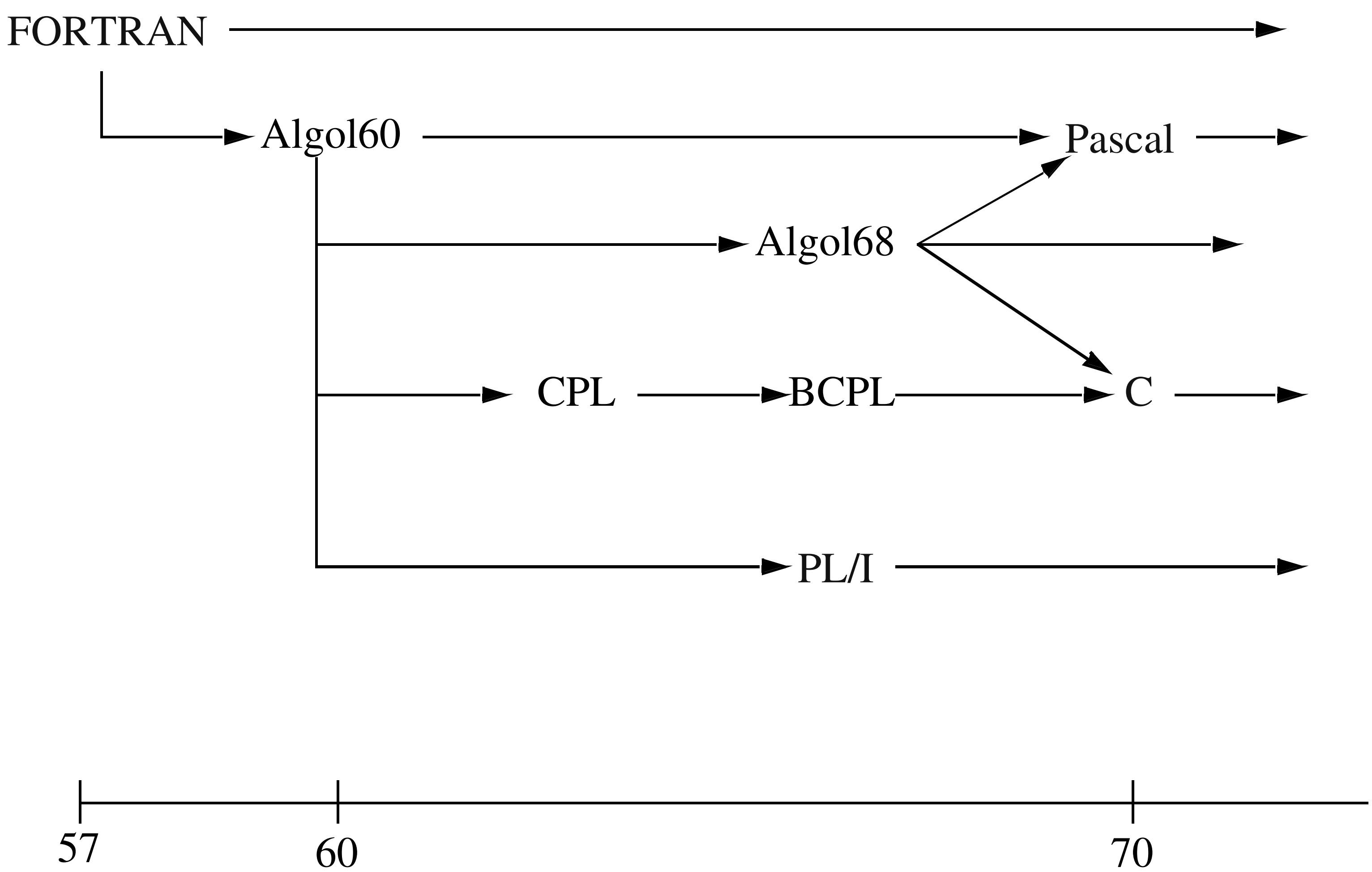
Language Paradigms

Paradigms

- Procedural (FORTRAN, ALGOL60, ALGOL68, Pascal, C)
- Object-based (CLU, Alphard, Euclid, Ada83)
- Object-oriented (Smalltalk, C++, Eiffel, Sather, Python, Ada95, Java, OCAML)
- Functional (LISP, Scheme, Common LISP, ML, Haskell)
- Logic based (Prolog, GHC)
- Parallel (CSP, Ada83, Ada95, Mentat/Legion)
- Non-deterministic (CSP, Unity)
- Aspect-oriented

Procedural: History



Procedural: The Paradigm

- (sometimes called "imperative languages")
- Collection of procedures
  - eliminates code duplication; uses parameters
- Control abstraction reached fairly advanced state
  - Functions/procedures
  - Separate compilation (often no type checking between compilation units)
- But data abstraction primitive
  - Block structure (in most of the languages)
  - Globals/locals
  - Passive local variables w/o state
  - User-defined types but with no code association

