SECRCT FLOW 隐语

隐私计算线上慕课【第17讲】

框架基础 | 隐语密态计算设备SPU背景与原理

谭晋 | 蚂蚁集团隐语团队

目录

Contents

01 Why SPU

02 What is SPU

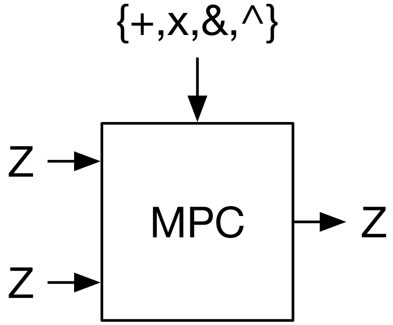
03 How to use SPU

Why SPU

01

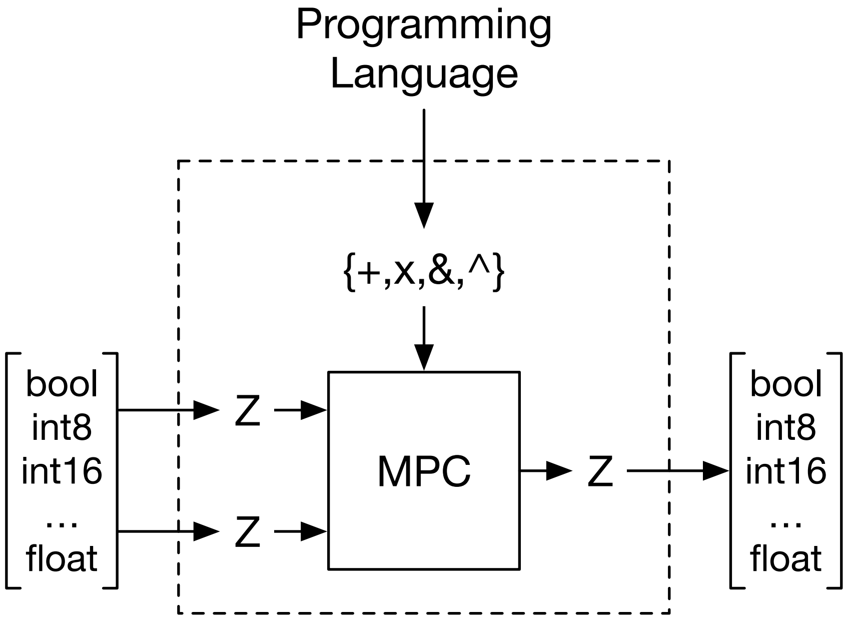
Why SPU

- MPC
  - Simple types (Z_{2k} , Z_q, Z_2)
  - Simple operators (+,x,&,^)
  - Hard to use



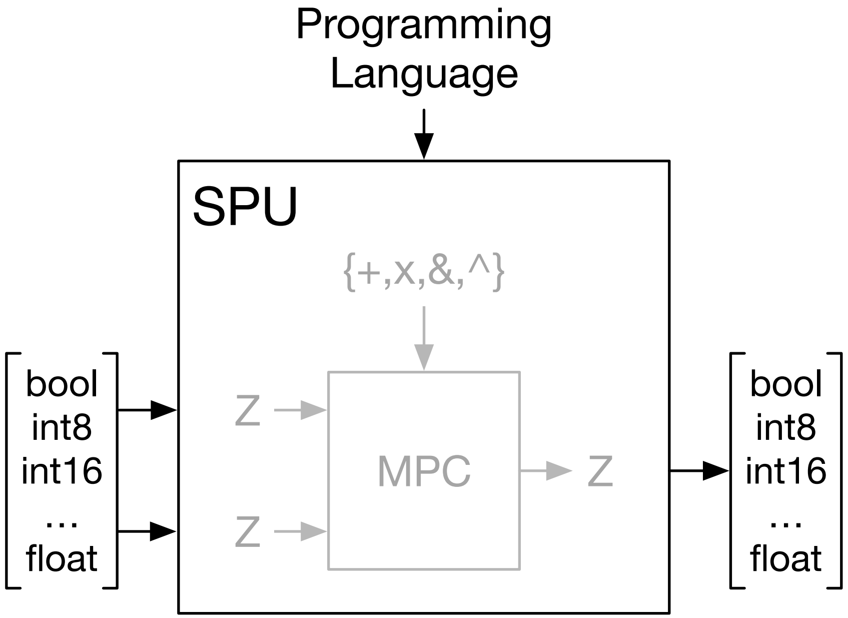
Why SPU

- MPC
  - Simple types (Z_{2k} , Z_q, Z_2)
  - Simple operators (+,x,&,^)
  - Hard to use
- What we want
  - Rich types (int, float, tensor)
  - Rich ops (programmable)
  - Easy to use



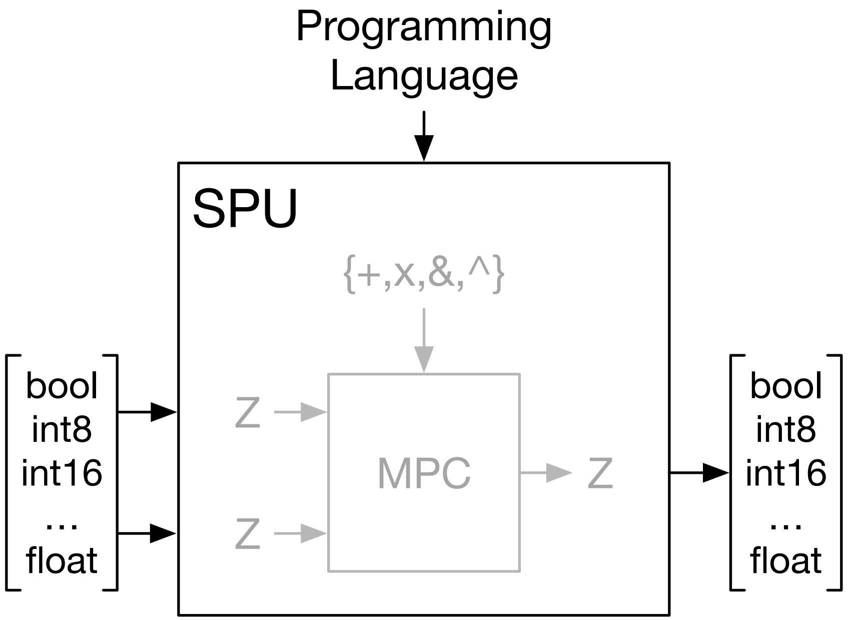
Why SPU

- MPC
  - Simple types (Z_{2^k} , Z_q, Z_2)
  - Simple operators (+,x,&,^)
  - Hard to use
- SPU – Secure Processing Unit
  - Rich types (int, float, tensor)
  - Rich ops (programmable)
  - Easy to use



Why SPU

- MPC
  - Simple types (Z_{2k} , Z_q, Z_2)
  - Simple operators (+,x,&,^)
  - Hard to use
- SPU – Secure Processing Unit
  - Rich types (int, float, tensor)
  - Rich ops (programmable)
  - Easy to use



