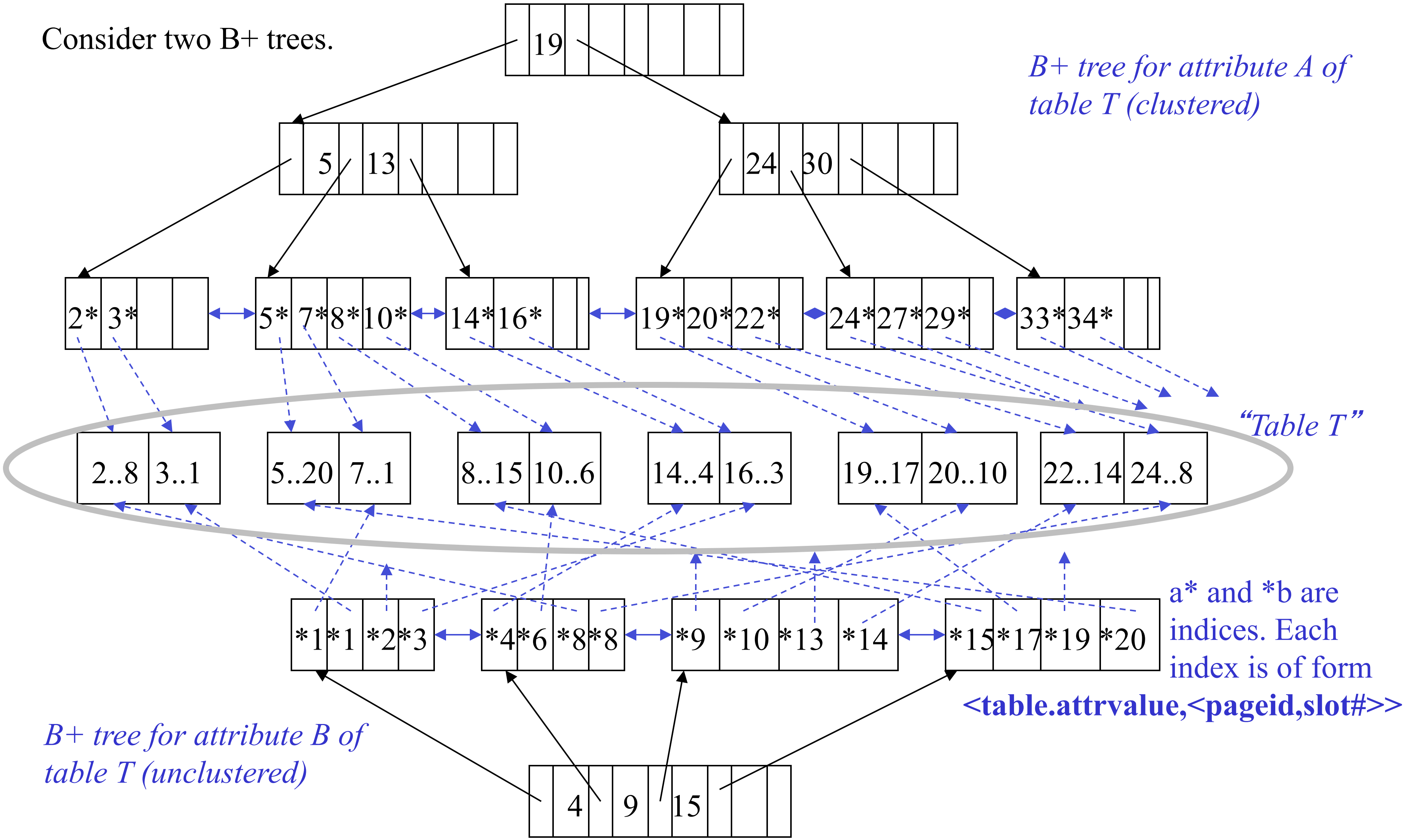
Indexing

CSx265

This lecture assumes that you have watched Widom's videos on indexing or read the corresponding material from Ullman and Widom, and watched Doug's four videos on B+ tree indexing and extendible-hash indexing

Let's define clustered and unclustered B+ tree indices

Consider two B+ trees.



SELECT T.C FROM T WHERE T.A > 14 AND T.B <= 10

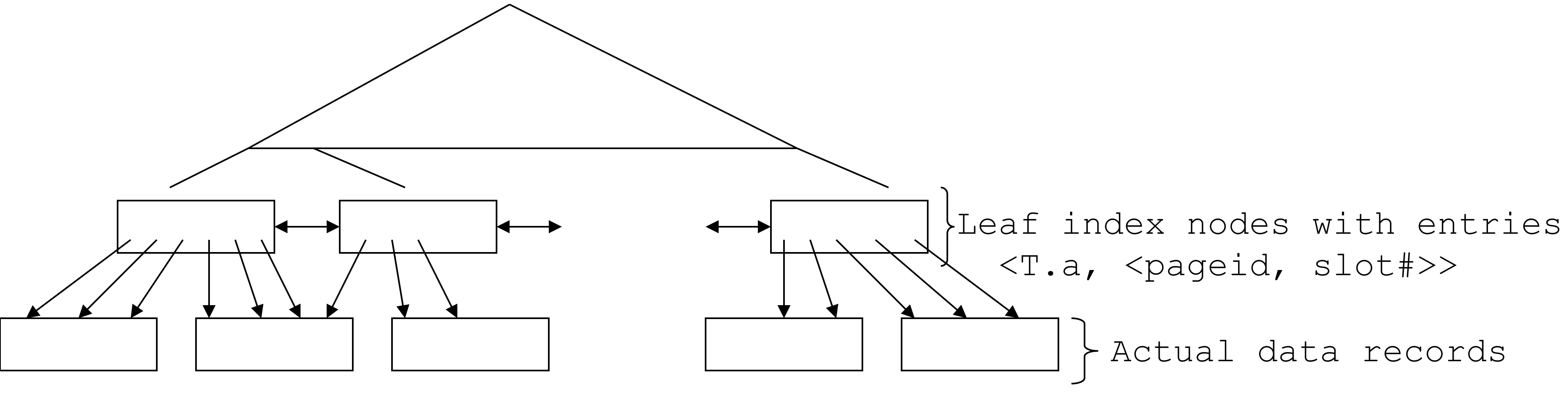
Exploiting T.A clustered B+ tree index will result in fewer pages being read from disk.

