Benchmarking and Resource Measurement

Dirk Beyer Stefan Löwe Philipp Wendler

SoSy-Lab
Software Systems

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Benchmarking is Important

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

Reliable, reproducible, and accurate results needed!

Benchmarking is Hard

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

Not relevant for most verification tools

- Different hardware architectures
- Heterogenity of tools
- Parallel benchmarks

Relevant!

Goals

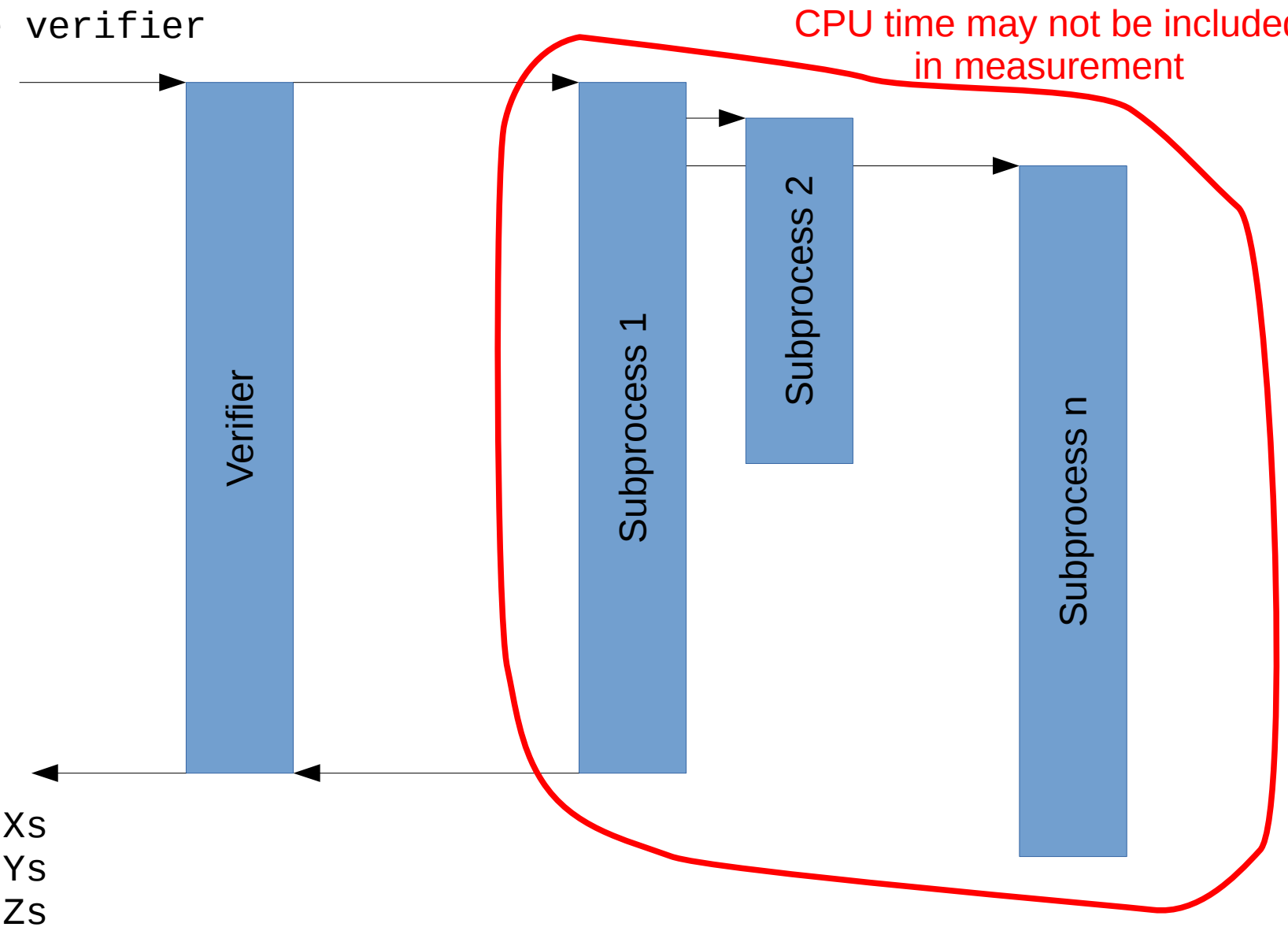
- Reproducibility
  - Avoid non-deterministic effects
  - Provide defined set of resources
- Accurate results
- For verification tools (and similar)
- On Linux

