Ciências ULisboa

Carnegie Mellon
SCHOOL OF COMPUTER SCIENCE

SECURE IDENTIFICATION of ACTIVELY EXECUTED CODE on a GENERIC TRUSTED COMPONENT

BRUNO VAVALA
CMU / UL

Nuno Neves
UL

Peter Steenkiste
CMU

IEEE/IFIP Conference on Dependable Systems & Networks (DSN'16)

Trusted Executions: trends & tradeoffs

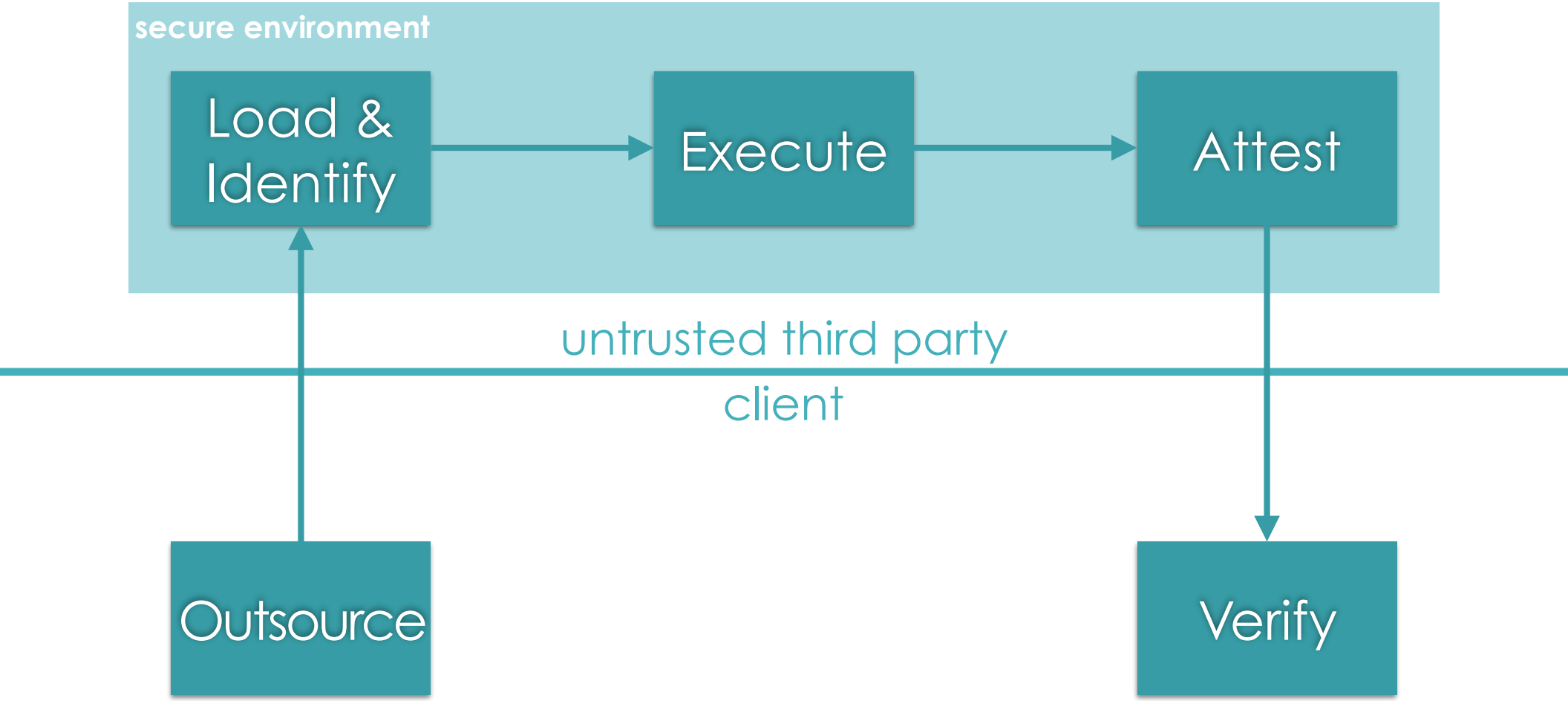
Anatomy of a Trusted Execution

untrusted third party

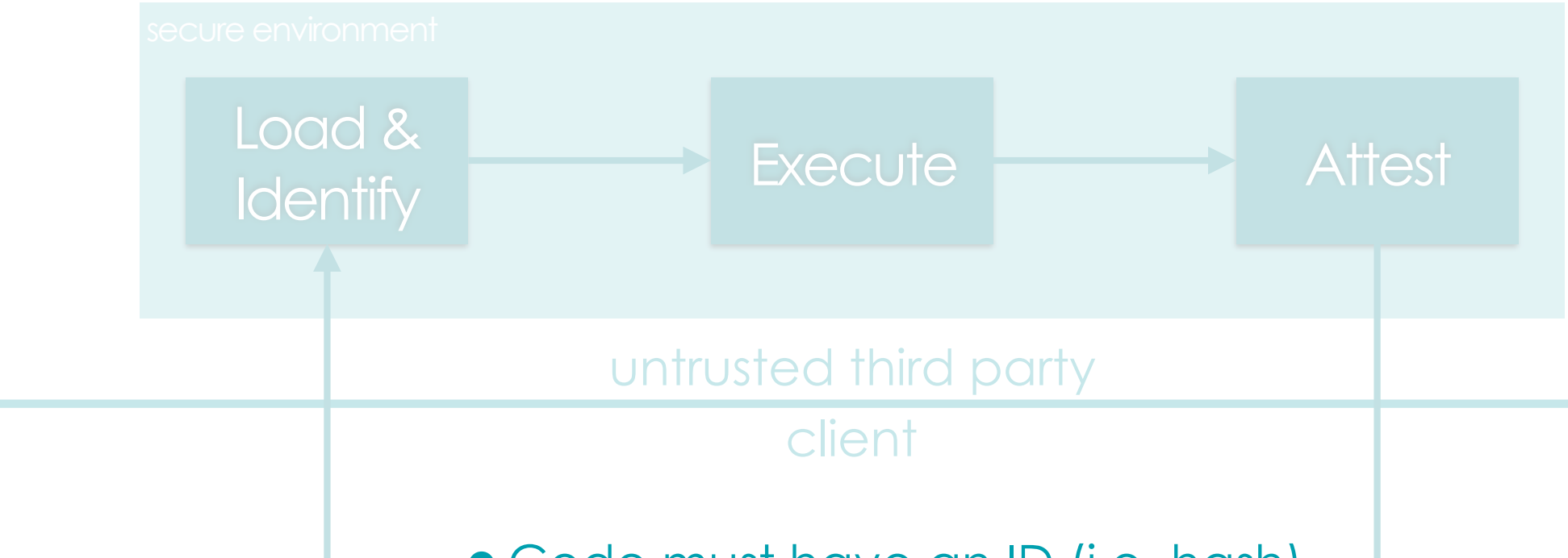
client

Execute

Anatomy of a Trusted Execution



Anatomy of a Trusted Execution



- Code must have an ID (i.e. hash)
- Trusted hardware computes ID
- Hardware attests the ID
- Client trusts hardware & ID

Generic TCC Interface

UTP-side

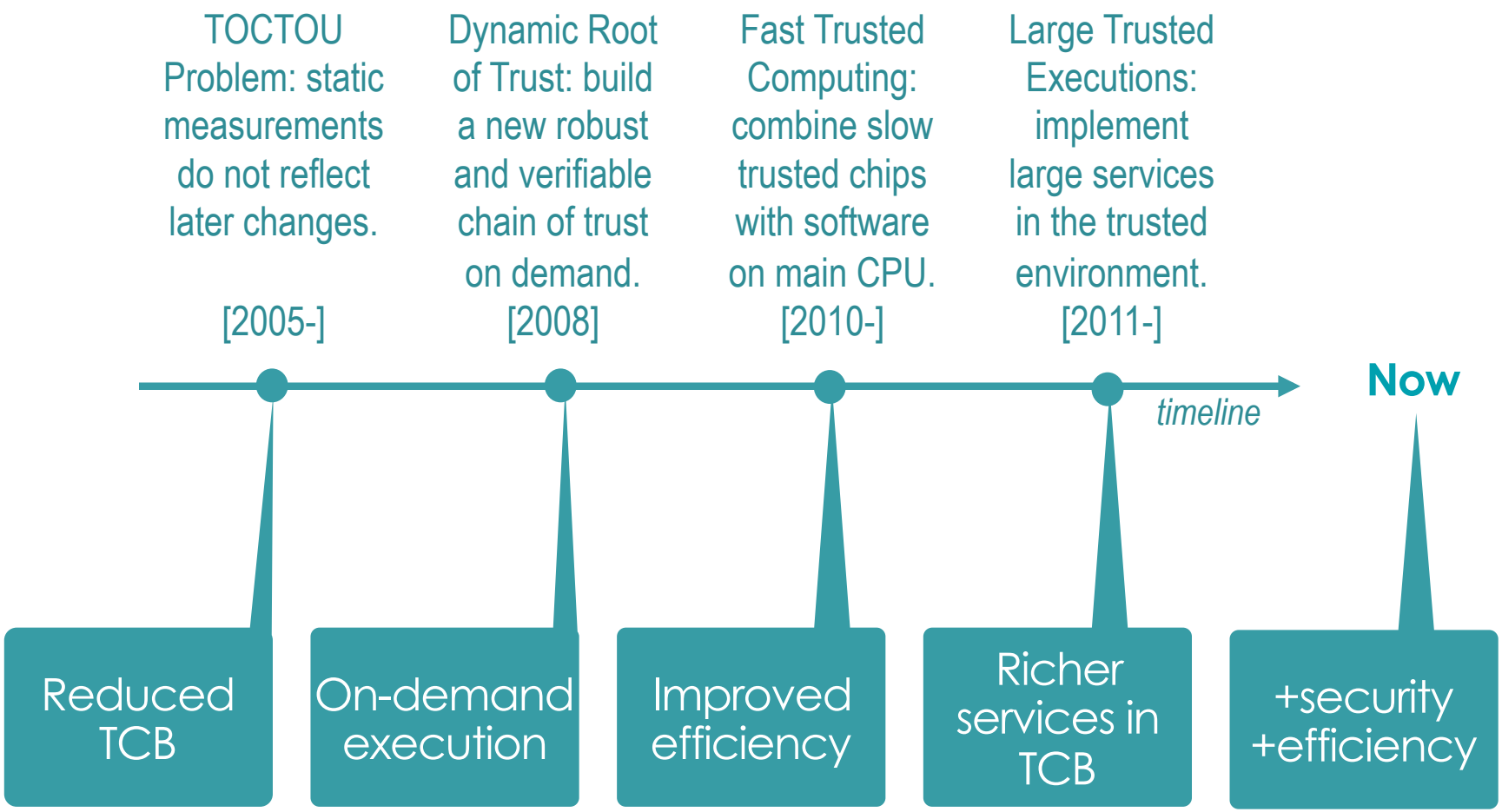
- **execute**
  - code loading + identification + isolated execution
- **attest**
  - TCC-signed code identity and I/O data

client-side

- **verify**

Implementable with:
- **Intel TXT + TPM**
- **Hypervisor-based TCC**
- **Intel SGX**
- **…**

