Trust? In *my* hardware?
It's more likely than you think.

HEART Computer Security and Privacy for the Modern World

💙 `EN.600.111(33)`

September 30, 2021

Tushar Jois

Trust? In *my* hardware?
It's more likely than you think.

HEART Computer Security and Privacy for the Modern World

🧡 `EN.600.111(26)`

October 6, 2021

Tushar Jois

Recap

- Symmetric encryption prevents evesdroppers from reading a message
- Diffie-Hellman key exchange helps parties agree on a shared secret
- Transport Layer Security (TLS) combines key exchange, encryption, and digital signatures to provide security

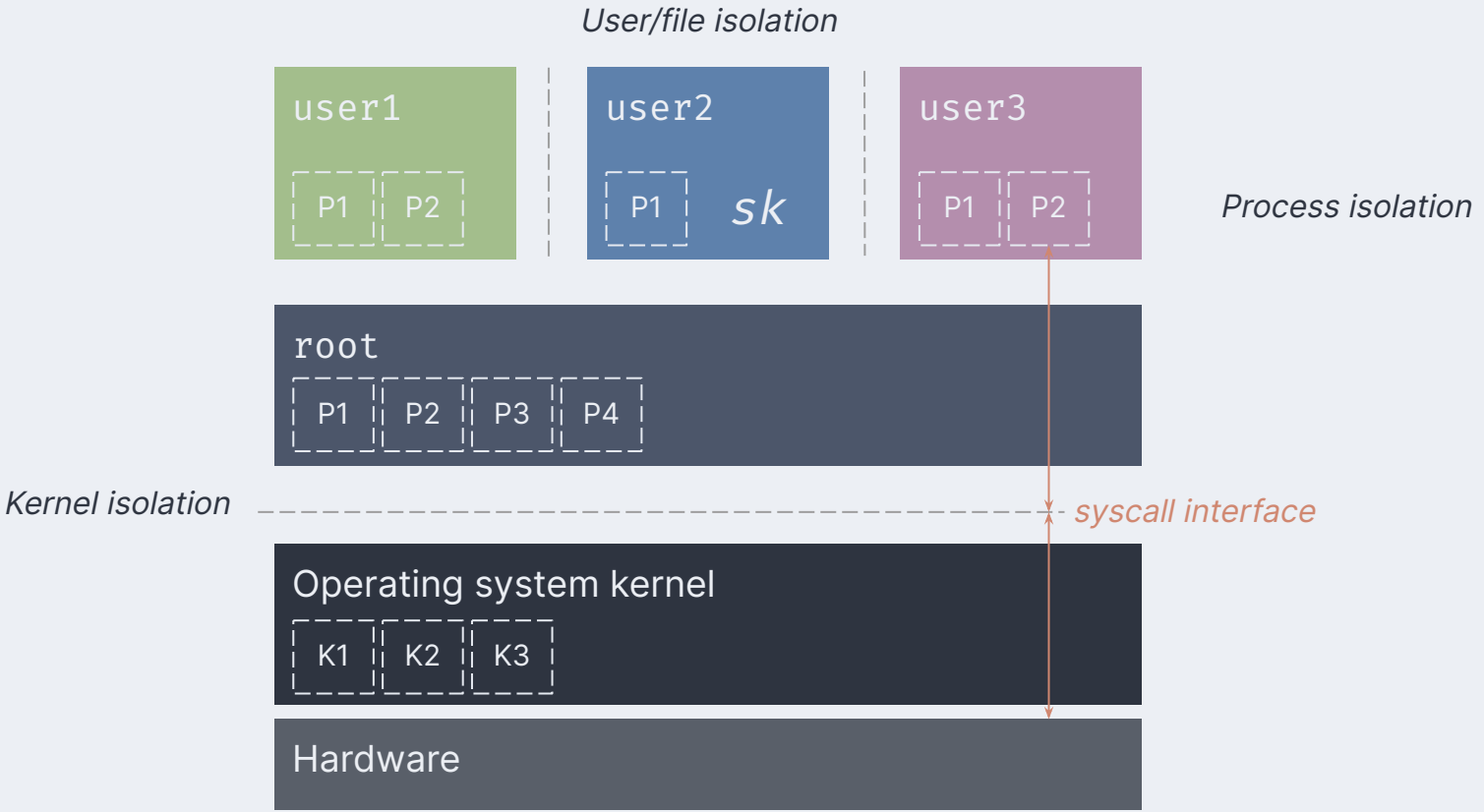
Lesson objectives

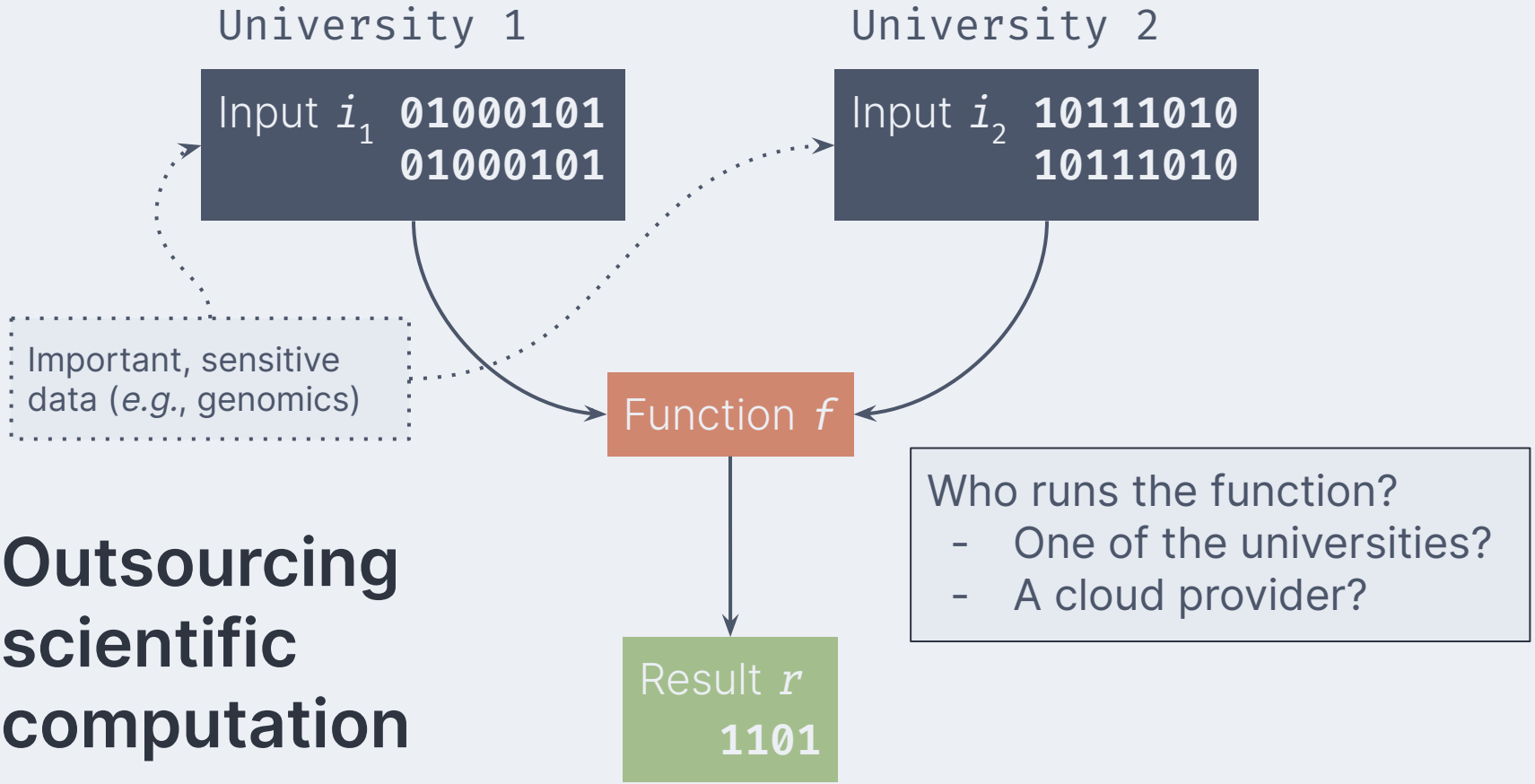
- Explain how trusted hardware can be used to provide secure execution on cloud environments
- Use an execution trace to identify microarchitectural side-channels
- Apply data-oblivious computation to close side-channels

