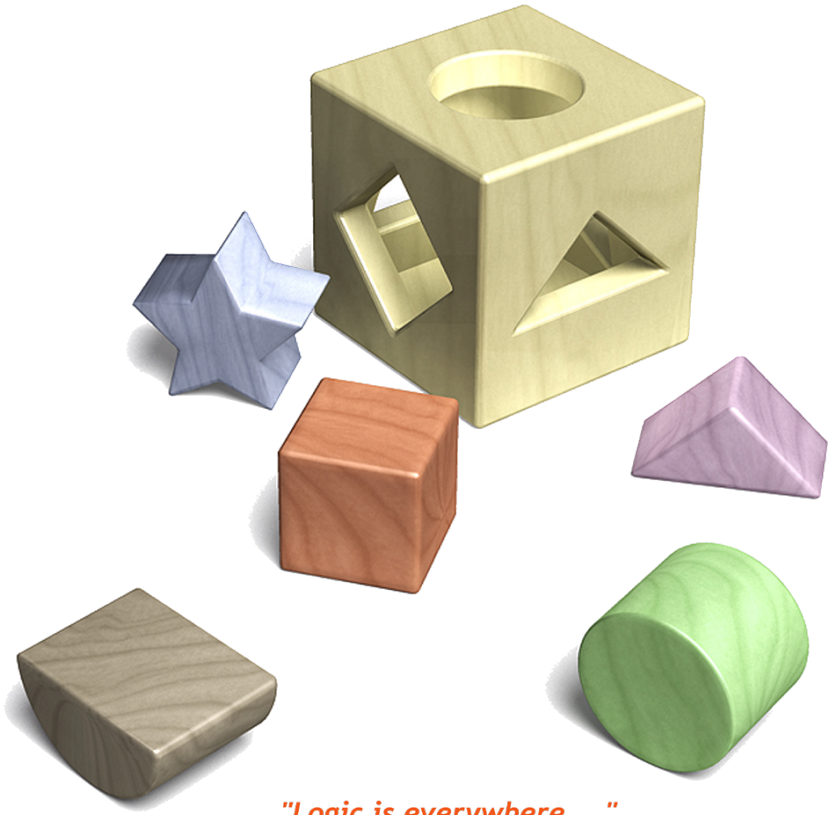
Solving the SAT Problem on Multicore CPUs

Norbert Manthey
International Center for Computational Logic
Technische Universität Dresden
Germany

- Motivation
- Parallel SAT Solving
- Solving SAT in Parallel
- Conclusion

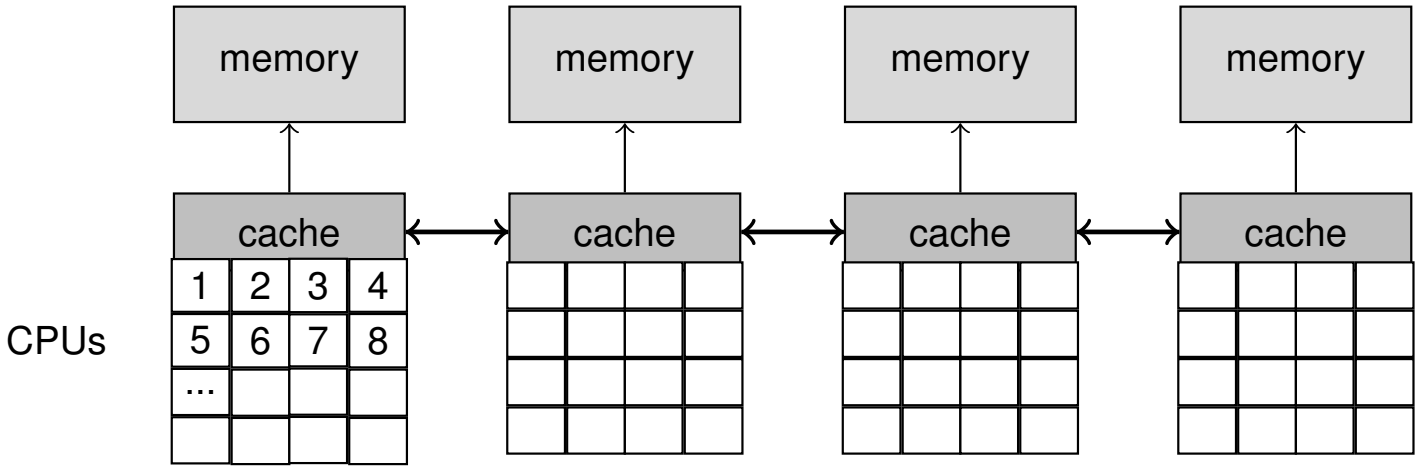


"Logic is everywhere ..."

Motivation

Why should we think about parallel SAT approaches?

Modern Architecture



- **Todays architecture offers four 16-core CPUs on a single board**
- **The memory bandwidth is limited**
- **Single thread performance decreased**
- **The reason: more energy efficient than having 16 CPUs**

Parallel SAT algorithms try to use more cores, to overcome the sequential slowdown.

