Transforming Code to Drop Dead Privileges

Xiaoyu Hu
BitFusion.io Inc

Jie Zhou
University of Rochester

Spyridoula Gravani
University of Rochester

John Criswell
University of Rochester

Least Privilege Principle

“Every program and every user of the system should operate using the least set of privileges necessary to complete the job.”

Saltzer, Jerome H., and Schroeder, Michael D. "The protection of information in computer systems." Proceedings of the IEEE 63, no. 9 (1975): 1278-1308.

An Example of An Over-Privileged Program

`ping`: send ICMP packets to a network host

privileges needed: open a raw socket

```
socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

some other network related privileges in

```
setsockopt(icmp_sock, SOL_SOCKET, SO_DEBUG, (char *)&hold, sizeof(hold));
setsockopt(icmp_sock, SOL_SOCKET, SO_MARK, &mark, sizeof(mark));
```

However

```
$ file /bin/ping
/bin/ping: setuid ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamicall
y linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.24, BuildID[
sha1]=c525a765d86dbe1bdb61f56a497e6113871ef37b, stripped
jie@zimmer: ~
$ ls -l /bin/ping
-rwsr-xr-x 1 root root 44168 May  7  2014 /bin/ping
```

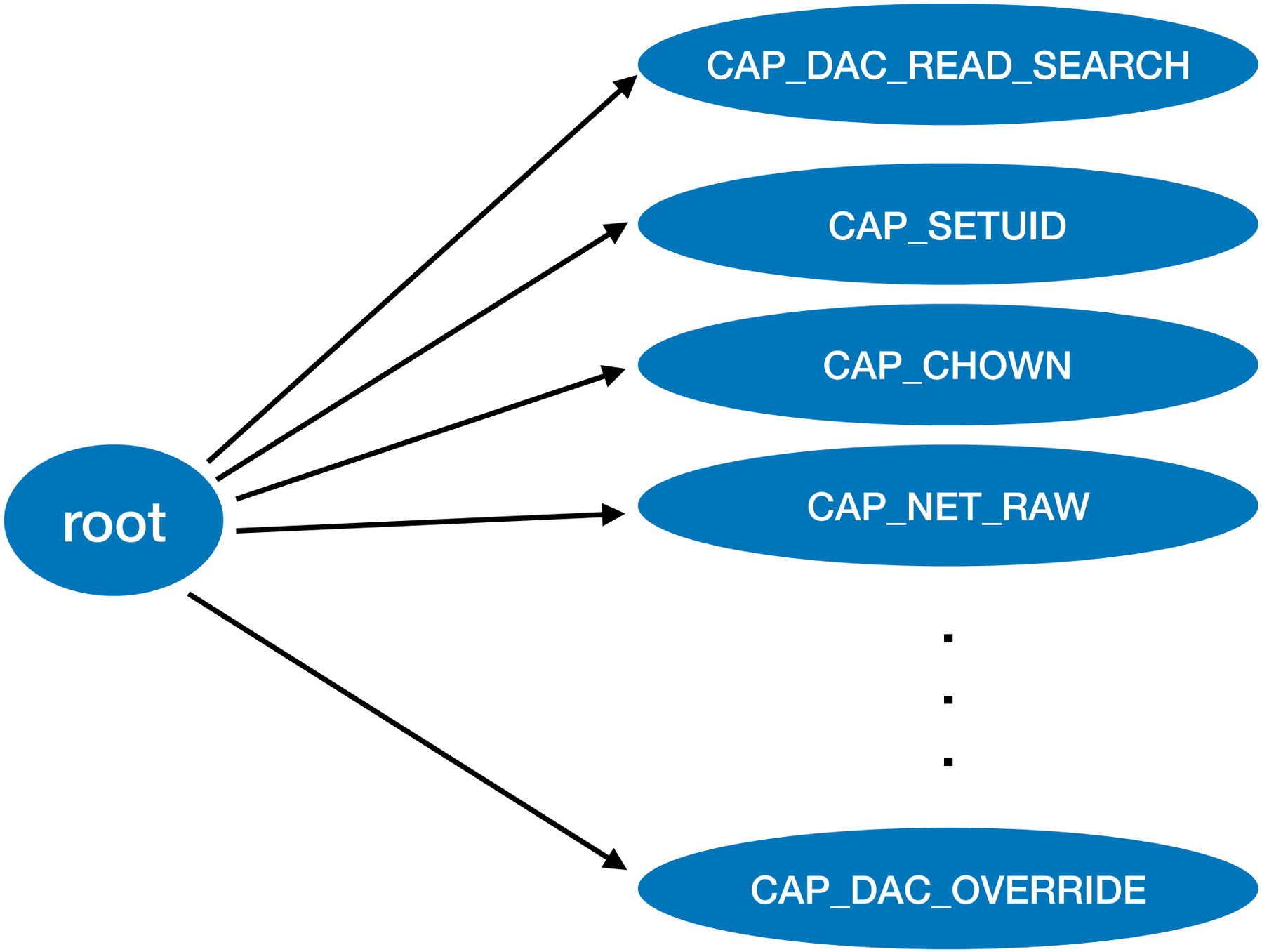
Privileged Programs

In `Linux`,

Privilege: the ability to override kernel's access control rule

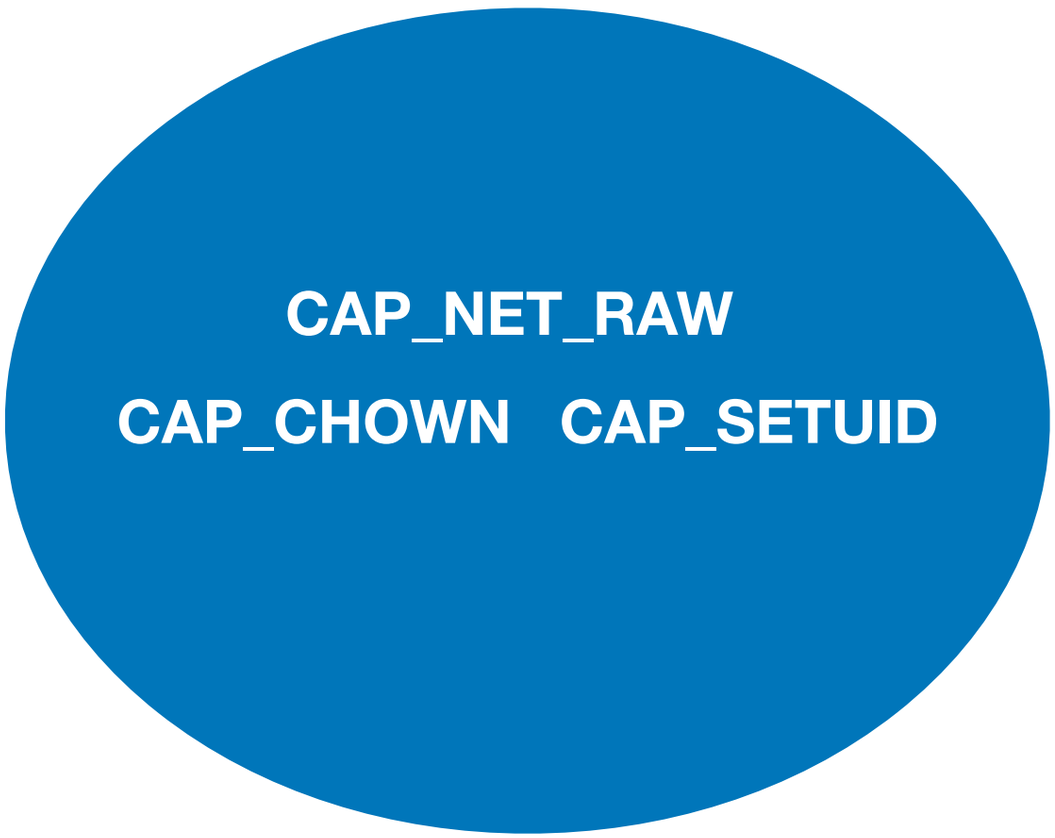
Privileged Program: program with one or more privileges

Linux Capabilities

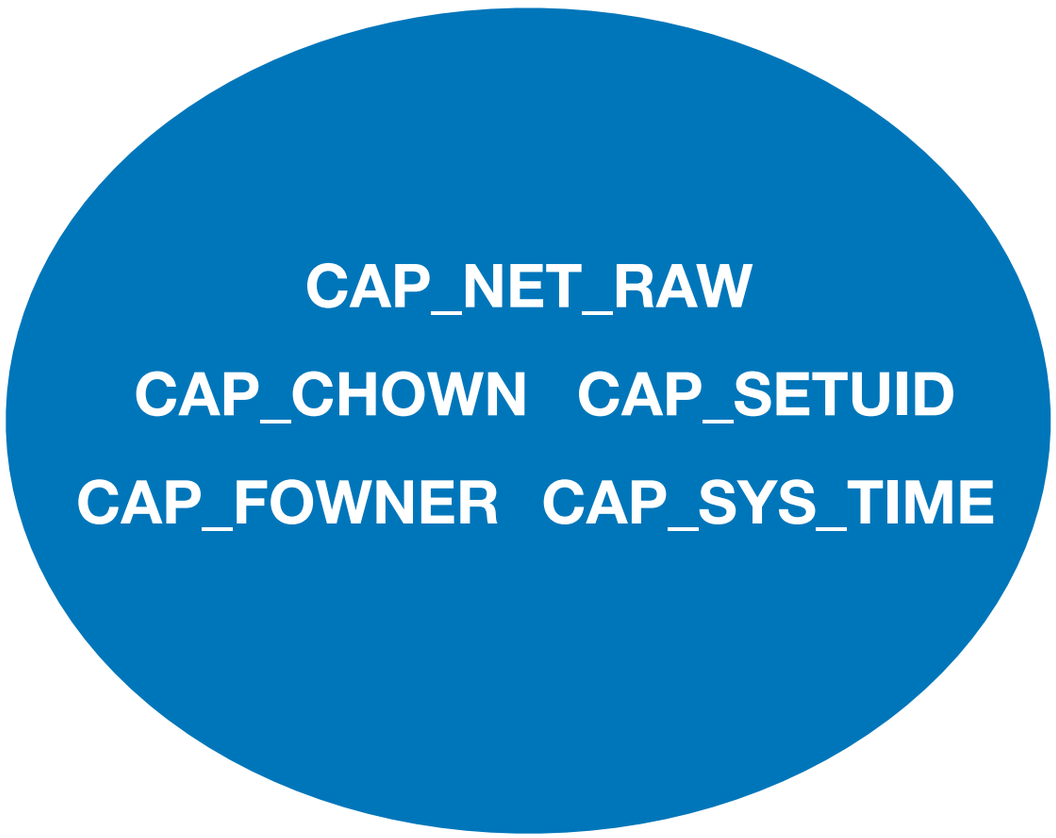


Linux Capabilities

effective set



permitted set



priv_raise: copy a privilege from permitted set to effective set
priv_lower: delete a privilege from effective set *temporarily*
priv_remove: remove privilege from permitted set *permanently*

Linux Capabilities

`ping`: send ICMP packets to a network host

privileges needed: open raw socket

