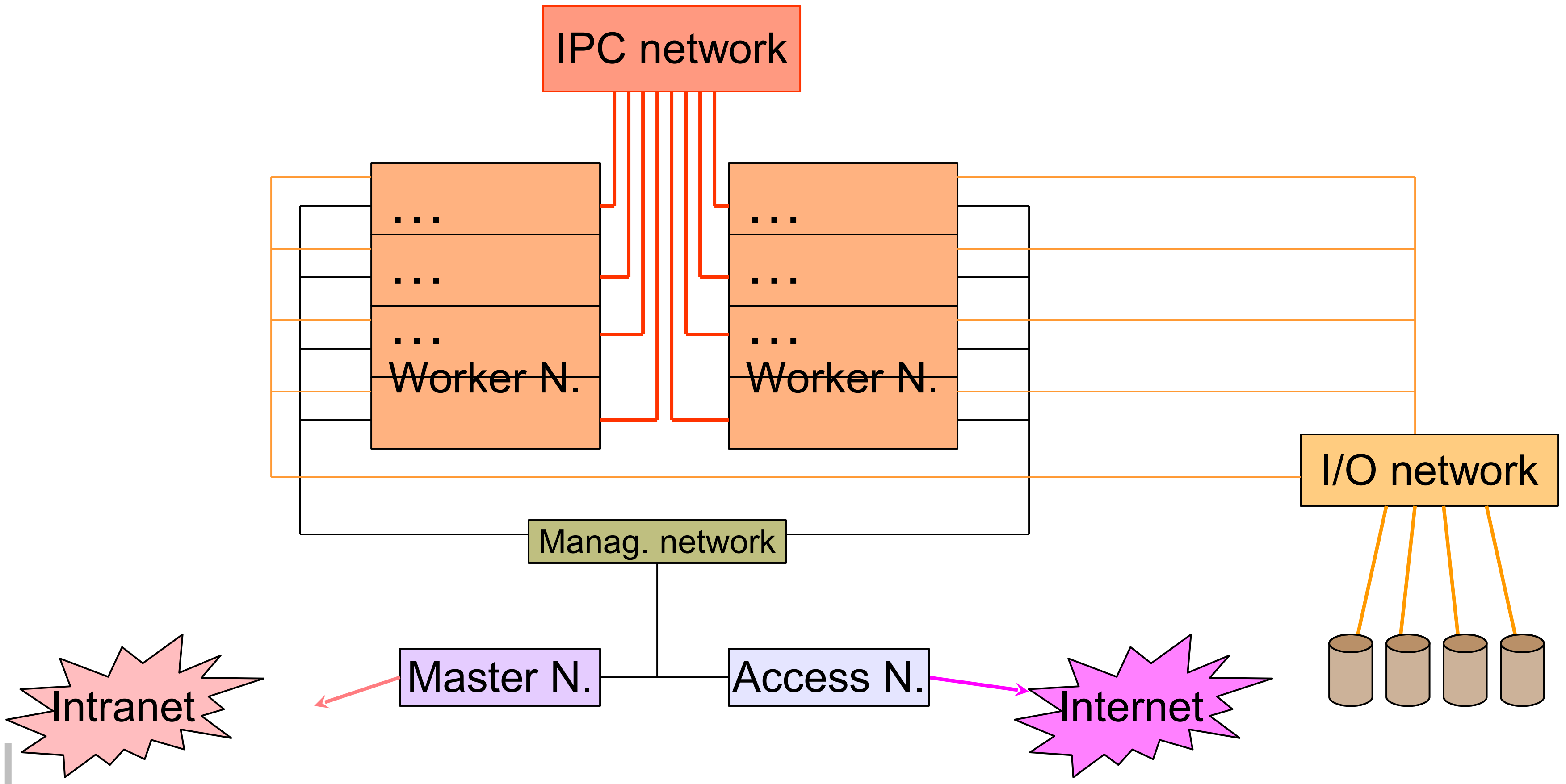
Networks

- Standard:
  - Fast Ethernet
  - Gigabit Ethernet
- High Speed Network
  - SCI (Dolphin)
  - Qsnet
  - Myrinet
  - Infiniband
  - 10Gigabit

Which networks for your cluster ?

