Grammatical Aspects for Language Descriptions

Andrey Breslav
abreslav@gmail.com

St. Petersburg SU ITMO / University of Tartu

Euroopa Liit
Euroopa Sotsiaalfond

Eesti tuleviku heaks

This work was partly supported by

DoRa

ELLU VIIB SIHTASUTUS
ARCHIMEDES

Outline

- Motivation
  - Cross-cutting concerns in grammars
- Approach
  - Grammatical aspects and how they work
- Examples (related to SQL)
  - Defining IDE components
  - Defining SQL dialects

Central statement

- There are cross-cutting concerns in DSL definitions
  - Many things are mixed together
- Separating them is beneficial for software quality
  - Better readability
  - Better maintainability

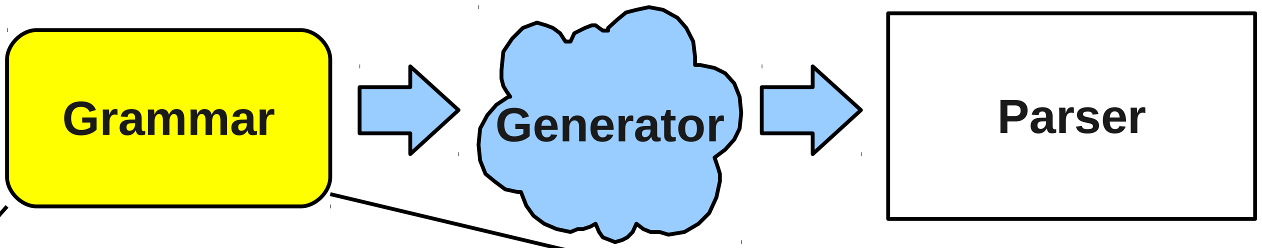
Cross-cutting concerns in programs

- Functionality **scattered** across many subsystems and **tangled** together
  - Business logic
  - Logging
  - Security
  - Component life-cycle
  - ...

Cross-cutting concerns in grammars

- Artefacts **scattered** across many rules and **tangled** together
  - Syntax descriptions
  - Semantic actions
  - IDE features
    - Highlighting specifications
    - Pretty-printing rules
    - ...
  - Language variations (dialects)

Example: Syntactic Concern



```
expr   : term ((PLUS | MINUS) term)* ;
term   : factor ((MULT | DIV) factor)* ;
factor : NUMBER ;
```

Example: Syntax & Actions (ANTLR)

```
expr returns [int result] :
    t=term {result = t;}
    ({int sign = 1;} (PLUS | MINUS {sign = -1;}) t=term {result += sign * t;})*;

term returns [int result] :
    f=factor {result = f;}
    ({boolean div = false;} (MULT | DIV {div = true;}) f=factor {
        if (div)
            result /= f;
        else
            result *= f;
     })*;

factor returns [int result] :
    n=NUMBER {result = Integer.parseInt(n);};
```

