WYSINWYX: What You See Is Not What You eXecute
(Static Analysis of Executables)

Gogul Balakrishnan
NEC Laboratories America

Work done while at UW Madison with J.Lim (UW), T. Reps (UW, GrammaTech), T. Teitelbaum (Cornell, GrammaTech)

The Vision

- A tool for analyzing executables
  - to find security vulnerabilities and bugs
  - to analyze/understand malicious code like viruses/worms
  - to perform code obfuscation and de-obfuscation
  - to perform de-compilation
  - etc.,

- Who cares?
  - CERT, Govt. agencies like NSA, etc
  - Anti-Virus/Anti-Malware companies
  - Even researchers in program analysis/verification should!

Why Executables? (1)

- Source code may not be available
  - Viruses and Worms
  - Commercial-Off-The-Shelf (COTS) components
  - Browser plug-ins
  - Java applets
  - etc.,
- Allows analysis of library code
  - Otherwise, must model library code with stubs
  - In turn, helps analysis of source-code
- The usual suspects . . .

Why Executables? (2)

- Executables can be a better platform for finding security vulnerabilities
  - Many exploits utilize particular quirks of the compiler (e.g., details of memory-layout or register usage)

- Analysis of source code may give incorrect answers!
  - "WYSINWYX": What You See Is Not What You eXecute

- Analysis of source code may be less accurate!
  - Executable reflects actual behaviors that may arise
  - Allows us to take into account platform-specific aspects
    - e.g., order of evaluation of arguments of a function

Example: Minimizing Data Lifetime?

- Windows login process
  - keeps a user's password in the heap
- Should minimize data lifetime by
  - clearing memory after login
  - calling free()
- May not work
  - the compiler might "optimize" away the memory-clearing code ("dead-code" elimination)

```
memset(buffer, '\0', len);
free(buffer);
```

$\Longrightarrow$

```
free(buffer);
```

Puzzle

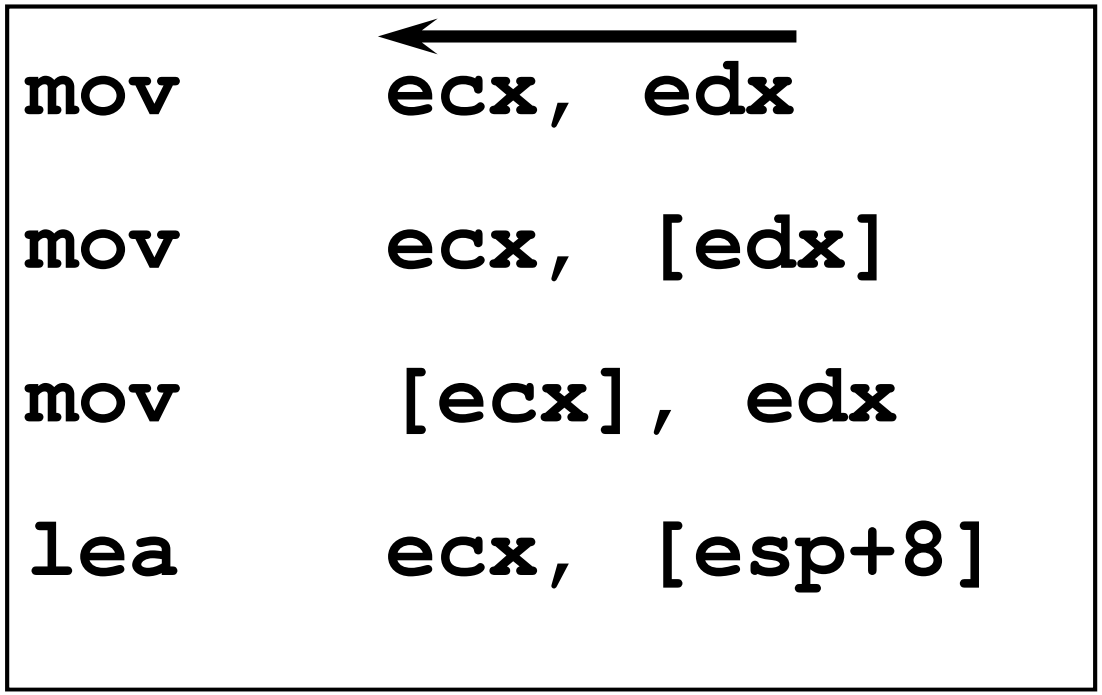
```
int callee(int a, int b) {
  int local;
  if (local == 5) return 1;
  else return 2;
}

int main() {
  int c = 5;
  int d = 7;

  int v = callee(c,d);
  // What is the value of v here?
  return 0;
}
```

Answer: 1
(for the Microsoft compiler)

Tutorial on x86 (Intel Syntax)

```
 p = q;

 p = *q;

*p = q;

 p = &a[2];
```

Tutorial on x86 (Intel Syntax)



```
mov     ecx, edx
mov     ecx, [edx]
mov     [ecx], edx
lea     ecx, [esp+8]
```

```
ecx = edx;
ecx = *edx;
*ecx = edx;
ecx = &a[2];
```

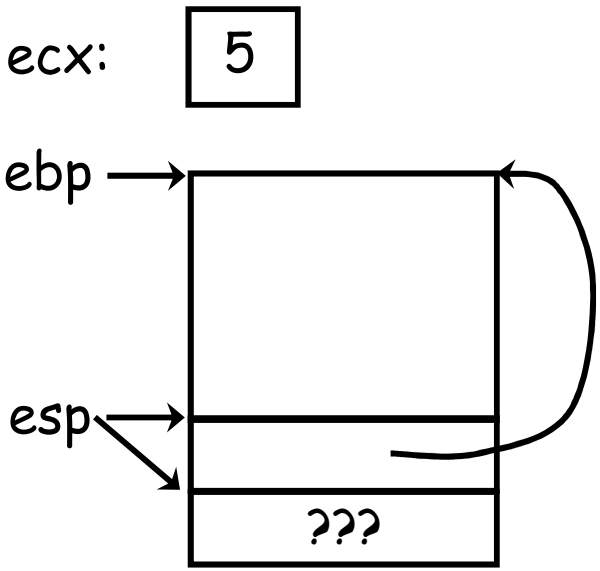
Standard prolog		Prolog for 1 local	
push	ebp	push	ebp
mov	ebp, esp	mov	ebp, esp
sub	esp, 4	push	ecx

```
int callee(int a, int b) {
  int local;
  if (local == 5) return 1;
  else return 2;
}

int main() {
  int c = 5;
  int d = 7;

  int v = callee(c,d);
  // What is the value of v here?
  return 0;
}
```

Answer: 1
(for the Microsoft compiler)

Standard prolog

```
push    ebp
mov     ebp, esp
sub     esp, 4
```

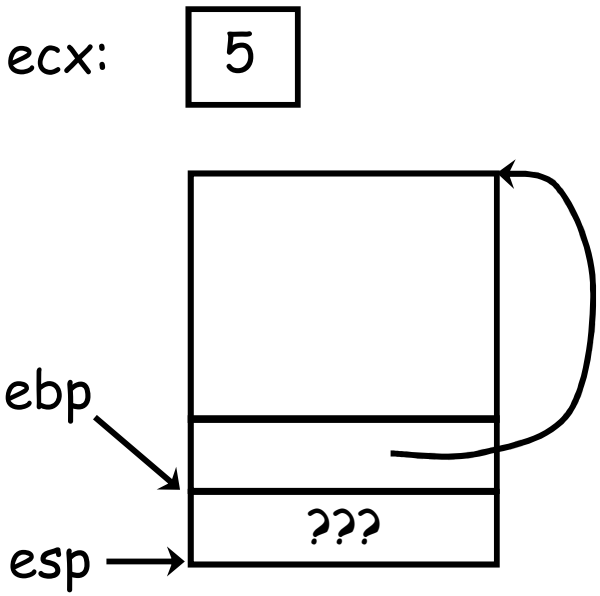


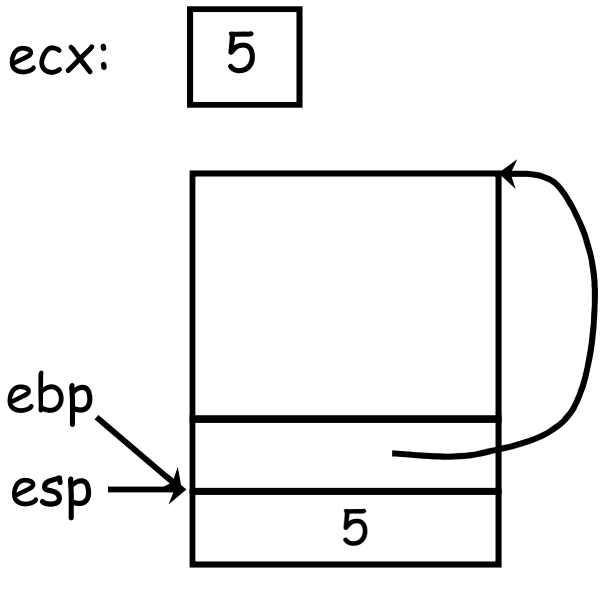
Standard prolog

```
push    ebp
mov     ebp, esp
sub     esp, 4
```

Prolog for 1 local

```
push    ebp
mov     ebp, esp
push    ecx
```





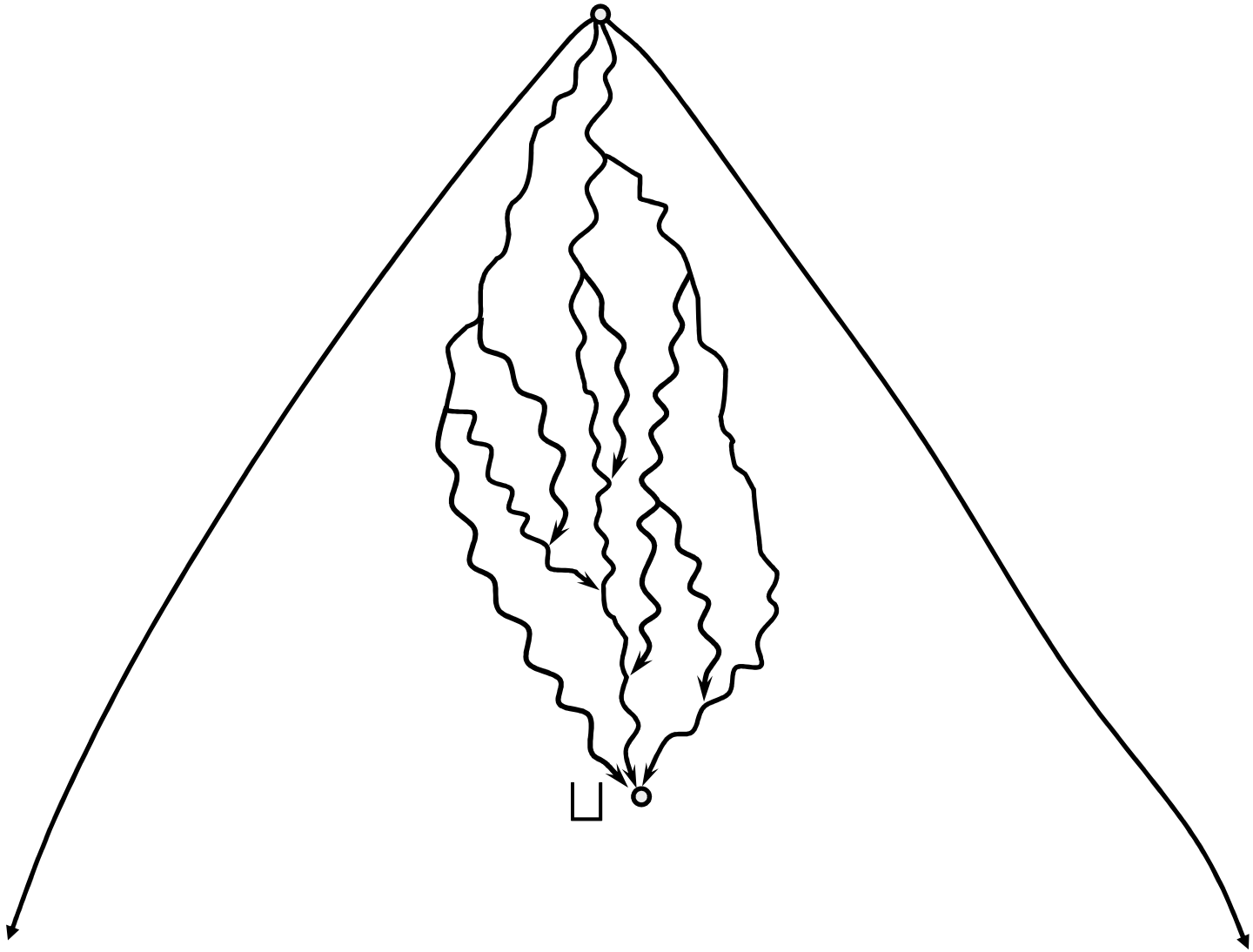
```
int callee(int a, int b) {
  int local;
  if (local == 5) return 1;
  else return 2;
}

int main() {
  int c = 5;
  int d = 7;

  int v = callee(c,d);
  // What is the value of v here?
  return 0;
}
```

Standard prolog		Prolog for 1 local	
push	ebp	push	ebp
mov	ebp, esp	mov	ebp, esp
sub	esp, 4	push	ecx

Answer: 1
(for the Microsoft compiler)

```
mov   [ebp+var_8], 5
mov   [ebp+var_C], 7
mov   eax, [ebp+var_C]
push  eax
mov   ecx, [ebp+var_8]
push  ecx
call  _callee
...
```

"Platform-Specific" ⇒ Fewer Behaviors



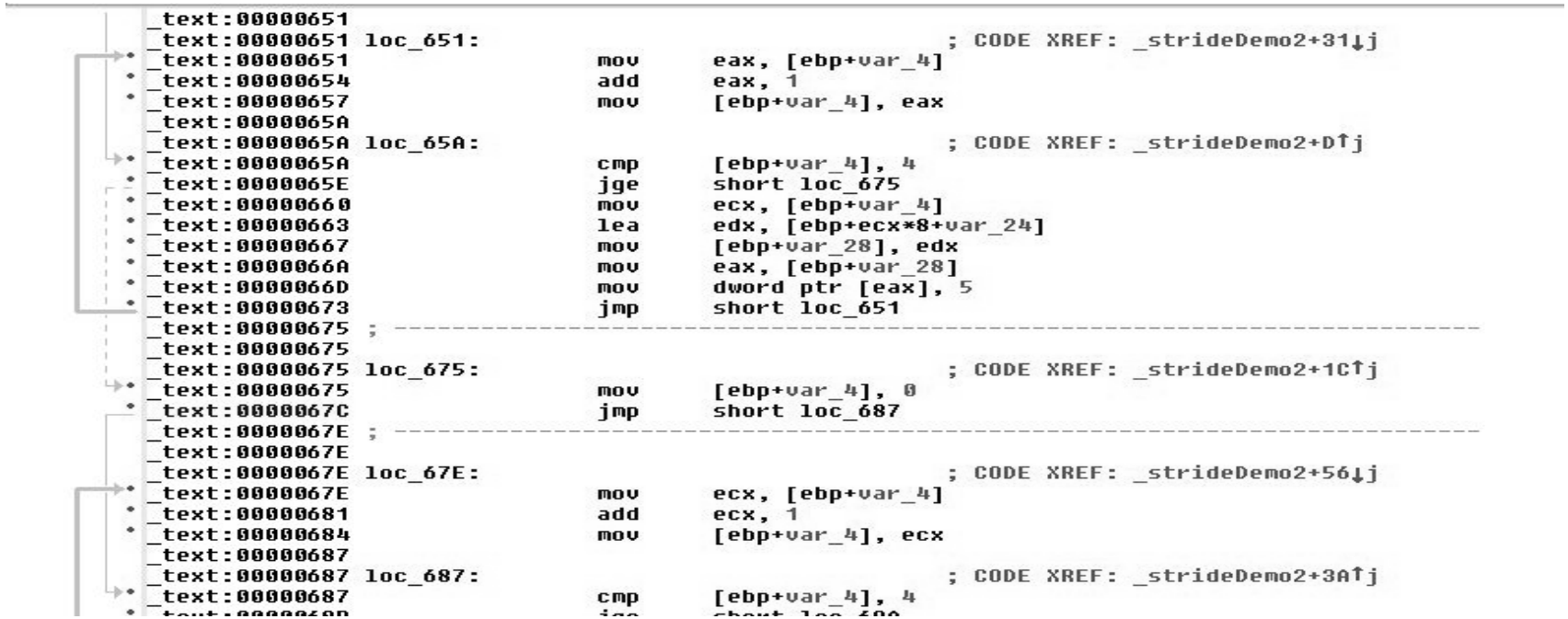
Analyze source code or executable?

An issue for any verification/analysis method

- Theorem Proving
- Model Checking
- Abstract Interpretation

Executable-Analysis Tools

- Disassemblers
  - Distinguish code and data
  - Requires manual effort to understand the program
  - e.g., IDAPro, OllyDbg, etc.



Executable-Analysis Tools

- Tools that perform data-flow analysis
  - e.g., EEL, Tools by Cifuentes, Debbabi, Debray
  - Able to track only data movements via registers
  - Poor treatment of memory operations
    - Overly conservative treatment ⇒ many false positives
    - Non-conservative treatment ⇒ many false negatives
  - Some tools aid in decompilation

```
mov [ebp – 10], 20
. . .
mov eax, [ebp – 10]
mov ebx, [ecx]
```

```
mov eax, [ebp – 10]
mov [ecx], eax
```

```
mov [ecx], [ebp - 10]
```

Executable-Analysis Tools

- Ad-hoc special-purpose analyzers
  - An analyzer to identify indirect calls
  - An analyzer to identify strings
  - An analyzer to check stack height
  - etc.,
- Program analysis frameworks
  - Rely on symbol-table/debugging info
  - e.g., Atom, Vulcan, Phoenix

What is the holy grail?

