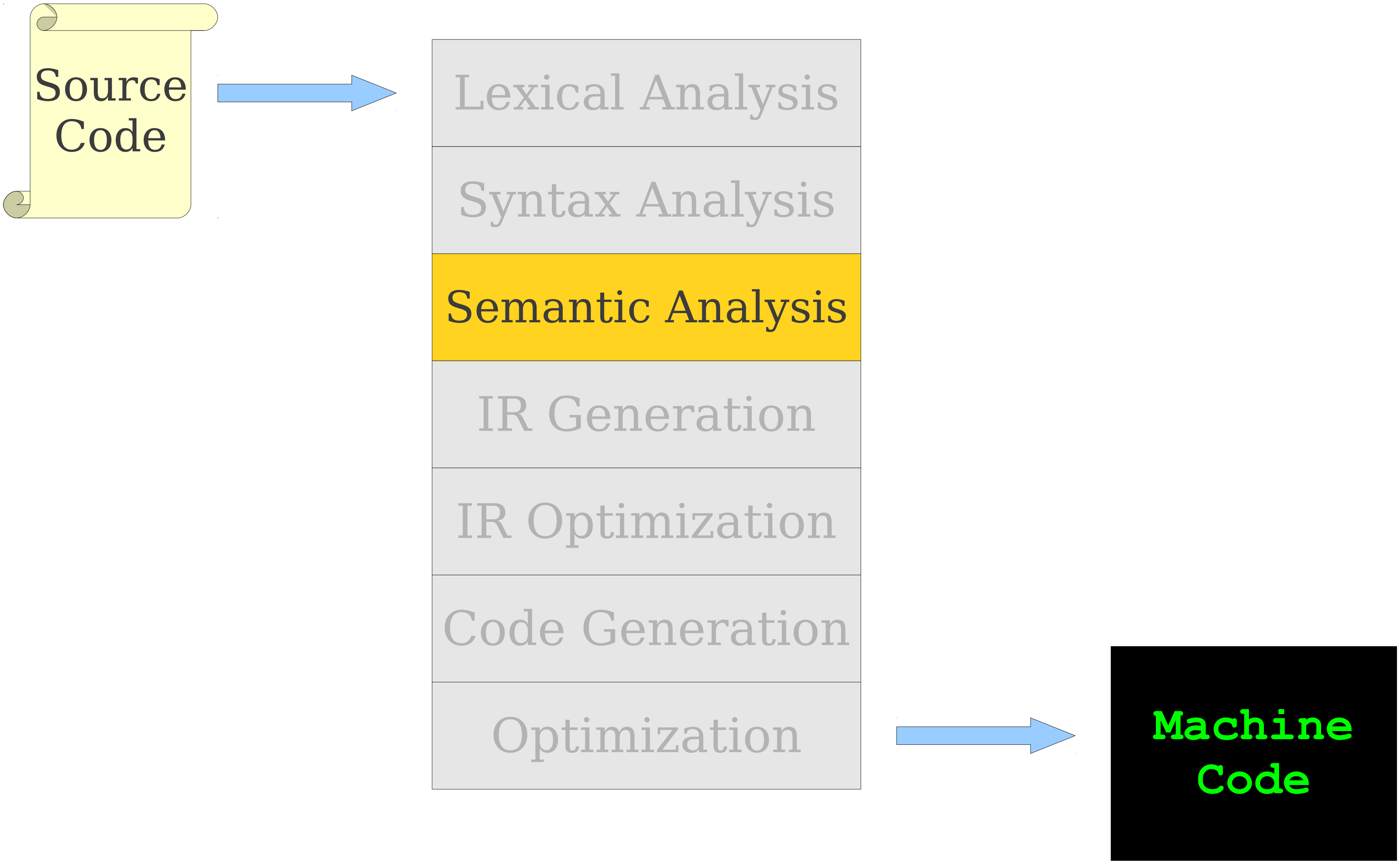
Type-Checking

Where We Are



Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

Review from Last Time

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x;
        x = new string;

        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

Review from Last Time

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x;
        x = new string;
        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

Interface not declared

Wrong type

Can't multiply strings

Variable not declared

Can't redefine functions

Can't add void

No main function

Review from Last Time

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x;
        x = new string;
        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

Wrong type

Can't multiply strings

Variable not declared

Can't redefine functions

Can't add void

No main function

Review from Last Time

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x;
        x = new string;
        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

Wrong type

Can't multiply strings

Variable not declared

Can't add void

No main function

Review from Last Time

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x;
        x = new string;
        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

Wrong type

Can't multiply strings

Can't add void

No main function

Review from Last Time

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x;
        x = new string;
        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

Wrong type

Can't multiply strings

Can't add void

What Remains to Check?

