12. Mengenorientierte DB-Schnittstelle

- **Übersetzungsverfahren für Datenbanksprachen**
  - Vorgehensweise
  - Übersetzung vs. Interpretation
- **Formen der Wirtsspracheneinbettung**
  - Direkte Einbettung
  - Aufruftechnik
- **Anfrageoptimierung**
  - Anfragedarstellung
  - Standardisierung und Vereinfachung
  - Restrukturierung und Transformation
  - Kostenmodelle
  - Erstellung und Auswahl von Ausführungsplänen
- **Code-Erzeugung**
  - Aufbau von Zugriffsmoduln
  - Bindezeitpunkte für Anweisungstypen
- **Behandlung von Ad-hoc-Anfragen**

Logische Datenstrukturen

- **Charakterisierung der Abbildung**

```
SELECT PNR, ABT-NAME
FROM ABTEILUNG, PERS, FAEHIGKEIT
WHERE BERUF = 'PROGRAMMIERER' &
      FAEHIGKEIT.FA-NR = PERS.FA-NR &
      PERS.ABT-NR = ABTEILUNG.ABT-NR
```

Abbildungsfunktionen

- Sichten	↔ Basisrelationen
- Relationale Ausdrücke	↔ Logische Zugriffspfade
- Satzmengen	↔ Einzelne Sätze, Positionsanzeiger

```
FETCH FAEHIGKEIT USING ...
FETCH NEXT PERS ...
FETCH OWNER WITHIN ...
```

- **Eigenschaften der oberen Schnittstelle**
  - Zugriffspfad-unabhängiges (relationales) Datenmodell
  - Alle Sachverhalte und Beziehungen werden durch Werte dargestellt
  - Nicht-prozedurale (deskriptive) Anfragesprachen
  - Zugriff auf Satzmengen

Aspekte der Benutzung deskriptiver DB-Sprachen

- **Prozedurale Sprachen**
  - Sie erlauben direkte Abbildung der DML-Befehle auf interne Satzoperationen des DBVS (~1:1)
  - **Verantwortung für die Zugriffspfadauswah**l liegt beim Programmierer; er bestimmt die Art und Reihenfolge der Zugriffe durch Navigation
  - Bei der Übersetzung sind lediglich Namensauflösung und Formatkonversionen erforderlich

- **Typische Beispiele** (Netzwerkmodell)
  - FIND ANY PERS bezieht sich auf die (vorausgesetzte) Klausel LOCATION MODE IS CALC des Satztyps PERS
  - FIND NEXT PERS WITHIN BESCHÄFTIGT SET bezieht sich auf eine relative Position (Currency) in einer Setstruktur
  - FIND OWNER WITHIN BESCHÄFTIGT SET stellt den OWNER-Satz der ‘current’ Set-Ausprägung zur Verfügung.

    ➥ Lediglich bei der allg. Suchanfrage fallen gewisse Optimierungsaufgaben an; sie ist jedoch auf einen Satztyp beschränkt

- **Deskriptive Anfragen erfordern zusätzlich**
  - Überprüfung auf syntaktische Korrektheit (komplexere Syntax)
  - Überprüfung von Zugriffsberechtigung und Integritätsbedingungen
  - **Anfrageoptimierung** zur Erzeugung einer effizient ausführbaren Folge interner DBVS-Operationen

Aspekte der Benutzung deskriptiver DB-Sprachen (2)

- **Zentrales Problem**
  - Umsetzung deskriptiver Anfragen in eine zeitoptimale Folge interner DBVS-Operationen
  - Anfrageübersetzer/-optimierer des DBVS ist im wesentlichen für eine effiziente Abarbeitung verantwortlich, nicht der Programmierer

- **Hohe Komplexität der Übersetzung,**
  da die Auswahlmächtigkeit
  - an der Prädikatenlogik erster Stufe orientiert ist; durch zusätzliche Prädikate wie EXISTS, MATCHES, NULL, LIKE u. a. wird diese sogar deutlich übertroffen
  - nicht auf einen Satztyp beschränkt ist
  - unabhängige oder korrelierte Teilanfragen zur Bestimmung von Such-argumenten in beliebiger Schachtelungstiefe zuläßt
  - zusätzlich den Einsatz von Built-in- und Sortier-Funktionen auf Partitionen der Satzmenge gestattet

- **Zusätzliche Anforderungen**
  - auch die Manipulationsoperationen sind mengenorientiert
  - referentielle Integrität ist aktiv mit Hilfe referentieller Aktionen zu wahren
  - Operationen können sich auf Sichten von Relationen beziehen
  - vielfältige Optionen der Datenkontrolle sind zu berücksichtigen

- **Anfrageformulierung**
  - Formulierung von ’nicht angemessenen’ Anfragen (ohne Zugriffspfad-unterstützung) erfordert in navigierenden Anfragesprachen einen erheblichen Programmieraufwand
  - in deskriptiven Anfragesprachen dagegen sind sie genauso leicht zu formulieren wie ‘günstige’ Anfragen

    ➥ Die Ausführung ist in beiden Fällen gleich langsam

Beispiele deskriptiver SQL-Anfragen

- **B1: Einfache Anfrage**

```
SELECT PNR, PNAME, GEHALT/12
FROM PERS
WHERE BERUF = W
      AND PROV > GEHALT
```

Ersetzung durch

```
DECLARE C1 CURSOR FOR SELECT PNR, PNAME, GEHALT/12
INTO :X, :Y, :Z
FROM PERS
WHERE BERUF =: W
      AND PROV > GEHALT
```

mit Operatoren

```
      OPEN C1
      FETCH C1 INTO :X, :Y, :Z
      CLOSE C1
```

- **B2: Komplexe Anfrage**

mit Verbundoperation sowie unabhängige (T2) und korrelierte (T3) Teilanfragen

```
T1   SELECT    P.PNR, P.NAME, A.ANAME
     FROM      PERS P, ABT A
     WHERE     P.ANR = A.ANR

T2   AND P.GEHALT   <   (SELECT MAX (PROV)
                         FROM PERS)

T3   AND P.GEHALT   >   (SELECT AVG (PROV)
                         FROM PERS
                         WHERE ANR = P.ANR
```

Beispiele deskriptiver SQL-Anfragen (2)

- **B3: Vereinfachte SQL-Anfrage**, durch Tool zur Entscheidungsunterstützung (Online Analytical Processing, OLAP) und GUI-Nutzung automatisch erzeugt.

```
select distinct a.fn
from T1 a
where a.owf =
        (select min (b.owf)
        from T1 b
        where (1=1) and (b.aid='SAS' and
                b.fc in (select c.cid
                        from T2 c
                        where c.cn='HKG') and
                b.tc in (select d.cid
                        from T2 d
                        where e.cn='HLYD') and
                b.fid in (select e.fid
                        from T3 e
                        where e.did in
                                (select f.did
                                from T4 f
                                where f.dow='saun')) and
                b.fdid in (select g.did
                        from T4 g
                        where g.dow='saun'))) and
                (1=1) and (a.aid='SAS' and
                a.fc in (select h.cid
                        from T2 h
                        where h.cn='HKG') and
                a.tc in (select i.cid
                        from T2 i
                        where i.cn='HLYD') and
                a.did in (select j.fid
                        from T3 j
                        where j.did in
                                (select k.did
                                from T4 k
                                where k.dow='saun')) and
                a.fdid in (select 1.did
                        from T4 1
                        where 1.dow='saun'))
```

Auswertung von DB-Anweisungen

- **Verarbeitungsschritte** zur Auswertung von DB-Anweisungen:

1. Lexikalische und syntaktische Analyse

- Erstellung eines Anfragegraphs (AG) als Bezugsstruktur für die nachfolgenden Übersetzungsschritte
- Überprüfung auf korrekte Syntax (Parsing)

2. Semantische Analyse

- Feststellung der Existenz und Gültigkeit der referenzierten Tabellen, Sichten und Attribute
- Einsetzen der Sichtdefinitionen in den AG
- Ersetzen der externen durch interne Namen (Namensauflösung)
- Konversion vom externen Format in interne Darstellung

3. Zugriffs- und Integritätskontrolle

sollen aus Leistungsgründen, soweit möglich, schon zur Übersetzungszeit erfolgen

- Zugriffskontrolle erfordert bei Wertabhängigkeit Generierung von Laufzeitaktionen
- Durchführung einfacher Integritätskontrollen (Kontrolle von Formaten und Konversion von Datentypen)
- Generierung von Laufzeitaktionen für komplexere Kontrollen

Auswertung von DB-Anweisungen (2)

4. Standardisierung und Vereinfachung

dienen der effektiveren Übersetzung und frühzeitigen Fehlererkennung

- Überführung des AG in eine Normalform
- Elimination von Redundanzen

5. Restrukturierung und Transformation

- Restrukturierung zielt auf globale Verbesserung des AG ab; bei der Transformation werden ausführbare Operationen eingesetzt
- Anwendung von heuristischen Regeln (**algebraische Optimierung**) zur Restrukturierung des AG
- Transformation führt Ersetzung und ggf. Zusammenfassen der logischen Operatoren durch Planoperatoren durch (nicht-algebraische Optimierung): Meist sind mehrere Planoperatoren als Implementierung eines logischen Operators verfügbar
- Bestimmung alternativer Zugriffspläne (**nicht-algebraische Optimierung**): Meist sind viele Ausführungsreihenfolgen oder Zugriffspfade auswählbar
- Bewertung der Kosten und Auswahl des günstigsten Ausführungsplanes

➥ Schritte 4 + 5 werden als Anfrageoptimierung zusammengefasst

6. Code-Generierung

- Generierung eines zugeschnittenen Programms für die vorgegebene (SQL-) Anweisung
- Erzeugung eines ausführbaren Zugriffsmoduls
- Verwaltung der Zugriffsmodule in einer DBVS-Bibliothek

Übersetzung vs. Interpretation

- **Was heißt „Binden“?**

AP:

```
SELECT   Pnr, Name, Gehalt
FROM     Pers
WHERE    Beruf = ‘Programmierer‘
```

DB-Katalog:

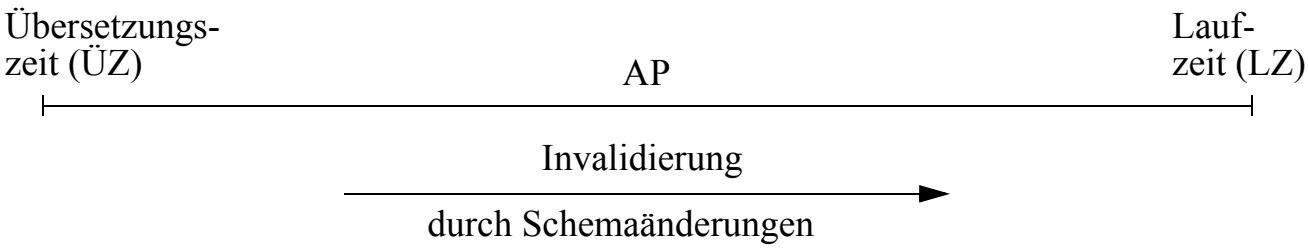
SYSTAB:
Tabellenbeschreibungen: Pers, . . .

