Reliable Benchmarking: Requirements and Solutions

Dirk Beyer, Stefan Löwe, and Philipp Wendler

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

SoSy-Lab Software Systems

Evaluation of Research Result

- Result “Theorem”; evaluation “Proof”
- Result “Algorithm”; evaluation “Algorithm Analysyis, properties, Big-O”
- Result “Heuristics for Complex Problems”; evaluation “Performance Experiments”

Comparative Evaluation

- Old: Done by competitors
- New: Done by independent competitions

Notions from Experimental Research

Experimental science needs:

Repeatability
: Same team, same experimental setup

Replicability
: Different team, same experimental setup

Reproducibility
: Different team, different experimental setup

Source:
https://www.acm.org/publications/policies/artifact-review-badging

Notions from Experimental Research

Example: You implemented new algorithm in AFL and compared it against KLEE.

Repeatability

You execute same version of AFL again. Are the numbers the same?

Replicability

Somebody else takes same version of AFL and benchmark set and executes it.

Reproducibility

Somebody implements both algorithms in a different tool and compares them.

Notions from Experimental Research

Repeatability
: Can you produce the same results
for the camera-ready version again?

Replicability
: Can others take your tool etc.
and perform the experiment?
(main goal of providing artifacts)

Reproducibility
: Can others come to the same conclusion
in a different experiment?

Background: Wording

experiments	can be replicable
experiments	can be repeatable (weaker than replicable)
effects	can be reproducible
conclusions	can be reproducible
performance results	can be replicable (but better avoid this)

Background: Wording

experiments can be replicable

experiments can be repeatable (weaker than replicable)

effects can be reproducible

conclusions can be reproducible

performance results can be replicable (but better avoid this)

measurements can be accurate and precise

benchmarking can be reliable

runs are executed

Background: Wording

experiments can be replicable

experiments can be repeatable (weaker than replicable)

effects can be reproducible

conclusions can be reproducible

performance results can be replicable (but better avoid this)

measurements can be accurate and precise

benchmarking can be reliable

runs are executed

We avoid

- benchmark
- to run

Background: Requirements

Repeatability

- everything documented
  (machine, version of tool and OS, parameters)
- deterministic tool
- **reliable benchmarking**

Replicability

- everything above
- availability of tool, benchmark set,
  configuration, environment
  (published and archived, appropriate license)

Reproducibility

(not discussed here)

Benchmarking is Important

- Evaluation of new approaches
- Evaluation of tools
- Competitions
- Tool development (testing, optimizations)

Reliable, replicable, and accurate results needed!

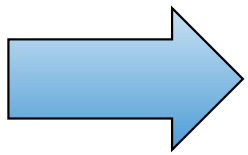
Benchmarking is Hard

- Influence of I/O
- Networking
- Distributed tools
- User input

Benchmarking is Hard

- Influence of I/O
- Networking
- Distributed tools
- User input

Not relevant for
most verification tools



Easy?

Benchmarking is Hard

- Influence of I/O
- Networking
- Distributed tools
- User input

Not relevant for most verification tools

- Different hardware architectures
- Heterogeneity of tools
- Parallel benchmarks

Relevant!

Goals

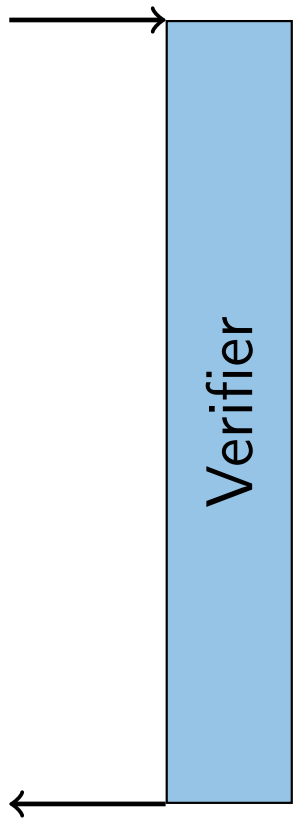
- Replicability
  - Avoid non-deterministic effects and interferences
  - Provide defined set of resources
- Accurate results
- For verification tools (and similar)
- On Linux

Checklist

1. Measure and Limit Resources Accurately
   - Time
   - Memory
2. Terminate Processes Reliably
3. Assign Cores Deliberately
4. Respect Non-Uniform Memory Access
5. Avoid Swapping
6. Isolate Individual Runs
   - Communication
   - File system