Parallel SAT Solving

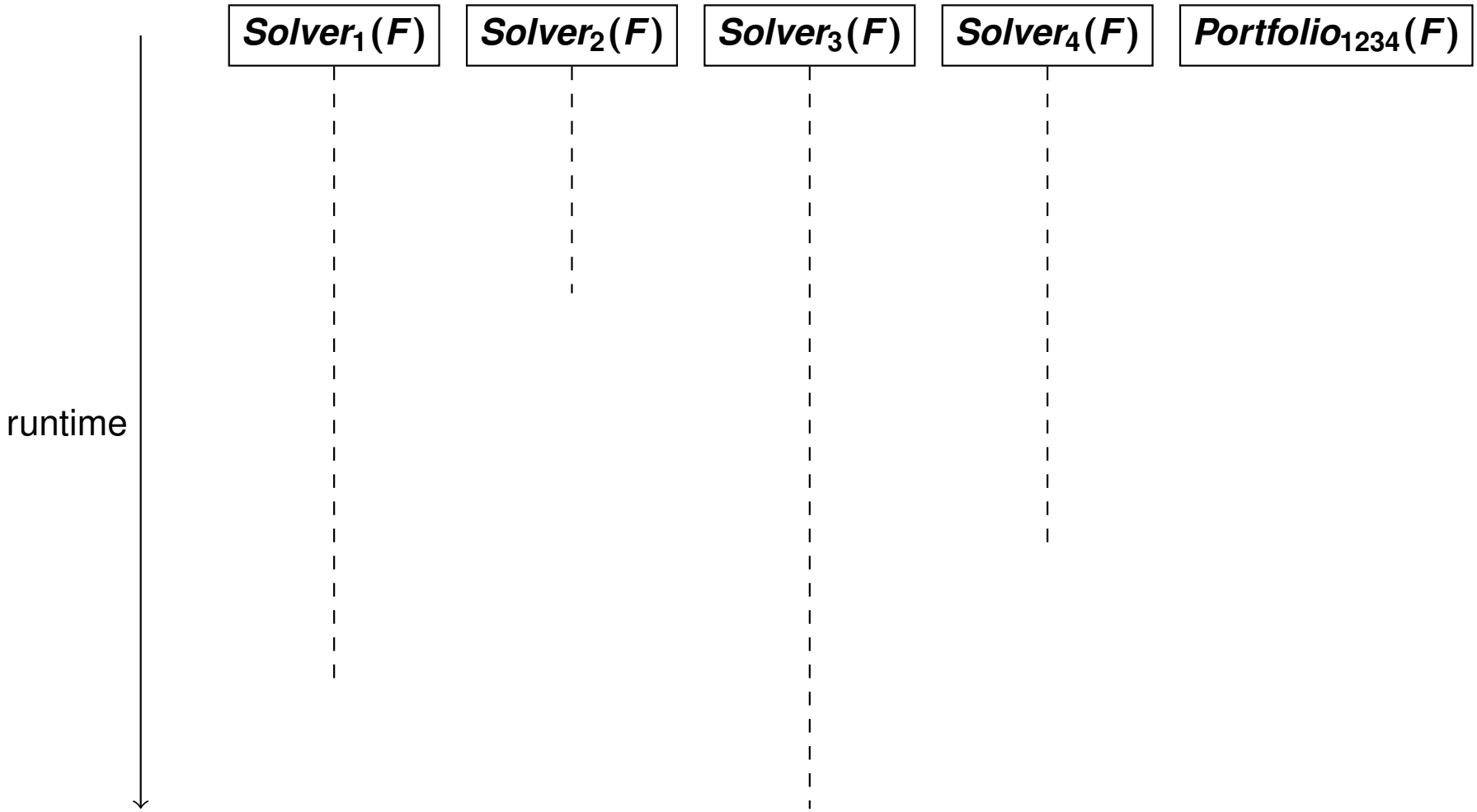
Low Level Parallelization

High Level Parallelization

High Level Parallelization

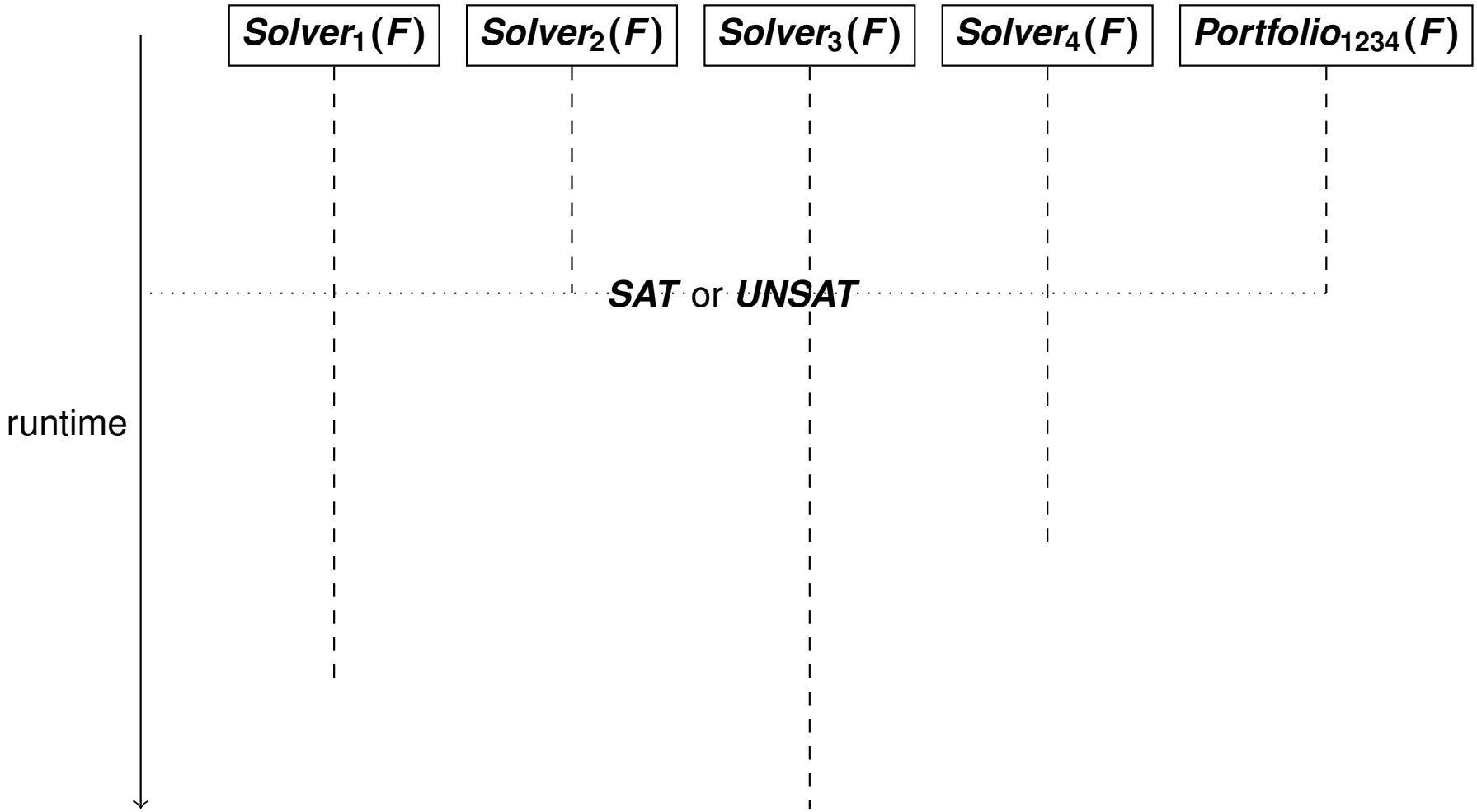
- **Competitive search**, where sub tasks compete and the fastest returns the solution
  - **Portfolio** systems: combine different (specialized) solvers
- **Cooperative search**, where the solution is constructed by combining the solution of sub tasks
  - **Partitioning** systems: partition the search space and solve each sub space with a sequential SAT solver
- **Hybrid**, where sub tasks both compete, but also cooperate
  - **Iterative Partitioning** systems: partition the search space, also dynamically partition the sub spaces, and solve **each** search space with a sequential SAT solver
  - **Guiding Path** systems: dynamically partitioning the search space of a running solver, and closing sub spaces once one solver found its solution
- Sequential performance of each solver is reduced by sharing resources [ABK+13]

