FireMarshal:

Software Workload Management

Nathan Pemberton

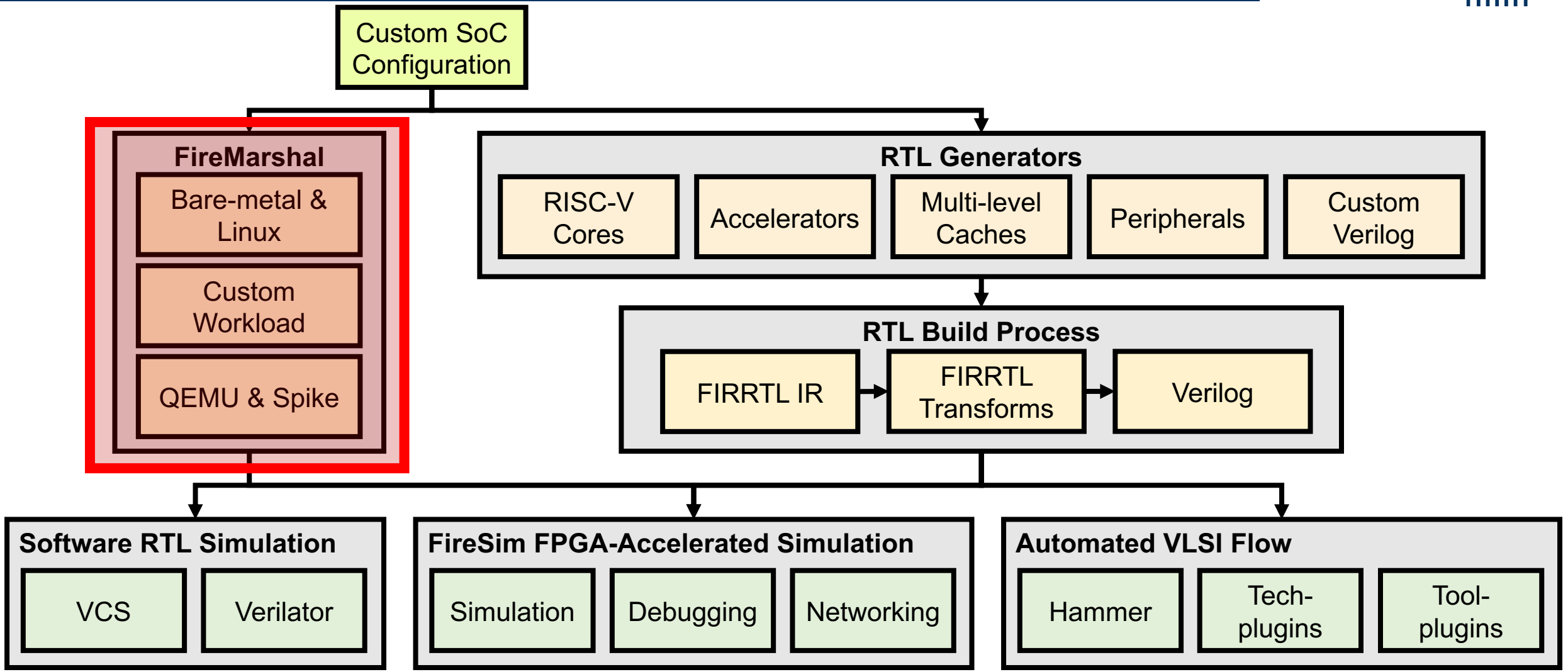
UC Berkeley

nathanp@berkeley.edu

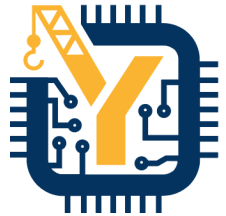
CHIPYARD

Berkeley Architecture Research

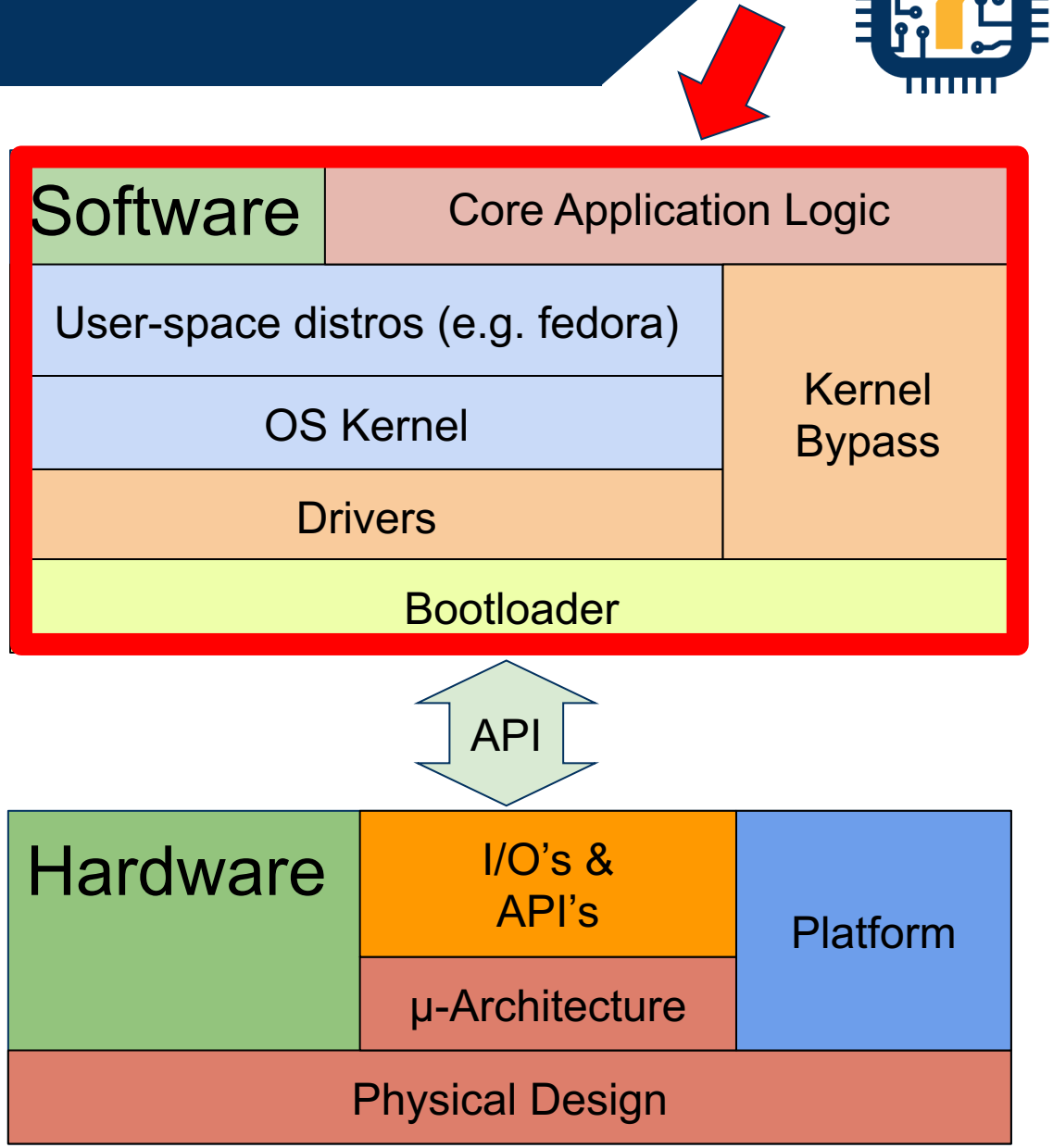
Tutorial Roadmap



FireMarshal Goals

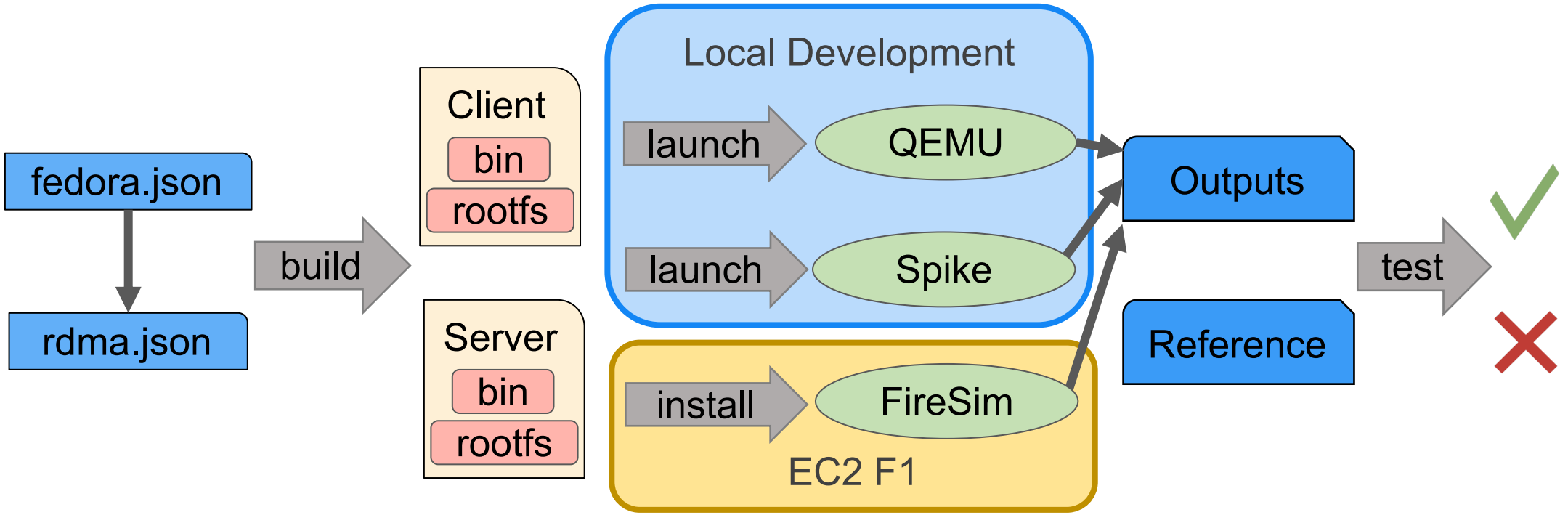


- Consistent Environments
  - Problem: Everyone working off slightly different versions of platform/OS/etc.
- Re-Usable Workloads
  - Problem: Tribal knowledge and non-reproducible results
  - No standard way to represent workloads
  - No version control for integration
- Decoupled Development
  - Easy integration from SW models (like spike or qemu) to real RTL (FireSim or actual chips)

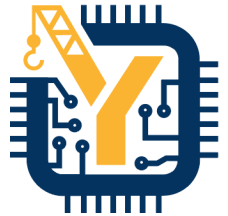


Berkeley Architecture Research

FireMarshal Overview
FireSim Workload Management

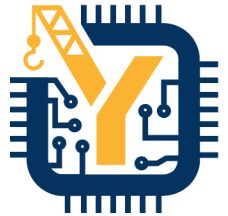


- Generate workload from machine-readable description
  - A collection of boot binaries and disk images that run together
- Run generated workloads locally on SW simulators
