# CSC488/2107 Winter 2019 — Compilers & Interpreters

https://www.cs.toronto.edu/~csc488h/

Peter McCormick
pdm@cs.toronto.edu

# Agenda

- What is a compiler? Why study compilers?
- Compiler techniques
- Traditional compilers vs the modern dynamic world
- What does a compiler do?
- Syllabus

# What is a compiler?

*“A compiler is a program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer”*

*Source: Google*

*"A compiler is a program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer"*

A compiler is *program* that recognizes program code written in a source language, analyzes it and ultimately transforms it into a target representation.

Typically the transformation moves from a higher level to a lower level of abstraction.

1. Recognize
2. Analyze
3. Transform / Lower

# Why study compilers?

# *Unwrapping abstractions*

# *Developing new languages*

# *A mature toolbox of technology*

# Compiled languages

*vs*

# Scripting languages

# Compiled language

Source Code

*reads file*

Compiler

*writes file*

Executable Binary

hello.c

*reads file*

```
gcc hello.c -o hello
```

*writes file*

./hello

# “*Scripting*” language

hello.py

*reads file*

```
python hello.py
```

*produces output*

“Hello World!”

# Real *vs* virtual/ pseudo machines

# Perspective on programming languages

# Machine-centric

*Harnessing the power of the machine*

# Application-centric

*Enabling users to solve their problems*

- Perl
- Python
- Ruby
- JavaScript*
- C
- C++
- Java*
- Go
- Haskell

# Compiler *techniques*

# Editors / IDEs

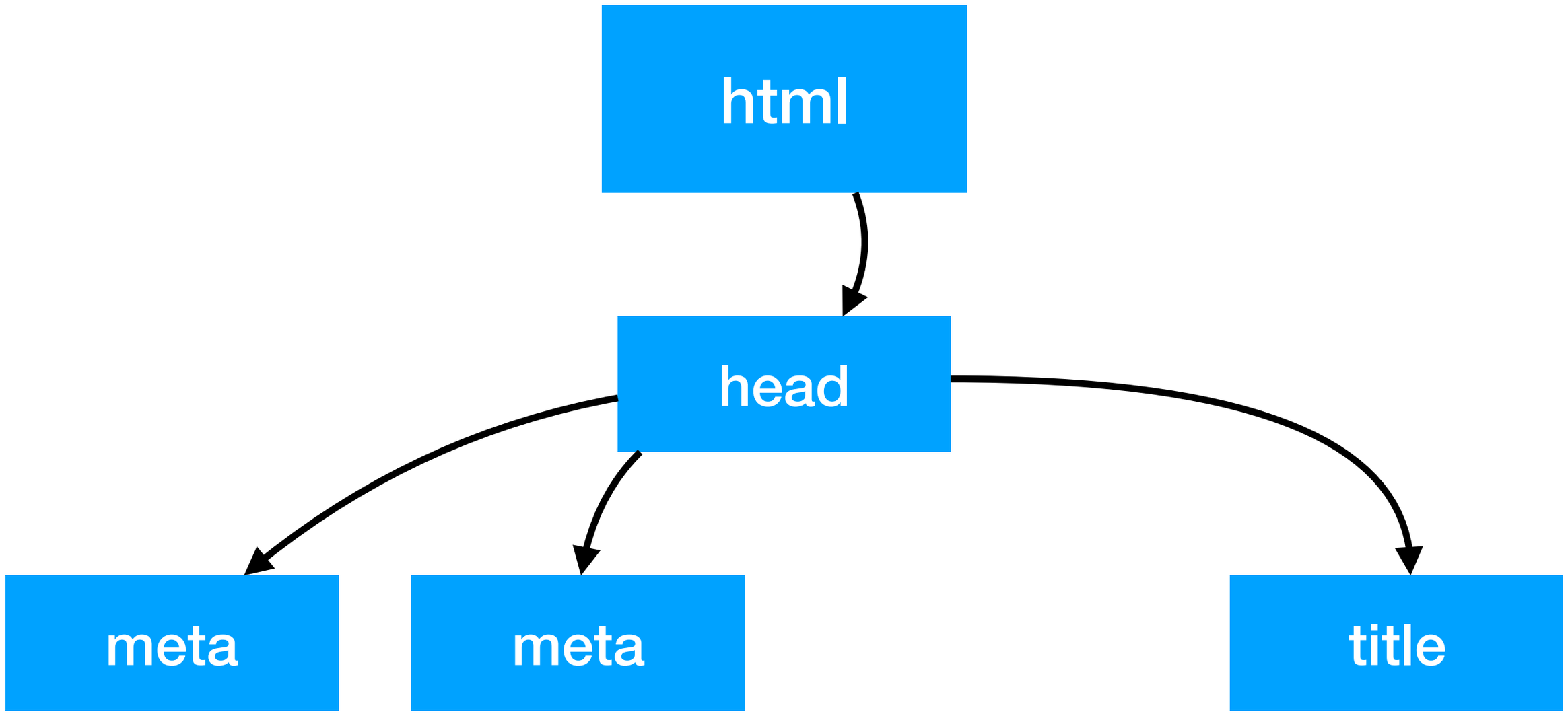
```
struct {
   int x;
} S;
S.x = 488;

if (S.x > 400) {
```

```
struct {
    int x;
} S;

S.x = 488;

if (S.x > 400) {
```

```
struct {
    int x;
} S;

S.x = 488;

if (S.□
```

