III. Funktionale Programmierung

- 1. Prinzipien der funktionalen Programmierung
- 2. Deklarationen
- 3. Ausdrücke
- 4. Muster (Patterns)
- 5. Typen und Datenstrukturen
- 6. Funktionale Programmiertechniken

Kenntnis verschiedener Sprachen

- Eigene Ideen bei der Software-Entwicklung können besser ausgedrückt werden
- Nötig, um in konkreten Projekten geeignete Sprache auszuwählen
- Erleichtert das Erlernen weiterer Programmiersprachen
- Nötig für den Entwurf neuer Programmiersprachen

Übersicht

Imperative Sprachen

- Folge von nacheinander ausgeführten Anweisungen

Prozedurale Sprachen

- Variablen, Zuweisungen, Kontrollstrukturen

Objektorientierte Sprachen

- Objekte und Klassen
- ADT und Vererbung

Deklarative Sprachen

- Spezifikation dessen, *was* berechnet werden soll
- Compiler legt fest, wie Berechnung verläuft

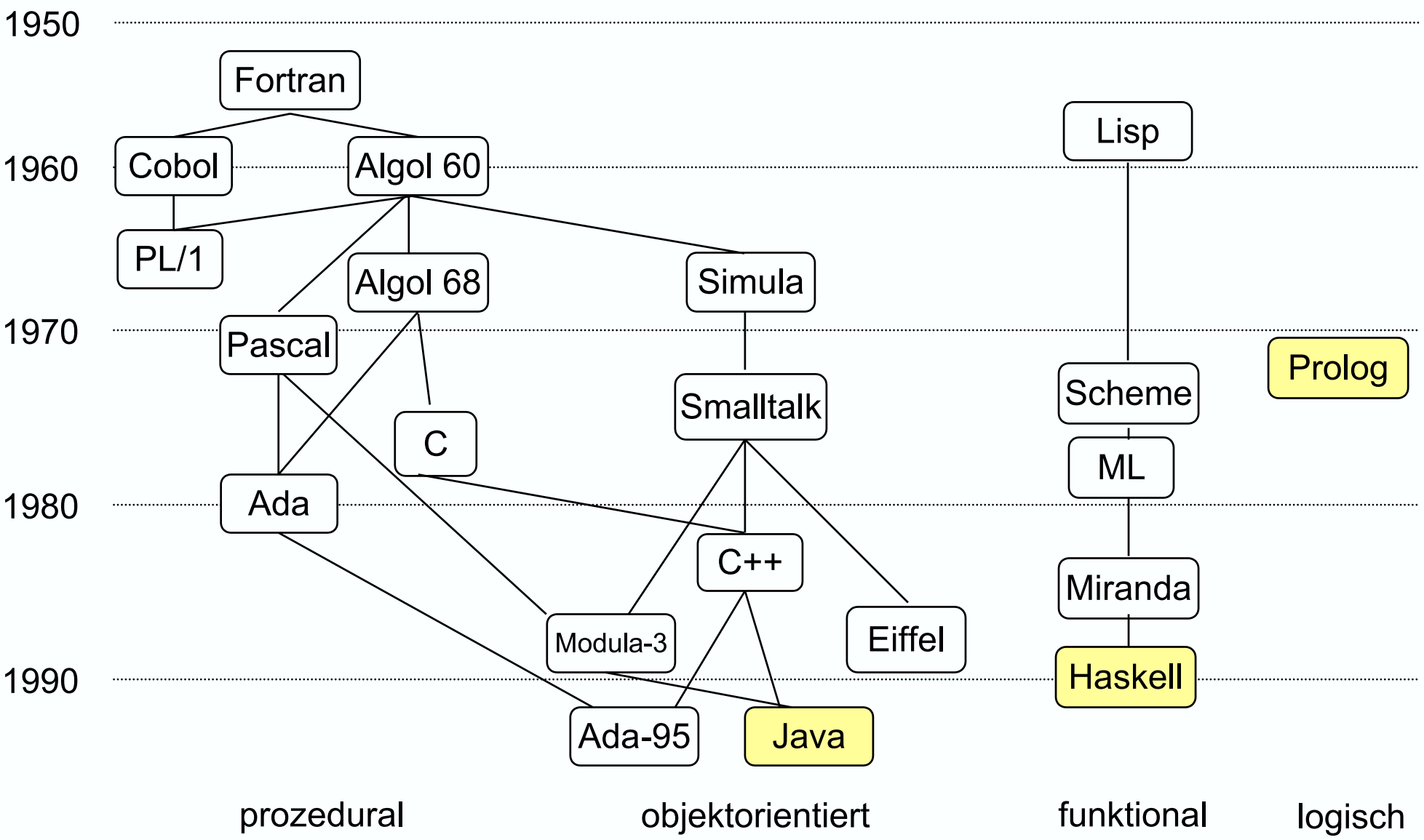
Funktionale Sprachen

- keine Seiteneffekte
- Rekursion

Logische Sprachen

- Regeln zur Definition von Relationen

Wichtige Programmiersprachen



Imperative Programmierung (*Java*)

```
Eingabe: Eine Liste x
Ausgabe: Länge der Liste x
Algorithmus:

1. Setze n = 0.

2. Falls x nicht die
   leere Liste ist, dann:

   2a. Lösche das erste
       Element von x.

   2b. Erhöhe n um 1.

   2c. Gehe zu Schritt 2.

3. Liefere n zurück.
```

```
class Element              {
 Vergleichbar wert;
 Element next; ...         }
```

```
public class Liste         {
 Element kopf;
 ...
 static int len (Liste x) {
      int n = 0;
      while (x.kopf != null) {
       x.kopf = x.kopf.next;
       n = n + 1;
      }
      return n;
 }
```

Kennzeichen imperativer Programmierung

