Producing Minimal Hook Placements to Enforce Authorization Policies

Divya Muthukumaran, Nirupama Talele, and Trent Jaeger

Prnn State University

With Vinod Ganapathy (Rutgers) and Gang Tan (Lehigh)

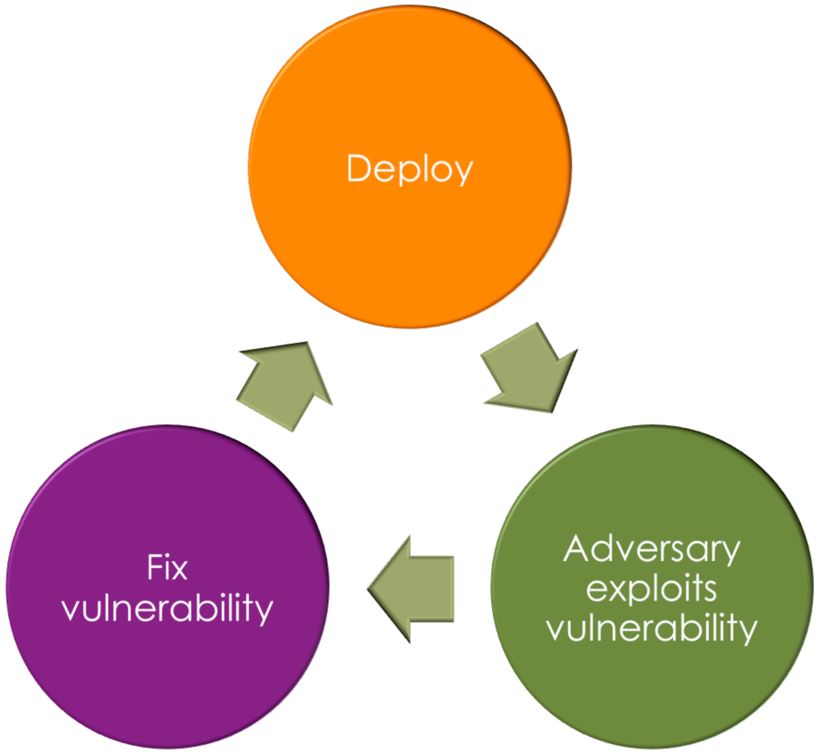
Security is not a ‘blocker’

“It isn't clear this qualifies as a blocker under any circumstances. The importance of security increases only as we are into serious deployment and start becoming a target. First things, first....”

- https://dev.laptop.org/ticket/260

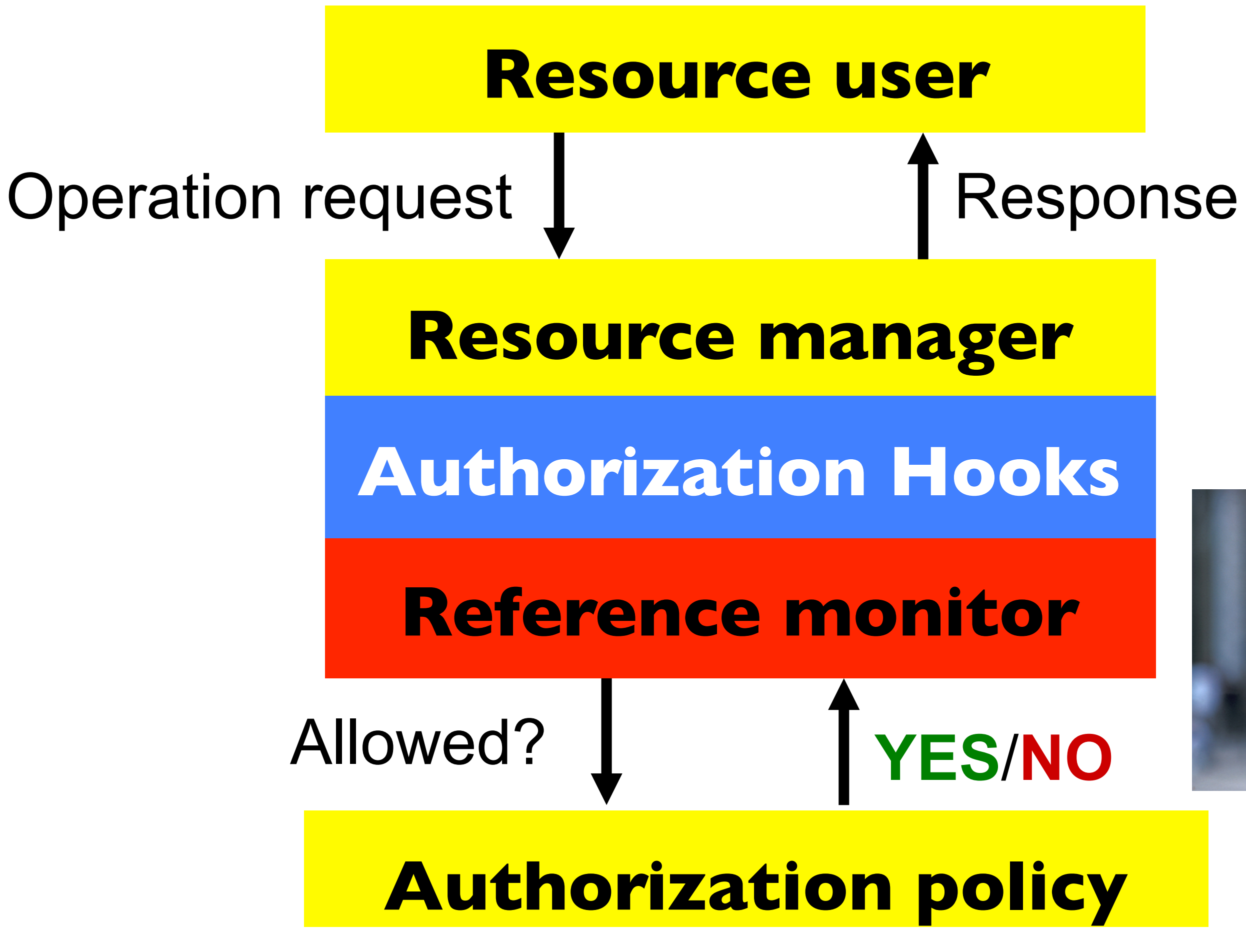
Need a way to protect against applications sniffing each other's keystrokes, which X permits by default.

Need for retroactive security



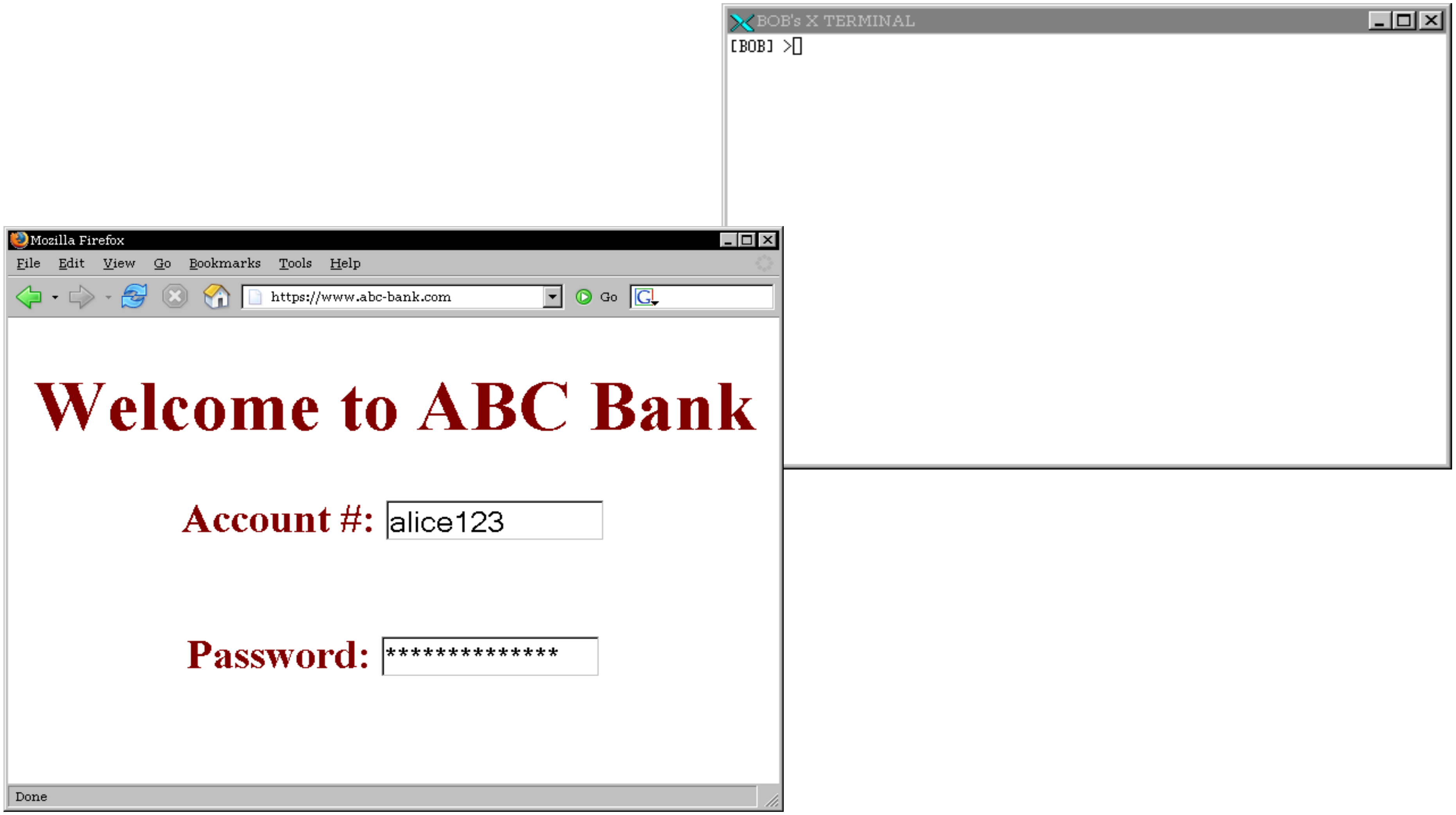
- Large codebases need retroactive security features.
  - X Server, postgres, OpenSSH, Linux Kernel, etc.
- Different problems:
  - Privilege separation, Memory errors, *Authorization*, Logging.

What is authorization ?



