# Databases

# Active Databases

# Passive Databases

- Reaction strategies in the integrity constraints are the first example of the need to introduce a **reactive** behaviour in the databases:

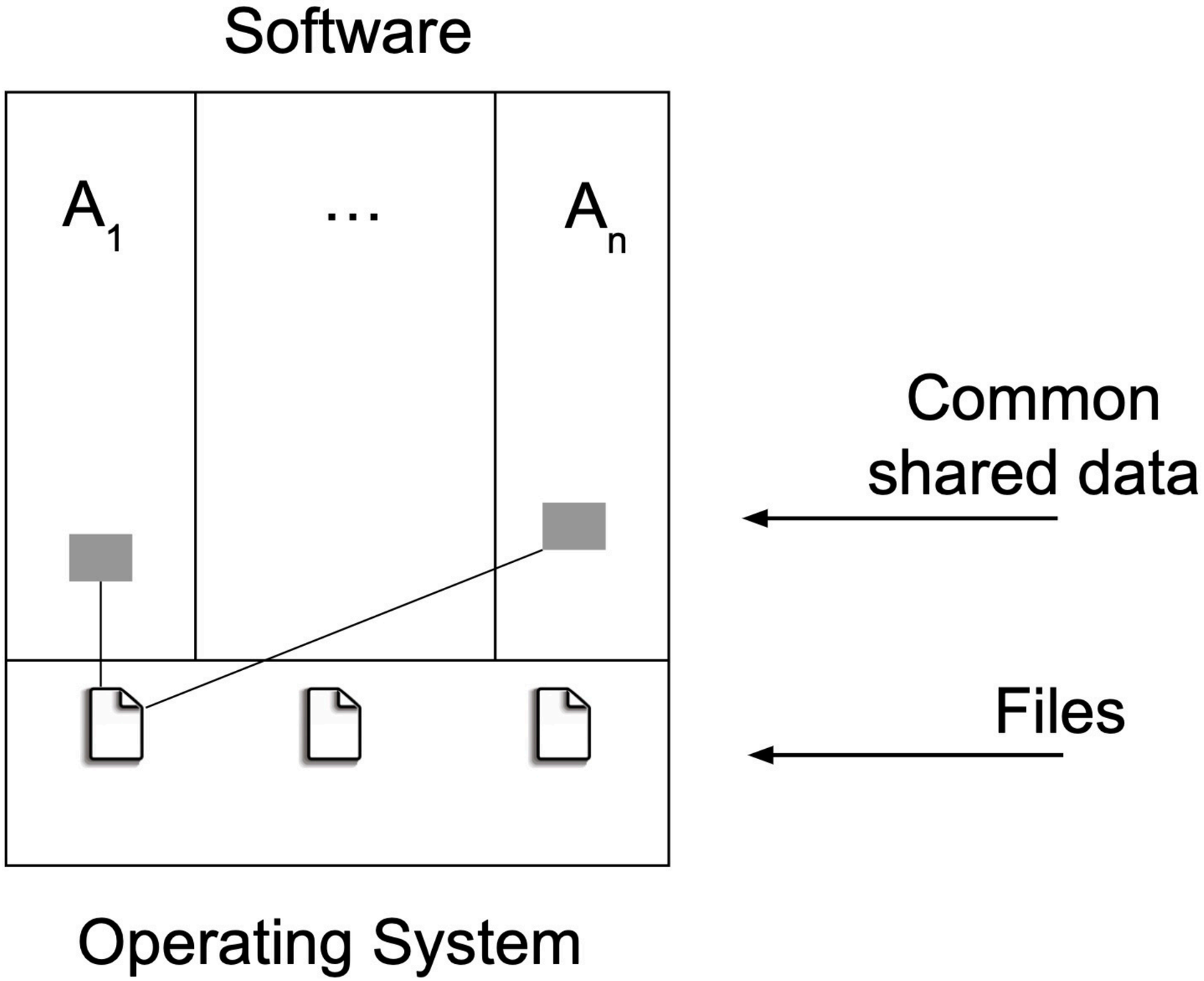
```
on    < delete | updated >
      < cascade | set null | set
        default | no action >
```

- The idea is to introduce **language constructs** specific for this goal
- These constructs are called (active) **rules** to handle a part of the procedural behaviour of an application
- Being at the database level, this behaviour is therefore “shared” among many applications, achieving data independence.

