Improving the Usability of Clusters

William D. Gropp
www.mcs.anl.gov/~gropp

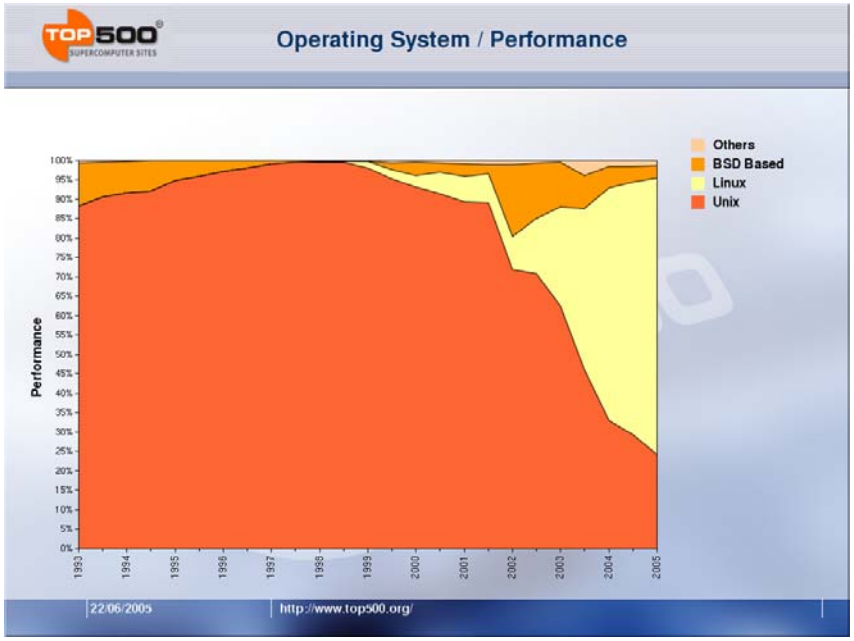
Argonne National Laboratory

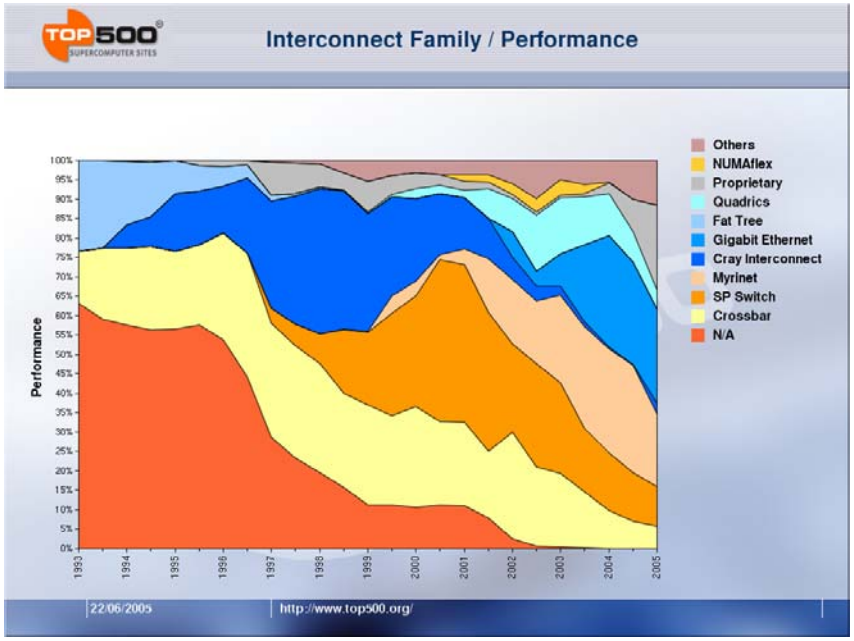
A U.S. Department of Energy
Office of Science Laboratory
Operated by The University of Chicago

A Usability Journey

- **Clusters are used by more people than ever**
  - But not everyone is enjoying the experience :)
- **Single node performance**
  - Performance is the reason for (many) clusters
- **Parallel performance**
  - Scalable operations
- **Managing 1000 nodes as one machine**
  - Getting away from O(p) files, steps, connections, …
- **Building components**
  - Commoditization of software
- **Challenges**
  - See you in Barcellona!

