COMPILER DESIGN

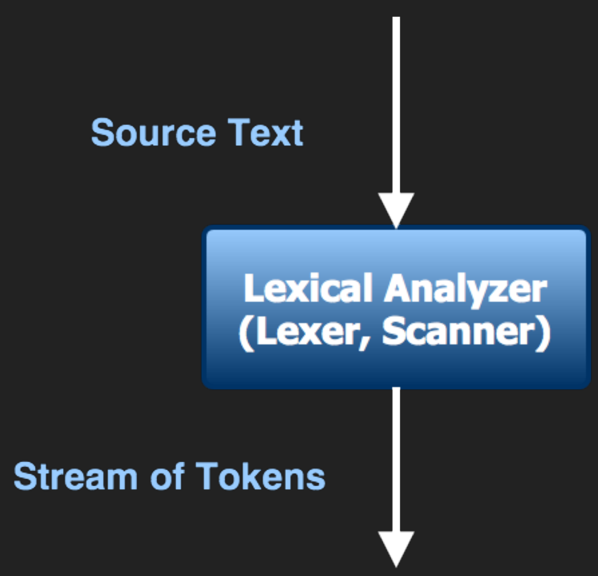
AND IMPLEMENTATION

ÁRPÁD GORETITY
BUDAPEST SWIFT MEETUP

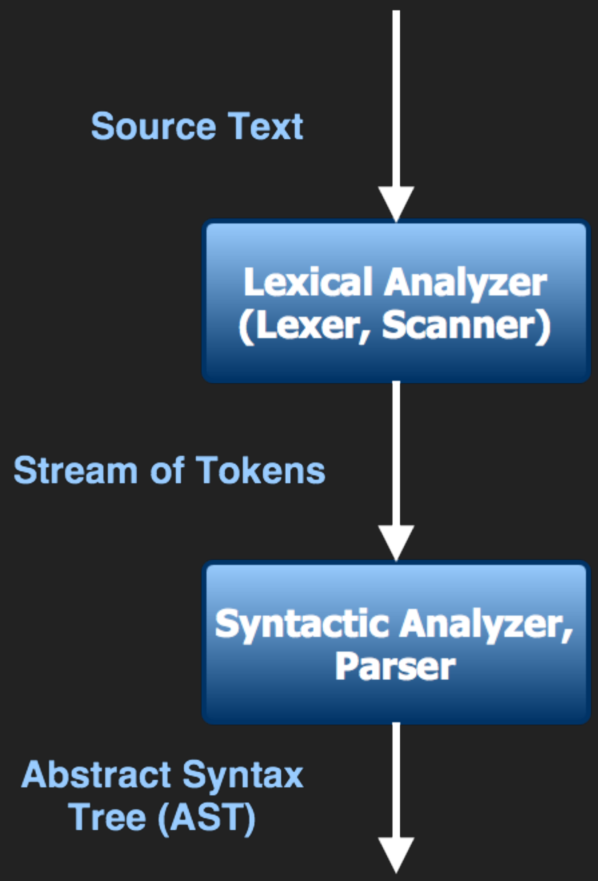
SEMANTIC ANALYSIS

PART 3

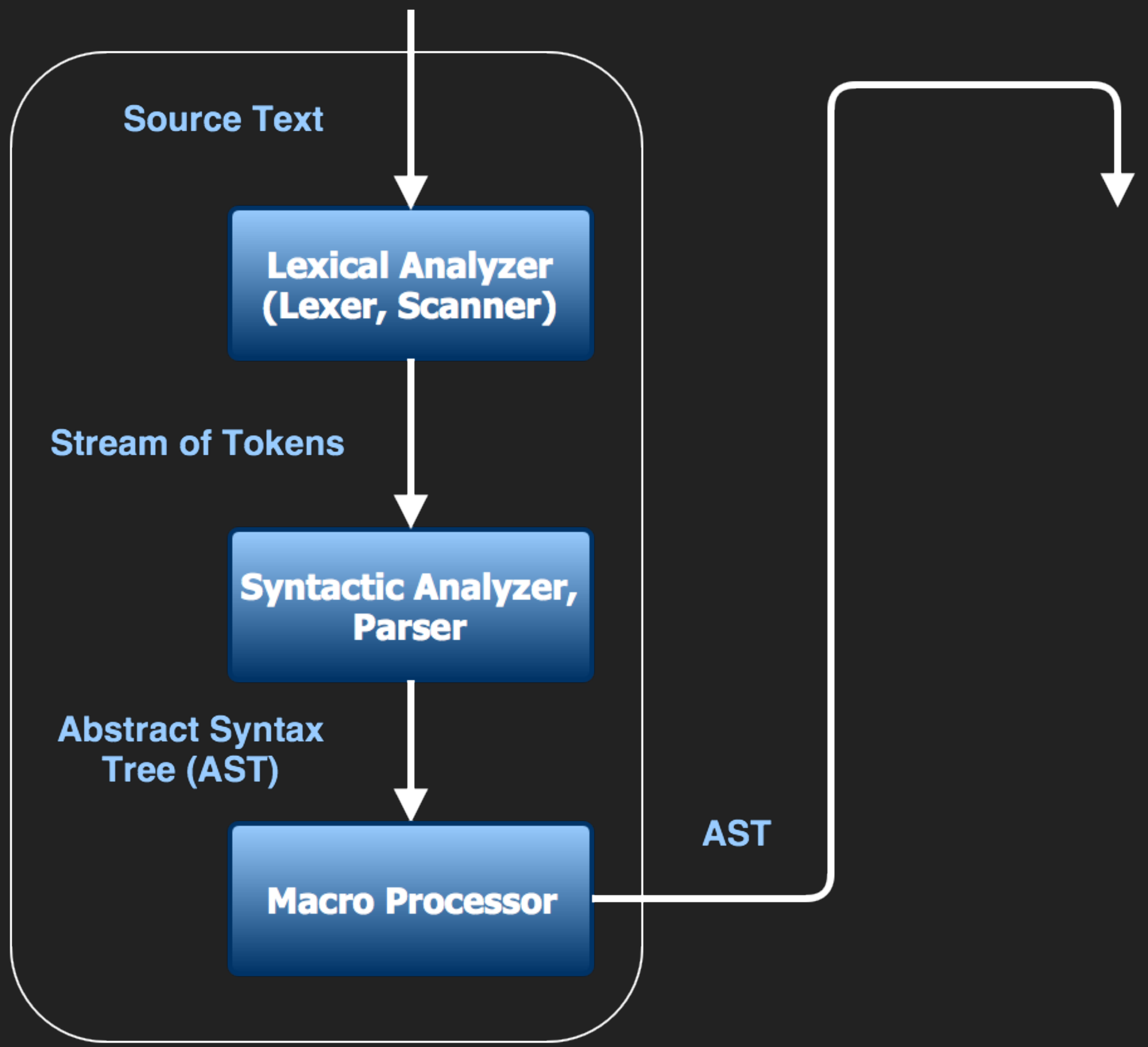
REMINDER


Source Text
Lexical Analyzer
(Lexer, Scanner)
Stream of Tokens

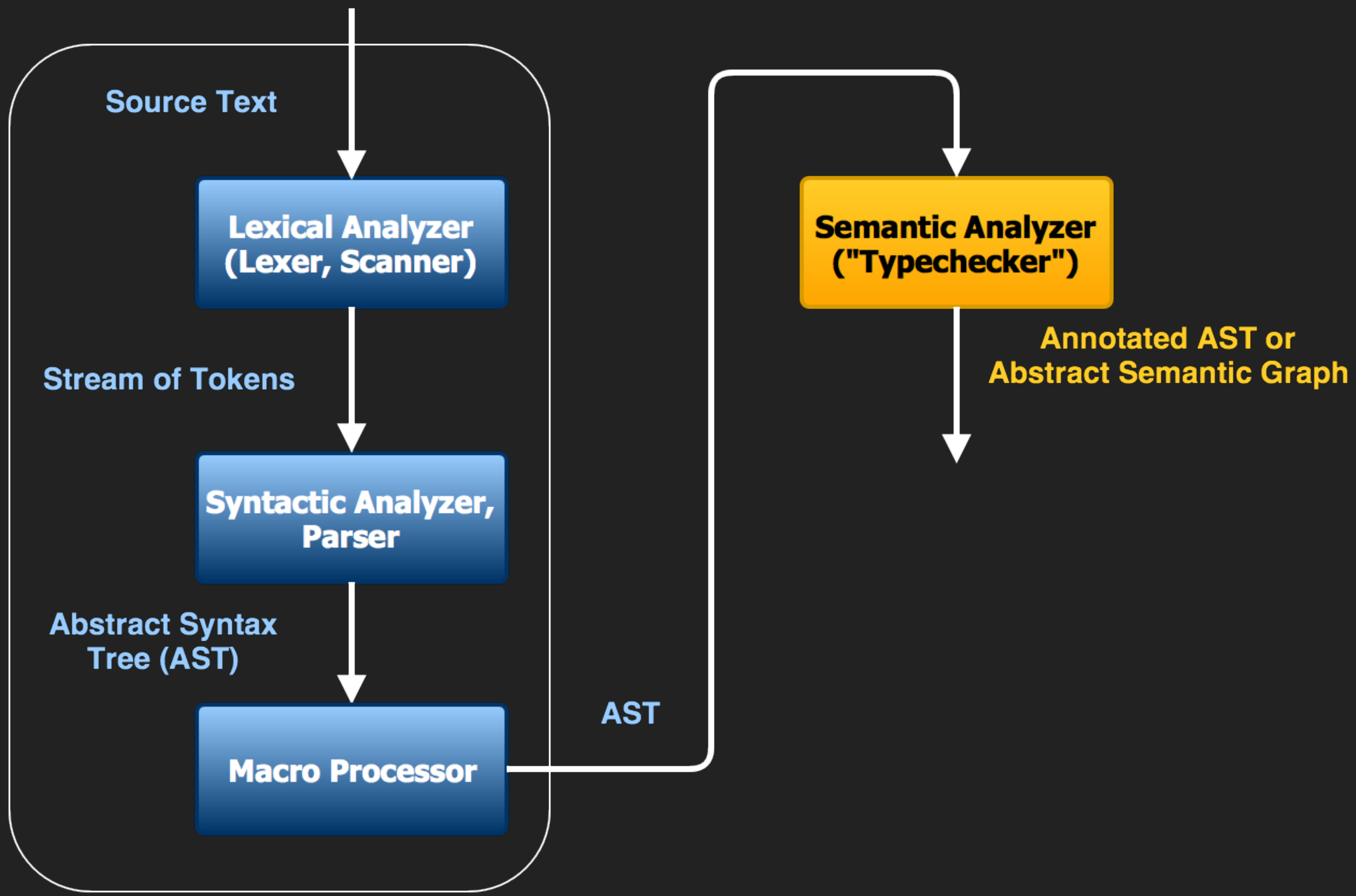
REMINDER


Source Text
Lexical Analyzer
(Lexer, Scanner)
Stream of Tokens
Syntactic Analyzer,
Parser
Abstract Syntax
Tree (AST)

REMINDER


Source Text
Lexical Analyzer
(Lexer, Scanner)
Stream of Tokens
Syntactic Analyzer,
Parser
Abstract Syntax
Tree (AST)
Macro Processor
AST

REMINDER


Source Text
Lexical Analyzer
(Lexer, Scanner)
Stream of Tokens
Syntactic Analyzer,
Parser
Abstract Syntax
Tree (AST)
Macro Processor
AST
Semantic Analyzer
("Typechecker")
Annotated AST or
Abstract Semantic Graph

WHY DO WE NEED A SEMANTIC ANALYZER?