# Web

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initia
        <meta name="description" content="">
        <meta name="author" content="Peter McCormick">

        <title>CSC488/2107 Winter 2019 &mdash; Syllabus</title>
```


html
head
meta
meta
title

```
componentDidMount() {
  var stopper = this.props.text + '...';
  this.interval = window.setInterval(function () {
    if (this.state.text === stopper) {
      this.setState(function () {
        return {
          text: this.props.text
        }
      })
    } else {
      this.setState(function (prevState) {
        return {
          text: prevState.text + '.'
        }
      });
    }
}.bind(this), this.props.speed)
```

```
"use strict";this.social_NotificationsOgbUi=this.social_Notifications
try{
var baa,caa,daa,eaa,faa,aaa,gaa,haa,ra,kaa,va,wa,laa,maa,naa,oaa;_.aa
eaa=function(a,b){return baa(a)?_.ka(b,function(){return _.la(null)})
_.pa=function(a){var b="undefined"!=typeof window.Symbol&&window.Symb
if("function"==typeof Object.setPrototypeOf)ra=Object.setPrototypeOf;
_.m=function(a,b){a.prototype=haa(b.prototype);a.prototype.constructo
va="function"==typeof Object.defineProperties?Object.defineProperty:f
wa("Promise",function(a){function b(){this.H=null}function c(a){retur
})};var e=function(a){this.R=0;this.T=void 0;this.H=[];var b=this.V()
function(a){var b=void 0;try{b=a.then}catch(n){this.W(n);return}"func
b){var c=this.V();try{a.call(b,c.resolve,c.reject)}catch(v){c.reject(
c;e.reject=function(a){return new e(function(b,c){c(a)})};e.race=func
maa=function(){var a=0;return function(b){return"jscomp_symbol_"+(b||
oaa=function(a,b,c){a instanceof String&&(a=String(a));for(var d=a.le
wa("String.prototype.endsWith",function(a){return a?a:function(a,c){v
var za=function(a,b){return Object.prototype.hasOwnProperty.call(a,b)
wa("WeakMap",function(a){function b(){}function c(a){za(a,e)||va(a,e,
var g=0,k=function(a){this.H=(g+=Math.random()+1).toString();if(a){a=
wa("Object.is",function(a){return a?a:function(a,c){return a===c?0!==
wa("Object.values",function(a){return a?a:function(a){var b=[],d;for(
wa("Map",function(a){if(function(){if(!a||"function"!=typeof a||!a.pr
{};this.H=g();this.size=0;if(a){a=_.pa(a);for(var b;!(b=a.next()).don
a.Nh.next.Sk=a.Nh.Sk,a.Nh.head=null,this.size--,!0):!1};c.prototype.c
this.entries(),d;!(d=c.next()).done;)d=d.value,a.call(b,d[1],d[0],thi
c.head;)return c=c.next,{done:!1,value:b(c)};c=null}return{done:!0,va
```

# Databases

