

SQLServerFast.com

Execution Plan Video Training

Block 1: Understanding execution plans

Level: Advanced

Chapter 3: Order of data in the data stream

Order of data in the data stream

Order of data in the data stream

Often irrelevant

Easier for the query optimizer

More options available

No extra work needed

Order of data in the data stream

Order of data in the data stream

Often irrelevant

Sometimes required

When ORDER BY is used

Data returned to client has to be in correct order

Order of data in the data stream

Order of data in the data stream

Often irrelevant

Sometimes required

When ORDER BY is used

Some operators require sorted input

May be the only operator for a specific task

Segment operator, for OVER clause

Guaranteeing sorted input cannot be avoided in such a case



Segment

Order of data in the data stream

Order of data in the data stream

Often irrelevant

Sometimes required

When ORDER BY is used

Some operators require sorted input

May be the only operator for a specific task

Or may have alternatives

Merge Join requires ordered input

Other join operators don't

Alternatives are typically more expensive

Trade-off versus overhead of sorting the input data

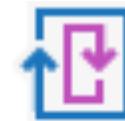
Merge Join only chosen when sorting is "cheap enough"



Adaptive Join
(Inner Join)



Hash Match
(Inner Join)



Nested Loops
(Inner Join)

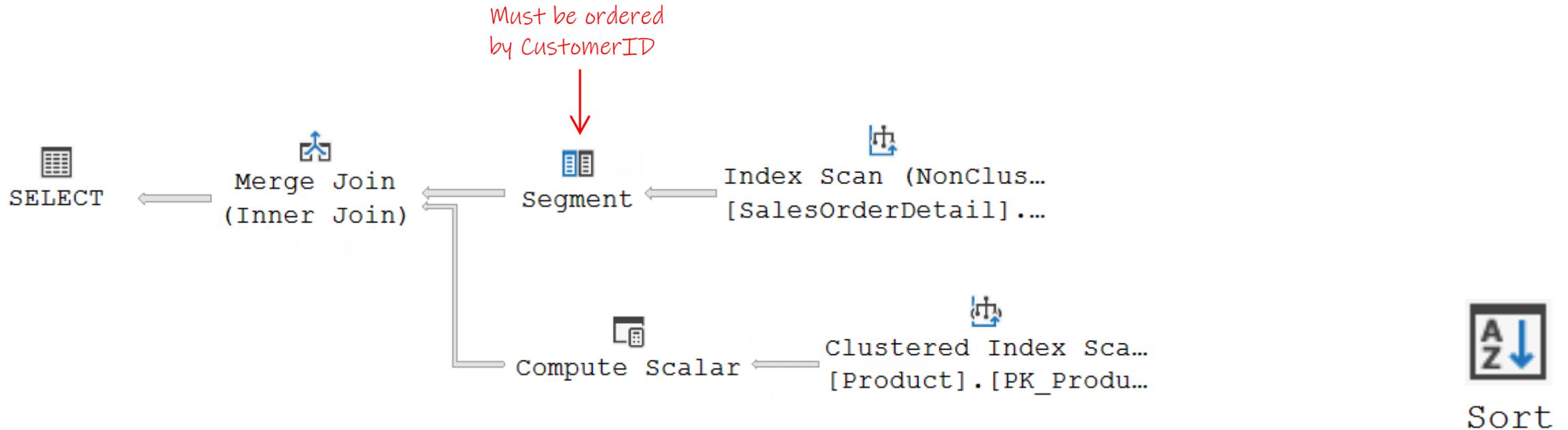


Merge Join
(Inner Join)

