RWset: Attacking Path Explosion in Constraint-Based Test Generation

Cristian Cadar, Peter Boonstoppel, Dawson Engler

STANFORD
UNIVERSITY

TACAS 2008, Budapest, Hungary
ETAPS 2008

Constraint-Based Test Generation

- Goal: generate inputs that explore (ideally) all paths of a program
- Run program on symbolic input, whose initial value is *anything*
- At conditionals that use symbolic inputs, fork execution and follow both paths:
  - On true branch, add constraint that condition is true
  - On false, that it is not
- When a path terminates, generate a test case by solving the constraints on that path

Constraint-Based Test Generation

- EGT / EXE / KLEE
- DART [Godefroid/Klarlund/Sen]
- CUTE [Sen et al.]
- SAGE, Pex [Godefroid et al.]
- Vigilante [Castro et al]
- BitScope [Song et al.]

- RWset applicable to any of these

EXE Results

- Effective bug-finding tool
  - File system code
    - ext2, ext3, JFS
  - Networking applications
    - bpf, udhcpd
  - Library code
    - PCRE, Pintos
  - Device drivers
    - Minix

Scalability challenge

- Exponential space!
  - Relatively small number of interesting paths: e.g., those that achieve maximum branch coverage
- Mixed symbolic/concrete execution (EXE/DART)
- Search heuristics
  - Best First Search (EXE)
  - Generational Search (SAGE)
- Symbolic execution + random testing (Hybrid CUTE)
- Caching function summaries (SMART)
- Demand-Driven Compositional Symbolic Execution (Pex)

RWset (read-write set) analysis

- Determine whether continuing to execute the current program path will explore new states
- Only a value observed by the program can determine the execution of new program states

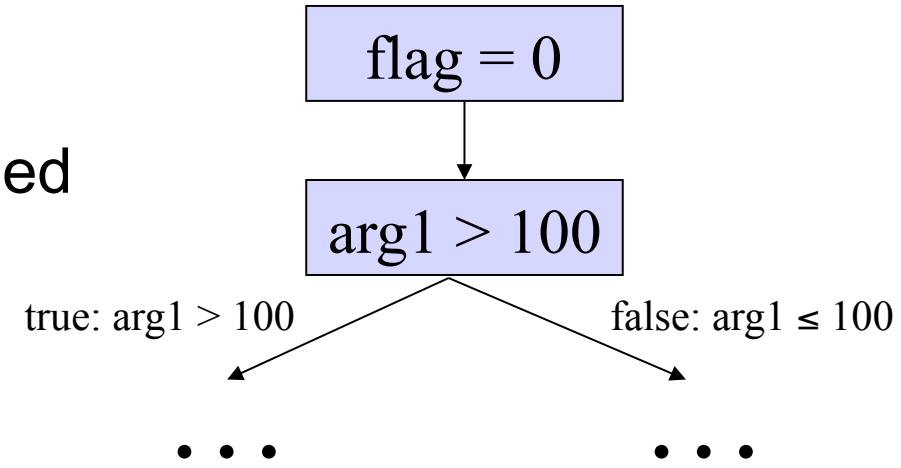
```
{data, arg1, arg2} = unconstrained

flag =  0;

if (arg1 > 100)
    flag = 1;

if (arg2 > 100)
    flag = 1;

process(data, flag);
```

flag = 0

```
{data, arg1, arg2} = unconstrained

flag =  0;

if (arg1 > 100)
    flag = 1;

if (arg2 > 100)
    flag = 1;

process(data, flag);
```

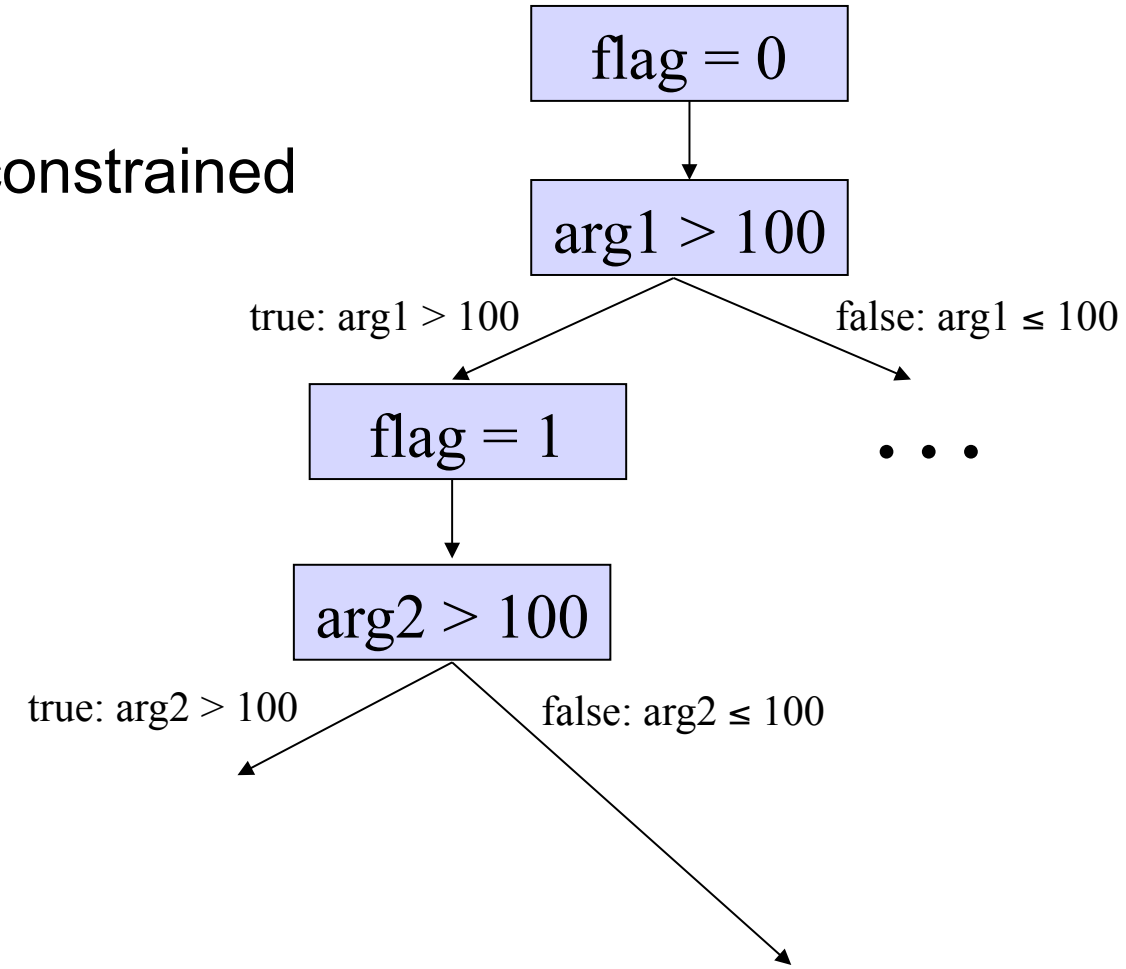
{data, arg1, arg2} = unconstrained

```
flag =  0;

if (arg1 > 100)
    flag = 1;

if (arg2 > 100)
    flag = 1;

process(data, flag);
```



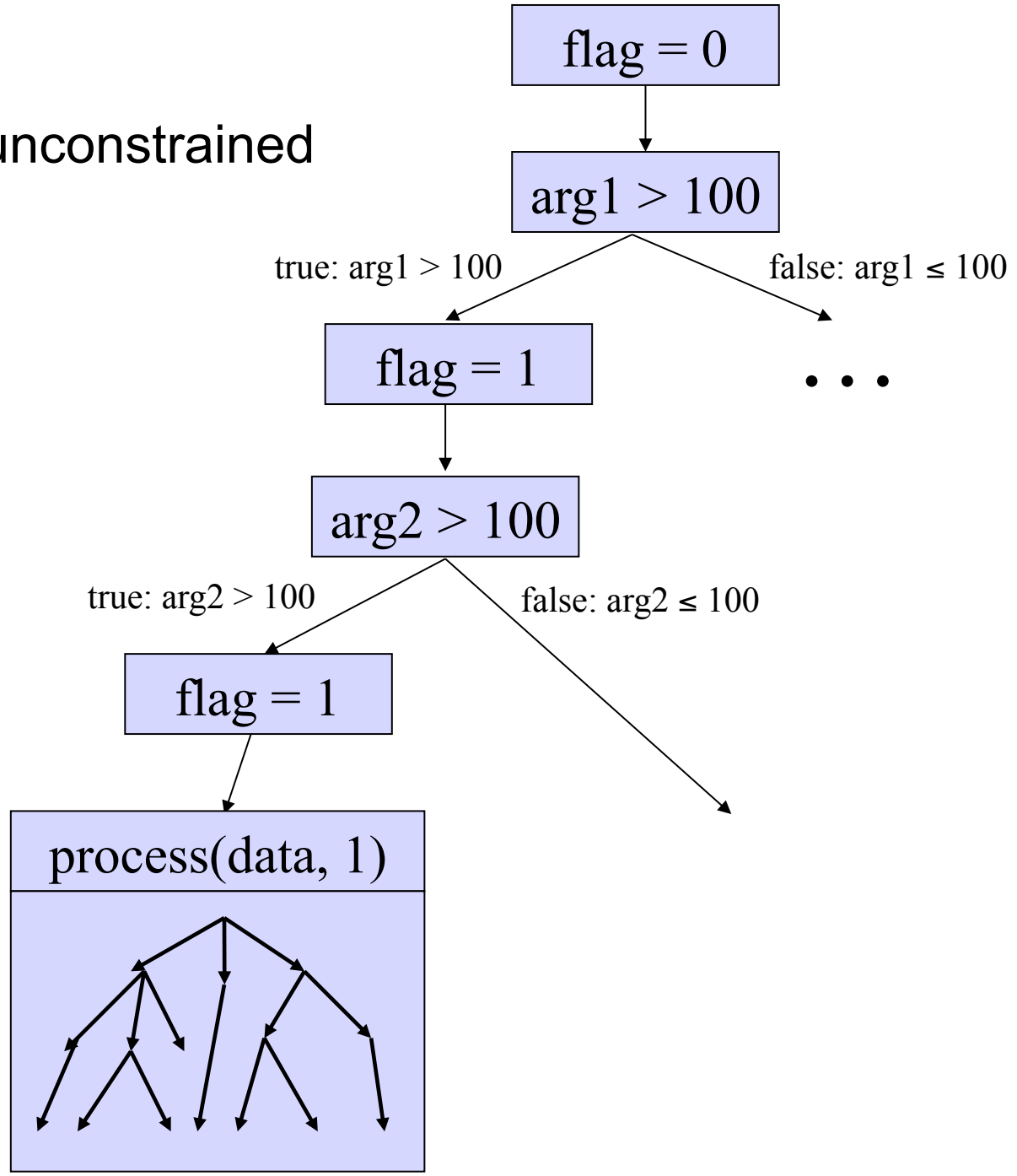
{data, arg1, arg2} = unconstrained

```
flag =  0;

if (arg1 > 100)
    flag = 1;

if (arg2 > 100)
    flag = 1;

process(data, flag);
```