- Correct Syntax != Correct Program
- “Colorless green ideas sleep furiously” (Chomsky)
- Syntax: OK – verb, subject, adverbs in the right place
- Meaning: not OK – complete nonsense

WHY DO WE NEED A SEMANTIC ANALYZER?

- Reminder: **Context-Free** Grammars
- Parser does not care about the meaning of symbols
- But we do! That is **the** point of programming…
- We need something to check correctness of meaning

KINDS OF SEMANTIC ANALYSIS

- Type Checking and/or Type Inference
- Scope Resolution
- Data Flow Analysis (DFA)
- Control Flow Analysis (CFA)
- …and possibly many more…

STEPS OF SEMANTIC ANALYSIS

- These steps are strongly interrelated
- Type checking and scope resolution: Typing Context
- Scope resolution and DFA:
  - Which constants and variables are accessible where?
  - When are pointers valid? – Lifetimes (Rust's affine types)
- DFA and CFA: which computation produces which value? Warnings: unused variable (DFA), unreachable code (CFA)

THEORETICAL BACKGROUND

- Formal logic
  - Lambda calculus
- Formal semantics: Denotational, Operational
- Type theory
  - Curry-Howard correspondence
  - Formal Type Systems (e.g. Hindley–Milner)
- DFA, CFA: Graph theory and algorithms (DFG, CFG)

THEORETICAL BACKGROUND

- Each of these subjects may span multiple semesters if you do a Computer Science degree at university…
- This is **not** an attempt to equip you with such detailed knowledge
- Focus is on some basic implementation techniques
  - Scope Resolution and Type Checking – **inevitable**
  - Type Inference – **local** inference is easy

THEORETICAL BACKGROUND

- DFA, CFA: lengthy, sometimes hard
  - So not today
  - **Except:** loop control flow statements (`break`, `continue`)
    - These are trivial to check for basic correctness

