

# SQLServerFast.com

## Execution Plan Video Training

Block 3: Combining data

Level: Advanced

Chapter 4: Adaptive Join (advanced)

# Adaptive Join

## Join types supported

### Adaptive Join

Inner Join

Left / ~~Right~~ / ~~Full~~ Outer Join

Left / ~~Right~~ Semi Join

Left / ~~Right~~ Anti Semi Join

~~Left Semi Join (probed)~~

~~Concatenation~~

~~Union~~

### Hash Match

Inner Join

Left / Right / Full Outer Join

Left / Right Semi Join

Left / Right Anti Semi Join

~~Left Semi Join (probed)~~

~~Concatenation~~

~~Union~~

### Nested Loops

Inner Join

Left / ~~Right~~ / ~~Full~~ Outer Join

Left / ~~Right~~ Semi Join

Left / ~~Right~~ Anti Semi Join

Left Semi Join (probed)

~~Concatenation~~

~~Union~~



Adaptive Join  
(Inner Join)

# Adaptive Join

## Join types supported

- Adaptive Join

  - Inner Join

  - Left Outer Join

  - Left Semi Join

  - Left Anti Semi Join

## Same changes to flowcharts

- For these operations only

- Build phase and probe phase in batch mode

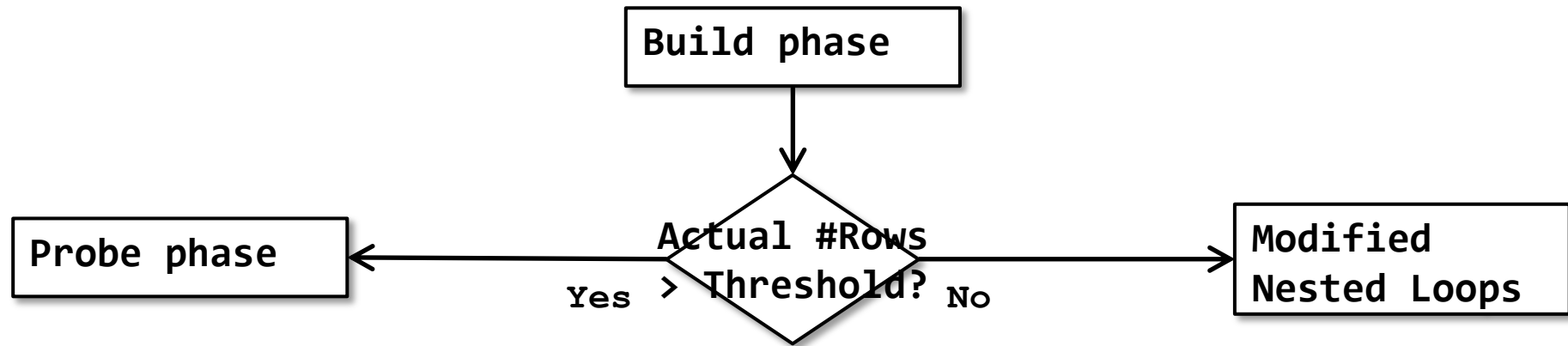
- Modified Nested Loops phase reads from hash table



Adaptive Join  
(Inner Join)

