linaro™

Arm Solutions at Lightspeed

Reproducible Embedded Linux Testing

Arm CCA on QEMU * OP-TEE on QEMU and Raspberry Pi 4

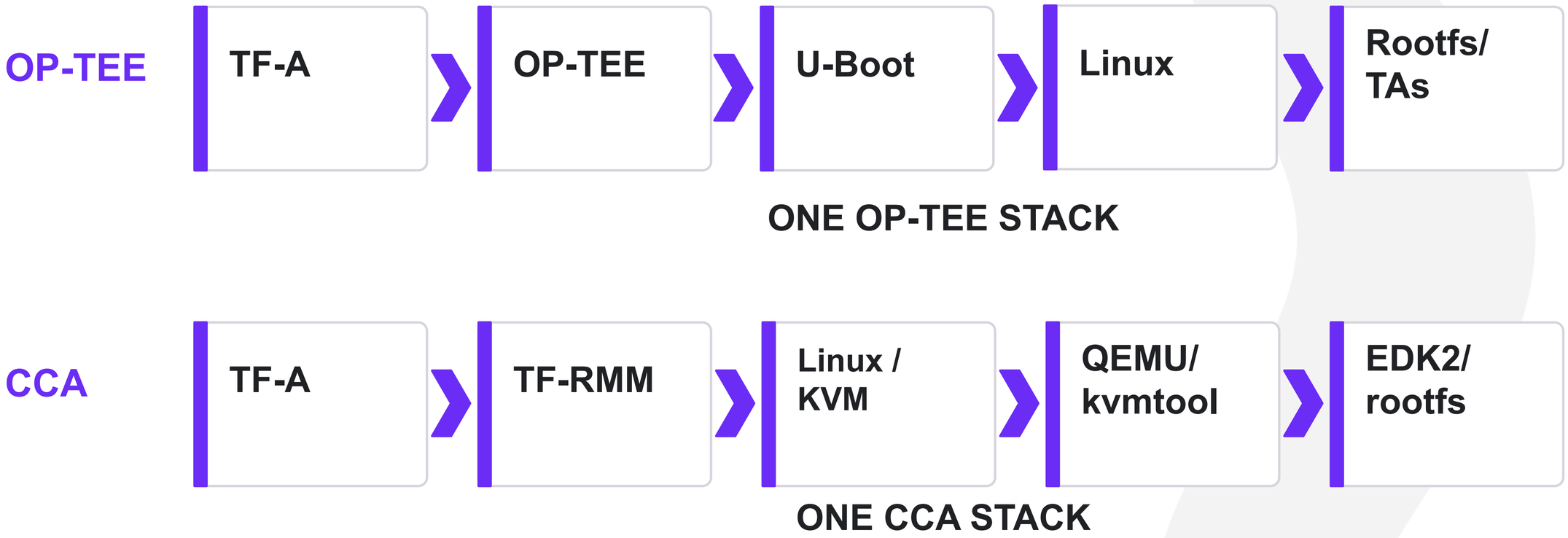
Anders Roxell

Different ports. Same cargo

The place changes. The thing being built/tested should not.