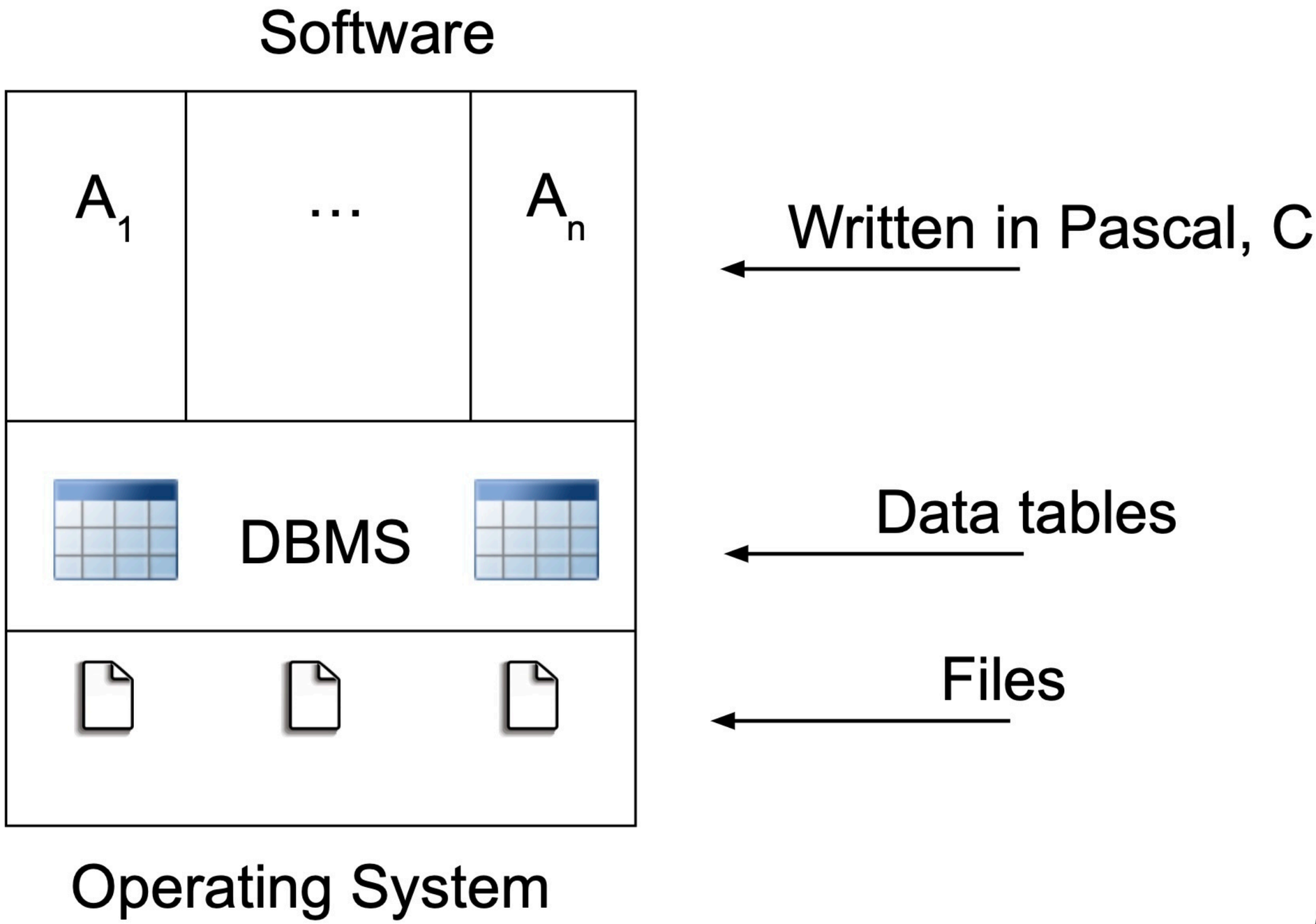
# Active Databases

- An active database has a component to handle active rules of the kind **Event-Condition-Action**:
  - events are changes in the database
  - a condition is verified, based on a true/false value
  - one or more actions are executed
- These databases have a **reactive** behaviour (opposite to **passive**): they do not just execute the user transactions, but also the rules
- Commercial DBMSs (SQL3) use **triggers** to specify rules, like the ones in the schema definition

