Don't Trust Your Roommate, or, *Access Control and Replication in "Home" Environments*

HotStorage 2012

Vassilios Lekakis, Yunus Basagalar, Pete Keleher
University of Maryland

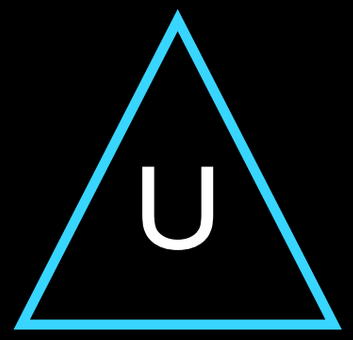
One picture is worth a thousand words

Anzere, Bayou, BlueFS, Cimbiosys, EnsemBlue, EYO, HomeViews, Perspective, PodBase, PRACTI, UIA, ZZFS

Building Blocks

Replication

Consistency

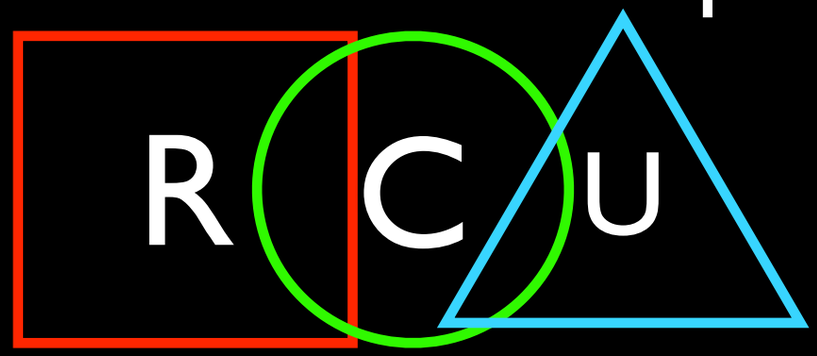


Update mechanism

Building Blocks

Replication

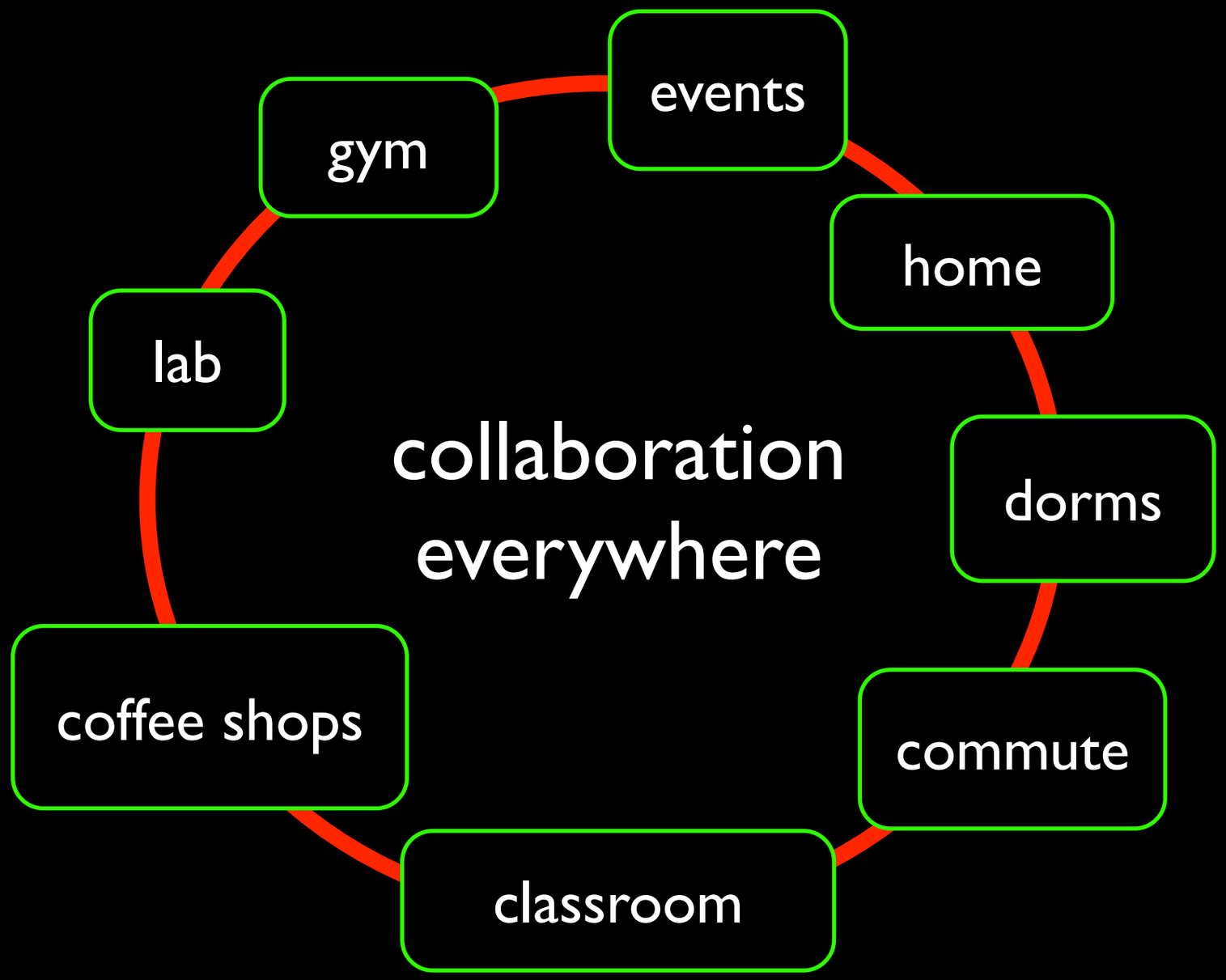
