Dynamic Thread Mapping Based on Machine Learning for Transactional Memory Applications

Márcio Castro[1], Luís Fabrício Wanderley Góes[2], Luiz Gustavo Fernandes[3], Jean-François Méhaut[1]

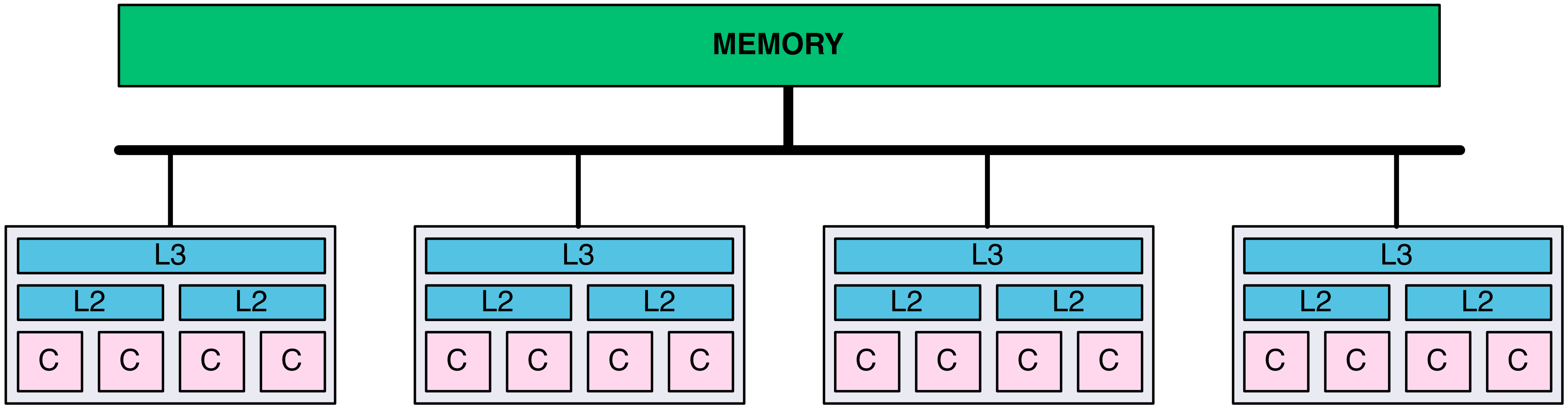
[1]INRIA - LIG Laboratory - MESCAL Group - Grenoble University, France
[2]Department of Computer Science - Pontifical Catholic University of Minas Gerais, Brazil
[3]PPGCC - Pontifical Catholic University of Rio Grande do Sul, Brazil

Outline

- Motivation
- Machine Learning to Thread Mapping on STM
- Static Thread Mapping
- Ongoing Work: Dynamic Thread Mapping
- Conclusions

Motivation

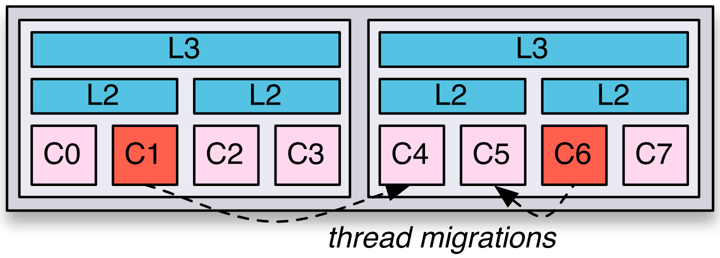
- Multicore processors
  - Mainstream approach to deliver higher performance
  - Complex memory hierarchies: different levels of cache
  - Limited and shared bandwidth



Motivation

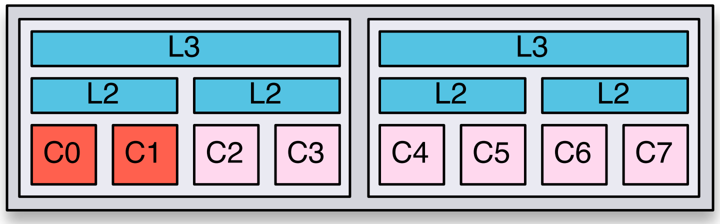
- Thread mapping:
  - Appealing approach to efficiently exploit the potential of multicores
  - Reduce latency or alleviate memory contention

Linux



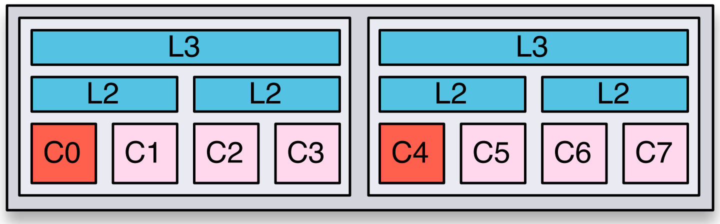
Linux scheduler decides where each thread will run (threads may migrate)