Order of data in the data stream

Ways to control the order

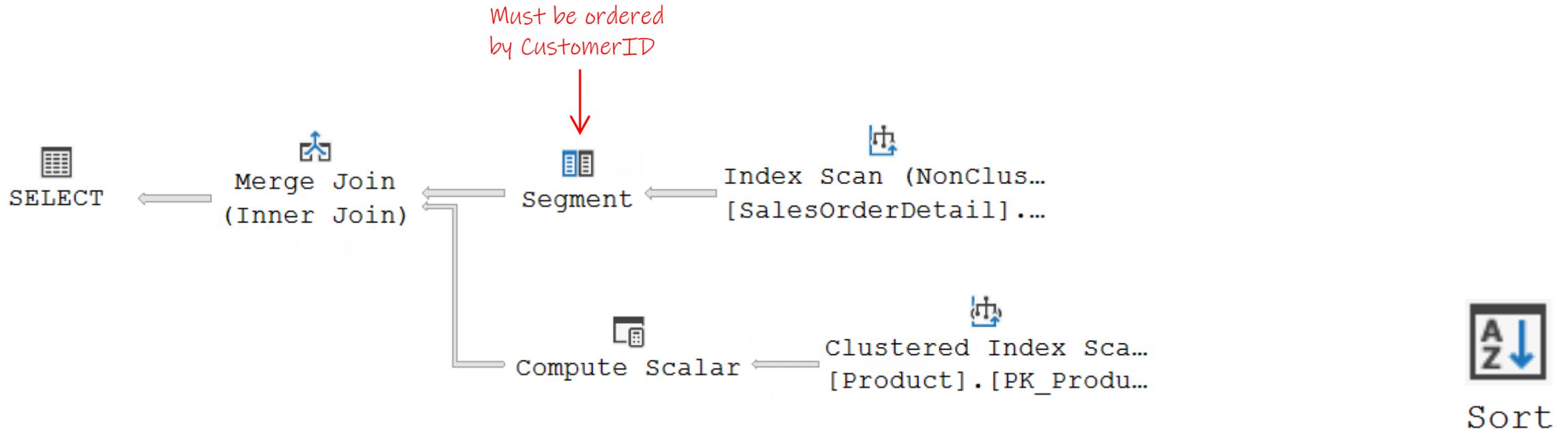
Use a Sort operator



Order of data in the data stream

Ways to control the order

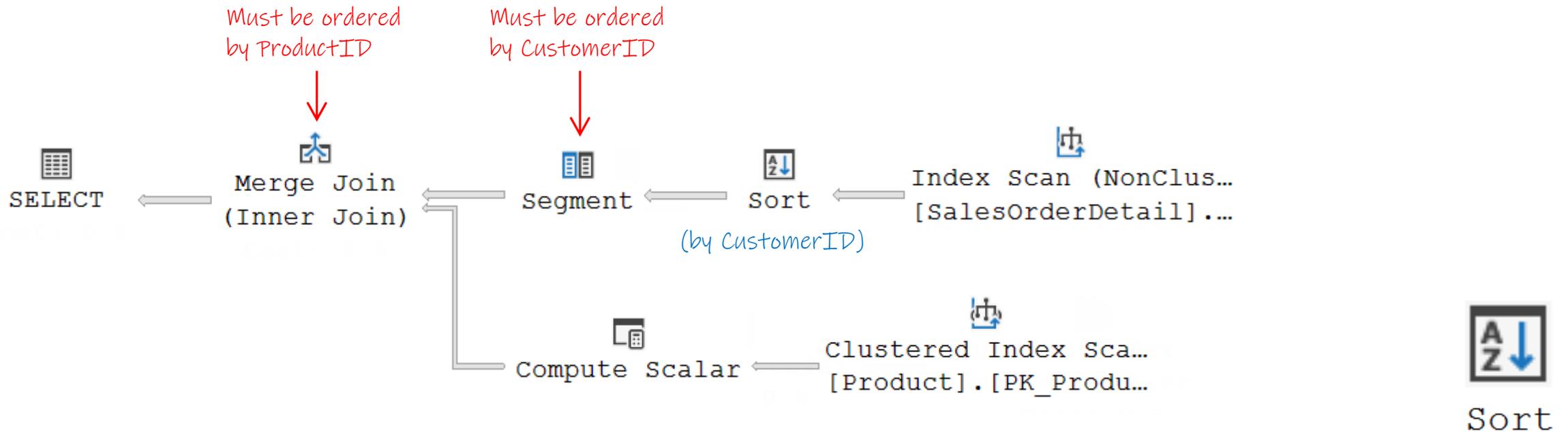
Use a Sort operator



Order of data in the data stream

Ways to control the order

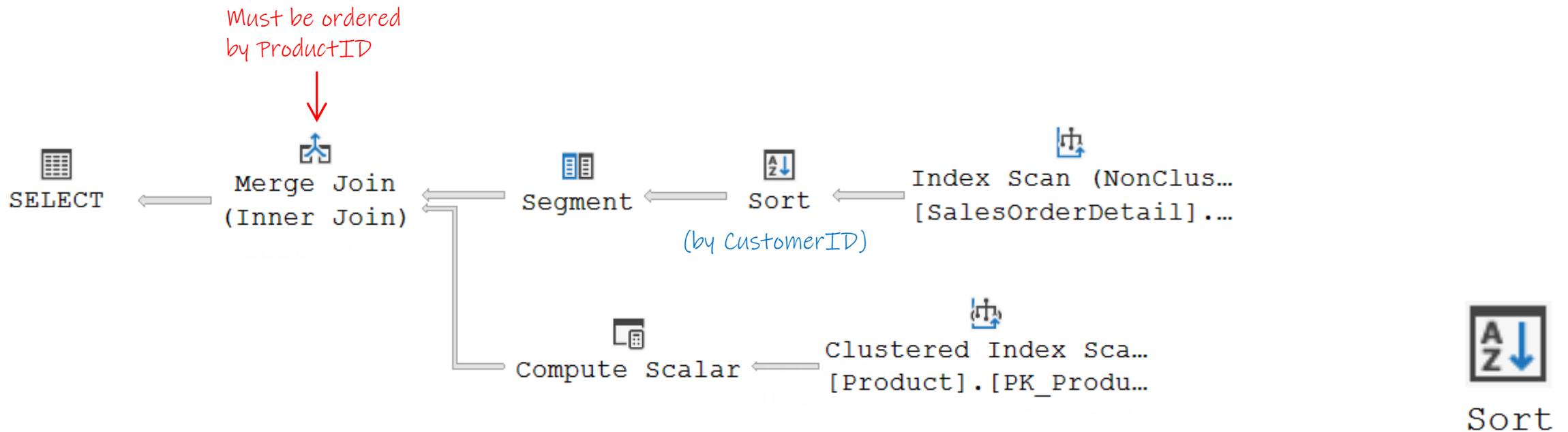
Use a Sort operator



Order of data in the data stream

Ways to control the order

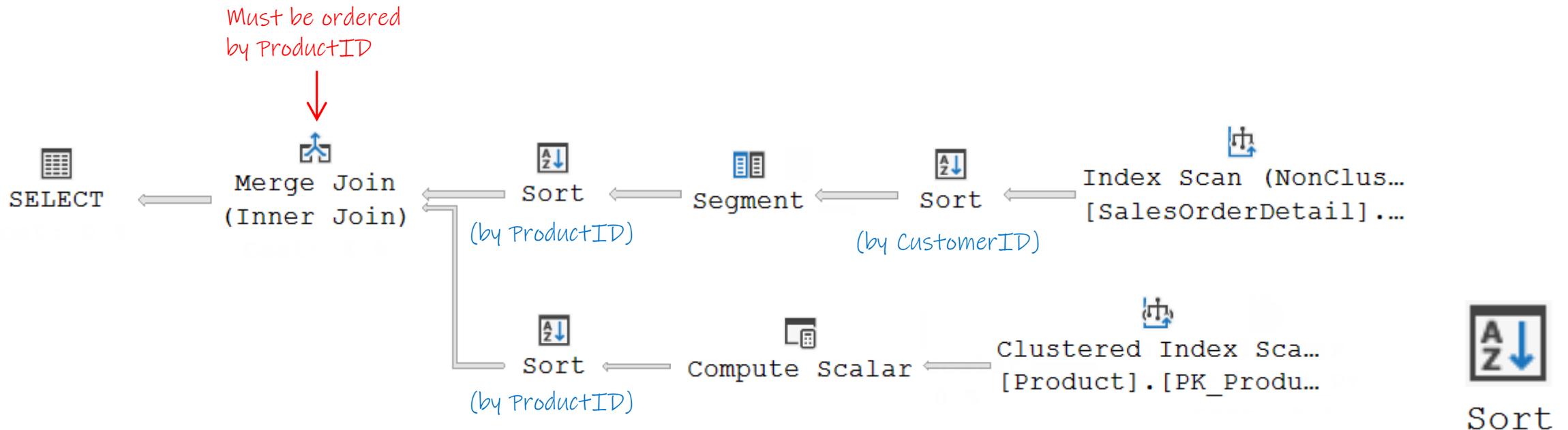
Use a Sort operator



Order of data in the data stream

Ways to control the order

Use a Sort operator



Order of data in the data stream

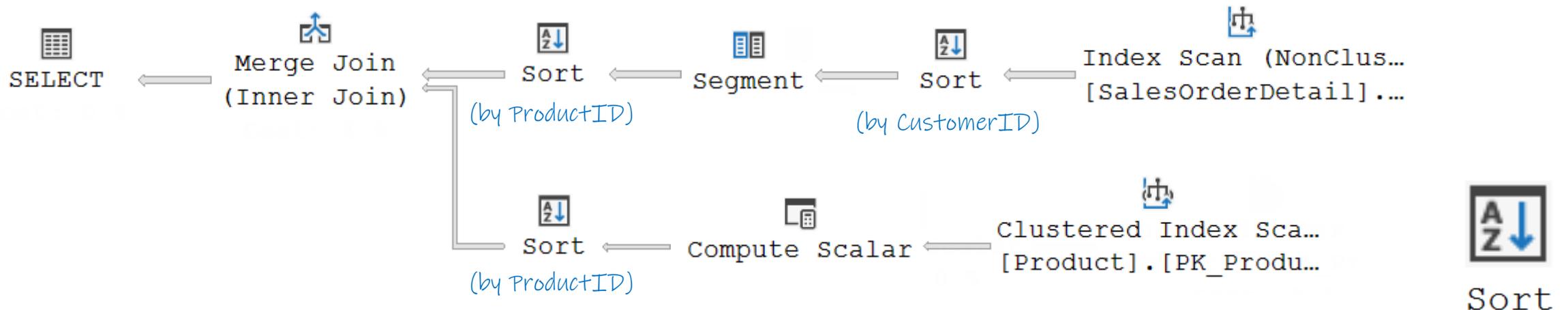
Ways to control the order

Use a Sort operator