A secure system is a software stack

Neither OP-TEE nor CCA is one project

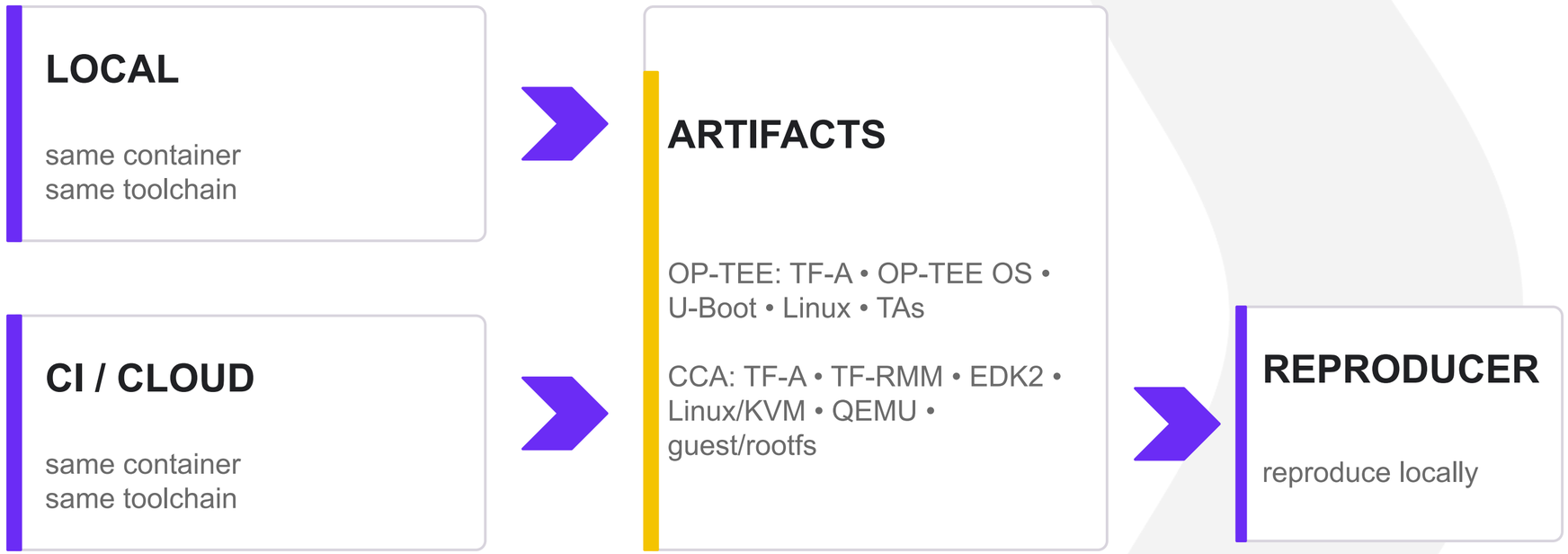


Each component has its own source tree, configuration and release cycle

The system only works when the pieces work together

One build definition, local or CI/cloud

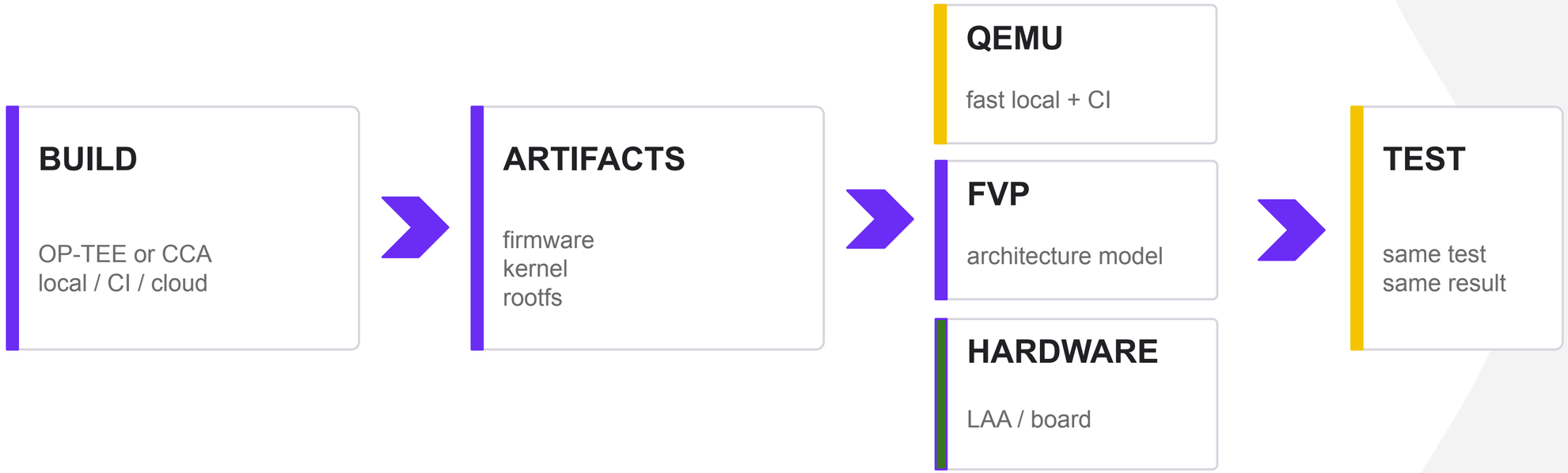
Where the build runs should not change what gets built



