



Secure Program Partitioning

FOSAD 2008

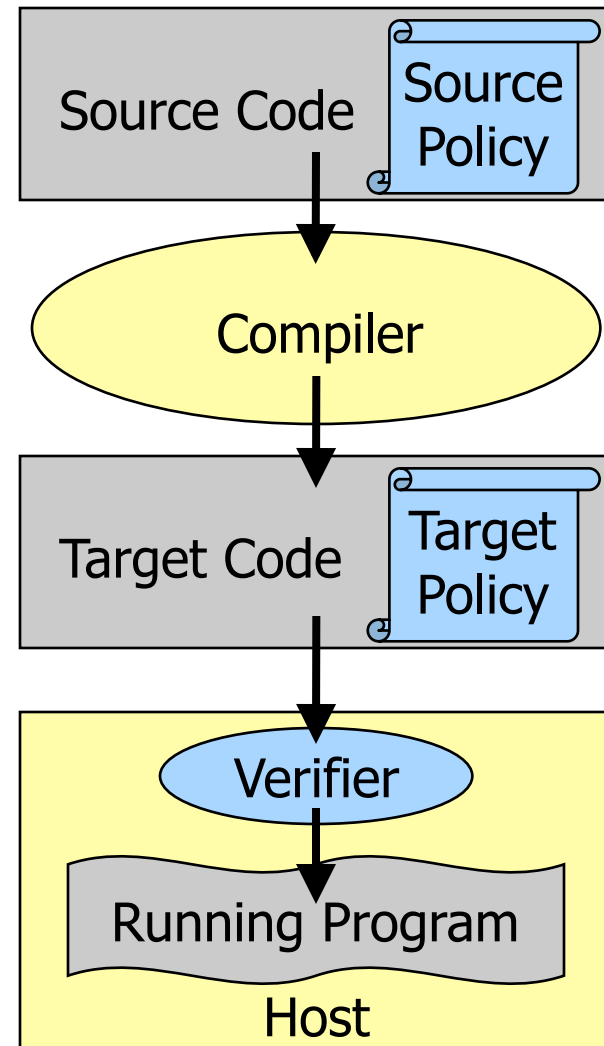
Steve Zdancewic
University of Pennsylvania

Overview

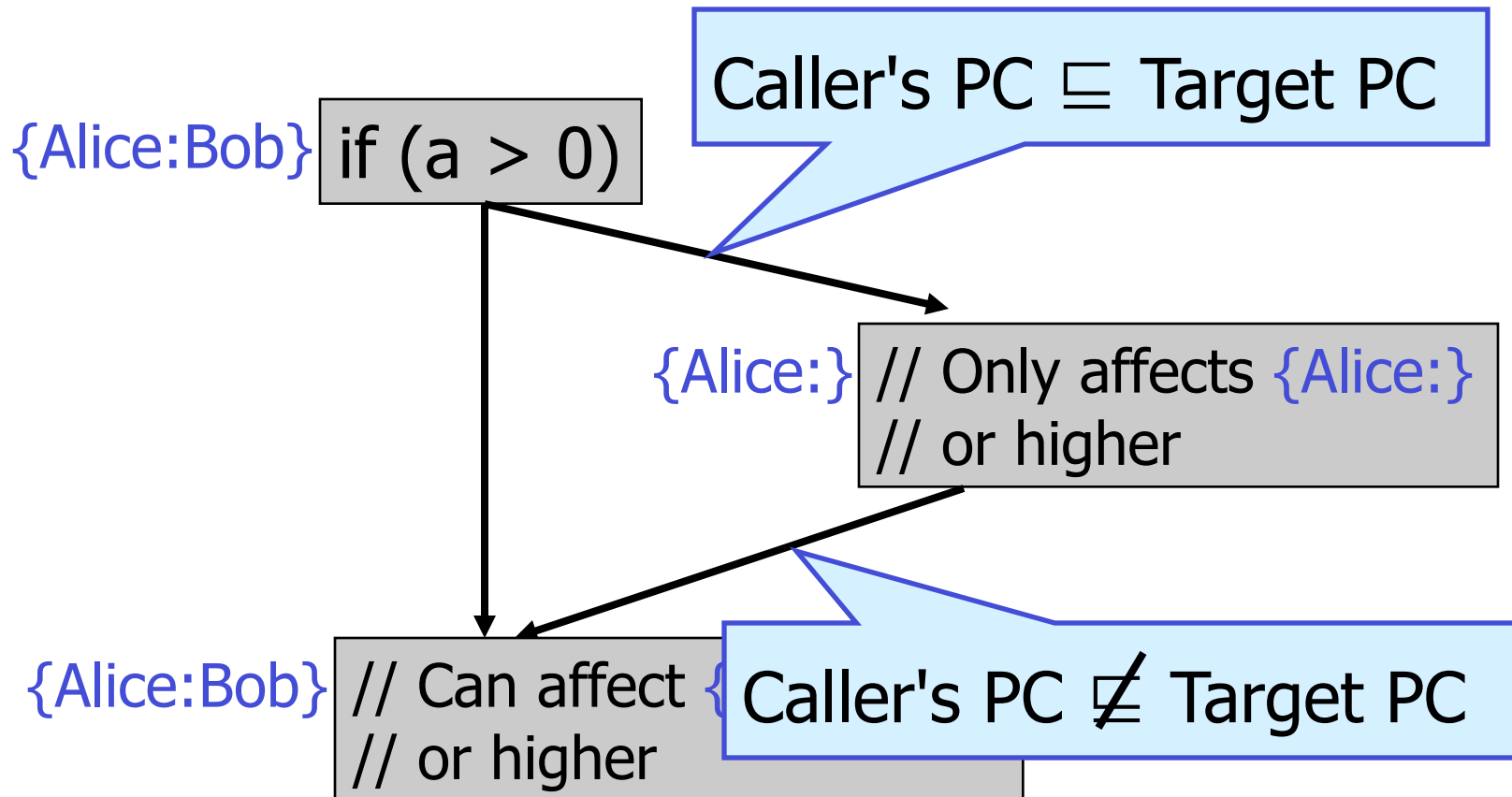
- Adding security policies to software gives us a lot more information: what else can we do with it?
- Typed compilation
 - Typed assembly language
 - Bytecode
- Distributed system implementation

Certified Compilation

- Typed Compilation
Proof-Carrying Code
[Morrisett] [Necula&Lee]
- Verify compiler *output*
- Verifier is much simpler than the full compiler
- Problem: Produce a simple-to-check & precise target policy.