Measure and Limit Resources Accurately

- Wall time and CPU time
- Define memory consumption
  - Size of address space? Too large
  - Size of heap? Too low
  - Size of resident set (RSS)?
- Measure peak consumption
- Always define memory limit for reproducibility
- Include sub-processes

Measuring CPU time with „`time`“

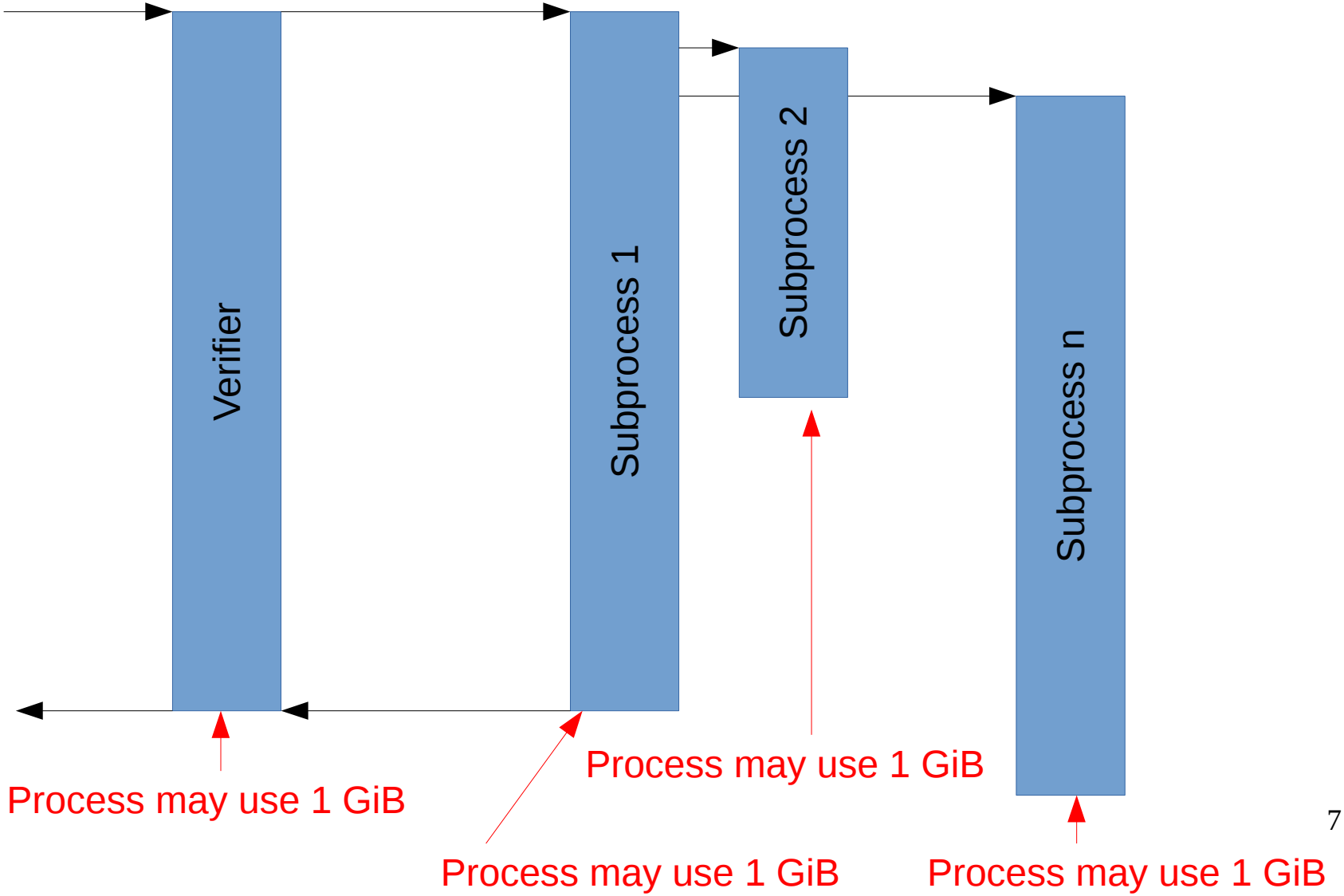
```
~$ time verifier
```




```
real    Xs
user    Ys
sys     Zs
```

