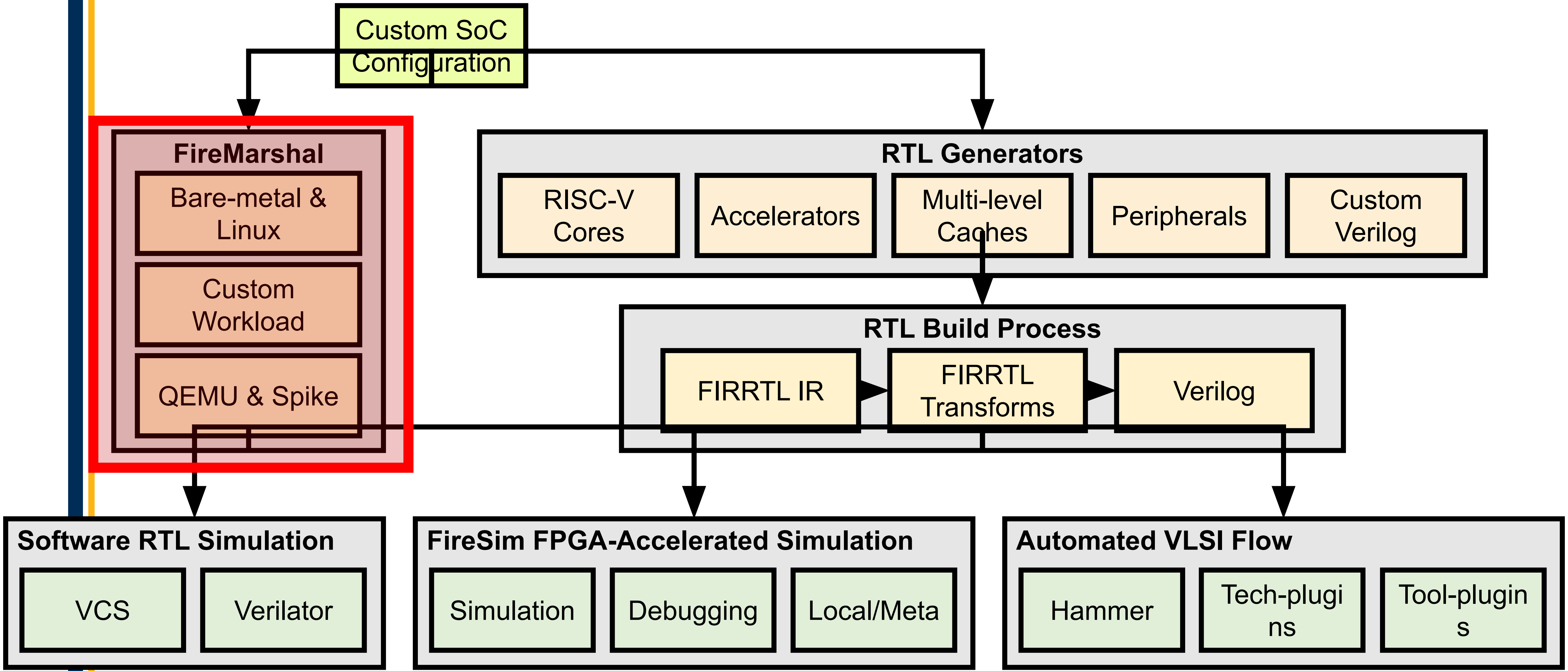
FireMarshal

Software Workload Management

Jerry Zhao
jzh@berkeley.edu
UC Berkeley

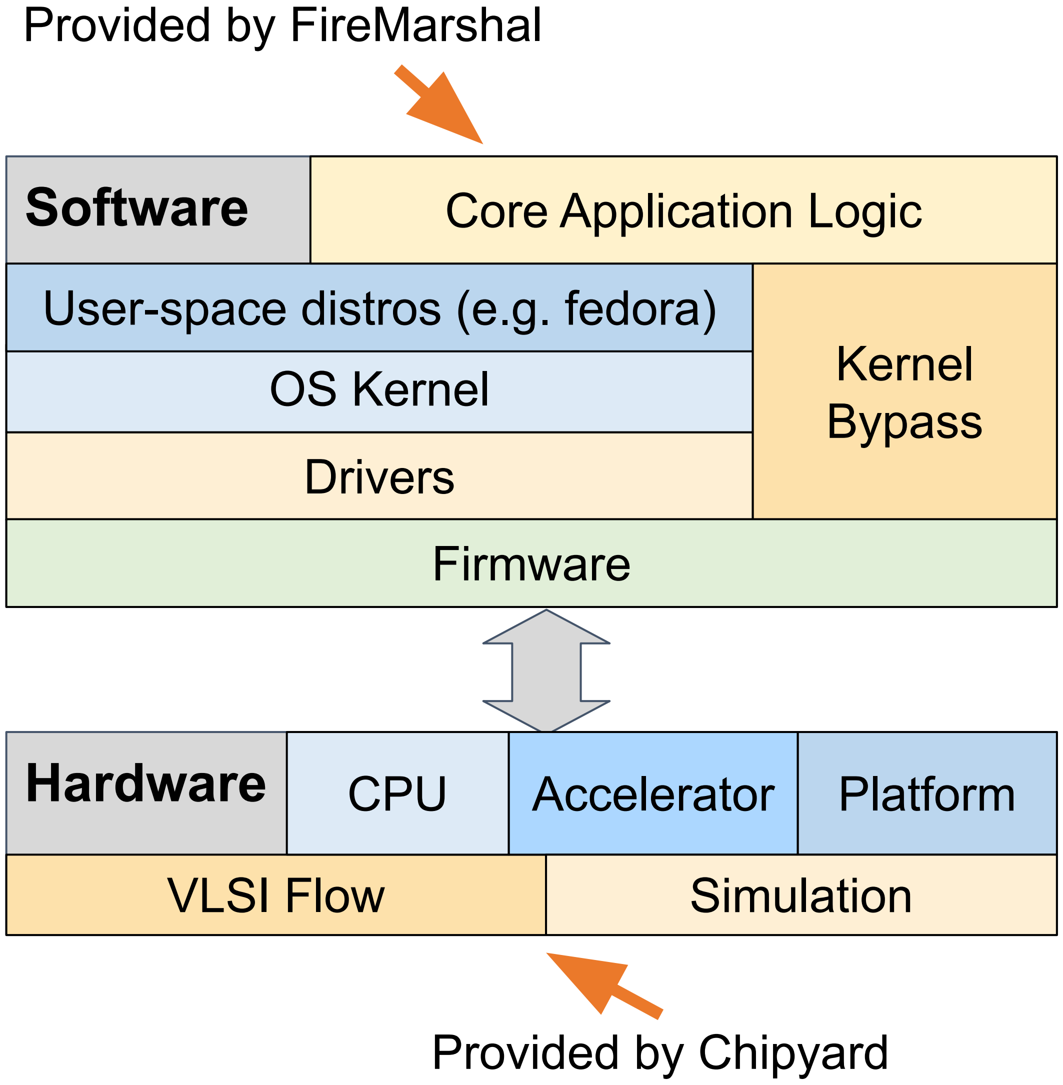
Tutorial Roadmap



Software Workload Management

Workload Management Tasks:

- Building Binaries and Filesystems
- Experiment Management
  - Inputs and outputs
  - Repeatable execution
  - Multiple levels of simulation
- Reproducibility and Reusability
  - Share workloads with the community



Hardware/Software Co-Design Flow

Write Spec

super_**c**ool_**a**ccelerator.md

Functional Model

Spike$_{sca}$

Write HW/SW

sca.chisel

sca.img/bin

Evaluate

FireSim

Requirements

Tool Challenges

Write Spec

super_cool_accelerator.md

Functional Model

$\text{Spike}_{\text{sca}}$

Write HW/SW

sca.chisel

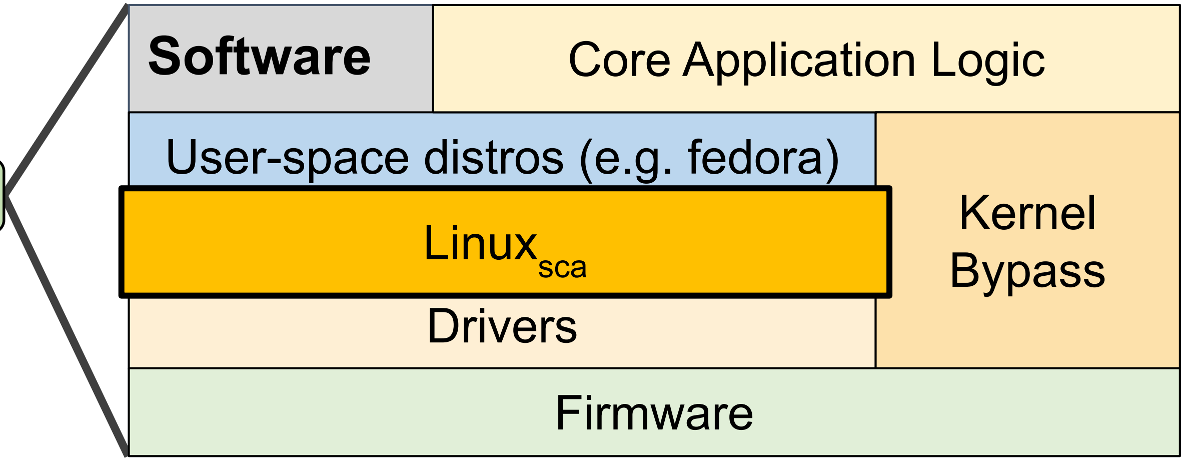
sca.img/bin

Evaluate

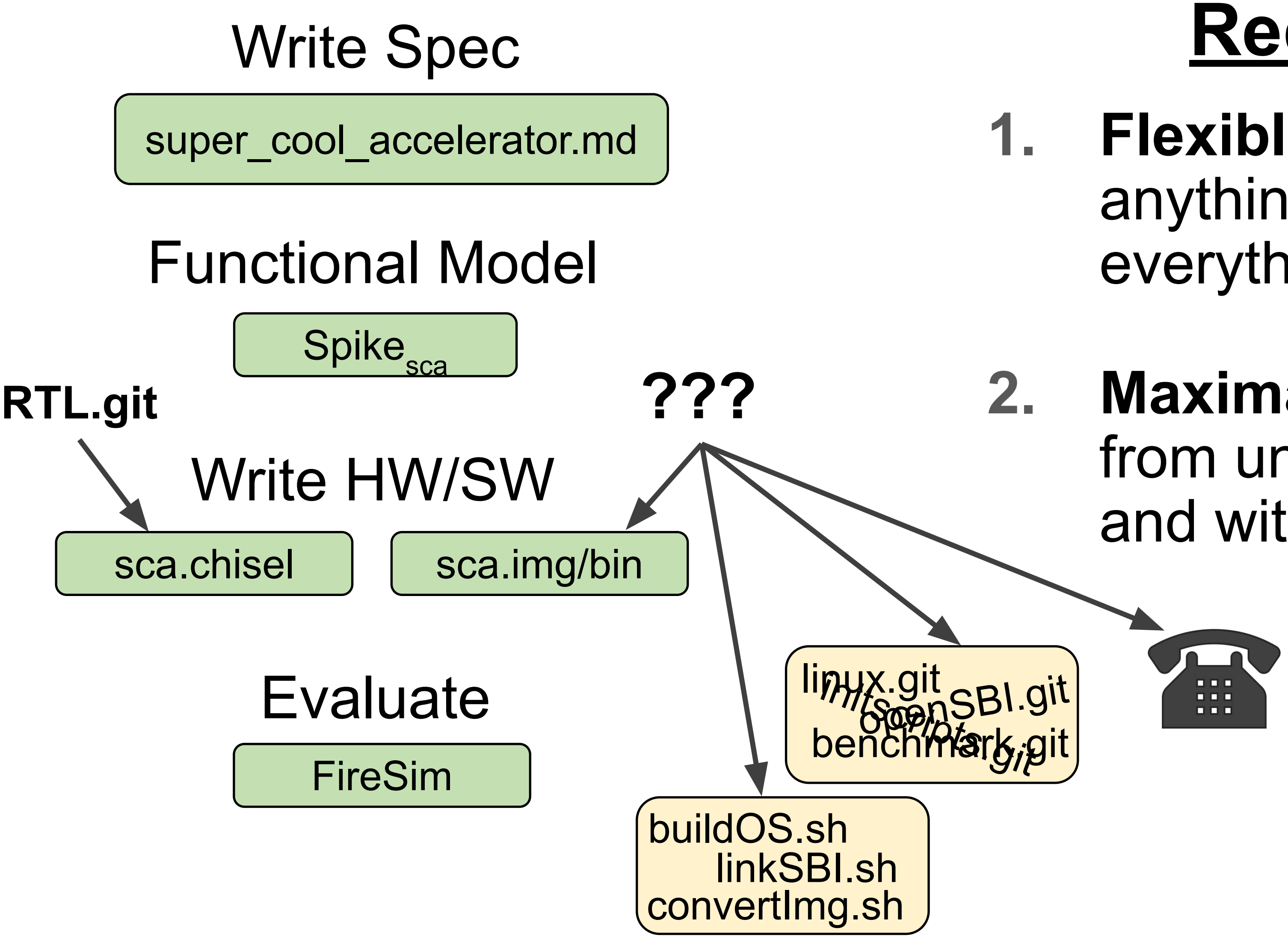
FireSim

Requirements

1. **Flexible Design:** Can change anything, without changing everything



Tool Challenges



Write Spec

super_cool_accelerator.md

Functional Model

Spike$_{sca}$

RTL.git

???