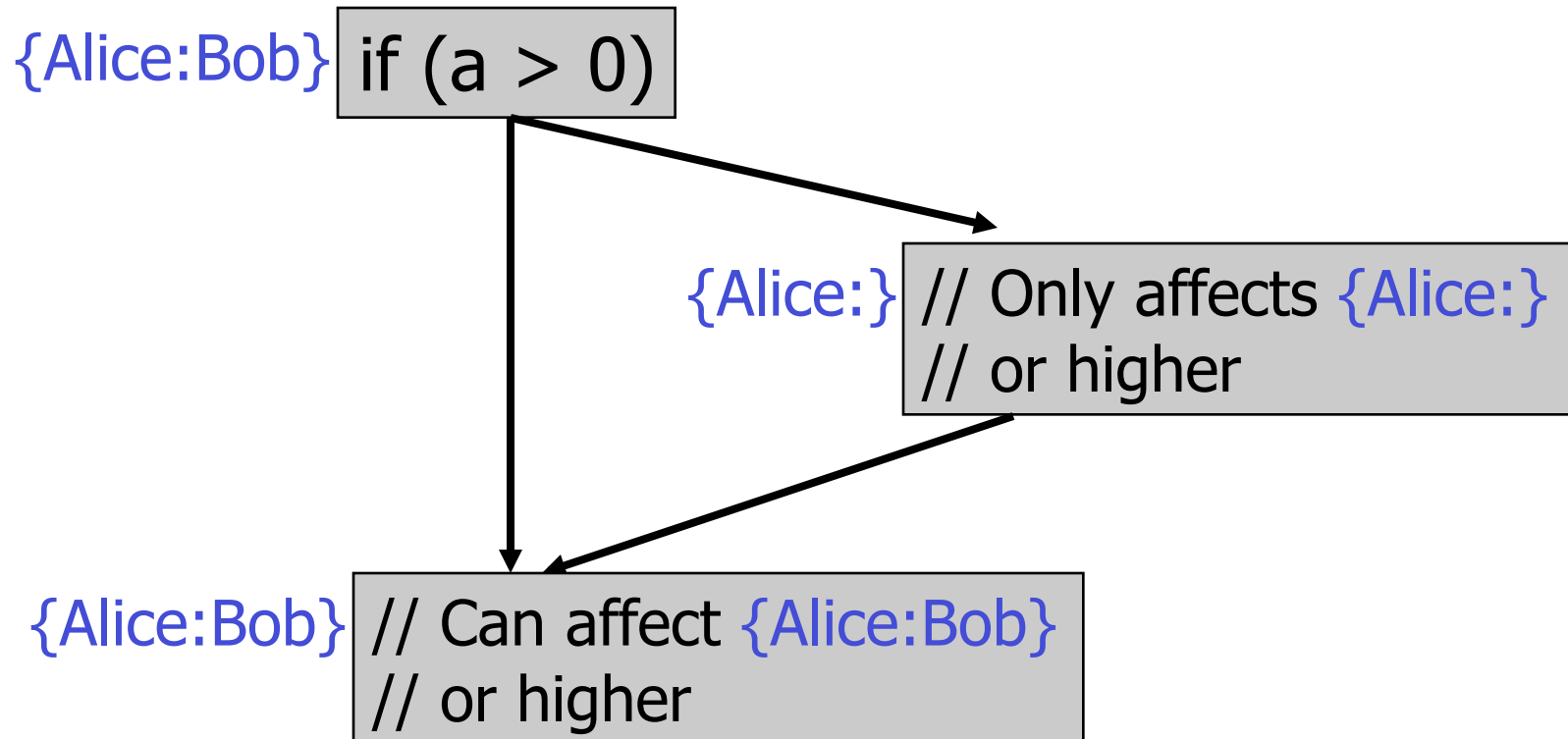


Control-Flow Graph

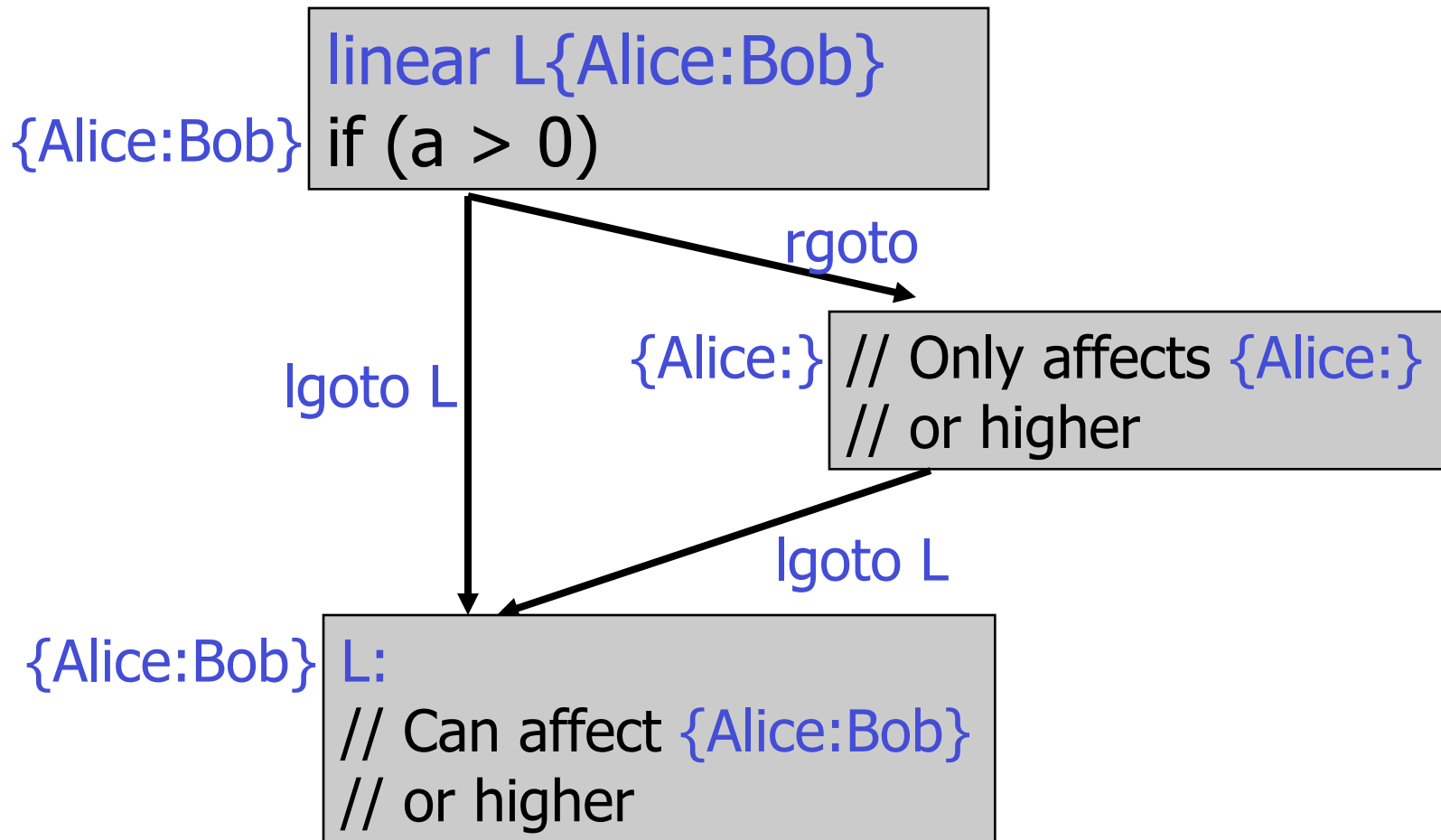


Postdominators

[Denning & Denning '77]



Linearity Annotations



Linearity Annotations

```
L0:  linear L3{Alice:Bob};  
     if (a > 0) rgoto L1;  
     lgoto L3;  
L1:  a = 4;  
     lgoto L3;  
L3:  b = b + 1;
```

Linearity Annotations

- Generalize the post-dominator analysis
 - lexical scoping (blocks, conditionals)
 - switch tables
 - (returns from) first class function calls
 - tail call optimization
 - "finally" clauses for Java-style exception handling
- Linear logic [Girard '87] [Wadler '90]
 - Principled foundation for the analysis

Soundness Results

[Zdancewic & Myers ESOP'01, HOSC'01]

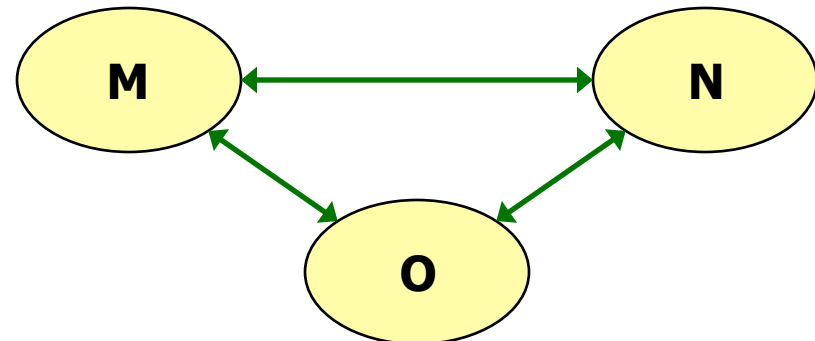
- Sound type system for low-level information-flow security
 - Not yet TAL level (See subsequent work)
 - Translation from source language
 - higher-order functions & state
 - simple, precise annotations
 - Theorem: *If a program type-checks in the low-level type system, it satisfies the noninterference policy expressed by its type.*
-

Heterogeneous Trust

- Previous work assumed "universally trusted" platform.
 - Not realistic
 - Violates principle of least privileges

"Alice trusts M & O"

"Bob trusts N & O"



[ZZNM SOSP'01, ZCZM Oakland'03]

Trust Configurations

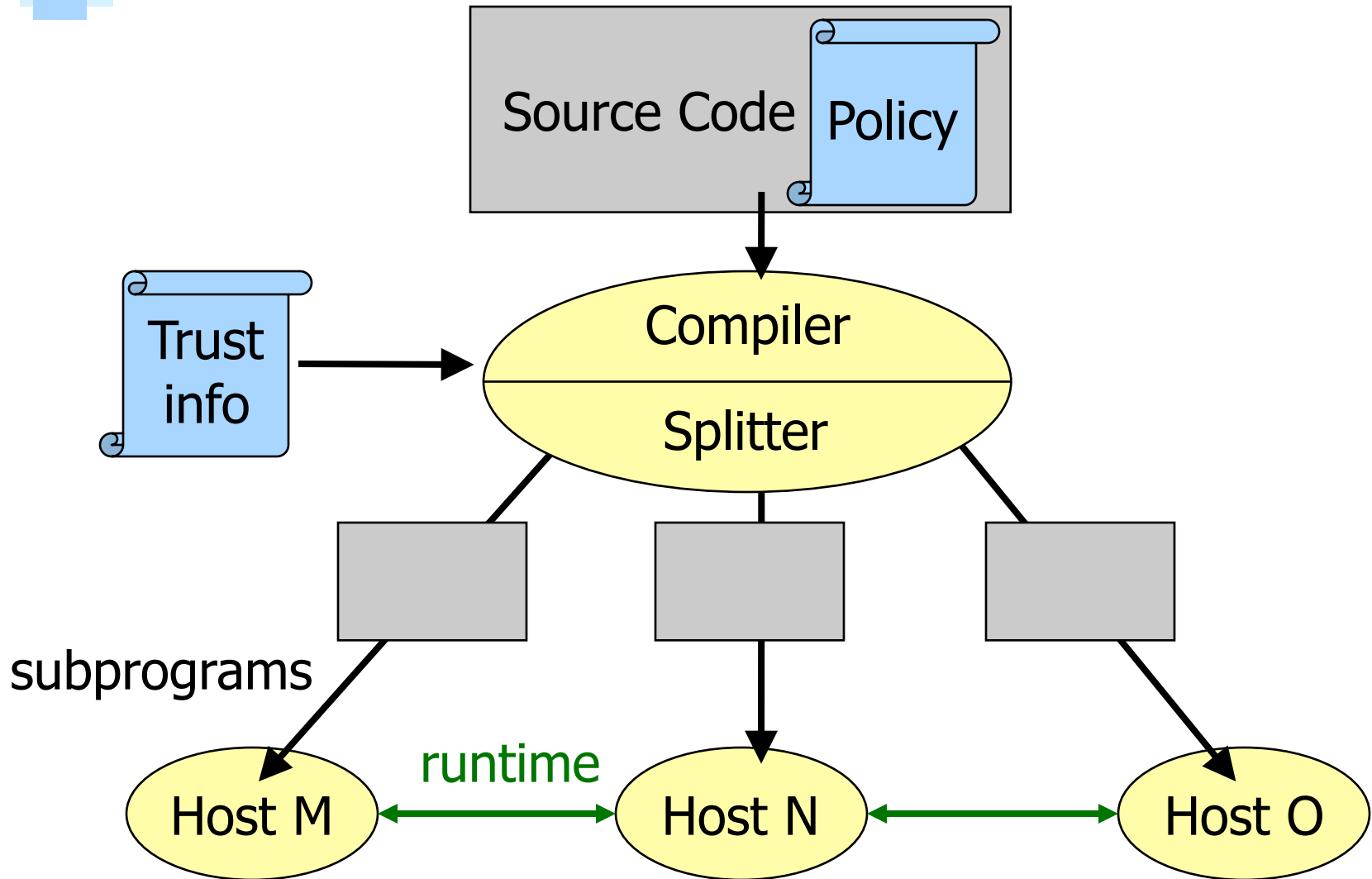
- Labels describe the trust relationship between principals and the available hosts.