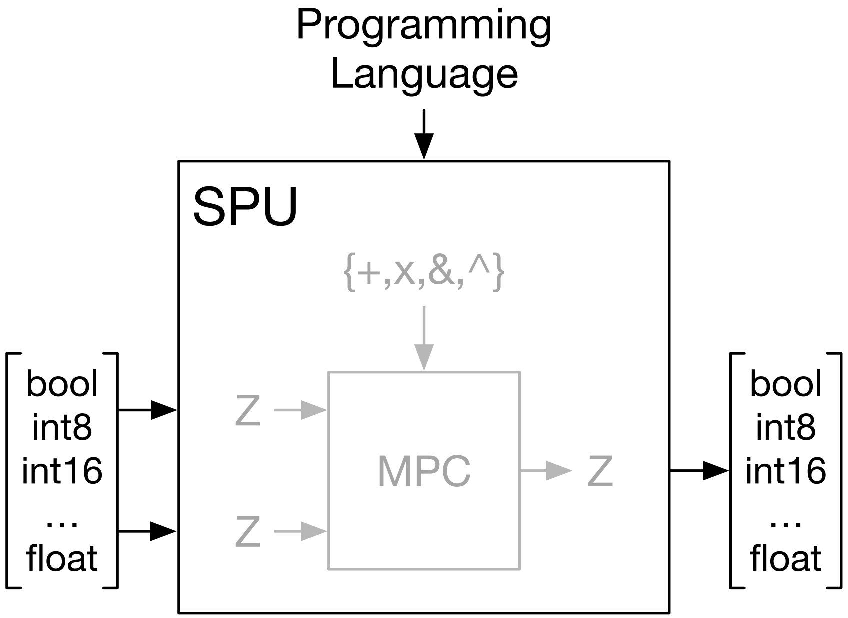
SPU is a domain specific compiler & runtime suite

What is SPU

02

What is SPU

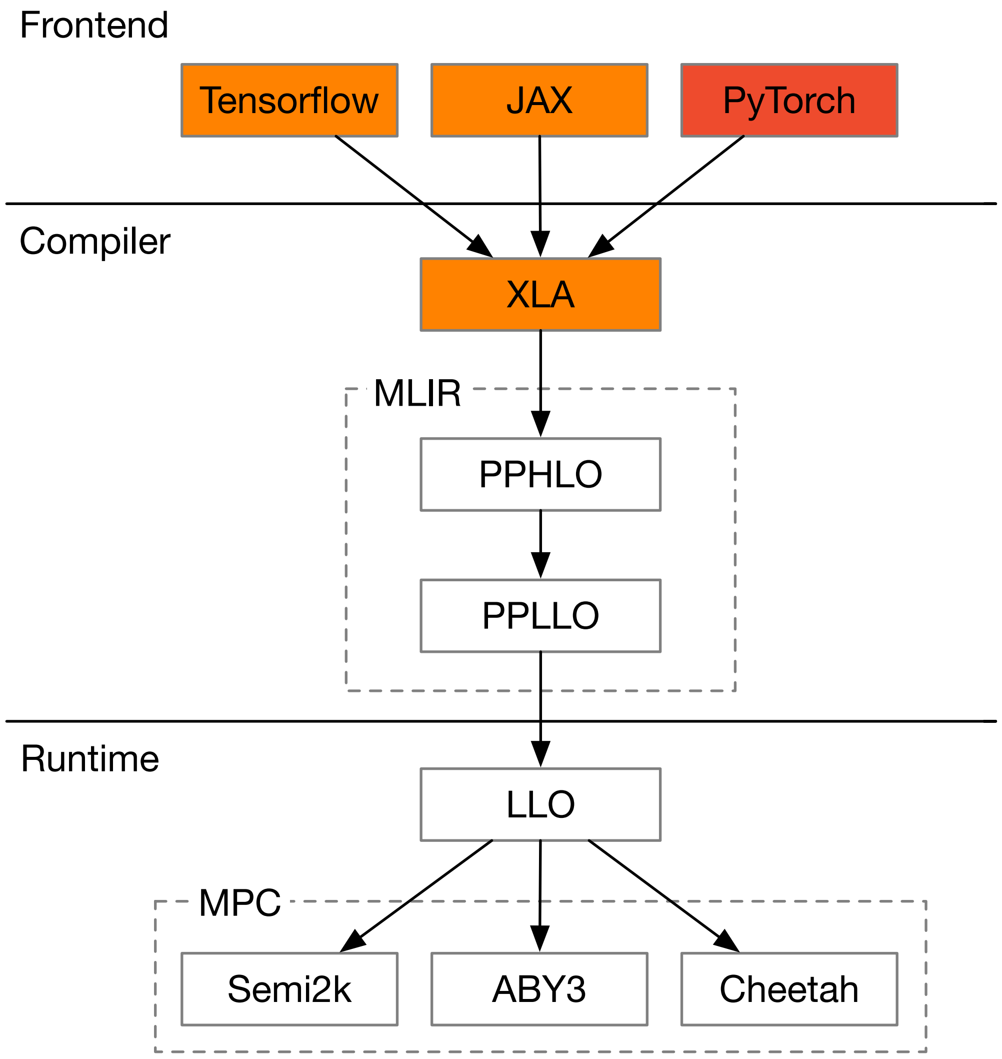
- Language
- Compiler stack
- Runtime stack



