HasTEE$^+$– Confidential Computing with Haskell

Abhiroop Sarkar
Chalmers University, Gothenburg

HasTEE: Programming Trusted Execution Environments with Haskell

Abhiroop Sarkar
Chalmers University
Gothenburg, Sweden
sarkara@chalmers.se

Robert Krook
Chalmers University
Gothenburg, Sweden
krookr@chalmers.se

Alejandro Russo
Chalmers University
Gothenburg, Sweden
russo@chalmers.se

Koen Claessen
Chalmers University
Gothenburg, Sweden
koen@chalmers.se

Abstract

Trusted Execution Environments (TEEs) are hardware en-

Keywords: Trusted Execution Environment, Haskell, Intel SGX, Enclave

Haskell Symposium '23

HasTEE$^+$: Confidential Cloud Computing and Analytics with Haskell

Abhiroop Sarkar[0000−0002−8991−9472] and Alejandro Russo[0000−0002−4338−6316]

Chalmers University, Gothenburg, Sweden
{sarkara,russo}@chalmers.se

Abstract. Confidential computing is a security paradigm that enables the protection of confidential code and data in a co-tenanted cloud de-

ESORICS '24
(Under submission)


CHALMERS

CHALMERS

John Hughes

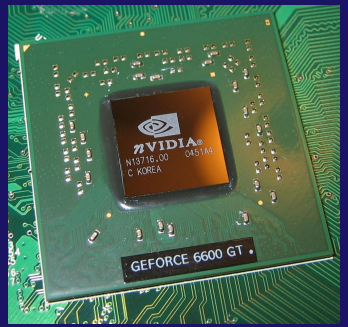
Lennart Augustsson

Thomas Johnsson

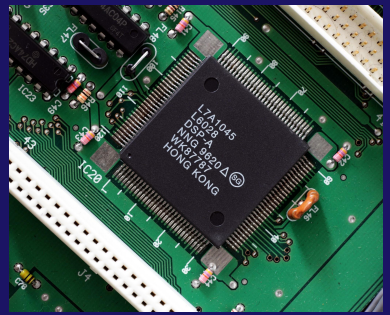
Mary Sheeran

(E)DSLs

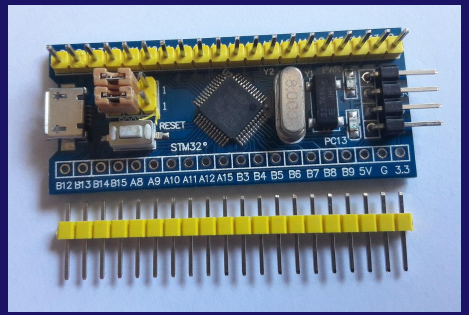
Obsidian



Feldspar



SynchronVM*



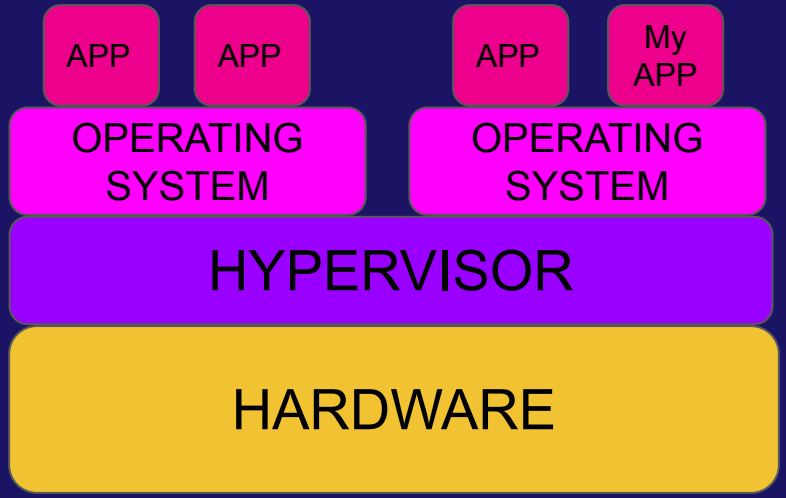
?

arm
TRUSTZONE

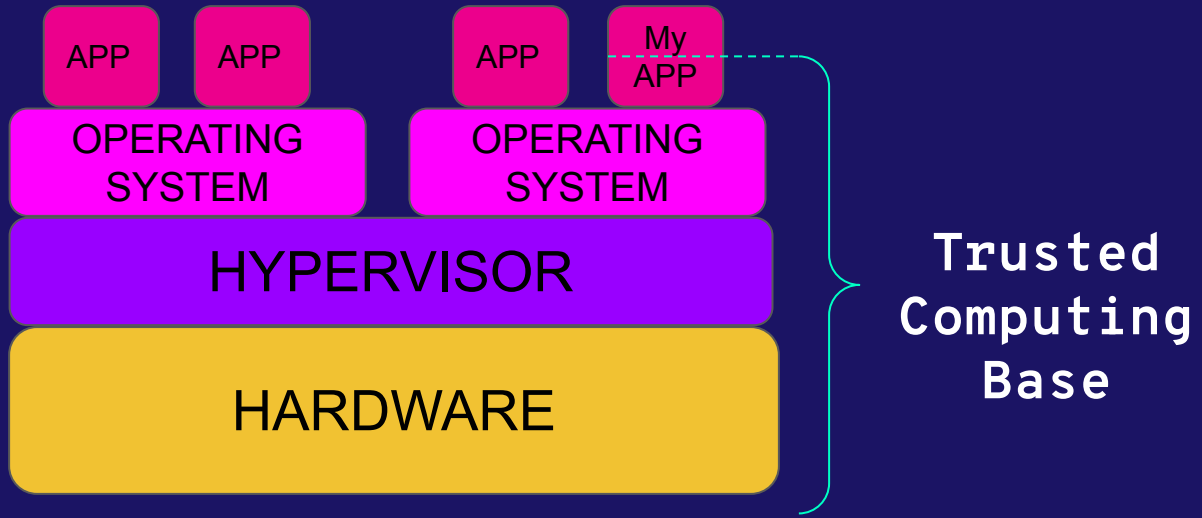
intel
SGX

Trusted Execution Environments

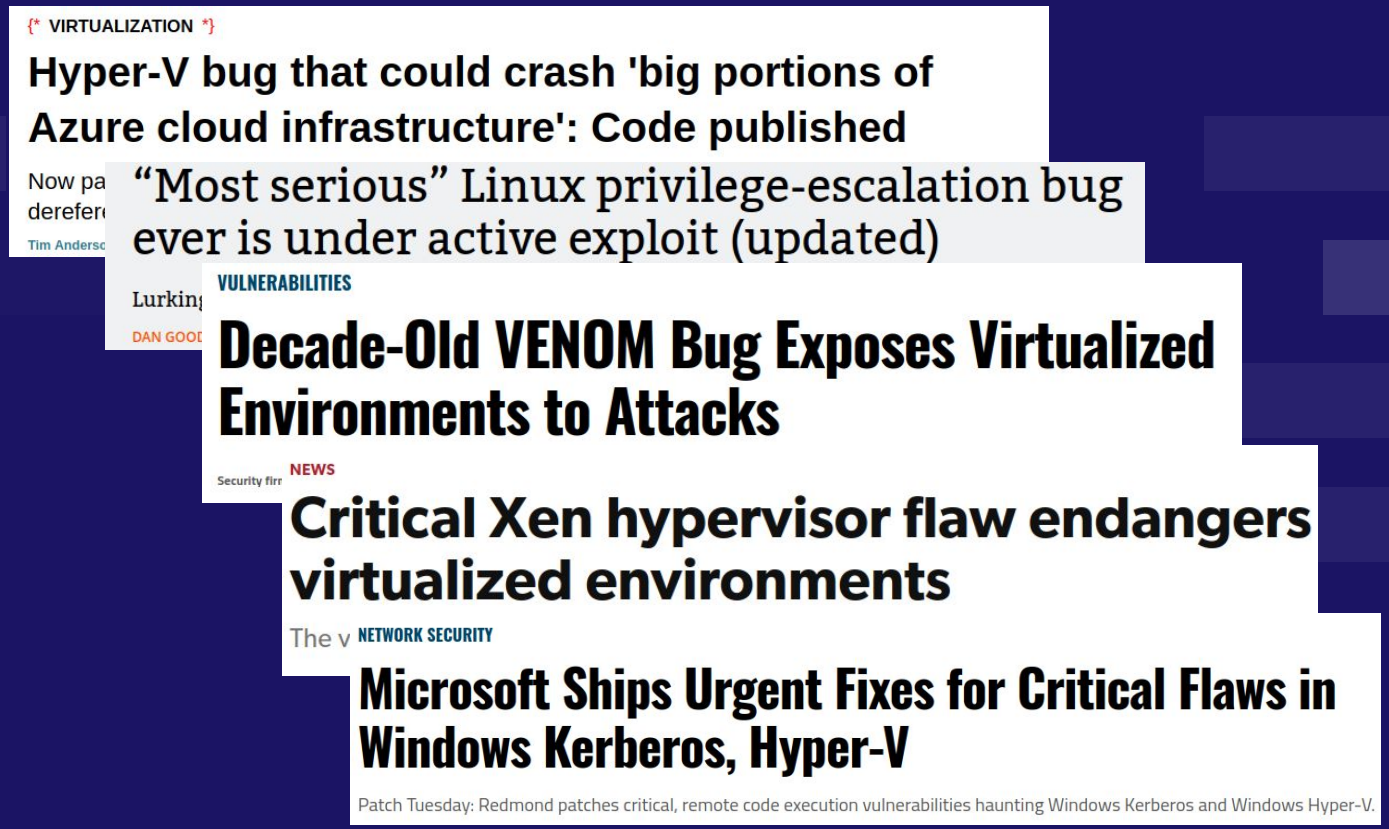
Cloud Deployments



Cloud Deployments



Hypervisor Vulnerabilities



OS Vulnerabilities

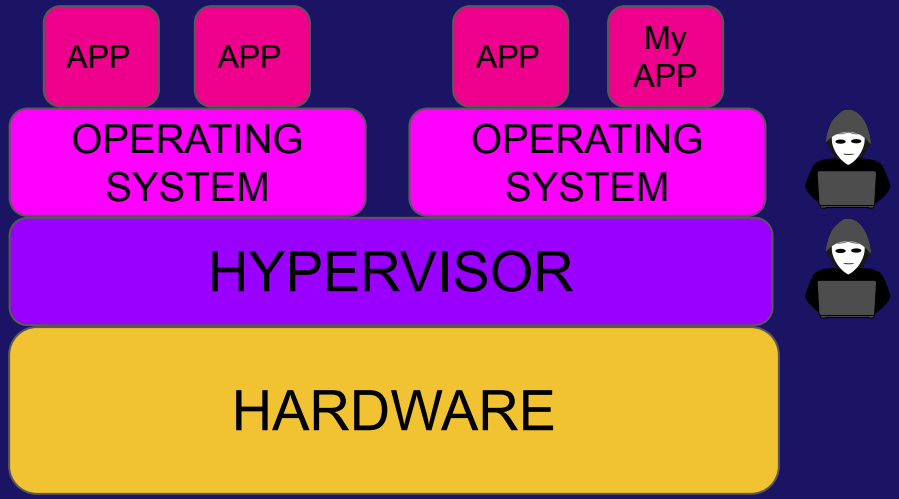
Vulnerability	Total	core	drivers	net	fs	sound
Missing pointer check	8	4	3	1	0	0
Missing permission check	17	3	1	2	11	0
Buffer overflow	15	3	1	5	4	2
Integer overflow	19	4	4	8	2	1
Uninitialized data	29	7	13	5	2	2
Null dereference	20	9	3	7	1	0
Divide by zero	4	2	0	0	1	1
Infinite loop	3	1	1	1	0	0
Data race / deadlock	8	5	1	1	1	0
Memory mismanagement	10	7	1	1	0	1
Miscellaneous	8	2	0	4	2	0
Total	141	47	28	35	24	7

Figure 2: Vulnerabilities (rows) vs. locations (columns).

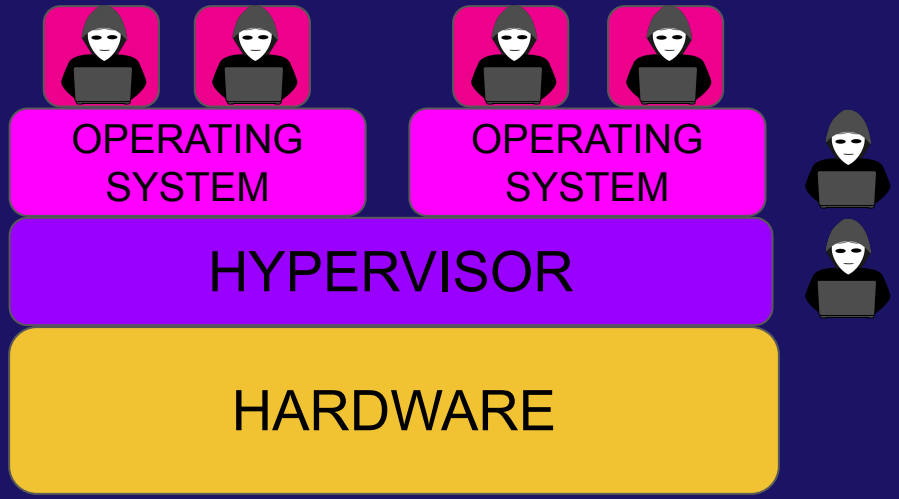
Characterizing hypervisor vulnerabilities in cloud computing servers. Perez-Botero et al. In *Proceedings of the 2013 international workshop on Security in cloud computing*.

