Verifying information security of dynamic, decentralized systems

Andrew Myers

IC3 retreat, May 2016

Cornell CIS
Computer Science

IC3

Where abstractions grew up

- Code ran in a friendly environment
- Language and library abstractions were designed for that environment

The modern environment

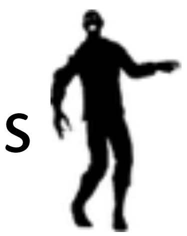
- Platforms, data, other computations controlled by adversaries
- Adversaries trying to learn secrets, corrupt results
- The old abstractions are fundamentally broken

Whither ad-hoc security?

Environment is only getting tougher:

untrusted mobile code

cloud computing platforms



federated systems with heterogeneous trust

side channel attacks

blockchain computations

Patching the old code & abstractions?

Doomed.

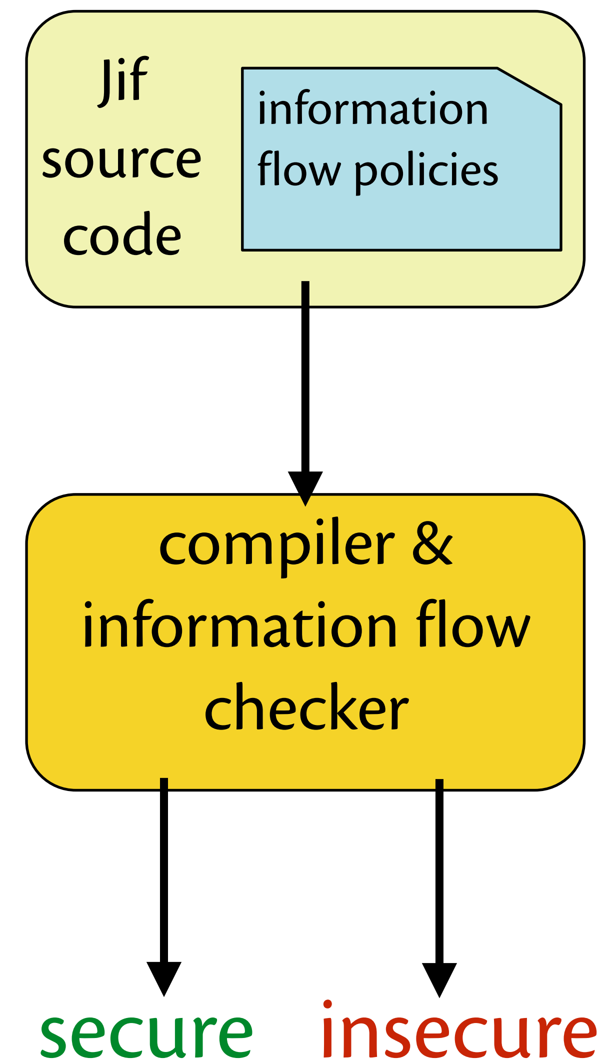
(and your legacy code is going to be toast anyway)

