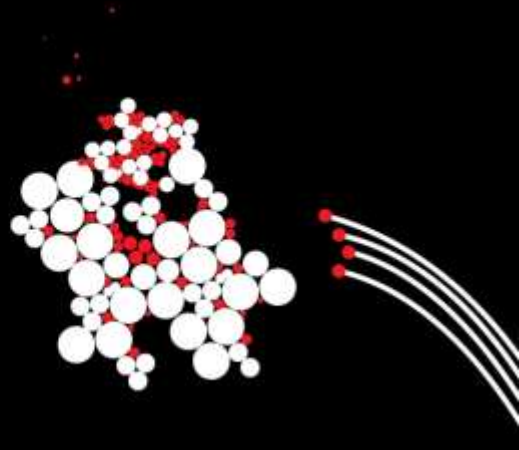
UNIVERSITY OF TWENTE.

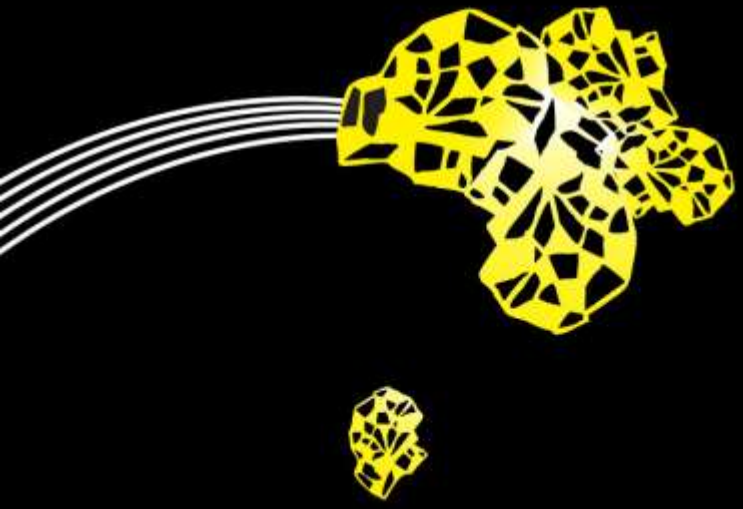


COMPILER CONSTRUCTION:

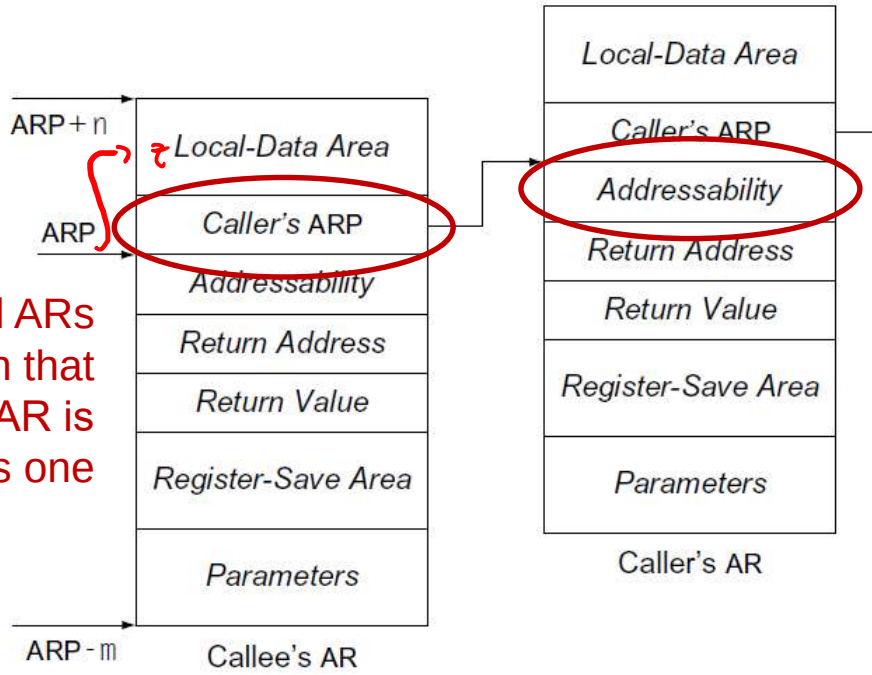
CC 6-1: MEMORY USAGE (1)

