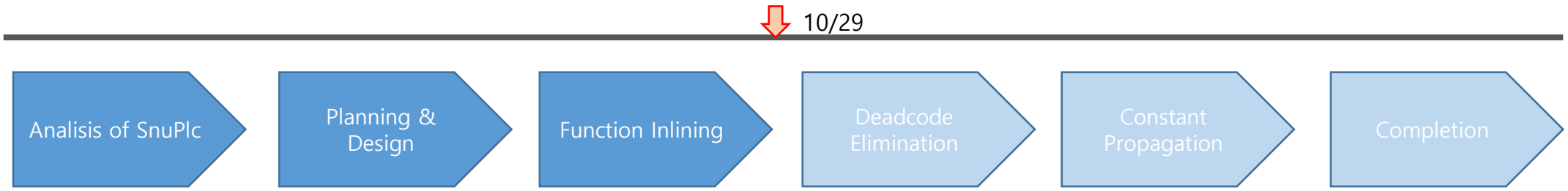
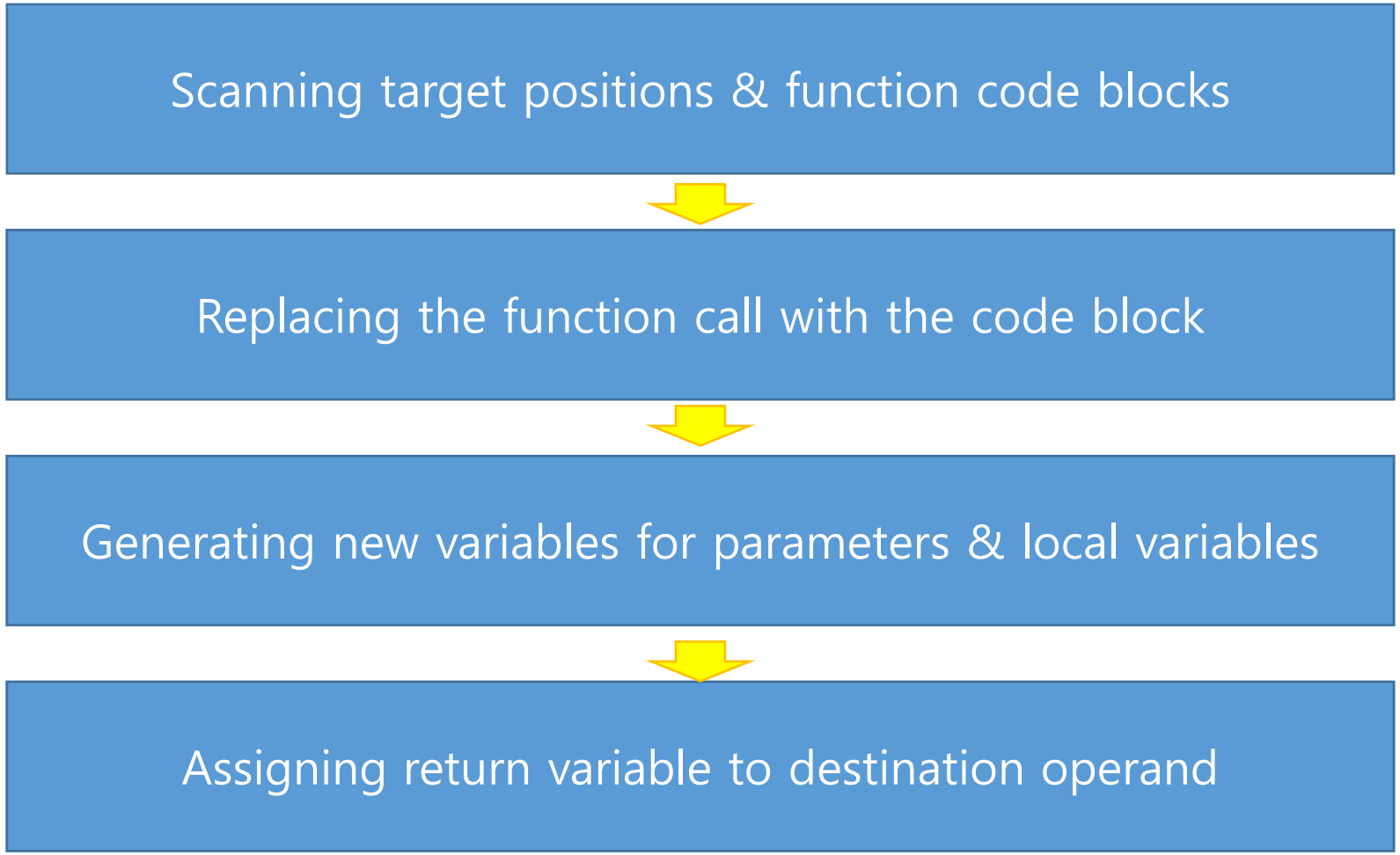
Milestone Presentation

