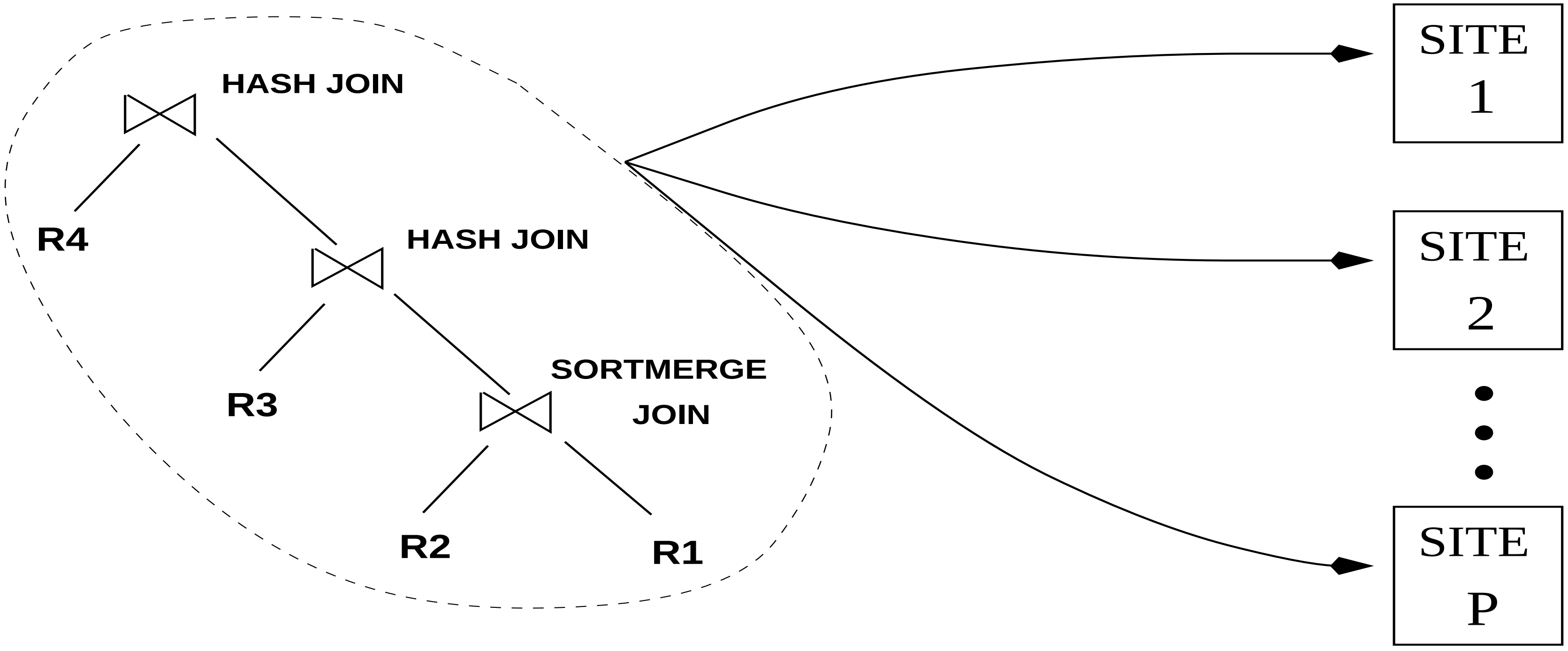
Query Scheduling and Optimization in Parallel and Multimedia Databases

Minos N. Garofalakis
University of Wisconsin – Madison

Introduction

- Judicious resource scheduling $\Rightarrow$ Fast response times & Effective utilization
- New application domains – Scheduling is an essential component of DB query processing and optimization
- Focus: *Parallel and Multimedia Database Systems*

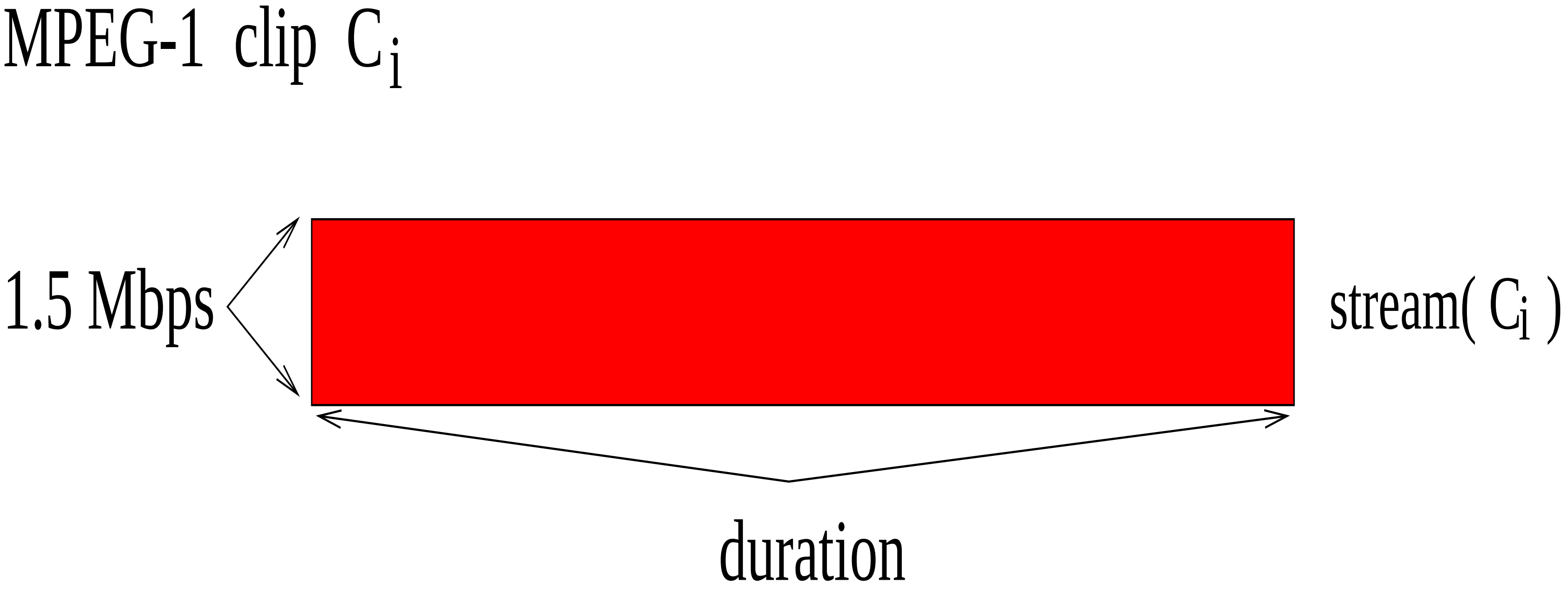
Parallel Database Systems



- Exploit parallelism and available resources to reduce response times and increase system throughput
- Effective mappings of Query Execution Plans on parallel architectures

Multimedia Database Systems

- *Continuous Media*: Real-time resource requirements for delivery



- Objective: Maximize number of concurrent clients given bounded resources

Problem Space and Contributions – Summer'96

	Parallel DBS	**Multimedia DBS**
Multi-dimensionality	$\surd$ – TS only	
Malleability	$\surd$ – TS only	
On-line scheduling		
Time-varying resource demands		
Admission control		
Periodic service		
Impact on Query Optimization		

Problem Space and Contributions – Summer'98

	Parallel DB Systems	Multimedia DB Systems
Multi-dimensionality	√ – TS only √ – TS & SS	√
Malleability	√ – TS only!	
On-line scheduling	√	√
Time-varying resource demands		√
Admission control		√
Periodic service		√
Impact on Query Optimization	√	

References

	Par. DBS	**MM DBS**
Multi-dimensionality	**1** – TS only **2** – TS & SS	**5**
Malleability	**1** – TS only!	
On-line scheduling	**2**	**4, 5**
Time-varying resource demands		**5**
Admission control		**4**
Periodic service		**3, 6**
Impact on Query Optimization	**2**	

1: Garofalakis and Ioannidis [SIGMOD'96]

2: Garofalakis and Ioannidis [VLDB'97]

3: Garofalakis, Özden, and Silberschatz [VLDB'97]

4: Garofalakis, Ioannidis, Özden, and Silberschatz [PODS'98]

5: Garofalakis, Ioannidis, and Özden [VLDB'98]

6: Garofalakis, Özden, and Silberschatz [RIDE'98]

Focus of this Talk

	Parallel DB Systems	**Multimedia DB Systems**
Multi-dimensionality	$\surd$ – TS only $\Rightarrow$ $\surd$ – TS & SS	$\surd$
Malleability	$\surd$ – TS only!	
On-line scheduling	$\surd$	$\surd$
Time-varying resource demands		$\surd$
Admission control		$\surd$
Periodic service		$\Rightarrow$ $\surd$
Impact on Query Optimization	$\Rightarrow$ $\surd$	

Part I

- ⇒ Multi-dimensional Parallel Query Scheduling & Optimization [SIGMOD'96, VLDB'97]
- Periodic Resource Scheduling for Continuous Media DBs [VLDB'97, RIDE'98]

Parallel Query Scheduling: Outline

- Introduction and Motivation
- Problem Formulation and Query Execution Model
- Heuristic Query Scheduling Algorithms
- Experimental Results
- Implications for Parallel Query Optimization

Introduction and Motivation

The Problem

- Architecture: Shared-Nothing or Hierarchical
- *Static* parallelization of set of complex query plans, based on size & cost estimates
- Objective: minimize query response time

Why?

- Most scalable architectural paradigms
- Part of cost-based query optimization (compile-time)
- Effective parallel query execution

Introduction and Motivation (cont.)

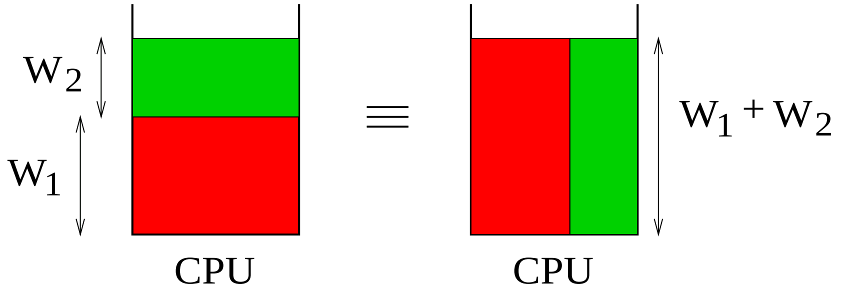
Sources of Complexity

- Various forms of intra/inter-operator parallelism
- Cost Model must account for communication overheads
- "Multi-dimensional" resource needs of DB operators (opportunities for resource sharing)
- Resources: *Time-Shared* (e.g, CPU(s)) or *Space-Shared* (e.g., memory)

- *NP-hard*, even for simple "one-dimensional" cases

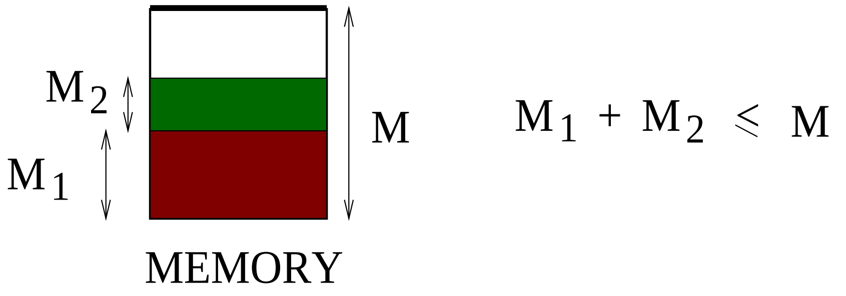
Goals Incorporate TS & SS dimensions; Effective scheduling heuristics; Implications for Query Optimization

Time-Shared vs. Space-Shared



Time-Shared (TS) Resources

- Sliced between operators at negligible overhead
- Effective time can be stretched over execution interval

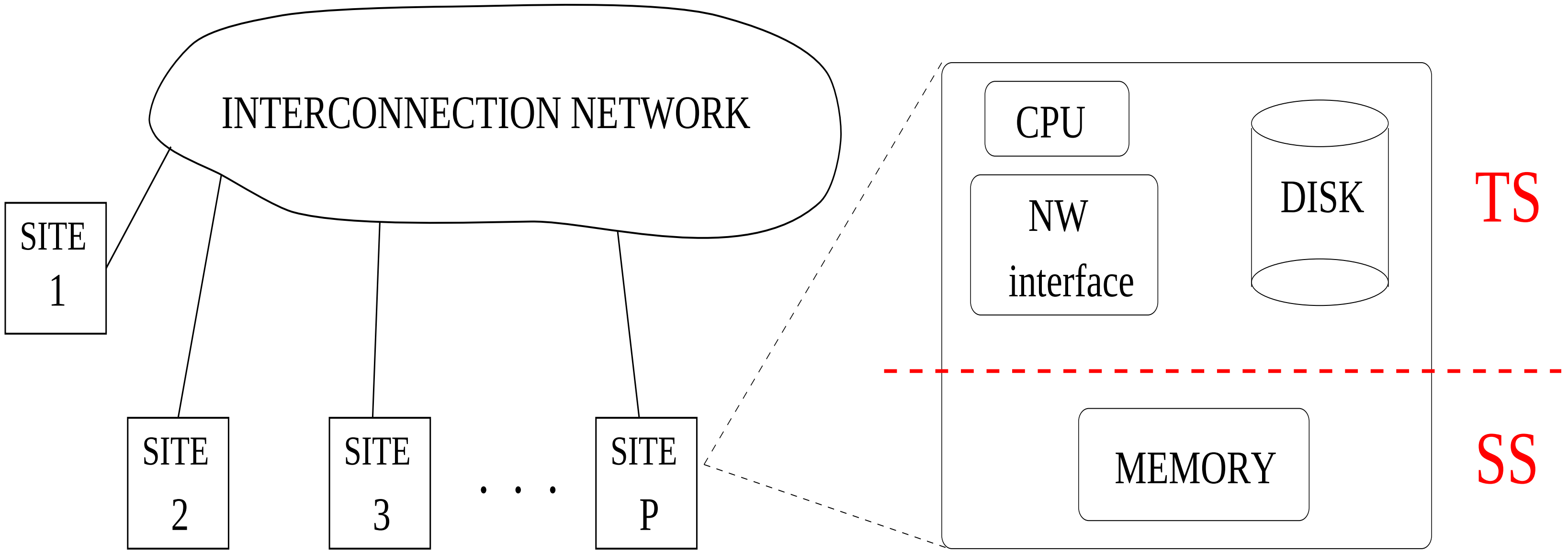


Space-Shared (SS)Resources

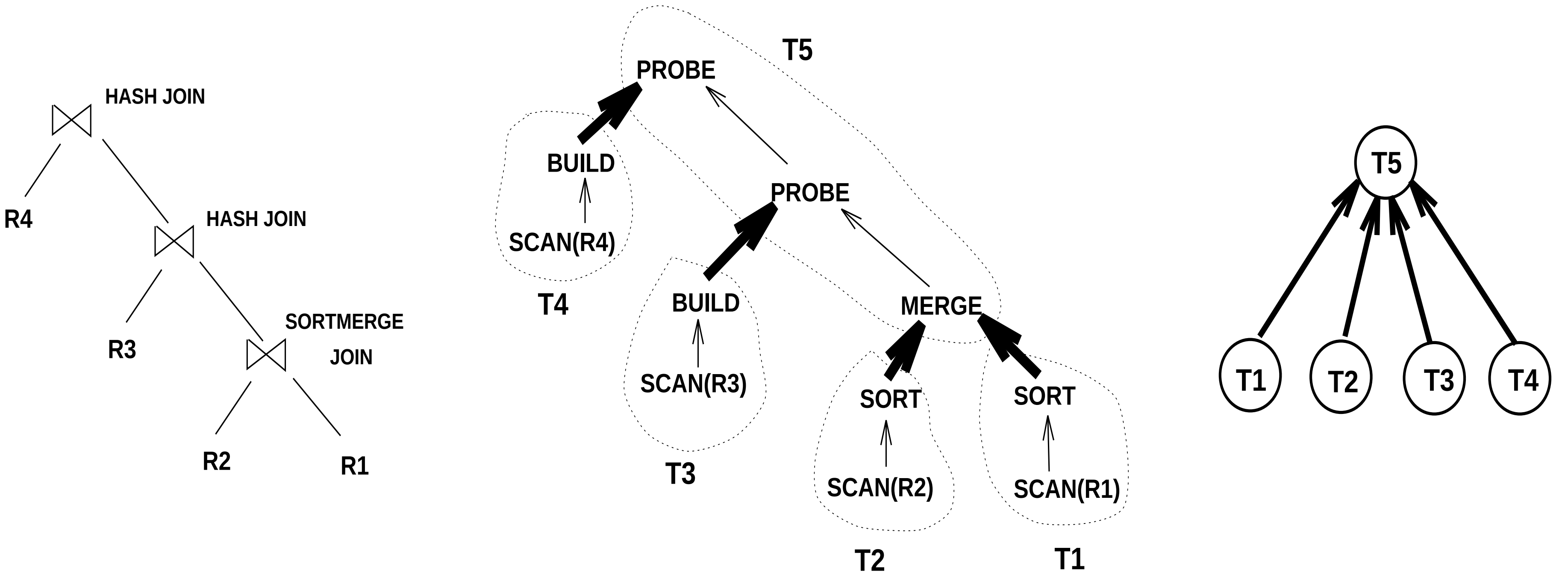
- Slicing implies prohibitive overhead
- Limited capacity that cannot be exceeded by executing operators

