Static Type Checking

what is static type checking

- Static type checking is the process of ensuring that a program is **well-typed**

- Central idea: types constrain the behavior of a program
- A *well-typed* program is one whose run-time behavior stays within that set of constraints

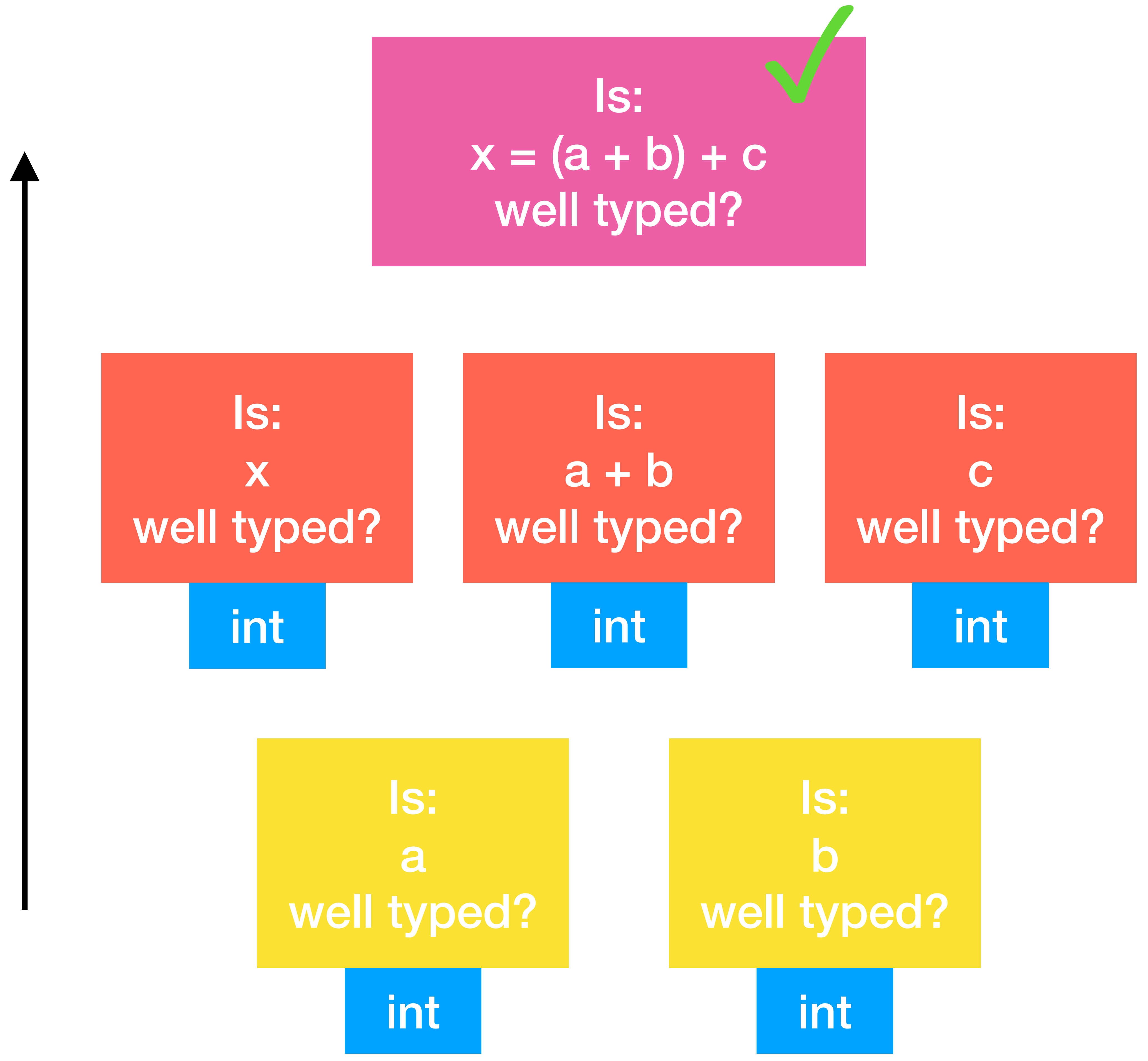
- Example: an expression that is well-typed and has type **int** will, at runtime, produce a value that is an **int**

syntactic type checking

- Static type checking is based on the **syntax** of the expressions being typed as well as the **context** in which the type checking happens
- Is x = y + z well-typed?
- The statement forms a context: are x, y, and z all the same type, and types that can have arithmetic operations performed on them?