Compact



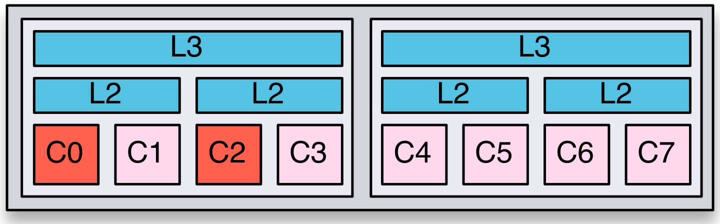
Threads share caches when possible

Scatter



Avoid cache sharing between threads

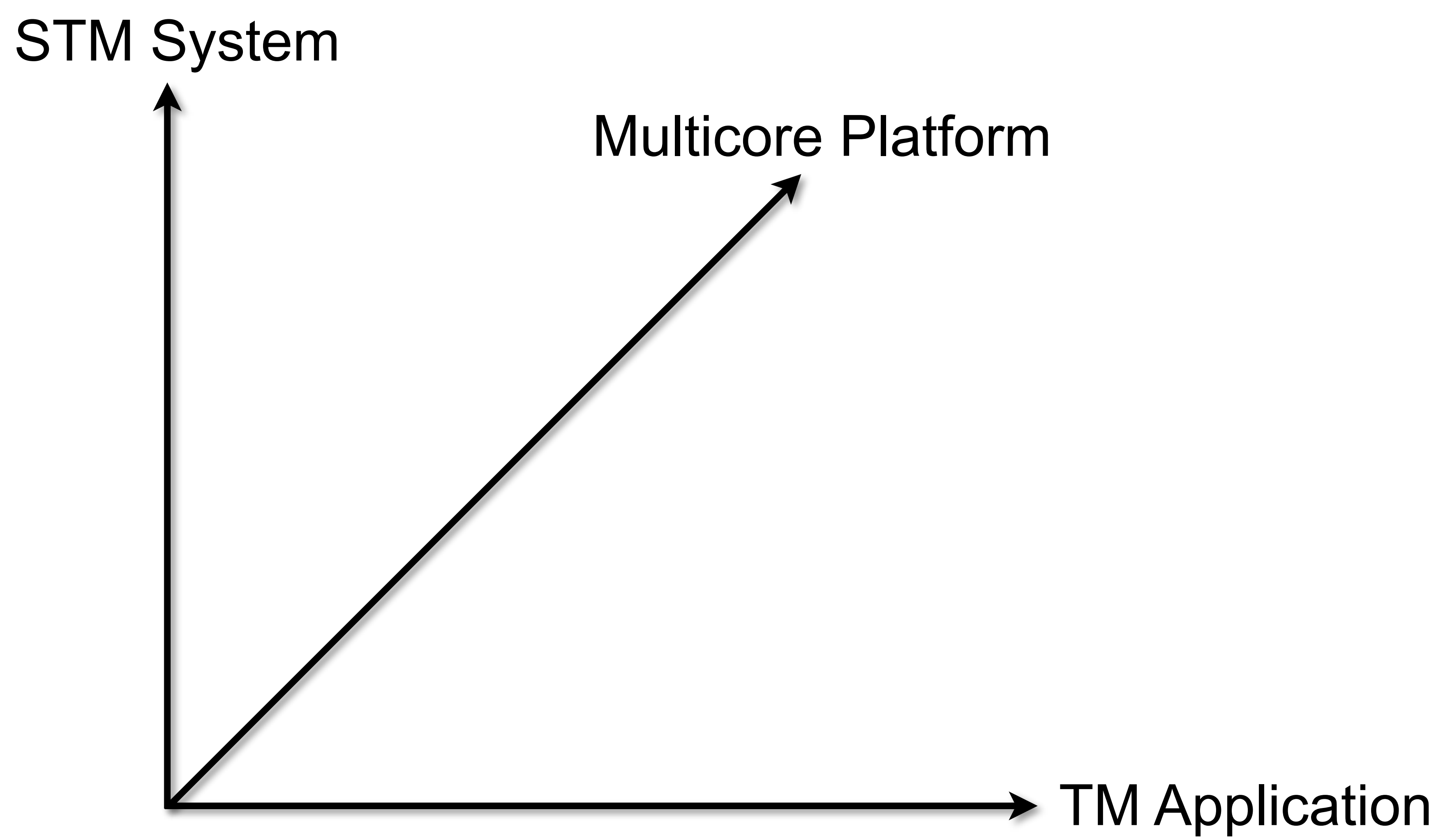
Round-Robin



Threads share higher levels of cache

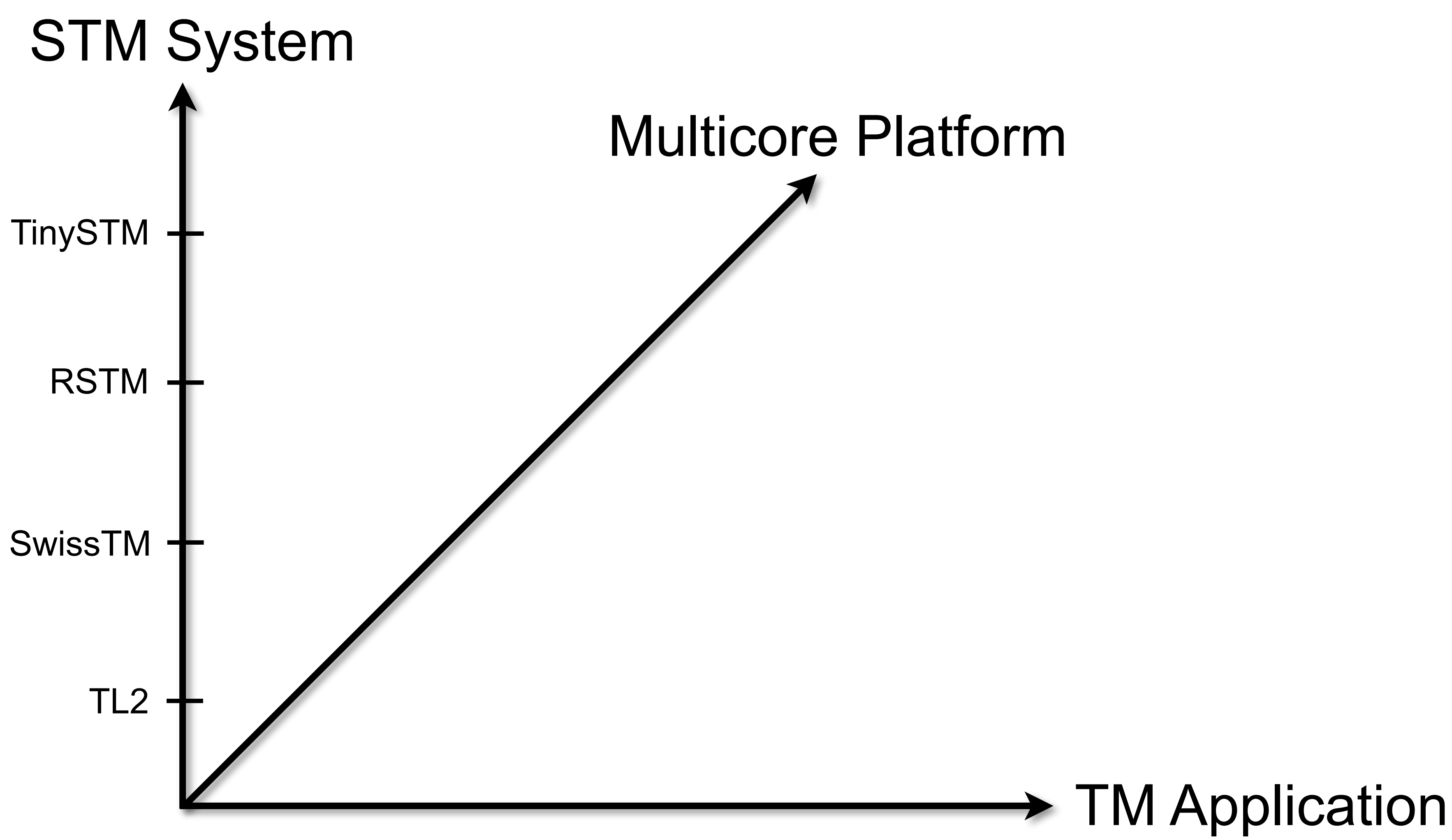
Motivation

- Problem statement:



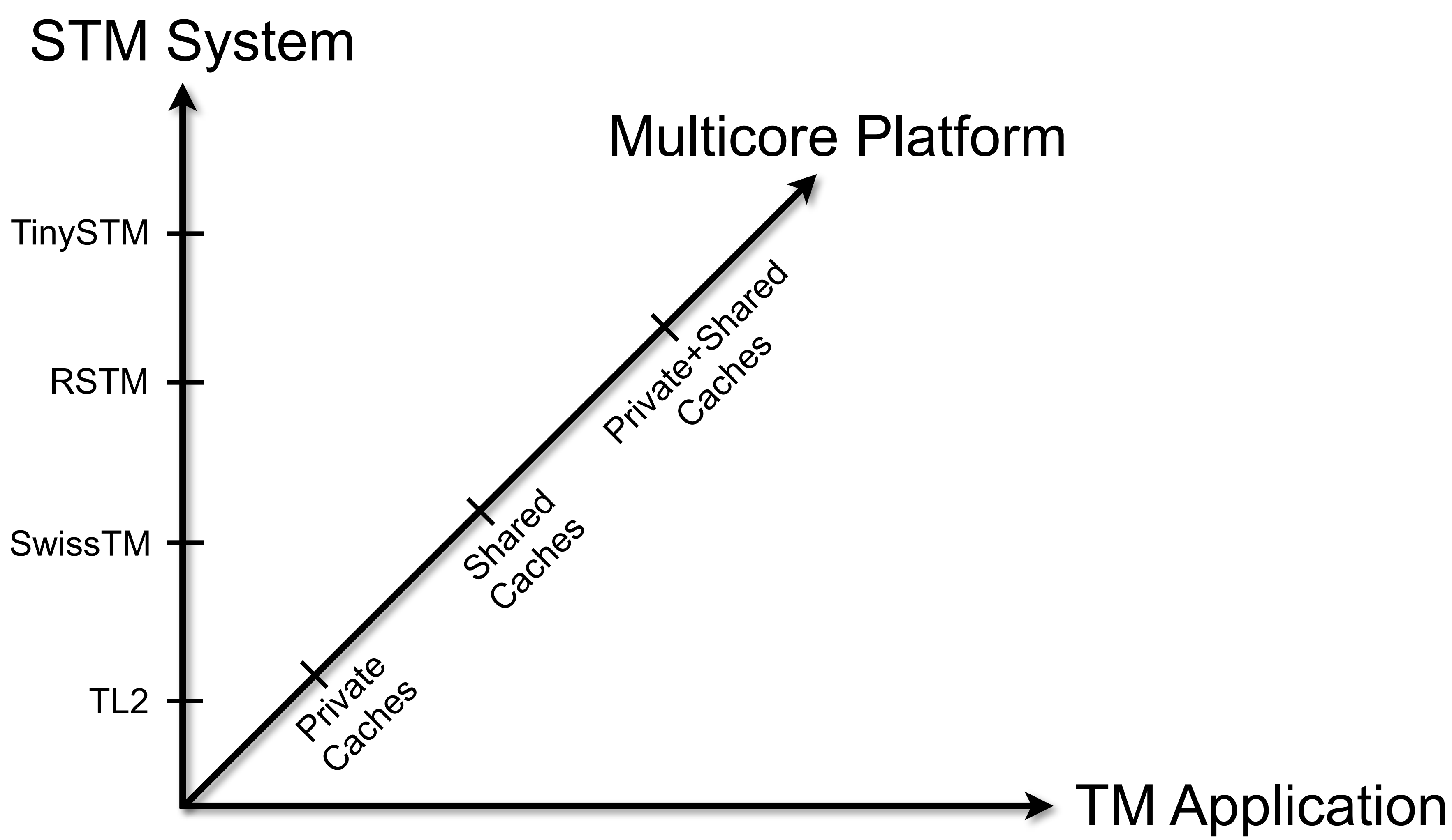
Motivation

- Problem statement:



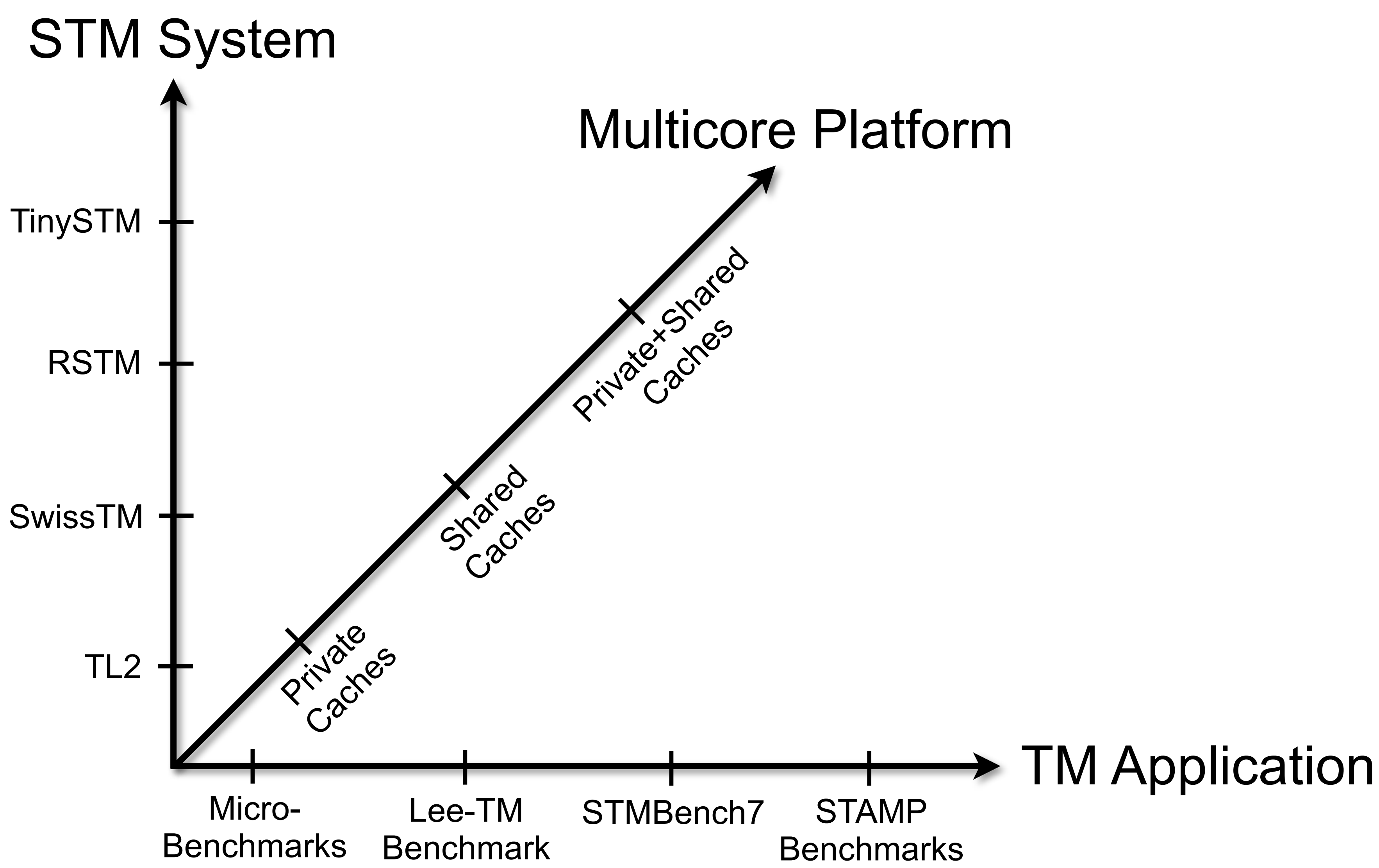
Motivation

- Problem statement:



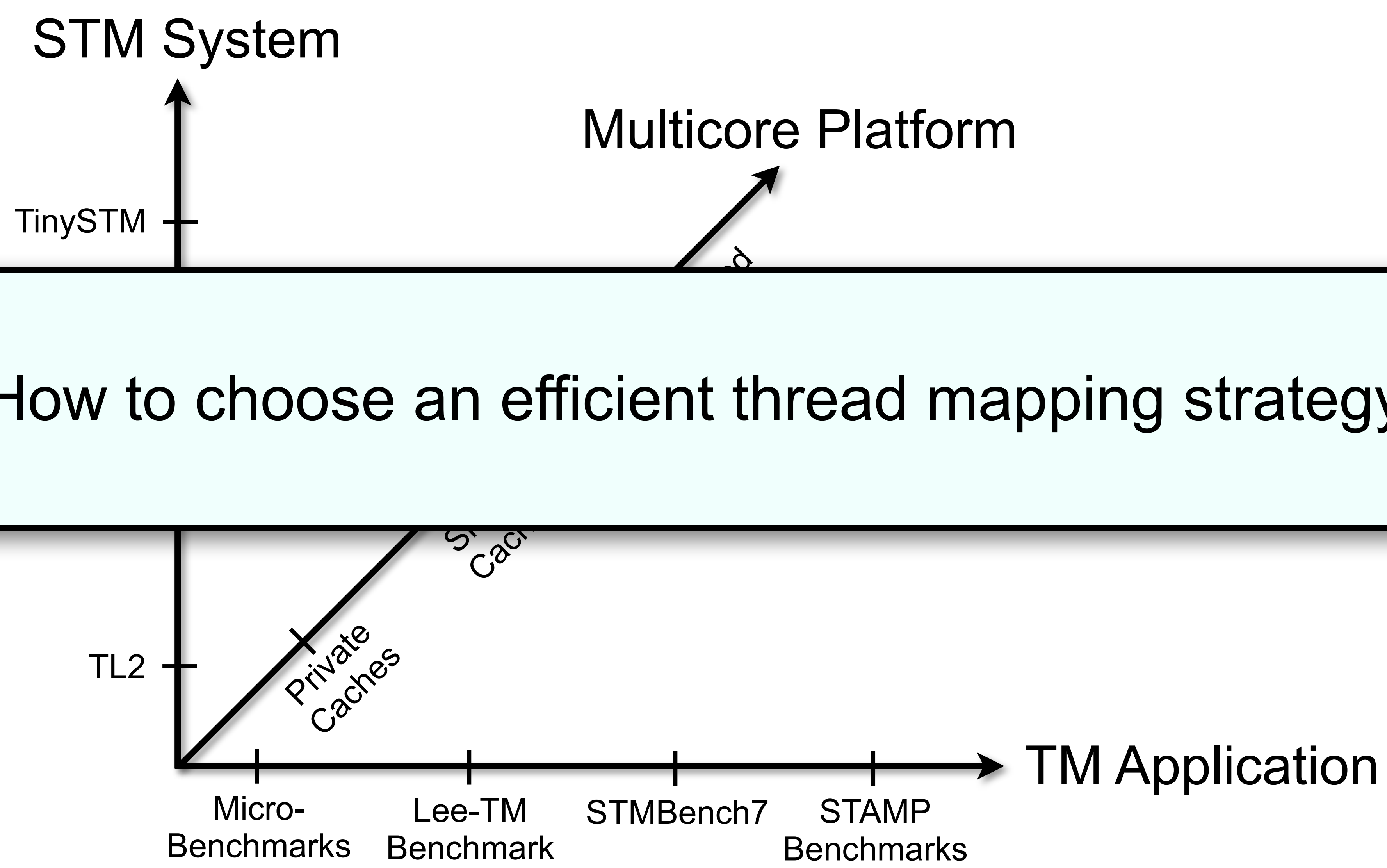
Motivation

- Problem statement:



Motivation

- Problem statement:



How to choose an efficient thread mapping strategy?

Motivation

Choosing a thread mapping for TM applications is complex...

Application	TinySTM			SwissTM			TL2		
	var (%)	*best*	*worst*	*var (%)*	*best*	*worst*	*var (%)*	*best*	*worst*
Bayes	17.6	Round-Robin	Linux	19.3	Round-Robin	Compact	–	–	–
Genome	26.8	Round-Robin	Compact	10.1	Scatter	Compact	16.7	Linux	Compact
Intruder	14.0	Compact	Scatter	6.8	Compact	Scatter	12.7	Compact	Scatter
Kmeans	10.3	Linux	Compact	10.6	Linux	Round-Robin	14.4	Round-Robin	Linux
Labyrinth	9.7	Scatter	Compact	9.5	Scatter	Round-Robin	18.6	Round-Robin	Scatter
Ssca2	25.9	Scatter	Compact	25.0	Scatter	Compact	21.6	Compact	Scatter
Vacation	9.2	Scatter	Compact	17.2	Scatter	Compact	8.0	Scatter	Compact
Yada	23.0	Compact	Scatter	18.8	Compact	Scatter	18.0	Compact	Scatter

Motivation

Choosing a thread mapping for TM applications is complex...

Application	TinySTM			SwissTM			TL2		
	var (%)	*best*	*worst*	*var (%)*	*best*	*worst*	*var (%)*	*best*	*worst*
Bayes	17.6	Round-Robin	Linux	19.3	Round-Robin	Compact	–	–	–
Genome	26.8	Round-Robin	Compact	10.1	Scatter	Compact	16.7	Linux	Compact
Intruder	14.0	Compact	Scatter	6.8	Compact	Scatter	12.7	Compact	Scatter
Kmeans	10.3	Linux	Compact	10.6	Linux	Round-Robin	14.4	Round-Robin	Linux
Labyrinth	9.7	Scatter	Compact	9.5	Scatter	Round-Robin	18.6	Round-Robin	Scatter
Ssca2	25.9	Scatter	Compact	25.0	Scatter	Compact	21.6	Compact	Scatter
Vacation	9.2	Scatter	Compact	17.2	Scatter	Compact	8.0	Scatter	Compact
Yada	23.0	Compact	Scatter	18.8	Compact	Scatter	18.0	Compact	Scatter

It is hard to determine a suitable thread mapping strategy of a STM application considering both STM system and platform characteristics.

Motivation

Choosing a thread mapping for TM applications is complex...

Application	TinySTM			SwissTM			TL2		
	var (%)	*best*	*worst*	*var (%)*	*best*	*worst*	*var (%)*	*best*	*worst*
Bayes	17.6	Round-Robin	Linux	19.3	Round-Robin	Compact	–	–	–
Genome	26.8	Round-Robin	Compact	10.1	Scatter	Compact	16.7	Linux	Compact
Intruder	14.0	Compact	Scatter	6.8	Compact	Scatter	12.7	Compact	Scatter
Kmeans	10.3	Linux	Compact	10.6	Linux	Round-Robin	14.4	Round-Robin	Linux
Labyrinth	9.7	Scatter	Compact	9.5	Scatter	Round-Robin	18.6	Round-Robin	Scatter
Ssca2	25.9	Scatter	Compact	25.0	Scatter	Compact	21.6	Compact	Scatter
Vacation	9.2	Scatter	Compact	17.2	Scatter	Compact	8.0	Scatter	Compact
Yada	23.0	Compact	Scatter	18.8	Compact	Scatter	18.0	Compact	Scatter

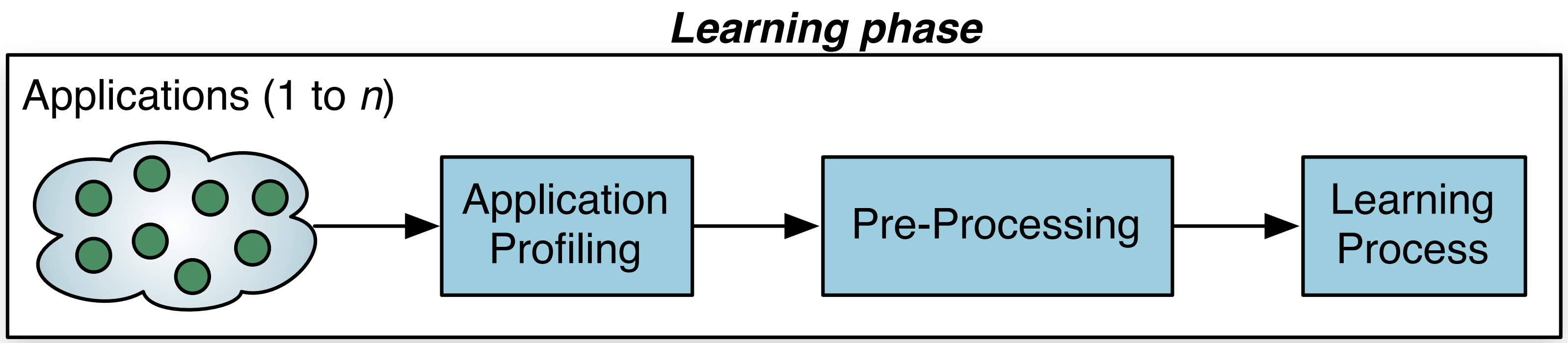
It is hard to determine a suitable thread mapping strategy of a STM application considering both STM system and platform characteristics.