```
SELECT *
FROM students
WHERE
    loves488=1
```

```
EXPLAIN SELECT sum(i) FROM
foo WHERE i < 10;

QUERY PLAN
──────────

Aggregate (cost=23.93..23.93 rows=1 width=4)
    ->  Index Scan using fi on foo
          (cost=0.00..23.92 rows=6 width=4)
          Index Cond: (i < 10)
```

# PDF and PostScript

# Compiler techniques

- Processing of highly structured text
- Structural analysis
- Source-to-source transformation
- Test coverage
- Execution optimization

# What does a compiler do?

# Recognize

- Recognizes correct **lexical composition**
    - Correctly utilizes spaces, identifiers, operators, etc.
- Recognizes correct **syntax**
    - Obeys the rules for variable/function definitions, statements, expressions, etc.

```
if g:
y = 488
else:
y = 2107
```

```
b = x >>> 100;
```

```
b = x >>> 100;
```

$$v = 488x;$$

```
v = 488x;
```

```
if (b) x = 1;
else x = 2;
else x = 3;
```

```
if (b) x = 1;
else x = 2;
else x = 3;
```

```
x =
    if (b) y = 2;
```

```
x =
    if (b) y = 2;
```

# Recognize

- Recognizes correct **lexical composition**
    - Correctly utilizes spaces, identifiers, operators, etc.
- Recognizes correct **syntax**
    - Obeys the rules for variable/function definitions, statements, expressions, etc.

# Analyze

- Analyzes programs for *sensibility*
  - All definitions are valid
  - All *uses* correspond to matching definitions
- Keep track of what relates to what

```
int x;
itn y;
```

```
int x;
itn y;
```

```
typedef itn int;

int x;
itn y;
```

```
typedef int itn;

int x;
itn y;
```

```
int x;

if (x[488] > 0) …
```

```
int x;