27 MAY 2019

MODULE 8: PROGRAMMING PARADIGMS



REMINDER: ACTIVATION RECORDS



Not yet discussed:
This enables addressing variables in enclosing scope

Only useful in languages with nested procedures

Not needed if all ARs are on stack: in that case, the caller's AR is just below this one

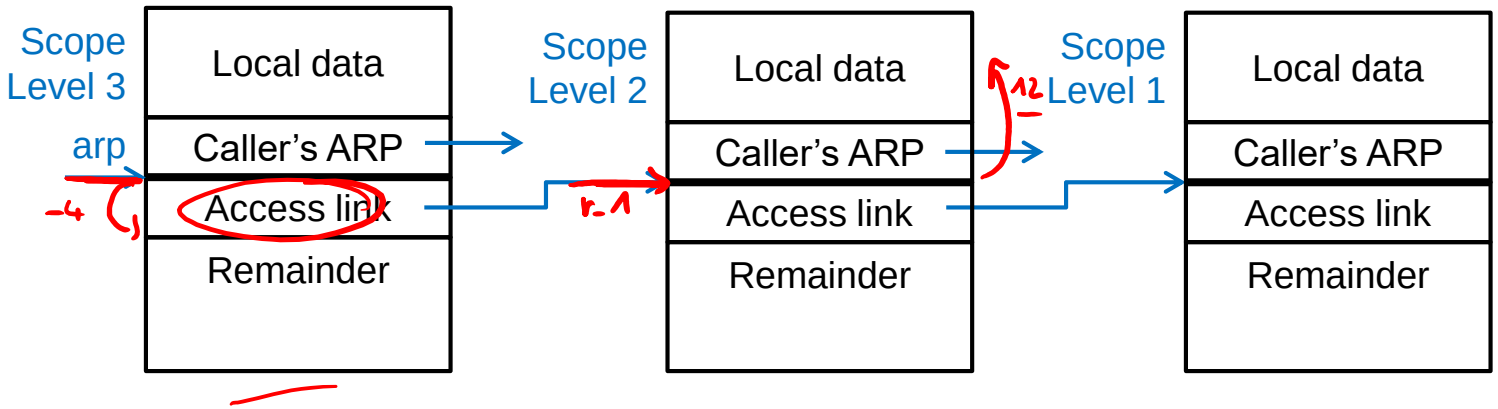
↳ yes, but needs cooperation with caller (size of its local data area not known to callee)

z: (3, 4)
x: (1, 4)

```
program Main;
①   var x, y, z: integer;
    procedure Fee(r: real);
    ②   var x: integer;
        begin { body Fee } end;
    procedure Fie(x: integer);
    ②   var y: real;
        procedure Foe;
        ③   var z: real;
            begin { body Foe }
                Fee(1.0);
            end;
        begin { body Fie }
            Foe;
        end;
    begin { Main }
        Fie(1);
    end.
```

ADDRESSABILITY BY ACCESS LINKS

- "Addressability" points to ARP of *lexically enclosing scope*
  - This is not necessarily the caller's ARP!