New abstractions to build code

secure by construction

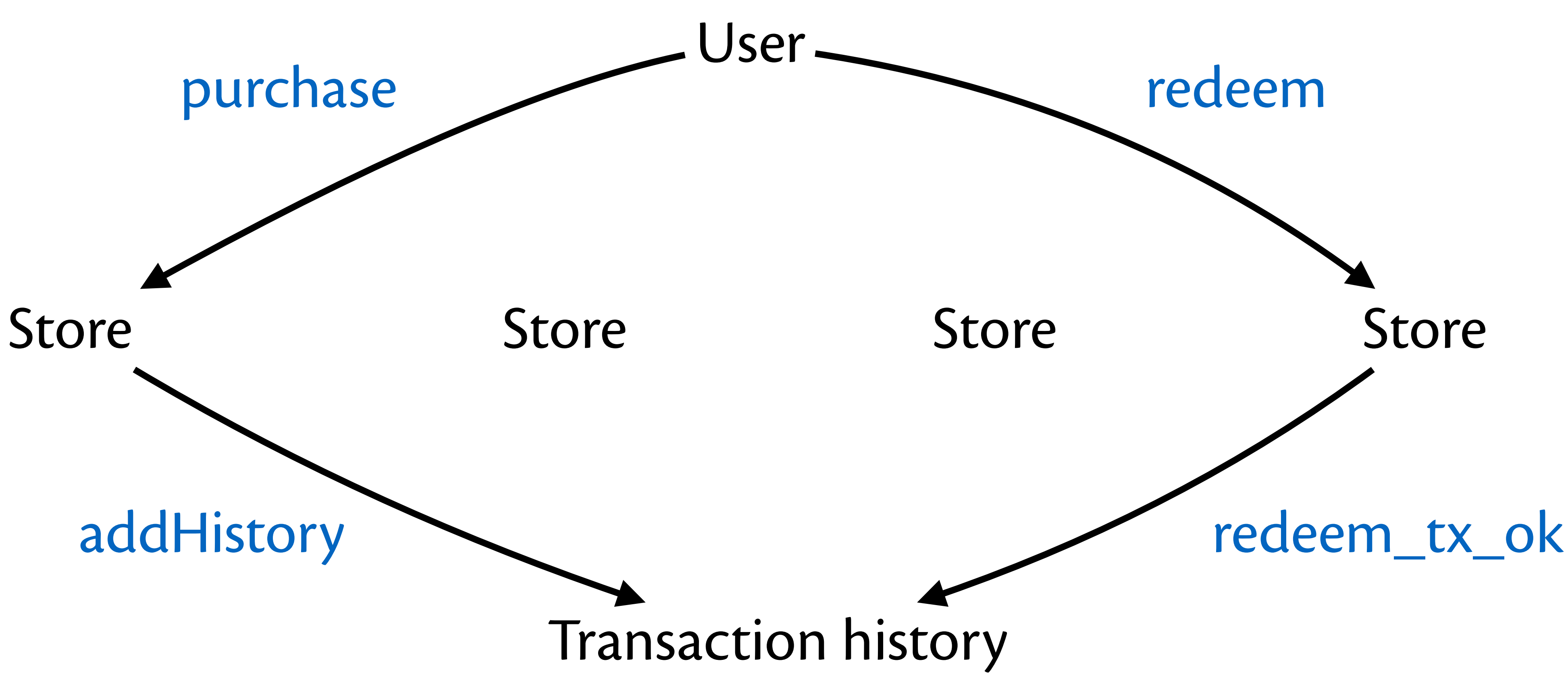
Language-based information flow

- Goals: **confidentiality** (secrecy, privacy) and **integrity** (trustworthiness)
- Idea: attach policies for confidentiality and integrity as **security types**
- Check **before code runs** that all flows of information satisfy policies. Proves:
  - code cannot leak information except through intended channels (confidentiality)
  - code actions cannot be corrupted except by explicitly allowed influence (integrity)



An example

- Multi-store joint loyalty program: loyal customers receive coupons they can redeem
- Security goals:
  customer purchases are private,
  customers can't fake purchases to get coupons



Example: Jif programming

- Principals: **U** = user (customer), S = store, L = loyalty program
- Labels: **U←** = trusted by user, **S→** = secret to store, **A⊓B** = flows to A & B

```
void purchase{U←}(User U, Transaction t, Store s)
where default label U←∧U→∧S→, authority S {
   …
}
```

