Tradeoffs in Buffering Memory State for Thread-Level Speculation in Multiprocessors

María Jesús Garzarán, M. Prvulovic, J. M. Llabería[*], V. Viñals[§], L. Rauchwerger[ψ], and J. Torrellas

U. of Illinois at Urbana-Champaign [§] U. of Zaragoza, Spain

[*] U.P.C. , Spain [ψ] Texas A&M University

Motivation: Speculative State Management under TLS

- Thread-Level Speculation (TLS) extracts parallelism from hard-to-analyze applications
  - Pointers, arrays with subscripted subscripts, …
- Speculative tasks generate *speculative* memory state
- Must buffer and manage this speculative state

Motivation: Speculative State Management under TLS

- *Speculative memory state buffering* varies across TLS schemes:
  - *Where* is the speculative state buffered?
  - *How many* speculative tasks and versions are supported?
  - *How* is the state merged with main memory?

→ **Must understand the performance/complexity tradeoffs**

Our Contributions

- New taxonomy of buffering schemes under TLS
  - Identifies major axes in the design space
- Performance/Complexity tradeoff analysis of schemes
  - Performance benefits
  - Design complexity
  - Application characteristics
- Performance comparison of schemes with a unified framework:
  - Same speculation protocol
  - Same architectural parameters
  - Same applications

Roadmap

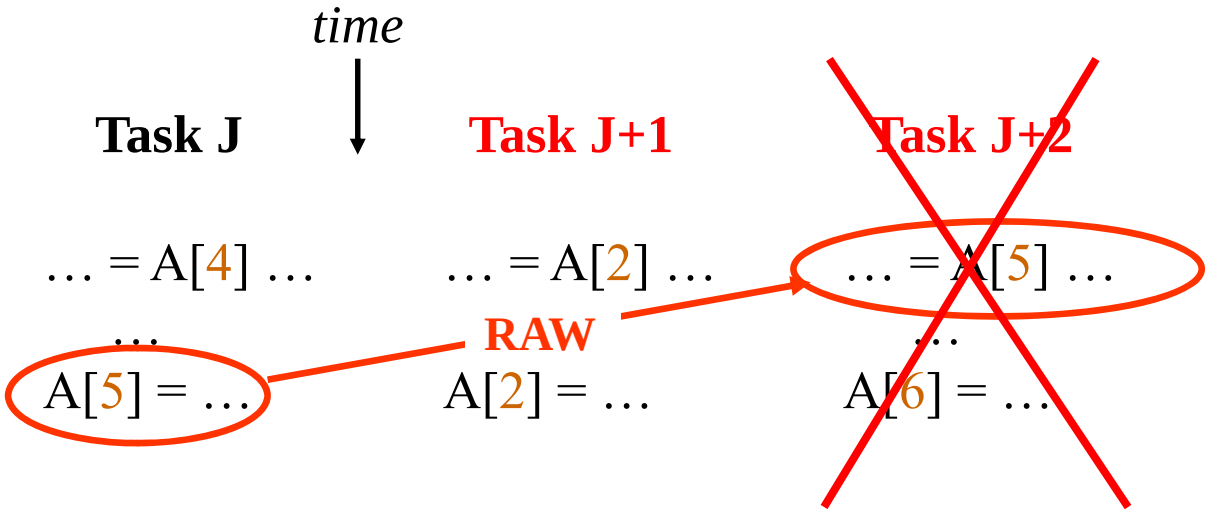
- **Introduction to Thread-Level Speculation (TLS)**
- Taxonomy of Buffering
- Tradeoffs Analysis
- Evaluation
- Conclusions

Thread-Level Speculation (TLS)

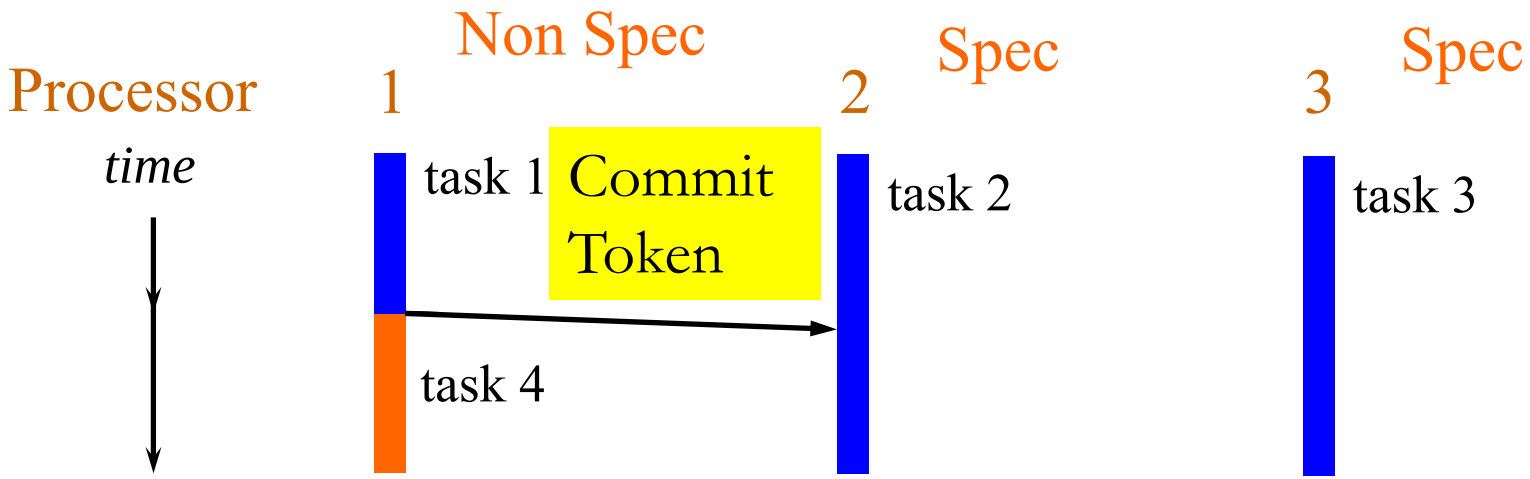
- Execute potentially-dependent tasks in parallel
  - Assume no cross-task dependence will be violated
  - Track memory accesses; buffer unsafe state
  - Detect any dependence violation
  - Squash offending tasks, repair polluted state, restart tasks

Example

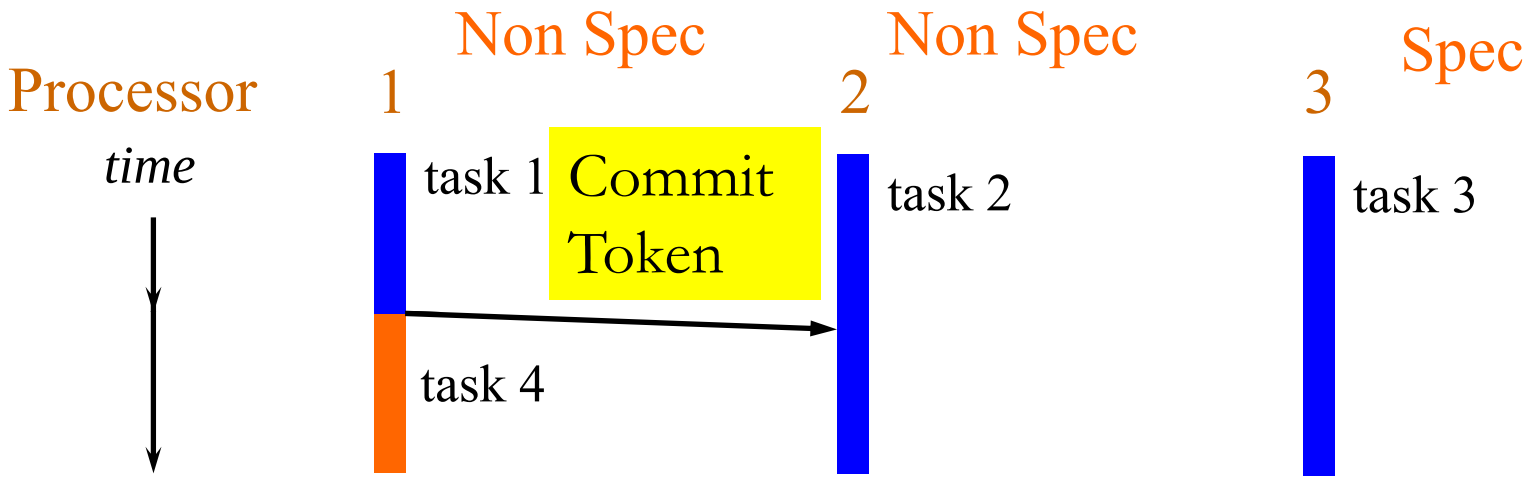
```
for (i=0;i<n;i++){
    … = A[B[i]] …

  A[C[i]] = …
}
```



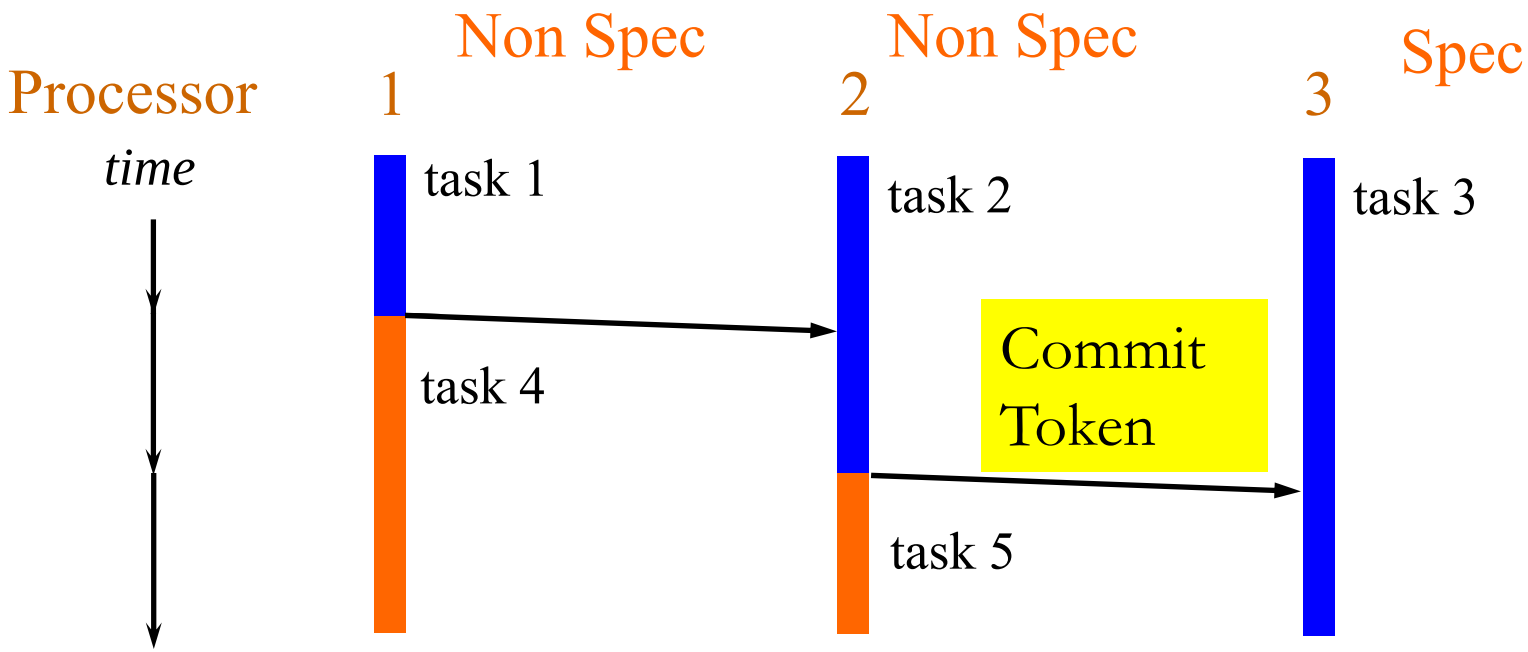
Task Execution under TLS



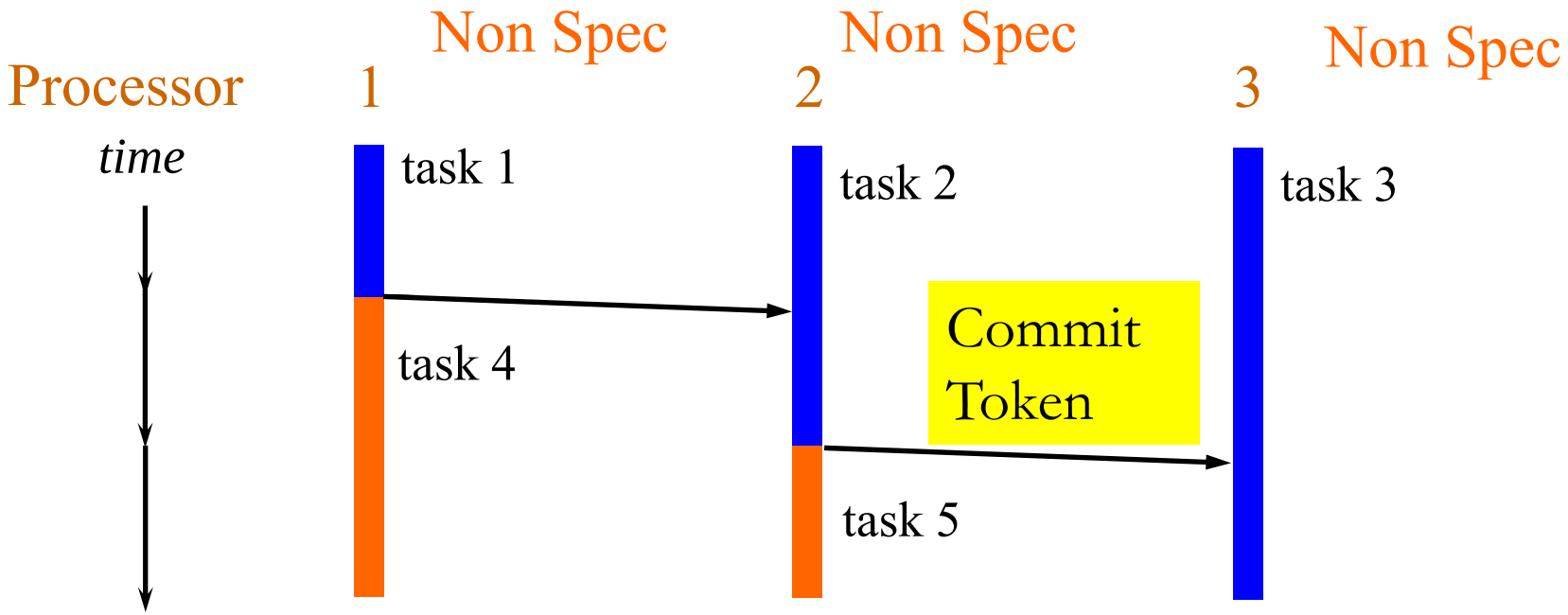
Task Execution under TLS



Task Execution under TLS

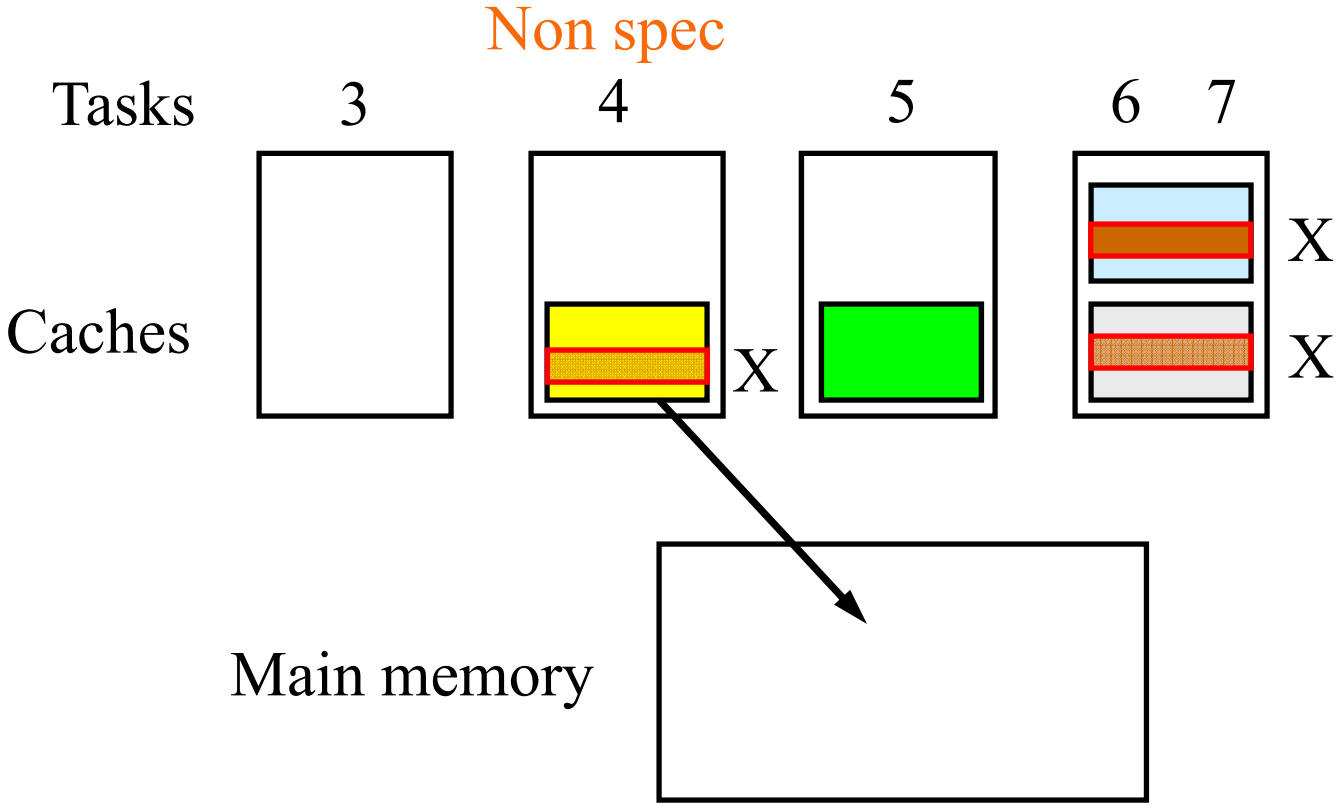


Task Execution under TLS



Challenges in Buffering Speculative State

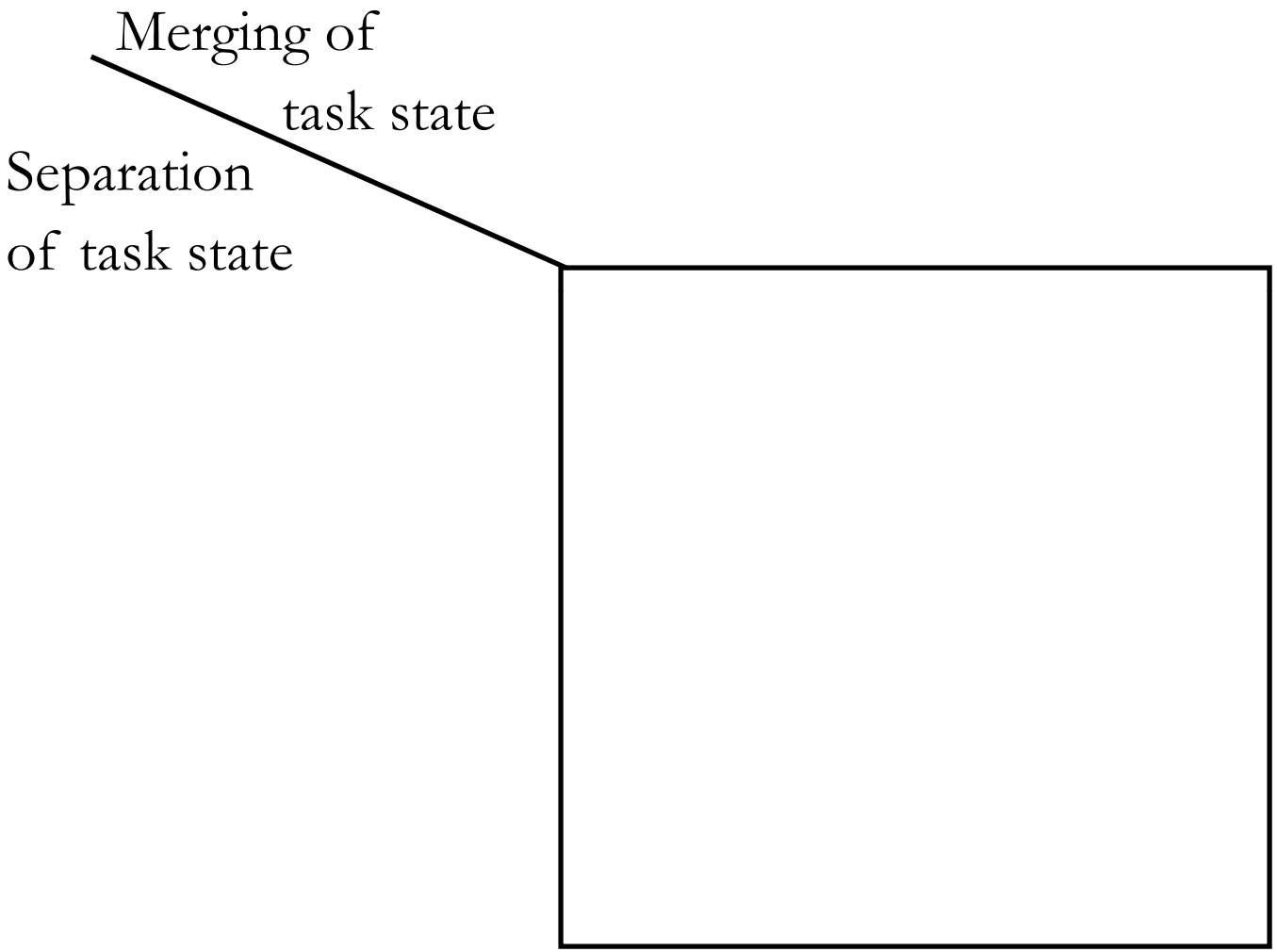
- State needs to be buffered until a task becomes non-speculative
- State must be merged in order



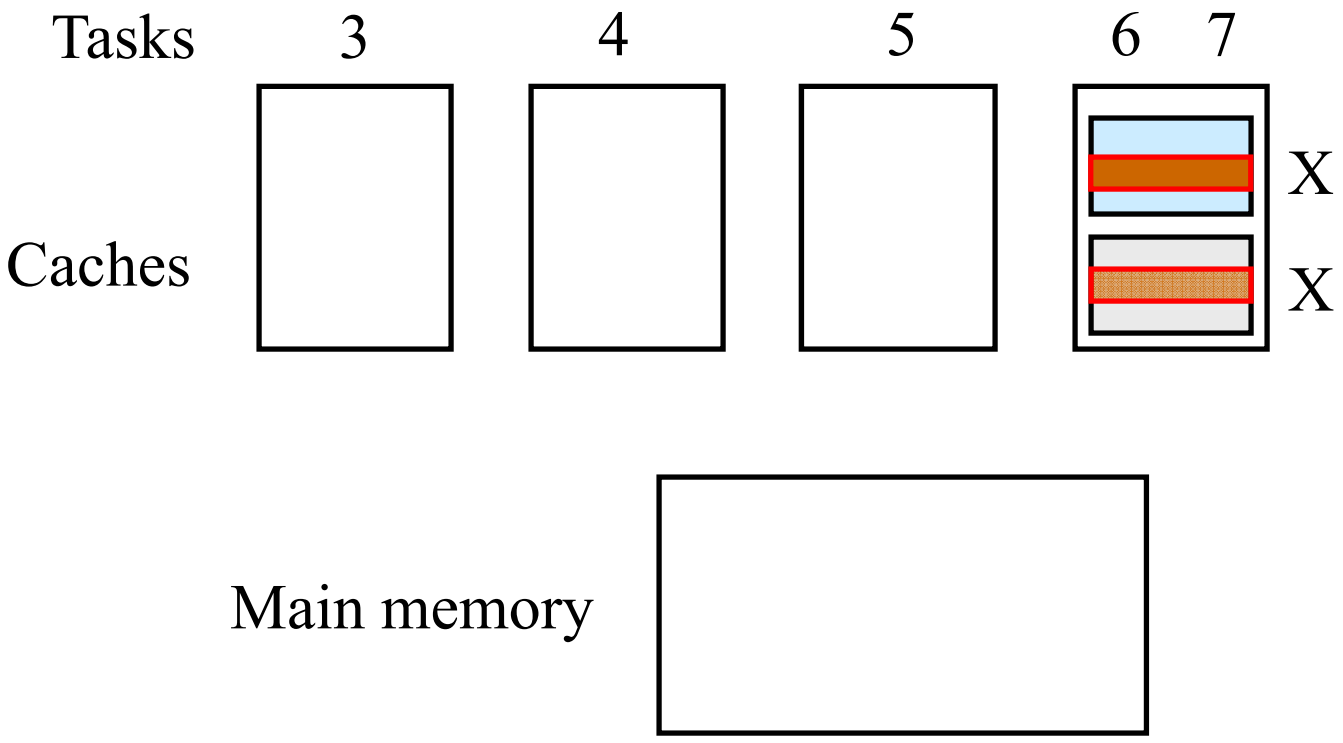
Roadmap

- Introduction to Thread-Level Speculation
- **Taxonomy of buffering**
- Tradeoffs Analysis
- Evaluation
- Conclusions

