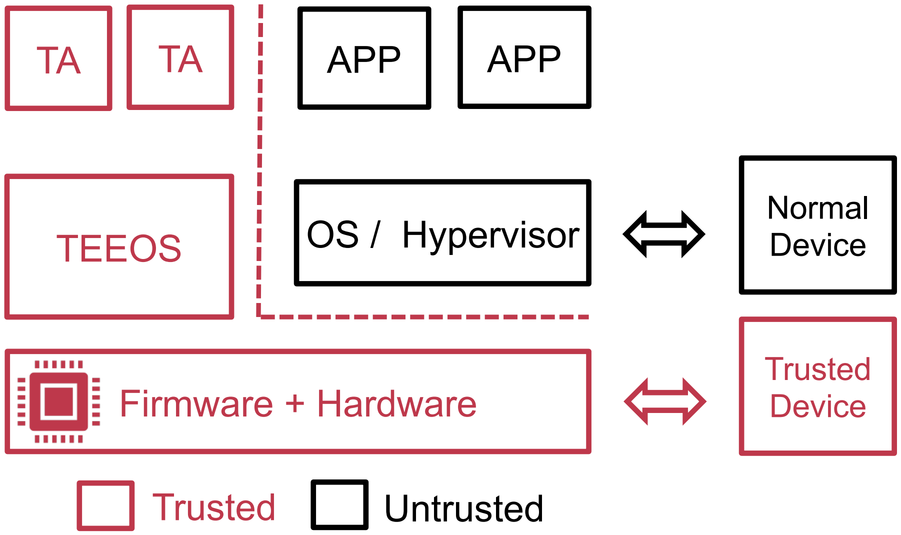
Openharmony-TEE

Erhu Feng

OH RISC-V SIG, Shanghai Jiao Tong University

2024.04.27

Trusted Execution Environment (TEE)



1. **TEE protects trusted app from untrusted software**
   - Hypervisor / OS
   - Other applications
2. **TEE contains secure hardware resources**
   - Secure CPU
   - Protected memory
   - Trusted Devices

- Intel SGX, TDX
- AMD SEV
- ARM TrustZone, CCA
- Penglai, Keystone

TEE is widely used in the mobile system

- **TEE protects the sensitive data and code for both users and developers**

Digital payment

Face recognition

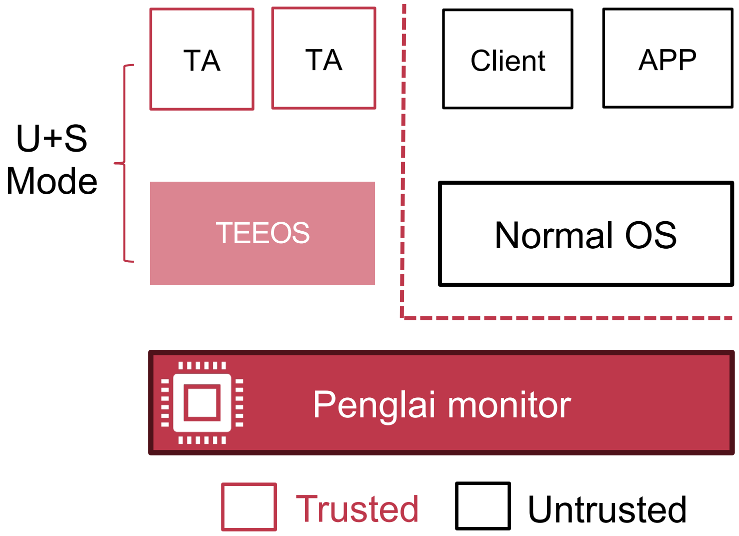
Digital Right Management

TEE in OpenHarmony

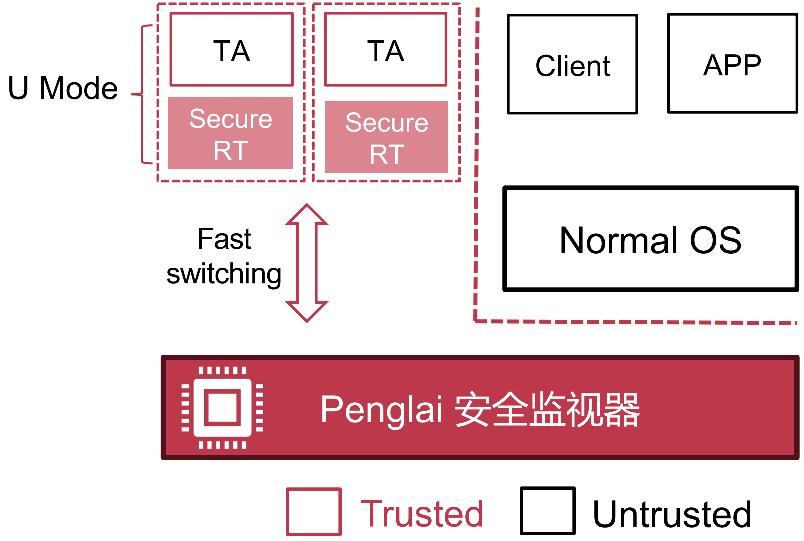
- **Provide a unified TEE architecture for both Arm and RISC-V**
  - Arm: TrustZone with OP-TEE OS
  - RISC-V: Penglai with OP-TEE OS / GP Runtime
- **Benefit: OpenHarmony+Penglai+RISC-V**
  - Open-sourced projects for both hardware and software stacks
  - Research platforms for OS, architecture and security
  - Easy to port the trusted applications from Arm ecology

Penglai Architecture

- Provide two TEE abstractions：Enclave (U mode), Zone (U+S mode)
- Suitable for difference scenarios（Standard device and IoT）



