A Compiler-Guided Approach for Reducing Disk Power Consumption by Exploiting Disk Access Locality

S. W. Son, G. Chen, and M. Kandemir
Dept. of CSE
The Pennsylvania State University

The 4th Annual International Symposium on
Code Generation and Optimization (CGO-4)
March 28, 2006

Outline

- Motivation
- Related Work
- TPM vs. DRPM
- Our Approach
  - Single vs. Multi-processor
- Experimental Evaluation
- Conclusion

Motivation

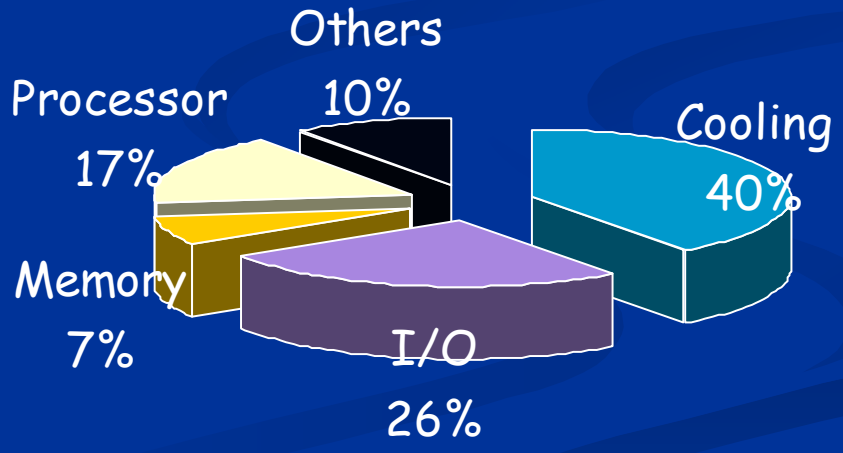
- High-end cluster/server systems consume a significant amount of power
- Disk subsystem is one of major contributors to overall power budget

Dell PowerEdge

2.1 kW

IBM eServer z990

5.3/15.8 kW

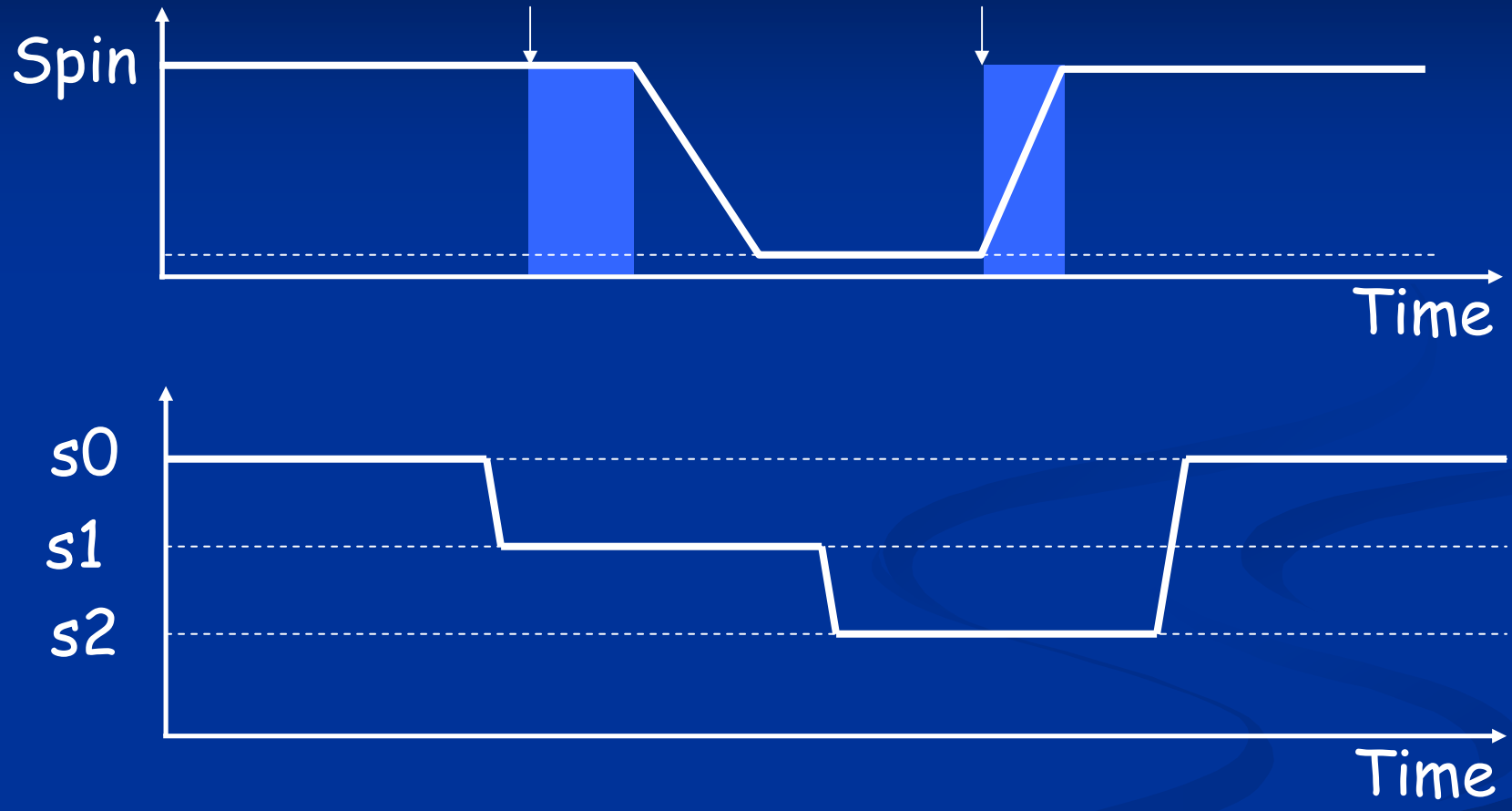


*Source: Mike Rosenfield, ACEED, February 2003.

