COS**341** Project **2** (2017): ***Static Semantics of SPL***

Part ***b***

SCOPE ANALYSIS

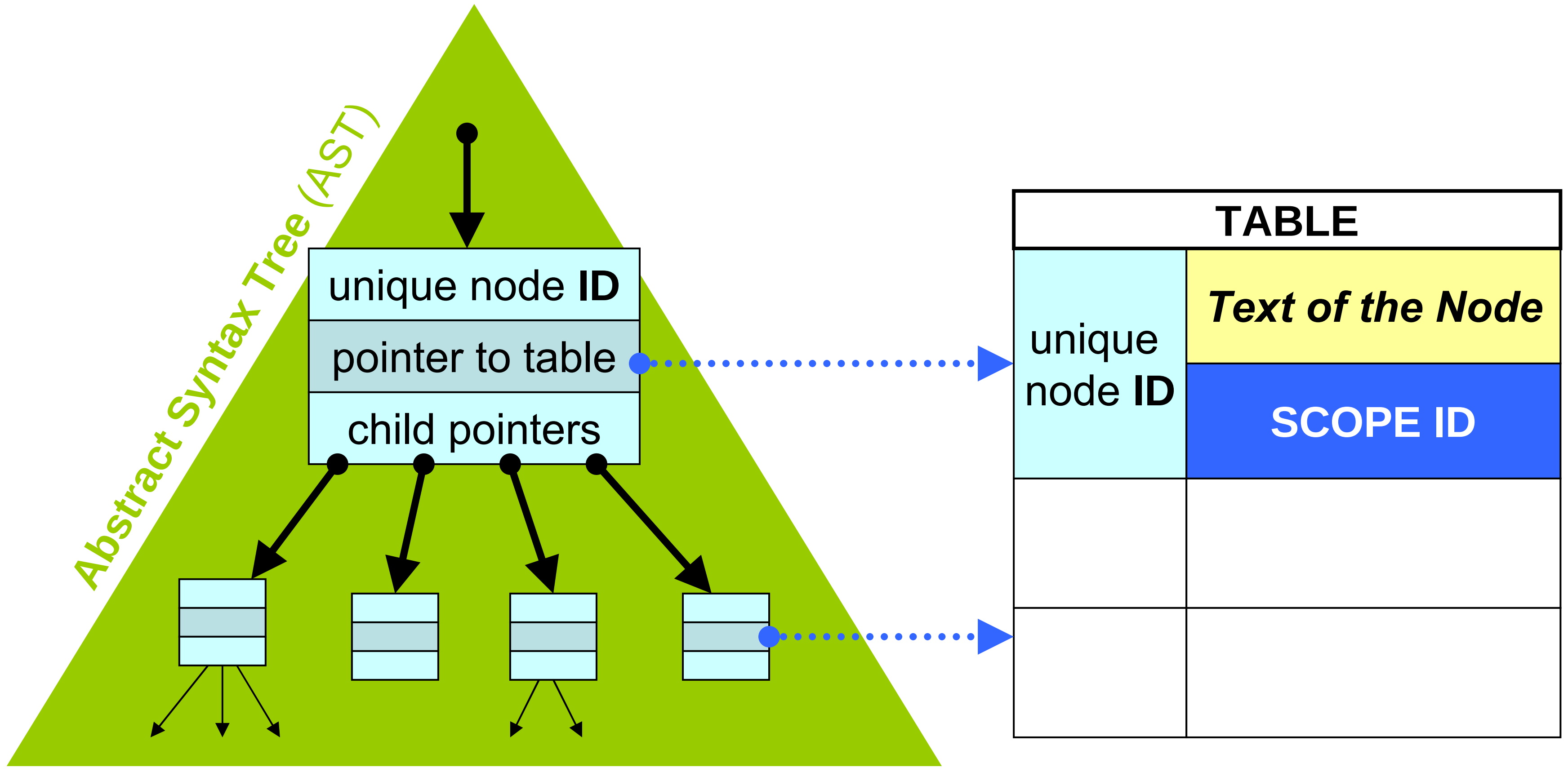
Preliminaries

- Whether or not two *same-named lexical entities* refer to the same value-object (in RAM) does not only depend on the TYPE of those same-named lexical entities,
  - (previous sub-project 2***a***)
- it also depends on **where** in a program two same-named-and-same-typed lexical **entities are positioned** (in the program's Syntax Tree)
  - (this sub-project 2***b***)