SYSATTR:
Attributbeschreibungen: Pnr, Name, Gehalt, . . .

SYSINDEX:
I_{Pers}(Beruf), . . .

SYSAUTH:
Nutzungsrechte
SYSINT/RULES:
Integritätsbedingungen, Zusicherungen, . . .

- **Zeitpunkt des Bindens**



Übersetzungskosten:

- unerheblich für Antwortzeit (AZ)

Zugriffe (zur LZ):

- effizient
- datenabhängig!

Ausgleich gesucht!

Interpretation:

- erheblich für AZ

Zugriffe (zur LZ):

- teuer
- datenunabhängig!

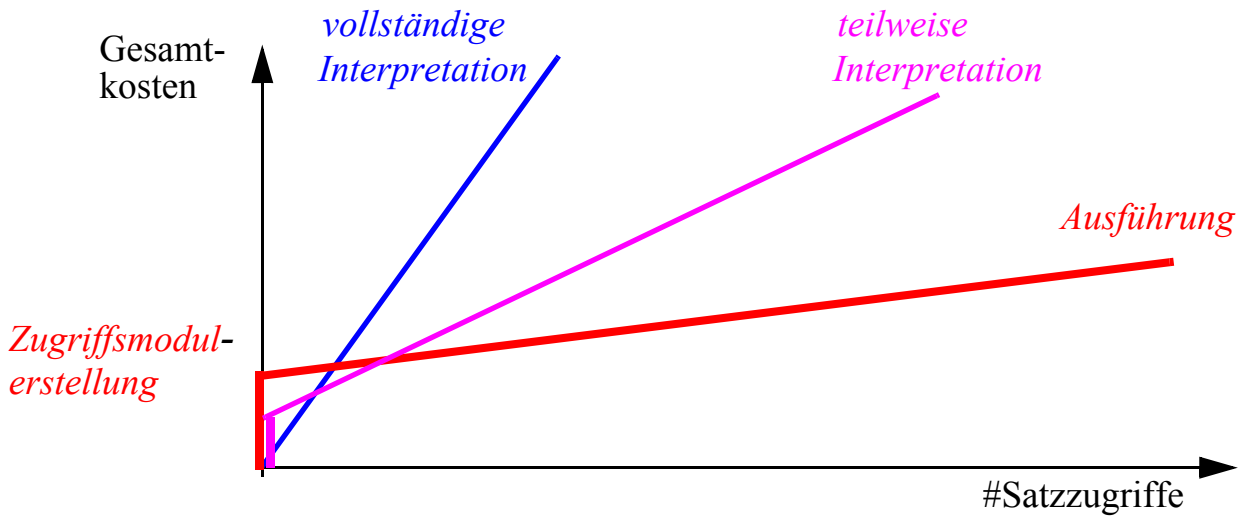
Übersetzung vs. Interpretation (2)

- **Maximale Vorbereitung einer DB-Anweisung**
  - aufwendige Optimierung und Erstellung eines Zugriffsmoduls
  - maximale Auswirkungen von Schemaänderungen, welche die DB-Anweisung betreffen
  - Änderungen des DB-Zustandes nach der Übersetzung werden nicht berücksichtigt (neue Zugriffspfade, geänderte Statistiken etc.)

  ➥ Invalidierung des Zugriffsmoduls und erneute Erstellung

- **Mischformen**
  - bestimmte Abhängigkeiten und Bindungen werden vermieden
  - jedoch: Invalidierungsmechanismus prinzipiell erforderlich

- **Interpretation einer DB-Anweisung**
  - Interpreter wertet Anweisung (als Zeichenfolge) zur Laufzeit aus
  - Aktueller DB-Zustand wird automatisch berücksichtigt
  - sehr hohe Ausführungskosten bei Programmschleifen sowie durch häufige Katalogzugriffe
  - interessant vor allem für Ad-hoc-Anfragen bzw. dynamisches SQL



Wirtssprachen-Einbettung und Übersetzung

- **Selbständige DB-Sprachen:** Deskriptive Anweisungen und mengenorientierte Zugriffe sind die natürliche Form des Einsatzes

- **Weiterverarbeitung der Ergebnisse verlangt Einbettung der DB-Anweisungen in eine Wirtssprache**

  ➥ Bei deskriptiven Sprachen mit mengenorientiertem Datenzugriff ergibt sich eine Fehlanpassung bei der Kopplung mit satzorientierter Verarbeitung ("impedance mismatch")

- **Prinzipielle Möglichkeiten**

  - Direkte Einbettung
    Dabei wird syntaktisch keine Unterscheidung zwischen Programm- und DB-Anweisungen gemacht. Eine DB-Anweisung wird als Zeichenkette A ins AP integriert.
  - Aufruftechnik
    Eine DB-Anweisung wird durch expliziten Funktionsaufruf an das Laufzeit-system des DBS übergeben (z. B. CALL DBS ('A'))

- **Mögliche Vorgehensweisen bei C' bzw. PC:**

  - A wird als Ergebnis von Übersetzung und Optimierung in eine interne Form überführt, die eine direkte Ausführung zur Laufzeit gestattet, d. h., die Bindung von A an DB-interne Namen und Zugriffspfade erfolgt zum Übersetzungszeitpunkt.
  - A wird als aktueller Parameter eines (internen) CALL-Aufrufs abgelegt, seine weitere Behandlung erfolgt erst zur Laufzeit. A kann entweder übersetzt oder interpretiert werden.

  ➥ Höherer konzeptioneller Abstand als bei direkter Einbettung

Formen der Wirtssprachen-Einbettung

1. Anweisung A eingebettet in AP (Spracherweiterung)

Übersetzung von AP mit

- modifiziertem Wirtssprachen-Compiler

$$C' : AP \rightarrow MC$$

- Pre-Compiler

$$PC : AP \rightarrow AP'$$
$$+\ C : AP' \rightarrow MC$$

Bindung von A

- zur ÜZ (Optimierung)
- zur LZ (interner CALL-Aufruf)

2. CALL DBS ('A')

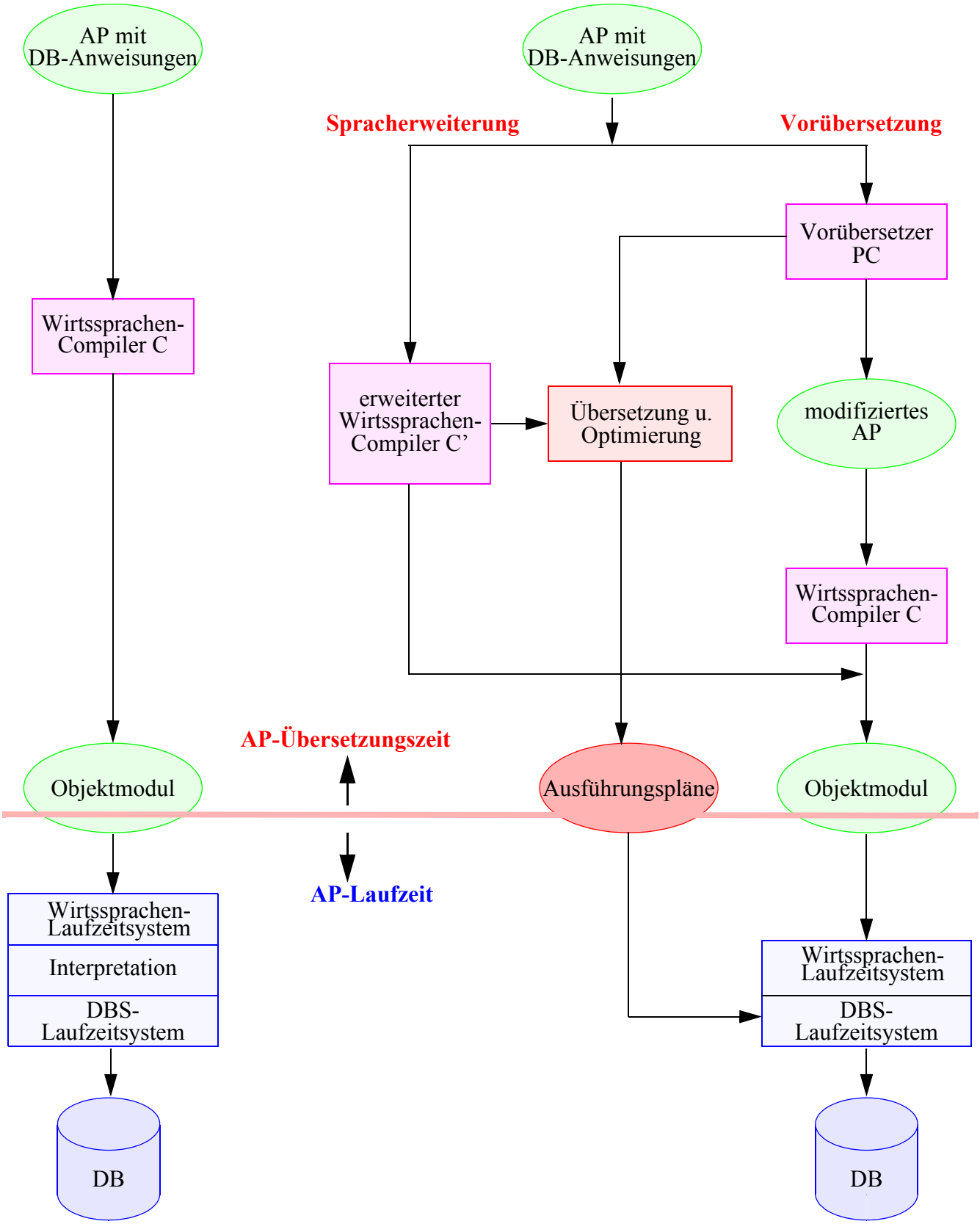
Übersetzung von AP mit

C: AP → MC

Bindung von 'A' : zur LZ

Interpretation: Übersetzung + Bindung zur Laufzeit

Übersetzungstechniken für AP mit eingebetteten DB-Anweisungen



Interpretationsansatz für Aufruftechnik (CALL)

Kompilationsansatz für Sprach-erweiterung bzw. Vorübersetzung

Formen der Wirtssprachen-Einbettung (2)

- **Datenstrukturen und Operatoren in Wirtssprache integriert**
  - Subschema in UWA (CODASYL)
  - DDL/DML

  FIND X-RECORD USING Y, Z

- **Datenstrukturen in Wirtssprache integriert**
  - Subschema in UWA

  CALL "DML" USING FUNKTIONSNAME, FUNKTIONSWAHL,
  ZUSATZWAHL, BENUTZERINFO, ...