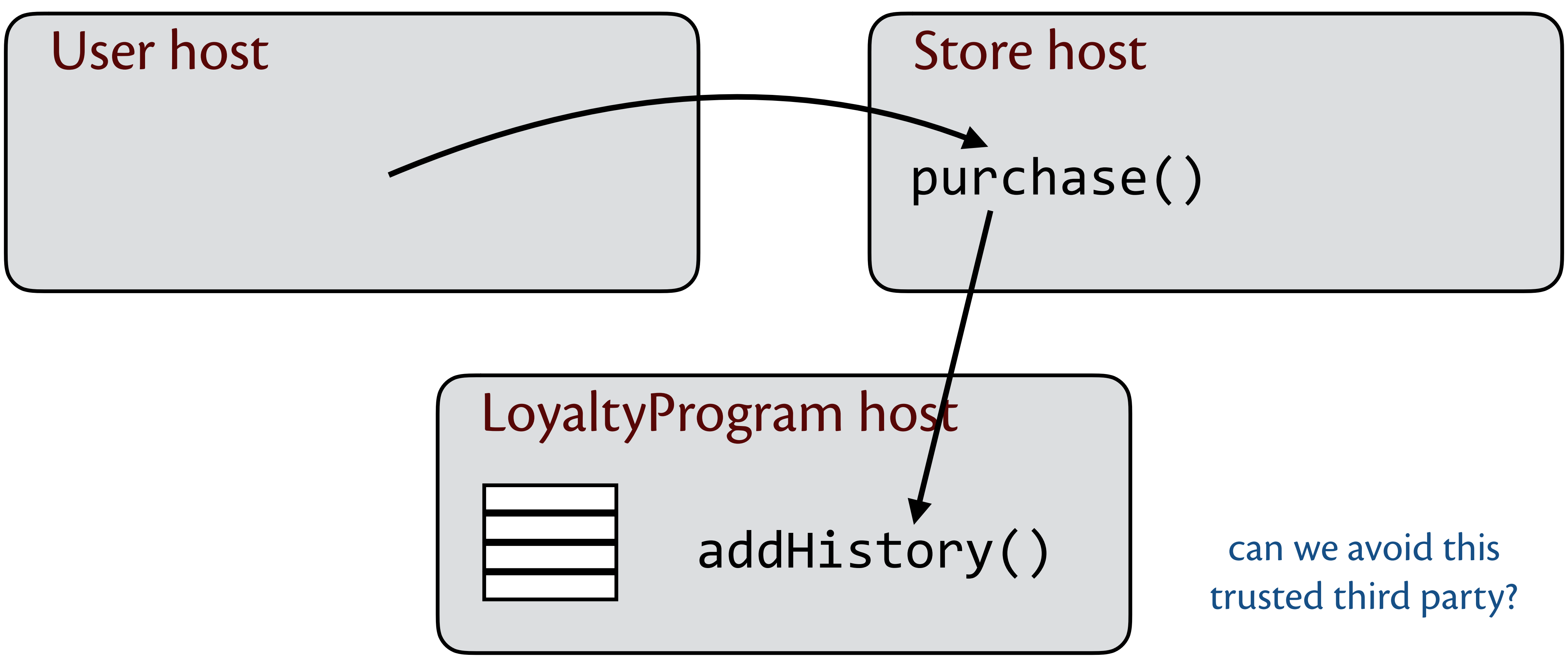
purchase only caused by U

unlabeled parameters U, t, s are trusted by U, U and store S should see transaction

code trusted by S

Secure distributed programming

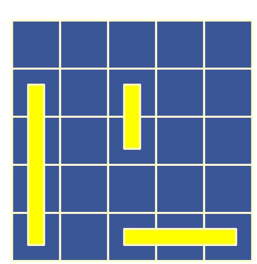
- **Fabric:** using information flow to build systems using distributed services from different providers
  - end-to-end secure information flow across network
  - securely combines code, data across trust boundaries



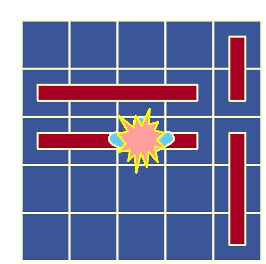
can we avoid this trusted third party?