Procedural: Example

```
Program main;
GLOBAL TYPE: svec: array[1..100] of char;
s1, s2, s3: svec;
...
<initialize s1, s2>
...
s3:= concat (reverse(s1), reverse(s2));
s3[10]:= ' ';
End;
```

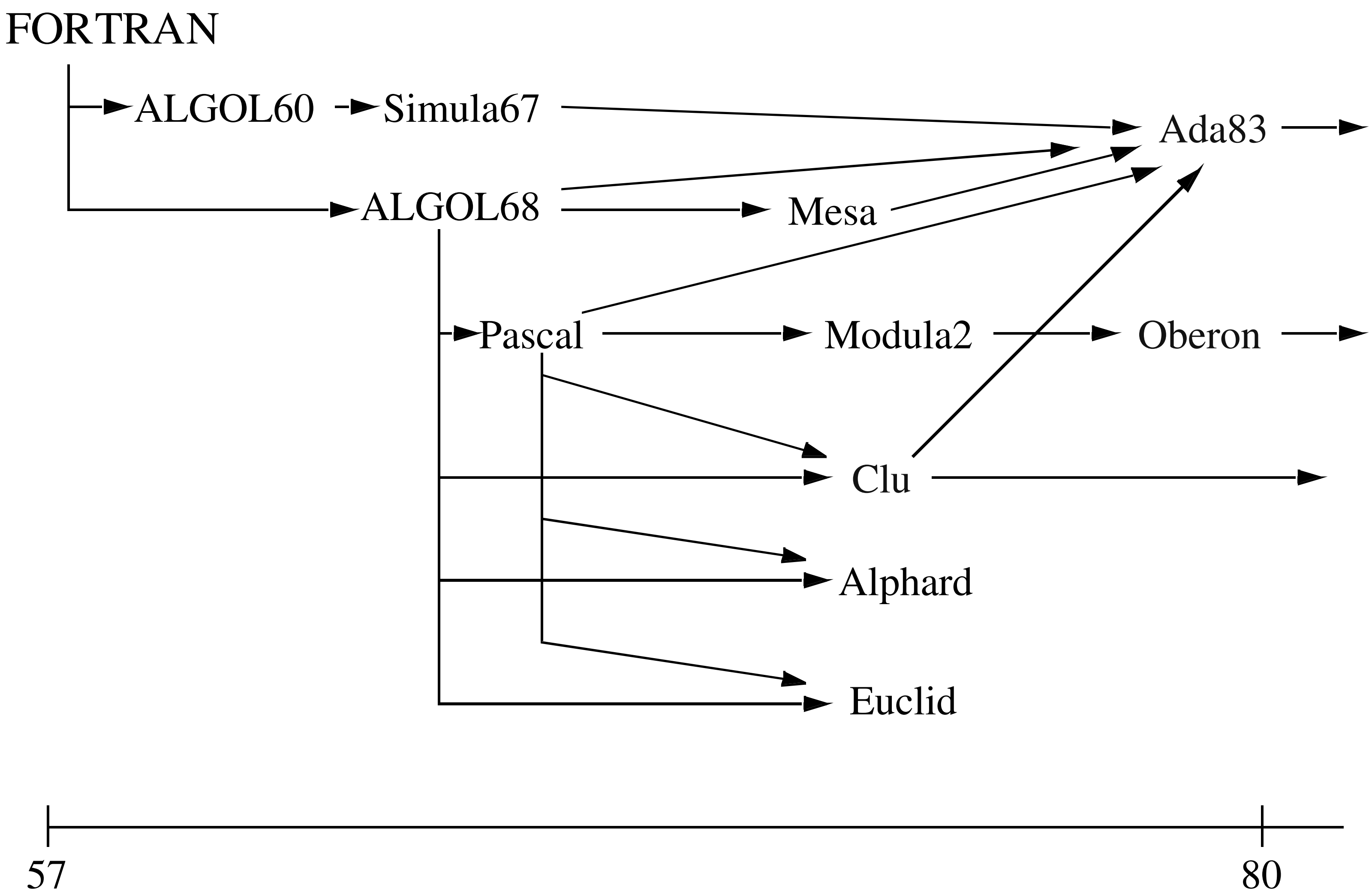
Acceptable string op?

```
Function reverse (instr: svec): svec;
i: integer;
For i:= 1 to len(instr)
  <reverse instr>;
End;

Function concat(instr1, instr2: svec): svec
<concatenate instr1 and instr2>;
End;
```

