Infrastructure for Analysis of Object Code

Gogul Balakrishnan

University of Wisconsin

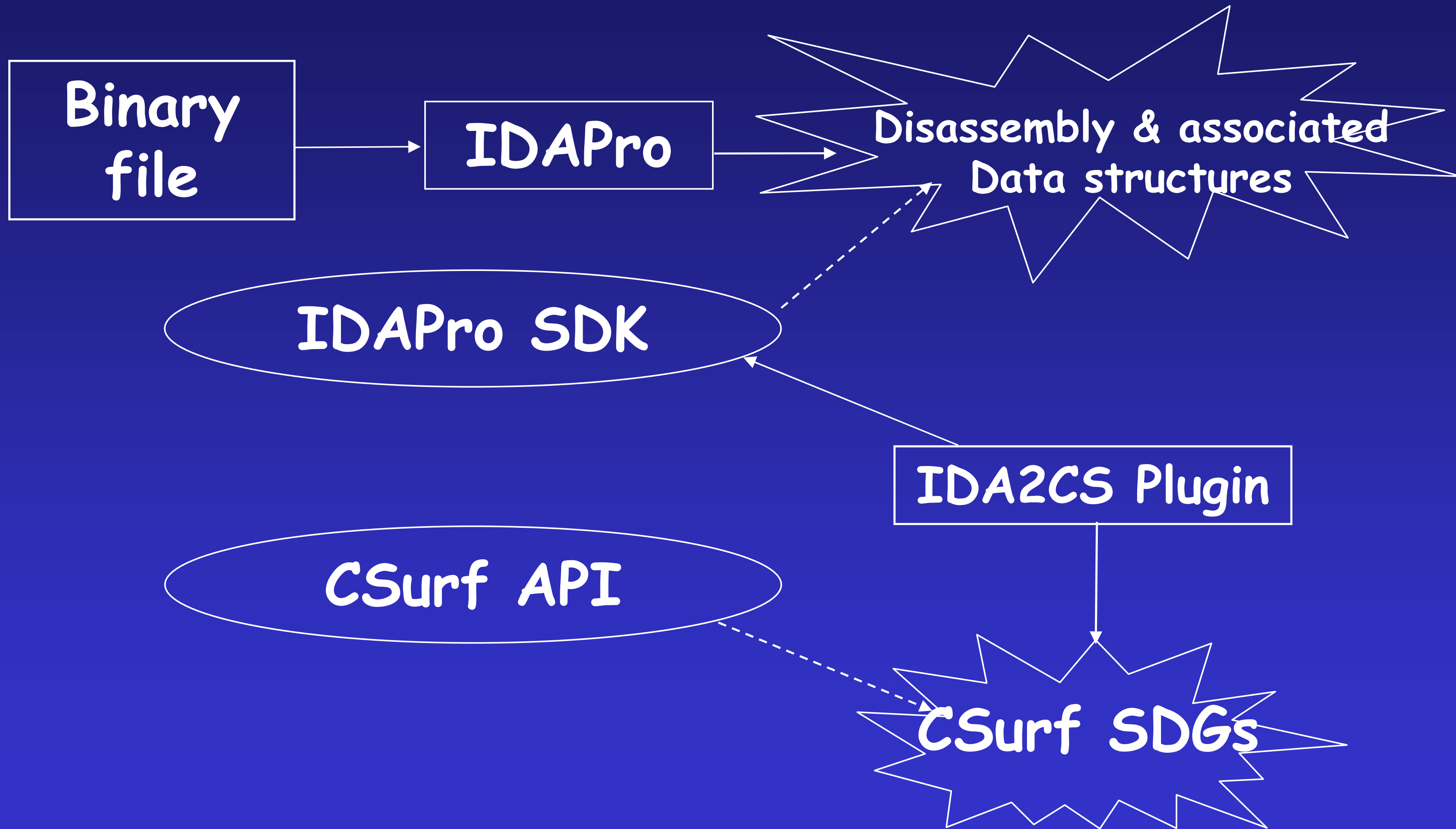
Overview

- About the existing infrastructure
  - IDAPro
  - CodeSurfer
- Extensions to CodeSurfer
  - Basic blocks
  - Templates for data flow analysis
- Points-to analysis on assembly code