Linux kernel vulnerabilities: State-of-the-art defenses and open problems. Mao et al. In *Proceedings of the Second Asia-Pacific Workshop on Systems* (pp. 1-5).

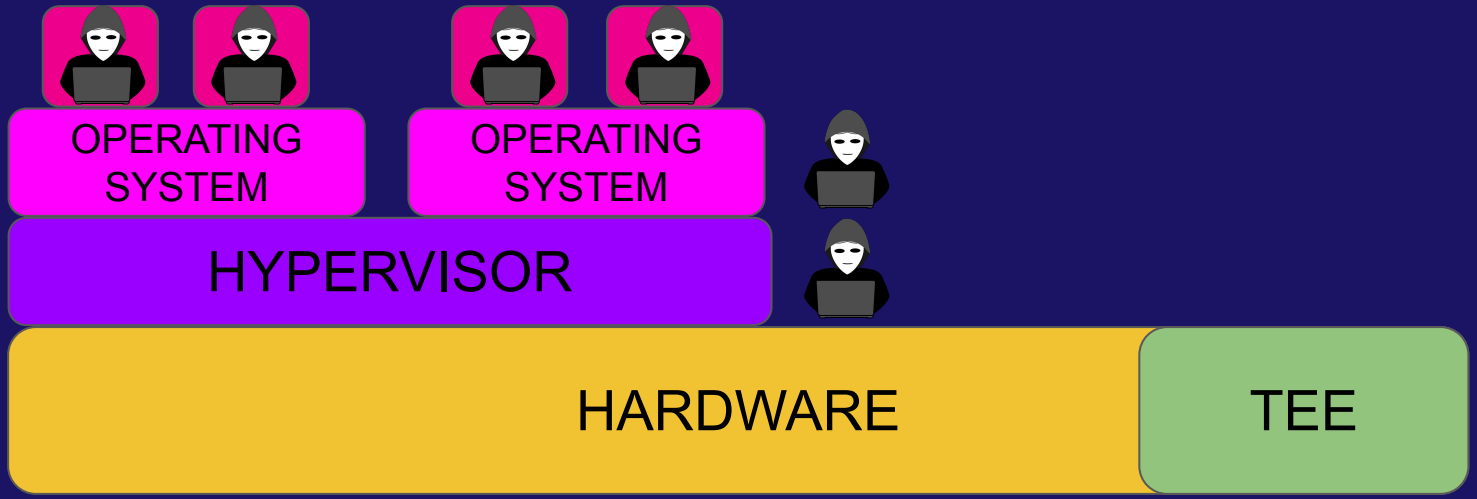
Cloud Deployments



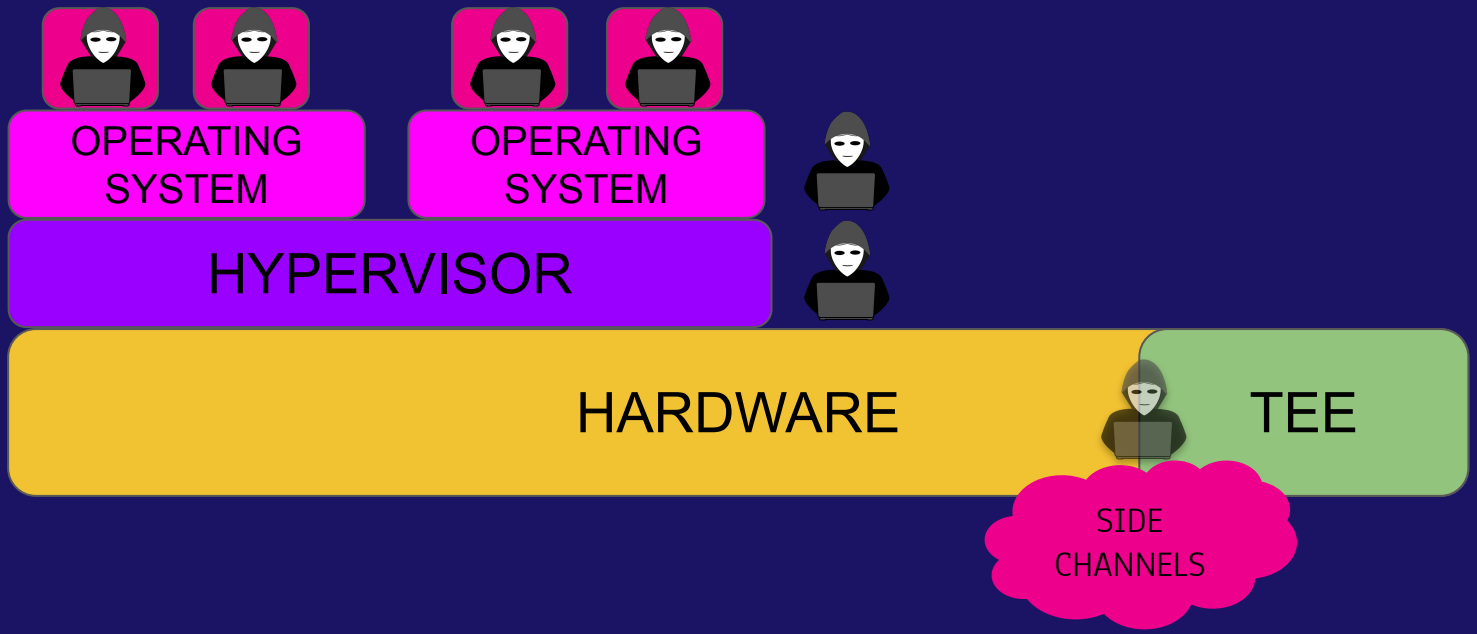
Cloud Deployments



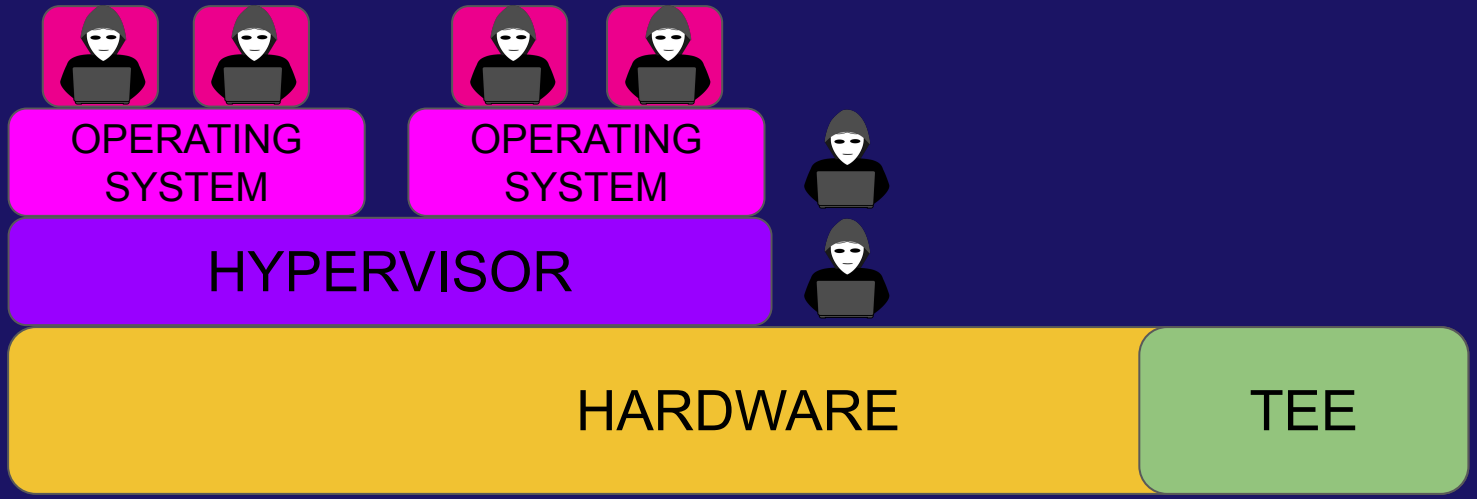
Trusted Execution Environment (TEE)



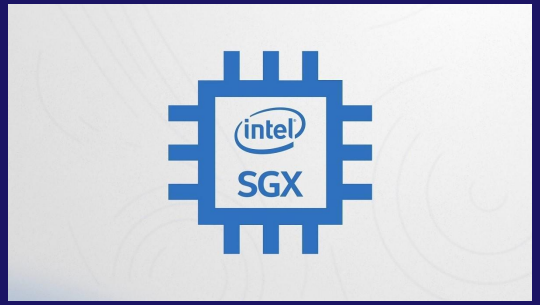
Trusted Execution Environment (TEE)



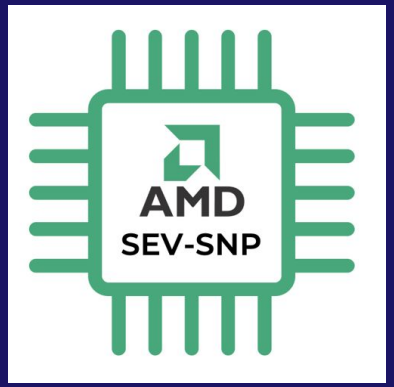
Trusted Execution Environment (TEE)



Trusted Execution Environment (TEE)