Update Mechanism



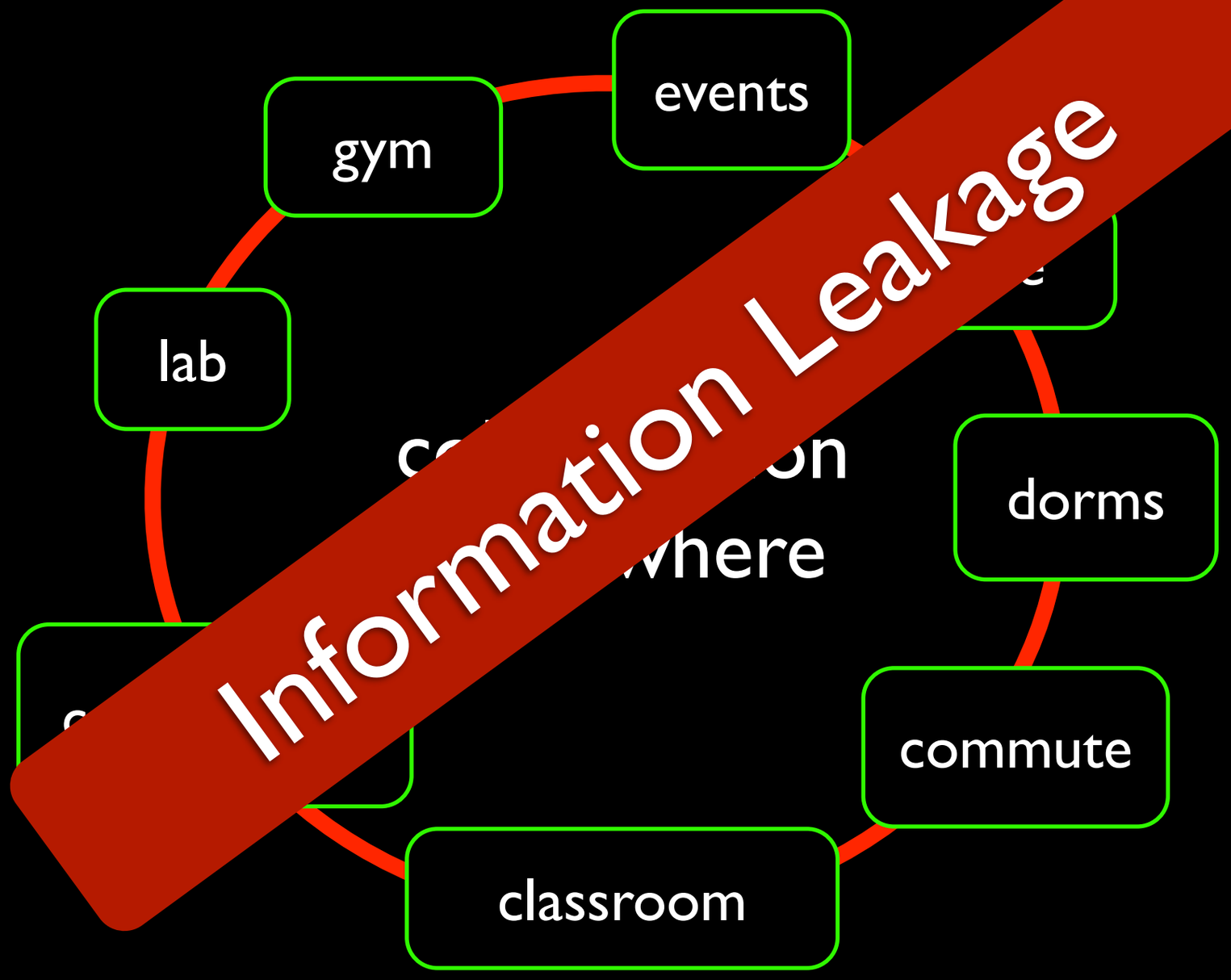
Consistency

Keep all devices synchronized

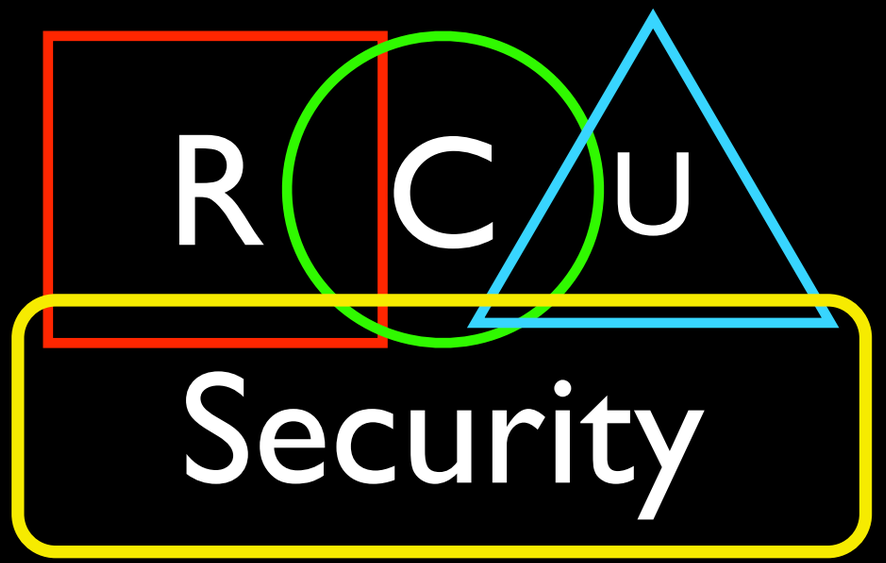
Personal Spaces



“Leaking” Spaces



The fourth element



Keep all devices synchronized

Eliminate information leakage

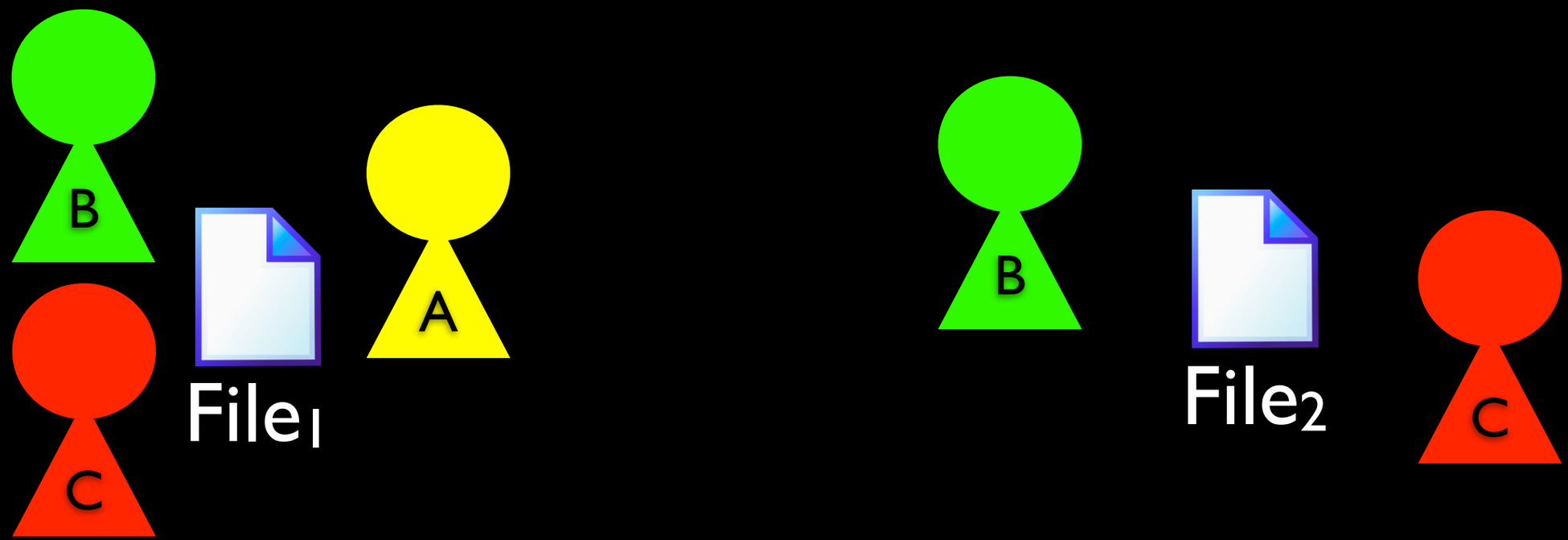
Contributions

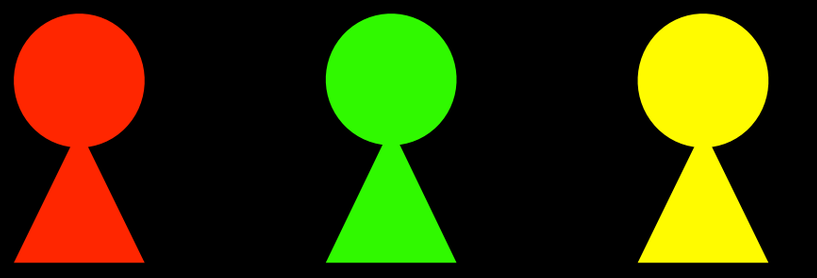
Security, as a building block

No information leakage

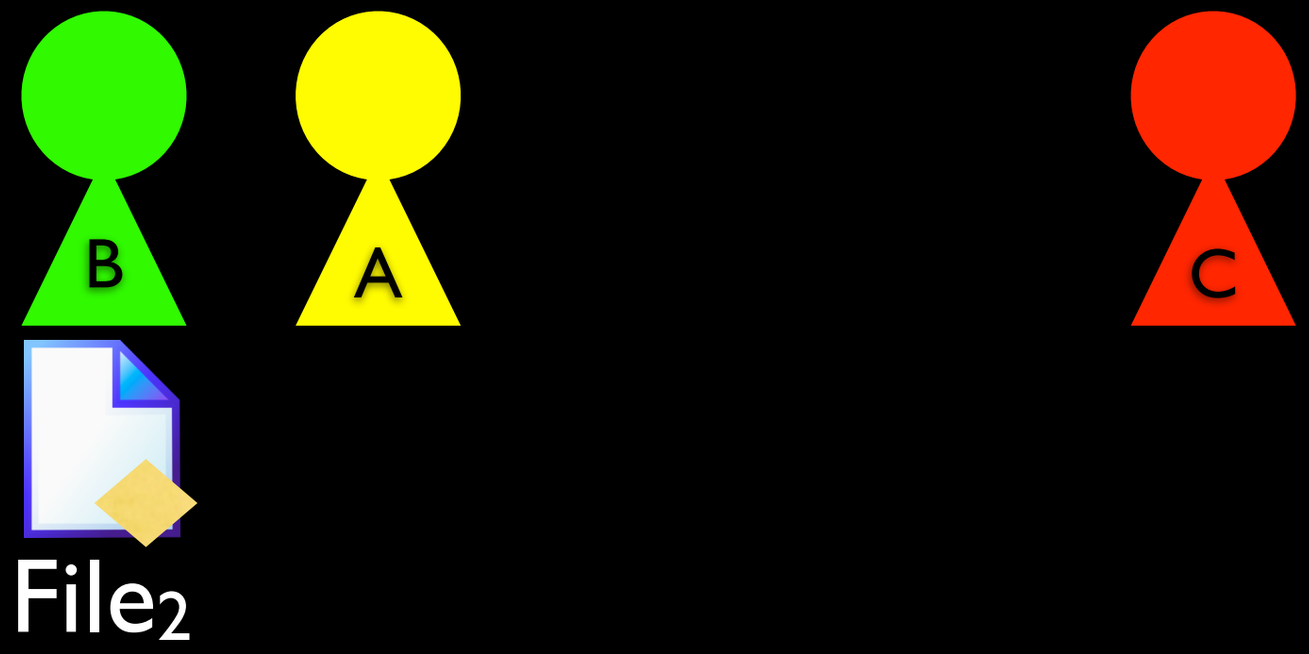
- Bloom filters
- Role-based consistency
- Object forking