SPU is a domain specific compiler & runtime suite

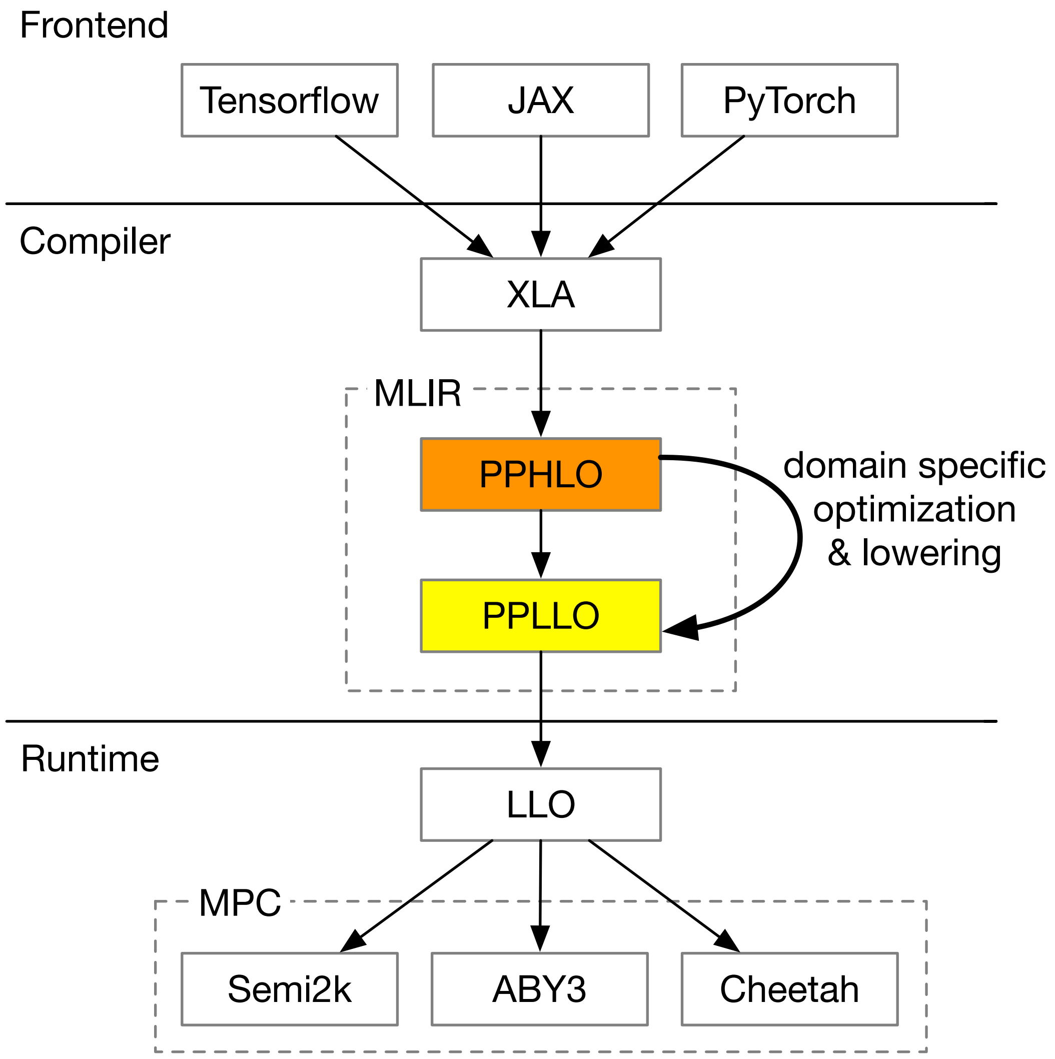
What is SPU

- Language
  - Native AI framework
  - Reduce learn costs
  - Reuse tensor ops, autodiff
- Compiler stack
- Runtime stack



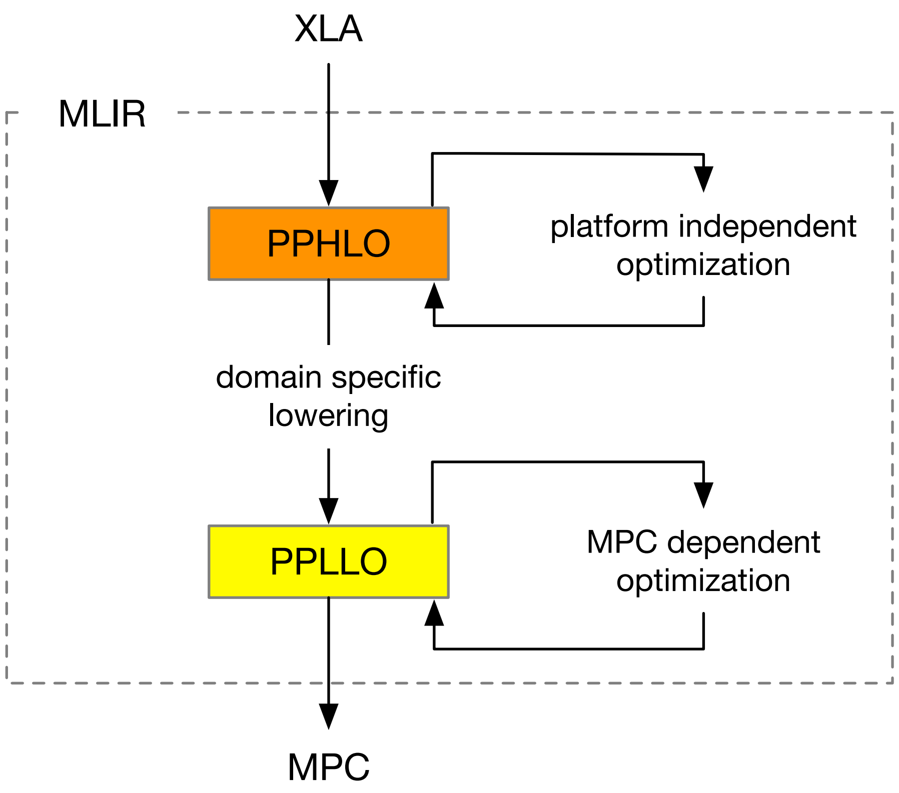
What is SPU

- Language
- Compiler stack
  - Privacy preserving semantic (HLO, LLO)
  - Reuse AI compiler stack
  - Domain specific optimization
  - Domain specific lowering
- Runtime stack



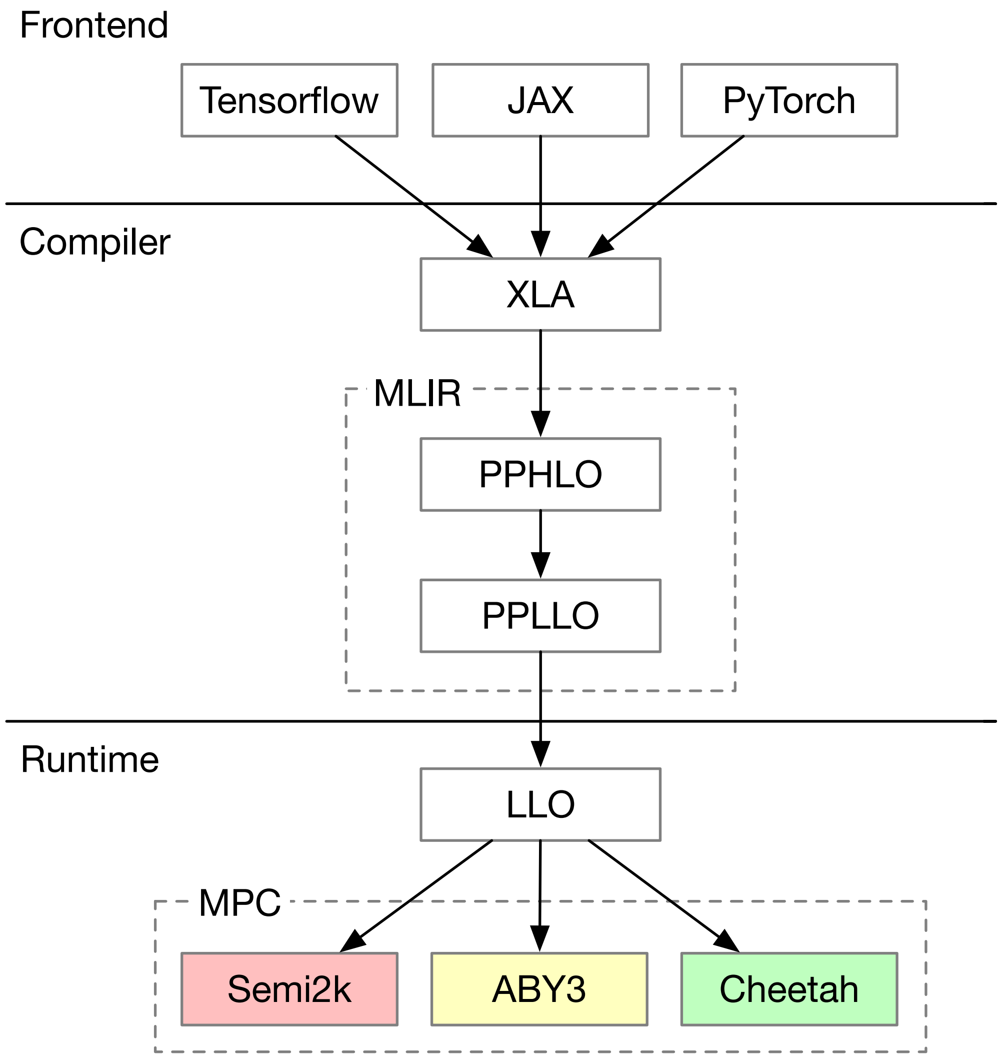
What is SPU

- Language
- Compiler stack
  - Privacy preserving semantic (HLO, LLO)
  - Reuse AI compiler stack
  - Domain specific optimization
  - Domain specific lowering
- Runtime stack



