# Eliminating Squashes Through Learning Cross-Thread Violations in Speculative Parallelization for Multiprocessors

Marcelo Cintra and Josep Torrellas

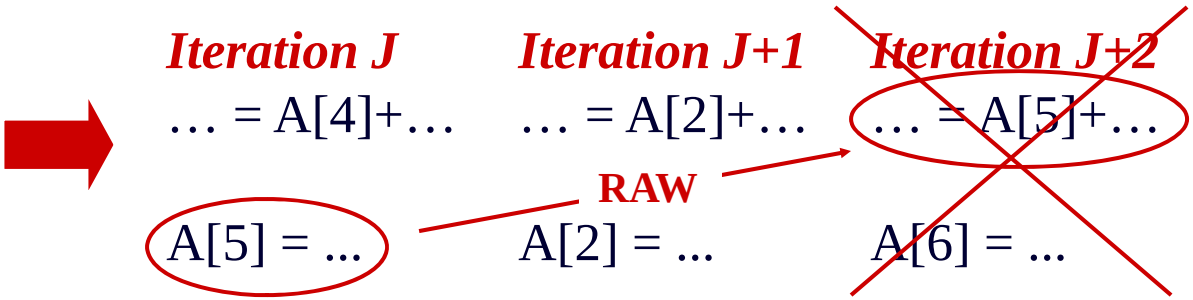
University of Edinburgh
http://www.dcs.ed.ac.uk/home/mc

University of Illinois
at Urbana-Champaign
http://iacoma.cs.uiuc.edu


# Speculative Parallelization

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

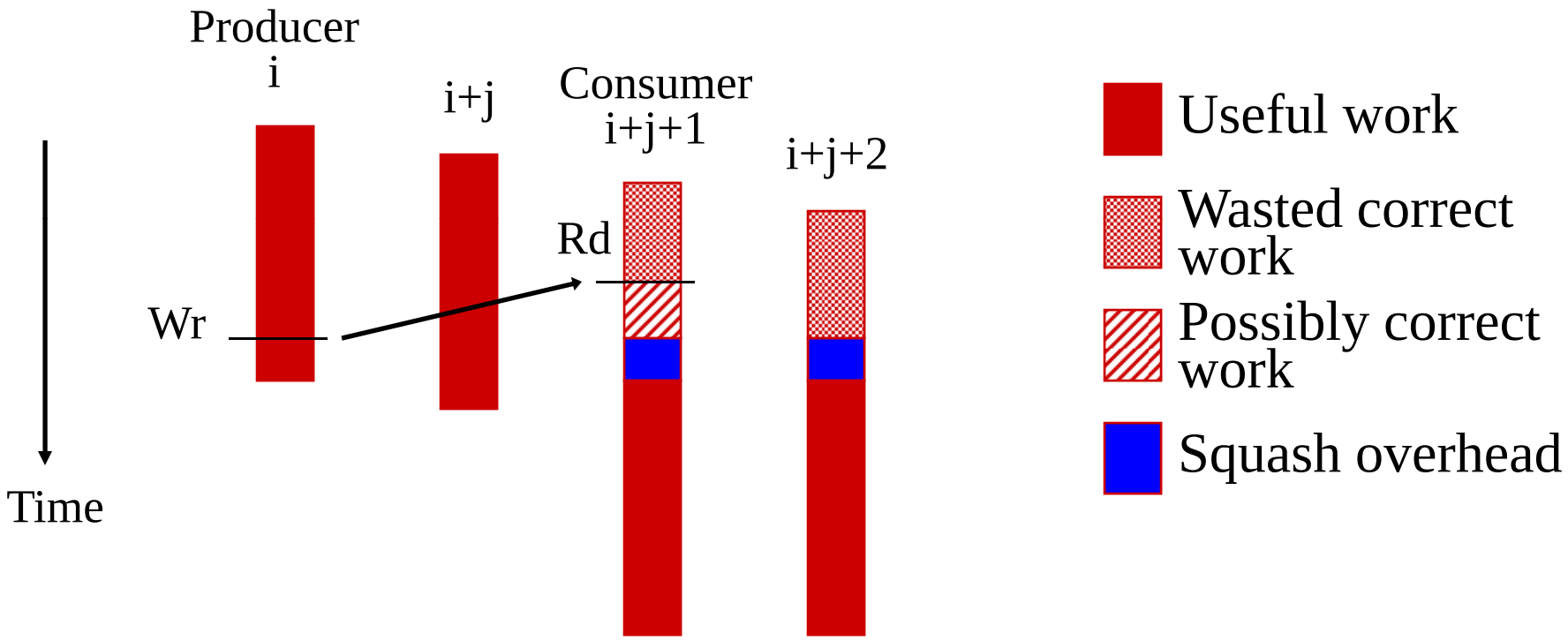
```
for(i=0; i<100; i++) {
   … = A[L[i]]+…

   A[K[i]] = …
}
```




# Squashing in Speculative Parallelization

- Speculative Parallelization: Threads may get squashed
- Dependence violations are statically unpredictable
- False sharing may cause further squashes

# Squashing




Squashing is very costly

# Contribution: Eliminate Squashes

- Framework of hardware mechanisms to eliminate squashes
- Based on learning and prediction of violations
- Improvement: average of 4.3 speedup over plain squash-and-retry for 16 processors

# Outline

- Motivation and Background
- Overview of Framework
- Mechanisms
- Implementation
- Evaluation
- Related Work
- Conclusions

# Types of Dependence Violations

| *Type* | *Avoiding Squashes* | *Mechanism* |
|---|---|---|
| **No dependence** | **Disambiguate line addresses** | **Plain Speculation** |

# Types of Dependence Violations

| *Type* | *Avoiding Squashes* | *Mechanism* |
|---|---|---|
| **No dependence** | **Disambiguate line addresses** | **Plain Speculation** |
| **False** | **Disambiguate word addresses** | ***Delay & Disambiguate*** |

# Types of Dependence Violations

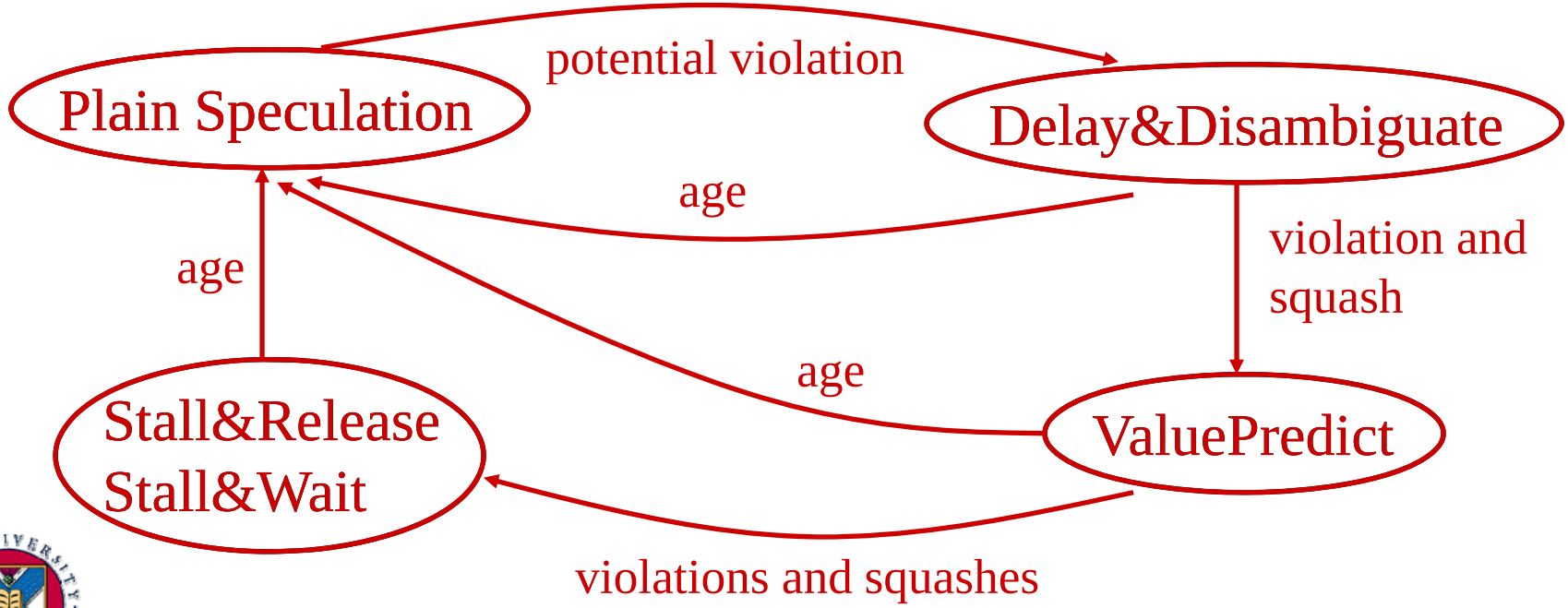
| *Type* | *Avoiding Squashes* | *Mechanism* |
|---|---|---|
| No dependence | Disambiguate line addresses | Plain Speculation |
| False | Disambiguate word addresses | *Delay & Disambiguate* |
| Same-word, predictable value | Compare values produced and consumed | *Value Predict* |

# Types of Dependence Violations

| *Type* | *Avoiding Squashes* | *Mechanism* |
|---|---|---|
| No dependence | Disambiguate line addresses | Plain Speculation |
| False | Disambiguate word addresses | *Delay & Disambiguate* |
| Same-word, predictable value | Compare values produced and consumed | *Value Predict* |
| Same-word, unpredictable value | Stall thread, release when safe | *Stall&Release Stall&Wait* |

# Learning and Predicting Violations

- Monitor violations at directory
- Remember data lines causing violations
- Count violations and choose mechanism



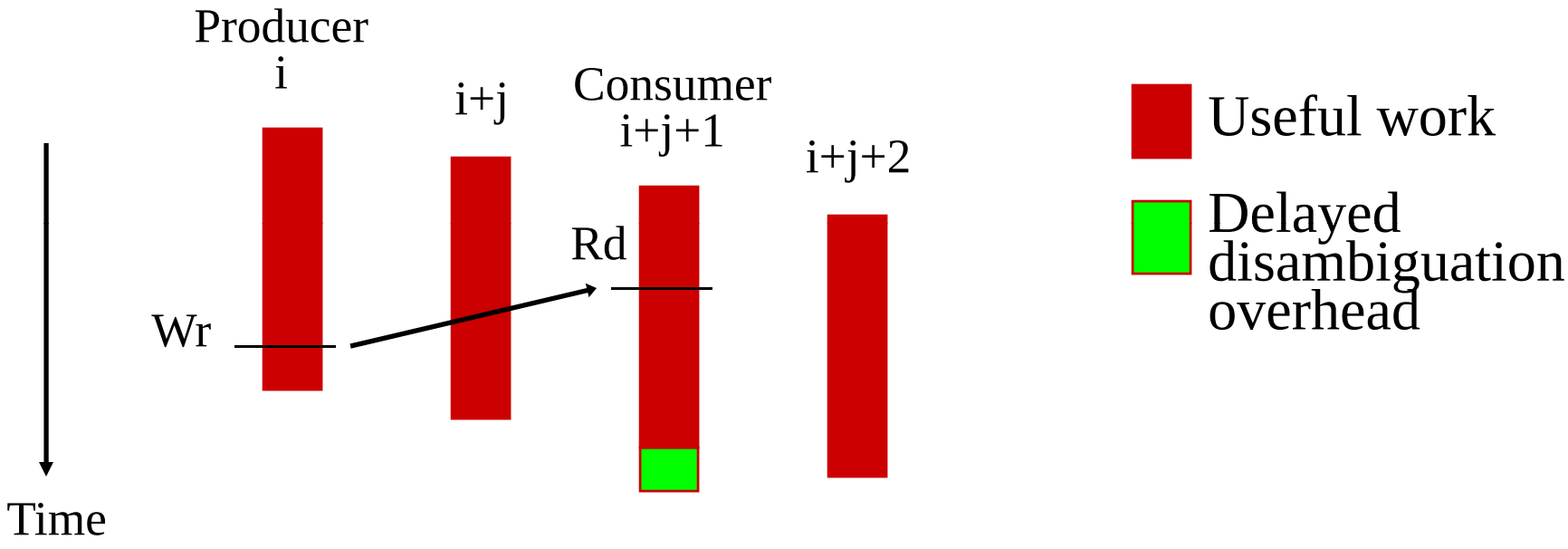

# Outline

- Motivation and Background
- Overview of Framework
- Mechanisms
- Implementation
- Evaluation
- Related Work
- Conclusions