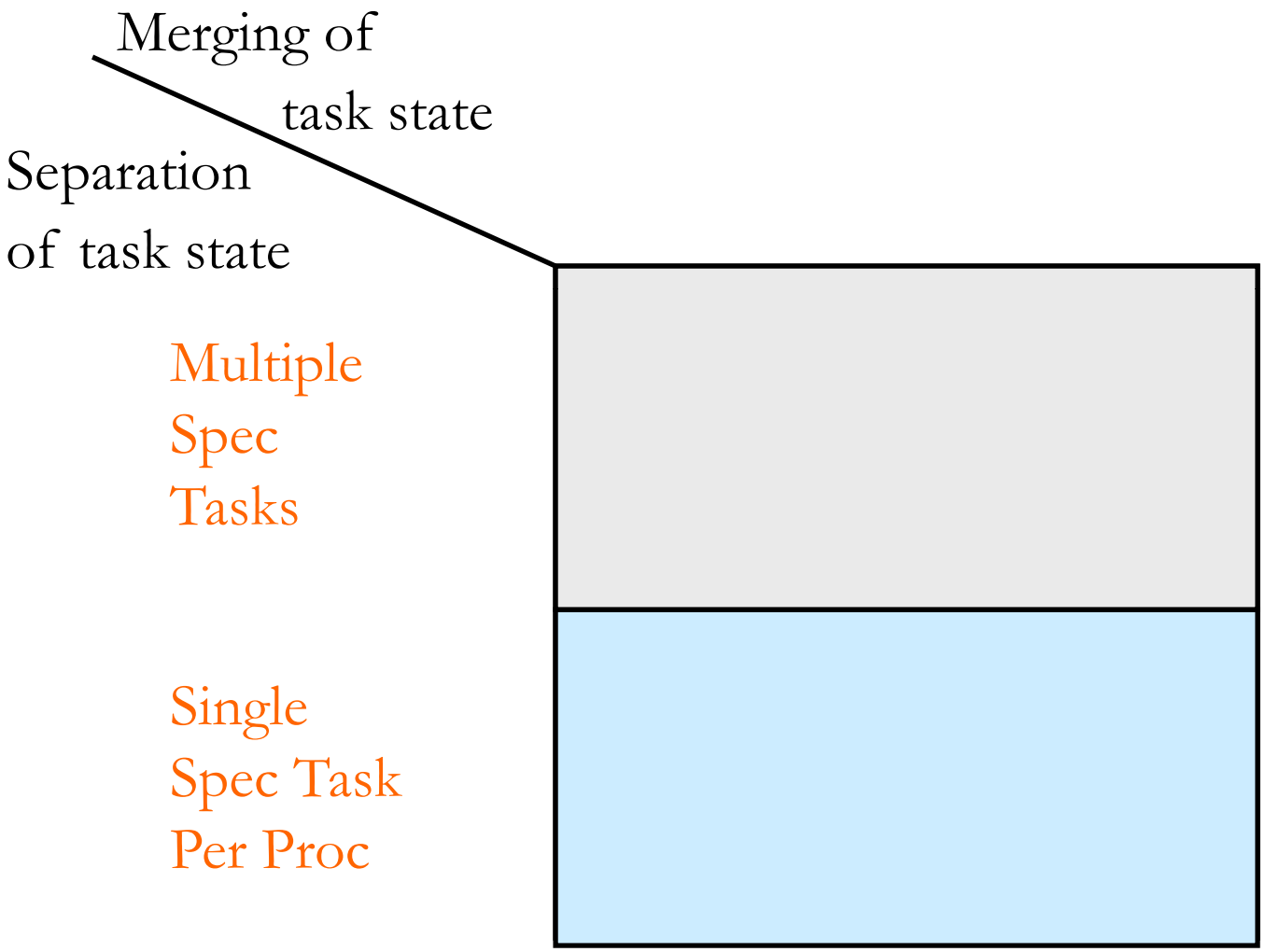
Taxonomy of Buffering



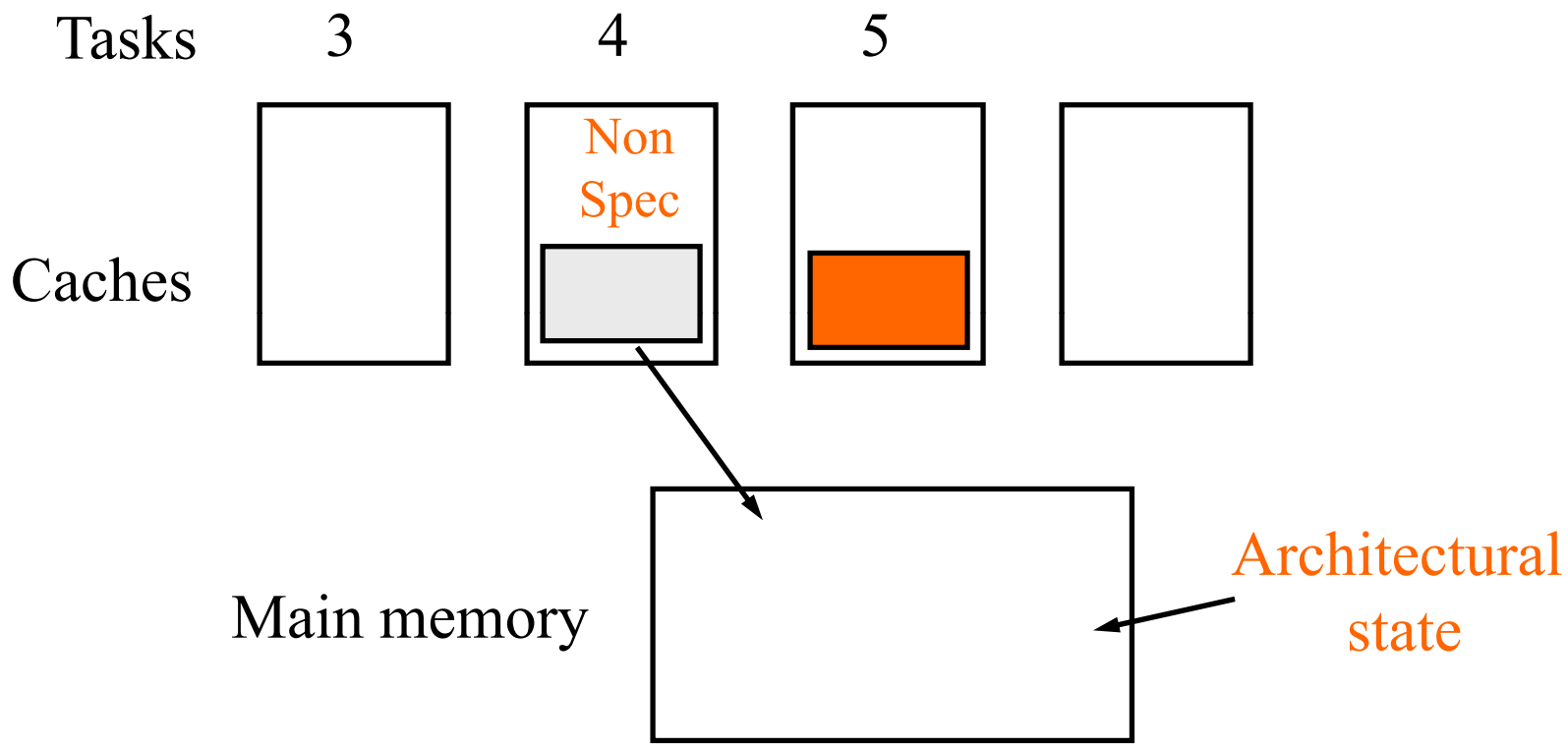
Separation of Task State



Taxonomy of Buffering

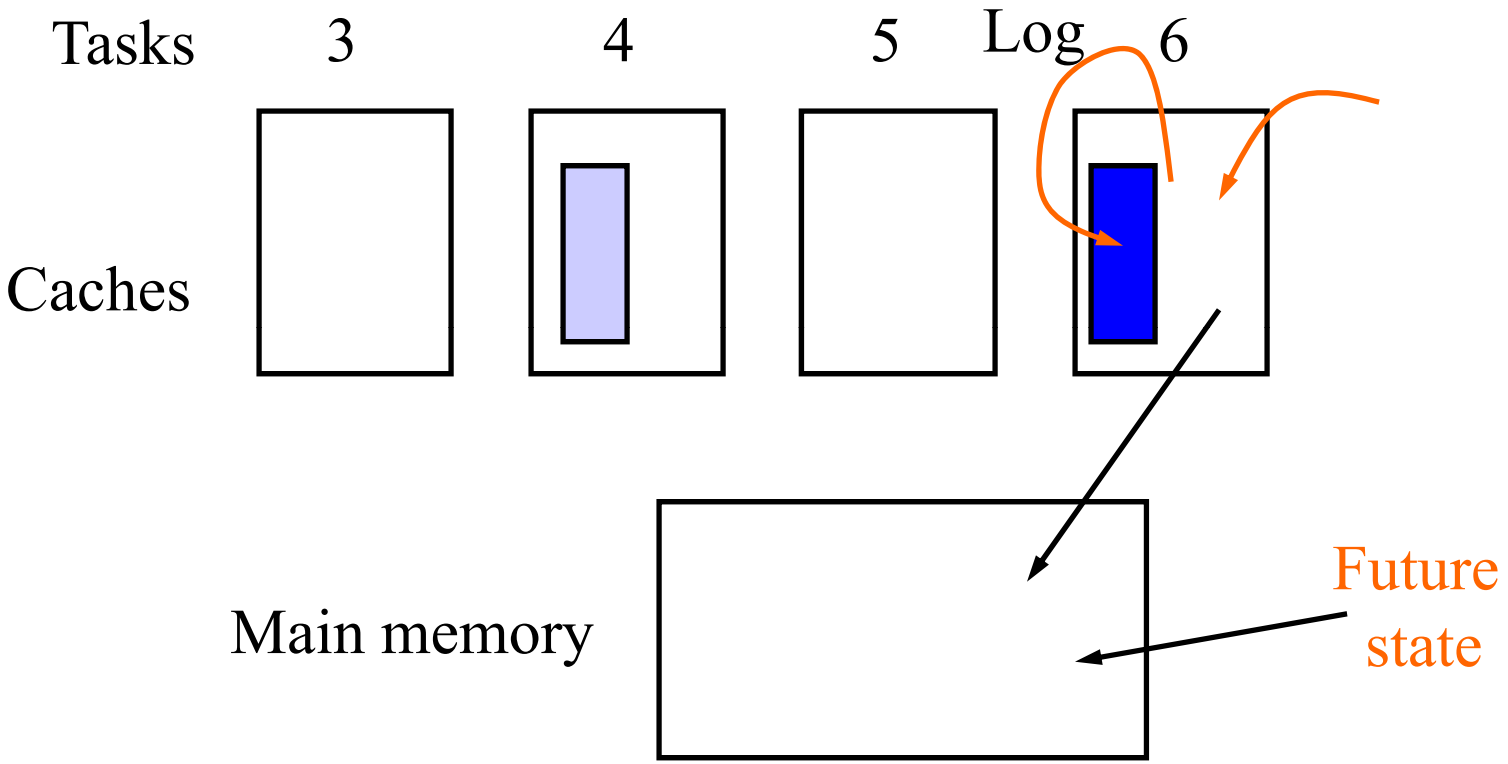


Merging of Task State: Architectural Main Memory



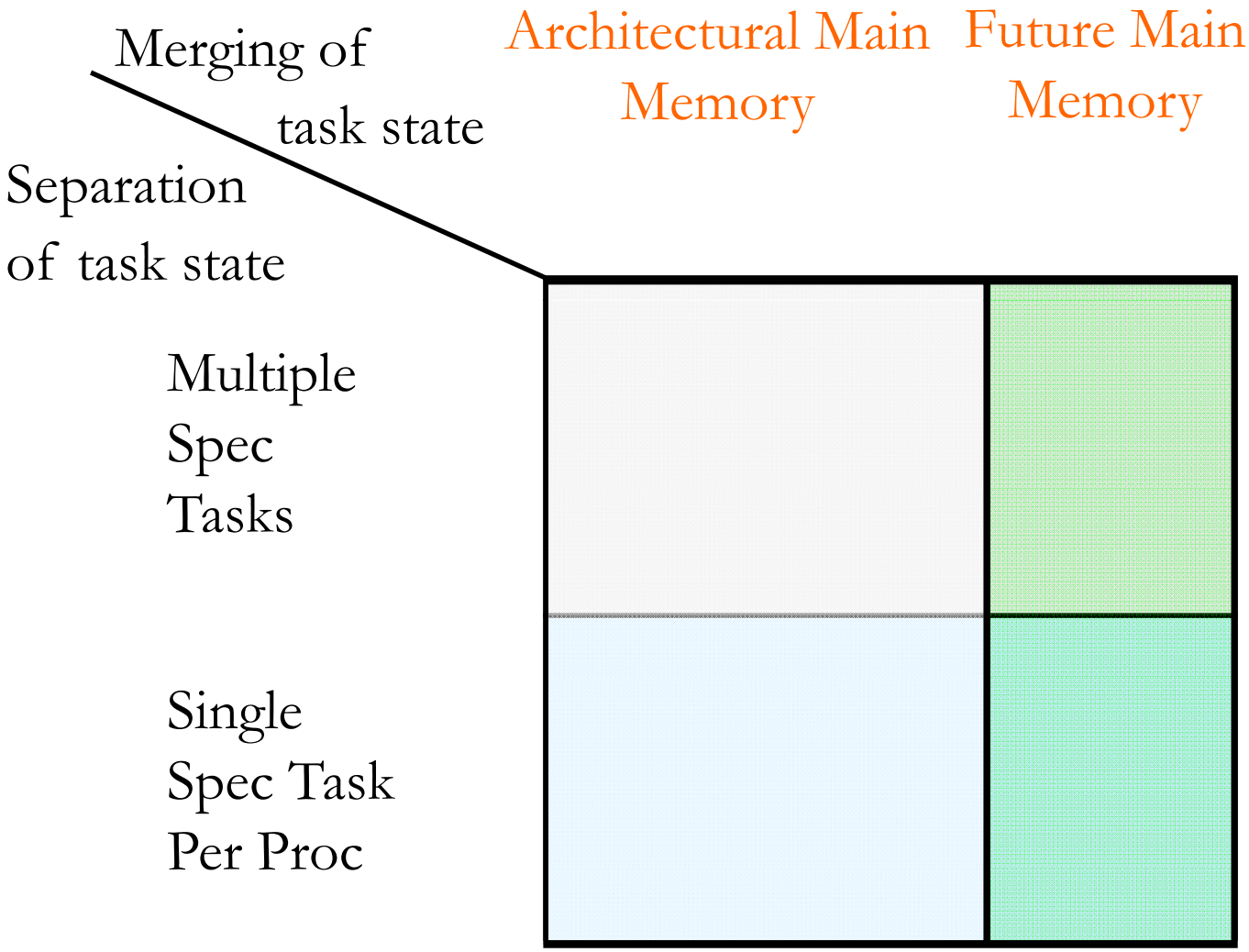
- Main memory keeps architectural or safe state
- Caches keep speculative state

Merging of Task State: Future Main Memory

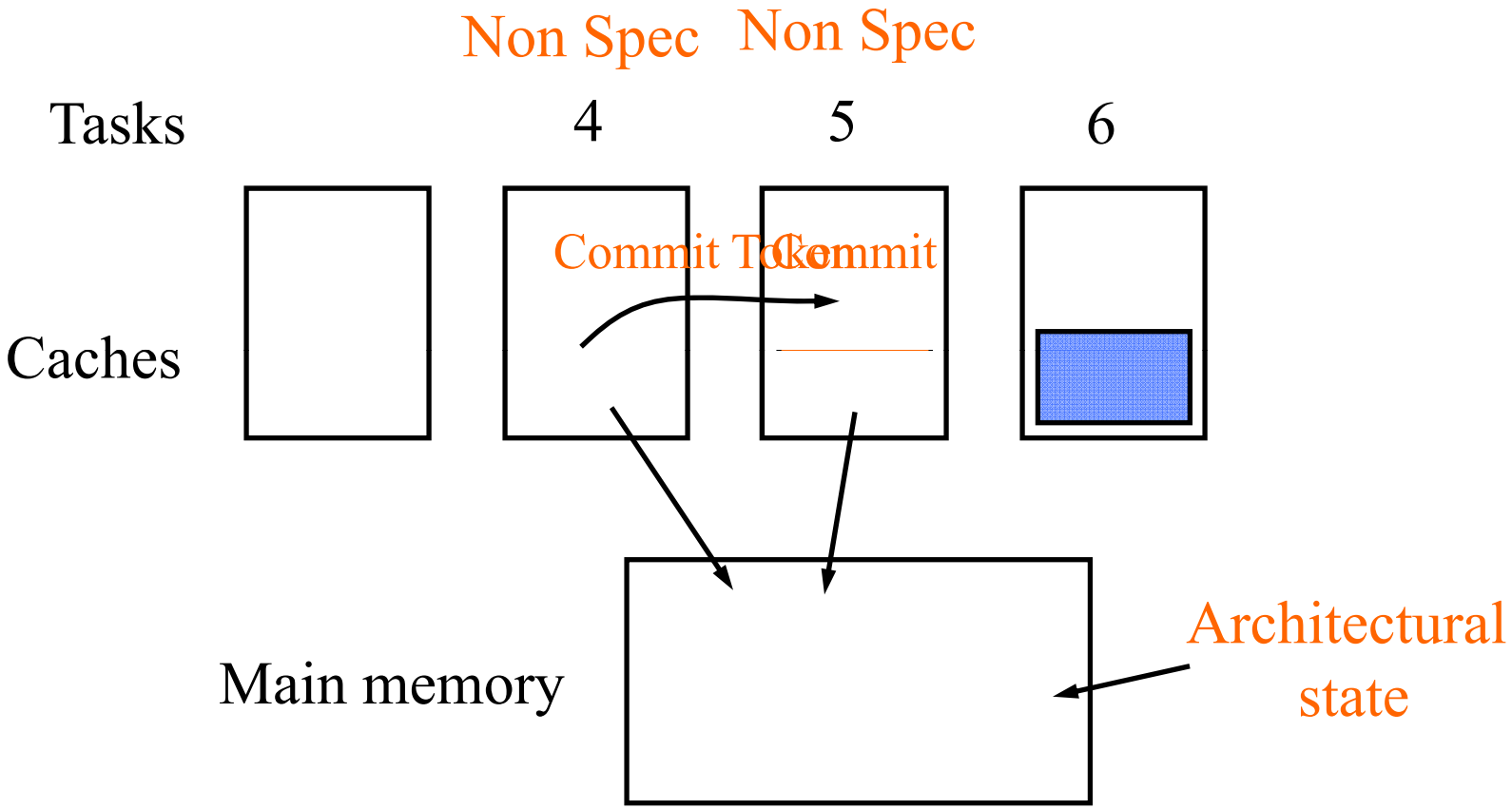


- Main memory keeps future state
- Logs keep previous state

Taxonomy of Buffering

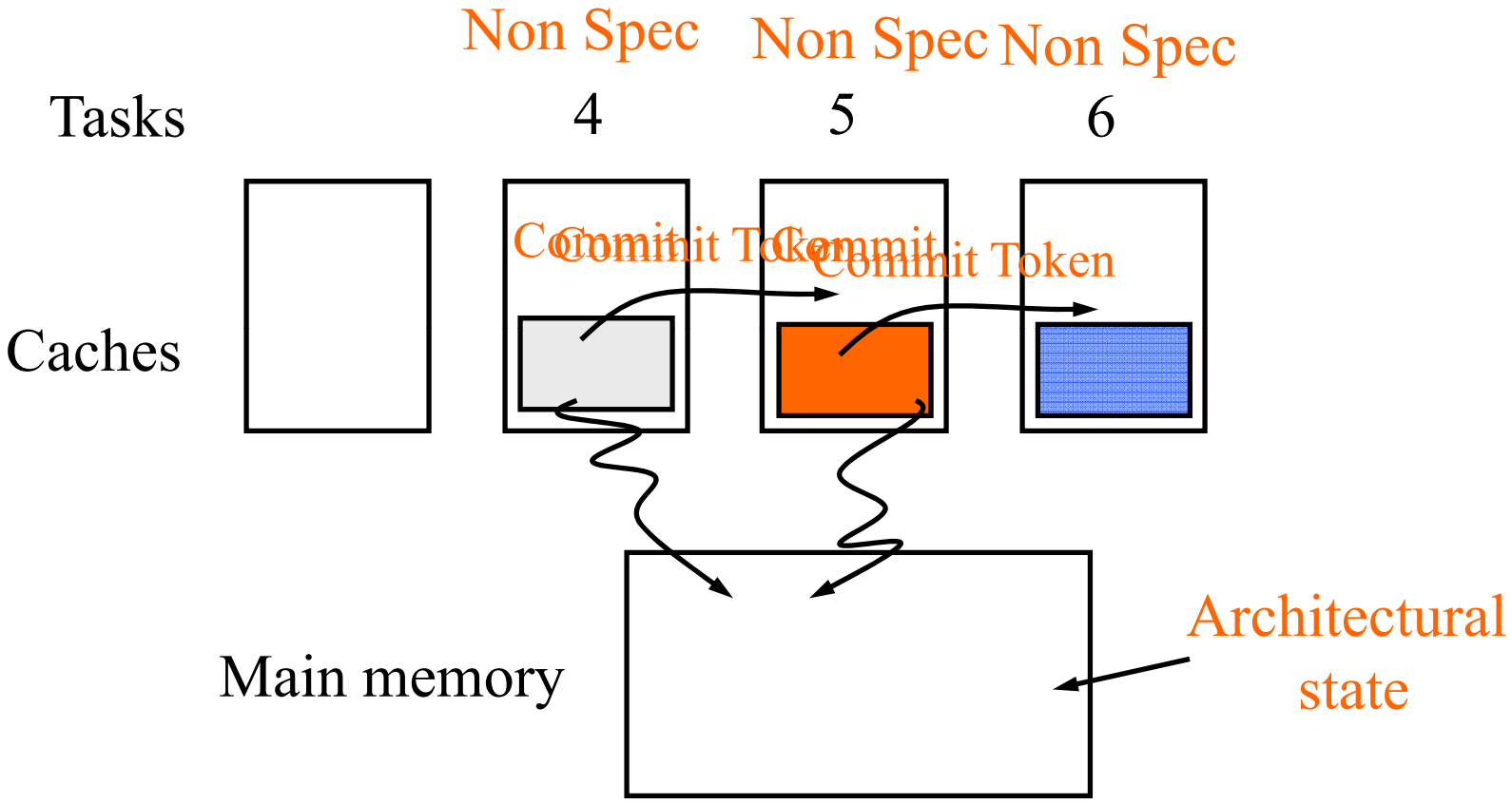


Merging with Architectural Main Memory: EAGER



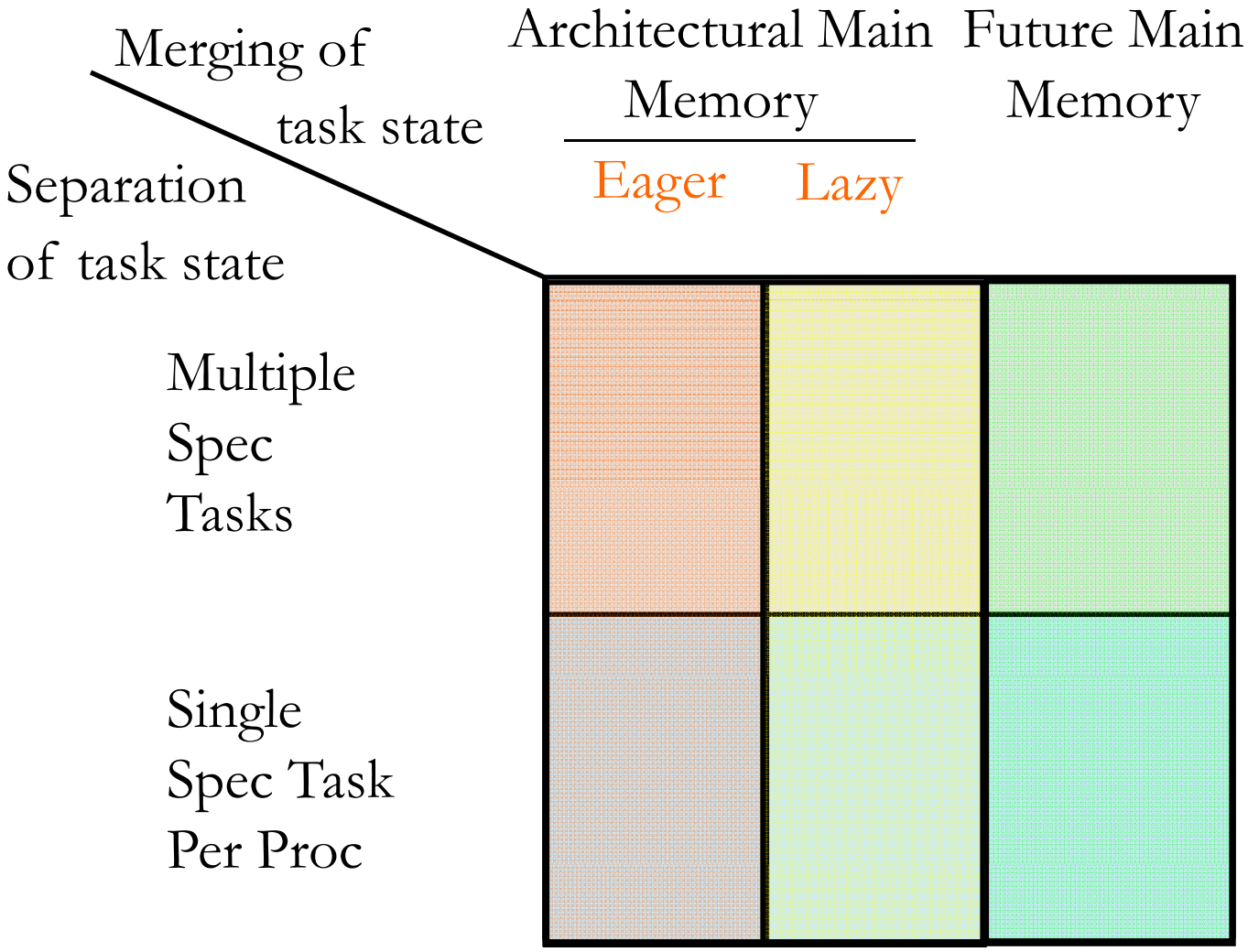
- State is merged with main memory at commit time

