Metacompilation

Based on slides by:

Dawson Engler, Ken Ashcraft, Ben Chelf, Andy Chou,
Seth Hallem, Yichen Xie, Junfeng Yang

Stanford University

Original Slides at:
http://www.stanford.edu/~engler/paste02-talk.ppt

Context: finding bugs w/ static analysis

- Systems have many ad hoc correctness rules

  "sanitize user input before using it"; "check permissions before doing operation X"

  One error = compromised system

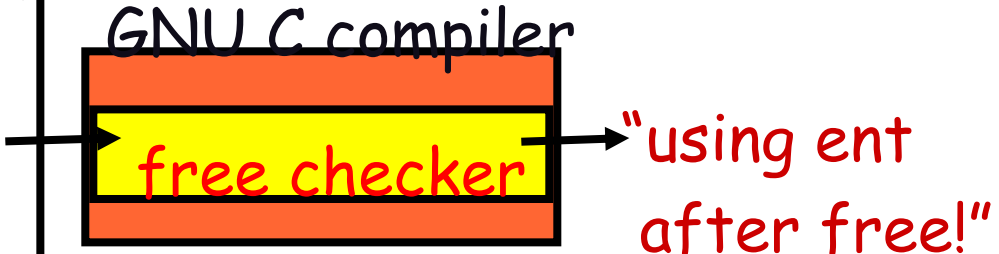
- If we know rules, can check with extended compiler

  Rules map to simple source constructs

  Use compiler extensions to express them

Linux fs/proc/ generic.c

```
ent->data = kmalloc(..)
if(!ent->data)
        free(ent);
        goto out;
...
out:    return ent;
```



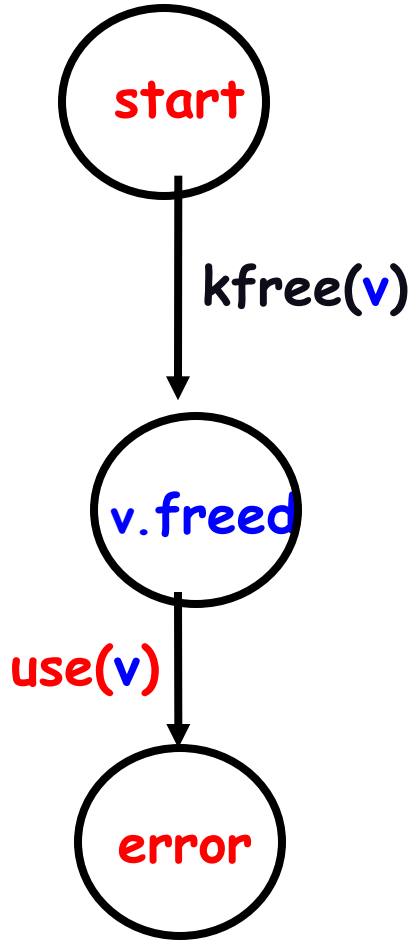
Nice: scales, precise, statically find 1000s of errors

A bit more detail

```
sm free_checker {
 state decl any_pointer v;
 decl any_pointer x;

 start: { kfree(v); } ==> v.freed
  ;
 v.freed:
   { v != x } || { v == x }
              ==> { /* do nothing */ }
 | { v } ==> { err("Use after free!"); }
   ;
}
```

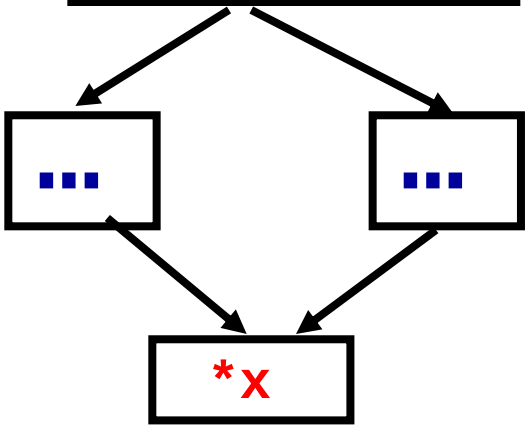
```
/* 2.4.1: fs/proc/generic.c */
ent->data = kmalloc(…)
if(!ent->data) {
     kfree(ent);
     goto out;
…
out: return ent;
```