if (x[488] > 0) …
```

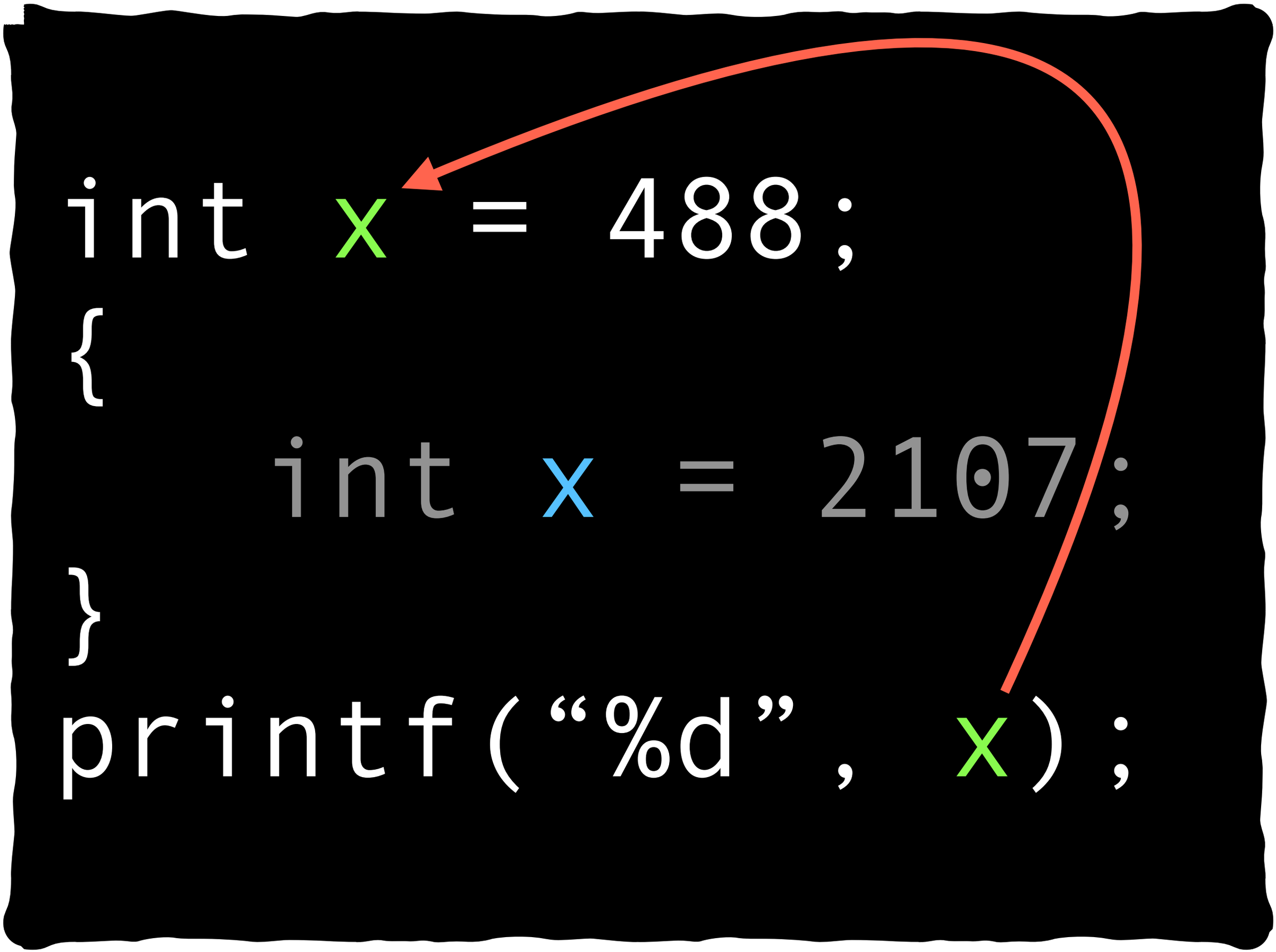
```
int y[488];

if (y > 0) …
```

```
int y[488];

if (y[ ] > 0) …
```

```
int y[488];

if (y[0] > 0) …
```

```
int zs[488];

// zs uninitialized
if (zs[0] > 0) …
```

```
int x = 488;
{
    int x = 2107;
}
printf(“%d”, x);
```


int x = 488;
{
int x = 2107;
}
printf("%d", x);

# Analyze

- Analyzes programs for *sensibility*
  - All definitions are valid
  - All *uses* correspond to matching definitions
- Keep track of what relates to what

# Transform / Lower

- Transform tree/graph structures into other kinds of tree/graph structures

- Produce *equivalent** machine code or virtual machine bytecode

- Apply *legal** transformations to optimize performance

```
int x = 400 + 88;
printf("%d", x);
```

# Constant Folding

```
int x = 488;
printf("%d", x);
```

# Constant Propagation

```
int x = 488;
printf("%d", 488);
```

# Dead Definition Elimination

~~int x = 488;~~