- General platform for analyzing executables
- Tracks data movement through memory
  - including heap
- Must not rely on debugging information
- Gives information to build further analysis
  - like a compiler front-end plus some more

Outline

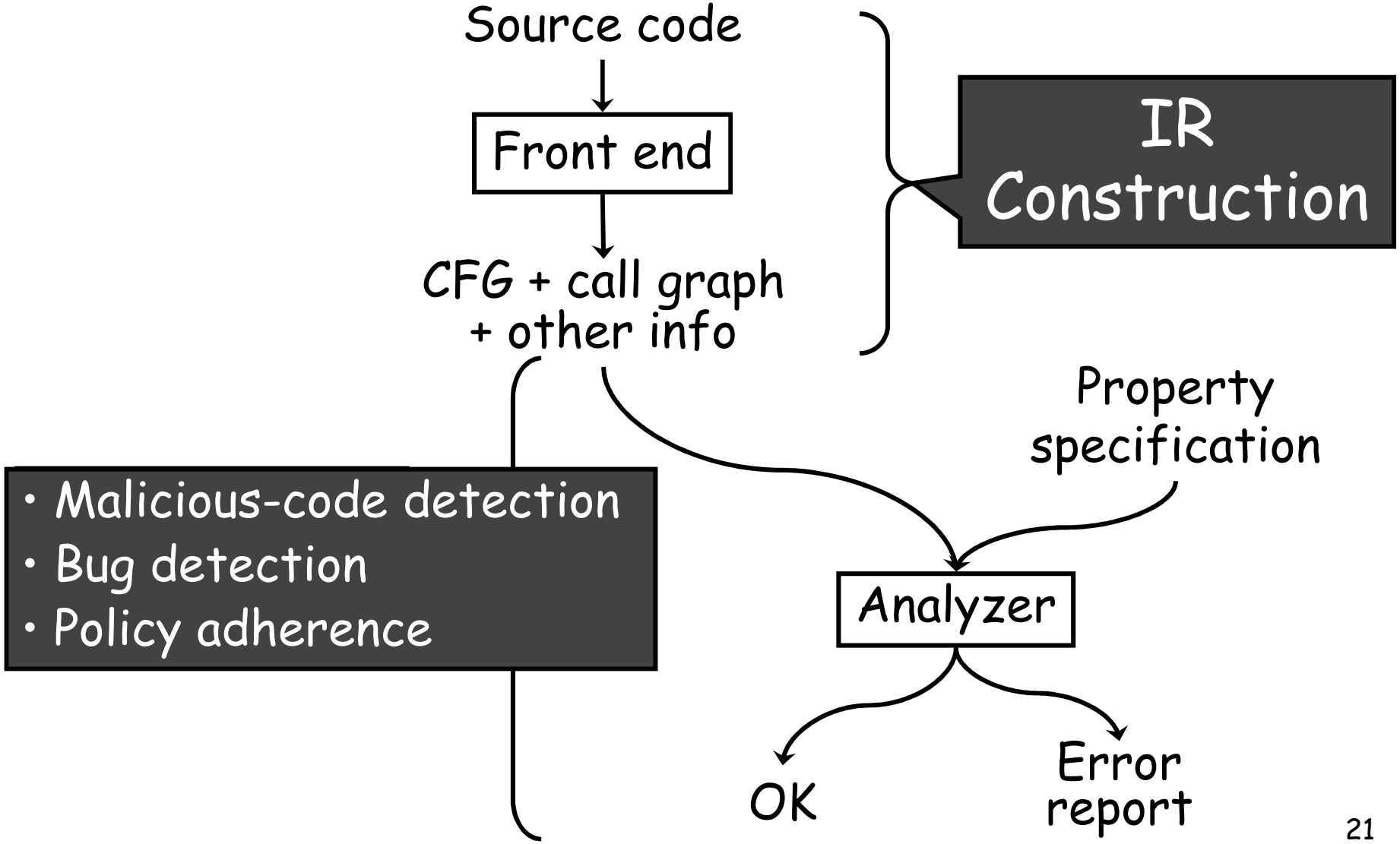
- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery

- What Next?

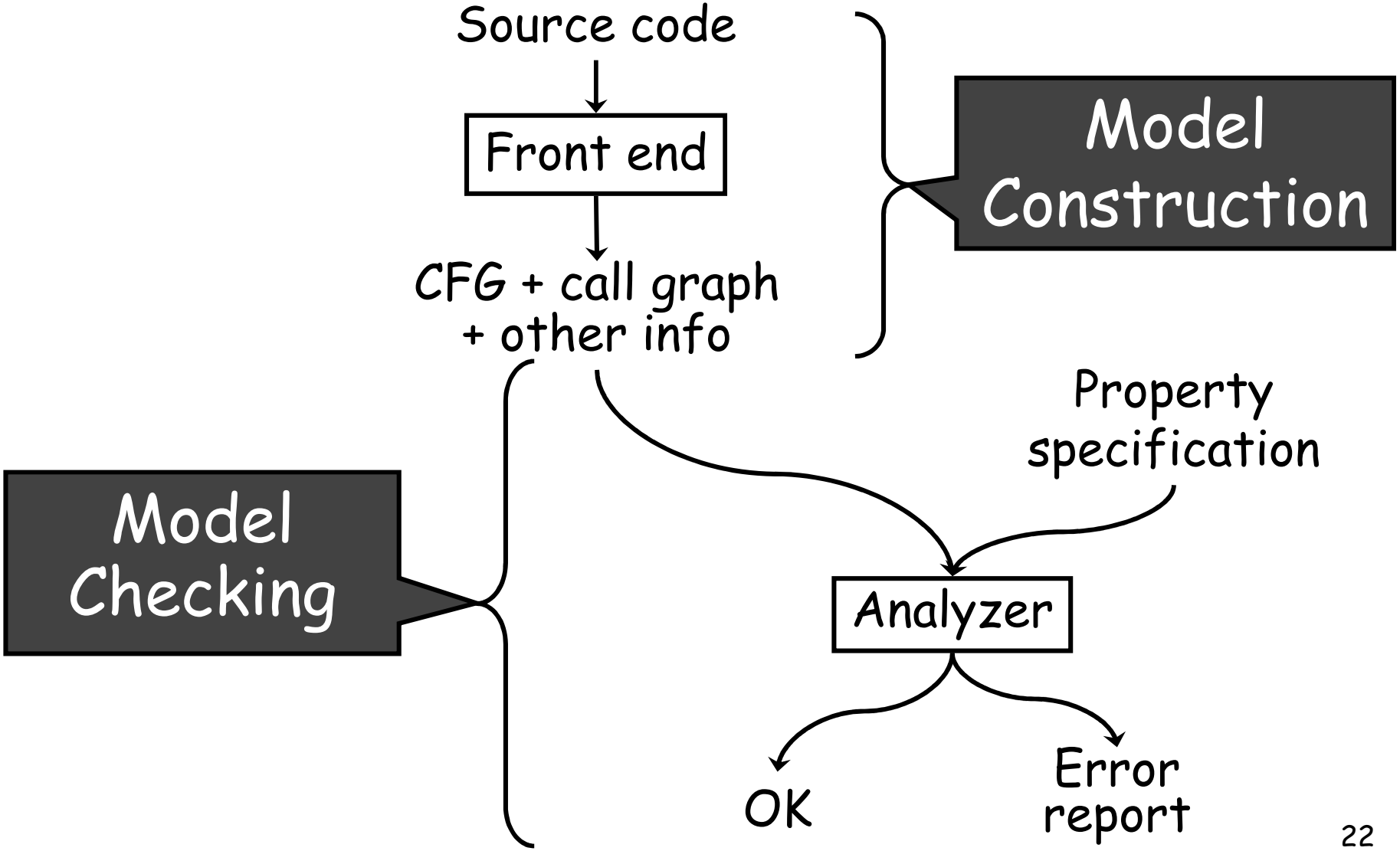
Outline

- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery
- What Next?

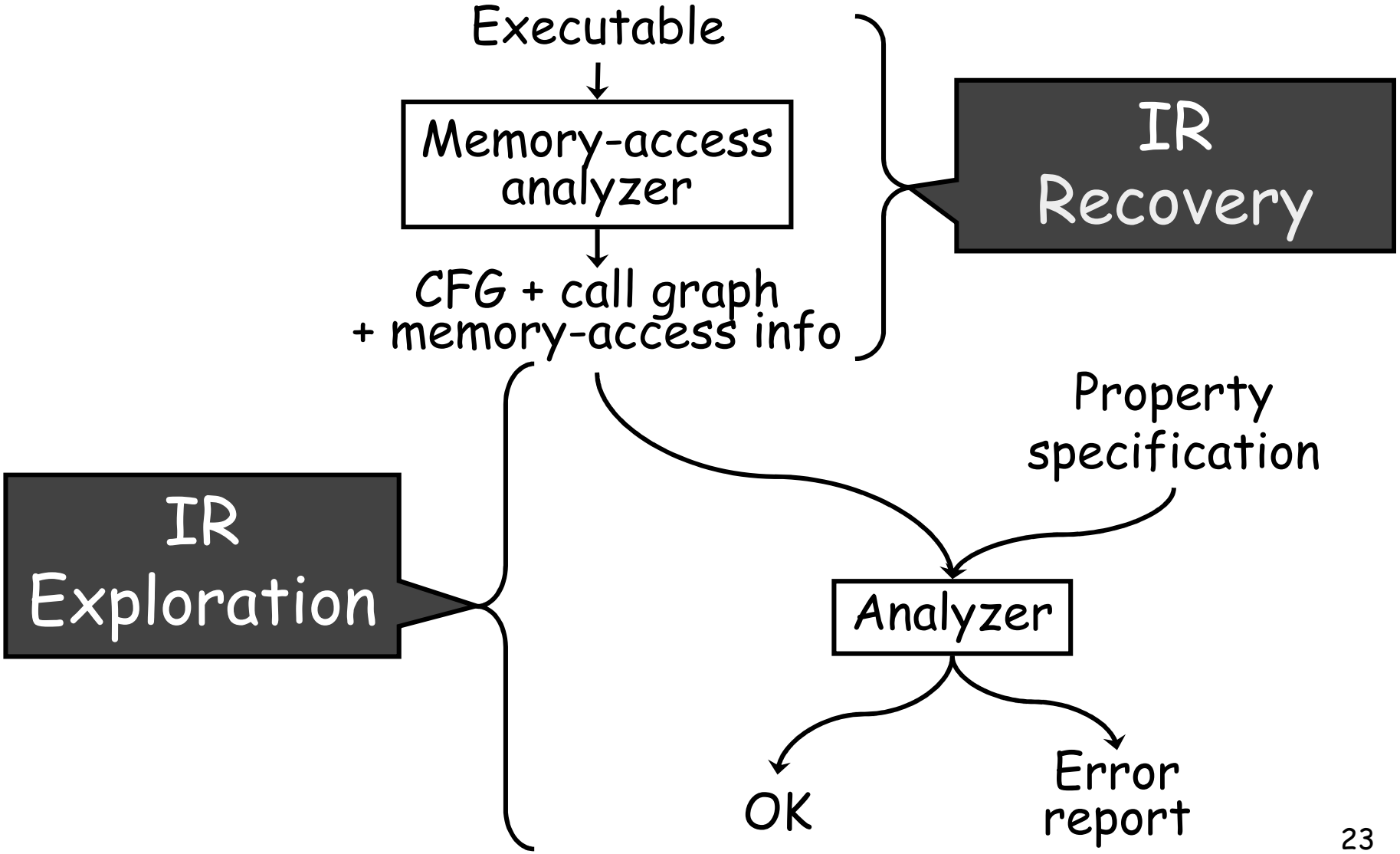
Static Program-Analysis Tools



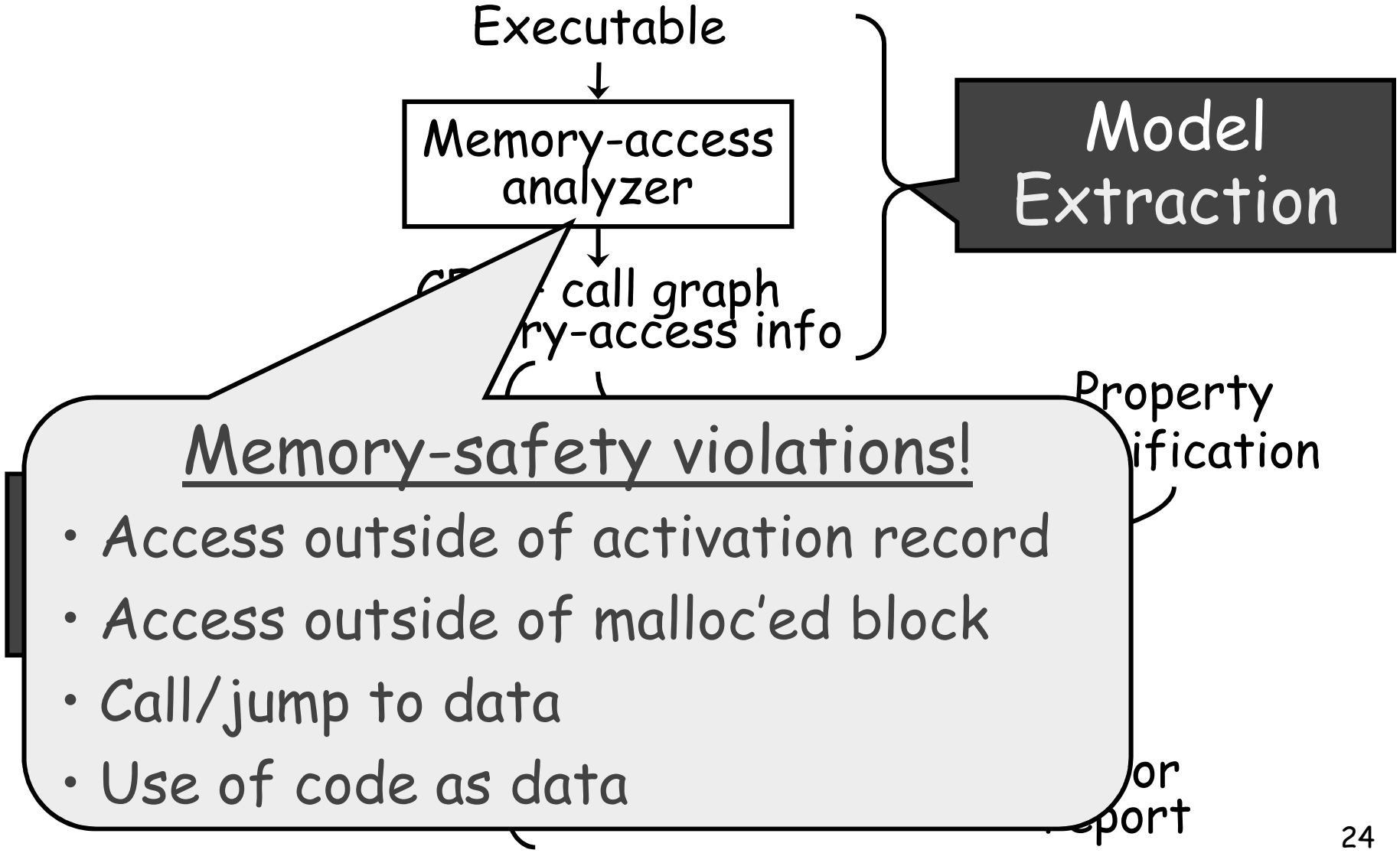
Static Program-Analysis Tools



Static Executable-Analysis Tools



Static Executable-Analysis Tools



Our Approach

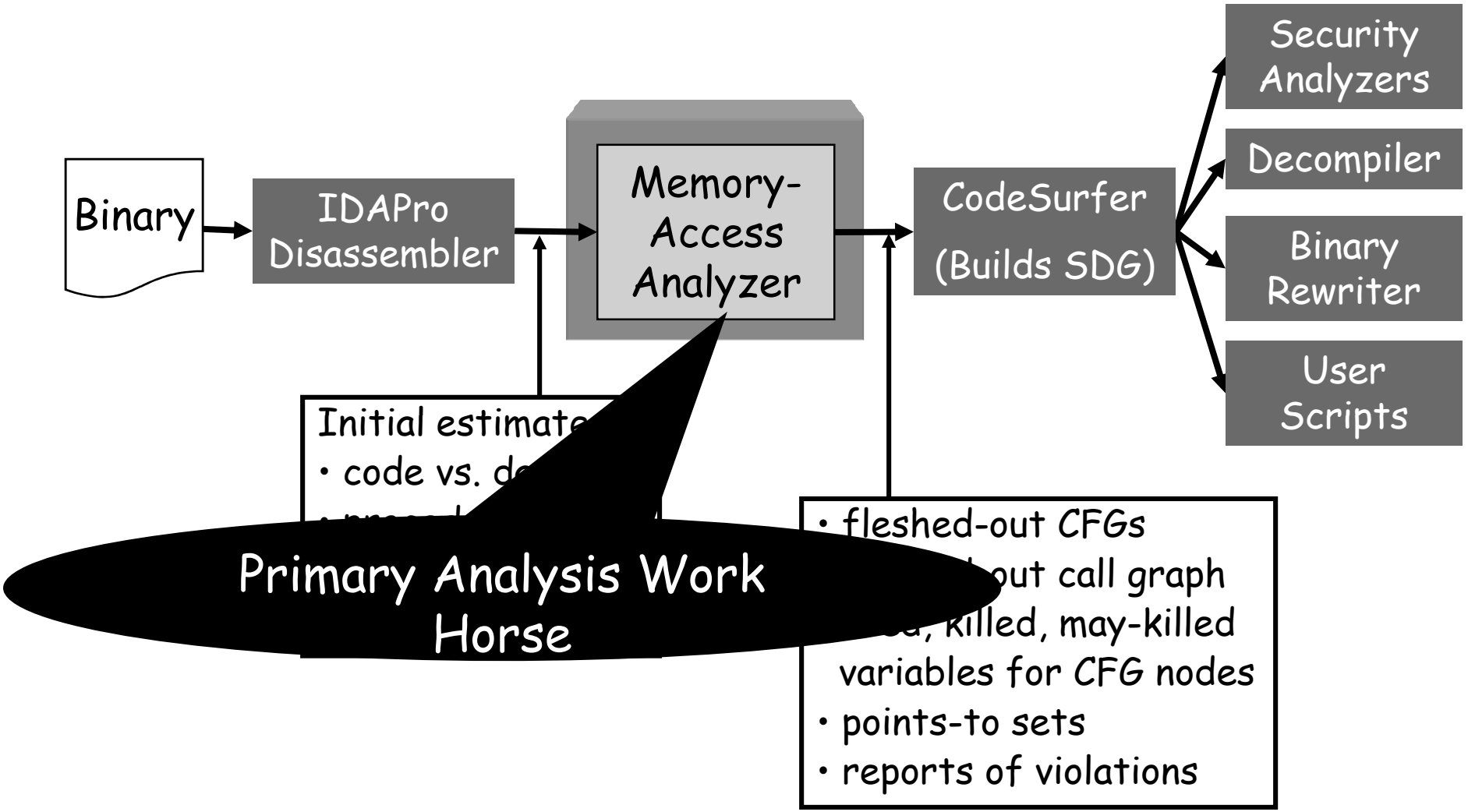
- Recover an Intermediate Representation (IR) from the executable
  - IR similar to that built by a compiler
    - control-flow graphs
    - call graph
    - set of variables
    - values of pointers
    - used, killed, and possibly-killed variables for CFG nodes
    - data dependences
    - types of variables: base types, pointer types, structs, and classes
- Use the recovered IR for further analysis
  - Finding bugs and security vulnerabilities
  - decompilation
  - . . .