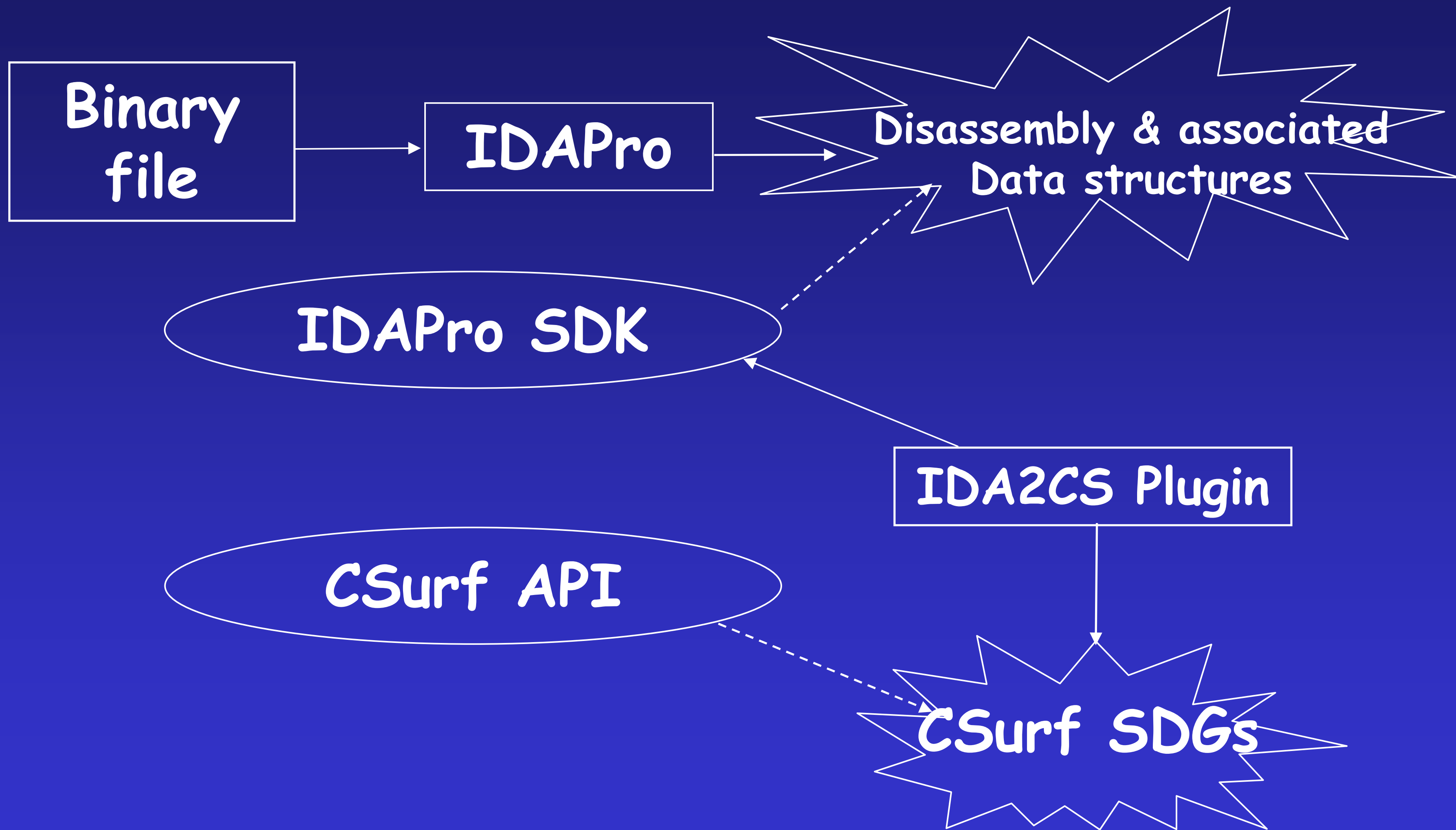
Infrastructure

- Use existing analysis software
- Augment them with features for analyzing object code
- IDA Pro (DataRescue)
- Codesurfer (GrammaTech)