Leakage in action





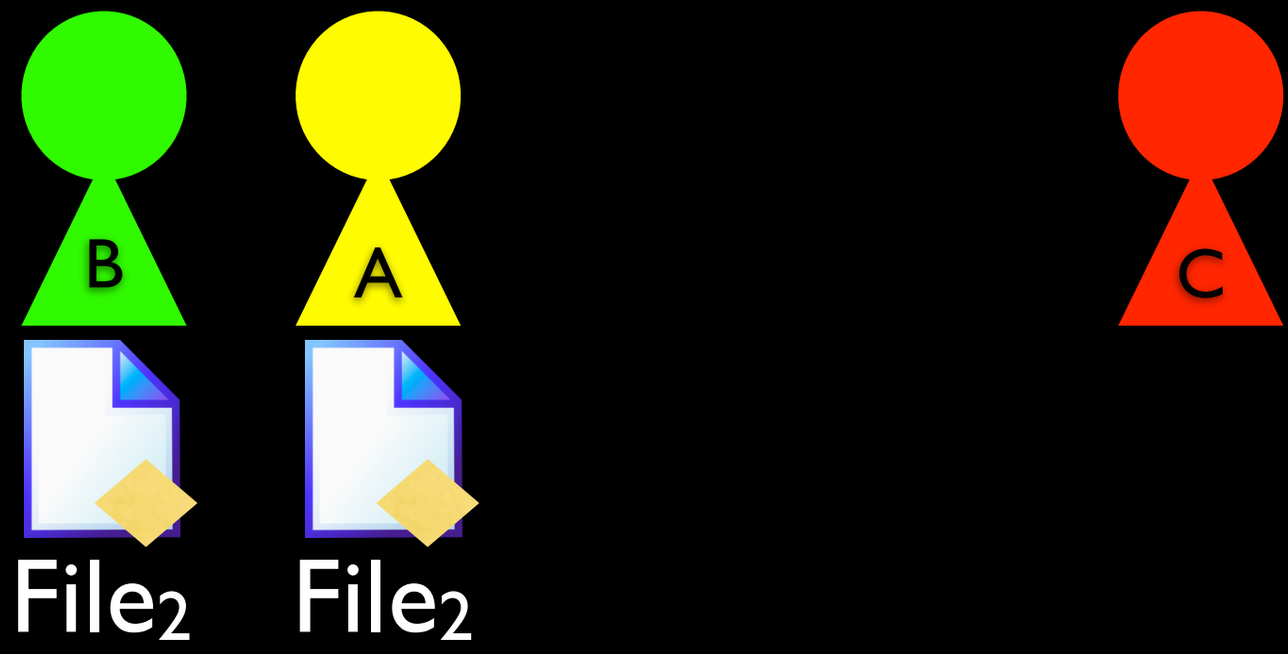
Leakage in action



Beach

Dorm

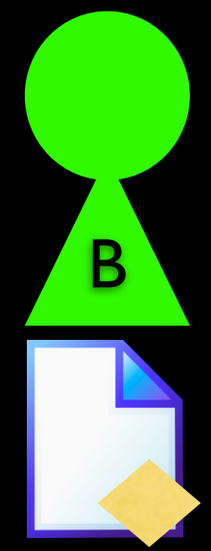
Leakage in action



Beach

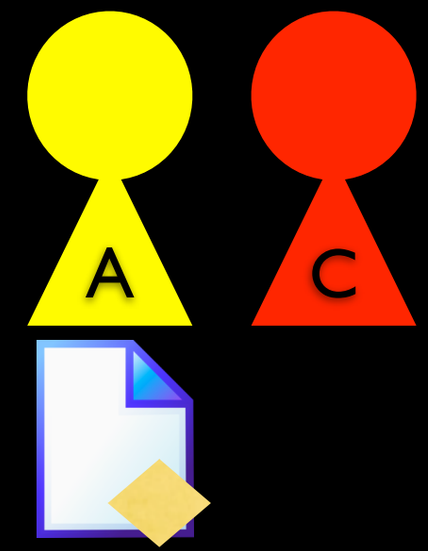
Dorm

Leakage in action



File$_2$

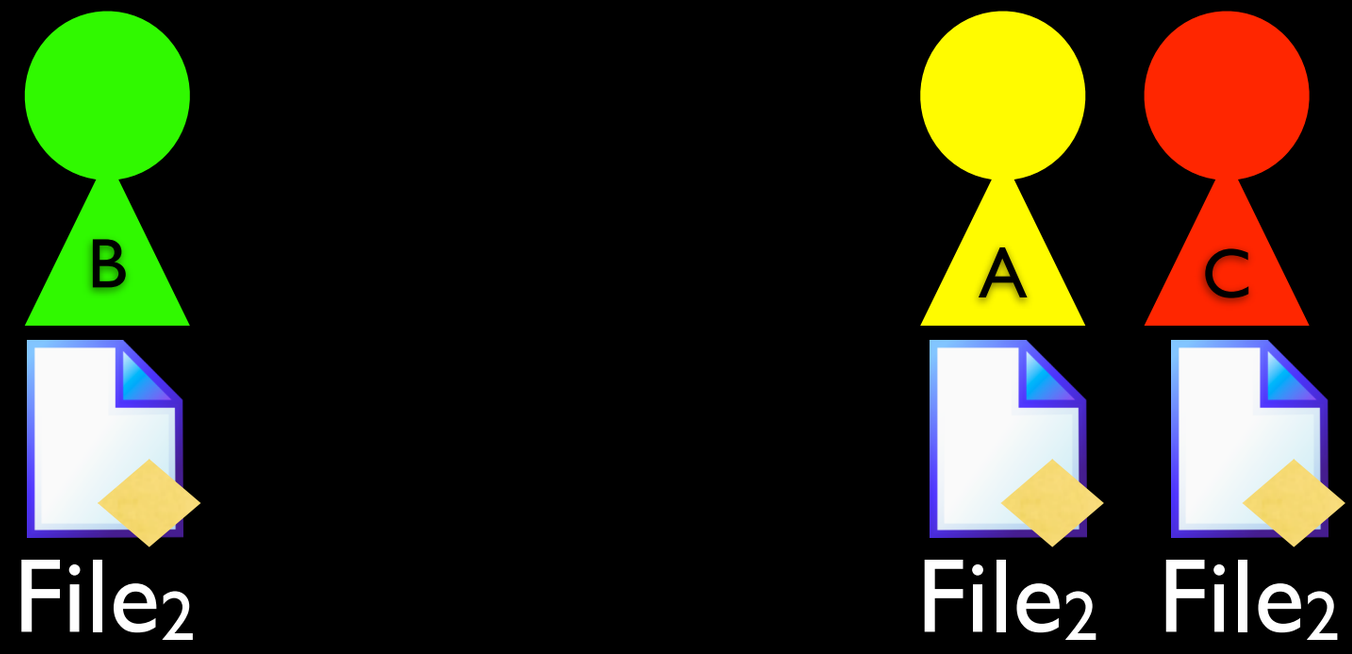
Beach



File$_2$

Dorm

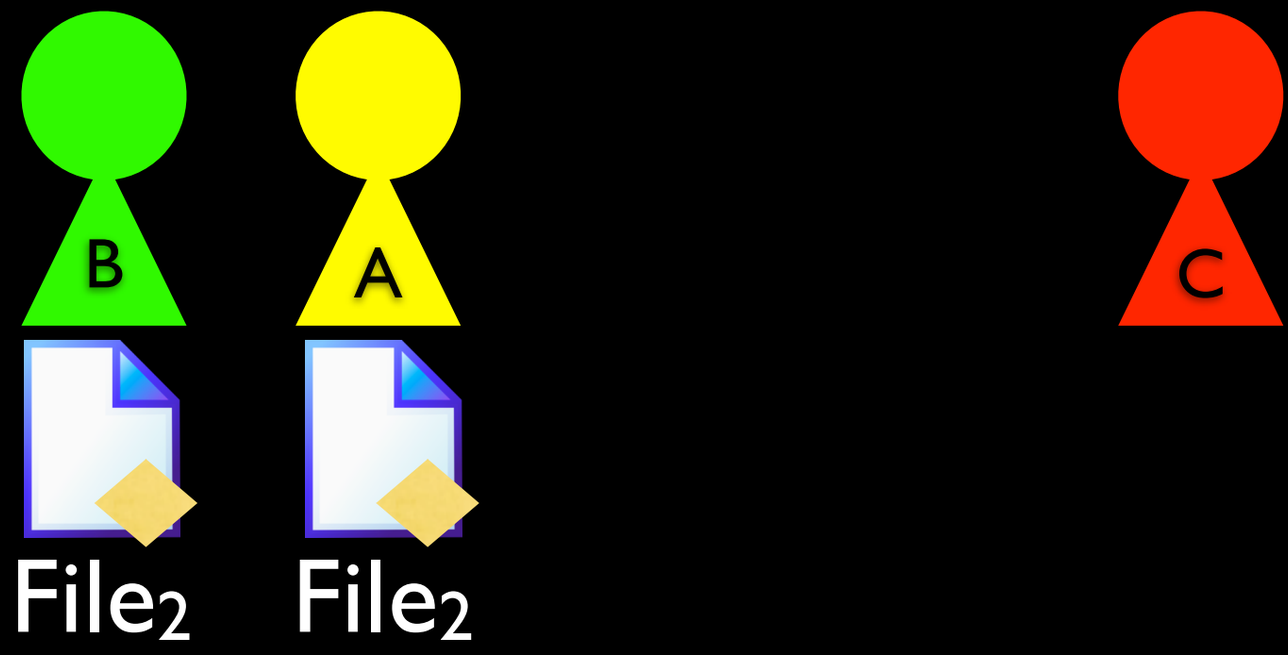
Leakage in action



Beach

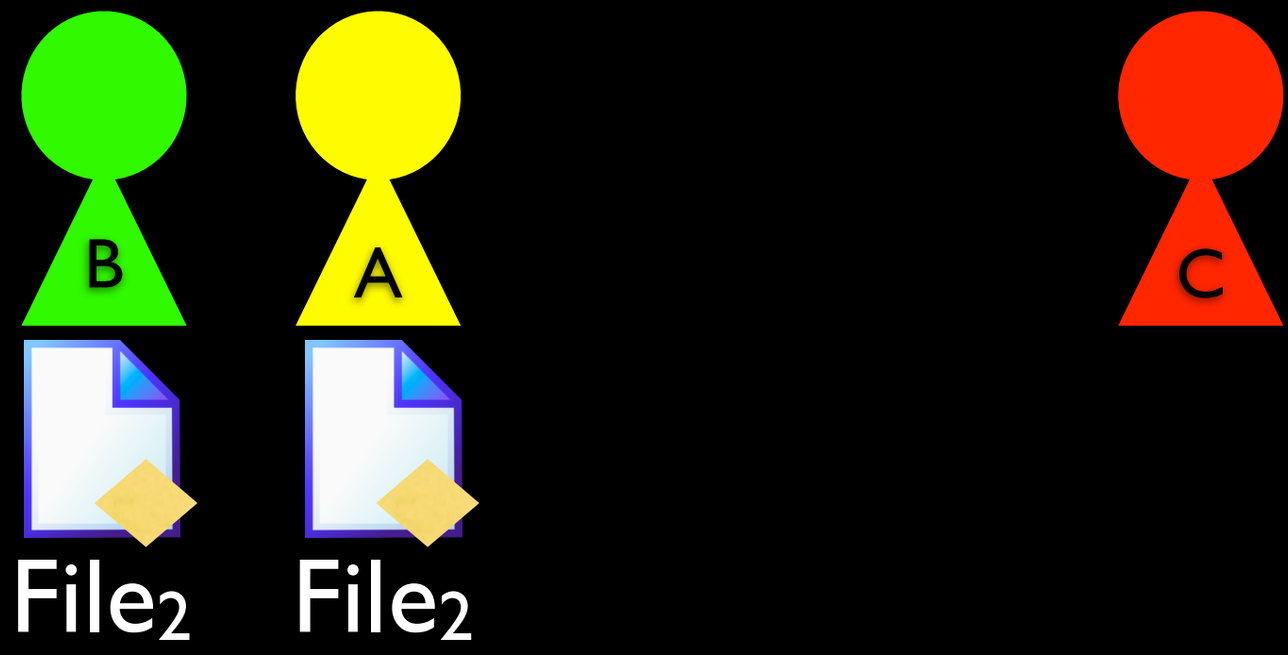
Dorm

Leakage in action



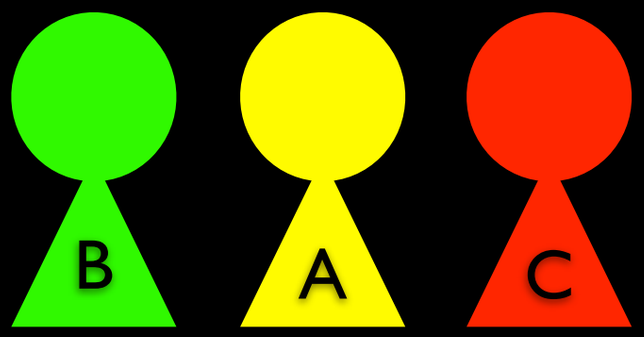
Information leakage: Alice learns about $File_2$

Can we do better?



