Coordinated Coscheduling in Time-Sharing clusters through a Generic Framework

Saurabh Agarwal
IBM Research (India Research Labs)
IIT New Delhi – 110016, India

Co-authors:

Gyu Sang Choi	:	PhD Student, CSE Dept. , Penn State University.
Dr. Chita Das	:	Prof. (& my Advisor), CSE Dept. , Penn State University.
Dr. Andy B. Yoo	:	LLNL, Livermore, CA
Dr. Shailabh Nagar	:	IBM T.J. Watson research center, NY

PENNSTATE
1855

Clusters Today

Scientific Applications

Web / Wireless Application Servers

Typical cluster based Application Characteristics

- Parallel and Distributed.
- Possibly Communication Intensive (High/Medium/Low).
  - Very common and is our research focus.
- Requirement to progress ?
  - Timely Inter-Node communication.
- Implication ?
  - Parallel Jobs must be co-scheduled !

PENNSTATE

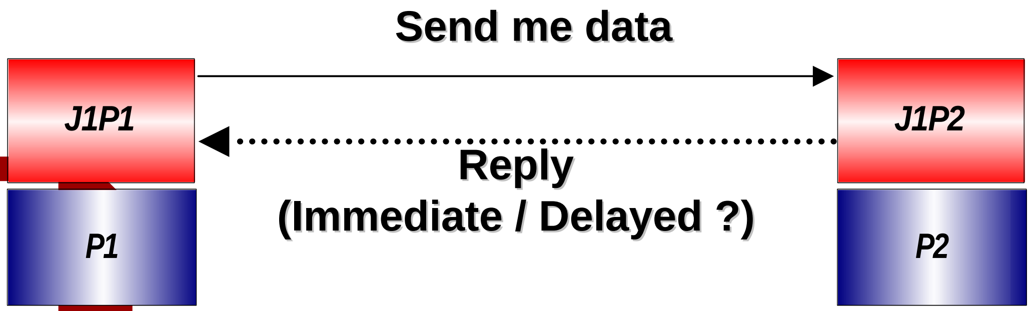
Outline Today

- Wh at is th co **heduling** problem ?
- Ho v has be solved **earlier** ?
  - Batc ang, DCS, ICS (SB), PB
- Why prio olutions **not enough** ?
- What do **propose** ?
- What are e **results** ?
- What can e **conclude** ?
- Is there a **future** for this ?

PENNSTATE

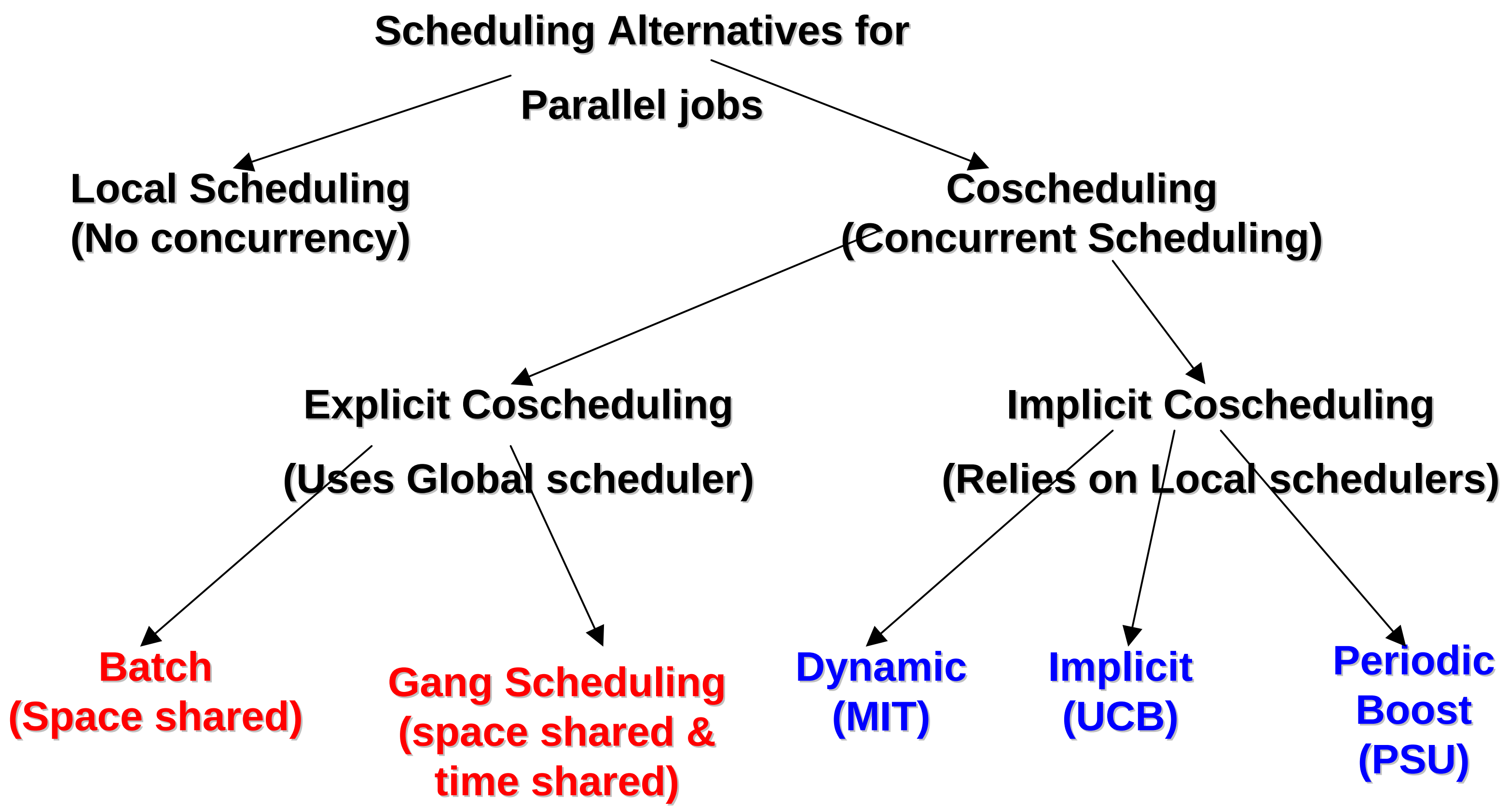
What is CoScheduling ?

CoScheduling* : **Co**ncurrently **S**chedule processes of a parallel job on individual nodes of a time-sharing cluster.



