Leveraging Modern Interconnects for Parallel Job Scheduling

Eitan Frachtenberg, Hebrew University

with

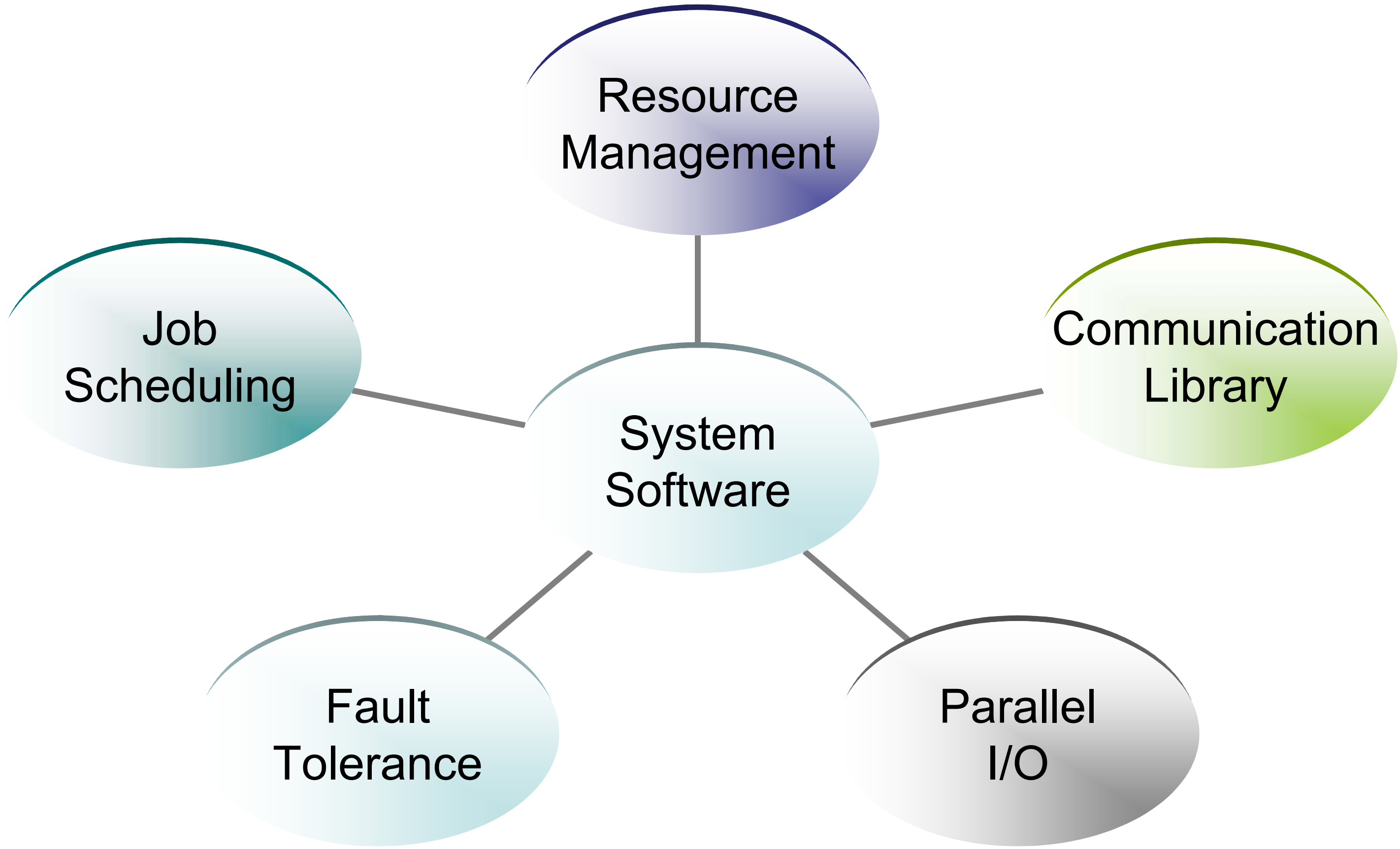
Dr. Dror Feitelson, Hebrew University

Dr. Fabrizio Petrini, Juan Fernandez, LANL

Cluster Supercomputers

- Growing in prevalence and performance, 7 out of 10 top supercomputers
- Dedicated to a small set of mission-critical problems, highly synchronous
- Advanced, high-end interconnects

System Software Components

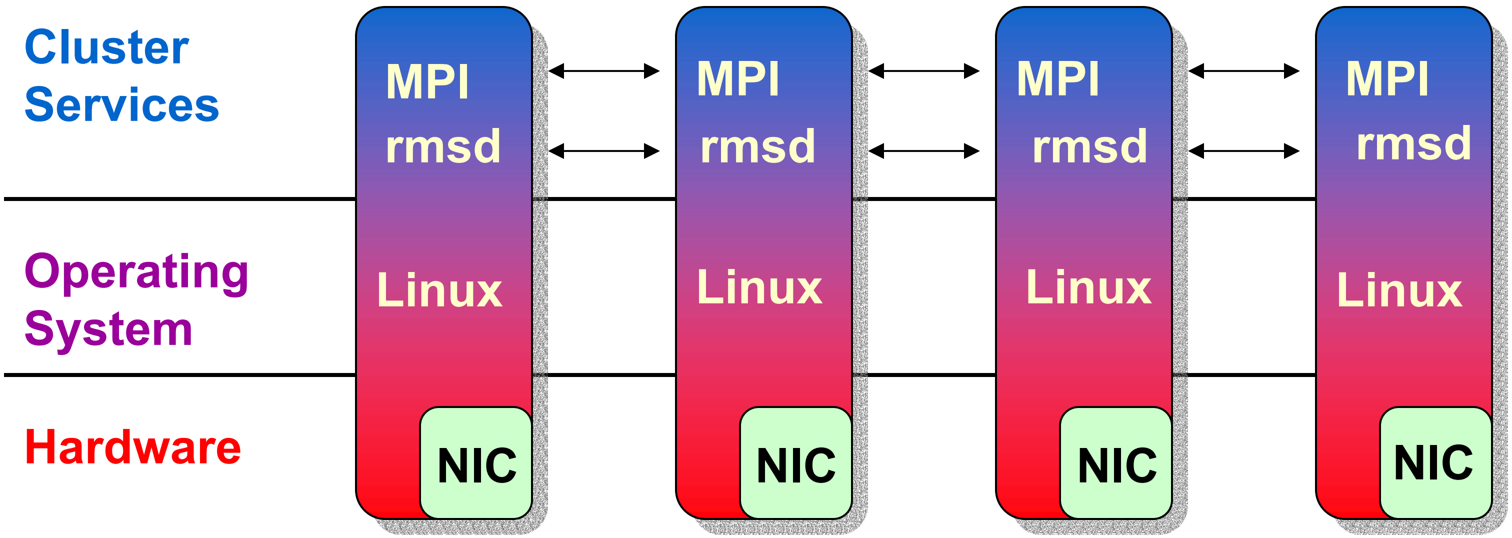


Cluster System Software

Typically composed of single-node OS (e.g. Linux) and connected by sets of daemons

Cluster System Software

Typically composed of single-node OS (e.g. Linux) and connected by sets of daemons



Problems with System Software

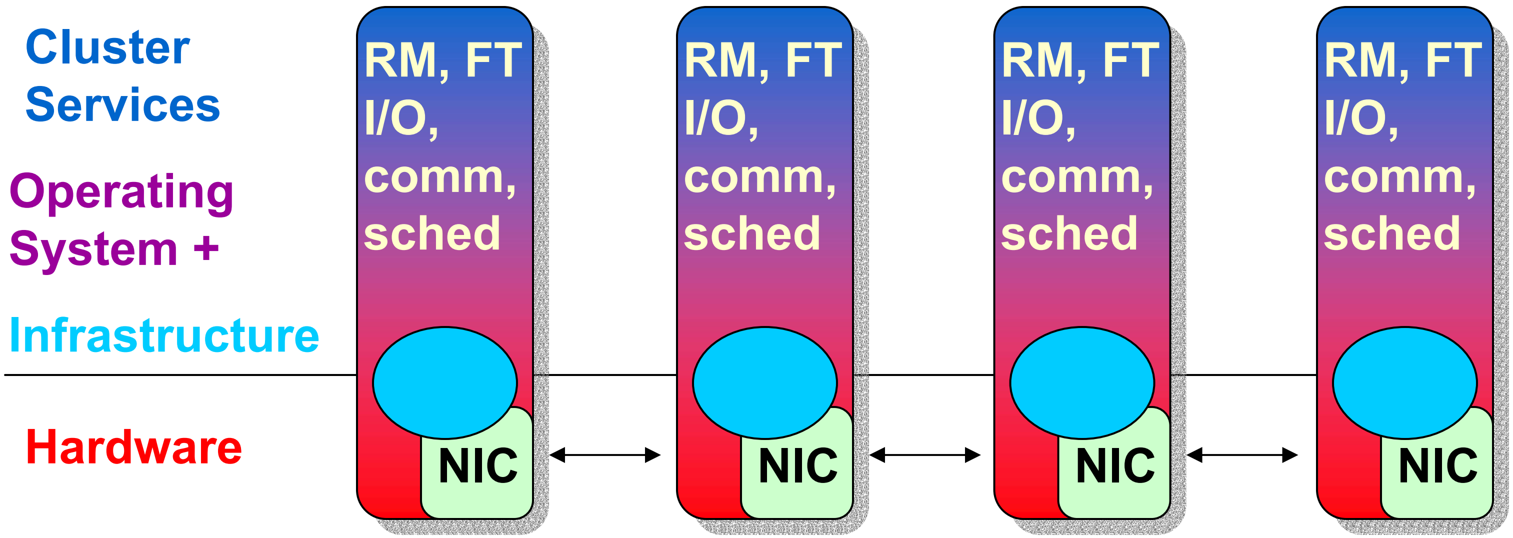
- Redundant components
- Performance hits
- Scalability issues

The Vision

- Modern interconnects offer powerful collective operations, programmable NICs and on-board RAM
- Use a small set of network mechanisms to create a common infrastructure, a parallel application in itself
- Build upon this infrastructure to create unified system software
- System software Inherits scalability and performance from network features

The Vision

All system software based on a common infrastructure, using network primitives



The Mechanisms

I. XFER-AND-SIGNAL
   - Multicast from a local address to global addresses
   - Optionally signal sender and/or receivers
   - Collective operation

