Optimal Task Placement to Improve Cache Performance

Sebastian Altmeyer and **Gernot Gebhard**
Compiler Design Lab
Saarland University

Wednesday, 3rd October, 2007

Introduction

Scope

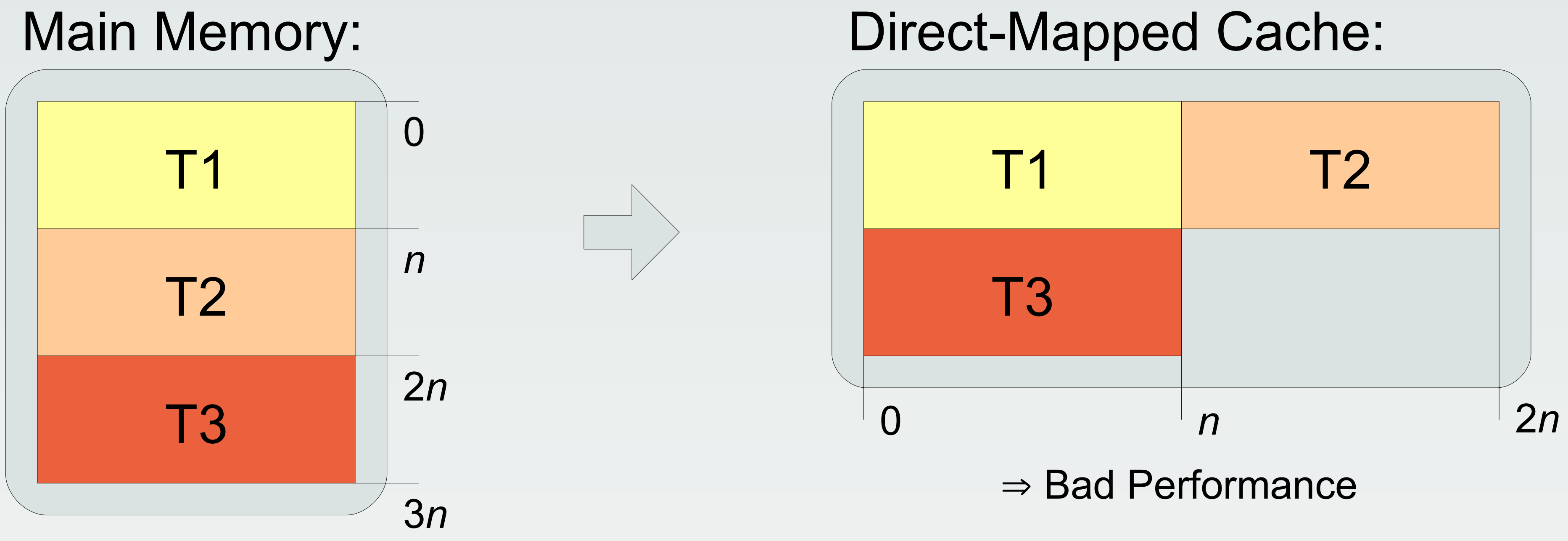
- Single-processor Embedded System
- Cache using any Replacement Policy
- Straight memory-to-cache mapping
  - No virtual memory
  - Memory maps consecutively to cache sets
- Preemptive task scheduling
- Schedule statically known

Motivation

- Cache performance optimization reasonable
  - Increasing demands at Embedded Systems
  - Steadily growing cache size
- Preemptive task scheduling
  - Some task sets only schedulable in this fashion
  - Renders previous timing guarantees invalid
  - Induces dynamic context-switch costs