- Difficult choice:
  - Which kind of cluster (HTC or HPC) ?
  - Which kind of application ?
    - Serial/Parallel
    - Parallel loosely coupled / tightly coupled
    - Latency or bandwidth dominated ?
  - Budget considerations
  - I/O considerations

HPC cluster structure



Luxury clusters: 3 networks

- HIGH SPEED NETWORK
  - parallel computation
    - low latency /high bandwidth
    - Usual choiches: Myrinet/SCI/Infiniband...
- I/O NETWORK
  - I/O requests (NFS and/or parallel FS)
    - latency not fundamental/ good bandwidth
    - GIGABIT is ok
- Management network
  - management traffic
    - any standard network (fast ethernet OK)

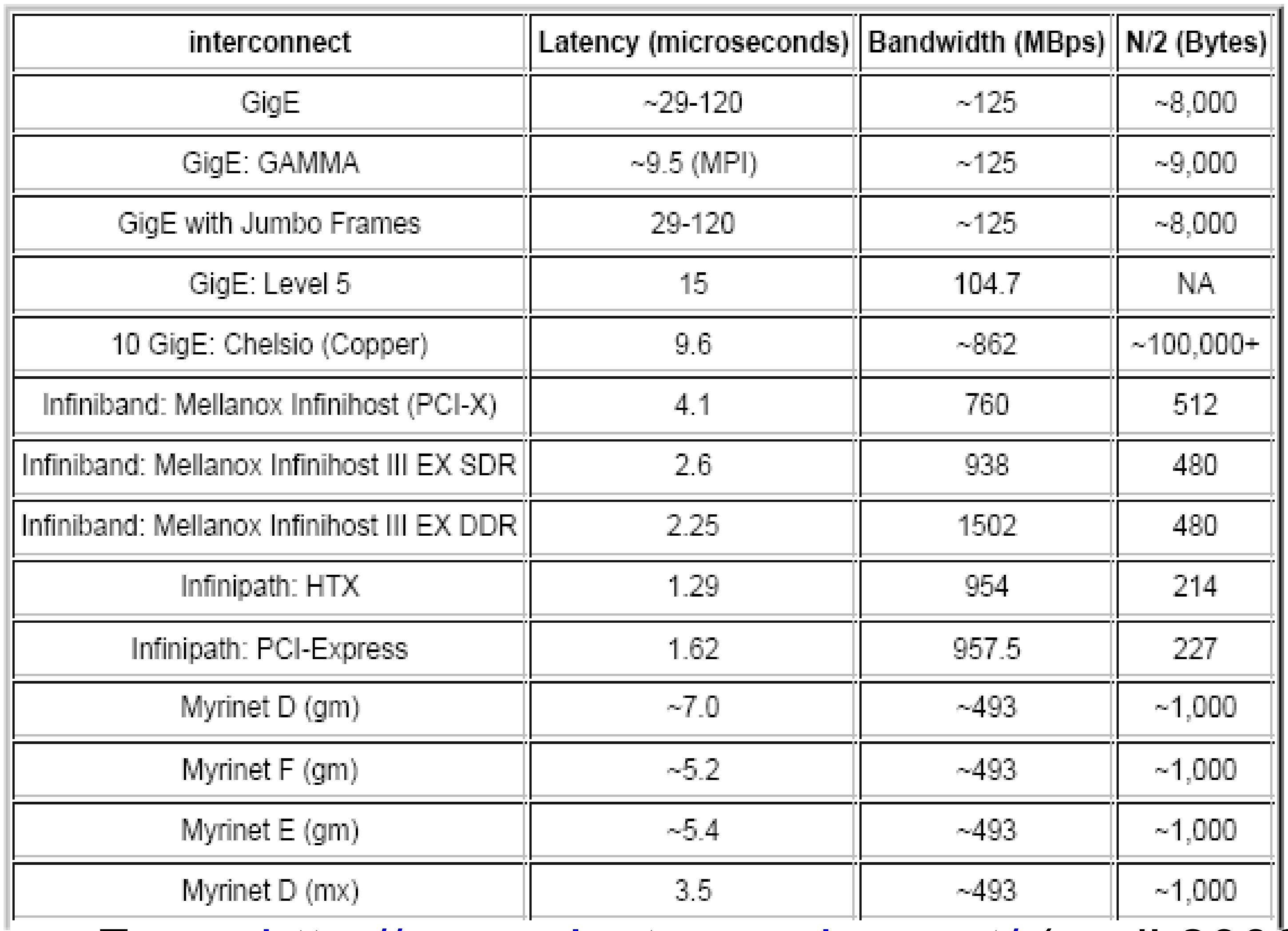
Network Considerations

- In the past 5 years the speed of the interconnects commonly found in clusters has improved by a factor of 20!
- While 1-3 years ago the PCs commonly available were unable to make full use of the available bandwidth, today’s systems are demonstrating some impressive performance
- There are now several interconnect technologies that each offer advantages in certain situations.

high speed network considerations

- In general the compute/communication ratio in a parallel program remains fairly constant.
- So as the computational power increases the network speed must also be increased.
- Another formulation of our beloved Ahmdal law…