Measuring CPU time with "`time`"

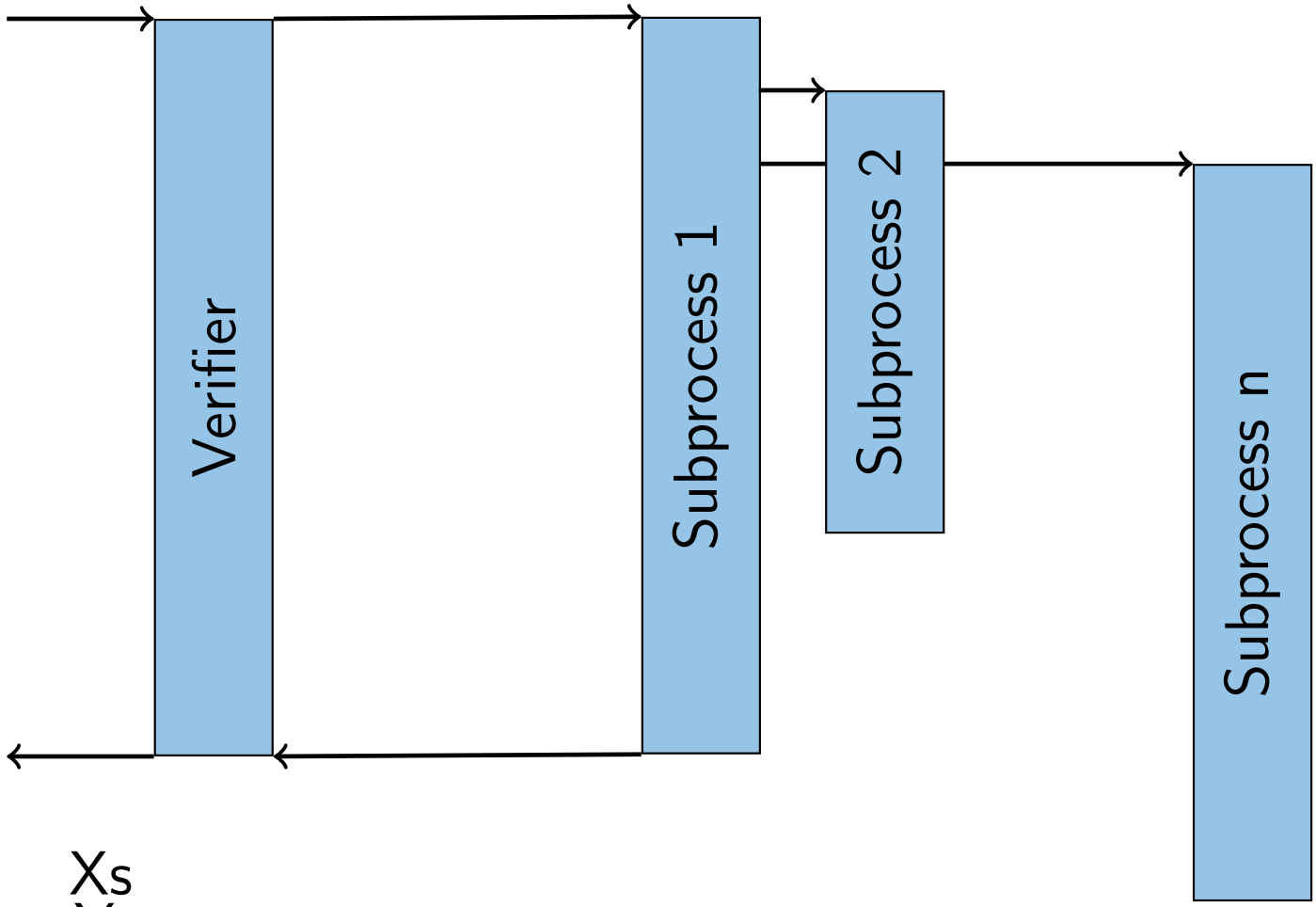
```
~$ time verifier
```



real	Xs
user	Ys
sys	Zs

Measuring CPU time with "`time`"

