Administrative

- CS staff (not us!) will process the wait list as space becomes available.
- If you decide to drop, please inform TeleBEARS as soon as possible to let others in.
- Get an account (if needed) and register with the course electronically (whether or not enrolled yet).
  - Should happen automatically with a CS164 class account.
  - With a named account, use the register command explicitly (and make sure you register with this course).
- Choose a partner (for projects).
- Books, etc.

Purposes of this Course

- Not everyone needs to be able to build a compiler, but there are other reasons to take this course:
  - Acquire tools for manipulating textual interfaces and data.
  - Better understand programming languages.
  - Better understand performance.
  - Learn aboutr structuring complex systems.
  - Practice programming.

Some History

- Initially, programs “hard-wired” or entered electro-mechanically: the Analytical Engine or the Jacquard loom, Eniac, punched-card-processing machines..
- Programs encoded as numbers (machine language) stored as data: Machester Mark I and the EDSAC.
- Symbolic representation of machine language (assembly language).
- FORTRAN in mid 1950s (FORmula TRANslator): algebraic notation in expressions; control structures close to machine language.
- LISP, late 1950s: dynamic, symbolic data structures.
- Algol 60: Europe’s answer to FORTRAN: modern syntax, variable scoping, and static typing. Described by BNF. Dijkstra: “Algol 60 was a marked improvement on its successors.”
- COBOL, late 1950s: business-oriented data structures, esp. records.
- The “60s explosion:” APL (array manipulation), SNOBOL (string manipulation), FORMAC (formula manipulation), and many others.
- 1967–68: Simula 67: inheritance, first “object-oriented” language

History, continued

- 1968: Algol 68 combines FORTRANish numerical and array constructs, COBOLish records, pointers, all described with an extended BNF. Pieces remain in C, but language was deemed too complex.
- 1968: announcement of the "Software Crisis." Trend toward simple languages: Pascal, Algol W, C (later).
- 1970s: emphasis on "methodology," modular programs: CLU, Smalltalk.
- Early 1970s: the Prolog language—declarative logic programming language.
- Mid 1970s: ML (MetaLanguage): type inference, pattern-directed definition.
- Mid-1970s: Ada, later (1995) in an object-oriented version.
- Complexity increases, leading to C++.
- Resimplification with Java in early 1990s.
- And more complexity with Java 1.5, C#, etc.

Problems to be Addressed

- How do we describe a programming language understandably for users, precisely for implementors.
- Given a description, how to implement it, and know it's right?
  - Testing
  - Automating the conversion of description to implementation.
- How do we save work?
  - Problem: multiple languages translated to multiple targets.
  - Automation (as above)
  - Designing so we can re-use pieces
  - Interpretation
- How do we make translators usable?
  - Reasonable handling of errors.
  - Detection of questionable constructs.
  - Compilation speed: separate compilation, JIT (Just In Time) translation.

Kinds of Translators

- The purpose of translation is to *execute* a program:

Compilation: source $\xrightarrow{translate}$ real machine language $\xrightarrow{execute}$ actions/results

