Dyn
inst

DyninstAPI

A Binary Instrumentation and Analysis Tool

University of Maryland 1856

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Challenges

Technologies

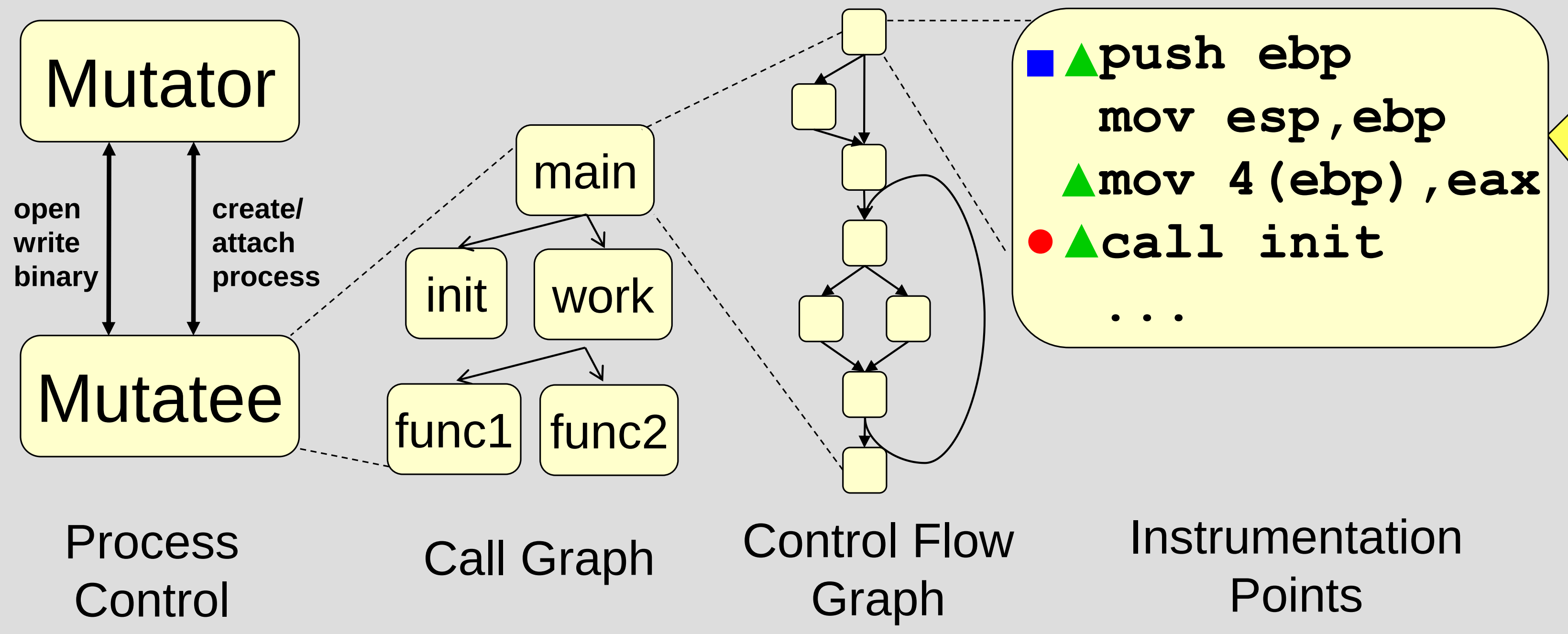
Interface

Complexity of binaries hidden from users.

Platform independent interface.

Mutator
(User Application)

- **Mutatee Control**
  Create/attach to a process
  Open and write to a binary
- **Binary Analysis**
  Present a model of a binary
- **Instrumentation**
  Specify what to insert where



```
■▲push ebp
   mov esp,ebp
  ▲mov 4(ebp),eax
 ●▲call init
   ...
```

Platform Independent Abstractions:
•Call Graph
•Control Flow Graph
•Instrumentation Points
- Memory instructions (▲)
- Function entry/exit (■)
- Call sites (●)
- Loops
- Arbitrary Instructions

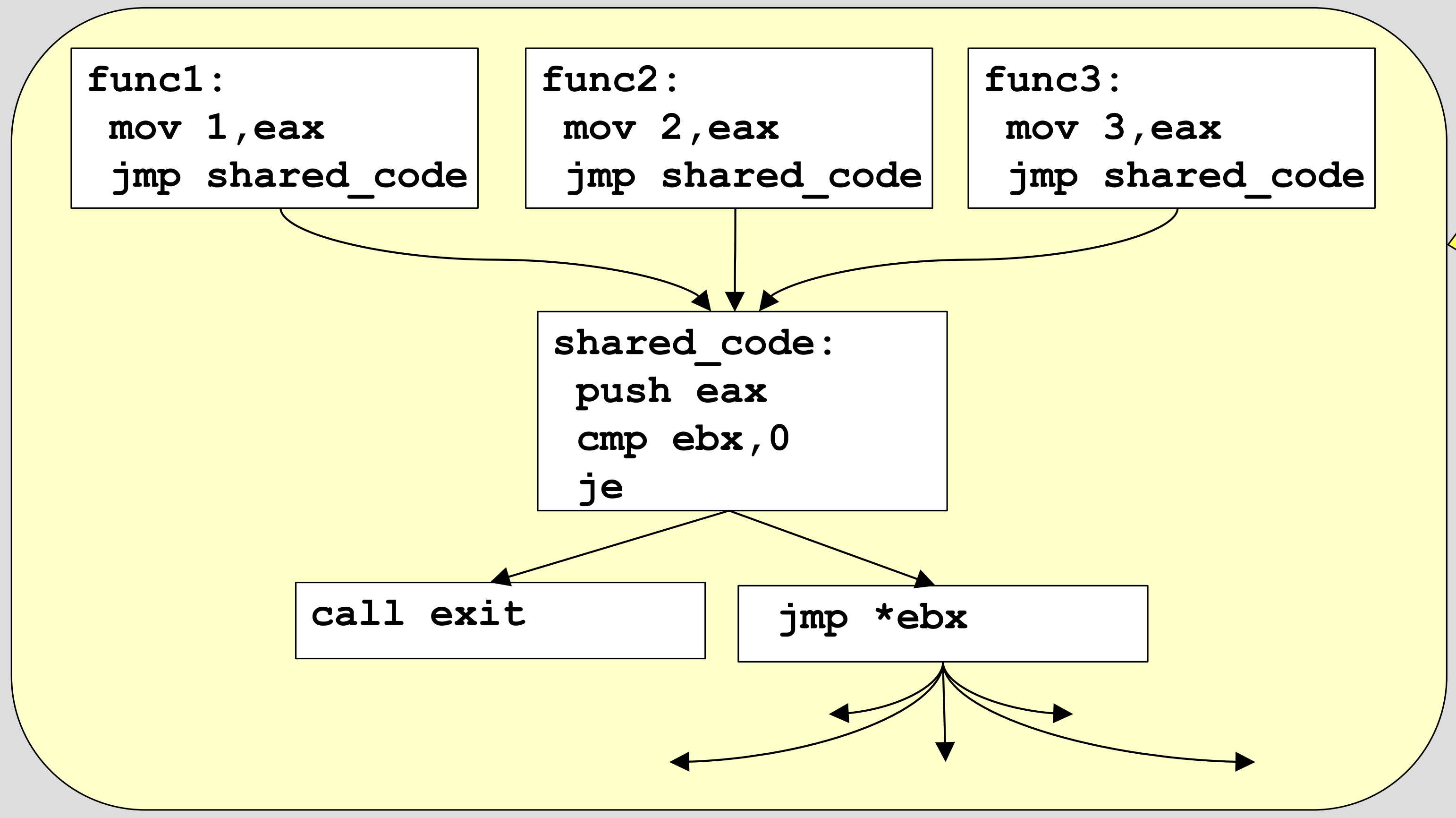
Binary Analysis

Stripped binaries lack symbols, debug information, or linker relocations.

Complex binaries produced by modern compilers:
- Share code between functions
- Interleave code and data
- Frequently use indirect control transfers
- Overlap instruction sequences
- Non-returning function calls

Mutatee
(App Being Instrumented)

```
1000: mov 1,eax
1005: jmp 101e
100a: mov 2,eax
100f: jmp 101e
1014: mov 3,eax
1019: jmp 101e
101e: push eax
101f: cmp ebx,0
1025: je 1030
102a: jmp *ebx
102c: 00 00 00 01
1030: call exit
1035: nop
```



```
func1:
 mov 1,eax
 jmp shared_code

func2:
 mov 2,eax
 jmp shared_code

func3:
 mov 3,eax
 jmp shared_code

shared_code:
 push eax
 cmp ebx,0
 je

call exit

jmp *ebx
```

Breadth first control flow traversal:
•Discovers code by parsing from known entry points.
•Opportunistically uses symbols, if available.
•Identifies code shared between functions, overlapping instructions, non-returning functions.

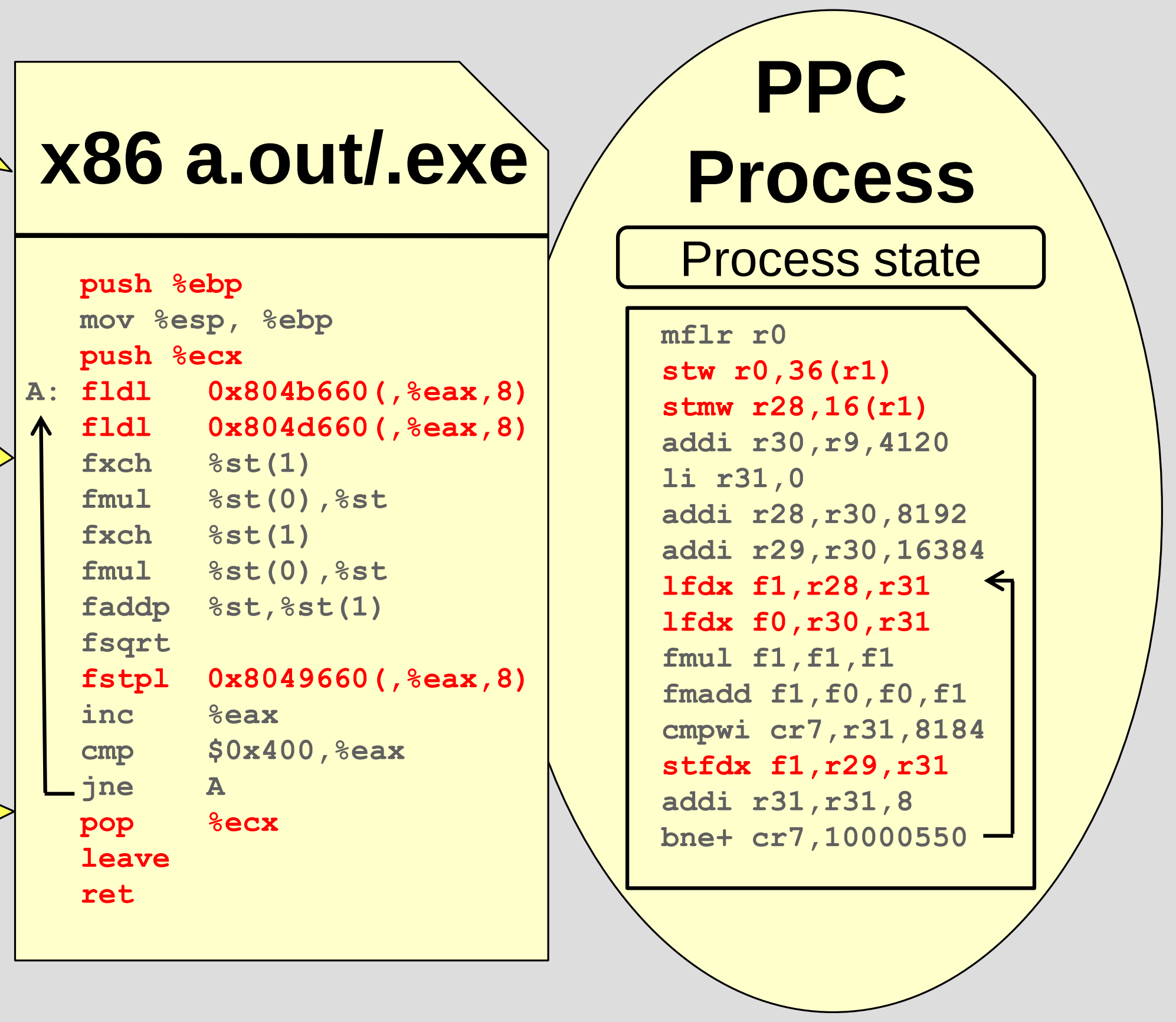
Gap parsing identifies functions that were targets of only indirect control flow in unparsed bytes.

Binary Modification

Cross-platform support across architectures and OSs.

Static instrumentation on binaries or dynamic instrumentation on processes.

Low instrumentation overhead and perturbation.



x86 a.out/.exe

```
   push %ebp
   mov %esp, %ebp
   push %ecx
A: fldl    0x804b660(,%eax,8)
   fldl    0x804d660(,%eax,8)
   fxch    %st(1)
   fmul    %st(0),%st
   fxch    %st(1)
   fmul    %st(0),%st
   faddp   %st,%st(1)
   fsqrt
   fstpl   0x8049660(,%eax,8)
   inc     %eax
   cmp     $0x400,%eax
   jne     A
   pop     %ecx
   leave
   ret
```

PPC Process

Process state

```
mflr r0
stw r0,36(r1)
stmw r28,16(r1)
addi r30,r9,4120
li r31,0
addi r28,r30,8192
addi r29,r30,16384
lfdx f1,r28,r31
lfdx f0,r30,r31
fmul f1,f1,f1
fmadd f1,f0,f0,f1
cmpwi cr7,r31,8184
stfdx f1,r29,r31
addi r31,r31,8
bne+ cr7,10000550
```

Red = instrumented memory instructions

Original Code

```
foo:
 push $1
 call bar
 ret
```

Instrumented Code

```
foo:
 ~~push $1~~ jmp instr_foo
 call bar
 ret

instr_foo:
 instrumentation
 push $1
 call bar
 instrumentation
 ret
```

Trampolines have overhead of only a single jump and do not perturb uninstrumented parts of application.

Create an instrumented binary.
-or-
Write instrumentation to a running process.

Available on Linux-x86/x86_64/ppc32/ppc64, Windows-x86, FreeBSD, Cray, BlueGene

http://www.paradyn.org