ILLINOIS INSTITUTE OF TECHNOLOGY 1890

Carnegie Mellon University Computer Science Department

Carnegie Mellon University CyLab Security and Privacy Institute

Towards End-to-End Verified TEEs via Verified Interface Conformance and Certified Compilers

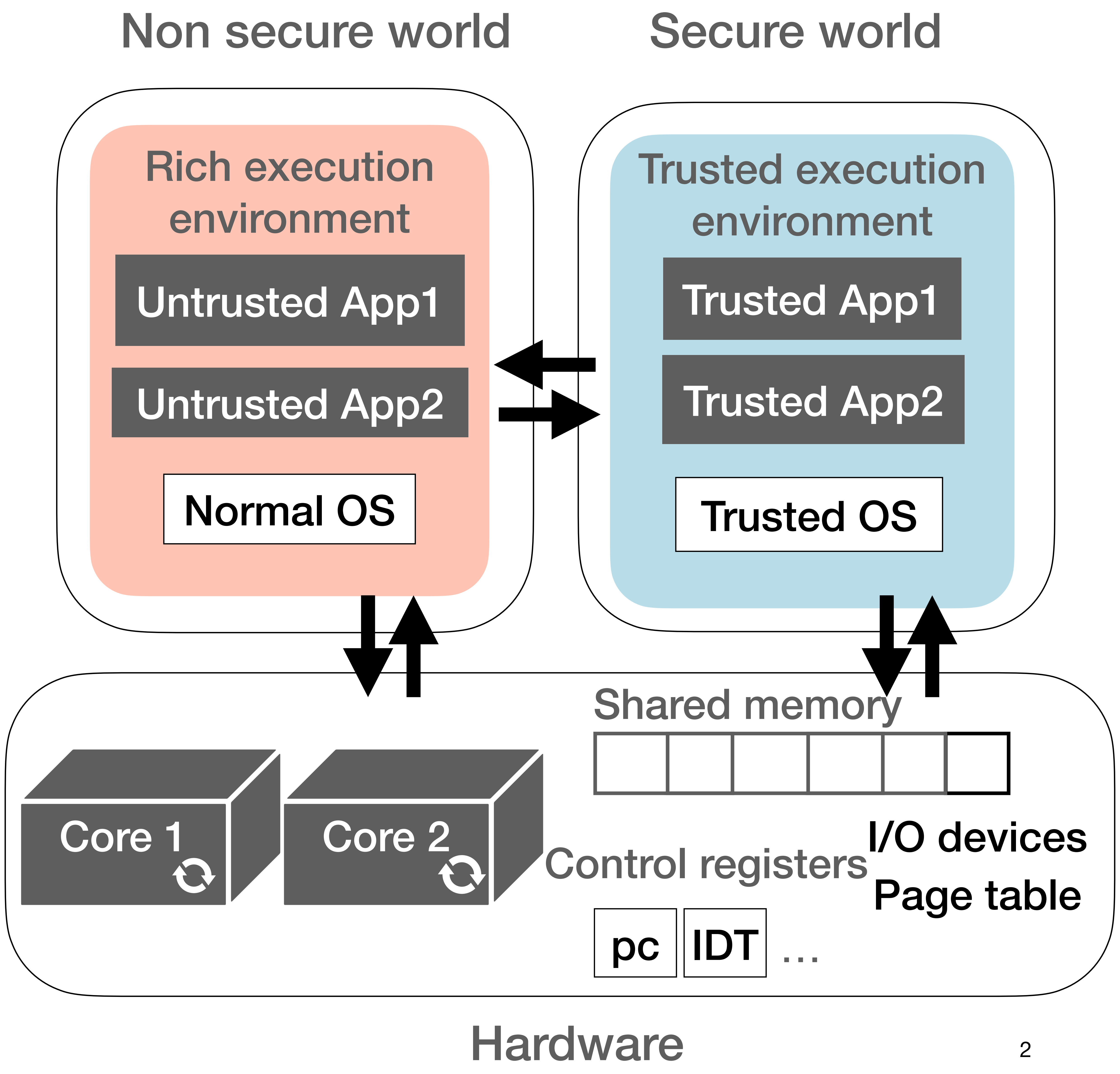
Farzaneh Derakhshan

(fderakhs@andrew.cmu.edu)

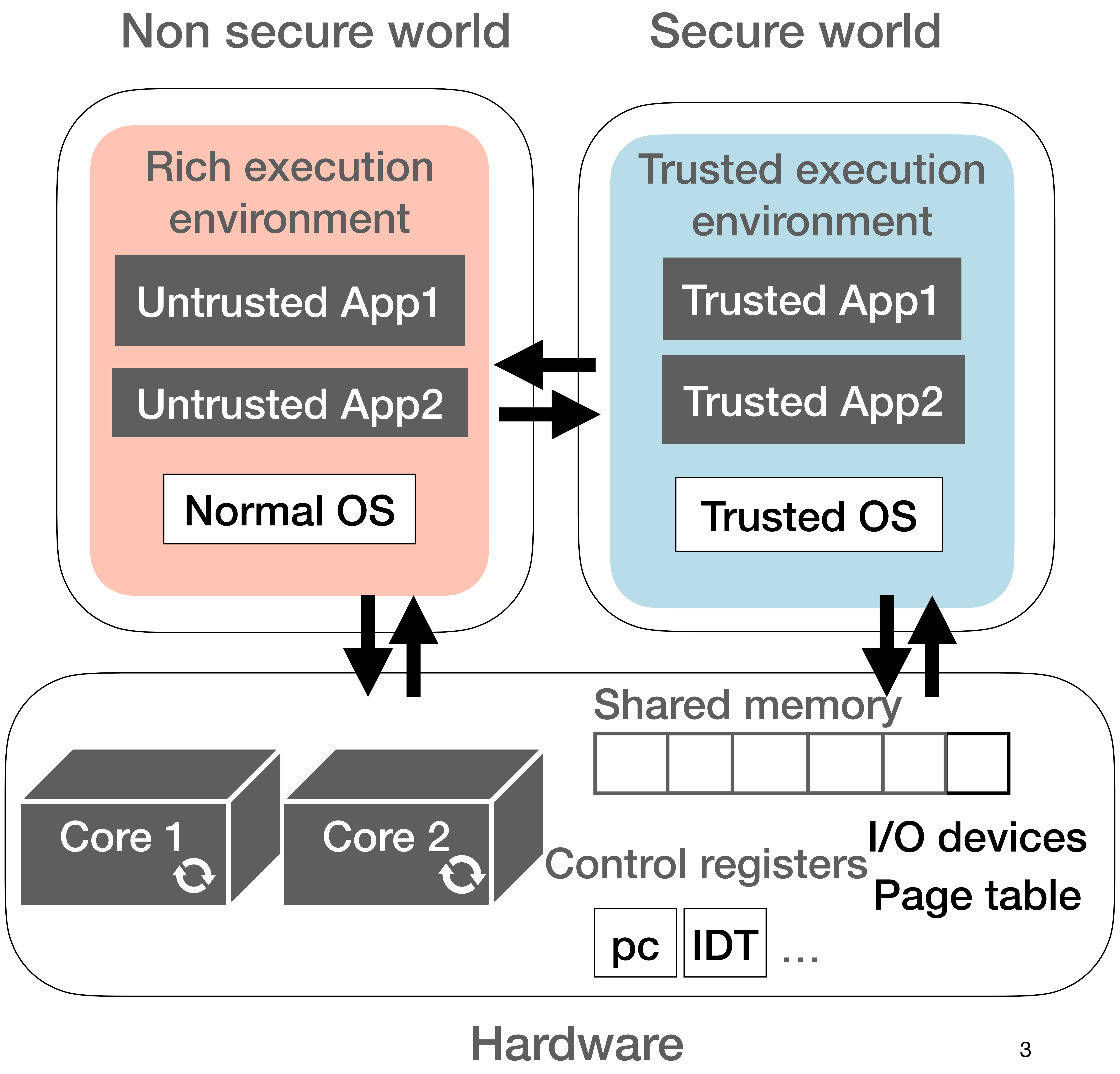
Joint work with Zichao Zhang, Amit Vasudevan, and Limin Jia

IEEE 36th Computer Security Foundations Symposium
CSF 2023, Jul 12

Trusted Execution Environments (TEE)



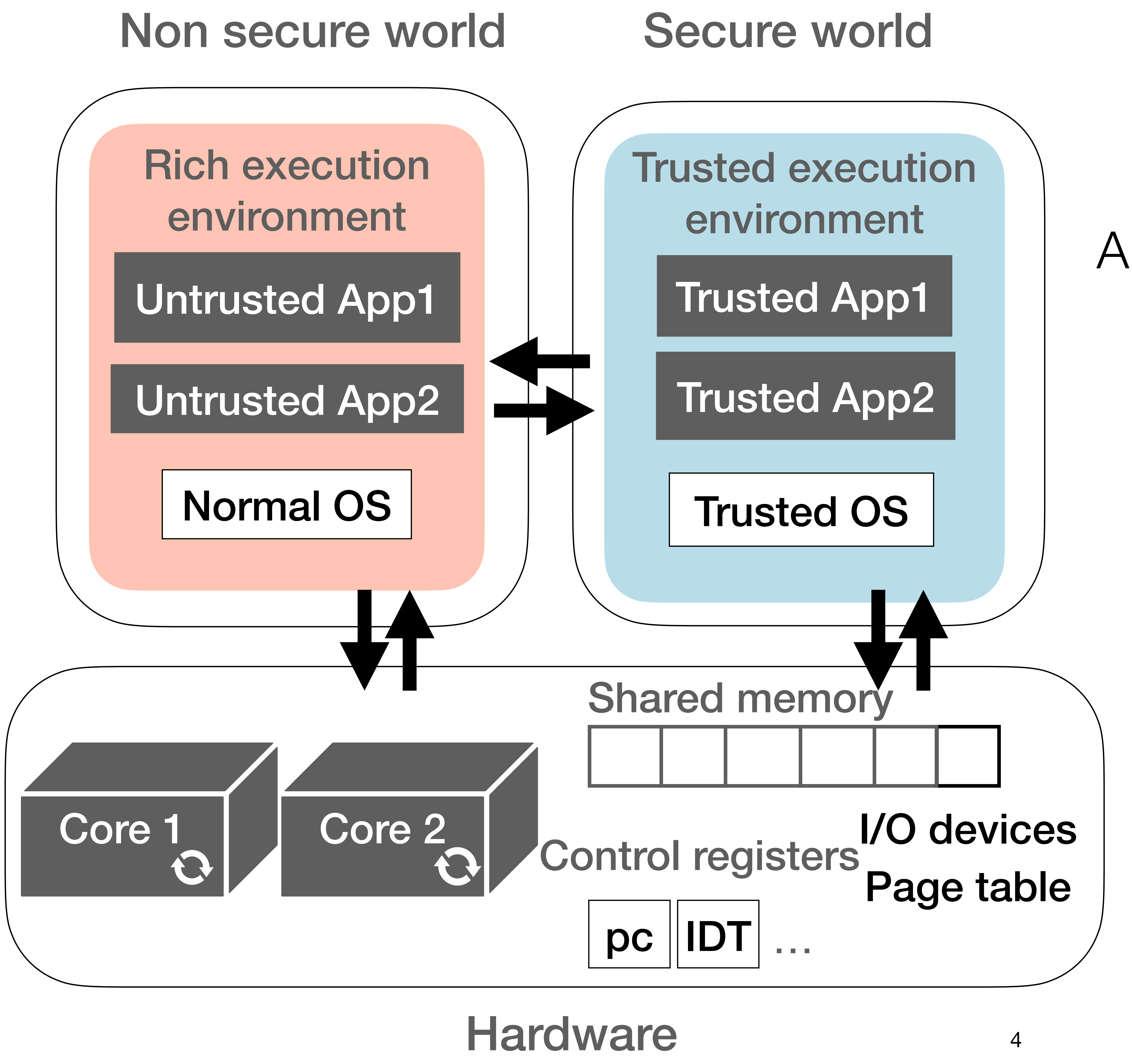
Trusted Execution Environments (TEE)



Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

Trusted Execution Environments (TEE)



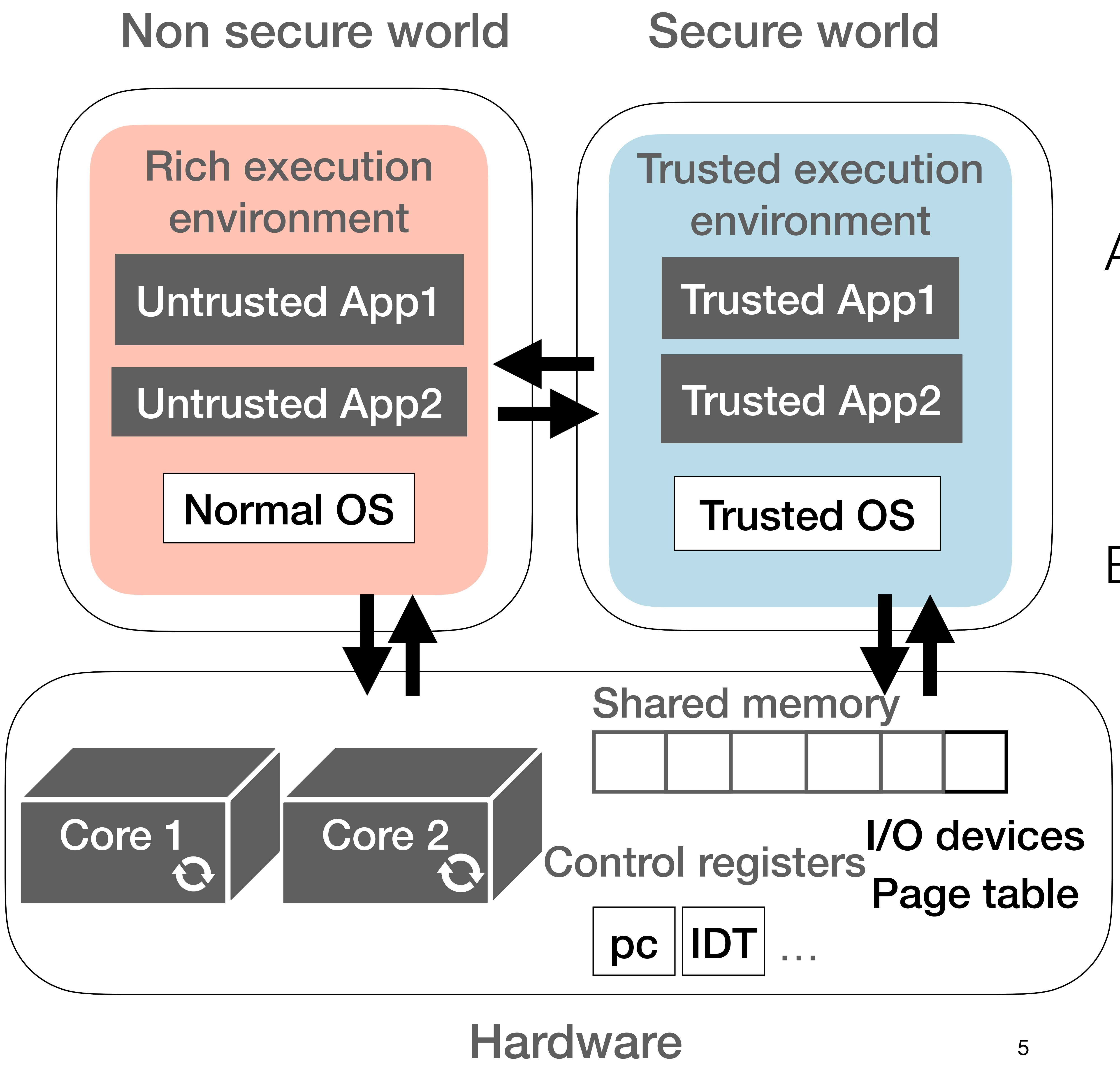
Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

Application

- Cloud computing
- Secure banking

Trusted Execution Environments (TEE)



Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

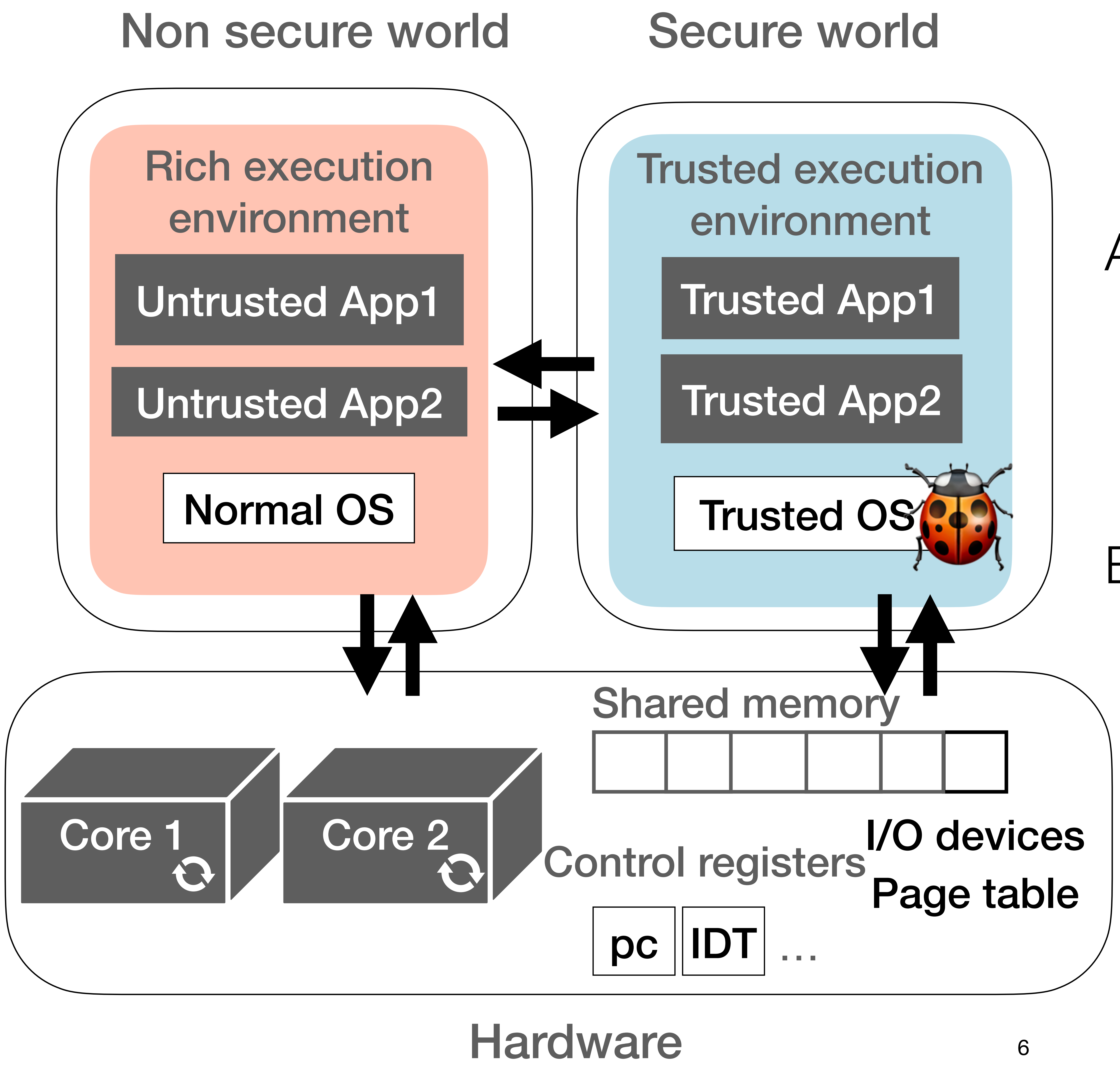
Application

- Cloud computing
- Secure banking

Example Trusted OS

- Hypervisors
- Trusty for Android
- OP-TEE for Arm

Trusted Execution Environments (TEE)



Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

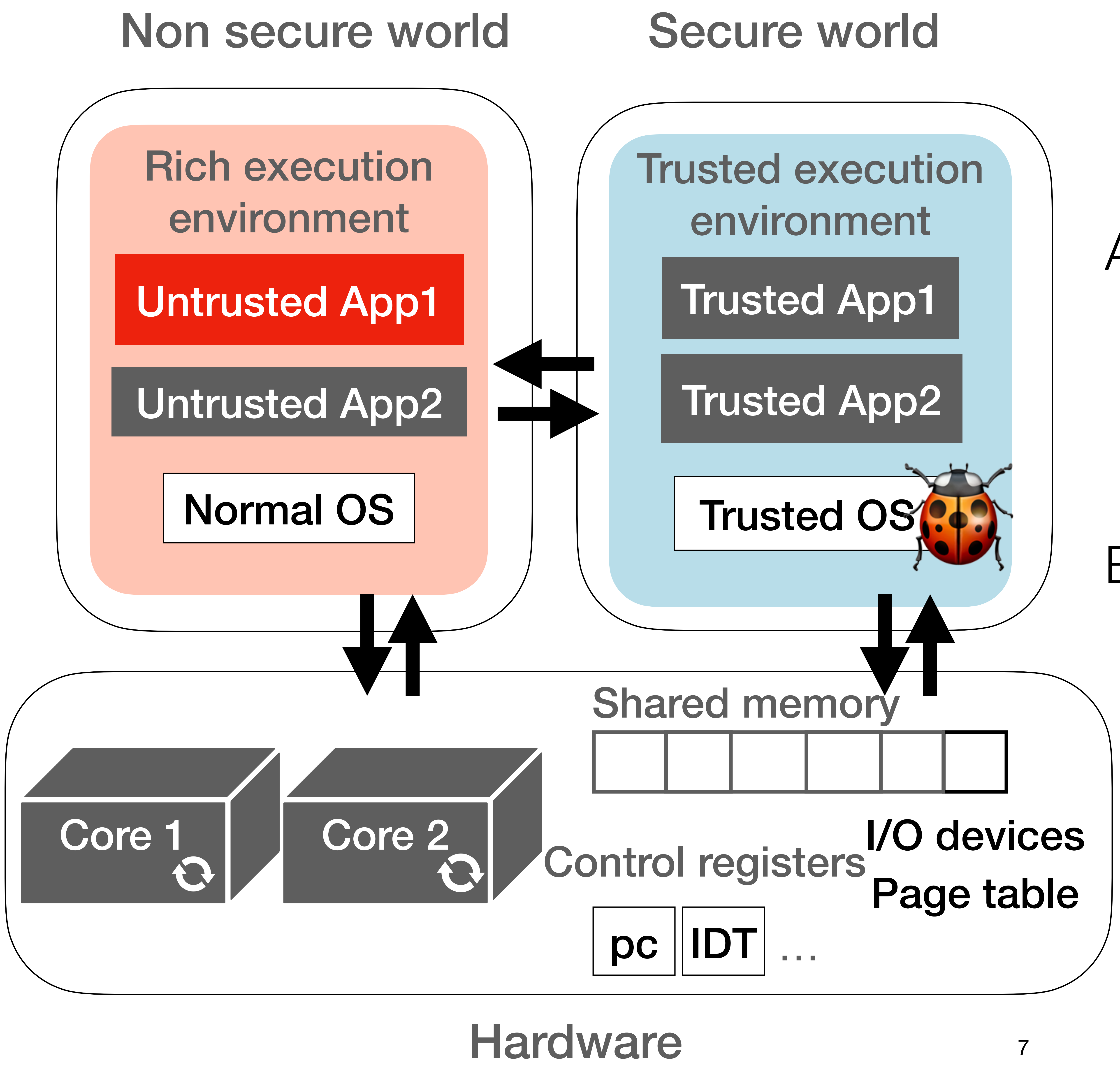
Application

- Cloud computing
- Secure banking

Example Trusted OS

- Hypervisors
- Trusty for Android
- OP-TEE for Arm

Trusted Execution Environments (TEE)



Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

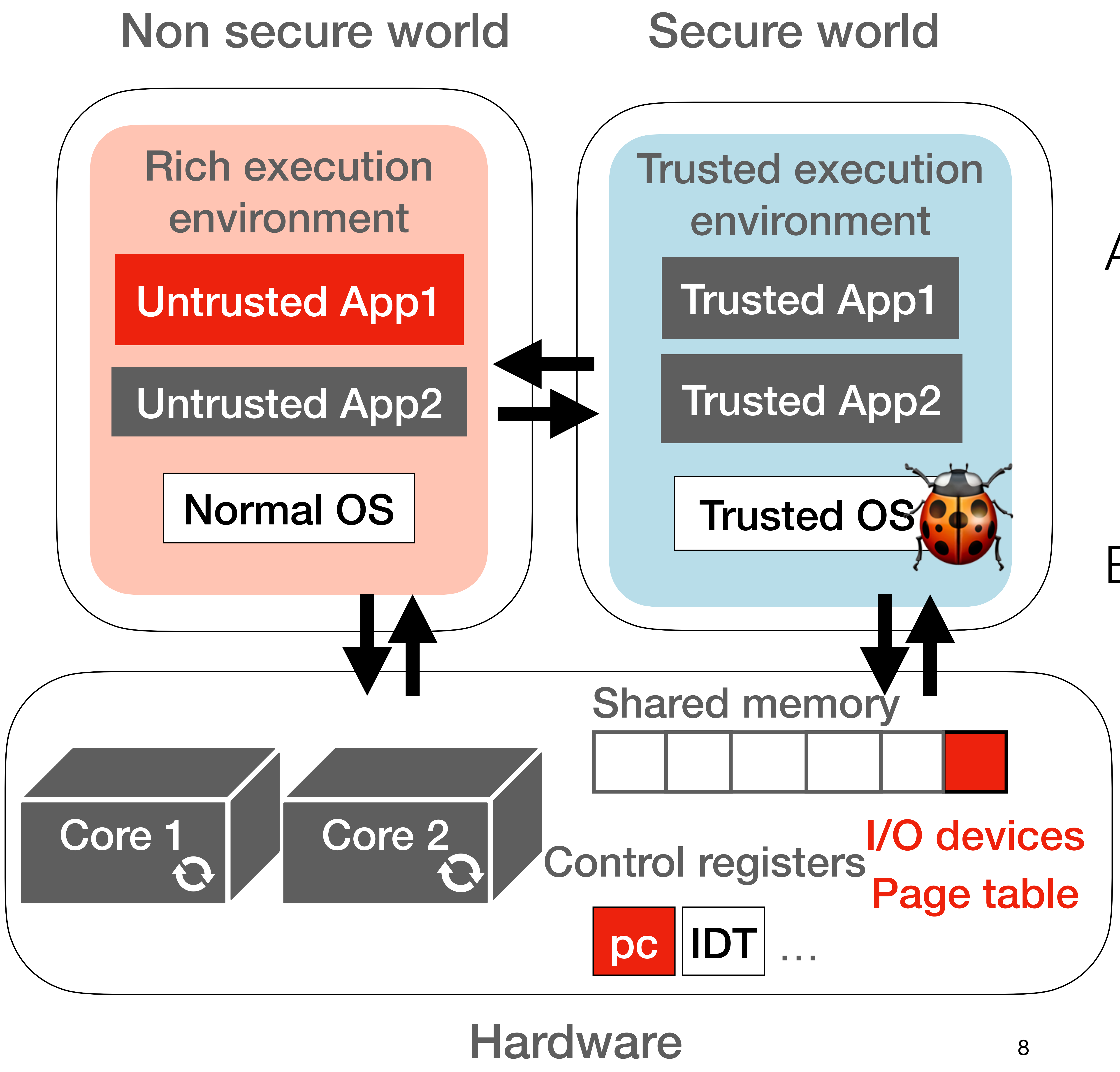
Application

- Cloud computing
- Secure banking

Example Trusted OS

- Hypervisors
- Trusty for Android
- OP-TEE for Arm

Trusted Execution Environments (TEE)



Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

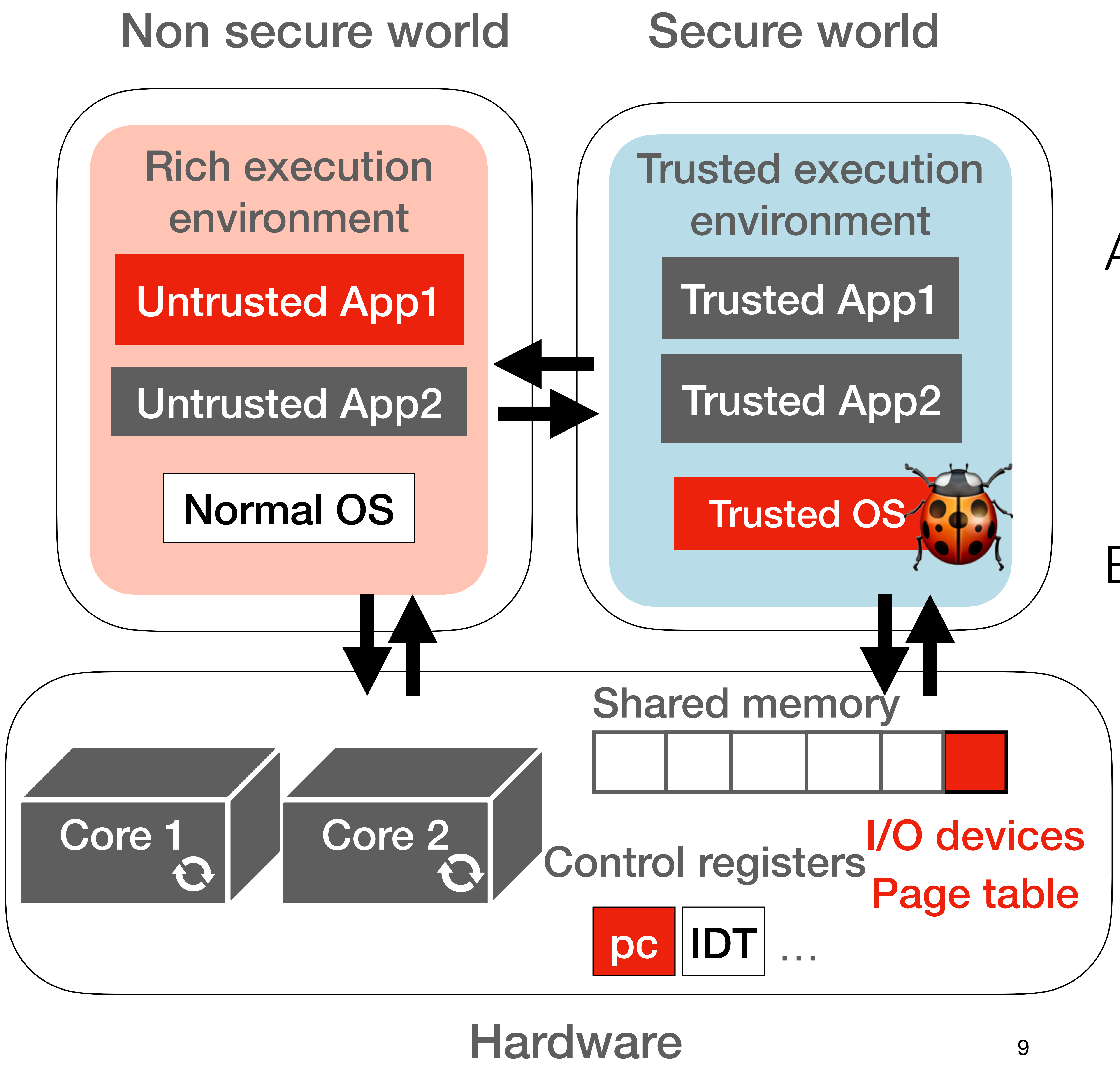
Application

- Cloud computing
- Secure banking

Example Trusted OS

- Hypervisors
- Trusty for Android
- OP-TEE for Arm

Trusted Execution Environments (TEE)



Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

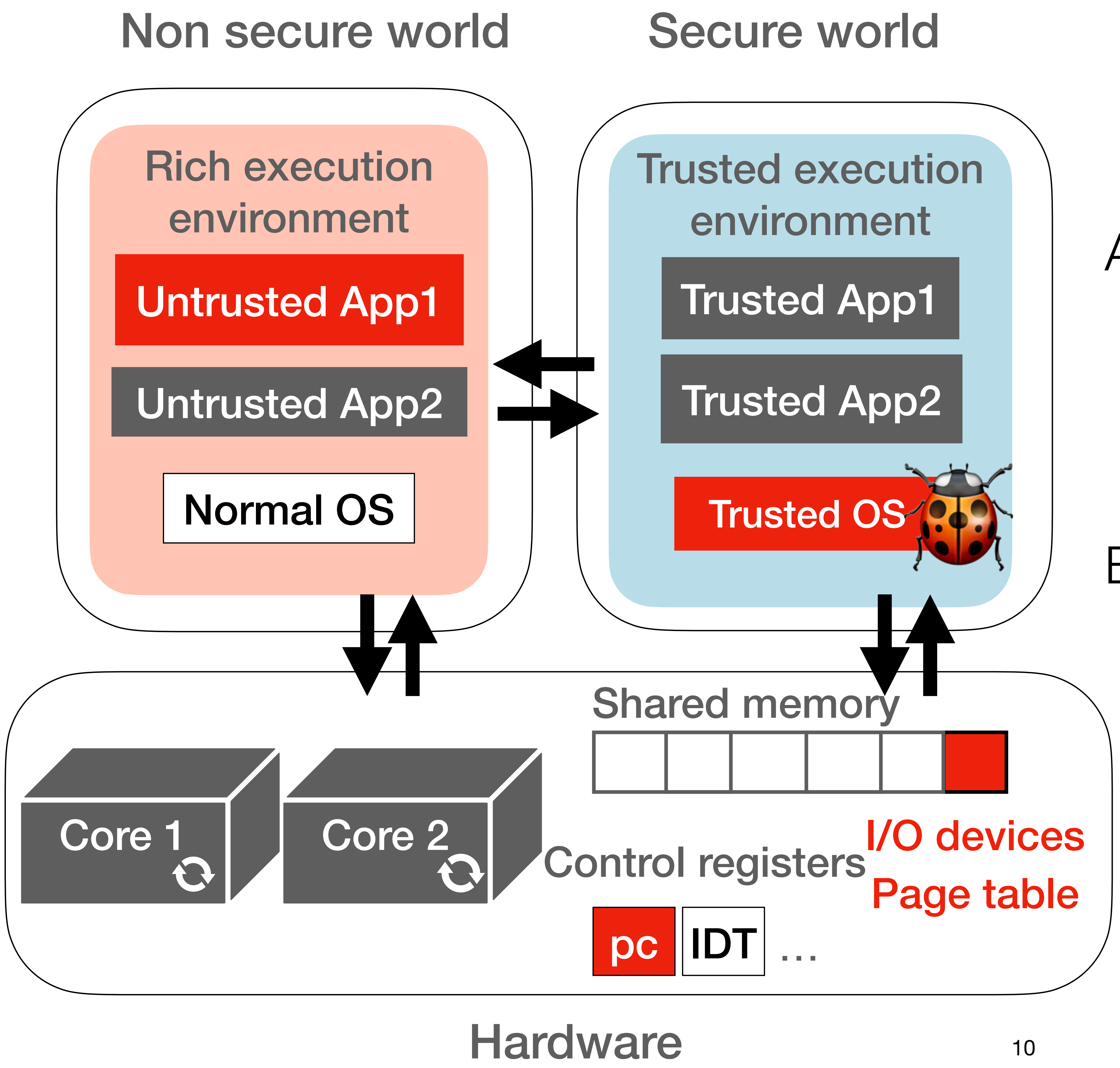
Application

- Cloud computing
- Secure banking

Example Trusted OS

- Hypervisors
- Trusty for Android
- OP-TEE for Arm

Trusted Execution Environments (TEE)



Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

Application

- Cloud computing
- Secure banking

Example Trusted OS

- Hypervisors
- Trusty for Android
- OP-TEE for Arm

Subversion of a TEE means the attacker takes full-control over the entire platform!

TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** *Ironclad apps, sel4, mCertiKOS*

TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** *Ironclad apps, sel4, mCertiKOS*
- **Advantages:** *Strong guarantee*

TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** *Ironclad apps, sel4, mCertiKOS*
- **Advantages:** *Strong guarantee*
- **Disadvantages:** *Not developer friendly, Not update friendly, High cost of verification (time and dollar!)*

TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** *Ironclad apps, sel4, mCertiKOS*
- **Advantages:** *Strong guarantee*
- **Disadvantages:** *Not developer friendly, Not update friendly, High cost of verification (time and dollar!)*

➡ Specific security properties in lieu of full-functional correctness

TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** *Ironclad apps, sel4, mCertiKOS*
- **Advantages:** *Strong guarantee*
- **Disadvantages:** *Not developer friendly, Not update friendly, High cost of verification (time and dollar!)*

➡ Specific security properties in lieu of full-functional correctness

- **Examples:** *XMHF, uberXMHF, Security Microvisor, Contiki*

[1] XMHF: S&P ‘2013. [2] uberXMHF: USENIX Security ’2016. [3] Security MIcrovisor: TDSCM ‘2019. [4] Contiki: DDECS ‘2015

TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** *Ironclad apps, sel4, mCertiKOS*
- **Advantages:** *Strong guarantee*
- **Disadvantages:** *Not developer friendly, Not update friendly, High cost of verification (time and dollar!)*

➡ Specific security properties in lieu of full-functional correctness

- **Examples:** *XMHF, uberXMHF, Security Microvisor, Contiki*
- **Advantages:** *Development friendly, use source-level automated verification tools*

[1] XMHF: S&P ‘2013. [2] uberXMHF: USENIX Security ’2016. [3] Security MIcrovisor: TDSCM ‘2019. [4] Contiki: DDECS ‘2015

TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** *Ironclad apps, sel4, mCertiKOS*
- **Advantages:** *Strong guarantee*
- **Disadvantages:** *Not developer friendly, Not update friendly, High cost of verification (time and dollar!)*

➡ Specific security properties in lieu of full-functional correctness

- **Examples:** *XMHF, uberXMHF, Security Microvisor, Contiki*
- **Advantages:** *Development friendly, use source-level automated verification tools*
- **Disadvantages:** *Weaker guarantee*

[1] XMHF: S&P ‘2013. [2] uberXMHF: USENIX Security ’2016. [3] Security MIcrovisor: TDSCM ‘2019. [4] Contiki: DDECS ‘2015

Specific security properties in lieu of full-functional correctness

- **Examples:** *XMHF, uberXMHF, Security Microvisor, Contiki*
- **Advantages:** *Development friendly, use source-level automated verification tools*
- **Disadvantages:** *Weaker guarantee*

Specific security properties in lieu of full-functional correctness

- **Examples:** *XMHF, uberXMHF, Security Microvisor, Contiki*
- **Advantages:** *Development friendly, use source-level automated verification tools*
- **Disadvantages:** *Weaker guarantee*

Prior approaches lack guarantees on the compiled code

Specific security properties in lieu of full-functional correctness

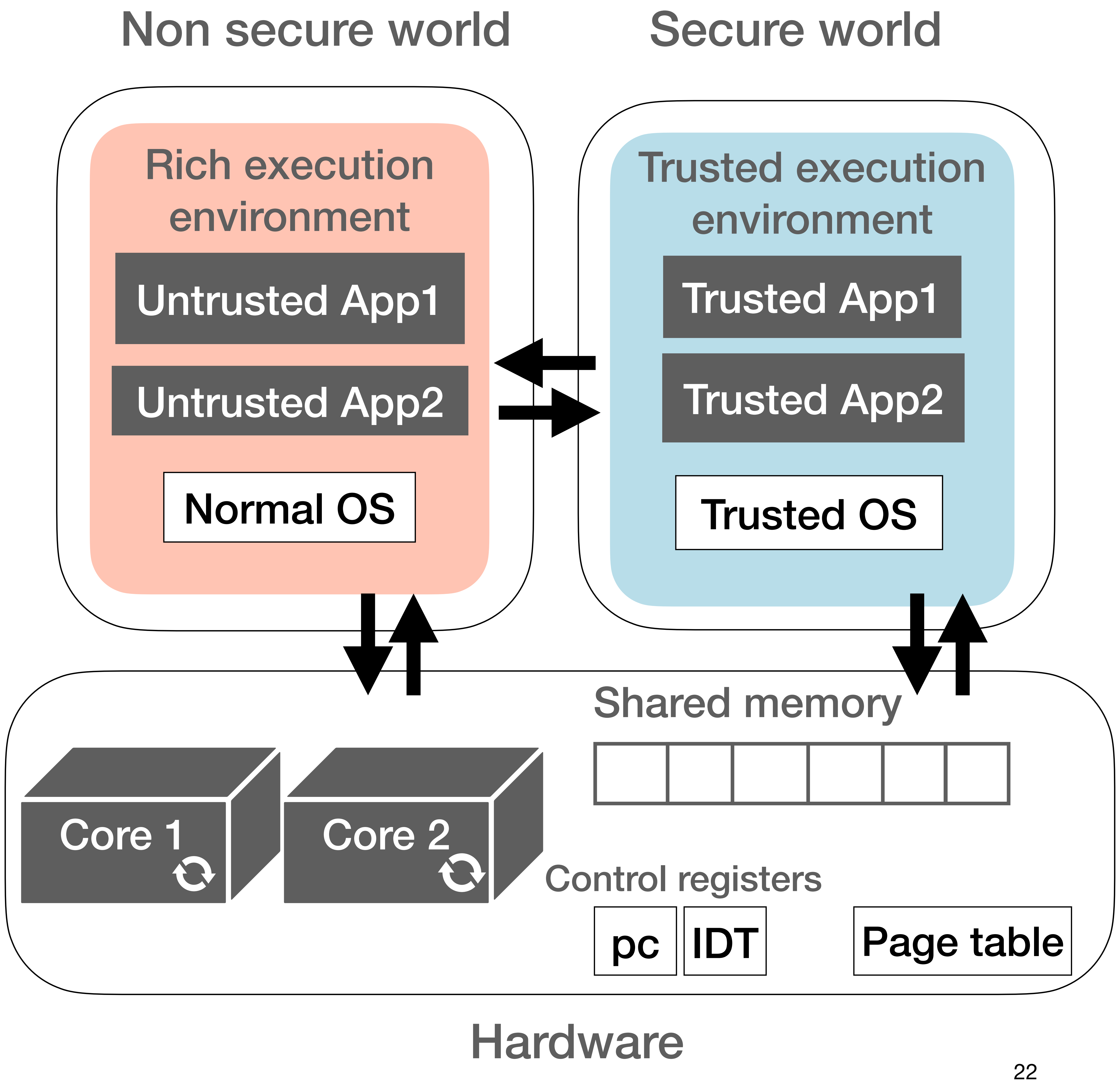
- **Examples:** *XMHF, uberXMHF, Security Microvisor, Contiki*
- **Advantages:** *Development friendly, use source-level automated verification tools*
- **Disadvantages:** *Weaker guarantee*

Prior approaches lack guarantees on the compiled code

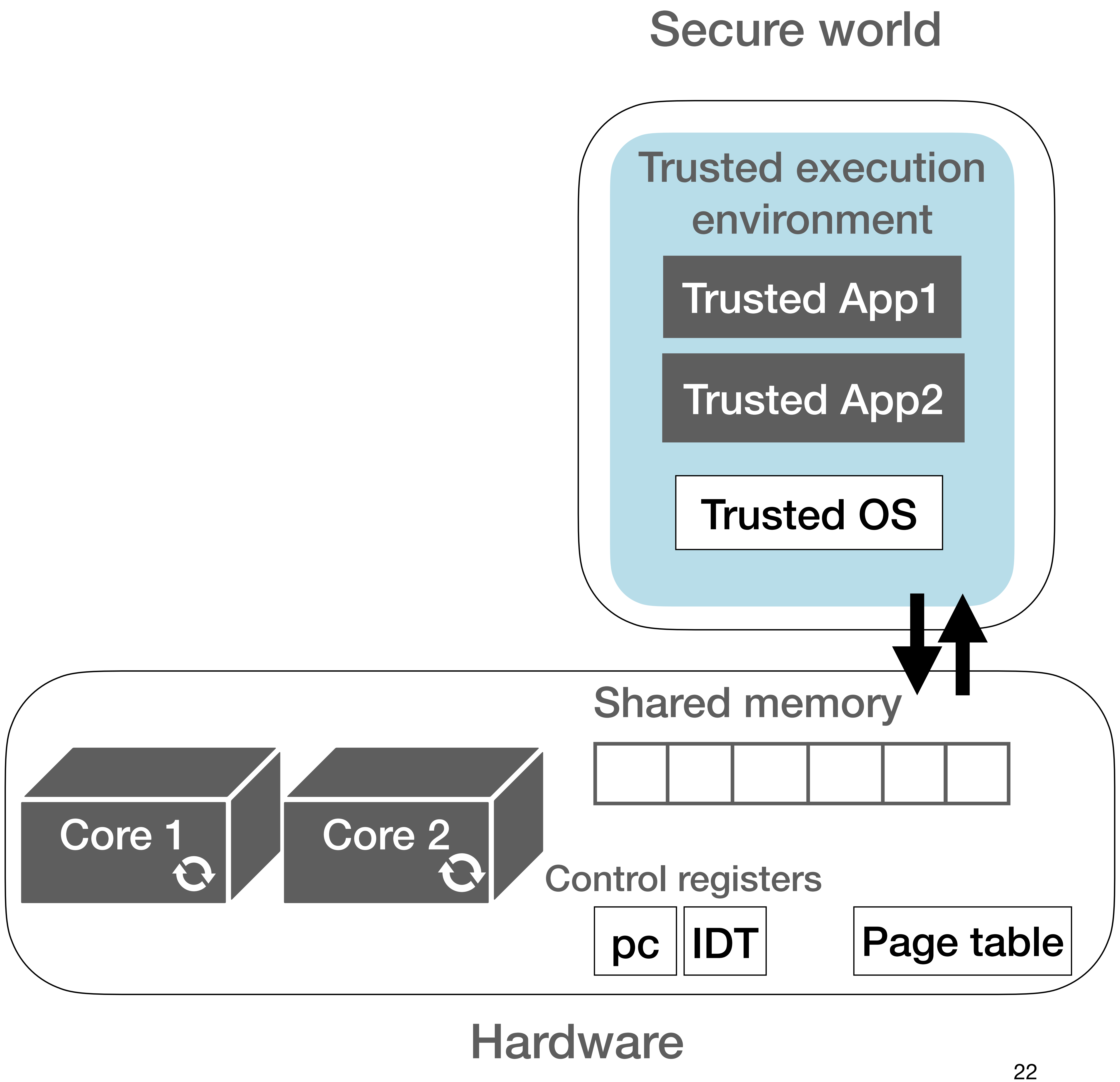
Our approach - Compartmentalization and certified compilers to aid verification:

- Compartments as units for verification and compilation.
- Allows us to bring the security properties down to the compiled code.

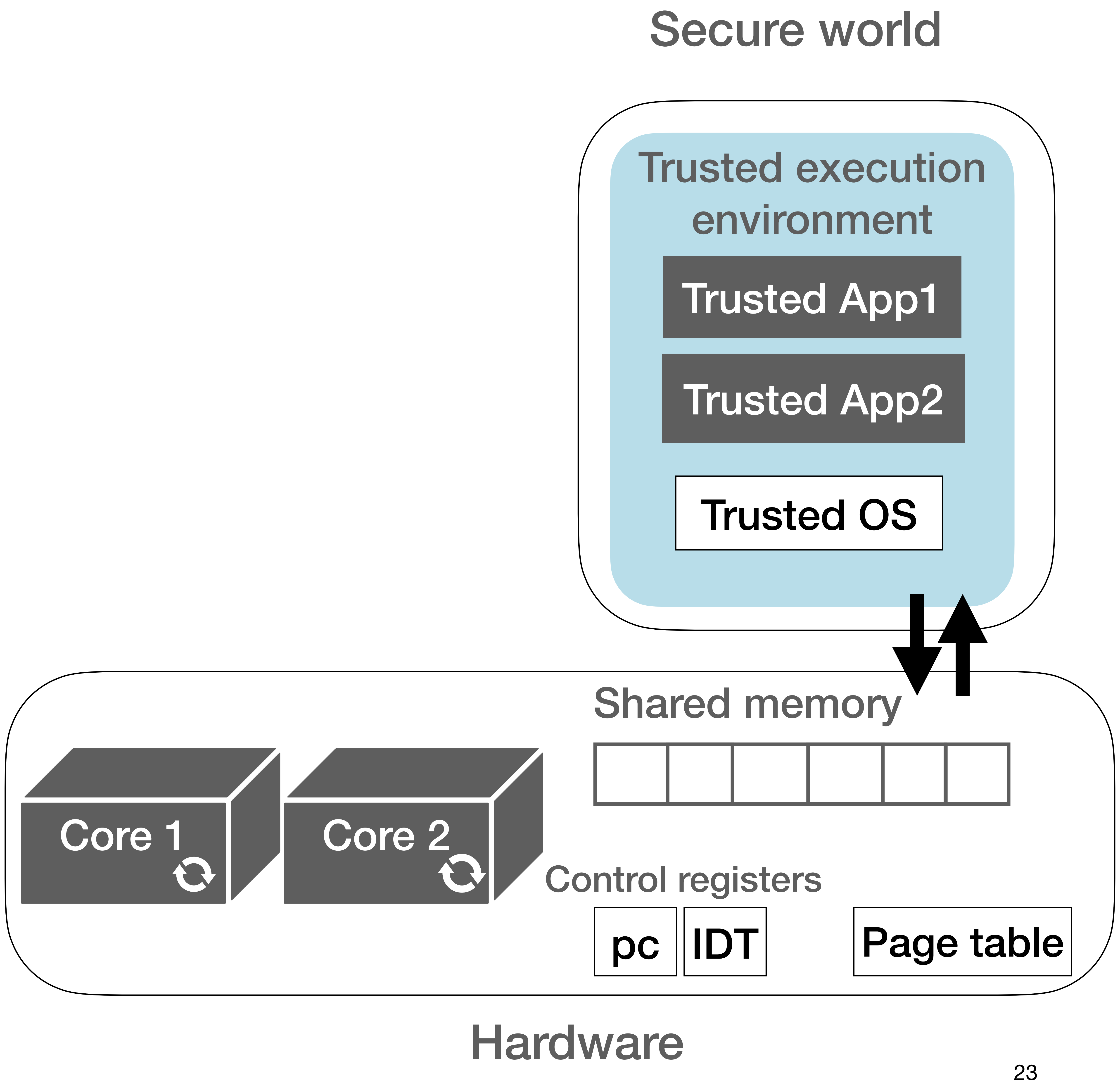
Compartments schema



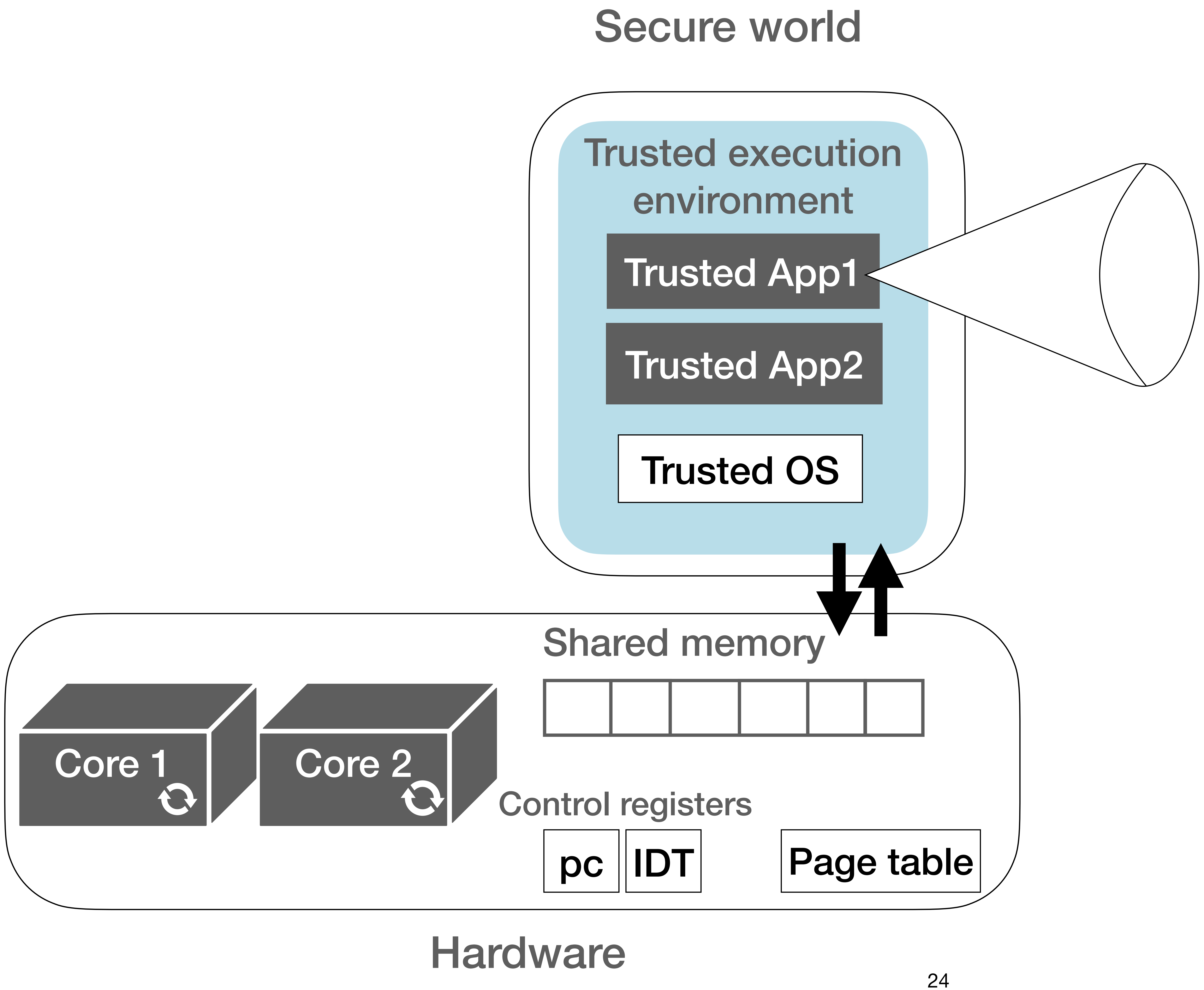
Compartments schema



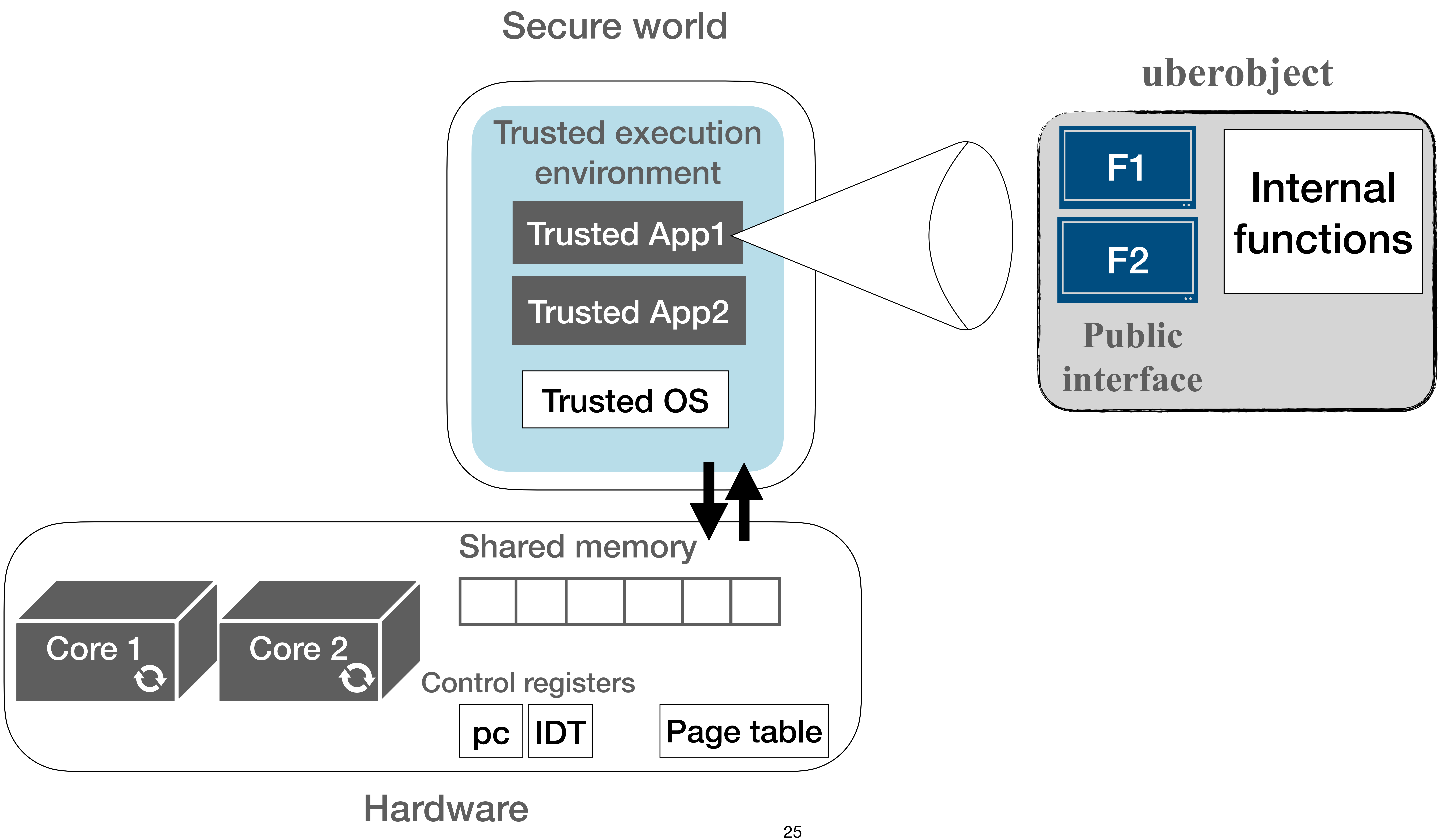
Compartments schema



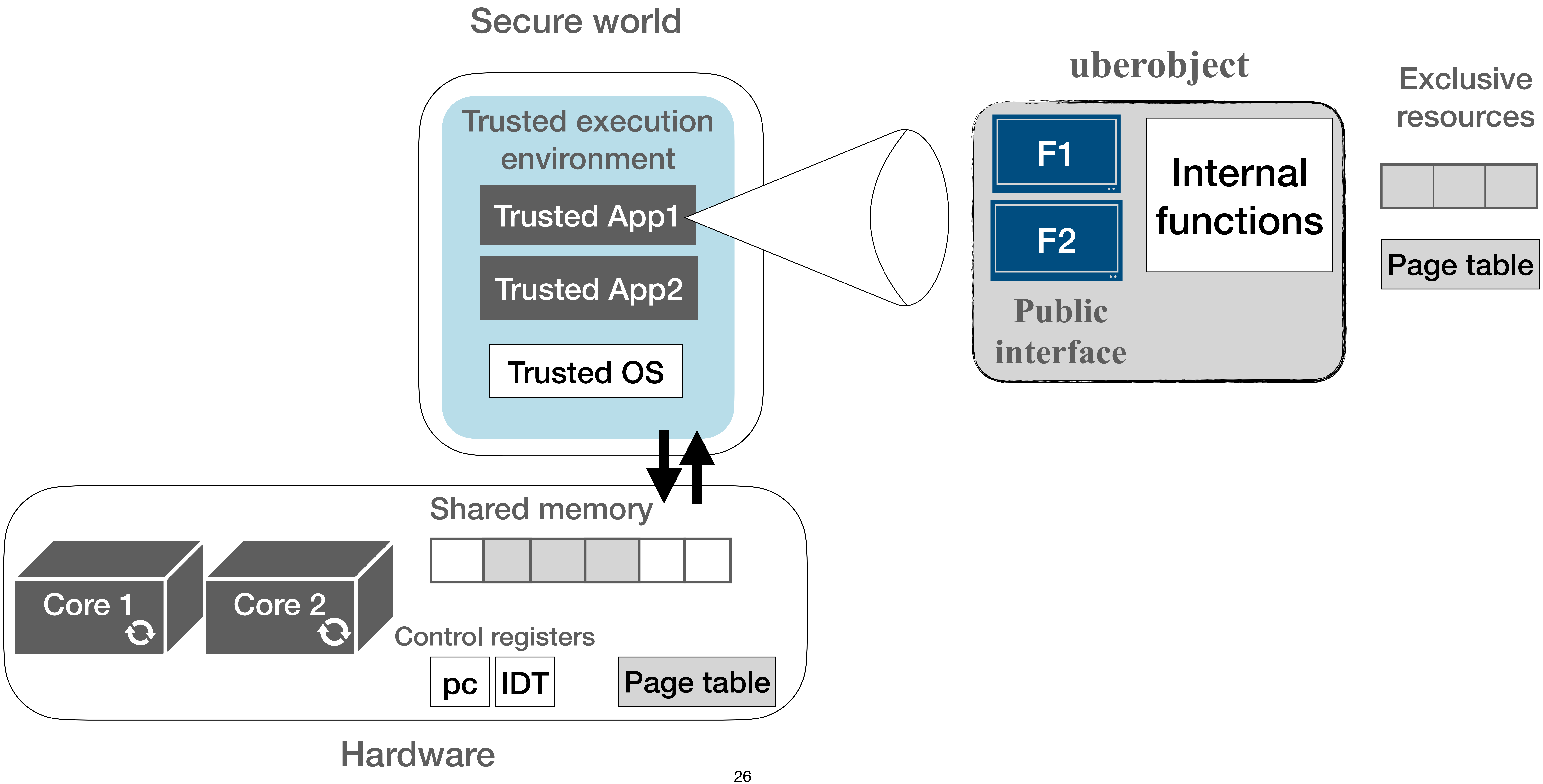
Compartments schema



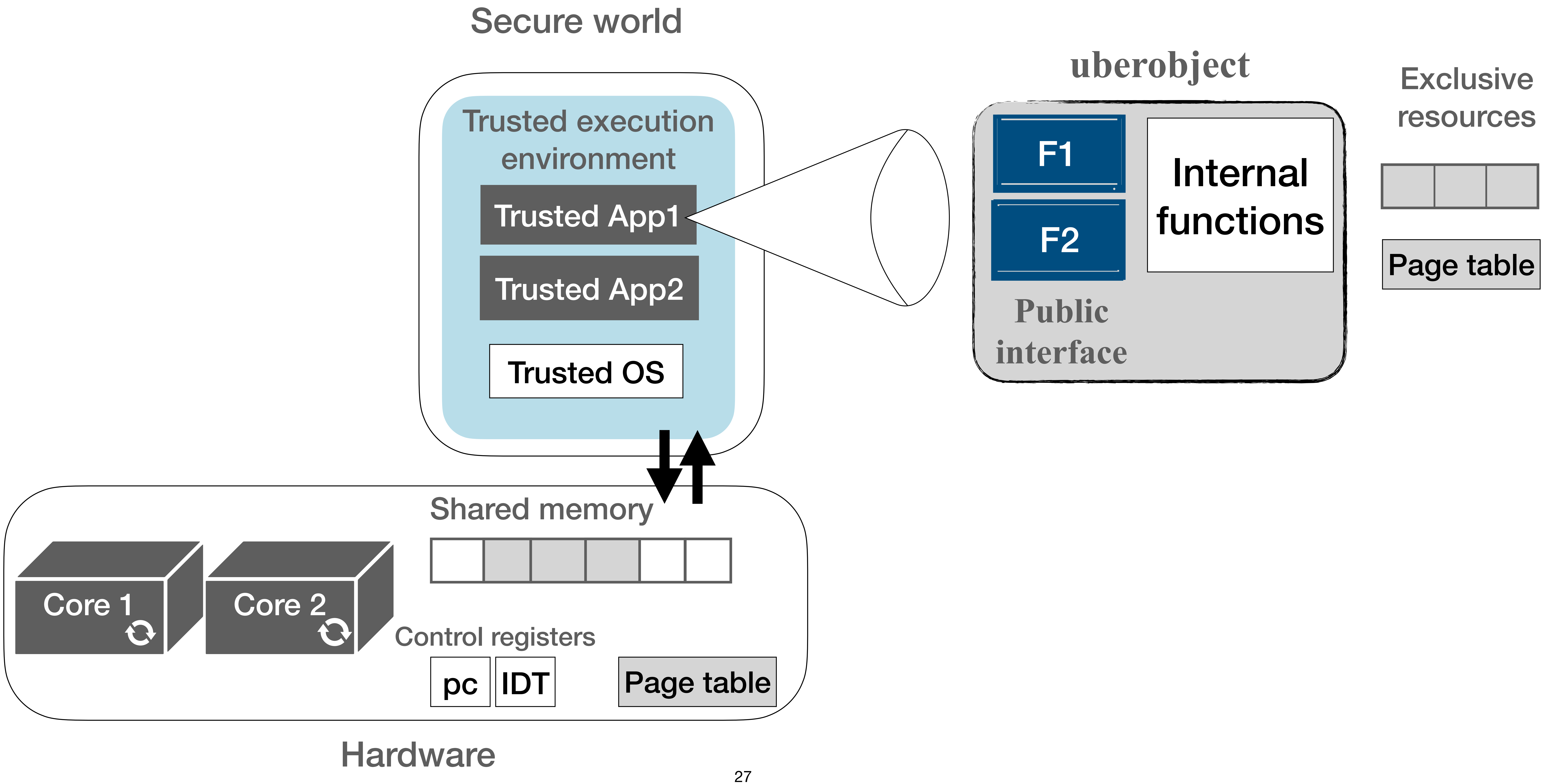
Compartments schema



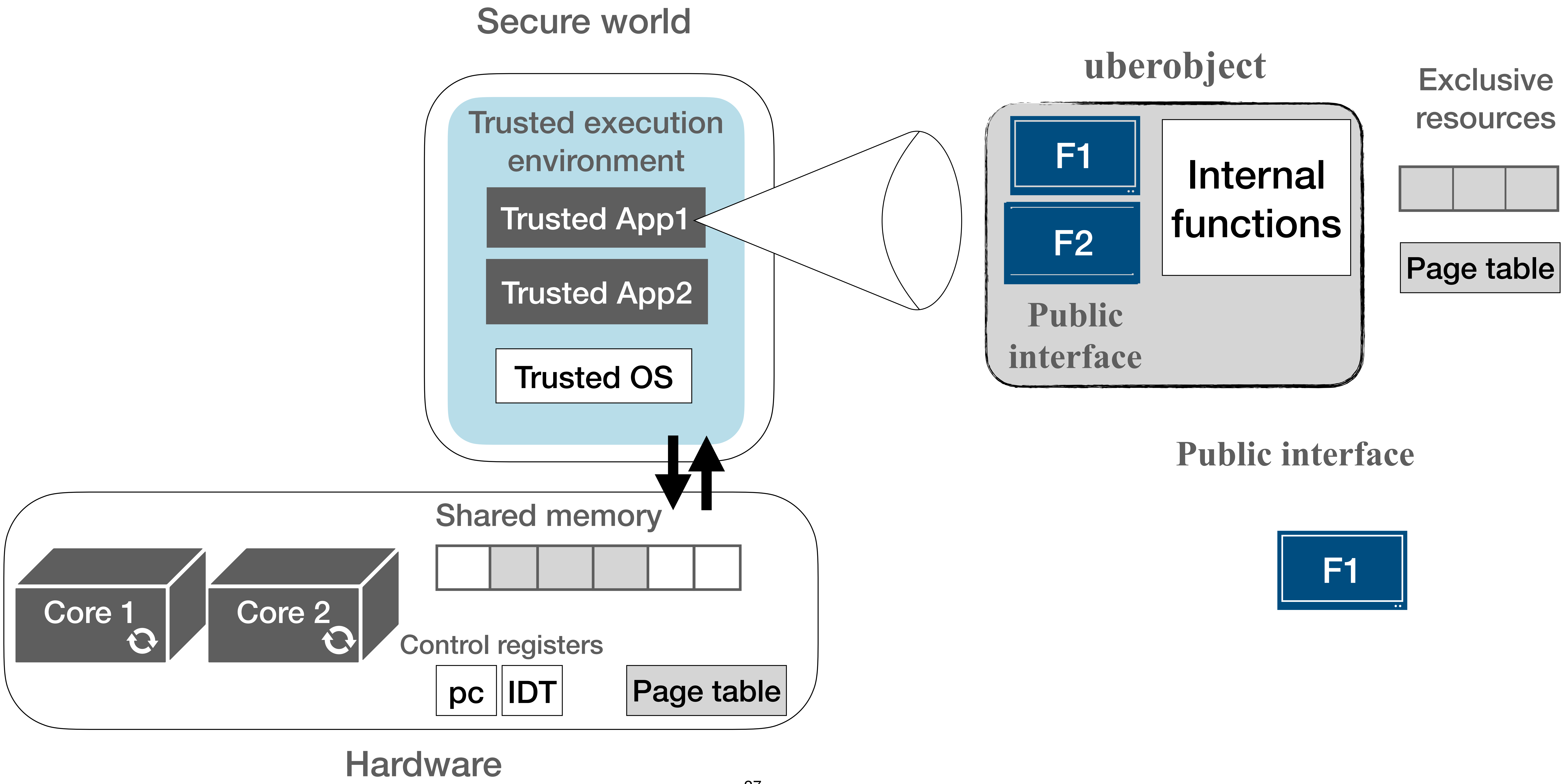
Compartments schema



Compartments schema



Compartments schema



Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

Shared memory

Core 1

Core 2

Control registers

pc

IDT

Page table

Hardware

uberobject

F1

F2

Internal functions

Public interface

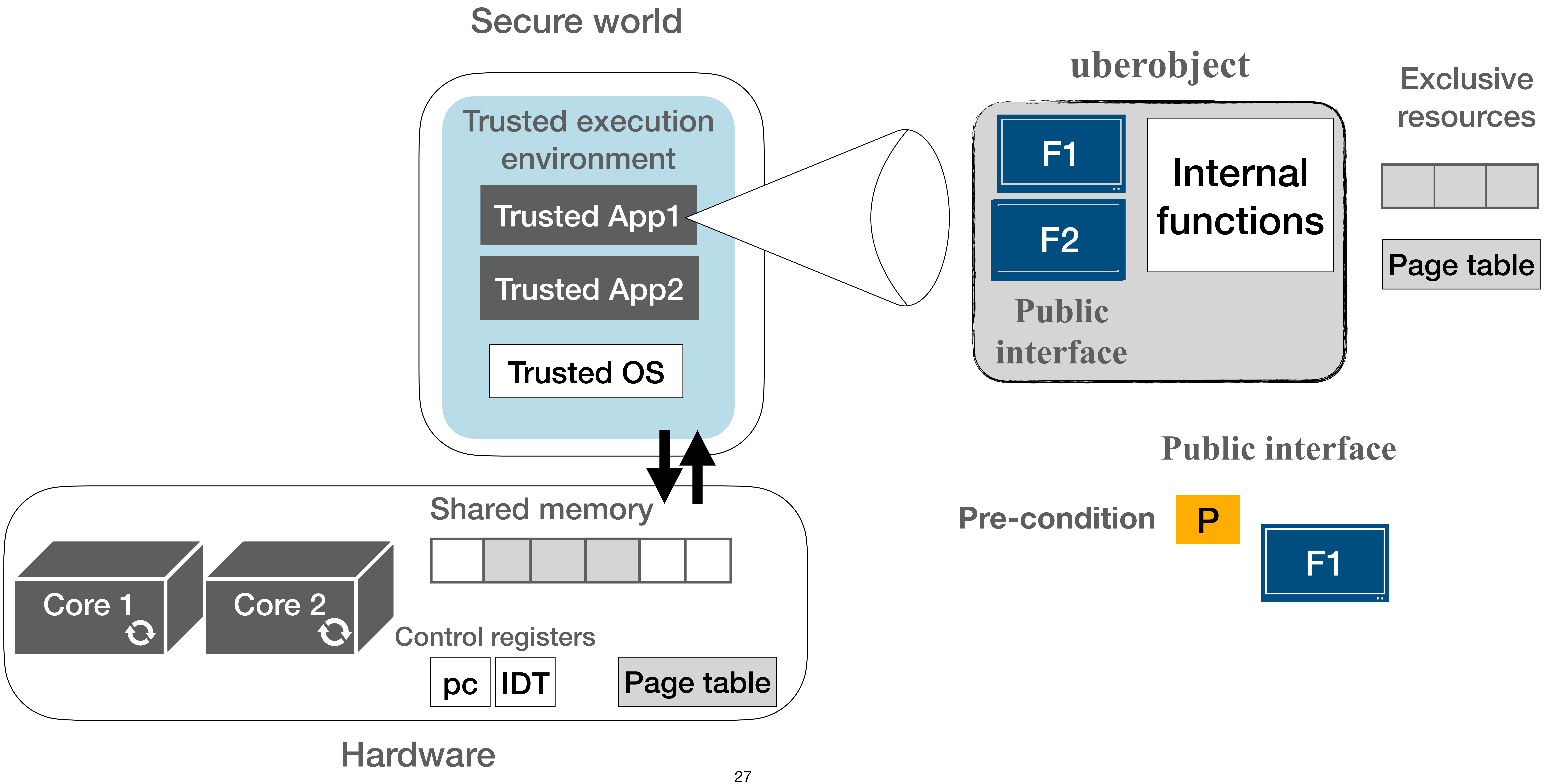
Exclusive resources

Page table

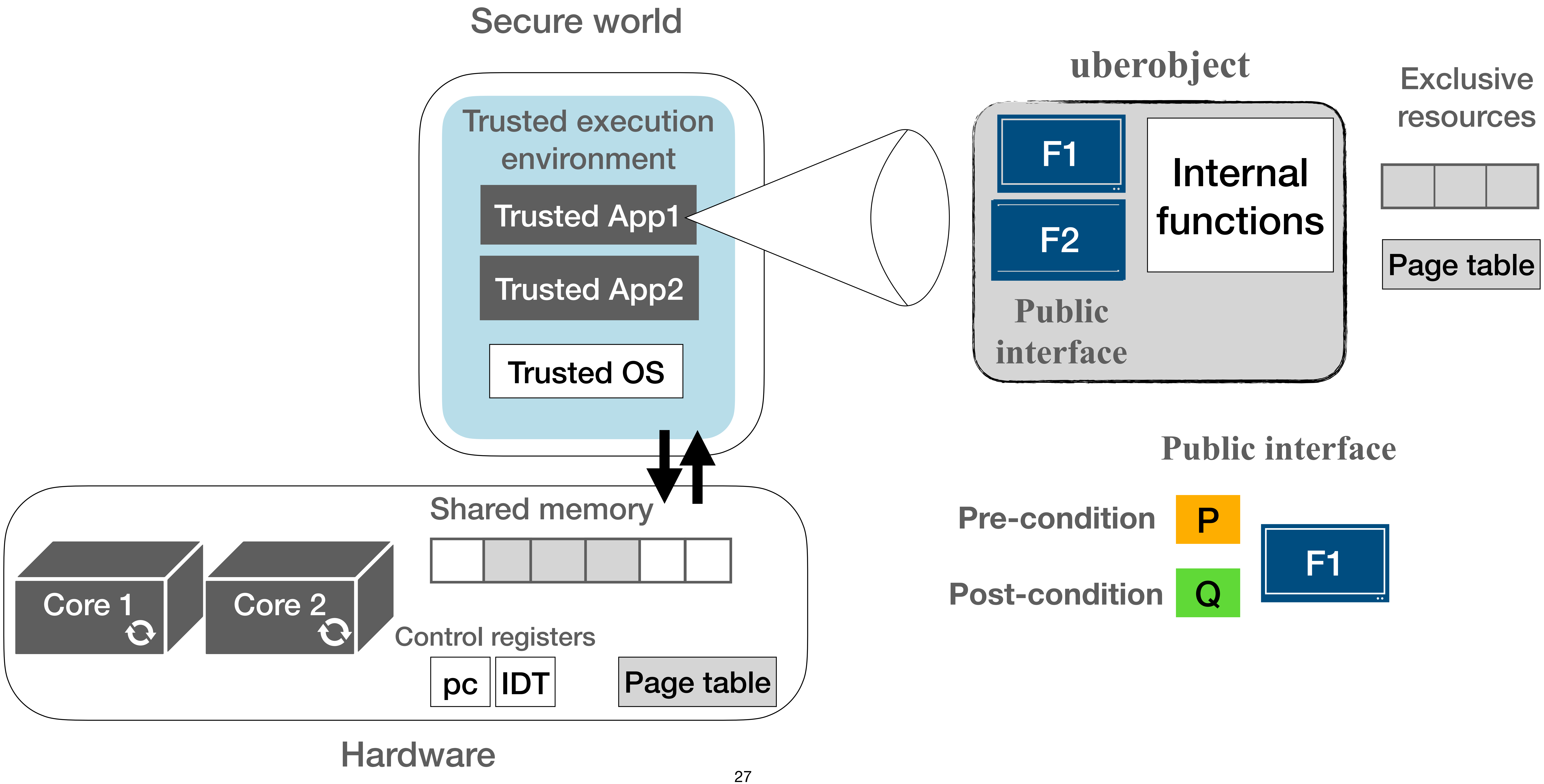
Public interface

F1

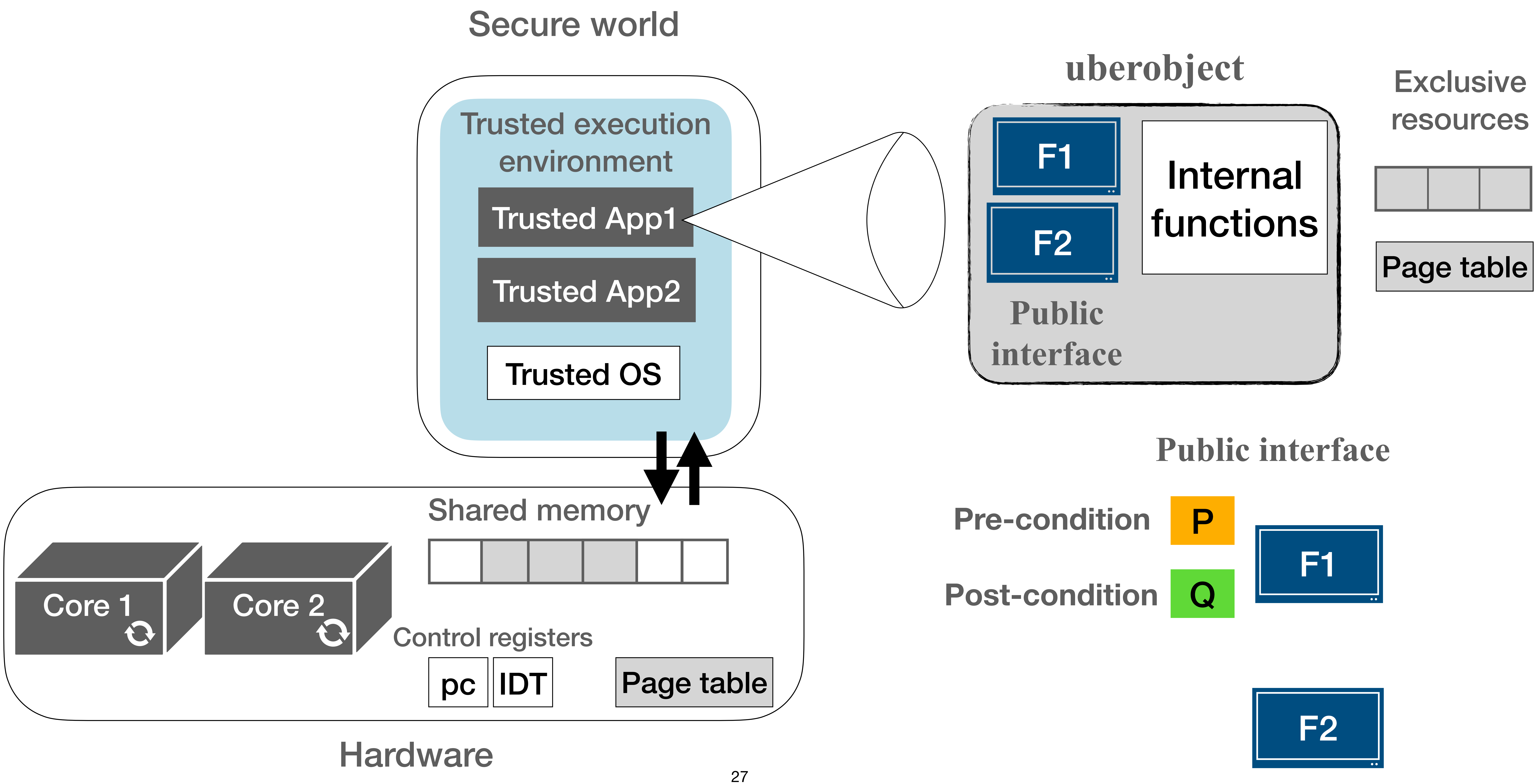
Compartments schema



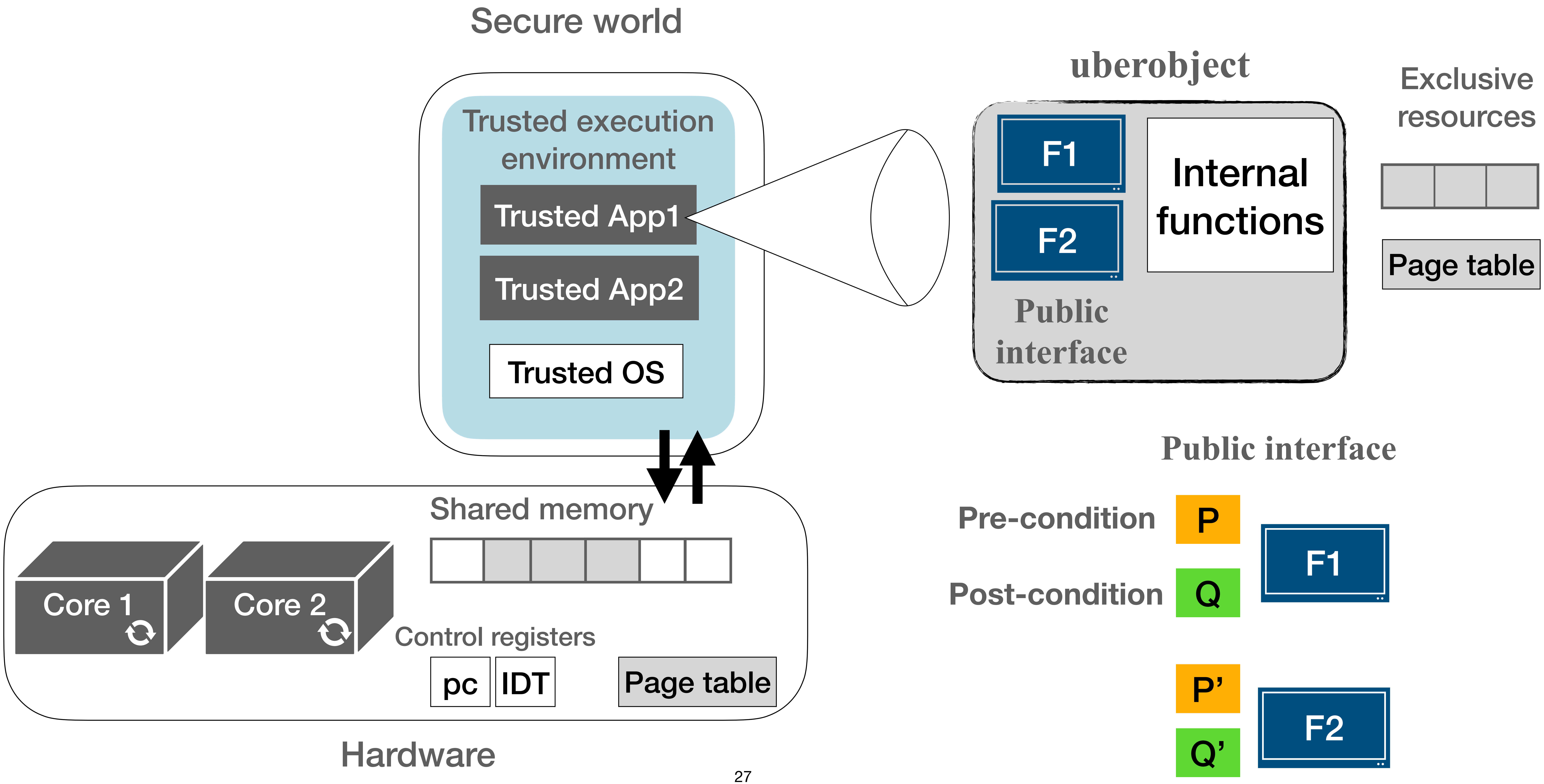
Compartments schema



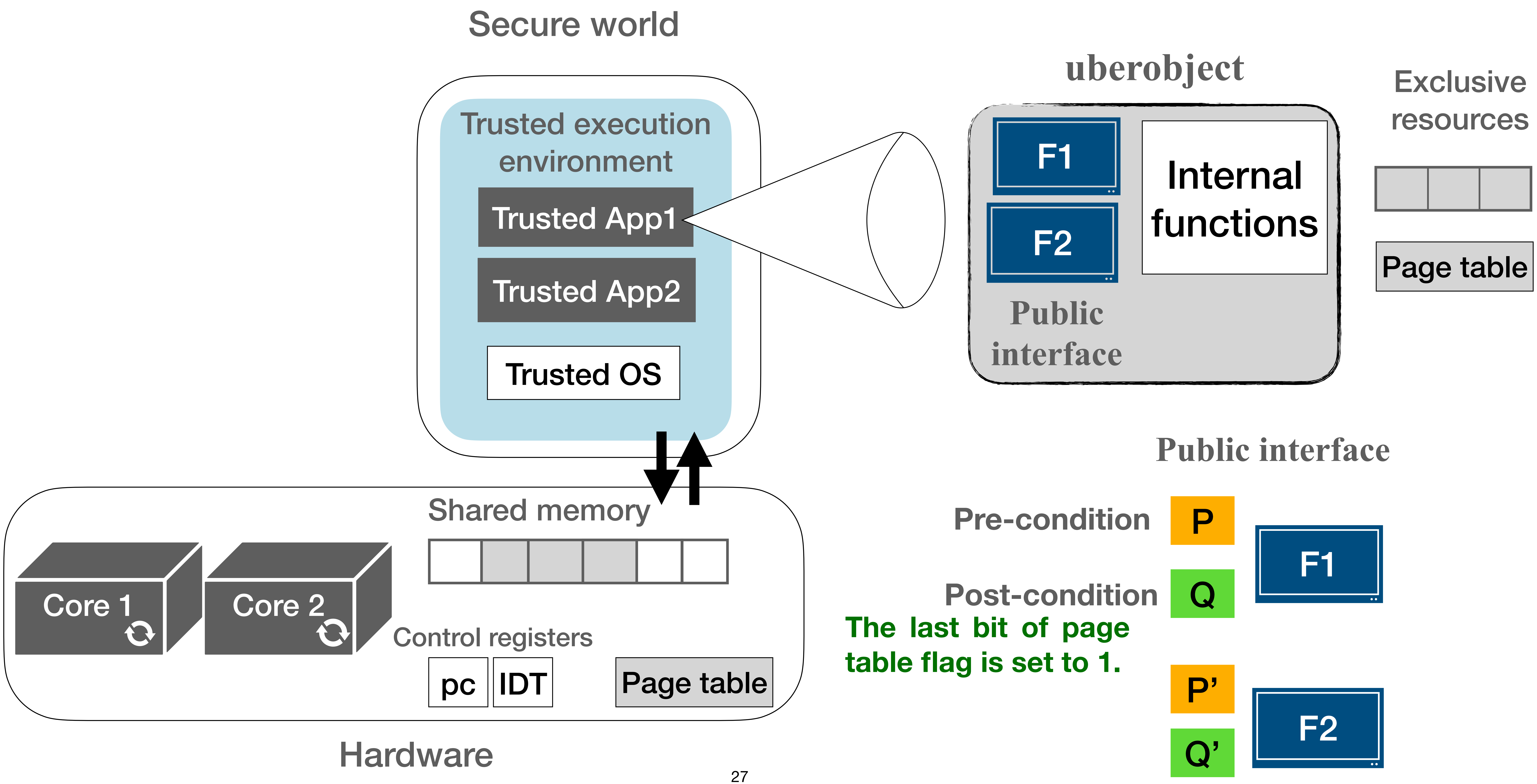
Compartments schema



Compartments schema



Compartments schema

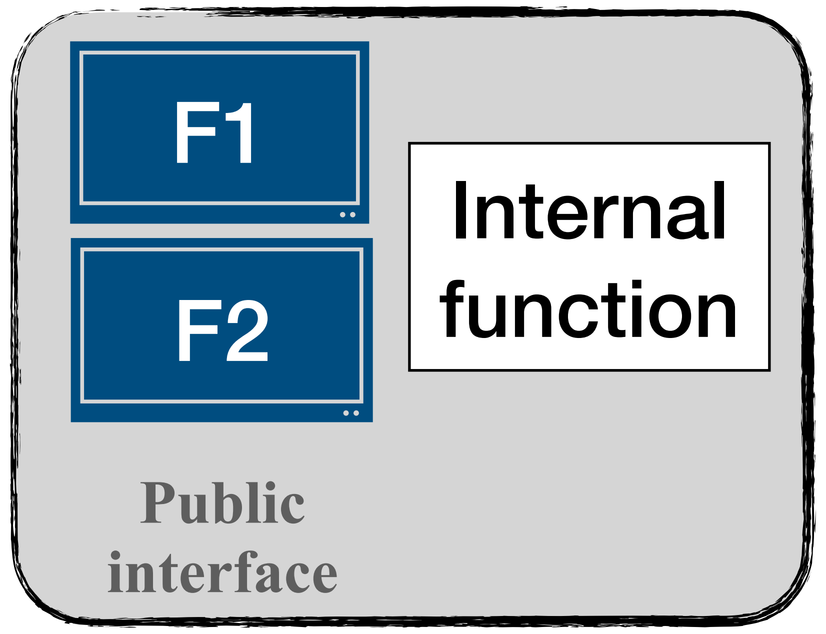


The last bit of page table flag is set to 1.

Compartments as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

Source-level compartments

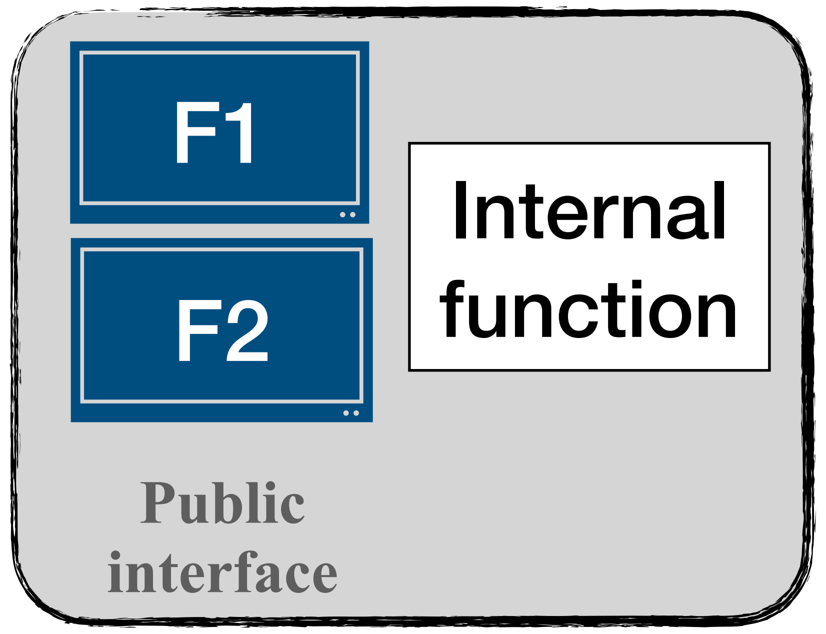


uberobject 1

Compartments as units of verification and compilation

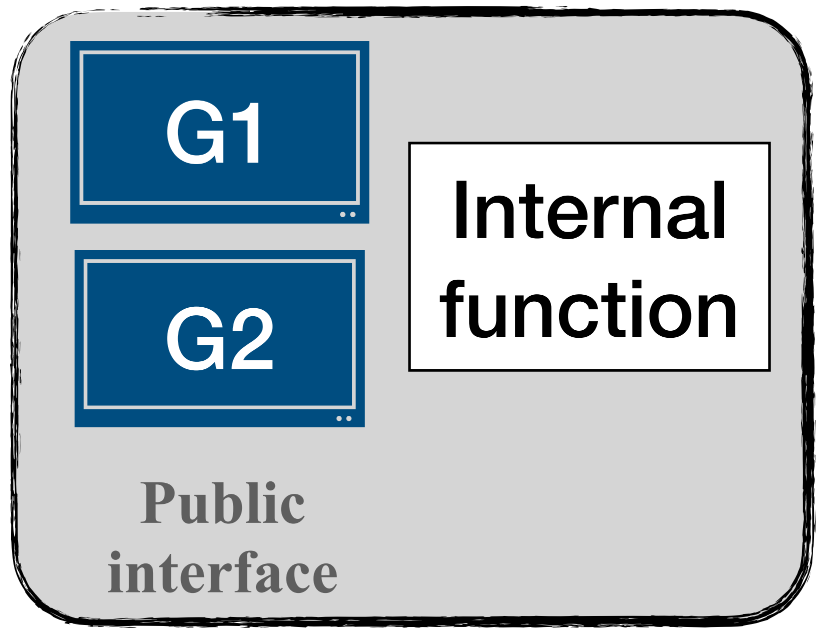
Source-level compartments

The last bit of page table flag is set to 1 (after function return)



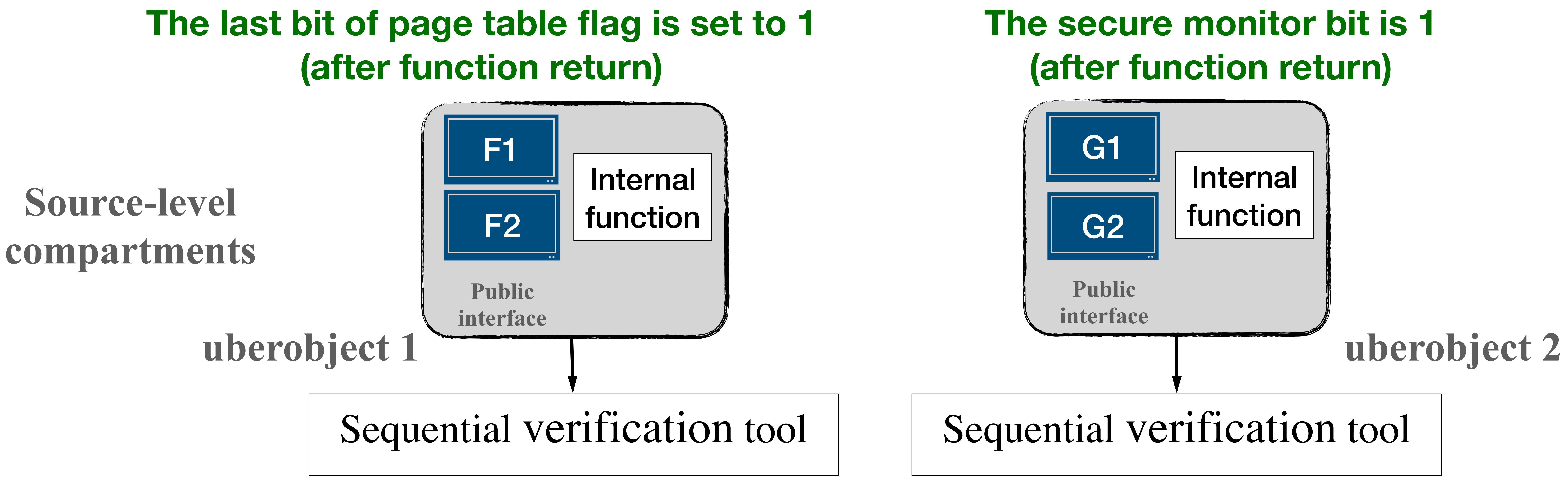
uberobject 1

The secure monitor bit is 1 (after function return)

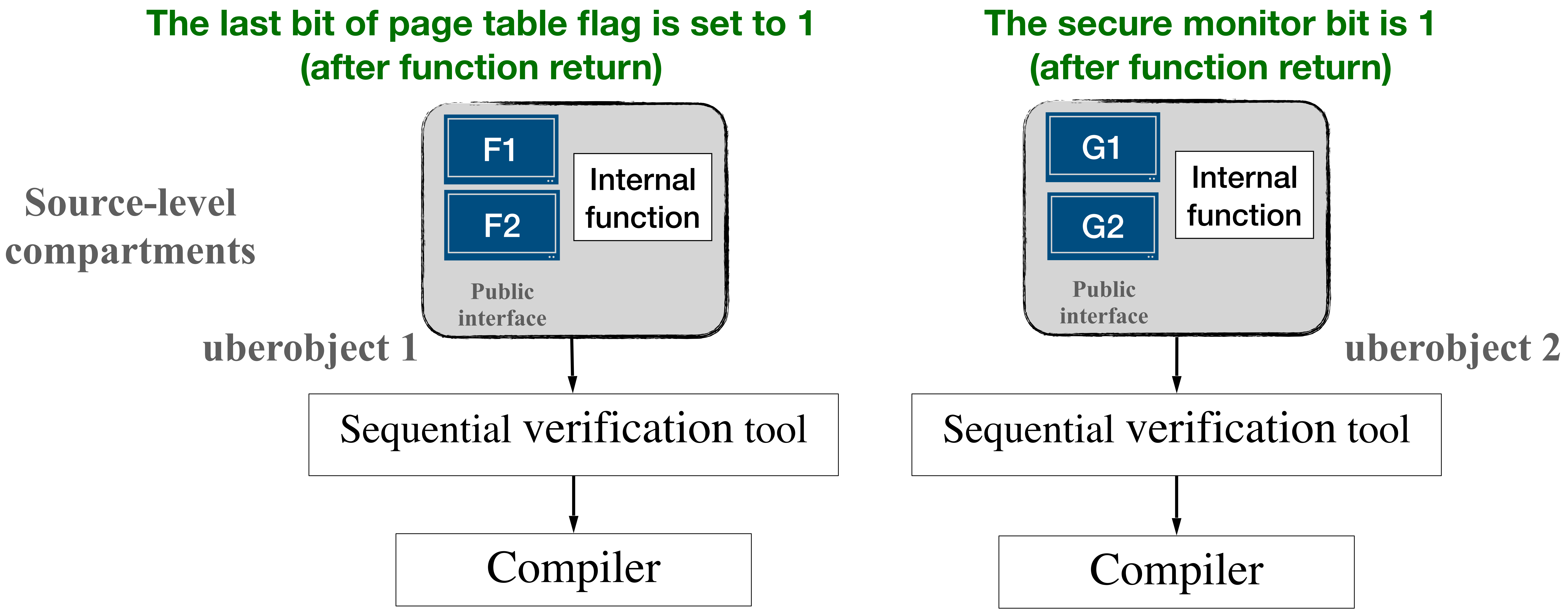


uberobject 2

Compartments as units of verification and compilation



Compartments as units of verification and compilation



The last bit of page table flag is set to 1 (after function return)

The secure monitor bit is 1 (after function return)

