# ACAI 2015 - Solvers Evaluation

Daniel Le Berre and Olivier Roussel

CRIL-CNRS UMR 8188

ACAI'5@CRIL - October 26, 2015 - Lille

# Objective of this course

- Highlight the benefit of independent evaluation of constraints-based techniques
- Provide some advices and pitfalls to avoid
- Explain how to read the evaluation results (go beyond the ranking!)
- Focus on the technical issues met when running an evaluation on modern hardware and OS

# Objective of this course

- Highlight the benefit of independent evaluation of constraints-based techniques
- Provide some advices and pitfalls to avoid
- Explain how to read the evaluation results (go beyond the ranking!)
- Focus on the technical issues met when running an evaluation on modern hardware and OS

Based on our own experience in organizing the SAT, PB or CSP competitions from 2002 to 2012!

# Objective of this course

- Highlight the benefit of independent evaluation of constraints-based techniques
- Provide some advices and pitfalls to avoid
- Explain how to read the evaluation results (go beyond the ranking!)
- Focus on the technical issues met when running an evaluation on modern hardware and OS

Based on our own experience in organizing the SAT, PB or CSP competitions from 2002 to 2012!
And our experience as participants (Sat4j, ppfolio)

# Outline of the talk

Motivation and conditions of success

Selecting Benchmarks

Ranking solvers

Understanding the results of the evaluation

# Outline

Motivation and conditions of success

Selecting Benchmarks

Ranking solvers

Understanding the results of the evaluation

# Evaluations: motivation

- Worst-case complexity vs reality: NP-complete does not mean intractable in practice
- Allow to observe specific algorithms implementations on specific benchmarks on specific hardware: produce data for analysis
- Identify which technique/approach is suitable for which benchmarks
- Foster the development of new algorithms
- Expand the application domain: gather new benchmarks

# Evaluations: motivation

- Worst-case complexity vs reality: NP-complete does not mean intractable in practice
- Allow to observe specific algorithms implementations on specific benchmarks on specific hardware: produce data for analysis
- Identify which technique/approach is suitable for which benchmarks
- Foster the development of new algorithms
- Expand the application domain: gather new benchmarks

Rankings and winners are a byproduct of the evaluations, not goals to achieve!

# Conditions of success

- Common input/output format for the benchmarks
- Critical mass of solvers
- Critical mass of diverse benchmarks
- Low entry level for participants

There should be publicly available benchmarks and solvers

# Common input/output format for the benchmarks

- Simple to understand and parse for the solver designer
- Simple to understand and generate by the benchmarks provider
- Readable by a human (text based)
- Main issues:
  - not oriented toward the end user
  - not space efficient

# Example: Dimacs format

## Do you understand this?

```
p cnf 4 7
1 2 3 4 0
-1 -2 0
-1 -3 0
-1 -4 0
-2 -3 0
-2 -4 0
-3 -4 0
```

# Example: Dimacs format

## Do you understand this?

```
p cnf 4 7
1 2 3 4 0
-1 -2 0
-1 -3 0
-1 -4 0
-2 -3 0
-2 -4 0
-3 -4 0
```

Encode $x_1 + x_2 + x_3 + x_4 = 1$

# Counter-Example: Mancoosi CUDF format

## Does this one look better?

```
package: supersolver
version: 1
depends: minisat

package: minisat
version: 1
conflicts: minisat

package: minisat
version: 2
conflicts: minisat

package: glucose
version: 2
provides: minisat
```

# Benefit of a common input format: program reuse

- for SAT, the second Dimacs challenge (1993) created a common input format
- since then, SAT solvers have been used as black boxes reading Dimacs formatted files
- without black boxes, no Planning as Satisfiability, no Bounded Model Checking, no Chaff ...

Requires also the availability of the solvers for research purposes, ideally in source form: without Grasp, SATO, Relsat available in source, no Chaff!

# Critical mass of solvers

- Need enough participants to simply check the results
- 4/5 research groups working on a specific topic are enough to start an evaluation (MUS/Group-MUS 2011)
- A basic/trivial approach can be used as a witness

Example: SAT,PB,MAXSAT,ASP,SMT,...
Counter-Examples: SAT 2005 non clausal track, SAT 2007 certified unsat track,

# Critical mass of benchmarks

- Variety of benchmarks is key for a good evaluation
- Sources of benchmarks are the biggest bias for an objective evaluation
- Random/Crafted/Application buckets
- New versus known benchmarks (do you allow Machine Learning?)
- Benchmarks coming from companies (e.g. IBM BMC benchmarks)

The more diverse are the benchmarks, the less significant is a global ranking in practice!

# Low entry level for participants

- The SAT competition has been a big success because a master student could write a good SAT solver 10 years ago (e.g. Siege)
- Building a CSP or an Automated Reasoning solver (CASC) is much more evolved
- Recent SAT solvers include many sophisticated techniques (pre/in-processing): raised entry level to at least a PhD
- How to lower the entry level? Minisat Hack (since 2009)!

# Checking solvers answers

Never trust the solvers!

- The answers of the solvers should be checked if possible
  - SAT check that the assignment satisfies all the constraints
  - UNSAT check that no other solver answered correctly SAT
- Sometimes it is quite hard to check answers (e.g. for QBF)
- Over the years, practical solutions appear (DRUP format and checker for logging and check UNSAT proofs in SAT solvers)

# "Incorrect" solvers

There are many reasons why a solver may be found "incorrect" during an evaluation:

- The solver contains at least one bug
- The solver enters a corner case: may be avoided using benchmarks normalization
- The solver does not interpret correctly the input (parsing problem)
- The solver does not interpret correctly the semantic of the input
- There is a problem with the evaluation platform

An incorrect answer does not make automatically the solver buggy!

# Decision vs Optimization

- On decision problems, the solver answers SAT/UNSAT/UNKNOWN
- On decision problems, the solver answers OPT/SAT/UNSAT/UNKNOWN
  - SAT check that the assignment satisfies all the constraints
  - OPT check that no other solver answered correctly a better SAT answer
- The quality of the solution can be taken into account
- Difference between finding the best solution and proving there is no better solution

# Outline

Motivation and conditions of success

Selecting Benchmarks

Ranking solvers

Understanding the results of the evaluation

# Benchmarks categories

| | |
|---|---|
| random | Randomly generated, obey a mathematical model (e.g. Random Uniform 3-SAT) |
| crafted | "Born to be hard" represent known [in]tractable classes of benchmarks (e.g. Pigeon Hole Problem) |
| application | represent modeling of real/artificial problems |

With the usual feature that
$size(crafted) < size(random) << size(application)$ and
$runtime(crafted) > runtime(random) >> runtime(application)$

# Benchmarks hardness

Benchmarks are used to discriminate solvers

- Easy benchmarks (solved by all participants) are of no help*
- Hard benchmarks (solved by no participant) are of no help
- Only the remaining benchmarks (let's call them Medium) are of interest
- Easy/Medium/Hard classification depends on the solvers!