Observation

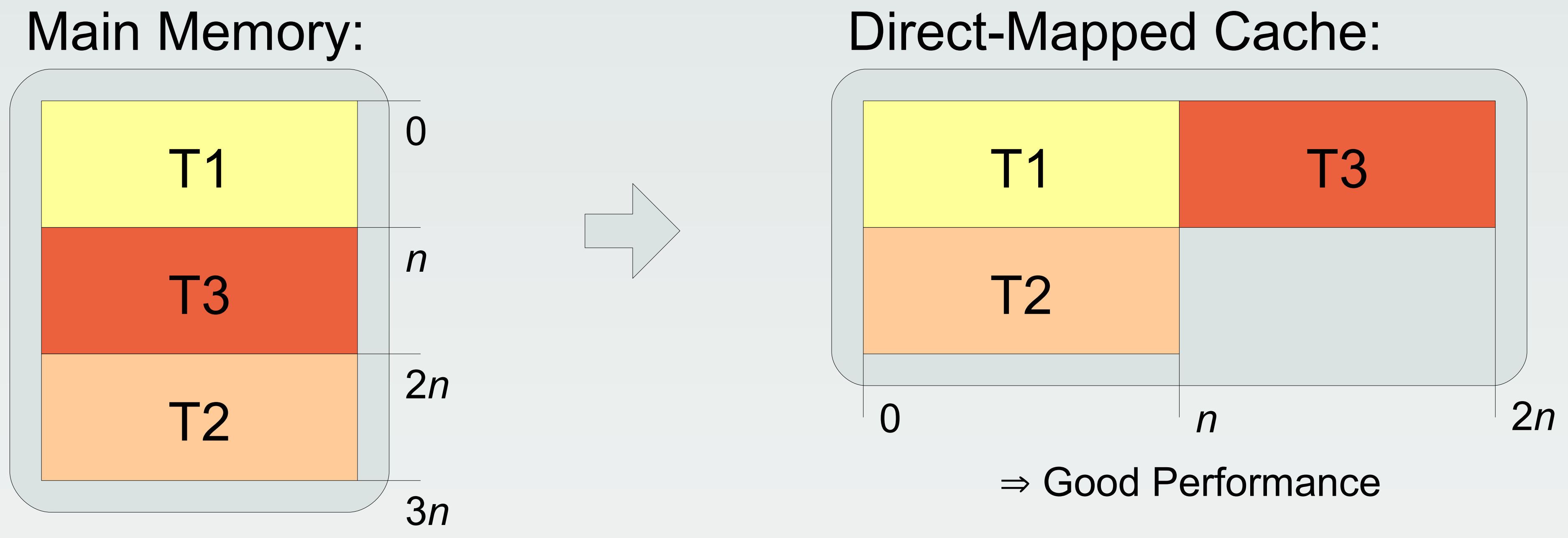
- Memory placement influences cache performance
- Example:
  - Tasks T1 and T2 execute mutual exclusively
  - Task T3 executes frequently, interrupting T1 and T2



⇒ Bad Performance

Observation cont.

- Memory placement influences cache performance
- Example:
  - Tasks T1 and T2 execute mutual exclusively
  - Task T3 executes frequently, interrupting T1 and T2



$\Rightarrow$ Good Performance

Goals

- Improve performance of Embedded Systems by
  - Trying to keep cached data persistent
  - Reducing context-switch costs
- Make static timing analyses feasible by
  - Identifying persistent cache sets
  - Being able to derive tight WCET bounds

Optimization Method

Method

- Determine performance factors
  - Cache configuration
  - Tasks
  - Schedule
- Compute optimal task placement
  - Building cost function
  - Determining start address of each task
- Arrange code according to placement