- **Type errors**.
- Today:
  - What are types?
  - What is type-checking?
  - A type system for Decaf.

What is a Type?

- This is the subject of some debate.
- To quote Alex Aiken:
  - **“The notion varies from language to language**.
  - The consensus:
    - A set of values.
    - A set of operations on those values”
- **Type errors** arise when operations are performed on values that do not support that operation.

Types of Type-Checking

- **Static type checking.**
  - Analyze the program during compile-time to prove the absence of type errors.
  - Never let bad things happen at runtime.
- **Dynamic type checking.**
  - Check operations at runtime before performing them.
  - More precise than static type checking, but usually less efficient.
  - (Why?)
- **No type checking.**
  - Throw caution to the wind!

Type Systems

- The rules governing permissible operations on types forms a **type system**.
- **Strong type systems** are systems that never allow for a type error.
  - Java, Python, JavaScript, LISP, Haskell, etc.
- **Weak type systems** can allow type errors at runtime.
  - C, C++

Type Wars

- *Endless* debate about what the “right” system is.
- Dynamic type systems make it easier to prototype; static type systems have fewer bugs.
- Strongly-typed languages are more robust, weakly-typed systems are often faster.

Type Wars

- *Endless* debate about what the “right” system is.
- Dynamic type systems make it easier to prototype; static type systems have fewer bugs.
- Strongly-typed languages are more robust, weakly-typed systems are often faster.
- **I'm staying out of this!**

Our Focus

- Decaf is typed **statically** and **weakly**:
  - Type-checking occurs at compile-time.
  - Runtime errors like dereferencing **`null`** or an invalid object are allowed.
- Decaf uses **class-based inheritance**.
- Decaf distinguishes primitive types and classes.

Typing in Decaf

Static Typing in Decaf

- Static type checking in Decaf consists of two separate processes:
  - Inferring the type of each expression from the types of its components.
  - Confirming that the types of expressions in certain contexts matches what is expected.
- Logically two steps, but you will probably combine into one pass.

An Example

```
while (numBitsSet(x + 5) <= 10) {

    if (1.0 + 4.0) {
        /* … */
    }

    while (5 == null) {
        /* … */
    }

}
```

An Example

```
while (numBitsSet(x + 5) <= 10) {

    if (1.0 + 4.0) {
        /* … */
    }

    while (5 == null) {
        /* … */
    }

}
```

An Example

```
while (numBitsSet(x + 5) <= 10) {

    if (1.0 + 4.0) {
        /* … */
    }

    while (5 == null) {
        /* … */
    }

}
```

An Example

```
while (numBitsSet(x + 5) <= 10) {

    if (1.0 + 4.0) {
        /* … */
    }

    while (5 == null) {
        /* … */
    }

}
```

Well-typed expression with wrong type.

An Example

```
while (numBitsSet(x + 5) <= 10) {

    if (1.0 + 4.0) {
        /* … */
    }

    while (5 == null) {
        /* … */
    }

}
```

An Example

```
while (numBitsSet(x + 5) <= 10) {

    if (1.0 + 4.0) {
        /* … */
    }

    while (5 == null) {
        /* … */
    }

}
```