〈Alice, /etc/passwd, ***File_Read***〉

X Server



```
[BOB] >
```

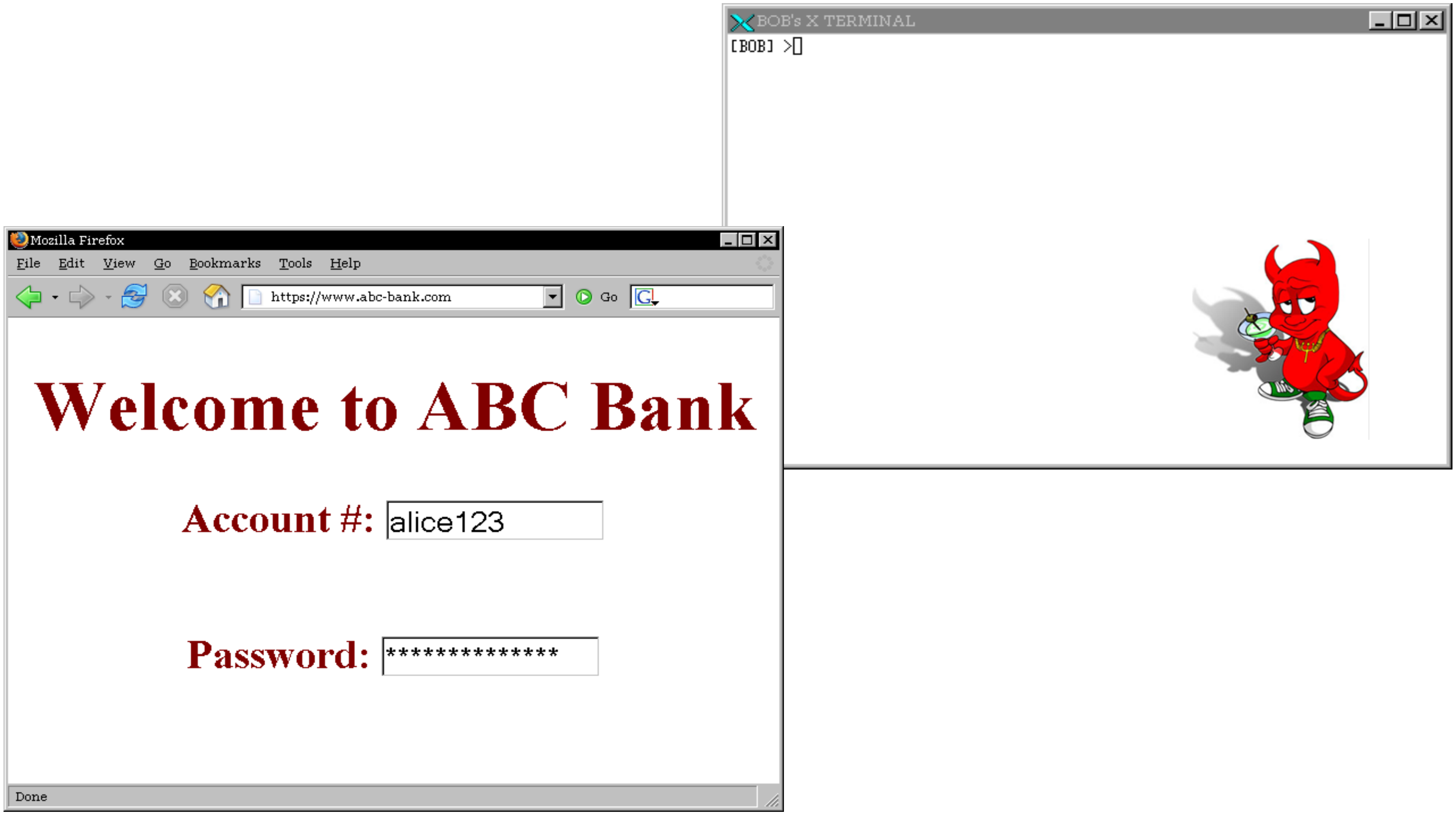
https://www.abc-bank.com

Welcome to ABC Bank

Account #: alice123

Password: **************

X Server



```
[BOB] >
```

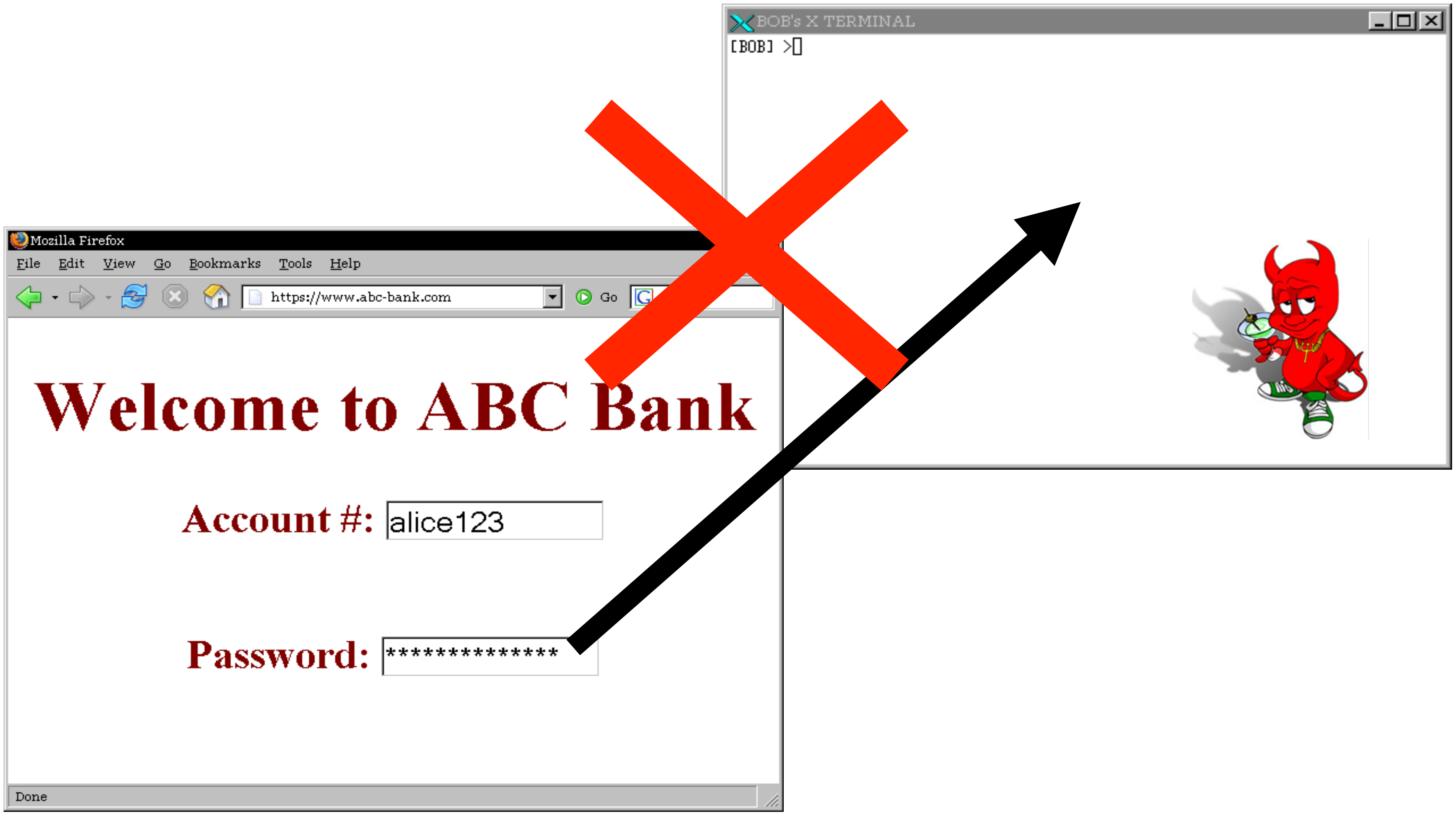
https://www.abc-bank.com

Welcome to ABC Bank

Account #: alice123

Password: **************

X Server



```
[BOB] >
```

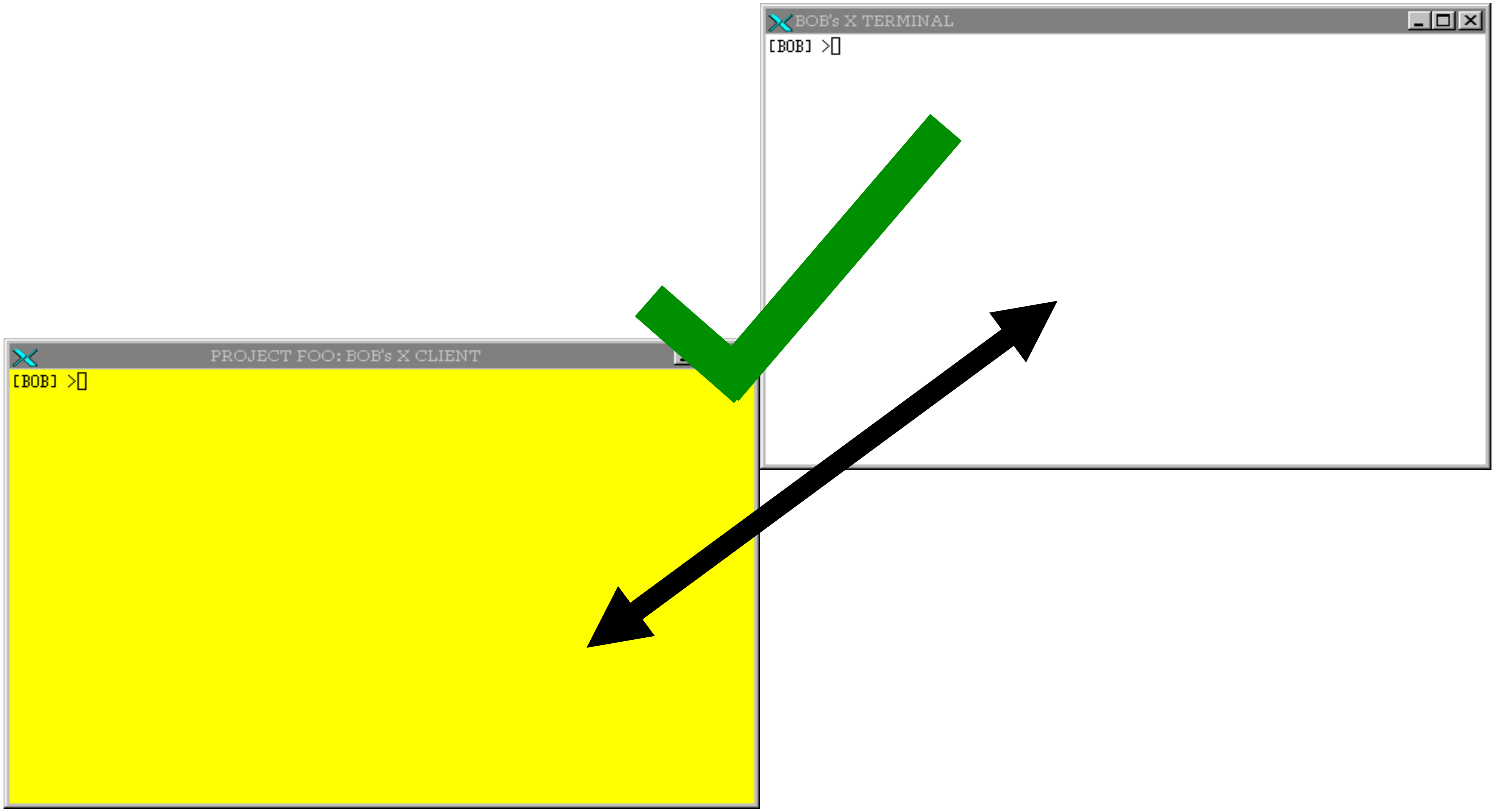
https://www.abc-bank.com

Welcome to ABC Bank

Account #: alice123

Password: **************

X Server

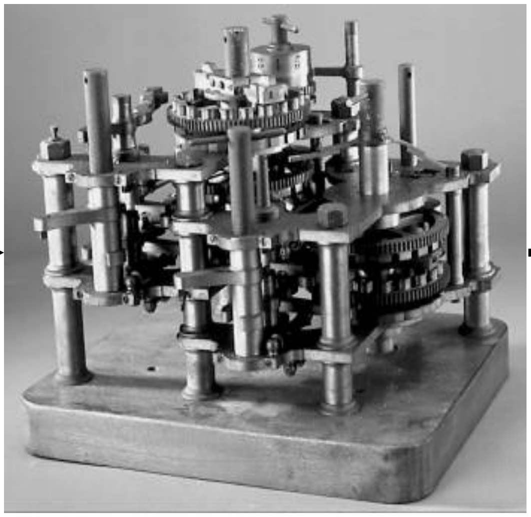


Retrofitting code for authorization

Need systematic techniques to retrofit legacy code for security

Legacy code →

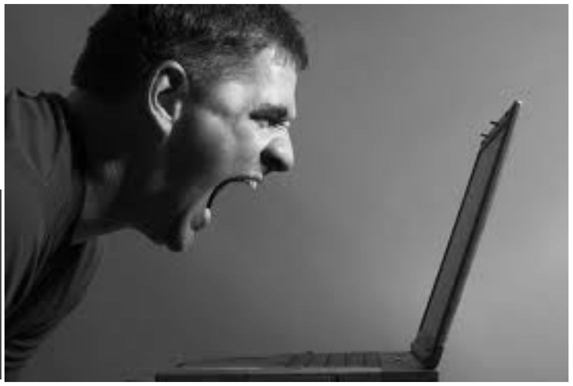
INSECURE



→ Retrofitted code

SECURE

Painstaking, manual procedure



- X11 ~ proposed 2003, upstreamed 2007, changing to date. [Kilpatrick *et al.*, '03]
- Linux Security Modules ~ 2 years [Wright *et al.*, '02]

At this point, SE-PostgreSQL has taken up a ***lot* of community resources**, *not to mention an* **enormous and doubtless frustrating amount of *the lead developer's* time and effort**, *thus far* **without a single committed patch, or even a consensus as to what it should (or could) do.** *Rather than continuing to blunder into the future, I think we need to do a reality check*
- http://archives.postgresql.org/message-id/20090718160600.GE5172@fetter.org

Past efforts

Verifying Hook Consistency

Assumptions: Mature Code

Inputs: Existing hook placement.

Examples:

For Kernels [Zhang et al., 2002, Edwards et al., 2002, Tan et al., 2008]

For Web Applications [Sun et al., 2011, RoleCast 2011, FixMeUp 2012]

Placing Hooks Automatically

Assumptions: No hooks

Inputs: Sensitive data types, hook code.

Examples:

Server Applications [Ganapathy et al., 2005, 2006, 2007]

Thesis Statement

We can ***automatically*** *retrofit server applications with* ***minimal*** *number of* ***authorization hooks*** *necessary to enforce the desired* ***authorization policies*****.**

Challenges

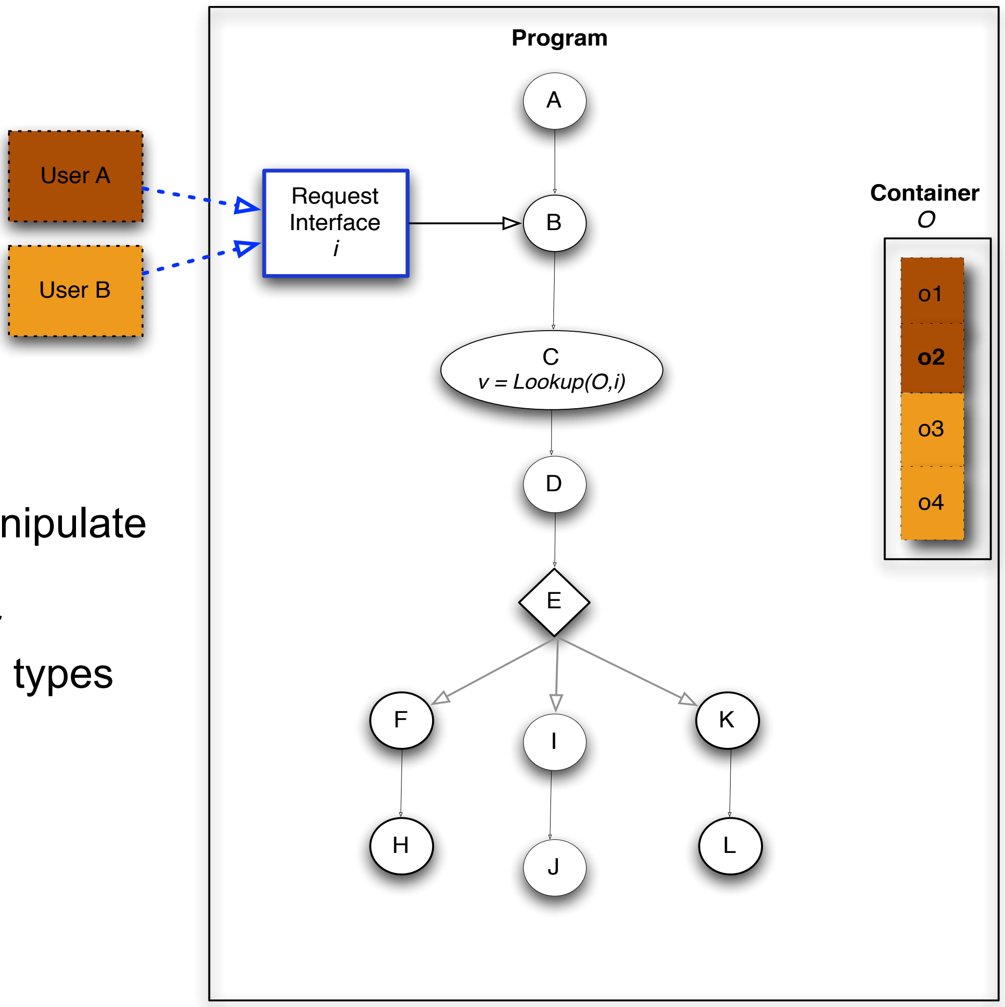
- *Infer security-sensitive **resources** and **operations**.*

- *Produce **hook placements**.*
  - Infer locations where hooks need to be placed.
    - Do not actually place the hooks.

Authorization Tuple

- [**Subject, Resource, Operation**]:
- ***Subject***: Entities served by the program. They make requests.
  - Determined at runtime.
- ***Resources* (objects)**: Entities to which access need to be controlled.
  - Given a large program, how can we identify resources that need access control?
- ***Operations***: Determines what needs to be done with the resources.
  - Operation is a set of statements. Which sets of statements are security sensitive?

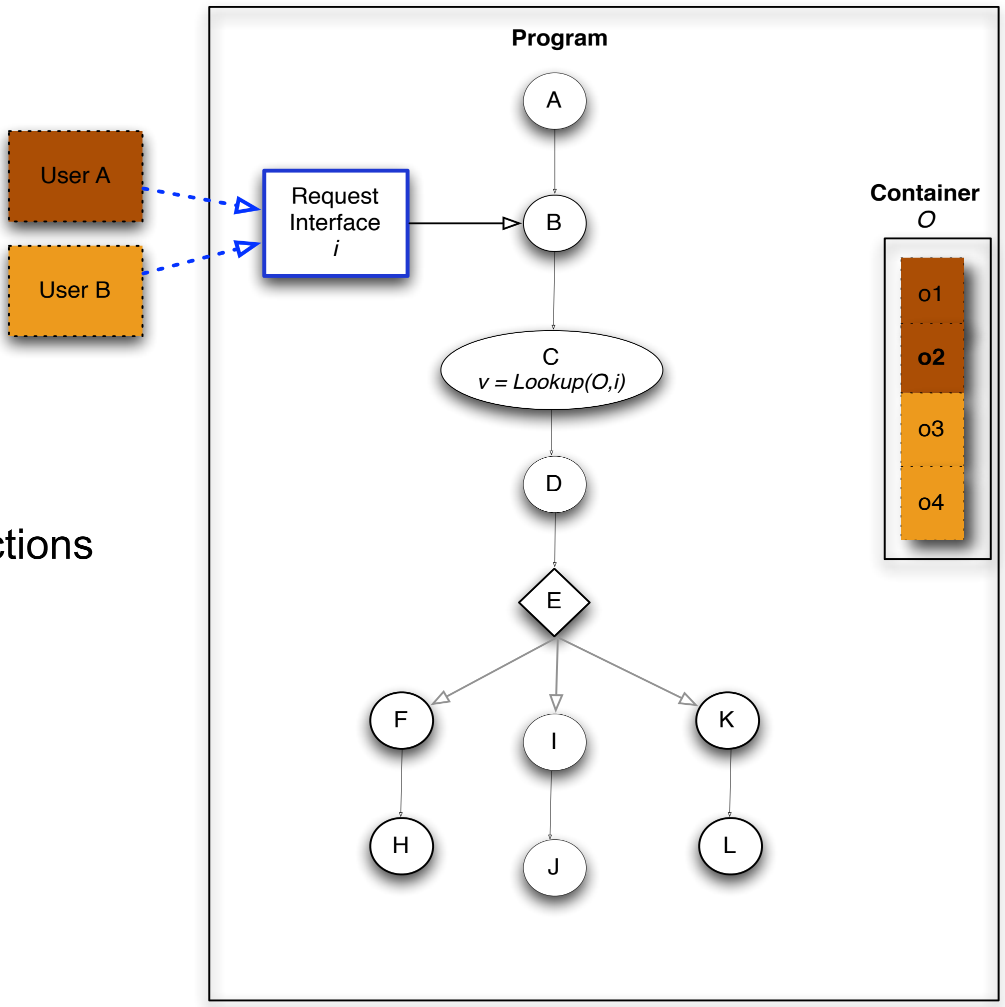
Inferring security-sensitive resources



Resources: Programs manipulate many variables

- 7800 in X Server
- Of over 400 data types

Inferring security-sensitive operations



Operations: set of instructions

- Only a subset are security-sensitive

Solution

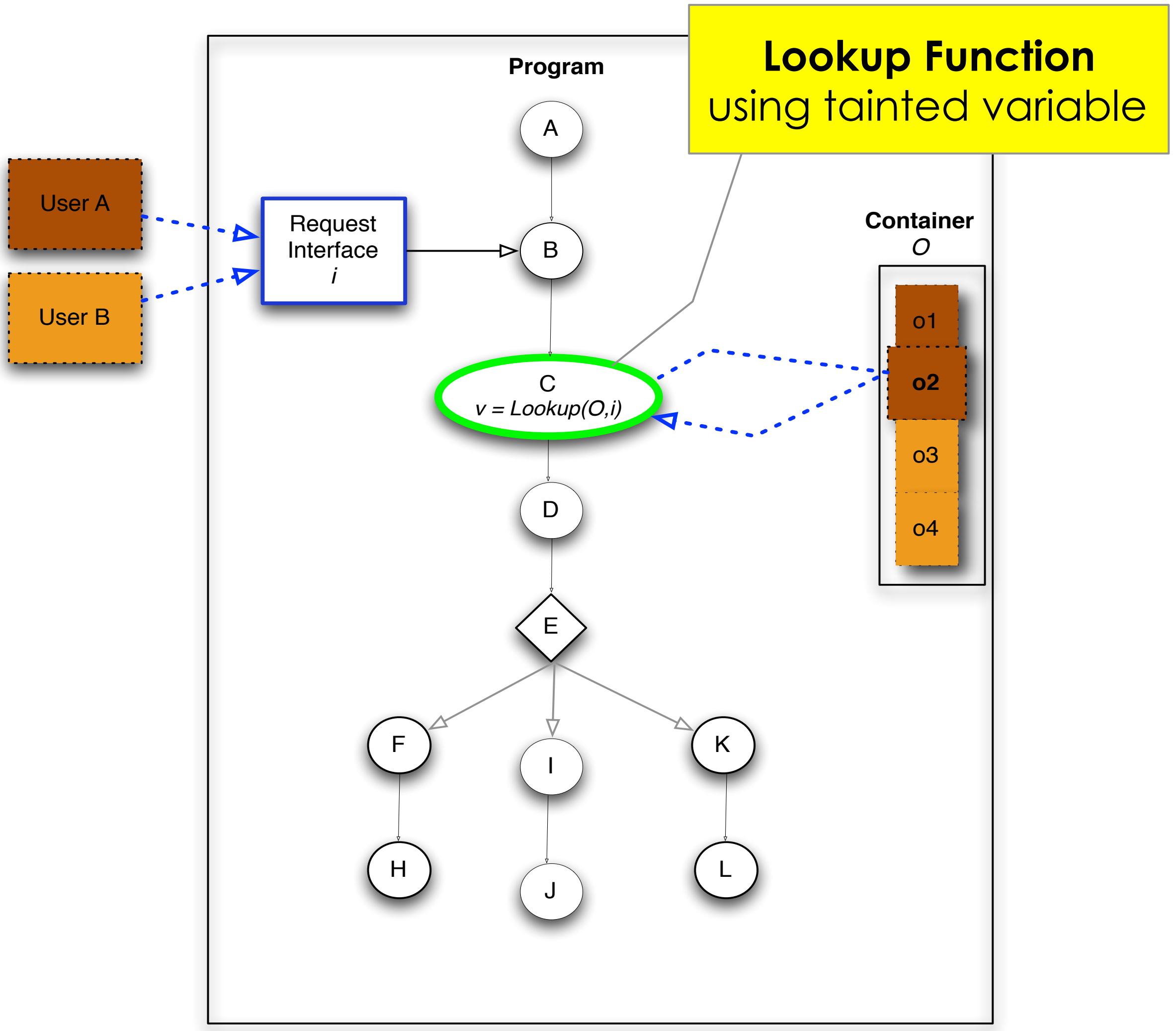
Requests make choices

•In servers, *client-request* determines *choices* that client subjects can make in the program

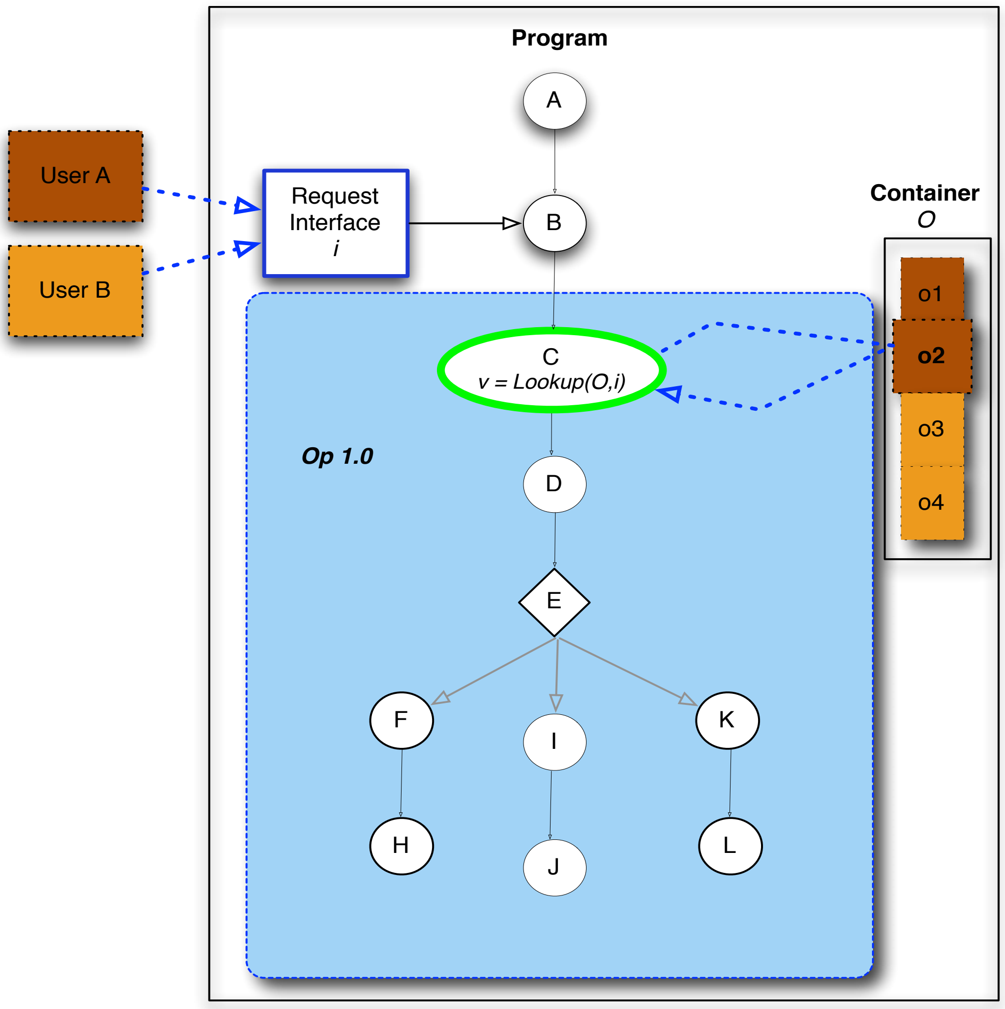
•“Choice”:

- **Resources**: Determine which *resources* are chosen from containers.
- **Operations**: Determine which *program path* is selected for execution.

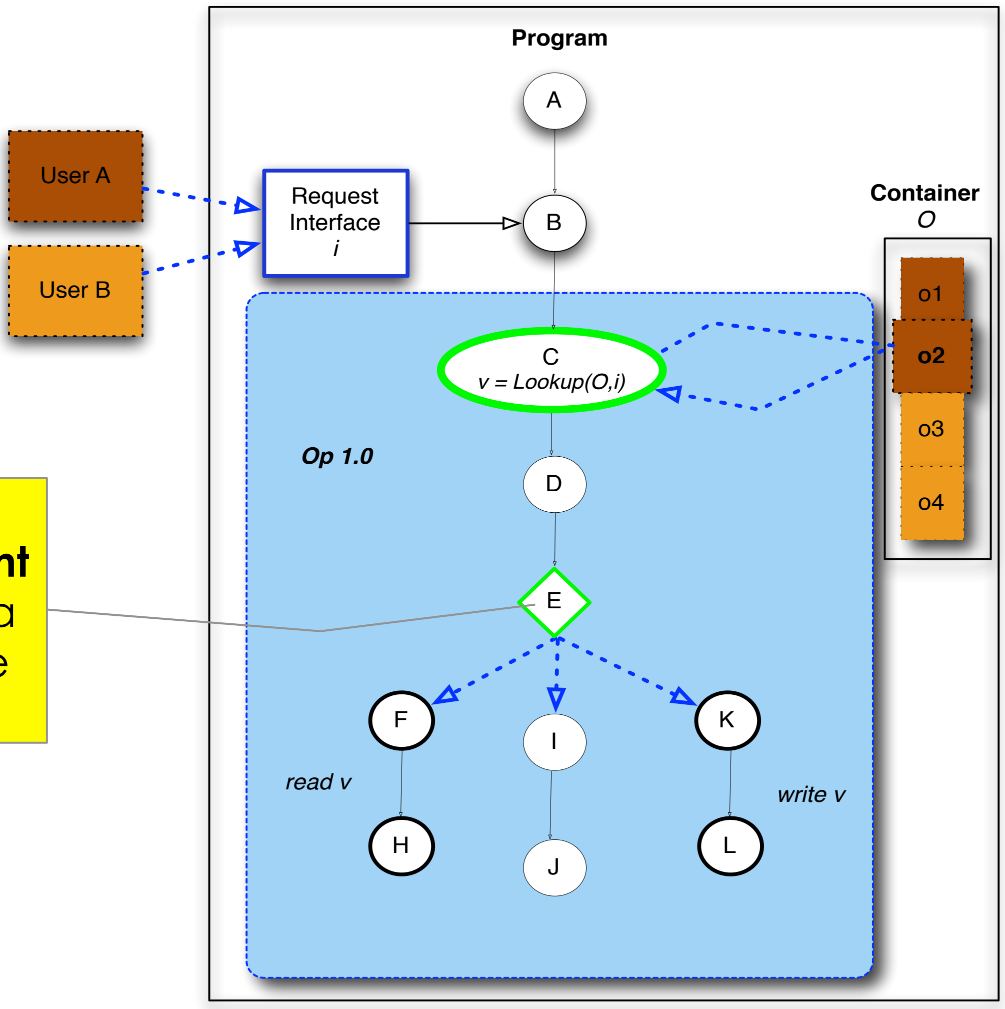
Inferring security-sensitive resources



Inferring security-sensitive operations

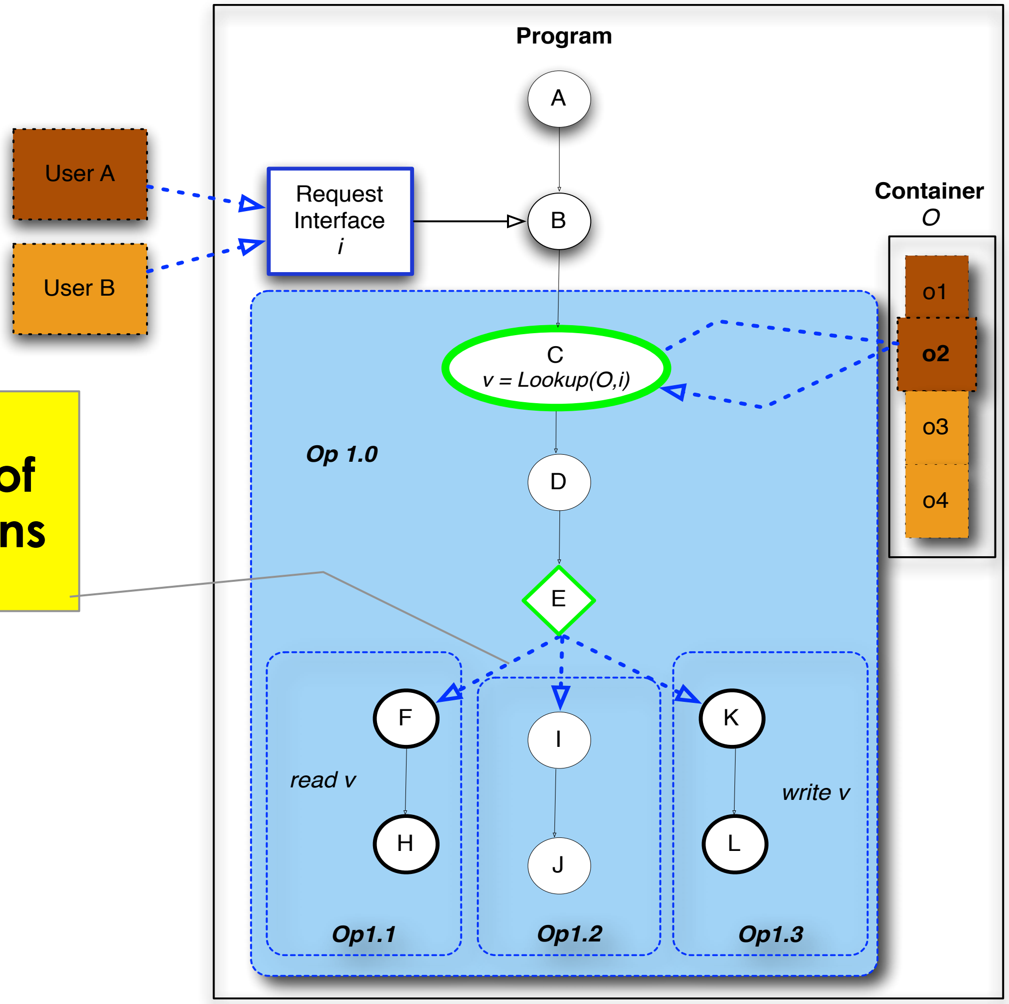


Inferring security-sensitive operations



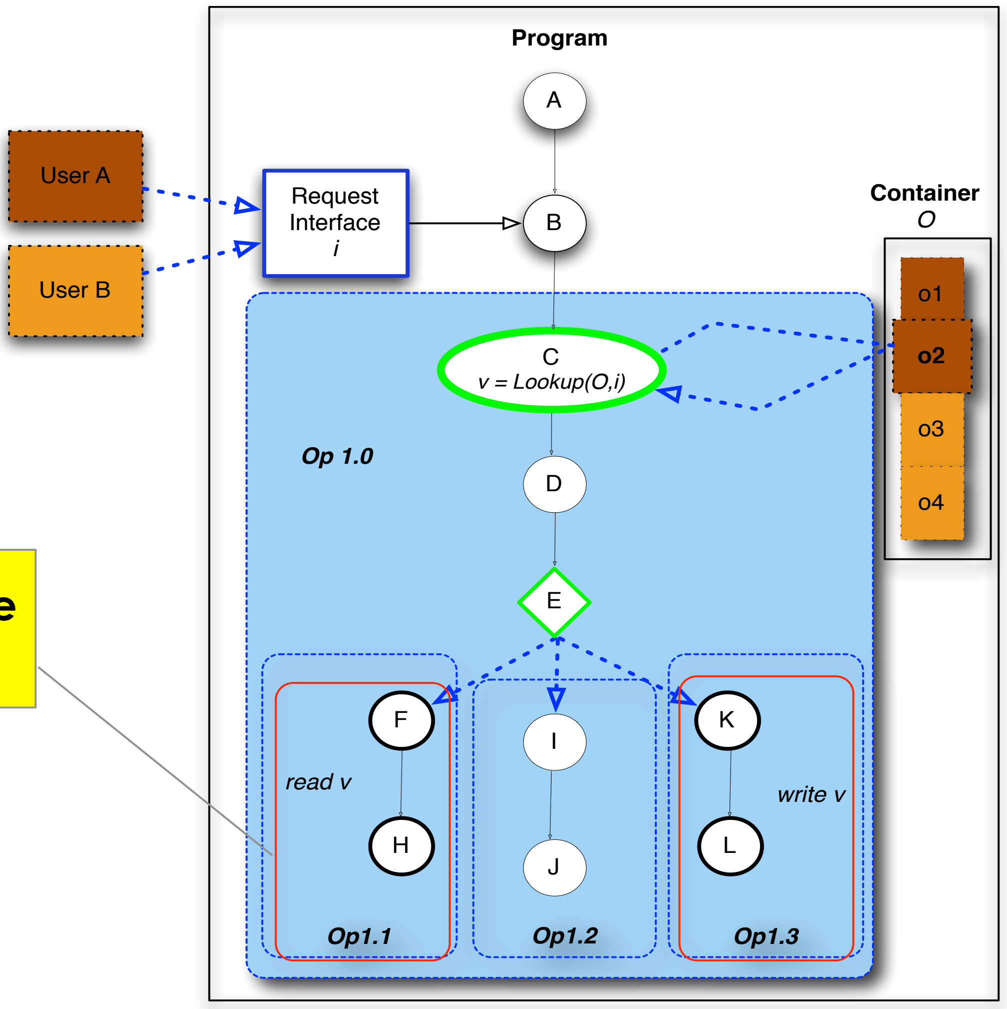
Control Statement predicated on a tainted variable

Inferring security-sensitive operations



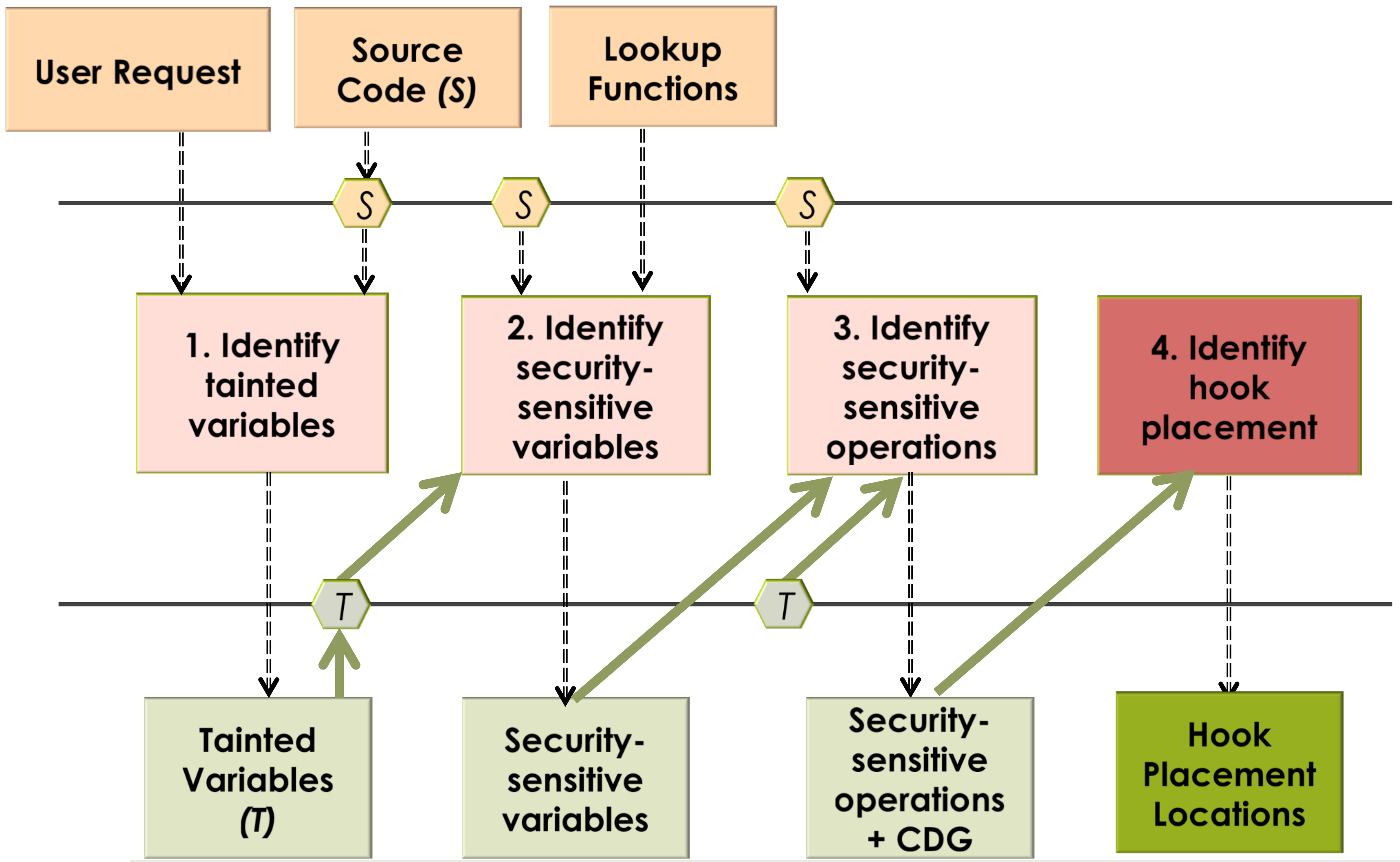
Choice of operations

Inferring security-sensitive operations



Security sensitive operation

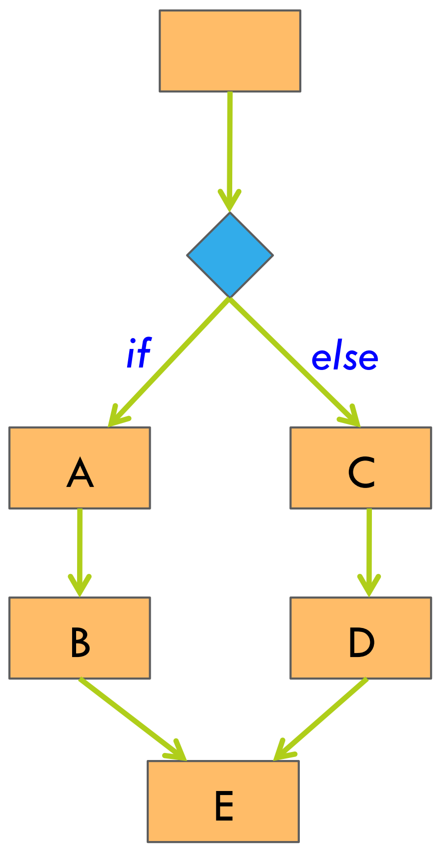
Design


User Request
Source Code (S)
Lookup Functions
S
S
S
1. Identify tainted variables
2. Identify security-sensitive variables
3. Identify security-sensitive operations
4. Identify hook placement
T
T
Tainted Variables (T)
Security-sensitive variables
Security-sensitive operations + CDG
Hook Placement Locations

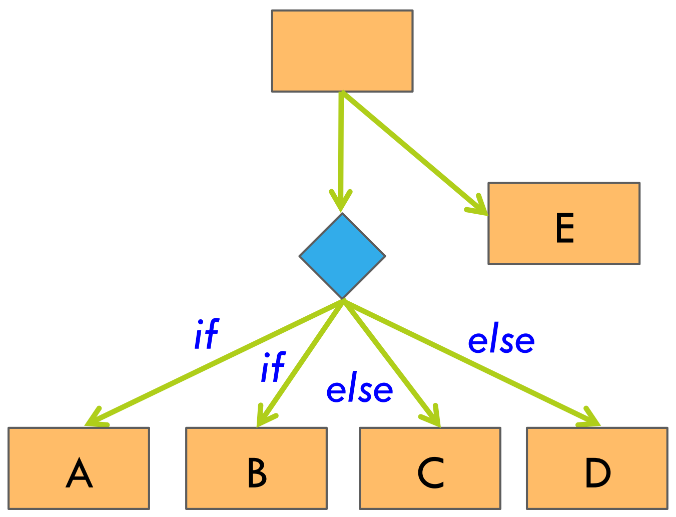
Techniques Used

- *Static taint analysis*
  - Identify variables tainted by user request.
  - Identify security-sensitive objects.
- *Control dependence analysis*
  - Identify security-sensitive operations.
  - Hoist and remove redundant hooks.

Control Dependence

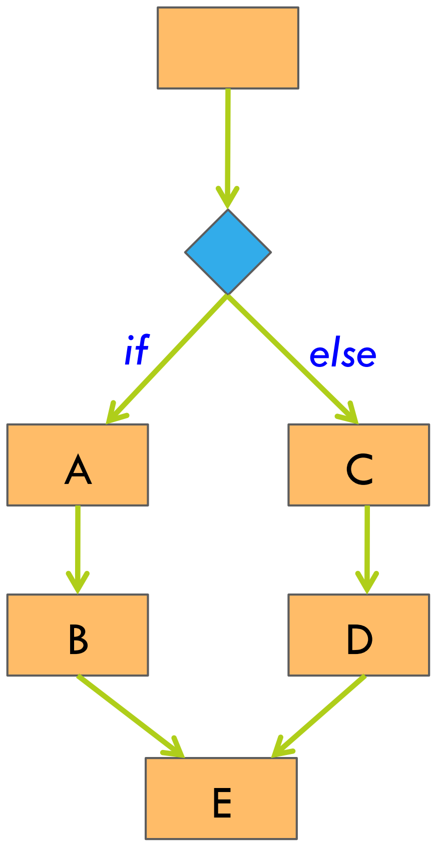


Control Flow Graph (CFG)

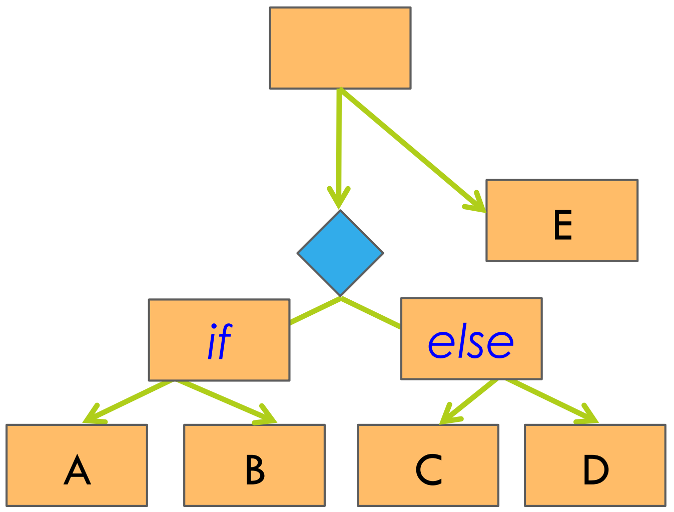


Control Dependence Graph (CDG)

Control Dependence

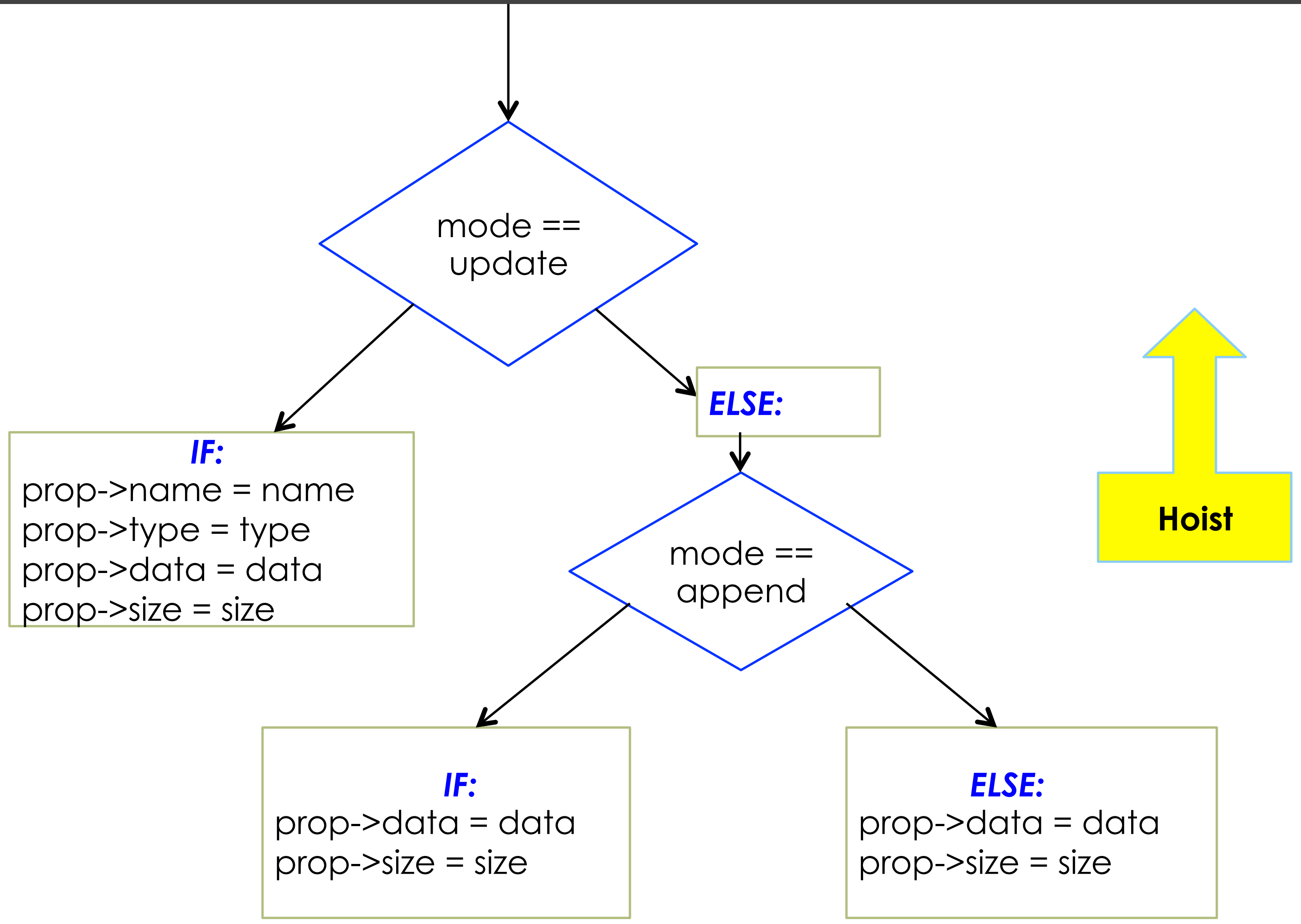


Control Flow Graph (CFG)

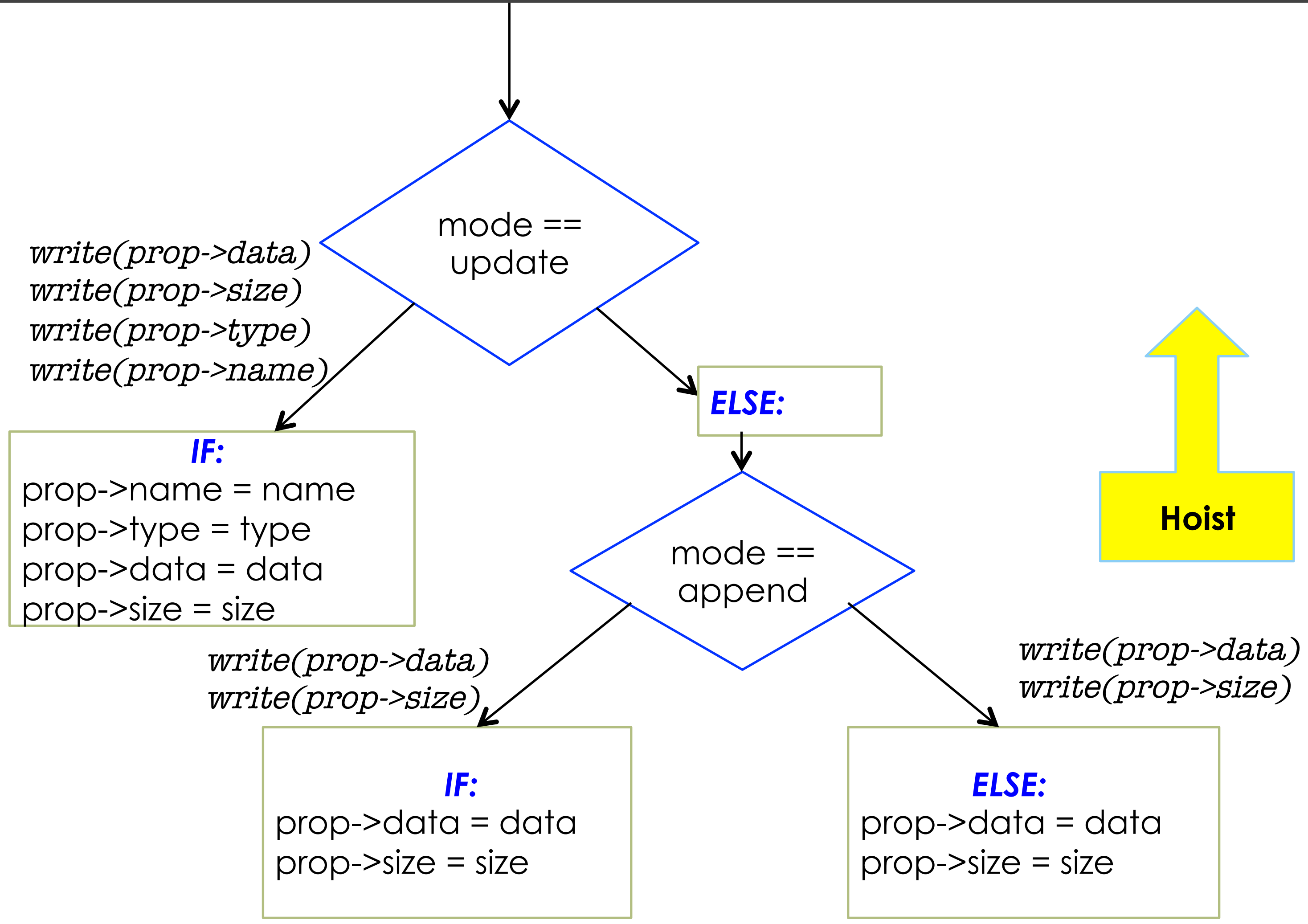


Control Dependence Graph (CDG)