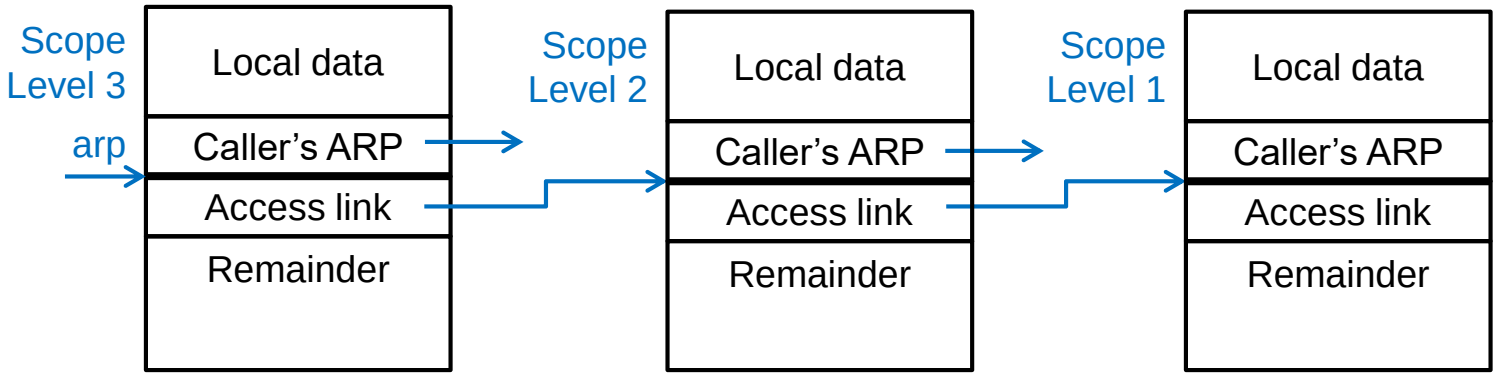


- From within level 3 code
  - To load coordinate (3,24) into `r_2`:
  - To load coordinate (2,12):
  - To load coordinate (1,16):

```
loadAI  r_arp, 24 => r_2
loadAI  r_arp, -4 => r_1
loadAI  r_1 , 12 => r_2
```

ADDRESSABILITY BY ACCESS LINKS

- "Addressability" points to ARP of *lexically enclosing scope*
  - This is not necessarily the caller's ARP!



- From within level 3 code
  - To load coordinate (3,24) into `r_2`:

```
loadAI r_arp, 24 => r_2
```

  - To load coordinate (2,12):

```
loadAI r_arp, -4 => r_1
loadAI r_1,12 => r_2
```

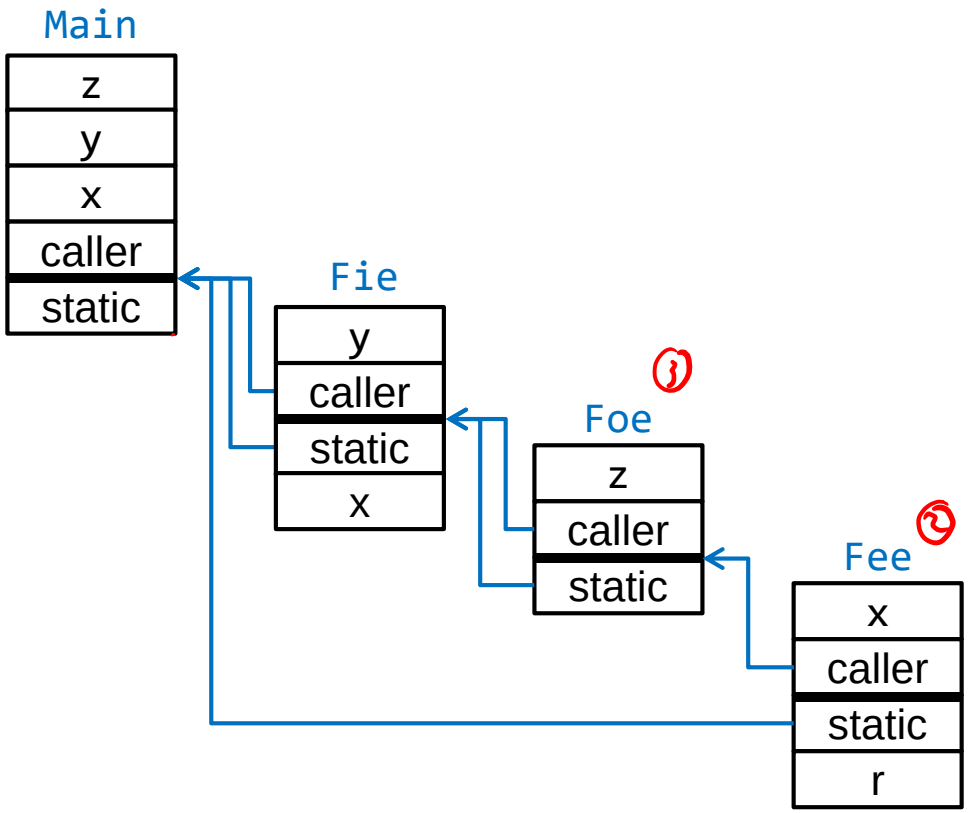
  - To load coordinate (1,16):

```
loadAI r_arp,-4 => r_1
loadAI r_1,-4 => r_1
loadAI r_1,16 => r_2
```