A quick analysis example

```
freeit(int *z) {
  kfree(z);
}
```

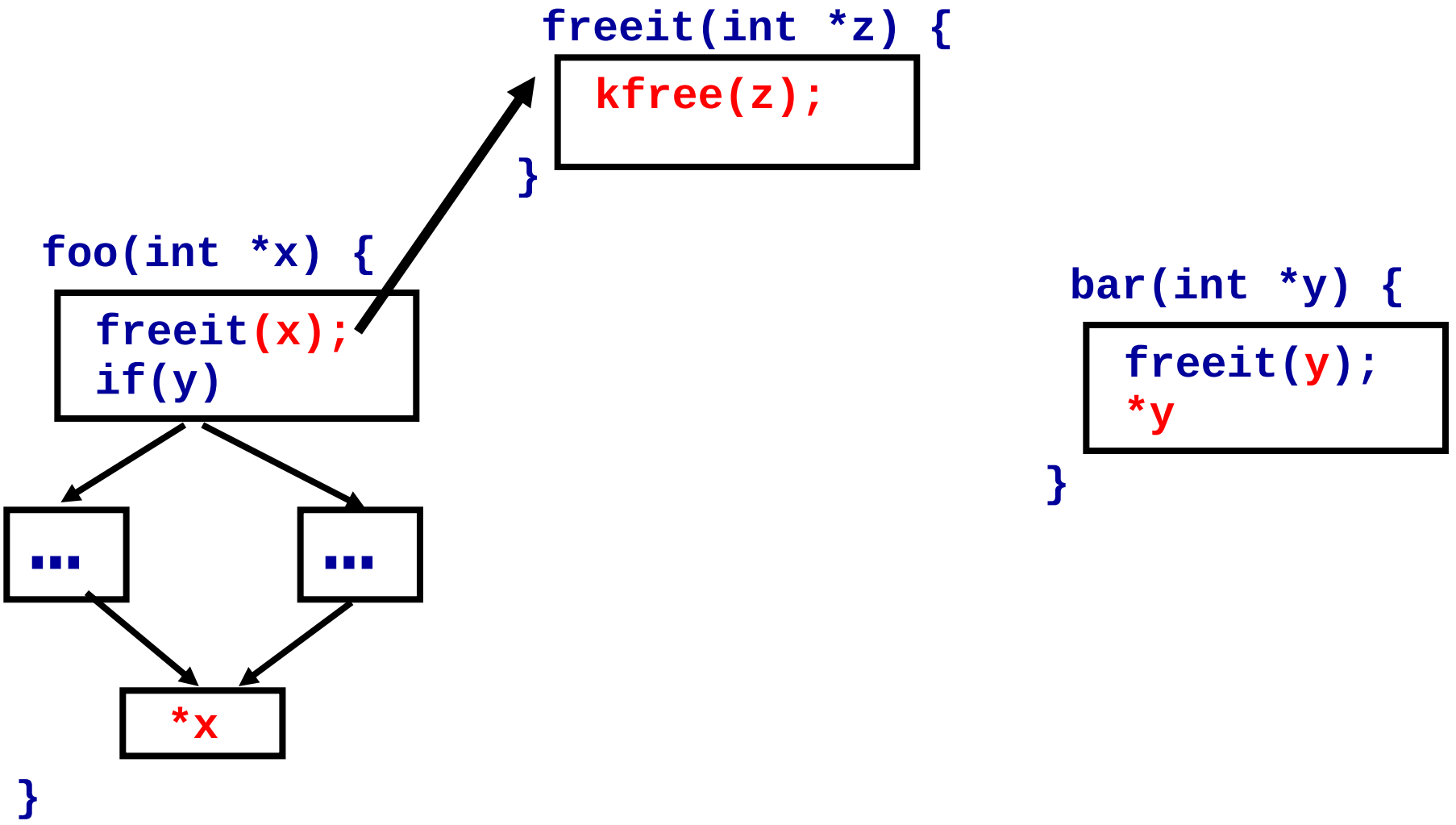
```
foo(int *x) {
  freeit(x);
  if(y)
```



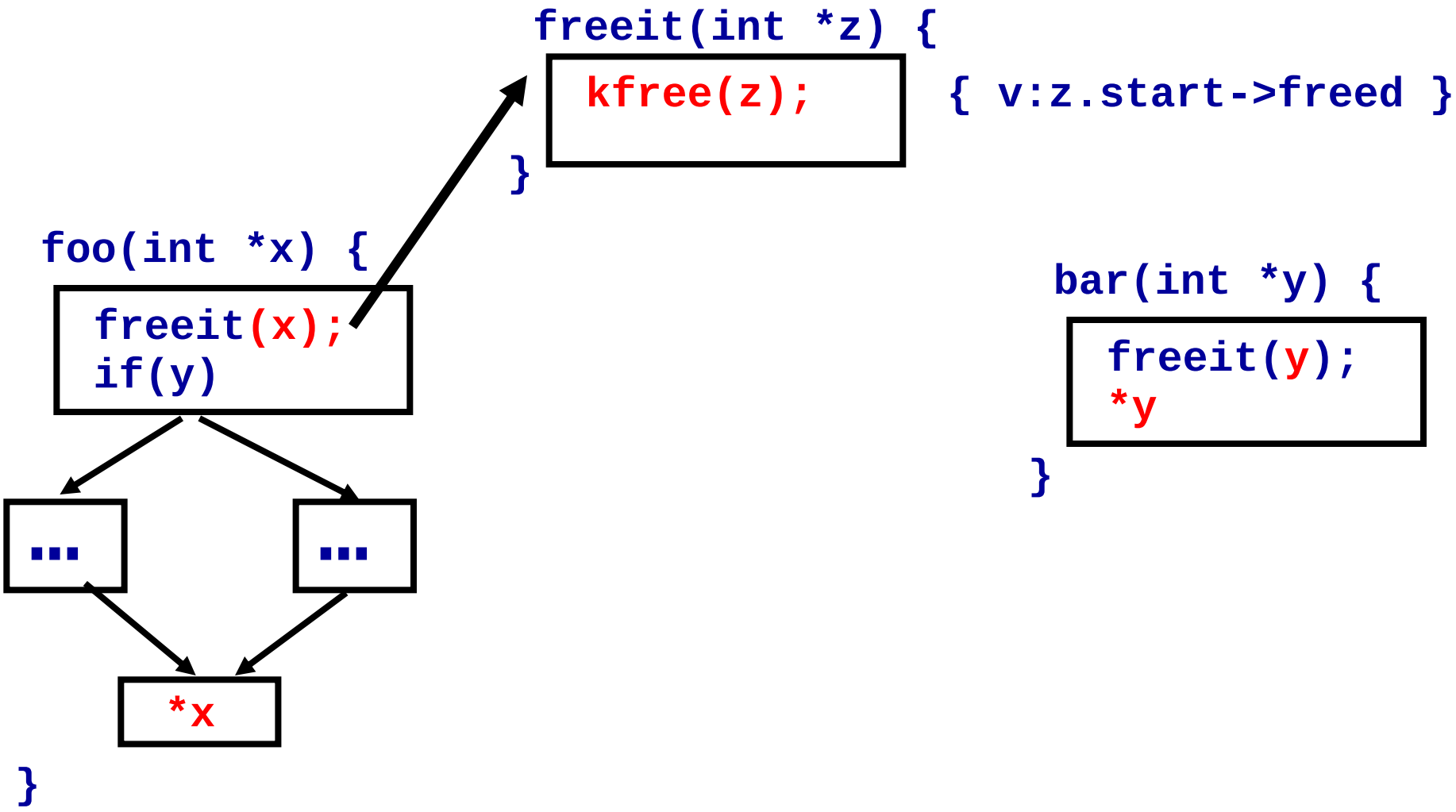

```
}
```

```
bar(int *y) {
  freeit(y);
  *y
}
```