Solving SAT in parallel with the Portfolio approach



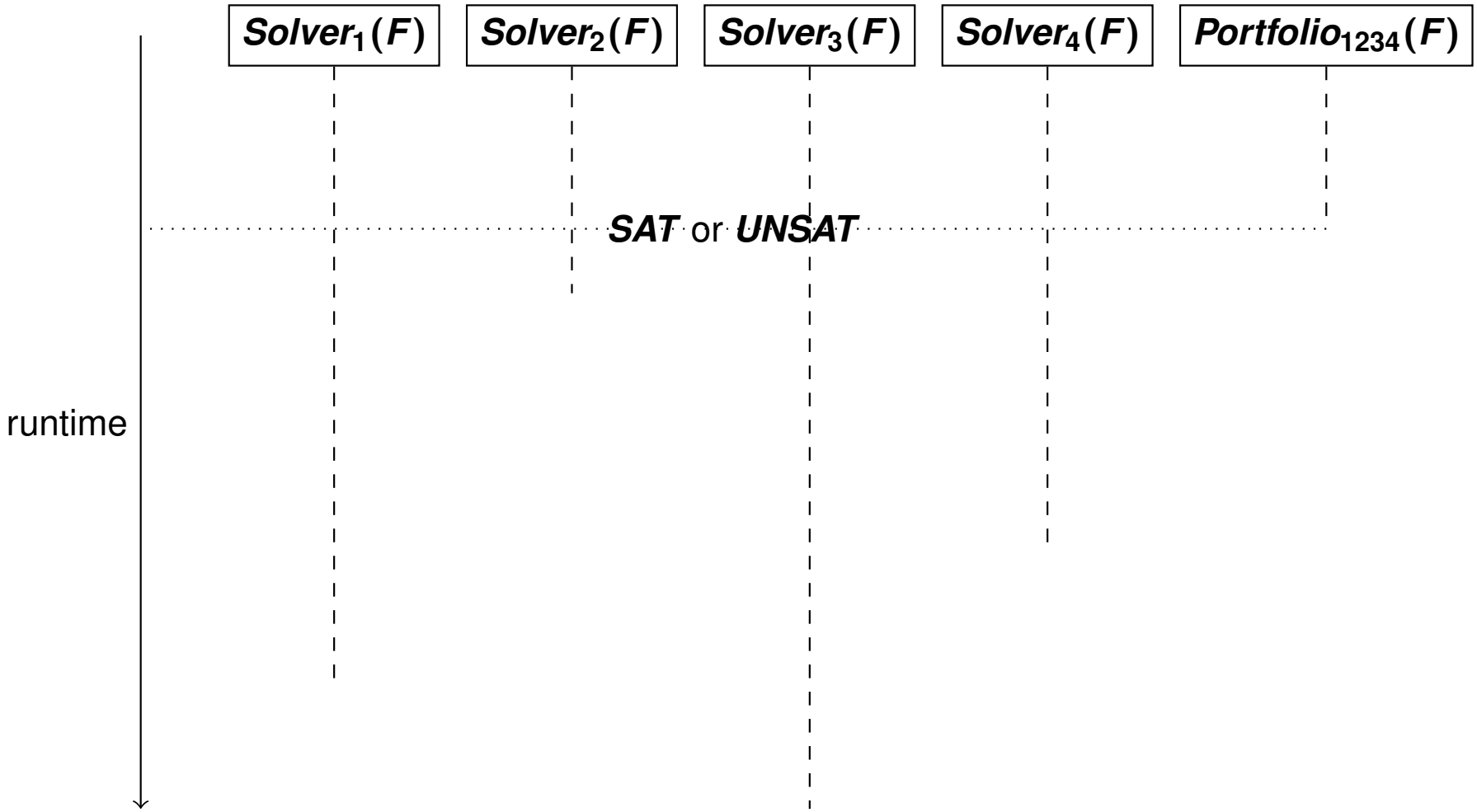
- **Different SAT solver (configurations) compete on the same formula *F***

Solving SAT in parallel with the Portfolio approach



- **The portfolio of these solvers requires the shortest run time**

Solving SAT in parallel with the Portfolio approach



- **By adding communication among the solvers, the performance can be improved**

Solving SAT in parallel with the Portfolio approach

- Properties of the portfolio approach
  - All solvers solve the same formula
  - The solution is returned as soon as one solver finds them
  - This approach improves robustness

- Clause Sharing
  - Clauses can be received, that are entailed by the own formula
  - Decide during sending already: send only clauses that depend on “safe” simplifications (that are entailed by the original formula) [MPW13]

- Limitations of this approach
  - The input formula is not simplified (everybody starts with the same formula)
  - Scales with the number and diversity of solvers