# Adaptive Join

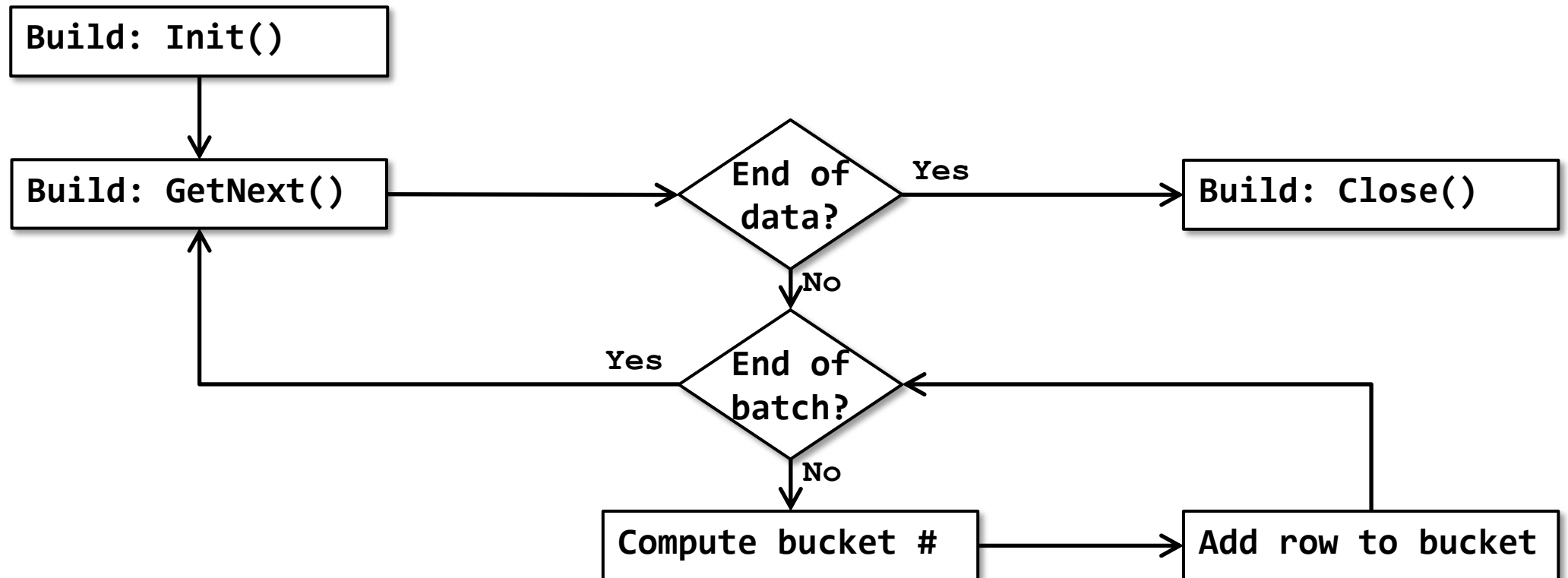
High level logic



# Adaptive Join (inner join, no spill)

Build phase

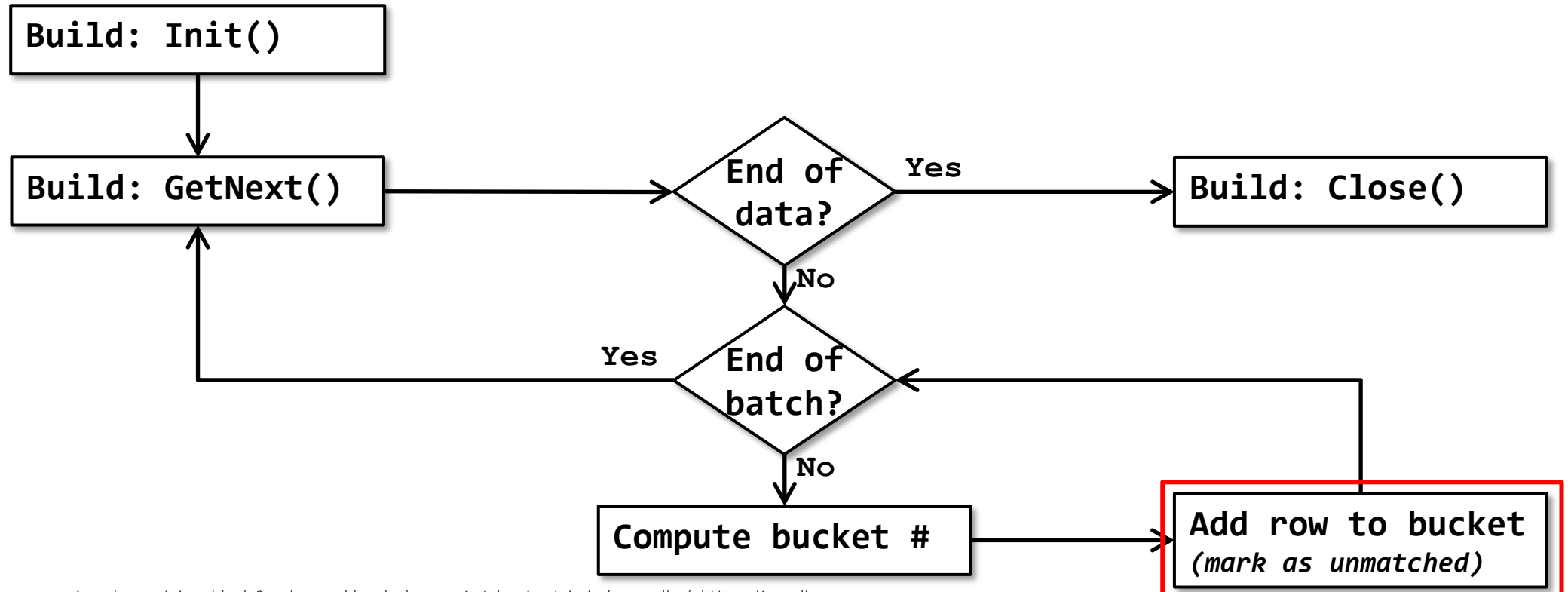
Optimized for batch mode processing



# Adaptive Join (all operations, no spill)

Build phase

Optimized for batch mode processing



