Reliable Benchmarking of Software Verification in the Cloud

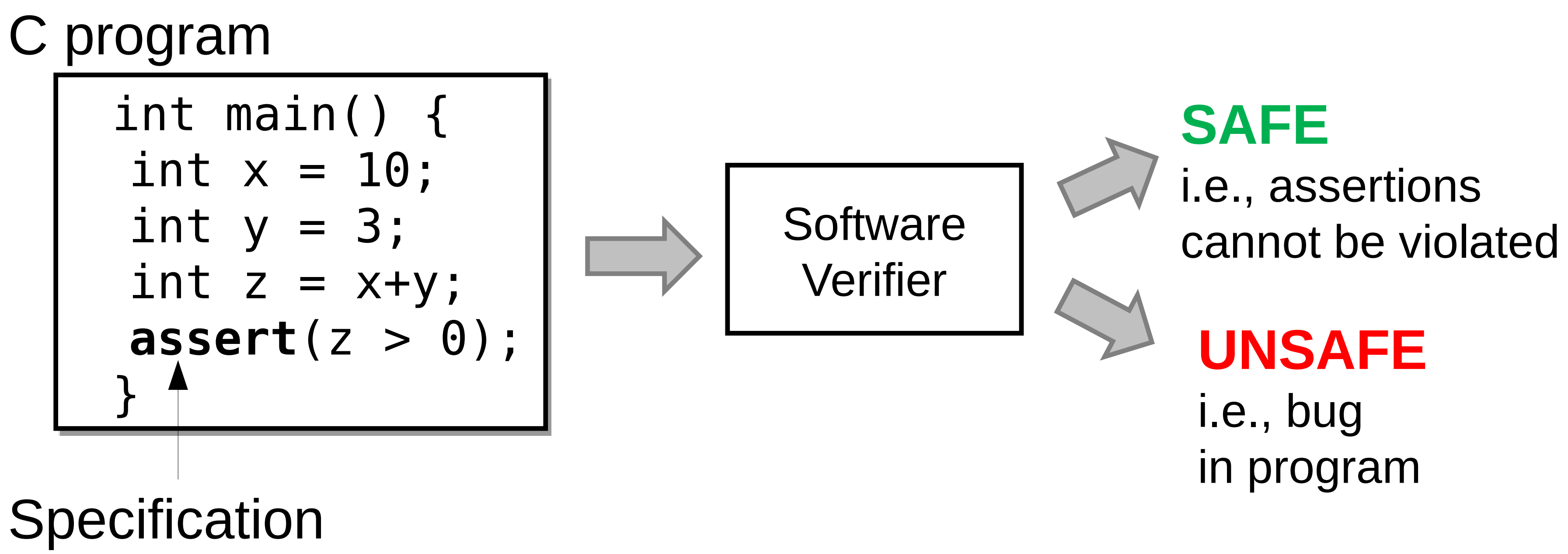
Philipp Wendler

SoSy-Lab
Software Systems

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Software Verification

Goal: Build an automatic software verifier



```
int main() {
 int x = 10;
 int y = 3;
 int z = x+y;
 assert(z > 0);
}
```