** may be used to quickly check the answers of the participating solvers*

# State Of The Art (SOTA) (CASC definition)

- The state of the art represents the set of all the benchmarks solved by at least one solver [in a given timeout]
- Improving the SOTA is thus to solve new benchmarks [in a given timeout]
- Improvement can come from faster solvers
- Improvement can come from orthogonal approaches
- Improvement cannot come from portfolio-solvers, i.e. combination of existing solvers

# Benchmarks selection

- Classify benchmarks per categories (origin)
- Benchmarks hardness determined by a selection of existing solvers
- Randomly pick a selection of benchmarks respecting a given ratio of easy/medium/hard benchmarks
  - easy needed to check that those problems are still easy for new techniques
  - medium needed to discriminate solvers
  - hard benchmarks needed to get a chance to improve the state of the art

# Outline

Motivation and conditions of success

Selecting Benchmarks

Ranking solvers

Understanding the results of the evaluation

# Aim of the ranking

- Order the solvers according to one or several performance criteria
- Spot interesting solvers
- Provide a winner

# Desirable properties

- Should be easy to check
- The score obtained by a solver should not depend on other participating solvers
- Should be able to exhibit "interesting" solvers

# Basic ranking scheme

- Rank the solvers according to the total number of problems solved
- Break ties with CPU-time

Simple lexicographic ranking based on two criteria used for CASC, SAT,...

# Purse-based ranking scheme (SAT COMP. 2005)

**Benchmark purse** to be divided equally among the solvers able to solve it.

**Speed purse** to be divided unequally among the solvers able to solve a given benchmark.

**Series** an extra credit is given for each series solved.

**Solver** his score is the sum of the credits obtained per benchmark solved plus the credits obtained per series solved.

# Purse based ranking: observations

- Easy benchmarks (solved by all solvers) do not provide much credit
- Benchmarks solved by a single solve provide a lot of credit
- Speed purse rewards fast solvers
- Series credit rewards wide scope/multipurpose solvers

Very difficult to check the results

# Penalty-based ranking scheme

(Penalized Average Runtime used in Machine Learning systems)

- Rank the solvers according to CPU-time
- Take a default CPU-time for unsolved benchmarks ($x$ times the timeout)

(PAR10 = take 10 times the timeout as penalty)

# Comparison of those rankings

(see SATSPEC2009 web page for details)

Application benchmarks

| Solver | Solved | CPU time | PAR10 | PAR100 | Purse |
|---|---|---|---|---|---|
| Rsat | **106** | **18043** | **1554043** | **15378043** | **44623** |
| TiniSatELite | 103 | 22193 | 1594193 | 15742193 | 30301 |
| picosat | 103 | 30610 | 1602610 | 15750610 | 25509 |
| minisat | 97 | 20931 | 1664931 | 16460931 | 29714 |

Crafted benchmarks

| Solver | Solved | CPU time | PAR10 | PAR100 | Purse |
|---|---|---|---|---|---|
| minisat | **71** | **13105** | **1573105** | **15613105** | 24601 |
| SATzilla | 69 | 12281 | 1596281 | 15852281 | 30336 |
| MiraXT | 57 | 13929 | 1741929 | 17293929 | 14737 |
| TTS | 37 | 3347 | 1971347 | 19683347 | **36457** |

# Outline

Motivation and conditions of success

Selecting Benchmarks

Ranking solvers

Understanding the results of the evaluation

# Poor man results of the SAT competition

## SAT 2014 competition

| | |
|---|---|
| Organizing committee | Anton Belov, Daniel Diepold, Marijn Heule, Matti Järvisalo |
| Judges | Pete Manolios, Lakhdar Sais and Peter Stuckey |
| Proceedings | Descriptions of the solvers and benchmarks |
| Benchmarks | Application, Hard combinatorial, Random |
| Solvers | Source code available in EDACC |

| | Application | | | Hard combinatorial | | | Random | | |
|---|---|---|---|---|---|---|---|---|---|
| | Gold | Silver | Bronze | Gold | Silver | Bronze | Gold | Silver | Bronze |
| **Core solvers** | | | | | | | | | |
| SAT+UNSAT | Lingeling | SWDiA5BY | Riss BlackBox | glueSplit_clasp | Lingeling | SparrowToRiss | | | |
| SAT | minisat_blbd | Riss BlackBox | SWDiA5BY | SparrowToRiss | CCAnr+glucose | SGSeq | Dimetheus | BalancedZ | CSCCSat2014 |
| Certified UNSAT | Lingeling (druplig) | glucose | SWDiA5BY | Riss BlackBox | Lingeling (druplig) | glucose | | | |
| **Core solvers, Parallel** | | | | | | | | | |
| SAT+UNSAT | Plingeling | PeneLoPe | Treengeling | Treengeling | Plingeling | pmcSAT 2.0 | | | |
| SAT | | | | | | | pprobSAT | Plingeling | CSCCSat2014 |
| **Minisat hack** | | | | | | | | | |
| SAT+UNSAT | MiniSat_HACK_999ED | minisat_blbd | ROKKminisat | | | | | | |