Related Work

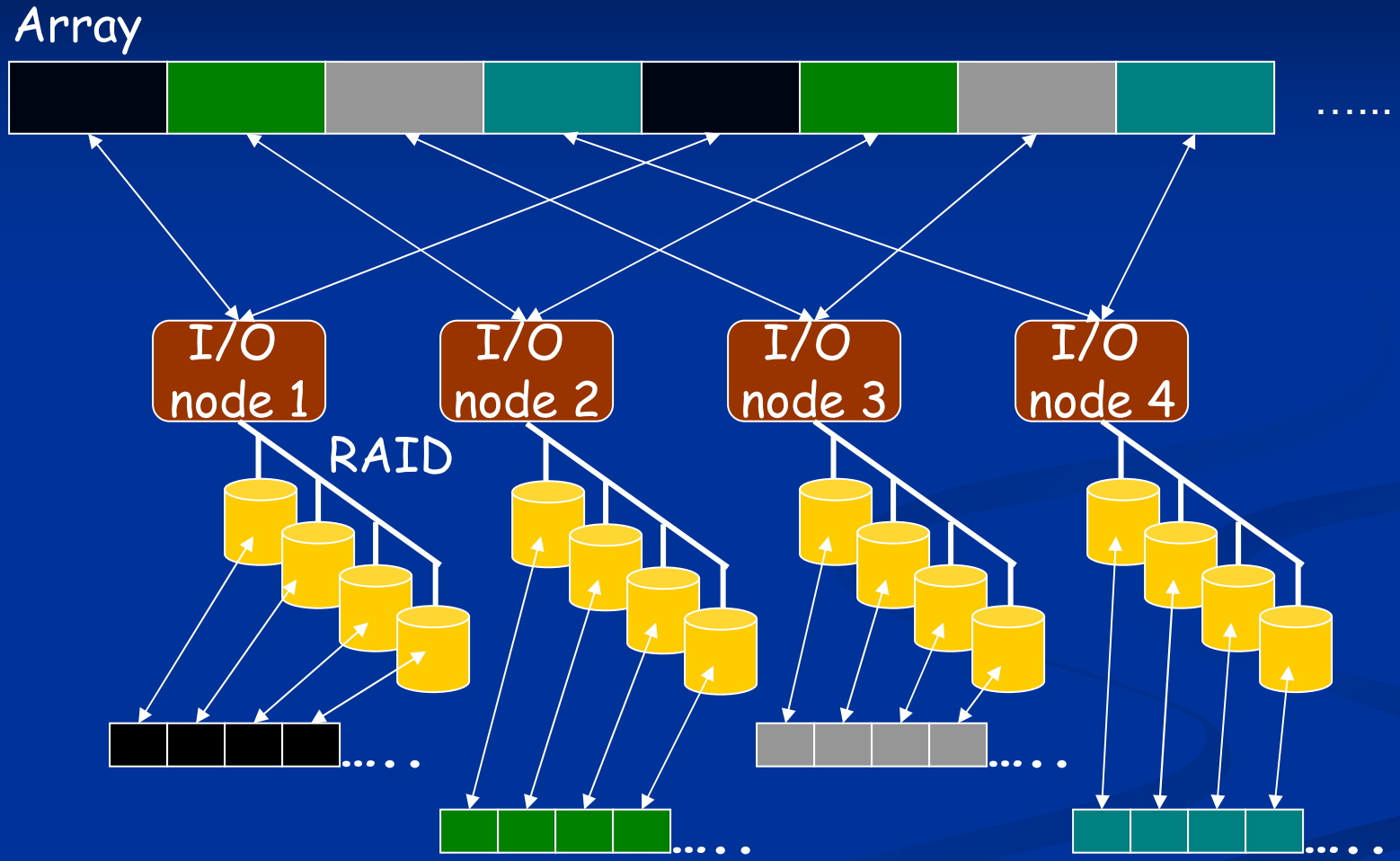
- Data Locality Optimizations
  - Target caches and main memories
  - Lots of prior studies on tiling, shackling, and fusion
- H/W Based Approaches
  - Spin up/down (TPM) vs. multi-speed disk (DRPM)
- OS Based Approaches
  - PA-LRU, PB-LRU, and PDC
- Compiler Directed Approaches
  - Compiler-directed proactive power management, energy-aware disk layout optimization
  - Our approach: compiler-directed code restructuring and generation for increasing disk idle periods

TPM vs. DRPM



- TPM disk: F. Douglis et al., "Thwarting the Power-Hungary Disk", in USENIX'94
- DRPM disk: S. Gurumurthi et al., "DRPM: Dynamic Speed Control for Power Management in Server Class Disks", in ISCA'03

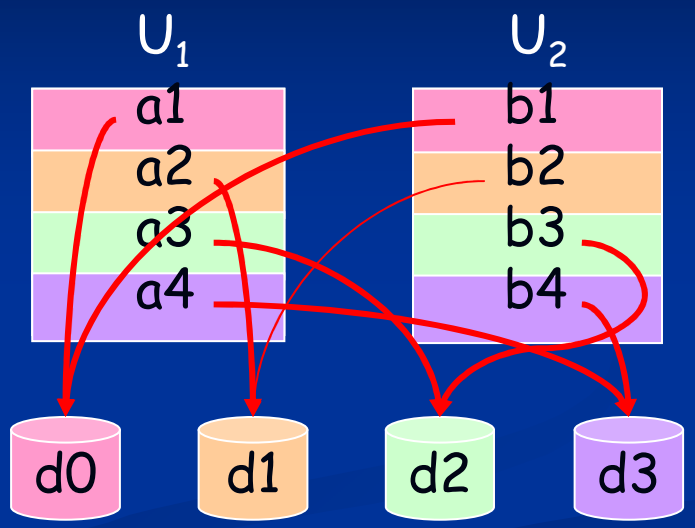
Two-Level File Striping



File Striping in Parallel File System, e.g., PVFS, GPFS

Single Processor Execution

```
L1: for i = 1..N-1
      for j = 1..N-1
        …U1[i-1][j+2]…
L2: for i = 1..N-1
      for j = 1..N-1
        …U2[i+1][2j-1]…
L3: for i = 1..N-1
      for j = 1..N-1
        …U1[2i][j+1]…
```