- Confidentiality: Host M: {Alice:}
"Alice trusts host M to hold her confidential data."

int{Alice:} a; a can be sent to M
int{Bob:} b; b cannot be sent to M

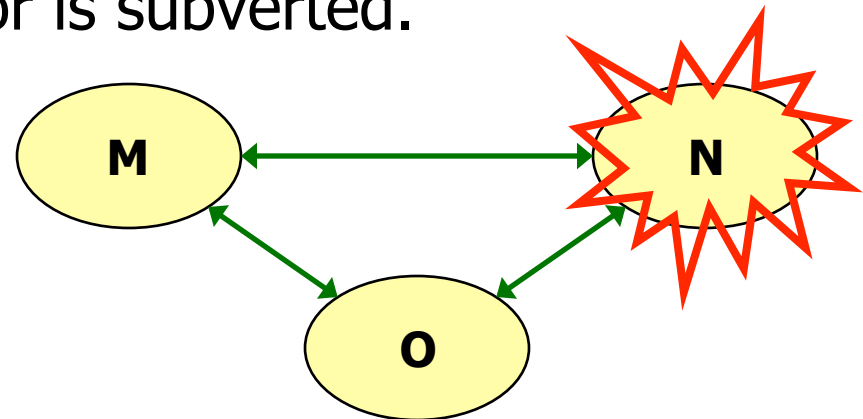
- Integrity: Host M: {Alice?}
"Alice trusts host M not to corrupt her high-integrity data."

Program Partitioning



Security Assurance

- **Goal:** Resulting distributed program performs the same computation as the source and also satisfies the security policies.
- **Guarantee:** Principal P's security policy is violated only if a host that P trusts fails or is subverted.
- **Example:**

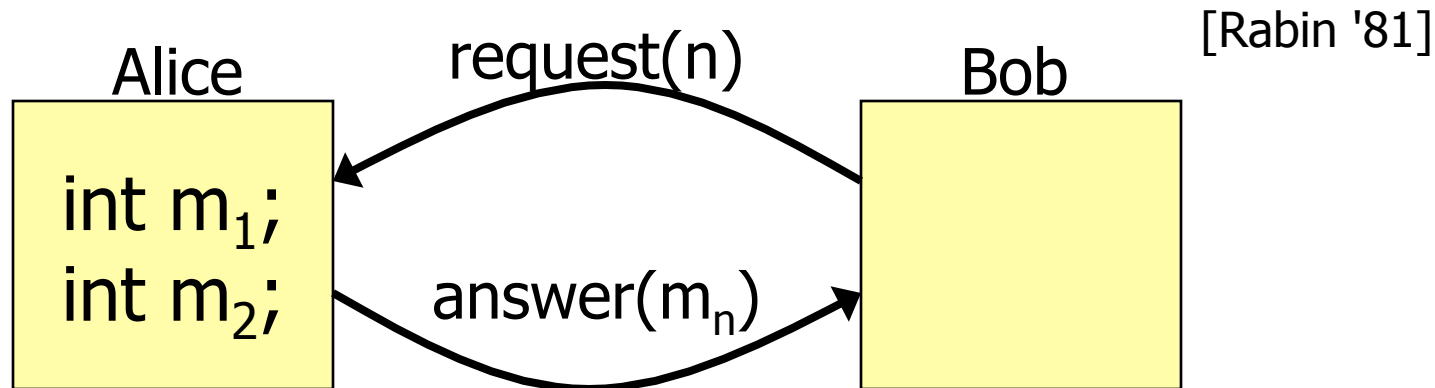


"Alice trusts M & O"

"Bob trusts N & O"

If N fails, Alice's policy is obeyed, Bob's policy may be violated.

Example: Oblivious Transfer



- Alice's Policy:
"Bob gets to choose exactly one of m_1 and m_2 ."
- Bob's Policy:
"Alice doesn't get to know which item I request."
- Classic Result: "Impossible to solve using 2 principals, with perfect security."

[Damgård, Kilian, Slavail '99]

Oblivious Transfer (Java)

```
int      m1, m2;    // Alice's data
boolean  accessed;
int      n, ans;    // Bob's data


---


n = choose();      // Bob's choice

if (!accessed) {  // Transfer
    accessed = true;
    if (n == 1)
        ans = m1;
    else ans = m2;
}
```


Adding Confidentiality Labels

```
int{Alice:}      m1, m2;    // Alice's data
boolean         accessed;
int{Bob:}       n, ans;    // Bob's data


---


n = choose();    // Bob's choice

if (!accessed) {      // Transfer
    accessed = true;
    if (n == 1)
        ans = m1;
    else ans = m2;
}
```



Using Declassification

```
int{Alice:}      m1, m2;    // Alice's data
boolean         accessed;
int{Bob:}       n, ans;    // Bob's data

```

```
n = choose();

if (!accessed) {
    accessed = true;
    if (n == 1)
        ans = declassify(m1, {Alice:Bob});
    else ans = declassify(m2, {Alice:Bob});
}

```

Verification Fails

choice

// Transfer

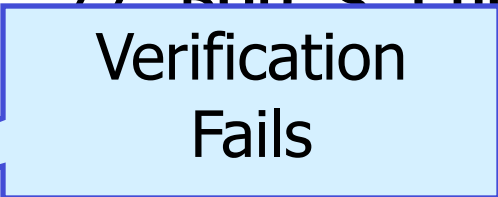
Integrity Constraints

```
int{Alice:}          m1, m2;    // Alice's data
boolean{Alice?}     accessed;
int{Bob:}           n, ans;    // Bob's data

```

```
n = choose(); // Bob's choice

if (!accessed) {
  accessed = true;
  if (n == 1)
    ans = declassify(m1, {Alice:Bob});
  else ans = declassify(m2, {Alice:Bob});
}
```



Verification Fails

Using Endorsement

```
int{Alice}          m1, m2;    // Alice's data
boolean{Alice?}    accessed;
int{Bob}           n, ans;    // Bob's data

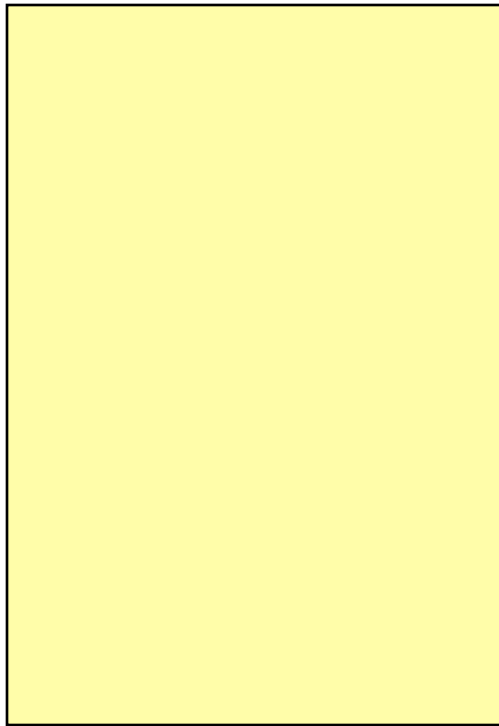

---


n = choose();      // Bob's choice

if (!accessed) {    // Transfer
    accessed = true;
    if (endorse(n by Alice) == 1)
        ans = declassify(m1 to Bob);
    else ans = declassify(m2 to Bob);
}
```

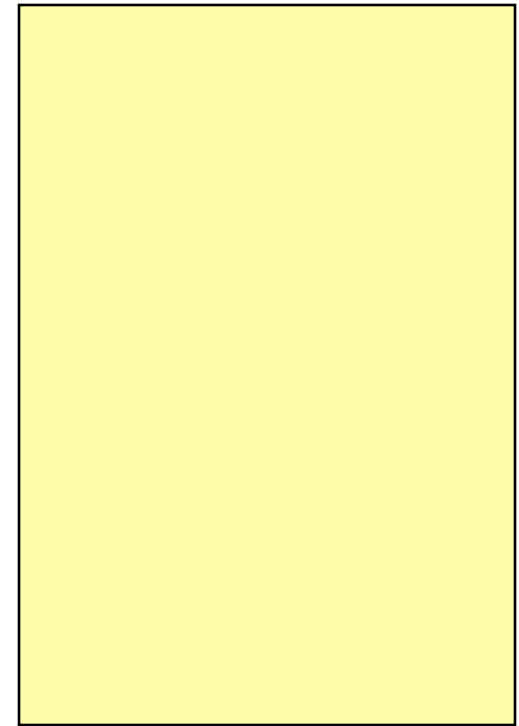
Two-Host Configuration

A



{Alice, Alice?}

B



{Bob, Bob?}

No Solution!