II. COMPARE-AND-WRITE
   - Compare Boolean expression (<,=,≠,…) on set of nodes
   - Collective operation

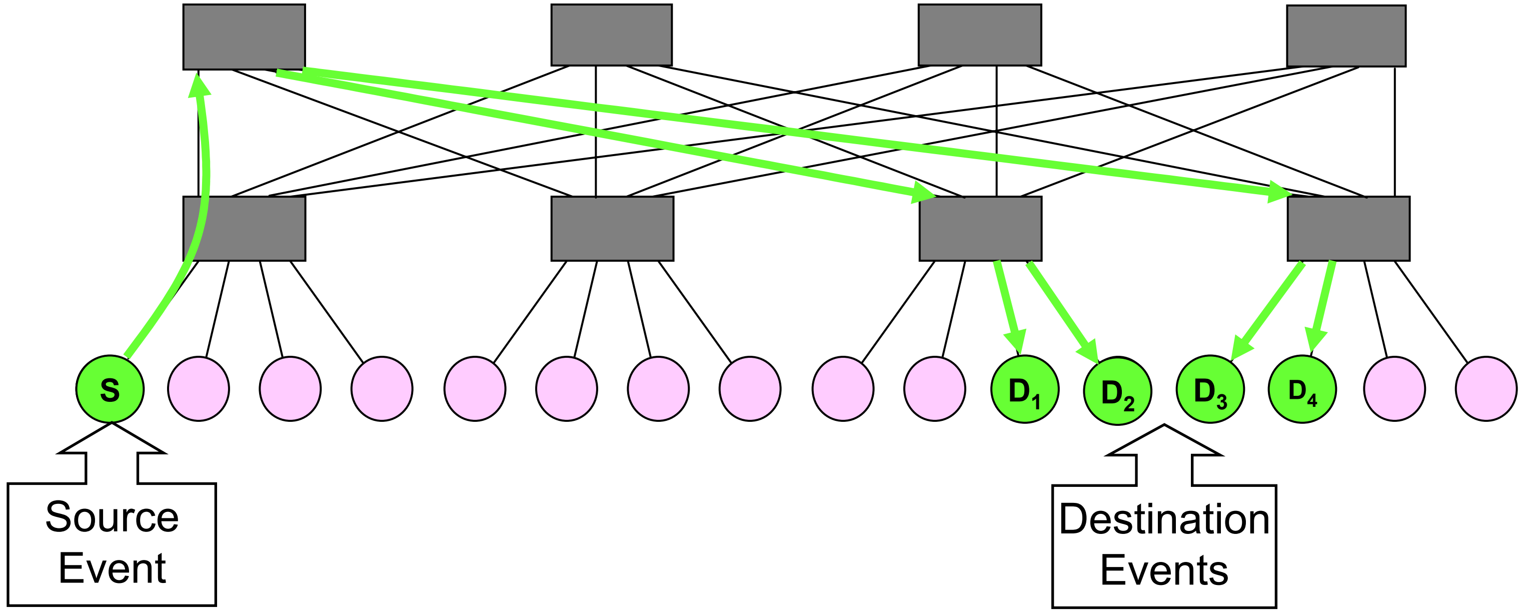
III. TEST-EVENT
   - Blocking test for completion of XFER-AND-SIGNAL
   - Local operation

Mechanisms and Network

- Performance and scalability of system software relies on performance and scalability of these mechanisms
- Simple implementation on top of QsNet HW primitives
- Other modern networks (Infiniband, Myrinet, BG/L) offer at least partial support

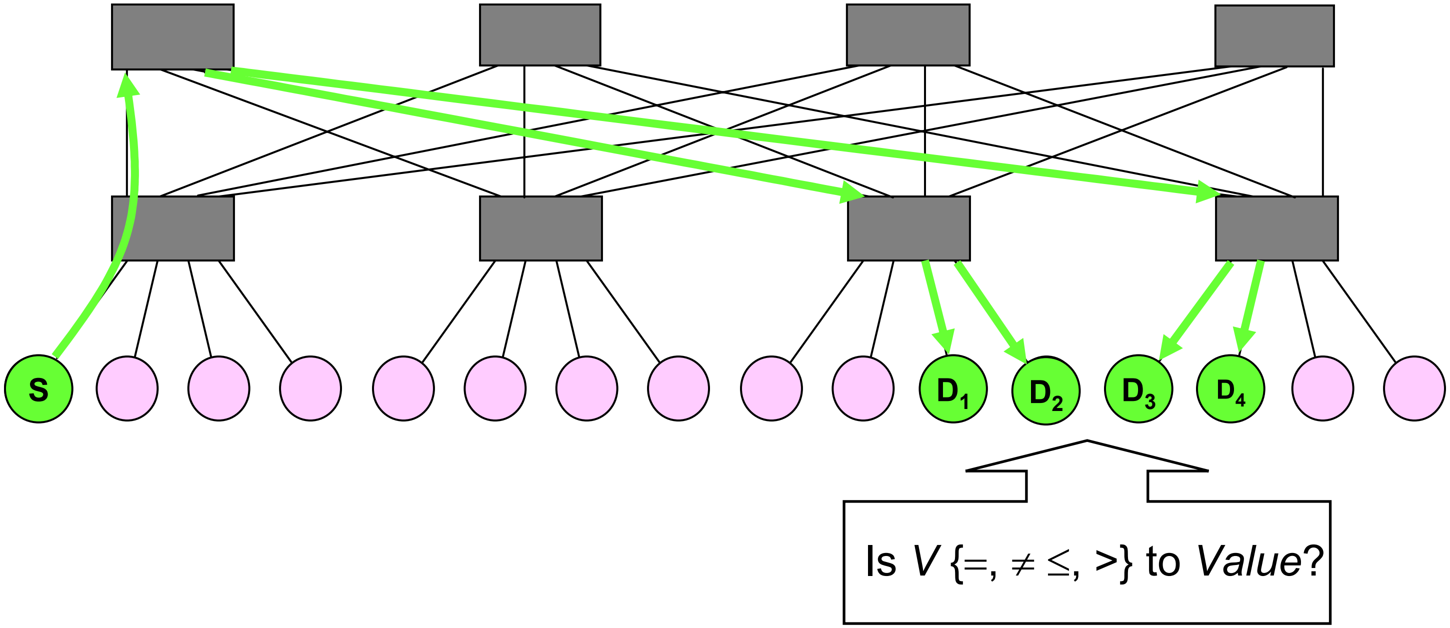
QsNet Collectives [Micro'02]

- Switches can combine answers by AND
- Example: COMAPRE-AND-WRITE
  - Node S transfers block of data to nodes D_1, D_2, D_3 and D_4
  - Events triggered at source and destinations



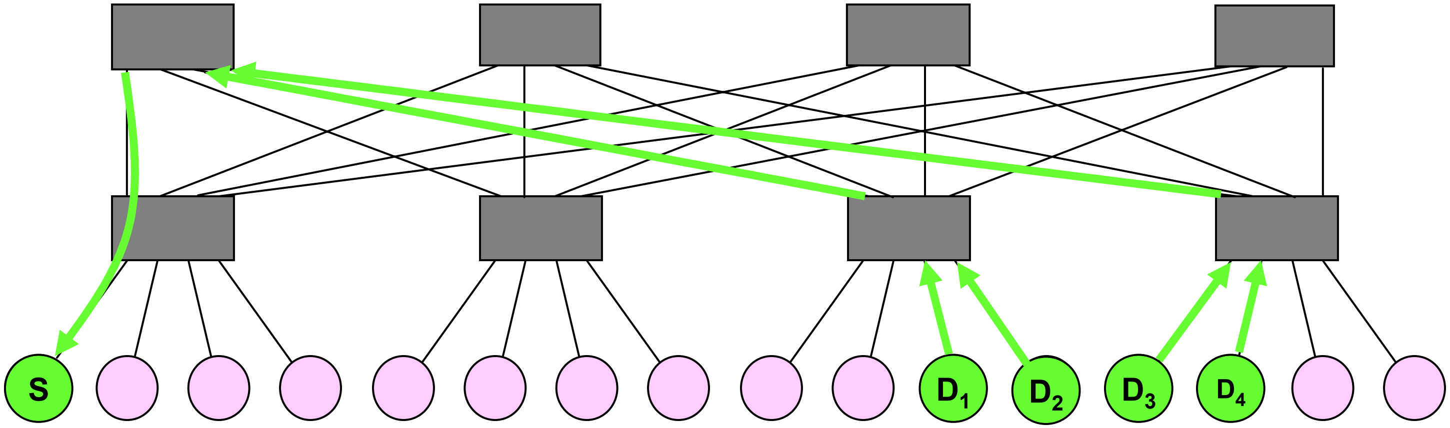
QsNet Collectives [Micro'02]

- Switches can combine answers by AND
- Example: COMAPRE-AND-WRITE
  - Node S transfers block of data to nodes D_1, D_2, D_3 and D_4
  - Events triggered at source and destinations



QsNet Collectives [Micro'02]

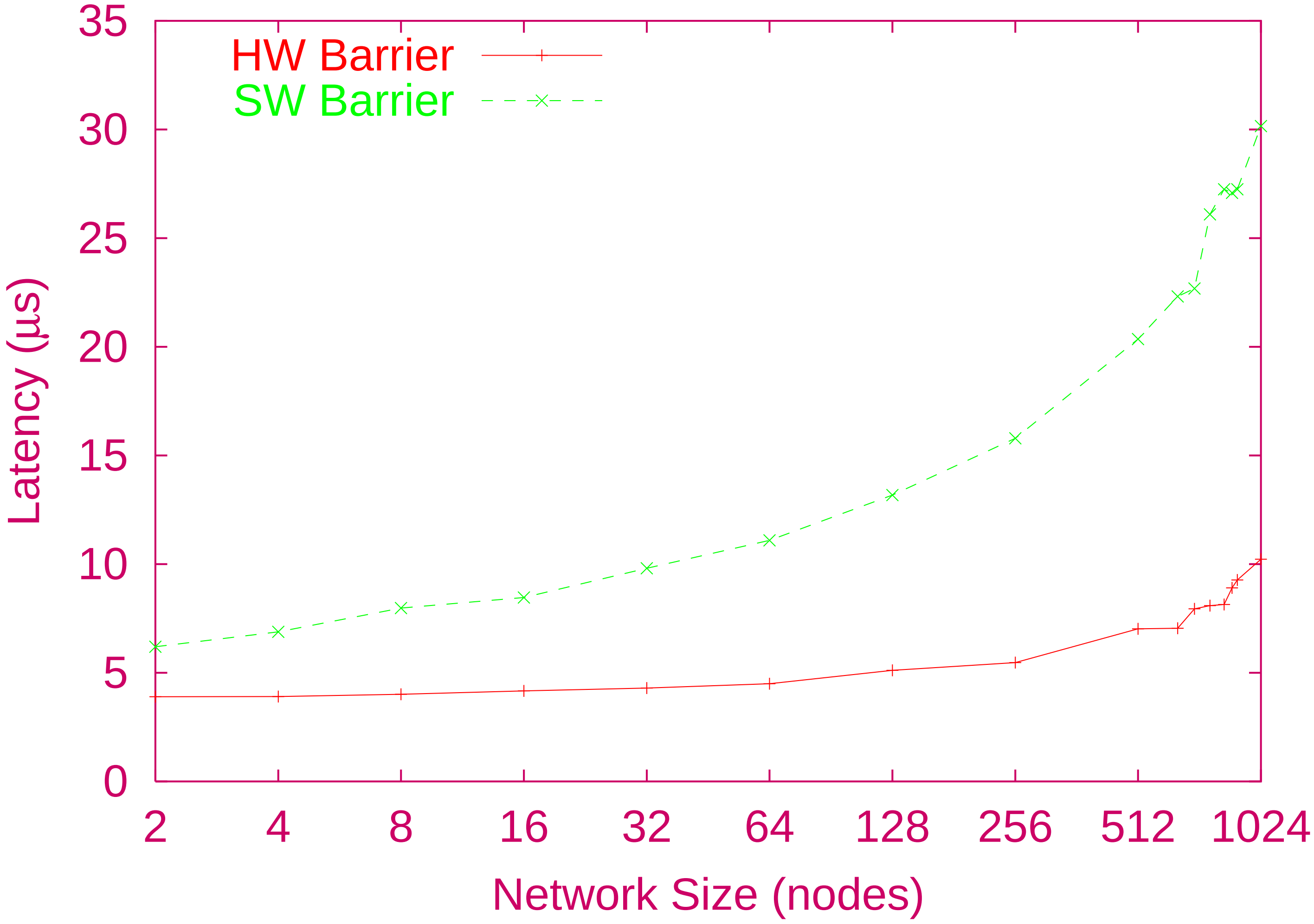
- Switches can combine answers by AND
- Example: COMAPRE-AND-WRITE
  - Node S transfers block of data to nodes D_1, D_2, D_3 and D_4
  - Events triggered at source and destinations