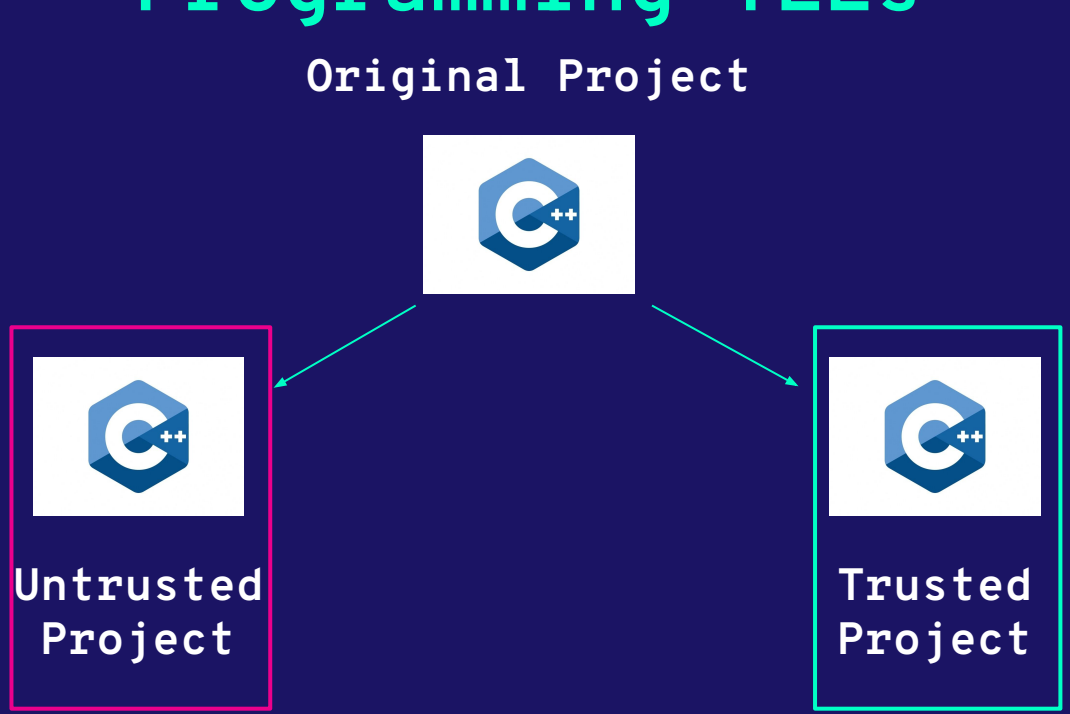
arm
TRUSTZONE



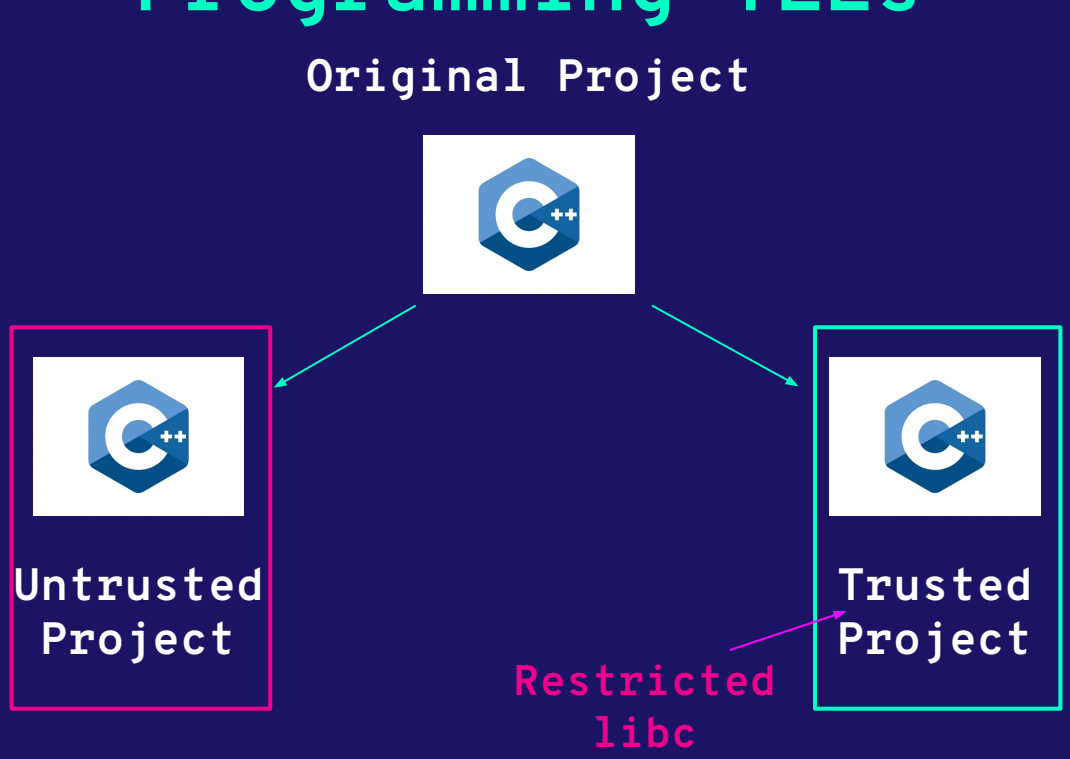
Physical Memory Protection

Programming TEEs

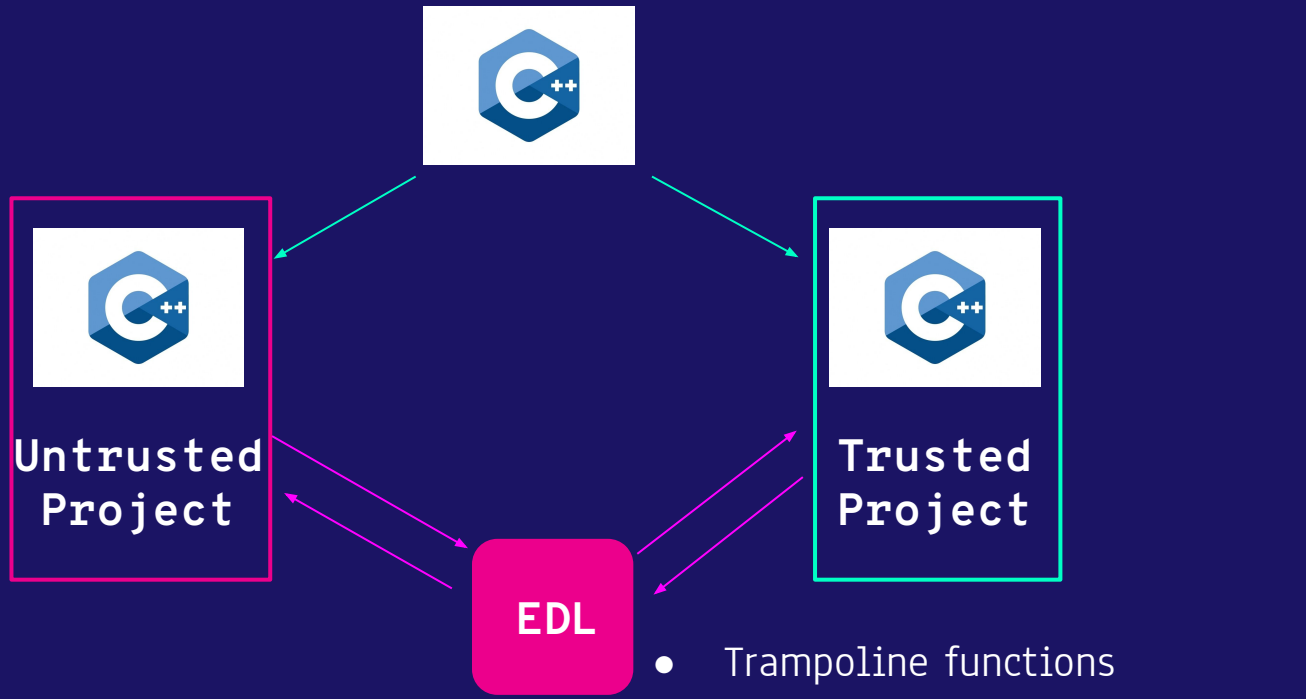
Programming TEEs



Programming TEEs



Programming TEEs



- Trampoline functions
- Arcane Makefiles
- Complex data copying protocol

Programming TEEs

You are passing a double pointer, it is, a pointer to pointer to char (`char **`).

While marshaling/unmarshaling pointers, the EDL Processor processes (copies and validates input and output) only the first level of indirection, it's up to the developer to handle the additional levels of indirection. Hence, for an array of pointers it will only copy the first array of pointers, not the pointed values, copying them is the developer's responsibility. 😢

Source: stackoverflow.com

Secure Program Partitioning

STEVE ZDANCEWIC, LANTIAN ZHENG, NATHANIEL NYSTROM, and ANDREW C. MYERS
Cornell University

Language Support for Secure Software Development with Enclaves

Aditya Oak, *TU Darmstadt*
Amir M. Ahmadian, *KTH Royal Institute of Technology*
Musard Balliu, *KTH Royal Institute of Technology*
Guido Salvaneschi, *University of St.Gallen*

First, **seamless integration** of enclave programming into software applications remains challenging. For example, Intel provides a C/C++ interface to the SGX enclave but no direct support is available for managed languages. As managed languages like Java and Scala are extensively used for developing distributed applications, developers need to either interface their programs with the C++ code executing in the enclave (e.g., using the Java Native Interface [12]) or compile

PtrSplit: Supporting General Pointers in Automatic Program Partitioning

Shen Liu
The Pennsylvania State University
University Park, PA
sxl463@cse.psu.edu

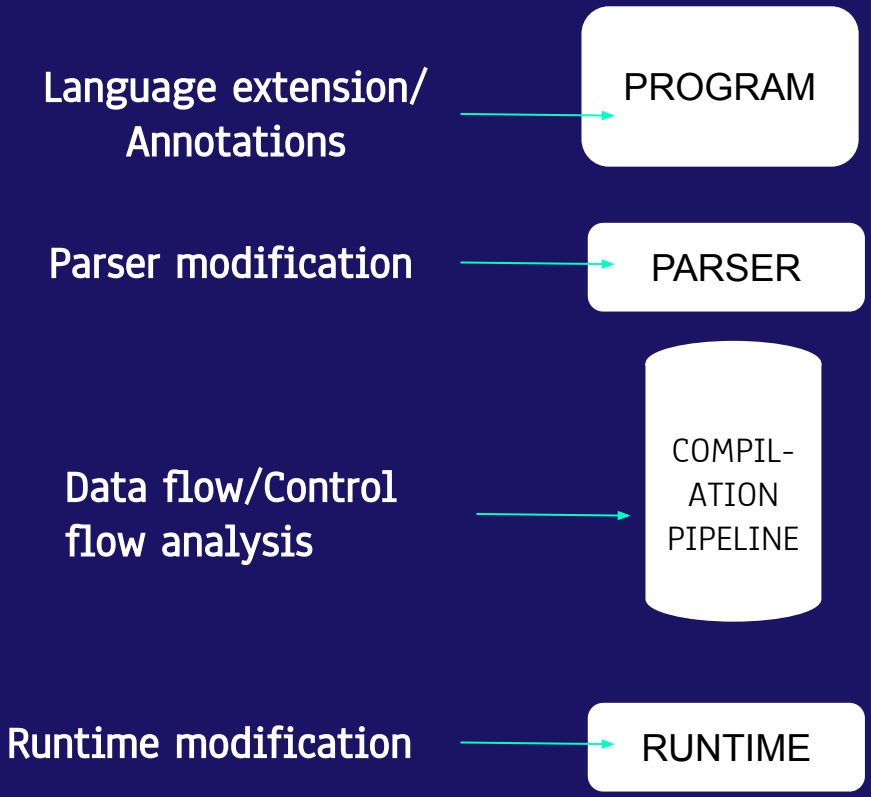
Gang Tan
The Pennsylvania State University
University Park, PA
gtan@

Trent Jaeger
The Pennsylvania State University
University Park, PA

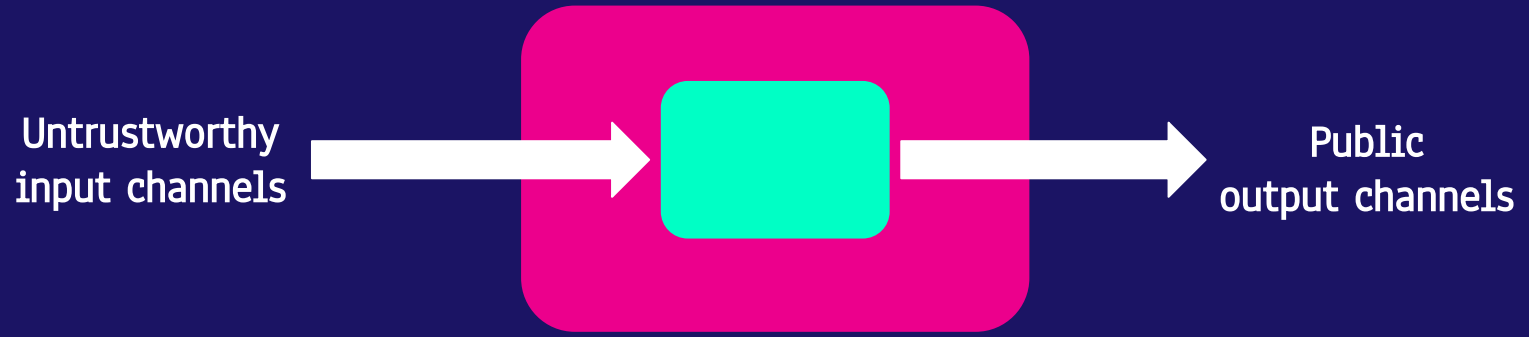
Glamdring: Automatic Application Partitioning for Intel SGX

Joshua Lind, *Imperial College London*
Christian Priebe, *Imperial College London*
Divya Muthukumaran, *Imperial College London*
Dan O'Keeffe, *Imperial College London*
Pierre-Louis Aublin, *Imperial College London*
Florian Kelbert, *Imperial College London*
Tobias Reiher, *TU Dresden*
David Goltzsche, *TU Braunschweig*
David Eyers, *University of Otago*
Rüdiger Kapitza, *TU Braunschweig*
Christof Fetzer, *TU Dresden*
Peter Pietzuch, *Imperial College London*

PROGRAM PARTITIONING

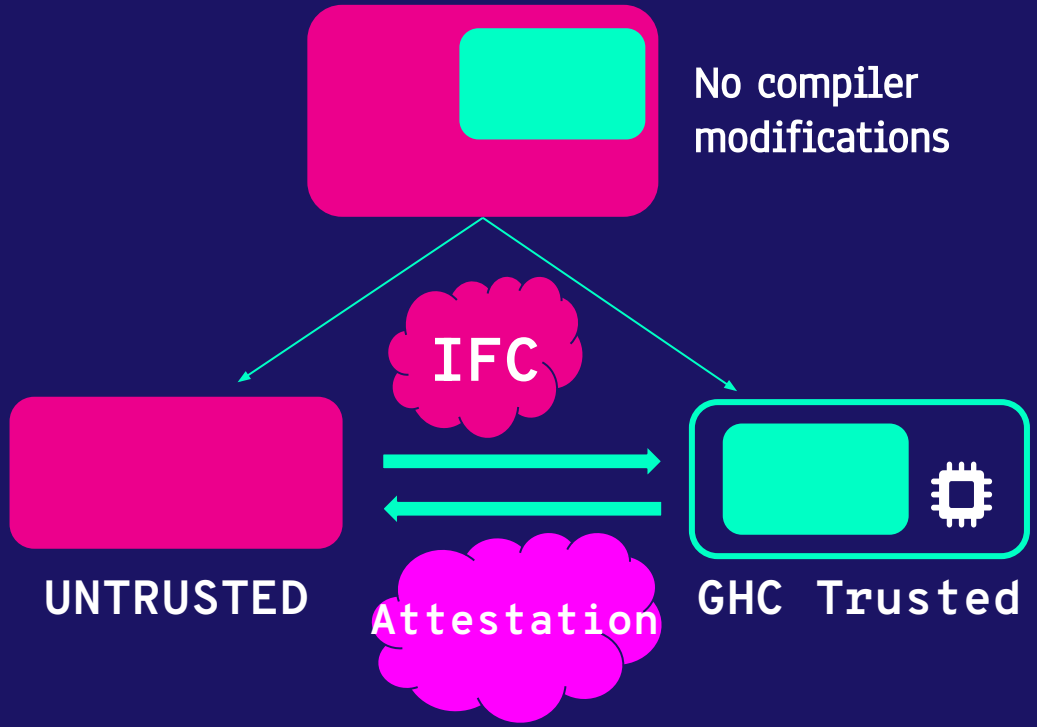


INFORMATION FLOW CONTROL?



