Dual-Thread Speculation: Two Threads in the Machine are Worth Eight in the Bush

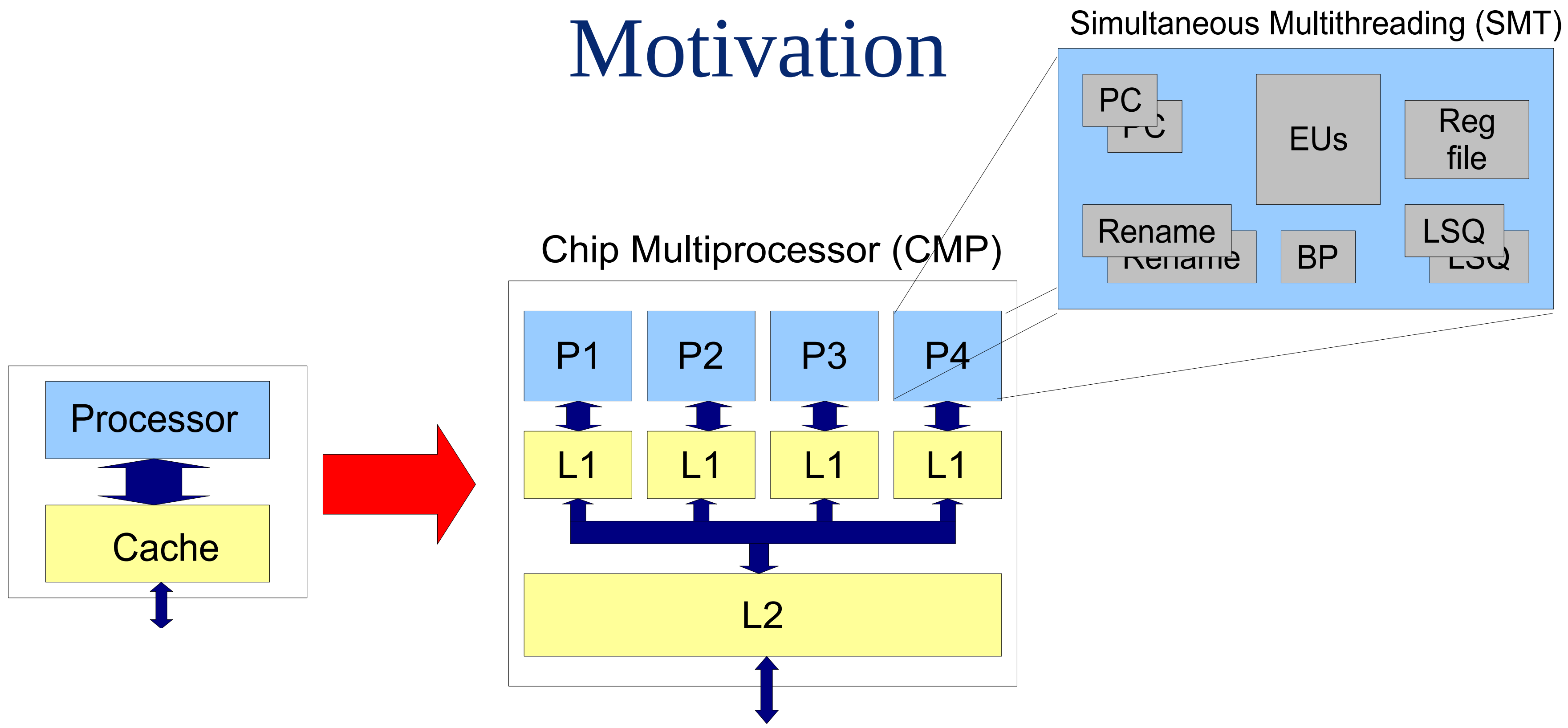
Fredrik Warg and Per Stenstrom

Department of Computer Science and Engineering

Chalmers University of Technology

SBAC-PAD 2006

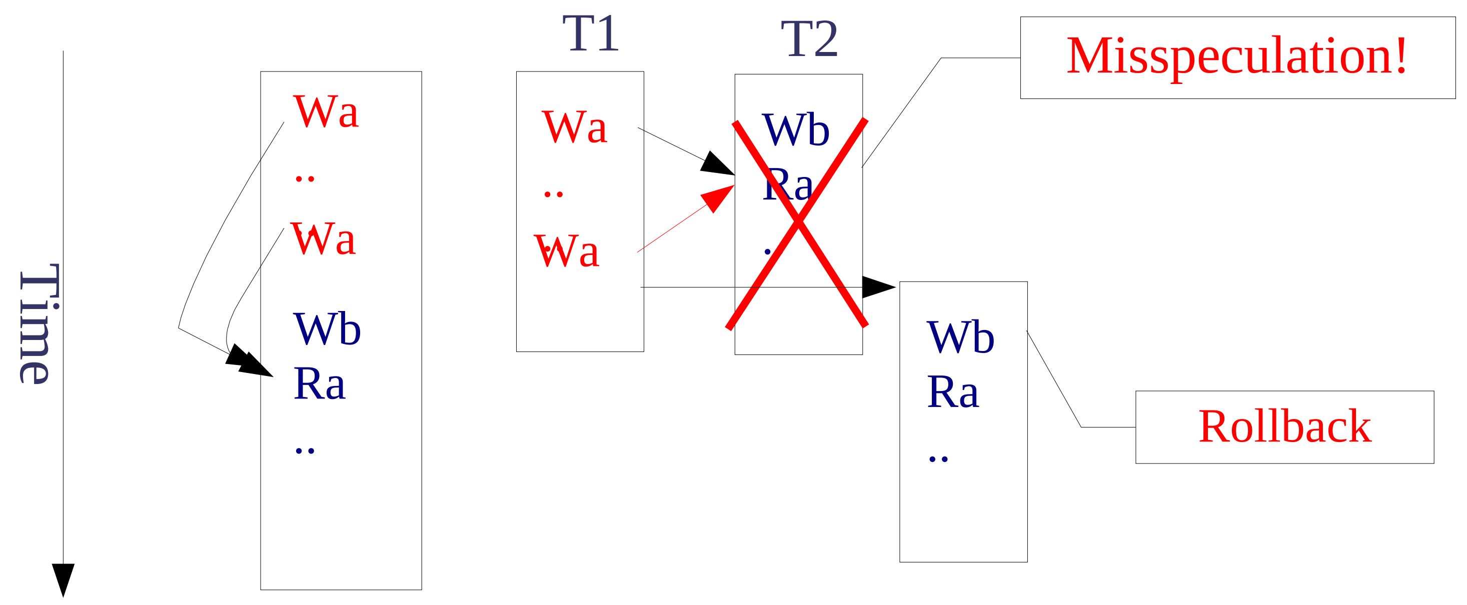
Motivation



- Single-thread processors replaced by **multithreaded architectures**.
- Performance gains expected from **thread-level parallelism** (TLP).
- But what about single-threaded application performance?

Thread-Level Speculation (TLS)

- Agressive parallelization - even when dependences are unknown



- Overheads a challenge for TLS performance
  - Thread start, inter-thread communication, rollback, etc.

Problem Statement

- Chip multiprocessors with SMT cores
  - Threads on same core (SMT-approach)
    - Faster communication and thread-start
  - Threads on different cores (CMP-approach)
    - No cache or processor resource contention
  - **When is each approach preferable for TLS?**

- TLS implementation complexity
  - Relatively costly to support TLS in hardware for common approaches, e.g. # TLS threads scale with #cores
  - **How well can a simpler dual-thread TLS implementation perform?**