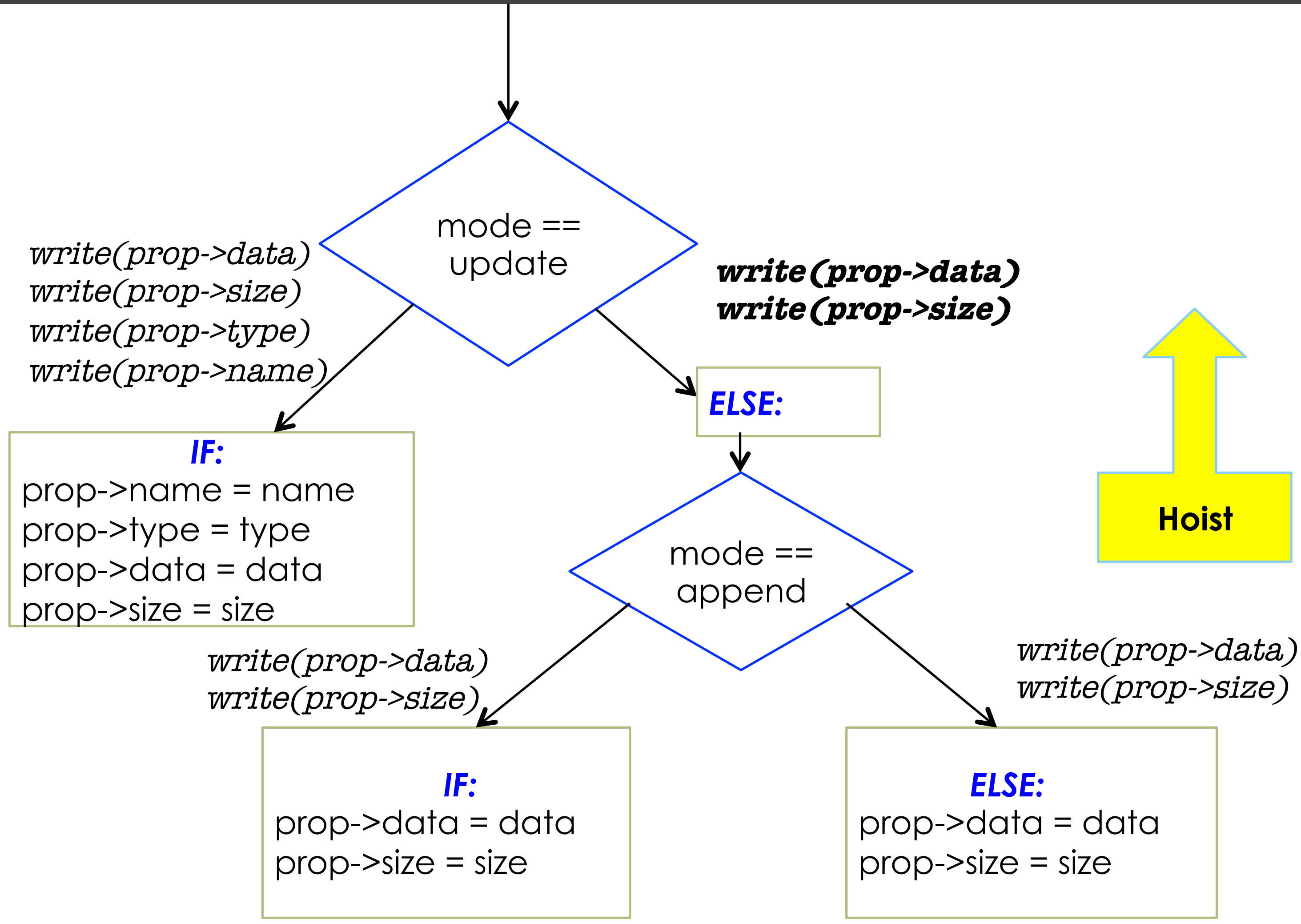
Non-redundant Hook Placement



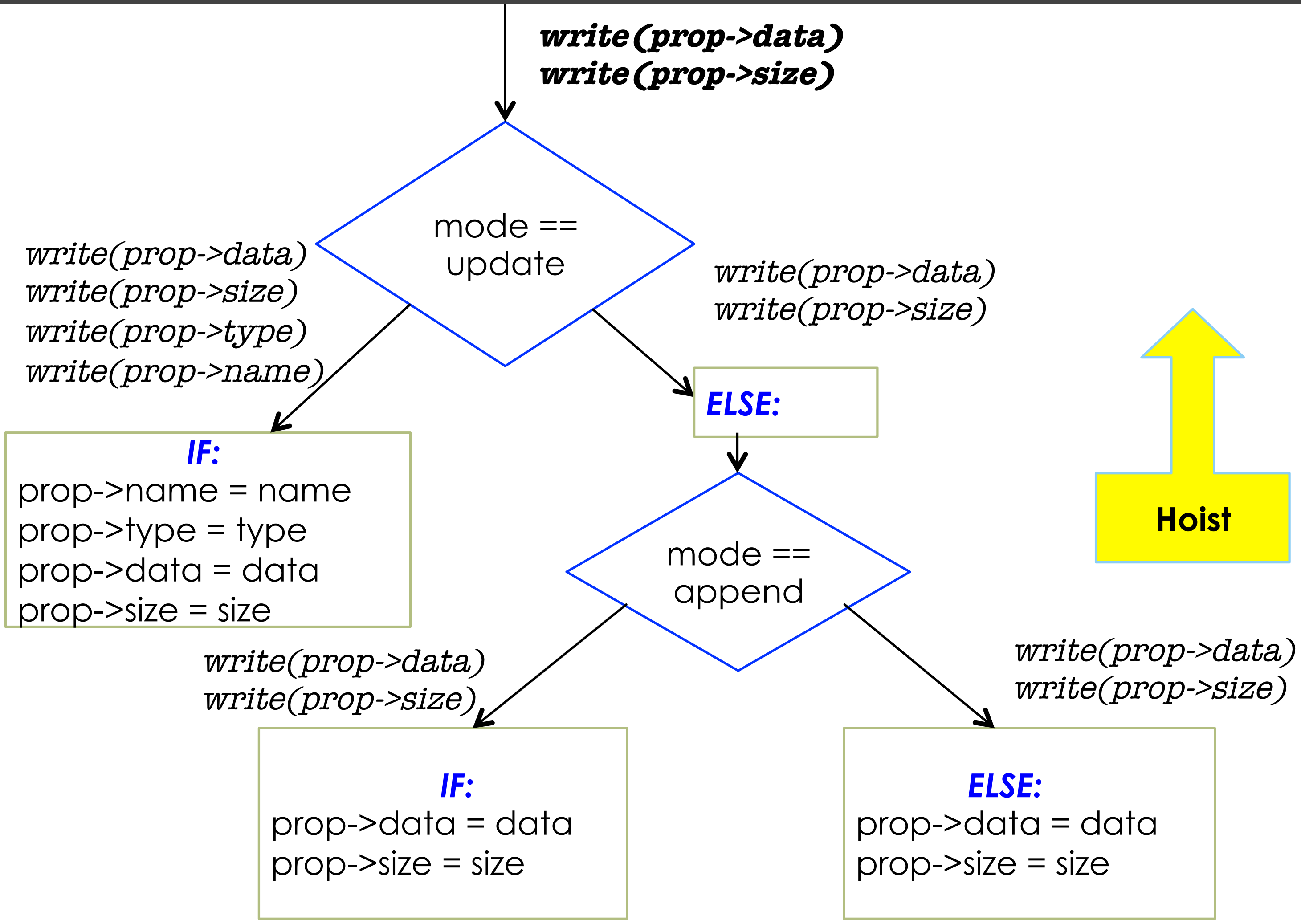
Non-redundant Hook Placement



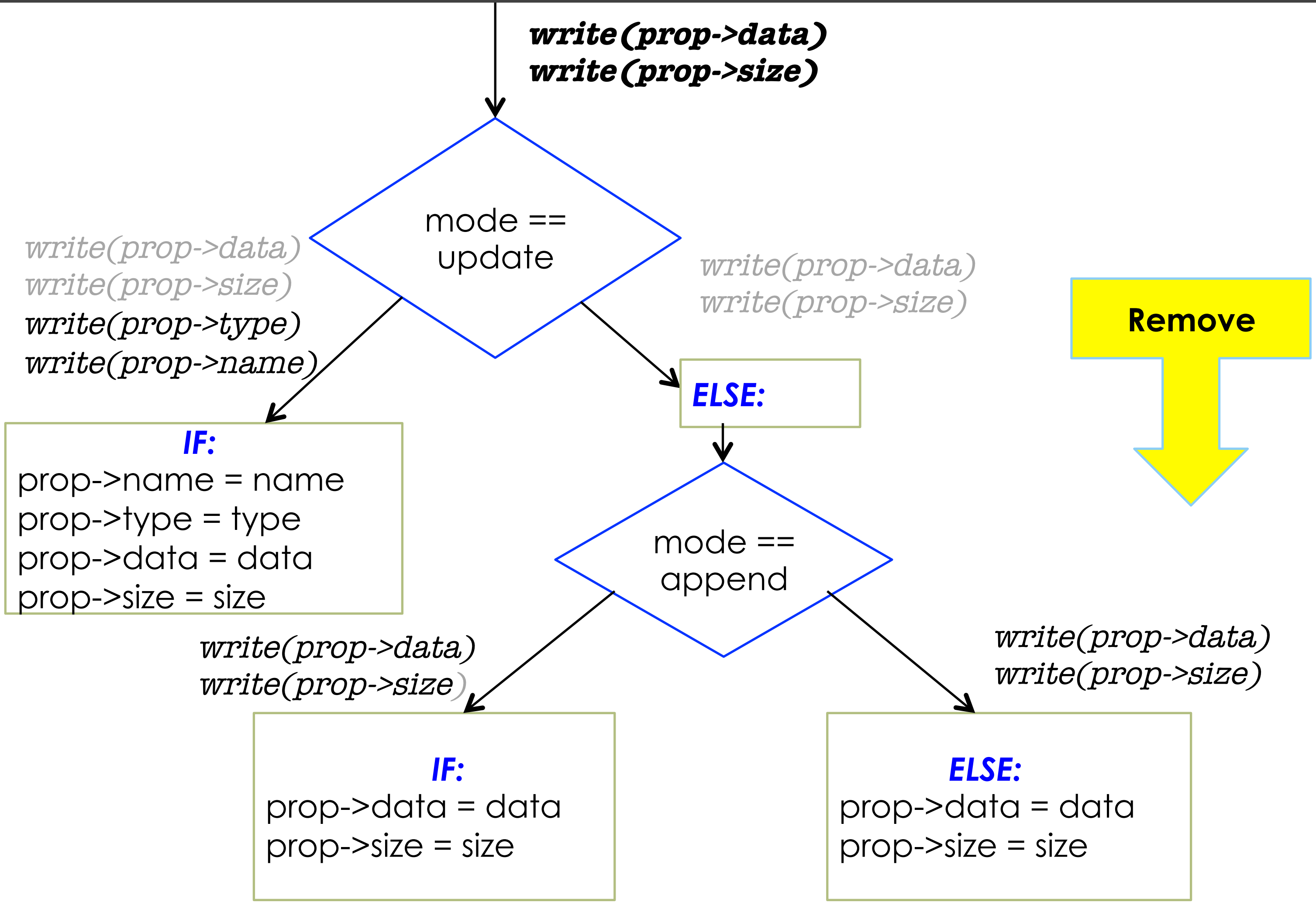
Non-redundant Hook Placement



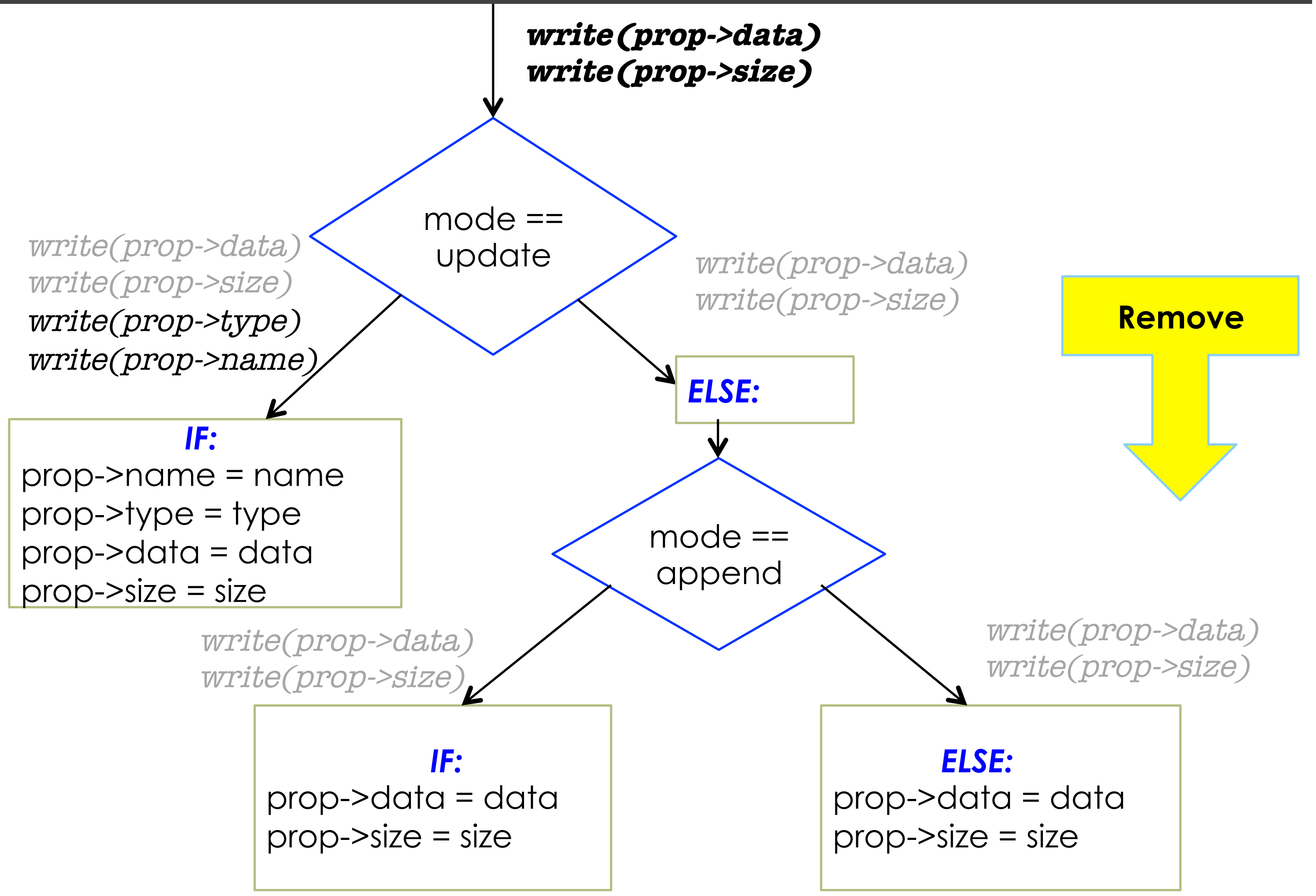
Non-redundant Hook Placement



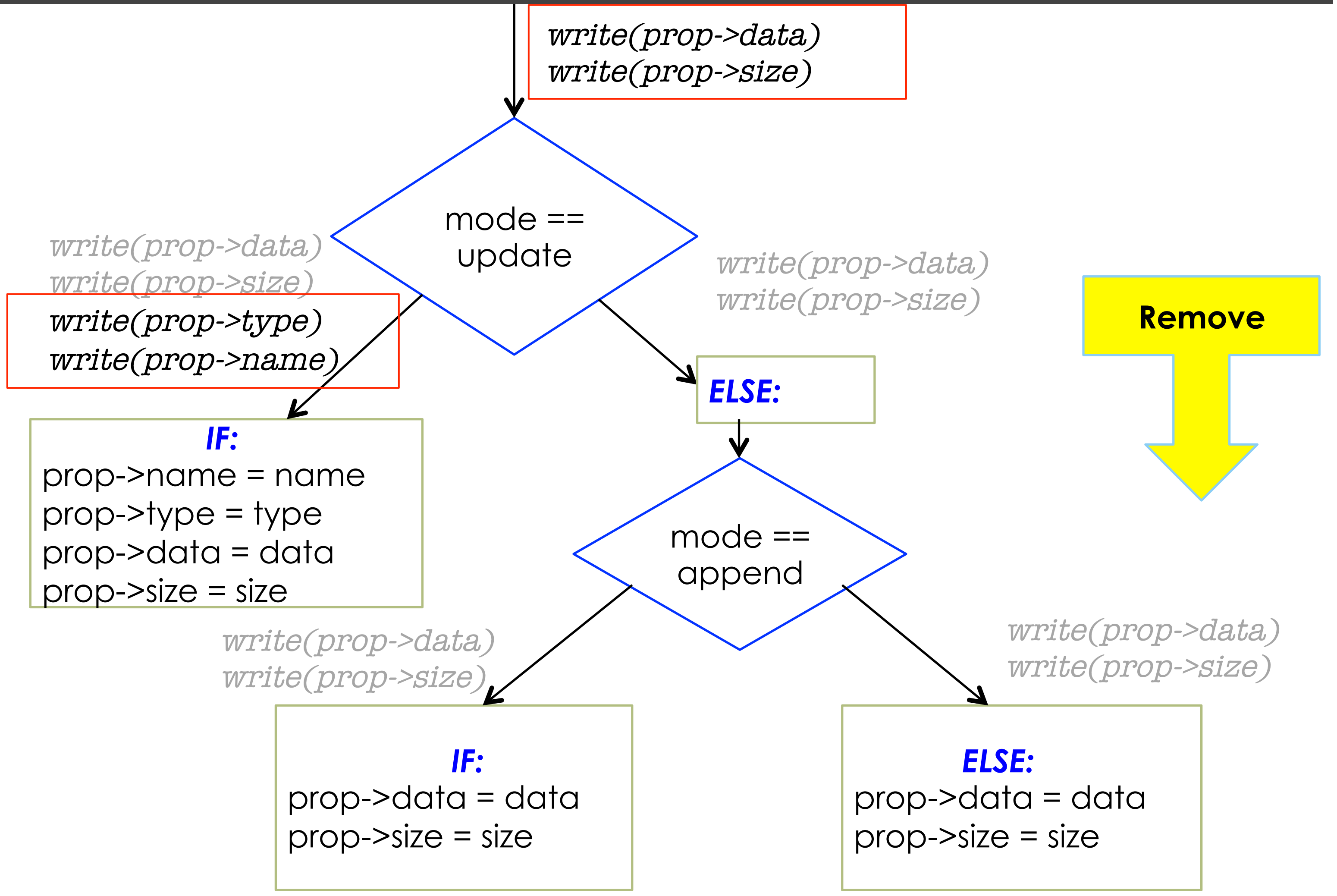
Non-redundant Hook Placement



Non-redundant Hook Placement



Non-redundant Hook Placement



Results

Program	X Server	postgres	pennmush	memcached
LOC	28k	49k	78k	9k
Total variables	*7795*	*12350*	*24372*	*2350*
Tainted variables	2975 (38%)	5100 (41%)	4168 (17%)	490 (20%)
Security sensitive variables	823 (10%)	402 (3%)	1573 (6%)	82 (3%)
Data Structures	*404*	*278*	*311*	*41*
Sensitive Data structures	61(15%)	30 (10%)	38 (12%)	7 (17%)
User-choice Operations	*4760*	*5063*	*6485*	*996*
Sensitive operations	1382 (29%)	1378 (27%)	1382 (21%)	203 (20%)
Hooks	532 (11%)	579 (11%)	714 (11%)	56 (5%)