Our tool: CPAchecker
http://cpachecker.sosy-lab.org

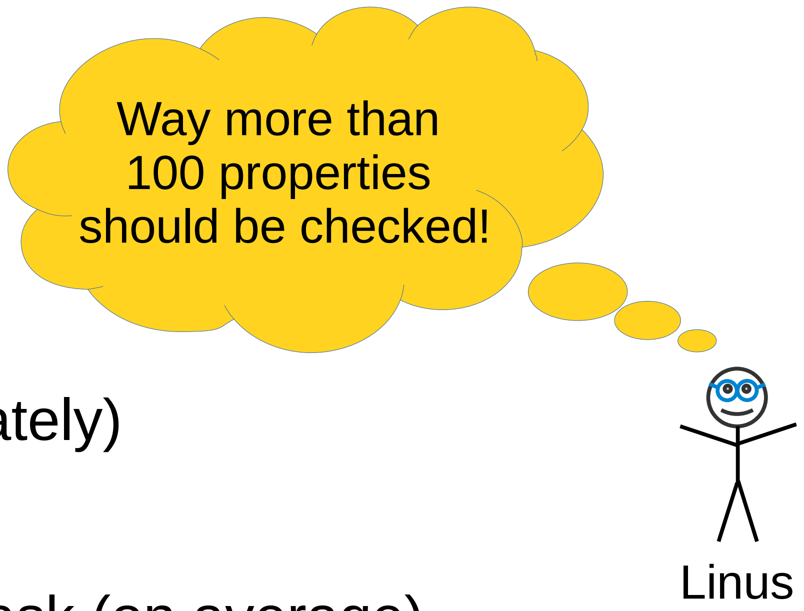
Linux Driver Verification

Photo: (c) Sean Bonner, 2013

Weekly Verification Workload

Time to verify...

- 1300 commits
- 4 drivers affected (approximately)
- 100 (safety) properties
- 12 seconds per verification task (on average)