Separation of concerns for semantic actions

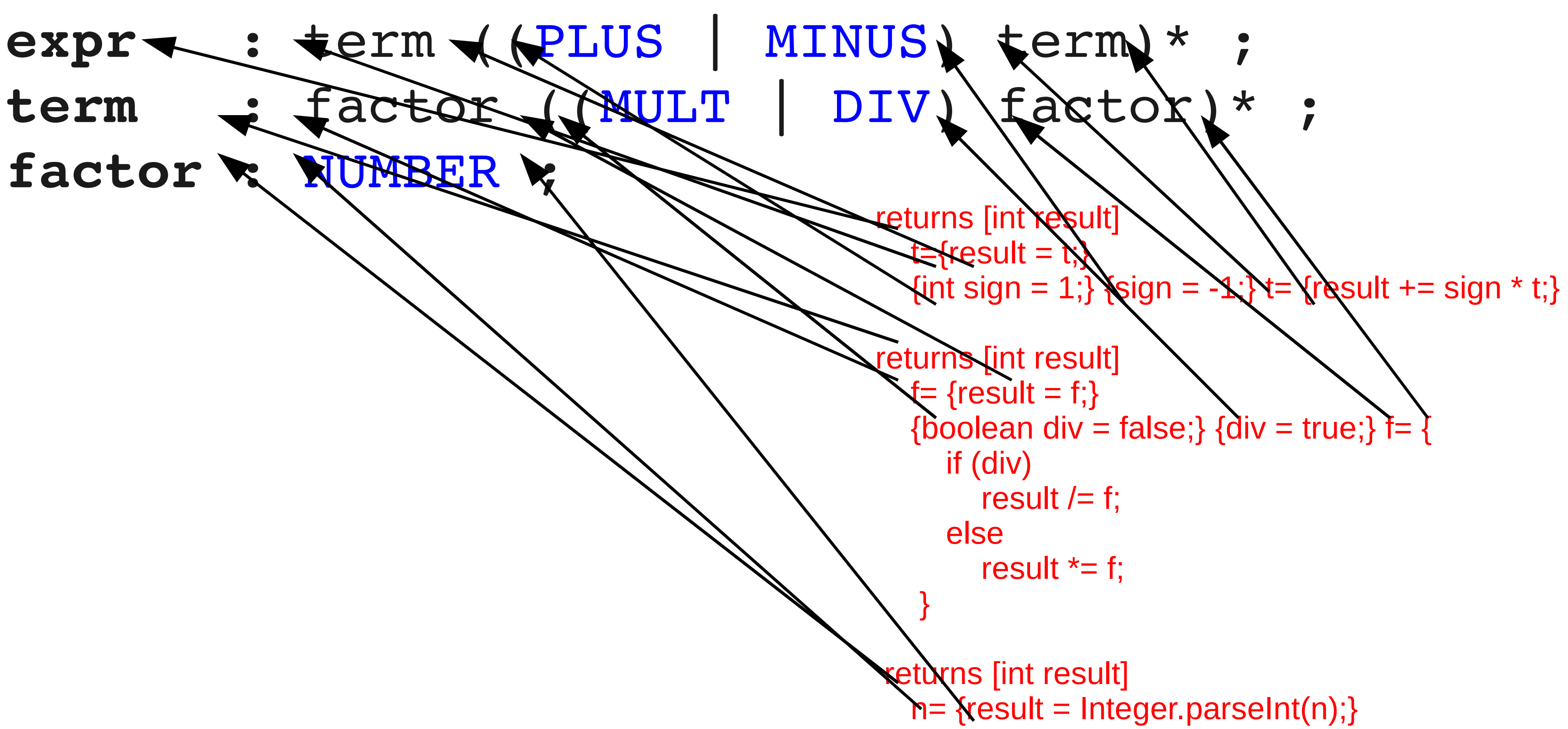
```
expr   : term ((PLUS | MINUS) term)* ;
term   : factor ((MULT | DIV) factor)* ;
factor : NUMBER ;
```

```
returns [int result]
    t={result = t;}
    {int sign = 1;} {sign = -1;} t= {result += sign * t;}

returns [int result]
    f= {result = f;}
    {boolean div = false;} {div = true;} f= {
        if (div)
            result /= f;
        else
            result *= f;
     }

 returns [int result]
    n= {result = Integer.parseInt(n);}
```

Separation of concerns: Attaching annotations **(Join Points)**



```
expr   : term ((PLUS | MINUS) term)* ;
term   : factor ((MULT | DIV) factor)* ;
factor : NUMBER ;
```

```
returns [int result]
   t={result = t;}
   {int sign = 1;} {sign = -1;} t= {result += sign * t;}

returns [int result]
   f= {result = f;}
   {boolean div = false;} {div = true;} f= {
      if (div)
         result /= f;
      else
         result *= f;
    }

returns [int result]
   n= {result = Integer.parseInt(n);}
```

AspectJ Aspects

```
pointcut getter() :
                  call(public int Example+.get*())
```

