Interacting with gem5 using WA & devlib

Anouk Van Laer

gem5 workshop
11/09/2017

Multiple methods to run gem5

- Run **gem5 standalone** – Via terminal and scripts

  + Easy setup

  - Inflexible and hard to share

- **devlib** – Device abstraction layer, allowing gem5 interaction using Python

  + Platform agnostic and easy to share using Python notebooks

  - Initial setup

- **workload-automation** – Framework to automate running workloads on Arm devices

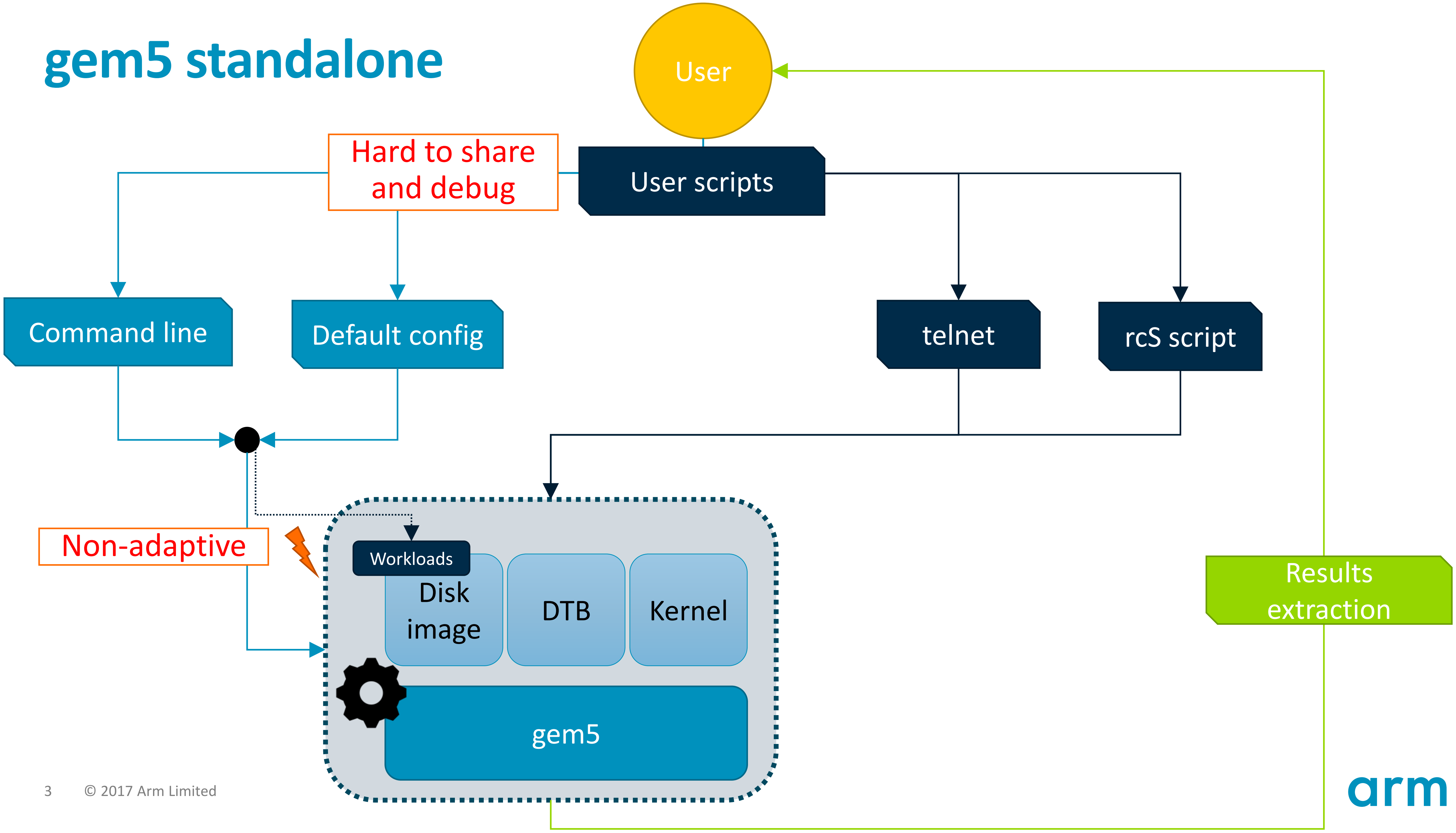
  + Platform agnostic, includes ready workloads and easy to share using agendas