Separately Compilable Subprograms

Object-Based: History



Object-Based: The Paradigm

- Key feature is abstract data types
  - Supports Parnas's modularity principles
- Provide encapsulation mechanism for ADT's
  - for grouping data and functions/procedures associated with that data
  - limit outside access to objects inside ADT
  - Examples: Ada packages, CLU clusters, Modula2 modules
- Encapsulating mechanisms themselves tend to be typeless
- Export control mechanisms for types, variables, func/procs in ADT's
- Sometimes import control as well (Euclid)
- Encapsulated ADT's tend to be separately compilable
  - Tends to support programming in the large

Object-Based: Example

```
Program main;
Use stringops;
Import svec, concat, reverse;
s1, s2, s3: svec;
...
<initialize s1, s2>
...
s3:= concat (reverse(s1), reverse(s2));
s3[10]:= ' ';
End;
```

Compiler would disallow

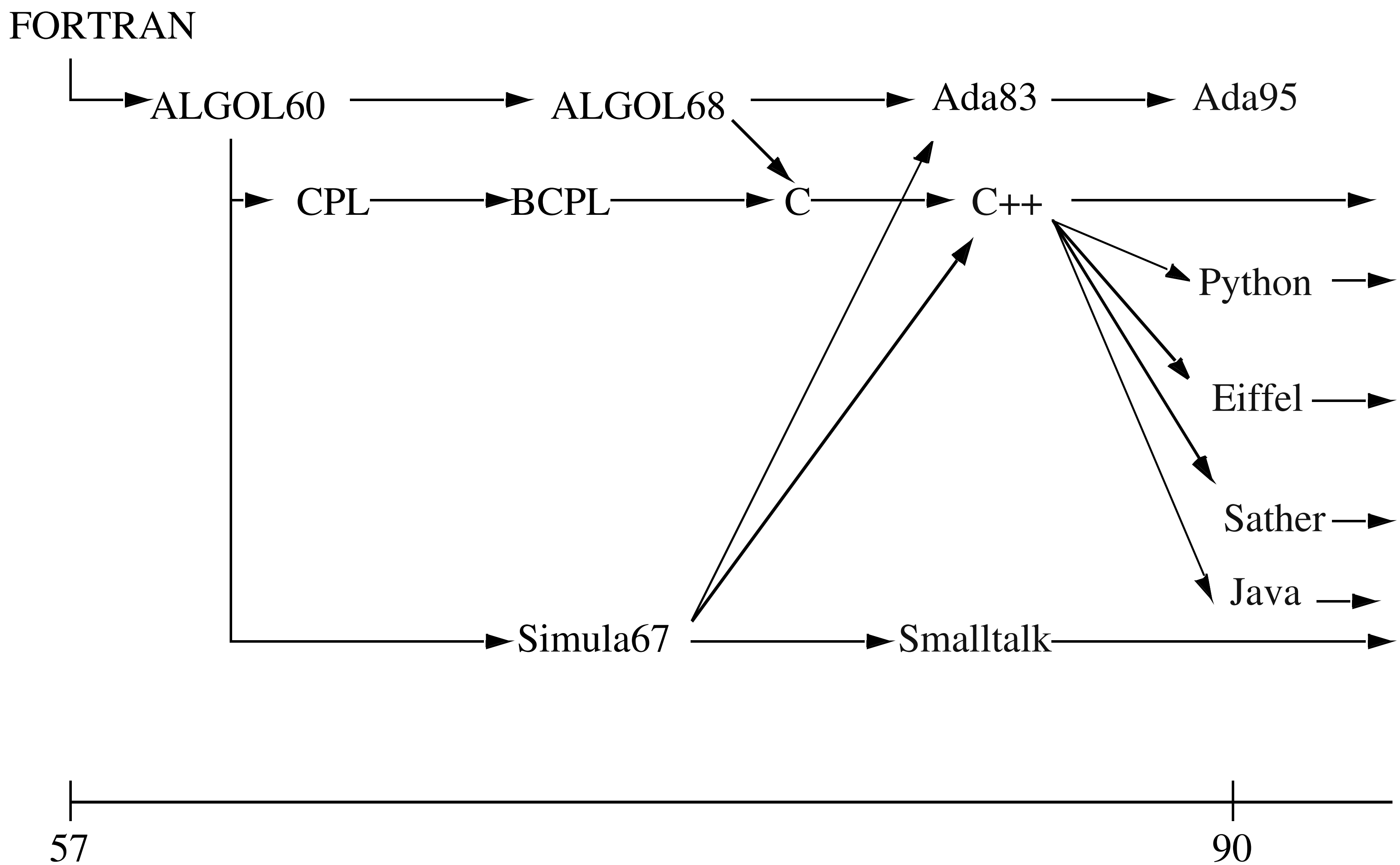
```
Package stringops;
svec: flex array[0:0];
function concat(svec, svec): svec;
function reverse(svec, svec): svec;
End Specification stringops;
```

Specification

```
Package body stringops;
 <defs for concat and reverse>;
End Package body stringops;
```

Separately compilable implementation

Object-Oriented: History



Object-Oriented: The Paradigm

- Software re-use
  - factoring out common elements
- User-defined classes
- User ability to define typed instances of a class
  - object
- Derived classes
  - requires inheritance
- User ability to use objects as first-class entities
  - assign values
  - use in expressions
  - pass as arguments

Without OOD

Conference Room Program
- size?
- furnishings?
- rented?
- rate?
- kitchen access?
- lighting?