System Model

- Identical sites consisting of d TS and s SS resources

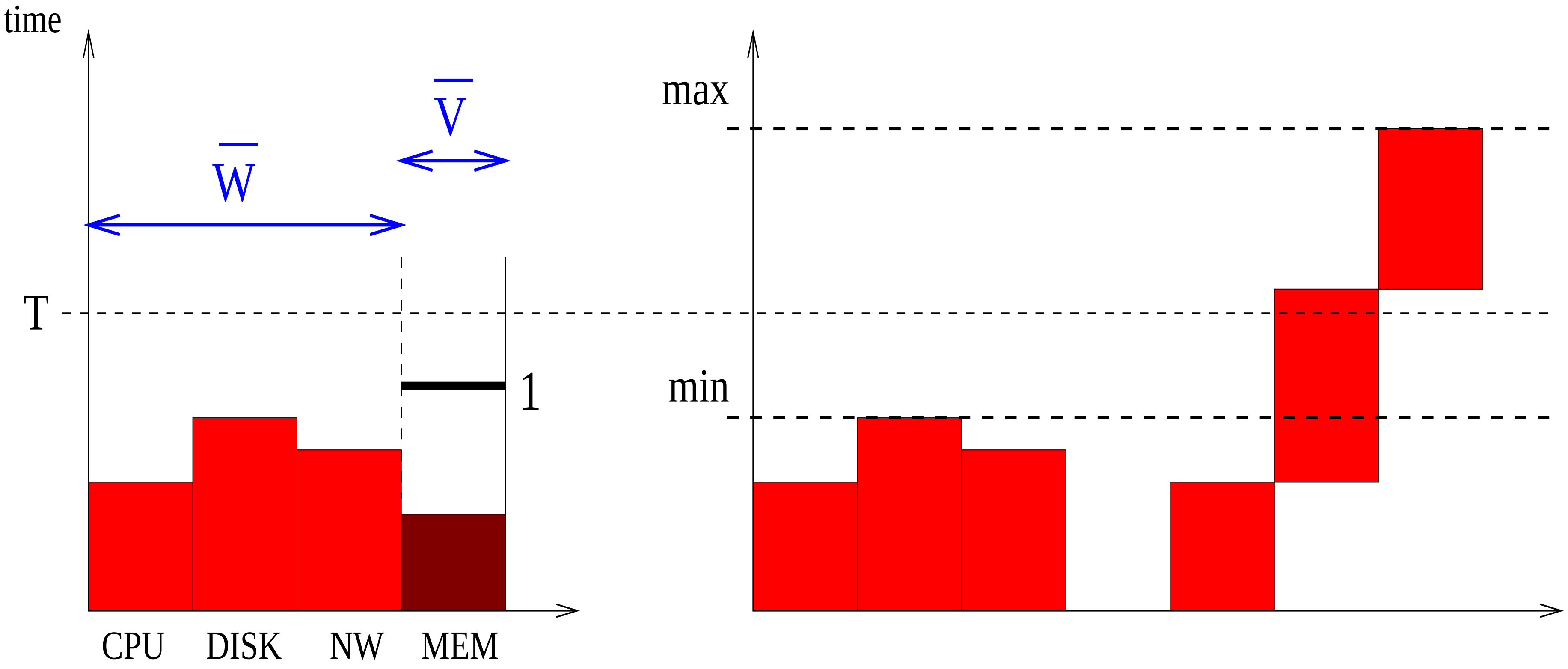


Query Plan Representation and Forms of Parallelism



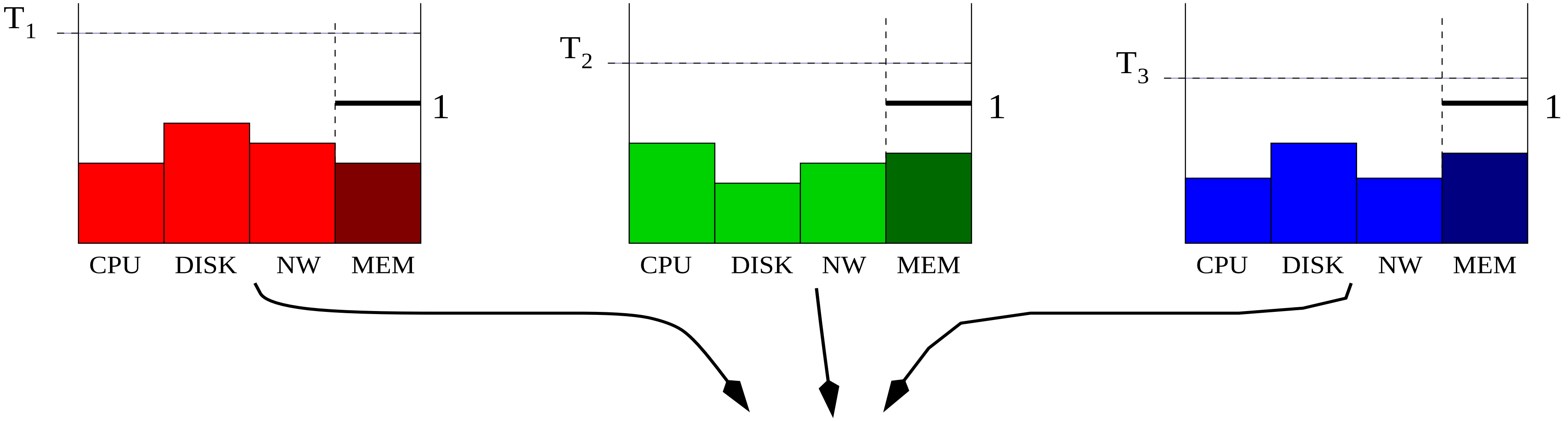
JOIN TREE   OPERATOR TREE   QUERY TASK TREE

Operator Cost Model

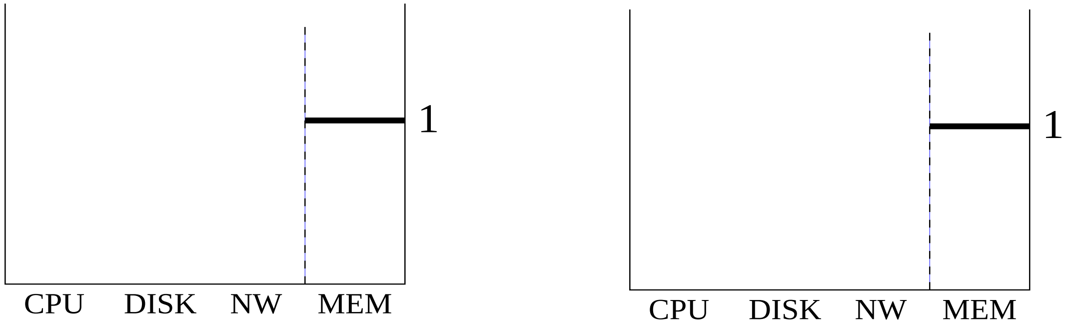


- Estimate *Work Vector* ($\overline{W}$) and *Demand Vector* ($\overline{V}$) components using conventional cost models and DB statistics
- Stand-alone execution time $T = \mathrm{f}(\ \overline{W}$, overlap), where $\overline{W}$ depends on $\overline{V}$

A Lower Bound on Response Time

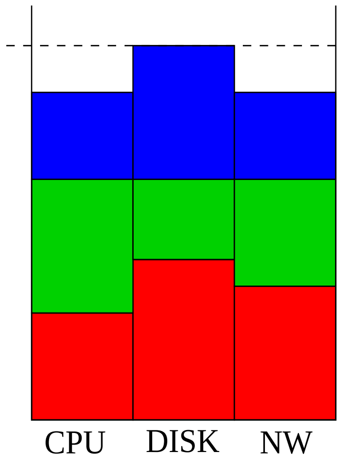


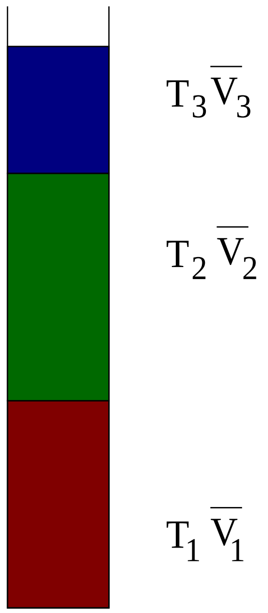
2 sites :



A Lower Bound on Response Time

- max { T_1 , T_2 , T_3 }: ''critical path'' of parallel execution





- $\dfrac{\text{max component} \left(\overline{W_1} + \overline{W_2} + \overline{W_3} \right)}{2}$

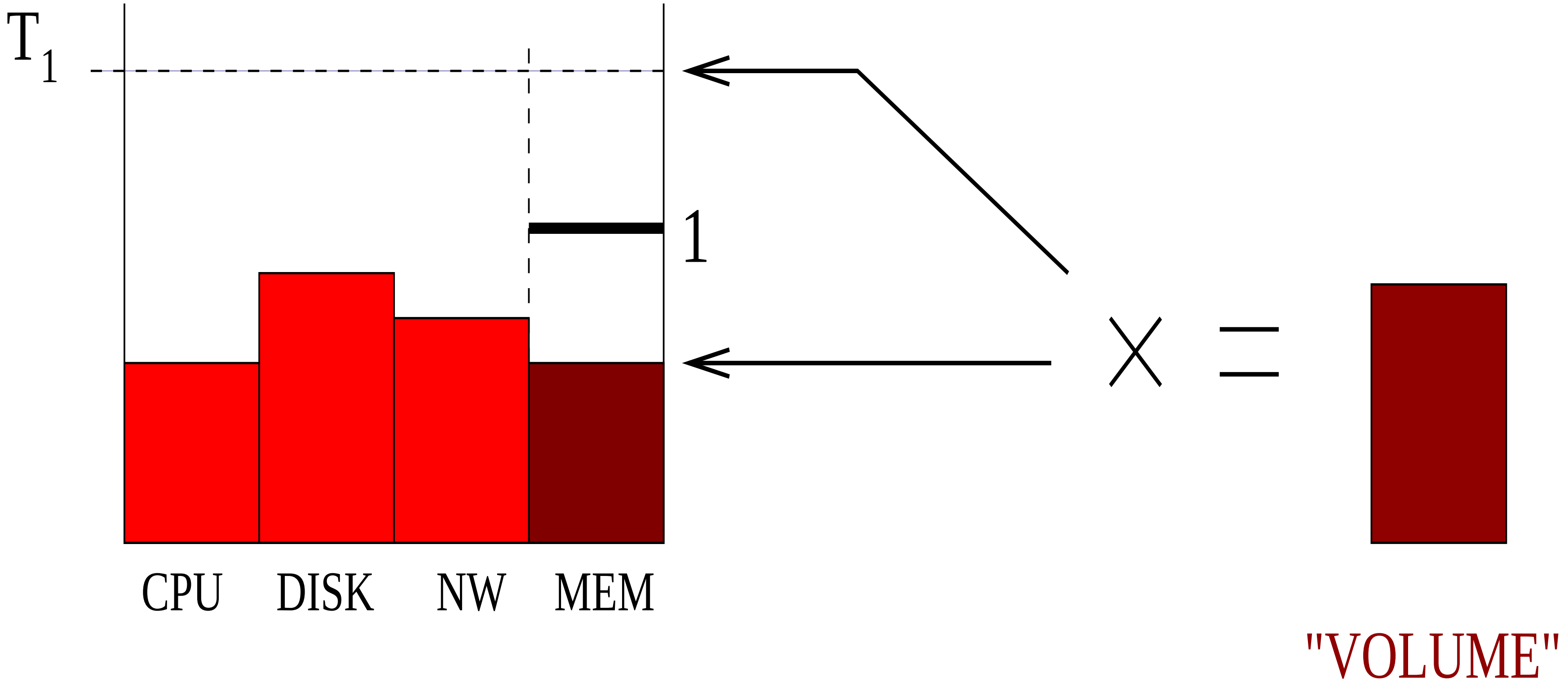
  ''average work'' per site (TS resources)

- $\dfrac{\text{max component} \left(T_1 \overline{V_1} + T_2 \overline{V_2} + T_3 \overline{V_3} \right)}{2}$

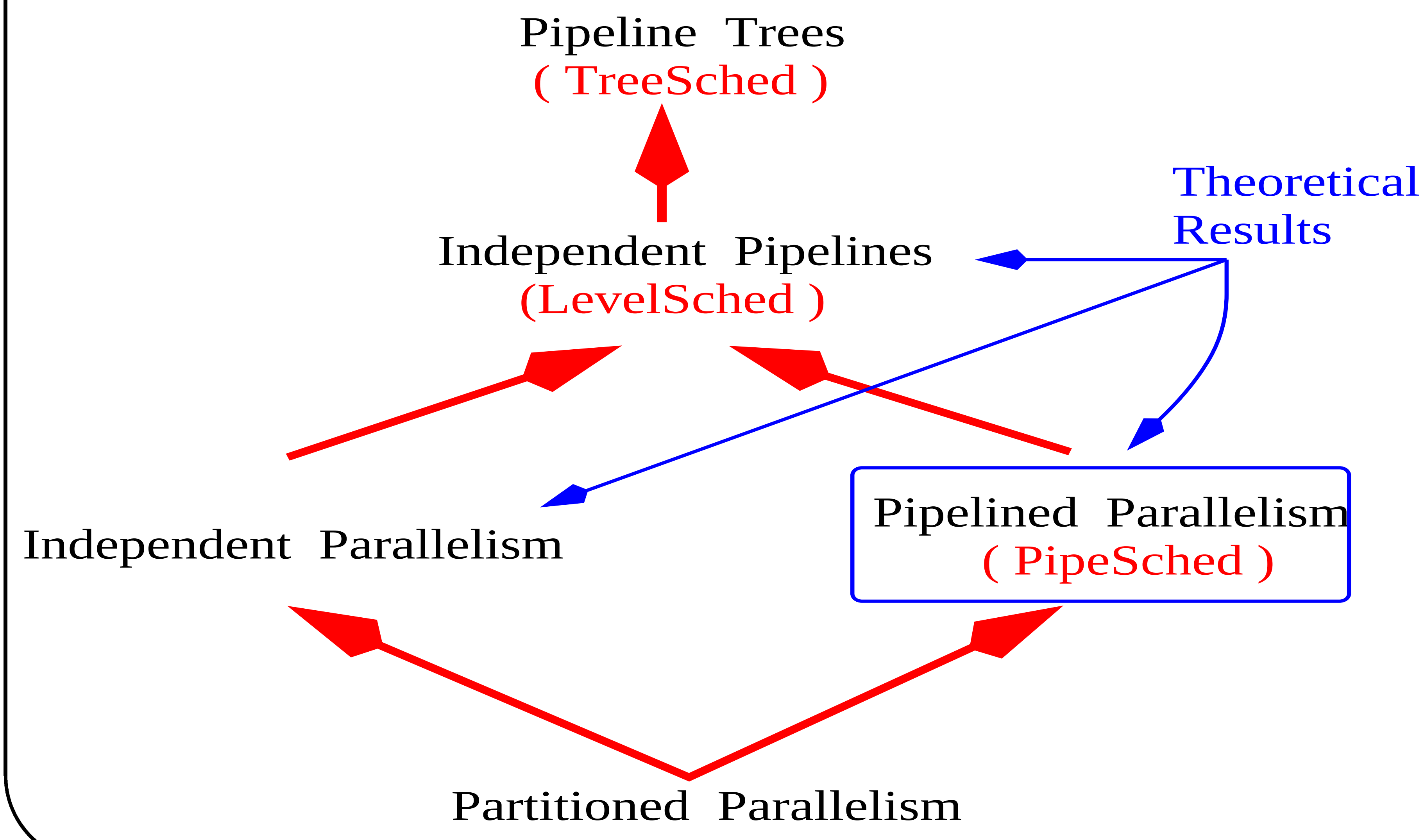
  ''average volume'' per site (SS resources)

LB = max { critical path , average work , average volume }

Operator Volume

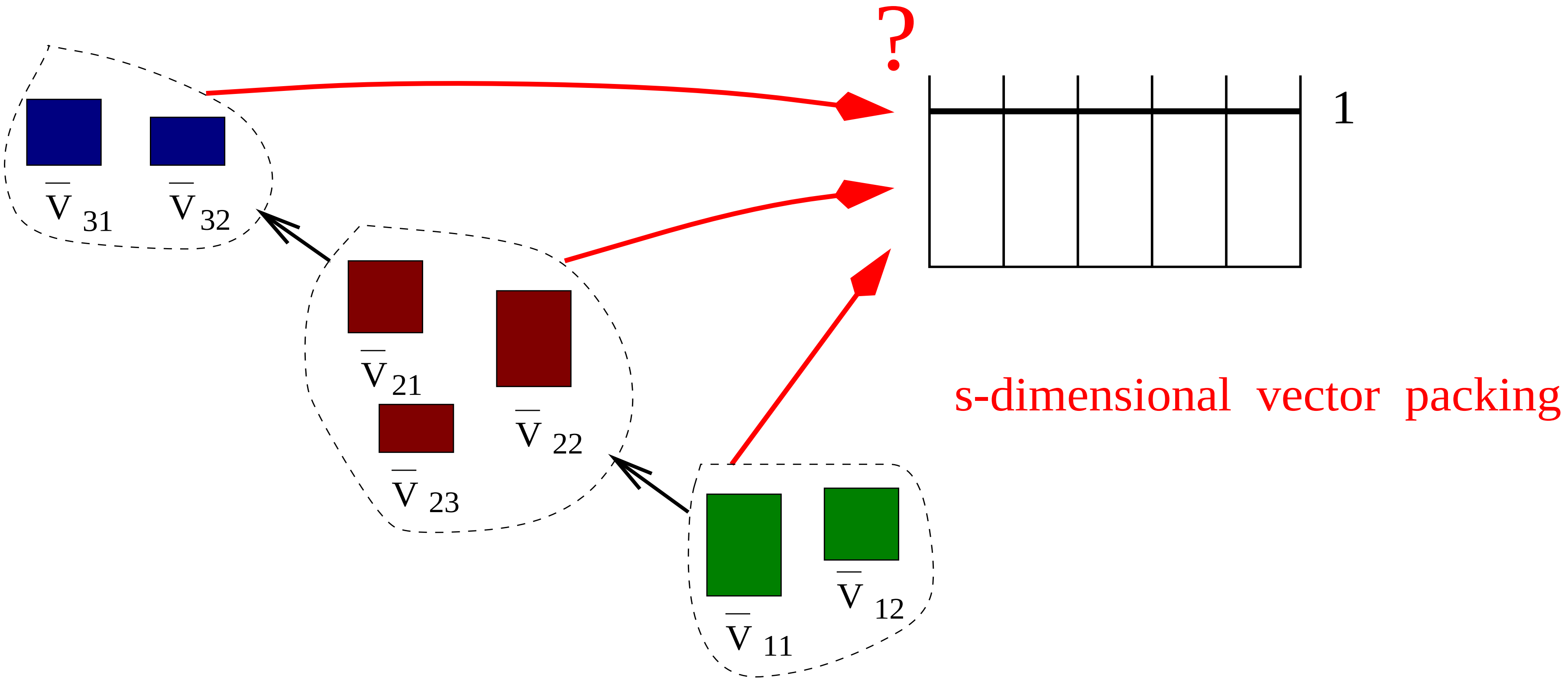


Scheduling Algorithms



Pipelined Parallelism

- Co-scheduling requirement: NP-hard decision problem!