Exit ticket from last time

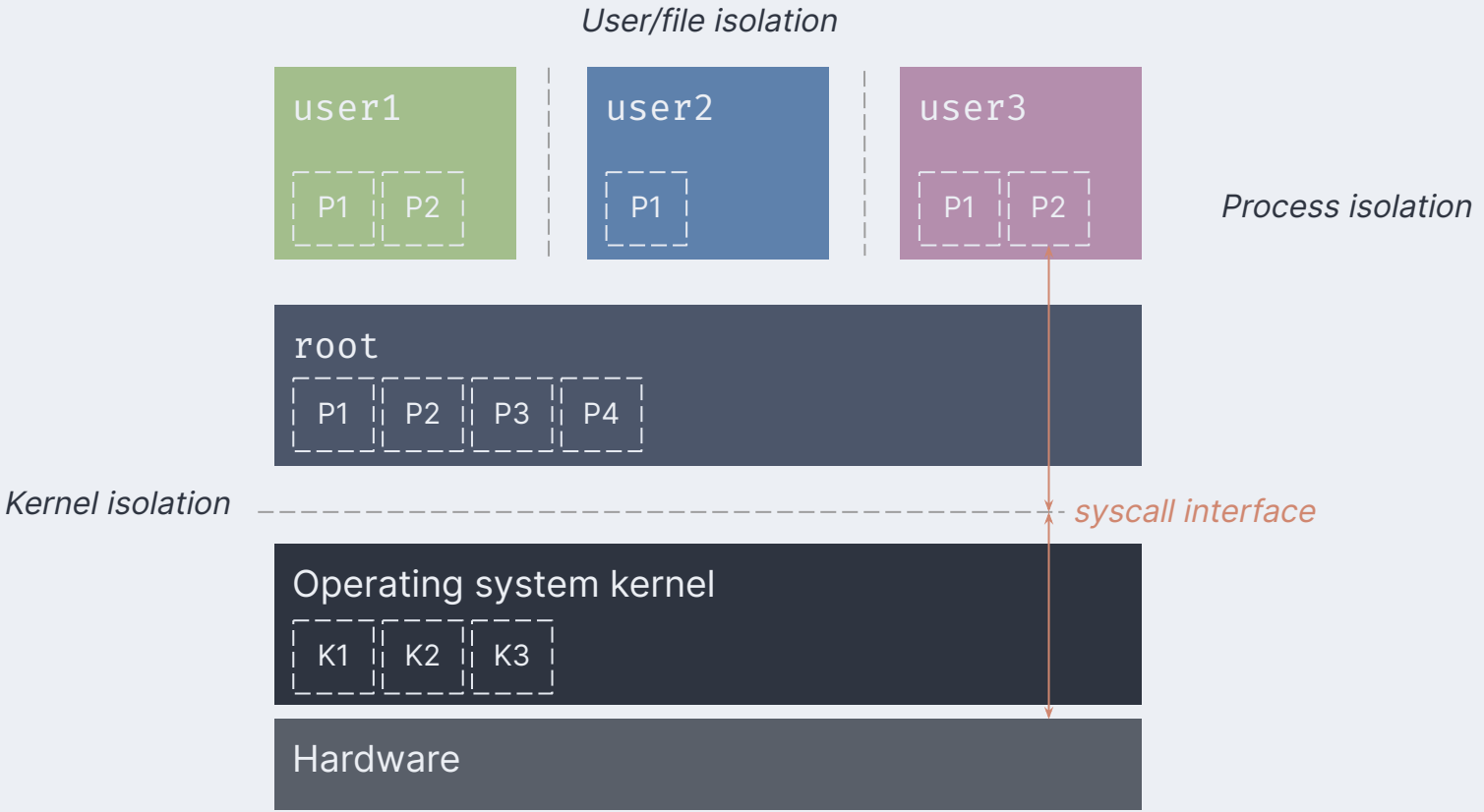
Cryptography requires the use of a secret key for encryption and signing. How can we use Unix systems security to protect this secret key from being attacked/leaked?