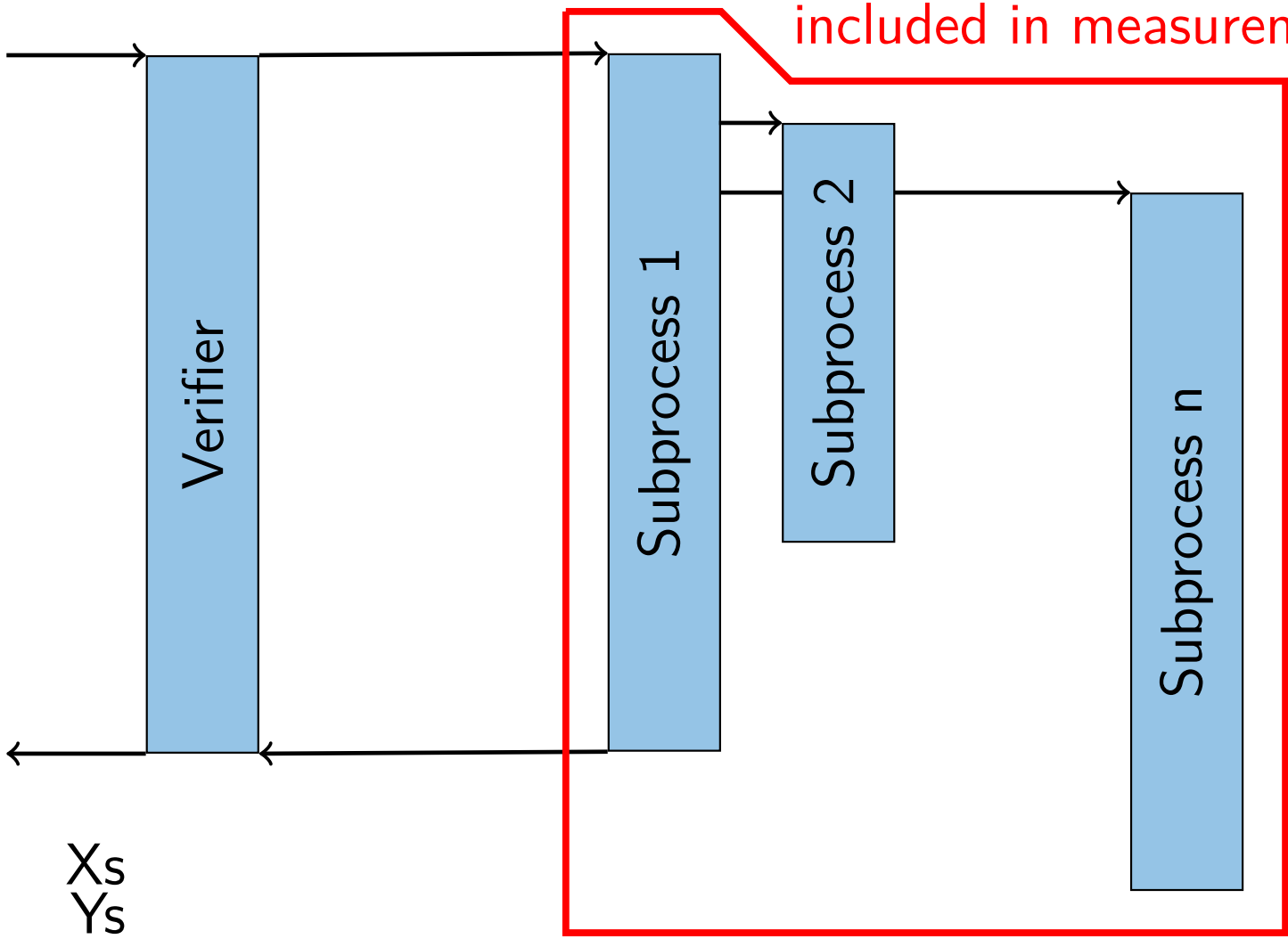
```
~$ time verifier
```



real Xs
user Ys
sys Zs

Measuring CPU time with "time"