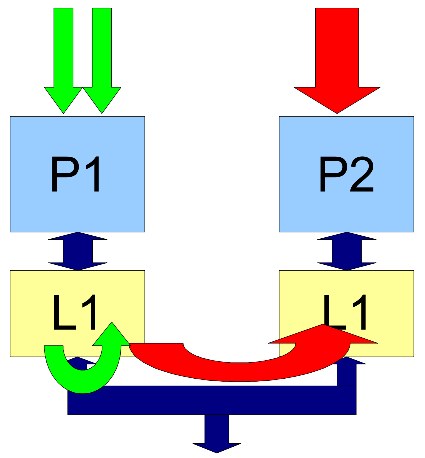
Contributions

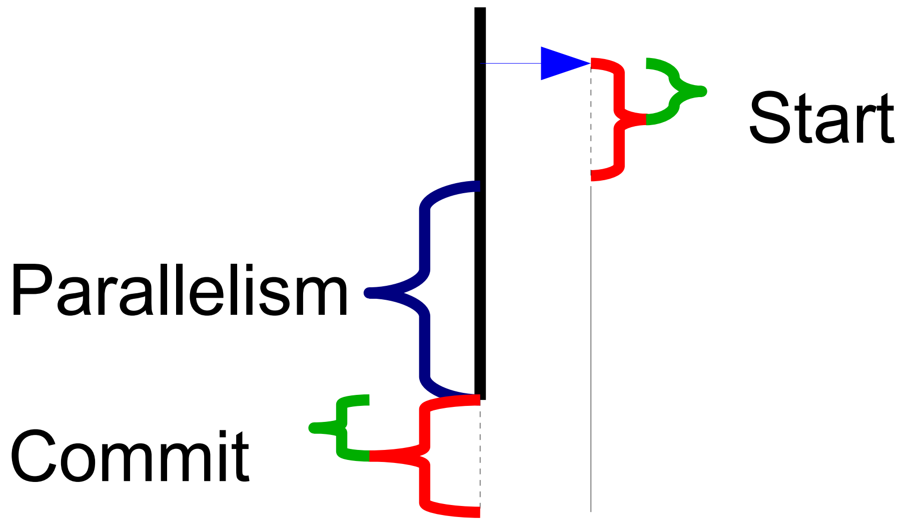
- Comparing CMP- and SMT-approach for TLS
  - Lower thread-management and communication costs most often gives SMT-approach threads an advantage.
- Comparing dual-thread and scalable TLS
  - Our dual-thread machine achieves most performance benefits of TLS compared to an 8-thread scalable TLS machine.
  - TLS implementation can be greatly simplified.

Outline

- CMP- vs SMT-approach threads
  - Main differences from a TLS perspective
  - Simulation methodolody
  - Simulation results
- Dual-thread vs. Scalable TLS
  - A scalable TLS implementation
  - The simplified dual-thread approach
  - Simulation results
- Conclusions