A

B

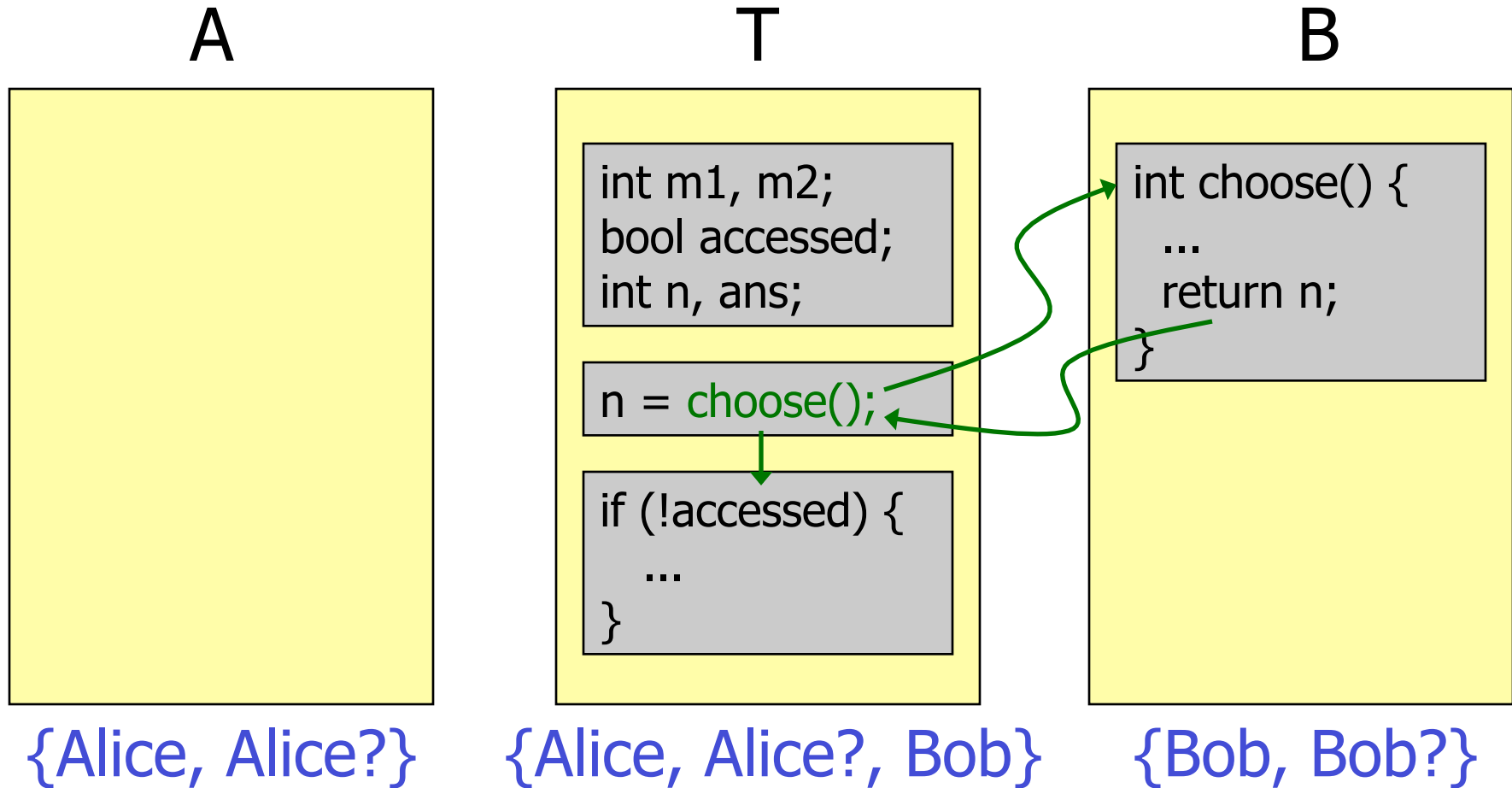
Can't find a host for the expression:
`endorse(n by Alice)`

It requires a host that has Alice's integrity yet can hold Bob's private data.

