Middleware and Distributed Systems

Cluster and Grid Computing

Peter Tröger

Architectural Classes

- Taxonomy of Flynn:
  - SISD - Uniprocessor machine, old mainframes
  - SIMD - Vector processor, GPU, Cell processor, SSE, AltiVec
    - Same instruction on different data, usually 1024-16384 processing units
  - MISD - For redundant parallelism, no mass market
  - MIMD
    - SPMD - Parallel programming
    - MPMD - Distributed systems, e.g. master-worker application

Shared Memory vs. Distributed Memory System

- Shared memory (SM) systems
  - SM-SIMD: Single CPU vector processors
  - SM-MIMD: Multi-CPU vector processors, OpenMP
  - Tendency to cluster shared-memory systems (NEC SX-6, CraySV1ex)
- Distributed memory (DM) systems
  - DM-SIMD: processor-array machines; lock-step approach; front processor and control processor
  - DM-MIMD: large variety in interconnection networks
- Virtual shared-memory systems (High-Performance Fortran, TreadMarks)

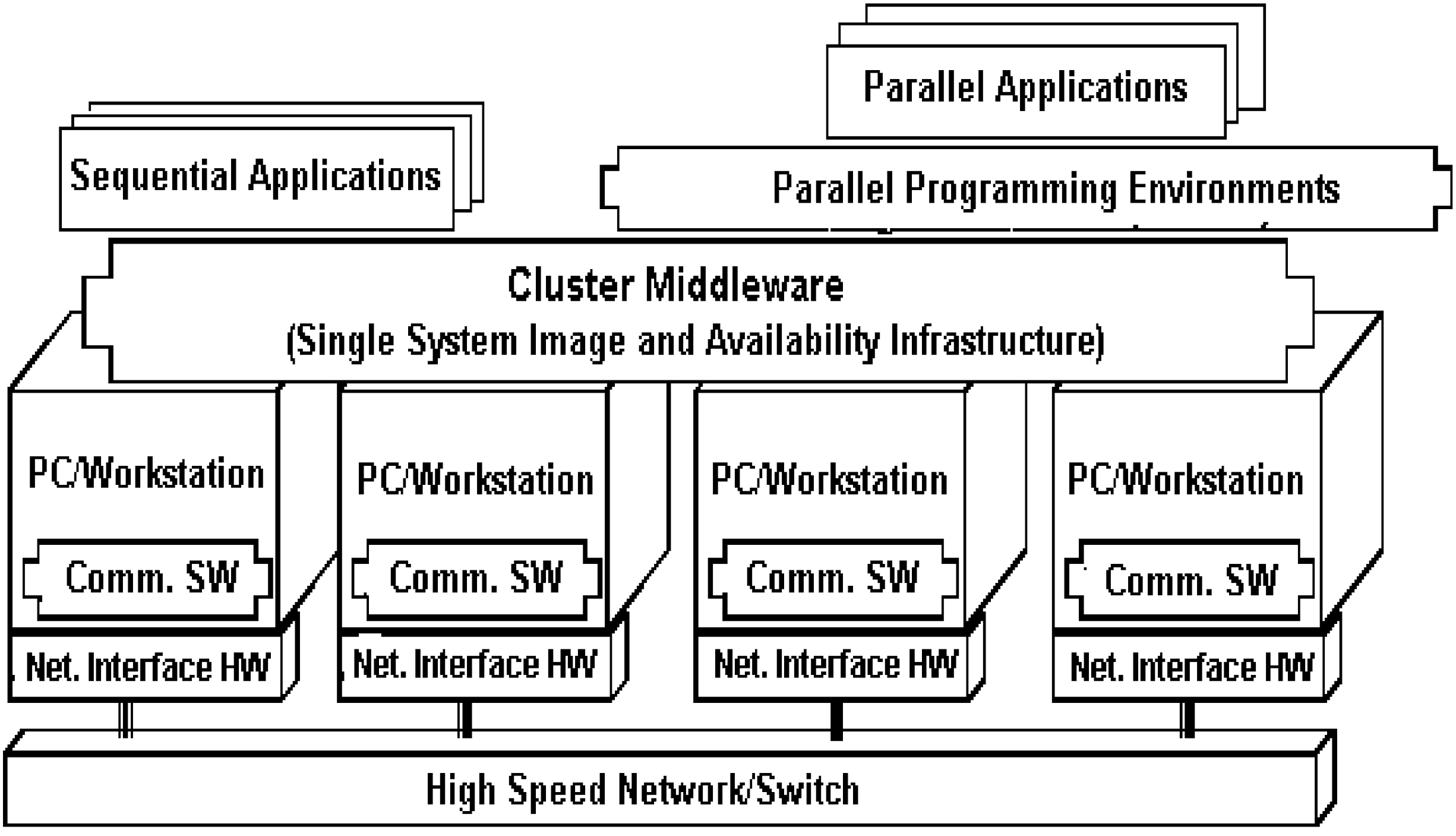
Cluster Systems

- Collection of stand-alone workstations/PC‘s connected by a local network
  - Cost-effective technique to connect small-scale computers to a large-scale parallel computer
  - Low cost of both hardware and software
    - Users are builders, have control over their own system (hardware infrastructure and software)
- Distributed processing as extension of **DM-MIMD**
  - Communication between processors is orders of magnitude slower
  - PVM, MPI as widely accepted programming standards
    - Used with cheap LAN hardware

Comparison

	MPP	**SMP**	**Cluster**	**Distributed**
Number of nodes	O(100)-O(1000)	O(10)-O(100)	O(100) or less	O(10)-O(1000)
Node Complexity	Fine grain	Medium or coarse grained	Medium grain	Wide range
Internode communication	Message passing / shared variables (SM)	Centralized and distributed shared memory	Message Passing	Shared files, RPC, Message Passing, IPC
Job scheduling	Single run queue on host	Single run queue mostly	Multiple queues but coordinated	Independent queues
SSI support	Partially	Always in SMP	Desired	No
Address Space	Multiple	Single	Multiple or single	Multiple
Internode Security	Irrelevant	Irrelevant	Required if exposed	Required
Ownership	One organization	One organization	One or many organizations	Many organizations

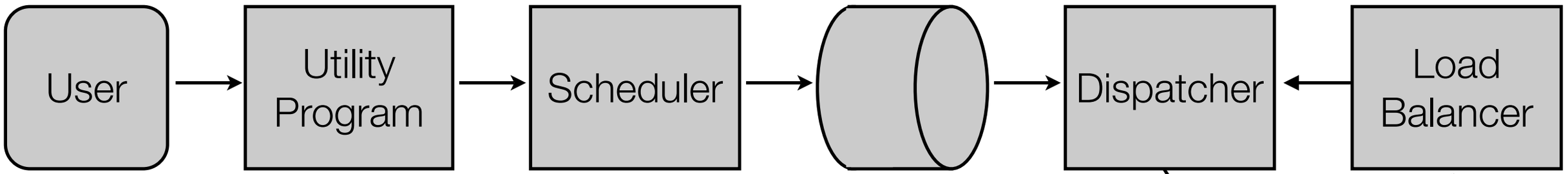
K. Hwang and Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming; WCB/McGraw-Hill, 1998


Parallel Applications
Sequential Applications
Parallel Programming Environments
Cluster Middleware
(Single System Image and Availability Infrastructure)
PC/Workstation
Comm. SW
Net. Interface HW
PC/Workstation
Comm. SW
Net. Interface HW
PC/Workstation
Comm. SW
Net. Interface HW
PC/Workstation
Comm. SW
Net. Interface HW
High Speed Network/Switch

Cluster System Classes

- High-availability (**HA**) clusters - Improvement of cluster availability
  - Linux-HA project (multi-protocol heartbeat, resource grouping)
- Load-balancing clusters - Server farm for increased performance / availability
  - Linux Virtual Server (IP load balancing, application-level balancing)
- High-performance computing (**HPC**) clusters - Increased performance by splitting tasks among different nodes
  - Speed up the computation of one distributed job (FLOPS)
  - Beowulf architecture as popular open source example
- High-throughput computing (**HTC**) clusters - Maximize the number of finished jobs
  - All kinds of simulations, especially parameter sweep applications
  - Special case: Idle Time Computing for cycle harvesting

Simple Queuing Management System



- Utility Program - Command line tool for the user
- Scheduler - Subsystem that services users requests
  - After user submits a job, scheduler queues job in its queue
  - Makes decision based on scheduling policy
- Queue - Collection of jobs, order based on attributes/policy
- Dispatcher - Performs the submission of jobs in queue
- Load Balancer - Selects appropriate set of compute nodes, based on monitoring

Batch Queuing System

- “Batch processing” : Run a number of jobs without human interaction
  - Queuing system ensures that jobs only get scheduled when resources are available to run job
  - Ensures that user's jobs are scheduled according to a predefined policy (e.g. fair share scheduling)
- Advanced queuing systems
  - Job priorities, suspension, runtime limits
  - Multiple queues with resource properties, matched against the user requirements (e.g. minimum amount of RAM, parallel jobs)
  - Support for interactive jobs, file staging, job migration or job monitoring
  - Checkpointing - save the current job state for fault tolerance or migration

Scheduling

- Queuing approach
  - Jobs are submitted to the scheduler, put into a batch queue
  - Jobs waits in a queue for execution (**wait time**), different queue priorities
  - Resources are assigned to waiting jobs, which are taken out of the queue and executed on the particular node (**execution time**)
  - **Turnaround time**: Time between job submission and completion (wait time + execution time)
- Scheduling has to consider different job types - Number of requested processors (job width), job size, estimated runtime, priority
- Conflicting optimization goals: system job throughput vs. average job turnaround time vs. resource utilization

Space-Sharing Scheduling

- Matching resource is assigned to a job until it terminates
- **First come first served** (FCFS) / **First in first out** (FIFO)
    - Run jobs in the order they arrive, good for low job load
    - Worst case: Long job arrives first
- **Round robin** (RR) - Current job is suspended when quantum is exhausted
    - Favours short process demands, depending on quantum
- **Shortest job first** (SJF) - Optimizes throughput and turnaround time
- **Longest job first** (LJF) - Maximize utilization at the costs of turnaround time
- **Shortest remaining time first** (SRTF) - Preemptive version of SJF, re-scheduling on each process arrival

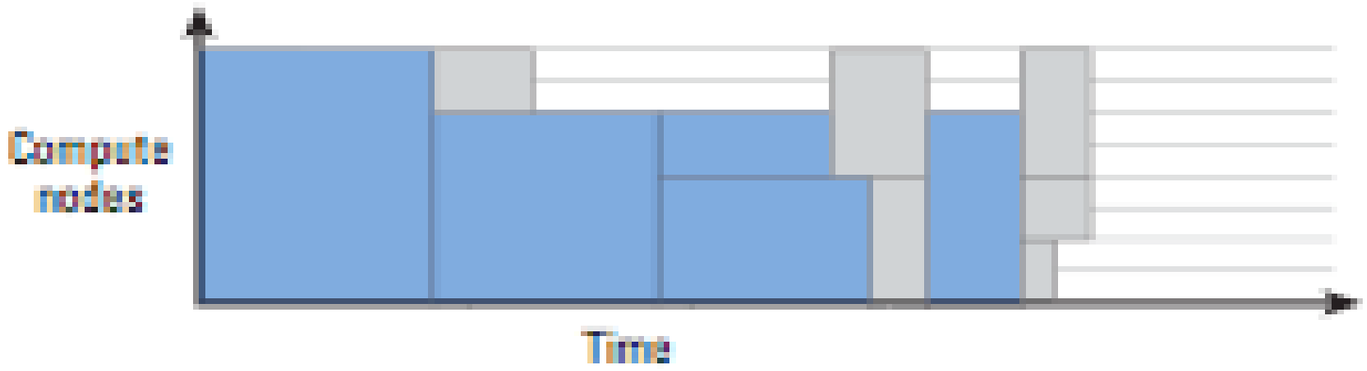
Example: Shortest-Job-First

- Though FCFS is fair, it may not optimize the average waiting time
- job1 (10 minutes), job2 (1 min), job3 (1 min)
  - job1 waits 0 minutes, job2 waits 10 minutes, job3 waits 11 minutes
  - Average waiting time is (0+10+11)/3 = 21/3 = 7 minutes
- If we do job2, job3, job1
  - job2 waits 0 minutes, job3 waits 1 minute, job1 waits 2 minutes
  - Average waiting time is (0+1+2)/3 = 3/3 = 1 minute
- Periodical sorting of the queue
- Good turnaround time for short jobs

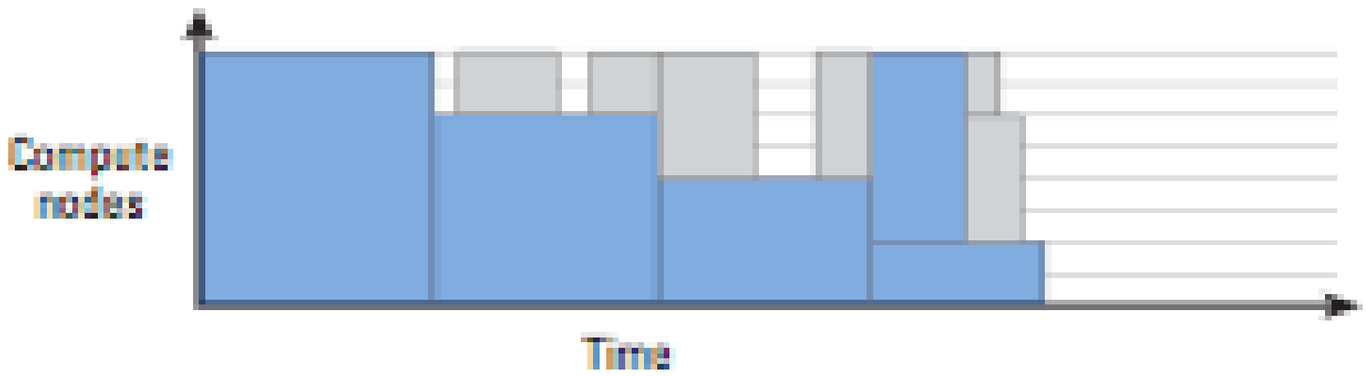
Backfilling / Advanced Reservation

- Backfilling
  - Optimize schedule by filling the scheduling gaps with small jobs
  - Demands priority scheduling
  - Does not alter the schedule sequence
    - Can either increase utilization or decrease response time
    - Need execution time prediction by the user
- Advanced reservation
  - Build a schedule based on wall clock time assumption