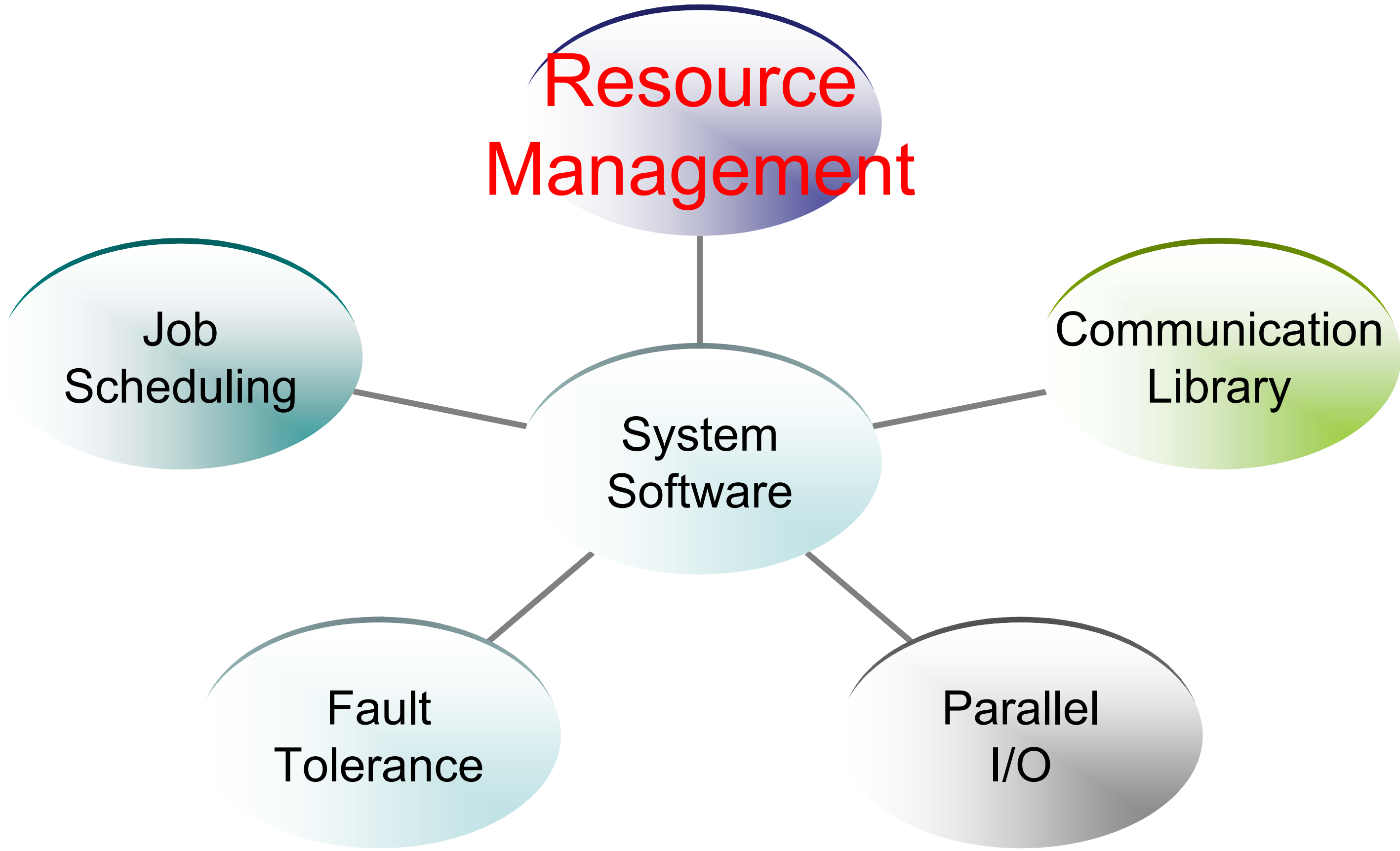
Example: ASCI Q

World's #2 Supercomputer at Los Alamos

Example: ASCI Q Barrier [HotI'03]



System Software Components



Resource Management

Scalable Tool for Resource Management

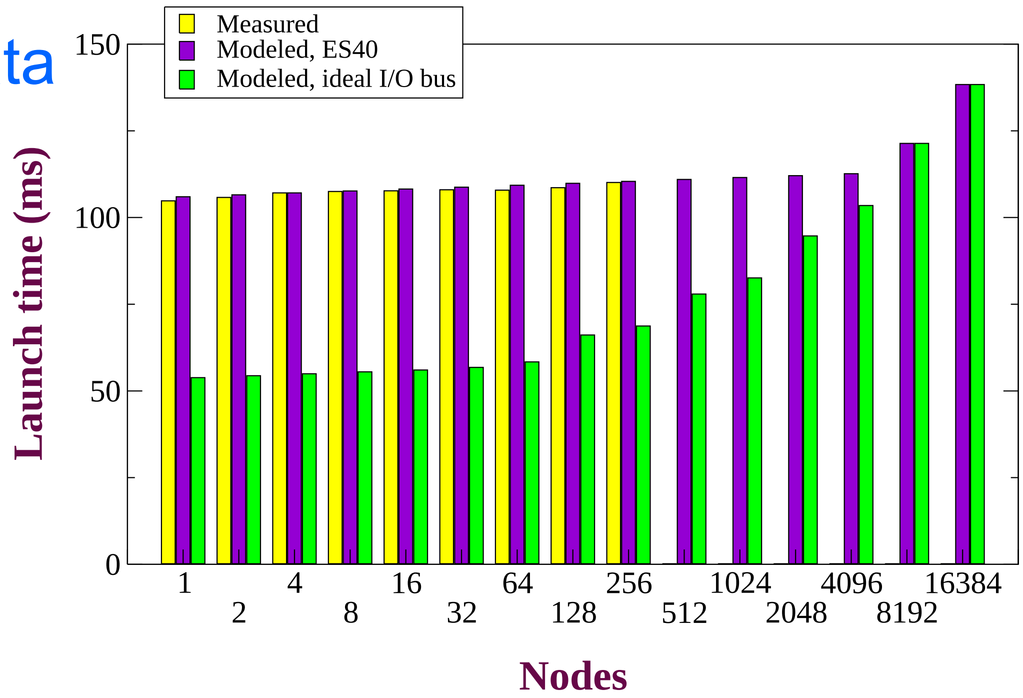
- Uses primitives for data dissemination and coordination
- Interactive job launching speeds
- Context-switching at milliseconds level
- Details in [SC'02]

Resource Management

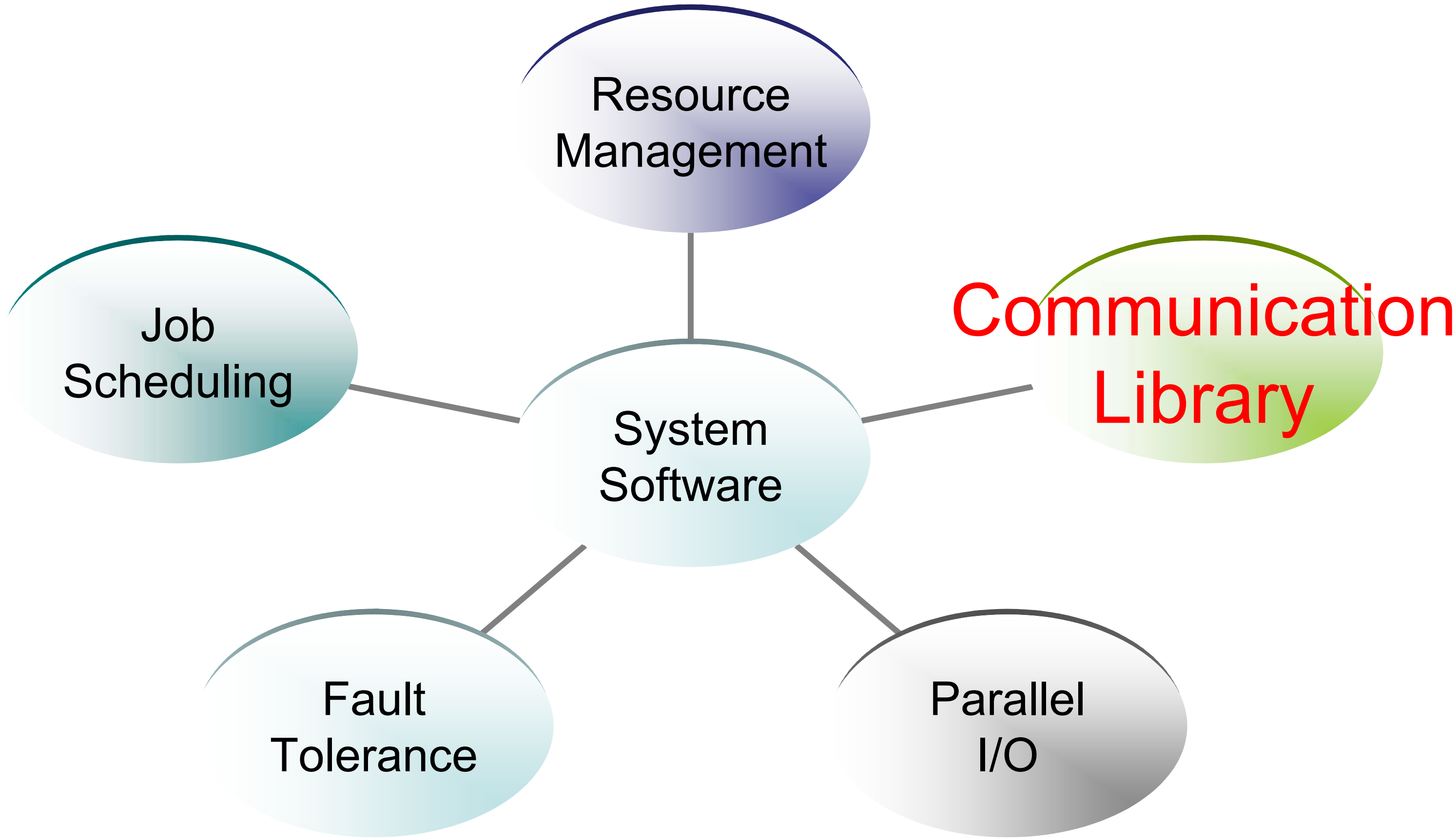
Scalable Tool for Resource Management

STORM

- Uses primitives for data dissemination and coordination
- Interactive job launching speeds
- Context-switching at milliseconds level
- Details in [SC'02]



System Software Components



Communication Library

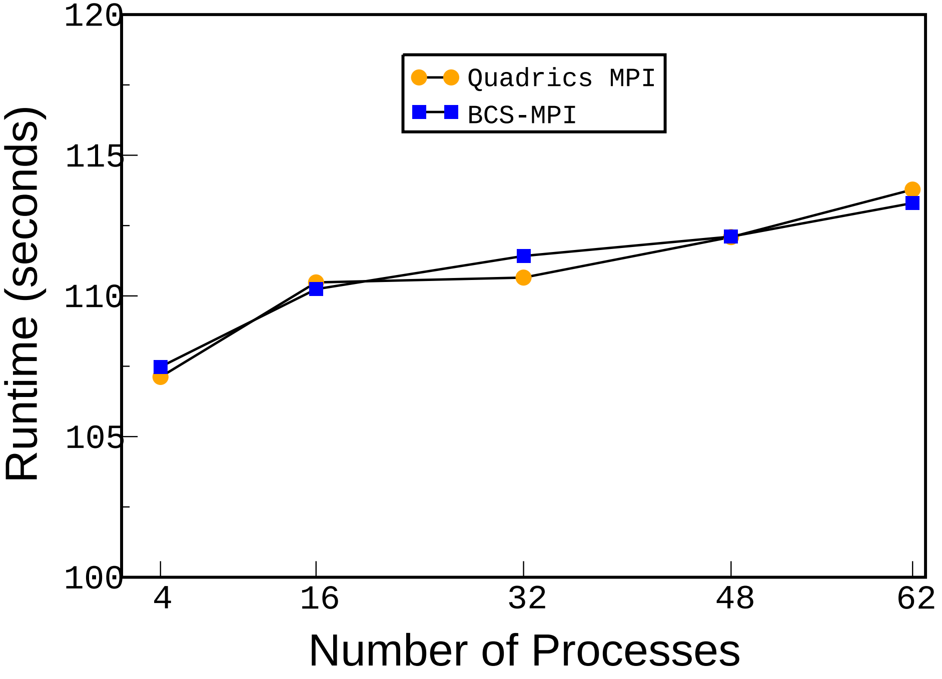
BCS-MPI [SC'03]

