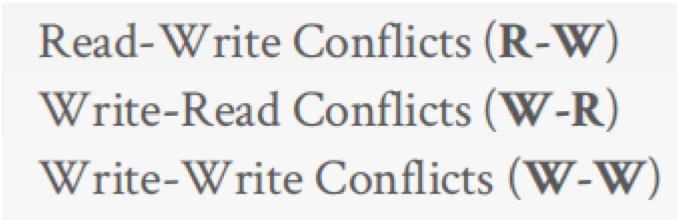
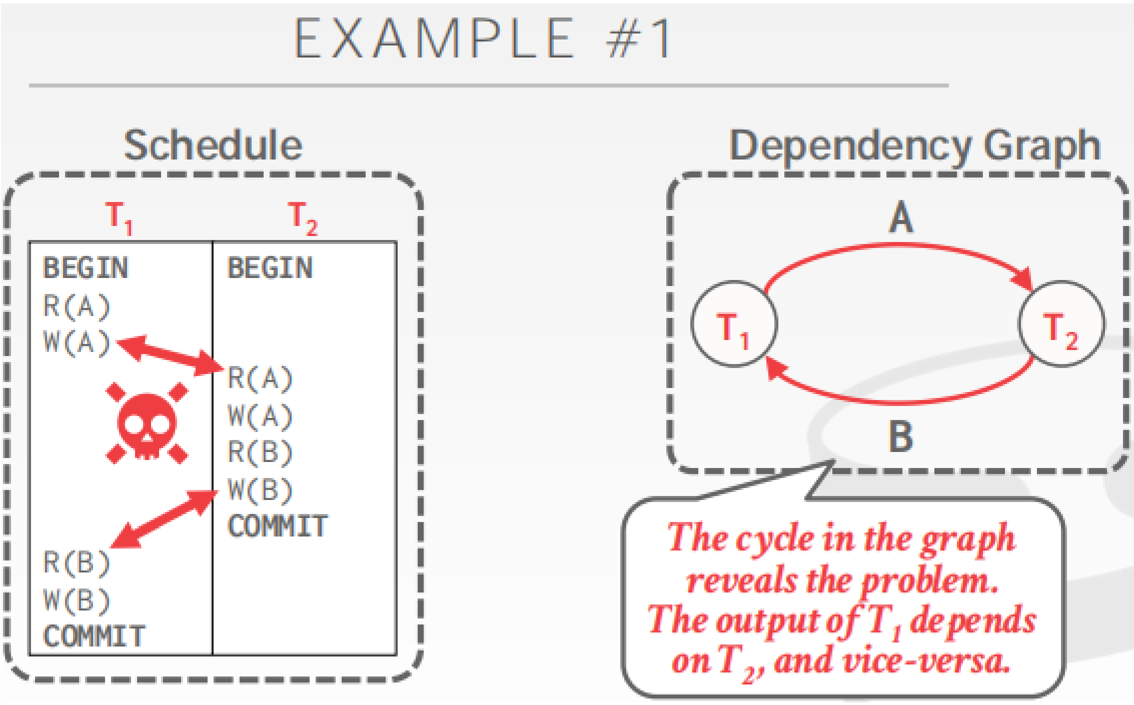
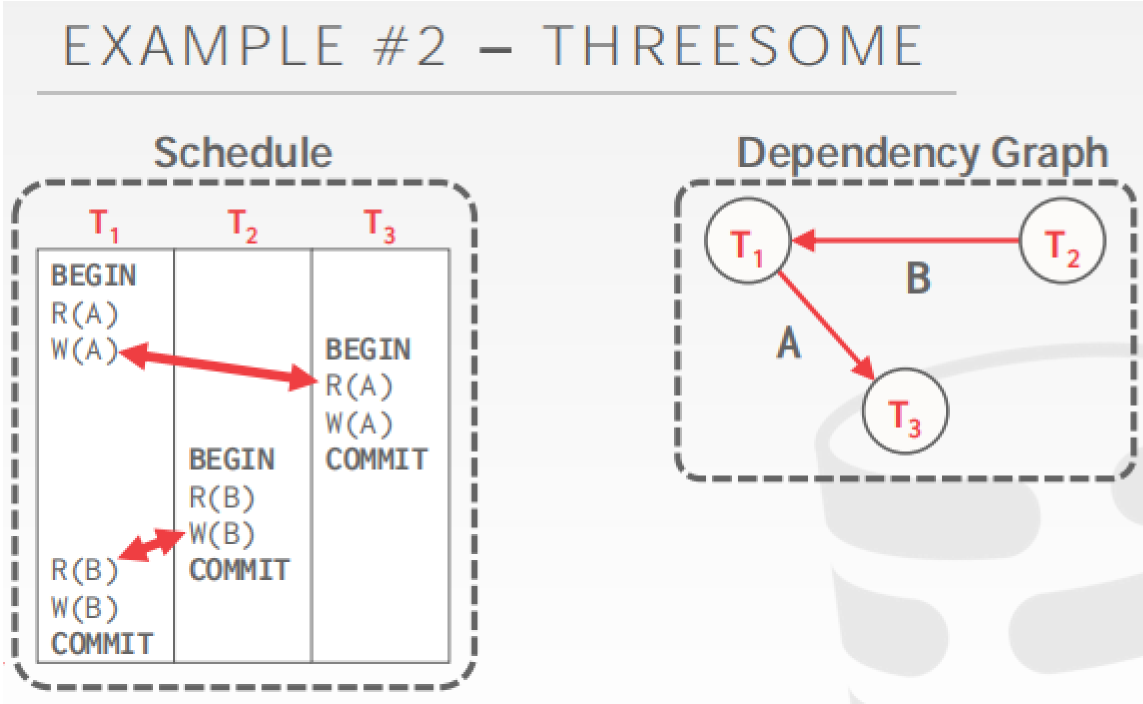
concurrency control 并发控制



右边这个图叫 precedence graph，优先图





VIEW SERIALIZABILITY

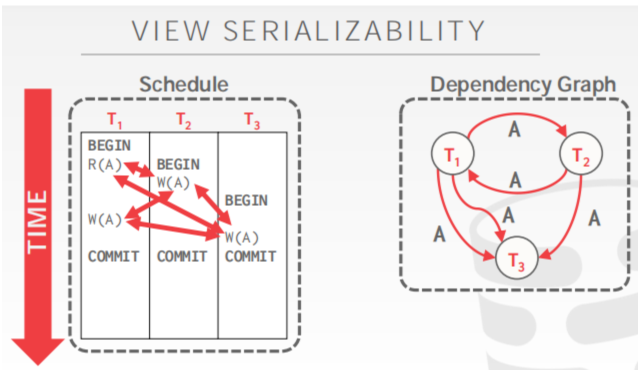
Alternative (weaker) notion of serializability.