Search Space Partitioning

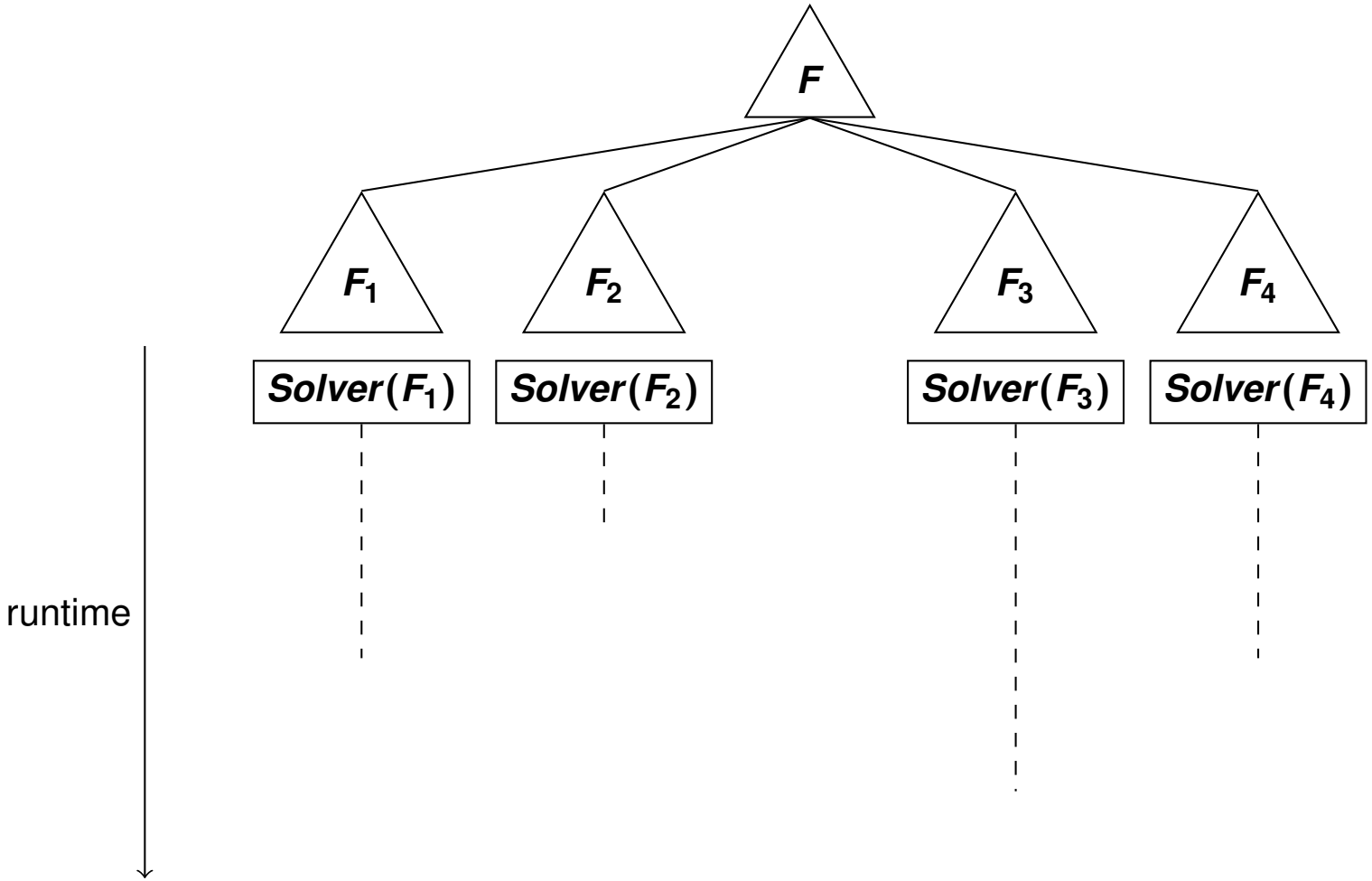
- Partition search space of formula F into sub spaces:
- Create r sub-formulas $F_i := F \wedge K_i$, $1 \leq i \leq r$
- For example:
  - $F_1 = F \wedge a$
  - $F_2 = F \wedge \neg a$
- Or, known as scattering (or tabu-scattering):
  - $F_1 = F \wedge (a \wedge b)$
  - $F_2 = F \wedge ((\neg a \vee \neg b) \wedge c)$
  - $F_3 = F \wedge ((\neg a \vee \neg b) \wedge \neg c \wedge d)$
  - $F_4 = F \wedge ((\neg a \vee \neg b) \wedge \neg c \wedge \neg d)$
- Hence, not only unit clause are added to the formula as partition constraint ("branch on formulas")
- Where do the literals come from? Look-ahead heuristics.

Search Space Partitioning

- Partition search space of formula F into sub spaces:
  - For some $r > 1$ create r "child"-formulas F_i, $0 < i \leq r$, such that
    - their disjunction is equal to the initial formula $F \equiv \bigvee F_i$
    - a partition constraint K_i in CNF is added, $F_i := F \wedge K_i$
  - Note, $F \models K_i$, or $F \equiv_{\text{SAT}} F \wedge K_i$ is not ensured.
  - To obtain a partitioning, additionally ensure
    - that the child-formulas represent disjoint search spaces

      $F_i \wedge F_j \equiv \bot$, for all $0 \leq i < j \leq r$.
  - Solve each child-formula with a sequential solver (including simplification)
  - If a solver proofed unsatisfiability of a sub formula
    - assign a new child formula
  - Conclude unsatisfiability by testing unsatisfiability of child formulas