Hotel Room Program
- rented?
- furnishings?
- rate?
- TV?
- size?

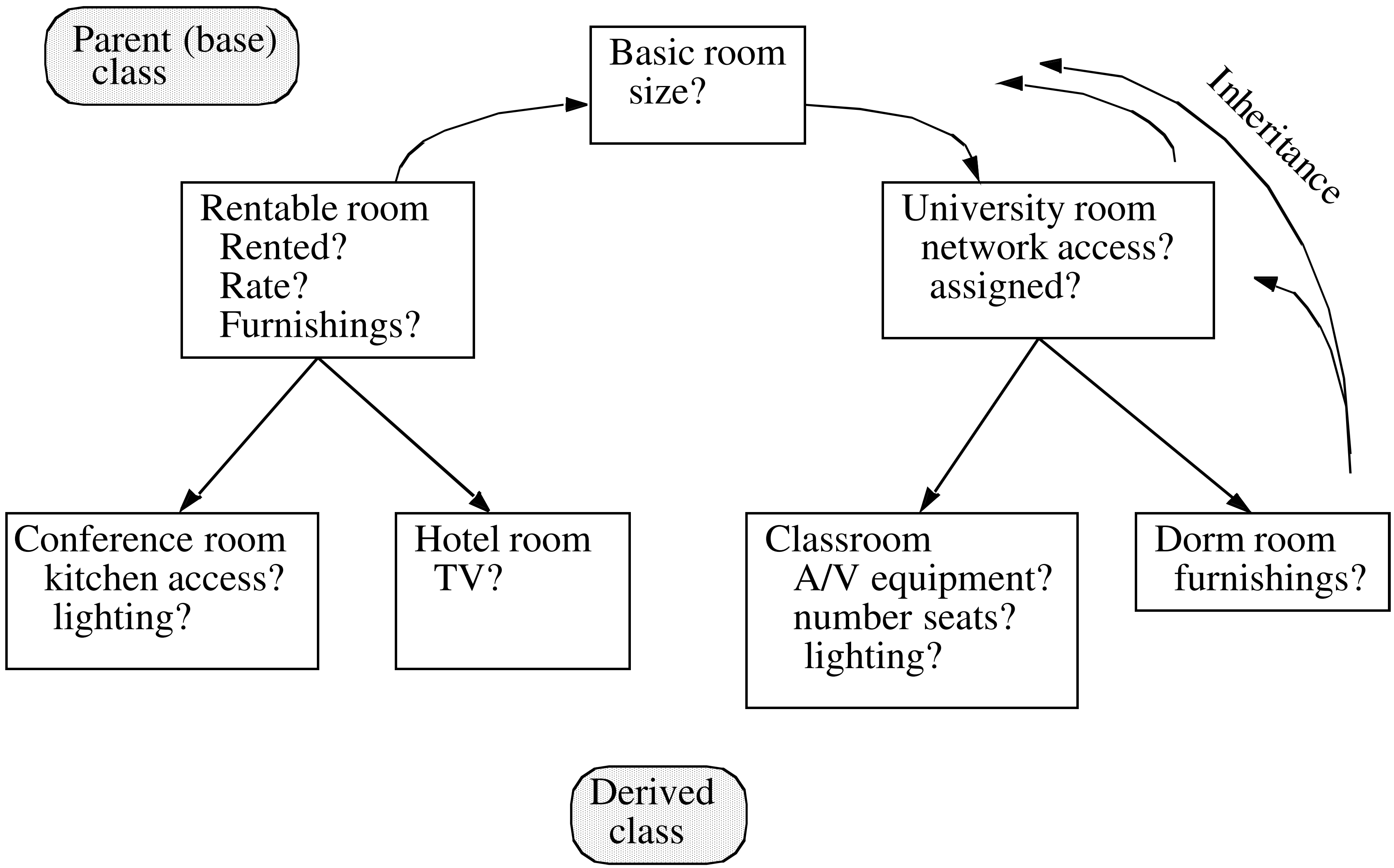
Limitations of *control abstraction*

Classroom program
- size?
- A/V equipment?
- number seats?
- lighting?
- networked?
- assigned?