1700 hours, or **72 days** of CPU time

Software Verification in the Cloud

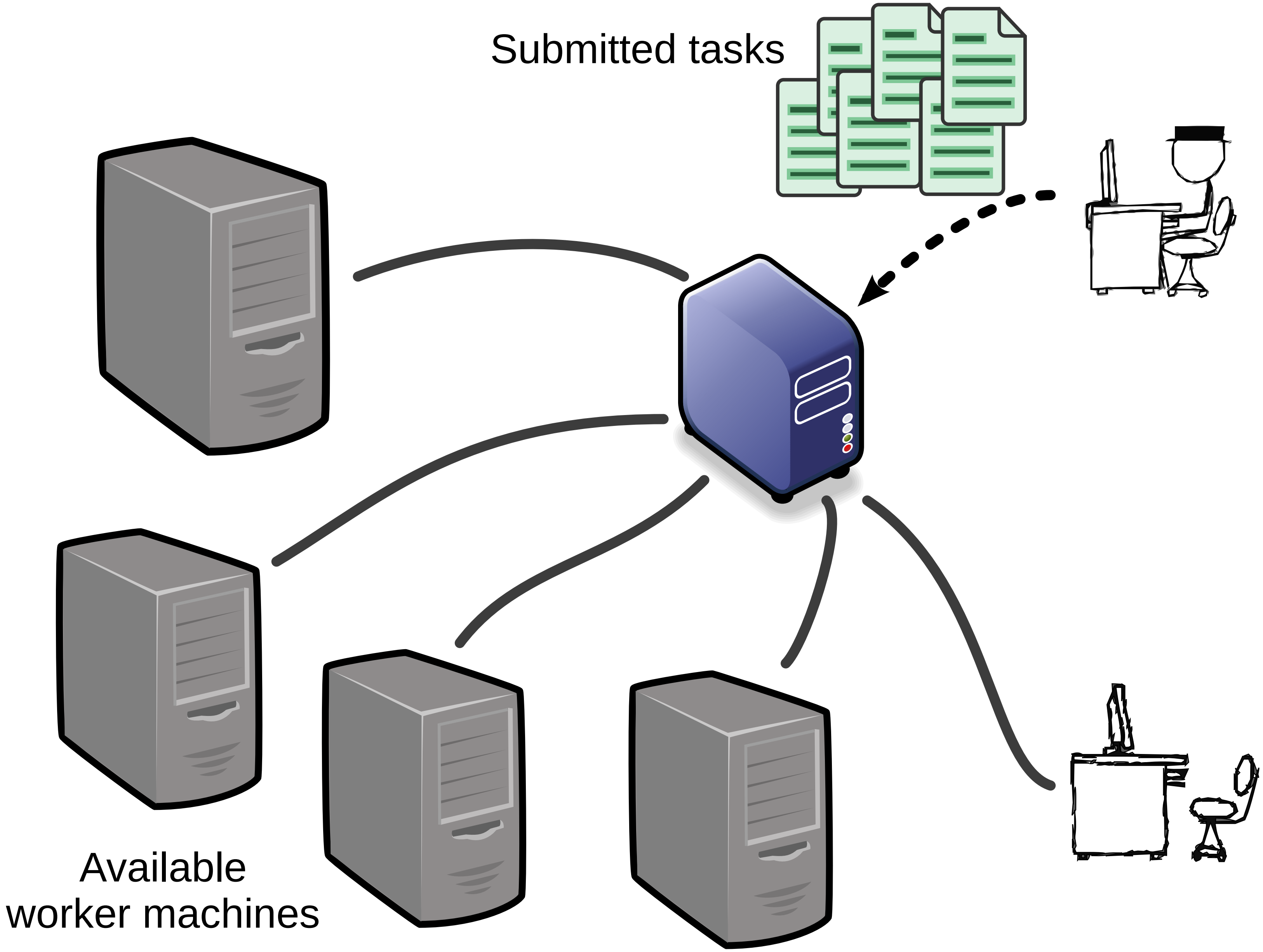
Verification of large number of tasks

Reliable benchmarking:
Accurate and reproducible results

VerifierCloud: Design

- Distribution system for verification tasks
- Task submission, e.g., via web frontend
- Creates necessary execution environment
- Supports arbitrary worker machines
  (no shared file system necessary)

