数据库系统概论新技术篇

内存数据库

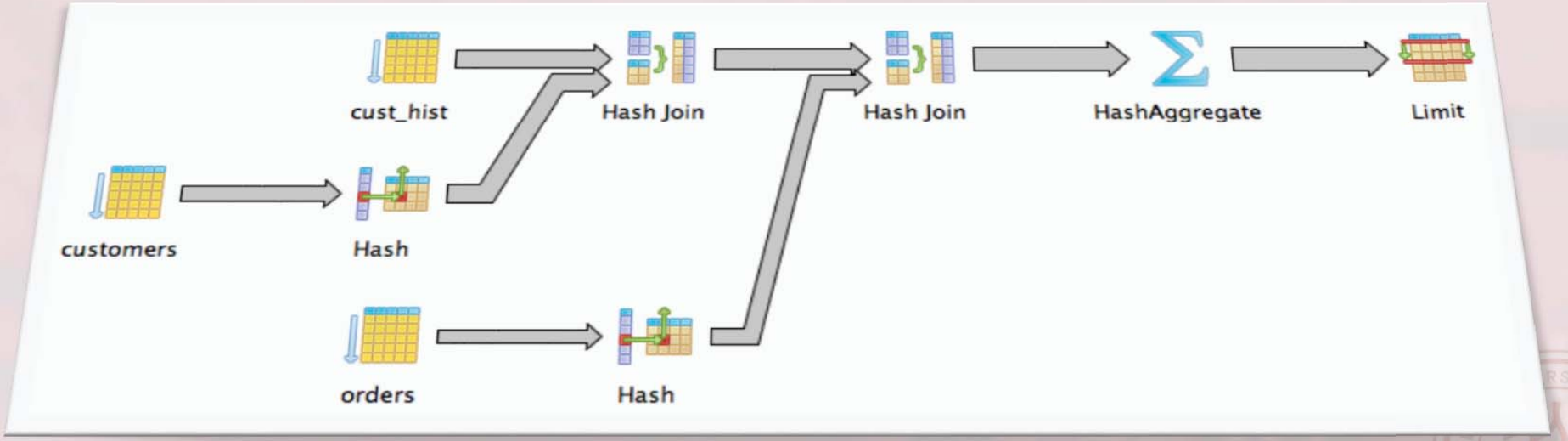
张延松

中国人民大学信息学院

2017年4月

目录

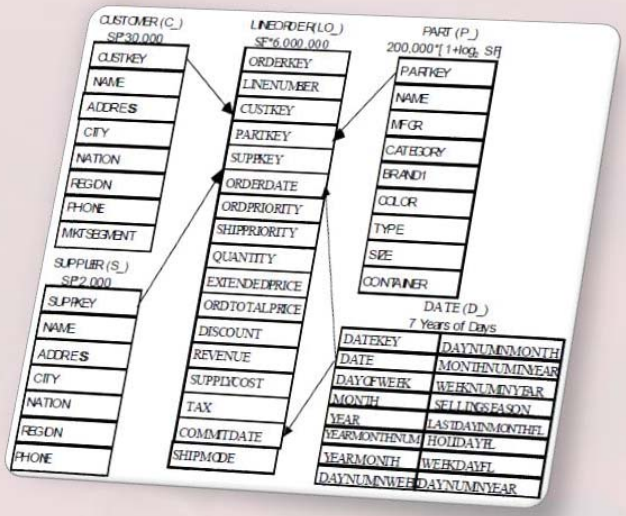
内存数据库技术示例


cust_hist
Hash Join
Hash Join
HashAggregate
Limit
customers
Hash
orders
Hash

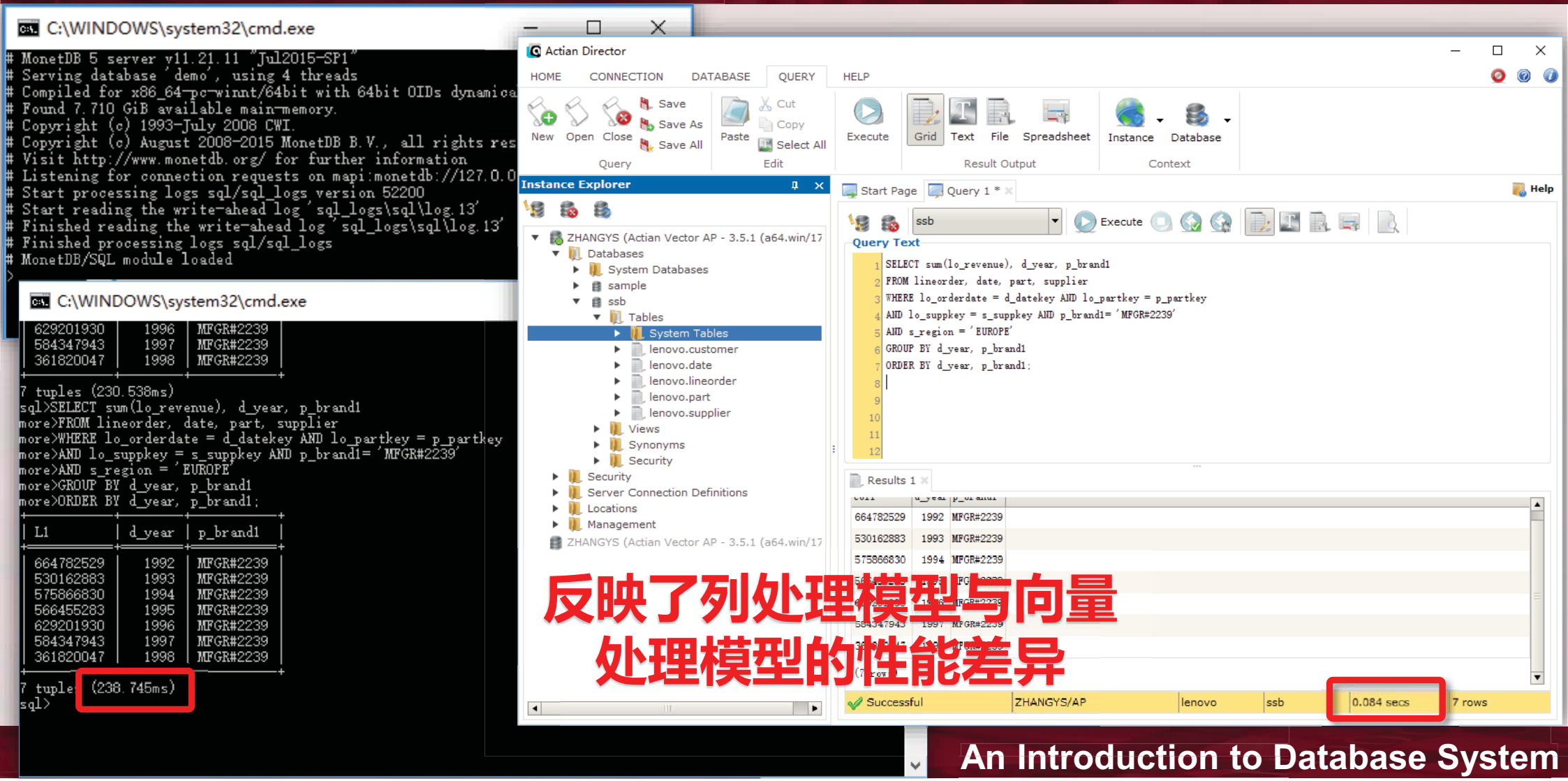
内存数据库系统示例

- 内存数据库：
  - MonetDB
  - Vectorwise
- 测试数据集：
  - SSB基准测试，SF=1，事实表包含6,000,000条记录
- 测试查询：

```
SELECT c_nation, s_nation, d_year, SUM(lo_revenue) AS revenue
FROM customer, lineorder, supplier, date
WHERE lo_custkey = c_custkey
AND lo_suppkey = s_suppkey AND lo_orderdate = d_datekey
AND c_region = 'ASIA' AND s_region = 'ASIA'
AND d_year >= 1992 AND d_year <= 1997
GROUP BY c_nation, s_nation, d_year
ORDER BY d_year asc, revenue desc;
```