(Plain) Search Space Partitioning

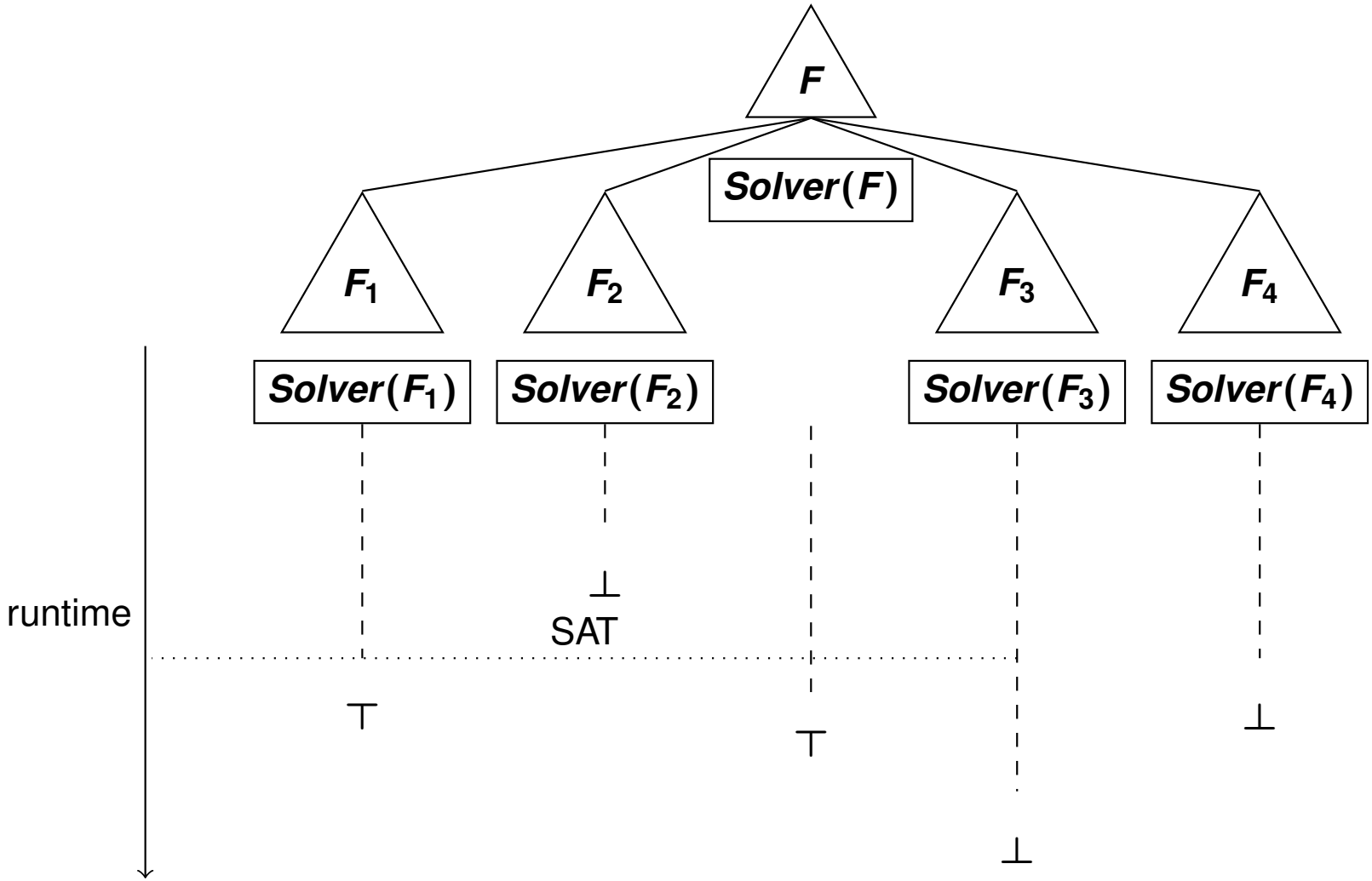


- **Due to heuristics and other side effects, not ensured:**
  $max(t_{Solver}(F_1), t_{Solver}(F_2), t_{Solver}(F_3), t_{Solver}(F_4)) \leq (t_{Solver}(F))$

Iterative Search Space Partitioning

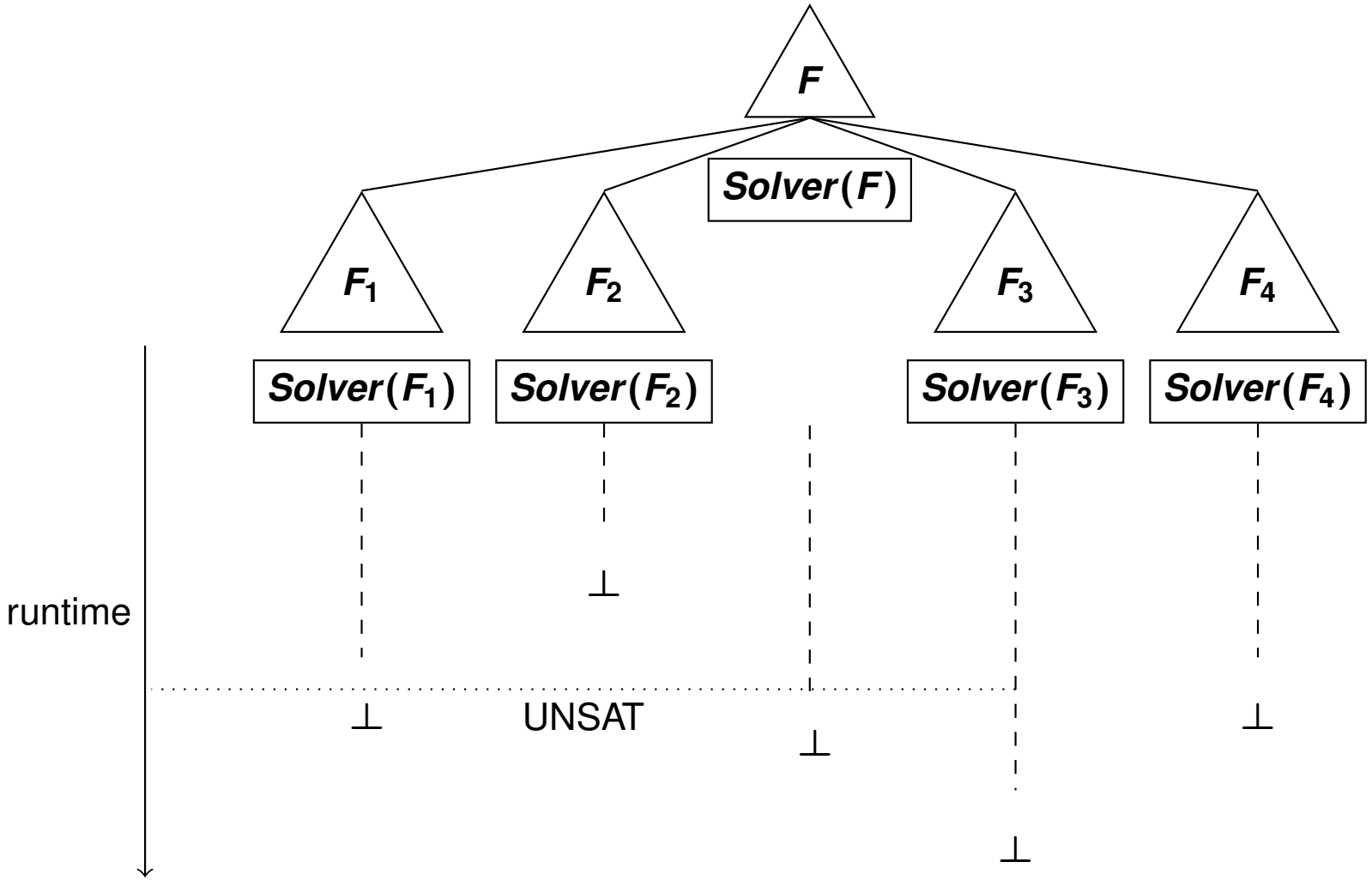
- Partition search space of formula F into sub spaces:
  - Solve all formulas with a sequential solver
  - If a solver proofed unsatisfiability of a sub formula, assign a new child formula
    - assign a yet unsolved child formula
    - or by iteratively applying the whole procedure to child formulas
  - Also partitions child formulas
  - Creates a breadth first search in the search space