Evaluation of relational operators

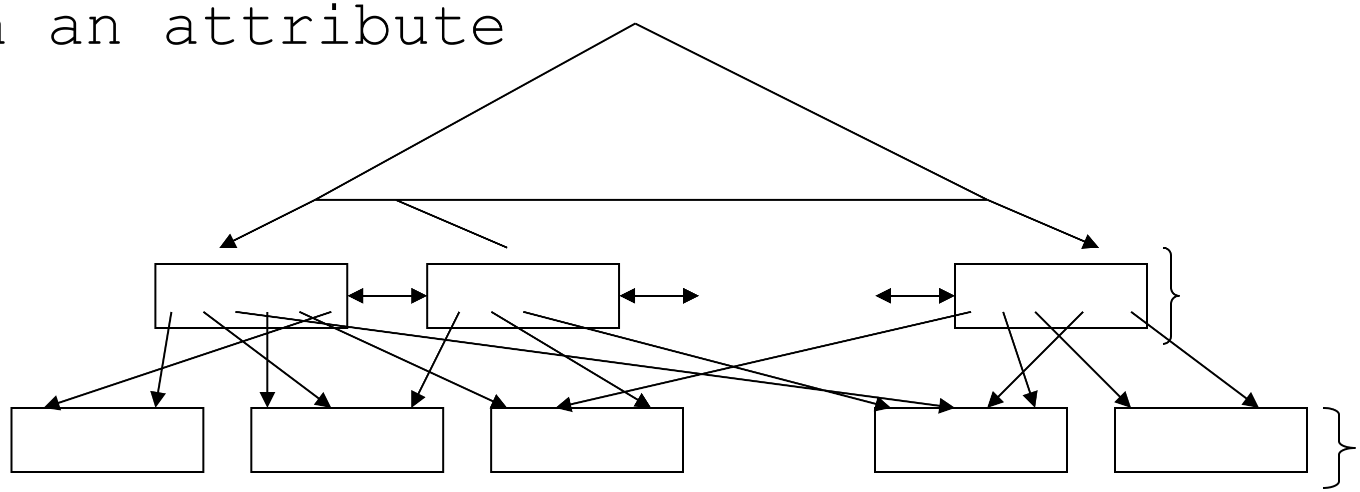
1) A file (data records for a table) may be unsorted (with no index)

2) A file may be sorted by the values of one attribute (with no index)

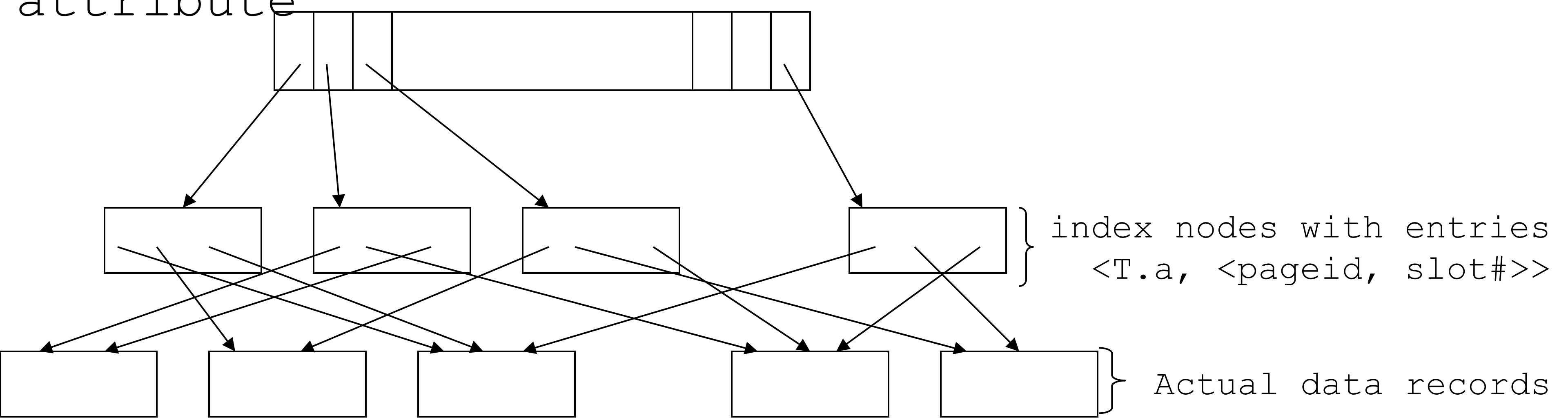
3) We can have a clustered B+ tree index for the file on an attribute



4) We can have an unclustered B+ tree index for a file on an attribute



5) We can have a hash index for a file on an attribute



Consider:

```
SELECT *
FROM Shipped
WHERE Shipped.ShipId = x
```

```
SELECT *
FROM Shipped
WHERE Shipped.ShipId > x
```

$\sigma_{\text{ShipId=x}}(\text{Shipped})$

$\sigma_{\text{ShipId>x}}(\text{Shipped})$

1) Shipped unsorted with respect to ShipId; No index on ShipId: perform file scan

2) Shipped sorted with respect to ShipId; no index on ShipId: perform file scan. Can terminate early.

