Architectural Support for Scalable Speculative Parallelization in Shared-Memory Multiprocessors

Marcelo Cintra, José F. Martínez, Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign

Speculative Parallelization

- Codes with access patterns not analyzable by compiler
- Speculative parallelization can extract some parallelism
- Several designs of speculative CMPs
- Goal & Contribution: Scalable speculative architecture using speculative CMPs as building blocks
  - trivially defaults for single-processor nodes

Avg. speedup of 5.2 for 16 processors
(for dominant, non-analyzable code sections)

Outline

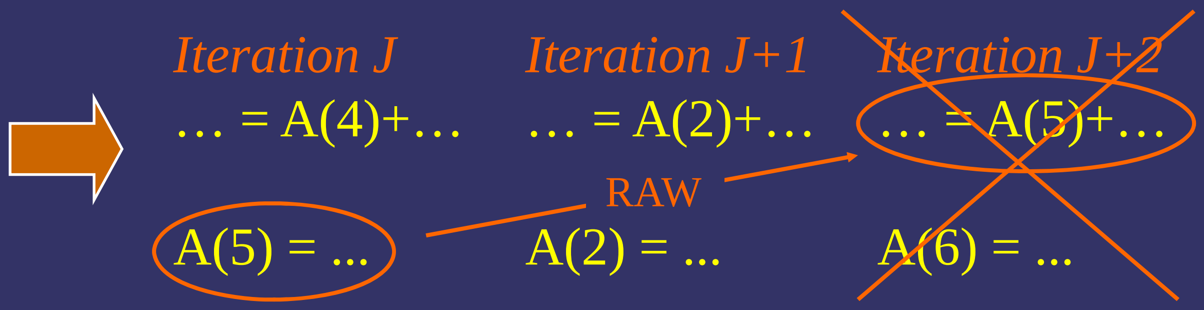
- Motivation
- Background
- Speculative CMP
- Scalable Speculation
- Evaluation
- Related Work
- Conclusions

Speculative Parallelization

- Assume no dependences and execute threads in parallel
- Track data accesses
- Detect violations
- Squash offending threads and restart them

```
Do I = 1 to N
   … = A(L(I))+…

   A(K(I)) = …
EndDo
```



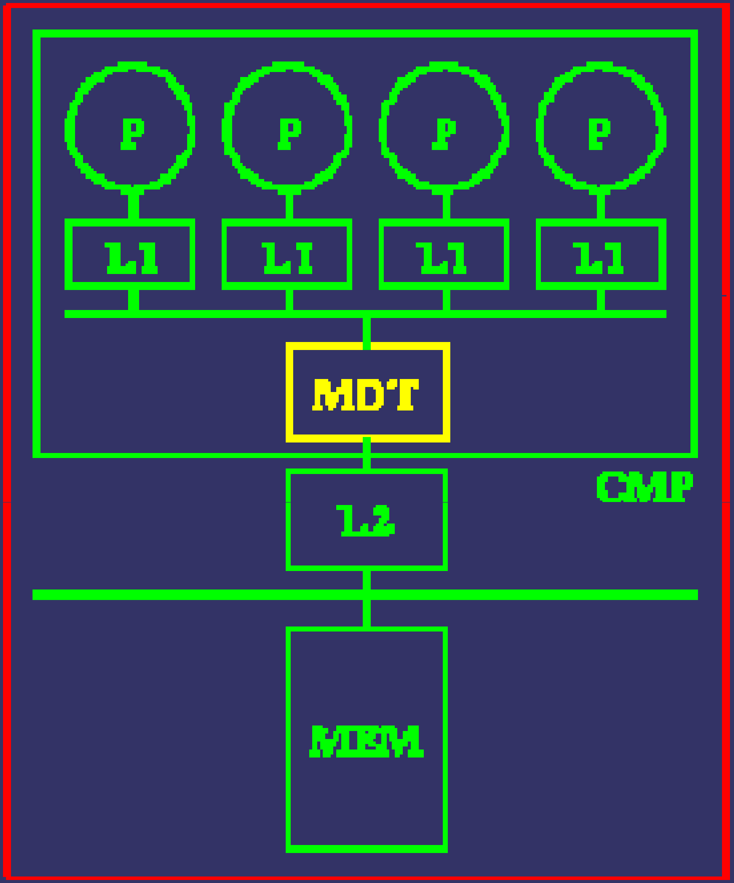
Iteration J	*Iteration J+1*	~~*Iteration J+2*~~
… = A(4)+…	… = A(2)+…	~~… = A(5)+…~~
A(5) = ...	A(2) = ...	~~A(6) = ...~~

RAW

Outline

- Motivation
- Background
- Speculative CMP
- Scalable Speculation
- Evaluation
- Related Work
- Conclusions

Speculative CMP



MDT Memory Disambiguation Table

[Krishnan and Torrellas, ICS 98]

Memory Disambiguation Table (MDT)

- Purpose: detect dependences + locate versions
- 1 entry per memory line touched
- Load and Store bits per word per processor

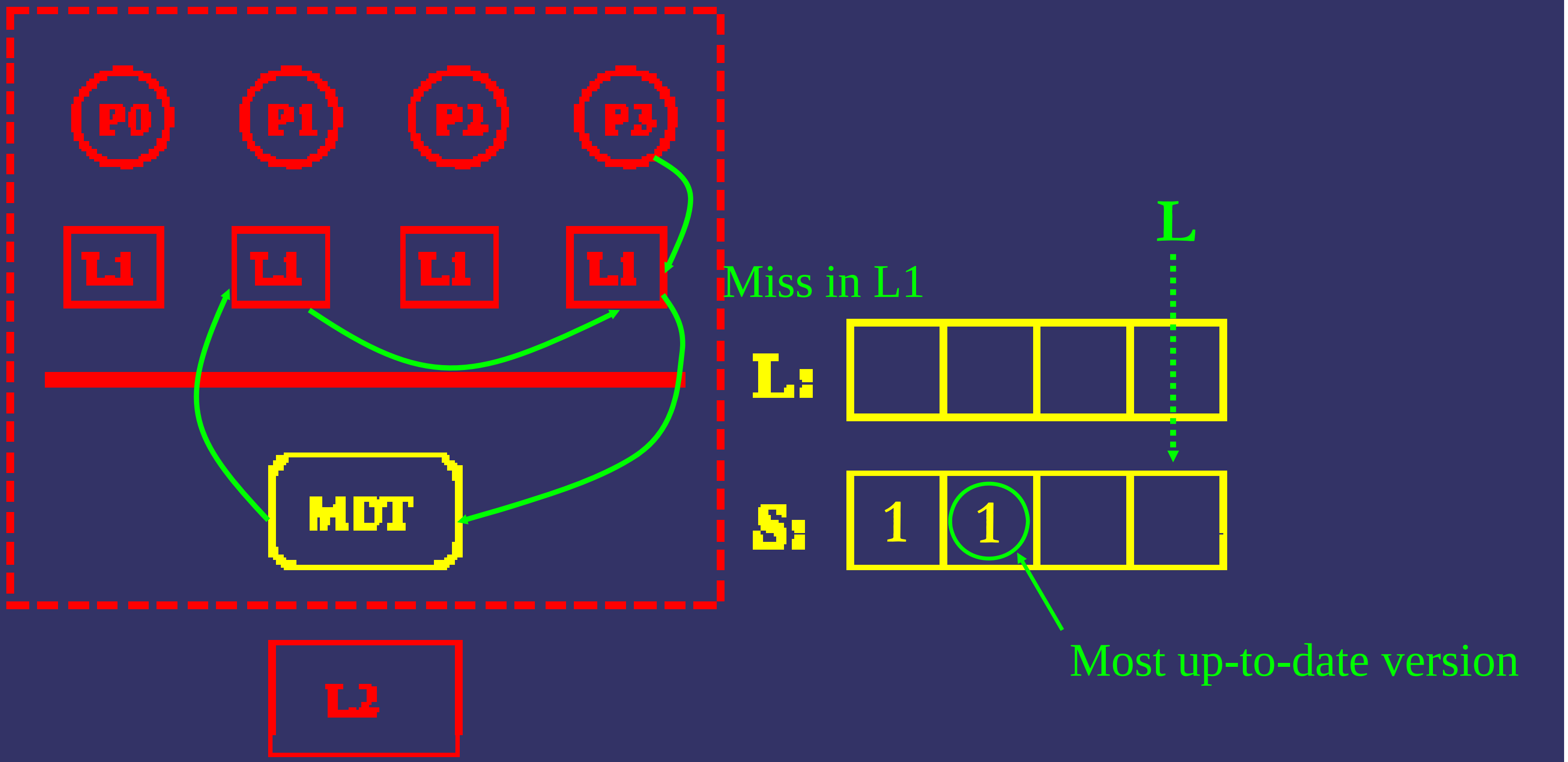
		Word 0								Word 1							
		Load				Store				Load				Store			
Valid	Tag	P0	P1	P2	P3	P0	P1	P2	P3	P0	P1	P2	P3	P0	P1	P2	P3
1	0x1234	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Processor 2 has loaded word 0