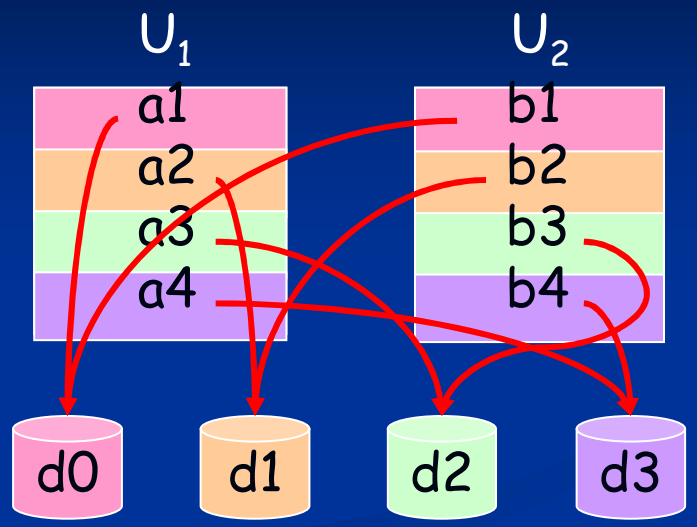


L1	×	×		
L2	×	×	×	
L3	×	×	×	×

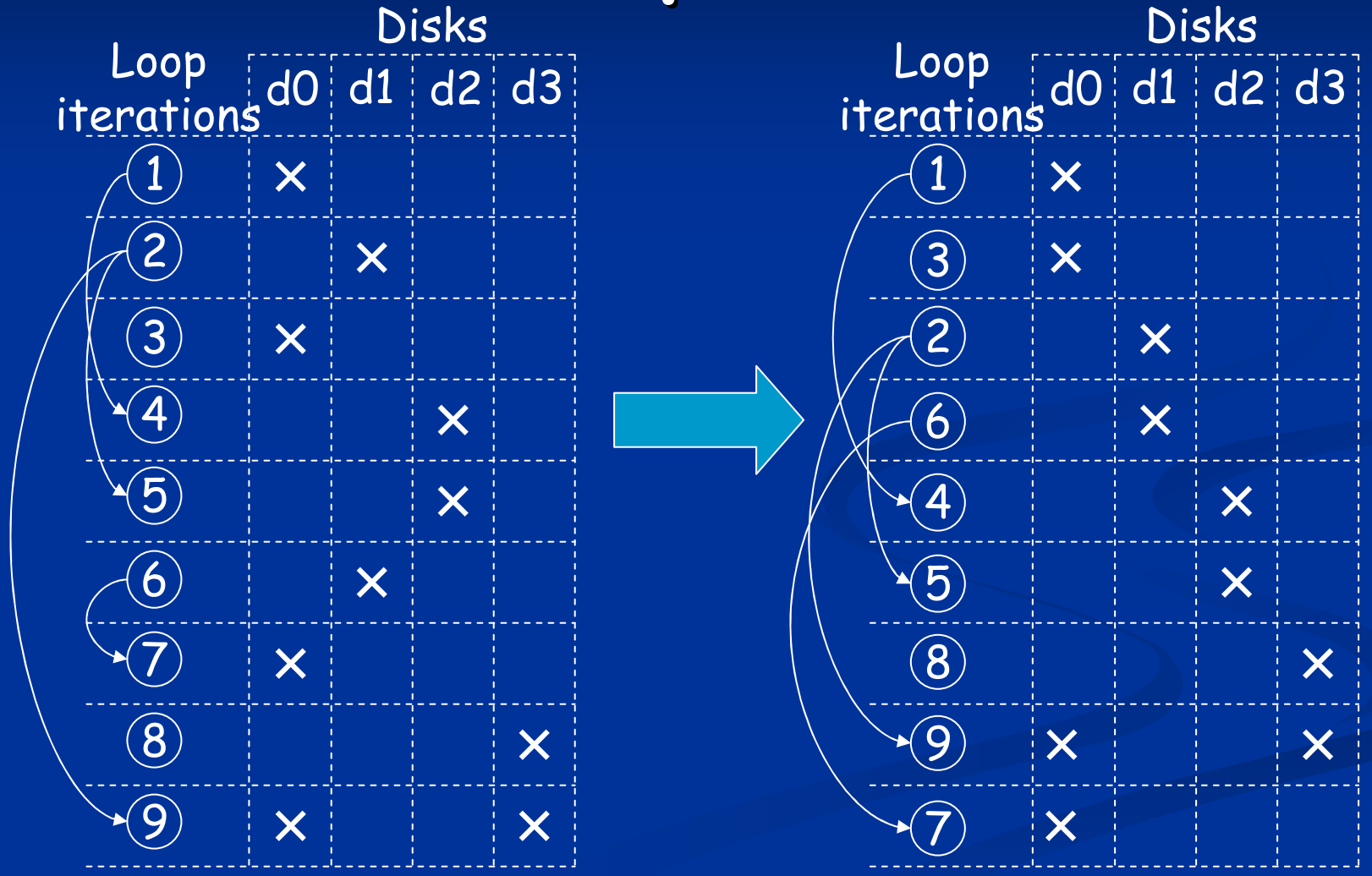
- U1 and U2 are of same size, 2N*2N

Single Processor Execution

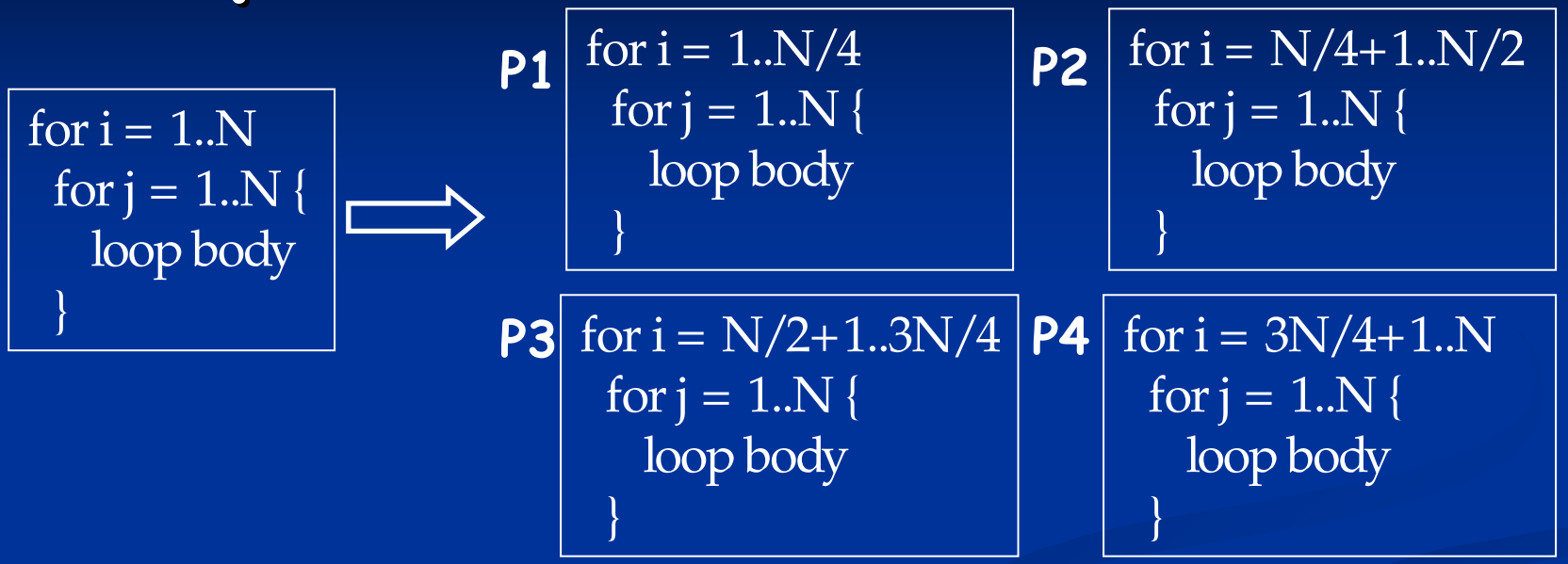
```
   for ii = 1..4 {
L1: for i = max(N/2*(ii-1)-1,1)
            ..min(N/2*ii,N-1)
     for j = 1..N-1
       …U1[i-1][j+2]…
L2: for i = max(N/2*(ii-1)-1,1)
            ..min(N/2*ii-2,N-1)
     for j = 1..N-1
       …U2[i+1][2j-1]…
L3: for i = max(N/4*(ii-1),1)
            ..min(N/4-1,N-1)
     for j=1..N-1
       …U1[2i][j+1]…
   }
```