HasTEE+

HasTEE+


No compiler
modifications
IFC
UNTRUSTED
Attestation
GHC Trusted

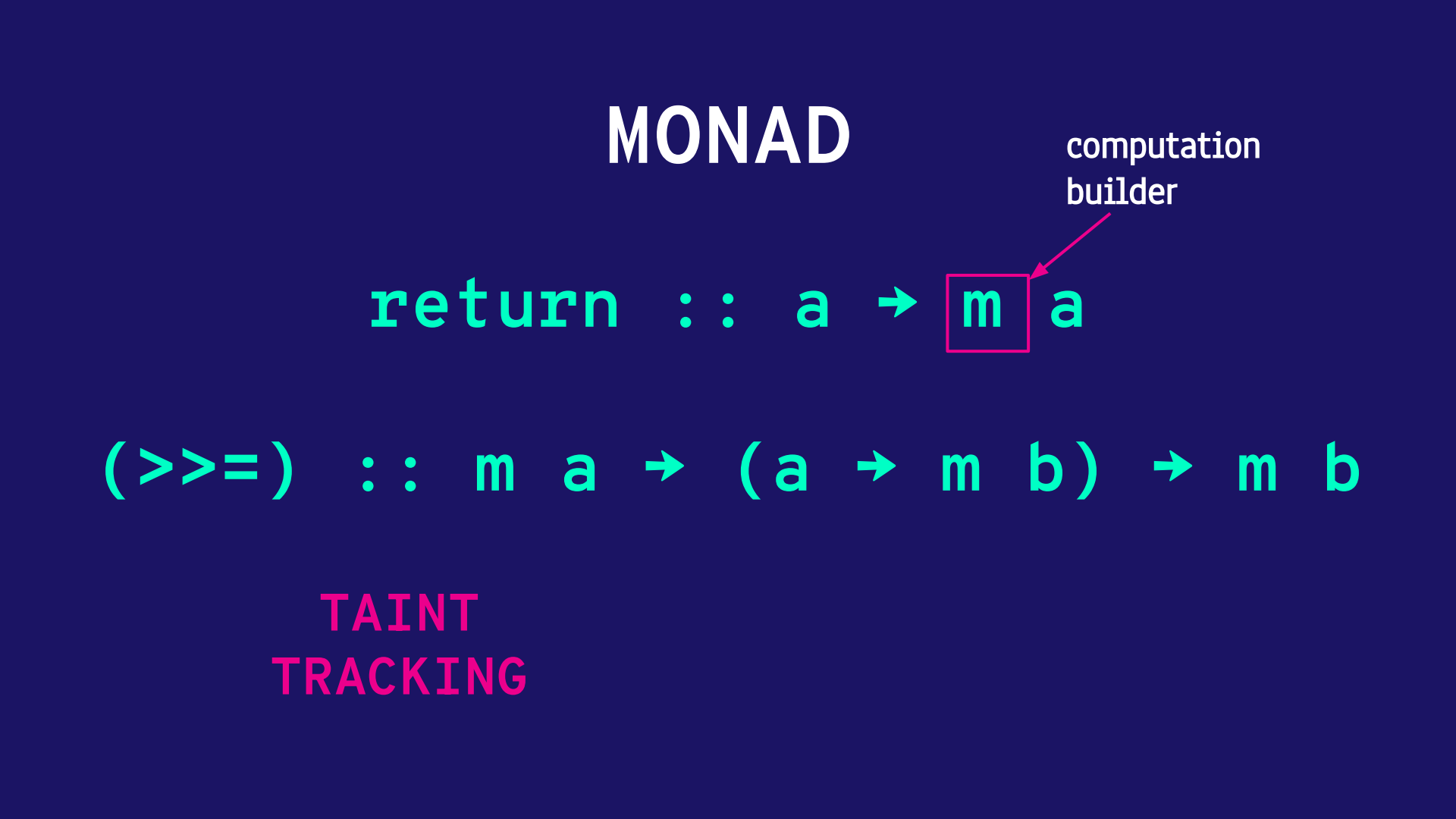
MONAD

```
return :: a → m a

(>>=) :: m a → (a → m b) → m b
```

MONAD


```
(>>=) :: m a → (a → m b) → m b
```

MONAD



TAINT
TRACKING

MONAD

```
return :: a → m a
```

```
(>>=) :: m a → (a → m b) → m b
```

TAINT TRACKING

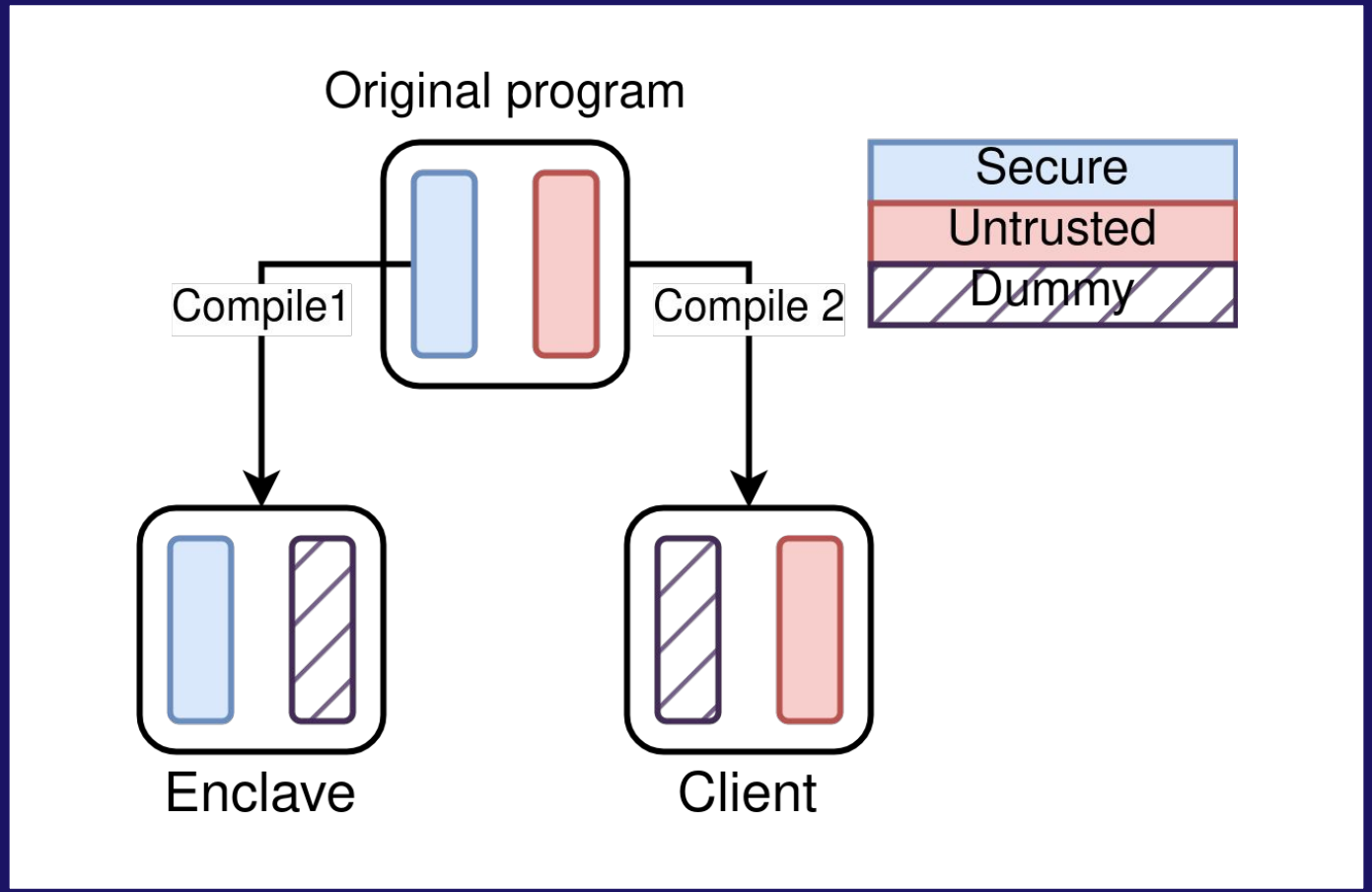
ALTERNATE SEMANTICS

Illustration : Password Checker

```
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

main = runApp passwordChecker
```

```
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConst "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
    liftIO $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

main = runApp passwordChecker
```

COMPILED TWICE


Original program
Secure
Untrusted
Dummy
Compile1
Compile 2
Enclave
Client

Compilation 1

```
-- Enclave
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  return DONE

-- wait for calls from Client
main = runApp passwordChecker
```

Compilation 2

```
-- Client
pwdChkr = -- gets optimised away

passwordChecker :: App Done
passwordChecker = do
  passwd <- return Dummy
  efunc <- inEnclave $ -- ignores pwdChkr body
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

-- drives the application
main = runApp passwordChecker
```

Compilation 1

```
-- Enclave
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  return DONE

-- wait for calls from Client
main = runApp passwordChecker
```

Compilation 2

```
-- Client
pwdChkr = -- gets optimised away

passwordChecker :: App Done
passwordChecker = do
  passwd <- return Dummy
  efunc <- inEnclave $ -- ignores pwdChkr body
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

-- drives the application
main = runApp passwordChecker
```

Compilation 1

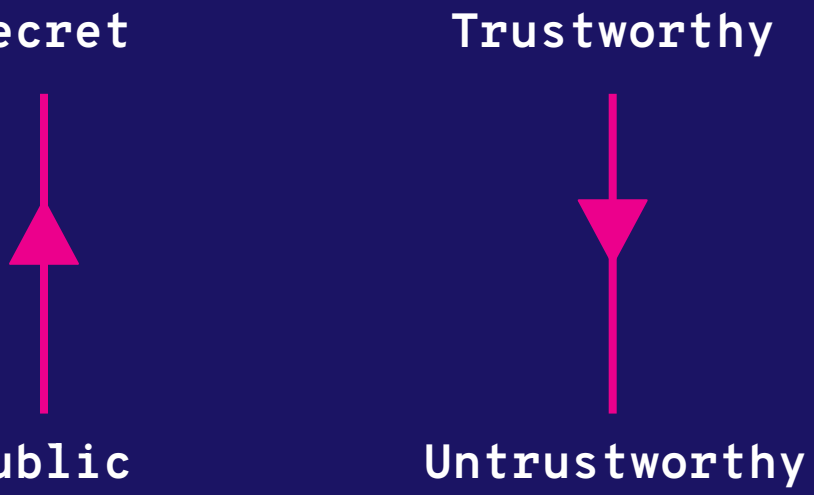
```
-- Enclave
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  return DONE

-- wait for calls from Client
main = runApp passwordChecker
```