  - Initial setup

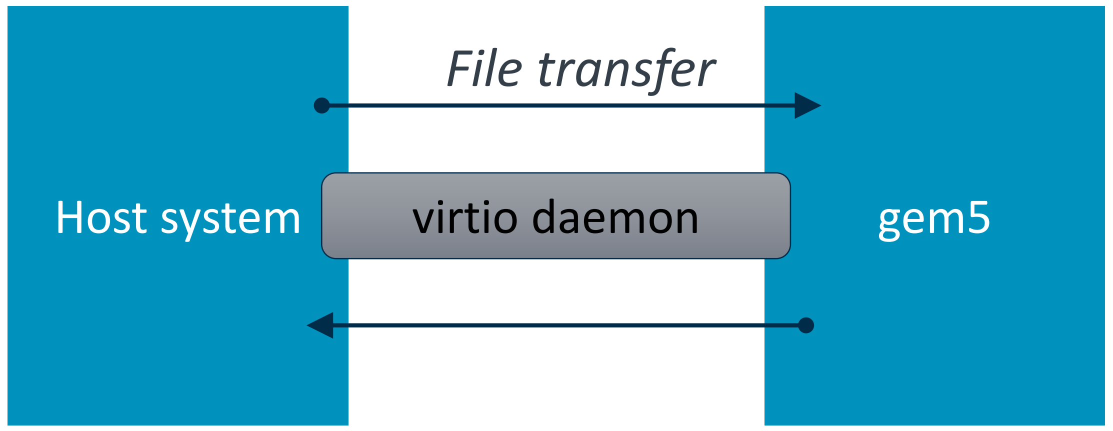
Can be found on

https://github.com/ARM-software

gem5 standalone



Prepping gem5 for interaction with WA/devlib

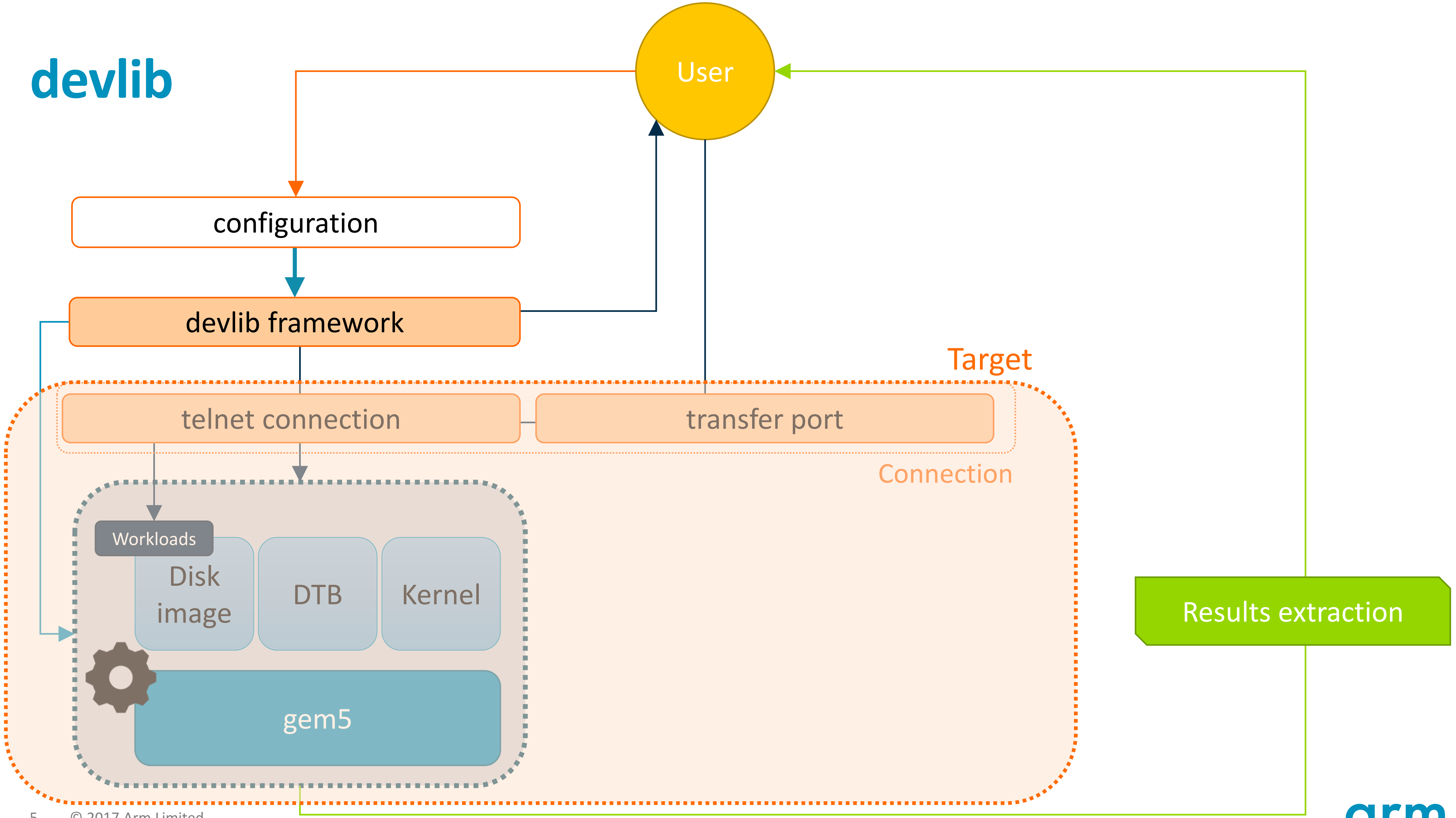


- Install diod on the host system

- Enable 9P/virtio support in kernel
- Add a couple of patches to gem5 itself
  - RealView.py
  - Configuration file

Exact details can be found on **gem5.org/WA-gem5**

devlib


User
configuration
devlib framework
Target
telnet connection
transfer port
Connection
Workloads
Disk image
DTB
Kernel
gem5
Results extraction

devlib - Usage

```
from devlib import *
stats_dir = '/home/gem5/output'

# Create the gem5 platform and set it up
platform = Gem5SimulationPlatform('gem5',stats_dir,
                                   gem5_bin='/home/gem5/build/ARM/gem5.opt',
                                   gem5_args='/home/gem5/configs/example/fs.py',
                                   gem5_virtio='--workload-automation-vio={}')
target = LinuxTarget(conn_cls=Gem5Connection, platform=p)
t.setup()

# Execute normal commands
t.execute('ls -l')
# Execute m5 commands
t.execute('m5 dumpstats')
# Pull (& push files across)
t.pull('file_in_gem5_system', 'destination_on_host')

# Nicely end simulation
t.disconnect()
```