```
socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

CAP_NET_RAW

some other network related privileges in

```
setsockopt(icmp_sock, SOL_SOCKET, SO_DEBUG, (char *)&hold, sizeof(hold));
setsockopt(icmp_sock, SOL_SOCKET, SO_MARK, &mark, sizeof(mark));
```

CAP_NET_ADMIN

Problem Solved!?

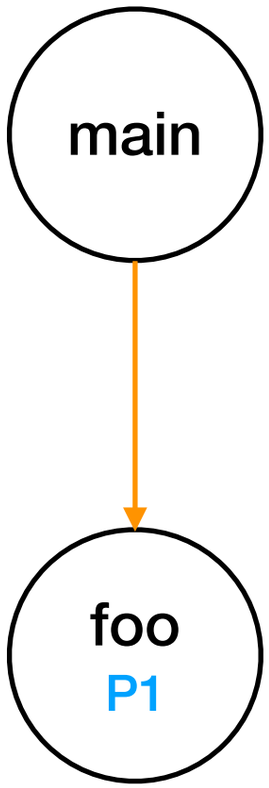
two steps

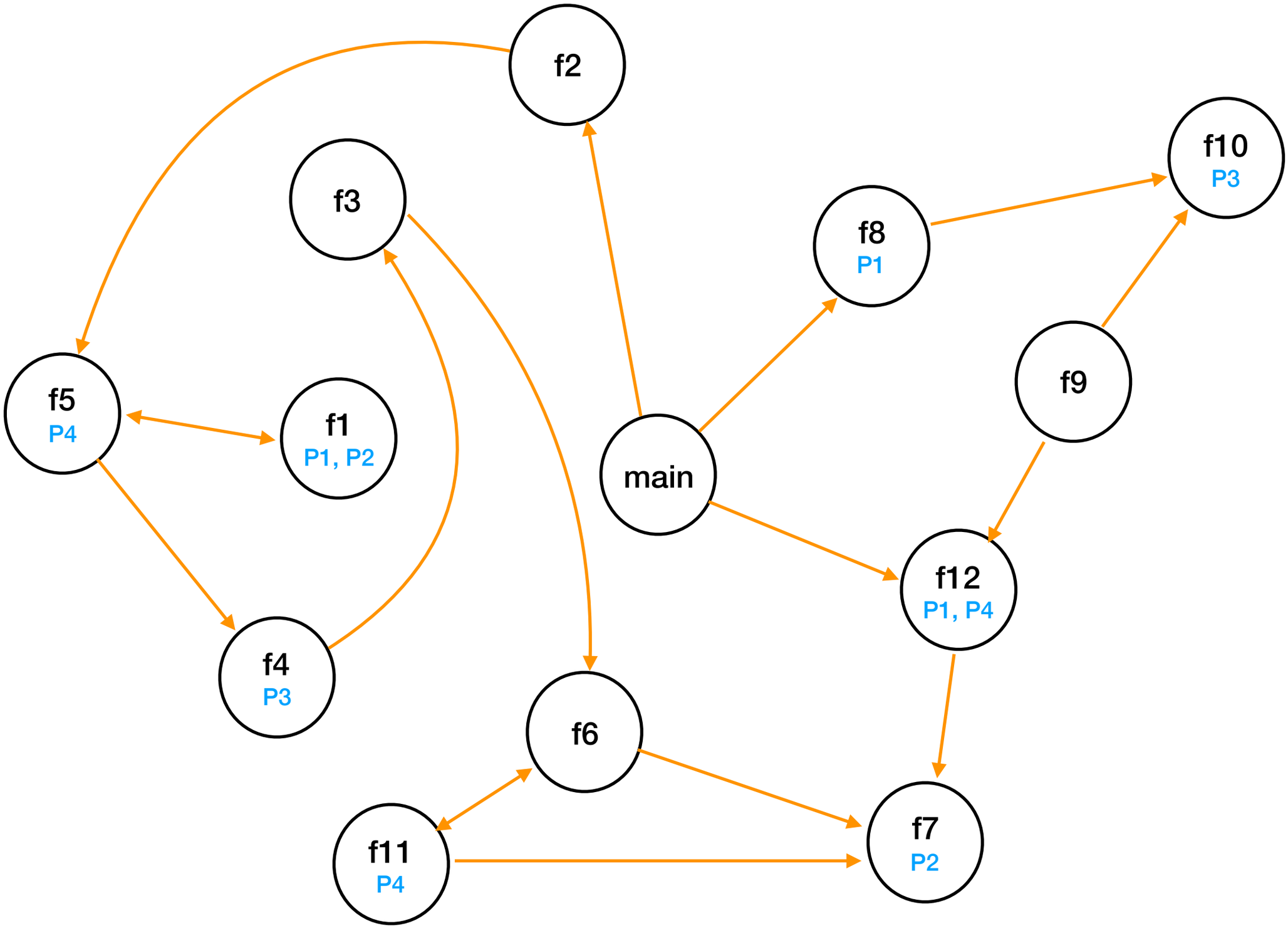
1. figure out which privileges a program requires
2. do priv_remove on a privilege when it's no longer needed

A Simple Program

```
int main(int argc, char *argv[]) {
    ......
    ......
    foo();
    ......
}

void foo() {
    ...
    ... // use privilege P1
}   priv_remove(P1)
```





*It's difficult to **manually** figure out when we can remove which capabilities permanently.*

We need a tool!

AutoPriv

- LLVM-based compiler
- uses *data-flow analysis* techniques to analyze and transform programs
- drops privileges when they are no longer needed