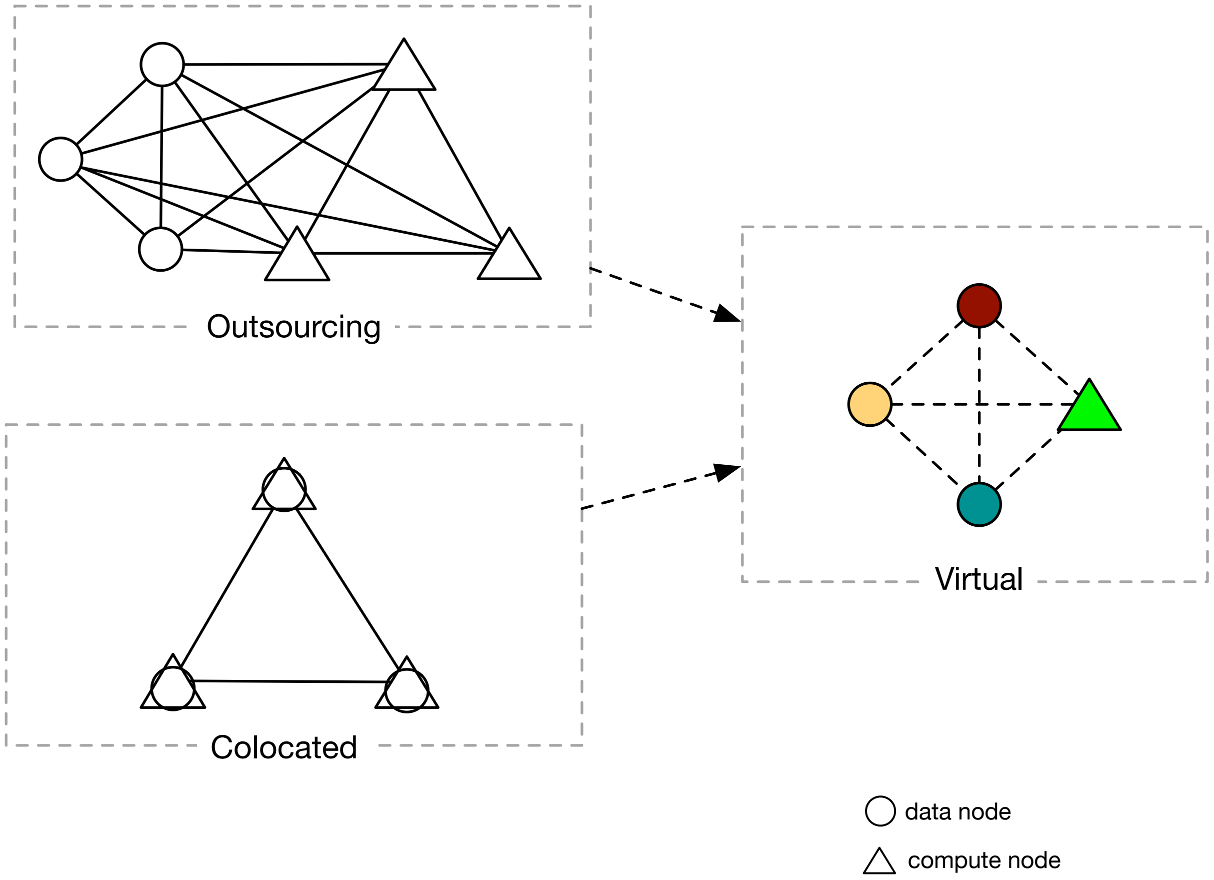
What is SPU

- Language
- Compiler stack
- Runtime stack
  - Multiple MPC protocols
    - 2PC/3PC/NPC
    - Semi-honest/Malicious
    - More
  - Deployment transparent

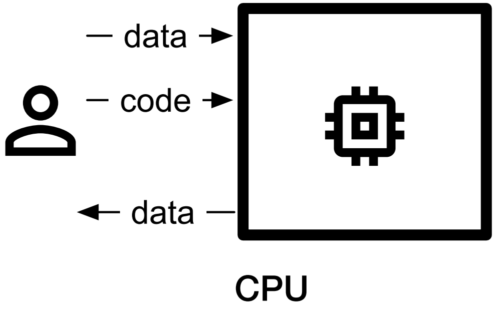


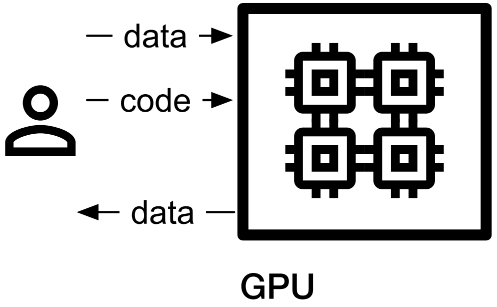
What is SPU

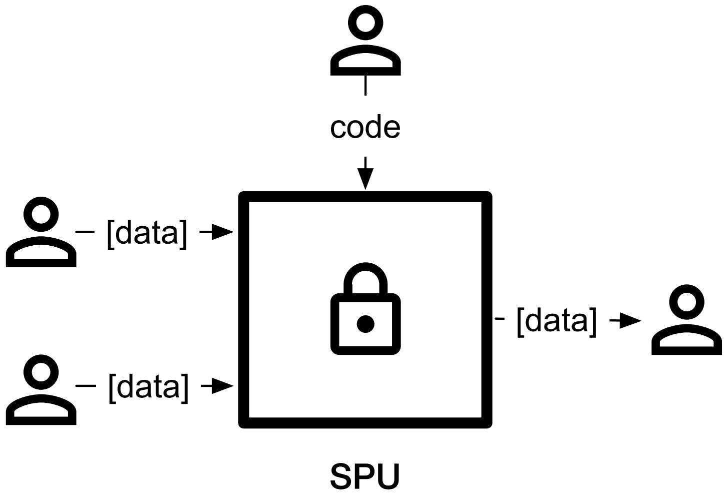
- Language
- Compiler stack
- Runtime stack
  - Multiple MPC protocols
  - Deployment transparent
    - Write once
    - Run everywhere



