unstrip: Restoring Function Information to Stripped Binaries Using Dyninst

Emily Jacobson and Nathan Rosenblum

Paradyn Project

Paradyn / Dyninst Week
Madison, Wisconsin
May 2-4, 2011

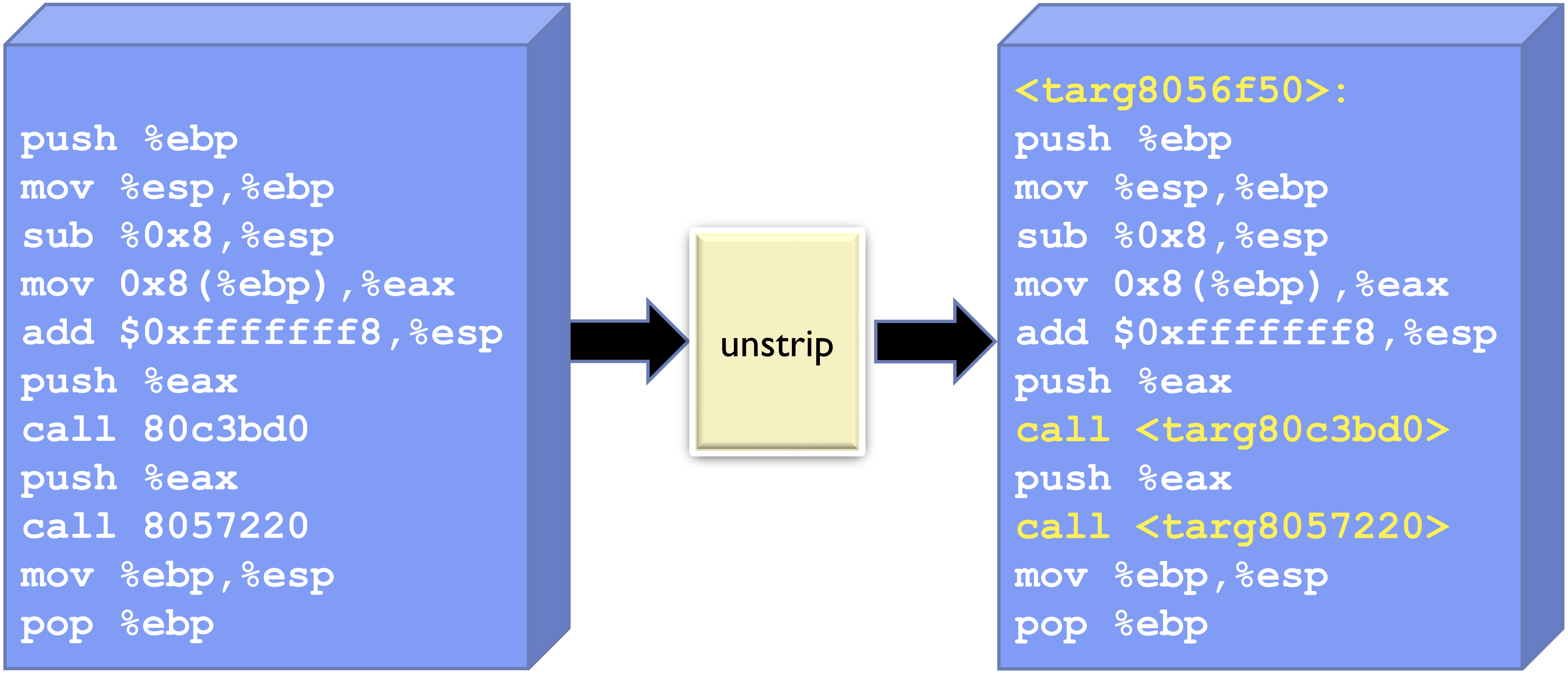
Binary Tools Need Symbol Tables

- Debugging Tools
  - GDB, IDA Pro…
- Instrumentation Tools
  - PIN, Dyninst,…
- Static Analysis Tools
  - CodeSurfer/x86,…
- Security Analysis Tools
  - IDA Pro,…

unstrip = stripped parsing

+

binary rewriting



```
push %ebp
mov %esp,%ebp
sub %0x8,%esp
mov 0x8(%ebp),%eax
add $0xfffffff8,%esp
push %eax
call 80c3bd0
push %eax
call 8057220
mov %ebp,%esp
pop %ebp
```

unstrip

```
<targ8056f50>:
push %ebp
mov %esp,%ebp
sub %0x8,%esp
mov 0x8(%ebp),%eax
add $0xfffffff8,%esp
push %eax
call <targ80c3bd0>
push %eax
call <targ8057220>
mov %ebp,%esp
pop %ebp
```

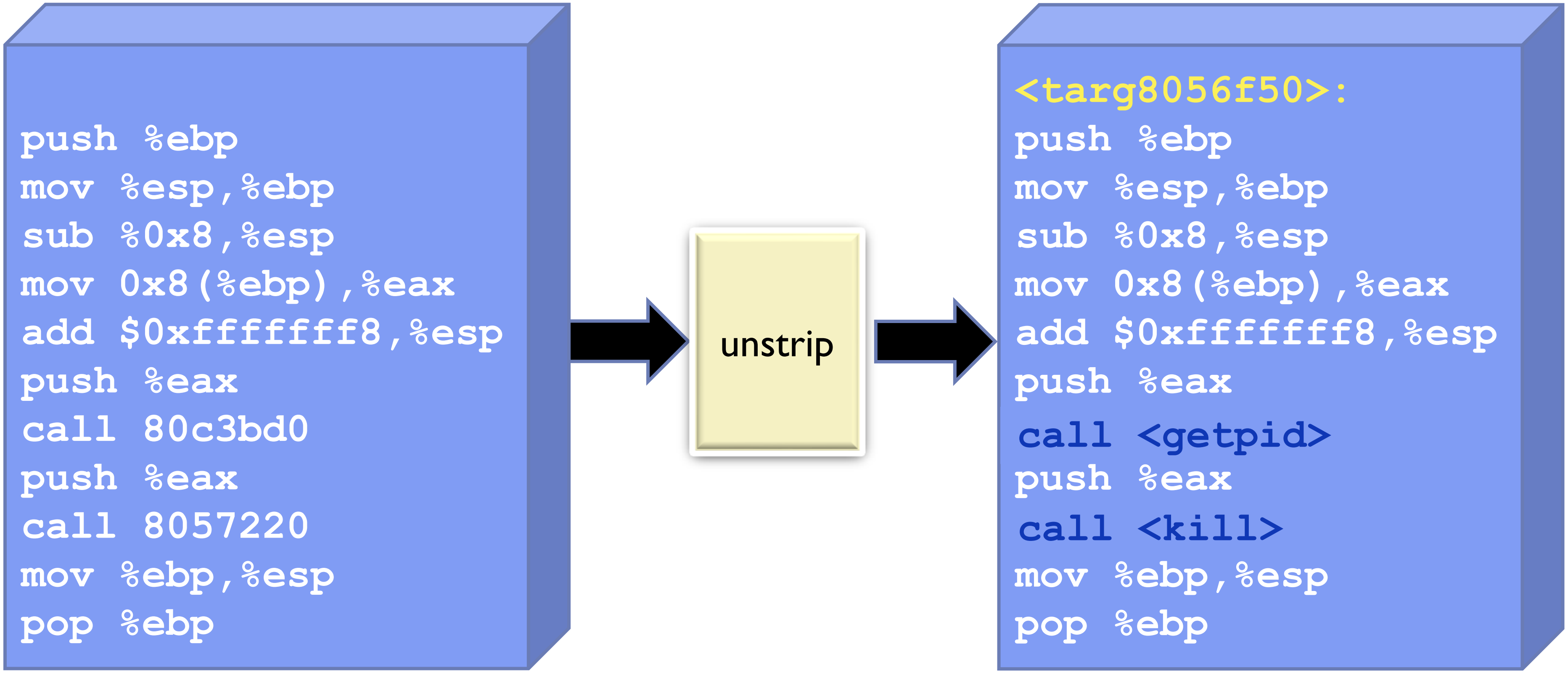
New Semantic Information

- Important semantic information: program's interaction with the operating system (*system calls*)
- These calls are encapsulated in *wrapper functions*

