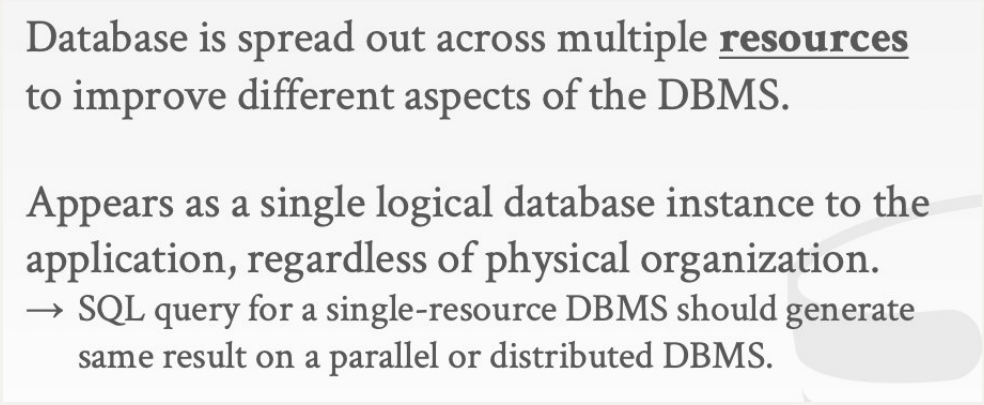
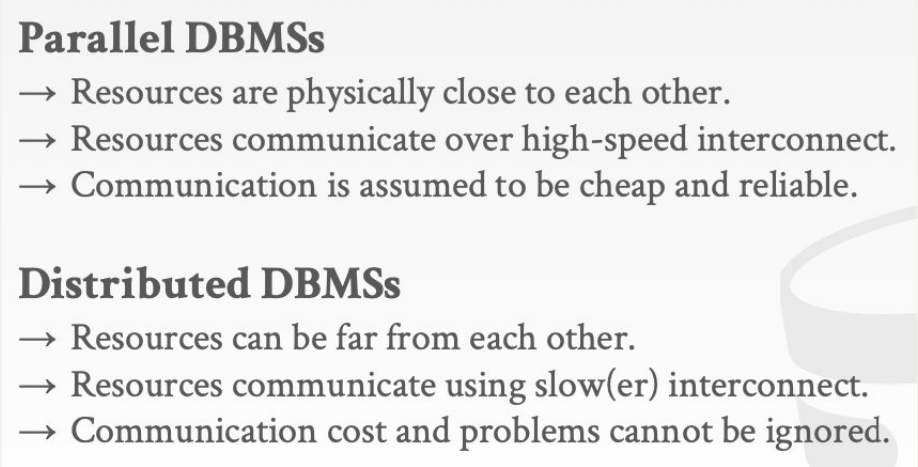
本Lec关于query的并发执行

查询语句的并行执行会提升DBMS的吞吐量（DBMS在单位时间内能执行完多少SQL语句），以及单条SQL语句的延迟（即单条语句的执行时间），从而给上层业务提供更好的服务

并行DBMS与分布式DBMS的共同点如下：



它们的区别如下：

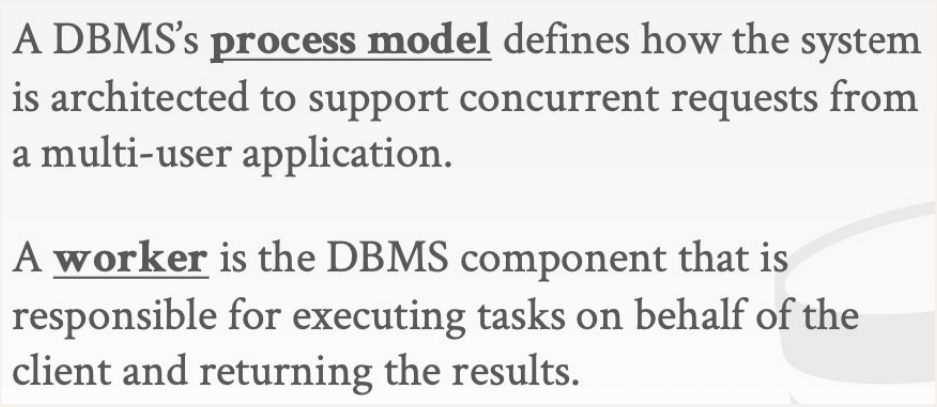


- 单节点并行的DBMS：资源都是在一块的（比如说多个线程都在同一个机器里面），资源之间的通信速度很快，简易且可靠（比如说多线程通过共享内存通信）
- 分布式DBMS：节点之间有可能间隔的很远（在世界上两个角落），节点之间的通信依托于网络而非内存，并且节点之间通信的问题与代价无法被回避

本Lec将探讨单节点上并行运行的DBMS的机制

Process Models

DBMS的process model描述的是多个用户的查询请求是如何并发执行的，一个worker可以理解为数据库的一个组件，用来执行一部分工作，比如说查询语句相应的执行计划可以被分解为多个部分，每个部分给一个worker去执行

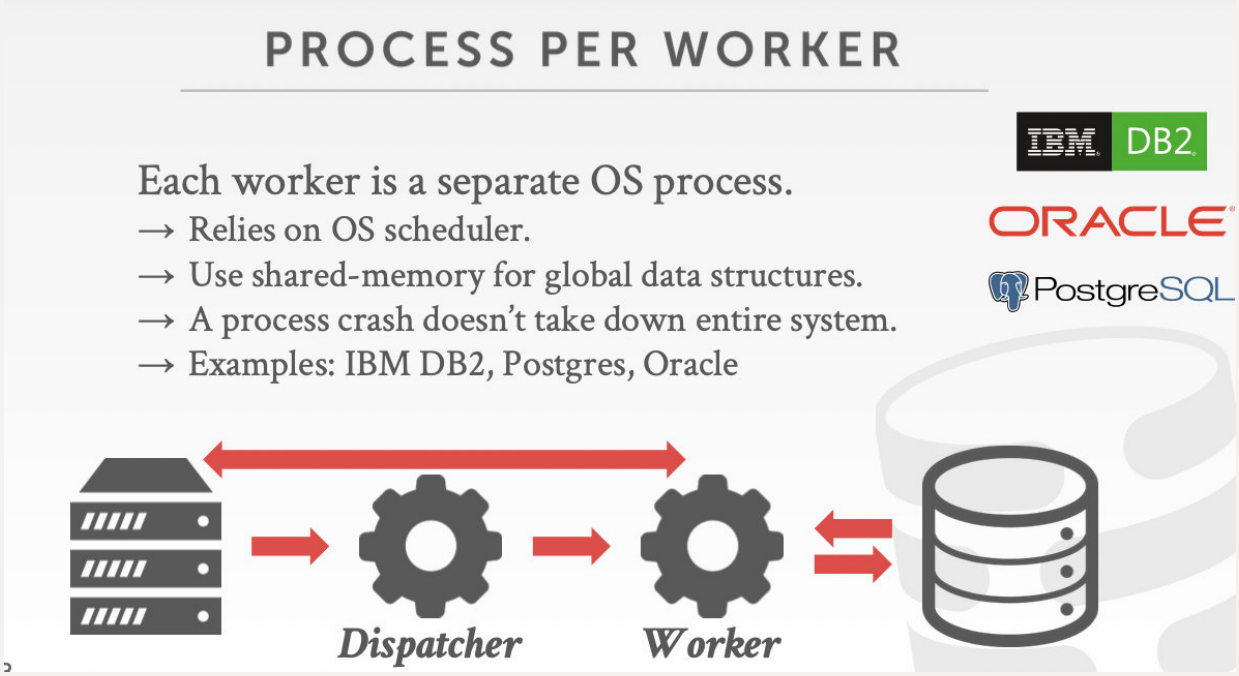


process model有三大方向：

1. Process per DBMS Worker，每个worker一个进程

   每个worker给它分配一个OS级别的进程去做，这依赖于操作系统对该进程的调度，两个worker之间可以通过共享内存的方式进行通信，这种模型有一个优点：单个进程的crash不会让整个系统宕机。比较古老的DBMS通常会采取这种模型。

   这种模型的工作流程如下：业务应用/客户端的请求传过来之后，分配器会把请求切成多份工作，每份分配给一个worker，每个worker负责对数据进行操作，完成工作之后把结果返回给客户端

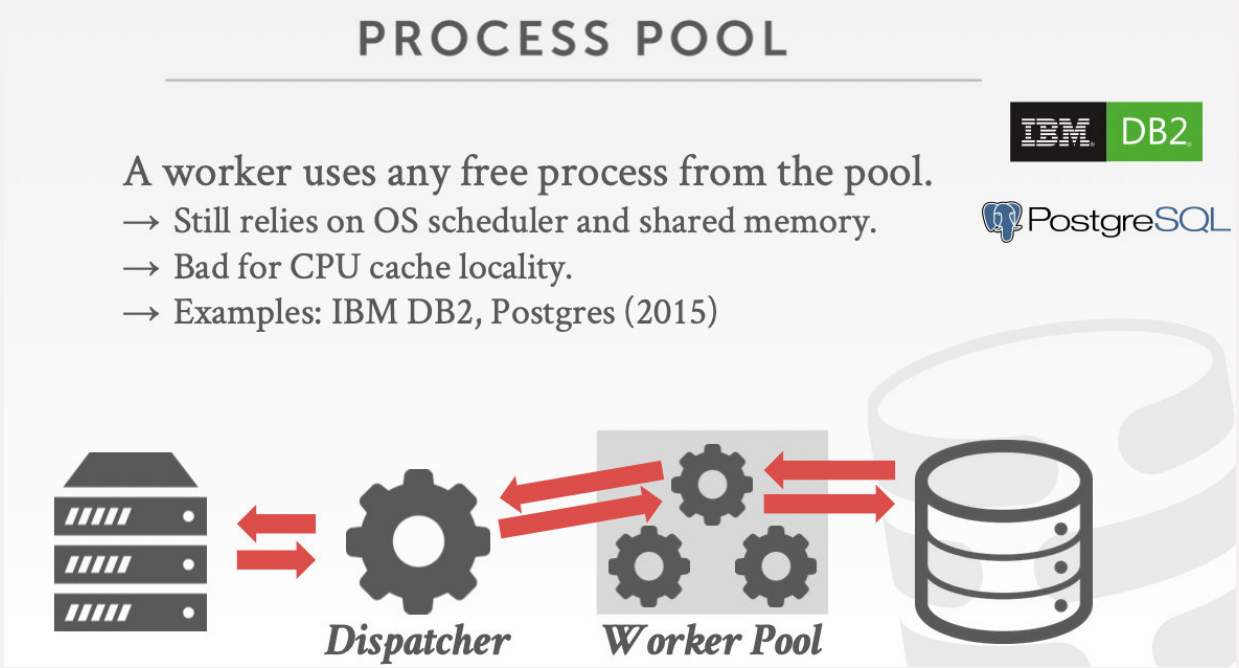


这种模型有一个问题，如果并发的量太大，那么操作系统上会有很多的进程，每个进程都会占用一定的计算机资源，调度它们也会带来不小的开销

2. Process pool，进程池

   Process per DBMS Worker模型下，由于系统中可能会有过多的进程从而降低性能，因此有了池化的思想：每个 Worker 可以使用 Worker Pool 中任意空闲的进程，其调度依然依赖于操作系统的调度器。当请求到达时，分配器会在进程池中寻找可用的进程，把工作分配给它。用户请求通过Dispatcher分配相应的 Worker 来完成查询，并将结果返回给Dispatcher，后者再返回给用户

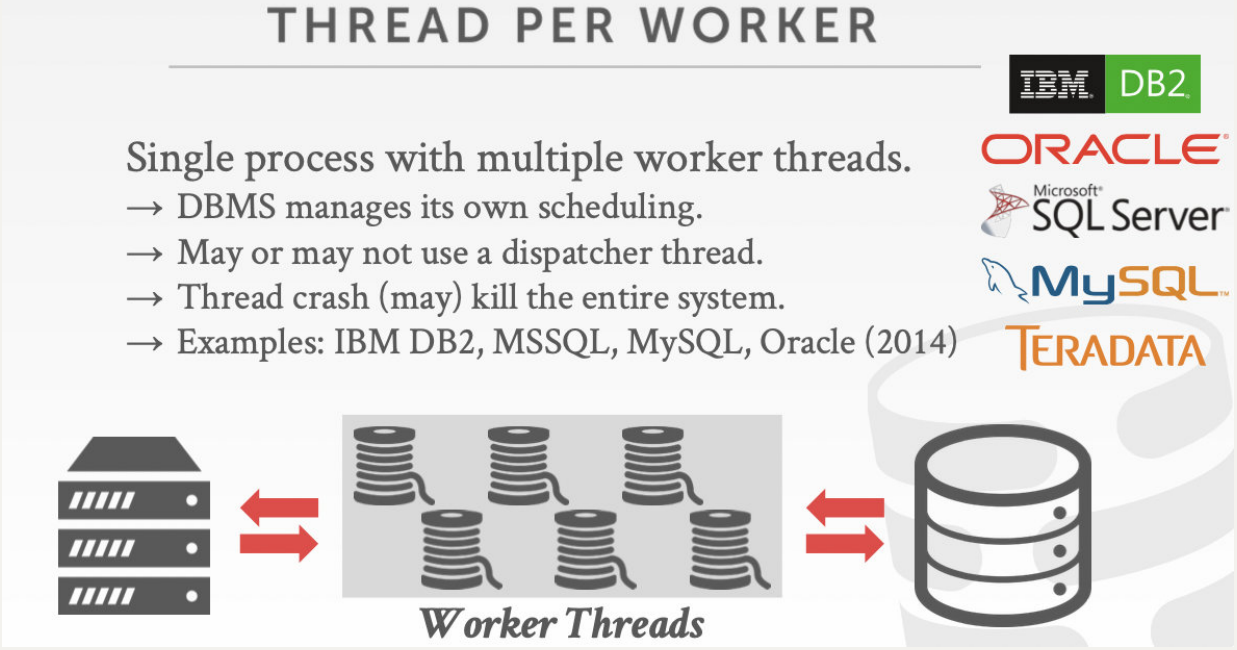
   这有利于控制进程的数量，减少对系统资源的占用



3. Thread per DBMS Worker，每个worker一个线程

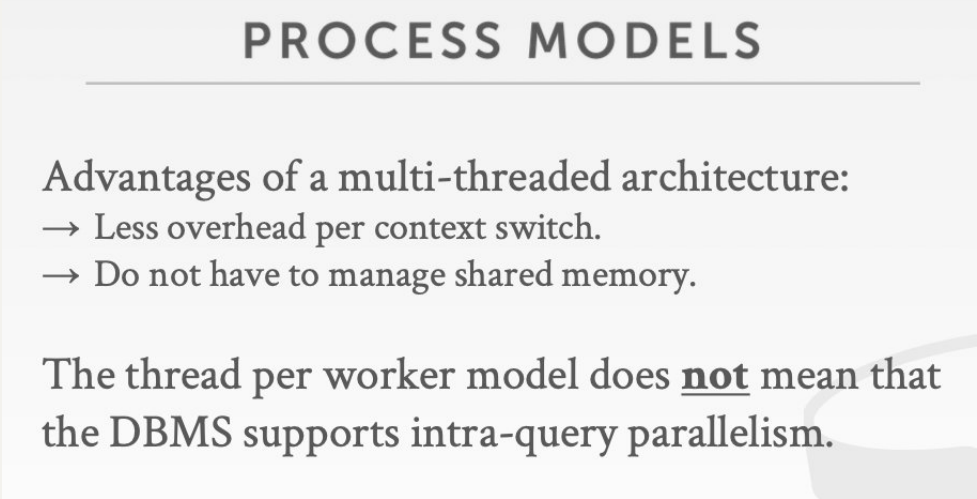
随着pthread库这种跨平台的创建与管理线程的接口的诞生，现代DBMS便开始使用thread per worker这种process model，每个worker对应一个线程，DBMS只由一个进程组成，但会分成多个worker线程去运行，线程之间的调度由DBMS负责

这个线程模型也存在一些问题：一个线程的崩溃会导致整个进程进而崩溃，整个DBMS也挂了，因此它的稳定性可能就比如前面process per worker的进程模型



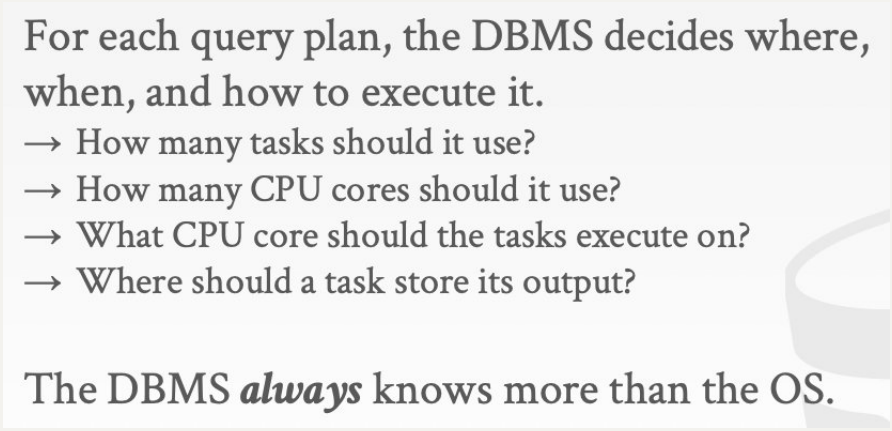
线程模型的优点也很明显：在一个进程内部的线程之间的切换带来的开销要比进程之间的切换的开销小得多，而且同一个进程的多个线程是天然地共享内存的（因为使用的是相同的页表），不需要像进程间通信那样人为地去管理共享内存

DBMS的多线程模型并不是表明每个SQL语句的执行都是并发的，而是说会同时有多个线程并发地去执行多个SQL语句

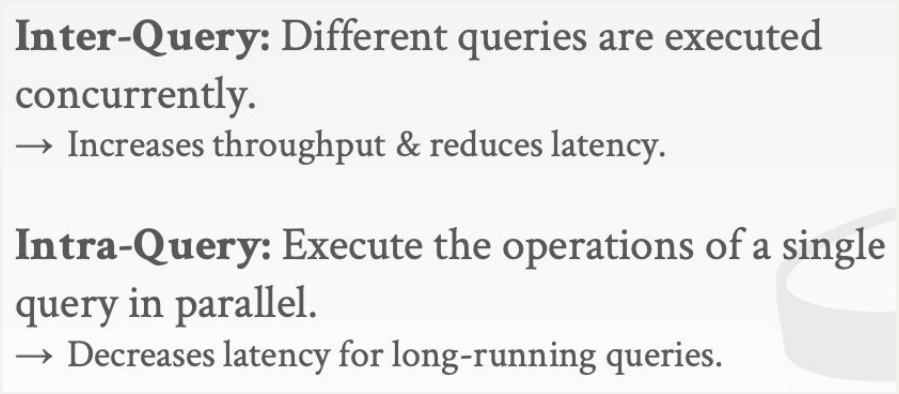


Execution Parallelism

DBMS对于查询计划的规划包含如下的内容



Inter-Query指的是查询之间的并发处理，Intra-Query指的是查询内部的并发处理，它们各自的优点如下：



首先介绍多个SQL语句之间的并发执行，即Inter-Query：

如果所有的查询都是read-only的，那就不容易产生并发导致的数据竞争，线程间的同步操作不多；但如果并发的查询都在更新数据库，那么这就是另一大问题，涉及到事务，并发控制，会在后面的Lecture介绍

其次是Intra-Query，单个查询内部的并行：

单个查询内部的并行执行会提升查询的效率，它的设计思想类似于生产者-消费者模型，成熟的DBMS中，每个算子都有并发的版本（比如说并发的table reader和hash join），并发算子的实现有两大思路：第一个思路是多个线程去操纵集中的全部数据（比如说多个线程同时读一个表），第二个思路是把集中的数据切开，把每部分分给相应的线程，使得线程可以

在本地处理

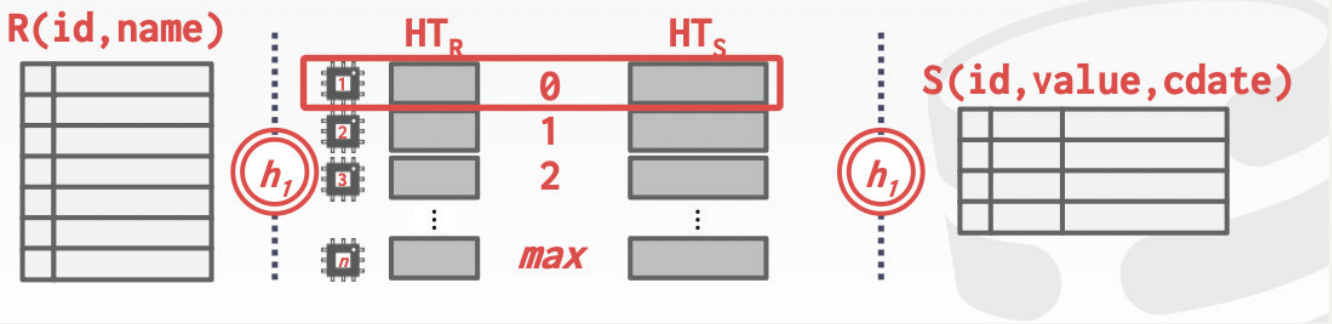
Improve the performance of a single query by executing its operators in parallel.

Think of organization of operators in terms of a ***producer/consumer*** paradigm.

There are parallel versions of every operator.
→ Can either have multiple threads access centralized data structures or use partitioning to divide work up.

e.g. 并发版本的grace hash join中，可以让一个线程（worker）处理一对哈希桶的join，最后再把多个线程各自的处理结果聚集起来

Use a separate worker to perform the join for each level of buckets for **R** and **S** after partitioning.



Intra-Query有三种实现方法

1. 水平切分执行计划树

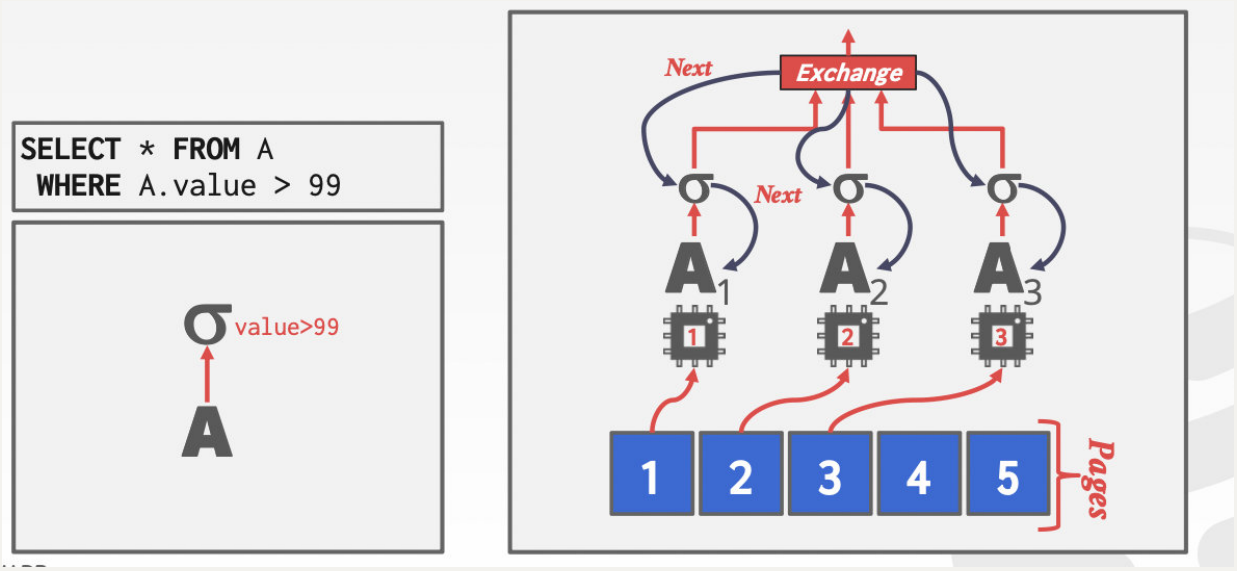
   把要处理的数据切开，每一部分给一个线程，且每个线程做的工作是一样的，DBMS还要再执行计划中插入一些exchange算子，用于做数据的聚集、拆分、重分布

Approach #1: Intra-Operator (Horizontal)

→ Decompose operators into independent **fragments** that perform the same function on different subsets of data.

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

如下所示，以筛选算子的执行为例，DBMS用exchange算子把查询计划切成三个路径，每个路径由一个worker/线程去负责，exchange算子负责去并发地调用这三个部分的`next`方法，从而达到并发地去读表的目的

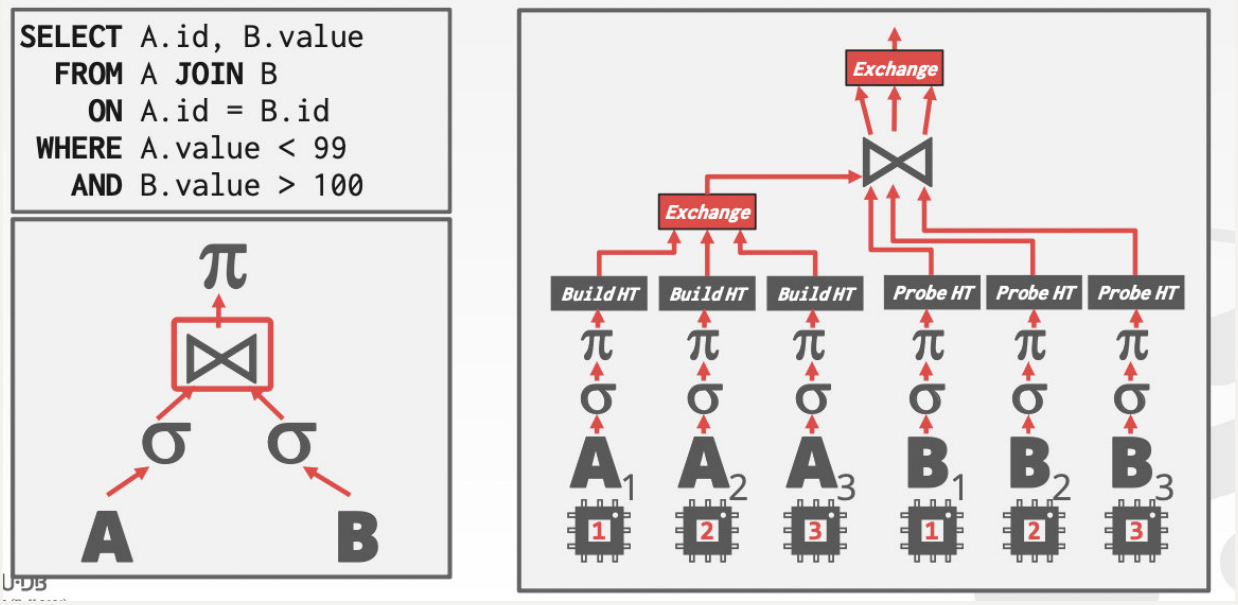


exchange算子上方可能会有其他的算子，比如说投影算子这种，它会一次一次地调用exchange算子的`next`方法，当exchange算子的`next`方法被调用时，exchange算子会并发地调用它的多个worker，然后将这些worker返回的数据聚集之后返回给上层的算子

exchange算子不仅存在于多线程的DBMS，一些分布式DBMS中也有，它包含三大类，如下所示

其中repartition是用于重分布的算子，它下面可能有3个worker，repartition算子可以把这三个worker汇聚之后再分成两个worker

下图是使用exchange算子的另外一个例子，做的是hash-join，其中涉及到了投影算子的下推



用三个线程并行地构建哈希表，之后用另外的三个线程去并行地进行点查询，并且用exchange算子把并行计算的结果汇聚起来

2. 垂直切分执行计划树

垂直切分是说把执行计划树中的每个算子丢给一个相应的线程/worker去执行，它们之间是并发执行的，同时算子之间也有数据的传递，这和现代处理器中的流水线很像

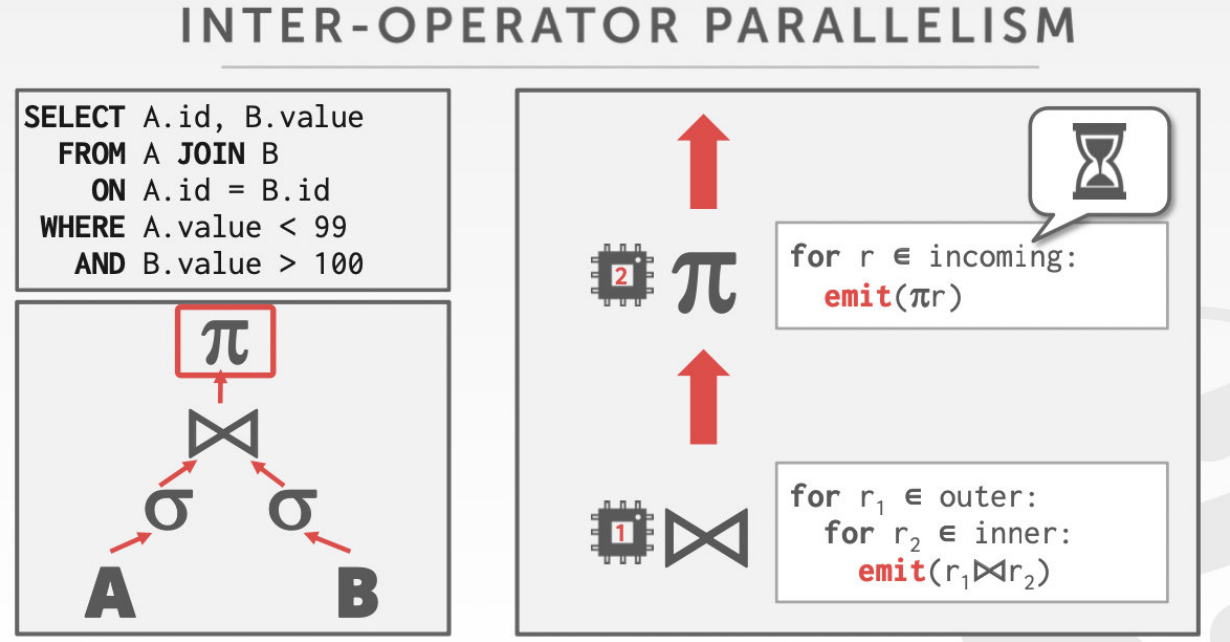
Approach #2: Inter-Operator (Vertical)

→ Operations are overlapped in order to pipeline data from one stage to the next without materialization.

→ Workers execute operators from different segments of a query plan at the same time.

Also called **pipeline parallelism**.

该策略的图解如下所示

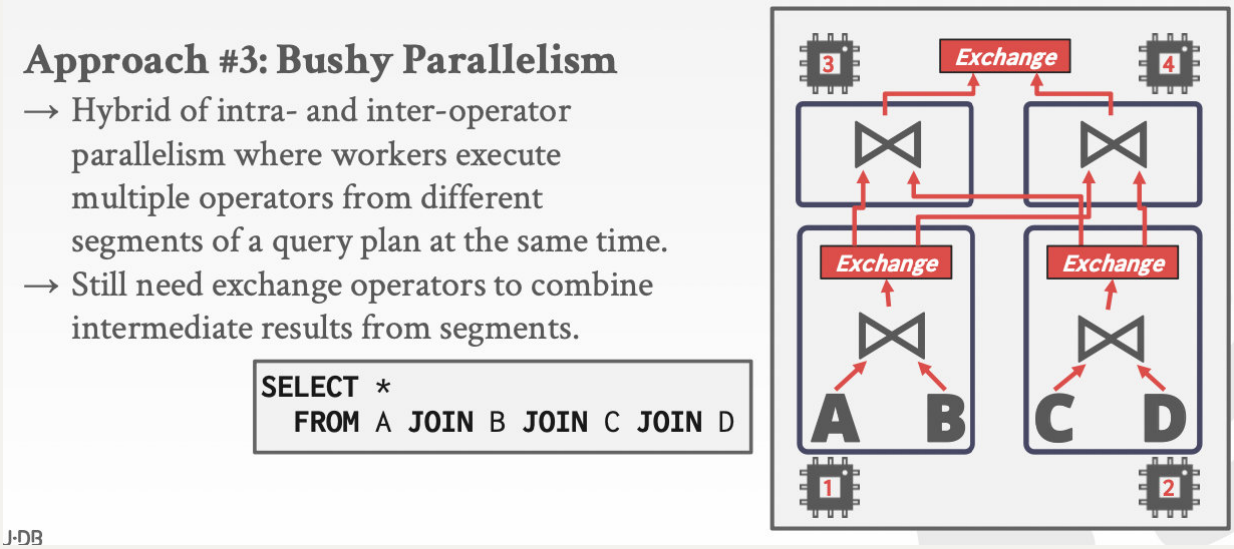


3. Bushy Parallelism，将前两种方法结合

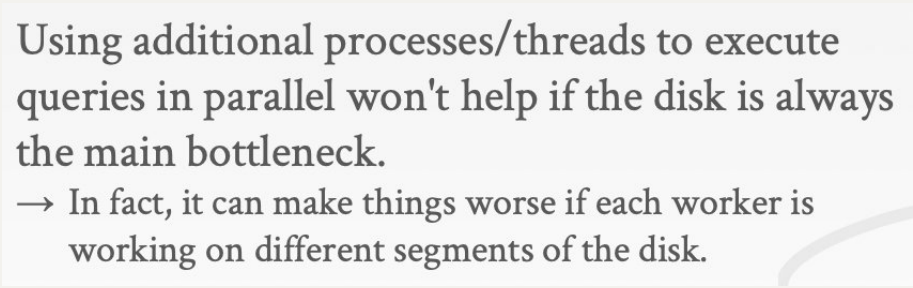
   既做到将算子切开并发执行，也做到算子和算子之间并发执行，并且依然需要exchange算子

   结合如下场景分析，要做A~D四个表的join，thread 1做A join B，thread 2做C join D，这便是算子之间的并行执行，属于对查询计划的垂直切分

   按理说应该把thread 1和thread 2的输出结果丢给一个join算子，但如下图所示，bushy模型下会将这个join算子的内部执行切开完成，也就是把中间结果切成两半，一半丢给thread 3，一半丢给thread 4，这两个线程的操作逻辑是一样的，就是负责的数据部分不同，这属于对查询计划的水平切分，并且其中的切分和汇聚都依赖exchange算子来完成

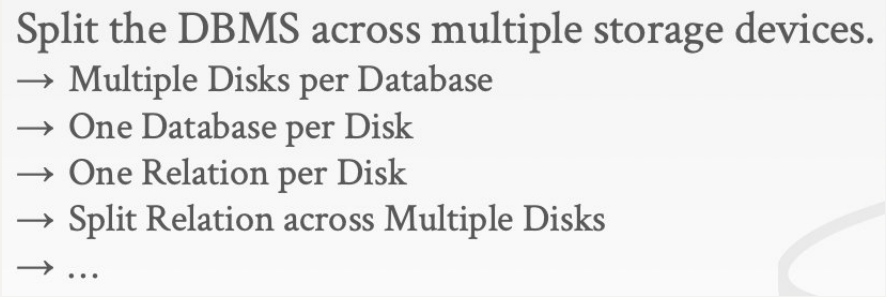


15-445针对的是基于硬盘的DBMS，因此大部分情况下硬盘I/O是性能瓶颈，前面所介绍的基于并行的优化策略也都是基于“数据已经被读到了内存中”的假设，如果数据无法及时从硬盘读入内存，再多的优化也是作用有限的。并且如果每个worker在尝试读硬盘的不同的部分的话，事情会变得更糟糕，因为这会导致对硬盘的频繁的随机访问，因此接下来会分析对硬盘I/O的性能优化



I/O Parallelism

如果把数据库的数据切成不同的部分，存到不同的存储设备中，这样在对数据库进行并发存取时就会做到在硬件层面上隔离开，从而通过对多个硬盘的并发读写提高DBMS的性能。



实现I/O并行有如下具体途径：

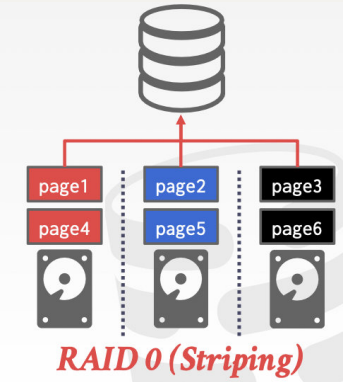
Multi-Disk Parallelism

多磁盘并行对于DBMS是透明的，它会通过对 OS 或硬件的配置将 DBMS 的数据文件存储到多个存储设备上

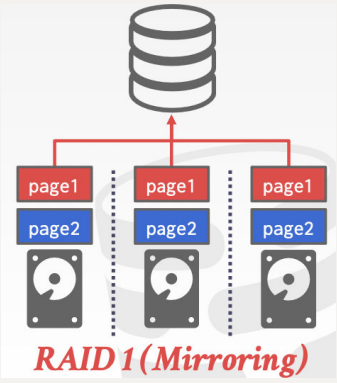
Configure OS/hardware to store the DBMS's files across multiple storage devices.
→ Storage Appliances
→ RAID Configuration

This is **transparent** to the DBMS.

比如说它可以借助存储硬件的RAID技术，RAID 0机制会将文件分成多份存入不同的磁盘，这样在读写文件时可以进行并行的读写



RAID 1机制会将每个硬盘做成彼此的镜像，这样可以避免由于单个硬盘的损坏导致数据的丢失，同时也可以在读文件数据的时候并行地从每个硬盘读取文件的不同部分，提升效率

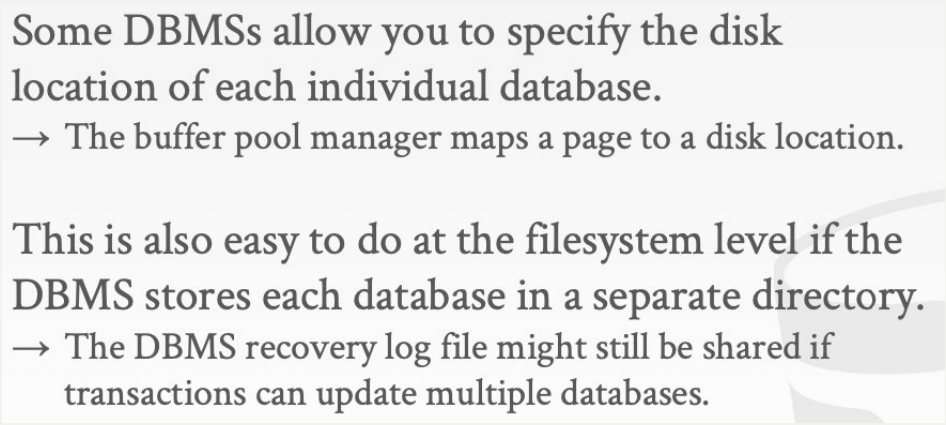


此外RAID还有其他的等级，比如说RAID 5中可以把文件数据拆分到两个硬盘存储，第三块硬盘存储的是前两块硬盘里的数据按位XOR的结果，这样的话如果前两块盘里有一个损坏，可以通过第三块盘来恢复数据，这就结合了RAID 0和RAID 1的优点

Database Partitioning

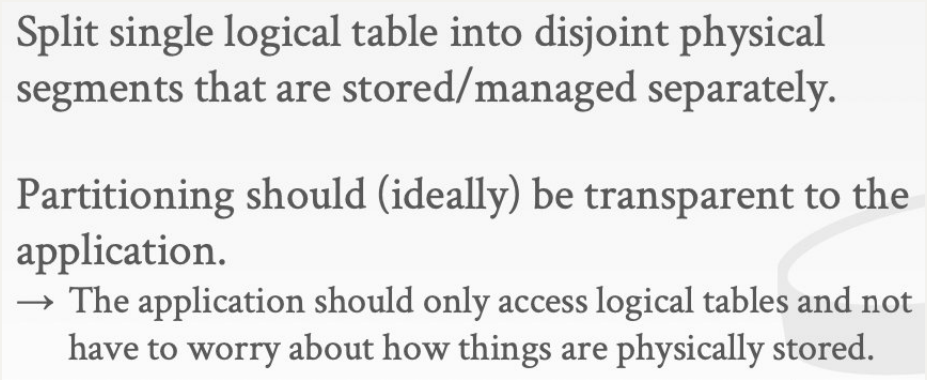
这是关于将DBMS所存储的数据切分成多个部分的策略

分库



我们可以把数据库实例中的不同Database分配到不同的硬盘当中，还可以在文件系统层面将不同的Database存入不同的文件夹中

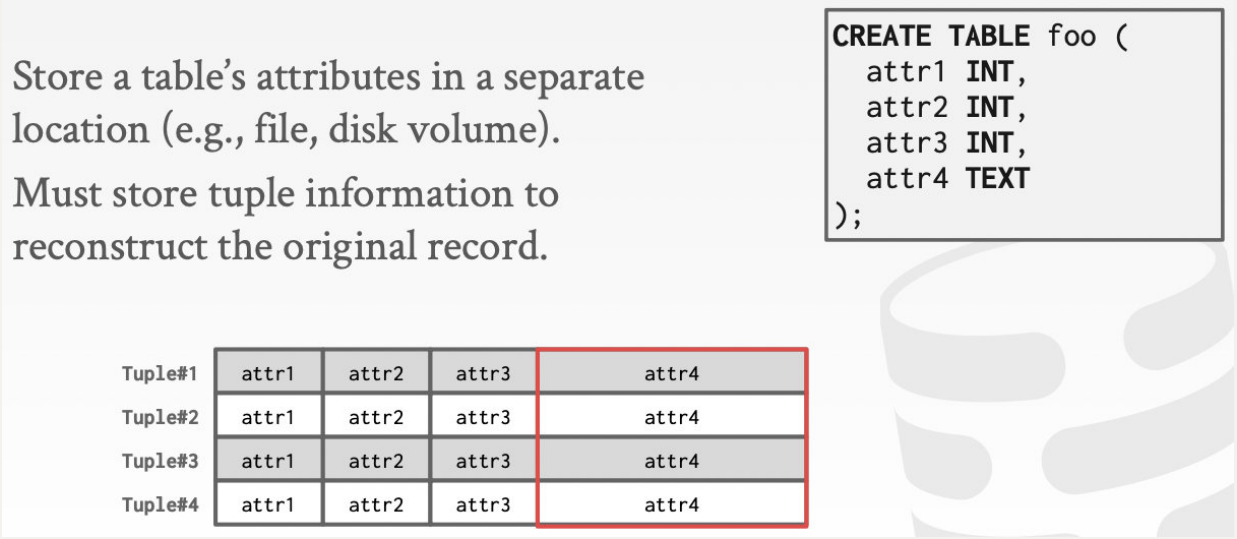
分区



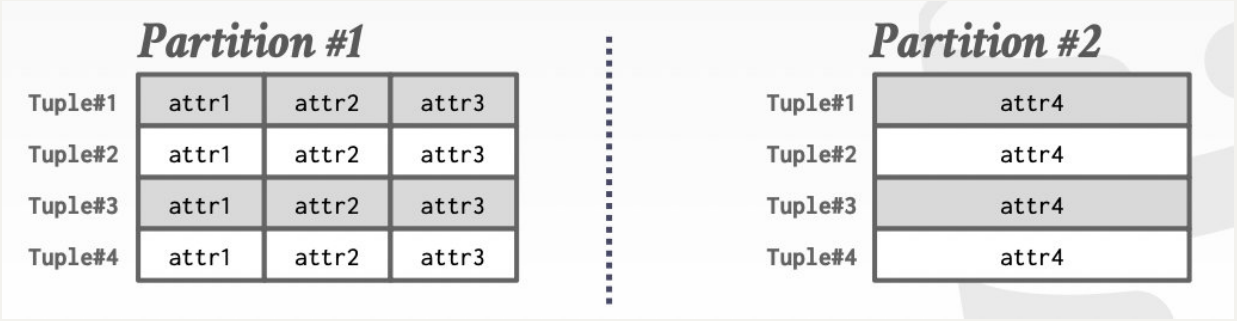
我们可以对单个的表进行分区，在物理上将它分成多个部分，每个部分落在不同的盘上，从而提升I/O的性能。对表进行分区的方法有如下几种：

- Vertical Partition

  垂直分区，如下所示：



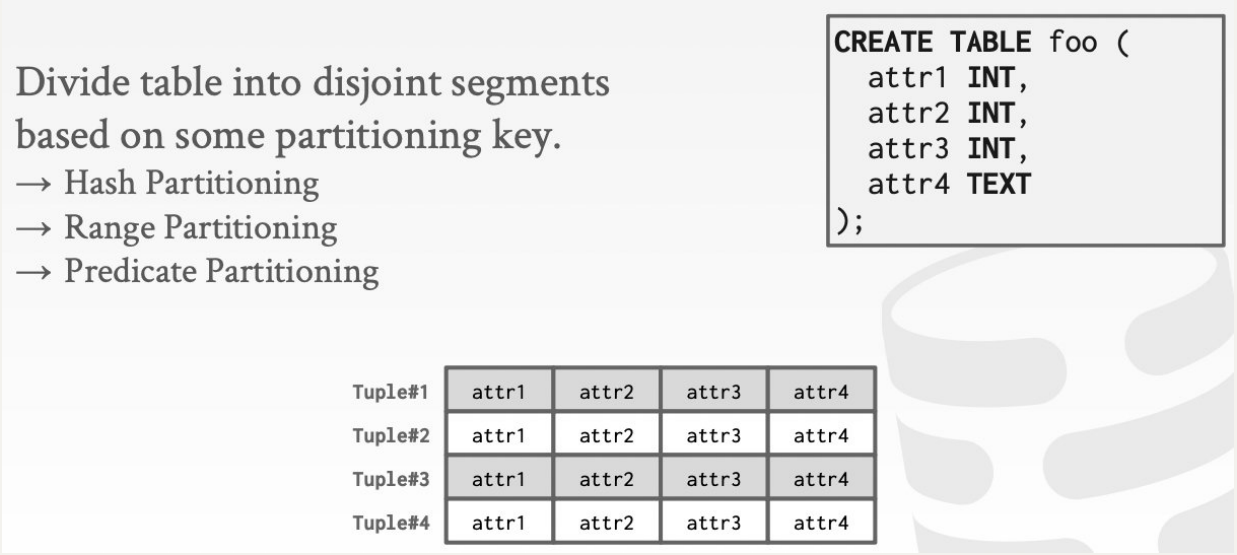
->next->



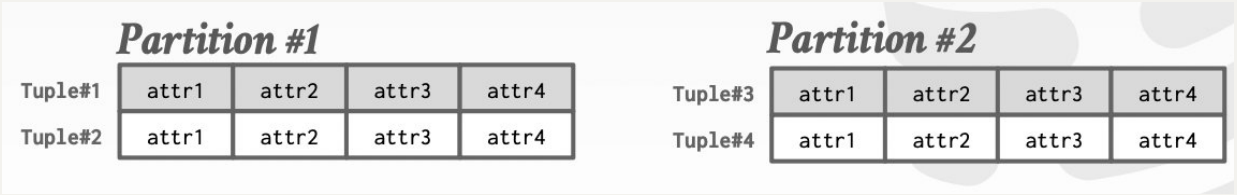
而且如果表中有几个不常用的属性，而且这些属性占用的存储空间特别的大（比如上图中的attr4），就可以使用这种垂直分区策略

- Horizontal Partitioning

  水平分区，如下所示：



->next->



Ref/参考自 :

https://www.bilibili.com/video/BV1Z3411x7aS/?spm_id_from=333.788

https://zhuanlan.zhihu.com/p/418937203