Same inputs, same build environment, reproducible outputs

Build per target, run the same test everywhere

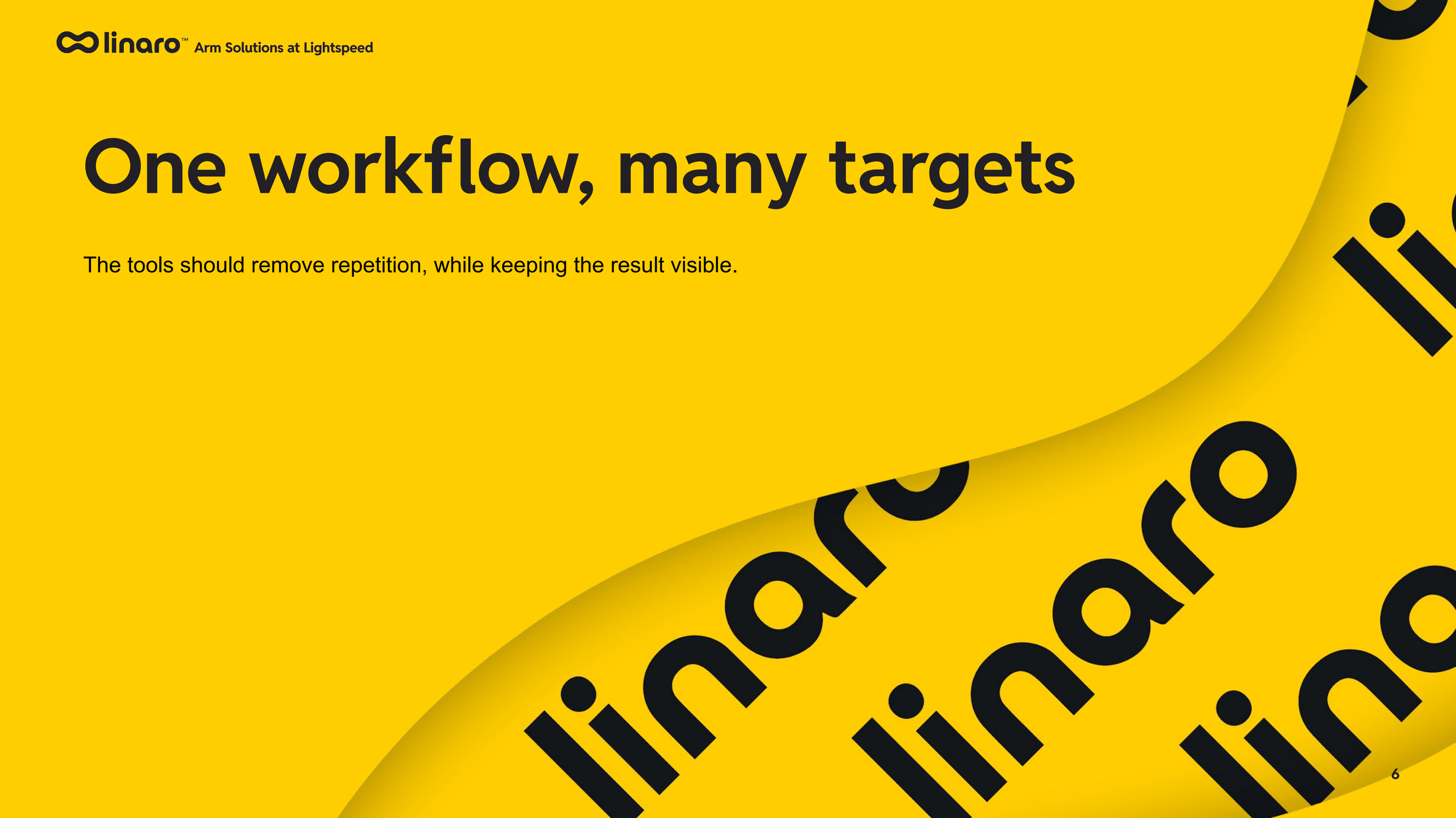
The firmware is target specific, the test, rootfs and the workflow are not



One workflow, different execution targets

One workflow, many targets

The tools should remove repetition, while keeping the result visible.



