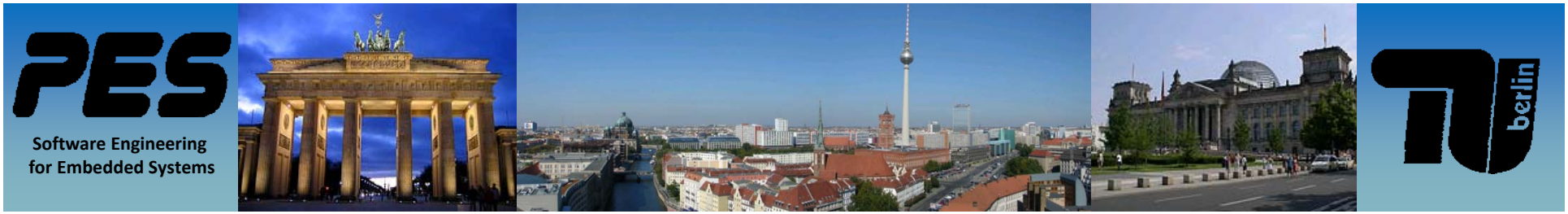


Intelligent Task Mapping for MPSoCs using Machine Learning

Dirk Tetzlaff
Technical University of Berlin

3rd Workshop on Mapping of Applications to MPSoCs
June 30th, 2010

Task Mapping for MPSoCs

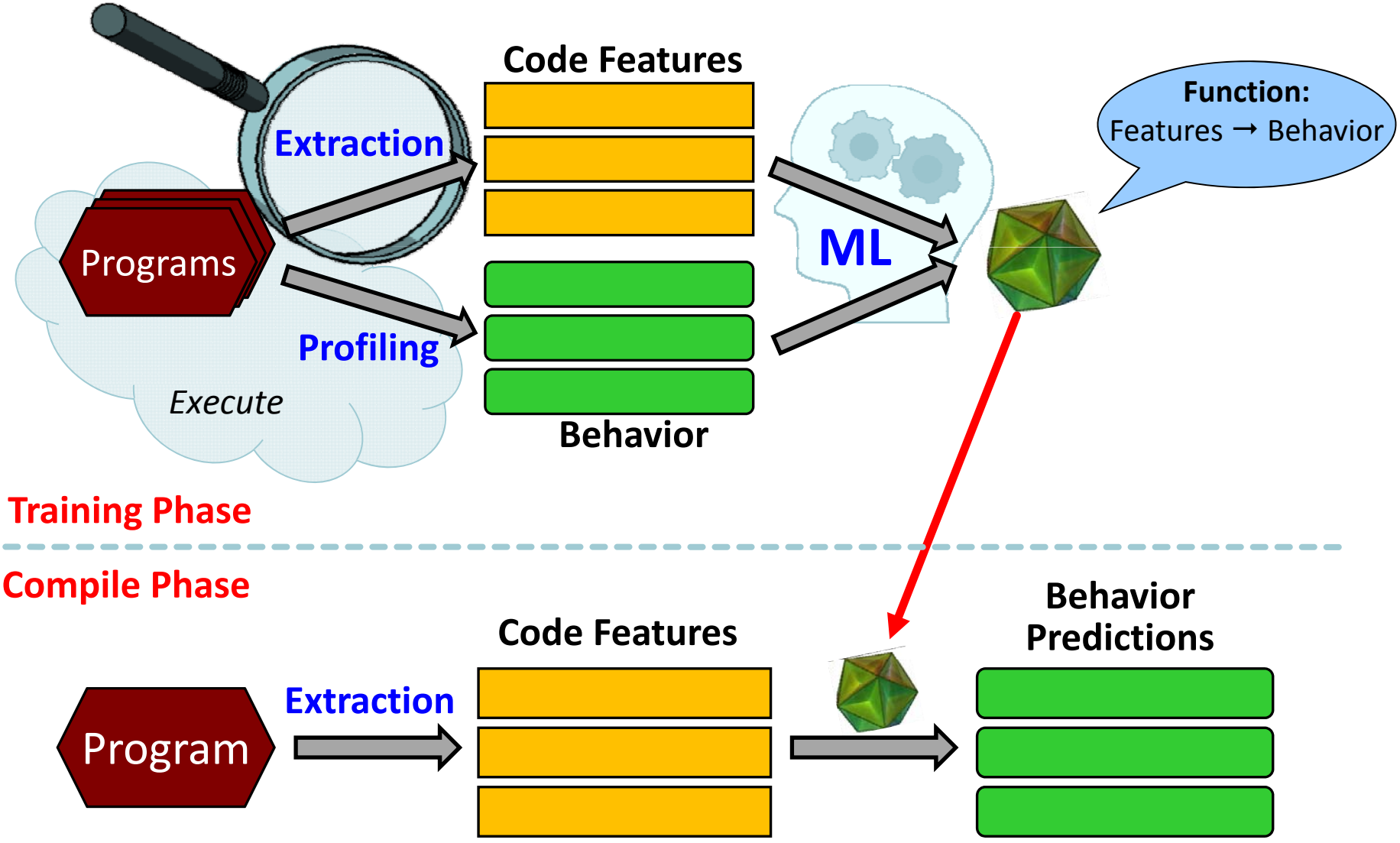
- Optimal solving NP-complete
  - Genetic/Evolutionary Algorithms: many iterations [Y09],[YH09]
  - Common heuristics: do not fit to special MPSoCs
  - ILP-modeled: computational complex [YH08],[VM03]
- Requires information about runtime behavior
  - Static analyses: over-approximate
  - Profiling: strongly input data dependent and expensive