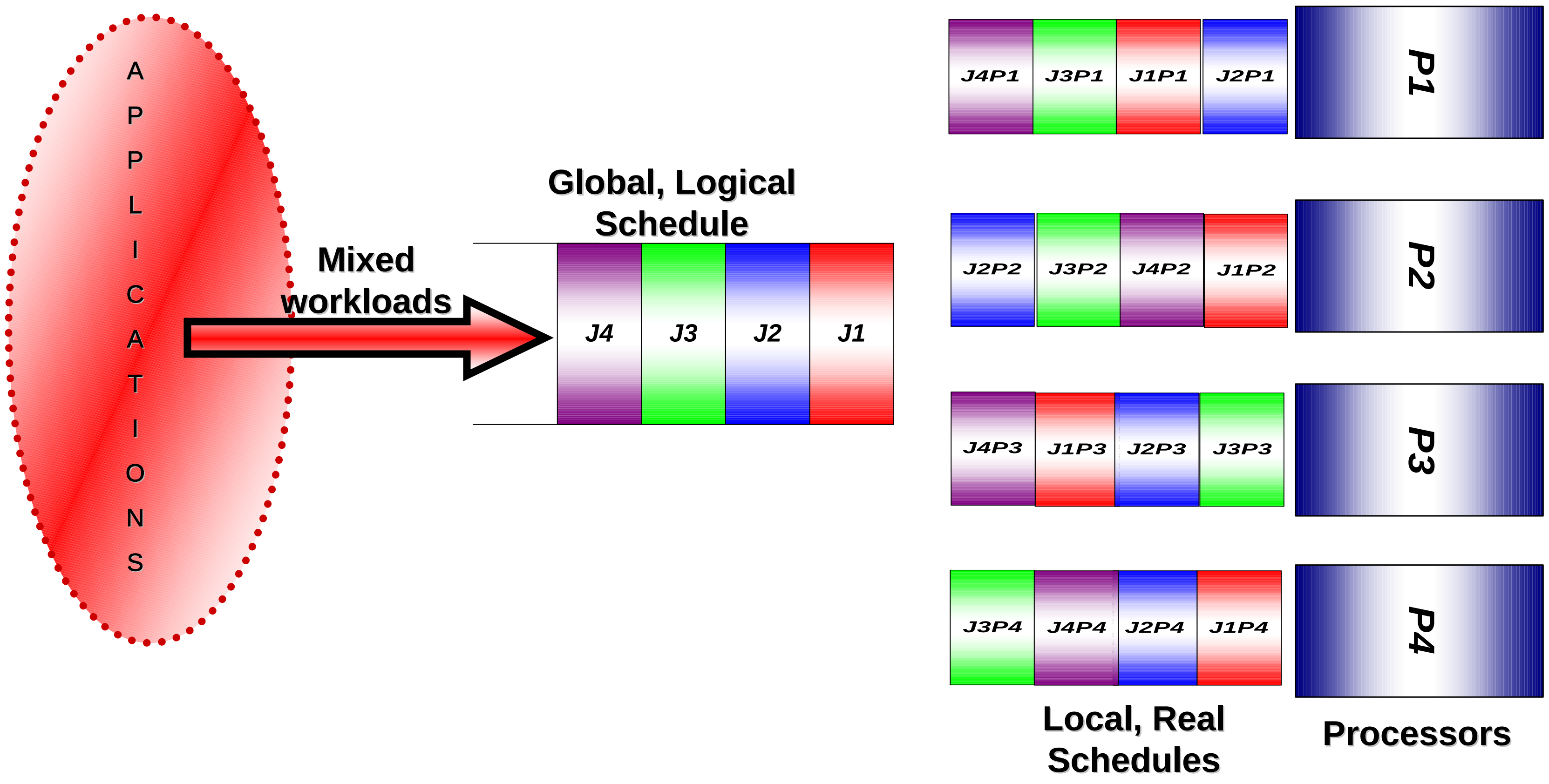
Why Needed ? : **Performance**

Parallel job Scheduling Techniques (Hierarchy)

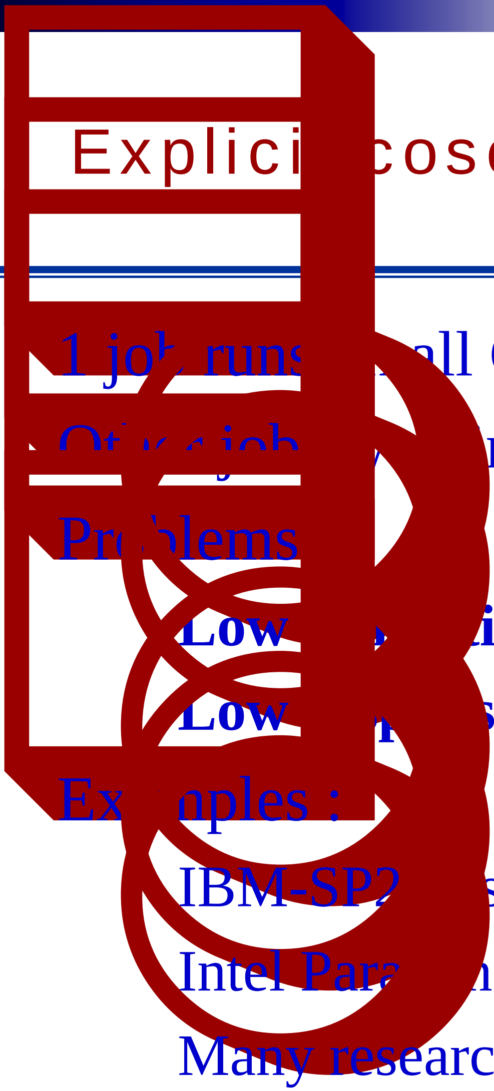


Scheduling Alternatives for Parallel jobs
Local Scheduling (No concurrency)
Coscheduling (Concurrent Scheduling)
Explicit Coscheduling (Uses Global scheduler)
Implicit Coscheduling (Relies on Local schedulers)
Batch (Space shared)
Gang Scheduling (space shared & time shared)
Dynamic (MIT)
Implicit (UCB)
Periodic Boost (PSU)

Un-coordinated (Local) scheduling

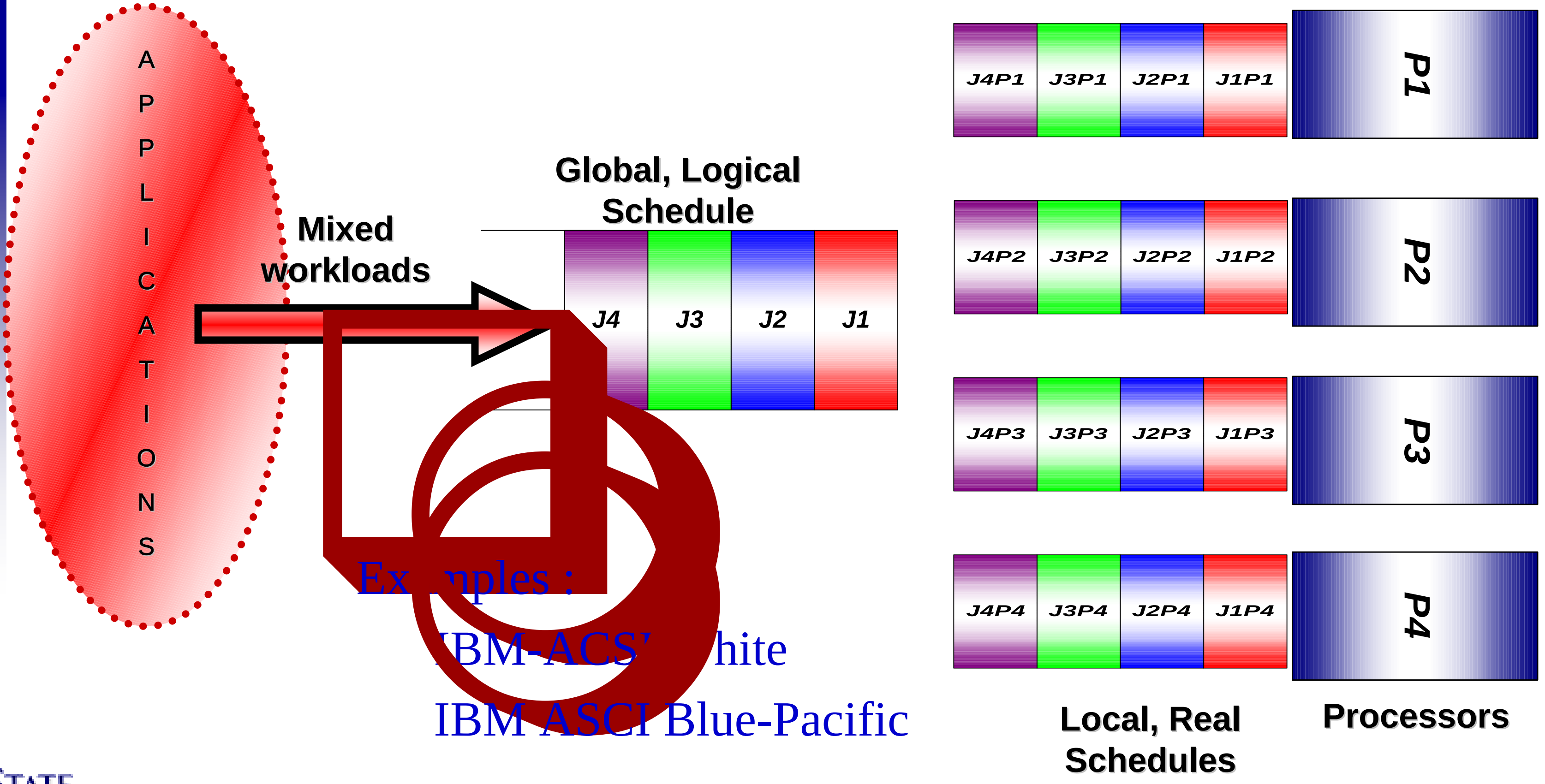


Explicit coscheduling: Batch Scheduling



- 1 job runs on all CPUs until completion
- Other jobs wait in scheduler queue for their turn
- Problems
  - **Low utilization**
  - **Low response times**
- Examples :
  - IBM-SP2 (Uses Load Leveler)
  - Intel Paragon (Uses Network Queuing System (NQS))
  - Many research COTS clusters use PBS