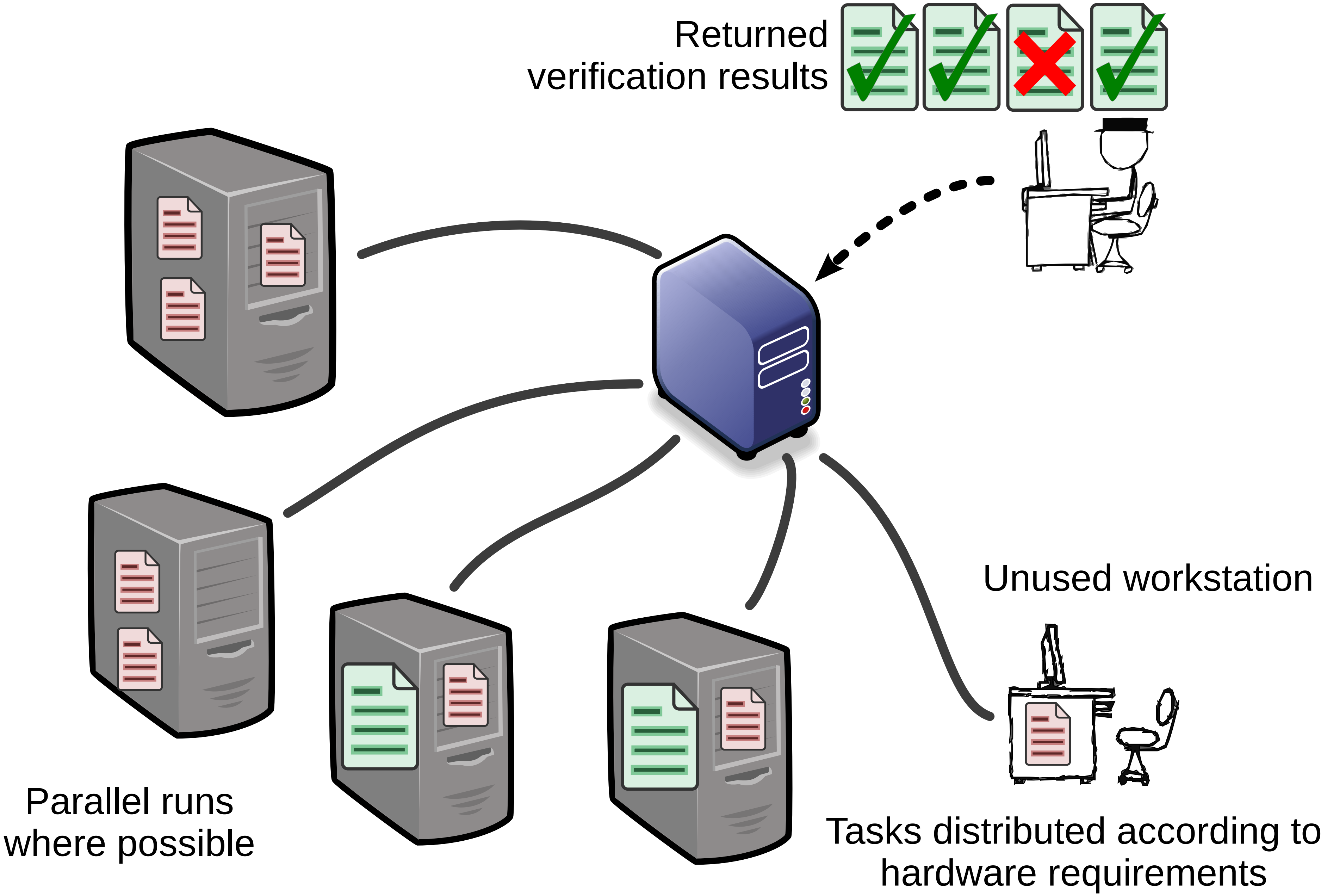
VerifierCloud: Architecture



VerifierCloud: Machine Allocation

- Partitions hardware resources
  and executes multiple runs
  in parallel on each machine
- Worker machines can be used dynamically
  when not needed by other users

VerifierCloud: Architecture

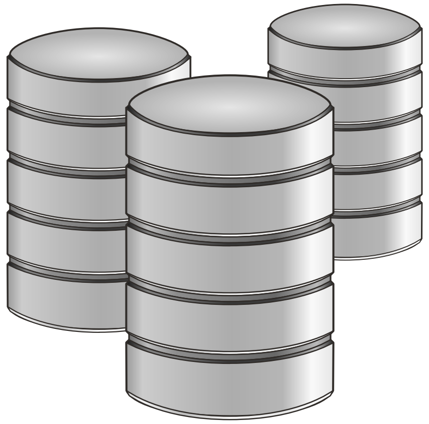

Returned verification results
Unused workstation
Parallel runs where possible
Tasks distributed according to hardware requirements

VerifierCloud: Results

- Successful use in development of our verifier CPAchecker
  - Multiple groups, ca. 40 developers
  - Significantly speeds up implement-test-roundtrip time
  - Up to 800 runs executing in parallel
  - 1 000 000 runs per week
- Used in teaching (Passau, Paderborn, Hamburg)
  - Students can learn verification without installation
  - Resources for experimenting with own implementations

VerifierCloud: Future

- Reuse of results
  - Similar verification tasks can benefit from reusing intermediate results
  - Example: new revision of same driver
- Store (intermediate) results in database?
- Automatically use stored information for new tasks?