```
printf(“%d”, 488);
```

# Final

```
printf("%d", 488);
```

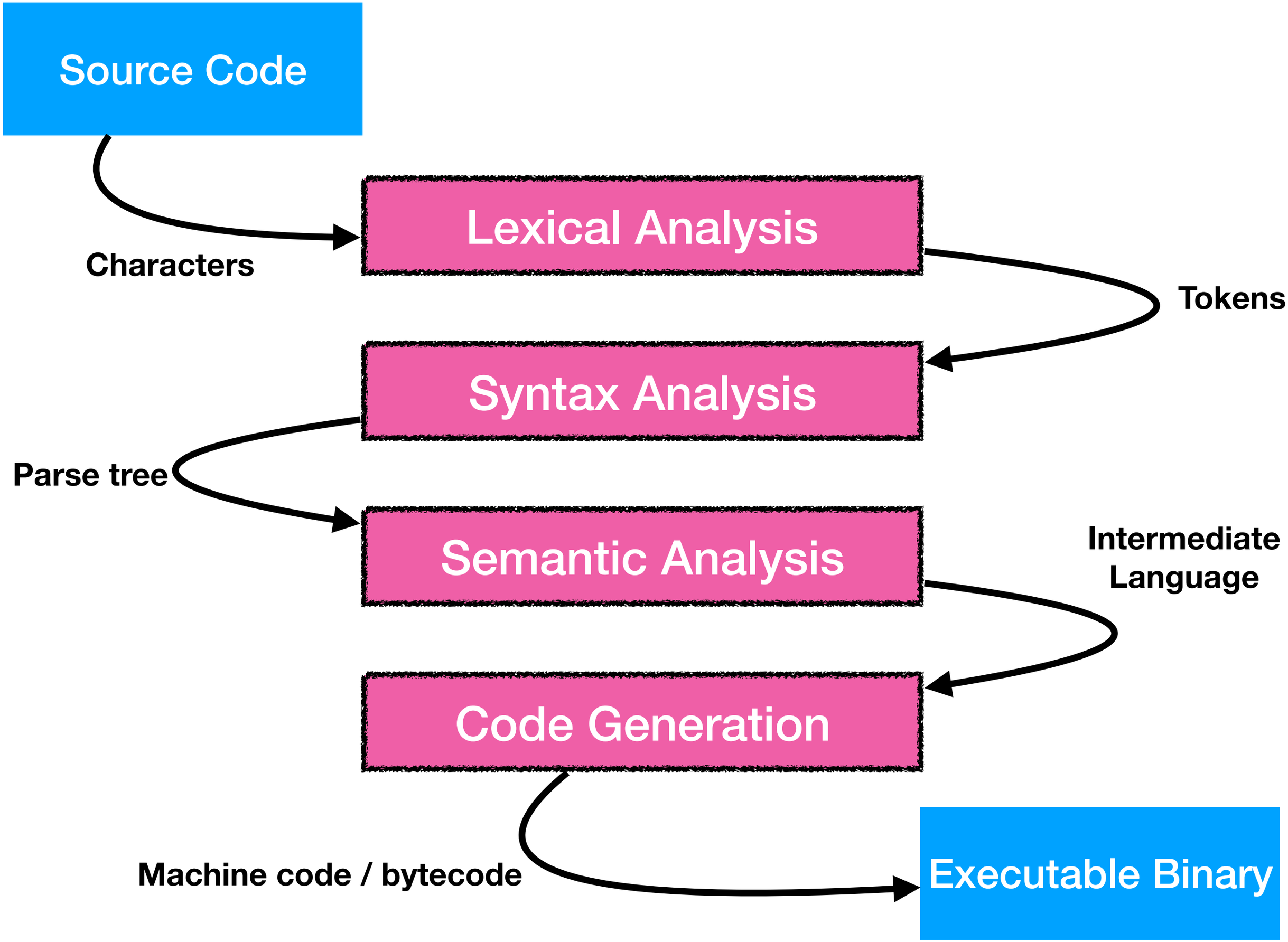
# Transform / Lower

- Transform tree/graph structures into other kinds of tree/graph structures

- Produce *equivalent** machine code or virtual machine bytecode

- Apply *legal** transformations to optimize performance

# Compiler pipeline


Source Code
Characters
Lexical Analysis
Tokens
Syntax Analysis
Parse tree
Semantic Analysis
Intermediate Language
Code Generation
Machine code / bytecode
Executable Binary