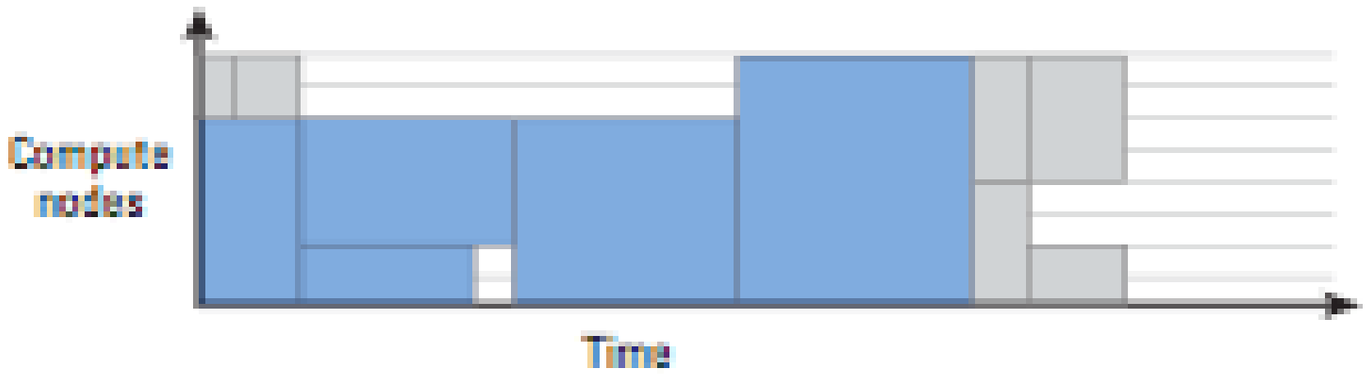
Longest job first schedule



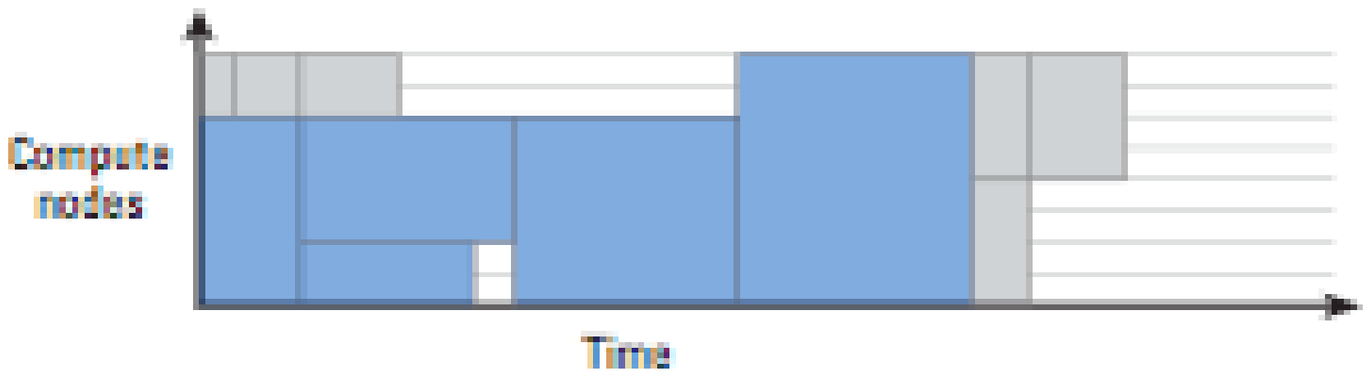
Longest job first and backfill schedule



Shortest job first schedule



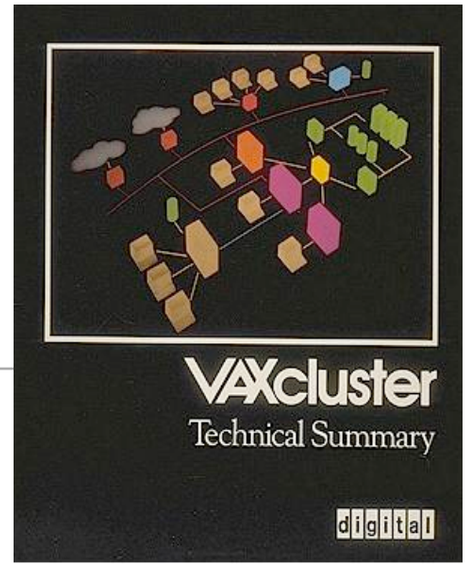
Shortest job first and backfill schedule



Cluster Benchmarking

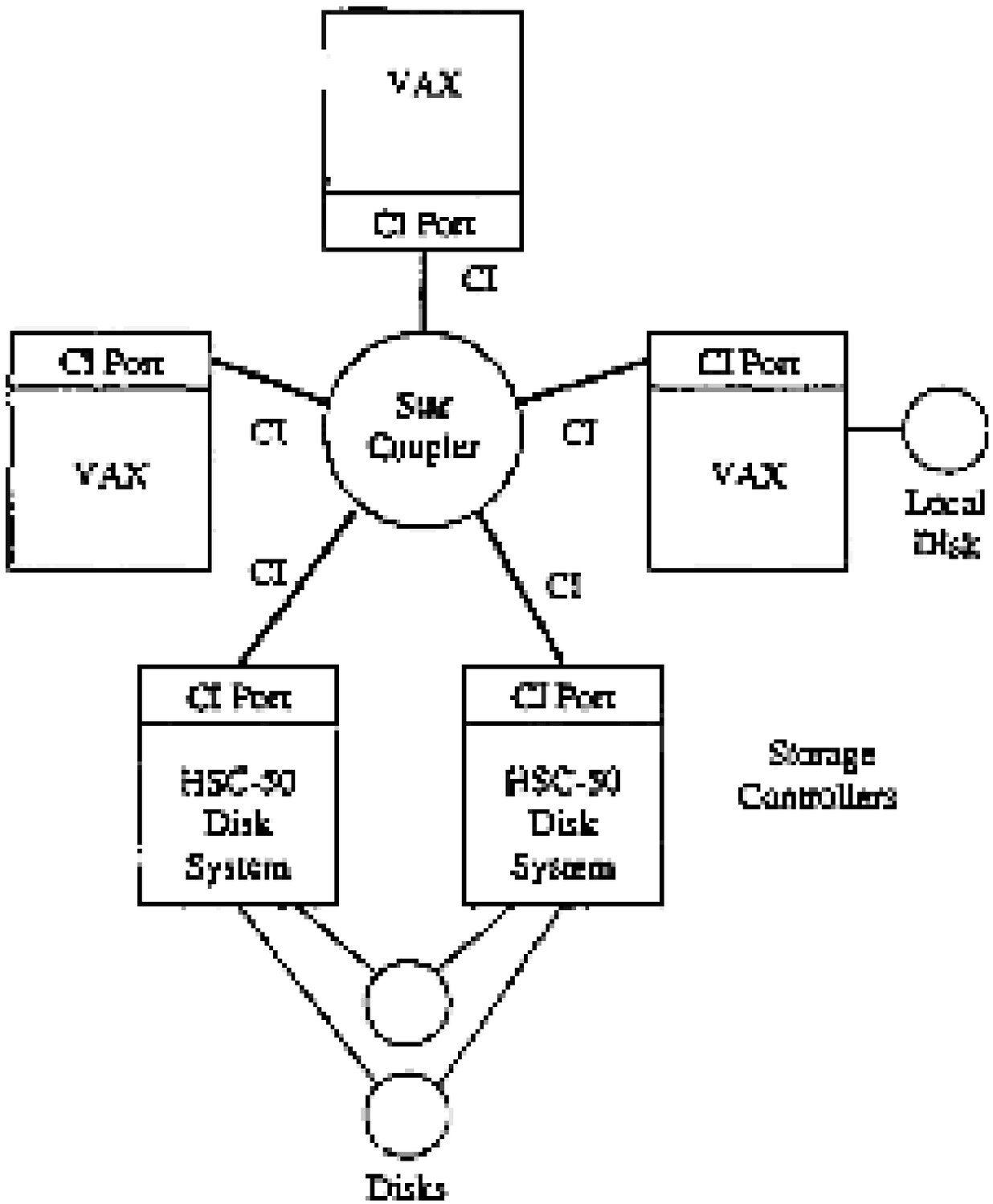
- TOP500.org
  - Collection started in 1993, updated every 6 months
  - 500 most powerful computers / clusters in the world
  - Comparison through Linpack benchmark results
- Linpack
  - Measures floating point rate of execution
  - Solving of a dense system of linear equations, grown since 1979
  - Compiler optimization is allowed, no changes to the code
  - 3 tests: Linpack Fortran n=100, Linpack n=1000, Linpack Highly Parallel Computing Benchmark (used for TOP500 ranking)

History



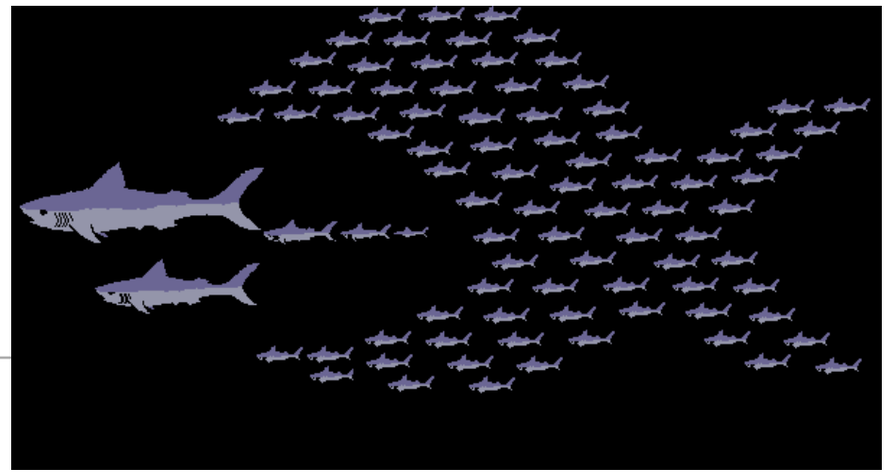
- 1977: ARCnet (Datapoint)
  - LAN protocol, such as Ethernet
  - DATABUS programming language, single computer with terminals
  - Transparent addition of ‚compute resource‘ and ‚data resource‘ computers
  - No commercial success
- May 1983: VAXCluster (DEC)
  - Cluster of VAX computers („Closely-Coupled Distributed System“)
  - No single-point-of-failure, every component that could fail was duplicated
  - High-speed message-oriented interconnect, distributed VMS
  - Distributed lock manager for shared resources

VAXCluster Architecture



Maximum node separation
of 90 meters
CI: dual path, serial
interconnect, 70MBit/s

NOW



- Berkeley Network Of Workstations (NOW) - 1995
- Building large-scale parallel computing system with COTS hardware
- GLUnix operating system
  - OS layer, transparent remote execution, load balancing
  - Network PID‘s and virtual node numbers for communication
- Network RAM - idle machines as paging device
- Collection of low-latency, parallel communication primitives - ‘active messages’
  - Berkeley sockets, shared address space in C, MPI

Beowulf

- Software package, first prototype in 1994 (DX/4, 10MB Ethernet)
- Commodity hardware, Linux operating system, MPI and PVM, Ethernet channel-bonding, global PID space
- DIPC - cluster-wide System V shared memory, semaphores and message queues
- Donald Becker and Thomas Sterling (watch your Linux boot screen !)
- Class 1 cluster: standard hardware (SCSI, Ethernet)
- Class 2 cluster: specialized hardware
- Large community, supported by NASA

Condor

- University of Wisconsin-Madison, available for more than 15 years
- Loosely coupled cluster of workstations
- Idle-Time- and High-Throughput-Computing
- ClassAd‘s: mechanism for job requirements
- No strict FIFO queues, implements priority scheduling, fair-share consideration for users
- Popular grid frontend (Condor-G)
  - Submit Condor job as usual, runs in a (Globus) grid
- Interconnection of Condor pools (flocking)

Condor Job Submission File

```
Executable      = foo
Requirements    = Memory >= 32 &&
                  OpSys == "IRIX65“ &&
                  Arch == "SGI“
Rank            = Memory >= 64
Image_Size      = 28 Meg
Error           = err.$(Process)
Input           = in.$(Process)
Output          = out.$(Process)
Log             = foo.log
Queue 150
```

Codine

- (CO)mputing in (DI)stributed (N)etworked (E)nvironments,
  GENIAS Software GmbH
- Large support for heterogeneous systems (CRAY, ULTRIX, HP-UX, AIX, IRIX, Solaris)
- Merger of DQS queuing framework and Condor checkpointing
- File staging through RCP or FTP
- First-in-first-out or fair-share scheduling
- Fault tolerance through scheduler take-over
- Resource limitation even with missing operating system functionality
- Company taken over by SUN, successor Sun Grid Engine

LoadLeveler

- Proprietary scheduler from IBM (modified Condor version)
- Support for IBM hardware (started with S/6000, still support for modern systems)
- GUI as single point of access
- Job accounting
- IP striping, hardware-supported checkpointing and shared-memory communication
- llsubmit, llq, llcancel, bstat

LoadLeveler Example

```
#!/bin/sh
#@ job_type              = parallel
#@ output                = loadl_ex3.llout
#@ error                 = loadl_ex3.llerr
#@ node_usage            = not_shared
#@ resources             = ConsumableCpus(1) \
#@                         ConsumableMemory(1968 mb)
#@ wall_clock_limit = 1:10:0,1:0:0
#@ node                  = 2
#@ tasks_per_node    = 32
#@ network.mpi           = sn_all,,us
#@ environment           = MEMORY_AFFINITY=MCM; \
#                          MP_SHARED_MEMORY=yes; \
#                          MP_WAIT_MODE=poll; \
#                          MP_SINGLE_THREAD=yes; \
#                          MP_TASK_AFFINITY=MCM
#@ queue
./a.out
```

MAUI

- Open source scheduler, plugs into PBS, LoadLeveler, LSF or GridEngine
- Provides backfilling and advanced reservation
  - First-fit, best-fit and balanced-fit backfilling
  - Consideration of job properties (number of processors, working set, …)
- No queue policy approach, all jobs are considered to be the next
- Multiple factors: job resource requirements, queue time, expansion, historical data (fair share), QoS (boosting)
- Expansion factor = (Queue_Time + Job_Time_Limit) / Job_Time_Limit
  - Relates the requested job time to the total queuing and expected runtime
  - Job with low time limit will increase it’s priority more quickly

Load Sharing Facility (LSF)

- Commercial product from Platform Inc.
- Support for several scheduling technologies
  - Fairshare, preemption
  - Advance reservation, resource reservation
  - SLA scheduling
- Tight coupling to grid infrastructures
- Smooth Windows integration

OpenPBS / PBSPro

- Portable Batch System, open source and commercial version
- Based on Network Queuing System (NQS) - Cray 1986
- Developed by NASA and NERSC
- PBSPro offers advanced reservation and multiple scheduling strategies
- Directives as script comments or *qsub* arguments
- *qstat, qsub, qdel, qalter, qhold, qrerun*
- Open source successor Torque

PBS Sample Script (qsub)

```
#/bin/sh
#
# Request 8 nodes
#PBS -l nodes=8
# Request 8 hour 10 minutes of wall-clock time
#PBS -l walltime=8:10:00
# Request that regular output (stdout) and
# terminal output (stderr) go to the same file
#PBS -j oe
# Send mail when the job aborts or terminate
#PBS -m ae
# Mail address
#PBS -M user@some.where
#Goto the directory from which you submitted the job
cd $PBS_O_WORKDIR
mpirun -np 8 ./foo
```

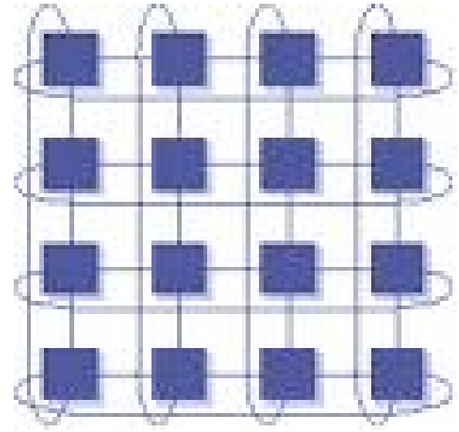
Cluster File Systems

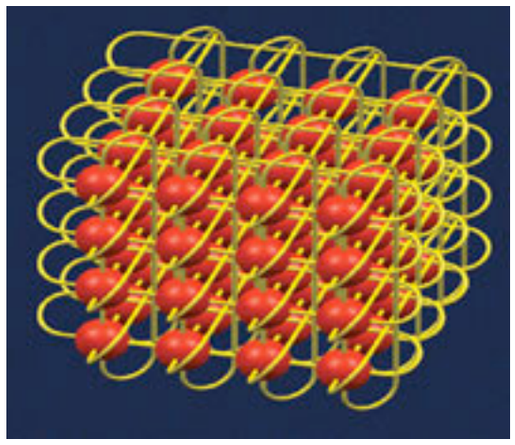
- Key features in comparison to NFS:
  - Fault-tolerant behaviour (fail-over for distributed data)
  - Load levelling, utilization of high-performance interconnections
  - ‚Single system data‘ view

- RedHat Global File System - up to 256 nodes, POSIX-compliant, SAN as backbone, no single-point-of-failure, Quotas, multi-path routing
- IBM General Parallel File System - support for thousands of disks, terabyte of data, up to 4096 nodes
  - Files are not localized (striped between nodes), multiple network path access, automatic logging and replication, fault-tolerant operation mode

Cluster Network Technology

- 100 MBit / Gigabit Ethernet
  - Cheap solution
  - High latency (around 100 µs)
- SCI
  - IEEE/ANSI standard since 1992
  - Ring structure, network as 1-D, 2-D or 3-D torus
  - Bandwidth of 400 MB/s
  - Latency under 3 µs





Cluster Network Technologies (contd.)