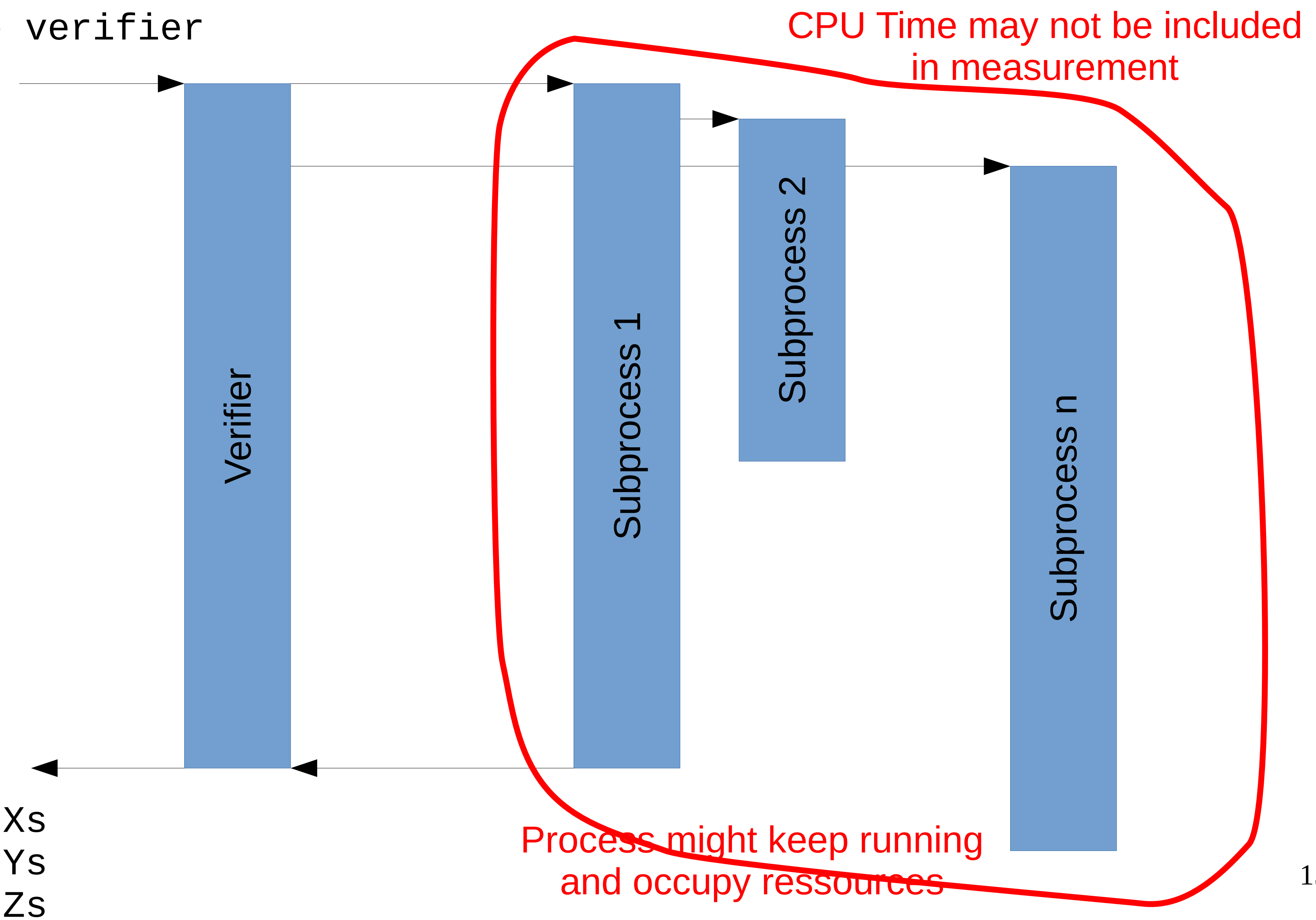


Reliable Benchmarking

- Shared Machines
- Arbitrary Tools
  - Non-interactive
  - CPU-/Memory bound (no I/O)
- Measure and limit CPU time
- Measure and limit memory
- Different Hardware Architectures