GHC Trusted

INTEL SGX

Information Flow

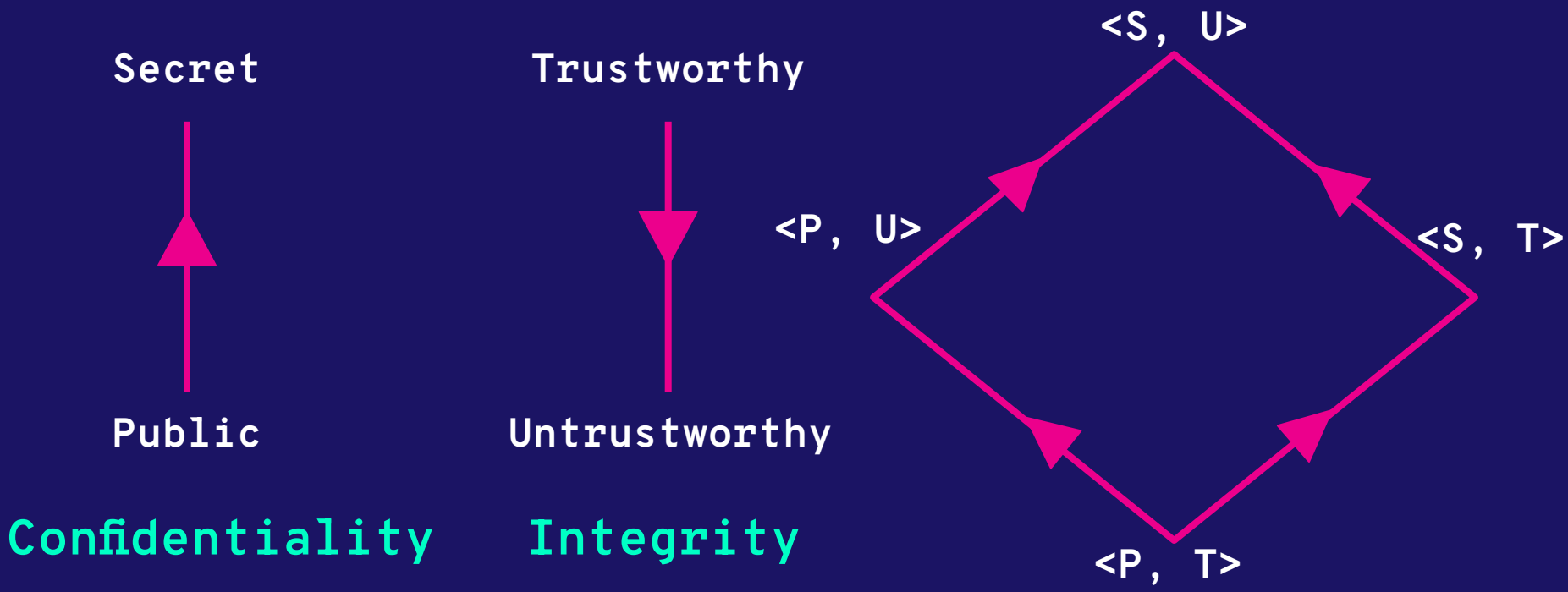
Information Flow Control



Confidentiality

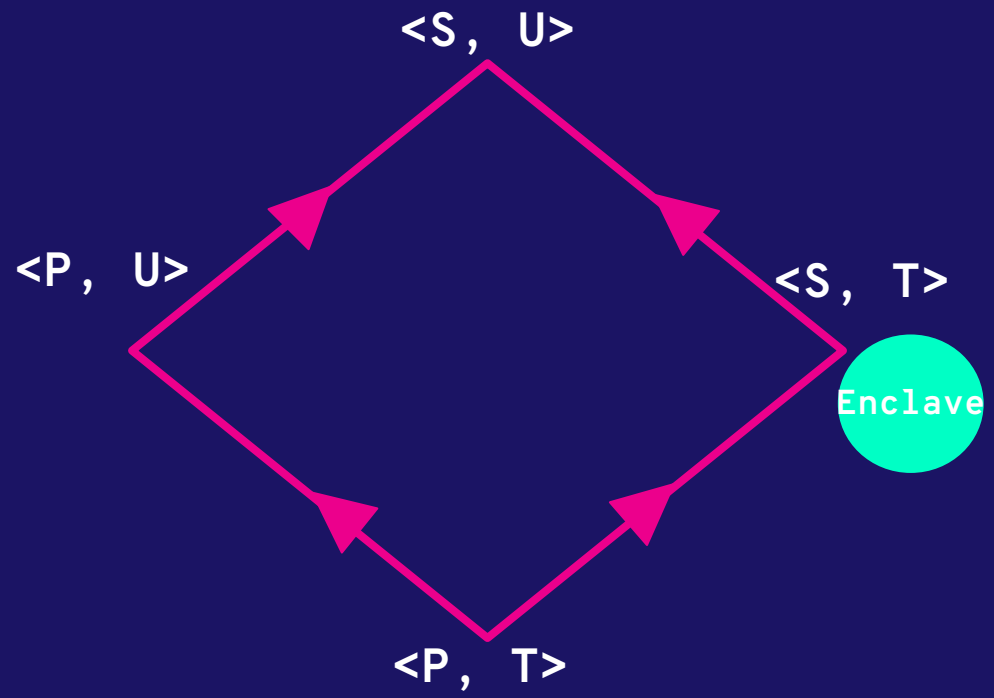
Integrity

Information Flow Control

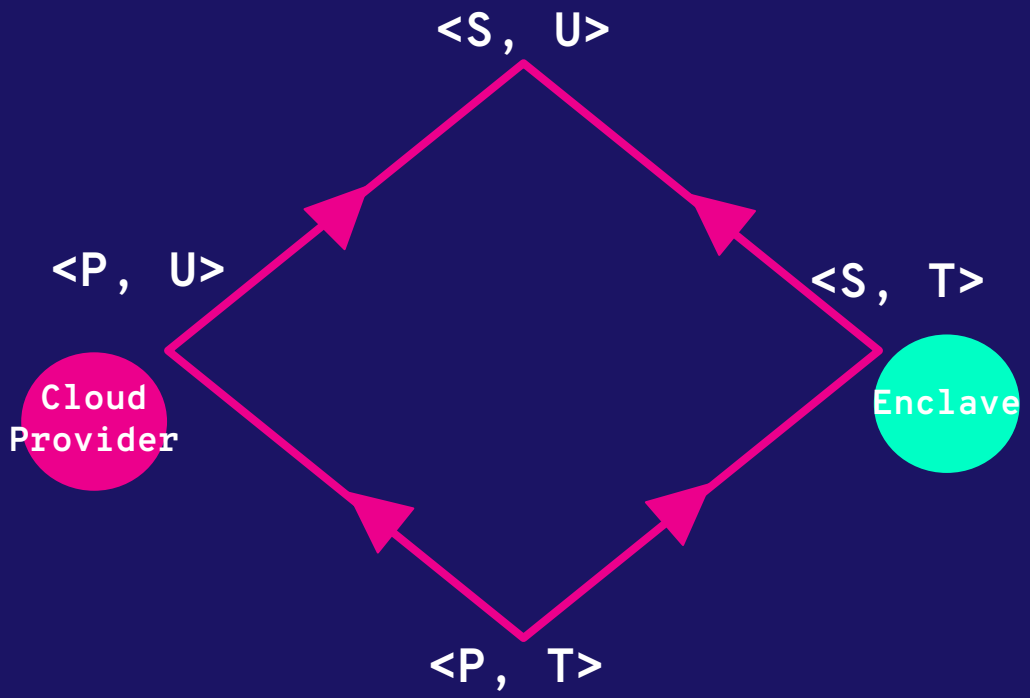


Denning, Dorothy E. "A lattice model of secure information flow." *Communications of the ACM* 19.5 (1976): 236-243.
Biba, K.J. Integrity considerations for secure computer systems. Technical Report. April 1977.