- Myrinet ANSI standard (1998)
  - Upstream and downstream fiber optic channel
  - New Myri-10G is compatible to Gigabit Ethernet physical layer
  - Entirely operated in user space, firmware interacts with host process
  - Maximum bandwidth of 10 GBit/s, latency around 2 µs
- Infiniband standard (2002)
  - Copper and glas fiber connections
  - Remote memory direct access (R-DMA) between host channel adapters
  - Maximum bandwidth of 96 GBit/s, latency around 1.3 - 2.6 µs
  - More expensive than Myrinet

Single System Image

- Each cluster node has exclusive set of resources, only usable from this node
- Resources should be transparently available regardless of their physical attachment
- SSI middleware offers unified access to operating system resources
- Examples: Mosix (Hebrew University of Jerusalem), OpenMosix, OpenSSI, Tornado (University of Toronto), Sprite (University of California, Berkeley), SCO UnixWare, Solaris MC, IBM cJVM

BCCD 2.2.1

Busy 100%
Load
Disk
Net
Bat

11:52:31

File Edit View Go Bookmarks T

http:/

Firefox Help Firefox Support

openMosixview 1.5

File View Config Collector Help

refresh 5 s

openMosixcollector status

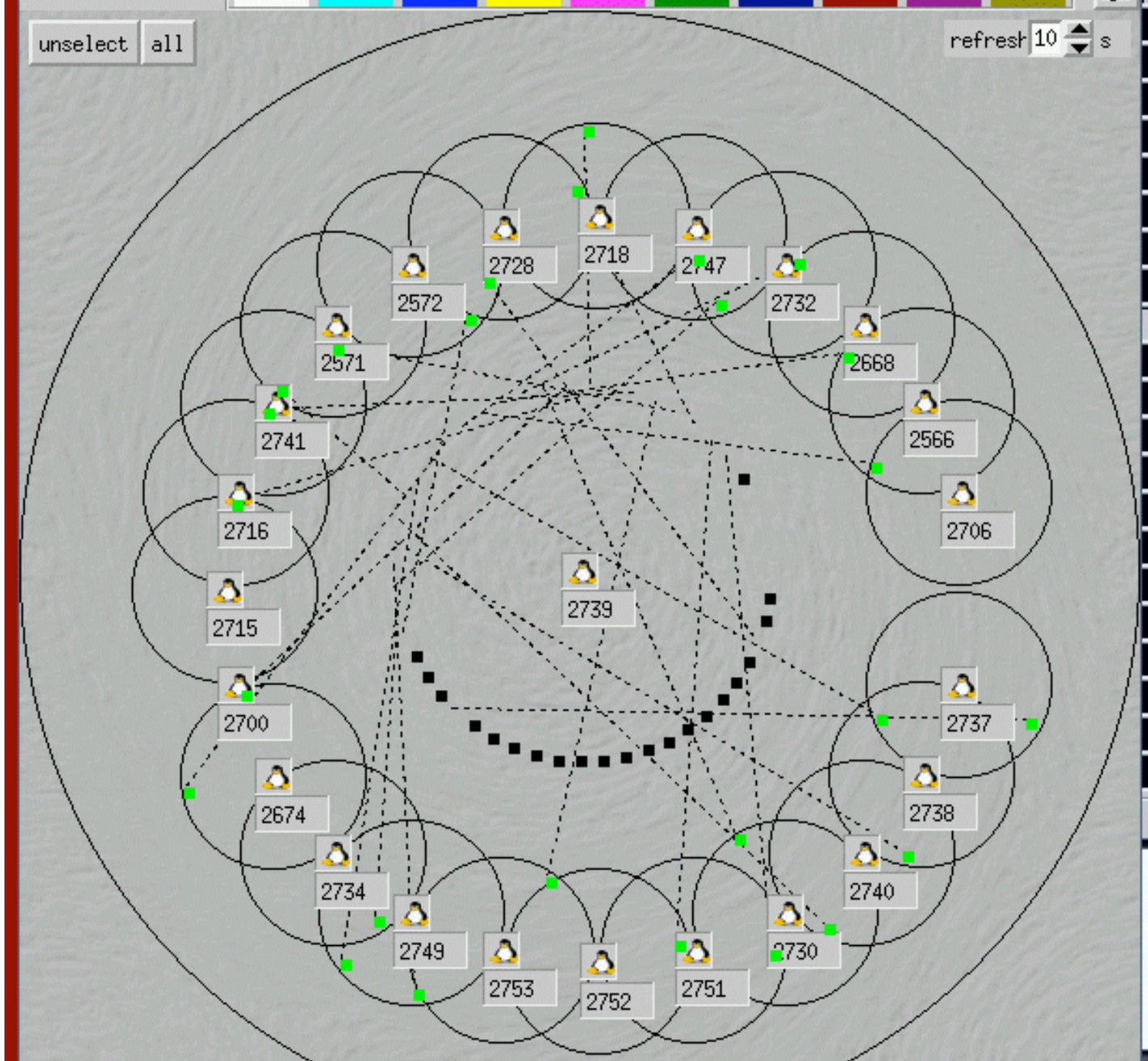
id	clusternodes	load-balancing efficiency	overall load	overall used memory	all memory	all cpu
all	all-nodes	50%	60%	13%	3304 MB	24
2737	192.168.10.177		62%	14%	127	1
				14%	127	1
			2%	17%	127	1
			64%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1
			63%	16%	127	1
			63%	14%	127	1
			100%	14%	127	1
			62%	14%	127	1
			100%	14%	127	1
			0%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1
			62%	6%	383	1
			62%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1
			62%	14%	127	1

openMosixmigmon 1.5

Process groups: 1 2 3 4 5 6 7 8 9 10

unselect all

refresh 10 s

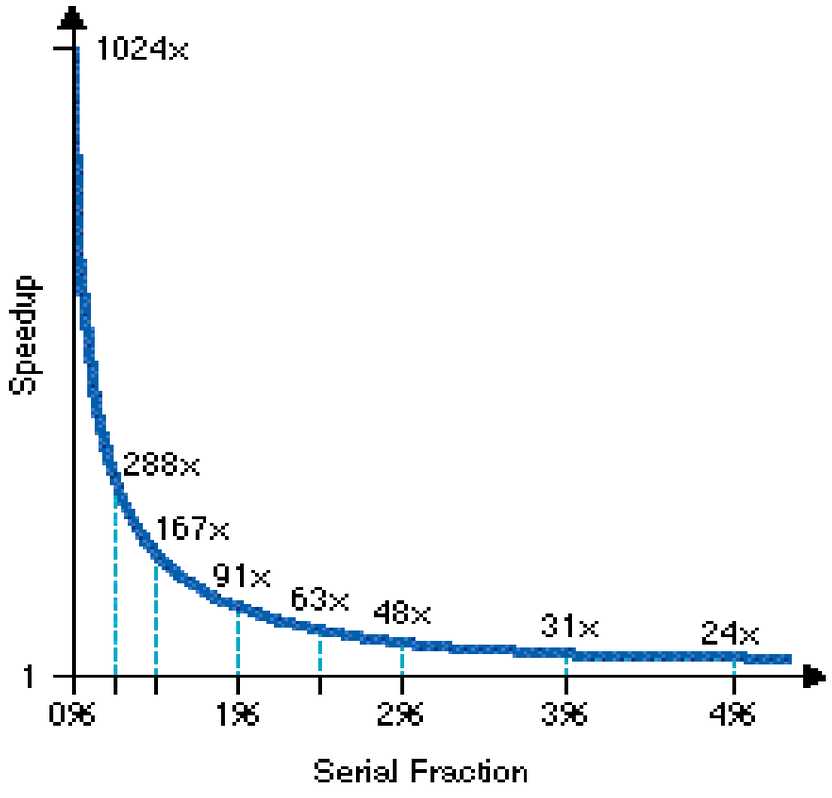


Cluster Programming

- Distributed shared memory (Linda, Tuplespaces)
- Threads
  - POSIX PThreads
  - Distributed processing with *Single System Image* cluster
- Message Passing
  - Parallel programming for distributed memory systems
  - Parallel Virtual Machine (PVM) from Oak Ridge National Laboratory
  - Message Passing Interface (MPI) defined by MPI Forum
  - Libraries for sequential programming languages
    (Fortran 77, Fortran 90, ANSI C, C++, Java)

Amdahl‘s Law

- Gene Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings, 1967
- Speedup = $$\frac{s + p}{(s + p)/N}$$
- s: Time spent on serial parts of a program
- p: Time spent on parts that can run in parallel
- N: Number of processors
- Re-evaluations from practice
  - Run time, not problem size, should be constant
  - Speedup should be measured by scaling the problem to the number of processors, not fixing problem size



The Parallel Virtual Machine (PVM)

- Intended for heterogeneous environments, integrated set of software tools and libraries
- User-configured host pool
- Translucent access to hardware, collection of virtual processing elements
- Unit of parallelism in PVM is a task, no process-to-processor mapping is implied
- Support for heterogeneous environments
- Explicit message-passing mode, multiprocessor support
- C, C++ and Fortran language

PVM (contd.)

- PVM tasks are identified by an integer task identifier (TID)
- User named groups
- Programming paradigm
    - User writes one or more sequential programs
    - Contains embedded calls to the PVM library
    - User typically starts one copy of one task by hand
    - Process subsequently starts other PVM tasks
    - Tasks interact through explicit message passing

PVM Example

```
main() {
  int cc, tid, msgtag;
  char buf[100];
  printf("i'm t%x\n", pvm_mytid());  //print id
  cc = pvm_spawn("hello_other",
            (char**)0, 0, "", 1, &tid);
  if (cc == 1) {
     msgtag = 1;
     pvm_recv(tid, msgtag); // blocking
     pvm_upkstr(buf);   // read msg content
     printf("from t%x: %s\n", tid, buf);
  } else
     printf("can't start it\n");
  pvm_exit();
}
```

PVM Example (contd.)

```
main() {
  int ptid, msgtag;
  char buf[100];
  ptid = pvm_parent(); // get master id
  strcpy(buf, "hello from ");
   gethostname(buf+strlen(buf), 64); msgtag = 1;
  // initialize send buffer
  pvm_initsend(PvmDataDefault);
  // place a string
  pvm_pkstr(buf);
  // send with msgtag to ptid
  pvm_send(ptid, msgtag); pvm_exit();
}
```

Message Passing Interface (MPI)

- Communication library for sequential programs, over 130 functions
- Based on MPI 1.0 (1994) and 2.0 (1997) standard
- Developed by MPI Forum (IBM, Intel, NSF, ...)
- Definition of syntax and semantics for portability and vendor support
- Fixed number of processes, determined on startup
  - Point-to-point o collective communication
  - Instances of the same program in different processes can communicate
- Optimized implementation for underlying communication infrastructure
- Fortran / C - Binding, also external bindings for other languages

MPI C Example

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv) {
  int myid, numprocs;
  MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);

if (myid == 0) {
  /* I am the root process */
  printf("I am the root process\n");
}else {
  /* I am only a worker */
  printf("I am worker %d out of %d\n", myid, (numprocs-1));
}
MPI_Finalize();}
```

Basic MPI

- `MPI_INIT, MPI_FINALIZE`
- *Communicators* as process group handle, sequential *process ID's*
- Nodes identified by 3-tupel: *Tag, source* and *comm*
  - `MPI_COMM_SIZE (IN comm, OUT size)`
  - `MPI_COMM_RANK (IN comm, OUT pid)`
  - `MPI_SEND (IN buf, IN count, IN datatype, IN destPid, IN msgTag, IN comm)`
  - `MPI_RECV (IN buf, IN count, IN datatype, IN srcPid, IN msgTag, IN comm, OUT status)` (blocking)
- Contants: `MPI_COMM_WORLD, MPI_ANY_SOURCE, MPI_ANY_DEST`
- Data types: `MPI_CHAR, MPI_INT, ..., MPI_BYTE, MPI_PACKED`

PVM vs. MPI

- PVM for interoperability, MPI for portability
  - PVM communicates across different architectures for the cost of some performance
  - PVM determines whether the destination machine supports the native communication functions of it‘s platform
- Programs written in different languages are not required to be interoperable with MPI
- Custom MPI implementation for each communication infrastructure
  - TCP, Myrinet, IPC, ...
  - Intel, SGI, Sun, ...

MPI Data Conversion

- No cross-language interoperability on same platform
  - „MPI does not require support for inter-language communication.“
- No standardized wire format
  - „The type matching rules imply that MPI communication never entails type conversion.“
  - „On the other hand, MPI requires that a representation conversion is performed when a typed value is transferred across environments that use different representations for the datatype of this value.“
  - Type matching through name similarity (without MPI_BYTE and MPI_PACKED)

MPI Communication Modes

- *Blocking*: do not return until the message data and envelope have been stored away
    - Standard: MPI decides whether outgoing messages are buffered
    - Buffered: MPI_BSEND returns always immediately; might be a problem when the internal send buffer is already filled
    - Synchronous: MPI_SSEND completes if the receiver started to receive the message
    - Ready: MPI_RSEND should be started only if the matching MPI_RECV is already available; can omit a handshake-operation on some systems
- Blocking communication ensures that the data buffer can be re-used

Non-Overtaking Message Order

- „If a sender sends two messages in succession to the same destination, and both match the same receive, then this operation cannot receive the second message if the first one is still pending.“

