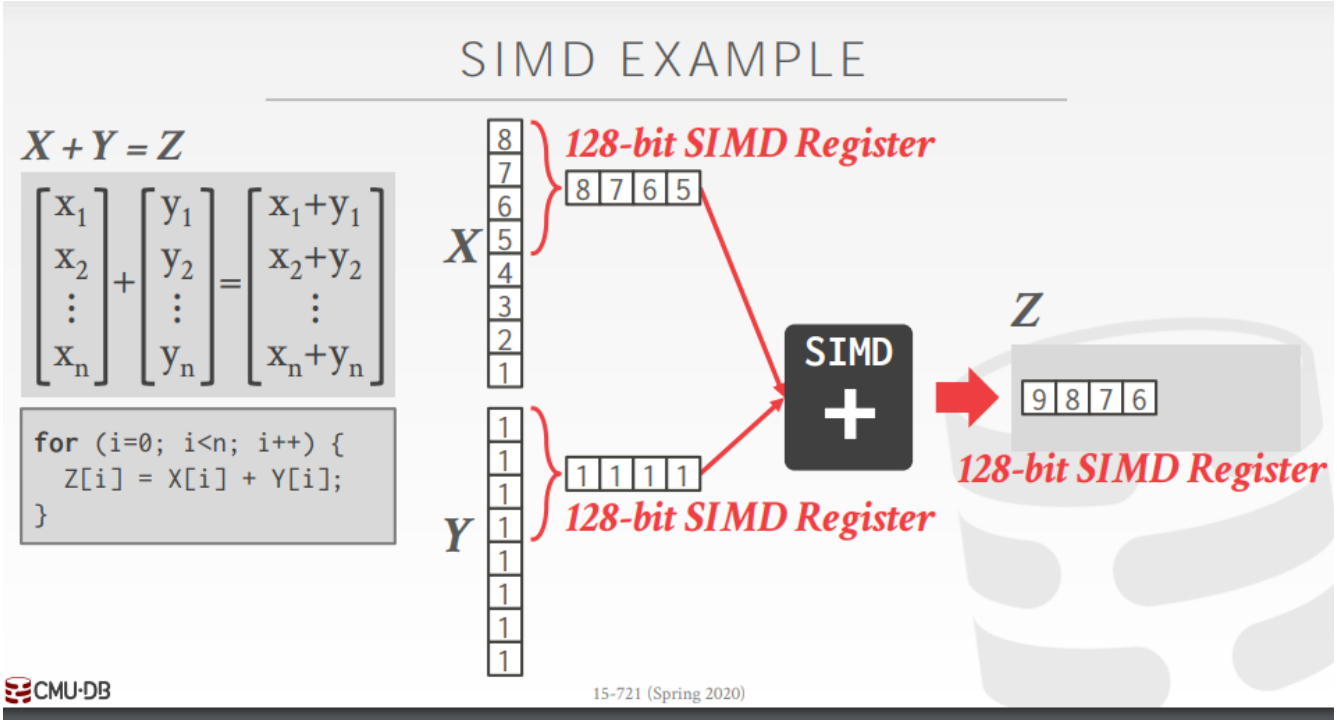
SIMD

Single Instruction Multiple Data

以加法指令为例，单指令单数据（SISD）的CPU对加法指令译码后，执行部件先访问内存，取得第一个操作数；之后再一次访问内存，取得第二个操作数；随后才能进行求和运算。而在SIMD型的CPU中，指令译码后几个执行部件同时访问内存，一次性获得所有操作数进行运算。这个特点使SIMD特别适合于多媒体应用等数据密集型运算。



我的理解是SIMD的大型寄存器允许单周期内把多个数加载到register内

但是往往需要人工来把算法改成SIMD模式

SIMD TRADE-OFFS

Advantages:

→ Significant performance gains and resource utilization if an algorithm can be vectorized.

Disadvantages:

→ Implementing an algorithm using SIMD is still mostly a manual process.
→ SIMD may have restrictions on data alignment.
→ Gathering data into SIMD registers and scattering it to the correct locations is tricky and/or inefficient.