# Adaptive Join (all operations, no spill)

## Build phase

- Optimized for batch mode processing

- Heavily optimized

  - Handle standard cases first, handle all exceptions later

  - Many other optimizations

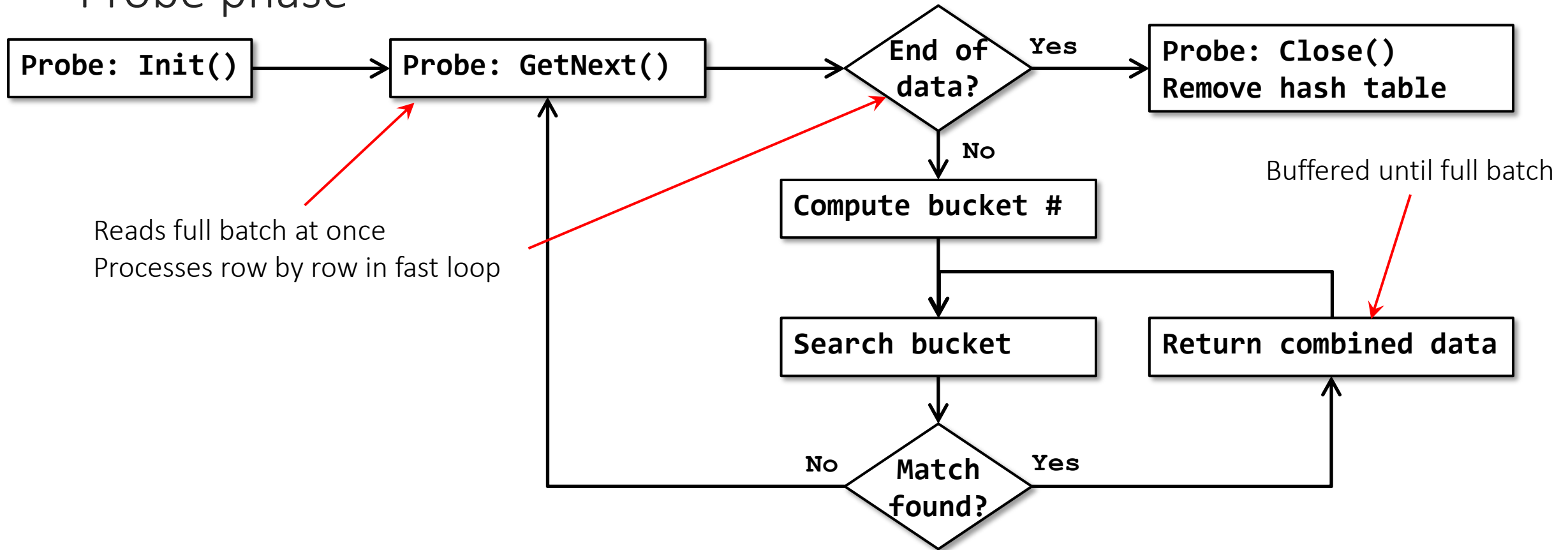
    - Special versions for different CPU architectures