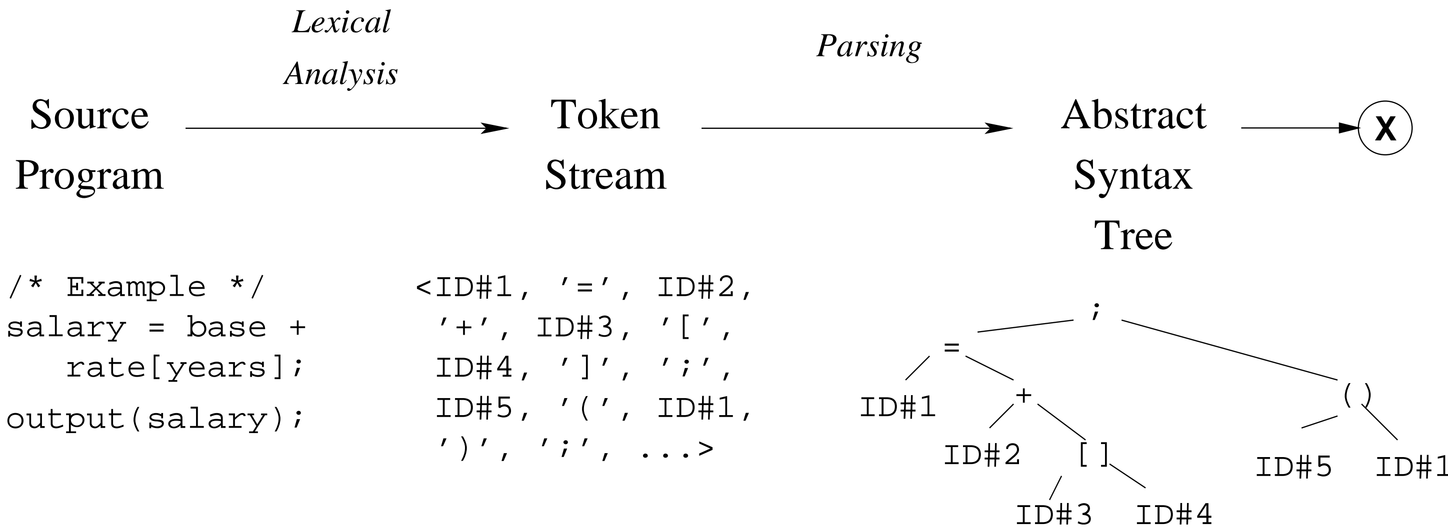
Interpretation: source $\xrightarrow{translate}$ virtual machine language $\xrightarrow{interpret}$ actions/results