Expensive operator

Memory, CPU, potential I/O

Optimizer tries to avoid this!

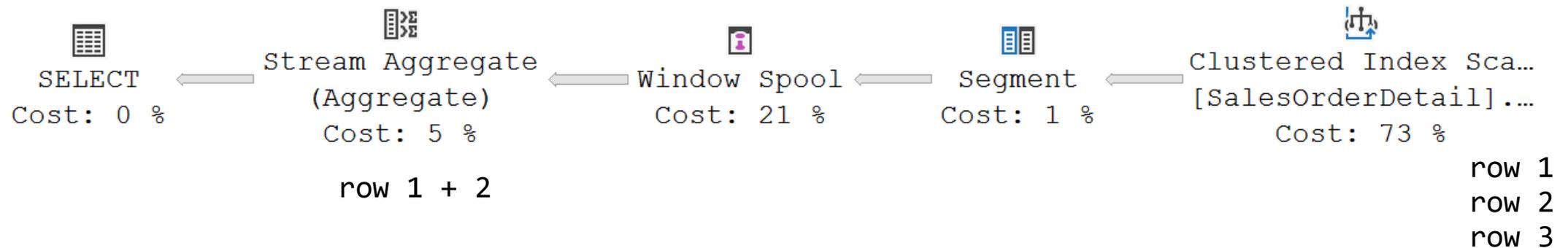


How operators interact with order

A typical* operator

Does work when GetNext() is called

1. Call GetNext() of child
2. Do “something” on row received
3. Return the (modified) row



* Not all operators behave like this (but a lot of them do)

How operators interact with order

Order-preserving operators

Get a row, do something, return the row

Other operators

Logical ordering of output matches logical ordering of input

How operators interact with order

Order-preserving operators

Order-imposing operators

Forces a specific logical ordering of output, independent of input

Example: Sort

Example: Index Scan

Guarantees output order when *Ordered* property is true

Output in this case considered **ordered** (by index columns)

No guarantees when *Ordered* is false

Output *may* still be ordered, but not guaranteed

Output in this case considered **unordered**



Index Scan (NonClustered) Sort

How operators interact with order

Order-preserving operators

Order-imposing operators

Other operators

Example: Hash Match (Aggregate)

Logical ordering of output is undetermined

Considered **unordered**



Hash Match
(Aggregate)

How operators interact with order

Order-preserving operators

Order-imposing operators

Other operators

Example: Hash Match (Aggregate)

Example: Hash Match (Inner Join)