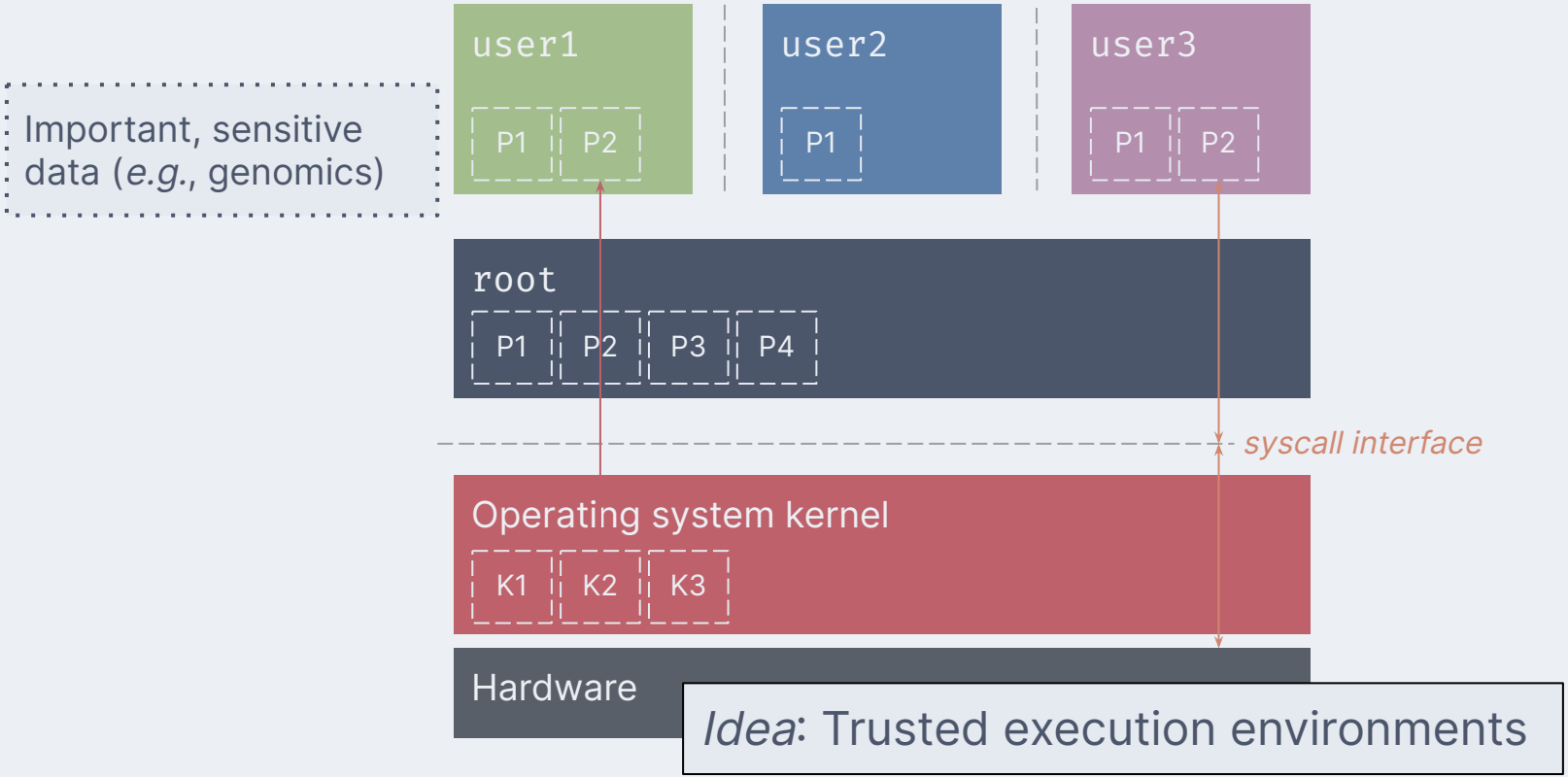
Unix isolation


User/file isolation
user1
P1
P2
user2
P1
sk
user3
P1
P2
Process isolation
root
P1
P2
P3
P4
Kernel isolation
syscall interface
Operating system kernel
K1
K2
K3
Hardware