    - (Mostly) undocumented

[https://sqlbits.com/sessions/event2023/Why\\_is\\_Batch\\_Mode\\_Fast](https://sqlbits.com/sessions/event2023/Why_is_Batch_Mode_Fast)

# Adaptive Join (inner join, no spill)

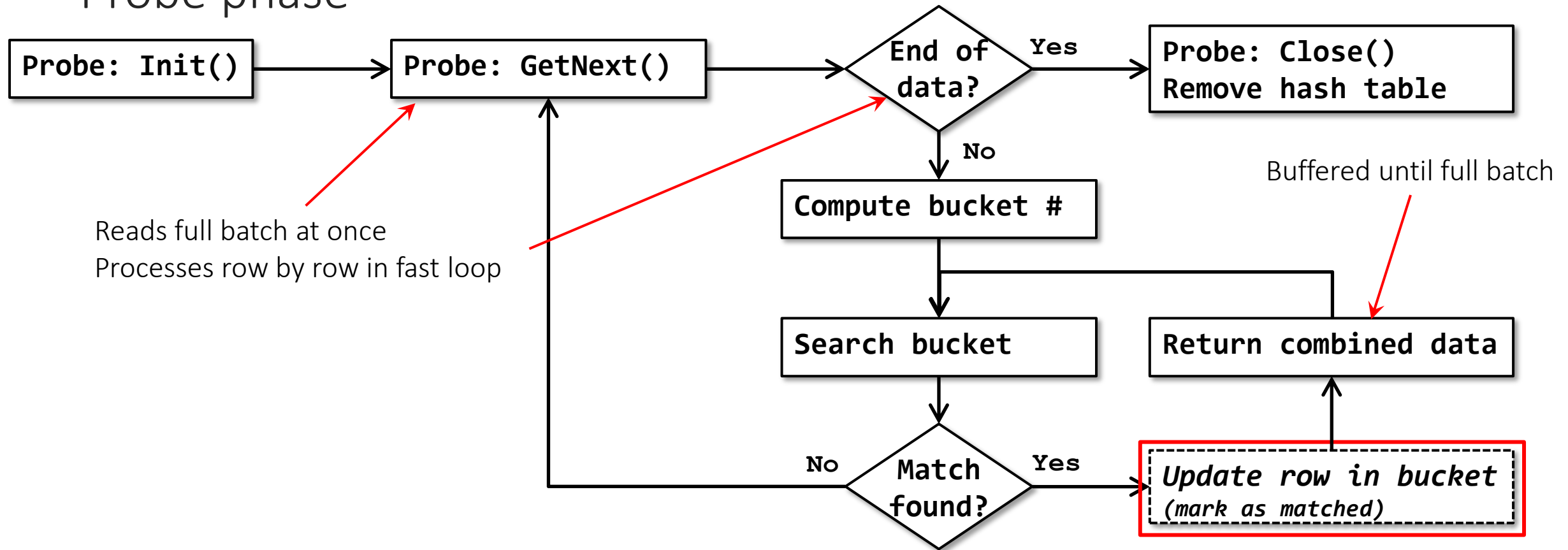
## Probe phase





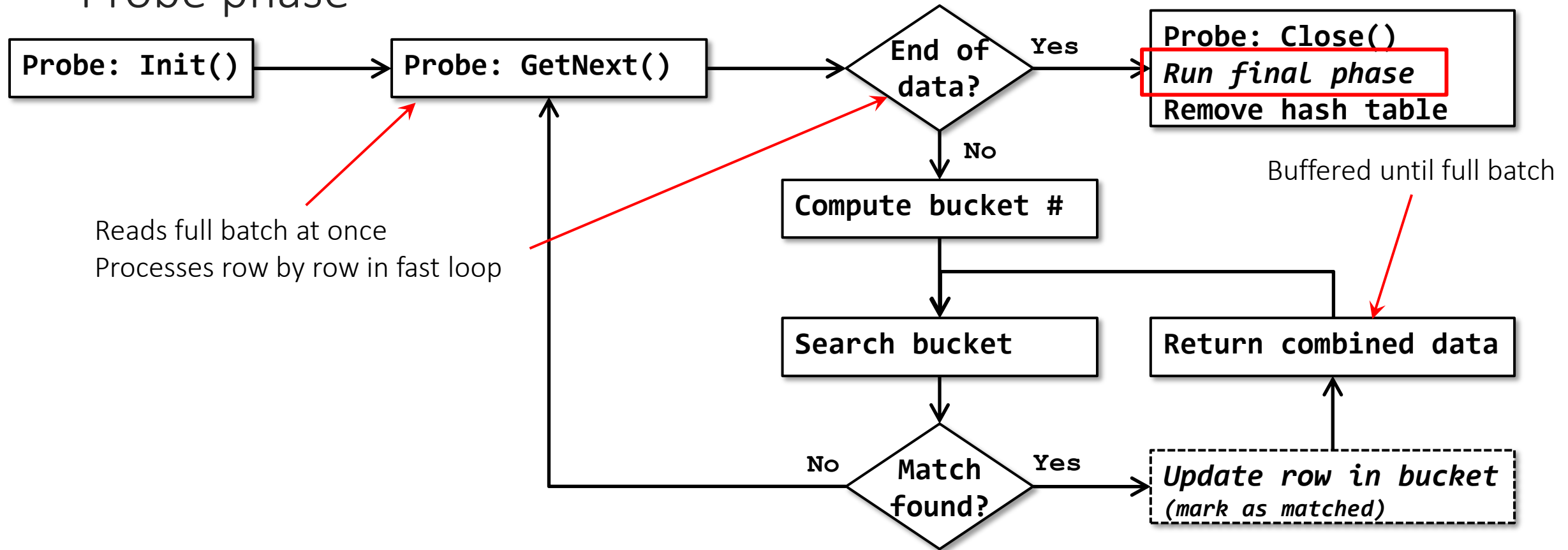
# Adaptive Join (all operations, no spill)