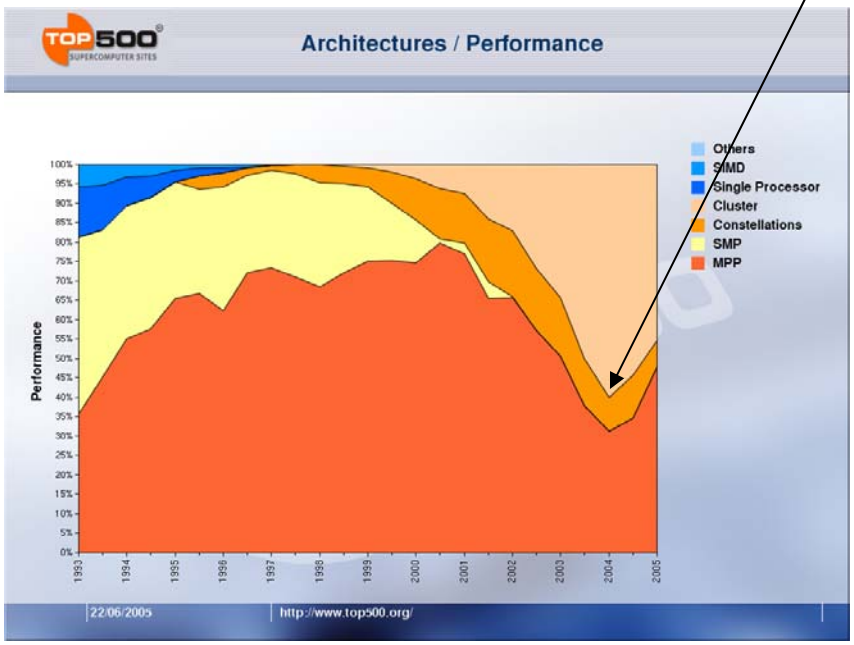
Clusters Dominate the Top500

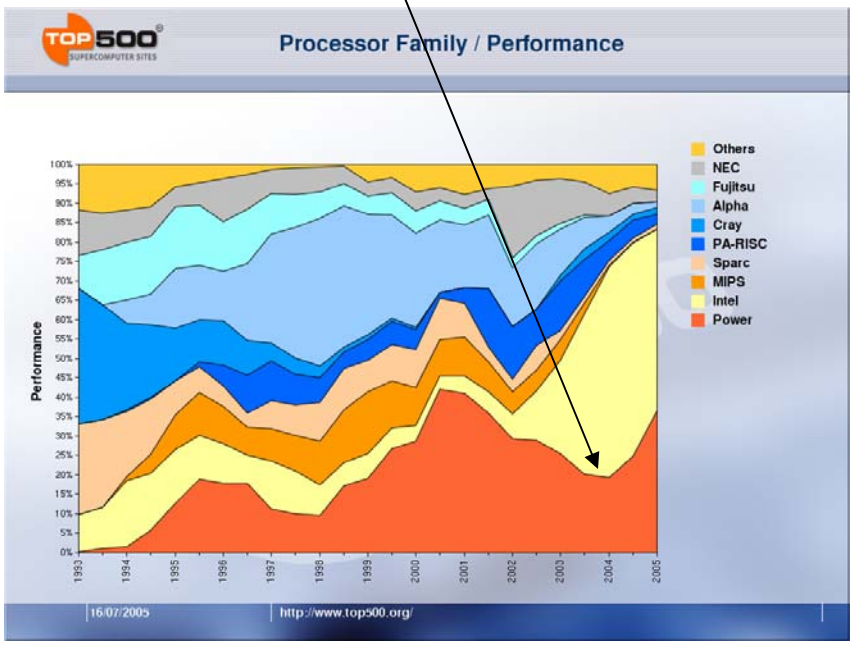




Clusters Face Competition on the Top500

IBM BlueGene Introduced

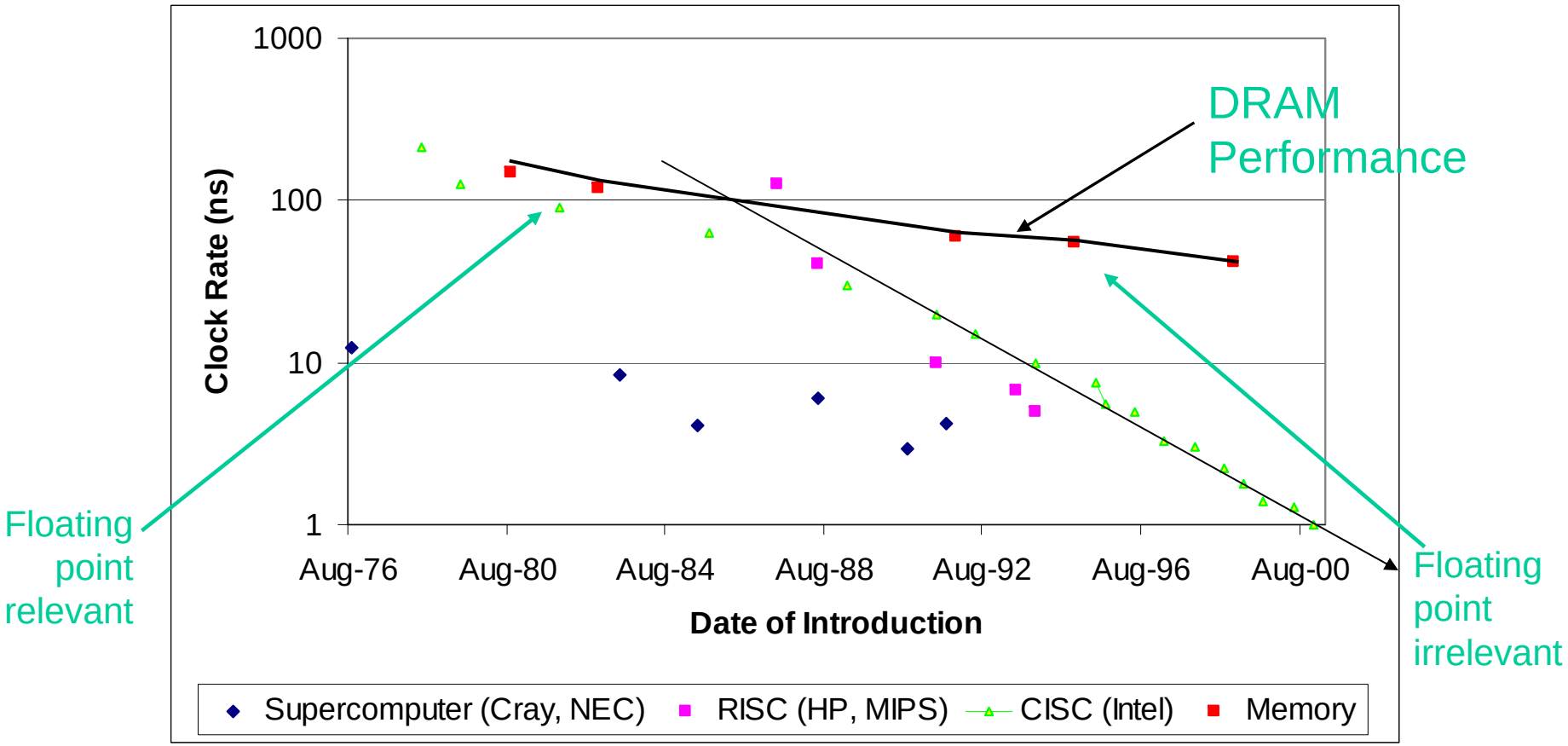




Single Node Performance

- **No one (well, almost no one) writes parallel programs or builds clusters for the fun of it**
  - Getting more performance is the reason for having a cluster
  - Applications may get as little as 5% of peak performance
  - Often blamed on “the parallel computer”
- **The fastest way to improve the performance of a parallel application is to speed up the single-node performance**
  - Of course, this decreases the scalability of the code