Our approach:
Machine Learning

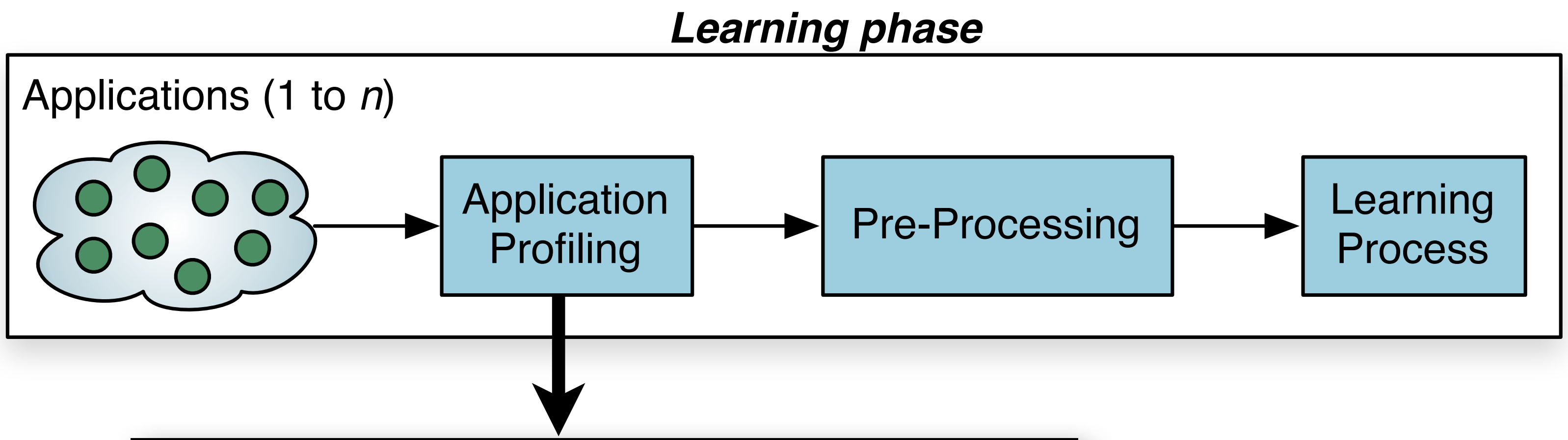
ML to Thread Mapping on STM

- Why Machine Learning (ML)?
  - Can model the behavior of complex interactions between applications, STM systems and platforms
  - Portable solution to predict future behaviors based on a priori profiled runs
- Proposal
  - Use of ML to automatically infer a suitable thread mapping strategy to be applied considering the application, STM system and platform characteristics

ML to Thread Mapping on STM



ML to Thread Mapping on STM



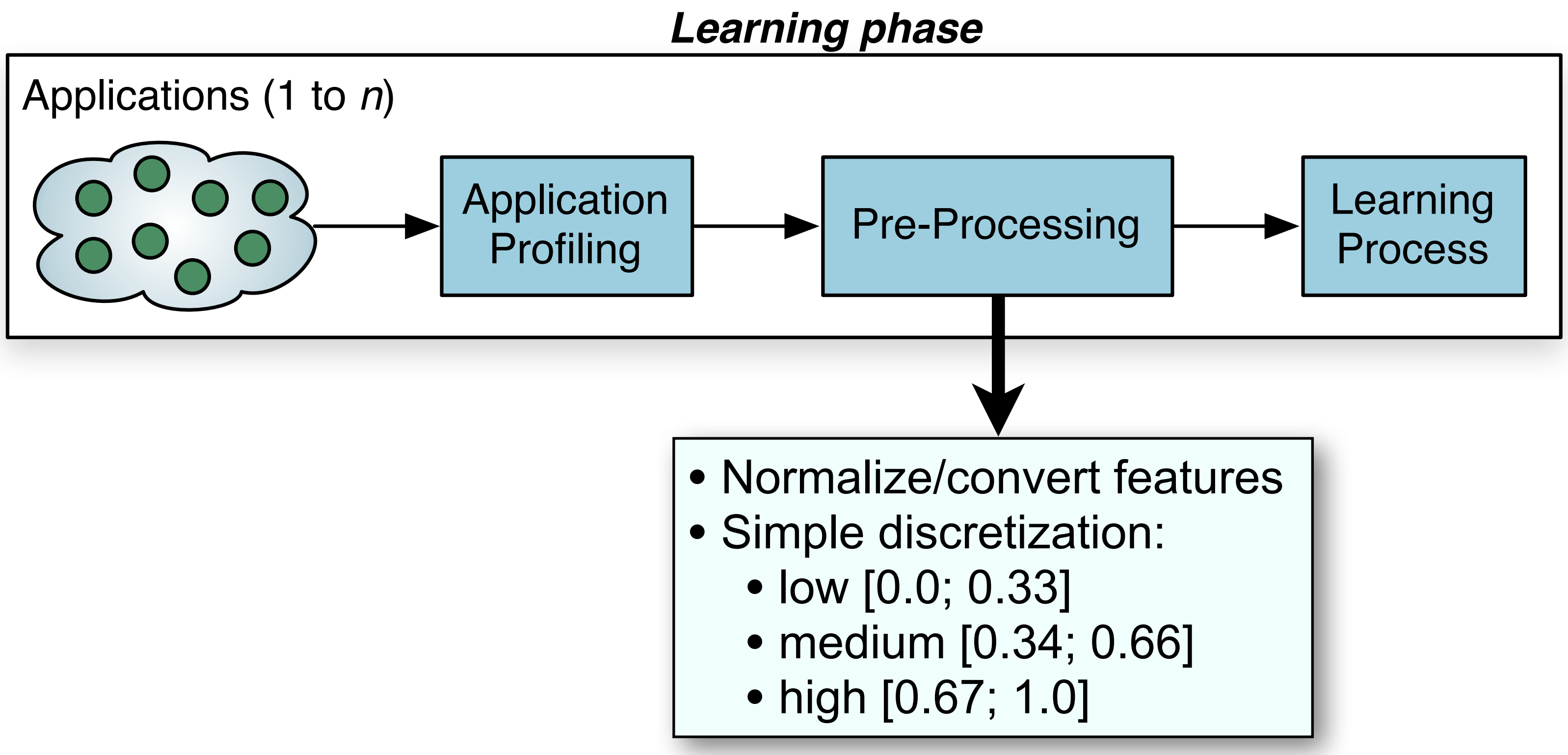
Profiling information (features):

- Tx time ratio (%)
- Tx abort ratio (%)
- Conflict detection and resolution policies:
  - eager/lazy
  - suicide/backoff
- LLC miss ratio (%)