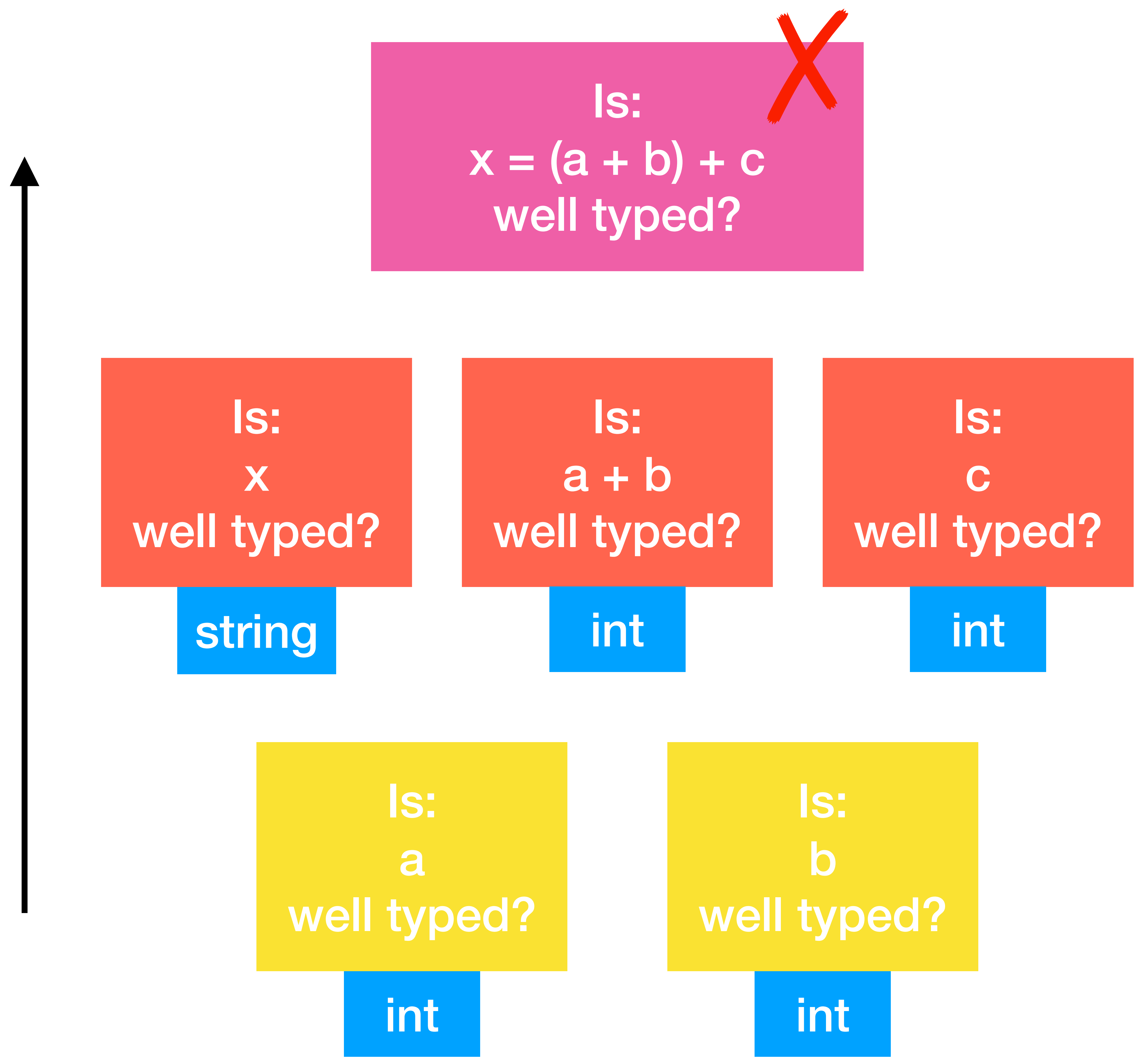
syntactic type checking

- Being **well-typed** is an inductive property
- Basic idea: assign a type to every expression
  - If you can assign a type to an expression, it is well-typed
- Type check expressions and statements by breaking them down into smaller components
  - Find the types of smaller expressions
  - Combine types of smaller expressions to assign a type to the larger expression