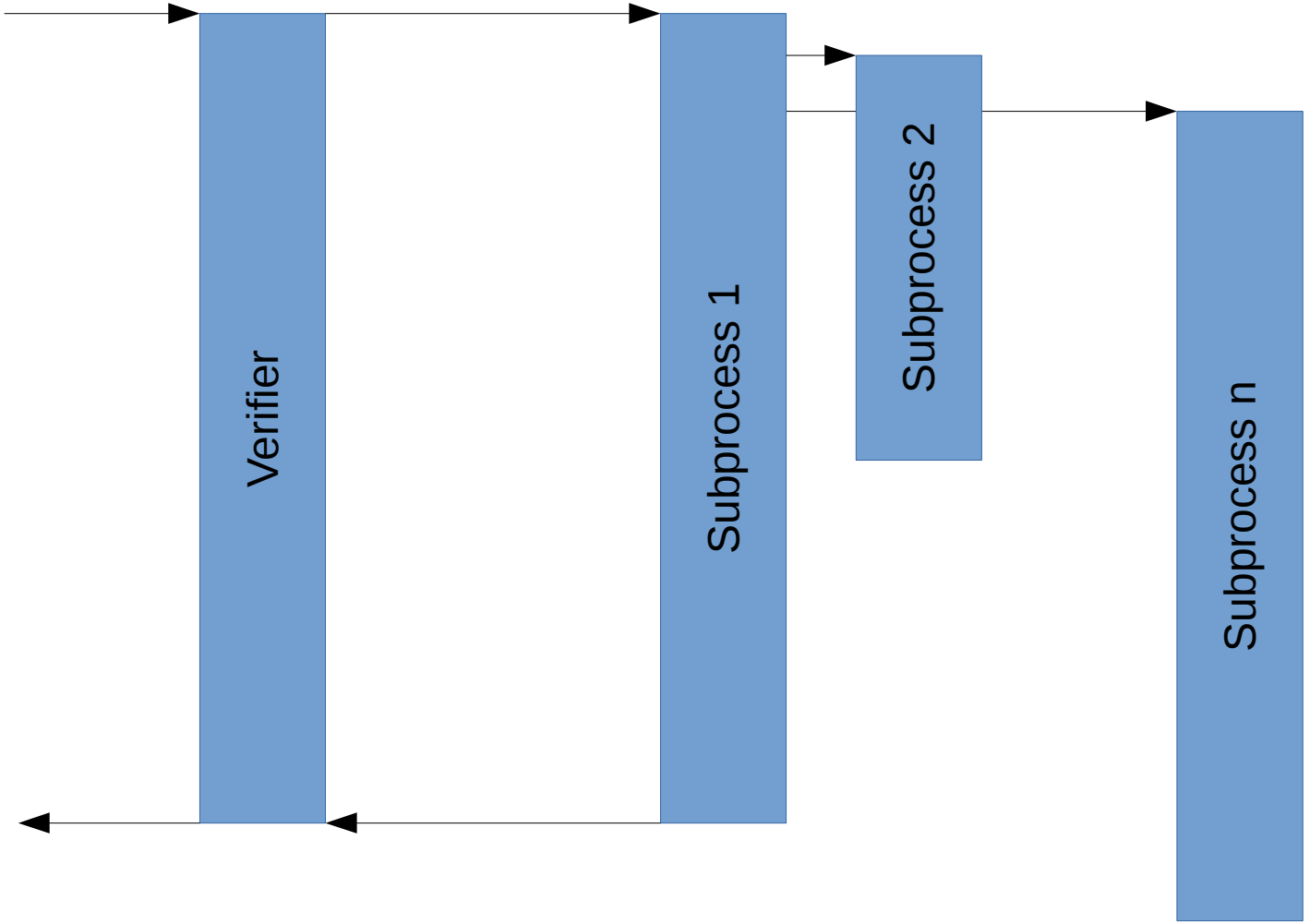
Limiting memory with „ulimit“

```
~$ ulimit -v 1048576 # 1 GiB
~$ verifier
```