```
CALL MPI_COMM_RANK(comm, rank, ierr)
IF (rank.EQ.0) THEN
   CALL MPI_BSEND (buf1, count, MPI_REAL, 1,
                   tag, comm, ierr)
   CALL MPI_BSEND (buf2, count, MPI_REAL, 1,
                   tag, comm, ierr)
ELSE   ! rank.EQ.1
   CALL MPI_RECV (buf1, count, MPI_REAL, 0,
                   MPI_ANY_TAG, comm, status, ierr)
   CALL MPI_RECV (buf2, count, MPI_REAL, 0,
                   tag, comm, status, ierr)
END IF
```

Interwoven Messages

```
CALL MPI_COMM_RANK(comm, rank, ierr)
IF (rank.EQ.0) THEN
   CALL MPI_BSEND(buf1, count, MPI_REAL, 1,
                  tag1, comm, ierr)
   CALL MPI_SSEND(buf2, count, MPI_REAL, 1,
                  tag2, comm, ierr)
ELSE ! rank.EQ.1
   CALL MPI_RECV(buf1, count, MPI_REAL, 0,
                 tag2, comm, status, ierr)
   CALL MPI_RECV(buf2, count, MPI_REAL, 0,
                 tag1, comm, status, ierr)
END IF
```

- First message must wait in the send buffer (BSEND), since there is no matching receive
- Messages arrive in reverse order

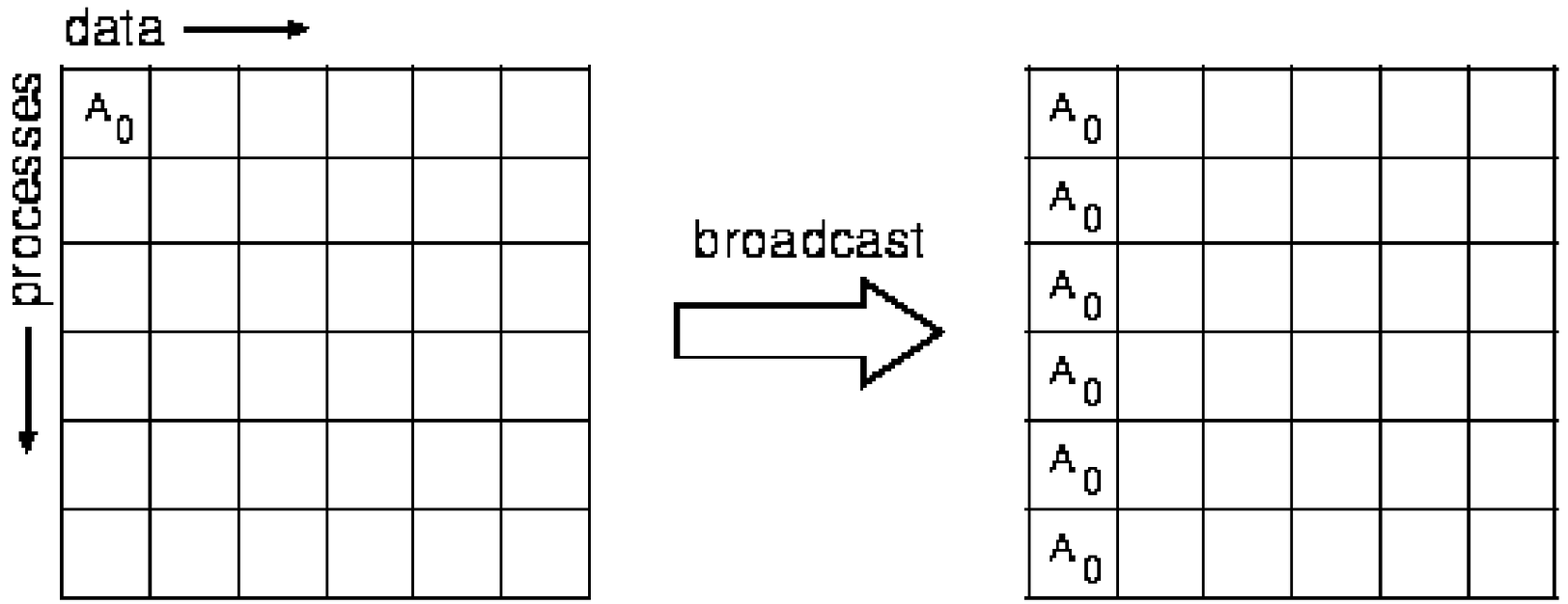
Non-Blocking Communication

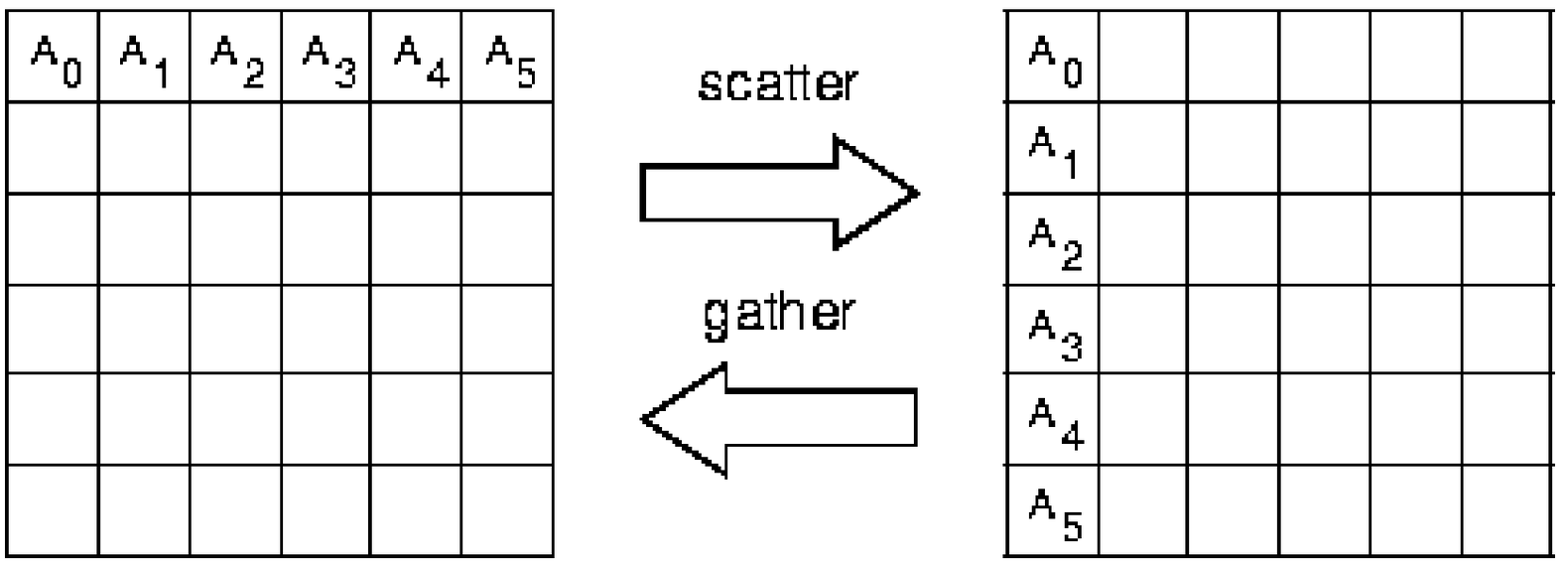
- Send/receive start and send/receive completion call, request handle
- Communication mode influences the behaviour of the completion call
- Buffered non-blocking send operation leads to an immediate return of the completion call
- ‚Immediate send‘ calls: MPI_ISEND, MPI_IBSEND, MPI_ISSEND, MPI_IRSEND
- Completion calls: MPI_WAIT, MPI_TEST, MPI_WAITANY, MPI_TESTANY, MPI_WAITSOME, ...
- Sending side: MPI_REQUEST_FREE

Collective Communication

- Global operations for a distributed application, could also be implemented manually
- `MPI_BARRIER (IN comm)`
  - Returns only if the call is entered by all group members
- `MPI_BCAST (INOUT buffer, IN count, IN datatype, IN rootPid, IN comm)`
  - Root process broadcasts to all group members, itself included
  - All group members use the same comm & root parameter
  - On return, all group processes have a copy of root‘s send buffer

Collective Move Functions



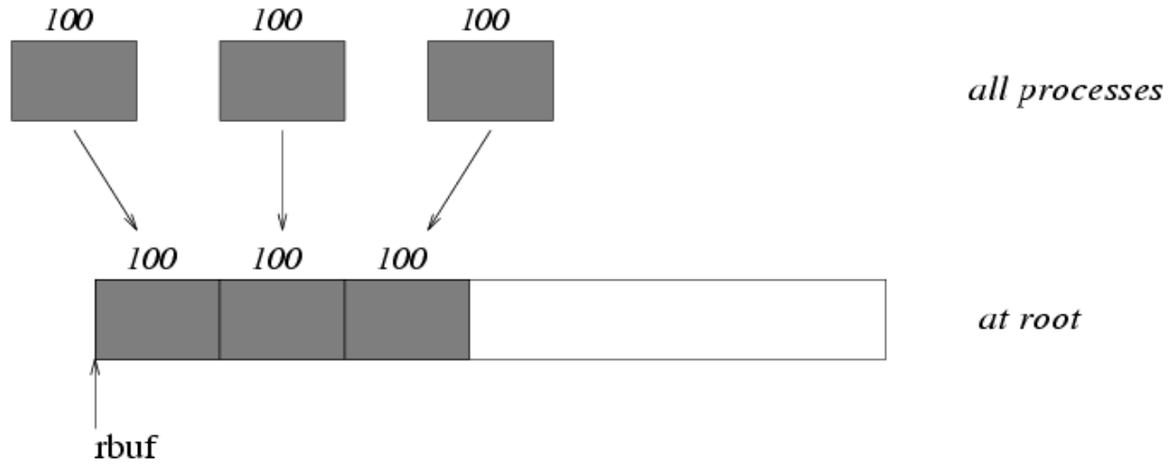


Gather

- `MPI_GATHER ( IN sendbuf, IN sendcount, IN sendtype, OUT recvbuf, IN recvcount, IN recvtype, IN root, IN comm )`
  - Each process sends its buffer to the root process (including the root process itself)
  - Incoming messages are stored in rank order
  - Receive buffer is ignored for all non-root processes
  - `MPI_GATHERV` allows varying count of data to be received from each process, no promise for synchronous behavior

MPI Gather Example

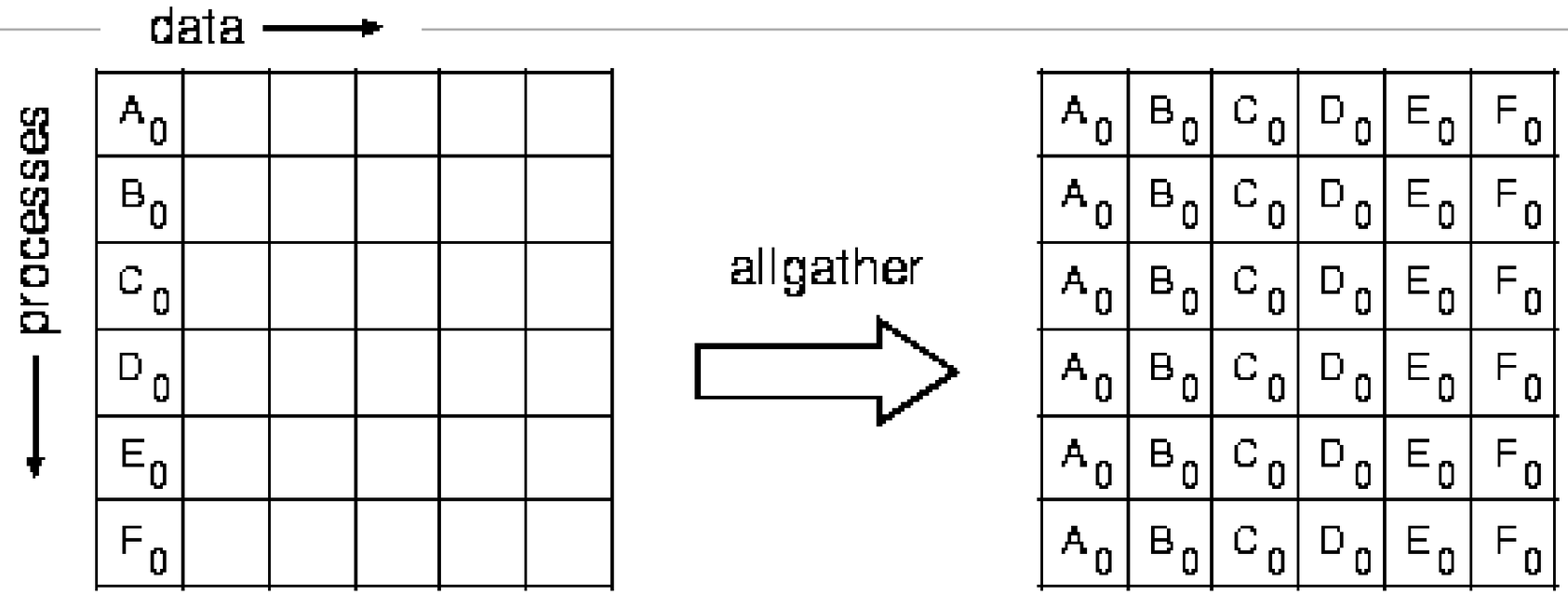
```
MPI_Comm comm;
int gsize,sendarray[100];
int root, myrank, *rbuf;
... [compute sendarray]
MPI_Comm_rank( comm, myrank);
if ( myrank == root) {
  MPI_Comm_size( comm, &gsize);
  rbuf = (int *)malloc(gsize*100*sizeof(int));
}
MPI_Gather ( sendarray, 100, MPI_INT, rbuf, 100,
            MPI_INT, root, comm );
```

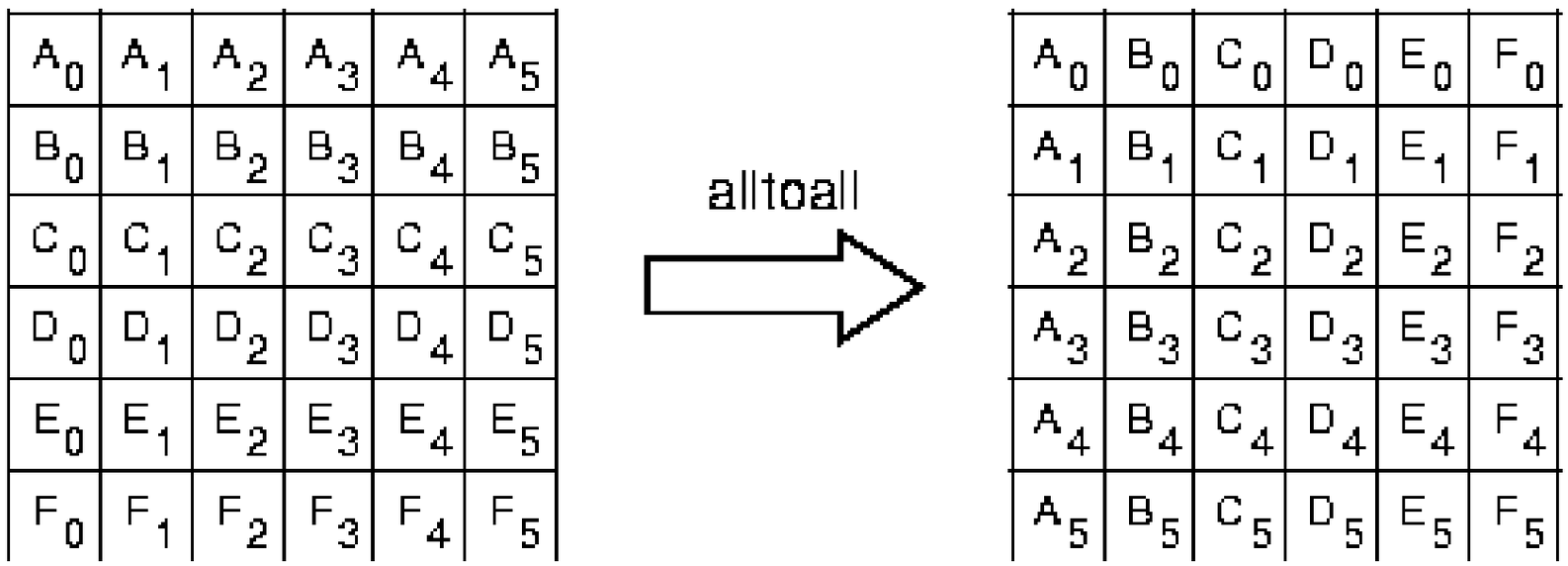


Scatter

- `MPI_SCATTER ( IN sendbuf, IN sendcount, IN sendtype, OUT recvbuf, IN recvcount, IN recvtype, IN root, IN comm )`
  - Sliced buffer of root process is send to all other processes (including the root process itself)
  - Send buffer is ignored for all non-root processes
  - `MPI_SCATTERV` allows varying count of data to be send to each process

Other Collective Functions





What Else

- `MPI_SENDRCV` (useful for RPC semantic)
- Global reduction operators
  - All processes perform a computation on their own buffer, which leads to a result transmitted to the root node (e.g. MAX; MIN, PROD, XOR, ...)
- Complex data types
- Packing / Unpacking (sprintf / sscanf)
- Group / Communicator Management
- Virtual Topology Description
- Error Handling, profiling Interface

MPICH library

- Development of the MPICH group at Argonne National Laboratory (Globus)
- Portable, free reference implementation
- Drivers for shared memory systems (ch_shmem), Workstation networks (ch_p4) , NT networks (ch_nt) and Globus 2 (ch_globus2)
- Driver implements MPIRUN (fork, SSH, MPD, GRAM)
- Supports multiprotocol communication (with vendor MPI and TCP) for intra-/ intermachine messaging
- MPICH2 (MPI 2.0) is available, GT4-enabled version in development
- MPICH-G2 is based on Globus NEXUS / XIO library
- Debugging and tracing support

Conclusion

- 20 years of research in cluster computing
- Focus on scheduling algorithms
- Hardware improvements bring DM-MIMD and cluster systems closer together
- Multiple frameworks available
- Programming with C, C++, Fortran, PVM and MPI
- Some domain-specific programming environments for parallel programs
- TOP500 Benchmark List (http://www.top500.org), Cluster Computing Info Centre (http://www.buyya.com/cluster/), http://www.linuxhpc.org

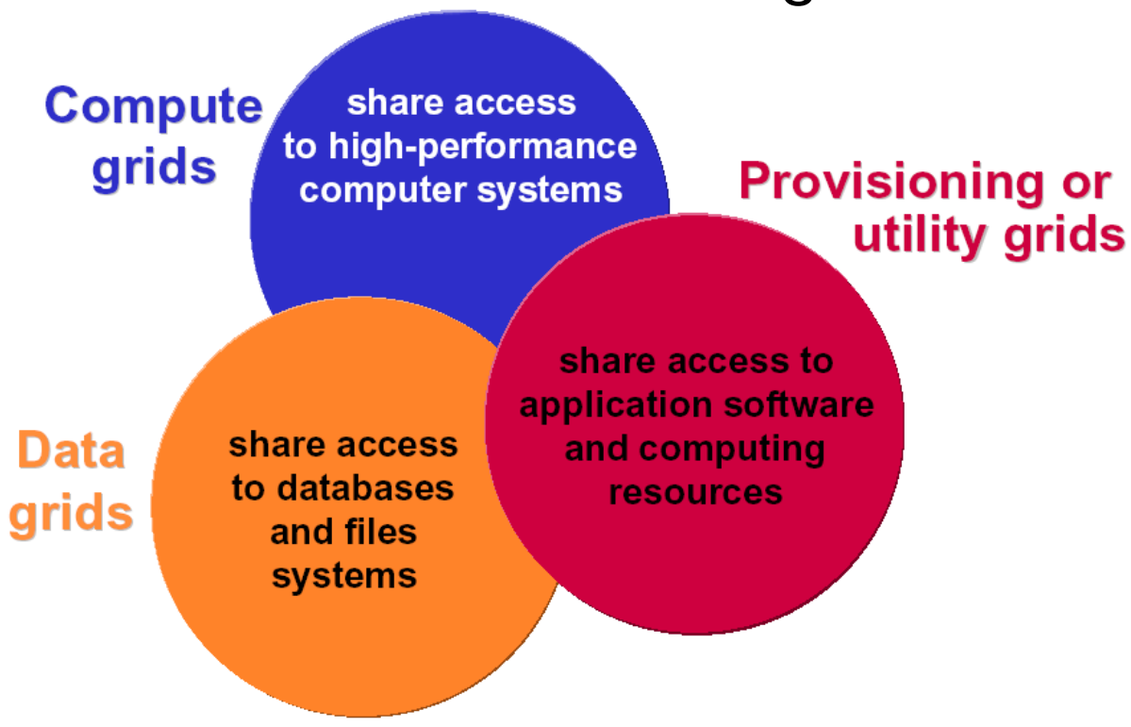
Grid Computing

„... coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.“
Foster, Kesselman, Tueke „The Anatomy of the Grid“, 2001