Source-level compartments

F1

F2

Internal function

Public interface

uberobject 1

Sequential verification tool

Compiler

G1

G2

Internal function

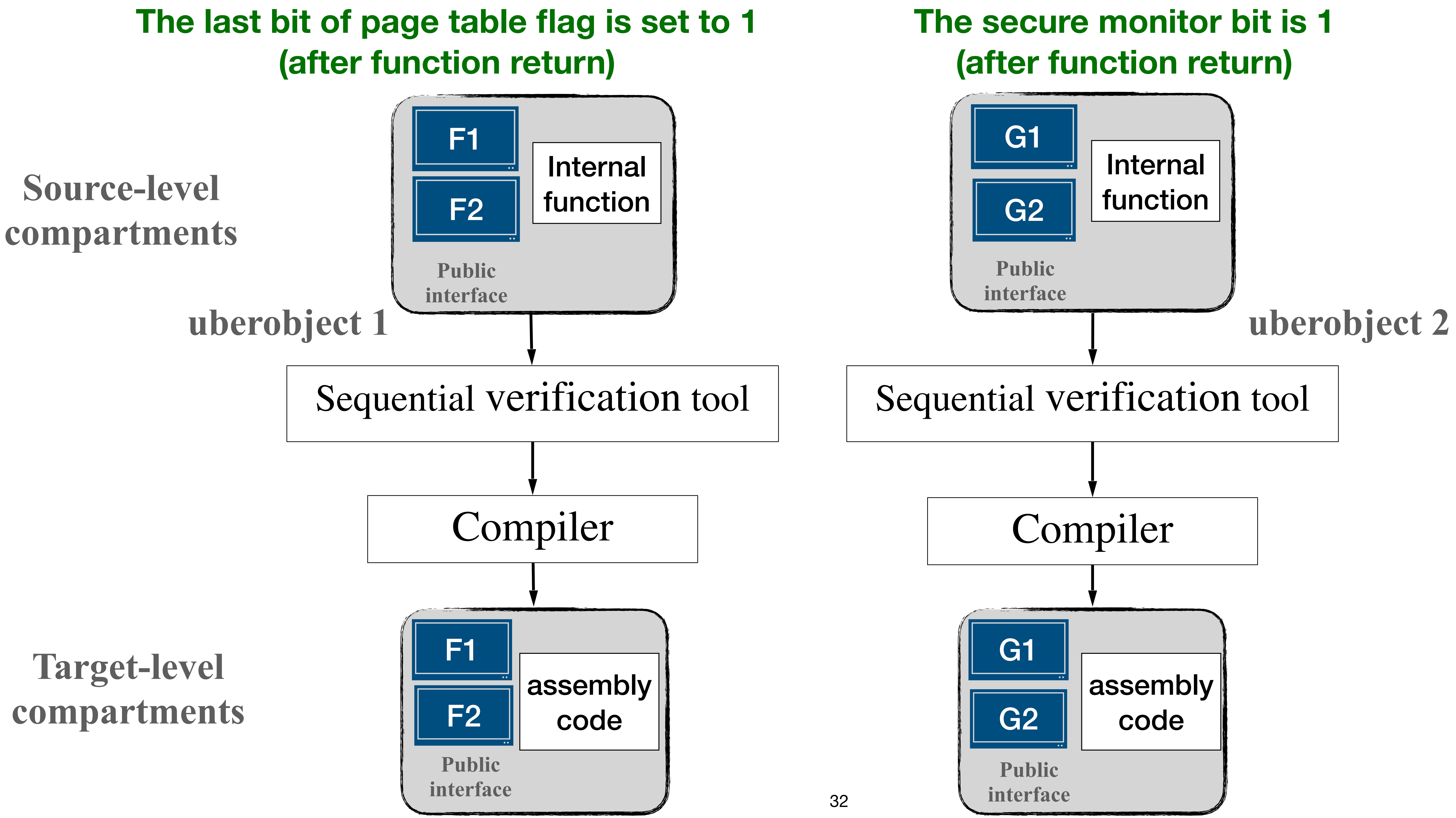
Public interface

uberobject 2

Sequential verification tool

Compiler

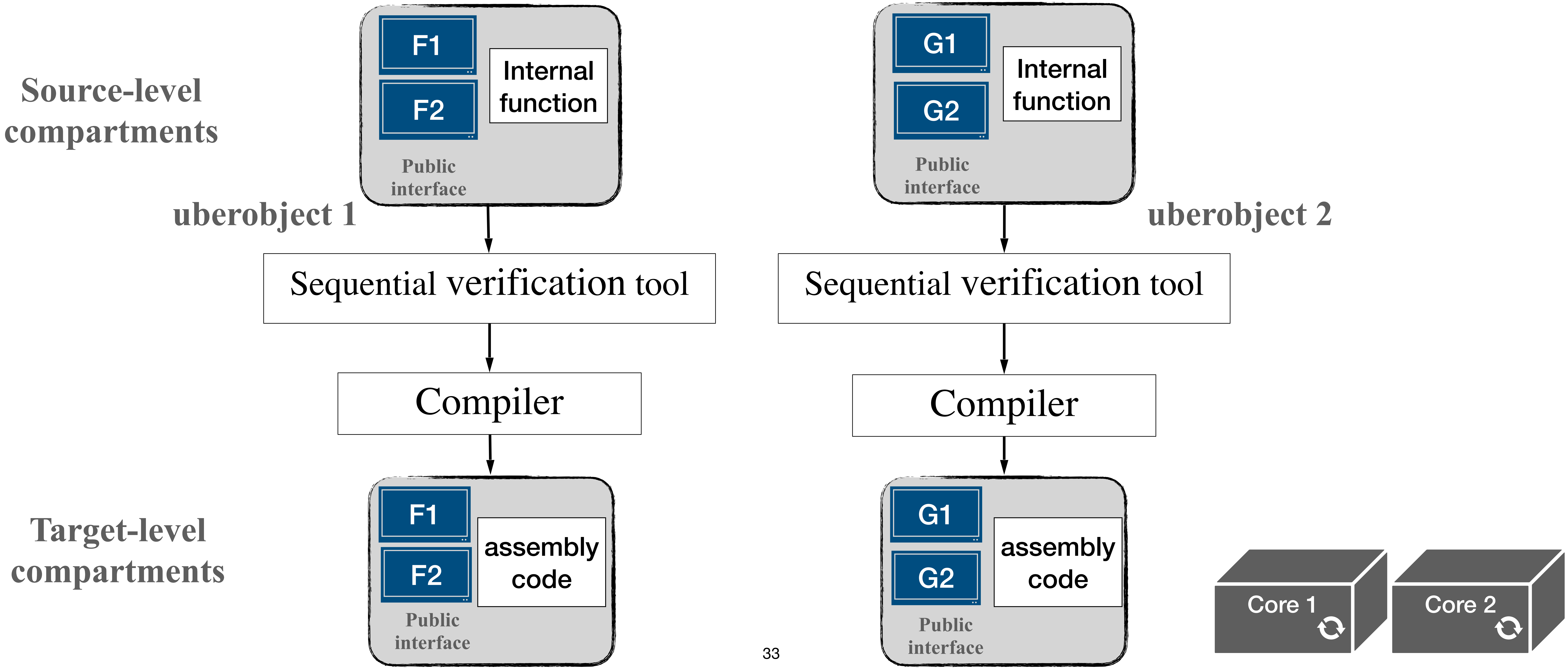
Compartments as units of verification and compilation



Compartments as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

The secure monitor bit is 1 (after function return)



Source-level compartments

F1

F2

Internal function

Public interface

uberobject 1

Sequential verification tool

Compiler

F1

F2

assembly code

Public interface

G1

G2

Internal function

Public interface

uberobject 2

Sequential verification tool

Compiler

G1

G2

assembly code

Public interface

Target-level compartments

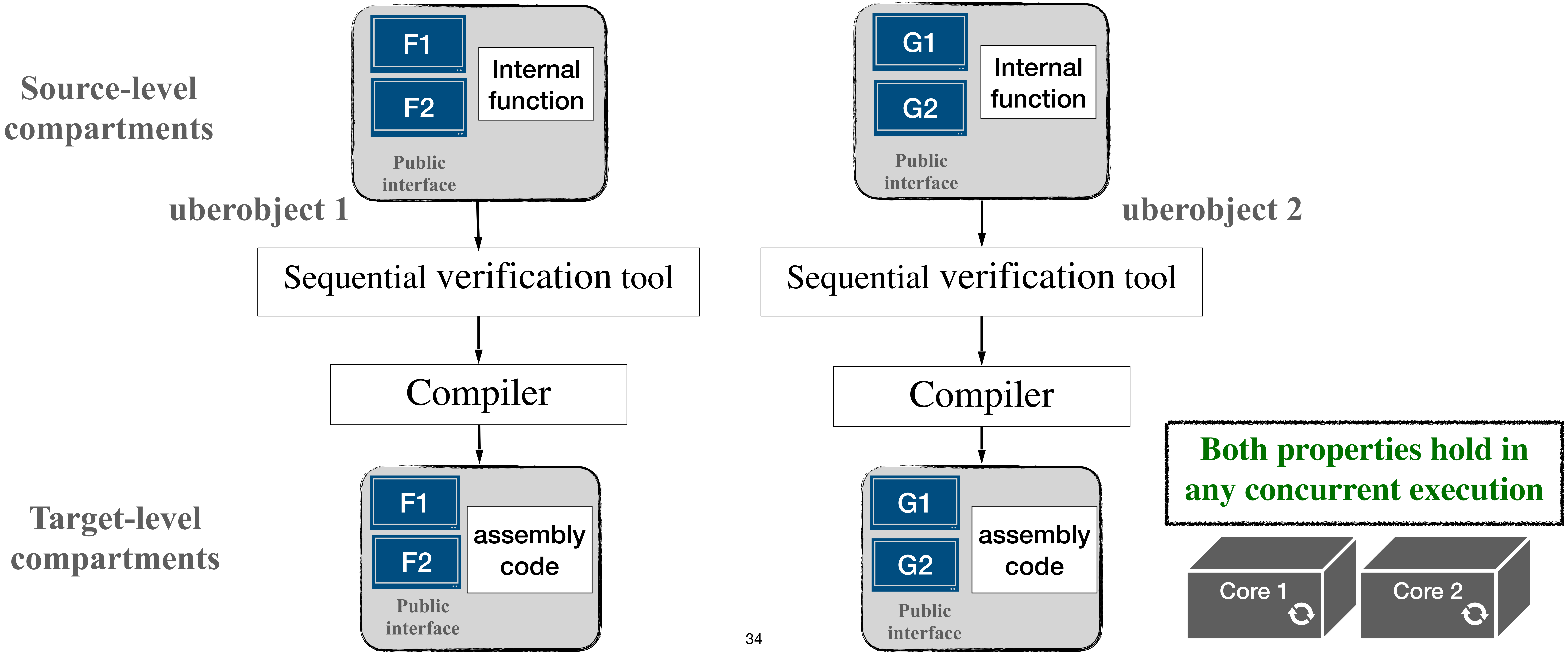
Core 1

Core 2

Compartments as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

The secure monitor bit is 1 (after function return)



Source-level compartments

F1

F2

Internal function

Public interface

uberobject 1

G1

G2

Internal function

Public interface

uberobject 2

Sequential verification tool

Sequential verification tool

Compiler

Compiler

Target-level compartments

F1

F2

assembly code

Public interface

G1

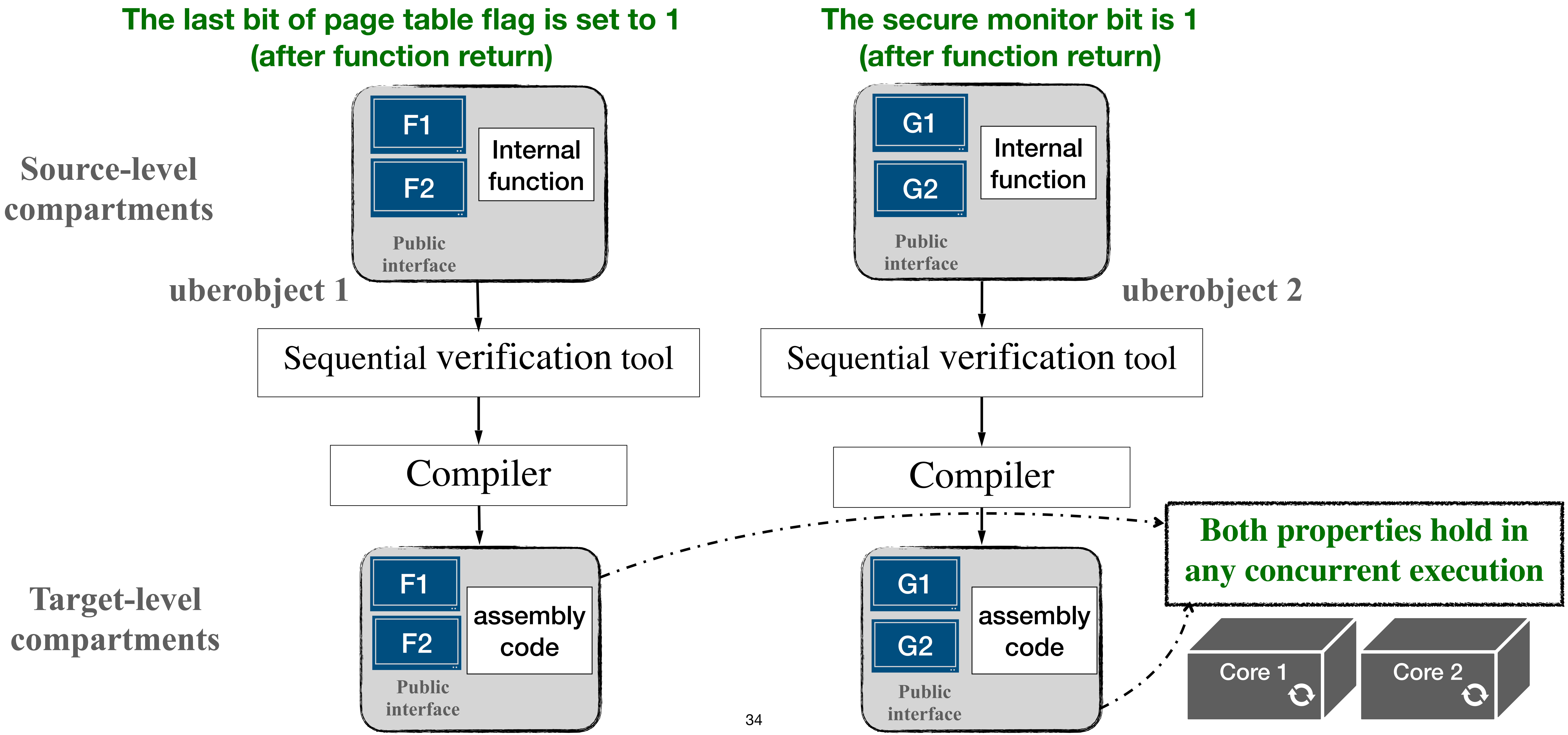
G2

assembly code

Public interface

Both properties hold in any concurrent execution

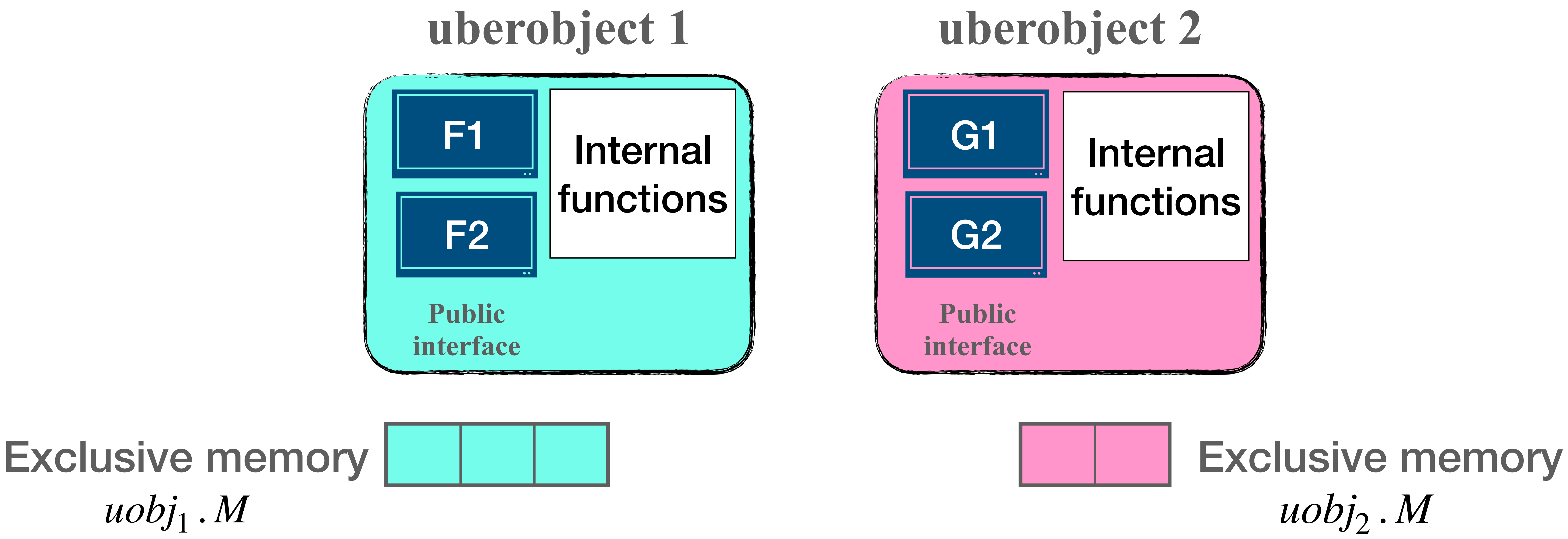
Compartments as units of verification and compilation



Outline

- **Concurrent execution** - an example
- **Verify source-level guarantees**
- **Preserve target-level guarantees**
- **Using off-the-shelf tools**
- **Case studies**
- **Related work**