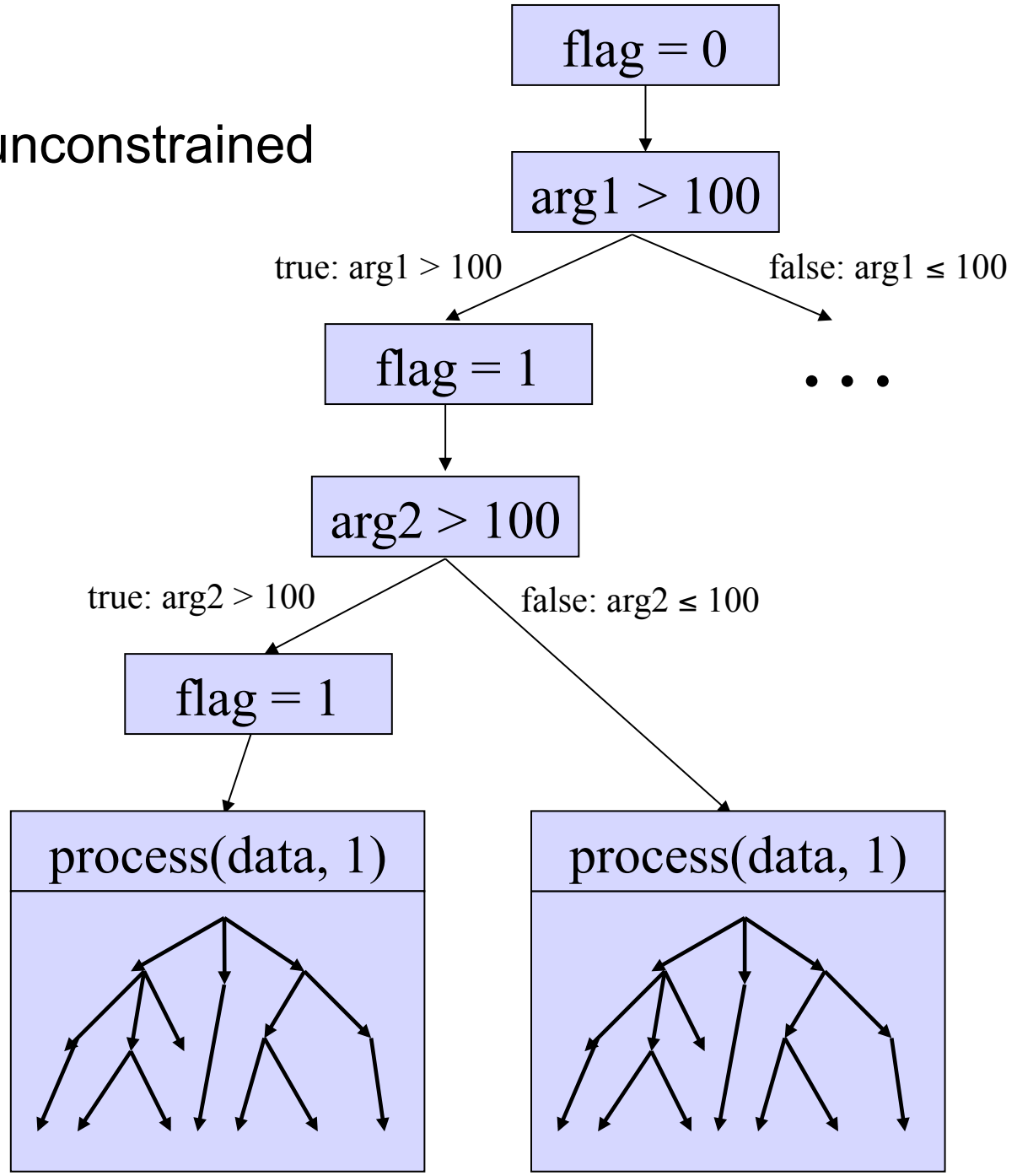
{data, arg1, arg2} = unconstrained

```
flag =  0;

if (arg1 > 100)
    flag = 1;

if (arg2 > 100)
    flag = 1;

process(data, flag);
```

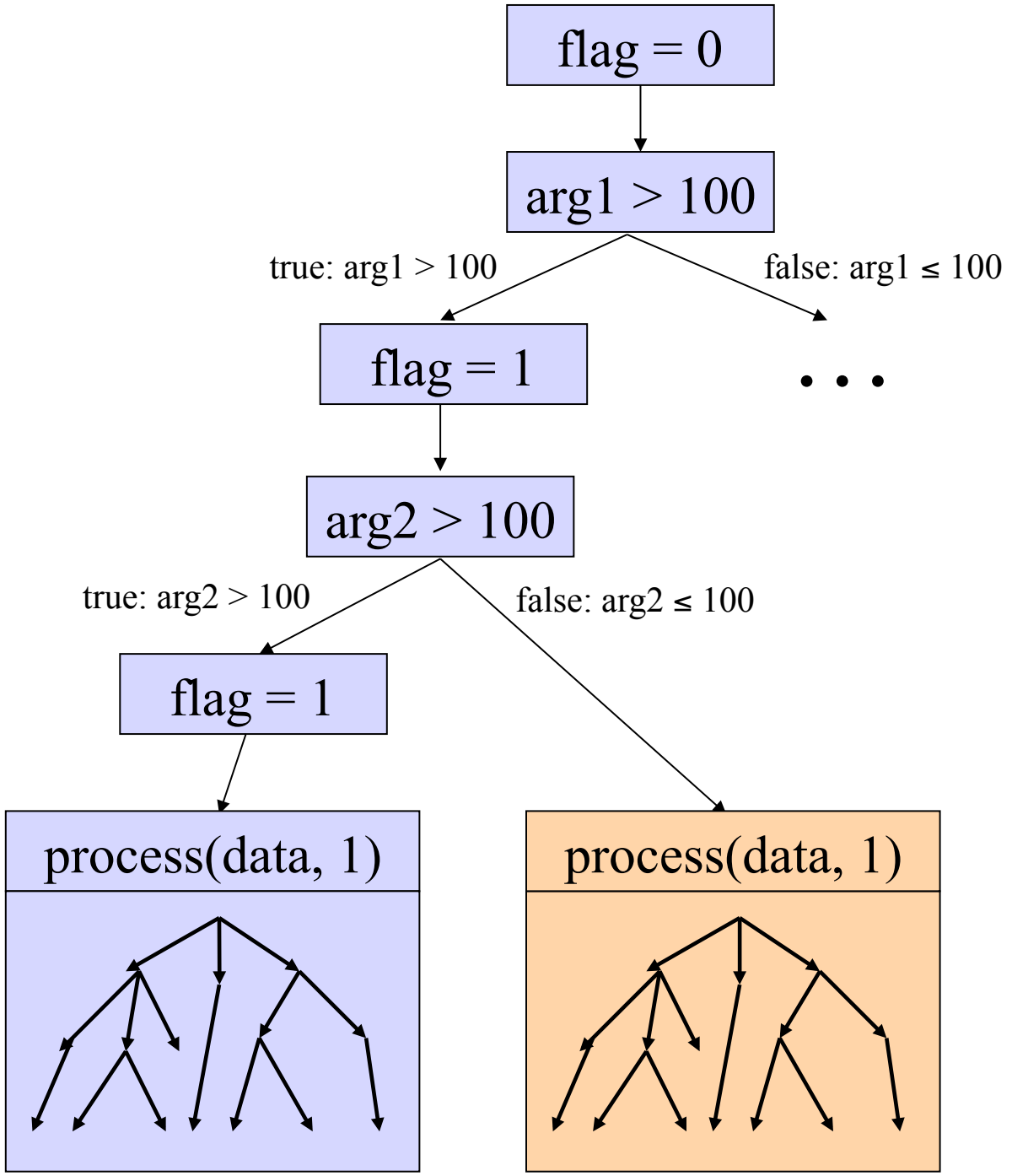


{data, arg1, arg2} = unconstrained

```
flag =  0;

if (arg1 > 100)
    flag = 1;

if (arg2 > 100)
    flag = 1;

process(data, flag);
```



If arg1, arg2 not read



Write-set analysis

- Look at values written in the past
- State abstraction in model checking
- Memory state = *write set* up to current progr. pt.
  - Concrete writes: *concr loc = concr val*
  - Symbolic writes: *constraint(sym loc)*

- Program point P, two paths w/ same write-set
  - prune the second one

Write-sets

- Precision: reason at the byte-level
- Minimize write-set size
  1. Overwrites
  2. Dead locations
  3. Alpha renaming
- Complicated in the symbolic domain

```
a[i] = 17;
a[j] = 20;
```

- Cannot discard all constraints on dead locations

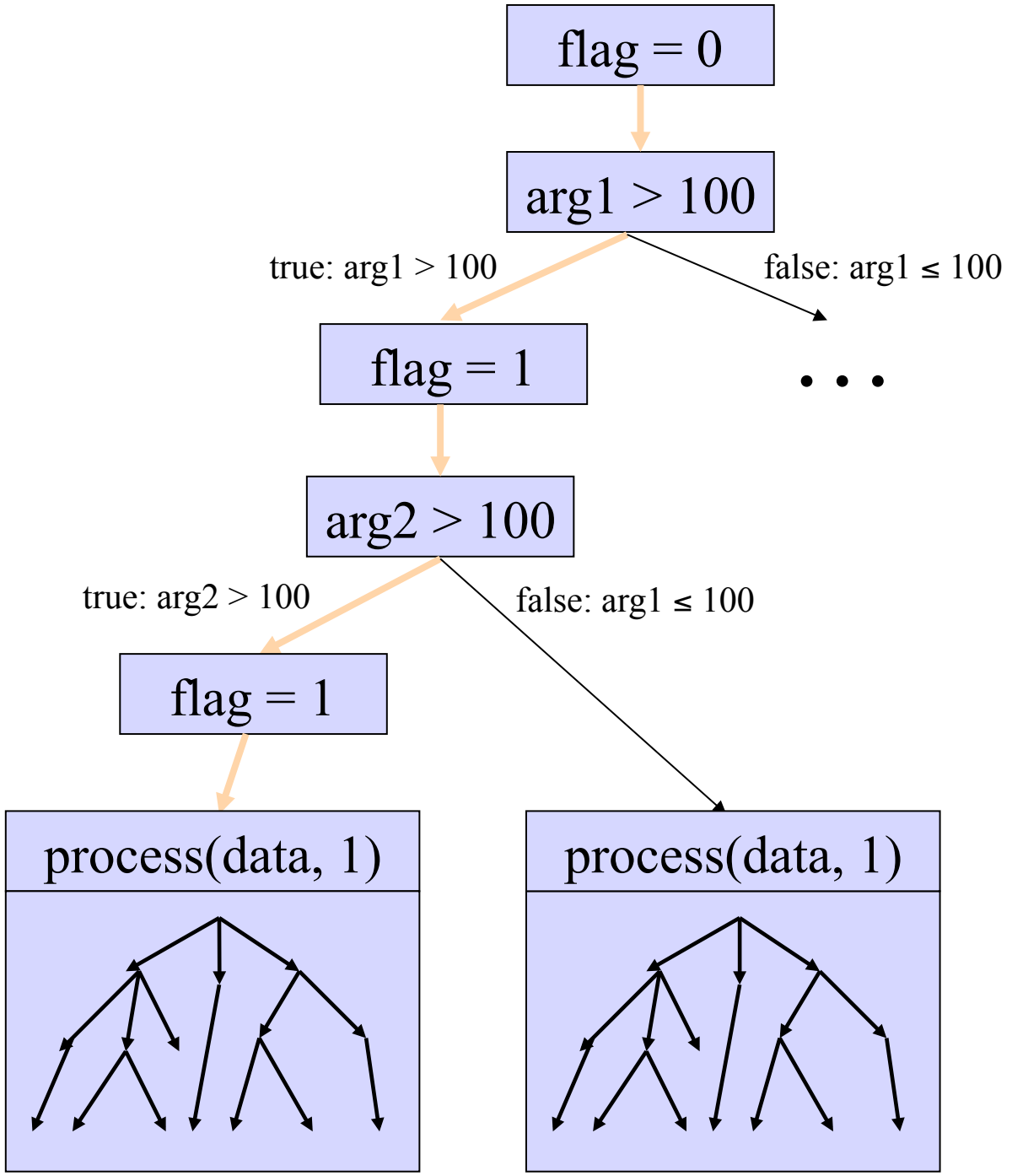
Read-set analysis

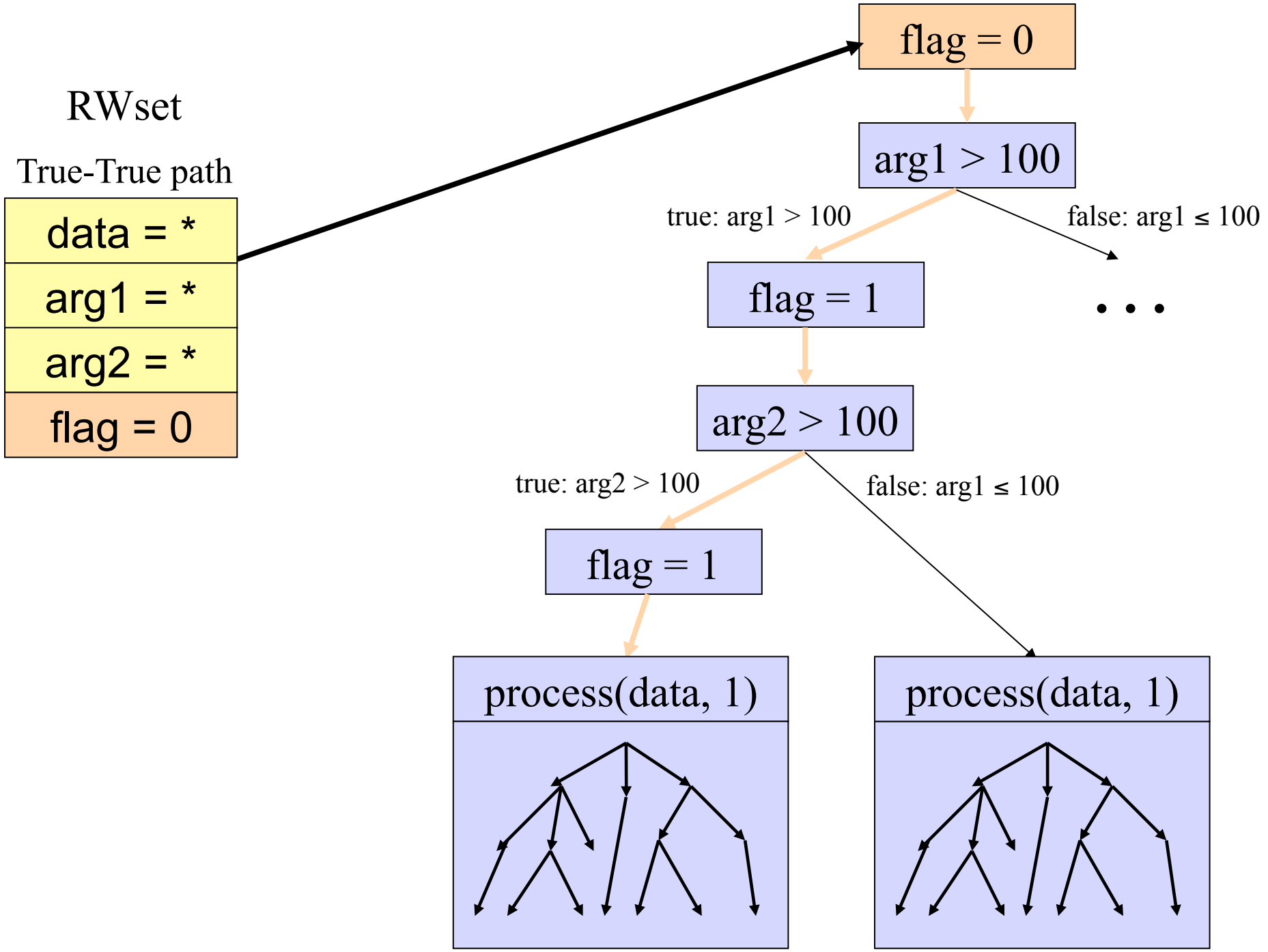
- *Key idea:* can ignore from write-set writes to locations that are never *read* again
  - Definition of *read* driven by goal to achieve high branch coverage
  - Location read if can hit a new branch by changing its value