MonetDB和Vector内存数据库查询执行



反映了列处理模型与向量处理模型的性能差异

MonetDB存储模型：MonetDB二元表BAT

❖Simple Data Model – BAT

ID	lo_shipmode	lo_quantity
10	TRUCK	2385
11	AIR	453
12	RAIL	5631
13	MAIL	3426

OID	ID
100	10
101	11
102	12
103	13

OID	lo_shipmode
100	TRUCK
101	AIR
102	RAIL
103	MAIL

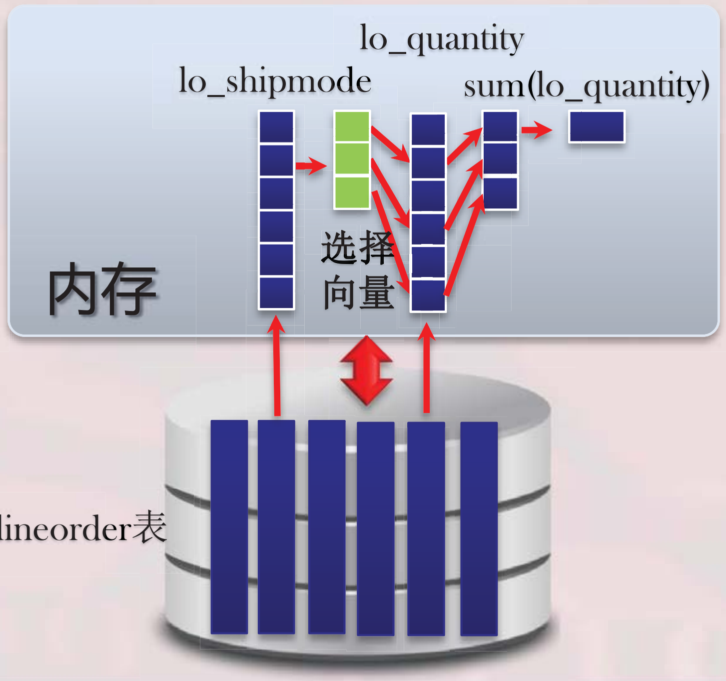
OID	lo_quantity
100	2385
101	453
102	5631
103	3426

列存储、列压缩、列即是索引、cache性能更高的列访问

MonetDB列式查询处理示例

❖ **MonetDB查询示例分析**

SELECT sum(lo_quantity)
FROM lineorder
WHERE lo_shipmode='TRUCK'