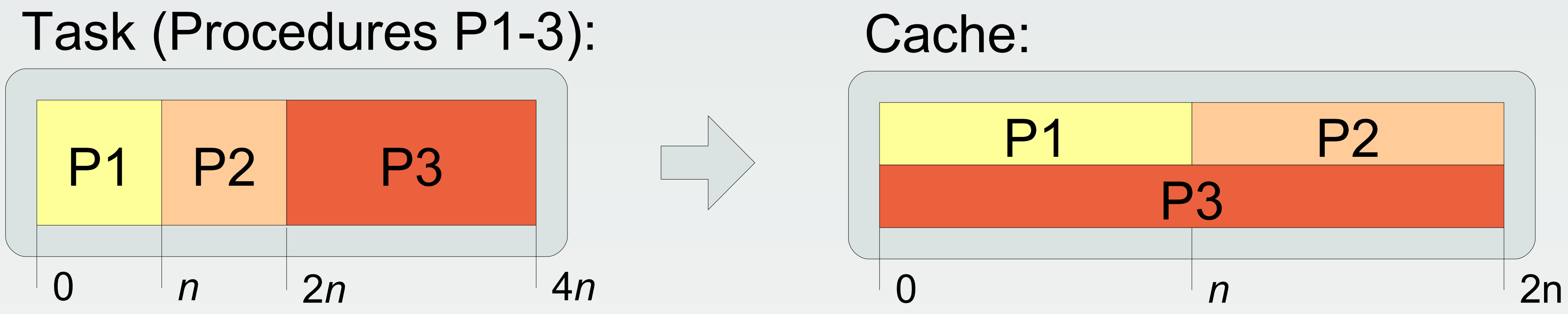
Performance Factor: Cache

- Cache size
  - Determines room for possible optimization
  - Some tasks should fit inside the cache
- Associativity and Replacement Policy
  - Strongly influences cache performance
  - Affects predictability of cache behavior (*minimum life span* [1]):

		Replacement Policy	
		PLRU	LRU
Associativity	1	1	1
	2	2	2
	4	3	4
	n	$\log_2(n) + 1$	n

Performance Factor: Task

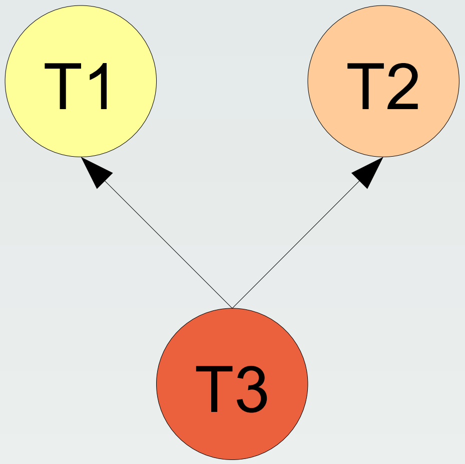
- Notion *Task* used ambiguously
  - Describes an operation processing data
  - Denotes a set of interdependent procedures
- Performance-affecting factors
  - *Period*: Indicates severity of data being evicted
  - *Start Address* & *Size*: Determine occupied cache sets
- Example:



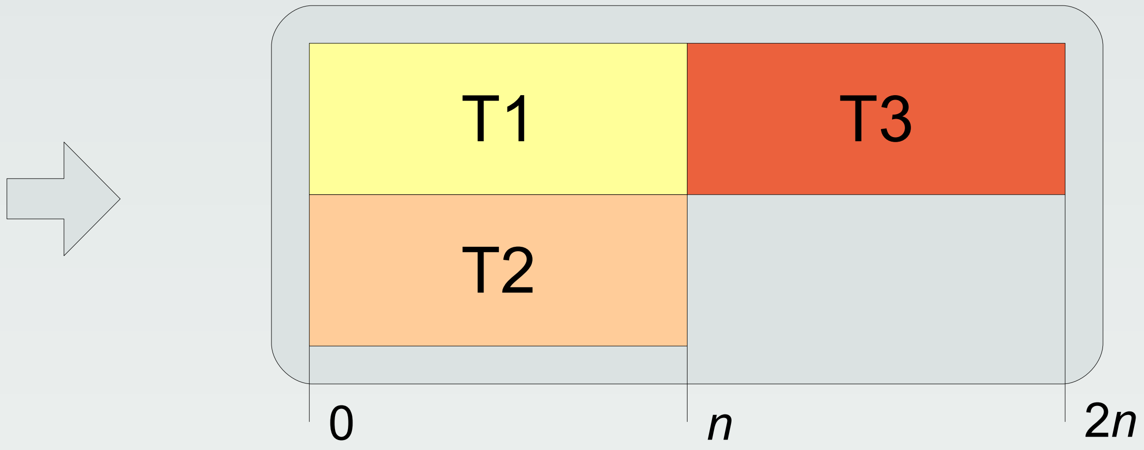
Performance Factor: Schedule

- Schedule induces *Task Interdependency Relation*
  - Defines which tasks a task might interrupt
  - Conflicting tasks should avoid each other
- Example:

Task Interdependency:



Optimal Task Placement:

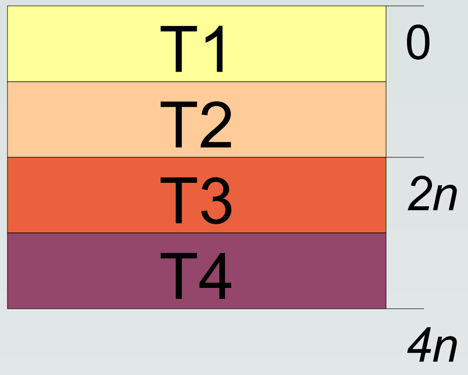


Optimization Problem

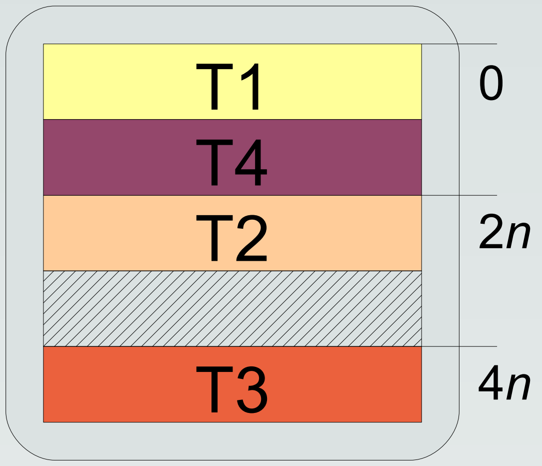
- Find task placement with minimal costs
  - Count conflicts with preempting tasks for cache sets
  - Ignore if number of conflicts < *minimum life span*
  - Weight conflicts proportional to task period
- Compute optimal solution via ILP
  - Complexity linear in number of tasks and cache sets

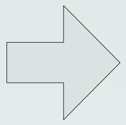
Example

Taskset:

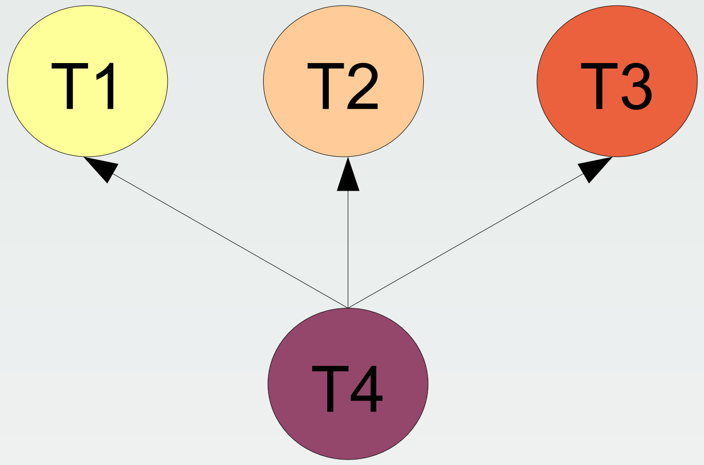


Main Memory:

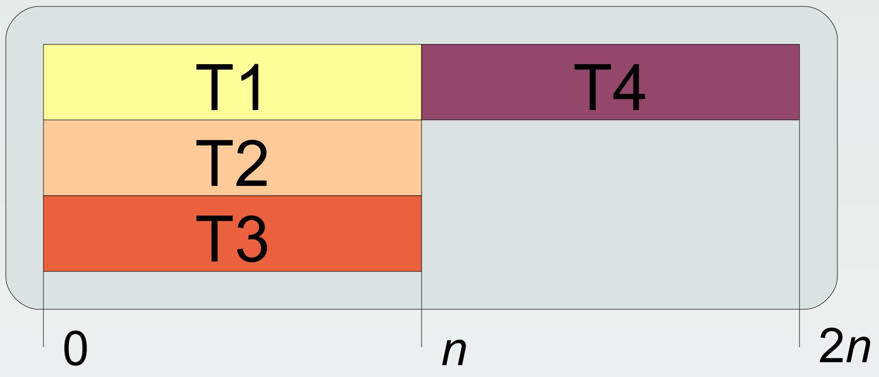




Task Interdependency:



Direct-Mapped Cache:

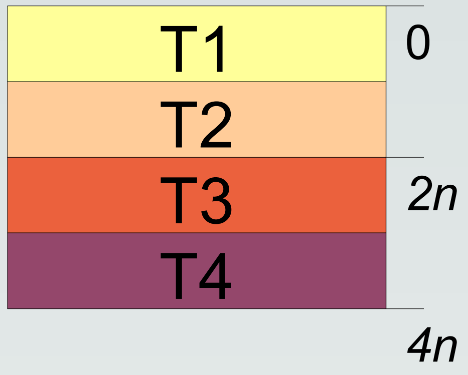


Simplified Optimization Problem

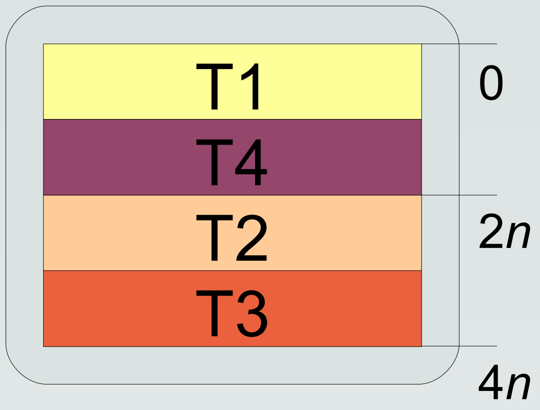
- Problems of ILP-based approach
  - Introduces memory gaps
  - NP-completeness (*k*-colorability [2])
  - Long solving time for large tasksets
- Idea: Reduce search space
  - Always arrange tasks consecutively in memory
  - Determine optimal permutation of tasks
- Simulated Annealing
  - Approximate optimal solution

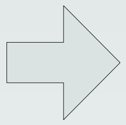
Example

Taskset:

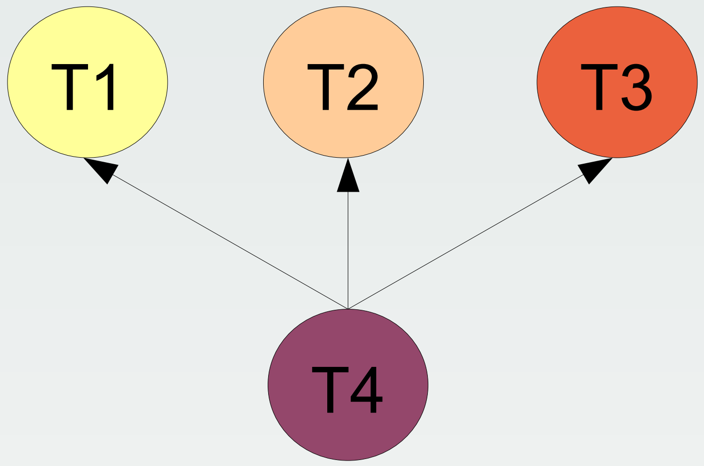


Main Memory:

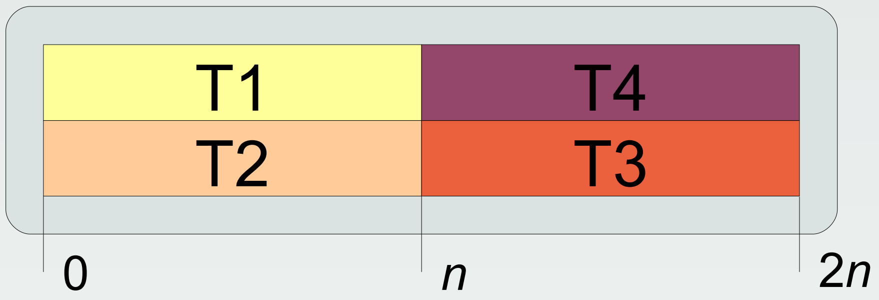




Task Interdependency:

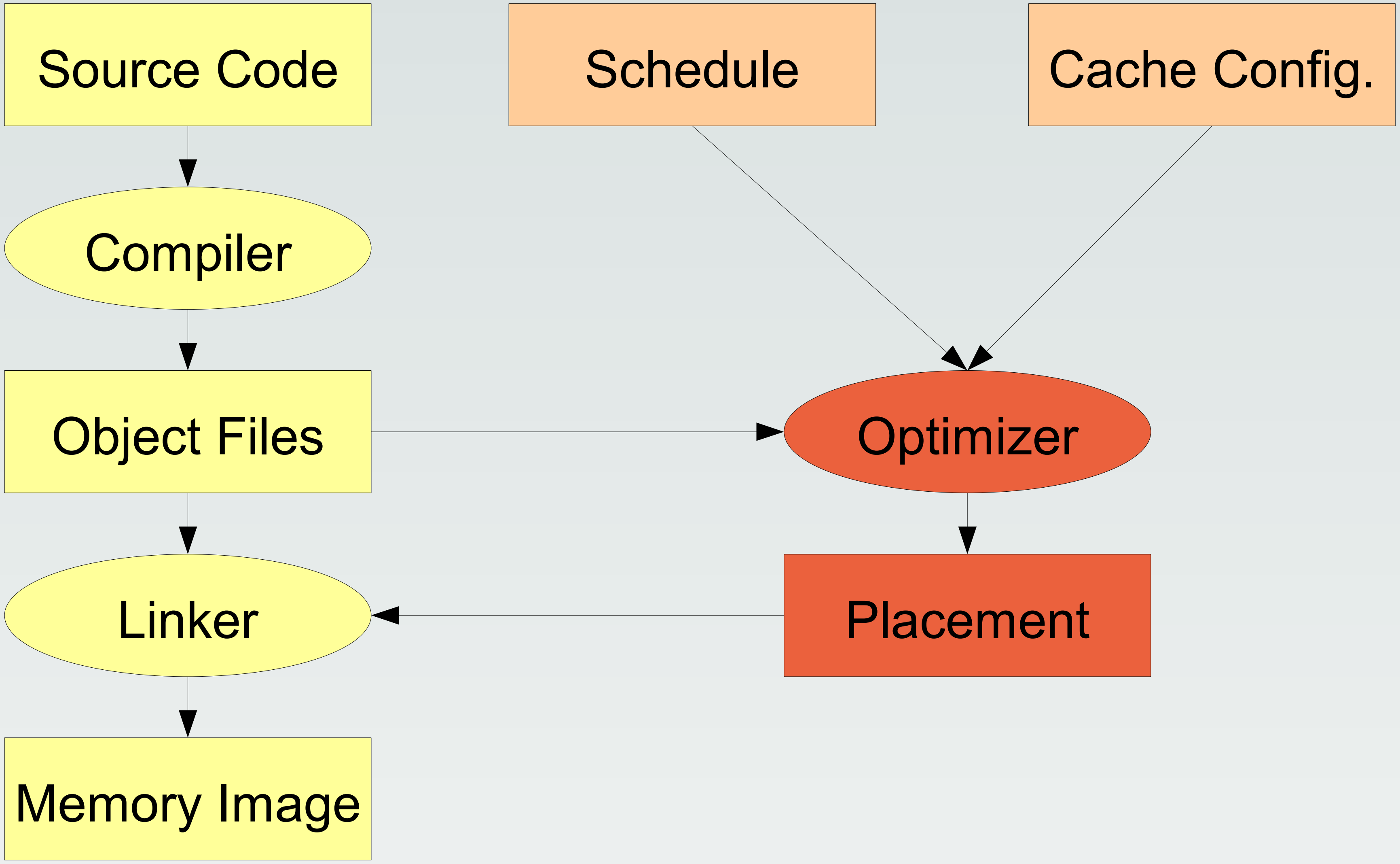


Direct-Mapped Cache:



Practice

Task Placement Framework

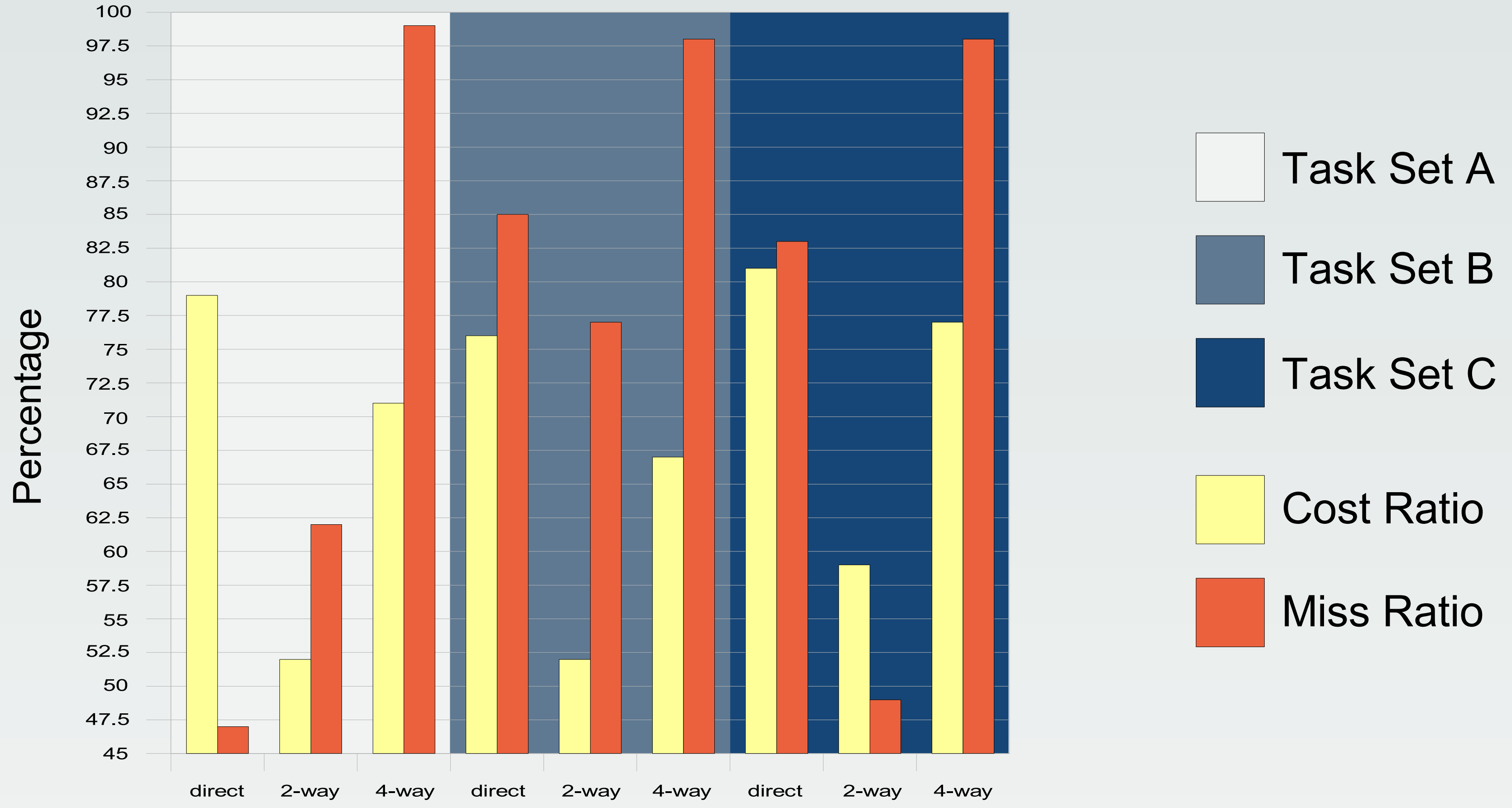


Experimental Results

- Optimized three task sets
  - Selected ten tasks from WCET Benchmark [3]
  - Executed under RTEMS operating system [4]
  - Simulated with ARM7 emulation MPARM [5]
- Performed optimization for three caches (LRU)
  - 16kb direct-mapped
  - 32kb two-way set-associative
  - 32kb four-way set-associative
- Task sets do not fit in any cache

