Preparing Object Code for Static Analysis

Gogul Balakrishnan
University of Wisconsin-Madison

Need for preprocessing

Data Dependence Analysis

```
int main(){
   int i,j, a[10];
   j=0;
   for(i=0;i<10;++i){
       a[i]=i;
   }
   return j;
}
```

Affects?
No!

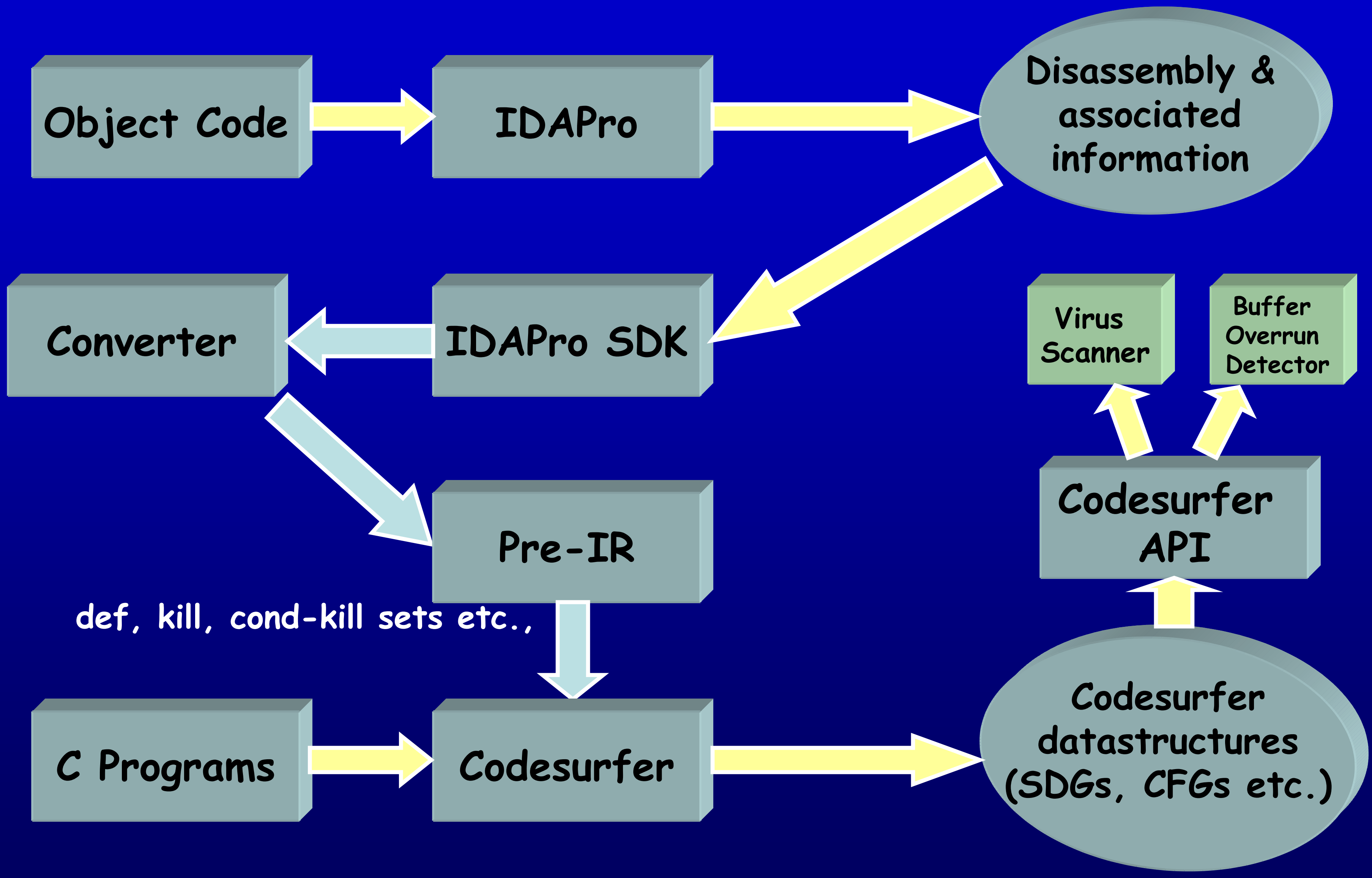
```
; ebx corresponds to variable i
sub      esp, 44
mov      [esp+40],0    ; j = 0
xor      ebx, ebx      ; i = 0
lea      ecx, [esp]
loc_9:
   mov      [ecx], ebx ; a[i]=i
   inc      ebx         ; i++
   add      ecx, 4
   cmp      ebx, 10     ; i<10?
   jl       short loc_9 ;
mov      eax, [esp+40] ; return j
add      esp, 44
retn
```

Affects??

Need for preprocessing

- In high level language programs
  - Variables
    - An abstract entity for memory
    - The entities on which the algorithm operates
    - We have a finite domain to operate on
- In object code
  - No properly defined entities
  - Has to be inferred

