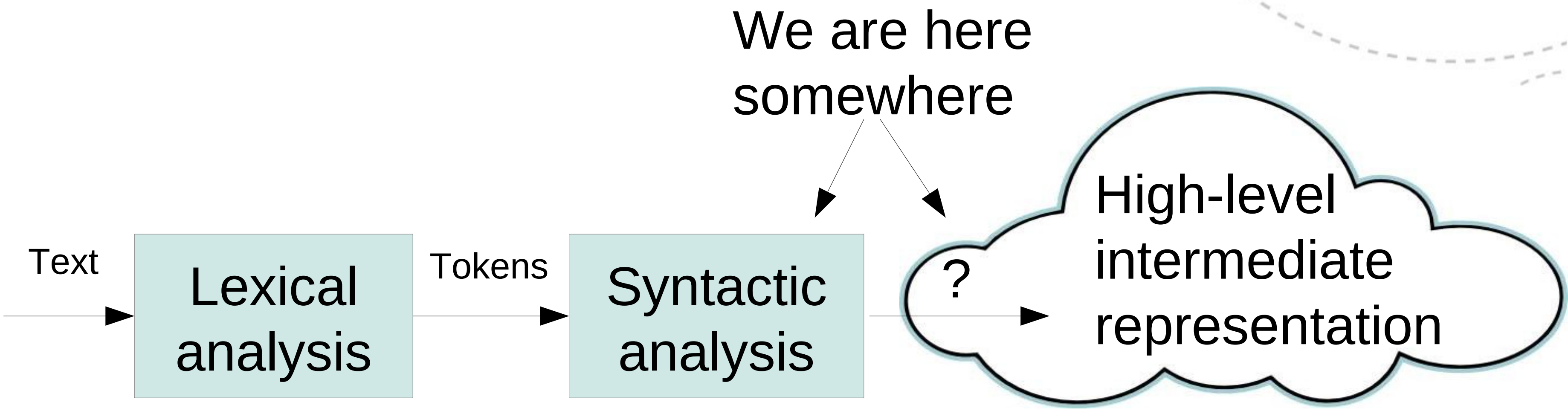
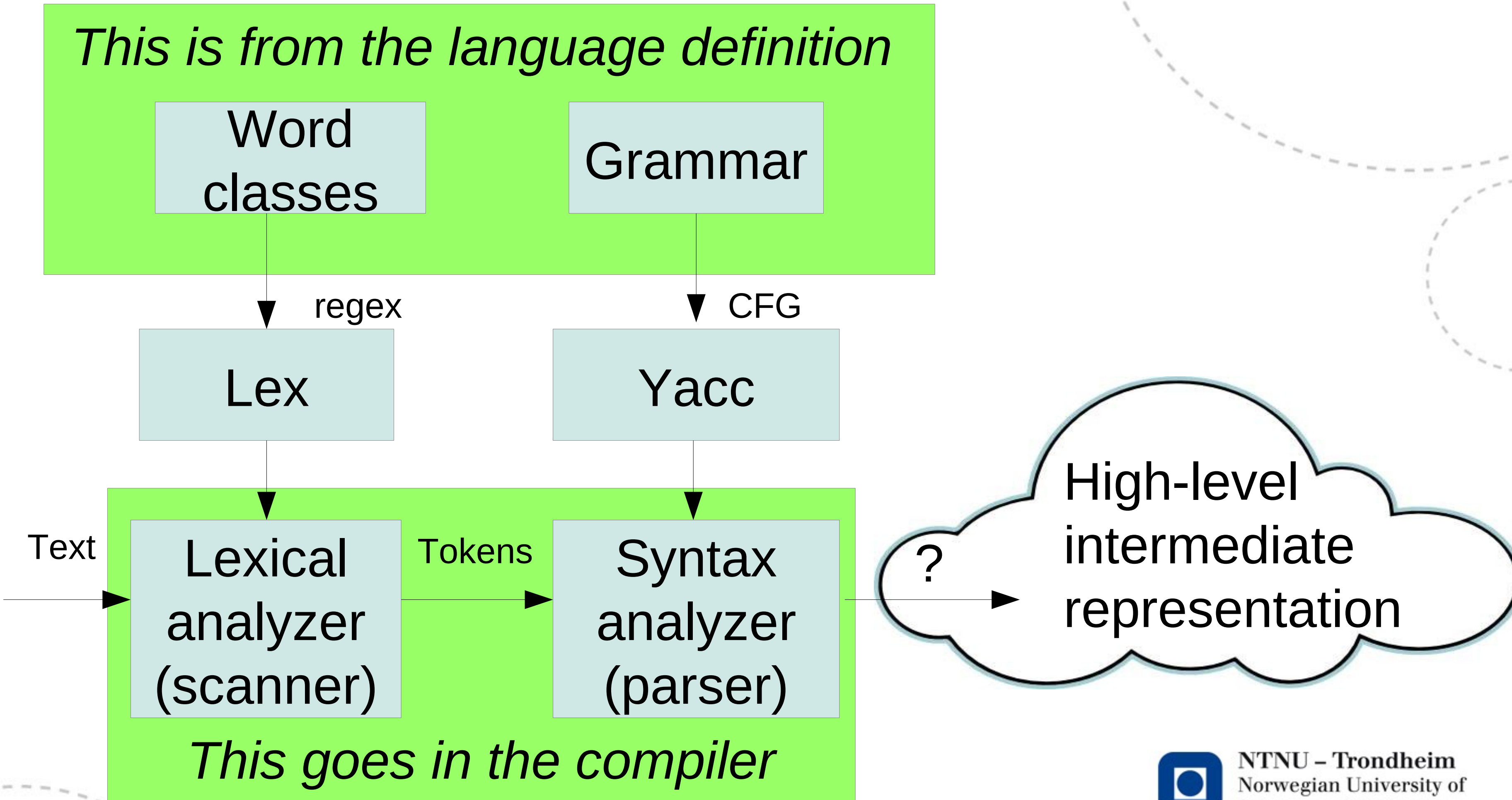
NTNU – Trondheim
Norwegian University of Science and Technology