devlib

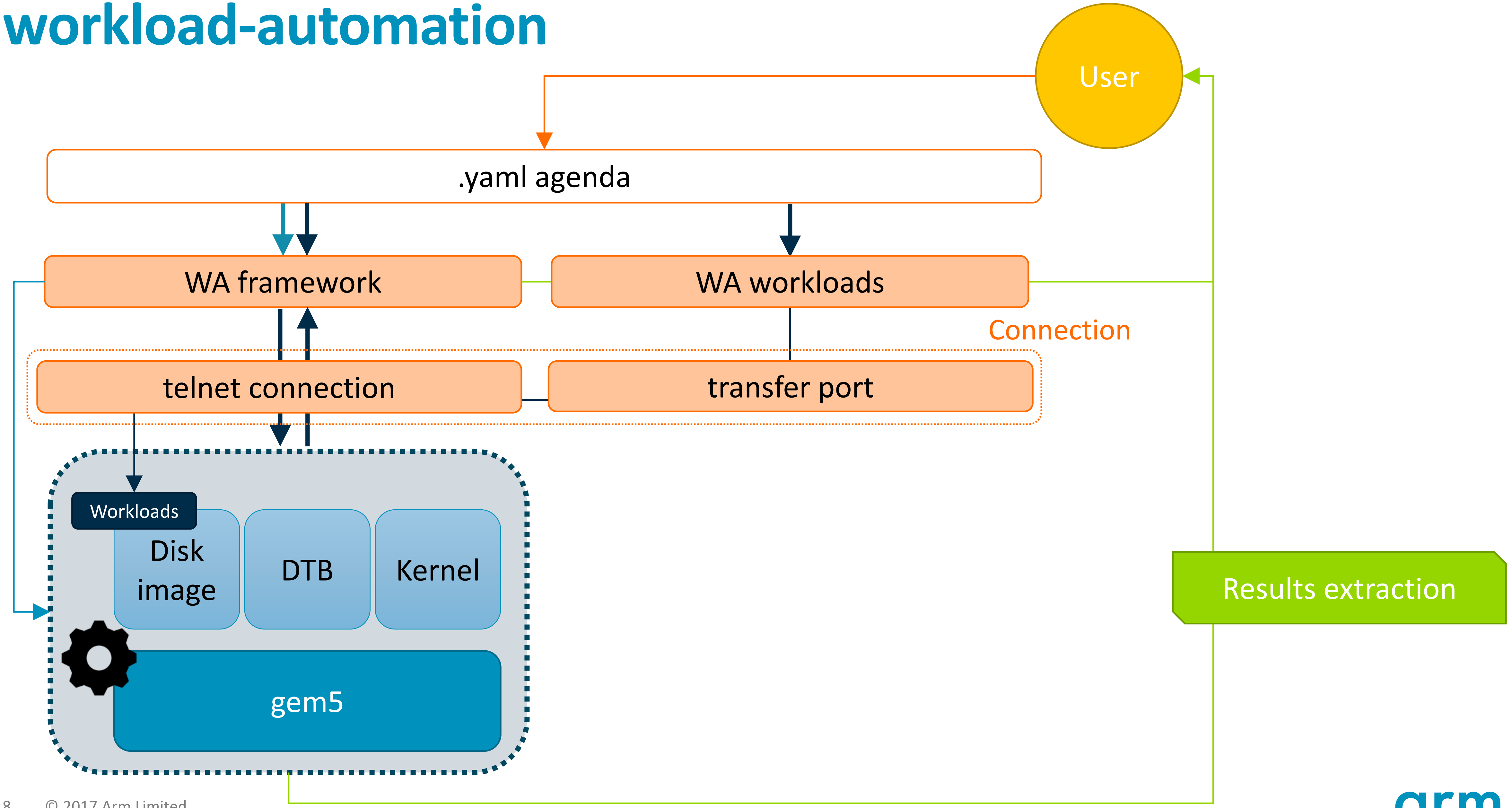
Usage

```python
from devlib import *
stats_dir = '/home/gem5/output'

# Create the gem5 platform and set it up
platform = Gem5SimulationPlatform('gem5',stats_dir,
                                  gem5_bin='/home/gem5/build/ARM/gem5.opt',
                                  gem5_args='/home/gem5/configs/example/fs.py',
                                  gem5_virtio='--workload-automation-vio={}')
target = LinuxTarget(conn_cls=Gem5Connection, platform=p)
t.setup()

# Execute normal commands
t.execute('ls -l')
# Execute m5 commands
t.execute('m5 dumpstats')
# Pull (& push files across)
t.pull('file_in_gem5_system', 'destination_on_host')

# Nicely end simulation
t.disconnect()
```