Write HW/SW

sca.chisel

sca.img/bin

Evaluate

FireSim

linux.git
initscripts.git
openSBI.git
benchmark.git

buildOS.sh
linkSBI.sh
convertImg.sh

☎

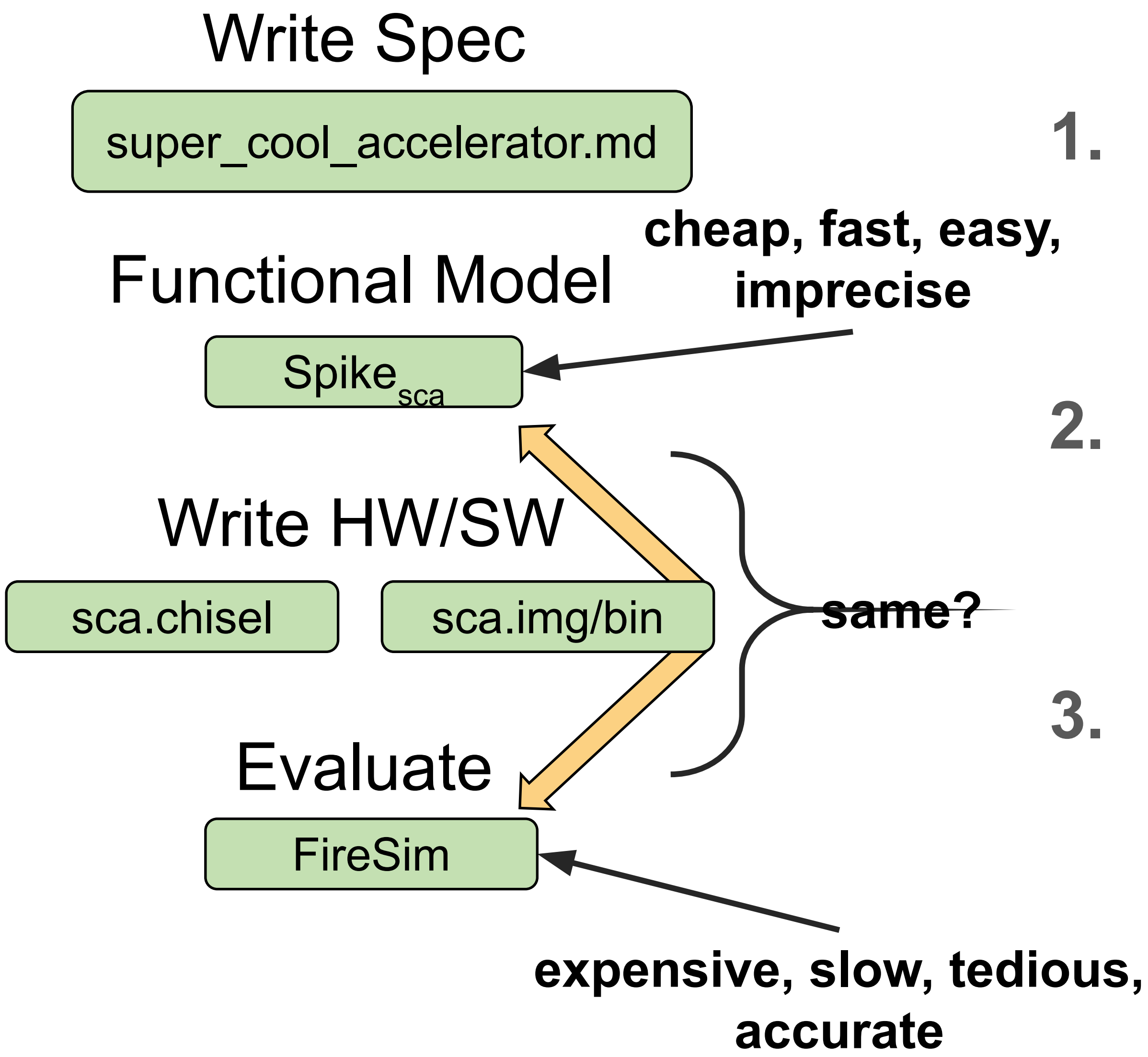
Alice: How big was your image?
Bob: lol, idk. Like 256? maybe?
Alice: I'll just try something

Requirements

1. **Flexible Design:** Can change anything, without changing everything

2. **Maximal Reuse:** Can rebuild from unambiguous description and without inside knowledge

Tool Challenges



Write Spec

super_cool_accelerator.md

Functional Model

Spike_{sca}

cheap, fast, easy, imprecise

Write HW/SW

sca.chisel

sca.img/bin

same?

Evaluate

FireSim

expensive, slow, tedious, accurate

Requirements

1. **Flexible Design:** Can change anything, without changing everything

2. **Maximal Reuse:** Can rebuild from unambiguous description and without inside knowledge

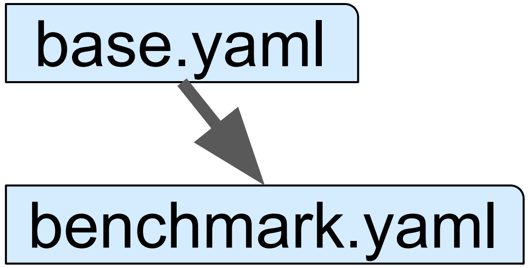
3. **Flexible Simulation:** Minimize SW differences across simulation levels

Where FireMarshal comes in…

FireMarshal Workflow

FireMarshal Workflow

Specify a “workload”



base.yaml

benchmark.yaml

What’s in a “workload”?

Sources

QEMU

OpenSBI

Inputs/Outputs

Overlays

```
/etc
  /init.d
    /S20foo
```

File Lists

```
[“/res.csv”,
 “/cfg.json]
```

Serial Console

```
# vda mounted
# run /init
experiment start
time: 50s
# poweroff
```