Protocol synthesis

- Information flow policies can guide automatic synthesis of cryptographic protocols
- Example: Battleship (from Jif/split)
  - A doesn't trust B's computer or vice versa
  - Confidentiality: A doesn't want B to see his board (and vice versa) $\mathbf{A}^{\rightarrow}$
  - Integrity: A doesn't want B to corrupt either board (and vice versa) $(\mathbf{A} \wedge \mathbf{B})^{\leftarrow}$



"C3"

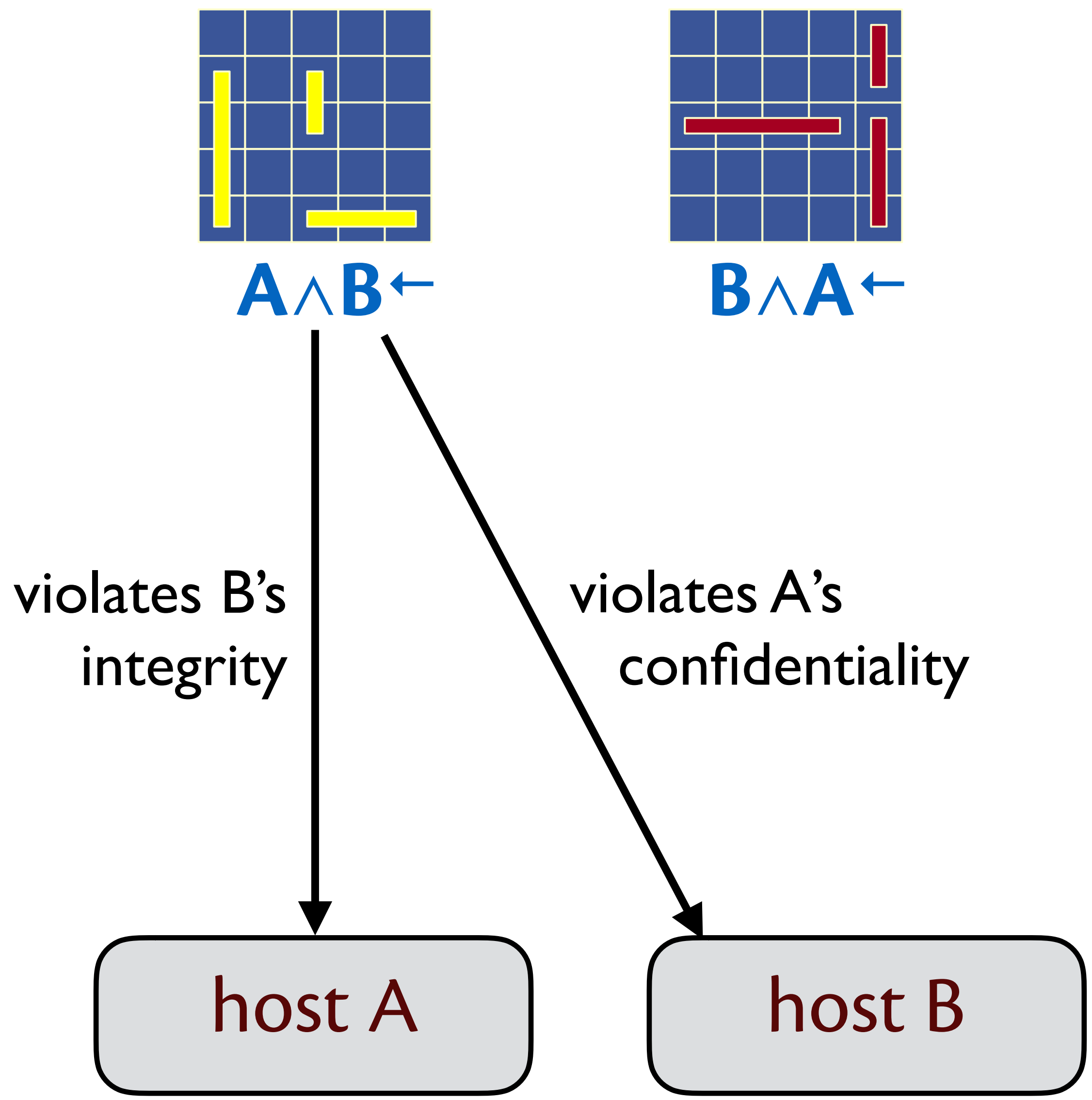


"you missed"

"hit"

Unsolvable constraint?