- Install to FireSim to run FPGA-accelerated simulation
- Automatically test and post-process results



Sha3 Example Workloads

FireMarshal Tutorial Outline



Workloads:

- Bare Metal Unit Tests
  - sha3-bare-sw
  - sha3-bare-rocc
- Linux-Based Unit Test
  - sha3-linux
  - sha3-linux-test
- Linux-Based Benchmark
  - sha3-linux-jtr
  - sha3-linux-jtr-crack

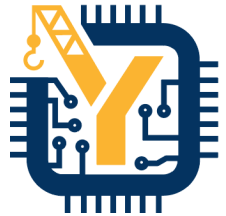
Provided For You:

- Sha3 functional model (Spike)
- RoCC-Enabled Linux Kernel

Everything defined in its own repository:

sha3-workload.git

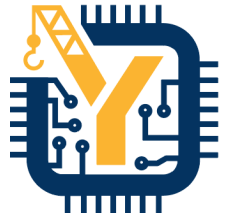
Example Workload:
Sha3 Workload Directory



```
centos@ip-192-168-2-41.us-west-2.compute.internal:~/chipyard/software/firemarshal$ ls workloads/
br-base       dummy.json        fedora-base       sha3-bare-rocc.json  sha3-linux-jtr-crack.json  sha3-linux-test.json
br-base.json  example-fed       fedora-base.json  sha3-bare-sw.json    sha3-linux-jtr.json
dummy         example-fed.json  sha3              sha3-linux.json      sha3-linux-jtr-test.json