User Scripts

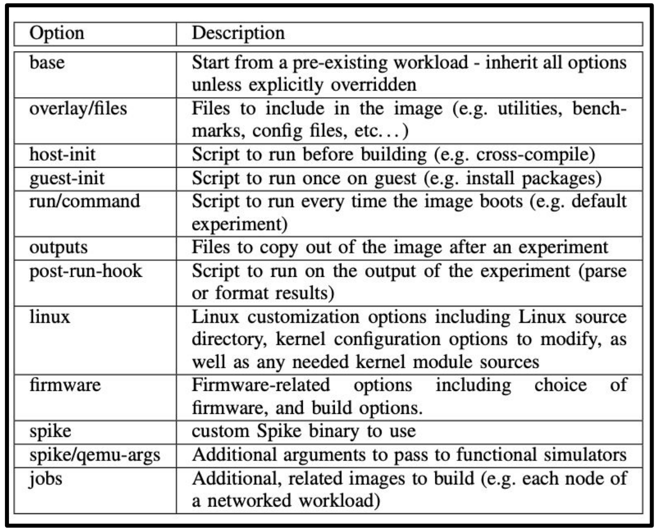
`host_setup.sh`

`target_setup.sh`

`on_boot.sh`

`after_run.sh`

Miscellaneous

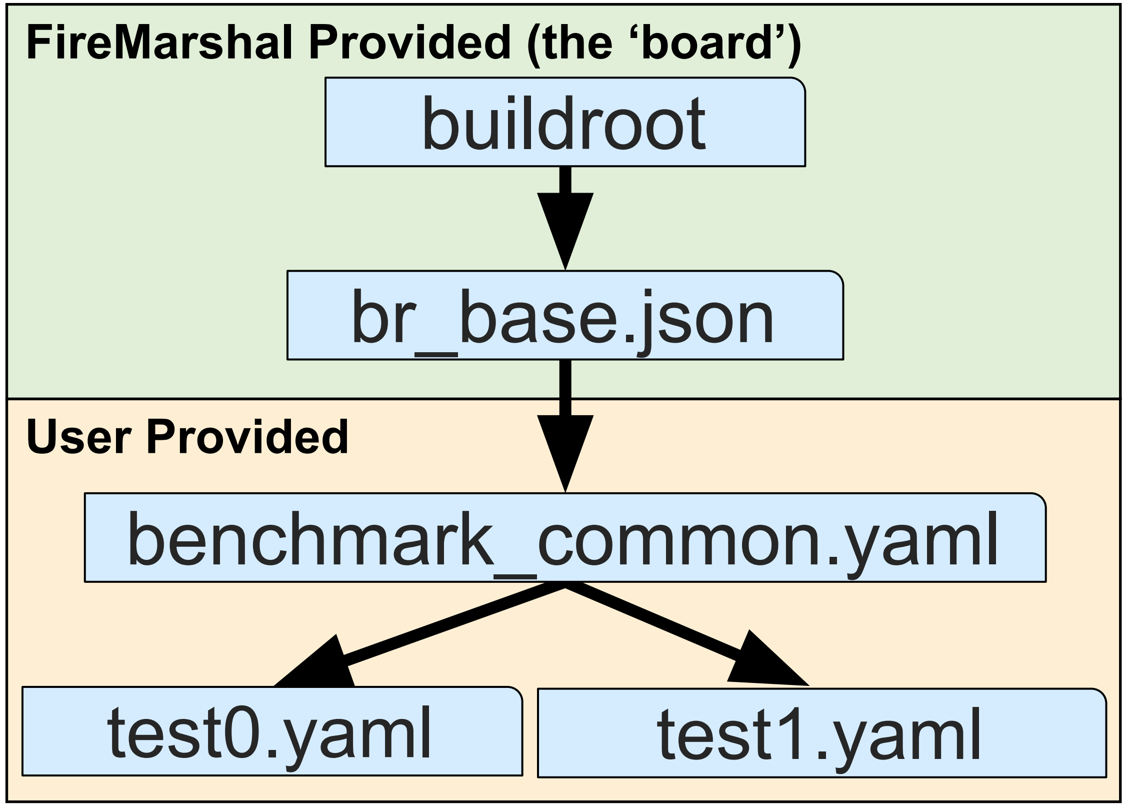


Option	Description
base	Start from a pre-existing workload - inherit all options unless explicitly overridden
overlay/files	Files to include in the image (e.g. utilities, benchmarks, config files, etc. . .)
host-init	Script to run before building (e.g. cross-compile)
guest-init	Script to run once on guest (e.g. install packages)
run/command	Script to run every time the image boots (e.g. default experiment)
outputs	Files to copy out of the image after an experiment
post-run-hook	Script to run on the output of the experiment (parse or format results)
linux	Linux customization options including Linux source directory, kernel configuration options to modify, as well as any needed kernel module sources
firmware	Firmware-related options including choice of firmware, and build options.
spike	custom Spike binary to use
spike/qemu-args	Additional arguments to pass to functional simulators
jobs	Additional, related images to build (e.g. each node of a networked workload)

. . .

Enabling Reuse

- **Inheritance:** Workloads are relative to a “base”
  - Inheritance rules vary by option
- **The “Board”:** Sane defaults for a hardware platform
- **User Hierarchies:** arbitrary inheritance tree


FireMarshal Provided (the ‘board’)