Schedules S_1 and S_2 are view equivalent if:

→ If T_1 reads initial value of A in S_1, then T_1 also reads initial value of A in S_2.

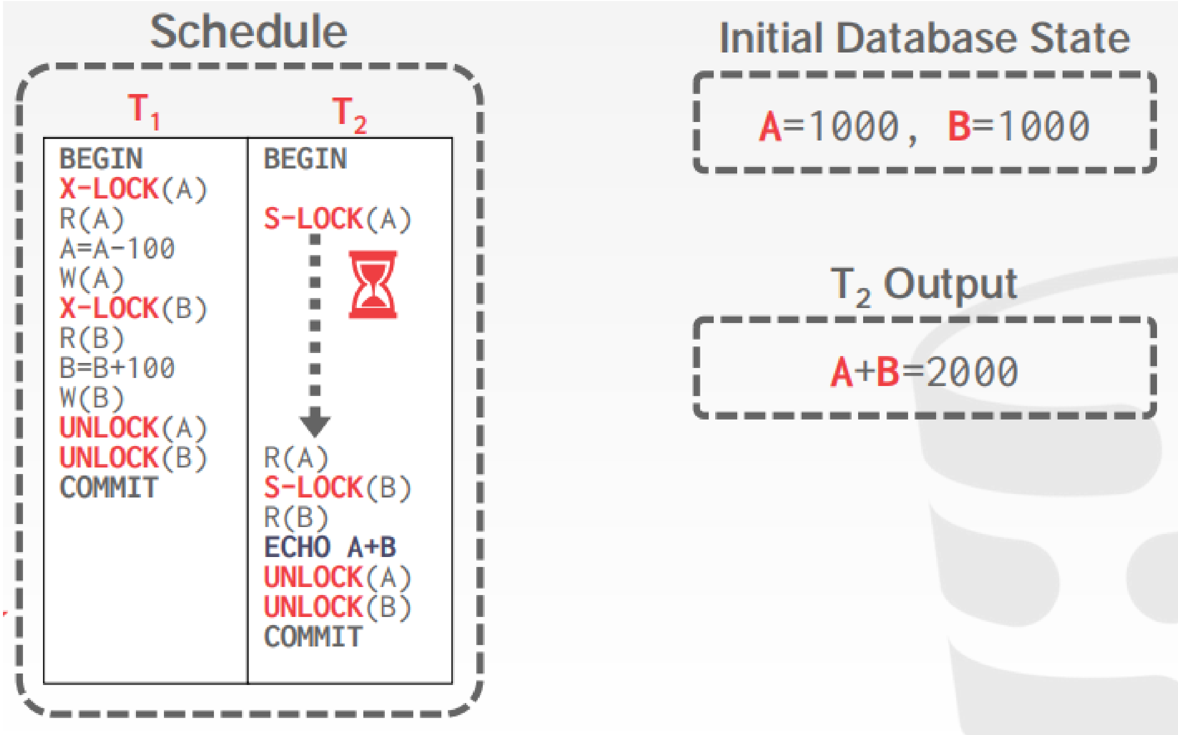
→ If T_1 reads value of A written by T_2 in S_1, then T_1 also reads value of A written by T_2 in S_2.

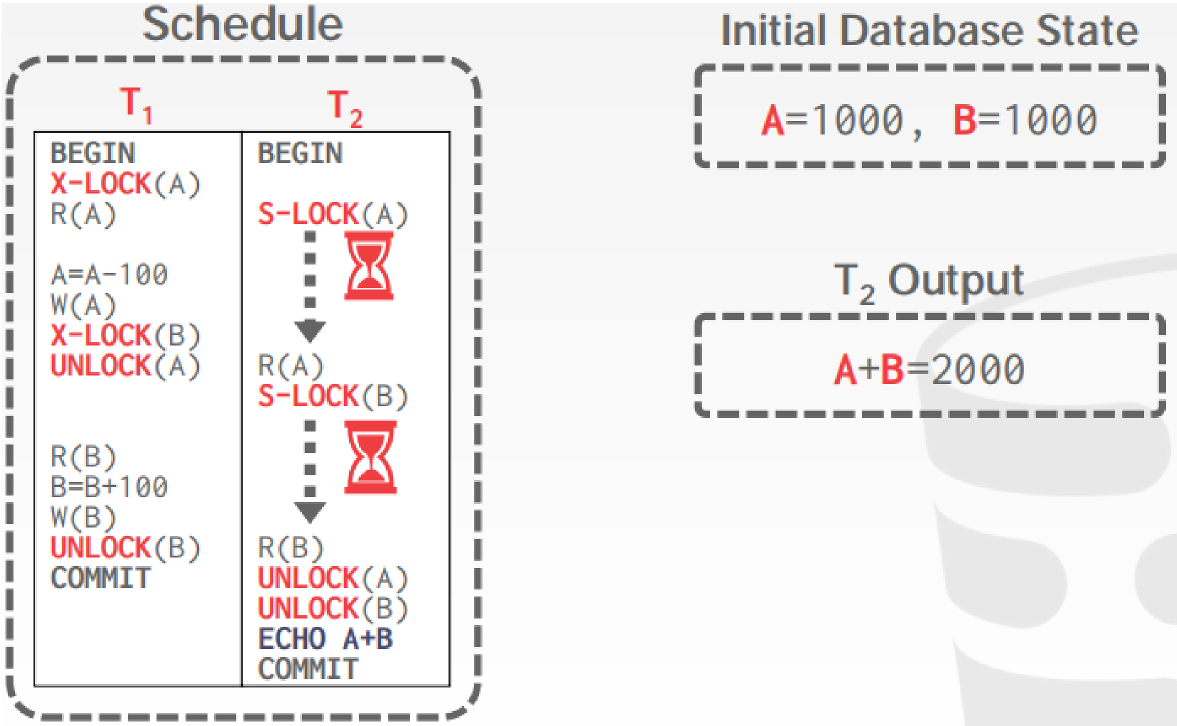
→ If T_1 writes final value of A in S_1, then T_1 also writes final value of A in S_2.