Outline

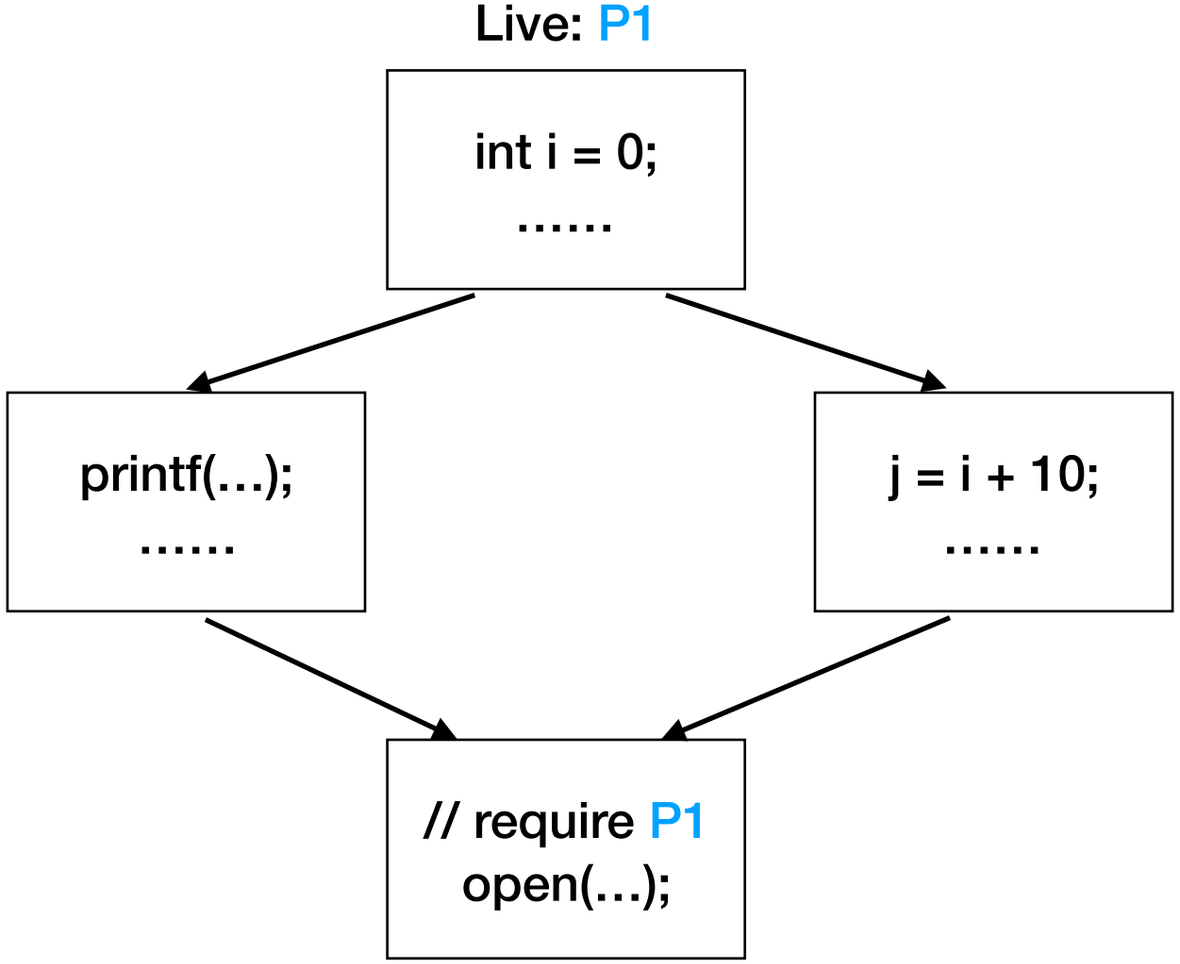
- Design
- Implementation
- Performance Experiments
- Conclusion

Outline

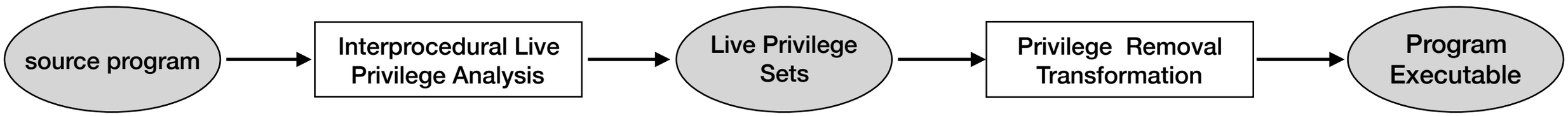
- Design
- Implementation
- Performance Experiments
- Conclusion

Design

Live Privilege: privilege that may be used along some path in the future



Design



AutoPriv Architecture

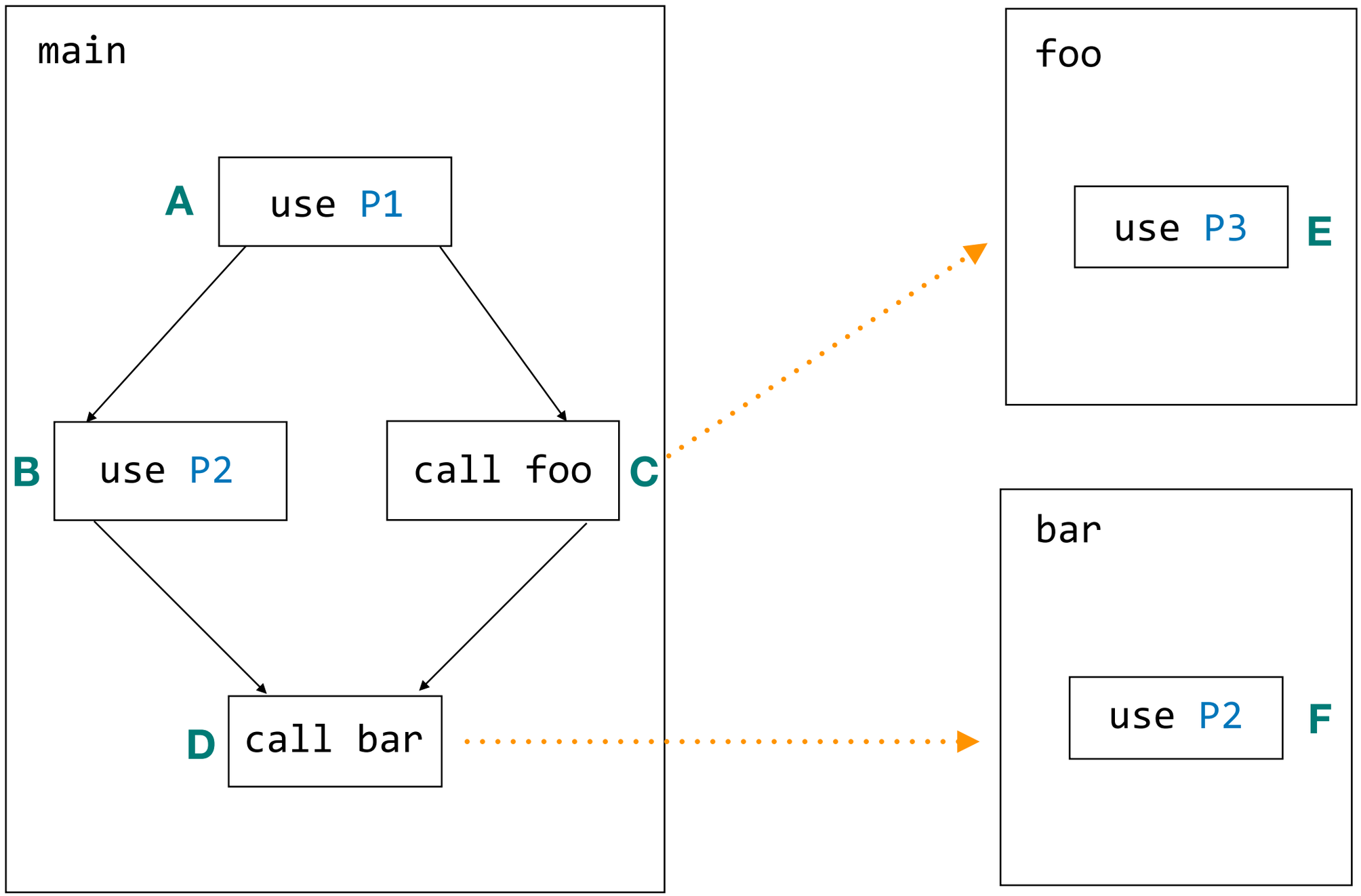
Design

Live Privilege Analysis

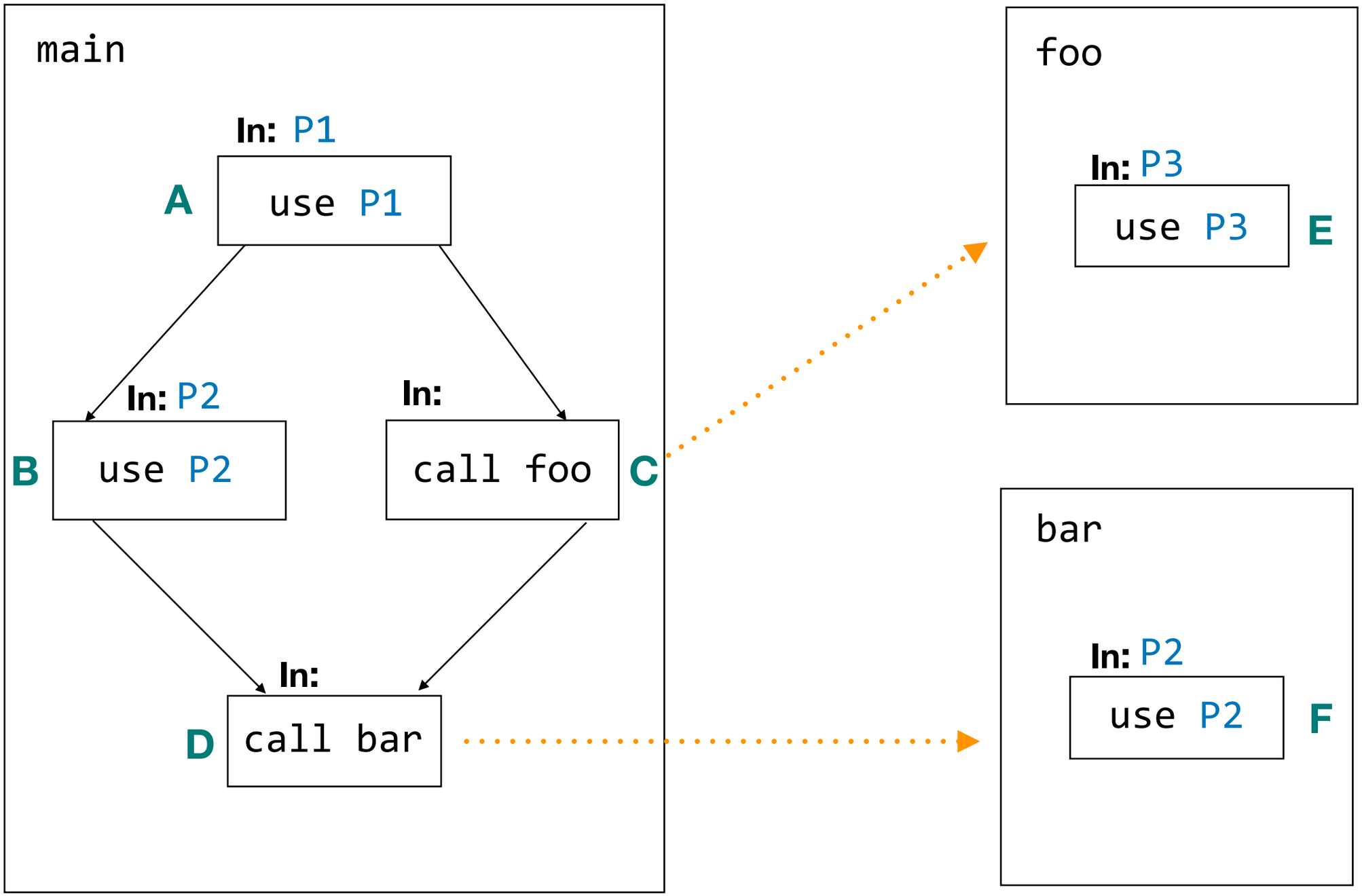
- iterative
- inter-procedural
- context-insensitive