Okay, what actually implements this?

tuxmake

Linux kernel

tuxfirmware

TF-A + OP-TEE
+ userspace

tuxuboot

U-Boot

shrinkwrap

CCA

→ **ARTIFACTS** →

tuxrun

QEMU • FVP • board

Common build environment, common inputs

Different ports. Same cargo

Local

python command
local/url files
build / test locally

CI / cloud

YAML plan
inputs are URLs
tuxsuite plan (build/test)

Same build/test workflow, different submission process

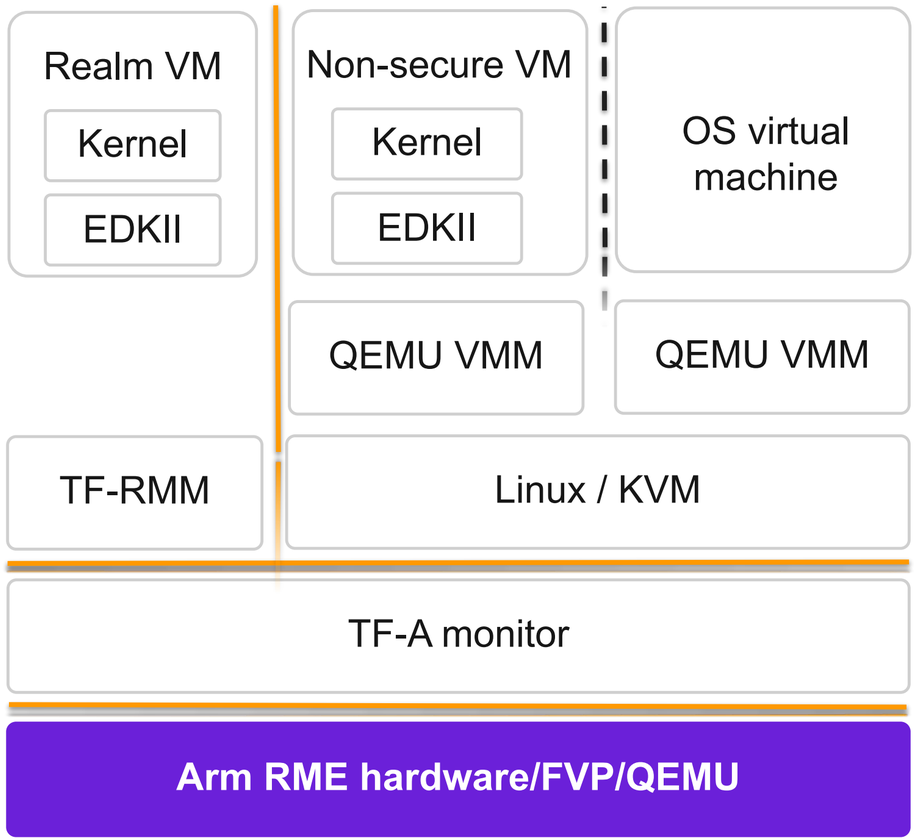
A local reproducer remains available for every CI/cloud run

Arm CCA

A fast- moving stack where every layer has to agree

CCA in one picture

A realm is a VM the host doesn't have access to



Host can manage the realm

but it should not be able to look inside it.

RMI host - RMM

RSI realm - RMM

TF-A boots the RMM. KVM drives it. QEMU starts the guest.

The version numbers do not line up

Different projects, different patch-series revisions, **thousands of combinations**

TF-A	TF-RMM	EDK2	Linux / KVM	QEMU / kvmtool	rootfs

The versions do not always line up just because the names look similar.