## SAT 2013 competition

| | |
|---|---|
| Organizing committee | Adrian Balint, Anton Belov, Marijn Heule and Matti Järvisalo |
| Judges | Roberto Sebastiani, Karem A. Sakallah and Youssef Hamadi |
| Proceedings | Descriptions of the solvers and benchmarks |
| Benchmarks | Application, Hard combinatorial, Random |
| Solvers | |

| | Application | | | Hard combinatorial | | | Random | | |
|---|---|---|---|---|---|---|---|---|---|
| | Gold | Silver | Bronze | Gold | Silver | Bronze | Gold | Silver | Bronze |
| **Core solvers** | | | | | | | | | |
| SAT+UNSAT | Lingeling aqw | Lingeling 587f | ZENN 0.1.0 | BreakIDGlucose 1 | gluebit_clasp 1.0 | glucose 2.3 | CSHCrandMC | MIPSat random sat_unsat | march_vfl 1.0 |
| SAT | Lingeling aqw | ZENN 0.1.0 | satUZK 48 | glucose 2.3 | gluebit_clasp 1.0 | BreakIDGlucose 1 | probSAT SC13 | sattime2013 2013 | Ncca+ V 1 |
| Certified UNSAT | glucose 2.3 (certified unsat) | glueminisat-cert-unsat 2.2.7j | Riss3g cert | Riss3g cert | glucose 2.3 (certified unsat) | forl drup-nocachestamp | | | |
| **Core solvers, Parallel** | | | | | | | | | |
| SAT+UNSAT | Plingeling aqw | Treengeling aqw | PeneLoPe 2013 | Treengeling aqw | Plingeling aqw | pmcSAT 1.0 | | | |
| **Minisat hack** | | | | | | | | | |
| SAT+UNSAT | SINNminisat 1.0.0 | minisat_bit 1.0 | MiniGolf prefetch | | | | | | |
| **Open track (multiple solver sources, mixed benchmarks)** | | | | | | | | | |
| | CSHCpar8 | | | MIPSat | | | GlucoRed+March r531 | | |

# Immediate, not obviously visible, results

- Much reliable solvers (e.g. less segmentation fault)
- “More correct" solvers (if bugs are detected they are fixed)
- More reusable solvers (they now agree on the evaluation I/O as well)
- Unique access point to the expected answers for SOTA benchmarks.

# Results you will find on presentations, papers

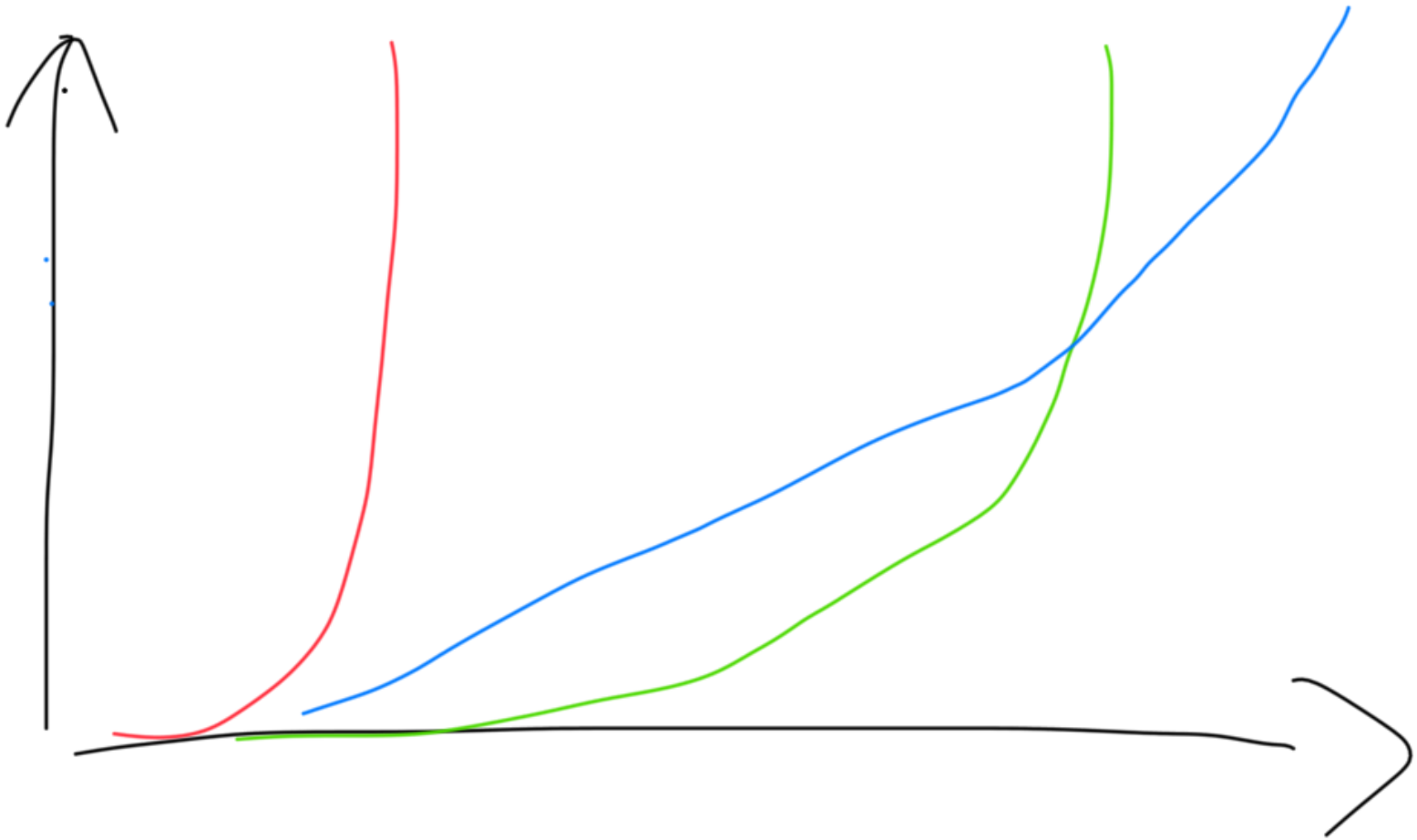
**Cactus plots** visual representation of the distribution of the runtimes of the solvers on solved benchmarks.