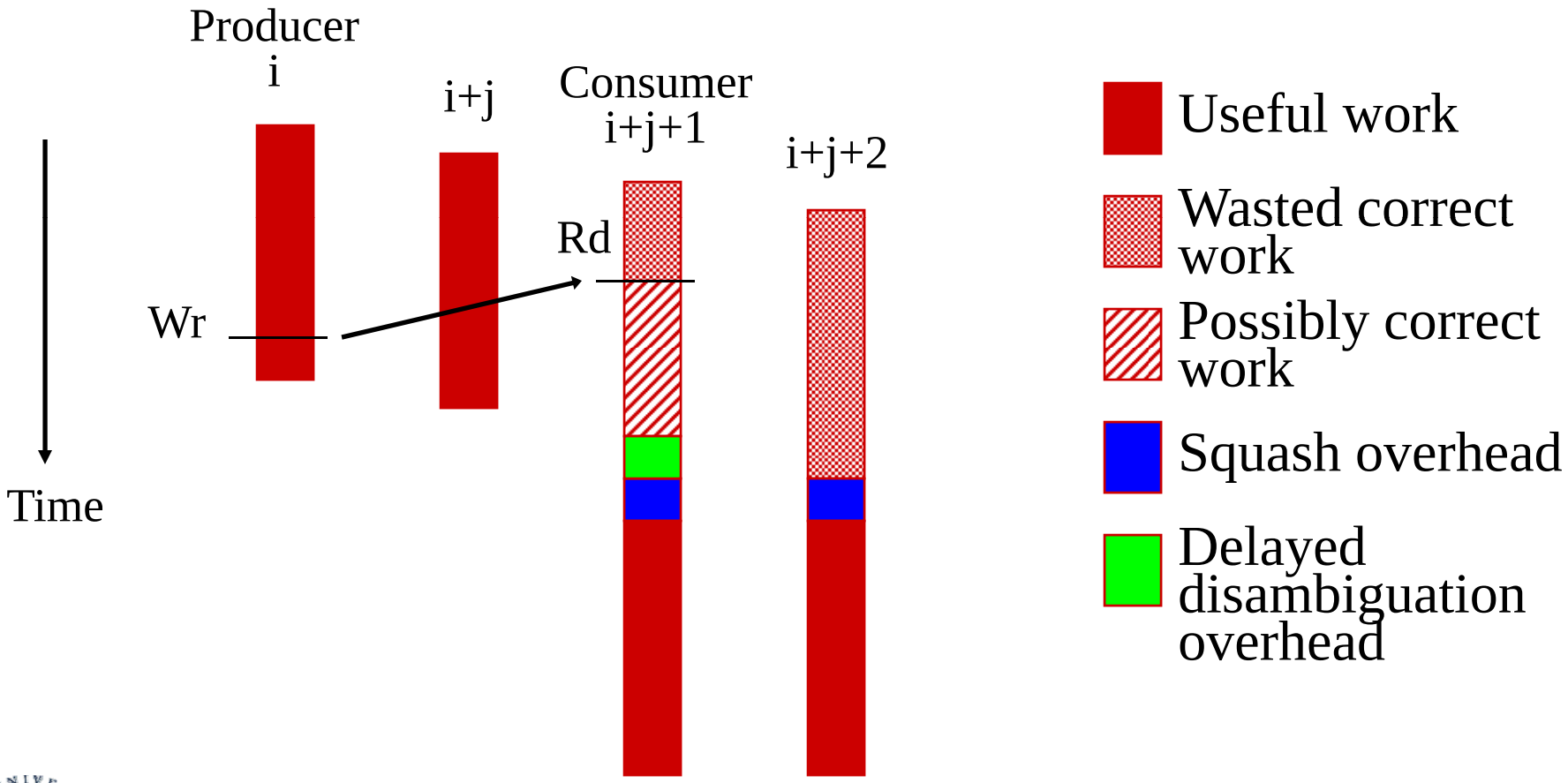
# Delay&Disambiguate

- Assume potential violation is false
- Let speculative read proceed
- Remember unresolved potential violation
- Perform a *late* or *delayed* per-word disambiguation when consumer thread becomes non-speculative
  - No per-word access information at directory
  - No word addresses on memory operations
- Squash if same-word violation

# Delay&Disambiguate (Successful)




# Delay&Disambiguate (Unsuccessful)



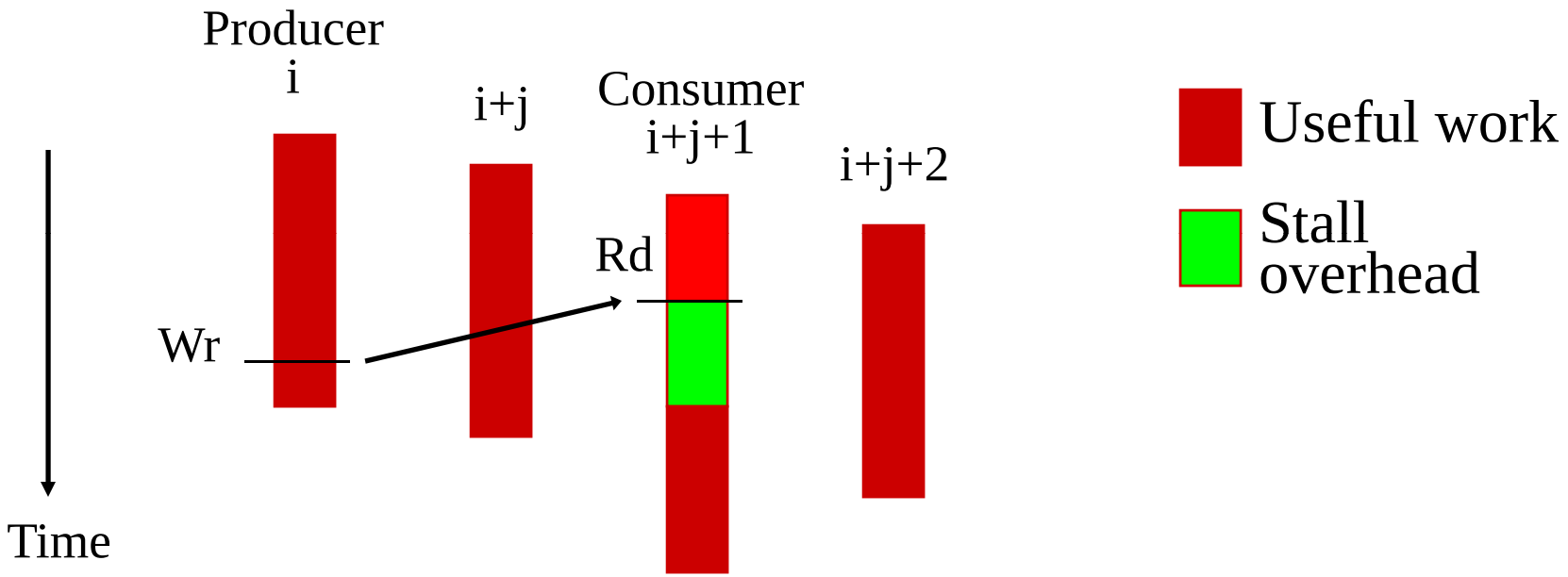

# ValuePredict

- Predict value based on past observed values
  - Assume value is the same as last value written (*last-value prediction, value reuse, or silent store*)
  - More complex predictors are possible
- Provide predicted value to consumer thread
- Remember predicted value
- Compare predicted value against correct value when consumer thread becomes non-speculative
- Squash if mispredicted value

# Stall&Release

- Stall consumer thread when attempting to read data
- Release consumer thread when a predecessor commits modified copy of the data to memory
  - Provided no intervening thread has a modified version
- Squash released thread if a later violation is detected

# Stall&Release (Successful)



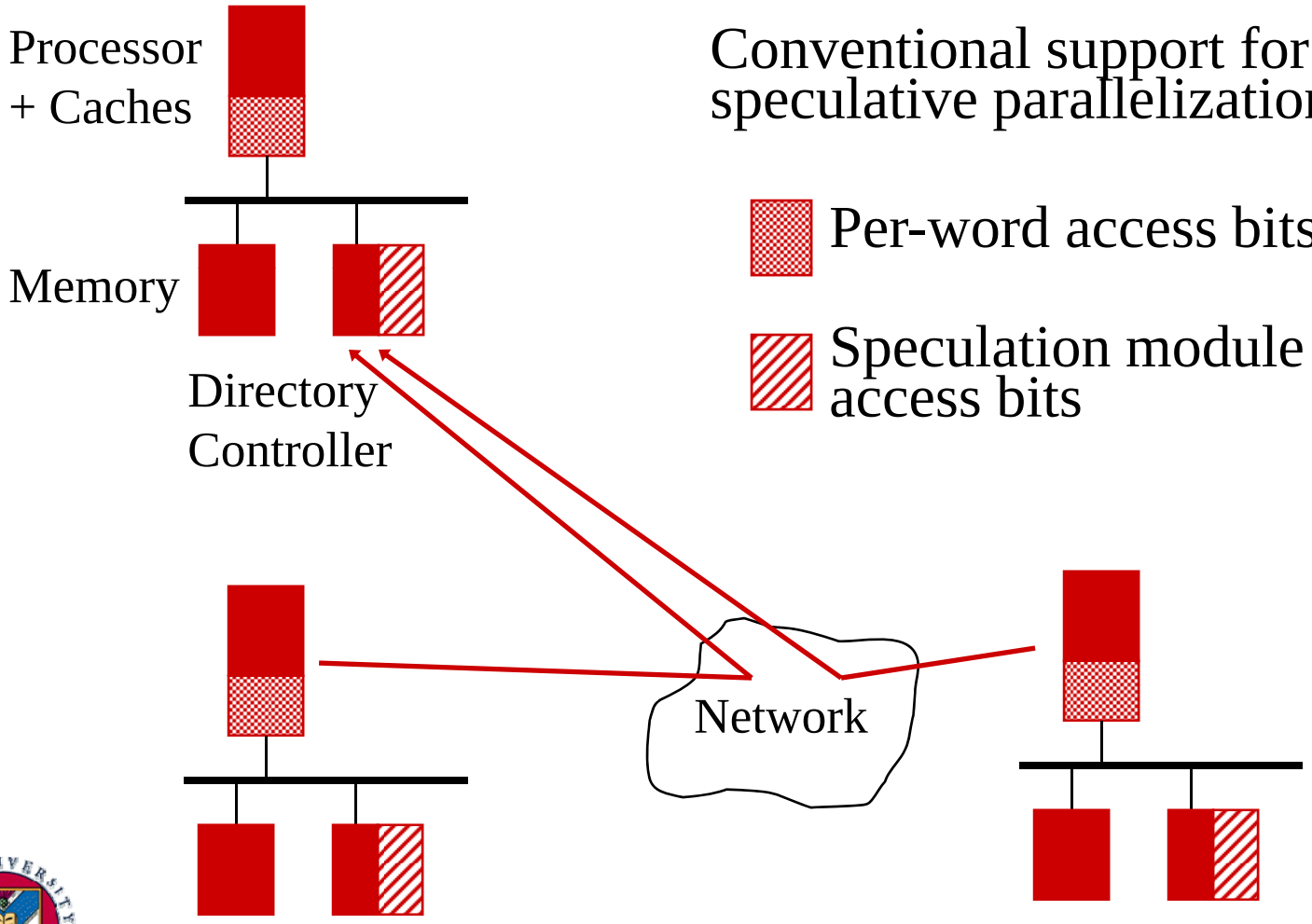

# Stall&Wait

- Stall consumer thread when attempting to read data
- Release consumer thread only when it becomes non-speculative