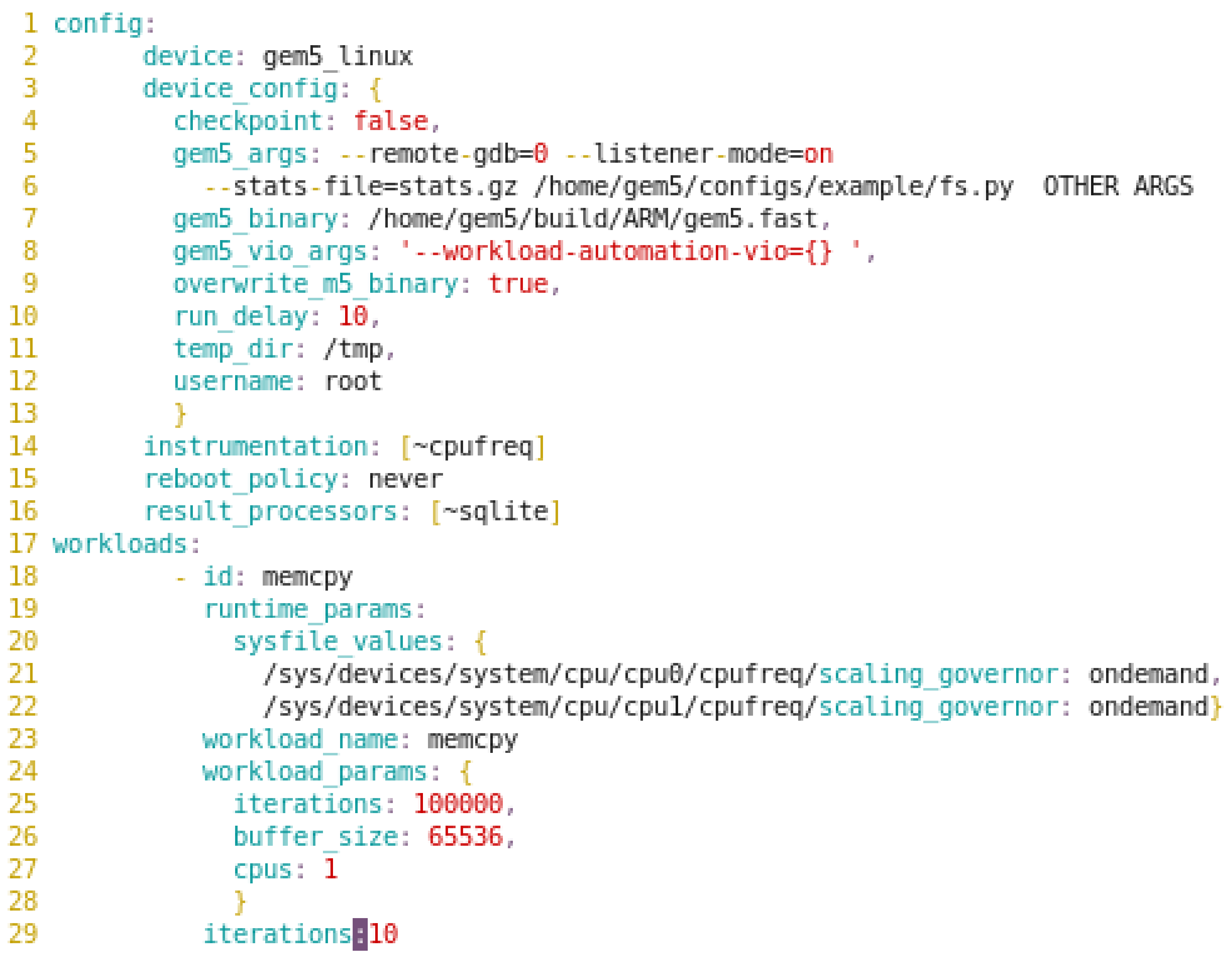
Additional functionality

- Modules add extra functionality to the target
  - *cpufreq* – change CPU frequency/governors
  - *gem5stats* – read statistics during runtime
- Instruments collect measurements from the target
  - *gem5power* – set/reset statistics dumps and read specific power related statistics