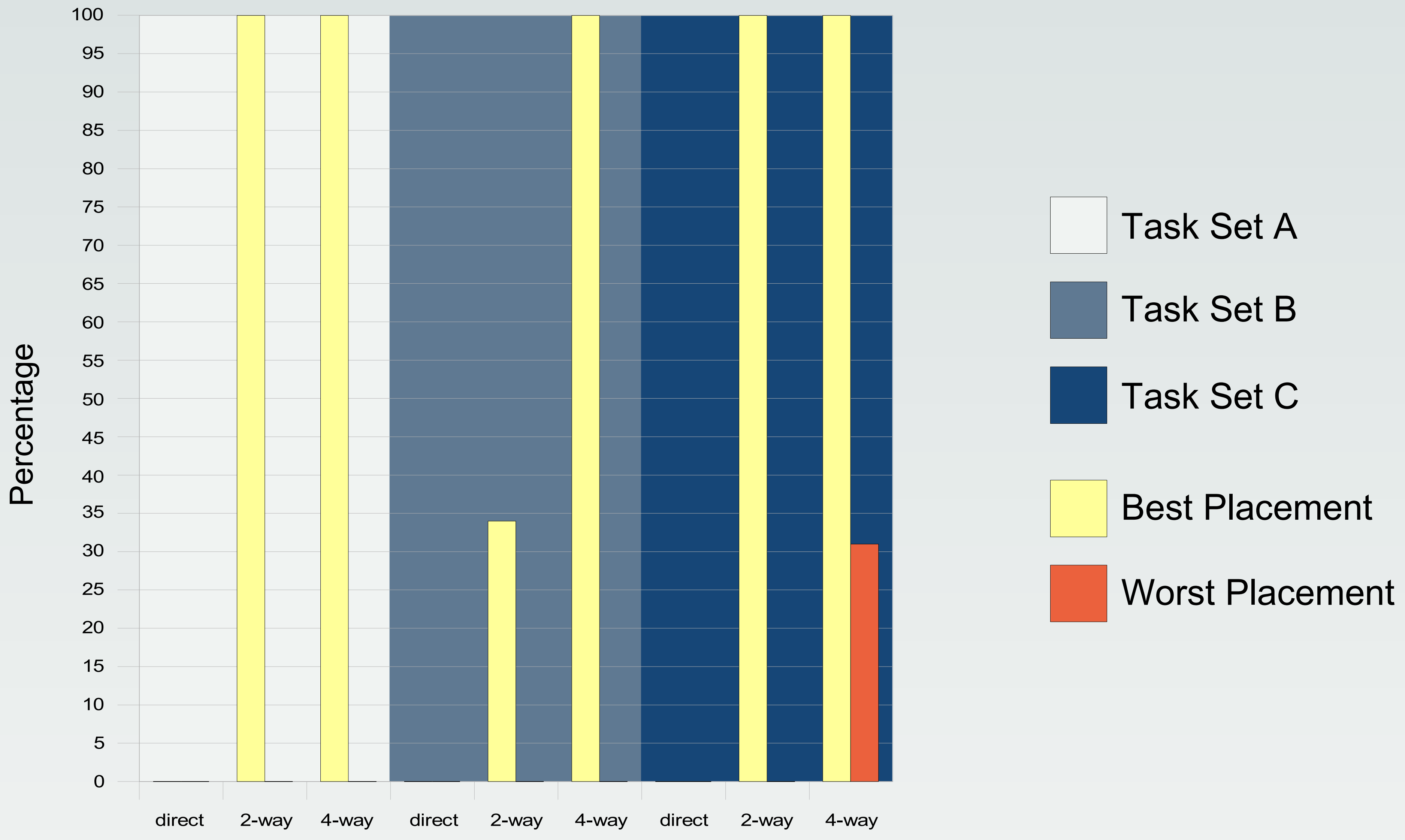
Experimental Results cont.

- Cost-function compared to cache misses
  - Cheaper placement leads to better avg. performance
  - Model not (yet) accurate enough



Experimental Results cont.

- Persistent cache sets for specific task (*compress*)



Conclusion

Achievements

- New method to optimize cache-performance
  - Program code needs not be modified
  - Arrange instructions and data differently in memory
- Computation of optimal task placement
  - Globally minimizes threat of eviction
  - Leads to better average performance
- Classification of cache sets (non-/persistent)
  - Allow tight timing guarantees for preemptive scheduling

Future Work

- Improve cost function
  - Weight loops differently than straight-lined code
- Place procedures instead of whole tasks
  - Allows higher variability
  - Achieve better results
- Restrict to preemption points
- Evaluation using real-world task sets

References

[1] J. Reineke, D. Grund, C. Berg, and R. Wilhelm. Predictability of Cache Replacement Policies. Reports of SFB/TR 14 AVACS 9, SFB/TR 14 AVACS, September 2006.

[2] C. Guillon, F. Rastello, T. Bidault, and F. Bouchez. Procedure placement using temporal-ordering information: Dealing with code size expansion. *Journal of Embedded Computing*, 1(4):437–459, 2005.

[3] Benchmarks: http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

[4] RTEMS Operating System: http://www.rtems.com/

[5] MPARM: http://www-micrel.deis.unibo.it/sitonew/research/mparm.html