Without Debugging Information!

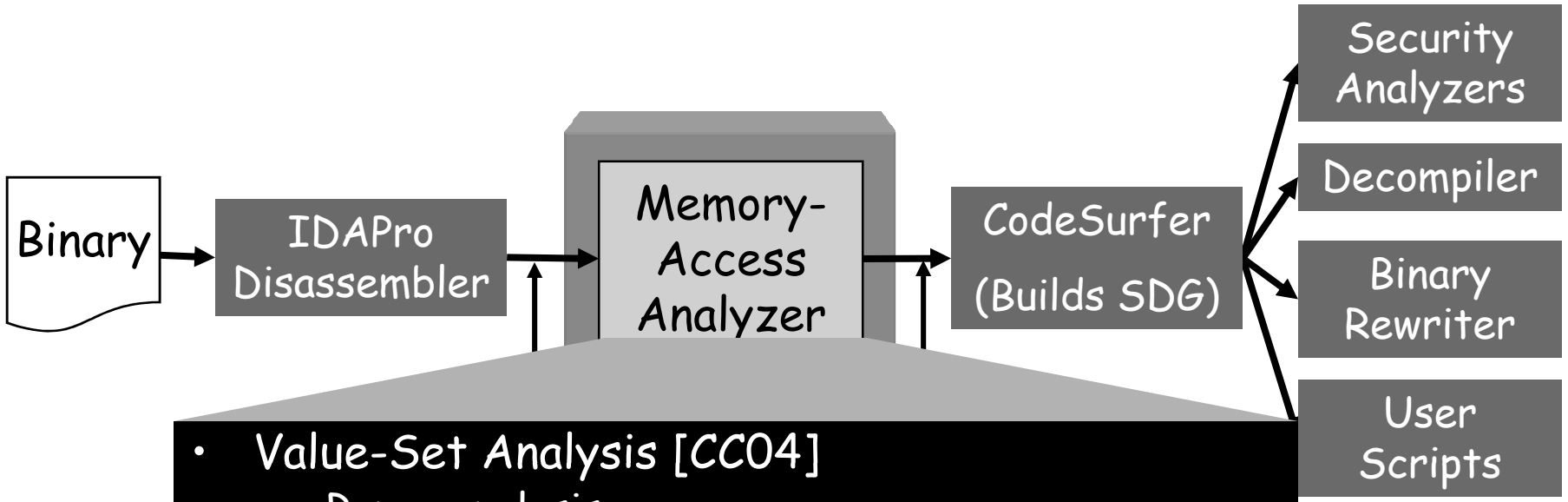
Scope

- Programs that conform to a "standard compilation model"
  - procedures
  - activation records
  - global data region
  - heap, etc.
- Report violations
  - violations of stack protocol
  - return address modified within procedure

CodeSurfer/x86 Architecture



CodeSurfer/x86 Architecture



- Value-Set Analysis [CC04]
  - Range analysis
  - Pointer analysis
  - Indirect-jump & indirect-call resolution
  - Induction variables
- Variable recovery algorithm [VMCAI07]
  - Structure and Type recovery
- Analyzes heap-allocated data
  - Recency-abstraction [SAS06]

Outline

- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery

- What Next?

Running Example

```
int arrVal=0, *pArray2;

int main() {
   int i, a[10], *p;
   /* Initialize pointers */
   pArray2 = &a[2];
   p = &a[0];
   /* Initialize Array */
   for(i = 0; i<10; ++i) {
      *p = arrVal;
      p++;
   }
   /* Return a[2] */
   return *pArray2;
}
```

```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub    esp, 40        ;adjust stack
lea    edx, [esp+8]   ;
mov    [8], edx       ;pArray2=&a[2]
lea    ecx, [esp]     ;p=&a[0]
mov    edx, [4]       ;

loc_9:
   mov    [ecx], edx  ;*p=arrVal
   add    ecx, 4      ;p++
   inc    ebx         ;i++
   cmp    ebx, 10     ;i<10?
   jl     short loc_9 ;

mov    edi, [8]     ;
mov    eax, [edi]   ;return *pArray2
add    esp, 40
retn
```

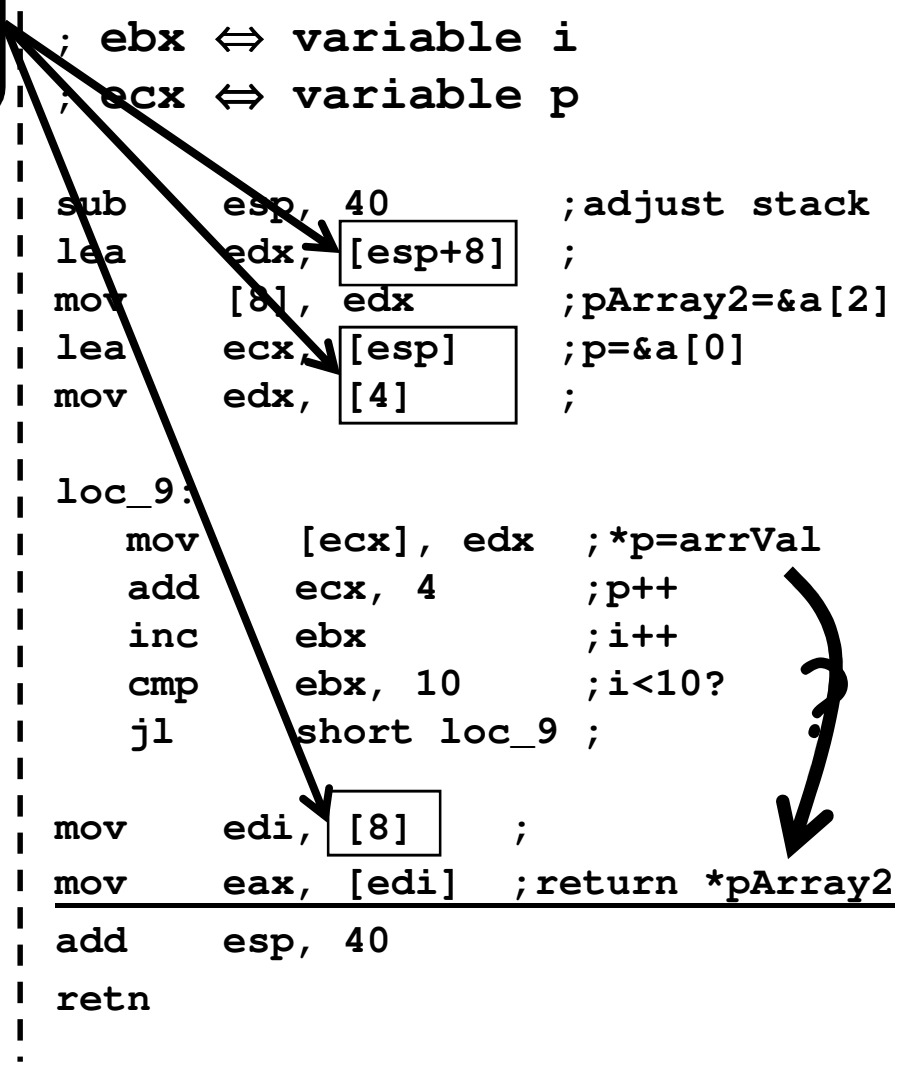
Running Example

```
int arrVal=0, *pArray2;

int main() {
    int i, a[10], *p;
   /* Initialize pointers */
    pArray2 = &a[2];
    p = &a[0];
   /* Initialize Array */
    for(i = 0; i<10; ++i) {
        *p = arrVal;
        p++;
    }
   /* Return a[2] */
    return *pArray2;
}
```

```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, [esp+8]    ;
mov     [8], edx        ;pArray2=&a[2]
lea     ecx, [esp]      ;p=&a[0]
mov     edx, [4]        ;

loc_9:
    mov     [ecx], edx  ;*p=arrVal
    add     ecx, 4      ;p++
    inc     ebx         ;i++
    cmp     ebx, 10     ;i<10?
    jl      short loc_9 ;

mov     edi, [8]        ;
mov     eax, [edi]      ;return *pArray2
add     esp, 40
retn
```

Challenges

No debugging information
(Unavailable/Unreliable)

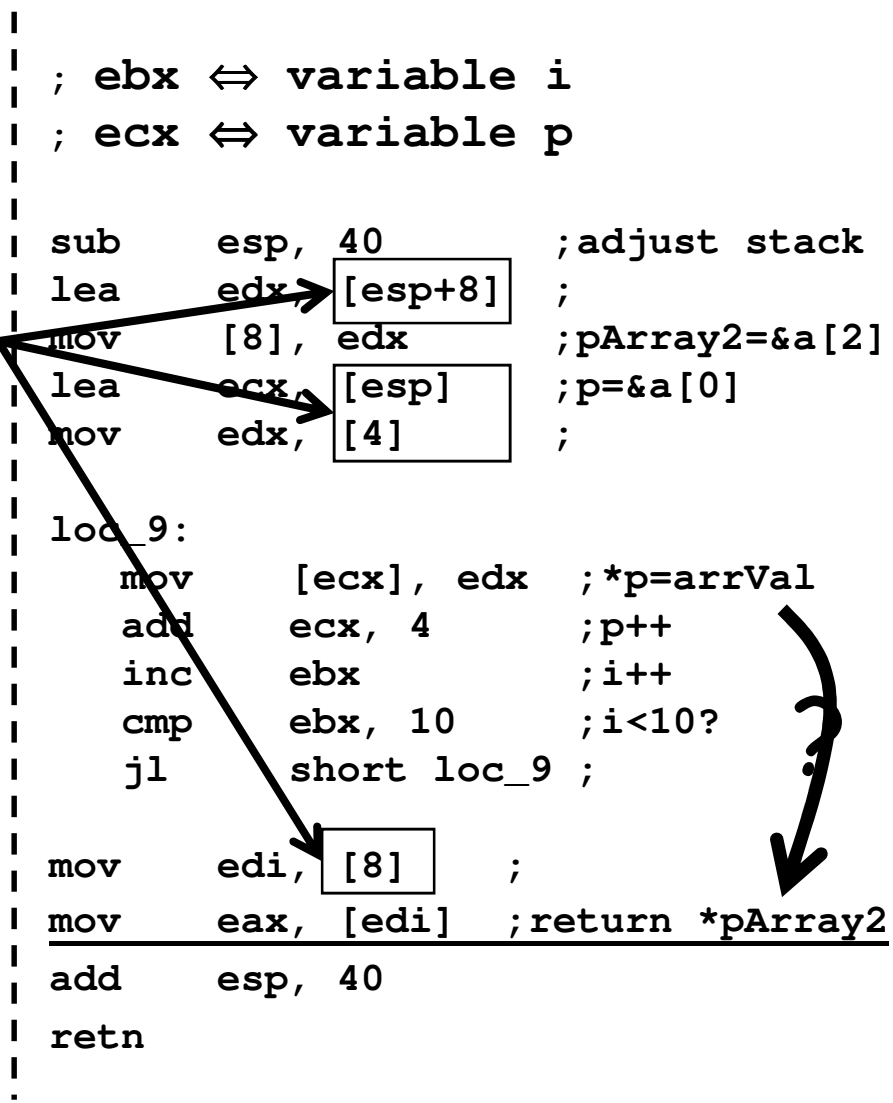


```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, [esp+8]    ;
mov     [8], edx        ;pArray2=&a[2]
lea     ecx, [esp]      ;p=&a[0]
mov     edx, [4]        ;

loc_9:
   mov     [ecx], edx   ;*p=arrVal
   add     ecx, 4       ;p++
   inc     ebx          ;i++
   cmp     ebx, 10      ;i<10?
   jl      short loc_9  ;

mov     edi, [8]     ;
mov     eax, [edi]   ;return *pArray2
add     esp, 40
retn
```

Challenges

No debugging information (Unavailable/Unreliable)

No notion of variables (Explicit memory addresses/offsets)

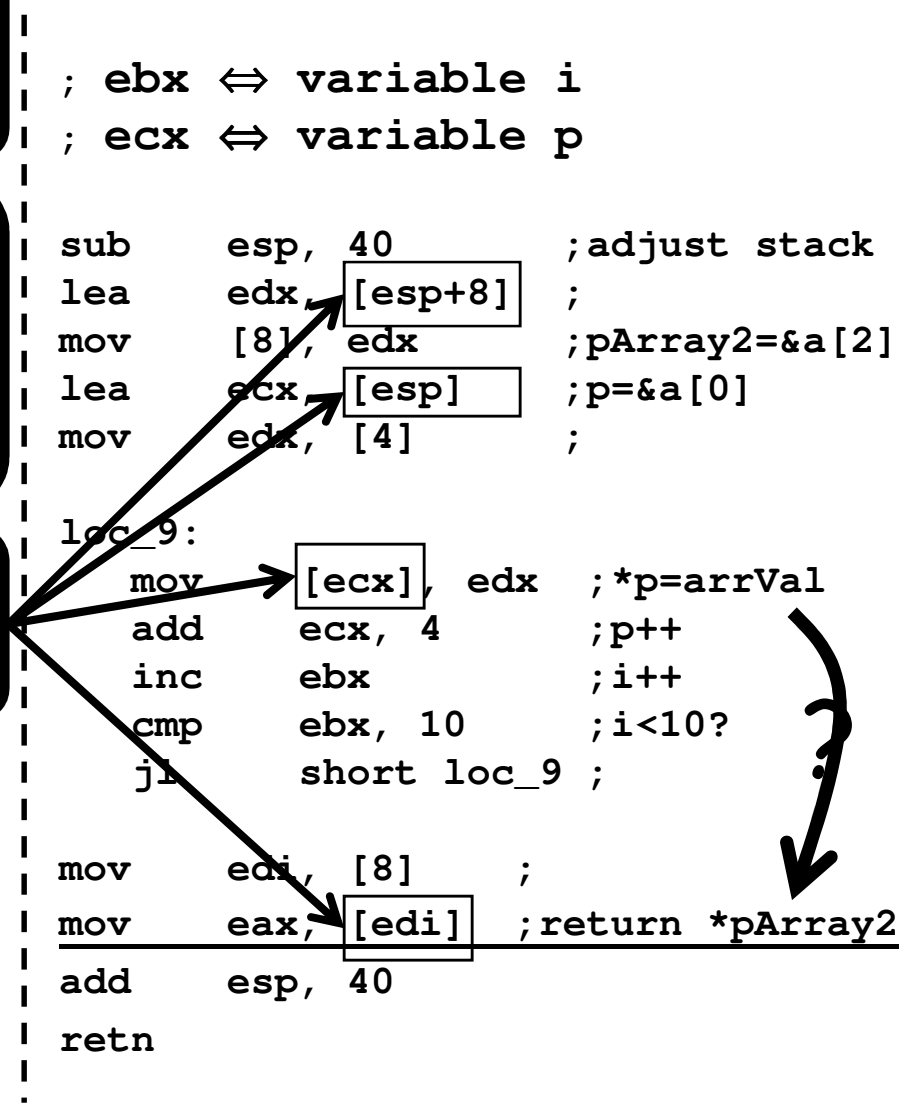


Challenges

No debugging information (Unavailable/Unreliable)

No notion of variables (Explicit memory addresses/offsets)