workload-automation



workload-automation - Usage

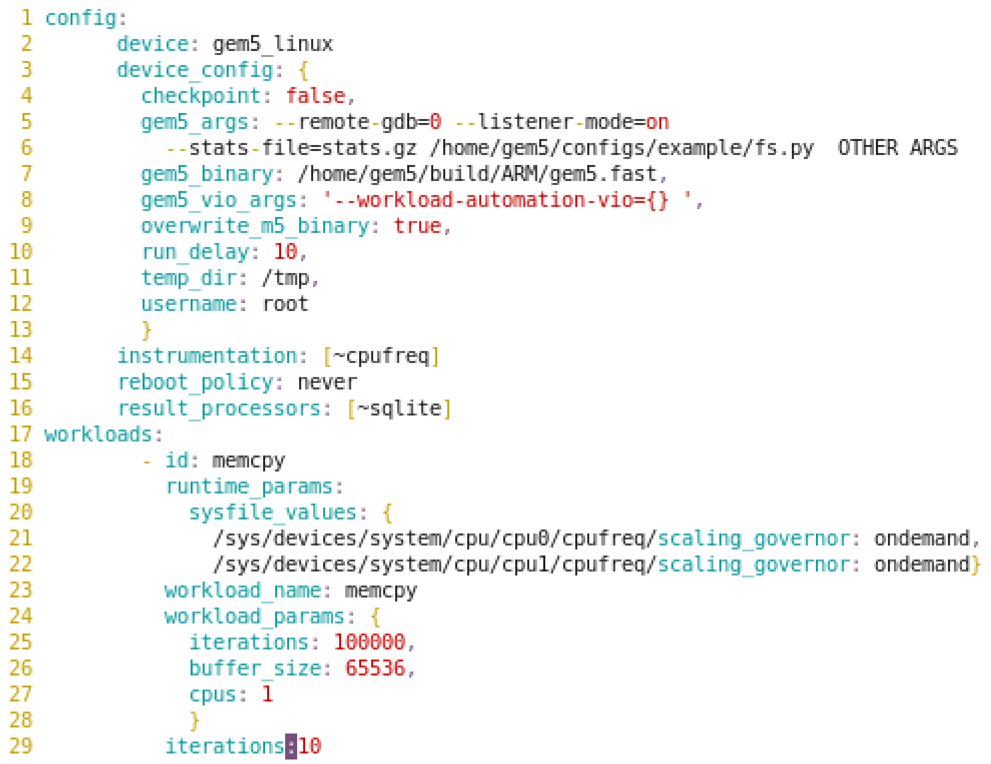


```
config:
   device: gem5_linux
   device_config: {
    checkpoint: false,
    gem5_args: --remote-gdb=0 --listener-mode=on
     --stats-file=stats.gz /home/gem5/configs/example/fs.py  OTHER ARGS
    gem5_binary: /home/gem5/build/ARM/gem5.fast,
    gem5_vio_args: '--workload-automation-vio={} ',
    overwrite_m5_binary: true,
    run_delay: 10,
    temp_dir: /tmp,
    username: root
    }
   instrumentation: [~cpufreq]
   reboot_policy: never
   result_processors: [~sqlite]
workloads:
    - id: memcpy
     runtime_params:
      sysfile_values: {
       /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor: ondemand,
       /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor: ondemand}
     workload_name: memcpy
     workload_params: {
      iterations: 100000,
      buffer_size: 65536,
      cpus: 1
      }
     iterations:10
```

workload-automation

1. *Addressing workload variability in architectural simulations*, Alameldeen A et al., IEEE Micro 2003

Usage



```
config:
      device: gem5_linux
      device_config: {
        checkpoint: false,
        gem5_args: --remote-gdb=0 --listener-mode=on
          --stats-file=stats.gz /home/gem5/configs/example/fs.py  OTHER ARGS
        gem5_binary: /home/gem5/build/ARM/gem5.fast,
        gem5_vio_args: '--workload-automation-vio={} ',
        overwrite_m5_binary: true,
        run_delay: 10,
        temp_dir: /tmp,
        username: root
        }
      instrumentation: [~cpufreq]
      reboot_policy: never
      result_processors: [~sqlite]
workloads:
        - id: memcpy
          runtime_params:
            sysfile_values: {
              /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor: ondemand,
              /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor: ondemand}
          workload_name: memcpy
          workload_params: {
            iterations: 100000,
            buffer_size: 65536,
            cpus: 1
            }
          iterations:10
```

Additional functionality

- Repetition[1] and automation!
  - Run the same workload multiple times → local iterations
  - Run multiple workloads consecutively
  - Run all of this multiple times → global iterations
- Workloads are already included (e.g. *dhrystone*, *memcpy, …*)
- Modules – similar concept to devlib
- Instrumentation – similar concept to devlib

Remarks

- Make sure the binary you use matches the system you are simulating
  → if gem5 is simulating a 64-bit system, it has to be a 64-bit binary

- If you add something interesting, please contribute it back!

Tool choice

	Standalone gem5	Workload-automation	devlib
One-off interaction *	✓		
Ready workloads		✓	
Direct interaction	✓		✓
Repetition		✓	
Push & pull files at runtime		✓	✓

* Not sure such a thing really exists in gem5 ☺

arm

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद