

Query Execution

In-memory的DBMS，没有了磁盘IO瓶颈，那么可能存在的优化点在于？

优化目标

减少要执行指令数目

减少每条指令执行的周期

- 分支预测
- 减少Cache Miss

并发执行

- 使用多线程

Operator Operation

- Query Plan Processing
- Scan Sharing
- 物化视图
- Query Compilation
- 向量化Operators
- 并行算法
- UDF

今天的内容

CPU

CPU指令以流水线组织运行

超标量CPU支持多流水线(如果多条指令互相独立,可以并发在单周期执行)

CPU / DBMS PROBLEMS

- Dependencies

如果指令相互依赖，不能够同时在同一条pipeline中执行。

- Branch Prediction

CPU会尝试预测下一个周期执行的是哪个分支，如果预测失败了就会flush pipeline

对OLAP的DBMS，范围搜索中where操作其实基本上是预测不了的

PROCESSING MODEL

决定DBMS如何执行query计划

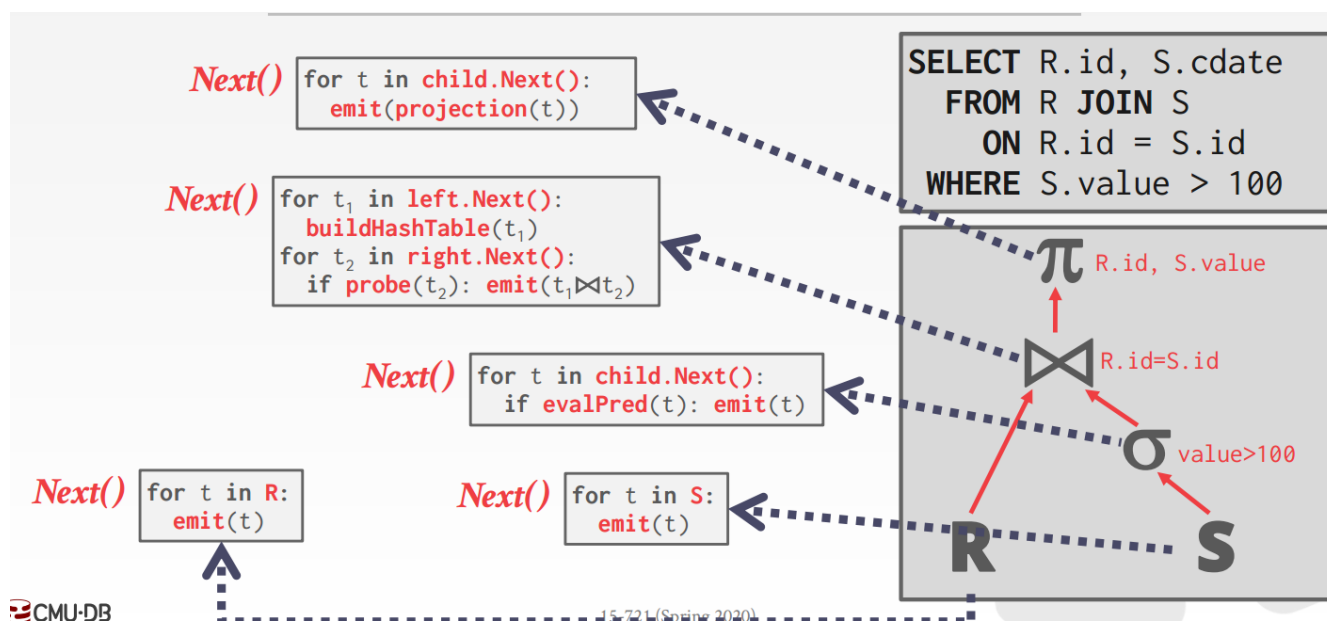
Iterator Model

每一个query plan实现一个next函数

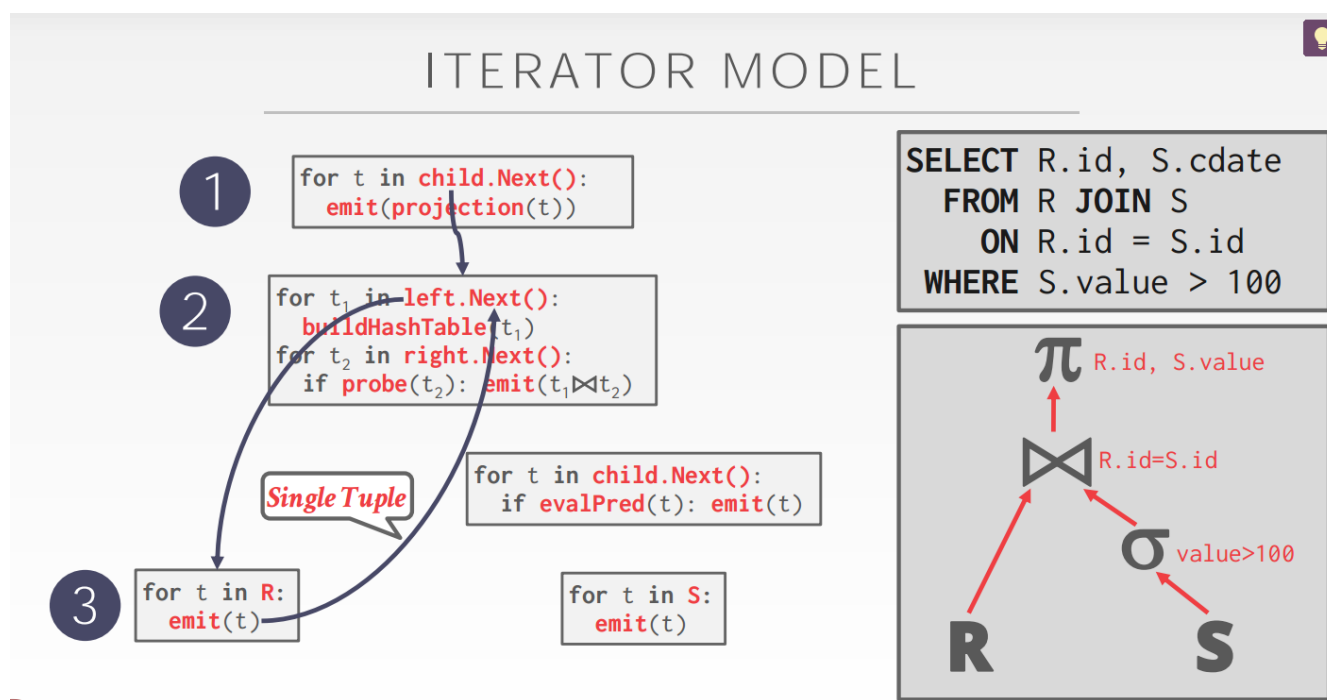
每次调用的时候，operator要么返回一个tuple结果，要么一个null标志没有其他tuple