What is SPU







- Physical
- General
- Fast

- Physical
- Parallel
- SuperFast

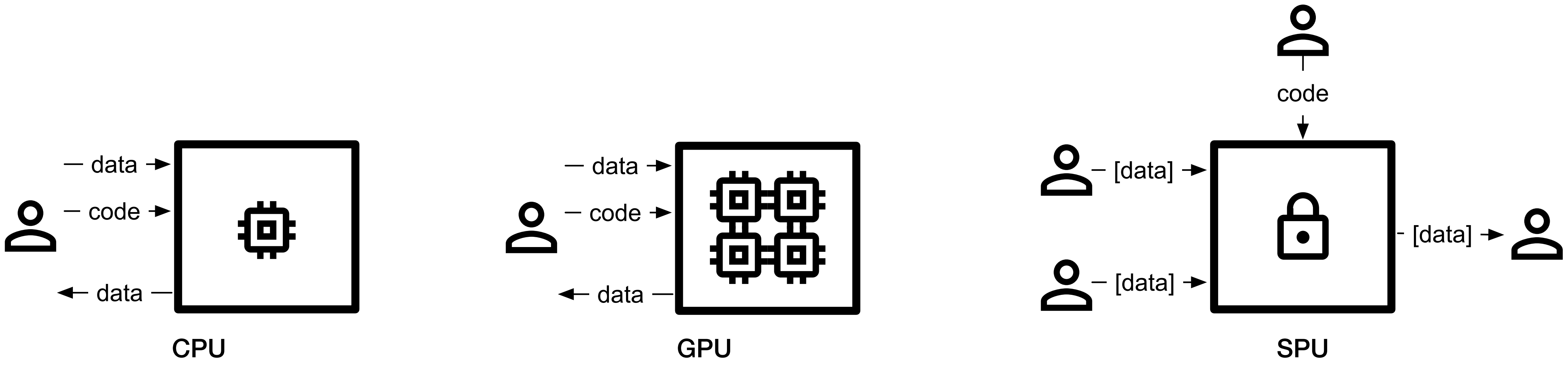
- Virtual
- Secure
- Slow

SPU is a just a virtual device

How to use SPU

03

How to use SPU



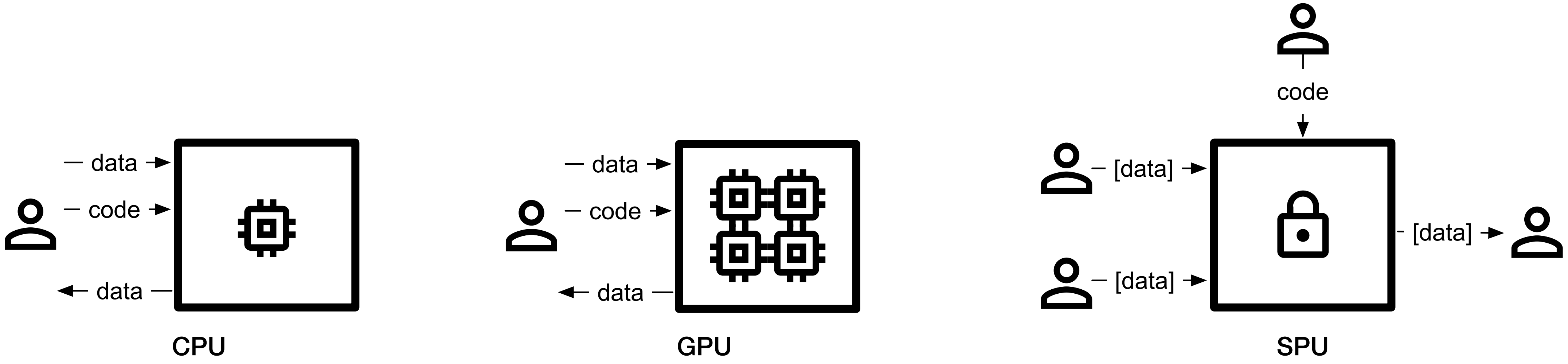
- Physical
- General
- Fast

- Physical
- Parallel
- SuperFast

- Virtual
- Secure
- Slow

SPU is a just a virtual device

How to use SPU



- Physical
- General
- Fast

- Physical
- Parallel
- SuperFast

- Virtual
- Secure
- Slow

SPU is a just a virtual device,
use it like a device.

How to use SPU

- Basic usage
  - Native python/JAX code
  - Just-in-time compilation
  - No MPC knowledge required
- Change protocol
- Runtime stack

```
import numpy as np
import jax.numpy as jnp
import spu.binding.util.distributed as ppd

def rand():
  return np.random.randint(100, size=(1,))

def compare(x, y):
  return jnp.maximum(x, y)

# make a random at P0, unknown to P1.
x = ppd.device("P0")(rand)()

# make a random at P1, unknown to P0.
y = ppd.device("P1")(rand)()

# compare the result at SPU, unknown to P0&P1 z =
ppd.device("SPU")(compare)(x, y)

#reveal the result.
print(f"reveal {ppd.get(z)")
```

How to use SPU

- Basic usage
  - Native python/JAX code
  - Just-in-time compilation
  - No MPC knowledge required
- Change protocol
- Runtime stack

```
import numpy as np
import jax.numpy as jnp
import spu.binding.util.distributed as ppd

def rand():
  return np.random.randint(100, size=(1,))

def compare(x, y):
  return jnp.maximum(x, y)

# make a random at P0, unknown to P1.
x = ppd.device("P0")(rand)()

# make a random at P1, unknown to P0.
y = ppd.device("P1")(rand)()

# compare the result at SPU, unknown to P0&P1
z = ppd.device("SPU")(compare)(x, y)

#reveal the result.
print(f"reveal {ppd.get(z)")
```