Syntax analysis and syntax-directed translation

Return to the big picture



Implementation by generators

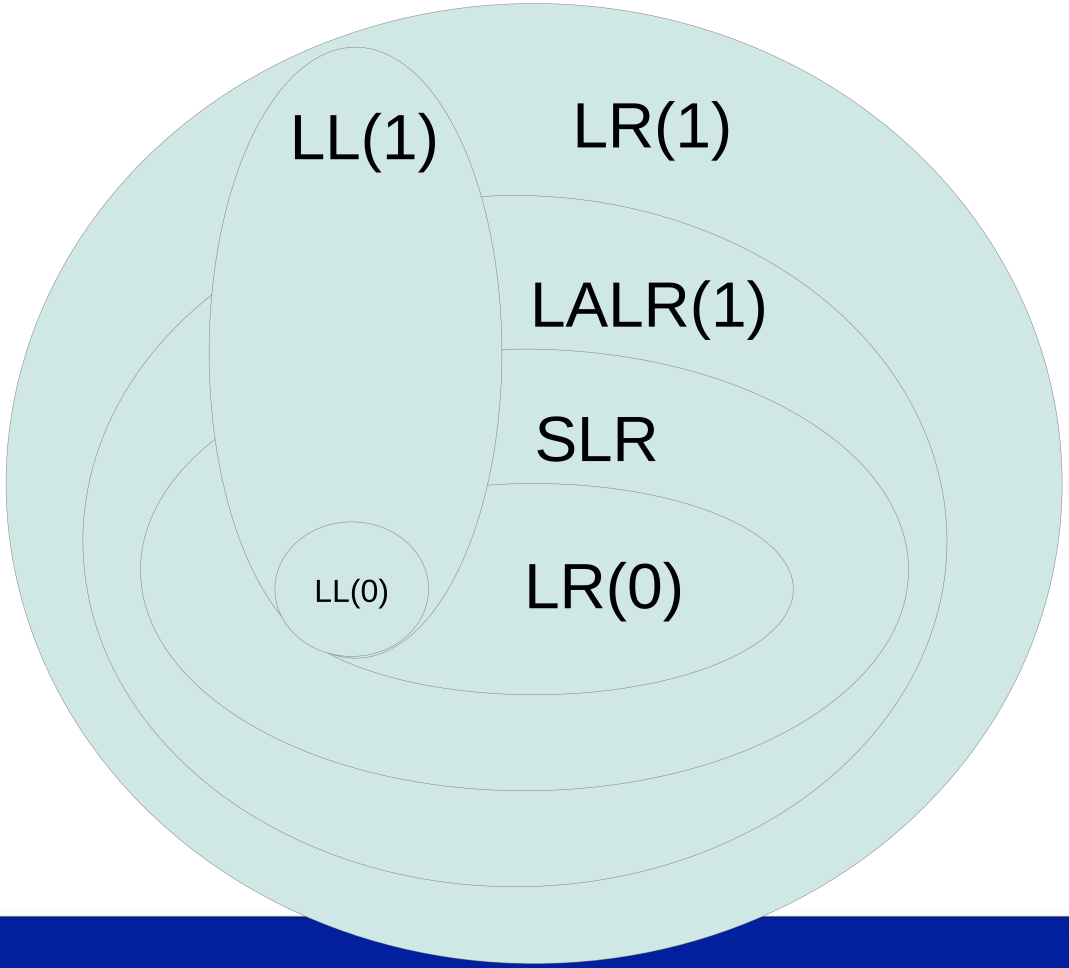


NTNU – Trondheim
Norwegian University of Science and Technology

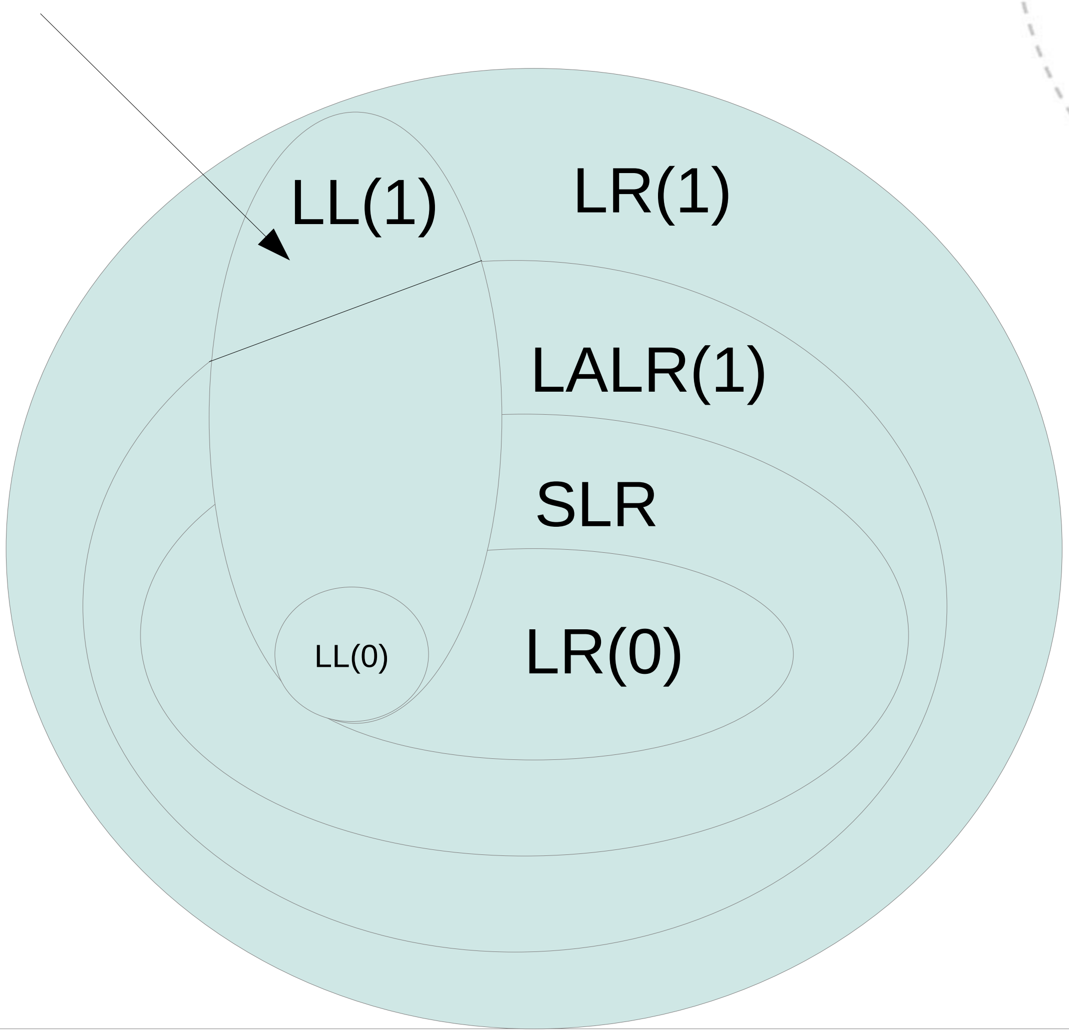
Footnotes on syntax analysis

- A few factoids didn't fit naturally anywhere
- We've looked at different classes of languages
  - Regular languages
  - Context-free languages
  - LL(1)-parseable languages
  - LR-parseable languages
    - ...in LR(0), SLR, LALR and LR(1) flavors…
- We've looked at *shift/reduce* conflicts
  - There are also reduce/reduce conflicts