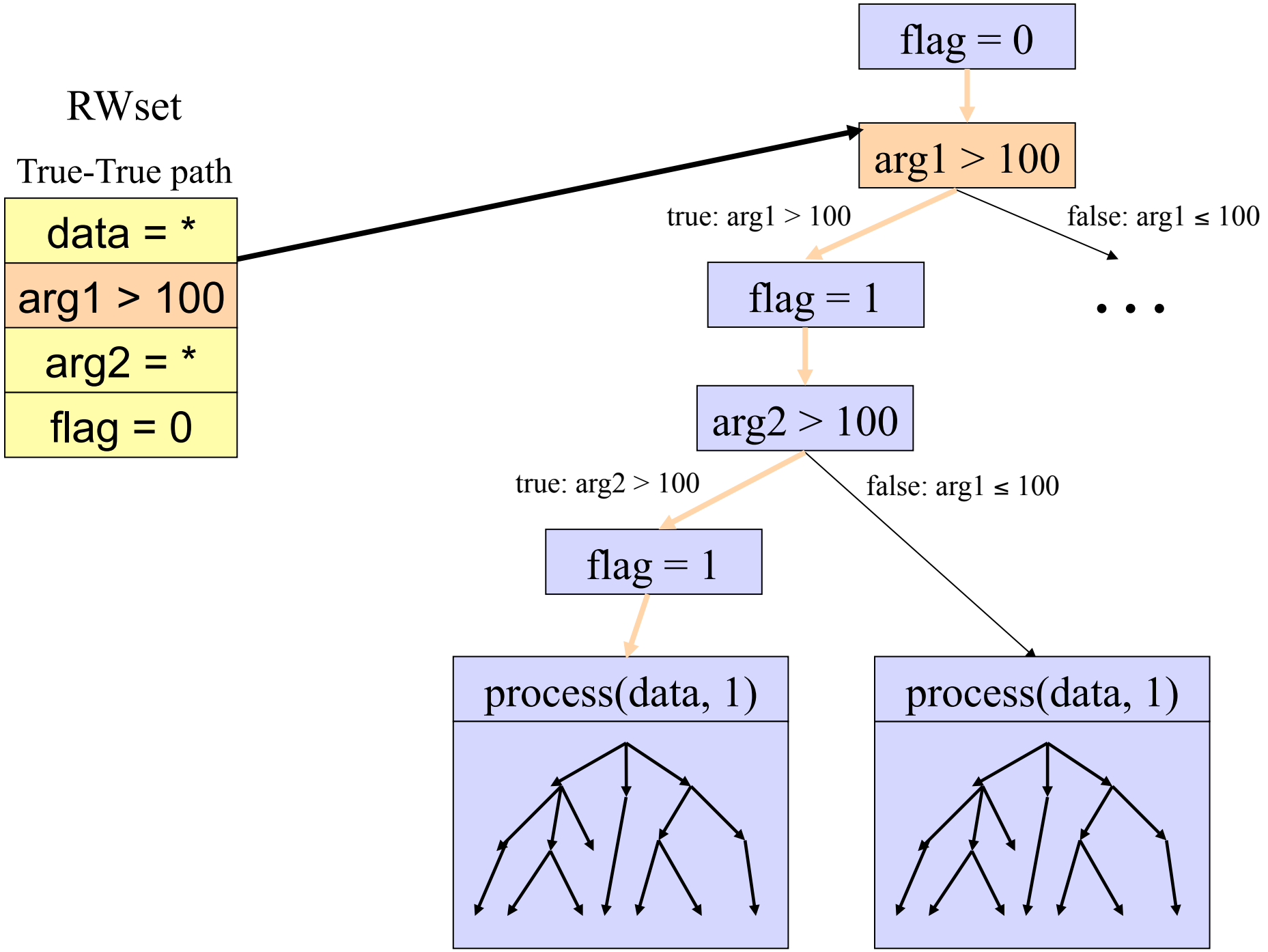
RWset

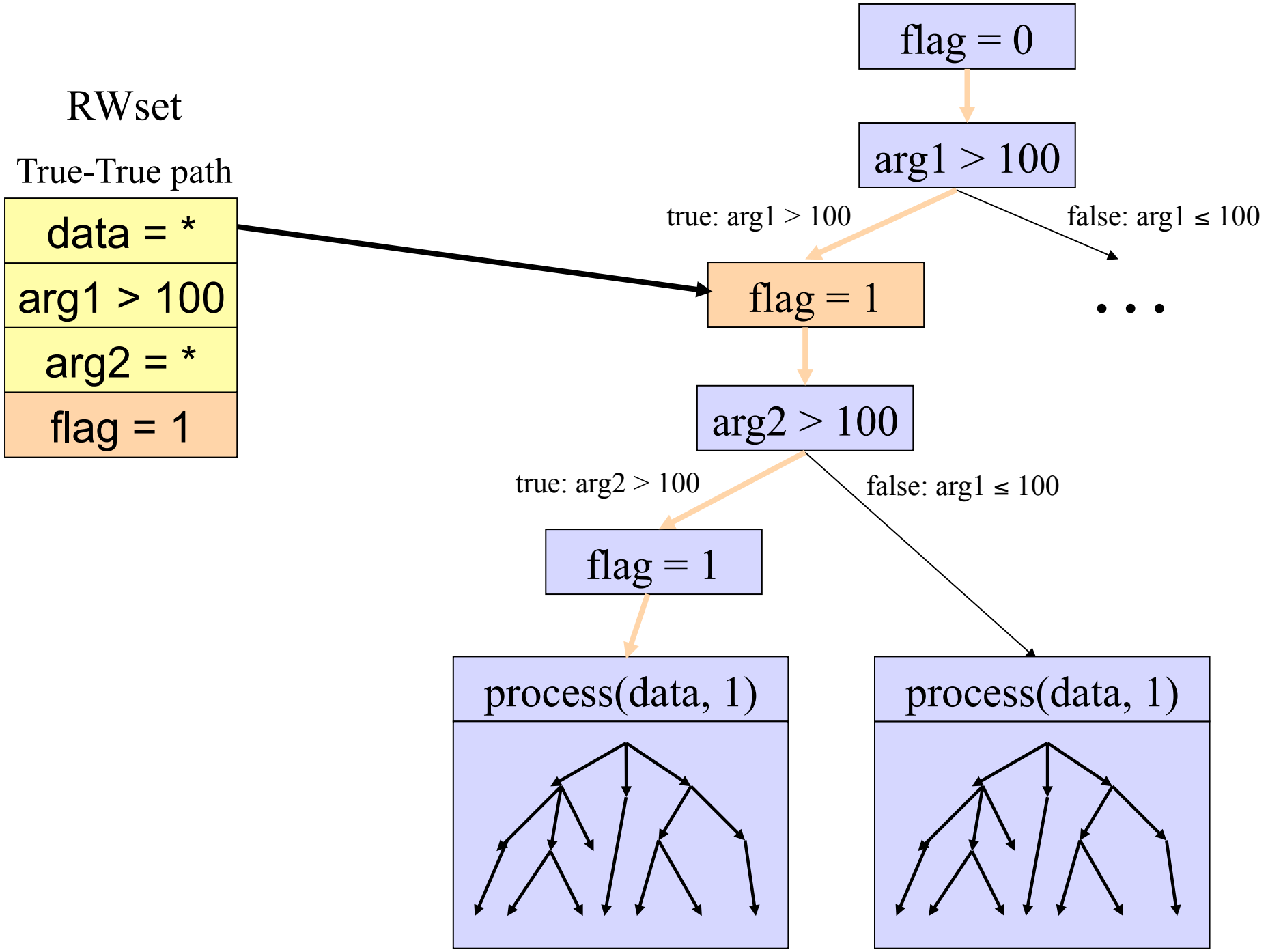
True-True path

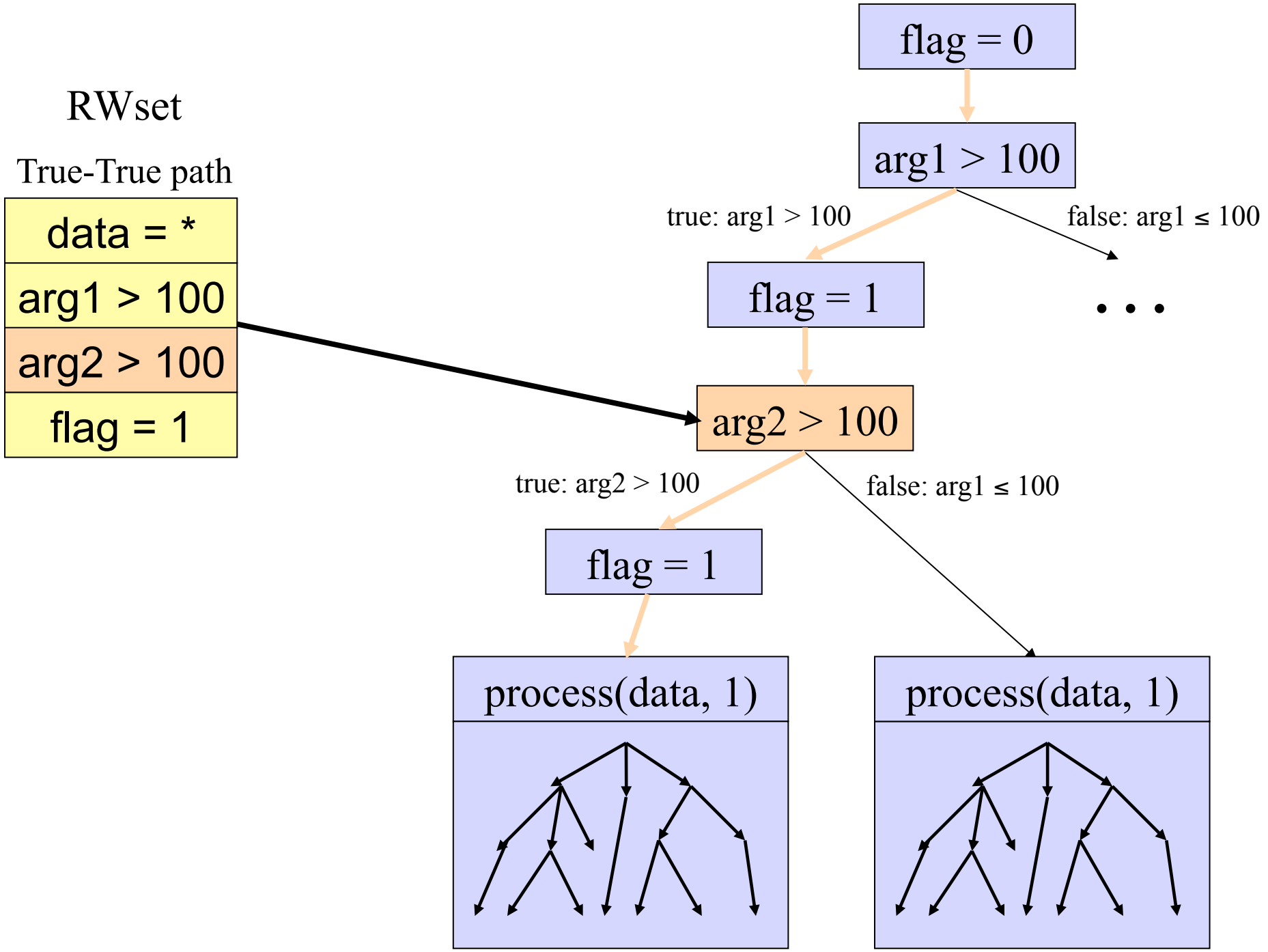
data = *
arg1 = *
arg2 = *

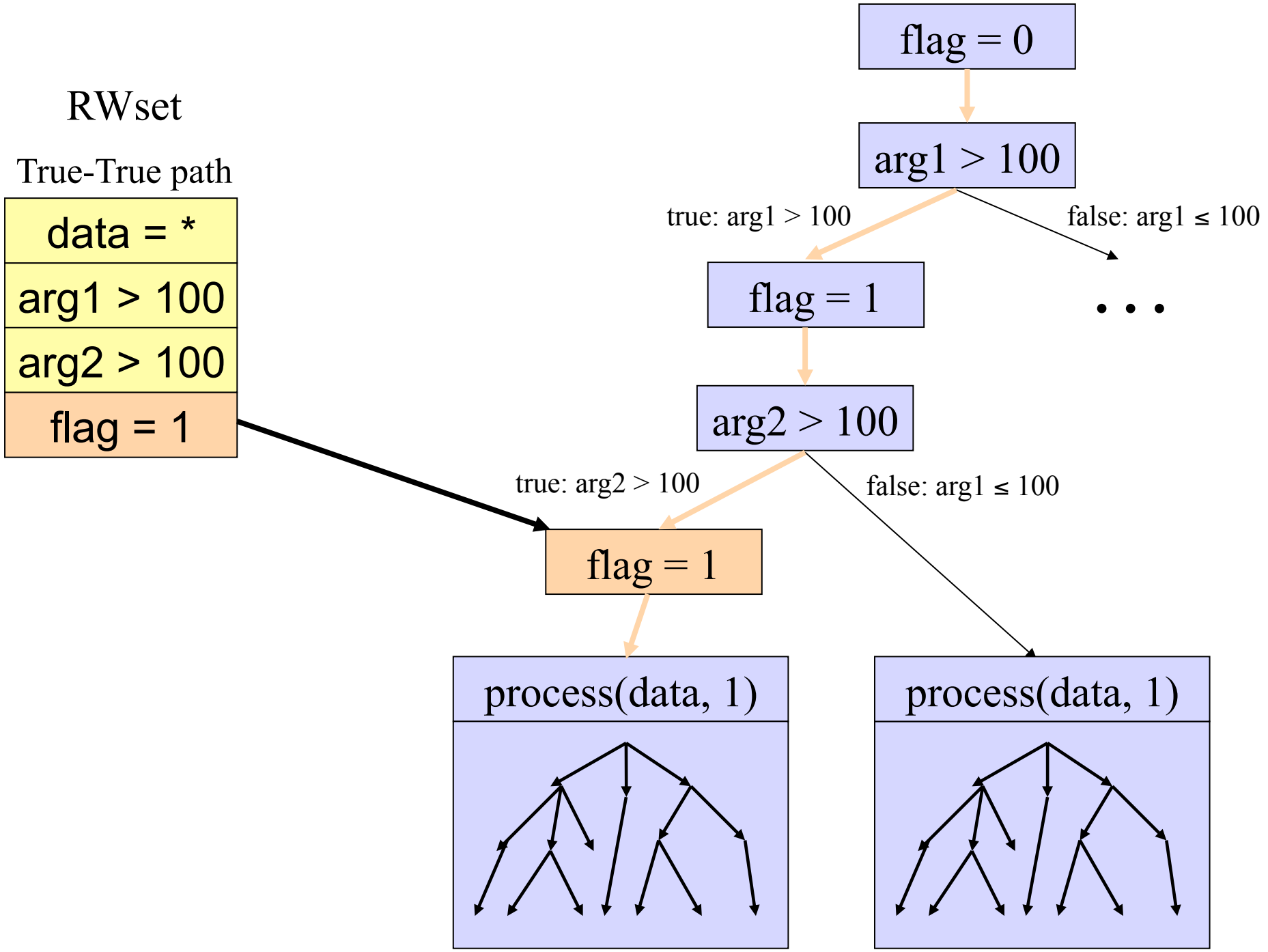


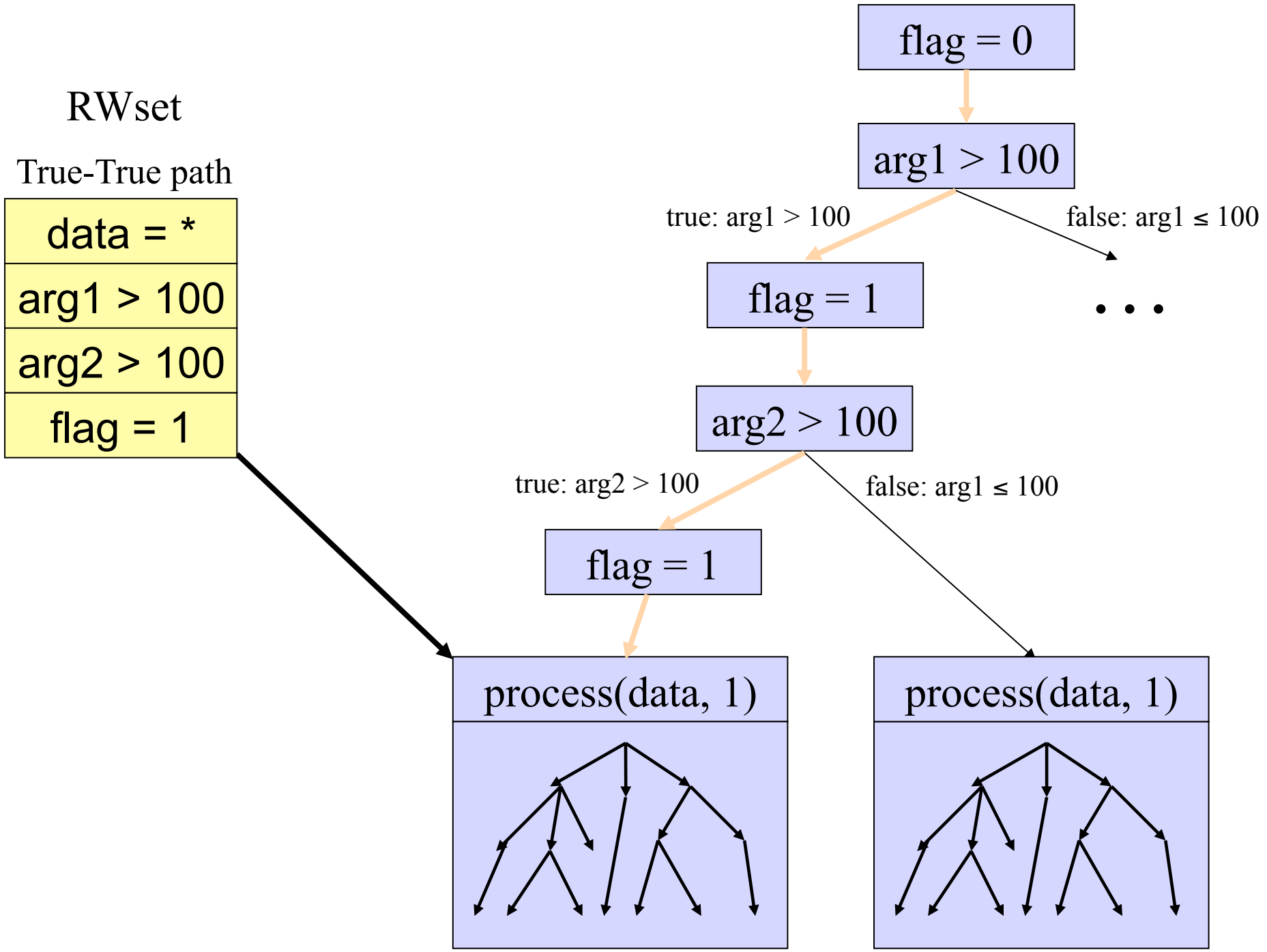

RWset
True-True path
data = *
arg1 = *
arg2 = *
flag = 0