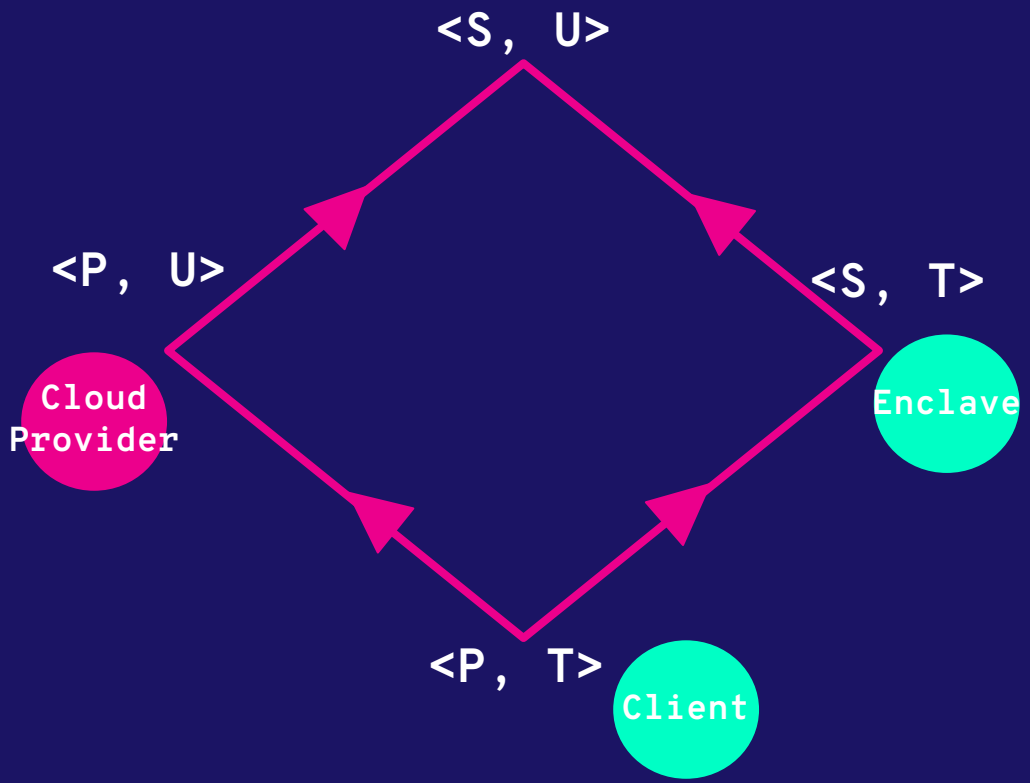
Information Flow Control



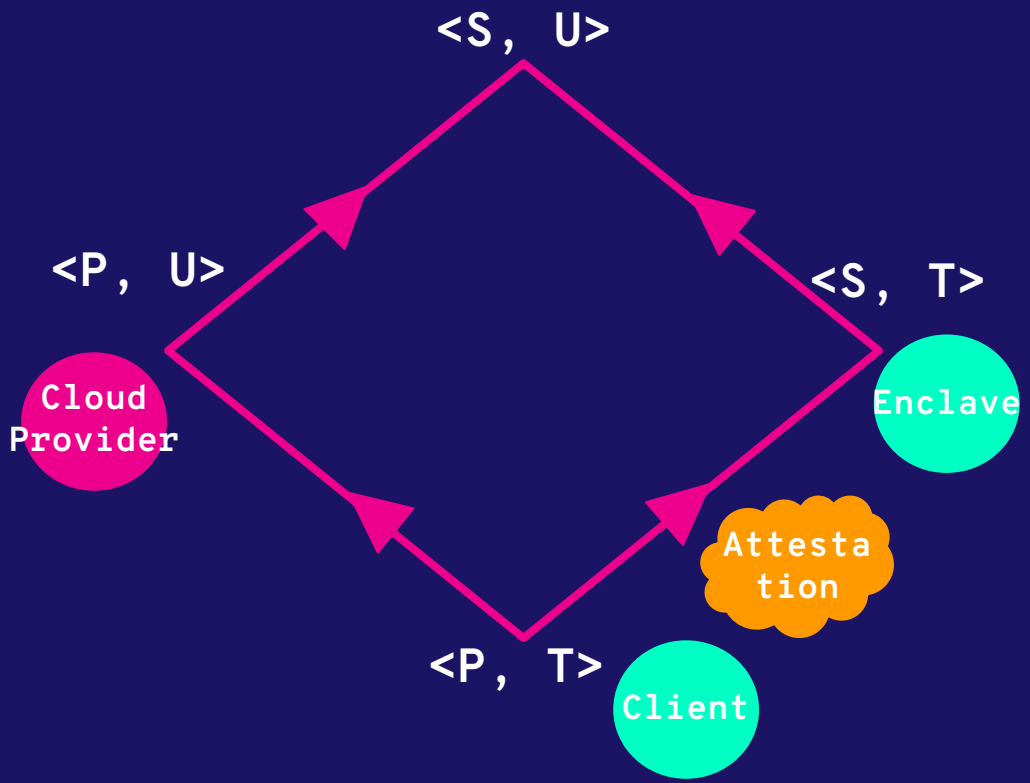
Information Flow Control



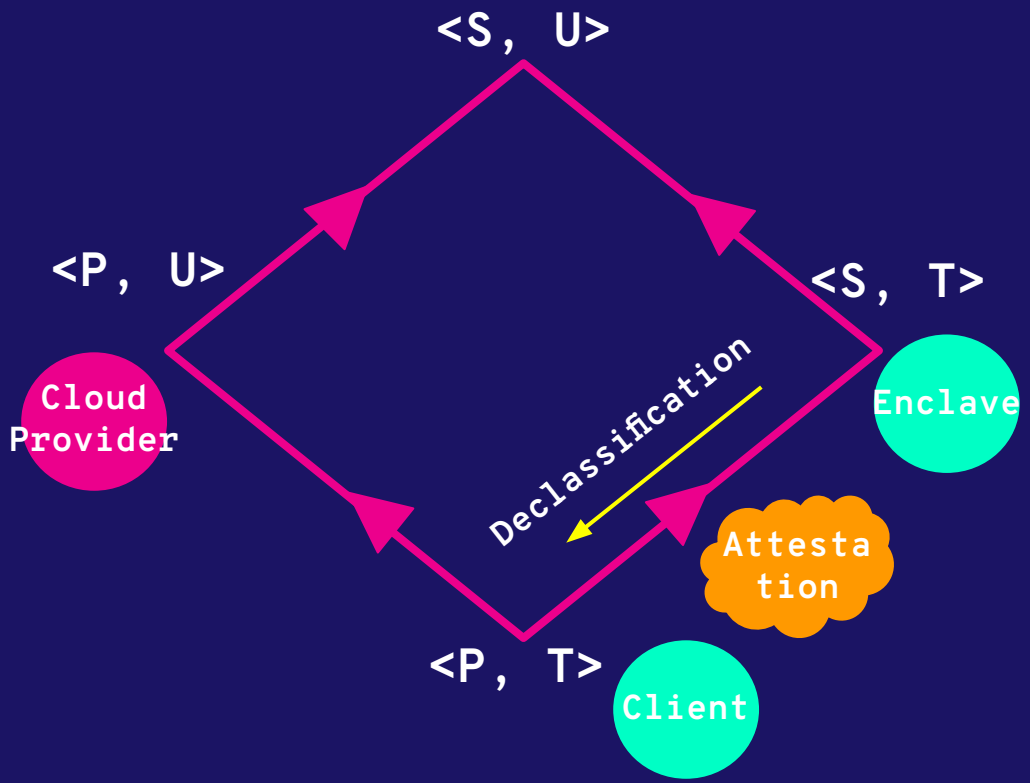
Information Flow Control



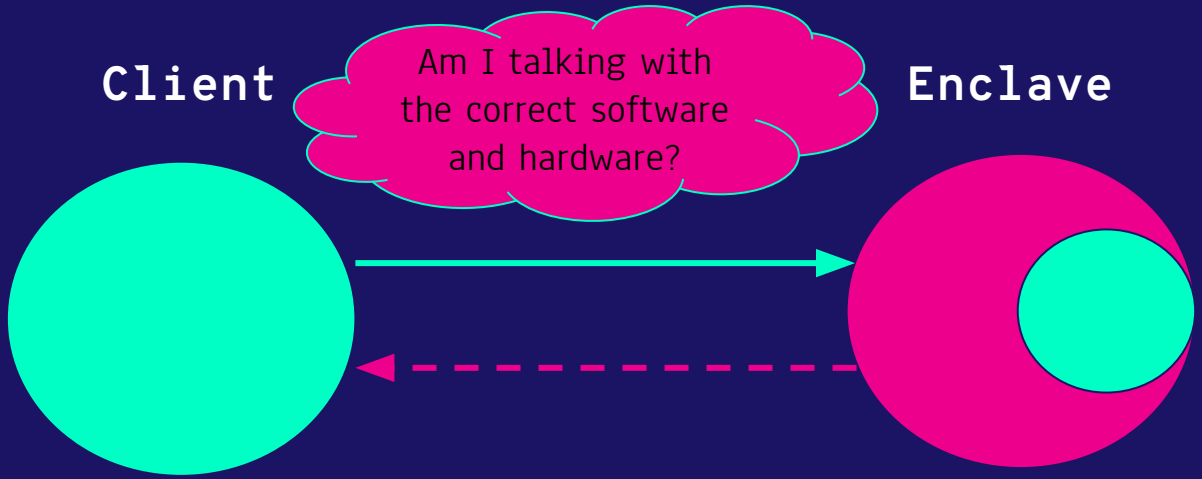
Information Flow Control



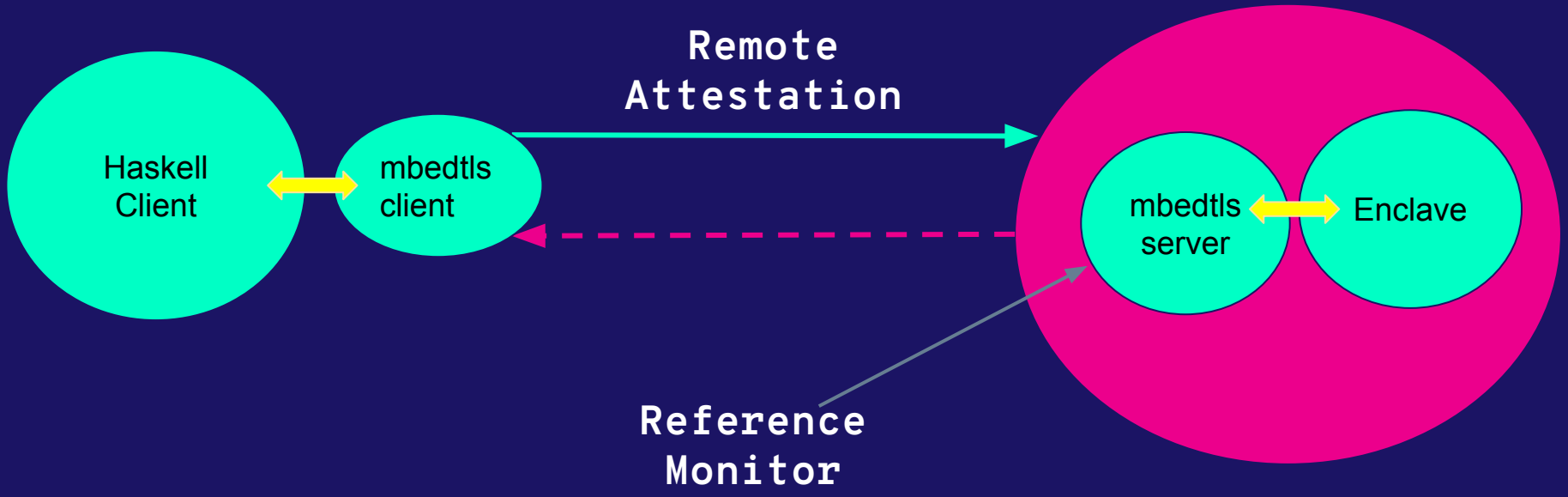
Information Flow Control



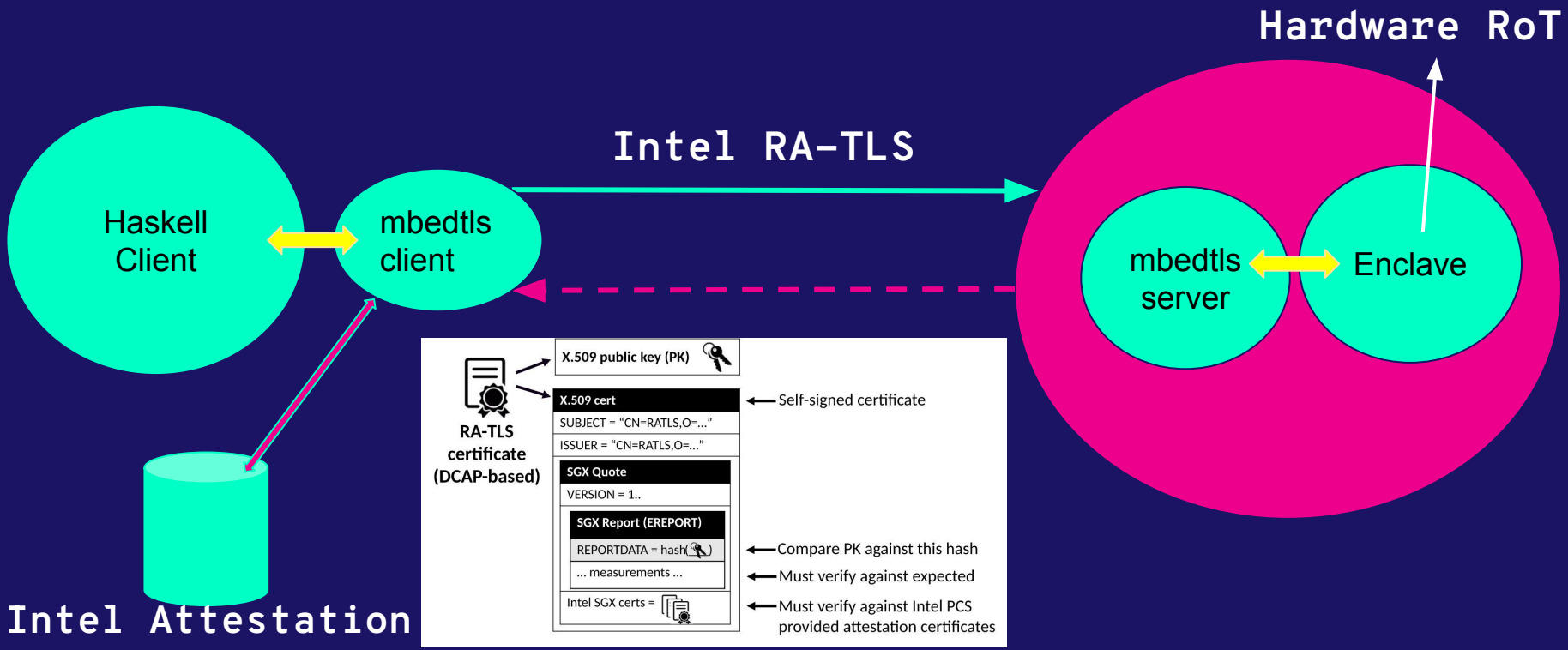
Attestation



Attestation

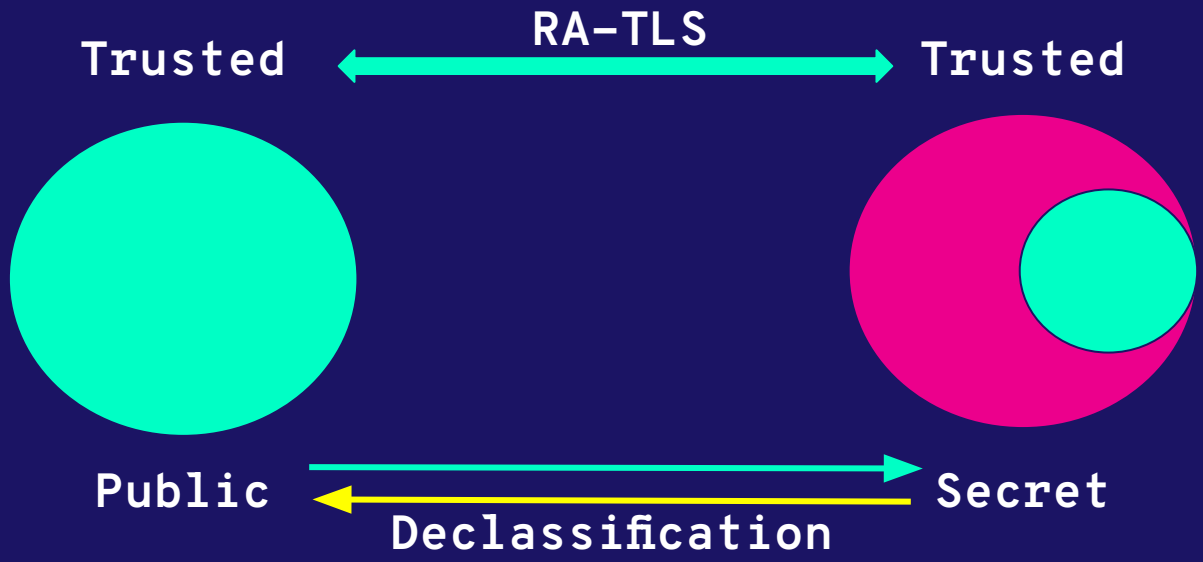


Attestation



Attestation

Information Flow



Information Flow Control

```
class Label l where
  lub :: l → l → l
  glb :: l → l → l
  canFlowTo :: l → l → Bool

Enclave l p a -- parameterised on Label l and
              -- privilege p
Labeled l a   -- parameterised on Label l
```

Stefan, D., Russo, A., Mitchell, J.C., Mazières, D.: Flexible Dynamic Information Flow Control. In *ACM Haskell Symposium 2011*

Information Flow Control

```
label   :: Label l ⇒ l → a → Enclave l p (Labeled l a)
unlabel :: Label l ⇒ Labeled l a → Enclave l p a
taint   :: Label l ⇒ l → Enclave l p ()


labelP   :: PrivDesc l p ⇒ Priv p → l → a → Enclave l p (Labeled l a)
unlabelP :: PrivDesc l p ⇒ Priv p → Labeled l a → Enclave l p a
taintP   :: PrivDesc l p ⇒ Priv p → l → Enclave l p ()
```

Information Flow Control

```
label   :: Label l ⇒ l → a → Enclave l p (Labeled l a)
unlabel :: Label l ⇒ Labeled l a → Enclave l p a
taint   :: Label l ⇒ l → Enclave l p ()


labelP   :: PrivDesc l p ⇒ Priv p → l → a → Enclave l p (Labeled l a)
unlabelP :: PrivDesc l p ⇒ Priv p → Labeled l a → Enclave l p a
taintP   :: PrivDesc l p ⇒ Priv p → l → Enclave l p ()
```

Information Flow Control

```
label    :: Label l ⇒ l → a → Enclave l p (Labeled l a)
unlabel  :: Label l ⇒ Labeled l a → Enclave l p a
taint    :: Label l ⇒ l → Enclave l p ()


labelP   :: PrivDesc l p ⇒ Priv p → l → a → Enclave l p (Labeled l a)
unlabelP :: PrivDesc l p ⇒ Priv p → Labeled l a → Enclave l p a
taintP   :: PrivDesc l p ⇒ Priv p → l → Enclave l p ()
```

Privilege/Capability
An unforgeable token of authority

Information Flow Control

```
label   :: Label l ⇒ l → a → Enclave l p (Labeled l a)
unlabel :: Label l ⇒ Labeled l a → Enclave l p a
taint   :: Label l ⇒ l → Enclave l p ()


labelP   :: PrivDesc l p ⇒ Priv p → l → a → Enclave l p (Labeled l a)
unlabelP :: PrivDesc l p ⇒ Priv p → Labeled l a → Enclave l p a
taintP   :: PrivDesc l p ⇒ Priv p → l → Enclave l p ()
```