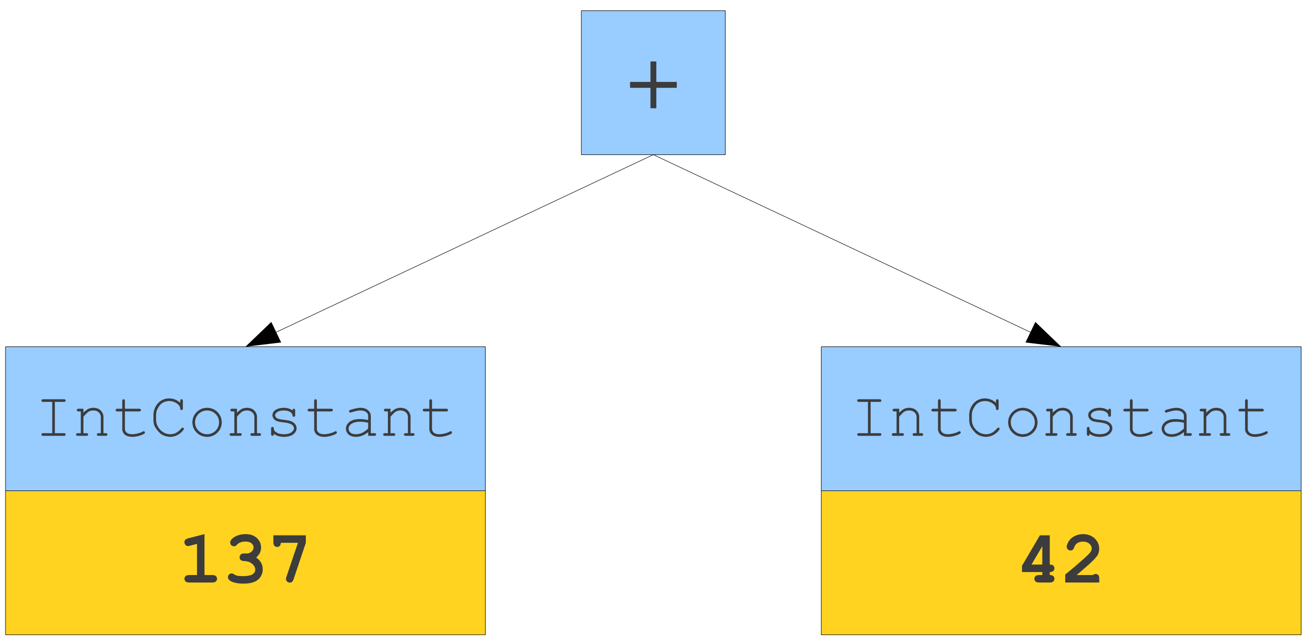
Expression with type error

Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.

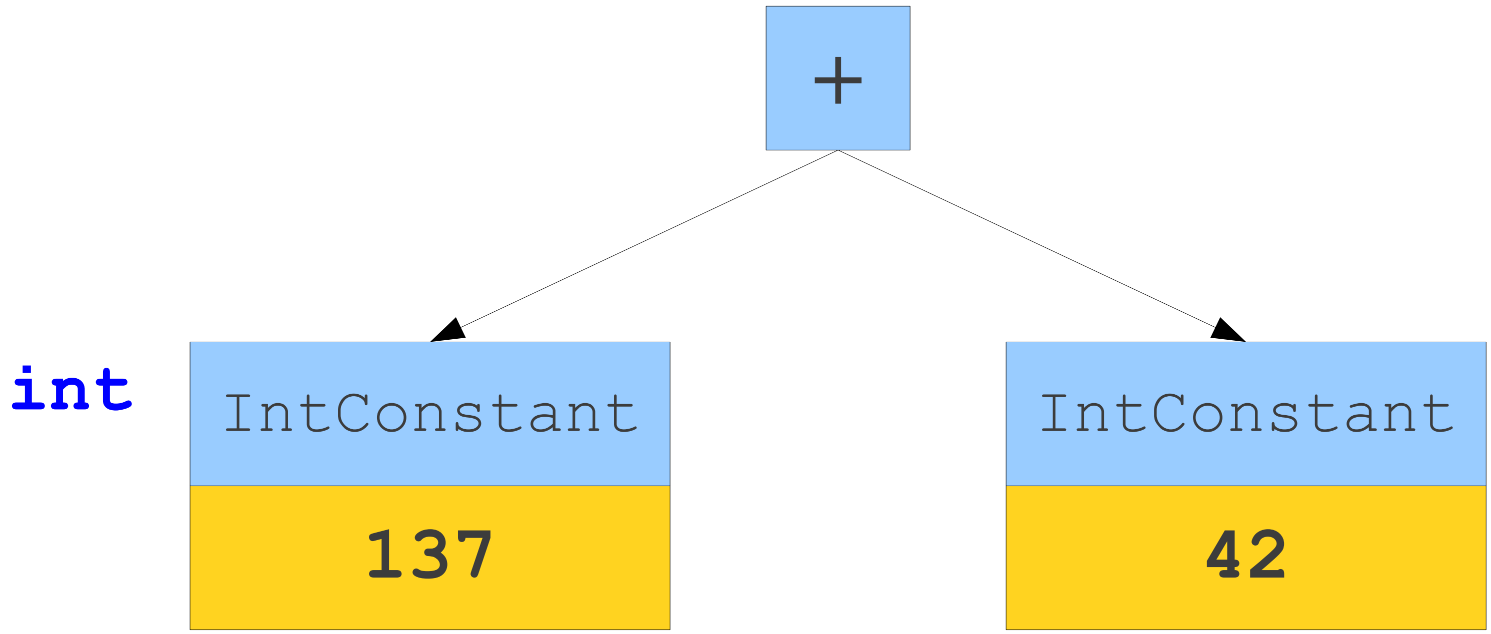
Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.