Preliminaries

- In all the following **examples** it is always **assumed** that **two same-named lexical entities also have** the **same type**.
  - If they were of different types, then they could never refer to the same value objects in RAM anyway,
    - (as we had already learned in sub-project 2a).

Goal of this Sub-Project 2*b*:



To attach to each node in the already existing Abstract Syntax Tree
a suitable **Scope ID**, on the basis of which it can be decided whether
or not two same-named-and-same-typed Variables belong to the same Scope

SPL Scoping:

Examples and Rules

In *SPL* the Condition of a FOR-Loop constitutes its own Scope

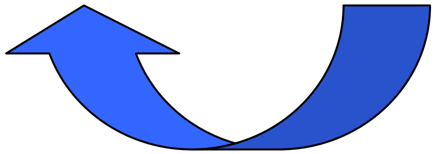
```
i = 12
j = 0
for(i=0 ; i<10 ; i=add(i,1))
    {
    j = add(j,i)
    i = sub(i,1)
    } // otherwise the FOR-Loop would never halt !!!
```

This is the FOR **i** in its own Scope

The green **i** is the one that gets modified by the sub calculation: The two green **i** are in the same scope (have the **same Scope ID**)

Scoping for the *SPL* Grammar Rule

PROG → **CODE ; PROC_DEFS**



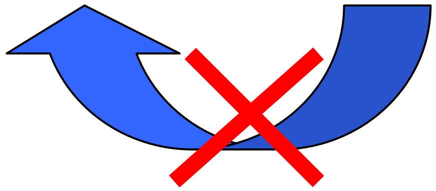
PROC_DEFS "**can see**" variables in CODE

- If two *same-named-and-same-typed lexical* entities (*Variables*) appear in CODE **and** in PROC_DEFS then these two Variables are *in the same Scope* (and hence: refer to the same value-object in RAM).

Exception: Scoping of the FOR-Loop-Condition, as on the previous slide**!**

Scoping for the *SPL* Grammar Rule

PROC_DEFS → PROC PROC_DEFS



PROC_DEFS "**cannot see**"
variables in PROC nor vice-versa

- If two *same-named-and-same-typed lexical* entities (*Variables*) appear in two *separate* (*non-nested*) *Procedure Definitions*, then they are ***NOT*** in the same scope (hence: refer to *different value-objects* in the RAM)

Exception:
Scoping of PROG → CODE ; PROC_DEFS as stated on the previous slide.

SPL Example

the **x** entities are **the same**

```
x = 12 ;
myproc { a = sub(x,1) }
yourproc { a = add(x,7) }
```

The red **a** and the blue **a** *refer to different value objects;* they **are NOT in the same scope**

Scoping for the **combination of** PROG → CODE ; PROC_DEFS PROC_DEFS → PROC PROC_DEFS

- If *two same-named procedure definitions* occur “behind” CODE; (at the level of PROG) then the *second* procedure definition is in the *scope level “infinity”* ∞ (i.e.: un-reachable, or illegal)