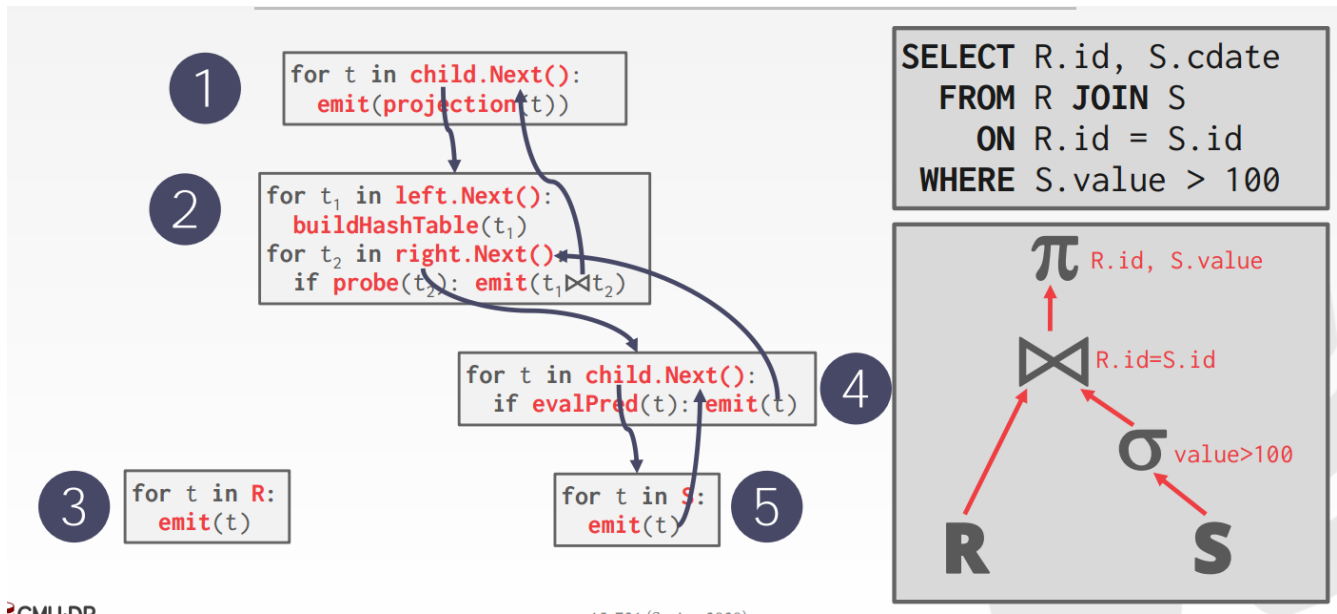
operator会实现一个loop，对其子operator调用next来获得其对应的tuple然后处理他们。

也叫做 volcano 或者 Pipeline 模型



像下图那样，每个operator对应一个for loop,每次子operator会返回单个tuple的结果





几乎所有的DBMS都会采用这种模型。允许流水线执行tuple
 一些算子operator会阻塞直到其子operator输出完所有数据
 eg: Join, Subqueries, Order by

This is used in almost every DBMS. Allows for tuple **pipelining**.

Some operators must block until their children emit all their tuples.

→ Joins, Subqueries, Order By

Output control works easily with this approach.