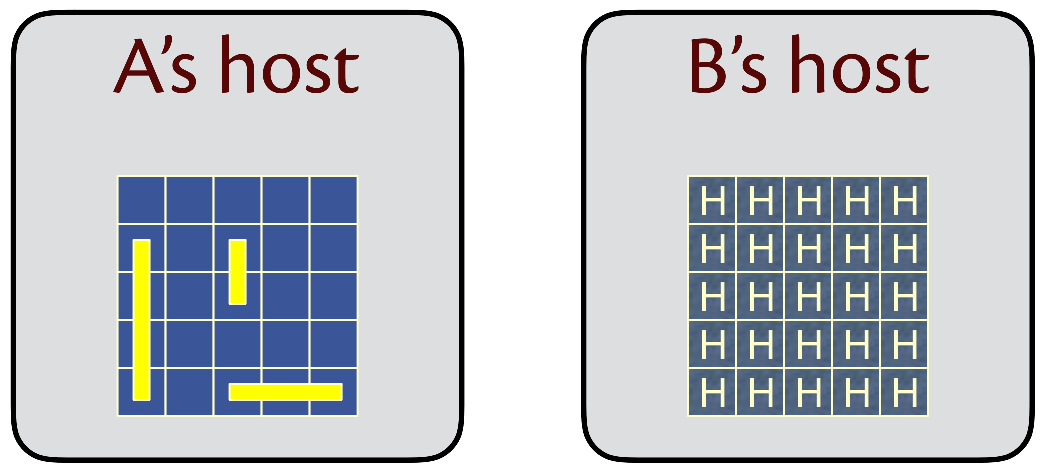
- Label on A's board: **A∧B←**



Need trusted third party?