SPL Example

```
x = 12
myproc
output(x)
myproc { x = add(x,1) }
myproc { x = sub(x,1) }
```

in the ***same scope:*** *will be* ***called***

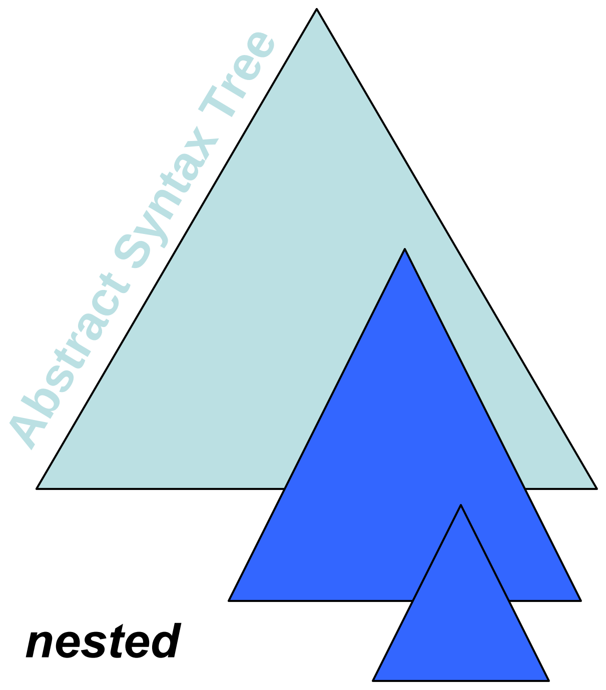
name clash *in* PROG *!!*

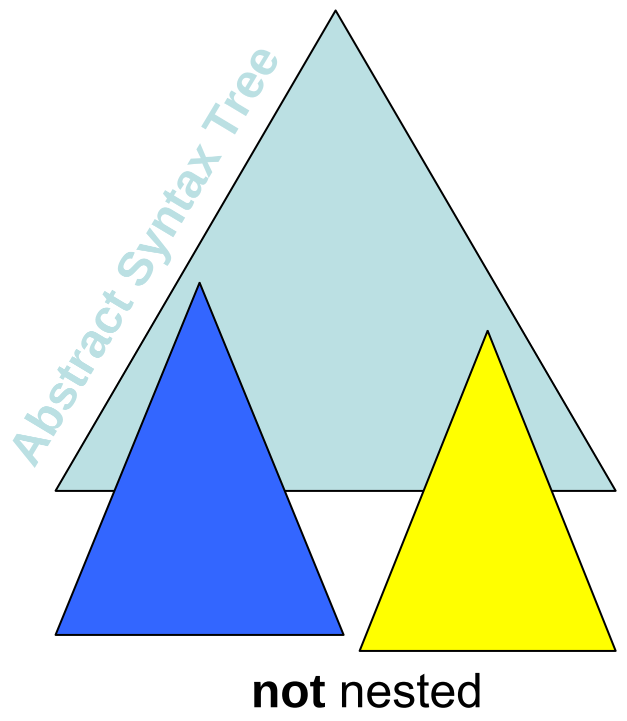
Scope Level ∞ *makes the "clashing" procedure declaration "invisible"*