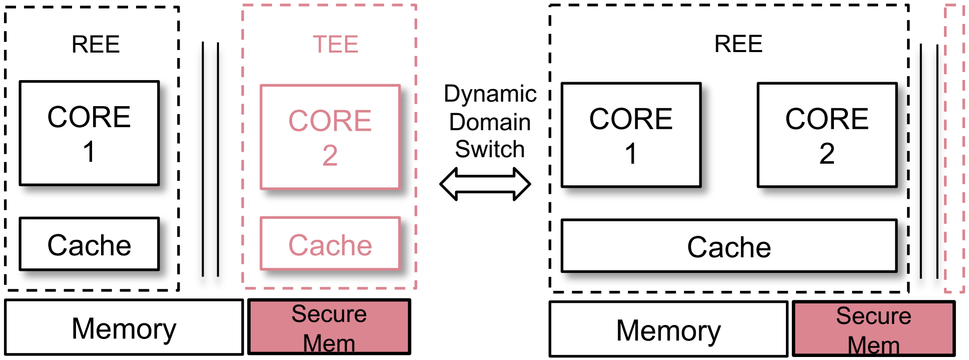
Penglai Zone



Penglai Enclave

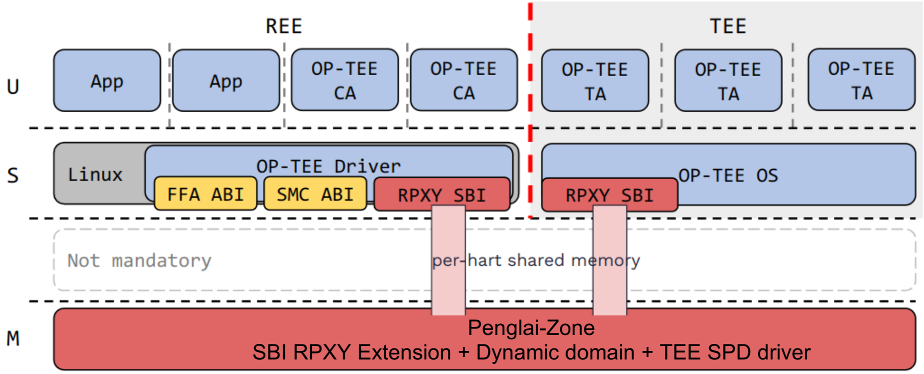
1. Penglai-Zone architecture

- **Underlying mechanizes**
  - Provide the TEE model which is similar to the TrustZone: TEE (Trusted execution environment) and REE (Rich execution environment)
  - Strong isolation between CPU, memory and I/O device
    - A presentation in Main program: Session 10D - sIOPMP
  - Dynamic domain switch between REE and TEE



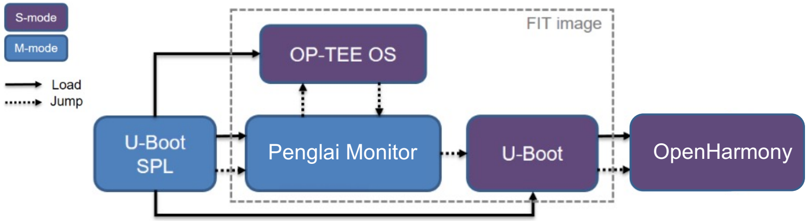
Penglai-Zone architecture

- **Components**
  - (secure) Penglai Monitor: Running as secure firmware in the M mode
  - (secure) OP-TEE OS: Trusted TEE OS running in the secure S mode
  - (Non-secure) OP-TEE Driver: Linux kernel driver installed in the REE
  - (secure) OP-TEE TA: Trusted application running in the TEE
  - (Non-secure) OP-TEE CA: Client application running in the REE



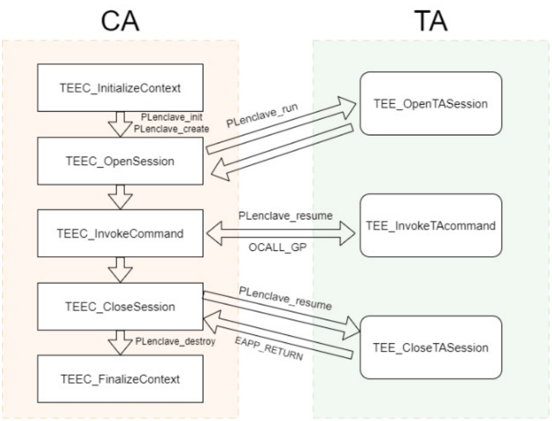
Penglai-Zone architecture

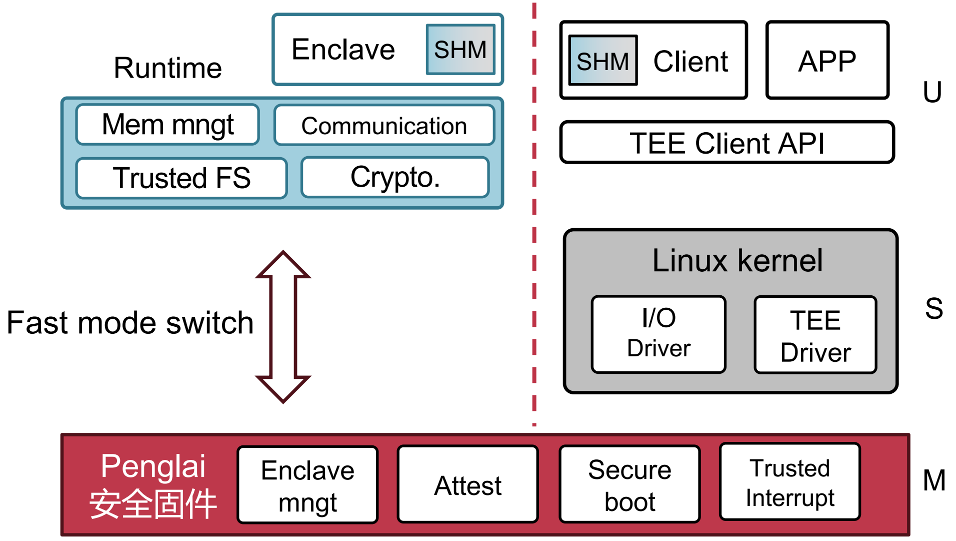
- **Boot flow**
  - U-boot SPL loads and verifies the Penglai Monitor
  - Penglai monitor verifies the OP-TEE OS, and jumps to the OPTEE OS in the secure domain for initialization
  - After return from OPTEE-OS, Penglai monitor jump to the non-secure domain for loading U-Boot and OpenHarmony



2. Penglai-Enclave Architecture

- **Provide a more lightweight TEE abstraction: Enclave (U mode)**
  - Support various enclave runtimes
  - Automatically generate ecall/ocall function
  - TLS/ Trusted FS supported
- **GP-like Programming**





Distributed TEE

- **Offload the TEE task**
  - Not all devices have the TEE support
  - Distributed TEE allows a TEE-unsupported device to enable the TEE capability
- **Aggregate the TEE hardware resource**
  - Different devices have the different TEE resources
  - Distributed TEE can aggregate all TEE resource to provide a unified TEE abstraction
- **Developer agnostic**
  - The developer does not need to care whether the underlying hardware supports TEE or not