# Outline

- Motivation and Background
- Overview of Framework
- Mechanisms
- Implementation
- Evaluation
- Related Work
- Conclusions

# Baseline Architecture




Conventional support for speculative parallelization:

- Per-word access bits in caches
- Speculation module with per-line access bits

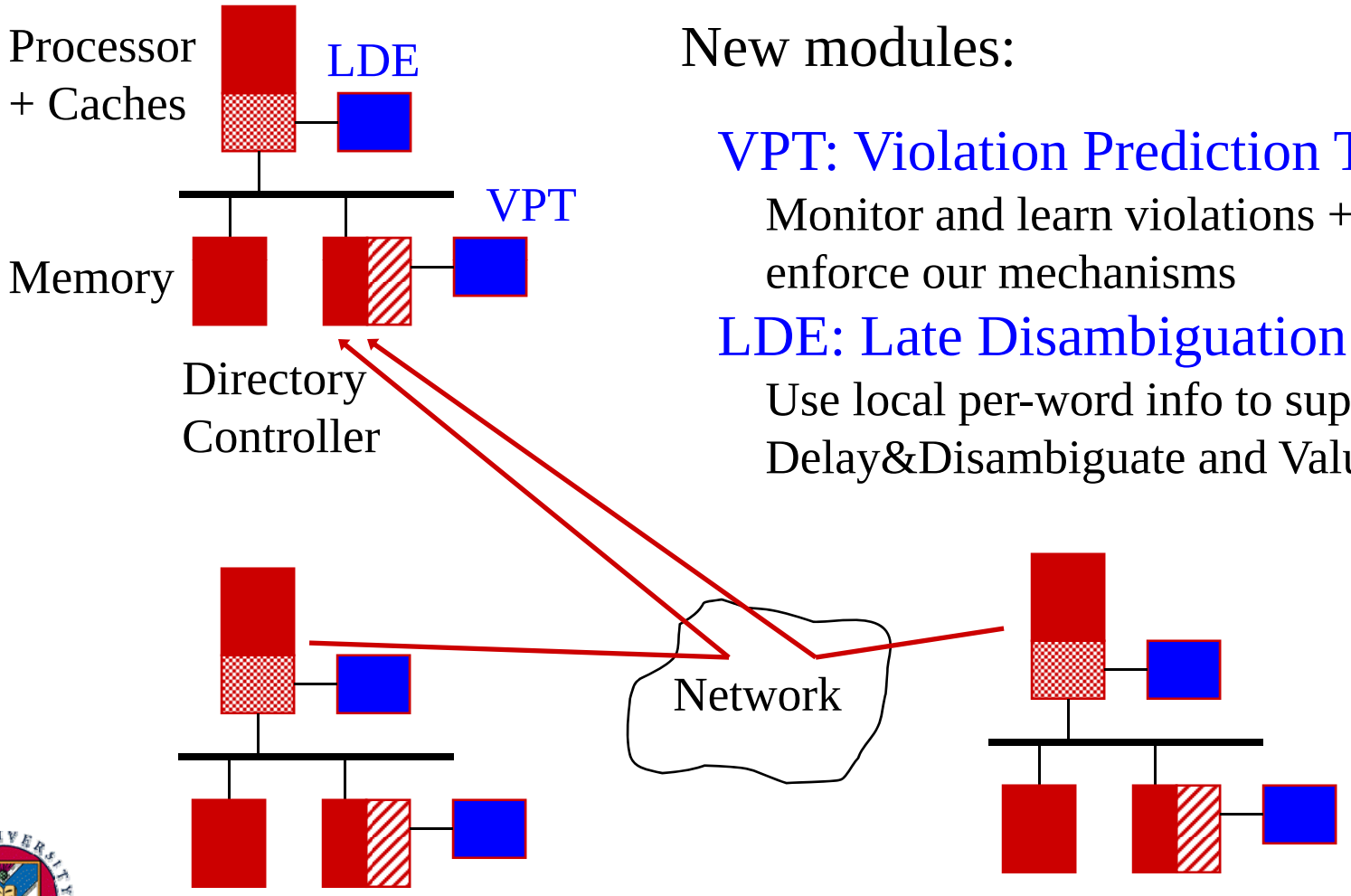
# Speculation Module

- One entry per memory line speculatively touched
- Load and Store bits per line per thread
- Mapping of threads to processors and ordering of threads

| Line Tag | Valid Bit | Load Bits | Store Bits |
|---|---|---|---|
| | | | |
| | | | |
| … | … | … | … |
| | | | |

Global Memory Disambiguation Table (GMDT)
(Cintra, Martínez, and Torrellas – ISCA'00)

# Enhanced Architecture




New modules:

VPT: Violation Prediction Table

Monitor and learn violations + enforce our mechanisms

LDE: Late Disambiguation Engine

Use local per-word info to support the Delay&Disambiguate and ValuePredict

# Violation Prediction Table (VPT)

- Entries for lines recently causing potential violations
- Appended to every row in the GMDT

| Line Tag | Valid Bit | Load Bits | Store Bits | Valid Bit | State Bits | Squash Counter | SavedSquash Bit |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| … | … | … | … | … | … | … | … |
| | | | | | | | |

# VPT Pending Transactions Buffer

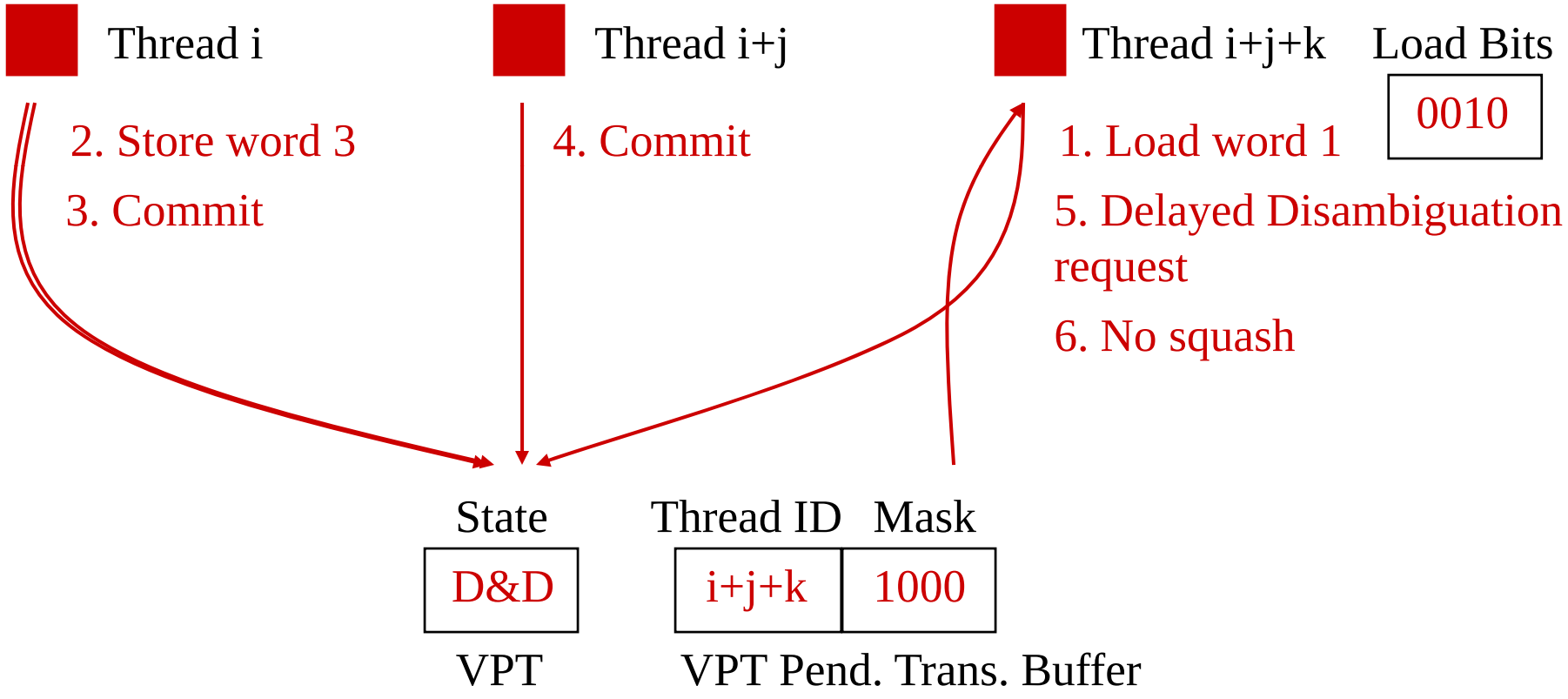
- Entries for pending transactions on VPT lines

| Line Tag | Valid Bit | Thread ID | State Bits | Modified Mask | Predicted Line Value |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| … | … | … | … | … | … |
| | | | | | |

