Leveraging Modern Interconnects for Parallel Job Scheduling

Eitan Frachtenberg, Hebrew University
With
Dror Feitelson, Hebrew University
Fabrizio Petrini and Juan Fernandez, LANL

Background

- Clusters and grids are growing in prevalence and performance
- Node count rises
- Advanced, high-end interconnects
- Large variety of application characteristics
- Underdeveloped cluster operating systems

Problems with System Software

Independent single-node OS (e.g. Linux) connected by distributed dæmons:

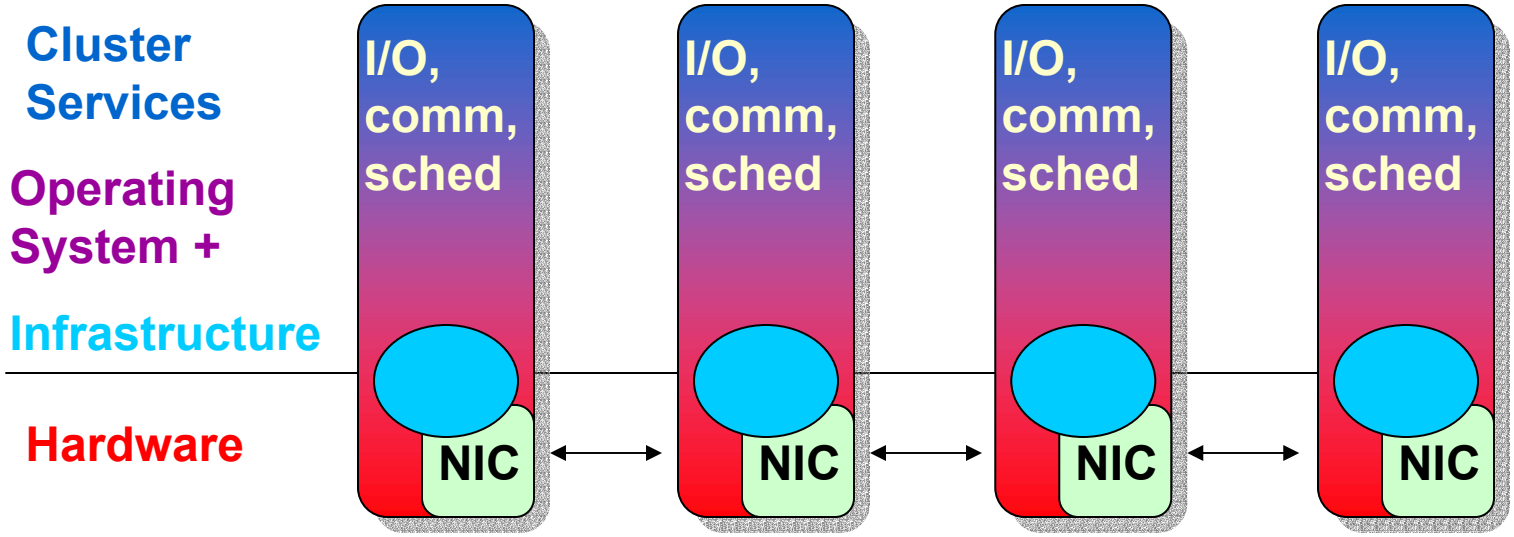
- Redundant components
- Performance hits
- Scalability issues
- Load balancing issues

The Vision

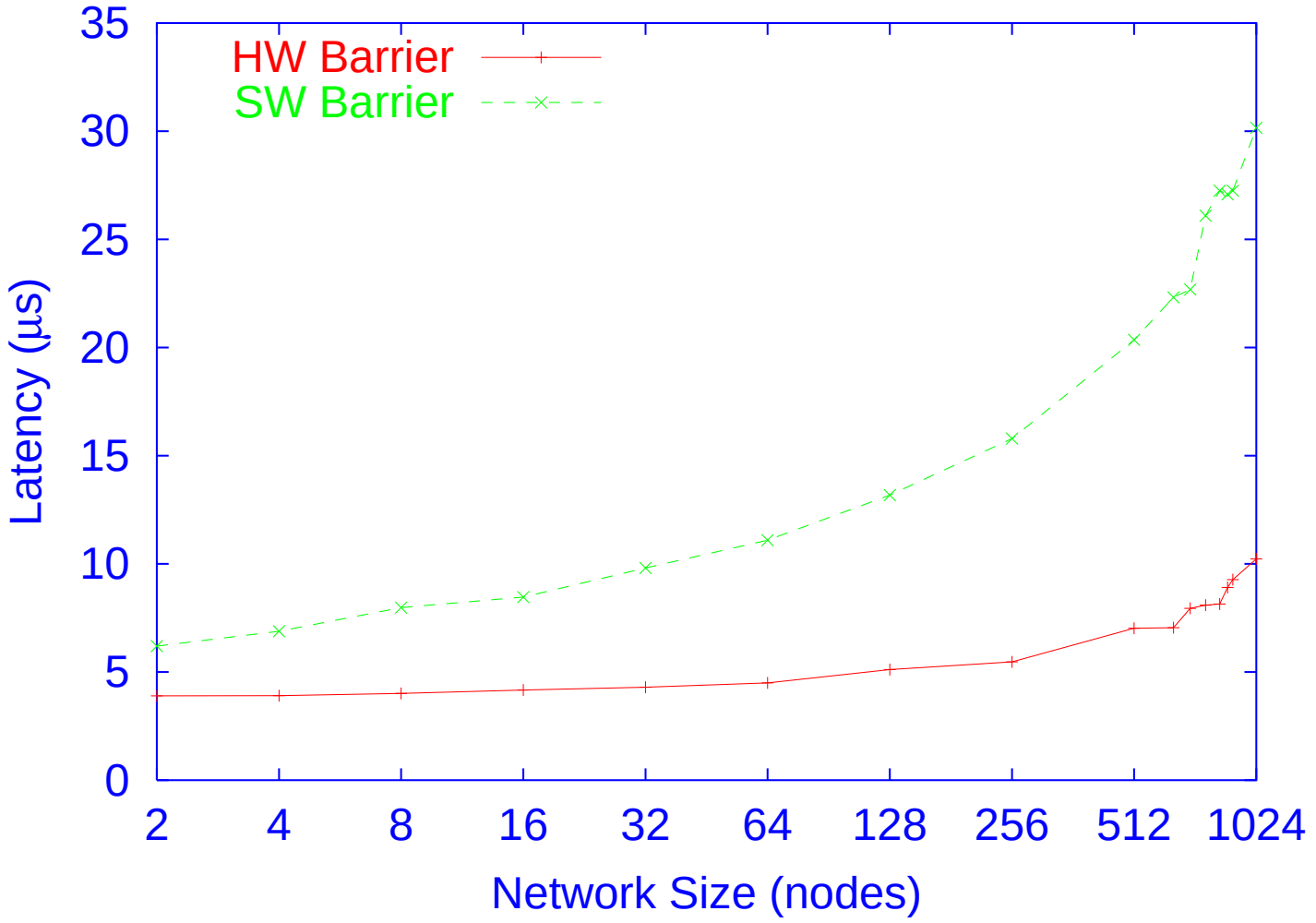
- Modern interconnects offer powerful collective operations, programmable NICs and on-board RAM
- Use a small set of network mechanisms to create a common infrastructure, a parallel application in itself
- Build upon this infrastructure to create unified system software
- System software Inherits scalability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