- *λ-granular clone*: all components of $\overline{V}$ are $\leq \lambda$ (≤ 1)
- Simple sufficient condition for schedulability of pipelines of λ-granular clones

Pipelined Parallelism (cont.)

Algorithm PipeSched: fast List-Scheduling Heuristic (à la Graham) for a schedulable pipeline

1. Sort all clones in decreasing order of

$$\textit{work density} \;=\; \max\{\overline{W}\} \;/\; \max\{\overline{V}\}$$

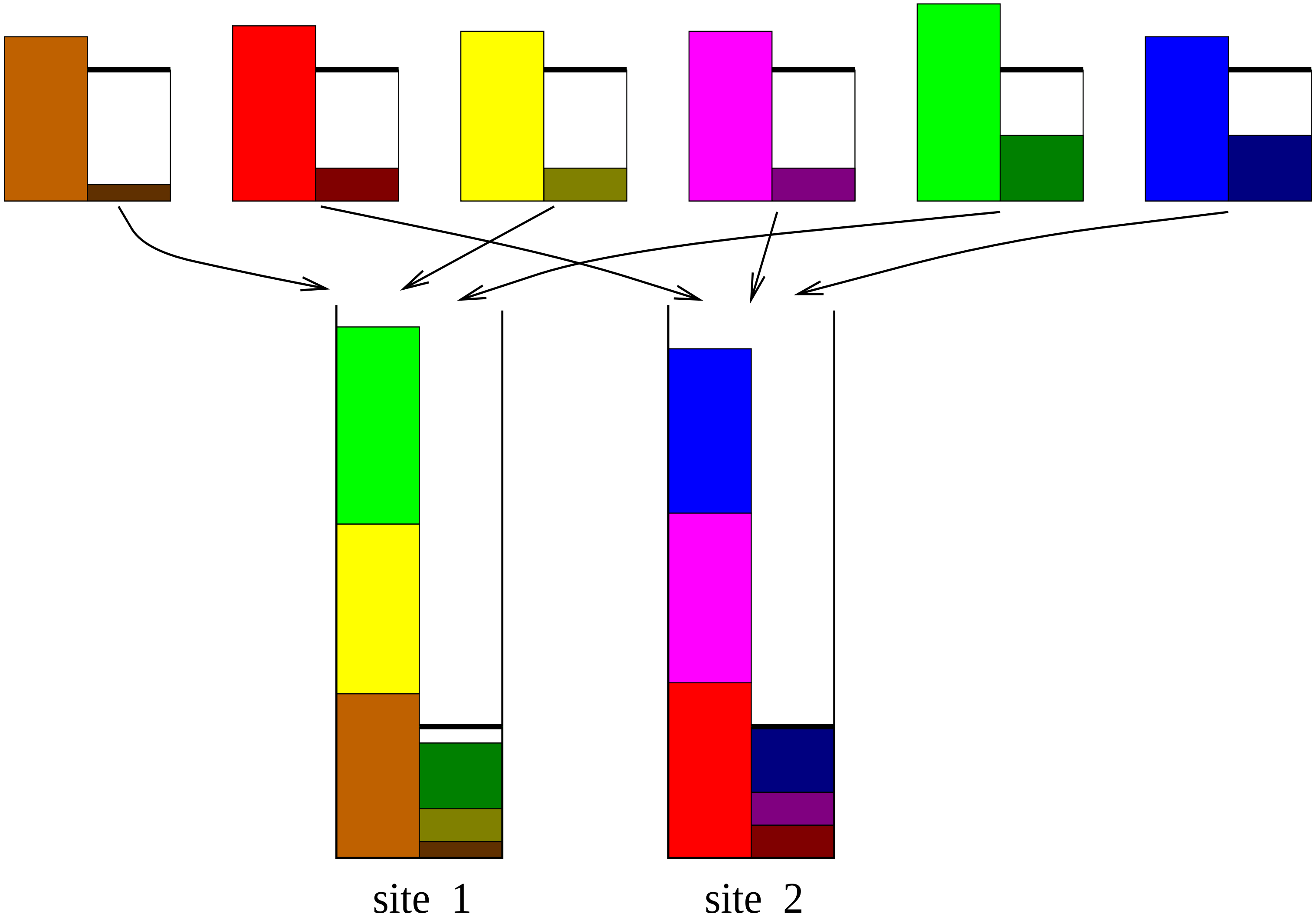
2. In that order, place each clone in the eligible site with the least amount of work

Theorem: Given a schedulable λ-granular pipeline, PipeSched produces a schedule that is within $d\left(1 + \frac{s}{1-\lambda}\right) + 1$ of optimal

A Simple Example ($d = 1$, $s = 1$)

1. Sort

2. Place



Independent Pipelines

Algorithm LevelSched:

1. Group the pipelines into maximal schedulable subsets

2. Schedule each subset in a separate level ("shelf") using PipeSched

Theorem: LevelSched is within $O\left(\frac{s(d^2+s)}{1-\lambda}\right)$ of optimal

Cost Model Experiments

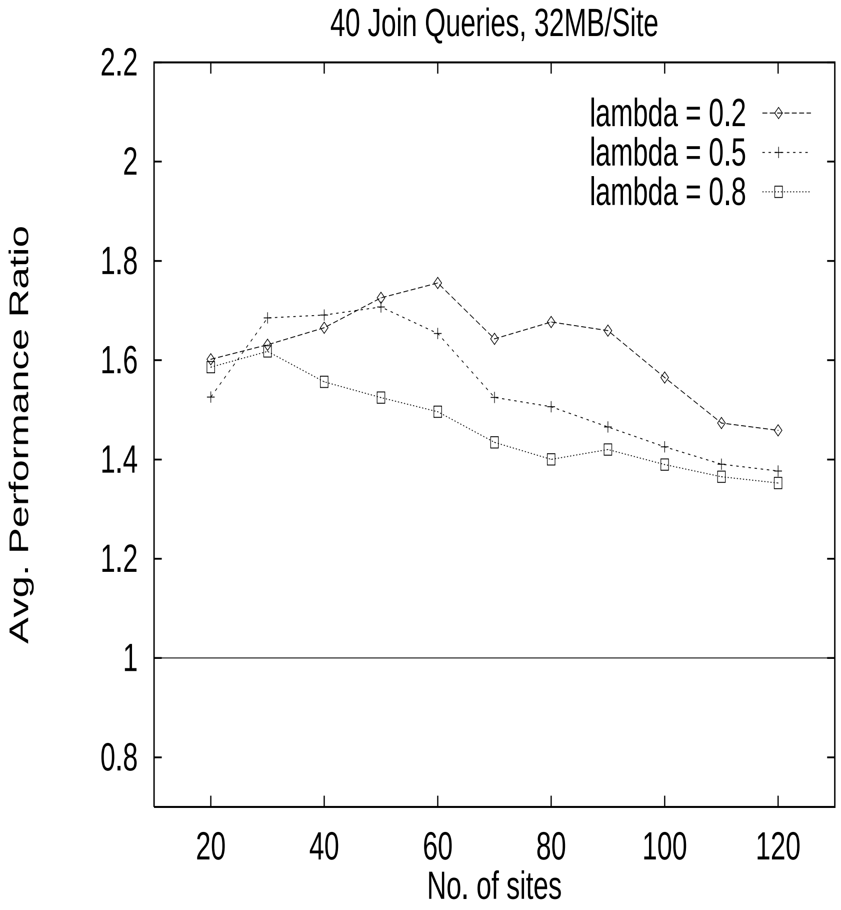
- TreeSched vs. lower bound on optimal response time (LB)
- *Average* effectiveness based on *cost model*

Experimental Assumptions

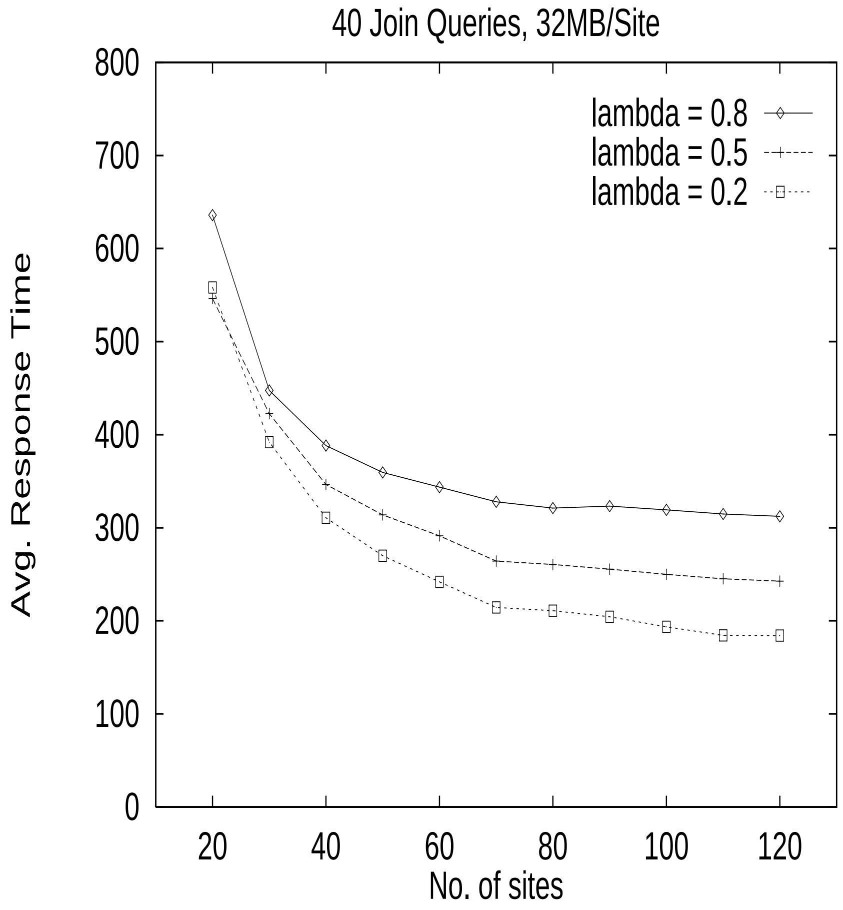
- Only simple hash-joins (one-phase pipelines)
- No execution skew (except for startup cost)
- Uniform resource overlap (single parameter)

Cost Model Experiments (cont.)

- TreeSched / LB

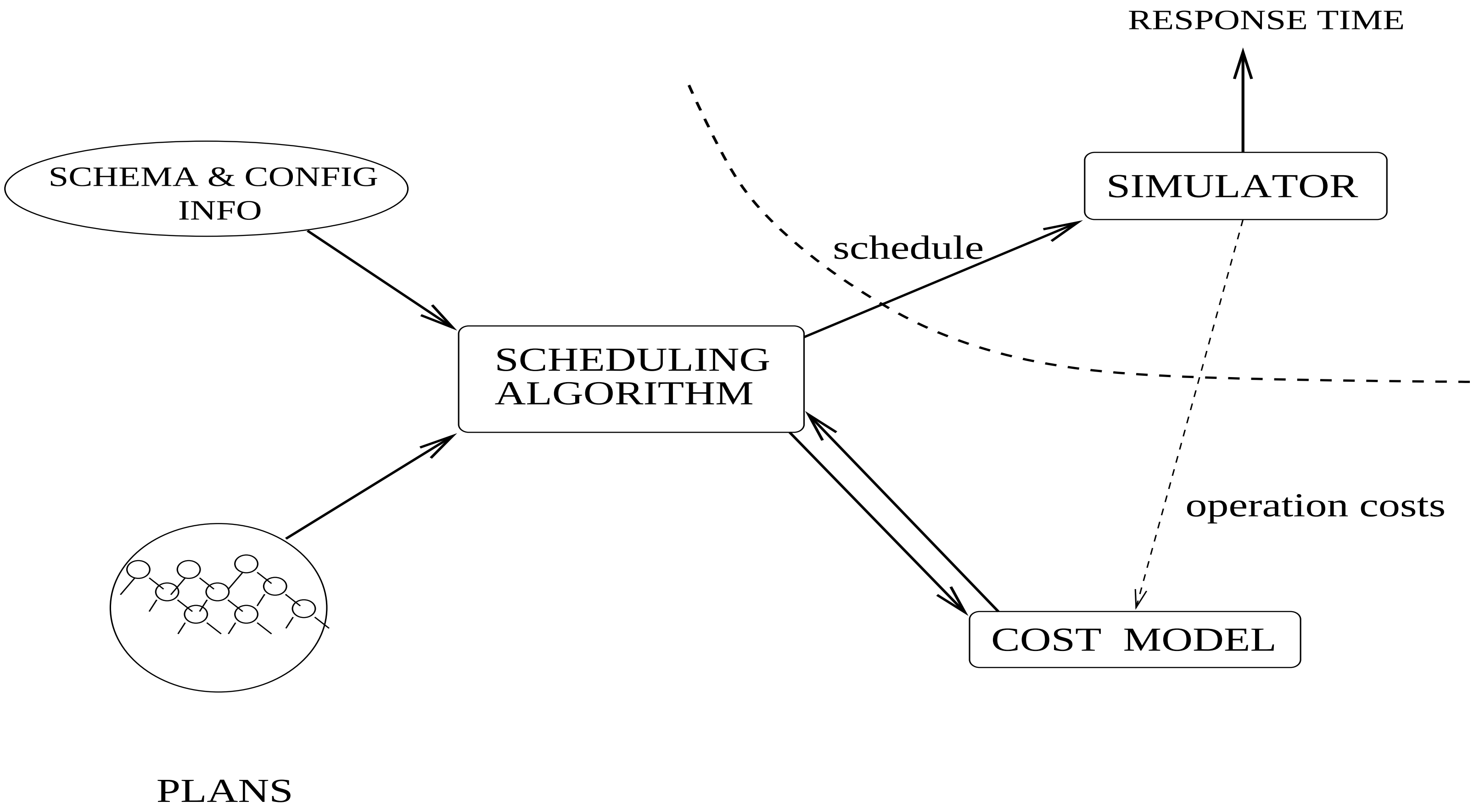


- TreeSched



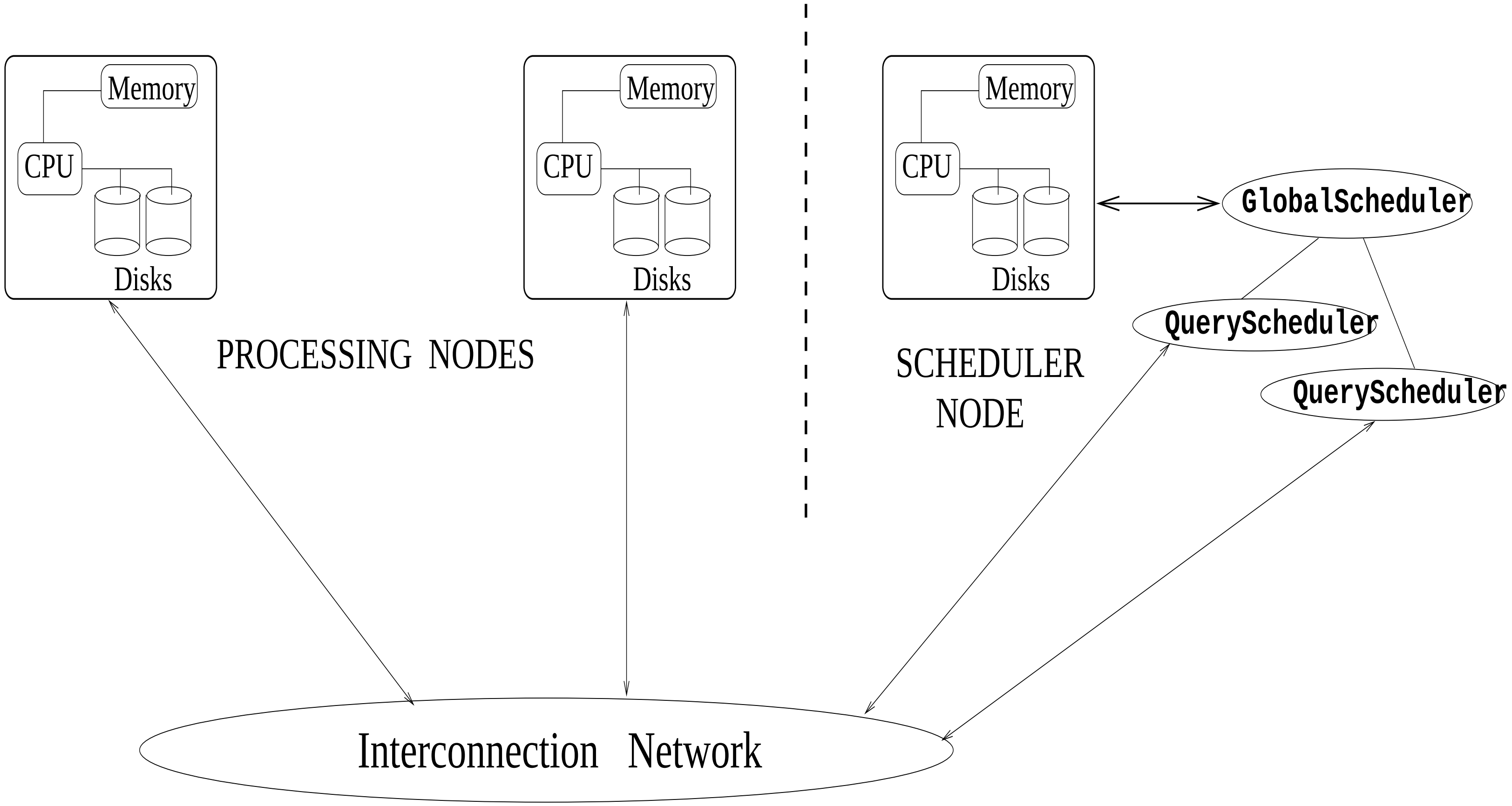
Simulation Experiments

- LevelSched on top of the ZetaSim detailed simulation model



Simulation Experiments (cont.)