Materializaion Model

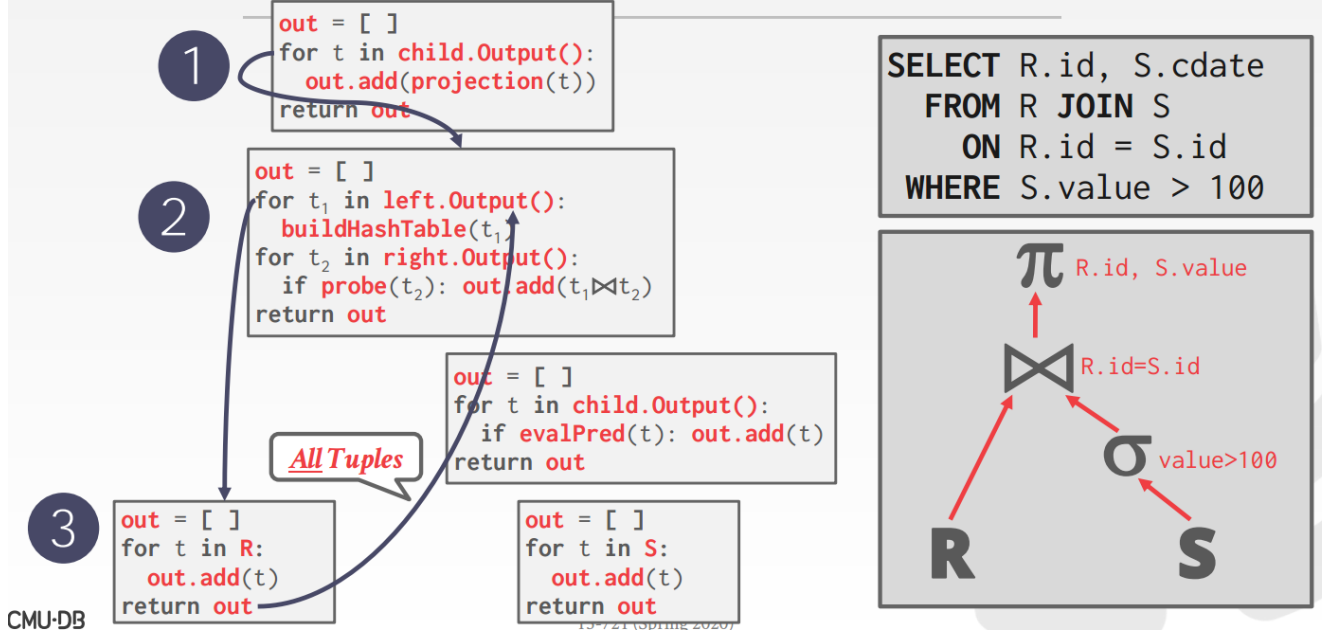
每个算子一次性处理所有输入输出

物化指的是把输出结果物化成一个单一的结果

DBMS可以把把hints下移，避免扫描过多的tuples

输出结果可以是全部tuple(行式存储),也可以是某些列(列式存储)

MATERIALIZATION MODEL



每个operator的 `out[]` 是buffer，结果会放入buffer中，operator处理完之后会把buffer中的所有结果交付给父operator

适用于OLTP，因为OLTP一次只会访问少量的tuple结果

- 更少的execution/coordination 开销
- 更少的函数调用

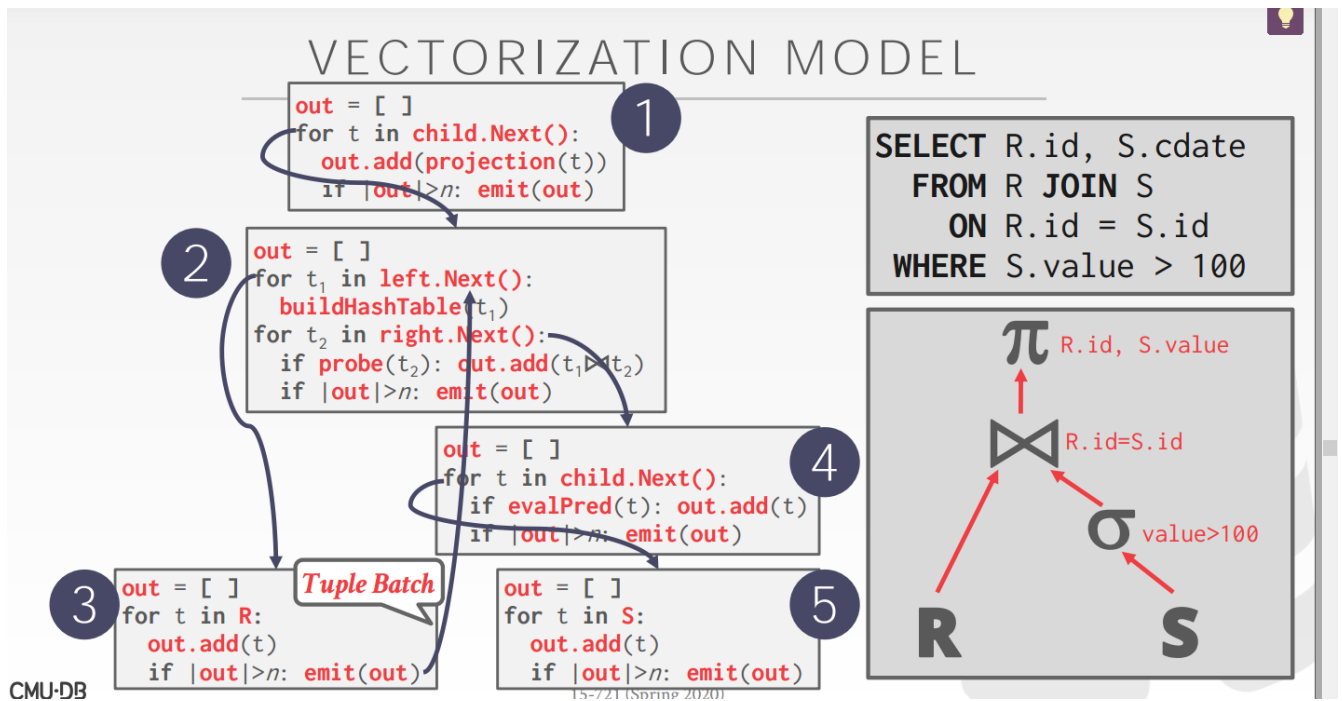
不适用于OLAP，因为OLAP会访问部分列的所有tuple，对结果实时性要求强

Vectorized/Batch Model

与Iterator Model类似，每一个operator会实现一个next函数

每一个operator不会emit单个tuple结果，而是emit批结果

- 每个operator的内部循环会一次处理多个tuple
- 处理的批的大小取决于硬件和query的性质



适用于OLAP数据库

因为极大地减少了每个operator的调用次数

允许operator使用SIMD(单指令多数据)指令来对tuple做批次处理

生成计划处理方向

方法一： 由上至下

- 从根operator开始，从子operator处理完的数据pull到父节点
- tuple往往通过函数调用来传递

方法二： 由下至上

- 从叶子节点开始，并且把子operator处理的数据push到父operator
- 对缓存和寄存器会有更精细化的管理

INTER-QUERY 并行

同时允许多条query执行

Improve overall performance by allowing multiple queries to execute simultaneously.

→ Provide the illusion of isolation through concurrency control scheme.

The difficulty of implementing a concurrency control scheme is not significantly affected by the DBMS's process model.

INTRA-QUERY 并行

单条Query内允许并行执行多个operator (SINGLE QUERY PARALLEL OPERATOR)

INTRA-QUERY PARALLELISM

Improve the performance of a single query by executing its operators in parallel.

Approach #1: Intra-Operator (Horizontal)

Approach #2: Inter-Operator (Vertical)

These techniques are not mutually exclusive.

There are parallel algorithms for every relational operator.

Intra-Operator

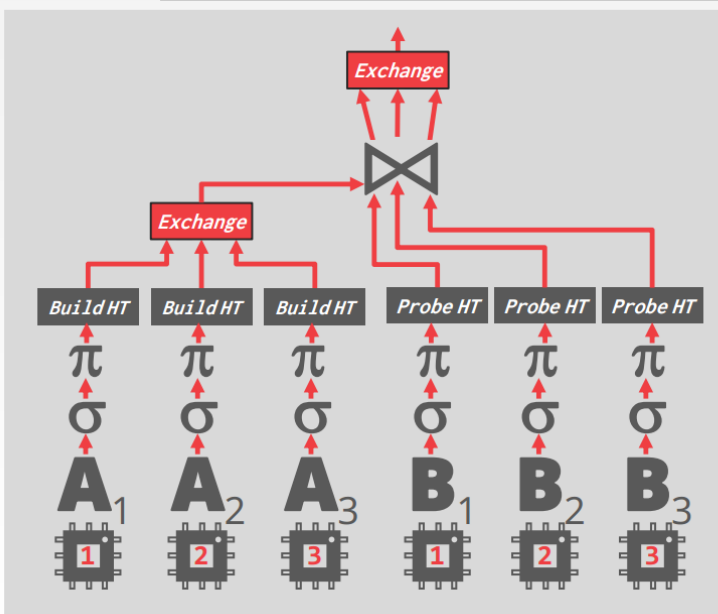
INTRA-OPERATOR PARALLELISM

Approach #1: Intra-Operator (Horizontal)

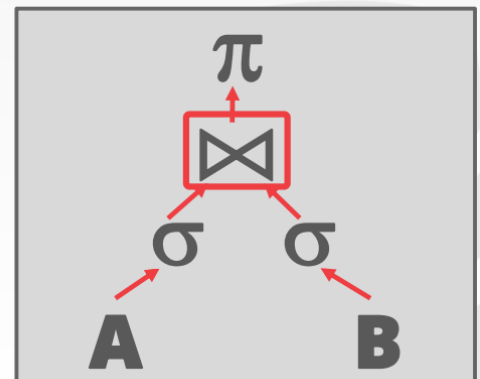
→ Operators are decomposed into independent instances that perform the same function on different subsets of data.