Explicit Coscheduling: Gang Scheduling



Ex[illegible]mples :

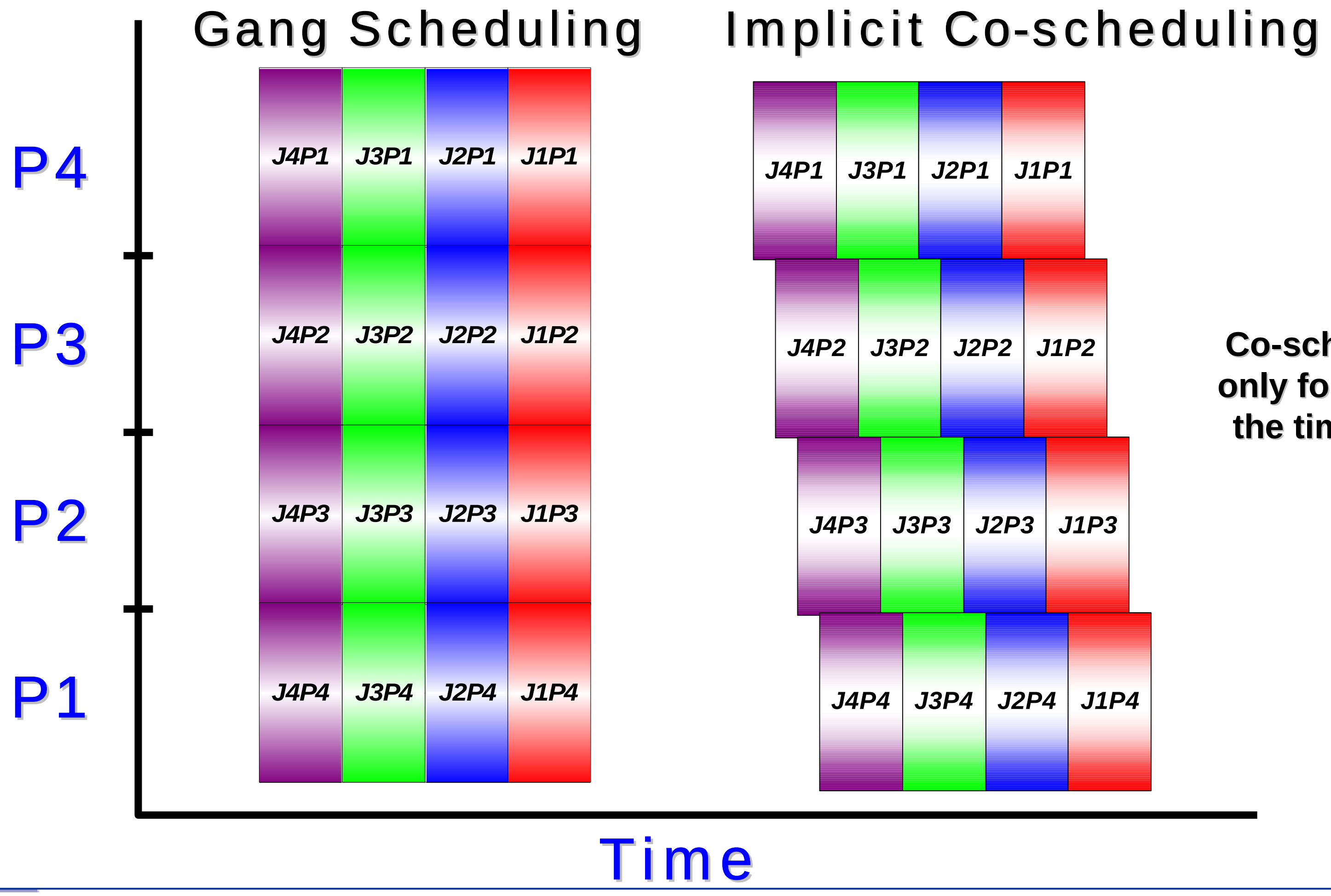
[illegible]IBM-ACS[illegible]hite

IBM ASCI Blue-Pacific

Why [illegible]o Gang Scheduling on Clusters ?

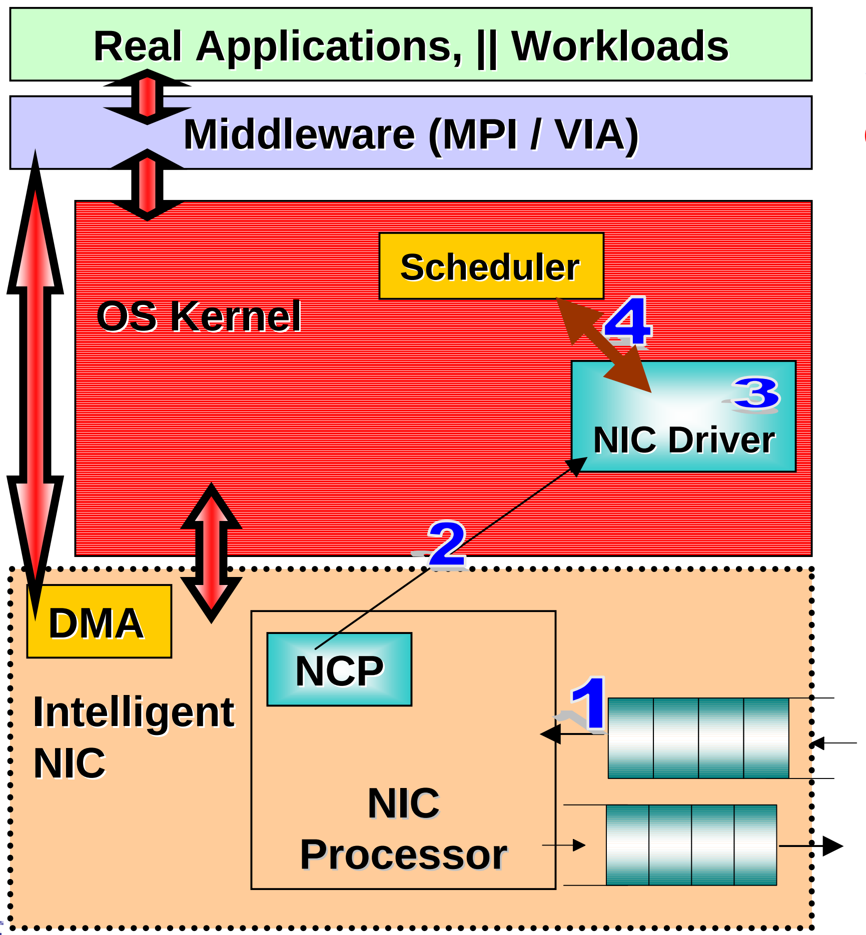
- Ne[illegible]d a [illegible]heduler controlling jobs on ALL nodes
  - Im[illegible]nplement as :-
    - [illegible]quent Synchronization (Order milli-sec)
      - [illegible] in **NOW** because of higher wire latencies [illegible]).
    - R[illegible]nge in ‘time quantum’ (of order seconds).
      - [illegible]Reduces system responsiveness.
- Not scalable for NOW (For same reason as above).

What is the solution ?



Co-scheduled only for part of the time slot.

DCS : Dynamic Coscheduling



“***Dynamic Adjustment***” of process priorities by using “***messages***” (from communicating jobs) as hints.

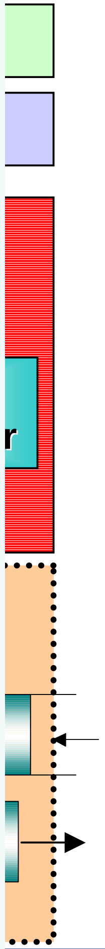
NIC Co[illegible]ol Program (NCP) senses i[illegible]ming message.

NCP rai[illegible] interrupt(s) to NIC Dri[illegible]r, *if required*.