- An MPI subset
- Buffers communication in short time slices
- Simplified design on top of mechanisms

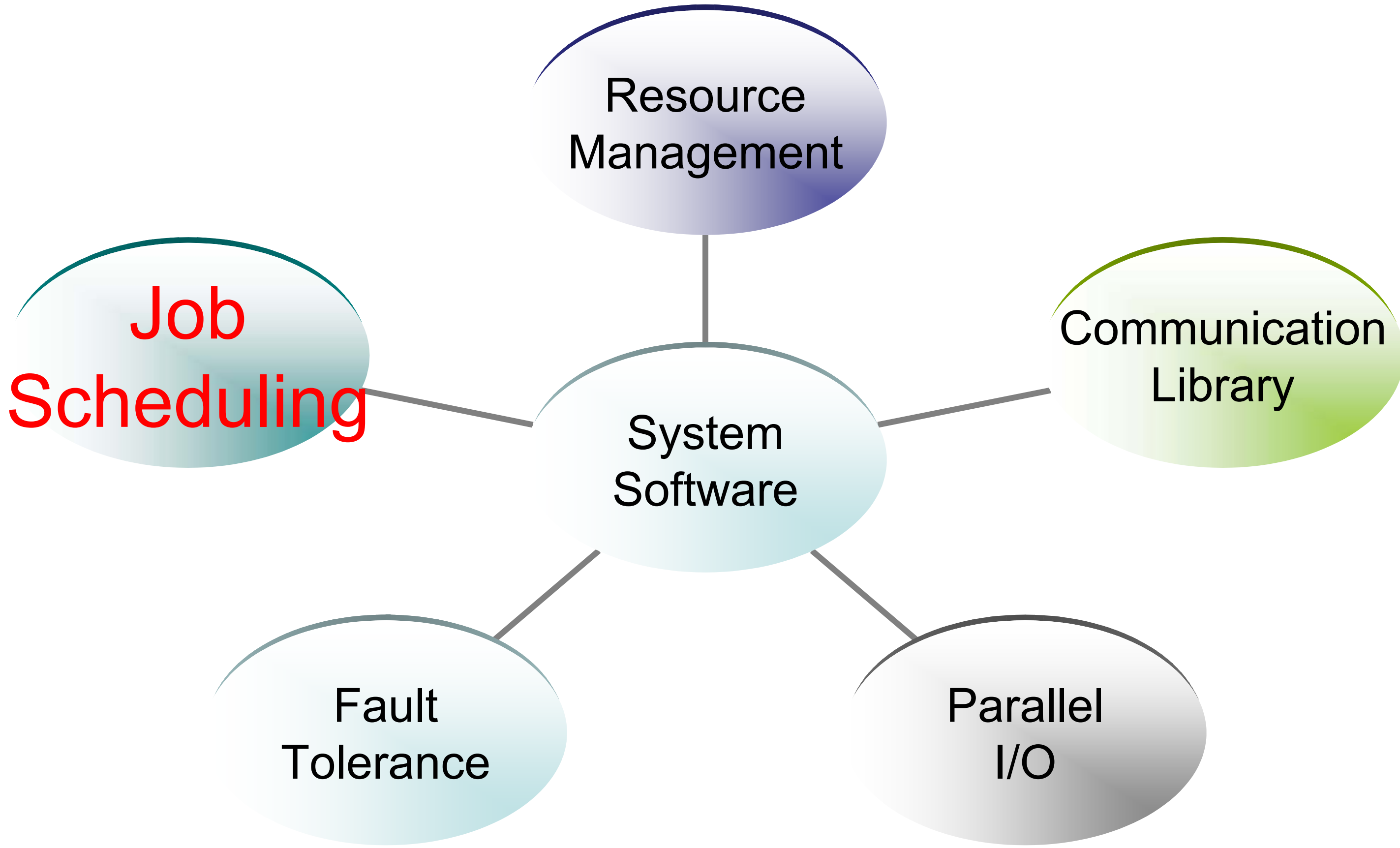
Communication Library

BCS-MPI [SC'03]

- An MPI subset
- Buffers communication in short timeslices
- Simplified design on top of mechanisms



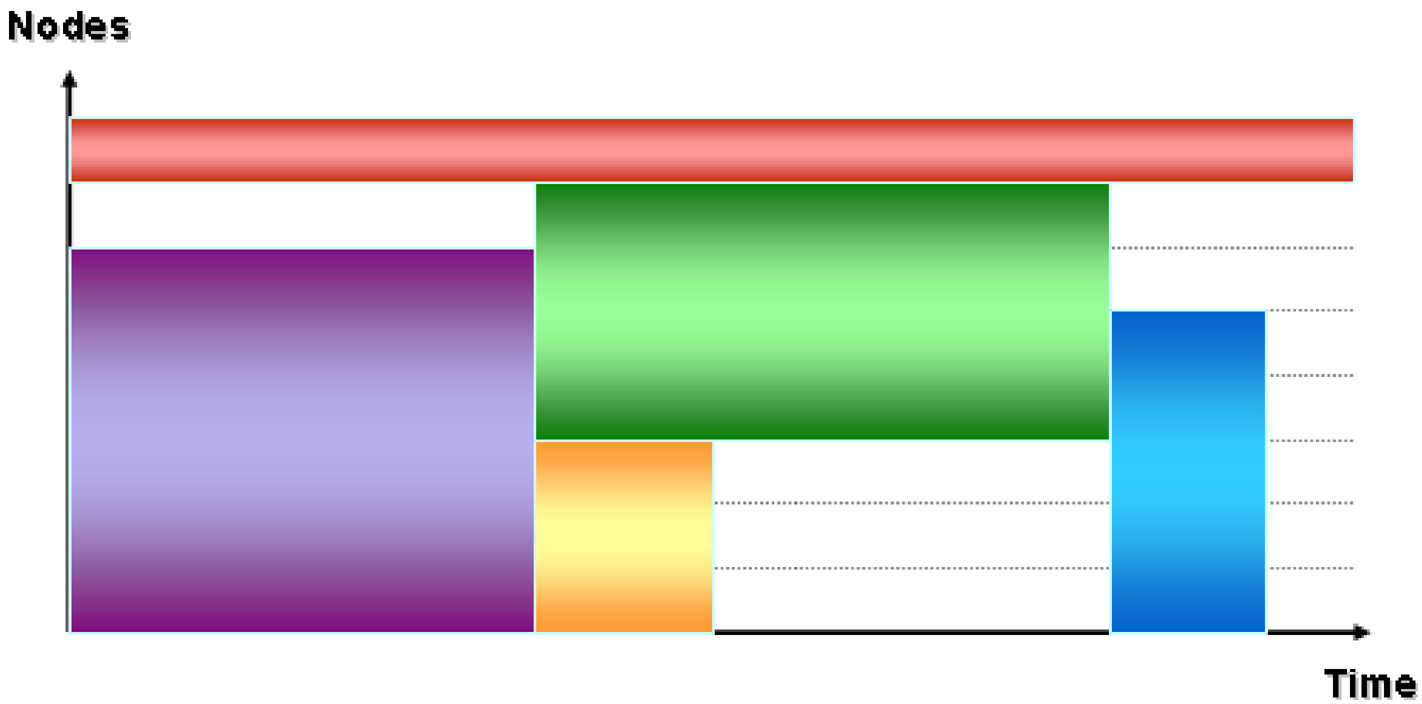
System Software Components



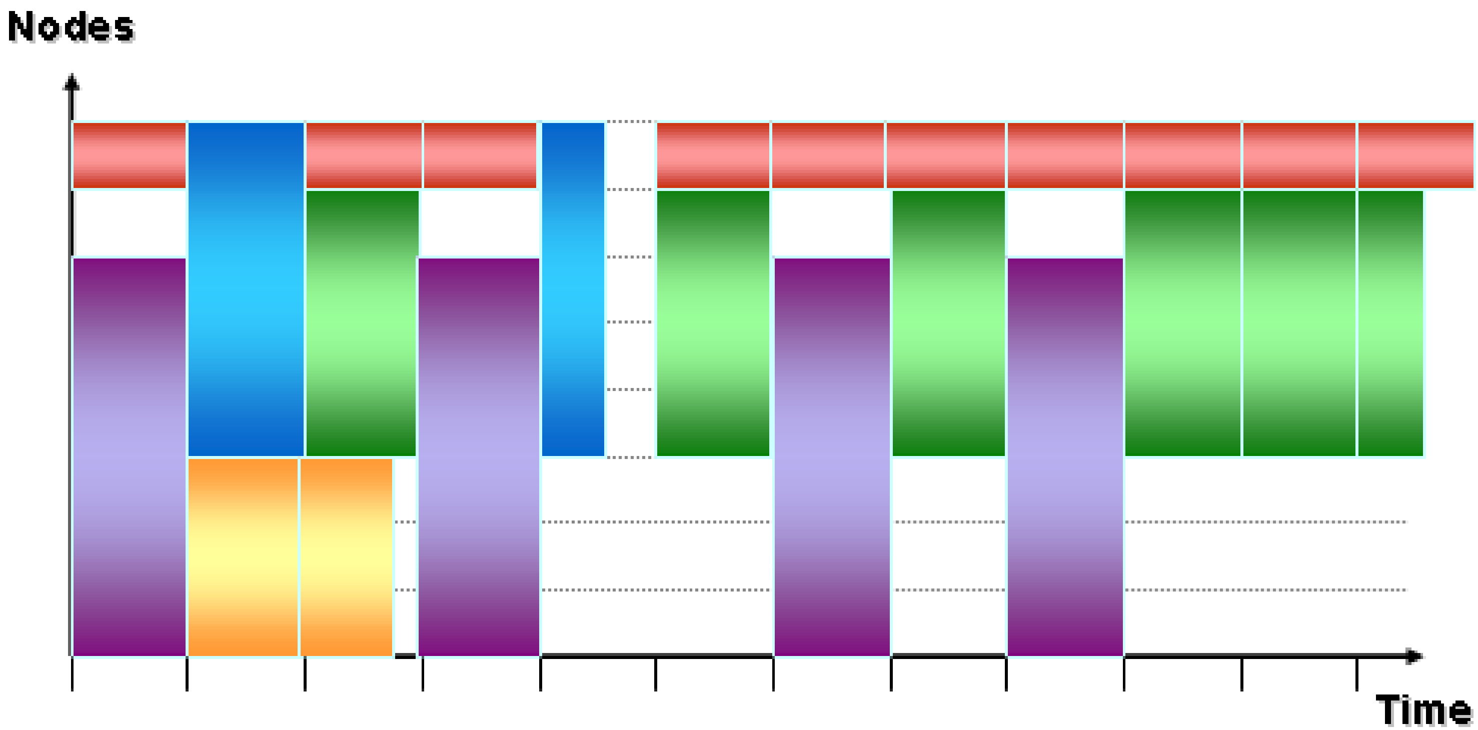
Job Scheduling

- Controls the allocation of space and time resources to jobs
- HPC apps have special requirements
  - Multiple processing and network resources
  - Synchronization (< 1ms granularity)
  - Potentially memory hogs with little locality
- Has significant effect on throughput, responsiveness, and utilization