buildroot

br_base.json

User Provided

benchmark_common.yaml

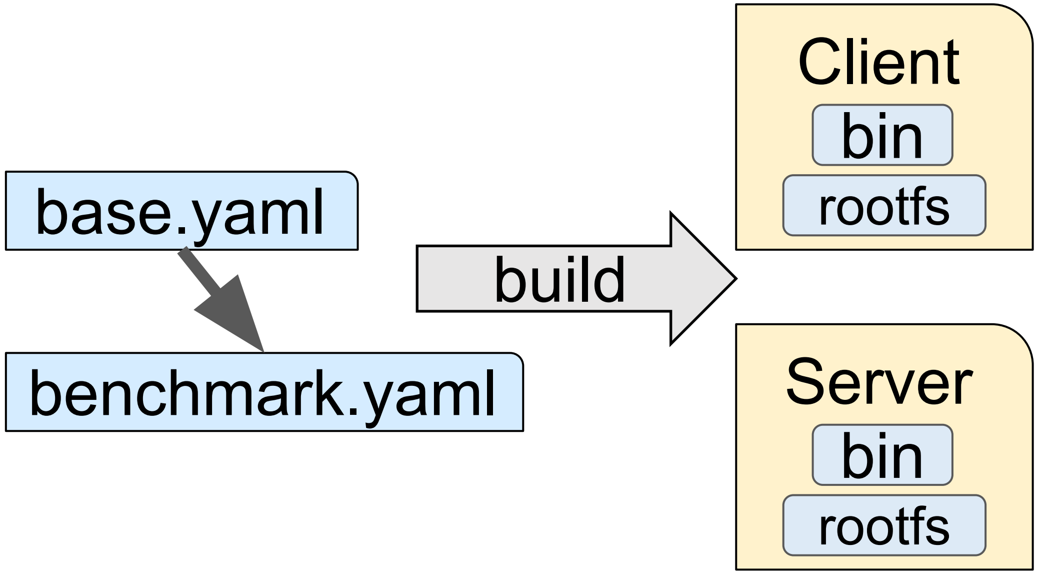
test0.yaml

test1.yaml

FireMarshal Workflow

Specify

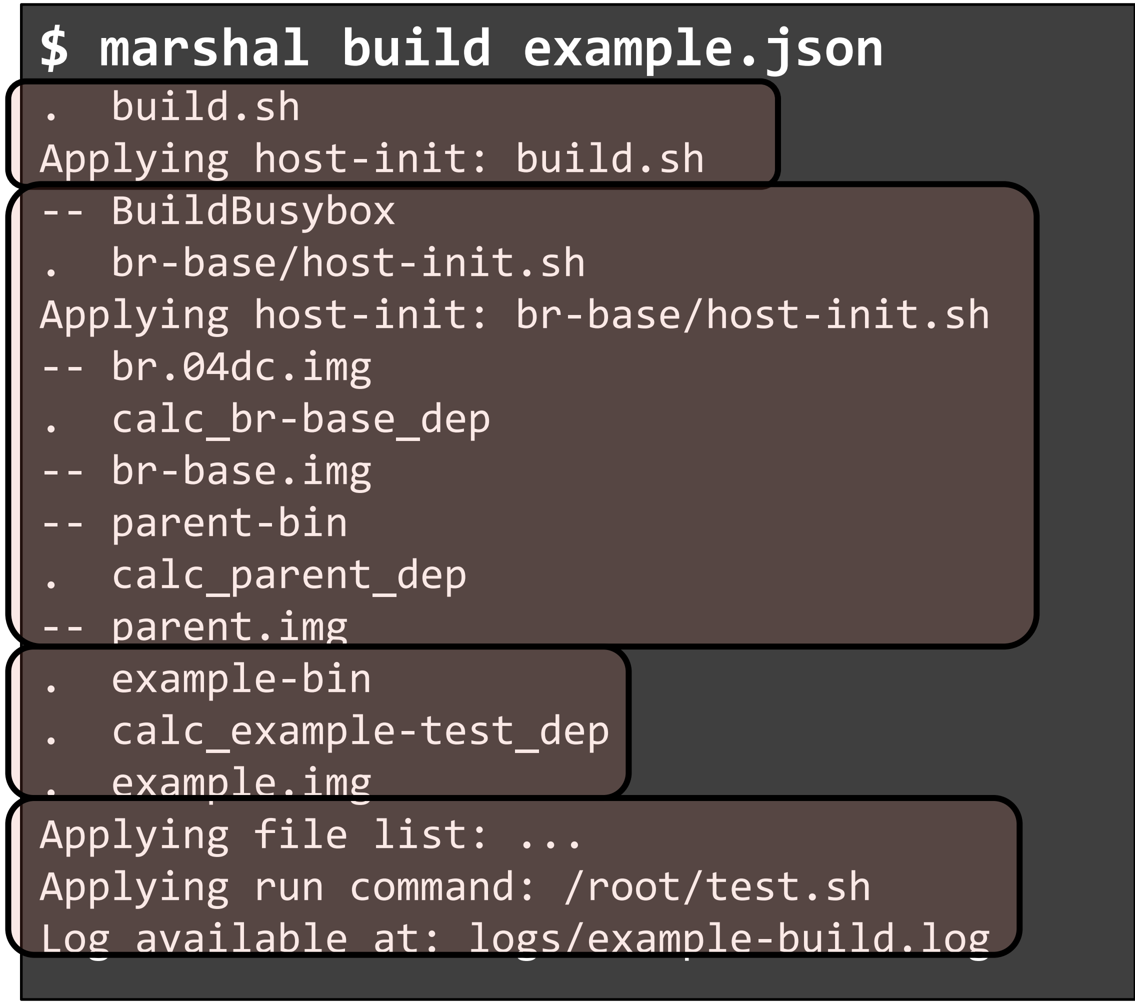
Build



Build Steps

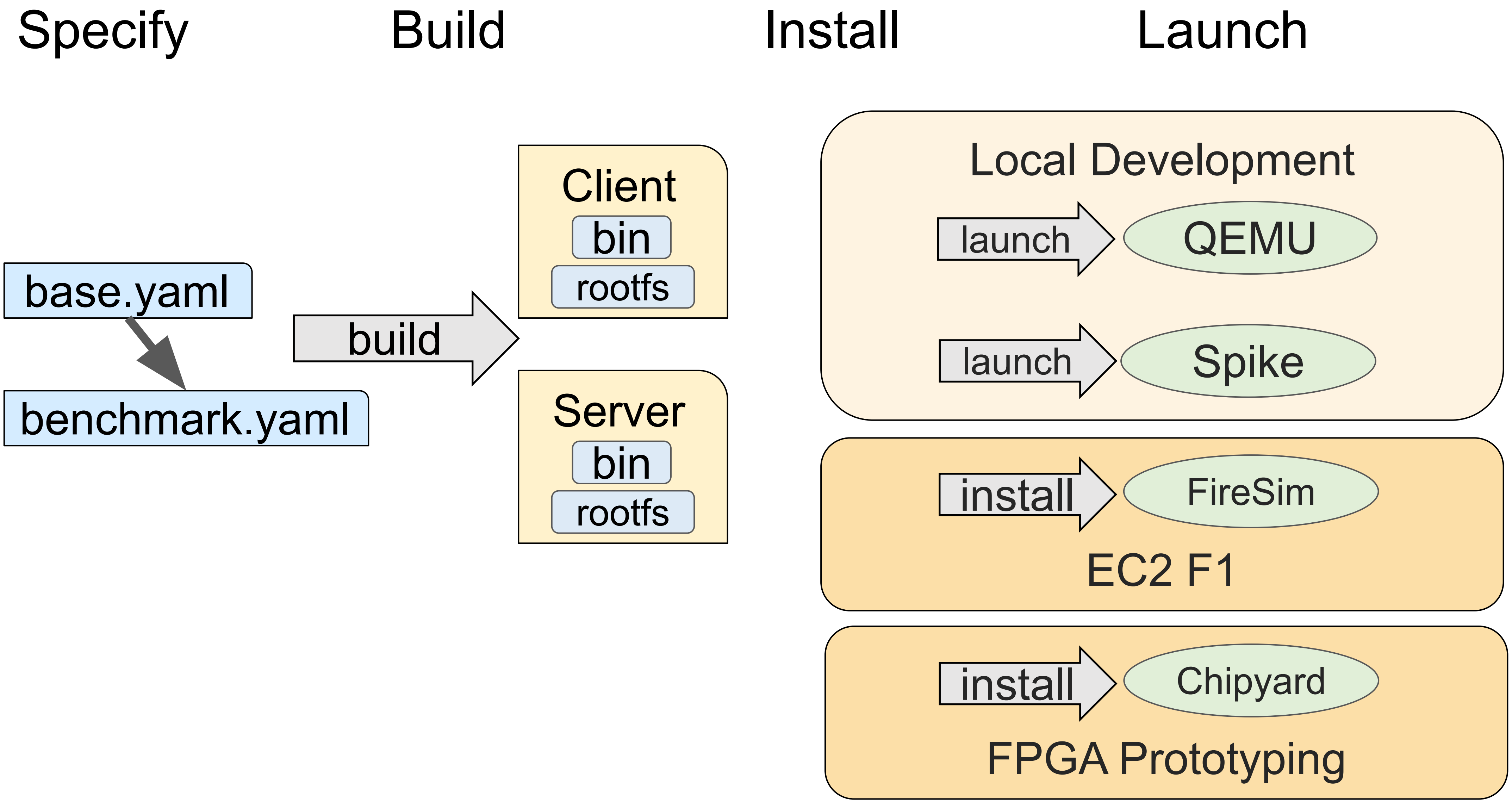
- Construct software artifacts
  - Make-like dependency tracking

- Steps
  1. Run workload setup script (host-init)
  2. Recursively construct parent images
  3. Compile Linux kernel and link firmware
  4. Copy parent image and modify as needed



```
$ marshal build example.json
.  build.sh
Applying host-init: build.sh
-- BuildBusybox
.  br-base/host-init.sh
Applying host-init: br-base/host-init.sh
-- br.04dc.img
.  calc_br-base_dep
-- br-base.img
-- parent-bin
.  calc_parent_dep
-- parent.img
.  example-bin
.  calc_example-test_dep
.  example.img
Applying file list: ...
Applying run command: /root/test.sh
Log available at: logs/example-build.log
```