- **Operatoren in Wirtssprache integriert**

  Bsp.: SQL

  ```
  DECLARE C1 CURSOR FOR
    SELECT    PNR, PNAME
    FROM      PERS
    WHERE     BERUF= :Z AND ANR = 'K55'

  . . .

  FETCH C1 INTO :X, :Y
  ```

- **Keine Integration**

  CALL ADABAS
  (CONTROL_BLOCK, FOBU, REBU, SEBU, VABU, ISNBU)

Klassifikation der DB-Schnittstelle

- Die Datenstrukturen, auf die das AP zugreift, sind als Subschema (oder Sicht) explizit im AP deklariert, d. h., das AP besitzt einen statisch zugeordneten Bereich für die Datenstrukturen der DB (UWA = User Working Area)
- Im AP ist kein Speicherplatz für DB-seitige Datenstrukturen reserviert. DB-Werte werden explizit an normale Programmvariablenübergeben.

		Operatoren in Wirtssprache integriert	
		ja	nein
Datenstrukturen im AP deklariert	ja	direkte Einbettung Übergabe in statisch zugeordneten Datenstrukturen B-Typ =	CALL-Technik Übergabe in statisch zugeordneten Datenstrukturen B-Typ =
	Nein	direkte Einbettung Übergabe an Programm-Variable B-Typ =	CALL-Technik Übergabe an Programm-Variable/Pufferbereiche B-Typ =

Bindungstyp: B-Type (Datenstrukturen/Operatoren)
C = Compilezeitbindung L = Langzeitbindung

Einbettung einer mengenorientierten Schnittstelle

- Anweisung zur **Spezifikation der gesuchten Tupelmenge** (Qualifikationsoperator)

```
SELECT   PNR, PNAME, GEHALT/12
FROM     PERS
WHERE    BERUF='Operateur'
         AND PROV>GEHALT
```

```
DECLARE C1 CURSOR FOR
SELECT   PNR, PNAME, GEHALT/12
FROM     PERS
WHERE    BERUF= :W
         AND PROV>GEHALT
```

- Anweisung zur **sukzessiven Bereitstellung der qualifizierten Tupel** (Abholoperator)

```
OPEN C1;                  → Bindung von W, z.B. 'Operateur',
                            Aktivierung des Cursors

FETCH C1 INTO :X, :Y, :Z; → Abholen eines Tupels

CLOSE C1;                 → Deaktivierung des Cursors
```

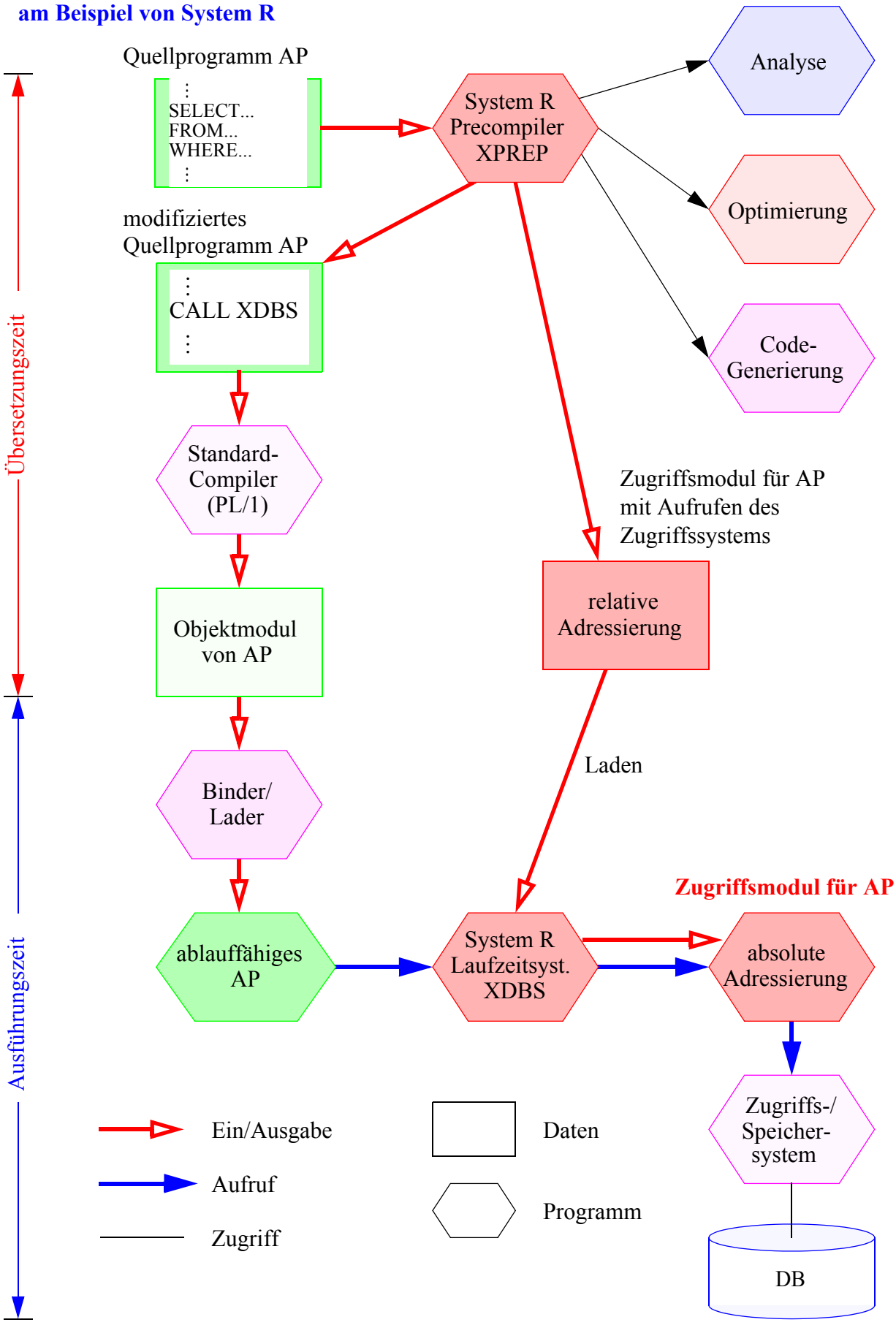
- Mögliche Lösung**: Ersetzung durch Pre-Compiler**

```
DECLARE C1 ...   → Kommentar

OPEN C1          → DCL T(3) POINTER;
                   T(1) = ADDR(W)
                   CALL XDBS (ZM1, 2, OPEN, ADDR(T))

FETCH C1 INTO ...→ T(1) = ADDR(X);
                   T(2) = ADDR(Y);
                   T(3) = ADDR(Z);
                   CALL XDBS (ZM1, 2, FETCH, ADDR(T));
```

Vorbereitung und Ausführung von DB-Anweisungen



Anfrageoptimierung*

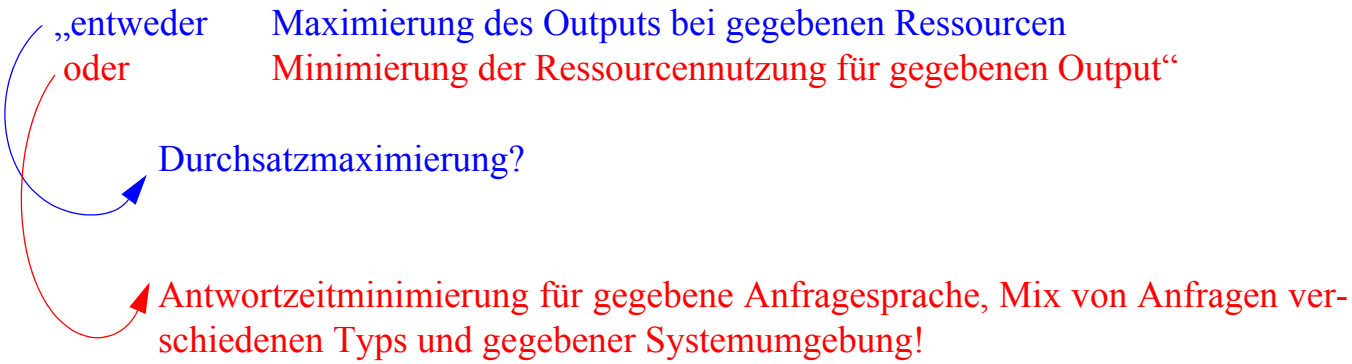
- **Von der Anfrage (Was?) zur Auswertung (Wie?)**

  ➥ **Ziel: kostengünstiger Auswertungsweg**

- **Einsatz einer großen Anzahl von Techniken und Strategien**
  - logische Transformation von Anfragen
  - Auswahl von Zugriffspfaden
  - optimierte Speicherung von Daten auf Externspeichern

- **Schlüsselproblem**
  - genaue Optimierung ist im allgemeinen „nicht berechenbar“
  - Fehlen von genauer statistischer Information
  - breiter Einsatz von Heuristiken (Daumenregeln)

- **Optimierungsziel**



* Jarke, M., Koch, J.: Query Optimization in Database Systems, in: ACM Comp. Surveys 16:2, 1984, pp. 111-152

Anfrageoptimierung (2)

- **Welche Kosten sind zu berücksichtigen?**

  - Kommunikationskosten
    (# der Nachrichten, Menge der zu übertragenden Daten)
    ➥ verteilte DBS!