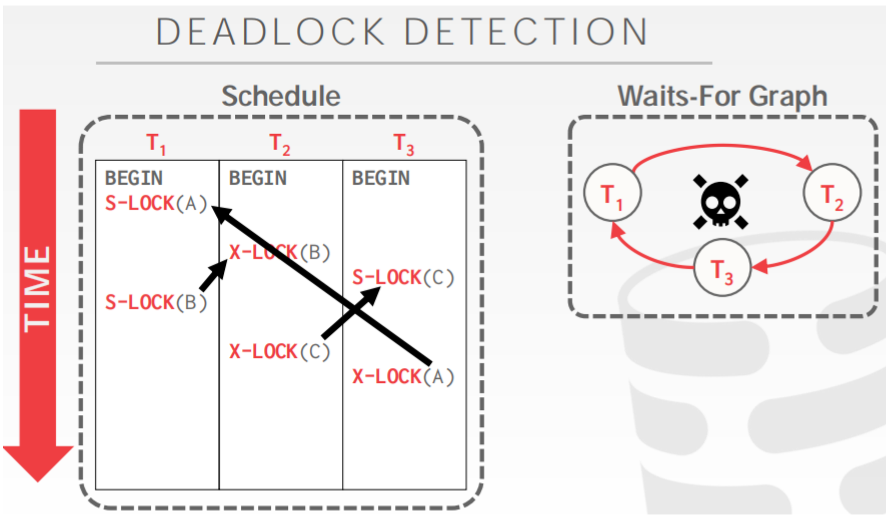


两种锁🔒 分享的 S 锁，专用的 X 锁

只要是要用这个量，就申请锁，用完就释放

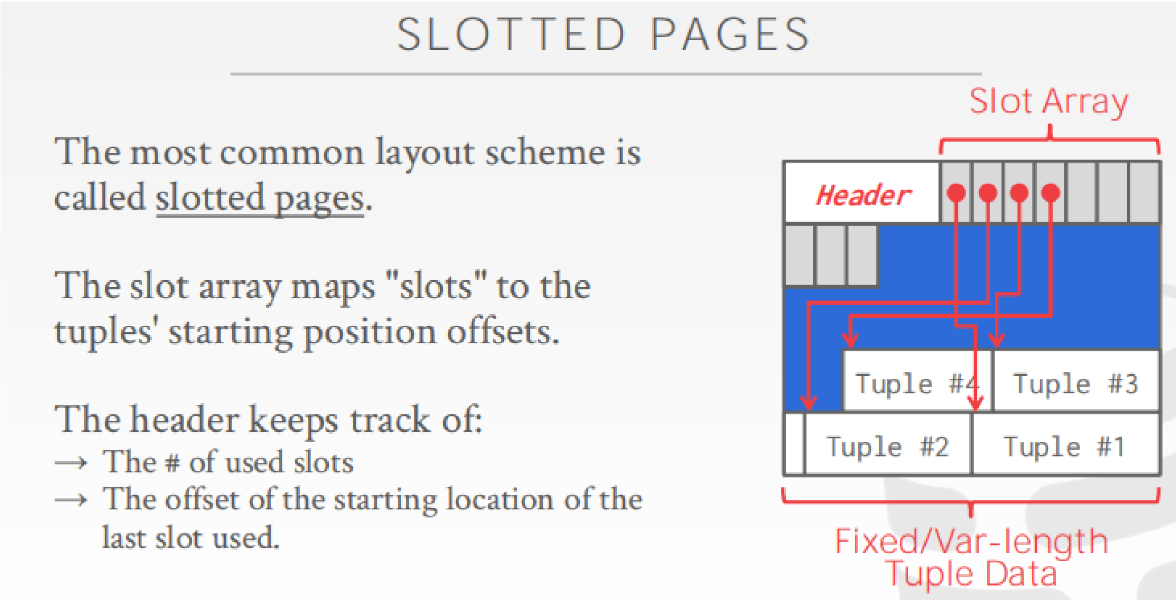

Schedule
T1
T2
BEGIN
X-LOCK(A)
R(A)
A=A-100
W(A)
X-LOCK(B)
R(B)
B=B+100
W(B)
UNLOCK(A)
UNLOCK(B)
COMMIT
BEGIN
S-LOCK(A)
R(A)
S-LOCK(B)
R(B)
ECHO A+B
UNLOCK(A)
UNLOCK(B)
COMMIT
Initial Database State
A=1000, B=1000
T2 Output
A+B=2000


Schedule
T1
T2
BEGIN
X-LOCK(A)
R(A)
A=A-100
W(A)
X-LOCK(B)
UNLOCK(A)
R(B)
B=B+100
W(B)
UNLOCK(B)
COMMIT
BEGIN
S-LOCK(A)
R(A)
S-LOCK(B)
R(B)
UNLOCK(A)
UNLOCK(B)
ECHO A+B
COMMIT
Initial Database State
A=1000, B=1000
T2 Output
A+B=2000


DEADLOCK DETECTION
Schedule
TIME
T1
T2
T3
BEGIN
S-LOCK(A)
S-LOCK(B)
BEGIN
X-LOCK(B)
X-LOCK(C)
BEGIN
S-LOCK(C)
X-LOCK(A)
Waits-For Graph
T1
T2
T3