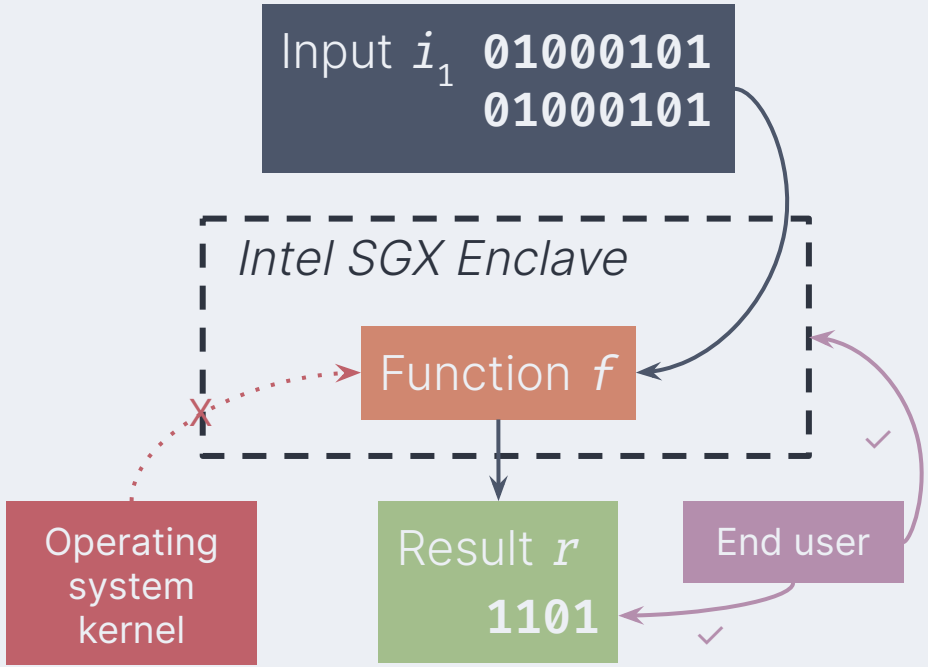
Outsourcing scientific computation

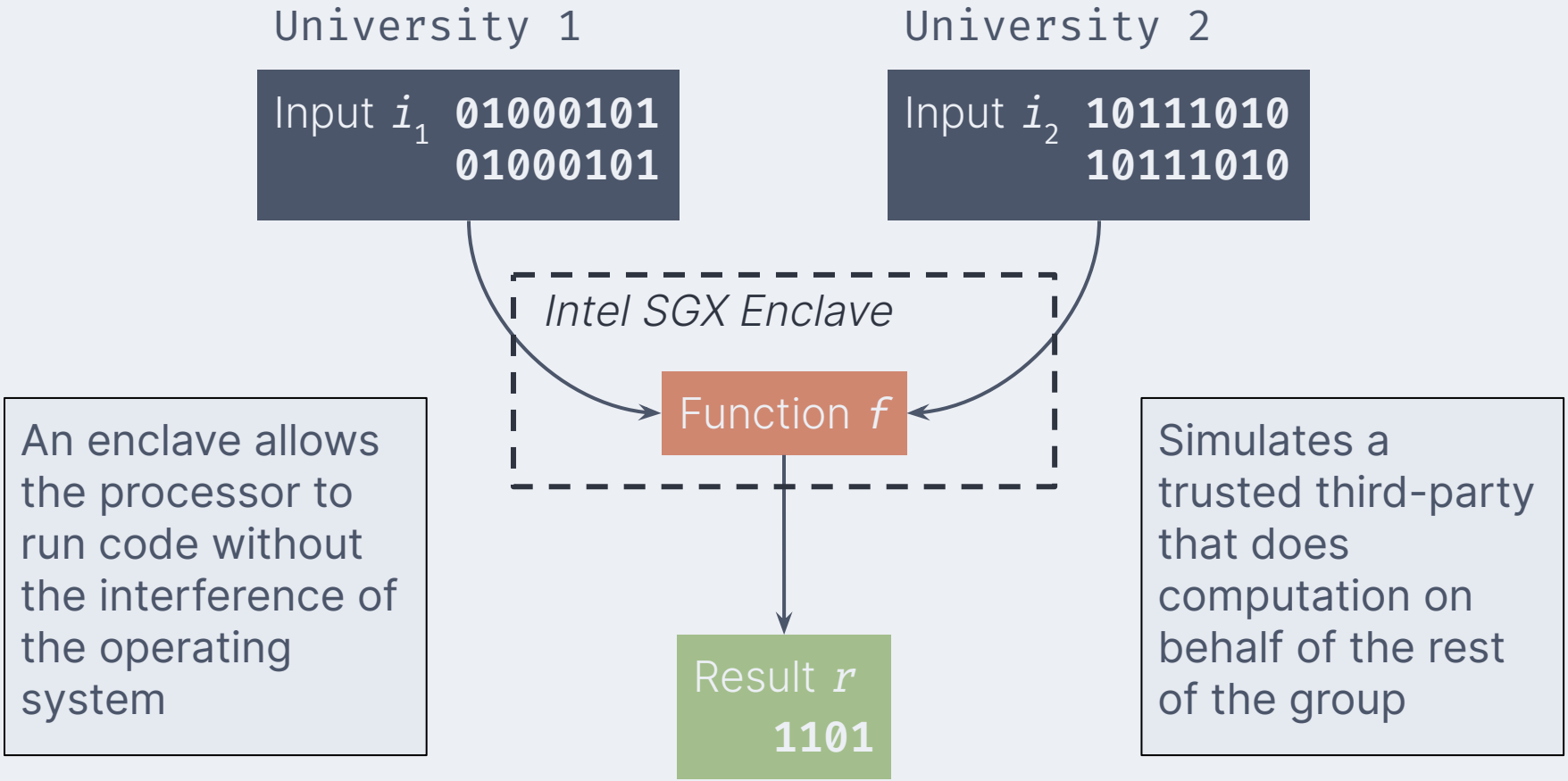

User/file isolation
user1
P1
P2
user2
P1
user3
P1
P2
Process isolation
root
P1
P2
P3
P4
Kernel isolation
syscall interface
Operating system kernel
K1
K2
K3
Hardware


Important, sensitive data (*e.g.*, genomics)
user1
P1
P2
user2
P1
user3
P1
P2
root
P1
P2
P3
P4
syscall interface
Operating system kernel
K1
K2
K3
Hardware
Idea: Trusted execution environments

Trusted execution environment

- A separate “part” of the processor
  - Untrusted applications run normally
  - Trusted applications run in an *enclave*
- When a processor computes on an enclave, it encrypts its memory
  - The operating system can’t access the memory/data unless it has the processor’s key
- Enclaves can use *remote attestation*
  - The processor signs what it’s executing
  - You can be sure nothing has changed
- Feature requires hardware support
  - Combines cryptography and systems security together