- ZetaSim overview



Simulation Experiments: Testbed

- LevelSched vs. (default) Zeta ("schedule join where build relation lives")
- Workload: Collection of right-deep join plans. Total no. of joins 8, 16, 32, and 64, spread across various combinations of query sizes (e.g., 32 = 2 X 8-join + 4 X 4-join)
- Metric: elapsed simulation time to schedule completion

Simulation Experiments: Testbed (cont.)

- Various data placement schemes:

  1. **Declust** (**NoDeclust**): full (no) relation declustering

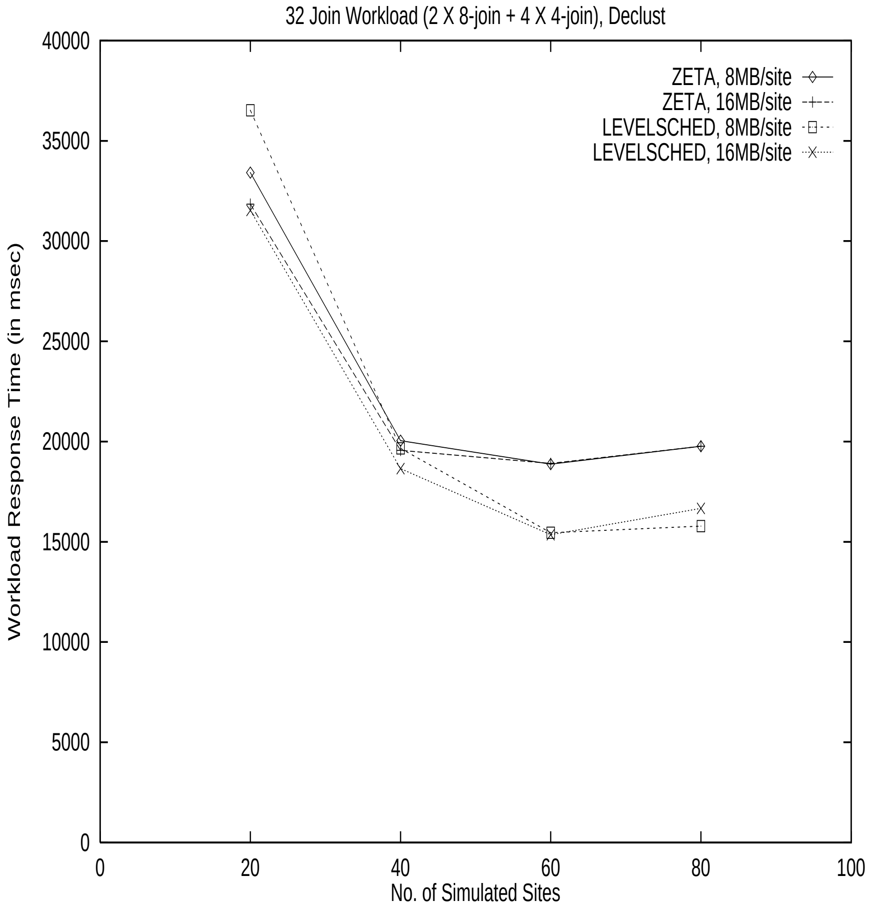
  2. **Declust-1/4** (**NoDeclust-1/4**): as above, but on 1/4th of sites

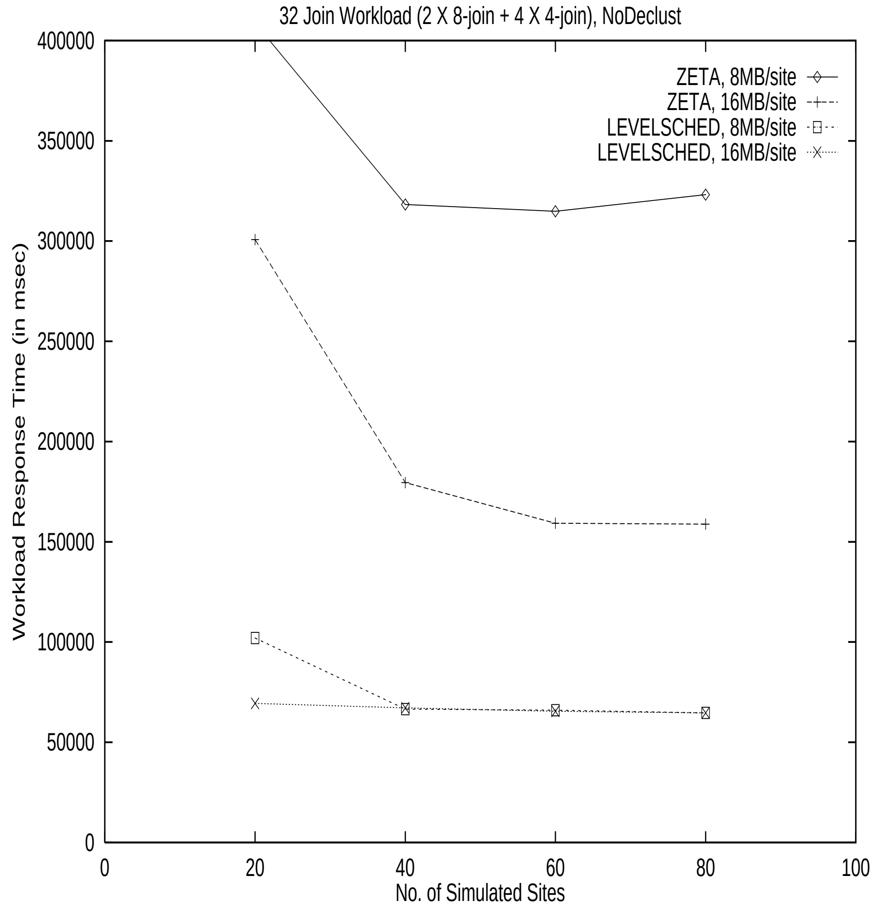
  3. **Random**: degree of declustering and home randomly chosen

  4. **QueryBasedDeclust**: degree of declustering according to relation size, no overlaps between joins

Simulation Results

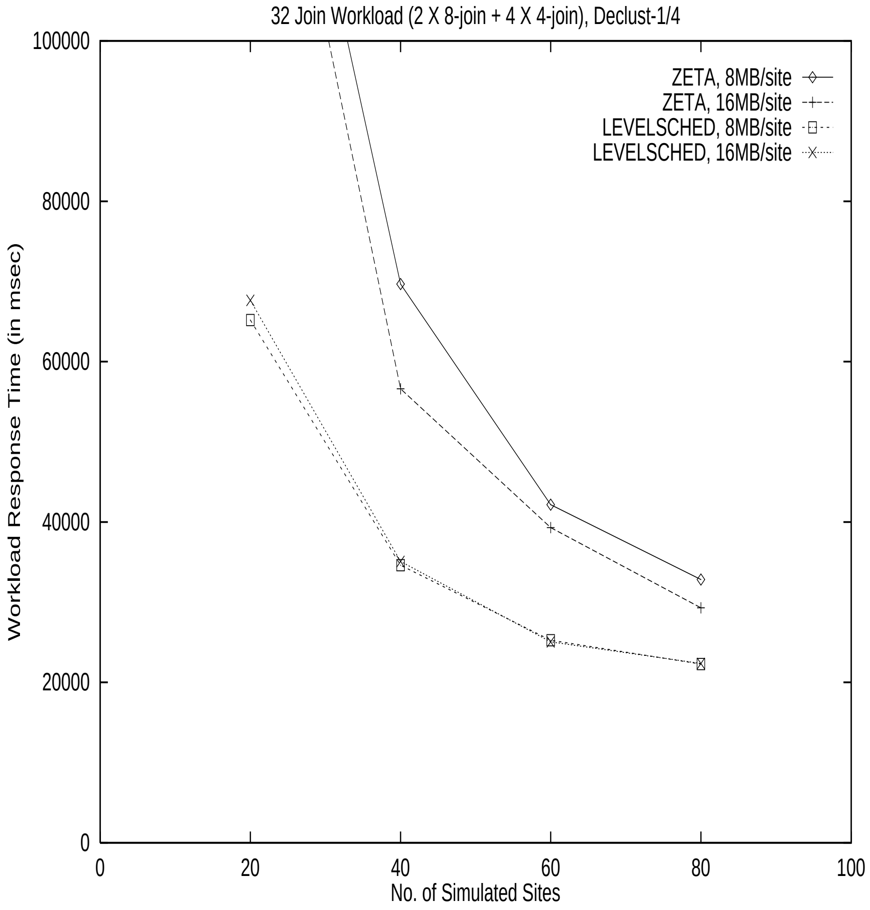
- Declust
- NoDeclust

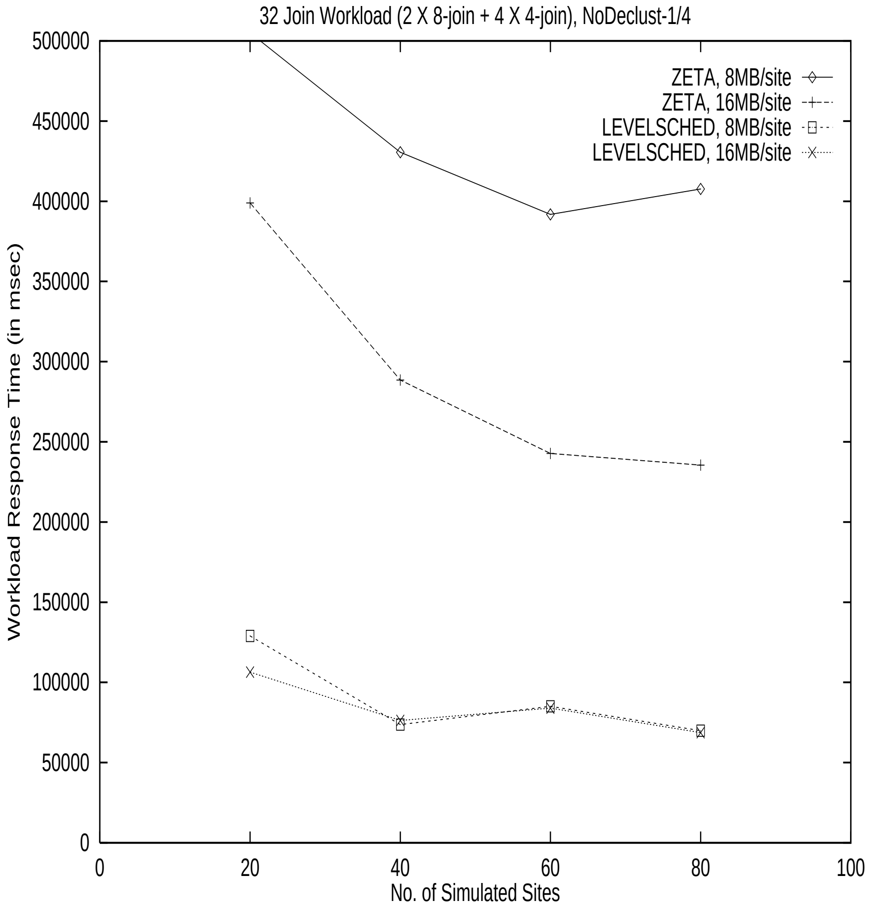




Simulation Results (cont.)

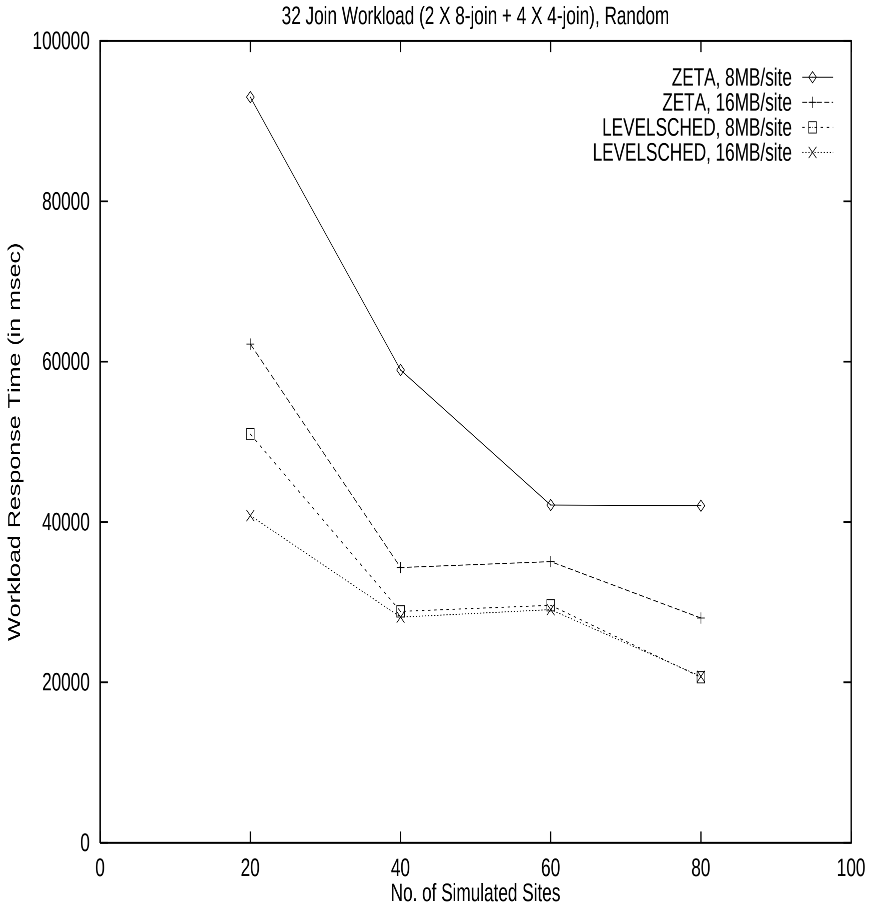
- Declust-1/4
- NoDeclust-1/4

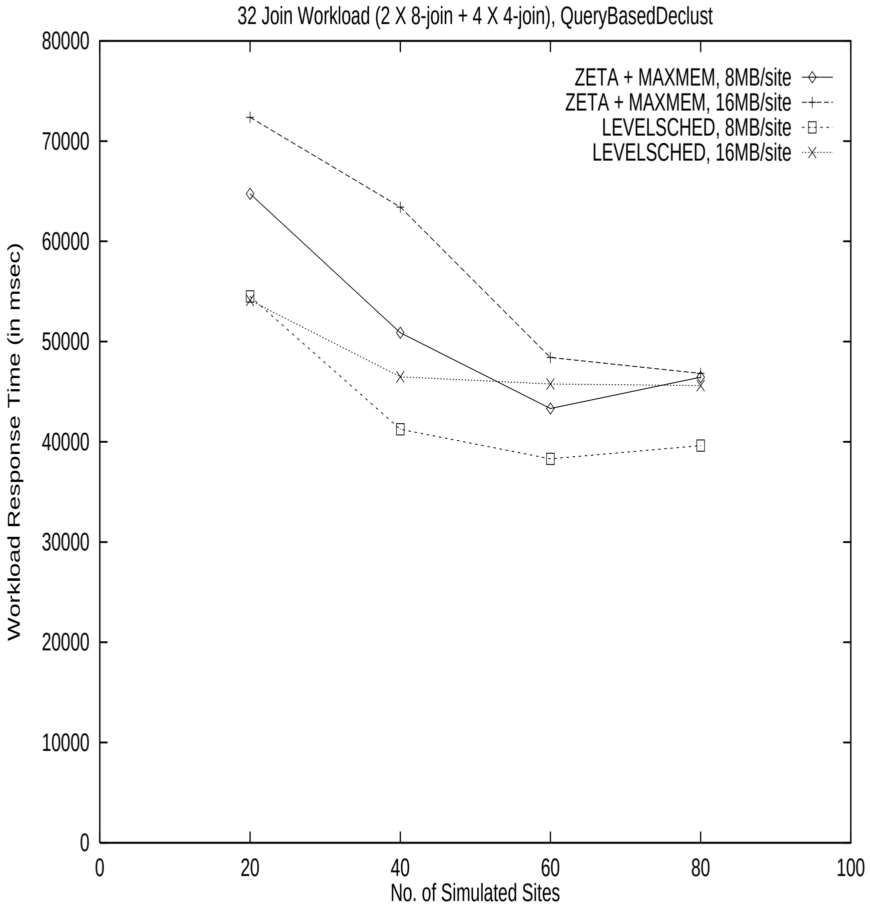




Simulation Results (cont.)