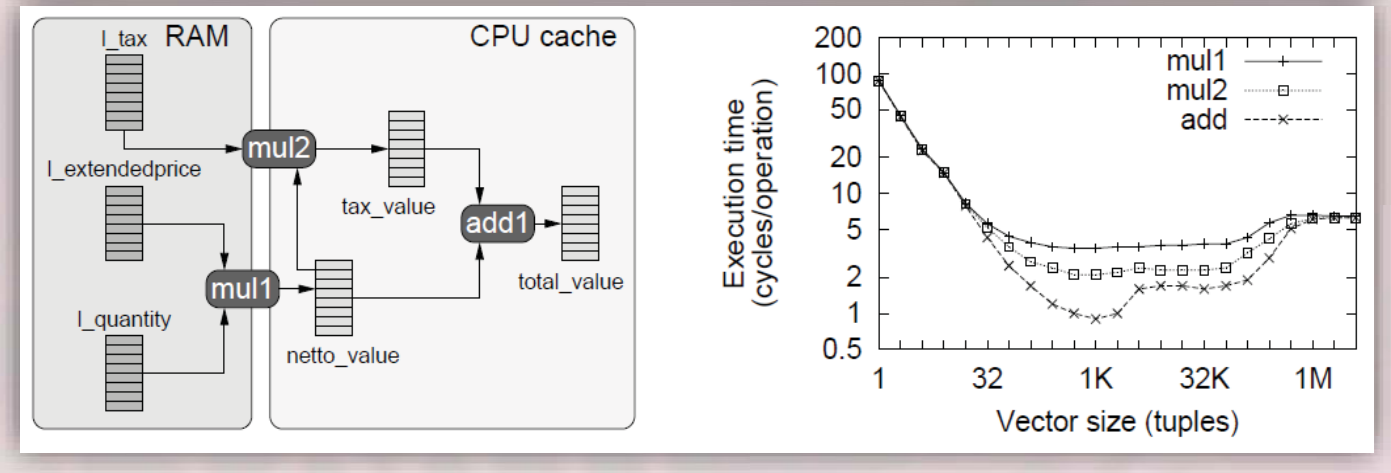
C:\WINDOWS\system32\cmd.exe

```
sql>SELECT sum(lo_quantity) FROM lineorder WHERE lo_shipmode='TRUCK';
+----------+
| L1       |
+==========+
| 17478486 |
+----------+
1 tuple (57.713ms)
sql>explain SELECT sum(lo_quantity) FROM lineorder WHERE lo_shipmode='TRUCK';
+------------------------------------------------------------------------------+
| mal                                                                          |
+==============================================================================+
| function user.s3_1{autoCommit=true}(A0:str):void;                            |
|     X_30:void := querylog.define("select sum(lo_quantity)\nfrom\n    lineor |
: der\nwhere lo_shipmode=\\'TRUCK\\';","default_pipe",18);                     :
|     X_3 := sql.mvc();                                                        |
|     X_7:bat[:oid,:str] := sql.bind(X_3,"sys","lineorder","lo_shipmode",0);   |
|     X_4:bat[:oid,:oid] := sql.tid(X_3,"sys","lineorder");                    |
|     X_46 := algebra.subselect(X_7,X_4,A0,A0,true,false,false);               |
|     (X_10,r1_10) := sql.bind(X_3,"sys","lineorder","lo_shipmode",2);         |
|     X_47 := algebra.subselect(r1_10,nil:bat[:oid,:oid],A0,A0,true,false,fal |
: se);                                                                         :
|     X_13:bat[:oid,:str] := sql.bind(X_3,"sys","lineorder","lo_shipmode",1);  |
|     X_49 := algebra.subselect(X_13,X_4,A0,A0,true,false,false);              |
|     X_15 := sql.subdelta(X_46,X_4,X_10,X_47,X_49);                           |
|     X_17:bat[:oid,:int] := sql.bind(X_3,"sys","lineorder","lo_quantity",0);  |
|     (X_19,r1_22) := sql.bind(X_3,"sys","lineorder","lo_quantity",2);         |
|     X_21:bat[:oid,:int] := sql.bind(X_3,"sys","lineorder","lo_quantity",1);  |
|     X_22 := sql.projectdelta(X_15,X_17,X_19,r1_22,X_21);                     |
|     X_23:lng := aggr.sum(X_22);                                              |
|     sql.resultSet("sys.L1","L1","bigint",64,0,7,X_23);                       |
| end user.s3_1;                                                               |
| # optimizer.mitosis()                                                        |
| # optimizer.dataflow()                                                       |
+------------------------------------------------------------------------------+
20 tuples (1.069ms)
sql>
```

面向SIMD(单指令多数据)的向量计算
简单BAT结构，简单计算模型

向量处理技术优化cache数据访问

- **目标**：减少中间结果的内存物化代价
- **方法**：以适合L1 cache的较小粒度的向量作为查询处理的数据粒度，通过L1 cache提高中间结果的物化和访问性能
- **优化策略**：使用向量个数越多，性能最佳时向量长度区间越窄，当向量能够完全容纳在L1 cache时查询性能最优