Single Processor Execution with Data Dependences

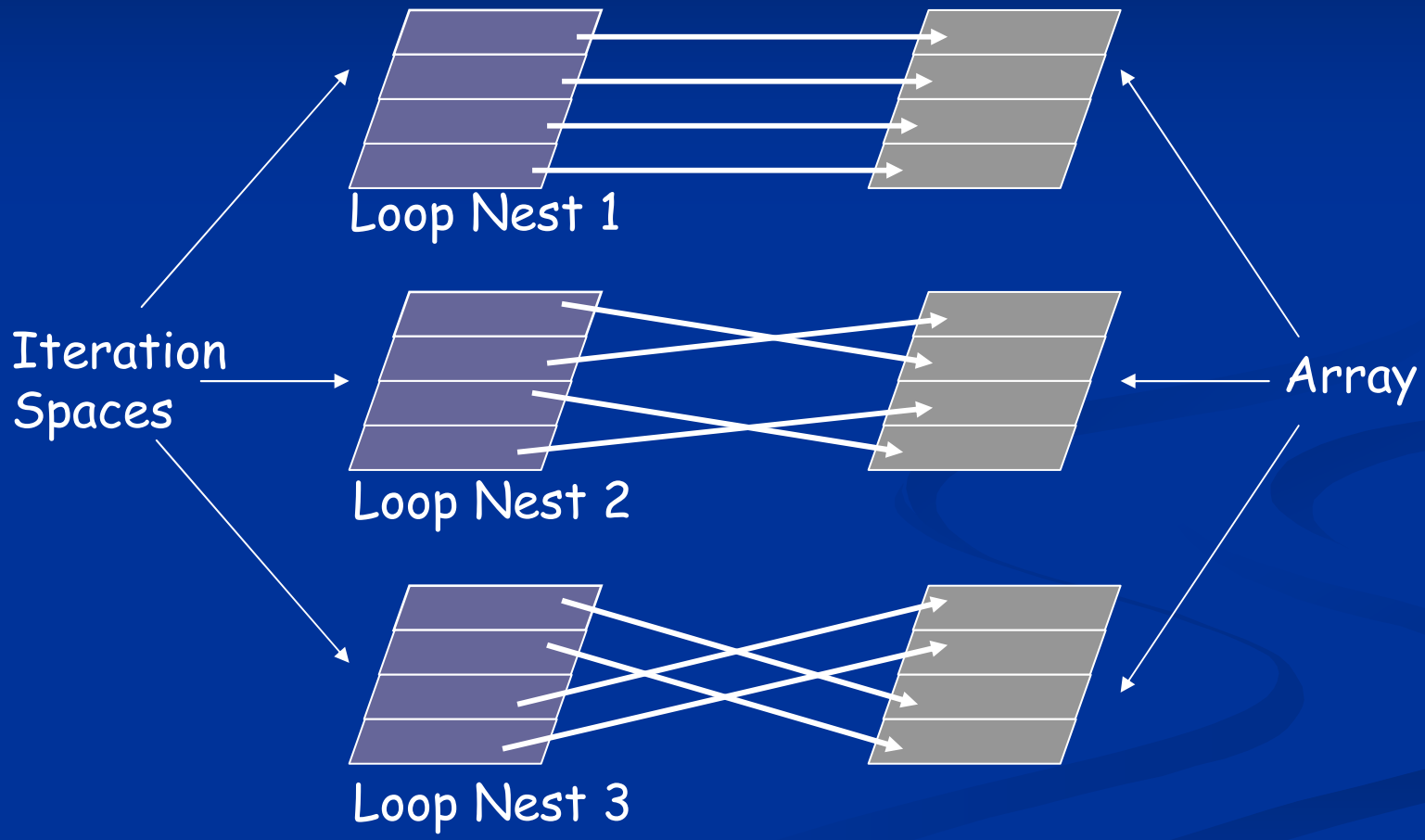


Loop Based Code Parallelization

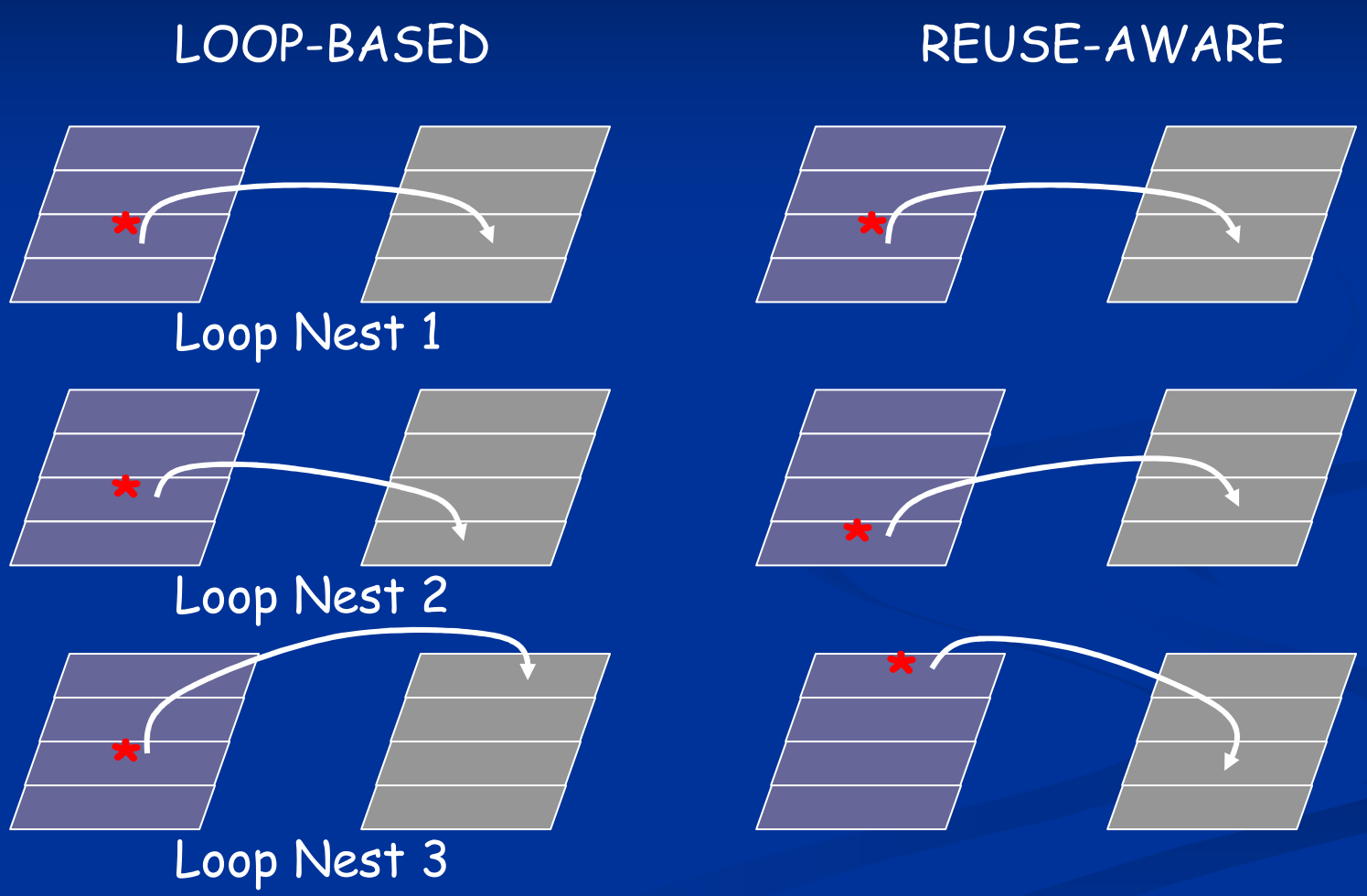


- Coarse grain parallelism: outermost loop
- Data dependence constraints
- Parallelize each loop nest individually
  - Drawback: not capturing data locality between different loop nests

Multi-Processor Execution - Motivating Example



Loop-Based vs. Reuse-Aware



Reuse-Aware Parallelization

- Main goal: improving disk locality for each processor – loop iterations accessing the same disk-resident array should be executed by the same processor
- Methodology: inter-loop-nest disk reuse aware workload assignment
- Constraint: inter-loop-iteration data dependences

Mathematical Model

- Assign loop iterations to each processor based on disk access patterns

  For processor s $(1 \leq s \leq p)$ and loop nest k $(1 \leq k \leq n)$.

  $Z_{s,j}$: data elements of array Z_j assigned to processor s.

  $Q_{s,j,k}$: the set of loop iterations from loop nest k accessing $Z_{s,j}$.

  $\Rightarrow$ Assign $Q_{s,j,k}$ to processor s.

Issues in Reuse-Aware Parallelization

1. Array element partitioning
   - Disk access patterns, disk reuse, and Parallelism of the program
2. Partial array access
   - A loop nest accesses a portion of the array
3. Multiple arrays
   - Multiple arrays accessed by the same processors share the same disk-resident array
   - Loop iteration mapping must consider all the arrays to improve disk reuse locality