Merging to Architectural Main Memory: **LAZY**

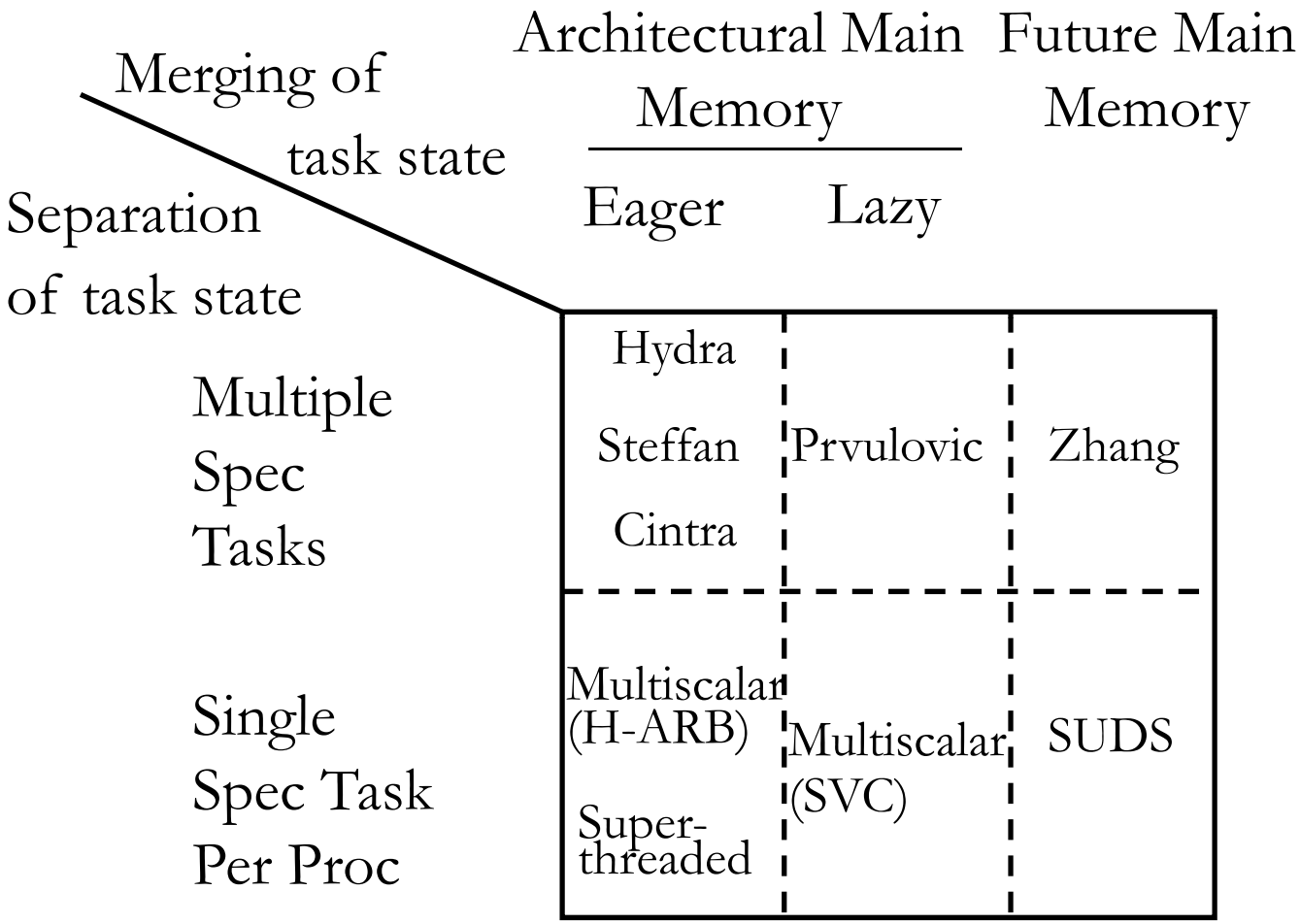


- State is merged with main memory at or after the commit

Taxonomy of Buffering



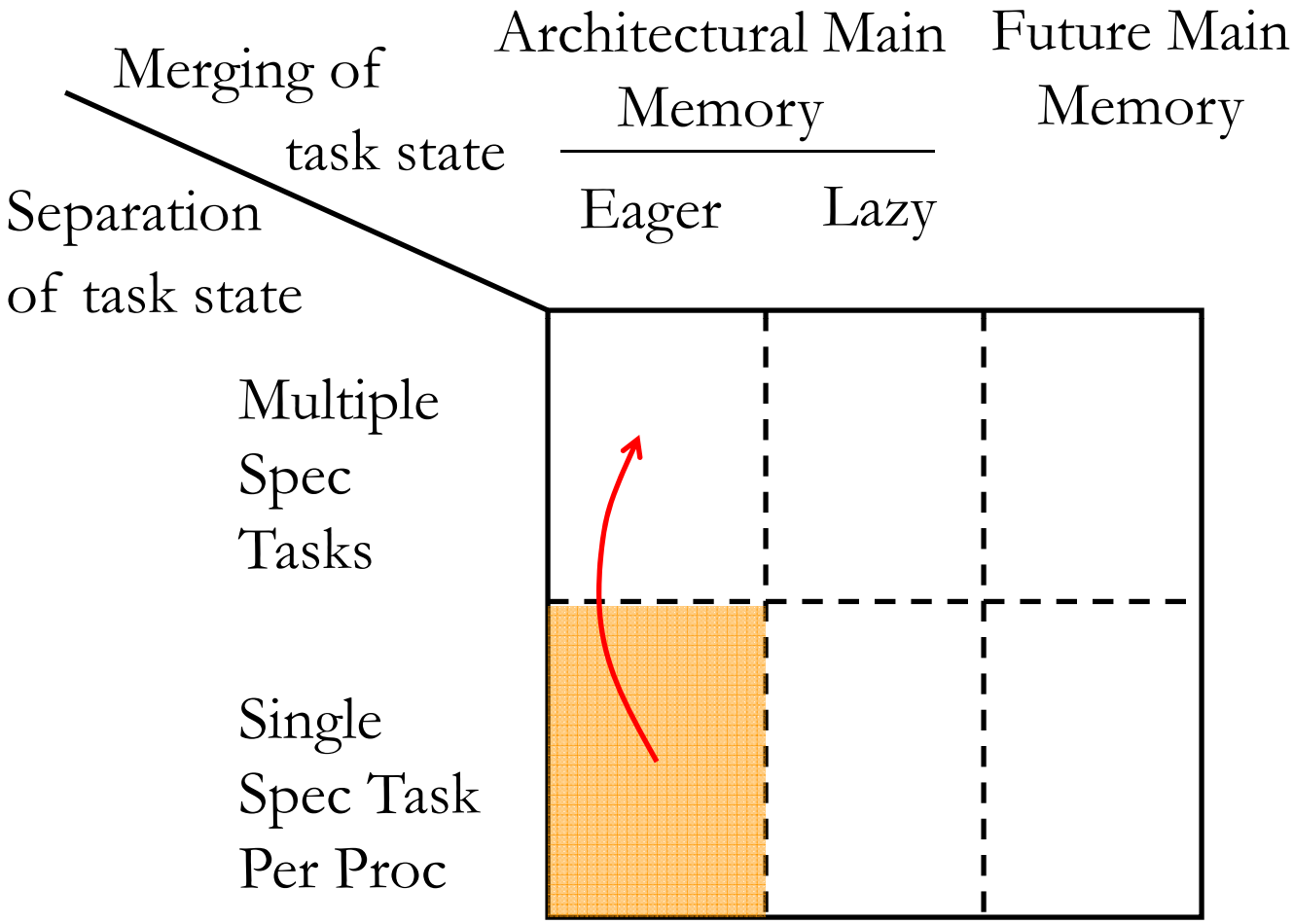
Mapping Existing Schemes



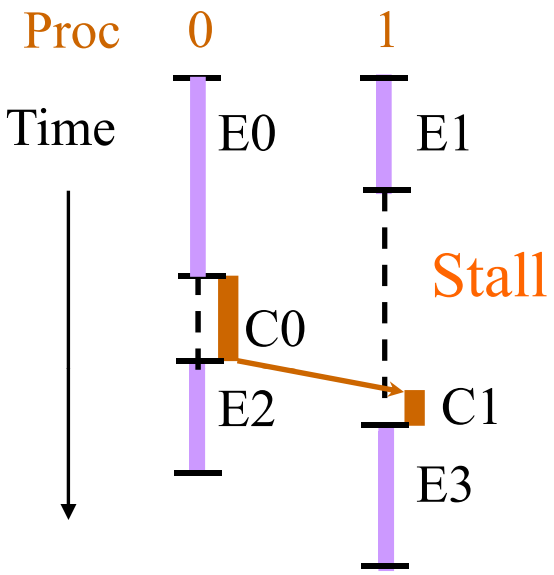
Roadmap

- Introduction to Thread-Level Speculation
- Taxonomy of Buffering
- **Tradeoffs Analysis**
- Evaluation
- Conclusions

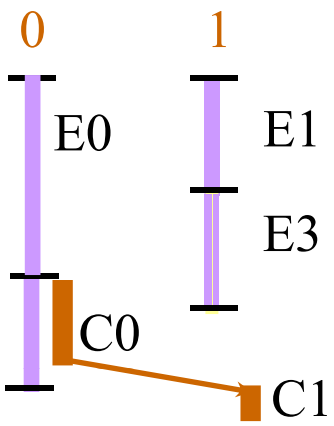
Comparing Schemes



Multiple Spec Tasks / Proc



Single Task per Processor



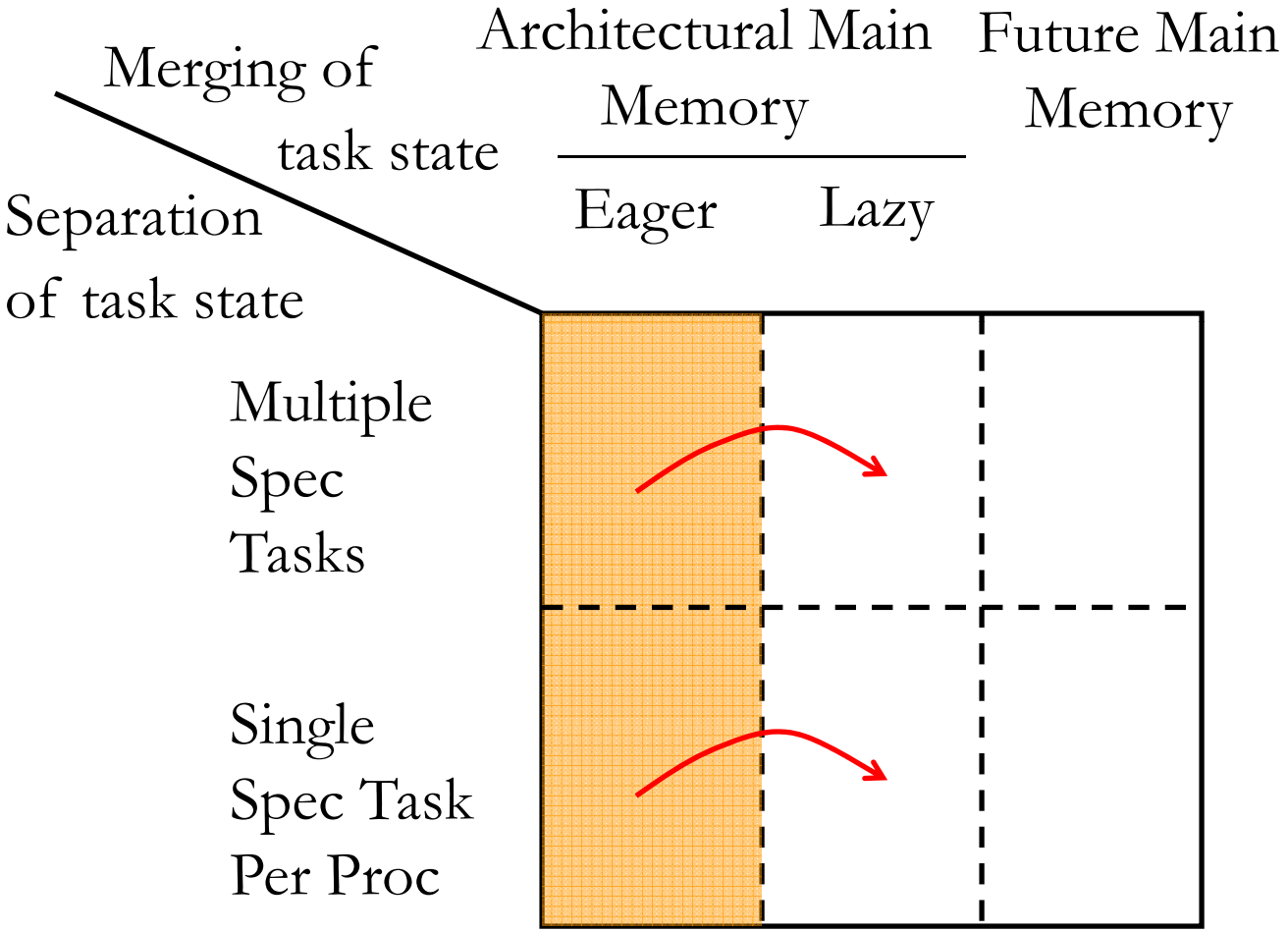
Multiple Tasks per Processor

Ei: Execution of Task i
Ci: Commit of Task i

Perf Tolerate load imbalance

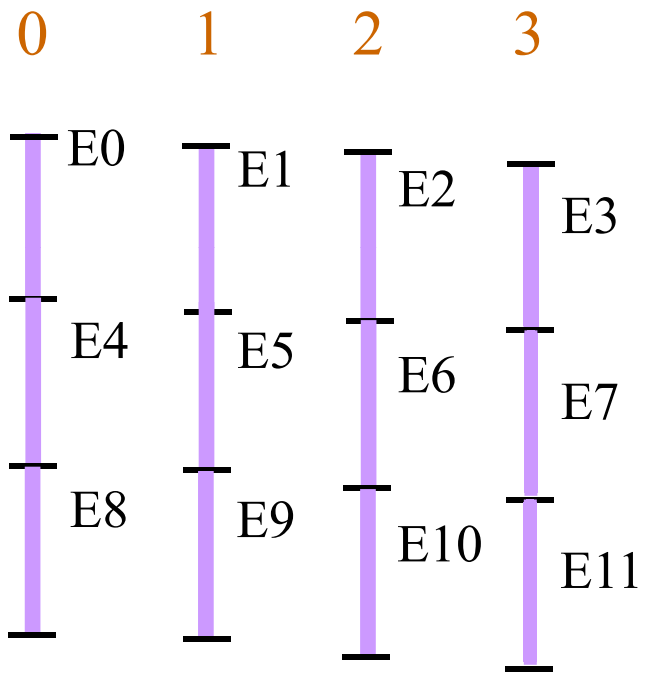
Cost Need Task ID in cache lines & advanced comparison logic in caches

Comparing Schemes



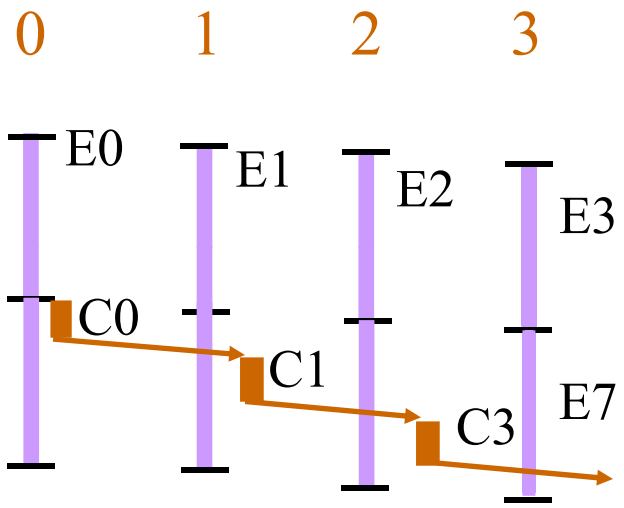
Two wavefronts: execution + commit

Execution Wavefront

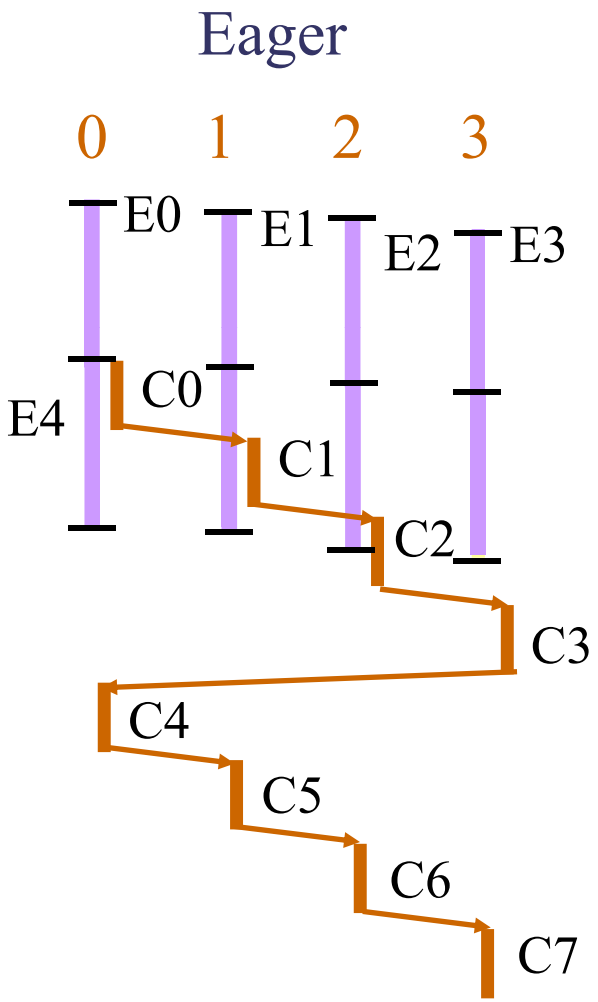


Two wavefronts: execution + commit

Commit Wavefront



Eager vs Lazy Merging in Architectural MM

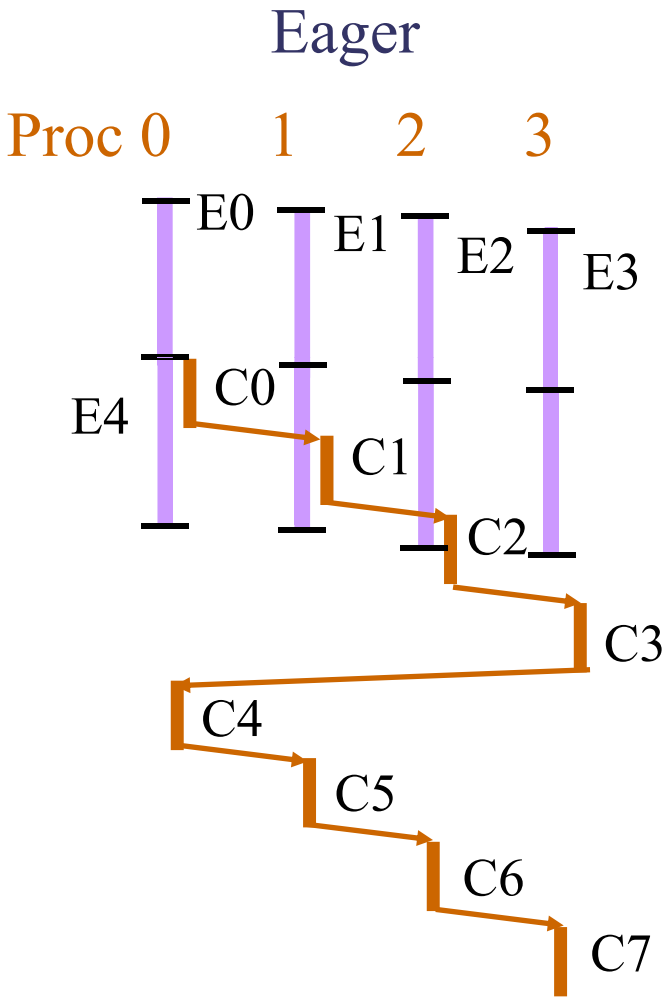


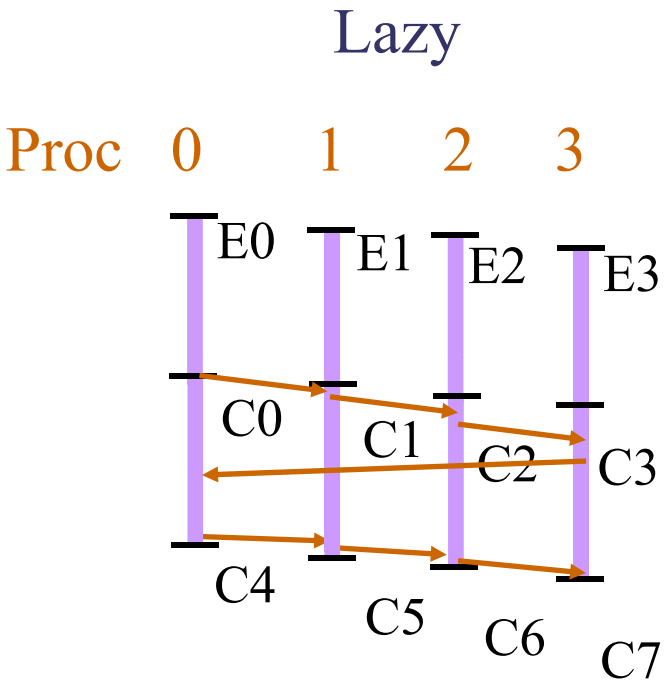
Multiple Tasks Per Processor

Commit wavefront appears in critical path when:

Ci * Nprocs > Ei

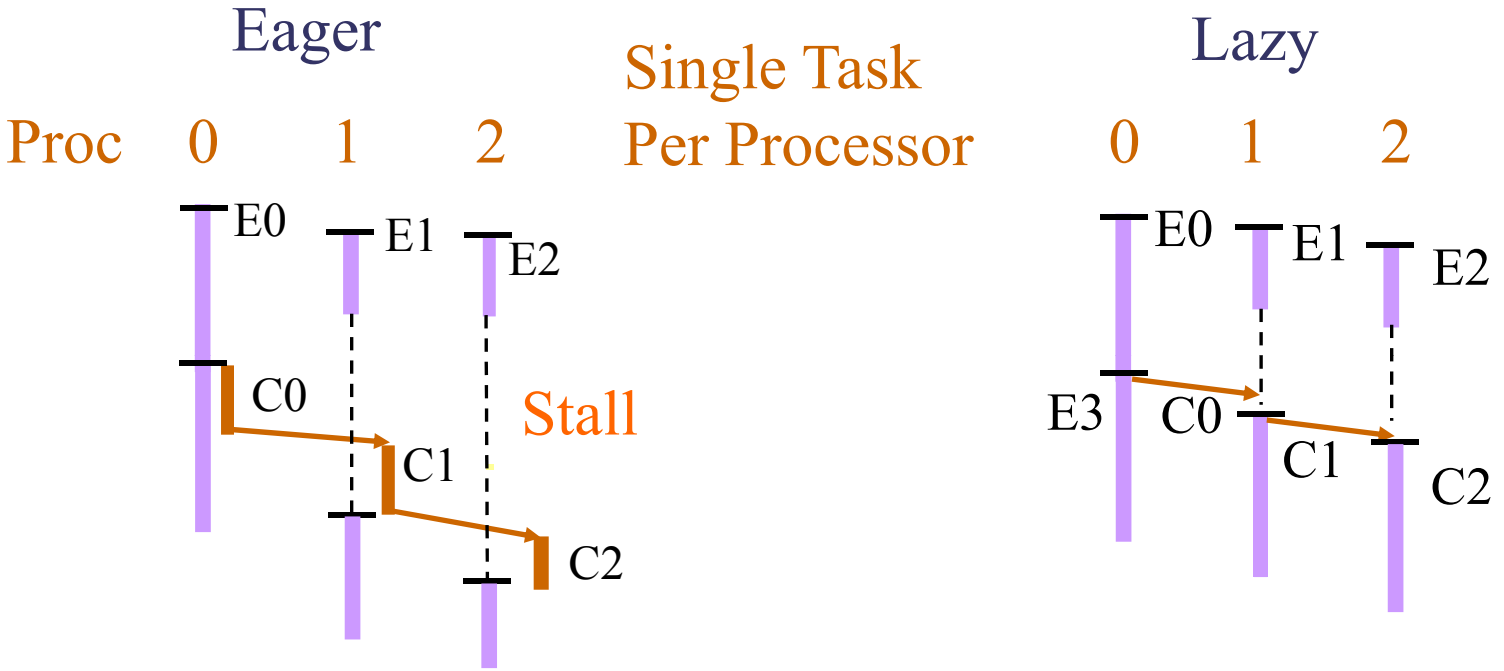
Eager vs Lazy Merging in Architectural MM





Removes the commit wavefront from the critical path

Eager vs Lazy Merging in Architectural MM

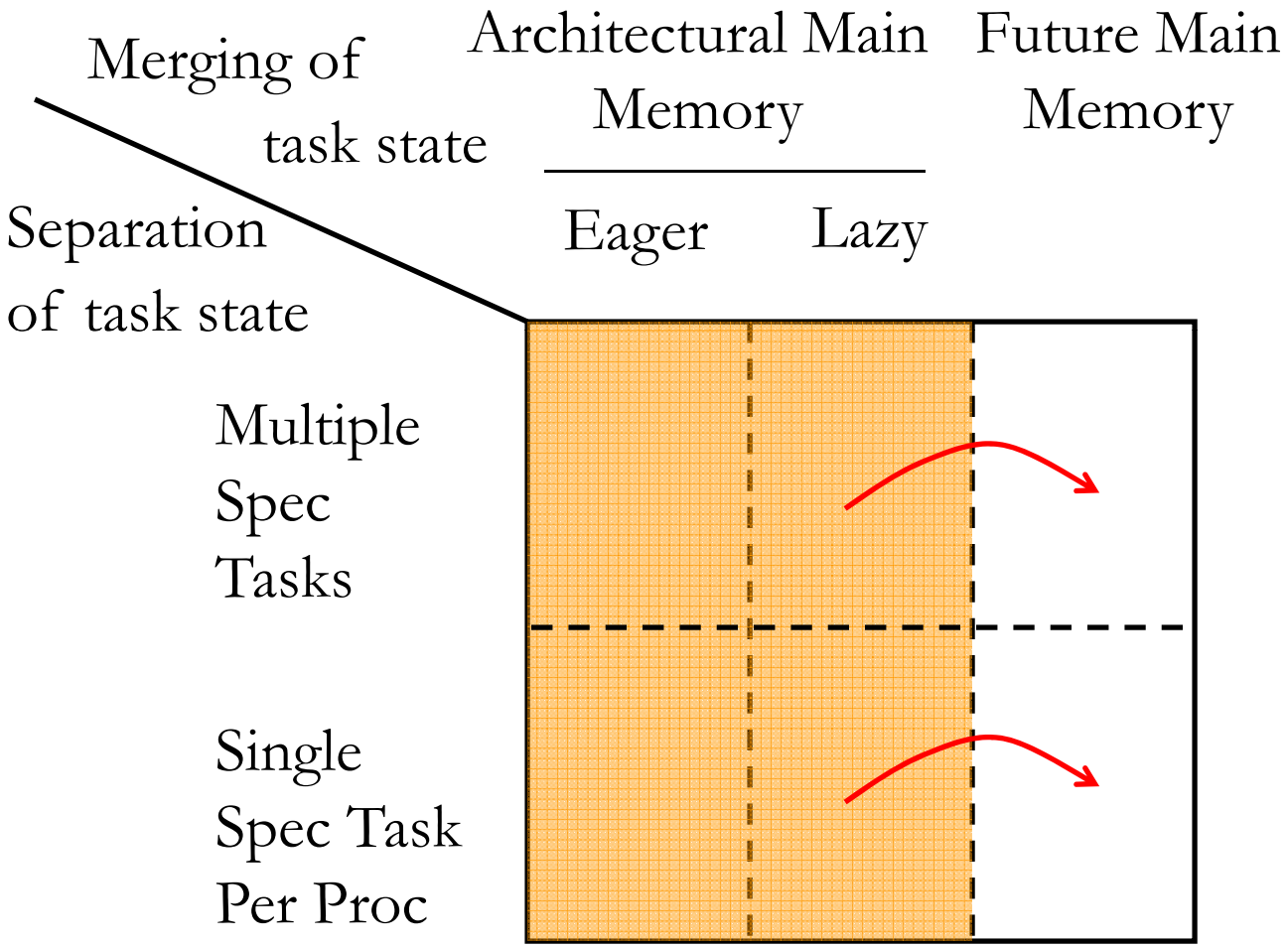


Perf Remove commit wavefront from critical path

Cost Need Task ID in cache lines

Cost Need version combining logic

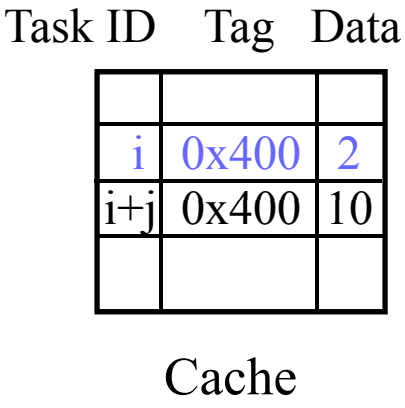
Comparing Schemes



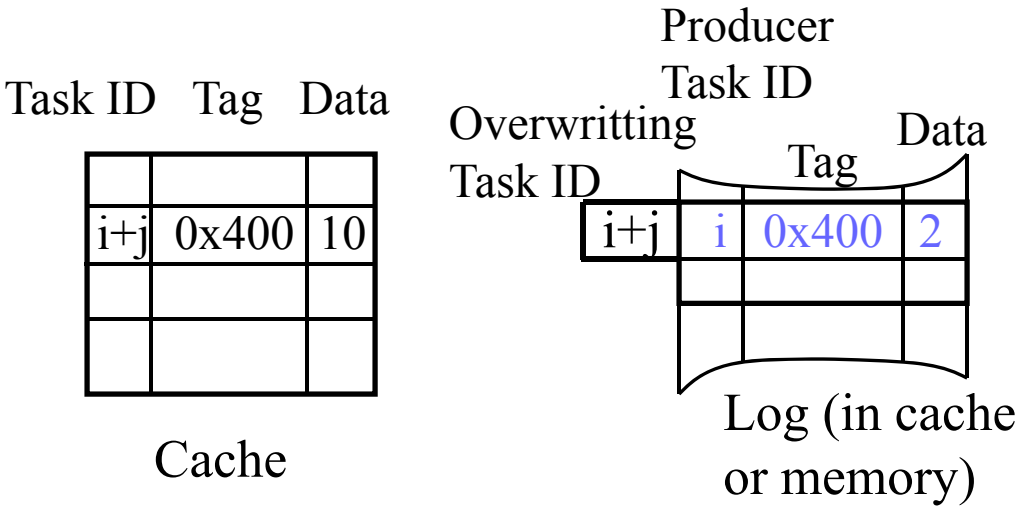
Future Main Memory

		value		address
Task i	writes	2	to	0x400
Task i+j	writes	10	to	0x400

Architectural MM



Future MM

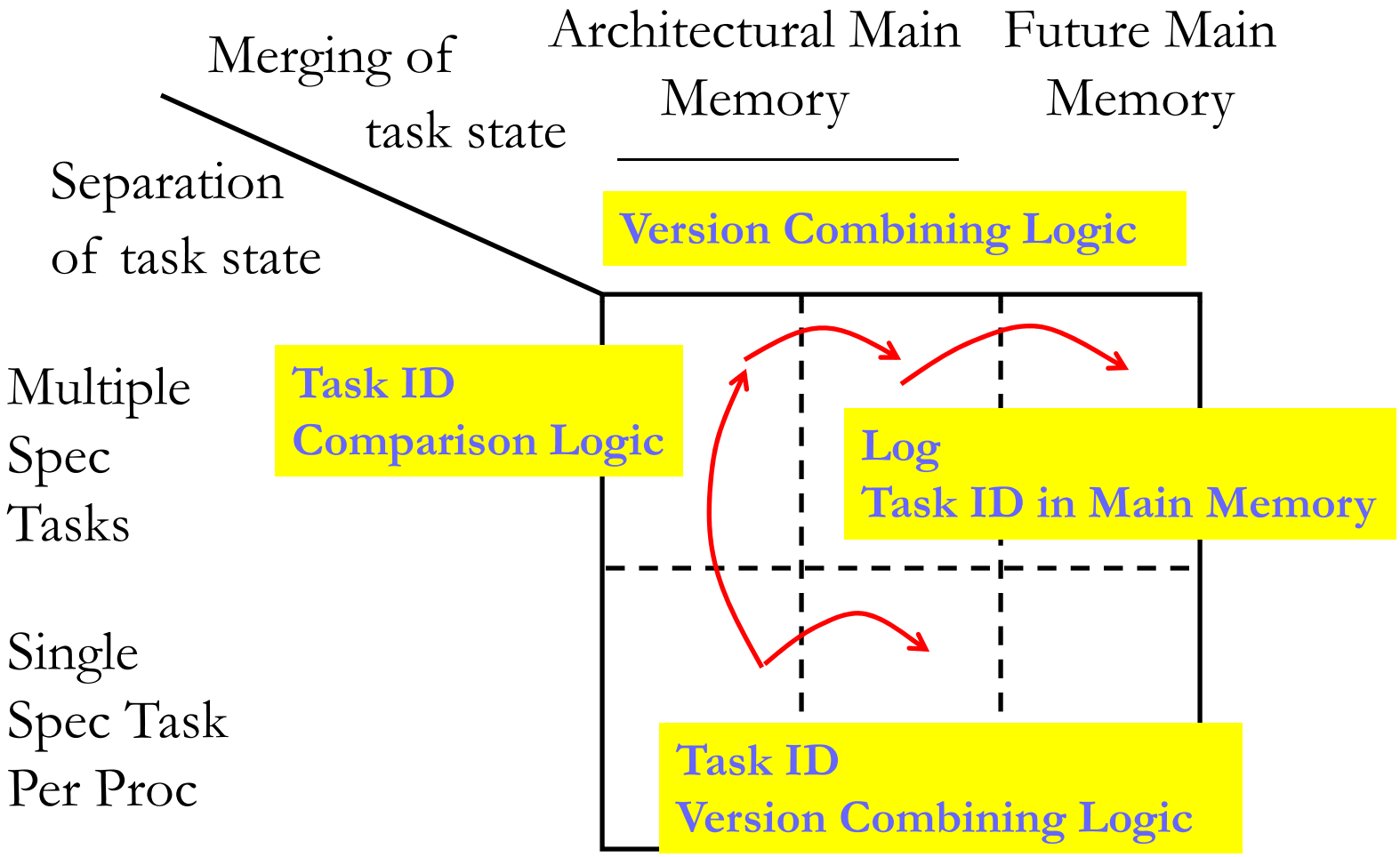


Perf Faster commit but slower version recovery

Cost Need log

Cost Need Task ID tags in main memory

Recap


Merging of task state
Architectural Main Memory
Future Main Memory
Separation of task state
Version Combining Logic
Multiple Spec Tasks
Task ID Comparison Logic
Log Task ID in Main Memory
Single Spec Task Per Proc
Task ID Version Combining Logic

Roadmap

- Introduction to Thread-Level Speculation
- Taxonomy of Buffering
- Tradeoffs Analysis
- **Evaluation**
- Conclusions

Evaluation Environment

- Execution-driven simulator
  - CC-NUMA multiprocessor with 16 processors
  - CMP (Chip multiprocessor) with 8 processors
- Out-of order superscalar processor + 2 levels of cache
  - Issues 4 instructions per cycle

Applications

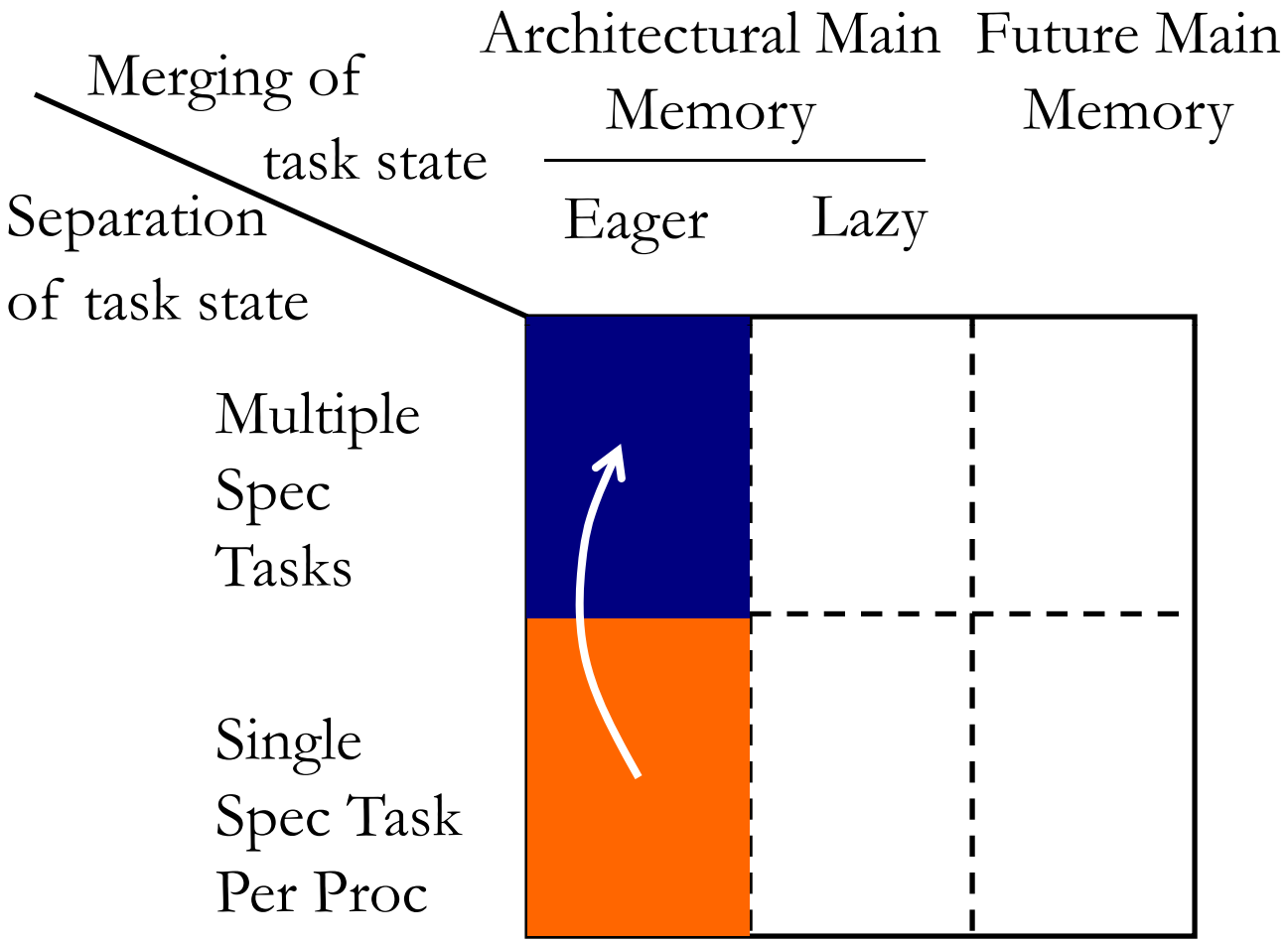
- Numerical applications:
  - Apsi (Specfp2000)
  - Dsmc3d and Euler (HPF-2)
  - P3m (NCSA)
  - Tree (Univ. of Hawaii)
  - Bdna and Track (Perfect)

The non-analyzable loops account on average for 59% of the serial execution time

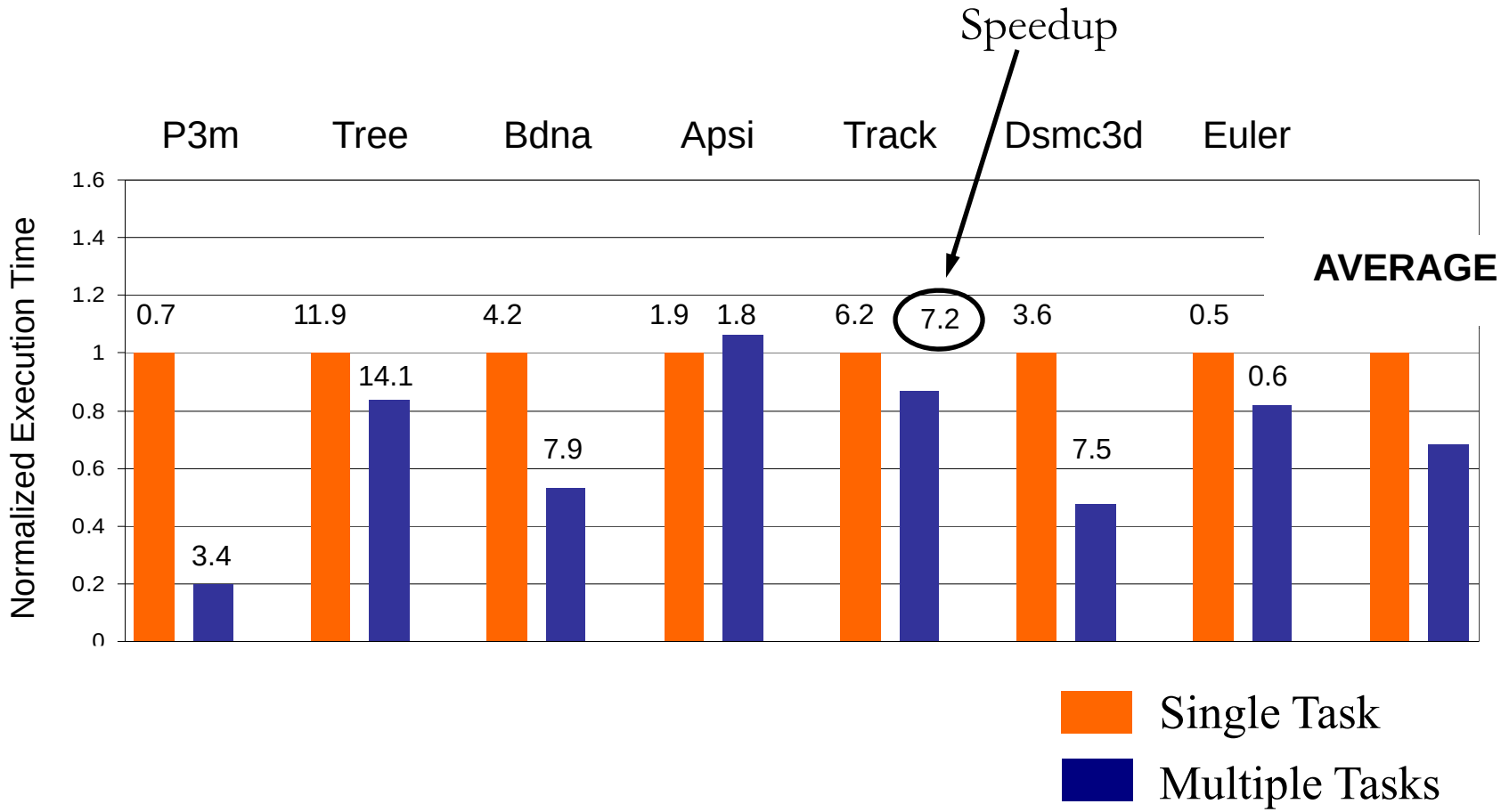
- Non-analyzable loops are identified with the Polaris parallelizing compiler

- Speed-ups shown for the non-analyzable loops only

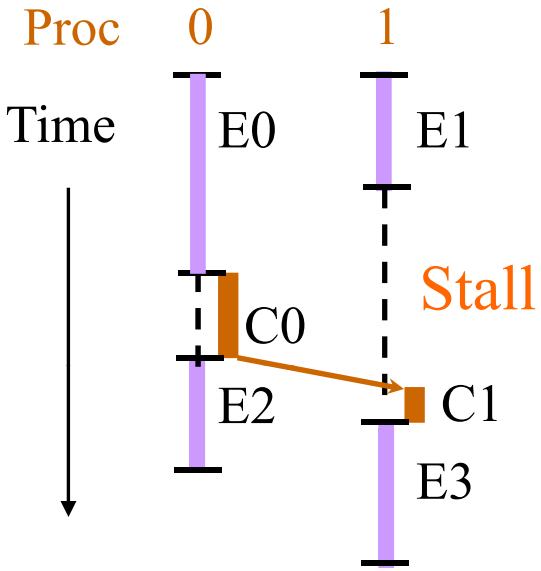
Supporting Multiple Tasks



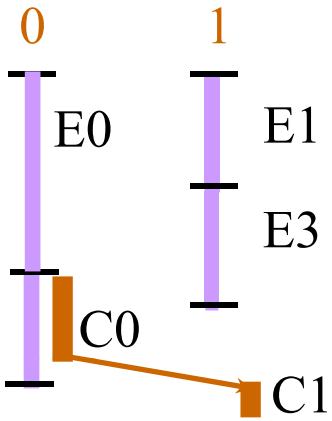
CC-NUMA: Supporting Multiple Tasks (Eager)



Supporting Multiple Tasks (Eager)