Indirect Addressing (Need Pointer Analysis)



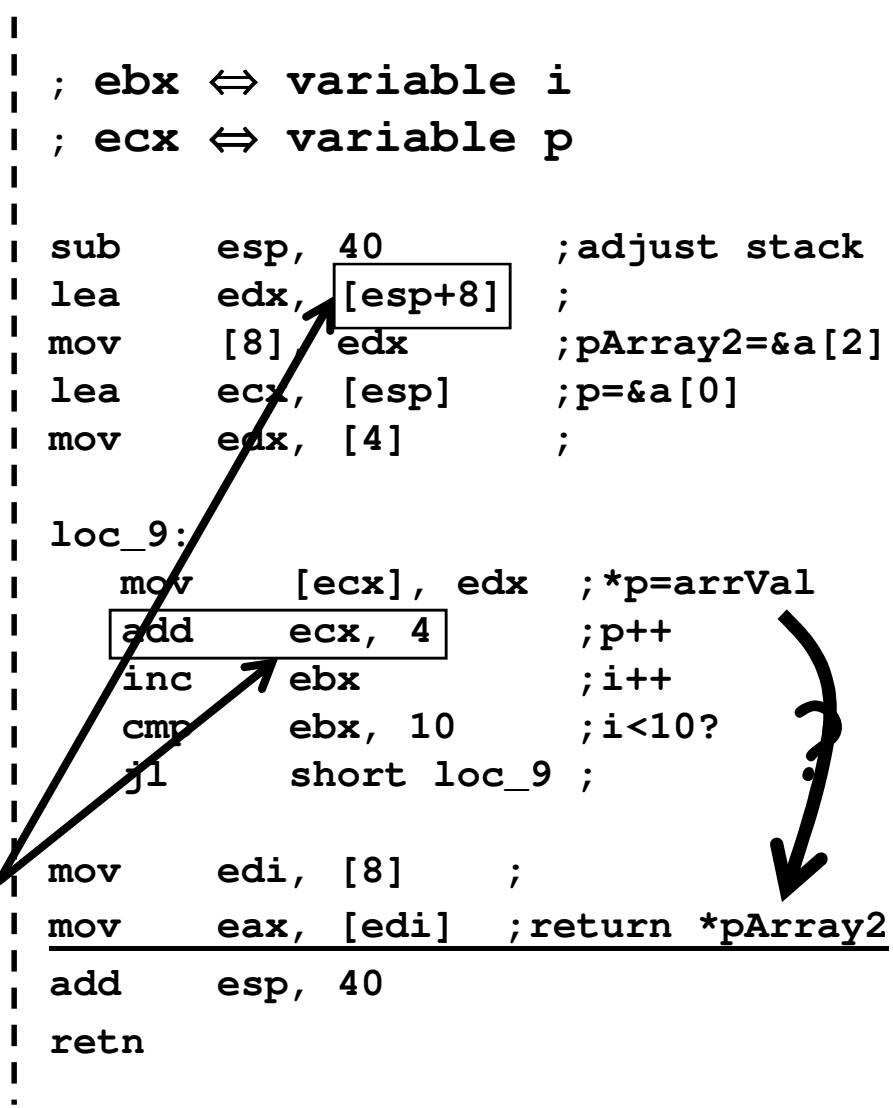
Challenges

No debugging information (Unavailable/Unreliable)

No notion of variables (Explicit memory addresses/offsets)

Indirect Addressing (Need Pointer Analysis)

Pointer Arithmetic (Need Numeric Analysis e.g., "Range Analysis")



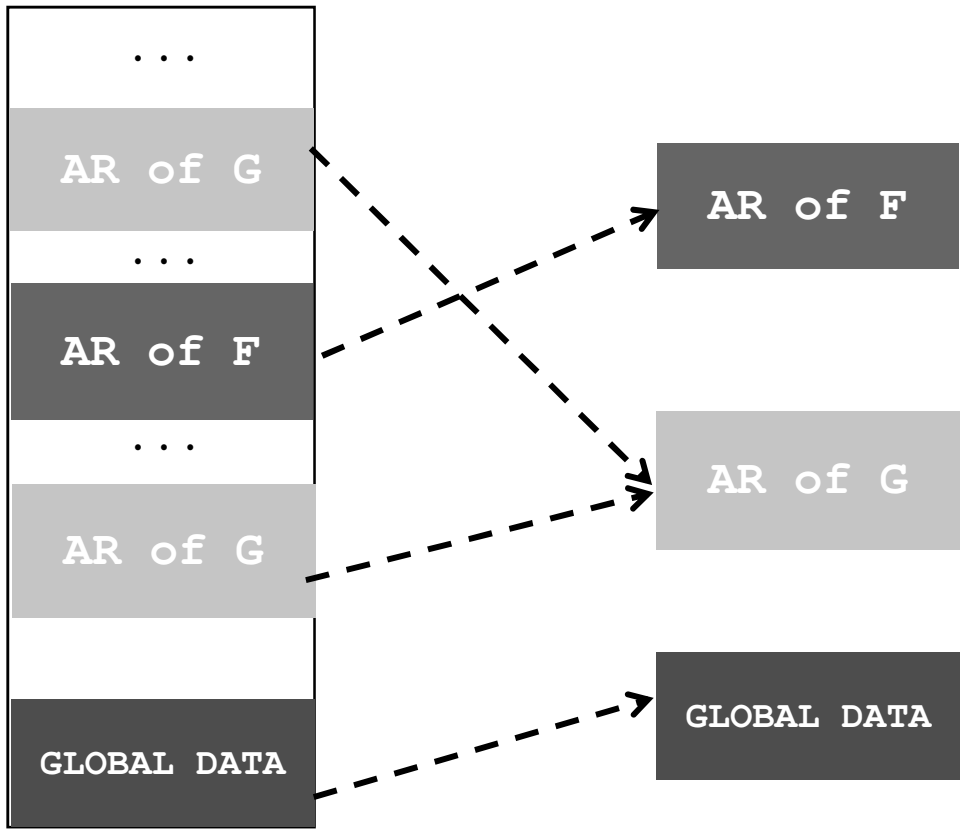
Outline

- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery

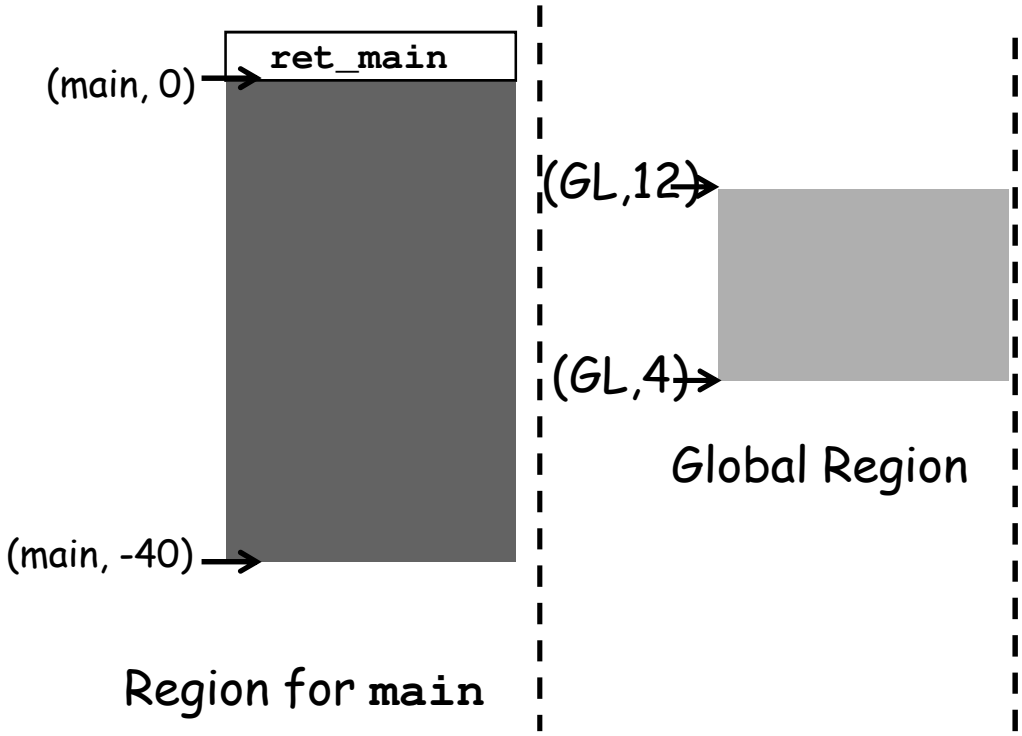
- What Next?

Memory-Regions

- A memory-region: a sequence of similar runtime addresses
  - AR-region: Addresses that belong to an activation record
  - Malloc-region: Addresses that are allocated at a malloc site
  - Global-region: Addresses that correspond to global data



Example - Memory Regions




```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, [esp+8]    ;
mov     [8], edx        ;pArray2=&a[2]
lea     ecx, [esp]      ;p=&a[0]
mov     edx, [4]        ;

loc_9:
   mov     [ecx], edx   ;*p=arrVal
   add     ecx, 4       ;p++
   inc     ebx          ;i++
   cmp     ebx, 10      ;i<10?
   jl      short loc_9  ;

mov     edi, [8]     ;
mov     eax, [edi]   ;return *pArray2
add     esp, 40
retn
```

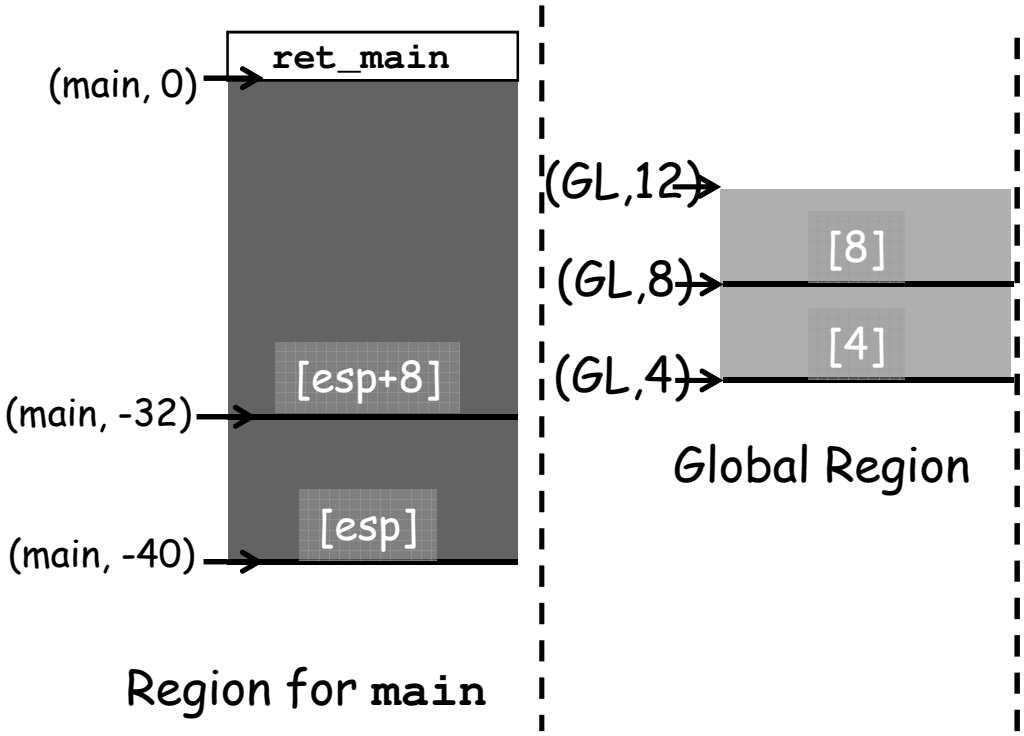
Recovering Variable-like Entities (a-loc)

- Data-layout known at assembly/compile time
  - some variables held in registers
  - global variables → absolute addresses
  - local variables → offsets in stack frame

- A-locs (for "Abstract-Locations")
  - locations between consecutive addresses
  - locations between consecutive offsets
  - Registers

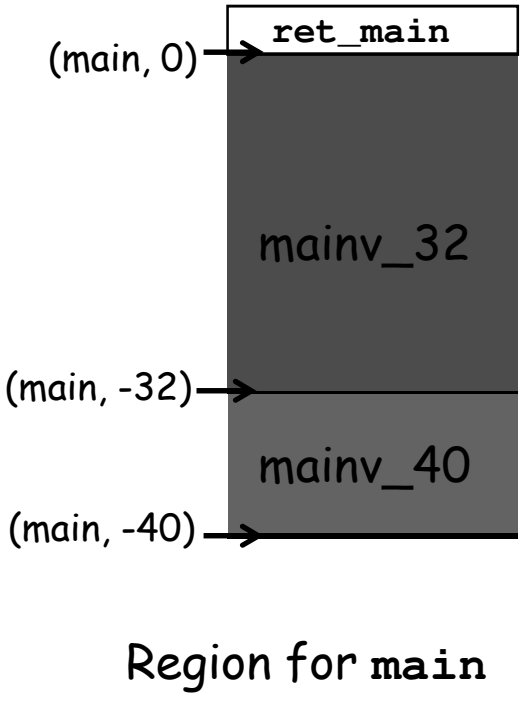
- Referred to as the "Semi-Naïve algorithm"

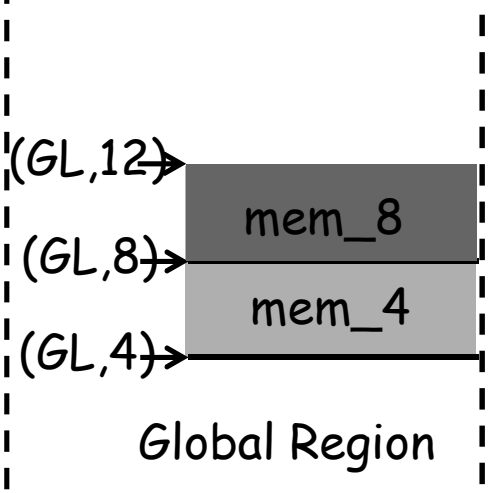
Example: Variable-like Entities




```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40          ;adjust stack
lea     edx, [esp+8]     ;
mov     [8], edx         ;pArray2=&a[2]
lea     ecx, [esp]       ;p=&a[0]
mov     edx, [4]         ;

loc_9:
   mov     [ecx], edx    ;*p=arrVal
   add     ecx, 4        ;p++
   inc     ebx           ;i++
   cmp     ebx, 10       ;i<10?
   jl      short loc_9   ;

mov     edi, [8]         ;
mov     eax, [edi]       ;return *pArray2
add     esp, 40
retn
```

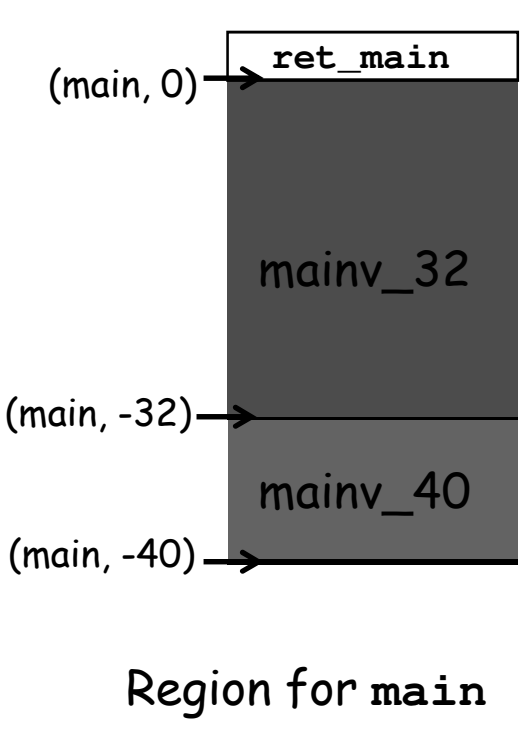
Example: Variable-like Entities

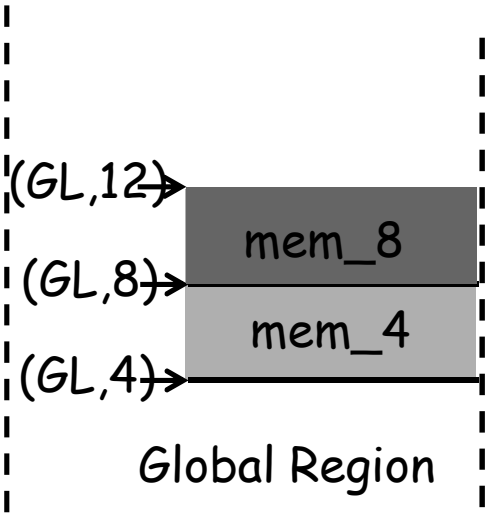





```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40          ;adjust stack
lea     edx, [esp+8]     ;
mov     [8], edx         ;pArray2=&a[2]
lea     ecx, [esp]       ;p=&a[0]
mov     edx, [4]         ;

loc_9:
   mov     [ecx], edx    ;*p=arrVal
   add     ecx, 4        ;p++
   inc     ebx           ;i++
   cmp     ebx, 10       ;i<10?
   jl      short loc_9   ;

mov     edi, [8]         ;
mov     eax, [edi]       ;return *pArray2
add     esp, 40
retn
```

Example: Variable-like Entities

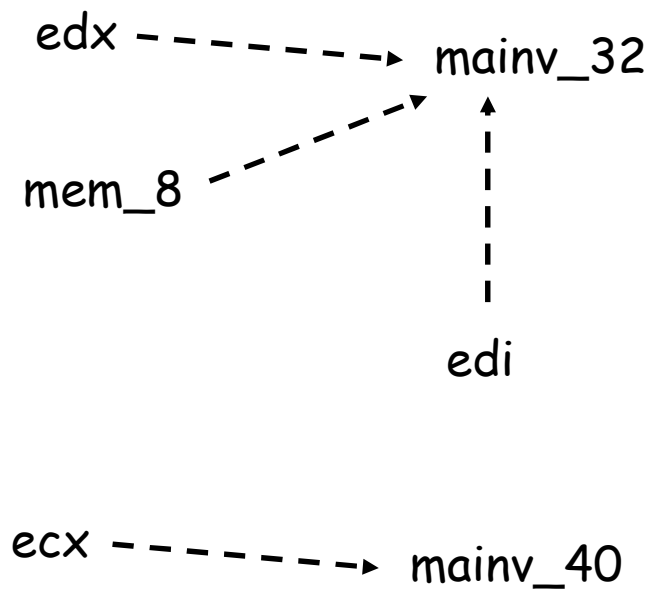





```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40          ;adjust stack
lea     edx, &mainv_32;
mov     mem_8, edx       ;pArray2=&a[2]
lea     ecx, &mainv_40;p=&a[0]
mov     edx, mem_4       ;

loc_9:
   mov     [ecx], edx    ;*p=arrVal
   add     ecx, 4        ;p++
   inc     ebx           ;i++
   cmp     ebx, 10       ;i<10?
   jl      short loc_9 ;

mov     edi, mem_8     ;
mov     eax, [edi]   ;return *pArray2
add     esp, 40
retn
```

Example: Standard Pointer Analysis?

