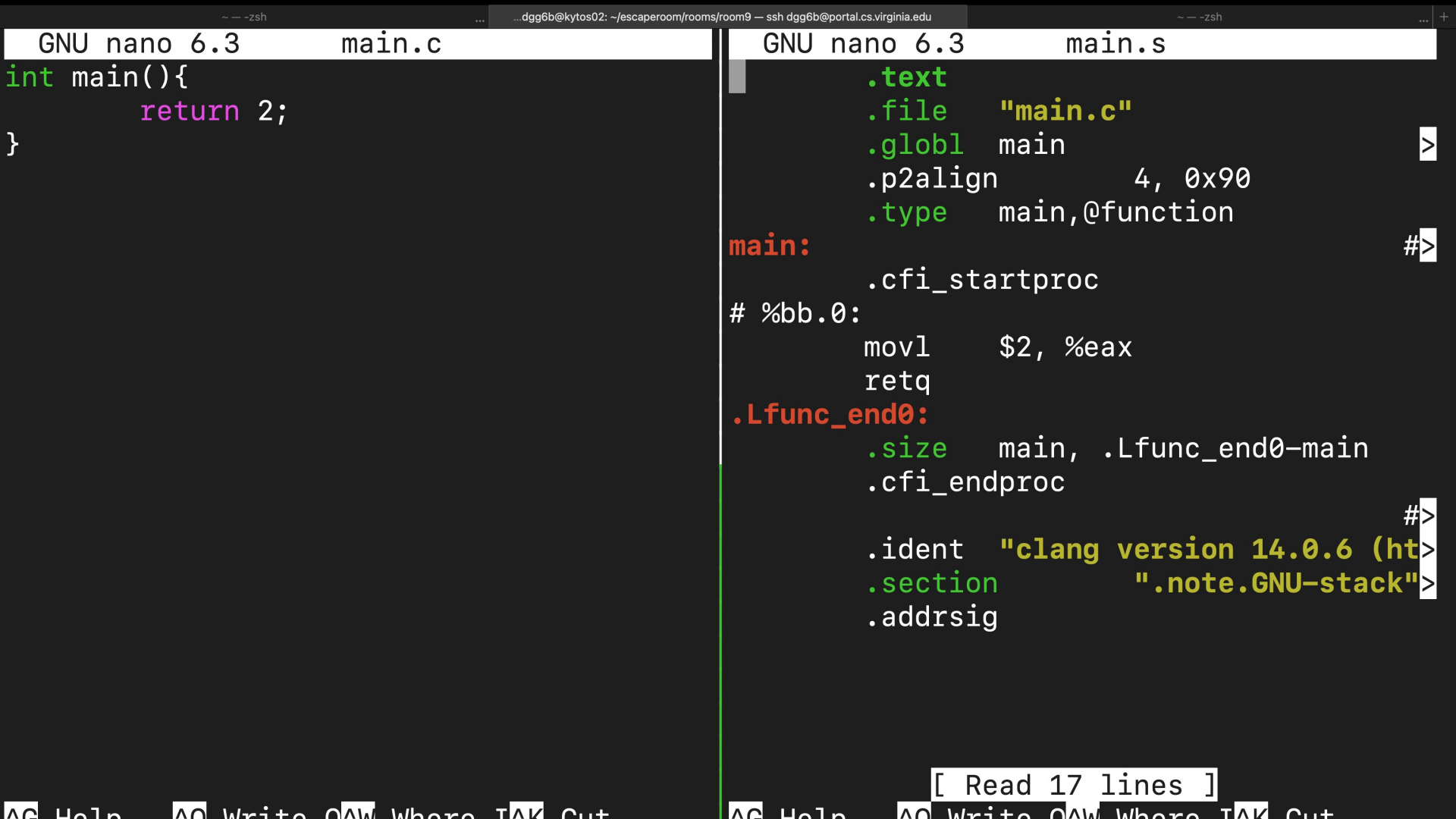
COMPUTER SYSTEMS AND ORGANIZATION

C compilation

Daniel G. Graham Ph.D

SIMPLE PROGRAM

```
int main() {
        return 7;
}
```


~ — -zsh
...dgg6b@kytos02: ~/escaperoom/rooms/room9 — ssh dgg6b@portal.cs.virginia.edu
~ — -zsh
GNU nano 6.3 main.c
int main(){
return 2;
}
GNU nano 6.3 main.s
.text
.file "main.c"
.globl main
.p2align 4, 0x90
.type main,@function
main:
.cfi_startproc
# %bb.0:
movl $2, %eax
retq
.Lfunc_end0:
.size main, .Lfunc_end0-main
.cfi_endproc
.ident "clang version 14.0.6 (ht
.section ".note.GNU-stack"
.addrsig
[Read 17 lines]
^G Help ^O Write ^W Where ^K Cut
^G Help ^O Write ^W Where ^K Cut

SIMPLE PROGRAM

```c
int main() {
    return 7;
}
```

```
        .text
        .file   "main.c"
        .globl  main
main:
        movl    $2, %eax
        retq
.Lfunc_end0:
        .size   main, .Lfunc_end0-main
        .cfi_endproc
```

SIMPLE PROGRAM

```c
int main() {
    return 7;
}
```

```
main:
    movl    $2, %eax
    retq
```

CAN WE BUILD SOMETHING REALLY SIMPLY THAT COMPILES THIS?

```
import sys, os

source_file = sys.argv[1]
assembly_file = os.path.splitext(source_file)[0] + ".s"
```

```
import sys, os

source_file = sys.argv[1]
assembly_file = os.path.splitext(source_file)[0] + ".s"

with open(source_file, 'r') as infile, open(assembly_file, 'w') as outfile:
    source = infile.read().strip()
```

```python
import sys, os

source_file = sys.argv[1]
assembly_file = os.path.splitext(source_file)[0] + ".s"

with open(source_file, 'r') as infile, open(assembly_file, 'w') as outfile:
    source = infile.read().strip()