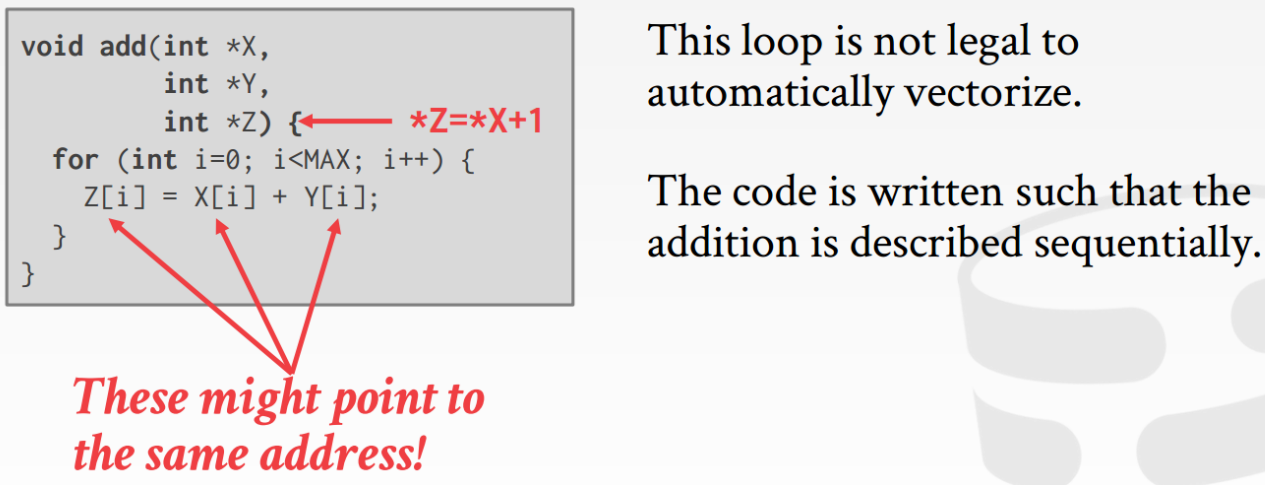
VECTORIZATION

- Automatic Vectorization

编译器来自动负责把循环内部的instruction重写为向量化版本。

但是只能适用于简单的循环，并且需要硬件支持

而且有时候vectorize也是不安全的:

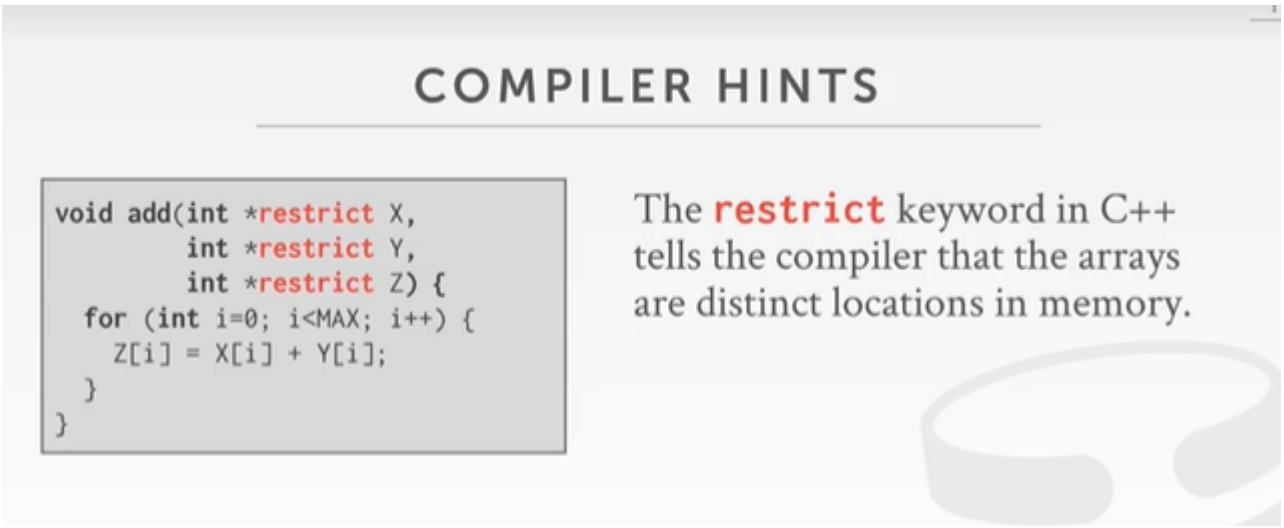


- Compiler Hints

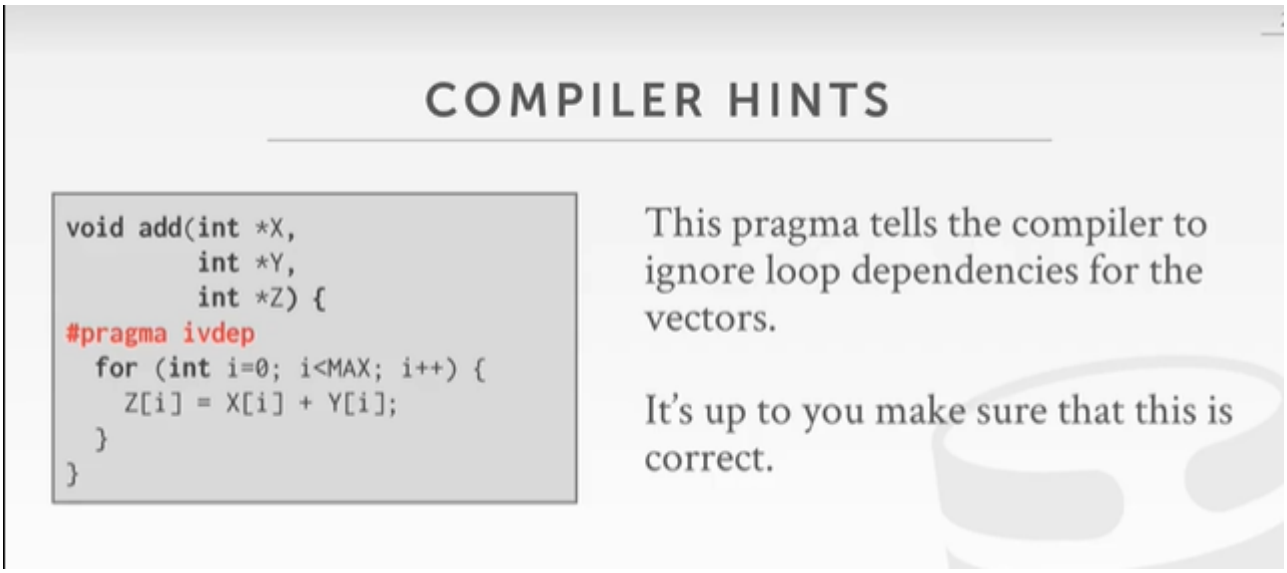
给编译器提供额外信息，告知什么时候才能安全地向量化

→ Give explicit information about memory locations.
→ Tell the compiler to ignore vector dependencies.

方法一：



方法二：



- Explicit Vectorization

EXPLICIT VECTORIZATION

Use CPU intrinsics to manually marshal data between SIMD registers and execute vectorized instructions.

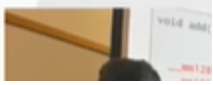
Potentially not portable.

EXPLICIT VECTORIZATION

```
void add(int *X,
         int *Y,
         int *Z) {
  __m128i *vecX = (__m128i*)X;
  __m128i *vecY = (__m128i*)Y;
  __m128i *vecZ = (__m128i*)Z;
  for (int i=0; i<MAX/4; i++) {
    _mm_store_si128(vecZ++,
      _mm_add_epi32(*vecX++,
                    *vecY++));
  }
}
```

Store the vectors in 128-bit SIMD registers.

Then invoke the intrinsic to add together the vectors and write them to the output location.



Vectorization Direction

- Horizontal

Perform operation on all elements together within a single vector.

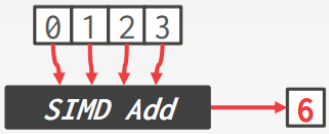
- Vertical

Perform operation in an elementwise manner on elements of each vector

VECTORIZATION DIRECTION

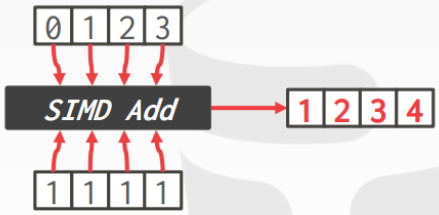
Approach #1: Horizontal

→ Perform operation on all elements together within a single vector.



Approach #2: Vertical

→ Perform operation in an elementwise manner on elements of each vector.