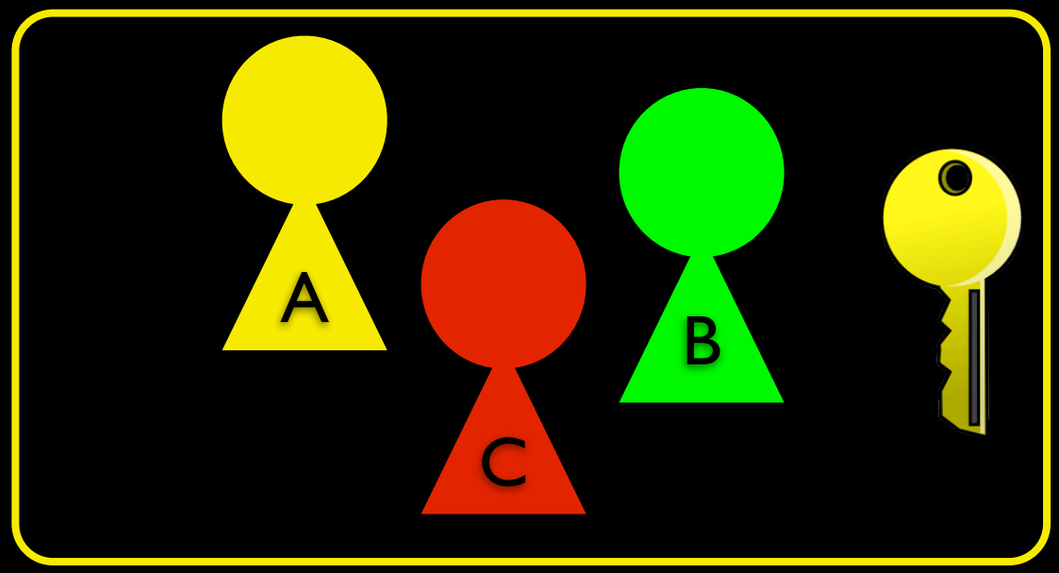
Eliminate information leakage
Maintain a flexible update mechanism

Access control elements



Principals

Alice.OS-Notes



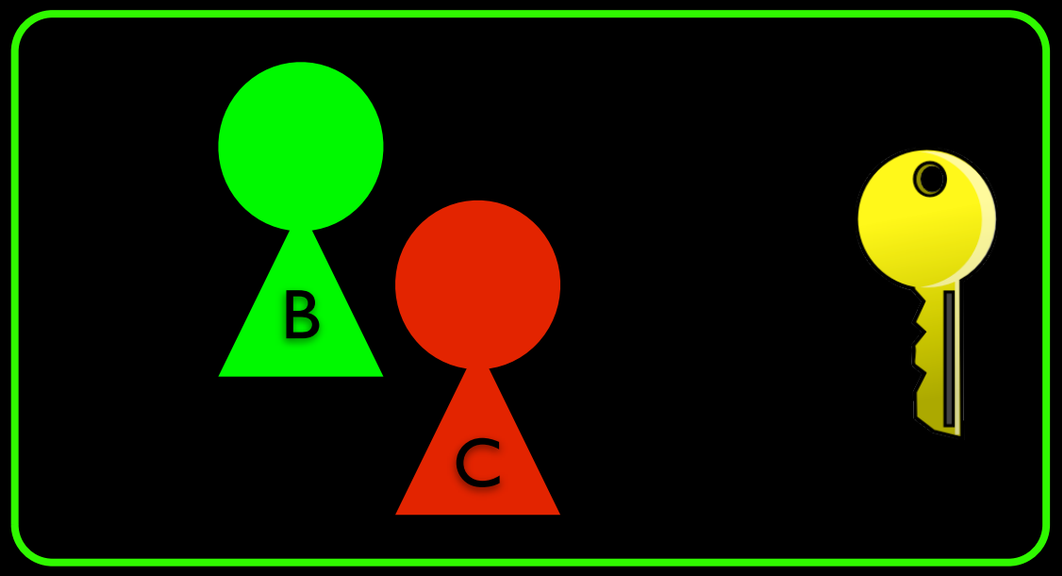
type:Notes & class:OS

What we consider leakage?

Any data access outside the realm of a role
Replicas should not reveal their roles to other replicas