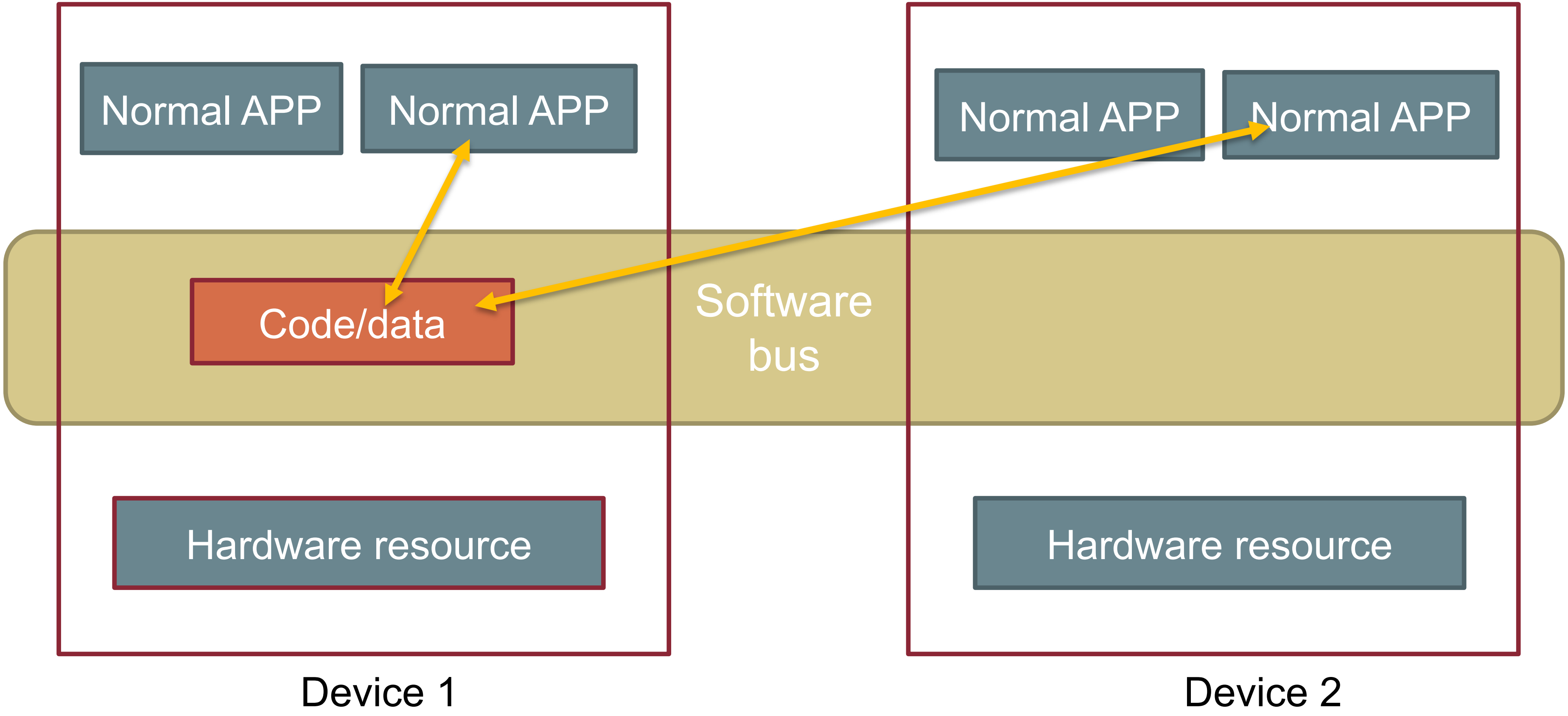
Demo1: Smart door lock with face recognition

- Re-use the camera in the mobile phone

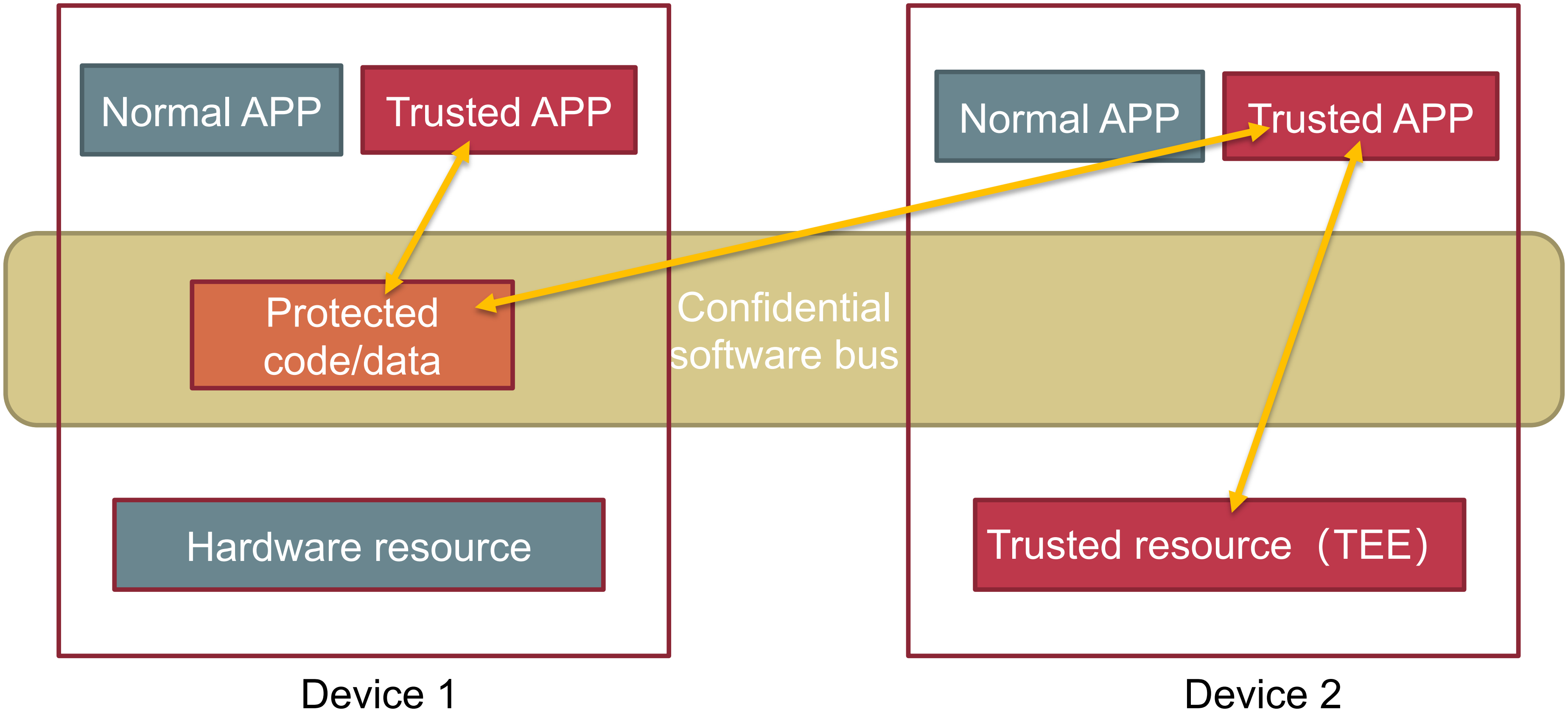
Demo2: Smart watch for personal health analysis