centos@ip-192-168-2-41.us-west-2.compute.internal:~/chipyard/software/firemarshal$ ls workloads/sha3
benchmarks  install.sh  marshal-configs  riscv-isa-sim  spike-local  test_local.sh
build.sh    jtr         README.md        riscv-linux    spike-sha3   test-reference
centos@ip-192-168-2-41.us-west-2.compute.internal:~/chipyard/software/firemarshal$ ls workloads/sha3/benchmarks/
bare  common.mk  linux  src
centos@ip-192-168-2-41.us-west-2.compute.internal:~/chipyard/software/firemarshal$ ls workloads/sha3/spike-sha3/
aclocal.m4   configure     dummy_rocc   README.md              riscv-riscv.pc.in      riscv-spike_main.pc.in  sha3        tests
build        configure.ac  LICENSE      riscv                  riscv-sha3.pc.in       riscv-spike.pc.in       softfloat
config.h.in  debug_rom     Makefile.in  riscv-dummy_rocc.pc.in  riscv-softfloat.pc.in  scripts                 spike_main
centos@ip-192-168-2-41.us-west-2.compute.internal:~/chipyard/software/firemarshal$
```

```
$ cd ~/chipyard-afternoon/software/firemarshal
$ ls workloads/
$ ls workloads/sha3/
```