- x-axis represents the number of problem solved
- y-axis represents the runtime
- a point represents the number of problems solved within the given runtime

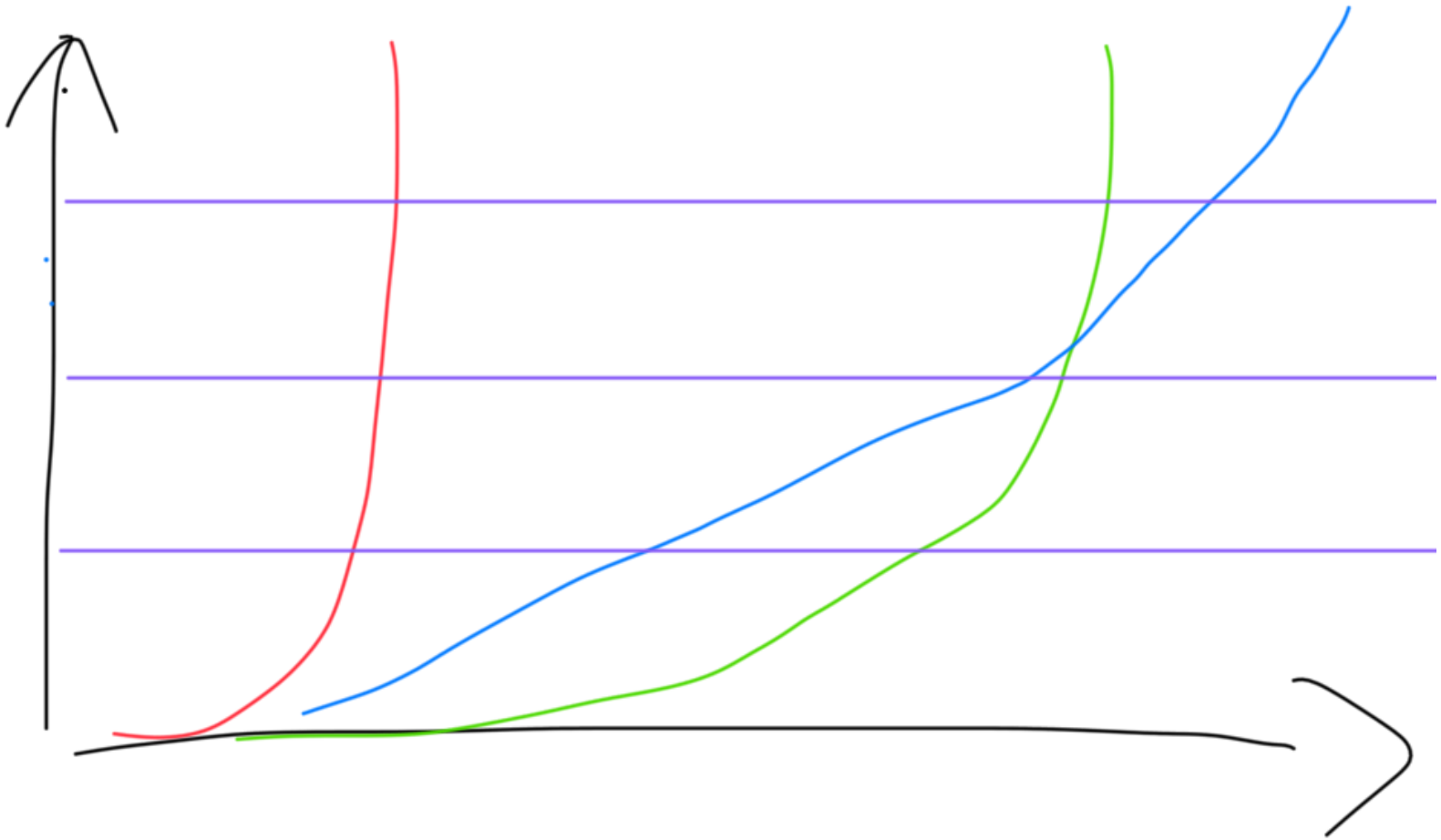
**Scatter plots** Visual comparison of a metric (usually runtime) for two solvers on a given set of benchmarks

- x-axis represents the runtime of solver A
- y-axis represents the runtime of solver B
- a point represents the metrics of solvers A and B for a given benchmark