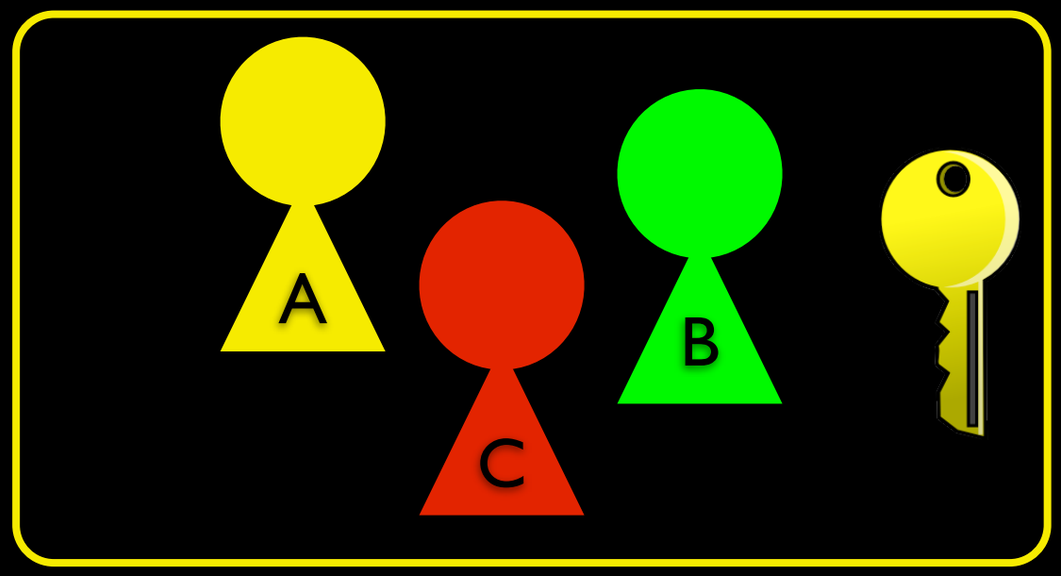
Leakage-free update

Bob.OS-Project



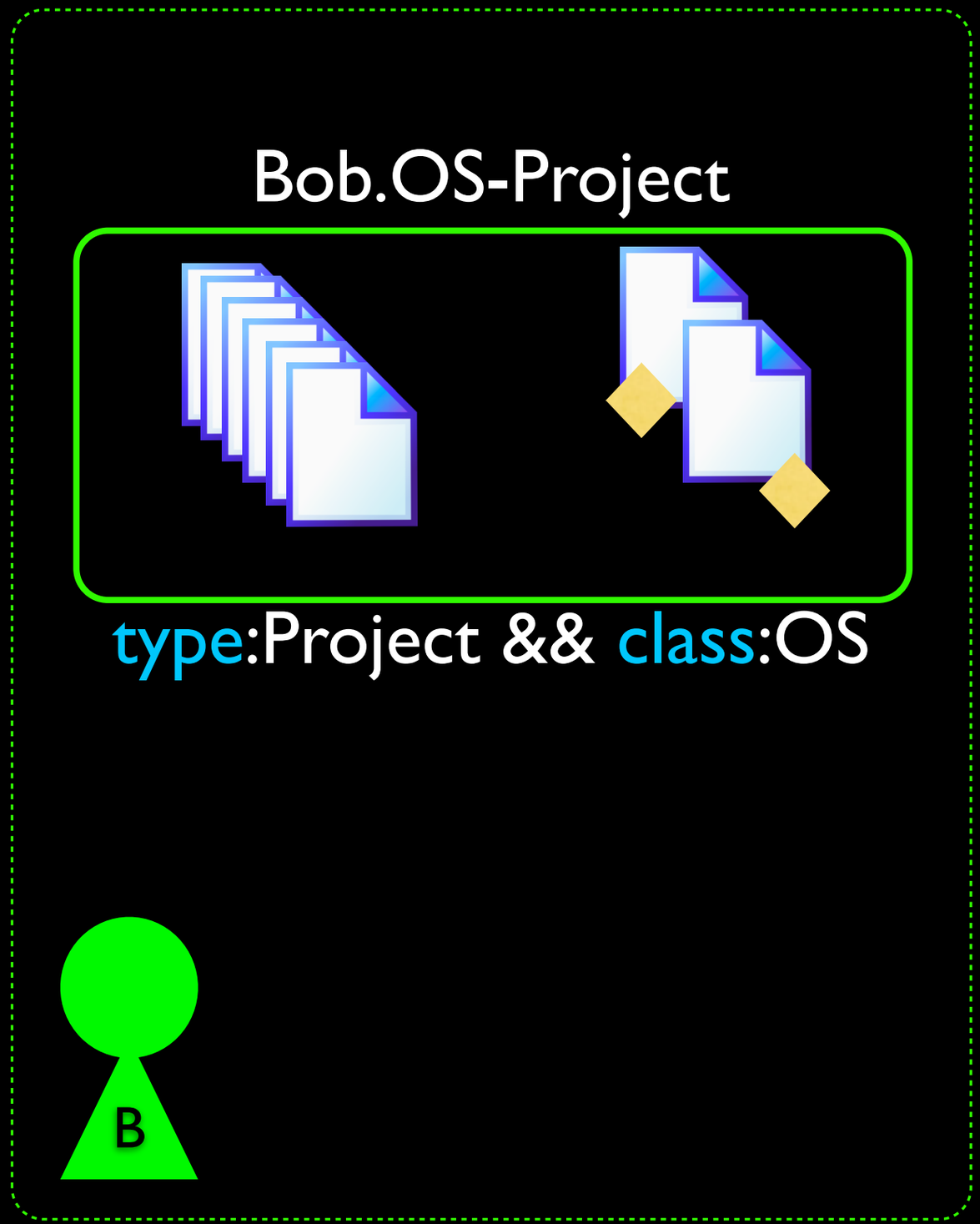
type:Project & class:OS

Alice.OS-Notes

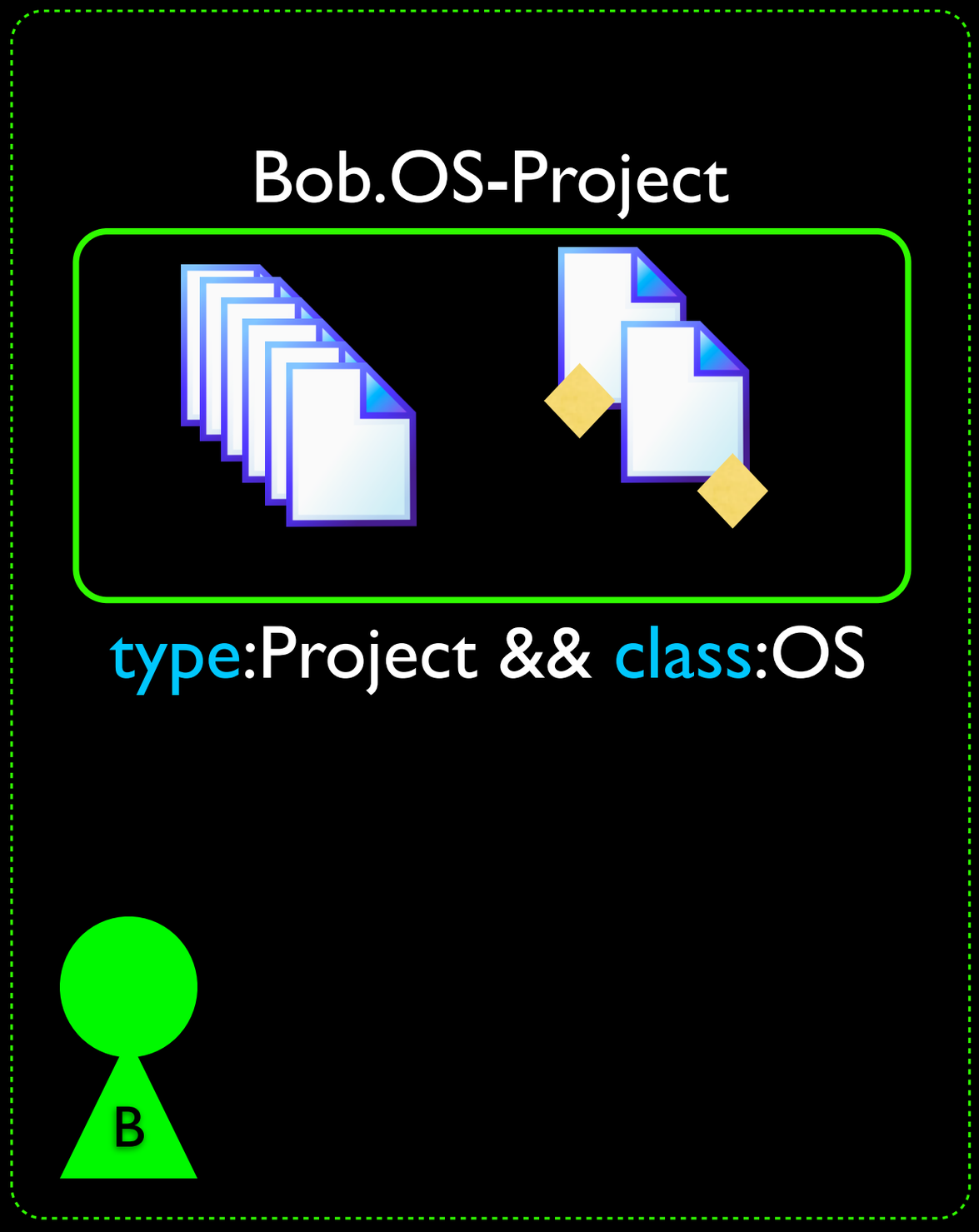


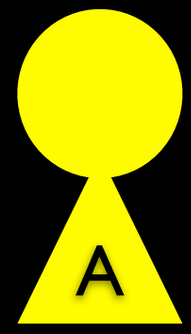
type:Notes & class:OS

Leakage-free update

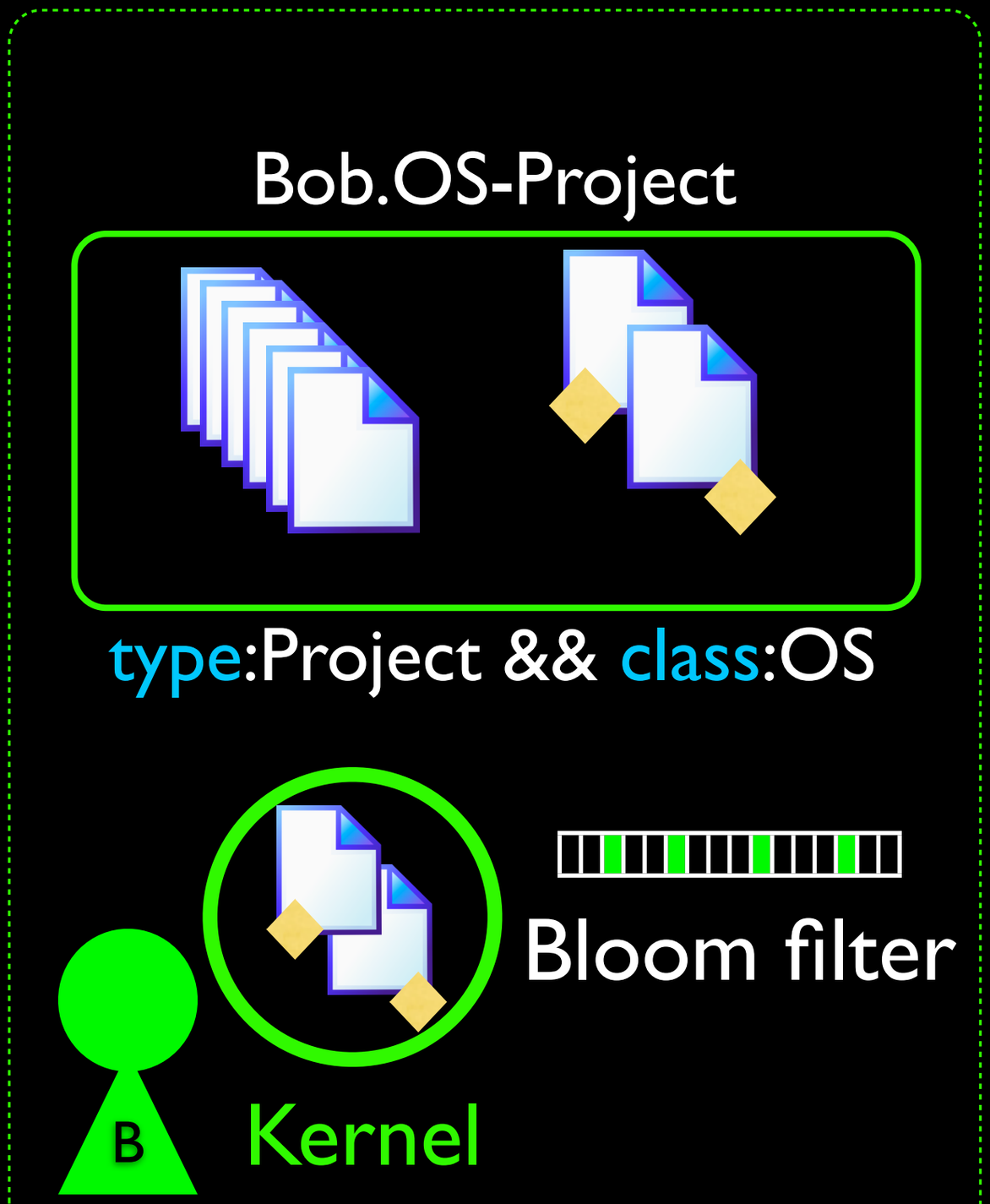


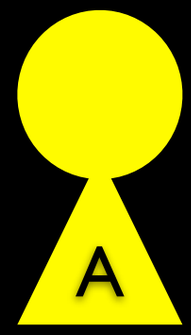
Leakage-free update



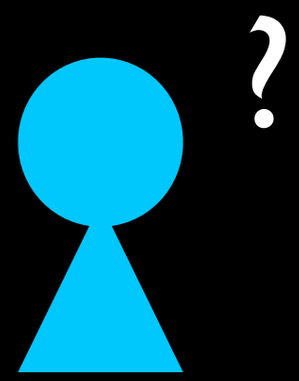


Leakage-free update



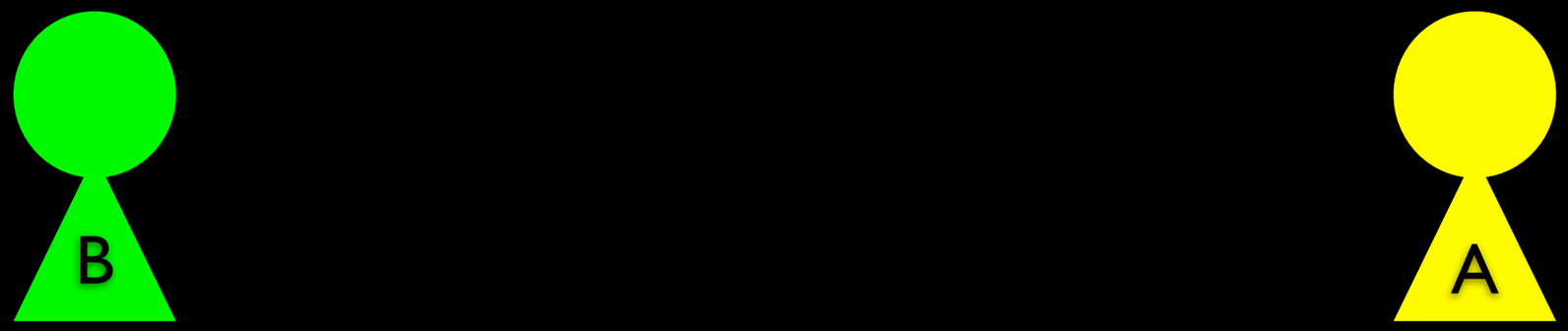


Why use a bloom filter?