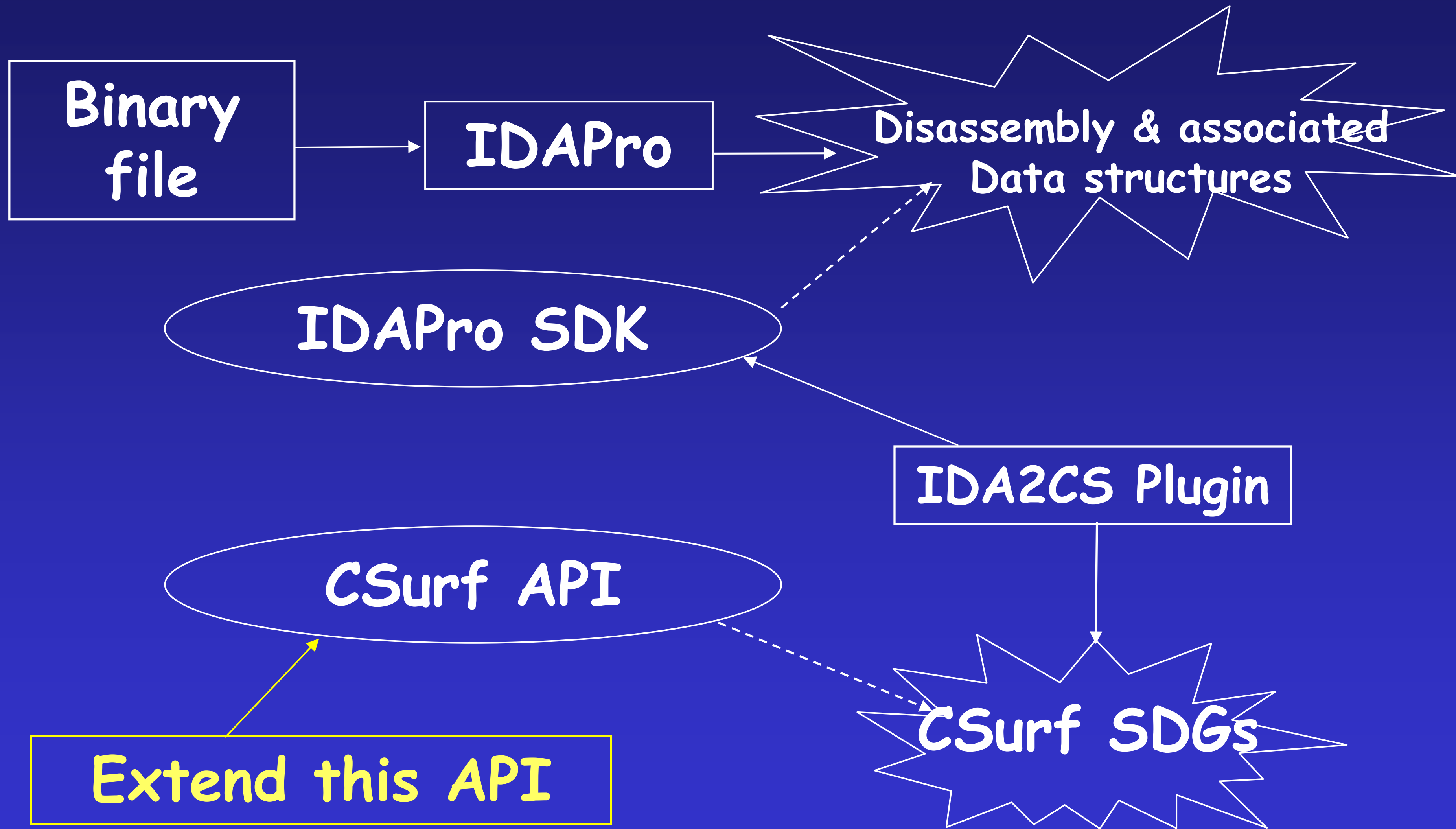
Codesurfer and IDAPro



Codesurfer and IDAPro



Codesurfer and IDAPro



Basic Blocks

```
sub_10018A0     proc near               ; CODE XREF: sub_1001AE3+445  p
                                ; _WinMain@16+CF  p
          mov     eax, hMem
          push    esi
          mov     esi, ds:GlobalFree
          test    eax, eax
          jz      short loc_10018B3
          push    eax             ; hMem
          call    esi ; GlobalFree

loc_10018B3:                            ; CODE XREF: sub_10018A0+E  j
          mov     eax, dword_1008BEC
          test    eax, eax
          jz      short loc_10018BF
          push    eax             ; hMem
          call    esi ; GlobalFree
loc_10018BF:                            ; CODE XREF: sub_10018A0+1A  j
          and     hMem, 0
          and     dword_1008BEC, 0
          pop     esi
          retn
```

Construct Basic Blocks

Data flow analysis templates - Live Variable Analysis

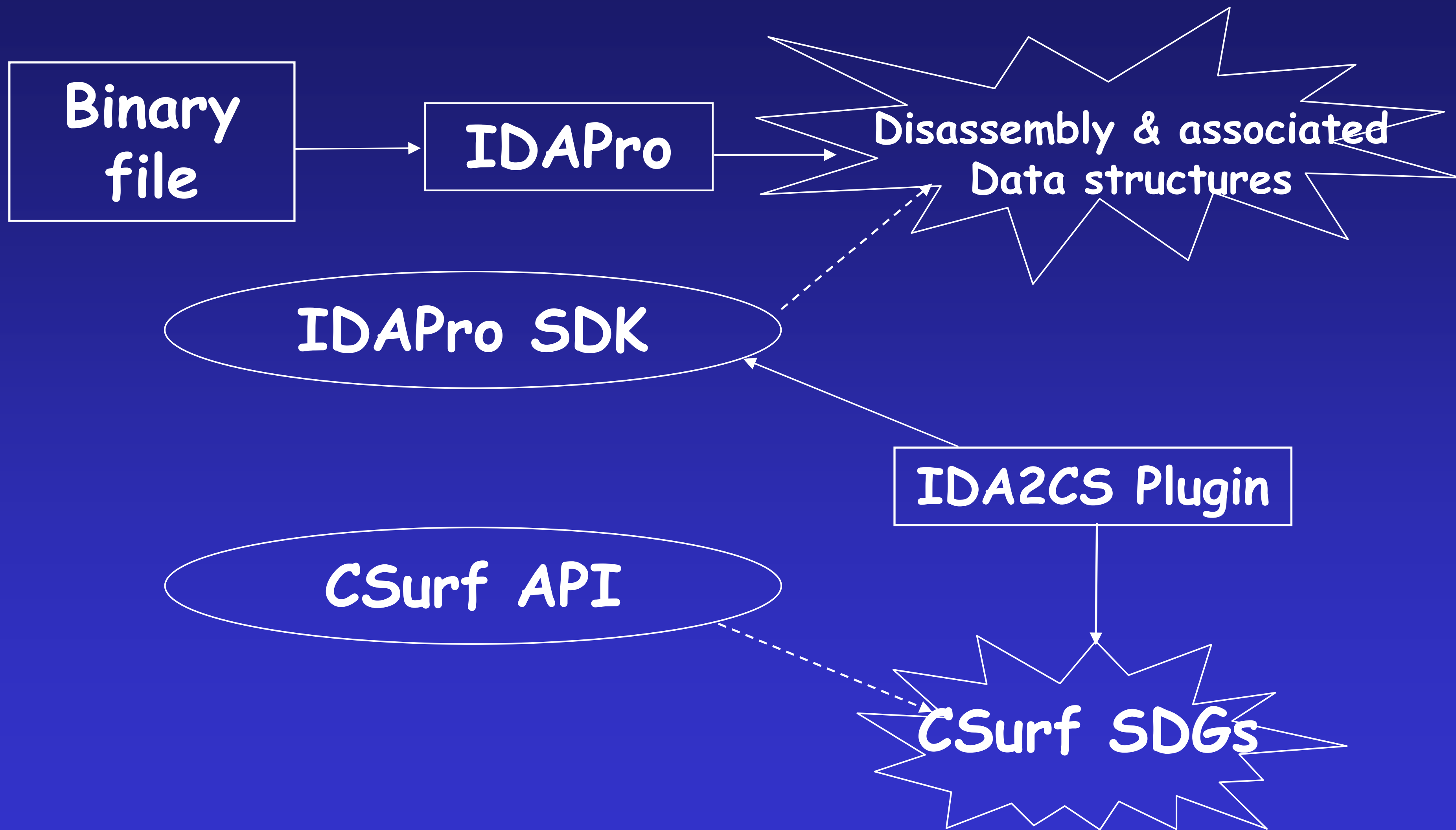
```
 /* Demo CodeSurfer */
#include <stdio.h>
#include "hello.h"

static int debug = 0;

void main(void)
{
 long f,i,n,j,k;
 j=1;  i=1;
 f=1; n=10;
 k=j*2;
 a: f=f*i;
 i=i+1;
 if(i<=n) goto a;
}
```

Demo

Overview

- About the existing infrastructure
  - IDAPro
  - CodeSurfer
- Extensions to CodeSurfer
  - Basic blocks
  - Templates for data flow analysis
- Points-to analysis on assembly code