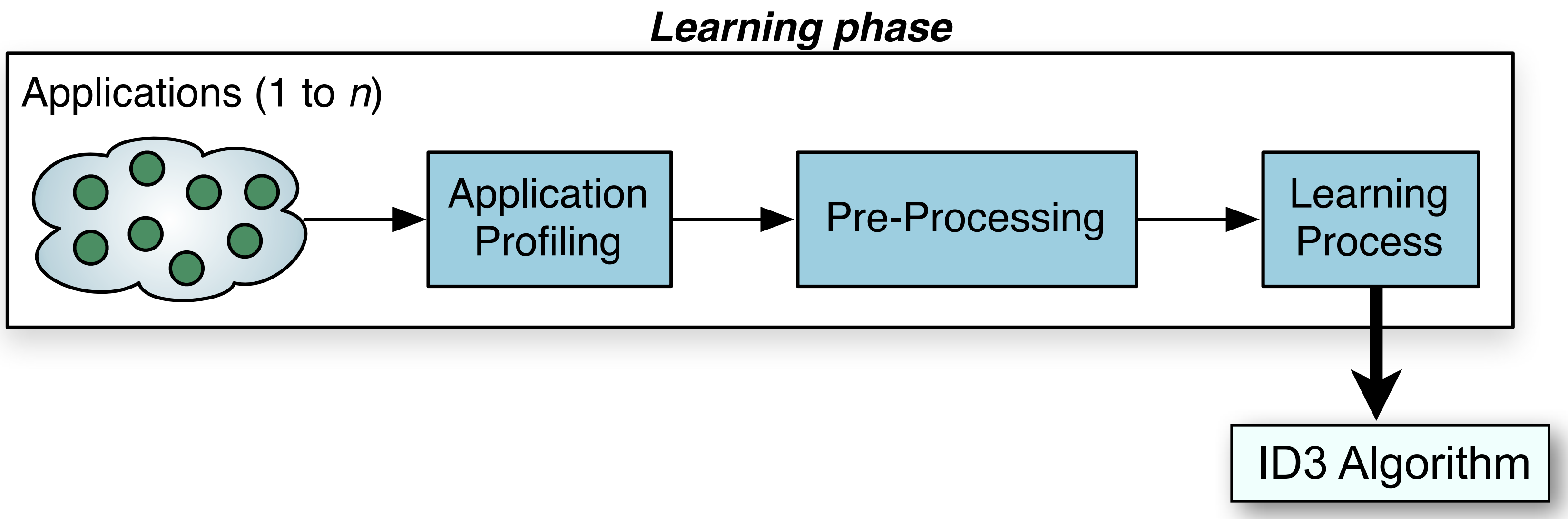
Performance metric: execution time

- Thread mapping strategies:
  - linux
  - compact
  - scatter
  - round-robin

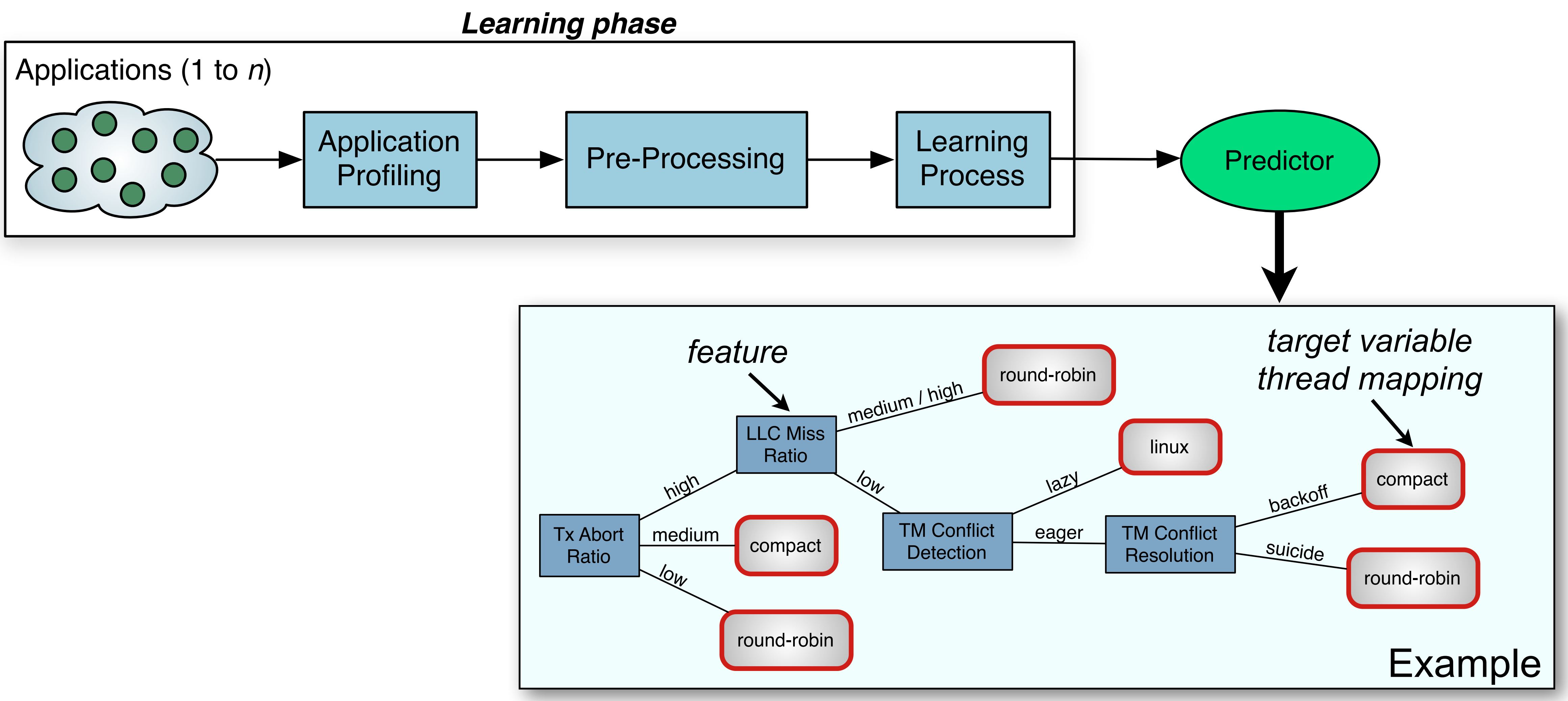
ML to Thread Mapping on STM



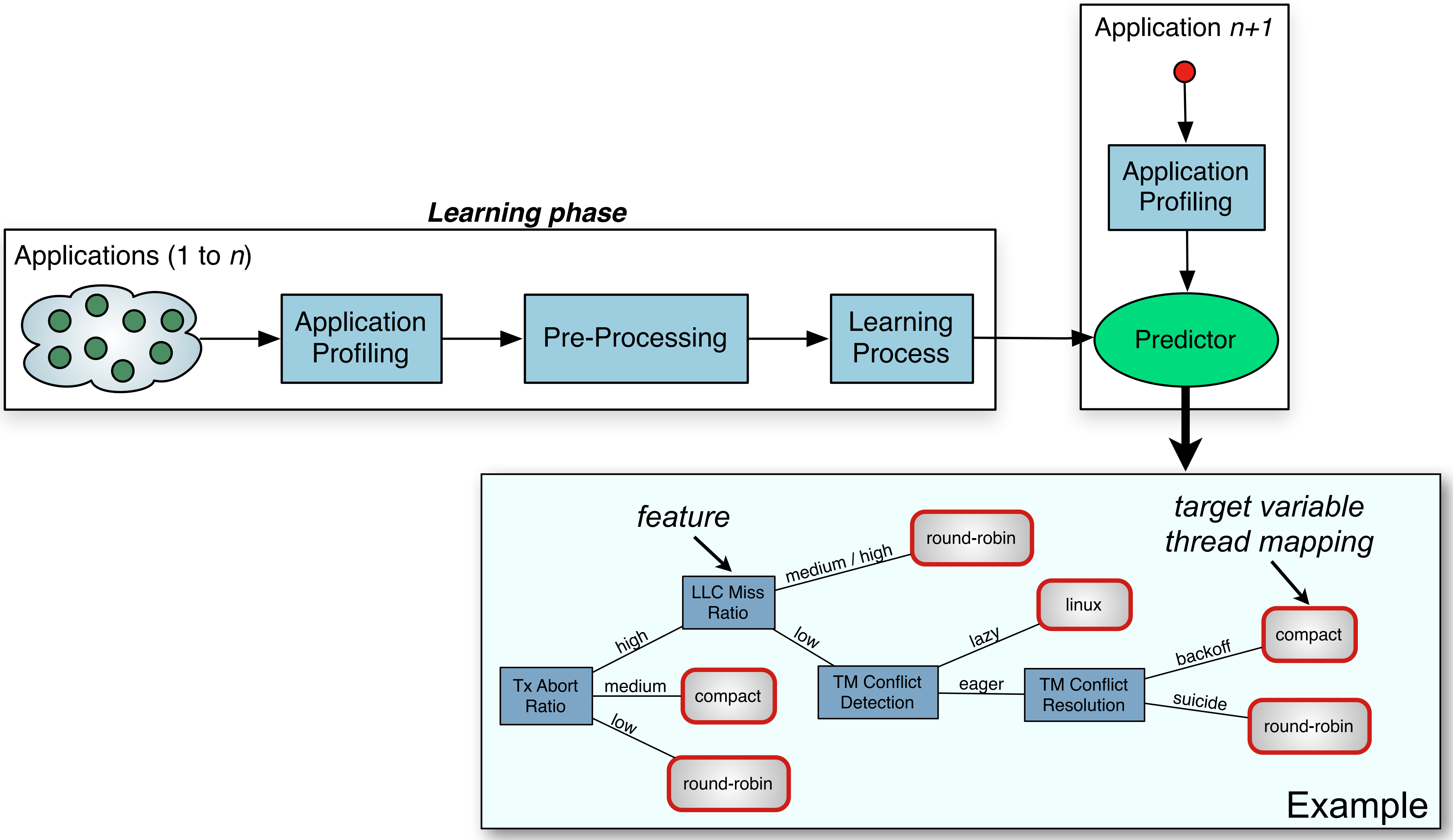
ML to Thread Mapping on STM



ML to Thread Mapping on STM



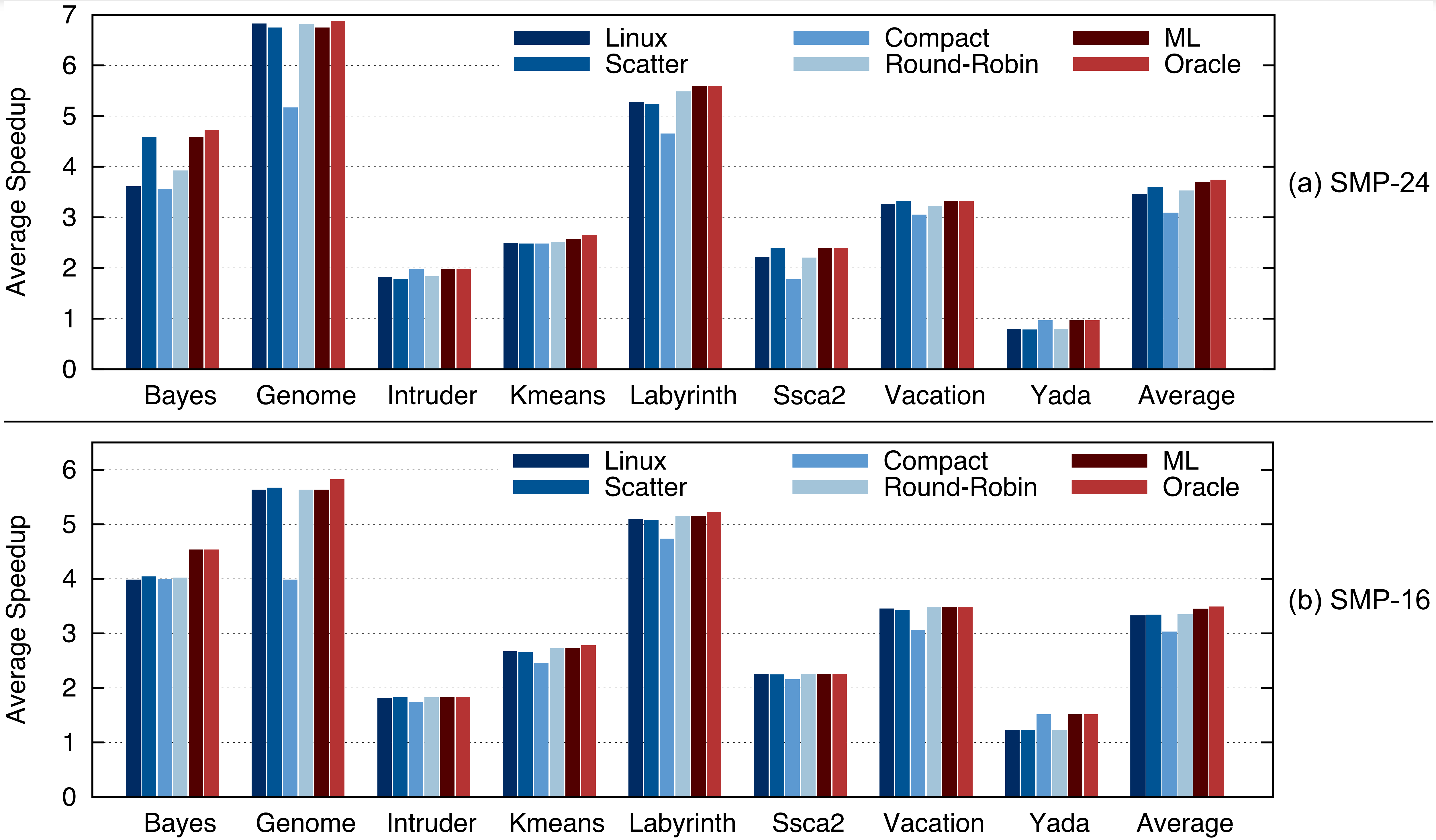