Limiting memory with „ulimit“

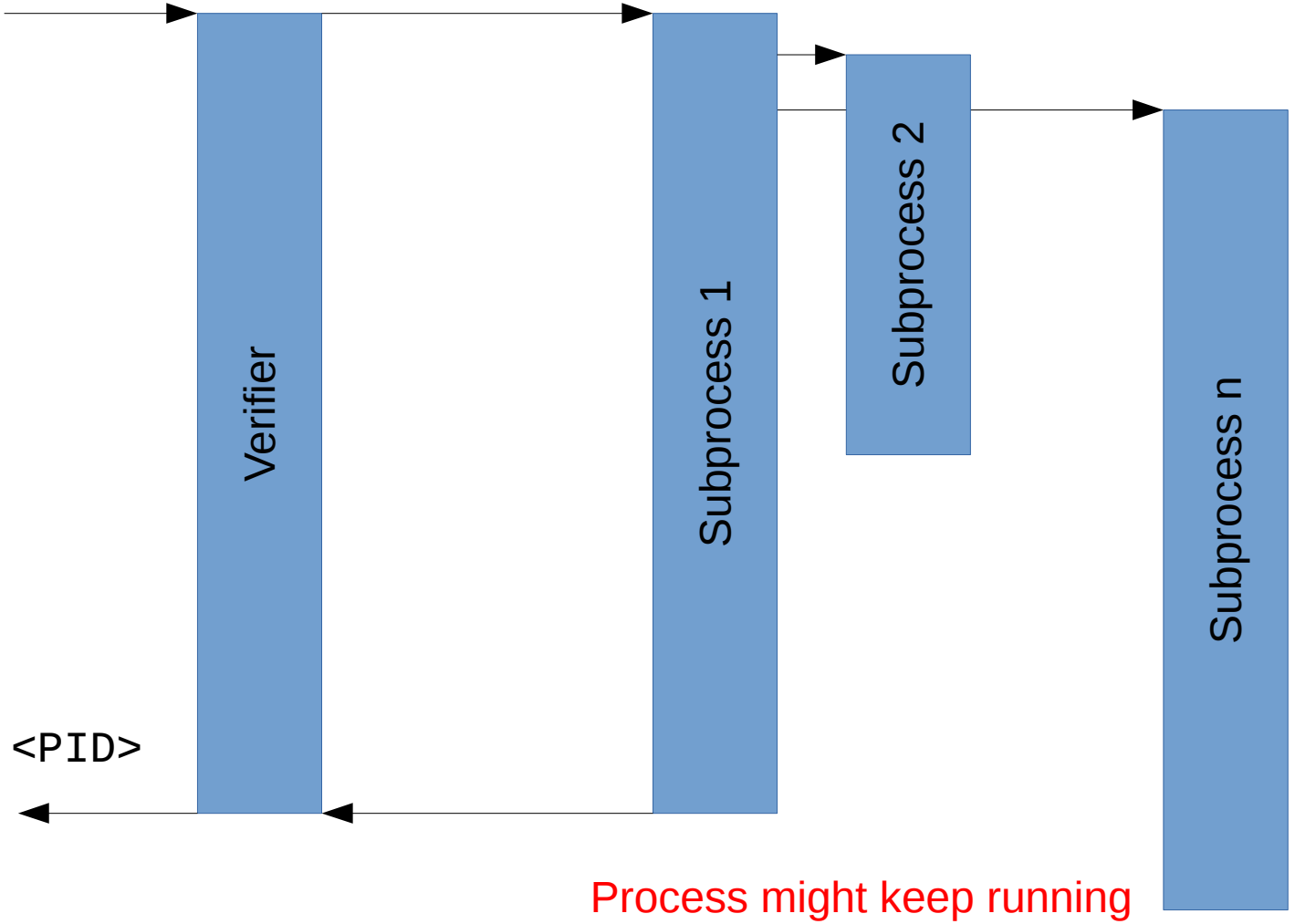
```
~$ ulimit -v 1048576 # 1 GiB
~$ verifier
```