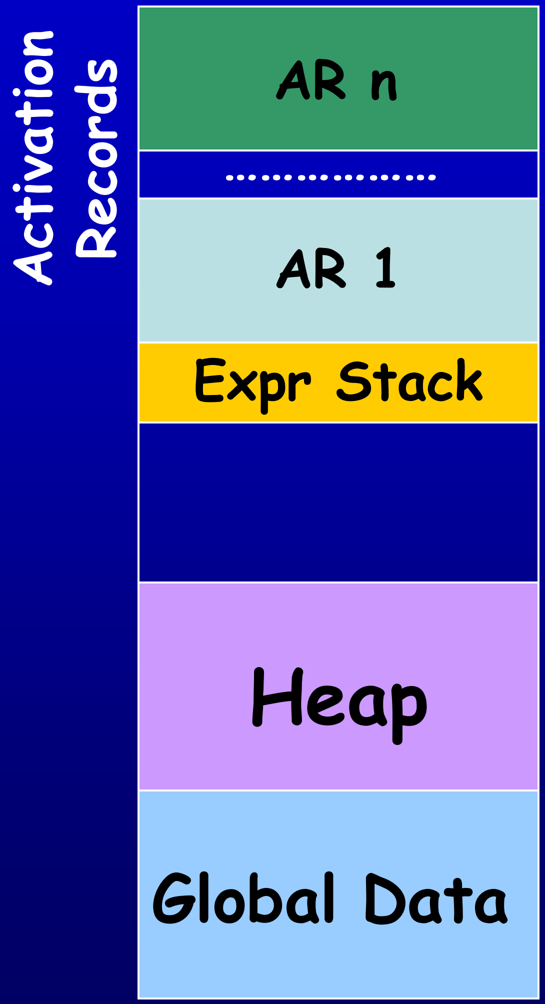
Existing infrastructure



Our Goal

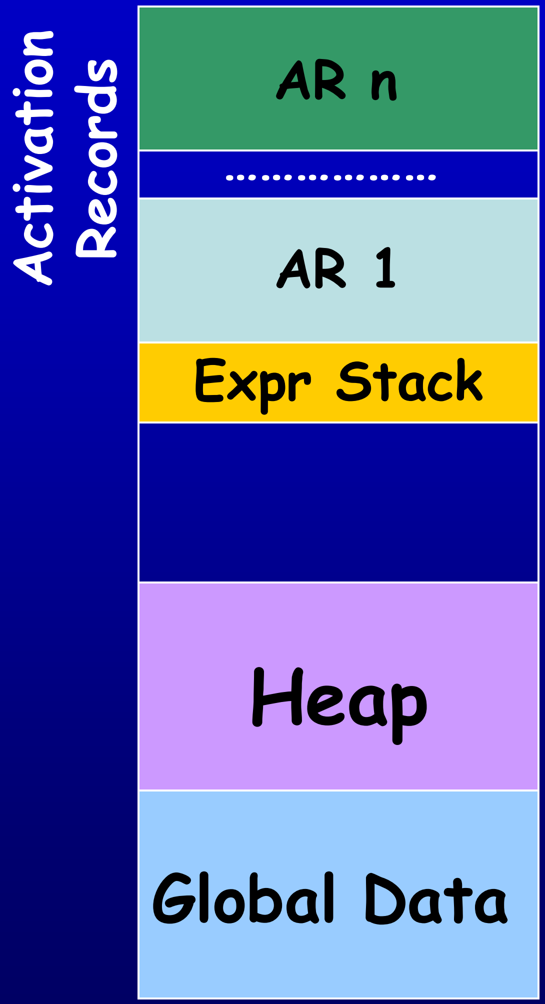
- Discover the entities
- Annotate each program statement with def, kill and conditionally kill sets
- Feed it to Codesurfer
  - Already has a lot of static analysis algorithms implemented – Slicing, Chopping etc.,
- Can benefit
  - Virus scanner (Mihai)
  - Buffer Overrun Detector (Vinod)

Memory Model



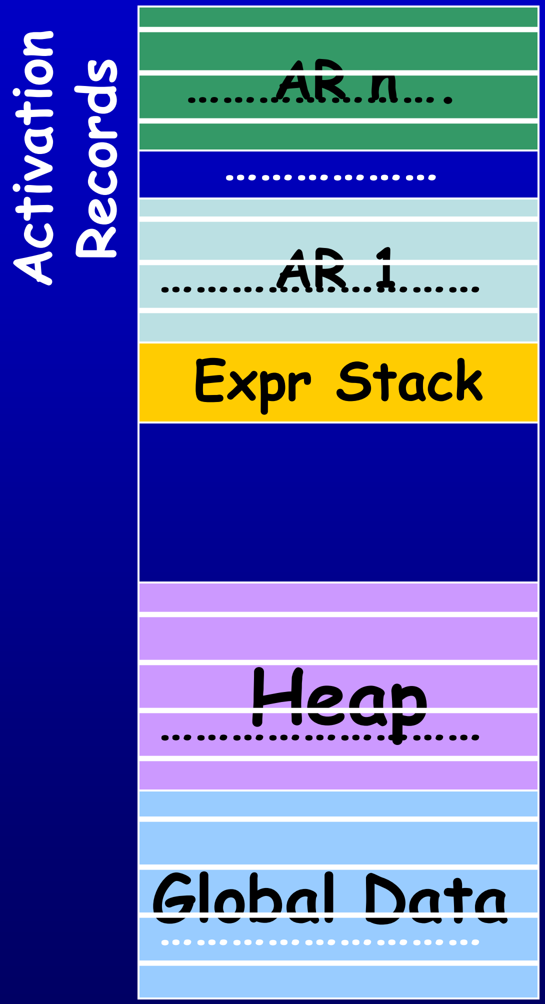
- Four areas
  - Activation record
  - Global data
  - Heap
  - Expression stack
- Assumed to be disjoint
- Assumption should be validated

What are the entities?



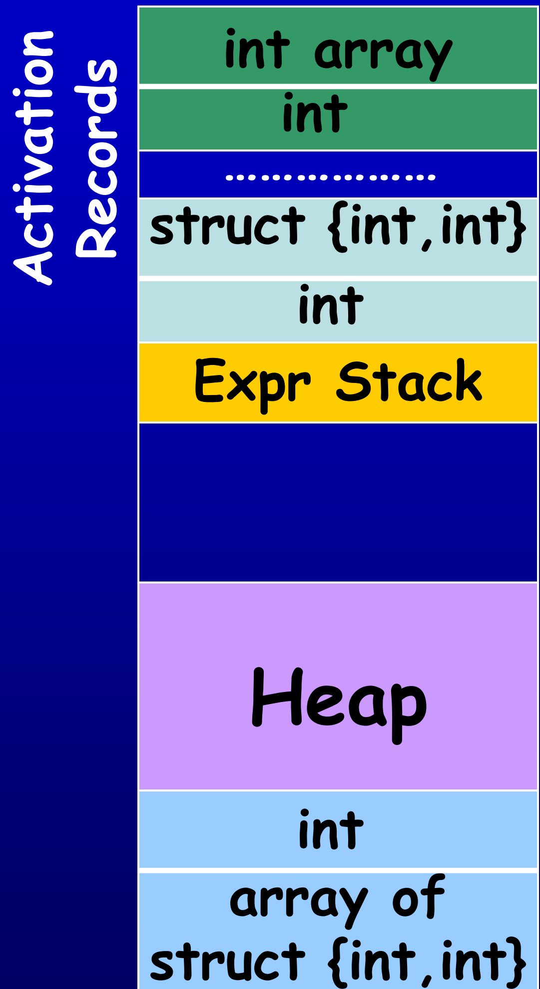
- Each area an entity?
  - Too inaccurate

What are the entities?



- Each area an entity?
  - Too inaccurate
- Each byte an entity?
  - Accurate
  - But analysis is slow
  - 2^32 addresses or more

What are the entities?



- Each area an entity?
  - Too inaccurate
- Each byte an entity?
  - Accurate
  - But analysis is slow
  - 2^32 addresses or more
- Suppose we know the layout
  - Use the constituents as entities
  - Balanced solution

How to identify the entites?

- Aggregate Structure Identification
  - G. Ramalingam et al
- Algorithm
  - Ignores declarative information about aggregates (arrays, structs)
  - Decomposes aggregates - based on access patterns in program
  - Identified components – atoms
  - Unifies atoms which ought to have same type

Aggregate Structure Identification - G. Ramalingam et al

- Year 2000 problem
  - Used to identify date type variables
  - Made maintenance easier
- Improving static analysis algorithms
  - Aggregates considered like scalars
  - Imprecision creeps in
  - Do analysis in terms of atoms
  - Precision improves!

Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```

Aggregate Structure Identification - Example

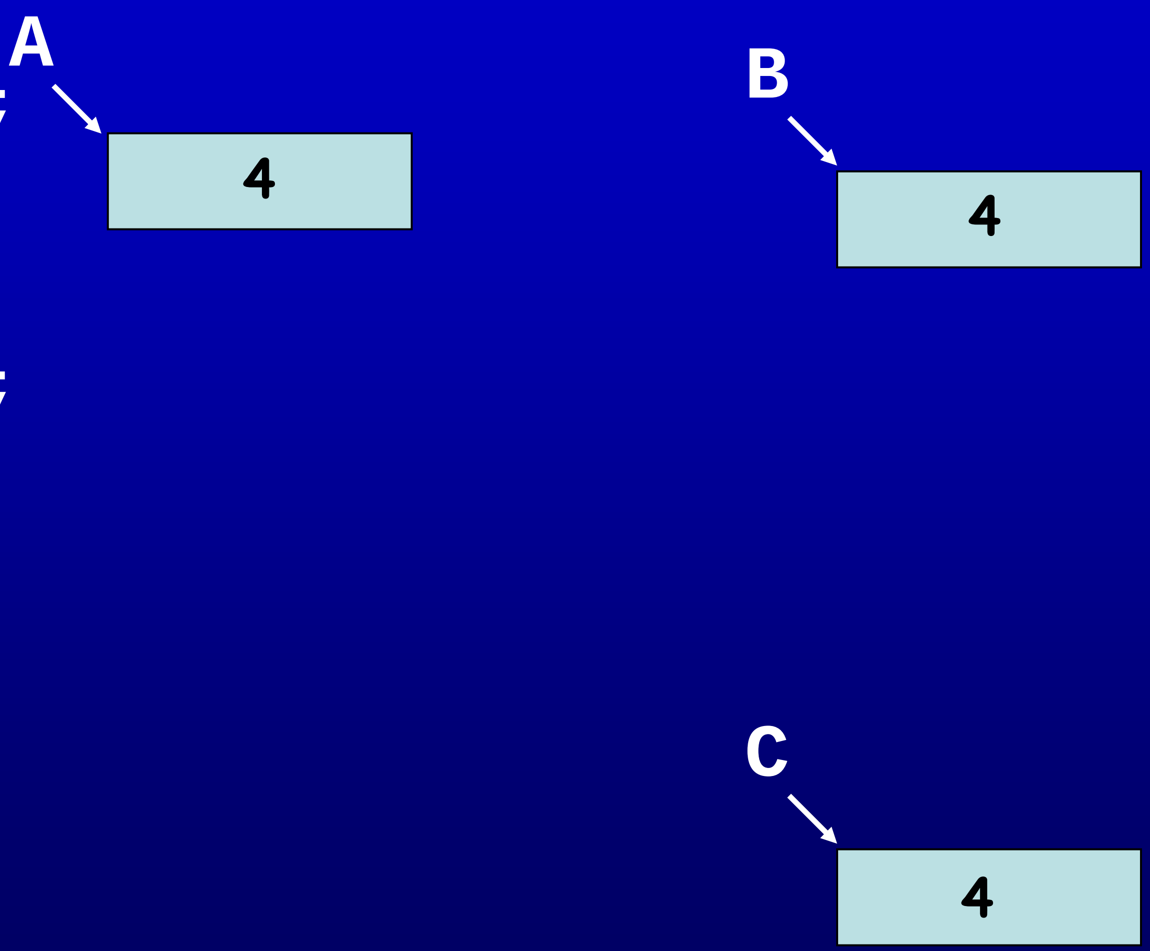
```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```

Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```

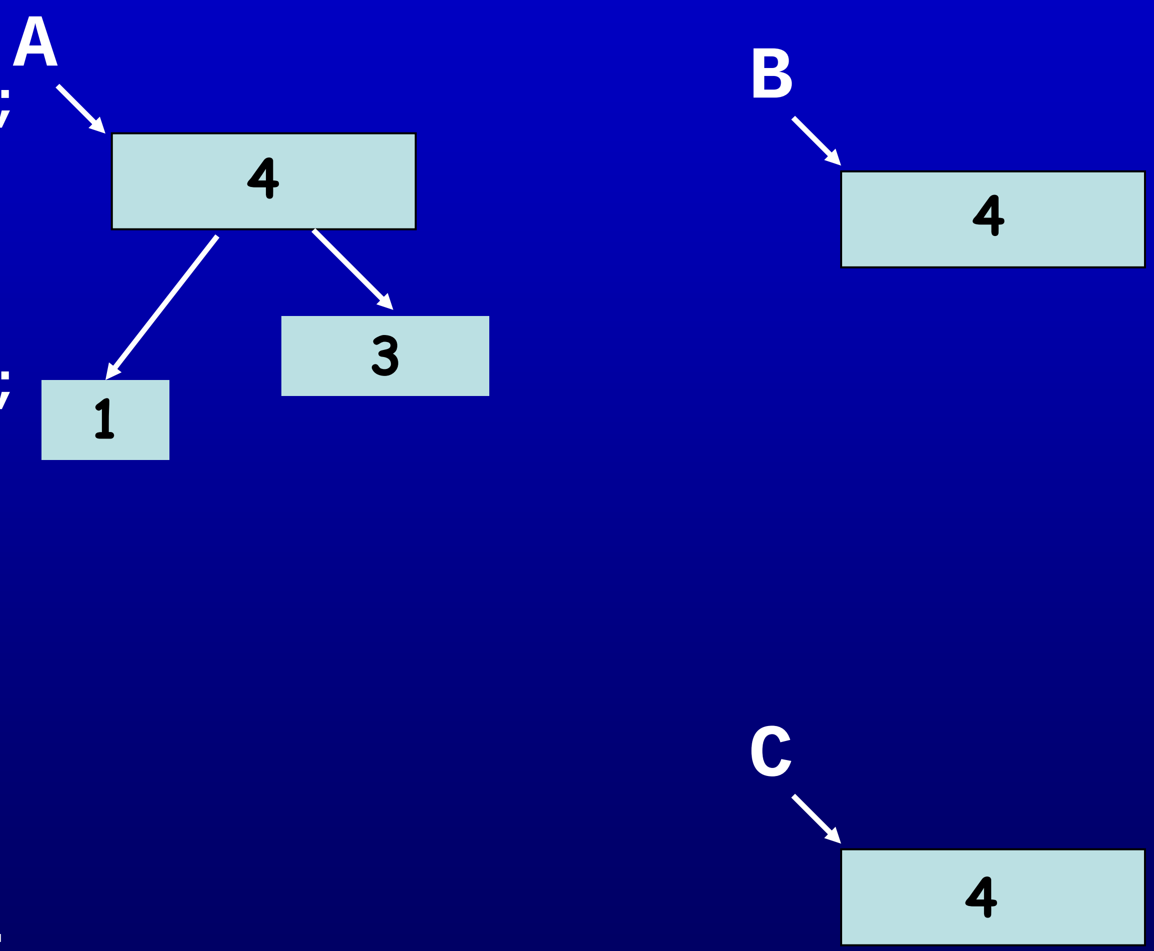
Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```



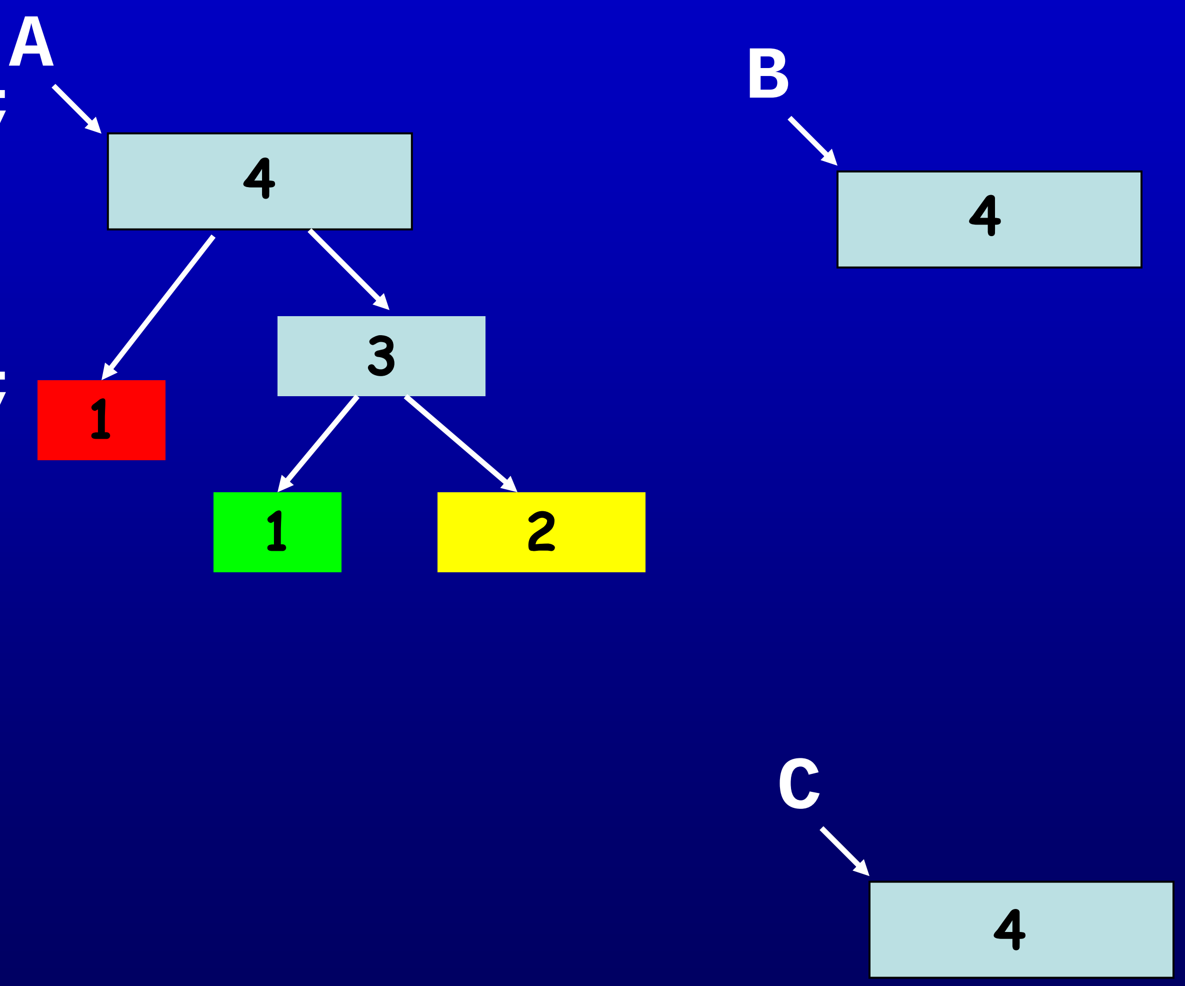
Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```



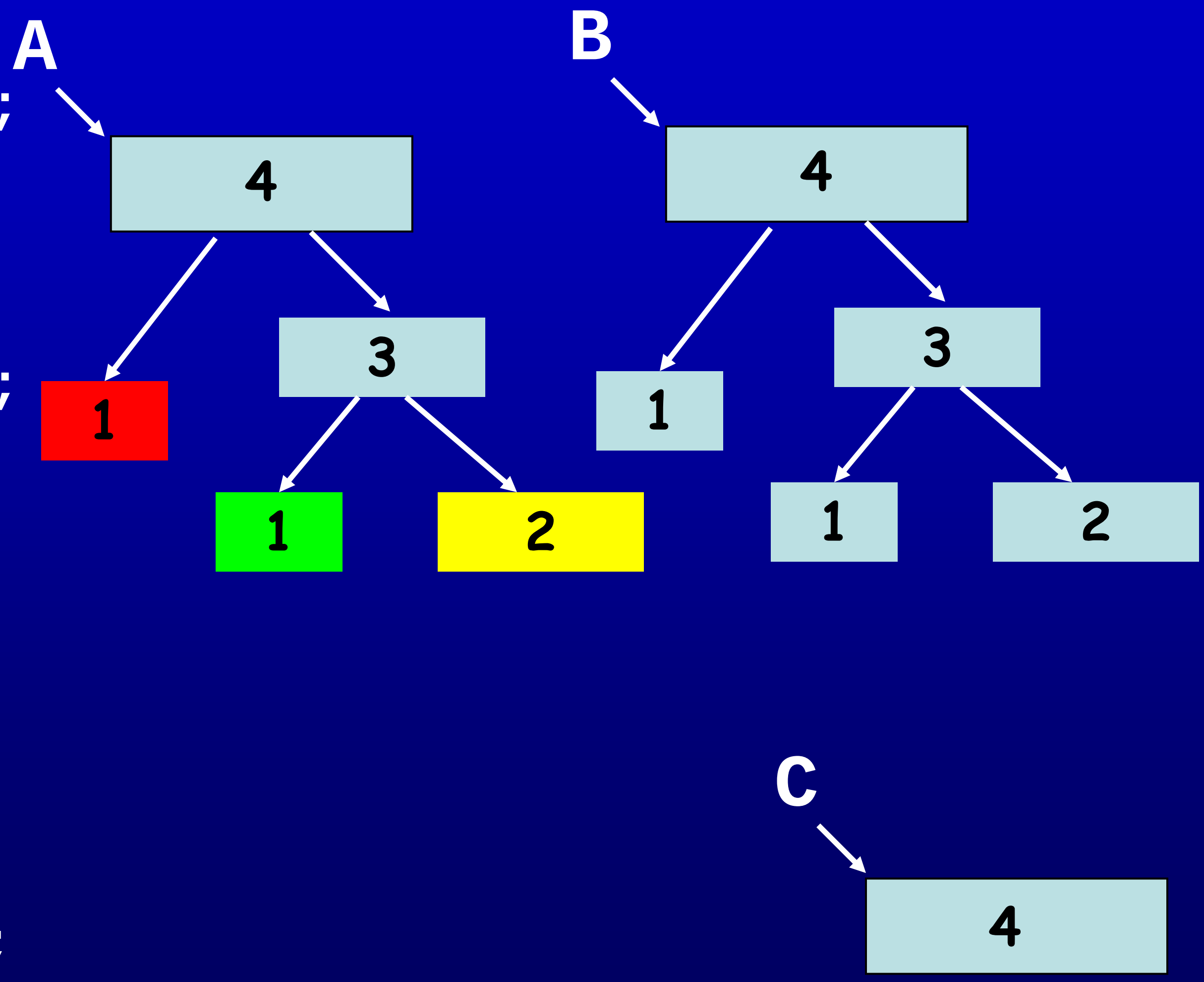
Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```



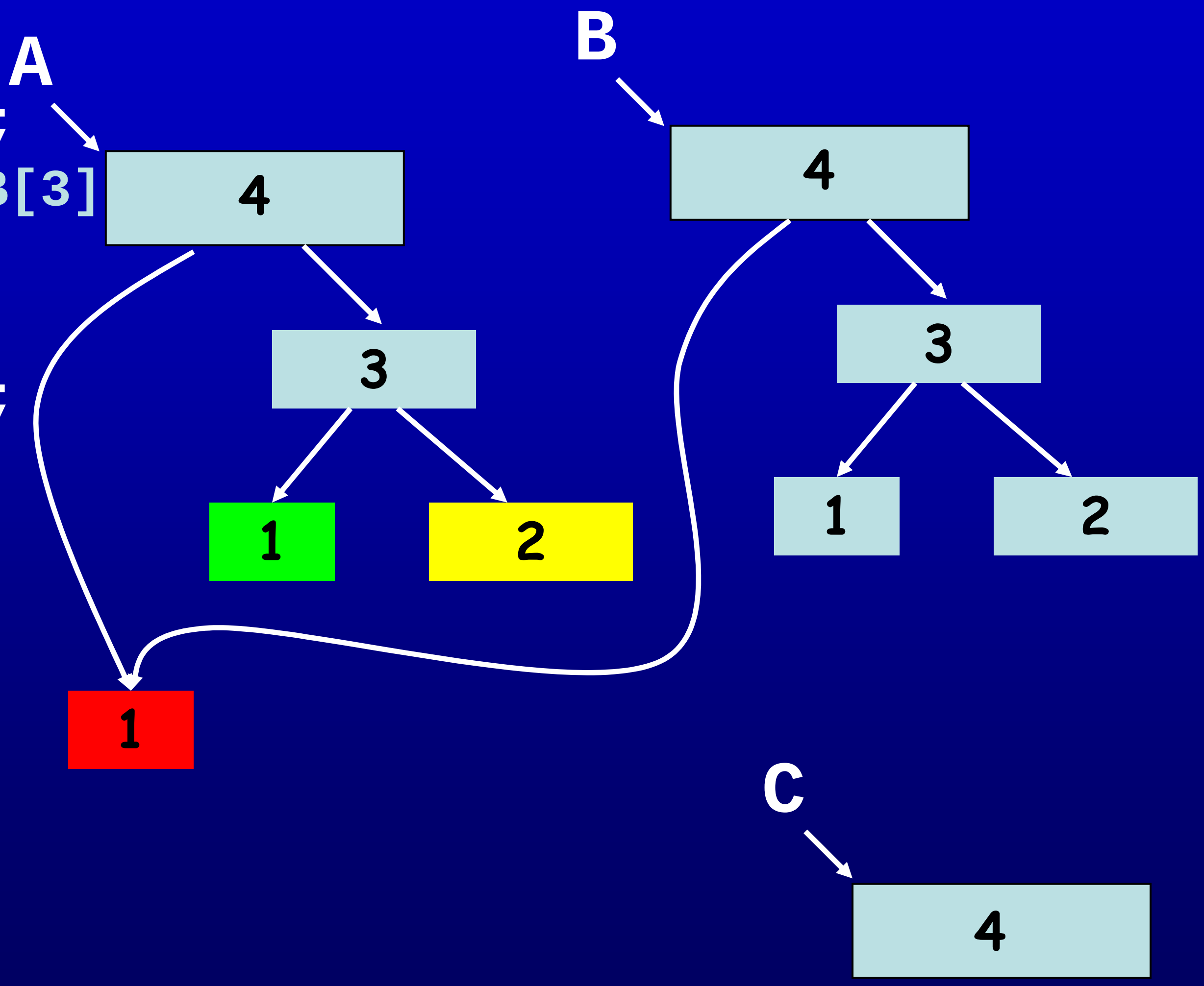
Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```



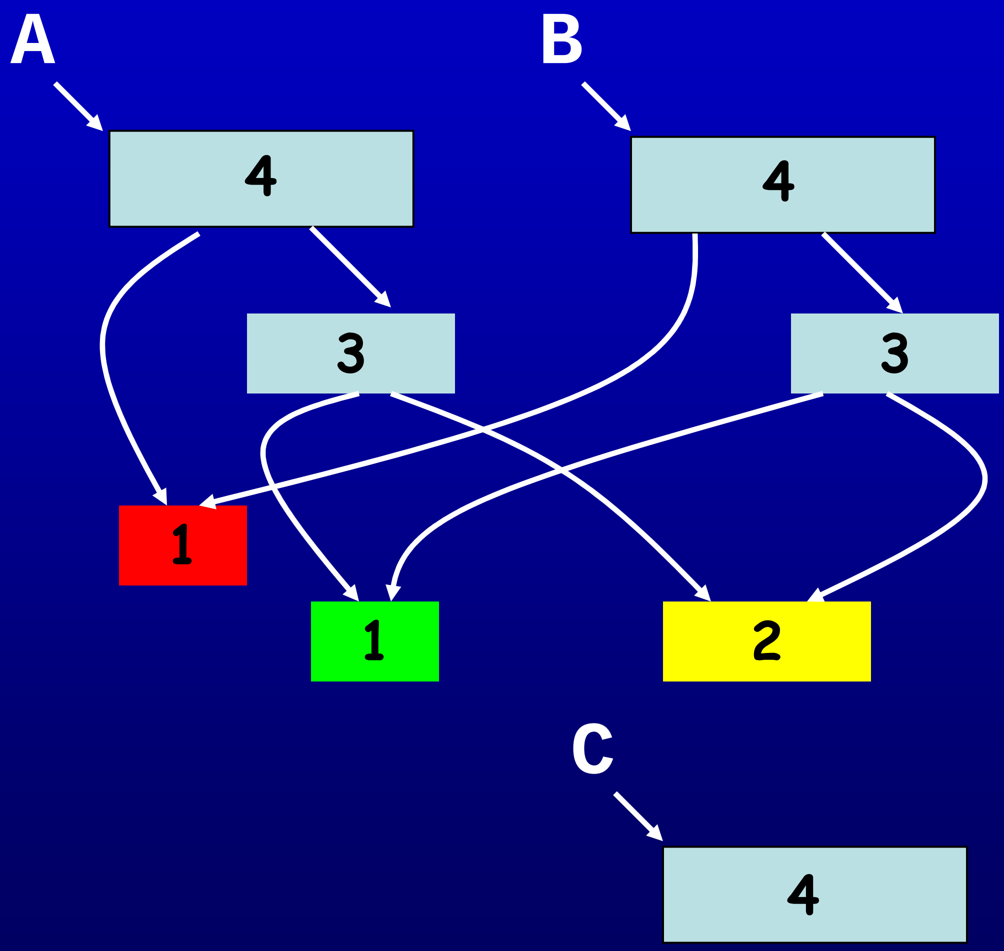
Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.B[0],B[1],B[2],B[3]
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```