➡ Use **Machine Learning** (ML)

Outline

- ML-based Compilation
- Intelligent Task Mapping
  - Learning
  - Task Graph Mapping
  - Experiments
  - Results
- Conclusions

ML-based Compilation



Outline

- ML-based Compilation
- Intelligent Task Mapping
  - Learning
  - Task Graph Mapping
  - Experiments
  - Results
- Conclusions

Intelligent Task Mapping

➡ Use **Machine Learning** (ML)

- ✔ provides compiler with knowledge of runtime behavior
- ✔ fast and precise heuristics

1) Learn **unknown loop bounds**

   ⇝ Reduce communication overhead

2) Learn **execution times of tasks**

   ⇝ Reduce power consumption

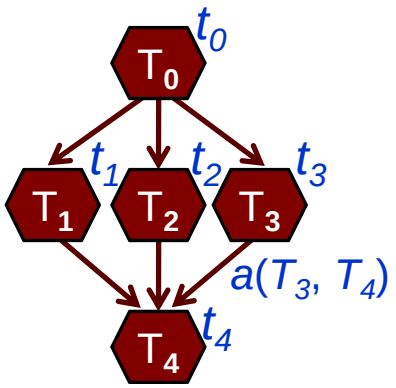
3) Learn **best performing Processing Element** (*PE*)

   ⇝ Treat heterogeneous MPSoCs

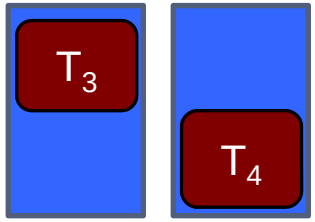
Code Features

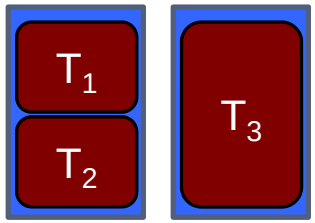
- Unknown loop bounds
  - Structure of loop bounds, number of loop exit branches, size of referenced arrays, file-IO
- Execution times of tasks
  - Latency of the most probable path, fraction of control instructions, loop nesting, amount of interprocessor communication
- Best performing PE
  - depends on architectural differences
    - Caches ⇝ e.g. sizes of loop bodies
    - Functional units ⇝ e.g. fraction of corresponding operations