- Propagate Privileges
    - within basic blocks (BB)
    - between successors BB to predecessors BB
    - callees to callers
- Remove Privileges

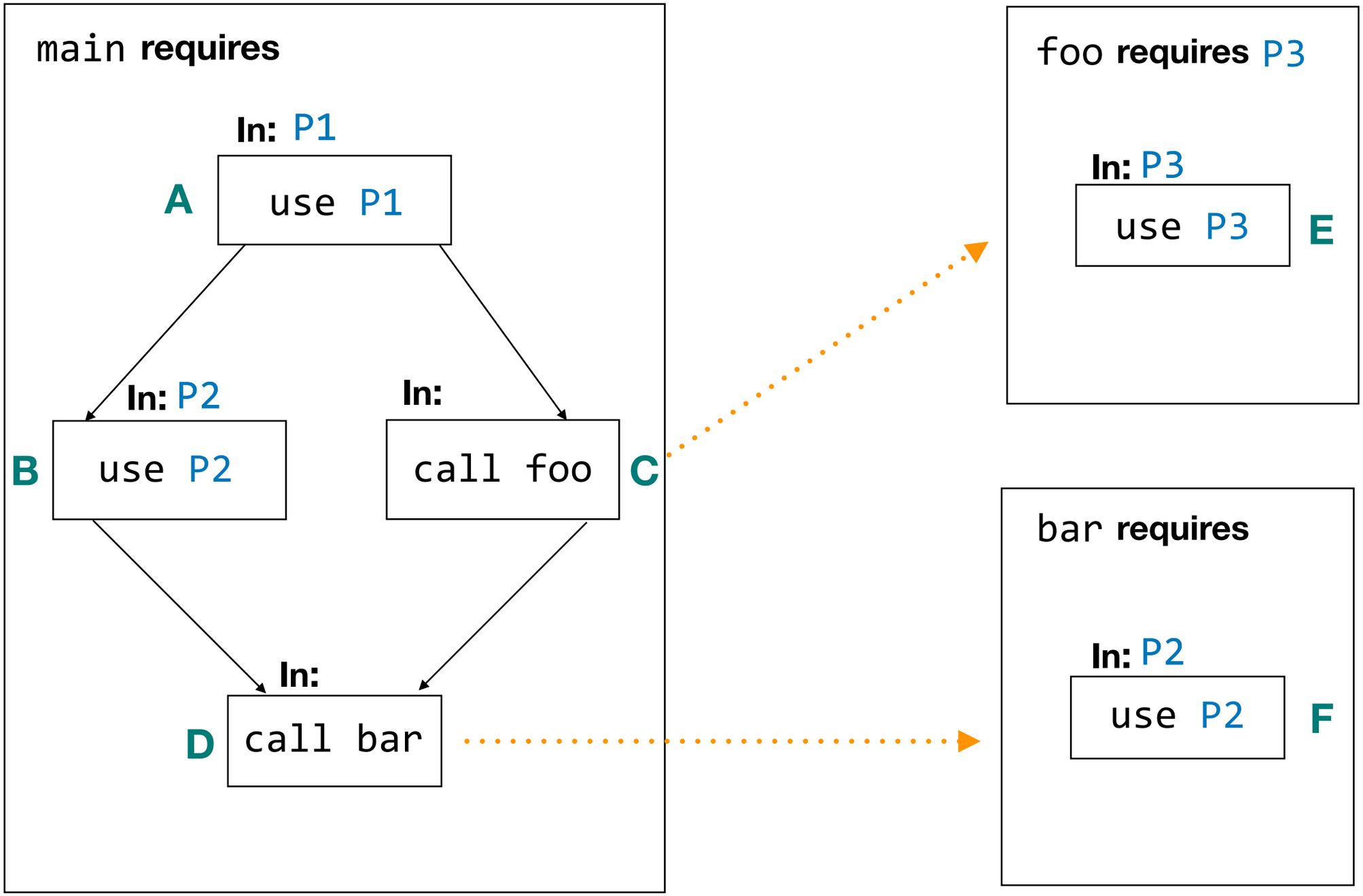
An Example Program



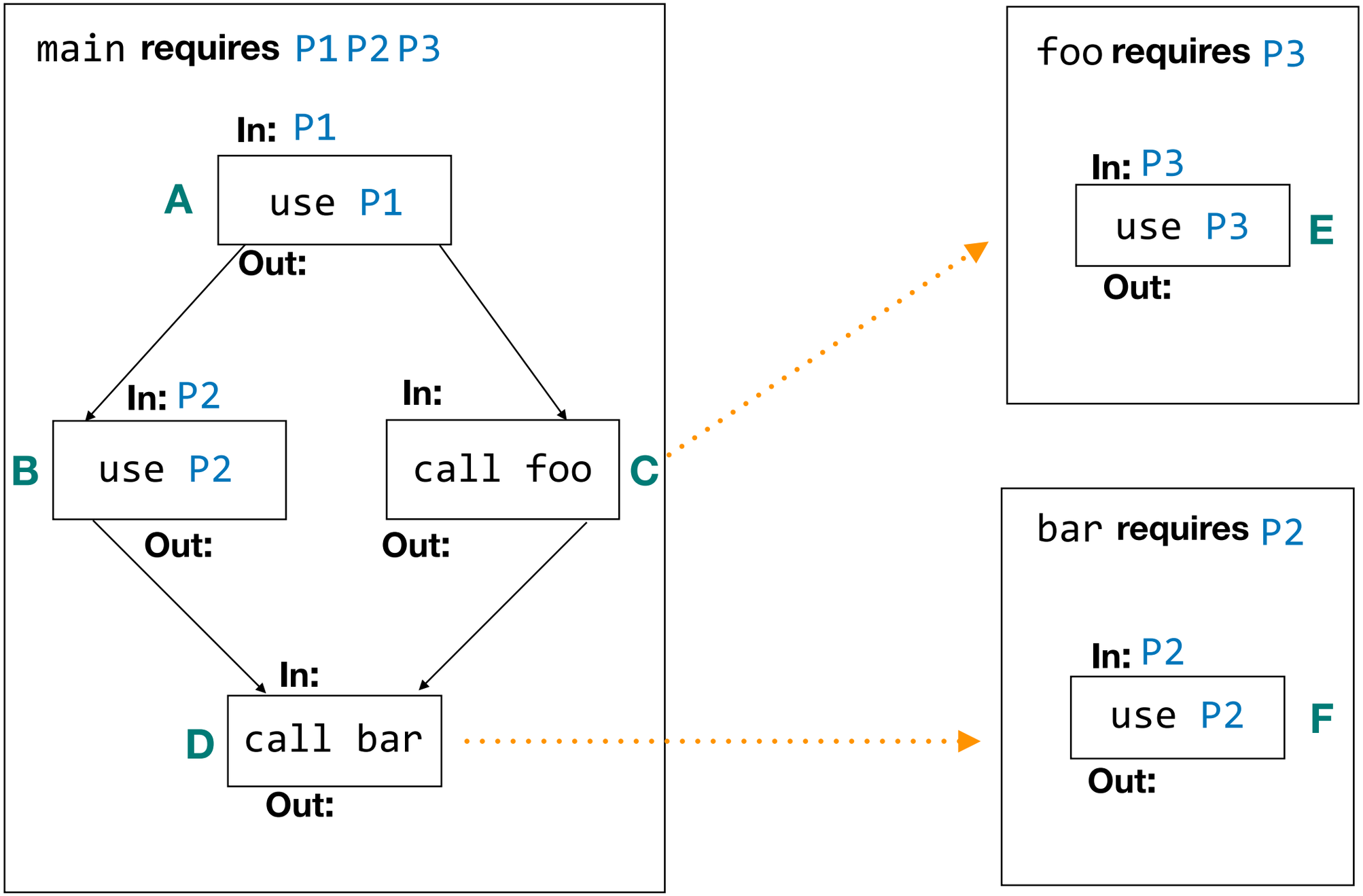
Local Privilege Analysis



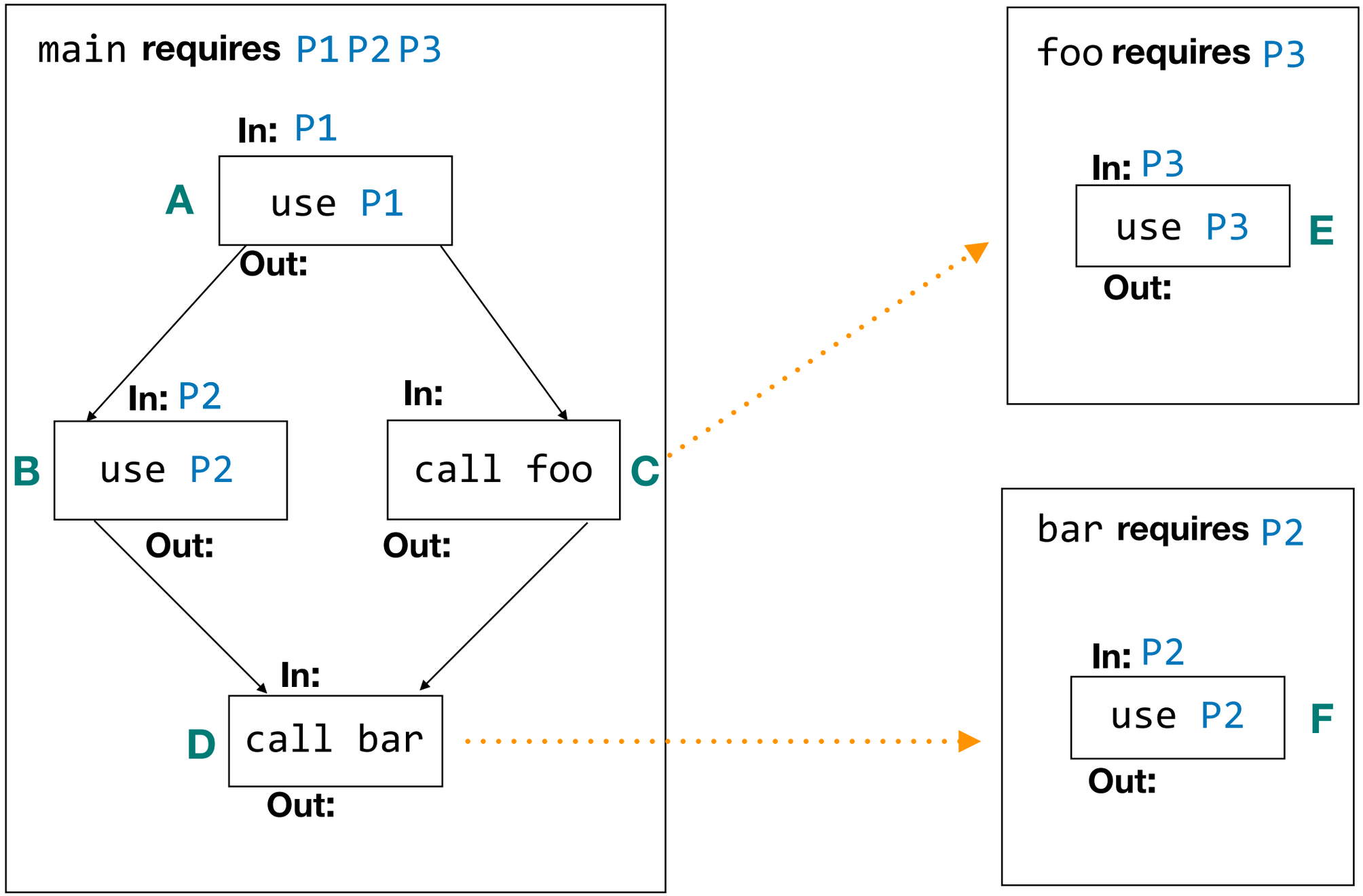
Interprocedural Privilege Analysis



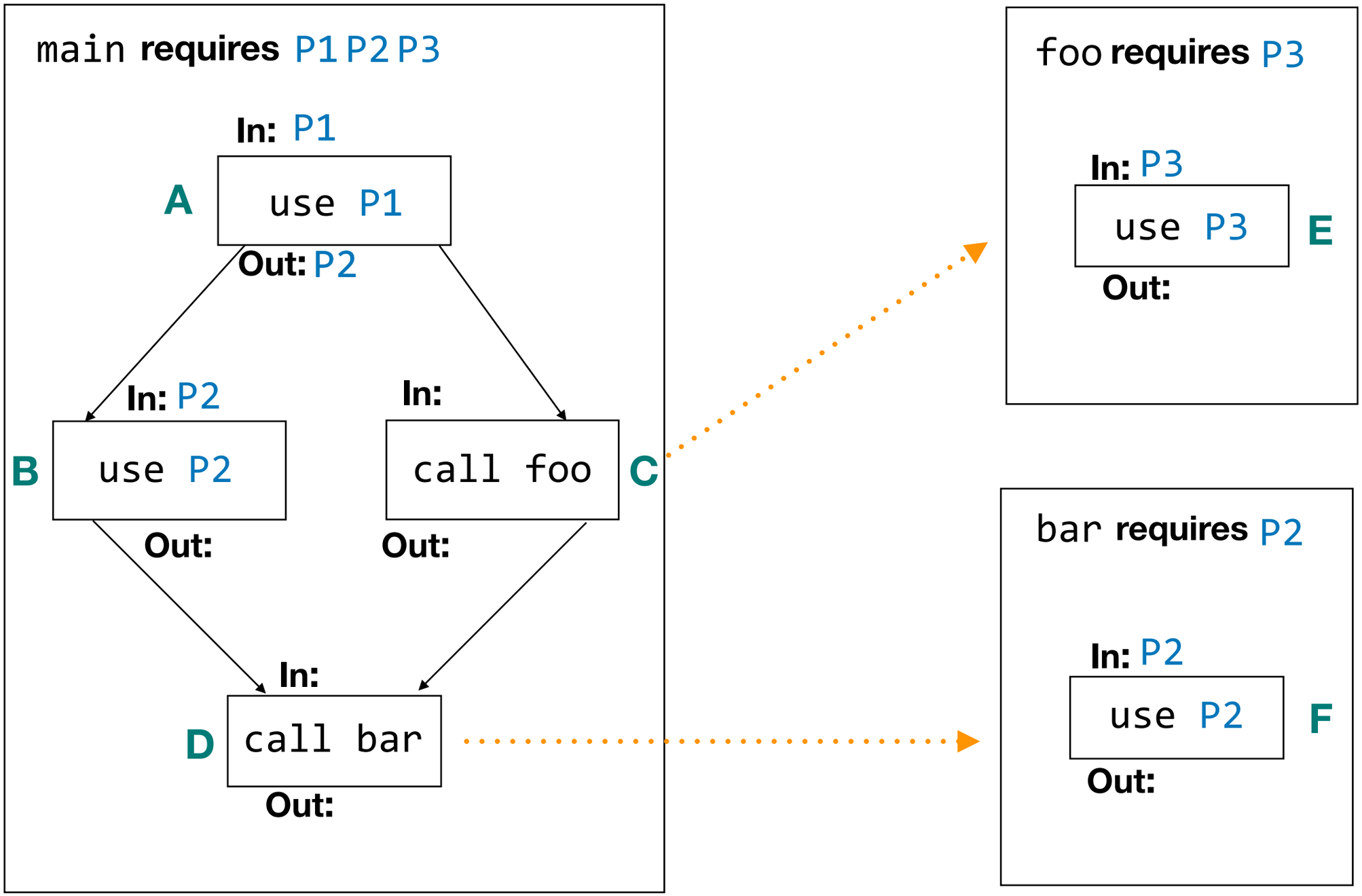
Interprocedural Live Privilege Analysis



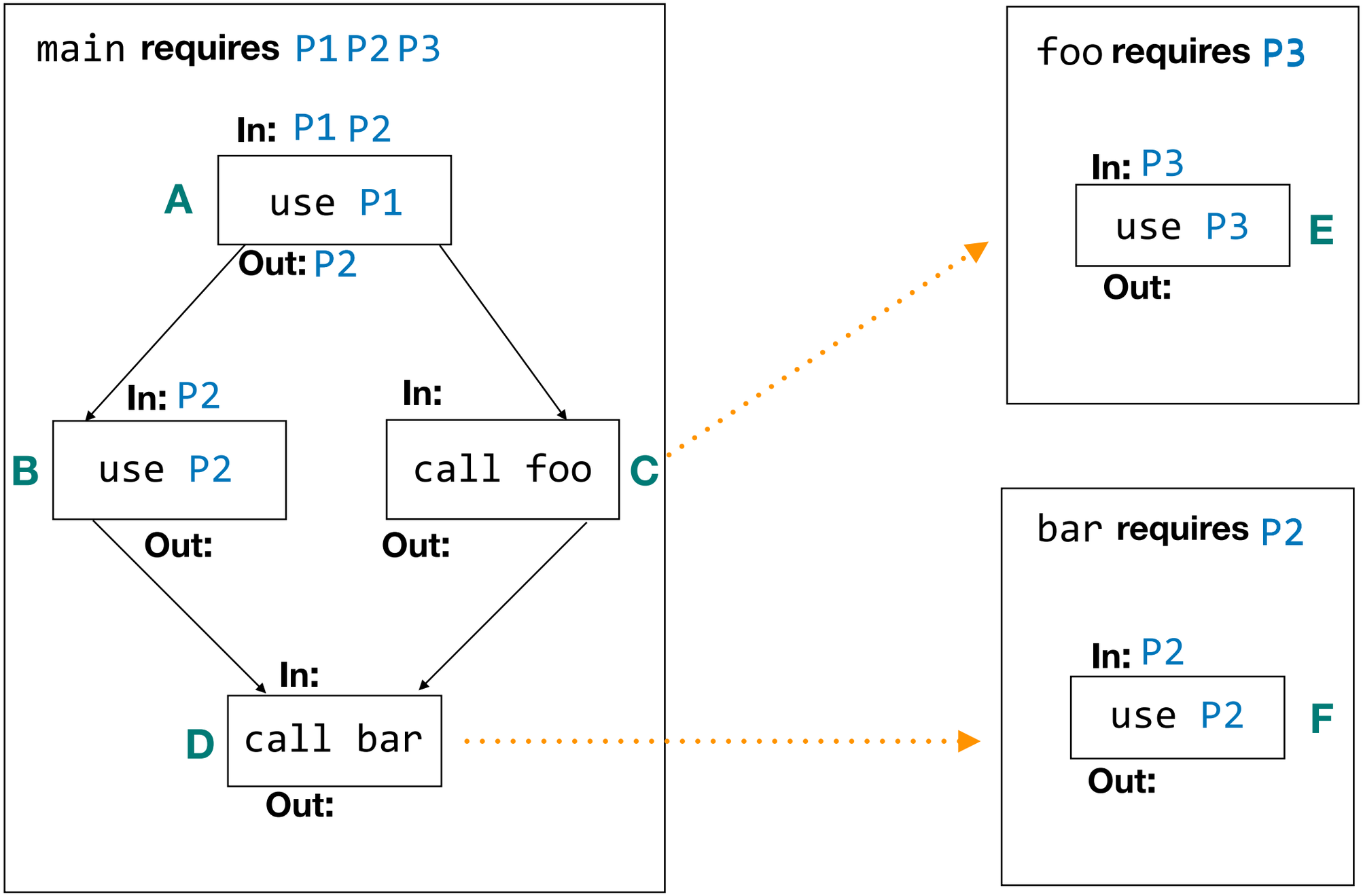
Propagate: Successor to Predecessors



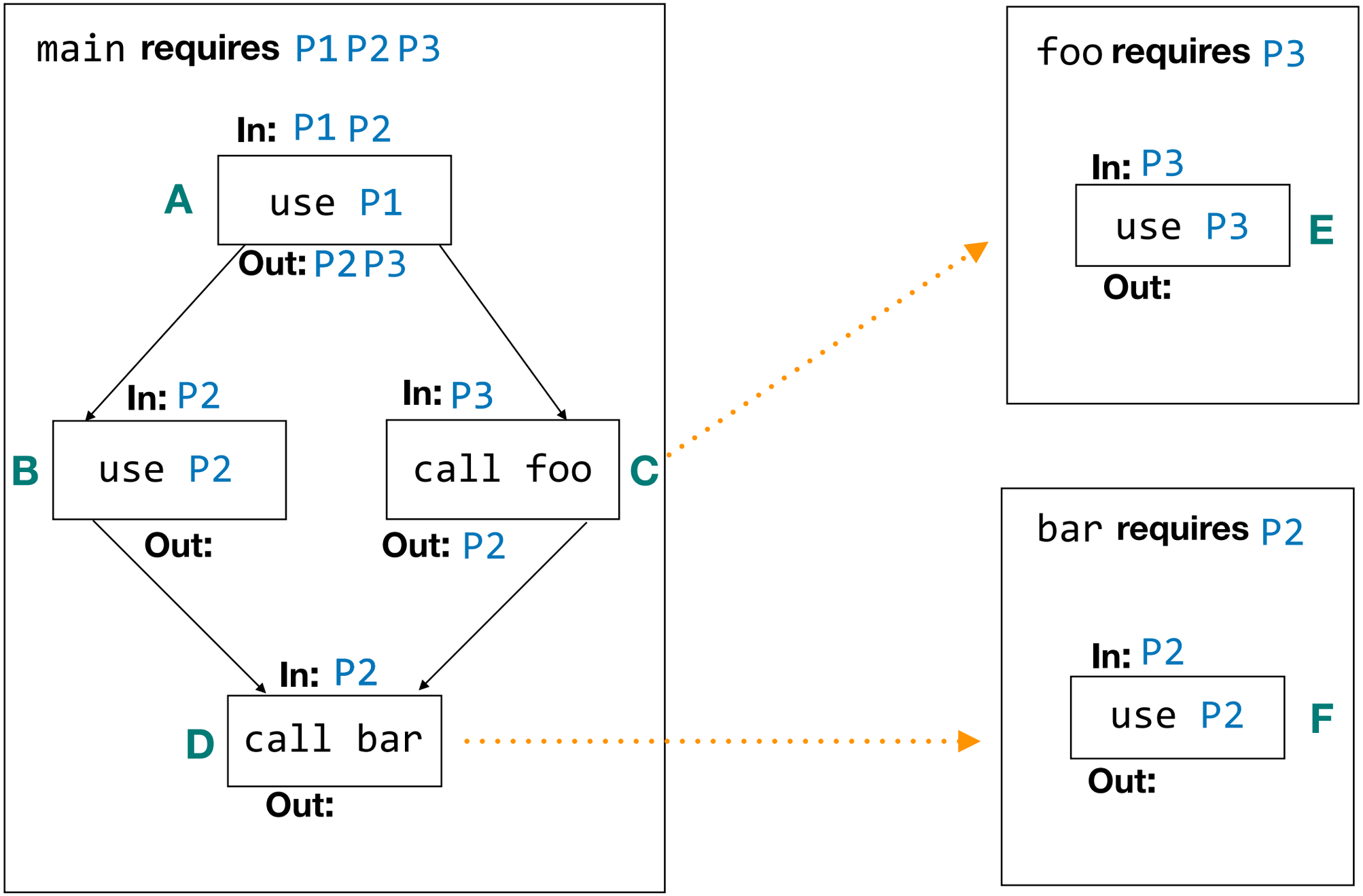
Propagate: Out to In



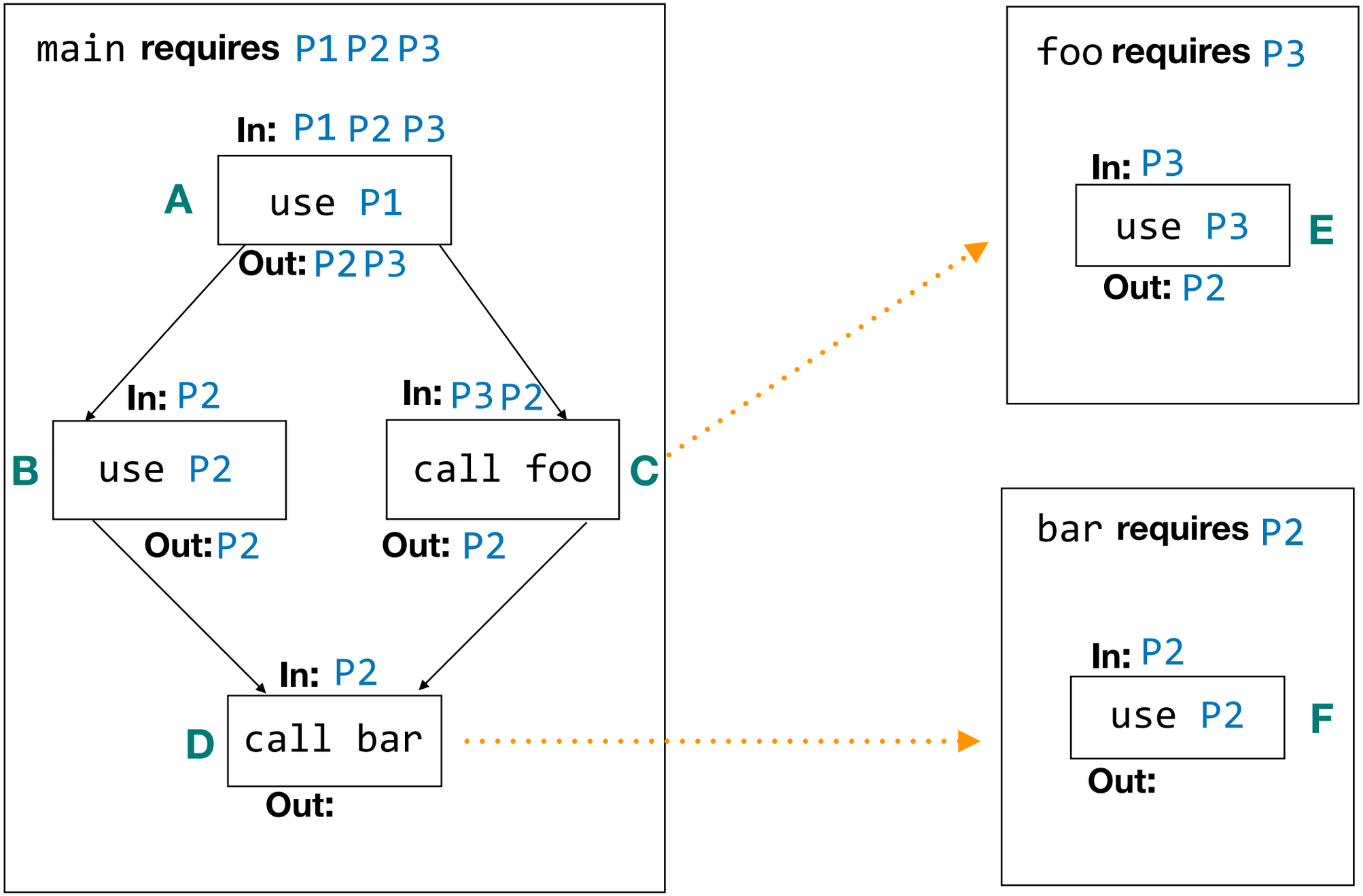
Propagate: Callee to Caller



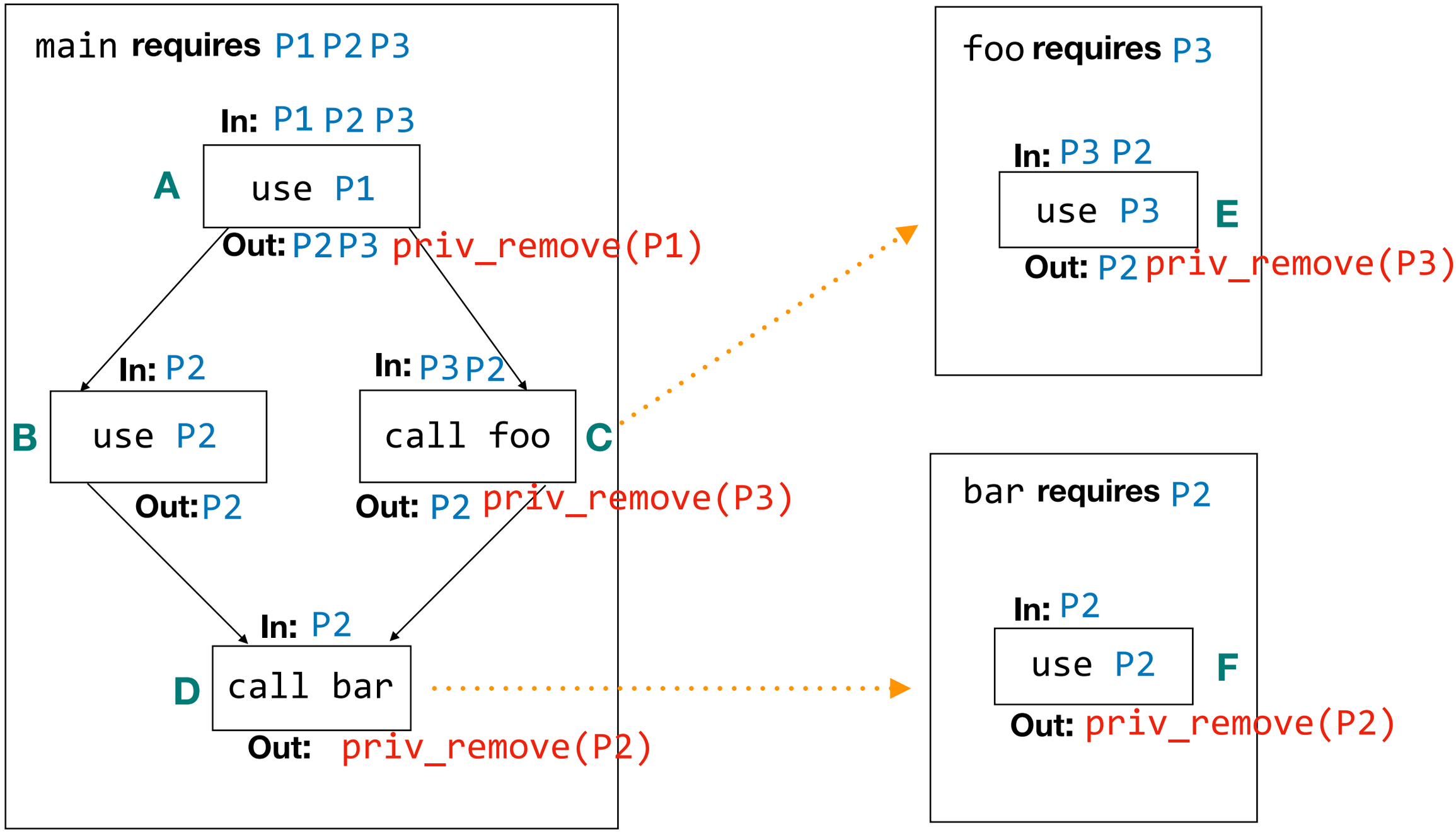
Propagate Iteratively



Propagate: Caller to Callee's Exit



Remove Dead Privileges (In - Out)



Outline

- Design
- Implementation
- Performance Experiments
- Conclusion

Implementation

Privilege Primitives