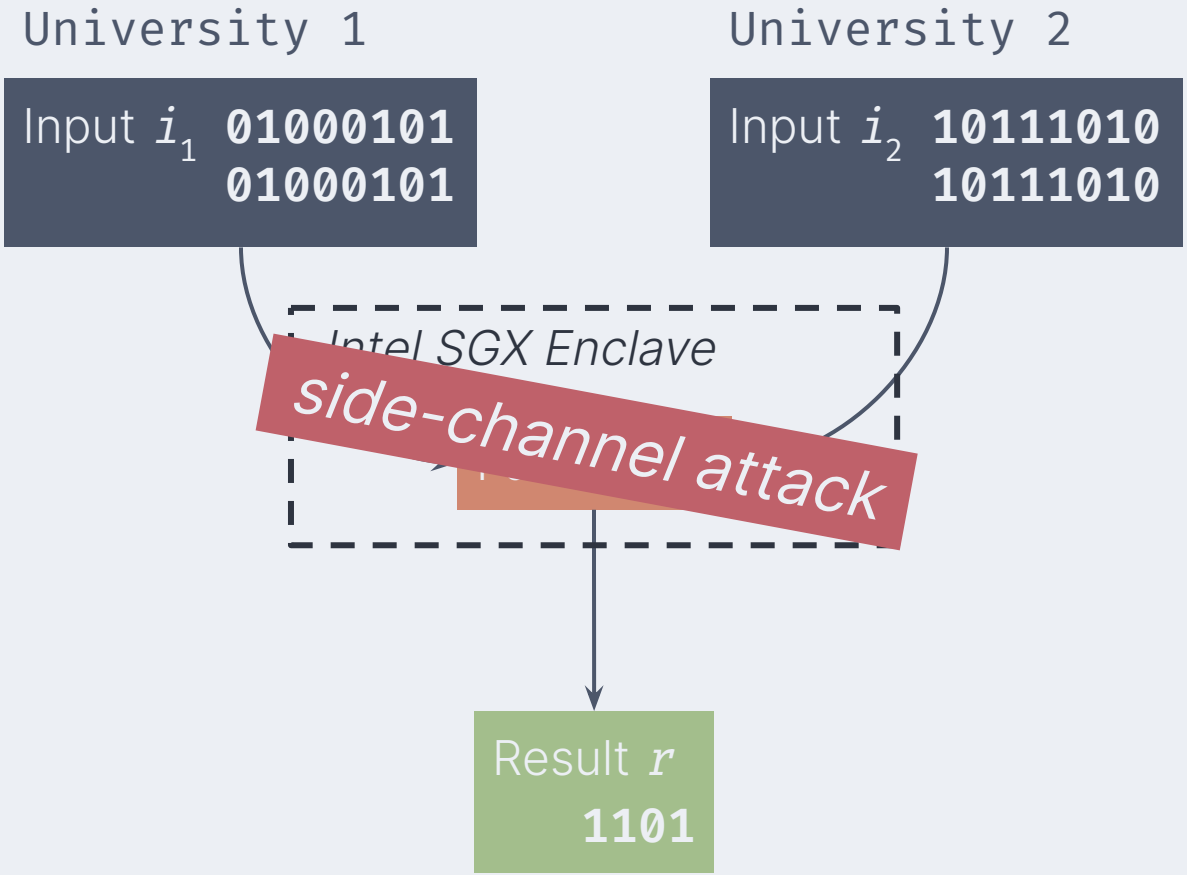



University 1
Input i_1 01000101 01000101
University 2
Input i_2 10111010 10111010
Intel SGX Enclave
Function f
Result r 1101
An enclave allows the processor to run code without the interference of the operating system
Simulates a trusted third-party that does computation on behalf of the rest of the group

minimizing trust

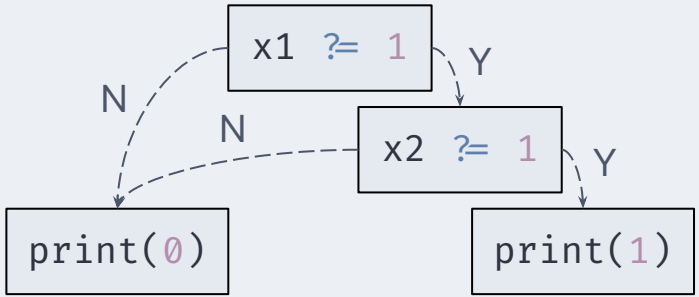
Trusted hardware

Utilize features of hardware to provide additional security beyond traditional software mechanisms such as operating system isolation.


University 1
Input i_1 01000101 01000101
University 2
Input i_2 10111010 10111010
Intel SGX Enclave
side-channel attack
Result r 1101

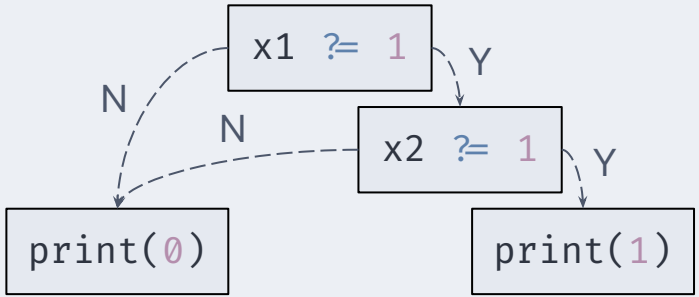
```python
# Side-channel Vulnerable
# Assume x1, x2 are private

if x1 and x2:
  print(1)
else:
  print(0)
```

Execution Trace



```
# Side-channel Vulnerable
# Assume x1, x2 are private

if x1 and x2:
  print(1)
else:
  print(0)
```

Execution Trace



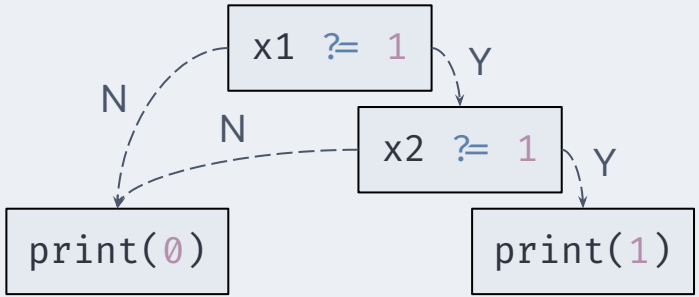
microarchitectural side-channel

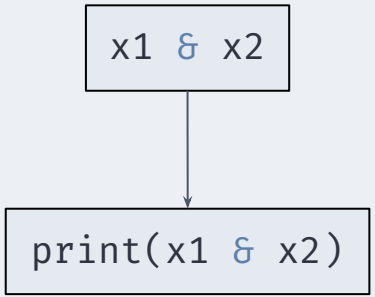
An attacker can infer the inputs of an operation by looking at the state of the processor during its execution.

```
# Side-channel Vulnerable
# Assume x1, x2 are private

if x1 and x2:
  print(1)
else:
  print(0)
```

```
# Fixing the vulnerability

print(x1 & x2)
```

Execution Trace





minimizing trust

Data-oblivious computation

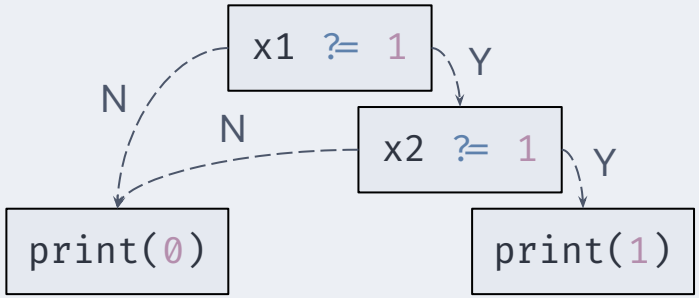
When working with secure hardware, write a program execution with the same observable characteristics regardless of the inputs provided, thus preventing microarchitectural side channel attacks.