Task Graph Mapping

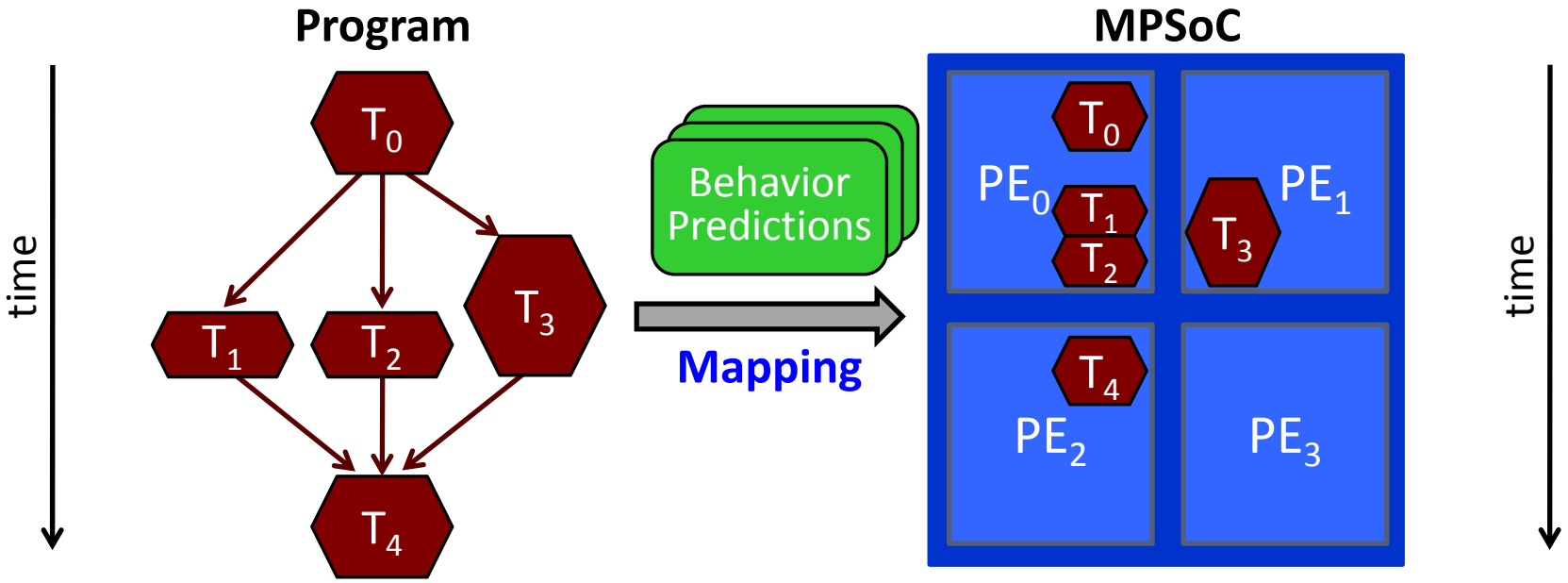


- Execution time t_i for task T_i
- Interprocessor communication
  - Amount $a(T_i, T_j)$
  - Cost $c(T_i, T_j)$
- Runtime $r(T_i, T_j)$
  - sequential: $t_i + t_j + a(T_i, T_j) * c(T_i, T_j)$
  - parallel on different PEs: $\max(t_i, t_j) + a(T_i, T_j) * c(T_i, T_j)$
  - parallel on same PE: $t_i + t_j$
- Latency-weighted list scheduling
- Map tasks to PEs with minimum penalty