CPU and Memory Performance



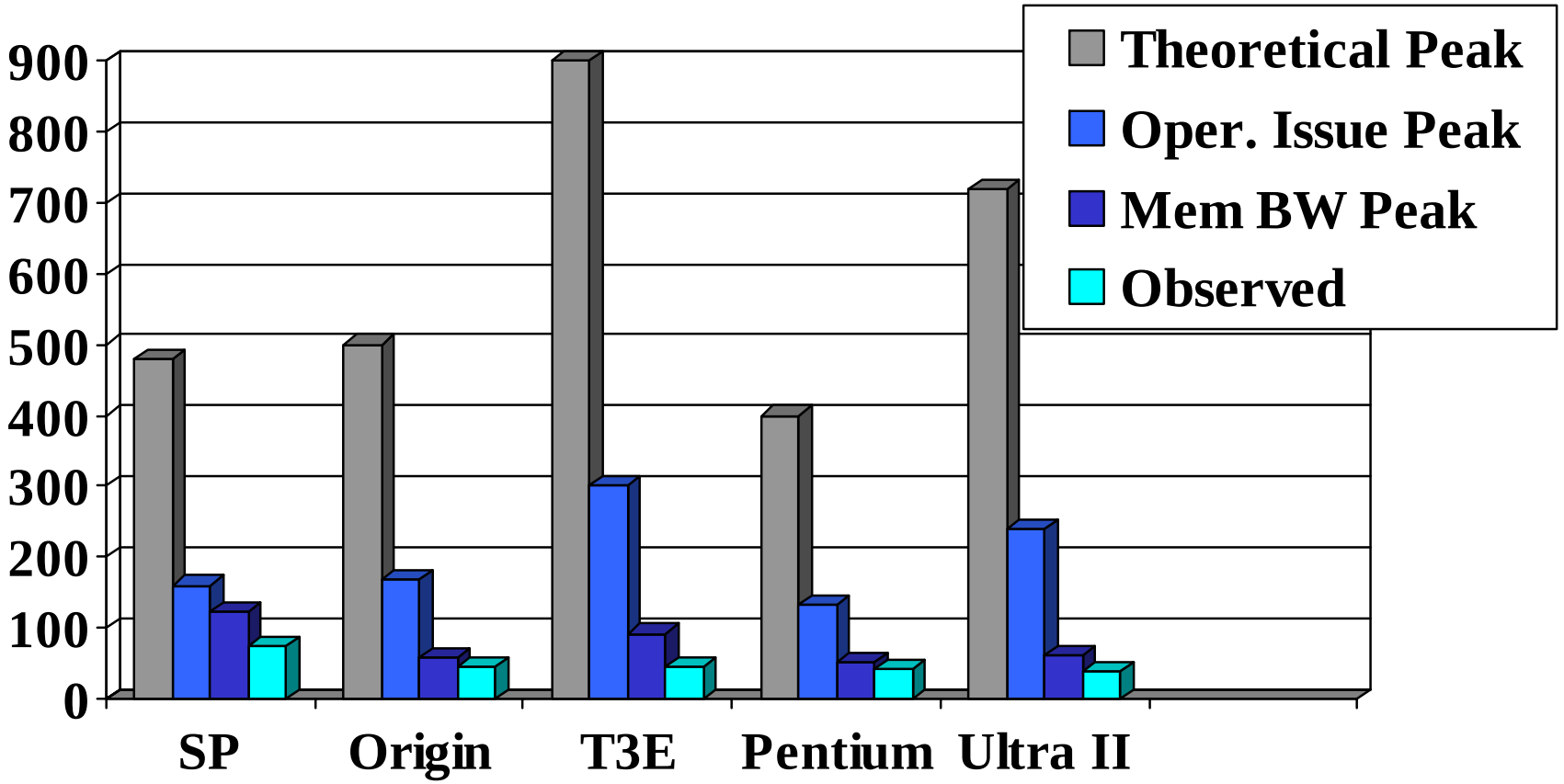
Consequences of Memory/CPU Performance Gap

- **Performance of an application may be limited by memory bandwidth or latency rather than CPU clock**
- **“Peak” performance determined by the resource that is operating at full speed for the algorithm**
  - Often memory system (STREAM numbers)
  - Sometime instruction rate/mix
- **For example, sparse matrix-vector operation performance is best estimated by using STREAM performance**
  - Note that STREAM performance is delivered performance to a Fortran or C program, not memory bus rate time width

Realistic Measures of Peak Performance

Sparse Matrix Vector Product

one vector, matrix size, m = 90,708, nonzero entries nz = 5,047,120



What About CPU-Bound Operations?

