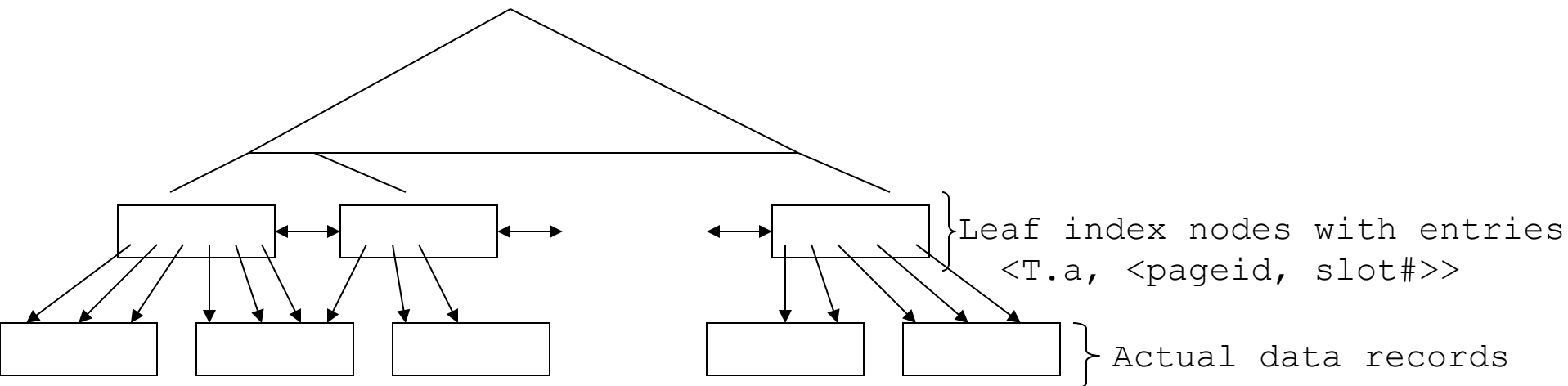
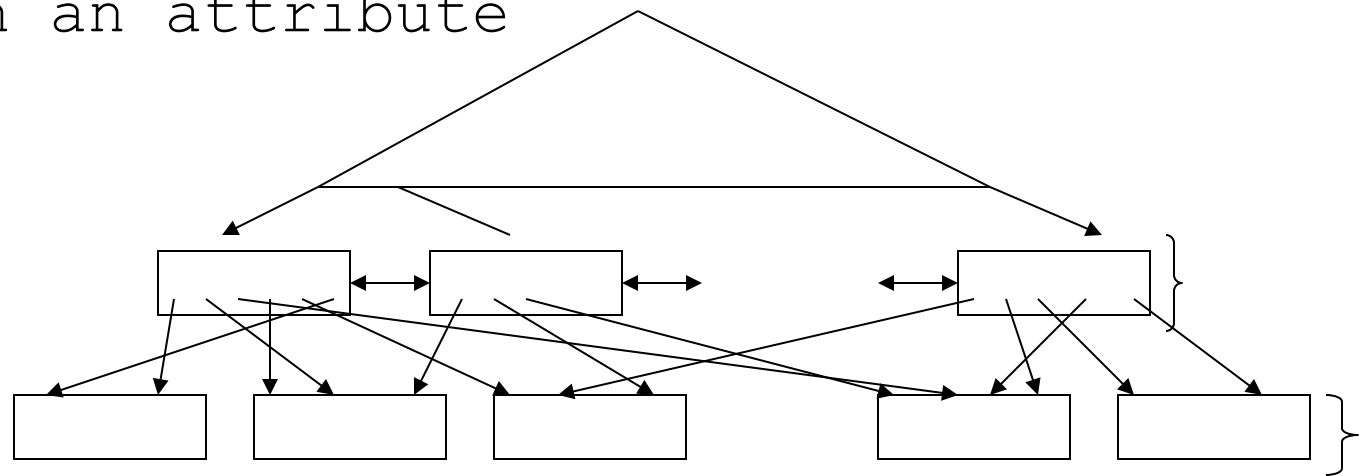