Measuring CPU time with „time“

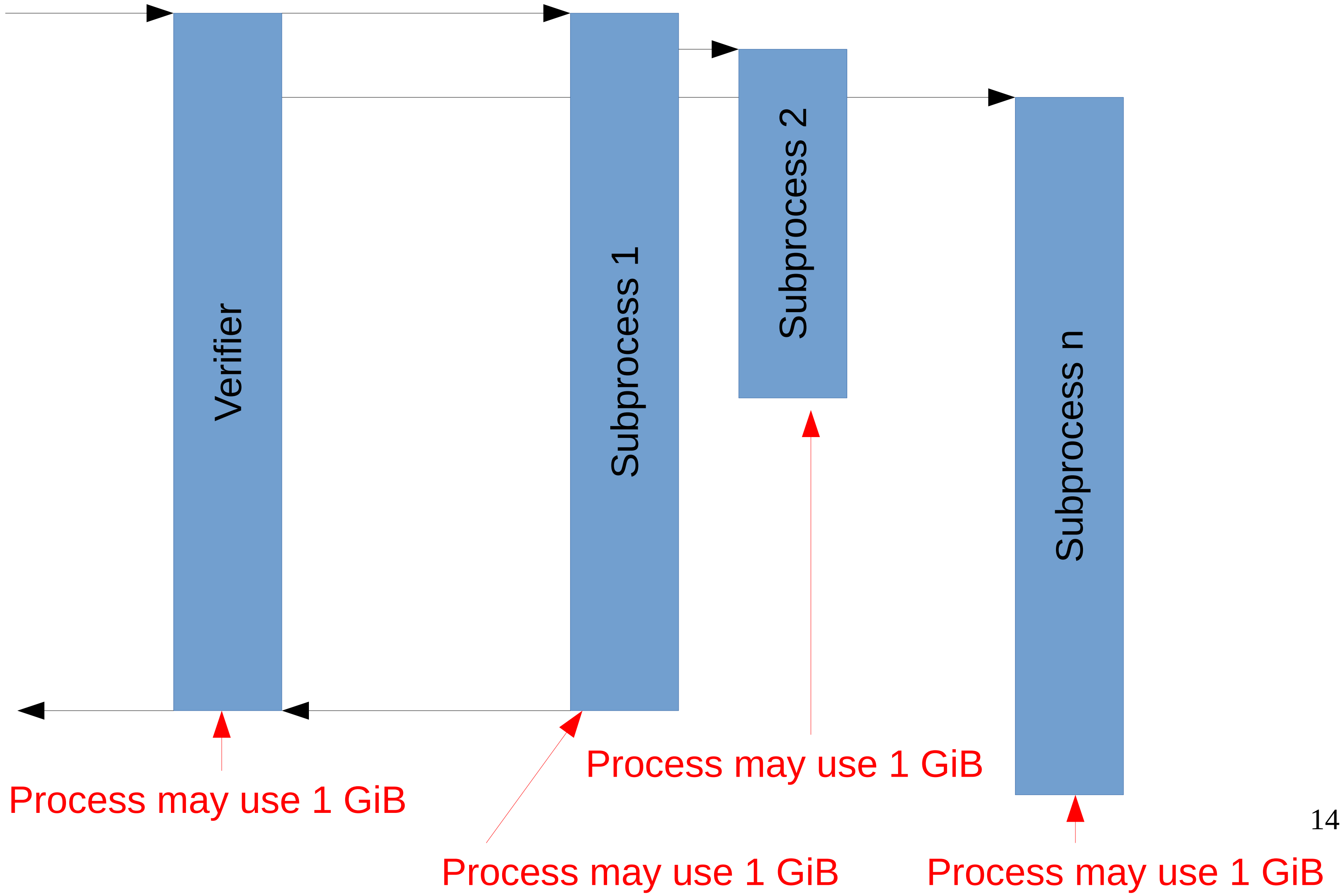
```
~$ time verifier
```


CPU Time may not be included in measurement
Verifier
Subprocess 1
Subprocess 2
Subprocess n
Process might keep running and occupy ressources