Results

Program	X Server	postgres	pennmush	memcached
LOC	28k	49k	78k	9k
Total variables	*7795*	*12350*	*24372*	*2350*
Tainted variables	2975 (38%)	5100 (41%)	4168 (17%)	490 (20%)
Security sensitive variables	**823 (10%)**	**402 (3%)**	**1573 (6%)**	**82 (3%)**
Data Structures	*404*	*278*	*311*	*41*
Sensitive Data structures	61(15%)	30 (10%)	38 (12%)	7 (17%)
User-choice Operations	*4760*	*5063*	*6485*	*996*
Sensitive operations	**1382 (29%)**	**1378 (27%)**	**1382 (21%)**	**203 (20%)**
Hooks	532 (11%)	579 (11%)	714 (11%)	56 (5%)

Results

Program	X Server	postgres	pennmush	memcached
LOC	28k	49k	78k	9k
Total variables	*7795*	*12350*	*24372*	*2350*
Tainted variables	2975 (38%)	5100 (41%)	4168 (17%)	490 (20%)
Security sensitive variables	823 (10%)	402 (3%)	1573 (6%)	82 (3%)
Data Structures	*404*	*278*	*311*	*41*
Sensitive Data structures	61(15%)	30 (10%)	38 (12%)	7 (17%)
User-choice Operations	*4760*	*5063*	*6485*	*996*
Sensitive operations	1382 (29%)	1378 (27%)	1382 (21%)	203 (20%)
Hooks	**532 (11%)**	**579 (11%)**	**714 (11%)**	**56 (5%)**

Results

Program	X Server	postgres	pennmush	memcached
LOC	28k	49k	78k	9k
Total variables	*7795*	*12350*	*24372*	*2350*
Tainted variables	2975 (38%)	5100 (41%)	4168 (17%)	490 (20%)
Security sensitive variables	823 (10%)	402 (3%)	1573 (6%)	82 (3%)
Data Structures	[illegible]	[illegible]	*311*	*41*
Sensitive Data structures	61 (15%)	30 (10%)	38 (12%)	7 (17%)
User-choice Operations	*4760*	*5063*	*6485*	*996*
Sensitive operations	1382 (29%)	1378 (27%)	1382 (21%)	203 (20%)
Hooks	**532 (11%)**	**579 (11%)**	**714 (11%)**	**56 (5%)**

90+% effort reduction

Limitations

- Alias Analysis:
  - Cannot prove *minimality* without *complete* alias analysis.
  - Cannot prove *completeness* without assuming *sound* alias analysis.
- Implicit flows:
  - Typically cause an unwieldy number of false positives in static analysis.

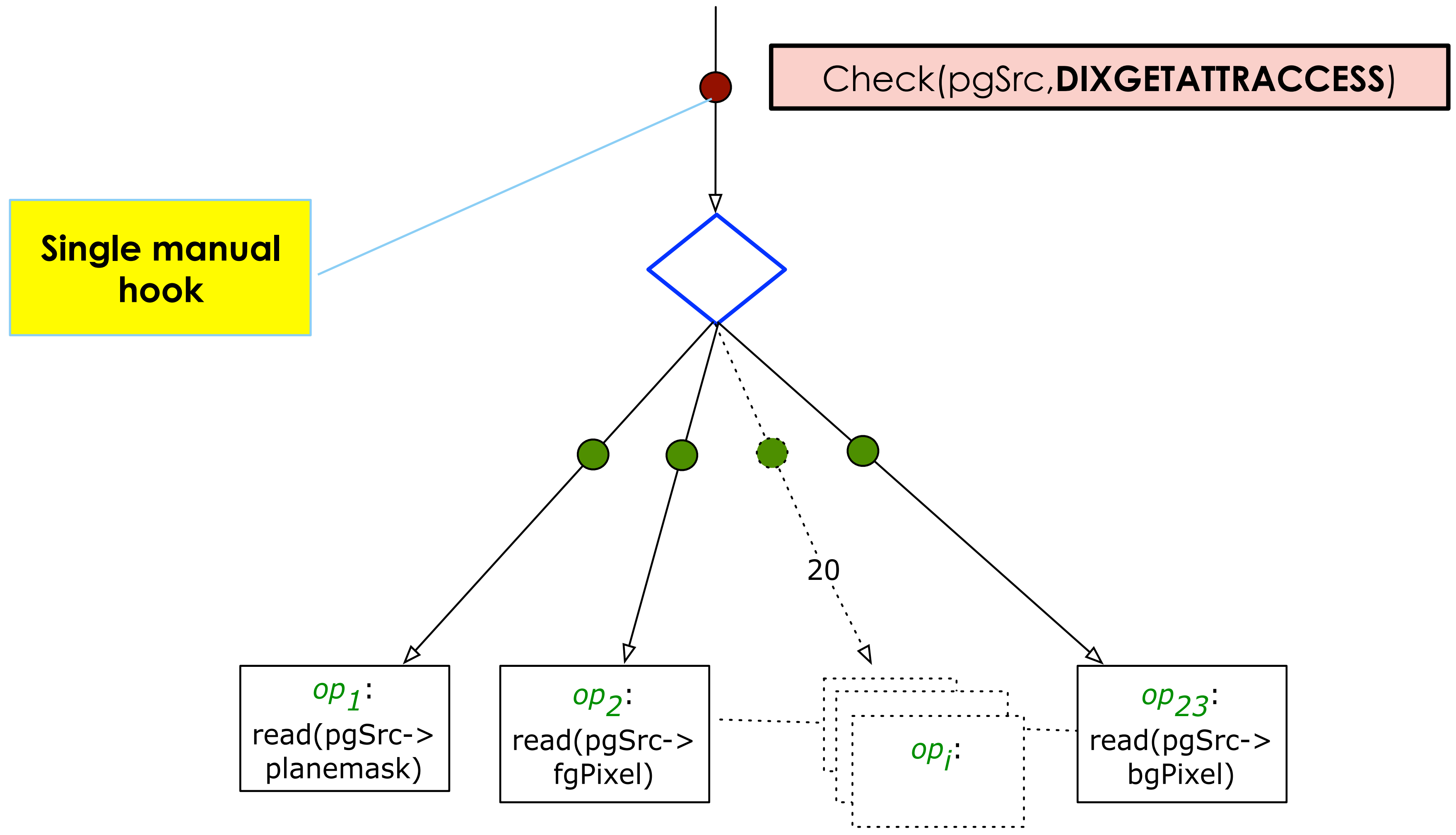
Comparison to Manual Hooks

Automated hook placement finer-grained than manual placement

- X Server (version 1.9 with XACE hooks):
  - Manual: 207 hooks
  - Automated: 532 hooks
- Postgres (version 9.0 with sepgql hooks):
  - Manual: 370 hooks
  - Automated: 579 hooks

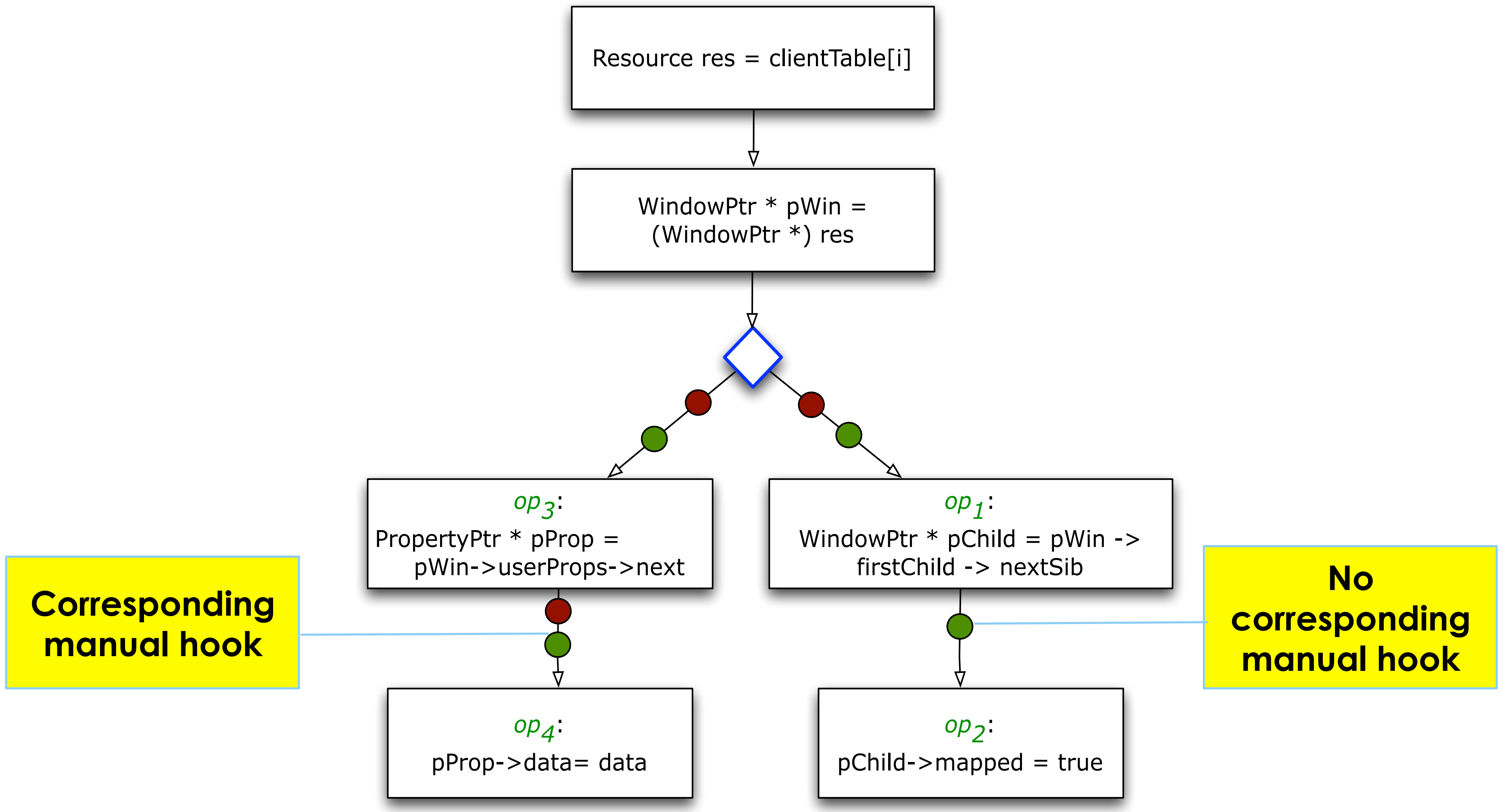
Claim: Placements must be *minimal* with respect to the *expected* authorization *policy*.

Comparison to Manual Hooks


Check(pgSrc,**DIXGETATTRACCESS**)
Single manual hook
20
op$_1$: read(pgSrc-> planemask)
op$_2$: read(pgSrc-> fgPixel)
op$_i$:
op$_{23}$: read(pgSrc-> bgPixel)

CopyGC @ gc.c (X Server 1.1.3)

Comparison to Manual Hooks



Resource res = clientTable[i]
WindowPtr * pWin = (WindowPtr *) res
op_3: PropertyPtr * pProp = pWin->userProps->next
op_1: WindowPtr * pChild = pWin -> firstChild -> nextSib
op_4: pProp->data= data
op_2: pChild->mapped = true
Corresponding manual hook
No corresponding manual hook

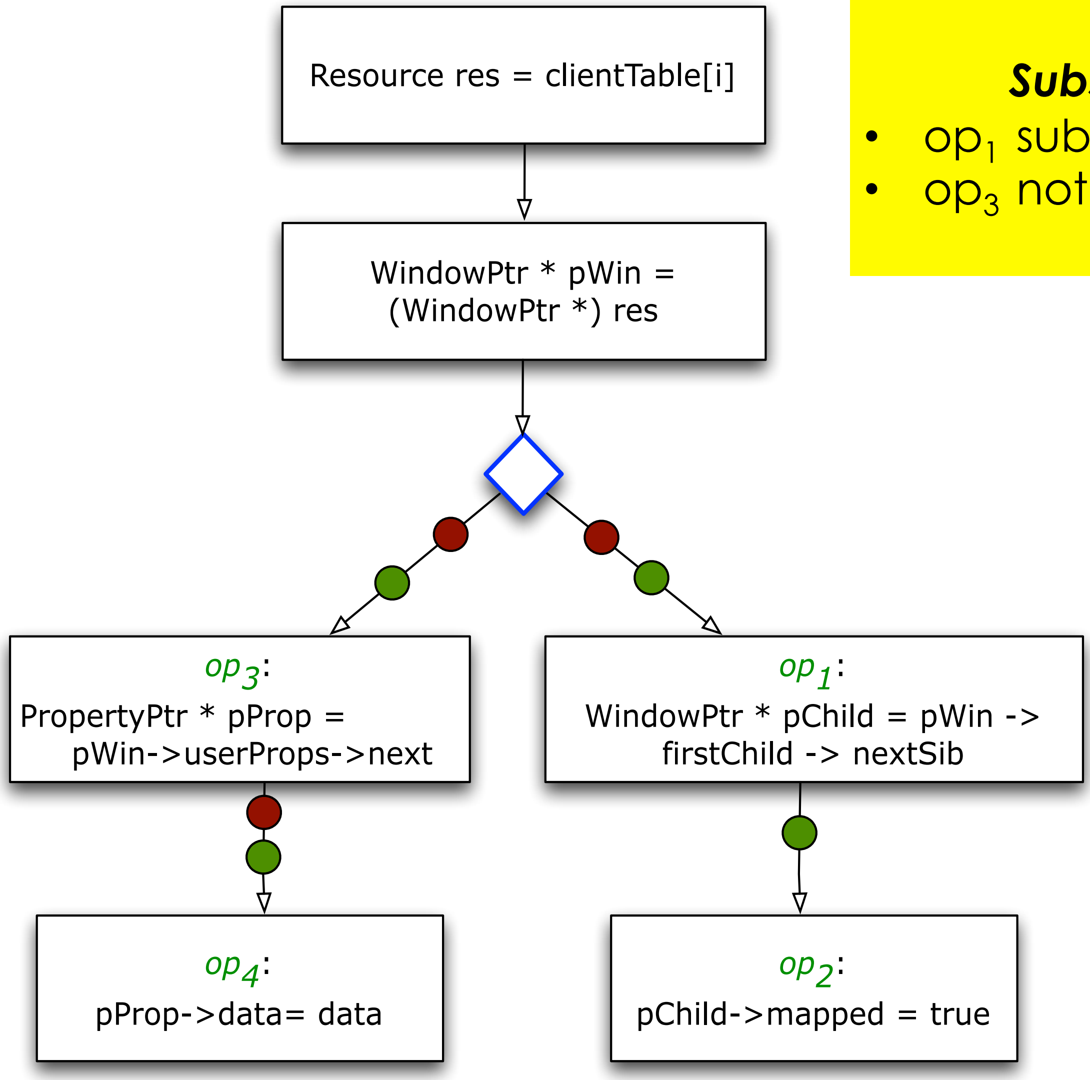
Authorization Constraints

- *Allowed(o)*: Subset of subjects in *U* that are allowed to perform operation o.
- Constraint I:
  - *Allowed(o1) = Allowed(o2)*, then ***o1 equals o2***
- Constraint II:
  - *Allowed(o1)* $\subset$ *Allowed(o2)*, then ***o1 subsumes o2***

Authorization Constraints

- *Allowed(o)*: Subset of subjects in *U* that are allowed to perform operation o.
  - Suppose operation o1 control-flow dominates operation o2
- Constraint I:
  - *Allowed(o1) = Allowed(o2)*, then ***o1 equals o2***
- Constraint II:
  - *Allowed(o1)* ⊂ *Allowed(o2)*, then ***o1 subsumes o2***

Constraints: *Subsumption*



Resource res = clientTable[i]
WindowPtr * pWin = (WindowPtr *) res
op3:
PropertyPtr * pProp = pWin->userProps->next
op1:
WindowPtr * pChild = pWin -> firstChild -> nextSib
op4:
pProp->data= data
op2:
pChild->mapped = true

Subsumption:

- op_1 subsumes op_2
- op_3 not subsumes op_4

Authorization Constraints

- *Allowed(o)*: Subset of subjects in *U* that are allowed to perform operation o.
- Invariant I:
  - *Allowed(o1) = Allowed(o2)*, then ***o1 equals o2***
- Invariant II:
  - *Allowed(o1)* $\subset$ *Allowed(o2)*, then ***o1 subsumes o2***

- **Access Control Policy is *not available*.**
- **Access Control Policy is not in terms of *code level operations***

Challenges

- *How to generate a placement that minimizes hooks w.r.t authorization constraints?*
- *How do we get authorization constraints?*

Challenges

- *How to generate a placement that minimizes hooks w.r.t authorization constraints?*
  - Use authorization constraints to eliminate “redundant” hooks
- *How do we get authorization constraints?*