# How to read a cactus plot



# How to read a cactus plot

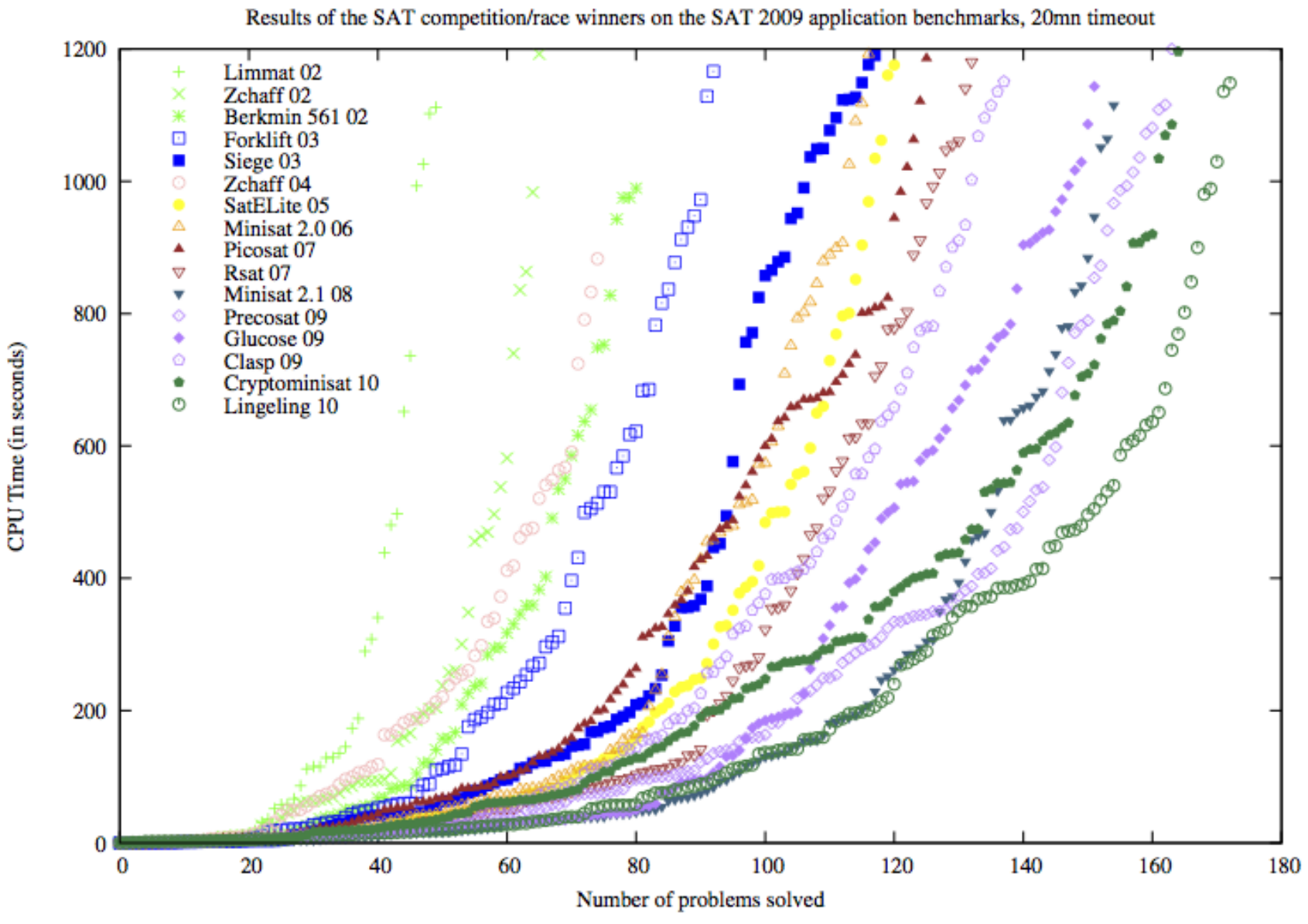


# Example: winners on SAT2009 benchmarks

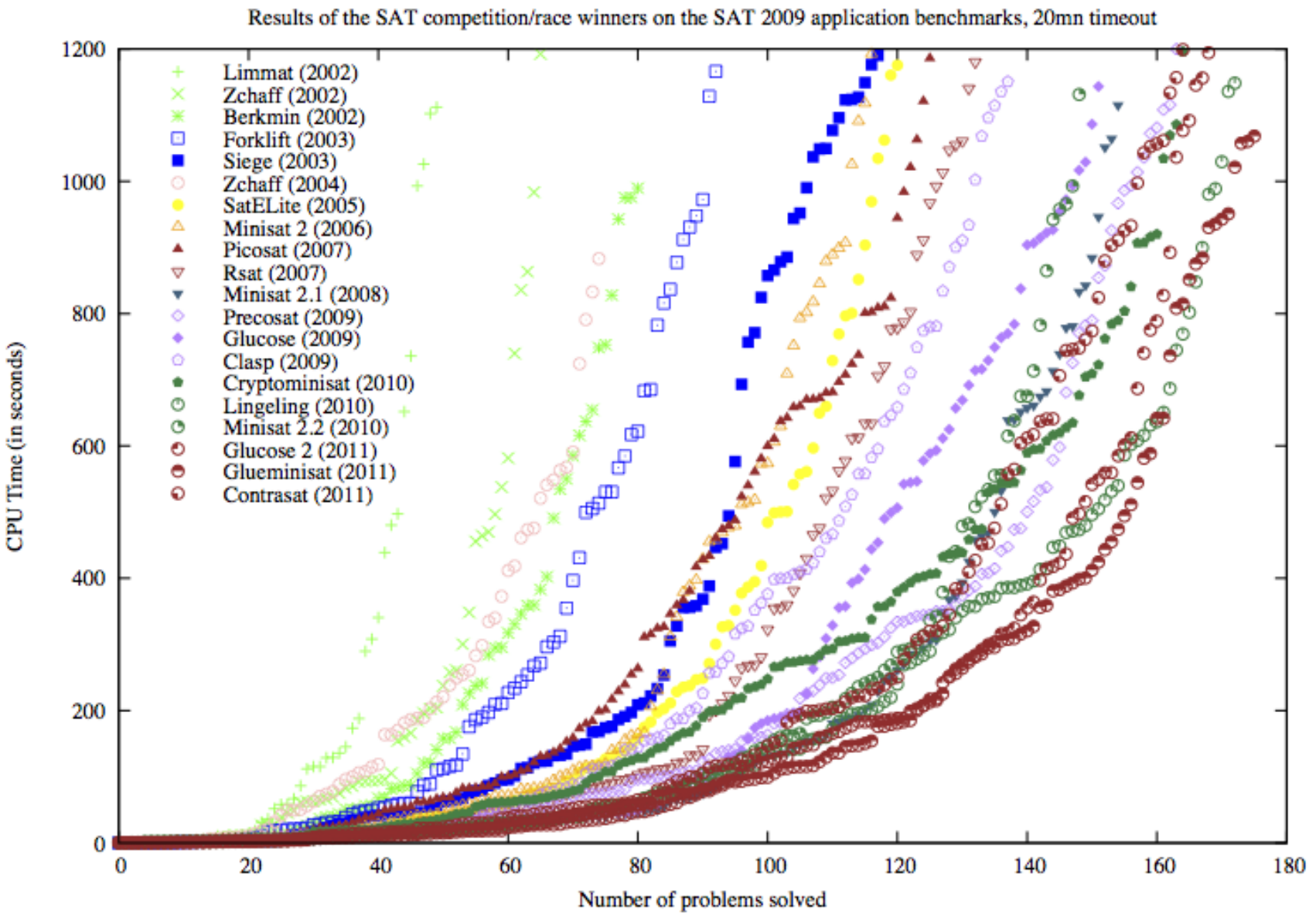
- Take the winners of the SAT competitions since 2002
- Take the benchmarks of the SAT 2009 competition
- Take the hardware of the SAT 2009 competition (2GB of RAM)
- Observe ...

Threat to validity:
winners after 2009 may have been trained on those benchmarks
the hardware does no longer represent a desktop computer