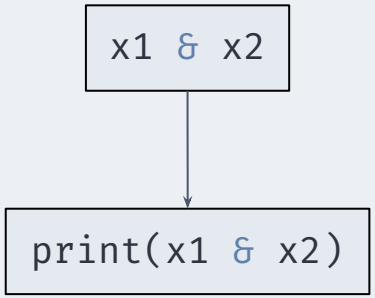
```python
# Side-channel Vulnerable
# Assume x1, x2 are private

if x1 and x2:
  print(1)
else:
  print(0)
```

```python
# Fixed (under assumptions)


print(x1 & x2)
```

Let's try it!

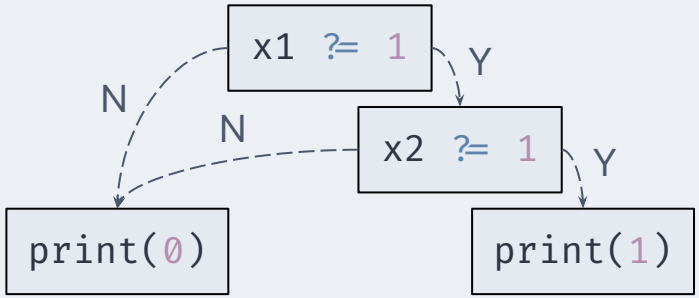
Execution Trace





```python
# Side-channel Vulnerable
# Assume x1, x2 are private

if x1 and x2:
  print(1)
else:
  print(0)
```

Execution Trace



For each function on the right:

- Draw the execution trace on the whiteboard
- Determine if the function is data-oblivious using the execution trace

```python
def func1(x):
    y = 4
    if x == 6:
        y = 6
    return y
```

```python
def func2(x):
    y = 0
    if x == 6:
        y = 6
    else:
        y = 4
    return y
```

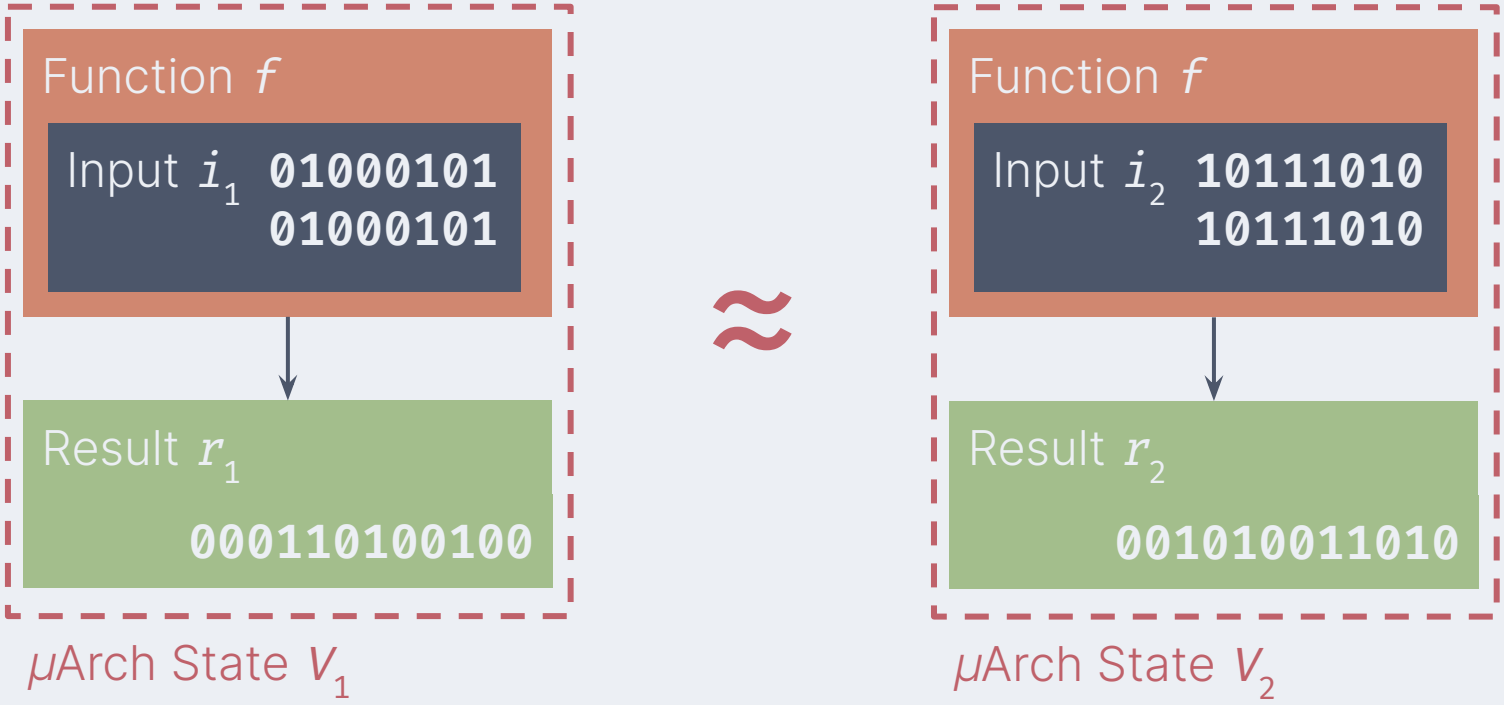
```python
def func3(x):
    y = 1
    if x == 4:
        y = 7
    else:
        y = 0
    return y
```