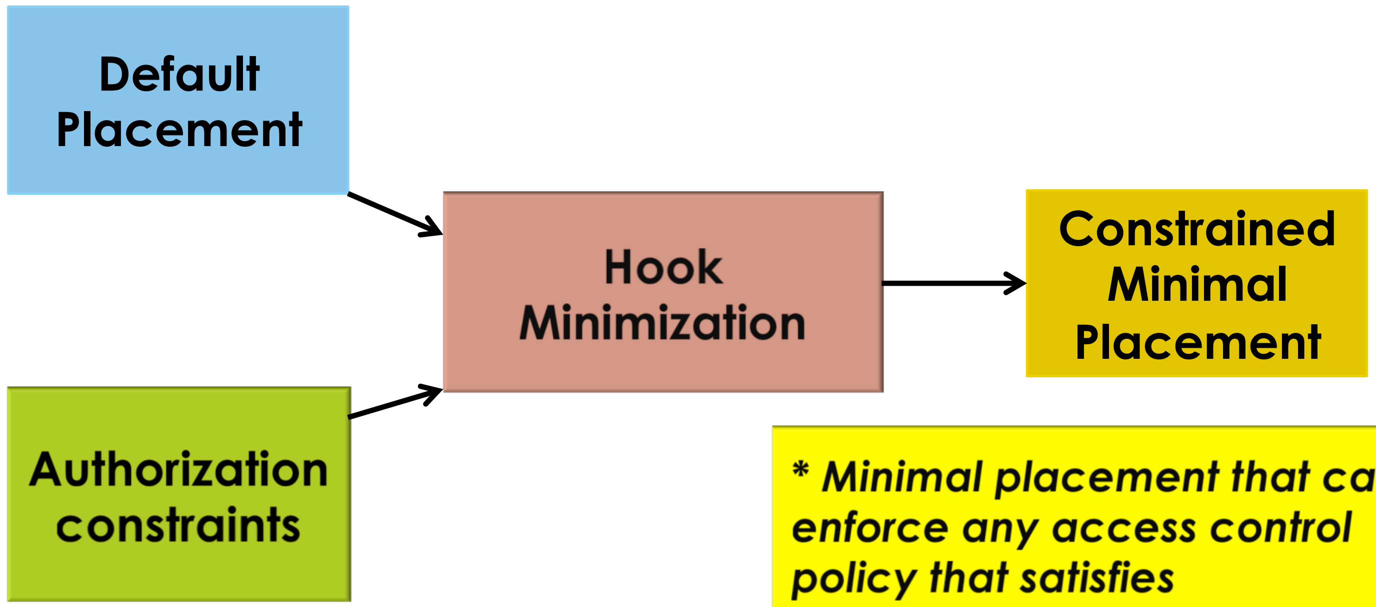
Challenges

- *How to generate a placement that minimizes hooks w.r.t. authorization constraints?*
  - Use authorization constraints to eliminate "redundant" hooks
- *How do we get authorization constraints?*
  - "Top-down"
    - Programmers propose placement and we compute authorization constraints
    - Programmers choose authorization constraints
  - "Bottom-up"
    - Start from any placement, such as computed default
    - Compute constraints relative to that placement
    - Could "select" a group of constraints that satisfy a high-level constraint automatically

Constraints and Placements

Default Placement	***Can enforce any access control policy.***
Authorization constraints	***Equivalence and subsumption relationships on operations***
Candidate Placement	***Can enforce any access control policy that satisfies authorization constraints***

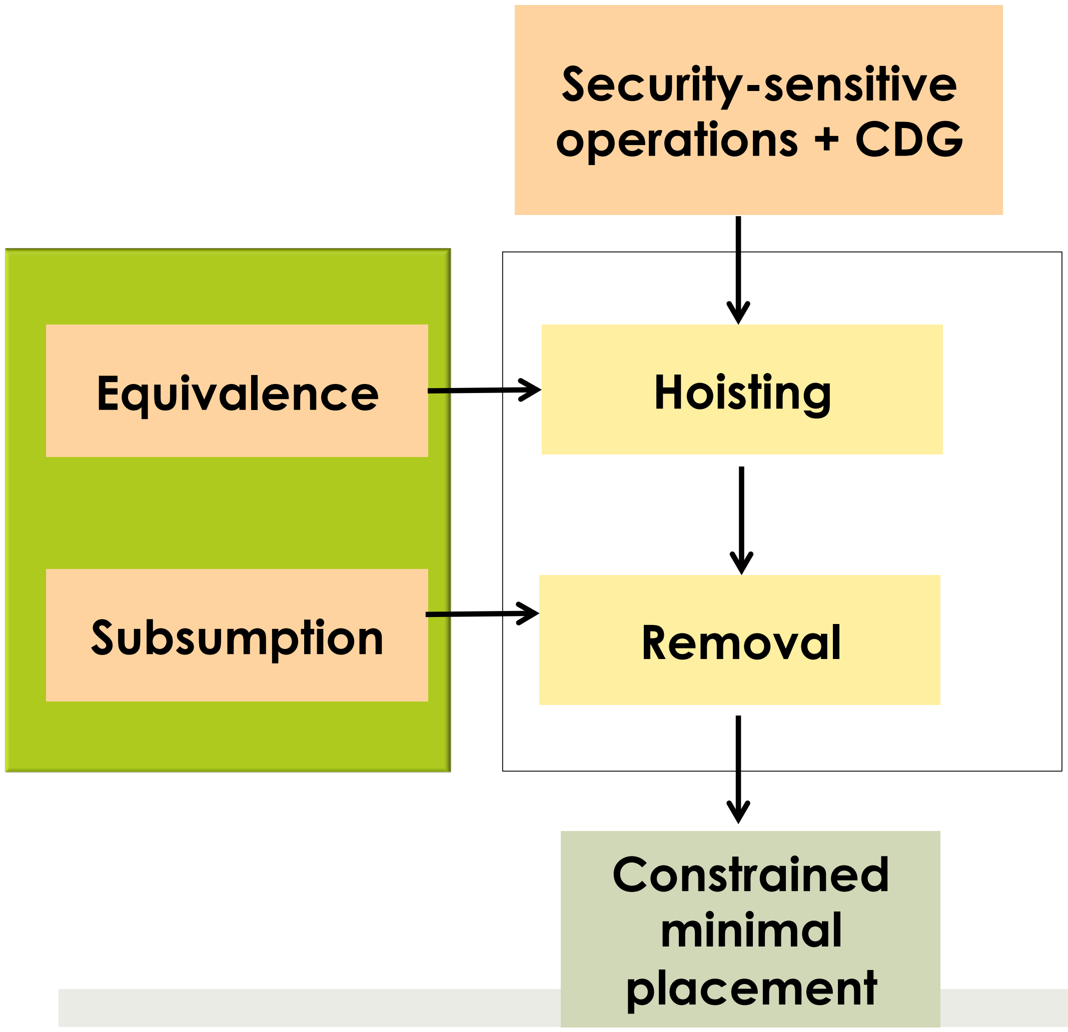
Compute Minimal Placement



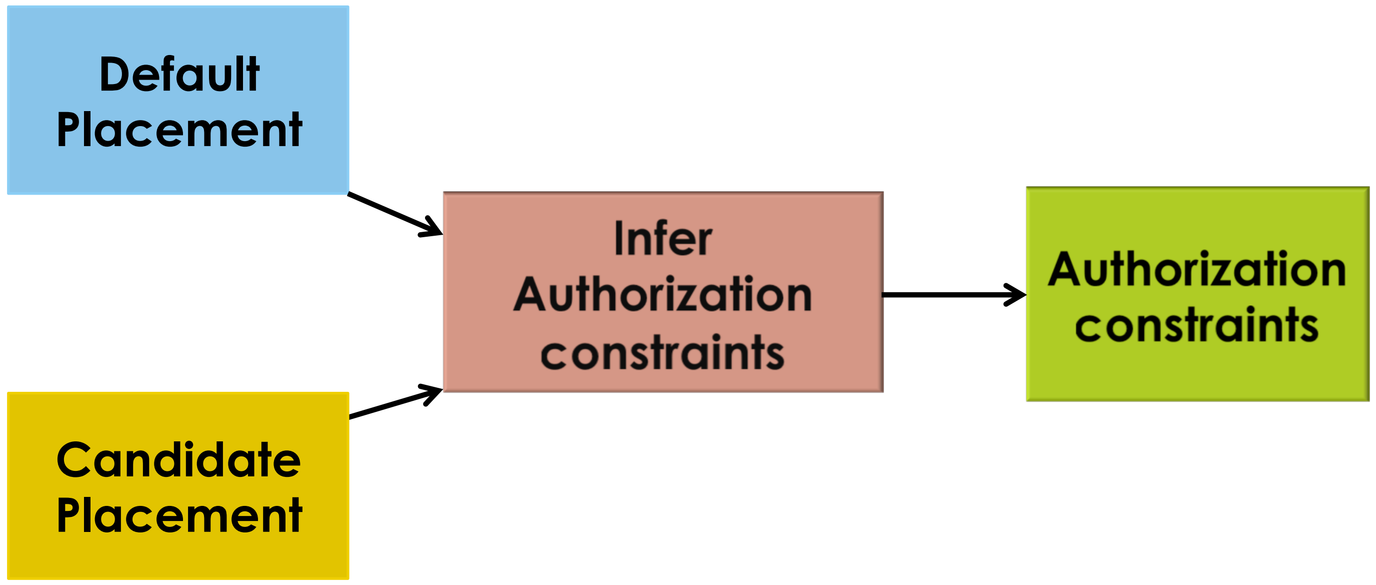
**** Minimal placement that can enforce any access control policy that satisfies authorization constraints***

Generating a minimal placement

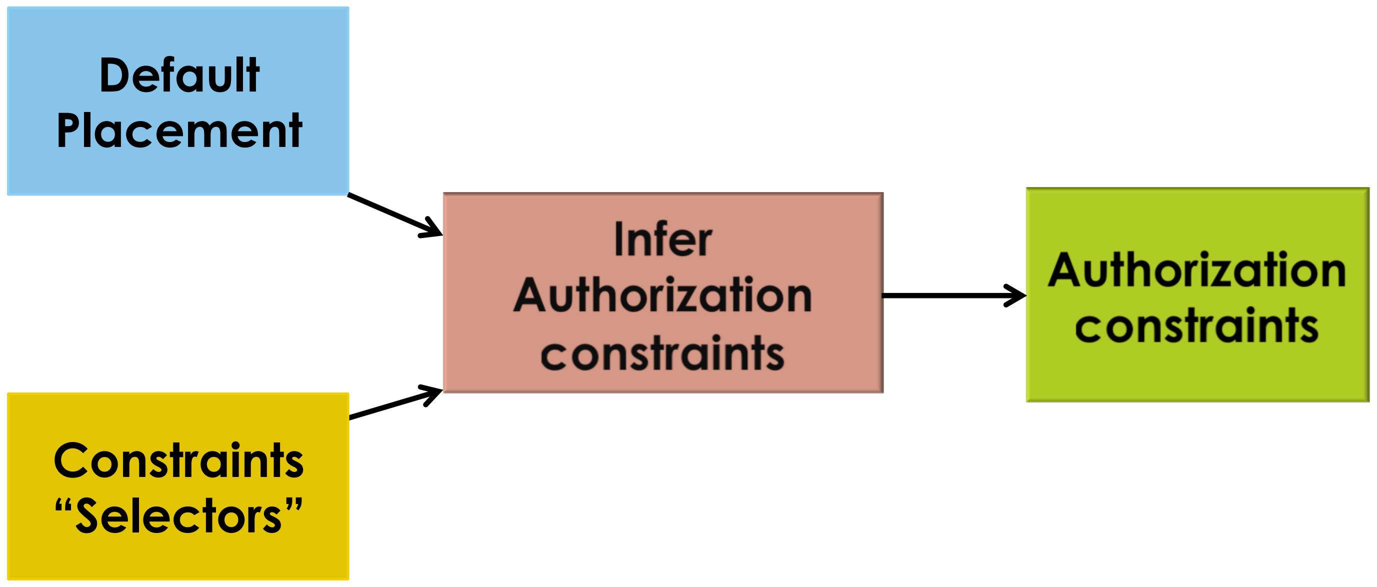
- Equivalence (Q) + Subsumption (S)


Security-sensitive operations + CDG
Equivalence
Subsumption
Hoisting
Removal
Constrained minimal placement

Top-Down: Infer Authorization Constraints



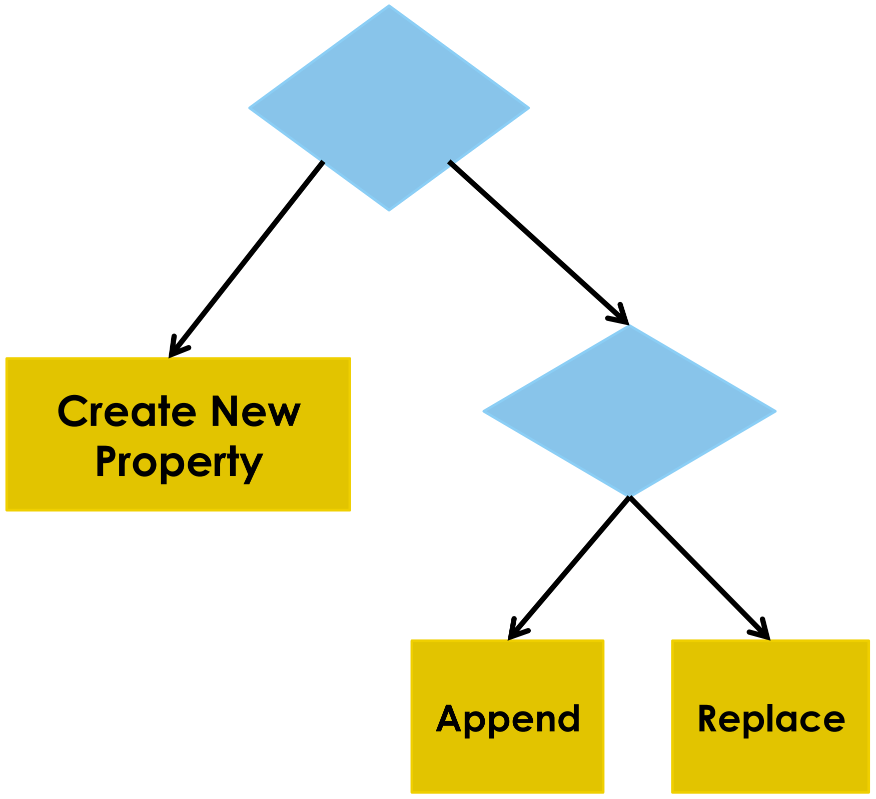
Bottom-Up: Infer Authorization Constraints



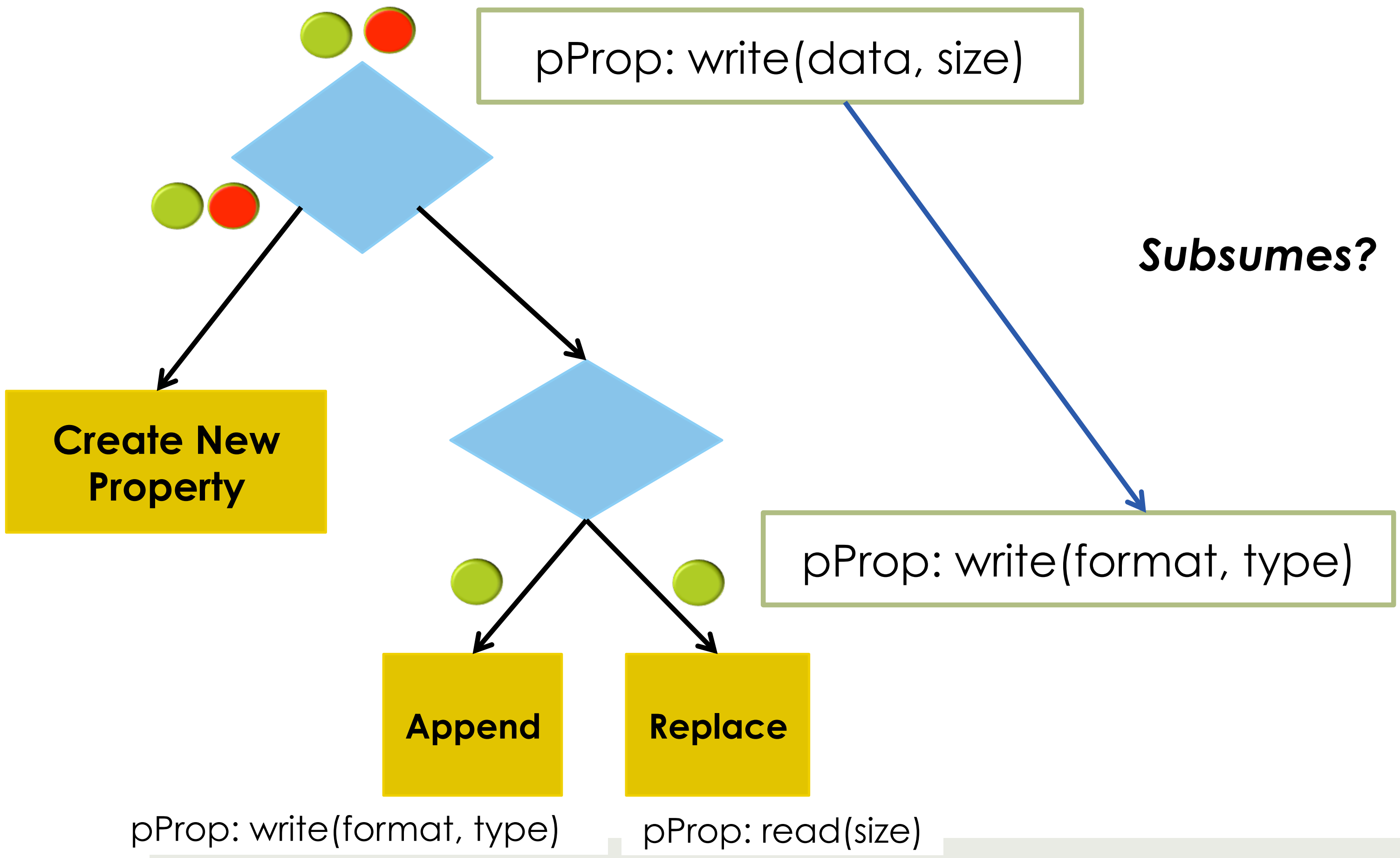
Show additional choices

- Given a placement,
  - What additional hoisting can be done?
    - Sets of control statement hooks with a common control statement.
  - What additional removal can be done?
    - Hook that have at least one dominating hook.