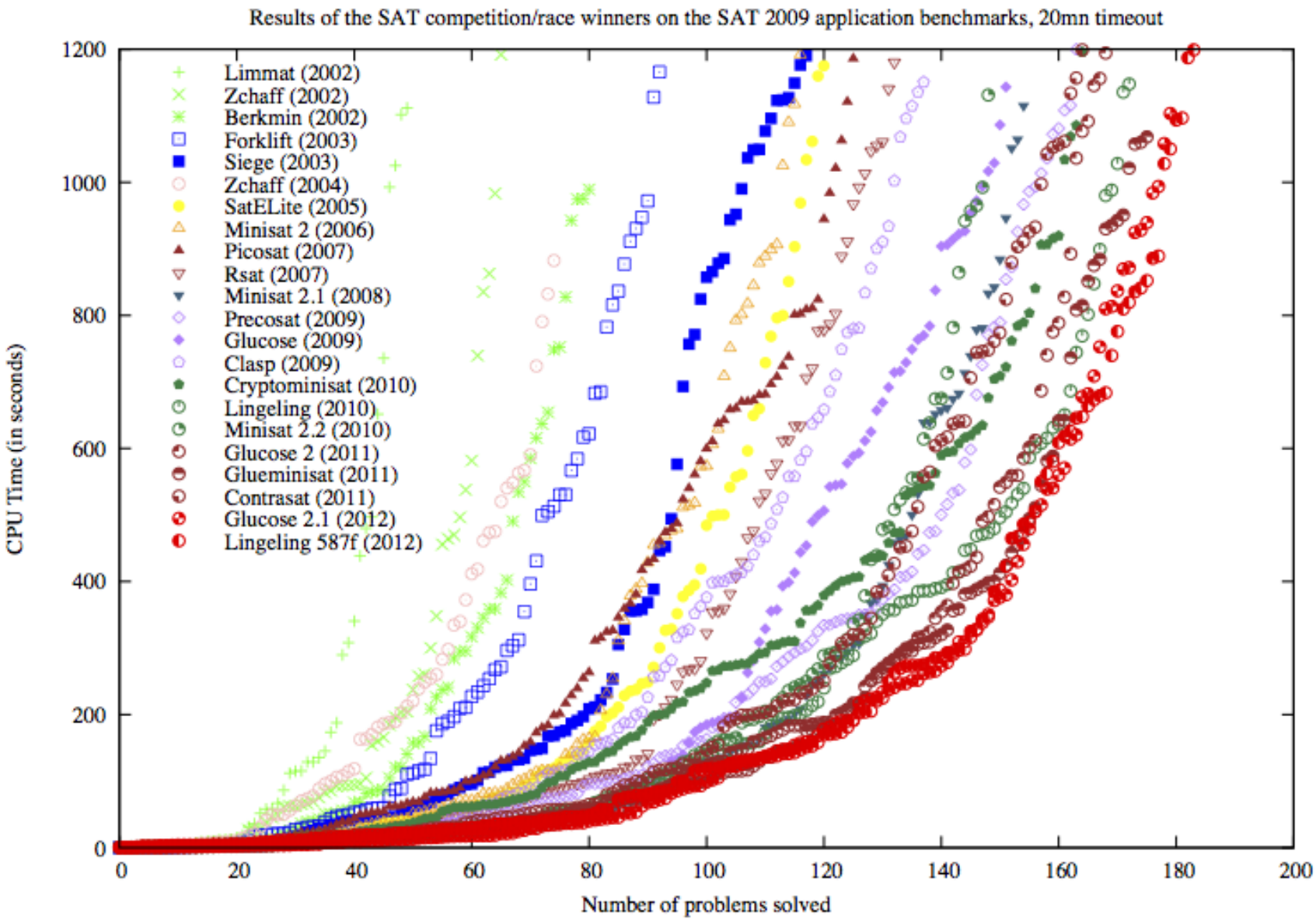
# Example: winners on SAT2009 benchmarks



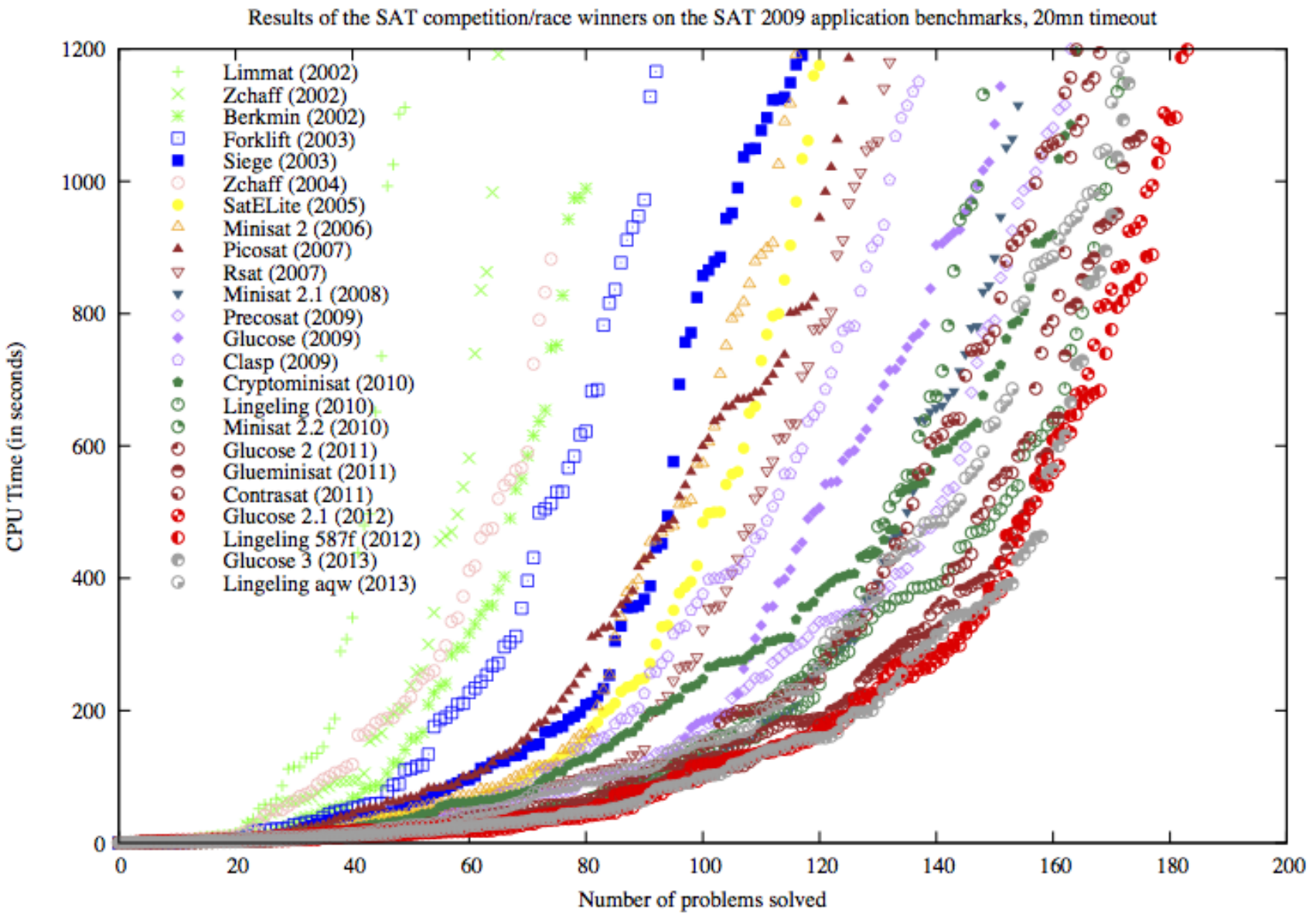
# Example: winners on SAT2009 benchmarks