Learn how to update peers without leaking information

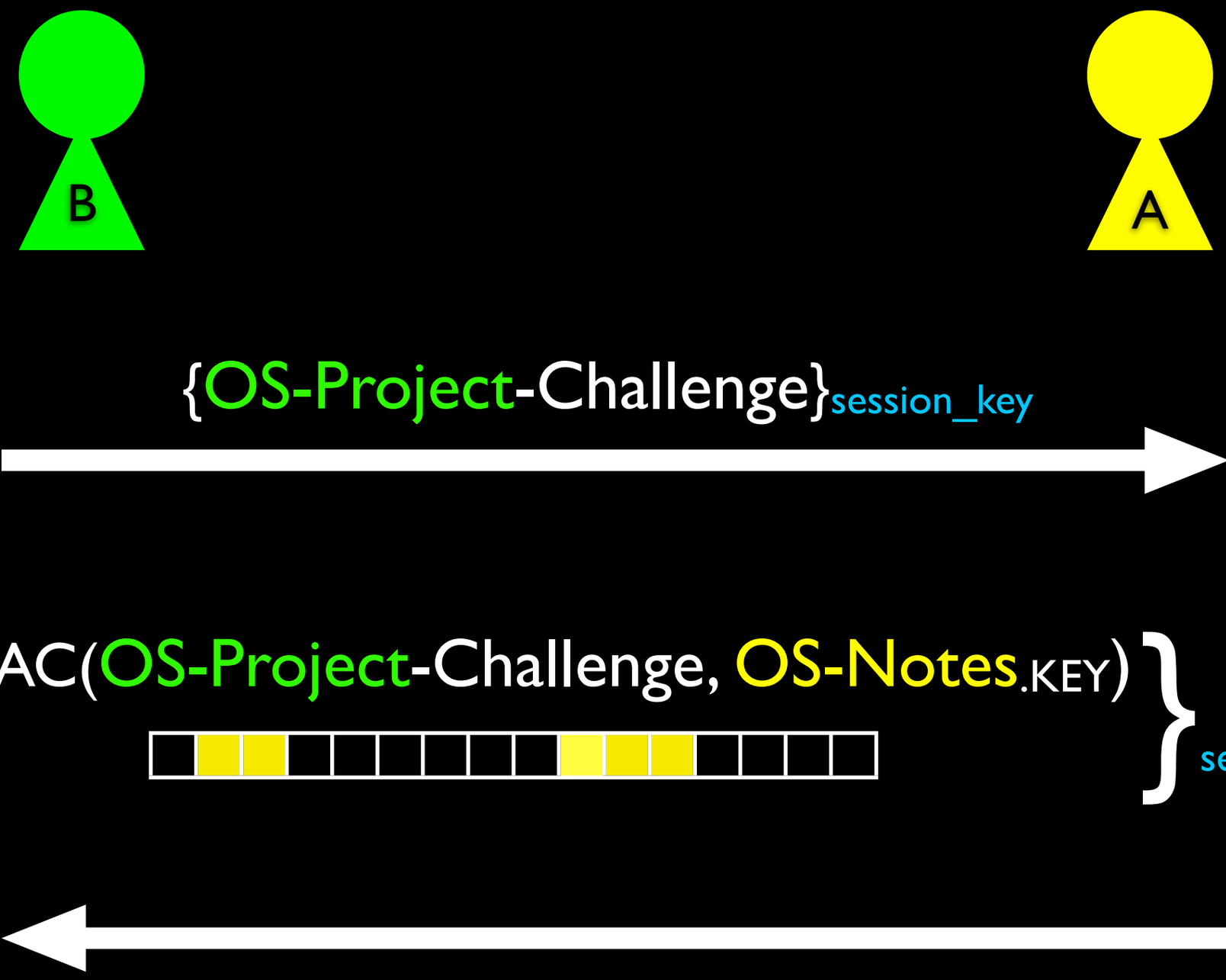
Leakage-free update



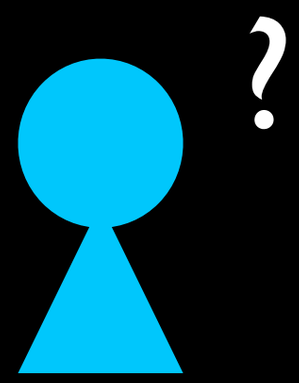
Confidentiality:
Session Key Establishment

Leakage-free update

Leakage-free update



Why the HMAC step?



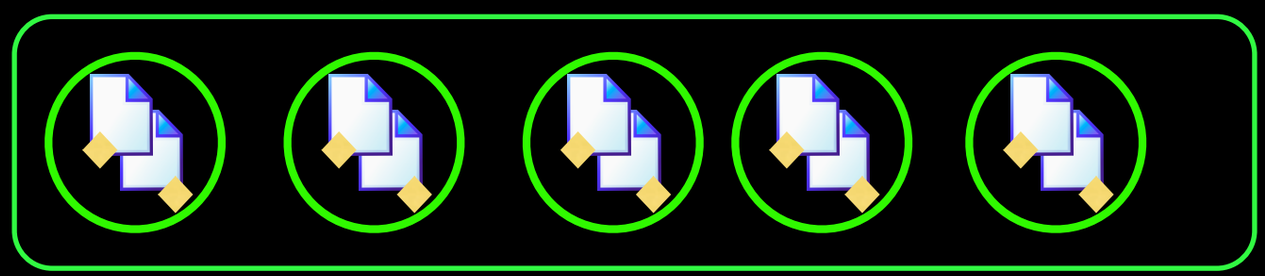
To decide when to use a pure log exchange approach without leaking information

Leakage-free update

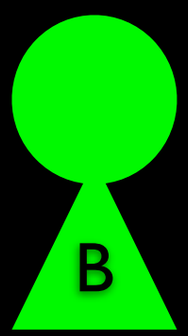
CMP{ HMAC(OS-Project.CHALLENGE, OS-Notes.KEY)