Trends


TOCTOU Problem: static measurements do not reflect later changes.
[2005-]
Dynamic Root of Trust: build a new robust and verifiable chain of trust on demand.
[2008]
Fast Trusted Computing: combine slow trusted chips with software on main CPU.
[2010-]
Large Trusted Executions: implement large services in the trusted environment.
[2011-]
timeline
Now
Reduced TCB
On-demand execution
Improved efficiency
Richer services in TCB
+security +efficiency

Security/Efficiency Tradeoff for Large-Scale Services

Problem definition

Code Identity

```
/* SQLite code */
int main () {
 switch(op) {
  case SELECT:
   do_select();

  case DELETE:
   do_delete();

  case INSERT:
   do_insert();
  .
  .
  case FOOBAR:
  do_foobar();
}
```

source code

COMPILE

1	0	1	0	1	0	1	1	1	0	1	0	0	1	0	0	1	0	1	1
1	0	1	0	1	1	1	0	1	0	0	1	0	0	0	1	1	0	1	0
0	1	1	1	0	1	0	1	0	0	1	0	0	0	1	1	1	1	1	1
1	1	1	0	1	0	1	1	1	1	0	1	0	1	0	0	0	1	0	1
1	1	0	1	0	1	1	1	1	0	1	0	1	1	1	1	1	1	0	1
0	1	0	1	1	1	0	1	0	1	0	1	1	1	1	1	1	0	1	0
1	1	0	1	0	1	0	0	1	0	1	0	1	0	1	0	0	0	0	0
1	0	1	1	1	1	0	1	1	0	1	0	1	1	1	1	1	0	1	0
1	0	1	0	1	0	1	1	0	1	1	0	1	0	0	0	0	1	0	1
1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	0	1	0	1	0
1	1	1	1	0	1	0	0	1	0	1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	0	1	1	1	0	1	0	1	0	1	1	0	1	0	0
1	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0
0	1	0	1	1	1	1	1	0	1	1	0	0	0	1	1	1	0	1	1
1	1	0	1	1	1	1	0	1	0	1	0	0	1	0	1	0	1	0	0
1	0	1	0	0	1	0	1	0	1	1	0	1	1	0	1	1	1	0	0
1	0	1	0	1	0	0	1	1	0	1	0	1	1	0	1	0	0	1	1
1	1	1	1	0	1	0	0	0	1	1	0	0	1	0	0	1	0	1	0
1	1	0	0	1	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0
1	0	1	0	1	0	1	1	1	0	1	0	0	1	0	1	0	1	0	1
0	1	1	1	1	0	1	1	0	0	1	0	0	0	1	0	0	1	0	1
1	0	0	1	0	1	0	1	1	0	1	0	0	1	0	0	1	1	0	1
1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	0	1	0	1	0
1	1	0	1	0	1	1	0	1	0	0	1	0	0	0	1	1	0	0	1
1	0	1	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	1	0	1	1	1	0	1	0	0	1	0	0	1	1	1	0
1	1	1	0	1	1	1	0	1	0	0	1	0	0	1	1	1	0	0	1
1	1	0	1	0	0	1	0	0	1	0	0	0	1	1	1	0	1	1	0
1	0	1	0	1	1	0	1	0	0	1	0	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0	1	0	0	1	0	0	0	1	1	1

binary

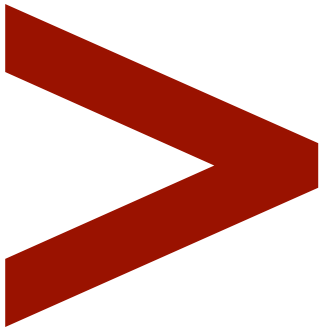
IDENTIFY

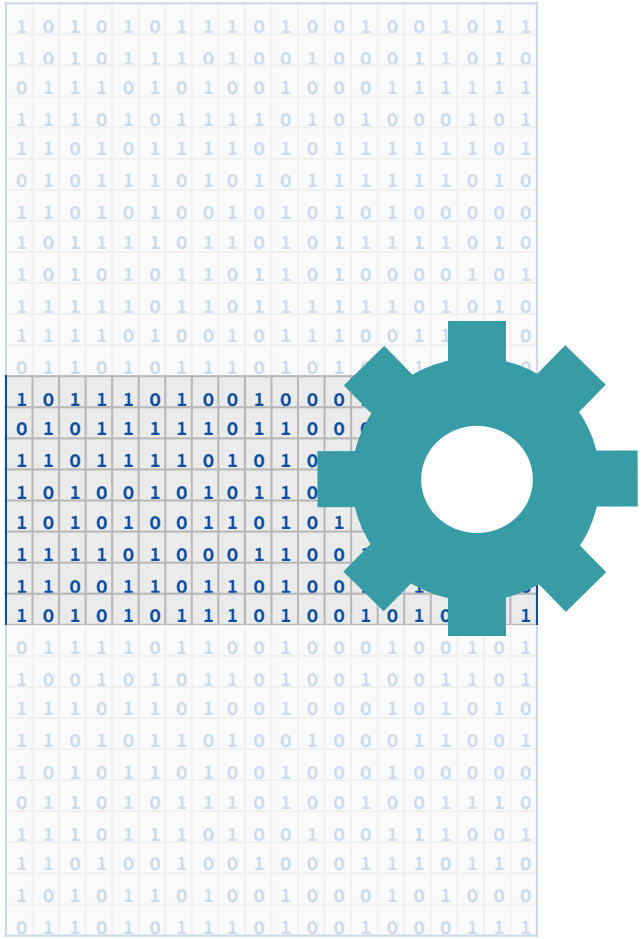
code identity

Execution Verification

Identified ≠ Executed

identified
binary code





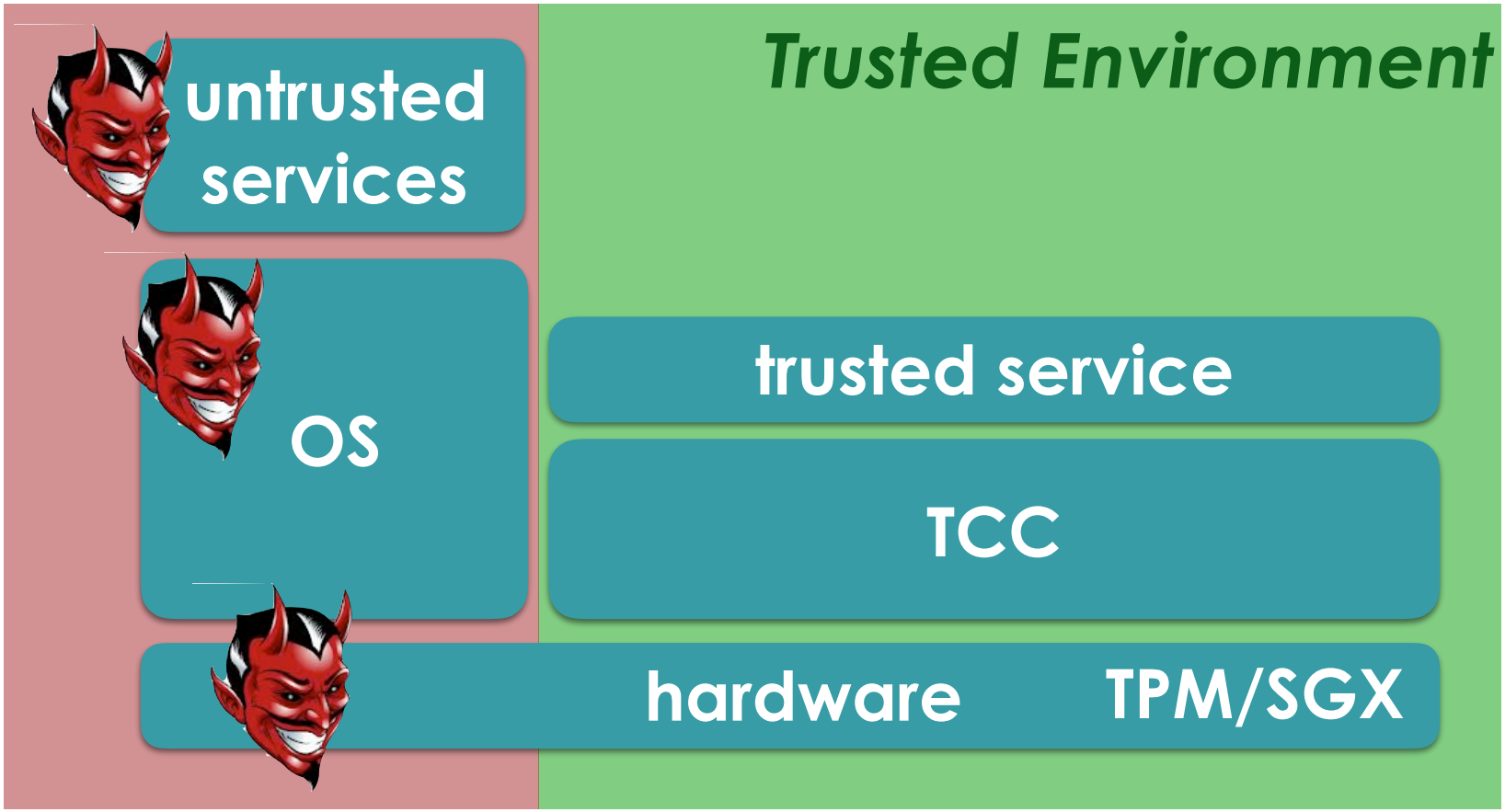
actually executed
binary code

Desirable Properties

- Identifying what is “actually” executed
- TCC agnostic execution
- Keeping efficient client-side verification

Identifying Actively Executed Code

Model



- OS and services are untrusted
- Client knows service identity and TCC certificate

Enriching the Interface

UTP-side

- **execute**
  - code loading + identification + isolated execution
- **attest**
  - TCC-signed code identity and I/O data
- **auth-put**
  - secure storage for a specific recipient (TCC authenticates the sender)
- **auth-get**
  - secure storage from a specific sender (TCC authenticates the recipient)

client-side

- **verify**

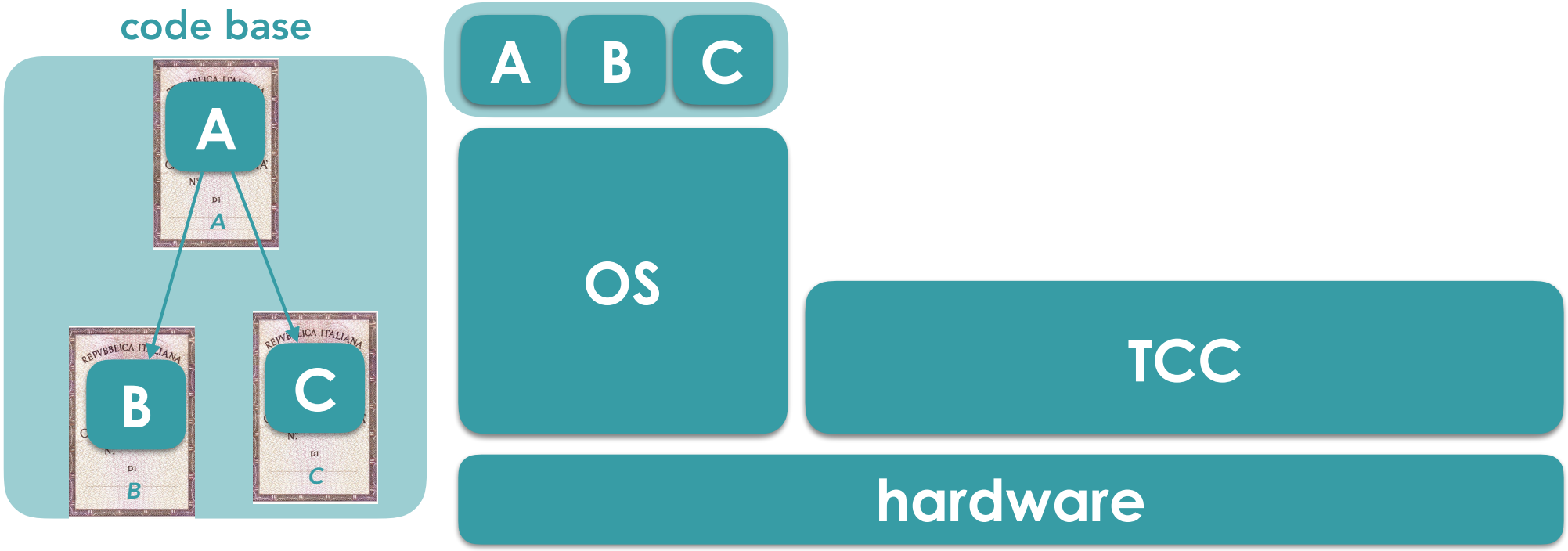
Implementable with:

- **Intel TXT + TPM**
- **Hypervisor-based TCC**
- **Intel SGX**
- **…**

One ID Per Code Module

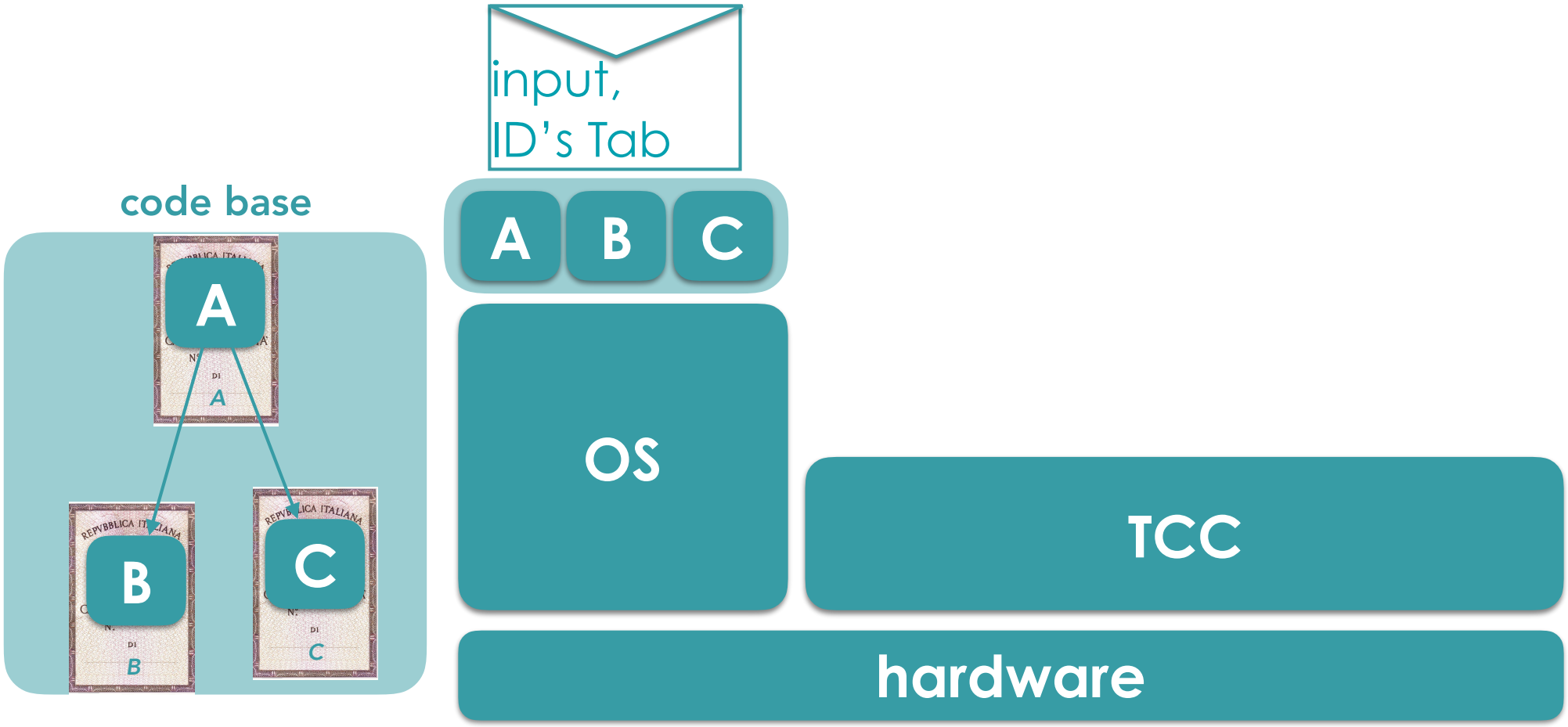
```
/* SQLite code */
int main () {
 switch(op) {
   case SELECT:
    do_select();

   case DELETE:
    do_delete();

   case INSERT:
    do_insert();
    .
    .
   case FOOBAR:
   do_foobar();
}
```

Execution Protocol

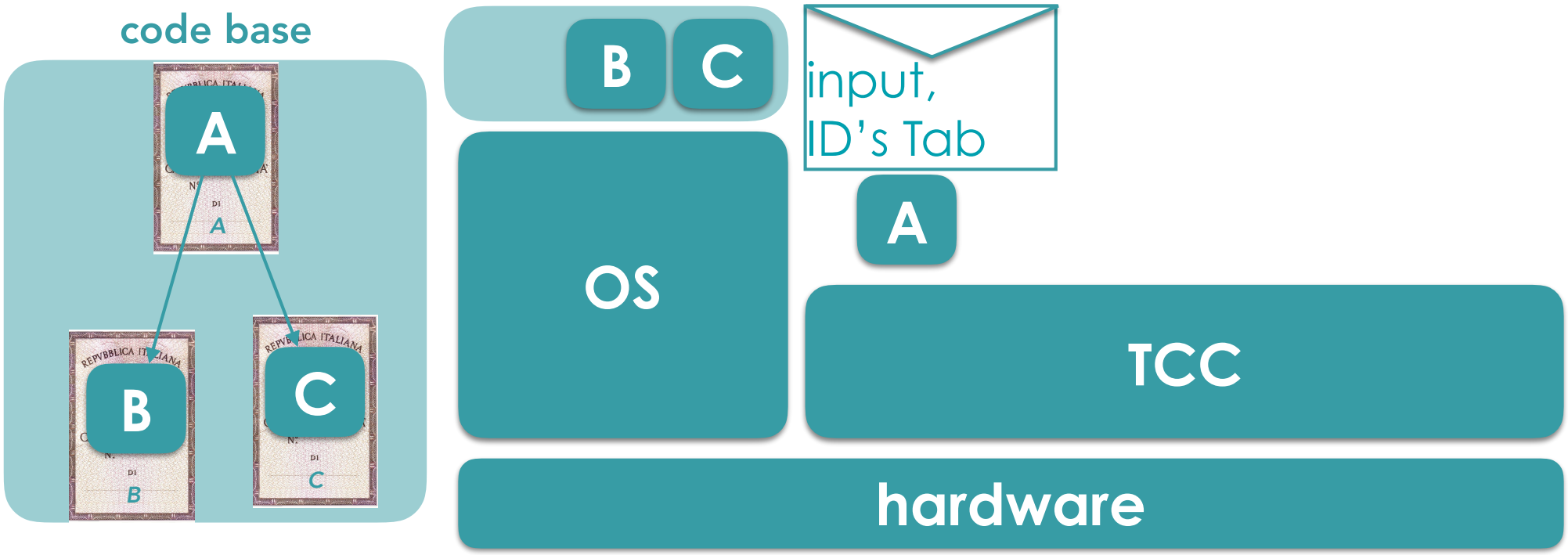


- Execution flows: A-to-B, A-to-C
- If C must be executed, then B is not loaded

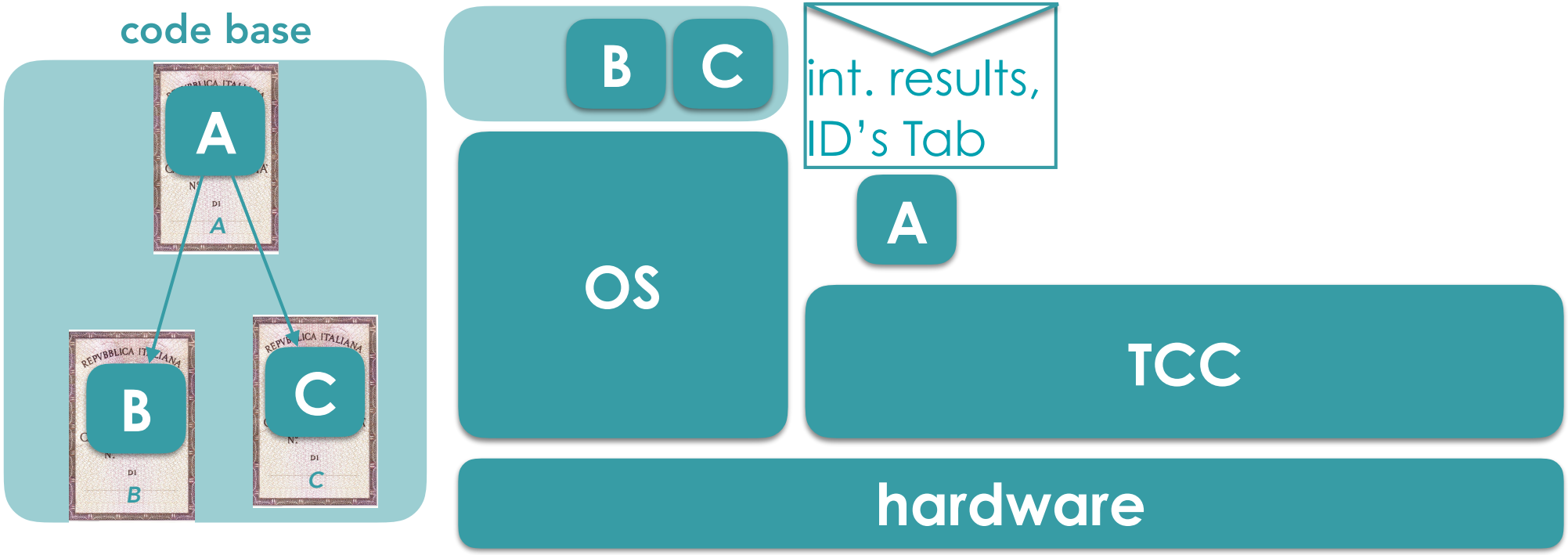
Execution Protocol



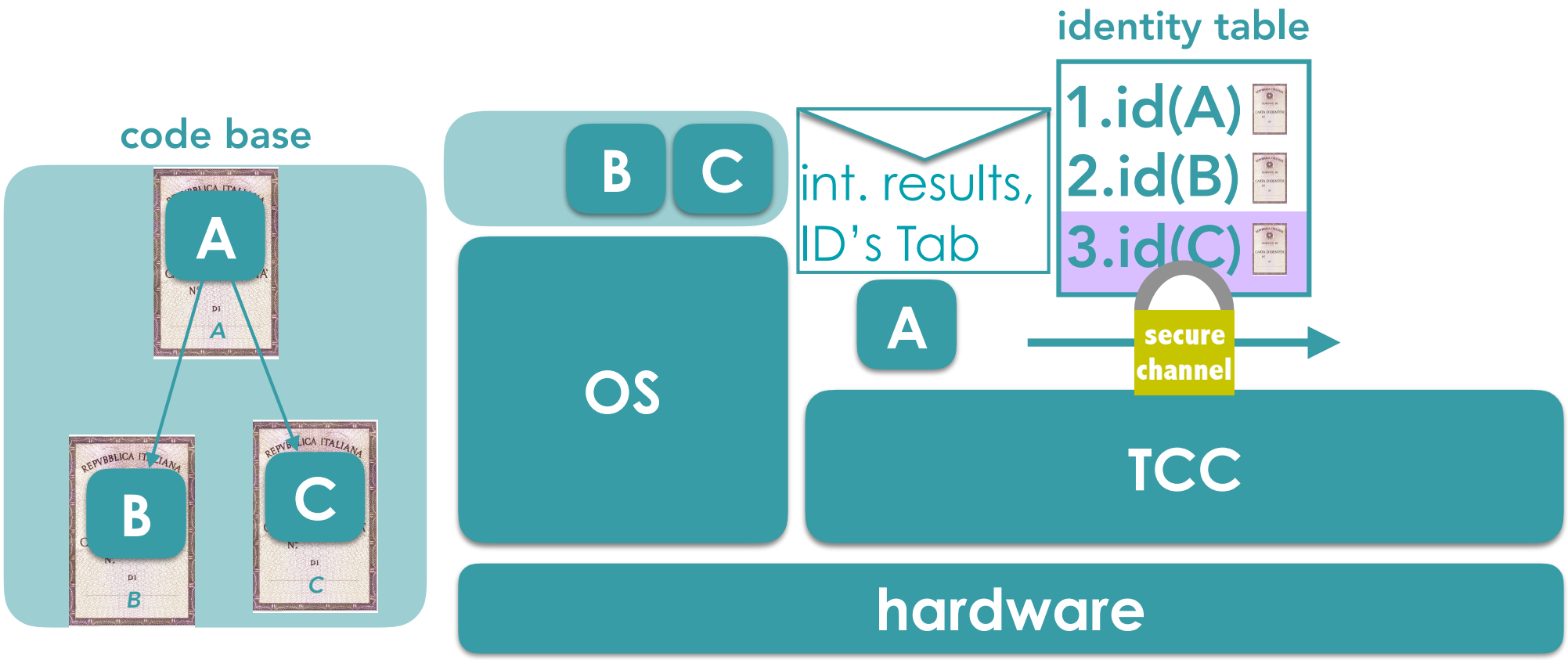
Execution Protocol



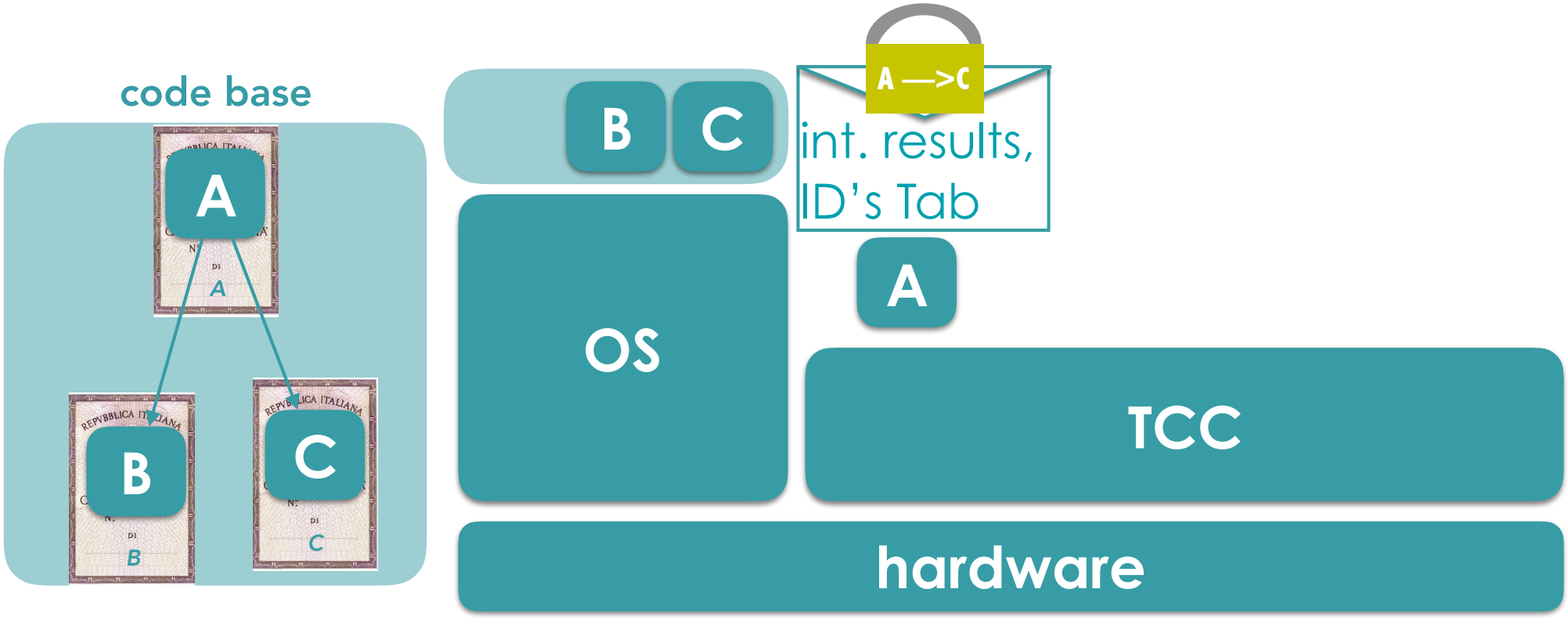
Execution Protocol



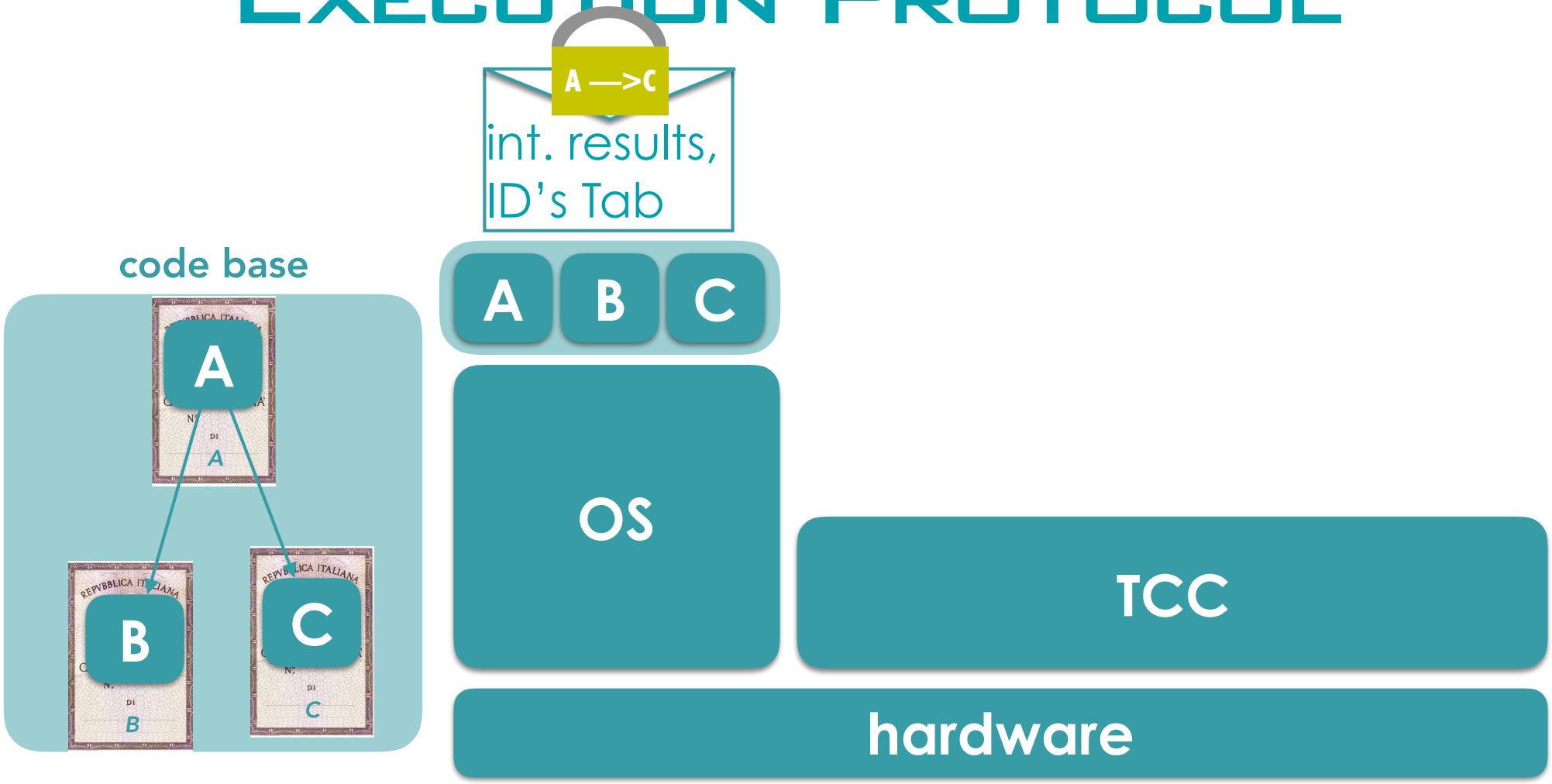
Execution Protocol



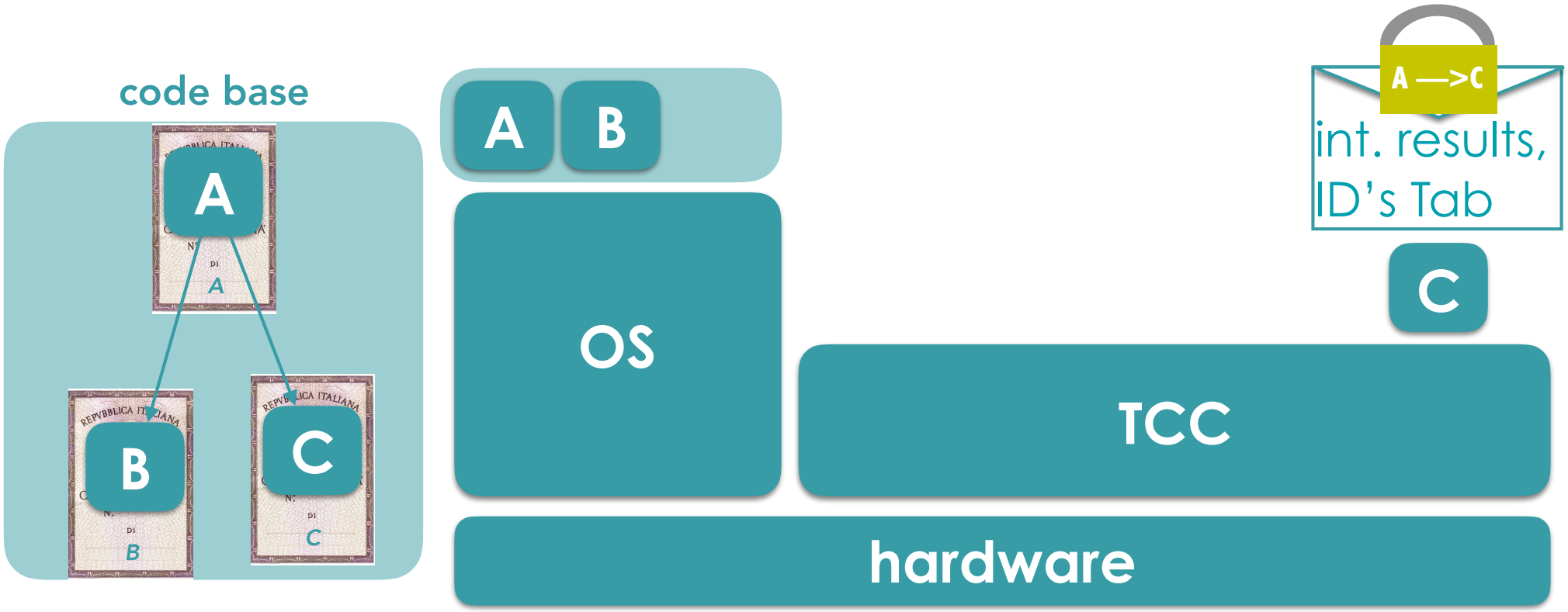
Execution Protocol



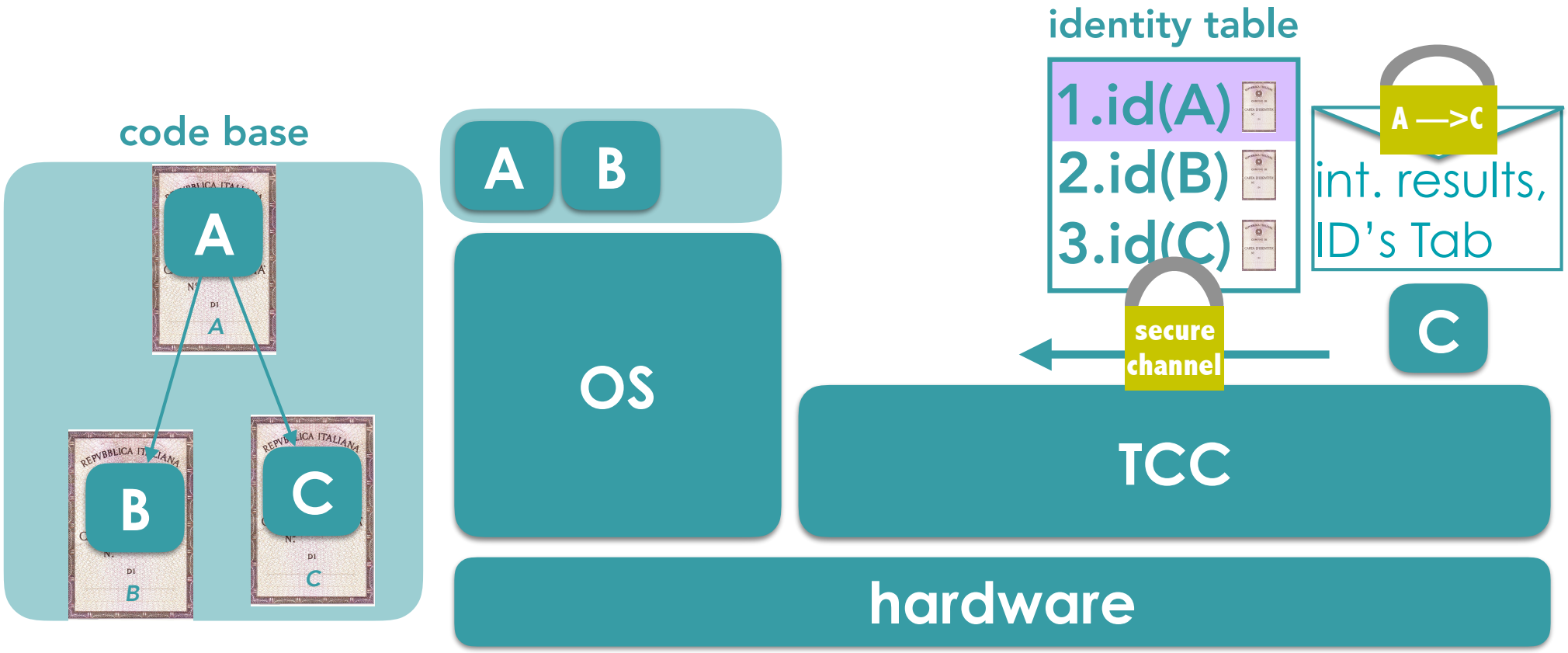