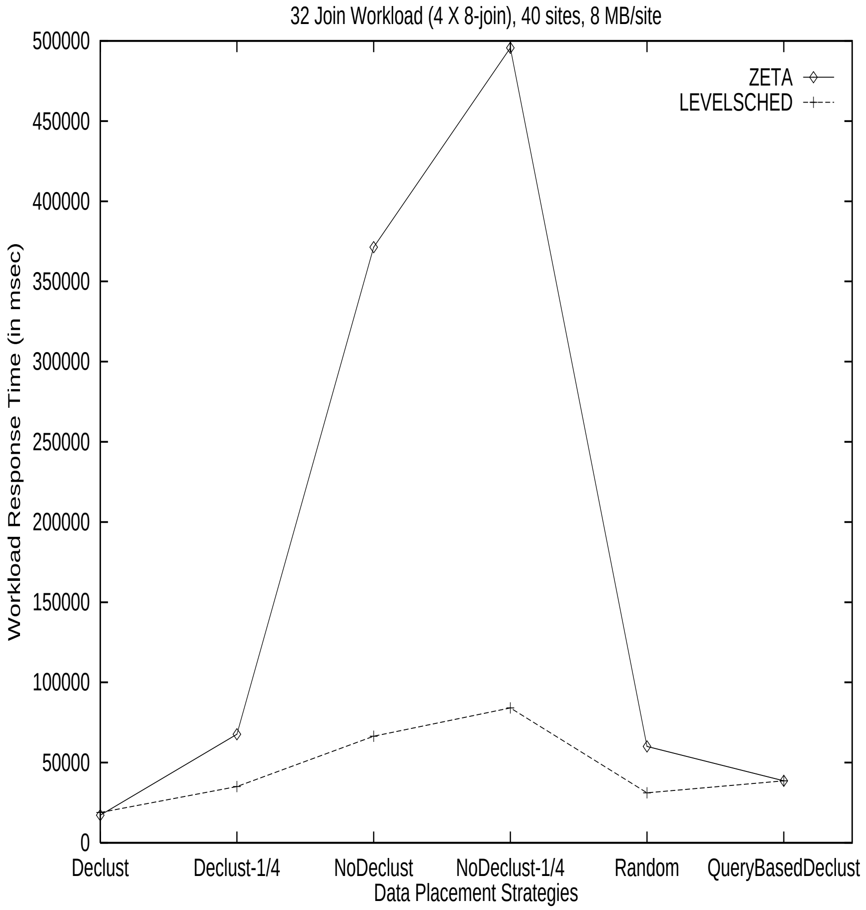
- Random
- QueryBasedDeclust

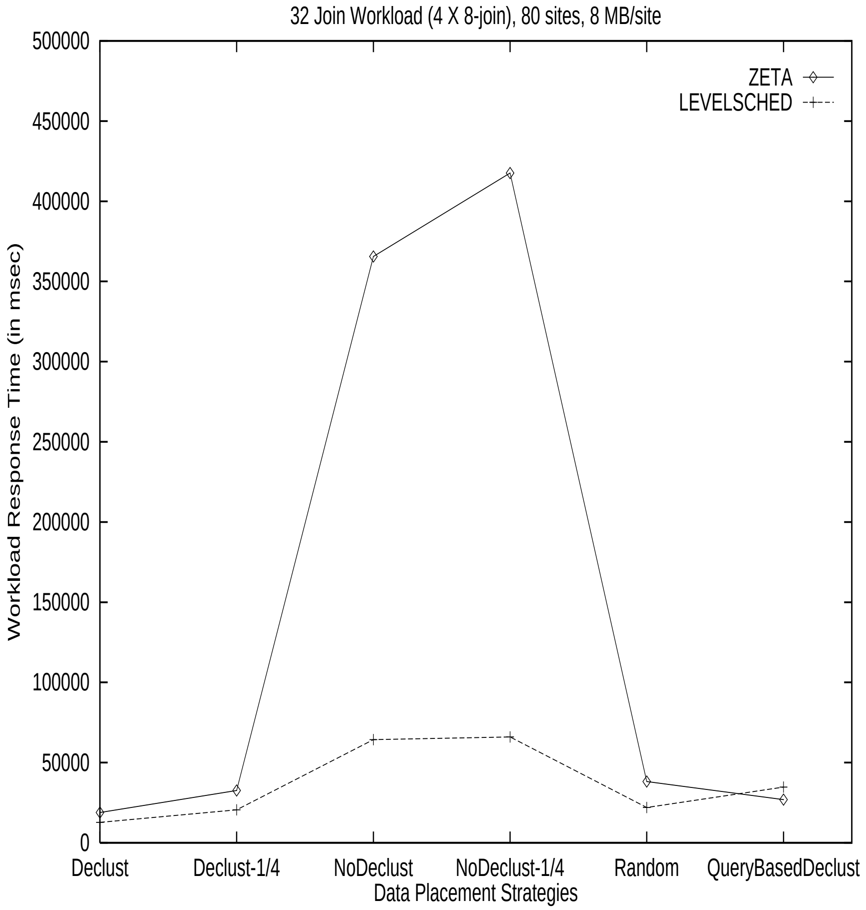




Simulation Results (cont.)

- 40 sites, 8MB/site
- 80 sites, 8MB/site



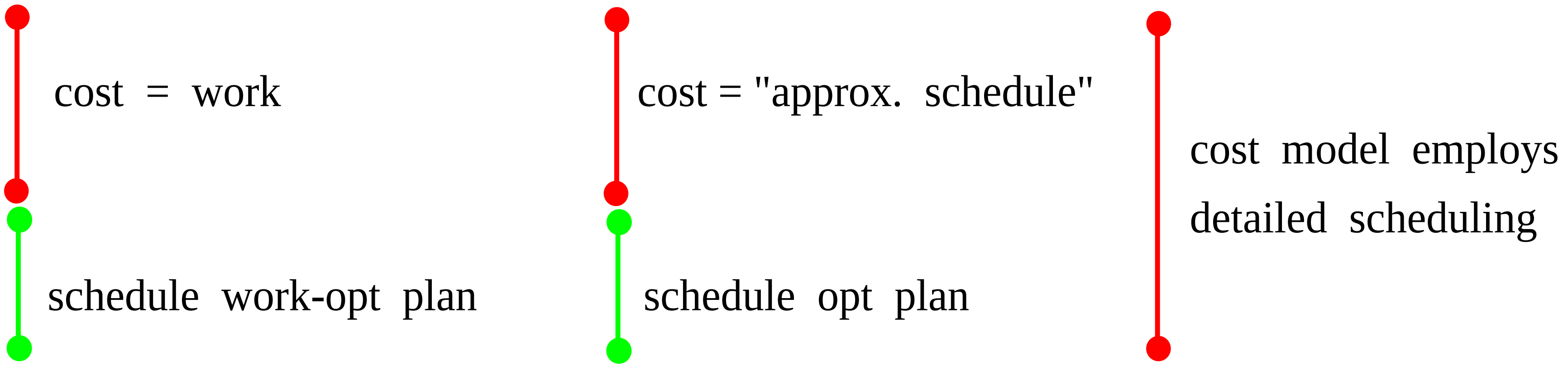


Parallel Query Optimization

Cost Model: "plan quality" by abstracting execution environment

2-phase

1-phase



- [Ganguly et al., PODS'96]: max { critical path, average work }
- Our results: *average volume* captures the effects of SS resources
- $<$ *critical path, average work* , *average volume* $>$: crucial bulk parameters of a parallel plan for efficient optimization

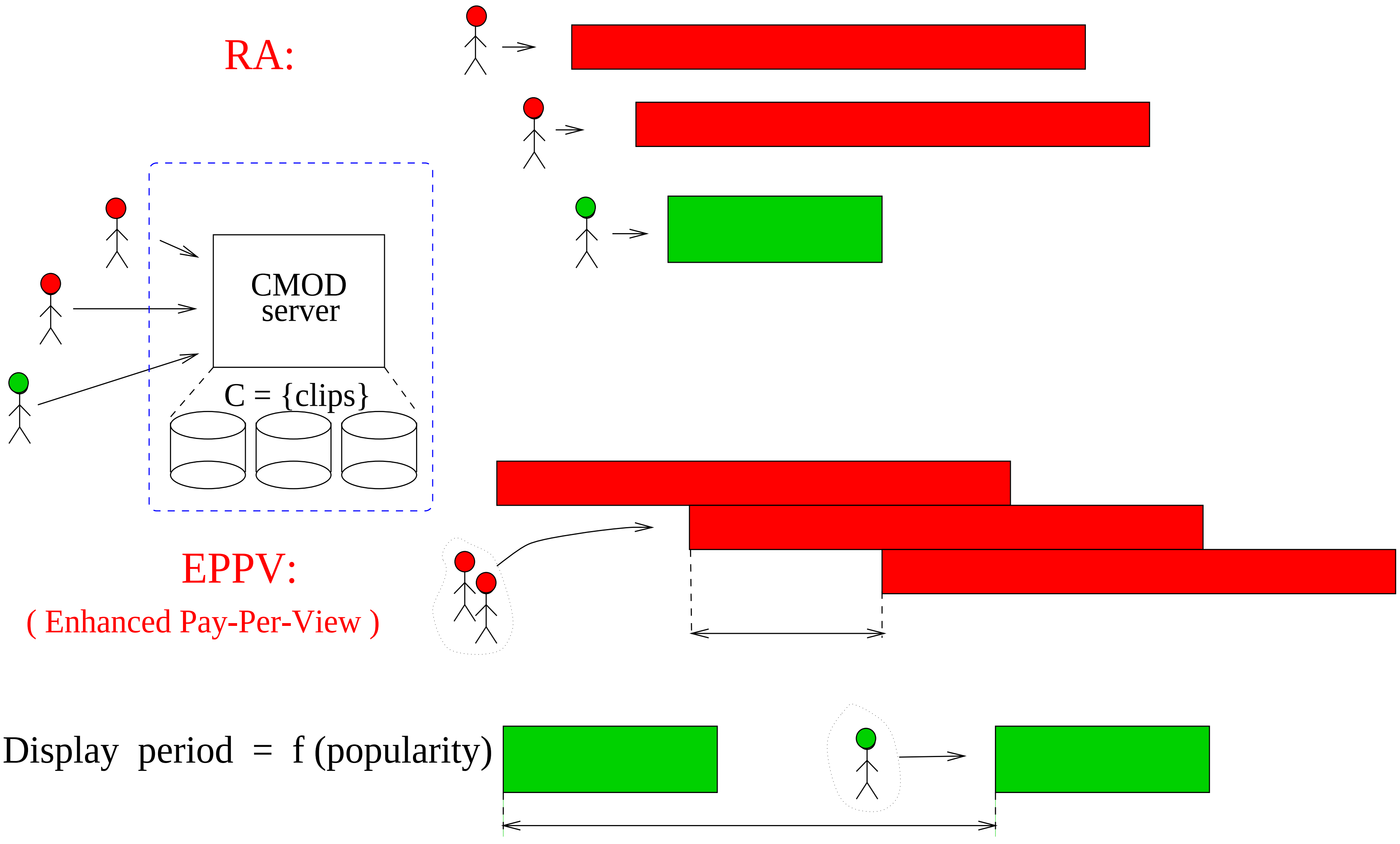
Part II

- Multi-dimensional Parallel Query Scheduling [SIGMOD'96, VLDB'97]
- ⇒ Periodic Resource Scheduling for Continuous Media DBs [VLDB'97, RIDE'98]

Periodic Scheduling in Continuous Media DBs: Outline

- Introduction and Motivation
- Problem Formulation and System Model
- Clustering
- Striping
- *Main Contribution:* Scheduling Tree Structure & Algorithms
- Experimental Results

Service Models: Random Access vs. EPPV



Introduction and Motivation

Why EPPV?

- Cost-effective: concurrency not limited by available resources
- Predictable response times

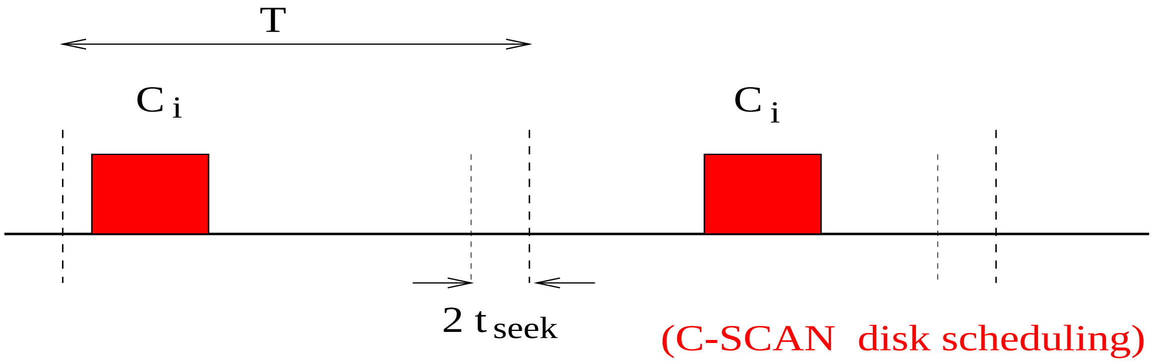
Challenge: Complex periodic task scheduling problems

The Problem:

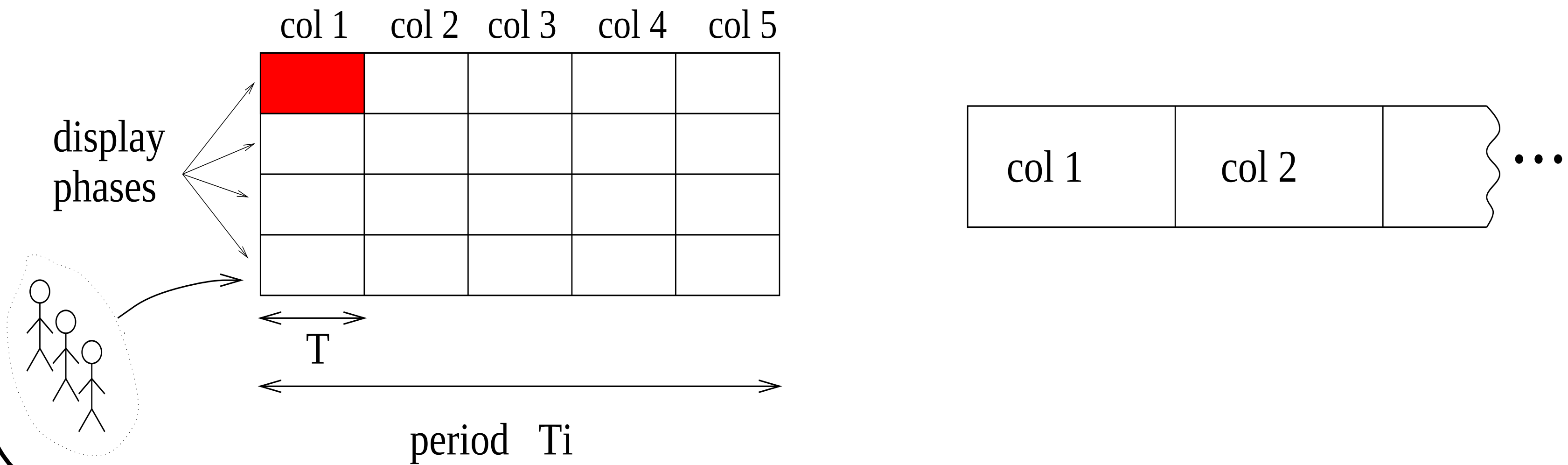
- **Given:** disk H/W parameters, data layout strategy, and set of clips C
- **Find:** placement and feasible schedule for subset of C that maximizes a "benefit" (e.g., scheduled disk bandwidth)

System Model

- Round-based Retrieval: data for streams retrieved in rounds of length T

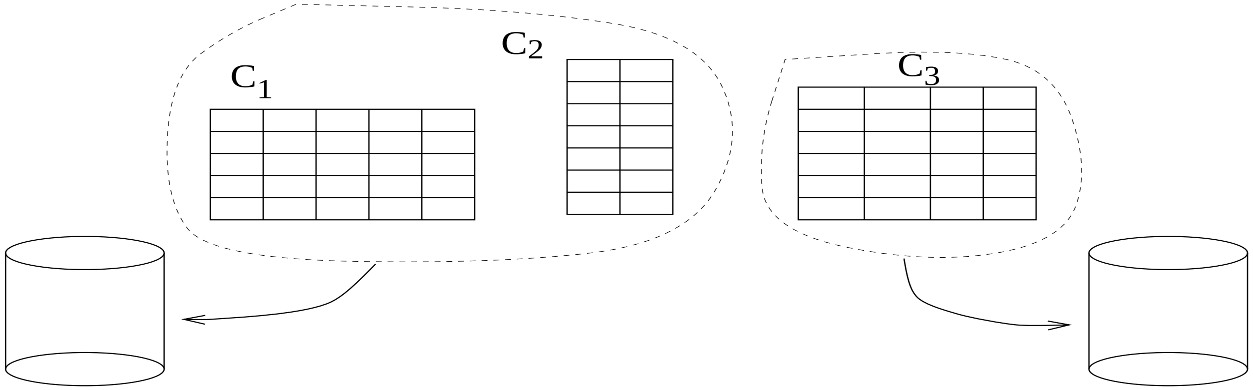


- Matrix-based Allocation [Özden et al., VLDB'94]: designed for EPPV



EPPV under Clustering

- Clustering: entire clips mapped to individual disks



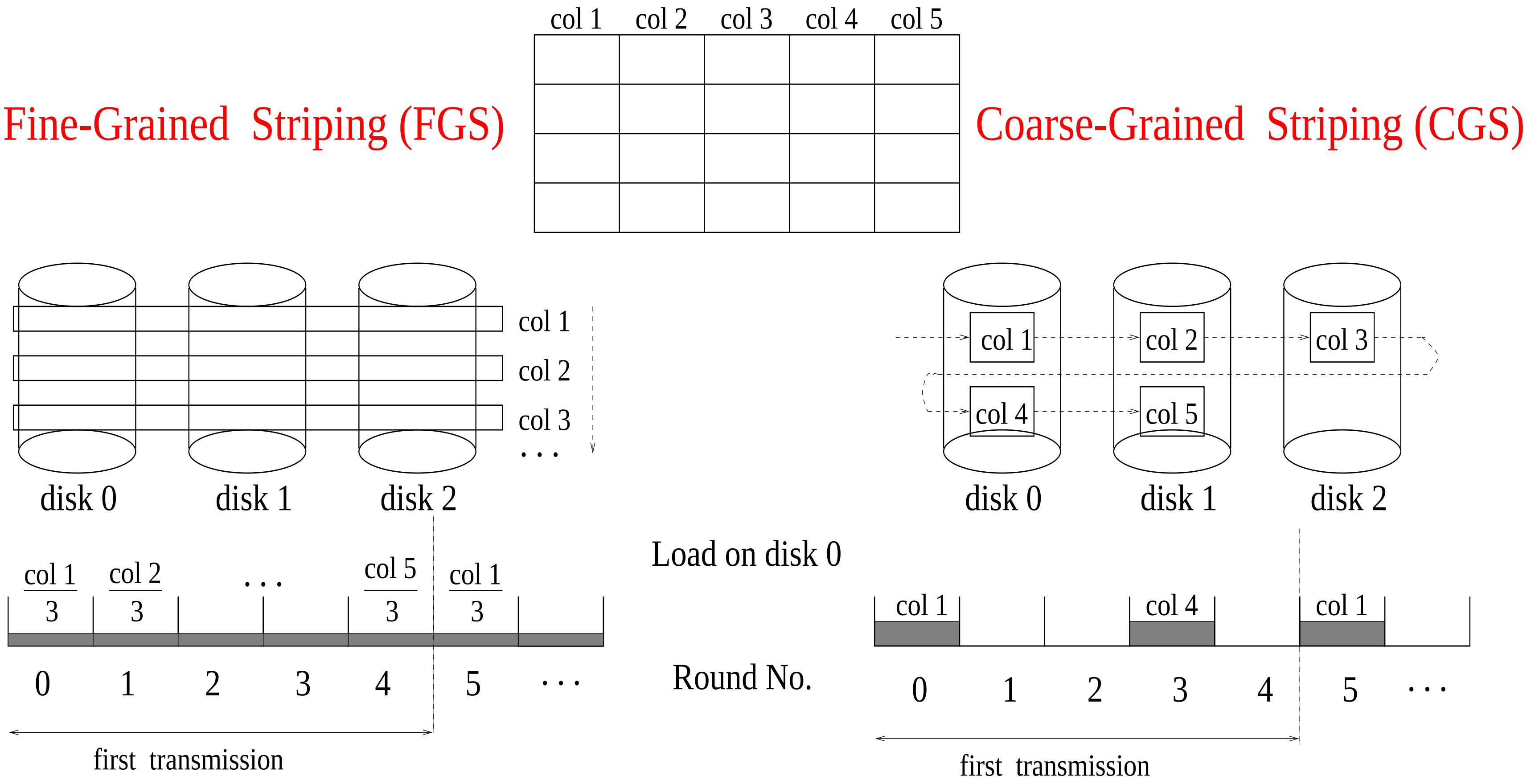
- size(C_i) = 2-dimensional: fraction of disk bandwidth and fraction of disk storage consumed by C_i

  value(C_i) = disk bandwidth effectively utilized by C_i

- NP-hard "knapsack" problem

- Near-optimal heuristic solution (PackClips)