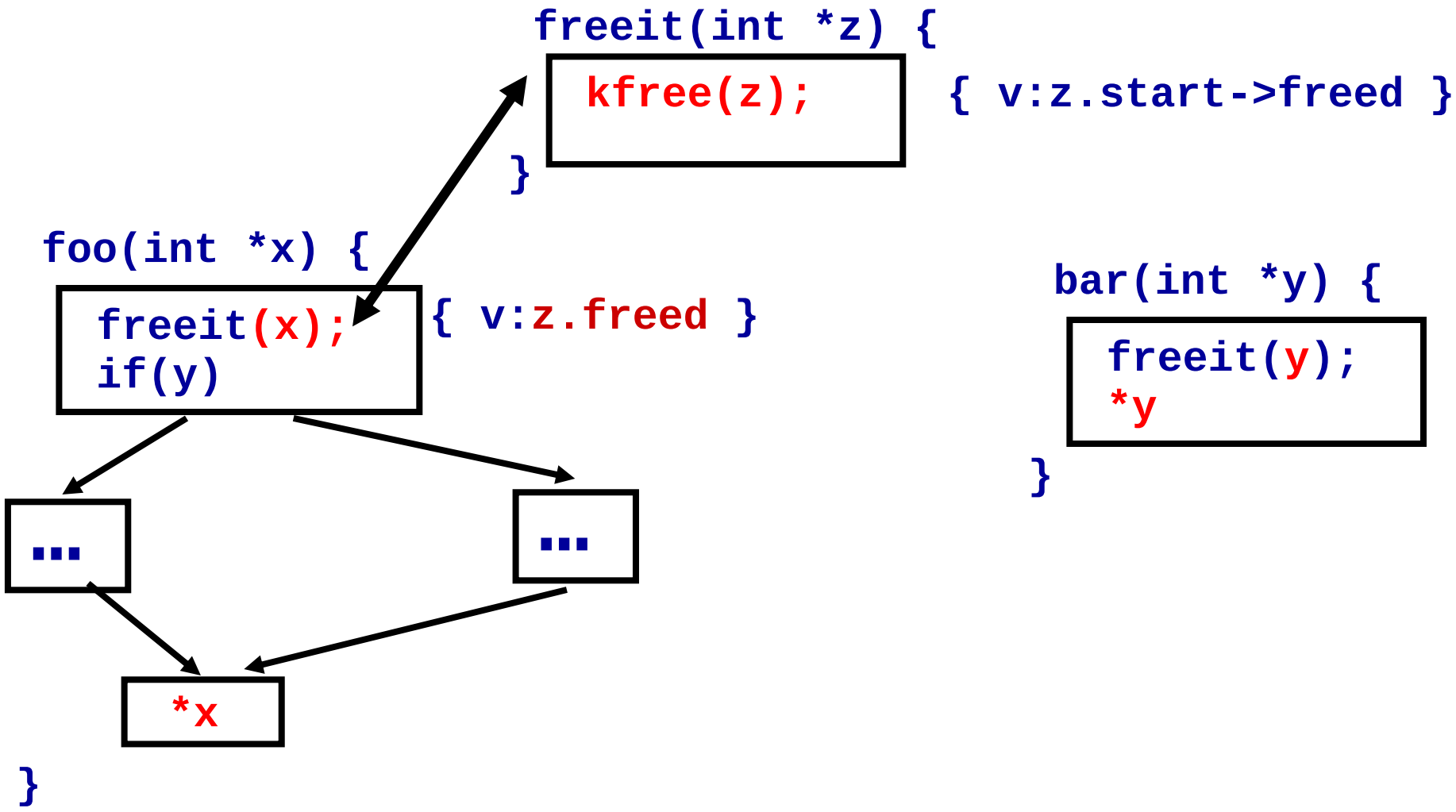
A quick analysis example



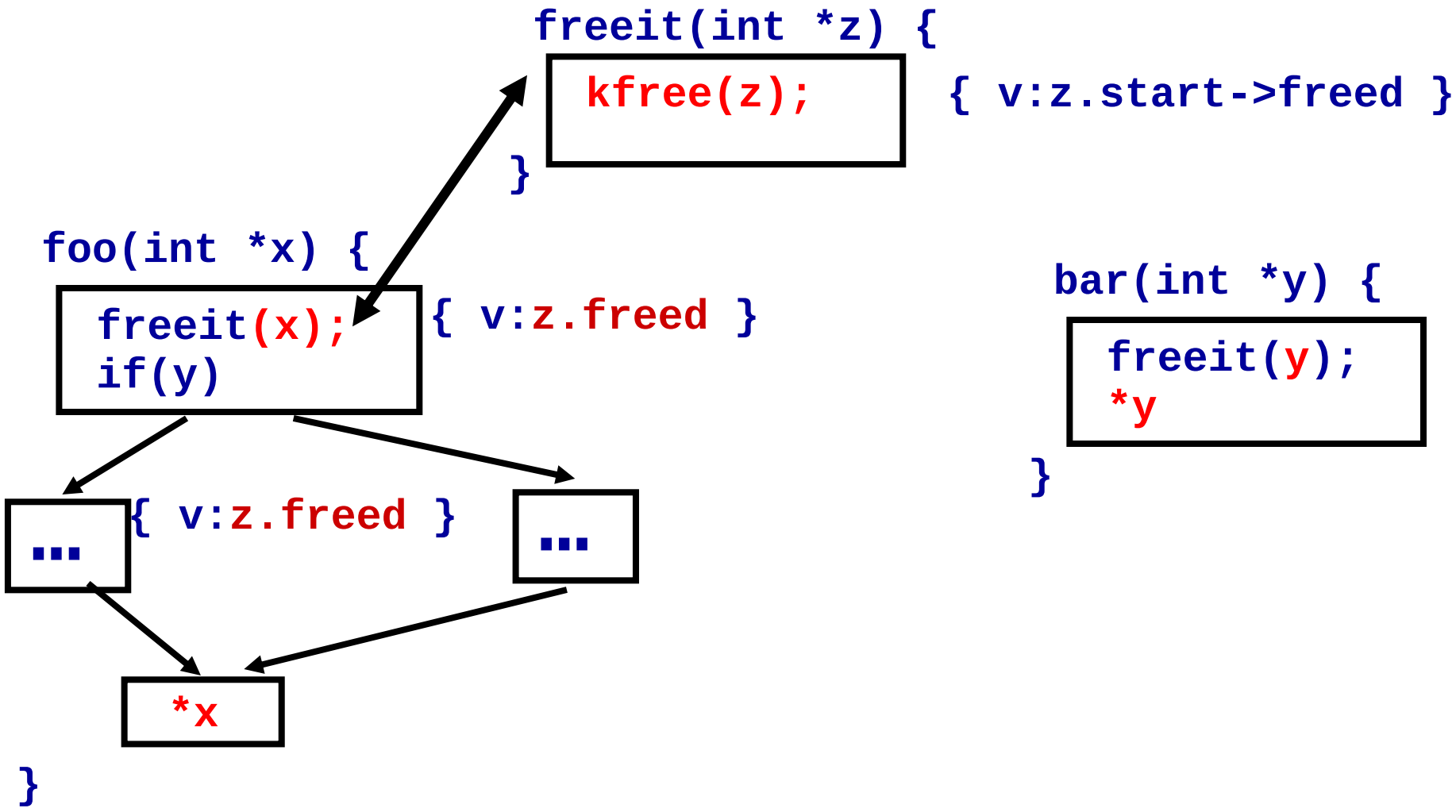
A quick analysis example



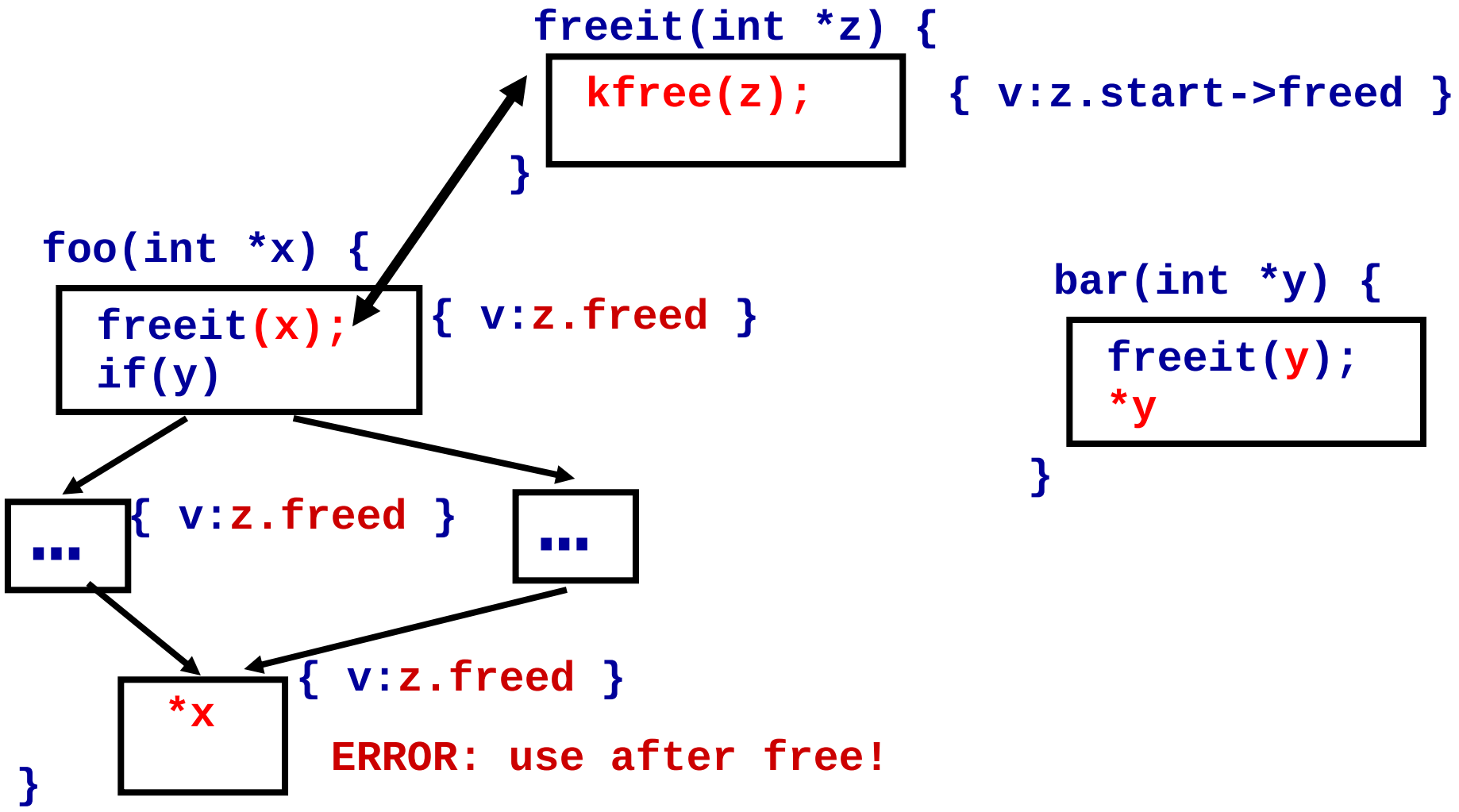
A quick analysis example



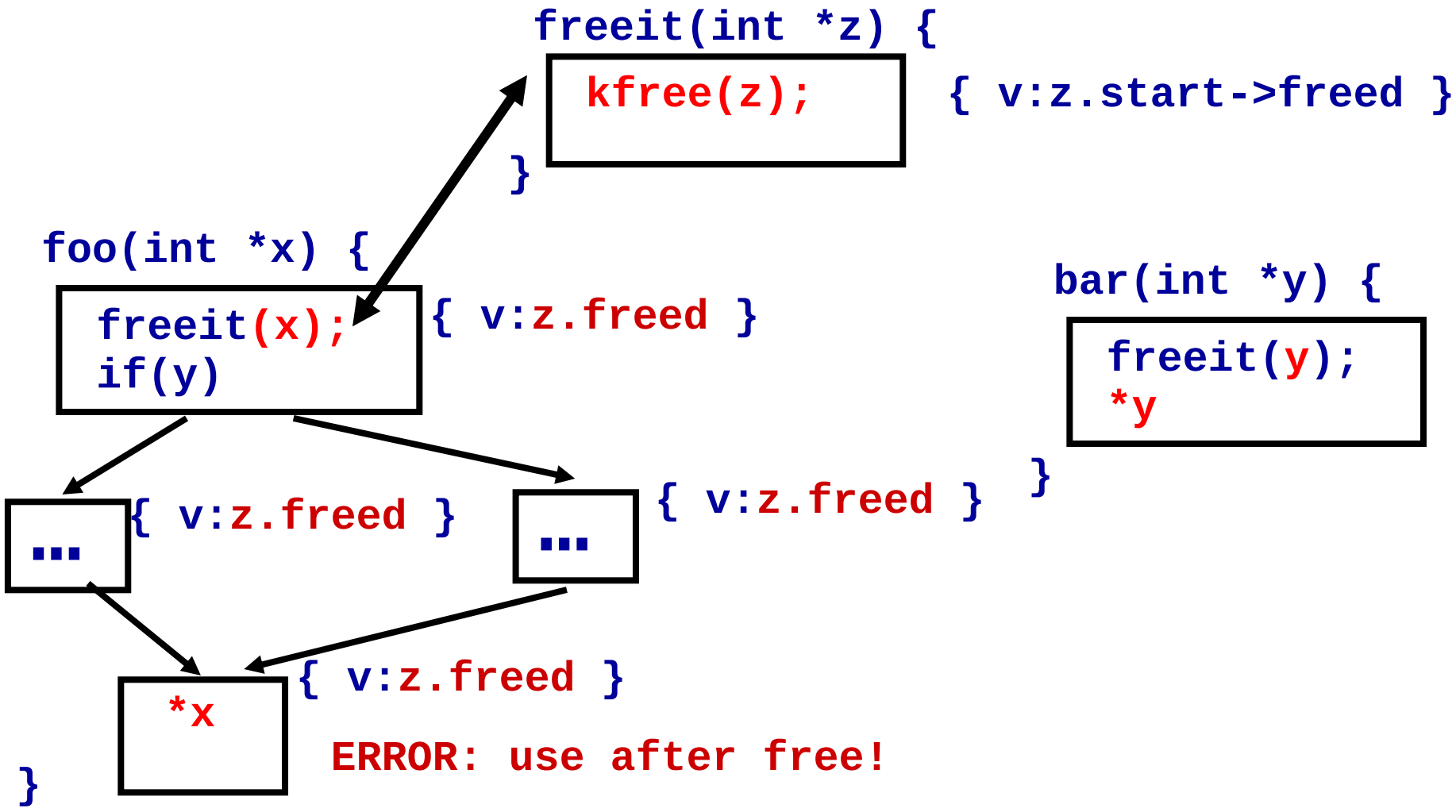
A quick analysis example



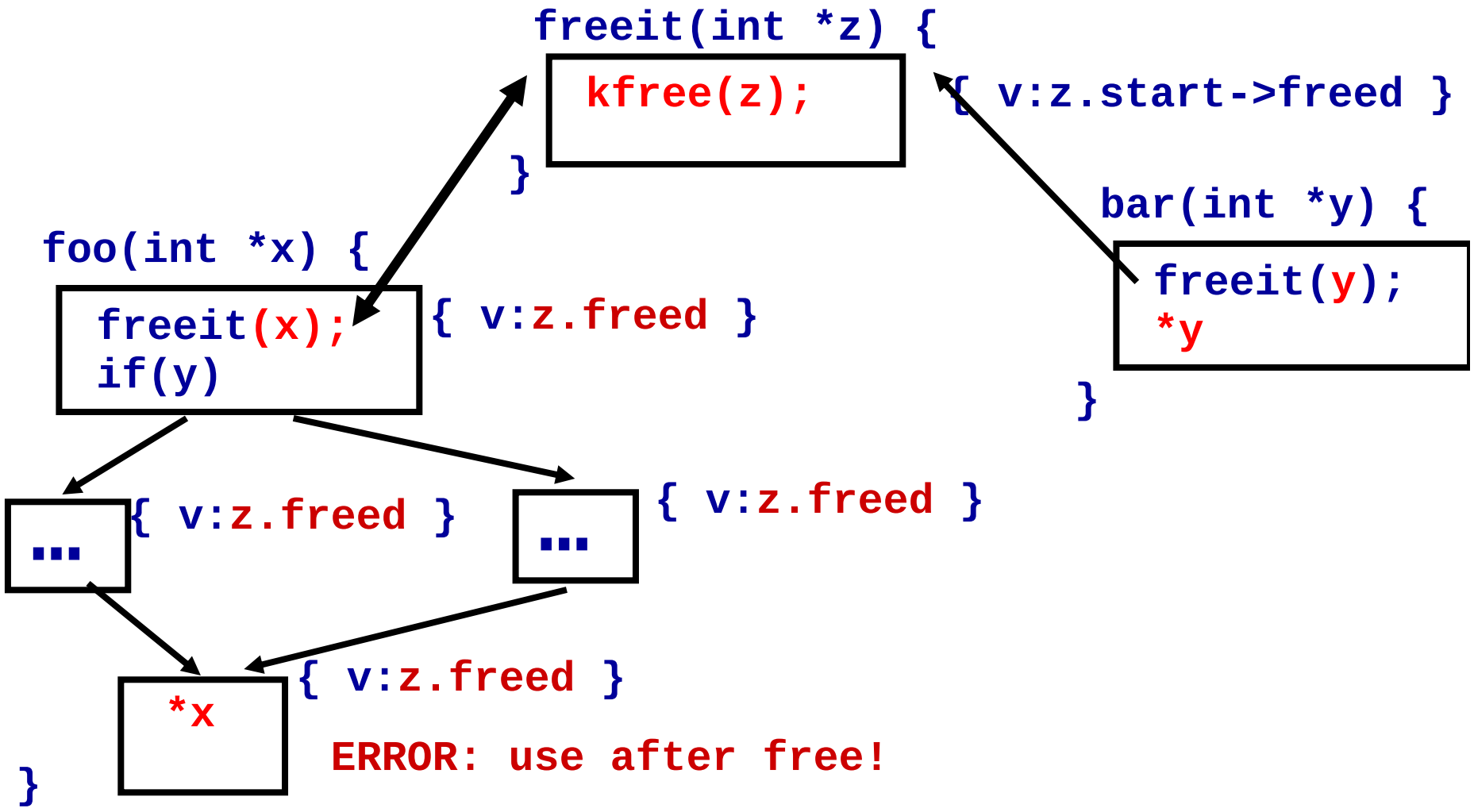
A quick analysis example



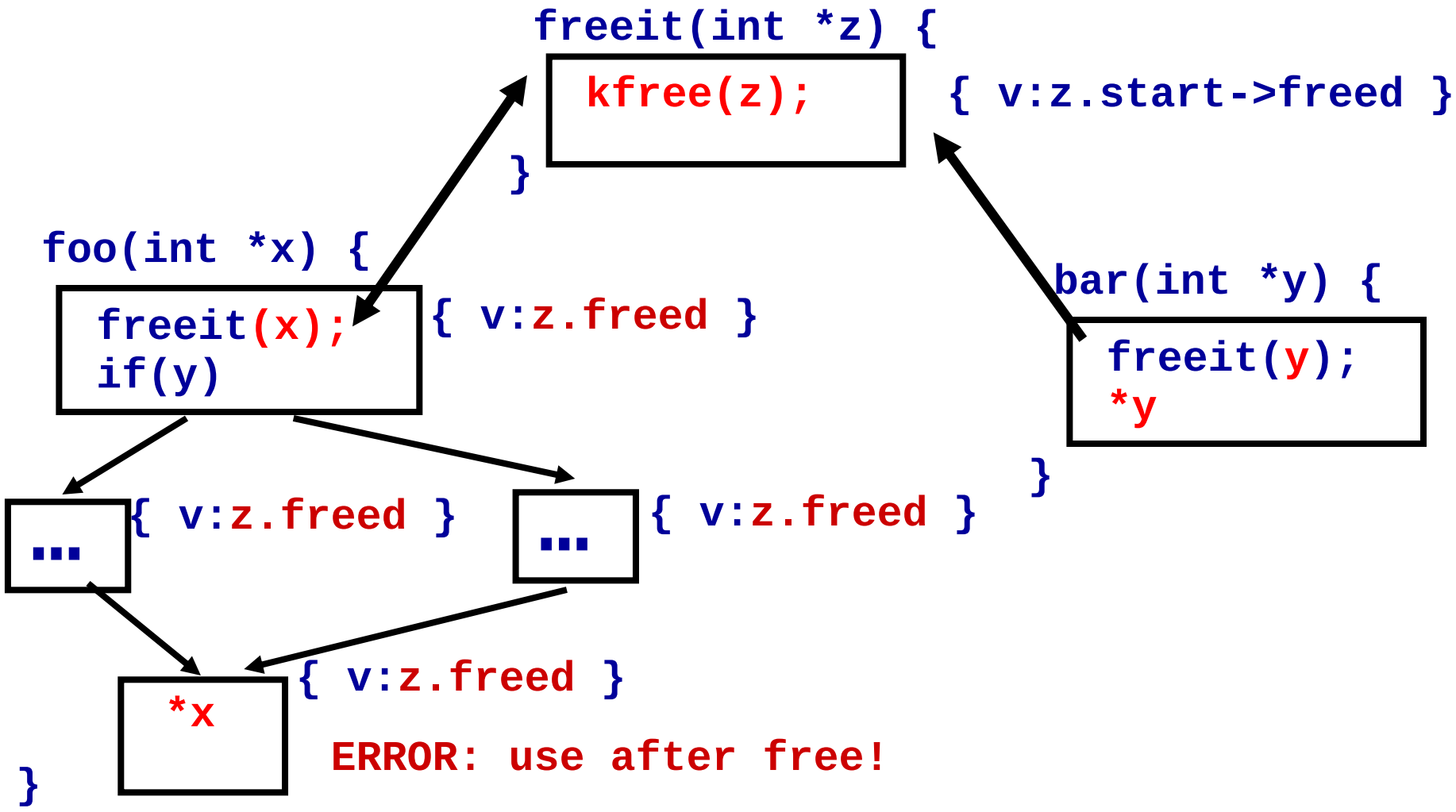
A quick analysis example



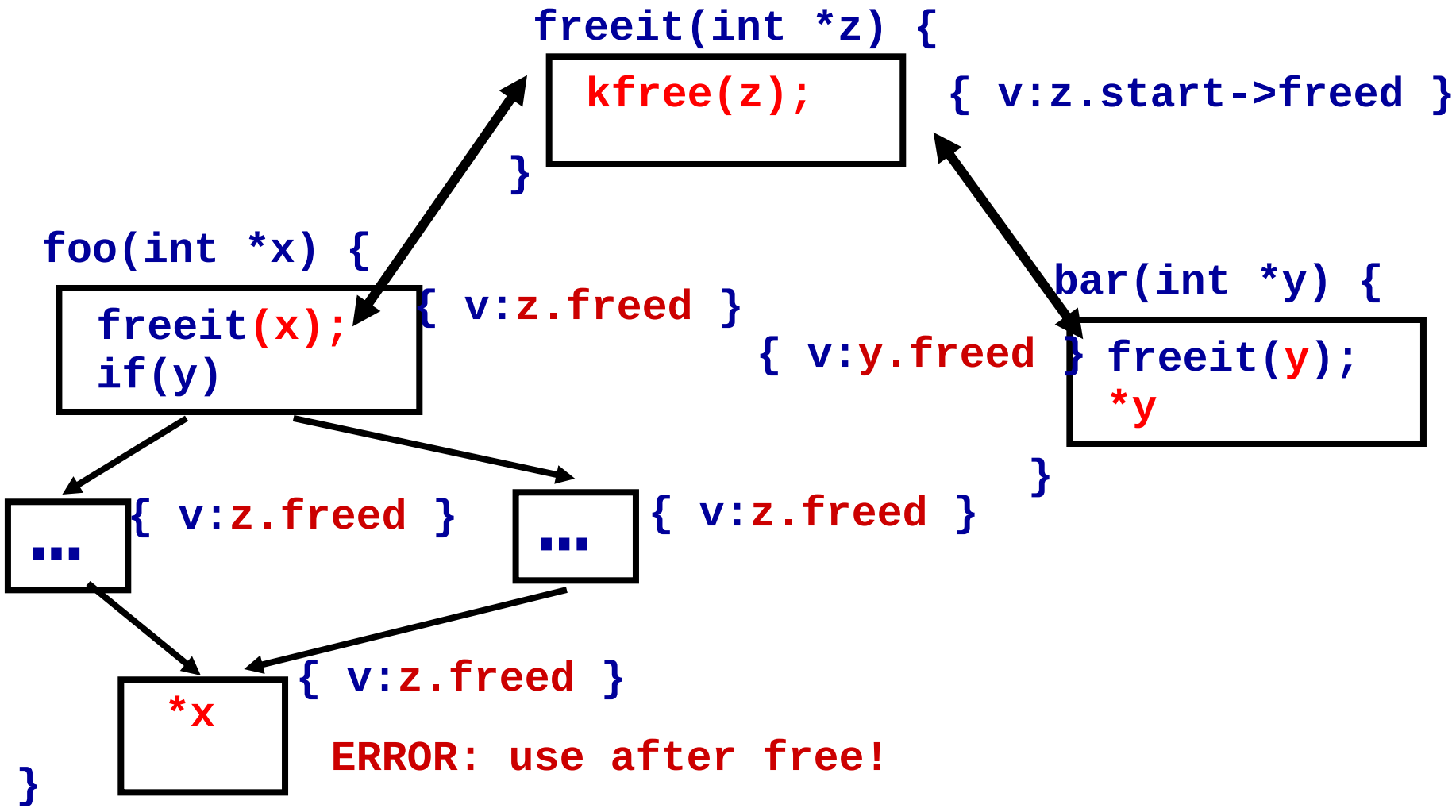
A quick analysis example



A quick analysis example

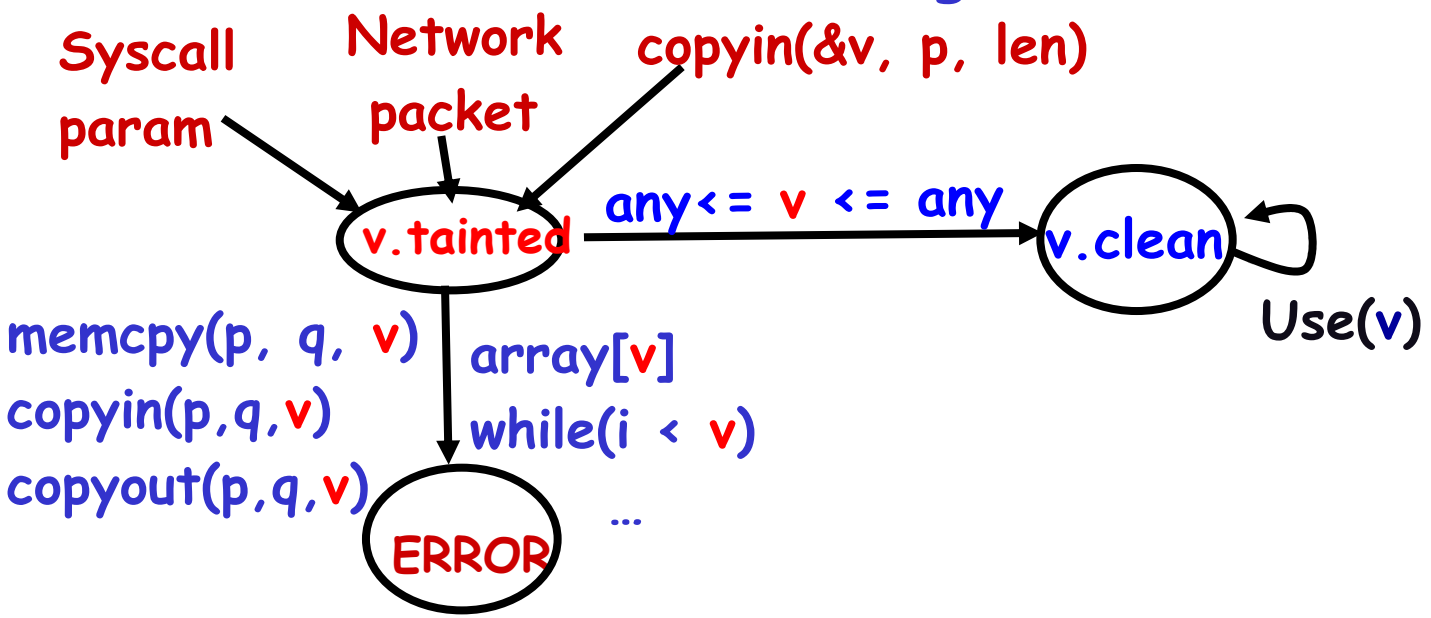