Relationships of grammar classes



This part exists



- You **can** construct languages that are LL(1) but not LALR(1)
- It’s mostly an artificial exercise in order to prove a point

Reduce/reduce conflicts

- These arise when a state contains multiple reducing items for different productions
- Consider the grammar

  $A \rightarrow By \mid Cy$

  $B \rightarrow x \mid z$

  $C \rightarrow w \mid z$

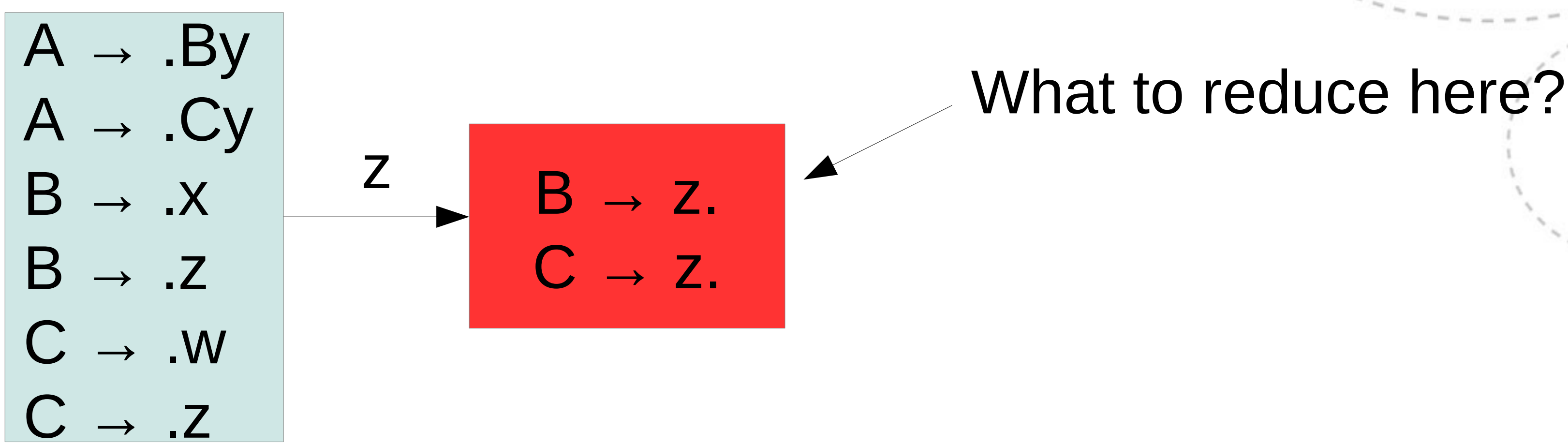
- It's ambiguous, you can derive

  $A \rightarrow By \rightarrow zy$

  $A \rightarrow Cy \rightarrow zy$

Reduce/reduce conflicts

- Writing out part of an automaton,



That was a trivial example

- It can happen if you're not careful, suppose we allow

  <type> <identifier> (<identifier-list>)

  for function declarations,

  <type> <identifier> (<size>)

  for array declarations, and

  <identifier-list> → <identifier> (1-element argument lists)

  <size> → <identifier> (Variable size arrays)

  then

  int my_thing (some_number)

  can declare either an array or a function

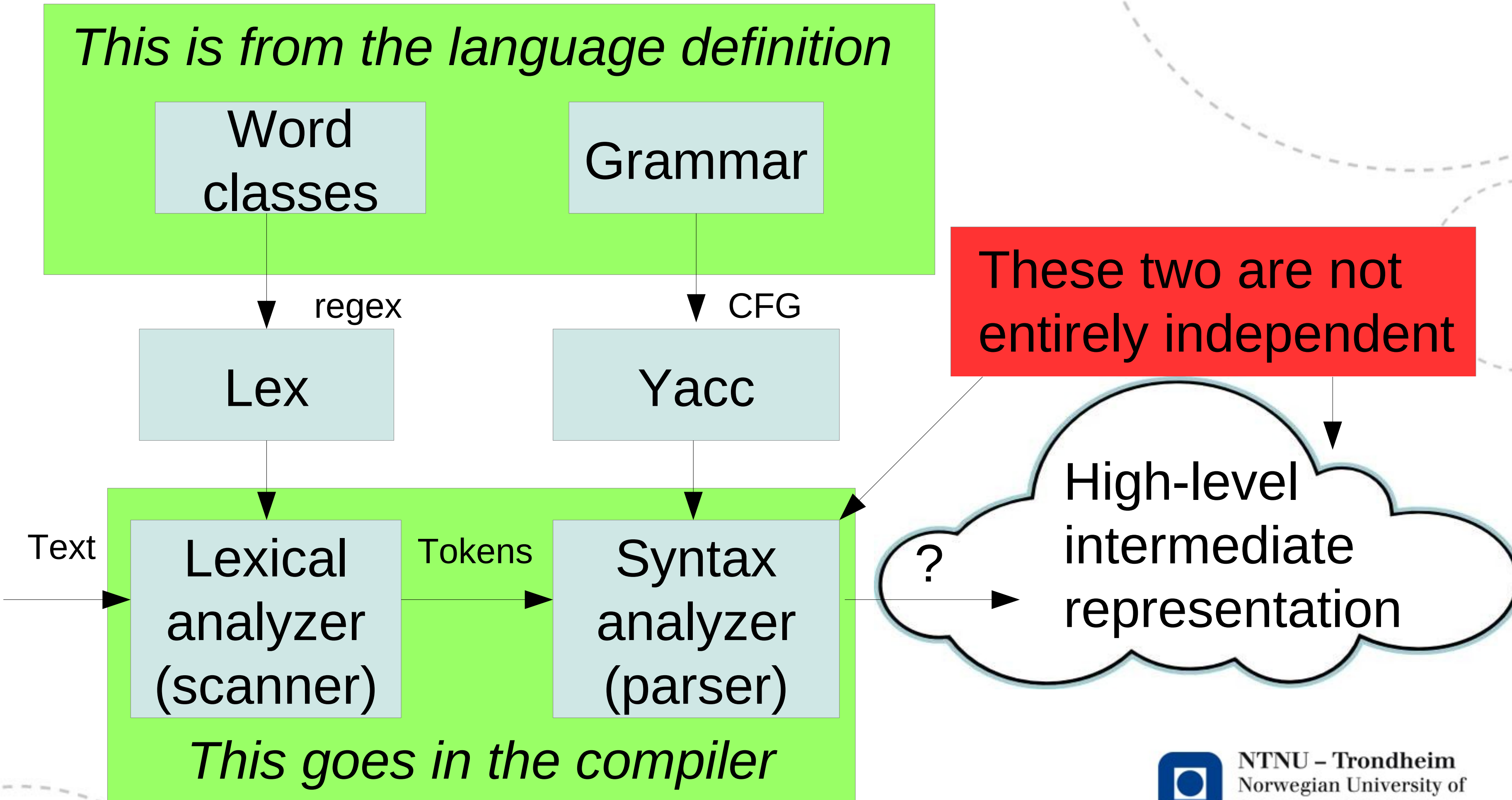
Living with conflicts

- As far as ambiguity goes, shift/reduce conflicts are relatively benevolent, they can be resolved by imposing a rule
  - Colloquially, how far should we look to select an interpretation
- Reduce/reduce conflicts are a symptom that the grammar is broken
  - Colloquially, one block of text has two meanings
  - Fixing these by enforcing a precedence creates languages with confusing rules, because they're entirely implicit in the source text
  - In my opinion, it is better to repair the grammar in this case