Processor 1 has modified word 0

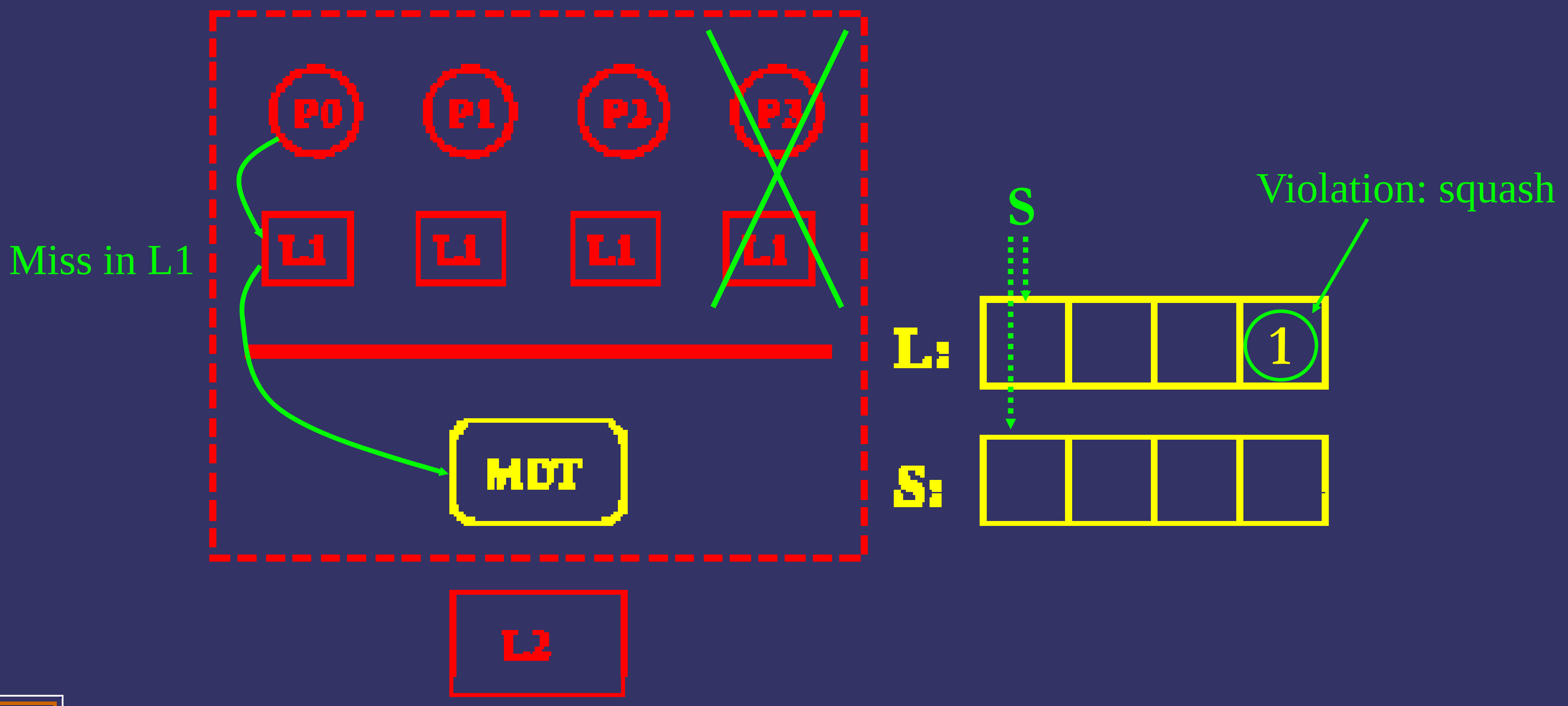
Handling Data Dependences: Loads

- On a load use S bits to find most up-to-date version



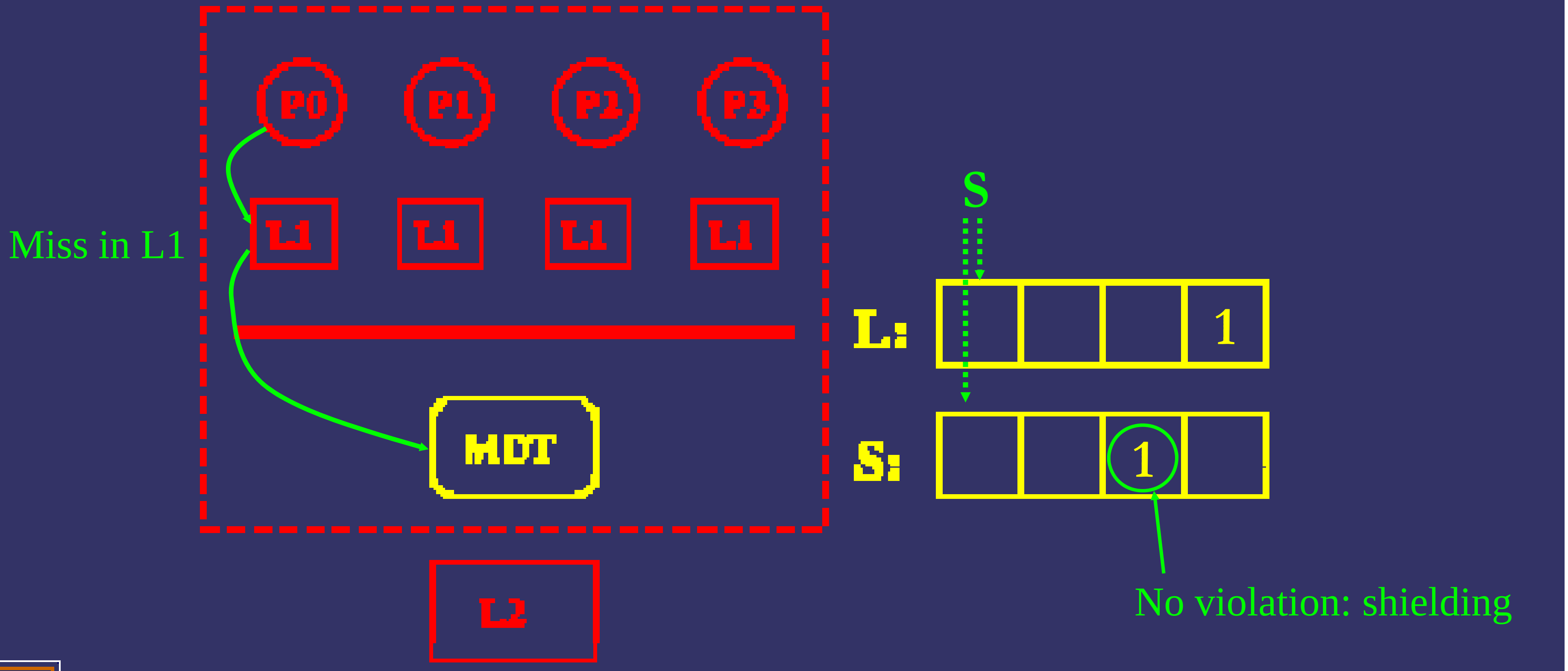
Handling Data Dependences: Stores (I)

- Use L and S bits to detect RAW violations and squash



Handling Data Dependences: Stores (II)

- Use L and S bits to detect RAW violations and squash



Summary of Protocol

- Per-word speculative information
- Multiple versions

Squash?	RAW		WAR		WAW	
	same-word	false	same-word	false	same-word	false
In-order	No	No	No	No	No	No
Out-of-order	Yes	No	No	No	No	No

Speculative CMP

- Storing Versions
  - – L1 maintains versions of speculative threads
  - – L2 maintains only safe versions
  - – Commit: write back dirty versions from L1 to L2
- Static mapping and in-order scheduling of tasks
- L1 and MDT overflows cause stalls

Outline

- Motivation
- Background
- Speculative CMP
- <u>Scalable Speculation</u>
- Evaluation
- Related Work
- Conclusions

Scalable Speculative Multiprocessor

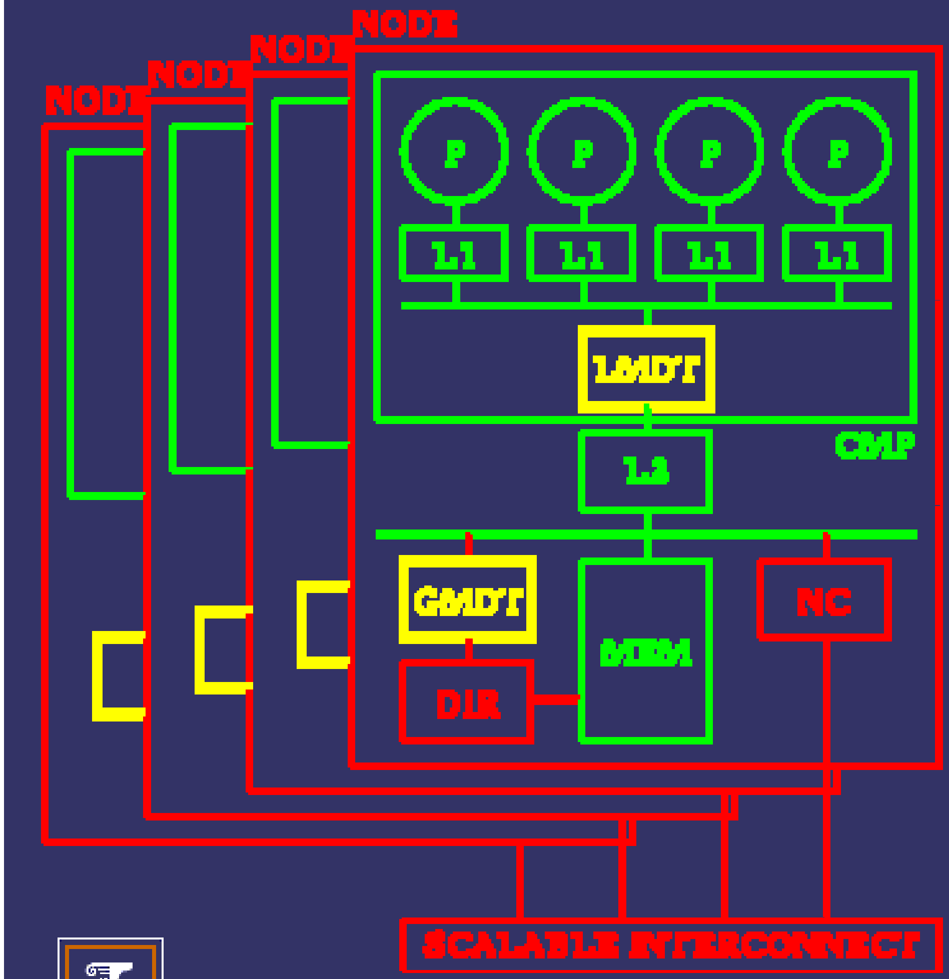
- Goals:

  1. Use largely unmodified speculative CMP in a scalable system

     – trivially defaults for single-processor nodes

  2. Provide simple integration into CC-NUMA

Hierarchical Approach: CMP as Nodes

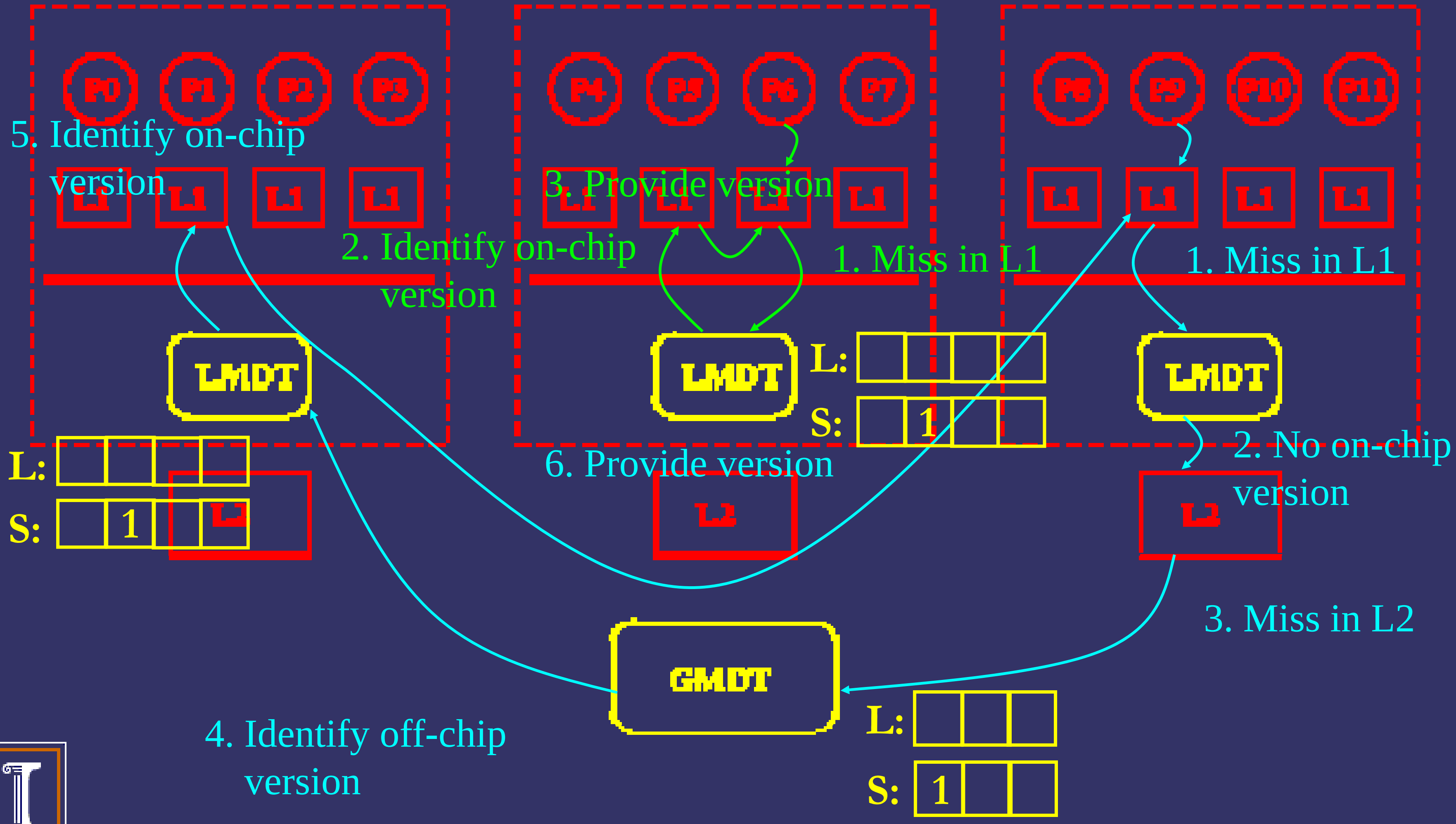


DIR	Directory
NC	Network Controller
LMDT	Local Memory Disambiguation Table
GMDT	Global Memory Disambiguation Table

Hierarchy

		Node	*System*
Versions	*Spec*	L1	L2
	Safe	L2	Mem
Commits		Processor L1 → L2	Node L2 → Mem
Spec info		LMDT	GMDT
Granularity of info		Processors	Nodes
Mapping of tasks		Thread Static	Chunk Dynamic

Hierarchical Approach: Loads



5. Identify on-chip version

3. Provide version

2. Identify on-chip version

1. Miss in L1

1. Miss in L1

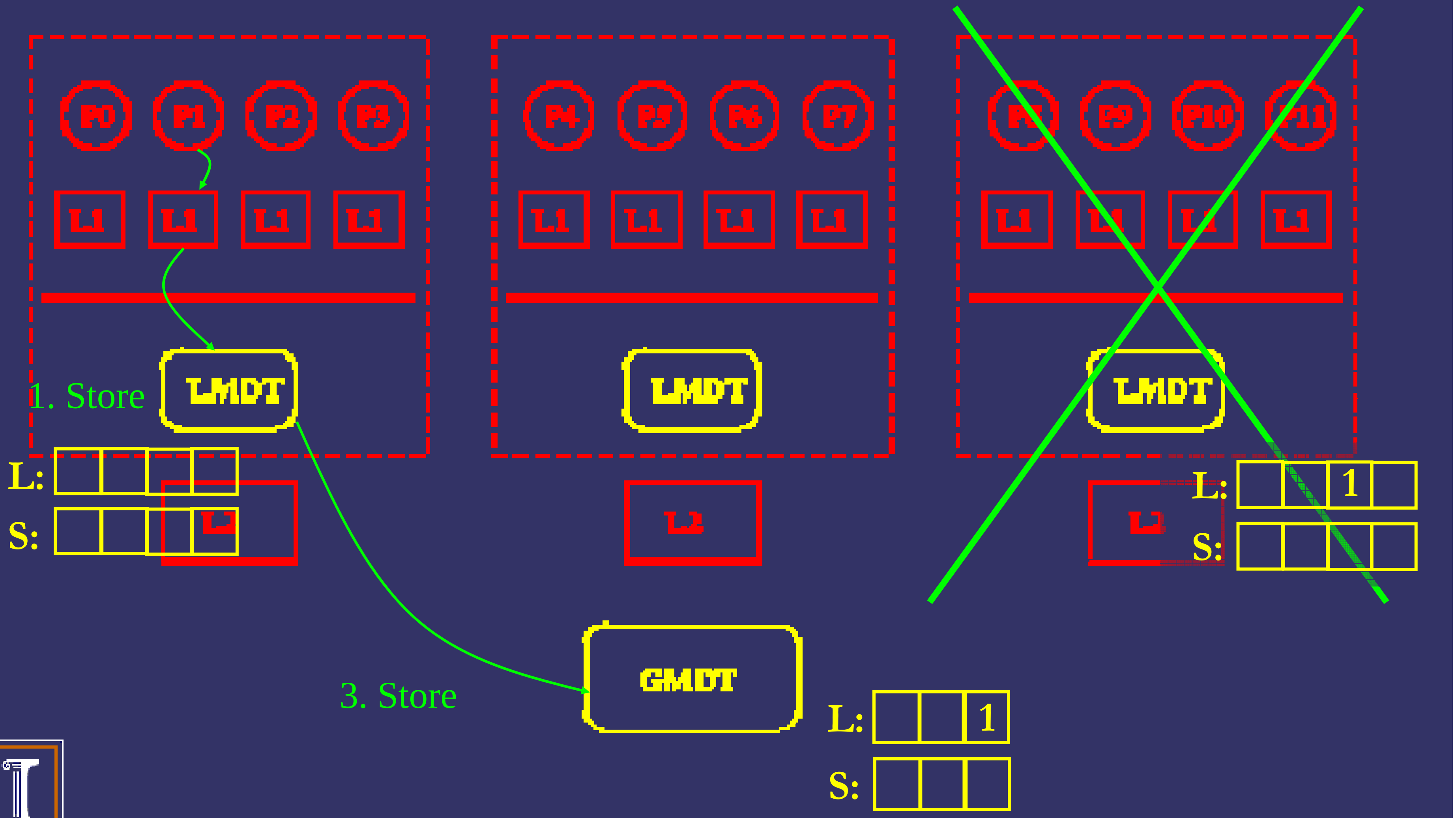
6. Provide version

2. No on-chip version

3. Miss in L2

4. Identify off-chip version

Hierarchical Approach: Stores



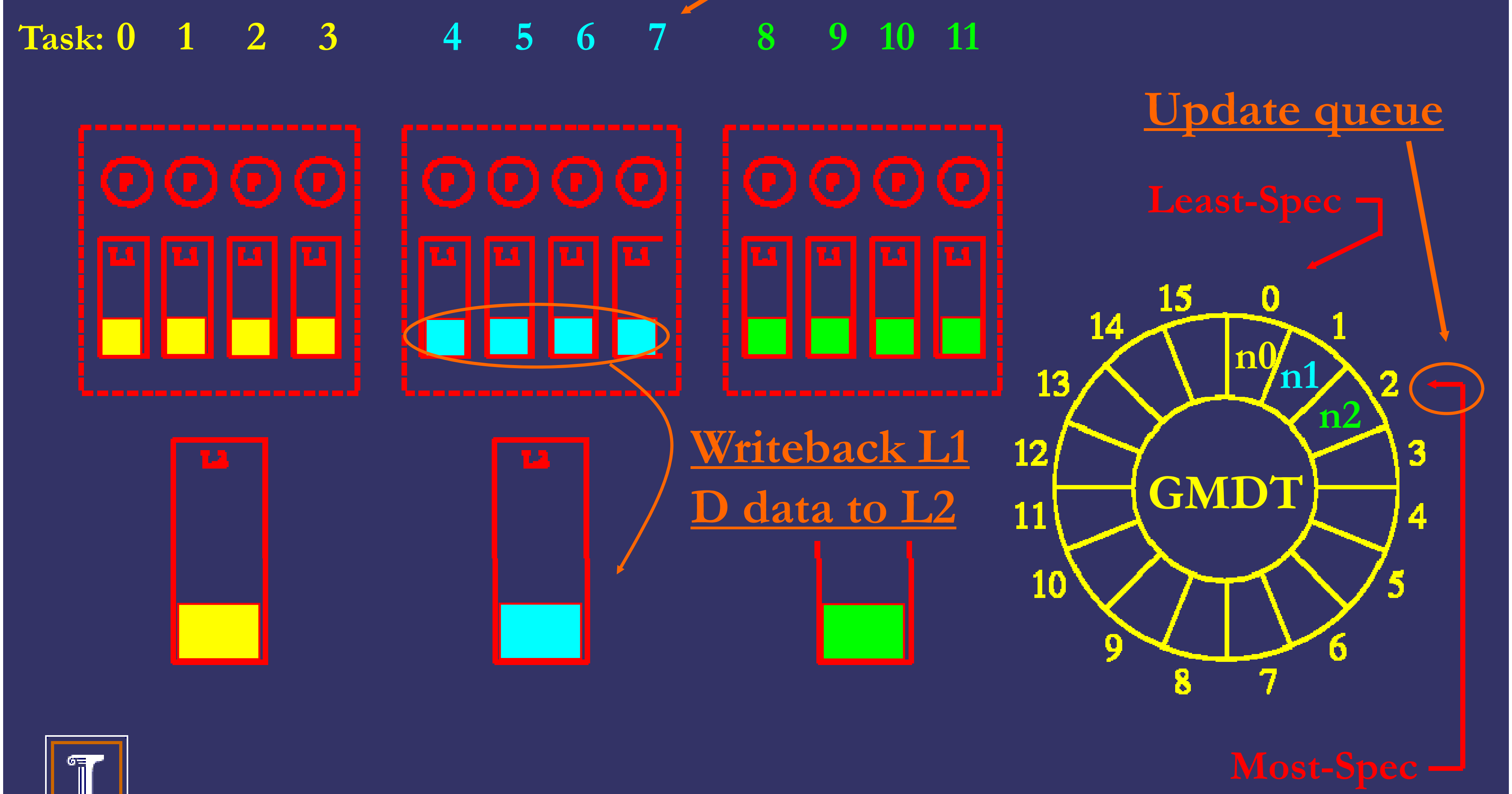
Mapping of Tasks

	Node	*System*
Assignment	**Threads → Processors**	**Chunks → Nodes**
Mapping	**Static**	**Dynamic**
Reason	**Minimize spec CMP modifications**	**Minimize load imbalance**