向量查询处理技术案例分析

- 以适合L1 cache大小的向量作为查询处理的单位，查询处理中间结果在L1 cache中物化，减少内存访问代价，提高代码执行效率。

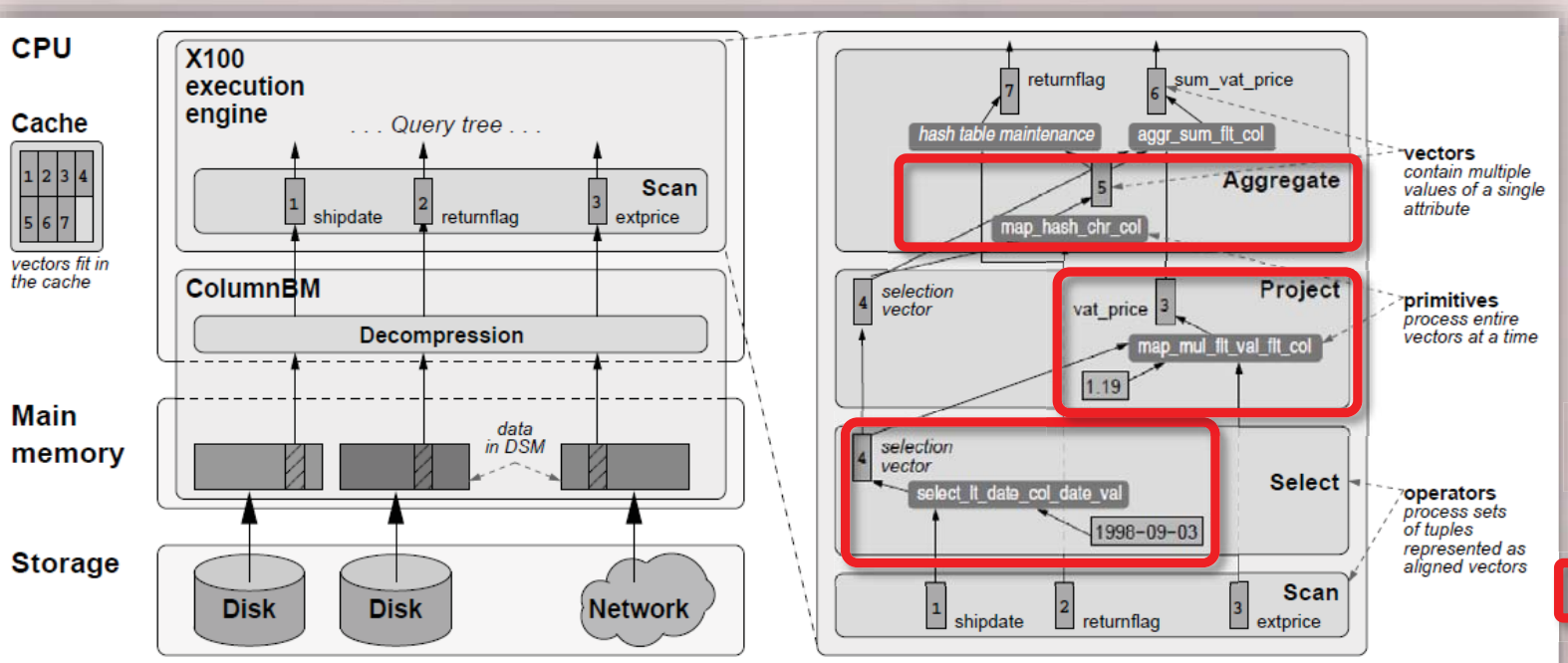
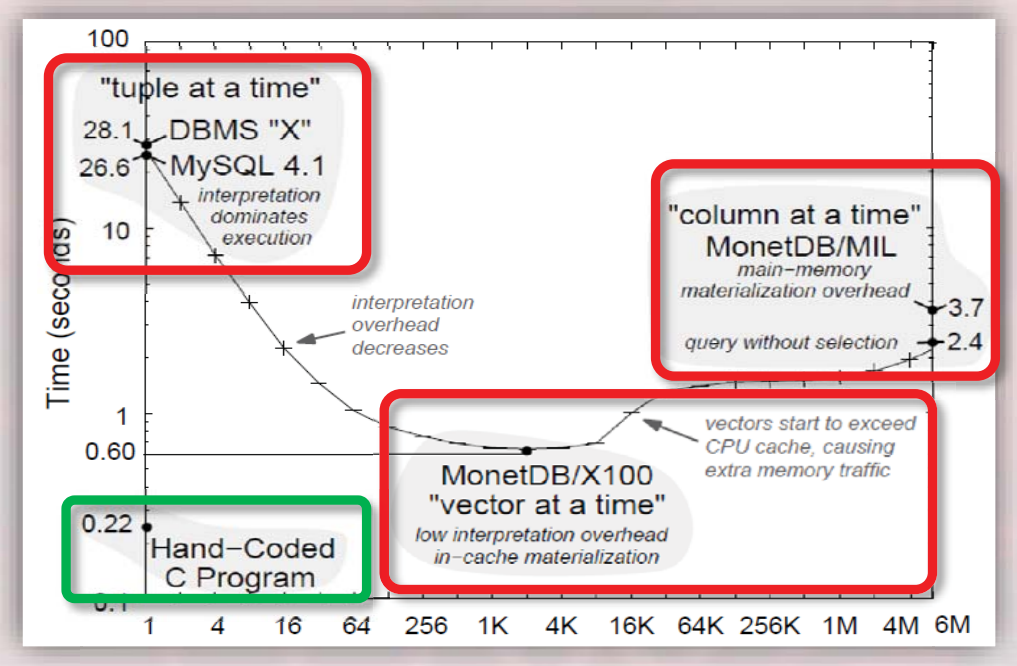