On our way to high IR

- With a keen eye, you may have noticed that we've been slipping some ideas sideways into the syntax
  - Precedence rules for ambiguous operators
  - Matching rules for ambiguous nesting of if-statements
- These aren't technically syntax analysis, since they imply that statements say meaningful things
  - We're imposing *semantics*

I have been telling a small lie



NTNU – Trondheim
Norwegian University of Science and Technology

The connection

- When a parser applies a grammar rule (by predicting or reducing), we have an opportunity to affect the overall program state however we want
  - Yacc allows productions to carry blocks of C that are run when the rule reduces
  - The symbols of the prod. body are on stack, positions $1, $2, $3, …
  - The newly created, reduced symbol is available as $$
  - We can take the opportunity to attach a data structure to those, to capture all the information that isn't evident in the grammar

Syntax-directed translation

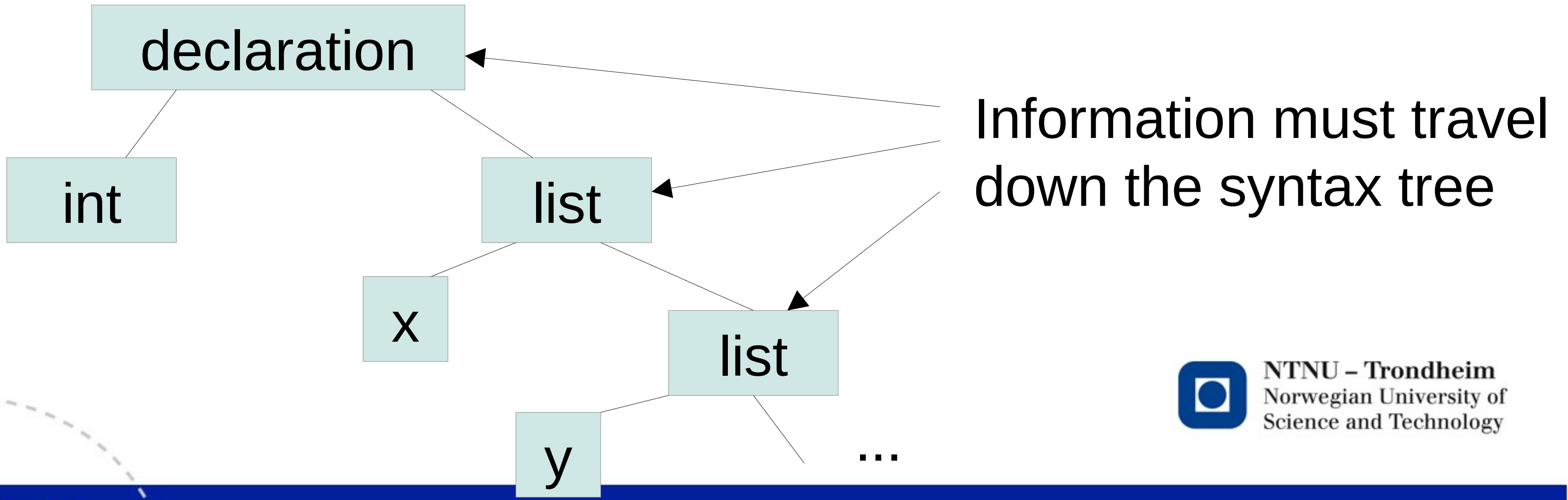
- The *syntax-directed* bit is that this information is derived by connecting it with the relevant production
  - Thus, we can be sure to cover the translation of every possible type of statement
- The *translation* part is to go from text into another structure that equivalently captures the meaning of the program

In olden times

- Compilers are complicated programs, their speed and size used to be of great importance
  - It still is when programs grow large, but we don't need to worry quite as much in the age of fast processors and vast memory
- If you take care with how you define your language, everything interesting about the source program can be detected during parsing
  - You can write the entire compiler into the semantic actions, and make one that directly blurts out machine code as soon as it sees a construct
  - One big pass of reading and writing, very efficient

That requires a certain ordering

- Symbols need to be annotated with stuff that is detected along the way:
  - When you see an identifier in syntax, what is its name? (attach the lexeme)
  - When you see a list of declarations, how to remember their type?



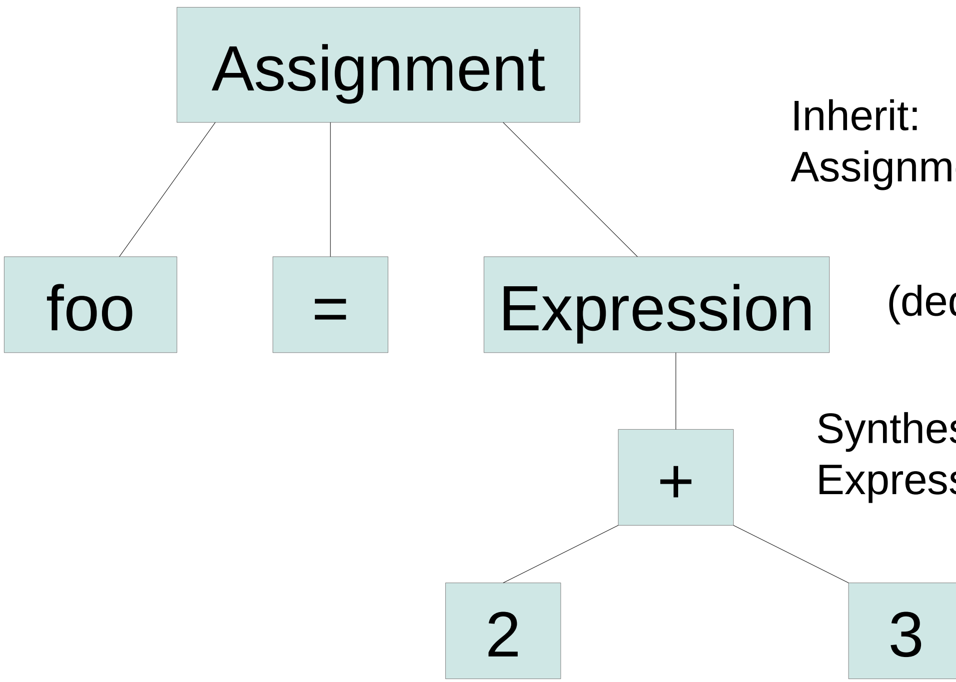
Attributes

- The internal representation of a symbol can be any ol’struct, object, what-have-you
- Rather than just a token value, it can have elements that capture the additional information
  - Number symbols naturally invite a property Number.value
  - Identifiers might have Identifier.name and Identifier.type
  - Functions can be well served with a Function.argument_count

    etc. etc.