    # Find the index of "int main()" and "return" in the source code
    main_start = source.find("int main()")
    return_start = source.find("return", main_start)
```

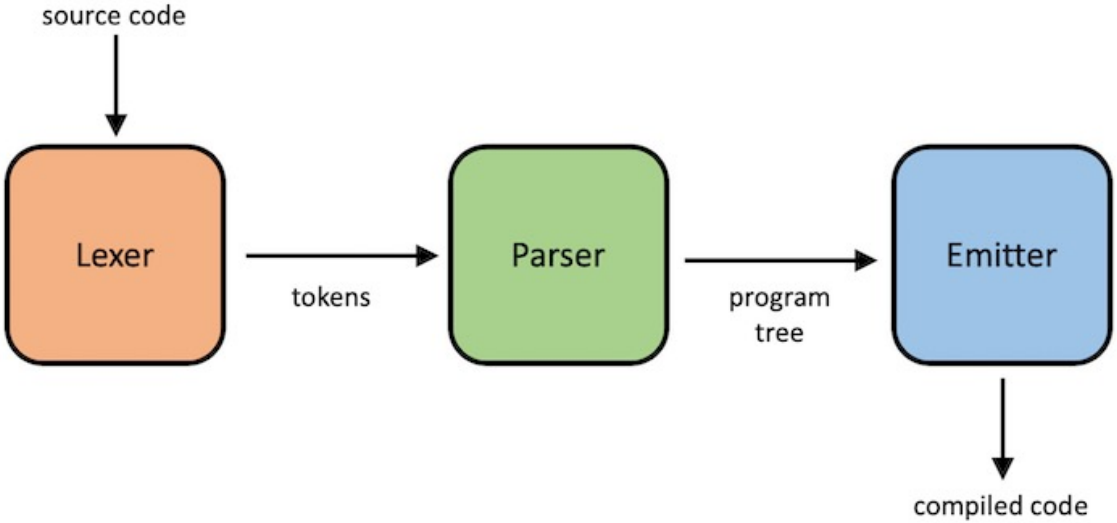
```python
import sys, os

source_file = sys.argv[1]
assembly_file = os.path.splitext(source_file)[0] + ".s"

with open(source_file, 'r') as infile, open(assembly_file, 'w') as outfile:
    source = infile.read().strip()

    # Find the index of "int main()" and "return" in the source code
    main_start = source.find("int main()")
    return_start = source.find("return", main_start)

    if main_start != -1 and return_start != -1:
        # Extract the return value
        return_value = source[return_start + 6:].strip().rstrip(";\n}")
```

```python
import sys, os

source_file = sys.argv[1]
assembly_file = os.path.splitext(source_file)[0] + ".s"

with open(source_file, 'r') as infile, open(assembly_file, 'w') as outfile:
    source = infile.read().strip()

    # Find the index of "int main()" and "return" in the source code
    main_start = source.find("int main()")
    return_start = source.find("return", main_start)

    if main_start != -1 and return_start != -1:
        # Extract the return value
        return_value = source[return_start + 6:].strip().rstrip(";\n}")
        # Write the assembly code to the output file
        assembly_code = f"""
        .globl main
        main:
        movl ${return_value}, %eax
        ret
        """
        outfile.write(assembly_code)
    else:
        print("Error: Couldn't find 'int main()' or 'return' in the source code.")
```