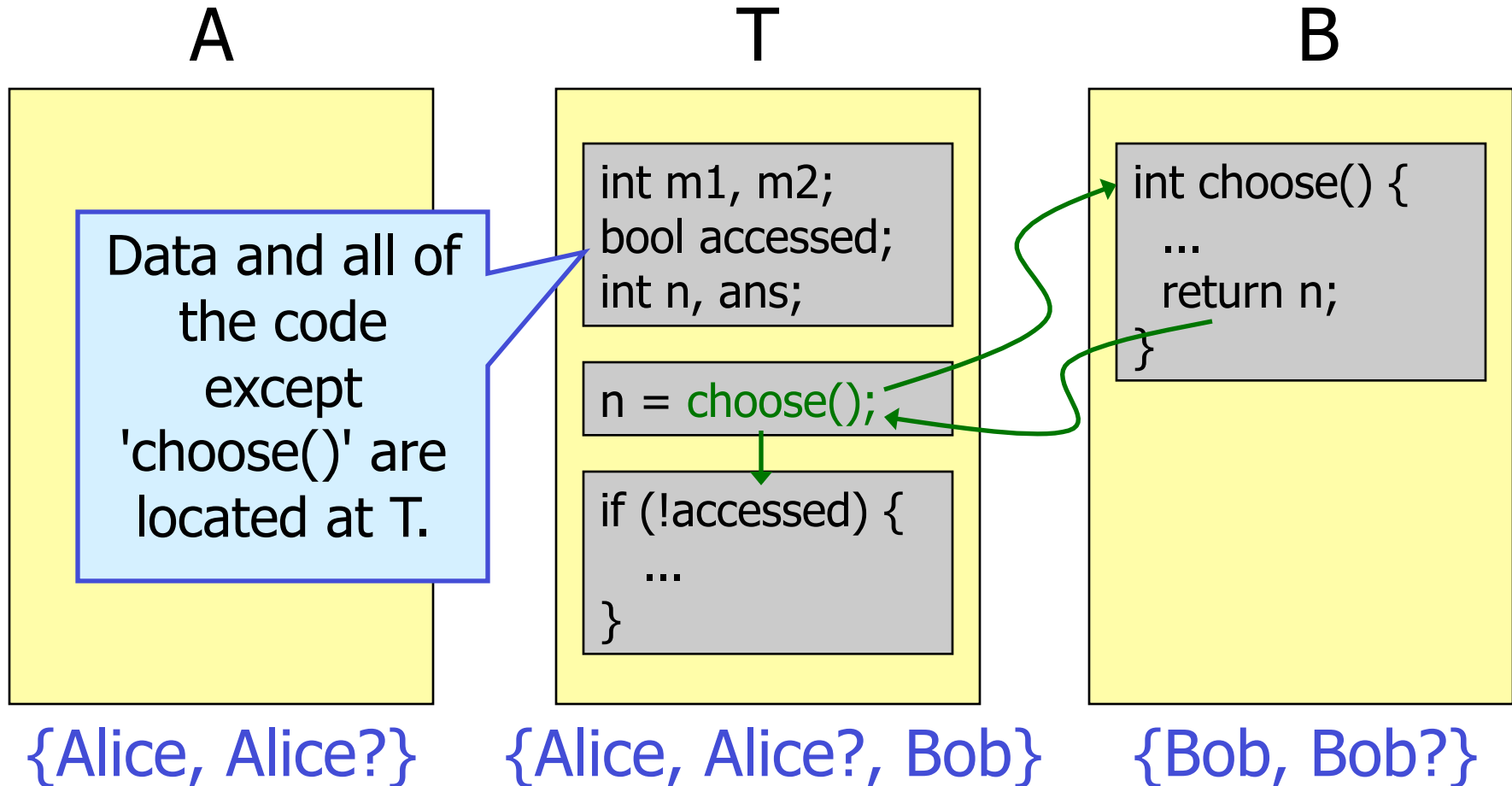
{Alice, Alice?}

{Bob, Bob?}

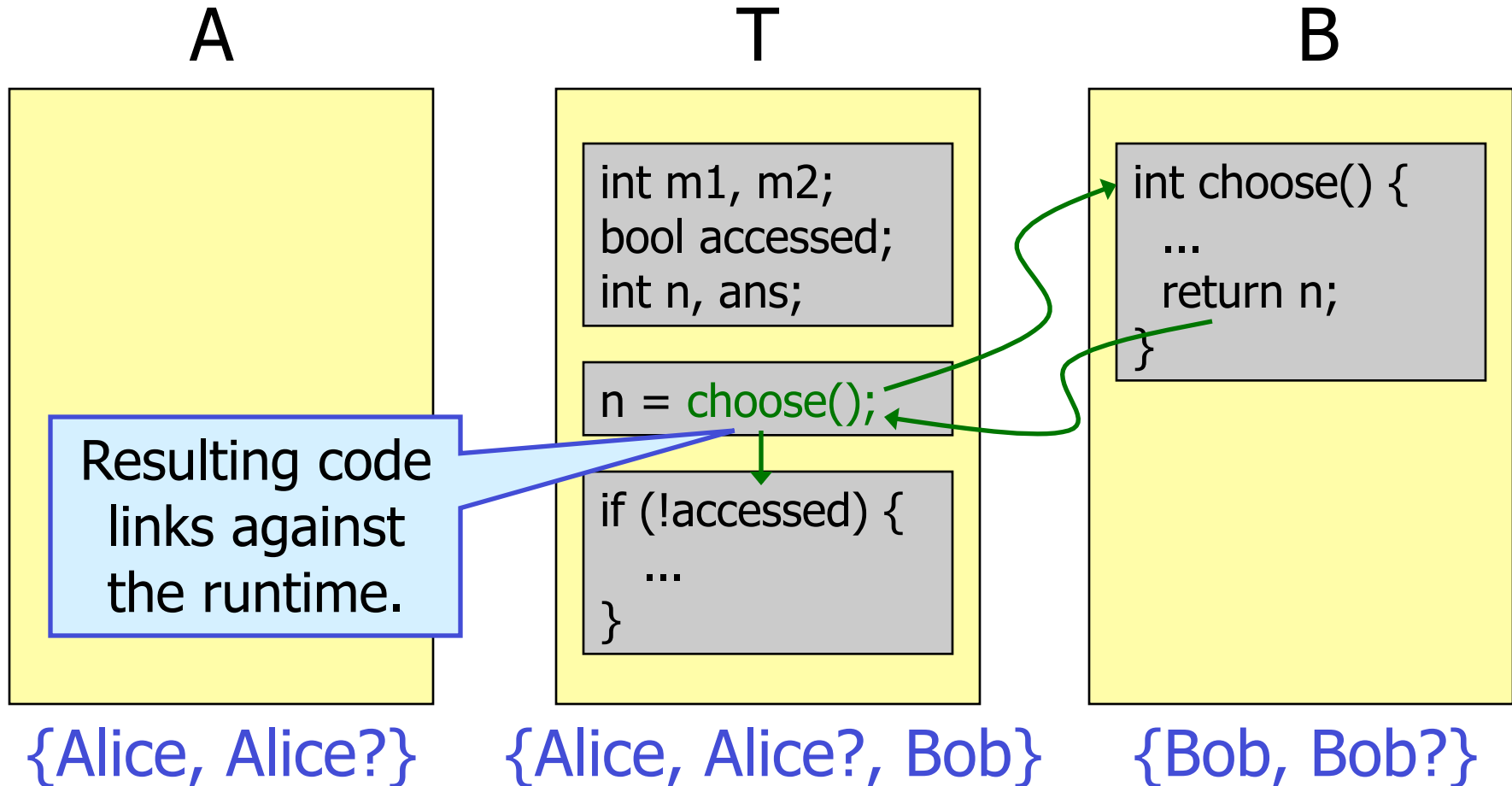
Splitter Output



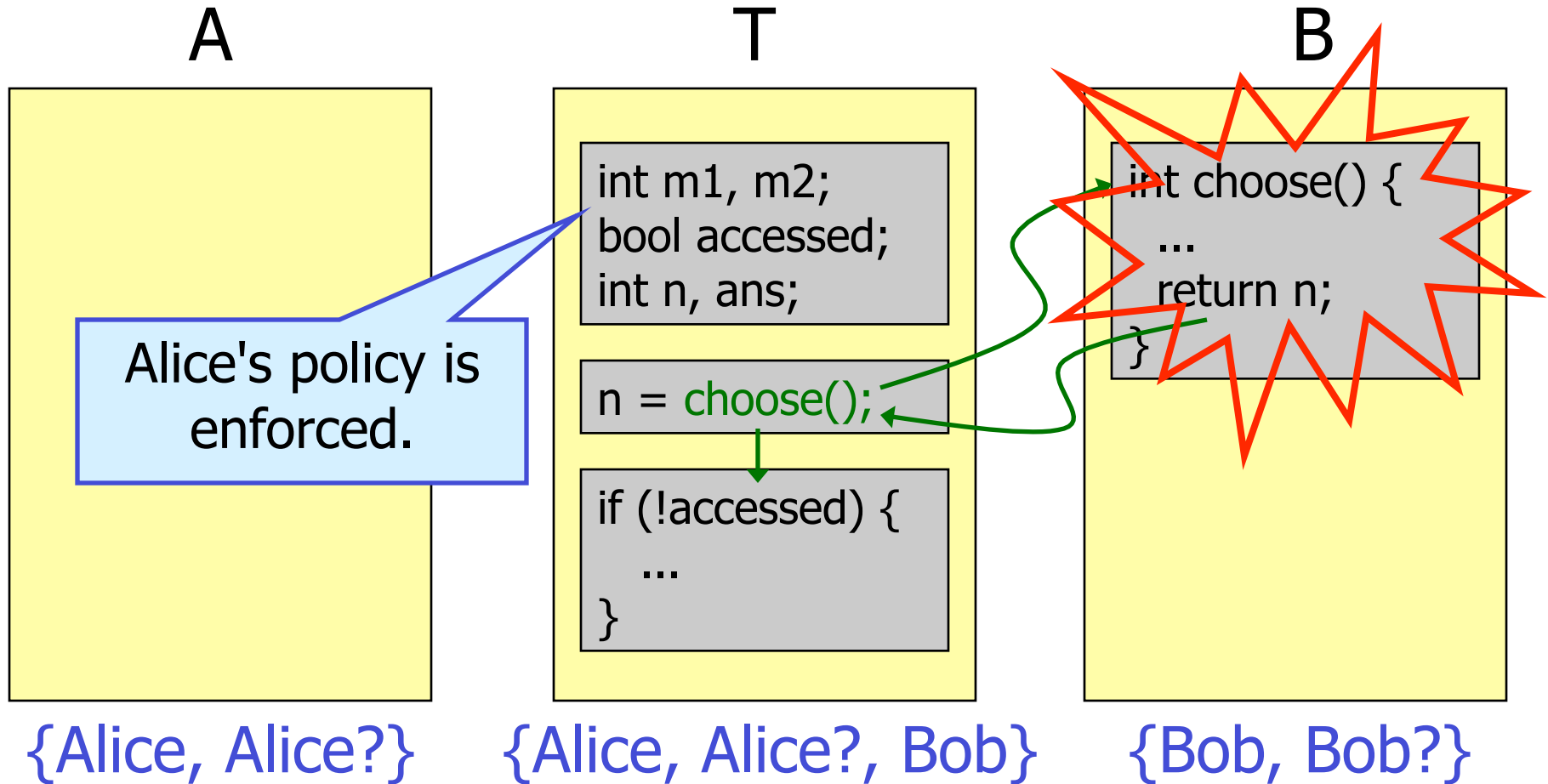
Splitter Output



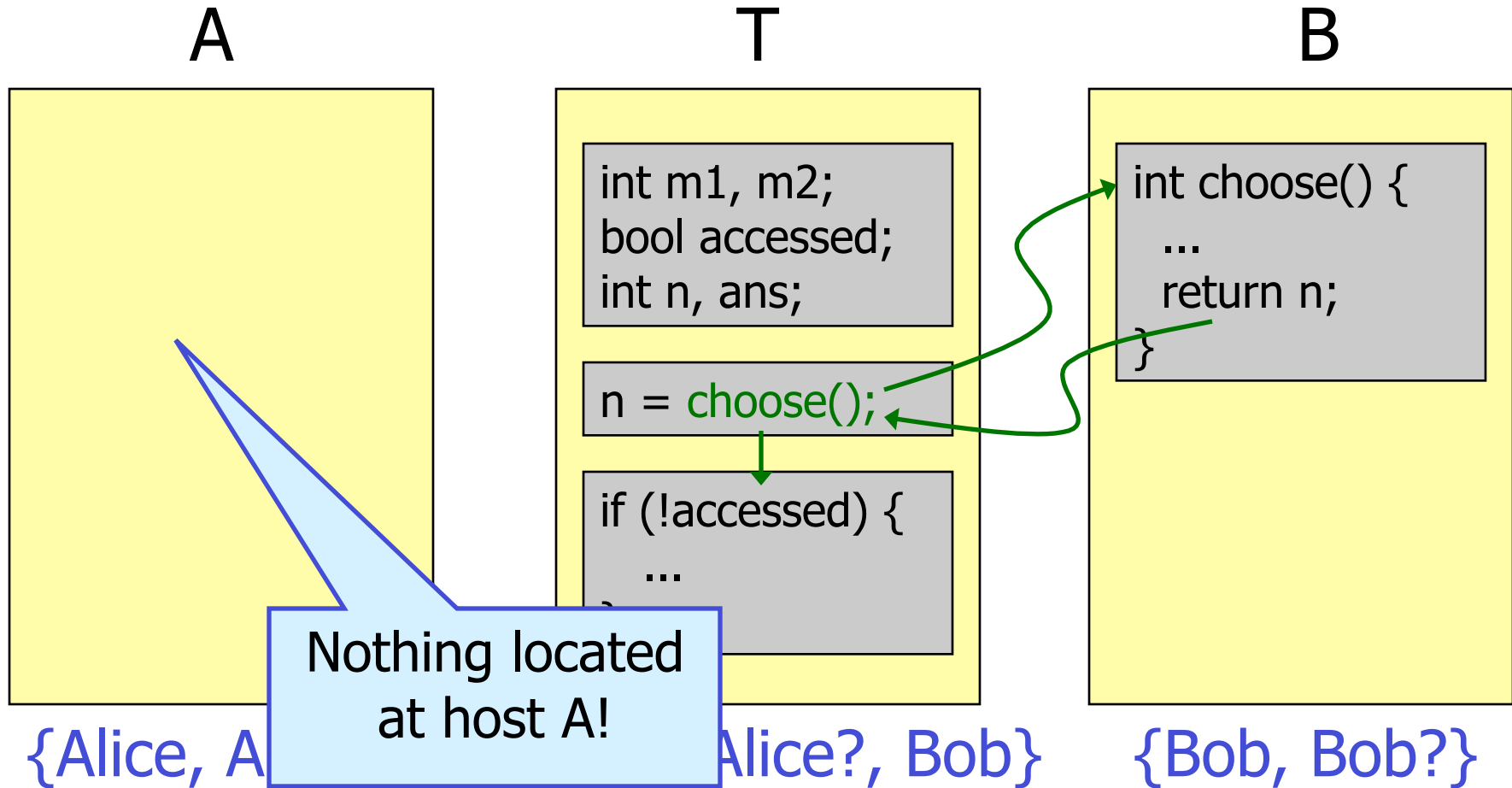
Run-time Calls