What about shared memory?

Kill Processes Reliably

```
~$ verifier
```




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

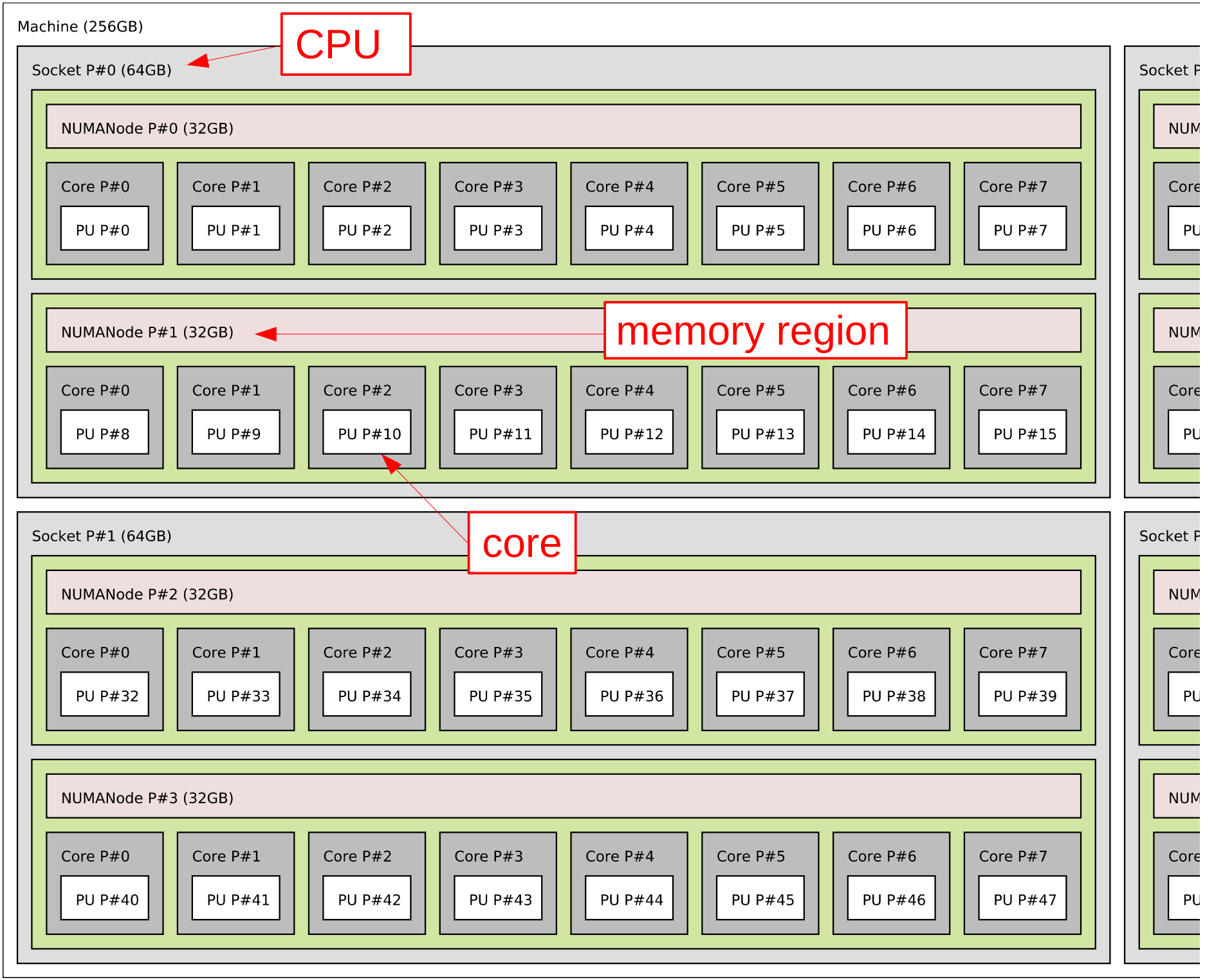
Process might keep running
and occupy ressources