Evaluation of relational operators

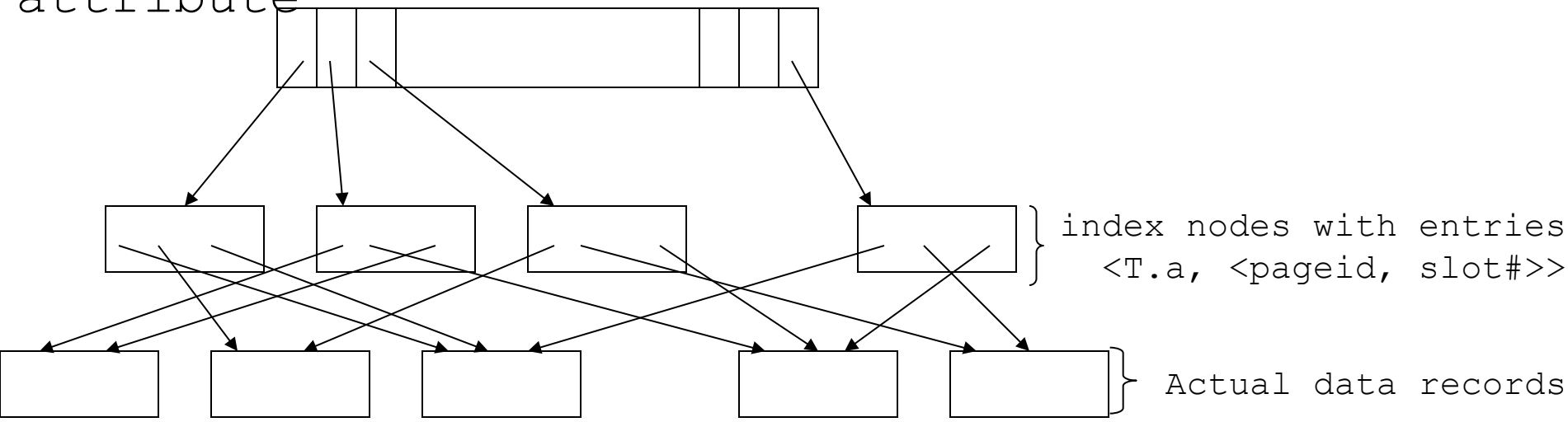
- 1) A file (data records for a table) may be unsorted (with no index)
- 2) A file may be sorted by the values of one attribute (with no index)
- 3) We can have a clustered B+ tree index for the file on an attribute



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
```

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

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

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

- 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:

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

```
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.

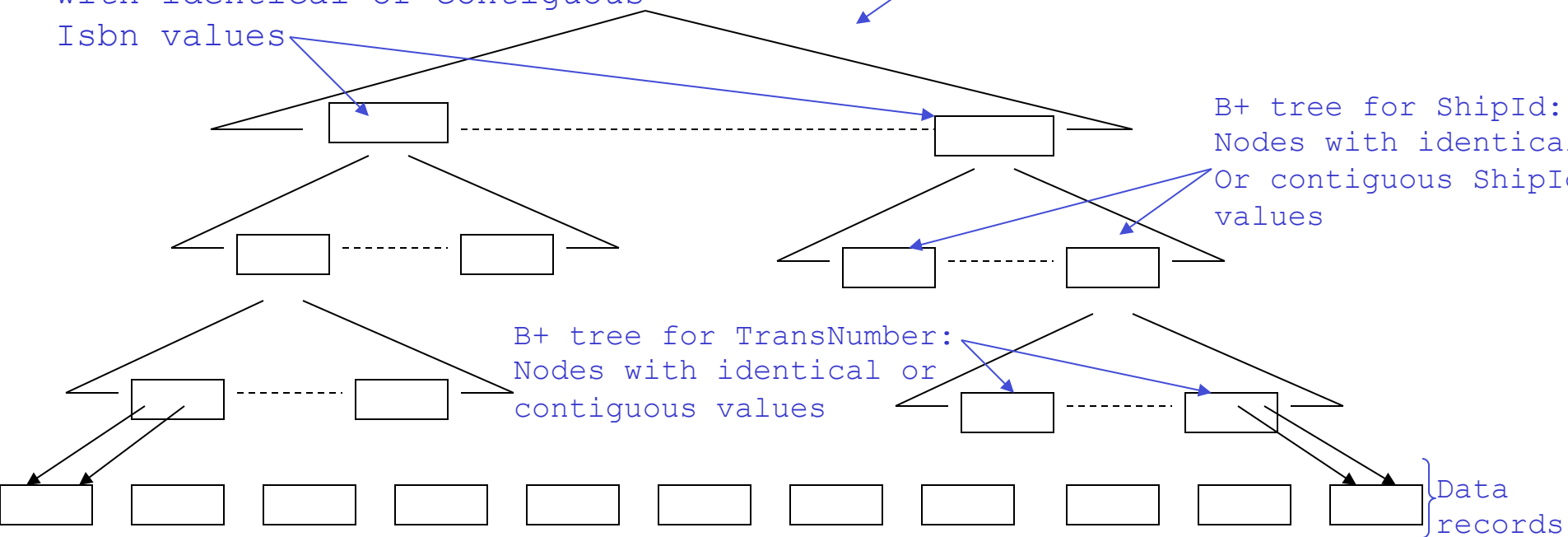
5) Clustered composite B+ tree on (Isbn, ShipId, TransNumber):