Direct Execution: source $\xrightarrow{interpret}$ actions/results

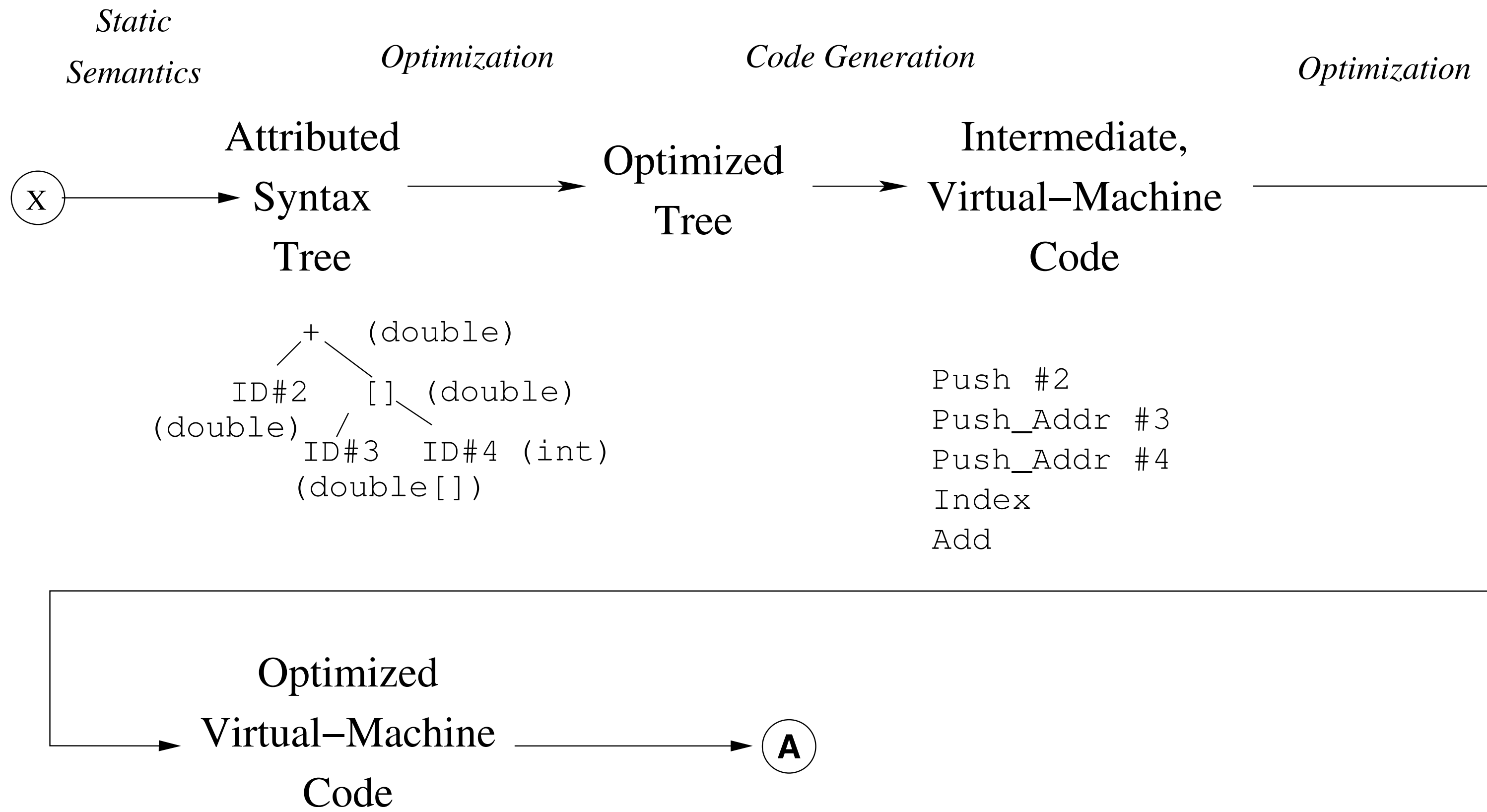
Mixed Strategies

- Most *C/C++* systems are examples of compilation.
- Lisp interpretation a kind of direct execution (trivial translation), but production Lisp systems compile.
- Java systems may interpret, compile, or compile "just in time" (while executing).
- No such thing as "an interpreted language."
- Some processors are interpreters (microcode).