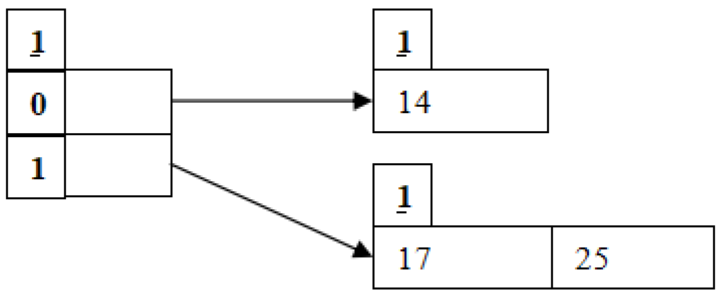
存储 Storage

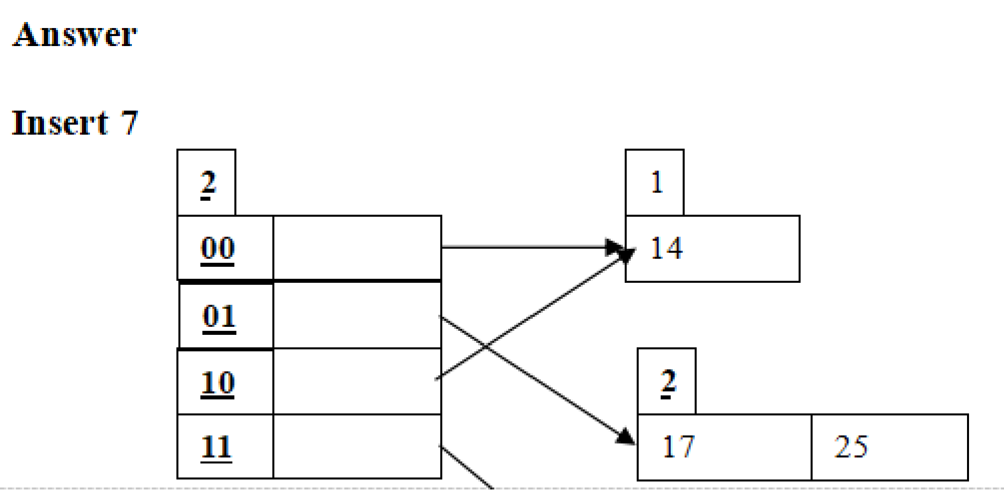
slotted pages

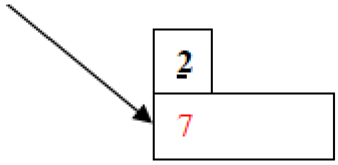


可拓展哈希

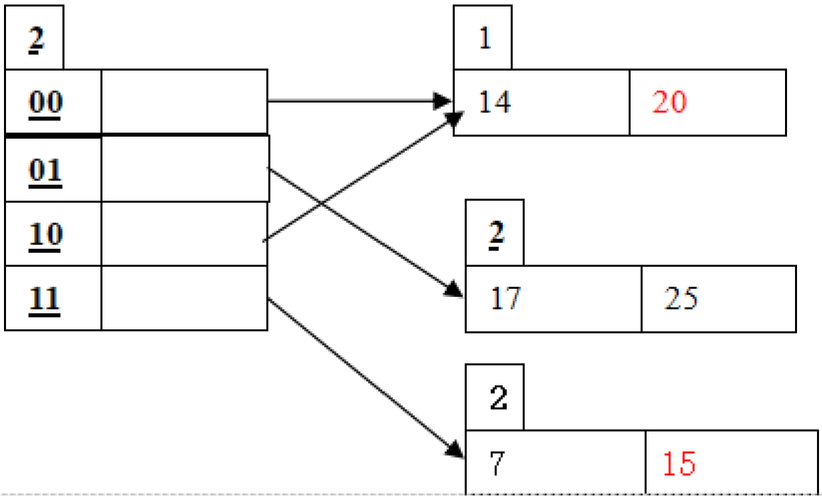
左边这个表叫 catalog ，是可以拓展的，左上角叫 global depth，指着一个桶，一个桶只能放两个键