Assign Cores Deliberately

- Hyper Threading:
  Multiple threads sharing execution units
- Shared caches

Respect Non-Uniform Memory Access (NUMA)

- Memory regions have different performance depending on current CPU core
- Hierarchical NUMA exists

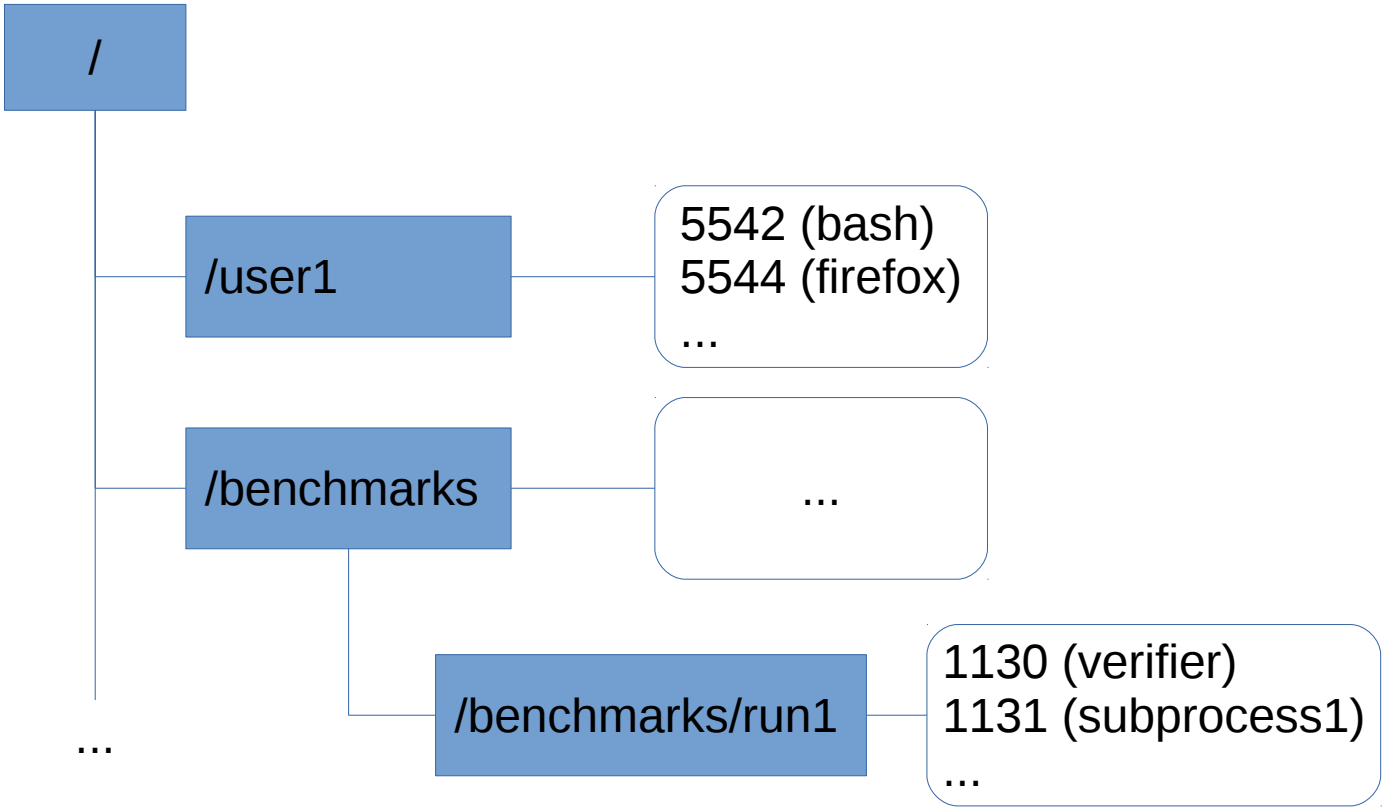

Machine (256GB)
CPU
Socket P#0 (64GB)
NUMANode P#0 (32GB)
Core P#0
PU P#0
Core P#1
PU P#1
Core P#2
PU P#2
Core P#3
PU P#3
Core P#4
PU P#4
Core P#5
PU P#5
Core P#6
PU P#6
Core P#7
PU P#7
NUMANode P#1 (32GB)
memory region
Core P#0
PU P#8
Core P#1
PU P#9
Core P#2
PU P#10
Core P#3
PU P#11
Core P#4
PU P#12
Core P#5
PU P#13
Core P#6
PU P#14
Core P#7
PU P#15
Socket P#1 (64GB)
core
NUMANode P#2 (32GB)
Core P#0
PU P#32
Core P#1
PU P#33
Core P#2
PU P#34
Core P#3
PU P#35
Core P#4
PU P#36
Core P#5
PU P#37
Core P#6
PU P#38
Core P#7
PU P#39
NUMANode P#3 (32GB)
Core P#0
PU P#40
Core P#1
PU P#41
Core P#2
PU P#42
Core P#3
PU P#43
Core P#4
PU P#44
Core P#5
PU P#45
Core P#6
PU P#46
Core P#7
PU P#47

Checklist

- Measure and Limit Resources Accurately
  - Time
  - Memory
- Kill Processes Reliably
- Assign Cores Deliberately
- Respect Non-Uniform Memory Access
- Avoid Swapping

Cgroups

- Linux kernel „control groups“
- Hierarchical tree of sets of processes



Cgroups

- Reliable tracking of spawned processes
- Resource limits and measurements per cgroup
  - CPU time
  - Memory
  - I/O etc.

Only solution on Linux
for race-free handling of multiple processes!

BenchExec

- A Framework for Reliable Benchmarking and Resource Measurement
- Based on cgroups
- Handles multiple processes
- Allocates hardware resources appropriately
- Low system requirements (Linux and cgroups)

BenchExec

- Open source: Apache 2.0 License
- Written in Python 3
- https://github.com/dbeyer/benchexec
- Paper under submission
- Used in International Competition on Software Verification (SV-COMP)

  22 tools this year
- Originally developed for software verification, but applicable to arbitrary tools

BenchExec Architecture

- `runexec`
  - Benchmarks a single run of a tool
  - Measures and limits resources
  - Easy integration into other frameworks
- `benchexec`
  - Benchmarks multiple runs
    (e.g., a set of configurations against a set of files)
  - Allocates hardware resources
  - Can check whether tool result is as expected
- `table-generator`
  - Generates CSV and interactive HTML tables (with plots)

Conclusion

Be careful when benchmarking!

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

Try out BenchExec:
https://github.com/dbeyer/benchexec

Or ask me for a demo!