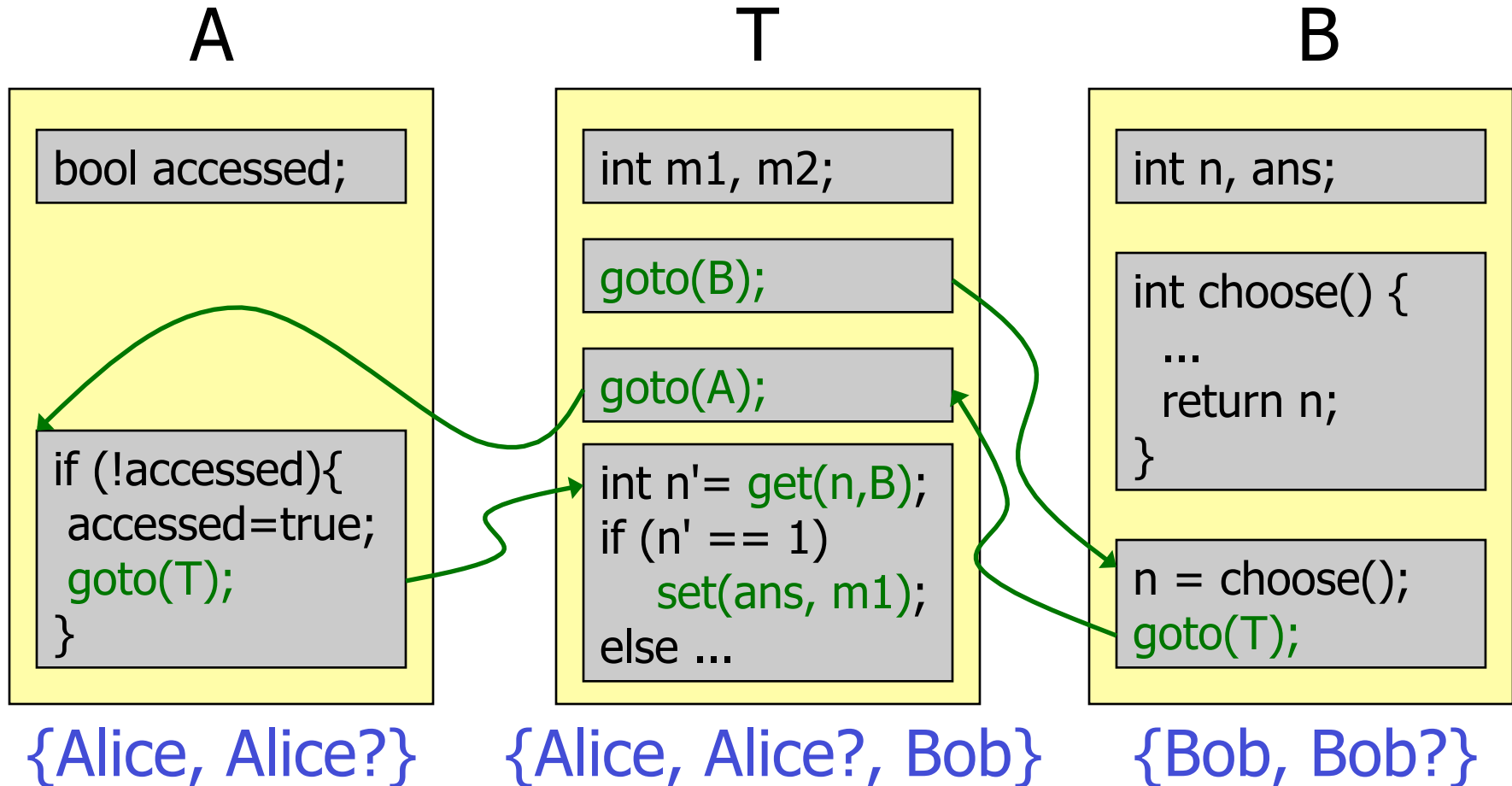
Security Assurance



Network Overhead



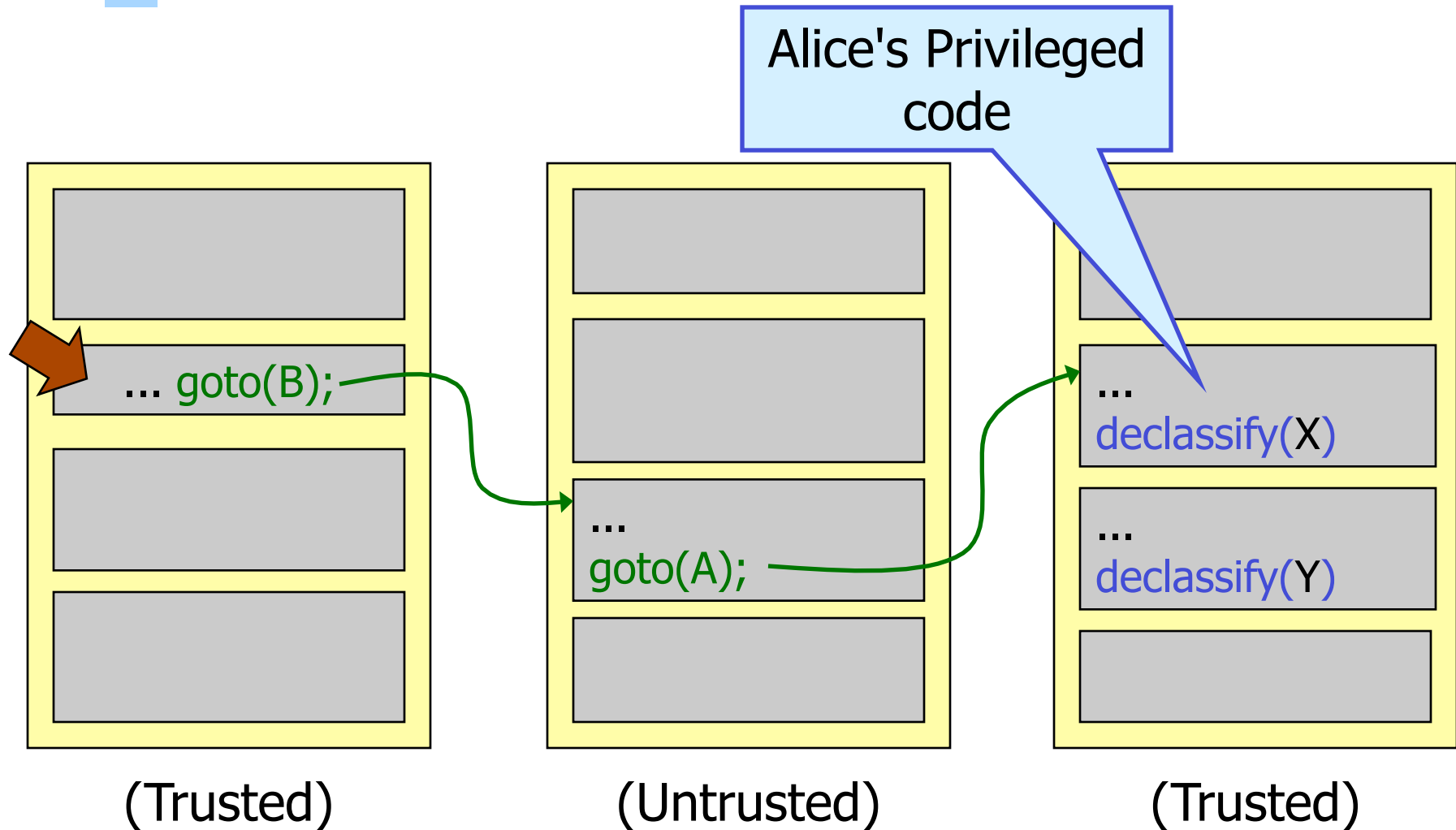
Another Solution



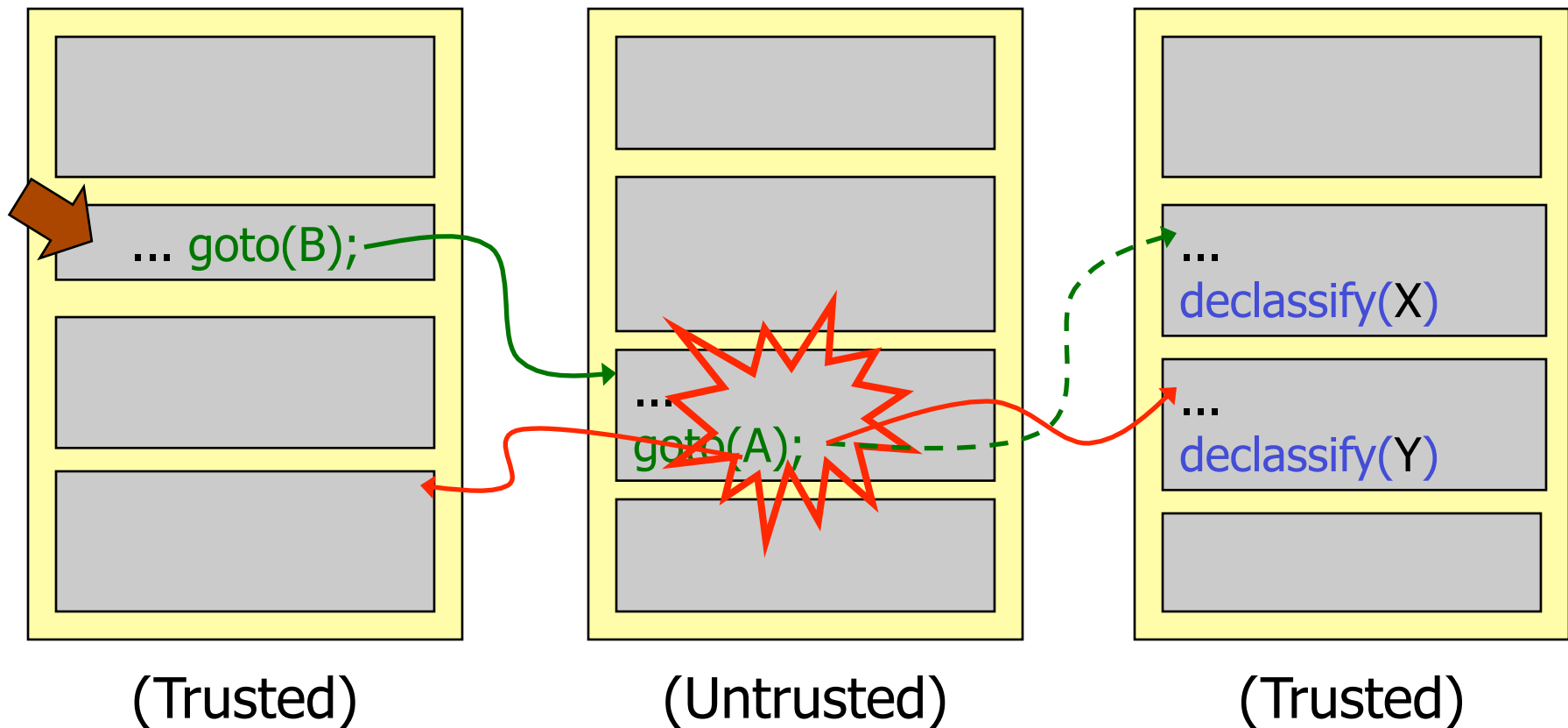
Core Runtime Protocol

- Data Transfer
 - getField, setField, forward
- Control Transfer
 - rgoto, lgoto, sync
- Prototype Implementation
 - Very small runtime (1700 LOC)
 - Authentication & access control
- Rely on:
 - cryptographic technology
 - secure network communication [kerberos, ssh]