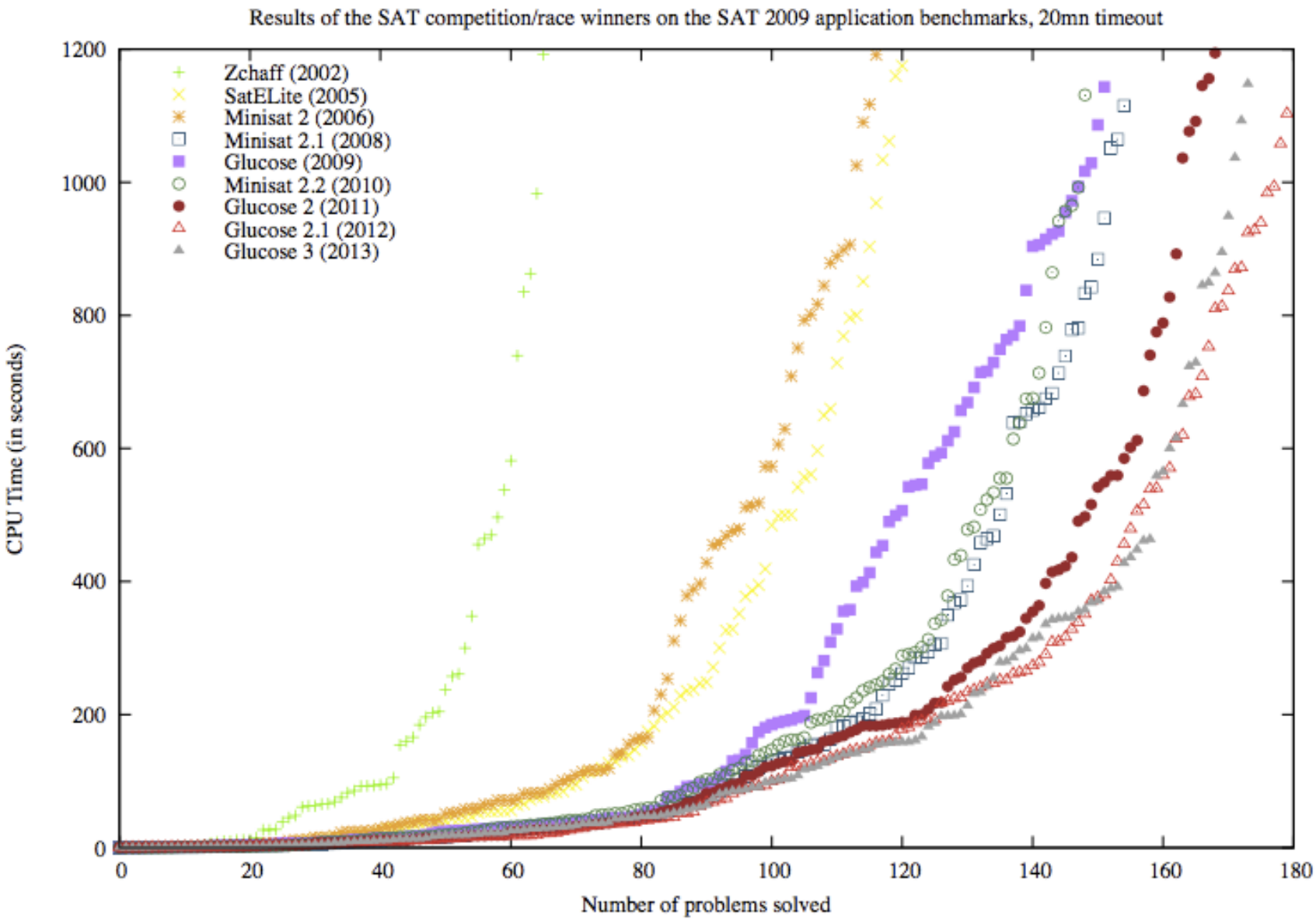
# Example: winners on SAT2009 benchmarks



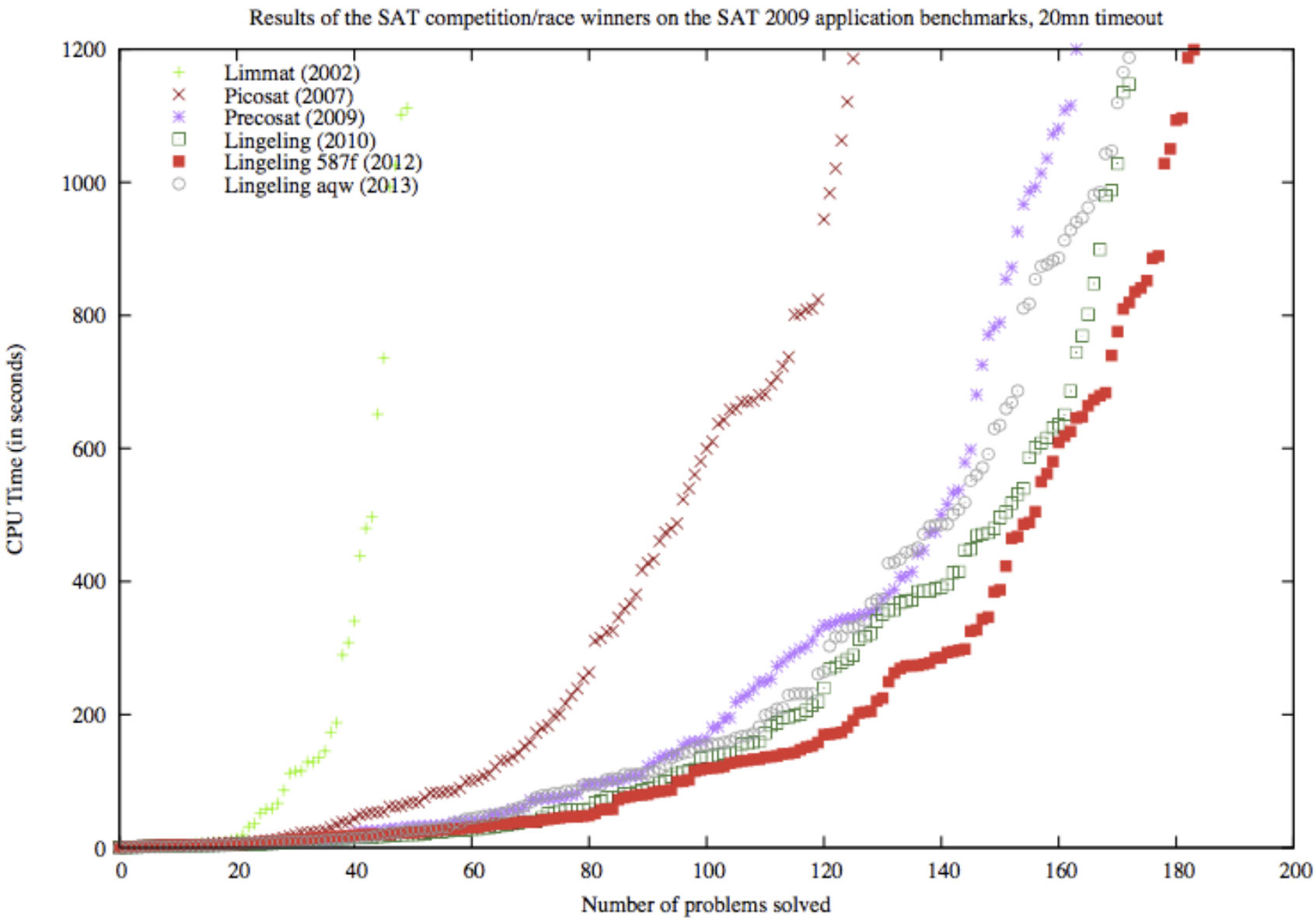
# Example: winners on SAT2009 benchmarks