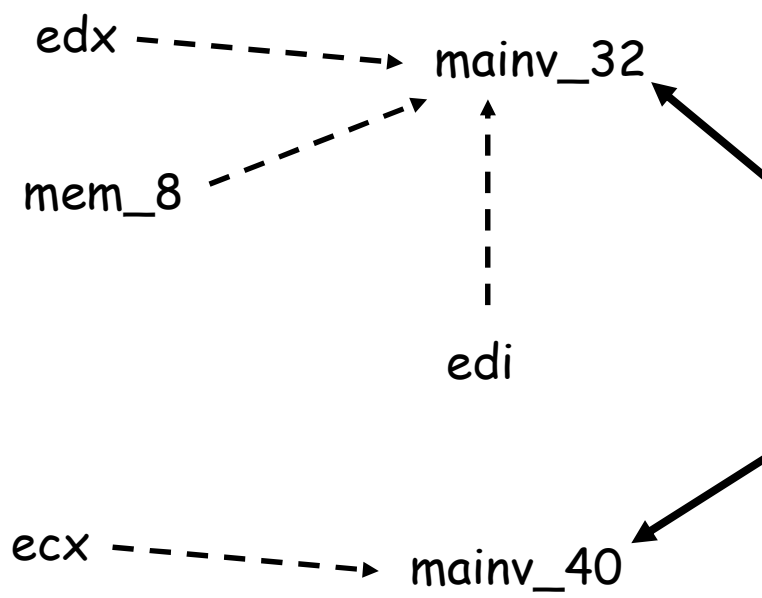
locals: mainv_40, mainv_32
{a[0..1], a[2..9]}
globals: mem_4, mem_8
{arrVal, pArray2}




```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, &mainv_32;
mov     mem_8, edx      ;pArray2=&a[2]
lea     ecx, &mainv_40;p=&a[0]
mov     edx, mem_4      ;

loc_9:
   mov     [ecx], edx   ;*p=arrVal
   add     ecx, 4       ;p++
   inc     ebx          ;i++
   cmp     ebx, 10      ;i<10?
   jl      short loc_9 ;

mov     edi, mem_8      ;
mov     eax, [edi]   ;return *pArray2
add     esp, 40
retn
```

Example: Standard Pointer Analysis?

locals: mainv_40, mainv_32
{a[0..1], a[2..9]}
globals: mem_4, mem_8
{arrVal, pArray2}




```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, &mainv_32;
mov     mem_8, edx      ;pArray2=&a[2]
lea     ecx, &mainv_40;p=&a[0]
mov     edx, mem_4      ;

loc_9:
   mov     [ecx], edx   ;*p=arrVal
   add     ecx, 4       ;p++
   inc     ebx          ;i++
   cmp     ebx, 10      ;i<10?
   jl      short loc_9 ;

mov     edi, mem_8      ;
mov     eax, [edi]   ;return *pArray2
add     esp, 40
retn
```

Outline

- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery

- What Next?

Value-set Analysis [CC04]

- For each a-loc, at every program point
  - Find the set of addresses/values held by the a-loc
  - Based on Abstract Interpretation
  - Abstract domain: value-set

- Resembles a pointer-analysis algorithm
  - Over-approximation for the set of addresses
  - Interprets pointer-manipulation operations
  - Pointer arithmetic, too

- Resembles a numeric-analysis algorithm
  - Over-approximation for the set of values

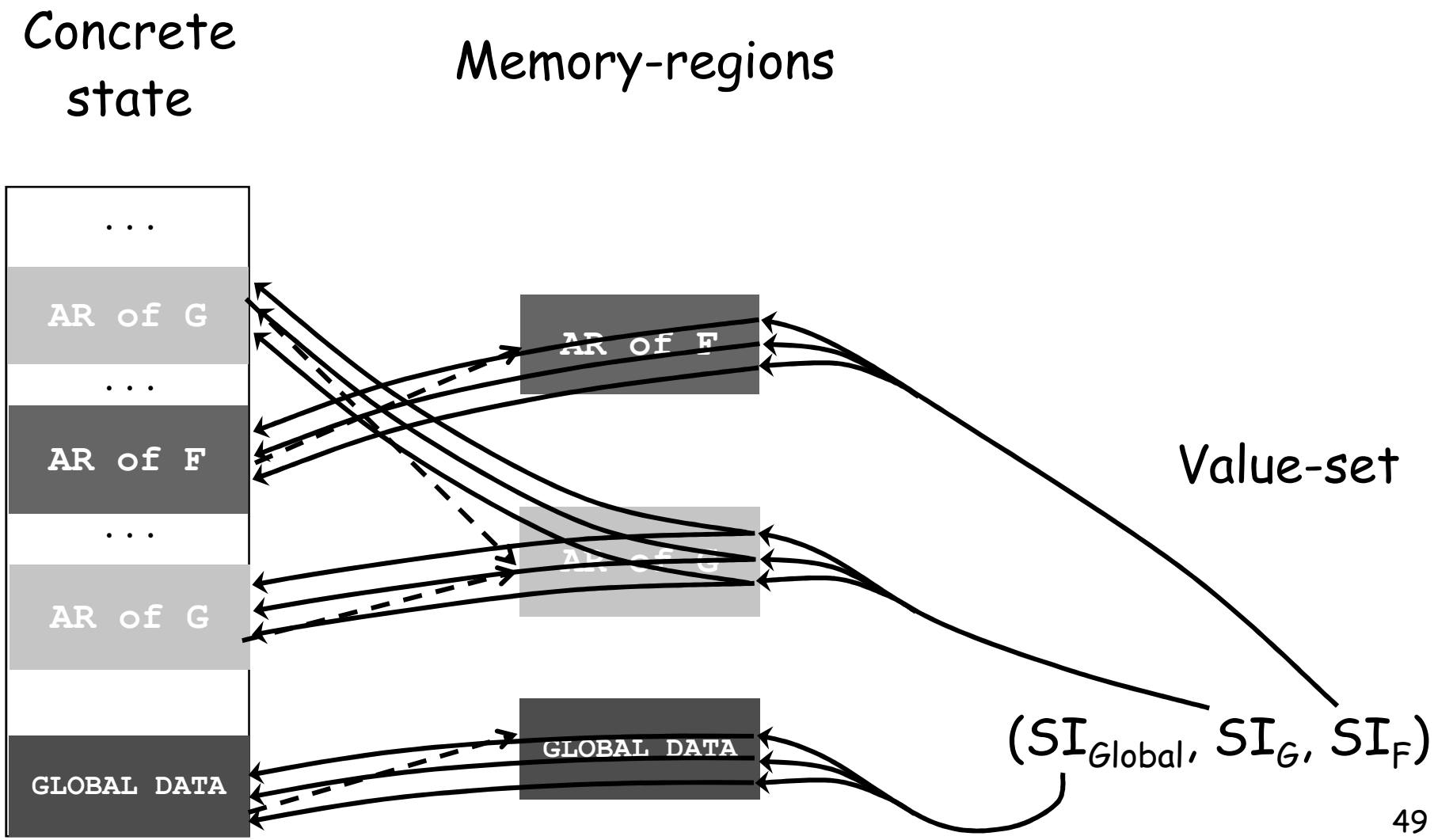
Value-set [PEPM06]

- A representation for a set of addresses/values

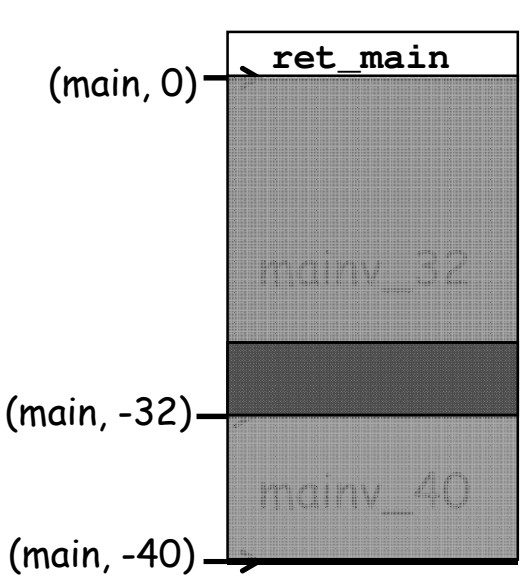
- Strided-Interval (SI)
  - represents a set of values
  - records a range and a stride
  - {1, 3, 5, 9} represented as 2[1,9]
  - conservative: 2[1,9] represents {1, 3, 5, 7, 9}

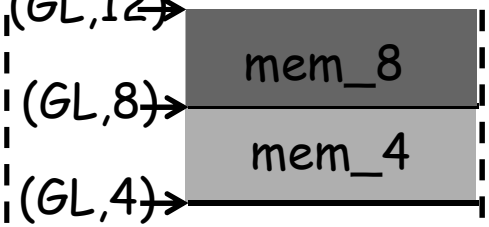
- Value-set
  - r-tuple of SIs: (si_1, . . ., si_r)
  - i^{th} component — offsets in i^{th} memory-region
  - si_1 – offsets in global region
  - (si_1, Ø, . . ., Ø) – set of numbers

Value-Set:
A Set of Abstract Addresses and Values



Example - Value-set analysis





Global Region

Region for **main**

1 **ecx** → (Ø , 4[-40,-4])
ebx → (1[0,9] , Ø)

2 **edi** → (Ø, -32)

```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, [esp+8]    ;
mov     [8], edx        ;pArray2=&a[2]
lea     ecx, [esp]      ;p=&a[0]
mov     edx, [4]        ;

loc_9:
   mov     [ecx], edx   ;*p=arrVal   1
   add     ecx, 4       ;p++
   inc     ebx          ;i++
   cmp     ebx, 10      ;i<10?
   jl      short loc_9  ;

mov     edi, [8]     ;
mov     eax, [edi]   ;return *pArr   2
add     esp, 40
retn
```

Outline

- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery

- What Next?

Static-Analysis Algorithms in CodeSurfer/x86

- Value-Set Analysis [CC04]
- Improved Aloc-Recovery Algorithm [VMCAI07]
- Recency-Abstraction For Heap-Allocated Storage [SAS06]
- Affine-Relation Analysis (ARA) [CC04, CAV05]

- Path-Sensitive VSA [TACAS08]
- Improvements to VSA
  - GMOD-based merge function [CC08]
  - Priority-based iteration
  - Improved widening with Bourdoncle components
- Windows Device-driver Analysis [TACAS08]

Outline

- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery

- What Next?

Device-Driver Analysis

- Device Driver
  - like a library that exports procedures
  - each procedure: actions for an I/O request
    - e.g., `AddDevice` routine
      - invoked by OS when a new device is added
    - referred to as "dispatch routines"

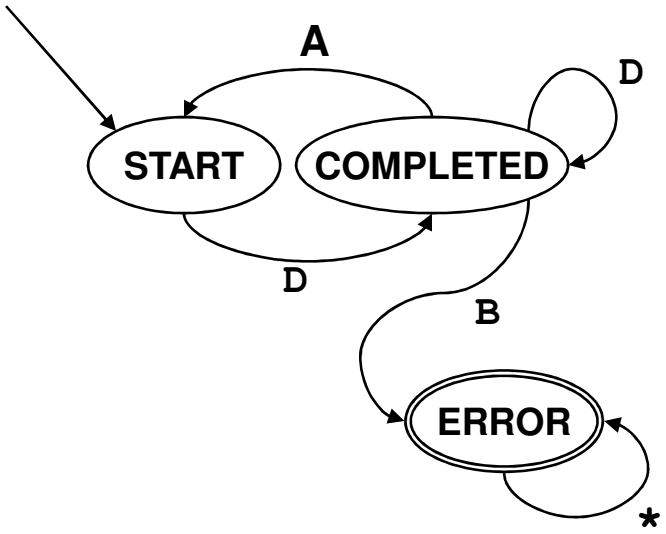
- Windows Kernel API is complex
  - 85% of crashes in Windows due to driver bugs
    - [Swift et al. 2005]

Device Driver Analysis

". . . You must examine the object code to be sure it matches your expectations, or at least will work correctly in the kernel environment. . ."

(From a document on Microsoft's WHDC website)

PendedCompletedRequested Rule (simplified version)



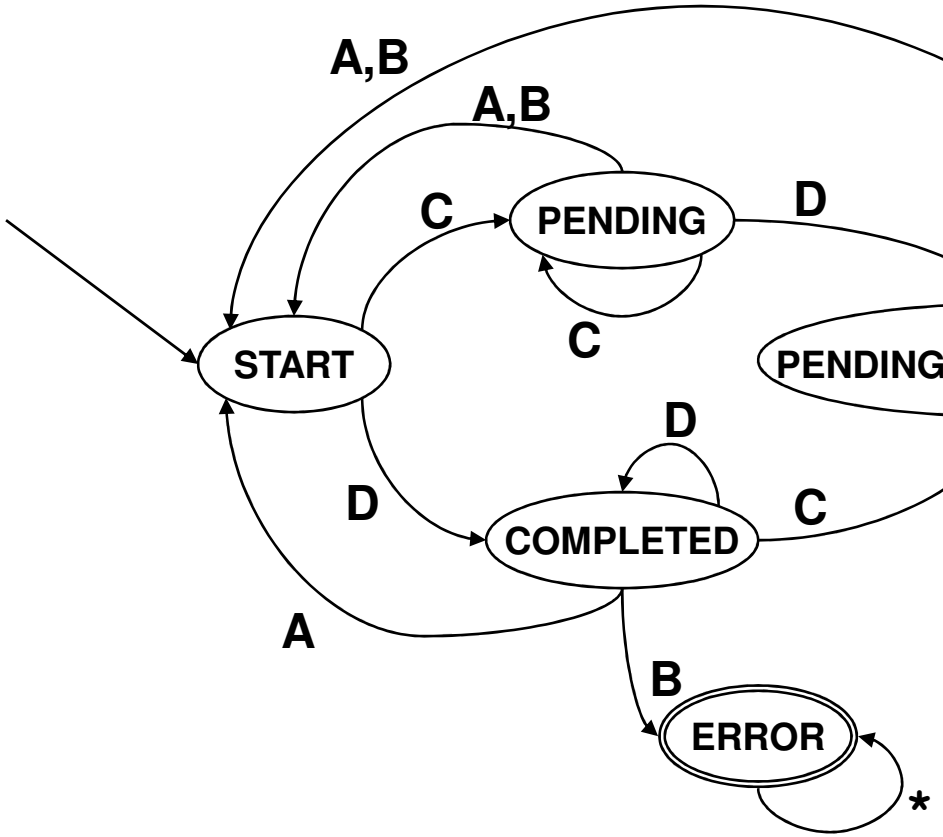
A: `return value ≠ STATUS_PENDING`

B: `return value = STATUS_PENDING`

D: `IoCompleteRequest`

"A driver's dispatch routine does not return **`STATUS_PENDING`** on an I/O Request Packet (IRP) if it has called ***`IoCompleteRequest`*** on the IRP."