- Analogy to the power grid
  - Demand-driven usage of standardized services with high reliability, high availability and low-cost access
    - Innovation was not the power itself, but the coordinated distribution
- Goals
  - Coordination of resources which are not under a centralized control („virtual organization“)
  - Application of open standards (Open Grid Forum, OASIS)
  - Consideration of non-trivial quality requests (like reliability, throughput)

Classification

- Computational Grid
  - *Distributed Supercomputing:* co-scheduling of expensive resources, scalability of infrastructures, interconnection of heterogeneous HPC systems
  - *High-Throughput Computing:* Varying availability of unused resources, loosely - coupled single tasks
  - *On-Demand Computing:* Dynamic allocation of resources with billing
- Data Grid
  - Management of large data sets
- Resource Grid
  - Collaborative Work



Why Grid ?

- Computation intensive
  - Interactive simulation (climate modeling)
  - Very large-scale simulation and analysis (galaxy formation, gravity waves, battlefield simulation)
  - Engineering (parameter studies, linked component models)
- Data intensive
  - Experimental data analysis (high-energy physics)
  - Image and sensor analysis (astronomy, climate study, ecology)
- Distributed collaboration
  - Online instrumentation (microscopes, x-ray devices, etc.)
  - Remote visualization (climate studies, biology)
  - Engineering (large-scale structural testing, chemical engineering)

- Problems where big enough to require people in several organization to collaborate and share computing resources, data, and instruments

History

- 1990: US Gigabit Testbed program
- 1995: Fafner (Bellcore Labs, Syracuse University)
  - Factorizing of big numbers for cryptography
  - Web-based job distribution, anonymous registration, hierarchical servers
  - TeraFlop competition price at SC95
- I-WAY (Ian Foster, Carl Kesselman)
  - Information Wide Area Year, started at SC95
  - Interconnection of 17 supercomputing centres of 10 networks,
    I-POP server on every node
  - I-Soft software: Kerberos, central meta resource scheduler, node schedulers,
    Andrew File System (AFS)

Grid Infrastructures

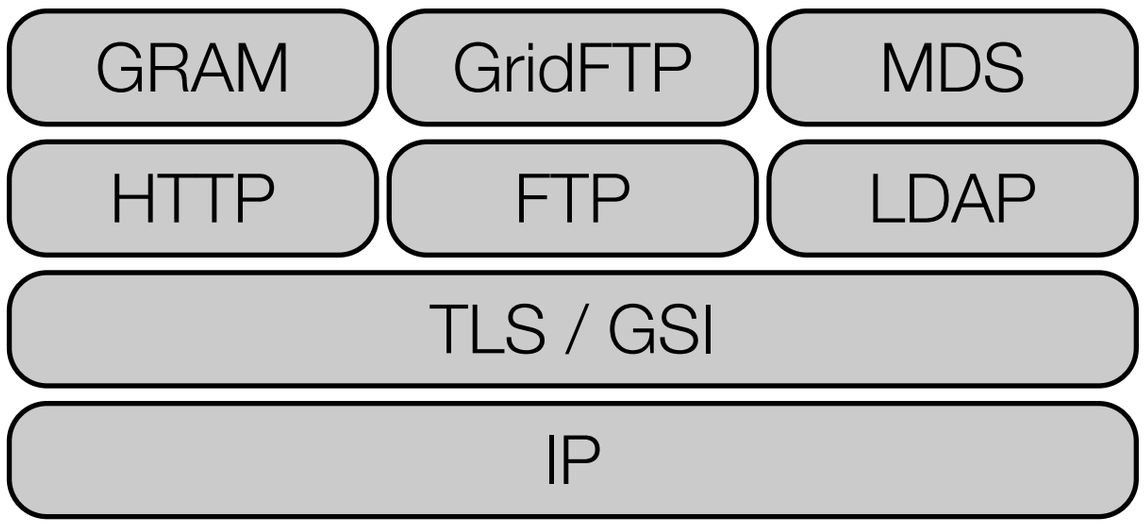
- Authentication and authorization
- Resource discovery and meta-scheduling
  - “List all of the Solaris machines with at least 16 processors and 2 Gig’s of memory available for me”
- Data access and movement
  - “Locate / create replica of this data, move it from host A to host B”
- Resource monitoring
- Intra-/Inter-Node communication
- Goal: Simplify development of distributed applications for existing heterogeneous execution environments

The Globus Toolkit

- Globus Project (Argonne National Laboratory)
- Open source, integrates several third-party projects (OpenSSL, RSA, Axis, OpenLDAP, wuFTP)
- De-facto standard for grid infrastructures, reference implementation for grid-related OGF / OASIS / W3C standards
- Offers building blocks and tools for developers and integrators
- All major Unix platforms, Windows as client; C, Perl, Java, Python API‘s
- Version 2 (1997) : Proprietary grid architecture
- Version 3 (2003) : OGSI-based architecture
- Version 4 (fall 2004) : WSRF-based architecture

Globus Toolkit Version 2 (GT2)

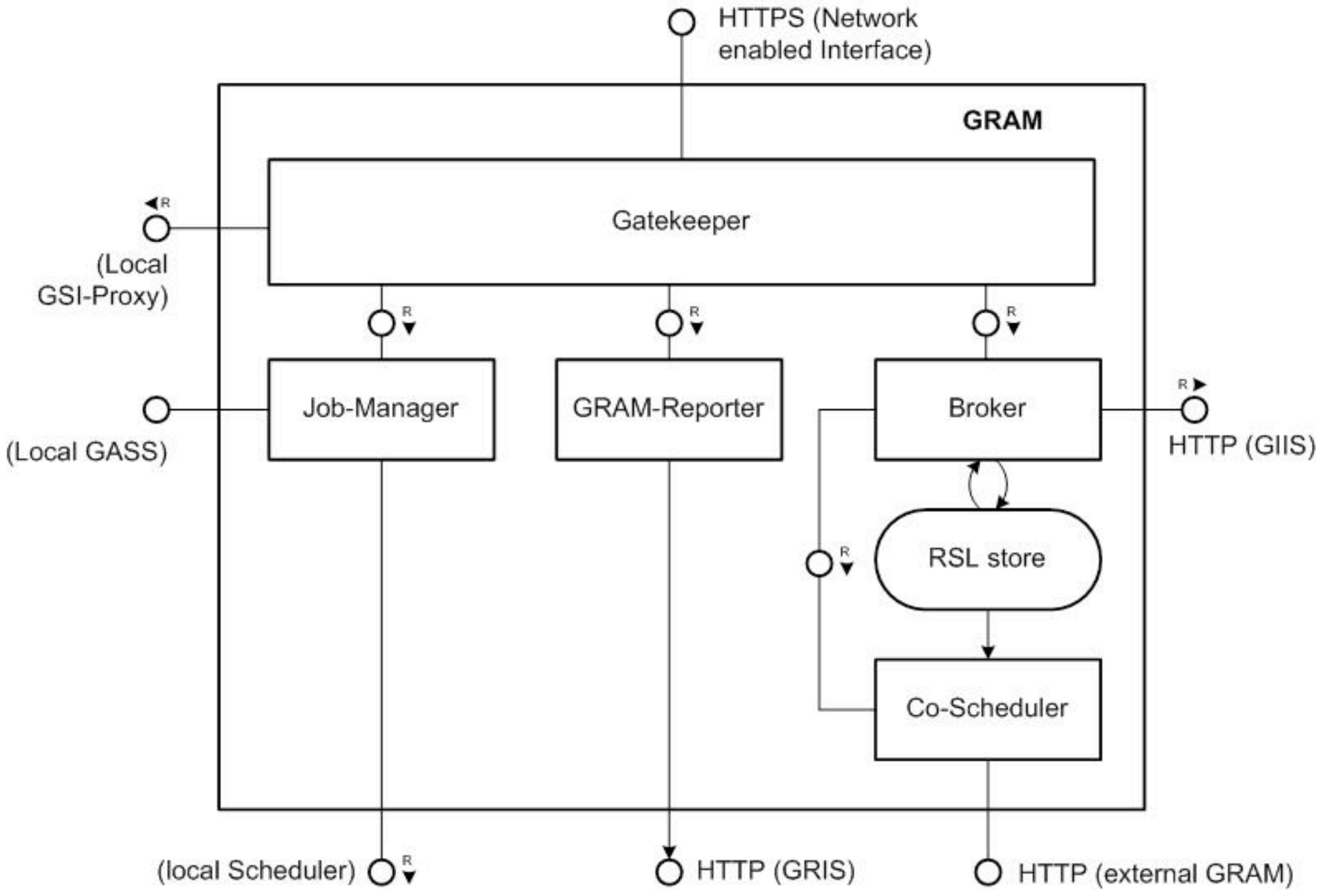
- Globus Resource Allocation Management (GRAM, RSL)
- Metacomputing Directory Service (MDS)
- Globus Security Infrastructure (GSI)
- Heartbeat Monitor (HBM)
- Globus Access to Secondary Storage (GASS, GridFTP)
- Communication (Nexus, MPICH-G2)



GRAM

- Translates generic resource request (RSL) into explicit commands for a set of resources (cluster, single machine)
- Gatekeeper
  - Frontend for all GT2 machines, performs security check and proxy validation
- Job Manager
  - Monitors and controls jobs on the resource (single machine, cluster)
  - Interacts with multiple local schedulers (Condor, LSF)
- GRAM reporter
  - Collects and manages system-specific resource information
  - Local GIS database is synchronized with LDIF to global GIIS

GRAM


HTTPS (Network enabled Interface)
GRAM
Gatekeeper
R
(Local GSI-Proxy)
R
R
R
Job-Manager
GRAM-Reporter
Broker
R
HTTP (GIIS)
(Local GASS)
RSL store
R
Co-Scheduler
(local Scheduler)
R
HTTP (GRIS)
HTTP (external GRAM)

Other GT2 Services

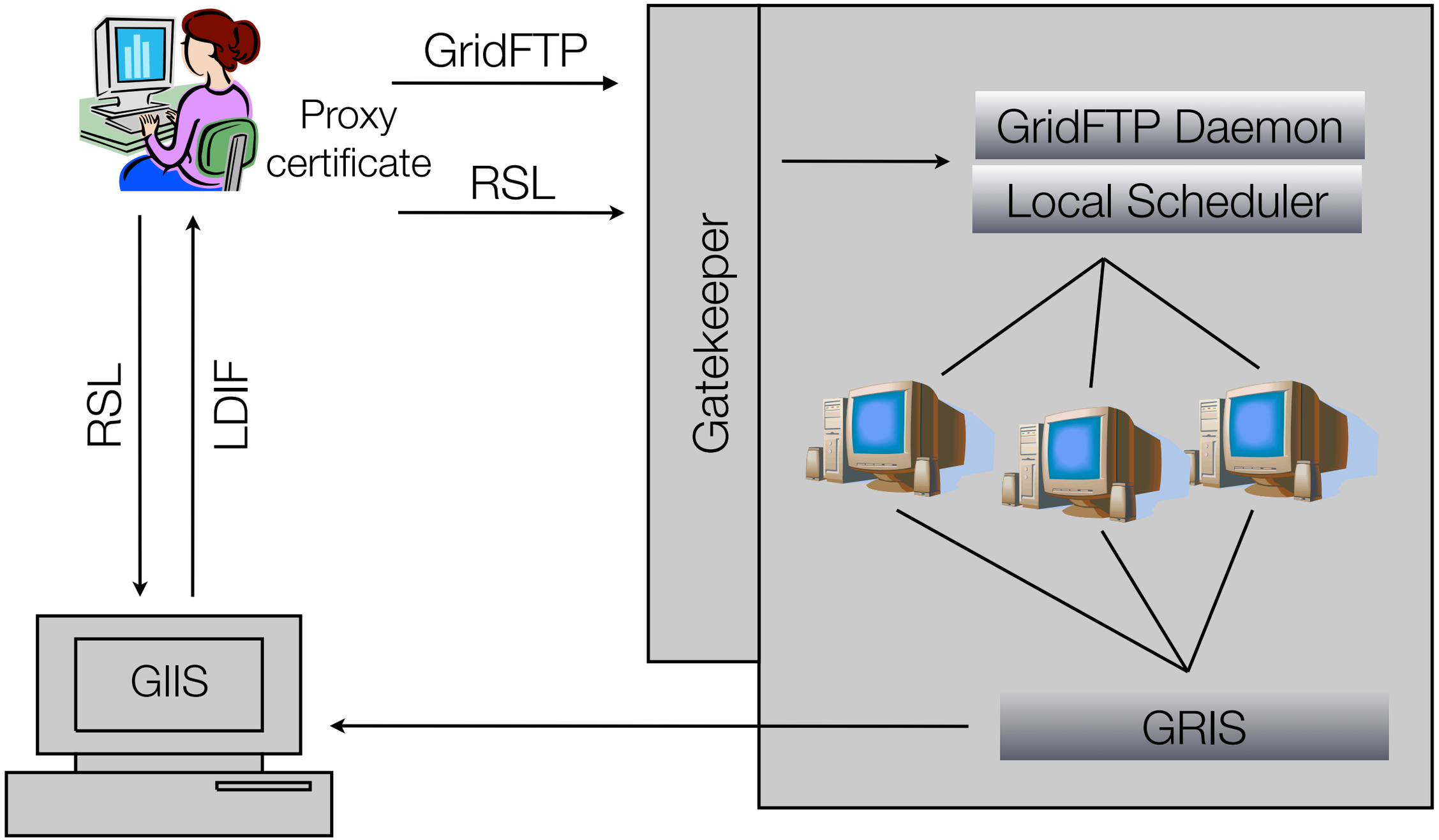
- Metadata Directory Service (MDS)
  - Hierarchical directory information tree, based on LDAP
  - Globus Resource Information Service (GRIS) installed on a grid node, supplies information about a specific resource
  - Globus Institution Indexing Server (GIIS) queries with RSL over HTTP
- GridFTP
  - Based on standard FTP protocol (RFC 2228)
  - Striped, partial, restartable and parallel file transfer, replica management
  - Security protocols on connectivity layer (GSI)

Resource Specification Language