Execution Protocol



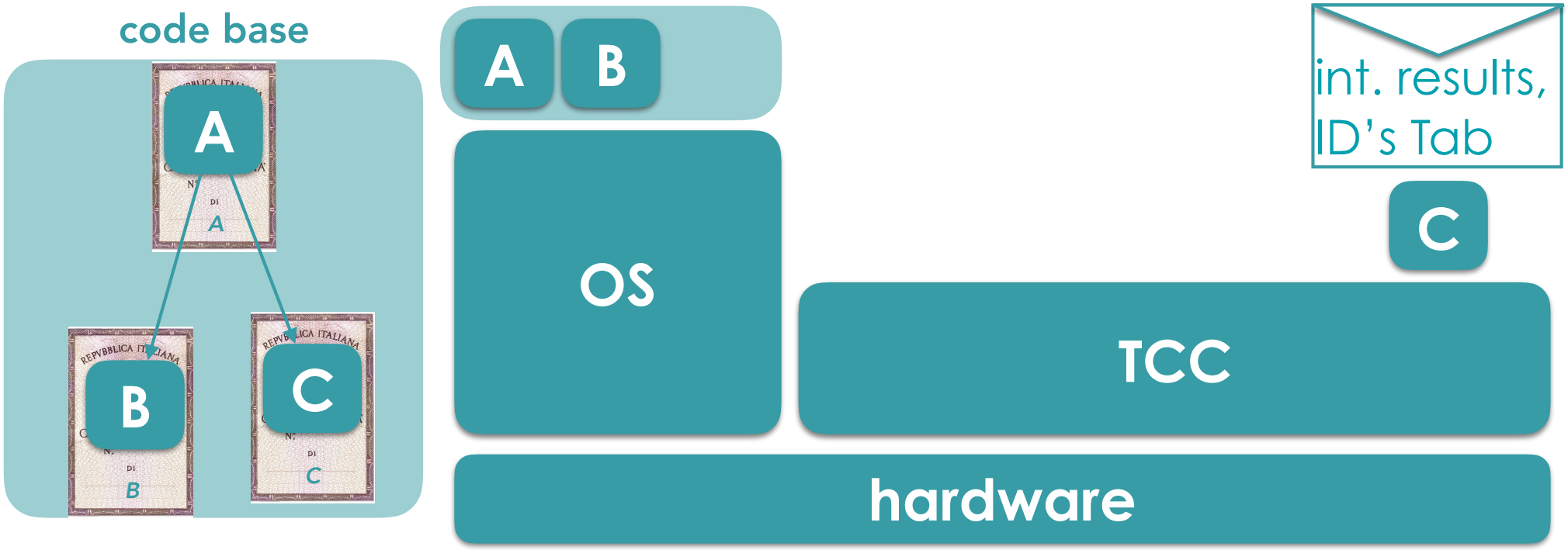
Execution Protocol



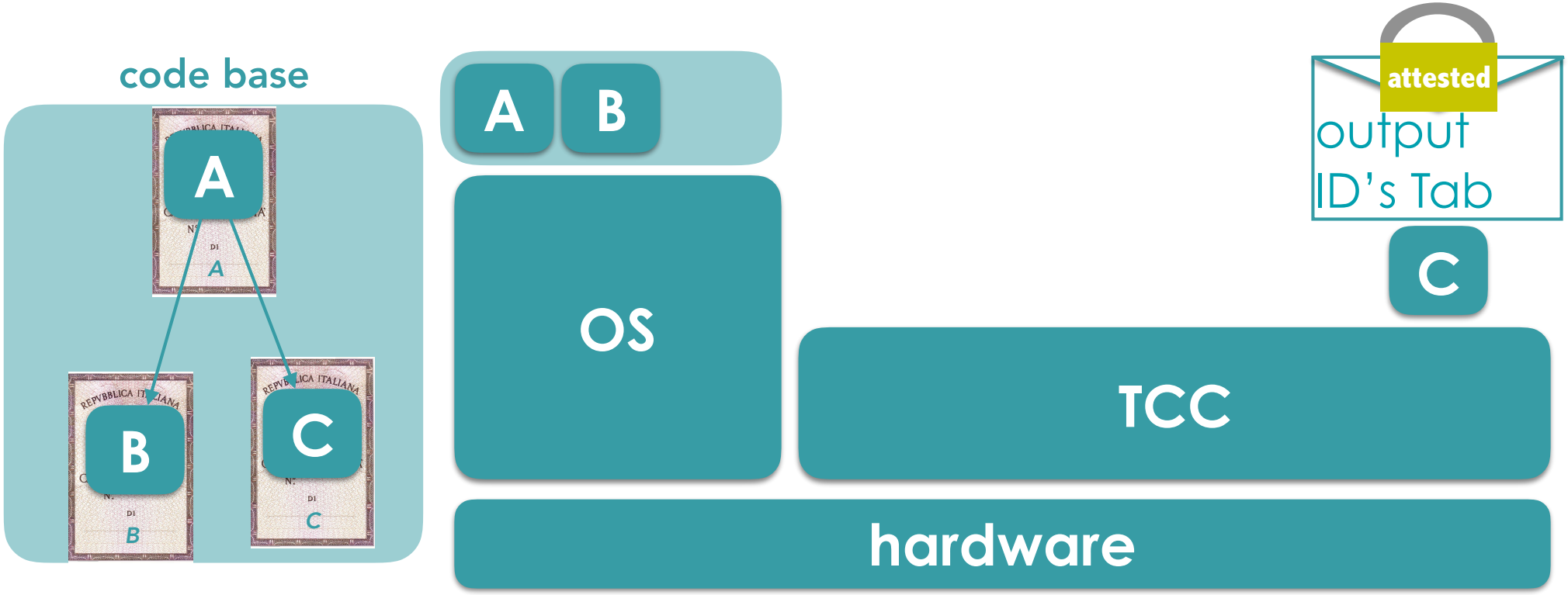
Execution Protocol



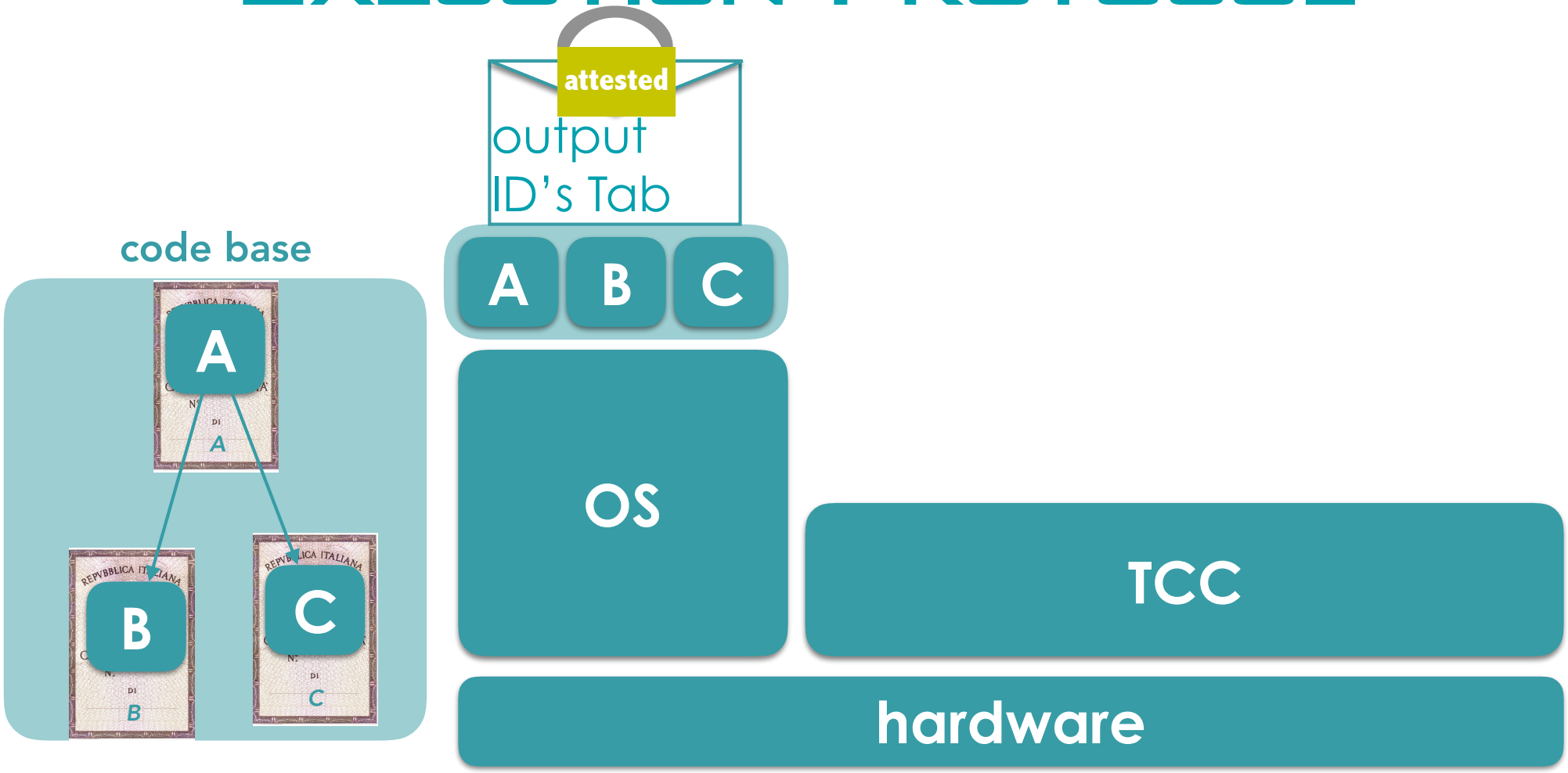
Execution Protocol



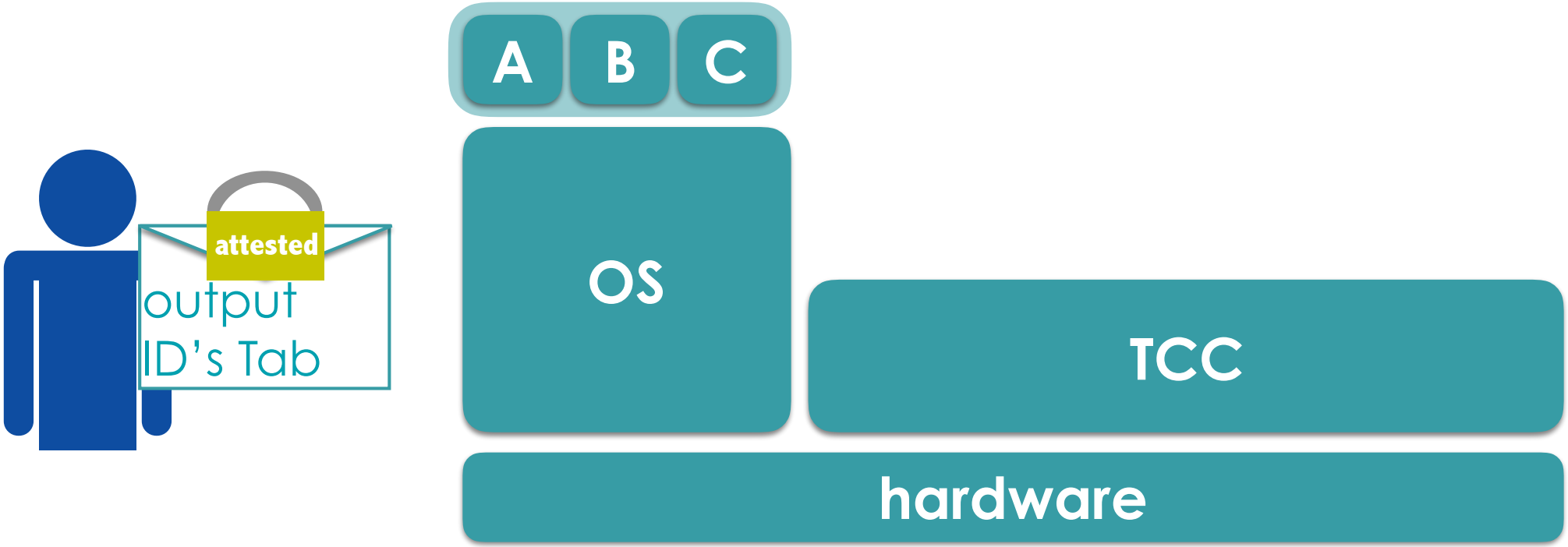
Execution Protocol



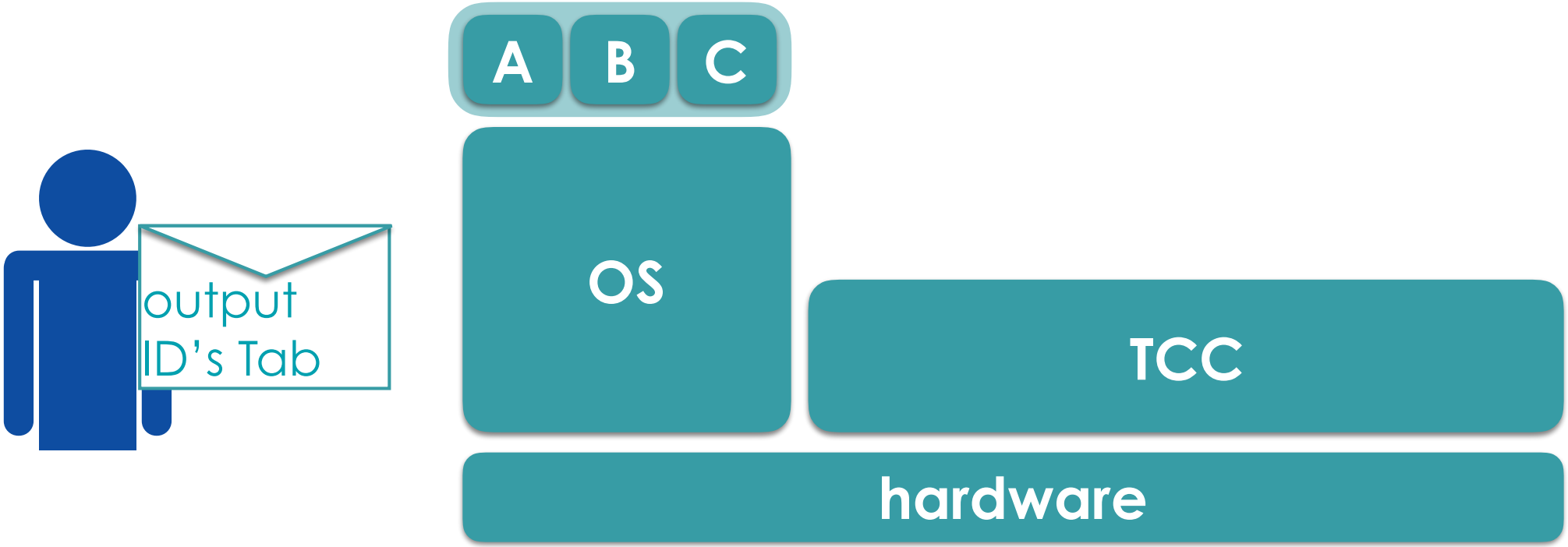
Execution Protocol



Execution Protocol



Execution Protocol

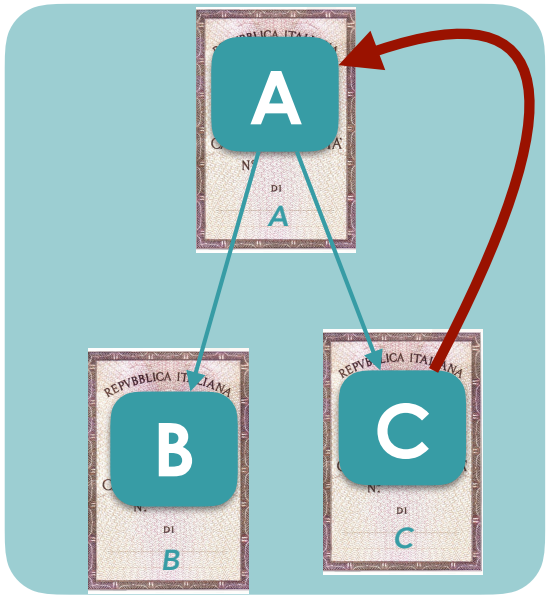


Solved Challenges for General Executions

1. Client/Verifier does not verify intermediate results