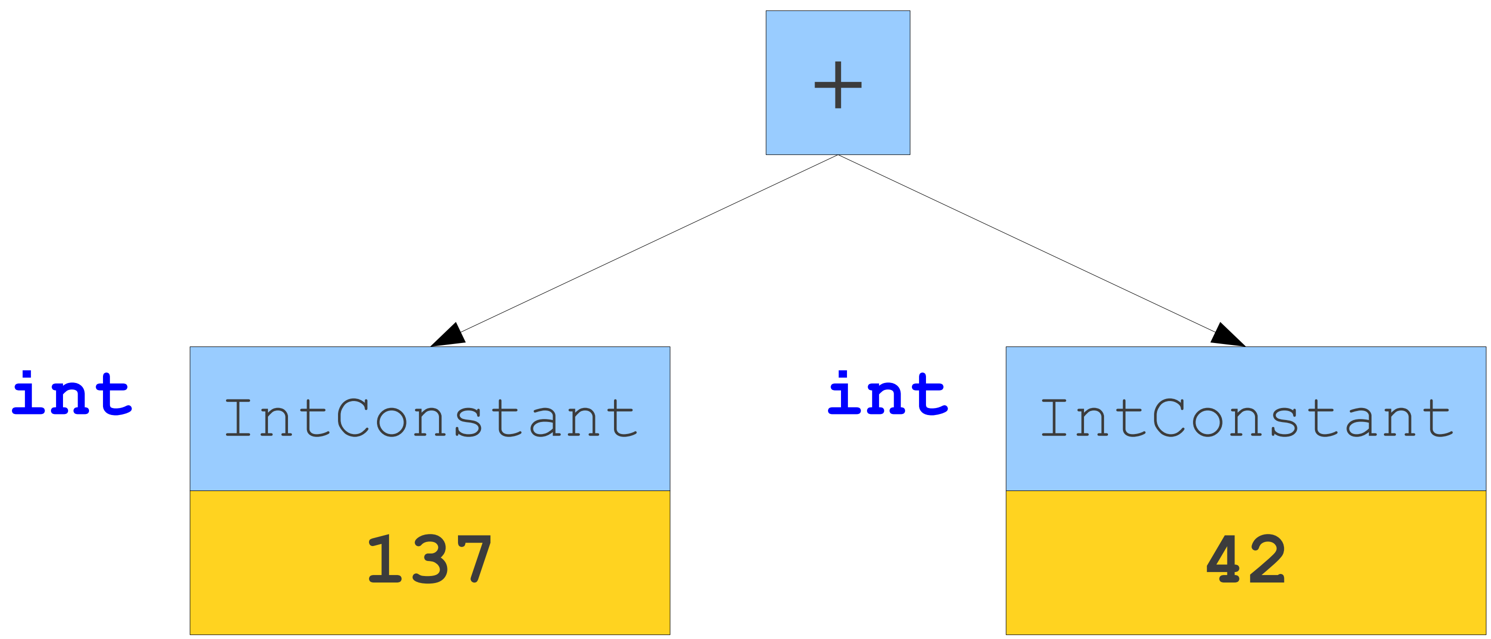
Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.