Declassification with Privileges

Can $<C_1, I_1>$ flow to $<C_2, I_2>$ in the presence of privilege P?

$$\frac{P \wedge C_2 \Longrightarrow C_1 \qquad P \wedge I_1 \Longrightarrow I_2}{< C_1, I_1 > \sqsubseteq_P < C_2, I_2 >}$$

Stefan, D., Russo, A., Mazières, D., & Mitchell, J. C. (2012). Disjunction Category Labels. In *Information Security Technology for Applications: 16th Nordic Conference on Secure IT Systems, NordSec 2011*

Declassification with Privileges

Can <C_1, I_1> flow to <C_2, I_2> in the presence of privilege P?

$$\frac{P \wedge C_2 \Longrightarrow C_1 \qquad P \wedge I_1 \Longrightarrow I_2}{< C_1, I_1 > \sqsubseteq_P < C_2, I_2 >}$$

A CNF represents a conjunction of *Principals*
Eg: (Alice /\ Bob)

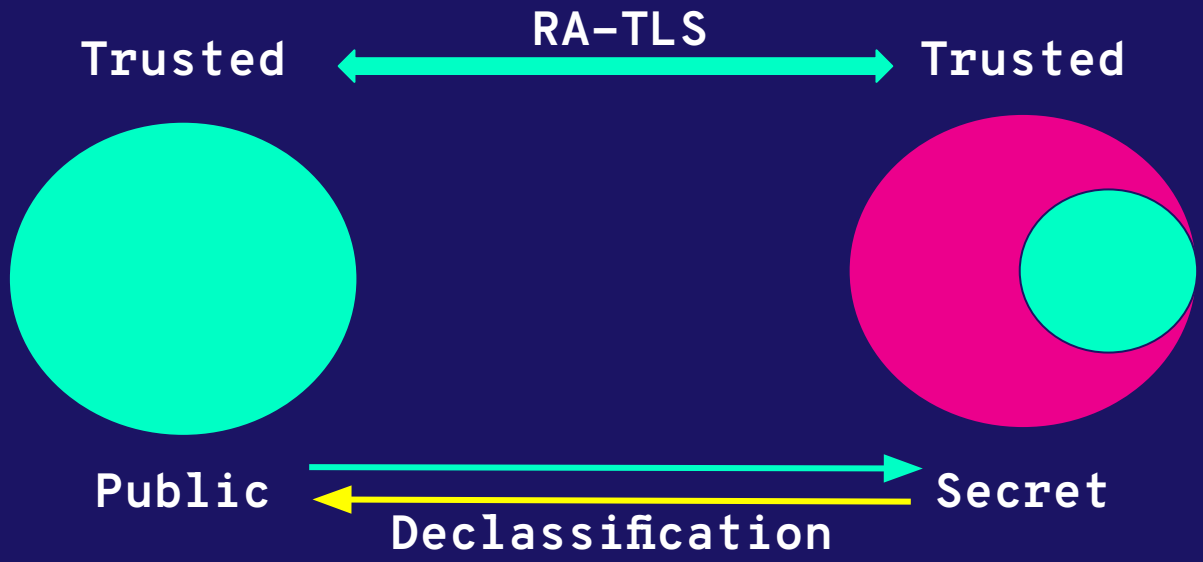
Declassification with Privileges

Can <C_1, I_1> flow to <C_2, I_2> in the presence of privilege P?

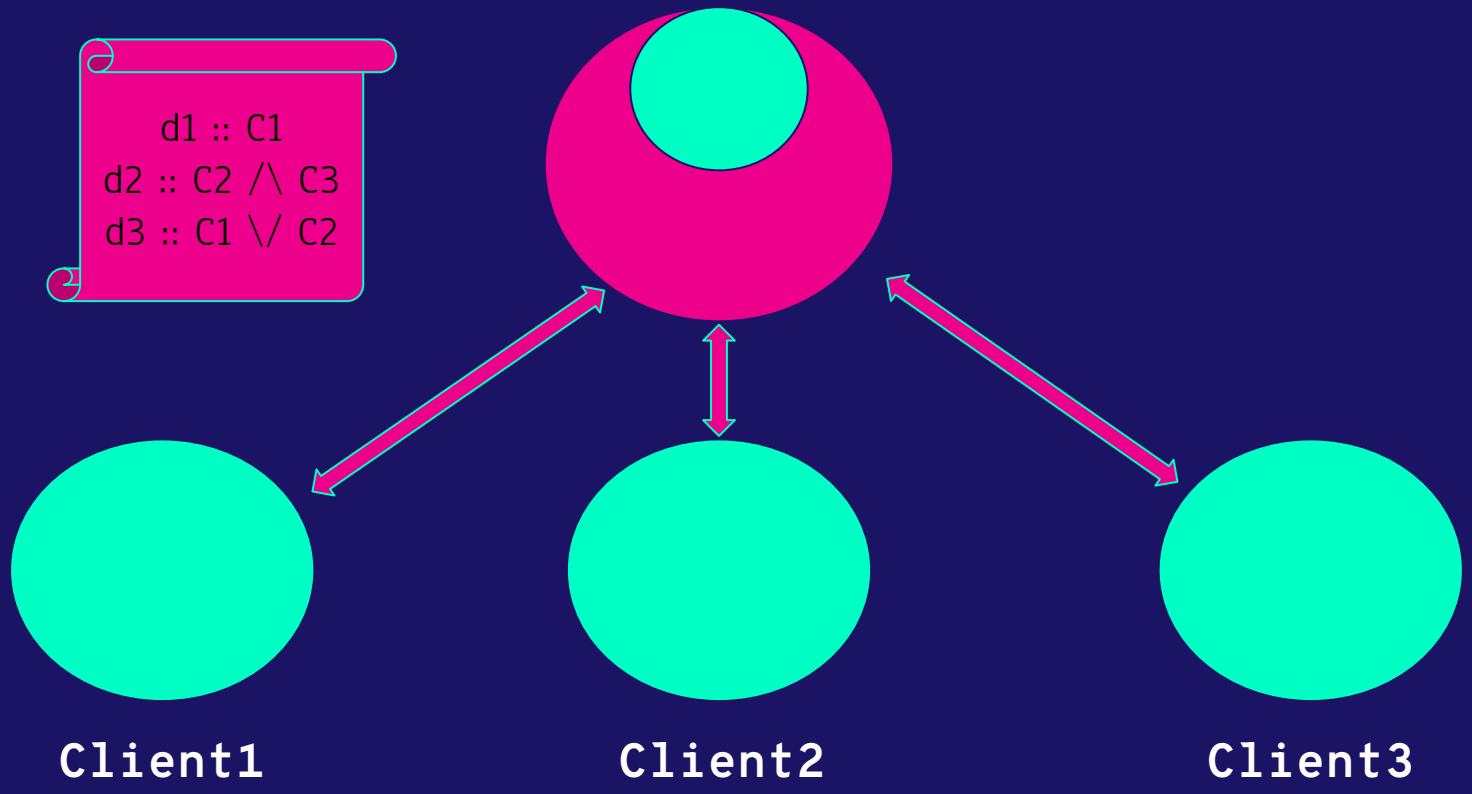
$$\frac{P \wedge C_2 \Longrightarrow C_1 \qquad P \wedge I_1 \Longrightarrow I_2}{< C_1, I_1 > \sqsubseteq_P < C_2, I_2 >}$$

Raises the confidentiality level of C_2 → *Declassification*

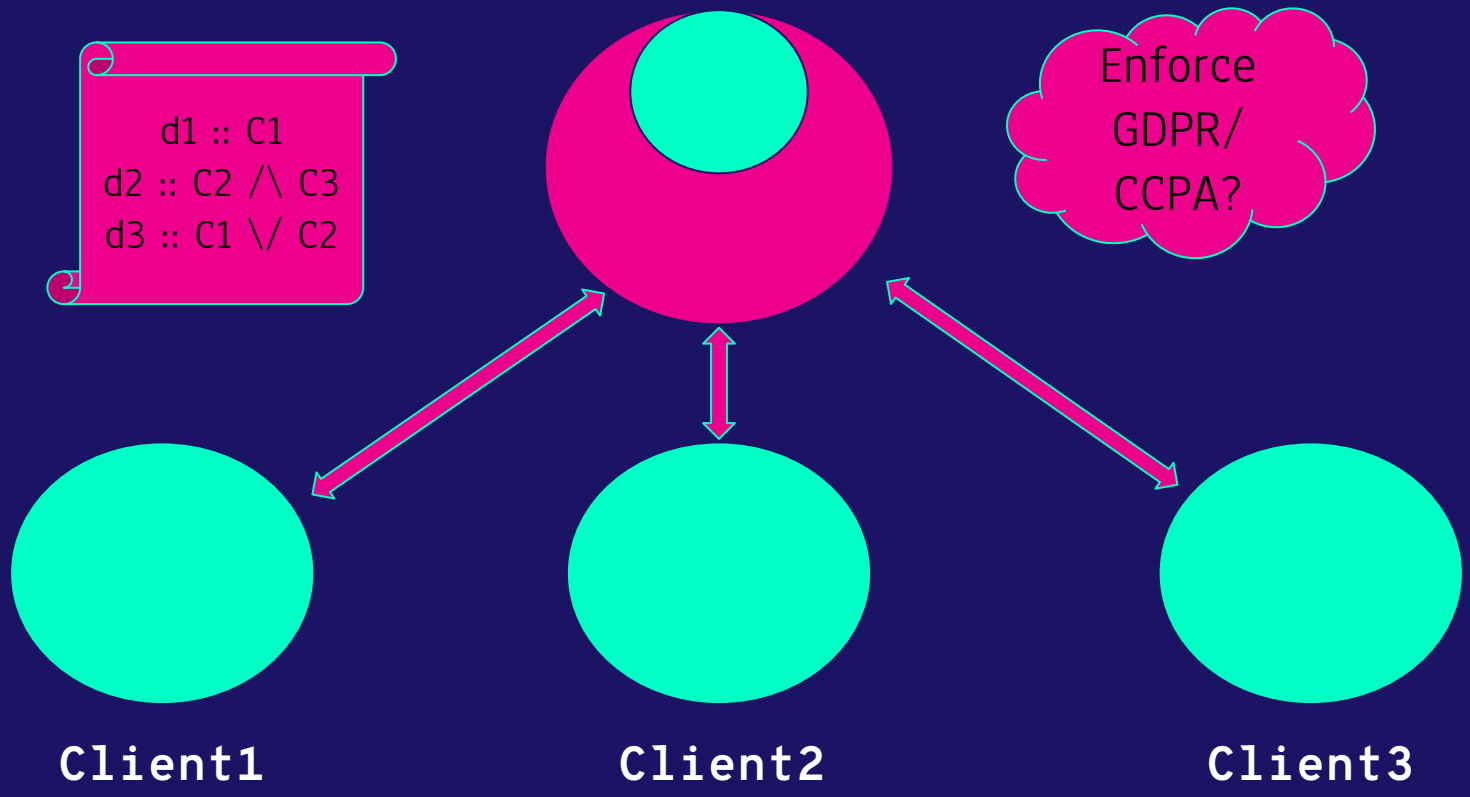
Information Flow



Data Clean Room

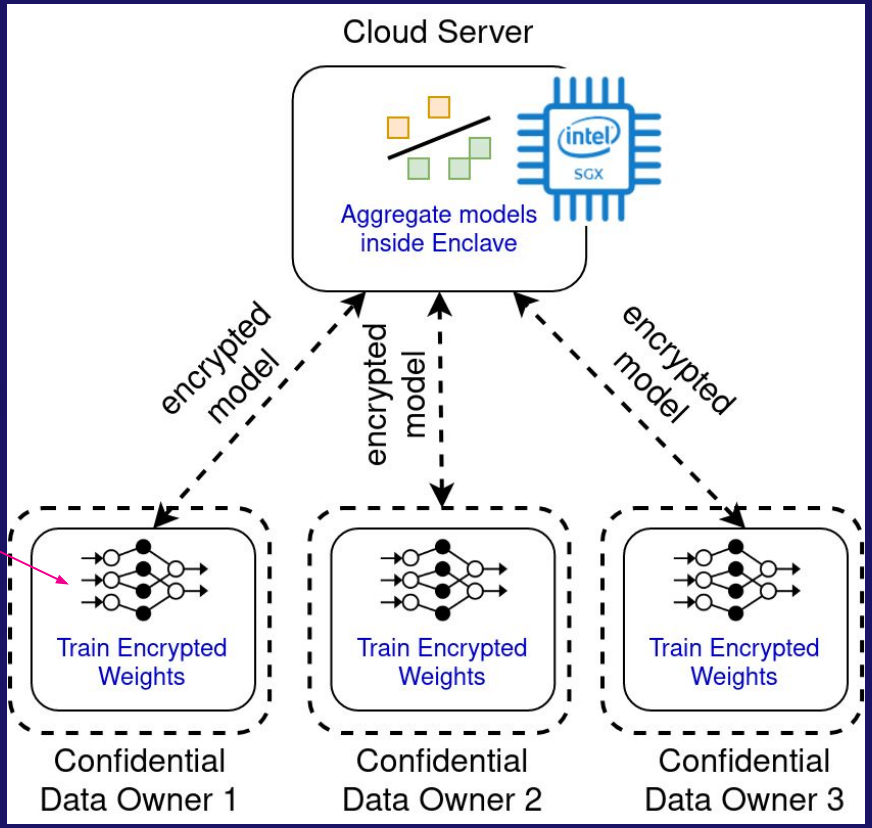


Data Clean Room



Zero Trust Federated Learning

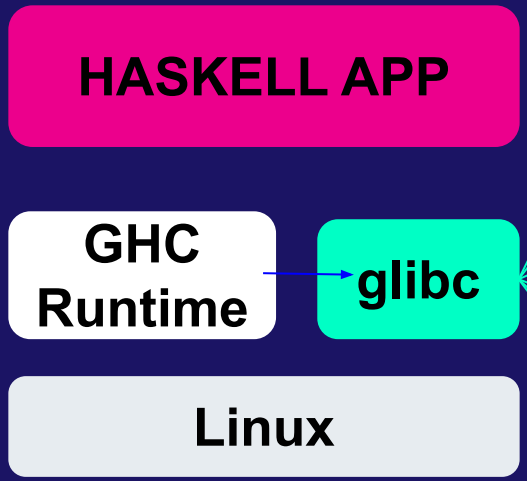
Uses homomorphic encryption for training