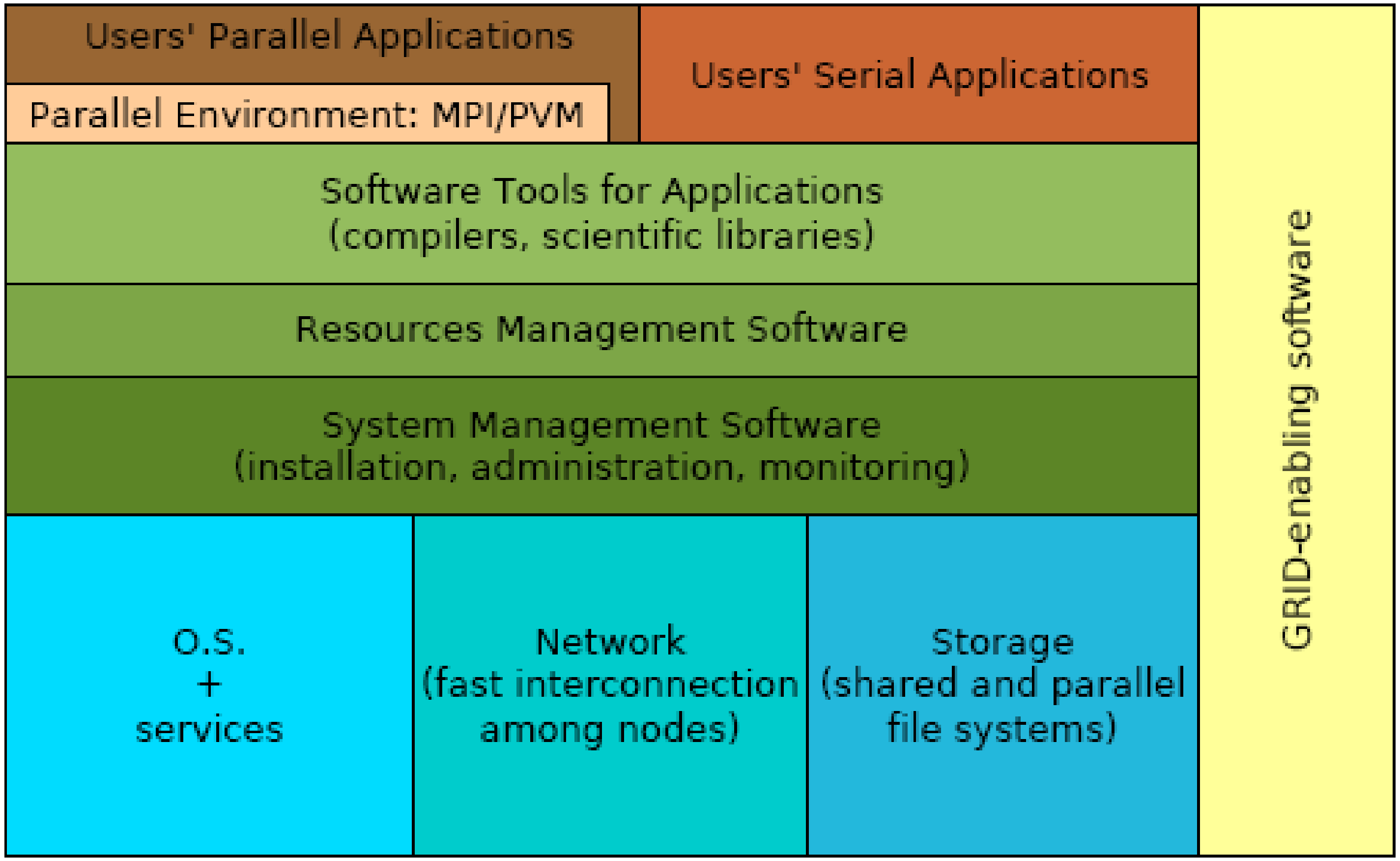
Network performance



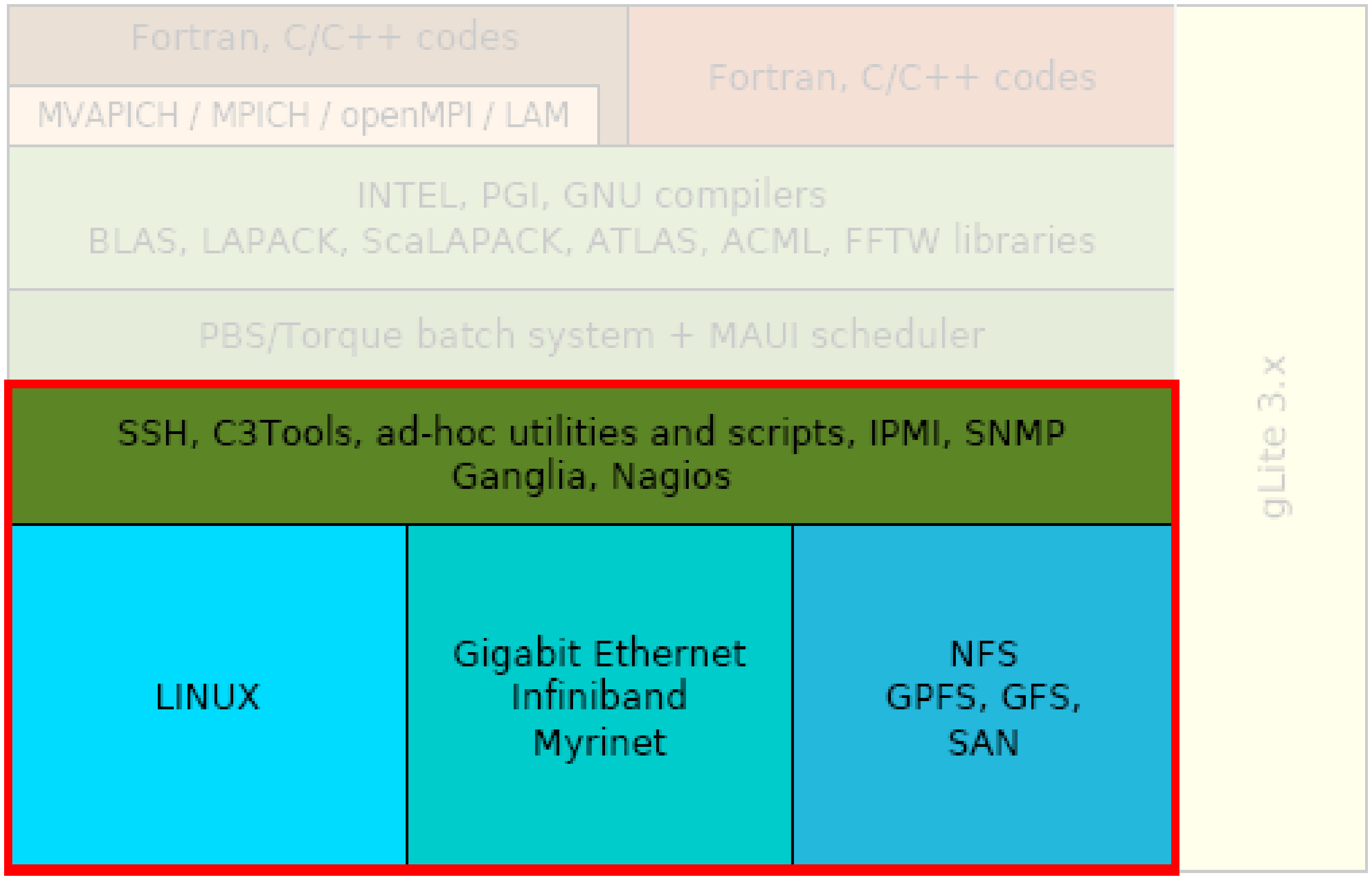
interconnect	Latency (microseconds)	Bandwidth (MBps)	N/2 (Bytes)
GigE	~29-120	~125	~8,000
GigE: GAMMA	~9.5 (MPI)	~125	~9,000
GigE with Jumbo Frames	29-120	~125	~8,000
GigE: Level 5	15	104.7	NA
10 GigE: Chelsio (Copper)	9.6	~862	~100,000+
Infiniband: Mellanox Infinihost (PCI-X)	4.1	760	512
Infiniband: Mellanox Infinihost III EX SDR	2.6	938	480
Infiniband: Mellanox Infinihost III EX DDR	2.25	1502	480
Infinipath: HTX	1.29	954	214
Infinipath: PCI-Express	1.62	957.5	227
Myrinet D (gm)	~7.0	~493	~1,000
Myrinet F (gm)	~5.2	~493	~1,000
Myrinet E (gm)	~5.4	~493	~1,000
Myrinet D (mx)	3.5	~493	~1,000

From: http://www.clustermonkey.net/ (april 2006)

Linuux Cluster: the software stacks


Users' Parallel Applications
Parallel Environment: MPI/PVM
Users' Serial Applications
Software Tools for Applications (compilers, scientific libraries)
Resources Management Software
System Management Software (installation, administration, monitoring)
O.S. + services
Network (fast interconnection among nodes)
Storage (shared and parallel file systems)
GRID-enabling software

Linux Cluster: the sys. Adm. stacks


Fortran, C/C++ codes
MVAPICH / MPICH / openMPI / LAM
Fortran, C/C++ codes
INTEL, PGI, GNU compilers
BLAS, LAPACK, ScaLAPACK, ATLAS, ACML, FFTW libraries
PBS/Torque batch system + MAUI scheduler
SSH, C3Tools, ad-hoc utilities and scripts, IPMI, SNMP
Ganglia, Nagios
LINUX
Gigabit Ethernet
Infiniband
Myrinet
NFS
GPFS, GFS,
SAN
gLite 3.x

Operating System: Gnu/Linux

- Linux Kernel (http://www.kernel.org)
  - Open-Source/ freeware
- Features:
  - The /proc file system
  - Loadable kernel modules
  - Virtual consoles
  - Package management
  - Many distribution to choose

Why Linux ?