Additional choices



Subsumption choices



Helping programmers infer constraints

Top Down:

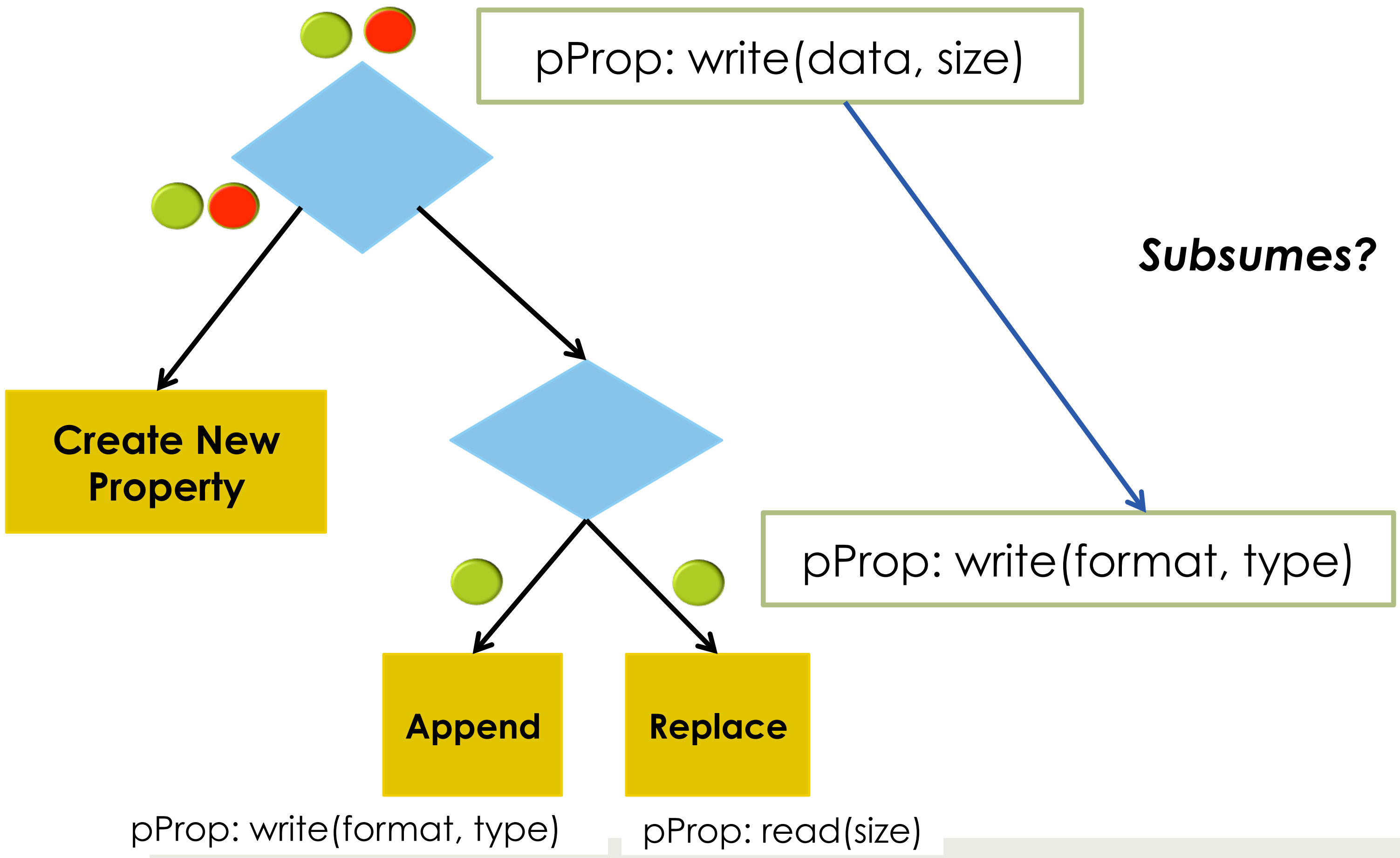
- Programmers *guess* at placement
- We show au
  implied.

Constraint Selectors:
At each hoisting or removal point apply rules to decide whether the operations are equivalent or subsuming

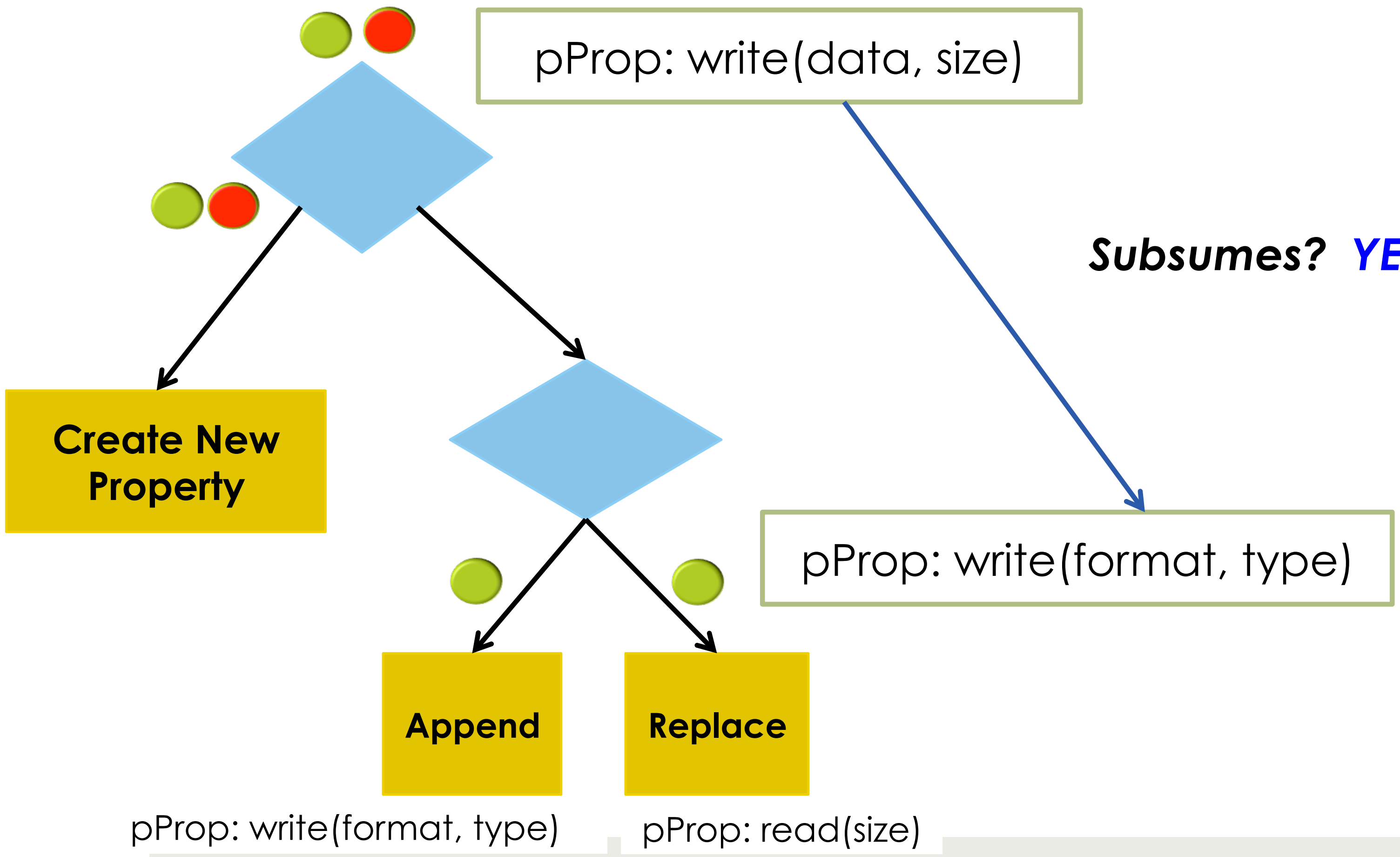
m Up:

ement (say
w what hoists and
ill happen.
- Advise the programmer about constraints that may be useful.

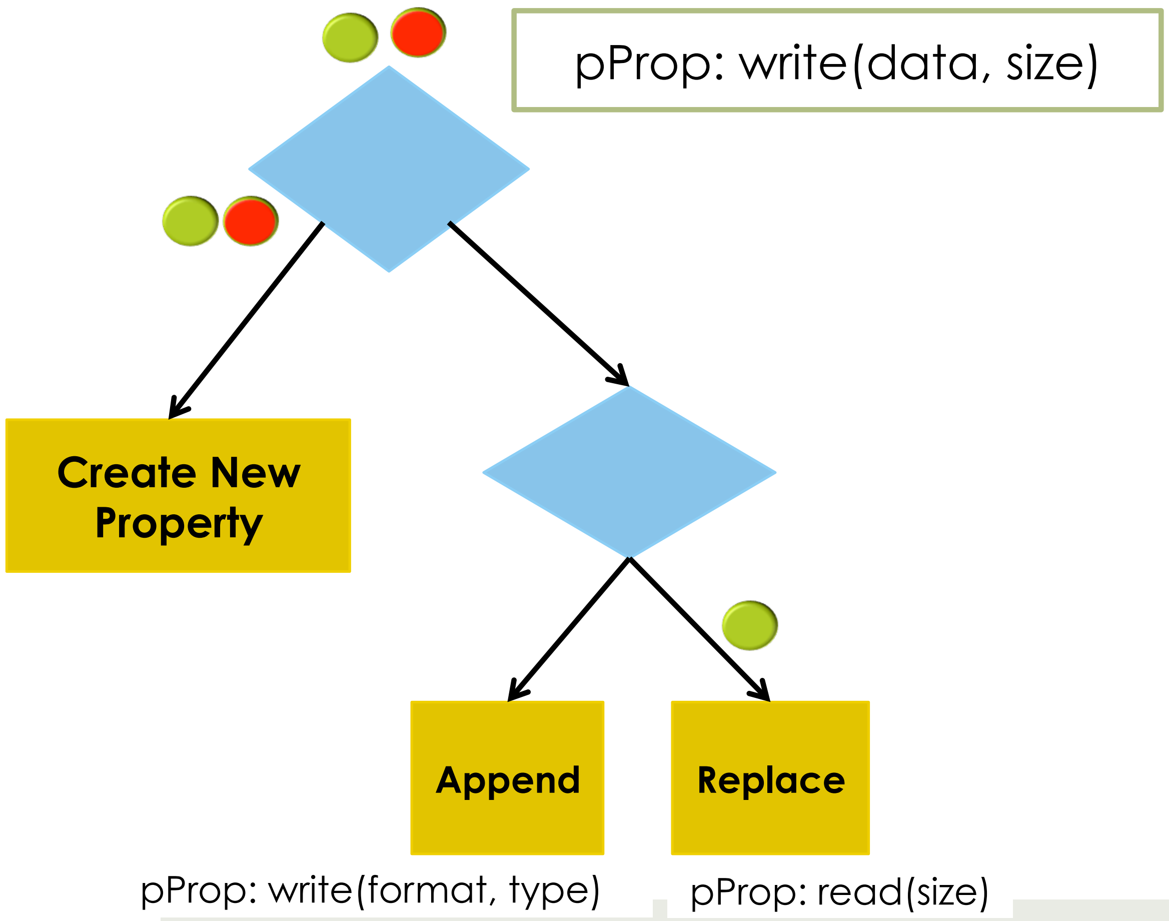
MLS Selector Example



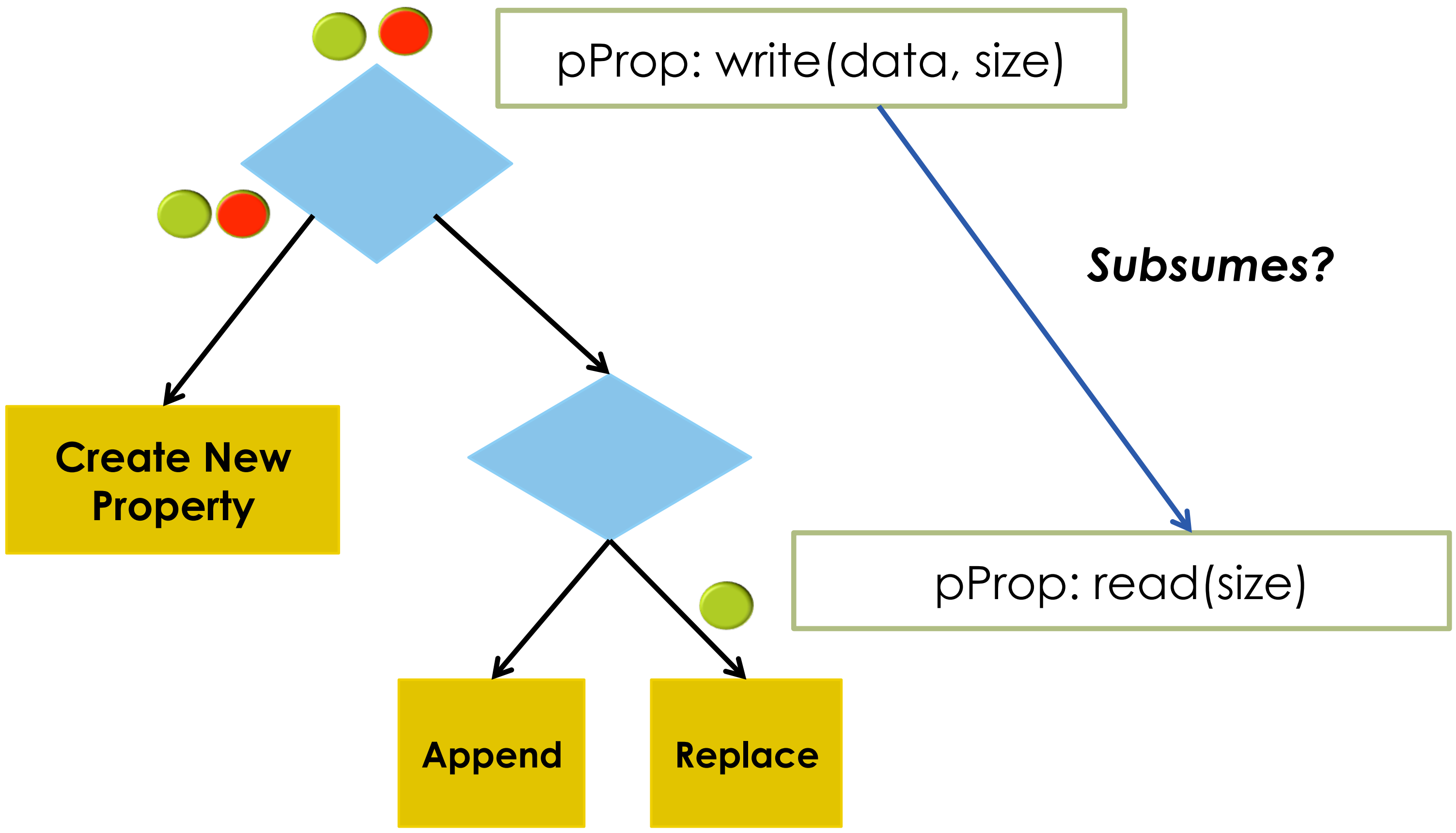
MLS Selector Example



MLS Selector Example



MLS Selector Example



Experiments

- 4 programs:
  - X Server 1.13 (*manual*)
  - Postgres 9.1.9 (*manual*)
  - Linux Kernel VFS 2.6.38.8 (*manual*)
  - memcached
- 4 selectors:
  - DEF
  - FIL-MLS
  - FIL-RUN
  - FIL-MLS + FIL-RUN

Experiments

- How many authorization constraints do programmers have to look at?
  - How do selectors help with that?
- How many hoisting and removal choices do programmers have to make?
  - How do selectors help with that?