  - Berechnungskosten (CPU-Kosten, Pfadlängen)

  - E/A-Kosten (# der physischen Referenzen)

  - Speicherungskosten (temporäre Speicherbelegung im DB-Puffer und auf Externspeichern)

  ➥ Kostenarten sind nicht unabhängig voneinander

  ➥ in zentralisierten DBS oft
  „gewichtete Funktion von Berechnungs- und E/A-Kosten“

- **Wie wird am besten vorgegangen?**

  Schritt 1: Finde nach Übersetzung geeignete Interndarstellung für die Anfrage (Anfragegraph)

  Schritt 2: Wende die logische Restrukturierung auf den Anfragegraph an

  Schritt 3: Bilde die restrukturierte Anfrage auf alternative Folgen von Planoperatoren (Transformation) ab
  (➥ Mengen von Ausführungsplänen)

  Schritt 4: Berechne Kostenvoranschläge für jeden Ausführungsplan und wähle den billigsten aus

Interndarstellung einer Anfrage

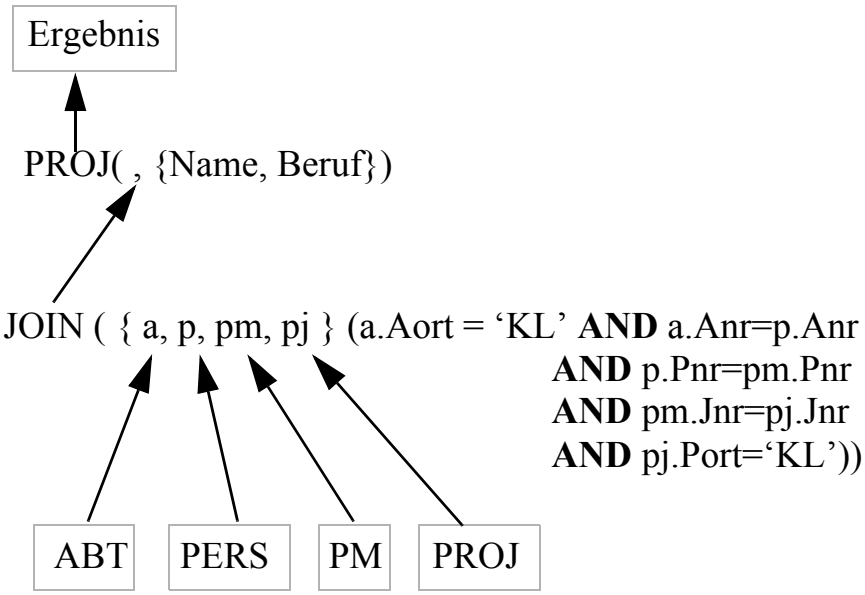
- **Klassen von Darstellungsschemata**
  - lineare oder auch matrixförmige Interndarstellung
    Relationenalgebra
    Relationenkalkül
  - strukturierte Interndarstellung
    Zerlegungsbaum
    Objektgraph
    Operatorgraph

- **Beispiel**

Finde Name und Beruf von Angestellten, die Projekte in 'KL' durchführen und deren zugehörige Abteilung sich ebenfalls in 'KL' befindet"

SELECT Name, Beruf
FROM ABT a, PERS p, PM pm, PROJ pj
WHERE a.Anr = p.Anr **AND** a.Aort = 'KL' **AND**
p.Pnr = pm.Pnr **AND**
pm.Jnr = pj.Jnr **AND** pj.Port = 'KL';

Operatorgraph

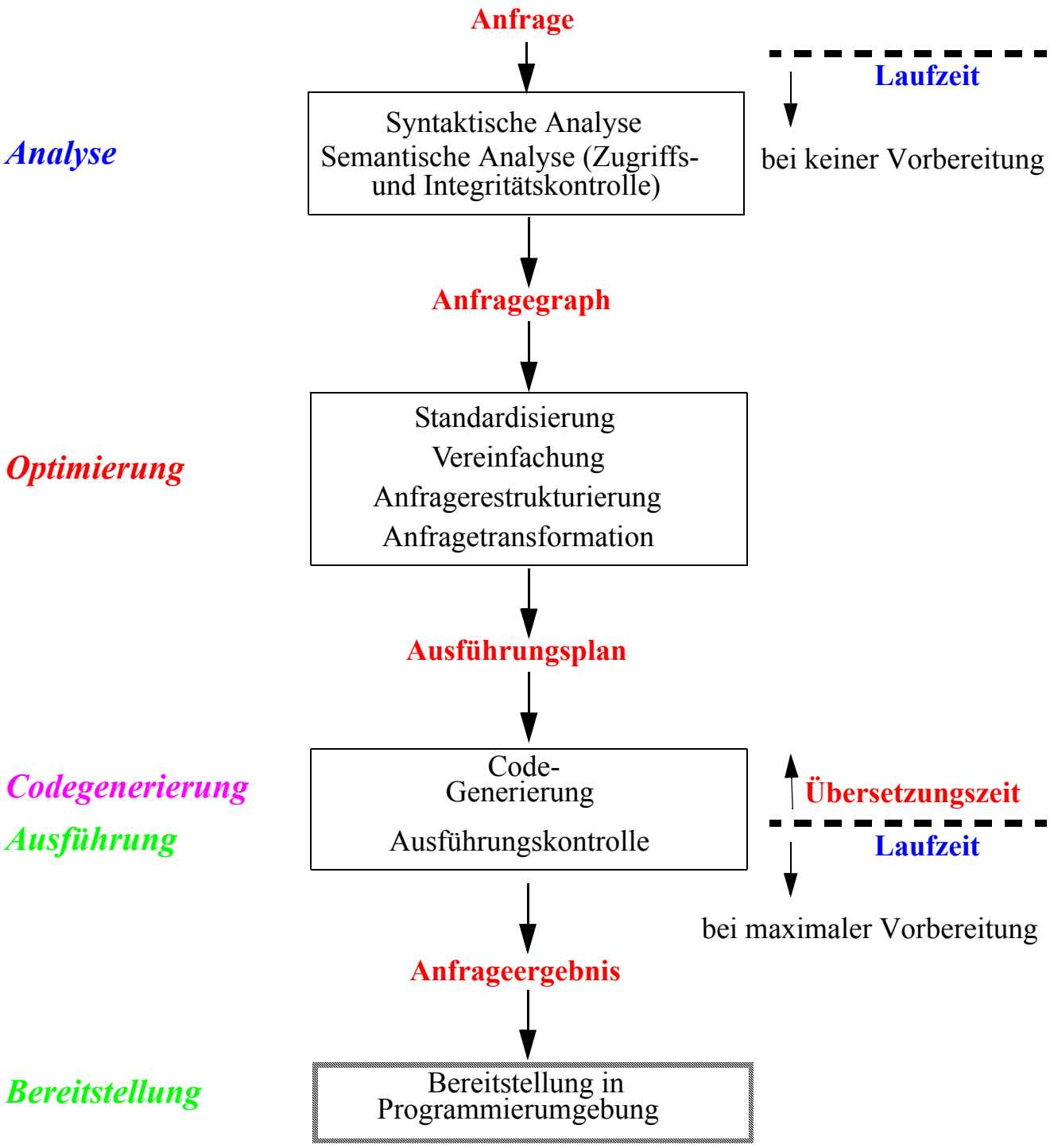


Anfrageverarbeitung – Kostenaspekte

Wie teilen sich die Kosten der Transaktionsverarbeitung auf?

DB-System : Kommunikationssystem : Anwendung

Wann fallen Kosten für die Anfrageverarbeitung an?



Standardisierung einer Anfrage

- **Standardisierung**
  - Wahl einer Normalform

    z.B. konjunktive Normalform

    (A11 OR ... OR A1n) AND ... AND (Am1 OR ... OR Amn)
  - Verschiebung von Quantoren

- **Umformungsregeln für Boolesche Ausdrücke**

Kommutativregeln

A	**OR**	B	⇔	B	**OR**	A
A	**AND**	B	⇔	B	**AND**	A

Assoziativregeln

(A	**OR**	B)	**OR**	C	⇔	A	**OR**	(B	**OR**	C)
(A	**AND**	B)	**AND**	C	⇔	A	**AND**	(B	**AND**	C)

Distribtivregeln

A	**OR**	(B	**AND**	C)	⇔	(A	**OR**	B)	**AND**	(A	**OR**	C)
A	**AND**	(B	**OR**	C)	⇔	(A	**AND**	B)	**OR**	(A	**AND**	C)

De Morgan'sche Regeln

NOT (A	**AND**	B)	⇔	**NOT** (A)	**OR**	**NOT** (B)
NOT (A	**OR**	B)	⇔	**NOT** (A)	**AND**	**NOT** (B)

Doppelnegationsregel

NOT (**NOT** (A))	⇔	A

- **Idempotenzregeln für Boolesche Ausdrücke**

A	**OR**	A	⇔	A
A	**AND**	A	⇔	A
A	**OR**	**NOT** (A)	⇔	TRUE
A	**AND**	**NOT** (A)	⇔	FALSE
A	**AND**	(A **OR** B)	⇔	A
A	**OR**	(A **AND** B)	⇔	A
A	**OR**	**FALSE**	⇔	A
A	**OR**	**TRUE**	⇔	TRUE
A	**AND**	**FALSE**	⇔	FALSE

Vereinfachung einer Anfrage

- **Äquivalente Ausdrücke**
  - können einen unterschiedlichen Grad an Redundanz besitzen
  - Behandlung/Eliminierung gemeinsamer Teilausdrücke

$$\begin{aligned} & (A_1 = a_{11} \text{ OR } A_1 = a_{12}) \\ \text{AND} \quad & (A_1 = a_{12} \text{ OR } A_1 = a_{11}) \end{aligned}$$

  - Vereinfachung von Ausdrücken, die an “leere Tabellen” gebunden sind

- **Konstanten-Propagierung**

$$A \text{ op } B \text{ AND } B = \text{const.}$$

➥ $A \text{ op const.}$

- **Nicht-erfüllbare Ausdrücke**

$$A \geq B \text{ AND } B > C \text{ AND } C \geq A$$

➥ $A > A \rightarrow \text{false}$

- **Nutzung von Integritätsbedingungen (IB)**
  - IB sind wahr für alle Tupel der betreffenden Tabelle
  - A ist Primärschlüssel: $\pi_A \rightarrow$ keine Duplikateliminierung erforderlich
  - Regel: FAM-STAND = ‘verh.’ AND STEUERKLASSE ≥ 3

➥ Ausdruck: (FAM-STAND = ‘verh.’ AND STEUERKLASSE = 1) → false

- **Verbesserung der Auswertbarkeit**
  - Hinzufügen einer IB zur WHERE-Bedingung verändert den Wahrheitswert eines Auswahlausdrucks nicht

➥ Einsatz zur verbesserten Auswertung (knowledge-based query processing)

  - einfachere Auswertungsstruktur, jedoch effiziente Heuristiken benötigt

Anfragerestrukturierung

- **Wichtigste Regeln für Restrukturierung und Transformation**
  - Frühzeitige Ausführung von Selektion (σ) und Projektion (π) ohne Duplikateliminierung
  - Unäre Operatorfolgen (wie σ und π) zu einer Operation zusammenzufassen
  - Gleiche Teile im AG nur einmal auswerten
  - Binäre Operatorfolgen (wie $\cap$, $\cup$, $-$, $\times$, $|\times|$): Zwischenergebnisse minimieren

  ➥ selektive Operationen (σ, π) vor konstruktiven Operationen ($\times$, $|\times|$)

- **Zusammenfassung von Operationsfolgen**

  R1: $\pi_{An}(...\pi_{A2}(\pi_{A1}(Tab))...) \Leftrightarrow \pi_{An}(Tab)$

  R2: $\sigma_{pn}(...\sigma_{p2}(\sigma_{p1}(Tab))...)...) \Leftrightarrow \sigma_{p1 AND\ p2\ ...\ AND\ pn}(Tab)$

- **Restrukturierungsalgorithmus**

(1) Zerlege komplexe Verbundprädikate so, daß sie binären Verbunden zugeordnet werden können (Bilden von binären Verbunden).