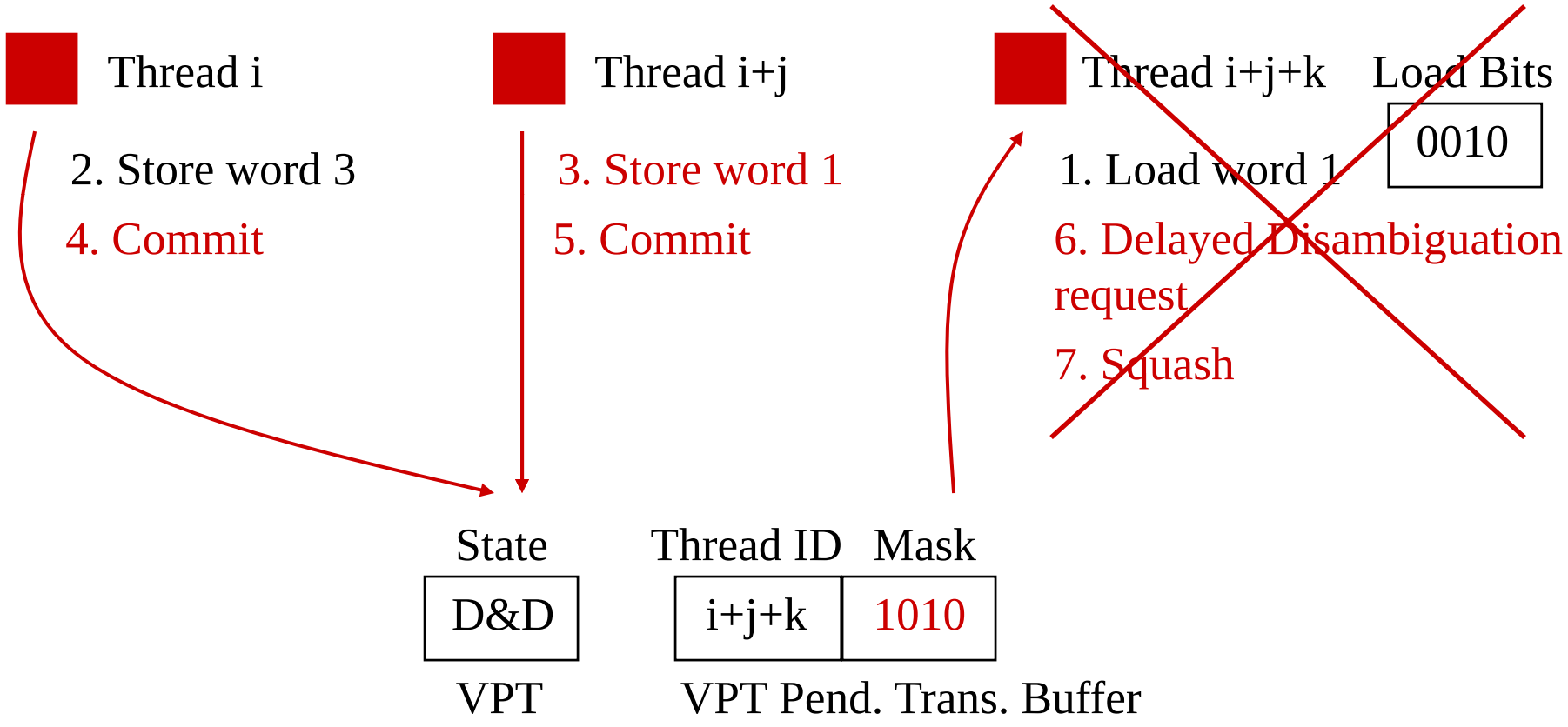
2K-entry VPT: < 2Kbytes

128-entry Pend. Trans. Buffer : < 10Kbytes

# Operation under Delay&Disambiguate



# Operation under Delay&Disambiguate



# Outline

- Motivation and Background
- Overview of Framework
- Mechanisms
- Implementation
- Evaluation
- Related Work
- Conclusions

# Simulation Environment

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

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

  CMP:  4 x (processor + 32K  L1)

  Processor: 4-issue, dynamic

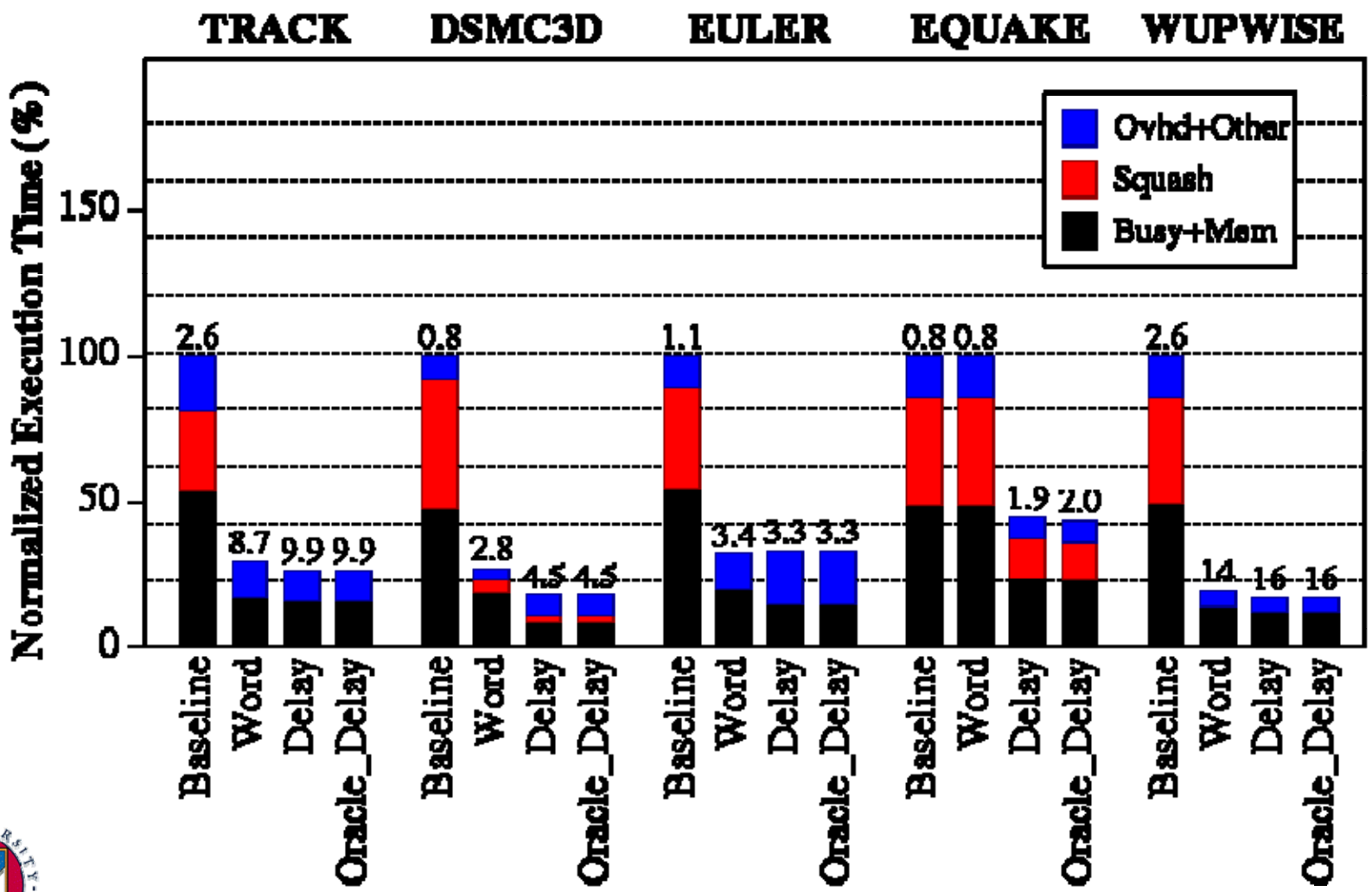
# Applications

- Applications dominated by non-analyzable loops with cross-iteration dependences → average of 60% of sequential time
- TRACK (PERFECT)

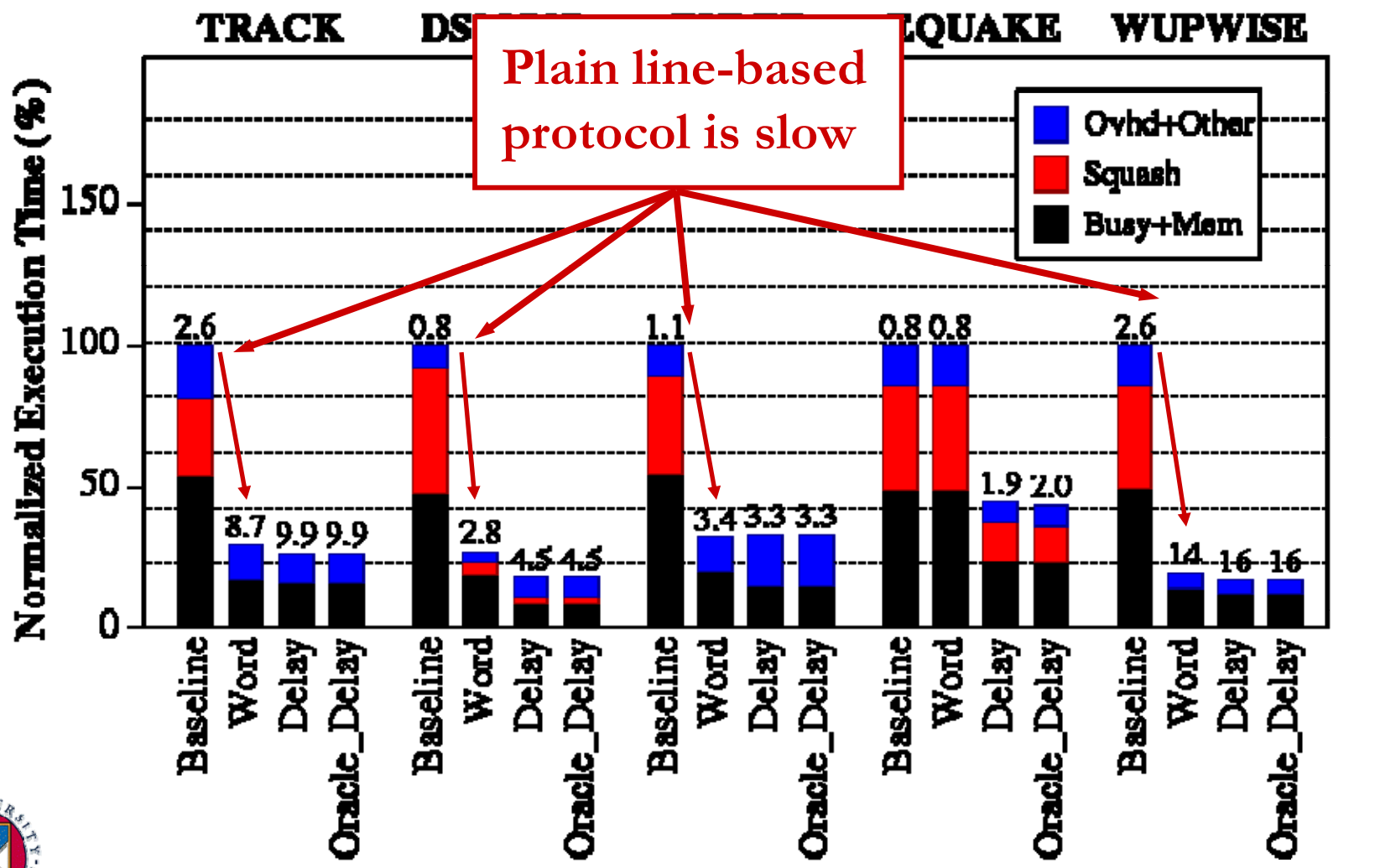
  EQUAKE, WUPWISE (SPECfp2000)

  DSMC3D, EULER (HPF2)
- Non-analyzable loops and accesses identified by the Polaris parallelizing compiler
- Results shown for the non-analyzable loops only