- Send the personal health data to the TEE in mobile phone

Distributed APP in OpenHarmony

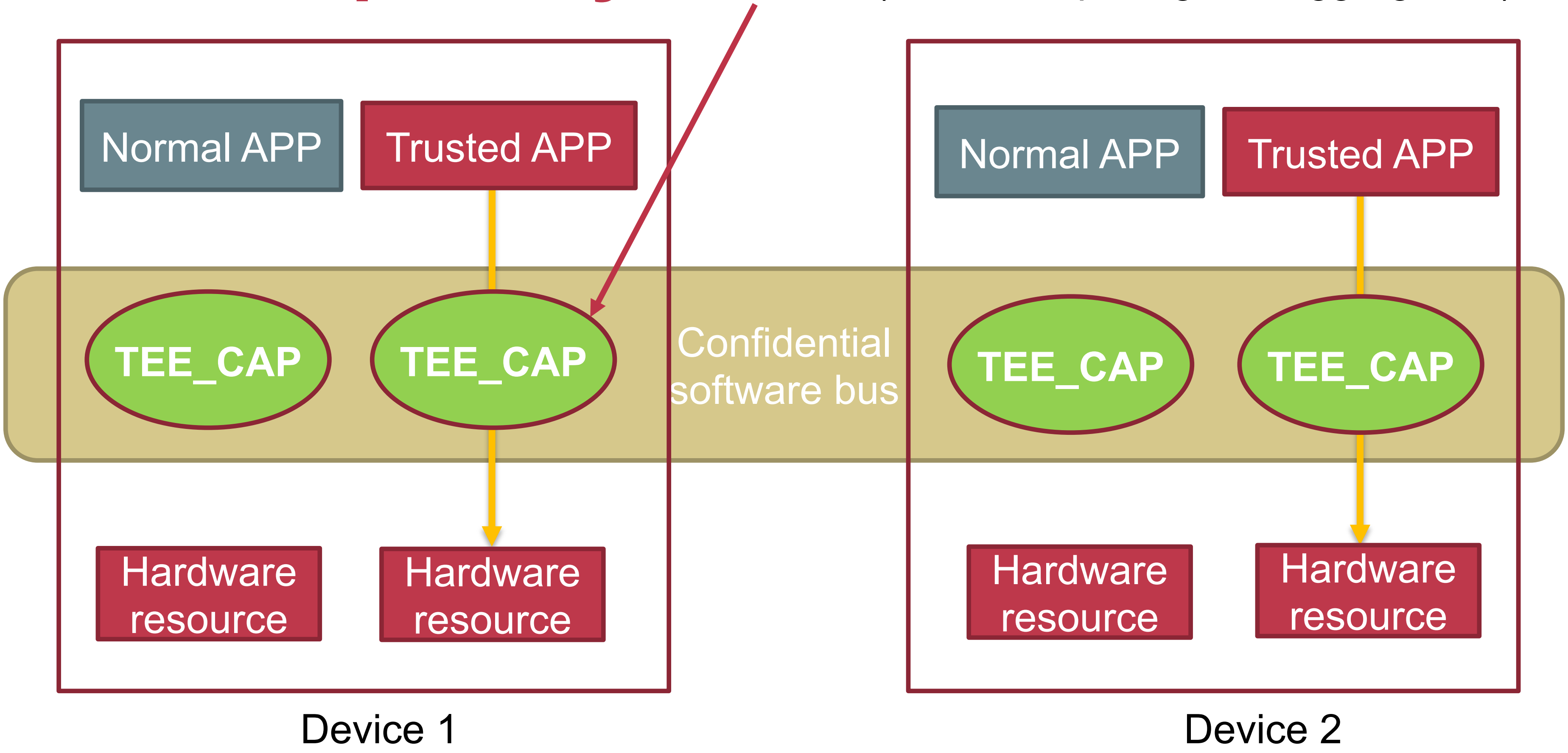


Distributed TEE design in OpenHarmony



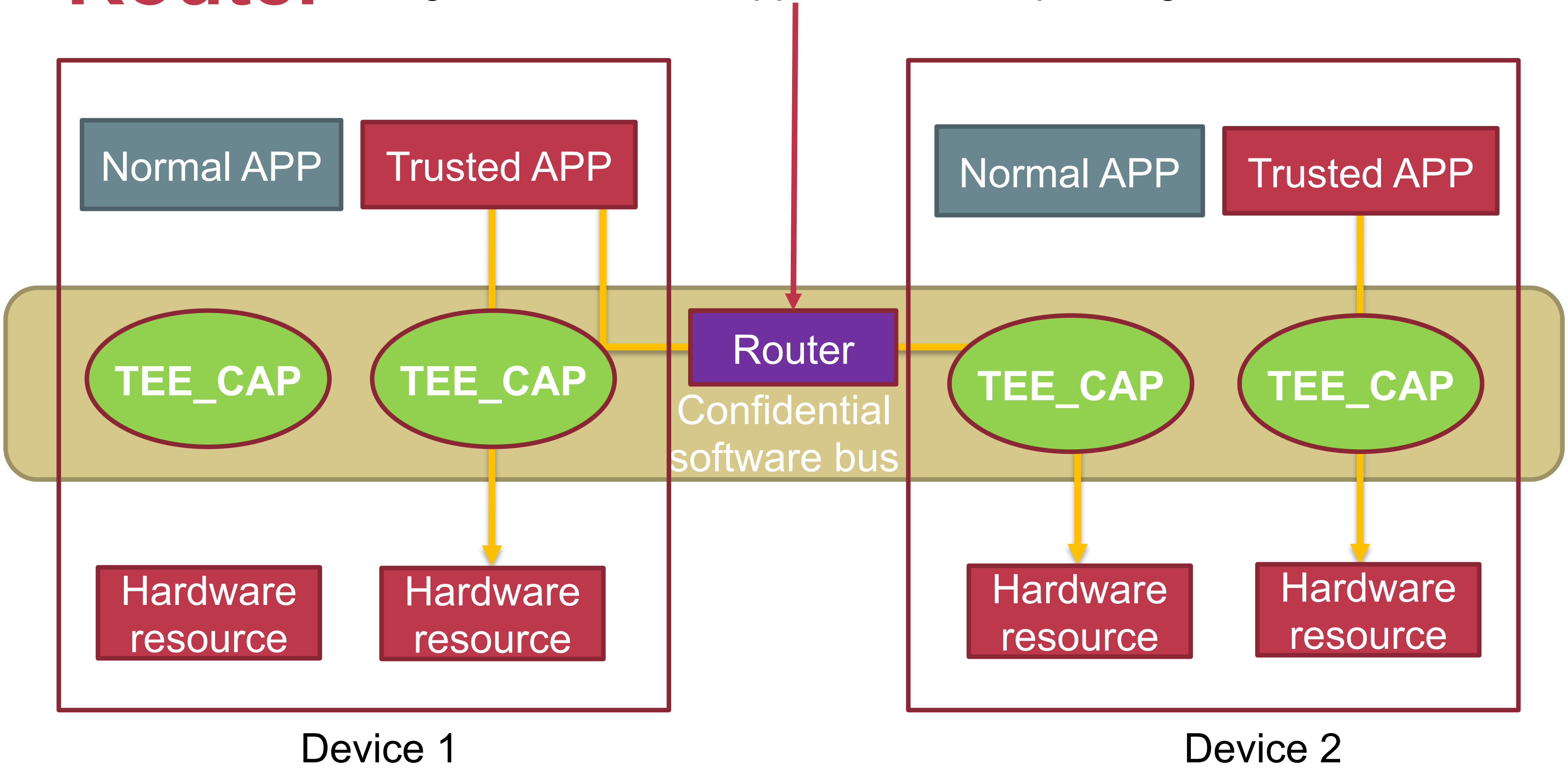
TEE Capability