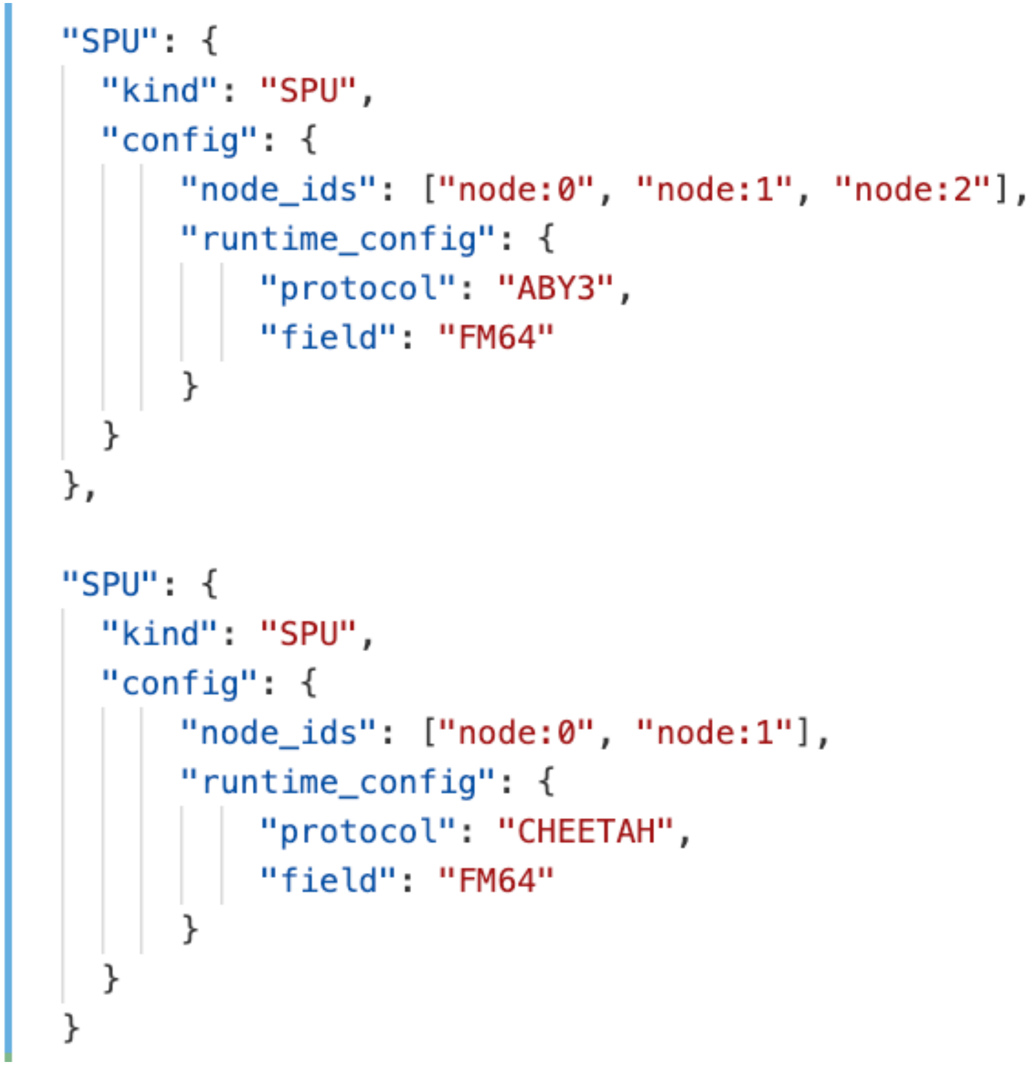
How to use SPU

- Basic usage
  - Native python/JAX code
  - Just-in-time compilation
  - No MPC knowledge required
- Change protocol
- Runtime stack

```
import numpy as np
import jax.numpy as jnp
import spu.binding.util.distributed as ppd

def rand():
  return np.random.randint(100, size=(1,))

def compare(x, y):
  return jnp.maximum(x, y)

# make a random at P0, unknown to P1.
x = ppd.device("P0")(rand)()

# make a random at P1, unknown to P0.
y = ppd.device("P1")(rand)()

# compare the result at SPU, unknown to P0&P1
z = ppd.device("SPU")(compare)(x, y)

#reveal the result.
print(f"reveal {ppd.get(z)")
```

How to use SPU

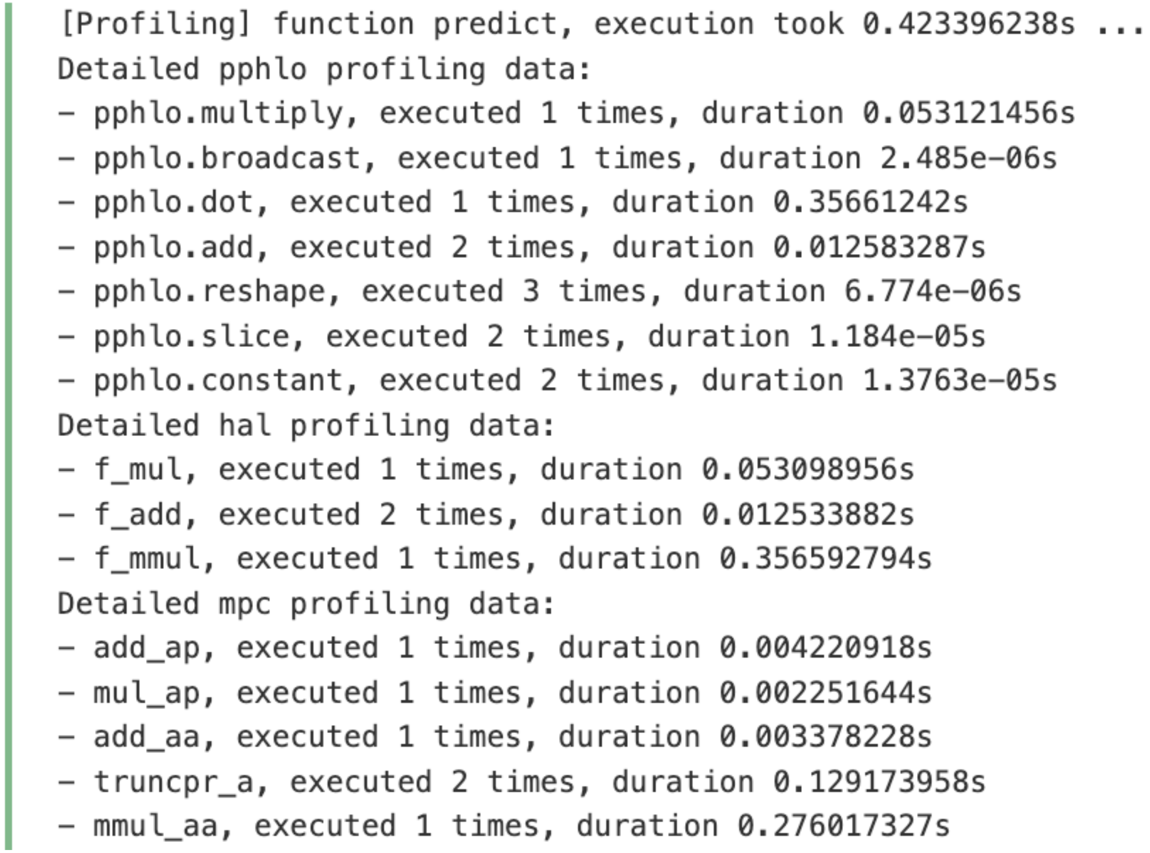
- Basic usage
- Change security setting
  - Configuration only
  - No code change
- Advanced



```
"SPU": {
  "kind": "SPU",
  "config": {
    "node_ids": ["node:0", "node:1", "node:2"],
    "runtime_config": {
      "protocol": "ABY3",
      "field": "FM64"
    }
  }
},

"SPU": {
  "kind": "SPU",
  "config": {
    "node_ids": ["node:0", "node:1"],
    "runtime_config": {
      "protocol": "CHEETAH",
      "field": "FM64"
    }
  }
}
```