EPPV under Striping



- FGS: limited scalability because of latency overheads
- CGS: "composite" periodic tasks (period $n_i \cdot T$, subtasks $n_{disk} \cdot T$ apart)

CGS and Periodic Maintenance

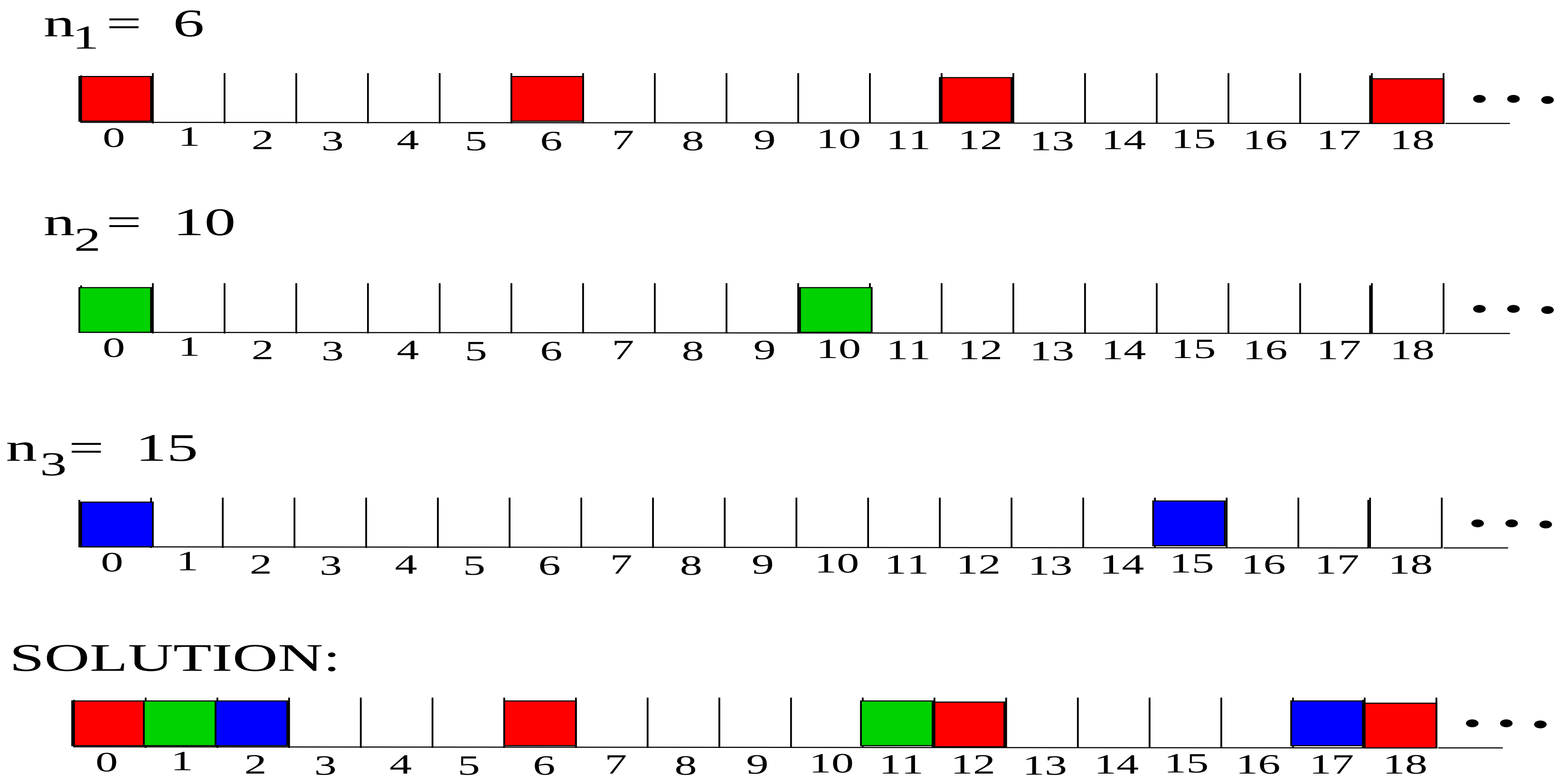
Collision-free scheduling for CGS

- Special cases: conditions based on *Generalized Chinese Remainder Thm*
- General case: generalization of *Periodic Maintenance*

Periodic Maintenance: Given periodic tasks $\{C_i\}$ with periods $\{n_i\}$, is there a mapping onto integer time slots s.t. successive occurrences of C_i are exactly n_i apart and no collisions occur?

- Special case of CGS scheduling (one subtask per task)
- NP-complete in the strong sense [Baruah et al., 1990]

Periodic Maintenance: Example



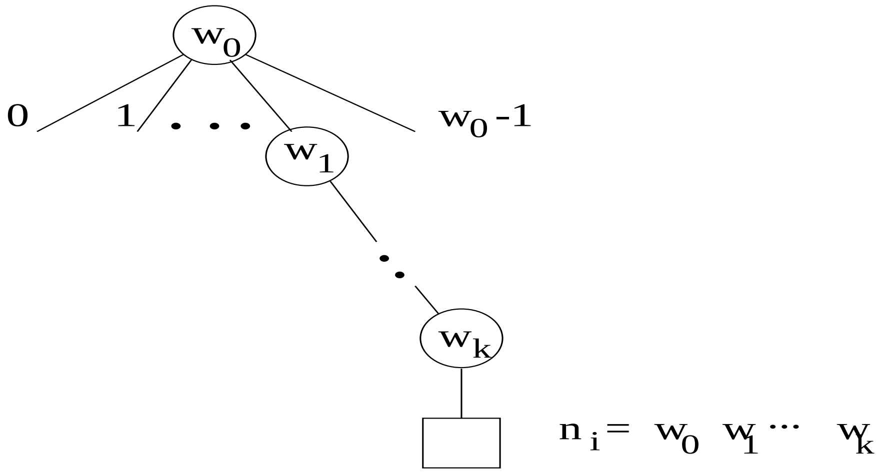
- No solution exists, e.g. for $\{3, 5\}$ (*Chinese Remainder Theorem*)

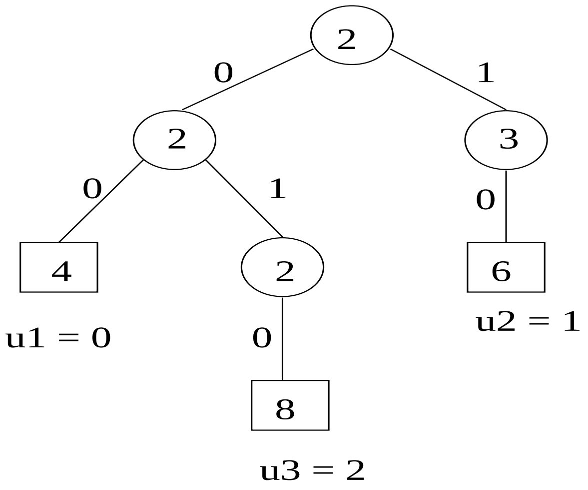
Our Main Contribution: Scheduling Trees

Goal: provide solutions for Periodic Maintenance; extend to CGS

Scheduling Tree: integer weighted nodes and edges, where

1. an internal node with weight w has $\leq w$ edges with distinct weights in $\{0, \ldots, w-1\}$
2. a leaf node corresponds to period n_i s.t. n_i = product of all ancestors

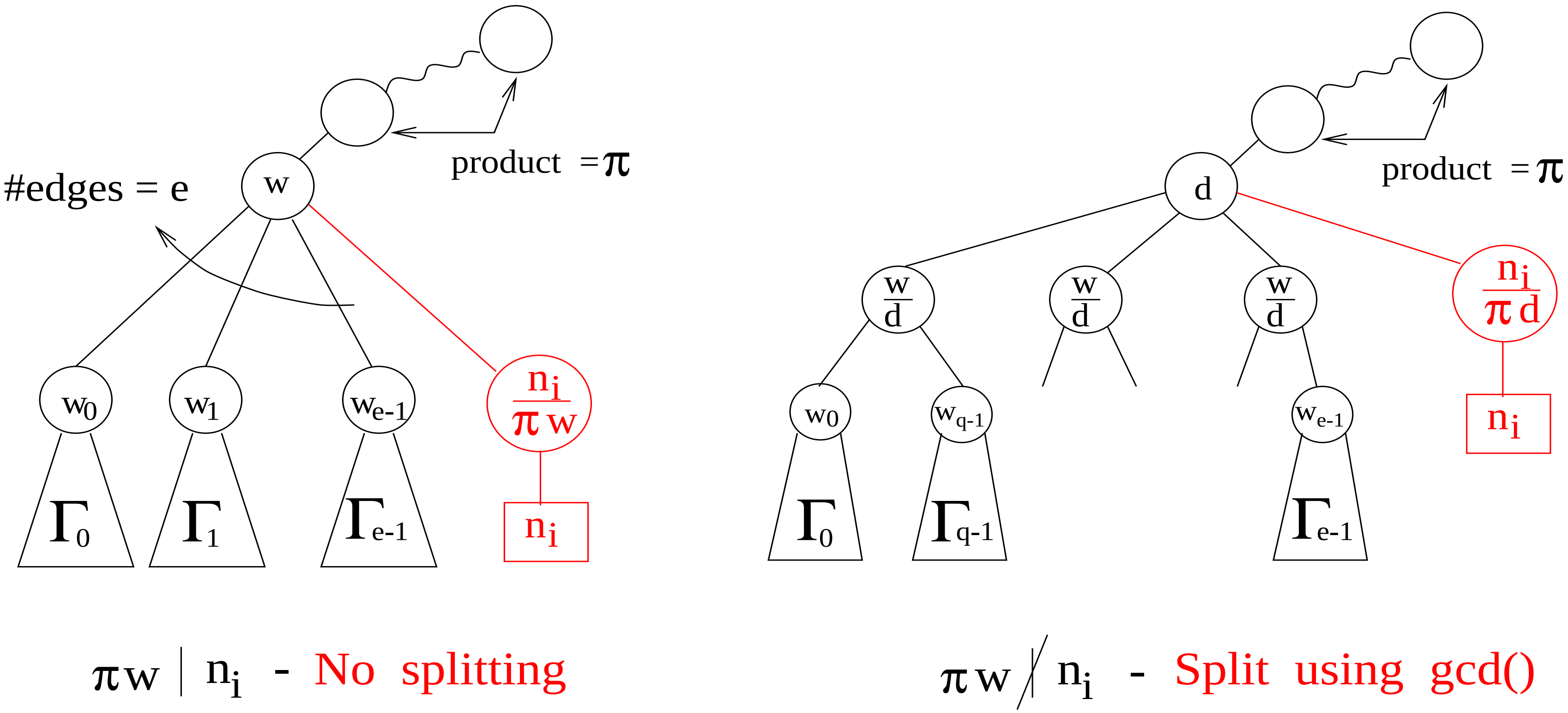




- $\exists$ Scheduling Tree is only sufficient for Periodic Maintenance; e.g., $\{6, 10, 15\}$

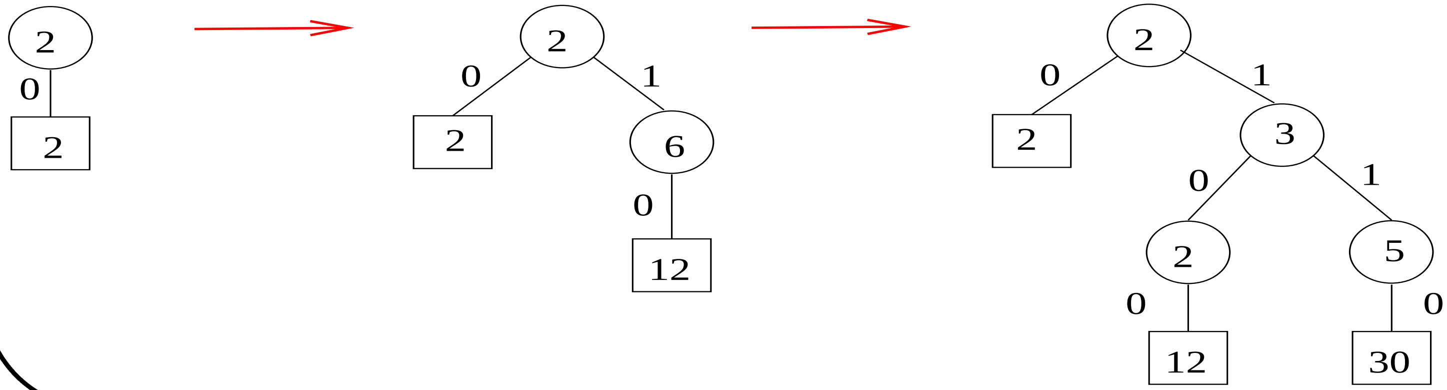
BUILDTREE: A Heuristic for Building Scheduling Trees

- Based on *candidate nodes*
- Node is candidate for n_i iff $\gcd(w, \frac{n_i}{\pi}) \geq \frac{w}{w-e}$



BuildTree: A Heuristic for Building Scheduling Trees

- Main Ideas:
  - Maintain candidates incrementally as tree is built (efficient...)
  - Consider tasks/periods in decreasing order of benefit
  - Place under candidate that minimizes total loss over all remaining periods
- Complexity = O(N^3)
- Simple example: Period list = < 2 ,12 , 30 >



Scheduling Tree Solutions for CGS

- Generalize BuildTree to handle CGS periodic scheduling problem
- Experimental results: “CGS + Scheduling Trees” is clear winner

Conclusions

Parallel Query Scheduling

- Comprehensive formal approach to multi-dimensional query scheduling in parallel systems
- Efficient, near-optimal heuristic algorithms
- Novel, effective cost model for parallel query optimization capturing TS & SS dimensions

Resource Scheduling Support for EPPV

- CGS + Periodic Scheduling is better
- General framework and methodology for a large class of periodic scheduling problems
- Extensions: periodic scheduling can benefit other layout and storage allocation strategies

Future Work

Parallel Query Scheduling

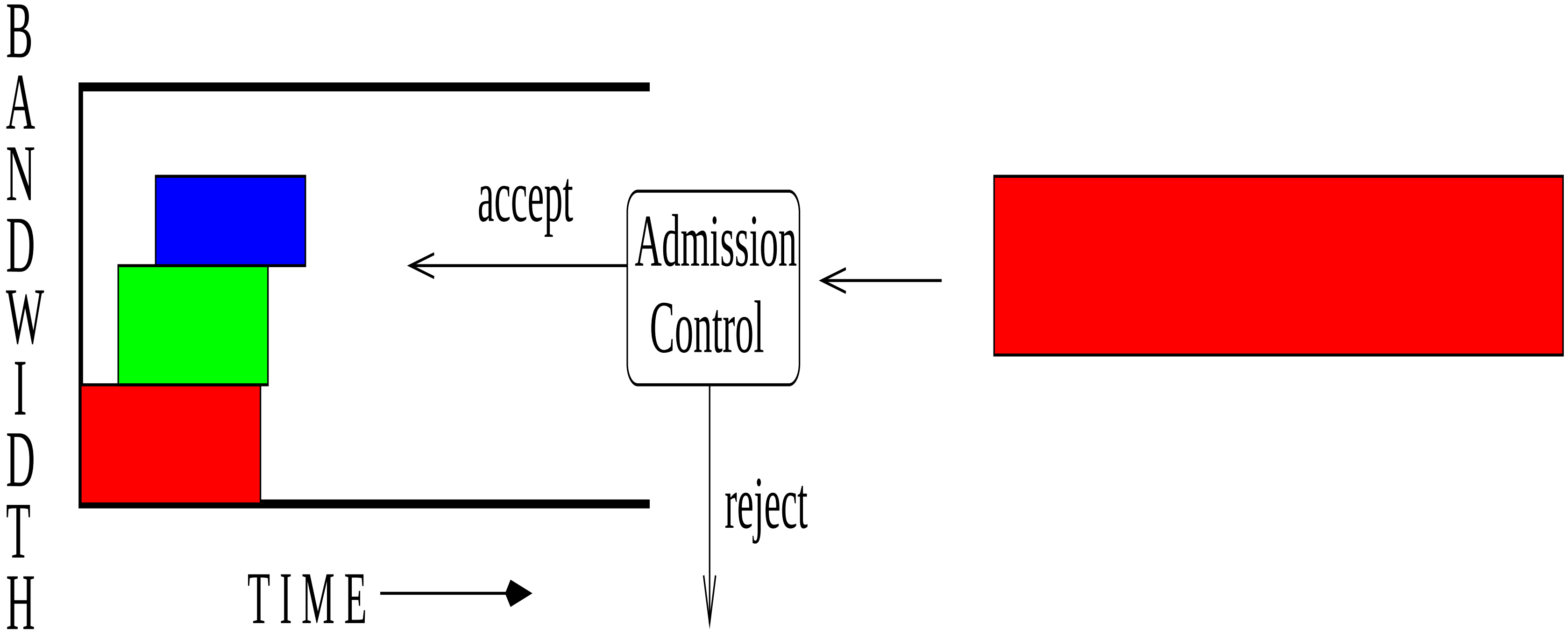
- *Malleable* memory demands
- More complex pipelining strategies that work with limited memory
- Different degrees of "Time-Shareability" (e.g., disk vs. CPU)
- Validate scheduling algorithms and optimizer cost model in a real system