(2) Teile Selektionen mit mehreren Prädikatstermen in separate Selektionen mit jeweils einem Prädikatsterm auf.

(3) Führe Selektionen so früh wie möglich aus, d. h., schiebe Selektionen hinunter zu den Blättern des AG (selection push-down).

(4) Fasse einfache Selektionen zusammen, so daß aufeinanderfolgende Selektionen (derselben Tabelle) zu einer verknüpft werden.

(5) Führe Projektionen ohne Duplikateliminierung so früh wie möglich aus, d. h., schiebe sie soweit wie möglich zu den Blättern des AG hinunter (projection push-down).

(6) Fasse einfache Projektionen (einer Tabelle) zu einer Operation zusammen.

Anfragetransformation

- **Aufgabe**
  Zusammenfassung von logischen Operatoren (Ein- und Zwei-Variablen-Ausdrücke) und ihre Ersetzung durch **Planoperatoren**

- **Typische Planoperatoren in relationalen Systemen**

  - **auf einer Tabelle:**
    Selektion, Projektion, Sortierung, Aggregation, Änderungsoperationen (IUD) und ACCESS zum Zugriff auf Basistabellen
    +
    Erweiterungen: Rekursion, Gruppierung, . . .

  - **auf zwei Tabellen:**
    Verbund- und Mengen-Operationen, Kartesisches Produkt.

- **Anpassungen im AG zum effektiven Einsatz von Planoperatoren**

(1) Gruppierung von direkt benachbarten Operatoren;

z. B. lassen sich durch einen speziellen Planoperator ersetzen:
Verbund (oder Kartesisches Produkt) mit Selektionen und/oder Projektionen auf den beteiligten Tabellen.

(2) *Bestimmung der Verknüpfungsreihenfolge bei binären Operationen*;

dabei sollen die minimalen Kosten für die Operationsfolge erzielt werden.
Als Heuristik ist dazu die Größe der Zwischenergebnisse zu minimieren,
d. h., die kleinsten (Zwischen-)Tabellen sind immer zuerst zu verknüpfen.

(3) *Erkennung gemeinsamer Teilbäume*,

die dann nur jeweils einmal zu berechnen sind. Allerdings steht dieser Einsparung die Zwischenspeicherung der Ergebnistabelle gegenüber.

Bewertung von Ausführungsplänen – Grundsätzliche Probleme

- **Anfrageoptimierung beruht i.a. auf zwei „fatalen“ Annahmen**
  1. Alle Datenelemente und alle Attributwerte sind gleichverteilt
  2. Suchprädikate in Anfragen sind unabhängig

  ➥ beide Annahmen sind falsch (im allgemeinen Fall)

- **Beispiel**

  (GEHALT ≥ ‘100K’) AND (ALTER BETWEEN 20 AND 30)

  ➥ lineare Interpolation, Multiplikation von Wahrscheinlichkeiten

  Obwohl die Kostenabschätzungen meist falsch sind . . .

- Lösung ?
  - Verbesserung der Statistiken/Heuristiken
  - Histogramme, Sampling

- **Stichproben**
  - Geschwindigkeit: Große Datenmengen, komplexe Algorithmen
  - Beispiel: Abschätzung „Umsatz in Europa“ bei TPC-H:
    1%: 8.46 Mio. in 4 sec.
    10%: 8.51 Mio. in 52 sec.
    100%: 8.54 Mio. in 200 sec.

- **Einsatzgebiete des Sampling**
  - näherungsweise Anfragebeantwortung, Antwortzeitschätzung
  - Anfrageoptimierung
  - Lastbalancierung
  - Data Mining, interaktives Anfragedesign

Erstellung und Auswahl von Ausführungsplänen

- **Eingabe:**
  - optimierter Anfragegraph (AG)
  - existierende Speicherungsstrukturen und Zugriffspfade
  - Kostenmodell

- **Ausgabe: optimaler Ausführungsplan (oder wenigstens „gut“)**

- **Vorgehensweise:**
  1. Generiere alle „vernünftigen“ logischen Ausführungspläne zur Auswertung der Anfrage
  2. Vervollständige die Ausführungspläne durch Einzelheiten der physischen Datenrepräsentation (Sortierreihenfolge, Zugriffspfadmerkmale, statistische Information)
  3. Wähle den billigsten Ausführungsplan gemäß dem vorgegebenen Kostenmodell aus

  ➥ Alternative Ausführungspläne für einen AG entstehen vor allem dadurch, daß für jeden Planoperator verschiedene Methoden (Implementierungen) vorliegen, und daß Operationsreihenfolgen (z. B. bei Mehrfachverbunden) variiert werden können. So bilden sich bei komplexen Anfragen sehr große Suchräume mit Alternativen (z. B. 10^{70} mögliche Ausführungspläne bei einer Anfrage mit 15 Verbunden).

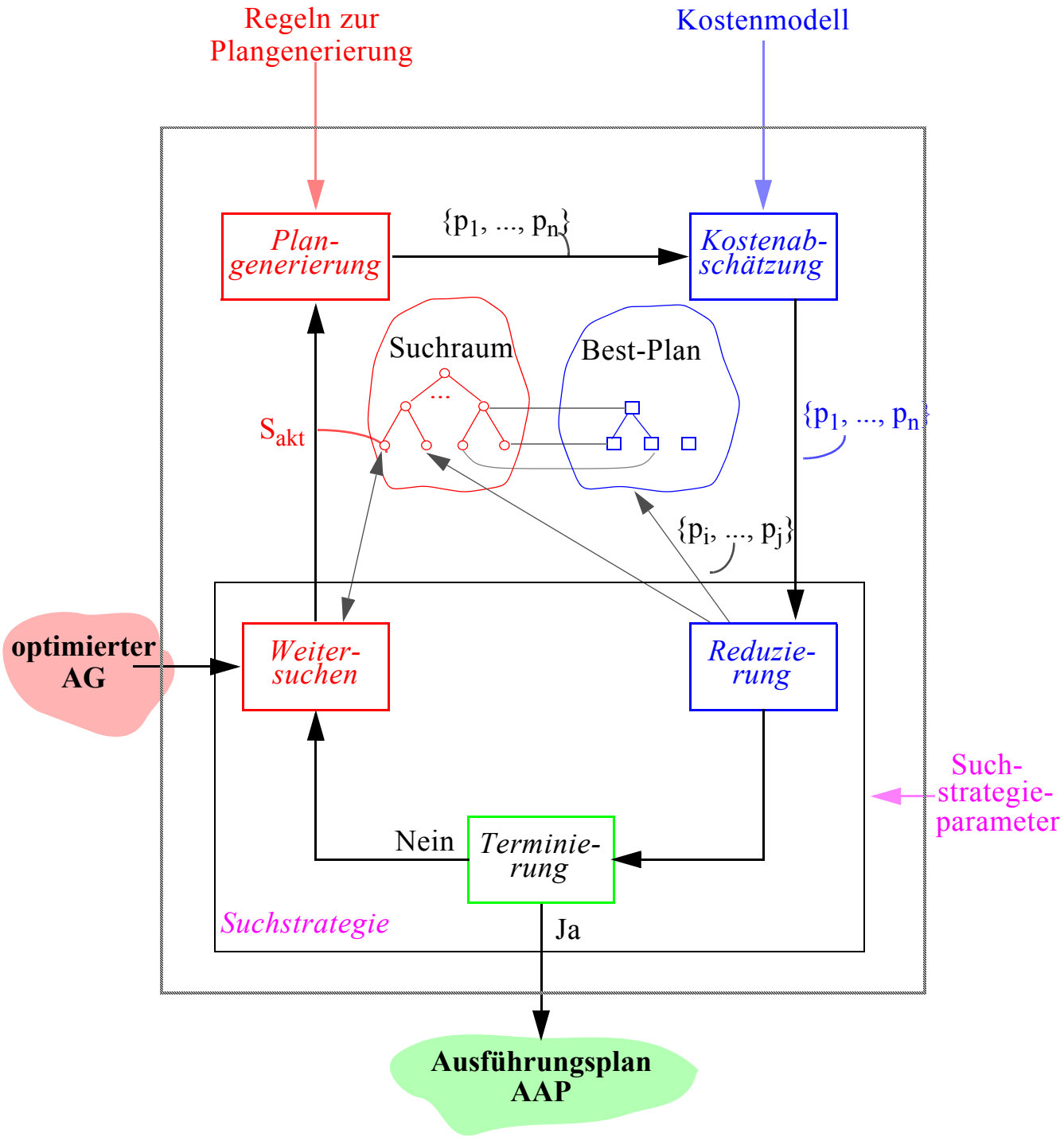
- **Generierung durch Anfrageoptimierer**

  ➥ kleine Menge der Pläne, die den optimalen Plan enthält

  ➥ Einschränkung durch Heuristiken
  - hierarchische Generierung basierend auf dem Schachtelungskonzept von SQL
  - Zerlegung in eine Menge von Teilanfragen mit höchstens Zwei-Variablen-Ausdrücken

Erstellung und Auswahl von Ausführungsplänen (2)*

- **Zusammenspiel der Komponenten**



AAP: Anfrageausführungsplan (engl. QEP: Query Evaluation Plan)

* Mitschang, B.: Anfrageverarbeitung in Datenbanksystemen: Entwurfs- und Implementierungskonzepte, Reihe Datenbanksysteme, Vieweg, 1995

Erstellung und Auswahl von Ausführungsplänen (3)

- **Plangenerierung** soll
  - immer und möglichst schnell den „optimalen“ Plan finden
  - mit einer möglichst kleinen Anzahl generierter Pläne auskommen