HMAC(OS-Project.CHALLENGE, OS-Project.KEY) ✗

OS-Project Kernels

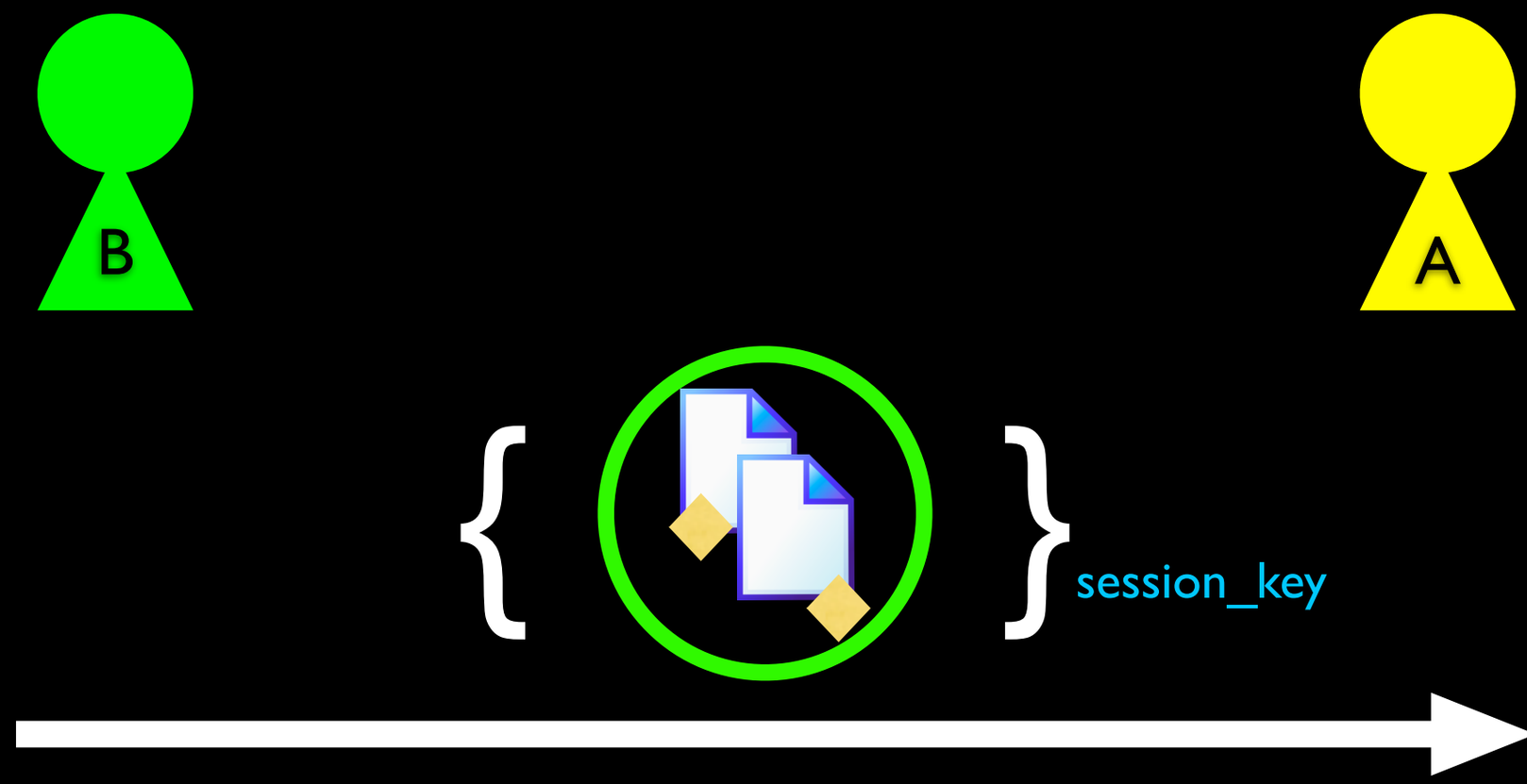


VS

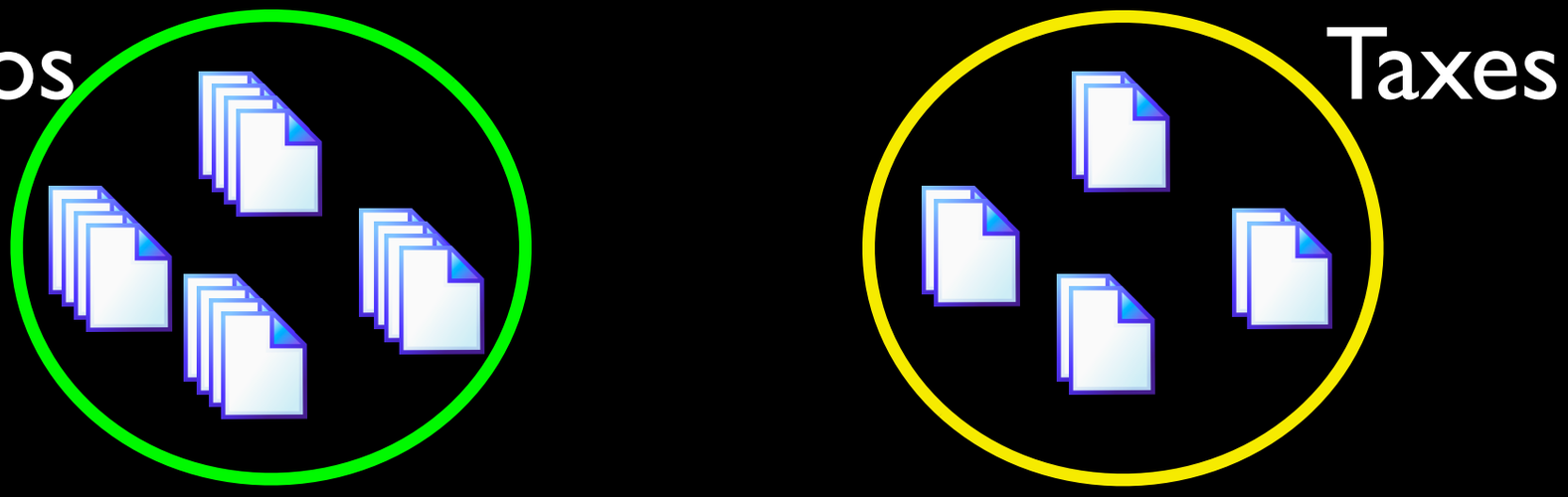


Alice's Bloom Filter

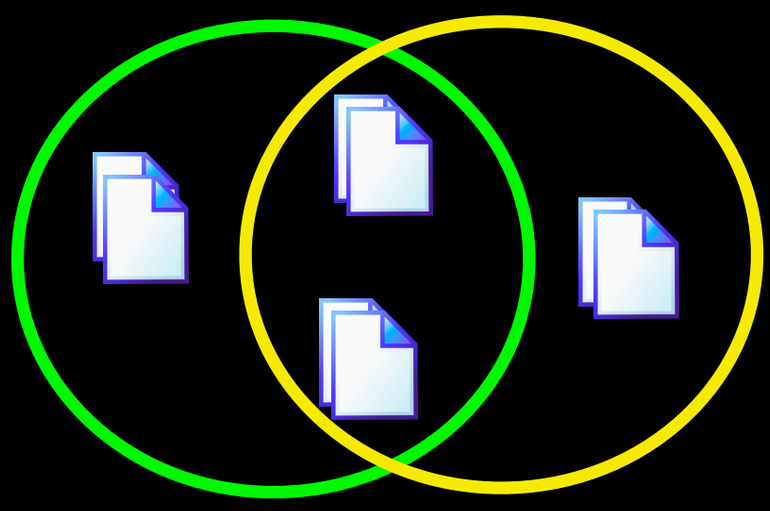
Leakage-free update



Double hatted replicas

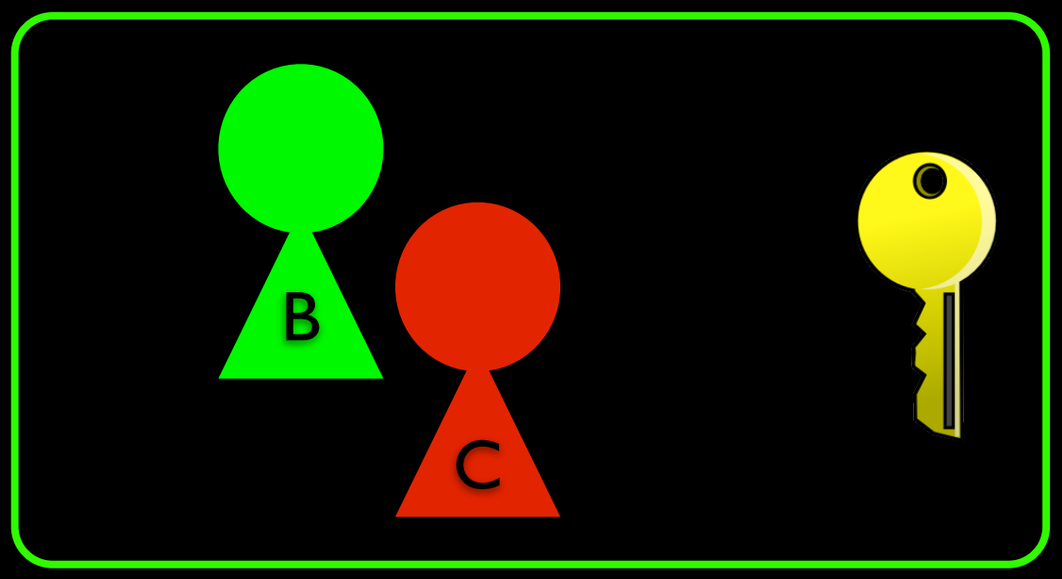


Orthogonal VS Intersecting Roles



Consistency & information leakage

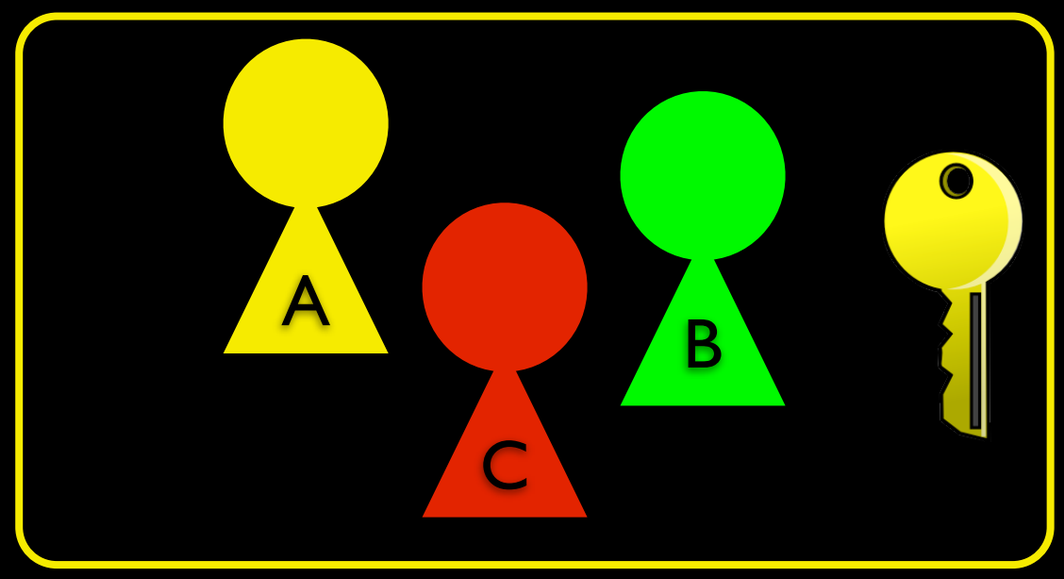
Bob.OS-Project



~~type:Project & class:OS~~

type:Project & class:OS
OR
type:Notes & date: June 14

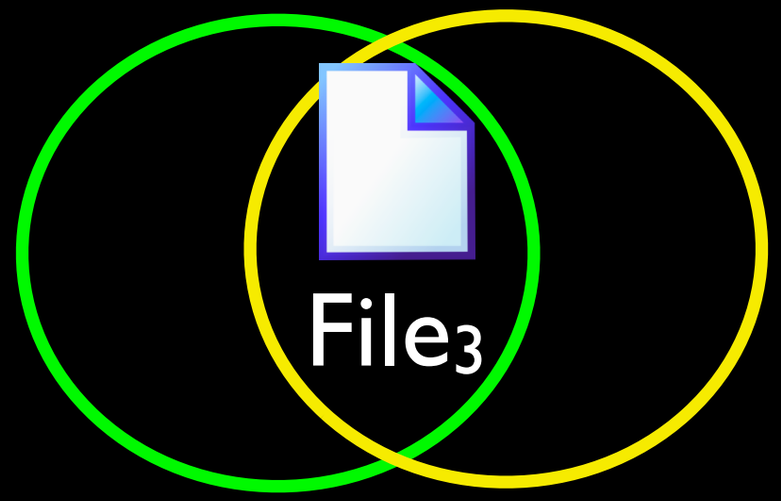
Alice.OS-Notes



type:Notes & class:OS

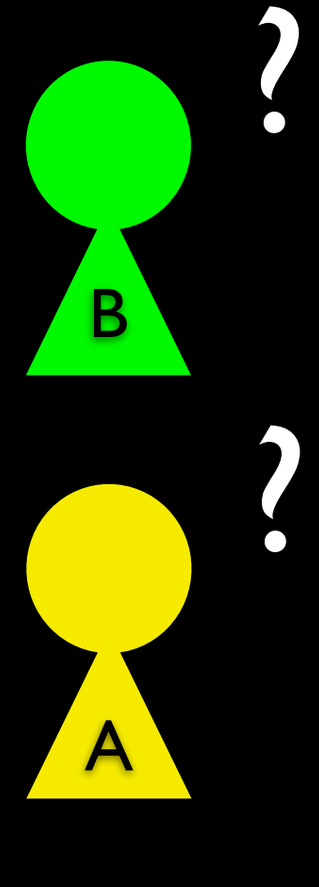
Consistency & information leakage

Bob.OS-Project



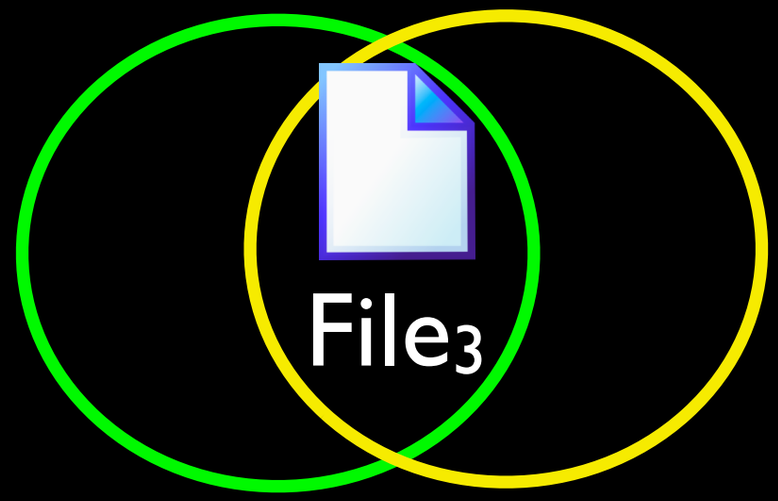
Alice.OS-Notes

File3

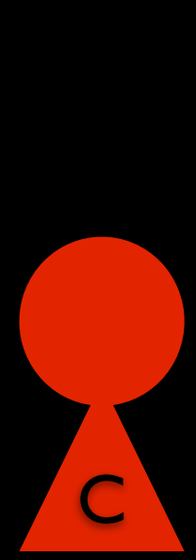


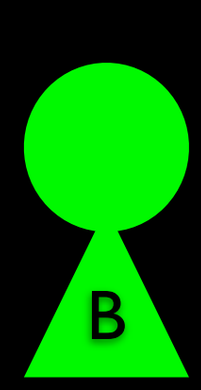