Iterative Search Space Partitioning



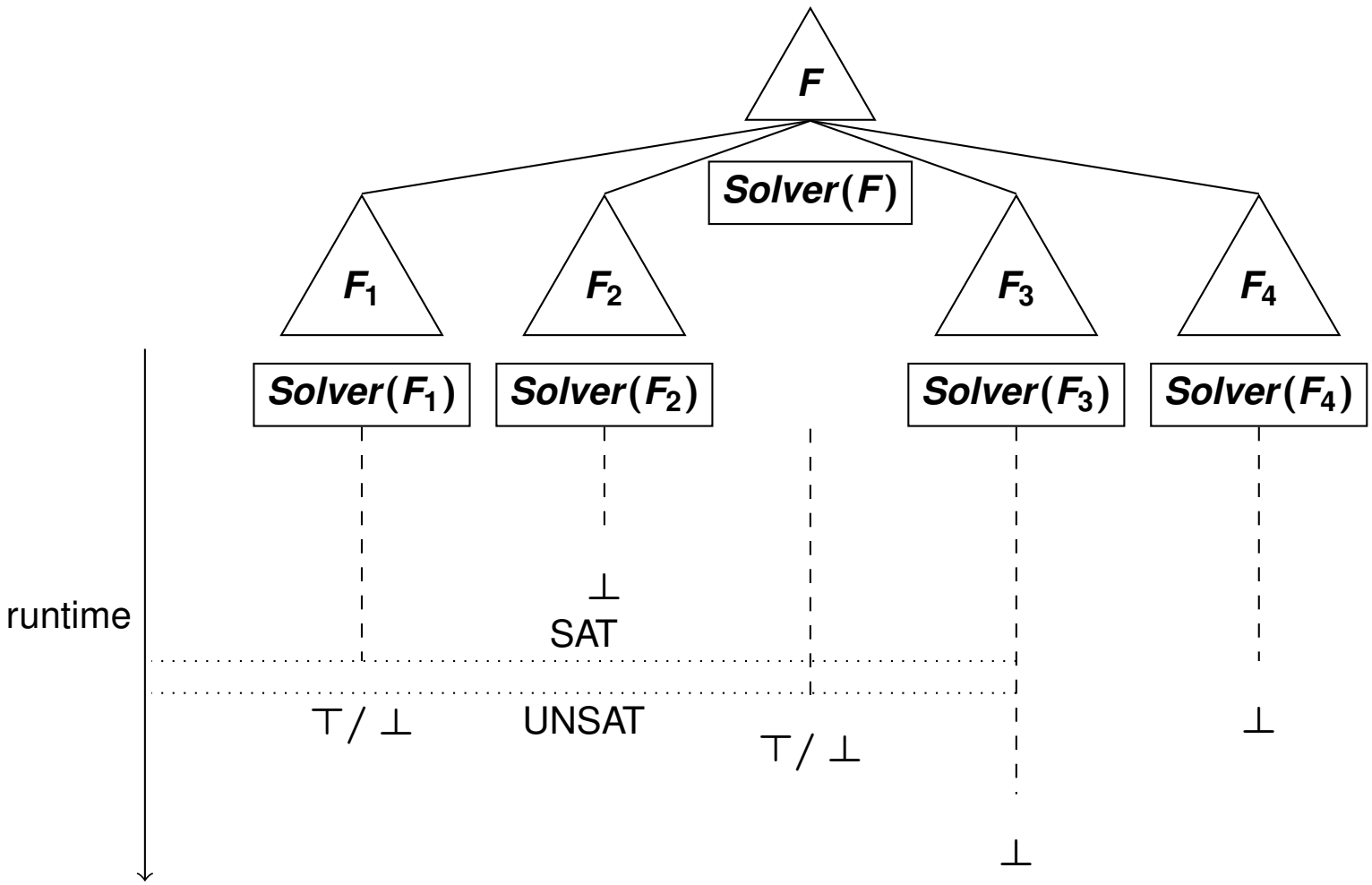
- **finds models as fast as the fastest solver**

Iterative Search Space Partitioning



- **proofs unsatisfiability as fast as the slowest "necessary" solver**

Iterative Search Space Partitioning



- **by iteratively partitioning the search space, new child formulas become more constrained**