Dynamic and fine-grained management for TEE resources (resource splitting and aggregation)



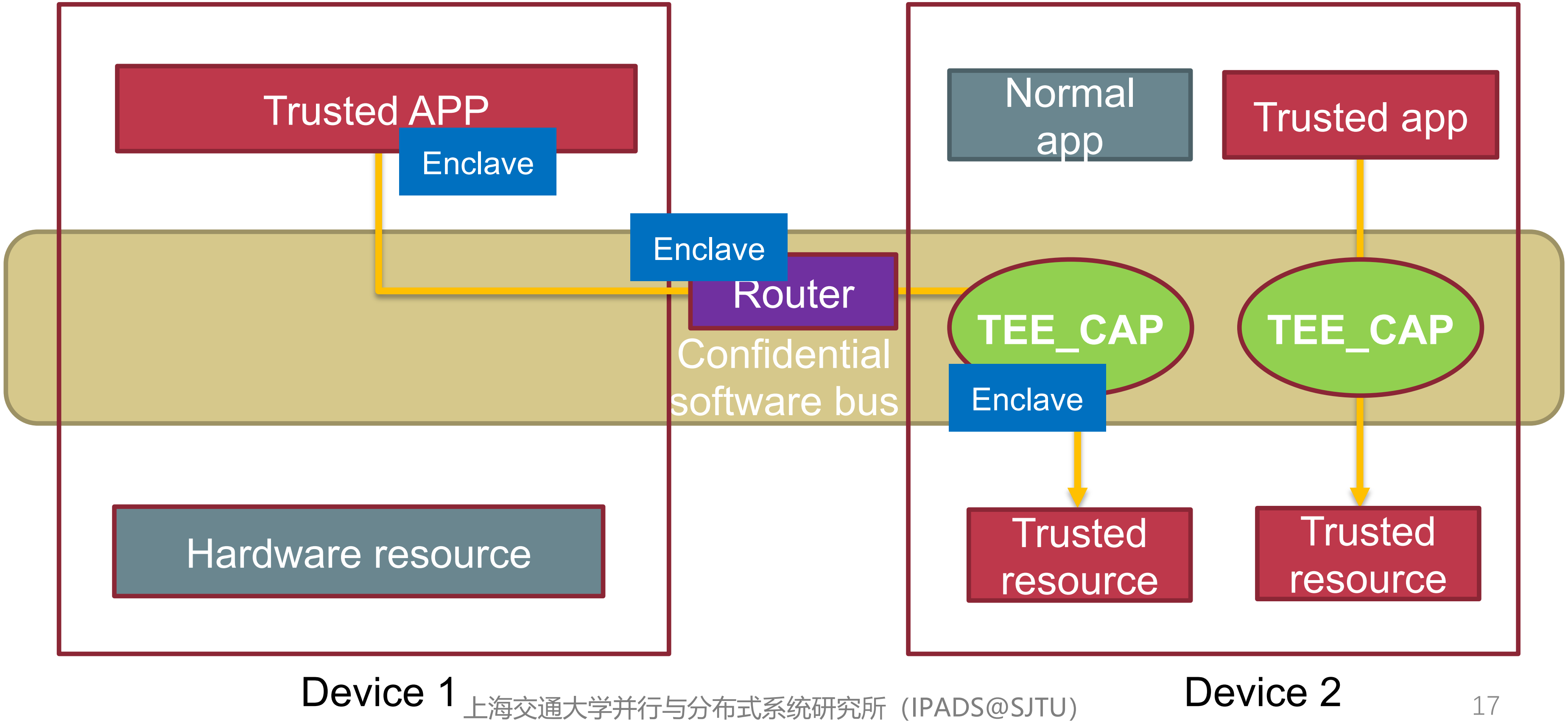
Router

Automatically select the idle TEE resource, and migrate the trusted app to the corresponding device