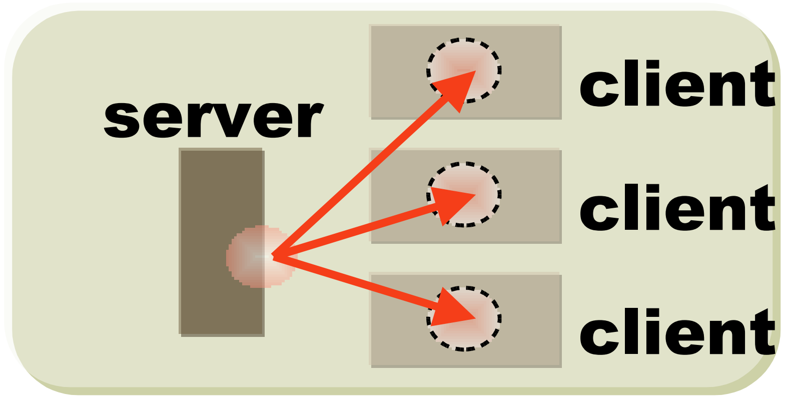
- Access to cheap hardware
- Access to Source code is needed to implement desired features.
- Availability of software
- Access to cheap graduate students
- Access to large community
  - response speed from community sometime much better then vendor/support ones.
- open source/ free software: no license Issues.
- Availability of Scientific Tools/Resources.

Middleware Design Goals

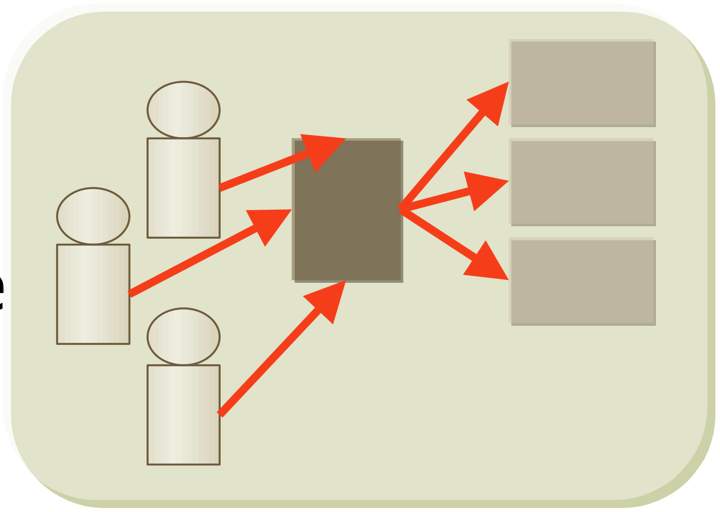
- Complete Transparency (Manageability):
  - Lets the see a single cluster system..
    - Single entry point, ftp, ssh, software loading...
- Scalable Performance:
  - Easy growth of cluster
    - no change of API & automatic load distribution.
- Enhanced Availability:
  - Automatic Recovery from failures
    - Employ checkpointing & fault tolerant technologies
  - Handle consistency of data when replicated..

Cluster middleware: beowulf approach

- Administration software:
  - NFS
  - user accounts
  - NTP



- Resource management and scheduling software (RMS)
  - Process distribution
  - Load balance
  - Job scheduling of multiple tasks



CLUSTER MANAGEMENT
Administration Tools

- Requirements:
  - cluster-wide command execution
  - cluster-wide file distribution and gathering
  - password-less environment
  - must be simple, efficient, easy to use for CLI addicted

Cluster Management Toolkits

- Are generally made of an ensemble of already available software packages thought for specific tasks, but configured to operate together, plus some add-ons.
- Sometimes limited by rigid and not customizable configurations, often bound to some specific LINUX distribution and version. May depend on vendors' hardware.
- Free and Open
  - OSCAR (Open Source Cluster Application Resources)
  - NPACI Rocks
  - xCAT (eXtreme Cluster Administration Toolkit)
  - Warewulf ....
- Commercial
  - Scyld Beowulf
  - IBM, HP, SUN and other vendors' Management Software...