How to use SPU

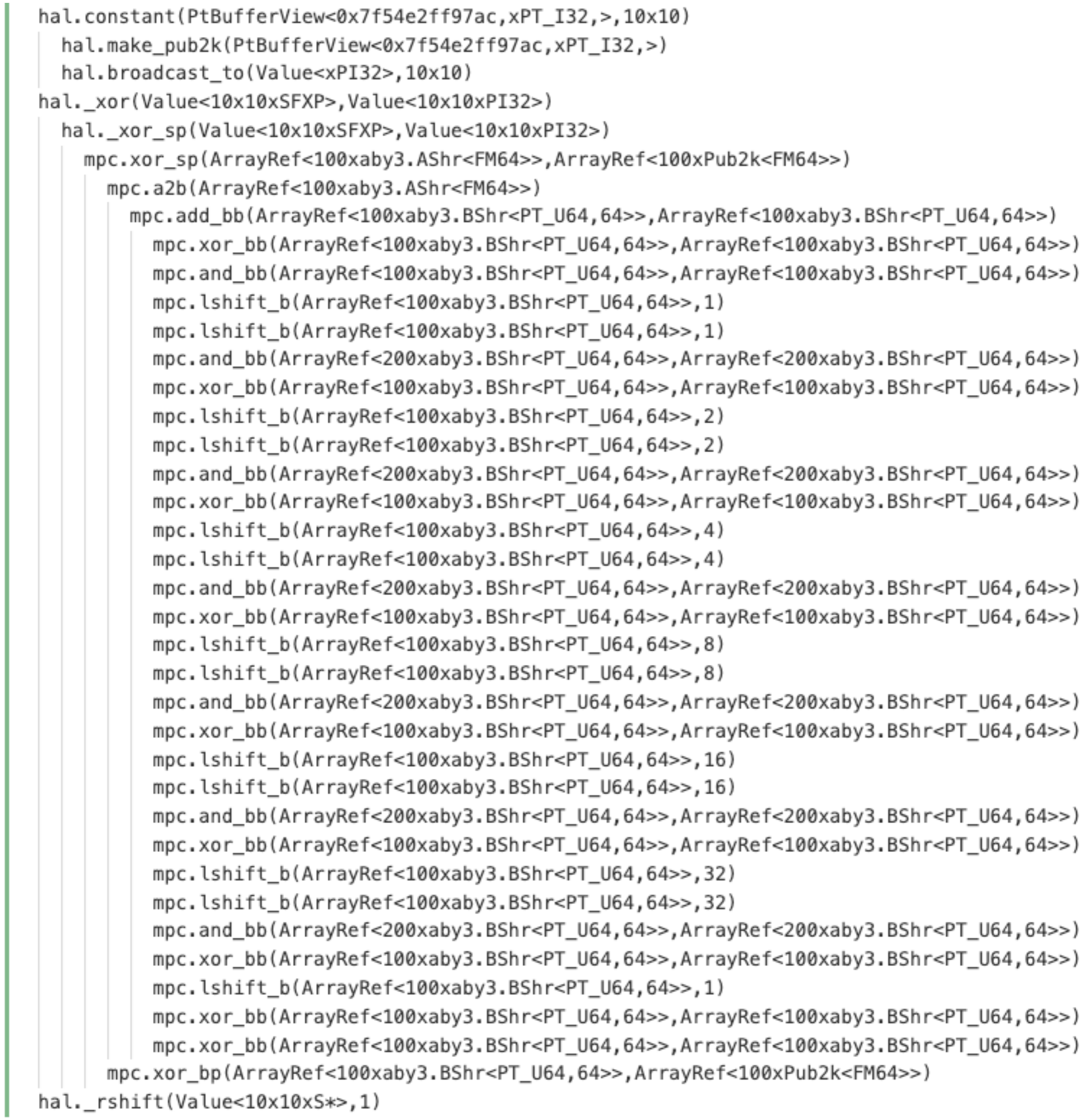
- Basic usage
- Change security setting
- Advanced
  - Profiling – multiple layer data
  - Tracing – full stack trace
  - Debugging – e2e verification



```
[Profiling] function predict, execution took 0.423396238s ...
Detailed pphlo profiling data:
- pphlo.multiply, executed 1 times, duration 0.053121456s
- pphlo.broadcast, executed 1 times, duration 2.485e-06s
- pphlo.dot, executed 1 times, duration 0.35661242s
- pphlo.add, executed 2 times, duration 0.012583287s
- pphlo.reshape, executed 3 times, duration 6.774e-06s
- pphlo.slice, executed 2 times, duration 1.184e-05s
- pphlo.constant, executed 2 times, duration 1.3763e-05s
Detailed hal profiling data:
- f_mul, executed 1 times, duration 0.053098956s
- f_add, executed 2 times, duration 0.012533882s
- f_mmul, executed 1 times, duration 0.356592794s
Detailed mpc profiling data:
- add_ap, executed 1 times, duration 0.004220918s
- mul_ap, executed 1 times, duration 0.002251644s
- add_aa, executed 1 times, duration 0.003378228s
- truncpr_a, executed 2 times, duration 0.129173958s
- mmul_aa, executed 1 times, duration 0.276017327s
```

How to use SPU

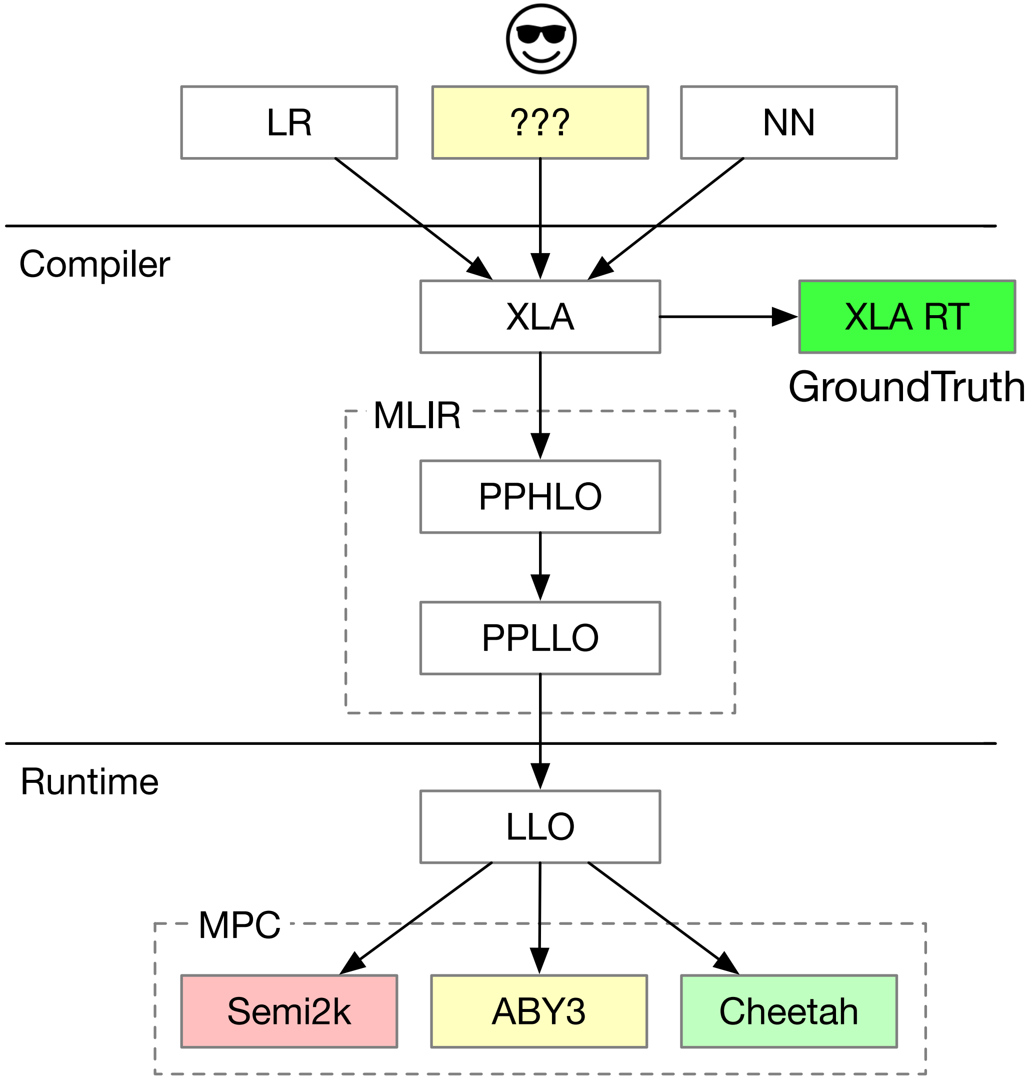
- Basic usage
- Change security setting
- Advanced
  - Profiling - multiple layer data
  - Tracing - full stack trace
  - Debugging - e2e verification



```
hal.constant(PtBufferView<0x7f54e2ff97ac,xPT_I32,>,10x10)
  hal.make_pub2k(PtBufferView<0x7f54e2ff97ac,xPT_I32,>)
  hal.broadcast_to(Value<xPI32>,10x10)
hal._xor(Value<10x10xSFXP>,Value<10x10xPI32>)
  hal._xor_sp(Value<10x10xSFXP>,Value<10x10xPI32>)
    mpc.xor_sp(ArrayRef<100xaby3.AShr<FM64>>,ArrayRef<100xPub2k<FM64>>)
      mpc.a2b(ArrayRef<100xaby3.AShr<FM64>>)
        mpc.add_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.and_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,1)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,1)
          mpc.and_bb(ArrayRef<200xaby3.BShr<PT_U64,64>>,ArrayRef<200xaby3.BShr<PT_U64,64>>)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,2)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,2)
          mpc.and_bb(ArrayRef<200xaby3.BShr<PT_U64,64>>,ArrayRef<200xaby3.BShr<PT_U64,64>>)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,4)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,4)
          mpc.and_bb(ArrayRef<200xaby3.BShr<PT_U64,64>>,ArrayRef<200xaby3.BShr<PT_U64,64>>)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,8)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,8)
          mpc.and_bb(ArrayRef<200xaby3.BShr<PT_U64,64>>,ArrayRef<200xaby3.BShr<PT_U64,64>>)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,16)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,16)
          mpc.and_bb(ArrayRef<200xaby3.BShr<PT_U64,64>>,ArrayRef<200xaby3.BShr<PT_U64,64>>)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,32)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,32)
          mpc.and_bb(ArrayRef<200xaby3.BShr<PT_U64,64>>,ArrayRef<200xaby3.BShr<PT_U64,64>>)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.lshift_b(ArrayRef<100xaby3.BShr<PT_U64,64>>,1)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
          mpc.xor_bb(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xaby3.BShr<PT_U64,64>>)
      mpc.xor_bp(ArrayRef<100xaby3.BShr<PT_U64,64>>,ArrayRef<100xPub2k<FM64>>)
hal._rshift(Value<10x10xS*>,1)
```