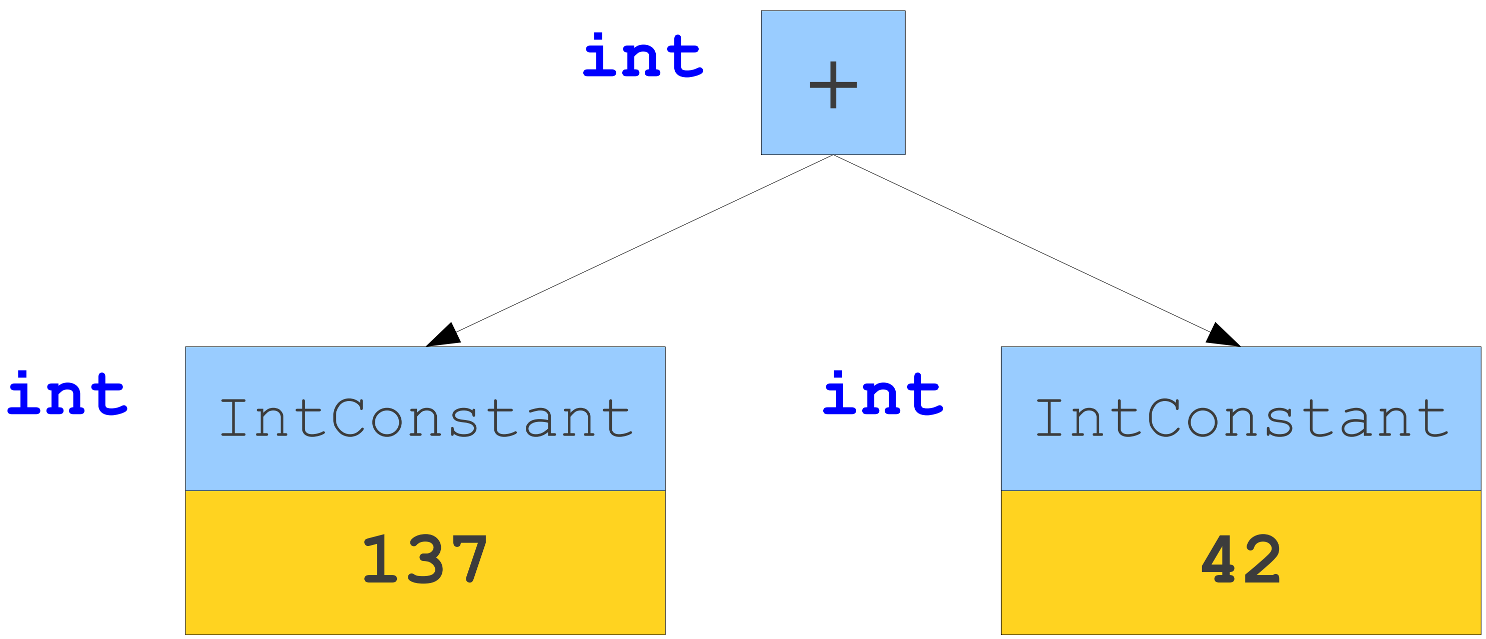
Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.



Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.