One example of combination:

QEMU upstream-v4

Linux/KVM Steven Price v16

TF-A cca/v13

RMM cca/v16

Shrinkwrap: glues the stack together

cca-3world.yaml
+ target overlays



local: shrinkwrap

sync every source tree
drive every component build
use tuxmake underneath



Aligned TF-A, RMM,
EDK2, Linux and QEMU

Build locally

Shrinkwrap: cloud build support

cca-3world.yaml
+ target overlays



Cloud: shrinkwrap

sync every source tree
drive every component build
use tuxmake underneath



Aligned TF-A, RMM, EDK2, Linux and QEMU

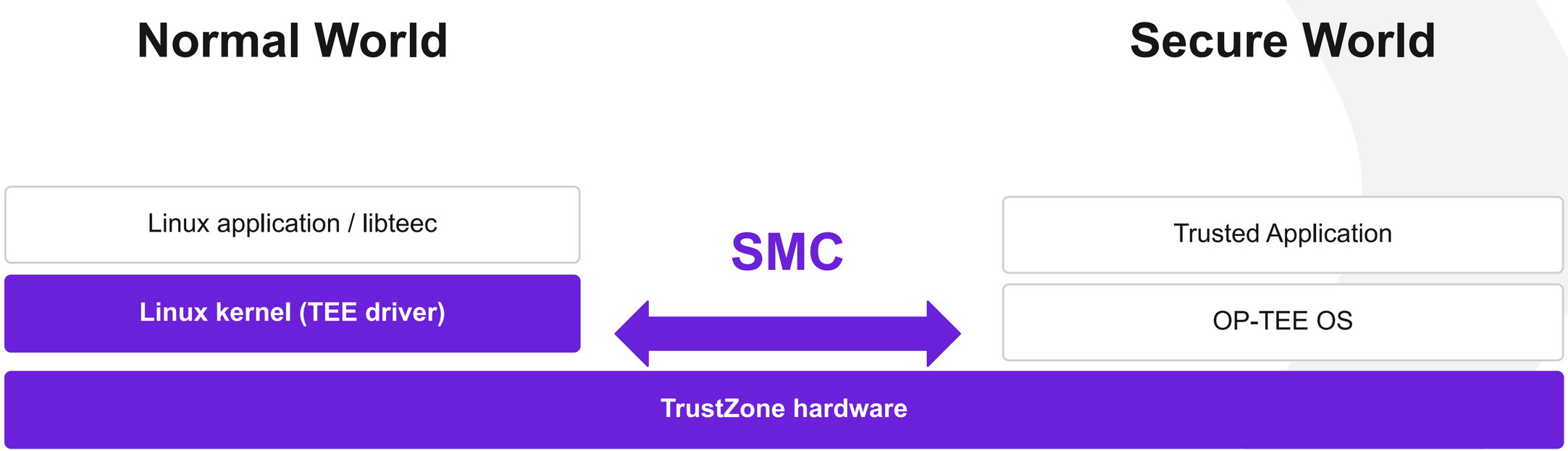
locally reproducible

Submit the build through the cloud

OP-TEE: same problem, different stack

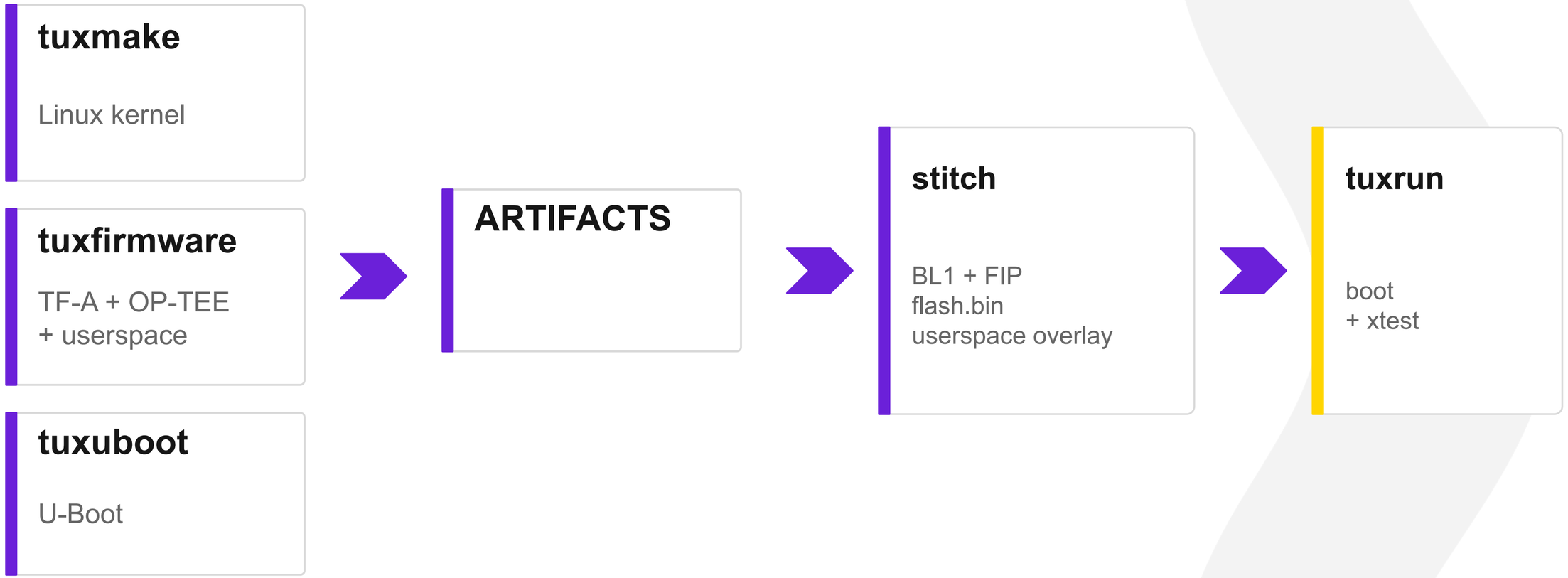
Build and test it on QEMU, then take the test to hardware