`priv_raise(int cap_num, int capability, …)`: copy a set of privileges from the permitted set to effective set. The permitted set remains unchanged.

`priv_lower(int cap_num, int capability, …)`: delete a set of privileges from the effective set *temporarily*.

`priv_remove(int cap_num, int capability …)`: remove a set of privileges from both the effective set and the permitted set *permanently*.

Implementation

Manually put a pair of `priv_raise()` and `priv_lower()` around a system call or library function call that uses some privilege(s), e.g.,

```
priv_raise(1, CAP_NET_RAW);
icmp_sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
priv_lower(1, CAP_NET_RAW);
```

Program	Version	SLOC	Number of Privilege-Bracketing Function Calls
passwd	4.1.5.1	51,371	29
su	4.1.5.1	51,371	34
ping	s20121221	12,001	6
sshd	6.61p	82,376	59
thttpd	2.26	8,367	8

Number of Required Privilege-Bracketing

Implementation

5 `LLVM IR` passes

1. Split basic blocks
2. Local Privilege Analysis
3. Interprocedural Privilege Analysis
4. Interprocedural Live Privilege Analysis
5. Privilege Removal Instrumentation

Outline

- Design
- Implementation
- Performance Experiments
- Conclusion

Performance Experiments

- *Compiler overhead* induced by program analysis and instrumentation
- *Program overhead* induced by raising, lowering, and removing privileges

Experiment Setup

- LLVM: 3.7.0
- OS: 64-bit Ubuntu 16.04
- CPU: Intel Core i5-6600 3.30GHZ
- RAM: 8GB
- Disk: 256GB SSD

Test Programs

Program	Version	SLOC	Description
passwd	4.1.5.1	51,371	Password change utility
su	4.1.5.1	51,371	Run programs as another user
ping	s20121221	12,001	Send ICMP packets to a remote host
sshd	6.61p	82,376	Remote login server with encrypted connection
thttpd	2.26	8,367	a lightweight HTTP server

Test Programs

Compiler Overhead

1. Use `Clang` to compile programs with `-O2` to `LLVM bitcode`
2. Run LLVM's `opt` with and without our passes
3. Run `opt`'s `-time-passes` to measure the user and system execution time

Ran experiments 20 times for each program

Program	Version	Average	Standard Deviation	Overhead
passwd	Original AutoPriv	136.17 ms 173.19 ms	0.43 ms 0.57 ms	27.18%
su	Original AutoPriv	205.96 ms 237.16 ms	1.00 ms 1.09 ms	15.15%
ping	Original AutoPriv	159.02 ms 171.54 ms	0.46 ms 0.40 ms	7.87%
sshd	Original AutoPriv	4,090.87 ms 5,203.24 ms	21.16 ms 29.56 ms	27.19%
thttpd	Original AutoPriv	317.95 ms 372.38 ms	0.77 ms 0.66 ms	17.12%

Compiler Overhead

Application Overhead

Program	Configuration	Repetition
passwd	change current user's password	200
su	ran `echo` as another user	200
ping	`ping` -c 10 localhost	50
sshd	ran `scp` to fetch files from 16 KB to 16 MB	500
thttpd	`ab` -c 32 -n 10000	60