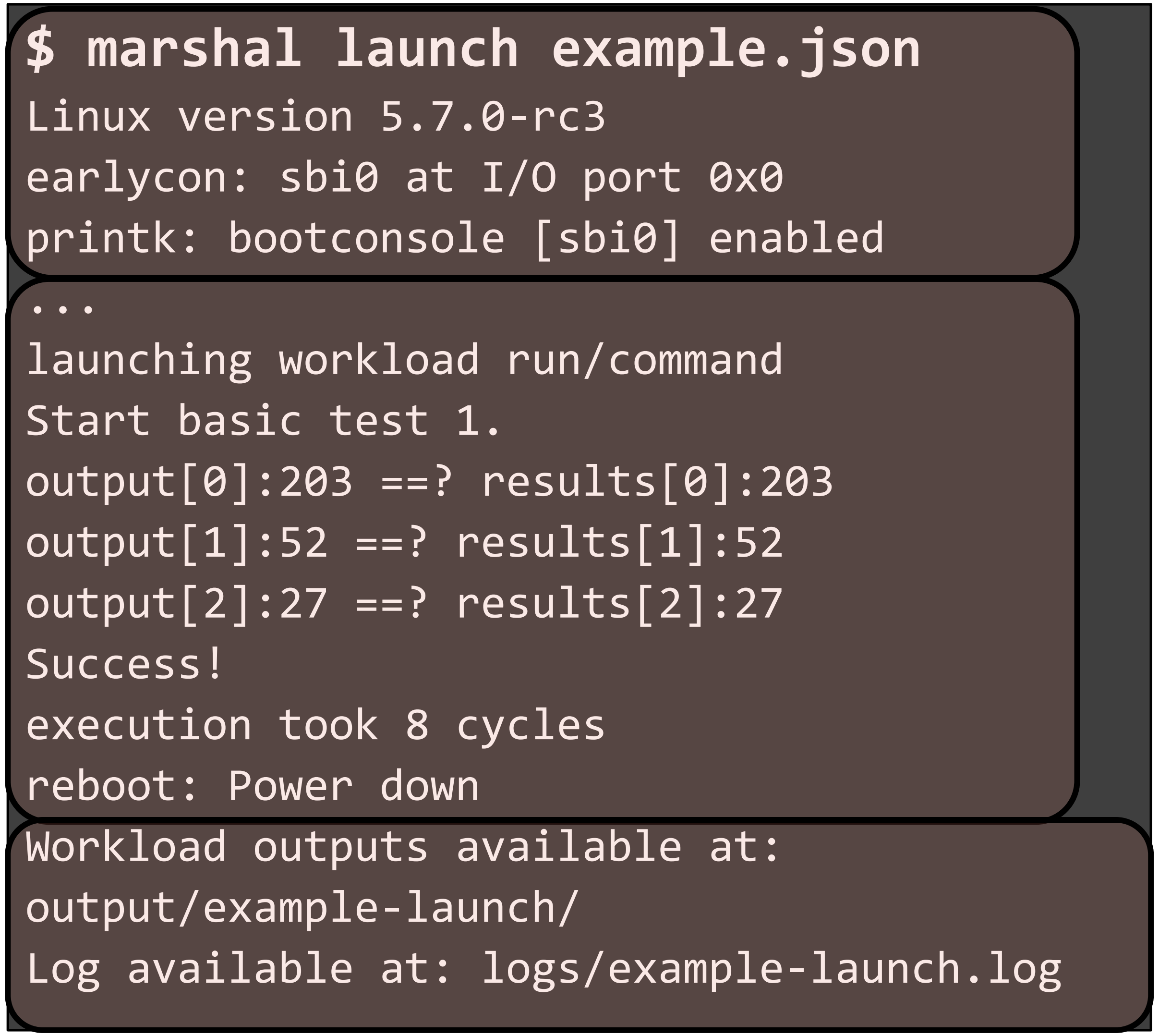
FireMarshal Workflow



Launch Workload in SW Sims

- Run workload in built-in functional simulation
  - QEMU
  - Spike
  - *Your tool here*

- Steps
  1. Boot Linux
  2. Automatically run benchmark
  3. Return serial and file outputs



```
$ marshal launch example.json
Linux version 5.7.0-rc3
earlycon: sbi0 at I/O port 0x0
printk: bootconsole [sbi0] enabled
...
launching workload run/command
Start basic test 1.
output[0]:203 ==? results[0]:203
output[1]:52 ==? results[1]:52
output[2]:27 ==? results[2]:27
Success!
execution took 8 cycles
reboot: Power down
Workload outputs available at:
output/example-launch/
Log available at: logs/example-launch.log
```

Install To Other Tools

- Hook to export workload to external tools
  - FireSim and Chipyard FPGA prototyping supported
  - Extensible to other simulation infrastructures

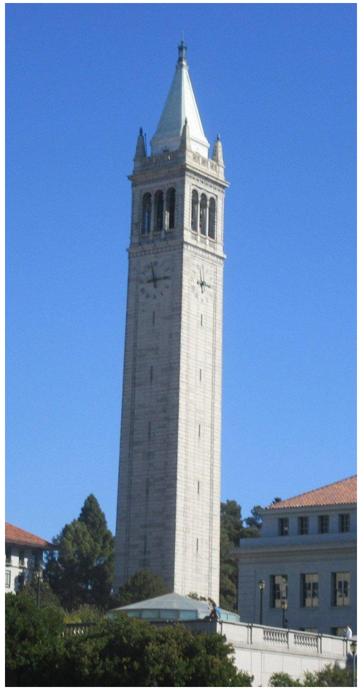
- Does *not* rebuild workload: The same workload runs on all platforms
  - Simplify debugging
  - Ensure consistency

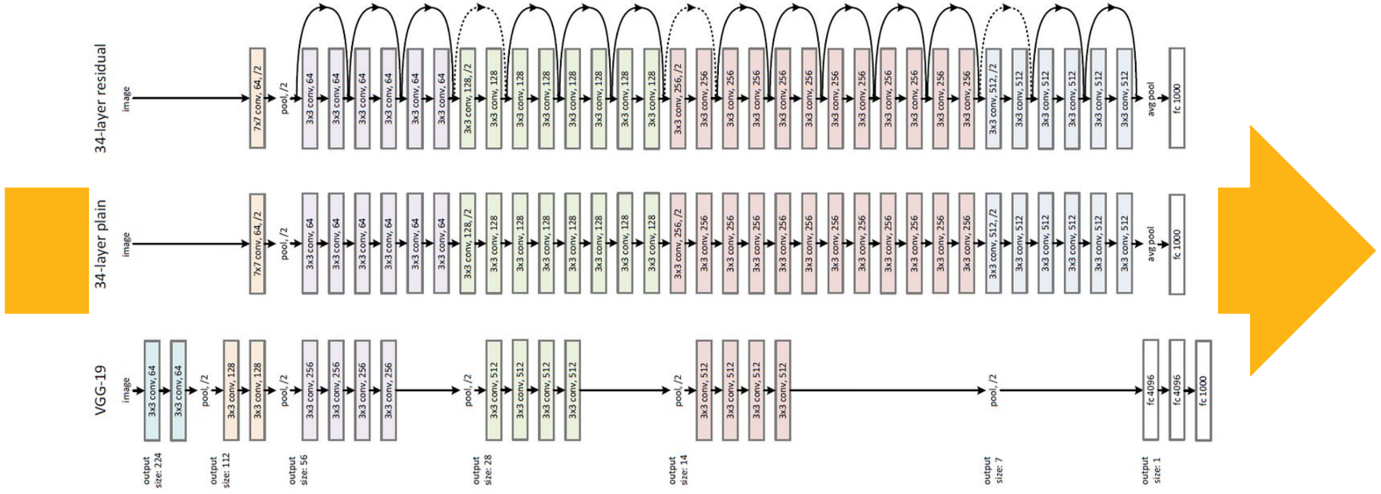
```
$ marshal install example.json

Workload installed to FireSim at
  firesim/example.json
Log available at:
  logs/example-install.log
```