EXAMPLE STATIC LINK (HEAP-BASED ARS)

```
program Main;
var x, y, z: integer;
  procedure Fee(r: real);
  var x: integer;
    begin { body Fee } end;
  procedure Fie(x: integer);
  var y: real;
    procedure Foe;
    var z: real;
      begin { body Foe }
        Fee(1.0);
      end;
    begin { body Fie }
      Foe;
    end;
  begin { Main }
    Fie(1);
  end.
```

Calling Sequence:
Main → Fie → Foe → Fee

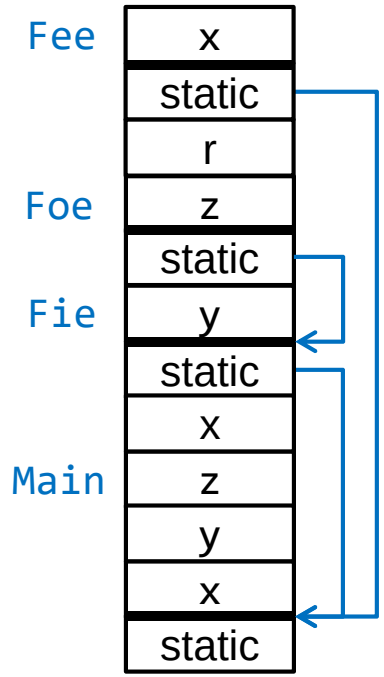


EXAMPLE STATIC LINK (STACK-BASED ARS)

```
program Main;
  var x, y, z: integer;
  procedure Fee(r: real);
    var x: integer;
    begin { body Fee } end;
  procedure Fie(x: integer);
    var y: real;
    procedure Foe;
      var z: real;
      begin { body Foe }
        Fee(1.0);
      end;
    begin { body Fie }
      Foe;
    end;
  begin { main }
    Fie(1);
  end.
```