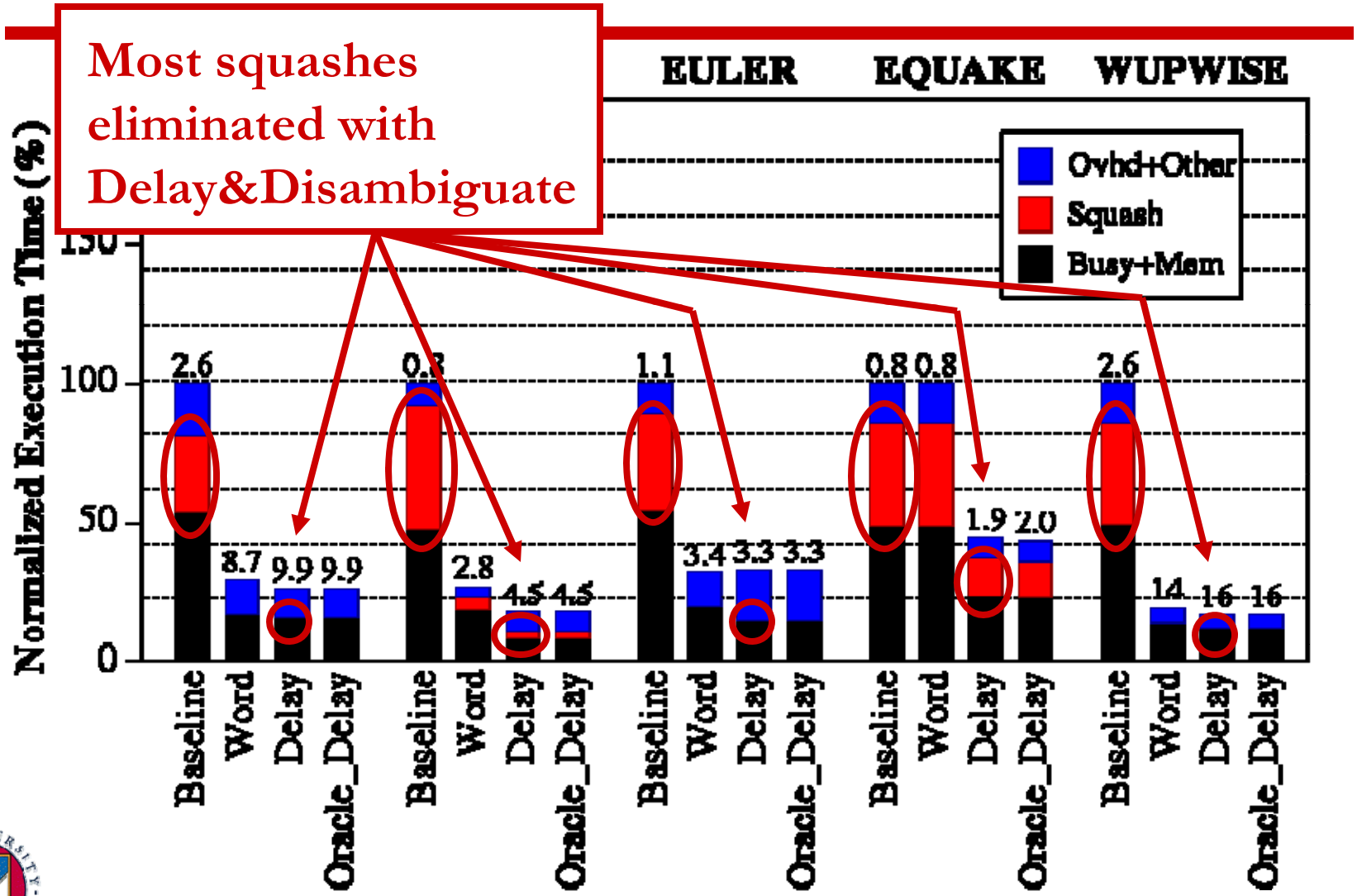
# Delay&Disambiguate Performance



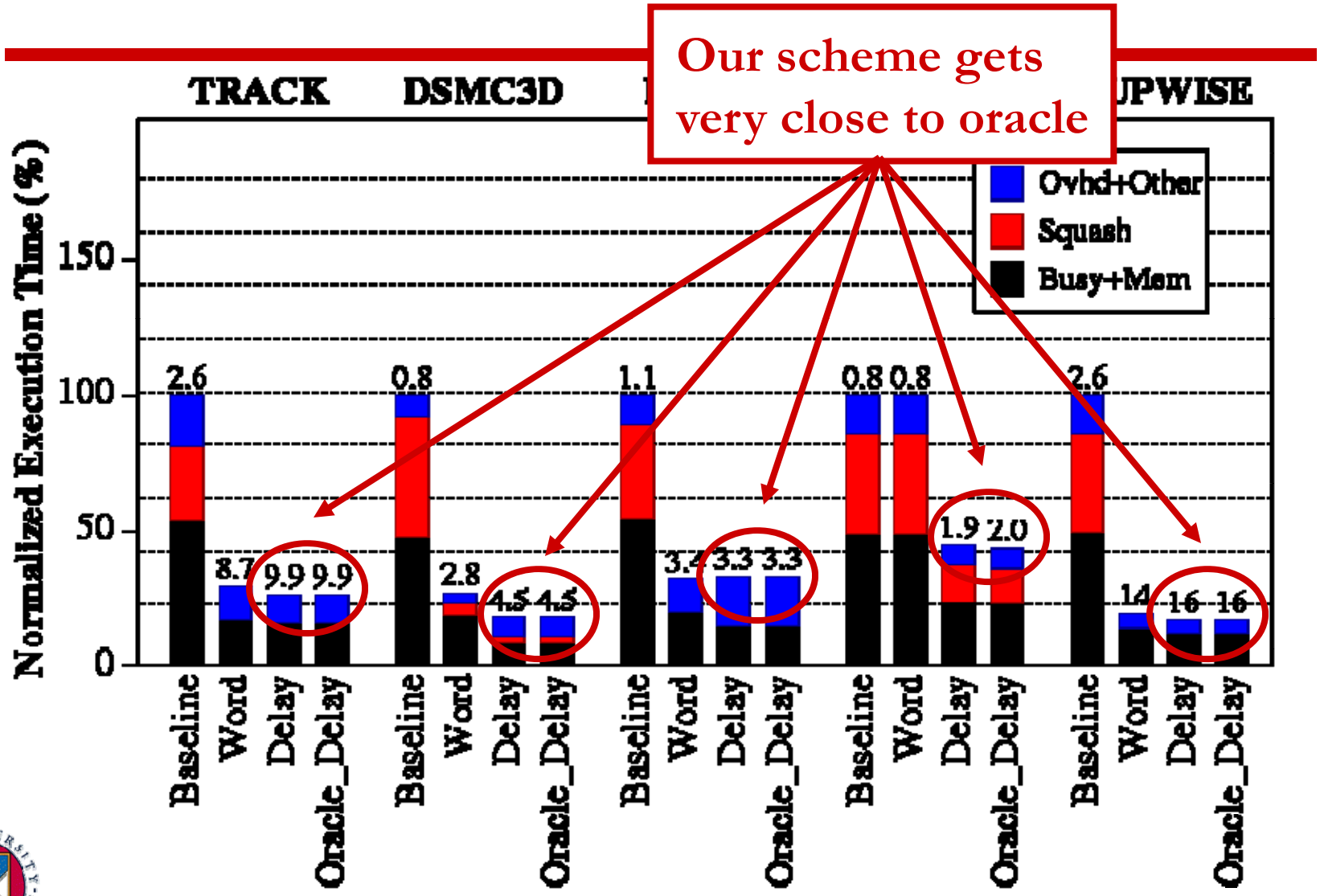

# Delay&Disambiguate Performance



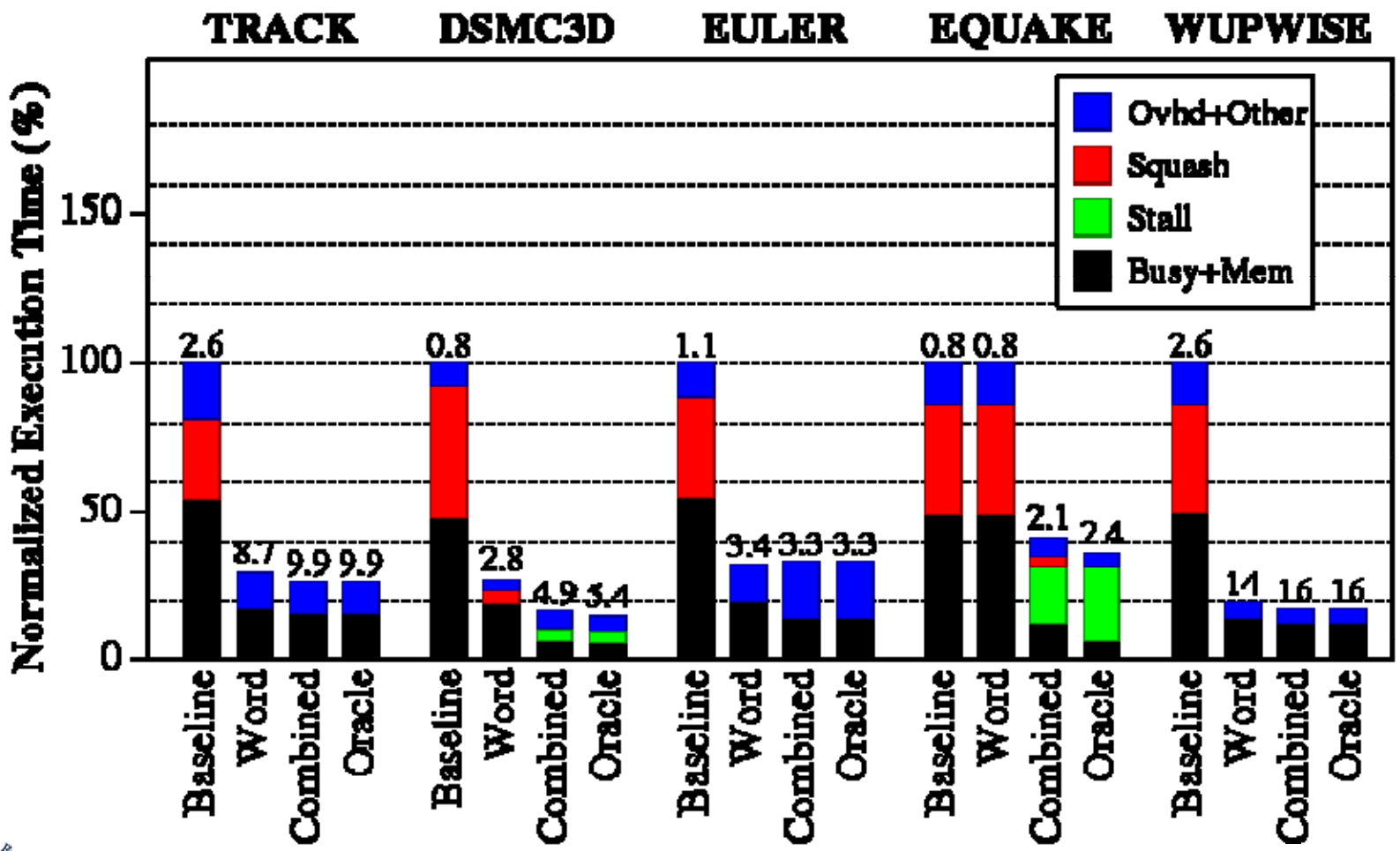

# Delay&Disambiguate Performance



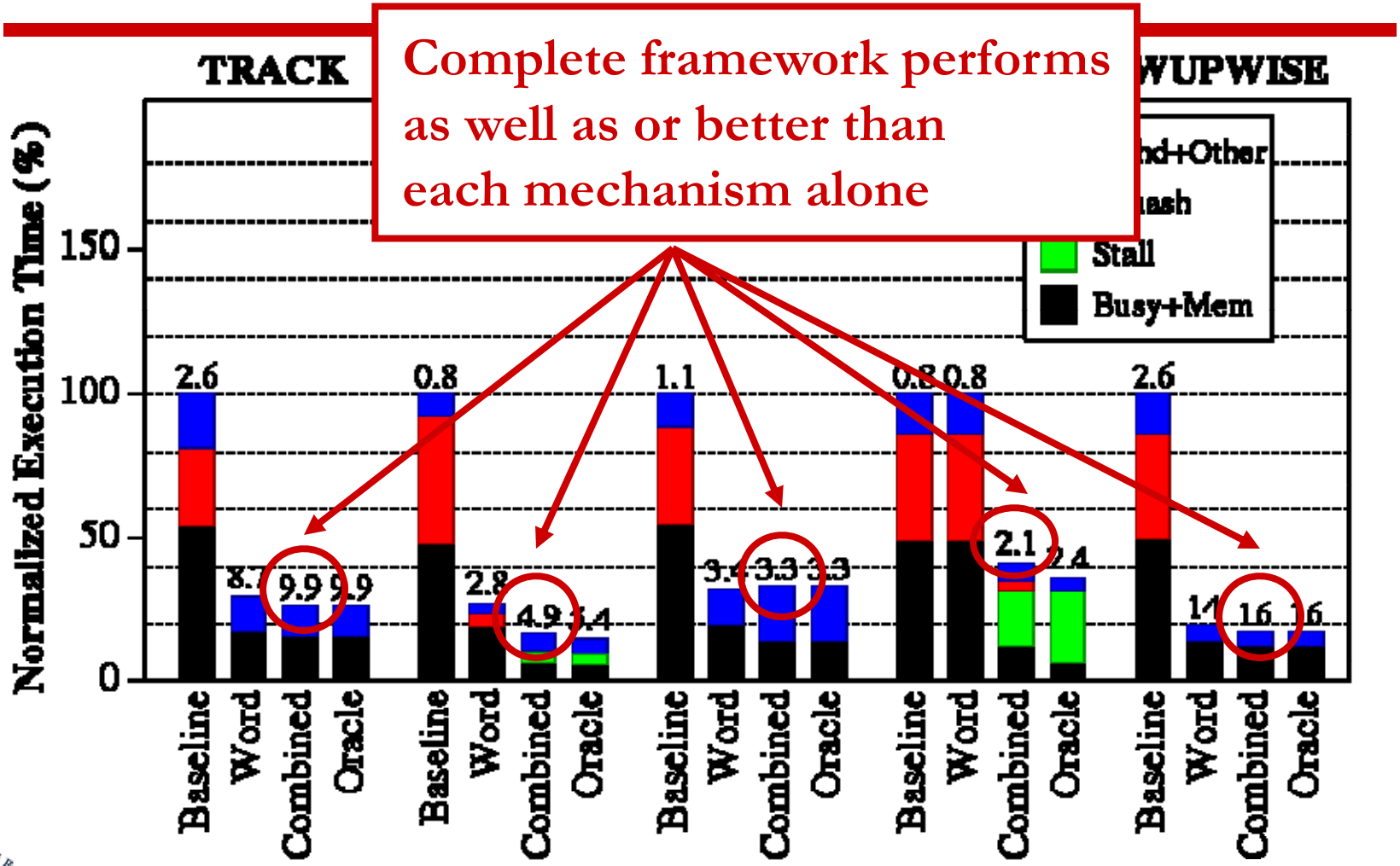

# Delay&Disambiguate Performance



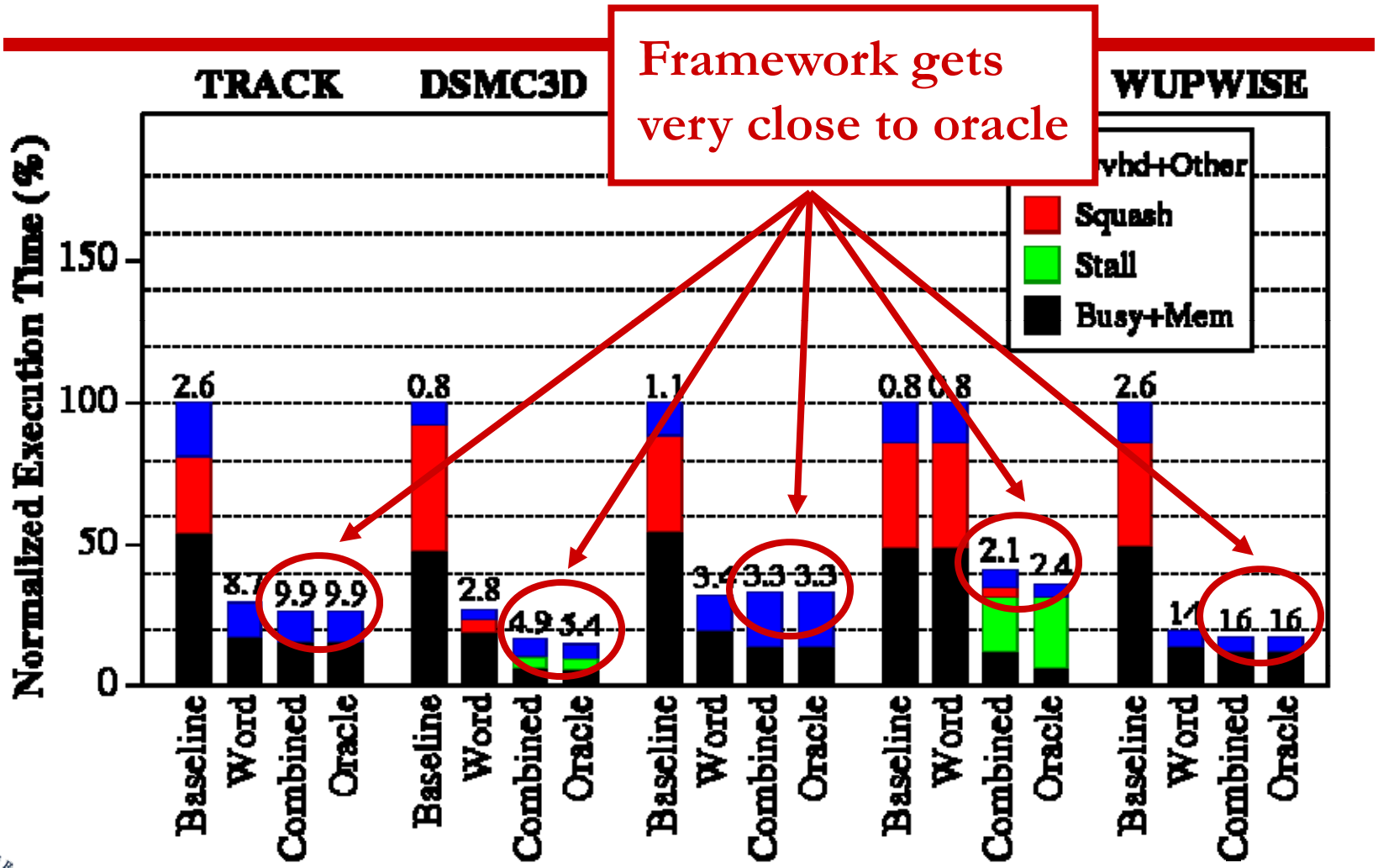

# Complete Framework Performance



# Complete Framework Performance



# Complete Framework Performance




# Outline

- Motivation and Background
- Overview of Framework
- Mechanisms
- Implementation
- Evaluation
- Related Work
- Conclusions

# Related Work

Other dynamic learning, prediction, and specialized handling of dependences for speculative parallelization:

- Synchronization: Multiscalar; TLDS
  - Mostly tailored to CMP
  - Learning based on instruction addresses (Multiscalar)
- Value Prediction: Clustered Speculative; TLDS
  - We use compiler to eliminate trivial dependences (like TLDS)
  - We use floating-point applications
- Mixed word/line speculation: I-ACOMA
  - We never require word addresses on memory operations

# Outline

- Motivation and Background
- Overview of Framework
- Mechanisms
- Implementation