A quick analysis example



"X before Y": sanitize integers before use

- Security: OS must check user integers before use
- MC checker: Warn when unchecked integers from untrusted sources reach trusting sinks



Global; simple to retarget (text file with 2 srcs&12 sinks)

Linux: 125 errors, 24 false; BSD: 12 errors, 4 false

Some big, gaping security holes.

- Remote exploit, no checks

```
/* 2.4.9/drivers/isdn/act2000/capi.c:actcapi_dispatch */
isdn_ctrl cmd;
...
while ((skb = skb_dequeue(&card->rcvq))) {
   msg = skb->data;
   ...
   memcpy(cmd.parm.setup.phone,msg->msg.connect_ind.addr.num,
                               msg->msg.connect_ind.addr.len - 1);
```

Unexpected overflow:

```
/* 2.4.9-ac7/fs/intermezzo/psdev.c */
error = copy_from_user(&input, (char *)arg, sizeof(input));
input.path = kmalloc(input.path_len + 1, GFP_KERNEL);
if ( !input.path )
      return -ENOMEM;
error =copy_from_user(input.path,user_path, input.path_len);
```

Results for BSD 2.8 & 4 months of Linux

All bugs released to implementors; most serious fixed

	Linux		BSD	
Violation	Bug	Fixed	Bug	Fixed