The Vision

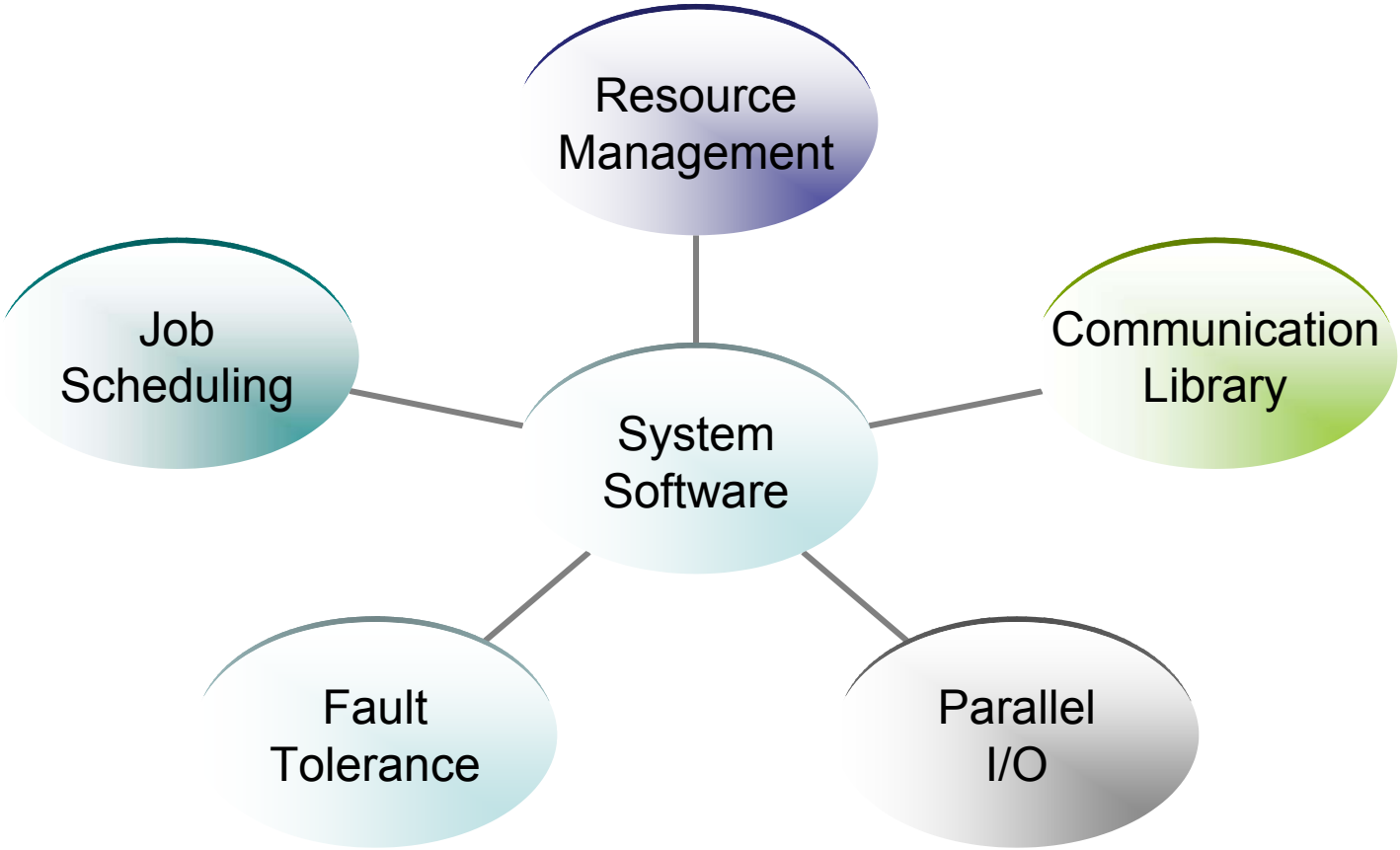
All system software based on a common infrastructure, using network primitives



Example: ASCI Q Barrier [HotI'03]