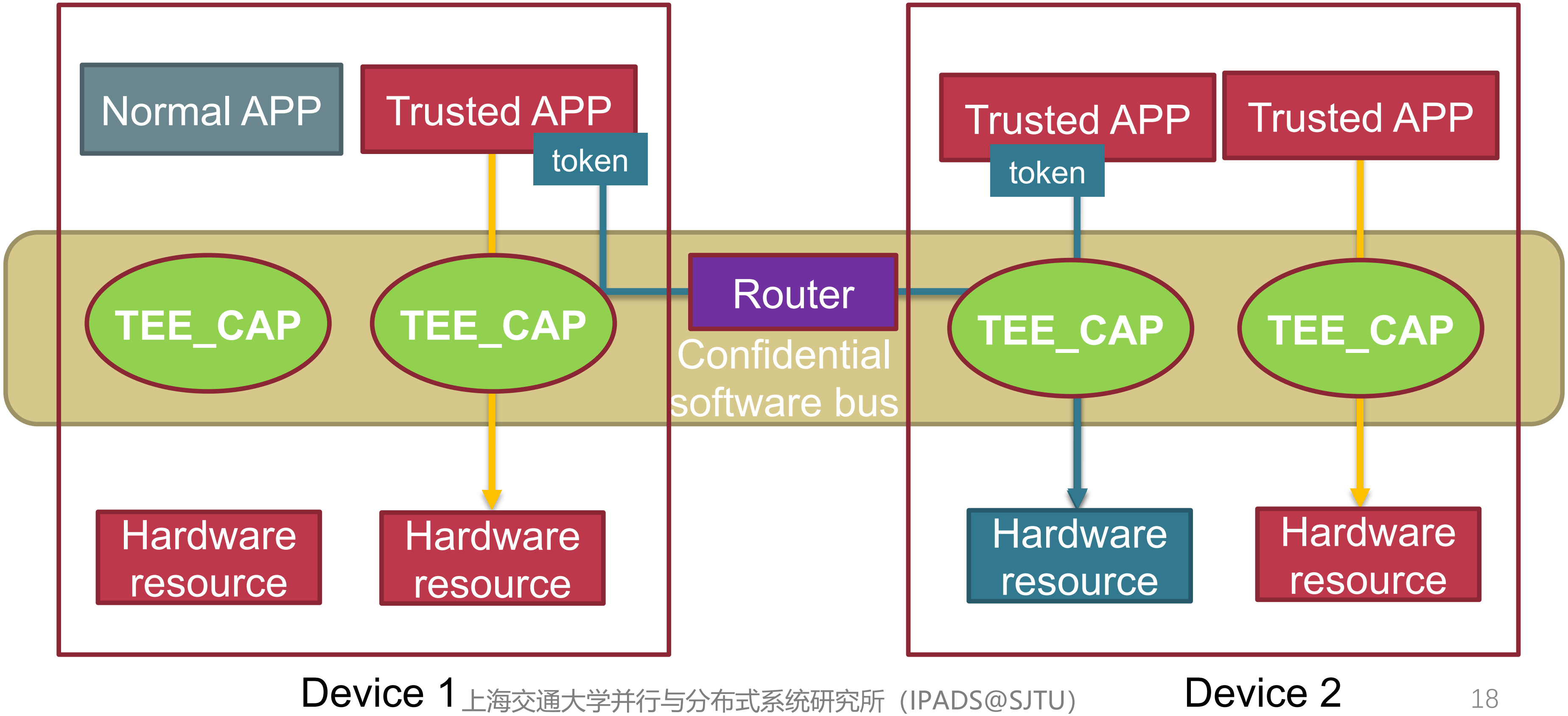
Use case 1： offload the trusted app

- Deploy the trusted app to a remote device with TEE



Use case 2: Sharing the TEE resource

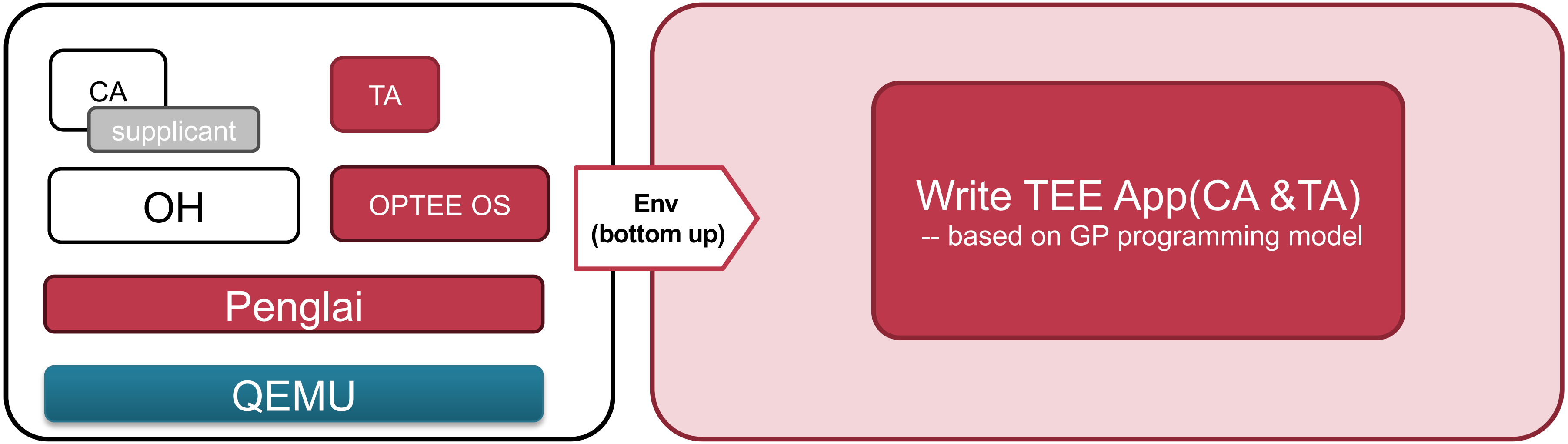
- **Trusted app can share the same TEE_cap with token**



In tutorial 1, we will

– Prepare the OpenHarmony development environment for you

– Prepare the Penglai-Zone TEE development environment and go through the development workflows with you



How to run Penglai-Zone in OpenHarmony

- **As before, use Qemu v8.x**
- **Download Penglai-Zone setup proj source code**

```
export WORKDIR=`pwd`

git clone https://github.com/Shang-QY/test_polyos_with_optee.git
```

- **Prepare Device Tree Blob**

```
sudo apt install dtc

dtc -I dts -O dtb -o qemu-virt-new.dtb test_polyos_with_optee/qemu-virt-restrict.dts
```

- **Compile Penglai-Zone opensbi**

```
cd $WORKDIR/test_polyos_with_optee
git clone https://github.com/Penglai-Enclave/opensbi.git -b dev-rpxy-optee-v3
cd opensbi
CROSS_COMPILE=riscv64-linux-gnu- make PLATFORM=generic
cp build/platform/generic/firmware/fw_dynamic.elf $WORKDIR
```

How to run Penglai-Zone in OpenHarmony

- **Compile OPTEE OS/ client/ examples**

```
cd $WORKDIR/test_polyos_with_optee
./script/build_optee.sh # for easily compilation all together
```