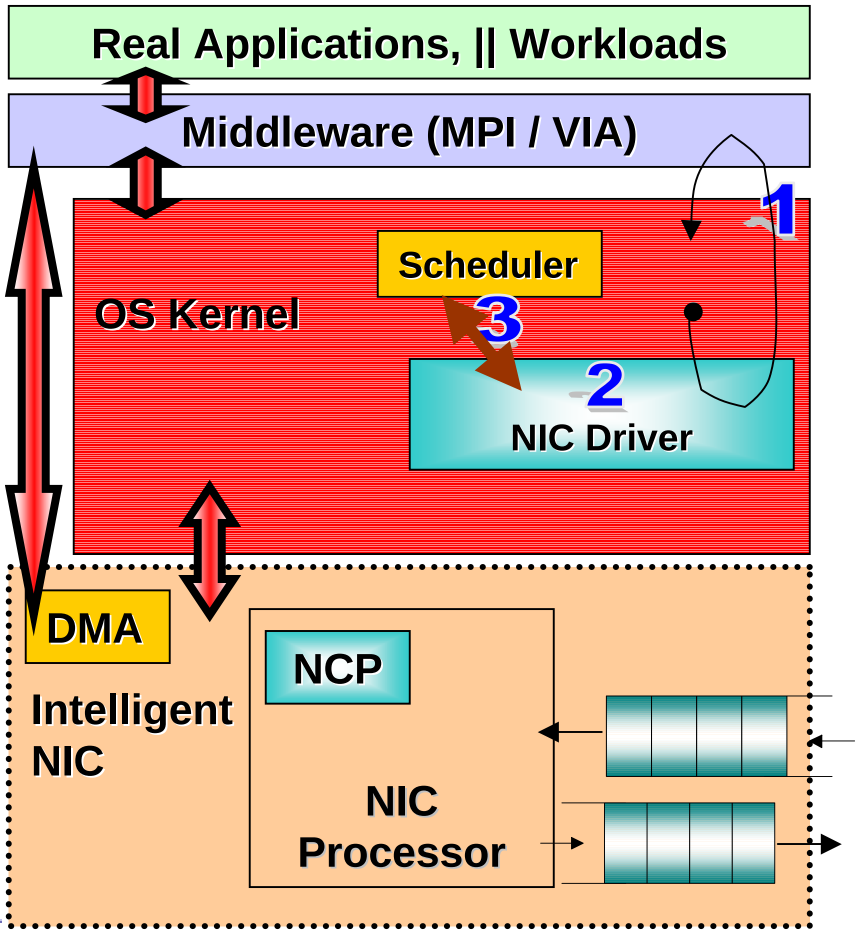
NIC Dri[illegible]r *finds* & *places* relevant [illegible]ocess into the highest-[illegible]ority queue.

Scheduler schedules that process.

in Block



PB : Periodic Boost



"***Periodically***" check end-points in host memory, "**Boost**" the process with pending messages.

Driver p s VI end-points in user-pin d memory, every 10 ms.

Picks 1s ending-message VI , finds co esponding process, boosts i riority.

Scheduler schedules that process next.

'Imp[illegible]cit' Coscheduling: Various Issues

DC[illegible]

- To[illegible] [illegible]nterrupts (< 1ms)
- Re[illegible] : [illegible] VI, instea[illegible] of per process

[illegible]

- No [illegible]
- No [illegible]de optimizati[illegible].
- Va[illegible] : [illegible]S For Ti[illegible]tly coupled (Split-C) environment.

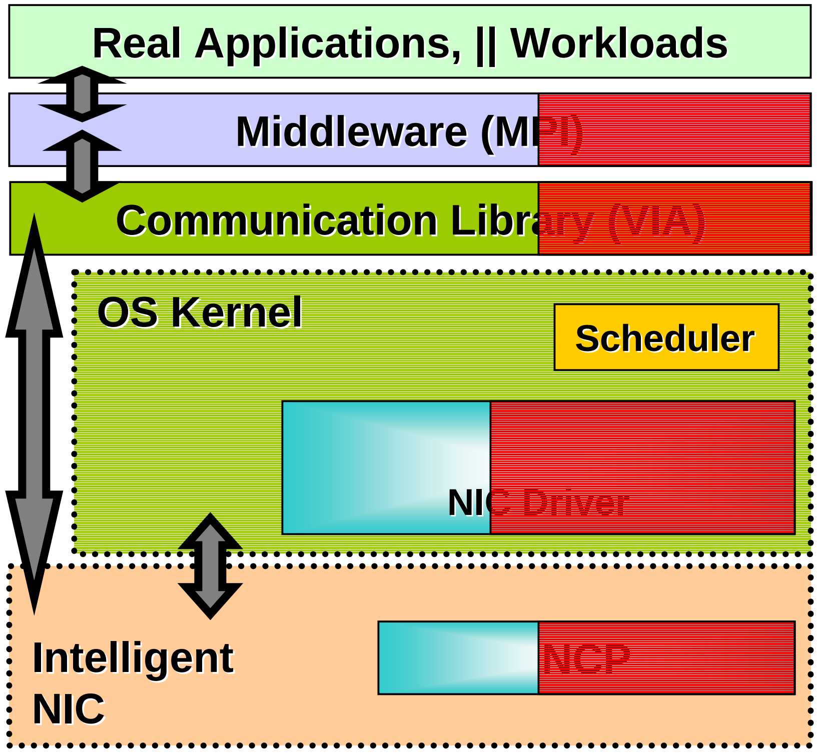
PB[illegible]

- U[illegible]rent form.
- De[illegible]uracy : Polling done after DMAs
- Lo[illegible]ing in fair version. (Too many polls)
  - Reason : Per VI, instead of per process.

No Commercial Implementation. WHY ?

1. La[illegible]k of exha[illegible]ve experimentation on multiple platforms.
   - Not easy [illegible]code custom-solutions for each platform.
   - No gene[illegible] standard approach available.
2. No[illegible]scheme b[illegible]for all types of workloads.
   - None pr[illegible]ses extensibility, generality, adaptability ?
   - Support [illegible]QoS not addressed at all.
3. No[illegible]eal ince[illegible]es (results) demonstrated yet
   - Coschedu[illegible]g against batch scheduling ?
   - Presence [illegible]other sequential workloads (CPU, I/O) ?
   - High Multi-programming degree ?

Addressing global issue 1 :- Prior Design



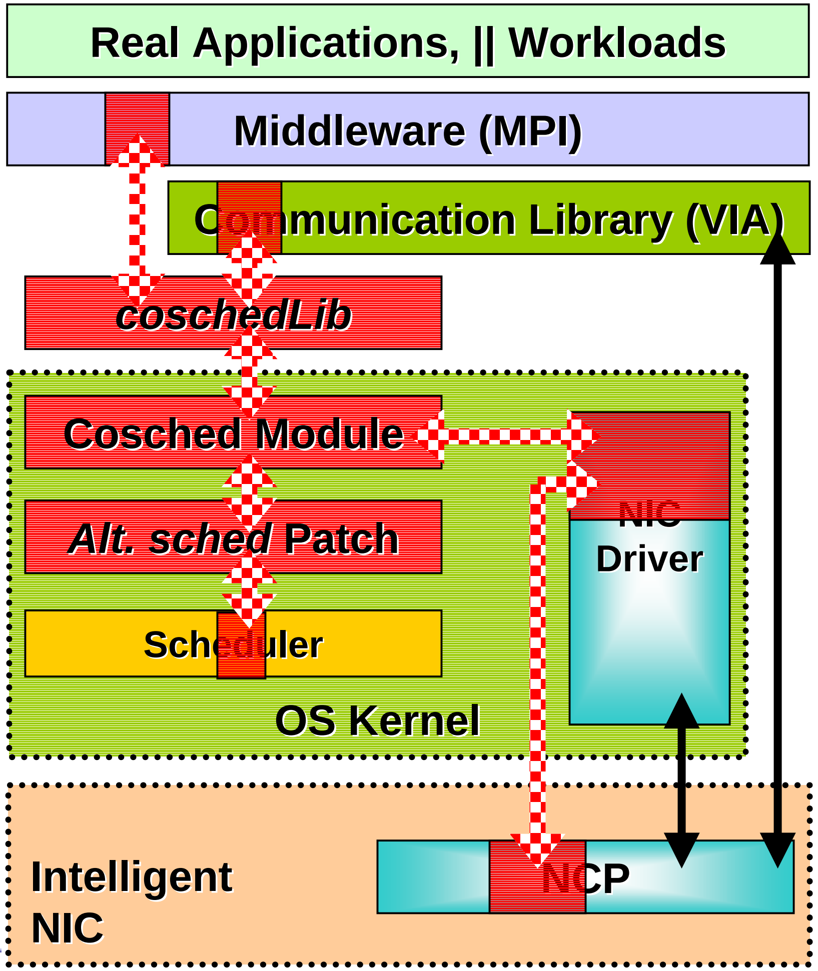
P[illegible]rts of co[illegible] to change :