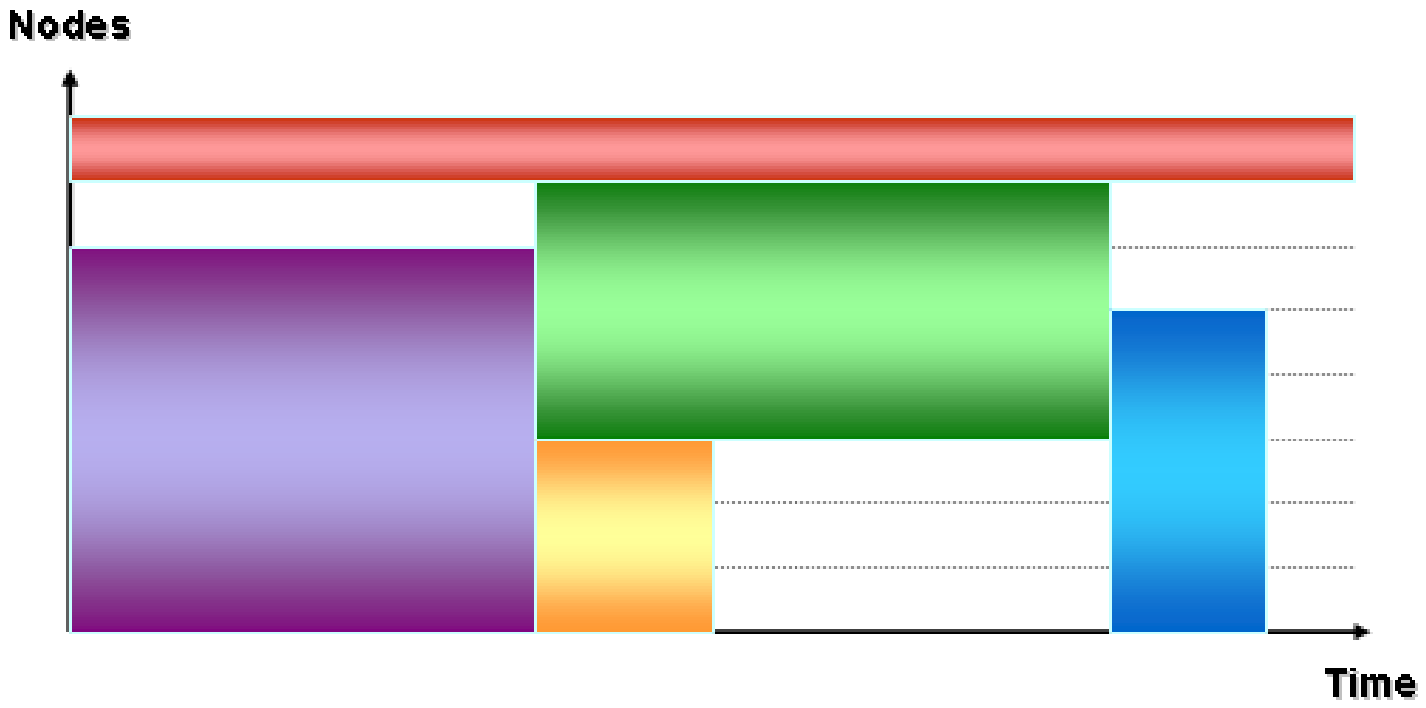
System Software Components



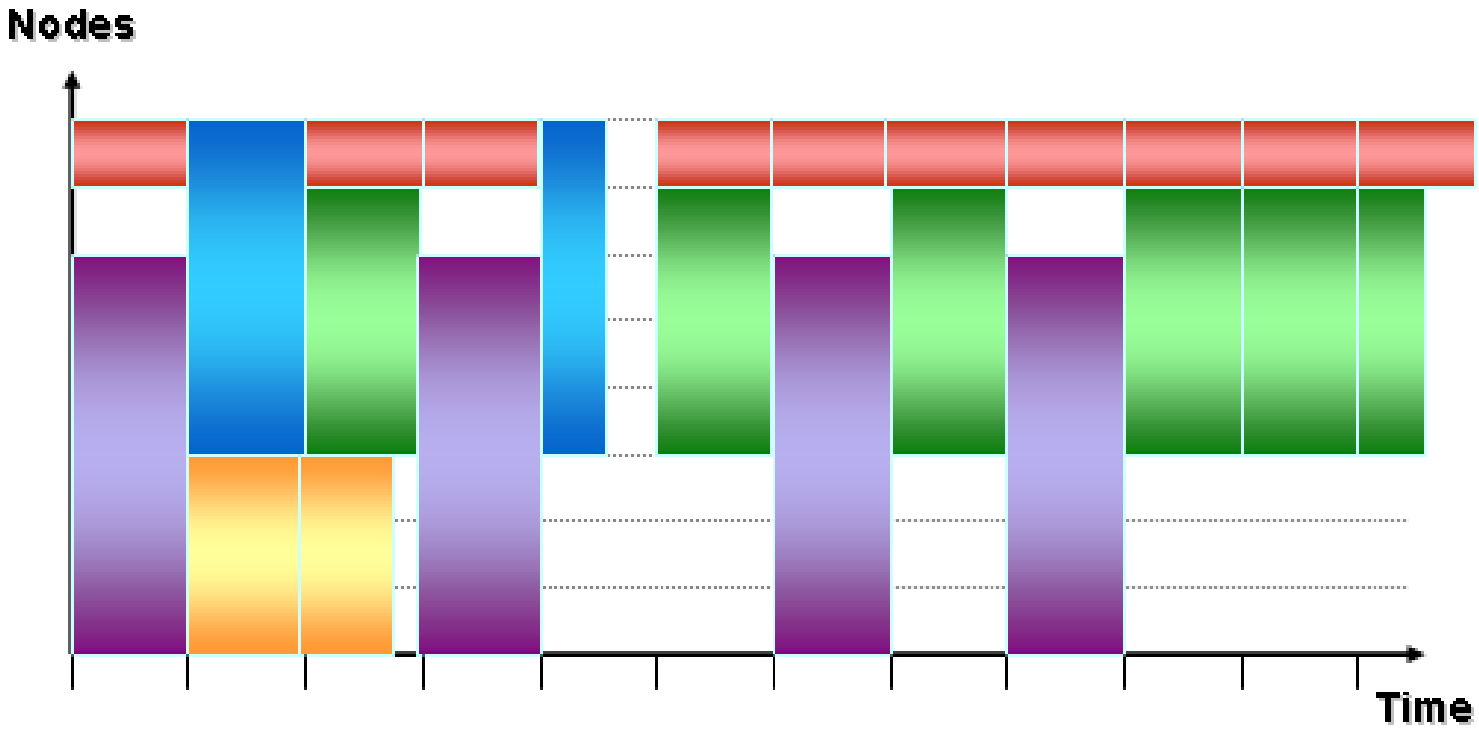
Job Scheduling

- Controls the allocation of space and time resources to jobs
- Has significant effect on throughput, responsiveness, and utilization
- Some apps have special needs
  - Synchronization (< 1ms granularity)
  - Load imbalance

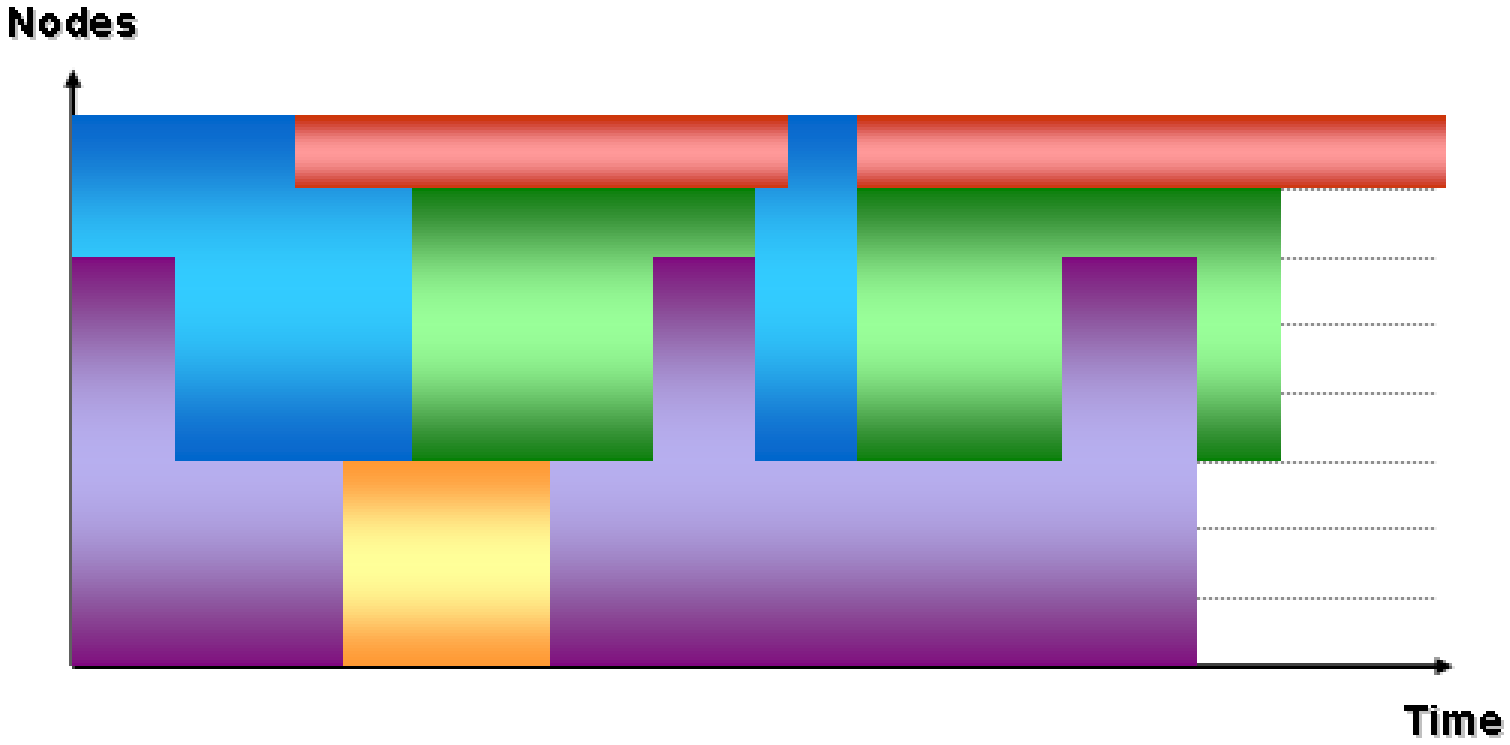
First-Come-First-Serve (FCFS)