- **Suchstrategien**
  - voll-enumerativ
  - beschränkt-enumerativ
  - zufallsgesteuert

  ➥ **Reduzierung:** Bestimmte Suchpfade zur Erstellung von AAPs werden nicht weiter verfolgt

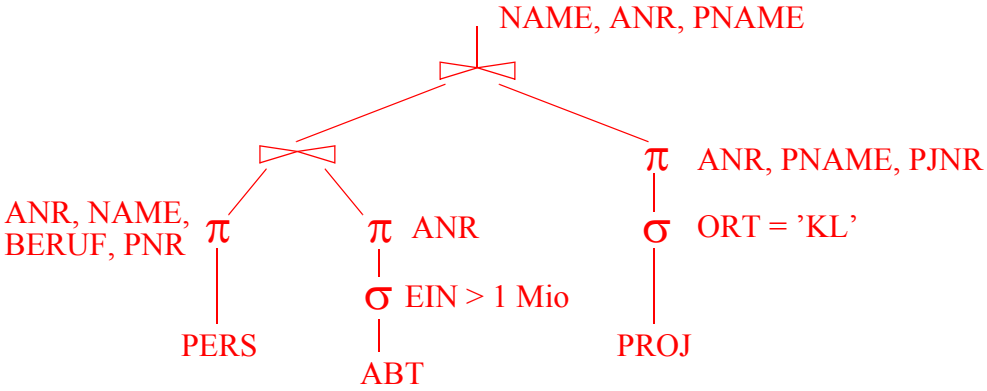
- **Kostenabschätzung**
  - verlangt hinreichend genaues Kostenmodell
  - wird bei allen Suchverfahren inkrementell durchgeführt

- **Problemdarstellung – Beispiel**

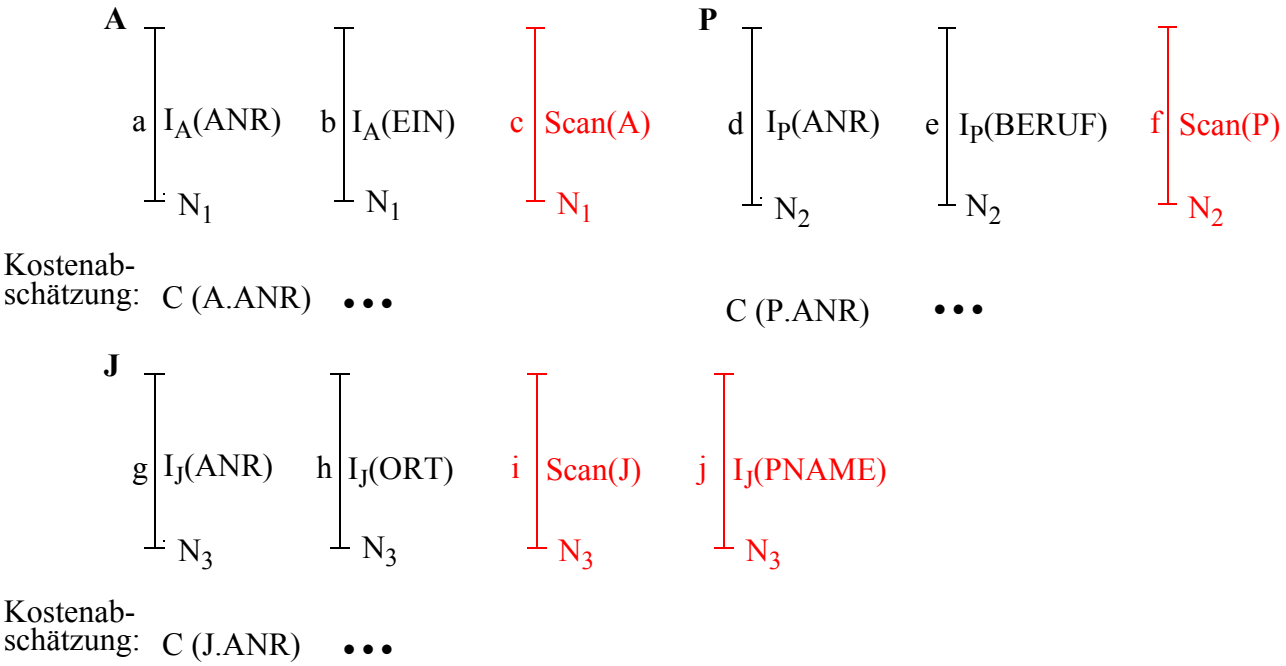
SQL:

```
SELECT   P.NAME, P.BERUF, J.PNAME
FROM     PERS P, ABT A, PROJ J
WHERE    A.EIN > 1000000 AND J.ORT = 'KL'
  AND    A.ANR = P.ANR AND A.ANR = J.ANR;
```

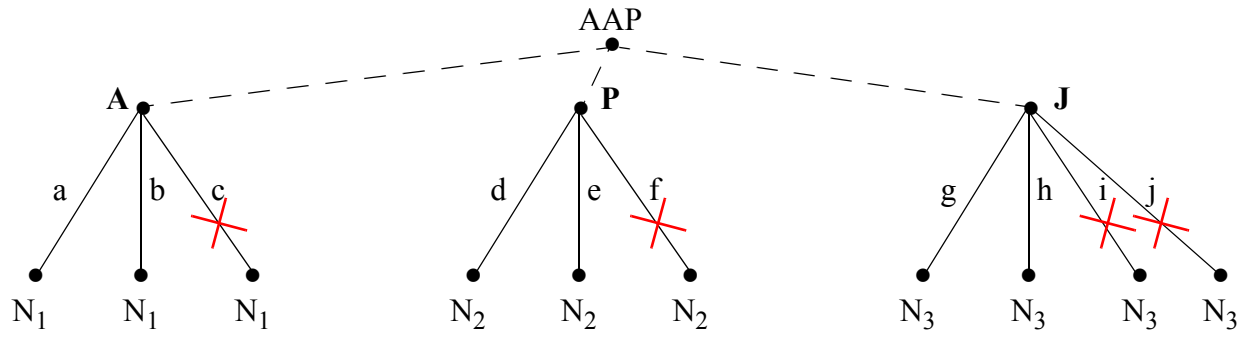
Zugehöriger Anfragegraph



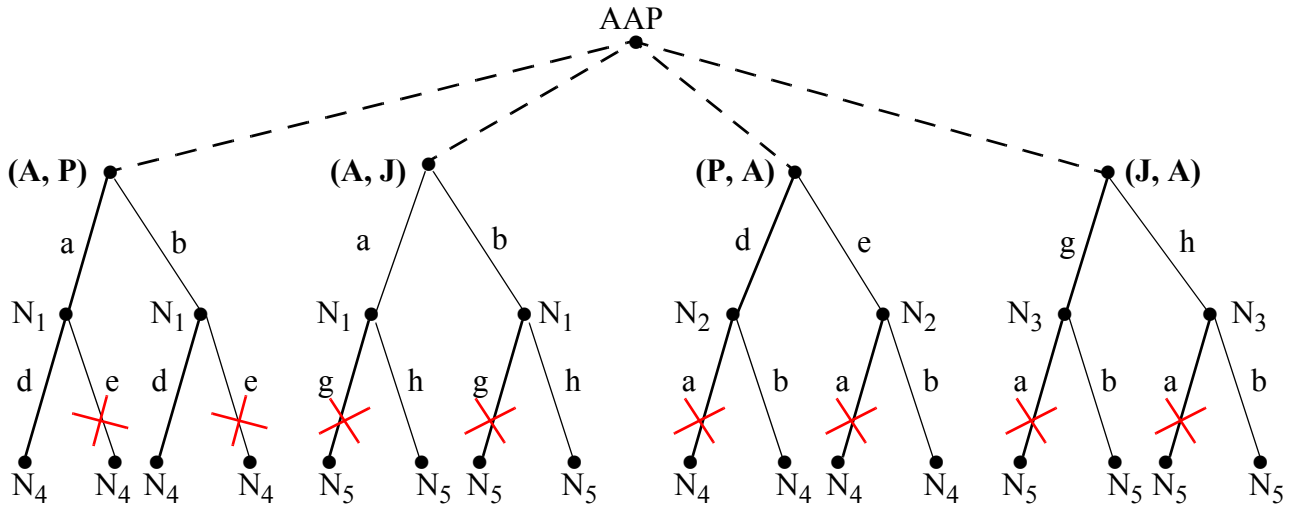
Erstellung und Auswahl von Ausführungsplänen (4)



a) mögliche Zugriffspfade für die einzelnen Tabellen



b) Lösungsbaum für einzelne Tabellen:
Reduzierung durch Abschneiden von Teilbäumen



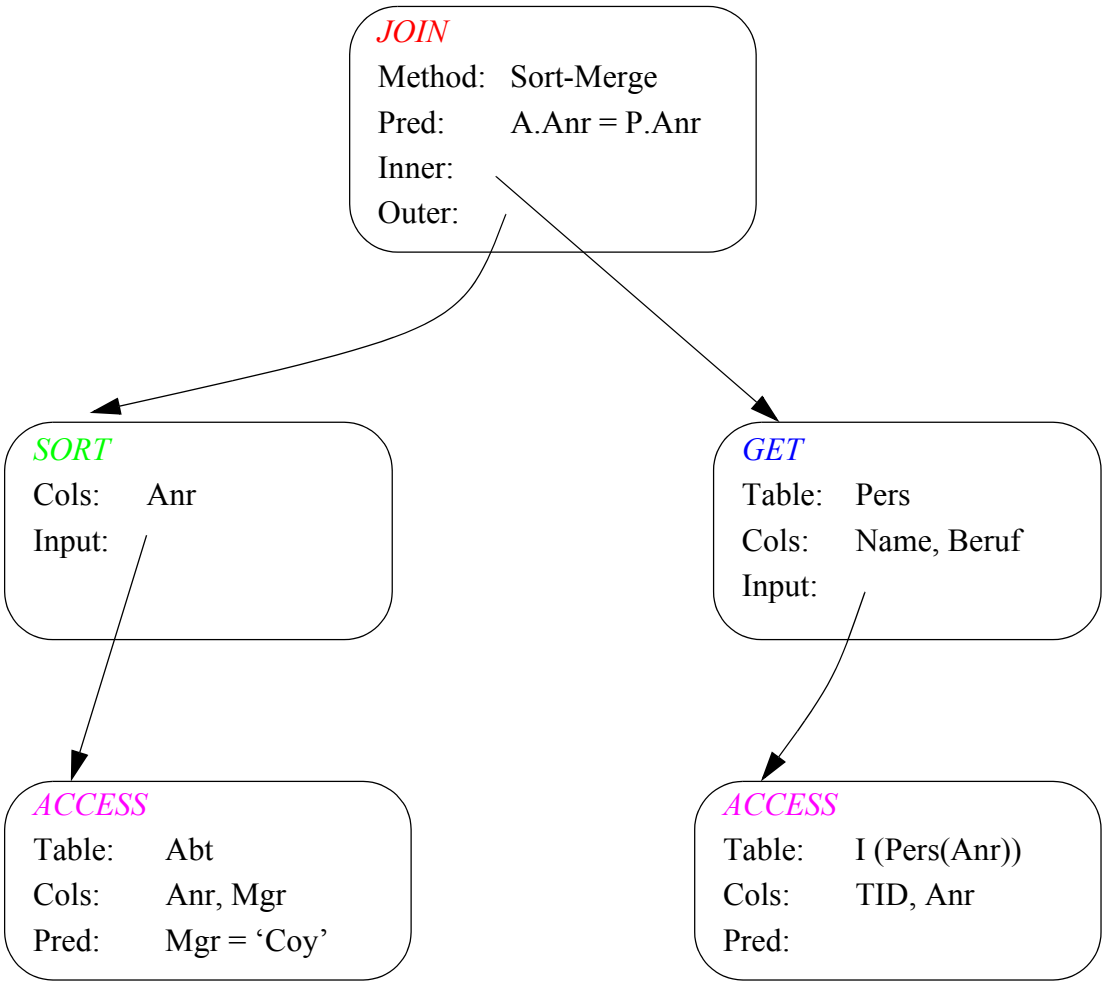
c) Erweiterter Lösungsbaum für den Nested-Loop-Verbund mit der zweiten Tabelle
Kostenabschätzung pro Pfad: z. B. durch C(C(A.ANR) + C(P.ANR) + Verbundkosten)

