A high-level abstraction for Securely Programming the Internet

www.cs.cornell.edu/projects/fabric


Jed Liu · Owen Arden · Tom Magrino · Andrew Myers

Decentralized sharing

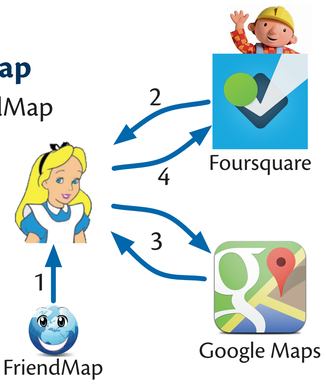
Goal

An open, decentralized platform that makes it easy to build secure software that integrates code and data across organizational and trust boundaries

Example application: FriendMap

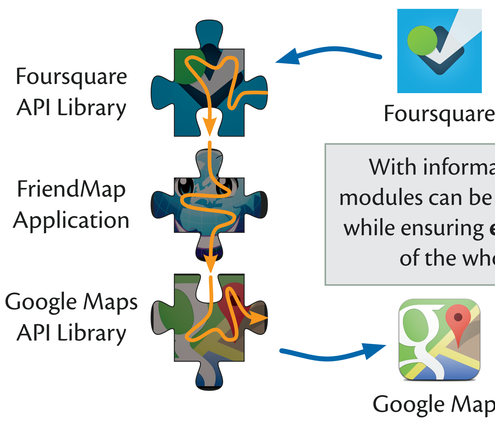
1. Alice downloads and runs FriendMap
2. Fetch friends' locations
3. Get a map and place pins for nearby friends
4. Post to network

Must ensure code running on Alice's client doesn't leak Bob's information inappropriately.



Need composable decentralized security

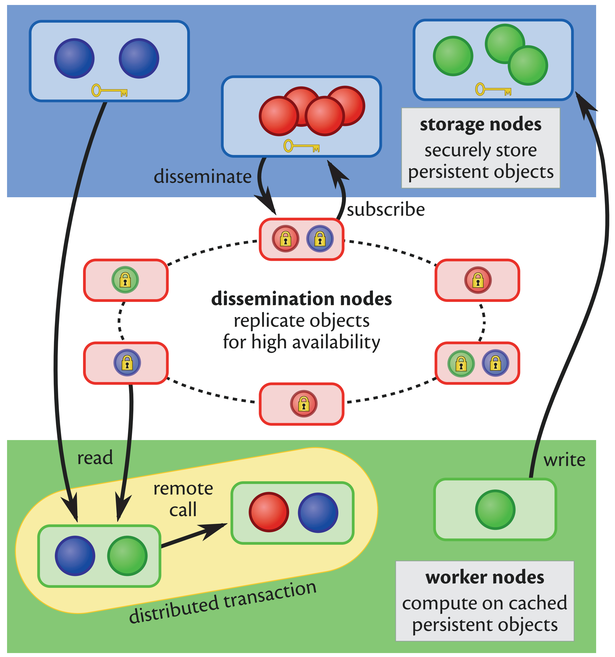
- Access control is inadequate
  - Only controls **who** is allowed to see information
  - Doesn't constrain **what** is done with it
- **Information flow control** is inherently compositional
  - Rejects **unsafe flows** instead of users



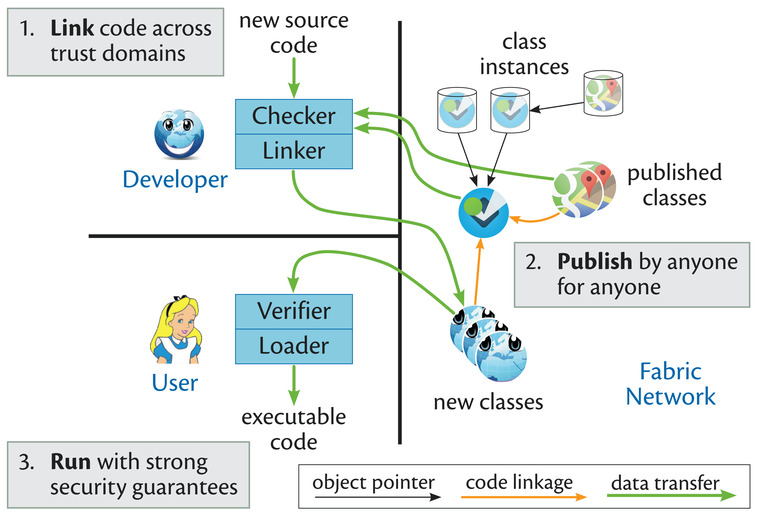
With information flow control, modules can be **checked separately**, while ensuring **end-to-end security** of the whole application

System model

System architecture

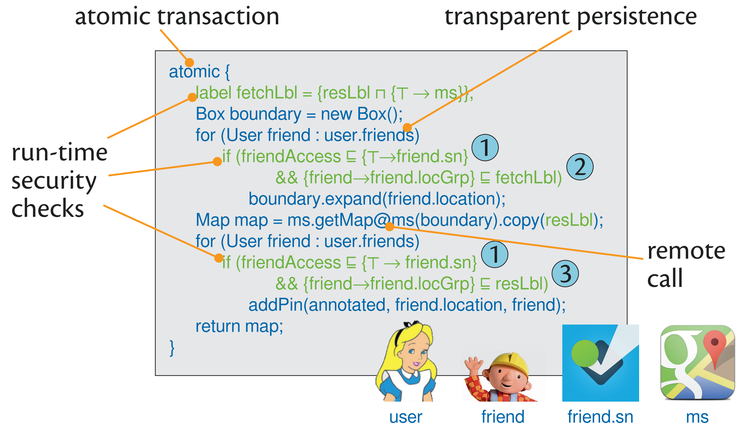


Software lifecycle



Programmability & Security

Code example



Security checks **forced by compiler:**

1. Querying social network doesn't leak user's or friend's information.
2. Friend allows map service to learn location.
3. Friend allows location to be in posted result.

Decentralized security principle

You can't be harmed by what you don't trust

Decentralized security labels

- Base policies: confidentiality ($o \rightarrow r$) and integrity ($o \leftarrow w$)
- Label ordering $\sqsubseteq$ specifies allowed information flows
- Information flows can be checked statically:
  - for statement $y=x$, compiler checks $L_x \sqsubseteq L_y$