Inherited and synthesized

- In a syntax tree representation, inherited attributes come from above, synthesized attributes come from below



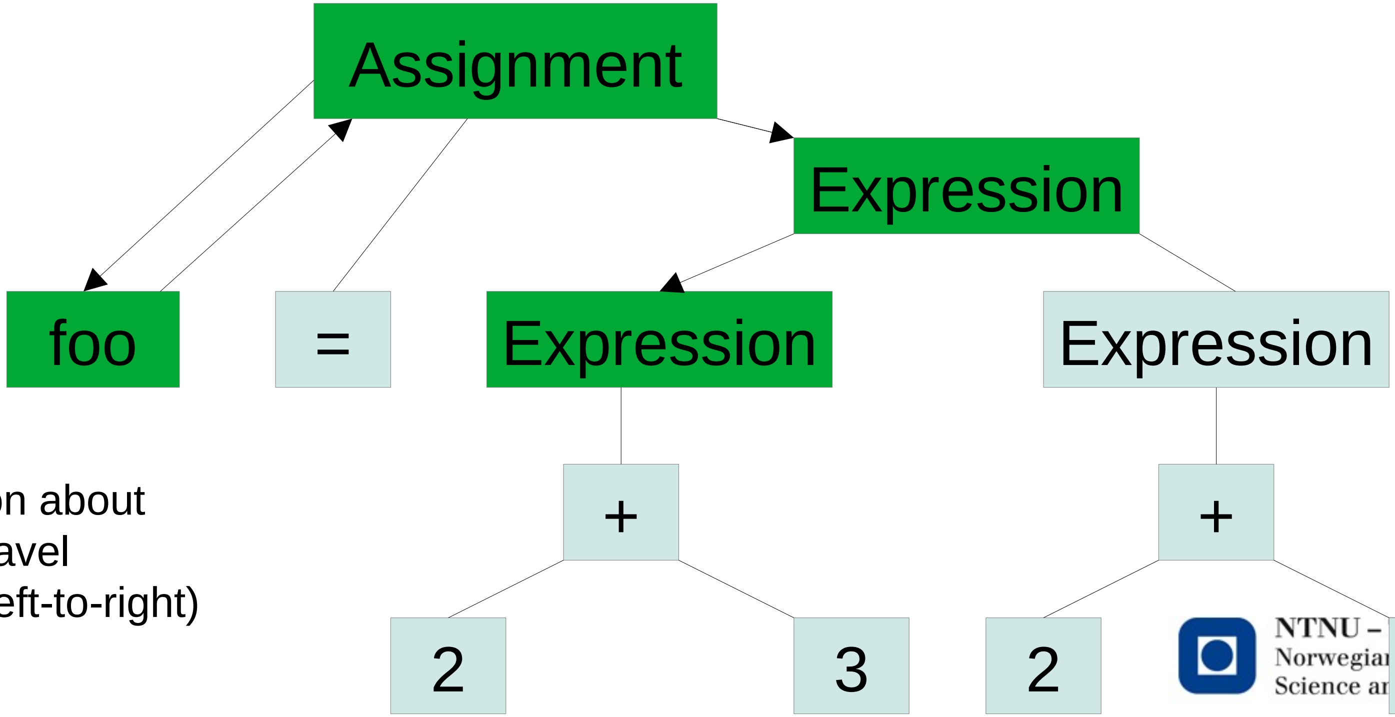
Inherit:
Assignment.target_type = float

(deduce: this 5 needs to be 5.0)

Synthesize:
Expression.value = 5

L-attribution

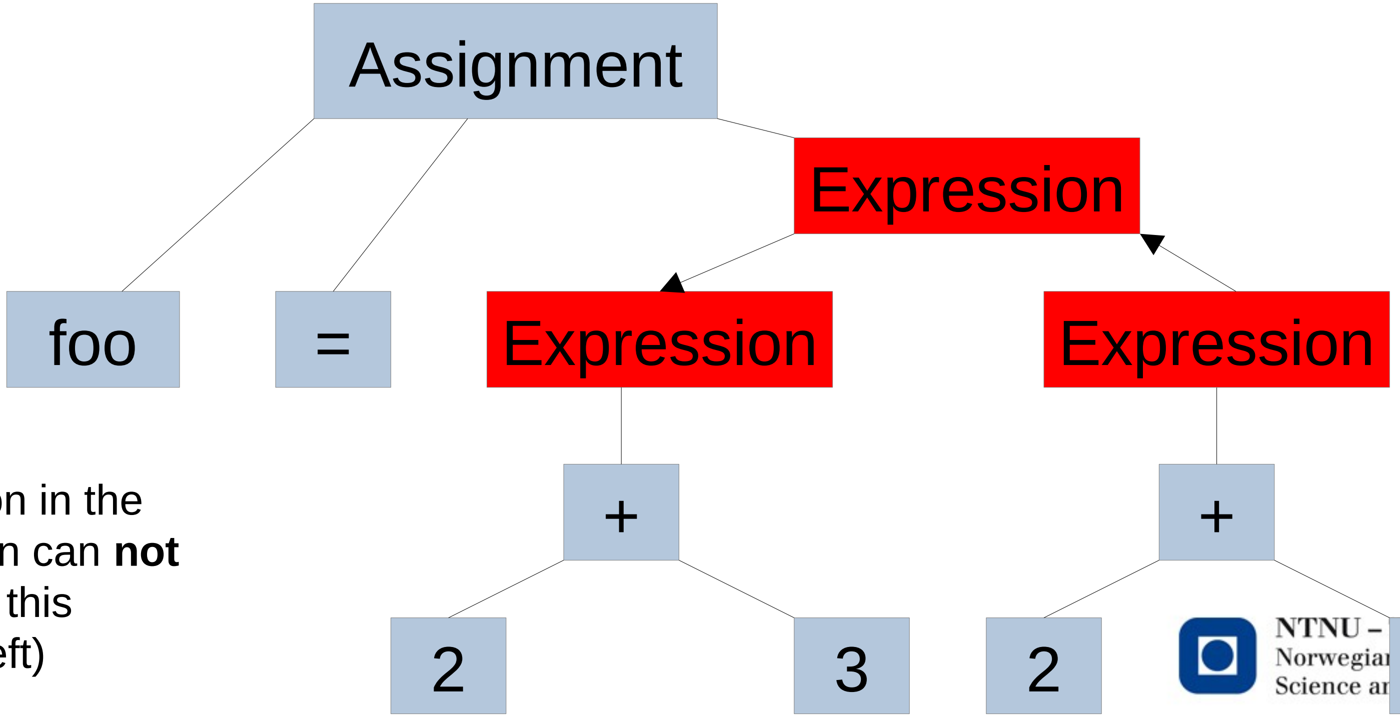
- L-attributed grammars allow synthesized attributes, and inheritance from the left