- Chunk = consecutive set of threads

Mapping of Tasks

Node 1 finishes iterations



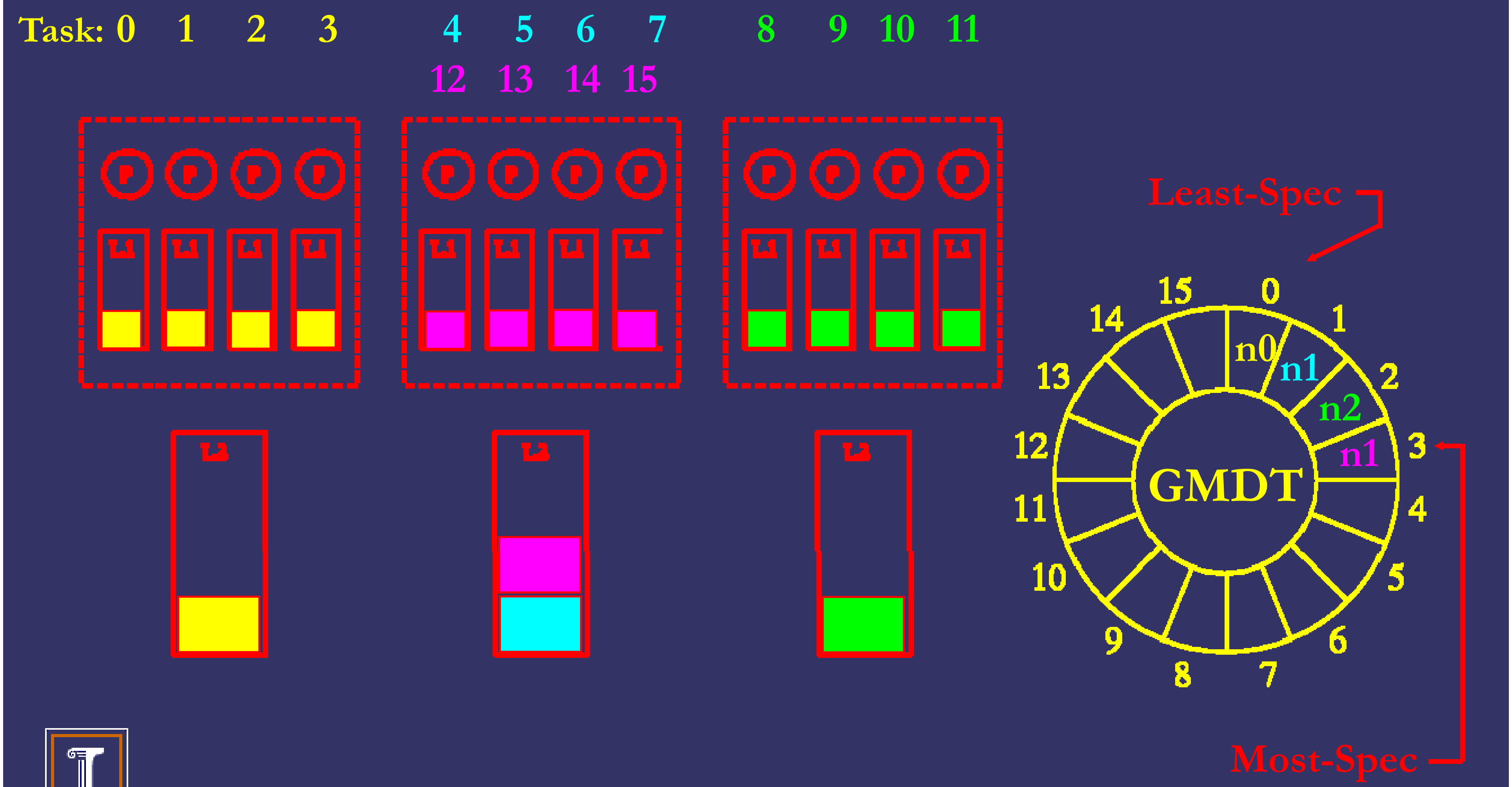
Update queue

Least-Spec

Writeback L1 D data to L2

Most-Spec

Mapping of Tasks



Task: 0 1 2 3 4 5 6 7 8 9 10 11

12 13 14 15

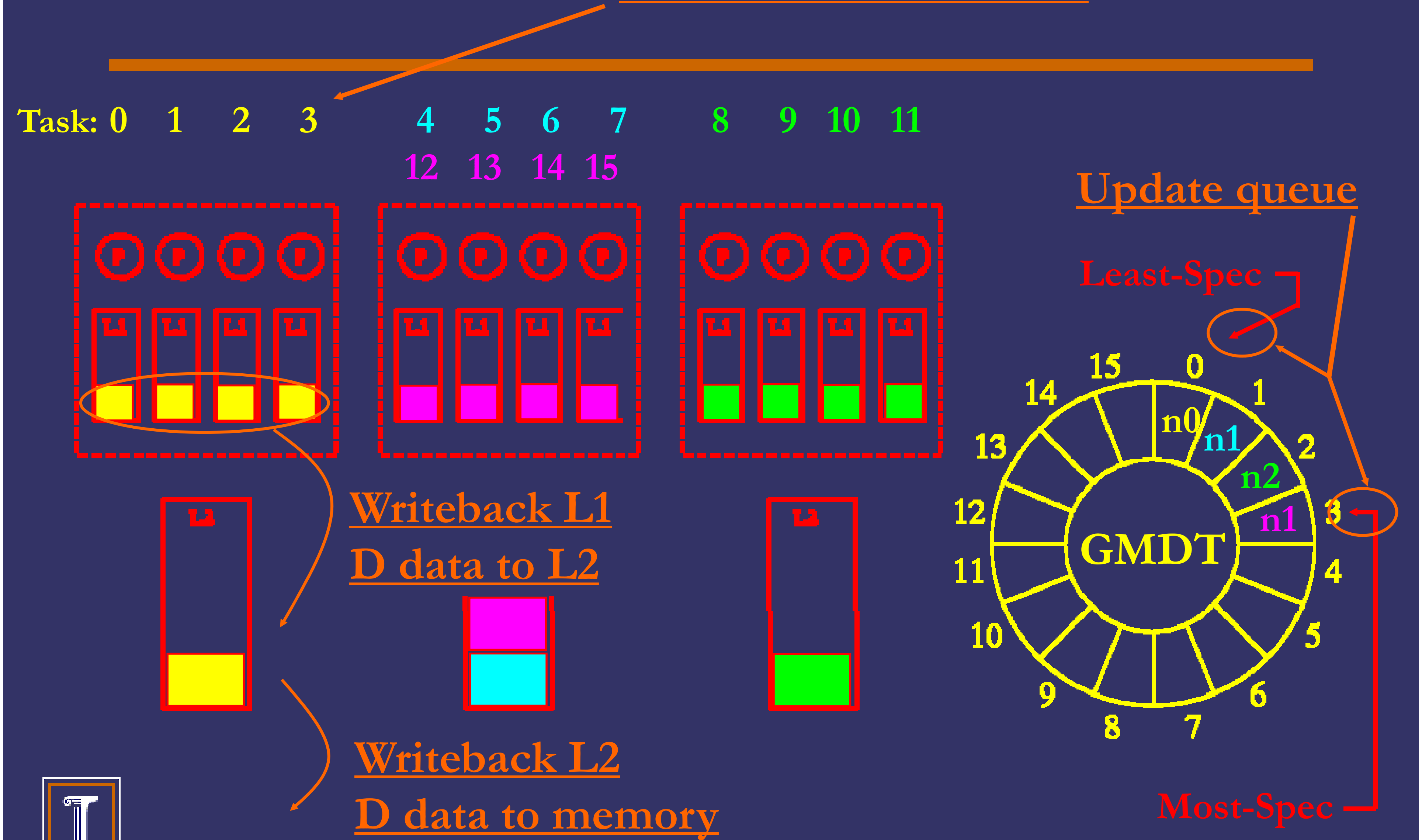
Least-Spec

Most-Spec

GMDT

Node Commits

Node 0 finishes iterations



Update queue

Least-Spec

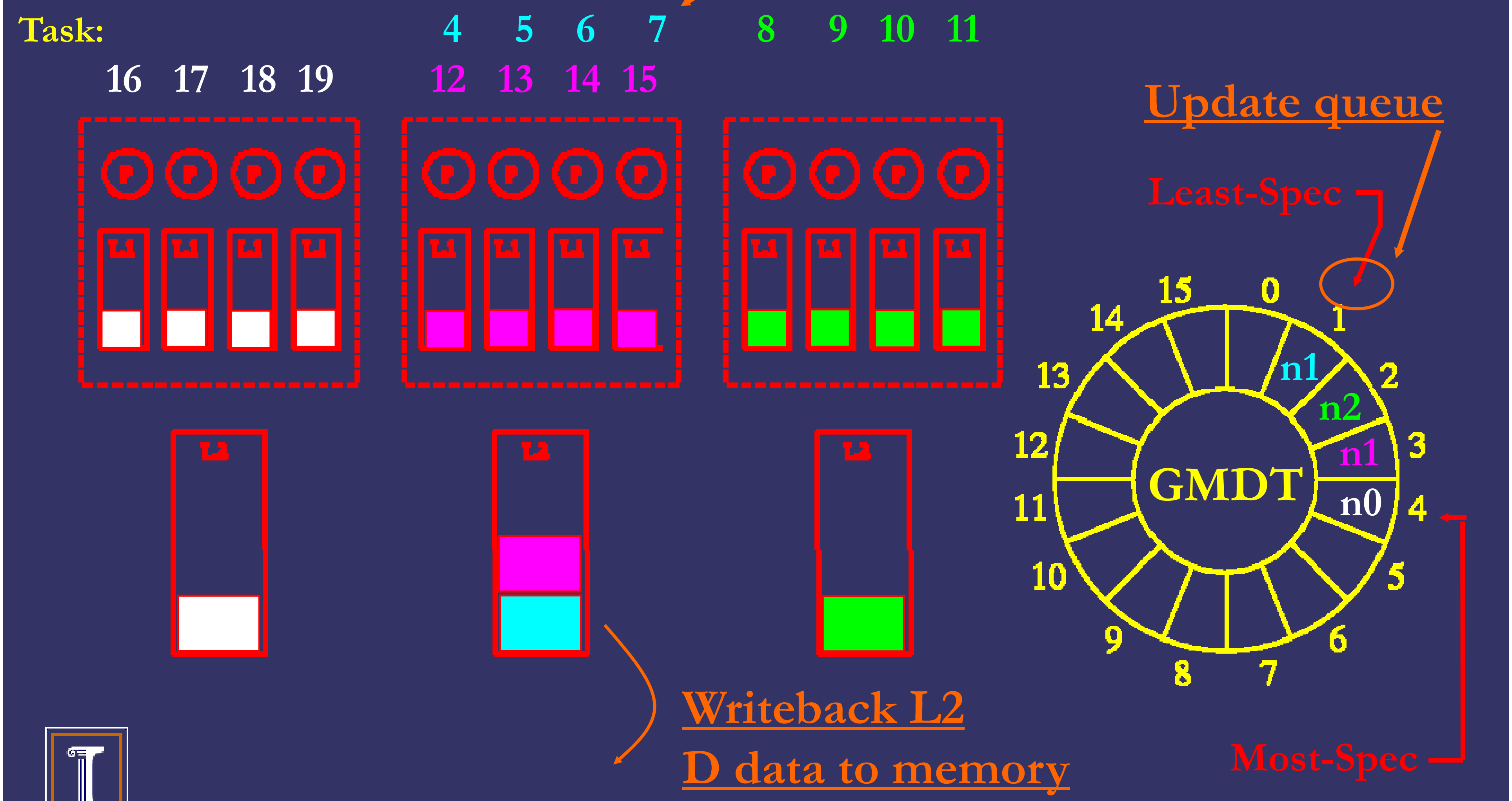
Most-Spec

Writeback L1
D data to L2

Writeback L2
D data to memory

Node Commits

Node 1 has already finished chunk 1



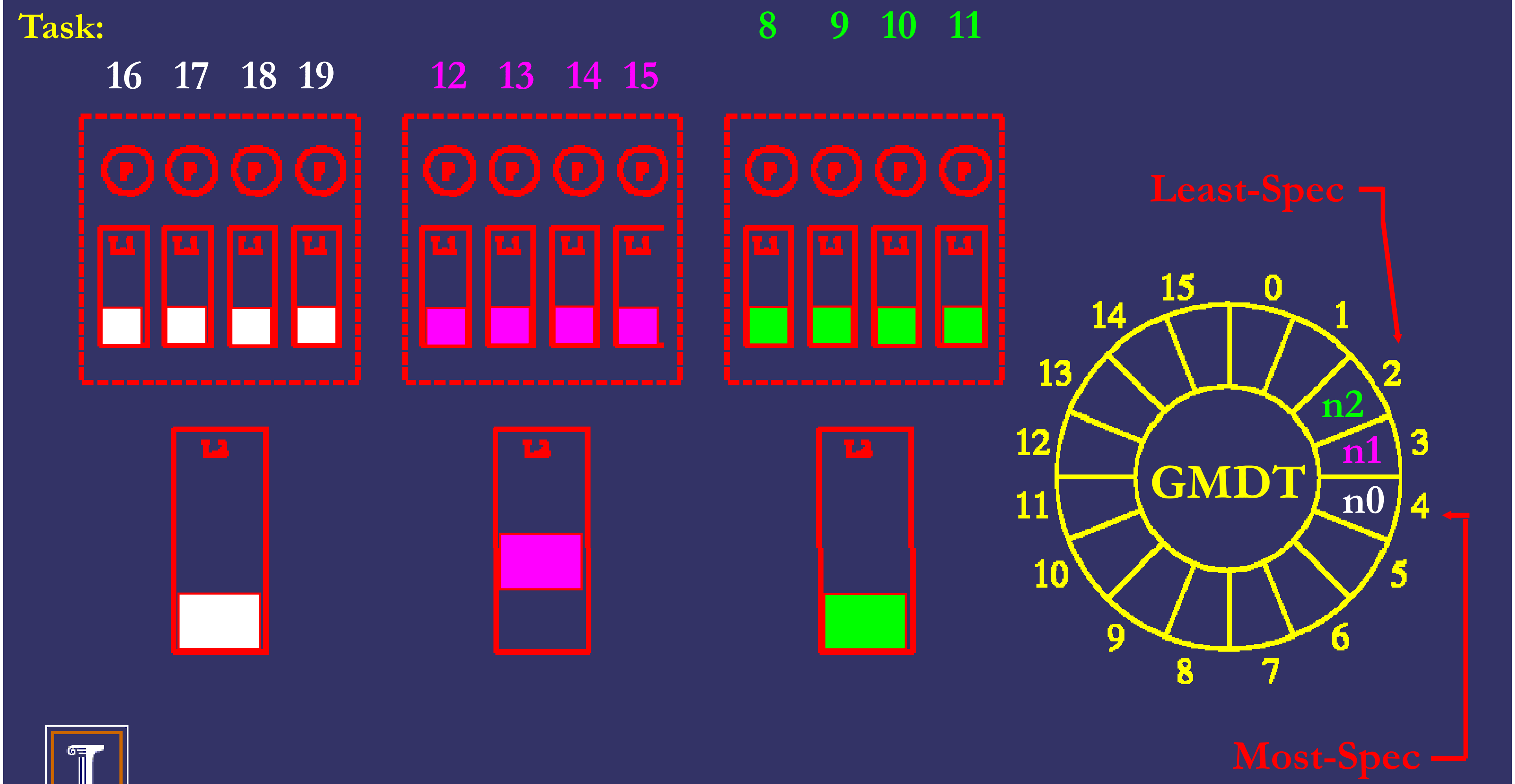
Update queue

Least-Spec

Most-Spec

Writeback L2 D data to memory

Node Commits



GMDT Features

- + Allows displacement of clean speculative data from caches
  - 30% faster because of no stall
- + Identify versions to read with a single lookup
- + Selective invalidations with shielding
  - 50% fewer network messages