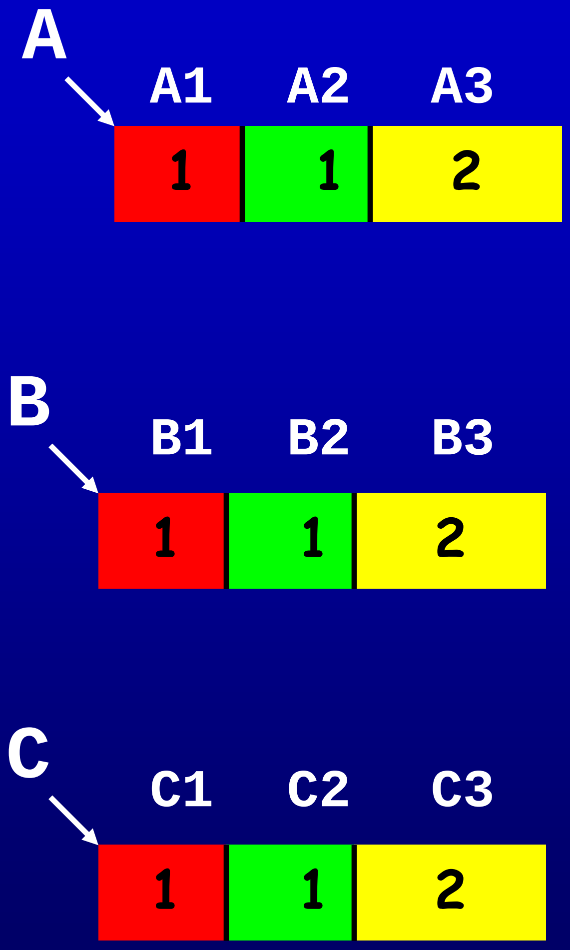
Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.B[0],B[1],B[2],B[3]
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```



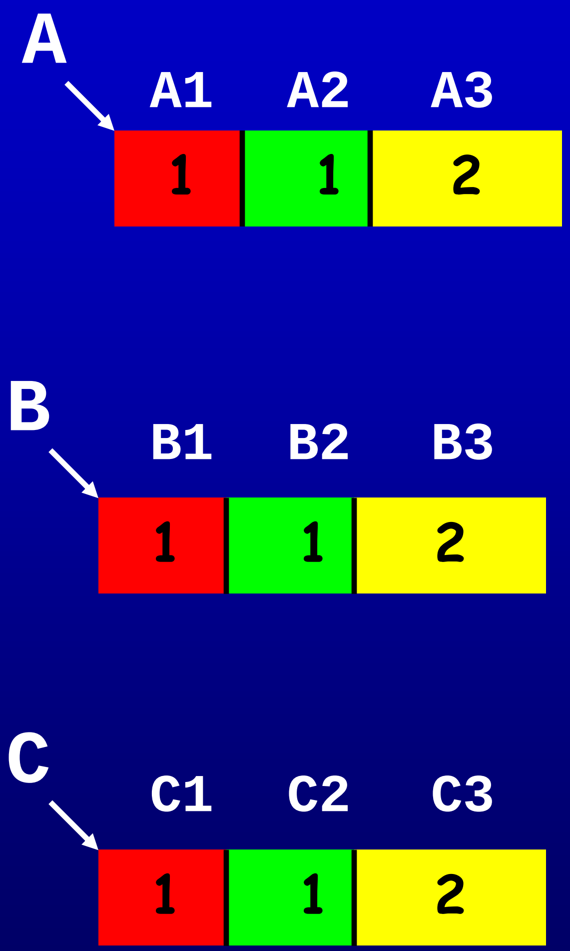
Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B. B[0],B[1],B[2],B[3]
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;
move 17 to F1;
move 18 to F2;
move A to B;
move B to C;
move F5 to RESULT;
```