```
& (rsl_substitution  = (TOPDIR  "/home/nobody")
                       (DATADIR $(TOPDIR)"/data")
                       (EXECDIR $(TOPDIR)/bin) )
  (executable = $(EXECDIR)/a.out
        (* ^-- implicit concatenation *))
  (directory  = $(TOPDIR) )
  (arguments  = $(DATADIR)/file1
        (* ^-- implicit concatenation *)
                $(DATADIR) # /file2
      (* ^-- explicit concatenation *)
                '$(FOO)'                (* <-- a quoted literal *))
(environment = (DATADIR $(DATADIR)))
(count = 1)
```

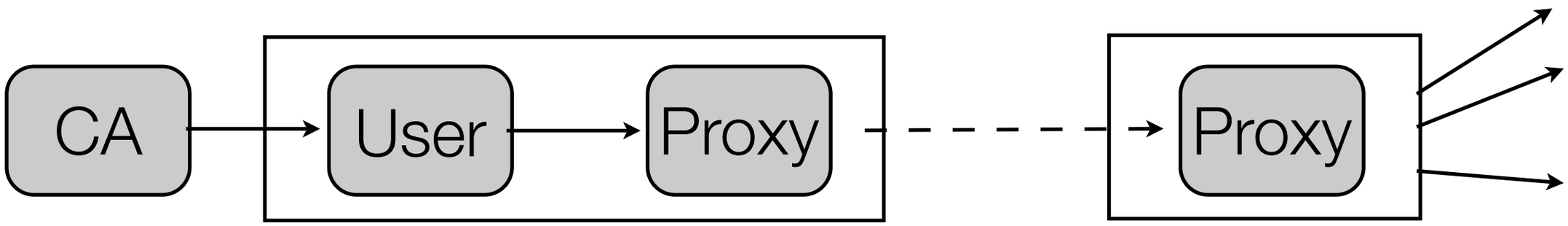
arguments	maxWallTime	
count	maxCpuTime	
directory	gramMyjob	project
executable	stdin	dryRun
environment	stdout	maxMemory
jobType	stderr	minMemory
maxTime	queue	hostCount

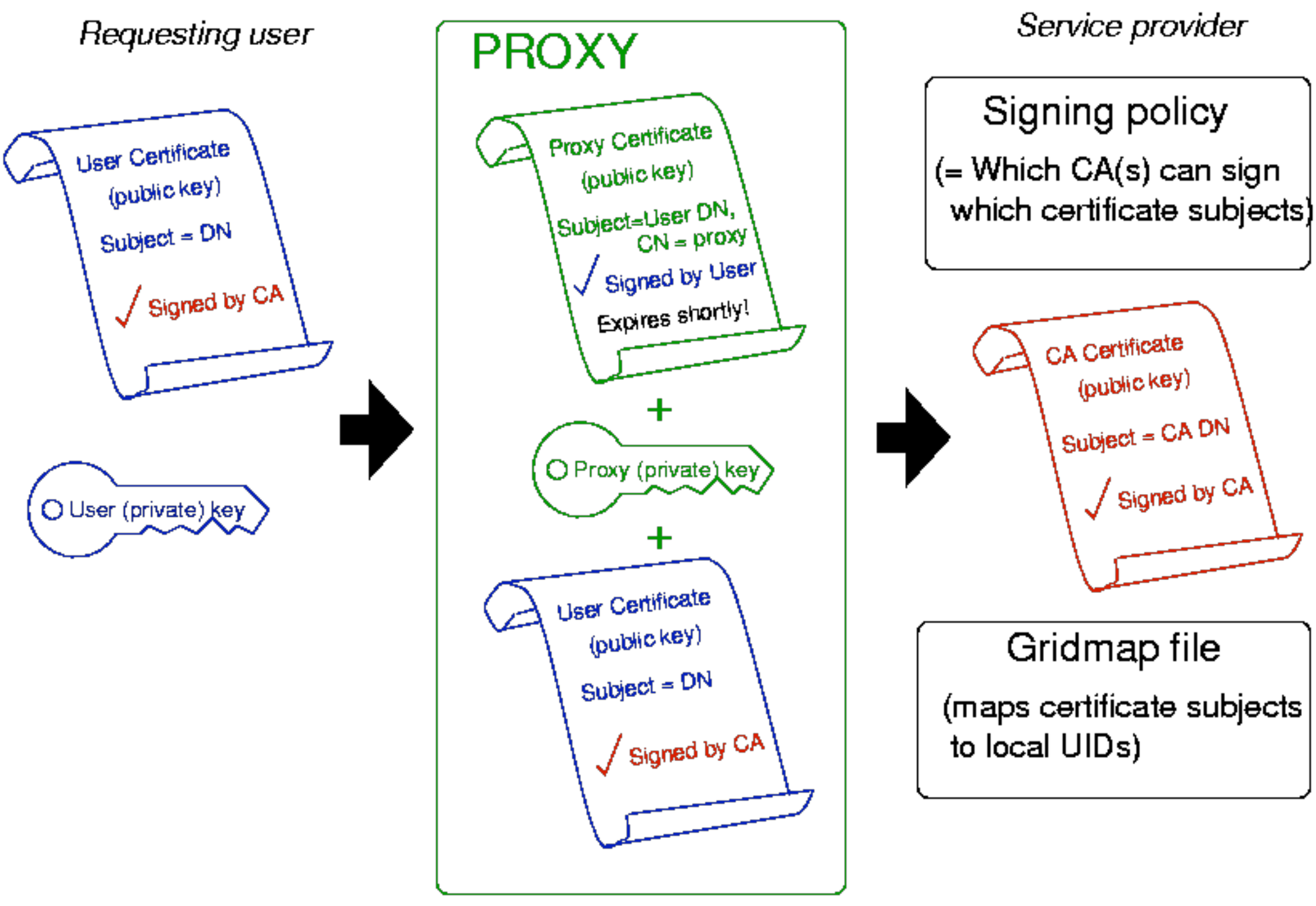
Globus 2 Architectur



Globus Security Infrastructure (GSI)

- Based on X.509 PKI
  - Security across organizational boundaries
  - Single sign-on and credential delegation
- RFC 3820 - Generation of proxy certificate credentials
  - Generation of short-lived proxy certificate, based on user credentials
  - New public / private key pair; subject field is the issuer field with additional single CN component
  - Special extension for policies, most X.509 implementations still work

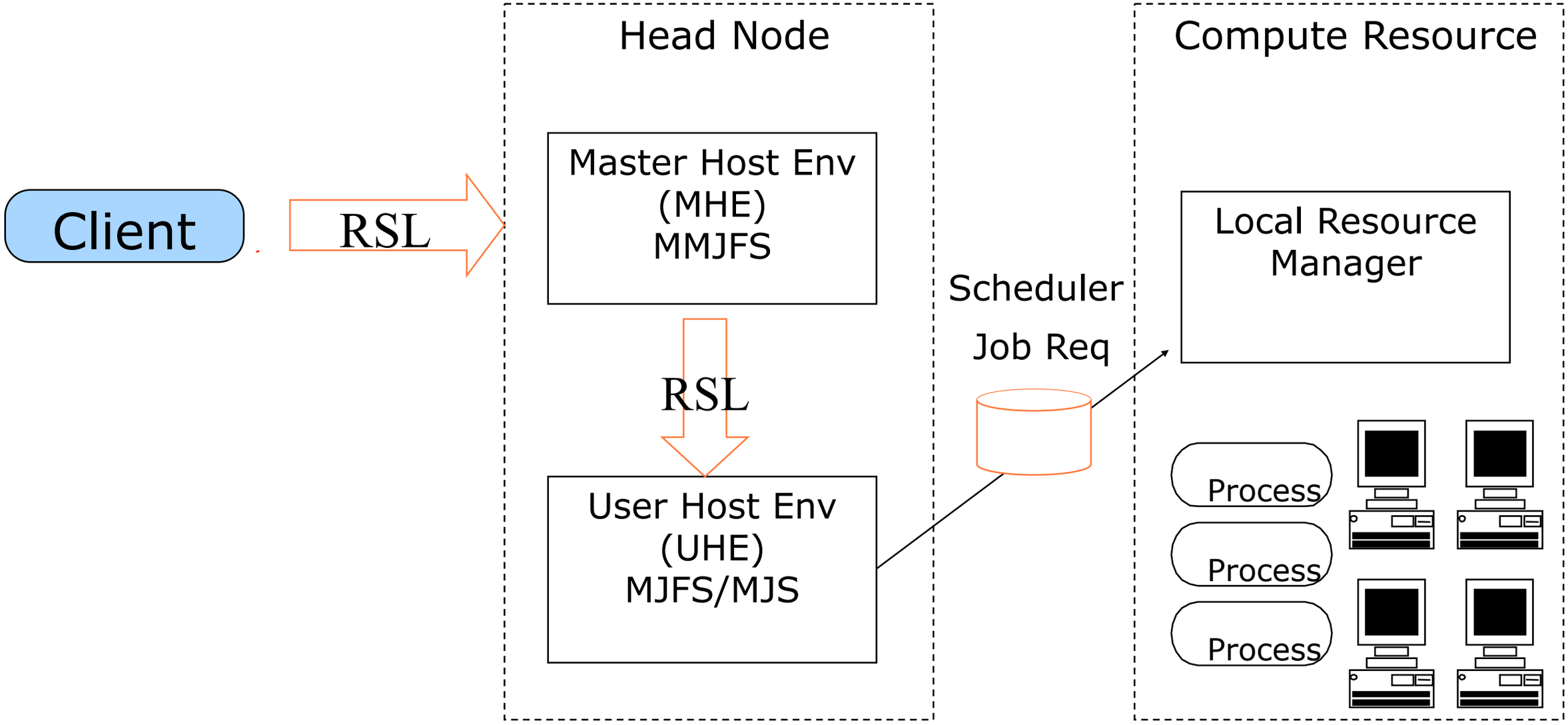



Requesting user
User Certificate
(public key)
Subject = DN
Signed by CA
User (private) key
PROXY
Proxy Certificate
(public key)
Subject=User DN,
CN = proxy
Signed by User
Expires shortly!
+
Proxy (private) key
+
User Certificate
(public key)
Subject = DN
Signed by CA
Service provider
Signing policy
(= Which CA(s) can sign
which certificate subjects)
CA Certificate
(public key)
Subject = CA DN
Signed by CA
Gridmap file
(maps certificate subjects
to local UIDs)

OGSA Service Model

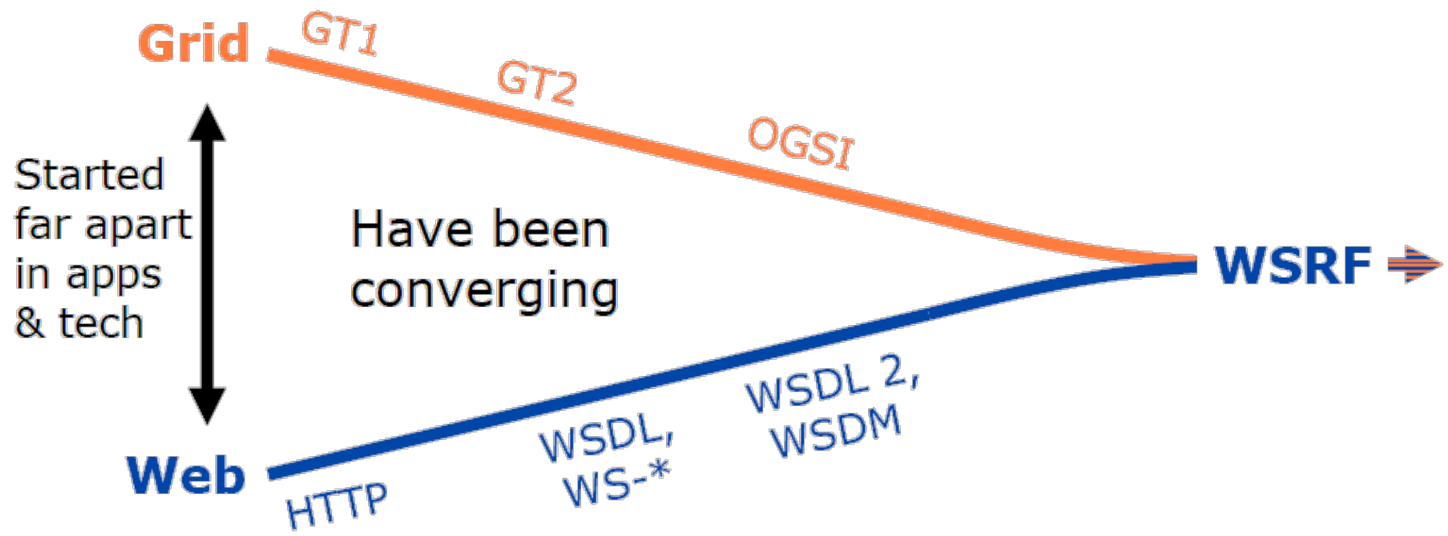
- Representing stateful grid resources over stateless SOAP
  - Grid node provides services, each job is modelled as service
- System comprises (a typical few) persistent services & (potentially many) transient services
- All services adhere to specific interfaces and behavior
  - Reliable invocation, lifetime management, discovery, authorization, notification