Example Workload:
Sha3 Bare-Metal Unit Test



sha3-bare-rocc.json

```
{
"name" : "sha3-bare-rocc",
"workdir" : "sha3",
"base" : "bare",
"host-init" : "build.sh",
"bin" : "benchmarks/bare/sha3-rocc.riscv",
"spike" : "spike-local/bin/spike",
"spike-args" : "--extension=sha3"
}
```

Specifies any parent workload to inherit settings from ('bare' is a minimal workload that runs hard-coded RISCV binaries)

Script to run when building this workload (build.sh cross-compiles the unit test)

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

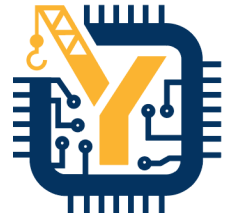
Golden-model sw simulator to use when launching this workload

Example Workload:
Sha3 Bare-Metal Unit Test

DEMO

```
$ cd ~/chipyard-afternoon/software/firemarshal
$ ./marshal build workloads/sha3-bare-rocc.json
$ ./marshal launch -s workloads/sha3-bare-rocc.json
$ ./marshal test -s workloads/sha3-bare-rocc.json
```

Example Workload:
SHA3 on Linux



sha3-linux.json

```
{
"name" : "sha3-linux",
"base" : "br-base.json",
"workdir" : "sha3",
"host-init" : "build.sh",
"files" : [
  ["bmarks/sha3-sw.rv", "/root/sha3-sw"],
  ["bmarks/sha3-rocc.rv","/root/sha3-rocc"],
],
"linux-src" : "riscv-linux",
"spike" : "spike-local/bin/spike",
"spike-args" : "--extension=sha3"
}
```

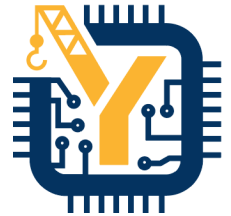
Basic Buildroot-based Linux distribution (provided by Marshal)

Run by Marshal at build time (cross-compiles the Linux benchmarks)

Files to copy into the guest root filesystem (the pre-compiled benchmarks in this case)

Optional custom Linux source to compile (needed in this case to enable rocc)

Example Workload:
Linux-based Unit Test



sha3-linux-test.json

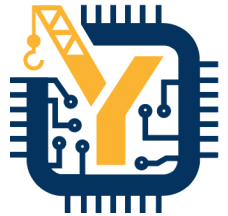
```
{
  "name" : "sha3-linux-test",
  "base" : "sha3-linux.json",
  "workdir" : "sha3",
  "command" : "/root/sha3-rocc"
  "testing" : {
    "refDir" : "goldenOutput/"
  }
}
```

Inherit everything we did for the basic sha3 workload, no need to repeat ourselves.

Run by the guest every time it boots. Target will shutdown after running the command.

Known-good output. Marshal will compare the run output against this when you test the workload

Example Workload:
Linux-based Unit Test



```
(fs) [xarc@xarc0 firesim-software]$ ./marshal -i build workloads/sha3-linux-test.json
To check on progress, either call marshal with '-v' or see the live output at:
/data/repos/firesim-software/logs/sha3-linux-test-build-2019-08-30--19-47-30-PVJTFAKVJFUR3DXS.log
.  /data/repos/firesim-software/images/sha3-linux-test-bin
-- /data/repos/firesim-software/wlutil/br/buildroot/output/images/rootfs.ext2
-- /data/repos/firesim-software/images/br-base.img
-- /data/repos/firesim-software/images/sha3-linux.img
.  /data/repos/firesim-software/images/sha3-linux-test.ima
```