Information about foo can travel like this (left-to-right)

L-attribution

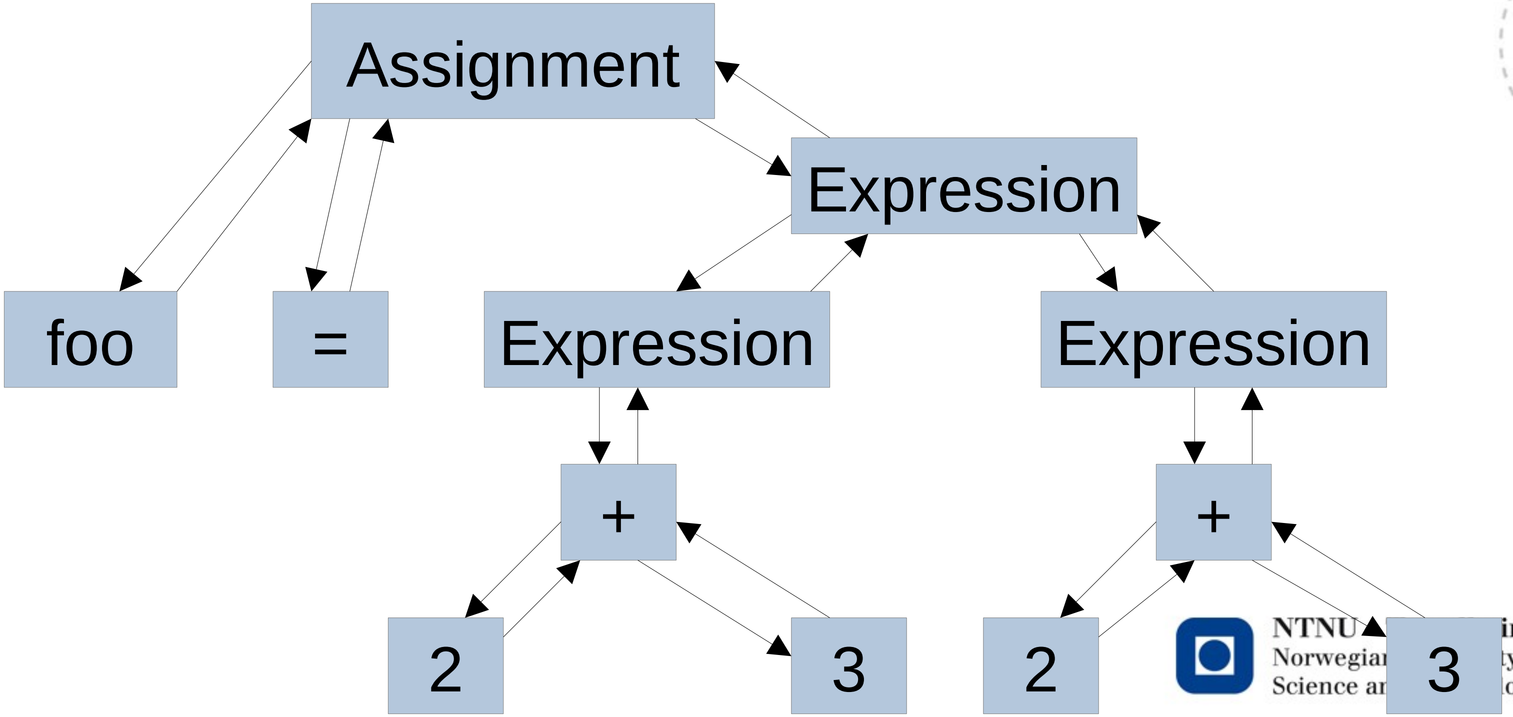
- L-attributed grammars allow synthesized attributes, and inheritance from the left



Information in the expression can **not** travel like this (right-to-left)

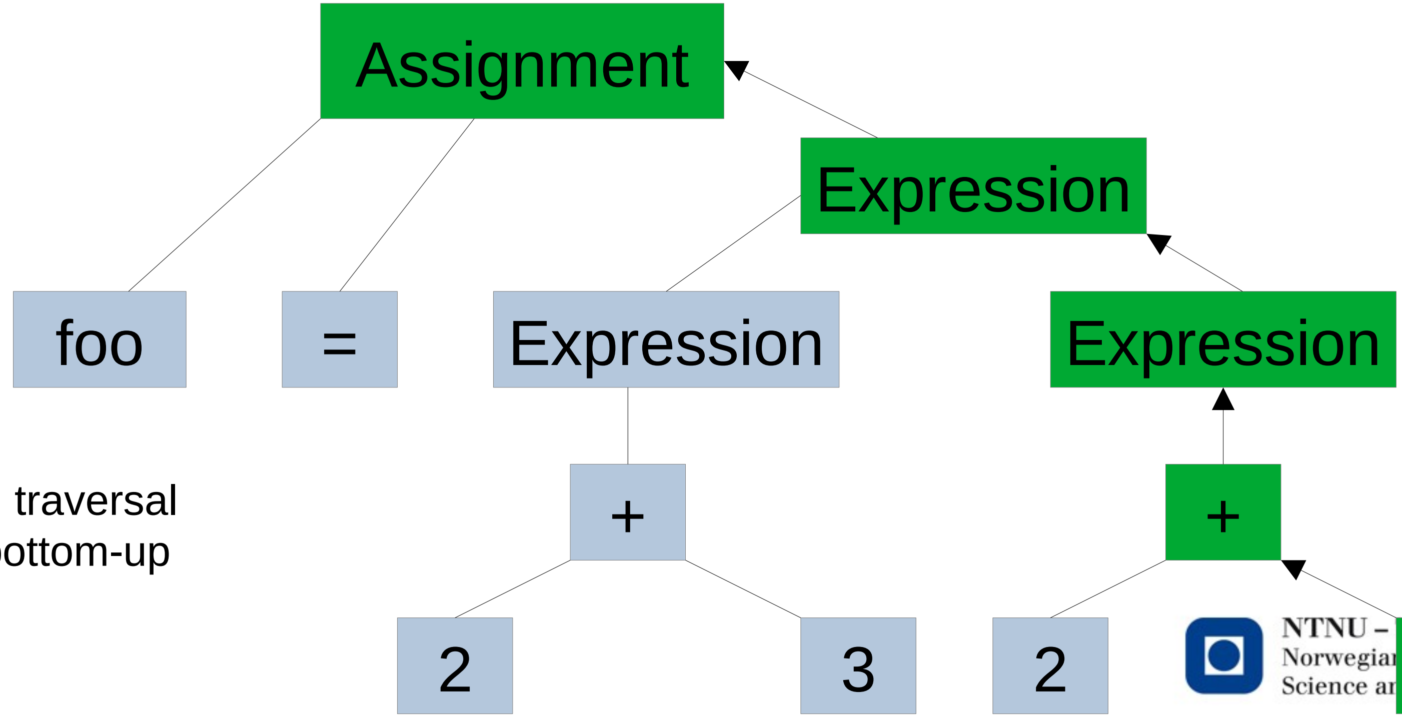
L-attribution

- This makes sense if you look at the traversal order of a predictive parser
- It goes from top-to-bottom and back, but left-to-right at any given level in the tree