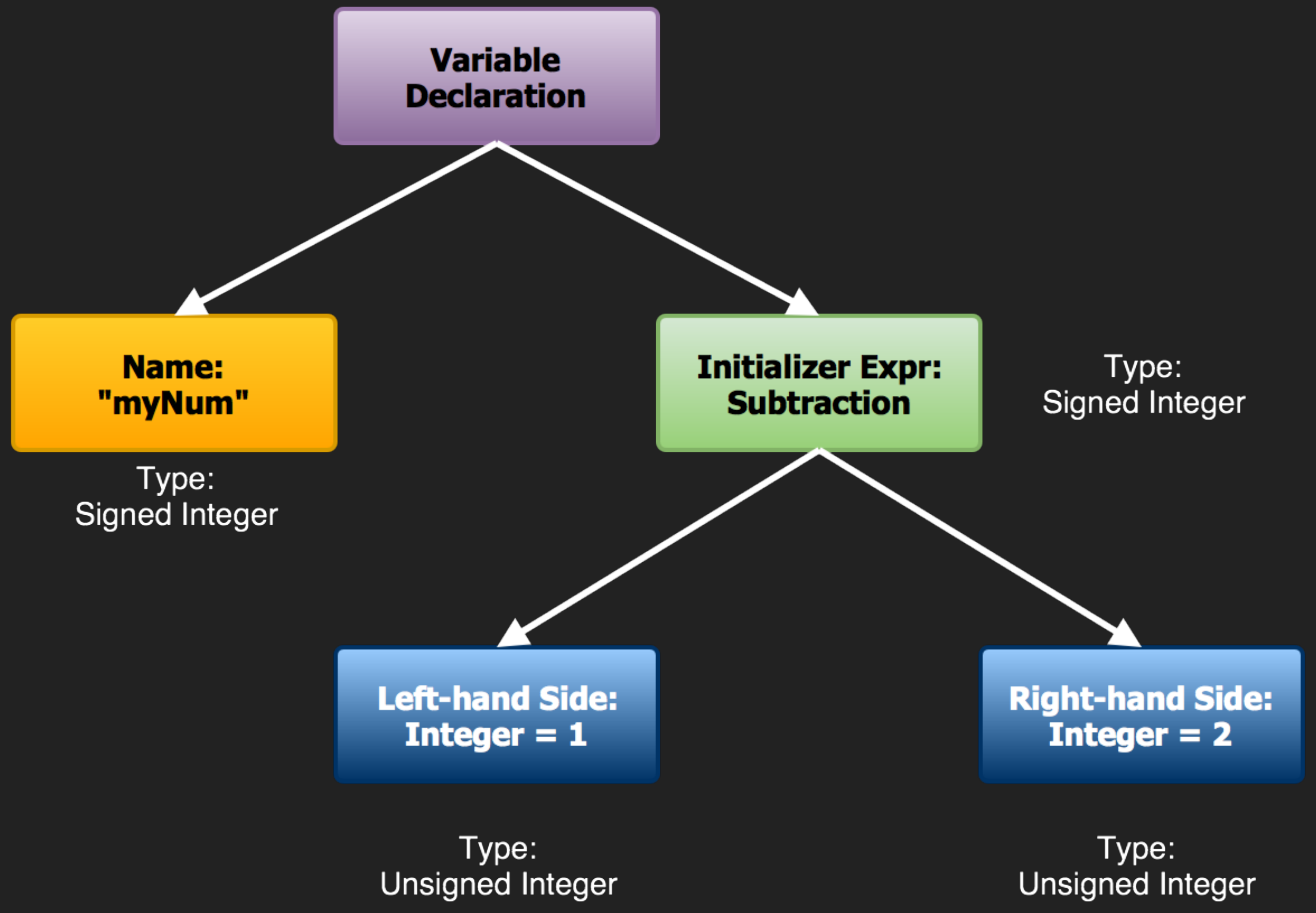
REPRESENTATION OF SEMANTICS – AN EXAMPLE

- **Scopes**
  - Stack of sets containing variable names (or indices)
  - Search starts at top of stack, going downwards

```
{                      // []
  var a = 42           // [('a')]
  {
    var b              // [('a'), ('b')]
    var c              // [('a'), ('b', 'c')]
  }                    // [('a')]
}                      // []
```

REPRESENTATION OF SEMANTICS – AN EXAMPLE

- **Types:** Type annotations on AST nodes



LOCAL TYPE INFERENCE

- Relatively easy: similar to naive code generation
- Only needs e.g. recursive traversal of the AST
- Bottom-up: types propagate from leaves (children) towards the root (parent)
  - e.g. basic mathematical operators, return value of function calls
- Top-down: types propagate from parents towards children
  - e.g.: argument types of a function literal passed to a higher-order function

LET'S GET TO WORK!