OP-TEE in one picture



TF-A at EL3 switches between worlds

Build the pieces together



The userspace overlay contains the TAs signed for this OP-TEE build.

The same build can be done locally or through TuxSuite

OP-TEE on QEMU

```
tuxrun --device qemu-arm64 \
  --enable-trustzone \
  --bios flash.bin \
  --kernel Image.gz \
  --rootfs rootfs.ext4.xz \
  --overlay optee-userspace.tar.xz \
  --tests optee-xtest
```

One regression case

Seconds

Full xtest suite

~10 min under emulation

Run the same xtest used later on hardware

From QEMU to real hardware

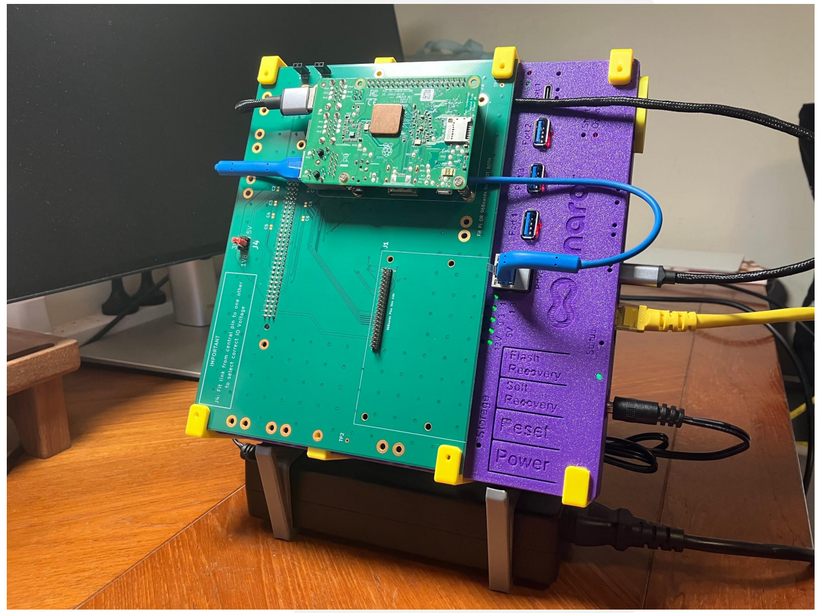
Same OP-TEE xtest

QEMU

TuxRun
run xtest

Real hardware

LAA
Raspberry PI 4
run the same xtest



Same optee (client|xtest), different execution target

Important: BCM2711 has no protected secure memory. Useful for testing, not a real trust boundary

Where I got to

Environments	LOCAL Platform	CI / CLOUD Platform	Real HW Platform via LAA
OP-TEE	QEMU	QEMU	Raspberry PI 4
Arm CCA	QEMU / FVP	QEMU / FVP	Not available to me



linaro.org

Thank you