## Probe phase



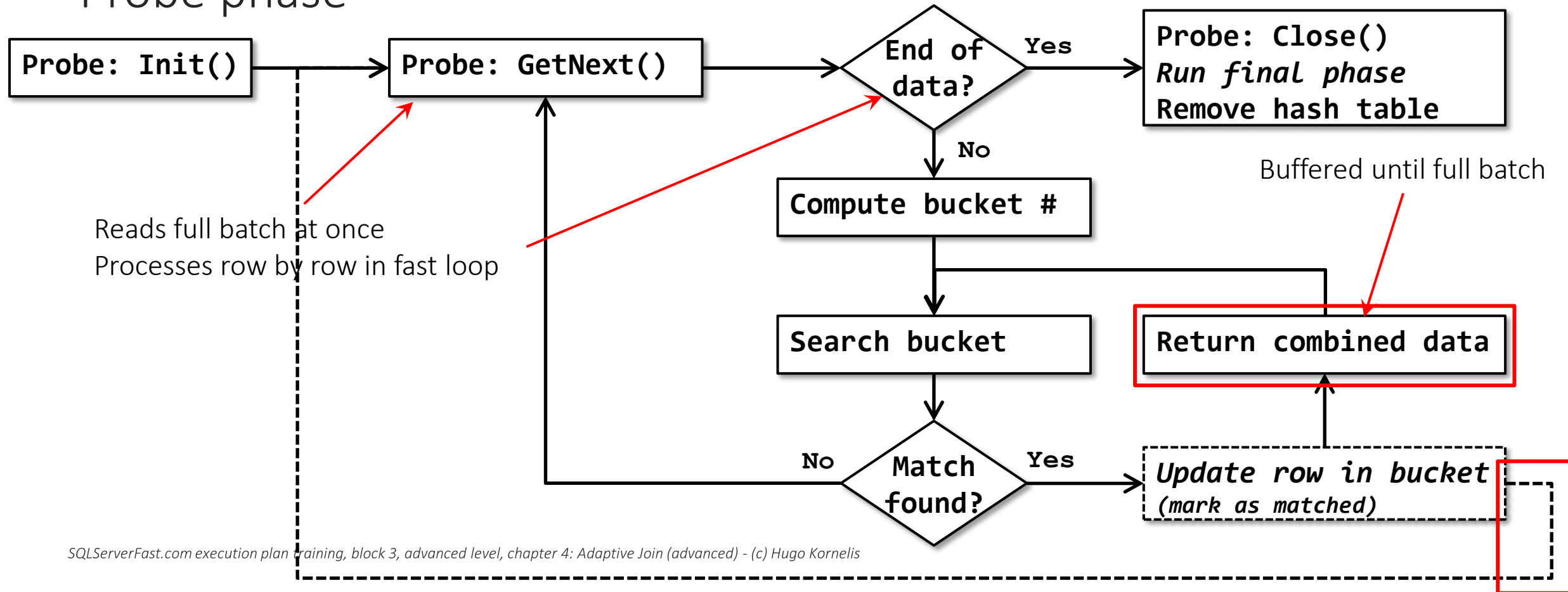
# Adaptive Join (all operations, no spill)