Gang Scheduling (GS)



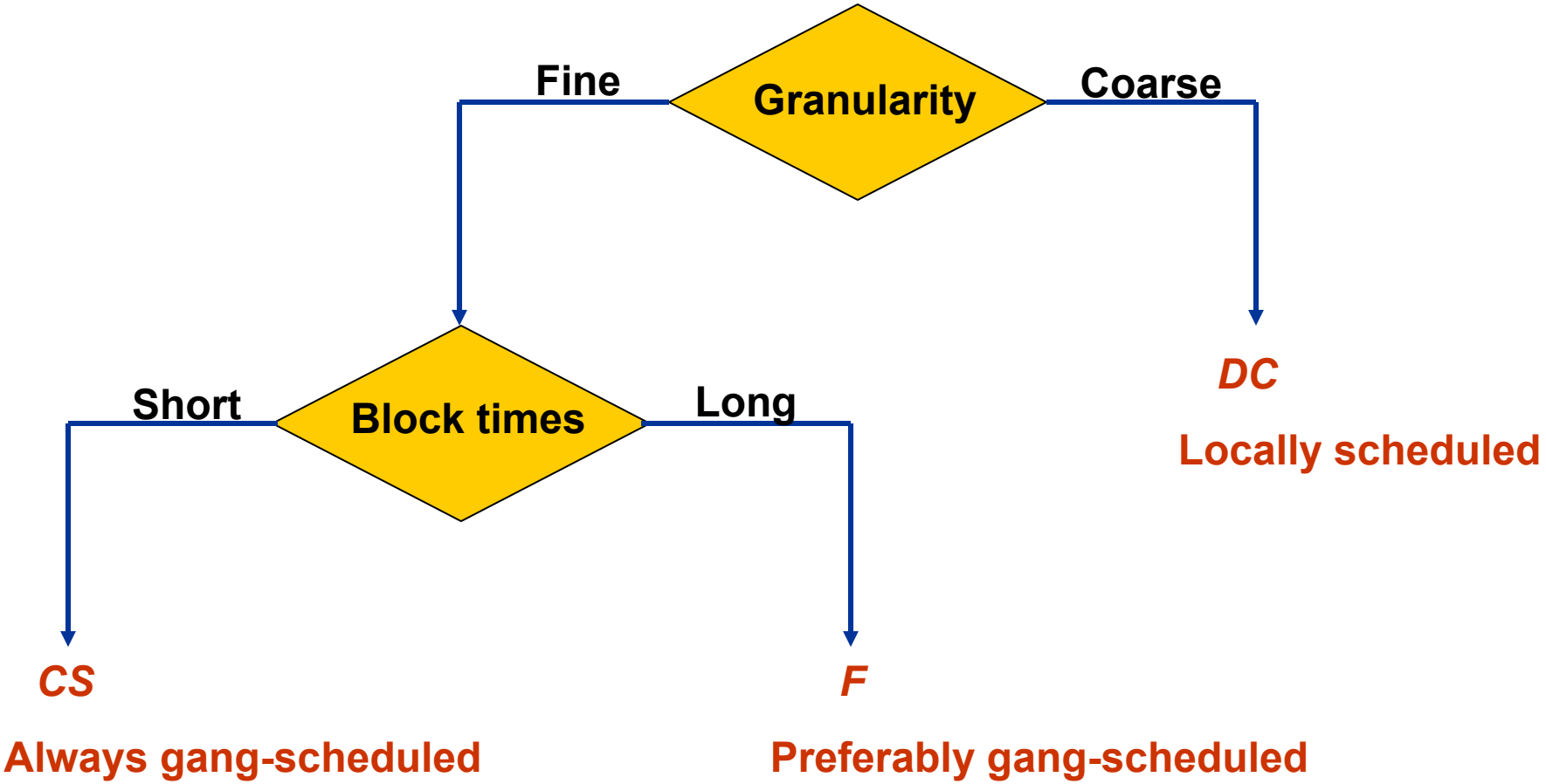
Implicit CoScheduling



Flexible CoScheduling (FCS)

- Hybrid: relies on both global coordination and local information
- Measure communication characteristics, such as granularity and wait times
- Classify processes based on synchronization requirements
- Schedule processes based on class
- Details in [IPDPS'03]