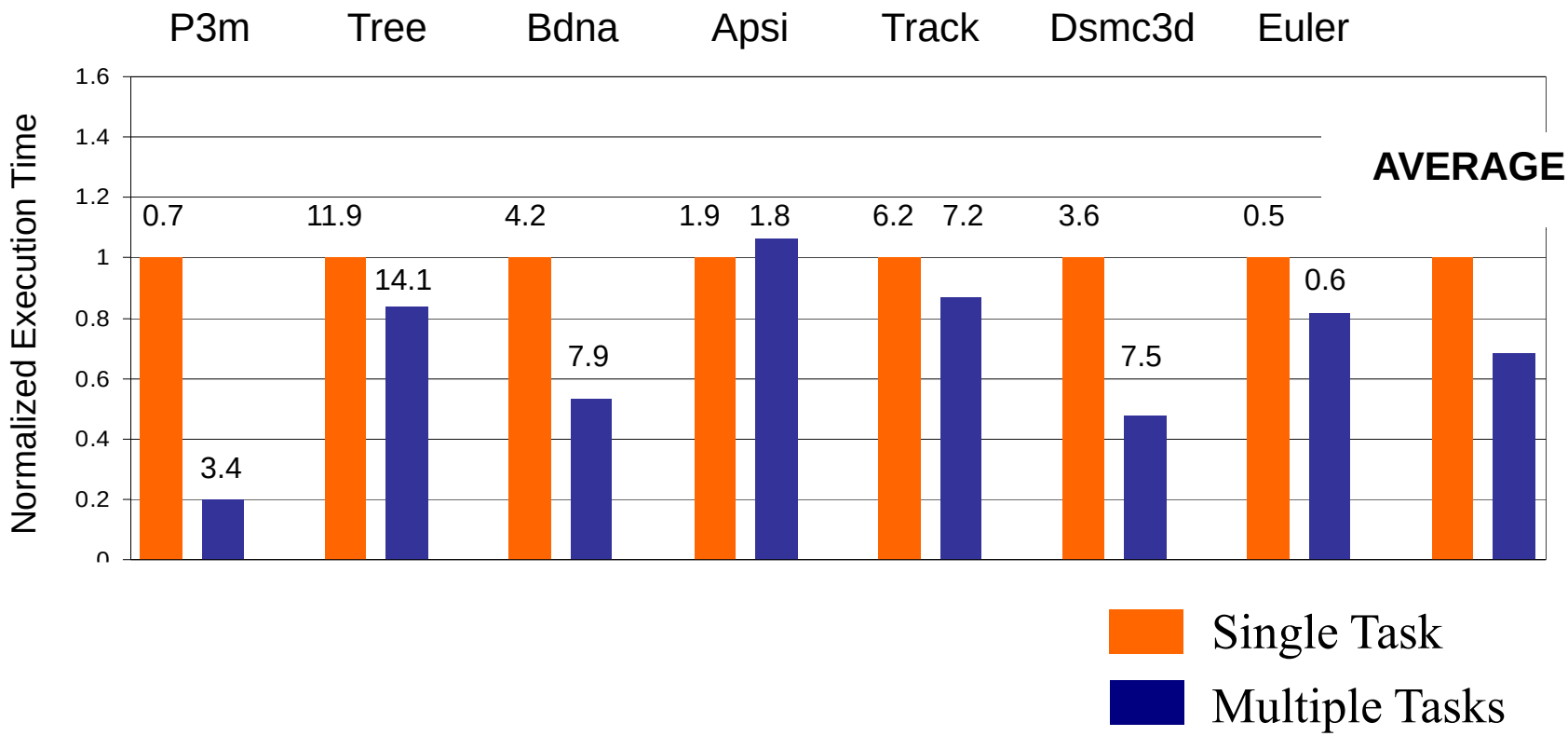
Single Task per Processor



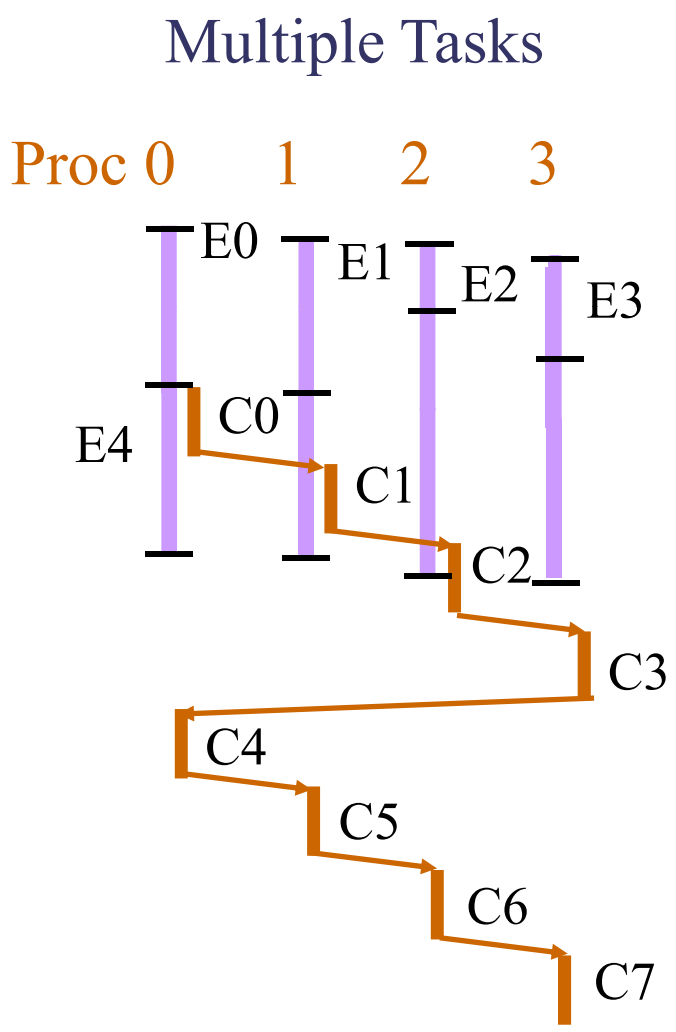
Multiple Tasks per Processor

Ei: Execution of Task i
Ci: Commit of Task i

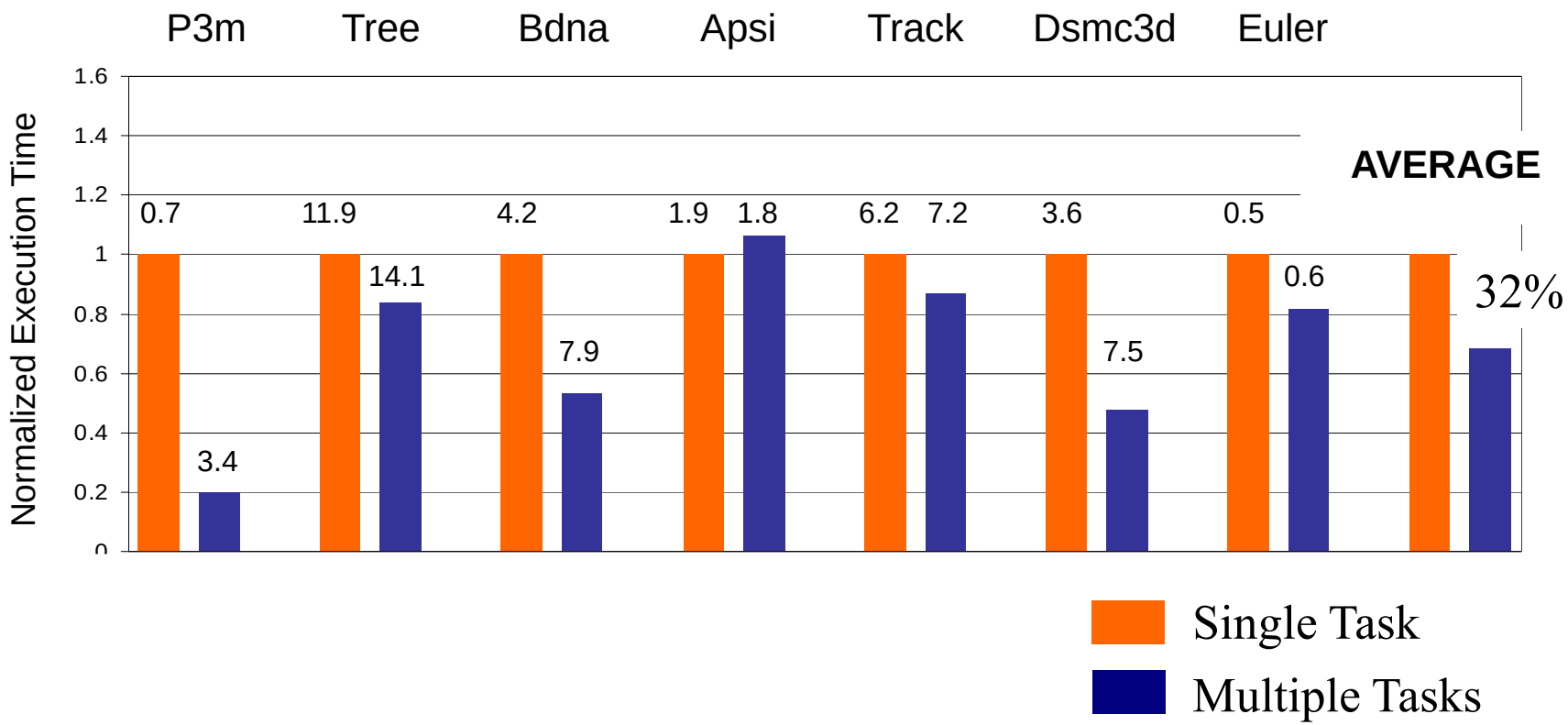
CC-NUMA: Supporting Multiple Tasks (Eager)



Supporting Multiple Tasks (Eager)

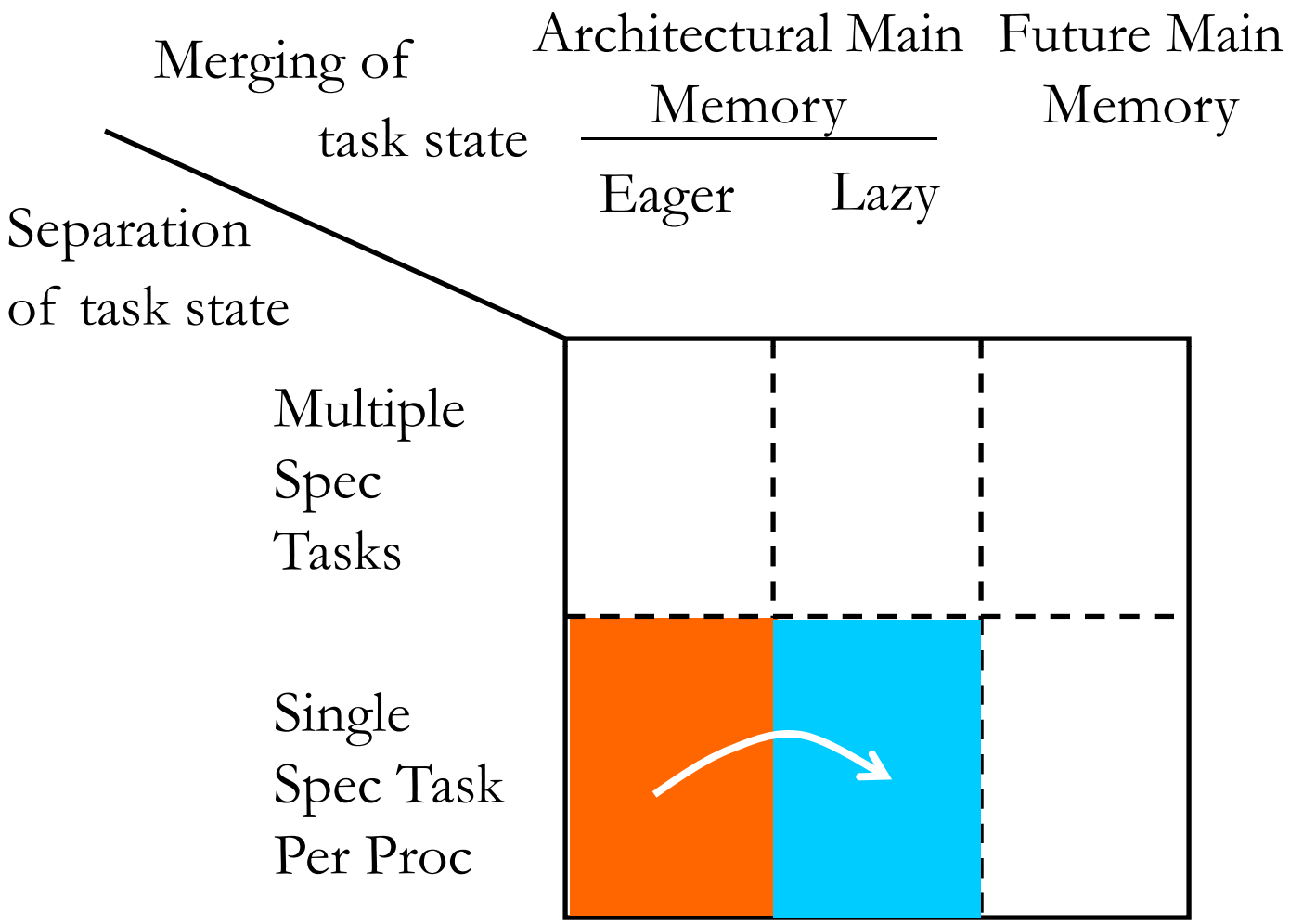


CC-NUMA: Supporting Multiple Tasks (Eager)