Codesurfer and IDAPro



Need for points-to analysis

- Better understanding of program behavior

```
main( ){
    int c,b=10,a=20,*pa=&a;
    c=*pa+b;
}
```

Need for points-to analysis

- Better understanding of program behavior
- Pointers – a possible covert channel

```
main( ){
    int b,a=20,*pa=&a;
    b=*pa;
}
```

'a' – High security level variable

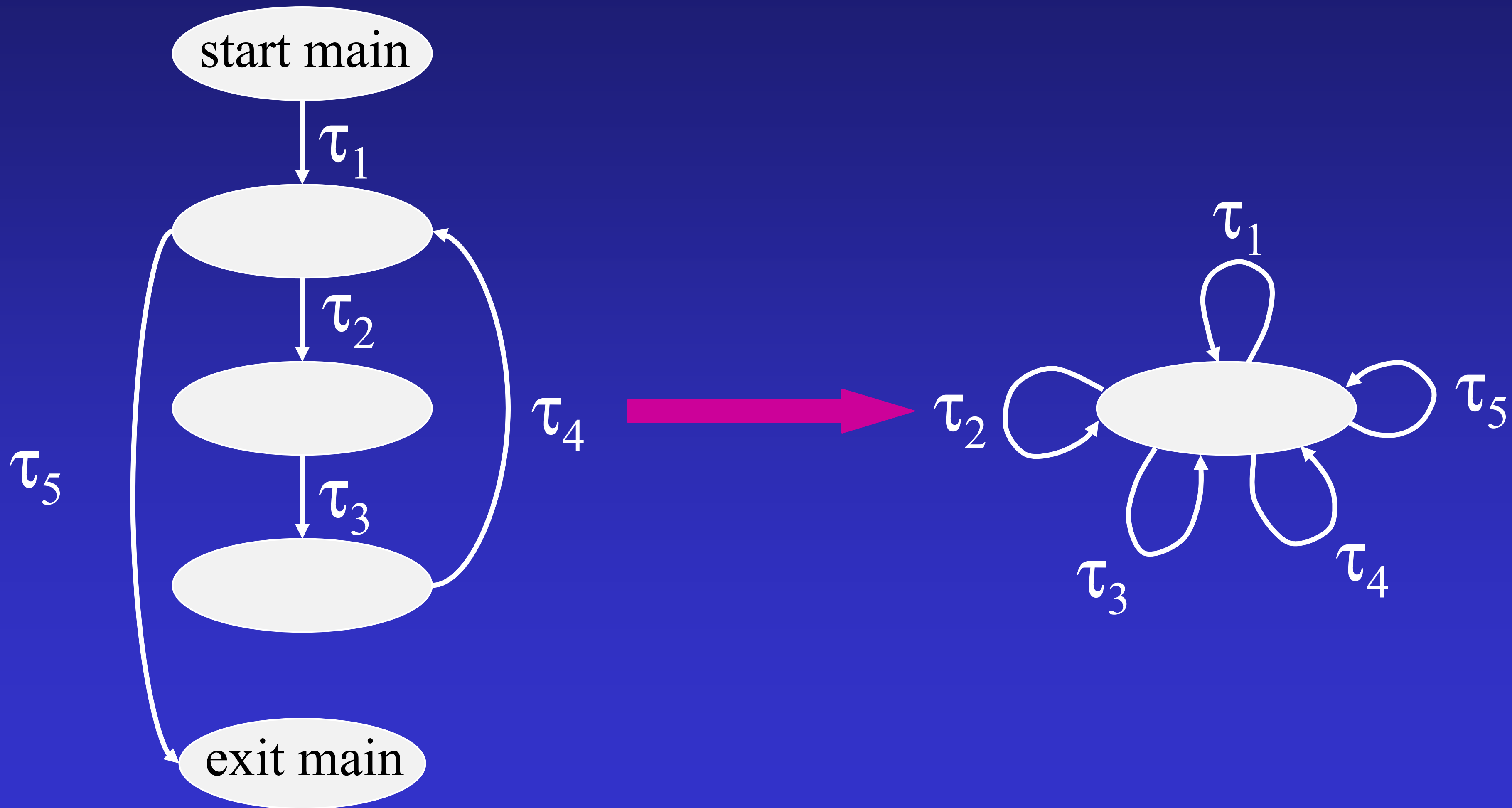
'b' – Low security level variable

Need for points-to analysis

- Better understanding of program behavior
- Pointers – a possible covert channel
- Can detect more buffer overruns

```
void main(){
    char str[5],*a;
    a=str;
    strcpy(a,"Hello");
}
```

Flow-Sensitive → Flow-Insensitive



Flow Sensitive vs. Flow Insensitive

- Flow-insensitive analysis
  - Less precise
  - More efficient in space and time
  - But works poorly with assembly code
    - Register ebp used to access all variables
      $\Rightarrow$ All variables treated as one
- Recover a degree of flow sensitivity
  - Rename registers according to live ranges
  - Perform flow-insensitive analysis

Live Ranges

```
push ebp
mov ebp, esp
lea eax, [ebp-4]
mov ebx, [eax]
mov ecx, ebx
mov edx, eax
mov ebx,edx
lea eax, [ebp-8]
mov ebx, ecx
mov esi,eax
pop ebp
```