Advice:

```
after() : getter() {
   Log.write("A get method called");
}
```

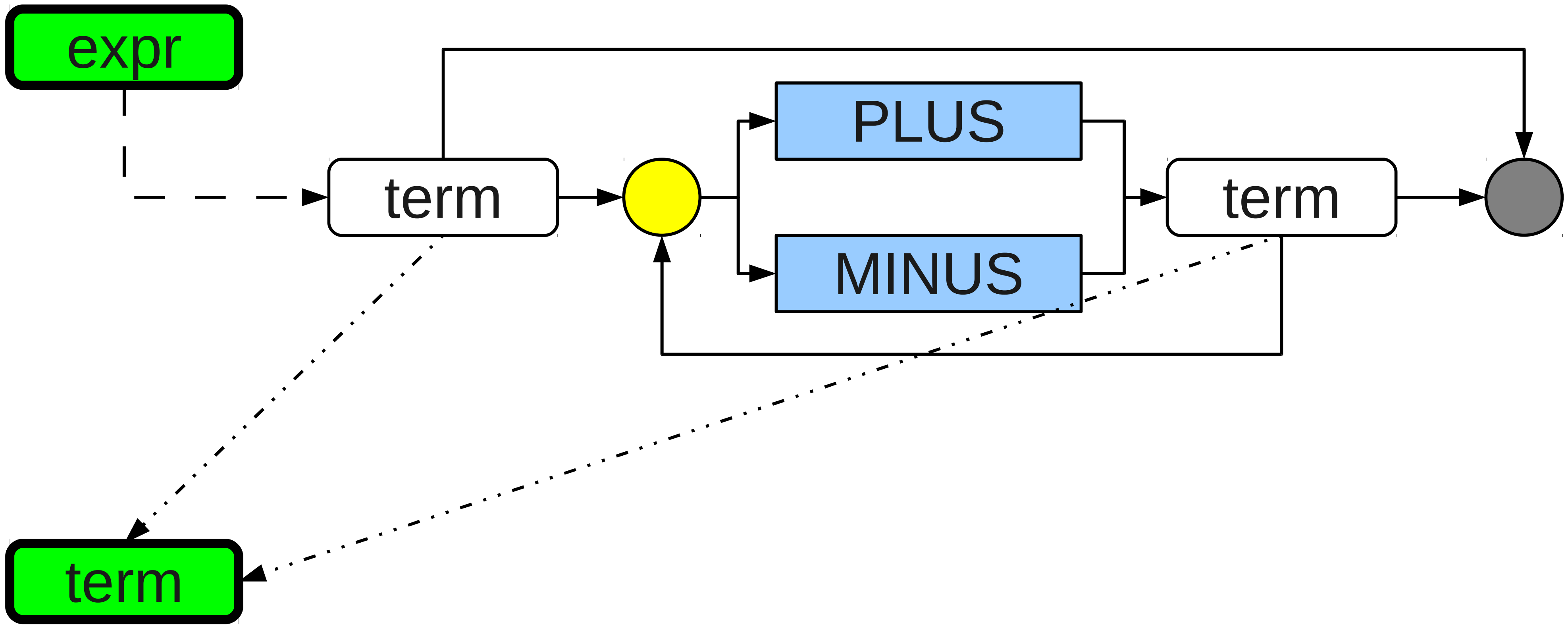
Example:

```
class Example {
   public int getFoo() { .. }
   public void doBar() {
      int x = getFoo(); // Matches the pointcut
      /*advice is woven here*/ }
```