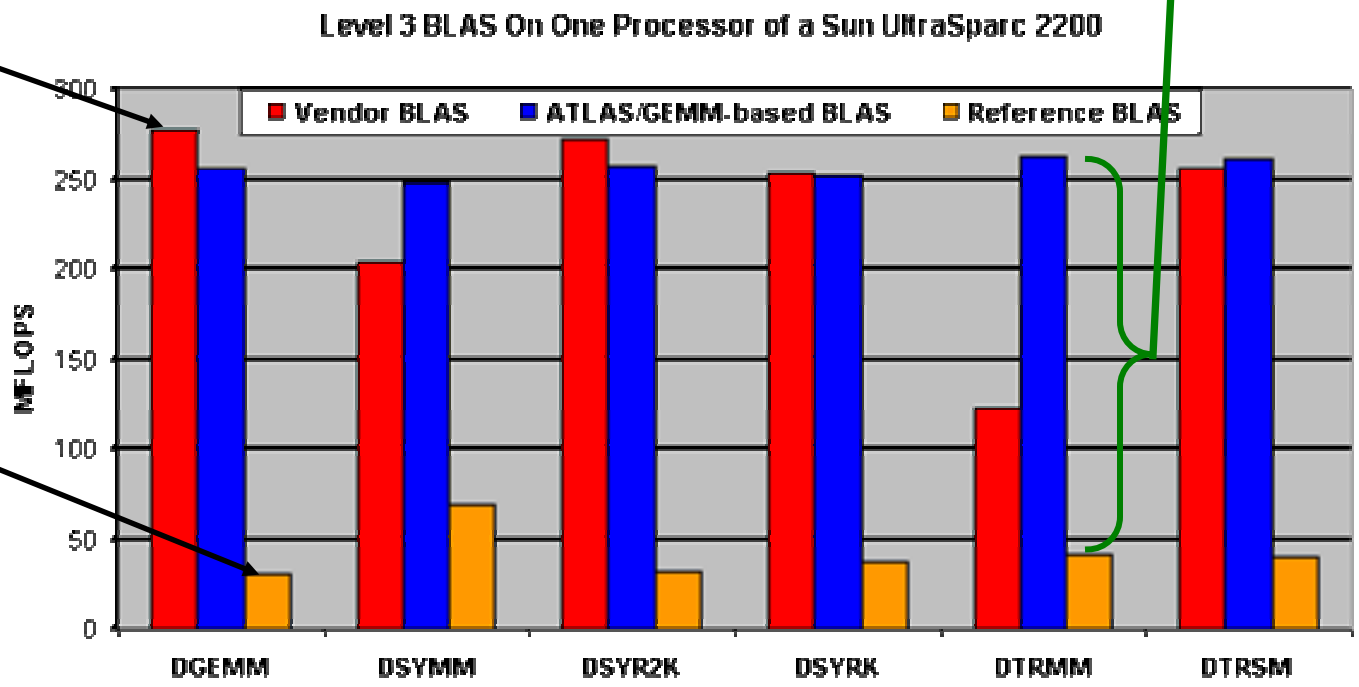
- **Dense Matrix-Matrix Product**
  - Most studied numerical program by compiler writers
  - Core of some important applications
  - More importantly, the core operation in High Performance Linpack (HPL)
  - Should give optimal performance…

How Successful are Compilers with CPU Intensive Code?

Large gap between natural code and specialized code

Hand-tuned

Compiler



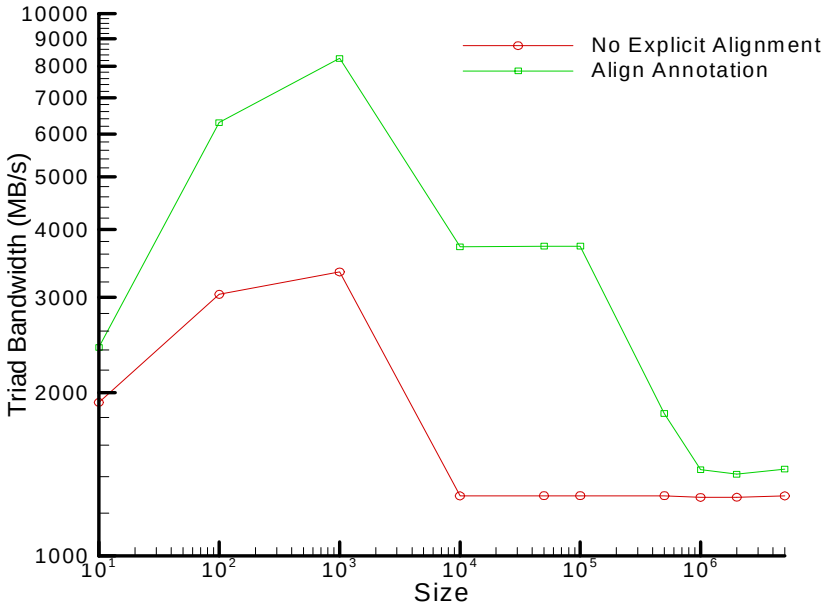
From Atlas

Enormous effort required to get good performance

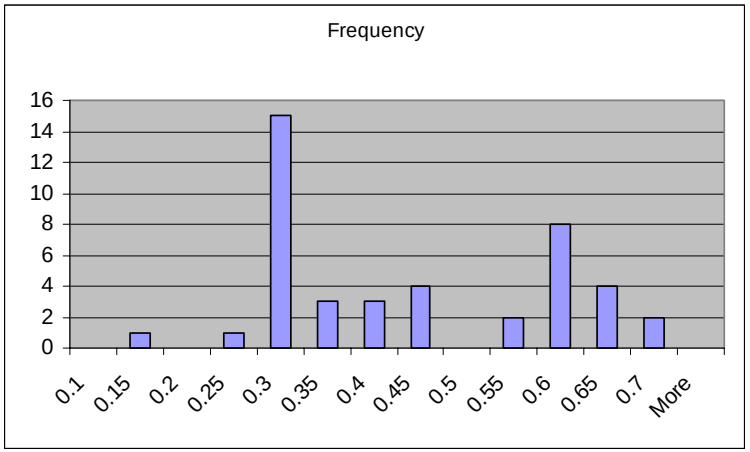
One Approach to Improving Performance Productivity: Source Annotations

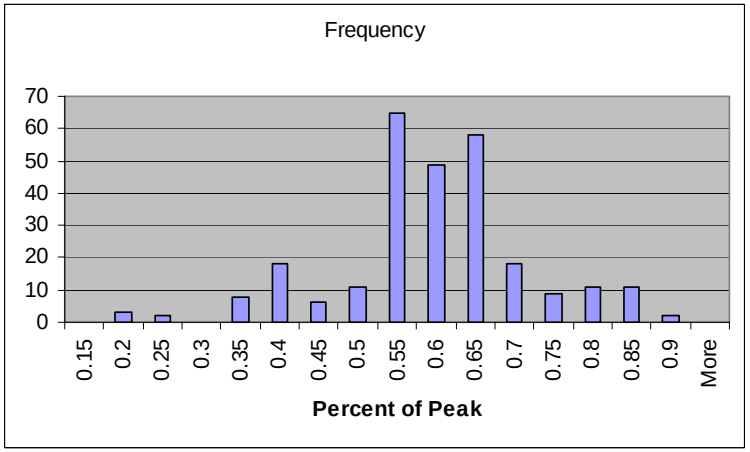
- **On BG/L, use of the second FPU requires that data be aligned on 16-byte boundaries**
- **Source code requires non-portable pseudo-functions (__alignx(16,var))**
- **By using simple, comment-based annotations, speeds up triad by 2x while maintaining portability and correctness**

```
void triad(double *a, double *b,
           double *c, int n)
{
  int i;
  double ss = 1.2;
  /* --Align;;var:a,b,c;; */
  for (i=0; i<n; i++)
    a[i] = b[i] + ss*c[i];
  /* --end Align */
}
```