Key insight:

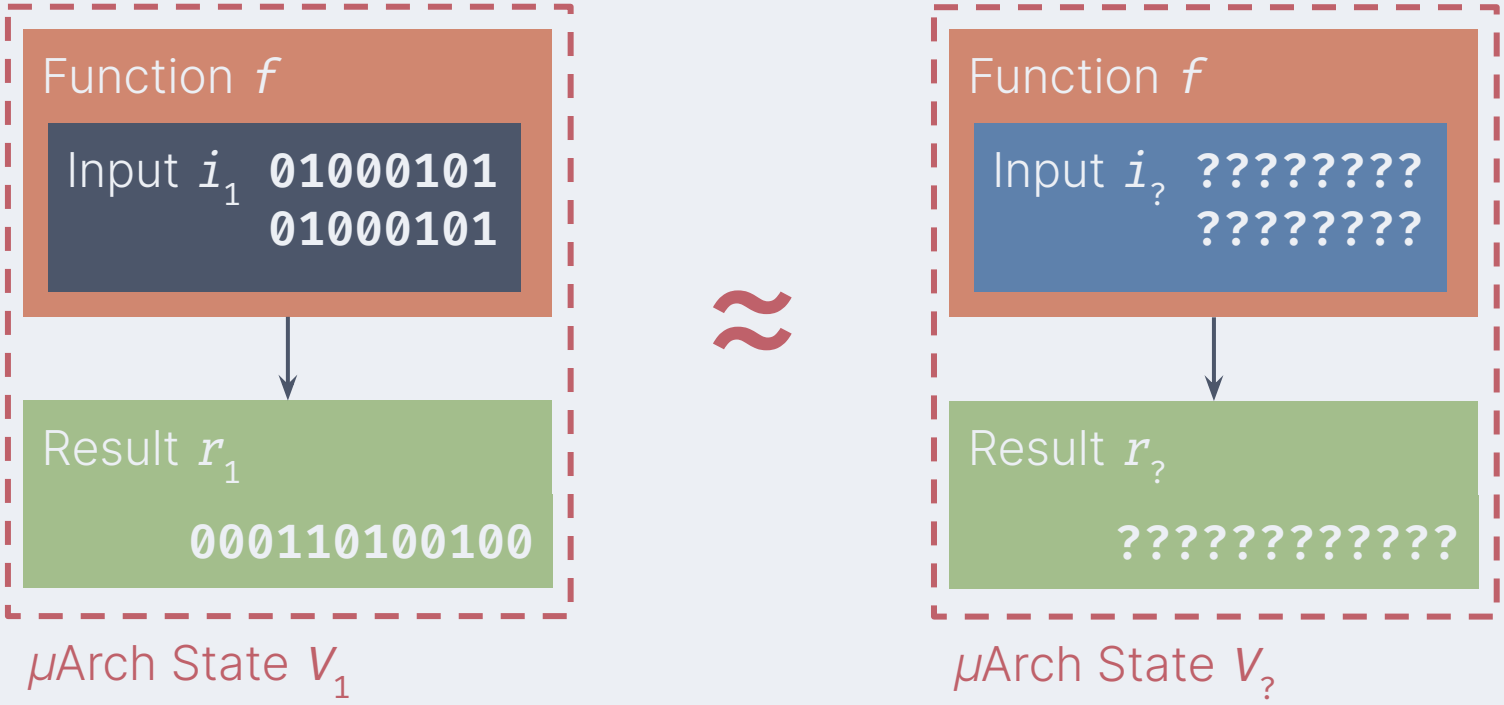
data-oblivious → data-unnecessary

Assuming a **data-oblivious** function

Assuming a **data-oblivious** function



Assuming a **data-oblivious** function



DOVE: A Data-Oblivious Virtual Environment

Hyun Bin Lee, Tushar M. Jois, Christopher W. Fletcher, Carl A. Gunter

Network and Distributed Security Symposium 2021 (NDSS '21)

Function f

Input i_1 **01000101 01000101**

Result r_1 **000110100100**

?????? ????????

Data Oblivious Transcript

Input i_1 **01000101 01000101**

Separate the execution of a script from the actual operations on sensitive data

Replay Execution Trace on a Dataset

Result r_1 **000110100100**

Looking ahead

- We’re going to pivot from talking about security to talking about privacy
- Do the reading: “Tor Overview” article and video
  - If you can, skim through the optional “Meteor: Cryptographically Secure Steganography for Realistic Distributions”
- Exit ticket before class is done

Lesson objectives

- Explain how hardware can be used to provide trusted execution environments
- Use an execution trace to identify microarchitectural side-channels
- Apply data-oblivious computation to close side-channels