:LiveRange 0 of eax

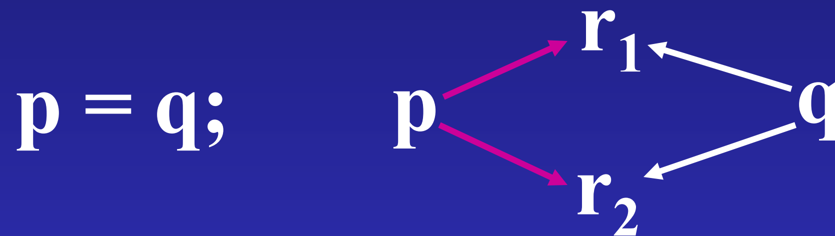
:LiveRange 1 of eax

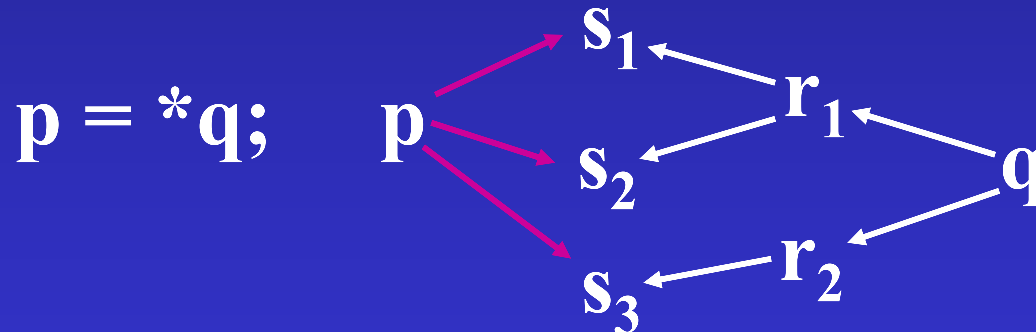
:LiveRange 0 of ecx

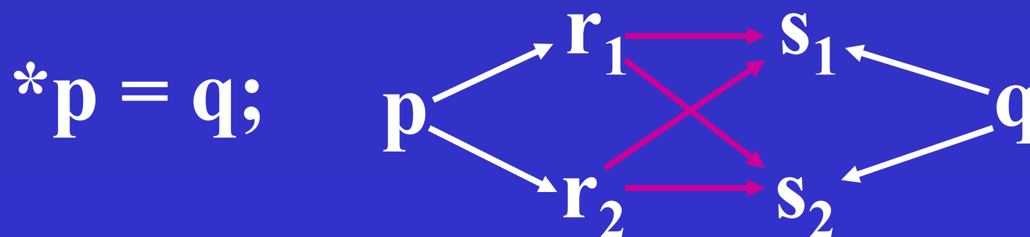
Flow-**Insensitive** Points-to Analysis

[Andersen 94, Shapiro & Horwitz 97]

p = &q; p → q

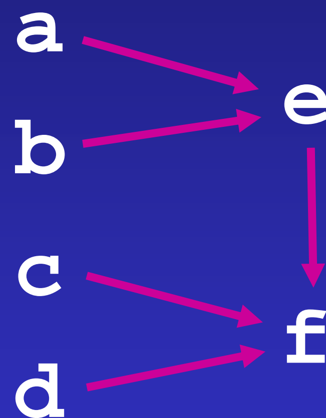






Flow-Insensitive Points-To Analysis

```
 a = &e;
 b =  a;
 c = &f;
*b =  c;
 d = *a;
```



What are the entities?

- Two kinds of storage areas
  - Registers
  - Memory
- Possible points-to relations
  - Registers → Memory
  - Memory → Memory
  - Registers → Registers <u>not</u> possible
  - Memory → Registers <u>not</u> possible

What are the abstract variables?

- Memory
  - Register + Displacement
    - E.g., mov ebx,[ebp-4]
  - Displacement
    - E.g., mov ebx,[12]

∴ Abstract variables have 2 components

- [Optional] Register name
- Displacement

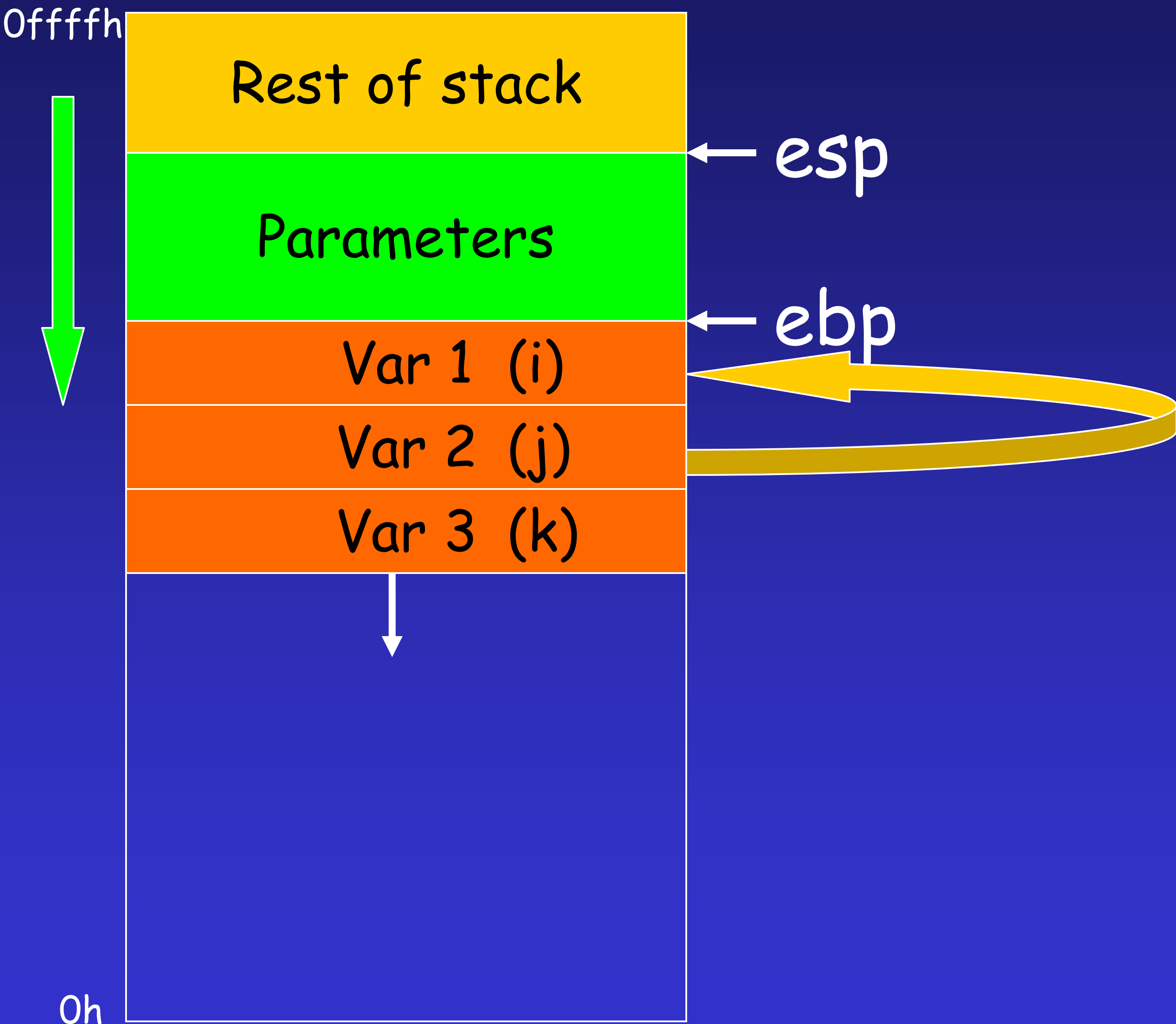
What are the abstract variables?