CMP- versus SMT-approach





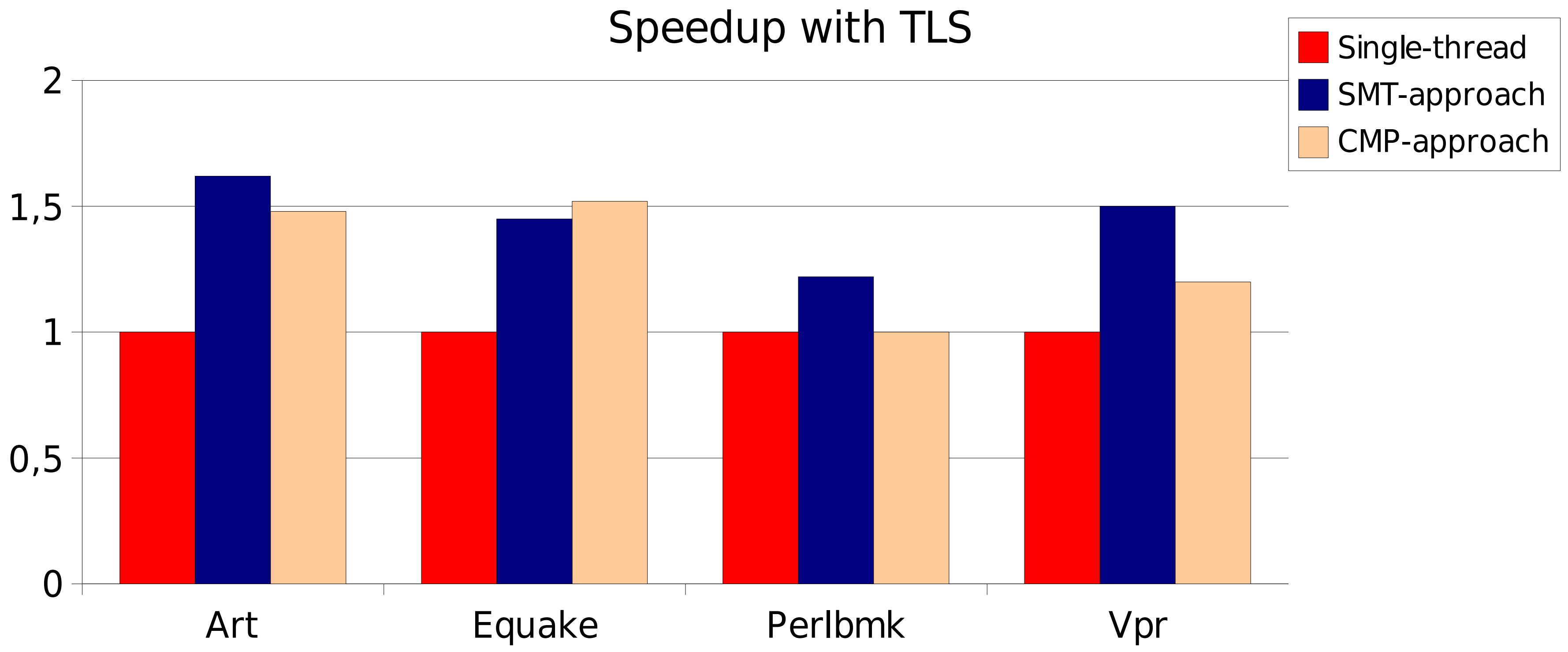
	SMT-approach	*CMP-approach*
Thread start, commit	No comm needed	Comm through interconnect
Thread communication	Comm through L1 cache	Comm through interconnect
L1 cache space	Shared between threads	Scale with number of cores
Execution units	Shared between threads	Scale with number of cores

Methodology

- Benchmarks
  - Applications where parallelizing compilers have not been successful, primarily from SPEC benchmark suites
- Simulation setup
  - Simulates hardware TLS implementation
  - Threads are spawned automatically from unmodified binaries
  - Typical 2/4/8-issue SMT-enabled OoO processor cores
  - Memory hierarcy with 2-level caches (2nd level shared)
- **Compare TLS execution time to running the application sequentially on one core**