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

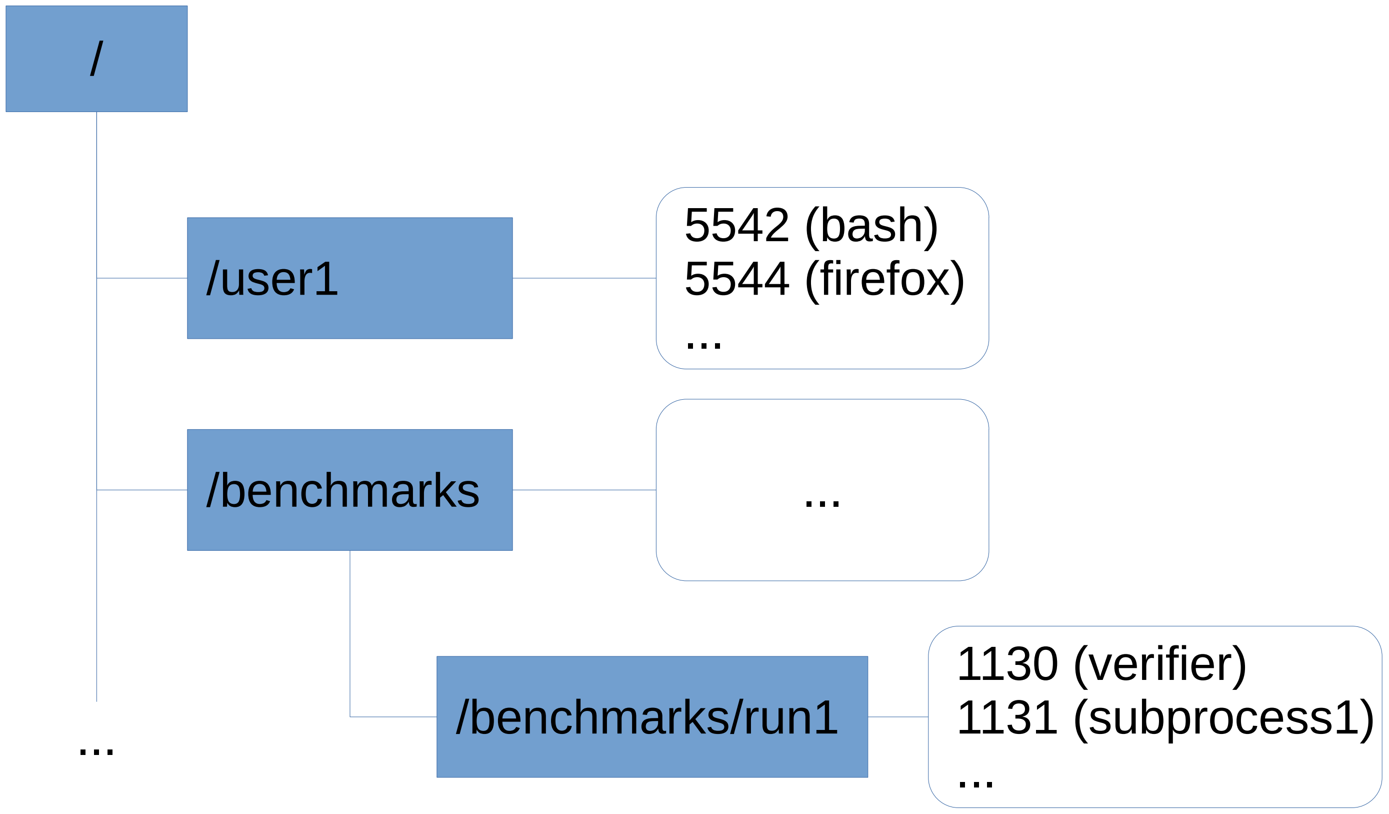
Limiting memory with „ulimit“

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



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!