## Probe phase



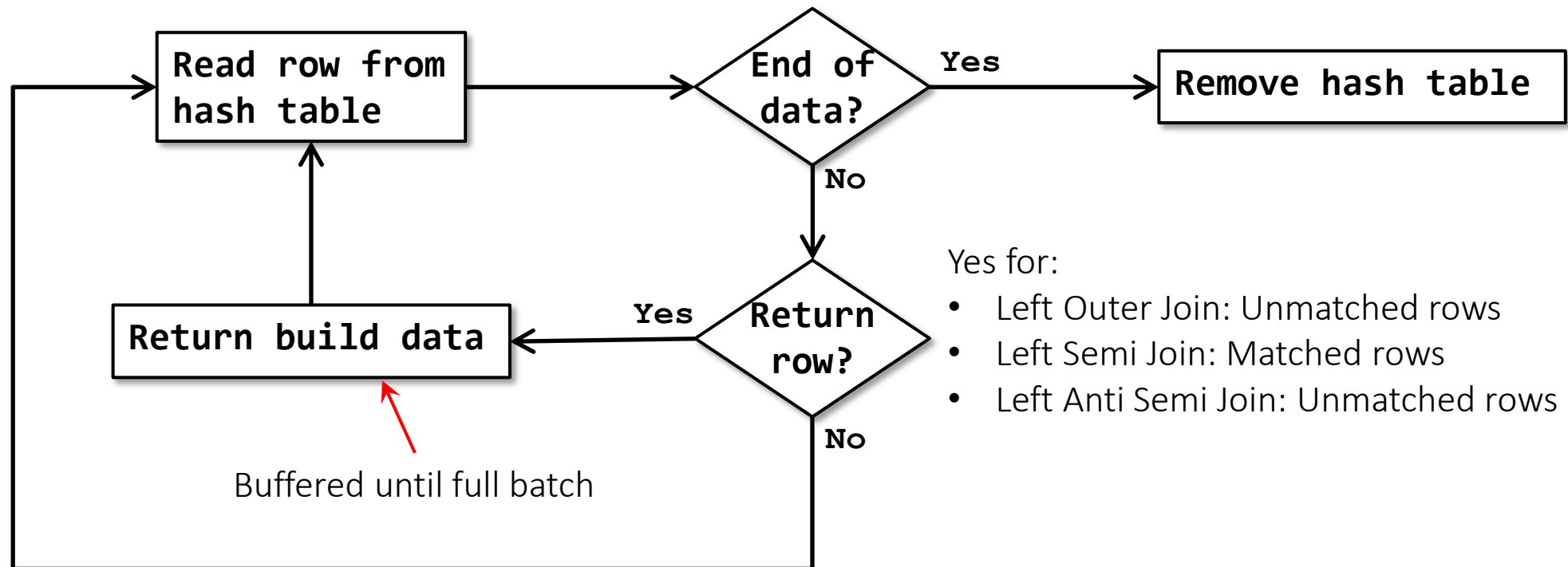
# Adaptive Join (all operations, no spill)