- + Squash all faulting threads in parallel & restart in parallel
  - 5% faster in a 4 node system
- – Commit and squash require sync of GMDT (not processors)

Outline

- Motivation
- Background
- Speculative CMP
- Scalable Speculation
- Evaluation
- Related Work
- Conclusions

Simulation Environment

- Execution-driven simulation
- Detailed superscalar processor model
- Coherent+speculative memory back-end
- Scalable multiprocessor: 4 nodes

Node: spec CMP + 1M L2 + 2K-entry GMDT

CMP: 4 x (processor + 32K L1) + 512-entry LMDT

Processor: 4-issue, dynamic

Applications

- Applications dominated by non-analyzable loops (subscripted subscripts)
- Track, BDNA (PERFECT)

  APSI (SPECfp95)

  DSMC3D, Euler (HPF2)

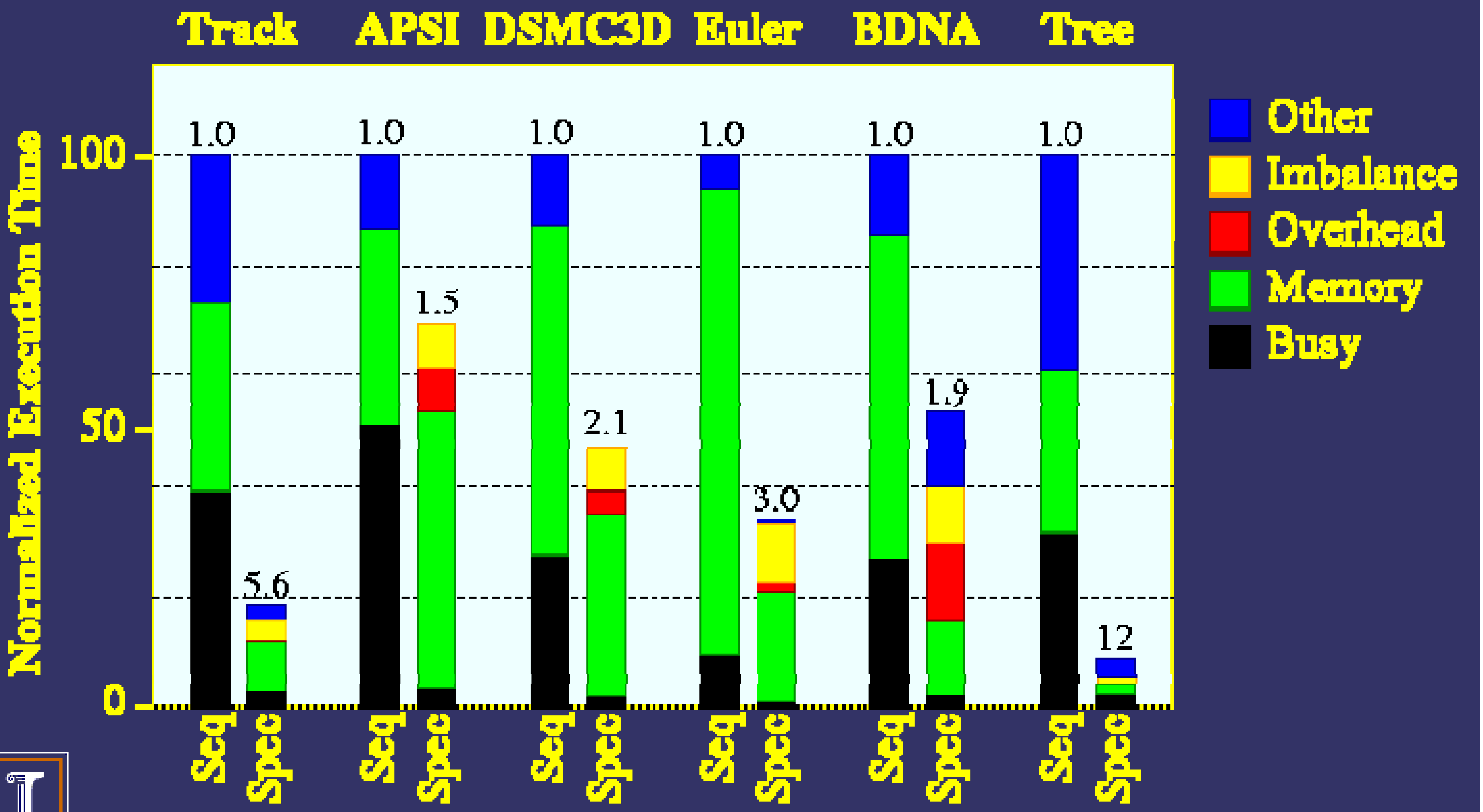
  Tree (Univ. of Hawaii)

Non-analyzable loops take on avg. 51% of sequential time

- Non-analyzable loops and accesses identified by the Polaris parallelizing compiler
- Results shown for the non-analyzable loops only

