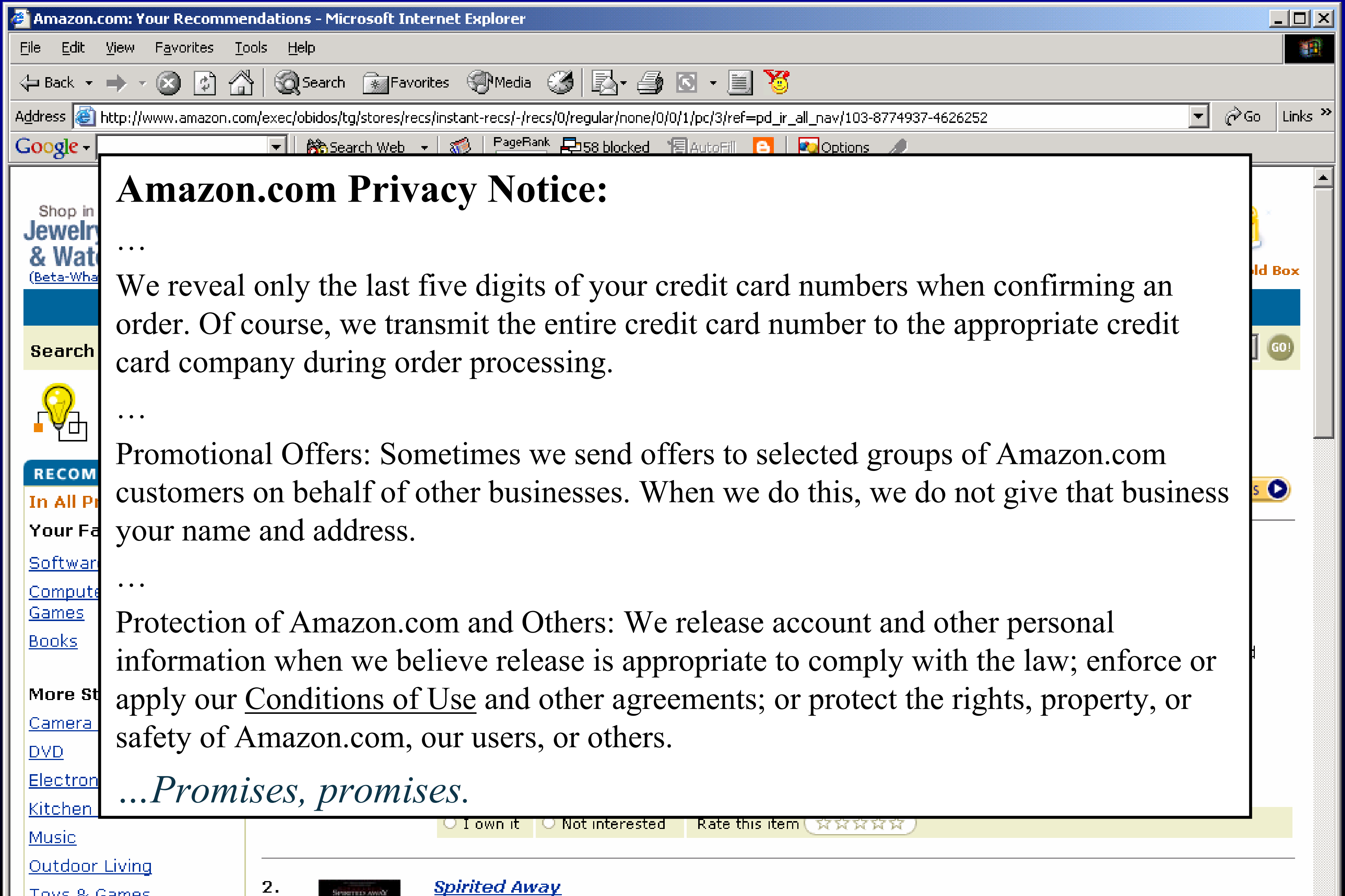
Making Distributed Systems Secure with Program Analysis and Transformation

Andrew Myers
Cornell University

Joint work with
Stephen Chong, Nate Nystrom,
Steve Zdancewic, Lantian Zheng

Information security



http://www.amazon.com/exec/obidos/tg/stores/recs/instant-recs/-/recs/0/regular/none/0/0/1/pc/3/ref=pd_ir_all_nav/103-8774937-4626252

Amazon.com Privacy Notice:

…

We reveal only the last five digits of your credit card numbers when confirming an order. Of course, we transmit the entire credit card number to the appropriate credit card company during order processing.

…

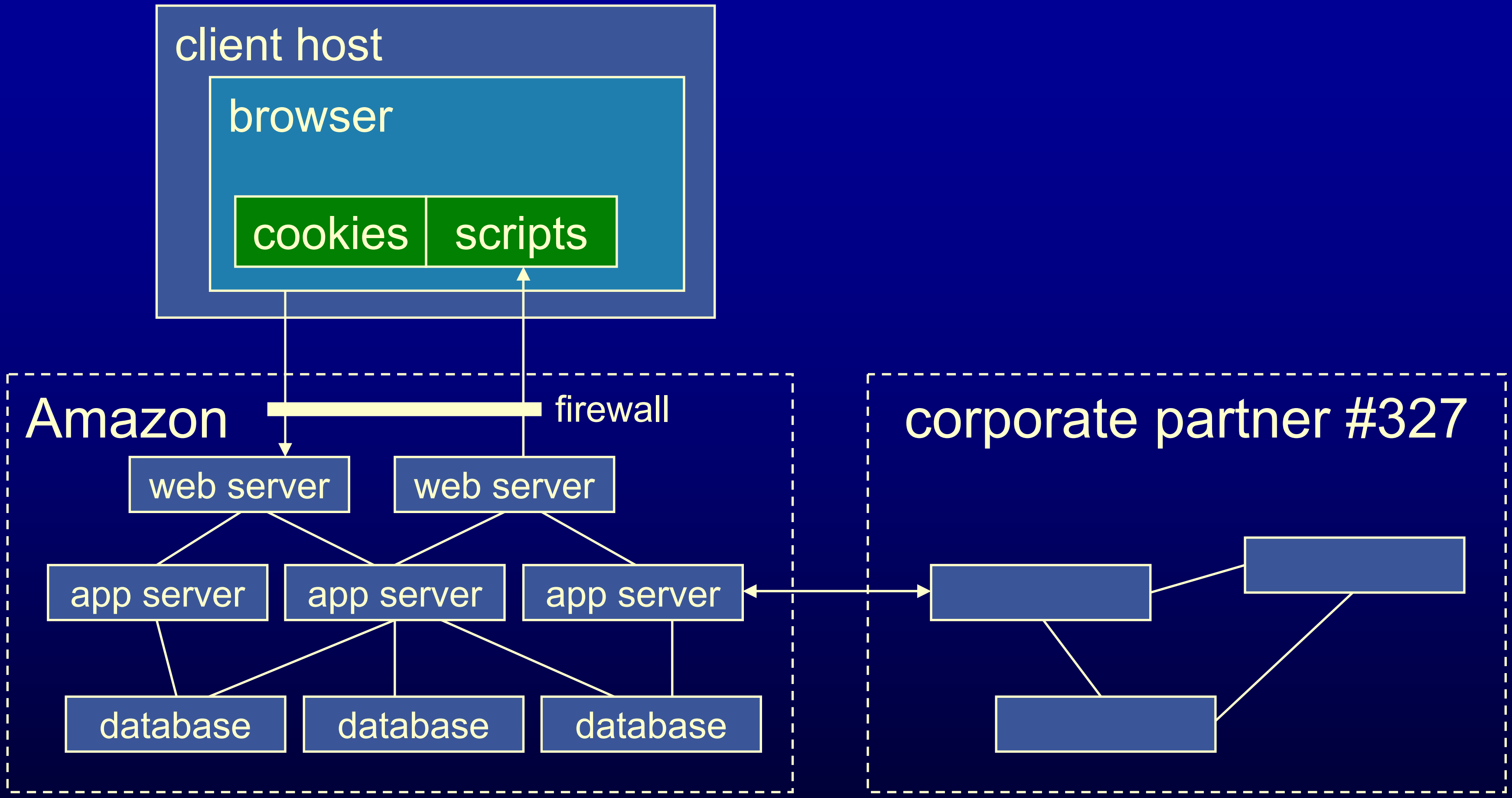
Promotional Offers: Sometimes we send offers to selected groups of Amazon.com customers on behalf of other businesses. When we do this, we do not give that business your name and address.