Hands On: Gemmini Workloads

Running ResNet50 on a Gemmini SoC

- ResNet50 – deep learning model used for computer vision
- Gemmini – Berkeley’s machine learning accelerator
- Predicting four unique images





Berkeley Tower!

Workload Inheritance

FireMarshal-Provided

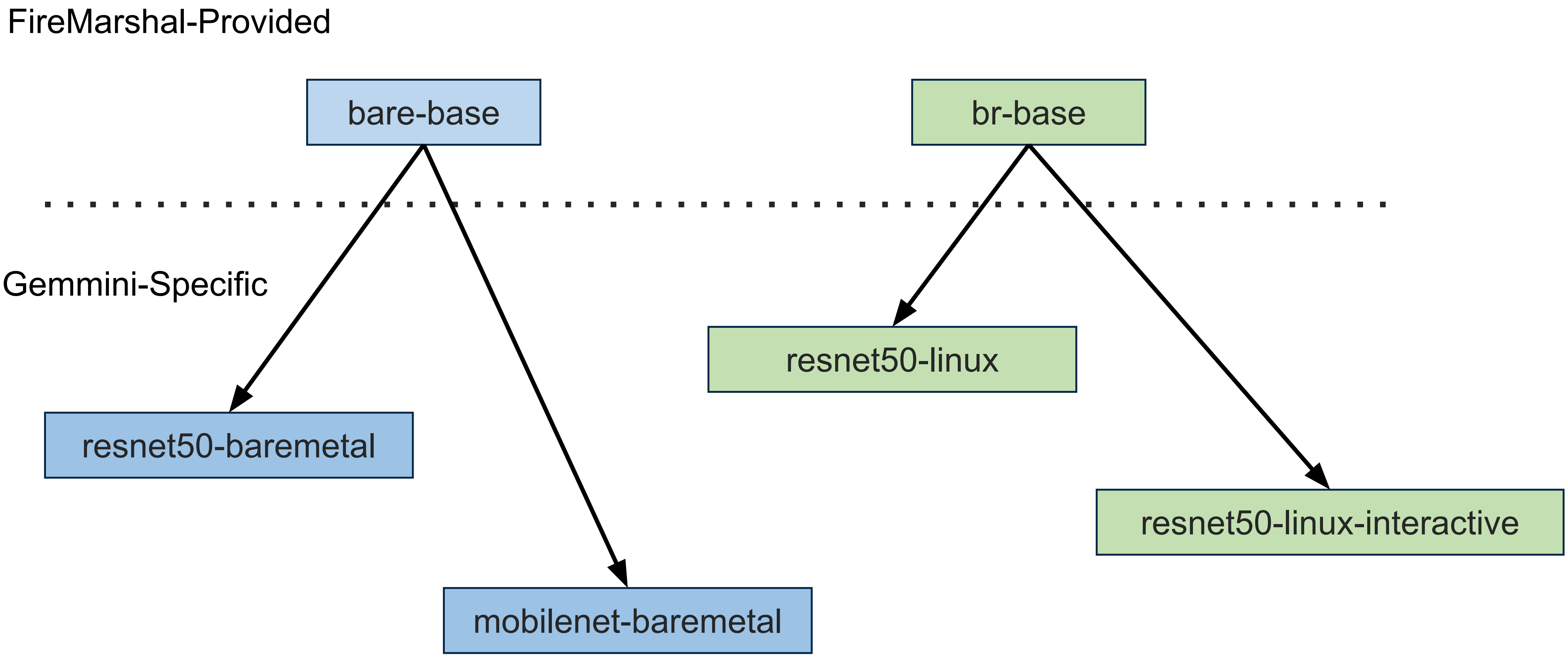
bare-base

br-base

Gemmini-Specific

- FireMarshal provides several “base” workloads to act as starting points for user workloads.
- Today, we will use:
  - “br-base”: Interactive Buildroot-based Linux workload.
  - “bare-base”: A trivial workload used for bare-metal experiments

Workload Inheritance



Interactive

Navigate to the Gemmini FireMarshal directory

```
$ cd $ACYDIR/software/tutorial
$ ls

$ cd marshal-configs
$ ls
```

```
chipyard-afternoon/
  software/
    tutorial/
      build.sh
      marshal-configs/
     *.yaml
```

Interactive

Build the baremetal test

```
$ marshal -v build resnet50-baremetal.yaml
```

This should be fast (~1m)!

Next run baremetal workload with the Gemmini functional model

```
$ marshal launch -s resnet50-baremetal.yaml
```

```
chipyard-afternoon/
  software/
    tutorial/
      build.sh
      marshal-configs/
        resnet50-*.yaml
```

resnet50-baremetal.yaml

Script to run on host first (build.sh cross-compiles the unit test)

```
{
  "name" : "resnet50-baremetal",
  "base" : "bare-base.json",
  "workdir" : "..",
  "host-init" : "build.sh",
  "bin" : "overlay/root/resnet50-baremetal",
  "spike-args" : "--extension=gemmini"
}
```

Hard-coded binary to use (produced by build.sh)

Custom Spike arguments (use Gemmini functional model)

Show the output

- Show the output

```
chipyard-afternoon/
  software/
    tutorial/
      build.sh
      marshal-configs/
        resnet50-*.yaml
```

Interactive

```
$ marshal -v -d build resnet50-linux.yaml
```

- This should take ~20m but is cached (~3m)!
- Next run the Linux simulation with the Gemmini functional model

```
$ marshal -d launch -s resnet50-linux.yaml
```

- While that is running…

```
chipyard-afternoon/
  software/
    tutorial/
      build.sh
      marshal-configs/
     *.yaml
```

resnet50-linux.yaml

Script to run on host first (build.sh cross-compiles the unit test)

```
{
  "name" : "resnet50-linux",
  "base" : "br-base.json",
  "workdir" : "..",
  "host-init" : "build.sh",
  "overlay" : "overlay",
  "command" : "/root/resnet50-linux"
  "spike-args" : "--extension=gemmini"
}
```

Hard-coded binary to use after Linux boots (produced by build.sh)

Custom Spike arguments (use Gemmini functional model)

Running ResNet50 on Linux

- We’ll get back to this once complete…

```
chipyard-afternoon/
  software/
    tutorial/
      build.sh
      marshal-configs/
     *.yaml
```