PendedCompletedRequested Rule



A: `"return status ≠ STATUS_PENDING"`

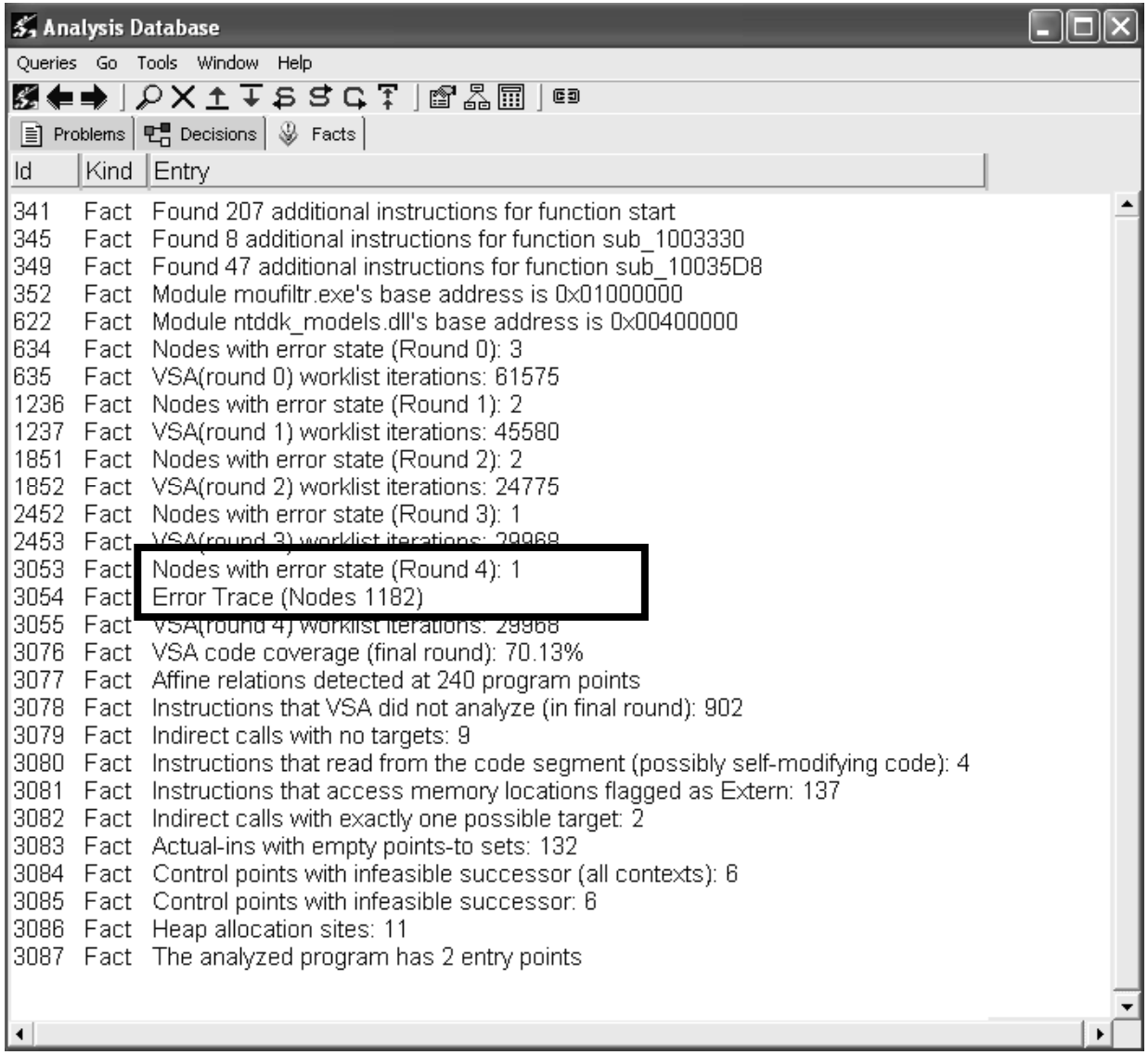
B: `"return status = STATUS_PENDING"`

C: ***`IoMarkPending`***

D: ***`IoCompleteRequest`***

"A driver's dispatch routine does not return **`STATUS_PENDING`** on an I/O Request Packet (IRP) if it has called ***`IoCompleteRequest`*** on the IRP, unless it has also called ***`IoMarkIrpPending`***."

CS/x86: Device Driver Analysis



SLAM Error Trace

```
KeInitializeEvent(&event,
                  NotificationEvent,
                  FALSE
                  );

IoSetCompletionRoutine(Irp,
                       (PIO_COMPLETION_ROUTINE) MouFilter_Complete,
                       &event,
                       TRUE,
                       TRUE,
                       TRUE); // No need for Cancel

status = IoCallDriver(devExt->TopOfStack, Irp);

if (STATUS_PENDING == status) {
    KeWaitForSingleObject(
       &event,
       Executive, // Waiting for reason of a driver
       KernelMode, // Waiting in kernel mode
       FALSE, // No allert
       NULL); // No timeout
}

if (NT_SUCCESS(status) && NT_SUCCESS(Irp->IoStatus.Status)) {
    //
    // As we are successfully now back from our start device
    // we can do work.
    //
    devExt->Started = TRUE;
    devExt->Removed = FALSE;
    devExt->SurpriseRemoved = FALSE;
}

//
// We must now complete the IRP, since we stopped it in the
// completetion routine with MORE_PROCESSING_REQUIRED.
//
Irp->IoStatus.Status = status;
Irp->IoStatus.Information = 0;
IoCompleteRequest(Irp, IO_NO_INCREMENT);

break;
}
```

DDA/x86 Error Trace

```
push    1
push    1
push    1
lea     ecx, dword ptr [ebp + var_1C]
push    ecx
push    sub_1002270
mov     edx, dword ptr [ebp + arg_4]
push    edx
call    dword ptr [_sdv_IoSetCompletionRoutine@24]
mov     edx, dword ptr [ebp + arg_4]
mov     eax, dword ptr [ebp + var_4]
mov     ecx, dword ptr [eax + 8]
call    dword ptr [@IofCallDriver@8]
mov     dword ptr [ebp + var_20], eax
cmp     dword ptr [ebp + var_20], 103h
jnz     loc_10013E1
push    0
push    0
push    0
push    0
lea     ecx, dword ptr [ebp + var_1C]
push    ecx
call    dword ptr [_sdv_KeWaitForSingleObject@20]

loc_10013E1:
cmp     dword ptr [ebp + var_20], 0
jl      loc_1001405
mov     edx, dword ptr [ebp + arg_4]
cmp     dword ptr [edx + 18h], 0
jl      loc_1001405
mov     eax, dword ptr [ebp + var_4]
mov     byte ptr [eax + 30h], 1
mov     ecx, dword ptr [ebp + var_4]
mov     byte ptr [ecx + 32h], 0
mov     edx, dword ptr [ebp + var_4]
mov     byte ptr [edx + 31h], 0

loc_1001405:
mov     eax, dword ptr [ebp + arg_4]
mov     ecx, dword ptr [ebp + var_20]
mov     dword ptr [eax + 18h], ecx
mov     edx, dword ptr [ebp + arg_4]
mov     dword ptr [edx + 1Ch], 0
push    0
mov     eax, dword ptr [ebp + arg_4]
push    eax
call    dword ptr [_sdv_IoCompleteRequest@8]
jmp     loc_10014B0

loc_1001429:
```

SLAM Error Trace

```
status = IoCallDriver(devExt->TopOfStack, Irp);

if (STATUS_PENDING == status) {
    KeWaitForSingleObject(
        &event,
        Executive, // Waiting for reason of a driver
        KernelMode, // Waiting in kernel mode
        FALSE, // No allert
        NULL); // No timeout
}
```

DDA/x86 Error Trace

```
mov     edx, dword ptr [ebp + arg_4]
mov     eax, dword ptr [ebp + var_4]
mov     ecx, dword ptr [eax + 8]
call    dword ptr [@IofCallDriver@8]
mov     dword ptr [ebp + var_20], eax
cmp     dword ptr [ebp + var_20], 103h
jnz     loc_10013E1
push    0
push    0
push    0
push    0
lea     ecx, dword ptr [ebp + var_1C]
push    ecx
call    dword ptr [_sdv_KeWaitForSingleObject@20]
```

SLAM Error Trace

```
if (NT_SUCCESS(status) && NT_SUCCESS(Irp->IoStatus.Status)) {
    //
    // As we are successfully now back from our start device
    // we can do work.
    //
    devExt->Started = TRUE;
    devExt->Removed = FALSE;
    devExt->SurpriseRemoved = FALSE;
}
```

DDA/x86 Error Trace

```
loc_10013E1:
    cmp     dword ptr [ebp + var_20], 0
    jl      loc_1001405
    mov     edx, dword ptr [ebp + arg_4]
    cmp     dword ptr [edx + 18h], 0
    jl      loc_1001405
    mov     eax, dword ptr [ebp + var_4]
    mov     byte ptr [eax + 30h], 1
    mov     ecx, dword ptr [ebp + var_4]
    mov     byte ptr [ecx + 32h], 0
    mov     edx, dword ptr [ebp + var_4]
    mov     byte ptr [edx + 31h], 0
```

SLAM Error Trace

```
//
// We must now complete the IRP, since we stopped it in the
// completetion routine with MORE_PROCESSING_REQUIRED.
//
Irp->IoStatus.Status = status;
Irp->IoStatus.Information = 0;
IoCompleteRequest(Irp, IO_NO_INCREMENT);

break;
```

DDA/x86 Error Trace

```
loc_1001405:
    mov     eax, dword ptr [ebp + arg_4]
    mov     ecx, dword ptr [ebp + var_20]
    mov     dword ptr [eax + 18h], ecx
    mov     edx, dword ptr [ebp + arg_4]
    mov     dword ptr [edx + 1Ch], 0
    push    0
    mov     eax, dword ptr [ebp + arg_4]
    push    eax
    call    dword ptr [_sdv_IoCompleteRequest@8]
    jmp     loc_10014B0
```

Results For *PendedCompletedRequested* Rule

Driver	Procedures	Instructions	⊖ Result	⊖ Feasible Trace?	⊛ reported by Result	⊛ Feasible Trace?	◎ Result	◎ Feasible Trace?	★ Result	★ Feasible Trace?
src/vdd/dosioctl/krnldrvr	70	2824	FP	-	√	-	√	-	√	-
src/general/ioctl/sys	76	3504	FP	-	√	-	√	-	√	-
src/general/tracedrv/tracedrv	84	3719	FP	-	√	-	√	-	√	-
src/general/cancel/startio	96	3861	FP	-	FP	-	√	-	√	-
src/general/cancel/sys	102	4045	FP	-	√	-	√	-	√	-
src/input/moufiltr	93	4175	×	No	×	No	×	No	×	Yes
src/general/event/sys	99	4215	FP	-	√	-	√	-	√	-
src/input/kbfiltr	94	4228	×	No	×	No	×	No	×	Yes
src/general/toaster/toas2tmon	123	6261	FP	-	FP	-	FP	-	√	-
src/storage/filters/diskperf	121	6584	FP	-	FP	-	FP	-	√	-
src/network/modem/fakemodem	142	8747	FP	-	FP	-	FP	-	√	-
src/storage/fdc/flpydisk	171	12752	FP	-	FP	-	FP	-	FP	-
src/input/mouclass	192	13380	FP	-	FP	-	FP	-	FP	-
src/input/mouser	188	13989	FP	-	FP	-	FP	-	FP	-
src/kernel/serenum	184	14123	FP	-	FP	-	FP	-	√	-
src/wdm/1394/driver/1394diag	171	23430	FP	-	FP	-	FP	-	FP	-
src/wdm/1394/driver/1394vdev	173	23456	FP	-	FP	-	FP	-	FP	-

√: passes rule, ×: a real bug found, and FP: False positive.

⊖: A-locs from semi-naïve algorithm

⊛: No GMOD-based merge function

◎: With GMOD-based merge function

★: With cross-product automaton

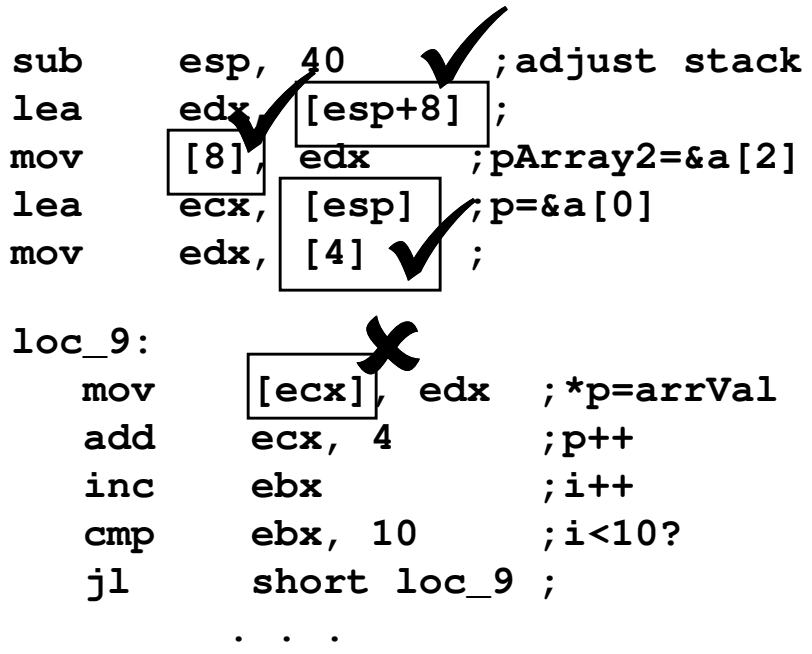
Outline

- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery

- What Next?

Improved A-loc Identification [VMCAI07]

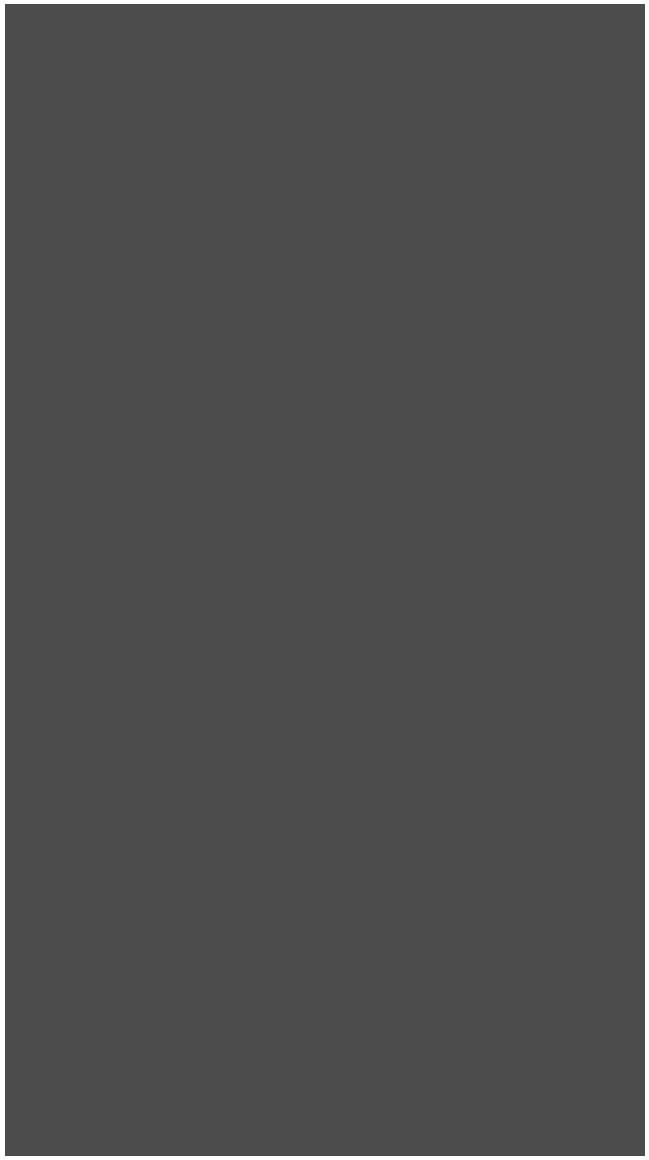
- A-locs from semi-naïve algorithm
  - Based only on explicitly specified addresses/offsets
  - Access based on indirect operands not taken into account



Device Extension Structure for "moufiltr" driver

PDEVICE_OBJECT
PDEVICE_OBJECT
PDEVICE_OBJECT
LONG
PVOID
PI8042_MOUSE_ISR
PI8042_ISR_WRITE_PORT
PVOID
PI8042_QUEUE_PACKET
CONNECT_DATA
DEVICE_POWER_STATE
BOOLEAN
BOOLEAN
BOOLEAN

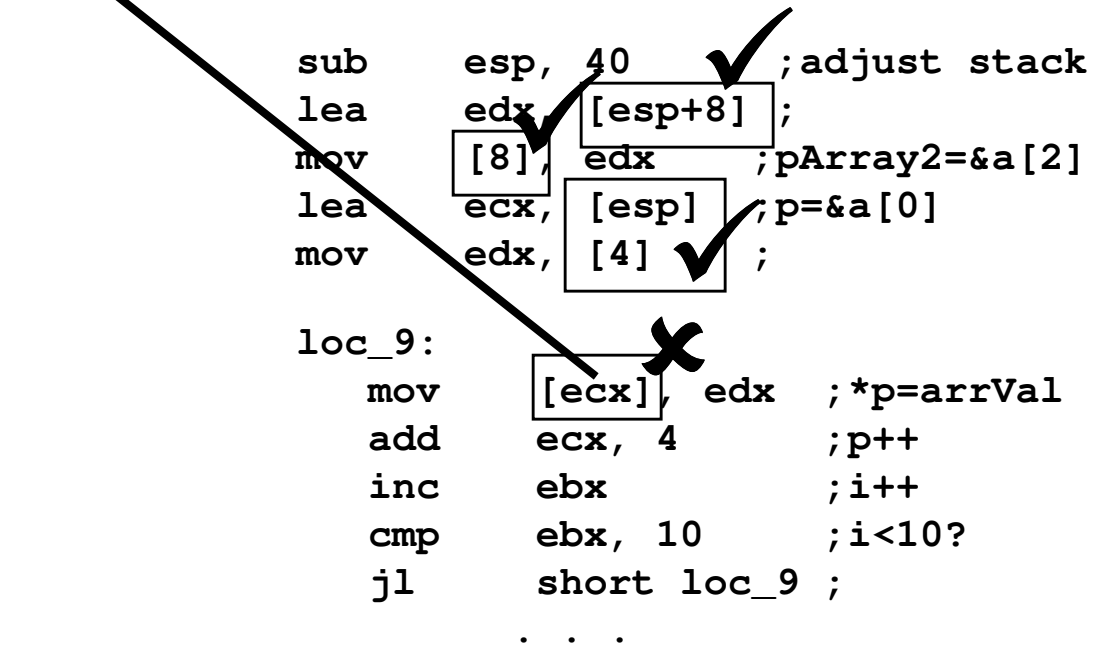
Declaration in C Source



Structure in Executable

Improved A-loc Identification [VMCAI07]