CMP- vs. SMT-approach Results

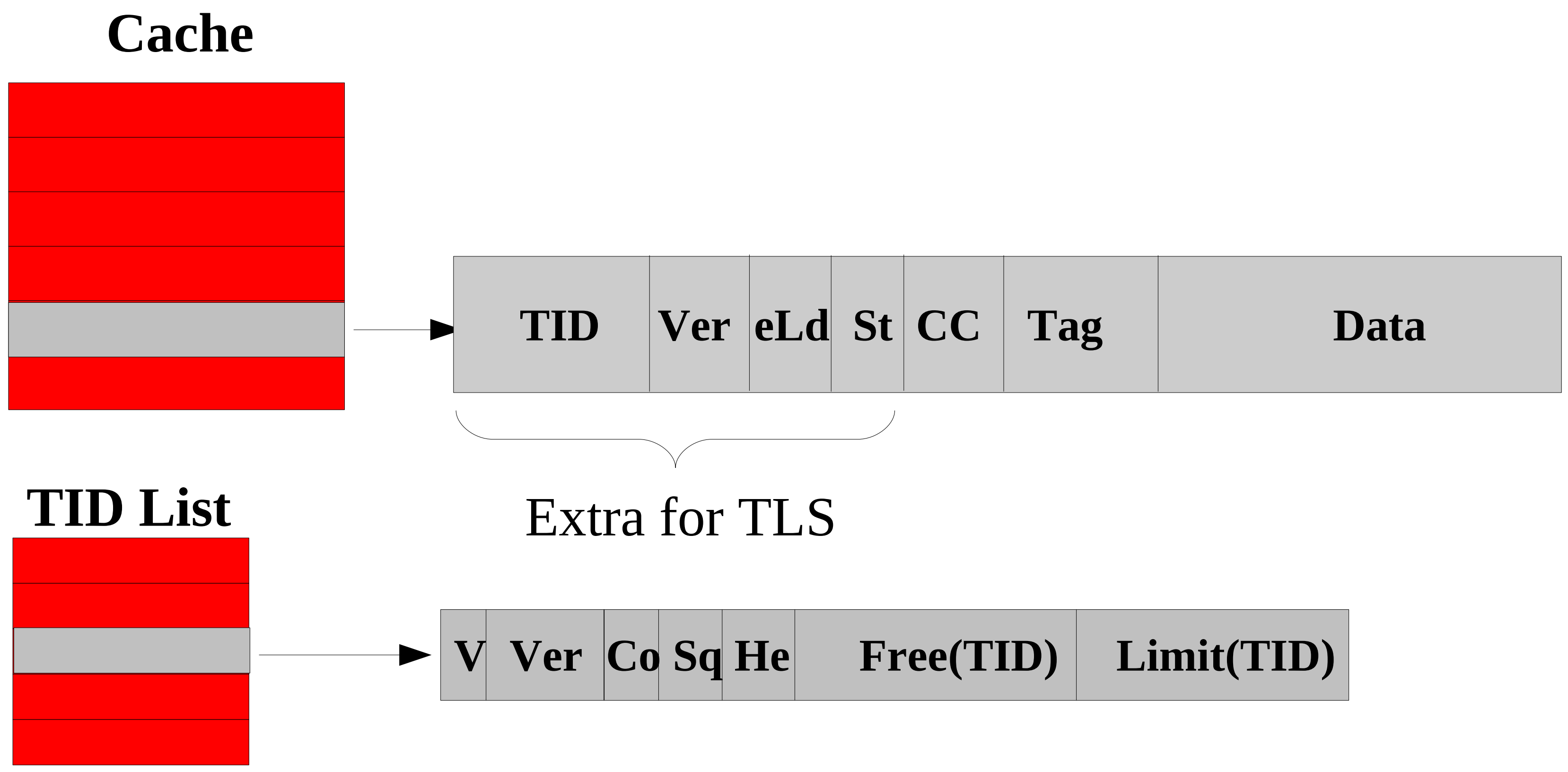
4-thread 8-issue SMT vs 4-way 2-issue CMP