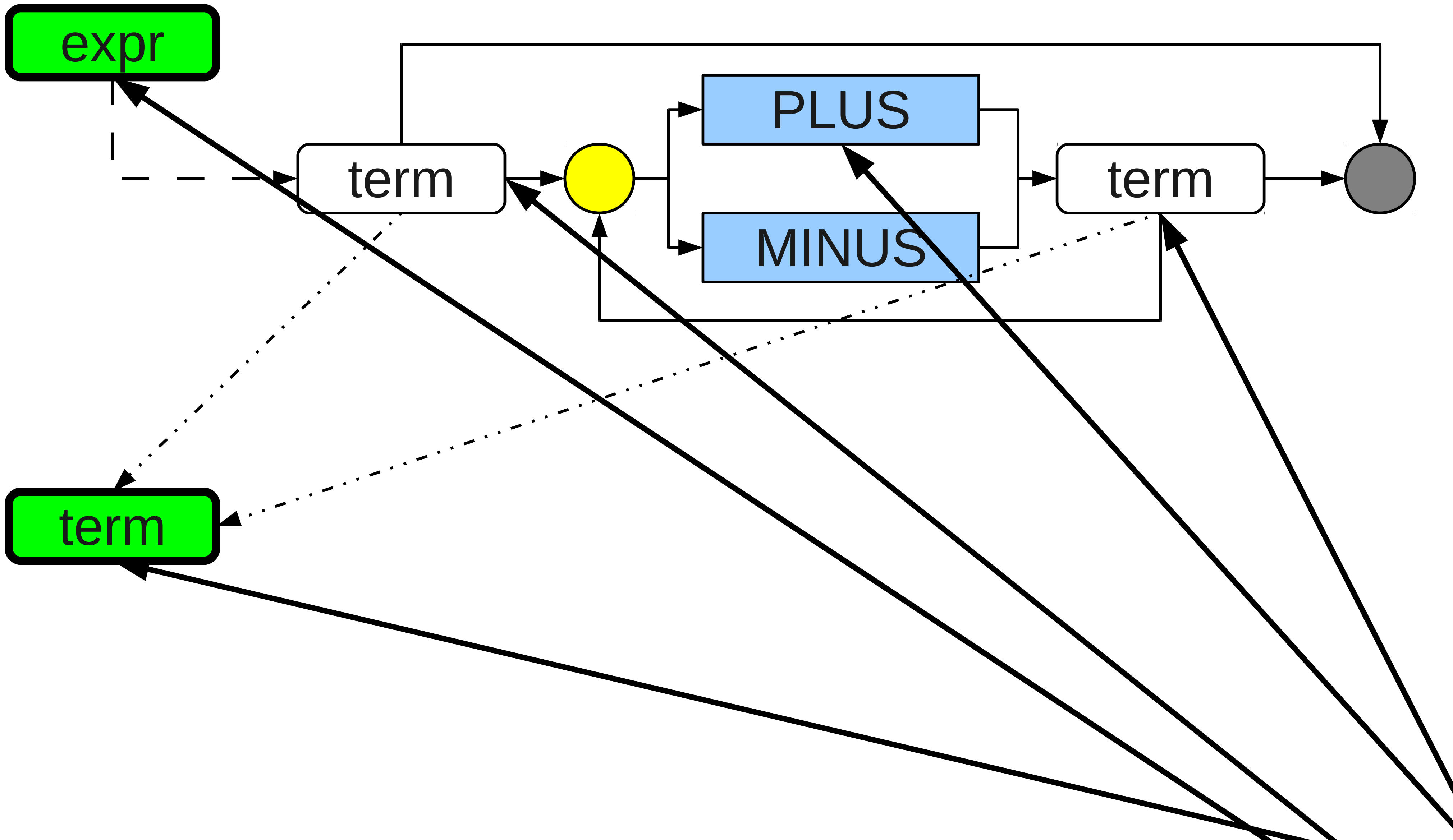
Grammatical Aspects

- Pointcuts
  - Patterns over grammatical structure
    - “productions starting with (foo)+”
- Advice
  - Annotations
    - processed by generators
  - Modifications to grammatical structure

Grammar is not a Text



Grammar is not a Text



Point-cut examples

- **`expr`** `: term ((PLUS | MINUS) term)* ;`
    - Exact match
- **`expr`** `:` **`{...}`** `;`
    - Production wildcard
- `# : term .. ;`
    - Symbol wildcard and sequence wildcard
- `# :` **`$t`**`=# ((PLUS | MINUS)` **`$t`**`)* ;`
    - Symbol wildcard and a variable

Advice examples: SQL Pretty-printer

- Aspect rule for metadata:

```
querySpecification : SELECT .. ?sl=selectList;
      @sl.before = {{ '\n' increaseIndent }}
      @sl.after = {{ '\n' }}
```