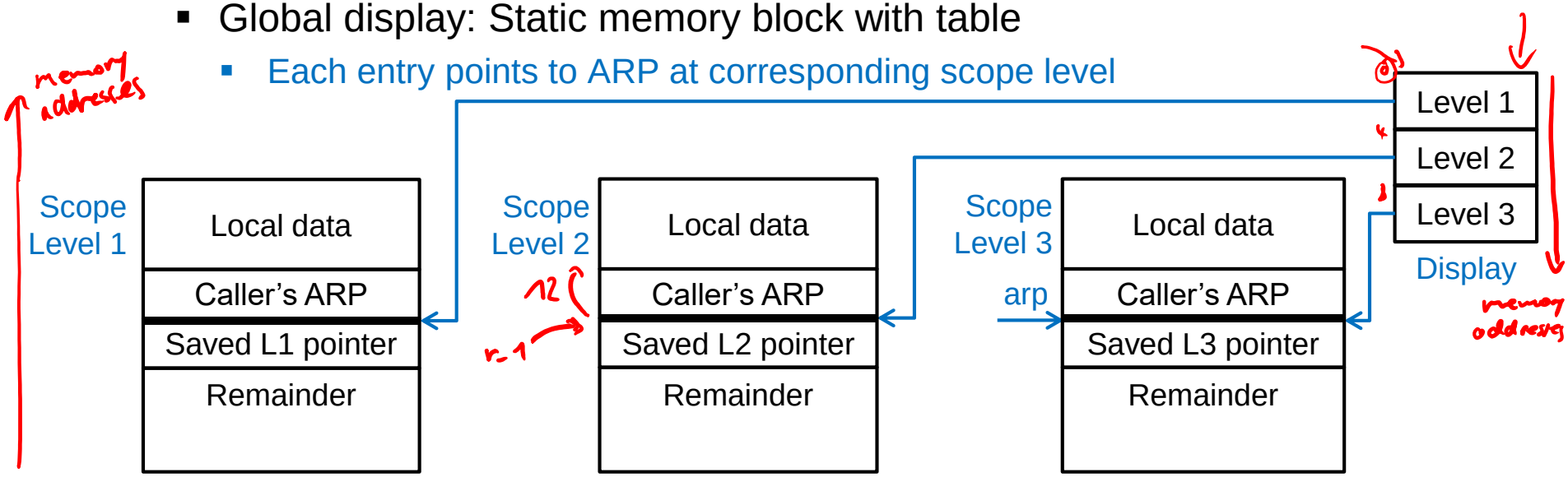
Caller's ARP can be omitted.

Calling Sequence:
`Main → Fie → Foe → Fee`



ADDRESSABILITY BY GLOBAL DISPLAY

- Global display: Static memory block with table
  - Each entry points to ARP at corresponding scope level

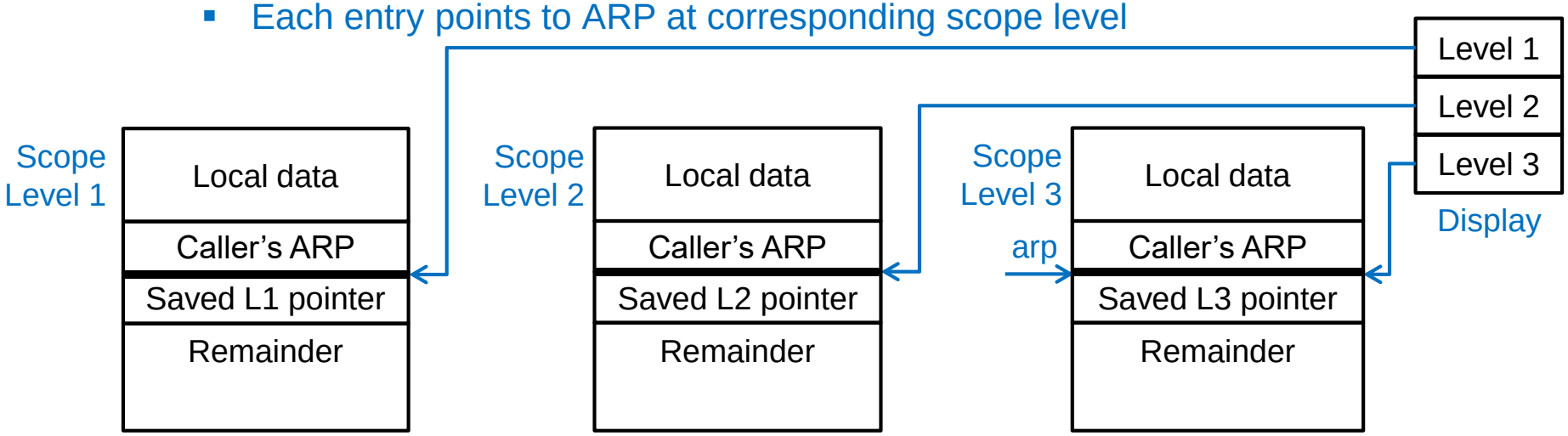


- From within level 3 code
  - To load coordinate (3,24) into `r_2`:
  - To load coordinate (2,12):
  - To load coordinate (1,16):

ADDRESSABILITY BY GLOBAL DISPLAY

Note:
'Global' display
is thread-local!