Usually preserved order of rows from lower input

When out of memory, algorithm changes and order is different

Order-preserving behavior is therefore not guaranteed

Output has to be considered **unordered**



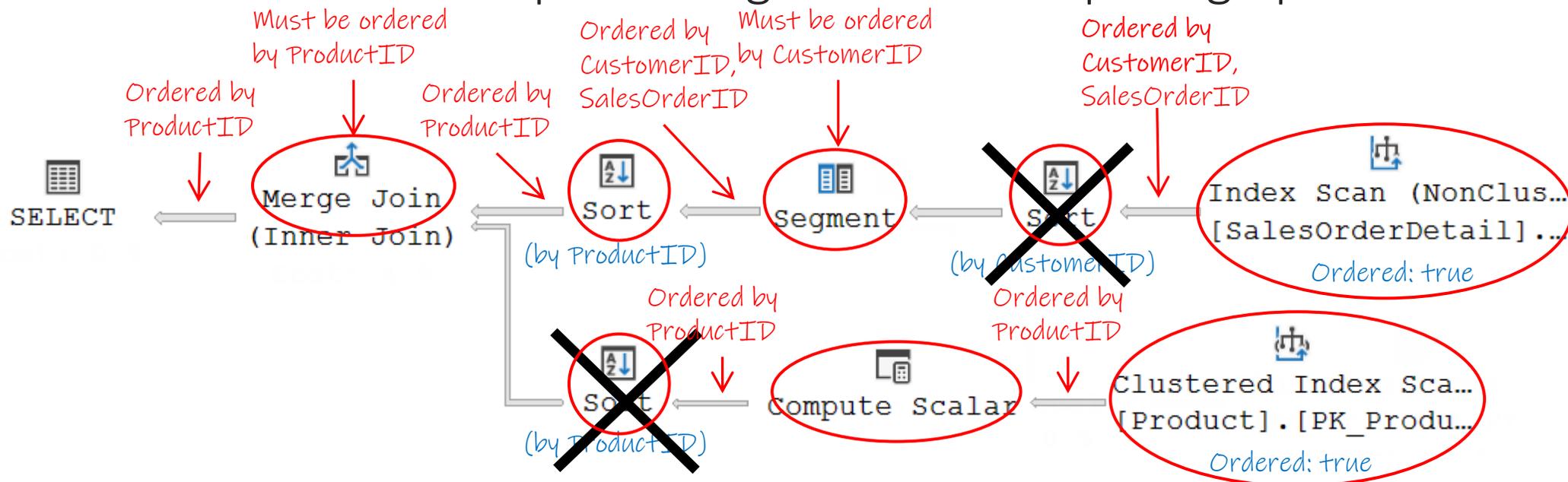
Hash Match
(Inner Join)

Order of data in the data stream

Ways to control the order

Use a Sort operator

Benefit from order-preserving and order-imposing operators



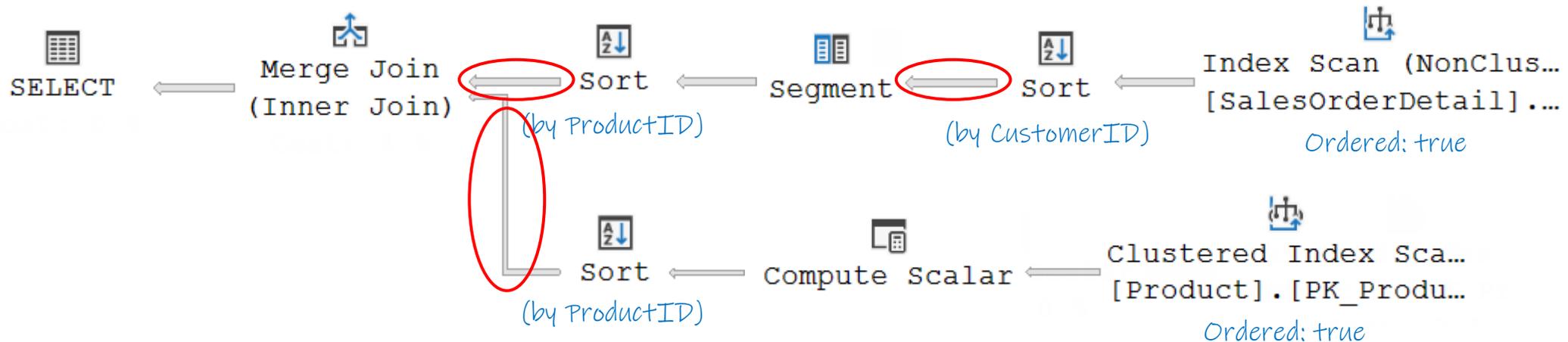
Order of data in the data stream

Ways to control the order

Use a Sort operator

Benefit from order-preserving and order-imposing operators

Avoid Sort when possible



Order of data in the data stream

Ways to control the order

- Use a Sort operator

- Benefit from order-preserving and order-imposing operators

 - Avoid Sort when possible

 - Move Sort to location with cheapest rows, smallest rows

 - Before joining to table with one to many relationship

 - After filtering rows

 - When possible, reorder other operators to maximize effect

- Use other order-requiring operators if ordering is needed anyway

 - When needed and possible, maximize effect by reordering operators

Summary

Sorting data

- Sort operators: expensive

- Other order-imposing operators: often cheaper

- Order-preserving operators give freedom where to sort

 - Sort once, benefit multiple times

 - Sort where amount of data is low

Next chapters

Chapter 4: Missing nodes

- NodeID: Unique number for each operator

- Sometimes missing

- Can cause unusual effects

Chapter 5: Batch mode versus row mode