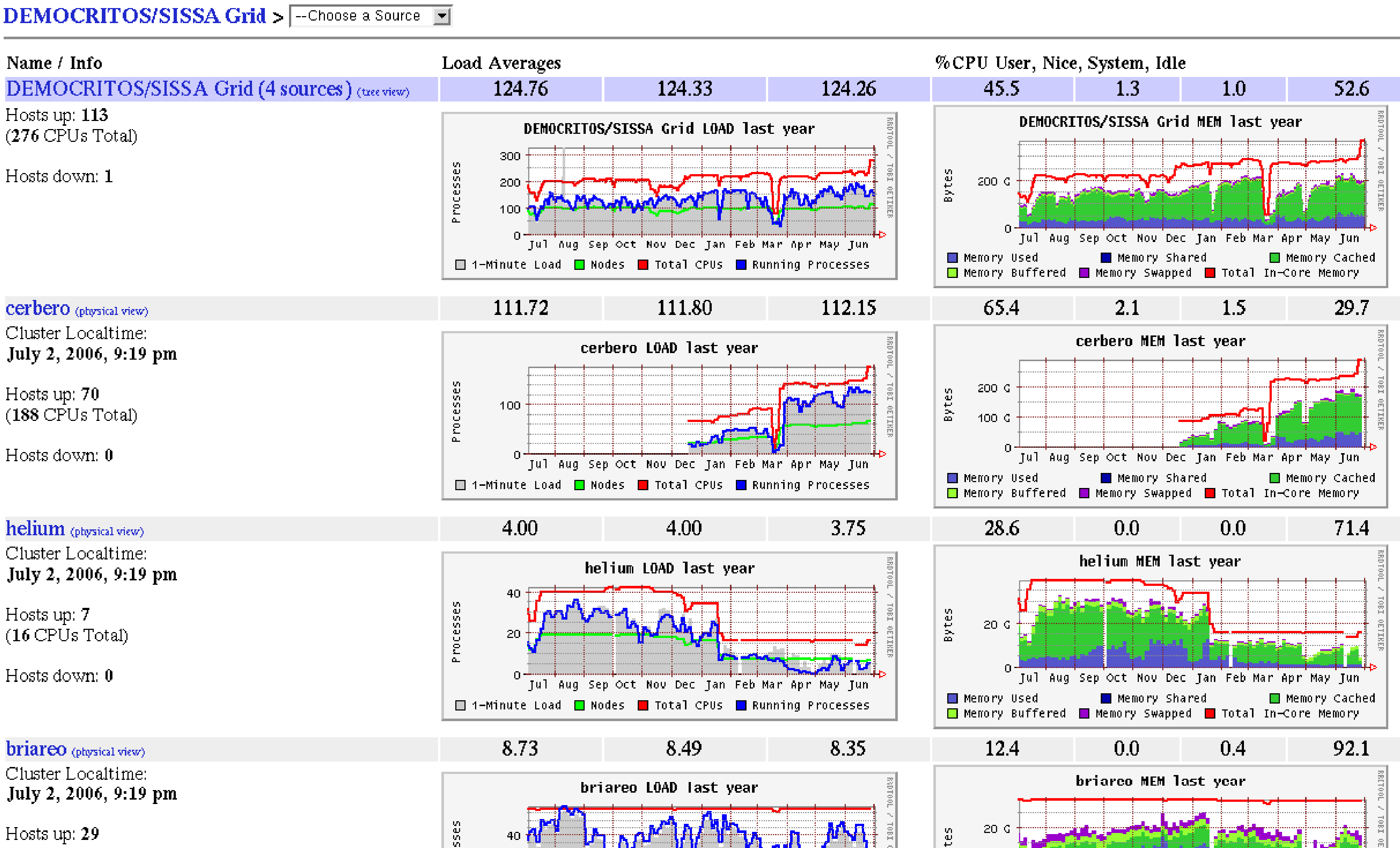
CLUSTER MANAGEMENT
Administration Tools

- C3 tools – The Cluster Command and Control tool suite
  - allows configurable clusters and subsets of machines
  - concurrently execution of commands
  - supplies many utilities
    - cexec (parallel execution of standard commands on all cluster nodes)
    - cexecs (as the above but serial execution, useful for troubleshooting and debugging)
    - cpush (distribute files or directories to all cluster nodes)
    - cget (retrieves files or directory from all cluster nodes)
    - ... and many more
- **PDSH – Parallel Distributed SHell**
  - same features as C3 tools, few utilities
- **And many others...**

CLUSTER MANAGEMENT: monitoring tools:

- Ad-hoc scripts (BASH, PERL, ...) + cron
- Ganglia
  - excellent graphic tool
  - XML data representation
  - web-based interface for visualization
  - http://ganglia.sourceforge.net/
- Nagios
  - complex but can interact with other software
  - configurable alarms, SNMP, E-mail, SMS, ...
  - optional web interface
  - http://www.nagios.org/

Ganglia at work..