Consistency & information leakage

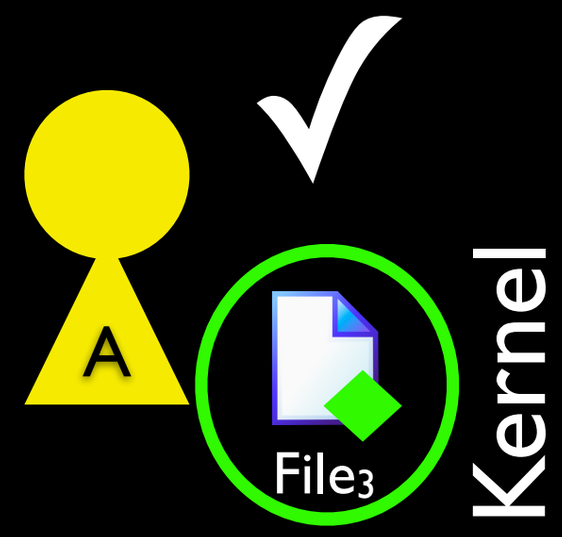
Bob.OS-Project



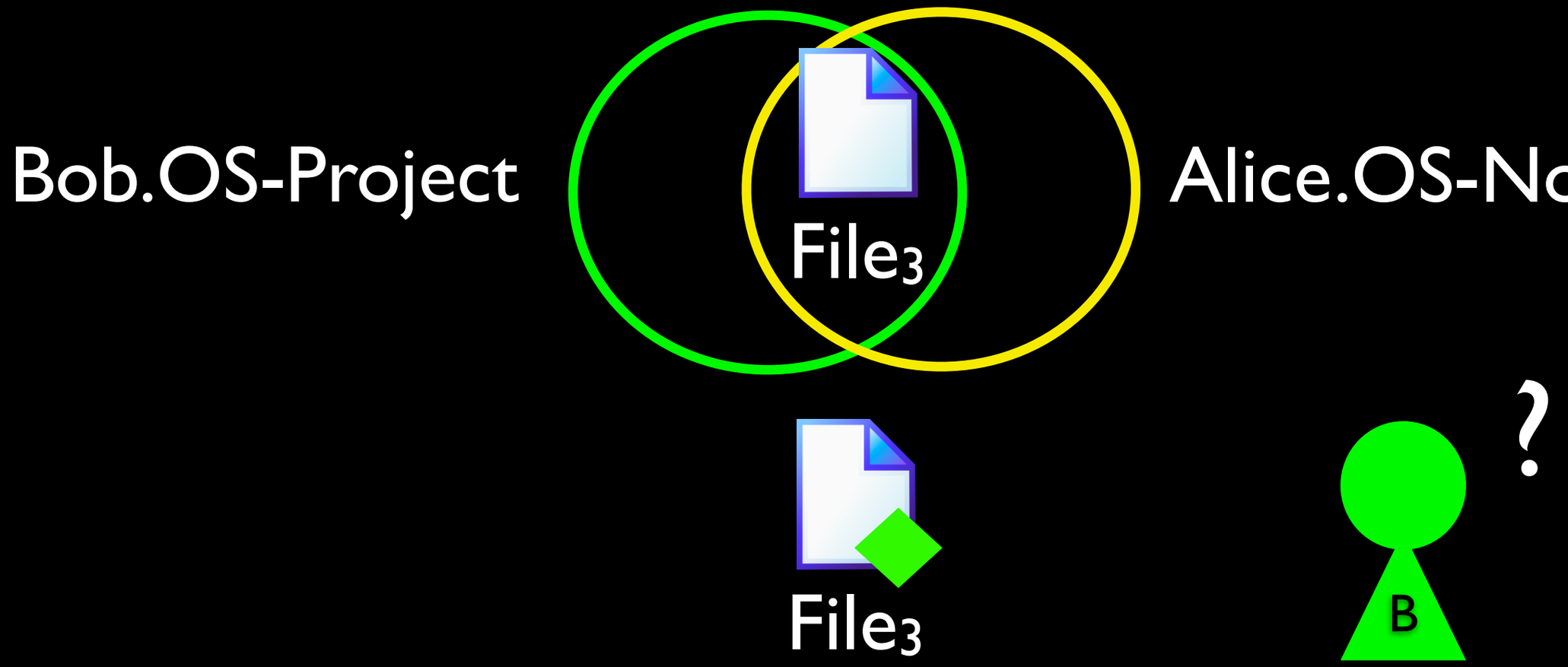
Alice.OS-Notes







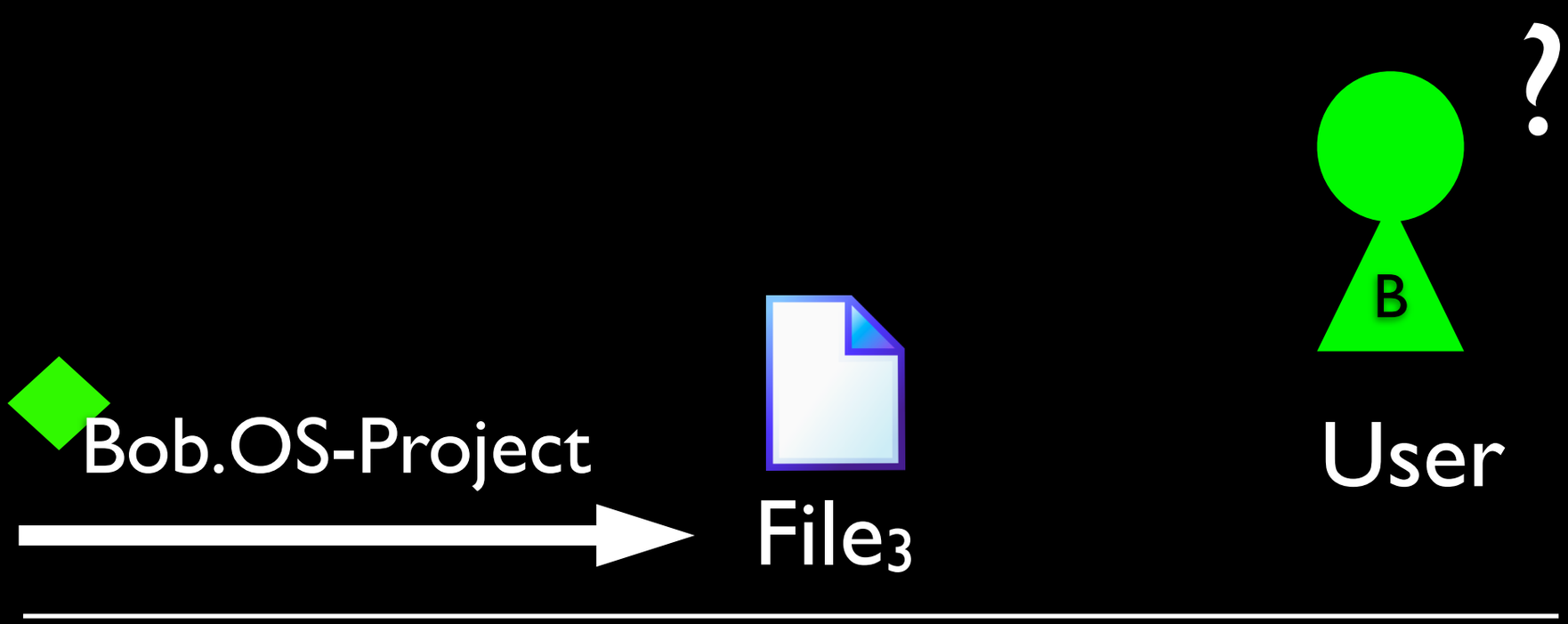
Object-based consistency



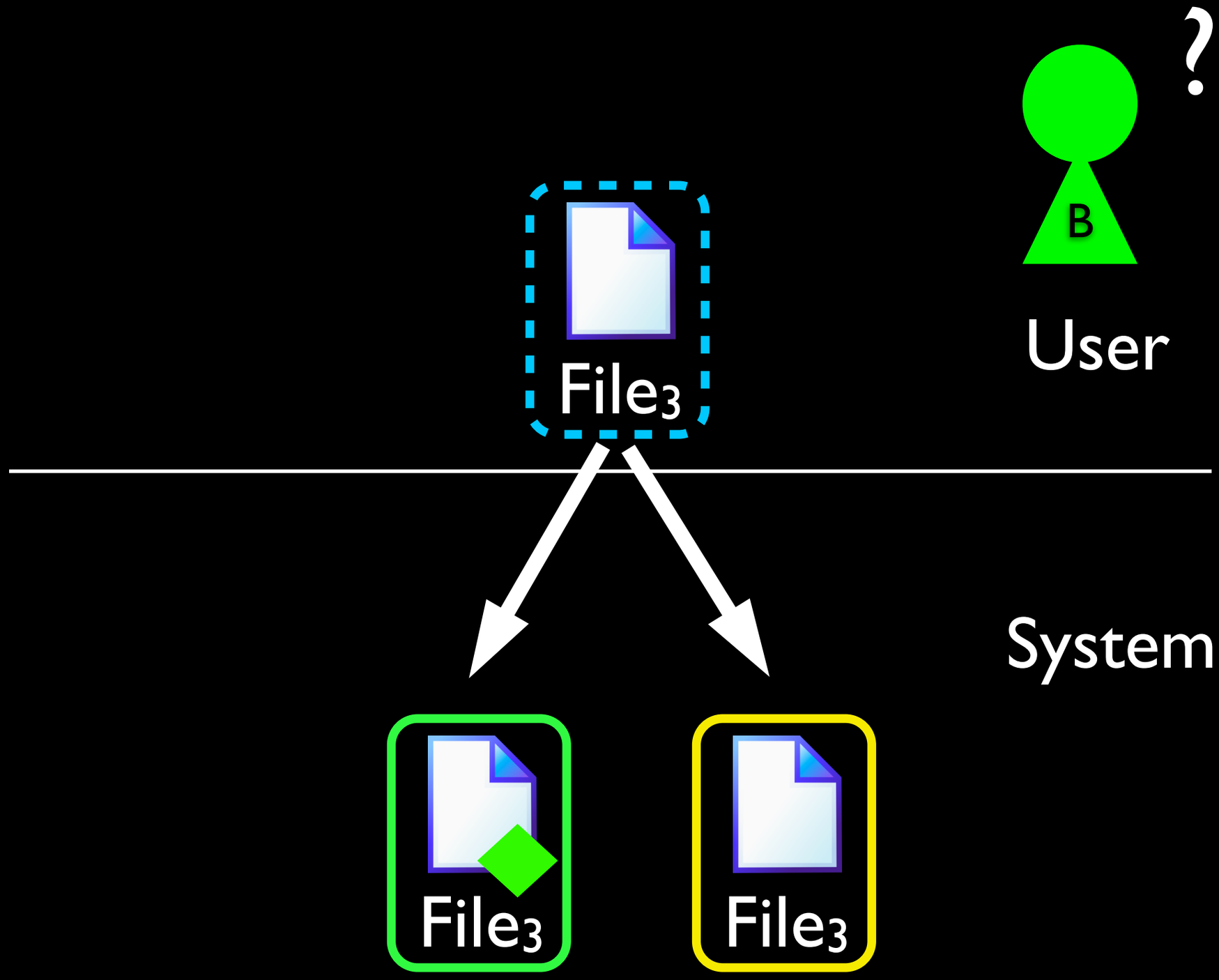
Do nothing
Updates appear to intersecting roles

Object consistency VS Information leakage

Role-based consistency & forking objects



Role-based consistency & forking objects



Role-based consistency discussion

- No Information leakage between roles
- Storage overhead
- Programming complexity
- Local accesses become an issue

Takeaway

- Current protocols in a multi-user setting leak information

- Thesis: Security should become a major building block of personal data management systems

- Elimination of leakage through
  - Role-based access control
  - Object forking
  - Role-based consistency