THIS DOESN’T SCALE TO MORE COMPLEX PROGRAMS

THE PROCESS OVERVIEW

THE PROCESS OVERVIEW



STEP 1: LEXING/SCANNING TOKENING

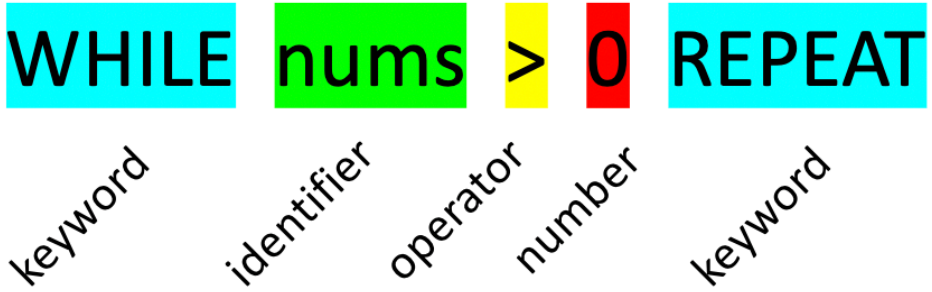
Raw text: WHILE nums > 0 REPEAT

Tokens: WHILE nums > 0 REPEAT

TOKENS HAVE MEANINGS

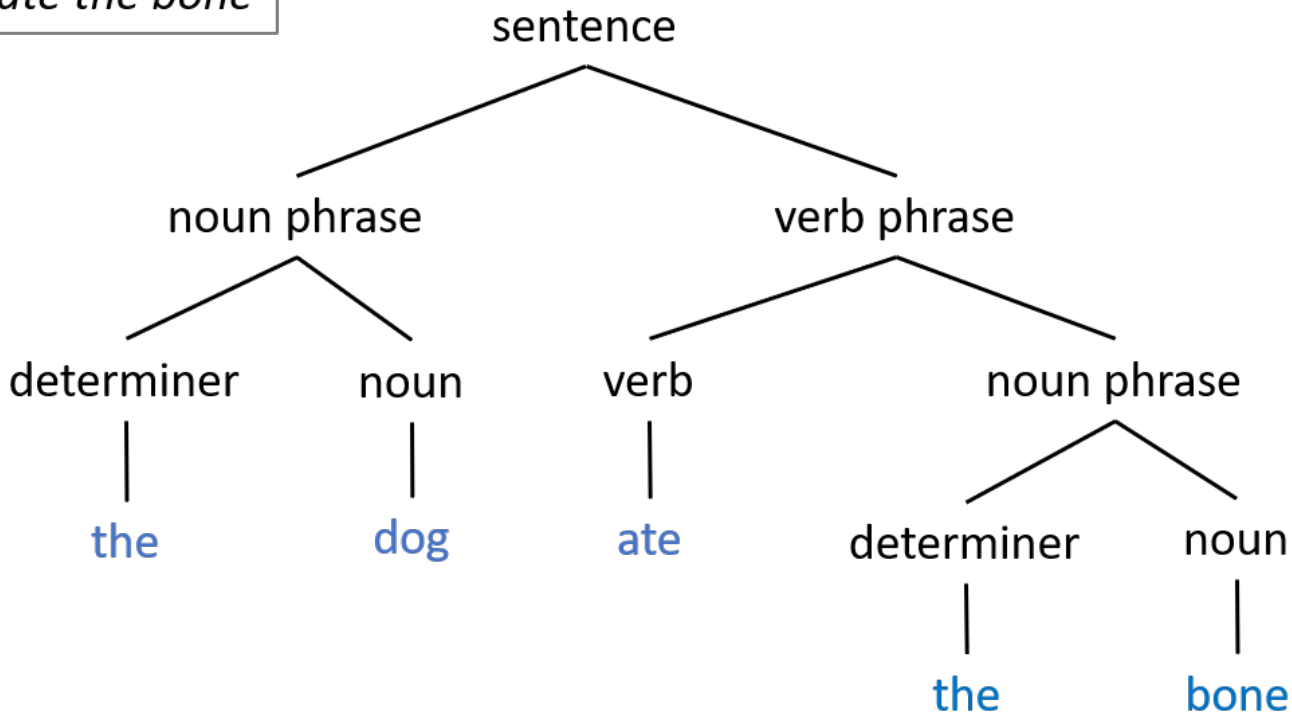
Raw text: WHILE nums > 0 REPEAT

Tokens:

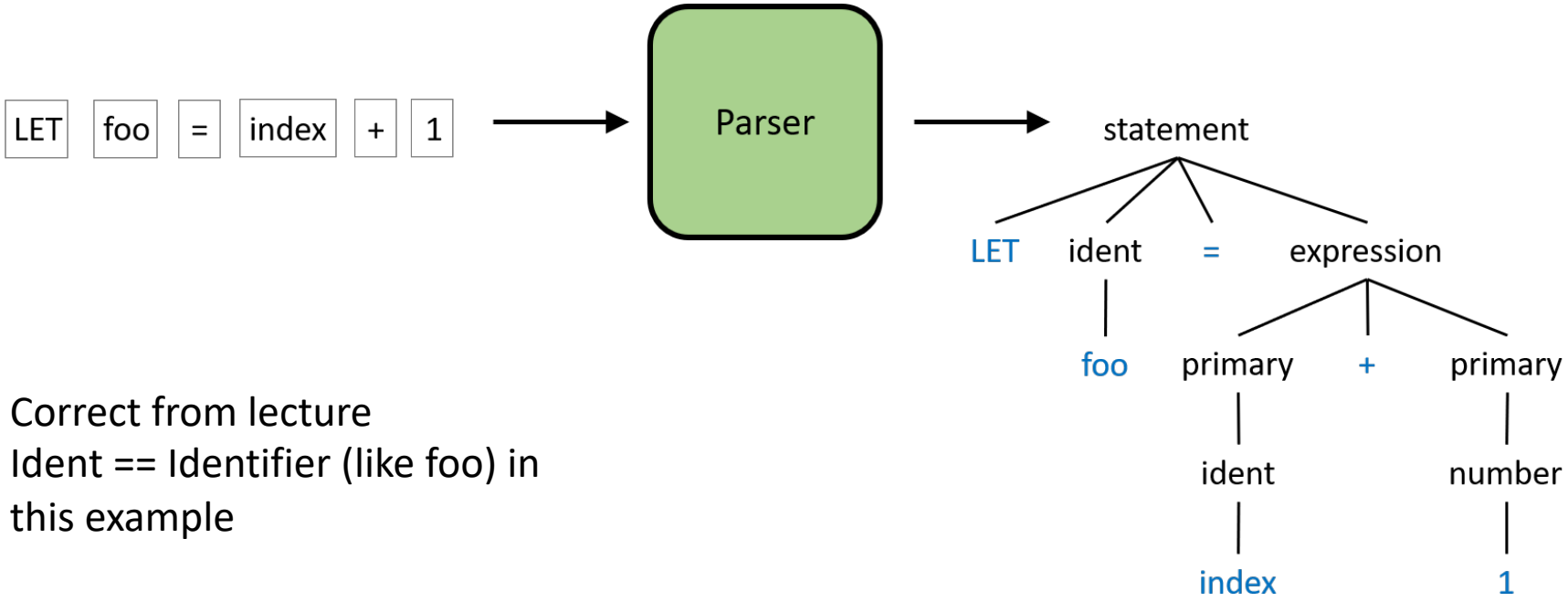


STEP 2: PARSING

the dog ate the bone



STEP 2: PARSING



Correct from lecture
Ident == Identifier (like foo) in this example