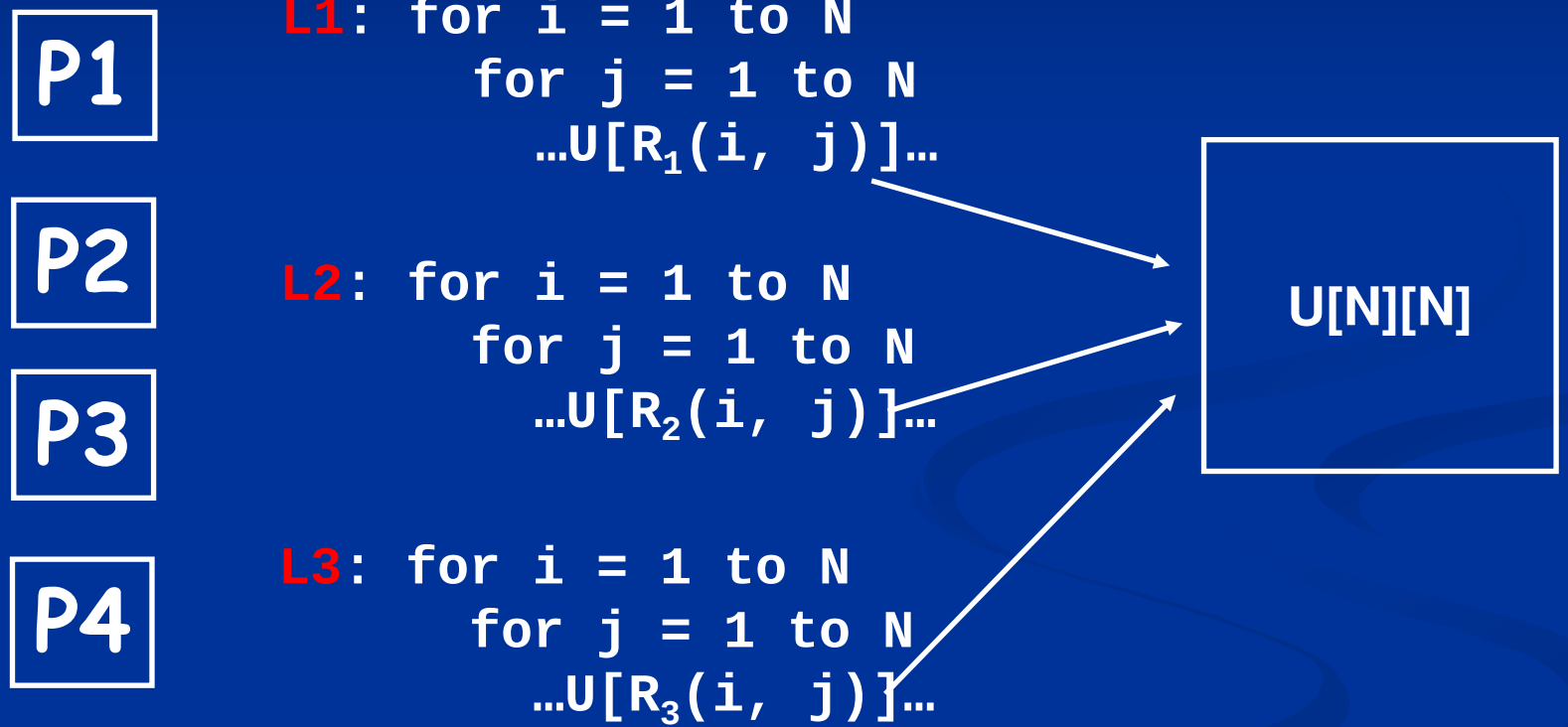
Issue1: Array Element Partitioning

- Use loop based parallelization to extract maximum parallelism
- Given iteration set I_s executed by processor s, and the array reference function set R_s, determine the accessed array region

$$\mathrm{D}_s = \{\vec{d} \mid \exists \vec{I} \in \mathcal{I}_s, \quad \exists R \in \mathcal{R}_s \text{ such that } R(\vec{I}) = \vec{d}\}.$$

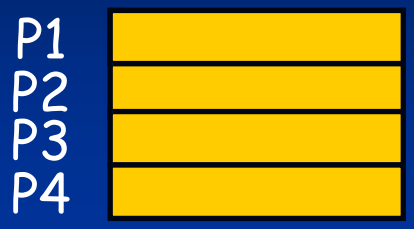
- D_s can be different for different loop nests
  - A unification step to obtain a globally acceptable array partitioning

A Unification Example



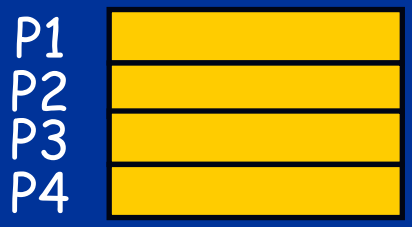
A Unification Example

L1:



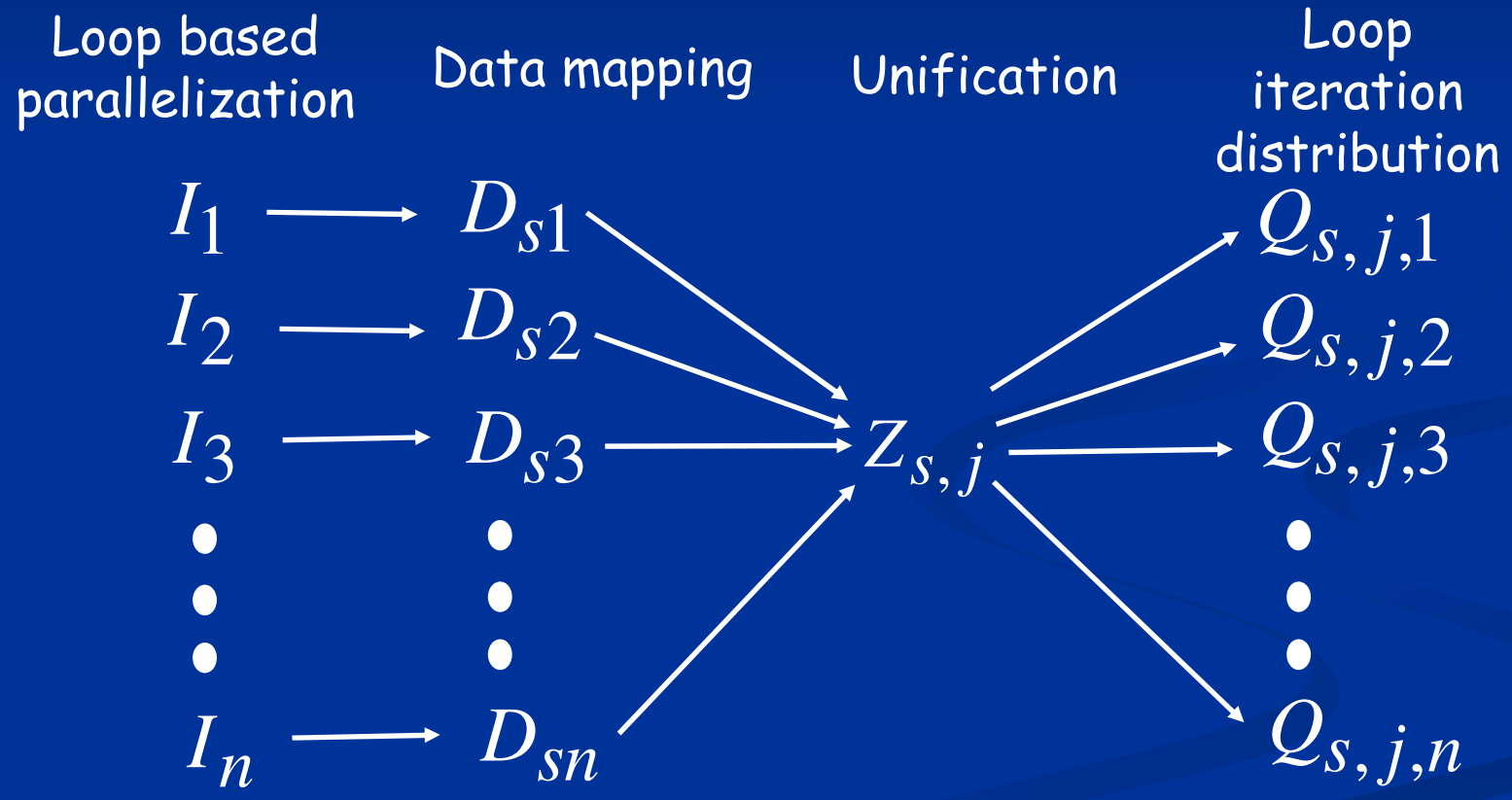
L2:

L3:



Row-block array partitioning is selected

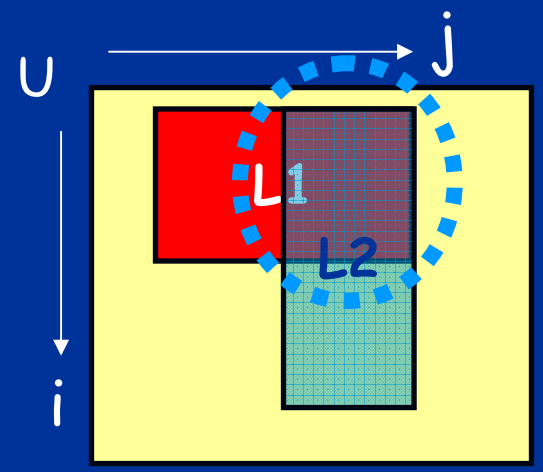
Data Mapping and Loop Iterations Assignment