Really two kinds of abstract entities

- Addresses: A_reg_lr_Displ
- Memory Locations: M_reg_lr_Displ
- e.g.,
  - A_ebp_0_4 for ebp_0 - 4
  - M_ebp_0_4 for [ebp_0 - 4]

- Implicit points-to facts
  - A_reg_lr_D → M_reg_lr_D

Memory model




```
fn(..) {
 int i,*j,k;
    j=&i;
}
```

Points-to analysis on assembly code

```
void main() {
        int i,*j;
        j=&i;
        return ;
}
```

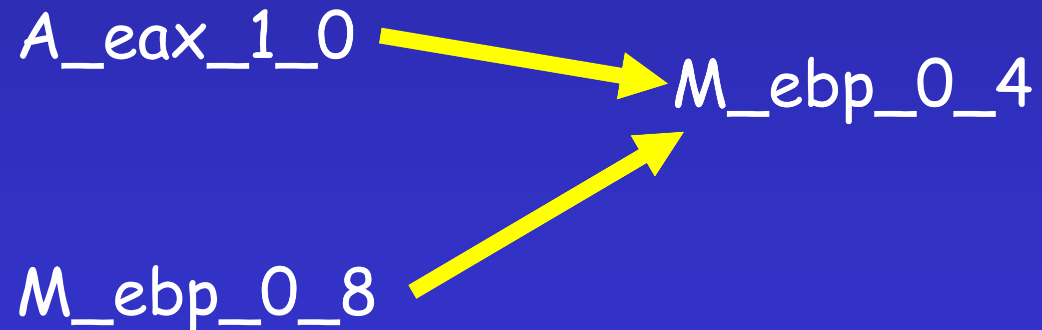
```
push    ebp
mov     ebp, esp
sub     esp,8
lea     eax, [ebp-4]
mov     [ebp-8], eax
mov     esp, ebp
pop     ebp
retn
```

Points-to analysis on assembly code

Equivalent statements:

- A_eax_1_0 = &M_ebp_0_4
- M_ebp_0_8 = A_eax_1_0

```
push    ebp
mov     ebp, esp
sub     esp,8
lea     eax_1, [ebp_0-4]
mov     [ebp_0-8], eax_1
mov     esp, ebp
pop     ebp
retn
```



Points-to analysis on assembly code

0ffffh

Rest of stack

Parameters

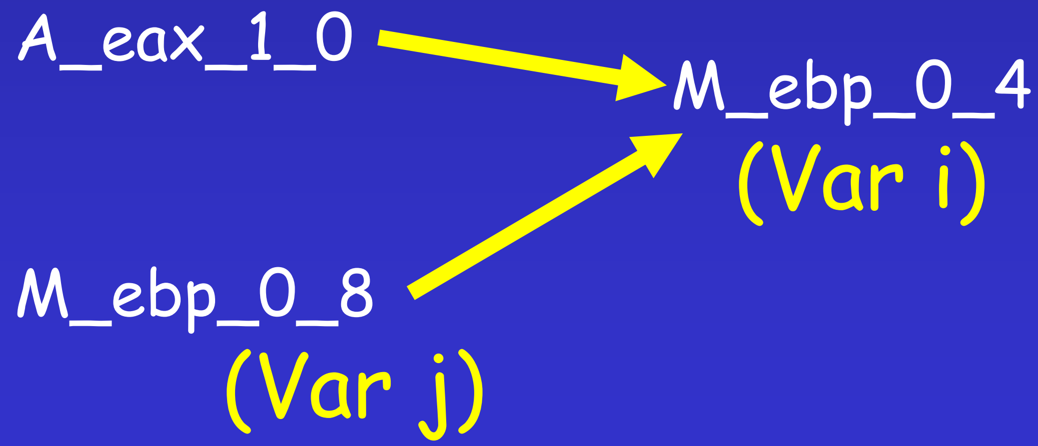
Var 1 (i)

Var 2 (j)

0h

ebp_0

```
push    ebp
mov     ebp, esp
sub     esp,8
lea     eax_1, [ebp_0-4]
mov     [ebp_0-8], eax_1
mov     esp, ebp
pop     ebp
retn
```