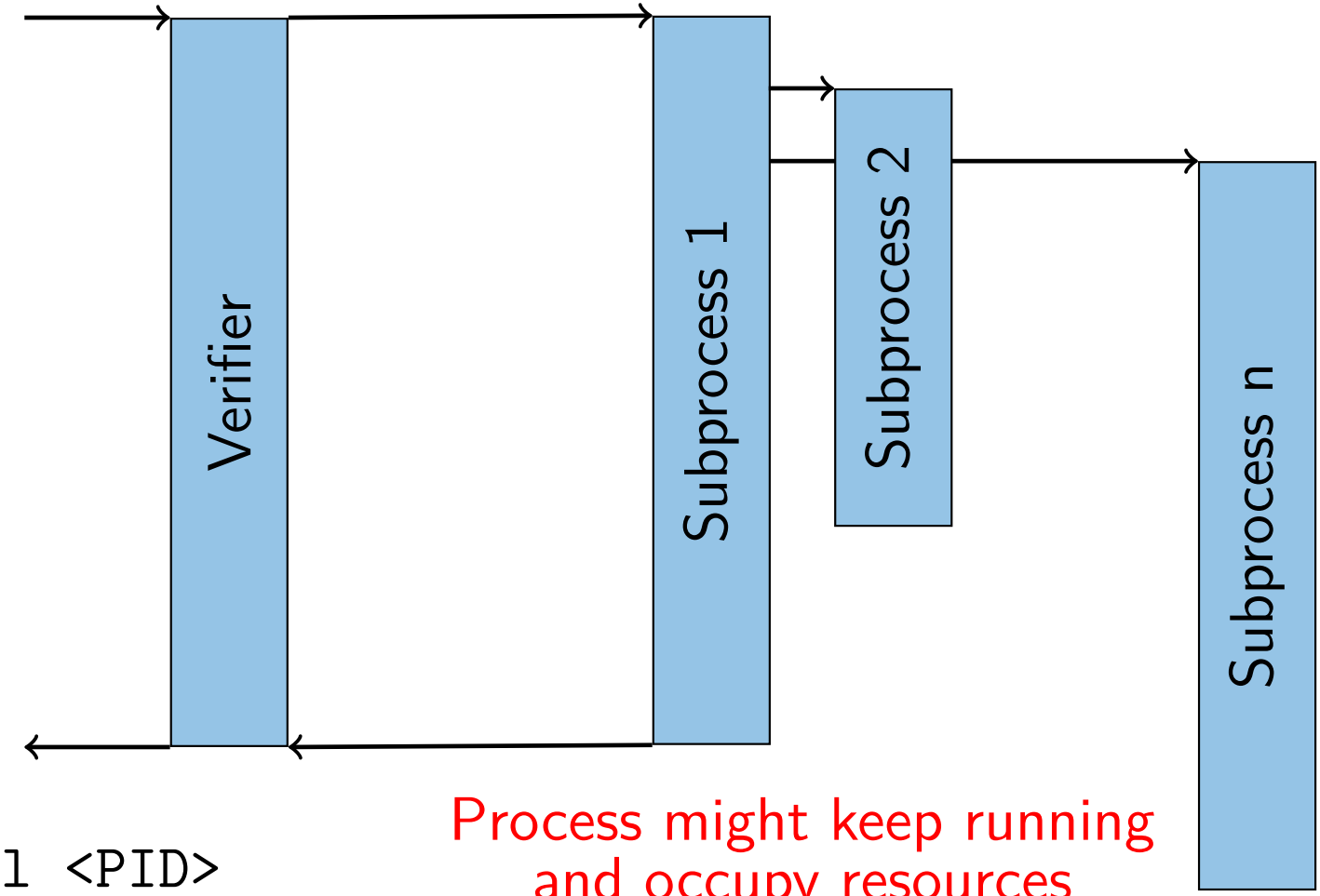
```
~$ time verifier
```



real Xs
user Ys
sys Zs

Terminate Processes Reliably

```
~$ verifier
```




```
~$ kill <PID>
```

Process might keep running
and occupy resources

Isolate Individual Runs

- Excerpt of start script taken from some verifier in SV-COMP:

  # ... (tool started here)

  ```
  killall z3 2> /dev/null
  killall minisat 2> /dev/null
  killall yices 2> /dev/null
  ```

- Thanks for thinking of cleanup

Isolate Individual Runs

- Excerpt of start script taken from some verifier in SV-COMP:

  # ... (tool started here)

  ```
  killall z3 2> /dev/null
  killall minisat 2> /dev/null
  killall yices 2> /dev/null
  ```

- Thanks for thinking of cleanup
- But what if there are parallel runs?