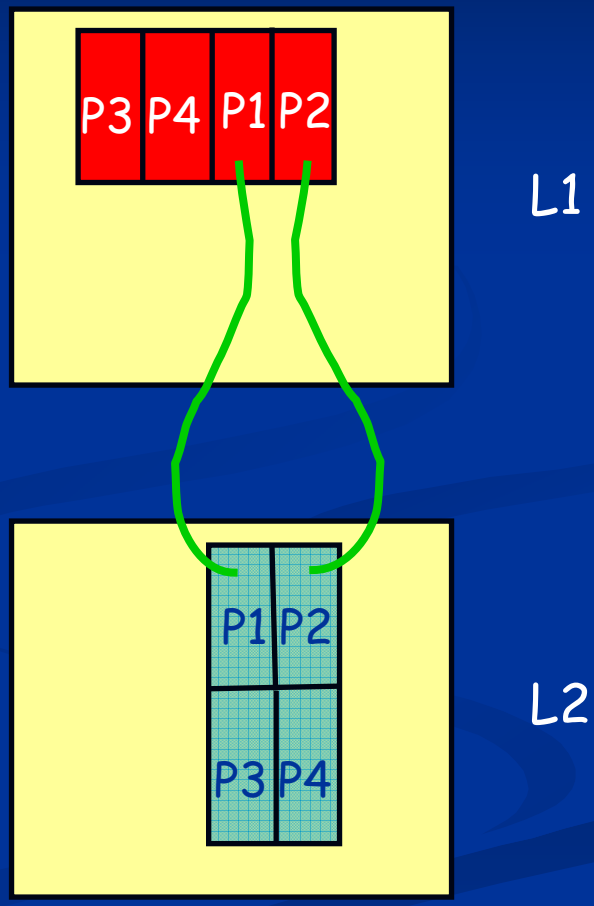


Issue2: Partial Array Access Pattern

```
L1: for i = 10 to 300
      for j = 100 to 900
        …U[i][j]…

L2: for i = 10 to 600
      for j = 500 to 900
        …U[i][j]…
```

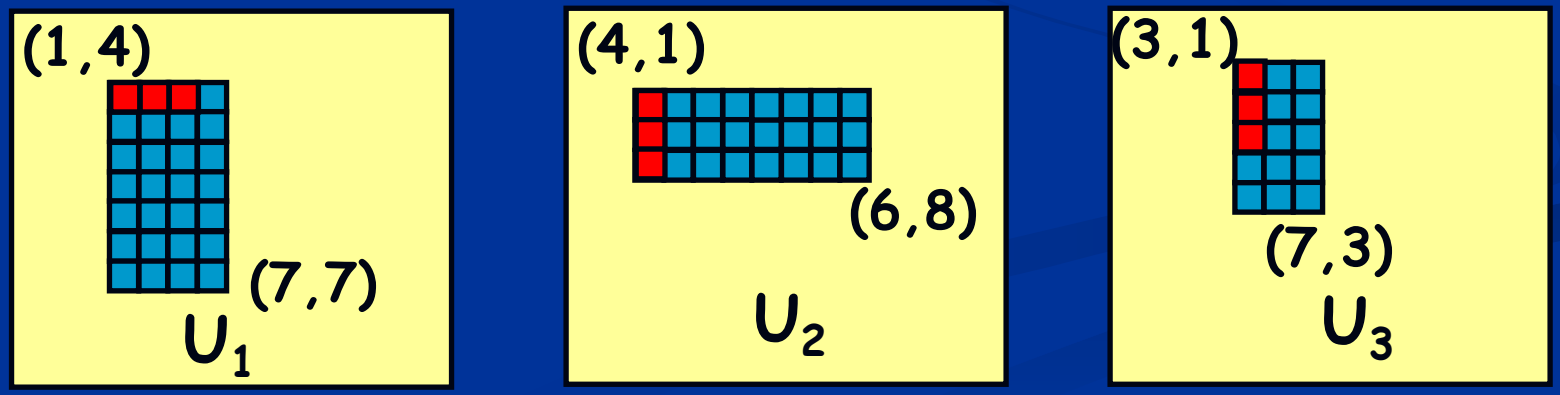




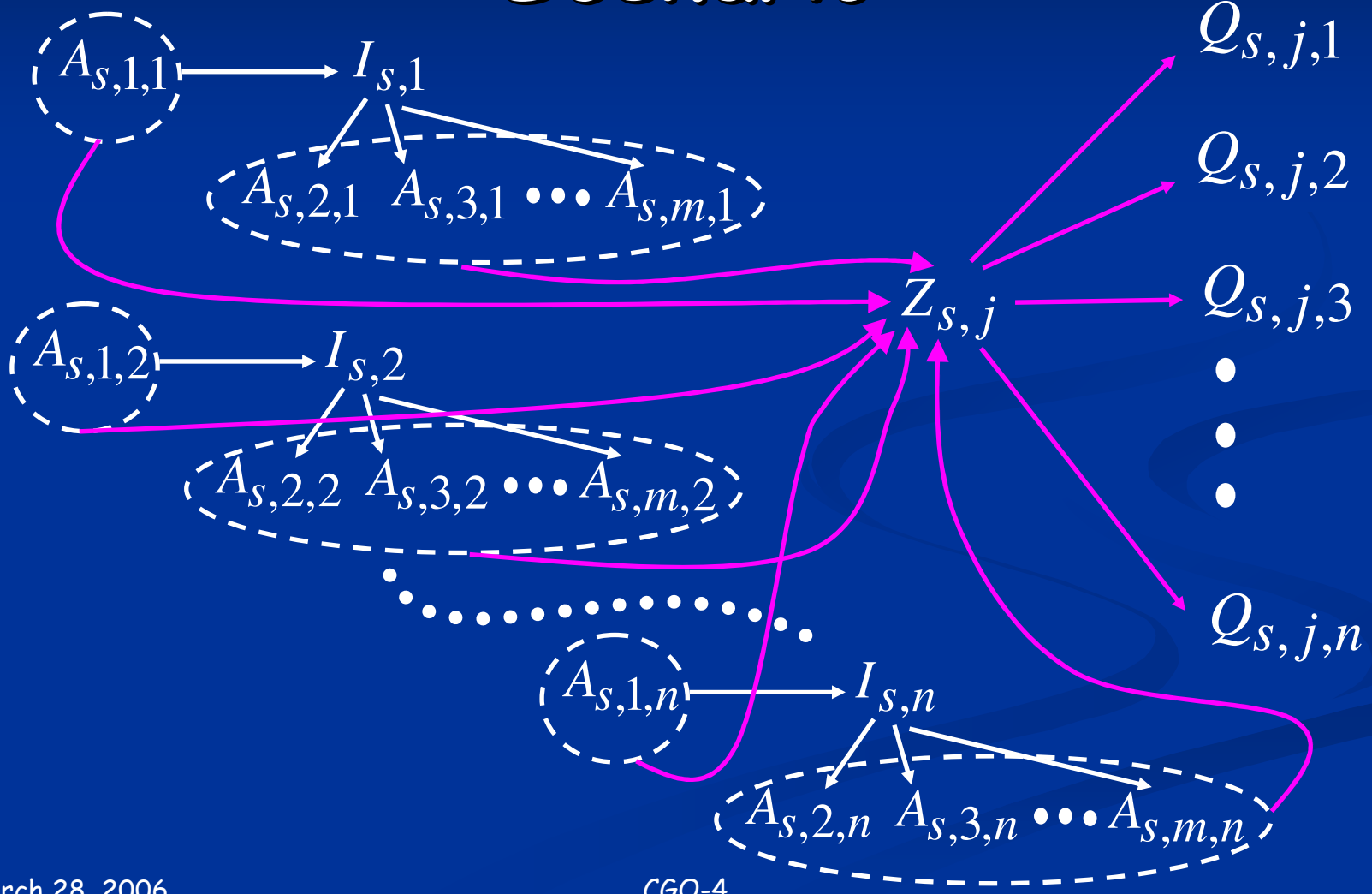
Issue3: Multiple arrays

- Two data items accessed by the same loop iteration are said to exhibit *affinity*
- *Affinity class*: a set of data elements that exhibit affinity

```
for i = 1..M-2
 for j = 4..N
 …U1[i][j]…U2[j][i]…U3[i+2][j-3]
```