- Evaluation
- Related Work
- Conclusions

# Conclusions

- Framework of hardware mechanisms eliminates most squashes
- Very good performance of line-based speculative machine with framework:
  - 4.3 times faster than a line-based speculative machine without framework
  - 1.2 times faster than expensive word-based speculative machine
- Delay&Disambiguate has largest impact but combination with Stall&Wait is best
- ValuePredict did not work well in our environment

# Eliminating Squashes Through Learning Cross-Thread Violations in Speculative Parallelization for Multiprocessors

Marcelo Cintra and Josep Torrellas

University of Edinburgh
http://www.dcs.ed.ac.uk/home/mc

University of Illinois
at Urbana-Champaign
http://iacoma.cs.uiuc.edu

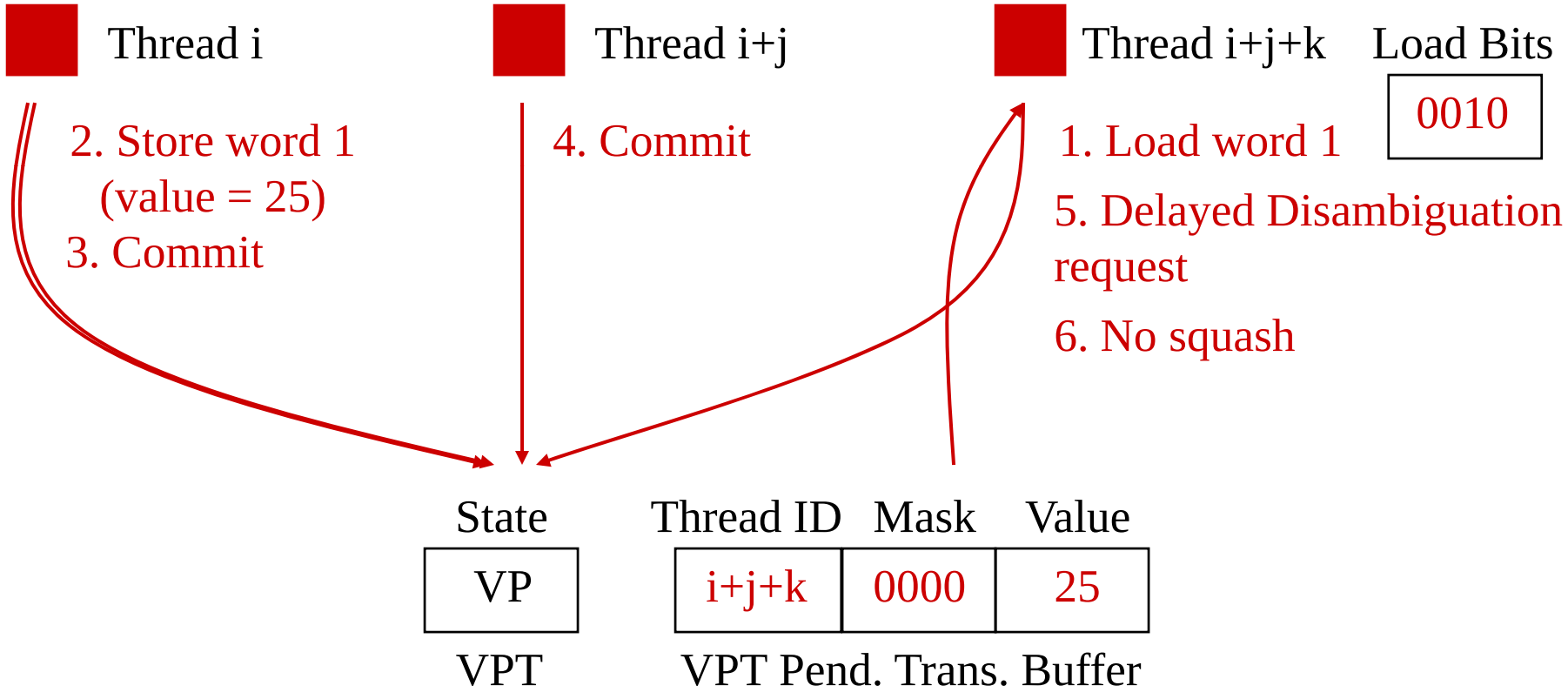

# VPT Fields

- State Bits: the mechanism currently in use for the line
- Squash Counter: number of squashes to the line incurred with current mechanism
- SavedSquashBit: squash saved by the current mechanism
- ThrsStallR: number of squashes to trigger Stall&Release mechanism
- ThrsStallW: number of squashes to trigger Stall&Wait mechanism
- AgePeriod: number of commits to age the state of line