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
• • •
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)

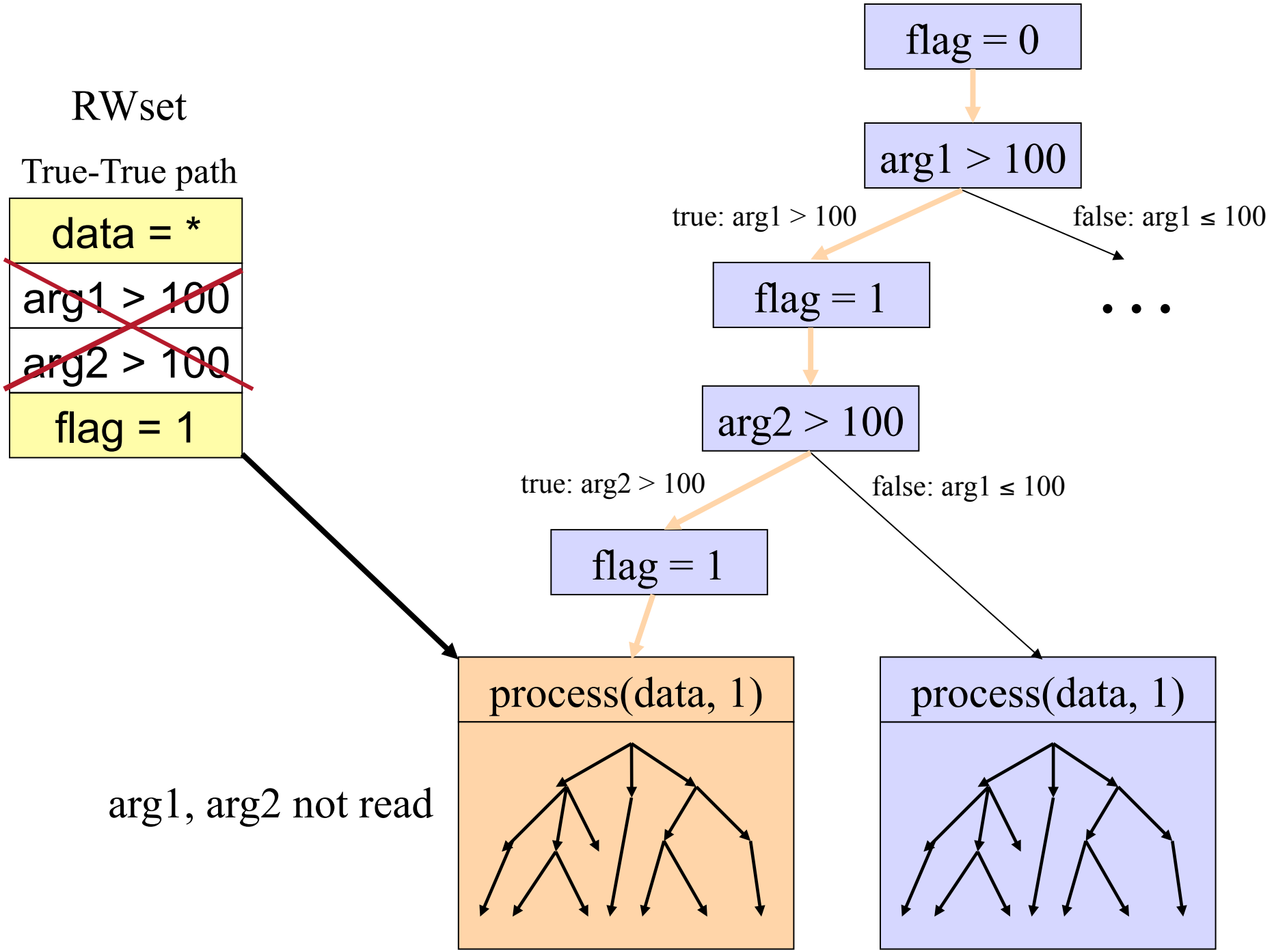

RWset
True-True path
data = *
arg1 > 100
arg2 = *
flag = 0
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
. . .
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)