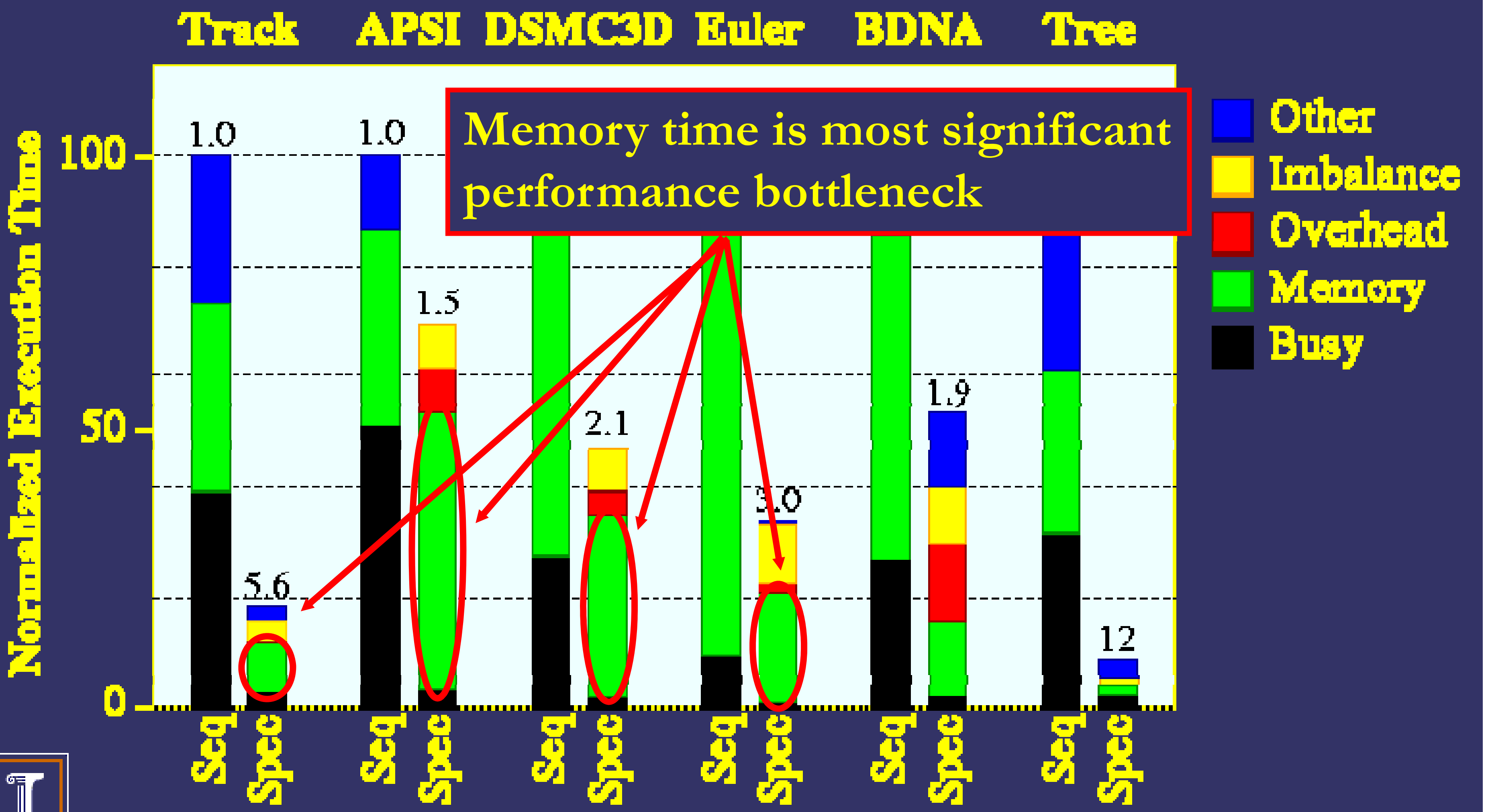
Overall Performance

Avg. speedup=4.4



Overall Performance

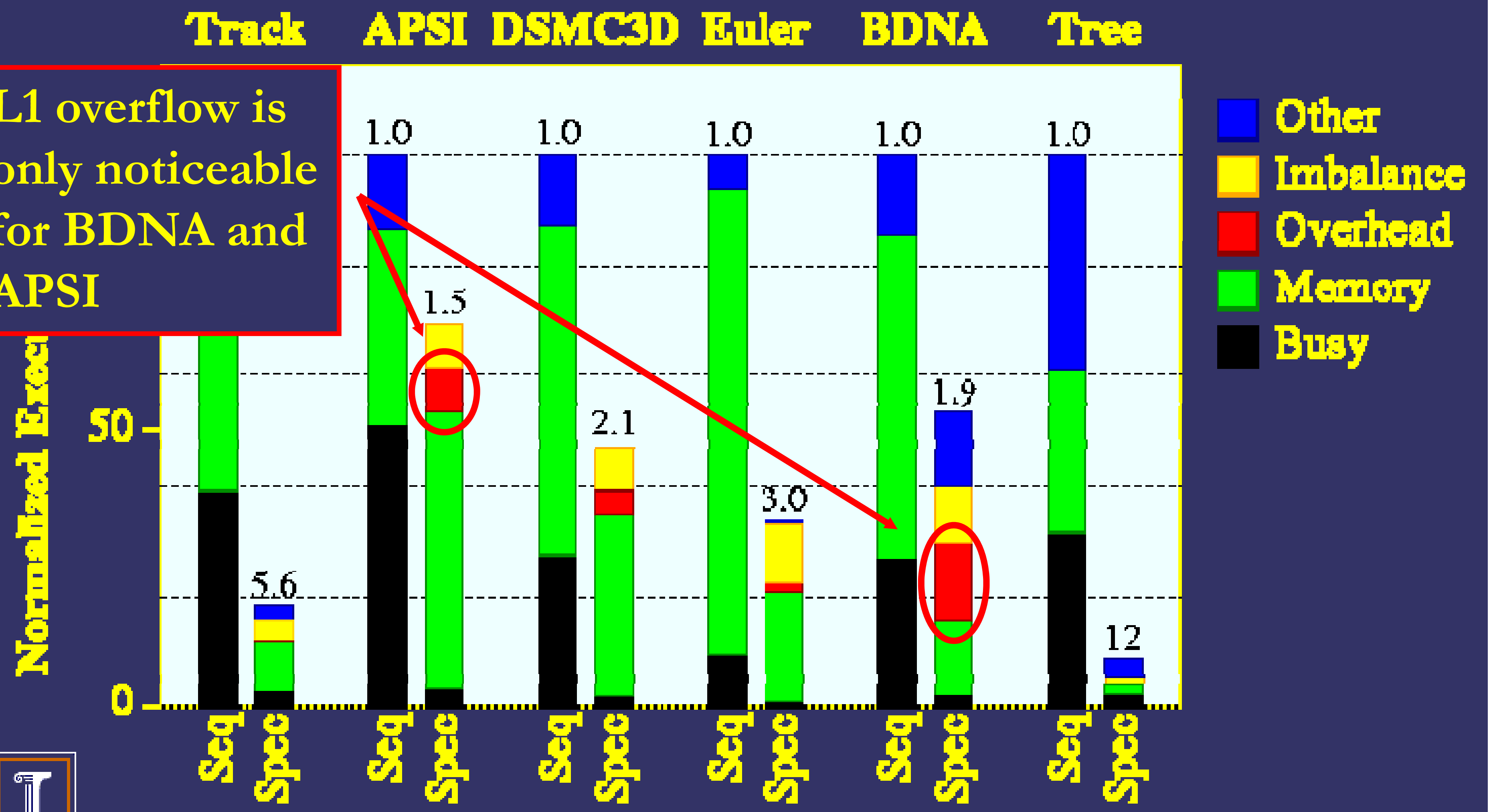
Avg. speedup=4.4



Memory time is most significant performance bottleneck

Overall Performance

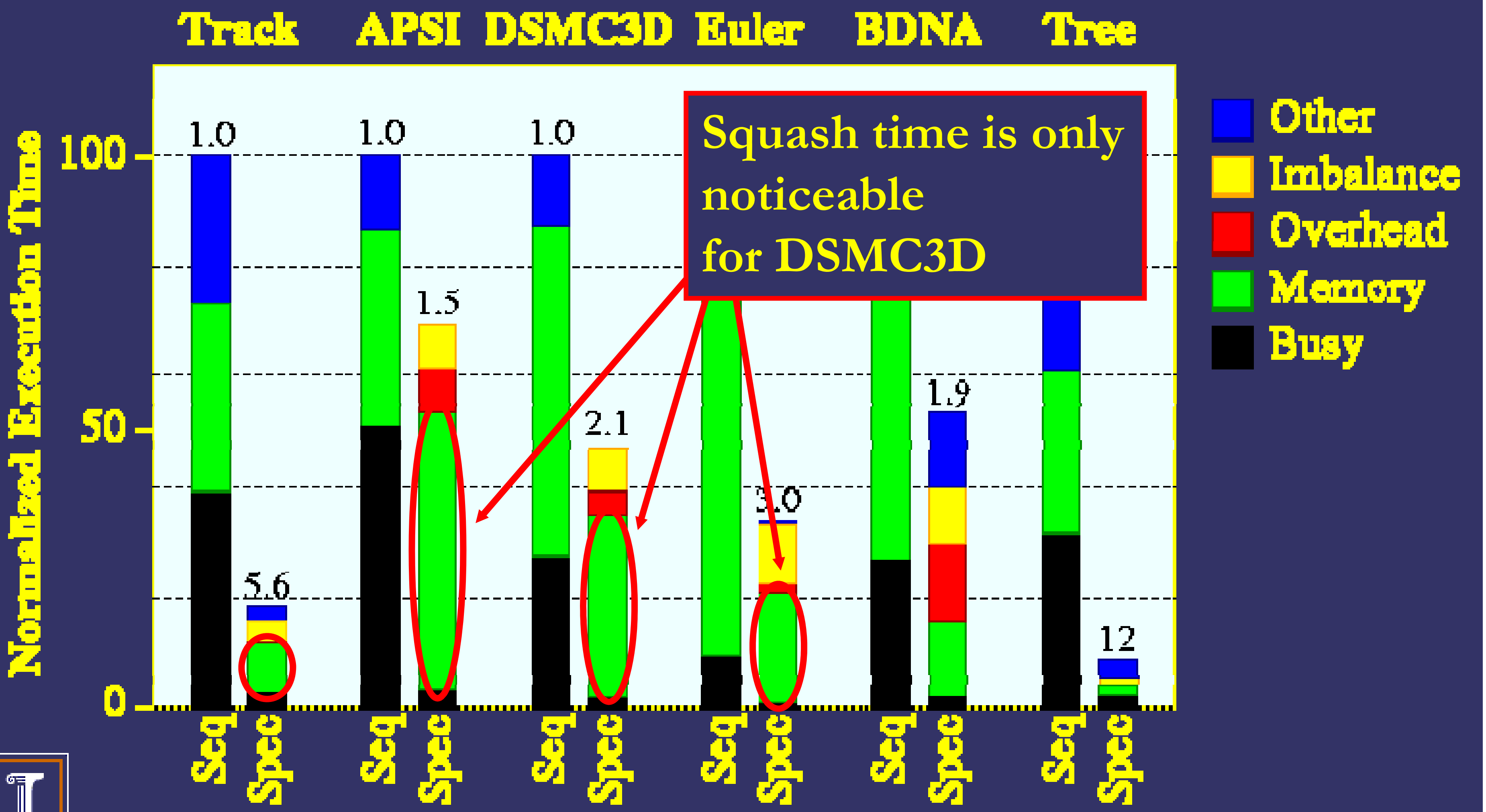
Avg. speedup=4.4



L1 overflow is only noticeable for BDNA and APSI

Overall Performance

Avg. speedup=4.4



Squash time is only noticeable for DSMC3D

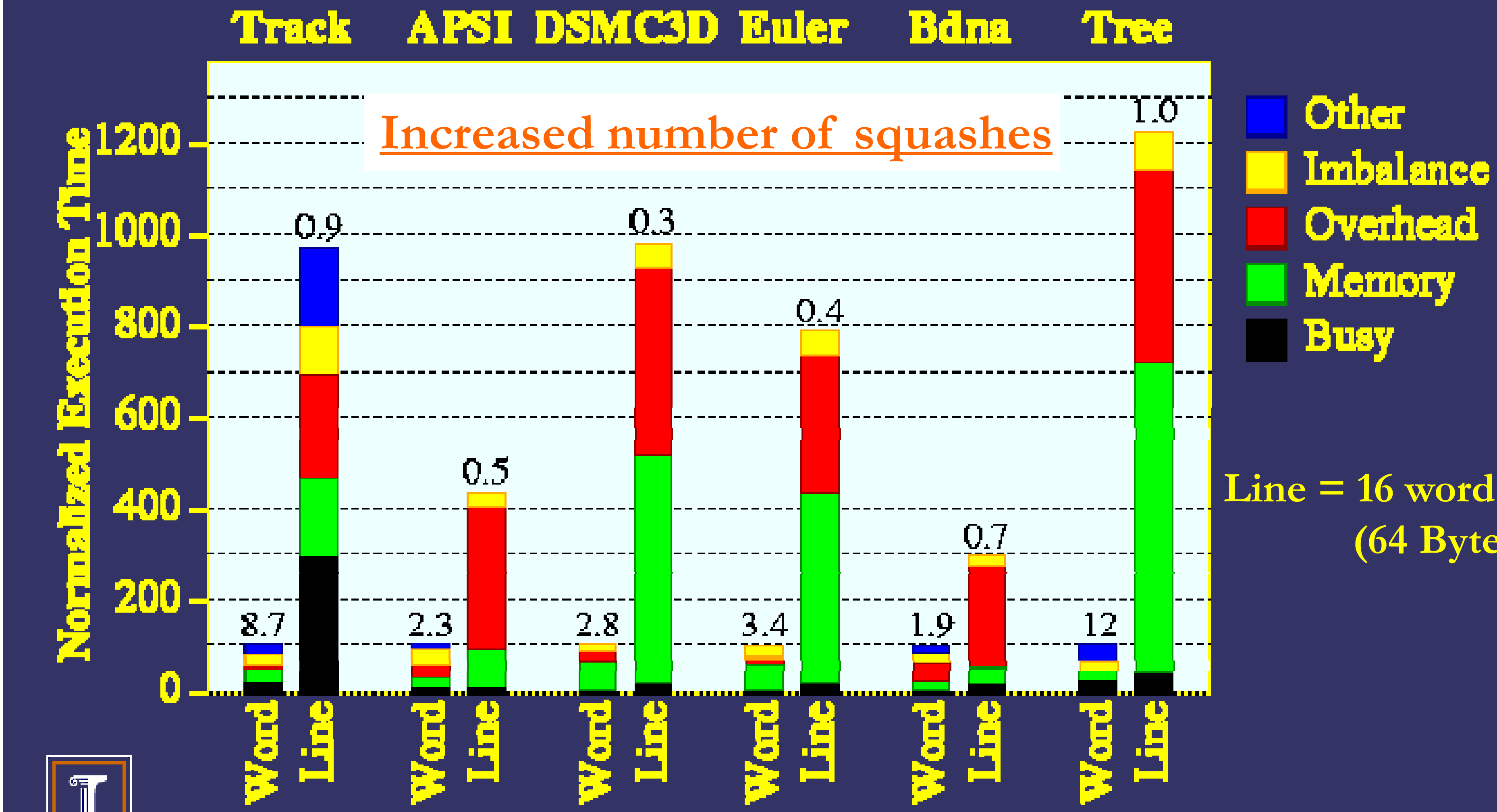
Loop Unrolling

Base: 1 iteration per processor

Unrolling: 2 and 4 iterations per processor

- Increased performance: avg. speedup of 5.2
- Reduction in memory time
- Increase in L1 overflows

Granularity of Speculative State



Increased number of squashes

Line = 16 words
(64 Bytes)

Related Work (I)

- CMP schemes:

  Multiscalar (Wisconsin), TLDS (CMU), Hydra (Stanford), MDT (Illinois), Superthreaded (Minnesota), Speculative Multithreading (UPC)

  – designed for tightly-coupled systems: not scalable

Related Work (II)

- Scalable schemes:

  TLDS (CMU), Zhang et al. (Illinois):
  - Speculative state dispersed along with data
  - Flat view of processors
  - Zhang et al: More sophisticated (handles reduction, load imbalance, large working sets) but complex

Conclusions

- Extended speculative parallelization to scalable system
- Integrated largely unmodified speculative CMP
  - trivially defaults for single-processor nodes
- Promising results: speedup of 5.2 for 16 processors
- Need to support per-word speculative state to avoid excessive squashes

Architectural Support for Scalable Speculative Parallelization in Shared-Memory Multiprocessors

Marcelo Cintra, José F. Martínez, Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign

Cross Iteration Dependences

Application	*Parameter (Average)*	*RAW*		*WAR*		*WAW*	
		Same Word	*False*	*Same Word*	*False*	*Same Word*	*False*
Track	*Number*	0.1	4,869	0.1	47	0	4,880
	Distance	1.0	1.6	1.0	3.1	0	1.6
