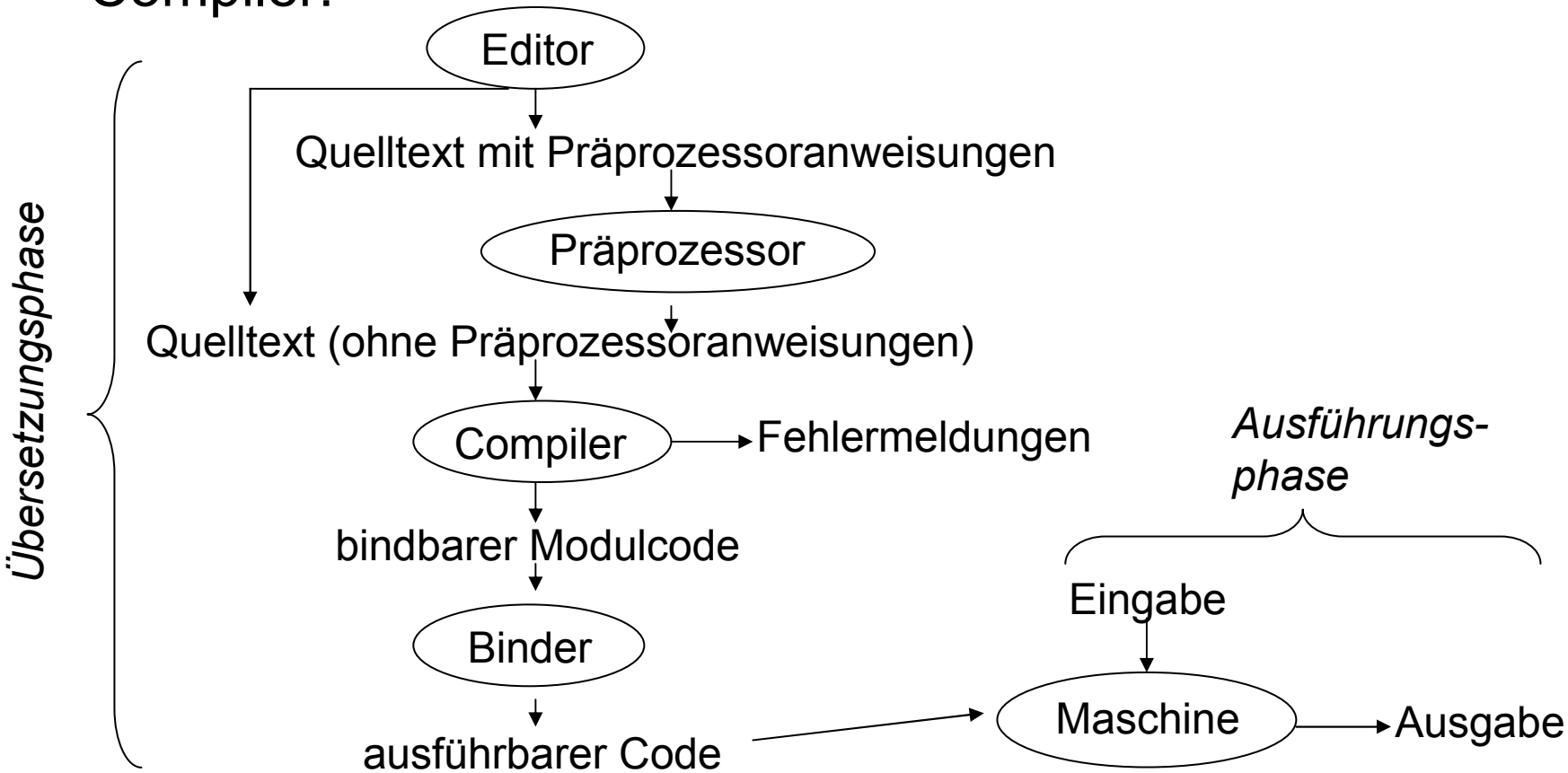
Programmierumgebungen und Sprachwerkzeuge

„semantische Lücke“ zwischen Programmiersprache und Maschine:

	Programmiersprache	Maschine
Datenstrukturen	komplizierte Datenstrukturen	primitive Maschinenworte
Kontrollstrukturen	ausgefeilte Strukturen (Schleifen,..)	bedingte und unbedingte Sprünge
Prozeduren/ Funktionen	lokale u. globale Daten, Parameter-übergabe, Rekursion	Unterprogramm-sprung

Programmierumgebungen und Sprachwerkzeuge

- Zur Überbrückung der Lücke:
  Compiler (Übersetzer) oder *Interpreter*
- Compiler:



Programmierumgebungen und Sprachwerkzeuge

Präprozessor:

- bearbeitet Programmtext vor eigentlichem Übersetzer
- Kommandos zur Textsubstitution
- ist sprachunabhängig
- Beispiel:

```
#include <stdio.h>                          Datei hier einfügen
#include “induce.h“
#define MAXATTR 265                         benannte Konstante
#define ODD(x)  ((x)%2 == 1)                parametrisierte Text-
#define EVEN(x) ((x)%2 == 0)                Makros
static void early (int sid)
     {int attrs[MAXATTR];                   Konstante wird substituiert
      …
      if (ODD(num)) num--;                  Makro wird substituiert
#ifndef GOTO                                bedingter Textblock
     printf(“early for %d num: %d\n“, sid, num);
#endif
```

Programmierumgebungen und Sprachwerkzeuge

Compiler:

- Quellcode → Maschinenprogramm
- mehrere Phasen:
  1. Lexikalische Analyse: Eingabe wird zu sinnvollen Einheiten (*token*) zusammengefasst
  2. Syntax-Analyse: Token werden zu syntaktischen Einheiten (z.B. Ausdruck, Anweisung, Deklaration) zusammengefasst; sog. Parsen erzeugt Parse-Tree
  3. Semantische Analyse: Regeln überprüfen, ob Ausdruck o. Anweisung im Kontext stimmt (z.B. ob Typen und Operationen passen); Parse-Tree wird um Typinformationen und Deklarationsort erweitert

Programmierumgebungen und Sprachwerkzeuge

Compiler:

4. Zwischencode-Generator: Programm in einfache Zwischensprache übersetzen
5. Code-Optimierer: Standard-Optimierungen auf dem Zwischencode

   z.B. Eleminieren von Anweisungen, die nie erreicht werden; evtl. Eleminieren von Anweisungen wie x=y und im weiteren statt x y nehmen; usw.
6. Code-Generierung: optimierter Zwischencode wird in Maschinencode überführt

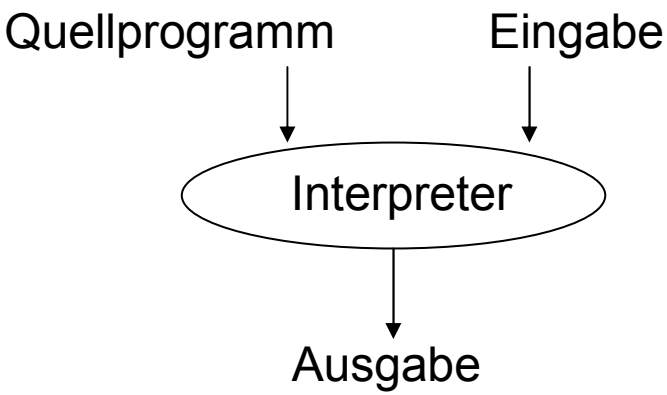
Programmierumgebungen und Sprachwerkzeuge

Compiler:

- Vorteile:
  - Quelltext wird während Übersetzung auf Fehler untersucht
  - bei Programmausführung besonders hohe Ausführungsgeschwindigkeit (da optimiert)
  - effizient: Programm muss nicht bei jedem Programmlauf analysiert werden
  - Compilersprachen können größer und komplexer sein: Entwicklungsumgebung auf leistungsfähiger Maschine, erzeugtes Programm auf kleinerer Maschine lauffähig
- Nachteile:
  - Compilersprachen haben relativ starres Typsystem
  - erschweren Zuordnung von Laufzeitfehlern zu Programmtext

Programmierumgebungen und Sprachwerkzeuge

Interpreter:



- zur Laufzeit liest Interpreter Quelltext,
- arbeitet ihn Befehl für Befehl sofort ab

Programmierumgebungen und Sprachwerkzeuge

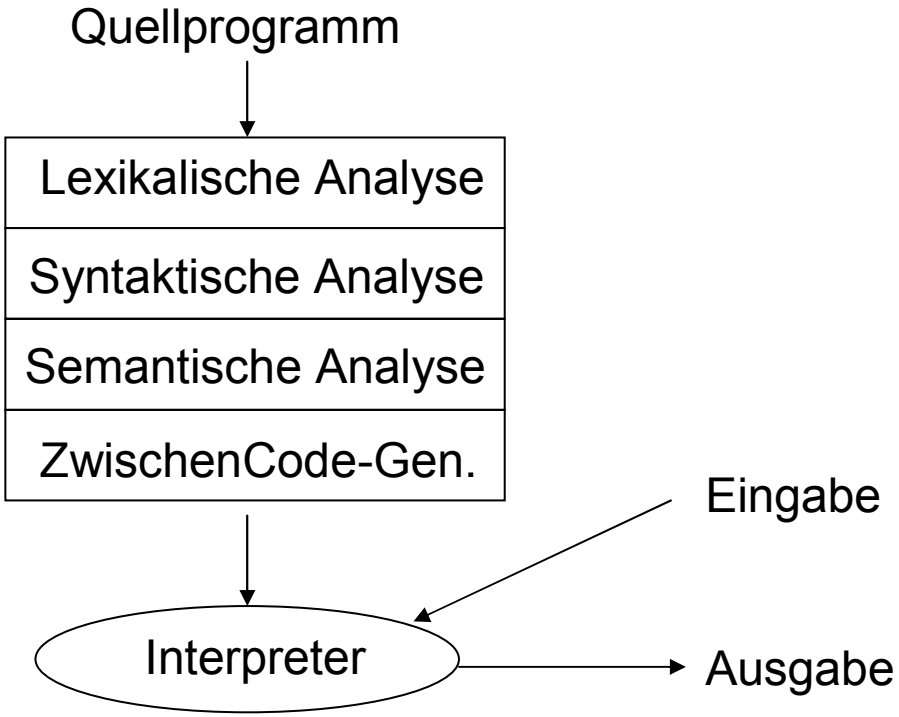
Interpreter:

- Vorteile:
  - Interpreter kann die Ausführung überwachen
  - aufwendiger Edit-Compile-Test-Zyklus entfällt: gut bei kleinen Programmänderungen, Try-and-error-Programmierung, Prototyping

- Nachteile:
  - Abarbeitung langsamer als bei compilierten Programmen
  - Maschine muss gewisse Größe haben, damit Interpreter ablauffähig ist

Programmierumgebungen und Sprachwerkzeuge

Hybride Systeme:

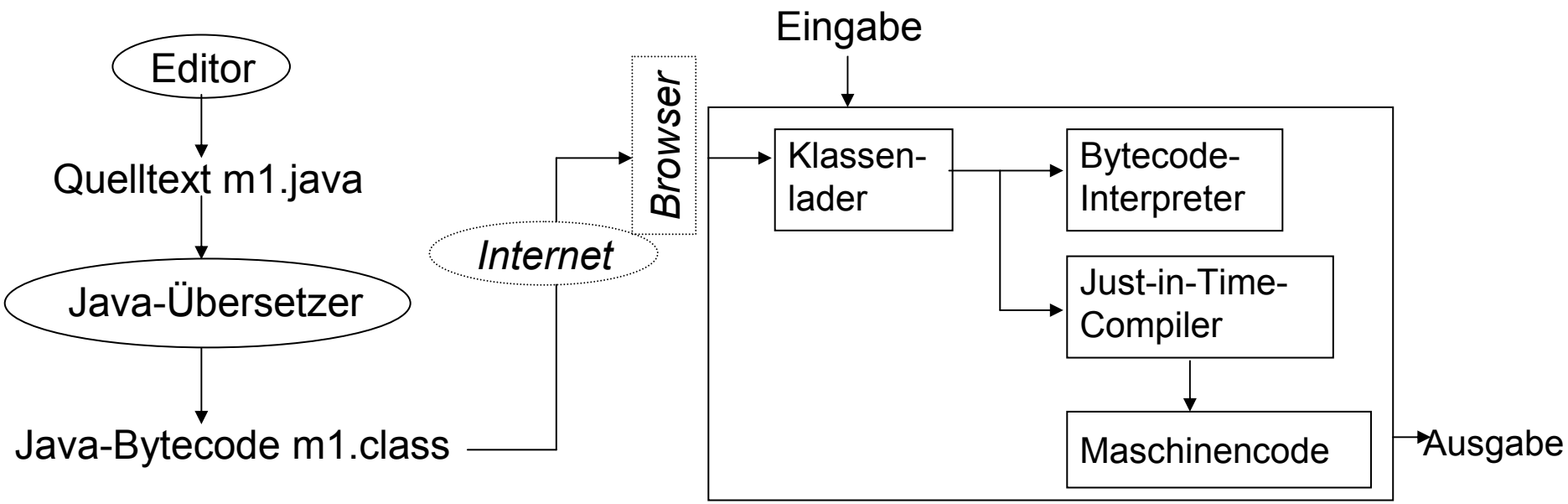


- Quelltext durchläuft Analyse-Part des Compilierens
- Zwischensprache wird interpretiert

Programmierumgebungen und Sprachwerkzeuge

Java-Übersetzer (heute):

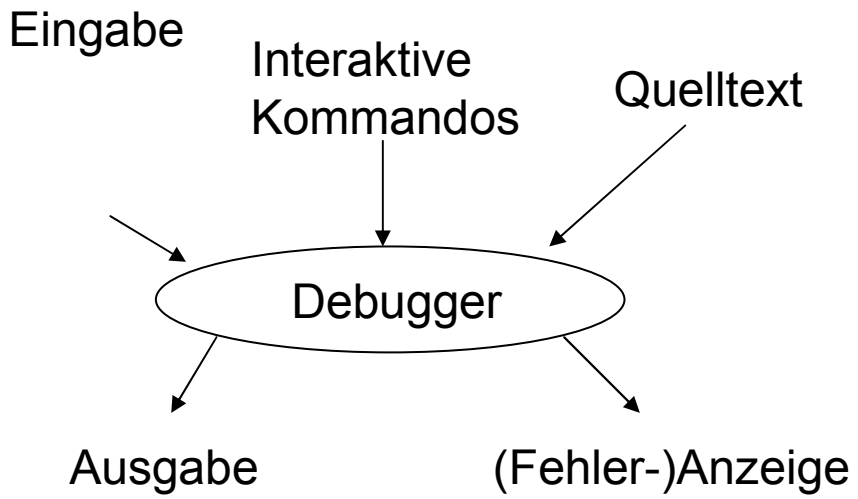
- Kombination aus Compiler und Interpreter
- Quelltext wird in Java-Bytecode **übersetzt**
- Bytecode wird von Zielmaschine (JVM) **interpretiert**
- **JIT-System** kompiliert Methoden zur Aufrufzeit, speichert Maschinencode im Cache



Programmierumgebungen und Sprachwerkzeuge

Weitere Sprachwerkzeuge:

- Entwicklungsumgebungen (eclipse, NetBeans, JBuilder, MS Visual Studio,…)
- Testhilfen/Testumgebungen (Debugger):



Programmierumgebungen und Sprachwerkzeuge

- Sprachdokumente
  - **Reference Manual**: enthält verbindliche Sprachdefinition, beschreibt alle Konstrukte und Eigenschaften vollständig und präzise,

    kann standardisiert sein (ANSI,ISO,DIN,…),
  - **Formale Sprach-Definition**: verwendet formales Kalkül

    (KFG, …), für Implementierer oder Sprachforscher
  - **Benutzer-Handbuch**: erläutert typische Anwendungen der Sprachkonstrukte, meist mit vielen Programmausschnitten
  - **Lehrbuch**: didaktische Einführung in den Gebrauch der Sprache, mit Programmbeispielen, selten vollständig