3) Clustered B+ tree on ShipId: Lookup x and scan data records directly

4) Unclustered B+ tree on ShipId: Lookup x and scan index leaves, only reading/scanning data pages that satisfy Query

5) Hash Index on ShipId: Lookup x and scan data pages in case of '='; file scan in case of '>'

Consider:

$$\sigma_{\text{Isbn=x \& Quantity < y \& ShipId > z}}(\text{Shipped})$$

```
SELECT *
FROM Shipped
WHERE Isbn = x AND Quantity < y AND ShipId > z
```

1) No indices and unsorted with respect to Isbn, Quantity, ShipId: file scan

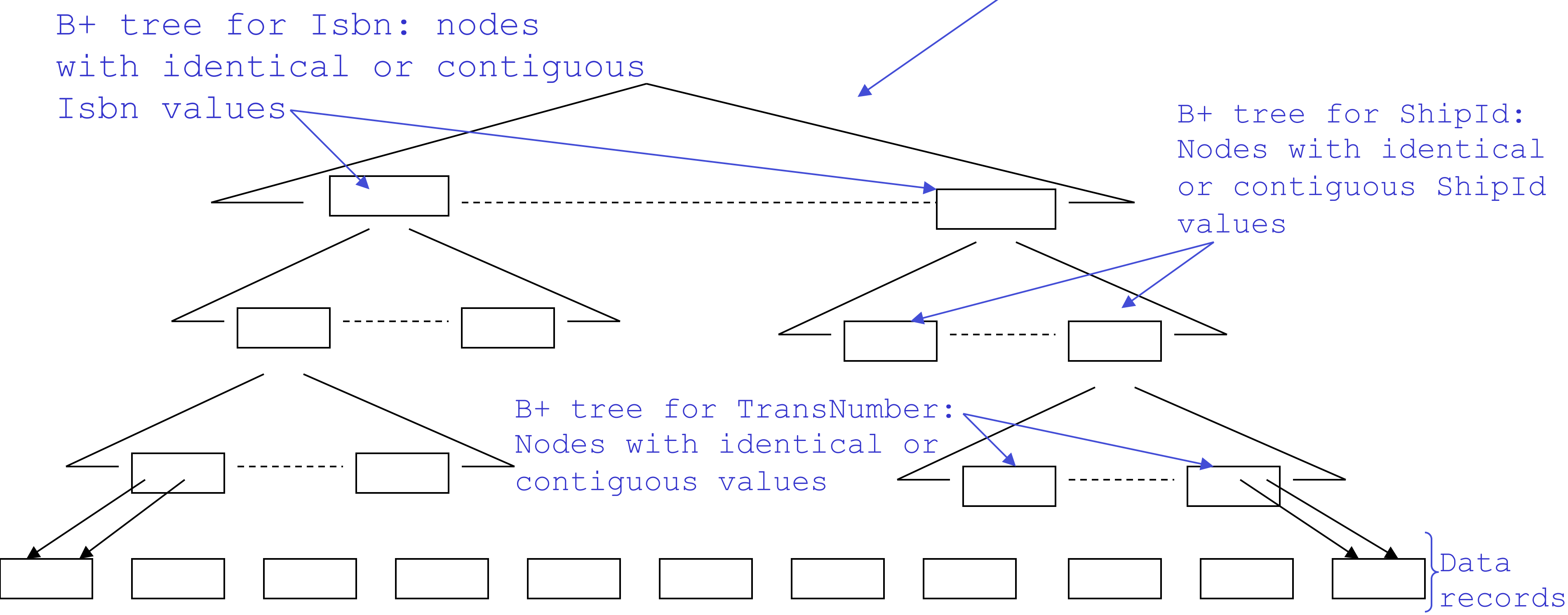
2) Hash Index on Isbn and no index/sort on other two: scan data pages with matching Isbn and check for other conditions.

3)Clustered B+ tree index on ShipId, no index on Quantity, hash index on Isbn: Scan data pages with matching ShipId and check for other conditions **OR** scan data pages with matching Isbn and check for other conditions **OR** Intersect indices with matching Isbn and ShipId and check for Quantity condition

4) Clustered composite B+ tree index on (Isbn, ShipId) and no other indices: scan data pages with matching Isbn,ShipId and check for Quantity condition.

5) Clustered composite B+ tree on (Isbn, ShipId, TransNumber):

6) Clustered composite B+ tree on (TransNumber, ShipId, Isbn):



$$\sigma_{Isbn=x \ \& \ Quantity < y \ \& \ ShipId > z}(Shipped)$$

Consider the queries: $\pi_{\texttt{Isbn, ShipId}}(\texttt{Shipped})$

```
SELECT Isbn, ShipId
FROM Shipped
```

```
SELECT DISTINCT Isbn, ShipId
FROM Shipped
```

```
SELECT Isbn, Quantity
FROM Shipped
```

```
SELECT DISTINCT Isbn, Quantity
FROM Shipped
```

$\pi_{\texttt{Isbn, Quantity}}(\texttt{Shipped})$

How might sorting be used?

How might hashing be used?

Consider the query:

```
SELECT *
FROM Transactions T, Shipped S
WHERE S.TransNumber = T.TransNumber
```

$$\text{Shipped} \bowtie_{\text{S.TN=T.TN}} \text{Transactions}$$

$$\text{Shipped} \bowtie \text{Transactions}$$

```
JoinResult ← Empty
For each tuple, s, in Shipped
    For each tuple, t, in Transactions
        If (s.TN=t.TN) add s+t to JoinResult
```

$$S \bowtie_{(s\ R\ t)} T$$

```
JoinResult ← Empty
For each tuple, s, in S
   For each tuple, t, in T
      if (s R t) add s+t to JoinResult
```

```
JoinResult ← Empty
FOR each tuple, s, in S
   FOR each tuple, t, in σ(sRt)(T)
      add s+t to JoinResult
```

Index Nested Loops join

Index on right (inner) table of a join is most important

Consider the query:

```
SELECT *
FROM Transactions T, Shipped S
WHERE S.TransNumber = T.TransNumber
```

$$\text{Shipped} \bowtie_{S.TN=T.TN} \text{Transactions}$$

$$\text{Shipped} \bowtie \text{Transactions}$$

No indices, no sorts?

S sorted on TN?

T sorted on TN?

Index on S.TN only? Clustered?

Index on T.TN only? Clustered?

Index on both S.TN and T.TN?