Ausführungsplan – Beispiel

- **Anfrage-Beispiel**

```
SQL:    SELECT    Name, Beruf
        FROM      Pers P, Abt A
        WHERE     P.Anr = A.Anr
        AND       A.Mgr = 'Coy'
```

- **Ein möglicher Operatorbaum**



- **Dazugehöriges „Programm“**

```
JOIN (Sort-Merge, A.Anr = P.Anr,
        SORT (ACCESS (Abt, {Anr, Mgr}, {Mgr = 'Coy'}), Anr),
        GET (ACCESS (I (Pers(Anr)), {TID, Anr}, ∅),
                Pers, {Name, Beruf} , ∅)).
```

Kostenmodell – Probleme

- **Optimiereraufgabe**
  - erstellt Kostenvoranschlag für jeden „aussichtsreichen“ Ausführungsplan
  - Einsatz einer gewichteten Kostenformel:

    C = #physischer Seitenzugriffe + W * (#Aufrufe des Zugriffssystems)

    - angestrebt: gewichtetes Maß für E/A- und CPU-Auslastung
    - W ist das Verhältnis des Aufwandes von ZS-Aufruf zu Seitenzugriff

- **Immerwährendes Problem**
  - 1985 war SQL nicht standardisiert
  - SQL2 und SQL3 wesentlich komplexer
    - BDTs
    - Typ- und Tabellenhierarchien
    - Rekursion, Constraints, Trigger, …

- **Übersetzung und Optimierung**
  - **kostenbasierter Optimierer**
    - Histogramme
    - aber: BDTs benötigen ihr eigenes Kostenmodell
    - “Optimizing the XXX optimizer”
  - Dynamische QEPs
    - alternative Pläne je nach Betriebsmittelverfügbarkeit, …
    - “reduce the braking distance”
  - Verführung zum Glücksspiel

Kostenmodell – statistische Werte

- **Statistische Größen für Segmente:**

  M_S Anzahl der Datenseiten des Segmentes S

  L_S Anzahl der leeren Seiten in S

- **Statistische Größen für Tabellen:**

  N_R Anzahl der Tupel der Tabelle R (Card(R))

  $T_{R,S}$ Anzahl der Seiten in S mit Tupel von R

  C_R Clusterfaktor (Anzahl Tupel pro Seite)

- **Statistische Größen pro Index** I auf Attributen A einer Tabelle R:

  j_I Anzahl der Attributwerte / Schlüsselwerte im Index
  (=Card (π_A(R))

  B_I Anzahl der Blattseiten (B*-Baum)

  . . .

  ➥ **Statistiken müssen im DB-Katalog gewartet werden**

- **Aktualisierung bei jeder Änderung sehr aufwendig**
  - zusätzliche Schreib- und Log-Operationen
  - DB-Katalog wird zum Sperr-Engpaß

- **Alternative:**
  - Initialisierung der statistischen Werte zum Lade- oder Generierungs-zeitpunkt von Tabellen und Indexstrukturen
  - periodische Neubestimmung der Statistiken durch eigenes Kommando/ Dienstprogramm (DB2: RUNSTATS)

Kostenmodell – Berechnungsgrundlagen

Mit Hilfe der statistischen Werte kann der Anfrageoptimierer jedem Verbundterm im Qualifikationsprädikat einen Selektivitätsfaktor ($0 \le SF \le 1$) zuordnen (erwarteter Anteil an Tupel, die das Prädikat erfüllen): Card ($\sigma_p(R)$) = SF(p) · Card (R)

- **Selektivitätsfaktor SF bei:**

$A_i = a_i$	SF =	$1/j_i$	wenn Index auf A_i
		$1/10$	sonst
$A_i = A_k$	SF =	$1 / \text{Max}(j_i, j_k)$	wenn Index auf A_i, A_k
		$1 / j_i$	wenn Index auf A_i
		$1 / j_k$	wenn Index auf A_k
		$1/10$	sonst
$A_i \ge a_i$ (oder $A_i > a_i$)	SF =	$(a_{max} - a_i) / (a_{max} - a_{min})$	wenn Index auf A_i und Wert interpolierbar
		$1/3$	sonst
A_i BETWEEN a_i AND a_k	SF =	$(a_k - a_i) / (a_{max} - a_{min})$	wenn Index auf A_i und Wert interpolierbar
		$1/4$	sonst
A_i IN $(a_1, a_2, ..., a_r)$	SF =	r / j_i	wenn Index auf A_i und SF < 0.5
		$1/2$	sonst

- **Berechnung von Ausdrücken**
  - SF (p(A) ∧ p(B)) = SF (p(A)) · SF (p(B))
  - SF (p(A) ∨ p(B)) = SF (p(A)) + SF (p(B)) - SF (p(A)) · SF (p(B))
  - SF (¬ p(A)) = 1 - SF (p(A))
- **Join-Selektivitätsfaktor (JSF)**
  - Card (R⋈S) = JSF ∗ Card(R) ∗ Card(S)
  - bei (N:1)-Verbunden (verlustfrei): Card (R⋈S) = Max(Card(R), Card(S))

Code-Erzeugung

- **Optimierter Anfragegraph**
  - Ergebnis der Optimierungsphase
  - Eingabe-Datenstruktur für Code-Generator

- **Nutzung der Operationen des Zugriffssystems**
  - direkte Operationen (z.B. INSERT <satz>)
  - Scan-Operationen (Beispiel SYSTEM R)
    - CALL RSS (OPEN, SCAN_STRUCTURE, RETURN_CODE)
    - CALL RSS (NEXT, SCAN_STRUCTURE, RETURN_CODE)
    - SCAN_STRUCTURE ist komplexe Datenstruktur zur Übergabe von Ein-/Ausgabewerten, Suchargumenten usw.

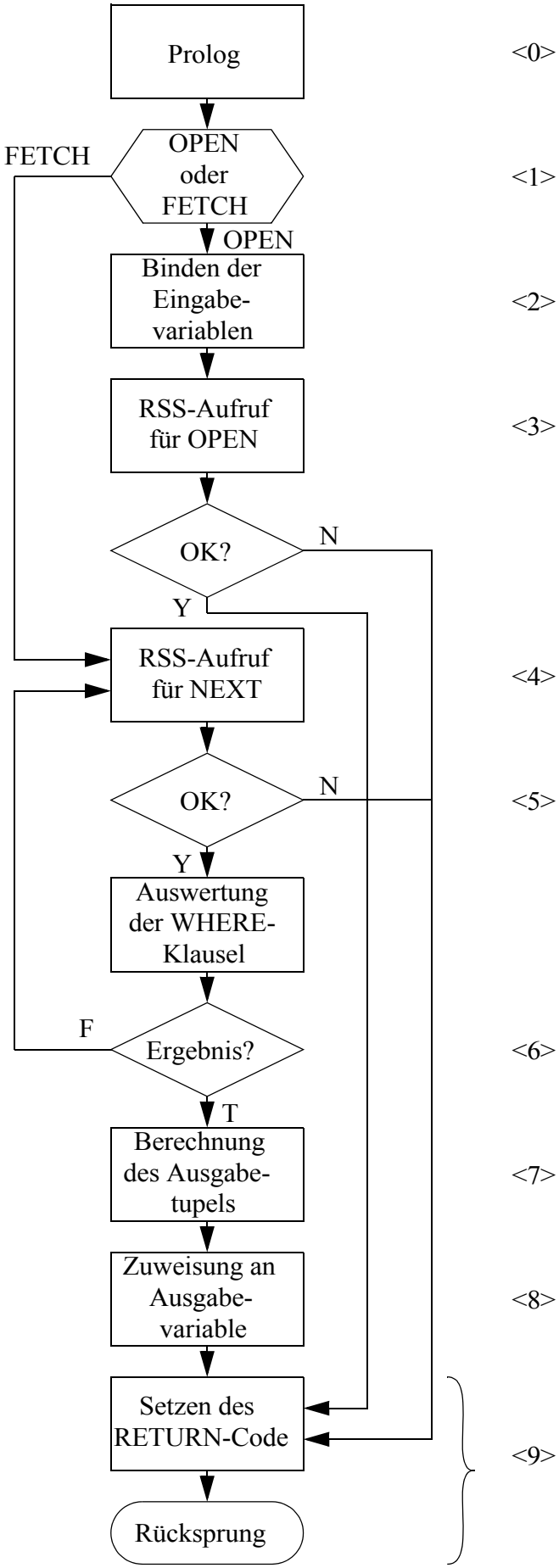
  ➥ **Diese Operationen werden bei der Code-Generierung als Primitive eingesetzt**