ML to Thread Mapping on STM



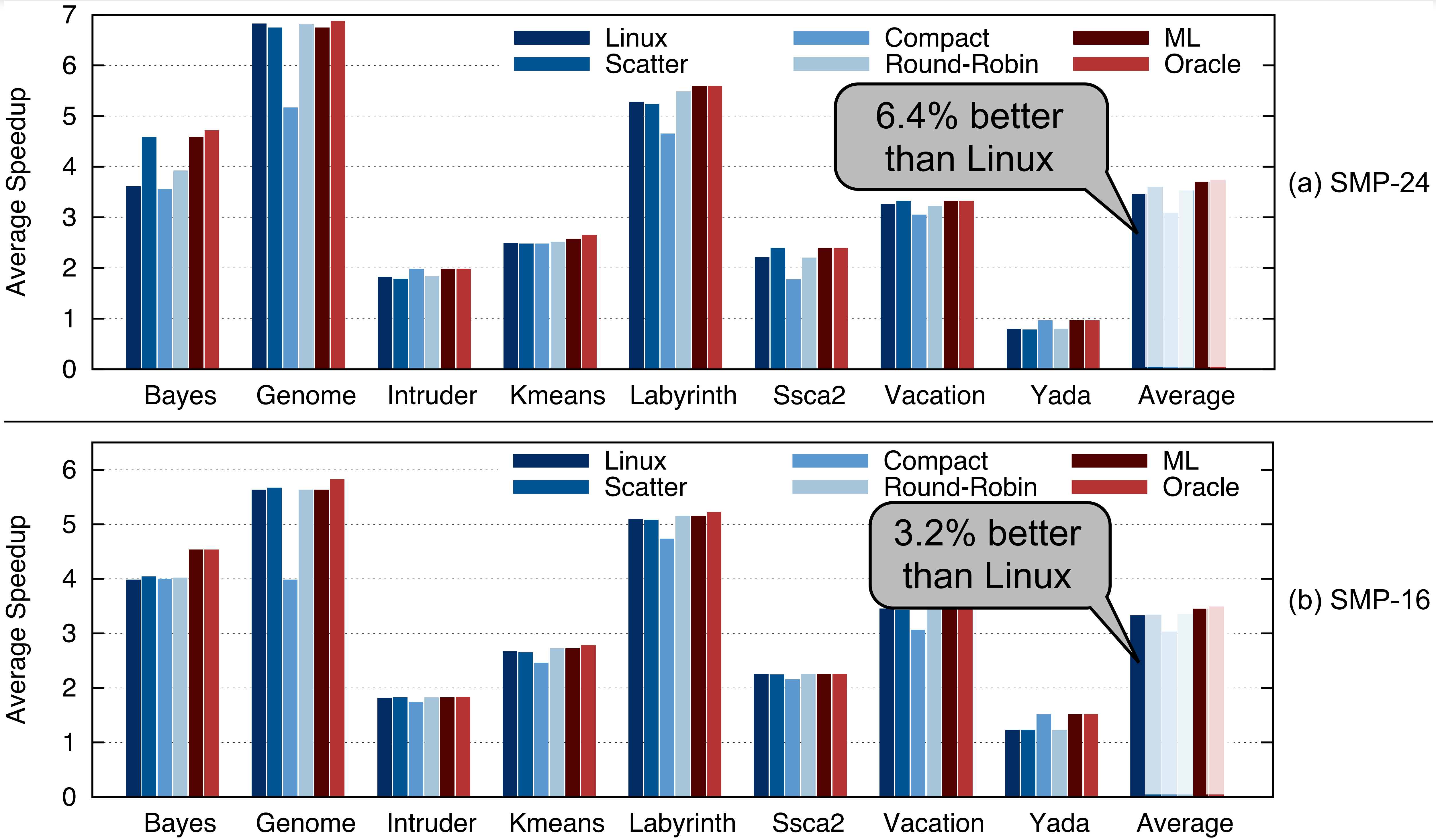
Static Thread Mapping

- Experimental evaluation (HiPC'11)
  - Profile several TM applications from STAMP
  - Construct the ML-based predictor
  - Apply the predicted thread mapping strategy *statically*:
    - At the beginning of the execution of the application
    - Remains unchanged during the whole execution