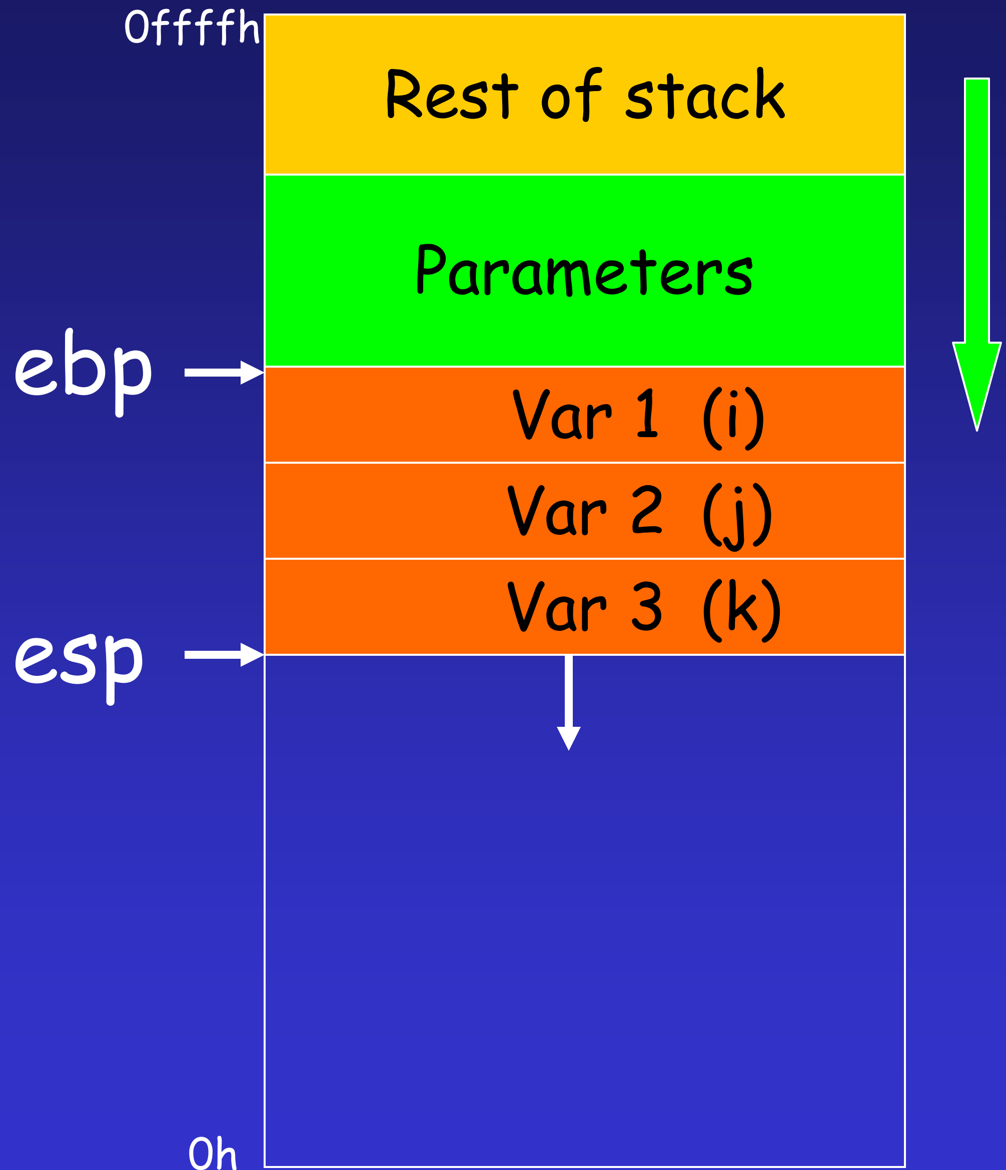
Points-to analysis on assembly code

```
push   ebp
mov    ebp, esp
sub    esp, -12
push   ebp
lea    eax, [ebp-4]
mov    ebp, eax
lea    ecx, [ebp-4]
mov    [ebp-8], ecx
pop    ebp
mov    esp,ebp
pop    ebp
retn
```

```
void main(){
        int i,*j,*k;
        _asm {
                push ebp
                lea eax,[ebp-4]
                mov ebp,eax
        }
        j=&i;
        _asm {
                pop ebp
        }
        return ;
}
```

Points-to analysis on assembly code

```
push    ebp
mov     ebp, esp
sub     esp, -12
push    ebp
lea     eax, [ebp-4]
mov     ebp, eax
lea     ecx, [ebp-4]
mov     [ebp-8], ecx
pop     ebp
mov     esp,ebp
pop     ebp
retn
```



Points-to analysis on assembly code

```
push    ebp
mov     ebp, esp
sub     esp, -12
push    ebp
lea     eax, [ebp-4]
mov     ebp, eax
lea     ecx, [ebp-4]
mov     [ebp-8], ecx
pop     ebp
mov     esp,ebp
pop     ebp
retn
```

```
void main(){
        int i,*j,*k;
        _asm {
                push ebp
                lea eax,[ebp-4]
                mov ebp,eax
        }
        j=&i;          (k=&j;)
        _asm {
                pop ebp
        }
        return ;
}
```

Points-to analysis on assembly code

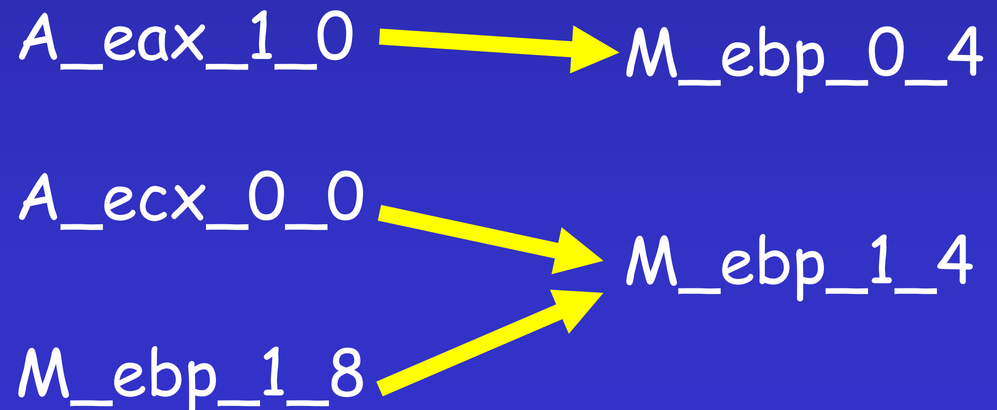
```
push   ebp
mov    ebp, esp
sub    esp, -12
push   ebp
lea    eax, [ebp-4]
mov    ebp, eax
lea    ecx, [ebp-4]
mov    [ebp-8], ecx
pop    ebp
mov    esp,ebp
pop    ebp
retn
```