- A-locs from semi-naïve algorithm
  - Based only on explicitly specified addresses/offsets
  - Access based on indirect operands not taken into account
- VSA provides access patterns for indirect operands
  - **ecx** $\rightarrow$ (Ø, 4[-40,-4])
  - 4[0,9]-40 = {-40, -36, . . . , -4}

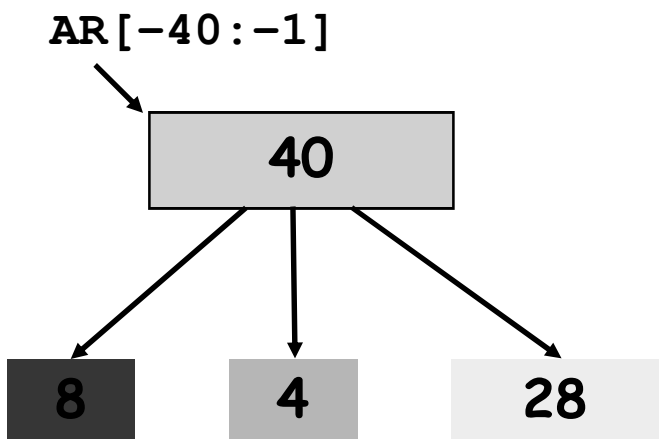


Improved A-loc Identification: VSA+ASI

- Aggregate Structure Identification (ASI)
  - Ramalingam et al. POPL99
  - Partition aggregates automatically
    - based on the program's memory-access patterns
  - Original motivation: Y2K

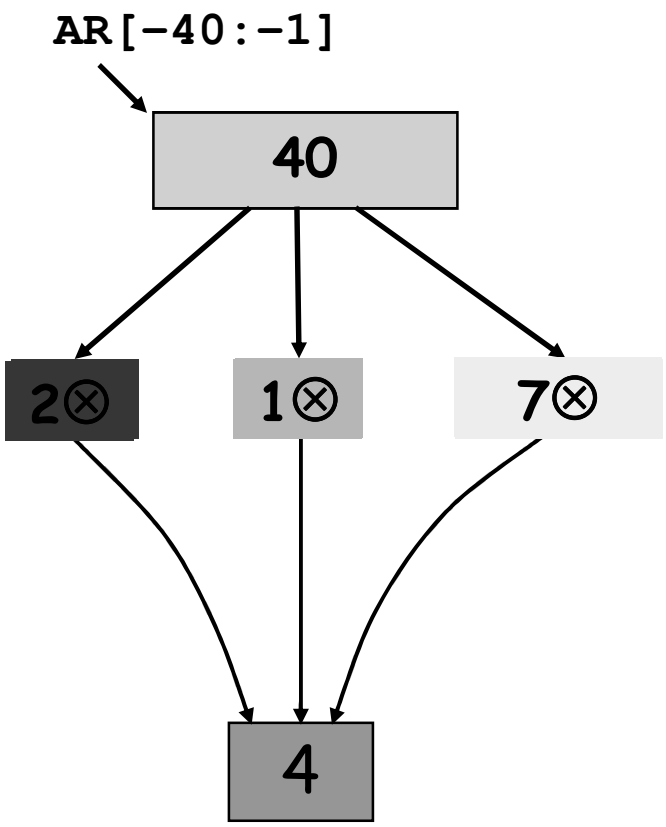
- ASI provides type information
  - Identifies structs and arrays
  - Propagates type information
    - from known parameter types
    - e.g., system calls & library functions

Aggregate Structure Identification



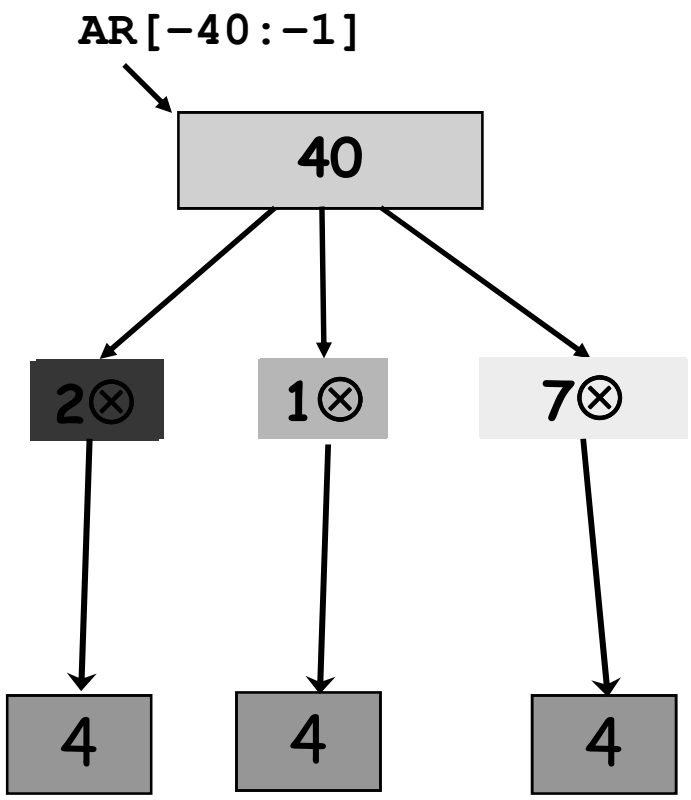

```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, [esp+8]    ;
mov     [4], edx        ;pArray2=&a[2]
lea     ecx, [esp]      ;p=&a[0]
mov     edx, [0]        ;

loc_9:
   mov     [ecx], edx   ;*p=arrVal
   add     ecx, 4       ;p++
   inc     ebx          ;i++
   cmp     ebx, 10      ;i<10?
   jl      short loc_9  ;

mov     edi, [4]     ;
mov     eax, [edi]   ;return *pArray2
add     esp, 40
retn
```

Aggregate Structure Identification




```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, [esp+8]    ;
mov     [4], edx        ;pArray2=&a[2]
lea     ecx, [esp]      ;p=&a[0]
mov     edx, [0]        ;

loc_9:
   mov     [ecx], edx   ;*p=arrVal
   add     ecx, 4       ;p++
   inc     ebx          ;i++
   cmp     ebx, 10      ;i<10?
   jl      short loc_9  ;

mov     edi, [4]     ;
mov     eax, [edi]   ;return *pArray2
add     esp, 40
retn
```

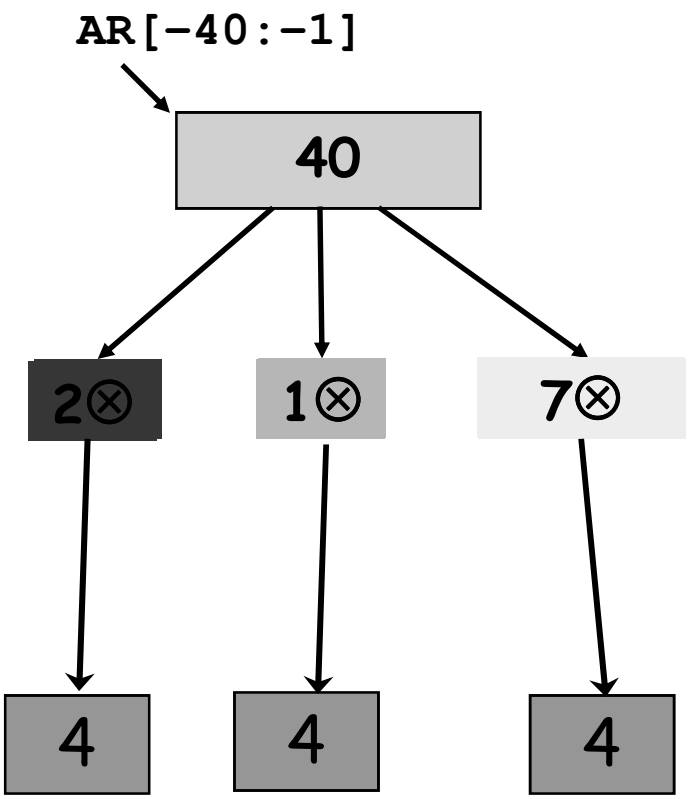
Aggregate Structure Identification

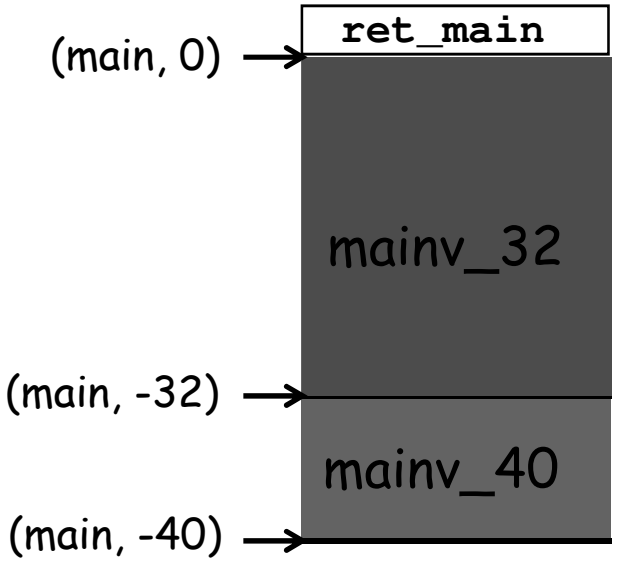


ASI: two arrays; one scalar

```
; ebx ⇔ variable i
; ecx ⇔ variable p

sub     esp, 40         ;adjust stack
lea     edx, [esp+8]    ;
mov     [4], edx        ;pArray2=&a[2]
lea     ecx, [esp]      ;p=&a[0]
mov     edx, [0]        ;

loc_9:
   mov     [ecx], edx   ;*p=arrVal
   add     ecx, 4       ;p++
   inc     ebx          ;i++
   cmp     ebx, 10      ;i<10?
   jl      short loc_9  ;

mov     edi, [4]     ;
mov     eax, [edi]   ;return *pArray2
add     esp, 40
retn
```