APSI	*Number*	0	0	95,232	333,312	95,232	333,312
	Distance	0	0	1.0	1.0	1.0	1.0
DSMC3D	*Number*	147,390	9,350,766	102,912	509,315	85,343	8,939,798
	Distance	2,640	225	260,051	228,047	2,608	89
Euler	*Number*	0	104,066	0	0	0	104,066
	Distance	0	415	0	0	0	415
BDNA	*Number*	0	0	32,422	48,518	998,500	1,492,510
	Distance	0	0	1.0	1.0	1.0	1.0

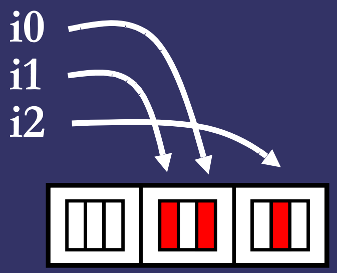
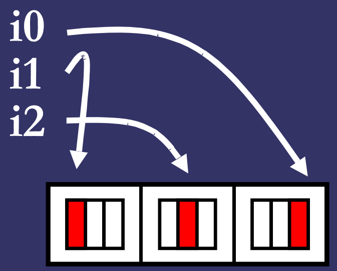
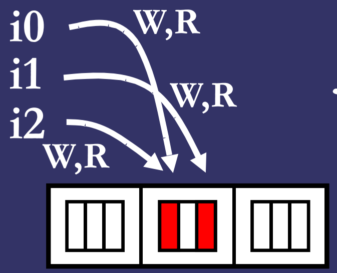
False: **dependence between different words of the same cache line**

GMDT Features

+ Allows displacement of clean speculative data from caches
  - 30% faster because of no stall
+ Identify versions to read with a single lookup
+ Selective invalidations with shielding
  - 50% fewer network messages
+ Squash all faulting threads in parallel & restart in parallel
  - 5% faster in a 4 node system

− Commit and squash require sync of GMDT (not processors)

Application Behavior

Access Pattern	*Multiple Versions*	*Per-Word State*	*Appl.*
 *1. Random, clustered*	**Yes**	**Yes**	**Track, DSMC3D**
 *2. Random, sparse*	**No, but may suffer more squashes**		**DSMC3D, Euler**
 *3. Often write followed by read*	**Yes**	**Yes**	**APSI, BDNA, Tree**

Minimizing Squashes

Support for		*Squash?*		
multiple versions	*per-word state*			
Y	*Y*	ooo	same-word	RAW
N	*Y*	ooo	same-word	RAW WAR WAW
Y	*N*	ooo	same-word	RAW
			false	WAW

False: **dependence between different words of the same cache line**

Interaction GMDT-Directory

- In general: GMDT operates on speculative data

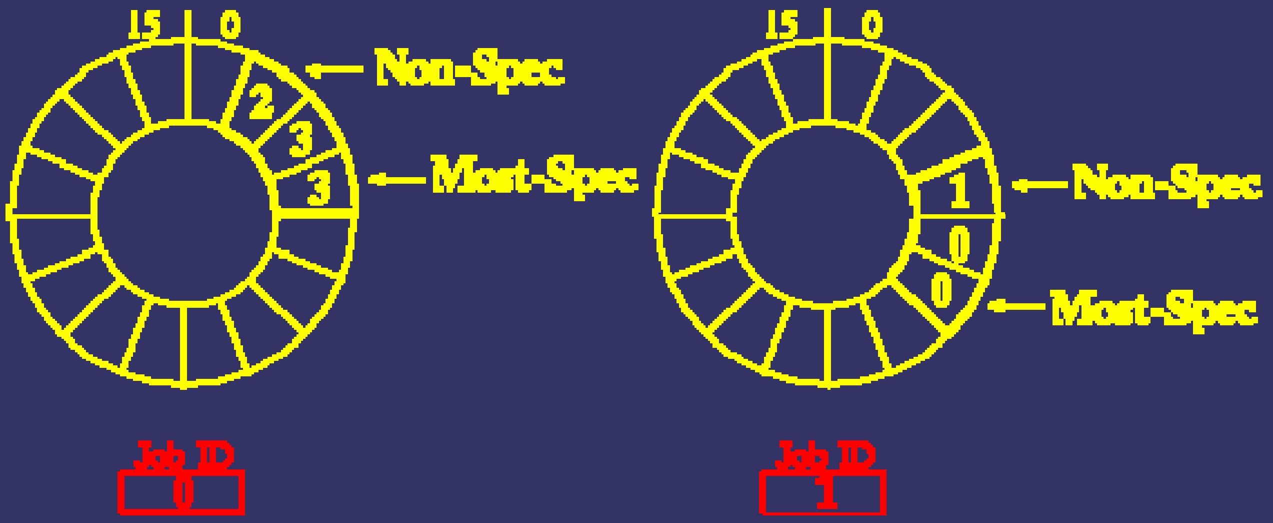
  Dir operates on coherent data

- However: Dir sharing vector is kept up-to-date for speculative data for two reasons:
  - smoothen transition in and out of speculative sections
  - further filter our invalidation messages on speculative stores

Multiprogramming

- Replicate window for each job
- Add job id to GMDT entries

Job ID	Valid	Tag	Word 0	...	Word n-1
0			L0-L15 S0-S15	...	