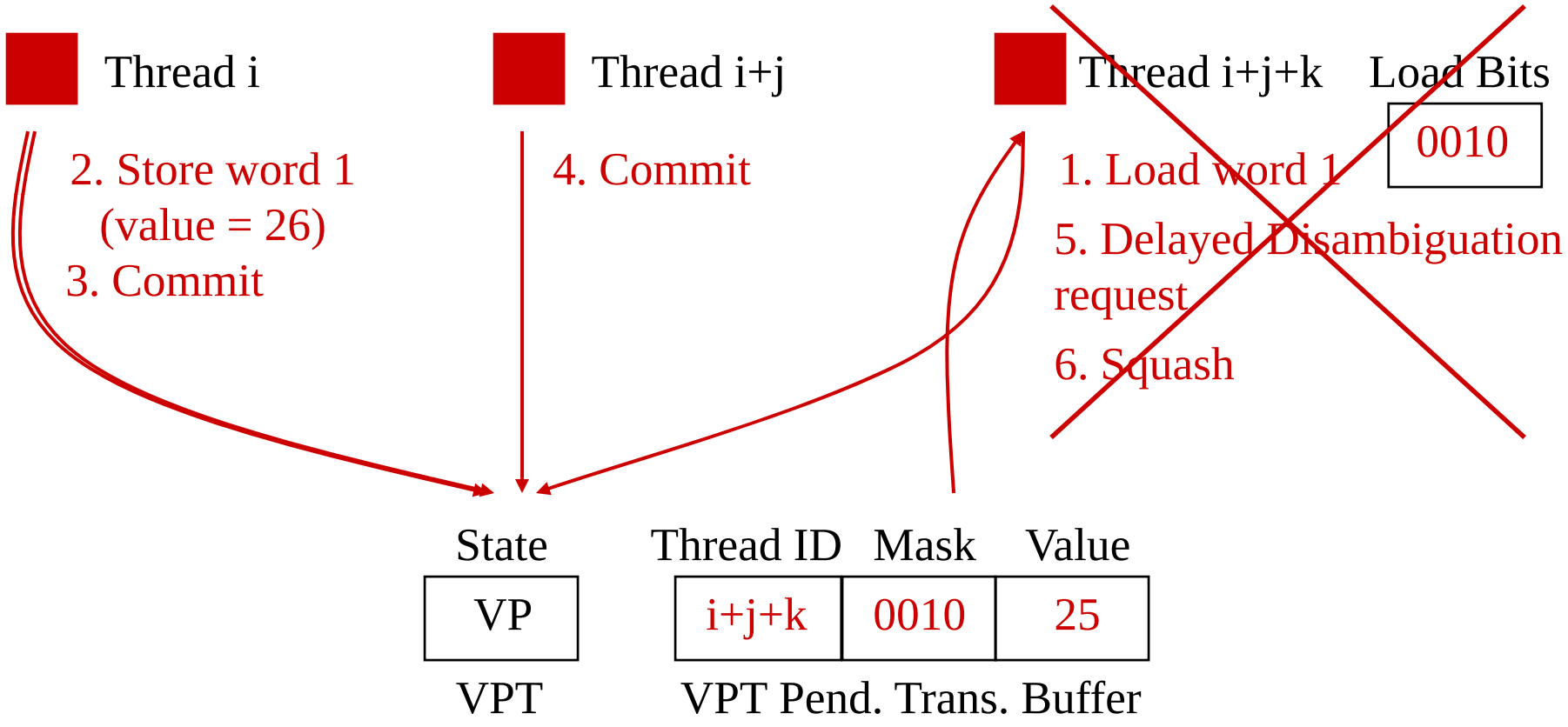
# Pending Transactions Buffer Fields

- Line Tag: address of line with unresolved violation
- Thread ID: consumer thread with a unresolved violation
- State Bits: mechanism used
- Modified Mask: bitmap of all modifications to words of the line by predecessor threads
- Predicted Line Value: actual data values provided to consumer thread

# Operation under ValuePredict




# Operation under ValuePredict




# Summary

+ Delay&Disambiguate offers word-level disambiguation without word addresses in most memory transactions
+ Simple implementation combines all different mechanisms seamlessly
+ Learning/Prediction policy effective when subset of speculative data tends to be accessed with dependences
− Learning/Prediction policy not so effective when subset of speculative instructions tends to cause dependences
− Learning/Prediction policy reacts to previous behavior but cannot extrapolate/anticipate behavior

# Simulation Environment

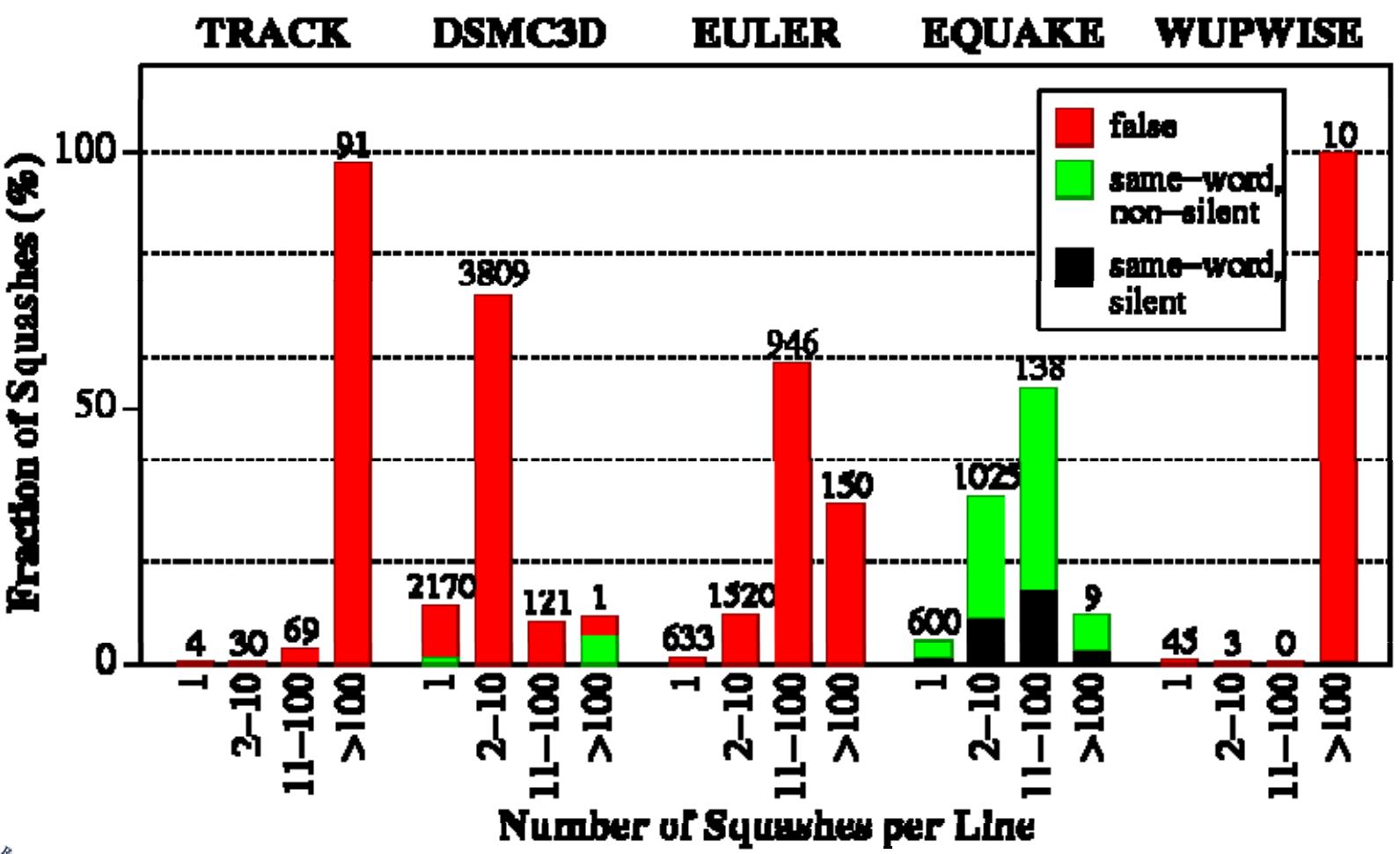
| Processor Param. | Value |
|---|---|
| Issue width | 4 |
| Instruction window size | 64 |
| No. functional units(Int,FP,Ld/St) | 3,2,2 |
| No. renaming registers(Int,FP) | 32,32 |
| No. pending memory ops.(Ld,St) | 8,16 |

| 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 |
| GMDT size | 2K entries |
| GMDT assoc. | 8-way |
| GMDT/VPT lookup | 20 cycles |
| Pend. Trans. Buffer size | 128 entries |
| Pend. Trans. Buffer scan | 3 cycles/entry |