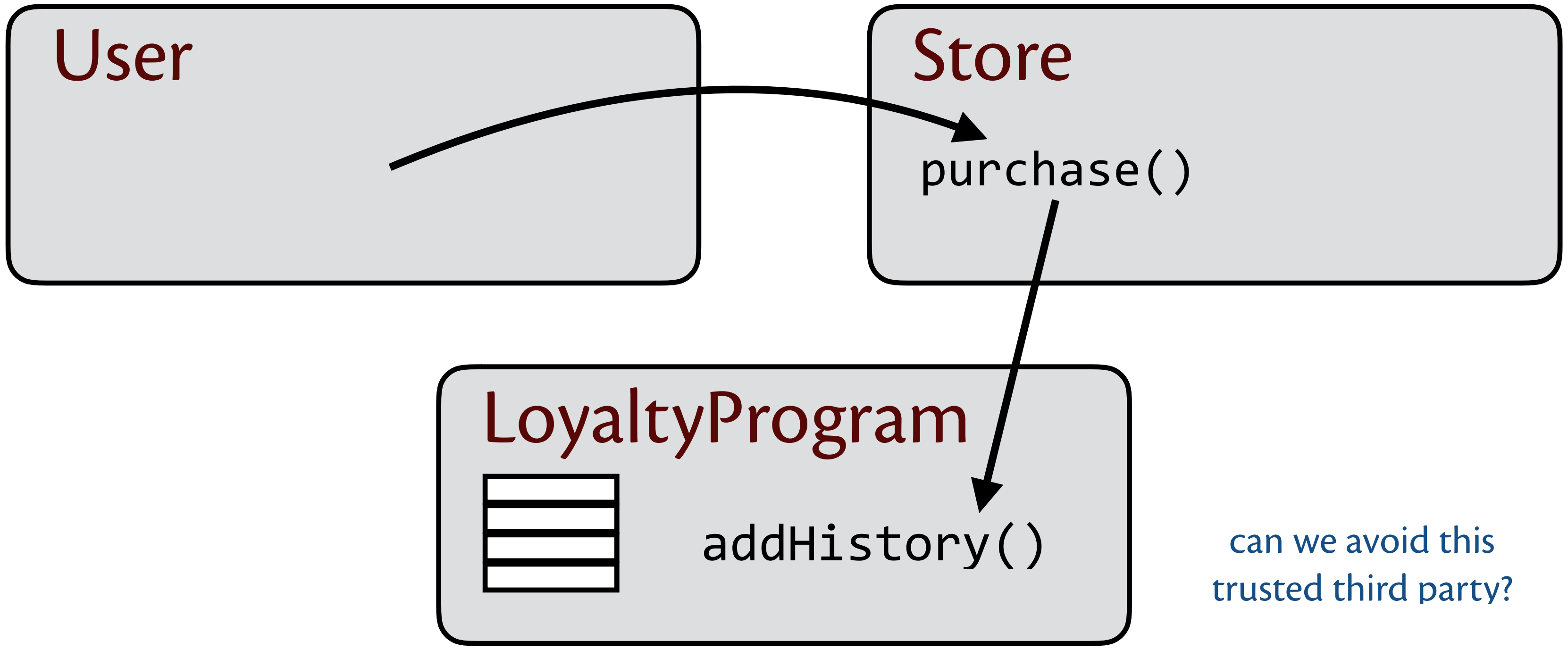
Solution: commitment

1. compiler replicates both boards onto *both* hosts to enforce *integrity* of A and B (**A**$^{\leftarrow}\wedge$**B**$^{\leftarrow}$)



2. To enforce confidentiality of A (**A**$^{\rightarrow}$), store on B only a *hash* of the board data with a random nonce
    1. Cleartext cells checked against hashed cells to provide assurance data is trusted by both A & B.
    2. Jif/split compiler **automatically** generates this commitment protocol!