Isolate Individual Runs

- Temp files with constant names like `/tmp/mytool.tmp` collide
- State stored in places like `~/.mytool` hinders reproducibility
  - Sometimes even auto-generated
- Restrict changes to file system as far as possible

Benchmarking Containers

- Encapsulate groups of processes
- Limited resources (memory, cores)
- Total resource consumption measurable
- All other processes hidden
  and no communication with them
- Disabled network access
- Adjusted file-system layout
  - Private `/tmp`
  - Writes redirected to
    temporary RAM disk

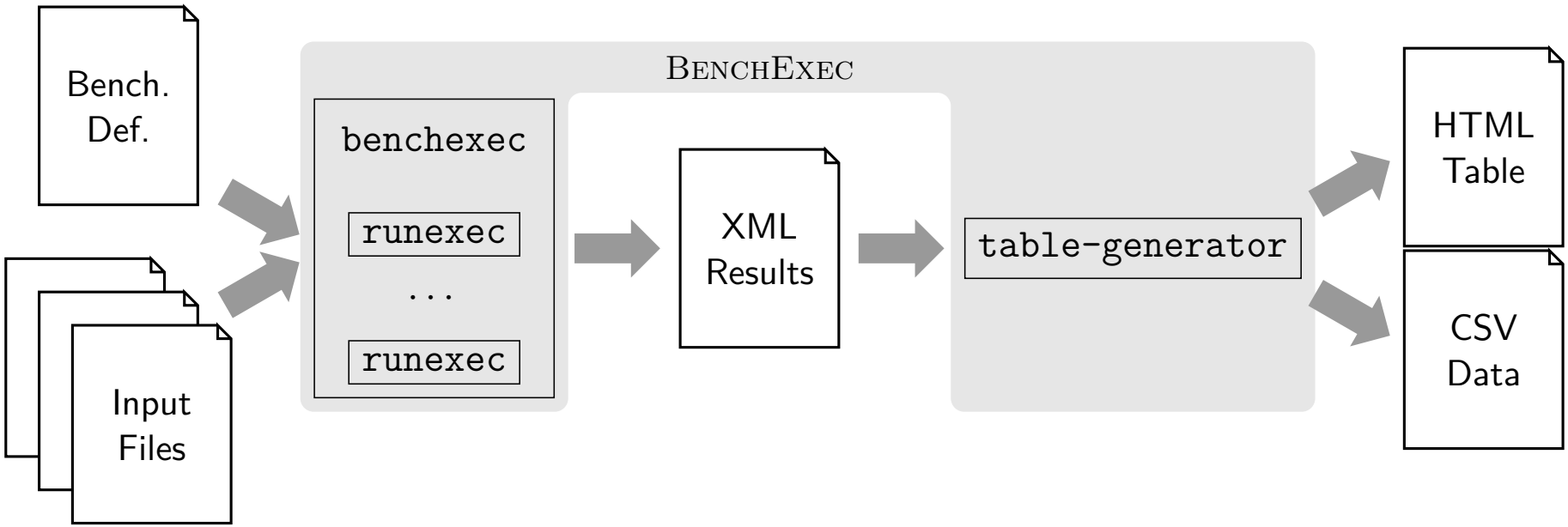
BenchExec

- A Framework for Reliable Benchmarking and Resource Measurement
- Provides benchmarking containers based on cgroups and namespaces
- Allocates hardware resources appropriately
- Low system requirements (modern Linux kernel and cgroups access)

BenchExec

- Open source: Apache 2.0 License
- Written in Python 3
- https://github.com/sosy-lab/benchexec
- Used in International Competition on Software Verification (SV-COMP) and by StarExec
- Originally developed for software-verification, but applicable to arbitrary tools

BenchExec Architecture



`runexec`

Benchmarks a single run of a tool (in container)

`benchexec`

Benchmarks multiple runs

`table-generator`

Generates CSV and interactive HTML tables

Conclusion

Be careful when benchmarking!

Don't use time, ulimit etc.
Always use cgroups and namespaces!

BenchExec
https://github.com/sosy-lab/benchexec

There's more

Dirk Beyer, *Stefan Löwe*, and *Philipp Wendler*.
Reliable Benchmarking: Requirements and Solutions. [1]
STTT 2019 (preprint available here)

- More details
- Study of hardware influence on benchmarking results
- Suggestions how to present results
  (result aggregation, rounding, plots, etc.)

References I

Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: Requirements and solutions. Int. J. Softw. Tools Technol. Transfer **21**(1), 1–29 (2019). https://doi.org/10.1007/s10009-017-0469-y