FCS Classification

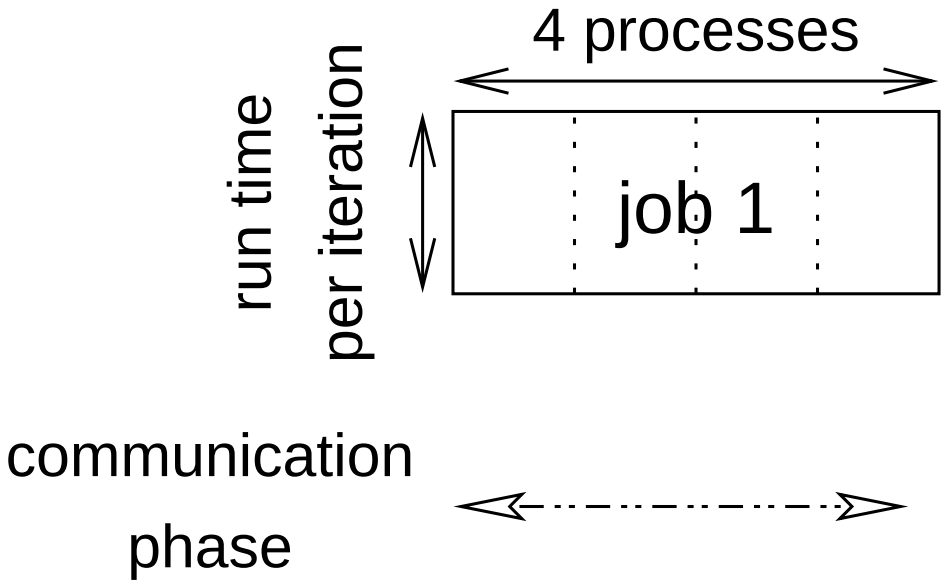


Methodology

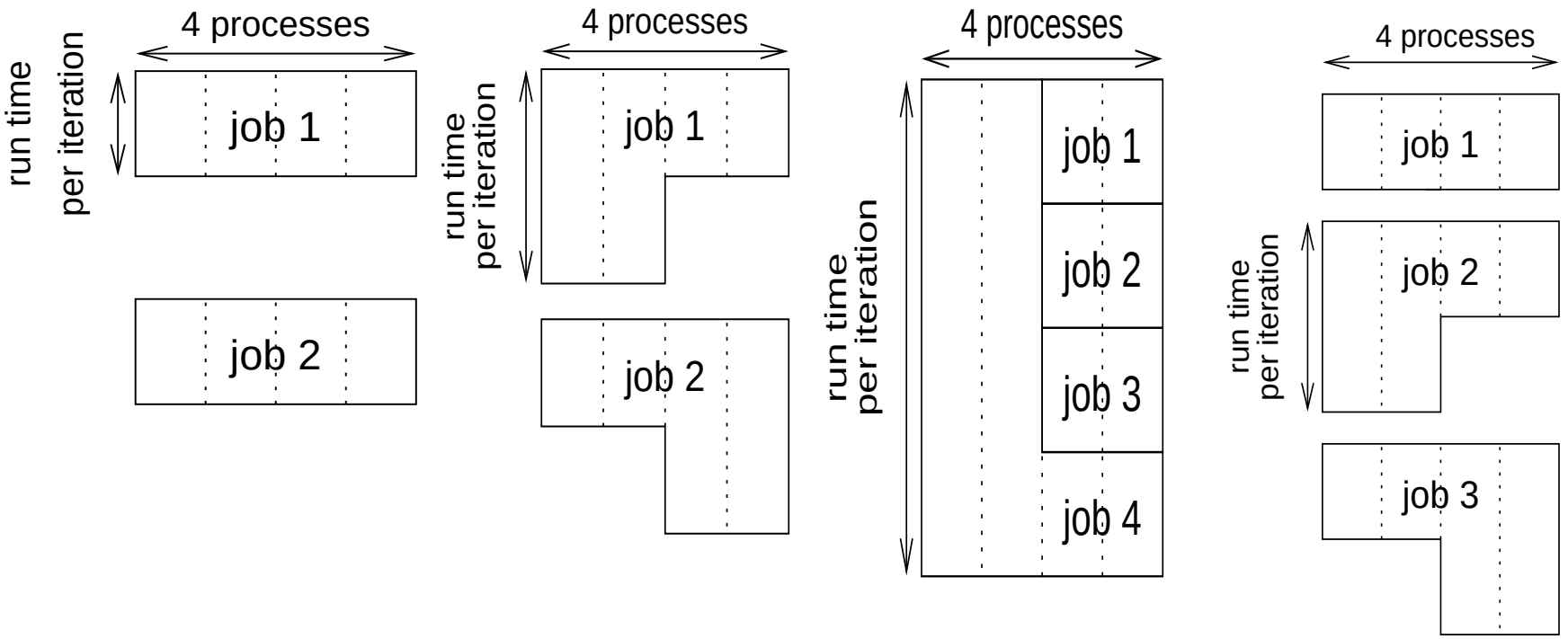
- Synthetic, controllable MPI programs
- Workload
  - Static: all jobs start together
  - Dynamic: different sizes, arrival and run times
- Various schedulers implemented:
  - FCFS, GS, FCS, SB (ICS)
- Emulation vs. simulation
  - Actual implementation on takes into account all the overhead and factors of a real system
  - Run on 32x2 Pentium-III and Itanium-II clusters