- Programm besteht aus einzelnen Anweisungen, die nacheinander abgearbeitet werden.
- Verschiedene Kontrollstrukturen, um Programmablauf zu steuern.
- Abarbeitung einer Anweisung ändert Werte der Variablen.
- Seiteneffekte
- Programmierer muss Speicherorganisation bedenken

Funktionale Programmierung (*Haskell*)

```
A.   Falls die Liste x leer ist, so ist len(x) = 0.

B.   Falls die Liste x nicht leer ist und
     "rest" die Liste x ohne ihr erstes Element ist,
     so ist len(x) = 1 + len(rest).
```

Programm in *Haskell*:

```
len []                =    0

len (kopf : rest)     =    1 + len rest
```

Ausführung funktionaler Programme

```
len []                  =    0

len (kopf : rest)       =    1 + len rest
```

```
    len [15, 70, 36]

=   1 + len [70, 36]

=   1 + 1 + len [36]

=   1 + 1 + 1 + len []

=   1 + 1 + 1 + 0

=   3
```

Kennzeichen funktionaler Programme

- **Rekursion statt Schleifen**
- **Keine Seiteneffekte (*referentielle Transparenz*)**
- **Funktionen als gleichberechtigte Datenobjekte (*Funktionen höherer Ordnung*)**
- **Verwendung von einer Funktions-Implementierung für verschiedene Typen (*Polymorphismus*)**
- **Programme sind kürzer, klarer, besser zu warten, zuverlässiger, schneller zu erstellen**

Funktionale Programmierung (*Haskell*)

- **Haskell-Interpreter *Hugs***
  - Eingabe: Auszuwertender Ausdruck
  - Ausgabe: Ergebnis des ausgewerteten Ausdrucks
- **Bsp:**
  - Eingabe: len [15, 36, 70]
  - Ausgabe: 3
- **Interpreter**
  - führen jede Zeile des Programms nacheinander aus.
  - Vorteil: Keine vorherige Übersetzung nötig, gut zur Programmentwicklung geeignet.
  - Nachteil: Ineffizienter als Compiler.