Percent of Peak for HPL in Clusters

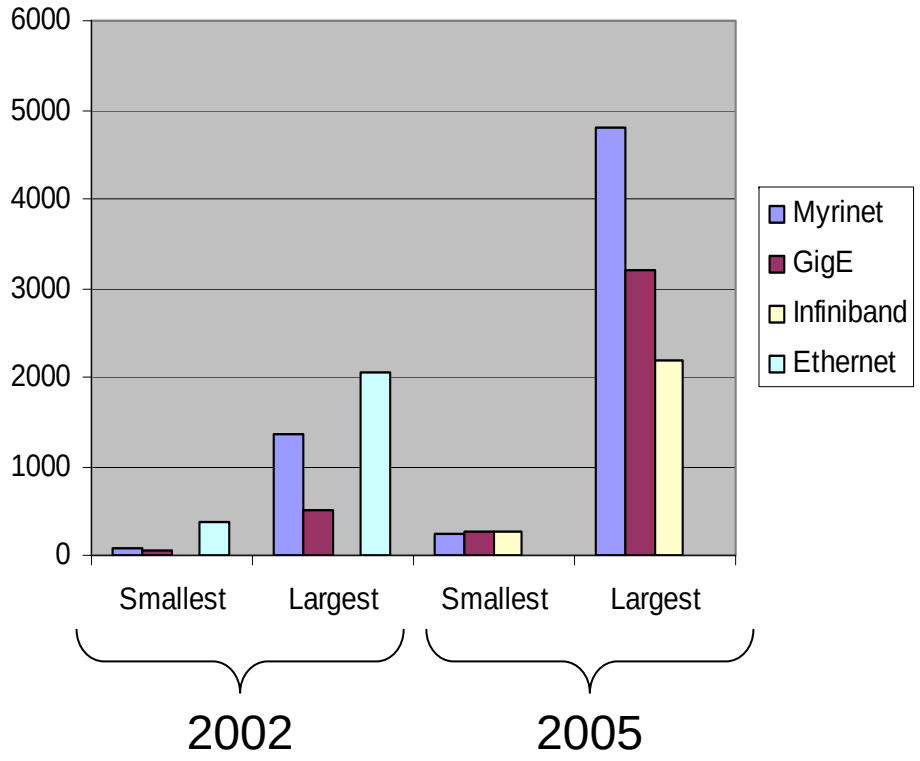




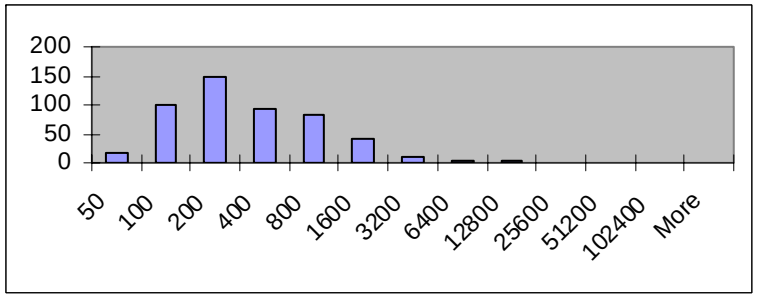
- **Percent of peak 2002 — 2005, for Intel Processors**
- **In 2002**
  - Many systems using Ethernet (not even “Fast” Ethernet)
- **In 2005**
  - Most systems clustered around the same “typical” percent of peak (65%)
  - Infiniband, Myrinet, Quadrics dominate at 80%+ of peak
  - Even for HPL, the network does matter…

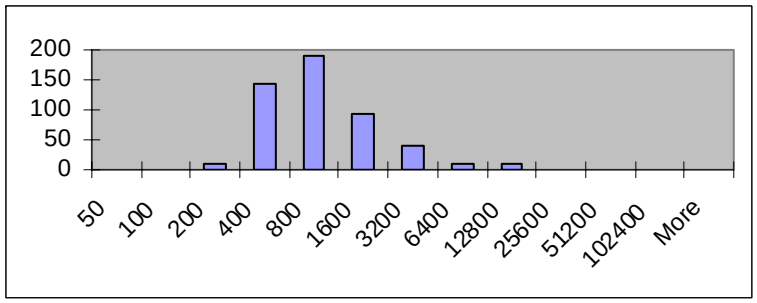
Trends in Scale

Processor count for Top500 by Interconnect



Processor Count for Top500





Parallel Performance

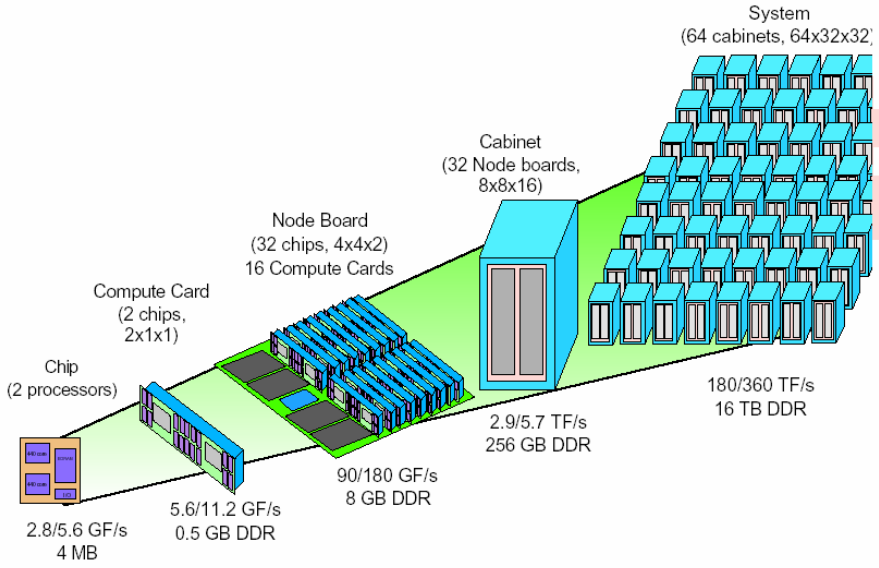
- **Scale of systems continues to increase**
  - Solutions that are adequate for 10-100 inadequate for 1000-10000 nodes
- **Consequences**
  - Physics sets a lower bound on latency
  - Maintaining consistency between different storage objects
    - *E.g., files, memory within a parallel application*
  - Non-scalable operations become an issue
  - View cluster as single computer (system) not a gathering of individuals
  - Cluster versus collection
    - *Collection – used individually by users but managed collectively by admins*
    - *Cluster – used collectively by users (e.g., MPI programs)*