DEMOCRITOS/SISSA Grid > --Choose a Source

Name / Info	Load Averages			%CPU User, Nice, System, Idle			
DEMOCRITOS/SISSA Grid (4 sources) (tree view)	124.76	124.33	124.26	45.5	1.3	1.0	52.6
Hosts up: **113** (**276** CPUs Total) Hosts down: **1**							
cerbero (physical view)	111.72	111.80	112.15	65.4	2.1	1.5	29.7
Cluster Localtime: **July 2, 2006, 9:19 pm** Hosts up: **70** (**188** CPUs Total) Hosts down: **0**							
helium (physical view)	4.00	4.00	3.75	28.6	0.0	0.0	71.4
Cluster Localtime: **July 2, 2006, 9:19 pm** Hosts up: **7** (**16** CPUs Total) Hosts down: **0**							
briareo (physical view)	8.73	8.49	8.35	12.4	0.0	0.4	92.1
Cluster Localtime: **July 2, 2006, 9:19 pm** Hosts up: **29**							

Local Resource Management Systems..

- a.k.a "Batch System"
  - Some pieces of software control available resources
  - Some other pieces of software decide which application to execute based on available resources
  - Some other pieces of software are devoted to actually execute applications
- Please note:
  - Batch does not necessarily mean "delayed"
  - Batch-interactive sessions are possible
  - Time sharing is possible with most systems

Resource Management and Access Control

- The batch system knows who will be allowed to run applications on which nodes
  - some mechanisms can be put in place to allow access to nodes only to these legitimate users
  - Usually batch system’s prologue and epilogue programs can be used for the purpose.

LRMS for Linux Clusters

- Several batch queuing systems available for Linux-based clusters.
- Most commonly used:
  - Condor (http://www.cs.wisc.edu/condor)
  - SGE (http://www.sun.com/sge)
  - LSF (http://www.platform.com, -- commercial)
  - Portable Batch System
    - PBSPro (commercial)
    - OpenPBS
  - Torque (successor of openpbs)

Cluster Pro&Cons

- Pro:
  - Price/performance when compared with a dedicated parallel supercomputer
  - Great opportunity for low budget institution
  - Flexibility: many ad hoc solution for different problems..
  - Open Technology
    - What you learn in this business can be used everywhere..
- Cons:
  - It is hard to build and operate medium and large cluster
    - Large collection of software that are not “talk to each other”
  - Lot of expertise needed (no plug and play yet)
  - How to use cluster power efficiently

Which cluster do I need ?

- Which applications ?
  - Parallel
    - Tightly coupled
    - Loosely coupled
  - Serial
    - Memory / I/O requirements
- Which user's community ?
  - Large /Small
  - Homogeneous /heterogeneous
- Understand your computational problem before buying/building a cluster !
- Run your own benchmarks before buying/building a cluster !

Which architectures in your infrastructure ?

- Parallel computing:
  - single systems with many processors working on same problem
- Distributed computing:
  - many systems loosely coupled by a scheduler to work on related problems
- Grid Computing:
  - many systems tightly coupled by software, perhaps geographically distributed, to work together on single problems or on related problems

Capability vs Capacity Computing

- Capability computing: the system is employed for one or a few programs for which no alternative is readily available in terms of computational capabilities
  - typical cluster usage
    - small research groups using a few bunch of scientific application
- Capacity computing: a system is employed to the full by using the most of its available cycles by many, often very demanding, applications and users.
  - typical computer center usage:
    - still clusters can be useful: they required much more work/tuning to fulfill all the requirements