Aggregate Structure Identification

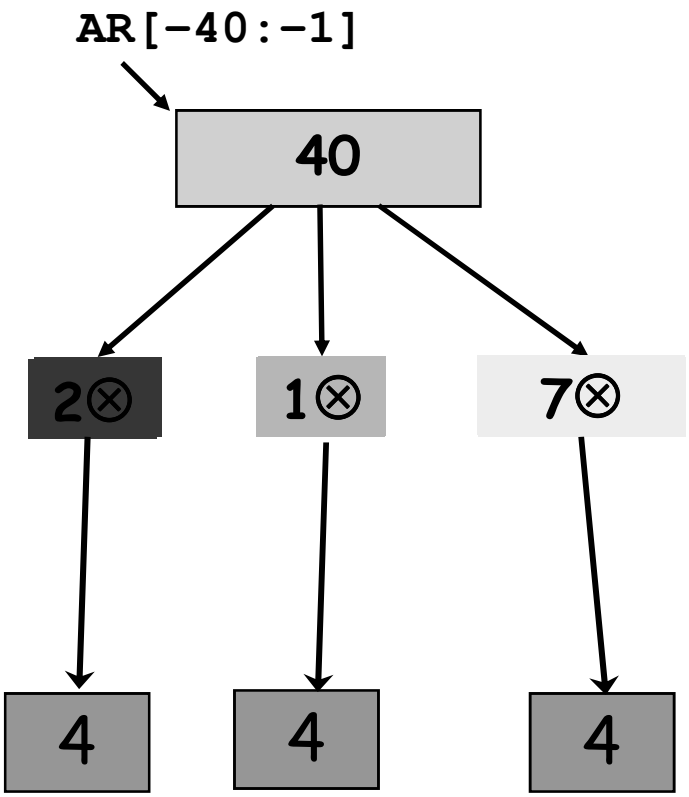




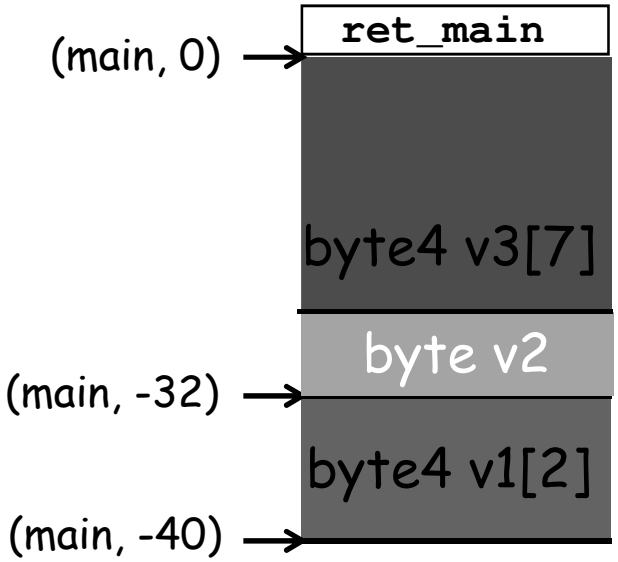
Region for **main**

ASI: two arrays; one scalar

Aggregate Structure Identification



ASI: two arrays; one scalar

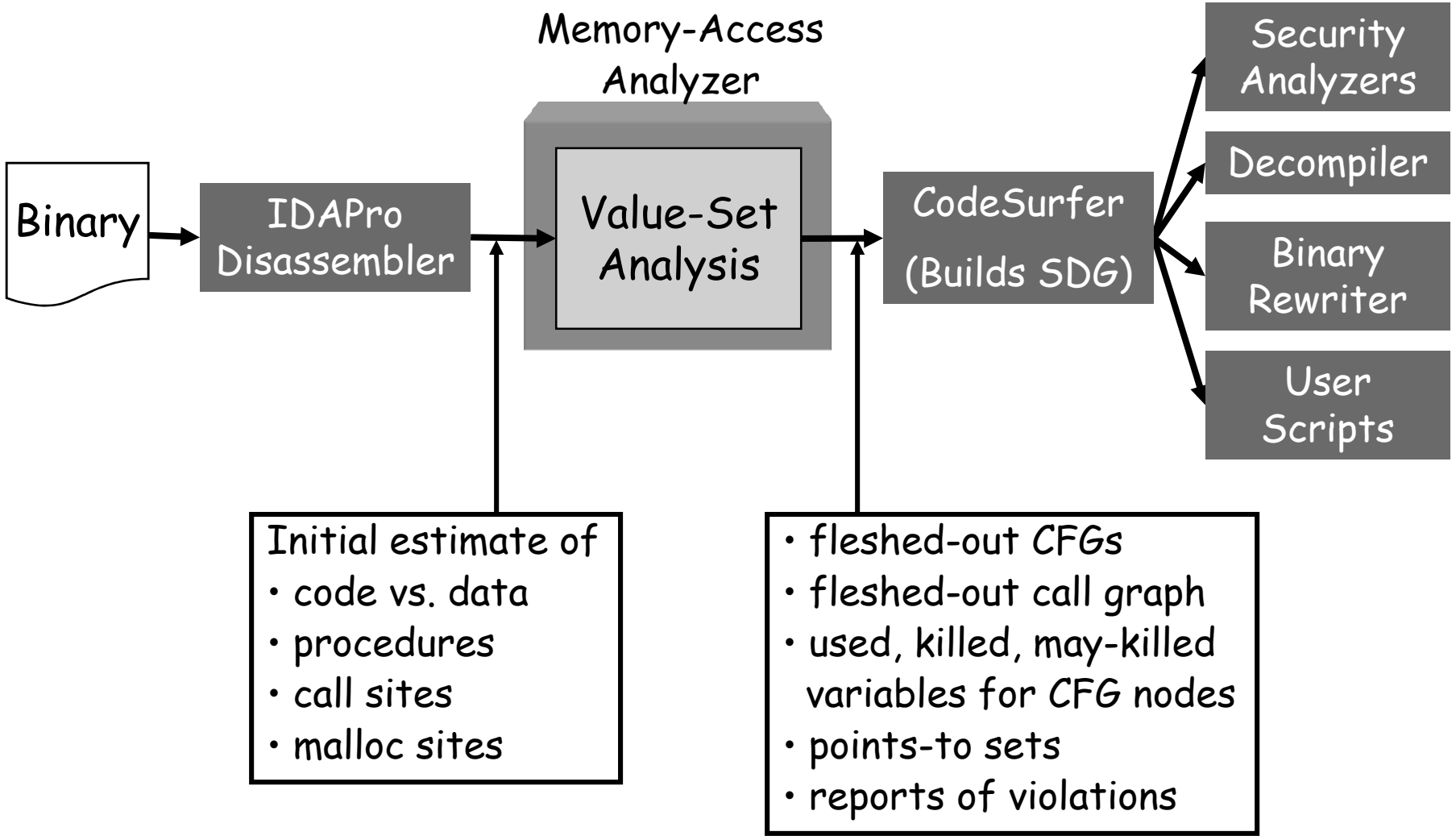


Region for **main**

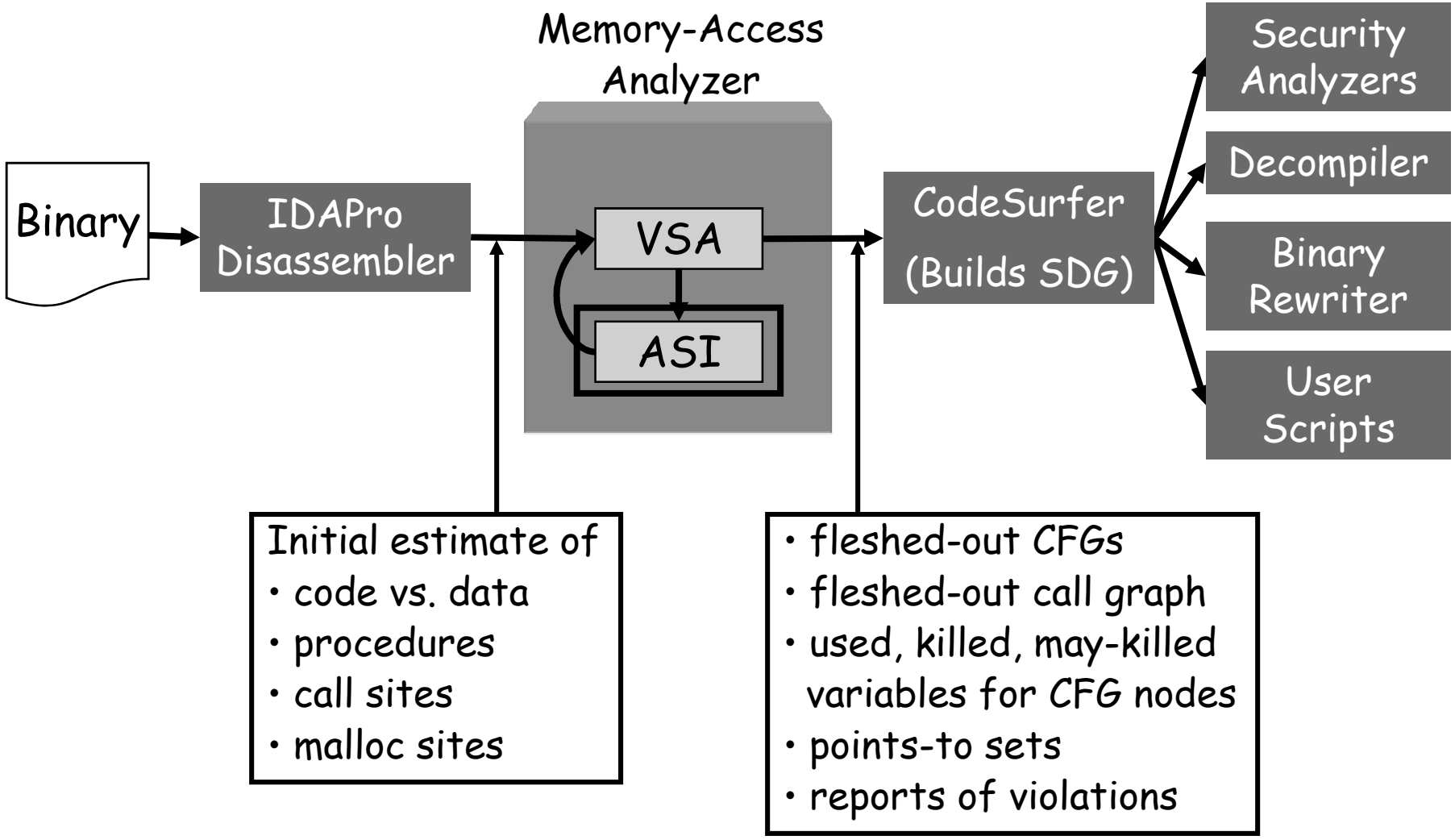
High level type:

```
struct {
   int v1[2];
   int v2;
   int v3[7];
};
```

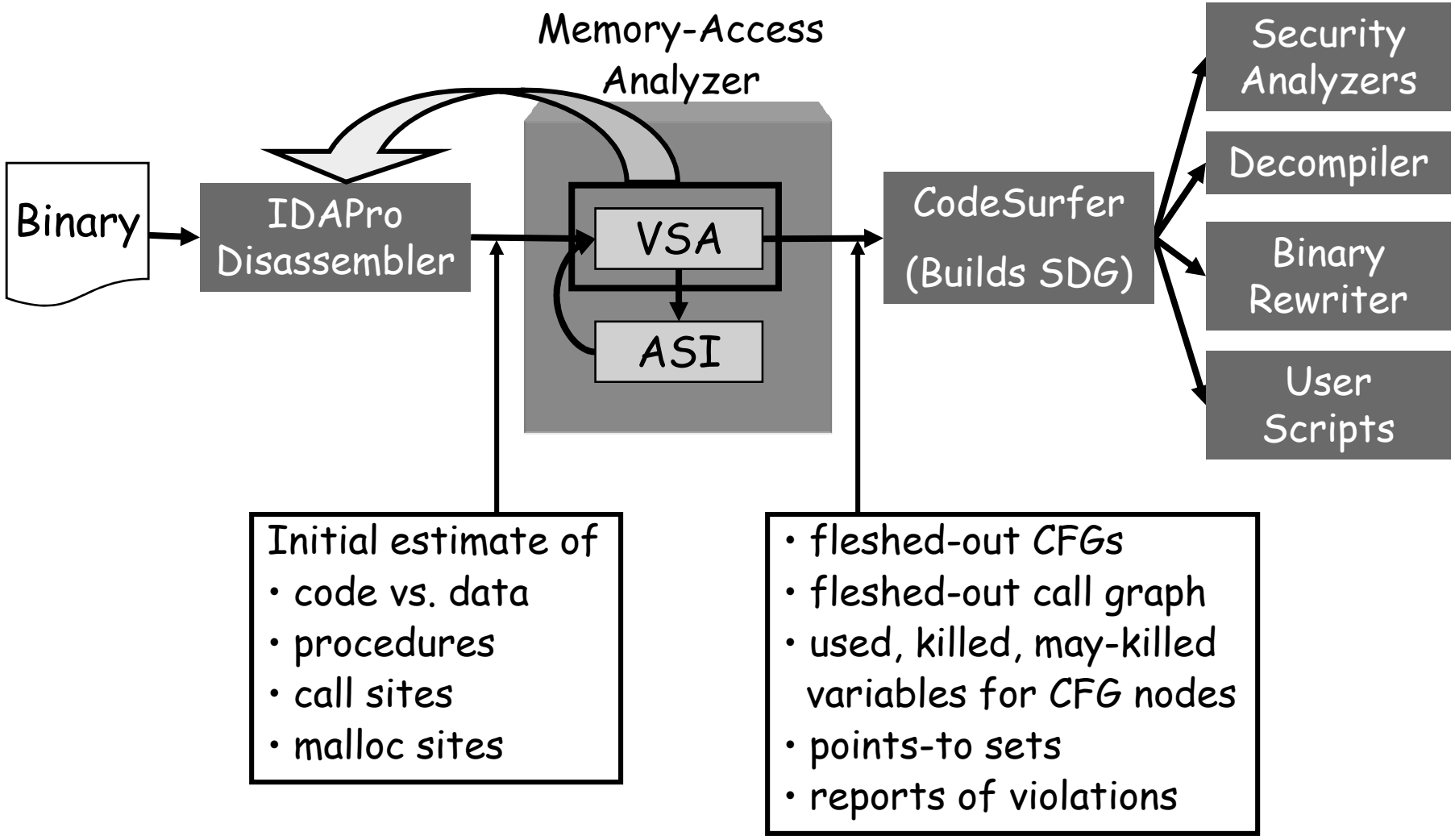
CodeSurfer/x86 Architecture



CodeSurfer/x86 Architecture



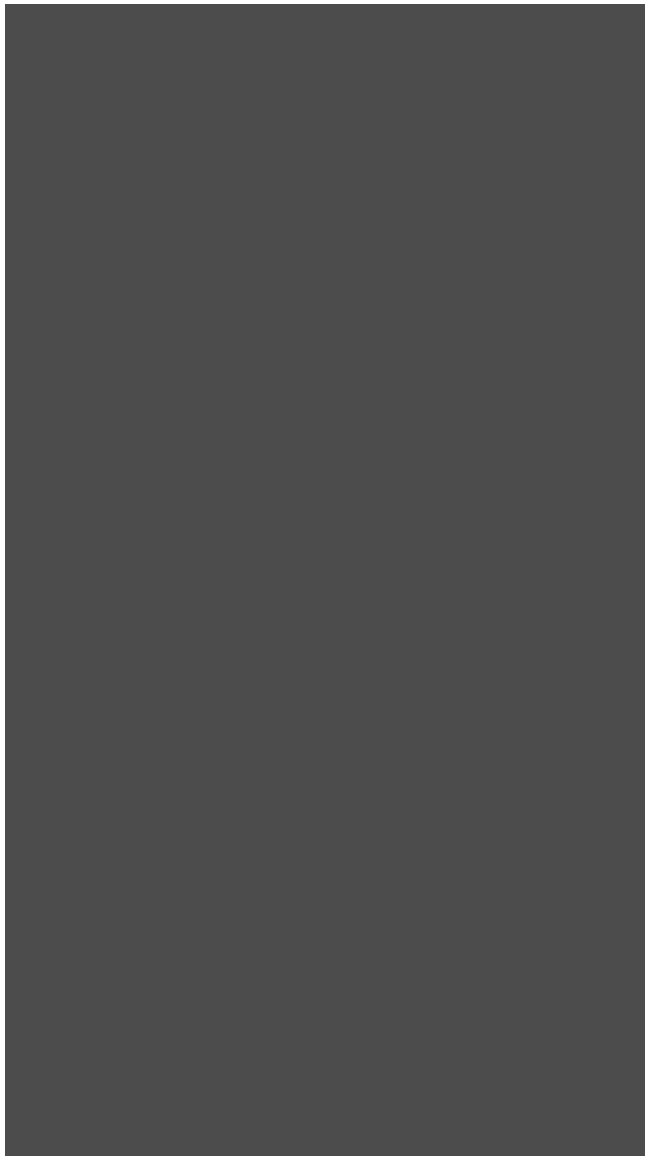
CodeSurfer/x86 Architecture



Device Extension Structure for "moufiltr" driver

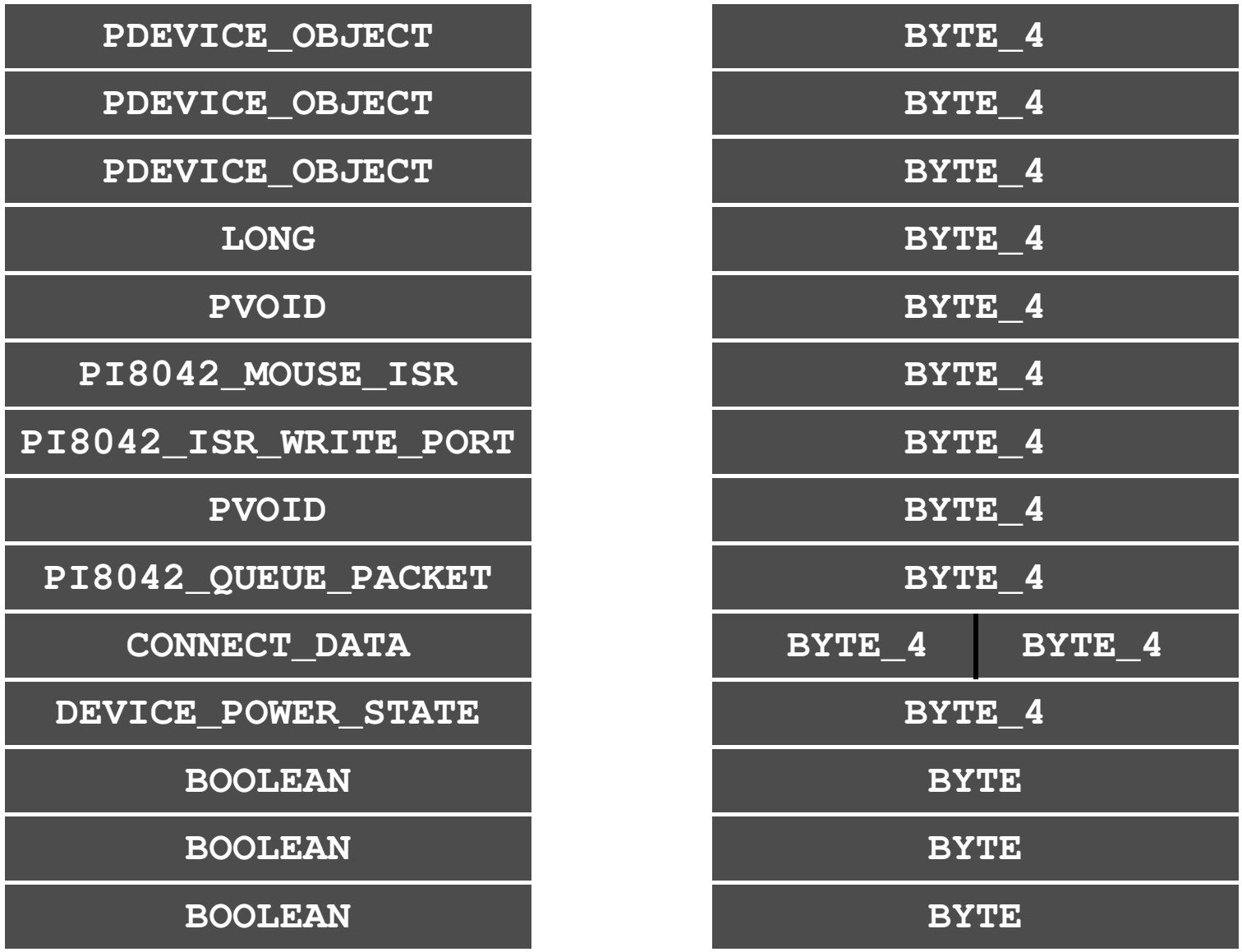
PDEVICE_OBJECT
PDEVICE_OBJECT
PDEVICE_OBJECT
LONG
PVOID
PI8042_MOUSE_ISR
PI8042_ISR_WRITE_PORT
PVOID
PI8042_QUEUE_PACKET
CONNECT_DATA
DEVICE_POWER_STATE
BOOLEAN
BOOLEAN
BOOLEAN

Declaration in C Source



Structure in Executable

Device Extension Structure for "moufiltr" driver



Declaration in C Source

Structure identified by CS/x86

Outline

- CodeSurfer/x86 architecture
- Challenges
- Core Analysis Algorithms
  - Recovering variable-like entities (a-loc)
  - Value-set Analysis (VSA)
  - Other Algorithms
- Device-Driver Analysis
  - Improved a-loc recovery
- What Next?

Other Uses for IR – Early Adopters

- Malicious code analysis - MIT Lincoln Labs
  - Given a worm . . .
    - What are its target-discovery, propagation, and activation mechanisms?
    - What is its payload?

- Extracting file formats from executables
  - J. Lim, T. Reps, B. Liblit (UW) [WCRE06]
  - Based on the file operations in the executable
    - Extract the format of the file being read or written to

- Extracting summaries for library functions
  - D. Gopan and T. Reps (UW) [CAV07]
  - Use information from VSA to extract numeric programs
  - Analyze numeric programs to generate summaries (for memory-safety properties)

What Next?

- Adapt to source code
  - High fidelity information in executables
    - e.g., Buffer Overflow
      - can pin-point the out-of-bounds variable that is touched
  - Useful in source-code analysis
    - e.g., to generate/find non-control data attacks

- Combine dynamic and static techniques
  - Polymorphic viruses and worms
  - Self-modifying code
  - etc.,

What Next?

- Automate Abstraction-Refinement
  - Required a lot of manual effort
    - to reduce false positives in device-driver analysis
  - Automate the process of refinement
    - Lazy Abstraction [Henzinger et al. POPL02]
    - Property simulation [Das et al. PLDI02, SAS06]
  - Seems to be the way to go...

- Lots of other applications
  - Binary-compatibility checking
  - Classifying malware
  - etc.

Summary

- The Vision: A platform for analyzing executables
  - Recover IR from stripped executables
  - Use IR for further analysis

- Why Executables?
  - Source code may not be available
  - Allows analysis of library code
  - Better for security analysis

- How can we build the tool?
  - Value-set analysis (VSA) [CC04]
    - A combined pointer-analysis and numeric-analysis algorithm
  - Algorithm to recover variable-like entities in executables [VMCAI07]
  - Recency-abstraction for heap-allocated data [SAS06]
  - Other analysis
    - ARA, low-level ESP, etc.

G. Balakrishnan and T. Reps, "Analyzing memory accesses in x86 executables," CC 2004, www.cs.wisc.edu/~reps/#cc04

T. Reps, G. Balakrishnan, J. Lim, and T. Teitelbaum, "A next-generation platform for analyzing executables," APLAS 2005, www.cs.wisc.edu/~reps/#aplas05.invited

T. Reps, G. Balakrishnan, and J. Lim, "Intermediate-representation recovery from low-level code," PEPM 2006, www.cs.wisc.edu/~reps/#pepm06.invited

G. Balakrishnan and T. Reps, "Recency-abstraction for heap-allocated storage", SAS 2006, www.cs.wisc.edu/~reps/#sas06-recency

G. Balakrishnan and T. Reps, "DIVINE: DIscovering Variables IN Executables," VMCAI 2007, www.cs.wisc.edu/~reps/#vmcai07.invited

G. Balakrishnan and T.Reps, "Analyzing Stripped Device-Driver Executables," TACAS 2008

WYSINWYX: What You See Is Not What You eXecute
(Static Analysis of Executables)

Gogul Balakrishnan
NEC Laboratories America

Work done while at UW Madison with J.Lim (UW), T. Reps (UW, GrammaTech), T. Teitelbaum (Cornell, GrammaTech)