…

Protection of Amazon.com and Others: We release account and other personal information when we believe release is appropriate to comply with the law; enforce or apply our Conditions of Use and other agreements; or protect the rights, property, or safety of Amazon.com, our users, or others.

…Promises, promises.

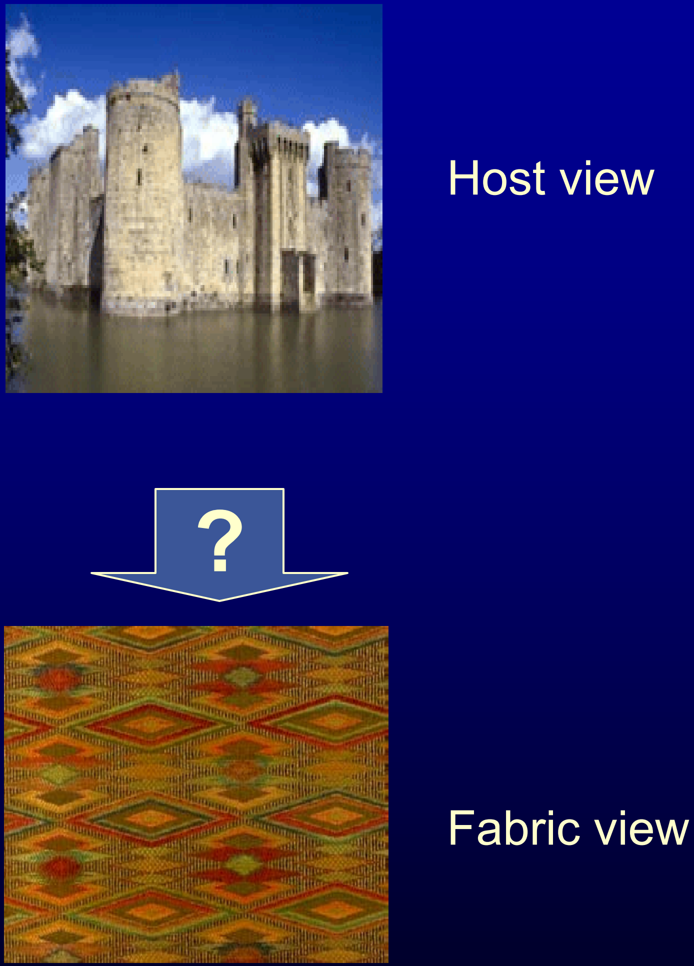
Possible implementation



Complex system -- how does Amazon know they are meeting their legal obligations?

Existing abstractions are defunct

- Old model: host devices running communicating programs
  - Host: a proxy for identity and privilege, data protection, persistent storage location
- Increasingly: pervasive networked devices (“fabric”)
  - Need to flexibly, adaptively map storage, computation onto available devices
  - Device perimeter no longer the right place to provide services, enforce system-level properties



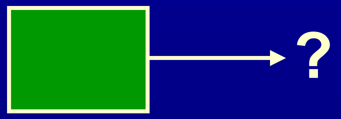
Secure distributed systems?

- How to build?
  - Encapsulation, access control lists, distributed protocols, encryption, signing,…
- How to validate?
  - Have analysis techniques only for individual mechanisms!

- Our goal: systems **secure by construction**
  - Programs annotated with explicit security policies
  - Compiler/static checker checks, transforms programs to satisfy policies

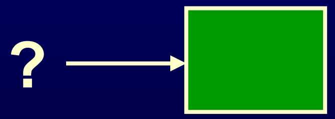
Information security properties

- Confidentiality (secrecy, privacy)
  - Making sure information isn't released improperly
  - Identify: information flows

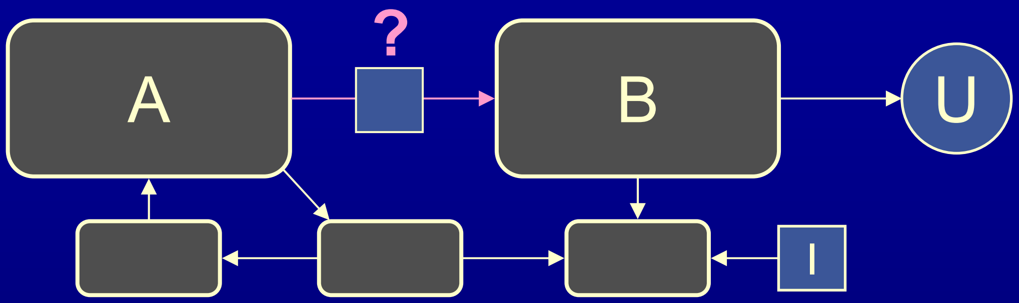


- Integrity
  - Making sure information only comes from the right places
  - Identify: dependencies

