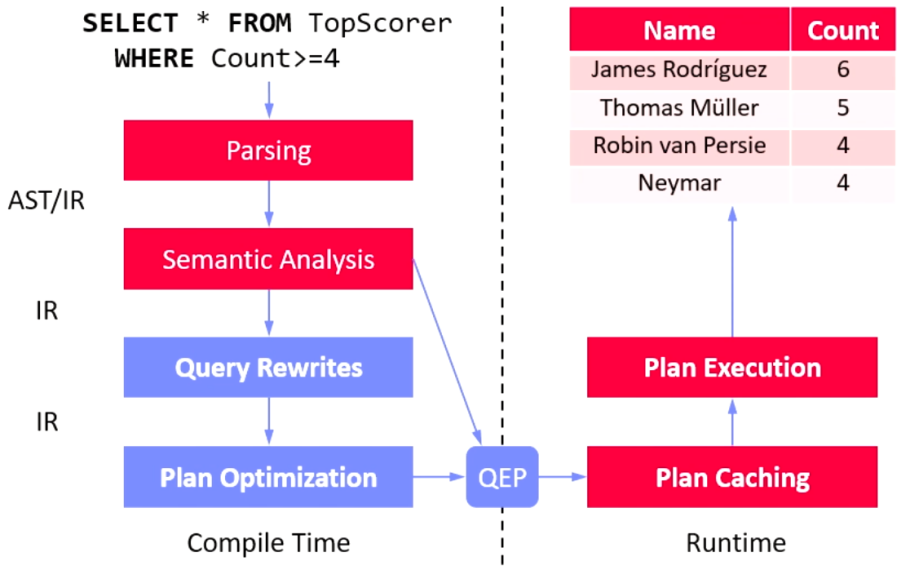
Overview

- query execution consists of four steps
  - parsing
  - semantic analysis
    - do all tables/tuples exist
    - checks user permissions
  - [[Query Rewriting]]
  - [[Plan Optimization]]
- query execution plan
  - semantic analysis creates QEP
  - plan optimization creates optimized QEP
  - can be executed by runtime
- runtime may store results in cache use again later
- 

Overview Execution Strategies

- different strategies with different pros and cons
- (Volcano) iterator model
  - see [[Physical Operators]]
- materialized intermediates
  - one column at a time
  - uses binary association tables (BATs)

```
SELECT count(DISTINCT o_orderkey)
  FROM orders, lineitem
  WHERE l_orderkey = o_orderkey
    AND o_orderdate >= date '1996-07-01'
    AND o_orderdate < date '1996-07-01'
      + interval '3' month
    AND l_returnflag = 'R';
```

Column-oriented storage
Efficient array operations
DAG processing
Reuse of intermediates
Memory requirements
Unnecessary read/write from and to memory

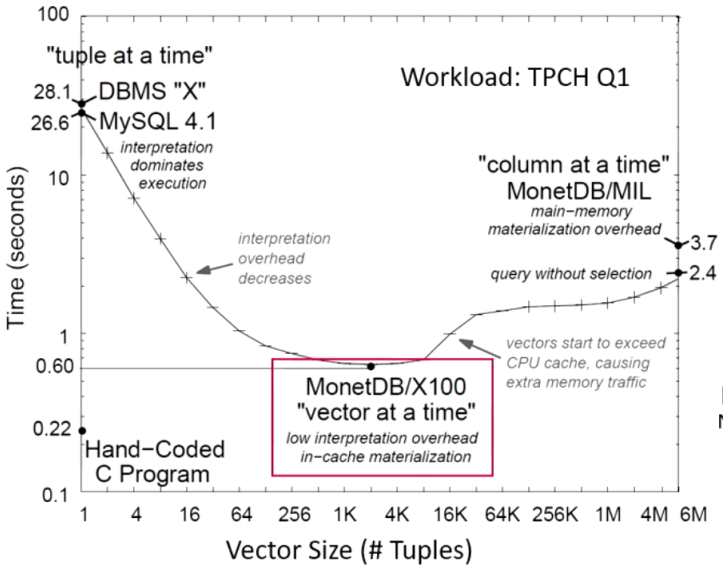
```
function user.s1_2(A0:date,A1:date,A2:int,A3:str):void;
  X5 := sql.bind("sys","lineitem","l_returnflag",0);
  X11 := algebra.uselect(X5,A3);
  X14 := algebra.markT(X11,0@0);
  X15 := bat.reverse(X14);
  X16 := sql.bindIdxbat("sys","lineitem","l_orderkey_fkey");
  X18 := algebra.join(X15,X16);
  X19 := sql.bind("sys","orders","o_orderdate",0);
  X25 := mtime.addmonths(A1,A2);
  X26 := algebra.select(X19,A0,X25,true,false);
  X30 := algebra.markT(X26,0@0);
  X31 := bat.reverse(X30);
  X32 := sql.bind("sys","orders","o_orderkey",0);
  X34 := bat.mirror(X32);
  X35 := algebra.join(X31,X34);
  X36 := bat.reverse(X35);
  X37 := algebra.join(X18,X36);
  X38 := bat.reverse(X37);
  X40 := algebra.markT(X38,0@0);
  X41 := bat.reverse(X40);
  X45 := algebra.join(X31,X32);
  X46 := algebra.join(X41,X45);
  X49 := algebra.selectNotNil(X46);
  X50 := bat.reverse(X49);
  X51 := algebra.kunique(X50);
  X52 := bat.reverse(X51);
  X53 := aggr.count(X52);
  sql.exportValue(1,"sys.orders","L1","wrd",32,0,6,X53);
end s1_2;
```

Binary Association Tables (BATs:=OID/Val)

[Milena Ivanova, Martin L. Kersten, Niels J. Nes, Romulo Goncalves: An architecture for recycling intermediates in a column-store. **SIGMOD 2009**]

–

- vectorized (batched) execution
  - one vector at a time
    - **Idea: Pipelining of vectors (sub columns) s.t. vectors fit in CPU cache**



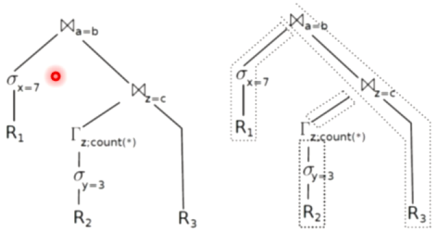
[Peter A. Boncz, Marcin
Niels Nes: MonetDB/X1
Pipelining Query

–

- query compilation
  - no longer operator centric ==> data centric
  - blurred boundaries between operators

- **Idea: Data-centric, not op-centric processing + LLVM code generation**

Operator Trees
(w/o and w/ pipeline boundaries)



[Thomas Neumann: Efficiently Compiling Efficient Query Plans for Modern Hardware, **PVLDB 2011**]

Compiled Query
(conceptual, not LLVM)

initialize memory of $\bowtie_{a=b}$, $\bowtie_{c=z}$, and Γ_z
for each tuple t in R_1
  if $t.x = 7$
    materialize t in hash table of $\bowtie_{a=b}$
for each tuple t in R_2
  if $t.y = 3$
    aggregate t in hash table of Γ_z
for each tuple t in Γ_z
  materialize t in hash table of $\bowtie_{z=c}$
for each tuple t_3 in R_3
  for each match t_2 in $\bowtie_{z=c}[t_3.c]$
    for each match t_1 in $\bowtie_{a=b}[t_3.b]$
      output $t_1 \circ t_2 \circ t_3$