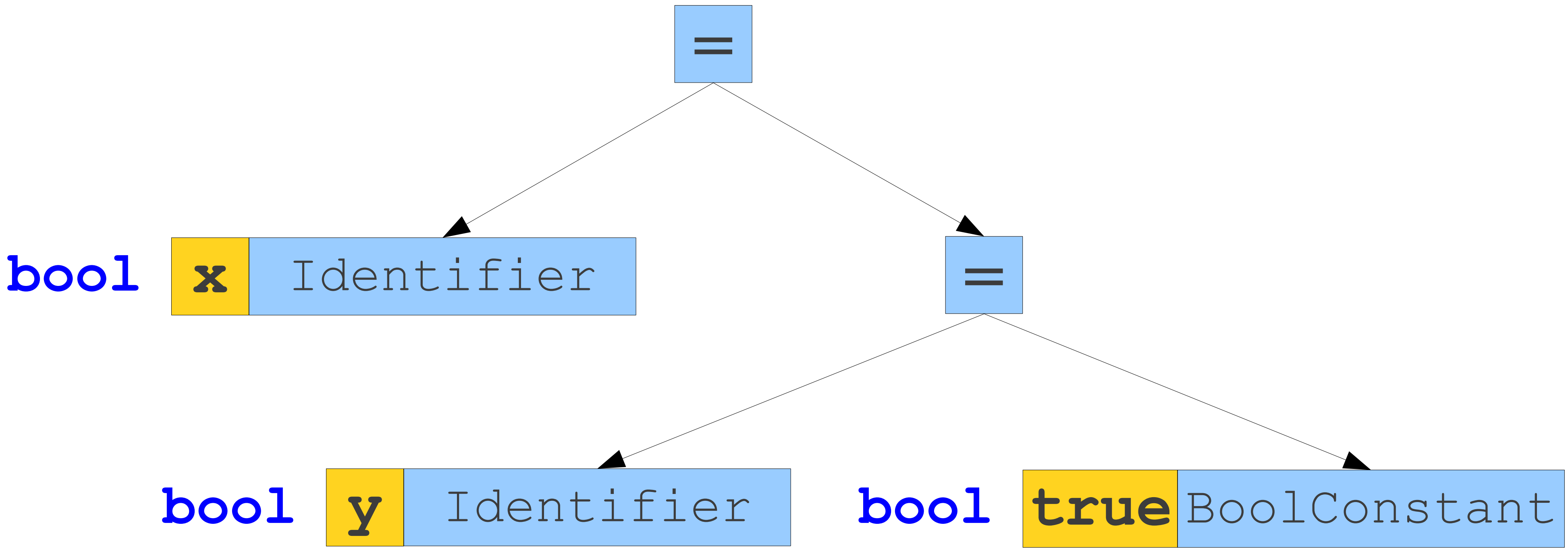


Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.

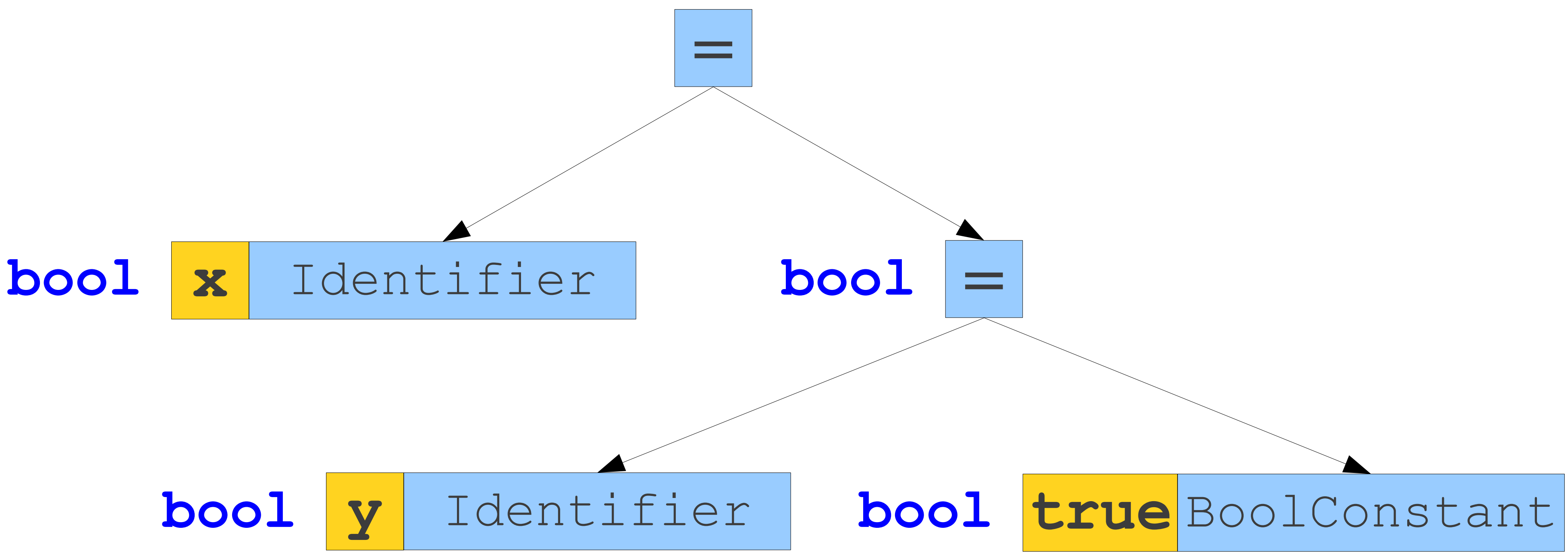
Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.