Consider the following Query in SQL and relational algebra:

```
SELECT *
FROM Shipped S1, Transactions T1
WHERE S1.TransNumber = T1.TransNumber AND
              S1.Isbn = I1 AND T1.PaymentClearanceDate = CD
```

I1 and **CD** are parameters

$(\sigma_{PCD=\mathbf{CD}}((\sigma_{Isbn=\mathbf{I1}}(Shipped)) \bowtie Transactions))$

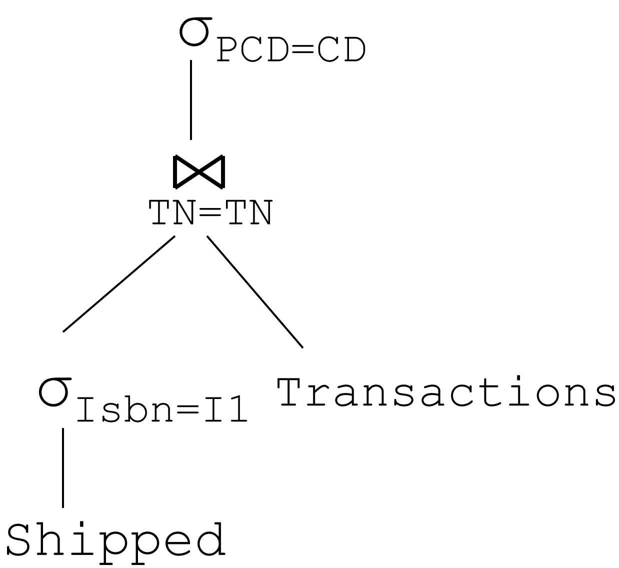
$((\sigma_{Isbn=\mathbf{I1}}(Shipped)) \bowtie (\sigma_{PCD=\mathbf{CD}}(Transactions)))$

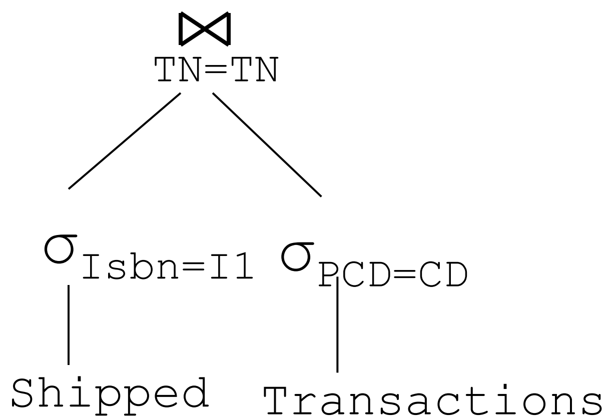
$(\sigma_{Isbn=\mathbf{I1}}(Shipped \bowtie (\sigma_{PCD=\mathbf{CD}}(Transactions))))$

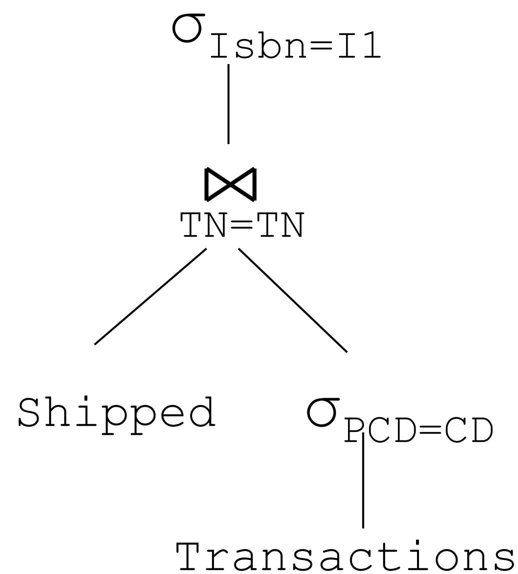
Other possibilities?

```
SELECT *
FROM Shipped S1, Transactions T1
WHERE S1.TransNumber = T1.TransNumber AND
      S1.Isbn = I1 AND T1.PaymentClearanceDate = CD
```

Query Evaluation Trees







Other trees ??

Left-deep tree: each right child of a join is a base table

Consider the following Query in SQL and relational algebra:

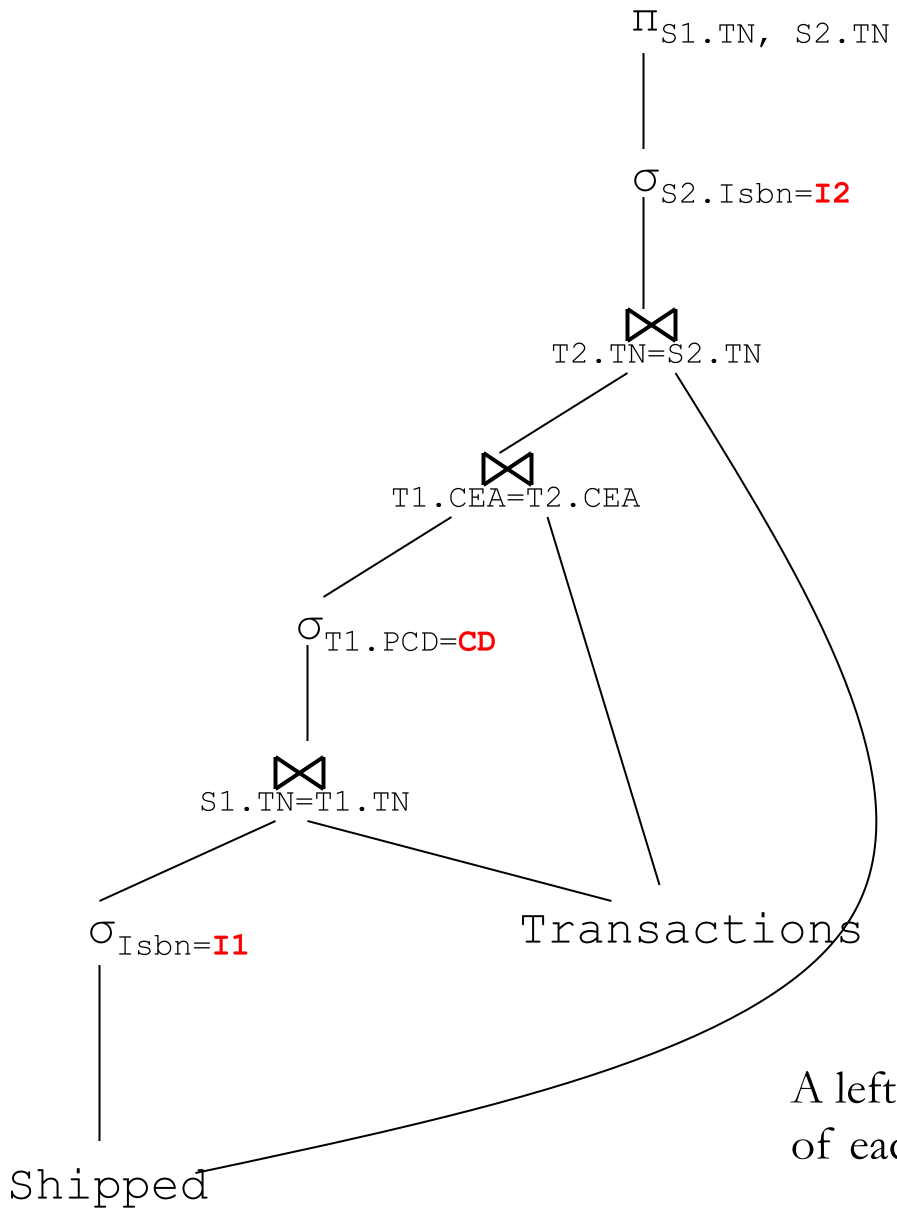
For each book, I1, bought on date CD, by a customer T1.CEA on transaction S1.TN, list the Transactions S2.TN for which T1.CEA bought a second book, I2. (this query might be an auxiliary/nested query for updating CoBought books or the like)

```
SELECT S1.TransNumber, S2.TransNumber
FROM Shipped S1, Shipped S2, Transactions T1, Transactions T2
WHERE S1.TransNumber = T1.TransNumber AND
      T2.TransNumber = S2.TransNumber AND
      S1.Isbn = I1 AND T1.PaymentClearanceDate = CD AND
      T1.CustomerEmailAddress = T2.CustomerEmailAddress AND
      S2.Isbn = I2
```