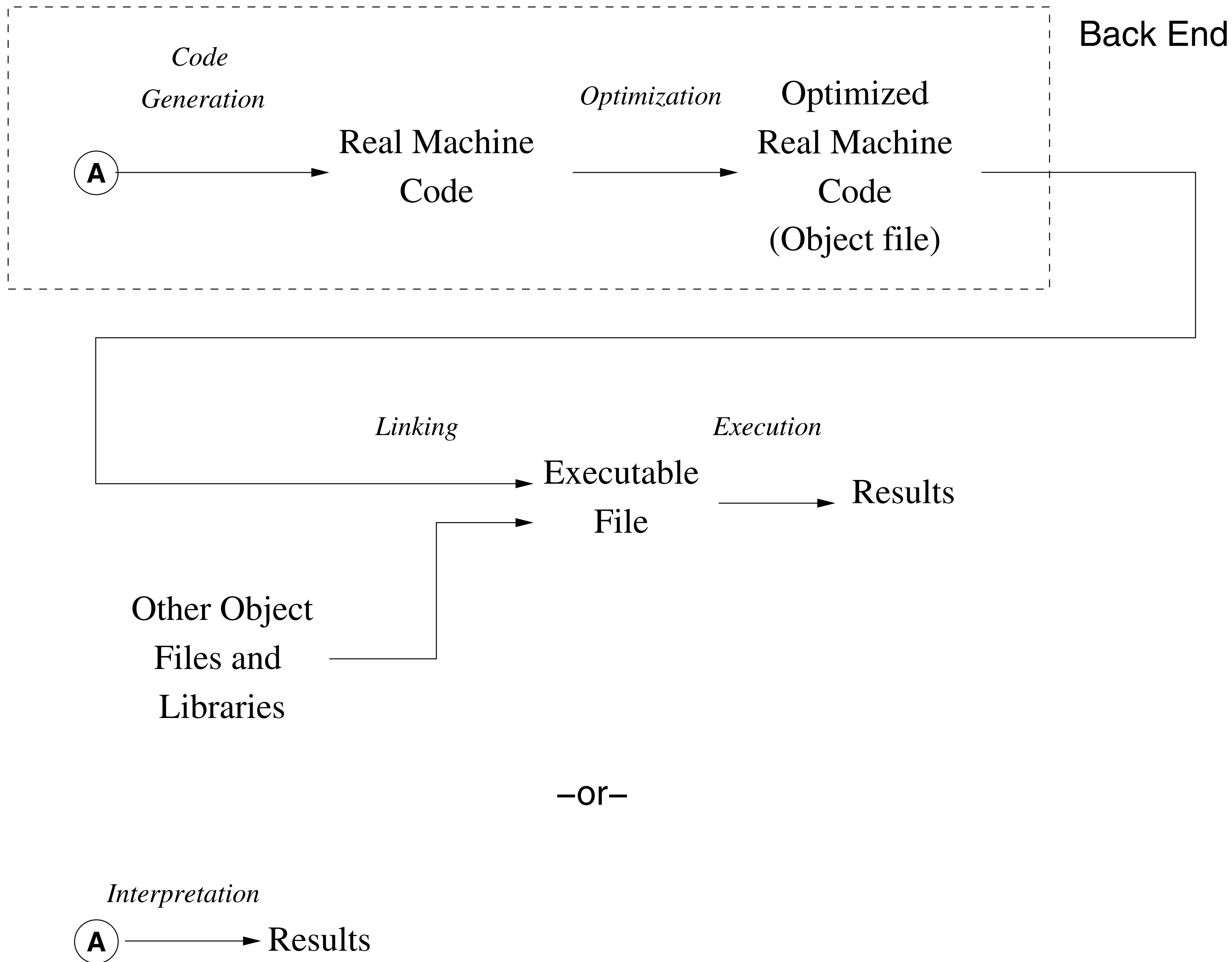
Classical Translation I



Classical Translation II



Classical Translation III



Example: FORTRAN

```
C FORTRAN (OLD-STYLE) SORTING ROUTINE
C
      SUBROUTINE SORT (A, N)
      DIMENSION A(N)
      IF (N - 1) 40, 40, 10
10    DO 30 I = 2, N
         L = I-1
         X = A(I)
         DO 20 J = 1, L
            K = I - J
            IF (X - A(K)) 60, 50, 50
C FOUND INSERTION POINT: X >= A(K)
50             A(K+1) = X
               GO TO 30
C ELSE, MOVE ELEMENT UP
60             A(K+1) = A(K)
20       CONTINUE
         A(1) = X
30    CONTINUE
40    RETURN
      END
```

```
C ----------------------------------

C MAIN PROGRAM
      DIMENSION Q(500)
100   FORMAT(I5/(6F10.5))
200   FORMAT(6F12.5)

      READ(5, 100) N, (Q(J), J = 1, N)
      CALL SORT(Q, N)
      WRITE(6, 200) (Q(J), J = 1, N)
      STOP
      END
```

Example: Algol 60

```
comment An Algol 60 sorting program;
procedure Sort (A, N)
   value N;
   integer N; real array A;
begin
   real X;
   integer i, j;
   for i := 2 until N do begin
      X := A[i];
      for j := i-1 step -1 until 1 do
         if X >= A[j] then begin
            A[j+1] := X; goto Found
         end else
             A[j+1] := A[j];
      A[1] := X;
   Found:
      end
   end
end Sort
```

Example: APL

```
⍝ An APL sorting program
∇ Z ← SORT A
  Z ← A [⍋A]
∇
```

Example: Prolog

```
/* A naive Prolog sort */

/* permutation(A,B) iff list B is a
   permutation of list A. */
permutation(L, [H | T]) :-
   append(V,[H|U],L),
   append(V,U,W),
   permutation(W,T).
permutation([], []).

/* ordered(A) iff A is in ascending order. */
ordered([]).
ordered([X]).
ordered([X,Y|R]) :- X <= Y, ordered([Y|R]).

/* sorted(A,B) iff B is a sort of A. */
sorted(A,B) :- permutation(A,B), ordered(B).
```