How to run Penglai-Zone in OpenHarmony

- **Copy CA, TA and startup script to OH images**

```
cd $WORKDIR
mkdir -p mnt
sudo mount images/system.img ./mnt

sudo cp -rf test_polyos_with_optee/optee_client/build/out/export/usr/sbin/tee-supplicant ./mnt/system/bin/

sudo mkdir -p ./mnt/system/lib/optee_armtz
sudo cp test_polyos_with_optee/optee_examples/hello_world/ta/8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta ./mnt/system/lib/optee_armtz/
sudo cp test_polyos_with_optee/optee_examples/hello_world/host/optee_example_hello_world ./mnt/system/bin/

sudo umount ./mnt
```

```
cd $WORKDIR
sudo mount -o loop images/userdata.img ./mnt

cat > mnt/start_optee_supplicant.sh << EOF
if [ -e /bin/tee-supplicant -a -e /dev/teepriv0 ]; then
        echo "Starting tee-supplicant..."
        tee-supplicant&
        ifconfig lo up
        exit 0
else
        echo "tee-supplicant or TEE device not found"
        exit 1
fi
;;
EOF

sudo chmod a+x mnt/start_optee_supplicant.sh
sudo umount ./mnt
```

How to run Penglai-Zone in OpenHarmony

- **Run polyos with optee**

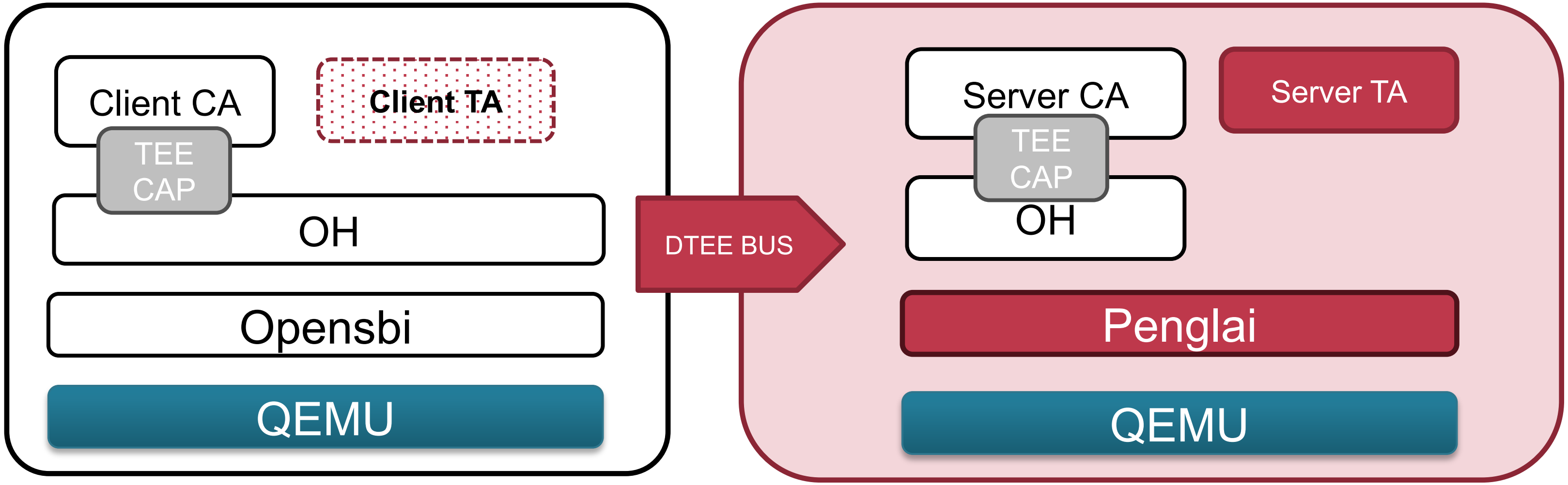
```
cd $WORKDIR
./test_polyos_with_optee/run_polyos.sh
```

- **After Login, execute**

```
cd data
./start_optee_supplicant.sh
optee_example_hello_world
```

In tutorial 2, we will

- Prepare the Distributed TEE development environment for you
- Offload the TEE application to another machine with TEE



How to run distributed TEE in OpenHarmony

- **Download the repo**

```
git clone https://github.com/iku-iku-iku/dteegen.git
cd dteegen
git submodule update --init --recursive
```

- **Download the prebuild OH image for distributed TEE**
  - It will take a few minutes

```
bash ./scripts/download_prebuilt.sh
export OH_HOME=`pwd`/polyos
export OH_IMAGES=$OH_HOME/out/riscv64_virt/packages/phone/images
```

How to run distributed TEE in OpenHarmony

- **Build Penglai monitor and driver**

```
cd Penglai-Enclave-sPMP
export PENGLAI_HOME=`pwd`

# build opensbi
bash ./build_opensbi.sh

# build the driver (optionally, we have prepared the Penglai driver in the OH image)
bash ./scripts/build_driver_for_oh.sh

cd ..
```

How to run distributed TEE in OpenHarmony

- **Create a quick demo**

```
# download dteegen tool
curl -o dteegen https://raw.githubusercontent.com/iku-iku-iku/dteegen/master/scripts/all_in_one.sh
chmod +x dteegen
sudo mv dteegen /usr/local/bin

# create new project
export PROJECT_NAME=new_project
export PROJECT_PATH=`pwd`/$PROJECT_NAME
dteegen create $PROJECT_NAME
dteegen deploy $PROJECT_NAME
```

How to run distributed TEE in OpenHarmony

- **Do some preparation for running OpenHarmony.**

```
# Copy opensbi to $OH_HOME
cp $PENGLAI_HOME/opensbi-1.2/build-oe/qemu-virt/platform/generic/firmware/fw_jump.bin
$OH_HOME

# Copy scripts to $OH_HOME
cp $PENGLAI_HOME/scripts/start_server.sh $OH_HOME
cp $PENGLAI_HOME/scripts/start_client.sh $OH_HOME

export MOUNT_PATH=/tmp/mount
mkdir -p $MOUNT_PATH

# Inject built files to OH images
./scripts/copy_penglai_app.sh

# Since instances can not share the same images, we need to copy them.
./scripts/create_images.sh
```

How to run distributed TEE in OpenHarmony

- **Create network bridge.**

```
sudo ip link add name br0 type bridge
sudo ip link set dev br0 up
sudo ip addr add 192.168.1.109/24 dev br0
sudo iptables -P FORWARD ACCEPT
```