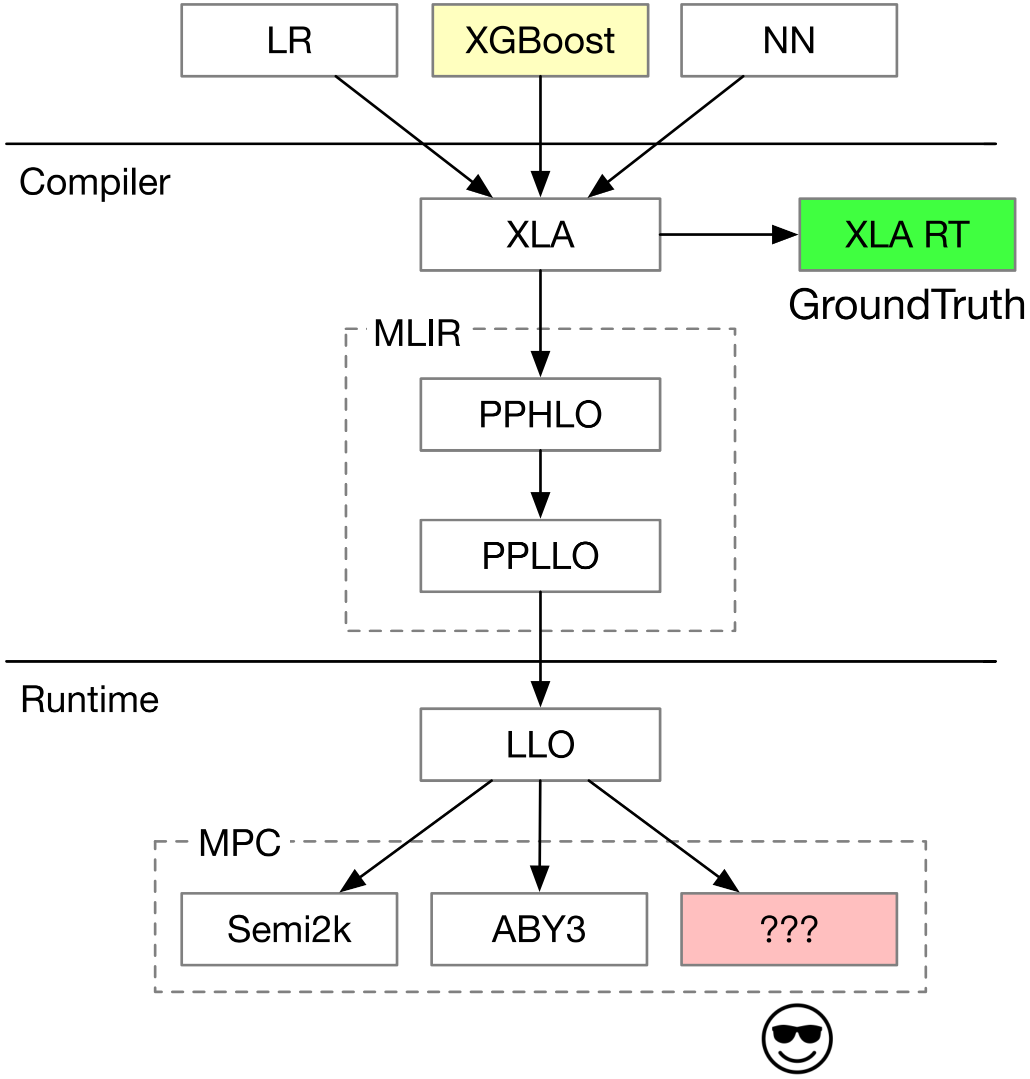
How to use SPU

- Basic usage
- Change security setting
- Advanced
  - Profiling – multiple layer data
  - Tracing – full stack trace
  - Debugging – e2e verification
    - Plaintext as ground truth
    - Step by step verification



How to use SPU

- Basic usage
- Change security setting
- Advanced
  - Profiling – multiple layer data
  - Tracing – full stack trace
  - Debugging – e2e verification
    - Plaintext as ground truth
    - Step by step verification



Reference

- SPU代码：https://github.com/secretflow/spu
- SPU文档：https://spu.readthedocs.io
- Secretflow代码：https://github.com/secretflow
- Secretflow文档：https://secretflow.readthedocs.io

THANKS