Synthetic Application

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

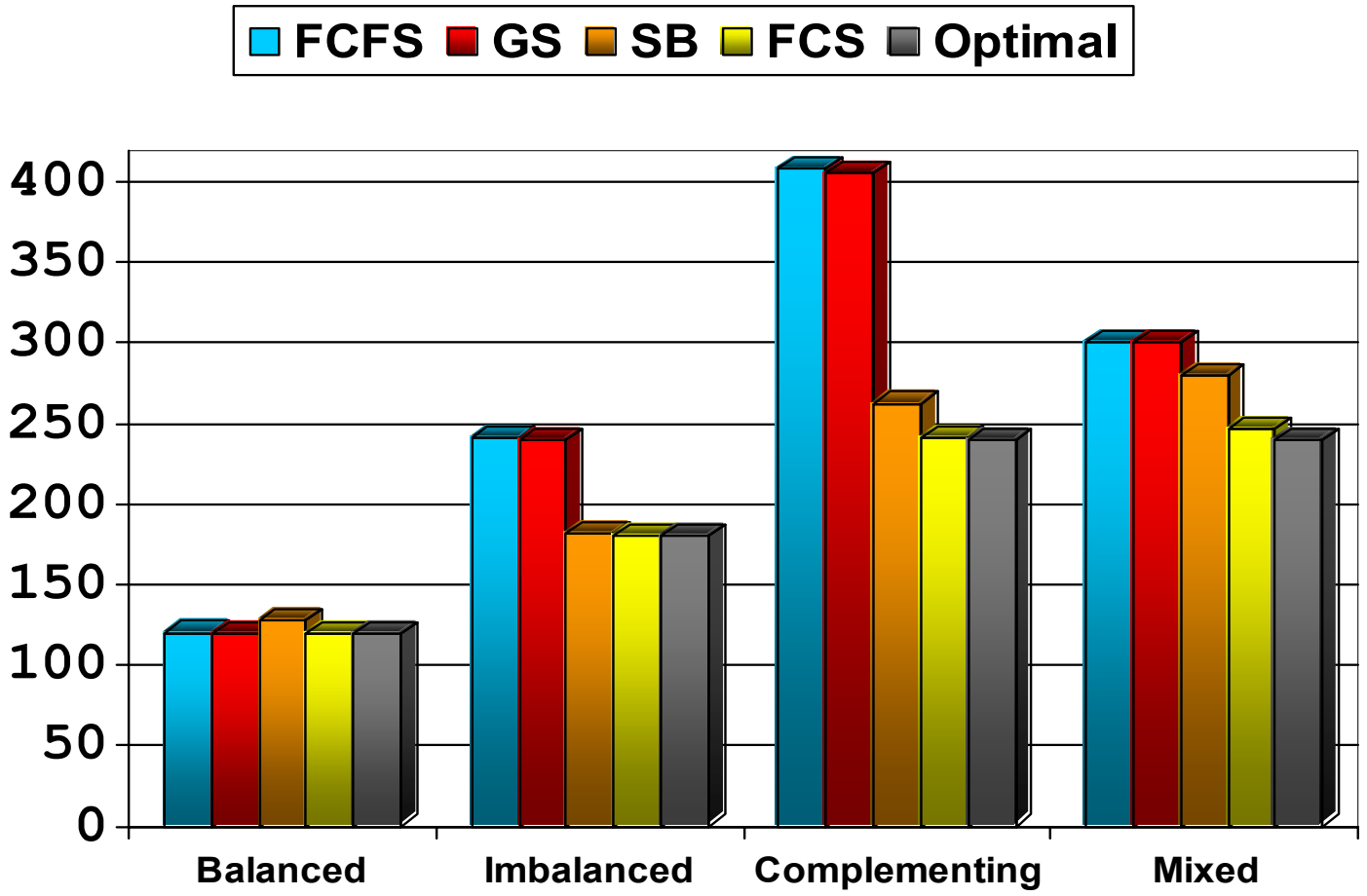


Synthetic Scenarios



Balanced Complementing Imbalanced Mixed

Turnaround Time



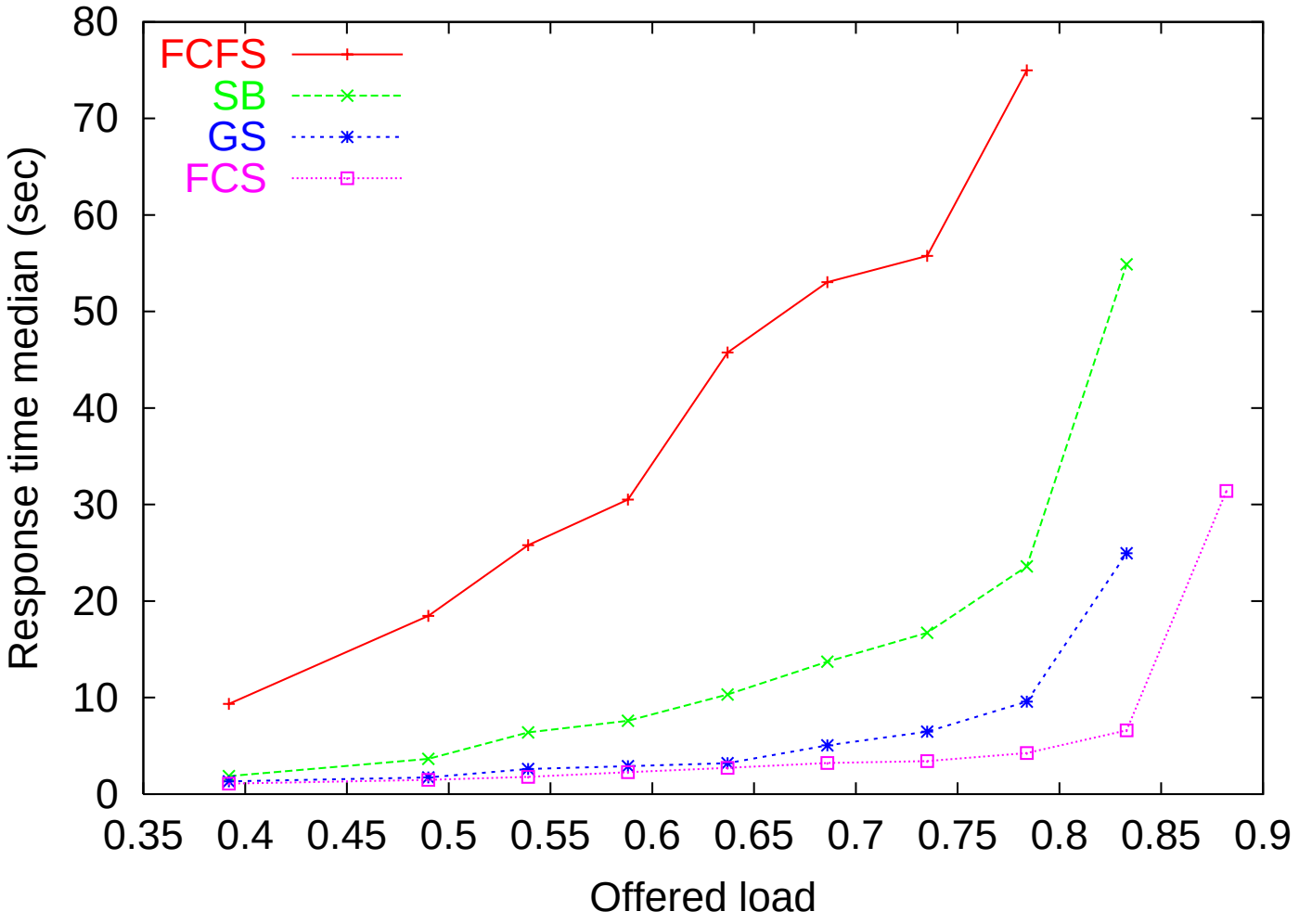
Dynamic Workloads [JSSPP'03]

- 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

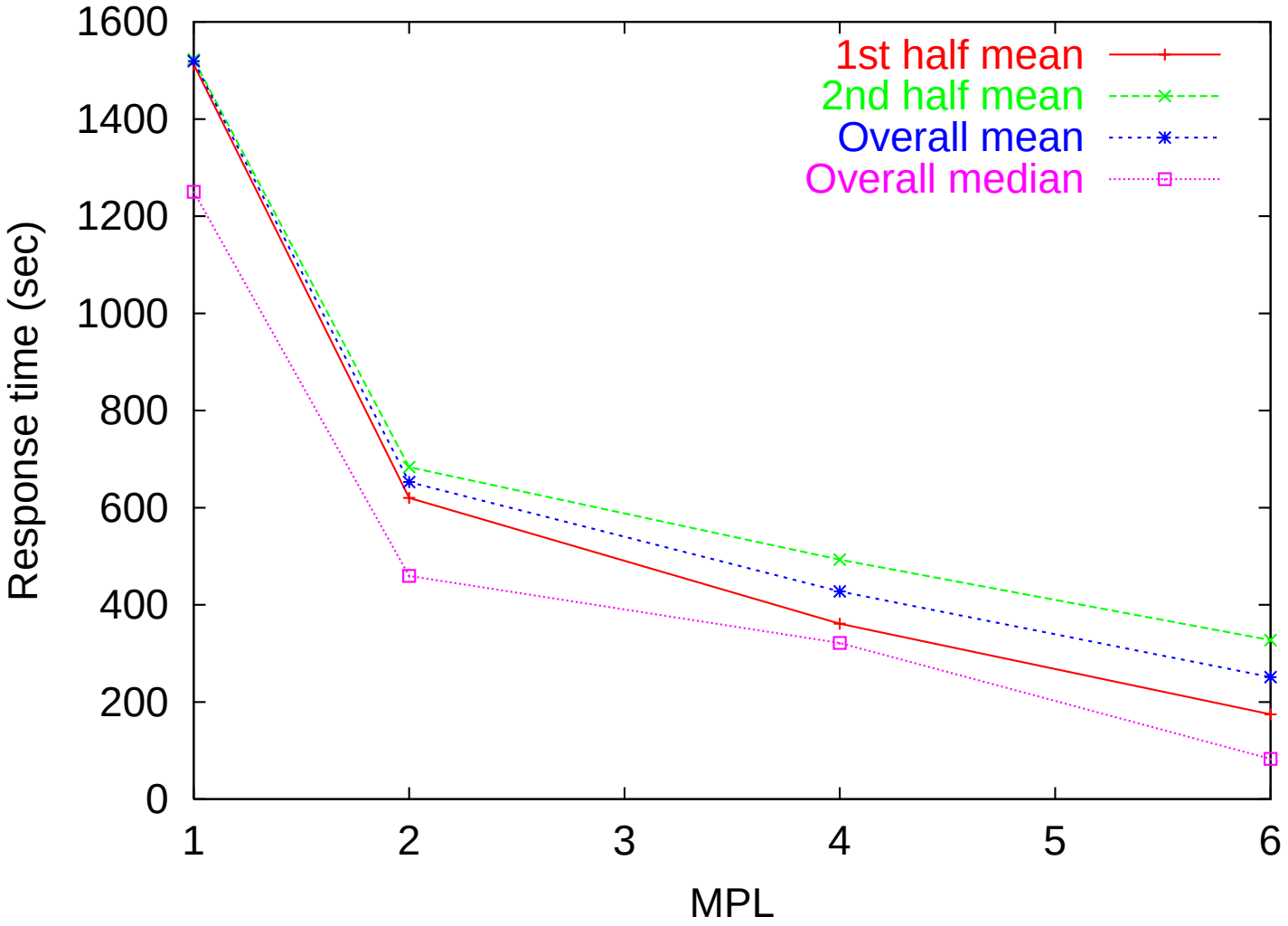
Dynamic Workload Methodology

- 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



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.)