Hardware Architectures

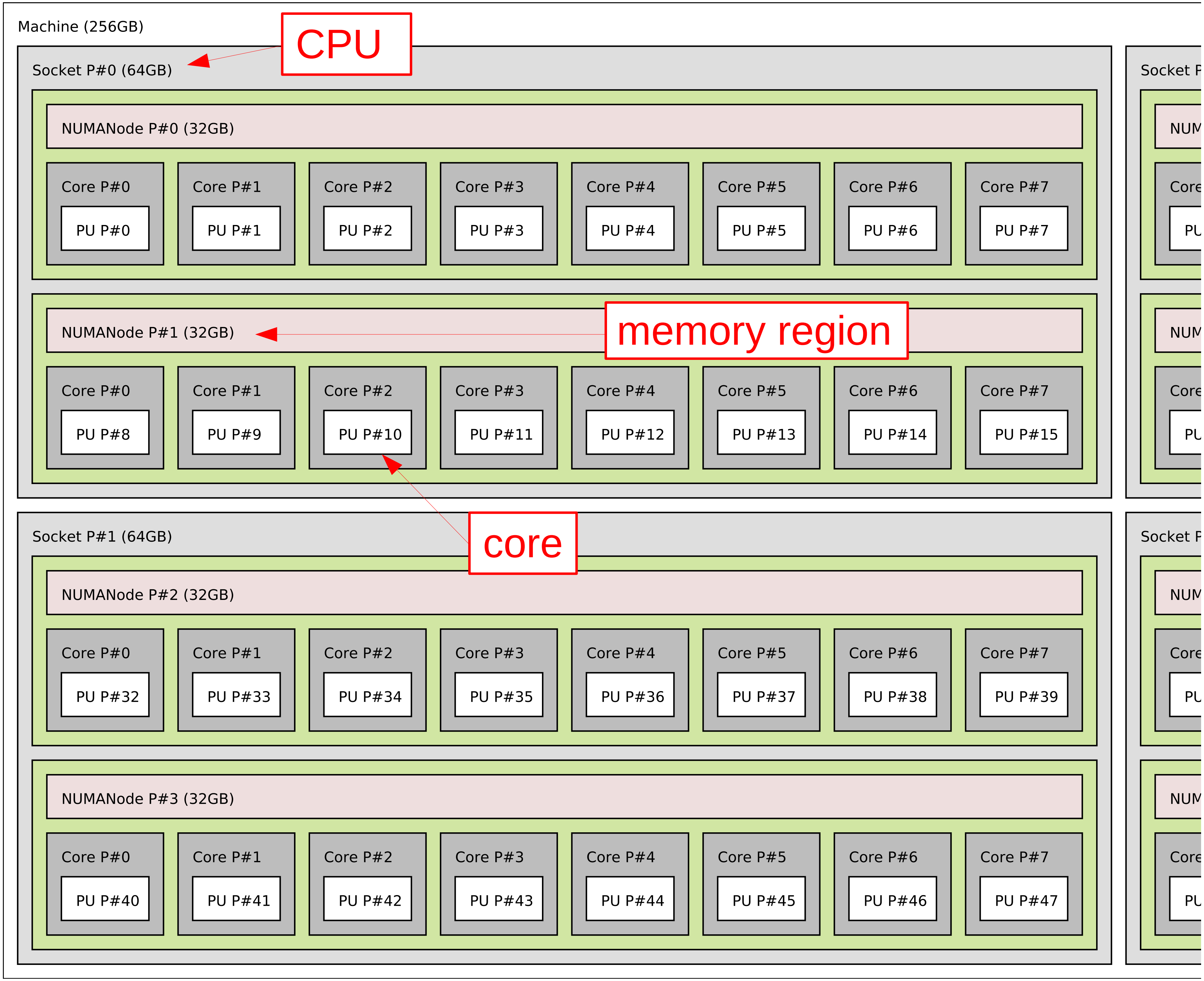
- Hyper Threading

  Multiple threads sharing execution units

- Non-Uniform Memory Access

  Memory regions have different performance depending on current CPU core

- And more (caches, …)
- Can lead to non-deterministic performance


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
core
Socket P#1 (64GB)
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

BenchExec

- A Framework for Reliable Benchmarking and Resource Measurement
- Based on cgroups
- Handles multiple processes
- Allocates hardware resources appropriately
- Used in International Competition on Software Verification (SV-COMP)

    22 tools this year

BenchExec

- Open source: Apache 2.0 License
- https://github.com/dbeyer/benchexec
- Paper under submission
- Extensible
  - Arbitrary tools
  - Not only for software verification