Library fingerprinting: identify functions based on patterns learned from exemplar libraries unstrip = stripped parsing

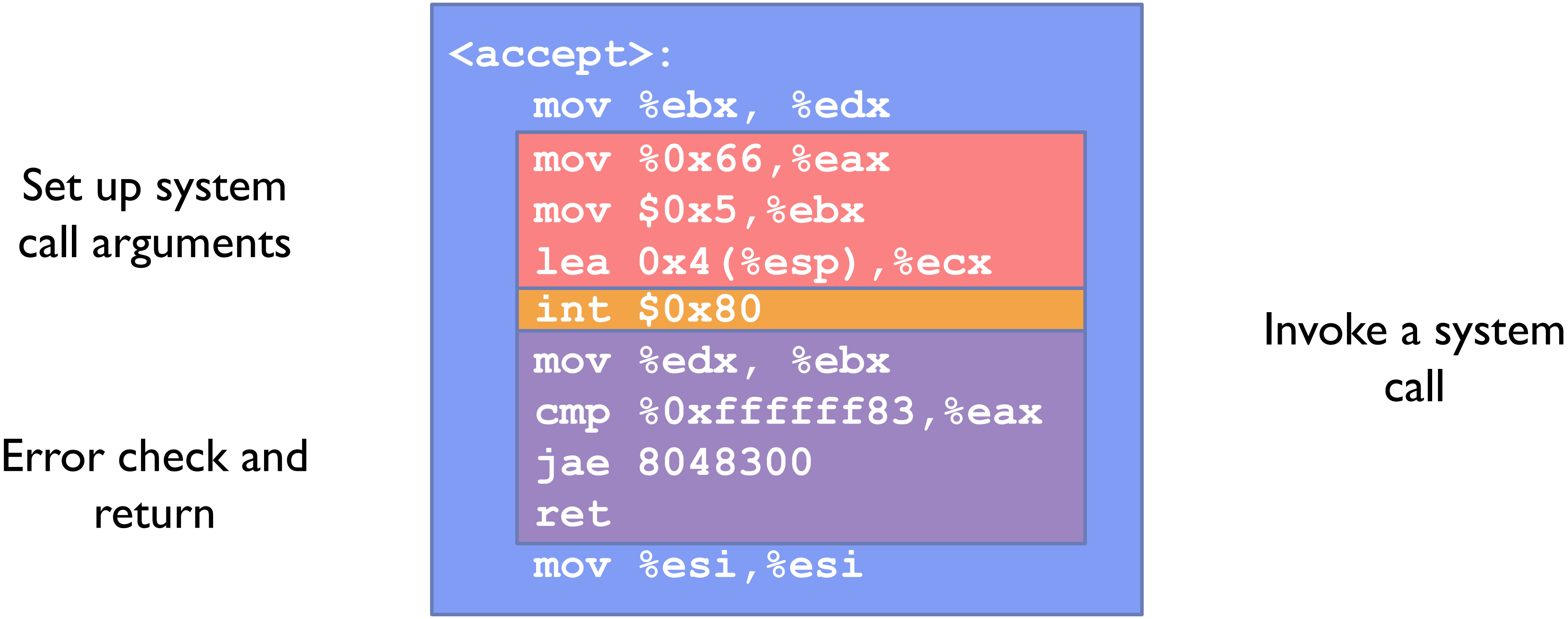
+ library fingerprinting

+ binary rewriting



```
push %ebp
mov %esp,%ebp
sub %0x8,%esp
mov 0x8(%ebp),%eax
add $0xfffffff8,%esp
push %eax
call 80c3bd0
push %eax
call 8057220
mov %ebp,%esp
pop %ebp
```

```
<targ8056f50>:
push %ebp
mov %esp,%ebp
sub %0x8,%esp
mov 0x8(%ebp),%eax
add $0xfffffff8,%esp
push %eax
call <getpid>
push %eax
call <kill>
mov %ebp,%esp
pop %ebp
```



```
<accept>:
    mov %ebx, %edx
    mov %0x66,%eax
    mov $0x5,%ebx
    lea 0x4(%esp),%ecx
    int $0x80
    mov %edx, %ebx
    cmp %0xffffff83,%eax
    jae 8048300
    ret
    mov %esi,%esi
```

Set up system call arguments

Invoke a system call

Error check and return

```
<accept>:
    mov %ebx, %edx
    mov %0x66,%eax
    mov $0x5,%ebx
    lea 0x4(%esp),%ecx
    int $0x80
    mov %edx, %ebx
    cmp %0xffffff83,%eax
    jae 8048300
    ret
    mov %esi,%esi
```

glibc 2.2.4 on RHEL

```
<accept>:
    cmpl $0x0,%gs:0xc
    jne 80f669c
    mov %ebx, %edx
    mov %0x66,%eax
    mov $0x5,%ebx
    lea 0x4(%esp),%ecx
    int $0x80
    mov %edx, %ebx
    cmp %0xffffff83,%eax
    jae 8048460
    ret
    push %esi
    call
    libc_enable_asyncancel
    mov %eax,%esi
    mov %ebx,%edx
    mov $0x66,%eax
    mov $0x5,%ebx
    lea 0x8(%esp),%ecx
    int $0x80
    mov %edx, %ebx
    xchg %eax,%esi
    call
    libc_disable_acynancel
    mov %esi,%eax
    pop %esi
    cmp $0xffffff83,%eax
    jae syscall_error
    ret
```