Dorm Room Program
- size?
- furnishings?
- networked?
- assigned?

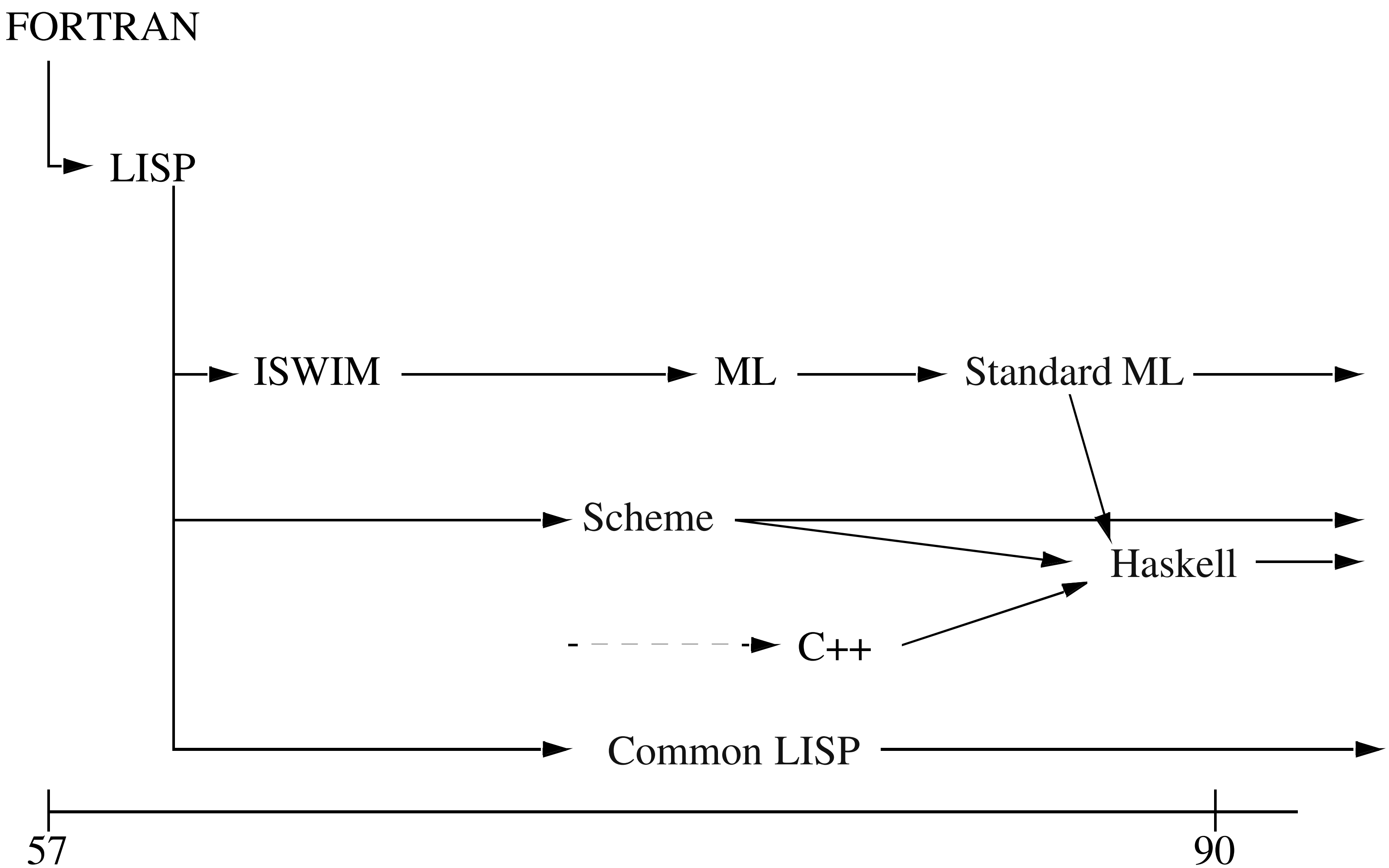
Look for common elements...