Applications

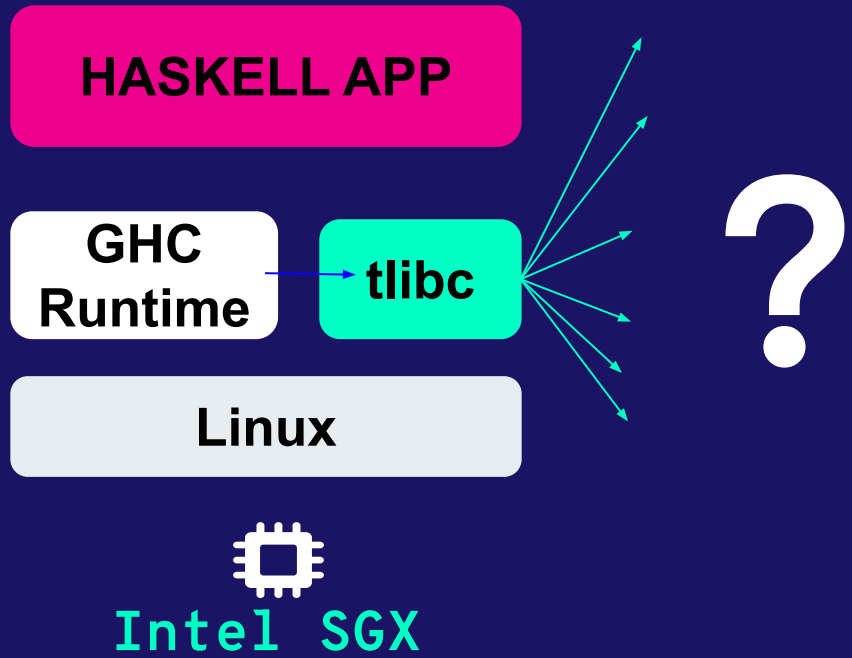
- Privacy-preserving Federated Learning
- Encrypted Password Wallet
- Data Clean Room with Differential Privacy

IMPLEMENTATION

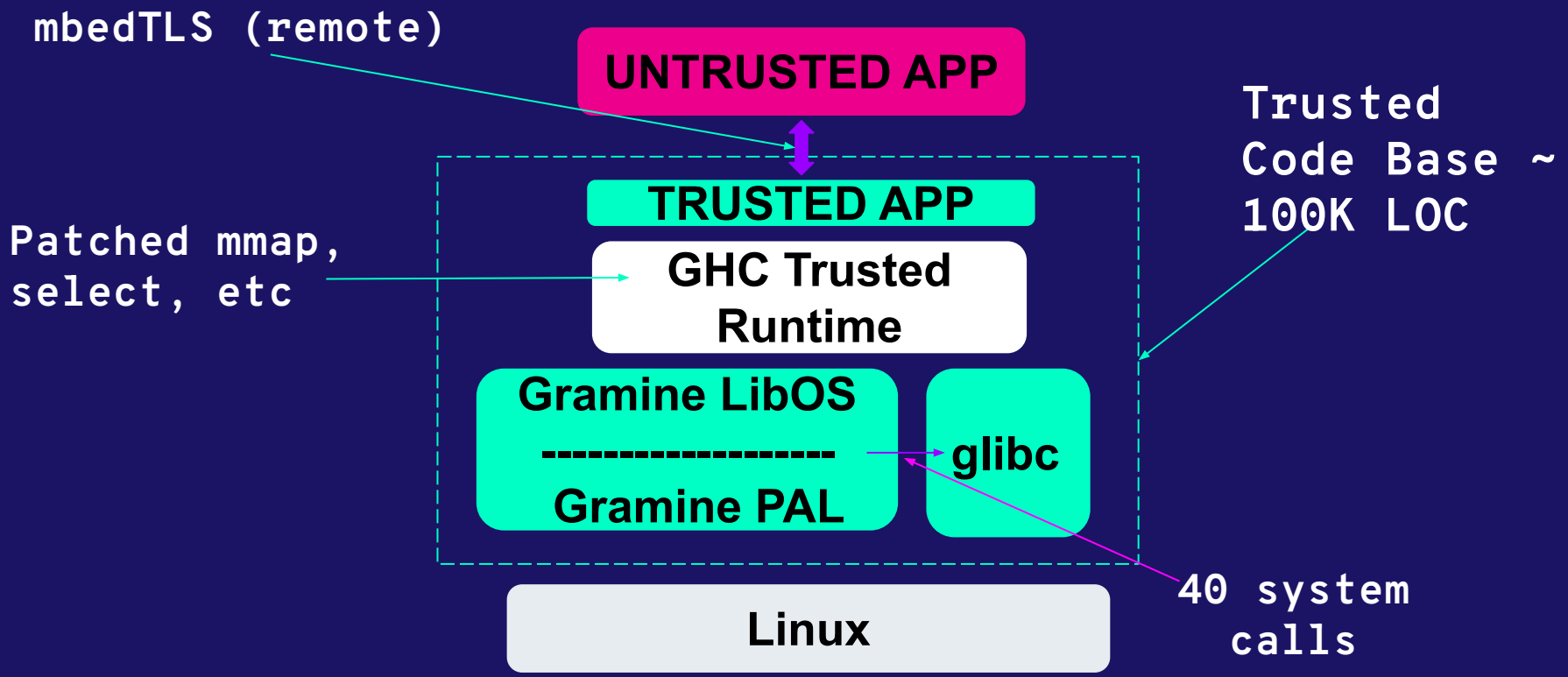


- mmap
- madvise
- getrusage
- poll
- select
- clock apis
- pthread apis
- libm apis
- …

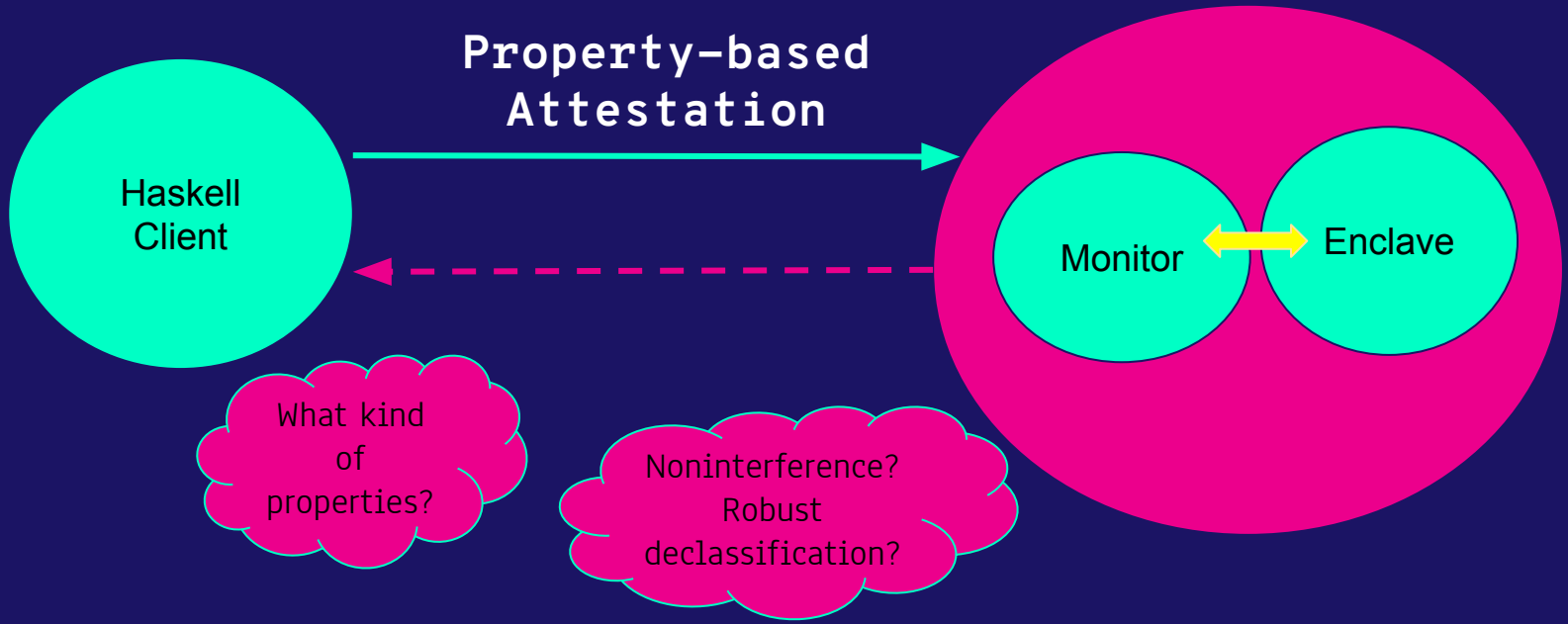
IMPLEMENTATION



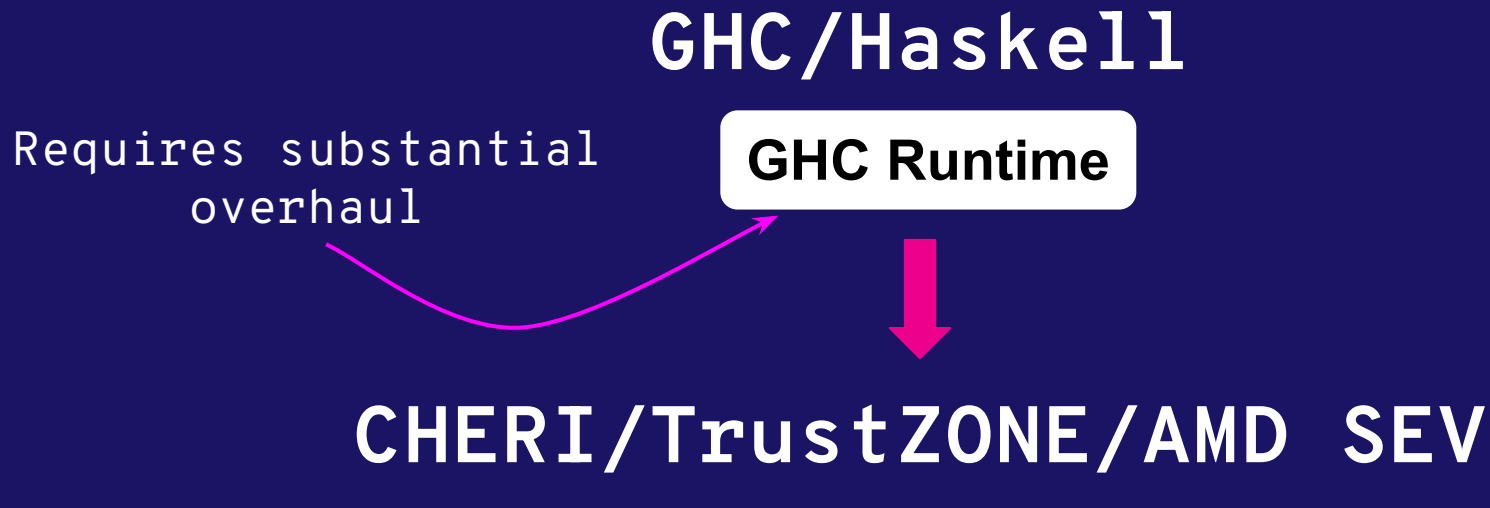
IMPLEMENTATION



FUTURE WORK


Property-based
Attestation
Haskell
Client
Monitor
Enclave
What kind
of
properties?
Noninterference?
Robust
declassification?

FUTURE WORK


GHC/Haskell
Requires substantial overhaul
GHC Runtime
CHERI/TrustZONE/AMD SEV

THANKS !

https://github.com/Abhiroop/HasTEE
https://abhiroop.github.io/publications/