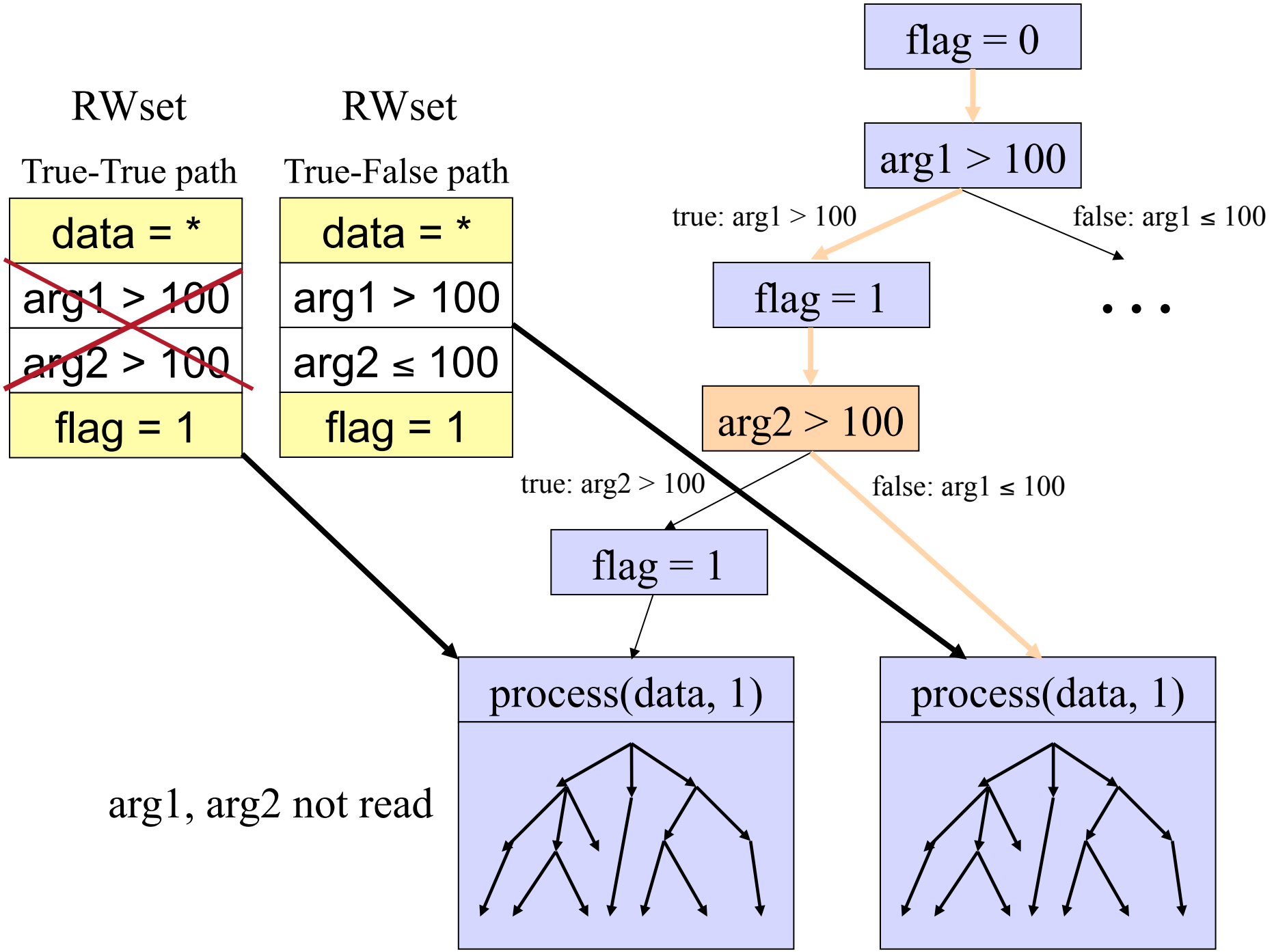

RWset
True-True path
data = *
arg1 > 100
arg2 = *
flag = 1
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)

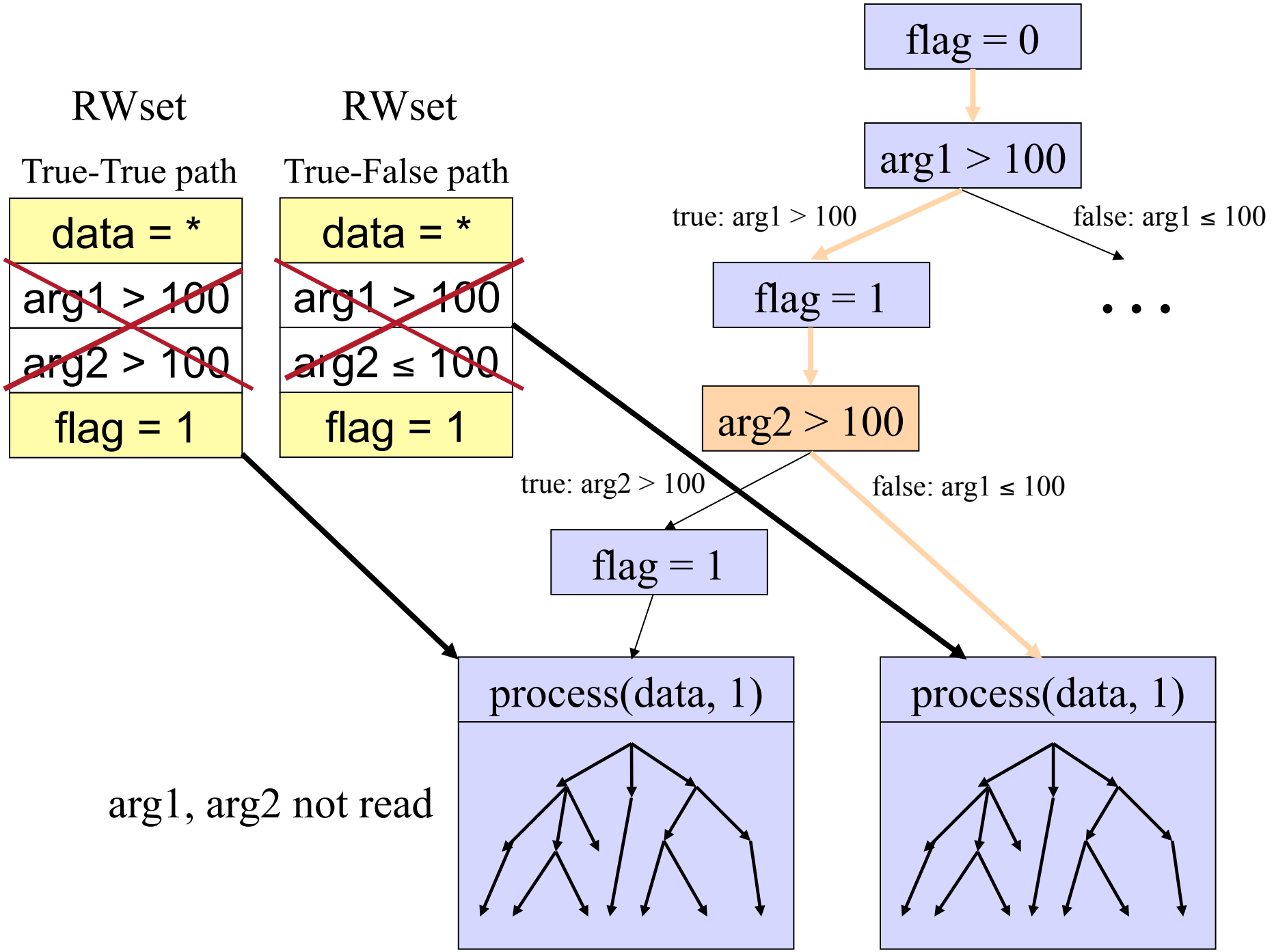

RWset
True-True path
data = *
arg1 > 100
arg2 > 100
flag = 1
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
• • •
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)

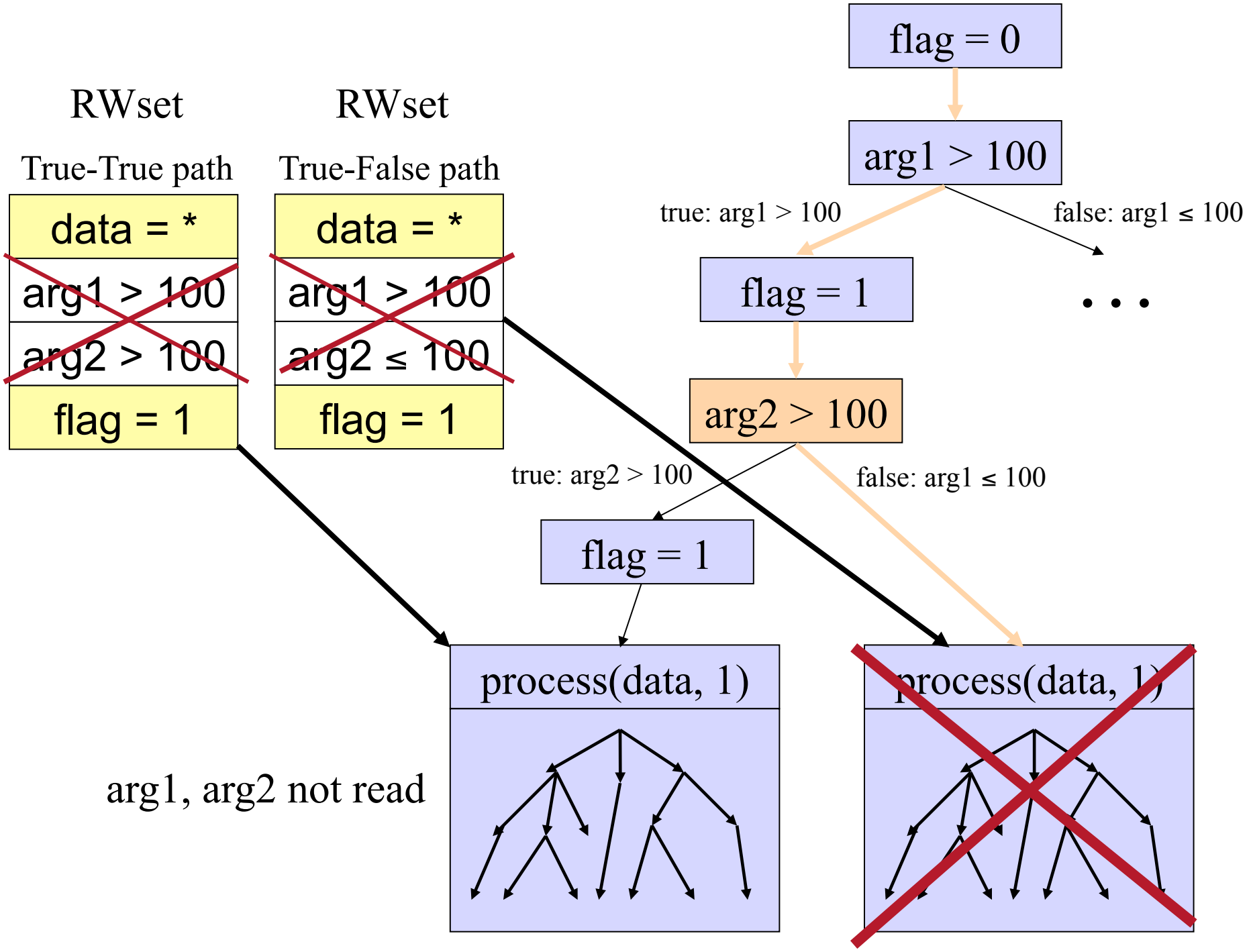

RWset
True-True path
data = *
arg1 > 100
arg2 > 100
flag = 1
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)


RWset
True-True path
data = *
arg1 > 100
arg2 > 100
flag = 1
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)


RWset
True-True path
data = *
arg1 > 100
arg2 > 100
flag = 1
arg1, arg2 not read
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)


RWset
True-True path
data = *
arg1 > 100
arg2 > 100
flag = 1
RWset
True-False path
data = *
arg1 > 100
arg2 ≤ 100
flag = 1
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)
arg1, arg2 not read

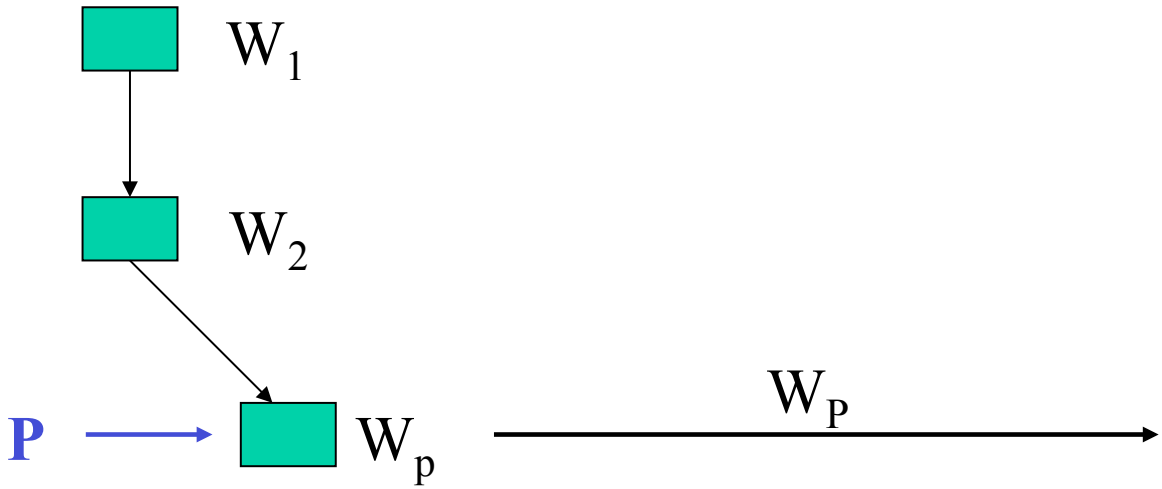

RWset
True-True path
data = *
arg1 > 100
arg2 > 100
flag = 1
RWset
True-False path
data = *
arg1 > 100
arg2 ≤ 100
flag = 1
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)
arg1, arg2 not read


RWset
True-True path
data = *
arg1 > 100
arg2 > 100
flag = 1
RWset
True-False path
data = *
arg1 > 100
arg2 ≤ 100
flag = 1
flag = 0
arg1 > 100
true: arg1 > 100
false: arg1 ≤ 100
flag = 1
arg2 > 100
true: arg2 > 100
false: arg1 ≤ 100
flag = 1
process(data, 1)
process(data, 1)
arg1, arg2 not read