EPPV Resource Scheduling

- Incorporating memory buffering
- Handling task arrival and departure dates
- Prototype implementation under way in *CineBlitz*

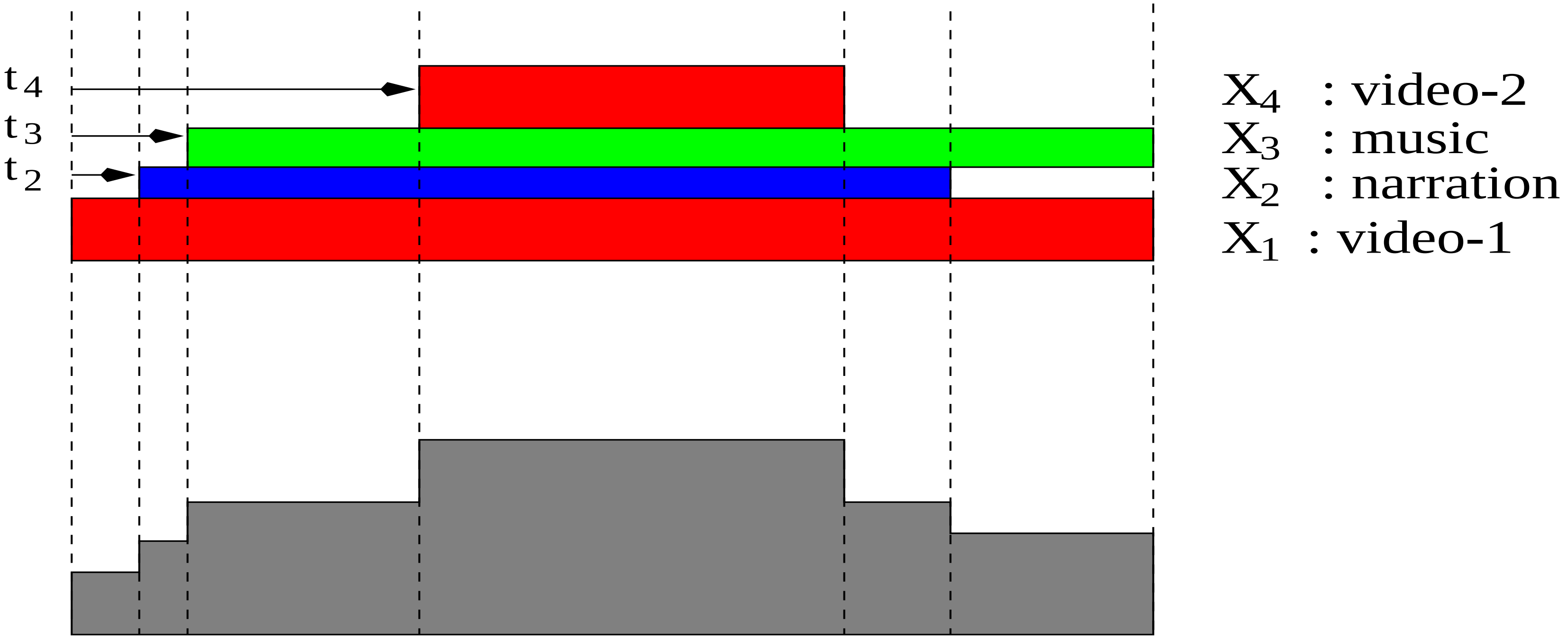
Other Thesis Work

- Online Admission Control in Continuous Media DBs [PODS'98]



Other Thesis Work (cont.)

- Resource Scheduling for Composite Multimedia Objects [VLDB'98]



Prior Work

(a) DB Research

- One-dimensional model of resource allocation (scalar “work” metric)
- Limited forms of parallel plans (often no bushy tree parallelism)

Schneider and DeWitt [VLDB’90], Lo et al. [SIGMOD’93], Shekita et al. [VLDB’93], Hsiao et al. [SIGMOD’94], Hasan and Motwani [VLDB’94], Chekuri et al. [PODS’95]

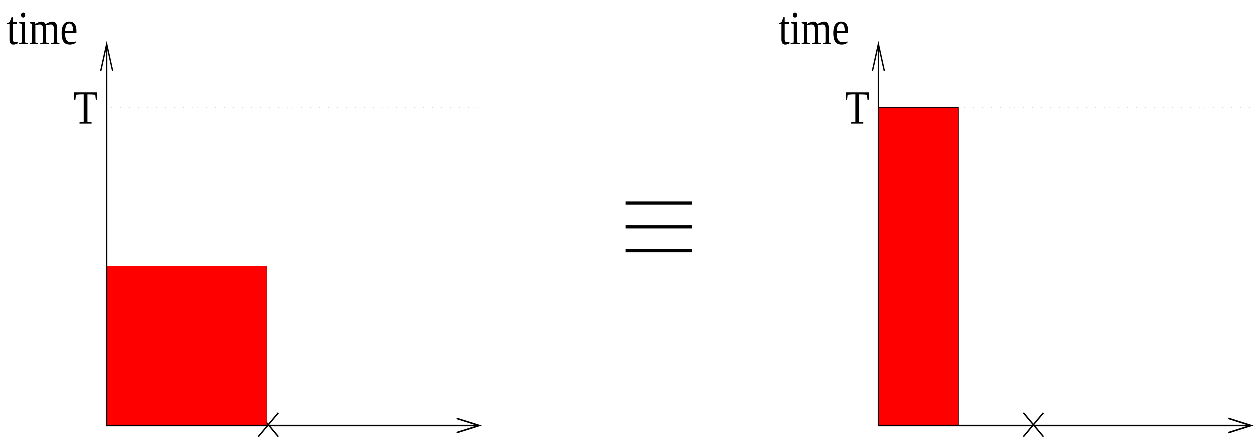
(b) Deterministic Scheduling Theory

- Multiprocessor scheduling: mapping DAGs of computation-intensive tasks on parallel machines
- One-dimensional models of parallelization or globally accessible resources

Main Assumptions

- No time-slicing overhead for TS resources

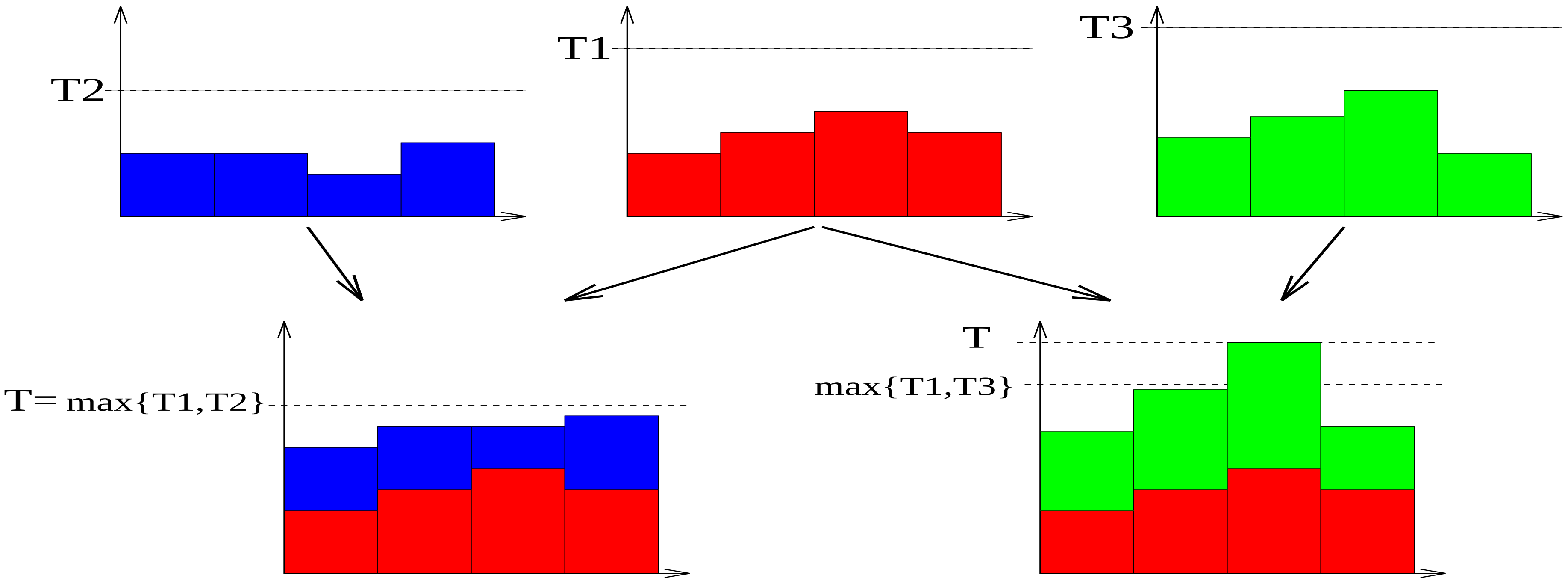
- Uniform TS resource usage



- Constant SS resource demands

Concurrent Execution

- Feasible *only if* total SS requirements ≤ 1
- Effects of TS resource sharing



- Response Time: slowest clone or most heavily congested resource

Partitioned Parallelism

Degree of Parallelism? Tradeoff...

- Finer clones: satisfy SS resource demands, better load balancing
- Coarser clones: reduce communication costs

Independent Parallelism

- *Compatible clones*: total SS resource demands ≤ 1
- Schedule: partition clones into compatible subsets; map onto sites
- Response time in a compatible partition is determined by TS components
- Algorithm OpSched: simple near-optimal heuristic

Query Task Trees: Handling Data Dependencies

Algorithm TreeSched:

- Maintain dynamic list of enabled ("ready") pipelines starting from leaves
- Ensure resource dependencies are maintained (e.g., `build` $\rightarrow$ `probe`)
- Same ideas can be applied for scheduling tasks from multiple queries *on-line* (inter-query parallelism)

Cost Model Experiments: Testbed

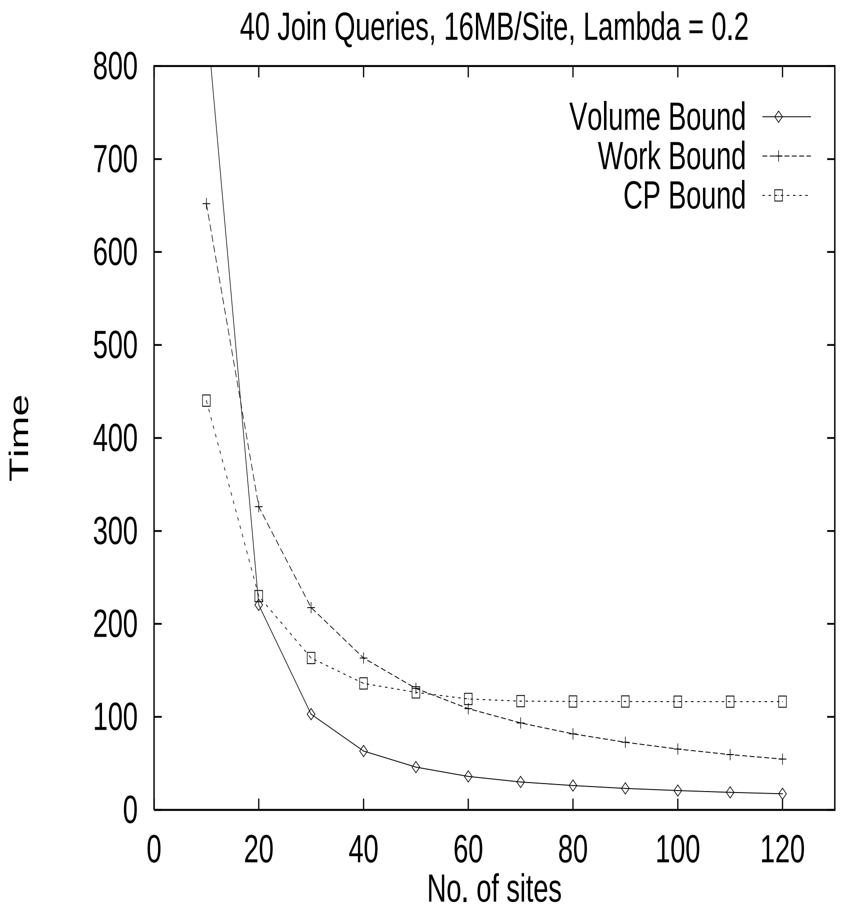
- Sites with d=3 (CPU, Disk, NW interface) and s=1 (Memory)
- Tree queries 10,20,30,40,50 joins (20 random query graphs/size)
- Memory/site: 16MB - 64MB, $\lambda = 0.1$ - 0.9
  Overlap: 10% - 70%, Coarse Granularity: 0.3 - 0.9
- Performance metrics: Average performance ratio, Average schedule response time

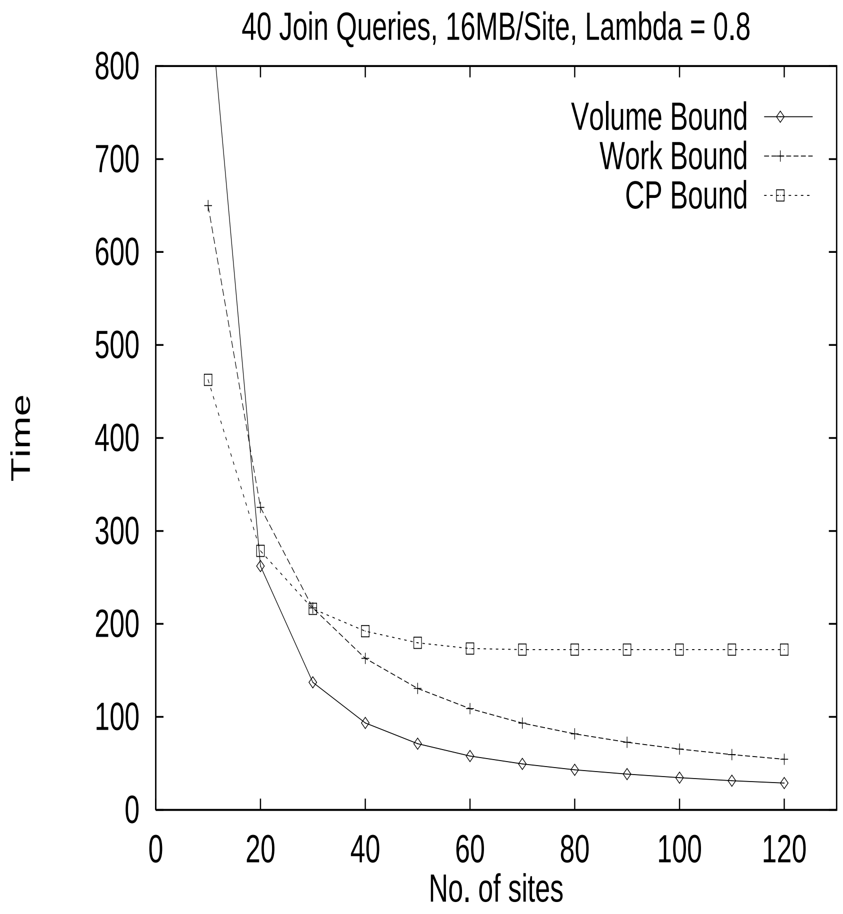
Cost Model Parameter Settings

Configuration/DB Catalog Params	**Value**
Number of Sites	10 – 120
CPU Speed	10 MIPS
Memory per Site	16 – 64 MB
Effective Disk Service Time per page	5 msec
Startup Cost per site (α)	15 msec
Network Transfer Cost per byte (β)	0.3 μsec
Fudge Factor (F)	1.4
Tuple Size	128 bytes
Page Size	32 tuples
Relation Size	10^4 – 10^6 tuples
CPU Cost Parameters	**No. Instr.**
Read Page from Disk	5000
Write Page to Disk	5000
Extract Tuple	300
Hash Tuple	100
Probe Hash Table	200

Experimental Results (more...)

- Lower Bound Components, $\lambda = 0.2$, 0.8





Simulation Parameter Settings - I

Configuration/Site Parameters	Value
Number of Sites	20 – 80
CPU Speed	20 MIPS
Number of Disks per Site	2
Memory per Site	8 – 16 MB
Page Size	8 KB
Latency for 8K Message	1.8 msec
Disk Seek Factor	0.617
Disk Rotation Time	16.667 msec
Disk Settle Time	2.0 msec
Disk Transfer Rate	3.09 MB/sec
Disk Cache Context Size	4 pages
Disk Cache Size	8 contexts
Disk Cylinder Size	83 pages
Buffer Manager	3-level LRU

Simulation Parameter Settings - II

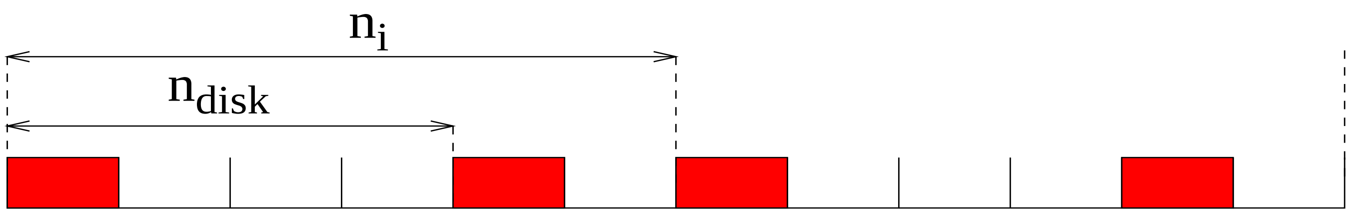
CPU Cost Parameters	No. Instr.
Initiate Select	20000
Initiate Join	40000
Initiate Store	10000
Terminate Select	5000
Terminate Join	10000
Terminate Store	5000
Read Tuple	300
Write Tuple into Output Buffer	100
Probe Hash Table	200
Insert Tuple in Hash Table	100
Hash Tuple Using Split Table	500
Apply a Predicate	100
Copy 8K Message to Memory	10000
Message Protocol Costs	1000

Simulation Schema & Workload Parammeters

Schema/Workload Parameters	Value
Tuple Size	200 bytes
Base Relation Size	1K – 100K tuples
Join Attribute Distribution	Uniform
Workload Size (Total No. of Joins)	8, 16, 32
Right-Deep Query Size	2, 4, 8 joins
Coarse Granularity Parameter (f)	0.001 – 0.1
Memory Granularity Parameter(λ)	0.15 – 0.25
Join Probability	1.0
Projection Factor	0.5

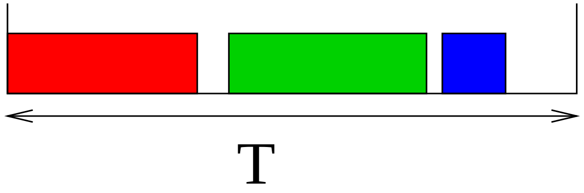
Scheduling Tree Solutions for CGS

Equidistant subtasks?