Ongoing: synthesizing more complex protocols

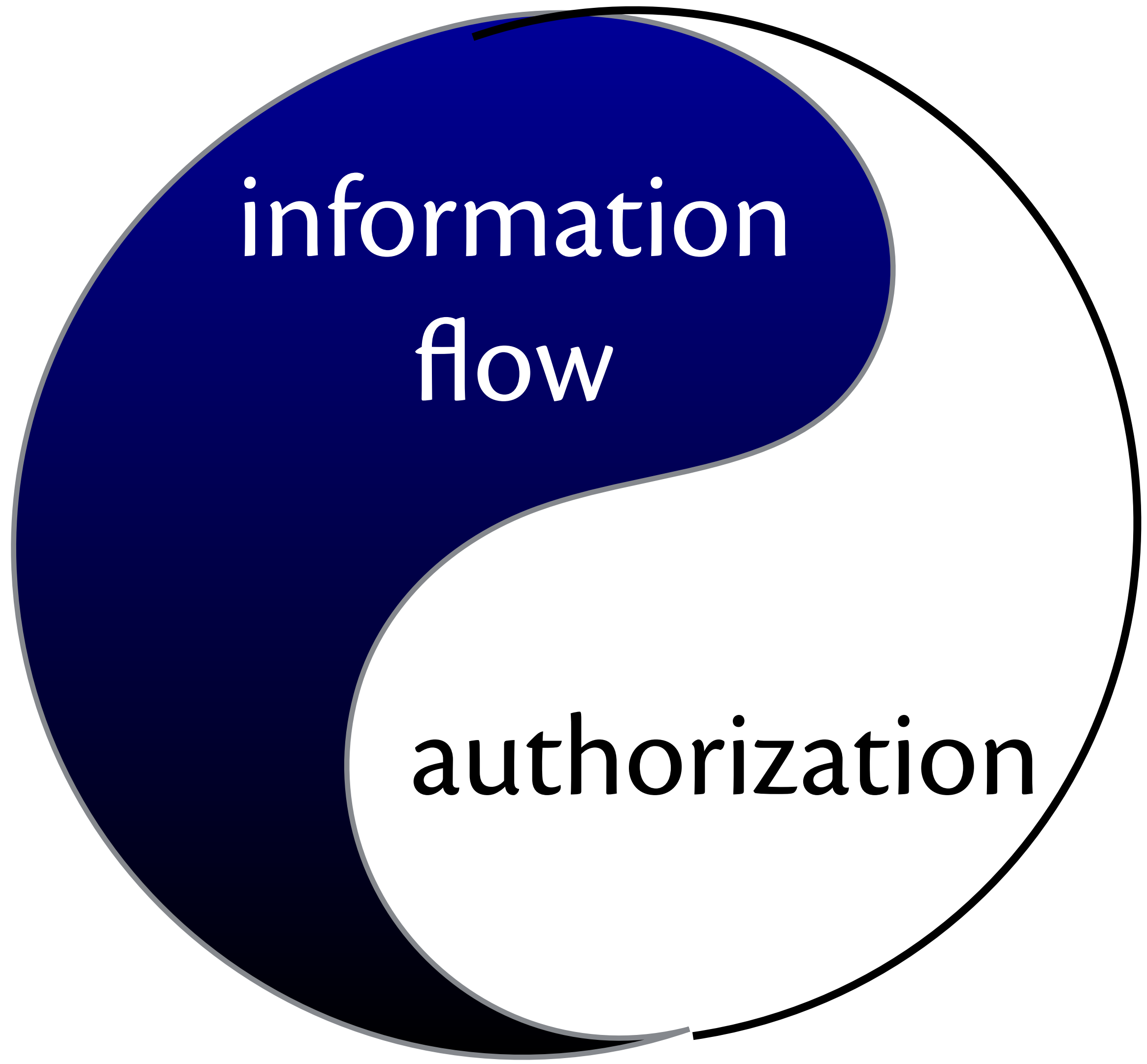


can we avoid this trusted third party?

- Insight: Loyalty program can be implemented using blockchain as “trusted third party” and zero-knowledge proofs
- Current work:
  - automatically partitioning code and data into secure cryptographic implementations using blockchain.
  - For performance: keep computation, storage off blockchain when possible.

Dynamic policies and dynamic trust

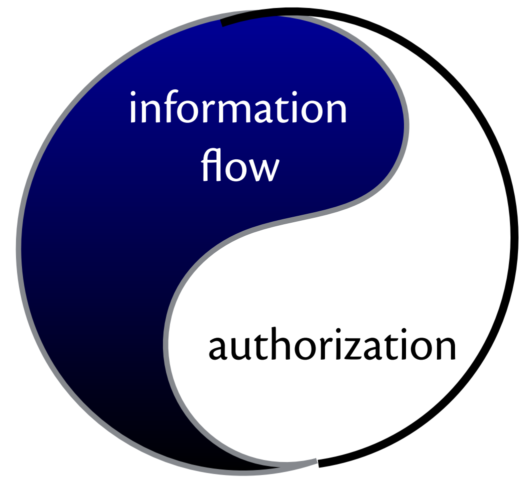
Two classic formal models for security policies



Answering different questions:

- information flow: *where can information go?*
- authorization/access control: *who is trusted to perform actions?*

Interactions between models



- Information flow needs authorization: *who is trusted to see information?*
  - Classic ideas of secure information flow break when trust can change!
- Authorization needs information flow: *who may learn about and affect trust?*
  - Authorization mechanisms create possibly insecure flows of information!

FLAM: a unified model

- The **F**low-**L**imited **A**uthorization **M**odel **[CSF'15, '16]**
  - A novel **principal model** unifying authorization and information flow control (notation used in this talk)
  - A **logic** for making **distributed**, **decentralized** decisions about authorization and information flow (Security properties verified in Coq)
  - A **programming language** (FLAC) for building authorization mechanisms securely.
- A programming model for smart contracts?

Security by construction

- Information flow policies offer a new kind of abstraction for building secure code
- Power of the adversary is explicit

  ⇒ compiler can check security,

  partition and replicate code and data,

  automatically deploy cryptography