SMT-approach typically performs better

- **Even with less cache and lower total issue width than the CMP**

Scalable TLS Implementation



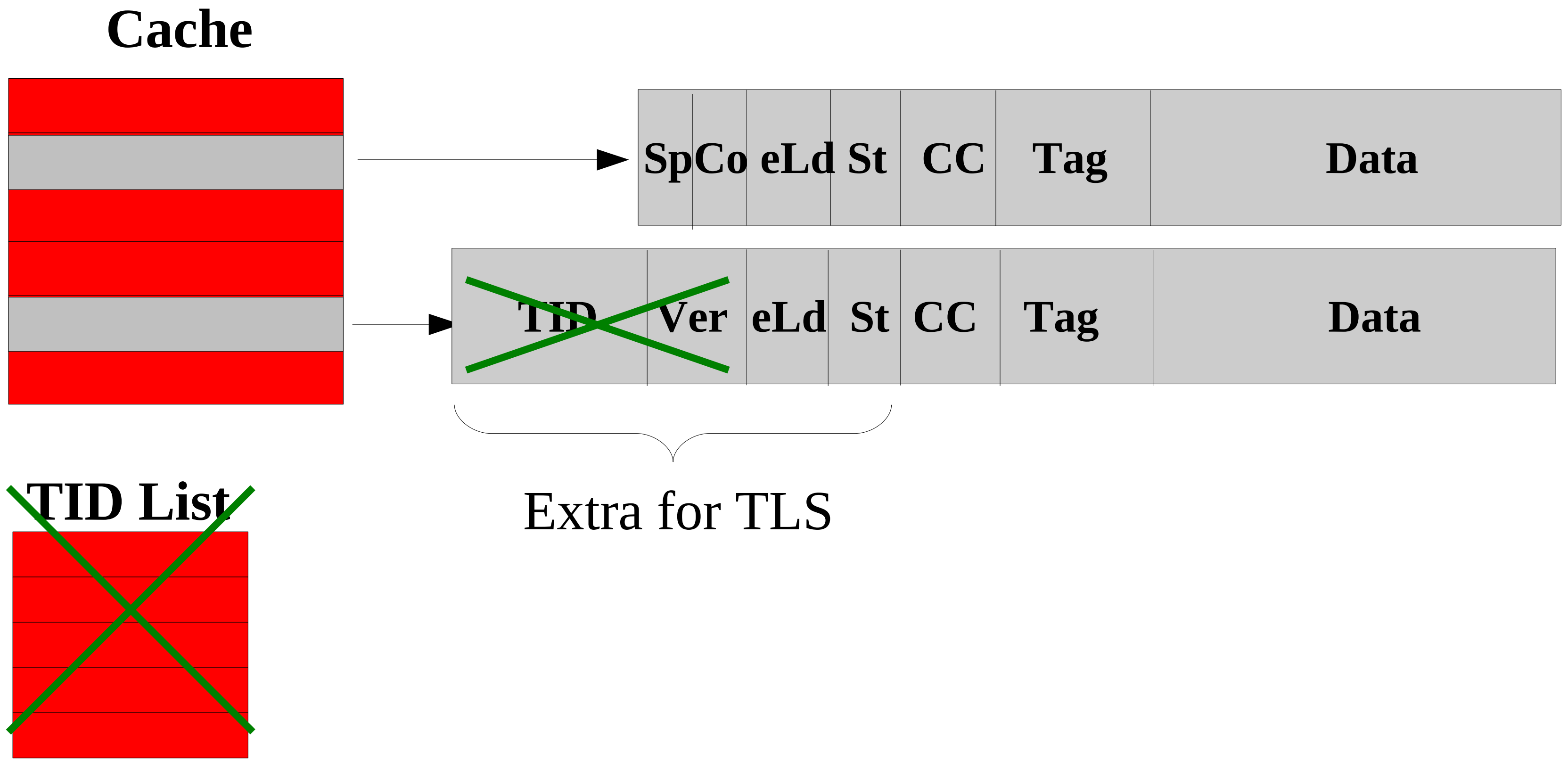
Relatively much extra storage used

Drawbacks With Scalable TLS

- Relatively complex logic
  - Finding correct thread order for TLS operations
  - Ability to selectively squash/commit versions of addresses
- Storage overhead
  - Extra state for each cache line
  - New structure to keep track of threads (TID list)
  - Potentially many versions (ie many copies of a cache line)

What is needed to support a single speculative thread?