- Global display: Static memory block with table
  - Each entry points to ARP at corresponding scope level



- From within level 3 code
  - To load coordinate (3,24) into `r_2`:

```
loadAI r_arp, 24 => r_2
```

  - To load coordinate (2,12):

```
loadI  @disp+4 => r_1
loadAI r_1,12 => r_2
```

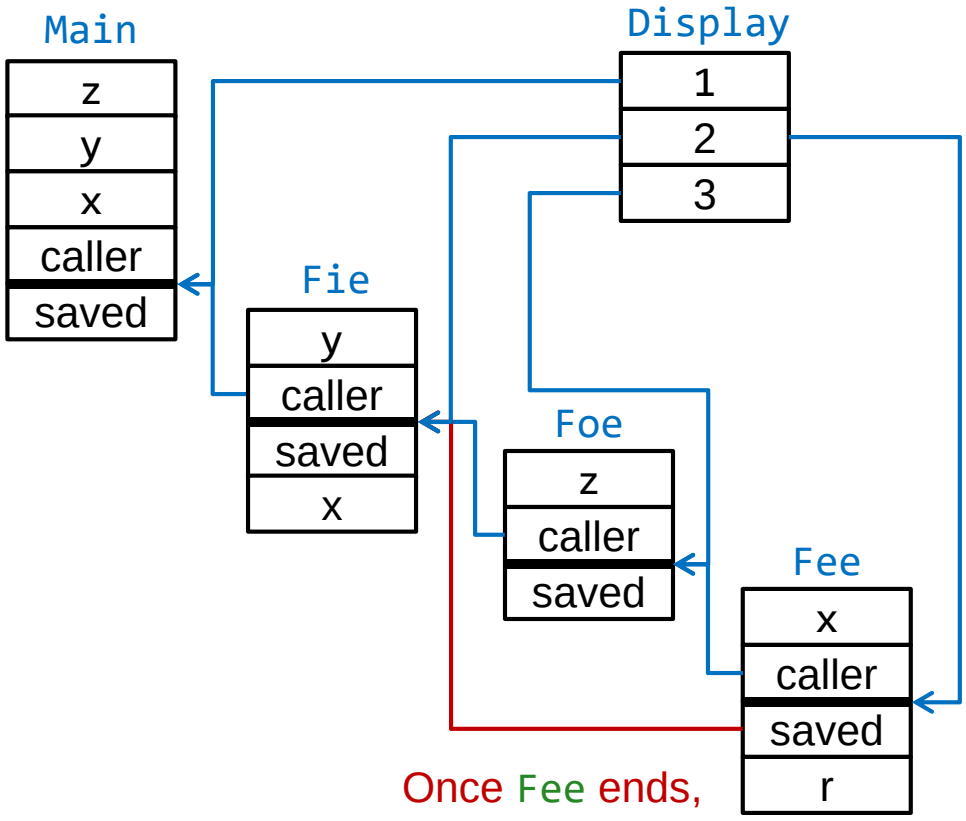
  - To load coordinate (1,16):

```
loadI  @disp => r_1
loadAI r_1,16 => r_2
```

EXAMPLE DISPLAY (HEAP-BASED ARS)

```
program Main;
① var x, y, z: integer;
  procedure Fee(r: real);
  ② var x: integer;
    begin { body Fee } end;
  procedure Fie(x: integer);
    var y: real;
    procedure Foe;
    ③ var z: real;
      begin { body Foe }
        Fee(1.0);
      end;
    begin { body Fie }
      Foe;
    end;
  begin { main }
    Fie(1);
  end.
```

Calling Sequence:
Main → Fie → Foe → Fee



Once Fee ends,
level 2 scope must be restored to Fie