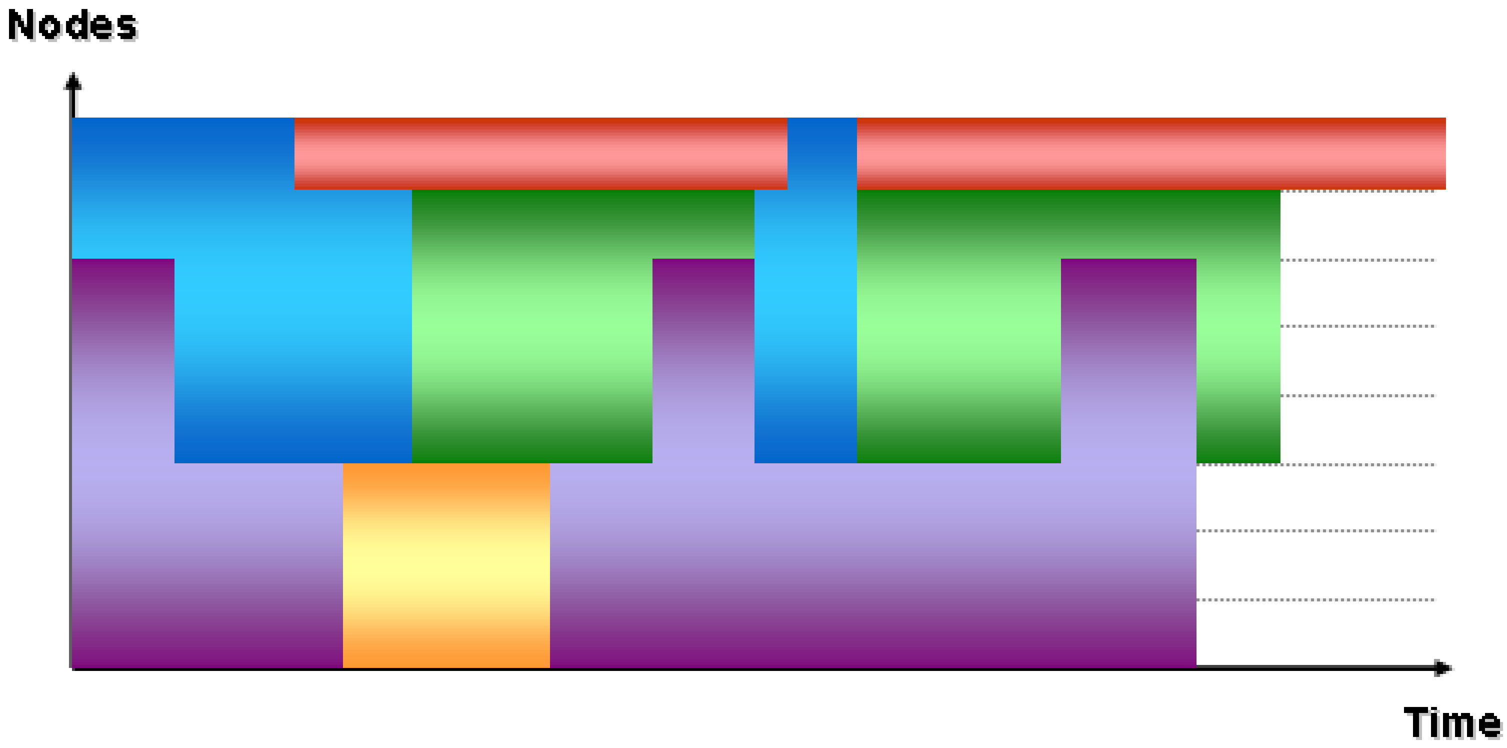
First-Come-First-Serve (FCFS)



Gang Scheduling (GS)



Implicit CoScheduling



Hybrid Methods

- Combines global synchronization with local information
- Relies on scalable primitives for global coordination and information exchange
- We Implemented two novel algorithms:
  - Flexible CoScheduling (FCS)
  - Buffered CoScheduling (BCS)

Flexible CoScheduling (FCS)

- Measure communication characteristics, such as granularity and wait times
- Classify processes based on synchronization requirements
- Schedule processes based on class
- Details in [IPDPS'03]

Buffered CoScheduling (BCS)

- Buffer all communications
- Exchange information about pending communication every time slice
- Schedule and execute communication
- Implemented mostly on the NIC
- Requires fine-grained heartbeats
- Details in [SC'03]

Methodology

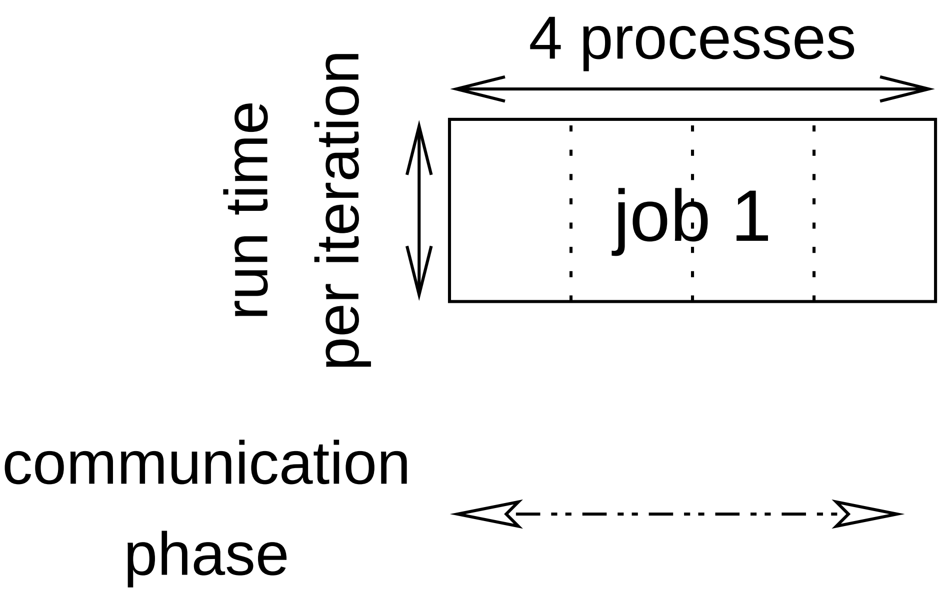
- Synthetic, controllable MPI programs
- Workload
  - Static: all jobs start together
  - Dynamic: different sizes, arrival and run times
- Various schedulers implemented:
  - FCFS, GS, FCS, SB (ICS), BCS
- Emulation vs. simulation
  - Actual implementation takes into account all the overhead and factors of a real system

Hardware Environment

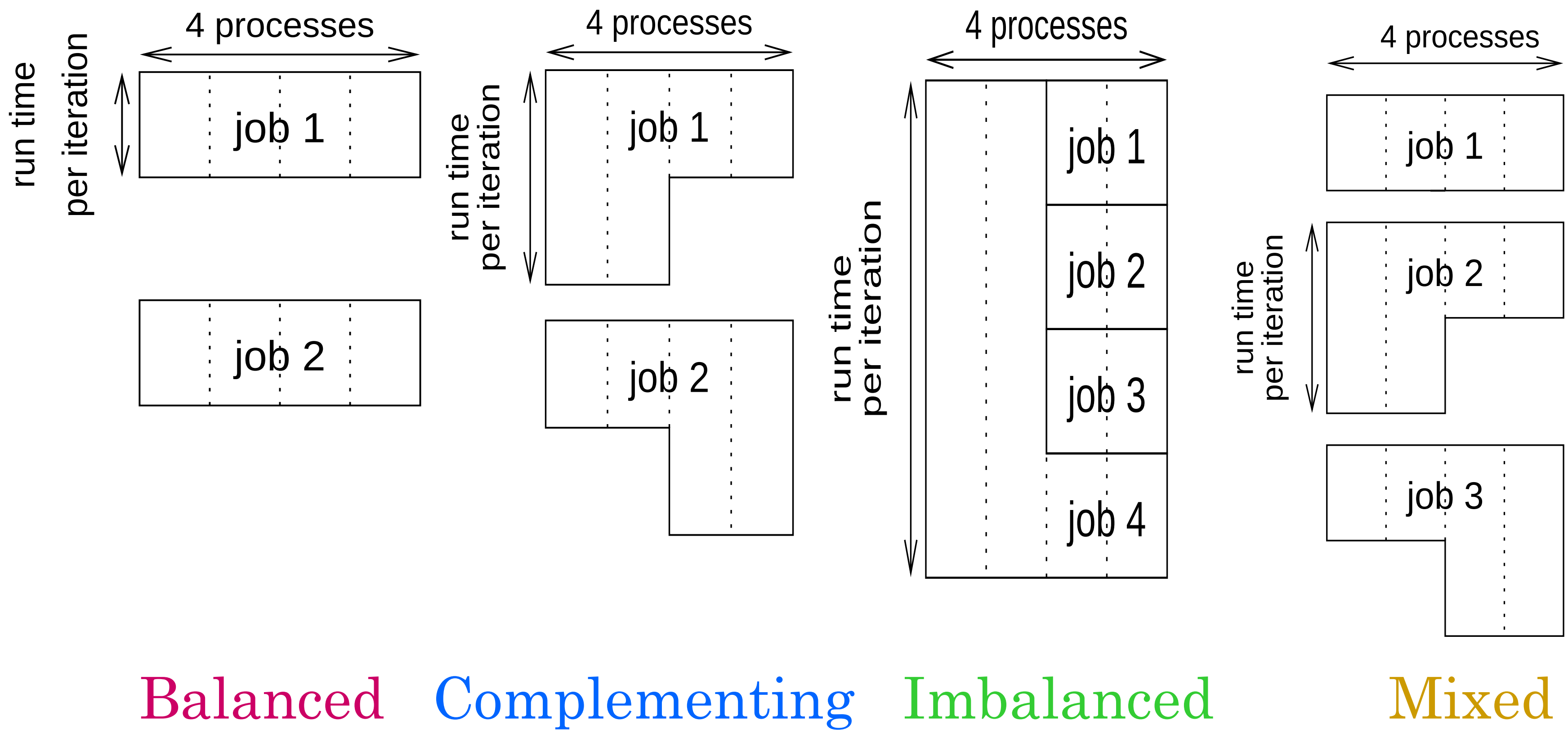
- Environment ported to three architectures and clusters:
  - Crescendo: 32x2 Pentium III, 1GB
  - Accelerando: 32x2 Itanium II, 2GB
  - Wolverine: 64x4 Alpha ES40, 8GB
- Quadrics' QsNet interconnect
  - 400 MB/s nominal bandwidth
  - <10ms global synchronization

Synthetic Application

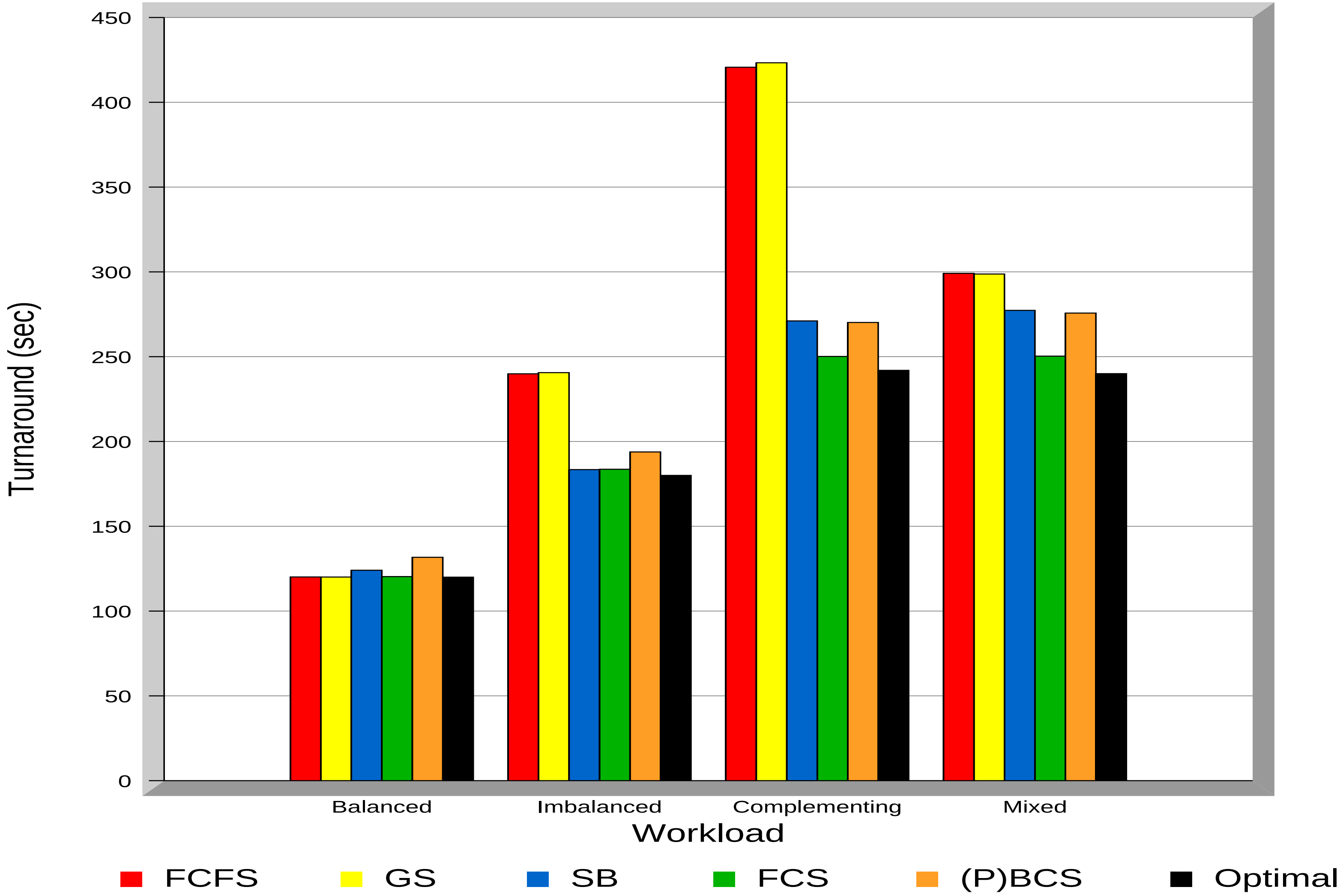
- Bulk synchronous, 3ms basic granularity
- Can control: granularity, variability and Communication pattern