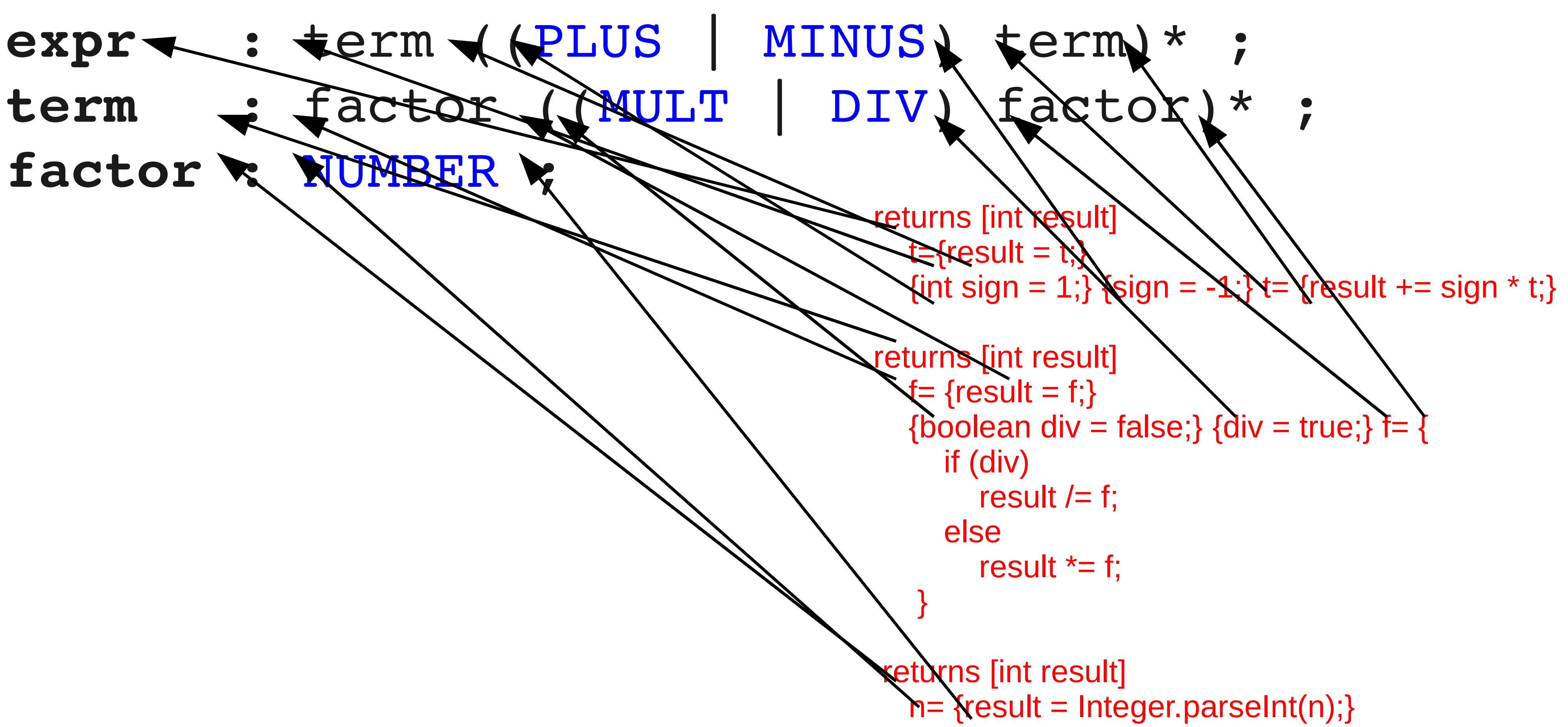
- Pretty-printed query:

SELECT DISTINCT

Authors.name **AS** name, title, year

FROM Books **LEFT JOIN** Authors **ON** (author = Authors.id)