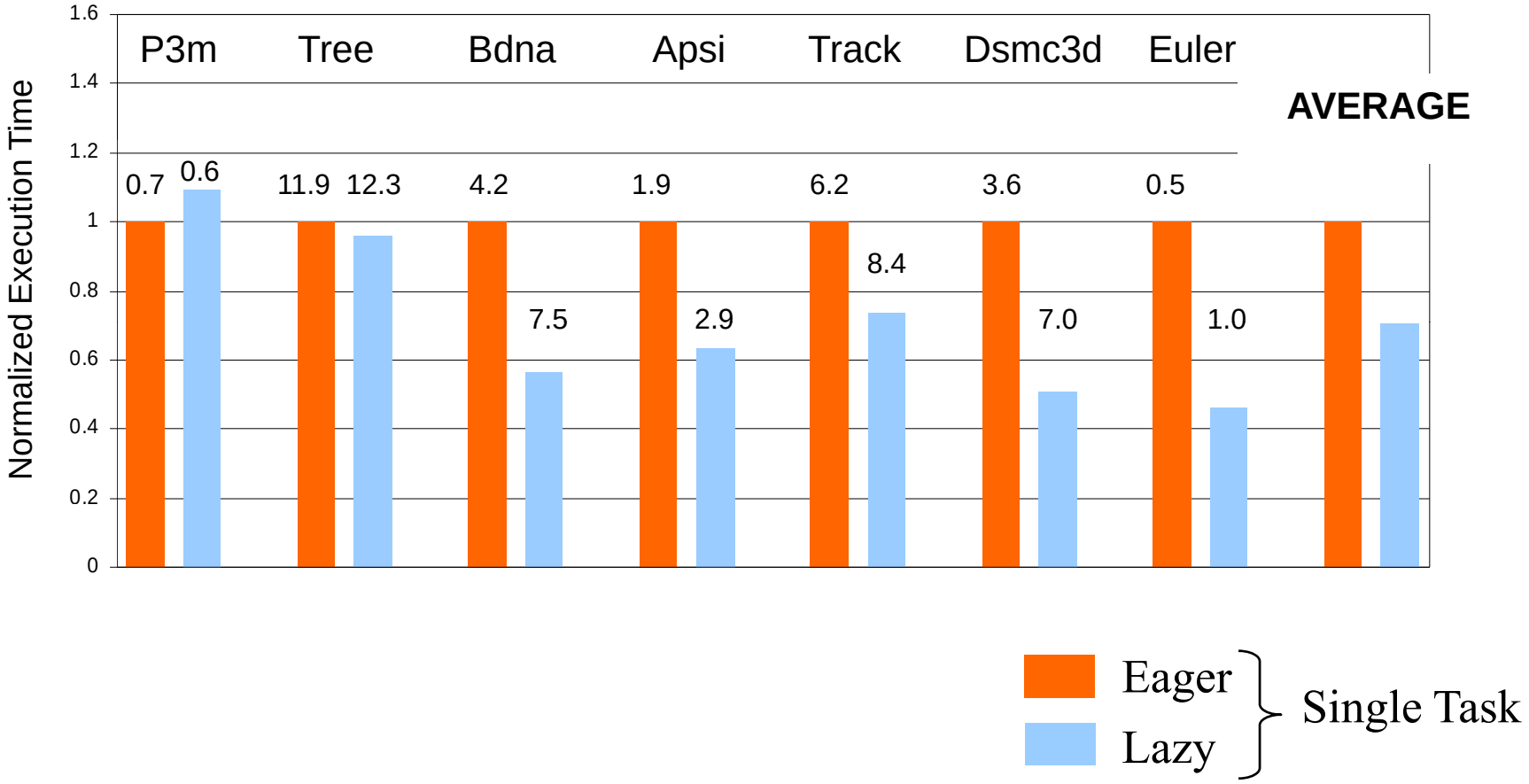


- Apps run 32% faster with Multiple Tasks

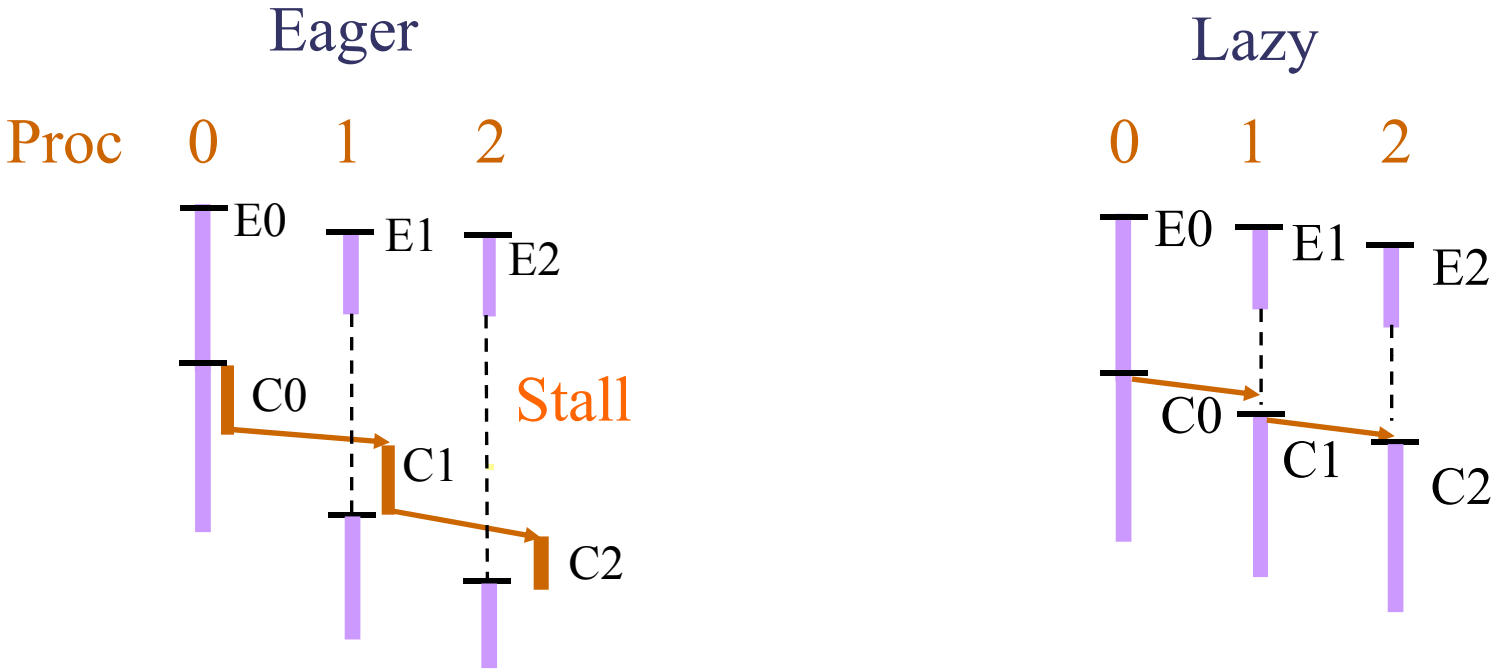
Adding laziness to Single Task



CC-NUMA: Single Task Eager & Lazy Merging

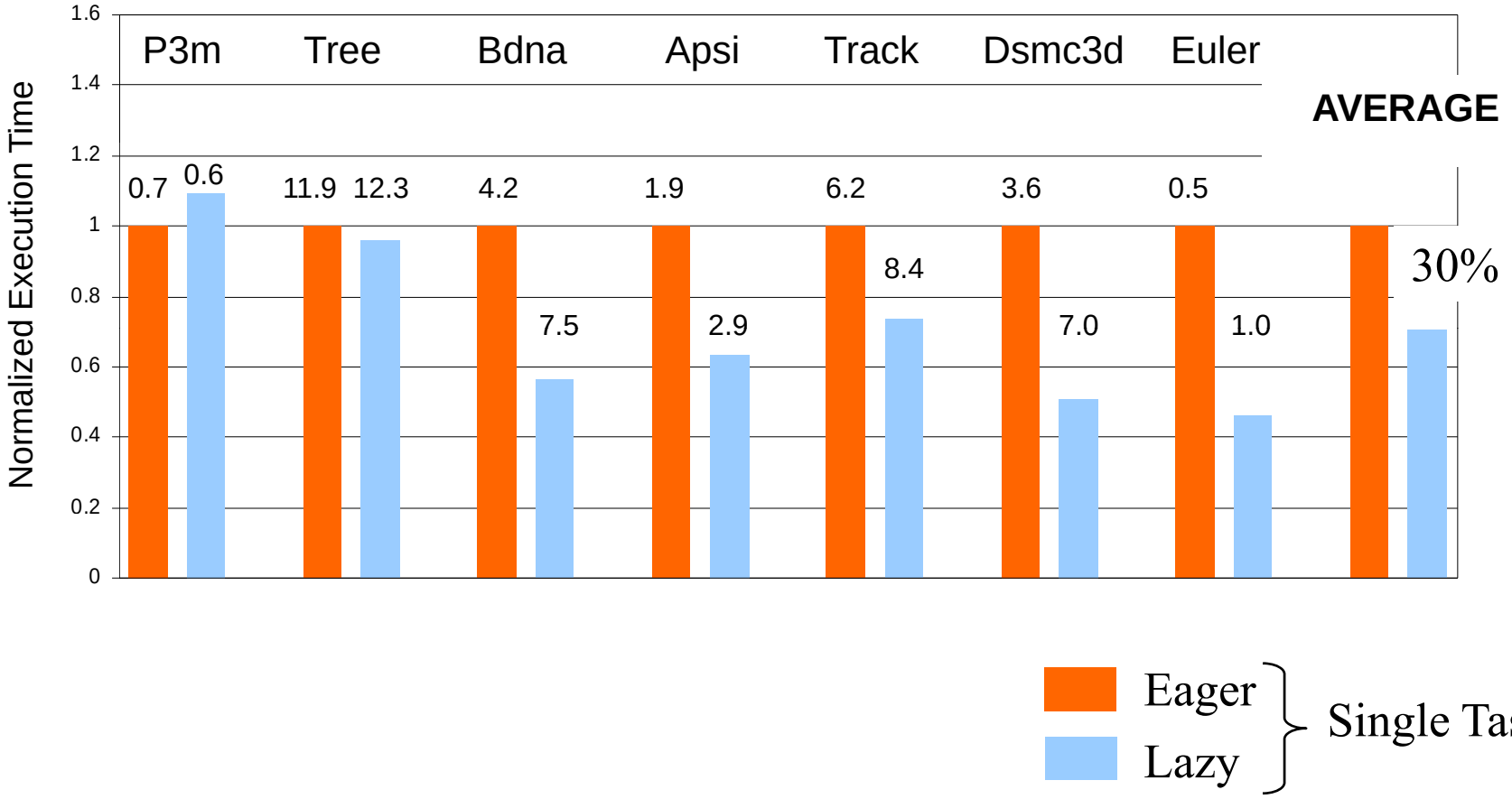


Single Task Eager & Lazy Merging



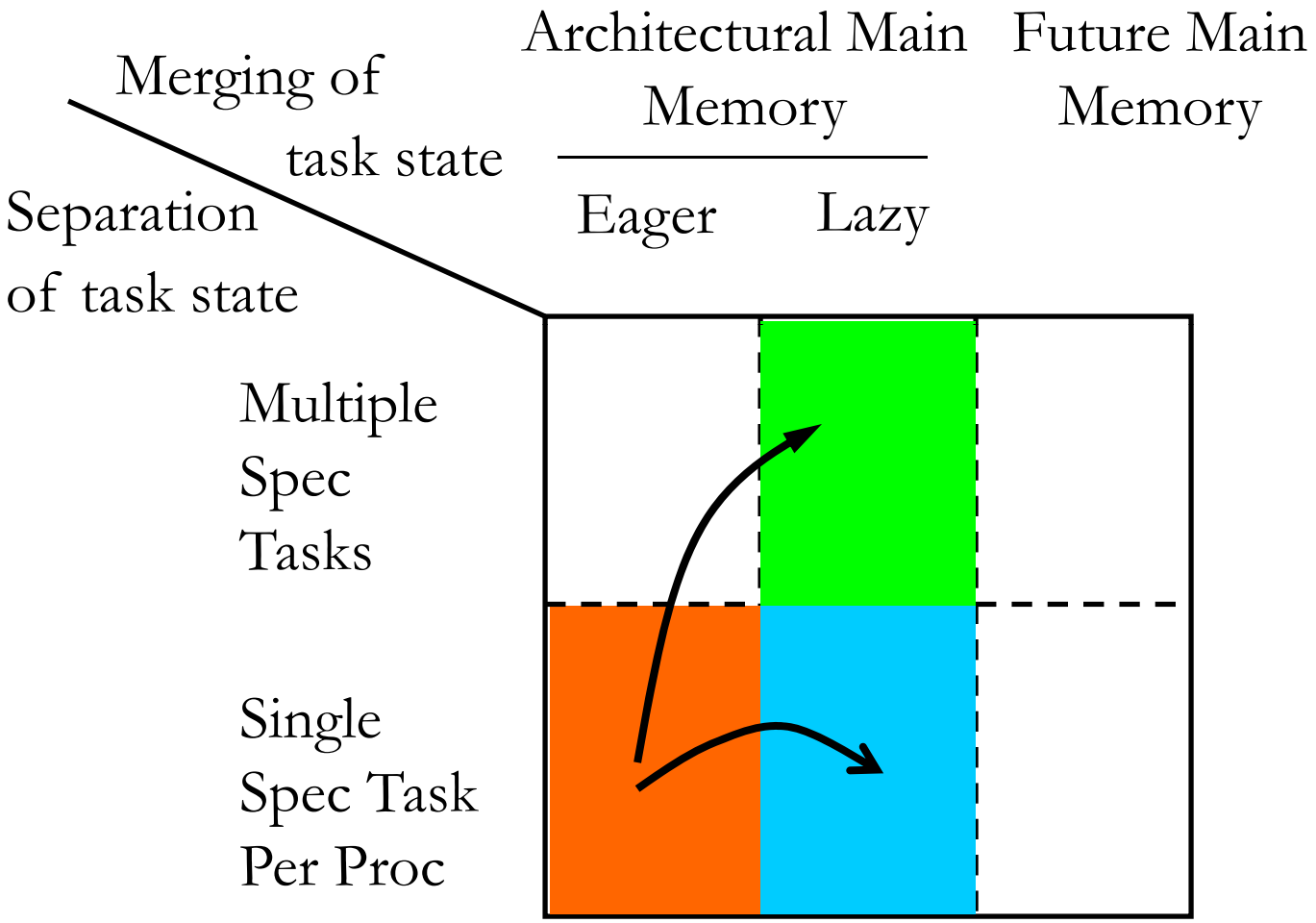
Single Task
Per Processor

CC-NUMA: Single Task Eager & Lazy Merging

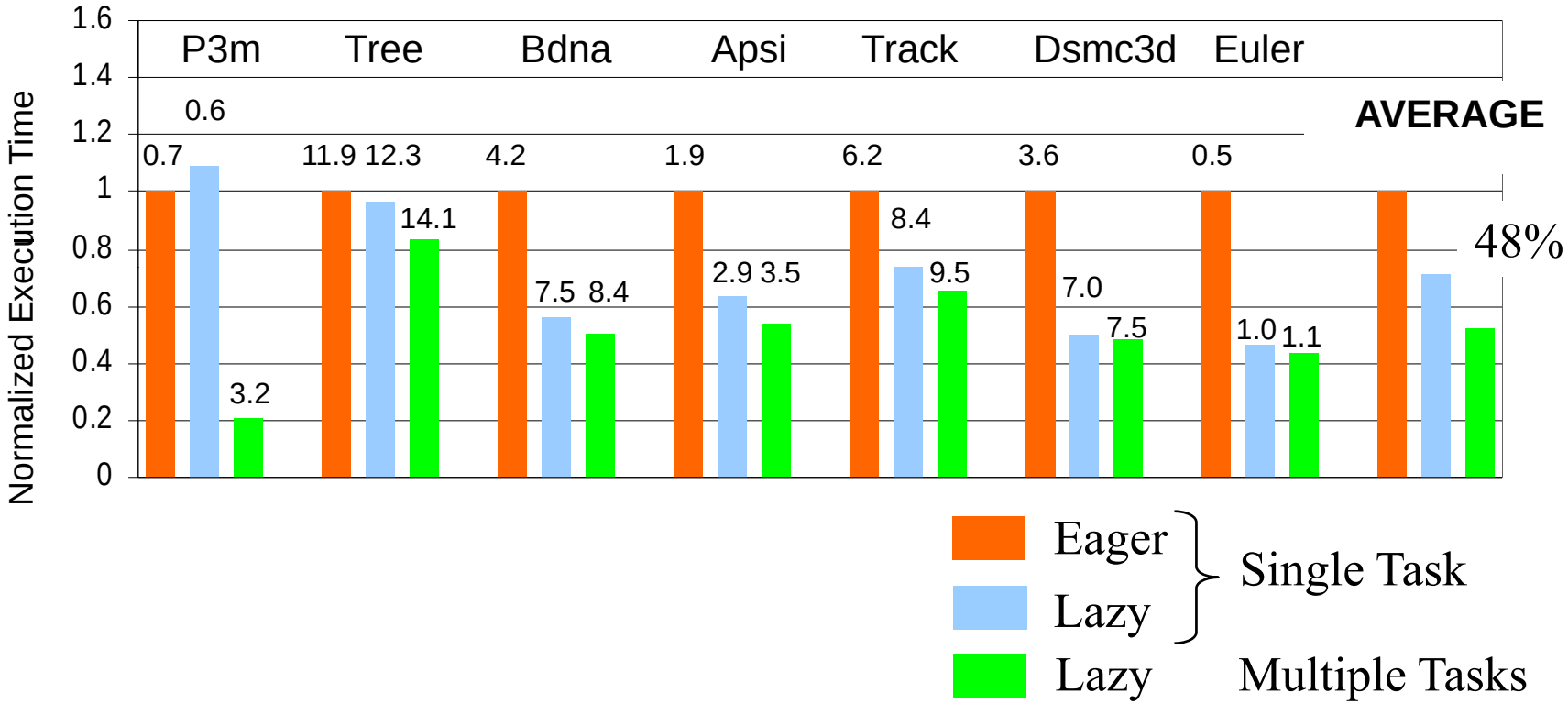


- Laziness speeds up the applications 30 %

Multiple Tasks & Laziness

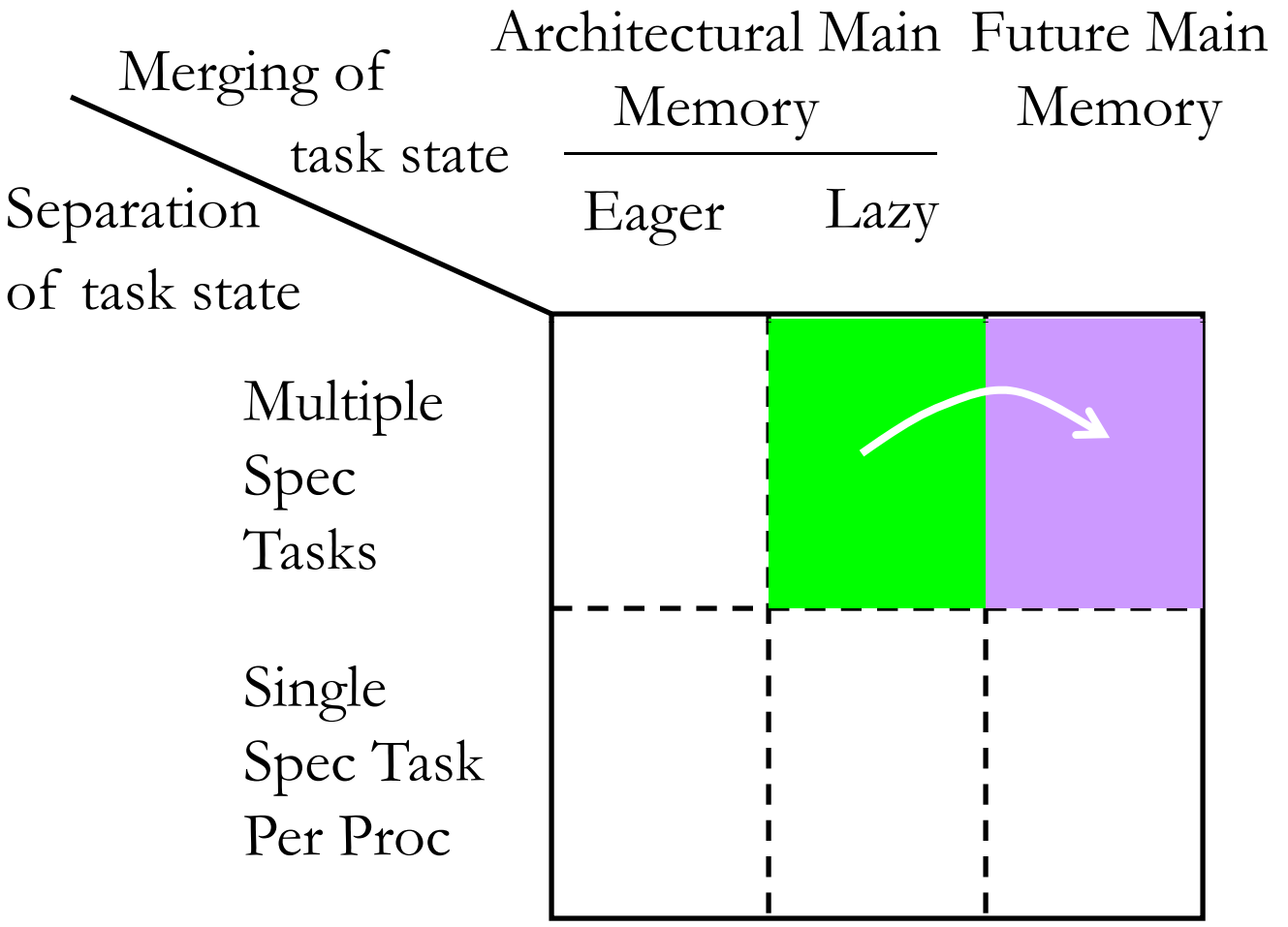


CC-NUMA: Multiple Tasks & Laziness

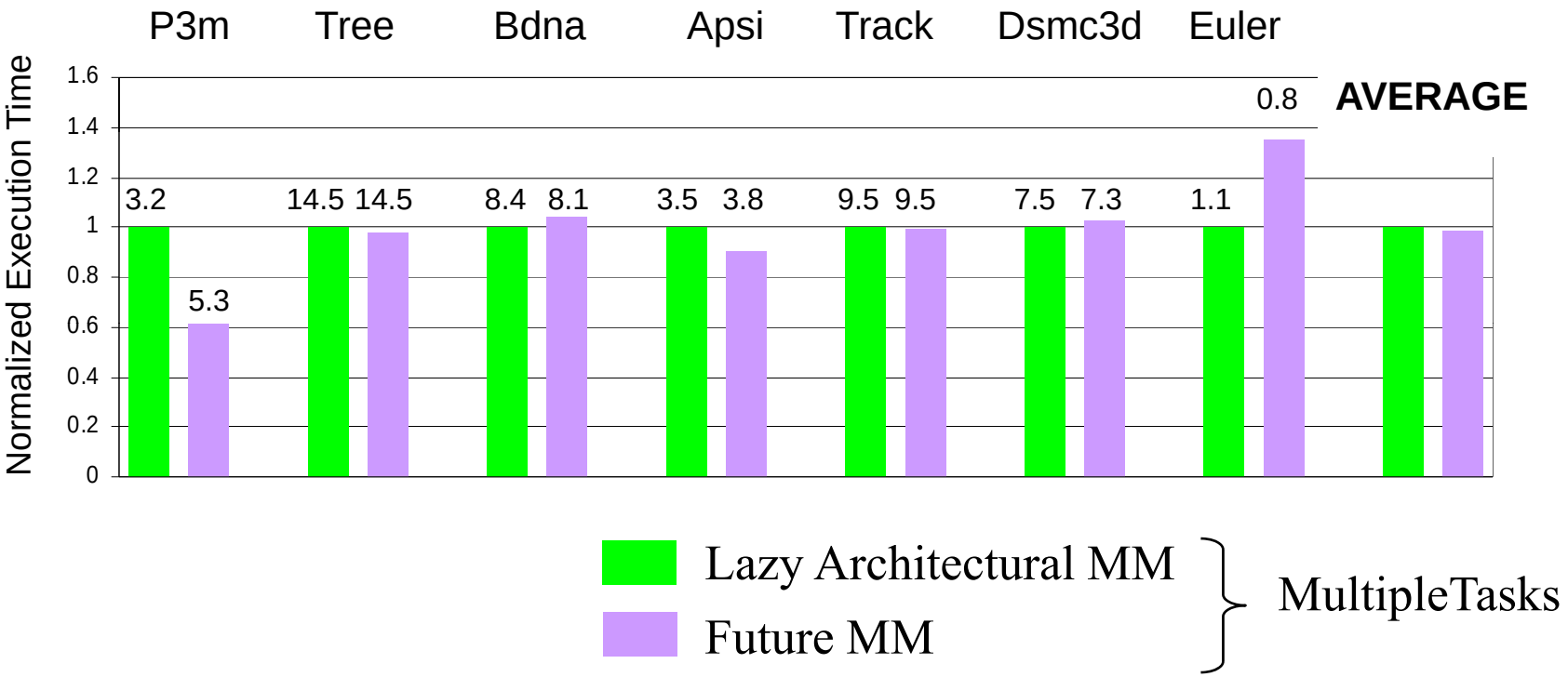


- Gains from Multiple Tasks and Laziness are **fairly orthogonal** (48% speedup)

Architectural and Future MM

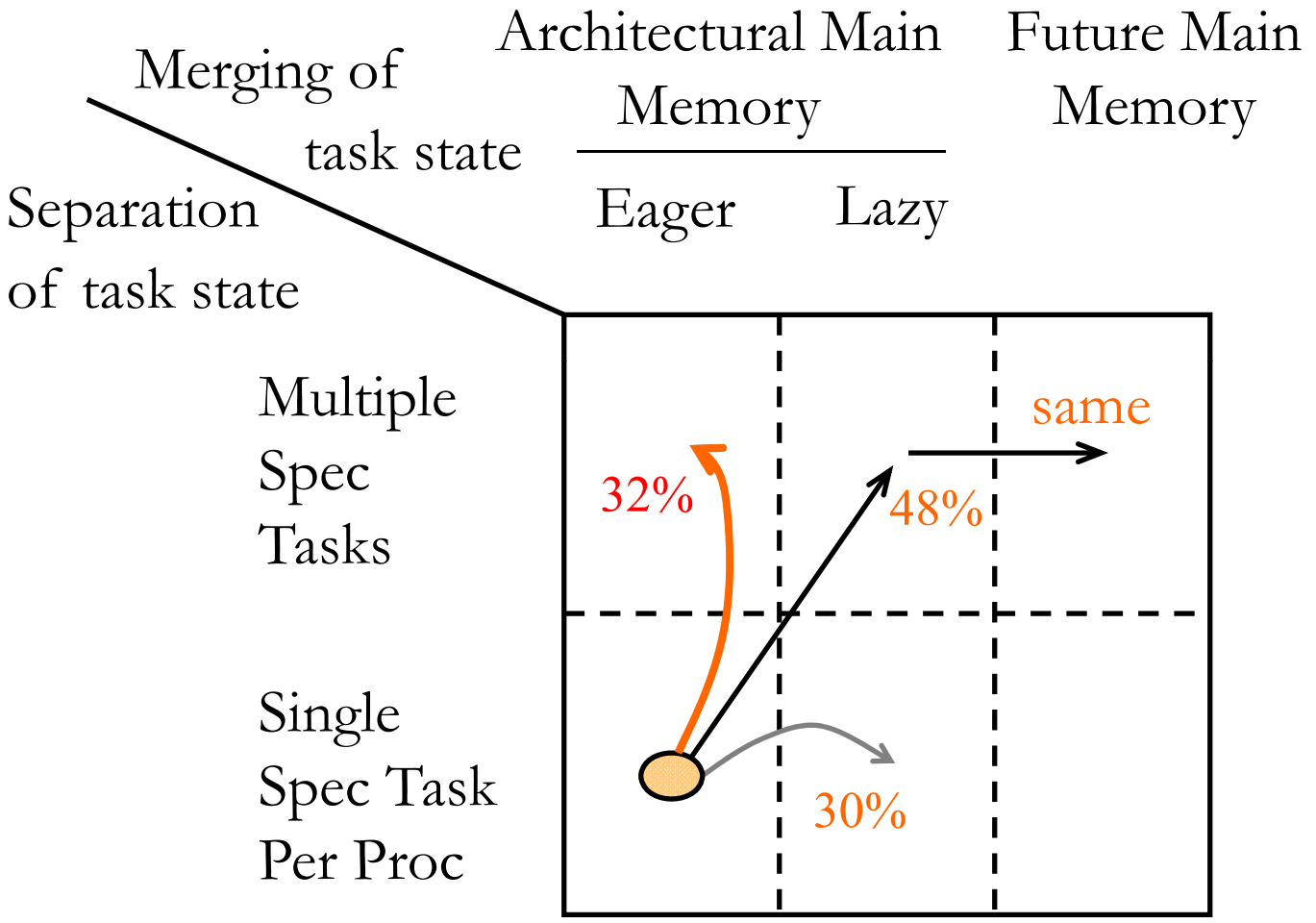


CC-NUMA: Architectural and Future Main Memory



- Future and Lazy Architectural MM have similar performance
- Future MM is better if cache pressure (P3m)
- Future MM may suffer if frequent squashes (Euler)

Summary


Merging of task state
Separation of task state
Architectural Main Memory
Future Main Memory
Eager
Lazy
Multiple Spec Tasks
Single Spec Task Per Proc
32%
48%
same
30%

Conclusions

1. Adds modest complexity

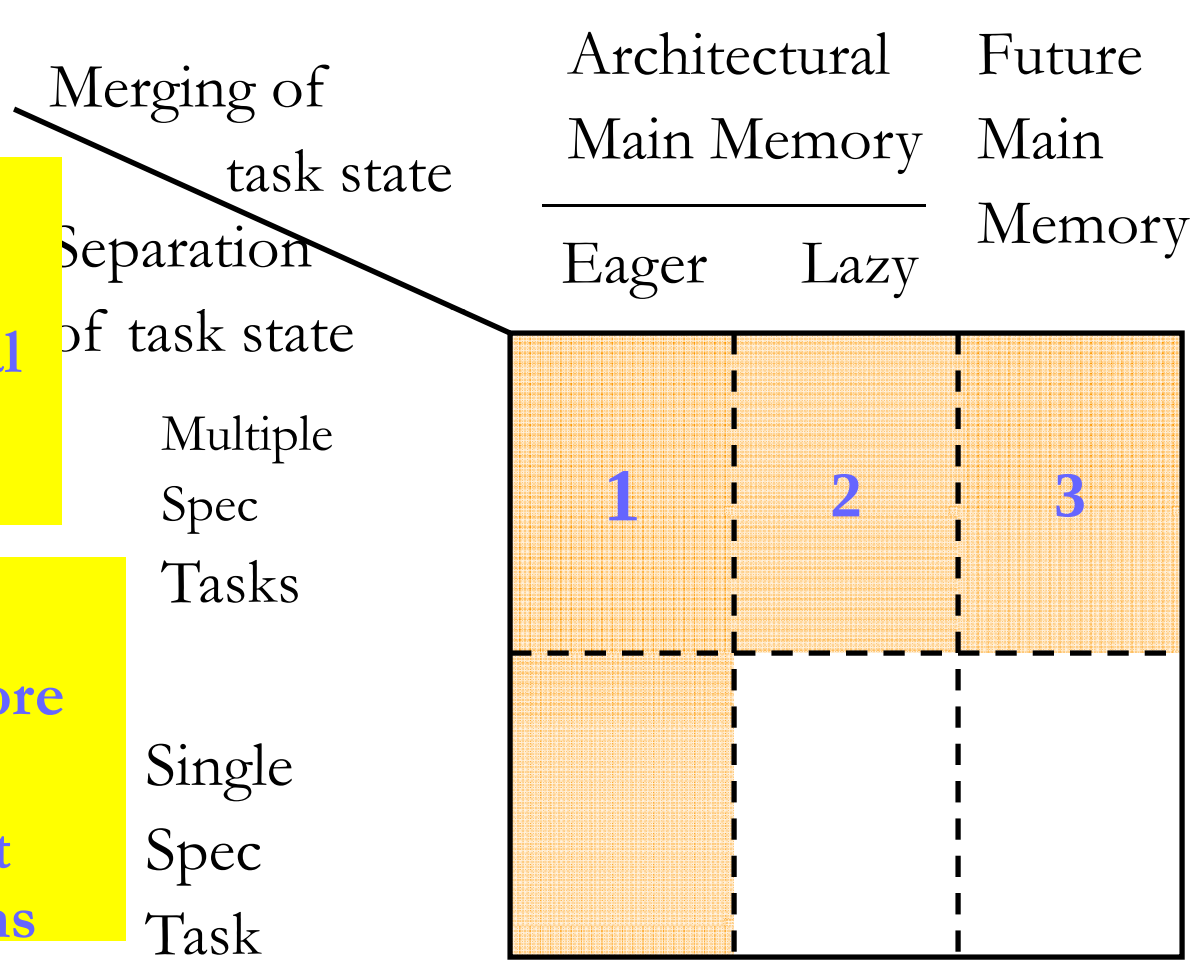
2. Additional performance gains

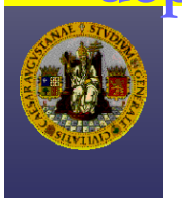
   Targets an orthogonal problem

3. Similar performance

   Makes the system more robust

   Can suffer if frequent dependence violations



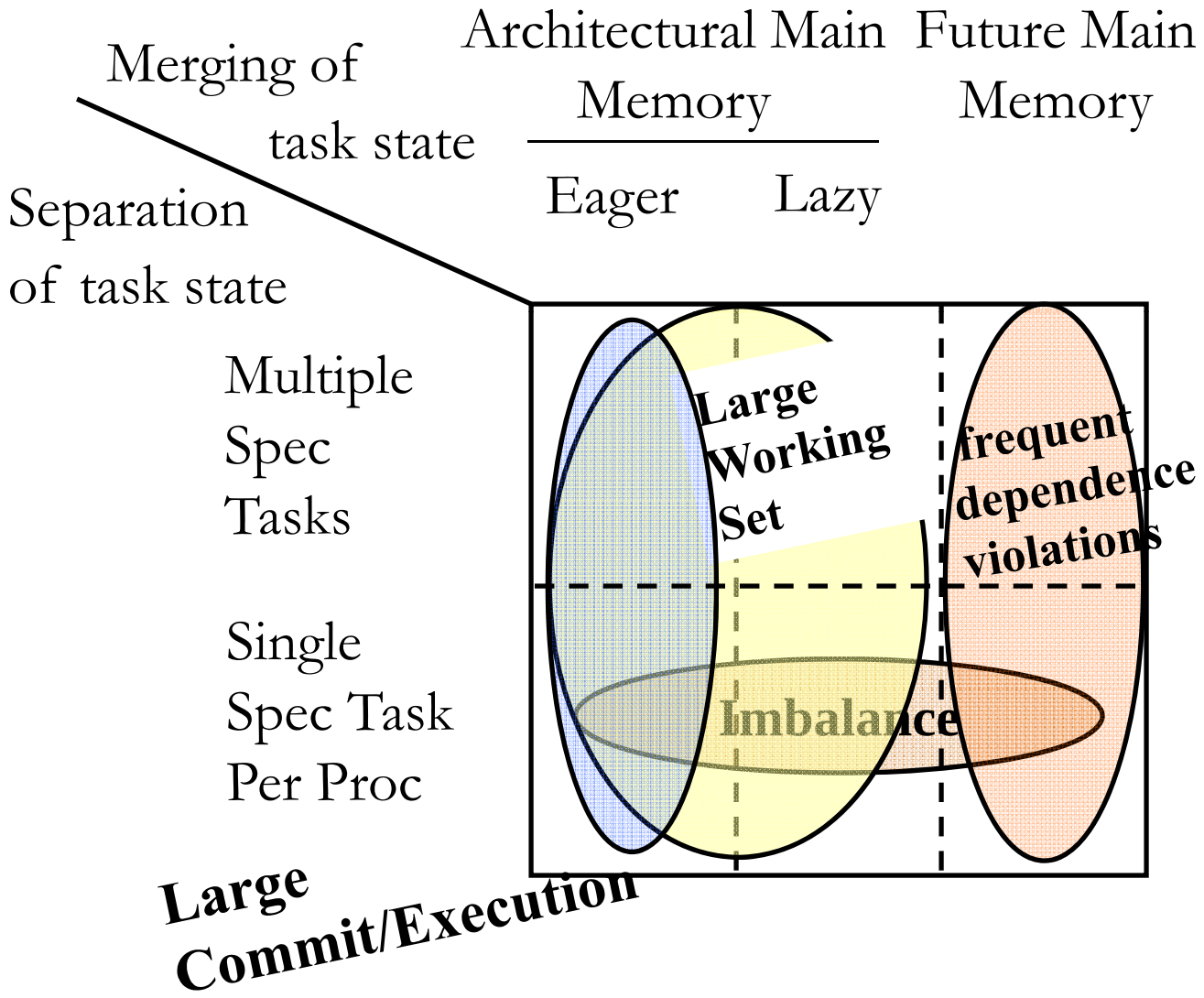


Tradeoffs in Buffering Memory State for Thread-Level Speculation in Multiprocessors