    = information flows



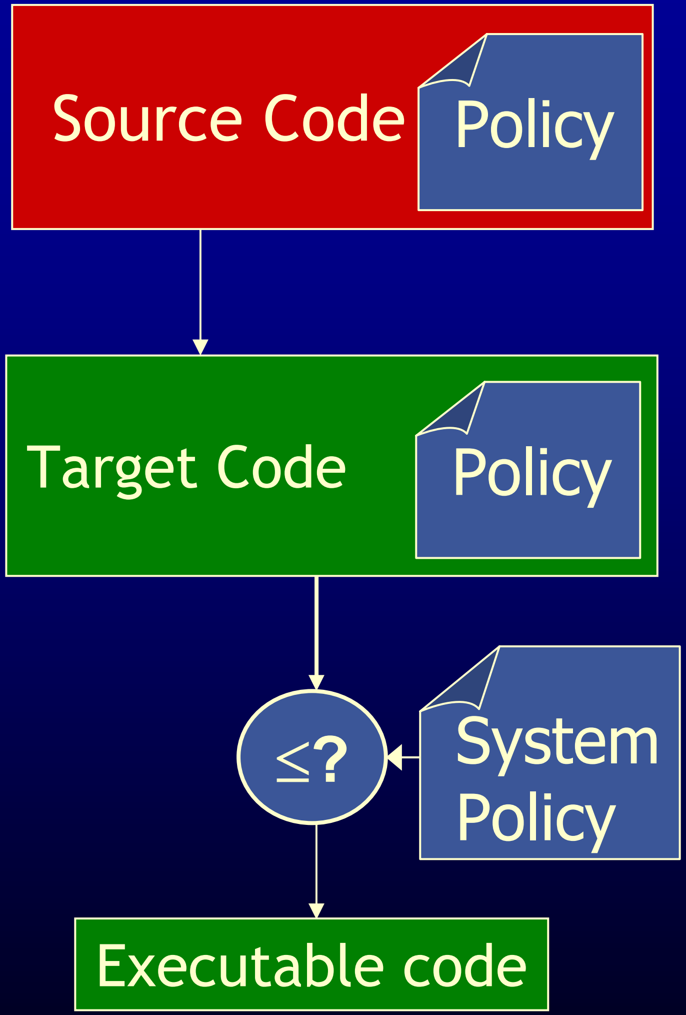
Policies vs. mechanisms



- Policy/mechanism mismatch
  - Conventional mechanisms (e.g., access control): control whether A is allowed to transmit to B
  - End-to-end confidentiality policy: information I can only be obtained by users U (no matter how it is transformed)
- How to map policy onto a mechanism?

Static information flow

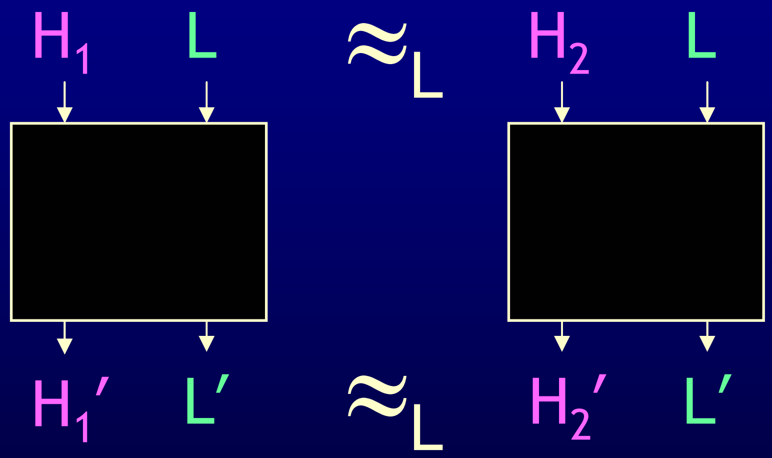
- Programs are annotated with information flow policies for confidentiality, integrity
- Compiler checks, possibly transforms program to ensure that all executions obey rules
- Loader, run-time validates program policy against system policies



Noninterference

"Low-security behavior of the program is not affected by any high-security data."

Goguen & Meseguer 1982



Confidentiality: high = confidential, low = public

Integrity: low = trusted, high = untrusted

Jif: Java + Information Flow

- Program types include security **labels**

```
        int{L} x;  // type of x is int{L}
```

- Compiler statically checks information flows

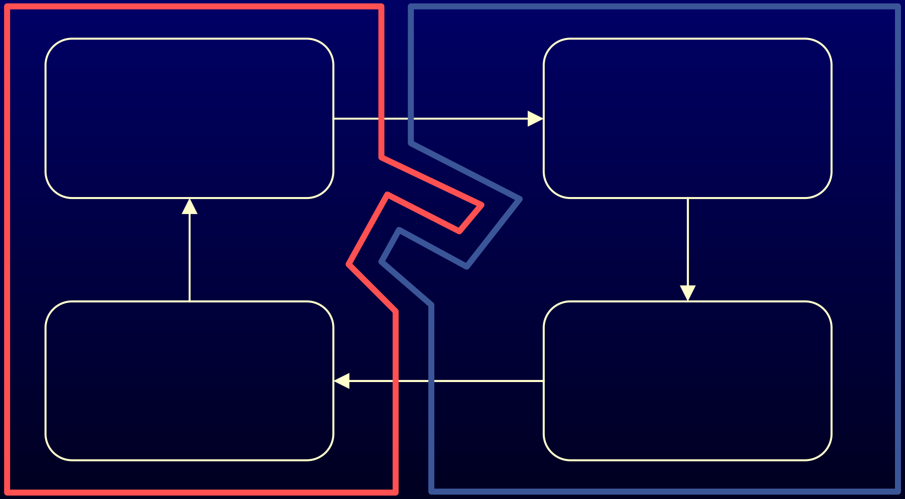
- Refinements:
  - Declassification and endorsement escape hatches
  - Label polymorphism
  - Parameterized types (on labels and principals)
  - Automatic label inference
  - First-class dynamic labels and principals
  - Static and dynamic access control
  - Application-defined authentication

- Publicly available:

  `http://www.cs.cornell.edu/jif`

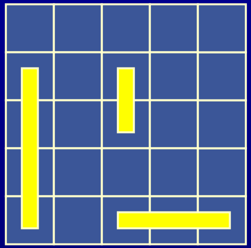
Type checking

- Static label checking is type checking in a security type system
- Decidable
- Little run-time overhead : labels erased
- Compositional!

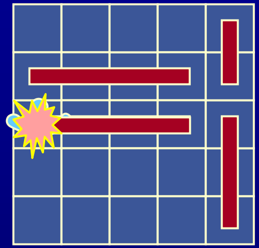


Distributed Battleship

- Two-player game in which each player tries to sink other’s ships



“A3”



“hit”

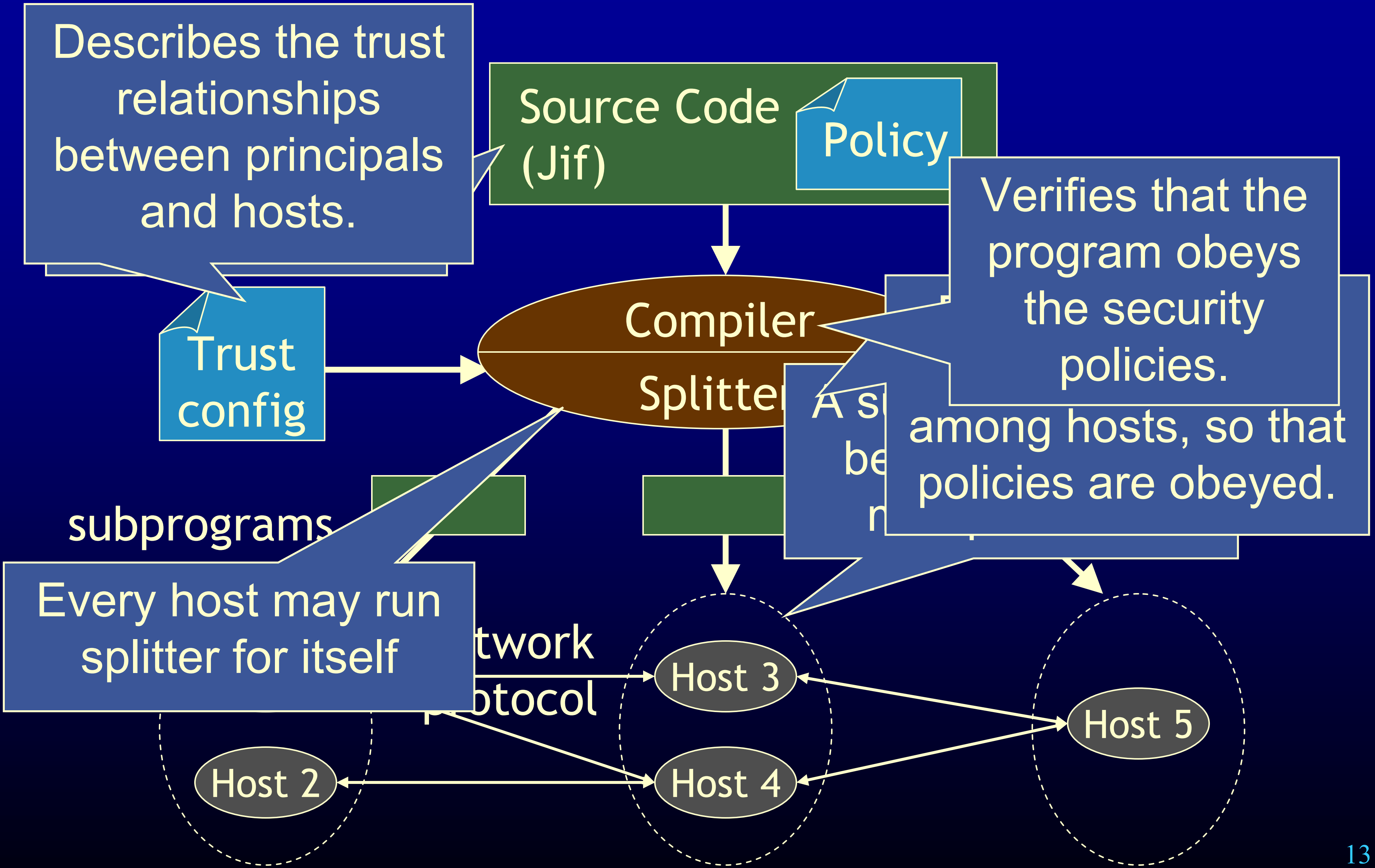
“you missed”

- General problem for multiplayer games/simulations: hard to prevent cheating

  Distrust ⇒ Multiplayer code must change.

- Idea: based on security types, compiler transforms code to run securely on untrusted hosts

Secure partitioning and replication



Describes the trust relationships between principals and hosts.

Verifies that the program obeys the security policies.

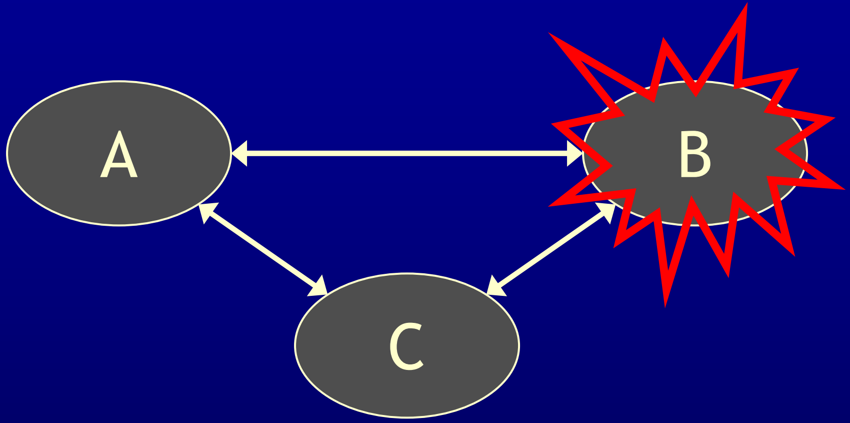
A s
be
n

among hosts, so that policies are obeyed.

Every host may run splitter for itself

Security for distrusting principals

- Principals vs. hosts



"Alice trusts hosts A & C"
"Bob trusts hosts B & C"

- Security guarantee: Principal P's security policy might be violated only if a host that P trusts fails

If B is subverted, Alice's policy is obeyed; Bob's policy might be violated.

Security policies in Jif/split

- Confidentiality labels:

```
int{Alice:} a1;
```

"a1 is Alice's private int"

- Integrity labels:

```
int{*:Alice} a2;
```

"Alice trusts a2"

- Combined labels:

```
int{Alice: ; *:Alice} a3;
```

(Both)

- Enforced in Jif language using static information flow analysis:

```
int{Alice:} a1, a2;
int{Bob:} b;
int{*:Alice} c;
```

Insecure

```
a1 = b;
b = a1;
c = a1;
```

Secure

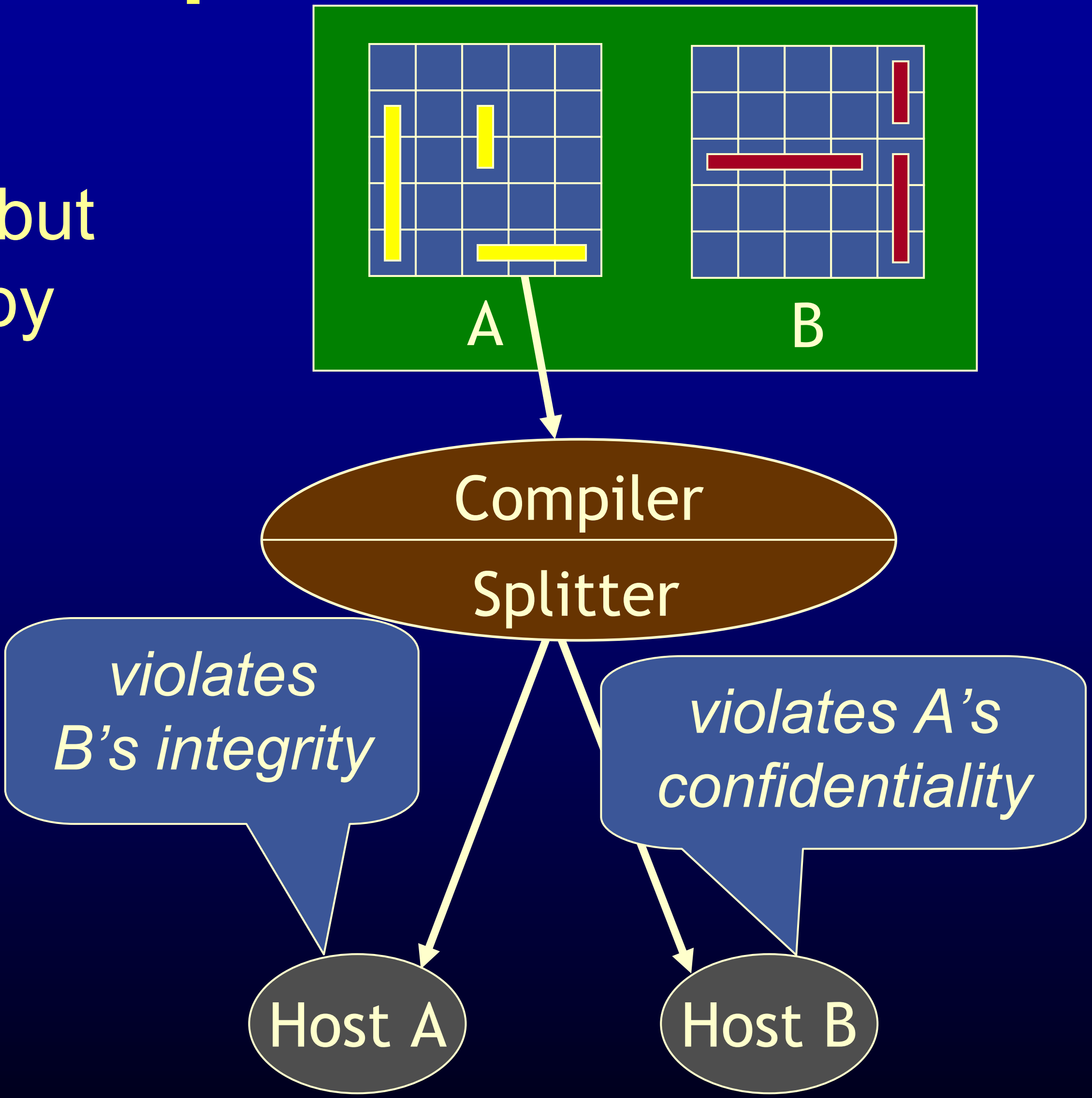
```
a1 = a2;
a1 = c;
```

Battleship example

- A’s board is confidential to A but must be trusted by both A and B:

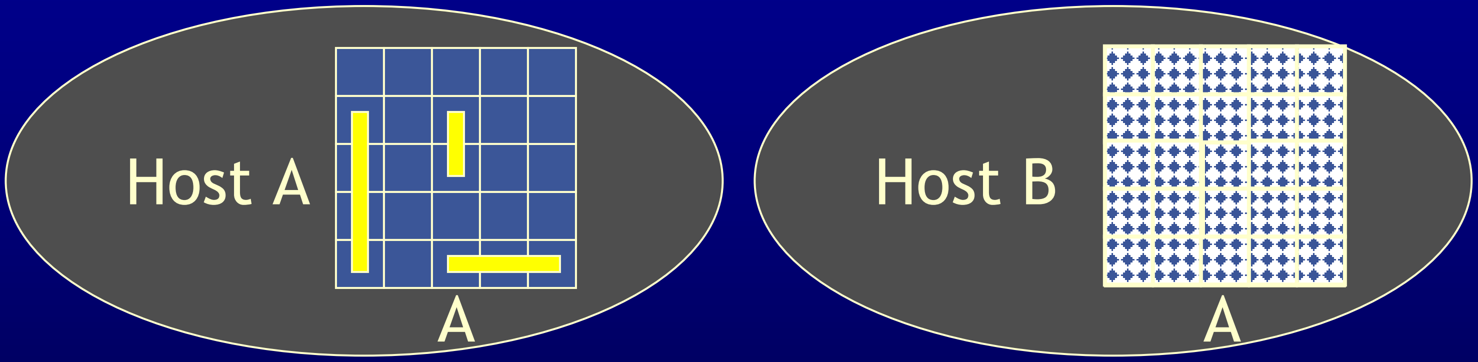
  `{A: ; *:A,B}`

- B’s board is symmetrical:
  `{B: ; *:A,B}`



Replication

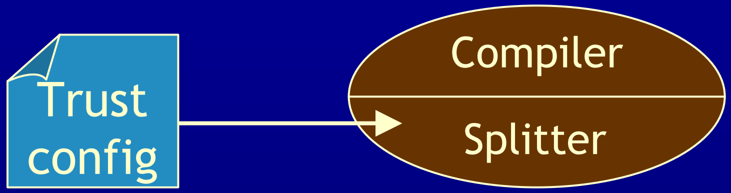
- Idea 1: replicate both boards onto *both* hosts so both principals trust the data.



- Problem: host B now has A's confidential data.
- Idea 2: host B stores a one-way hash of cells
  - Cleartext cells checked against hashed cells to provide assurance data is trusted by both A & B.
  - Compiler automatically generates this solution!

Host labels

- Trust in hosts described by *host labels*



- Battleship game:

`{A: ; *:A}` `{B: ; *:B}`

- Data with confidentiality *C* and integrity *I* can be securely placed on host *h* if:

$$C \sqsubseteq C_h \quad \text{and} \quad I_h \sqsubseteq I$$

- A's board: `{A: ; *:A,B}` but `{A:}`⋢`{B:}` and `{*:A}` ⋢ `{*:A,B}`

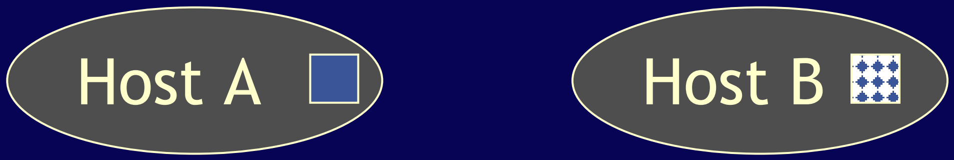
Secure replication condition

- Data with confidentiality *C*, integrity *I* can be securely placed on hosts h_i if:

$$C \sqsubseteq C_{h_j} \quad \text{for some host } h_j$$

$$\sqcap I_{h_i} \sqsubseteq I \quad (\text{instead of } I_h \sqsubseteq I)$$

Example *A*'s board: `{A:;*:A,B}`



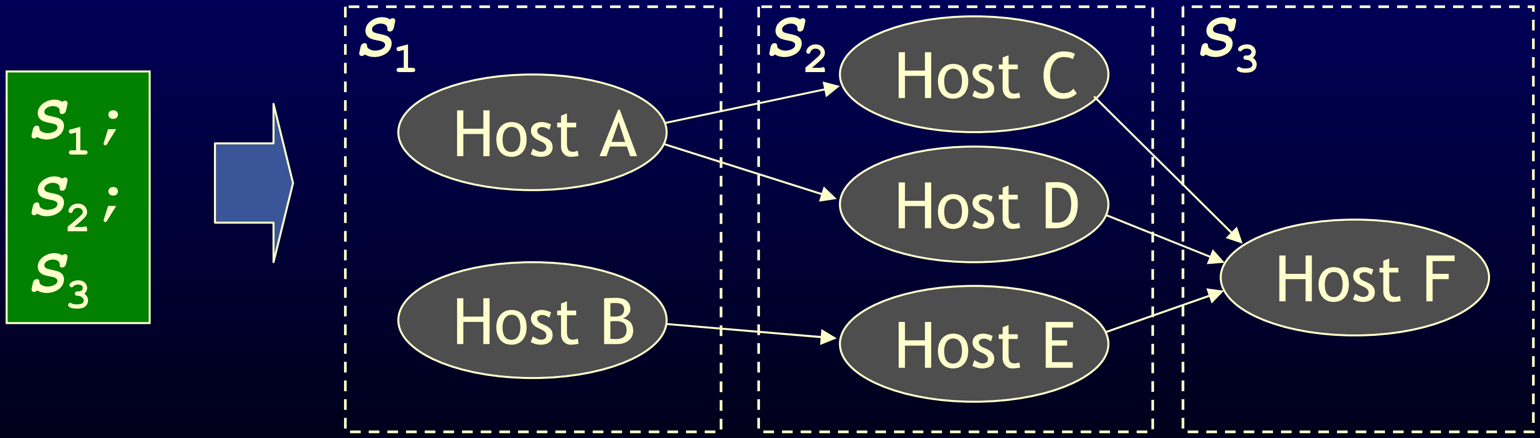
`{A:;*:A}` `{B:;*:B}`

Confidentiality: `{A:}`$\sqsubseteq$`{A:}`

Integrity: `{*:A}`$\sqcap$`{*:B}`$\sqsubseteq$`{*:A,B}`

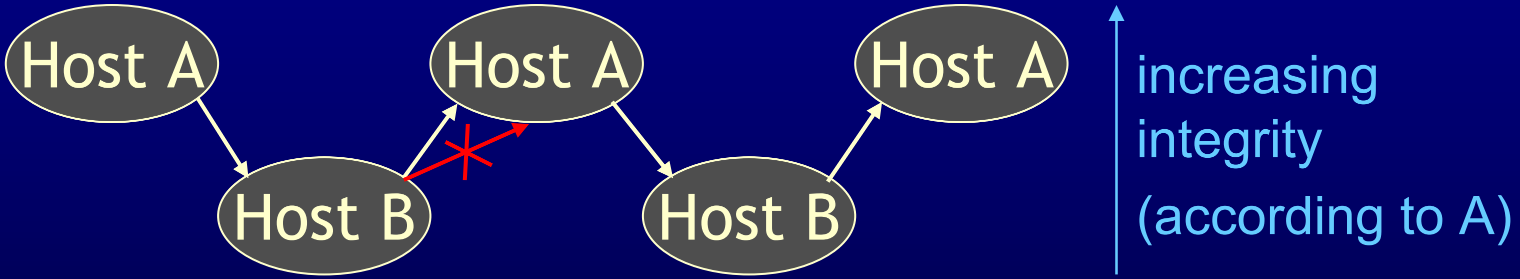
Replicating computation

- Replicated data ⇒ replicated computation
- Computation must be placed on hosts that are trusted to observe, produce data
- Control transfers in original program may become transfers among groups of hosts



Restoring integrity

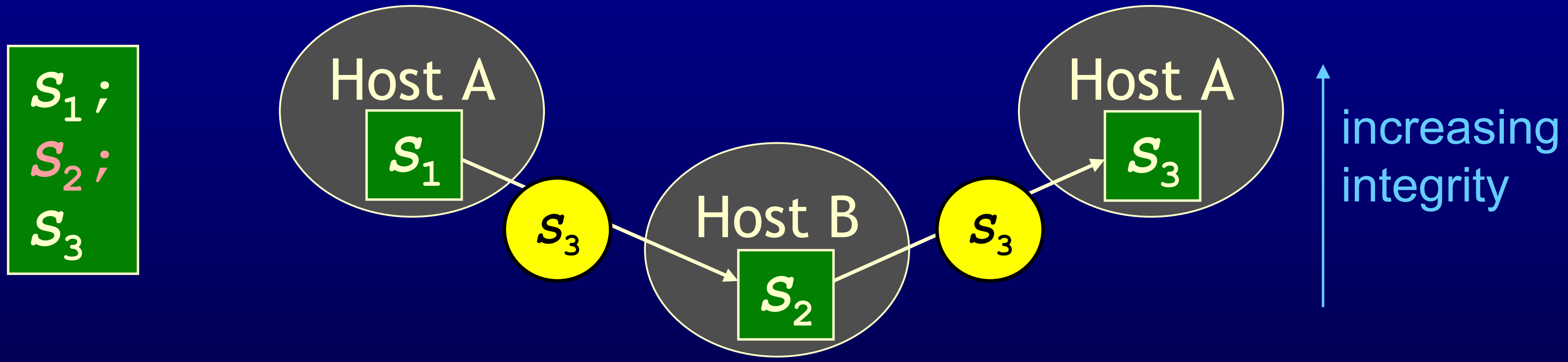
- Computation can transfer control between hosts with different integrity levels
- Battleship:



- How to prevent B from sabotaging integrity of computation with invalid invocations?
- Generally: how to prevent *group* of low-integrity hosts from sabotaging integrity?