- Interfaces for managing Grid service instances
  - Factory, registry, discovery, lifetime
- Globus 3: Open Grid Services Infrastructure (OGSI)
  - proprietary WSDL portType extensions

Managed Job Submission in GT3



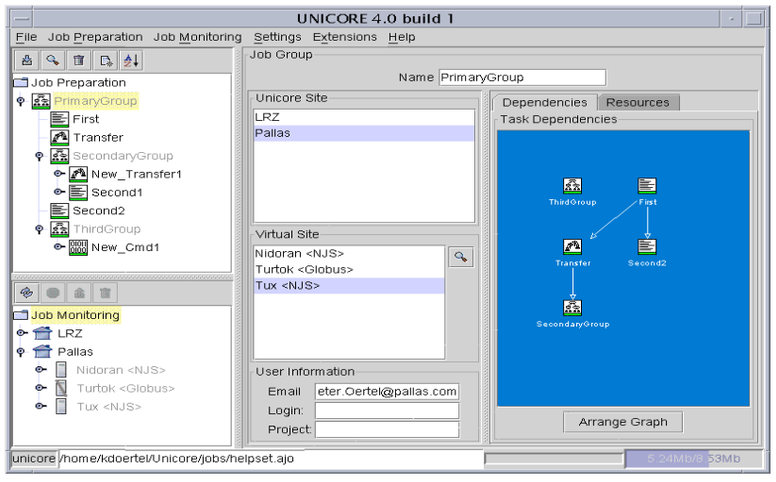
Web Services Resource Framework

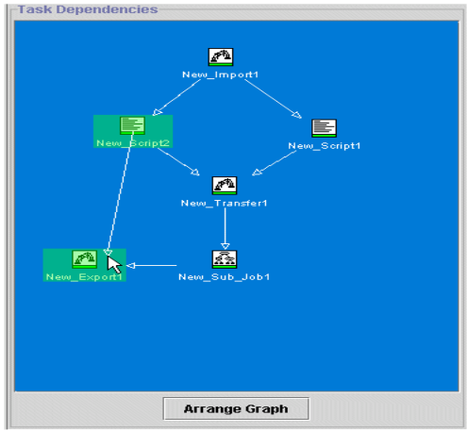
- Refactoring of OGSI
  - Based on set of industrial standards (*WS-Addressing, WS-ResourceNotification, WS-ResourceLifetime, WS-ServiceGroup*)
  - Initiative from major grid companies (IBM, Fujitsu, Sun, HP)
- No paradigm shift, implementation in Globus 4.X

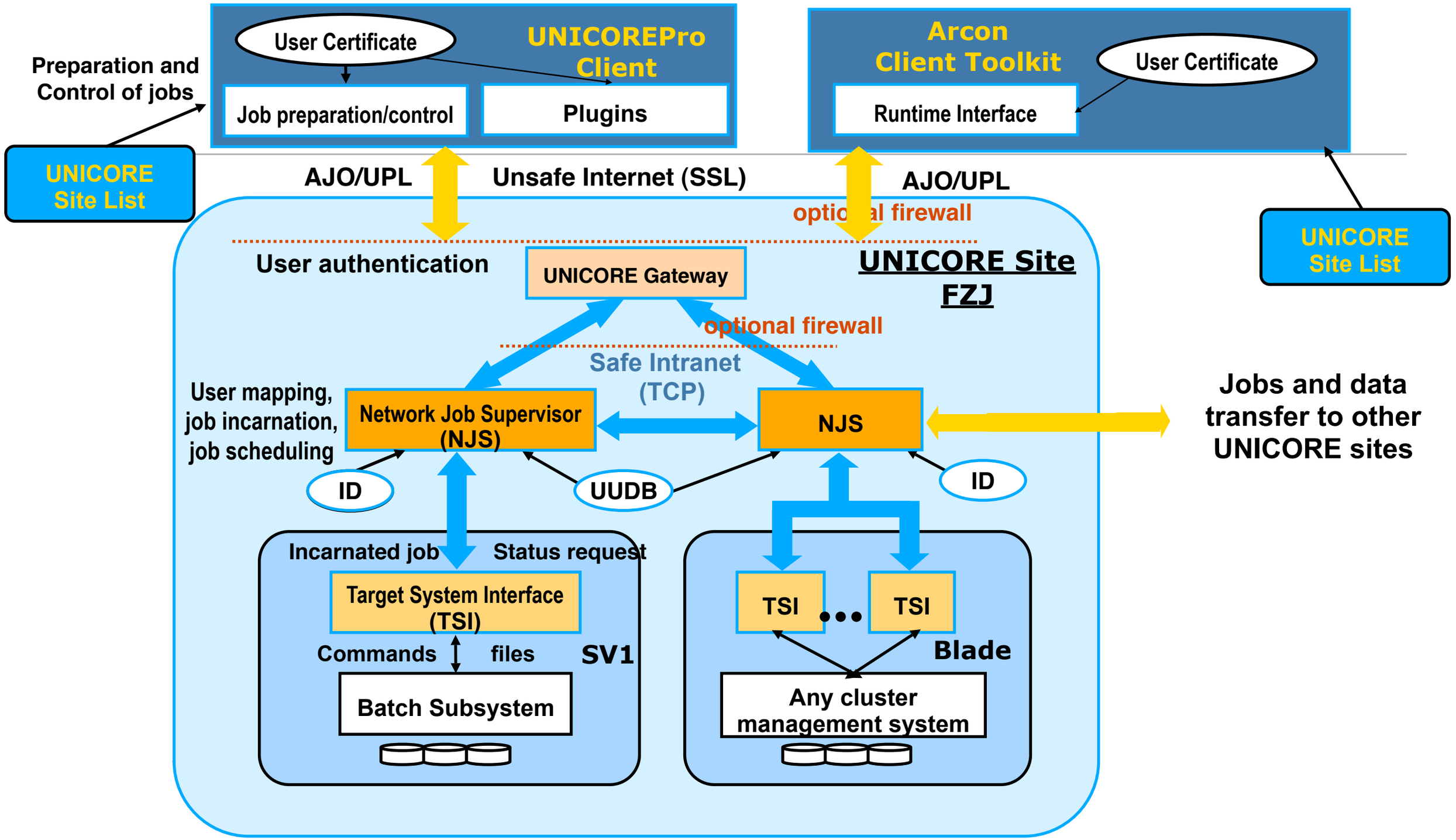


Other Grid Middleware

- UNICORE: (Uni)form interface to (co)mputer (re)sources
  - BMBF project, FZ Jülich, 5 german universities, first version in 2000
  - Abstract job concept, job submission GUI
- GridLab
  - Funded by the EU (5+ M€), January 2002 – March 2005
  - Application and Testbed oriented (e.g. Cactus code, Triana Workflow)



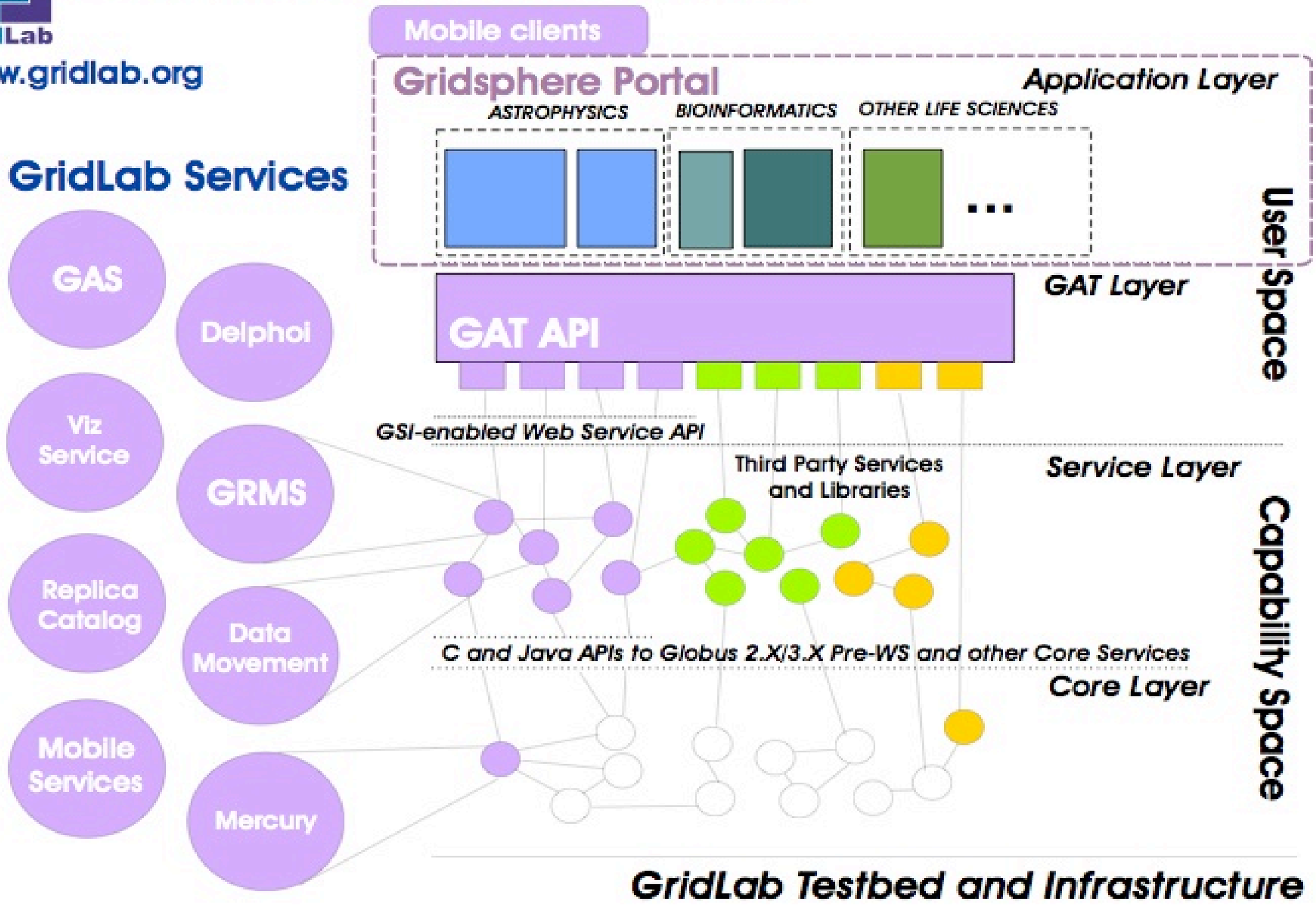



Preparation and Control of jobs
UNICORE Site List
User Certificate
UNICOREPro Client
Job preparation/control
Plugins
Arcon Client Toolkit
User Certificate
Runtime Interface
UNICORE Site List
AJO/UPL
Unsafe Internet (SSL)
AJO/UPL
optional firewall
User authentication
UNICORE Gateway
UNICORE Site FZJ
optional firewall
Safe Intranet (TCP)
User mapping, job incarnation, job scheduling
Network Job Supervisor (NJS)
NJS
Jobs and data transfer to other UNICORE sites
ID
UUDB
ID
Incarnated job
Status request
Target System Interface (TSI)
Commands
files
SV1
Batch Subsystem
TSI
TSI
Blade
Any cluster management system

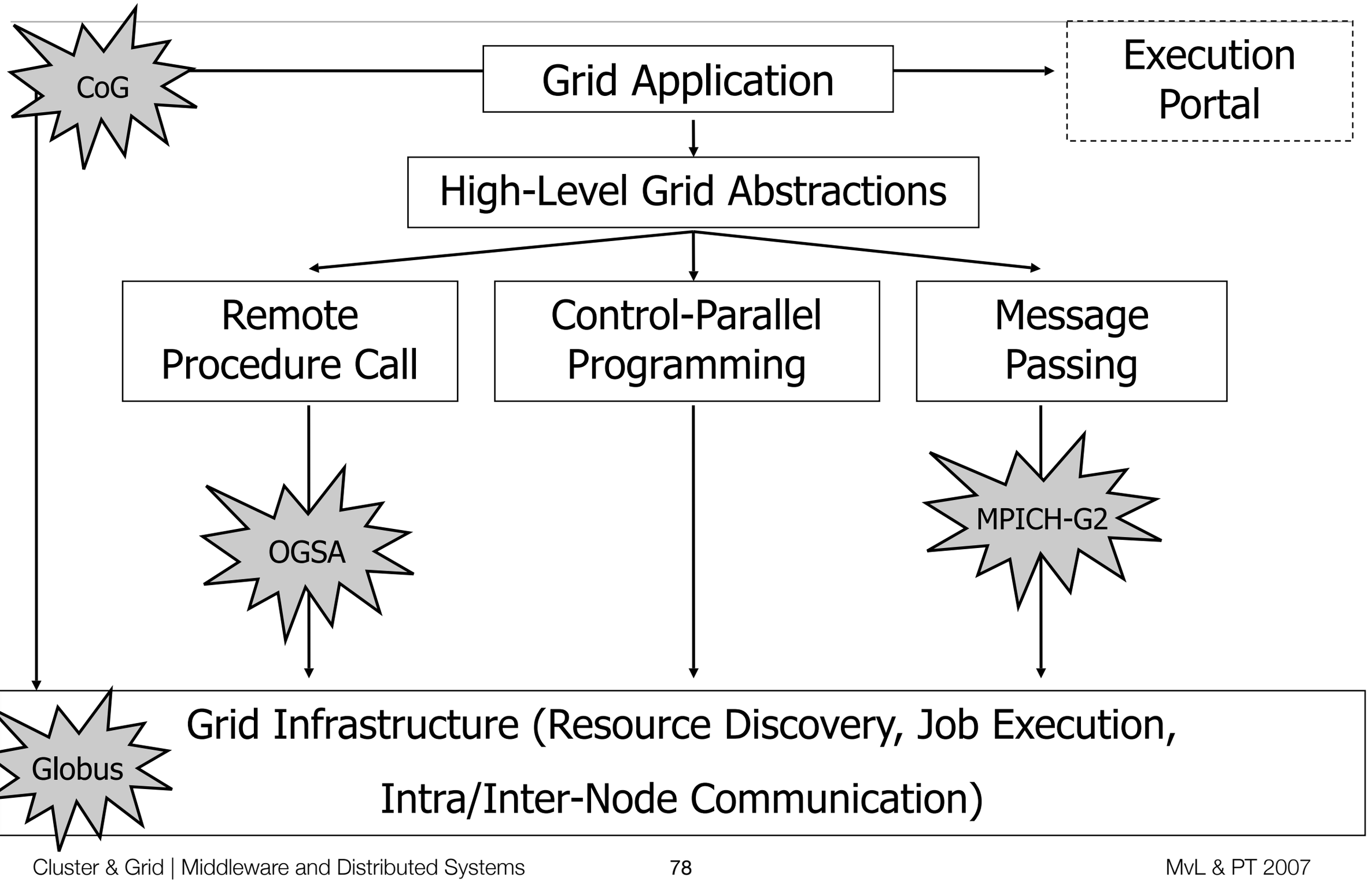
GridLab Architecture

GridLab

www.gridlab.org



Building a Grid Application



AMWAT

- AppLeS Master/Worker Application Template
- Library covers remote process spawning, work assignment, load balancing, failure handling and result gathering
- Multi-layered master-worker schema
  - Distribution of application-defined work units
  - One work cycle computes multiple work units
- Application adapter implements the functions expected by the scheduler
- IPC through sockets, shared memory, MPICH, PVM and Globus

AMWAT Application Interface