- **Klassifikation der SQL-Anweisungen**
  - jede Klasse wird durch Basisprozeß beschrieben (z.B. Auswahl einer Tupelmenge mit Hilfe eines Cursors)
  - Skelett eines Basisprozesses heißt Modell
  - Verarbeitungsschritt im Modell heißt Fragment (als Codefolge in einer Bibliothek abgelegt)

  ➥ Klassifikation erfolgt nach Art der Zugriffsaktionen

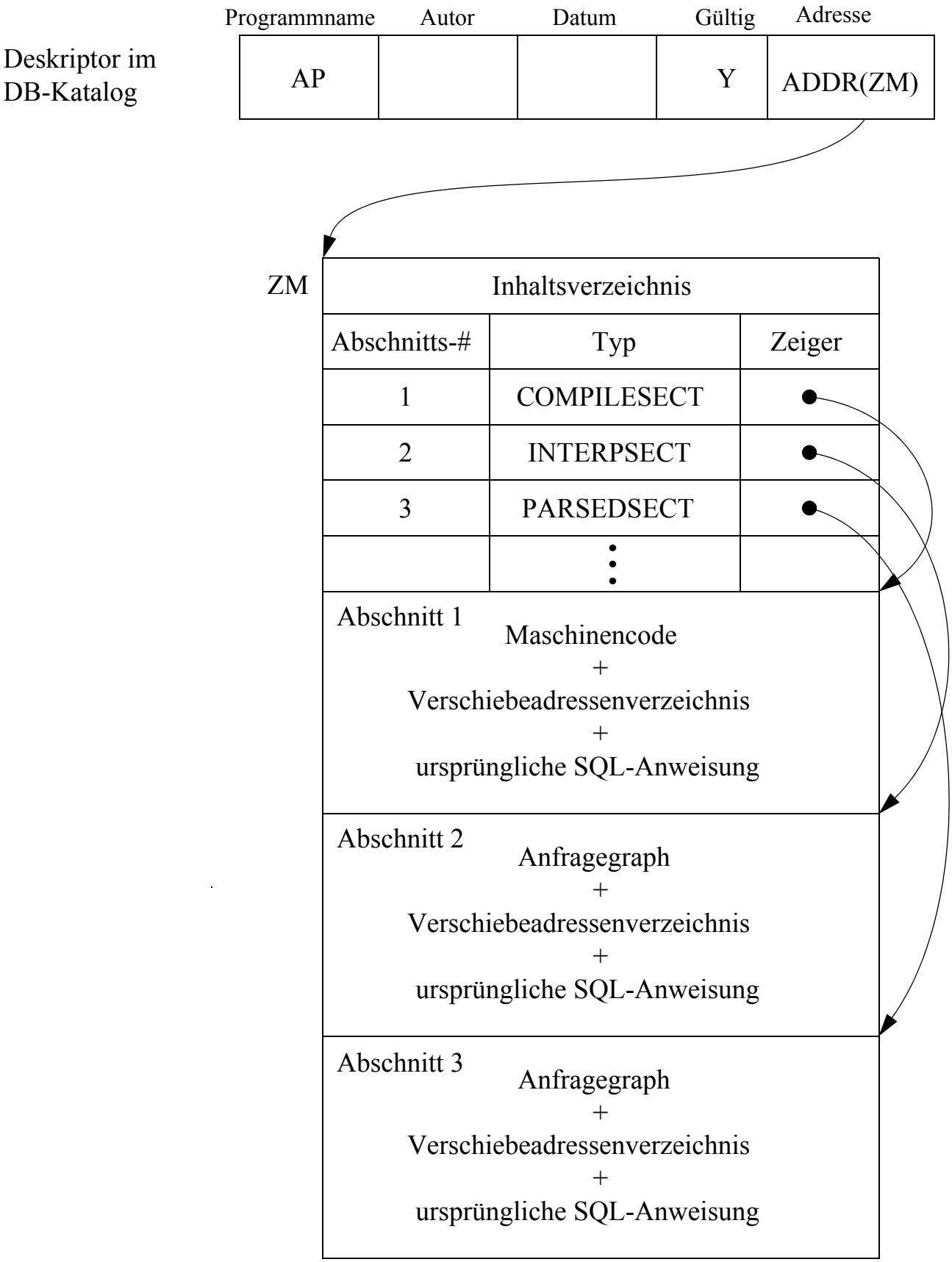
- **Bereitstellung von Modellen und Fragmenten**
  - 4 Modelle für einfache Fragen (Frageblöcke)
  - insgesamt 30 Modelle mit jeweils 5-10 Fragmenten (<100 Fragmente)

Flußdiagramm für einen Zugriffsmodul

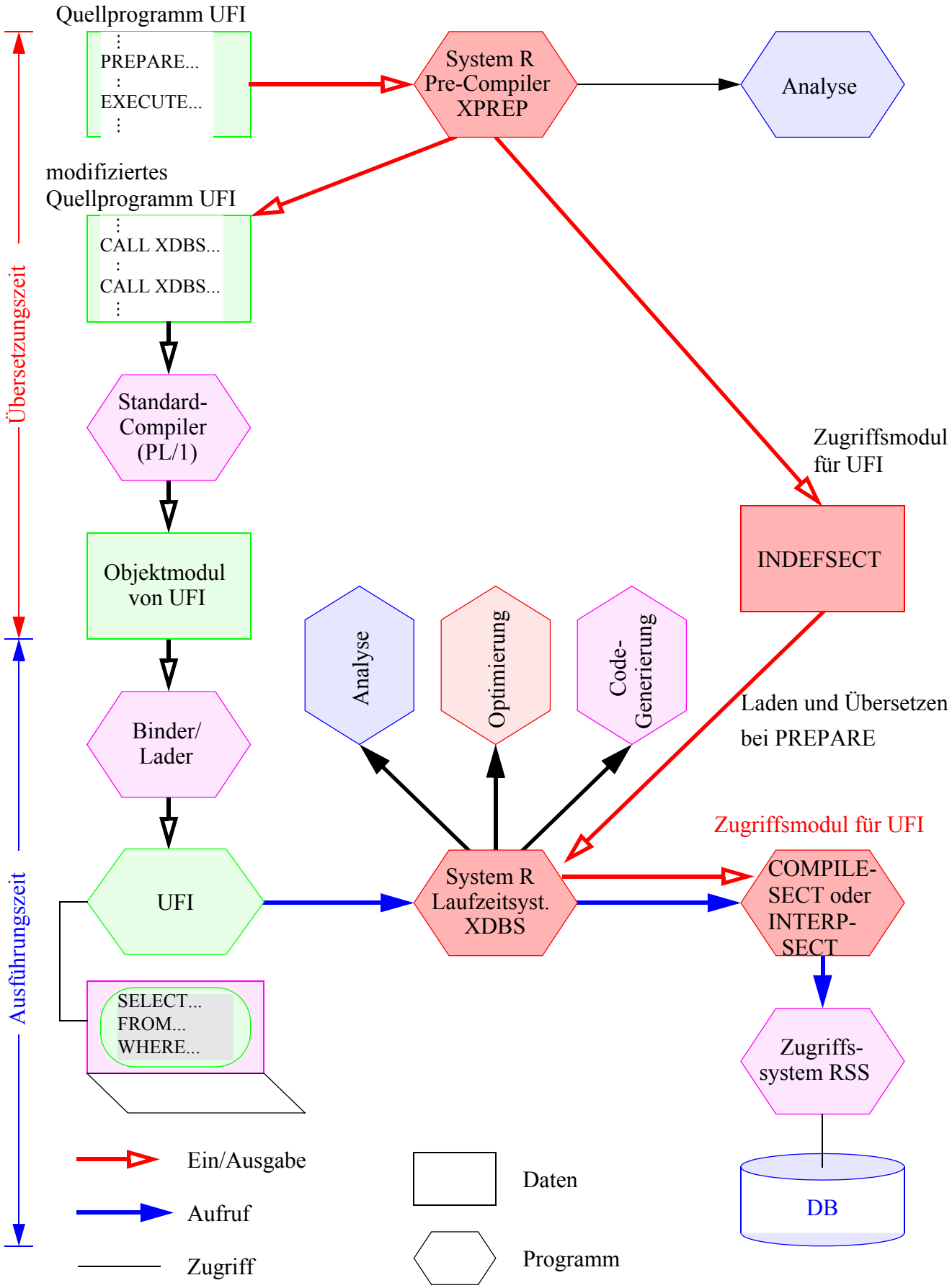


Modell für die Auswahl einer Tupelmenge mit Hilfe eines Cursors

Aufbau eines Zugriffsmoduls



Vorbereitung, Übersetzung und Ausführung von Ad-hoc-Anfragen



Spektrum der Bindezeiten in System R

Anweisungstyp	Abschnittstyp	Analyse	Optimierung	Code-Generierung	Ausführung
normale Operationen (Query, Insert, Delete, Update)	COMPILESECT	Übersetzungszeit			Laufzeit
nicht-optimierbare Operationen (Create/Drop Table, etc.)	INTERPSECT	Übersetzungszeit			Laufzeit
Operationen auf temporären Objekten	PARSEDSECT	Übersetzungszeit	Laufzeit		
Dynamisch definierte Anweisungen (Prepare, Execute)	INDEFSECT	Laufzeit			

Zusammenfassung

- **Interpretation einer DB-Anweisung**
  - Allgemeines Programm (Interpreter) akzeptiert Anweisungen der DB-Sprache als Eingabe und erzeugt mit Hilfe von Aufrufen des Zugriffssystems Ergebnis
  - hoher Aufwand zur Laufzeit (v.a. bei wiederholter Anweisungsausführung)

- **Übersetzung, Code-Erzeugung und Ausführung einer DB-Anweisung**
  - Für jede DB-Anweisung wird ein zugeschnittenes Programm erzeugt (Übersetzungszeit), das zur Laufzeit abgewickelt wird und dabei mit Hilfe von Aufrufen des Zugriffssystems das Ergebnis ableitet
  - Übersetzungsaufwand wird zur Laufzeit soweit wie möglich vermieden

- **Anfrageoptimierung: Kernproblem**
  der Übersetzung mengenorientierter DB-Sprachen
  - "fatale" Annahmen:
    - Gleichverteilung aller Attributwerte
    - Unabhängigkeit aller Attribute
  - Kostenvoranschläge für Ausführungspläne:
    - CPU-Zeit und E/A-Aufwand
    - Anzahl der Nachrichten und zu übertragende Datenvolumina (im verteilten Fall)
  - gute Heuristiken zur Auswahl von Ausführungsplänen sehr wichtig

- **Kostenmodell**
  - Minimierung der Kosten in Abhängigkeit des Systemzustandes
  - Problem: Aktualisierung der statistischen Kenngrößen