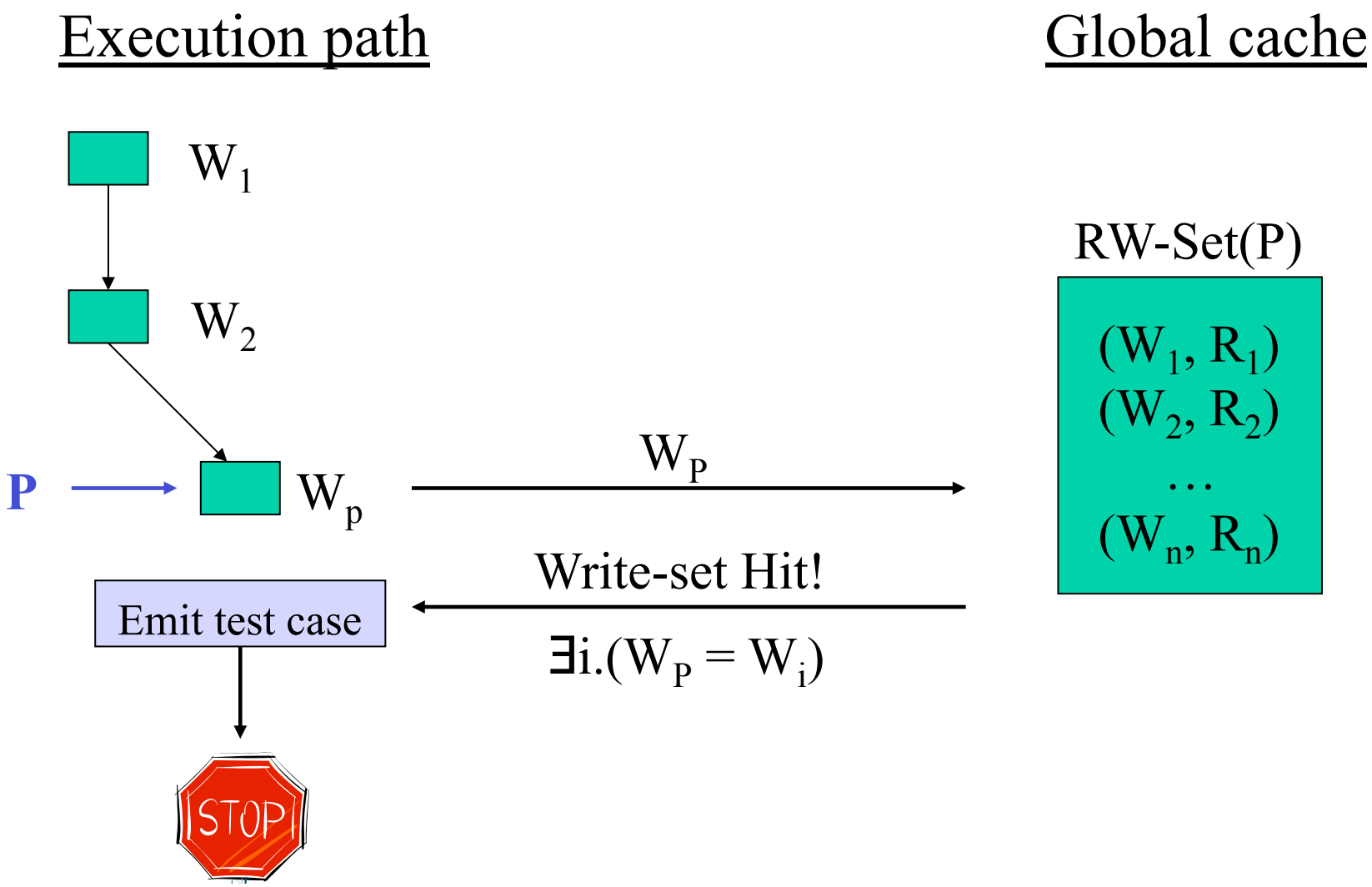
Implementation

Execution path

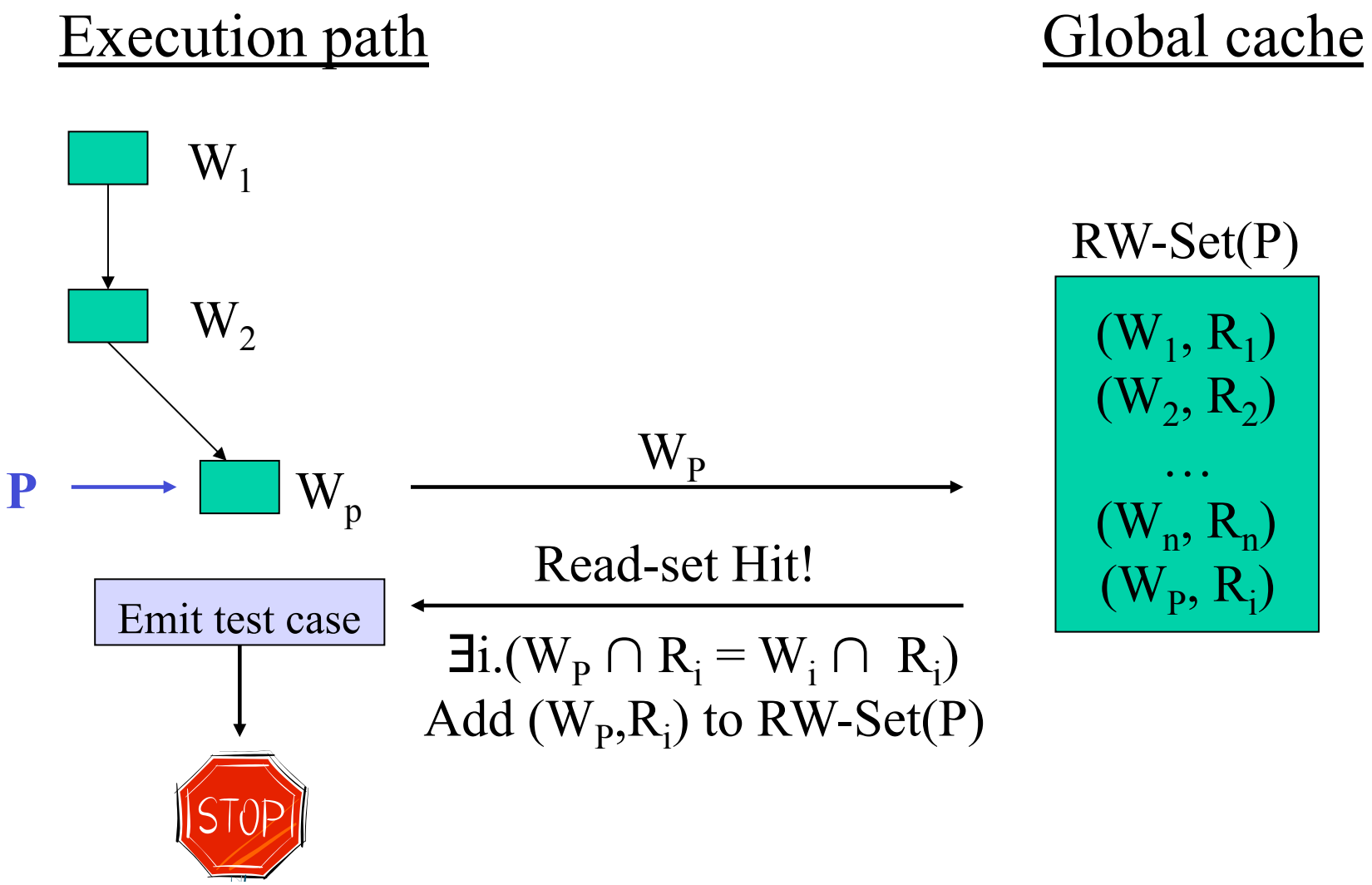
Global cache



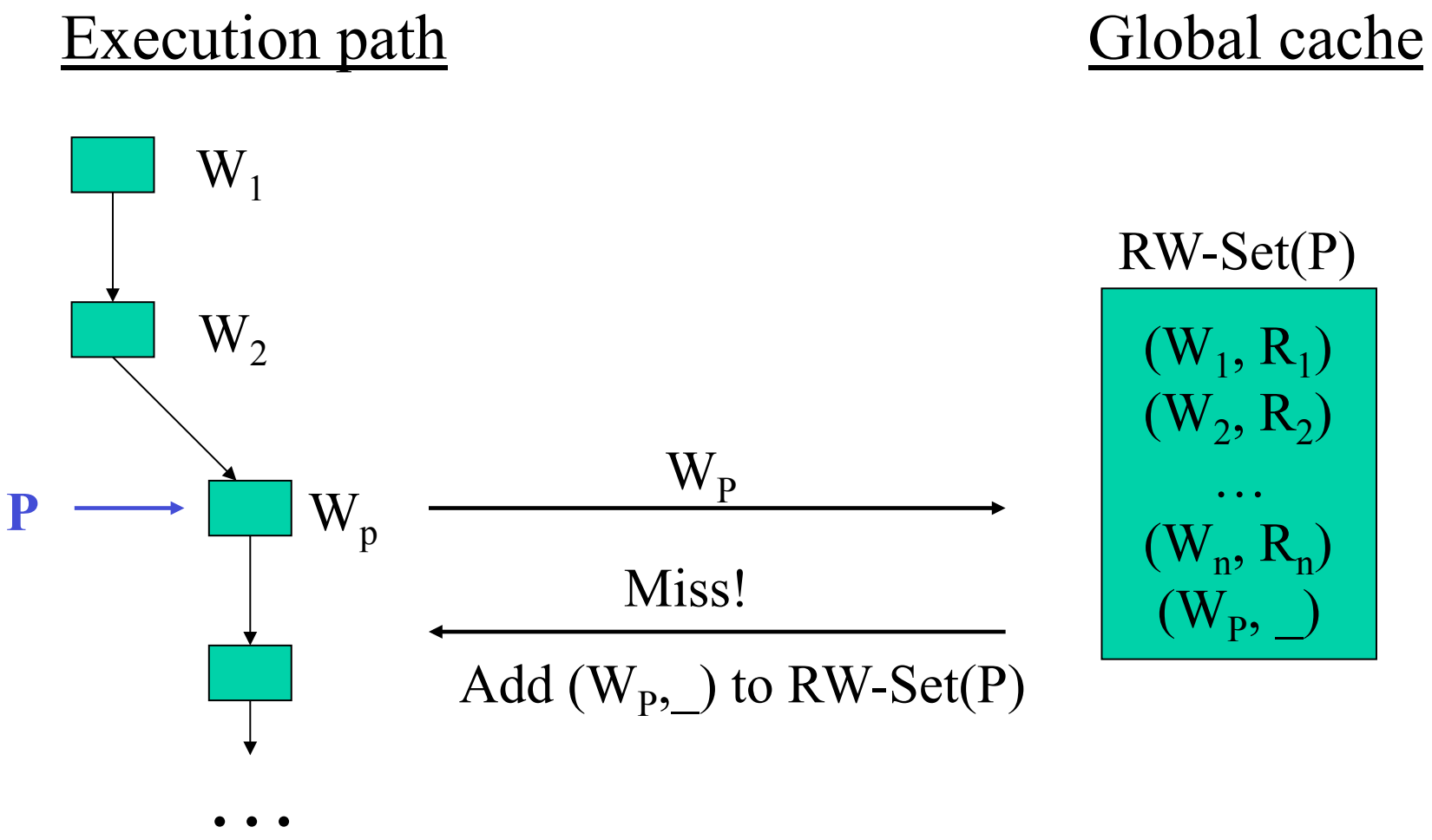
Implementation



Implementation



Implementation



Evaluation

- Medium-sized open source benchmarks
  - bpf, udhcpd, expat, tcpdump, pcre
- Minix 3 device drivers
  - lance, pci, sb16