   - Results are secured locally
2. Client does not verify execution flow
   - Verification of last module & ID's Table implies correct execution flow
3. Build mutually authenticated secure channels
   - Using TCC-based secure storage
4. Fast (zero round) identity-based key sharing
   - Construction: 1 hash using sender-receiver identity pairs (see paper for details)
5. Avoid hash loops in general executions
   - Detach identity from code module using the ID's Table

5. Avoiding Hash Loops

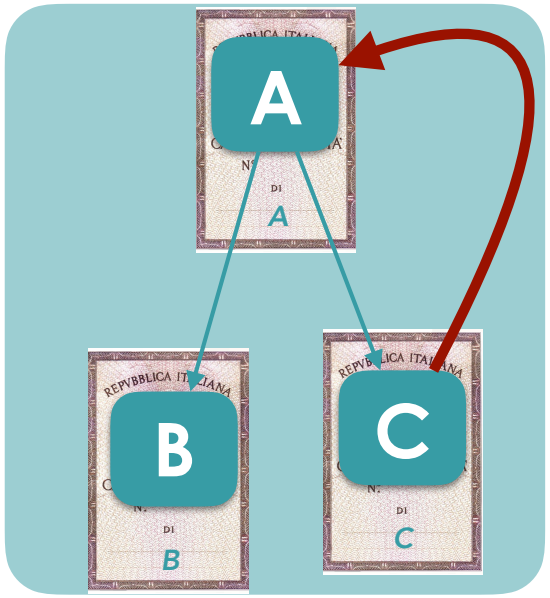
- General execution may have loops



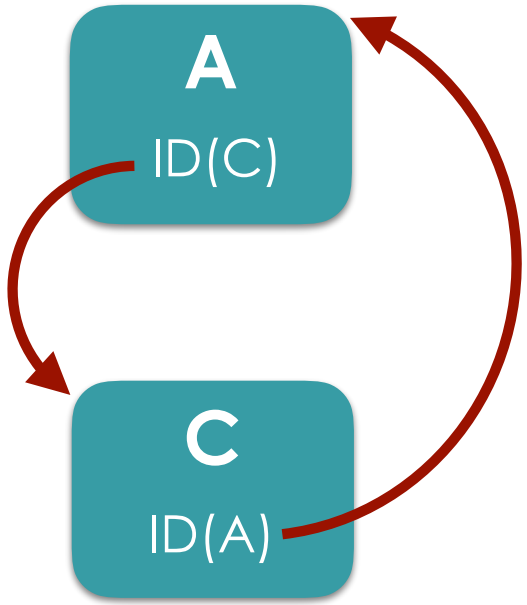
code base

5. Avoiding Hash Loops

- General execution may have loops



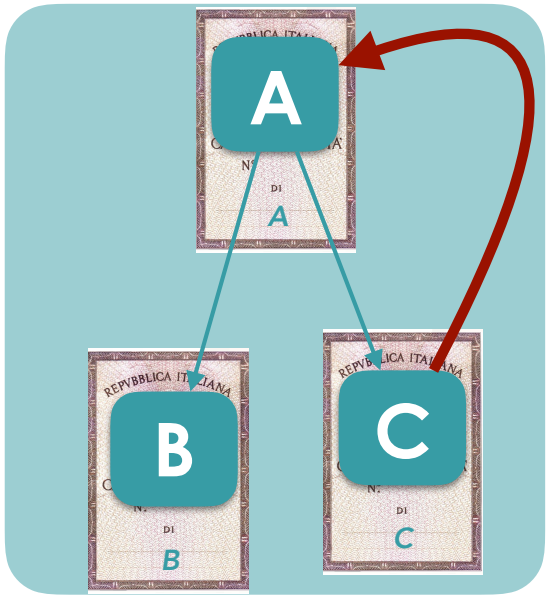
code base



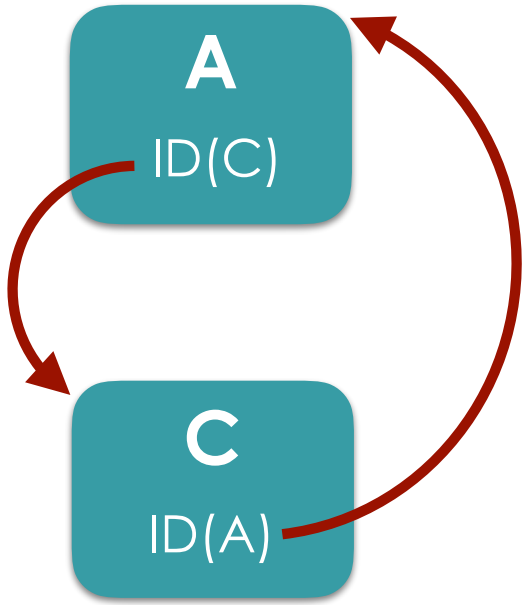
problem
(hash loop)

5. Avoiding Hash Loops

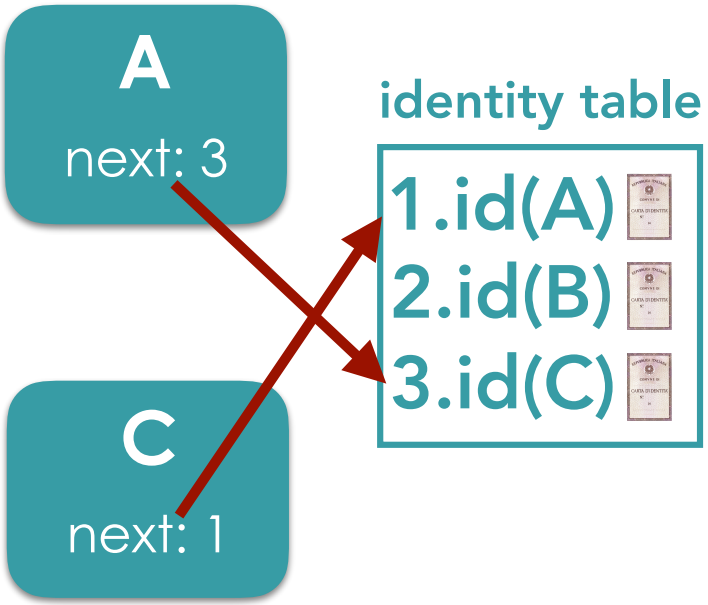
- General execution may have loops



code base



problem
(hash loop)



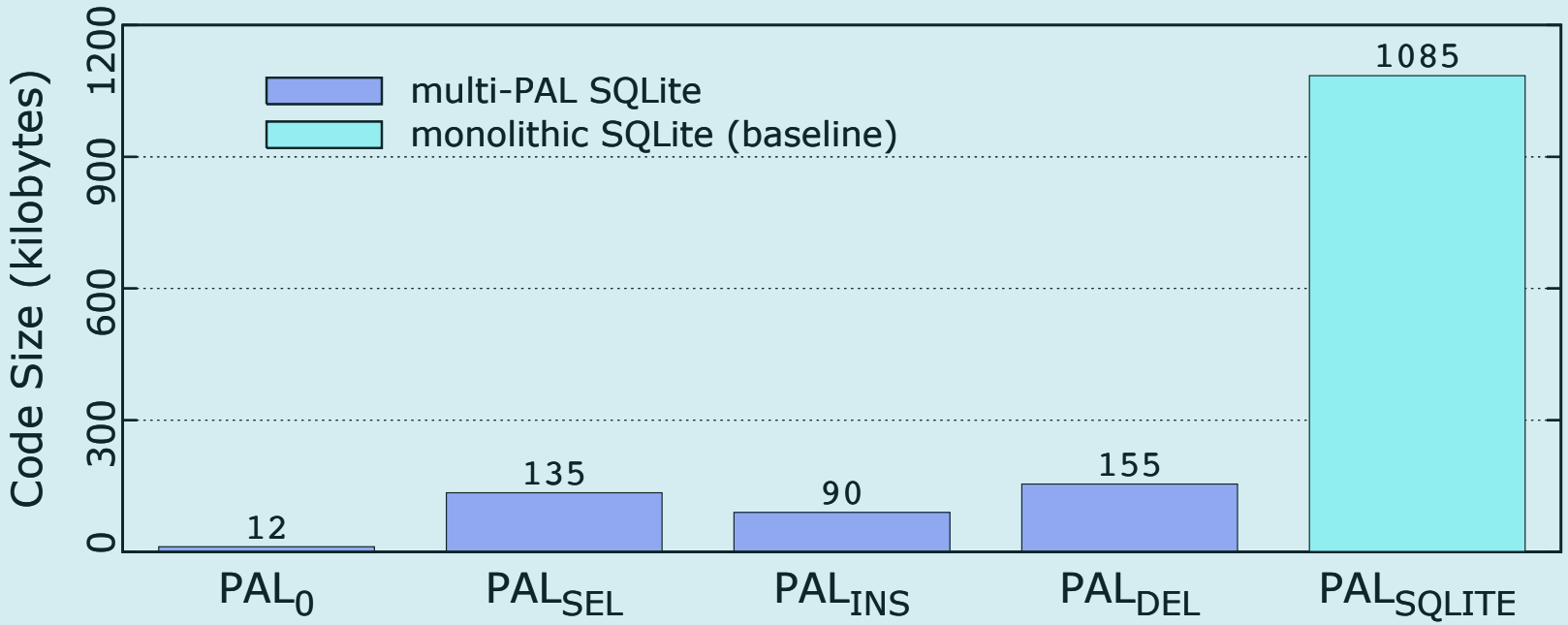
solution
(ID's in input table)