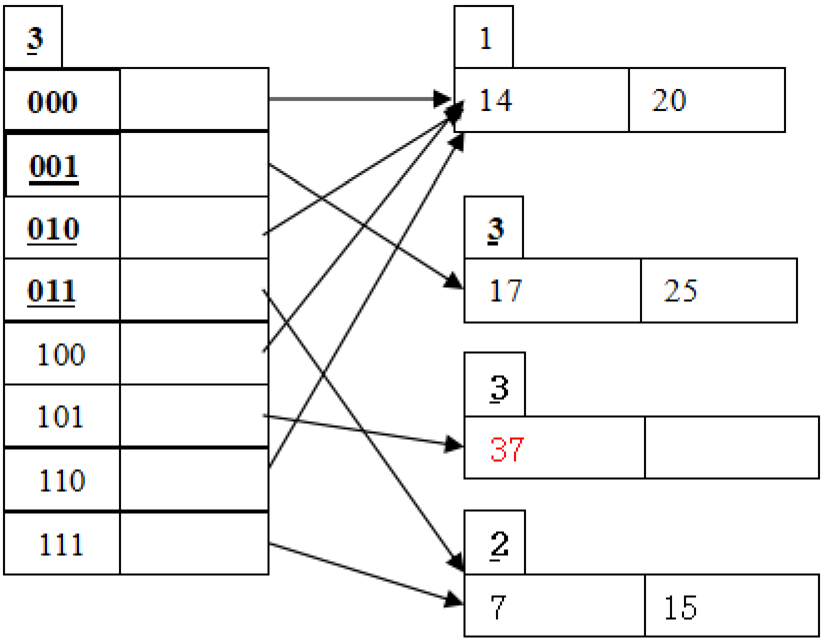




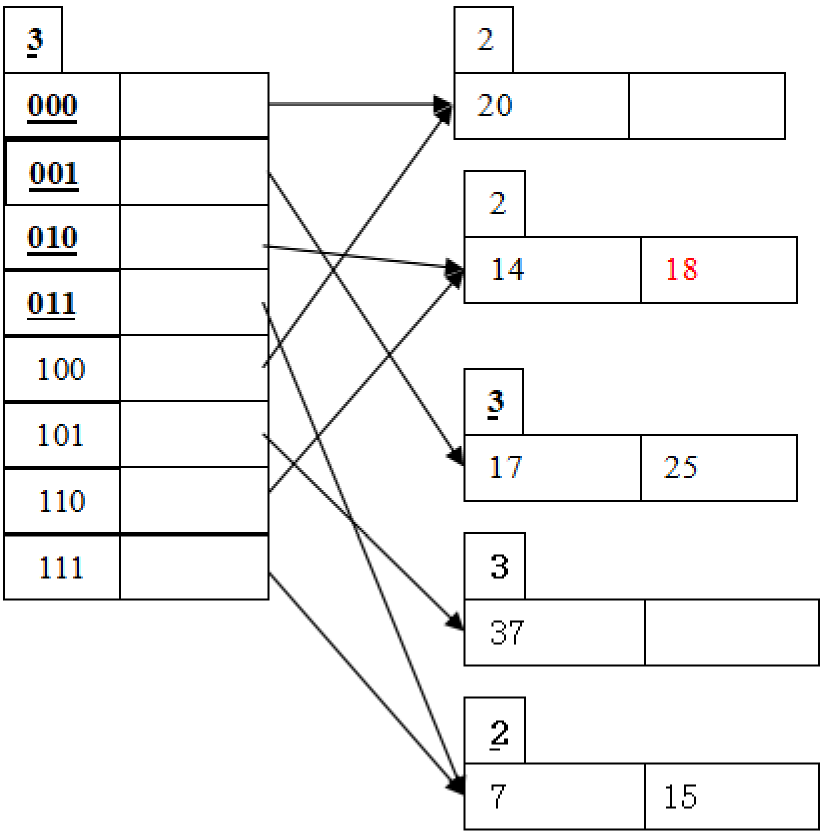
Insert 15 and 20



Insert 37



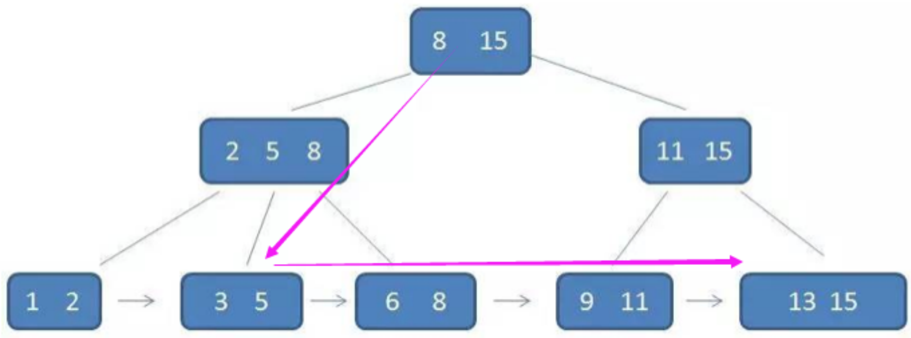
Insert 18

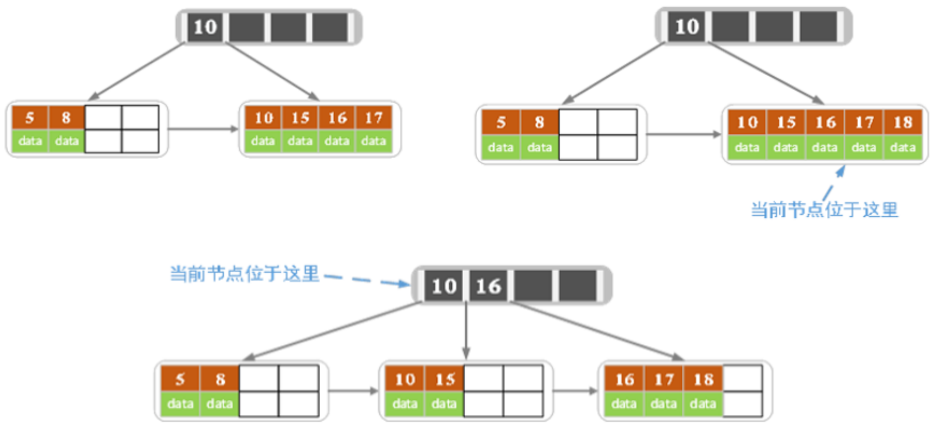


查询

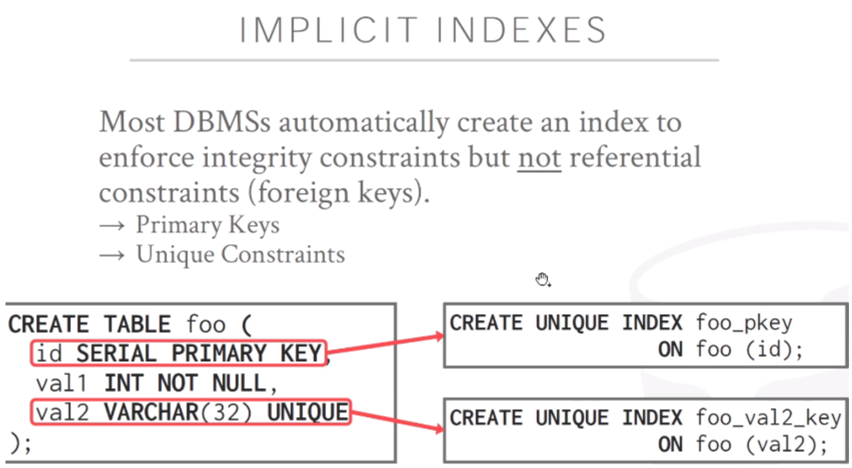
B + 树

满了就把中间的往上升，然后指向右边的是紧比它大的

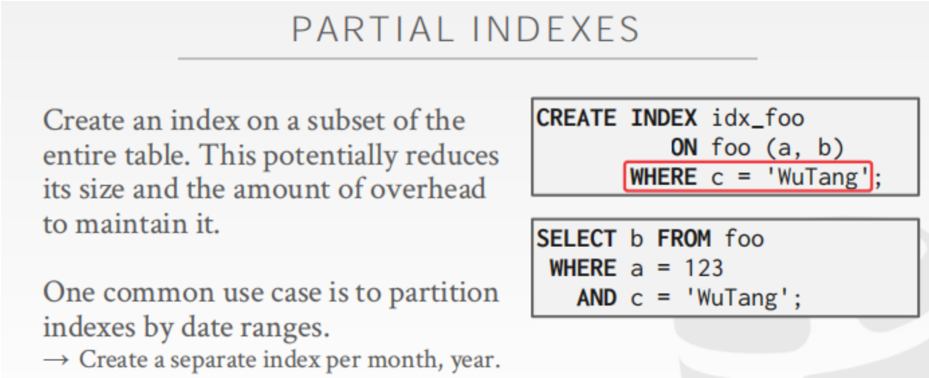




隐式的索引，即如果是 unique，也建一个 B+ 树成为一个索引，建索引，就是把这个属性建B+树

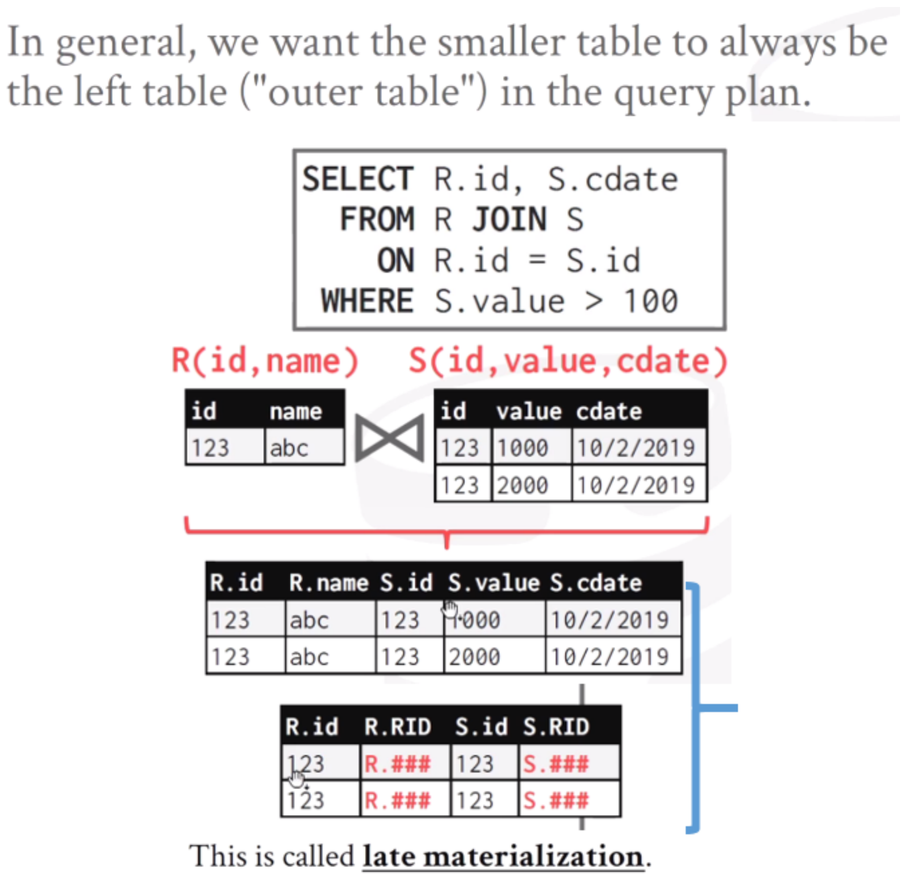


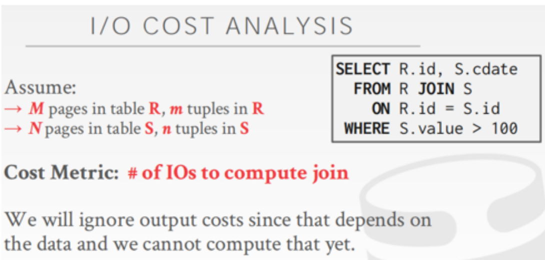
部分索引

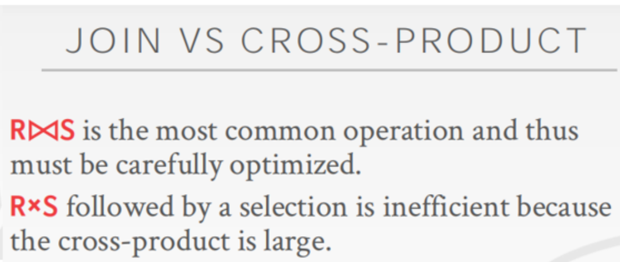


Join

我们想让小的在左边，作为 outer







Nested Loop Join：

→ Simple / Stupid

→ Block

→ Index

Sort-Merge Join

Hash Join

1. Simple / Stupid

Why is this algorithm stupid?
→ For every tuple in **R**, it scans **S** once

Cost: M + (m · N)

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

M pages
m tuples

S(id,value,cdate)

id	value	cdate
100	2222	10/2/2019
500	7777	10/2/2019
400	6666	10/2/2019
100	9999	10/2/2019
200	8888	10/2/2019

N pages
n tuples

前提是 S 无法全部在硬盘中完整的装下，所以每次都是全部读，应该S是外，这里错

2. Block

BLOCK NESTED LOOP JOIN

```
foreach block B_R ∈ R:
  foreach block B_S ∈ S:
    foreach tuple r ∈ B_R:
      foreach tuple s ∈ B_s:
        emit, if r and s match
```

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

M pages
m tuples

S(id,value,cdate)

id	value	cdate
100	2222	10/2/2019
500	7777	10/2/2019
400	6666	10/2/2019
100	9999	10/2/2019
200	8888	10/2/2019

N pages
n tuples

Cost Analysis:

→ $M + (M \cdot N) =$

3. Index

1. 内部表正好有索引，索引在连接的那个属性上
2. on the fly （临时）临时建索引

We can avoid sequential scans by using an index to find inner table matches.
→ Use an existing index for the join.
→ Build one on the fly (hash table, B+Tree).