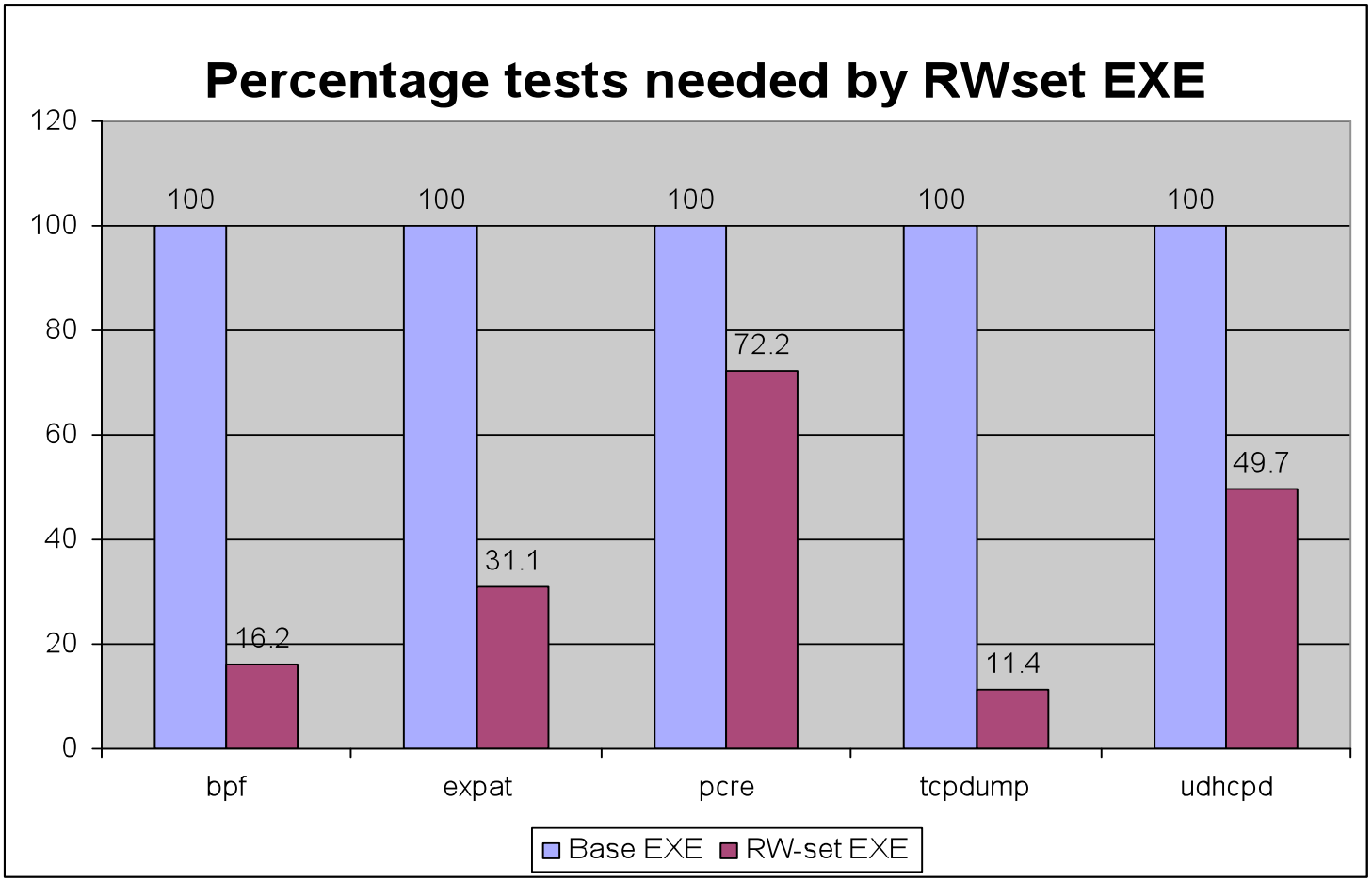
Medium-sized apps

- *bpf*: Berkeley Packet Filter
- *expat*: XML parsing library
- *pcre*: Perl compatible reg exp library
- *tcpdump*: tool for printing packet headers
- *udhcpd*: a DHCPD server

Medium-sized apps

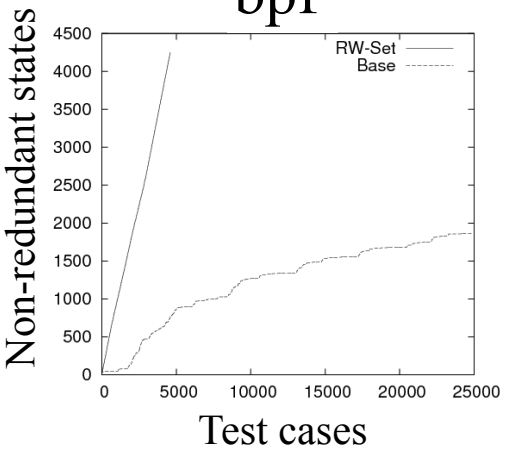
- Ran base EXE on each app for 30 minutes
  - Except 30,000 test cases for PCRE
- Recorded number of branches hit
- Reran in RWset mode until we reached the same branch coverage

Medium-sized apps

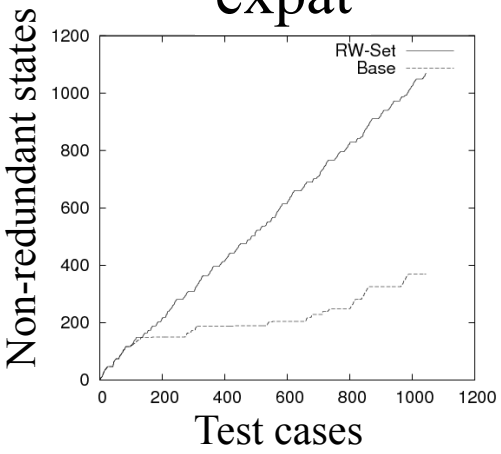


Non-redundant states

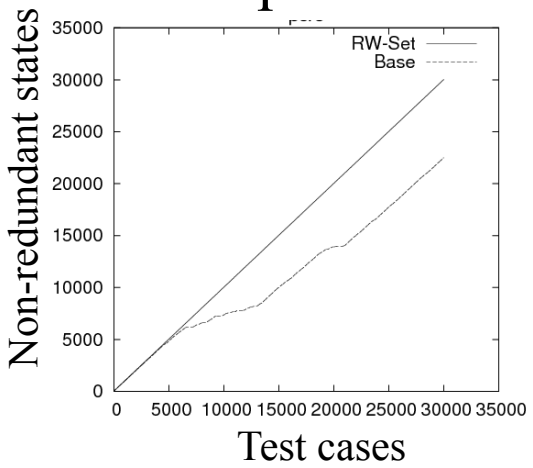
bpf



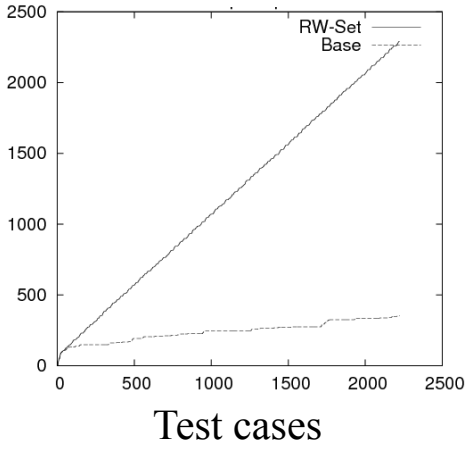
expat



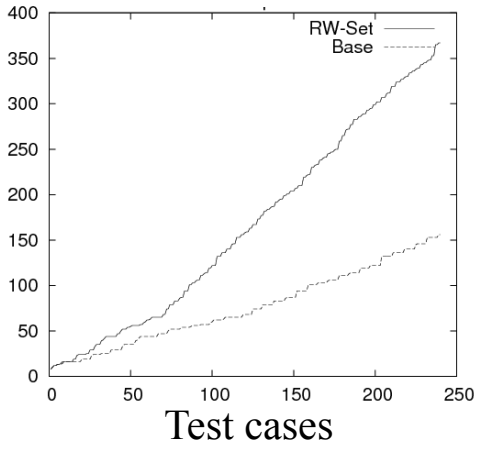
pcre



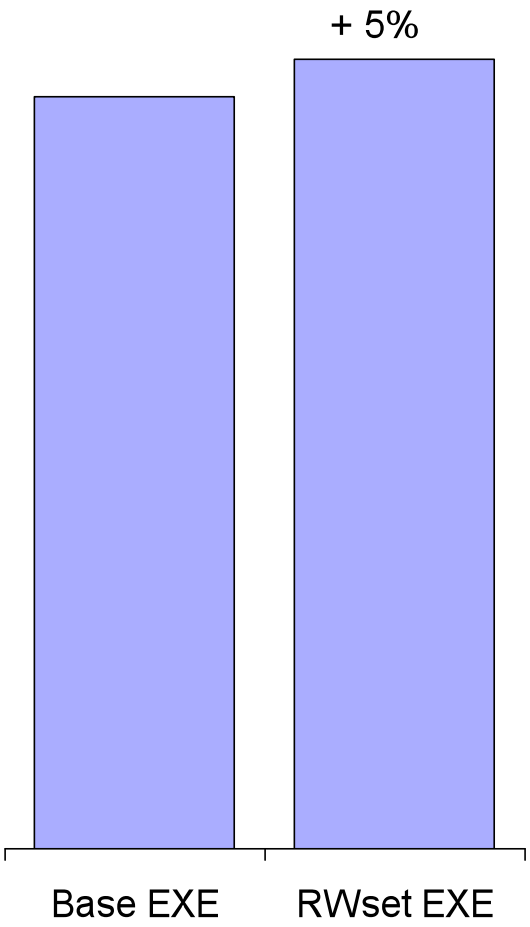
tcpdump



udhcpd



Performance



- Dry run for the medium-sized apps: compute write-sets and read-sets but do no pruning

Device drivers

- Hard to check w/o the physical device or outside of the kernel
- Focused on three MINIX 3 drivers:
  - lance: AMD lance ethernet card driver
  - pci: PCI bus driver
  - sb16: Sound Blaster 16 driver

Minix 3 device drivers

- Structured around a main dispatch loop

```
while (1) {
        /* receive message from other processes,
           the kernel, or the hardware */
        message = read_message();
        process(message);
  }
```

Minix 3 device drivers

- Structured around a main dispatch loop

```
while (1) {
        /* receive message from other processes,
           the kernel, or the hardware */
        message = read_symbolic_data();
        process(message);
  }
```

Minix 3 device drivers

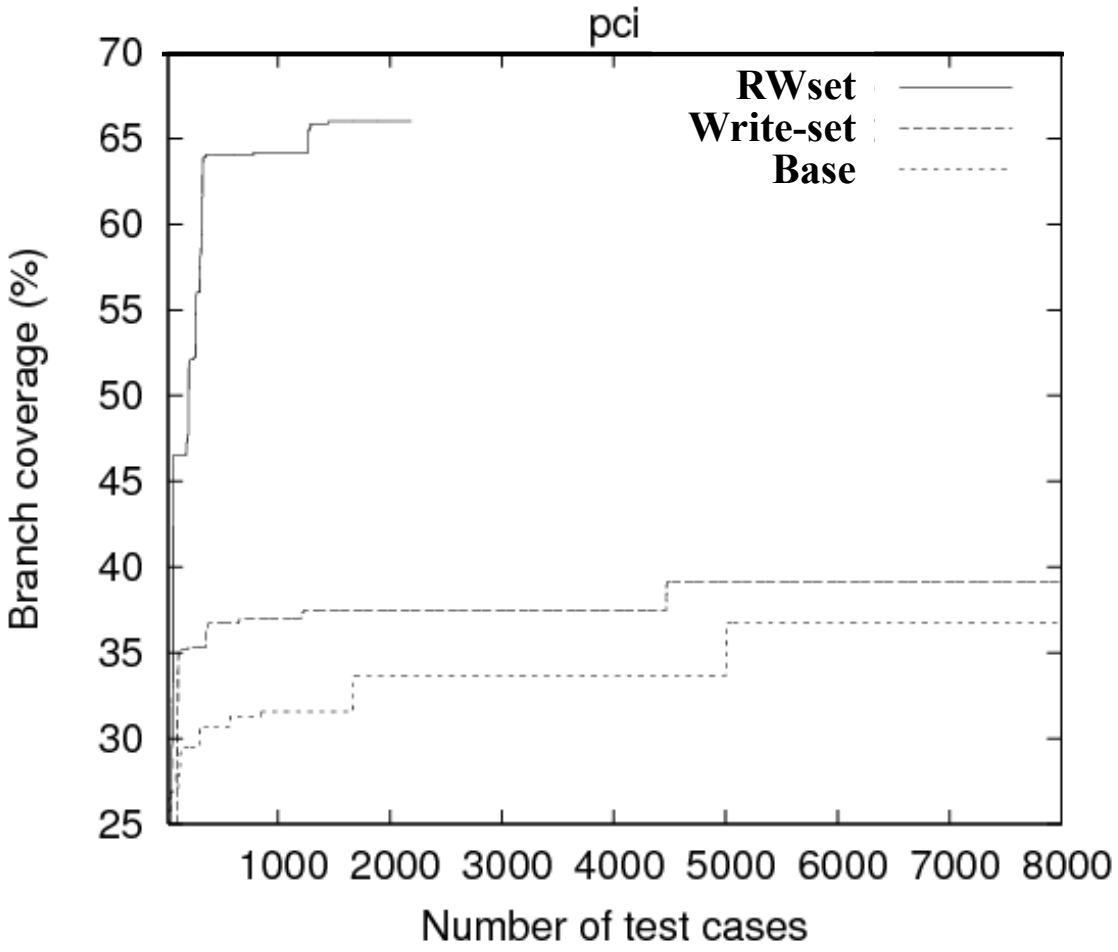
- Structured around a main dispatch loop

```
for (k=0; k < n; k++) {
        /* receive message from other processes,
           the kernel, or the hardware */
        message = read_symbolic_data();
        process(message);
  }
```