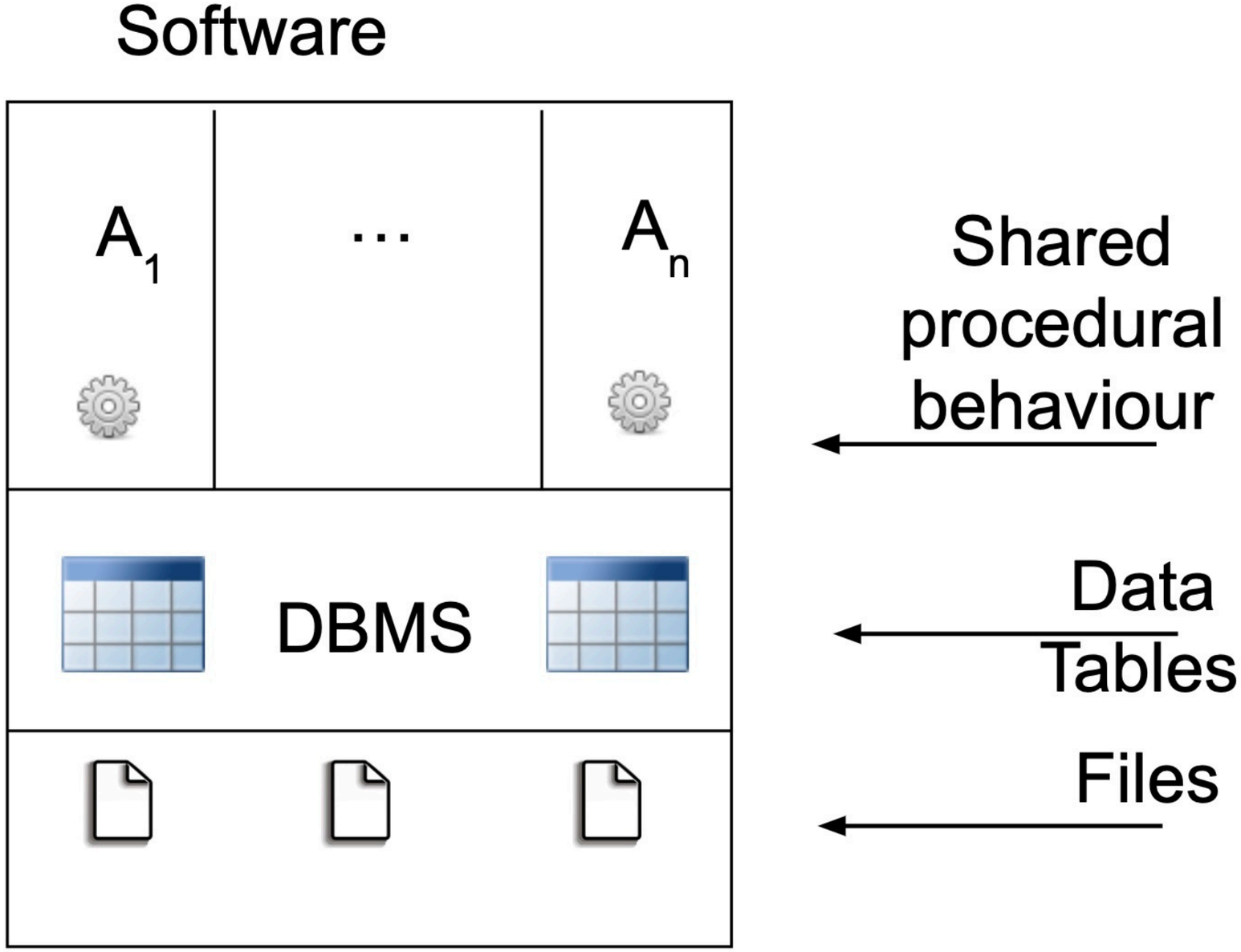
# The 70s: no DBMS



# The 80s: first DBMSs