Concurrent execution



uberobject 1

F1

F2

Internal functions

Public interface

Exclusive memory

$uobj_1 . M$

uberobject 2

G1

G2

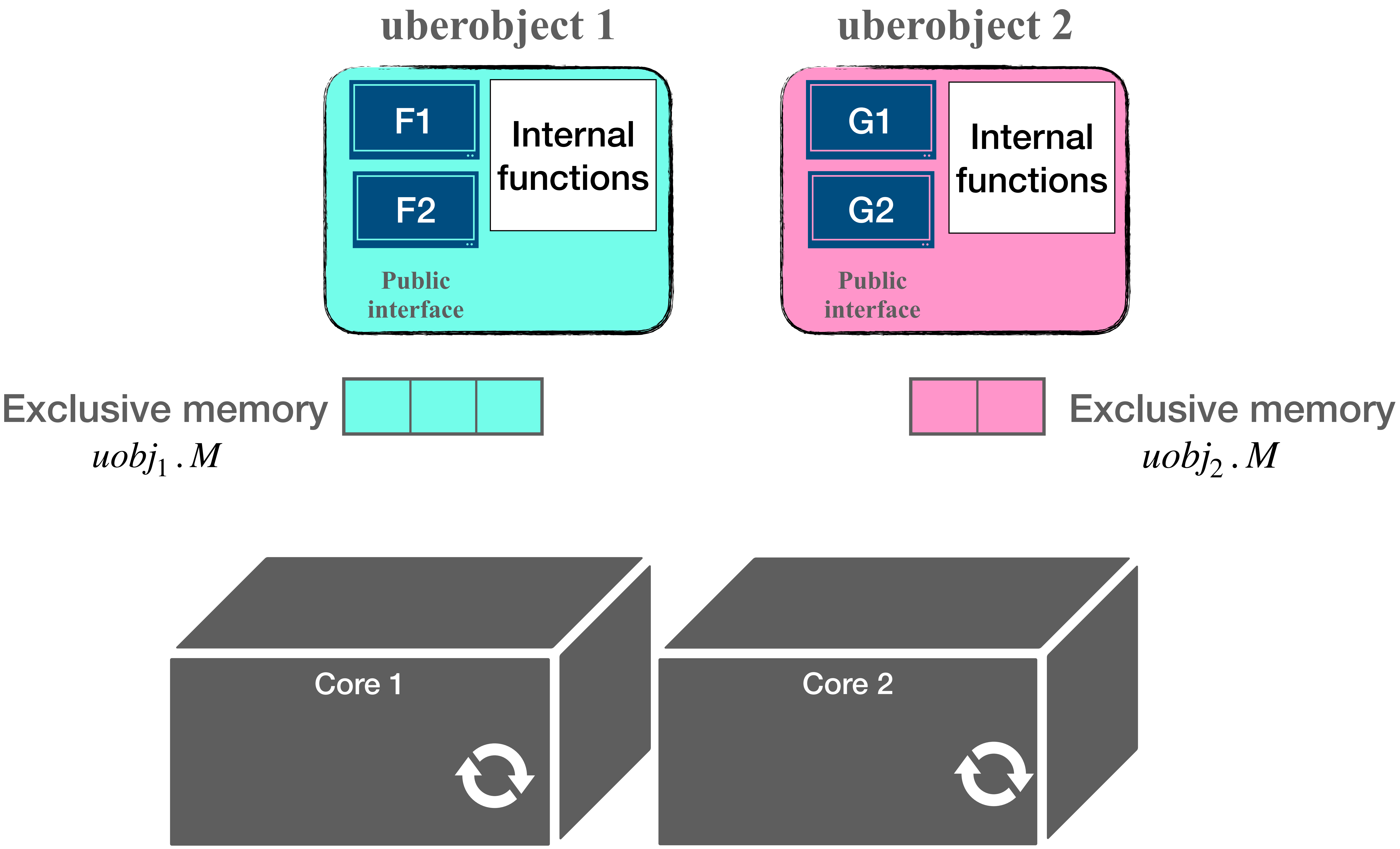
Internal functions

Public interface

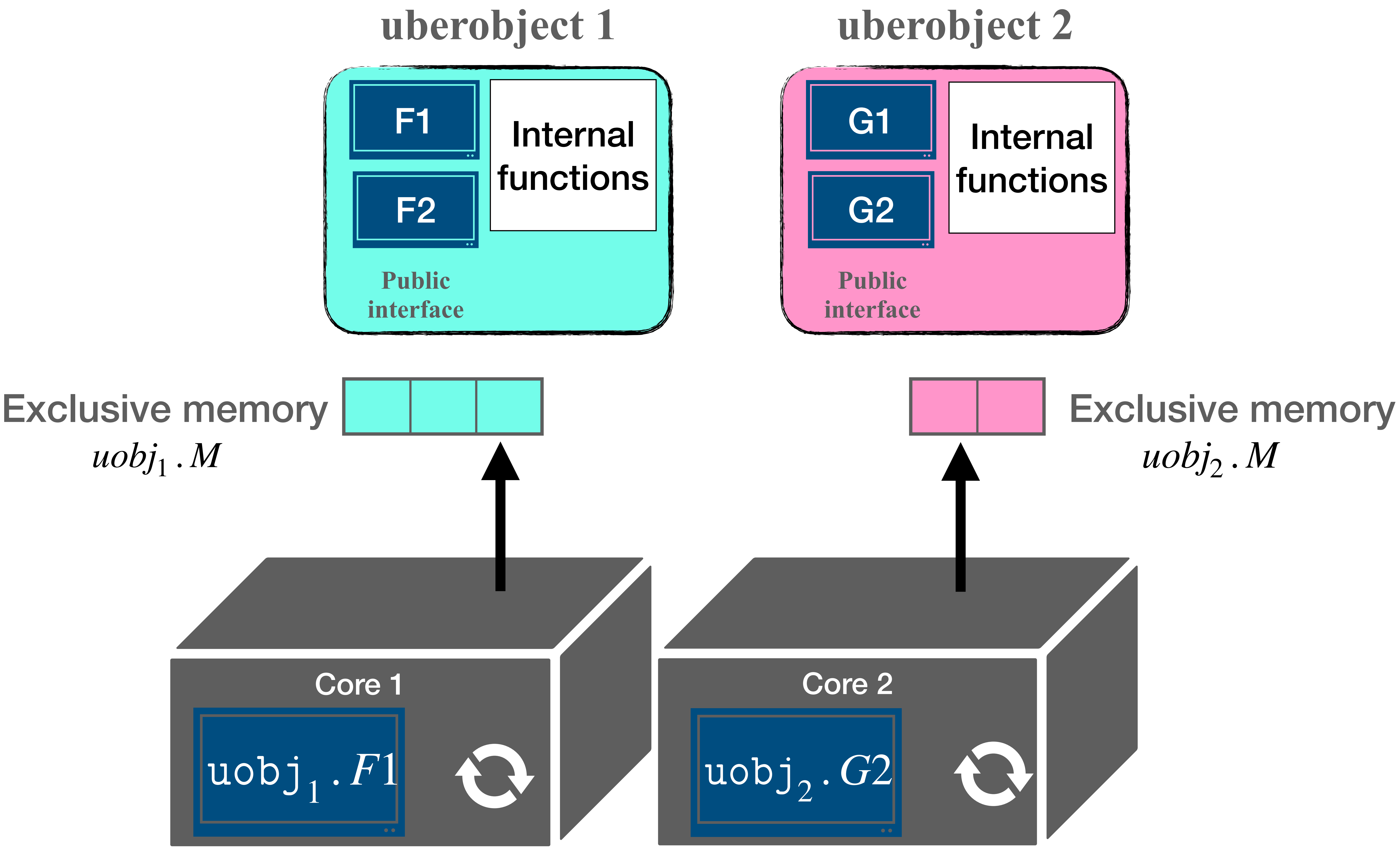
Exclusive memory

$uobj_2 . M$

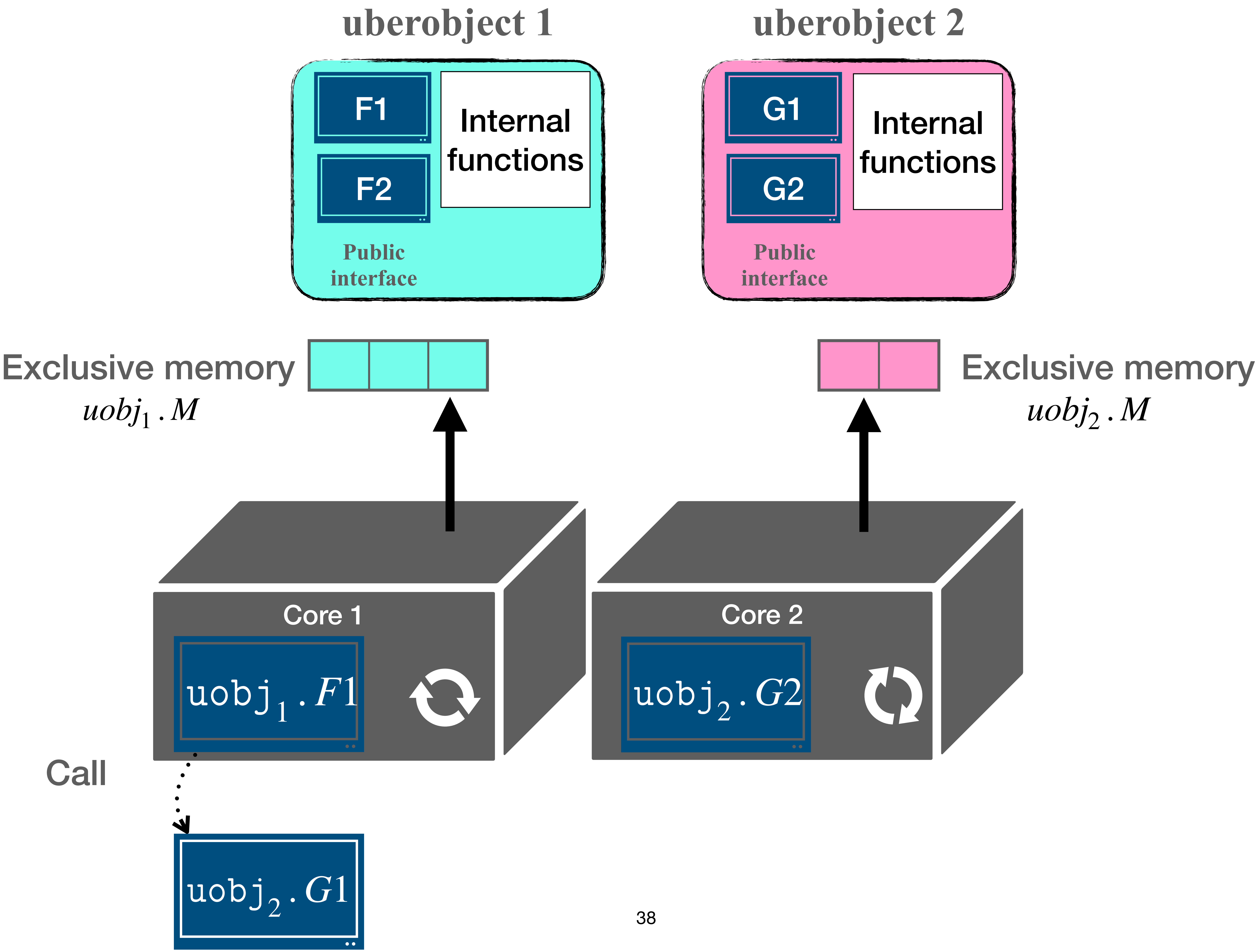
Concurrent execution



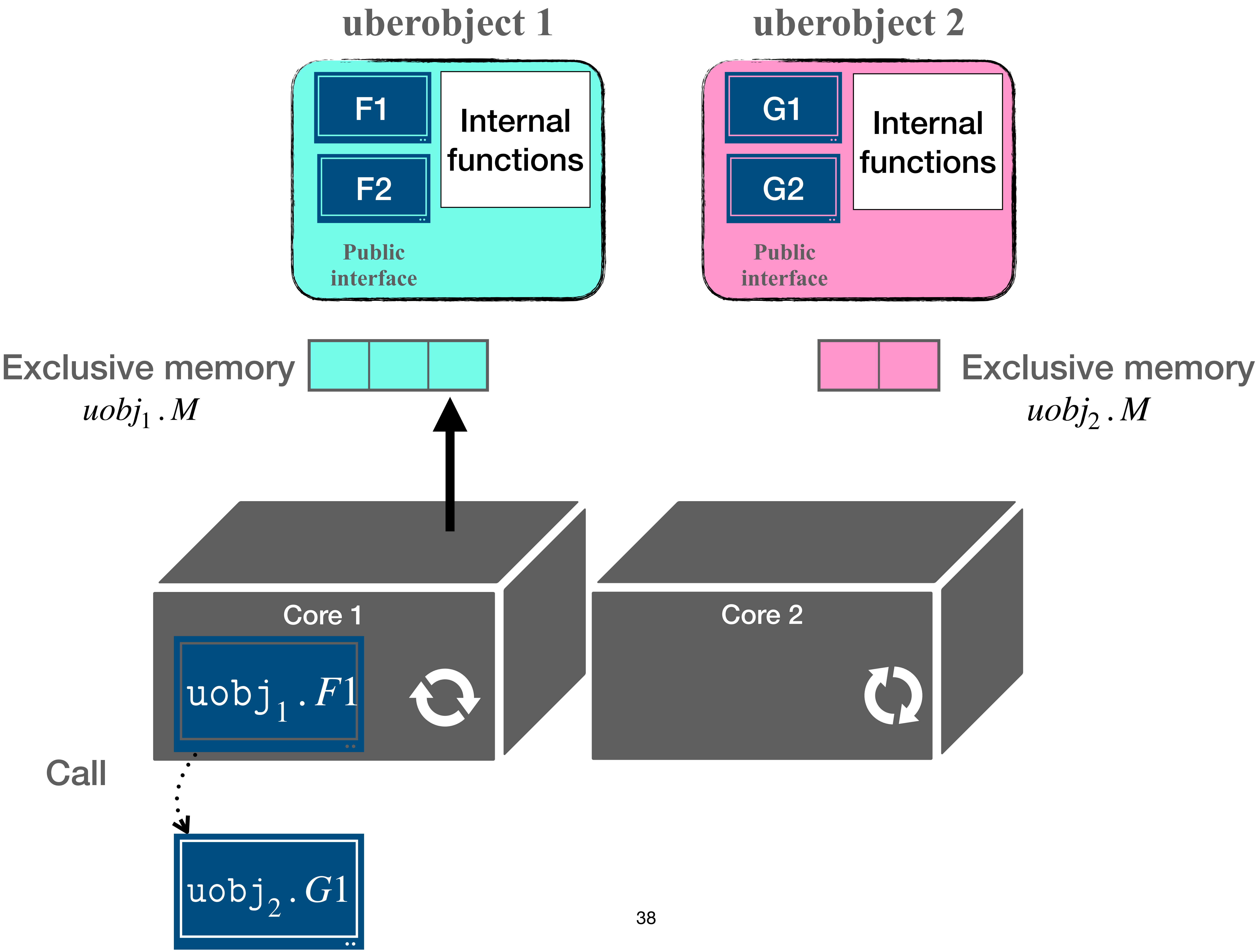
Concurrent execution



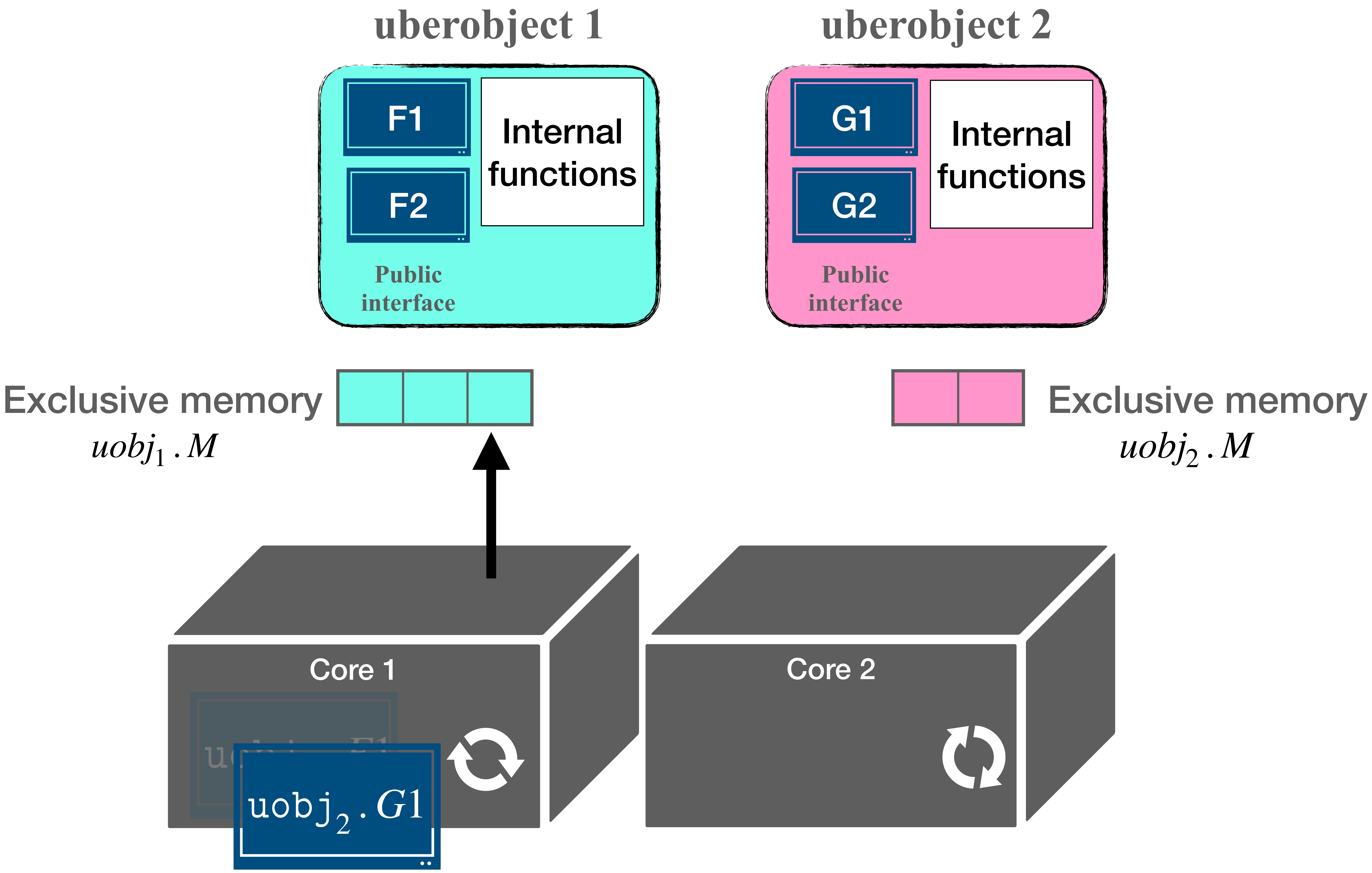
Concurrent execution



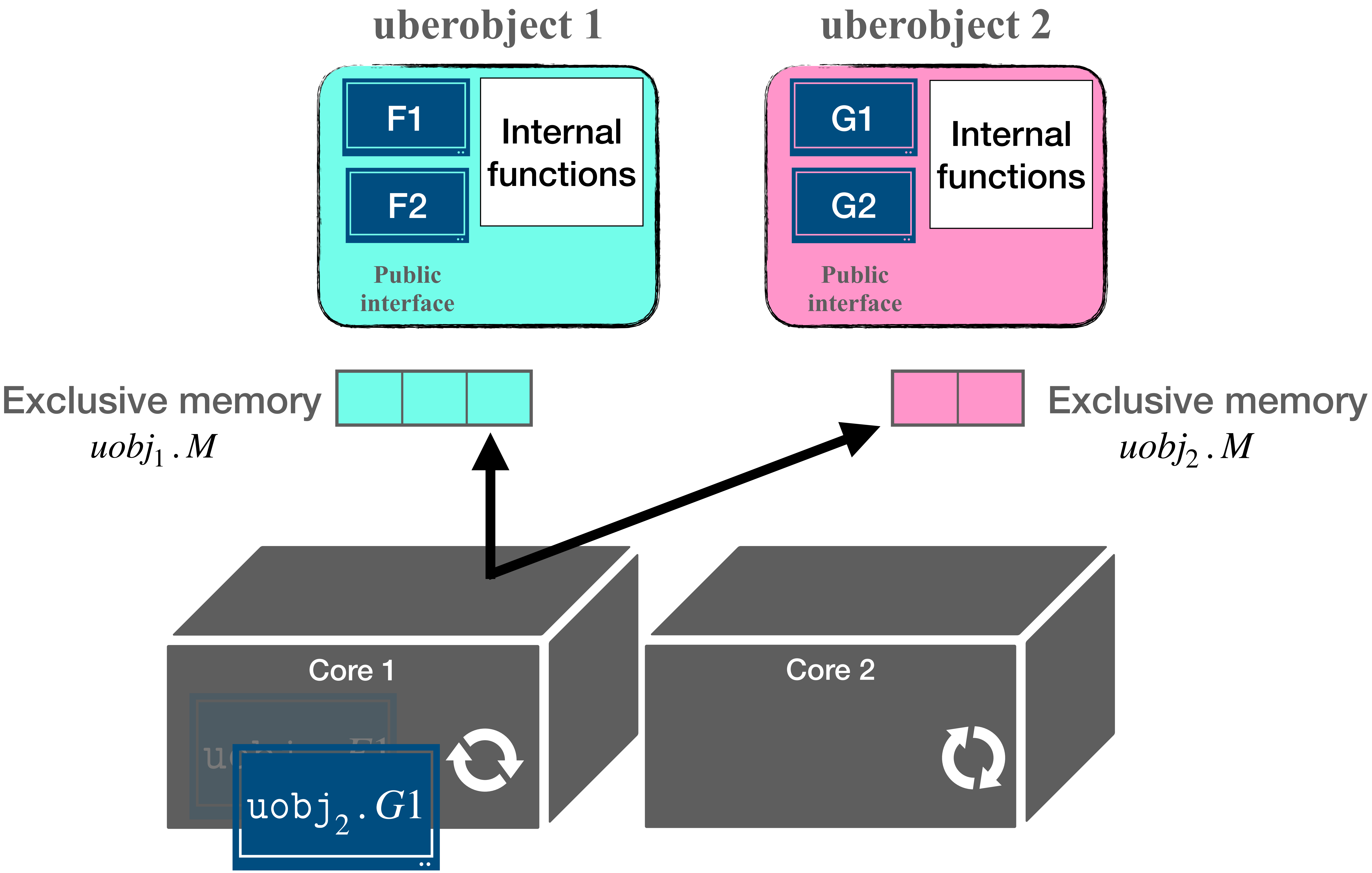
Concurrent execution



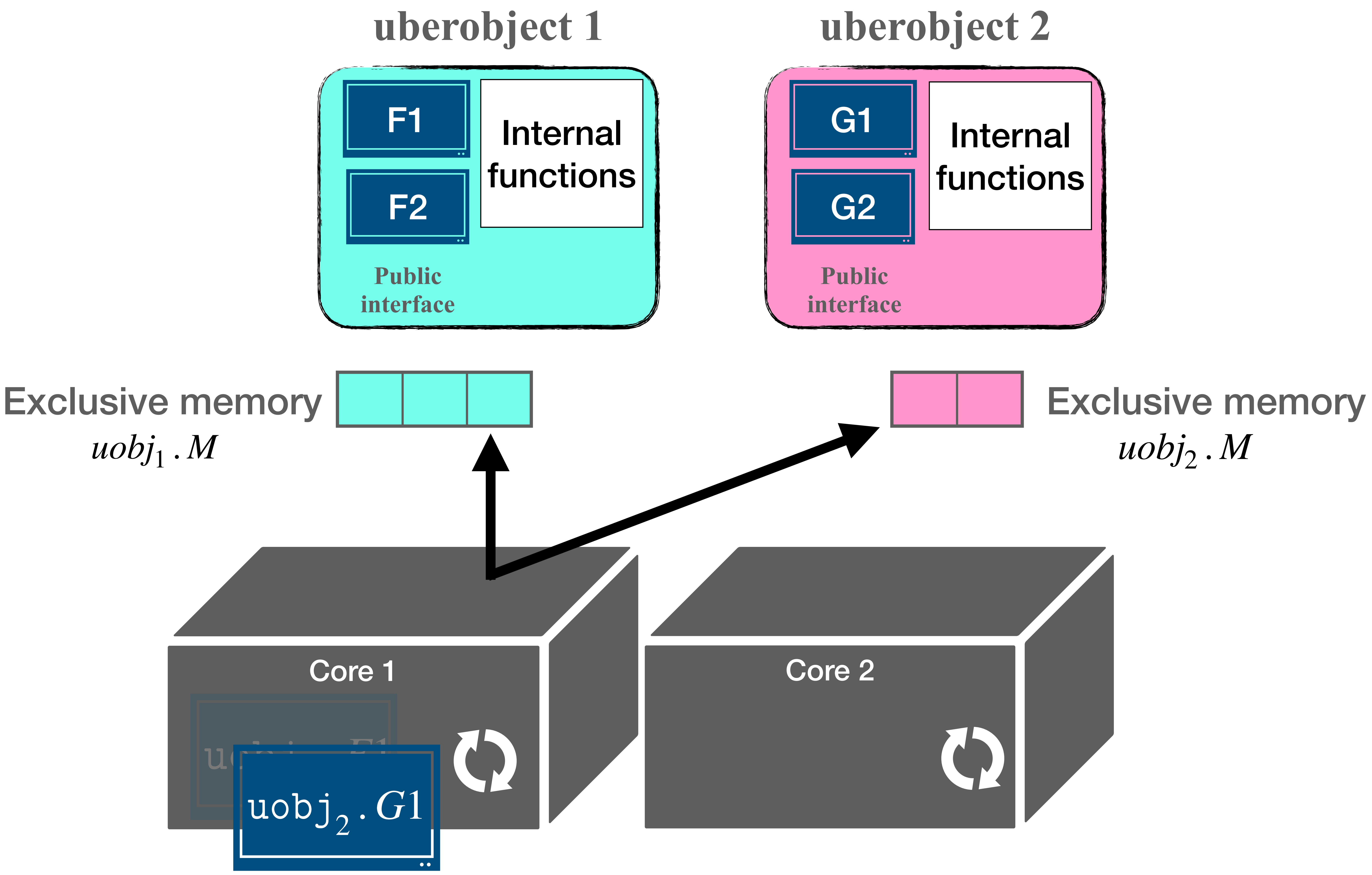
Concurrent execution



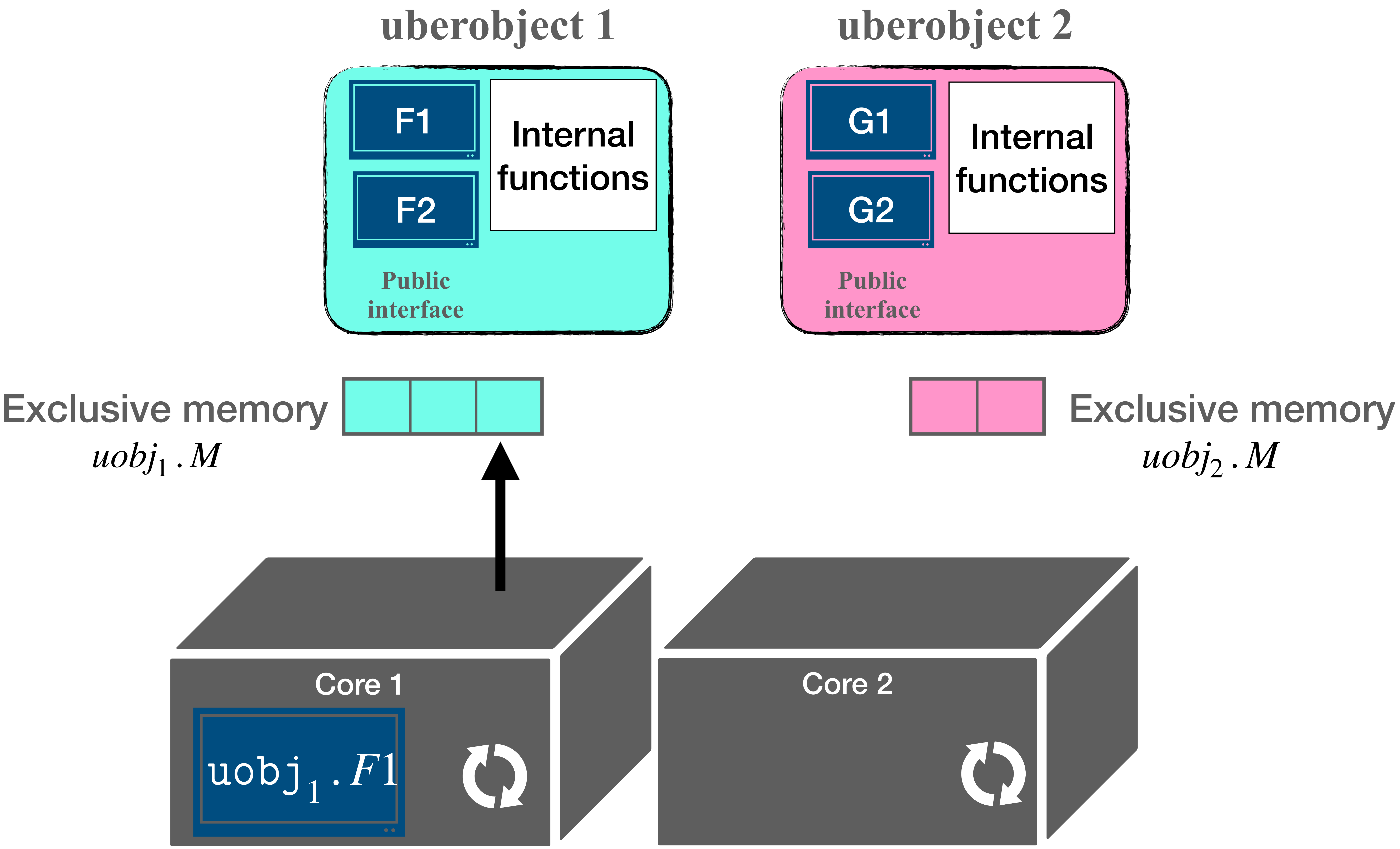
Concurrent execution



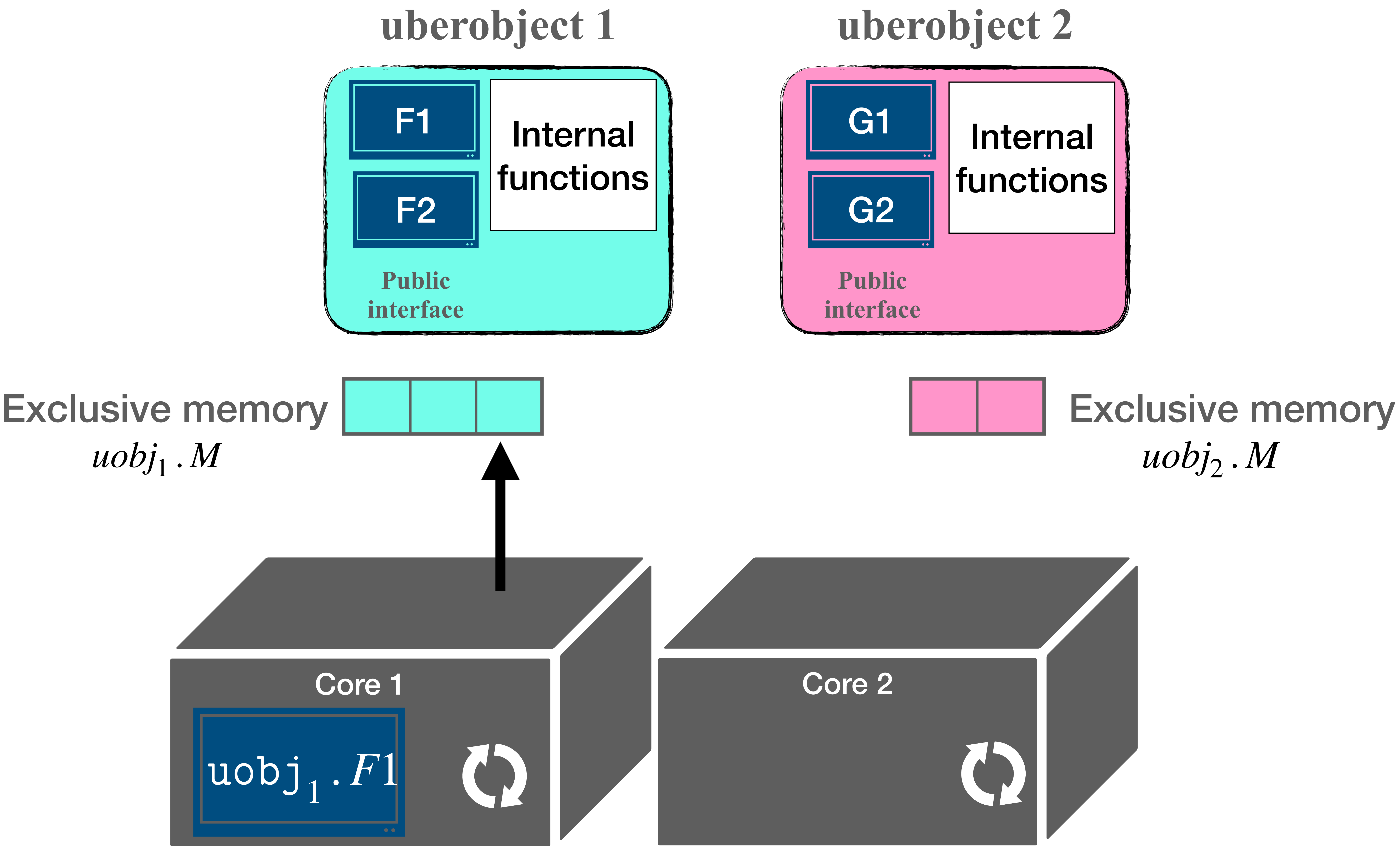
Concurrent execution



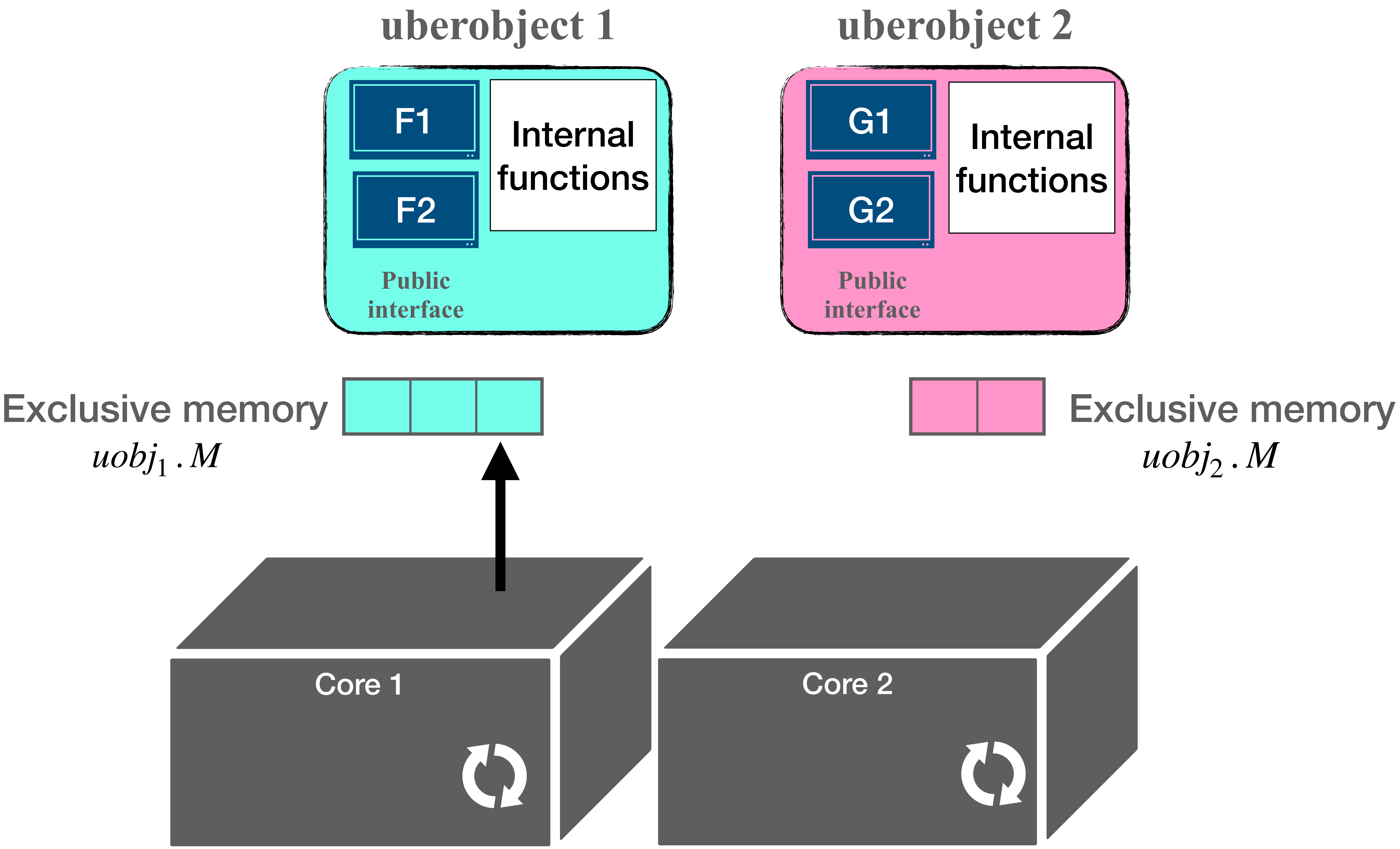
Concurrent execution



Concurrent execution



Concurrent execution



Source-level guarantees via verification of each compartment

— Respecting the interface —

Requirement from a source-level compartment—Respecting the interface

$\texttt{uobj}_1.F1$

$\texttt{uobj}_2.G2$

Requirement from a source-level compartment—Respecting the interface

$\texttt{uobj}_1.F1$

$\texttt{uobj}_2.G2$

Requirement from a source-level compartment—Respecting the interface

Guarantee: Any internal step of *this uberobject* can only read from/write to its own exclusive memory.

$\texttt{uobj}_1.F1$

$\texttt{uobj}_2.G2$

Requirement from a source-level compartment—Respecting the interface

Guarantee: Any internal step of *this uberobject* can only read from/write to its own exclusive memory.

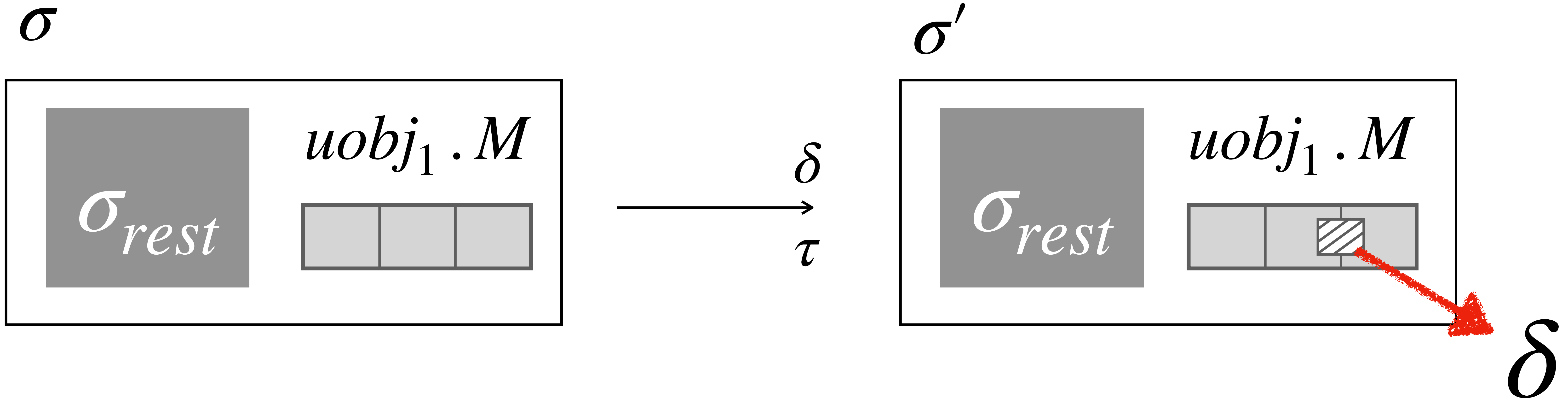
Rely: Any internal step of *other uberobjects* will never read from/write to this uberobject's exclusive memory.

$\texttt{uobj}_1.F1$

$\texttt{uobj}_2.G2$

Requirement from a source-level compartment—Respecting the interface

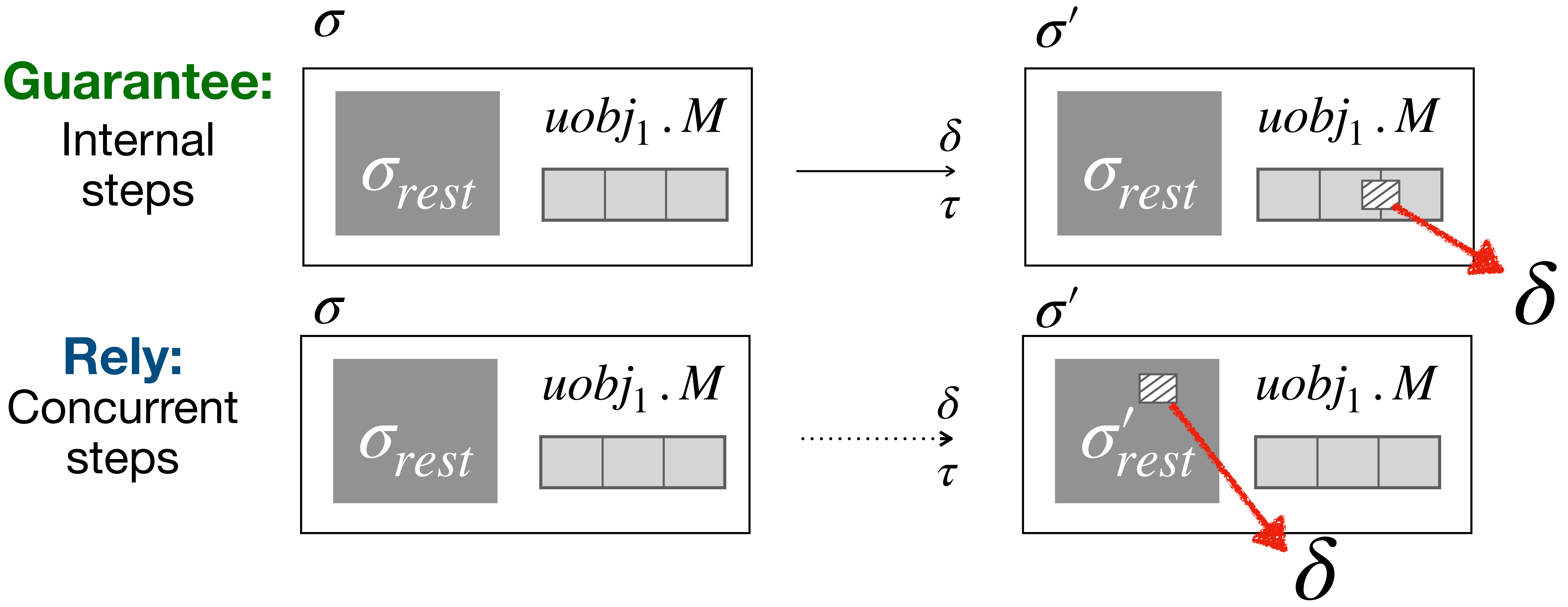
Guarantee:
Internal steps

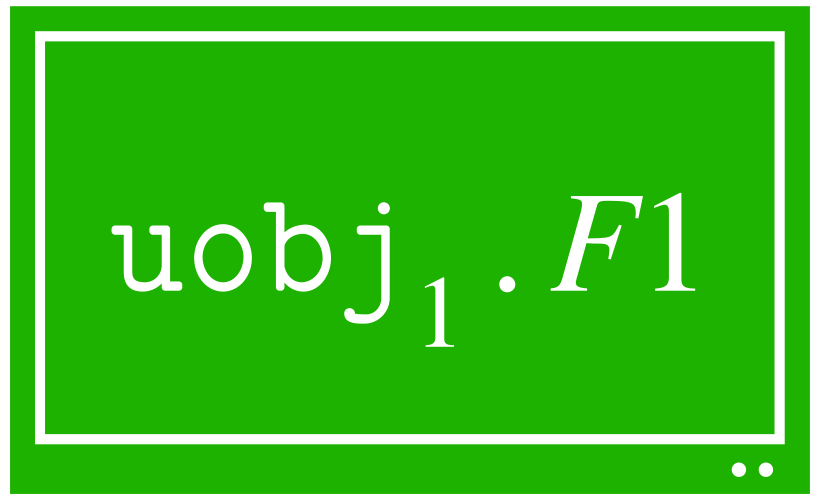


$\texttt{uobj}_1 . F1$

$\texttt{uobj}_2 . G2$

Requirement from a source-level compartment—Respecting the interface





Verifying Pre and Post conditions—Respecting the interface

Guarantee:

1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.

$\texttt{uobj}_1.F1$

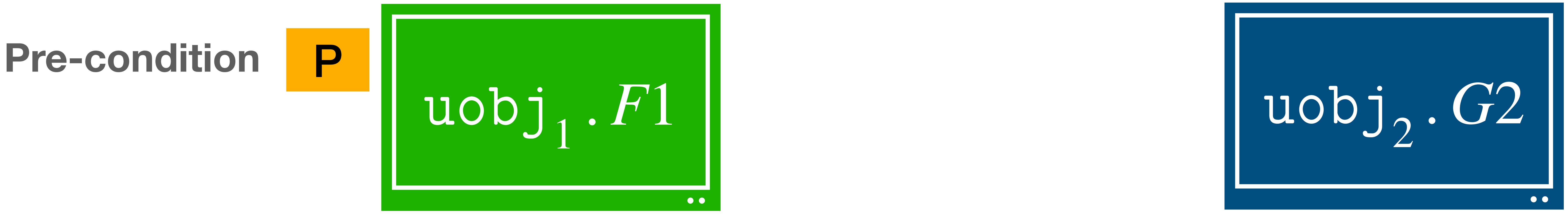
$\texttt{uobj}_2.G2$

Verifying Pre and Post conditions—Respecting the interface

Guarantee:

1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.

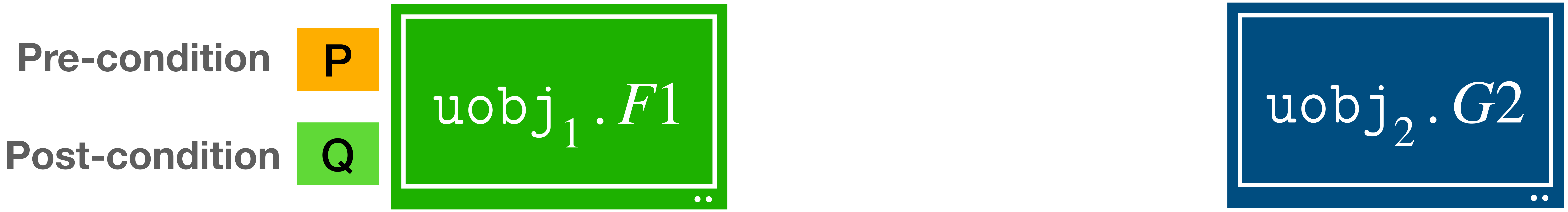


Verifying Pre and Post conditions—Respecting the interface

Guarantee:

1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.



Pre-condition P

Post-condition Q

$\texttt{uobj}_1.F1$

$\texttt{uobj}_2.G2$