With OOD


Parent (base) class
Basic room size?
Inheritance
Rentable room Rented? Rate? Furnishings?
University room network access? assigned?
Conference room kitchen access? lighting?
Hotel room TV?
Classroom A/V equipment? number seats? lighting?
Dorm room furnishings?
Derived class

OOD Example

Functional: History



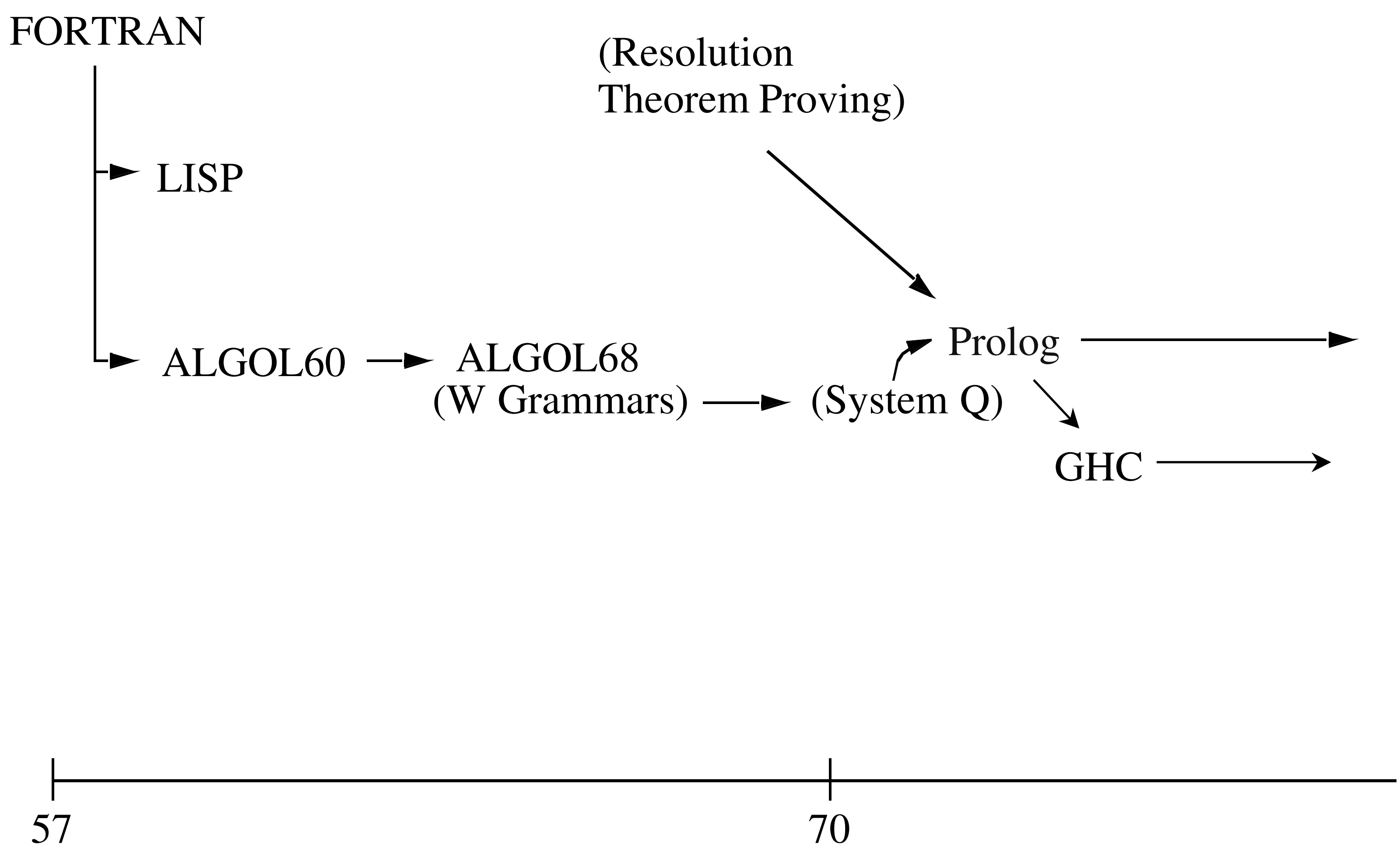
Functional: The Paradigm

- Based on Church's Lambda Calculus
- McCarthy's reaction to FLPL: FORTRAN List Processing Language
- Emphasis on conditional and recursion (lacking in FLPL)
- Assignment and iteration added only under duress
- Primary data types are atoms and lists
- Shifted from dynamic scoping (LISP) to static scoping (ML, Scheme, Haskell)
- Importance of polymorphism realized in newer languages (ML, Haskell)
- Functions have taken on first class status only in newer languages
  - passed as parameters
  - returned as values of other functions
    - supports higher order functions

Functional: Example

```
(function concat (s1  s2)
  (cond ((null s1) s2)
        (T (cons (car s1) (concat (cdr s1) s2)))
  )
)
```