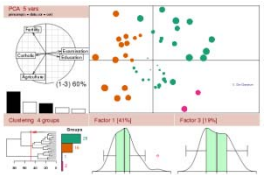


Intelligent Task Mapping



- Benefits
  - ✔ Communication-aware
  - ✔ Power-efficient

Implementation

- Compiler framework: *CoSy*  
  - Feature extraction
  - Static branch prediction
  - Path profiling
- Machine Learning: *R Project*  
  - Predictor construction
    - *supervised classification learning*
  - Program classification[AG09]
    - *hierarchical clustering* to minimize inner-cluster error

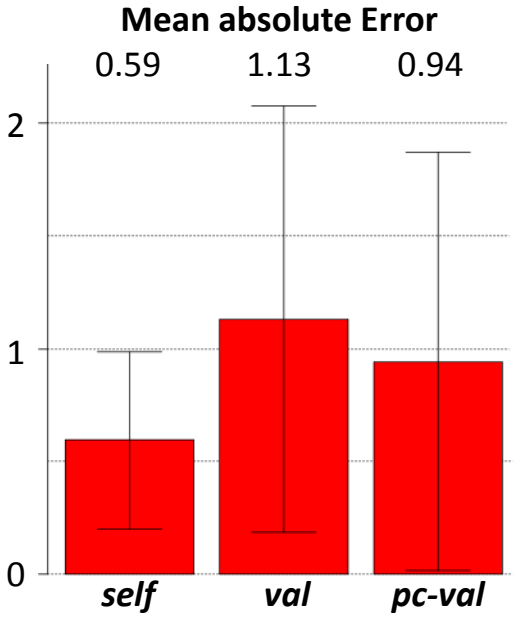
Experiments

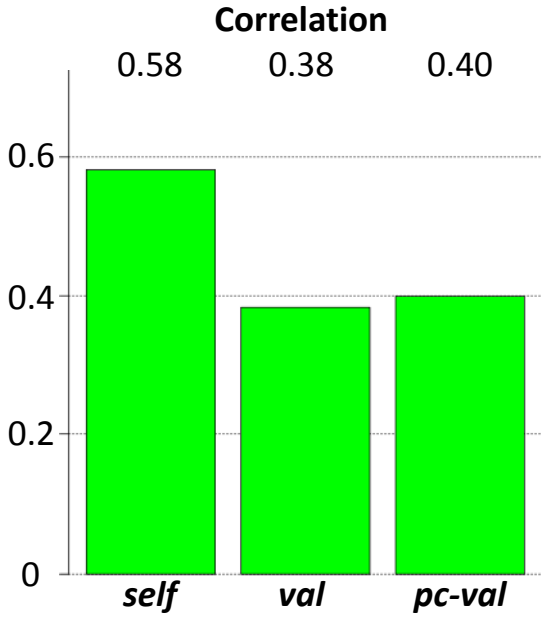
- Learning of unknown loop bounds
  - 66 programs from Ptrdist[A95], MiBench[GR01], SPEC CPU{95,2000,2006} benchmark suites
  - 7970 loops analysed
  - 1 – 98 million iteration counts
  - 115 loop features
  - Loop iterations classified using truncated $\log_{10}$
    - 0 .. 9 ⇝ class 1
    - 10 .. 99 ⇝ class 2

    ...
    - 10 million .. 99.999.999 ⇝ class 8

Experimental Results

- Self evaluation (*self*)
- Validation without program classification (*val*)
- Validation with program classification (*pc-val*)





Outline

- ML-based Compilation
- Intelligent Task Mapping
  - Learning
  - Task Graph Mapping
  - Experiments
  - Results
- Conclusions

Conclusions

- Compilation with knowledge of runtime behavior via ML
  - Unknown loop bounds
  - Execution times of tasks
  - Best performing PE
- Intelligent task mapping
  - ✔ Communication-aware
  - ✔ Power-efficient
- Experimental results
  - ✔ Precise prediction of runtime behavior (error < 1 class)

References

[YH08] H. Yang and S. Ha, "*ILP based data parallel multi-task mapping/scheduling technique for MPSoC*", ISOCC'08