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

B+ tree for Isbn: nodes
with identical or contiguous
Isbn values

B+ tree for ShipId:
Nodes with identical
Or contiguous ShipId
values

B+ tree for TransNumber:
Nodes with identical or
contiguous values



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

Consider the queries:

$$\pi_{\text{Isbn, ShipId}}(\text{Shipped})$$

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

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

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

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

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

How might sorting be used?

How might hashing be used?

Consider the query:

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

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

Shipped \bowtie Transactions

JoinResult \leftarrow Empty

For each tuple, s , in Shipped

For each tuple, t , in Transactions

If ($s.TN=t.TN$) add $s+t$ to JoinResult

$$S \bowtie T$$

$$(s \ R \ t)$$

```

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  $\sigma_{(sRt)}(T)$ 
        add s+t to JoinResult

```


Consider the query:

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

Shipped  Transactions
S.TN=T.TN

Shipped  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

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

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

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

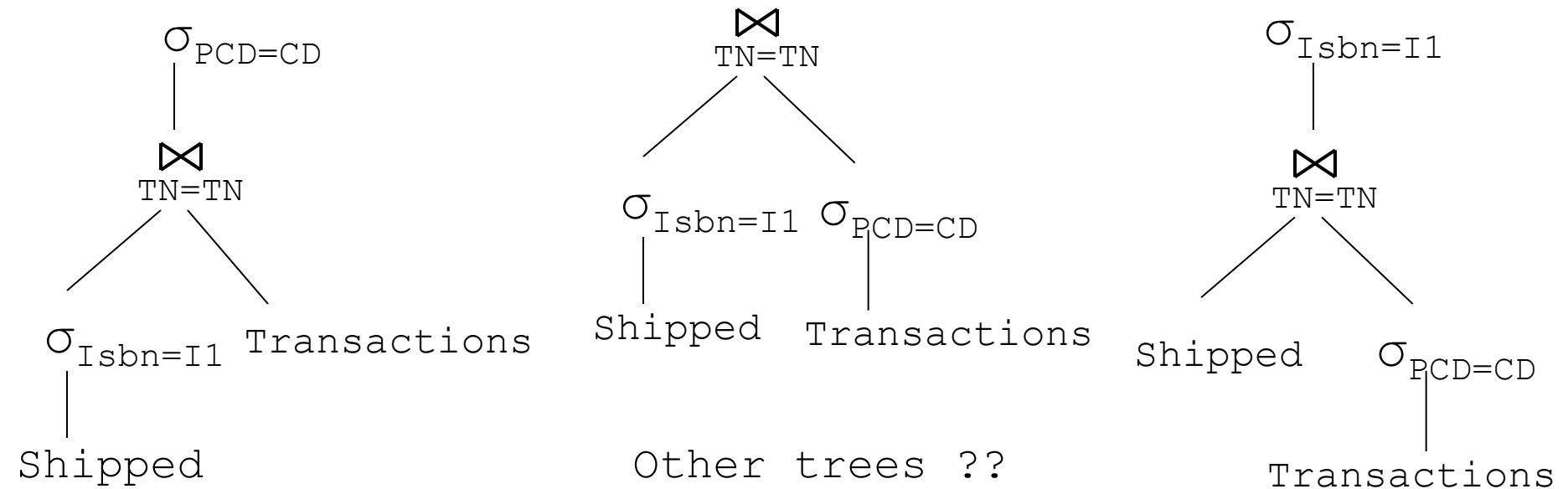
Other possibilities?

```

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

```

Query Evaluation Trees



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

Consider the following Query in SQL and relational algebra:

```
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