Synthetic Scenarios



Turnaround Time



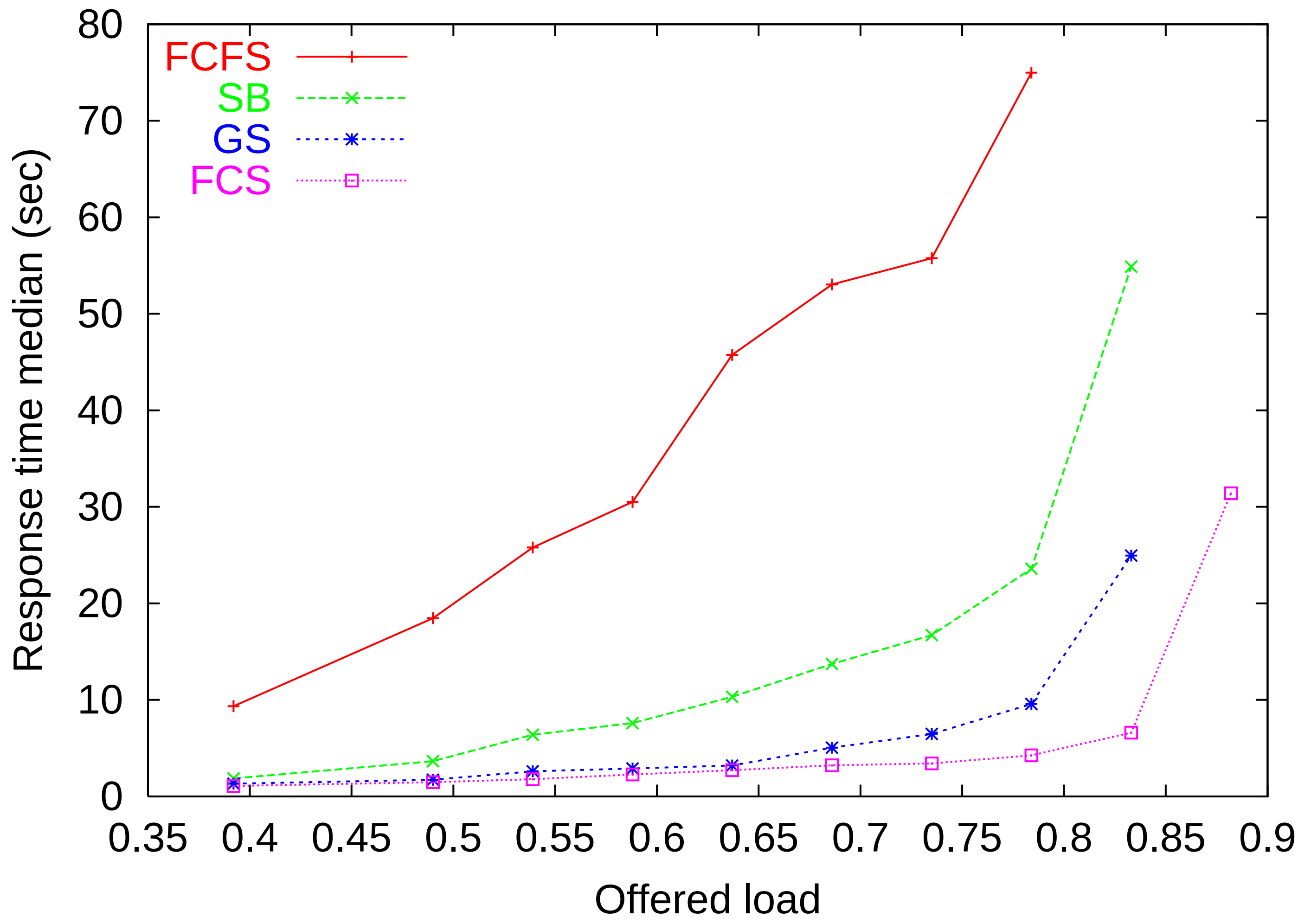
Dynamic Workloads [JSSPP'03]

- Static workloads are simple and offer insights, but are not realistic
- Most real-life workloads are more complex
- Users submit jobs dynamically, of varying time and space requirements

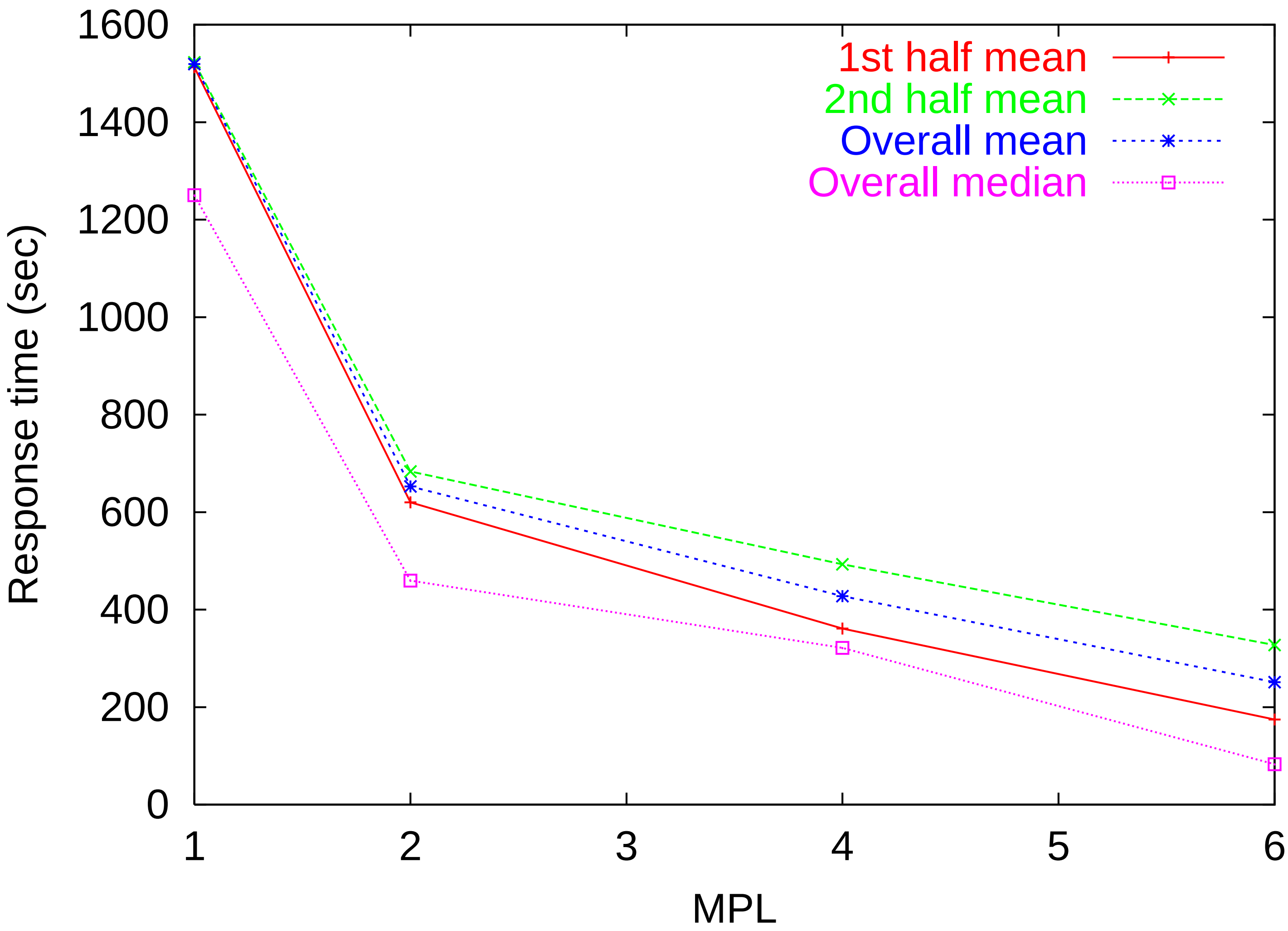
Dynamic Workload Methodology

- Emulation using a workload model [Lublin03]
- 1000 jobs, approx. 12 days, shrunk to 2 hrs
- Varying load by factoring arrival times
- Using same synthetic application, with random:
  - Arrival time, run time, and size, based on model
  - Granularity (fine, medium, coarse)
  - communication pattern (ring, barrier, none)