S-attribution

- All attributes are synthesized, information comes from below



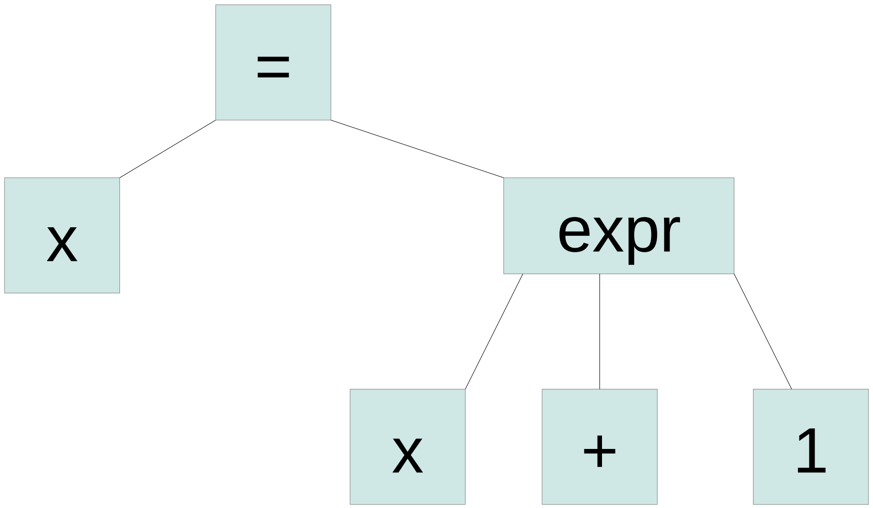
Confer w. traversal order of bottom-up parser

One convenient use

- I have been drawing syntax trees to illustrate the traversal orders of parsing all the while we were talking about them
- The act of parsing does not in and of itself construct a syntax tree, it just traces the traversal order
- When it's not so important to do everything at once
  - SDD actions offer a fine opportunity to build the syntax tree, by hooking tree nodes that represent the symbols together
  - That way, we can detect everything needed by going through the tree structure forwards, backwards, and sideways after parsing is finished

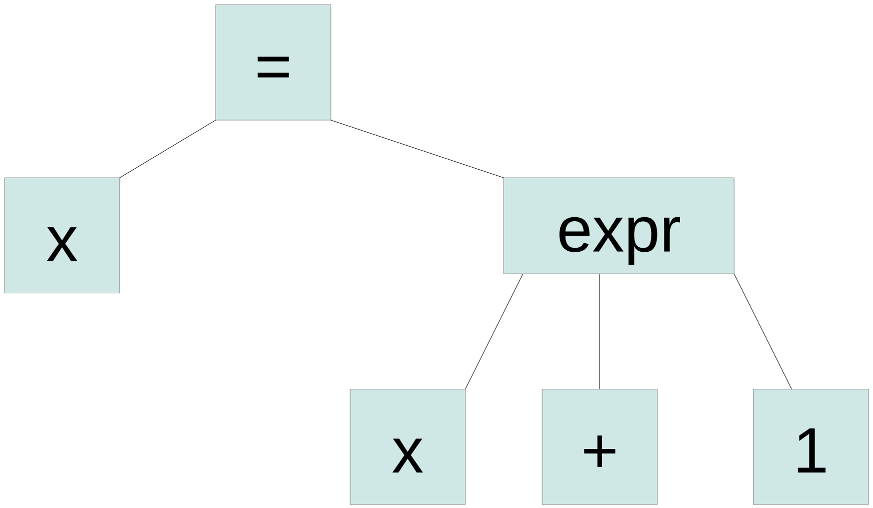
Connecting symbols

- Here's another syntax tree, for a familiar type of statement



Translation in pseudo-code

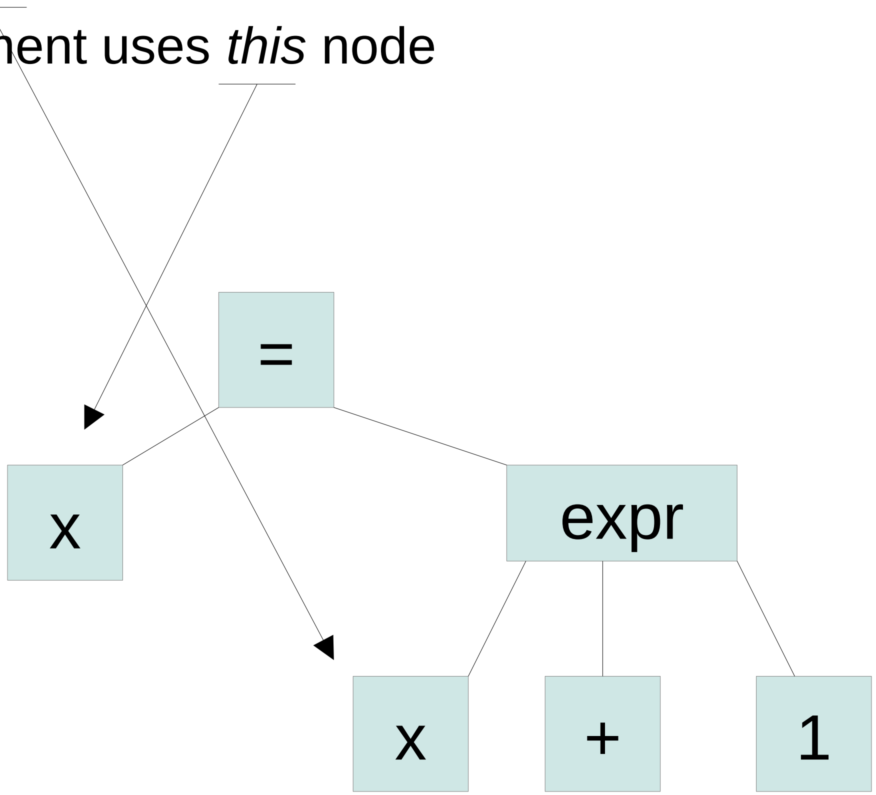
- What has to happen here is
  - Take a number out of a memory location
  - Add 1 to it
  - Put it back in the same memory location it came from