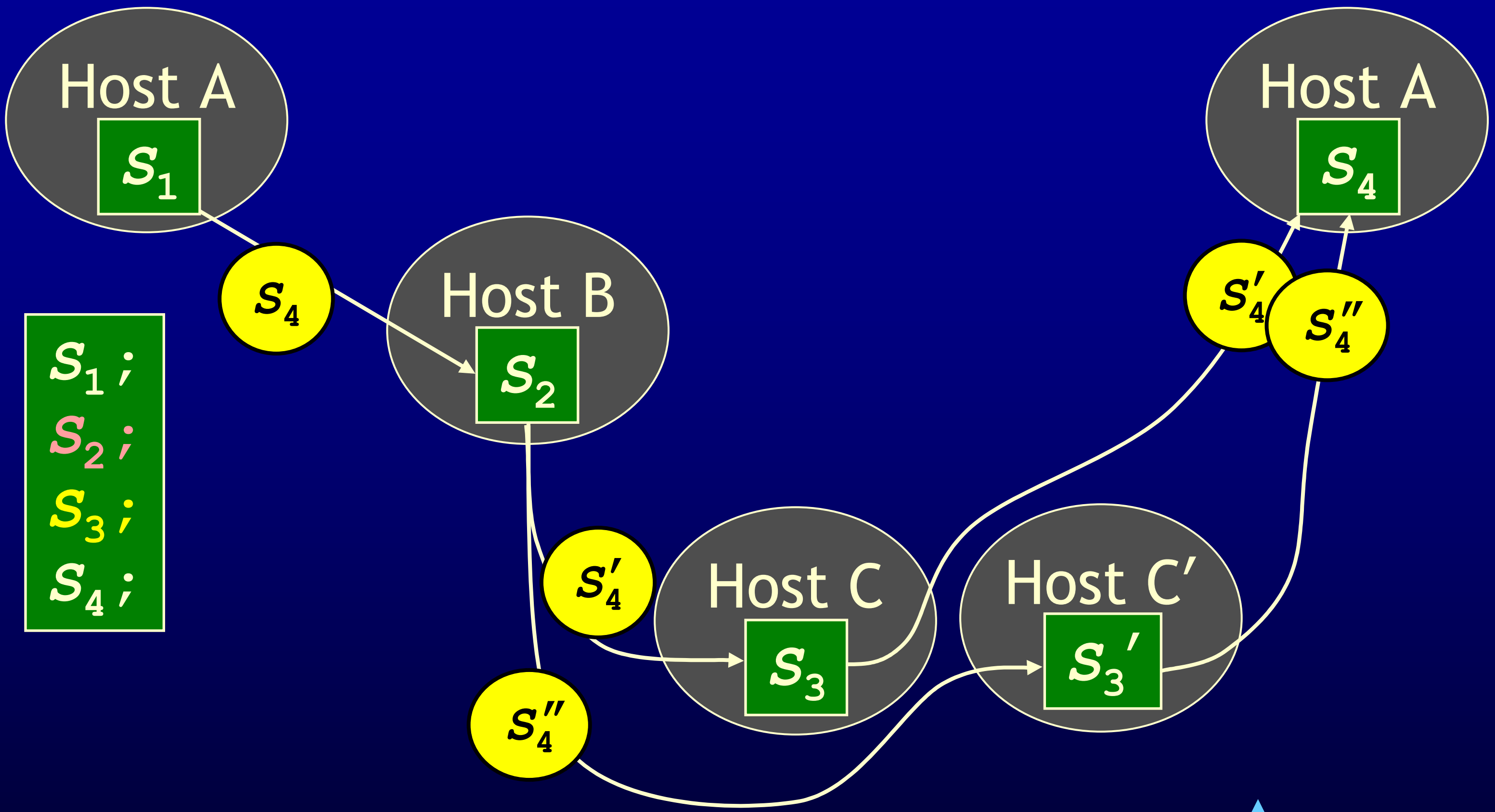
Capability tokens

- Solution: high-integrity hosts generate one-time **capability tokens** that low-integrity hosts use to return control



- At any given time, usable capabilities exist for at most *one* high-integrity program point
  - low-integrity hosts can't affect high-integrity execution

Splitting capability tokens



s_1 ;
s_2 ;
s_3 ;
s_4 ;

- Capabilities may be split into multiple tokens, recombined to return control.

increasing integrity (according to A)

Downgrading in Jif

Declassification (confidentiality)

```
int{Bob:; *:Alice} x;
y = declassify (x, {Bob:; *:Alice} to {*:Alice})
```

Endorsement (integrity)

```
int{Bob:} x;
y = endorse (x, {Bob:} to {Bob:; *:Alice})
```

- Unsafe escape hatch for richer confidentiality, integrity policies with intentional information flows
- Requires static authorization (access control)
- Requires `pc` integrity at downgrading point to ensure integrity of unsafe operations
  - Untrusted code cannot increase the information released: "Robust declassification" [CSFW01, CSFW04]

Downgrading in Battleship

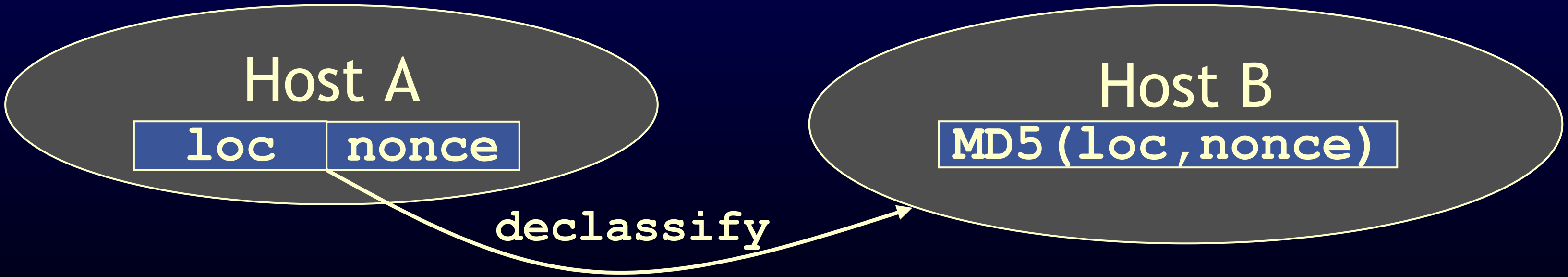
- **Declassification:** board location (i,j) not confidential once bomb dropped on it:

```
loc = declassify(board[move],
                 {A:; *:A,B} to {*:A,B})
```

- **Endorsement:** opponent can make any legal move, and can initially position ships wherever desired.

```
move = endorse(move_ , {*:B} to {*:A,B})
```

- `declassify`, `endorse` often correspond to network data transfers, hash value checks



Battleship main loop (simplified)

```
while (bobHits < NUM_SHIPS) {
  int aliceMove = alice.getNextMove();
  aliceHits +=
       bob.isHit(aliceMove)?1:0;
  if (aliceHits == NUM_SHIPS) break;
  int bobMove = bob.getNextMove();
  bobHits +=
       alice.isHit(bobMove)?1:0;
}
```

Unannotated isHit code

```
class PlayerAlice authority(Alice) {
  int[] public_board;
  int[] board;

  boolean isHit( int coord ) {
      public_board[coord] = board[coord];
      return public_board[coord] == SHIP;
  }
  …
}
```

Unannotated isHit (simplified)

```
class PlayerAlice authority(Alice) {
  int{*:Alice,Bob}[] {*:Alice,Bob} public_board;
  int{Alice:; *:Alice,Bob}[] {*:Alice,Bob} board;

  boolean{*:Alice,Bob} isHit(int coord) {
      public_board[coord] = board[coord];
      return public_board[coord] == SHIP;
  }
  …
}
```

Can’t assign from `{Alice:; *:Alice,Bob}` to `{*:Ali`ce,Bob}`

Battleship isHit code

```
class PlayerAlice authority(Alice) {
   int{*:Alice,Bob}[] {*:Alice,Bob} public_board;
   int{Alice:; *:Alice,Bob}[] {*:Alice,Bob} board;

   boolean{*:Alice,Bob} isHit(int coord)
       where authority(Alice) {
       public_board[coord] =
              declassify(board[coord],
                {Alice:; *:Alice,Bob} to {*:Alice,Bob});
       return public_board[coord] == SHIP;
   }
   …
}
```

Not enough integrity for declassification (`isHit`, `coord`)