Separation of concerns: Attaching annotations **(Join Points)**



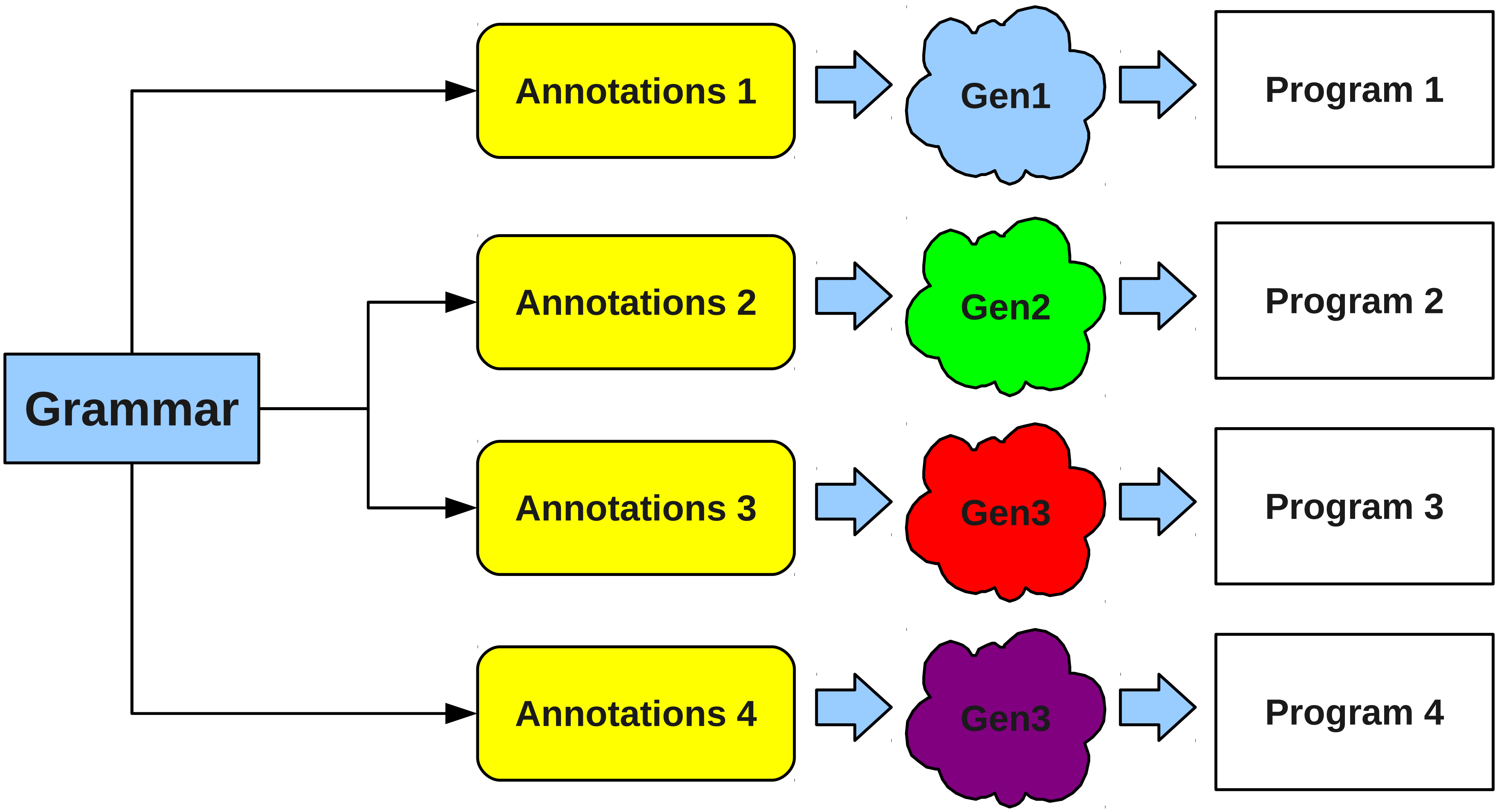
```
expr    : term ((PLUS | MINUS) term)* ;
term    : factor ((MULT | DIV) factor)* ;
factor  : NUMBER ;
```

```
returns [int result]
   t={result = t;}
   {int sign = 1;} {sign = -1;} t= {result += sign * t;}

returns [int result]
   f= {result = f;}
   {boolean div = false;} {div = true;} f= {
      if (div)
         result /= f;
      else
         result *= f;
    }

returns [int result]
   n= {result = Integer.parseInt(n);}
```

Useful features of metadata aspects



Advice examples: SQL Dialects

- Problem:
  - Describe how PostgreSQL synaxt differs from SQL92
  - Example: There can be ON-clause after DISTINCT
- Point-cut:
  - `setQuantifier : ?d=DISTINCT | ...;`
- Advice:
  - **after ?d** : (ON **'('** <List expression, ','> **')'**)?;