Clause Sharing in Iterative Search Space Partitioning

- Each child formula F_{ij} is of the form $F_i \wedge K_j$
  - **Downward** communication:
    - Any clause C that is entailed by F_i can be send to F_{ij} [HM12]
    - Again, simplification needs to be taken into account [Phi13]
  - **Upward** communication:
    - Any clause C that is entailed by F_i, but created in F_{ij}, can still be sent to F_i
    - Take partition constraint K_j into account
    - Track clauses during resolution and simplification
    - Absolute tracking is expensive [HJN10]
    - Use approximation by labelling each clause with its level in the tree [LM13]

Summary of Solving Approaches

- **Portfolio Approach**
  - **Improve robustness by using different solvers**
  - **Conceptually simple, distribute the same formula**
  - **Clauses can be shared depending on simplification techniques [MPW13]**
  - **Scales with the diversity of the SAT solvers or configurations**
- **Plain Partitioning**
- **Iterative Partitioning**

Summary of Solving Approaches

- **Portfolio Approach**
  - **Clauses can be shared depending on simplification techniques [MPW13]**
  - **Scales with the diversity of the SAT solvers or configurations**
- **Plain Partitioning**
  - **Try to improve performance by constraining child formulas**
  - **Partition the formula with partition constraints, solve child nodes**
  - **Clauses can be shared downwards depending on simplification [HM12]**
  - **Clauses can be shared upwards depending on partition constraints and simplification [HJN10, LM13, Phi13]**
  - **Scales with the number of children**
- **Iterative Partitioning**

Summary of Solving Approaches

- Portfolio Approach
  - Clauses can be shared depending on simplification techniques [MPW13]
  - Scales with the diversity of the SAT solvers or configurations
- Plain Partitioning
  - Clauses can be shared downwards depending on simplification [HM12]
  - Clauses can be shared upwards depending on partition constraints and simplification [HJN10, LM13, Phi13]
  - Scales with the number of children
- Iterative Partitioning
  - Solves also intermediate nodes to overcome heuristic problems
  - Same sharing rules as plain partitioning
  - Scales better than the other approaches (data on twelve cores) [HM12]
- Promising approach: solver in a portfolio be parallel partitioning

Low Level Parallelization

- From Amdahl's law it follows:
  - let P be the fraction of the program that can be parallelized,
  - the minimum run time is $(1 - P)t_{\text{seq}}$.
- This fact motivates to parallelize the most expensive part of the algorithm

Low Level Parallelization

- From Amdahl's law it follows:
  - let P be the fraction of the program that can be parallelized,
  - the minimum run time is $(1 - P)t_{\text{seq}}$.
- This fact motivates to parallelize the most expensive part of the algorithm

- Unit Propagation uses $\sim 80\,\%$ of the run time
- Conflict Analysis uses $\sim 13\,\%$ of the run time

Low Level Parallelization

- From Amdahl's law it follows:
  - let P be the fraction of the program that can be parallelized,
  - the minimum run time is $(1 - P)t_{\text{seq}}$.
- This fact motivates to parallelize the most expensive part of the algorithm

- Unit Propagation uses $\sim 80\,\%$ of the run time
  - Complexity already low, per analyzed literal $\mathcal{O}(\frac{\text{number of clauses}}{\text{number of variables}})$
  - Performs a depth first search, for example $1 \rightarrow 2$, $2 \rightarrow 3$, ...
  - Depth first search cannot be parallelized in the worst case (graph is a chain)
  - Implementation: improves slightly for two cores, degrades for more cores.
- Conflict Analysis uses $\sim 13\,\%$ of the run time

Low Level Parallelization

- From Amdahl's law it follows:
  - let P be the fraction of the program that can be parallelized,
  - the minimum run time is $(1 - P)t_{\text{seq}}$.
- This fact motivates to parallelize the most expensive part of the algorithm

- Unit Propagation uses $\sim 80\,\%$ of the run time
- Conflict Analysis uses $\sim 13\,\%$ of the run time
  - Creating the resolvent is also based on traversing the graph
  - Open problem: Is learning a first UIP clause p-complete?
  - Another approach: choose between different learnt clauses (first UIP, bi-asserting, ...)
  - Also open: Select the best clause for the current search state

Low Level Parallelization

- From Amdahl's law it follows:
  - let P be the fraction of the program that can be parallelized,
  - the minimum run time is $(1 - P)t_{\text{seq}}$.
- This fact motivates to parallelize the most expensive part of the algorithm

- Unit Propagation uses $\sim$ 80 % of the run time
- Conflict Analysis uses $\sim$ 13 % of the run time
- Nice parallelization of DPLL is not promising, because
  - Speedup of parallelization is polynomial
  - CDCL can produce exponentially shorter proofs (steps) than DPLL
- Yet, no search space partitioning for CDCL solvers has been found
- Only few work on parallel algorithms for polynomial formula simplification algorithms [GM13]

Conclusion

Implemented Systems

Take Home Message

Implemented Systems and Open Problems

- PPFOLIO: portfolio with simplification
- PLINGELING: portfolio with simplification and communication
- TREENGELING: plain partitioning with simplification and (simple) downward communication
- PCASSO: iterative partitioning with downward and upward communication

- What could be improved:
  - Open: How to select clauses for sharing?
  - Open: How to select partition constraints?
- No parallel solver produces a proof
  - For portfolio solvers, DRAT proofs should be doable based on time stamping
  - Open: How to handle the partition constraints in the other approaches?