[VM03] G. Varatkar and R. Marculescu, "*Communication-aware task scheduling and voltage selection for total systems energy minimization*", ICCAD'03

[Y09] M. Yoo, "*Real-time task scheduling by multiobjective genetic algorithm*", Journal. of System a. Software, 2009

[YH09] H. Yang and S. Ha, "*Pipelined data parallel task mapping/scheduling technique for MPSoC*", DATE'09

[AG09] L. Alvincz, S. Glesner, "*Breaking the curse of static analysis: making compilers intelligent via Machine Learning*", Proc. of SMART'09, 2009

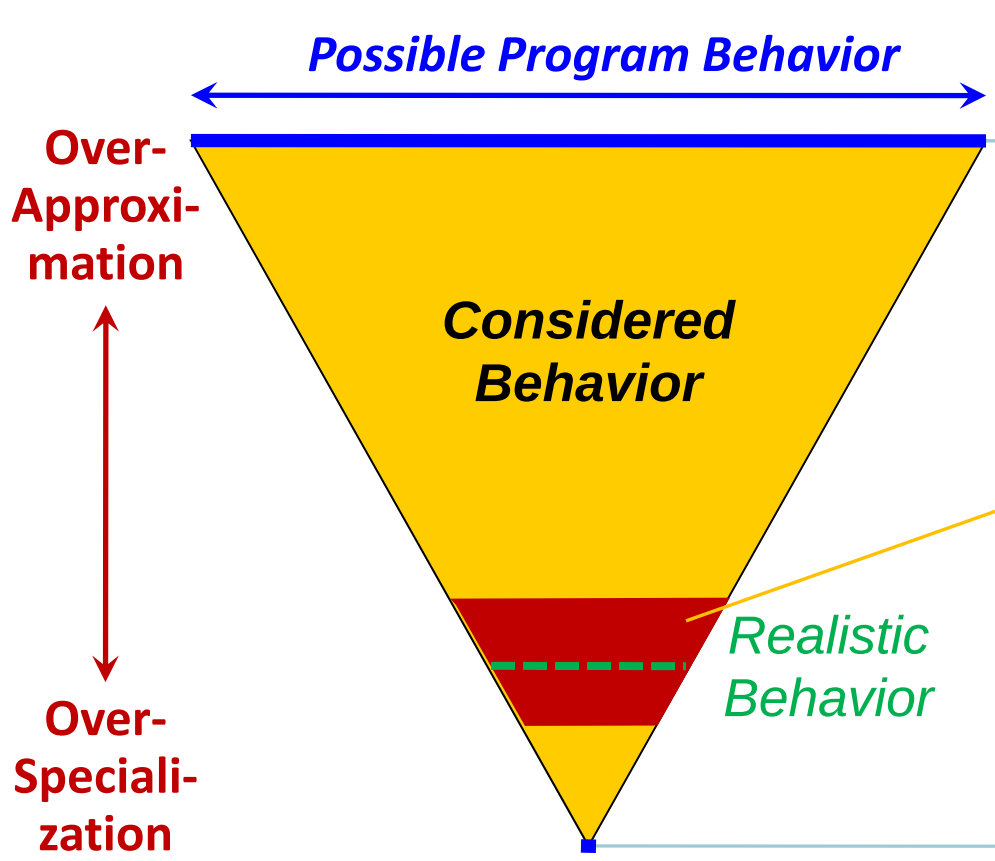
[A95] T. Austin, et al., The pointer-intensive benchmark suite, 1995, http://pages.cs.wisc.edu/~austin/ptr-dist.html.

[GR01] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "*Mibench: A free, commercially representative embedded benchmark suite*", Workshop on Workload Characterization, 2001.