How to run distributed TEE in OpenHarmony

- **Run server and client.**

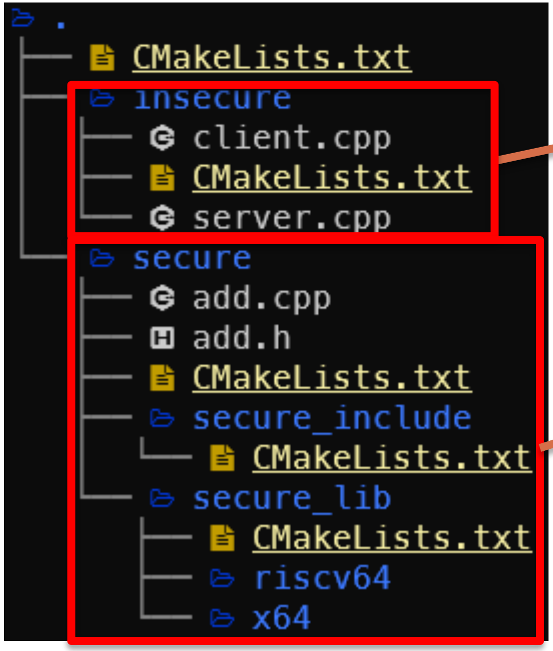
```
# run server in a machine with TEE
cd $OH_HOME
./start_server.sh

# in OH
cd data
insmod penglai.ko
./server
```

```
# run client in a machine without TEE

cd $OH_HOME
./start_client.sh
# in OH
cd data
./client
```

How to develop distributed tee project



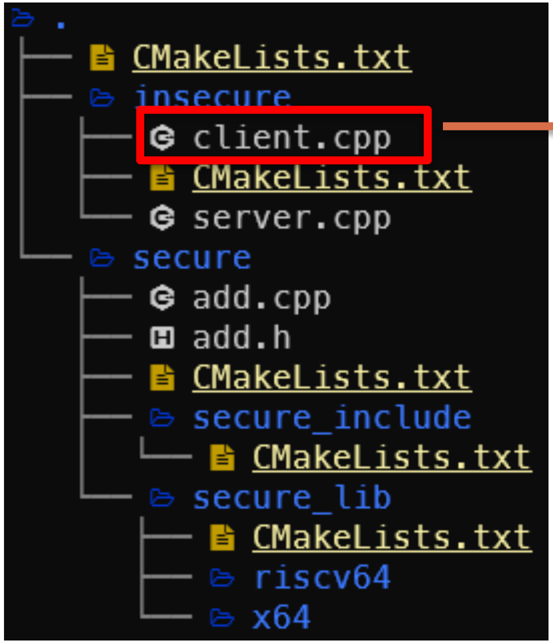
```
.
├── CMakeLists.txt
├── insecure
│   ├── client.cpp
│   ├── CMakeLists.txt
│   └── server.cpp
└── secure
    ├── add.cpp
    ├── add.h
    ├── CMakeLists.txt
    ├── secure_include
    │   └── CMakeLists.txt
    └── secure_lib
        ├── CMakeLists.txt
        ├── riscv64
        └── x64
```

Write untrusted logic in the “insecure” folder

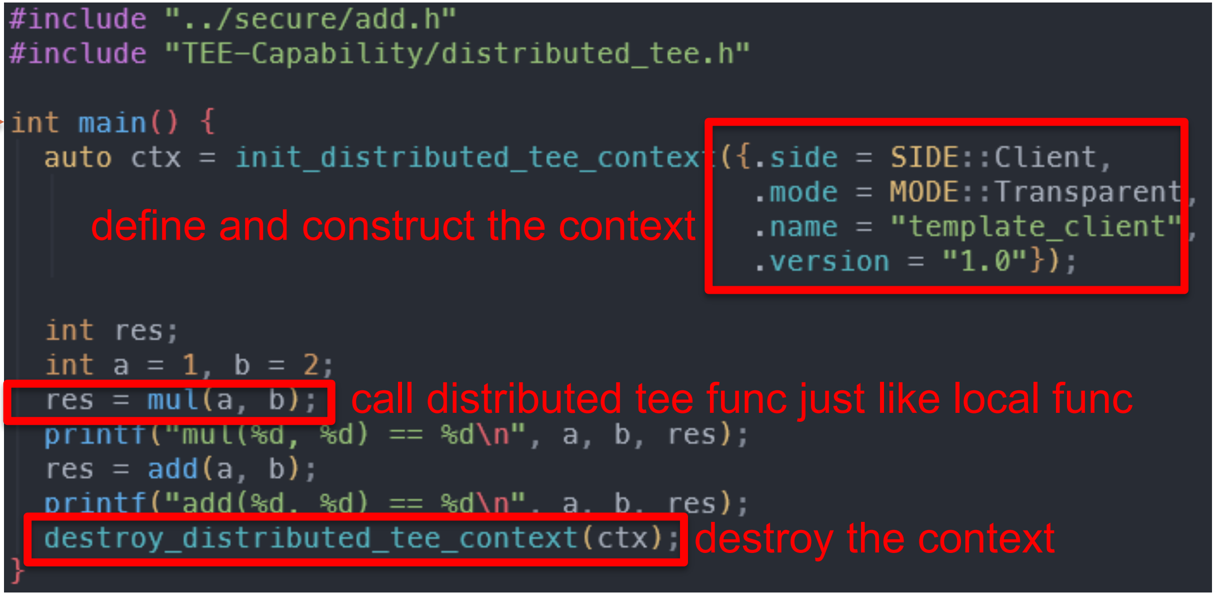
Write secure logic(run in TEE) in the “secure” folder

- Without distributed tee ability
- Debugging friendly
- Deployable with dteegen

How to develop distributed tee project



```
.
├── CMakeLists.txt
├── insecure
│   ├── client.cpp
│   ├── CMakeLists.txt
│   └── server.cpp
└── secure
    ├── add.cpp
    ├── add.h
    ├── CMakeLists.txt
    ├── secure_include
    │   └── CMakeLists.txt
    └── secure_lib
        ├── CMakeLists.txt
        ├── riscv64
        └── x64
```



```
#include "../secure/add.h"
#include "TEE-Capability/distributed_tee.h"

int main() {
  auto ctx = init_distributed_tee_context({.side = SIDE::Client,
                                            .mode = MODE::Transparent,
                                            .name = "template_client",
                                            .version = "1.0"});

  int res;
  int a = 1, b = 2;
  res = mul(a, b);
  printf("mul(%d, %d) == %d\n", a, b, res);
  res = add(a, b);
  printf("add(%d, %d) == %d\n", a, b, res);
  destroy_distributed_tee_context(ctx);
}
```

define and construct the context

call distributed tee func just like local func

destroy the context

Thanks