Battleship isHit code

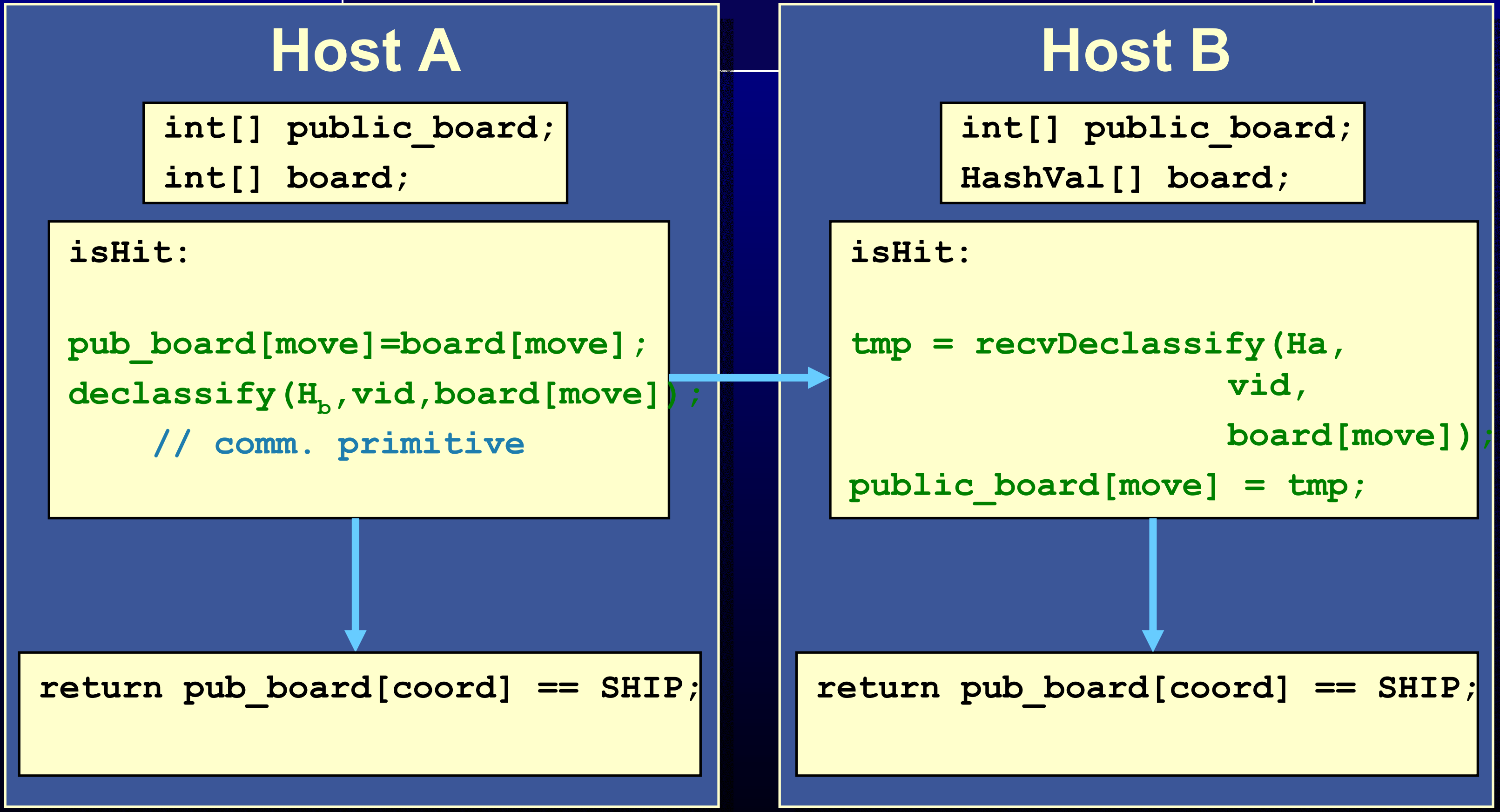
```
class PlayerAlice authority(Alice) {
   int{*:Alice,Bob}[] {*:Alice,Bob} public_board;
   int{Alice:; *:Alice,Bob}[] {*:Alice,Bob} board;

   boolean{*:Alice,Bob} isHit{*:Alice,Bob}
                       ({*:Alice,Bob} coord )
       where authority(Alice) {
       public_board[coord] =
             declassify(board[coord],
               {Alice:; *:Alice,Bob} to {*:Alice,Bob});
       return public_board[coord] == SHIP;
   }
   …
}
```

- Success!

Result

```
boolean{*:Alice,Bob} isHit{*:Alice,Bob}
    ({*:Alice,Bob} coord ) {
   public_board[coord] =
        declassify(board[coord],
        {Alice:; *:Alice,Bob} to
        {*:Alice,Bob});
   return public_board[coord] == SHIP;
}
```



Host A

```
int[] public_board;
int[] board;
```

```
isHit:

pub_board[move]=board[move];
declassify(H_b,vid,board[move]);
    // comm. primitive
```

```
return pub_board[coord] == SHIP;
```

Host B

```
int[] public_board;
HashVal[] board;
```

```
isHit:

tmp = recvDeclassify(Ha,
                     vid,
                     board[move]);
public_board[move] = tmp;
```

```
return pub_board[coord] == SHIP;
```

Experimental results

- Implemented a variety of small programs in JIF and used JIF/split compiler to compile to distributed systems.
  - Battleship, three secure auction protocols, simple financial transactions, oblivious transfer
  - Security-intensive, mutual distrust
  - Integrity constraints force use of replication
- Implemented same programs with hand-crafted Java/RMI code.
- JIF versions are 13-65% shorter, but send 2-4× more messages.

Related work

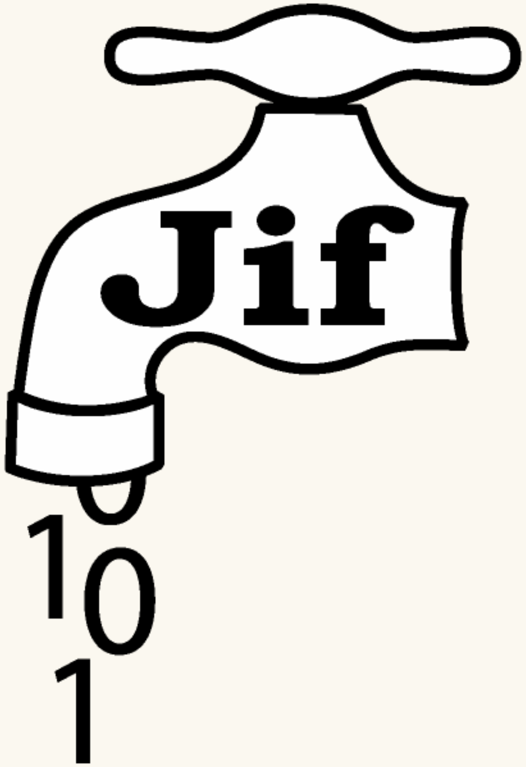
- Language-based security and static information flow (see [SM02])
  - mostly ignores distribution, distrust
- Multilevel security and information flow
- *Uniform* replication for improved integrity and availability
  - replicated state machines, BFT, file systems
- Stack Inspection
  - tries to protects downward control integrity
  - vulnerable to other integrity failures ("confused deputy")

Selected theoretical results

- Safety of capability token protocol [TOCS02]
- Noninterference for:
  - CPS language with state [HOSC02]
  - Simple language with dynamic labels [FAST04]
  - Concurrent language with secure message passing [CSFW03]
- Robust declassification property [CSFW01], proof that integrity check enforces it [CSFW04]

Conclusions

- Methods are needed for obtaining end-to-end assurance for distributed systems
- Information flow policies are a (the?) natural way to describe end-to-end information security
- JIF compiler provides a practical programming model while validating information flows
- JIF/split back end automatically uses a variety of common techniques to solve distributed security problems
  - Encryption, digital signing, secure one-way hashing, nonces, capabilities, access control, agreement protocols, commitment protocols
- Future work: incorporate more mechanisms, enforce richer security properties (e.g., availability)



http://www.cs.cornell.edu/jif