[RRG07] C. Roig, A. Ripoll, and F. Guirado, "*A new task graph model for mapping message passing applications*", IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 12, 2007

Appendix

Estimating the Program Behavior



Program Analysis
- considers all cases
- *safe*, but *imprecise*

Machine-learned Heuristics
- considers *realistic* cases
- *precise*, but *unsafe*

Profiling
- considers only one case
- *(too) precise*, but *unsafe*

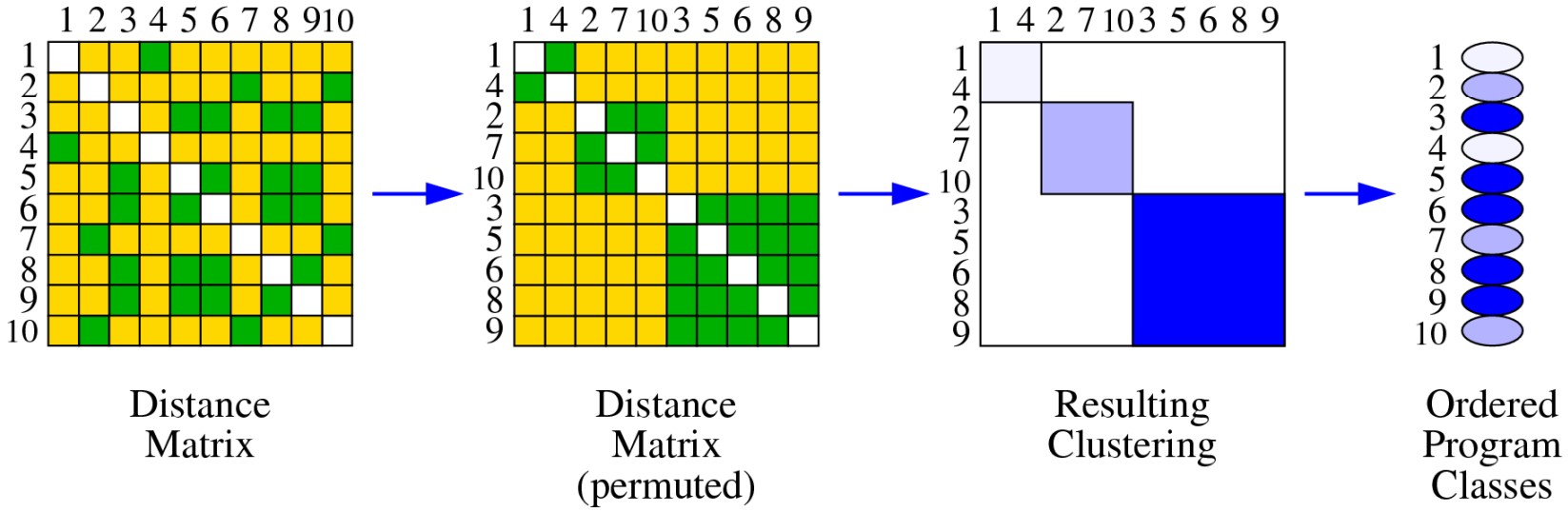
Program Classification[AG09]

- *One* predictor for *all* kinds of programs?
- Better: group *similar* programs, one predictor per group
- ➡ **Program classification** (ML: *unsupervised clustering*)
  - Input: set of programs (from the suite)
    distance measure/distance matrix
  - Output: program classes
- Which programs are *(dis-)similar*?
  - similar programs should be able to *explain* each other's behavior
- ➡ Define similarity based on *mutual predictability*

Program Classification: Clustering[AG09]

- *Mutual predictability:*
  - train one predictor for each program p_i of the suite
  - apply each predictor to every program p_j, compare predicted and correct classes ⇨ *mean deviation error*

➔ **Result: distance matrix**



Combination of Predictors[AG09]

- *n* programs ⇨ *n* training sets ⇨ *n* predictors
- How to obtain *one* predictor?
  - merge *n* training sets D_i to one, train predictor
  - build a composite predictor: consult all predictors and vote
    - take majority vote (if not unique, take min/max); take mean vote