EXPLICIT VECTORIZATION METHOD

EXPLICIT VECTORIZATION

Linear Access Operators

→ Predicate evaluation
→ Compression

Ad-hoc Vectorization

→ Sorting
→ Merging

Composable Operations

→ Multi-way trees
→ Bucketized hash tables

接下来介绍向量化的一些算法

VECTORIZED DBMS ALGORITHMS

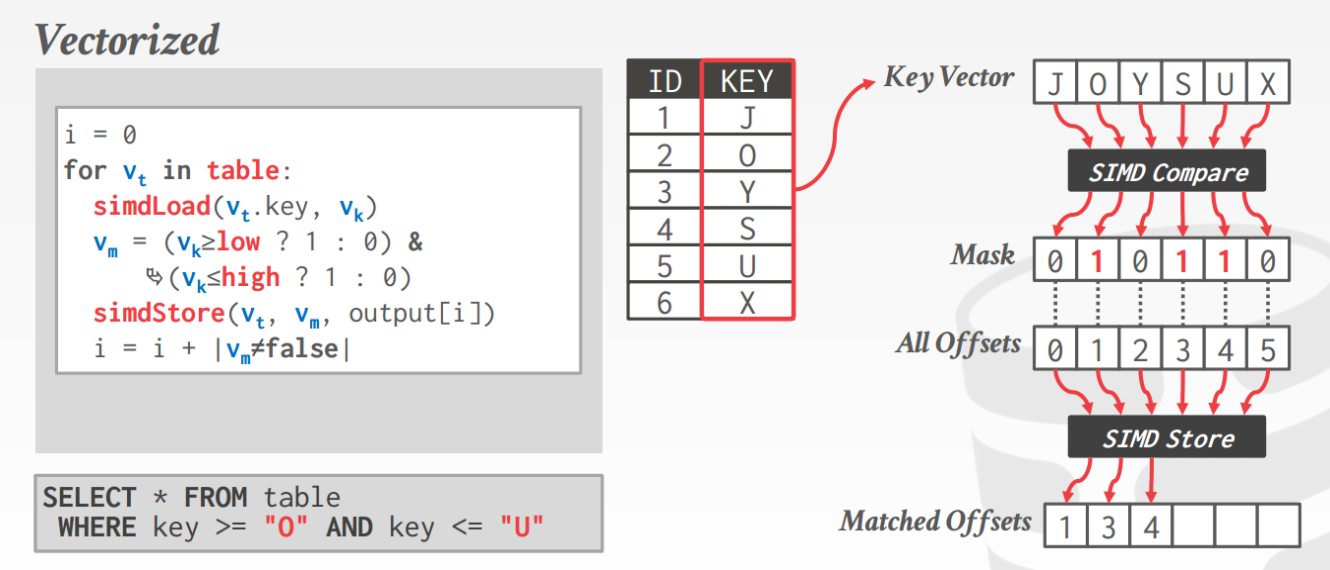
Principles for efficient vectorization by using fundamental vector operations to construct more advanced functionality.

→ Favor ***vertical*** vectorization by processing different input data per lane.

→ Maximize lane utilization by executing unique data items per lane subset (i.e., no useless computations).