- Experimental performance evaluation of scheduling algorithm demonstrates:
  - Multiprogramming parallel jobs is feasible and improves performance
  - FCS adjusts well to various application and load-balancing requirements
  - Better scheduling algorithms handle higher dynamic loads

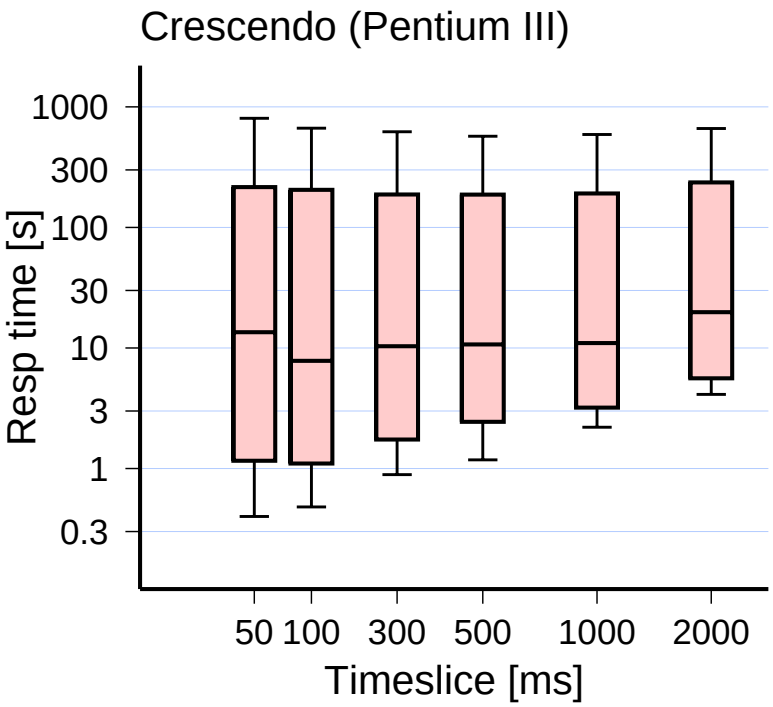
Active Research Areas

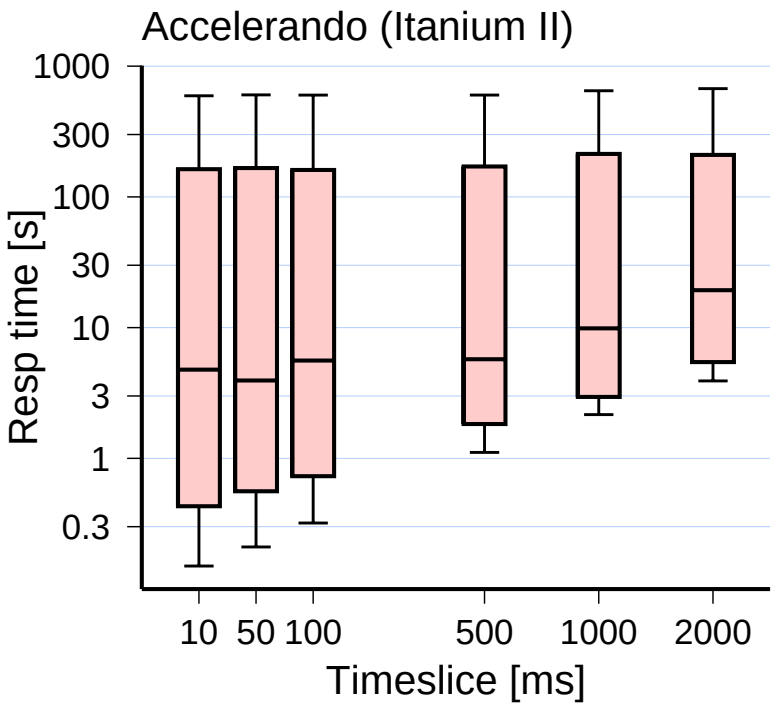
- Resource management [SC'02]
- User-level communication libraries [SC'03]
- Transparent fault-tolerance [IPDPS'04]
- Parallel I/O

For more information:

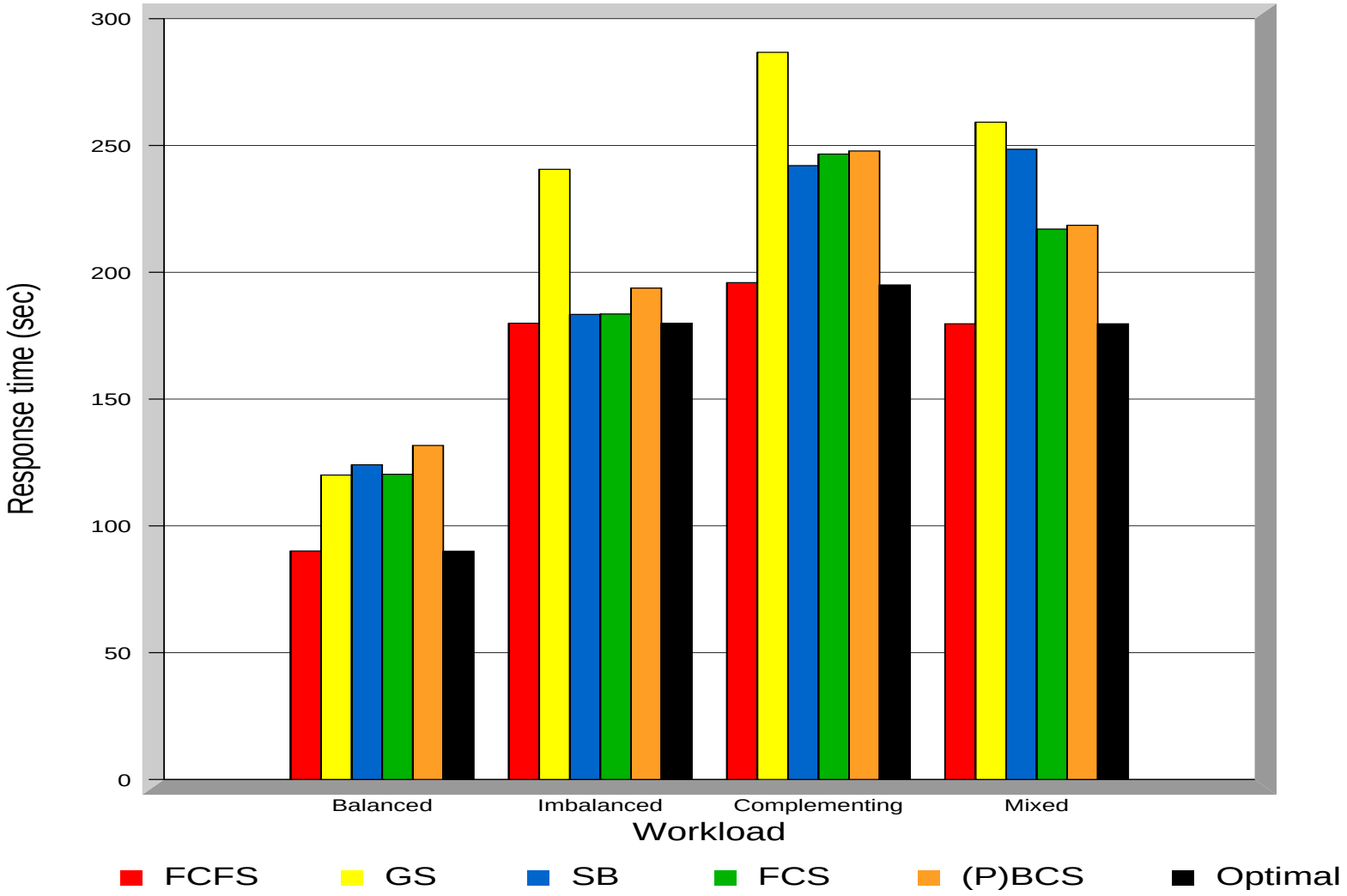
http://www.cs.huji.ac.il/~etcs/research.html