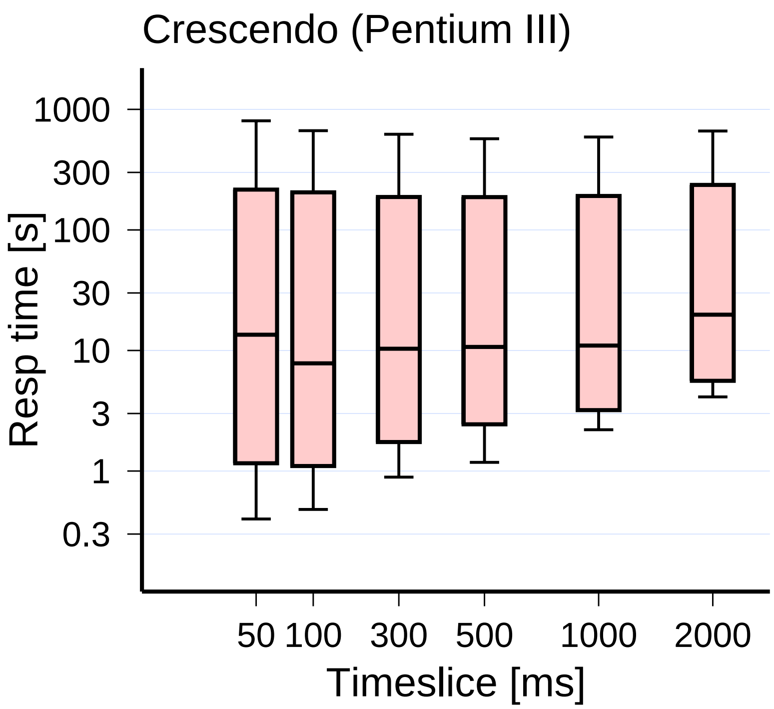
Load – Response Time

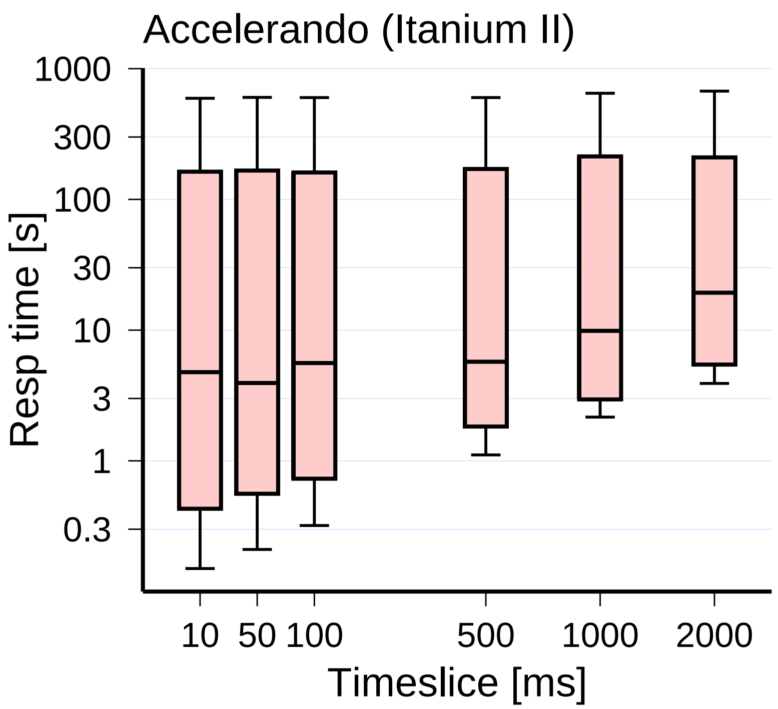


FCFS vs. GS and MPL

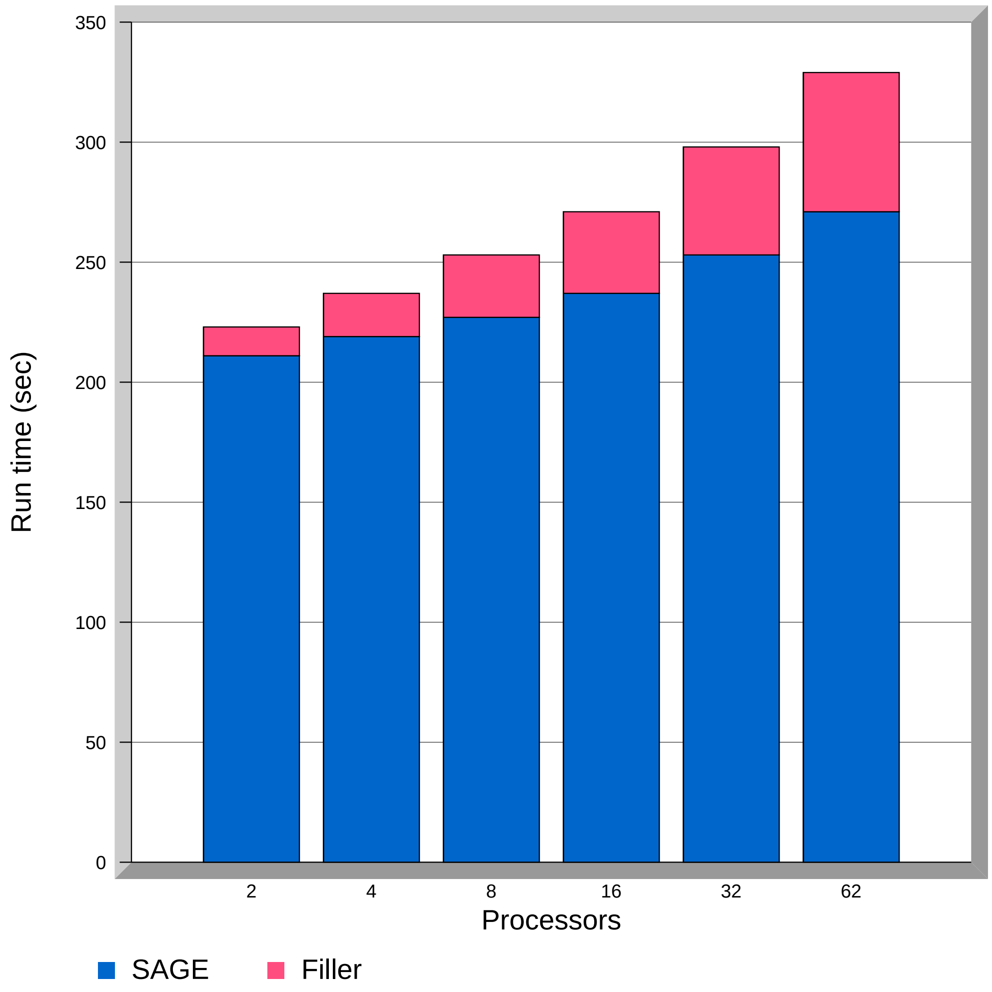


Timeslice – Response Time





Resource Overlapping



Conclusions

- As clusters grow, interconnection technology advances:
  - Better bandwidth and latency
  - On-board programmable processor, RAM
  - Hardware support for collective operations

Allows the development of common system infrastructure that is a parallel program in itself

Conclusions (cont.)

- On top of infrastructure we built:
  - Scalable resource management (STORM)
  - Simplified system design and communication library
  - Novel job scheduling algorithms

Conclusions (cont.)

- Experimental performance evaluation demonstrates:
  - Scalable interactive job launching and context-switching
  - Multiprogramming parallel jobs is feasible
  - Adaptive scheduling algorithms adjust to different job requirements, improving response times and slowdown in various workloads

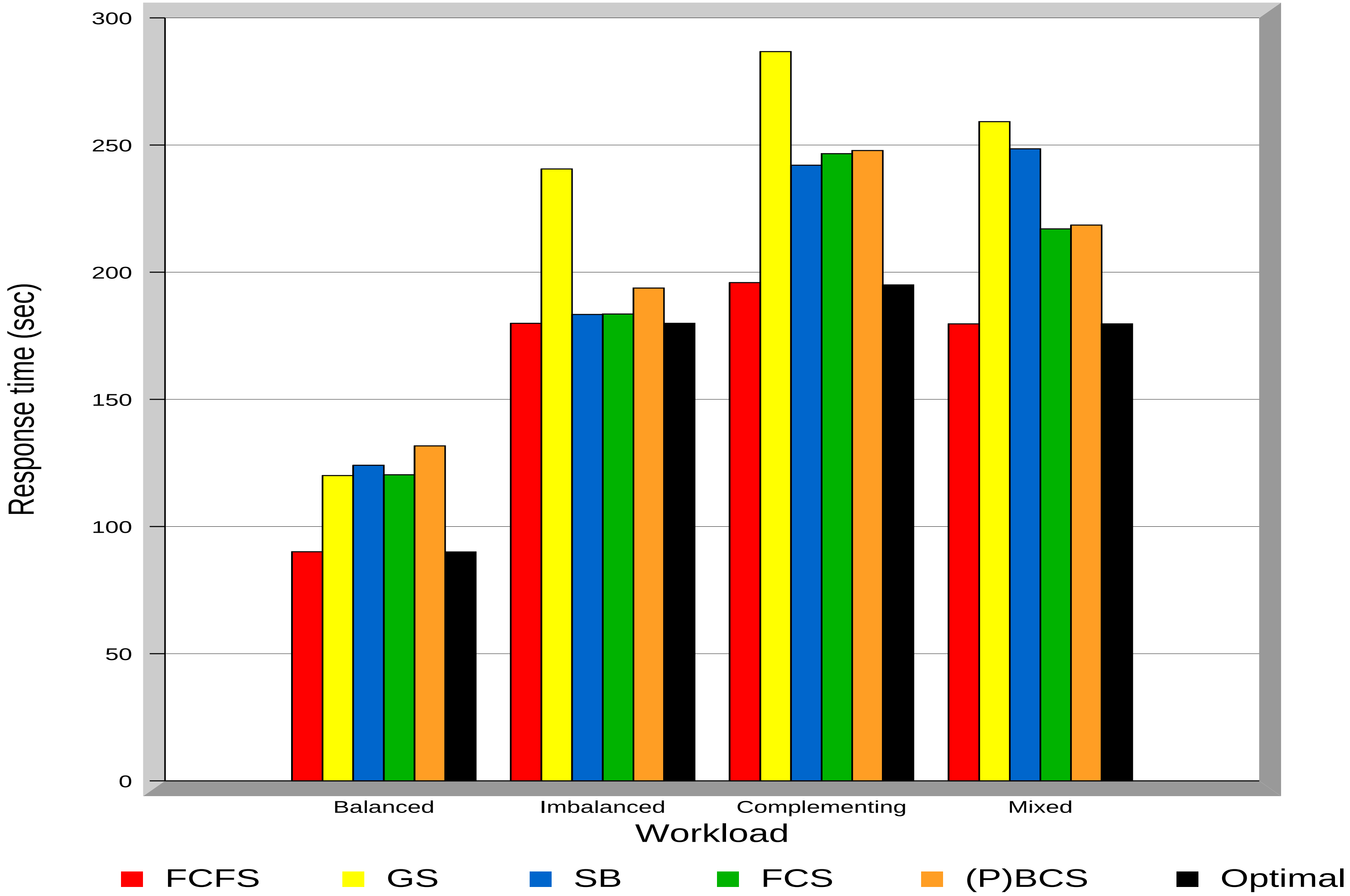
Future Directions

- Transparent fault-tolerance [IPDPS'04]
- Parallel I/O
- Kernel-level operation

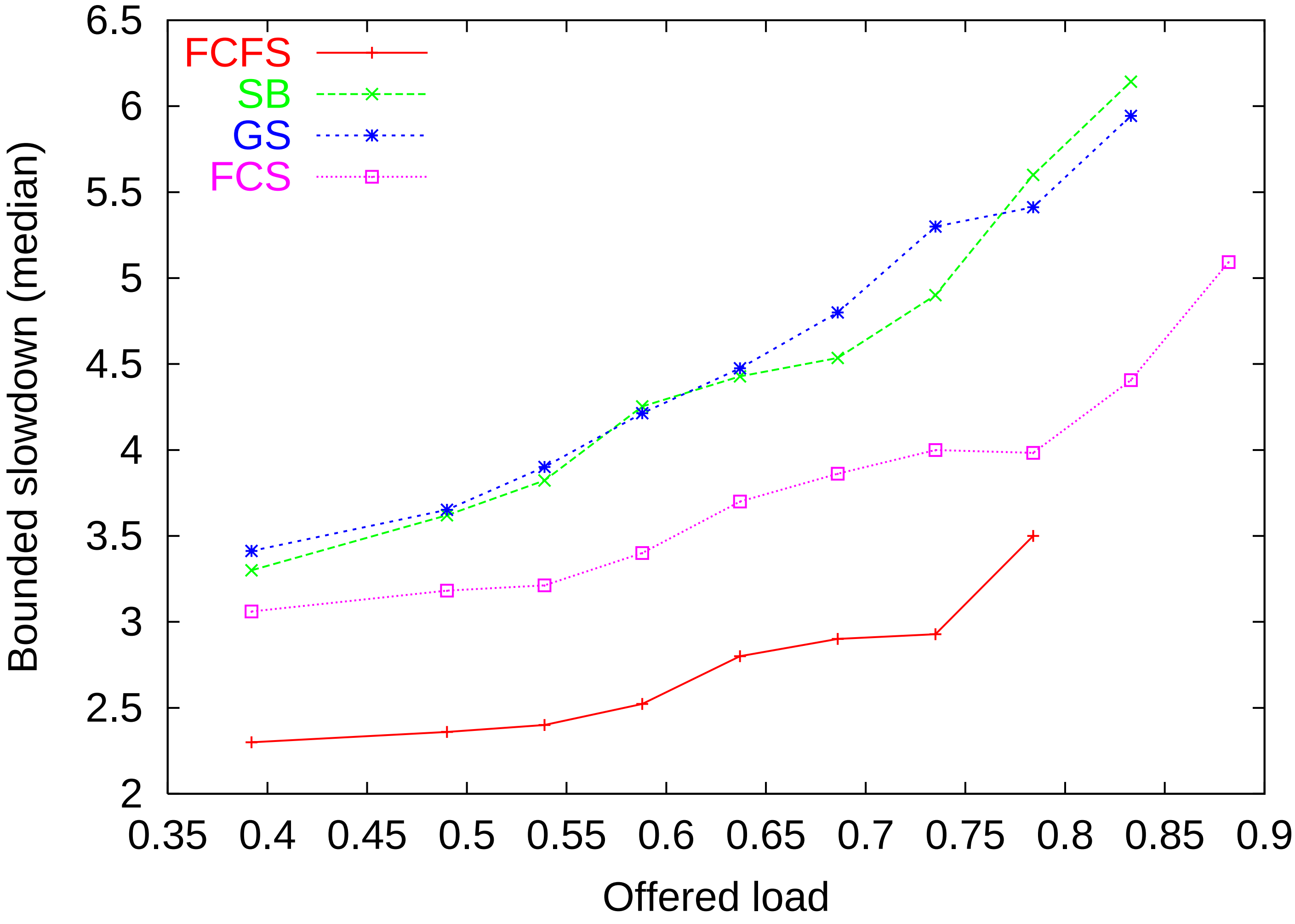
For more information:

http://www.cs.huji.ac.il/~etcs/research.html

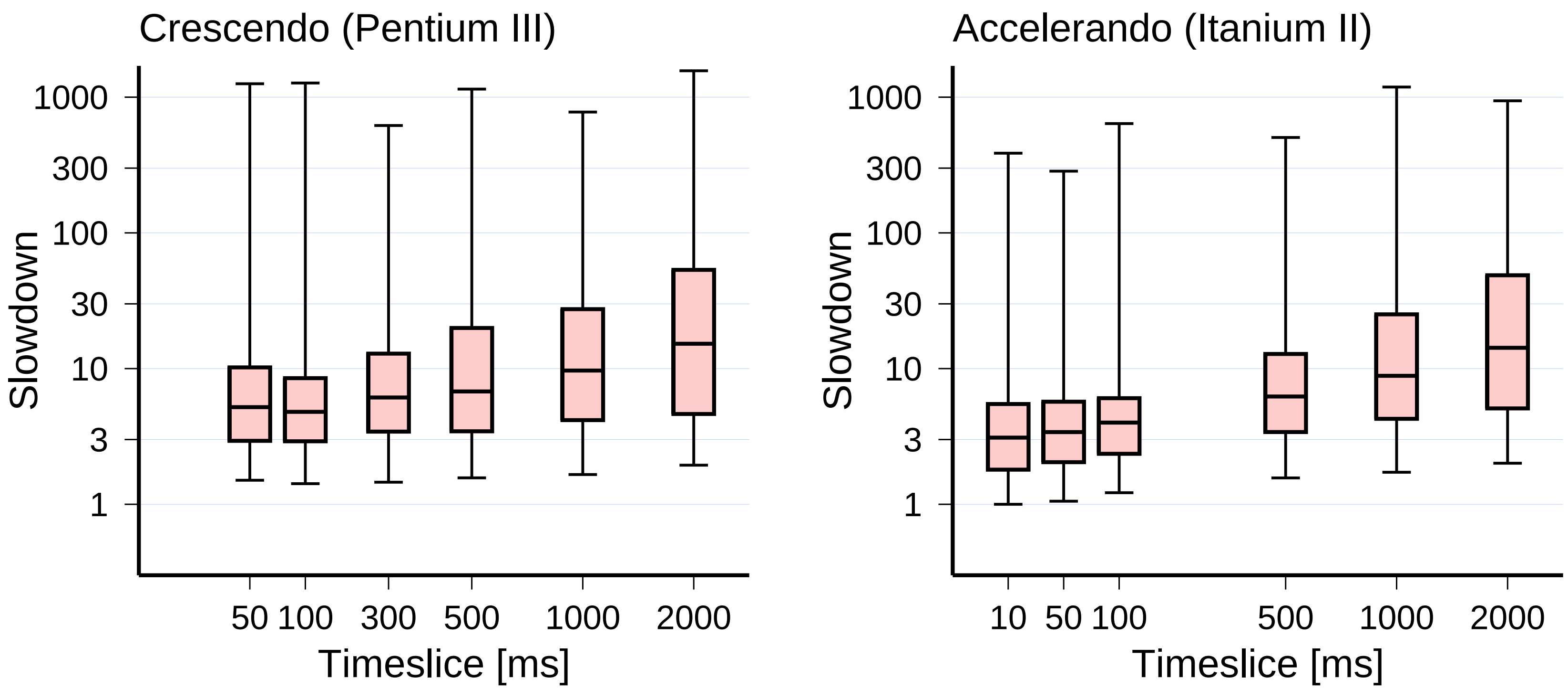
Response Time



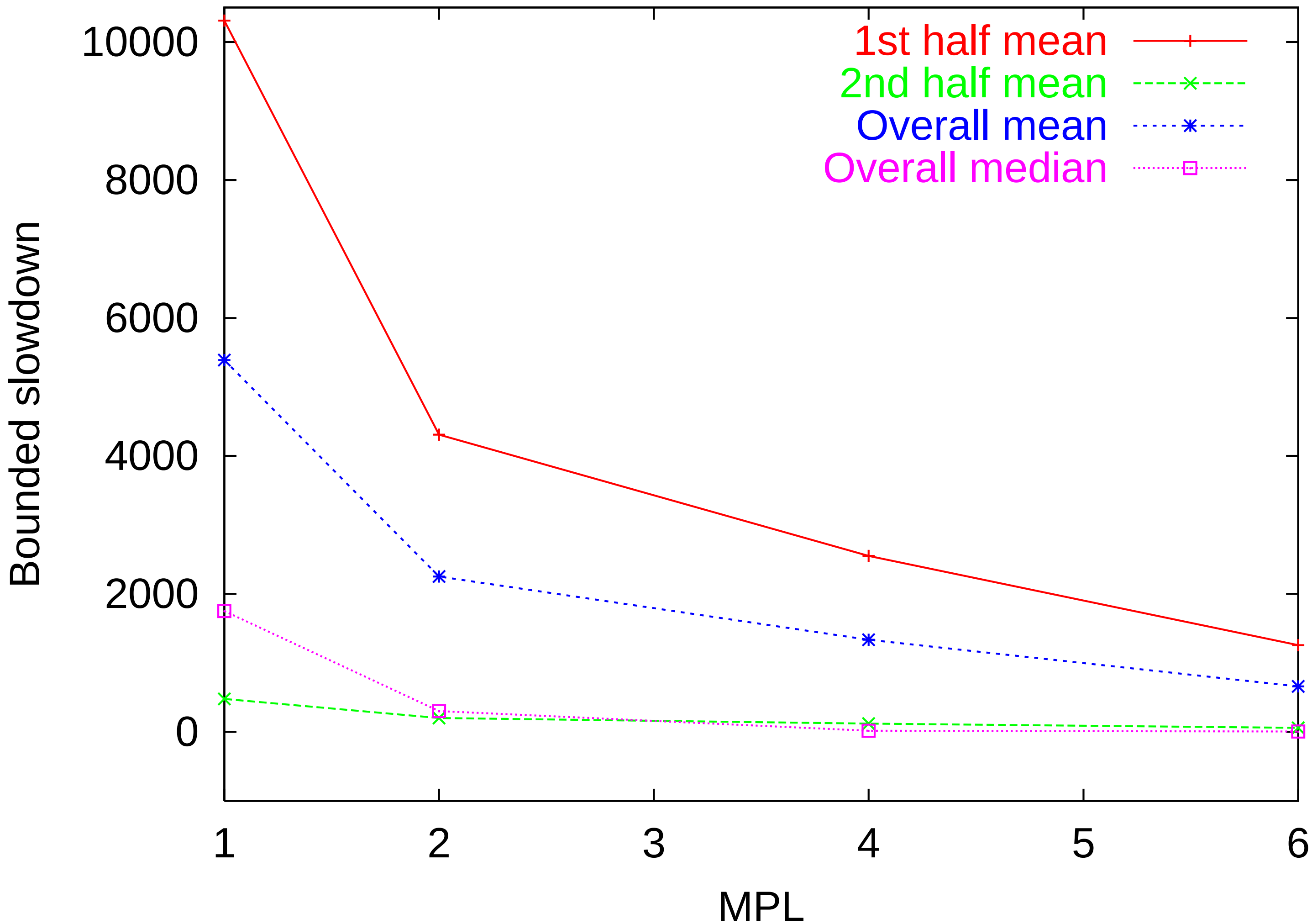
Load – Bounded Slowdown



Timeslice – Bounded Slowdown



FCFS vs. GS and MPL (2)

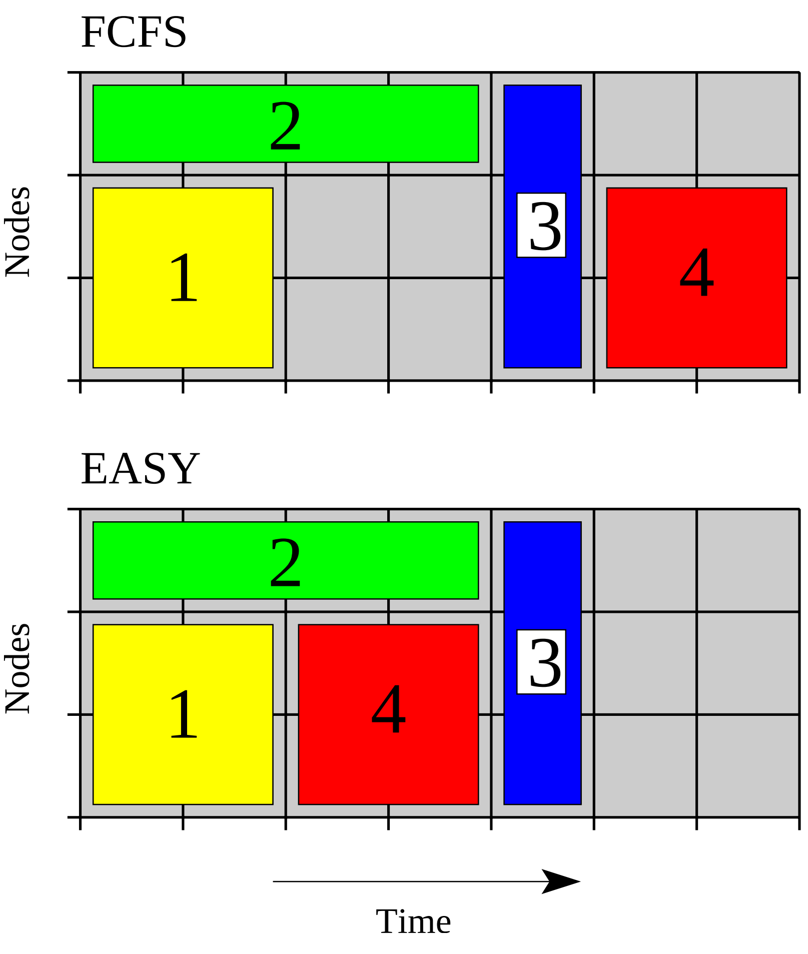


Backfilling

- Backfilling is a technique to move jobs forward in queue
- Can be combined with time-sharing schedulers such as GS when all timeslots are full

Backfilling

- Backfilling is a technique to move jobs forward in queue
- Can be combined with time-sharing schedulers such as GS when all timeslots are full



Effect of Backfilling