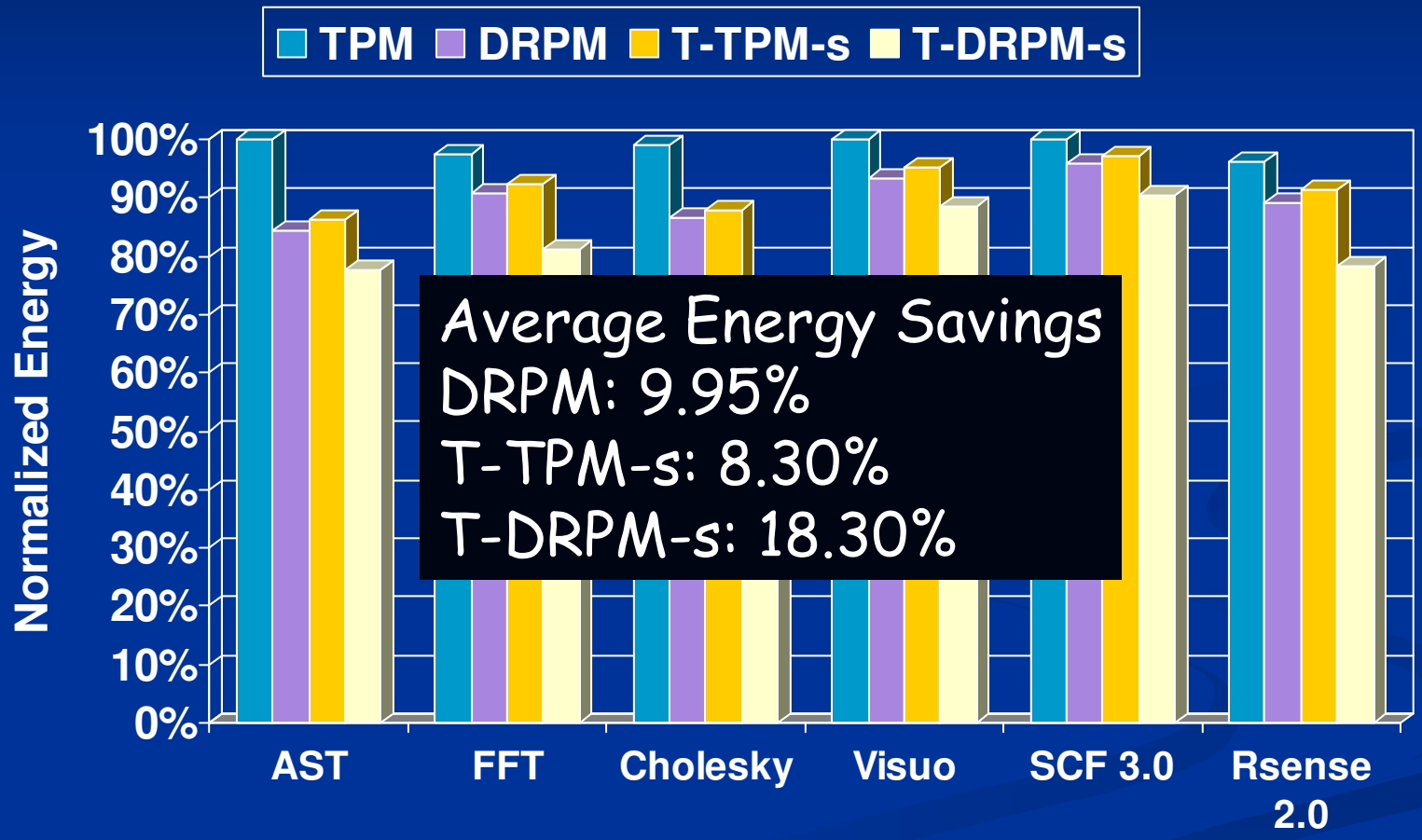
Determining the Workload of Processor under Multiple Arrays Scenario



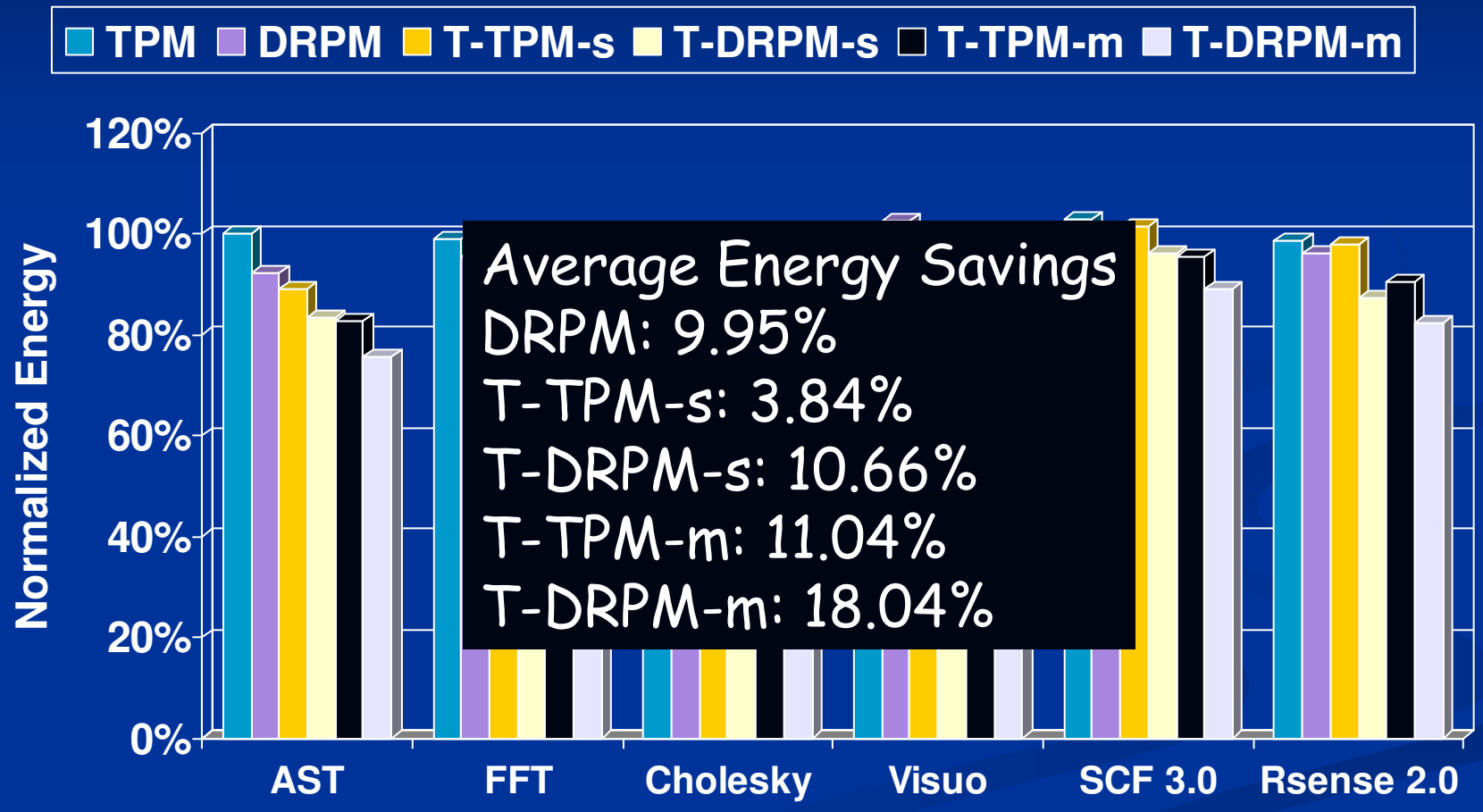
Experimental Platform

- Trace generator is used to collect I/O traces
- Omega Library is used to generate codes after loop iteration assignment
- Disk energy simulator
  - Based on IBM36Z15 disk power model
  - TPM and DRPM energy model
- Used six I/O intensive codes