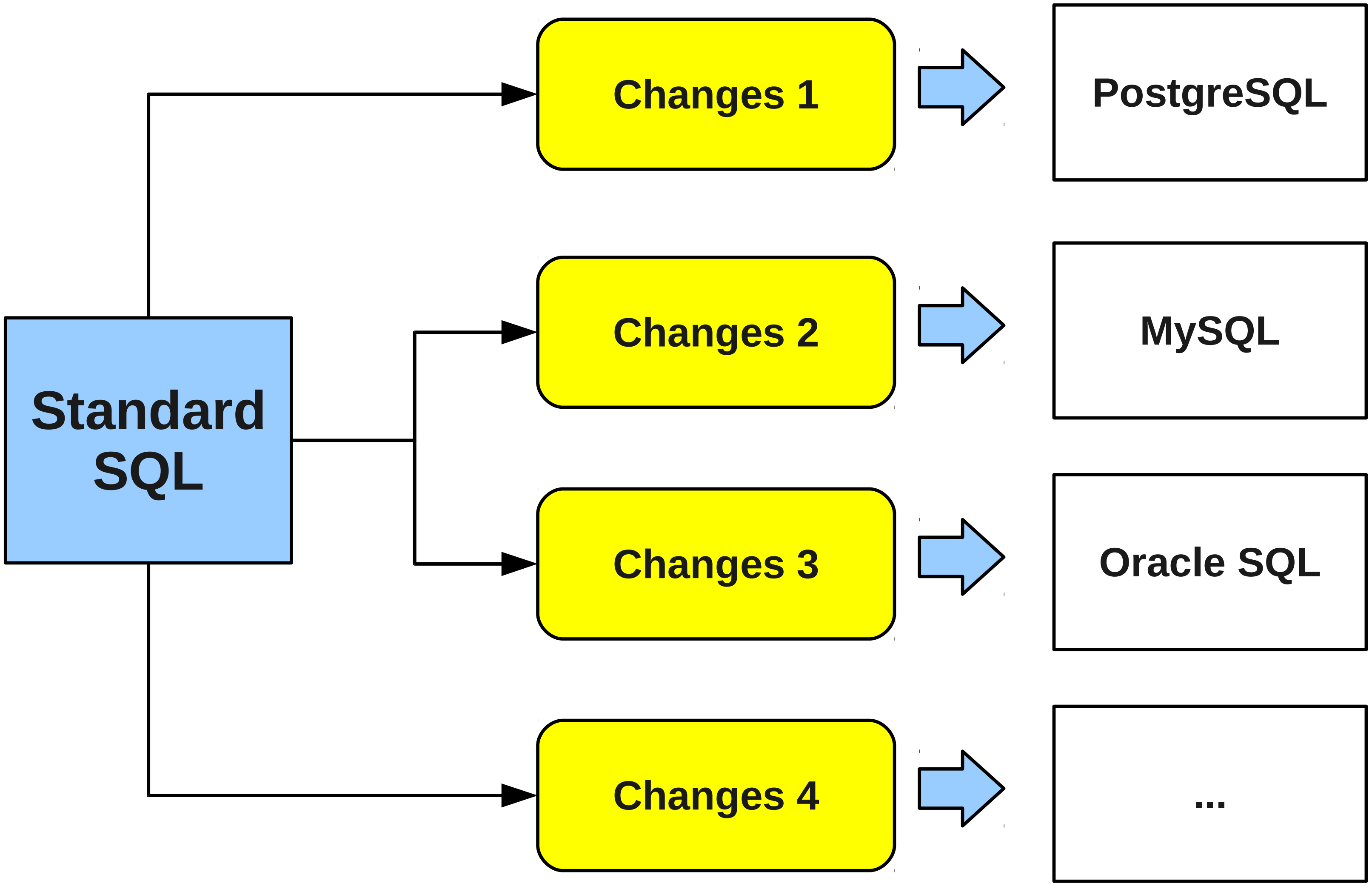
Advice examples: SQL Dialects

- Aspect rule
  - `tableExpression : ?f=fromClause .. ;`
    - `instead ?f : <?f>?;`
- Grammar rule

  -
    ```
    tableExpression
        : fromClause whereClause?
          groupByClause? HavingClause?;
    ```
- Result

  -
    ```
    tableExpression
        : fromClause? whereClause?
          groupByClause? havingClause?;
    ```

Using aspects for dialect definition



Managing language evolution

- Aspect rules may have **constraints** for **number of matches**
  - E.g., [1..1] – a rule must match once and only once
- **Wildcards** help to **abstract** the grammatical structure
  - Aspects tolerate changes that are matched by wildcards.

Summary

- We proposed **grammatical aspects (GAs)**
  - separation of concerns for grammars
  - tolerant to language evolution
- To solve the following problems:
  - Define IDE components
    - **Pretty-printers**
    - Syntax highlighters
    - ...
  - Define DSL families
    - E.g., **SQL dialects**