INDEX NESTED LOOP JOIN

Assume the cost of each index probe is some constant C per tuple. 索引探测

Cost: $M + (m \cdot C)$

Index(S.id)

R(id,name) — M pages, m tuples

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate) — N pages, n tuples

id	value	cdate
100	2222	10/2/2019
500	7777	10/2/2019
400	6666	10/2/2019
100	9999	10/2/2019
200	8888	10/2/2019

CMU 15-445/645 (F

这里还是假设，R是一页一页的读入的

4. 改进方法：
    1. Pick the smaller table as the outer table.
    2. Buffer as much of the outer table in memory as possible.
    3. Loop over the inner table or use an index.
5. Sort-Merge Join

Phase #1: Sort

→ Sort both tables on the join key(s).

→ We can use the external merge sort （外归并排序）algorithm that we talked about last class.

Phase #2: Merge

→ Step through the two sorted tables with cursors（光标） and emit matching tuples.

→ May need to backtrack （回溯）depending on the join type.

SORT-MERGE JOIN

```
sort R,S on join keys
cursorR ← Rsorted, cursorS ← Ssorted
while cursorR and cursorS:
  if cursorR > cursorS:
    increment cursorS
  if cursorR < cursorS:
    increment cursorR
  elif cursorR and cursorS match:
    emit
    increment cursorS
```

SORT-MERGE JOIN

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/2/2019
100	9999	10/2/2019
200	8888	10/2/2019
400	6666	10/2/2019
500	7777	10/2/2019

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/2/2019
100	Andy	100	9999	10/2/2019
200	GZA	200	8888	10/2/2019
200	GZA	200	8888	10/2/2019

注意这里有回溯

SORT-MERGE JOIN

公式是外排序的公式，M是页面数，B是缓冲区，一个缓冲区存一个页面

Sort Cost (R): $2M \cdot (1 + \lceil \log_{B-1} \lceil M / B \rceil \rceil)$

Sort Cost (S): $2N \cdot (1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil)$

Merge Cost: $(M + N)$

Total Cost: Sort + Merge

→ Sort Cost (**R**) = $2000 \cdot (1 + \lceil \log_{99} 1000 / 100 \rceil)$ = **4000 IOs**

→ Sort Cost (**S**) = $1000 \cdot (1 + \lceil \log_{99} 500 / 100 \rceil)$ = **2000 IOs**

→ Merge Cost = (1000 + 500) = **1500 IOs**

→ Total Cost = 4000 + 2000 + 1500 = **7500 IOs**

The worst case for the merging phase is when the join attribute of all of the tuples in both relations contain the same value.

Cost: (M · N) + (sort cost)

WHEN IS SORT-MERGE JOIN USEFUL?

连接属性已排序

One or both tables are already sorted on join key.

Output must be sorted on join key.

输出必排序

The input relations may be sorted by either by an explicit sort operator, or by scanning the relation using an index on the join key.

6. Hash Sort

HASH JOIN

If tuple $r \in R$ and a tuple $s \in S$ satisfy the join condition, then they have the same value for the join attributes.

If that value is hashed to some partition i, the R tuple must be in r_i and the S tuple in s_i.

Therefore, R tuples in r_i need only to be compared with S tuples in s_i.

BASIC HASH JOIN ALGORITHM

Phase #1: Build

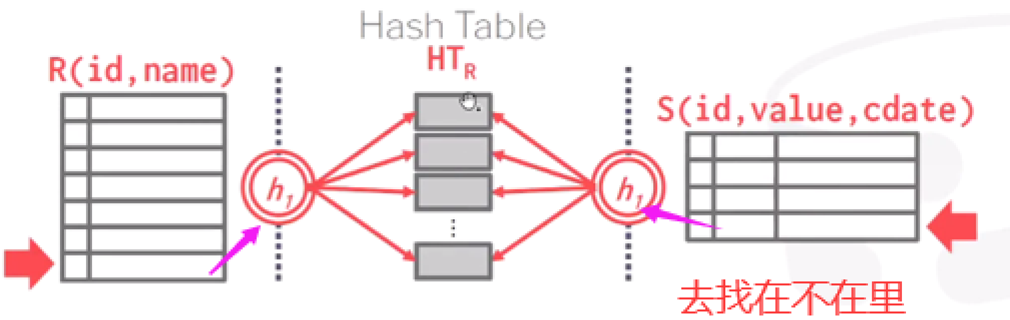
→ Scan the outer relation and populate a hash table using the hash function h_1 on the join attributes.

Phase #2: Probe

→ Scan the inner relation and use h_1 on each tuple to jump to a location in the hash table and find a matching tuple.

BASIC HASH JOIN ALGORITHM

```
build hash table HT_R for R
foreach tuple s ∈ S
  output, if h_1(s) ∈ HT_R
```



HASH TABLE CONTENTS

Key: The attribute(s) that the query is joining the tables on.

值就是最后选出来的要啥属性，这里就先存下什么属性

Value: Varies per implementation.

→ Depends on what the operators above the join in the query plan expect as its input.

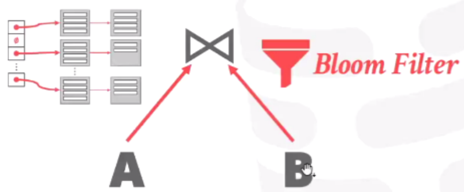
HASH TABLE VALUES

Approach #1: Full Tuple 全部都写进去
→ Avoid having to retrieve the outer relation's tuple contents on a match.
→ Takes up more space in memory.