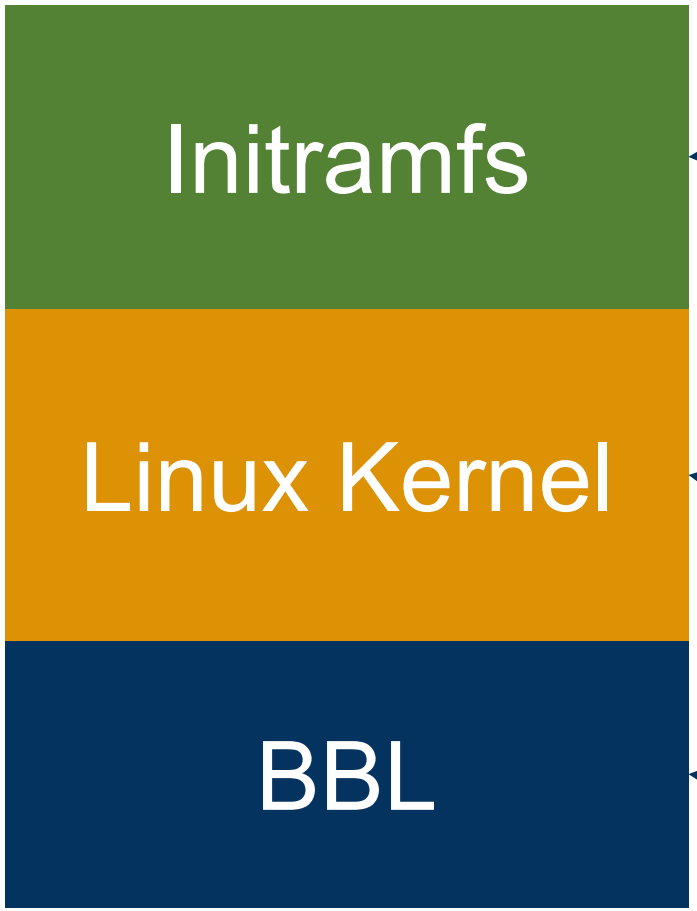
DEMO

```
$ cd ~/chipyard-afternoon/software/firemarshal
$ ./marshal -dv test -s workloads/sha3-linux-test.json
```

```
Running: /data/repos/firesim-software/workloads/sha3-linux-test.json
Success - output available in /data/repos/firesim-software/runOutput/sha3-linux-test-test-2019-08-30--19-54-04-8K300SSPEQUI19D2
Test Passed

Log available at: /data/repos/firesim-software/logs/sha3-linux-test-test-2019-08-30--19-54-04-8K300SSPEQUI19D2.log
SUCCESS: All Tests Passed (0 tests skipped)
(fs) [xarc@xarc0 firesim-software]$
```

Linux Build Internals:
What's in a binary?



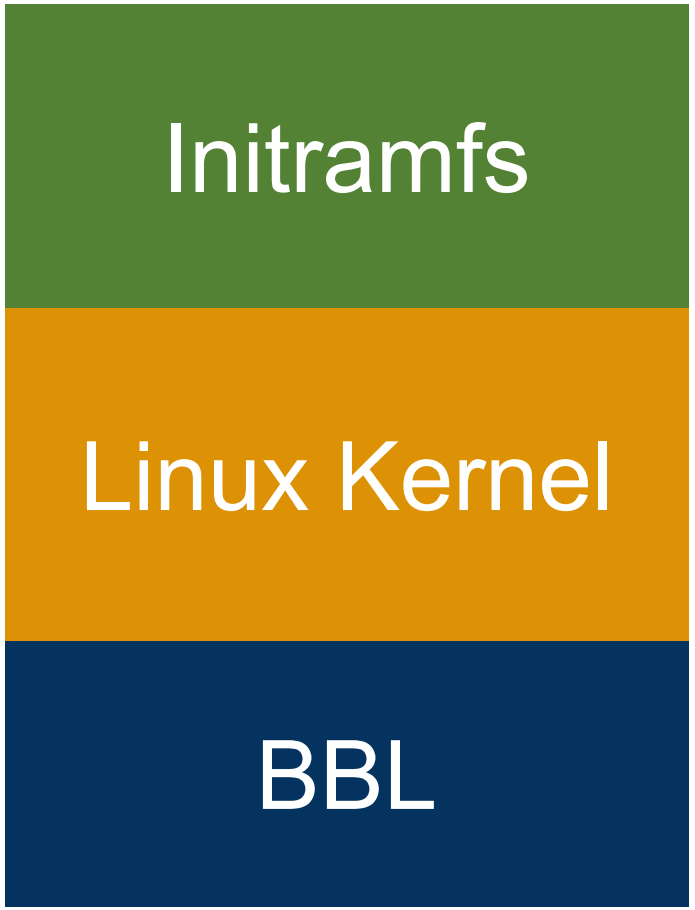
Contains platform drivers and a minimal busybox environment. Linked directly into the kernel