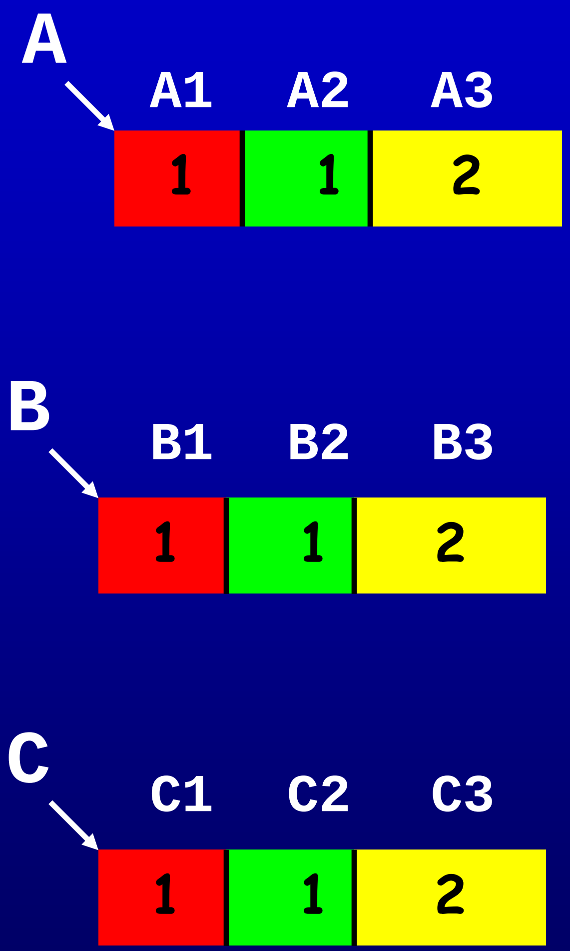
Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1; {17 -> A1}
move 18 to F2; {18 -> A2}
move A to B; {(A1,A2,A3)->(B1,B2,B3)}
move B to C; {(B1,B2,B3)->(C1,C2,C3)}
move F5 to RESULT; {C1 -> RESULT}
```

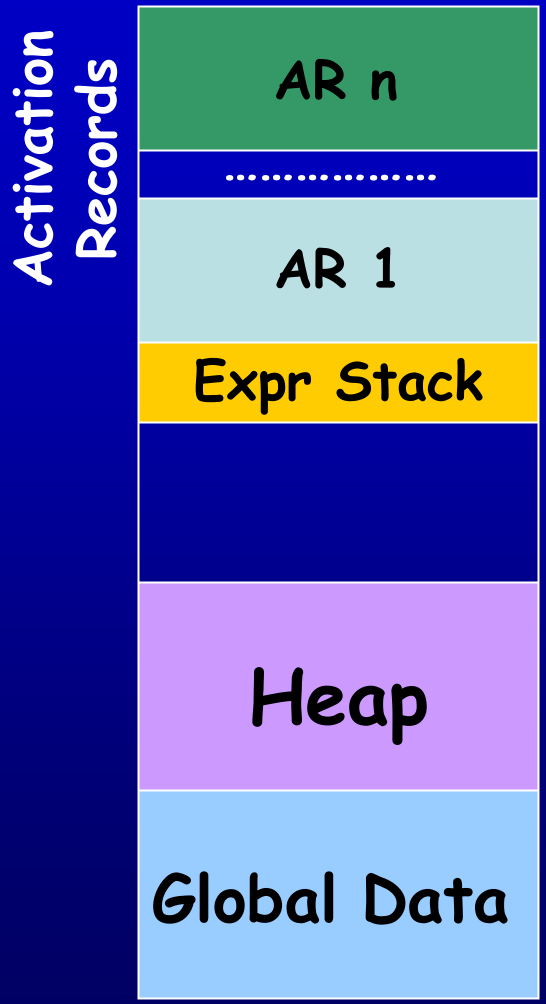


Aggregate Structure Identification - Example