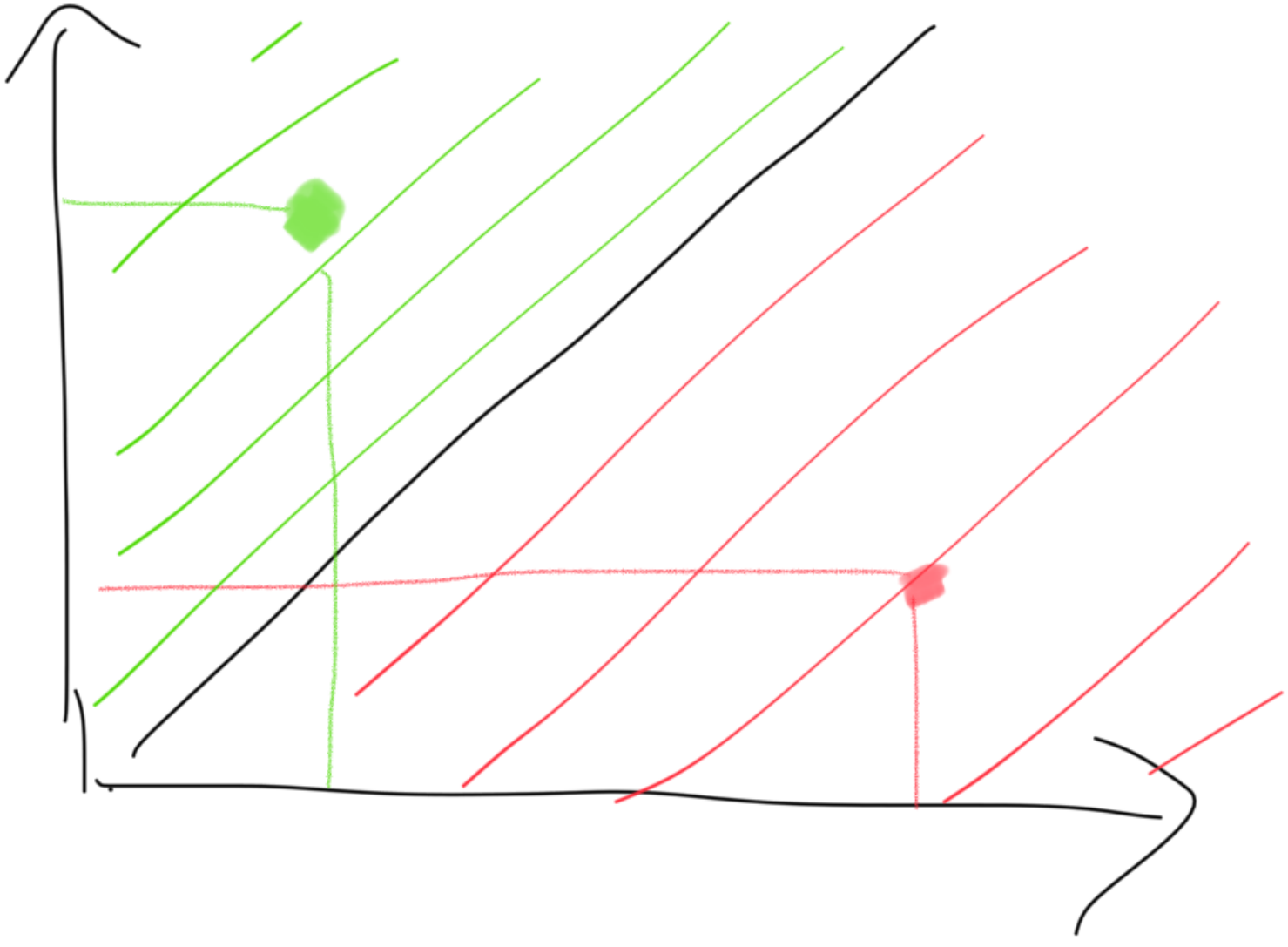
# Glucose


Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout
Zchaff (2002)
SatELite (2005)
Minisat 2 (2006)
Minisat 2.1 (2008)
Glucose (2009)
Minisat 2.2 (2010)
Glucose 2 (2011)
Glucose 2.1 (2012)
Glucose 3 (2013)
CPU Time (in seconds)
Number of problems solved
0
200
400
600
800
1000
1200
0
20
40
60
80
100
120
140
160
180

# Armin's Biere solvers



# How to read a scatter plot



# Example: Armin's Biere POS15 presentation