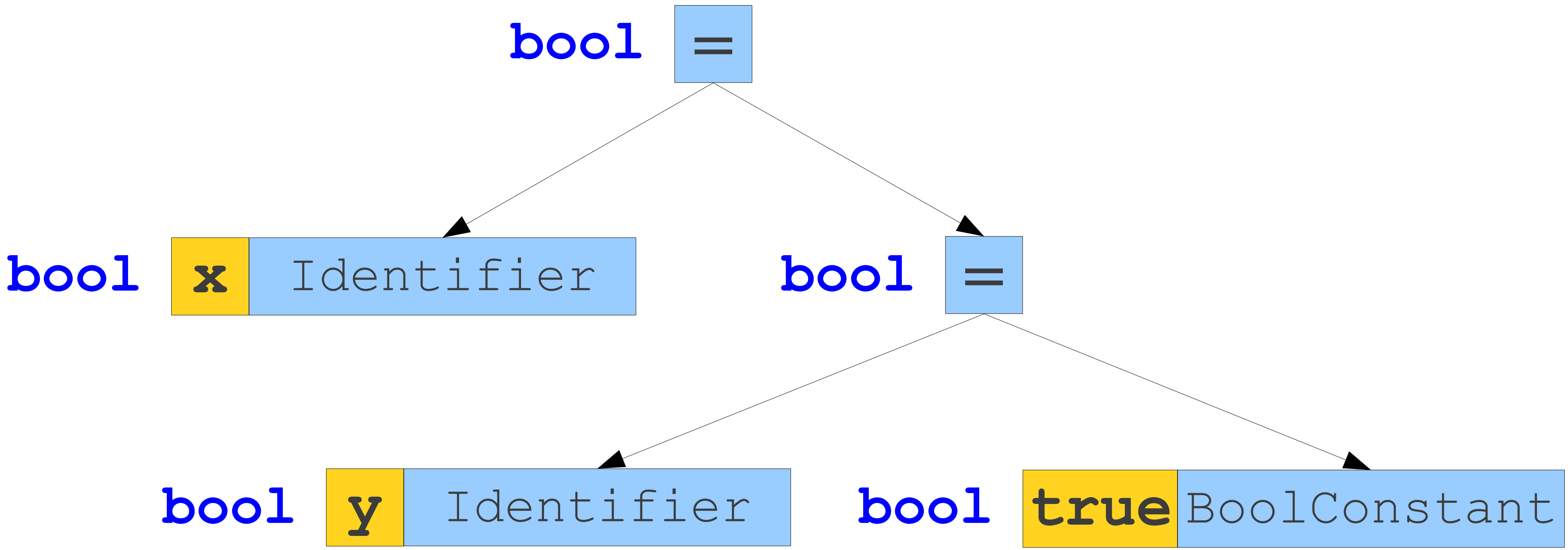
Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.



Inferring Expression Types

- How do we determine the type of an expression?
- Think of process as **logical inference**.



Type Checking as Proofs

- We can think of syntax analysis as proving claims about the types of expressions.
- We begin with a set of **axioms**, then apply our **inference rules** to determine the types of expressions.
- Many type systems can be thought of as proof systems.

Sample Inference Rules

- "If **x** is an identifier that refers to an object of type **t**, the expression **x** has type **t**."
- "If **e** is an integer constant, **e** has type **`int`**."
- "If the operands $\mathbf{e}_1$ and $\mathbf{e}_2$ of $\mathbf{e}_1 + \mathbf{e}_2$ are known to have types **`int`** and **`int`**, then $\mathbf{e}_1 + \mathbf{e}_2$ has type **`int`**."