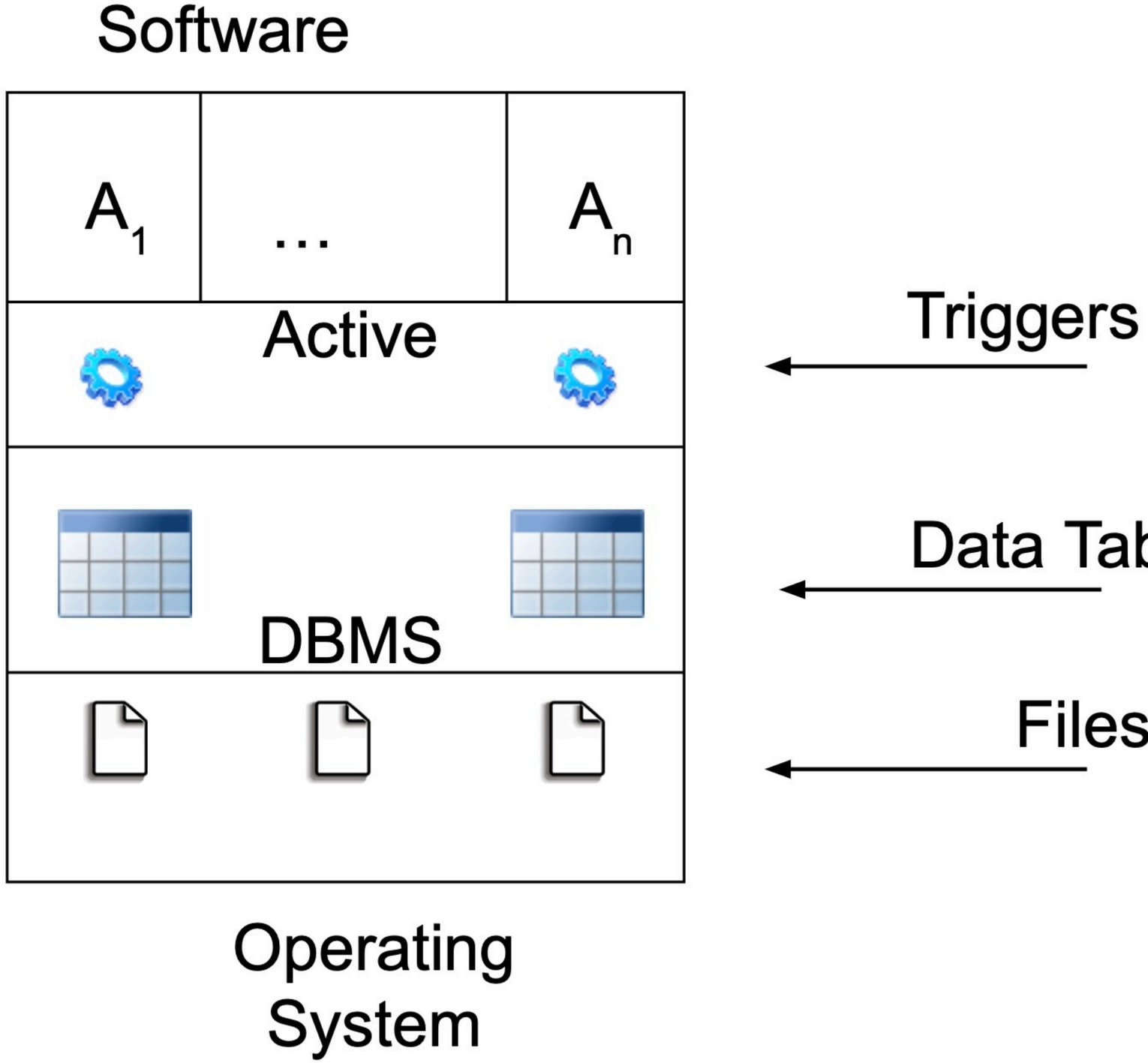
# The 90s: the procedural behaviour is shared among the software applications