Nested Procedure Definitions

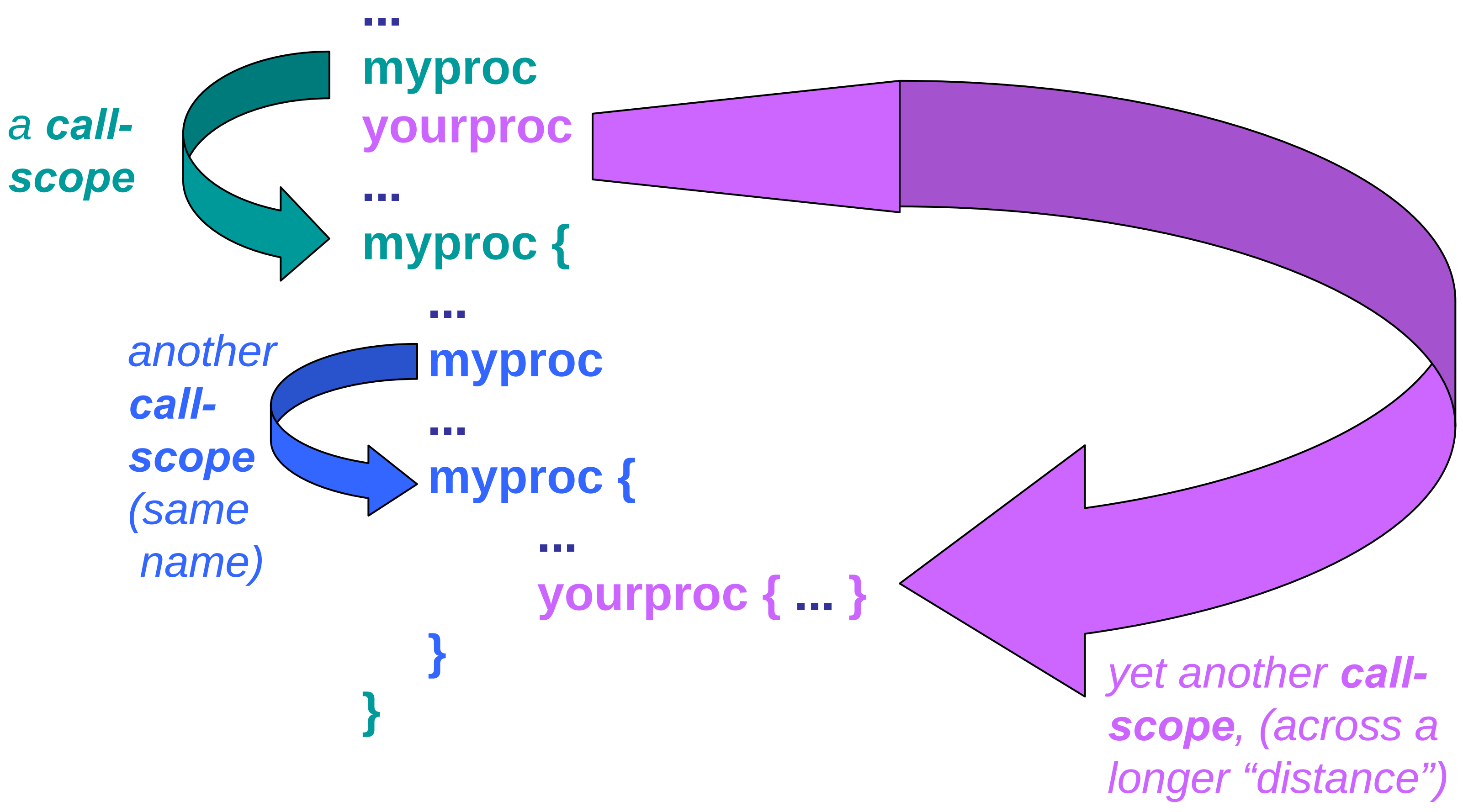
- **Scoping Rule** = "***The Outer-Most Visible*** *Procedure is* ***called from Top to Bottom***"

(*in particular: where* ***nested*** *procedures* are *same-named*)





SPL ***Example***: *Nested Procedures*



```
...
mypróc
yourproc
...
myproc {
    ...
    myproc
    ...
    myproc {
        ...
        yourproc { ... }
    }
}
```

*a **call-scope***
*another **call-scope** (same name)*
*yet another **call-scope**, (across a longer “distance”)*

<u>Note</u>: in ***SPL***, “***upwards***” is **<u>never</u>** a **legal** procedure **call-direction!!**

Special scope case: **output(VAR)**

- Because VAR can be number or string, we define that we shall always output the "*nearest fitting*" number or string variable "*before*" the output command in the AST.
- *SPL* Example:

In the ***Abstract Syntax Tree*** *(AST),* ***x*** *is the "nearest fitting" variable in the SCOPE of* **output(x)**

```
x = 12        // number type
x = "hello"   // string type
x             // procedure type
ouput(x)
proc x { ... }
```

Your TASK:

IMPLEMENT and TEST

the *SPL* **Scope Allocation**

Your algorithm must “crawl through” the Abstract Syntax Tree, ***until all*** Scope Information is found and written into the Table!

And **now**…

HAPPY PAIR-**CODING**!

☺☺

Note: Plagiarism is forbidden! Code swapping with other pairs of project students is also not allowed