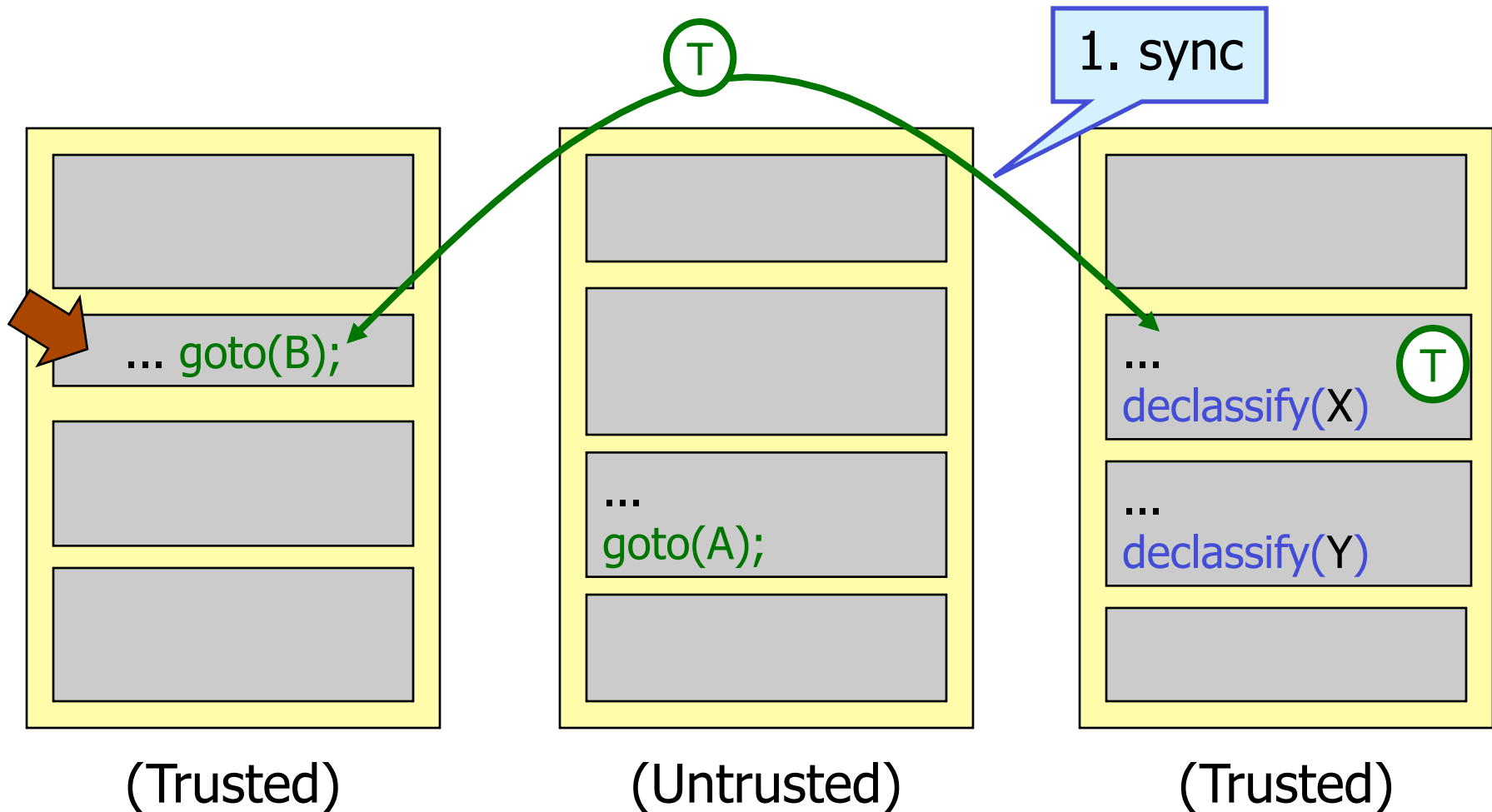
Control Transfer Integrity



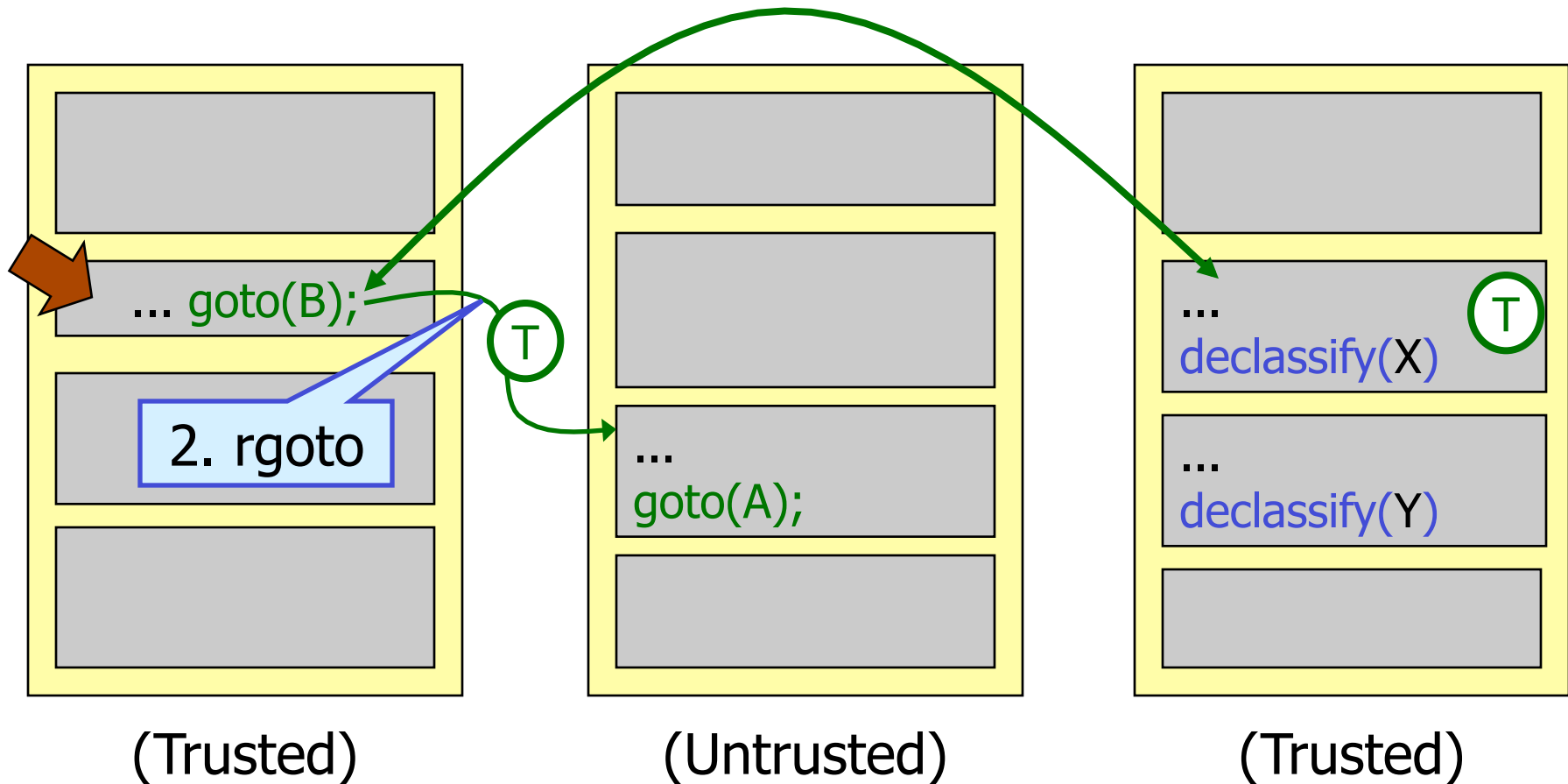
Control Transfer Integrity



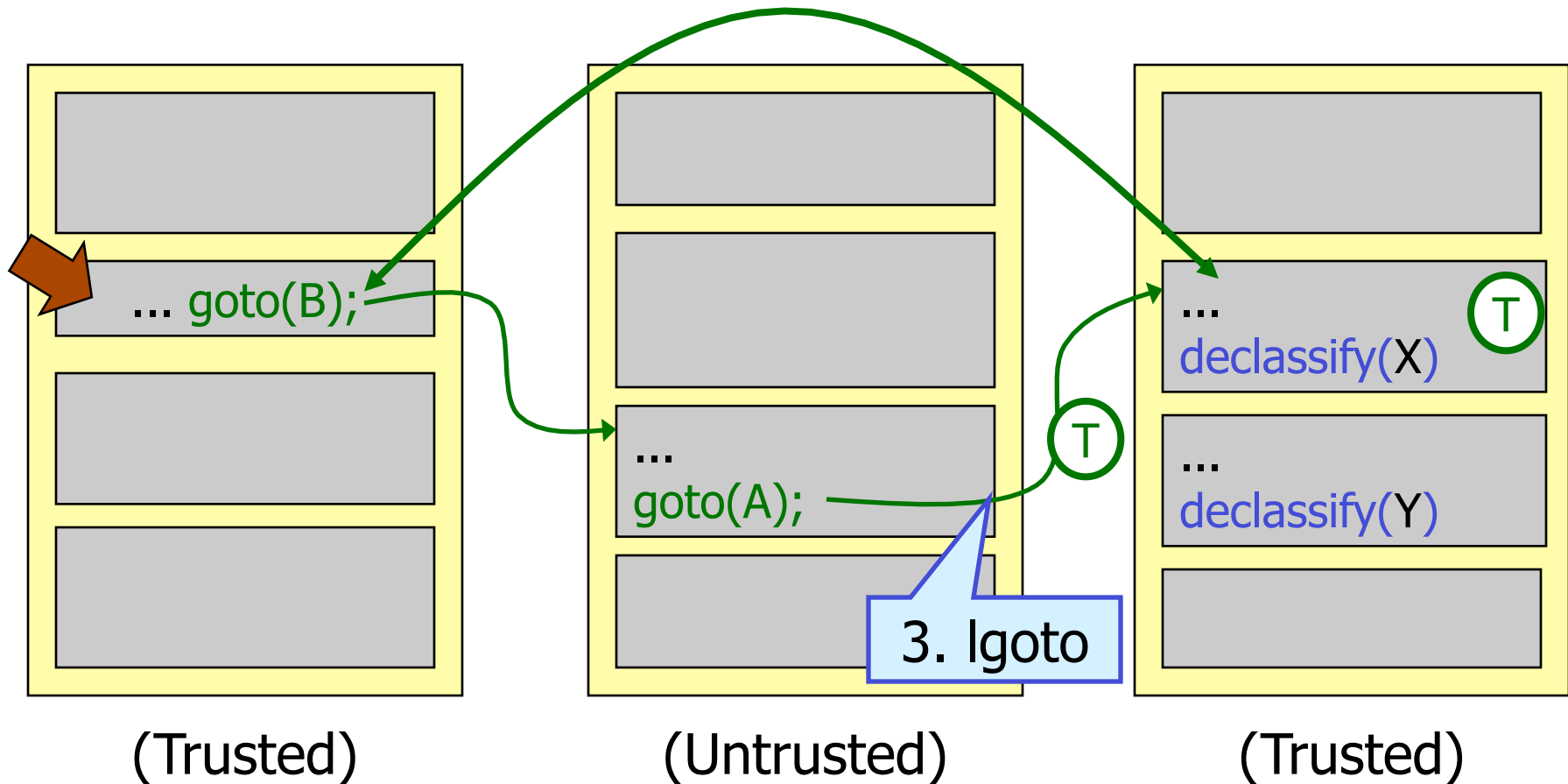
Control Transfer Integrity



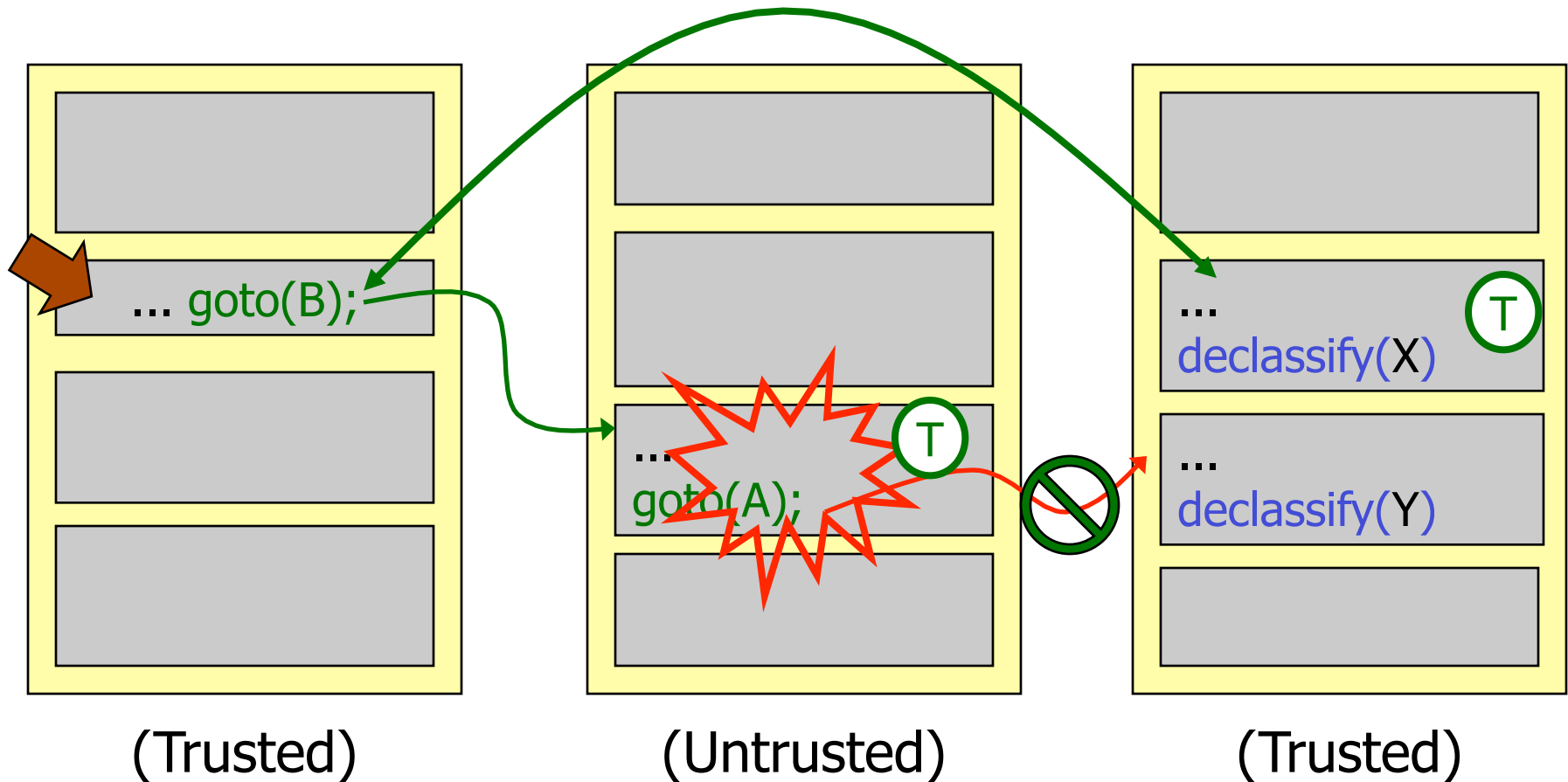
Control Transfer Integrity



Control Transfer Integrity



Control Transfer Integrity



Capabilities Protocol

- Theorem (Linearity):

When trusted hosts follow the protocol, there is at most one valid capability available to less trusted hosts.

Less trusted hosts have no choice about how privileges are restored to the computation.

Program partitioning

- Methodology:
 - programmers write explicit security policies
 - compiler/splitter transform code to satisfy them
- Needed Extensions: automatic replication
 - Easier to obtain integrity assurance
 - Useful for realistic computations
- Mechanisms to make it work:
 - One-way hashing with nonces
 - Capability tokens with secret splitting
 - Two-phase synchronization protocol
 - Consistent global identifier generation
- Results: greater assurance, shorter code, moderate performance impact

Results

- Implemented a variety of small programs in J_{IF} and used J_{IF}/split compiler to compile to distributed systems.
 - Battleship, three secure auction protocols, simple financial transactions, oblivious transfer
 - “Security-intensive”, mutual distrust
 - Integrity limitations prevented automatic partitioning of most by original system.
 - Implemented same programs with hand-crafted Java/RMI code.
 - J_{IF} versions are 13-65% shorter, but send 2-4× more messages (sometimes fewer).
-

Summary

- Methods are needed for obtaining system-wide security assurance in presence of mutual distrust.
- Whole computation can be the input, or just part
- JIF/split compiler *automatically* uses a variety of common techniques to solve secure compilation problems
 - encryption, digital signing, secure one-way hashing, nonces, agreement protocols, commitment protocols
 - Still more possibilities...
- Future work
 - Richer security policies
 - Language features
 - Experience