The Dimensions of a Typical Cluster

- **6.1 m x 2.1 m x 1m**
- **1-norm size (airline baggage norm) = 9.2m**
- **At 2.4Ghz, =**
  74 cycles
  (49 x 17 x 8)
- **Real distance is greater**
  - Routes longer
  - Signals travel slower than light in a vacuum
- **A "remote put" must cost > 74 cycles**
  - No clever trick will fix this

Why Intrinsically Scalable Semantics?

- **Large systems cannot be handled process by process**
  - What is easy for 10, works for 100, and works awkwardly for 1000 will fail for 10000
    - *Race conditions*
    - *Resource limits*
    - *Serialization*
- **Large systems are being built**
  - Cray Red Storm has over 10000 nodes
  - Several QCDOC systems at this scale
  - IBM Blue Gene/L has 64K nodes



How Do We Achieve Intrinsic Scalability?

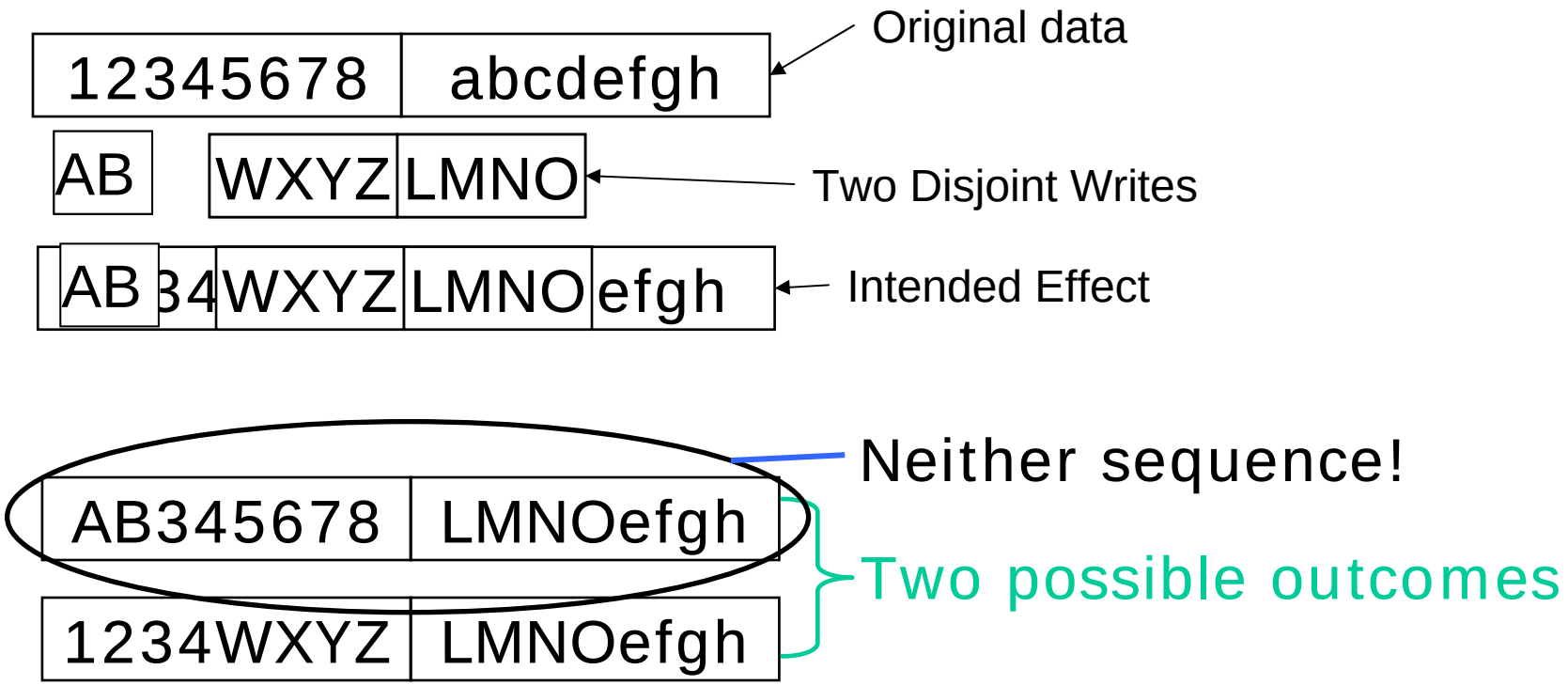
- **Operations must be collective**
- **Must not enumerate members of set of cooperating processes/threads**
- **Semantics of operations must prevent race conditions**
- **Semantics of operations must be achievable with reasonable implementation effort**
  - If effort is large, the implementation is likely to be slow
    - *Remember the rule for project goals:*
      fast, correct, or on-time – pick at most two
- **An example of a troublesome model: POSIX I/O**
  - Motivated by the best of intentions (as in “the road to hell is paved with”)

POSIX I/O Semantics

- **All writes visible to all processes immediately**
  - Write by one process can be read immediately by another process
  - Essentially sequential consistency for I/O operations
- **Severe performance problems**
  - NFS is not POSIX
    - *Precisely because of the performance problems*
  - Essentially a cache-consistency problem, with no hardware support
    - *NFS chose incoherent caches (!)*

The Problem with NFS

- **Writes to <u>disjoint</u> parts of the same file may be lost**
  - File operations on are a block basis
  - Programmer has no control over blocking