Logic-Based: History



Logic-Based: The Paradigm

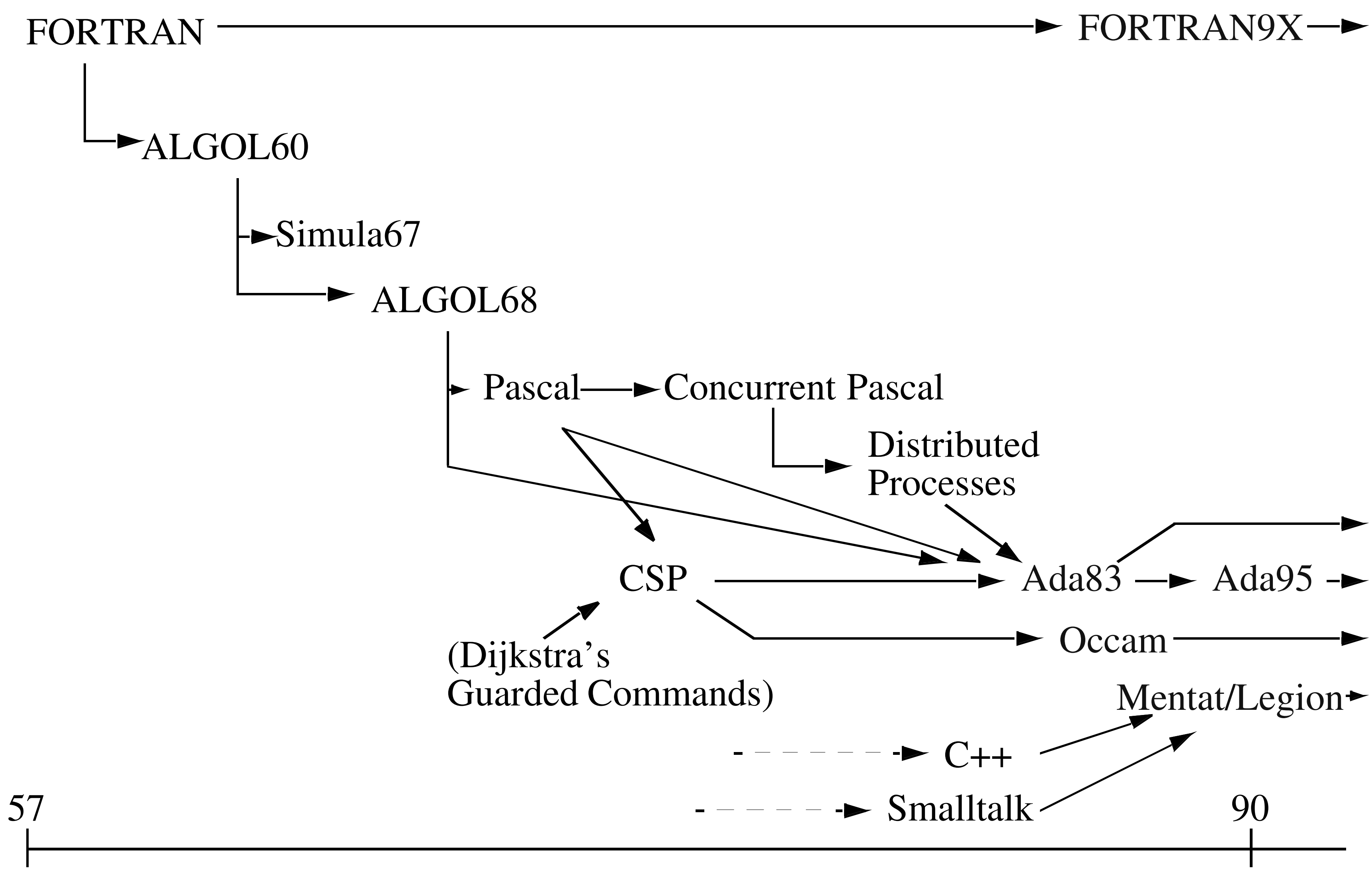
- Based on Resolution Theorem Proving
  - Horn Clauses
  - Deductive Logic
- Key concepts
  - facts
  - queries
  - unification (and two-way matching)
- Inefficient runtime
  - generally done using depth-first search
- Potential for non-termination significant
- Problems with closed world assumption

Logic-Based: Example

```
append([], Y, Y).
append([H|X], Y, [H|Z]) :- append(X, Y, Z).

     -- Prolog's version of concat
```

Parallel Languages: History



Parallel: The Paradigm

- Key Concepts:
  - Program consists of collection of processes (tasks)
  - Processes communicate
    - through shared memory
    - using messages
- Dijkstra’s Guarded Commands had profound influence on parallel language design
  - Guarded communications
- More recent trend is combining of OO and parallel
  - some confusion over messages
    - different semantics in the two paradigms