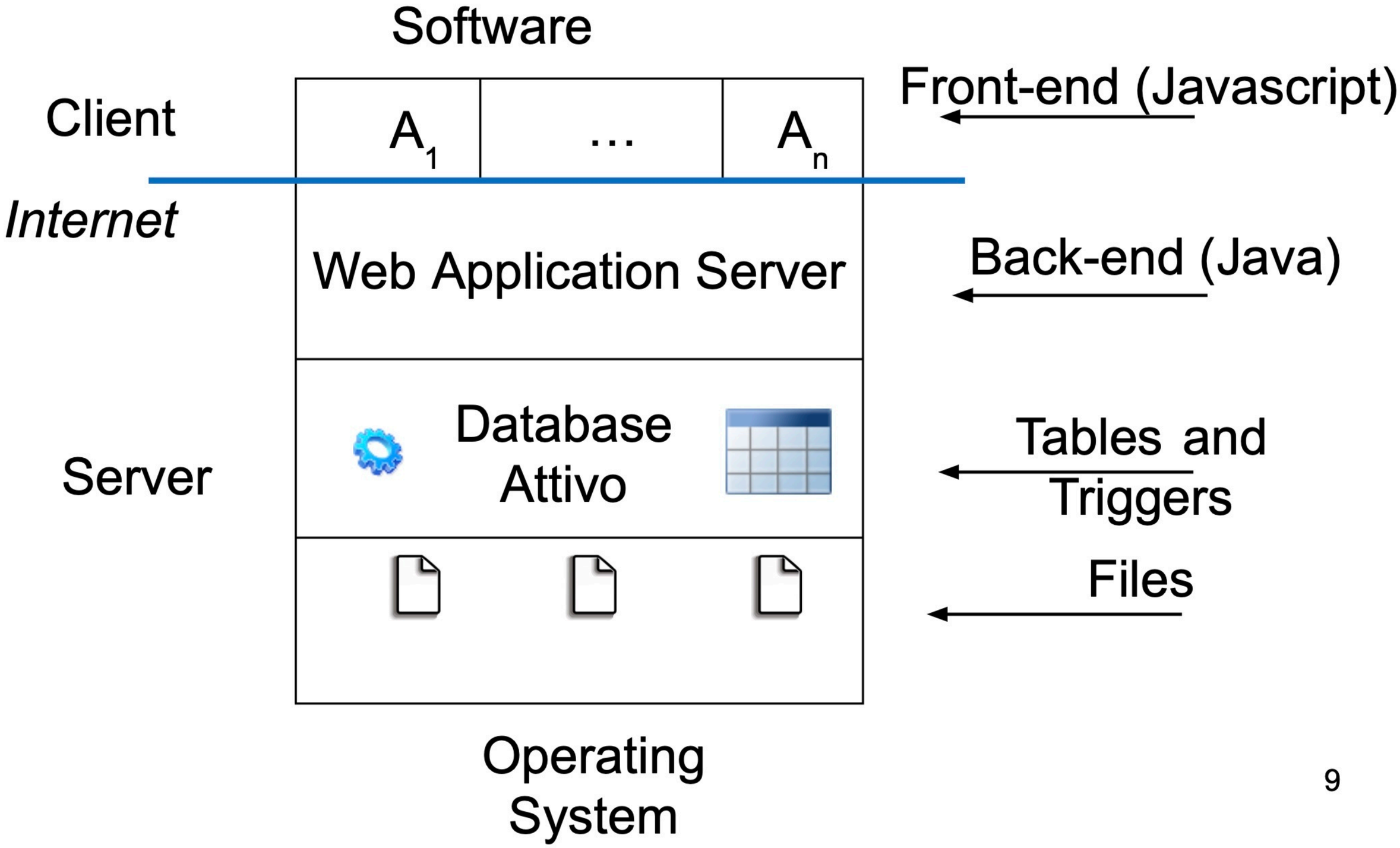
# Advances in DBMSs: Stored procedures

- Stored procedures have been introduced to share the common procedural behaviours between different software.
- Stored procedures are not standardized and are affected by the problem of impedance mismatch (we'll see later) with the language used to express such procedures
- As a result specific rules (*triggers*) have been introduced to model the procedural behaviour shared among different software and are handled by the DBMS itself.