MPI I/O Semantics

- **I/O operations visible only within the MPI "job" until special actions taken (e.g., MPI_File_sync or MPI_File_close)**
- **Matches common use of files by scientific applications**
- **Matches Fortran requirements**
- **Are a form of relaxed consistency**
  - Data stored is well-defined
    - *Avoids the NFS problem*
  - Just not instantaneously visible to everyone
- **A further advantage**
  - A parallel application, using MPI-IO, can easily create one file, rather than one per process
    - *Easier to manage for the user*
    - *Immediately suitable to post processing analysis tools*
- **However, it can be tripped up by the file system…**

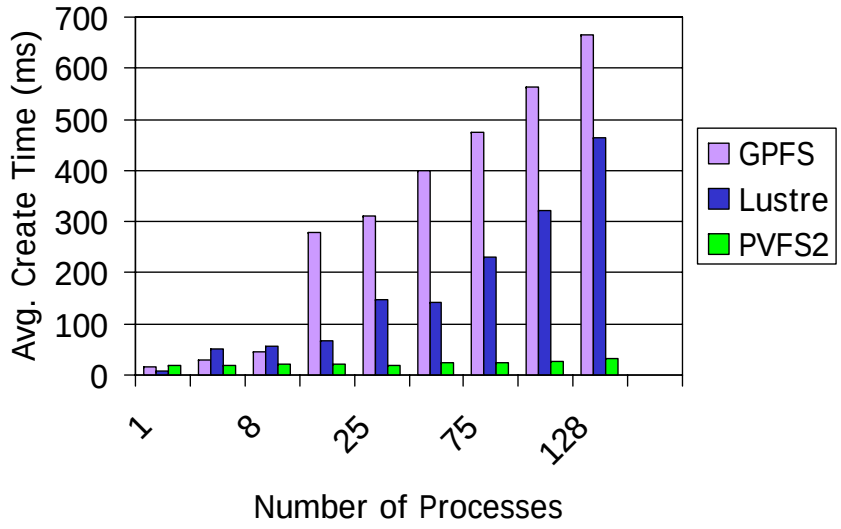
Scalability of Common Operations: Creating Files

- **Even creating files can take significant time on very large machines!**
- **Why?**
  - It's complicated ☺
  - …but it mostly has to do with the interface we have to work with and implications on synchronization
- **What happens if we change this interface?**

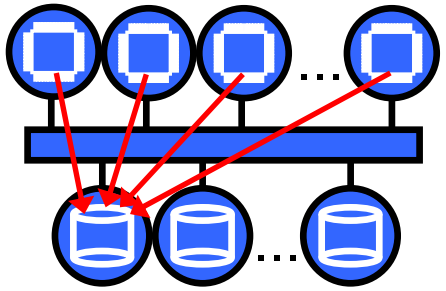
Creating Files Efficiently

- **If we improve the file system interface, we get better performance**
  - Better building block for MPI-IO

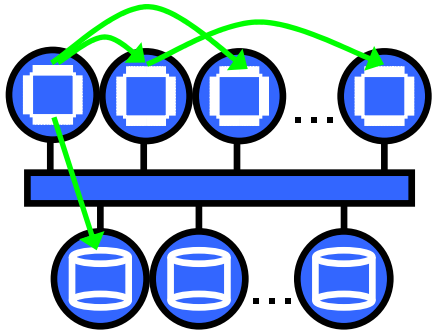
MPI File Create Performance (small is good)



Thanks to Rob Ross and the PVFS2 team



POSIX file model forces all processes to open a file, causing a storm of system calls.



A handle-based model uses a single FS lookup followed by a broadcast of the handle (implemented in PVFS2).

Emulating a Single System

- **GLUnix (1994)**
  - Early parallel Unix tools developed at UCBerkeley
  - Central master, provides simplest, most reliable fault tolerance; adequate for the largest clusters of the day (64-100 nodes)
- **Scalable Unix Tools (1994)**
  - Developed in response to need for faster (e.g., more scalable) tools for the IBM SP1 (ANL's 128 node system)
  - Many similar tools, extending Unix commands to clusters, have been developed since
- **Bproc (~1999)**
  - Linux extension for providing a distributed process space
- **openMosix (2002)**
  - Linux extension for single system image
- **MPISH (2005)**
  - Parallel shell language based on MPI
- **System/Shell Programming with data-parallel models**

A Concurrent Benchmarking Script

```
#!/usr/bin/env mpish2

rank=`rank.mpi`

slot="AA"
size="2"
base="0"
count="0"

while [ "$slot" == "A
    index=`expr $rank - $base`
    if [ "$index" -lt  "$size" ] ; then
        slot="$count"
    else
        count=`expr $count + 1`
        base=`expr $base + $size`
        size=`expr $size "*" 2`
    fi
done
```

These execute concurrently

```
time.mpi -t "size=full" progname

case $slot
    1)
      time.mpi -t "size=2" progname
      ;;
    2)
      time.mpi -t "size=4" progname
      ;;
    3)
      time.mpi -t "size=8" progname
      ;;
    4)
      time.mpi -t "size=16" progname
      ;;
esac
```

Parallel programs have “.mpi” extension

Narayan Desai, Rick Bradshaw, and Rusty Lusk – For more info, see http://www.mcs.anl.gov/cobalt/

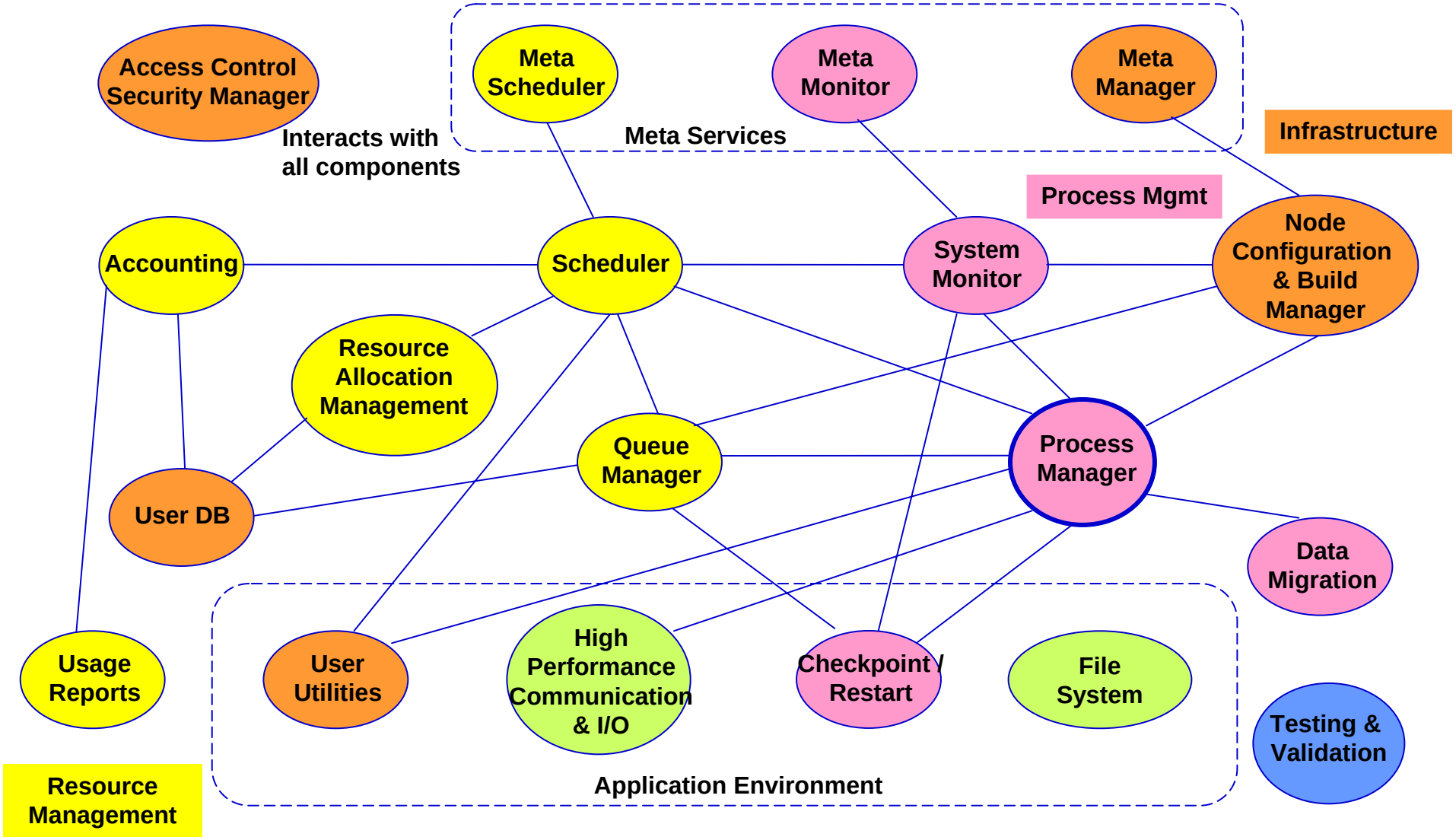
Features of MPISH

- **Data-parallel model**
  - Actions on collections of processes
  - Arbitrary collections of processes
    - *Not just all (MPI_COMM_WORLD)*
- **Ultimate Vision**
  - From high-level, scalable system software components to a high-level, scalable language for controlling clusters
    - boot_up_cluster(configfile);
    - if (cluster_broken);
      - fix_it;
    - else
      - main_loop();
      - …
- **Leverages MPI implementation**
  - For example, can use optimized MPI collective communication, such as MPI_Bcast, for file and executable staging
  - MPI-based version much faster and more robust than special purpose file-staging code developed for a large cluster
  - Leads us to the last step on our journey…

Building on the Work of Others

- **Real progress often comes from adding to and extending existing tools**
  - MPI is an example
    - *Successfully raised the level of abstraction beyond proprietary message-passing systems*
    - *Support for libraries (e.g., PETSc, ScaLAPACK, many others) allowed "MPI-less programming" (including several Gordon Bell Prize winners)*
    - *Leverages (and depends upon) advances in compiler technology, threads, development tools*
  - Exchanges some *potential* advantages from "mudball" integration with clean, component interfaces
  - Only a first, small step in parallel programming environments
- **It is time for clusters to embrace this**
  - Example: system management components
  - Goal: Build an ecosystem of tools
    - *Let competition accelerate tool development*
  - Be wary of software that takes control
    - *How many different kernel patches can you use to build a workable system?*

Scalable Systems Software SciDAC Components



Cobalt Architecture

- **Component architecture based on the SciDAC Scalable System Software project**
  - Provides easy portability to new platforms, currently supports clusters running Linux and MacOSX, and BG/L systems
  - Highly customizable – only 5K lines of component code
  - Well-definied interfaces provide mechanism for ad-hoc usage of component data
- **Think collectively:**
  - Collective operations on nodes
  - Allreduce on the return codes
  - Split nodes into “communicators”
  - Collectively handle those that succeeded
  - Collectively handle those that failed

Is Performance Everything?

"In August 1991, the Sleipner A, an oil and gas platform built in Norway for operation in the North Sea, sank during construction. The total economic loss amounted to about $700 million. After investigation, it was found that the failure of the walls of the support structure resulted from a serious error in the finite element analysis of the linear elastic model."
(http://www.ima.umn.edu/~arnold/disasters/sleipner.html)

Cluster Challenges

- **Enlarge the User Community**
  - Address performance needs
  - Simplify use and operations of cluster
  - Are new languages such as Chapel, Fortress, or X10 appropriate for clusters?
  - Quantify progress with appropriate measures (not students writing Jacobi sweeps — find out what real users *need*)
- **Build "Cluster-Centric" Solutions**
  - Exploit collective and data parallel models
  - Apply to I/O, system operations, and applications
- **Create Component Software**
  - "Play well with others"
    - *Components that interoperate with others*
    - *Cluster-oriented semantics*
      - Race-free, scalable
    - *Prove it!*
  - Build increasingly powerful tools by composing lower-level building blocks