Practical Analysis

Practical Analysis

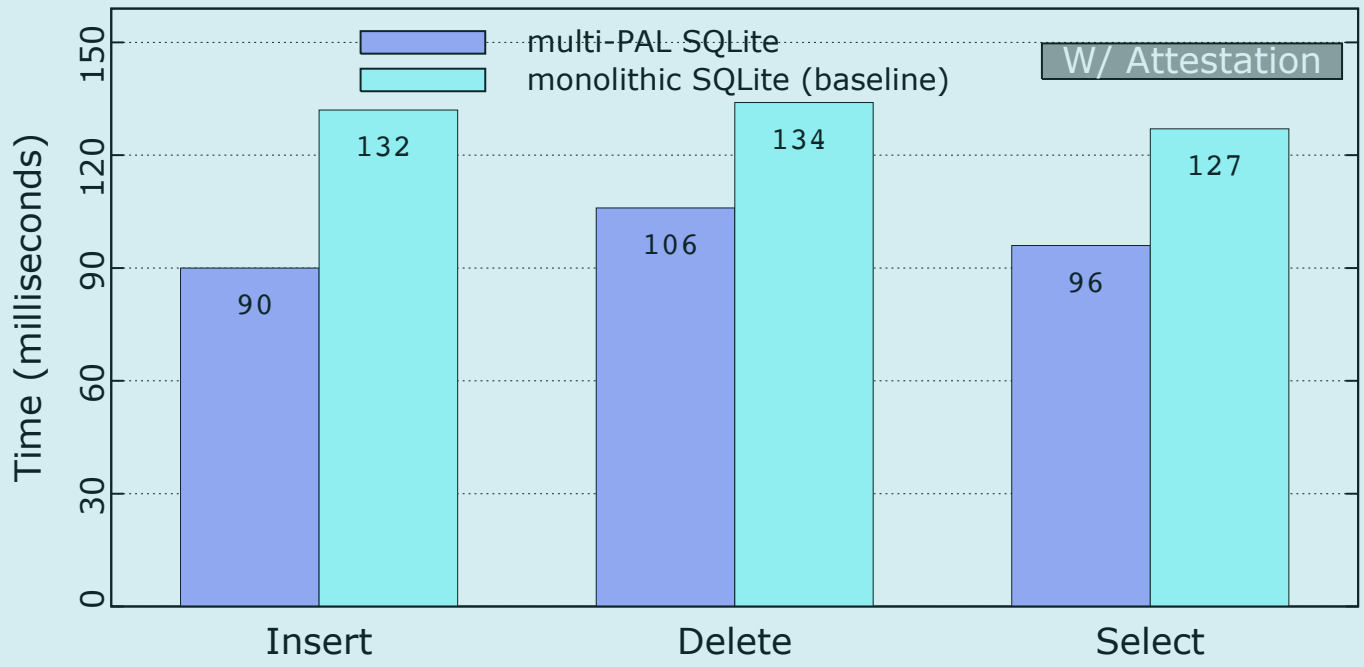
- Hypervisor-based TCC Implementation
- Protocol applied to a real-world service (SQLite)
- End-To-End experiments on server cluster

Code Size



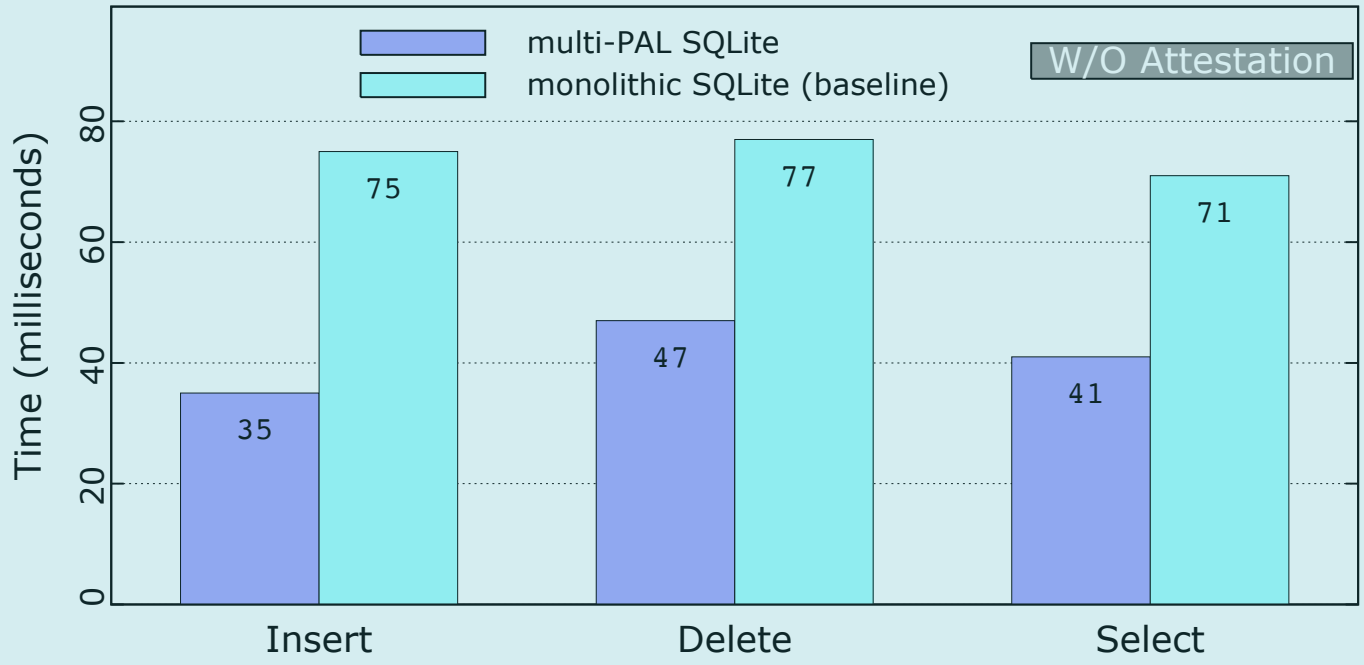
- SQLite (full implementation) is ~1MB
- 5-10x reduction of used code for single operation (PAL = Piece of Application Logic)

End-to-End Evaluation



- Same critical path, different code identification
- Monolithic SQLite is up 46% slower (w/ attestation)

End-to-End Evaluation



- Multi-PAL SQLite up to 2x faster (w/o attestation)

Conclusions

Conclusions

- **Code identification** has security/efficiency **tradeoffs**
- **Identification** of just **actively executed** code can:
  - provide **fresher integrity guarantees**
  - **improved resource usage** & **performance**
  - be done **retrofitting existing trusted components**

Real-World Service

Our Solution

Generic Trusted Component

Ciências ULisboa

Carnegie Mellon
SCHOOL OF COMPUTER SCIENCE

THANKS

BRUNO VAVALA
CMU/UL

Nuno Neves
UL

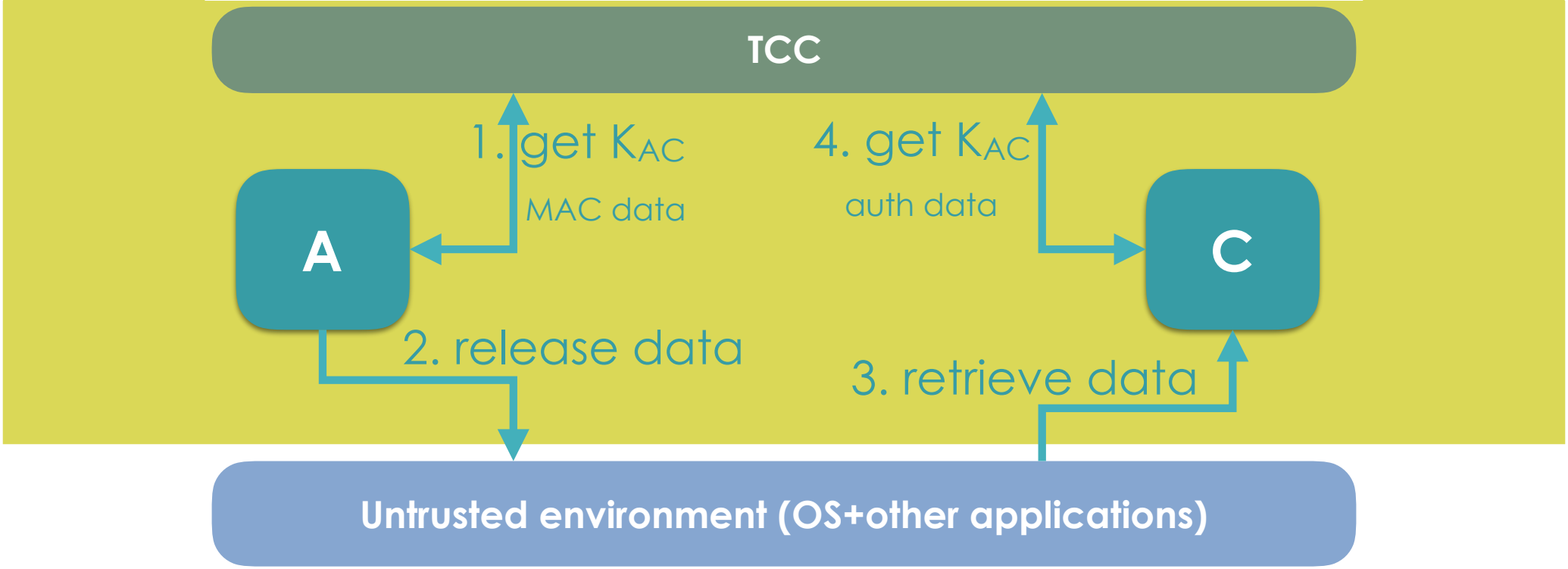
Peter Steenkiste
CMU

BLANK

BLANK

BACKUP SLIDES

Efficient Mutually-Authenticated Channels



- Identity Dependent keys
  - Sender specifies recipient's identity to TCC
  - Recipient specifies sender's identity to TCC
- Very efficient construction (one hash per-key)