Verifying Pre and Post conditions—Respecting the interface

Guarantee:

1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.

Pre-condition P

Post-condition Q

$\mathtt{uobj}_1.F1$

$\mathtt{uobj}_2.G2$

Pre-condition P’

Verifying Pre and Post conditions—Respecting the interface

Guarantee:

1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.

Pre-condition P

Post-condition Q

$\mathtt{uobj}_1.F1$

$\mathtt{uobj}_2.G2$

Pre-condition P’

Post-condition Q’

Verifying Pre and Post conditions—Respecting the interface

Rely:

1. If *other objects call* this uberobject's public interface, they will satisfy this uberobject's pre-condition.

2. When functions in *other uberobjects terminate*, their post-conditions hold.

$\texttt{uobj}_1.F1$

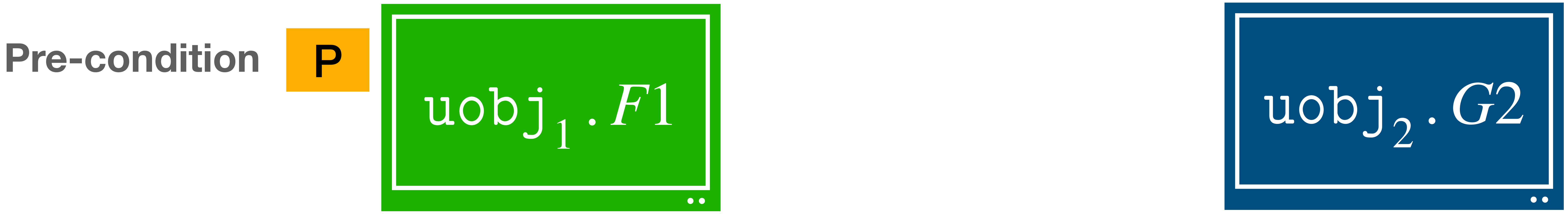
$\texttt{uobj}_2.G2$

Verifying Pre and Post conditions—Respecting the interface

Rely:

1. If *other objects call* this uberobject's public interface, they will satisfy this uberobject's pre-condition.

2. When functions in *other uberobjects terminate*, their post-conditions hold.



Pre-condition

P

$\texttt{uobj}_1.F1$

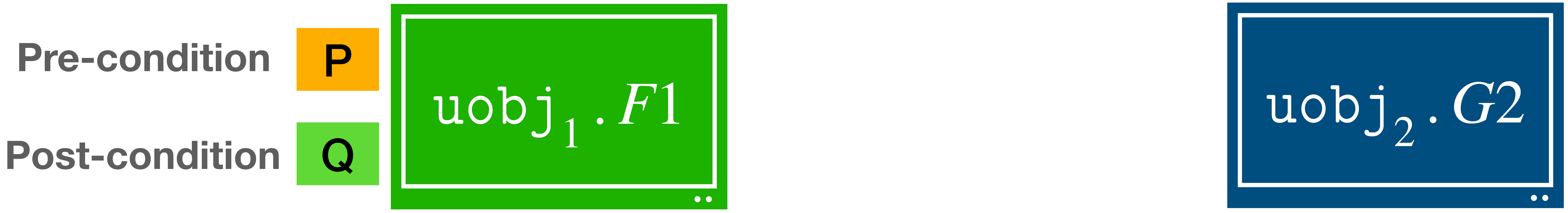
$\texttt{uobj}_2.G2$

Verifying Pre and Post conditions—Respecting the interface

Rely:

1. If *other objects call* this uberobject’s public interface, they will satisfy this uberobject’s pre-condition.

2. When functions in *other uberobjects terminate*, their post-conditions hold.



Pre-condition P

Post-condition Q

$\texttt{uobj}_1.F1$

$\texttt{uobj}_2.G2$

Verifying Pre and Post conditions—Respecting the interface

Rely:

1. If *other objects call* this uberobject’s public interface, they will satisfy this uberobject’s pre-condition.

2. When functions in *other uberobjects terminate*, their post-conditions hold.

Pre-condition P

Post-condition Q

$\texttt{uobj}_1.F1$

$\texttt{uobj}_2.G2$

Pre-condition P’

Verifying Pre and Post conditions—Respecting the interface

Rely:

1. If *other objects call* this uberobject's public interface, they will satisfy this uberobject's pre-condition.

2. When functions in *other uberobjects terminate*, their post-conditions hold.

Pre-condition P

Post-condition Q

$\texttt{uobj}_1.F1$

$\texttt{uobj}_2.G2$

Pre-condition P'

Post-condition Q'

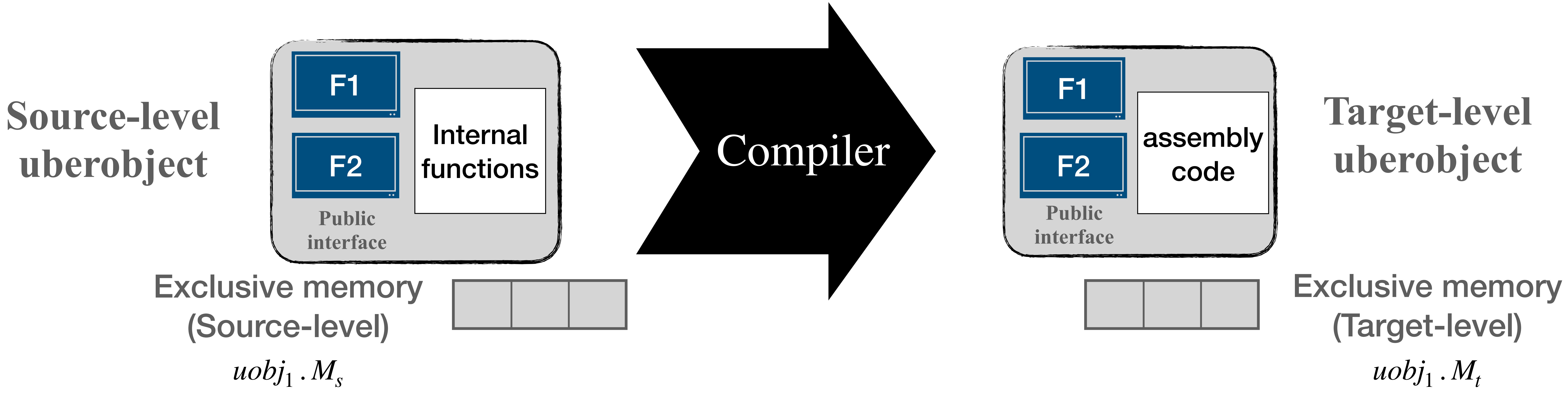
Verification result at the source-level:

If each uberobject in a system respects the interface, then:

- In any concurrent run, the **pre-conditions upon the call** and the **post-condition upon return** hold for all functions.
- Any concurrent execution is **data race free**, i.e., no two threads access a location concurrently when at least one of the accesses is a write.