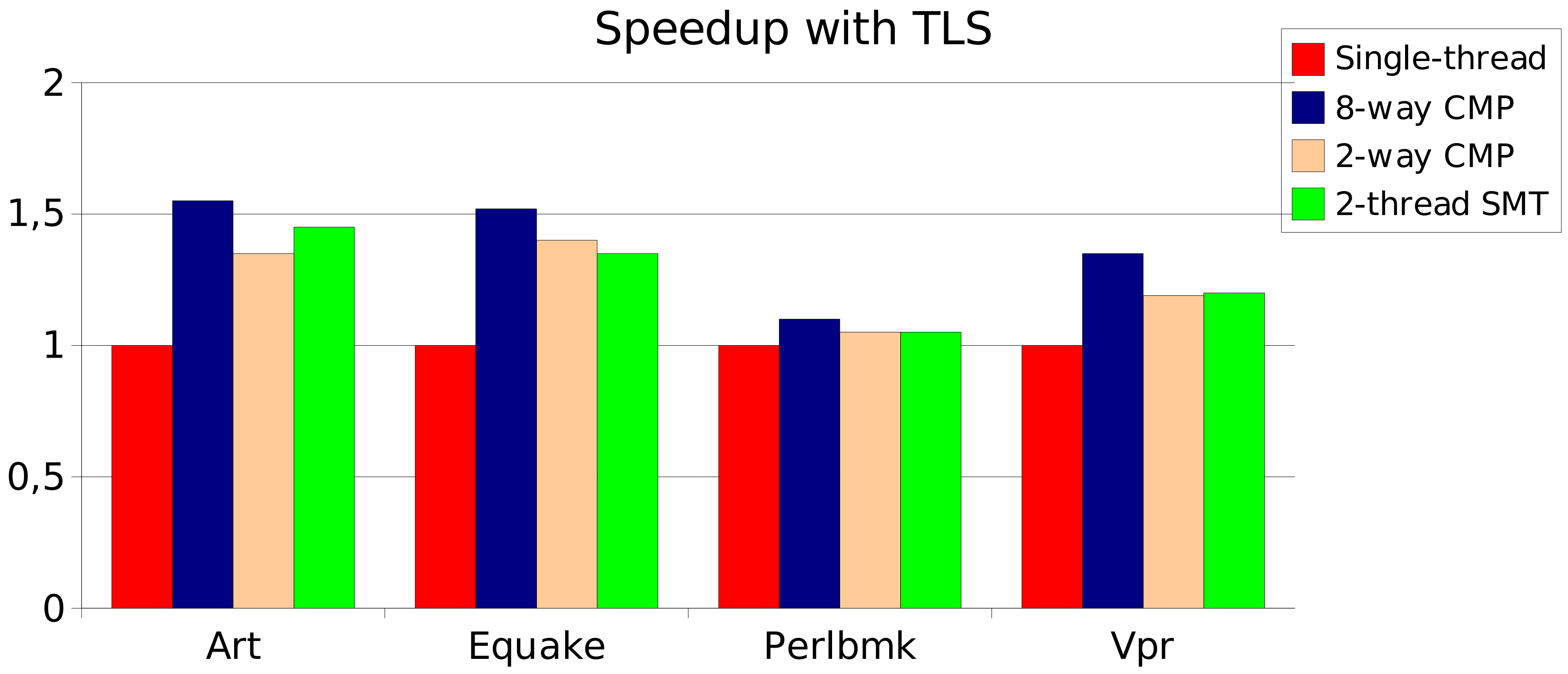
Dual-Thread TLS Implementation



Much lower space overhead (and less logic)!

Dual-Thread Speculation Results

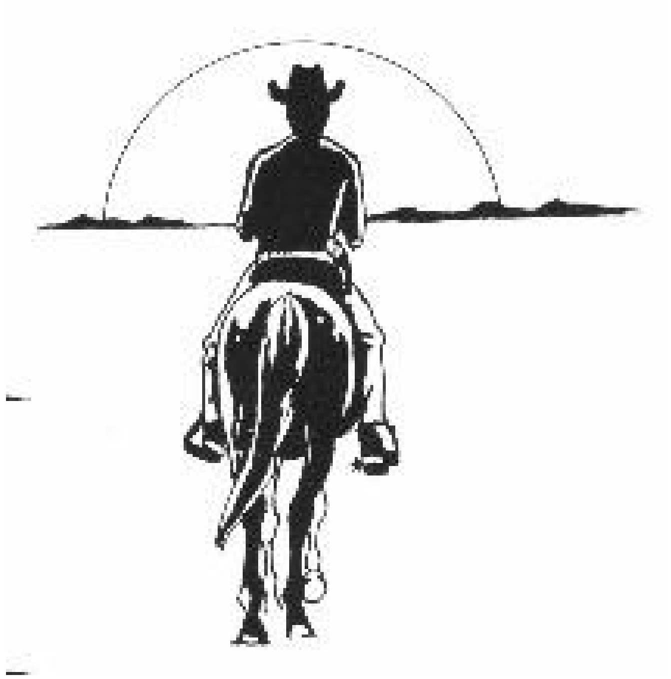
8-way CMP vs dual-thread CMP/SMT



Dual-thread performance good considering its simplicity

Conclusions

- CMP- vs. SMT-approach
  - SMT cores an interesting target for hardware TLS due to low-latency communication
- Dual-Thread Speculation
  - A viable alternative to scalable TLS in some situations. Better performance vs. complexity trade-off than scalable TLS.



The End

Time for questions!