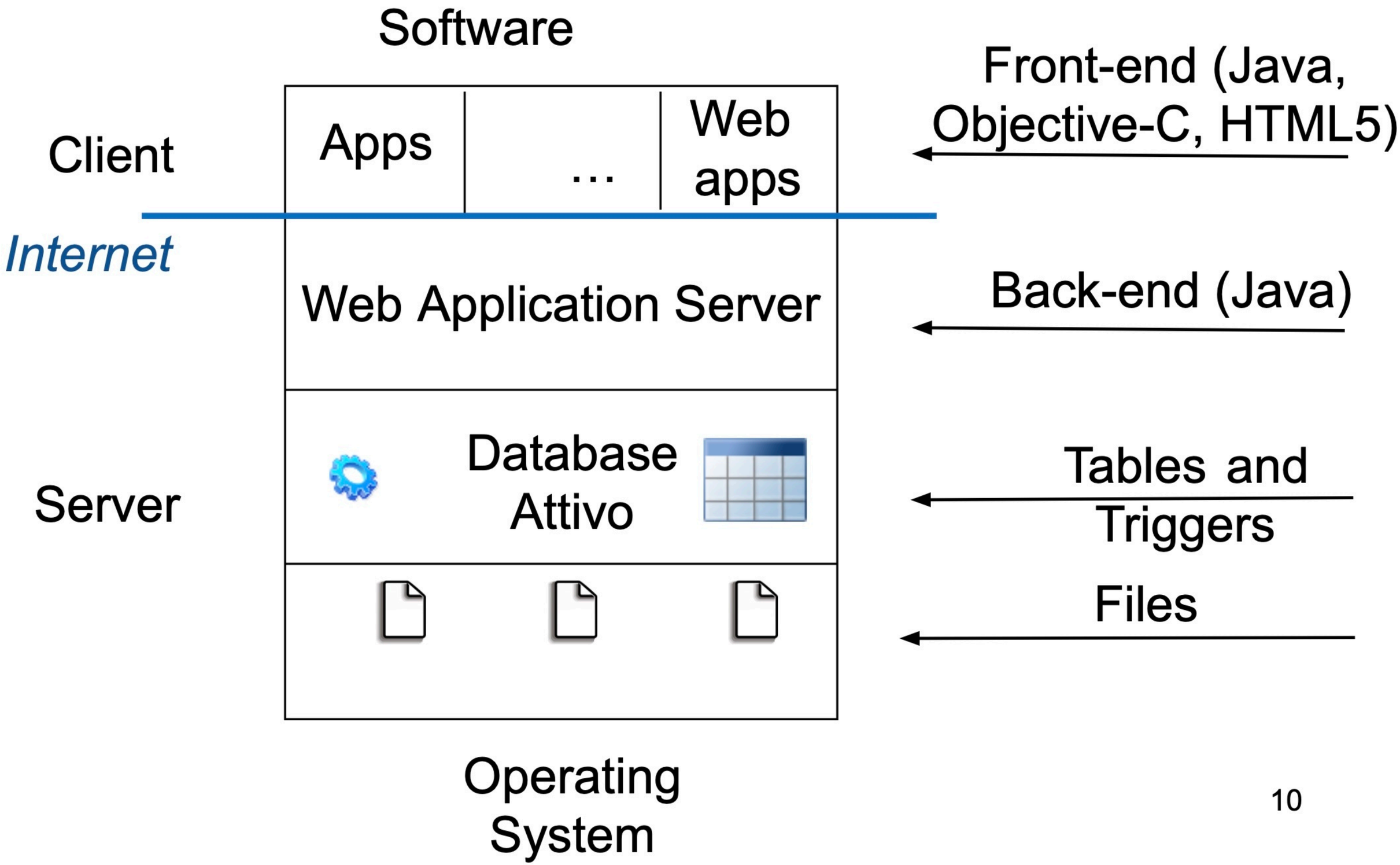
# The 90s: Active DBMS



# Advance in the 2000s



# The 2010s: mobile apps



# Trigger

- Defined with DDL instructions (`create trigger`)
  - based on the event-condition-action (ECA) paradigm:
    - event: change in the data, specified using `insert`, `delete`, `update`
    - condition: (optional) SQL predicate
    - action: sequence of SQL instructions (or extensions, for example PL/SQL in Oracle)
  - intuitively:
    - when there is an event (activation)
    - if the condition is satisfied (verification)
    - then execute the action (execution)
  - each trigger refers to a (target) table: it responds to events related to that table

# Trigger: granularities and modalities

- Granularity
  - of tuple (row-level): activation for each tuple involved in the operation
  - of operation (statement-level): only one activation for an SQL instruction, with reference to all the tuples involved ("set-oriented")
- Modality
  - immediate: right after (or right before) of the event
  - deferred: at the time of commit