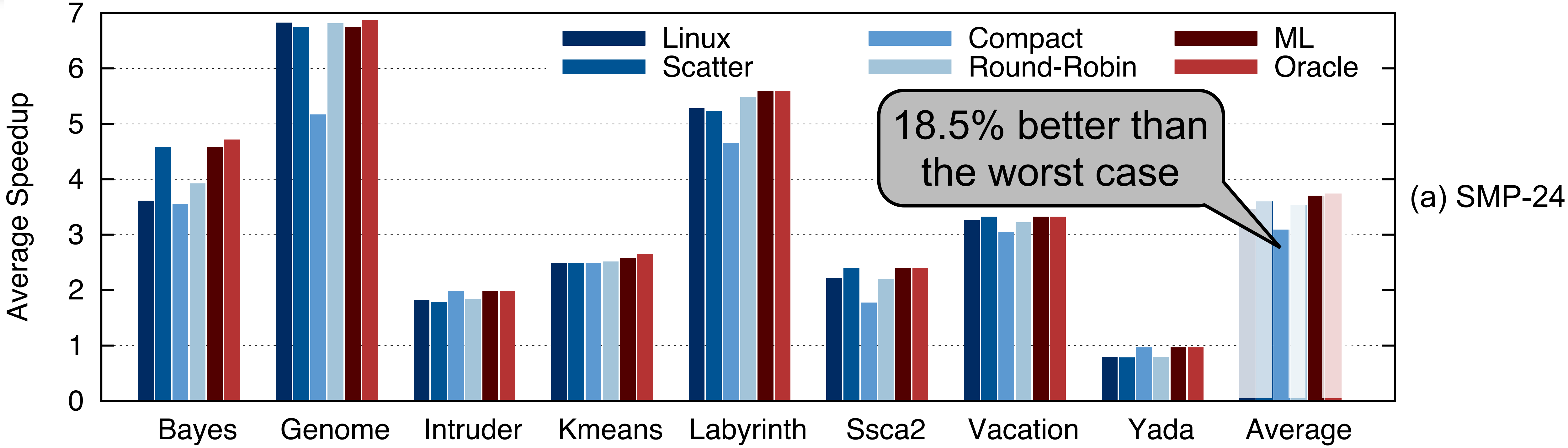
Static Thread Mapping

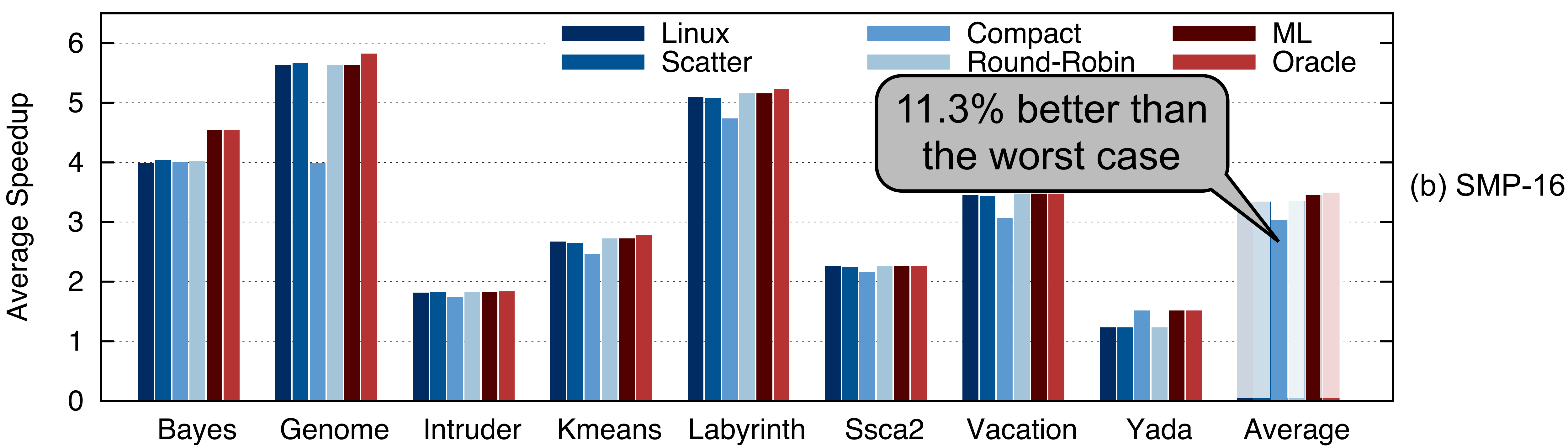


Static Thread Mapping

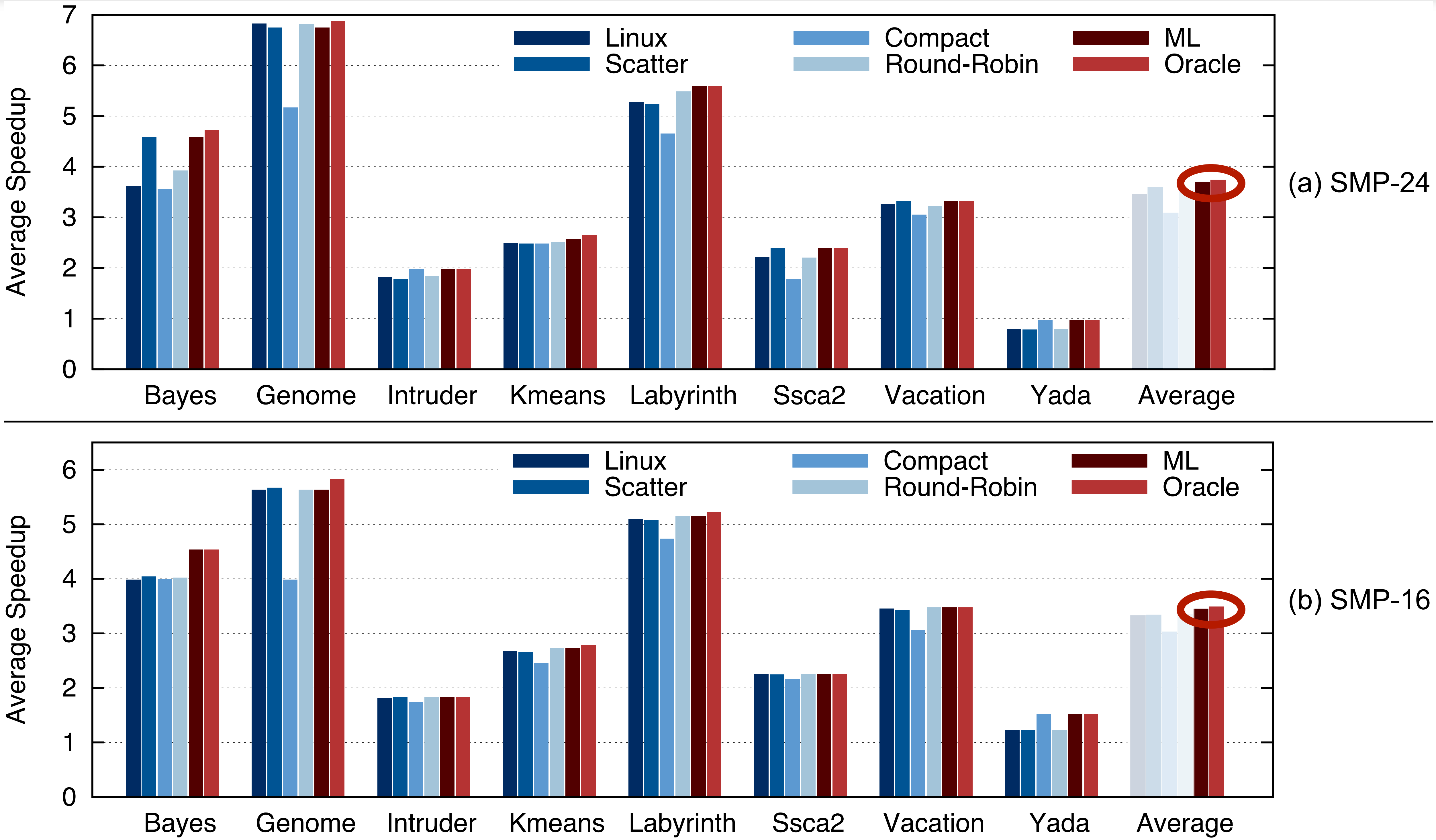


Static Thread Mapping





Static Thread Mapping



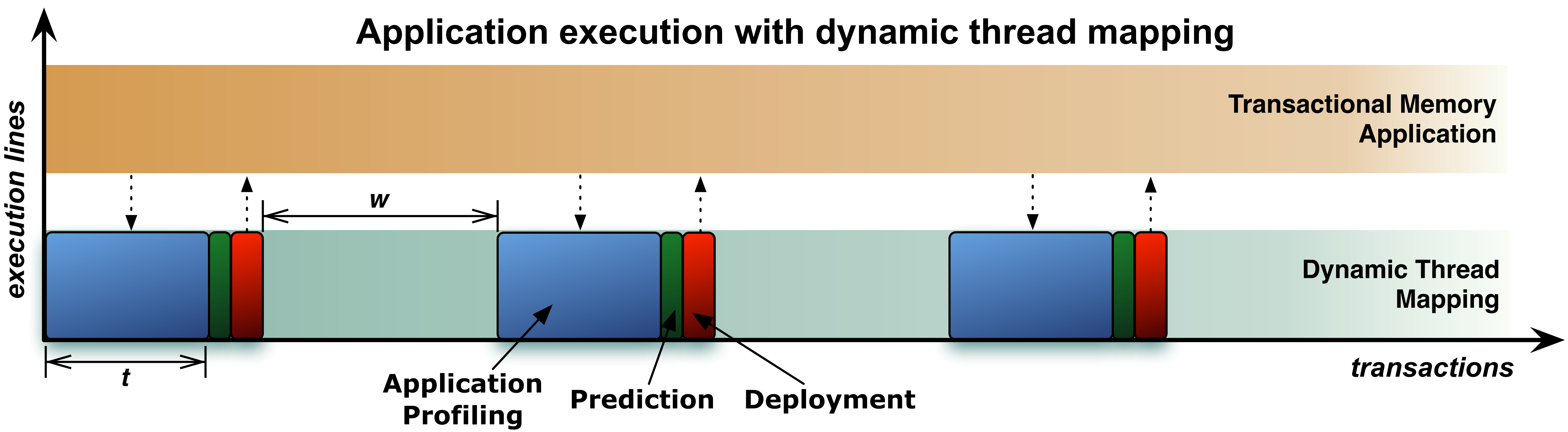
ML-based approach is within 1% of the oracle performance!

Dynamic Thread Mapping

- It is expected that more complex applications will make use of TM in a near future
  - TM support on GCC 4.7, Intel “Haswell” processor, ...
- Need of more dynamic approaches for thread mapping
  - Applications may be composed of more diverse workloads
  - Workloads may go through different execution phases
  - Each phase can potentially have different transactional characteristics

Dynamic Thread Mapping

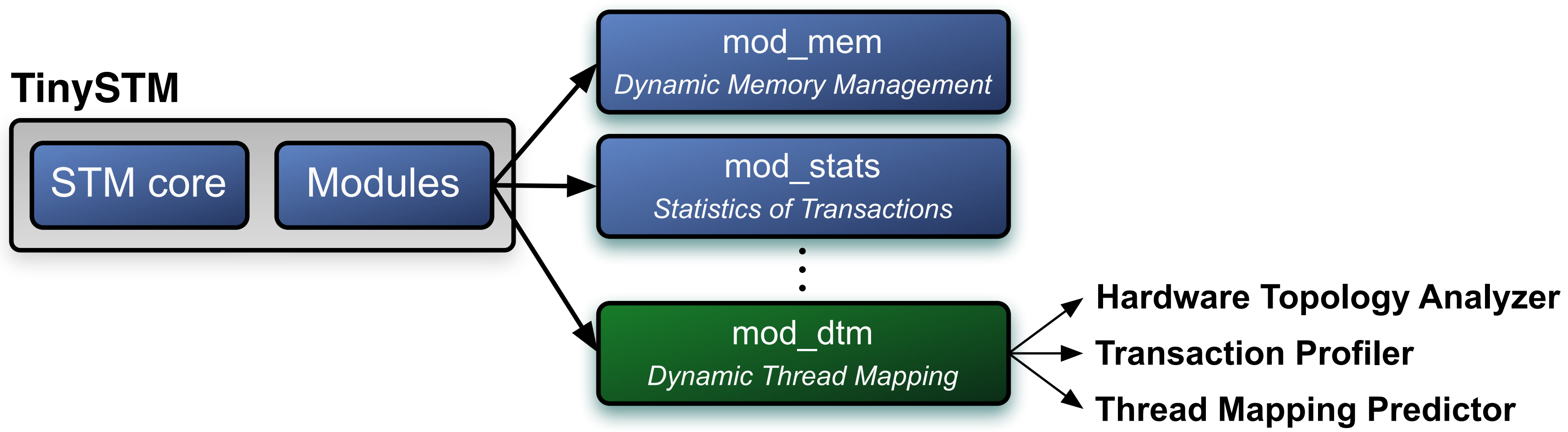
- Dynamic approach
  1. ***t*** transactions are profiled at runtime
  2. the profiled information is used by the ML-based predictor
  3. the thread mapping strategy is applied and remains unchanged during ***w***
  4. repeat 1, 2 and 3 until the application ends