I1, **I2**, and **CD** are parameters

$$\pi_{S1.TN,S2.TN}\ (\sigma_{S2.Isbn=I2}\ (((((\sigma_{PCD=CD}\ ((\sigma_{Isbn=I1}\ (\rho(S1,\ Shipped))) \bowtie \rho(T1,Transactions))) \bowtie \rho(T2,Transactions))) \bowtie \rho(S2,\ Shipped))))$$

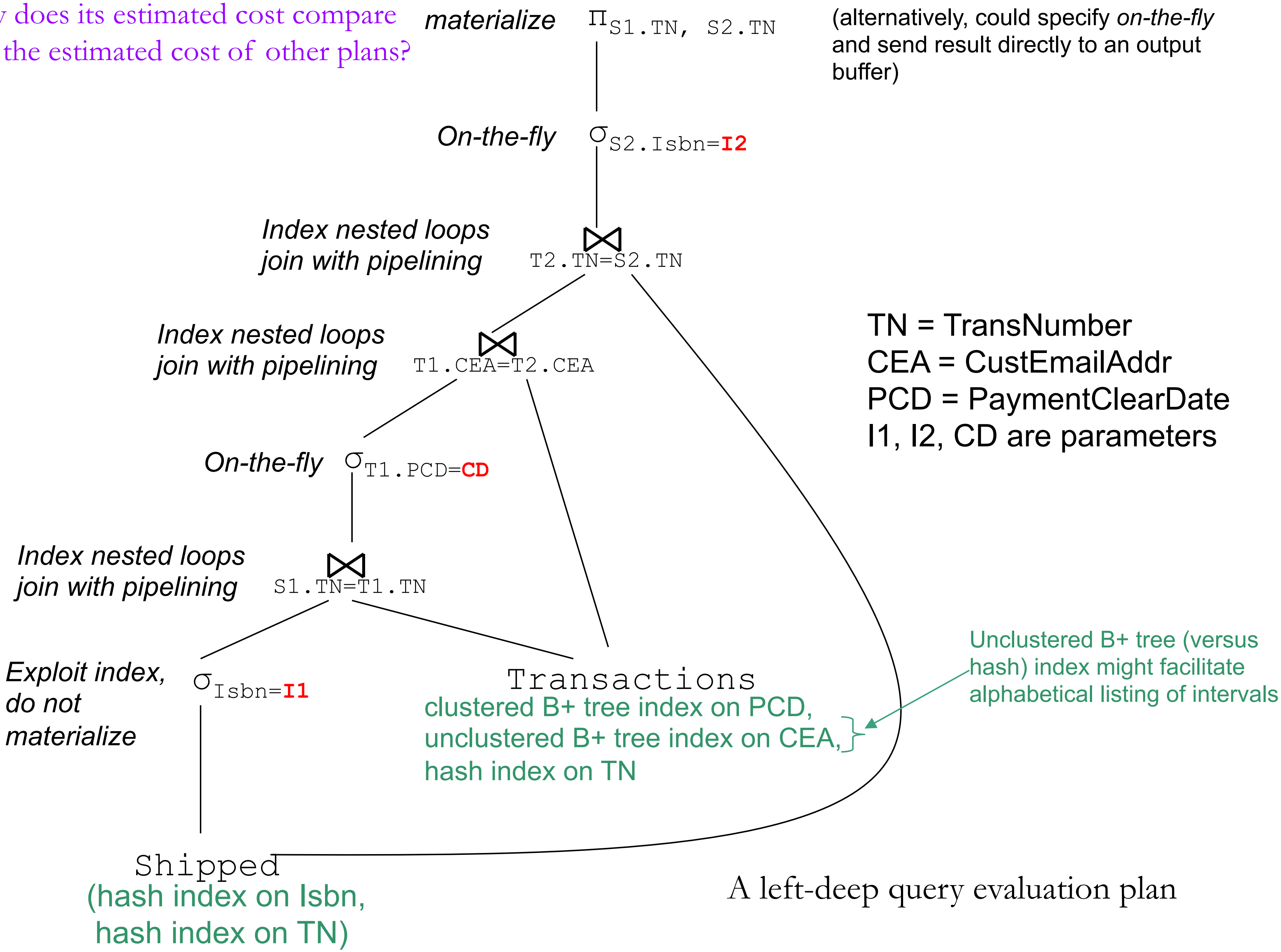
Draw left-deep tree(s) for this query



TN = TransNumber
CEA = CustEmailAddr
PCD = PaymentClearDate
I1, I2, CD are parameters

A left-deep query tree: the right child of each join is a base table.

What is the estimated cost of this plan?
How does its estimated cost compare to the estimated cost of other plans?