```
if x < y { v = 1 }
```

```
if x < y { v = 1 }
```

**reserved** if

**identifier** x

**operator** <

**identifier** y

**left brace**

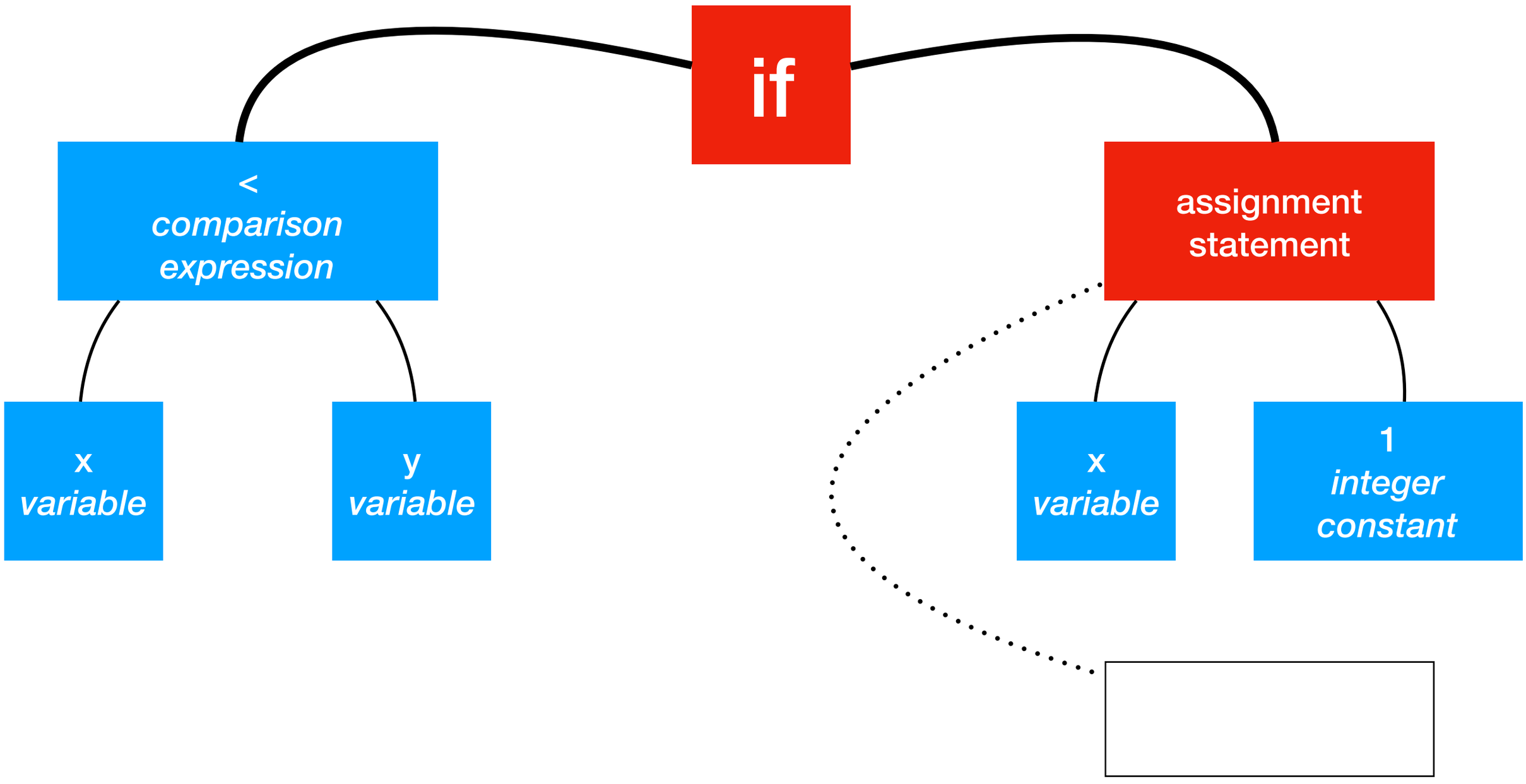
**identifier** v

**operator** =

**integer** 1

**right brace**

```
if x < y { v = 1 }
```


if
<
comparison
expression
x
variable
y
variable
assignment
statement
x
variable
1
integer
constant

```
if x < y { v = 1 }
```

```
  load r1, xl
  load r2, y
  lt r1, r2, r3
  jneq _end
  const r1, 1
  loadaddr r2, v
  store r2, r1
_end:
```

# Syllabus

# Office Hours

3-5pm in BA2283 immediately after lecture

# Is CSC488/2107 for you?

# Next Week

- More on the compiler pipeline
- Lexical analysis

*No tutorial on Tuesday Jan 15*

*Website correction:*
*Tutorials _are_ 2-3pm*