Timeslice – Response Time

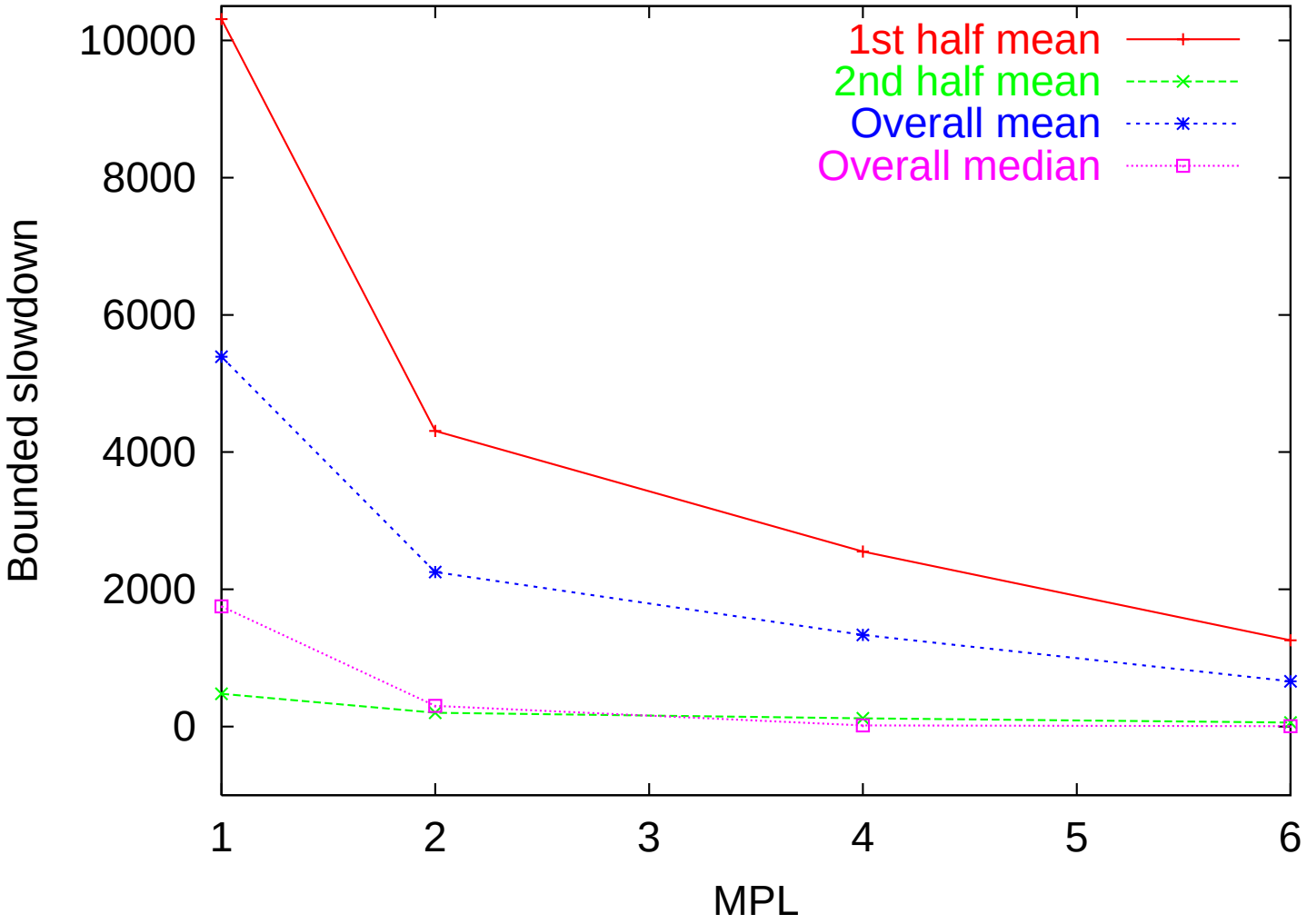




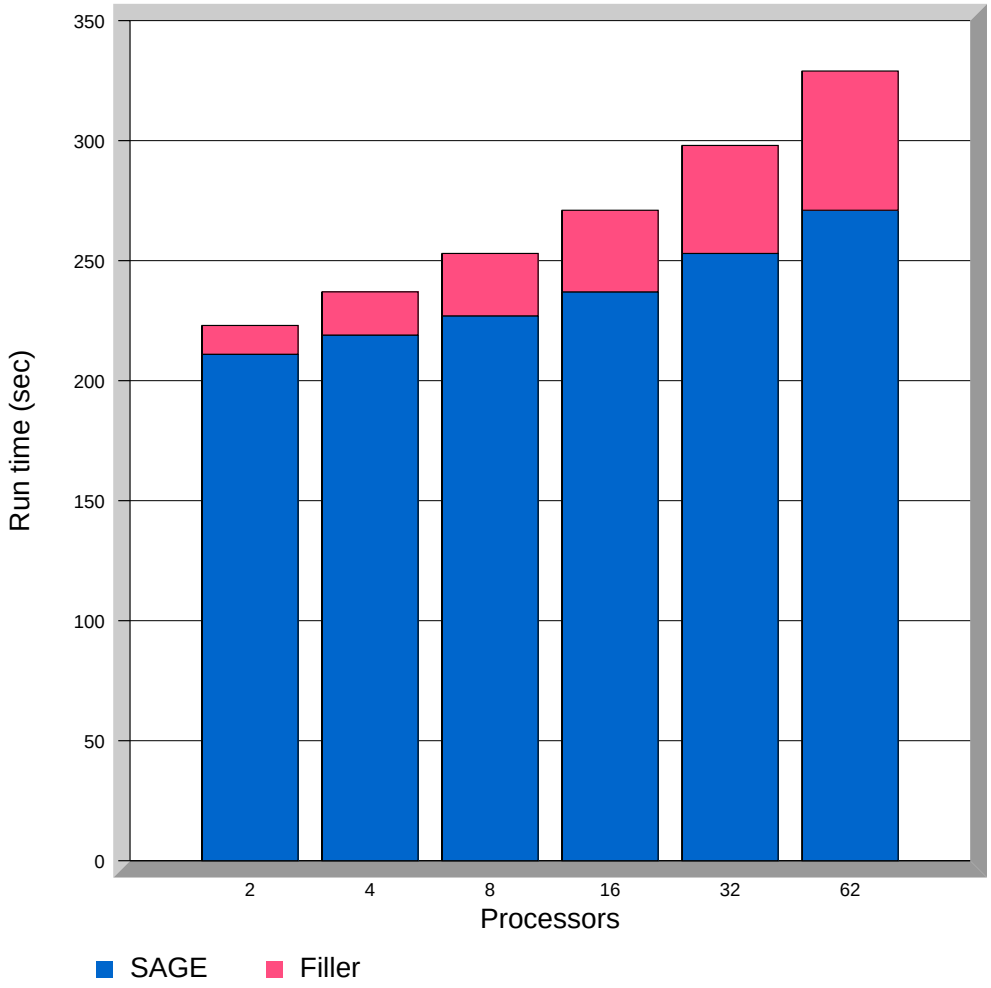
Response Time



FCFS vs. GS and MPL (2)



Resource Overlapping



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

Resource Management

Scalable Tool for Resource Management

TORM

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