Gain control of system	18	15	3	3
Corrupt memory	43	17	2	2
Read arbitrary memory	19	14	7	7
Denaial of service	17	5	0	0
Minor	28	1	0	0
Total	125	52	12	12

Local bugs	109	12
Global bugs	16	0
Bugs from inferred ints	12	0
False positives	24	4
Number of checks	~3500	594

New slides start here

- Previous slides were taken from Prof. Engler's PASTE '02 talk

Contribution

- ***Not in making new/faster/better dataflow algorithms***
- ***Contribution is showing that:***

  Cheap dataflow analysis can be used for bug-finding.

  We can encode properties to check in FSAs.

  Doing this actually works and is effective.
- ***Also, empirical studies:***

  MC vs. explicit-state model-checking (VMCAI 04)

  Studying errors in OSes (SOSP 01)

  Security vulnerabilities (IEEE S&P 02)

  Inferring specifications (SOSP 01)

Restrictions

- **Can't do full CTL with Metal.**
- **For a Metal SM:**

  Transitions cannot depend on variables outside the SM

  --> "Deterministic"

  Transitions cannot depend on states of other SMs

  Result: Track typestate of one variable in isolation

- **This is to prevent exponential blowup.**

  Can analyze each SM in isolation

  Guaranteed to hit a fixed point

Algorithms Used

- **A modification of RHS**
  - New addition: MC was originally intraprocedural
- **Summary edges say what a call to a function does to the state of an FSM**
  - Same as with ESP
- **MC doesn't seem to be exploding the supergraph**
  - Additional cost?
  - But they could just be presenting simplified pseudocode....
  - End of section 6.2 makes it sound like they don't
  - But they claim that their complexity is similar...?