Upstream* Linux Kernel:
Compiled per-workload based on configuration

**Has some temporary patches for rocket chip*

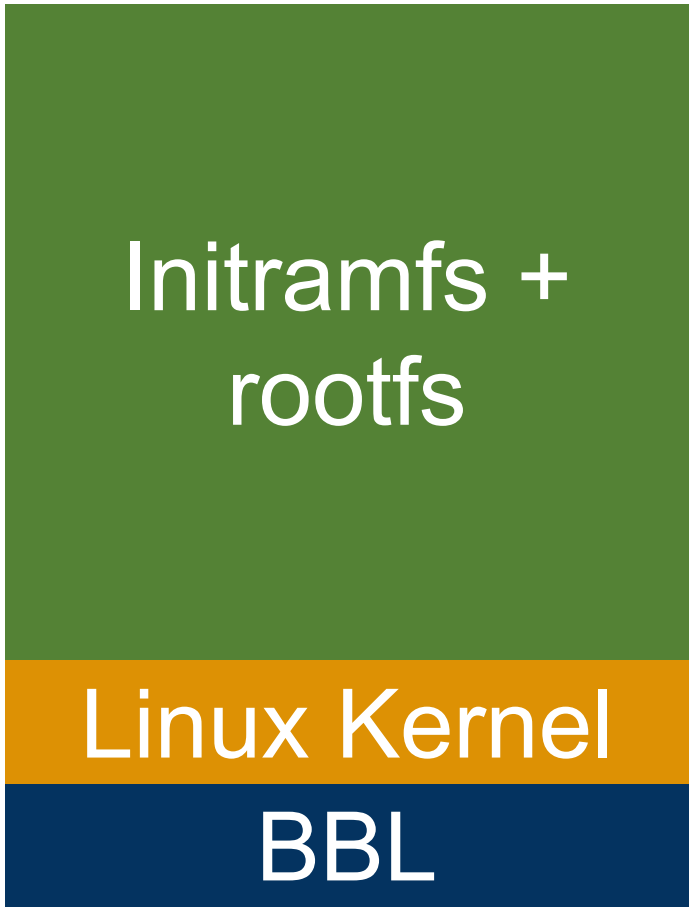
Berkeley Boot Loader:
Compiled into binary for now. Derived from the PK package

Linux Build Internals:
Diskless Designs



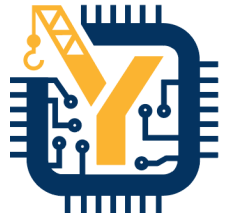
- Problem: Not every platform has a working disk device (e.g. spike)
- Solution: Compile the whole rootfs into the binary image!
  - ‘./marshal –nodisk …’

Linux Build Internals:
Diskless Designs



- Problem: Not every platform has a working disk device (e.g. spike)
- Solution: Compile the whole rootfs into the binary image!
  - ‘./marshal –nodisk …’

Example Workload:
Linux-based Benchmark – John the Ripper



sha3-linux-jtr.json

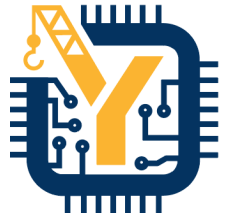
```
{
  "name" : "sha3-linux-jtr",
  "base" : "sha3-linux.json",
  "workdir" : "sha3",
  "host-init" : "jtr/build.sh",
  "overlay" : "jtr/overlay",
}
```

Inherit from sha3-linux again. Only need to specify that stuff once.

Run on host exactly once (cross-compiles benchmark).

John The Ripper must be installed to work correctly. The overlay allows us to specify a complex directory structure.