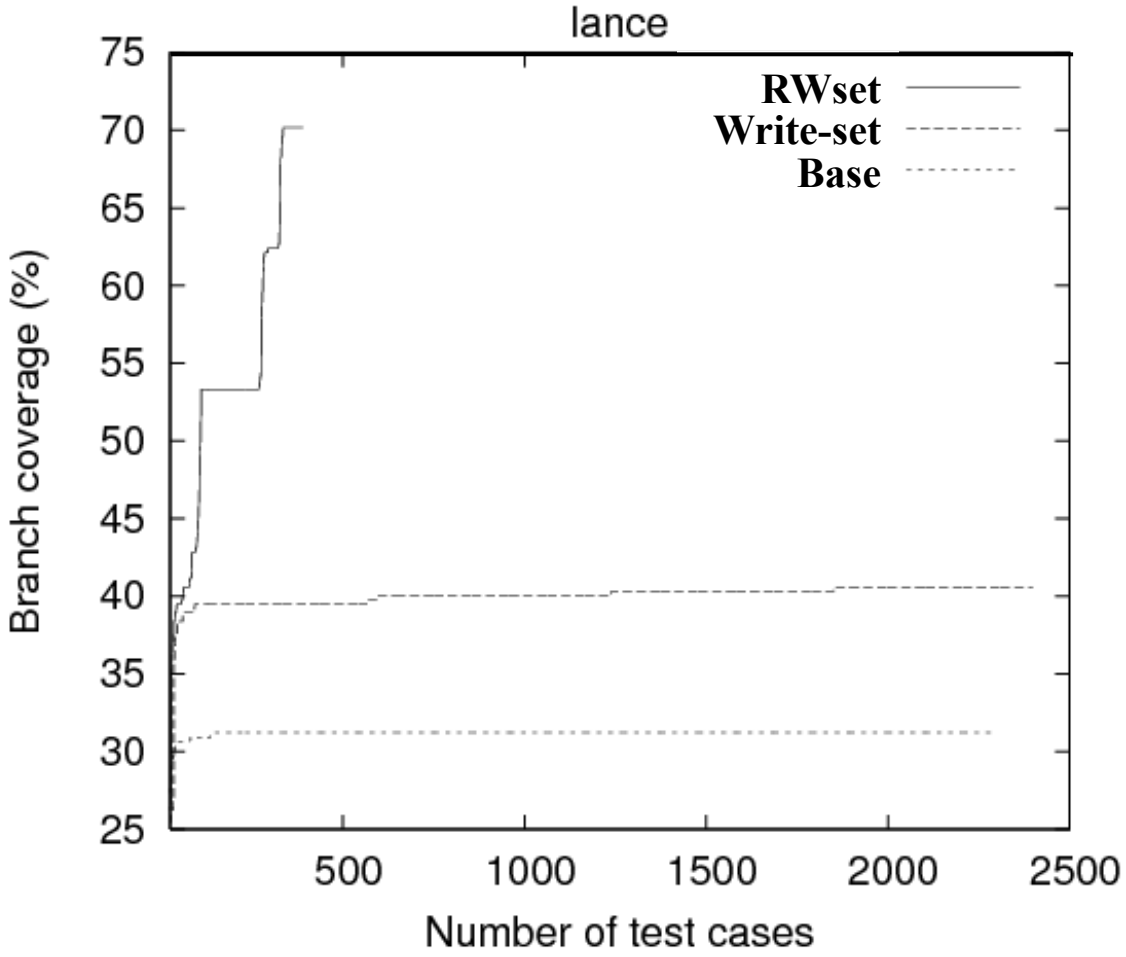
Minix drivers - evaluation

- Three versions of EXE:
  - Base
  - Write-set
  - RWset
- Fixed the # of iterations in each version
  - mainly to have the base version terminate
- Ran each version for an hour
  - recorded branch coverage + non-redundant states

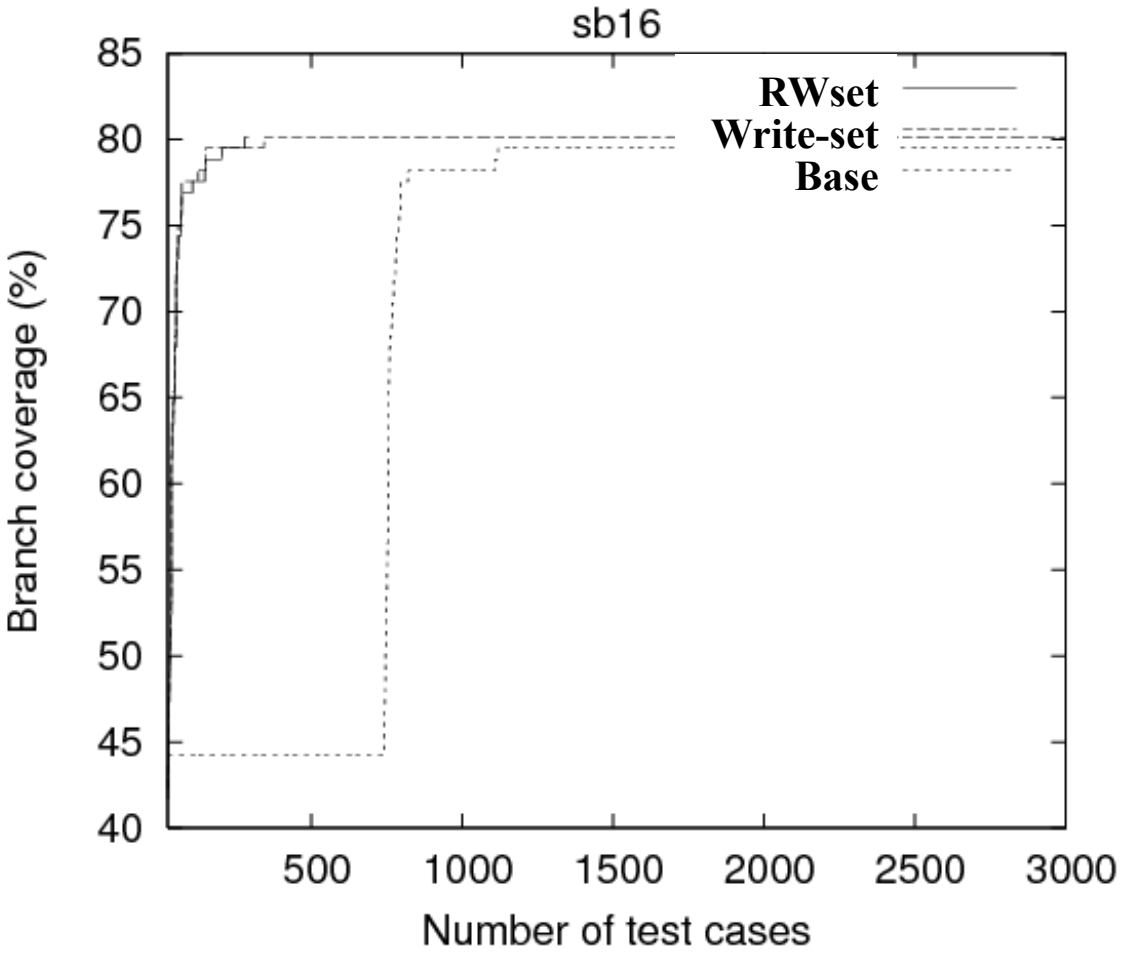
Branch coverage (pci)



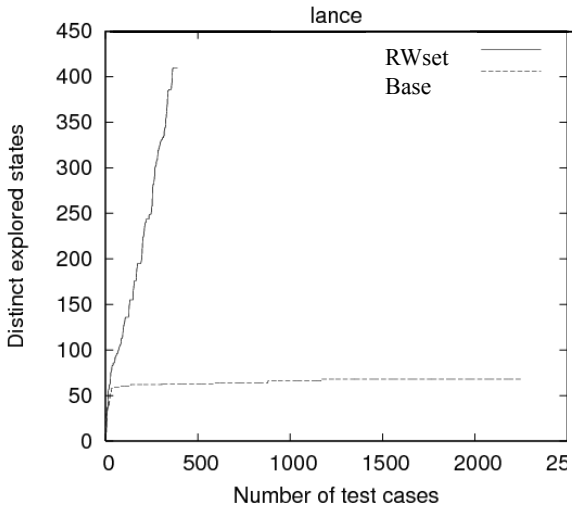
Branch coverage (lance)

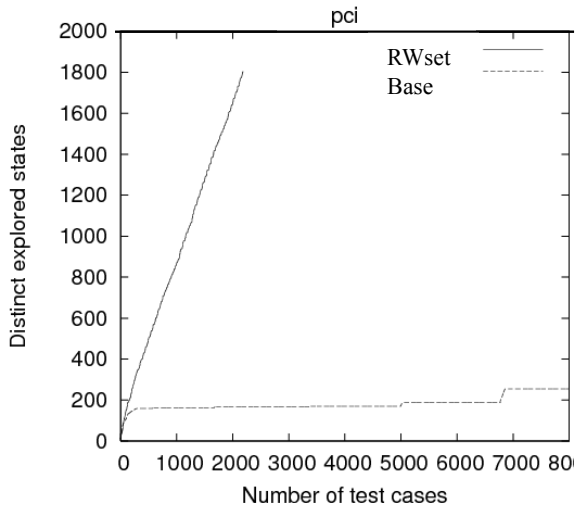


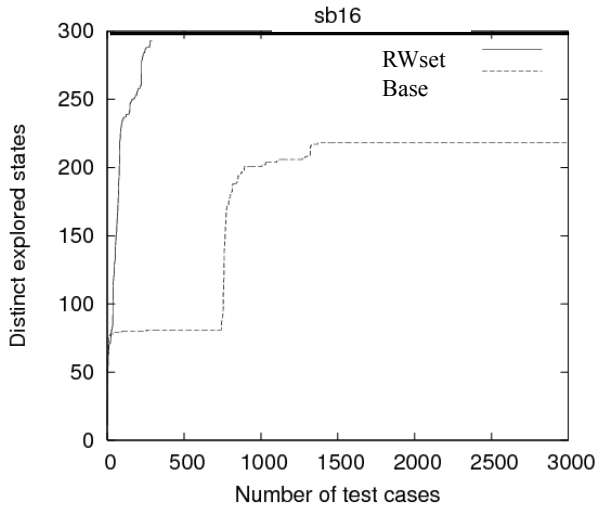
Branch coverage (sb16)



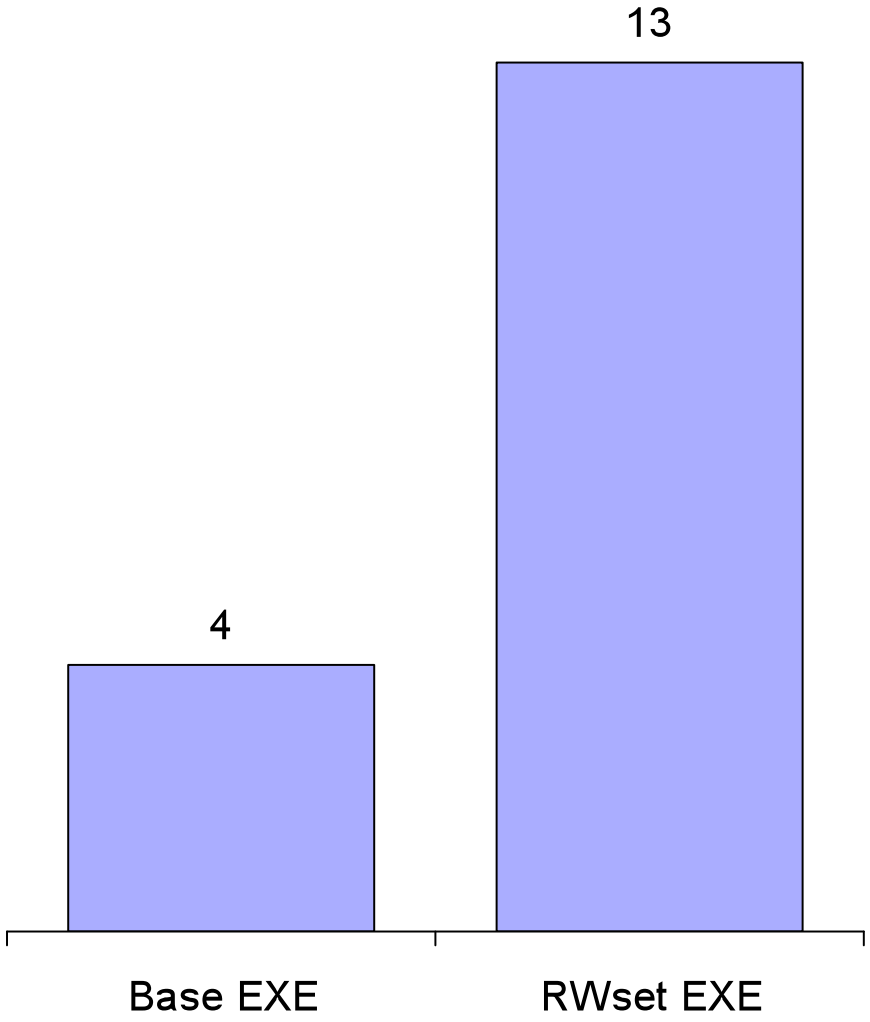
Non-redundant states







Bugs in device drivers



Summary

- RWset analysis
  - efficient way to prune large numbers of redundant paths
  - only values observed by the program trigger the execution of new paths
- Evaluated on Minix drivers and medium size applications
  - big reduction in the number of paths explored


LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

Questions?