2017-20845 Wonjae Jang
2018-25193 Dongwan Kang

Project progress



Basic process of Function Inlining



Our technique to prevent code explosion

- Scoring and inlining in order of score

```
1   void main(){
2       while(1){
3           while(1){
4               F();
5           }
6           F();
7       }
8       G();
9   }
10
11  void F(){
12      while(1){
13          G();
14      }
15  }
16
17  void G(){
18  }
```

Score =	**1/code_length**	**by default**
	3/code_length	**in function scope**
	5/code_length	**in while scope**

Length_limitation = origin size of CodeBlock x 2

Line 4 F() in double loop : 5*5 = 25

Line 6 F() in single loop : 1*5 = 25

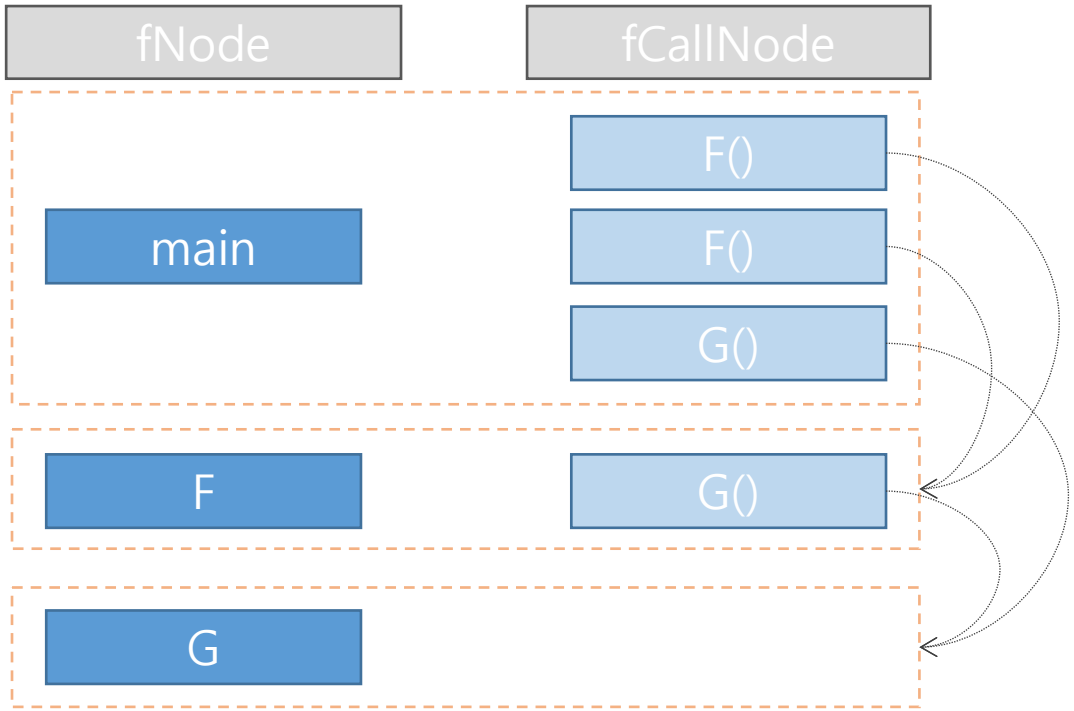
Line 8 G() in double loop : 1 = 1

Line 13 F() in loop in funcion: 3*5 = 15

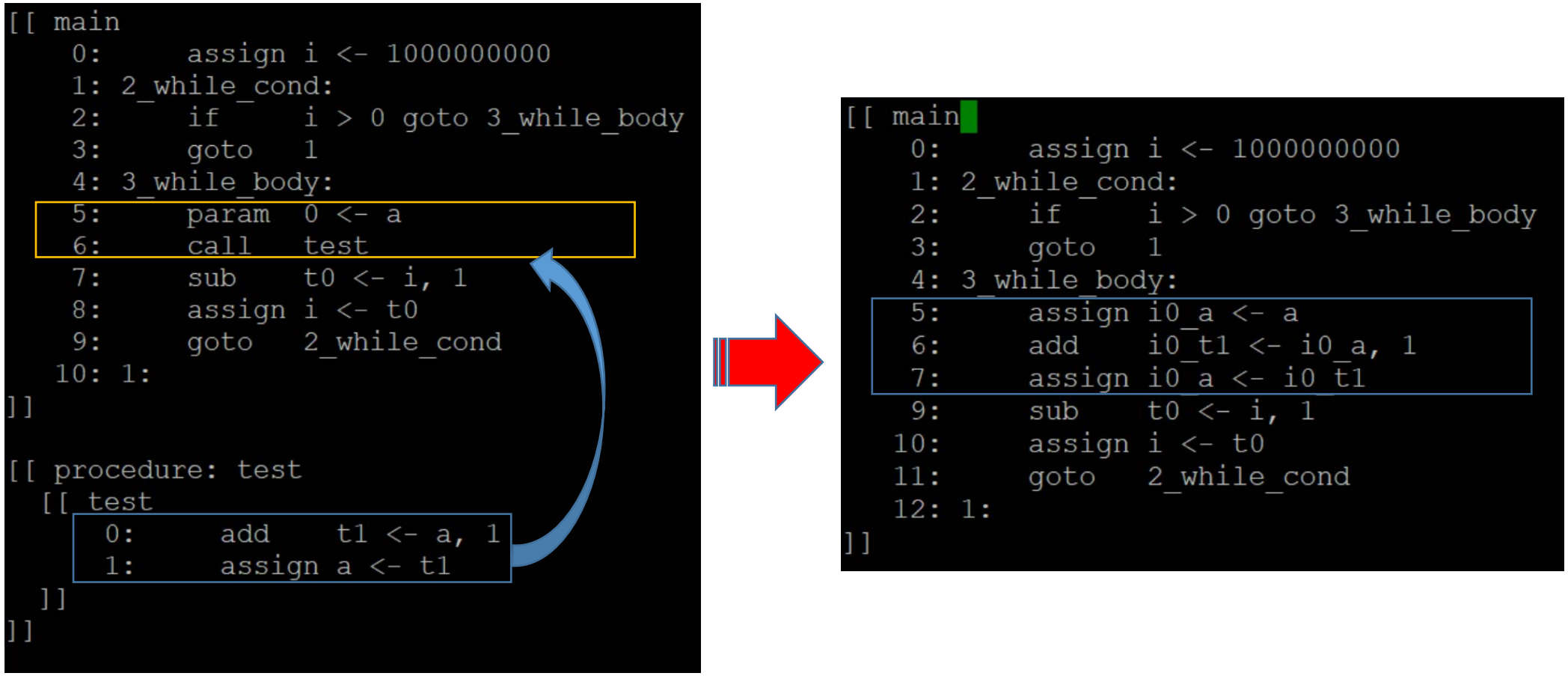
Data Structure

to process each function and each function call

```
class fNode {
    public:
        // member methods

    protected:
        int _code_length;
        vector<fCallNode*> _children;
        string _name;
        CScope* _module;
};

class fCallNode {
    public:
        // member methods

    protected:
        float _score;
        int _loop_level;
        fNode* _node;
        string _name;
};
```



function Inlining



Evaluation

```
[[ main
    0:     assign i <- 1000000000
    1: 2_while_cond:
    2:     if     i > 0 goto 3_while_body
    3:     goto   1
    4: 3_while_body:
    5:     param  0 <- a
    6:     call   test
    7:     sub    t0 <- i, 1
    8:     assign i <- t0
    9:     goto   2_while_cond
   10: 1:
]]

[[ procedure: test
  [[ test
      0:     add    t1 <- a, 1
      1:     assign a <- t1
  ]]
]]
```

```
[[ main
    0:     assign i <- 1000000000
    1: 2_while_cond:
    2:     if     i > 0 goto 3_while_body
    3:     goto   1
    4: 3_while_body:
    5:     assign i0_a <- a
    6:     add    i0_t1 <- i0_a, 1
    7:     assign i0_a <- i0_t1
    9:     sub    t0 <- i, 1
   10:     assign i <- t0
   11:     goto   2_while_cond
   12: 1:
]]
```

Compiler Optimization result with simple program

	Running time	TAC code size
compiled with basic snuplc	5064 ms	1281 byte
compiled with ours	3931 ms	1401 byte

Performance of running time : 29% ↑
TAC code size : 9% ↑

Schedule

Sep.	19	**Project Proposal**
	19 - 30	Studying SnuPL source codes & understanding SnuPL syntax & AST & TAC translations.
Oct.	1 - 24	Implementing the basic inline expansion of function calls. Additional ideation. Preparing the source file(mod) for testing, Making evaluation tool.
	25 - 29	Preparing presentation.
	29	**Milestone Presentation**
Nov.	1 - 17	Implementing dead code elimination & constant propagation
	19 - 30	Testing , finding bugs , Debugging , fixing
Dec.	1 - 9	Evaluating the final implementation & Preparing final presentation.
	10 – 12	**Final Presentation**
	19	**Project Report and Code**

Thanks