Points-to analysis on assembly code

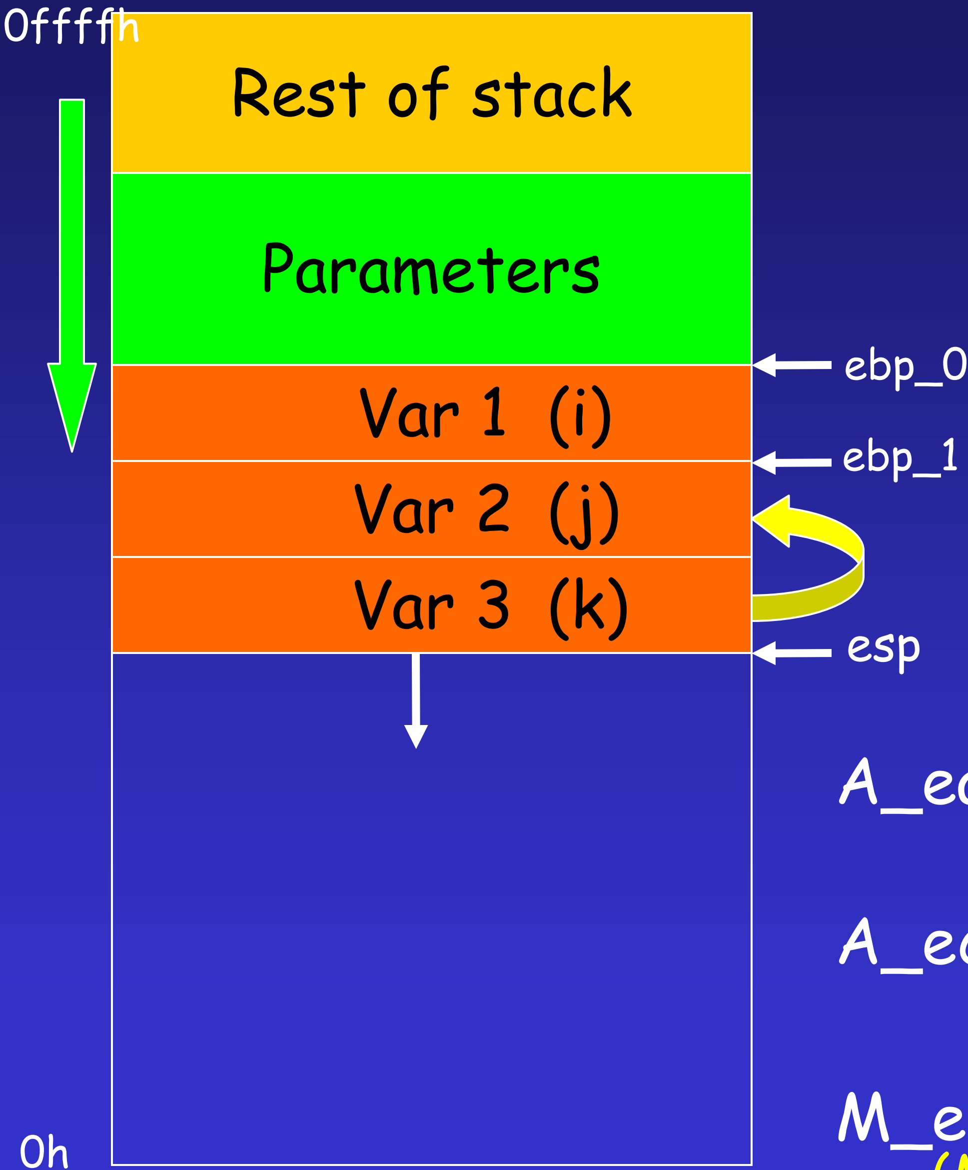
```
push   ebp
mov    ebp, esp
sub    esp, -12
push   ebp
lea    eax_1, [ebp_0-4]
mov    ebp_1, eax_1
lea    ecx_0, [ebp_1-4]
mov    [ebp_1-8], ecx_0
pop    ebp
mov    esp,ebp
pop    ebp
retn
```

Equivalent statements:

- A_eax_1_0 = &M_ebp_0_4
- ∀i, A_ebp_1_i = A_eax_1_i
- A_ecx_0_0=&M_ebp_1_4
- M_ebp_1_8=A_ecx_0_0



Points-to analysis on assembly code



- A_eax_1_0 = &M_ebp_0_4
  - ⇒ A_eax_1_0 = A_ebp_0_4
- ∀i, A_ebp_1_i = A_eax_1_i

∴ ebp_1 = ebp_0 - 4

- M_ebp_1_4 ⇔ M_ebp_0_8
- M_ebp_1_8 ⇔ M_ebp_0_12

A_eax_1_0 → M_ebp_0_4

A_ecx_0_0 → M_ebp_1_4 (M_ebp_0_8)

M_ebp_1_8 (M_ebp_0_12) → M_ebp_1_4 (M_ebp_0_8)

Future Work

- Complete implementation of flow-insensitive points-to analysis
- Use flow-insensitive analysis to reduce work for flow sensitive analysis
- Transforming machine instructions to C like expressions

```
lea eax,[ebp-4]
mov ebx,[ebp-8]        M_ebp_0_8= *A_ebp_0_4 + M_ebp_0_8
mov ecx,[eax]
add ebx,ecx
mov [ebp-8],ebx
```