Approach #2: Tuple Identifier 只写类似 RID 页号 槽号
→ Ideal for column stores because the DBMS doesn't fetch data from disk it doesn't need.
→ Also better if join selectivity is low.

PROBE PHASE OPTIMIZATION

Create a Bloom Filter during the build phase when the key is likely to not exist in the hash table.
→ Threads check the filter before probing the hash table. This will be faster since the filter will fit in CPU caches.
→ Sometimes called ***sideways information passing***.



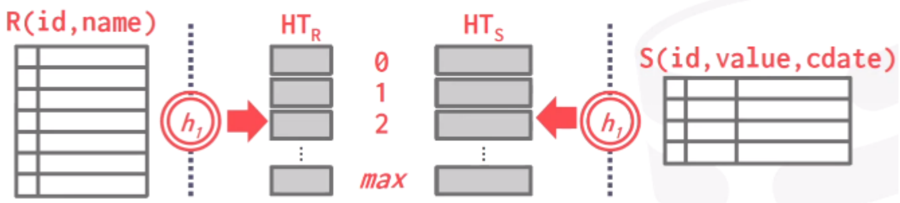
build 的时候顺便建一个 Bloom Filter 即 二进制的位图，因为这个非常小，所以容易查，小也可以使得它放到 cache 中

日本的 Grace Hash Join

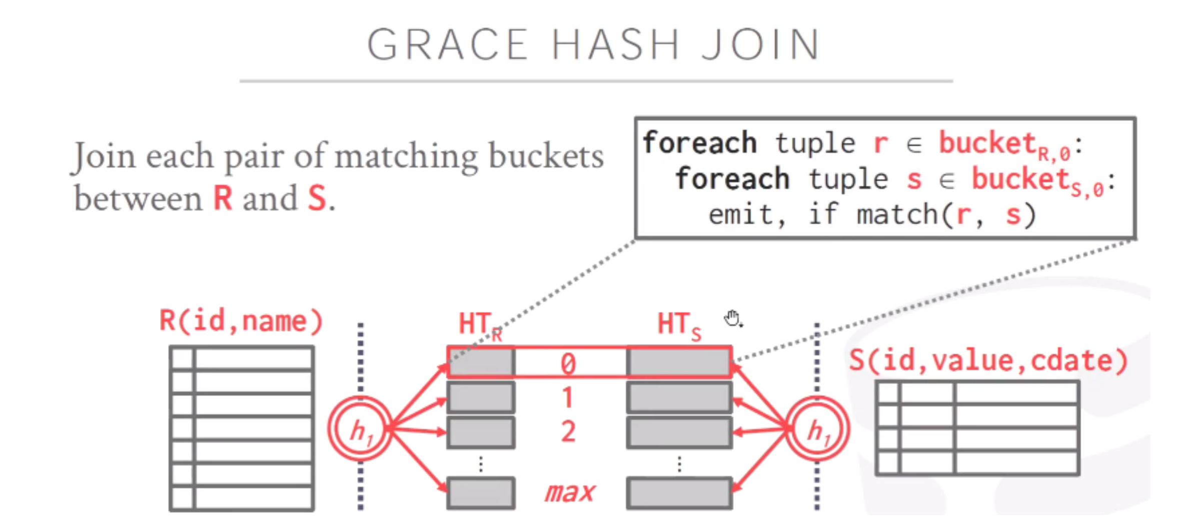
GRACE HASH JOIN

Hash **R** into (0, 1, ..., *max*) buckets.

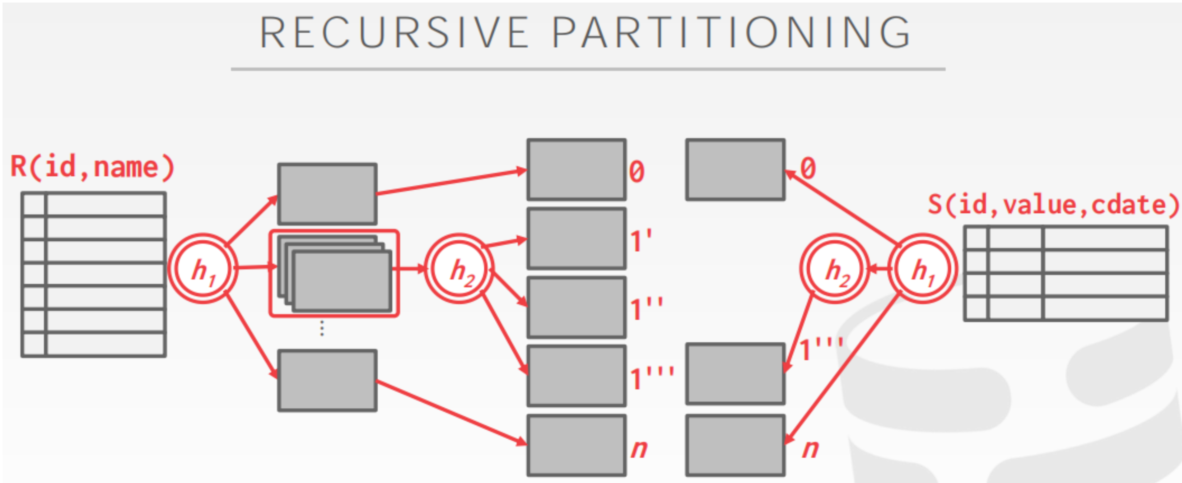
Hash **S** into the same # of buckets with the same hash function.



这里 HE_R 是 max + 1 个缓冲区页面，我们要 build 这个表，我们需要 max + 2 个缓冲区页面，因为那一个是用来读入的



一个桶放不下，递归分桶



Cost of hash join?

→ Assume that we have enough buffers.

→ Cost: $3(M + N)$

Partitioning Phase:

→ Read+Write both tables

→ $2(M+N)$ IOs

Probing Phase:

→ Read both tables

→ M+N IOs

CONCLUSION

Hashing is almost always better than sorting for operator execution.

Caveats:
→ Sorting is better on non-uniform data.
→ Sorting is better when result needs to be sorted.

Good DBMSs use either or both.