More Complex Use-Cases

Multi-Node Workloads (“jobs”)

job-example.yaml

```
{
  ”name” : “job-example”,
  “base” : “br-base.json”,
  “jobs” : [
    { “name” : “node0”,
      “command” : “ping -c 1 172.16.0.3”,
    },
    { “name” : “node1”,
      “command” : “ping -c 1 172.16.0.2”,
    }
  ]
}
```

- Each job runs on a single node in multi-node simulations.
- Described the same as any workload
  - implicitly ‘base’d on the enclosing workload
- Runs in parallel in SW simulation
  - FireSim already supported a network

Native Initialization (“guest–init”)

guest-init-example.yaml

```
{
  “name” : “guest-init-example”,
  “base” : “fedora-base.json”,
  “guest-init” : “init.sh”
}
```

init.sh

```
#!/bin/bash
yum install –y blas python3 …

cd cafe2_src/
make
```

- “guest-init” script is run once on the guest in build
  - Run in QEMU
  - Internet access available
- Useful for
  - Installing packages
  - Natively compiling benchmarks

Results Hooks (“post-run-hook”)

results-example.yaml

```
{
  ”name” : “results-example”,
  “base” : “mytest.yaml”,
  “outputs” : [“/root/res.csv”],
  “post-run-hook” : “results.py”
}
```

“post-run-hook” executed on the host after every run

- Good for post-processing of more complex experiments

results.py

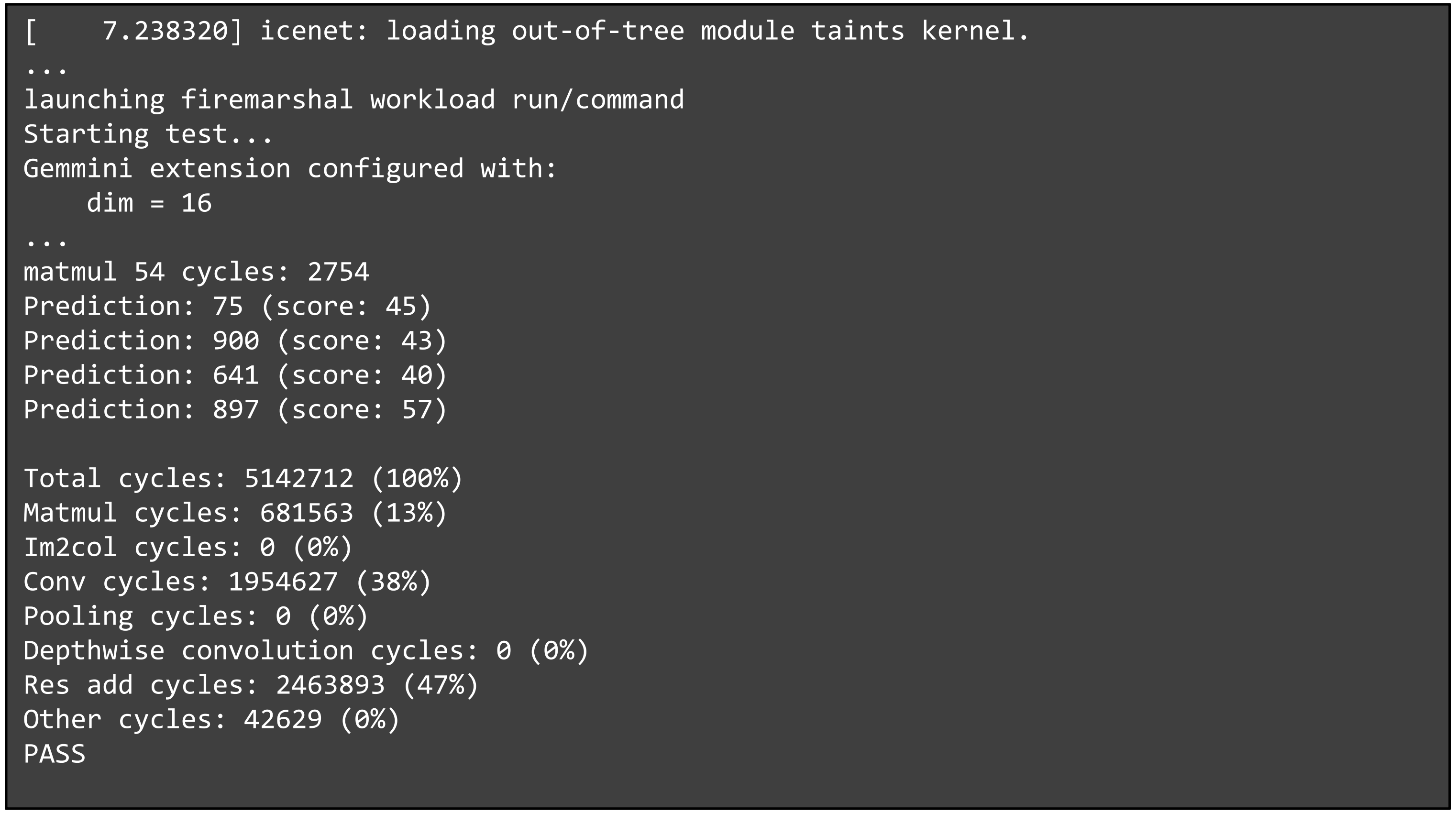
```
#!/usr/bin/env python
from pathlib import Path
import csv

resultPath = Path(sys.argv[1]) /
  ‘results-example’ / ‘res.csv’

processResult(resultPath)
```

Path to the results directory passed to the script

Do anything you want with the results. For example, copy to a known location, or sanity check

Running ResNet50 on Linux

- By now you should see output like…



```
[    7.238320] icenet: loading out-of-tree module taints kernel.
...
launching firemarshal workload run/command
Starting test...
Gemmini extension configured with:
    dim = 16
...
matmul 54 cycles: 2754
Prediction: 75 (score: 45)
Prediction: 900 (score: 43)
Prediction: 641 (score: 40)
Prediction: 897 (score: 57)

Total cycles: 5142712 (100%)
Matmul cycles: 681563 (13%)
Im2col cycles: 0 (0%)
Conv cycles: 1954627 (38%)
Pooling cycles: 0 (0%)
Depthwise convolution cycles: 0 (0%)
Res add cycles: 2463893 (47%)
Other cycles: 42629 (0%)
PASS
```

```
chipyard-afternoon/
  software/
    tutorial/
      build.sh
      marshal-configs/
     *.yaml
```

Exporting to FireSim

Interactive

Install/build the baremetal MobileNet workload to FireSim

```
$ marshal build mobilenet-baremetal.yaml
$ marshal install mobilenet-baremetal.yaml
```

View the installed FireSim-specific collateral

```
$ cd $FDIR/deploy/workloads
$ cat mobilenet-baremetal.json
```

```
chipyard-afternoon/
  software/
    tutorial/
      build.sh
      marshal-configs/
     *.yaml
```

Prefetching for future sections

Many many more features!

https://firemarshal.readthedocs.io/en/stable/