Not indent** Sorry

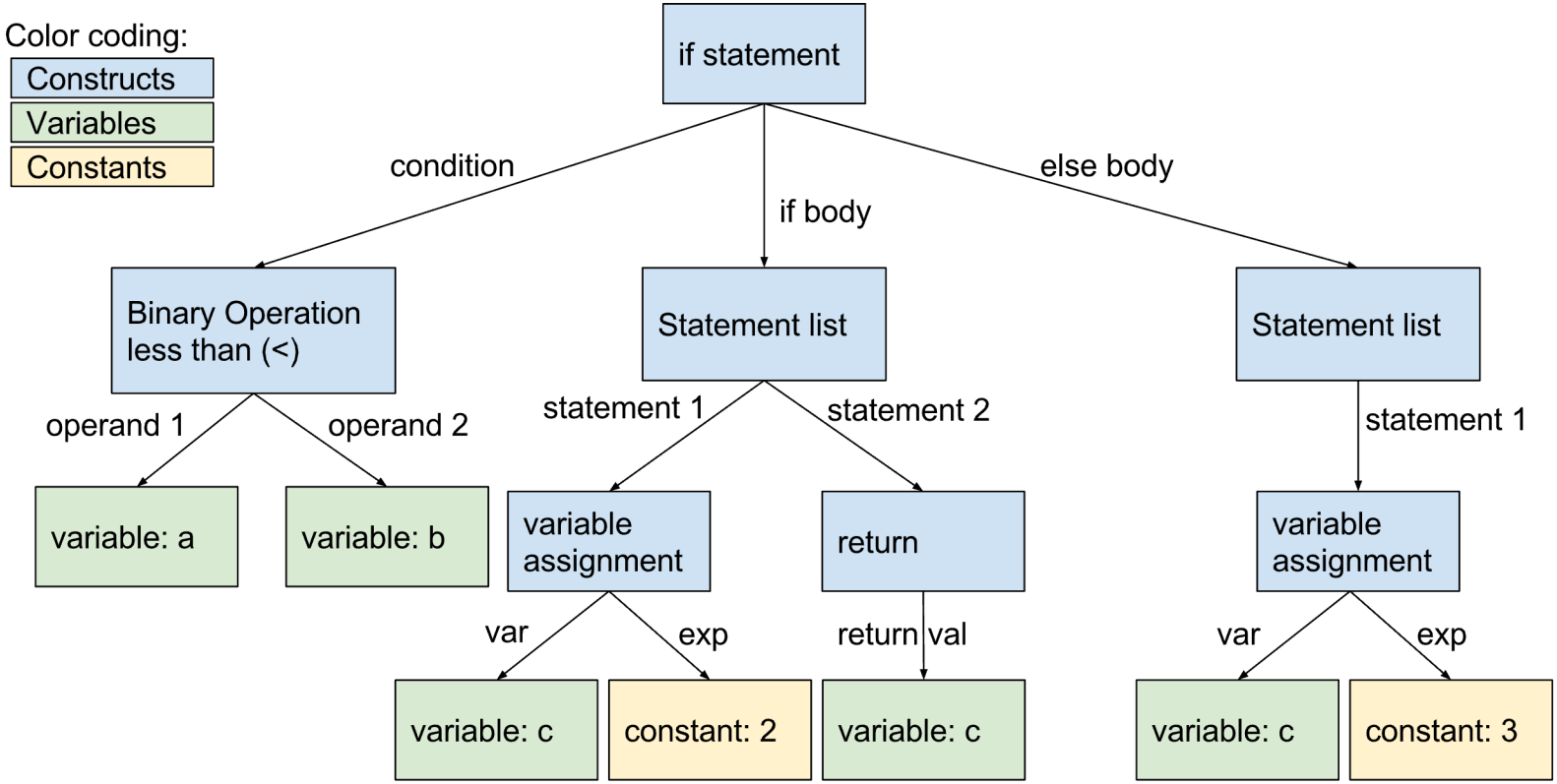
STEP 2: PARSING

```
if (a < b) {
      c = 2;
       return c;
 } else {
      c = 3;
 }
```