Target-level guarantees via certified compilers — Preserving the interface —

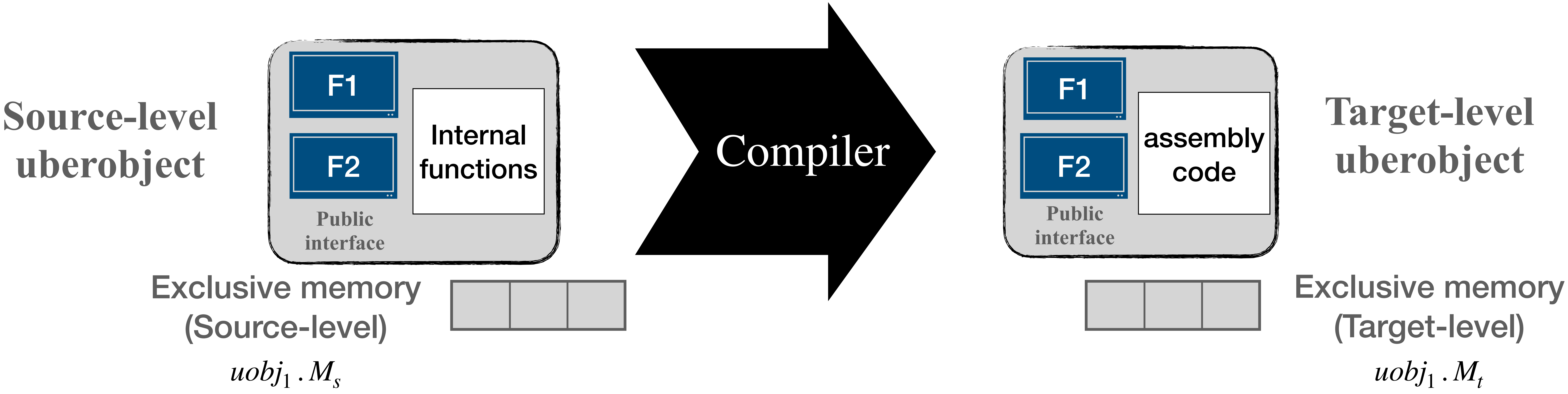
Requirement from a compiler—Preserving the interface



Requirement from a compiler—Preserving the interface

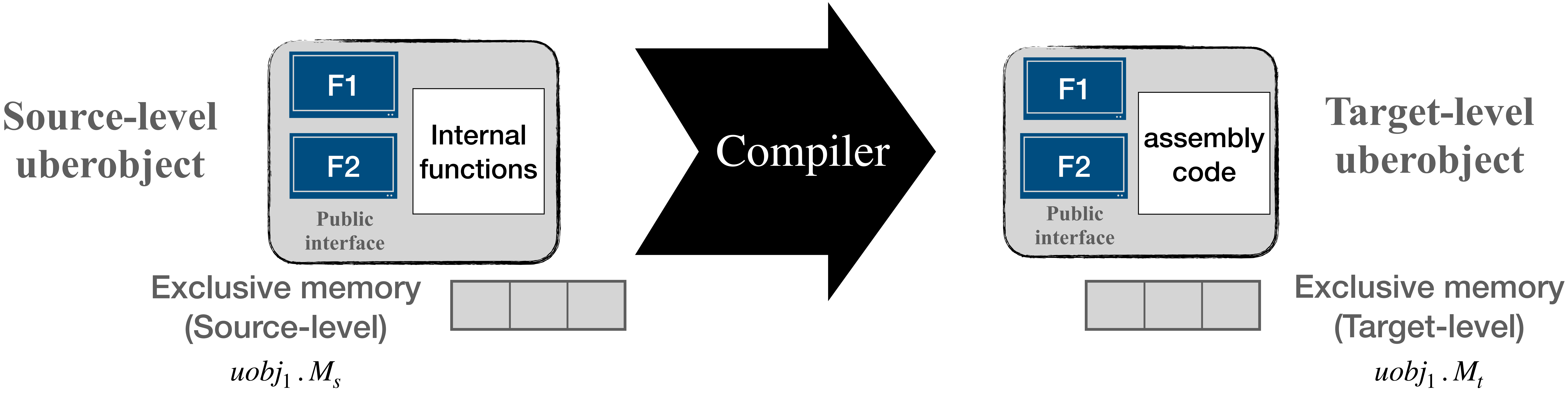
- Memory transformation function:

- Code transformation function:



Requirement from a compiler—Preserving the interface

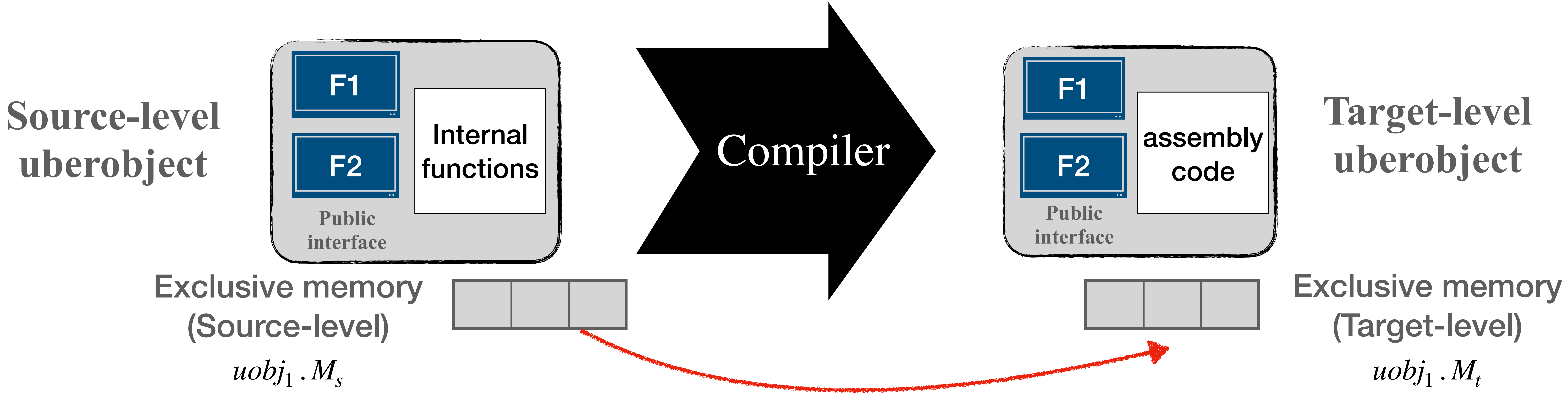
- Memory transformation function:
  - **Well-defined: Total and injective on heap locations, and map source-level heap locations to target-level heap locations.**
- Code transformation function:



Requirement from a compiler—Preserving the interface

- Memory transformation function:
  - **Well-defined: Total and injective on heap locations, and map source-level heap locations to target-level heap locations.**

- Code transformation function:



Source-level uberobject

F1

F2

Internal functions

Public interface

Exclusive memory (Source-level)

$uobj_1 . M_s$

Compiler

F1

F2

assembly code

Public interface

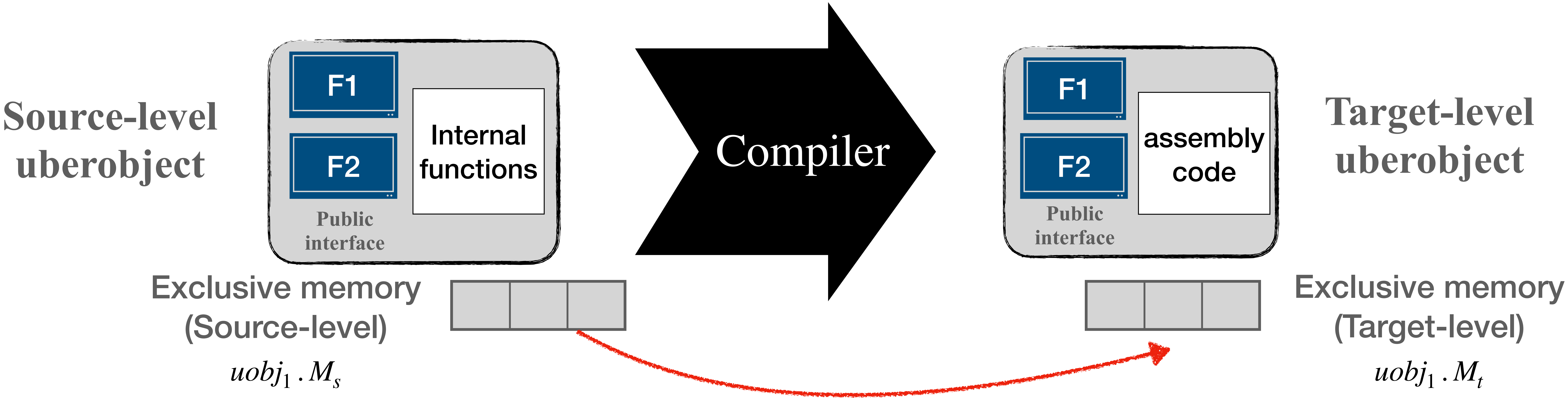
Target-level uberobject

Exclusive memory (Target-level)

$uobj_1 . M_t$

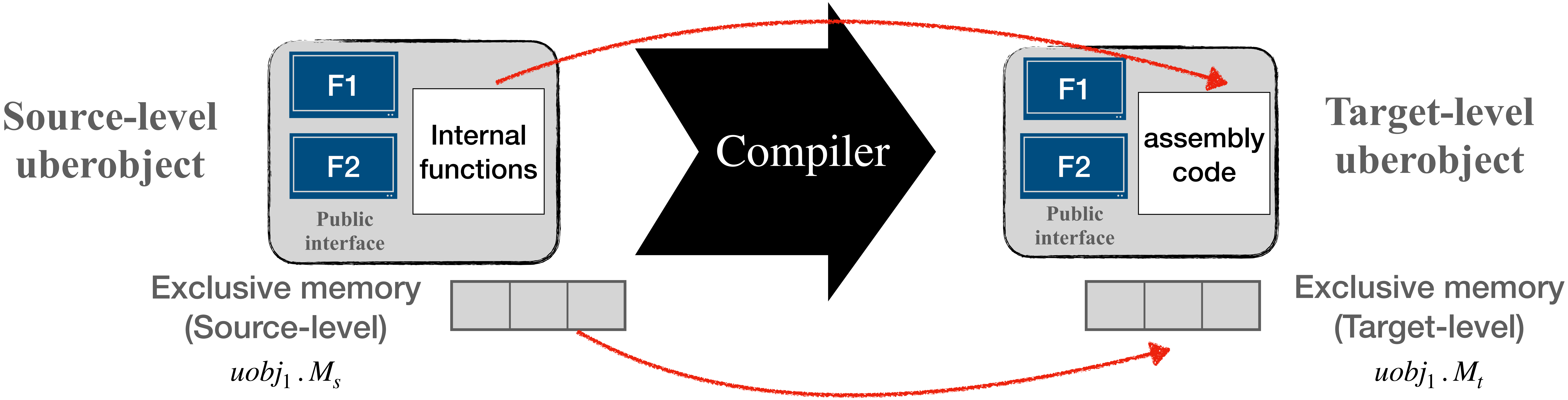
Requirement from a compiler—Preserving the interface

- Memory transformation function:
    - **Well-defined: Total and injective on heap locations, and map source-level heap locations to target-level heap locations.**
- Code transformation function:
    - **Interface-preserving: If an uobj respects the interface at the source level, then its compiled version respects the interface at the target level.**



Requirement from a compiler—Preserving the interface

- Memory transformation function:
  - **Well-defined: Total and injective on heap locations, and map source-level heap locations to target-level heap locations.**
- Code transformation function:
  - **Interface-preserving: If an uobj respects the interface at the source level, then its compiled version respects the interface at the target level.**



Target-level guarantees via interface preserving compilers

If each source-level uberobject in a system respects the interface and all compilers are interface-preserving, then

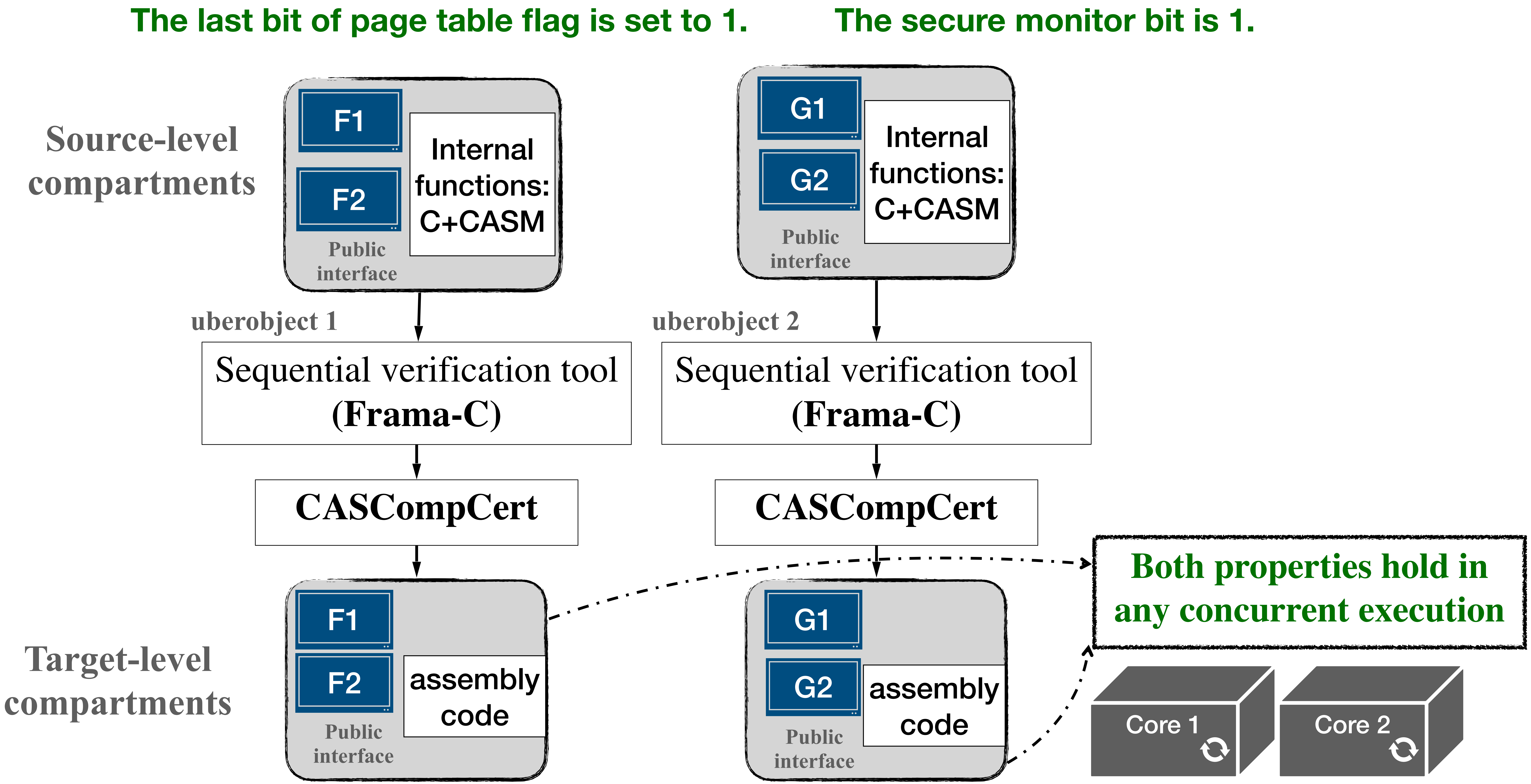
In any concurrent run at the target-level, the security properties hold:

All functions satisfy their post-conditions upon return.

CAS-Compcert is an interface-preserving compiler.

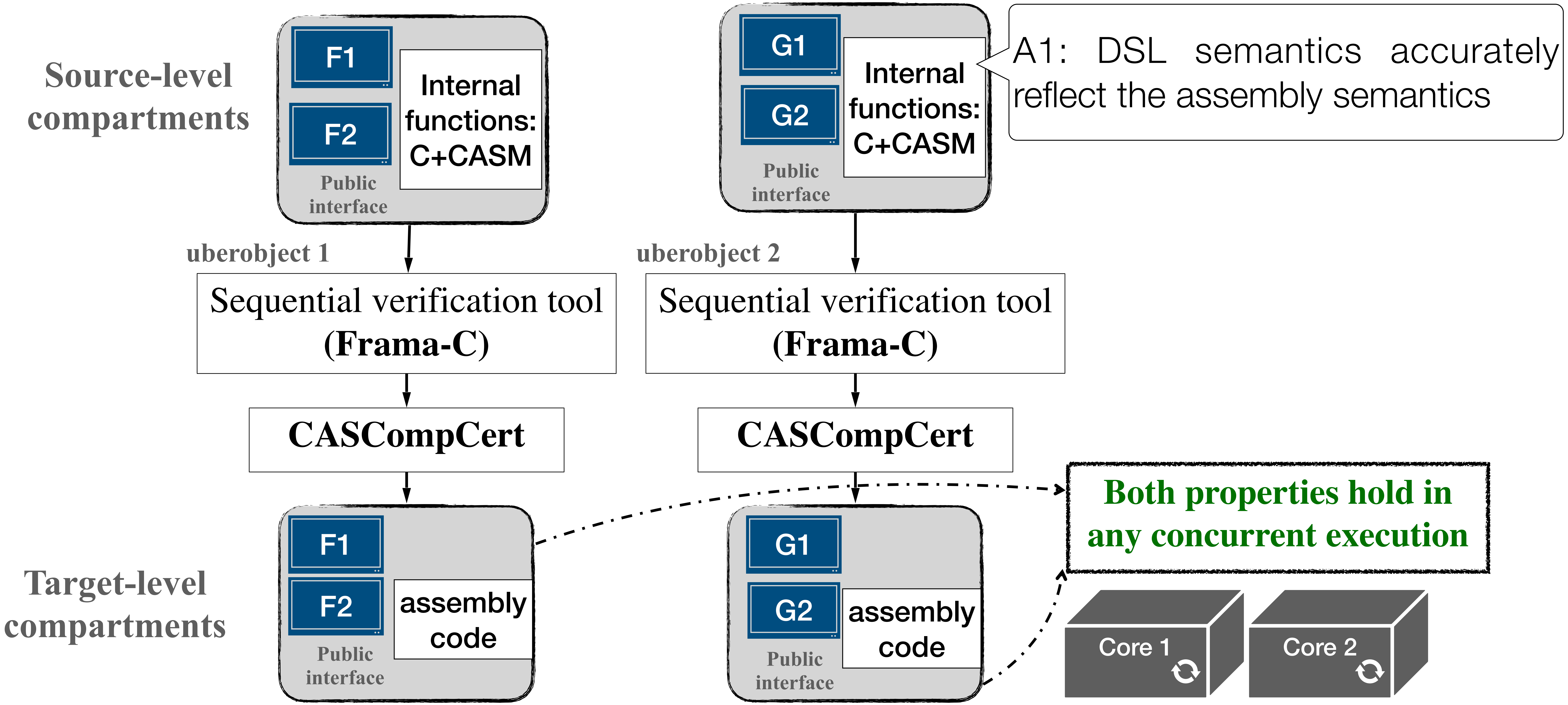
(PLDI'2019)

Our proposed tool-chain and its assumptions


The last bit of page table flag is set to 1.
The secure monitor bit is 1.
Source-level compartments
F1
F2
Internal functions: C+CASM
Public interface
G1
G2
Internal functions: C+CASM
Public interface
uberobject 1
uberobject 2
Sequential verification tool (Frama-C)
Sequential verification tool (Frama-C)
CASCompCert
CASCompCert
Both properties hold in any concurrent execution
Target-level compartments
F1
F2
assembly code
Public interface
G1
G2
assembly code
Public interface
Core 1
Core 2

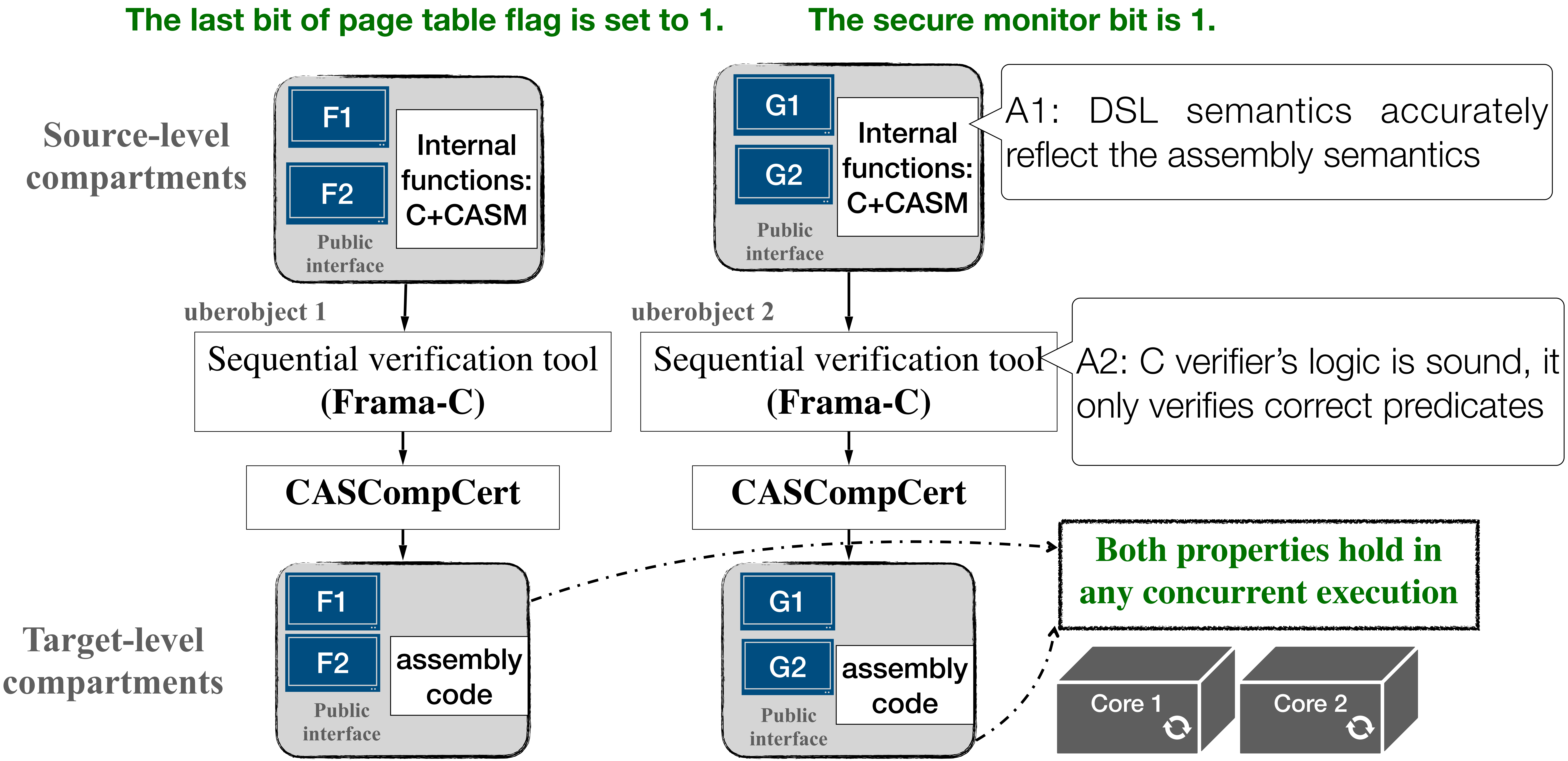
Our proposed tool-chain and its assumptions

The last bit of page table flag is set to 1. **The secure monitor bit is 1.**



Source-level compartments
F1
F2
Internal functions: C+CASM
Public interface
G1
G2
Internal functions: C+CASM
Public interface
A1: DSL semantics accurately reflect the assembly semantics
uberobject 1
uberobject 2
Sequential verification tool (Frama-C)
Sequential verification tool (Frama-C)
CASCompCert
CASCompCert
Both properties hold in any concurrent execution
Target-level compartments
F1
F2
assembly code
Public interface
G1
G2
assembly code
Public interface
Core 1
Core 2

Our proposed tool-chain and its assumptions



The last bit of page table flag is set to 1.

The secure monitor bit is 1.

A1: DSL semantics accurately reflect the assembly semantics

A2: C verifier's logic is sound, it only verifies correct predicates

Both properties hold in any concurrent execution

Source-level compartments
F1
F2
Internal functions: C+CASM
Public interface
G1
G2
Internal functions: C+CASM
Public interface
uberobject 1
uberobject 2
Sequential verification tool (Frama-C)
Sequential verification tool (Frama-C)
CASCompCert
CASCompCert
Target-level compartments
F1
F2
assembly code
Public interface
G1
G2
assembly code
Public interface
Core 1
Core 2

Our proposed tool-chain and its assumptions

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

Source-level compartments

F1

F2

Internal functions: C+CASM

Public interface

G1

G2

Internal functions: C+CASM

Public interface

A1: DSL semantics accurately reflect the assembly semantics

uberobject 1

Sequential verification tool (Frama-C)

uberobject 2

Sequential verification tool (Frama-C)

A2: C verifier's logic is sound, it only verifies correct predicates

A3: C semantics used by the C analysis tool and the CASCompCert compiler agree.

CASCompCert

CASCompCert

Both properties hold in any concurrent execution

Target-level compartments

F1

F2

assembly code

Public interface

G1

G2

assembly code

Public interface

Case studies

➡ UberXMHF TEE: Open source micrphypervisor TEE (x86 32-bit hardware)

- An execution environment for an untrusted OS
- Verify the security property of guest memory separation: page table permissions bit is set correctly.

➡ Trustzone TEE: A light-weight open-source Trustzone TEE (ARM 32-bit)

- An execution environment for a simple guest OS running at the highest privilege level
- Verify correct setup to get guest memory separation: the secure monitor mode is set correctly.

Related work

➡Verified TEEs

- Sel4 - S&P'2013
- CertiKOS - USENIX OSDI'2016
- XMHF - S&P '2013
- uberXMHF - USENIX Security '2016
- Security MIcrovisor - TDSCM '2019
- Contiki - DDECS '2015

➡Certified compilers:

- CASCompCert - PLDI'2019 , …

➡Compartmentalization:

- Secure Compartmentalizing compilation (SCC) - CSF'2016
- Robustly Safe Compartmentalizing Compilation (RSCC) - CCS'2018
- CHERI compartmentalization - SP '2015

Conclusion

➡ Summary:

- Compartmentalization for implementing TEEs enables us to:
    - achieve compositional verification results at the source level, and
    - leverage certified compilers to preserve the guarantees at the target level.
- Two case studies

➡ What else is in the paper?

- DSL semantics for assembly
- Interrupts
- Noninterference

Our proposed tool-chain and its assumptions

The last bit of page table flag is set to 1 (after function return)

The secure monitor bit is 1 (after function return)

Source-level compartments

F1

F2

Internal functions: C+CASM

Public interface

G1

G2

Internal functions: C+CASM

Public interface

A1: DSL semantics accurately reflect the assembly semantics

uberobject 1

Sequential verification tool **(Frama-C)**

uberobject 2

Sequential verification tool **(Frama-C)**

A2: C verifier's logic is sound, it only verifies correct predicates

A3: C semantics used by the C analysis tool and the CASCompCert compiler agree.

CASCompCert

CASCompCert

Both properties hold in any concurrent execution

Target-level compartments

F1

F2

assembly code

Public interface

G1

G2

assembly code

Public interface

Core 1

Core 2