NTNU – Trondheim
Norwegian University of Science and Technology

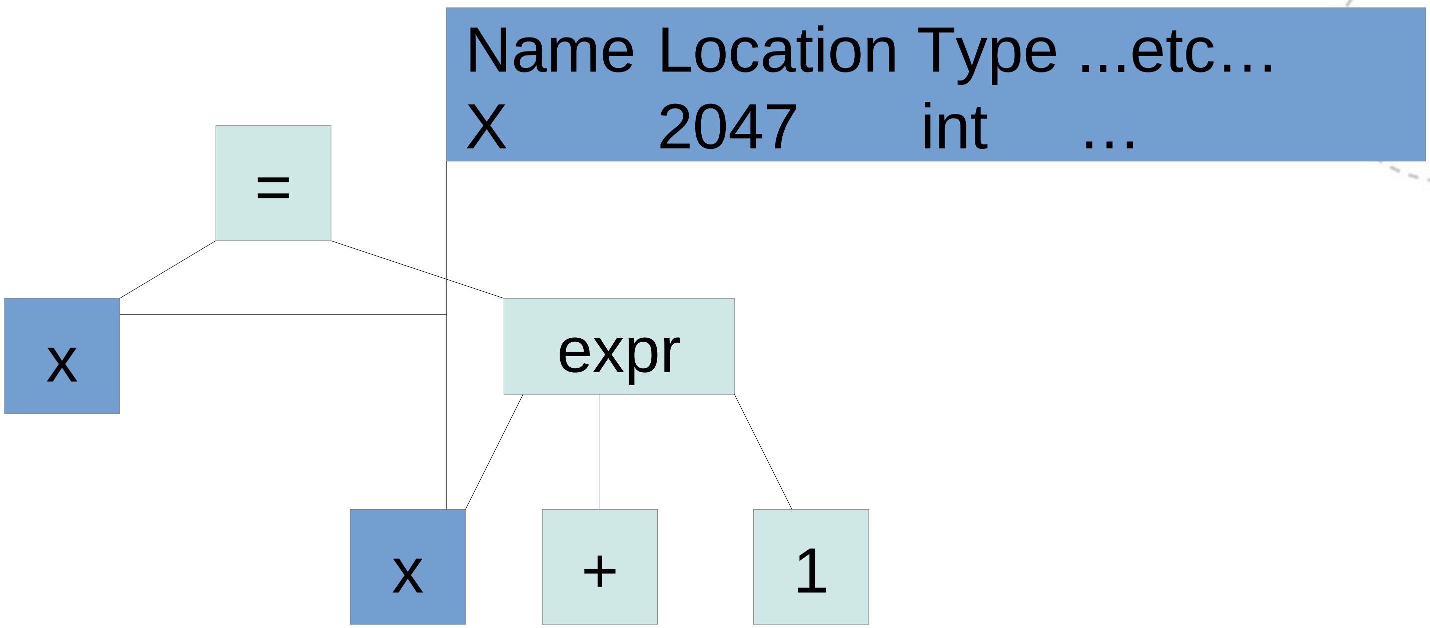
X marks the spot

- When we're translating the +, we have to get the memory location based on *this* node
- The assignment uses *this* node



Symbol tables

- It's convenient to keep a table where all the information about names go, and connect the nodes to it



Implementation of symbol tables

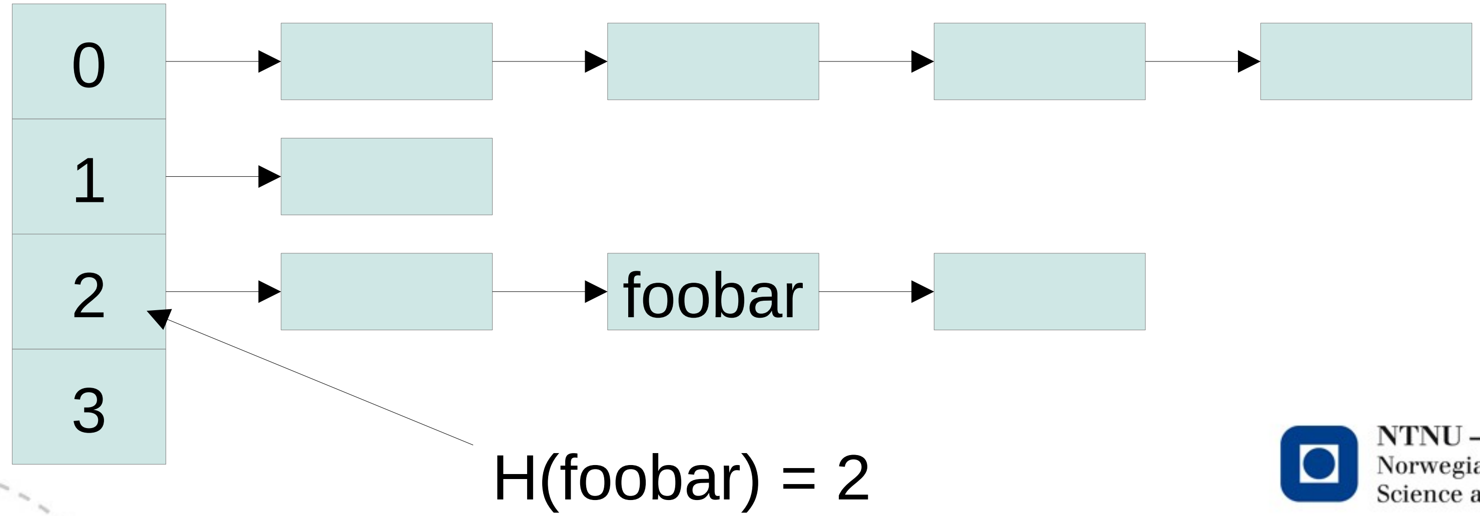
- Making this happen requires us to find the table entry for “x” every time that name appears
  - The name has to be enough to look it up, so we have a text search problem
- Three ways readily suggest themselves:
  - Direct indexing (Keep a table where index is a function of the text)
  - Linked list (Keep a dynamic list, go through it and compare)
  - Hash table

Direct index and linked list

- Compilers look up names all the time, programs are positively packed full of names
- Neither of these alternatives are great
  - Direct indexing is very fast, but limits the number of identifiers to the size of the symbol table
  - Linked list is perfectly flexible, but requires that we search through variables #1,#2,#3,#4... every time we look up variable #270

Hash tables

- An unpredictable, fixed-length code can be computed from any length of identifier
- Fixed-length array of linked lists, search and compare



Hash tables are a good compromise

- Constant time to find the right list to search
- If the hashing function distributes evenly, search time is divided by the number of lists
- Balance between static size limitation and list length can be adjusted depending on data that goes in