glibc 2.5 on RHEL with GCC 4.1.2

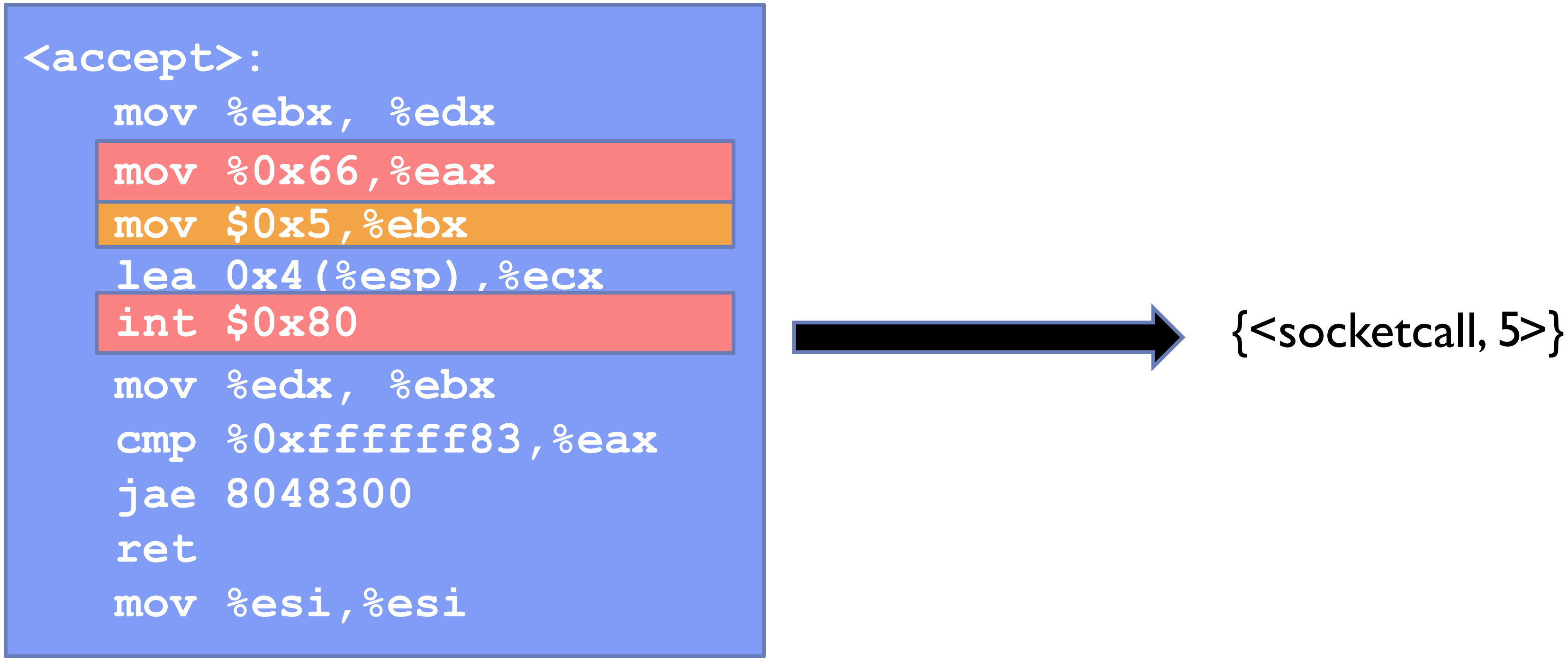
```
<accept>:
    cmpl $0x0,%gs:0xc
    jne 80f669c
    mov %ebx, %edx
    mov %0x66,%eax
    mov $0x5,%ebx
    lea 0x4(%esp),%ecx
    call *0x814e93c
    mov %edx, %ebx
    cmp %0xffffff83,%eax
    jae 8048460
    ret
    push %esi
    call
    libc_enable_asyncancel
    mov %eax,%esi
    mov %ebx,%edx
    mov $0x66,%eax
    mov $0x5,%ebx
    lea 0x8(%esp),%ecx
    call *0x8181578
    mov %edx, %ebx
    xchg %eax,%esi
    call
    libc_disable_acynancel
    mov %esi,%eax
    pop %esi
    cmp $0xffffff83,%eax
    jae syscall_error
    ret
```

glibc 2.5 on RHEL with GCC 3.4.4

The same function can be realized in a variety of ways in the binary

Semantic Descriptors

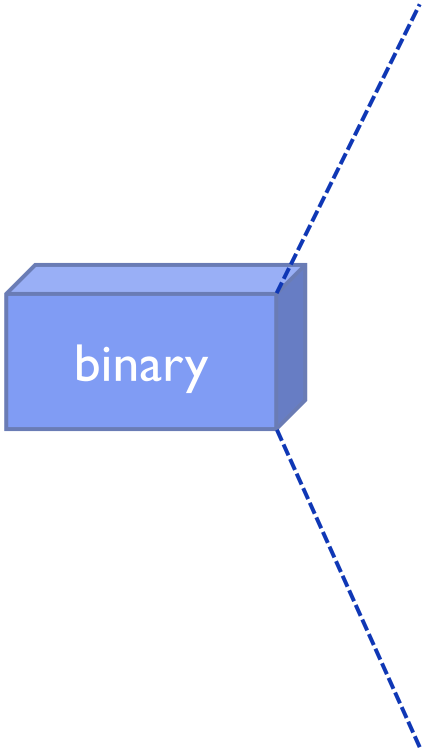
- Instead, we'll take a semantic approach
- Record information that is likely to be invariant across multiple versions of the function



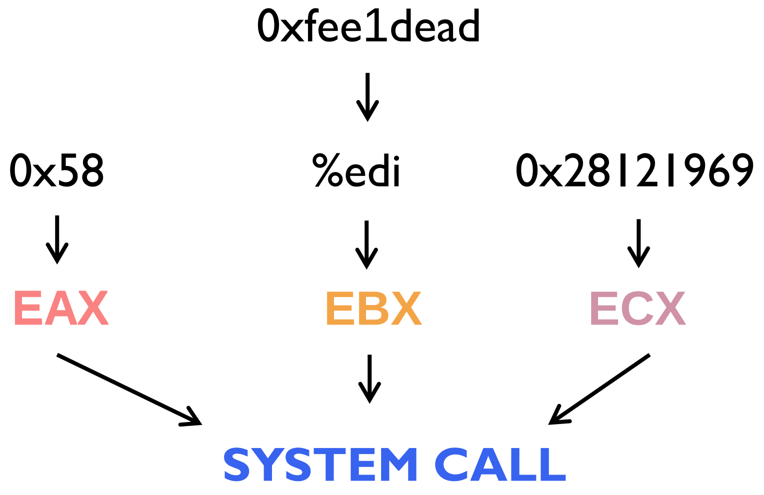
```
<accept>:
    mov %ebx, %edx
    mov %0x66,%eax
    mov $0x5,%ebx
    lea 0x4(%esp),%ecx
    int $0x80
    mov %edx, %ebx
    cmp %0xffffff83,%eax
    jae 8048300
    ret
    mov %esi,%esi
```

{<socketcall, 5>}

Building Semantic Descriptors



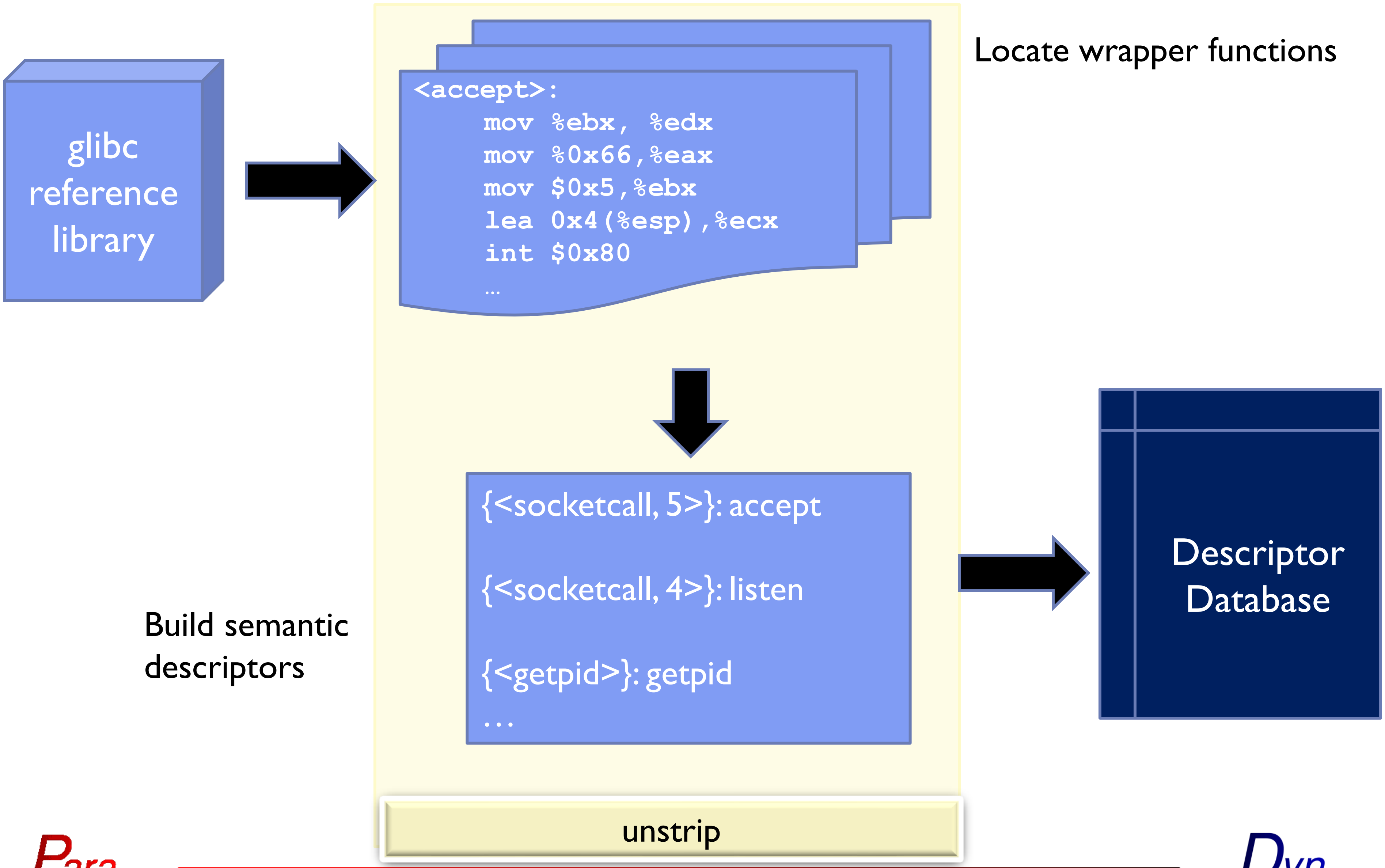
```
reboot:
    push %ebp
    mov %esp,%ebp
    sub $0x10,%esp
    push %edi
    push %ebx
    mov 0x8(%ebp),%edx
    mov $0xfee1dead,%edi
    mov $0x28121969,%ecx
    push %ebx
    mov %edi,%ebx
    mov $0x58,%eax
    int $0x80
    …
```



{<reboot, 0xfee1dead, 0x2812969>}

We parse an input binary, locate system calls and wrapper function calls, and employ dataflow analysis.

Building a Descriptor Database



```
<accept>:
      mov %ebx, %edx
      mov %0x66,%eax
      mov $0x5,%ebx
      lea 0x4(%esp),%ecx
      int $0x80
      …
```

Locate wrapper functions

{<socketcall, 5>}: accept

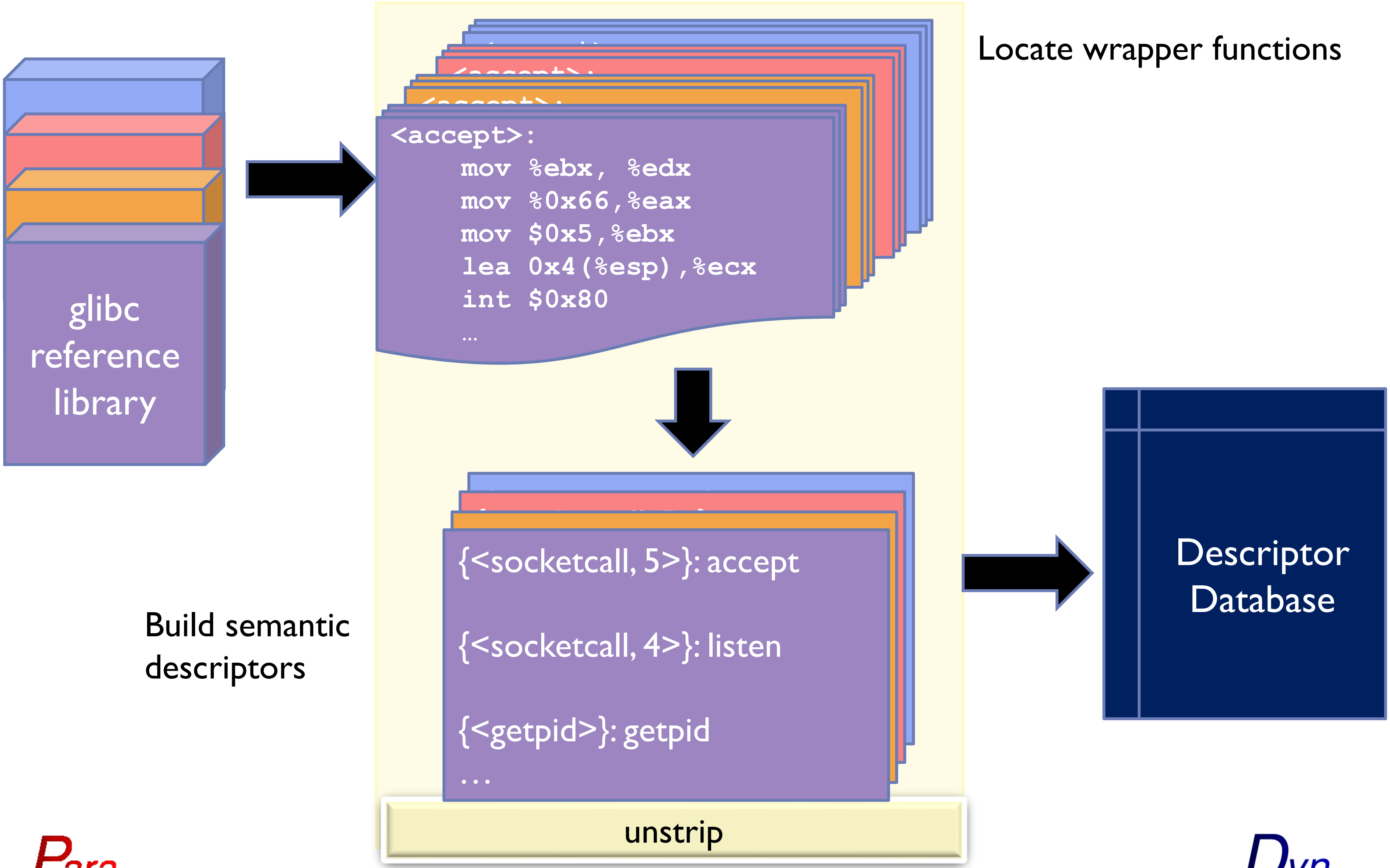
{<socketcall, 4>}: listen

{<getpid>}: getpid

…

Build semantic descriptors

Building a Descriptor Database



Locate wrapper functions

```
<accept>:
     mov %ebx, %edx
     mov %0x66,%eax
     mov $0x5,%ebx
     lea 0x4(%esp),%ecx
     int $0x80
     …
```

Build semantic descriptors

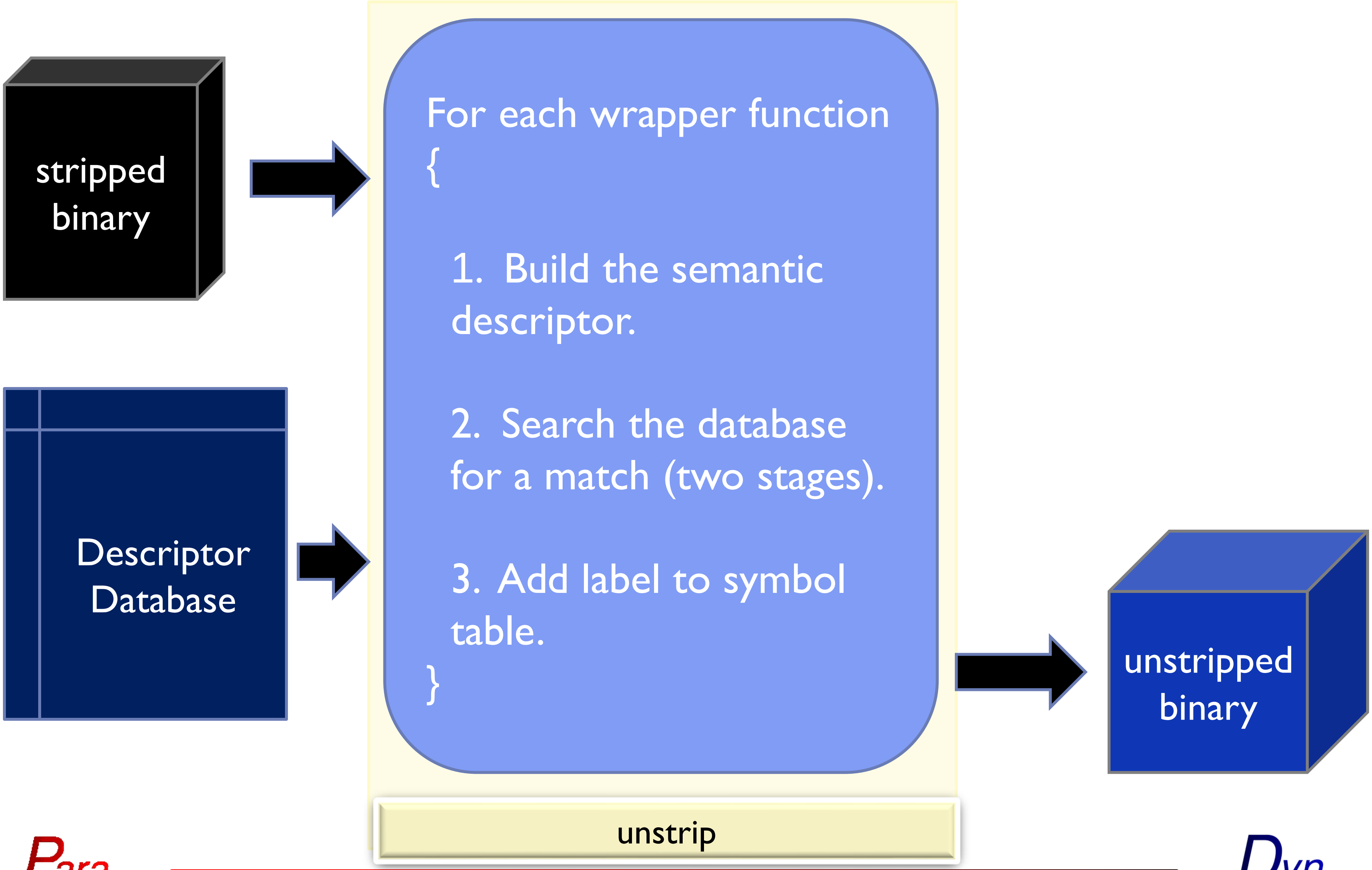
{<socketcall, 5>}: accept

{<socketcall, 4>}: listen

{<getpid>}: getpid

…

Identifying Functions in a Stripped Binary

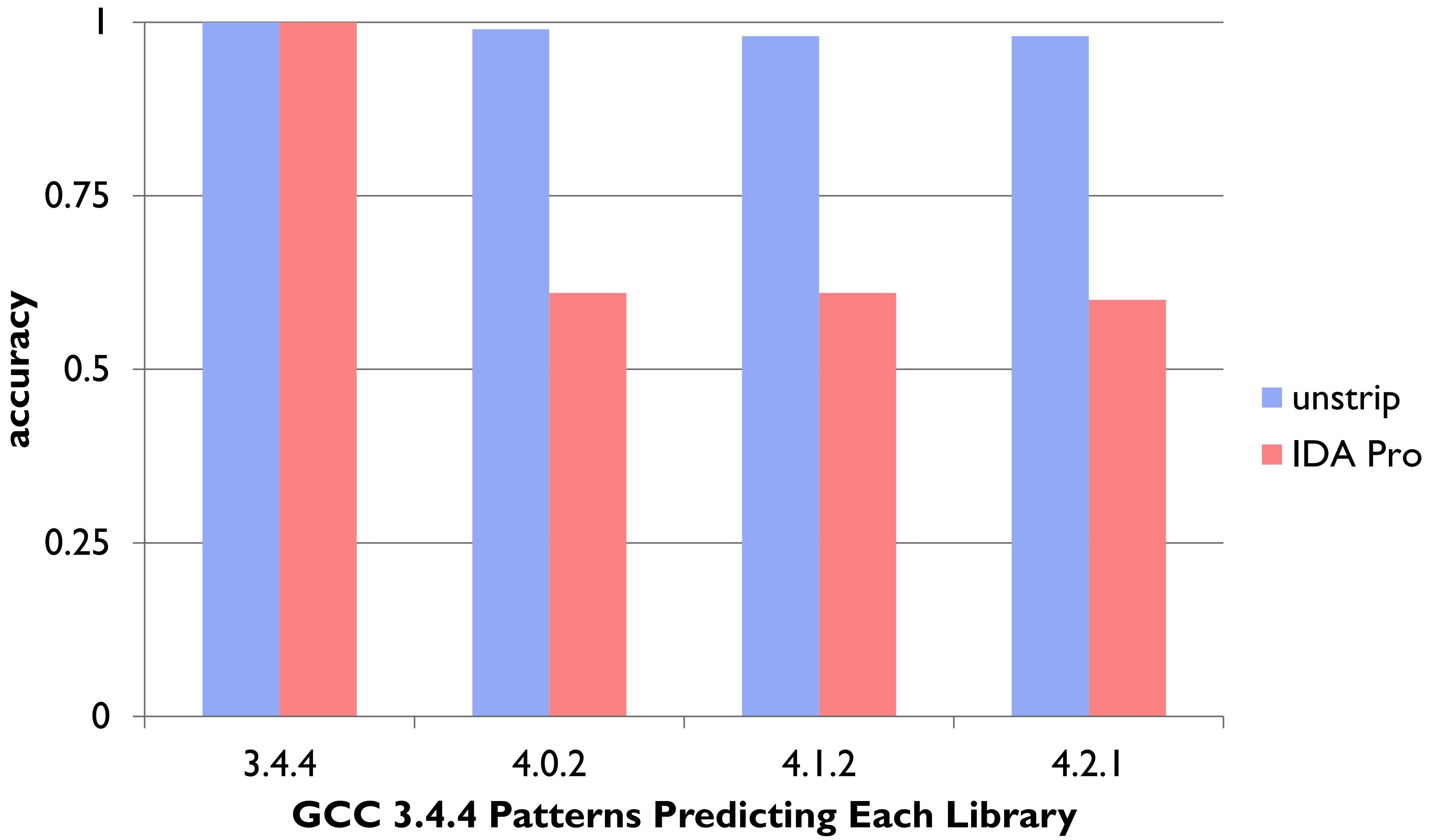


```
For each wrapper function
{
  1. Build the semantic descriptor.

  2. Search the database for a match (two stages).

  3. Add label to symbol table.
}
```

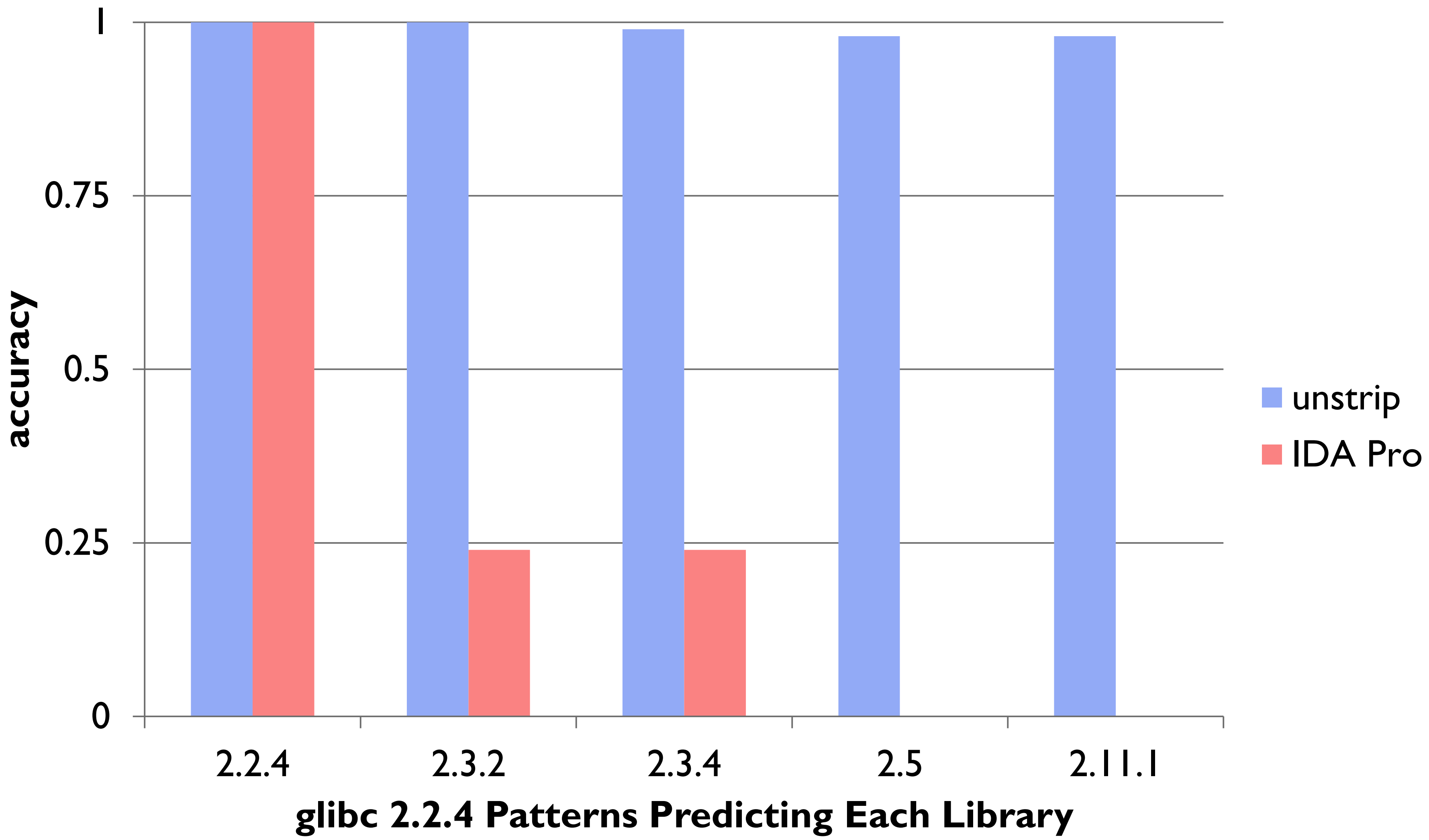
Evaluation

- To evaluate across three dimensions of variation, we constructed three data sets:
  - compiler version
  - library version
  - distribution vendor
- In each set, compile statically-linked binaries, build a DBB, compare unstrip to IDA Pro's FLIRT
- Evaluation measure is accuracy

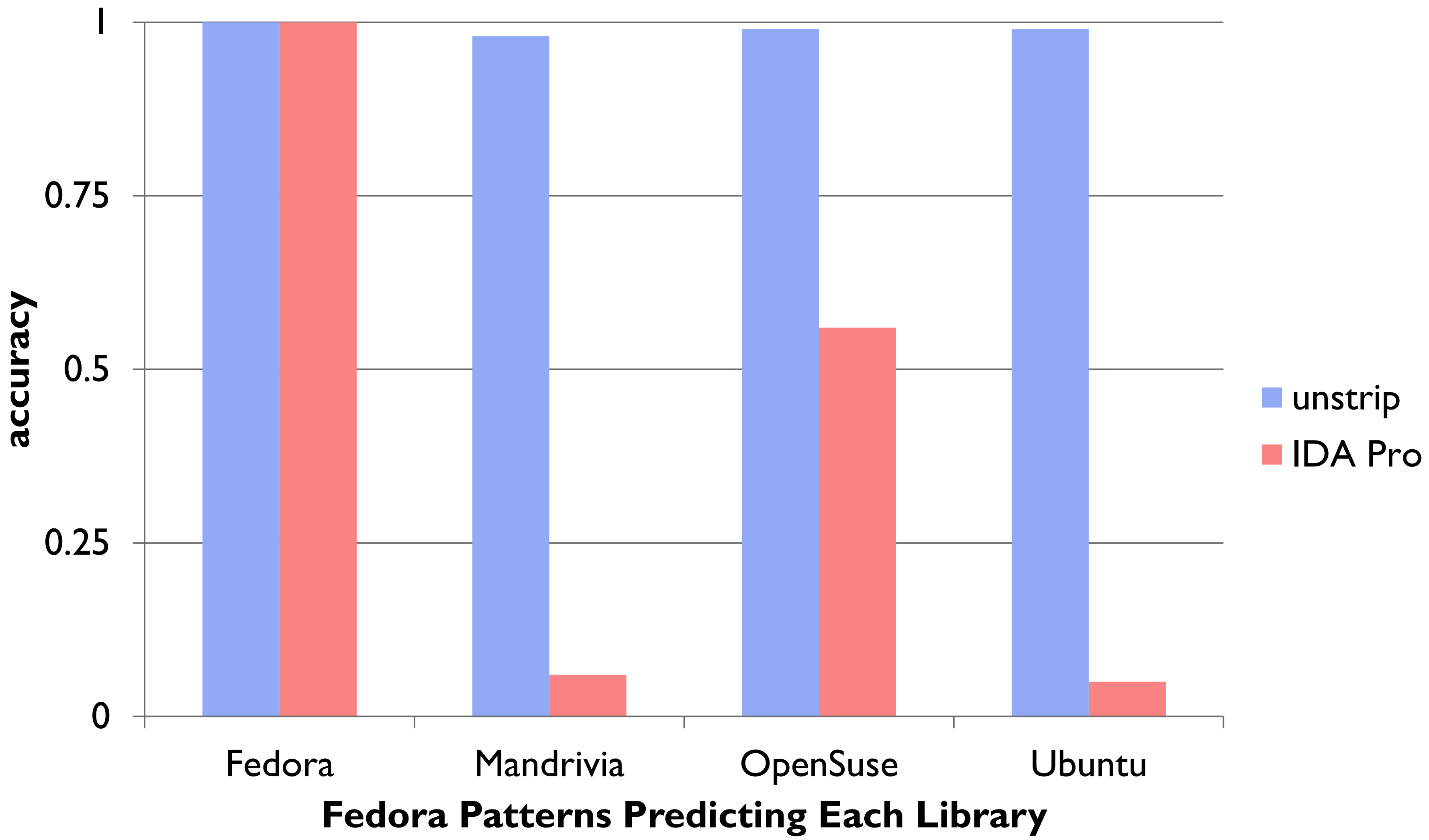
Evaluation Results: Compiler Version Study



Evaluation Results: Library Version Study



Evaluation Results: Distribution Study



For full details, tech report available online at:

ftp://ftp.cs.wisc.edu/paradyn/papers/Jacobson11Unstrip.pdf

unstrip is available at:

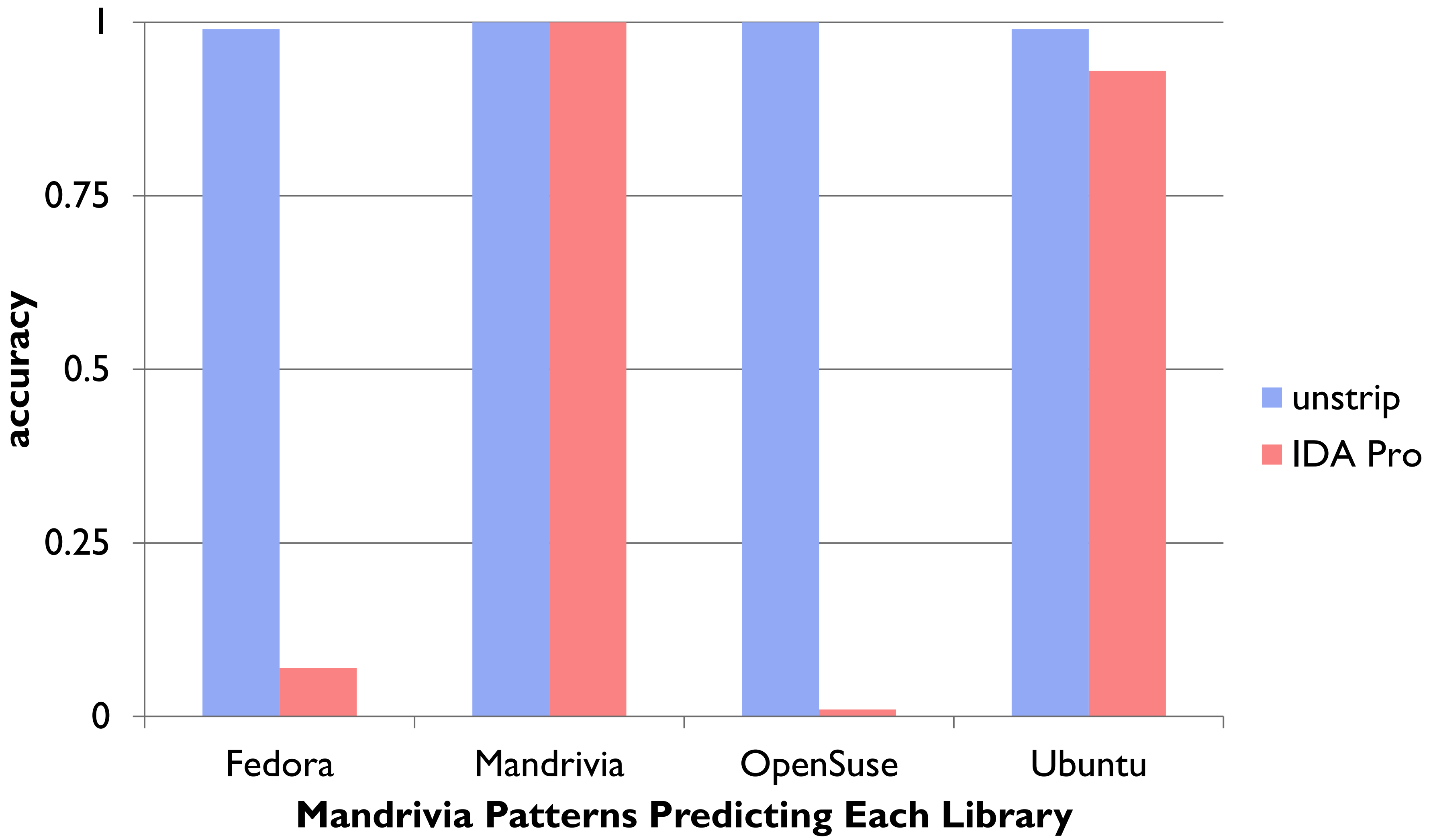
http://www.paradyn.org/html/tools/unstrip.html

Come see the unstrip demo today at
2:00 or 2:30 (in 1260 WID/MIR)