syntactic type checking

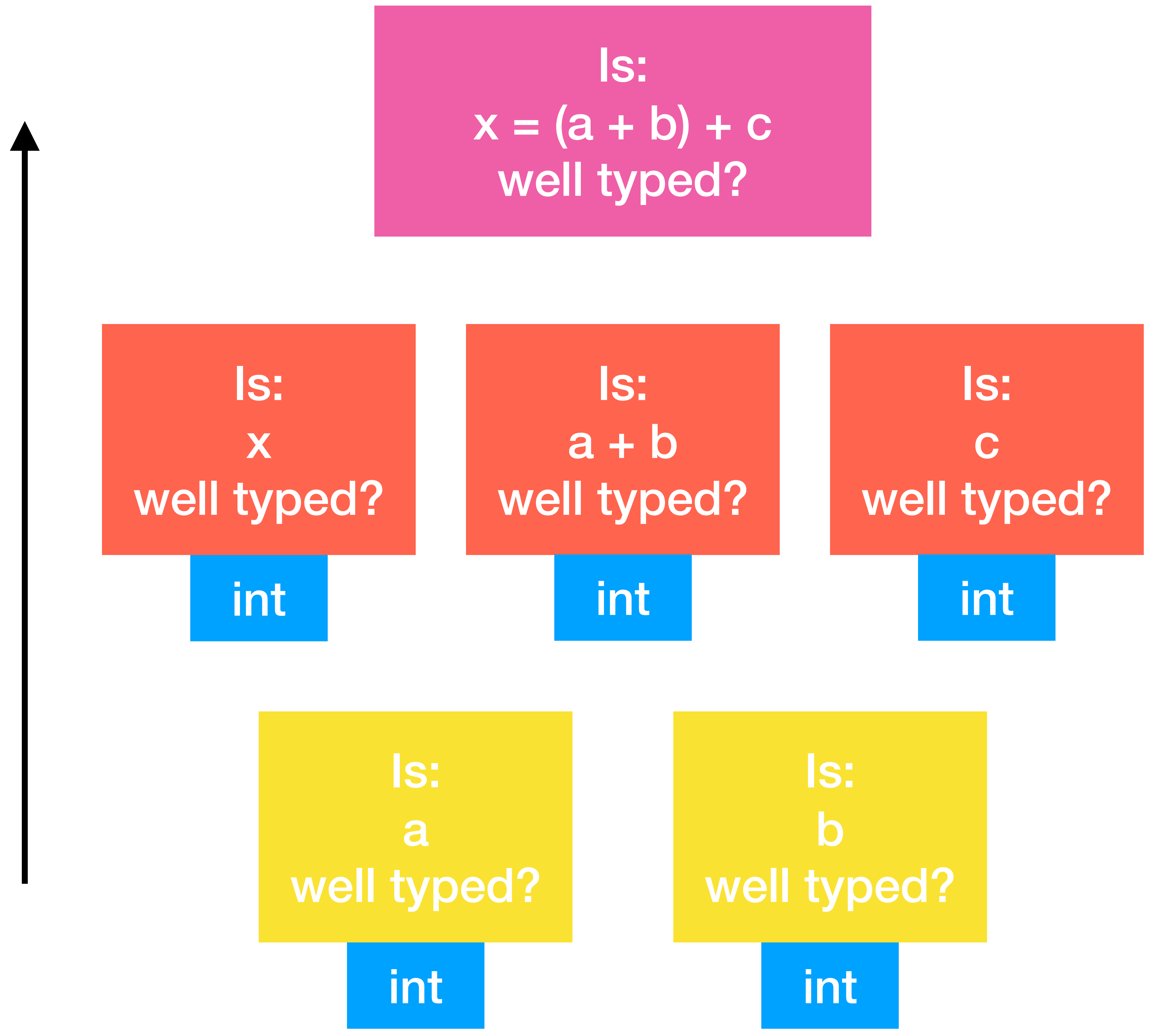
- Being **well-typed** is an inductive property
- Basic idea: assign a type to every expression
  - If you can assign a type to an expression, it is well-typed
- Type check expressions and statements by breaking them down into smaller components
  - Find the types of smaller expressions
  - Combine types of smaller expressions to assign a type to the larger expression



type rules

- For each syntactic form in your language
  - Expressions
  - Statements
- Describe the rules under which the form is well typed based on the types of its sub-components

- Structure captured by AST!


Is:
x = (a + b) + c
well typed?
Is:
x
well typed?
int
Is:
a + b
well typed?
int
Is:
c
well typed?
int
Is:
a
well typed?
int
Is:
b
well typed?
int