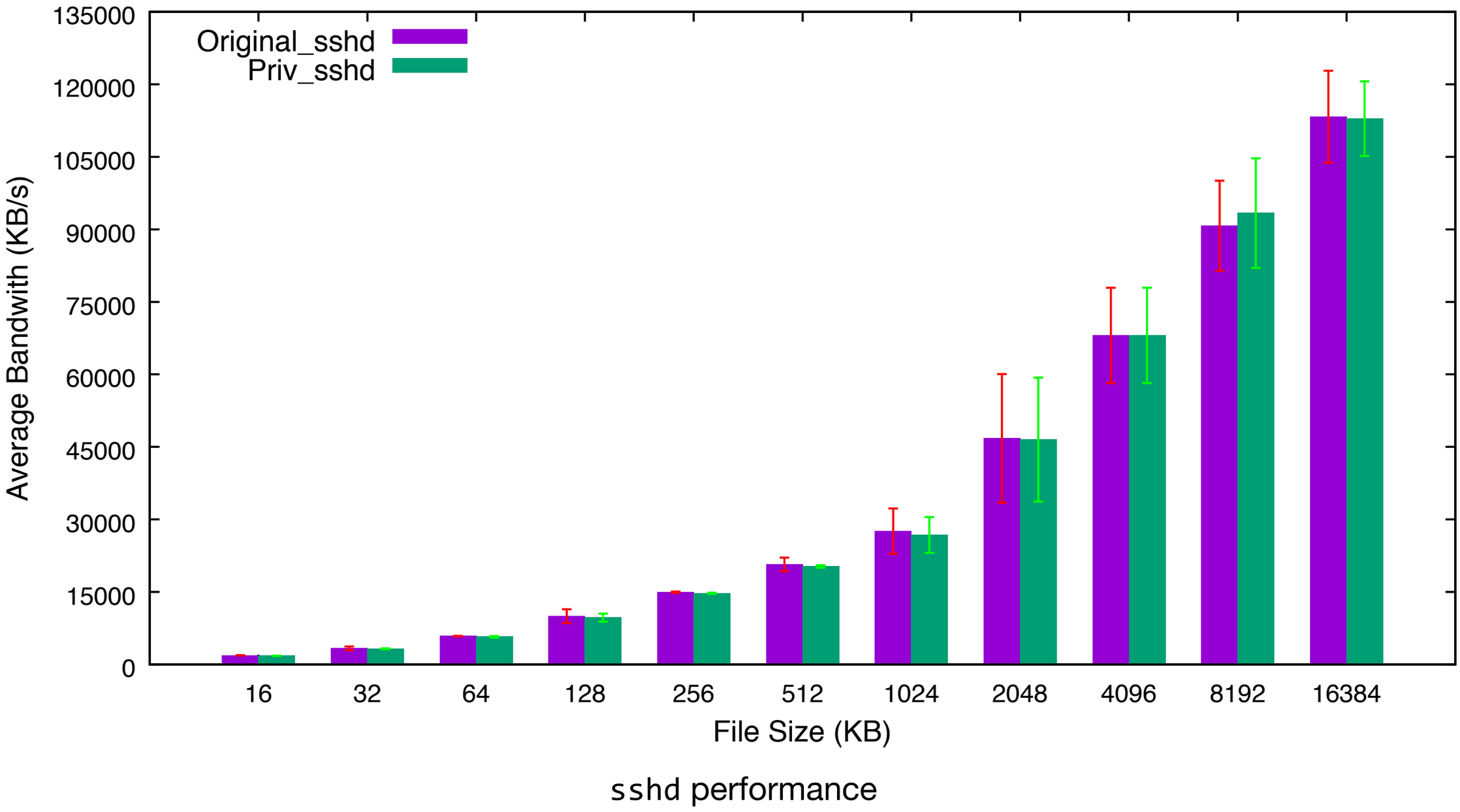
How We Ran Each Test Program

Application Overhead

Program	Version	Average	Standard Deviation	Overhead
passwd	Original AutoPriv	36.29 ms 36.66 ms	3.75 ms 2.94 ms	0.01%
su	Original AutoPriv	7.74 ms 7.78 ms	0.05 ms 0.04 ms	0.01%
ping	Original AutoPriv	9,211.49 ms 9,211.42 ms	0.82 ms 0.79 ms	0%

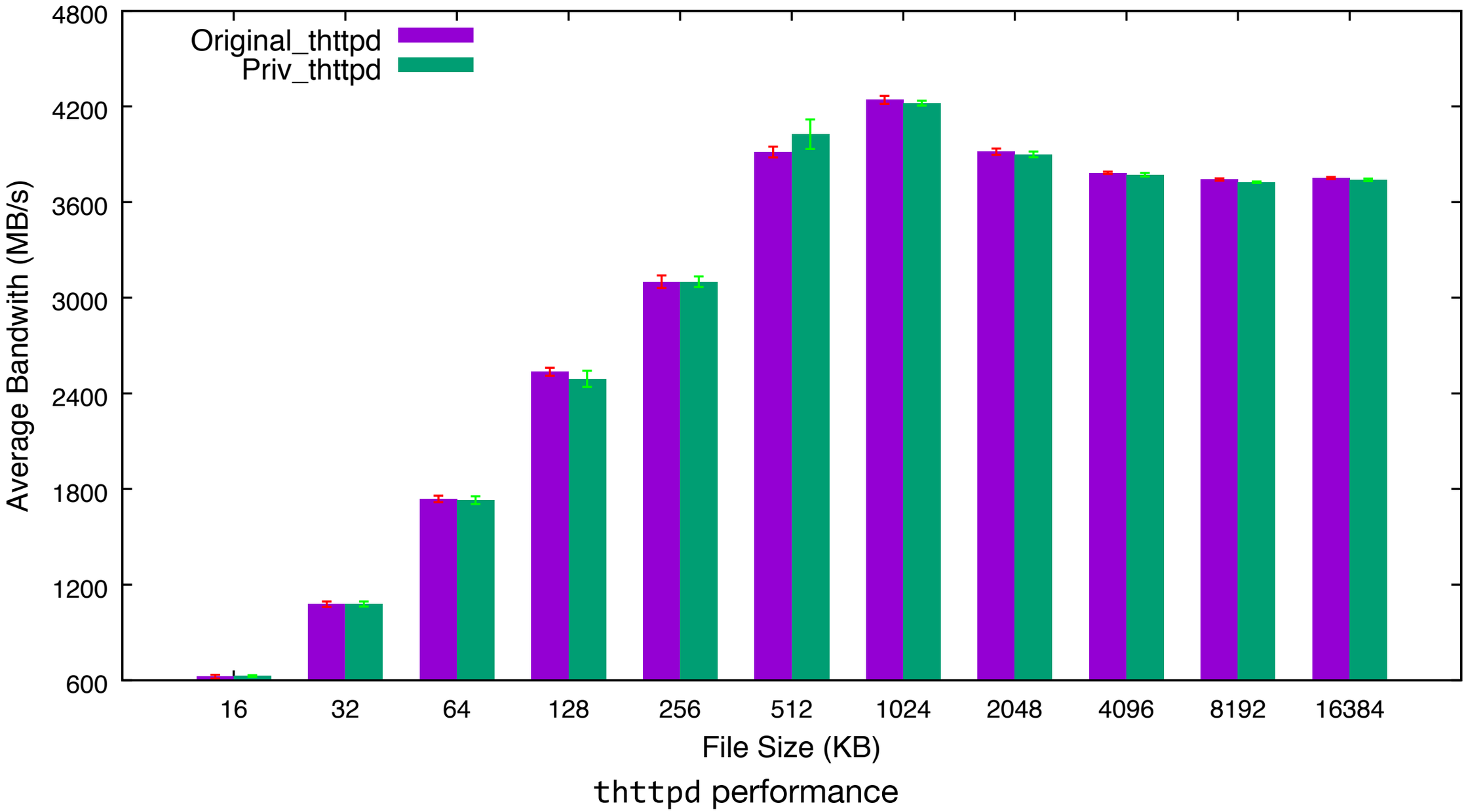
Performance of `passed`, `su`, and `ping`

Application Overhead



sshd performance

Application Overhead



Outline

- Design
- Implementation
- Performance Experiments
- Conclusion

Conclusion

- **`AutoPriv`** - an LLVM-based compiler that transforms code to drop dead privileges
- on average 19% overhead during optimization
- no overhead in the programs that `AutoPriv` transforms

Open-source: https://github.com/jtcriswell/AutoPriv/tree/AutoPriv

How More Secure Is AutoPriv'ed Program?

We measured how many dynamic instructions are executed with each privilege available in the permitted set.

Program	CAP_CHOWN	CAP_SETUID	CAP_SYS_CHROOT	CAP_NET_BIND_SERVICE	CAP_SETGID
thttpd	323 (0.00%)	323(0.00%)	4,686,266 (9.82%)	4,686,627 (9.82%)	4,693,826 (9.84%)

Program	CAP_DAC_READ_SEARCH	CAP_SETUID	CAP_CHOWN	CAP_DAC_OVERRIDE	CAP_FOWNER
passwd	2,654(3.81%)	43,952(63.02)	43,952(63.02)	69,582(99.77)	69,582(99.77)

How Did You Handle Function Pointers?

Our call graph is conservative.

- LLVM's built-in call graph
- Data Structure Analysis (DSA)

I’m Still Concerned With Manual Bracketing …

It’s very difficult for a compiler to figure out which function call needs which privileges.

For instance,

open(...)