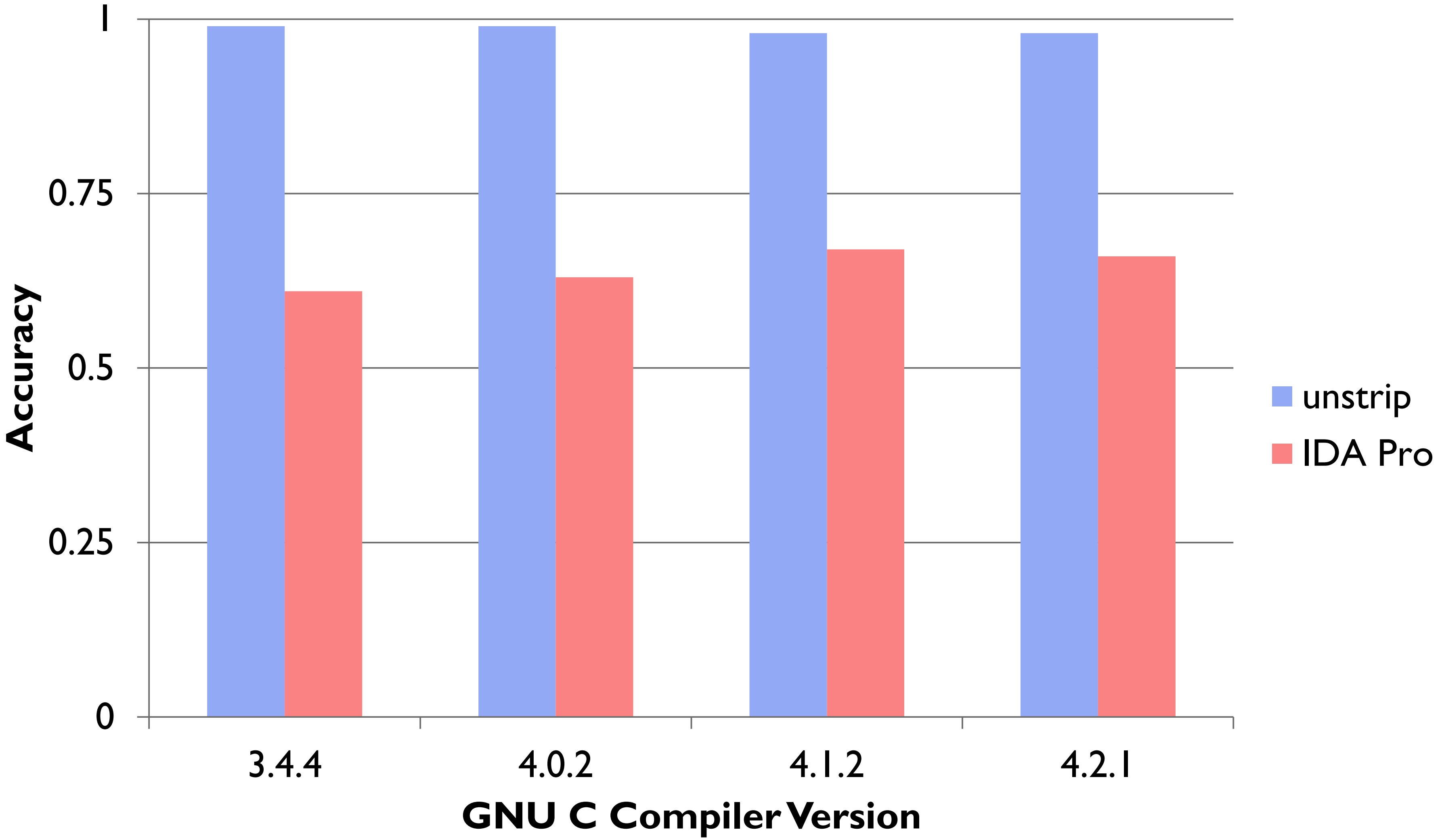
Extra Slides

- Some additional results

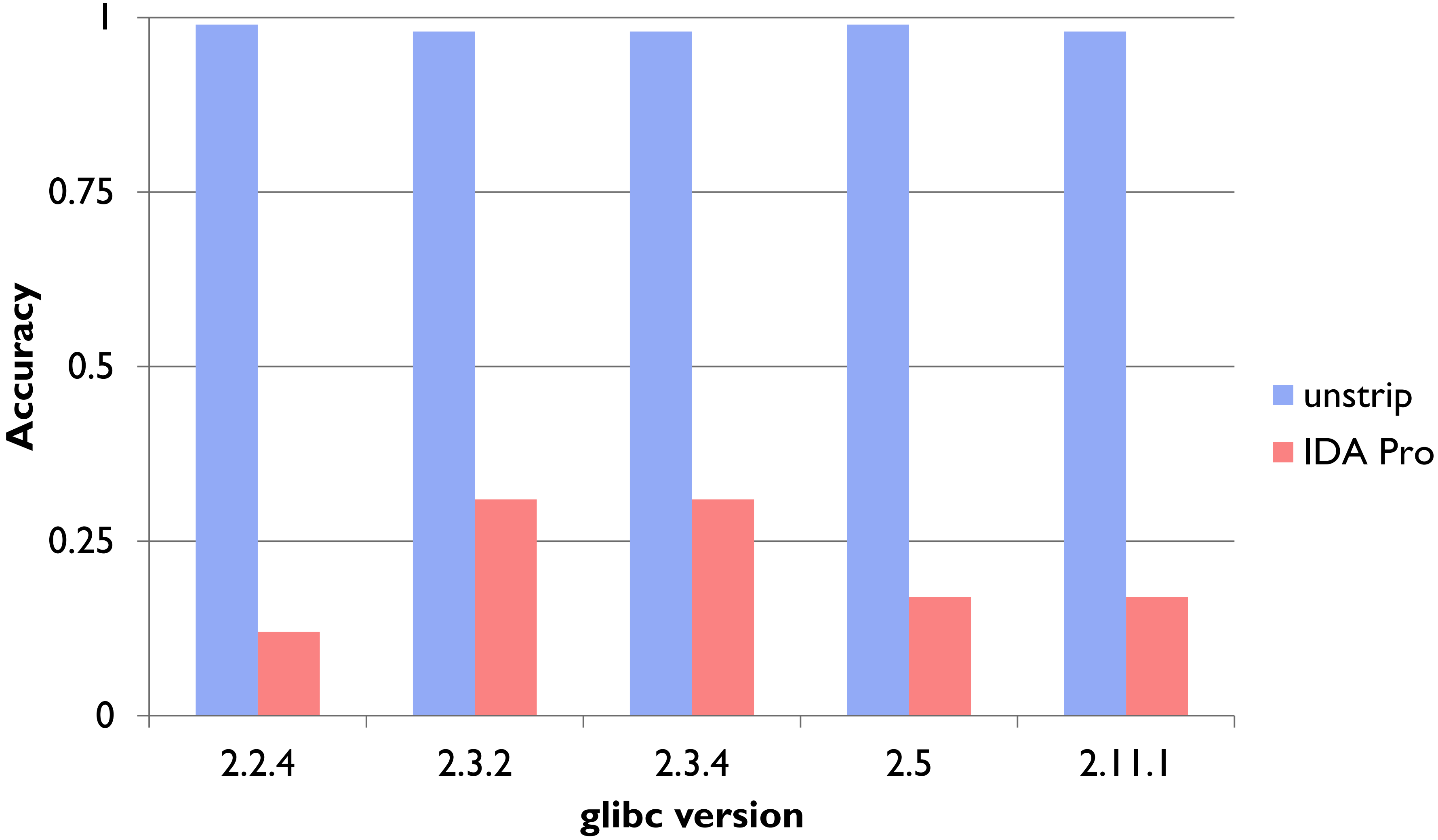
Evaluation Results: Distribution Study



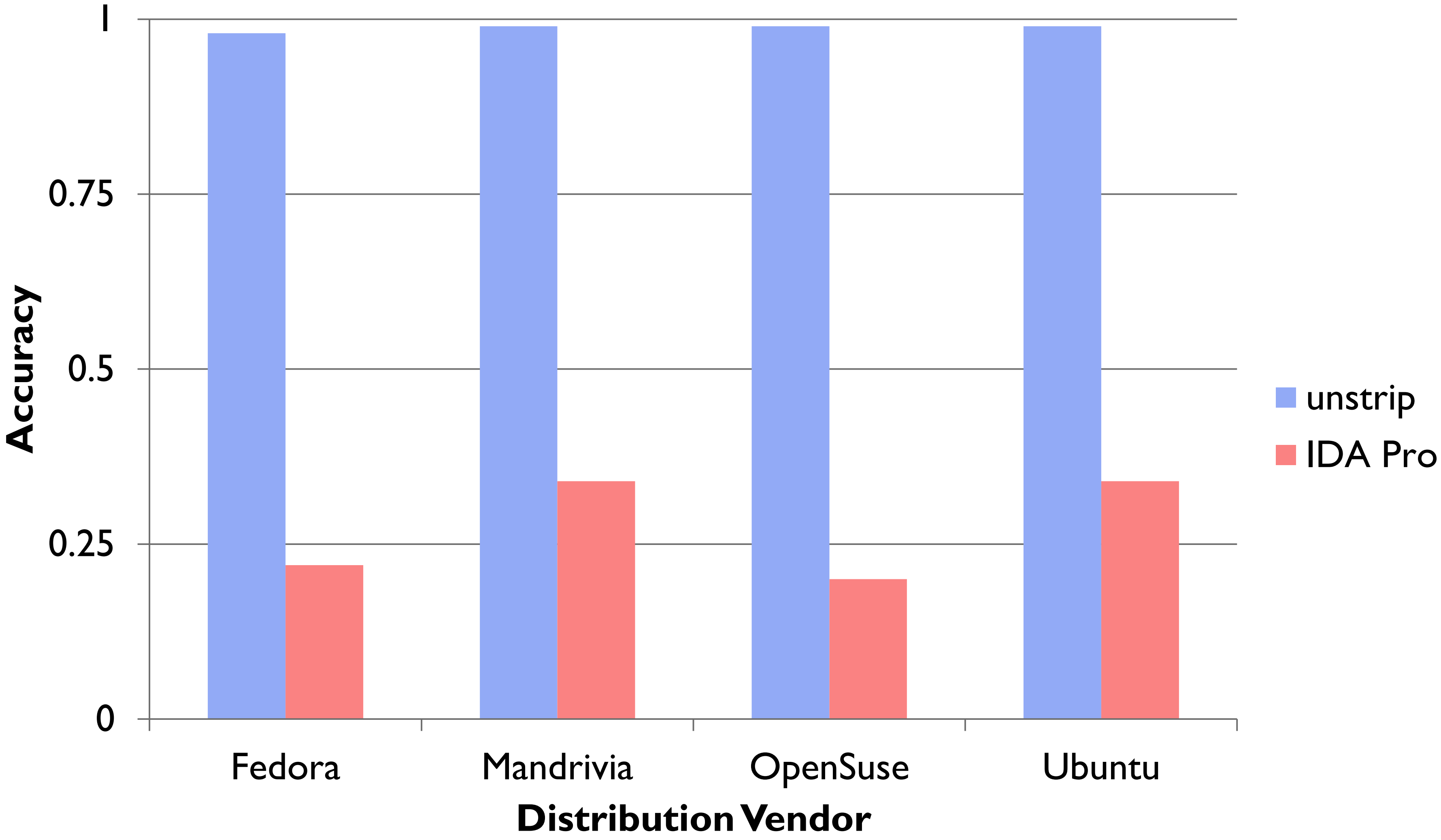
Evaluation Results: Toolchain Study (one predicts the rest)



Evaluation Results: Library Version Study (one predicts the rest)



Evaluation Results: Distribution Study (one predicts the rest)


Accuracy
Fedora
Mandrivia
OpenSuse
Ubuntu
Distribution Vendor
unstrip
IDA Pro