TN = TransNumber
CEA = CustEmailAddr
PCD = PaymentClearDate
I1, I2, CD are parameters

A left-deep query evaluation plan

Assume the following conditions hold for a relational DB that we've designed for an e-bookseller.

i) a block/page is 2^12 bytes.
ii) each tuple of Transactions requires 2^4 bytes
iii) each tuple of Shipped requires 2^4 bytes
iv) Each index (for any attribute of any table) requires 2^3 bytes
v) There are 2^27 tuples in Transactions
vi) There are 2^28 tuples in Shipped
vii) There are 2^17 tuples that satisfy PCD=CD
(PCD is PaymentClearanceDate, CD is a particular value, i.e., a constant)
viii) There are 2^20 unique Isbn distributed across Shipped
ix) There are 2^18 unique CEA distributed across Transactions (CEA is CustEmailAddress)
x) clustered B+ tree of order 2^8 index on PCD for Transactions, hash index on TN for Transactions, hash index on CEA for Transactions, hash index on Isbn for Shipped, hash index on TN for Shipped (TN is TransactionNumber)

• Which of these, (i) – (x), would be stored in the System Catalog. Elaborate as necessary with page references. I am particularly curious about (vii).

• Under the conditions listed above, what is the shallowest that the B+ tree on PCD can possibly be? What is deepest that it can be? Give your answers in terms of index nodes (root included) only (i.e., do not count the data pages as part of the tree).

Assume:

- a block/page is 2^{12} bytes (upper range)

Information found in System Catalog

- each tuple of Shipped relation/table requires 2^4 bytes
  - ➔ one block/page holds $2^{12}/2^4 = 2^8$ Shipped tuples
- each index on Isbn of form <Isbn, <pageid, slot#>> requires 2^3 bytes
  - ➔ each block/page holds $2^{12}/2^3 = 2^9$ indices

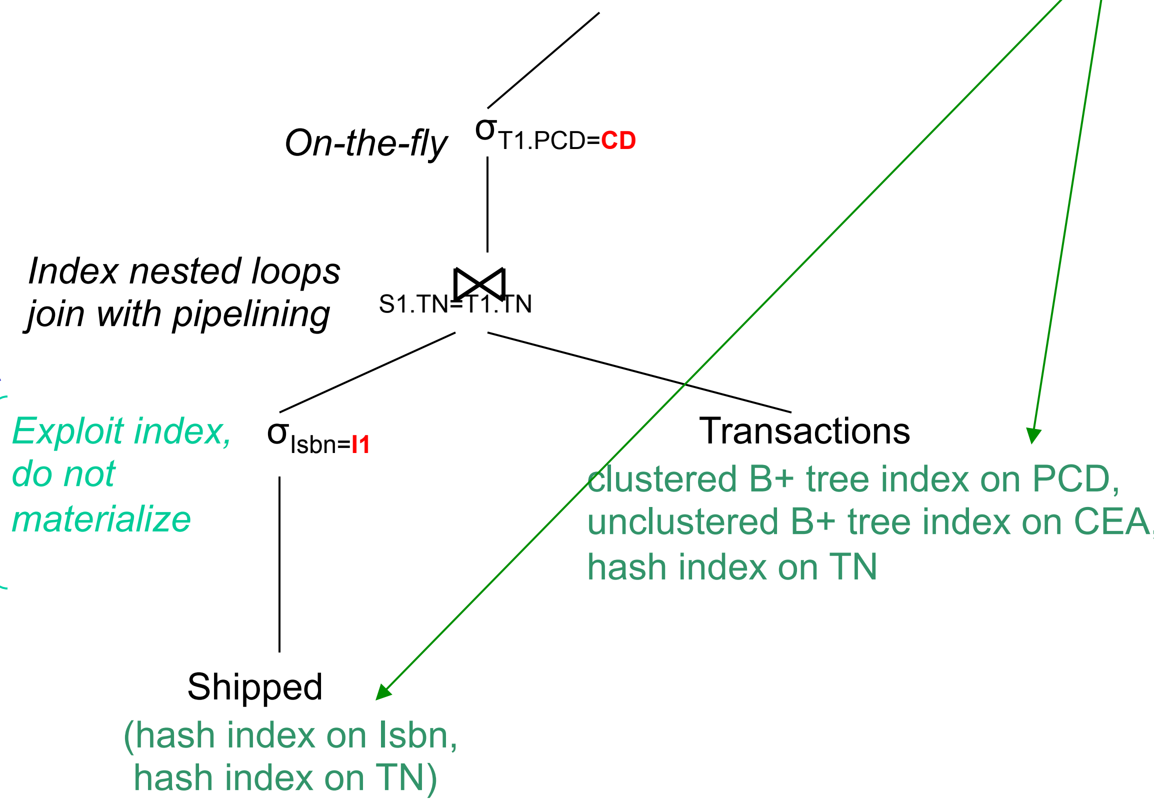
- there are 2^{28} tuples in Shipped (*Cardinality*) ➔ $2^{28}/2^8$ = 2^{20} pages <= *Size* <= 2^{21} = $2^{28}/2^7$ pages
- there are 2^{20} distinct Isbns in Shipped (*Index Cardinality*) ➔ $2^{28}/2^9$ = 2^{19} <= Index Size <= 2^{20} = $2^{28}/2^8$

1. Estimate size of result (under *uniform assumption*).

$2^{28}/2^{20} = 2^8$ tuples estimated to satisfy S.Isbn=I1

Estimated size of result = 2^8 tuples

$2^8/2^{28}$ < 5% of Shipped table (probably cheaper to use index, versus file scan, p. 401)



Assume:

- a block/page is 2^{12} bytes (upper range)

- each tuple of Shipped relation/table requires 2^4 bytes
  - ➔ one block/page holds $2^{12}/2^4 = 2^8$ Shipped tuples
- each index on Isbn of form <Isbn, <pageid, slot#>> requires 2^3 bytes
  - ➔ each block/page holds $2^{12}/2^3 = 2^9$ indices

- there are 2^{28} tuples in Shipped (*Cardinality*) ➔ $2^{28}/2^8$ = <u>2^{20} pages <= *Size* <= 2^{21}</u> = $2^{28}/2^7$ pages
- there are 2^{20} distinct Isbns in Shipped (*Index Cardinality*) ➔ $2^{28}/2^9$ = <u>2^{19} <= Index Size <= 2^{20}</u> = $2^{28}/2^8$