Parallel SAT Solving

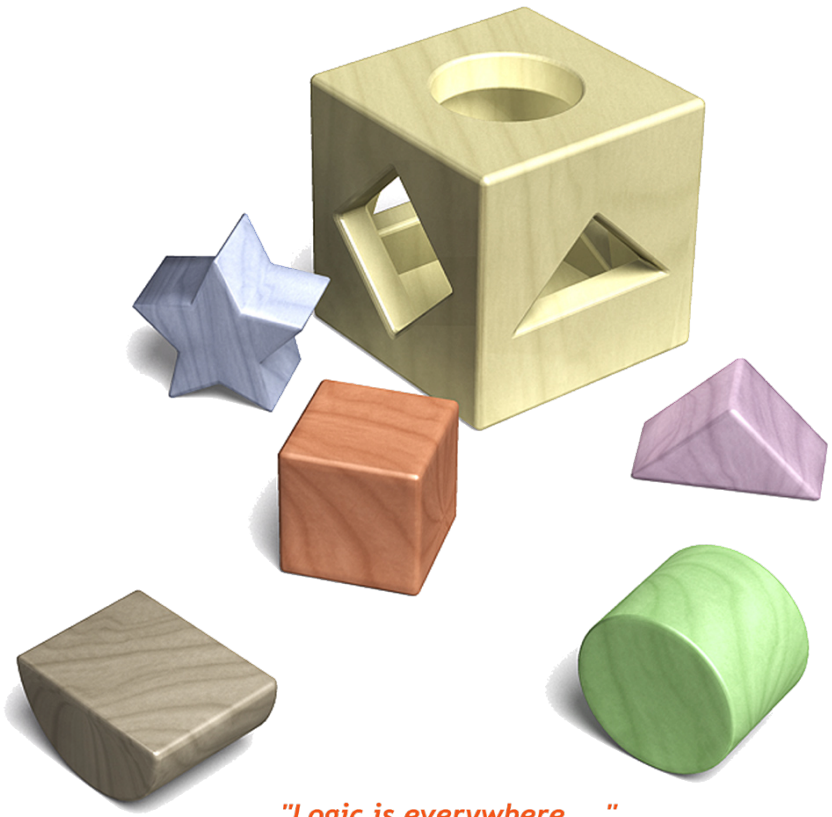
- Today, parallel SAT solvers depend on sequential development
  - Both use the same base CDCL engine
  - Parallel algorithms benefit from improvements to sequential solvers
- Improvements can be achieved by good partitioning and clause sharing
  - Are there proof related measurements for sharing and partitioning
- Current systems cannot be compared to theoretical proof theory
  - So far, no proof format is available to compare length, space, width, ...

- Not discussed here:
  - Current FPGA and GPGPU SAT solutions are not competitive
  - Differences between Many-Core CPUs and CPU clusters

Solving the SAT Problem on Multicore CPUs

Norbert Manthey
International Center for Computational Logic
Technische Universität Dresden
Germany

Thank you for your attention



"Logic is everywhere ..."

Related Work (incomplete)

- Surveys on parallel SAT Solving: [MML12, HMN+11]
- Partitioning
  - Comparisons of partitioning and portfolio [HM12], improve partitioning [ILM13]
  - Clause Sharing [HJN10, LM13]
  - Soundness of simplification and sharing [Phi13]
  - Systems: TREENELING, PCASSO, DOLIUS, ...
- Portfolio
  - Soundness of simplification and sharing [MPW13]
  - Scalability vs. resource usage [ABK+13]
  - Systems: MANYSAT, PLINGELING, PPFOLIO, ...
- Low Level
  - Parallel Unit Propagation [HW]
  - Parallel Preprocessing [GM13]
  - Simplification in parallel to search [WH13]
- Open Challenges [HW13]

References I

[ABK+13] M. Aigner, A. Biere, C. M. Kirsch, A. Niemetz, and M. Preiner.
Analysis of portfolio-style parallel SAT solving on current multi-core architectures.
In *Pragmatics of SAT(POS'13)*, 2013.

[GM13] Kilian Gebhardt and Norbert Manthey.
Parallel variable elimination on cnf formulas.
In Ingo J. Timm and Matthias Thimm, editors, *KI 2013: Advances in Artificial Intelligence*, volume 8077 of *Lecture Notes in Computer Science*, pages 61–73. Springer, 2013.

[HJN10] Antti E. J. Hyvärinen, Tommi Junttila, and Ilkka Niemelä.
Partitioning sat instances for distributed solving.
In *Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning*, LPAR'10, pages 372–386, Berlin, Heidelberg, 2010. Springer-Verlag.

[HM12] Antti E. J. Hyvärinen and Norbert Manthey.
Designing scalable parallel sat solvers.
In *Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing*, SAT'12, pages 214–227, Berlin, Heidelberg, 2012. Springer-Verlag.

[HMN+11] S. Hölldobler, N. Manthey, V.H. Nguyen, J. Stecklina, and P. Steinke.
A short overview on modern parallel SAT-solvers.
In I. Wasito et.al., editor, *Proceedings of the International Conference on Advanced Computer Science and Information Systems*, pages 201–206, 2011.
ISBN 978-979-1421-11-9.

[HW] Antti EJ Hyvärinen and Christoph M Wintersteiger.
Approaches for multi-core propagation in clause learning satisfiability solvers.

[HW13] Youssef Hamadi and Christoph M Wintersteiger.
Seven challenges in parallel sat solving.
AI Magazine, 34(2), 2013.

[ILM13] Ahmed Irfan, Davide Lanti, and Norbert Manthey.
Modern cooperative parallel sat solving.
In *Pragmatics of SAT(POS'13)*, 2013.

References II

[LM13] Davide Lanti and Norbert Manthey.
Sharing information in parallel search with search space partitioning.
In Giuseppe Nicosia and Panos M. Pardalos, editors, *LION*, volume 7997 of *Lecture Notes in Computer Science*, pages 52–58. Springer, 2013.

[MML12] R. Martins, V. Manquinho, and I. Lynce.
An overview of parallel SAT solving.
Constraints, 17(3):304–347, 2012.

[MPW13] Norbert Manthey, Tobias Philipp, and Christoph Wernhard.
Soundness of inprocessing in clause sharing sat solvers.
In M. Järvisalo and A. Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013*, volume 7962 of *LNCS*, pages 22–39. Springer, Berlin Heidelberg, 2013.

[Phi13] Tobias Philipp.
Expressive models for parallel satisfiability solvers.
Master thesis, Technische Universität Dresden, 2013.

[WH13] Siert Wieringa and Keijo Heljanko.
Concurrent clause strengthening.
In Matti Järvisalo and Allen Van Gelder, editors, *SAT*, volume 7962 of *Lecture Notes in Computer Science*, pages 116–132. Springer, 2013.