Figure 1: X100 – architecture overview and execution layer example

```
SELECT returnflag,sum
(1.19*extprice) as vat_price
FROM
lineitem
WHERE
shipdate<='1998-09-03'
GROUP BY
returnflag;
```

不同查询处理模型性能特征

- 行存储、列存储、向量存储查询性能
  - **Row-wise:** 迭代处理产生的函数调用及查询解析代价高
  - **Column-wise:** 中间结果内存物化操作的存储及处理代价高
  - **Vectorized processing:** 减少物化代价，分摊迭代处理代价



向量处理模型成为当前内存数据库的主流技术

如何进一步提高数据库查询执行性能：执行代码优化

基于JIT实时编译技术的查询处理模型

- 传统迭代处理模型：
  - “拉”模式迭代访问底层操作符
  - 物化破坏数据在寄存器的局部性
- JIT实时编译处理模型：
  - “推”模式执行查询处理
  - **Low Level Virtual Machine (LLVM)** 编译框架实时生成高效机器码

```
select   *
from     R1,R3,
         (select   R2.z,count(*)
         from      R2
         where     R2.y=3
         group by R2.z) R2
where    R1.x=7 and R1.a=R3.b and R2.z=R3.c
```

Figure 2: Example Query

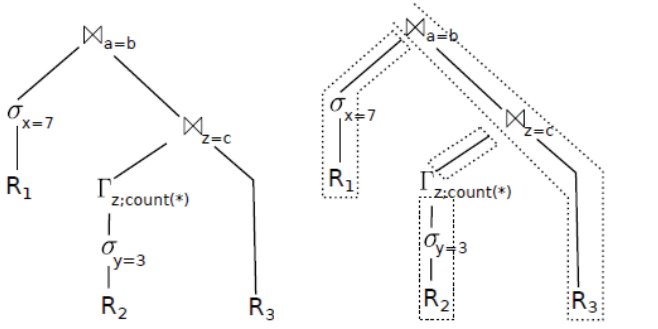
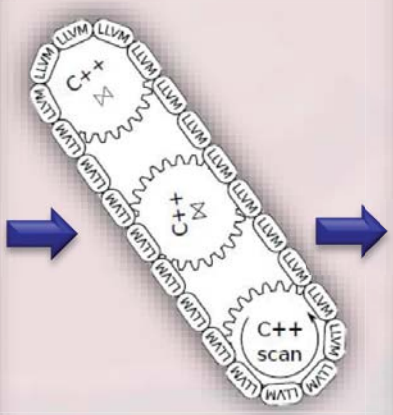
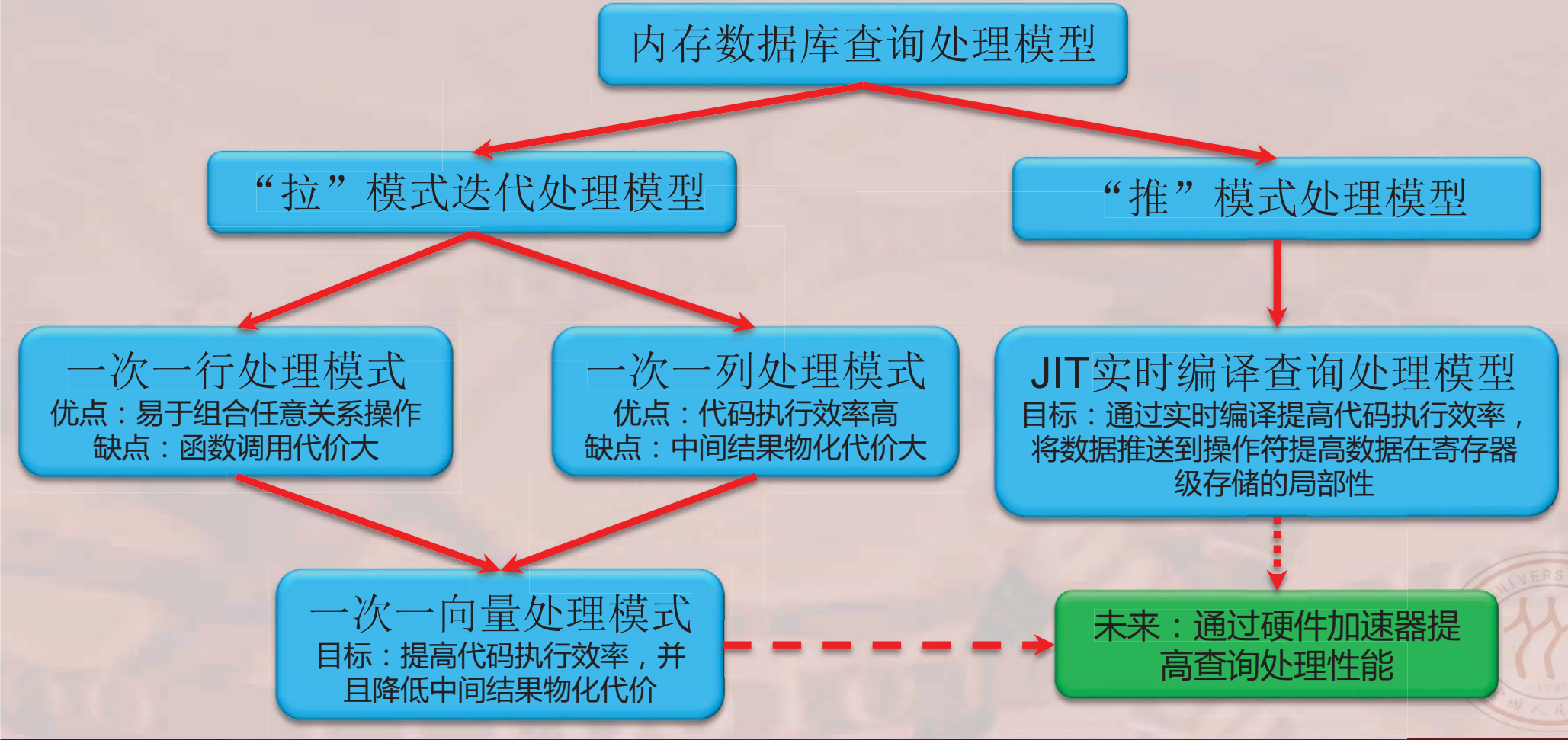


Figure 3: Example Execution Plan for Figure 2



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 $t_1 \circ t_2 \circ t_3$

小结


内存数据库查询处理模型
“拉”模式迭代处理模型
“推”模式处理模型
一次一行处理模式
优点：易于组合任意关系操作
缺点：函数调用代价大
一次一列处理模式
优点：代码执行效率高
缺点：中间结果物化代价大
JIT实时编译查询处理模型
目标：通过实时编译提高代码执行效率，将数据推送到操作符提高数据在寄存器级存储的局部性
一次一向量处理模式
目标：提高代码执行效率，并且降低中间结果物化代价
未来：通过硬件加速器提高查询处理性能