```
A.
   int F1,F2,F3,F4;
B.
   int [4];
C.
   int F5,F6,F7,F8;
RESULT.
   int;

move 17 to F1; {17 -> A1}
move 18 to F2; {18 -> A2}
move A to B; {(A1,A2,A3)->(B1,B2,B3)}
move B to C; {(B1,B2,B3)->(C1,C2,C3)}
move F5 to RESULT; {C1 -> RESULT}
```

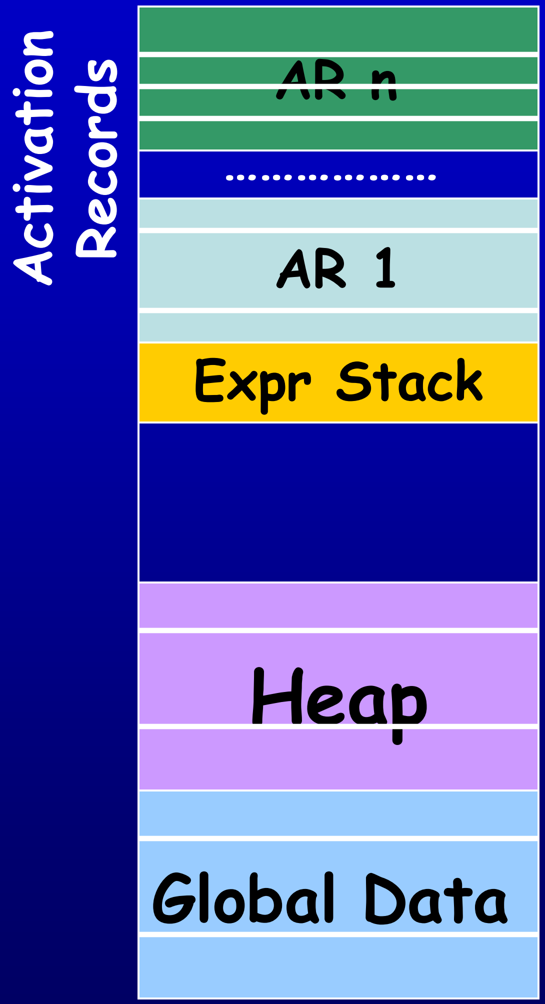


Aggregate Structure Identification in Object Code



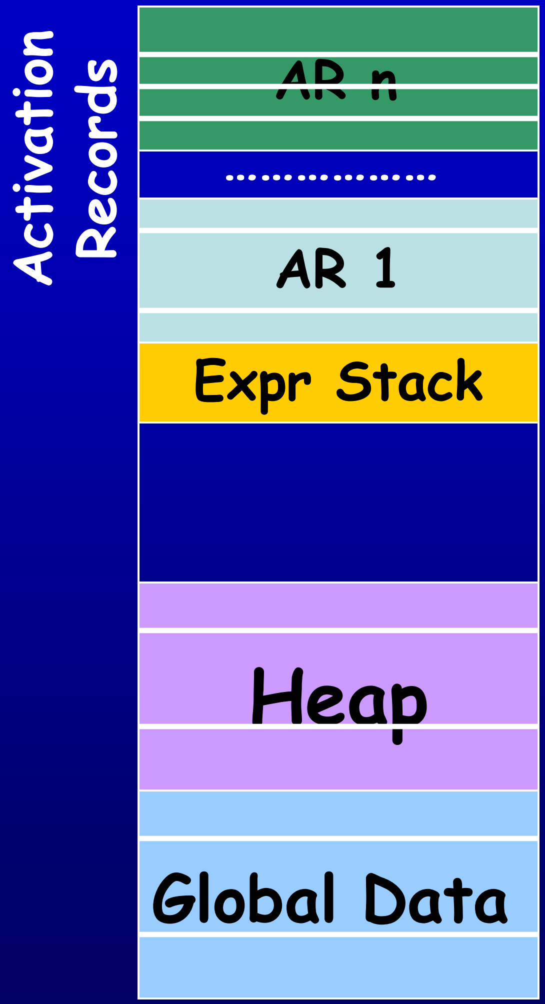
- These areas of memory
  - Aggregates in the algorithm

Aggregate Structure Identification in Object Code



- These areas of memory
  - Aggregates in the algorithm
- Identify the structure

Aggregate Structure Identification in Object Code



- These areas of memory
  - Aggregates in the algorithm
- Identify the structure
- Use the atoms of the aggregates as the entities

Minilanguage

- Input to the atomization algorithm
- Getting the minilanguage program
  - Retain only data transfer instructions
  - DataRef = DataRef
- DataRef
  - Data reference – three kinds
  - Program Variables
  - Range – for fields of aggregates
    - mov 17 to F1
    - A[1:4] = _int_const[1:4]
  - Statically indeterminate element of an array
    - mov 12 to B[i]
    - B[1:16]\4 = int_const[1:4]

Generating the Minilanguage file

- Which part of an aggregate is read/written?
- Clear in high level languages
- Not evident in object code

```
int main(){
   int a[10],i,j;
   j=0;
   for(i=0;i<10;++i){
        a[i]=i;
   }
   return j;
}
```

a[1:40]\10=i

```
; ebx corresponds to variable i
sub     esp, 44
mov     [esp+40],0    ; j = 0
xor     ebx, ebx   ; i = 0
lea     ecx, [esp]
loc_9:
   mov     [ecx], ebx ; a[i]=i
   inc     ebx          ; i++
   add     ecx, 4
   cmp     ebx, 10      ; i<10?
   jl      short loc_9 ;
..............................................
..............................................
```

Generating the Minilanguage file

- Which part of an aggregate is read/written?
- Clear in high level languages
- Not evident in object code

```
int main(){
   int a[10],i,j;
   j=0;
   for(i=0;i<10;++i){
       a[i]=i;
   }
   return j;
}
```

```
; ebx corresponds to variable i
sub     esp, 44
mov     [esp+40],0   ; j = 0
xor     ebx, ebx  ; i = 0
lea     ecx, [esp]
loc_9:
   mov     [ecx], ebx ; a[i]=i
   inc     ebx        ; i++
   add     ecx, 4
   cmp     ebx, 10     ; i<10?
   jl      short loc_9 ;
……………………………………
……………………………………
```

????=ebx[1:4]

Generating the Minilanguage file

- Which part of an aggregate is read/written?
- Inferred from the linear relationship among registers

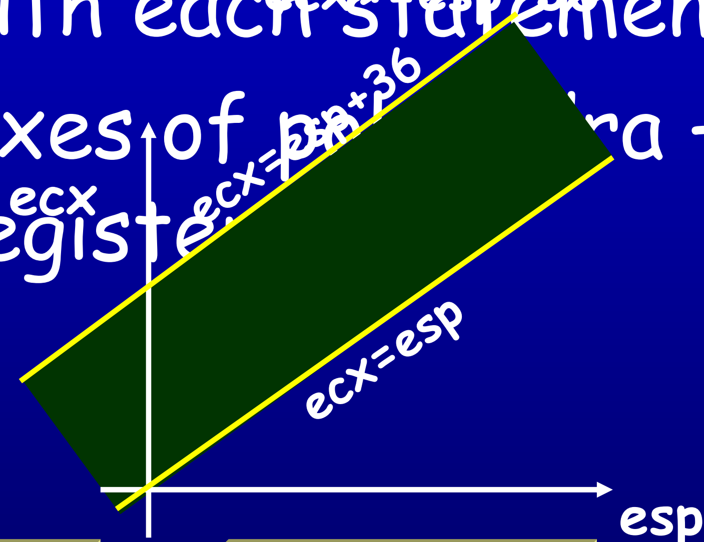
```
int main(){
   int a[10],i,j;
   j=0;
   for(i=0;i<10;++i){
         a[i]=i;
   }
```

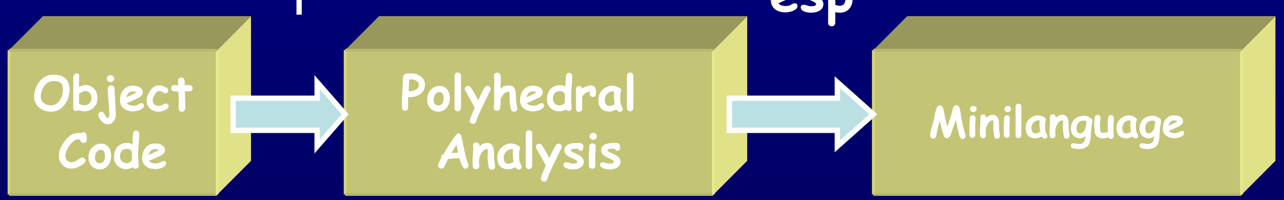
```
; ebx corresponds to variable i
sub     esp, 44
mov     [esp+40],0    ; j = 0
xor     ebx, ebx   ; i = 0
lea     ecx, [esp]
loc_9:
    mov     [ecx], ebx ; a[i]=i
    inc     ebx        ; i++
    add     ecx, 4
    cmp     ebx, 10     ; i<10?
    jl      short loc_9 ;
……………………………………
……………………………………
```

