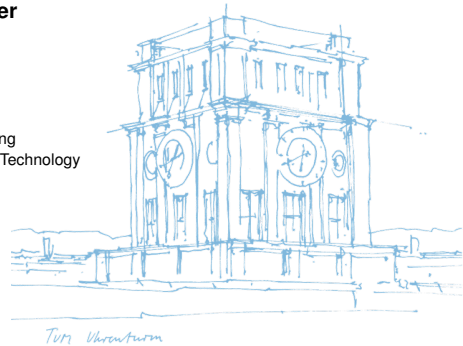
Database Systems on Modern CPU Architectures

Adrian Riedl, Stefan Lehner

Tuesday 25th June, 2024

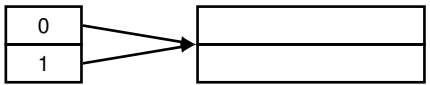
Chair of Data Science and Engineering
TUM School of Computation, Information and Technology
Technical University of Munich



Hashing

Insert the following tuples into an extendible hash table whose buckets can hold up to two entries.

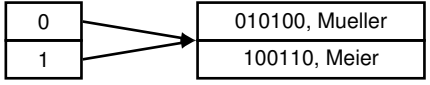
ID	Name	hash
10	Müller	010100
25	Meier	100110
30	Schmidt	011110
18	Krause	010010
40	Schulz	000101
45	Kaufmann	101101



Hashing

Insert the following tuples into an extendible hash table whose buckets can hold up to two entries.

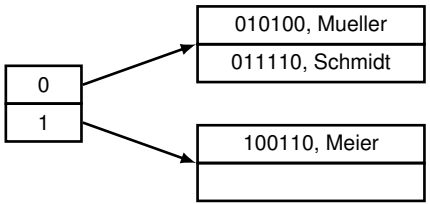
ID	Name	hash
10	Müller	010100
25	Meier	100110
30	Schmidt	011110
18	Krause	010010
40	Schulz	000101
45	Kaufmann	101101



Hashing

Insert the following tuples into an extendible hash table whose buckets can hold up to two entries.

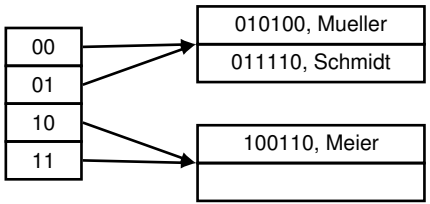
ID	Name	hash
10	Müller	010100
25	Meier	100110
30	Schmidt	011110
18	Krause	010010
40	Schulz	000101
45	Kaufmann	101101



Hashing

Insert the following tuples into an extendible hash table whose buckets can hold up to two entries.

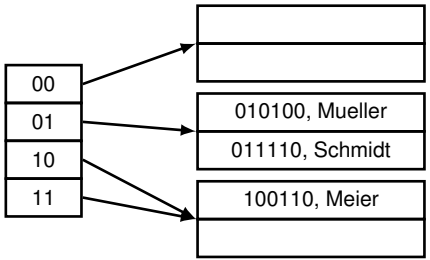
ID	Name	hash
10	Müller	010100
25	Meier	100110
30	Schmidt	011110
18	Krause	010010
40	Schulz	000101
45	Kaufmann	101101



Hashing

Insert the following tuples into an extendible hash table whose buckets can hold up to two entries.

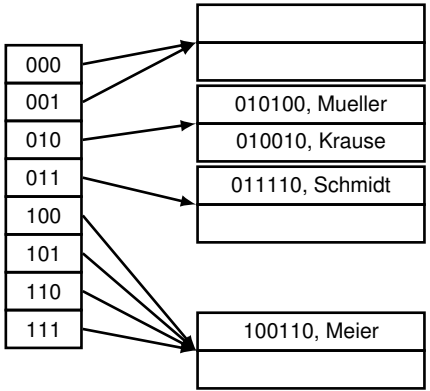
ID	Name	hash
10	Müller	010100
25	Meier	100110
30	Schmidt	011110
18	Krause	010010
40	Schulz	000101
45	Kaufmann	101101



Hashing

Insert the following tuples into an extendible hash table whose buckets can hold up to two entries.