- The condition (a < b)
- The if body (c = 2; return c;)
- The else body (c = 3;)

- Can be further broken down
  - The first operand (variable a)
  - The second operand (variable b)

```
if (a < b) {
        c = 2;
        return c;
} else {
        c = 3;
}
```



BACK-NAUS FORM

<program> ::= <function>
<function> ::= "int" <id> "(" ")" "{" <statement> "}”
 <statement> ::= "return" <exp> ";"
<exp> ::= <int>

CODE GENERATION

HOW DO WE GO FROM OUR AST TO ASSEMBLY

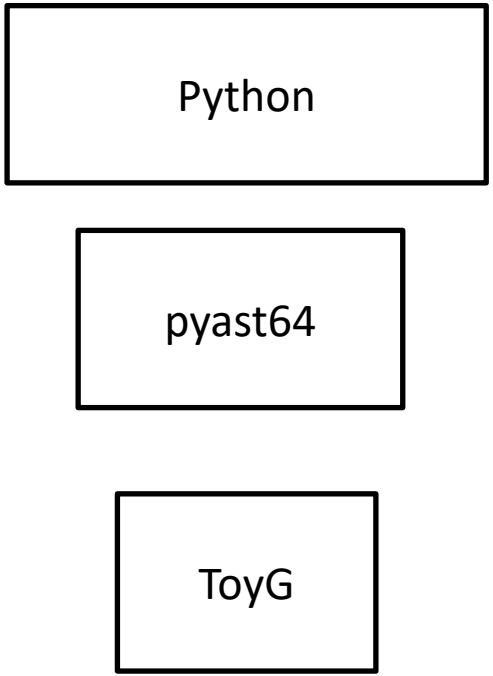
LET’S DESIGN OUR OWN LITTLE LANGUAGE

HOW ABOUT TOYC

EVEN BETTER WHAT ABOUT TOYG

WE'LL MAKE A SUBSET OF PYTHON

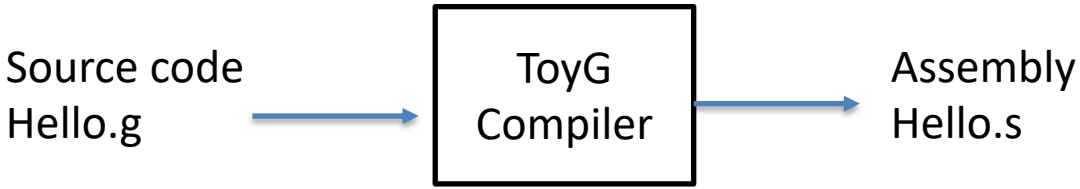
WE'LL MAKE IT A SUBSET OF PYTHON



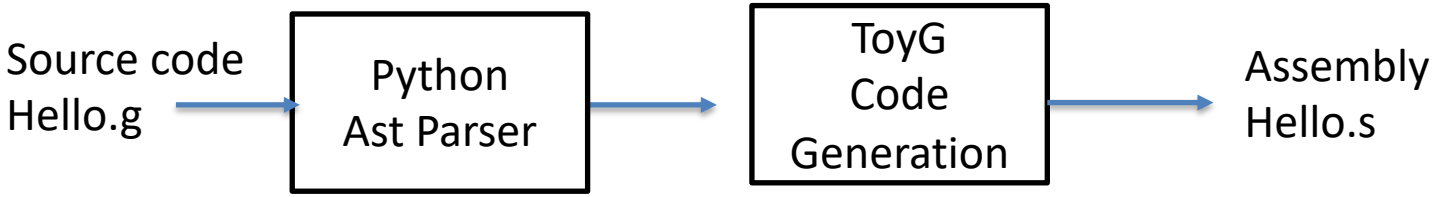
Instead of be interpreted like python.

Our language will be compiled meaning that will write a compiler that outputs assembly

THE PROCESS



LEVERAGE THE PYTHON AST PARSER



Common to use framework when developing your own language for example LLVM has library and support for developing your own lexer and parsers

SEE ZIP FOR CODE IMPLEMENTATIONS

REFERENCES

- https://norasandler.com/2017/11/29/Write-a-Compiler.html
- Abdulaziz Ghuloum's An Incremental Approach to Compiler Construction
- https://benhoyt.com/writings/pyast64/