- *MPI Libr[illegible]y (SB)*
- *VIA Libr[illegible]y (SB, PB)*
- *Device [illegible]ver (DCS, SB, PB)*
- *Firmware (DCS, SB, PB)*

D[illegible]

Flexibilit[illegible]	Tight driver/firmware coupling
General[illegible]y :	Different implementations for each scheme.
Modula[illegible]y :	No re-use across platforms.
Portabi[illegible]y :	No standard interface
Integrity :	Local scheduler isolated.

Proposed Design



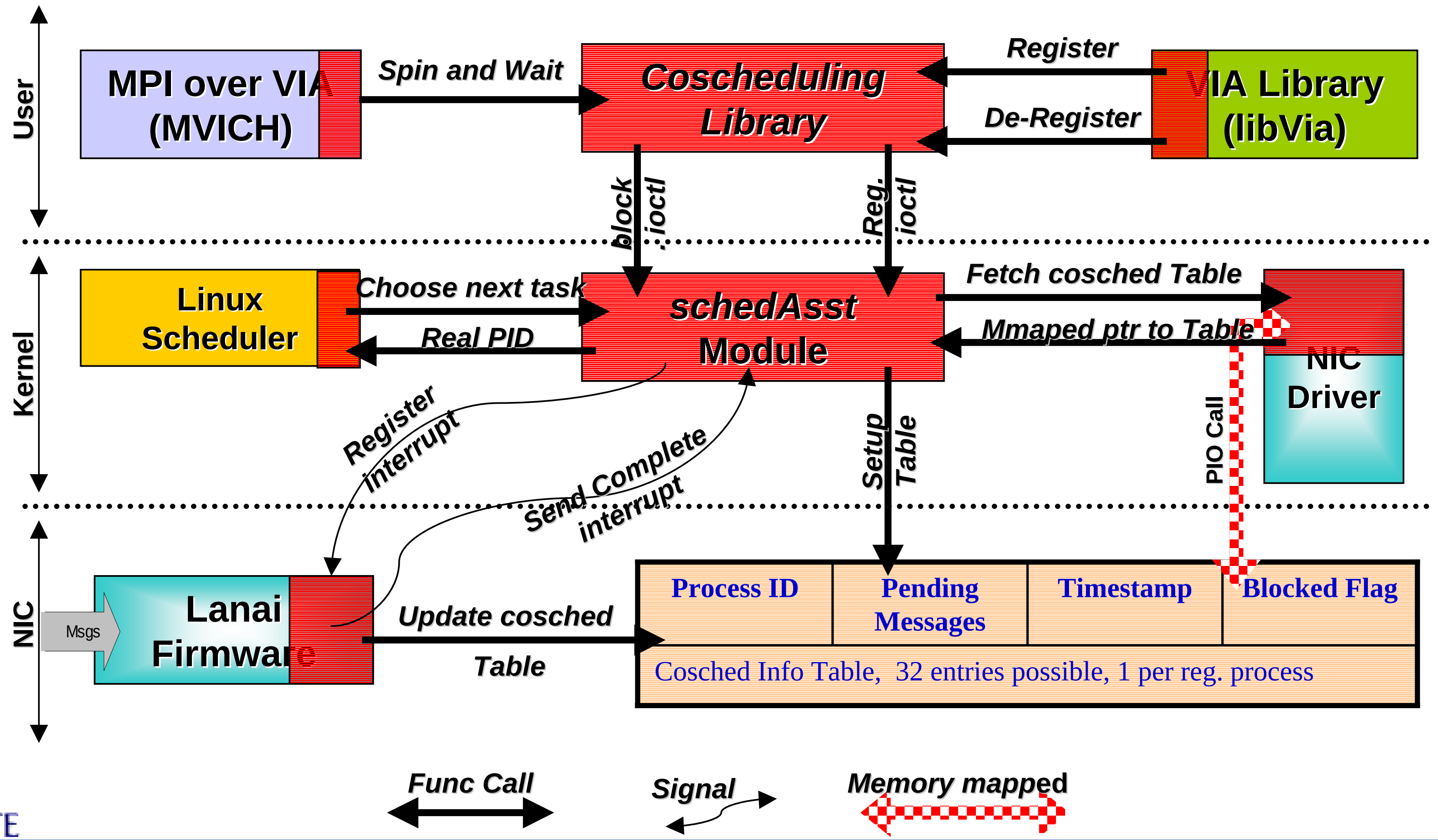
B[illegible]g Picture :

- Identify [illegible]nd isolate "*policies*" and "*m*[illegible]*anisms*".
- Re-use "[illegible]*olicies*"
- Re-implement "*mechanisms*"

A[illegible]lvantages

Flexibil[illegible] :	Coscheduling independent of device driver / firmware.
General[illegible]y :	Same module for all schemes.
Modula[illegible]y :	Re-use across platforms.
Portabi[illegible]y :	Standard interfaces defined.
Integrity :	Local scheduler involved.

Addressing global issue 2 :- Coordinated Coscheduling

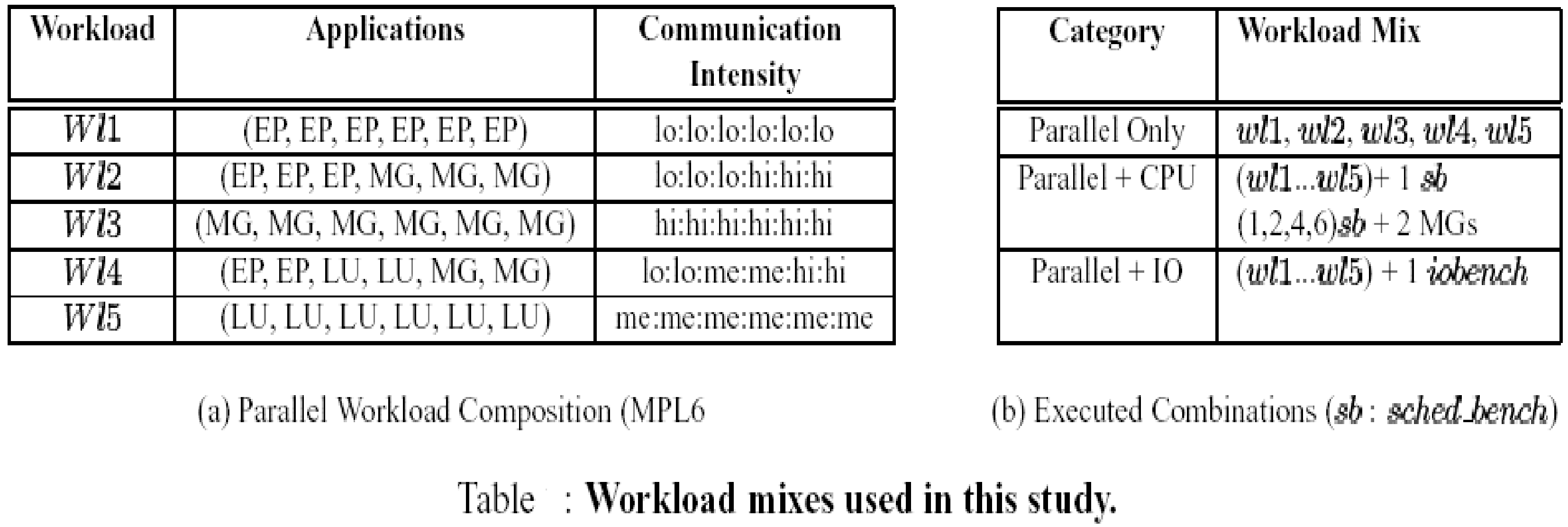


Addressing global issue 3 :- Workload / Environment

16 node Myrinet connected cluster, 1GB RAM .