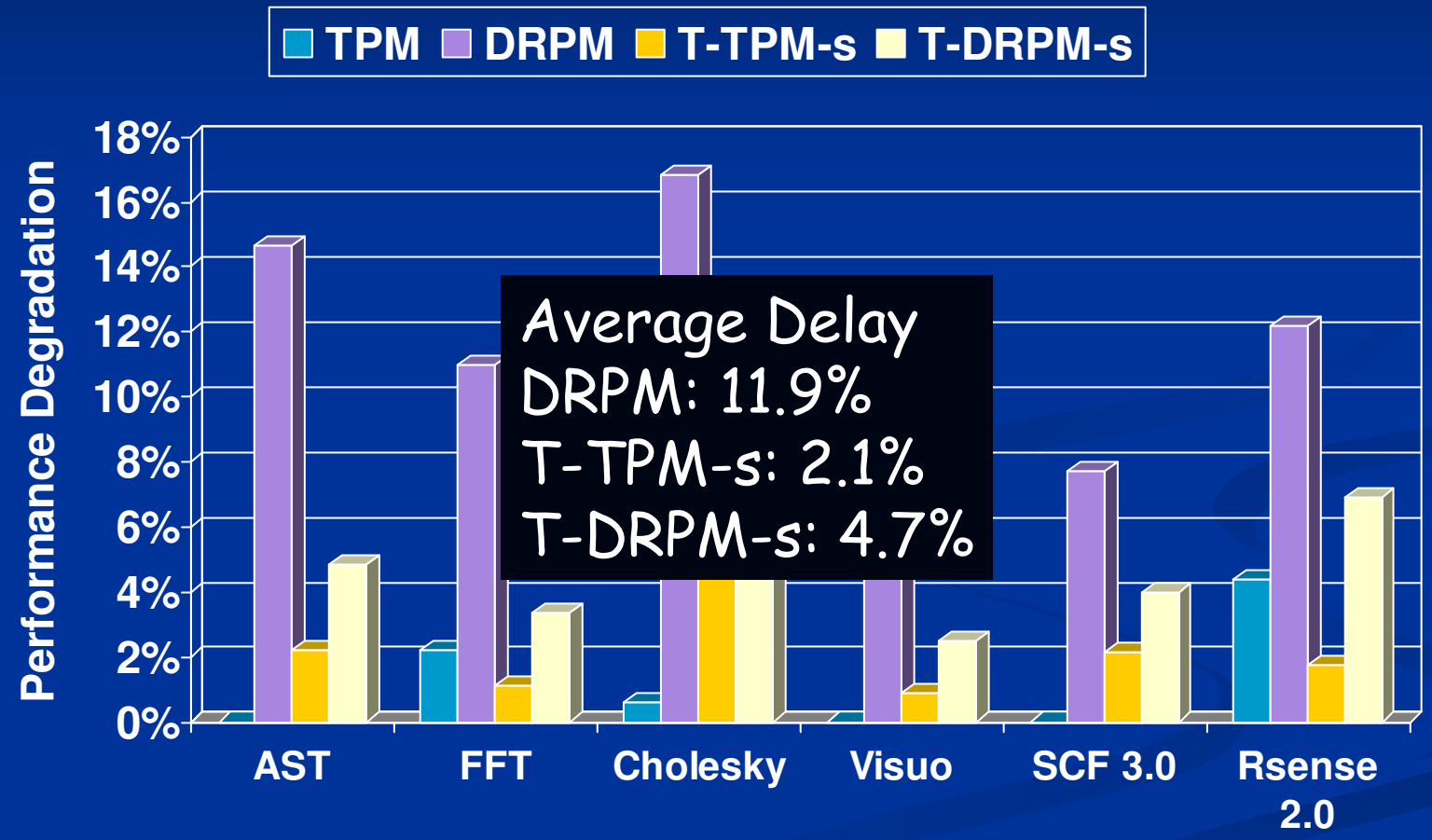
Energy Consumption – single processor execution



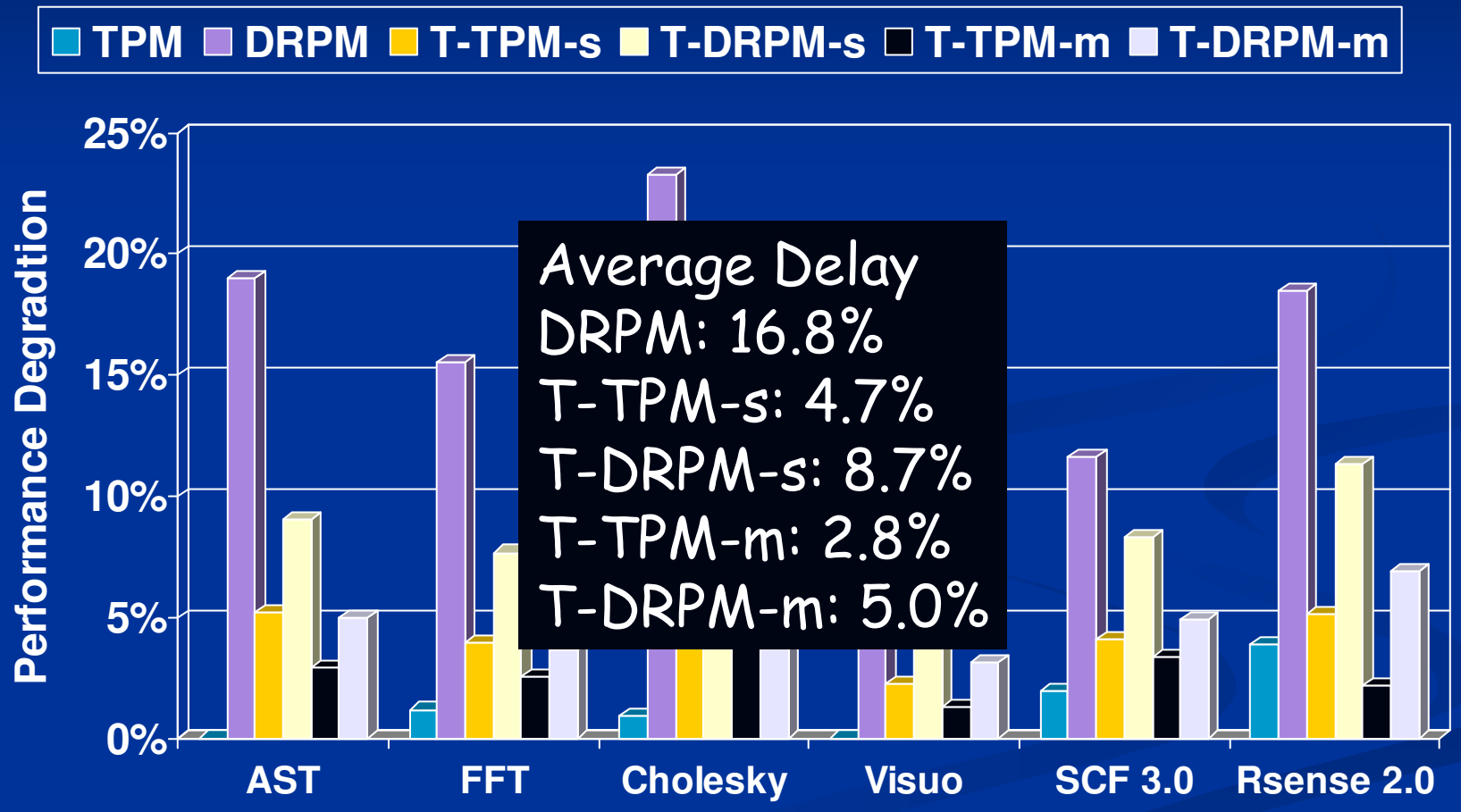
Energy Consumption – multi-processor execution



Performance Degradation – single processor execution



Performance Degradation - multi-processor execution



Conclusion

- Disk subsystems is one of major contributor to overall power consumption of high-end server systems
- We proposed a compiler-guided approach to increase the effectiveness of the previously-proposed disk power management schemes
- Simulation with both single and multi-processor execution shows that our approach achieves more energy savings than hardware schemes

Thank you!

sson@cse.psu.edu