- Each subtask is itself periodic with period n_i
- Splitting cannot distort relative placement
- Ensure all subtasks are scheduled together at regular intervals of n_{disk}

Scheduling Tree Solutions for CGS (cont.)



Multi-task slots?

- Leaf nodes/slots with sufficient capacity can also be candidates
- Combining (“packing”) independently built trees to improve slot utilization

Combining Scheduling Trees

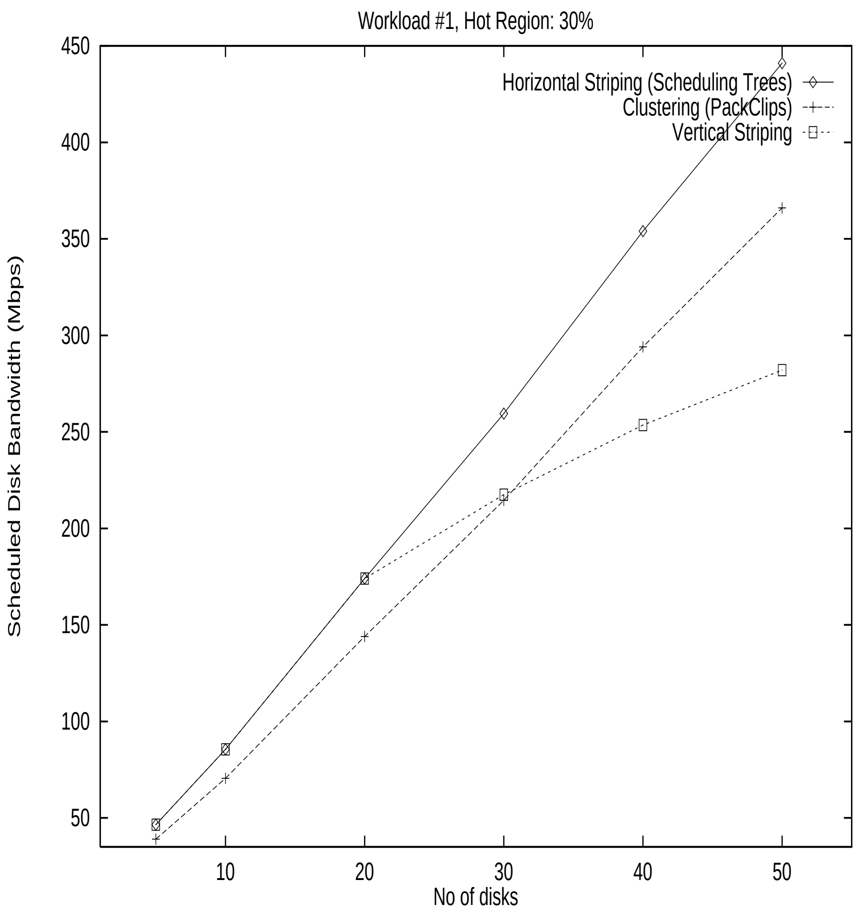
- *Worst-Case Collision Assumption*:
  - size(tree) = max. load among all leaves
  - value(tree) = total benefit of all tasks in tree
- Knapsack problem: use PackClips to approximate solution
- More sophisticated approaches: use Chinese Remainder Theorems

Experimental Results

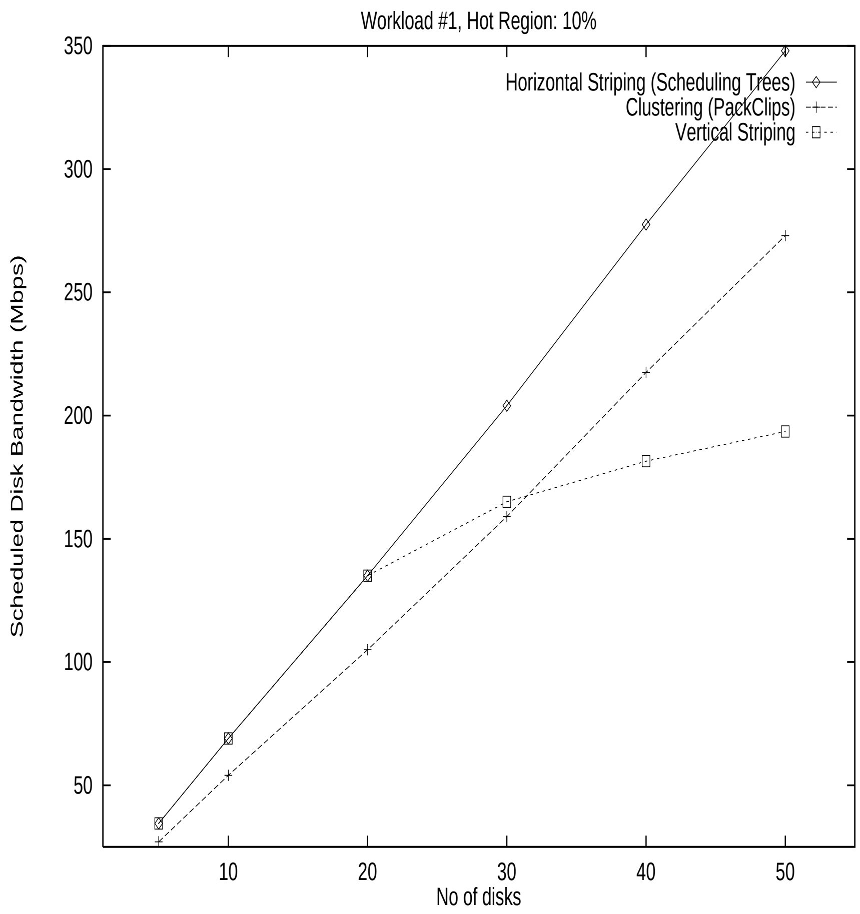
- Compare EPPV scheduling schemes under Clustering, FGS, and CGS
- Metric: effectively scheduled disk bandwidth
- Two workload components
    - Long (90-120min) MPEG-1 movies (1.5Mbps)
    - Short (2-10min) video clips (2-4Mbps)
- "Hot" and "Cold" regions (short/long periods)
- Storage requirements $\approx$ storage capacity of disks

Experimental Results (cont.)

- Long Clips, 30% "Hot"

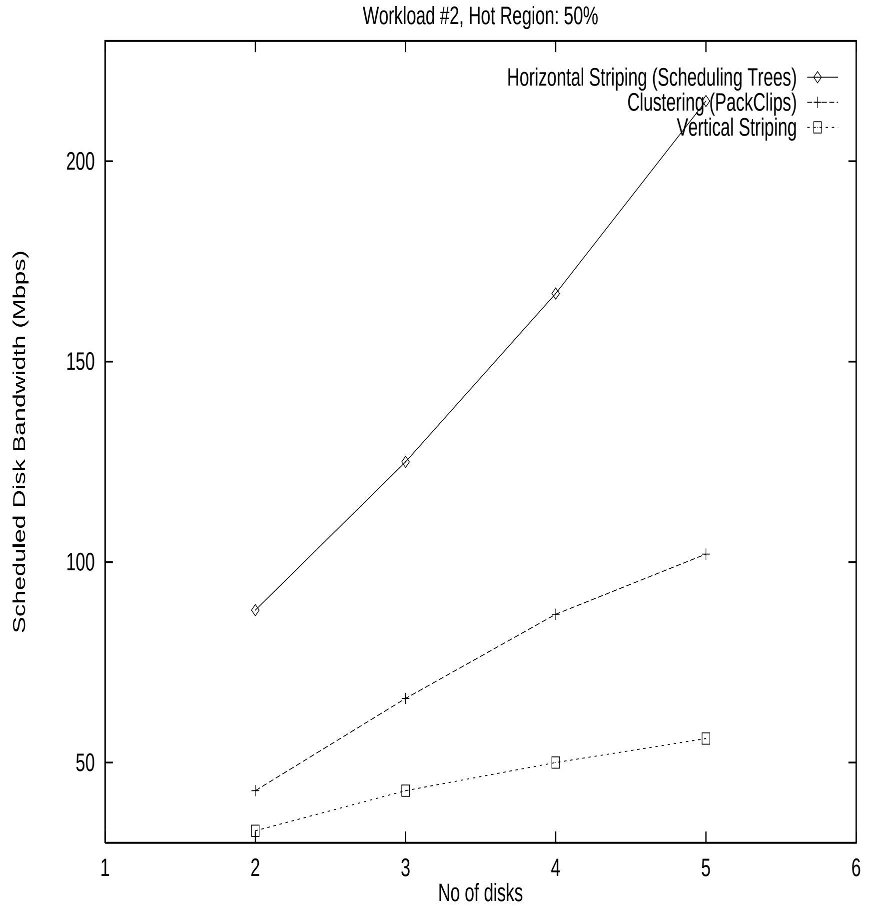


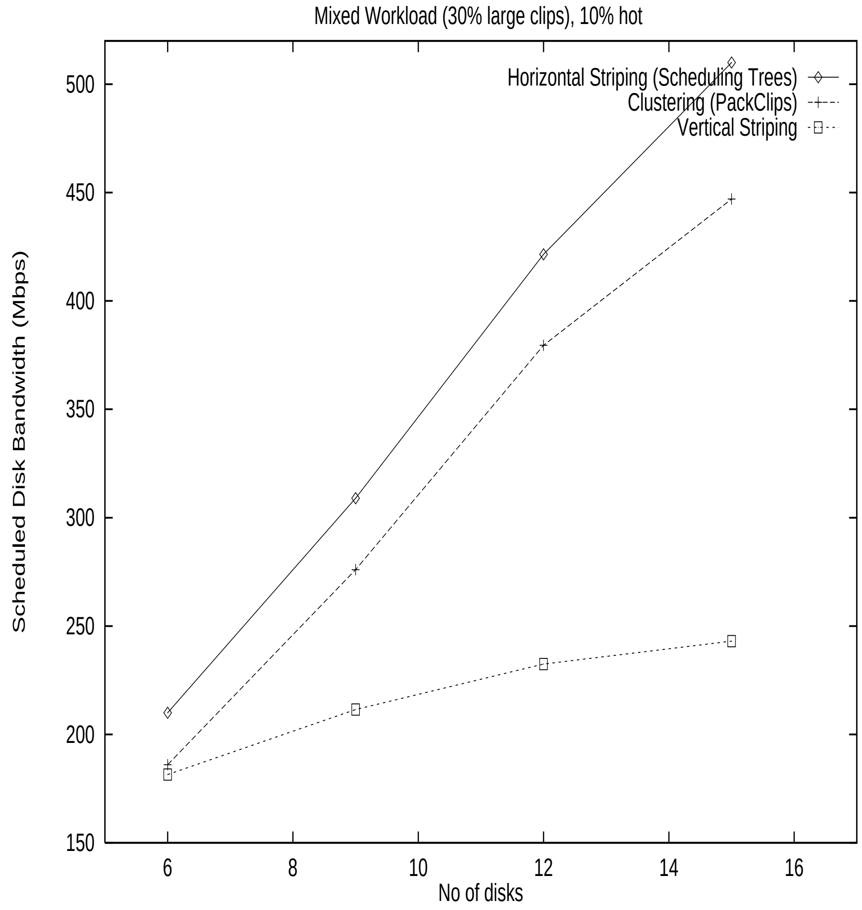
- Long Clips, 10% "Hot"



Experimental Results (cont.)

- Short Clips, 50% “Hot”
- Mixed Clips (30/70), 10% “Hot”





Service Models

Random Access (a.k.a., Fully Interactive) Service Models

- Independent channel per client
- Separate resource reservations per client
- Number of clients that can be serviced is limited with the capacity of available resources

Data Sharing Service Models

- Channels can be shared among different clients
- Goal is to increase number of clients beyond the capacity limitations of available resources

Data Sharing Models

Batching [Dan et al., ACM MM'94], [Shachnai et al., 1995], [Aggarwal et al., ICMCS'96]

- Allows several clients waiting in the server's queue for the same clip to share the same stream
- Goal is to optimize metrics such as fairness, average waiting time, and client reneging probability

Buffering (a.k.a., Bridging) [Kamath et al., DASFAA'95], [Özden et al., ICMCS'95], [Shi et al., ACM SIGMETRICS'97]

- Aims to eliminate disk accesses for some clients
- Uses extra memory to allow a client that arrives with small phase difference to the start of a stream to fetch its data directly from memory

Data Sharing Models (contd.)

Piggybacking [Golubchick et al., 1996]

- Goal is that clients viewing the same clip catch up with each other and share the same stream
- Speeds up or slows down clients compared to the display speed of the corresponding clip

Enhanced Pay-Per-View (EPPV) (a.k.a., Periodic Service) [Özden et al., VLDB'94], [Garofalakis et al., VLDB'97]

- Assigns each clip a *retrieval period*
- Streams retrieving a clip are initiated periodically at offsets equal to the clip's retrieval period
- Multiple clients share the stream

EPPV under Clustering (more...)

- Heuristic Solution: PackClips

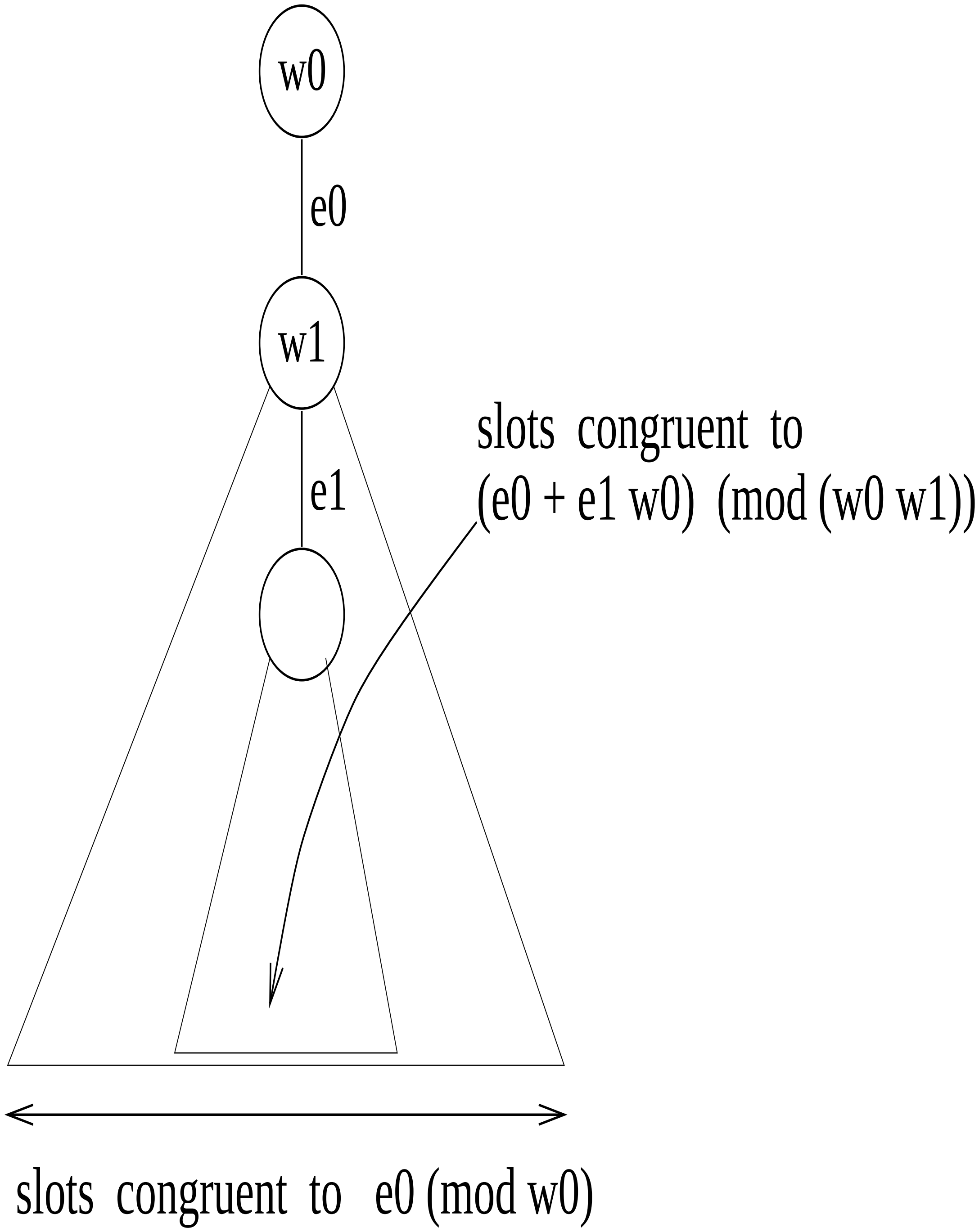
1. Sort C in decreasing order of *value density* = value()/size()
2. Pack into bins using First-Fit
3. Select the n_{disk} most "valuable" bins in the final packing

Bandwidth & Storage Constraints?

- 2-dimensional size() vector = [BANDWIDTH requirement, STORAGE requirement]
- *value density* = value()/max.component{size()} and 2-d vector packing

Theorem: (1) 1-d PackClips is 1/2-approximate (2) 2-d PackClips is 1/4-approximate, assuming STORAGE requirement $\leq 1/2$

Scheduling Tree: some intuition...



Experiment Parameter Settings

Disk Params.	
r_{disk}	80 Mbps
c_{disk}	4 GBytes
t_{seek}	24 ms
t_{lat}	9.3 ms
T	1 sec

Workload #1	
l_i	90 – 120 min
T_i	40-60 , 150-180 min
r_i	1.5 Mbps
No. clips	20 – 200
Hot clips	5% – 50%

Workload #2	
l_i	2 – 10 min
T_i	20-30, 40-60 min
r_i	2 – 4 Mbps
No. clips	80 – 200
Hot clips	5% – 50%