Linux 2.4 , MPI over VIA, Berkeley-VIA

Lanai-9 Myrinet NI cards, 8MB on-chip memory.



Workload	Applications	Communication Intensity
*Wl*1	(EP, EP, EP, EP, EP, EP)	lo:lo:lo:lo:lo:lo
*Wl*2	(EP, EP, EP, MG, MG, MG)	lo:lo:lo:hi:hi:hi
*Wl*3	(MG, MG, MG, MG, MG, MG)	hi:hi:hi:hi:hi:hi
*Wl*4	(EP, EP, LU, LU, MG, MG)	lo:lo:me:me:hi:hi
*Wl*5	(LU, LU, LU, LU, LU, LU)	me:me:me:me:me:me

(a) Parallel Workload Composition (MPL6

Category	Workload Mix
Parallel Only	*wl*1, *wl*2, *wl*3, *wl*4, *wl*5
Parallel + CPU	(*wl*1...*wl*5)+ 1 *sb* (1,2,4,6)*sb* + 2 MGs
Parallel + IO	(*wl*1...*wl*5) + 1 *iobench*

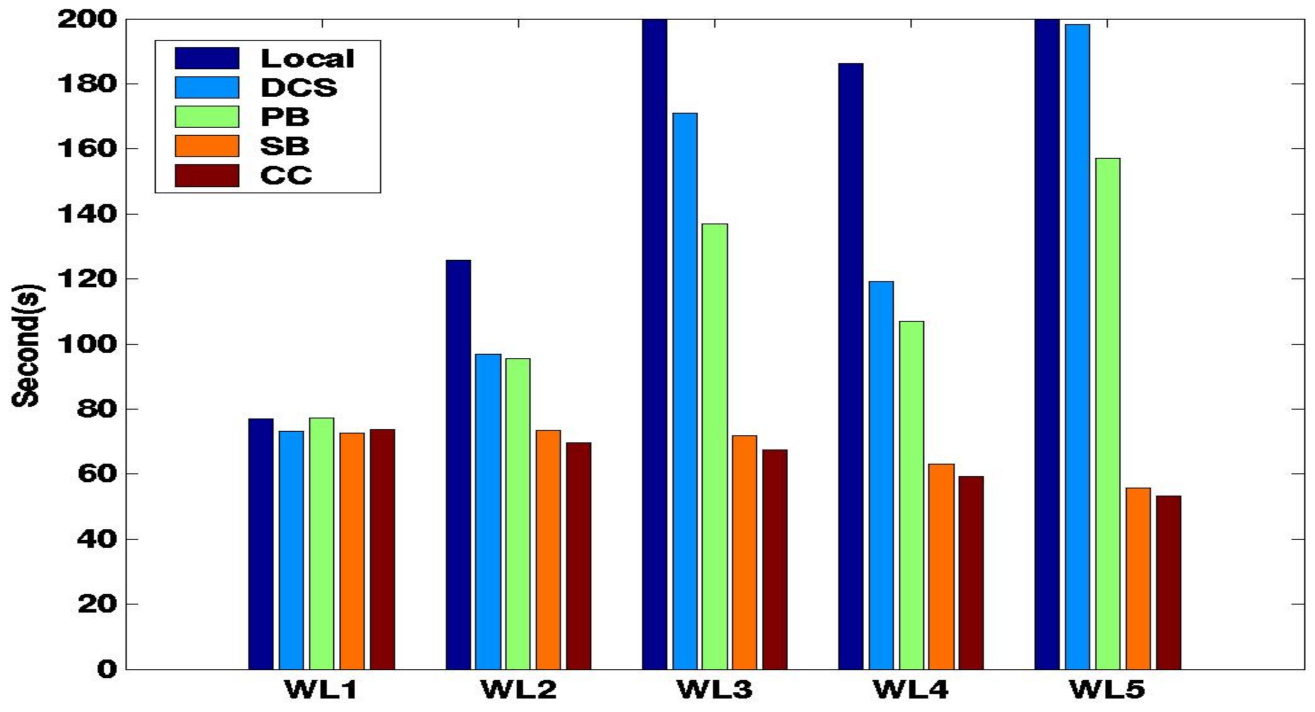
(b) Executed Combinations (*sb* : *sched_bench*)

Table : **Workload mixes used in this study.**

Performance : Execution Time

MPL = 6

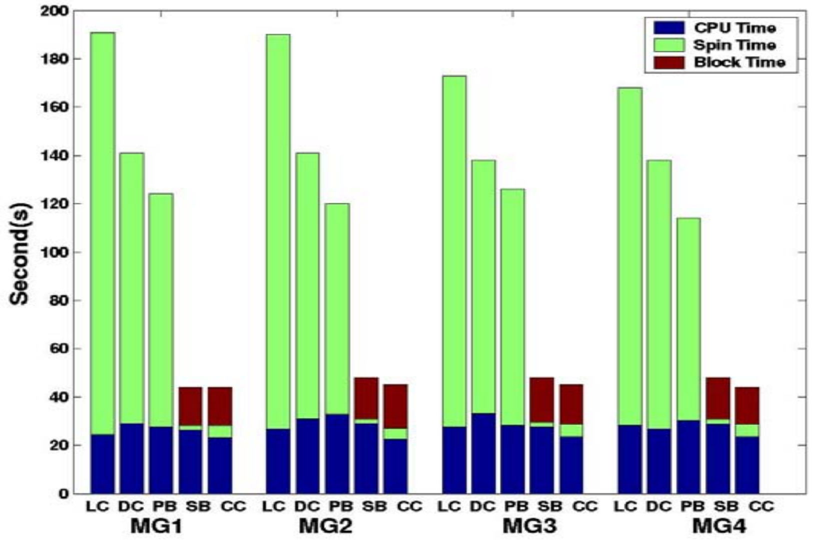
CC > SB > PB > DCS > Local



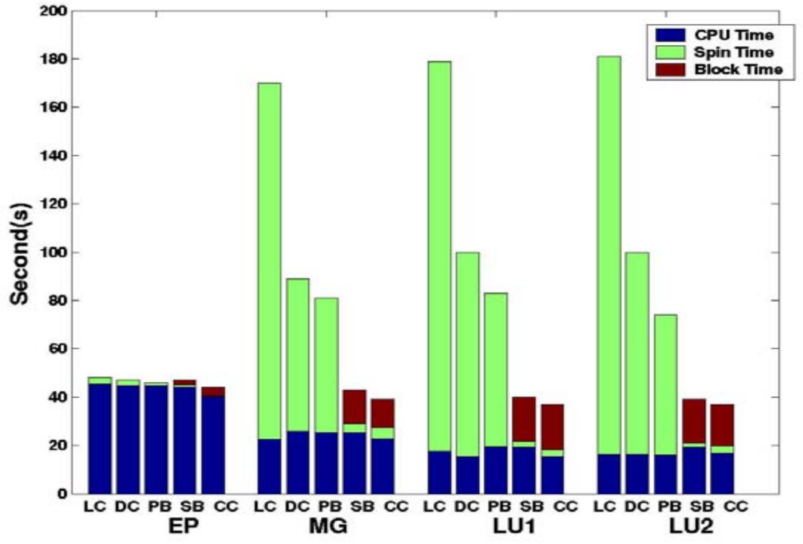
Where does time go ?

Total CPU Time = Useful work time + Non-useful spin time

Wall Clock Time = Total CPU time + Block time + others...