- Parameters ***t*** and ***w***:
  - Specified by the number of committed transactions
  - Hill-climbing strategy to automatically adapt them at runtime

Dynamic Thread Mapping

- Implementation in TinySTM
  - Modular structure can be easily extended with new features
  - *mod_dtm*: module for transparent dynamic thread mapping



- Components
  - **Hardware Topology Analyzer**: gathers information from the platform
  - **Thread Mapping Predictor**: decision tree generated by the ML
  - **Transaction Profiler**: performs runtime profiling

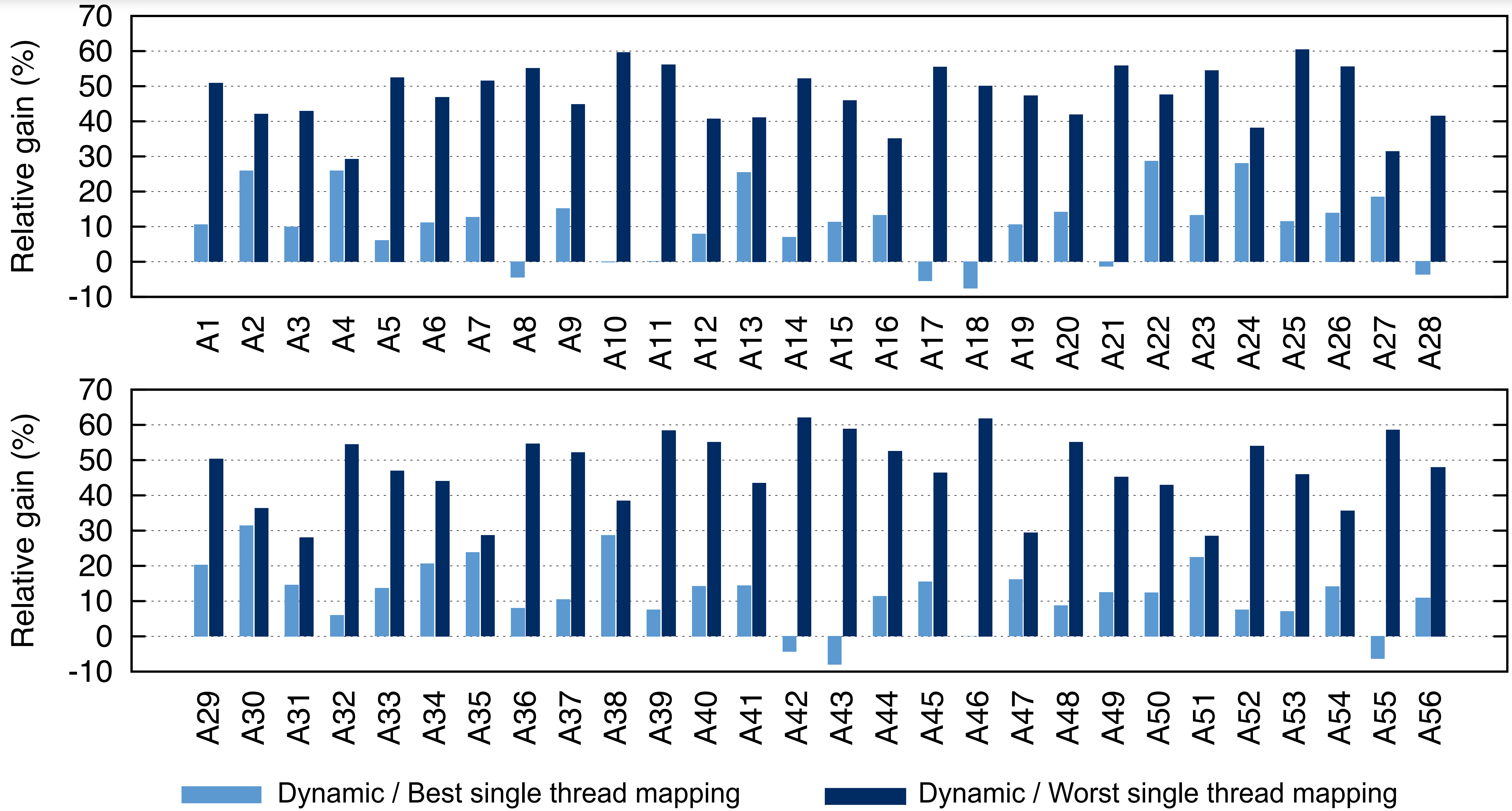
Dynamic Thread Mapping

- Experimental evaluation
  - We used EigenBench to create applications with different phases
  - We varied 4 out of 8 orthogonal transactional characteristics, assuming two possible discrete values for each one

Characteristic	**Definition**	**Values**
Tx Length	number of shared accesses per transaction	short (≤ 64) long (≥ 128)
Contention	probability of conflict	low-conflicting ($< 30\%$) contentious ($\geq 30\%$)
Density	fraction of the time spent inside transactions to the total execution time	sparse ($< 80\%$) dense ($\geq 80\%$)

  - All possible combinations generate 8 different **workloads** (*W1*, ..., *W8*)
  - We created applications with **3 phases**, thus each application will be composed by 3 distinct workloads
    - A1 = {*W1, W2, W3*}, A2 = {*W1, W2, W4*}, ..., A56 = {*W5, W6, W7*}
  - Phases are parallelized using Pthreads and there is no synchronization barrier between phases

Dynamic Thread Mapping

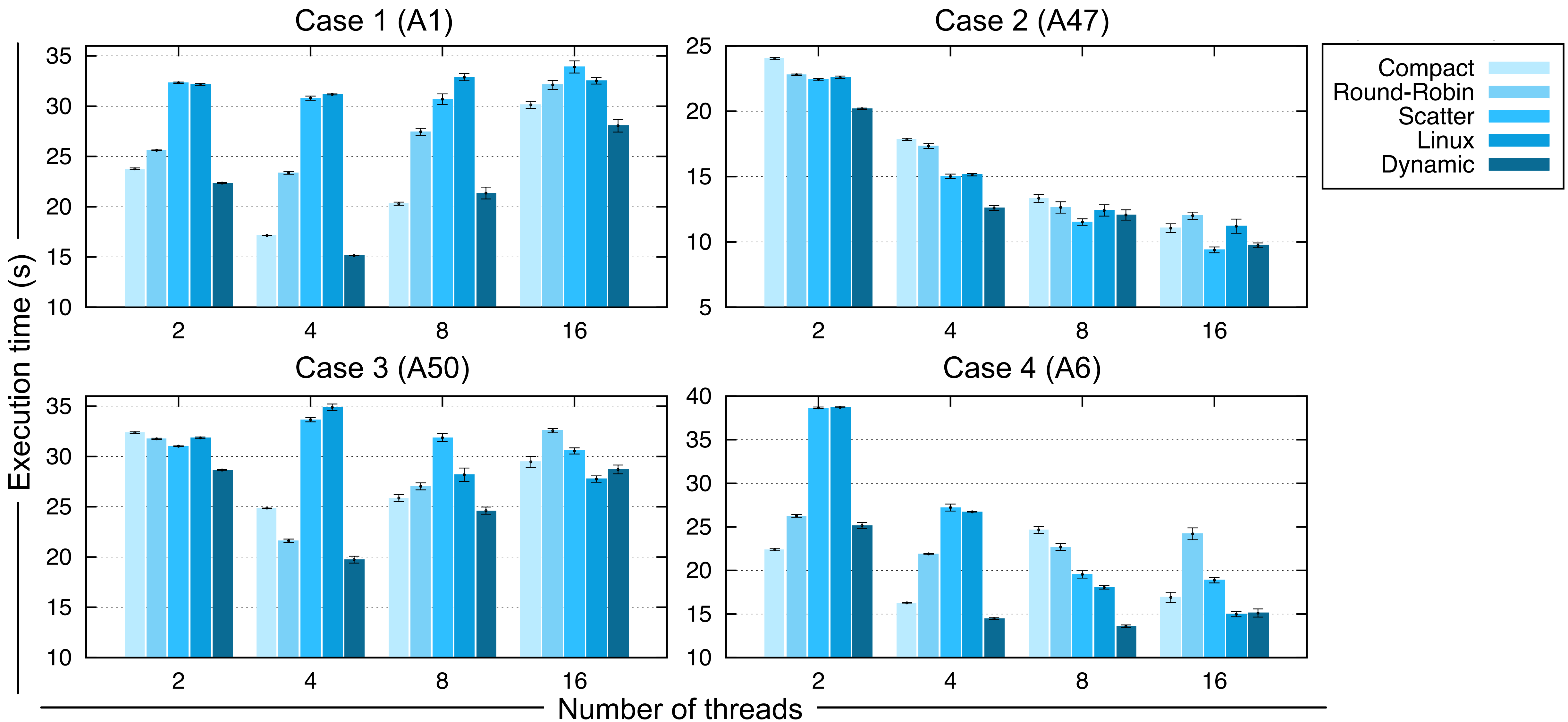


Performance gains:

Up to 31% and 62% when comparing to the best and worst single thread mappings respectively

Dynamic Thread Mapping

Varying concurrency



Conclusions

- Predicting a suitable thread mapping strategy for TM applications is not trivial
  - Applications with different behaviors
  - Several conflict detection and resolution mechanisms
  - Platform specifics

- ML-based approach to thread mapping for TM applications
  - Automatically infer an appropriate thread mapping
  - Application, STM system and platform are taken into account
  - Portable and can be easily extended to consider other features

- Future works
  - Extend the predictor to consider a broader range of conflict detection and resolution policies
  - Consider more TM characteristics to build more diverse applications
  - Use other ML algorithms to build new predictors