```
ecx>=esp &&
ecx<=esp+36
∴AR[1:40]\10= ebx[1:4]
```

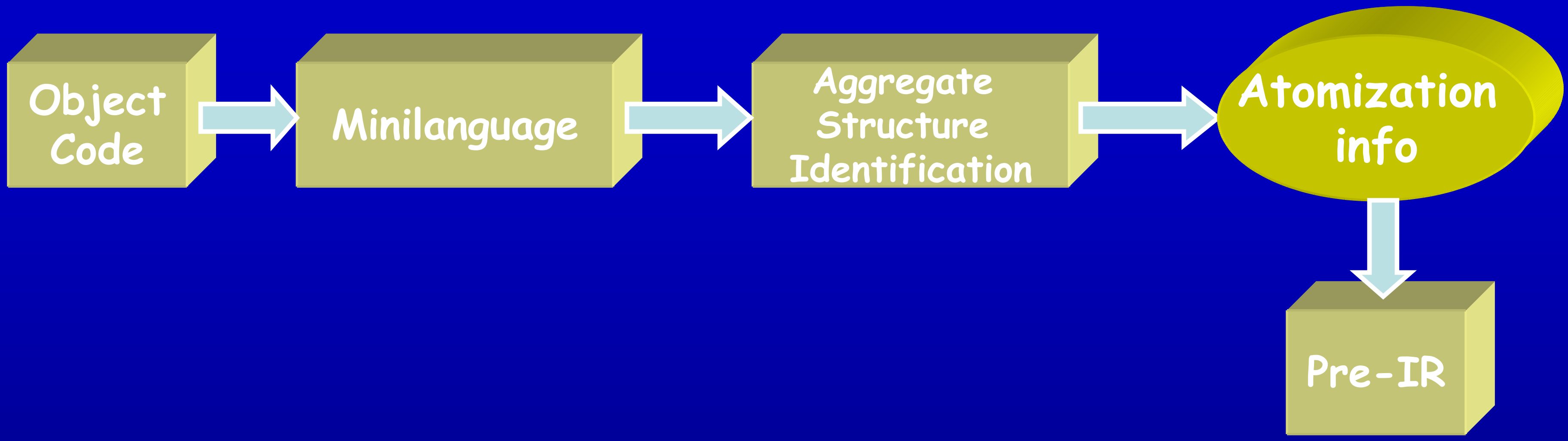
Linear Relationship among Registers

- Use convex polyhedra
- Associate a polyhedra with each statement
- Axes of polyhedra – registers





Steps in the Algorithm



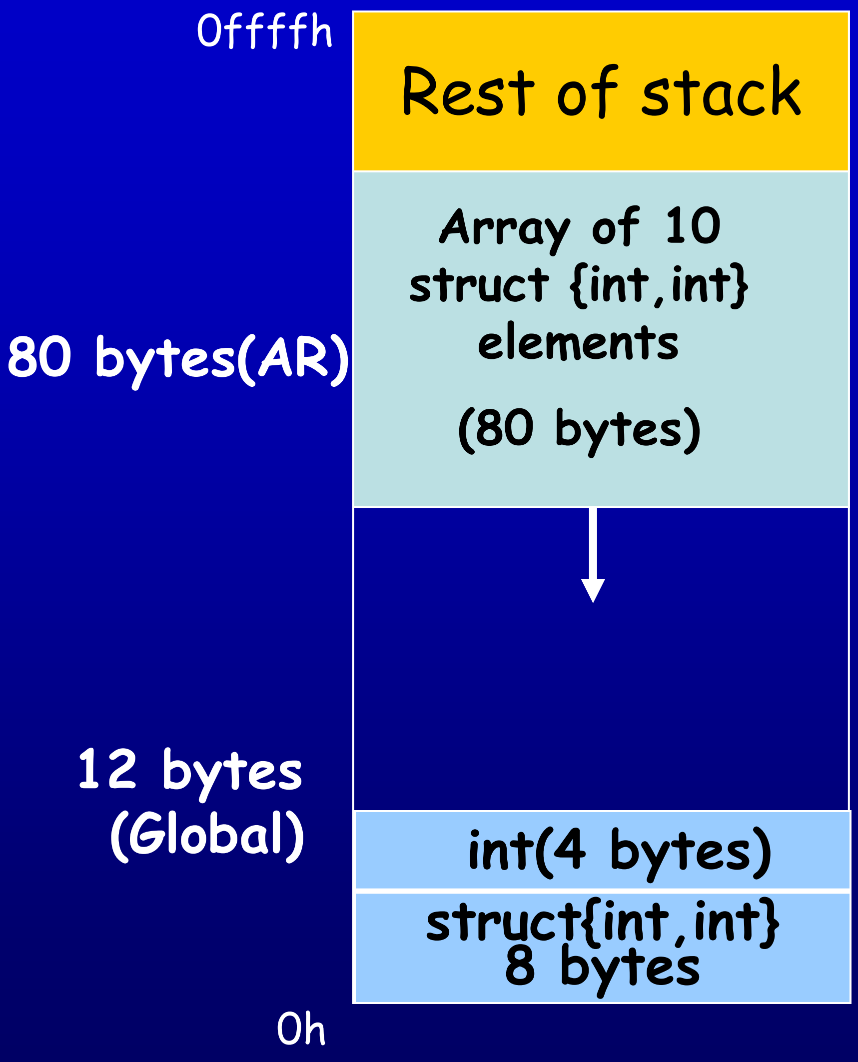
- Object Code → Minilanguage (Polyhedral analysis)
- Feed minilanguage to Ramalingam's analysis
- Identify the atoms
- Create pre-IR
- Feed it to codesurfer

Demo

```
struct Point{
    int x,y;
};
struct Point g_pt={10,20};
int gl_int=100;

int main() {
    struct Point l_a_pt[10];
    int i;

    g_pt.x=gl_int;
    for(i=0;i<10;++i) {
        l_a_pt[i].x=g_pt.x;
        l_a_pt[i].y=g_pt.y;
    }
    return 0;
}
```

```
public _main
        mov     edx, ds:gl_int
        sub     esp, 50h
        mov     ds:g_pt@x, edx
        lea     eax, [esp+50h+var_4C]
        mov     esi, ds:dword_4
        push    esi
        mov     ecx, 0Ah

loc_2B: mov     [eax-4], edx
        mov     [eax], esi
        add     eax, 8
        dec     ecx
        jnz     short loc_2B
        pop     esi
        add     esp, 50h
        retn
_main           endp
```

Demo




```
public _main
          mov     edx, ds:gl_int
          sub     esp, 50h
          mov     ds:g_pt@x, edx
          lea     eax, [esp+50h+var_4C]
          mov     esi, ds:dword_4
          push    esi
          mov     ecx, 0Ah

loc_2B:   mov     [eax-4], edx
          mov     [eax], esi
          add     eax, 8
          dec     ecx
          jnz     short loc_2B
          pop     esi
          add     esp, 50h
          retn
_main             endp
```

Demo

minilanguage file

```
decl eax 4;
decl ecx 4;
decl edx 4;
decl ebx 4;
decl esp 4;
decl ebp 4;
decl esi 4;
decl edi 4;
decl _main_AR 84;
decl Global 12;
decl const 4;

edx[1:4] = Global[9:12];
Global[1:4] = edx[1:4];
esi[1:4] = Global[5:8];
ecx[1:4] = const[1:4];
_main_AR[5:84]\10[1:4] = edx[1:4];
_main_AR[5:84]\10[5:8] = esi[1:4];
```

```
public _main
        mov     edx, ds:gl_int
        sub     esp, 50h
        mov     ds:g_pt@x, edx
        lea     eax, [esp+50h+var_4C]
        mov     esi, ds:dword_4
        push    esi
        mov     ecx, 0Ah

loc_2B: mov     [eax-4], edx
        mov     [eax], esi
        add     eax, 8
        dec     ecx
        jnz     short loc_2B
        pop     esi
        add     esp, 50h
        retn
_main           endp
```

Conclusions

- No properly defined entities in object code
- Ramalingam's atomization algorithm
  - Atoms can be the entities
- Now, existing static analysis algorithms can be adopted to object code

Preparing Object Code for Static Analysis

Gogul Balakrishnan
University of Wisconsin-Madison