Workload – ***wl3***

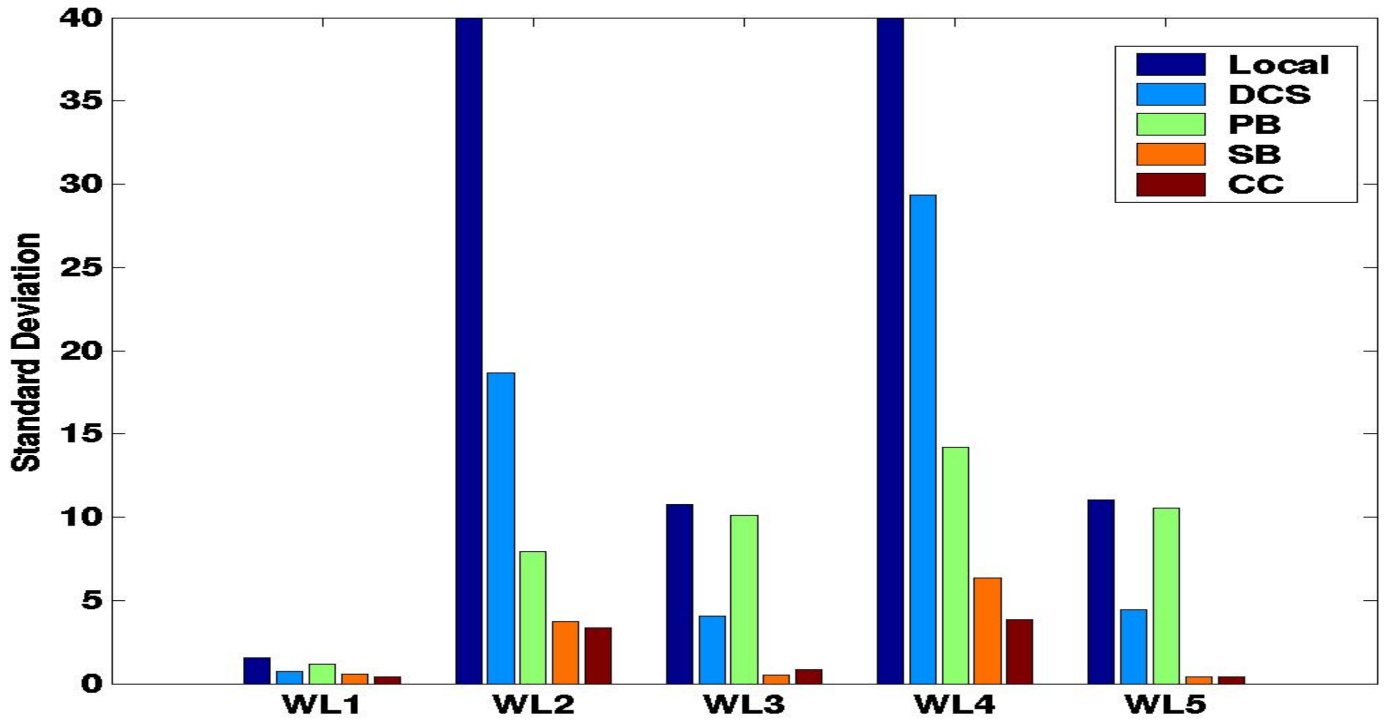


Workload – ***wl4***

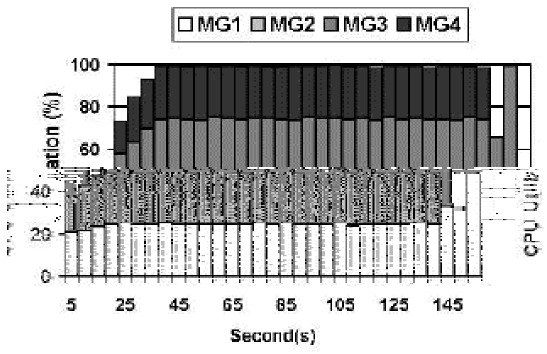
Tolerance : Standard Deviation

Wl2, Wl4 : Mixed

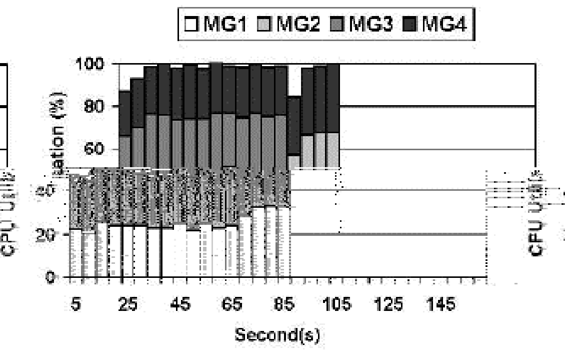
Wl1, Wl3, Wl5 : Uniform



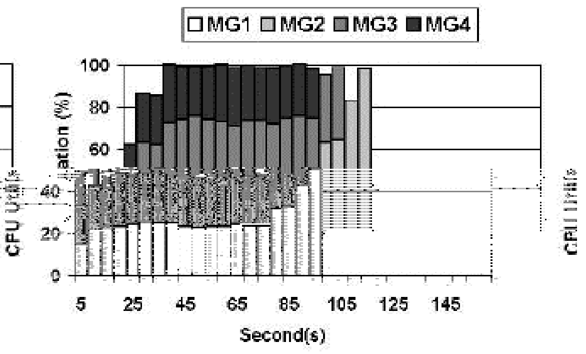
Fairness : CPU Utilization



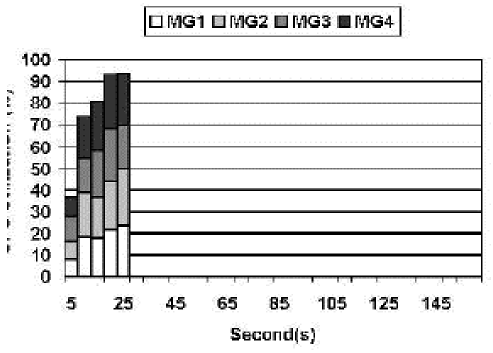
Local



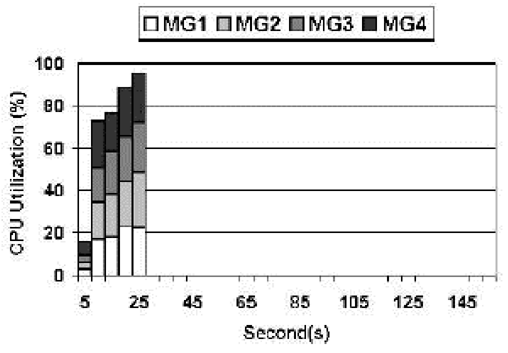
DCS



PB



SB



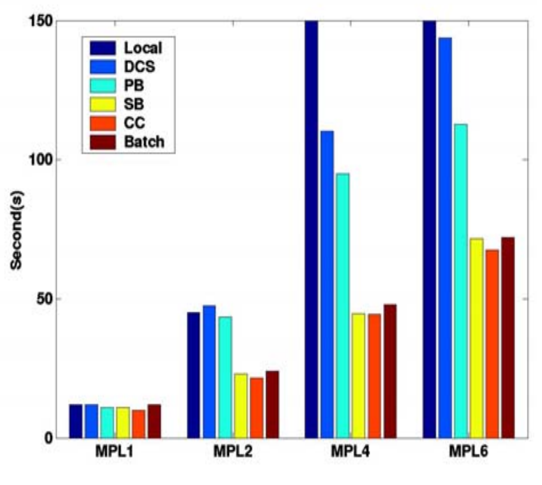
CC

Figure : *CPU Utilization results for wl3 (MPL=4)*

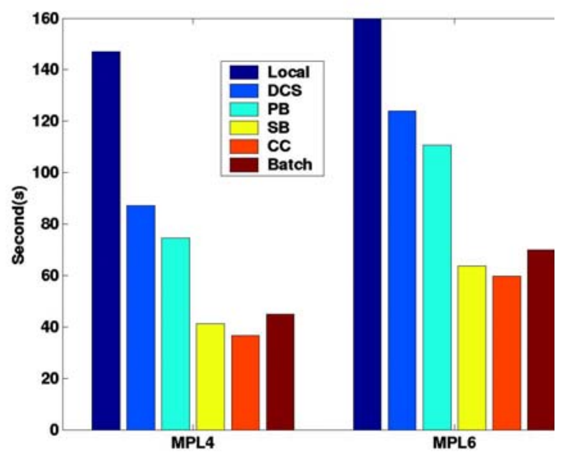
What are the **results** ? - CPU utilization on a representative node.