# Computational model

- Let $T^U = U_1; \ldots ; U_n$ the user transaction
- If the rules of P have form E, C → A with E, Event, C, Condition and A, Action, then

- **Immediate** semantic generates $T^I = U_1; \underline{U}^P_1; \ldots U_n; \underline{U}^P_n$
- **Deferred** semantic generates $T^D = U_1; \ldots U_n; \underline{U}^P_1; \ldots ; \underline{U}^P_n$

- Where $\underline{U}^P_i$ represents the sequence of actions induced by $U_i$ on P
- Problems:
  - Confluence
  - Termination
  - Equivalence

# Trigger in Oracle, syntax

```
create trigger TriggerName
    Mode Event {, Event}
    on TargetTable
    [[referencing Reference]
    for each row
    [when SQLPredicate]]
    PL/SQLBlock
```

- *Mode:* `before` o `after`
- *Event:* `insert, update, delete`
- `for each row` specifies the granularity
- *Reference:* it allows to define variable names (usable only for tuple granularity)

  `old as` *OldVariable* | `new as` *NewVariable*

# Trigger in Oracle, semantic

- **Immediate** modality (both `after` and `before`)
- execution plan
  - trigger `before statement`
  - for each tuple involved
    - trigger `before row`
    - operation
    - trigger `after row`
  - trigger `after row`
- in the event of an error, everything is discarded
- priorities between triggers
- max 32 triggers activated in cascade

# Trigger in Oracle, example

```
create trigger Reorder
    after update of QtyAvbl on Warehouse
    when (new.QtyAvbl < new.QtyLimit)
    for each row
        declare X number;
    begin
        select count(*) into X
        from PendingOrders
        where Part = new.Part;
        if X = 0
        then
            insert into PendingOrders
            values (new.Part, new.QtyReord, sysdate);
        end if;
end;
```

# Trigger in Oracle, example 2

Warehouse

| Part | QtyAvbl | QtyLimit | QtyReord |
|---|---|---|---|
| 1 | 200 | 150 | 100 |
| 2 | 780 | 500 | 200 |
| 3 | 450 | 400 | 120 |

```
T1: update Warehouse
    set QtyAvbl = QtyAvbl - 70
    where Part = 1

T2: update Warehouse
    set QtyAvbl = QtyAvbl - 60
    where Part <= 3
```

# Trigger in DB2, syntax

```
create trigger TriggerName
    Mode Event on TargetTable
    [referencing Reference]
    for each Level
    [when (SQLPredicate)]
    SQLProceduralStatement
```

- *Mode:* `before o after`
- *Event:* `insert, update, delete`
- `for each` *Level* specifies the granularity
- *Reference:* it allows to define variable names (depending on granularity):

```
        old as OldTupleVar | new as NewTupleVar
old_table as OldTableVar | new_table as NewTableVar
```

# Trigger in DB2, semantic

- **Immediate** modality (both `after` and `before`)
- `before` triggers cannot modify the database, apart from variants on the changes caused by the event (therefore they cannot in general activate other triggers)
- In the event of an error, everything is discarded
- No priority between triggers (the order is defined by the system), interaction with compensatory actions on referential integrity constraints
- max 16 triggers activated in cascade

# Trigger in DB2, example

```
foreign key (Supplier)
    references Distributor
    on delete set null

create trigger SoleSupplier
    before update of Supplier on Part
    referencing new as N
    for each row
    when (N.Supplier is not null)
    signal sqlstate '70005' ('Cannot change supplier')

create trigger AuditPart
    after update on Part
    referencing old_table as OT
    for each statement
    insert into Audit
    values(user, current-date, (select count(*) from OT))
```

# Extensions (usually not available)

- temporal events (also periodical) or “user-defined”
- boolean combinations of events
- clause `instead of`: it does not execute the operation that activated the event, but another action instead
- “detached” execution: an autonomous transaction is activated
- priorities definition
- groups rules, can be activated and deactivated
- rules associated also with queries (not just updates)

# Rules properties

- termination (essential)
- confluence
- determinism of observations

# Applications

- internal features
  - handling of integrity constraints
  - replication
  - view management
    - materialized: propagation
    - virtual: modification of the queries
- application features: description of the behaviour of the database