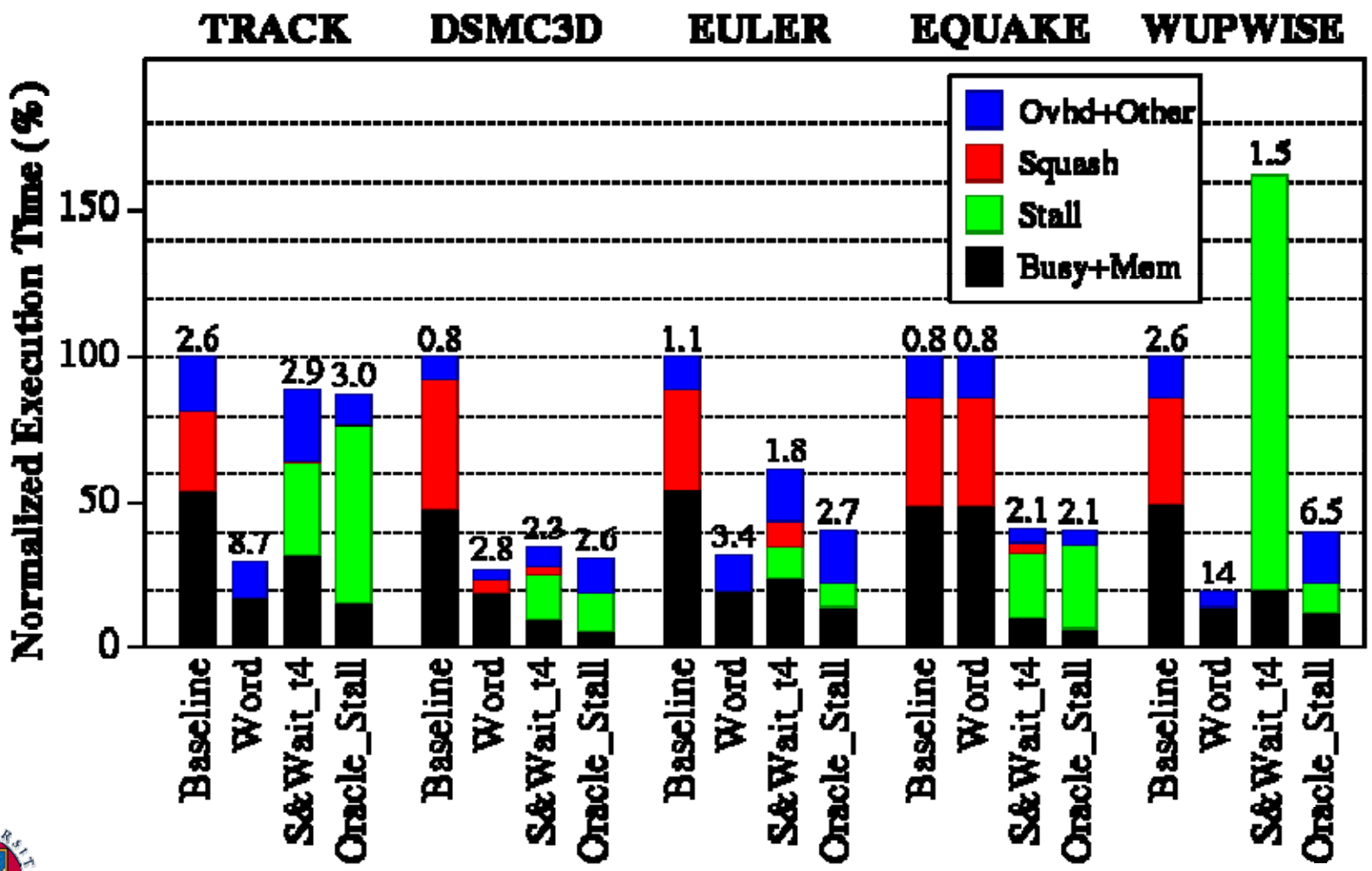
# Application Characteristics

| *Application* | *Loops* | *% of Seq. Time* | *RAW Dependences* |
|---|---|---|---|
| TRACK | nlfilt_300 | 58 | Same-word and False |
| DSMC3D | move3_100 | 41 | Same-word and False |
| EULER | dflux_[100,200]<br>psmoo_20<br>eflux_[100,200,300] | 90 | False |
| EQUAKE | smvp_1195 | 45 | Same-word |
| WUPWISE | muldeo_200'<br>muldoe_200' | 67 | Same-word and False |

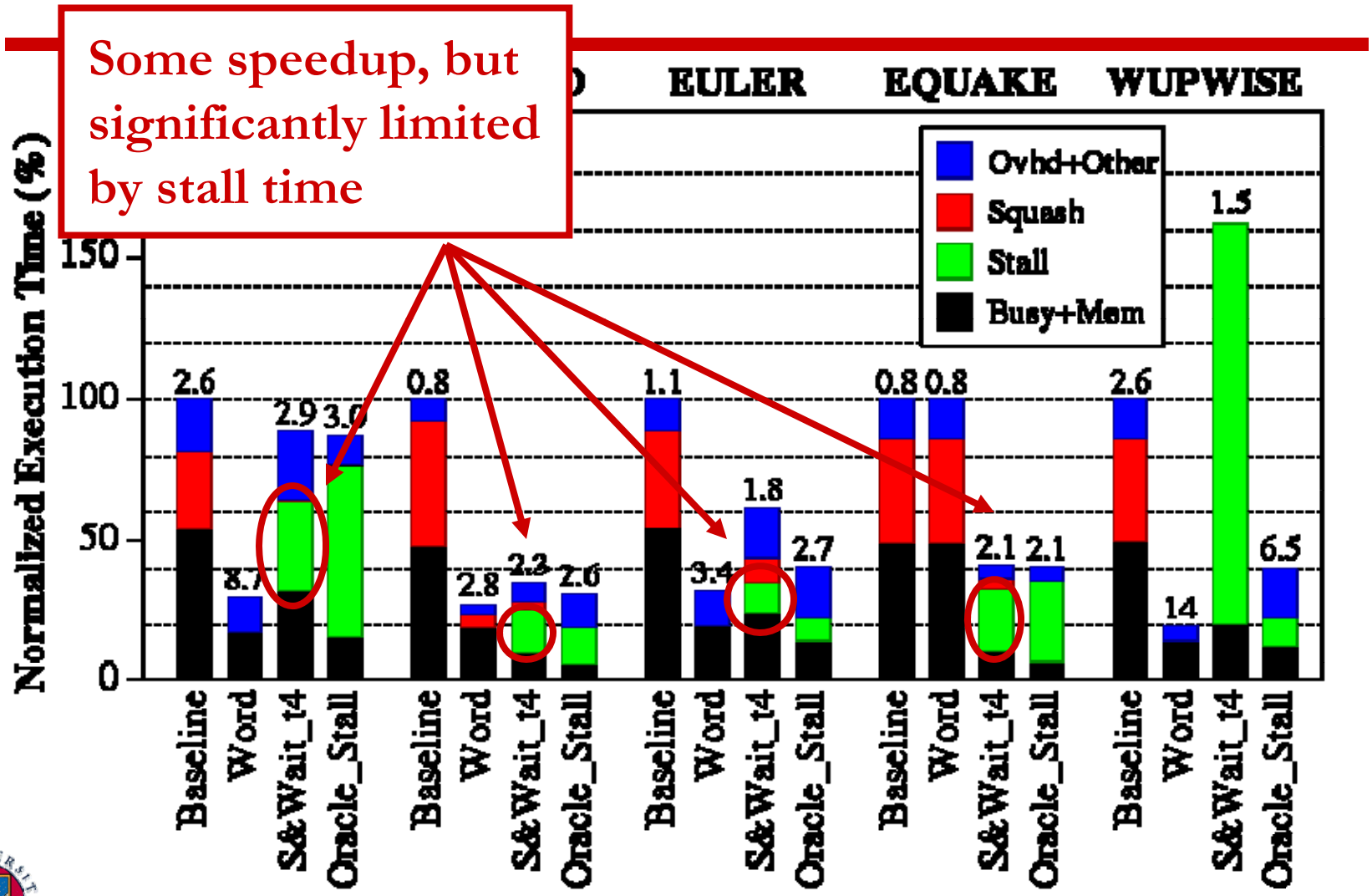
# Squash Behavior



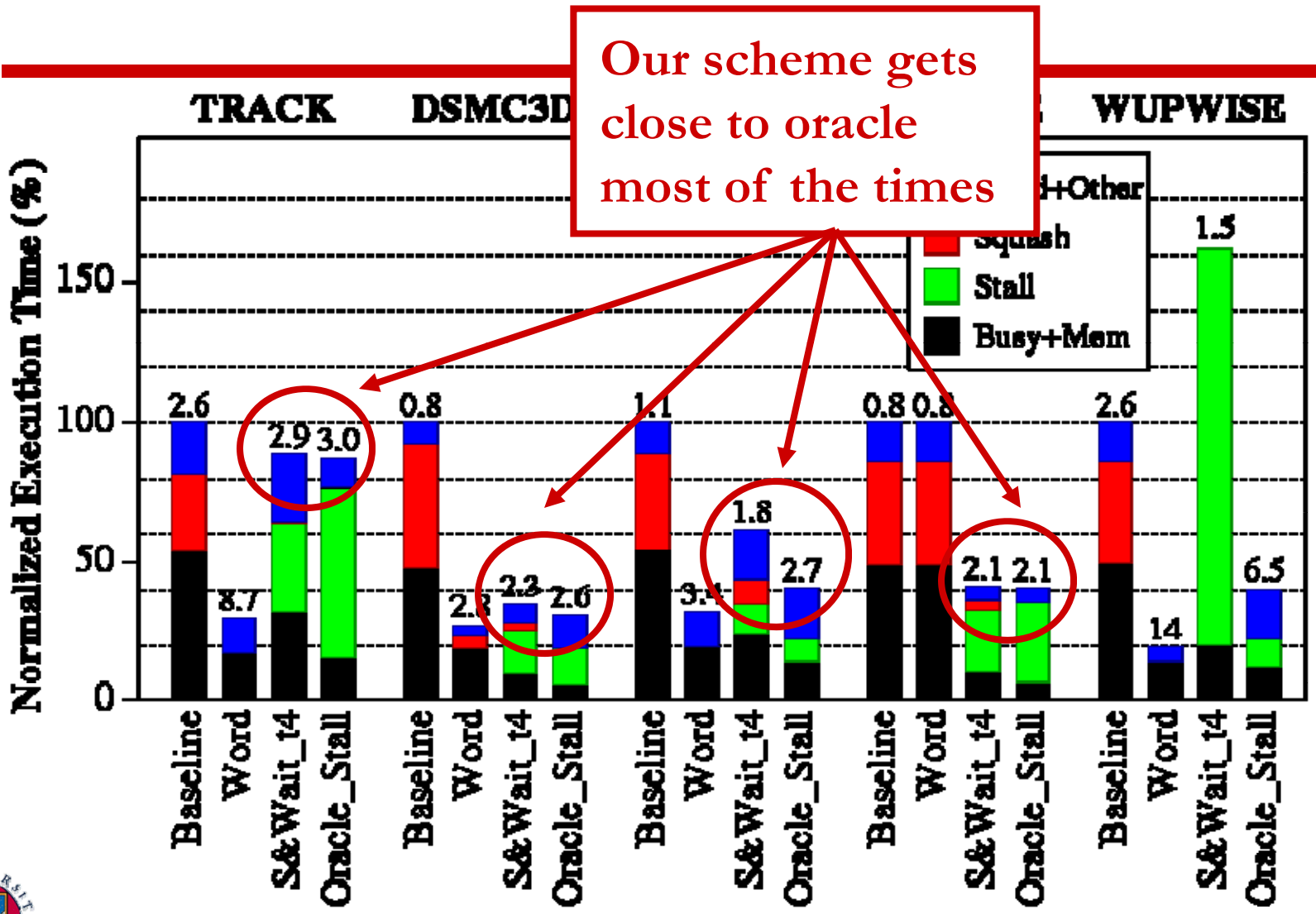

# Stall&Wait Performance




# Stall&Wait Performance



# Stall&Wait Performance



# Related Work

- Other hardware-based speculative parallelization:
  - Multiscalar (Wisconsin); Hydra (Stanford); Clustered Speculative (UPC); TLDS (CMU); MAJC (Sun); Superthreaded (Minnesota); Illinois
  - Mostly tailored for CMP
  - Mostly word-based speculation
  - Mostly squash-and-retry