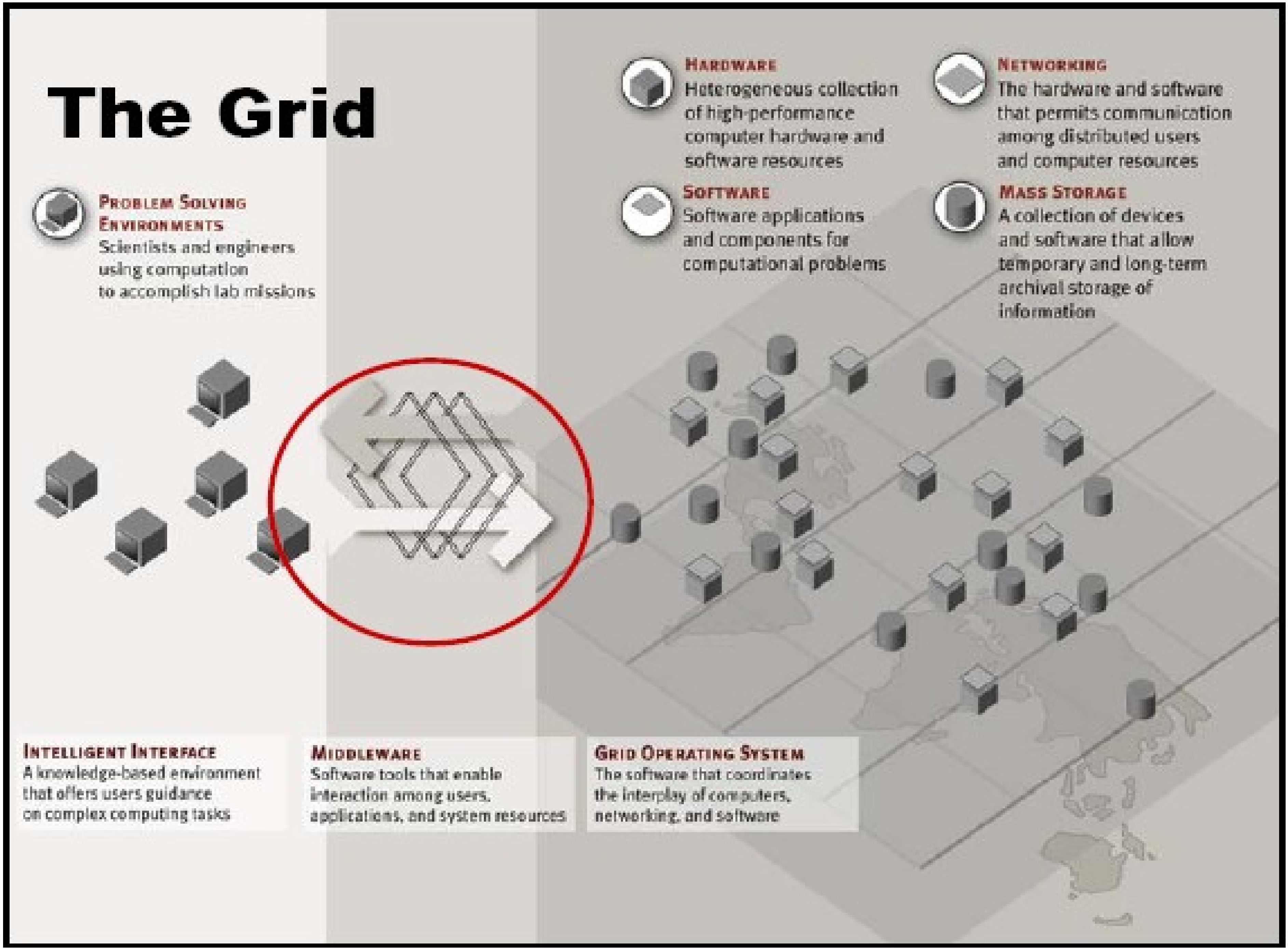
Agenda

- Introduction: what is e-science ?
- High Performance Computing:
  - introduction/ concepts /definitions
- Understanding parallel programming: some ideas
  - Speedup: the effectiveness of parallelism
  - Limits to parallel performance
  - Modern Serial processor and parallelism
- Parallel Machines
- Clusters:
  - definition and some other funny things
- Grid and all the rest
- Wrap-up

Why the GRID?

- Motivation: When communication is close to free we should not be restricted to local resources when solving problems.
- A Grid Infrastructure built on the Internet and the Web to enable and exploit large scale sharing of resources
- It should provides Scalable Secure Reliable mechanisms for discovery and for remote access of resources.

The Grid



Hardware
Heterogeneous collection of high-performance computer hardware and software resources

Networking
The hardware and software that permits communication among distributed users and computer resources

Problem Solving Environments
Scientists and engineers using computation to accomplish lab missions

Software
Software applications and components for computational problems

Mass Storage
A collection of devices and software that allow temporary and long-term archival storage of information

Intelligent Interface
A knowledge-based environment that offers users guidance on complex computing tasks

Middleware
Software tools that enable interaction among users, applications, and system resources

Grid Operating System
The software that coordinates the interplay of computers, networking, and software

Grids vs. HPC

- Not an "either/or" question
  - Each addresses different needs
  - Each are part of an integrated solution
- Grid strengths
  - Coupling necessarily distributed resources instruments, software, hardware, archives, and people
  - Eliminating time and space barriers
  - remote resource access and capacity computing
- Grids are not a cheap substitute for capability
- HPC Highest performance computing strengths
  - Supporting foundational computations
  - terascale and petascale "nation scale" problems
  - Engaging tightly coupled computations and teams
- Key is easy access to resources in a transparent way

Different level of parallelism

- Within the core/cpu
  - Instruction level parallelism
- Within the node:
  - Threaded libraries/ openMP
- Within the cluster:
  - Message passing approach (i.e. MPI)
- Within the GRID
  - Message passing/ client/server approach..

Wrap-up

- HPC and GRID computing are now fundamental tools for scientific research
- HPC means parallel computing
- HPC experienced a great change in the last ten years: from custom machine to Beowulf clusters
- The challenge is now to build your own HPC infrastructure driven by real needs.
- HPC and GRID computing are not mutually exclusive but can be both used to address computational resources in a transparent way.

References

- www.democritos.it/hpc-wiki
- All the material will be made available for this workshop !