1. Estimate size of result (under *uniform assumption, p. 401*).

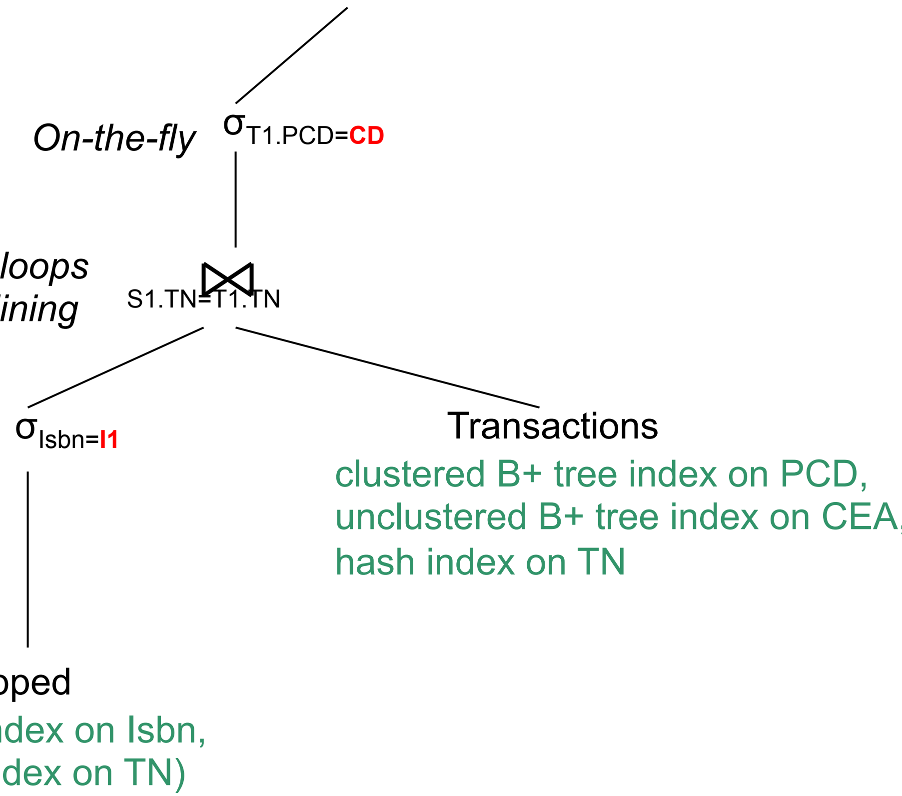
$2^{28}/2^{20} = 2^8$ tuples estimated to satisfy S.Isbn=I1

Estimated size of result = 2^8 tuples

2. Estimate # of page scans using Index on Isbn

1 index page since 2^8 per Isbn < 2^9 indices per block

between 1 data page (if all 2^8 tuples fit on 1 page) *and* 2^8 data pages (if each 2^8 tuples on different data page)



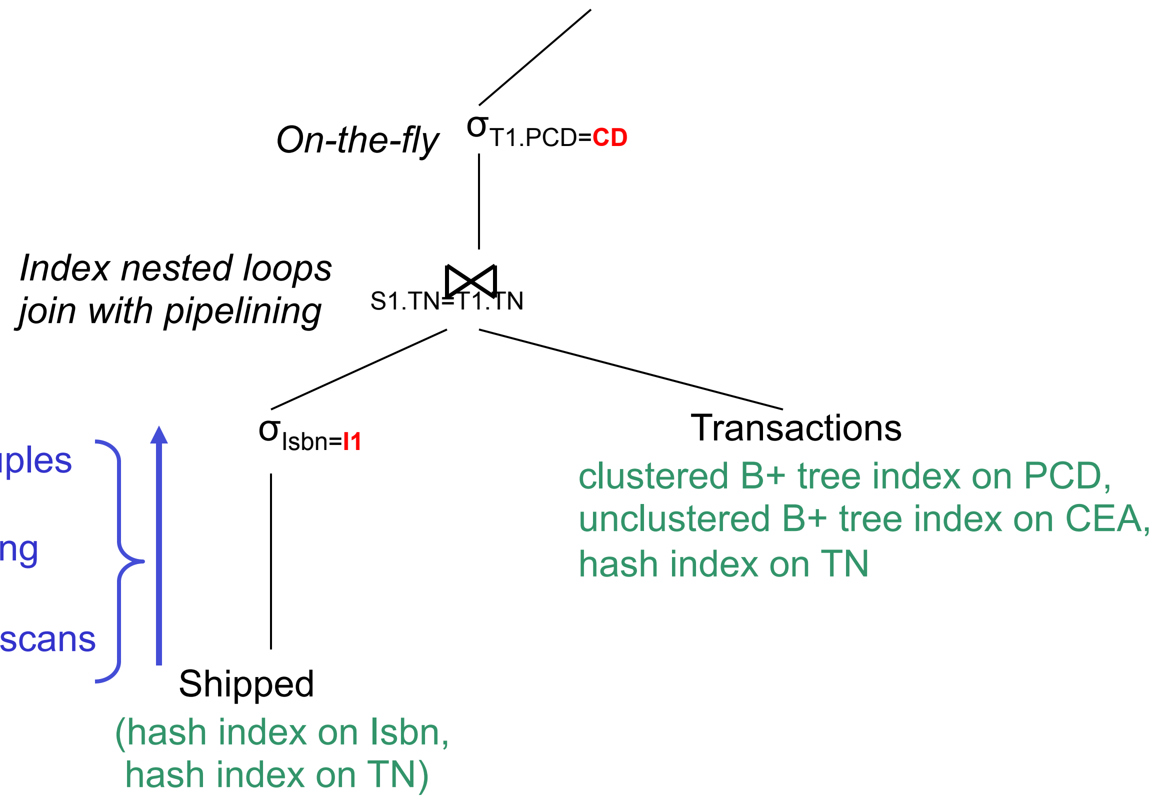
Exercise: can you find some reference to an "average" or expected number of data pages?