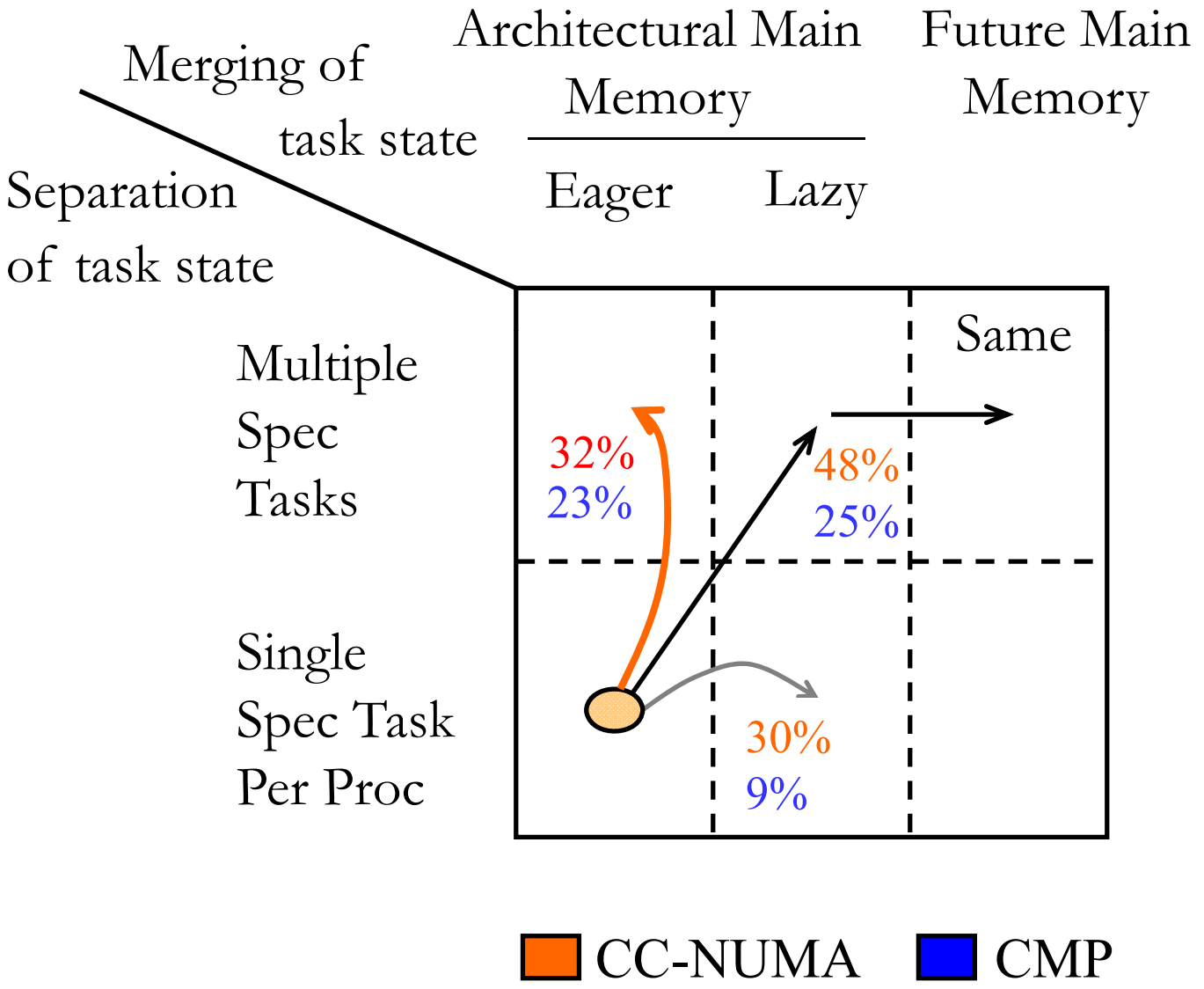
María Jesús Garzarán, M. Prvulovic[§], J. M. Llabería[*], V. Viñals, L. Rauchwerger[ψ], and J. Torrellas[§]

garzaran@cs.uiuc.edu
http://iacoma.cs.uiuc.edu/
http://www.cps.unizar.es/gaz

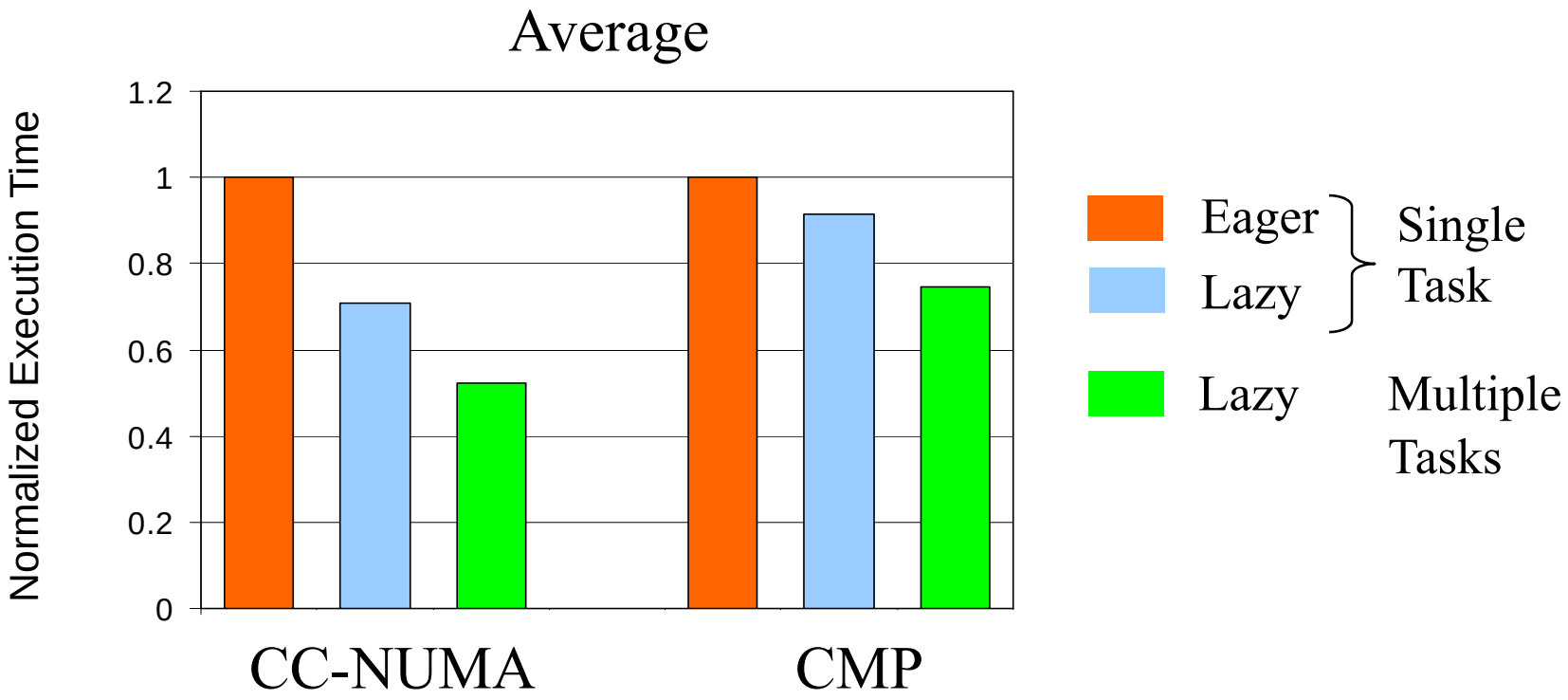
App characteristics that limit performance



Summary

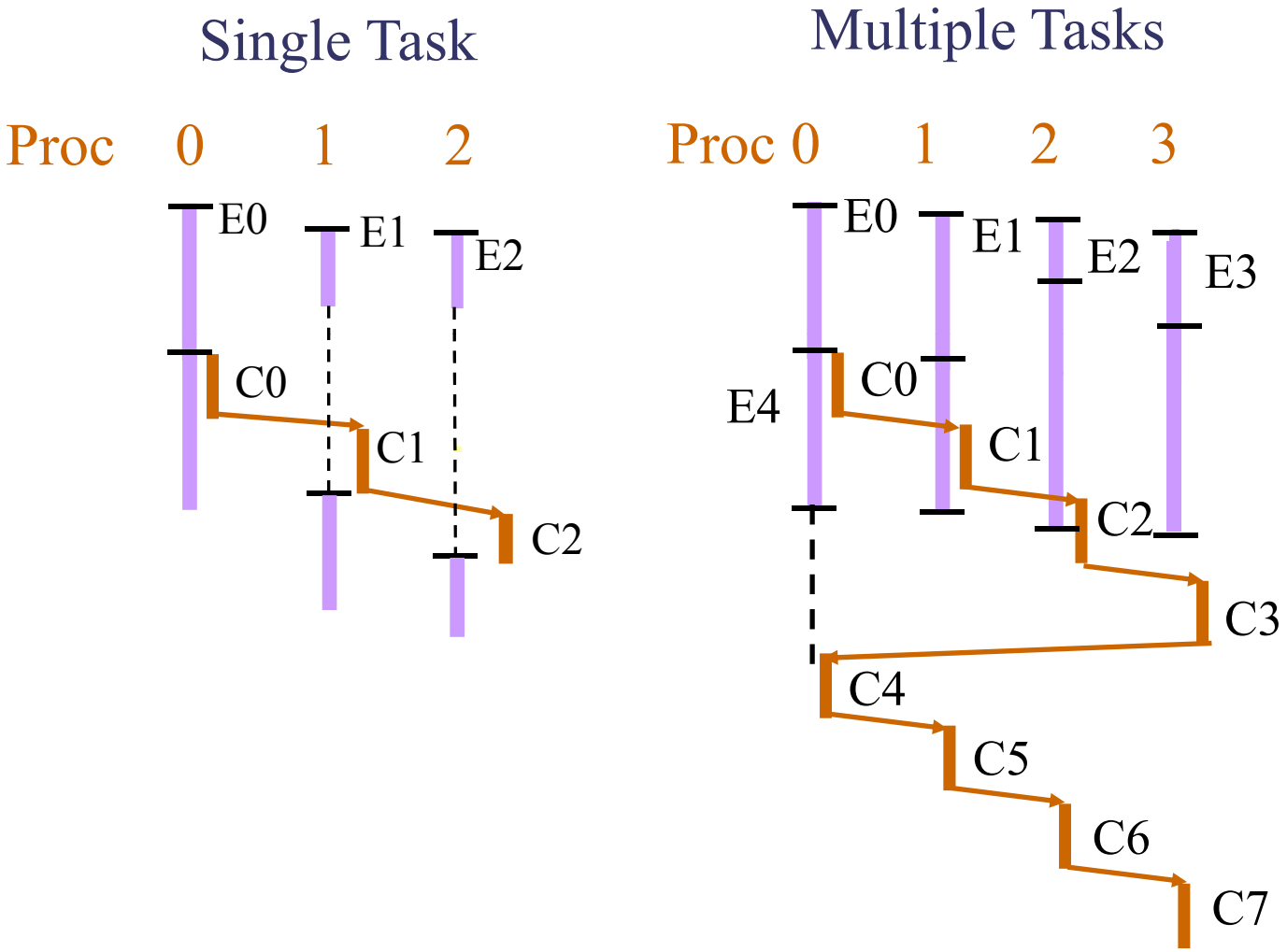

Merging of task state
Separation of task state
Architectural Main Memory
Future Main Memory
Eager
Lazy
Multiple Spec Tasks
Single Spec Task Per Proc
Same
32%
23%
48%
25%
30%
9%
CC-NUMA
CMP

CC-NUMA vs CMP: Architectural MM

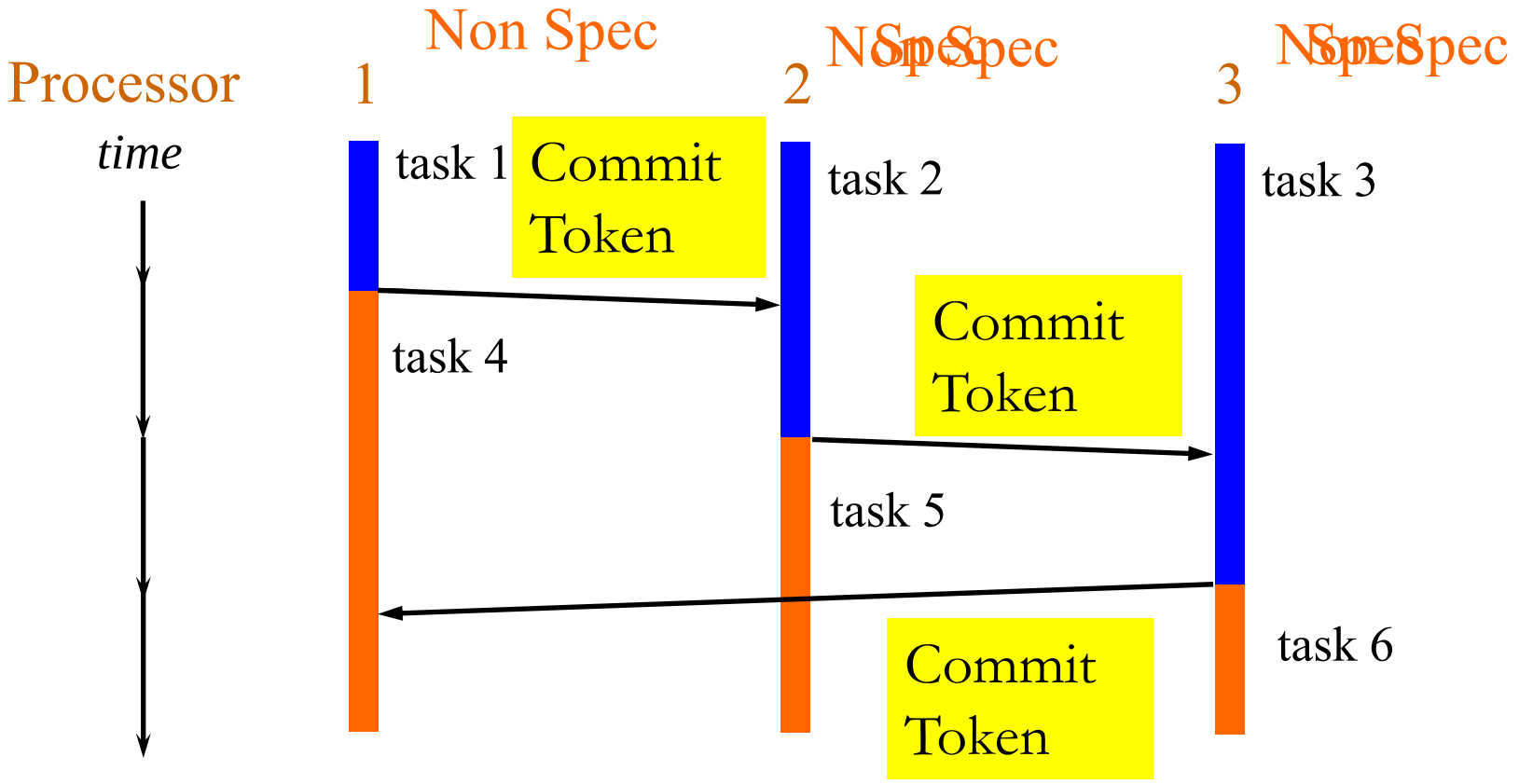


- Less differences in CMP

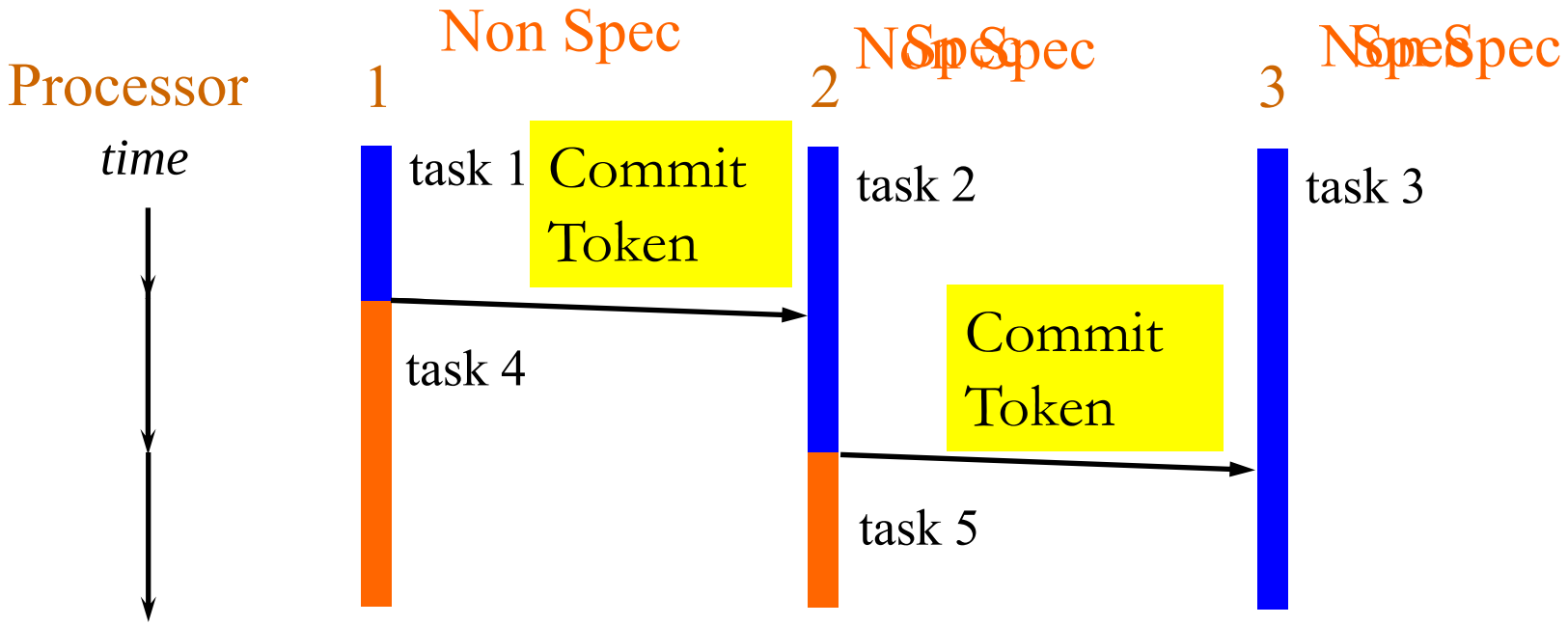
Supporting Multiple Tasks (Eager)



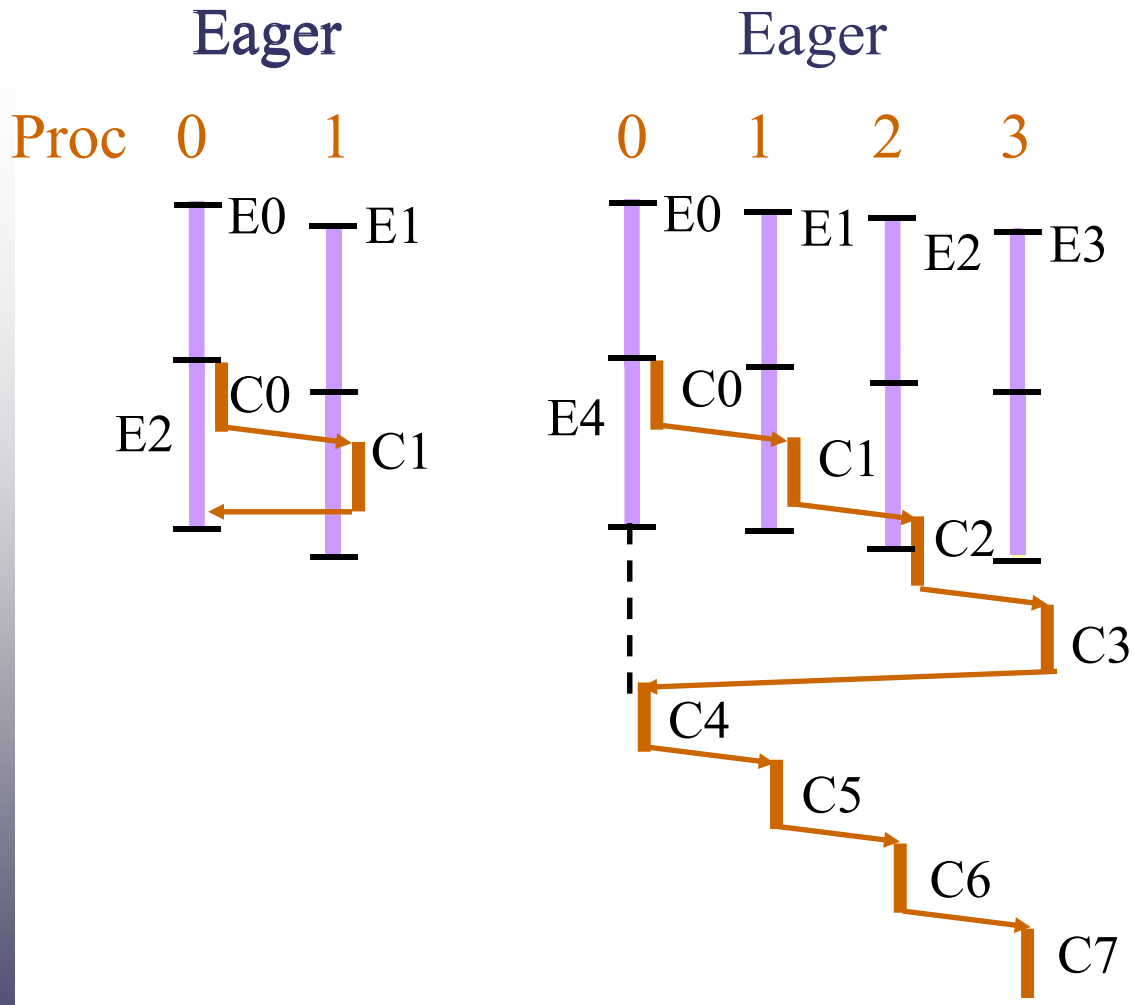
Task Execution under TLS



Task Execution under TLS



Eager vs Lazy Merging in Architectural MM

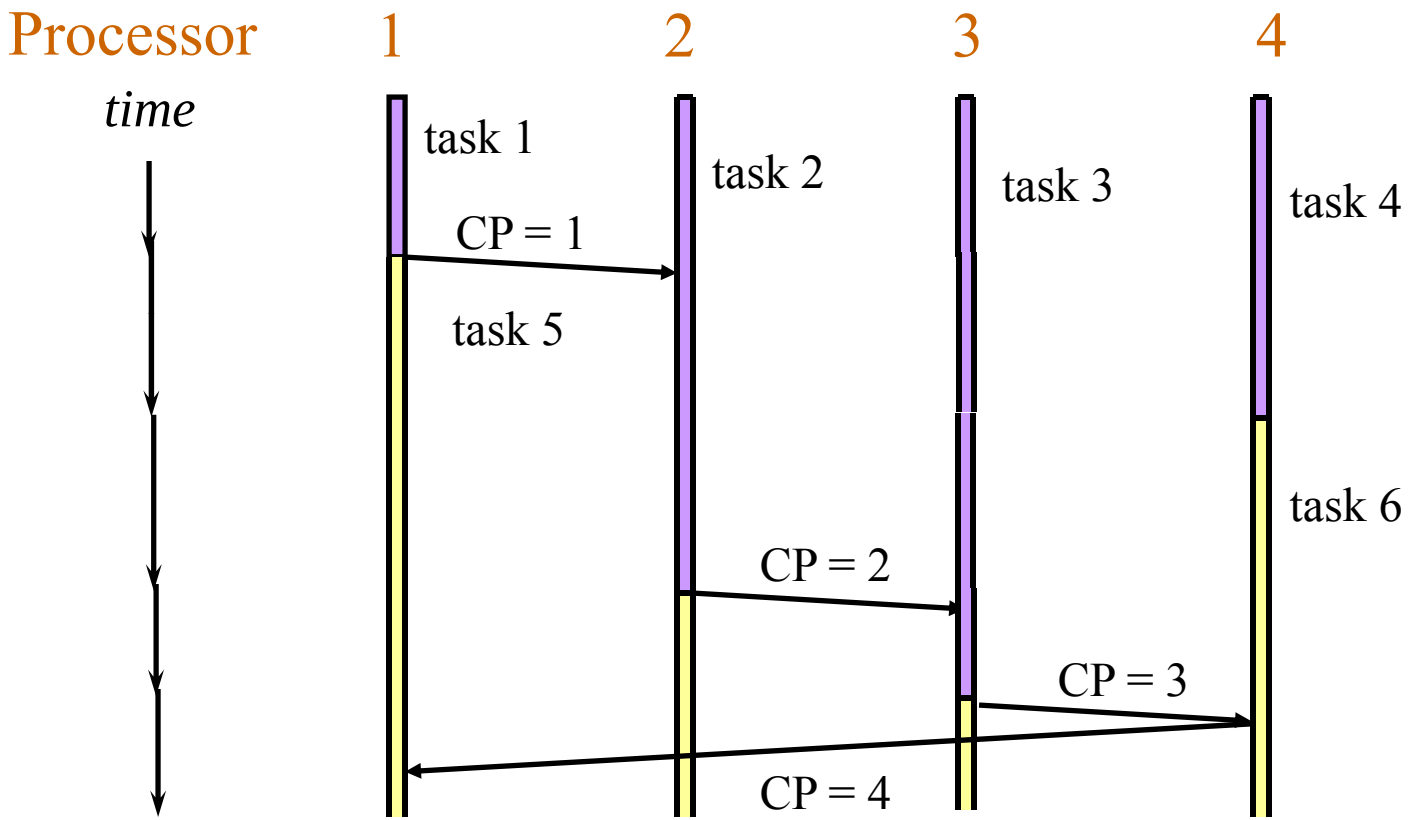


Multiple Tasks Per Processor

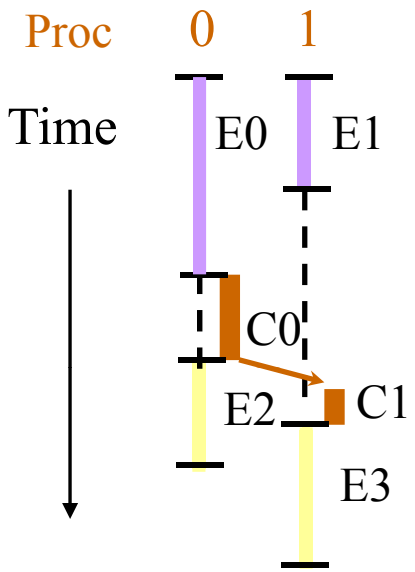
Commit wavefront appears in critical path when:

Ci * Nprocs > Ei

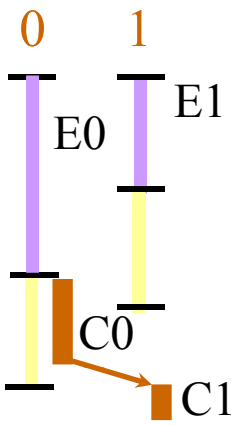
Thread-Level Speculation



Supporting Multiple Tasks



Single Task per Processor

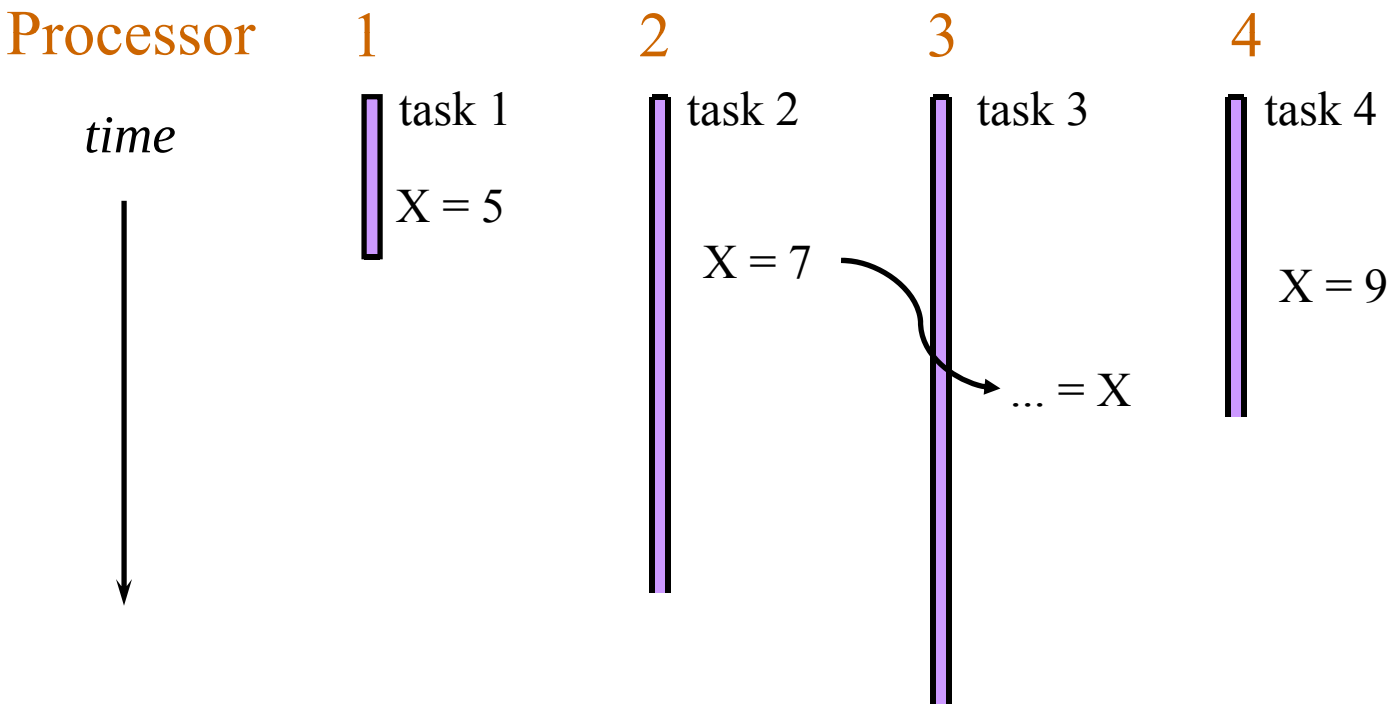


Multiple Tasks per Processor

Ei: Execution of Task i
Ci: Commit of Task i

Challenges in Buffering Speculative State

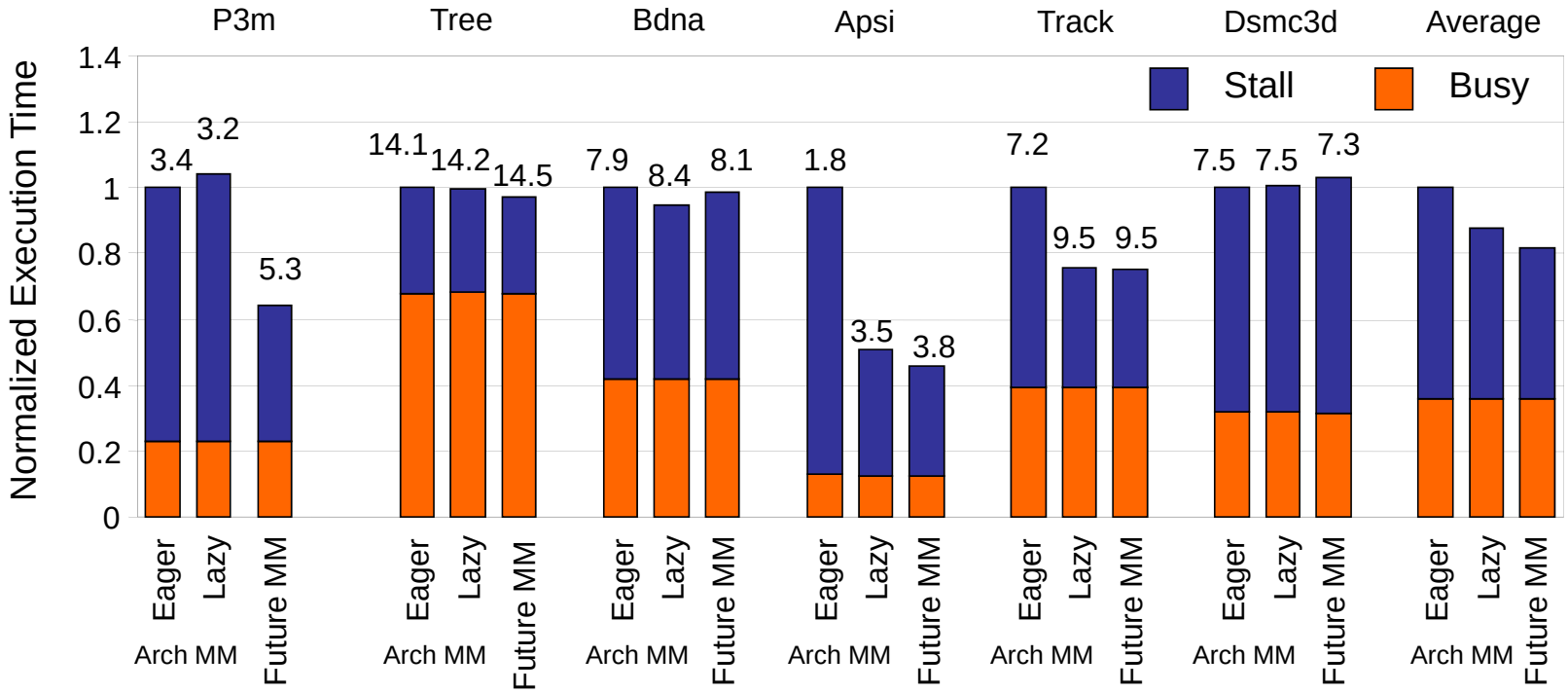
- Several versions of the same variable in the system
- Main memory needs to be correctly updated



Conclusions

- Starting from SingleT Eager Architectural Main Memory
  - Adding **multiple T&V** is more cost-effective than lazy merging:
    - Higher performance
    - Lower implementation complexity
  - With multiple T&V, **lazy merging** further increases performance
    - Fairly orthogonal impact
  - Further supporting **Future MM**:
    - Modest increase in performance
    - Can suffer if frequent squashes
    - Adds complexity

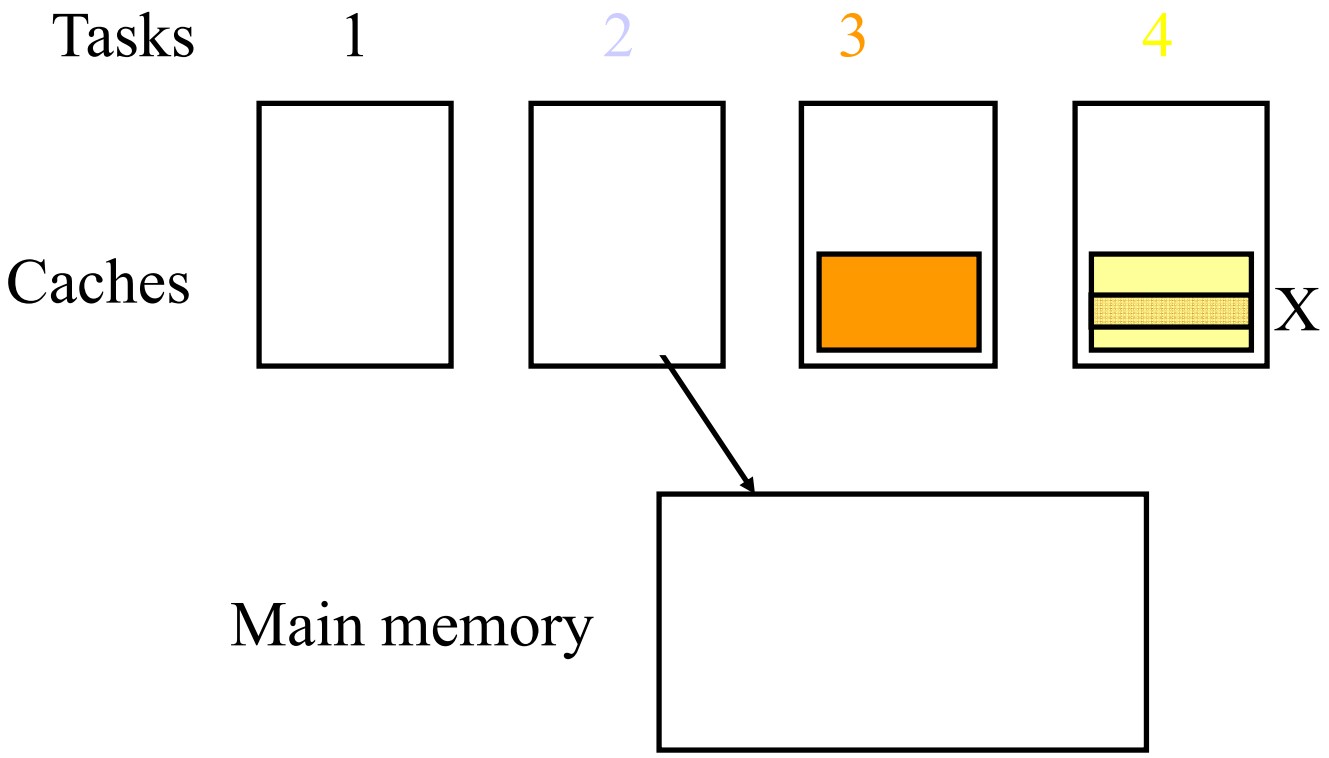
Architectural and Future MM



- Both Future and Lazy Architectural MM have similar perf
- Future MM is better if cache pression (P3m)
- Future MM may suffer if frequent squashes (Euler)

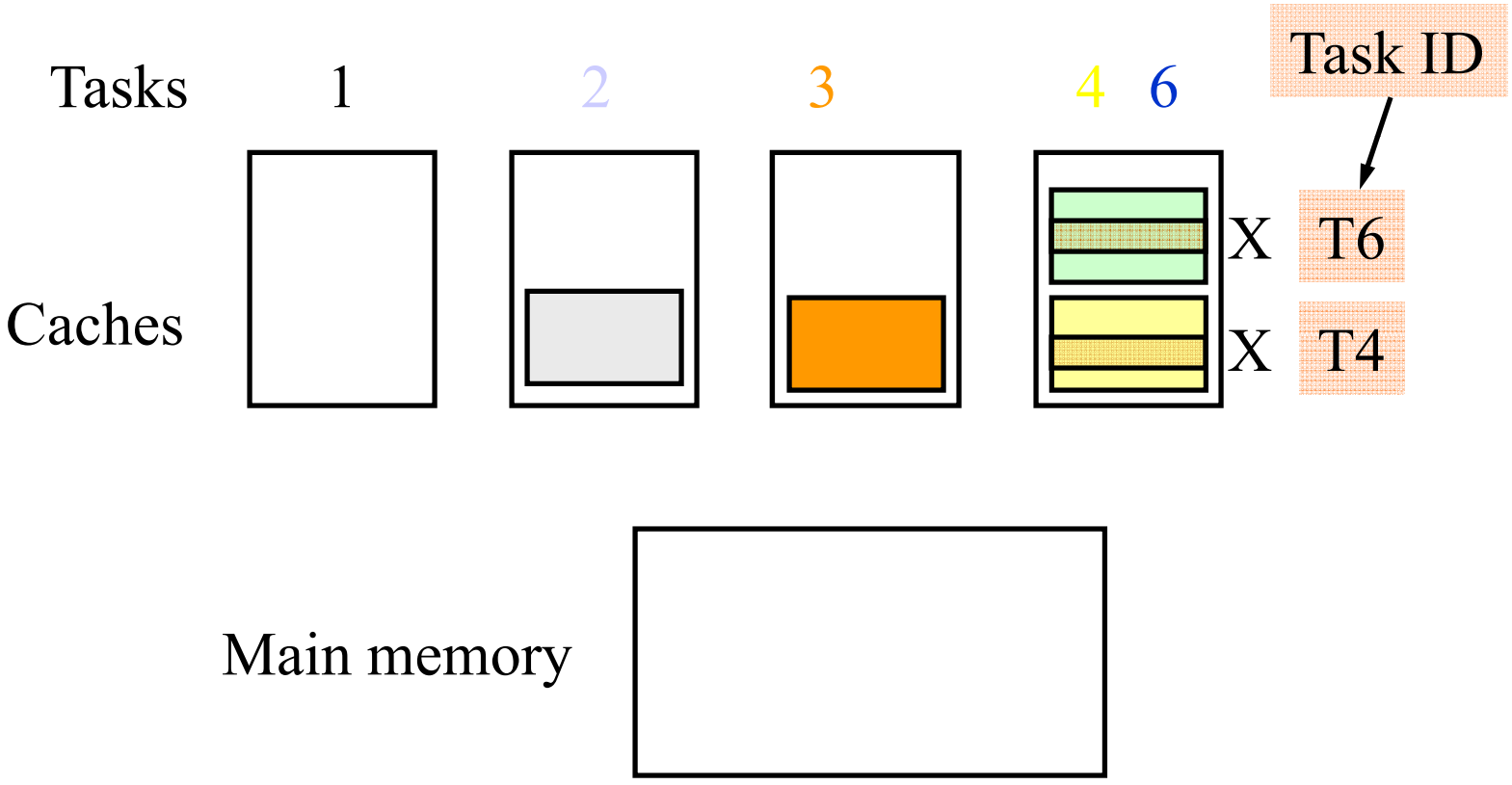
Challenges in Speculative State Buffering - II

- Several versions of the same variable in the system
- Main memory needs to be updated in the correct order

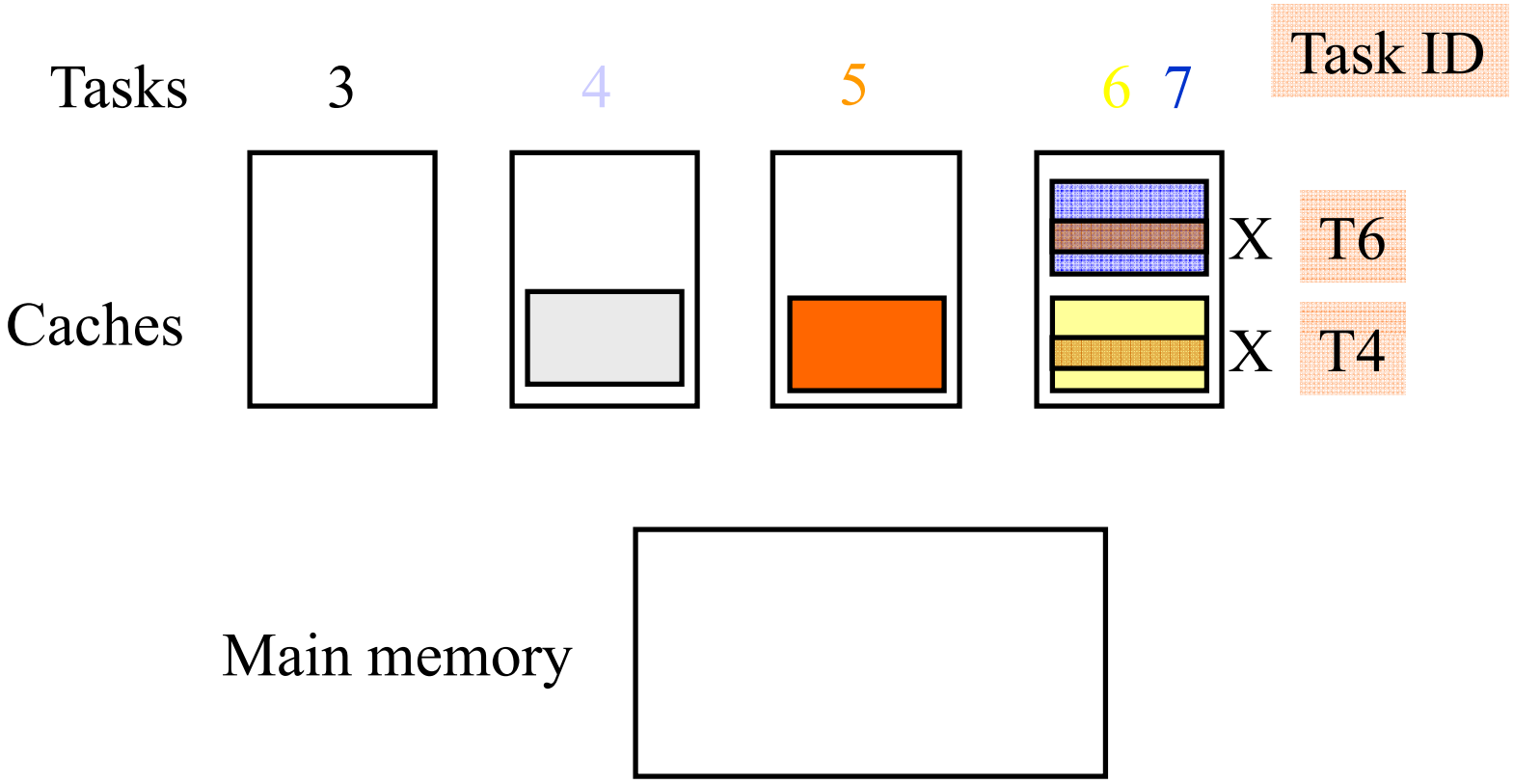


Challenges in Speculative State Buffering - III

- Several versions of the same variable in the same cache



Separation of Task State



Application Characteristics*

Application	Average # Speculative Tasks in the System	Average Written Footprint per Task	
		Total (KB)	Priv. (%)
P3m	800.0	1.7	87.9
Tree	24.0	0.9	99.5
Bdna	25.6	23.7	99.4
Apsi	28.8	20.0	60.0
Track	20.8	2.3	0.6
Dsmc3d	17.6	0.8	0.5

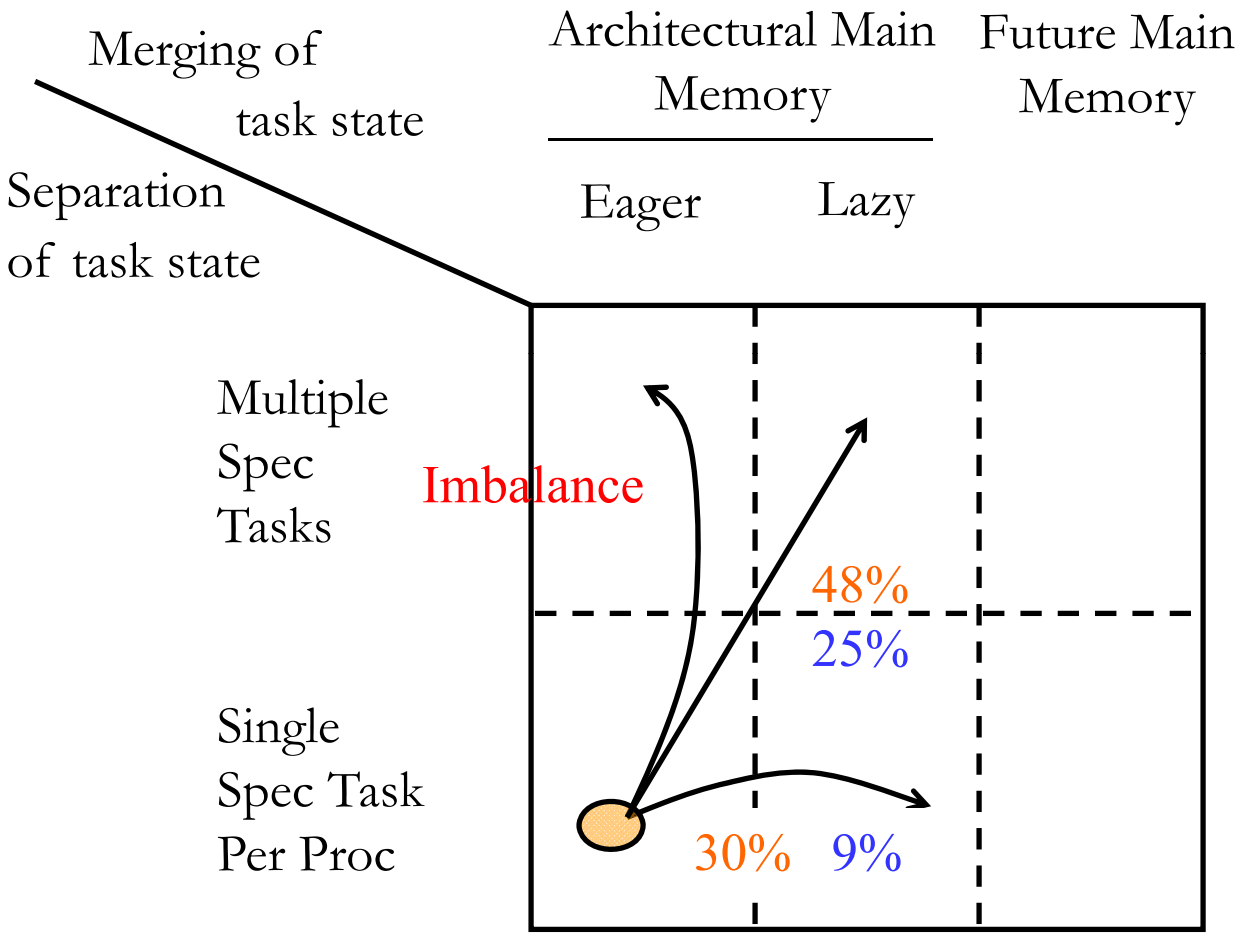
*Executing in a DSM with 16 processors

Goal

- Execute hard to analyze codes in parallel
  - Pointers
  - Indirectly-indexed structures
  - Possible interprocedural dependences
  - Input-dependent patterns

```
for(i=0;i<n;i++){
  … = A[B[i]] …
      …
  A[C[i]] = …
}
```

Recap


Merging of task state
Separation of task state
Architectural Main Memory
Future Main Memory
Eager
Lazy
Multiple Spec Tasks
Single Spec Task Per Proc
Imbalance
48%
25%
30%
9%