The DBMS inserts an **exchange** operator into the query plan to coalesce results from children operators.

INTRA-OPERATOR PARALLELISM



```
SELECT A.id, B.value
FROM A JOIN B
ON A.id = B.id
WHERE A.value < 99
AND B.value > 100
```



如上图，一个operator的任务可以有多个线程并发执行，DBMS使用 exchange operator

INTER-OPERATOR

流水线化的并行

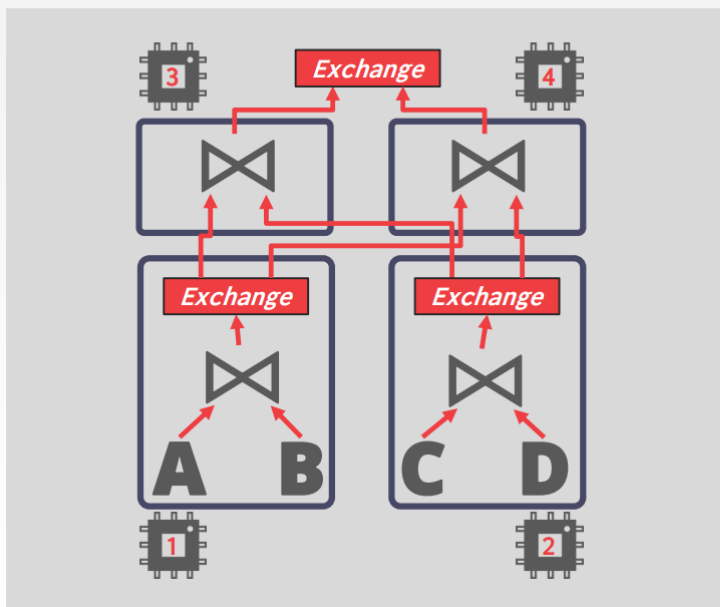
INTER-OPERATOR PARALLELISM

Approach #2: Inter-Operator (Vertical)

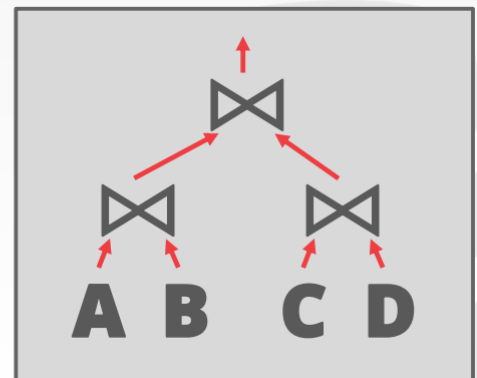
- Operations are overlapped in order to pipeline data from one stage to the next without materialization.
- Workers execute multiple operators from different segments of a query plan at the same time.
- Still need exchange operators to combine intermediate results from segments.

Also called pipelined parallelism.

INTRA-OPERATOR PARALLELISM



```
SELECT *  
FROM A  
JOIN B  
JOIN C  
JOIN D
```



那如何给operator分配worker数量?

WORKER ALLOCATION

WORKER ALLOCATION

Approach #1: One Worker per Core

- Each core is assigned one thread that is pinned to that core in the OS.
- See [sched_setaffinity](#)

Approach #2: Multiple Workers per Core

- Use a pool of workers per core (or per socket).
- Allows CPU cores to be fully utilized in case one worker at a core blocks.

One Worker per Core

Multiple Workers per Core

TASK ASSIGNMENT

Approach #1: Push

- A centralized dispatcher assigns tasks to workers and monitors their progress.
- When the worker notifies the dispatcher that it is finished, it is given a new task.

Approach #2: Pull

- Workers pull the next task from a queue, process it, and then return to get the next task.

总结而言对于OLAP系统来说，vectorized/bottom-up的执行策略是最佳的。