[illegible]calability (vertical)
Effect of MPL

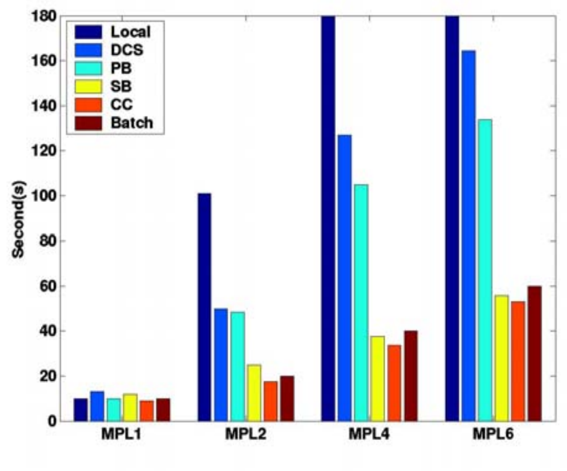
- Rate of in[illegible]ease higher in Local, DCS, PB than SB, CC.
- Glimpse comparison to batch scheduling.



Workload – *wl3*



Workload – *wl4*

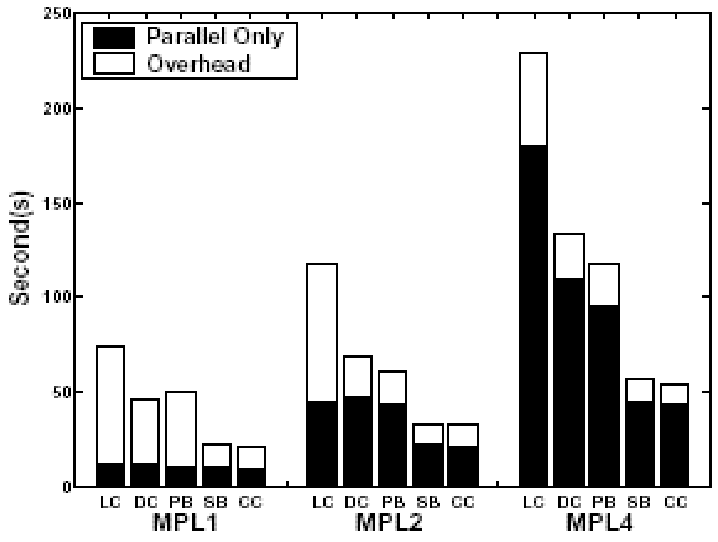


Workload – *wl5*

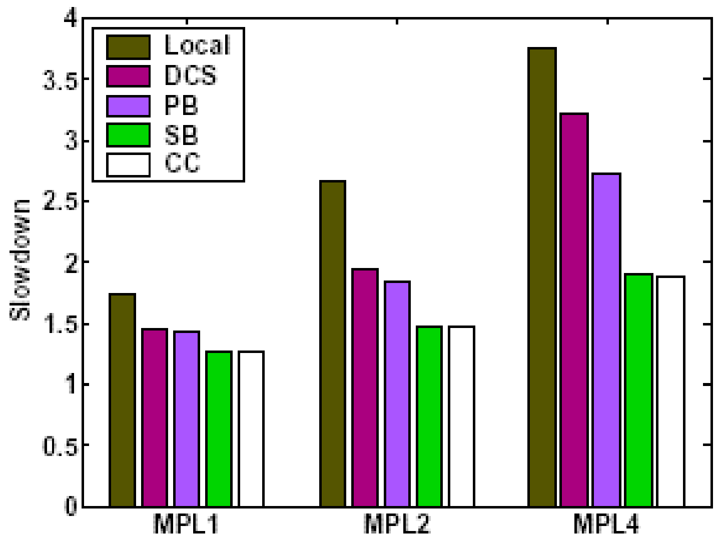
Mixin CPU intensive jobs (W/3)

CC & SB lerate load better (low *overhead* in (a)).

CC and SB exploit idle cycles well (low *slowdown* in (b))



(a) Effect on parallel job

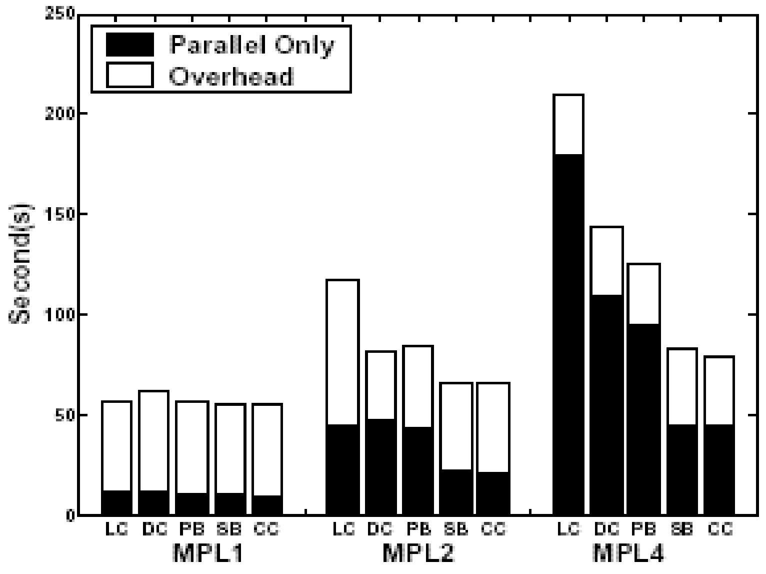


(b) Effect on CPU job

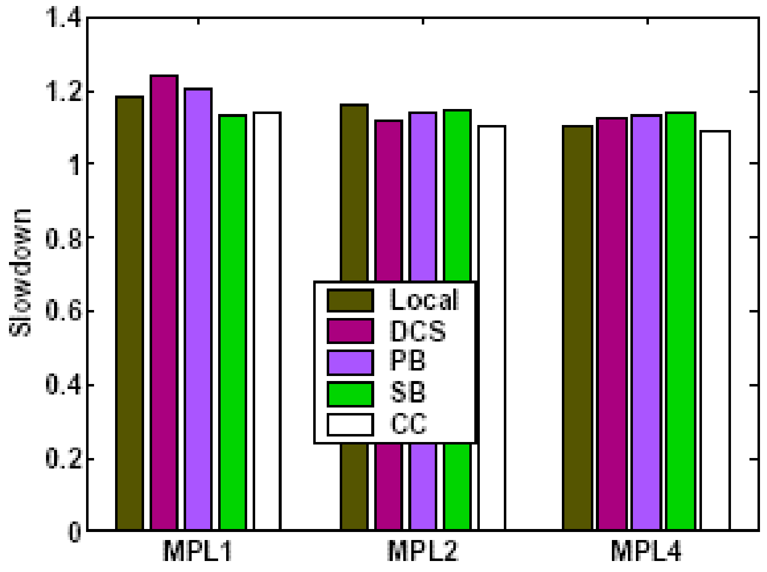
Mixing I/O intensive jobs (W/3)

|| jobs in all schemes get equally affected (similar *overheads*).

Very Insignificant *slowdown* in I/O across all schemes.



(a) Effect on parallel job.



(b) Effect on I/O job.

Conclusions

Primary Contributions :-

- Modu[illegible]exible framework for deploying coscheduling.
- Fair, [illegible]rforming, new CC scheme.

Significant findings :-

- [illegible]ed schemes better than spinning (Linux).
- CC a[illegible]cale well (Vertically) at high MPL of 6.
- CC a[illegible]get equal or better than Batch scheduling.
- CC m[illegible]ly better than SB, added QoS Potential.

Other Advantages :-

- Can impl[illegible]ent all policies with CC approach.
- Can use framework with all ULN libraries.

Future Work

- Horizontal scalability (> nodes, GM)
- Release assumptions :
  - No paging (All apps fit well in memory).
  - Homogeneous nodes availability.
  - All apps of same priority.
- Optimizations in CC mechanism.
- Allocation problems in coscheduling.
- Dynamic communication pattern identification.
- Integrated coscheduling as a feature in OS.
- True end-to-end QoS with support from scheduler.

PENNSTATE

Questions ?

Thanks for your Time !!

fference with ICS ?
(Optional slide)

- SR regist for ALL incoming messages.
- ICS regis s for expected messages for which a send has been done arlier.
- ICS is mc tightly coupled (works on a send-recv pair).
- ICS is not too suitable in MPI environment.