Assume:

- a block/page is 2^{12} bytes (upper range)

- each tuple of Shipped relation/table requires 2^4 bytes
  - ➔ one block/page holds $2^{12}/2^4 = 2^8$ Shipped tuples
- each index on Isbn of form <Isbn, <pageid, slot#>> requires 2^3 bytes
  - ➔ each block/page holds $2^{12}/2^3 = 2^9$ indices

- there are 2^{28} tuples in Shipped (*Cardinality*) ➔ $2^{28}/2^8 = \underline{2^{20} \text{ pages} <= \textit{Size} <= 2^{21}} = 2^{28}/2^7$ pages
- there are 2^{20} distinct Isbns in Shipped (*Index Cardinality*) ➔ $2^{28}/2^9 = \underline{2^{19} <= \text{Index Size} <= 2^{20}} = 2^{28}/2^8$



1. Estimated size of result = 2^8 tuples

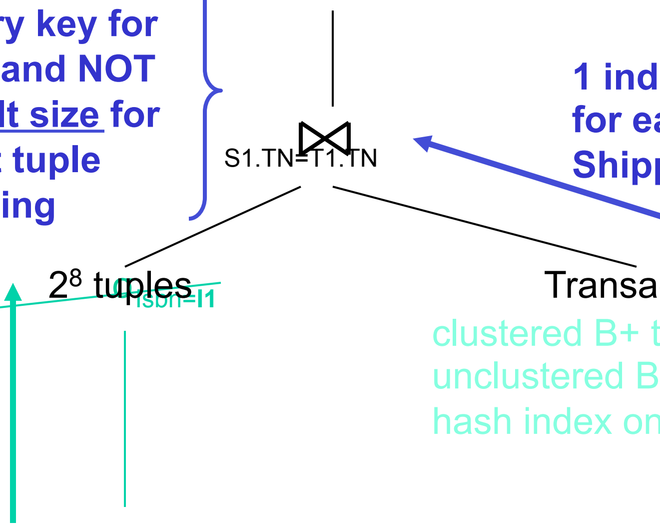
2. Estimated # of page scans using Index on Isbn
   = $\underline{1 + 2^8}$ (worst case) page scans

Assume:

- a block/page is 2^{12} bytes (upper range)
- each tuple of Shipped relation/table requires 2^4 bytes
  - ➔ one block/page holds $2^{12}/2^4 = 2^8$ Shipped tuples
- each index on Isbn of form <Isbn, <pageid, slot#>> requires 2^3 bytes
  - ➔ each block/page holds $2^{12}/2^3 = 2^9$ indices
- there are 2^{28} tuples in Shipped (*Cardinality*) ➔ $2^{28}/2^8$ = 2^{20} pages <= *Size* <= 2^{21} = $2^{28}/2^7$ pages
- there are 2^{20} distinct Isbns in Shipped (*Index Cardinality*) ➔ $2^{28}/2^9$ = 2^{19} <= Index Size <= 2^{20} = $2^{28}/2^8$

In general, a join can increase or decrease the number of tuples, but TN is the primary key for Transactions and TN is a foreign key (and NOT NULL) for Shipped, so estimated result size for join remains 2^8 tuples (but each result tuple is about twice the size of tuples resulting from initial select)

1 index page and 1 data page for each 2^8 tuples from σ on Shipped



1. Estimated size of result = 2^8 tuples

2. Estimated # of page scans using Index on Isbn
   = 1 + 2^8 (worst case) page scans

Shipped
(hash index on Isbn,
hash index on TN)

Transactions:
clustered B+ tree index on PCD,
unclustered B+ tree index on CEA,
hash index on TN

Estimate the expected result size and worst case page scans for this operation.
What additional informaion do you need to know?

On-the-fly

$\sigma_{T1.PCD=\mathbf{CD}}$

In general, a join can increase or decrease the number of tuples, but TN is the primary key for Transactions and TN is a foreign key (and NOT NULL) for Shipped, so expected result size for join remains $\underline{2^8}$ tuples (but each result tuple is about twice the size of tuples resulting from initial select)

2^8 tuples

$\bowtie_{S1.TN=T1.TN}$

(exercise: can you find a reference to lower expected cost stemming from possibil of Transaction Index or data pages being in page buffer?)

$(1+1)2^8 = \underline{2^9}$ page scans (worst case)

$\sigma_{Isbn=\mathbf{I1}}$

Transactions

clustered B+ tree index on PCD,
unclustered B+ tree index on CEA,
hash index on TN

Estimated # of page scans using
Index on Isbn
= $\underline{1 + 2^8}$ (worst case) page scans

Shipped

(hash index on Isbn,
hash index on TN)

Total estimated page scans so far:
$1 + 2^8 + 2^9$

1. Finish estimating the total cost of the example plan (found on slide 3).

2. Give 2 alternative left deep plans for the sample query.

3. Estimate the cost of these alternative left deep plans (remember: the index and other catalog assumptions will remain the same!!)

On Selecting Indexes

Selection of indexes should be informed by the frequency of

- queries
- inserts, deletes, and updates

that you expect will be run on the database.

If we were just worried about queries (SELECTs) then we might well index everything, but

inserts, deletes, and updates can be more costly with stupid indexes, since each indexing structure must also be revised when table entries are revised.

Thus we might be more liberal in our use of indexes in a table where inserts and deletes are relatively rare (e.g., the Books table in one of our illustrative databases), than in the Transactions table where inserts are frequent.

Professor Widom spoke of sophisticated software that could select indexes automatically given a set of queries, inserts, deletes, updates (call this set O for "operations").

Roughly speaking, this software will select an index if the expected cost associated with using the index is less than the expected cost of NOT using it, or:

$$\text{ExpectedCostSavings(Index I)} = \Sigma_O \, P(O)[\text{Cost}(O,\sim I) - \text{Cost}(O,I)],$$

where P(O) is the estimated proportion of time O is executed over all operations in the workload;

- Cost(O,~I) is an estimate of the cost of executing O without the index, I; and
- Cost(O,I) is an estimate of O's cost with I.

You can imagine that to be more accurate, this software would consider the effect of multiple indexes simultaneously, rather than considering them independently as above, so if you've had the AI class before, you can probably see the relevance to some of the methods studied there – search, constraints, optimization, planning.