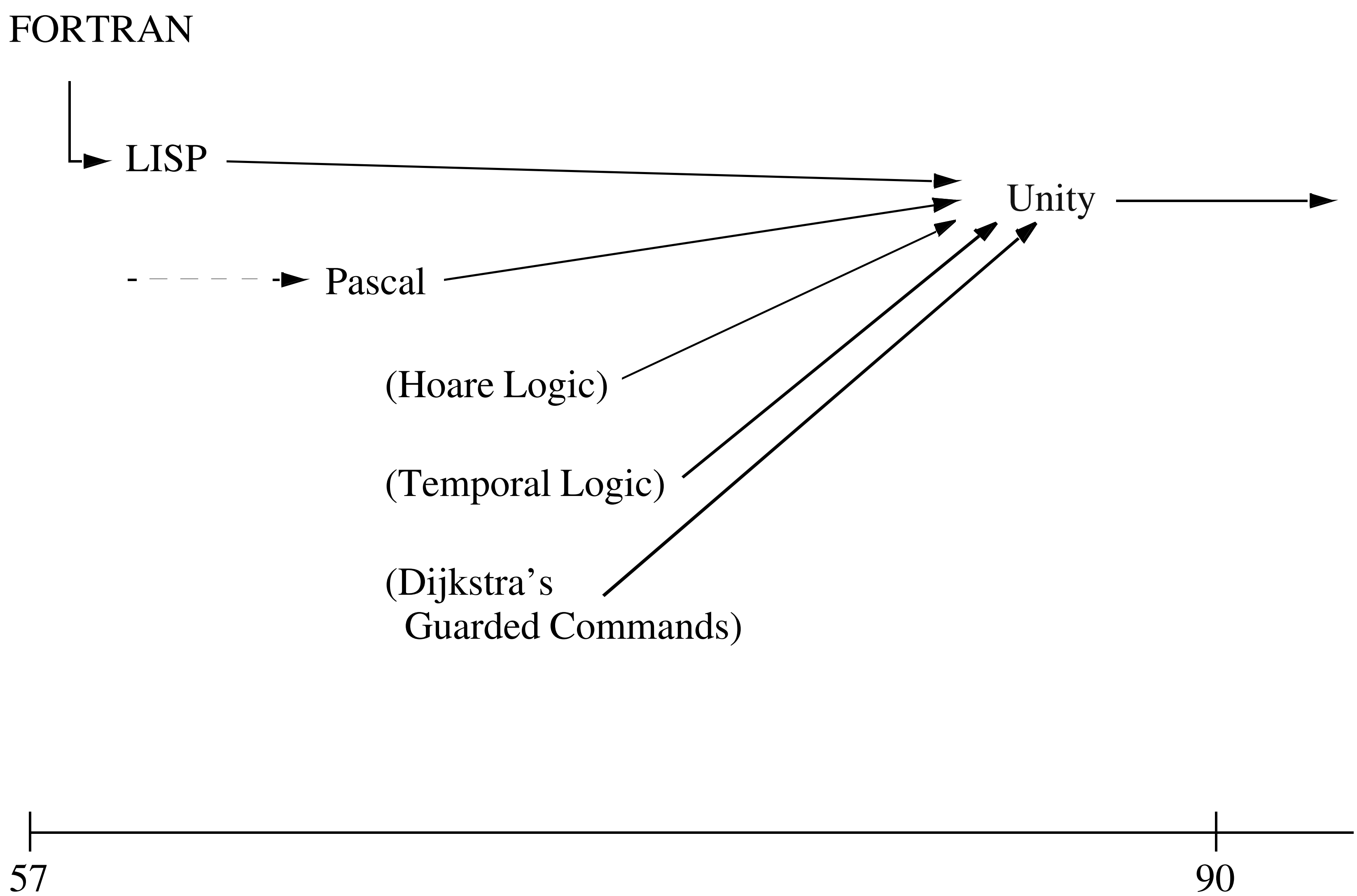
Parallel: Example

```
task type binary_semaphore is
  entry p;
  entry v;
end binary_semaphore;
task body binary_semaphore is
  begin
    loop
      accept p;
      accept v;
    end loop;
end binary_semaphore;
```

```
Process Q
  ...
  s.p;
  <critical section for Q>
  s.v;
end Q;
```

```
Process R
  ...
  s.p;
  <critical section for R>
  s.v;
end R;
```

Non-Deterministic: History



Non-Deterministic: The Paradigm

- Non-deterministic model of computation
  - Program is collection of statements
  - Statements selected and executed at random
- Interesting conditions, e.g. termination, are fixed points
  - Fixed point detection is left for "later"
- Unity programs generally very compact
- Associated proof system for program verification
- Mapping to more detailed solutions left for later.
- Goal was to define a paradigm for scientists

Non-Deterministic: Example

- Shortest path-
  - d_{xy}: distance from node x to node y
  - $sd_A(y)$: shortest distance from node A to node y

- Problem: express solution to finding distance from node A to all other nodes, y, in graph

$$sd_A(y) = \mathrm{Min}\ (sd_A(y), sd_A(x) + d(x,y)) \quad \text{all } x,y$$