Example Workload:
Linux-based Benchmark – John the Ripper



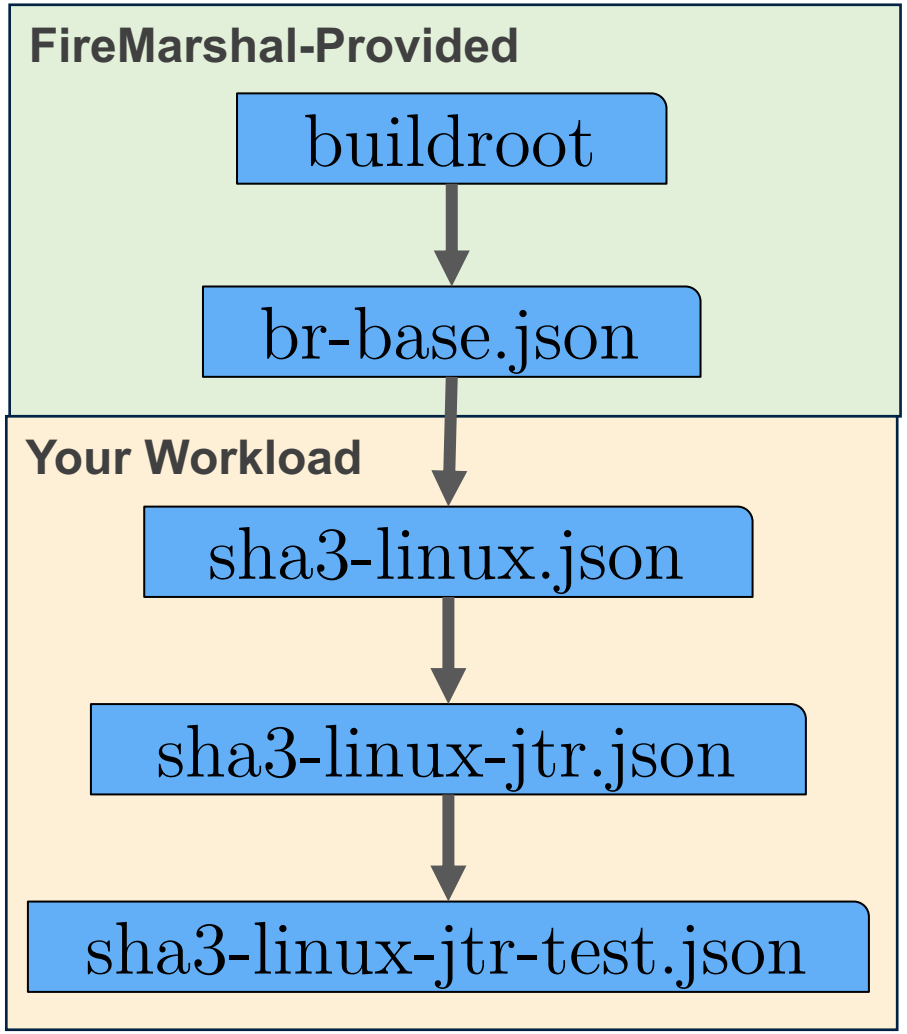
```
$ cd ~/chipyard-afternoon/software/firemarshal
$ ./marshal -d build workloads/sha3-linux-jtr.json
$ ./marshal -d launch -s workloads/sha3-linux-jtr.json
```

```
Applying run script: /home/centos/chipyard/sims/firesim/sw/firesim-software/wlutil/null_run.sh
```

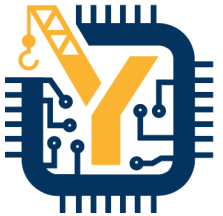
In the target:

```
user: root
password: firesim
$ cd sha3
$ john --format=Raw-SHA3-256-rocc short.txt
$ poweroff -f
```

Linux Build Internals:
Inheriting Workloads

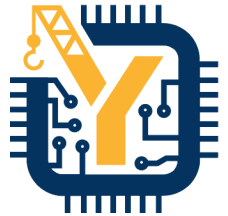


- Marshal avoids repeating work by inheriting from parents
- Inheritance Process (recursively)
  - Build parent completely
  - Copy parent rootfs
  - Apply child rules (e.g. overlays, guest-init, etc)
- GNU Make style dependency checking
  - FireMarshal only rebuilds if parents are out of date



More Complex Use-Cases

Multi-Node Workloads
(“jobs”)

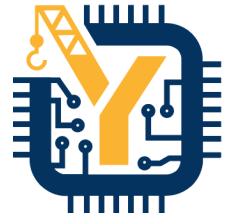


job-example.json

```
{
  "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
- Can run one at a time in SW simulation.
  - Must use FireSim to use the network

Native Initialization
(“guest-init”)



guest-init-example.json

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

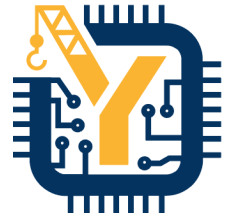
init.sh

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

cd cafe2_src/
make
```