VECTORIZED OPERATORS

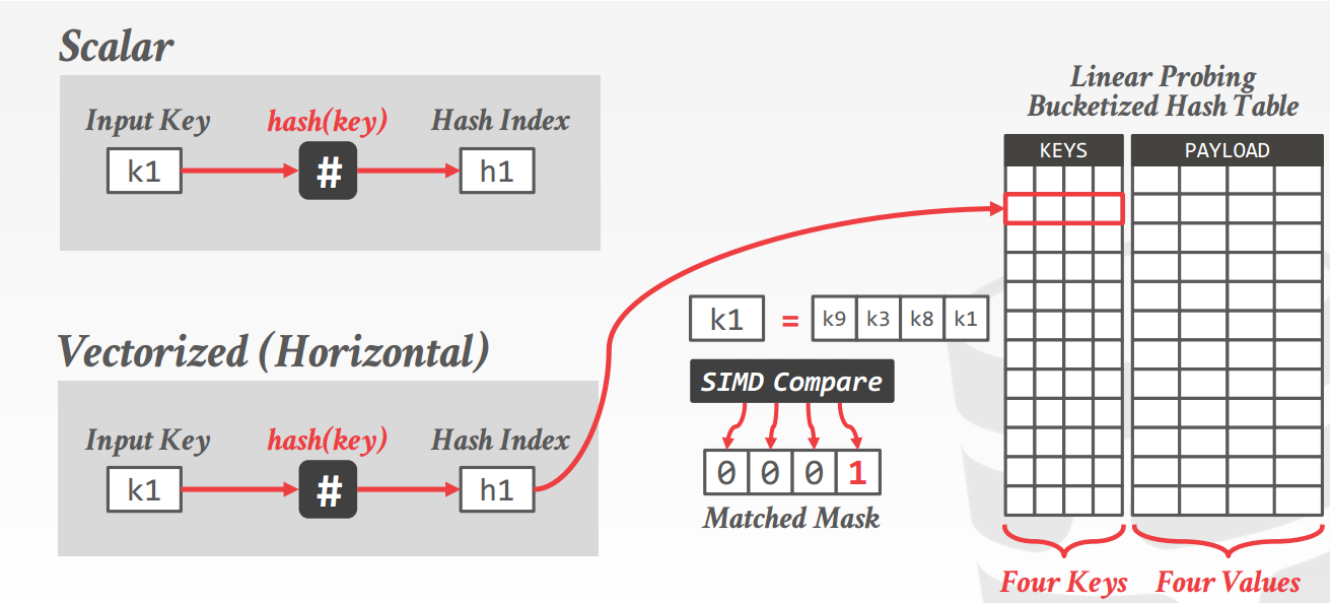
- Selection Scans



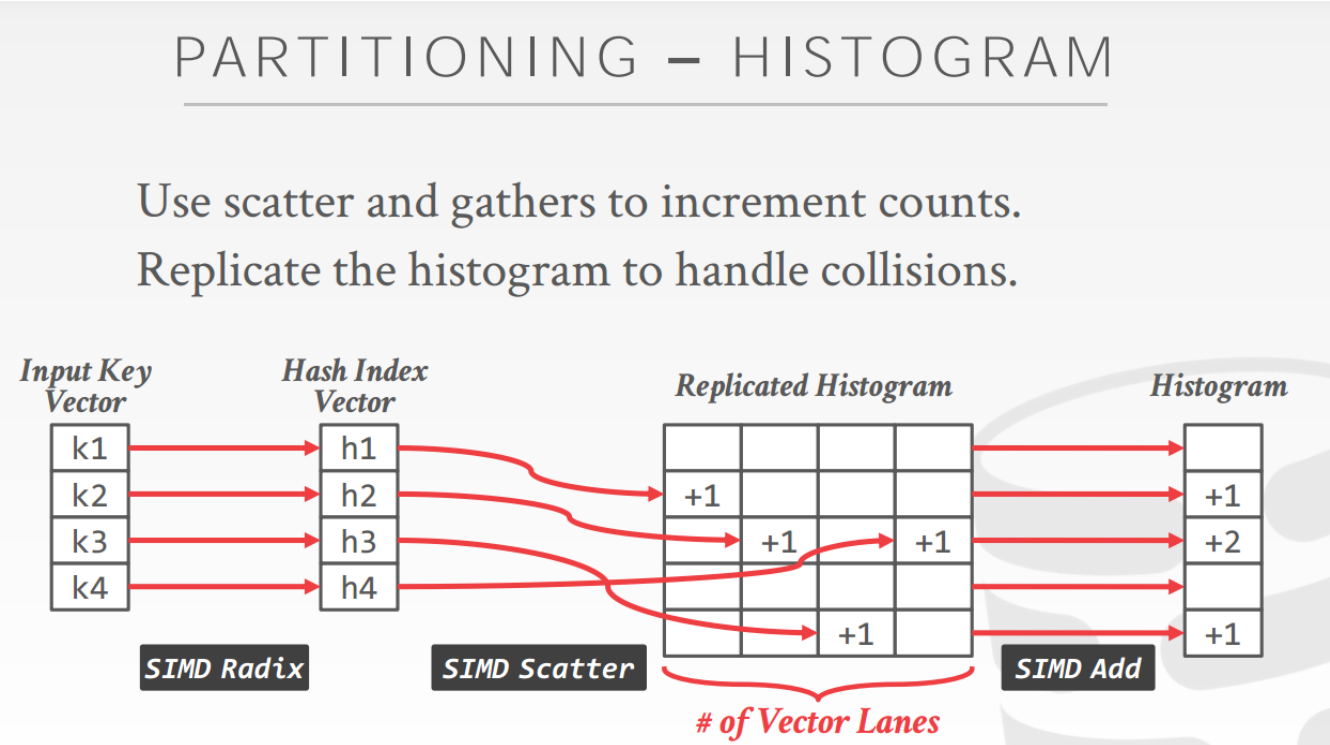
- Hash Tables

线性探测法解决哈希冲突的性能瓶颈在于:线性探测部分，即冲突之后在哈希表遍历的部分

向量化的线性探测：128-bit的寄存器允许一次性加载四个key/value,分摊掉原来线性探测的开销。



- Partitioning / Histograms



对于OLAP数据库来说，向量化执行是其提高性能的关键。

Vectorization is essential for OLAP queries.

These algorithms don't work when the data exceeds your CPU cache.

We can combine all the intra-query parallelism optimizations we've talked about in a DBMS.

→ Multiple threads processing the same query.
→ Each thread can execute a compiled plan.
→ The compiled plan can invoke vectorized operations.