Simulation Environment

Processor Param.	*Value*
Issue width	4
Instruction window size	64
No. functional units(Int,FP,Ld/St)	3,2,2
No. renaming registers(Int,FP)	32,32
No. pending memory ops.(Ld,St)	8,16

Memory Param.	*Value*
L1,L2,VC size	32KB,1MB,64KB
L1,L2,VC assoc.	2-way,4-way,8-way
L1,L2,VC,line size	64B,64B,64B
L1,L2,VC,latency	1,12,12 cycles
L1,L2,VC banks	2,3,2
Local memory latency	75 cycles
2-hop memory latency	290 cycles
3-hop memory latency	360 cycles
LMDT,GMDT size	512,2K entries
LMDT,GMDT assoc.	8-way,8-way
LMDT,GMDT lookup	4,20 cycles
L1-to-LMDT latency	3 cycles
LMDT-to-L2 latency	8 cycles
Max. active window	8 chunks

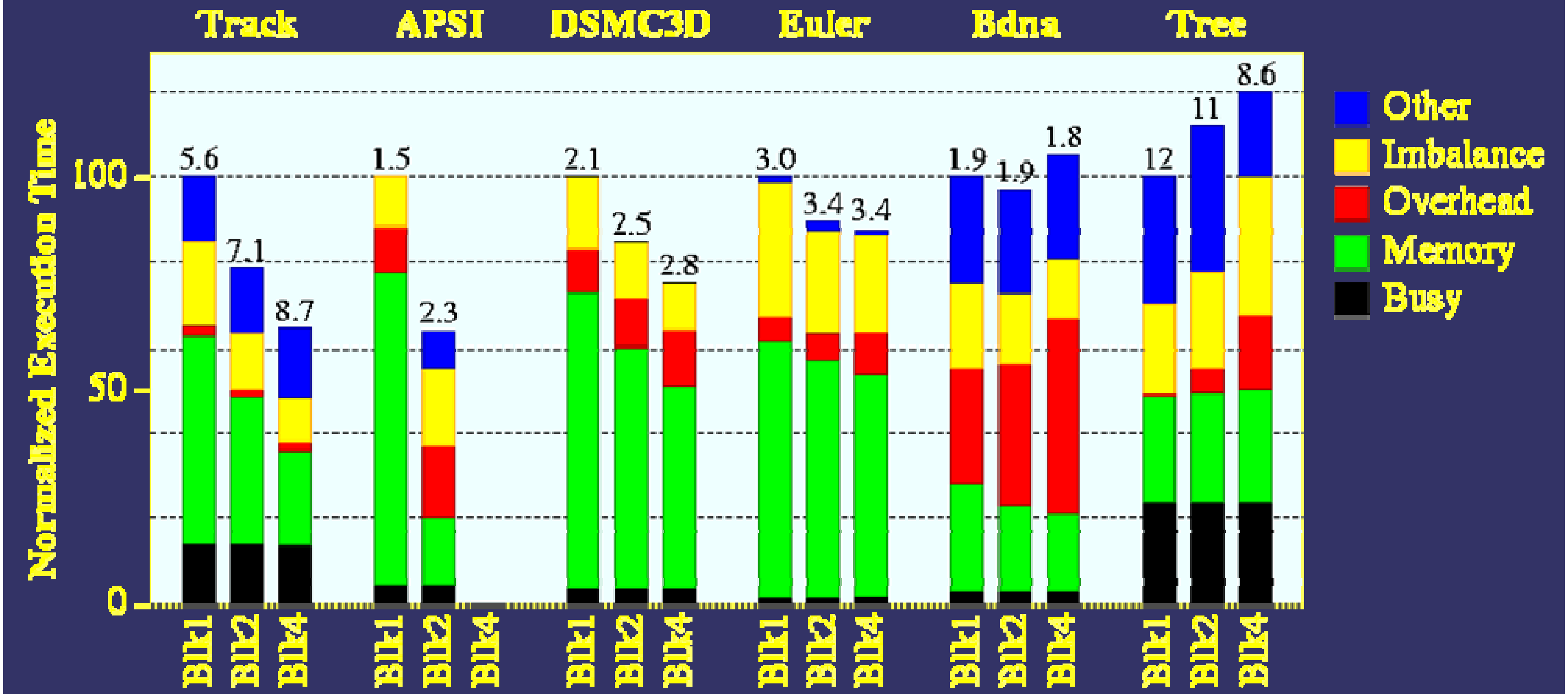
Application Characteristics

Application	*Loops to Parallelize*	*% of Sequential Time*	*Speculative Data (KB)*
Track	nfilt_300	41	240
APSI	run_[20,30,40,60,100]	21	40
DSMC3D	move3_200	33	24767
Euler	dflux_[100,200] eflux_[100,200,300] psmoo_20	90	686
BDNA	actfor_240	32	7
Tree	accel_10	90	1
		average: 51	

- Performance data reported refer to the loops only

Loop Unrolling

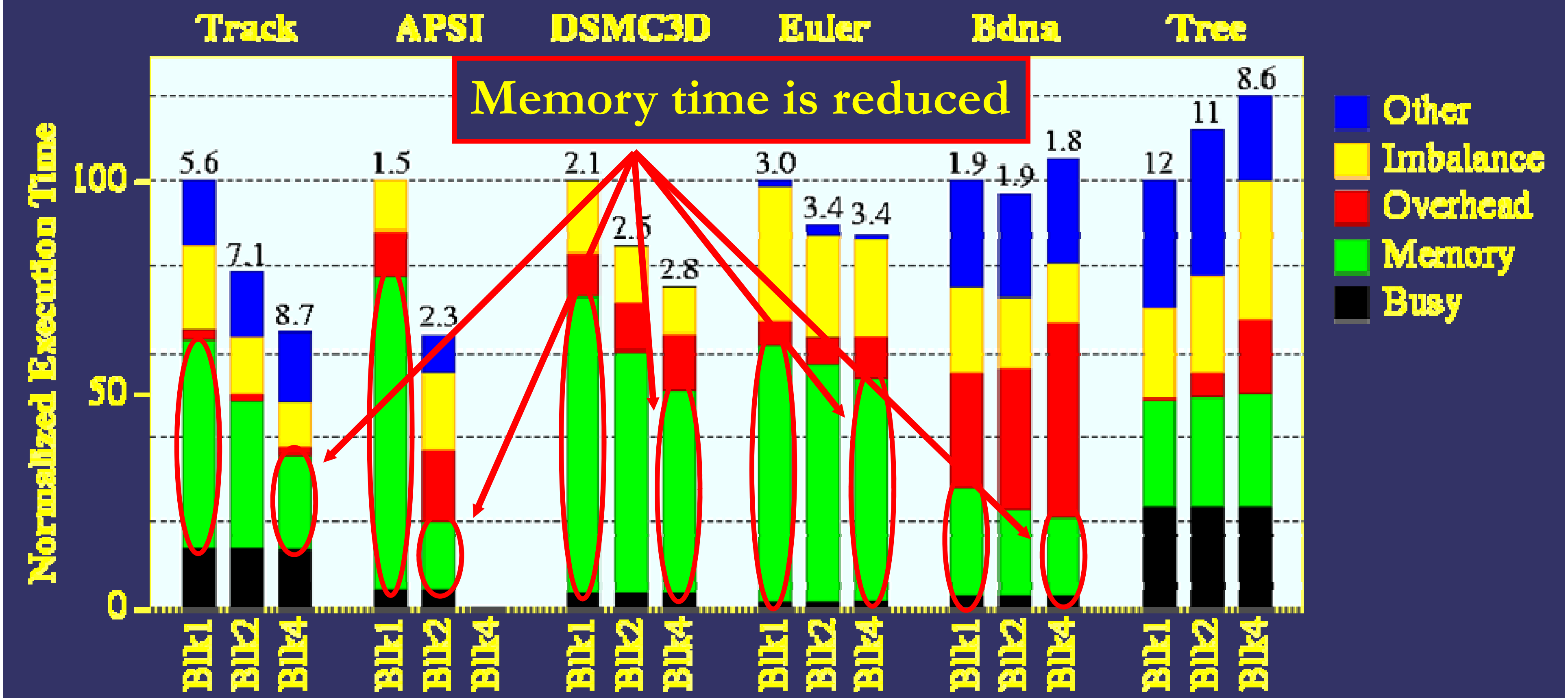
Avg. speedup=4.7 (Blk 2)
4.6 (Blk 4)



Intl. Symp. on Computer Architecture - June 2000

Loop Unrolling

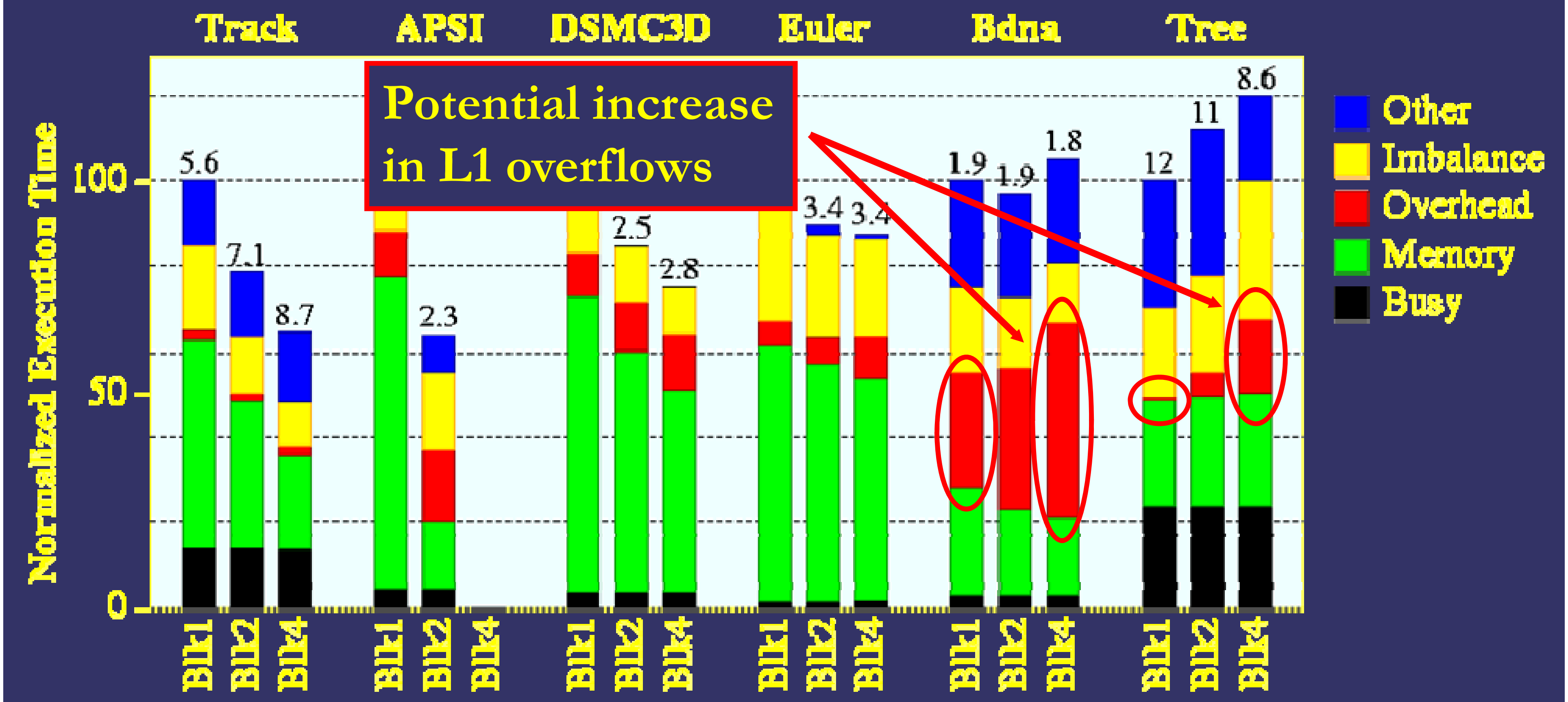
Avg. speedup=4.7 (Blk 2)
4.6 (Blk 4)



Memory time is reduced

Loop Unrolling

Avg. speedup=4.7 (Blk 2)
4.6 (Blk 4)



Potential increase in L1 overflows