- “guest-init” script is run once on the guest during build
  - Run in Qemu
  - Can access internet
- Useful for installing packages and/or natively compiling benchmarks

Automatic Results Processing
(“post-run-hook”)



results-example.json

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

“outputs” specifies files to copy from guest image after a run

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

- Good for post-processing of more complex experiments

results.py

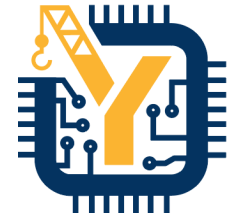
```
#!/usr/bin/env python
from pathlib import Path
import csv

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

processResult(resultPath)
```

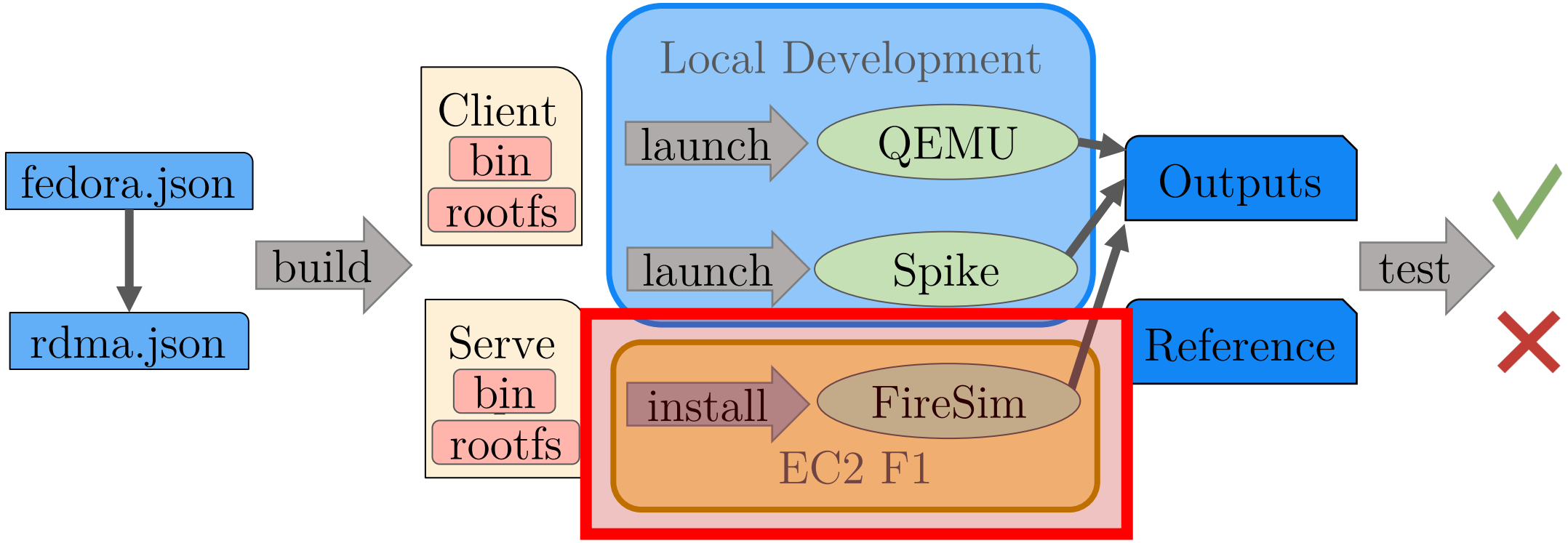
Path to the results directory passed to the script

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



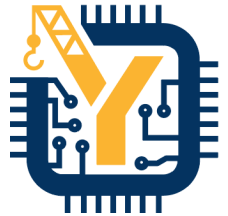
Running Workloads on FireSim

FireMarshal Overview
FireSim Install



- Generates FireSim-native workload configuration from FireMarshal
- After running install, you can use FireSim to launch the workload on the real RTL
  - Note: unlike functional simulation, FireSim makes a copy of the rootfs before running.

Installing Workloads to FireSim



```
$ cd ~/chipyard-afternoon/software/firemarshal
$ ./marshal install workloads/sha3*.json
$ cd ~/chipyard-afternoon/sims/firesim/deploy/
$ cat workloads/sha3-linux.json
```