```
typedef int AMWATApp_Cycle
typedef int AMWATApp_WorkUnit

int AMWATApp_InitializeCycle(AMWATApp_Cycle cycle,
     int generateWork, unsigned *workUnitsCount)

AMWATApp_ComputeOutcome AMWATApp_Compute(
     AMWATApp_WorkUnit unit)

Int AMWATApp_ReceiveWorkUnits(AMPIC_PeerId whoFrom,
     AMWATApp_WorkUnit *workUnits,
     unsigned workUnitsCount,
     AMPIC_MessageKind messageKind)
```

Grid-Occam

- Bring parallelism as first-level language construct to modern distributed environments
    - Consistent programming model for different granularities of distribution (threads, cluster nodes, grid nodes)
- Support for heterogeneous execution platforms
    - Rely on virtual execution environments (Java, .NET / Mono)
- Focus on coordination aspect (no HPC here)
- Clear distinction of Occam compiler from infrastructure-dependent runtime library - multithreaded runtime library / MPI runtime library
- Support nested nature of granularity levels

Occam History

- Parallel processing language
- Abstraction from underlying hardware, software and network environment
- Based on Sir T. Hoare's ideas of Communicating Sequential Processes (CSP)
- Developed by INMOS for transputer systems
  - Distributed memory system
  - 4 high-speed hardware links
  - Routing of messages for unconnected processors (virtual channel router)

Occam Language

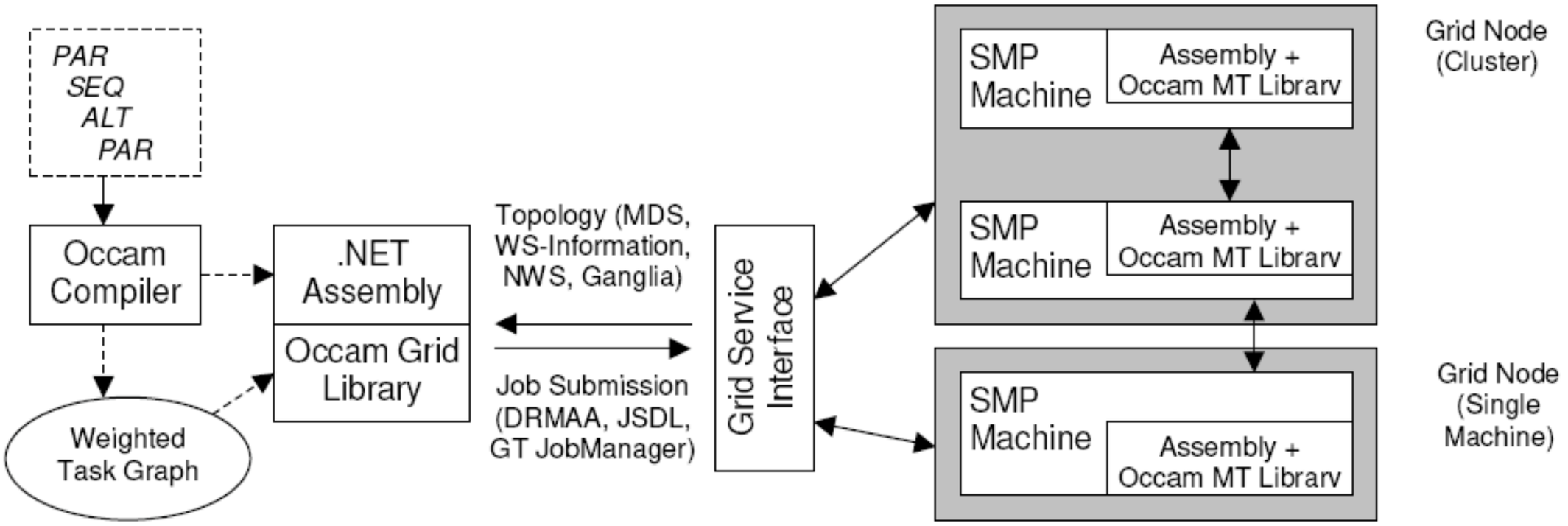
- Primitive actions
  - Variable assignment
  - Channel output
  - Channel input
  - SKIP
  - STOP
- Sequential process combination (SEQ)
- Parallel process combination (PAR)

```
PROC hello()
  INT x,y:
  CHAN OF INT c,d:
  PAR
    SEQ
      c ! 117
      d ? x
    SEQ
      c ? y
      d ! 118
:
```

```
ALT
  keyboard ? var1
    to.computer ! var1
  from.computer ? var1
    display ! var1
```

Occam Rules

- Rendezvous behavior of channels
  - Receiver blocks until the sender wrote the value
  - Sender continues after the receiver read the value
- Variables can only be written by one process in parallel



DRMAA - Problem Statement

- Product-specific APIs for job submission to distributed resource management (DRM) systems
  - Condor (Perl, WS, GAHP), Torque (C, Python, Perl), SGE 5 (GDI), XGrid (Cocoa), Globus (CoG), ...
    - Adapter concept in each grid middleware and grid portal kit
    - Usually focus on command-line tools
- Demand for standardized interface
  - Job submission, control, and monitoring
  - Application should work with any DRMS without change (or even recompilation)

DRMAA

- **D**istributed **R**esource **M**anagement **A**pplication **A**PI
  - Open Grid Forum (OGF) working group
  - Numerous contributors from academia and industry
  - Started in 2002
- Language-independent specification
  - Proposed recommendation spec in 2004
  - Full recommendation submitted in 2007
- Stable language binding specifications for C, Java and Perl
  - Mappings for C#, Python, and Ruby

Design Principles

- Keep it simple
  - Balancing act between feature demands, DRM compatibility, and simple API design
  - Lower entrance barrier for adopters
    - Maximum compatibility (list user jobs, workflows)
- Language independence
  - Procedures with input and output parameters, returning error code
- Support for multiple DRM systems in one application

Design Principles

- No security
  - DRM systems all have there own platform-specific solutions (uid/gid, SID, Kerberos token, X.509 certificate)
- Thread safety
  - Put the burden on the DRMAA library developer (==vendor)
- Site-specific policies
  - Cross-site behaviors not covered by the specification (e.g. MPI jobs)
  - Mark jobs as member of a class of applications, site administrator regulates according rules (e.g. queues)

API Basics

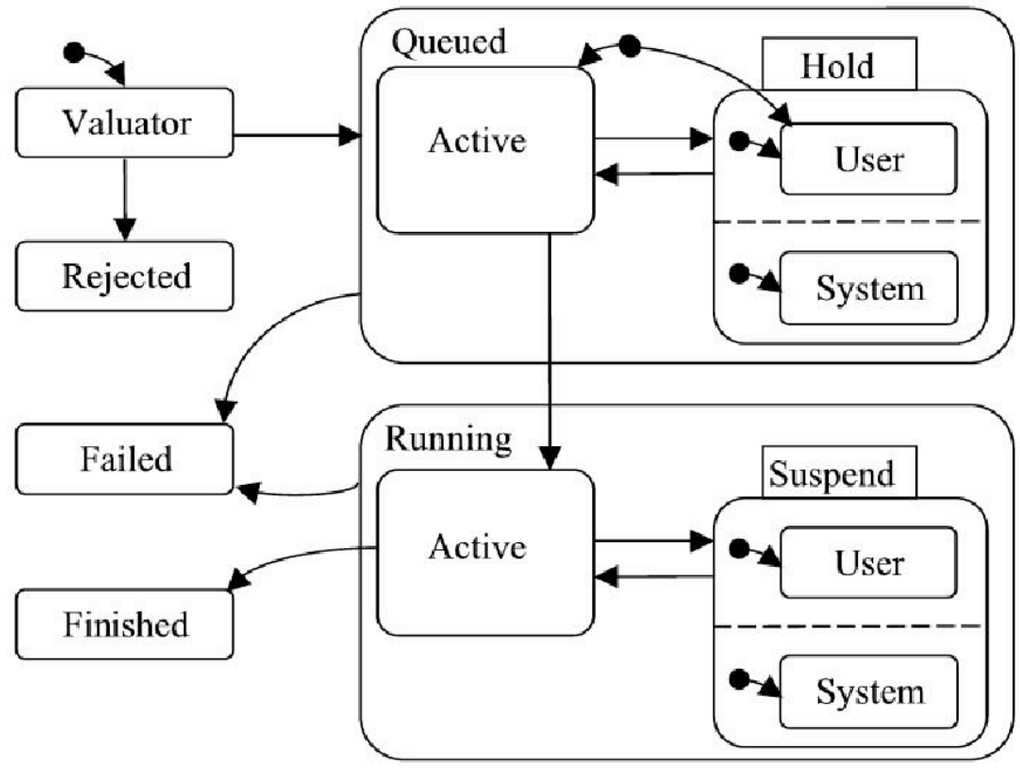
- Init and exit routine
- Session concept for collective monitoring and control
    - Ensure that DRMS jobs are no longer influenced by library
    - Unusual for garbage collection languages, but still problem of unpredictable finalizers
    - Practical demand for persistent sessions (unreliable software, long-running job workflows)
- Job template routines, job submission and monitoring routines, job control routines, auxiliary routines

Job Templates

- Set of key-value pairs, describing job requirements and parameters
- Mandatory, optional, and implementation-specific attributes
  - Least common denominator for mandatory attributes (executable, input / output stream, ...)
  - Optional attributes, e.g. job termination time, wall-clock limit
- From (very) early analysis and N1GE feature set
  - Most implementations ignore the optional attributes
- Evaluated at submission time
  - Might be rejected because of errors or policies
  - Only consistency errors for setter routines

Job Control / Monitoring

- Single job or bulk job submission with index
- Job status query
  - Considers temporary non-availability of status information
- Control routine (suspend, resume, hold, release, terminate)
- Synchronization with job(s) ending
  - With / without timeout, single / all jobs from session
- POSIX-like calls with job termination information



```
#include "drmaa.h"
...
int main(int argc, char *argv[]) {
   ...
   if (drmaa_init(NULL, diag, dsize) != DRMAA_ERRNO_SUCCESS) {...}
   /* create job template */
   if (drmaa_allocate_job_template(&jt, NULL, 0) != DRMAA_ERRNO_SUCCESS) {...}
   drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, '/bin/foo', NULL, 0);
   drmaa_set_attribute(jt, DRMAA_OUTPUT_PATH,DRMAA_PLACEHOLDER_INCR, NULL, 0);
   drmaa_set_attribute(jt, DRMAA_JOIN_FILES, "y", NULL, 0);

   /* submit 5 bulk jobs, with index increment by 1 */
   if (drmaa_run_bulk_jobs(&ids, jt, 1, 5, 1, diag, dsize)) != DRMAA_ERRNO_SUCCESS) {...}

   /* synchronize with job ending of all jobs */
   if (drmaa_synchronize(DRMAA_JOB_IDS_SESSION_ALL, DRMAA_TIMEOUT_WAIT_FOREVER, 0, diag,
   dsize) != DRMAA_ERRNO_SUCCESS) {...}
```

```
/* Analyze exit information */
for(int pos=0;pos<5;pos++) {
   if (drmaa_wait(JOB_IDS_SESSION_ANY, jobid, sizeof(jobid)-1, &stat,
                  DRMAA_TIMEOUT_WAIT_FOREVER, NULL, diag, dsize) != DRMAA_ERRNO_SUCCESS)
                  {...}
   drmaa_wifaborted(&aborted, stat, NULL, 0);
   if (aborted) {printf("job \"%s\" never ran\n", jobid);}
   else {
      drmaa_wifexited(&exited, stat, NULL, 0);
      if (exited) {
         drmaa_wexitstatus(&exit_status, stat, NULL, 0);
         ...
      } else {
         drmaa_wifsignaled(&signaled, stat, NULL, 0);
         ...
}}}
/* cleanup ... */
```

IDL Specification

- Original specification with procedural C-language slant
- Independent Java / C# bindings, collapsed to one OO-like IDL specification
  - ‚n-version standardization‘
    - Variable and method naming, job template mapping
    - Language-binding author maps to IDL constructs
      -> one-page binding specification
- Parallel document to official DRMAA 1.0 spec
  - OGF document process must be considered
  - Language bindings already started to rely on IDL description