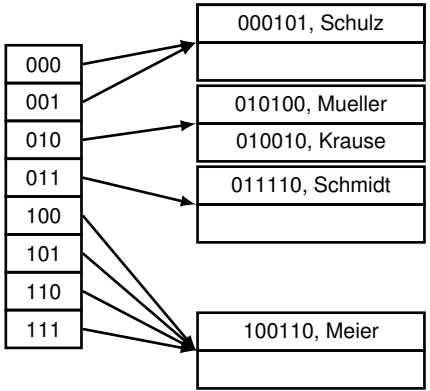
ID	Name	hash
10	Müller	010100
25	Meier	100110
30	Schmidt	011110
18	Krause	010010
40	Schulz	000101
45	Kaufmann	101101



Hashing

Insert the following tuples into an extendible hash table whose buckets can hold up to two entries.

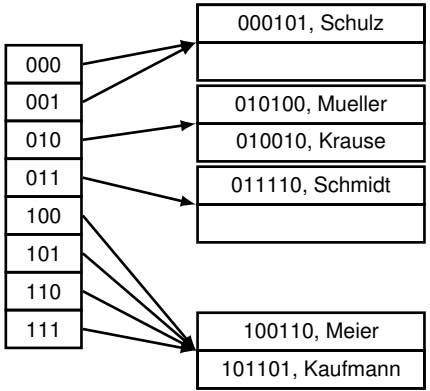
ID	Name	hash
10	Müller	010100
25	Meier	100110
30	Schmidt	011110
18	Krause	010010
40	Schulz	000101
45	Kaufmann	101101



Hashing

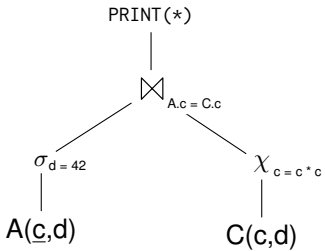
Insert the following tuples into an extendible hash table whose buckets can hold up to two entries.

ID	Name	hash
10	Müller	010100
25	Meier	100110
30	Schmidt	011110
18	Krause	010010
40	Schulz	000101
45	Kaufmann	101101



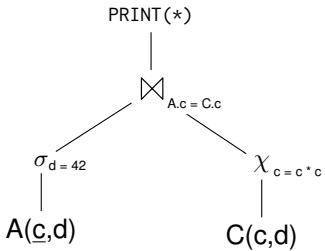
Data-Centric Query Execution

Generate (pseudo-)code for the following operator tree using the push model.



Exchange Parallelism

Parallelize the given query plan by introducing `XchgHashSplit(n:m)` and `Xchg(n:m)` operators for **2 threads** as necessary.



Recoverability

Which properties does this history fulfill?

$$w_1(x), r_2(y), w_3(y), w_2(x), w_3(z), c_3, w_1(z), c_2, c_1$$

LLVM

Reconstruct the arithmetic expression from the given LLVM function:

```
define i64 @meaningful_function_name(ptr %0) {
    entry:
    %1 = getelementptr double, ptr %0, i64 1
    %2 = load double, ptr %1, align 8
    %3 = fadd double 4.200000e+01, %2
    %4 = getelementptr double, ptr %0, i64 0
    %5 = load double, ptr %4, align 8
    %6 = fsub double 2.100000e+01, %5
    %7 = fsub double %3, %6
    %8 = fptosi double %7 to i64
    %9 = getelementptr i64, ptr %0, i64 2
    %10 = load i64, ptr %9, align 4
    %11 = add i64 %8, %10
    %12 = add i64 %11, 100
    ret i64 %12
}
```

Data-Centric Query Execution II

Draw the operator tree for the given code which was generated by the push model.

```
ht1 = Hashtable()
ht2 = Hashtable()

for c in C:
    if ht1.contains(c.x)
        ht1[c.x] += c.d
    else:
        ht1[c.x] = c.d

for (key, value) in ht1.entries()
    ht2.insert(value, key)

for a in A:
    if a.d == 42:
        if ht2.contains(a.c)
            c = ht2.lookup(a.c)
            print(...)
```

Questions?