## Probe phase



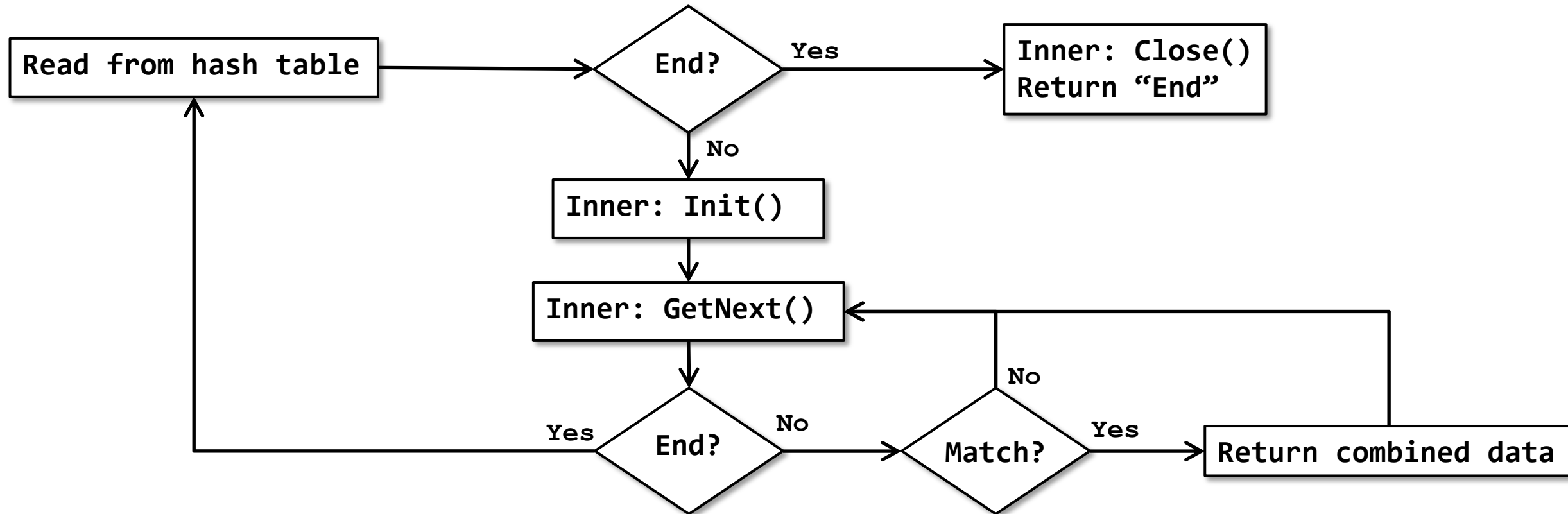
# Adaptive Join (all operations, no spill)

Final phase



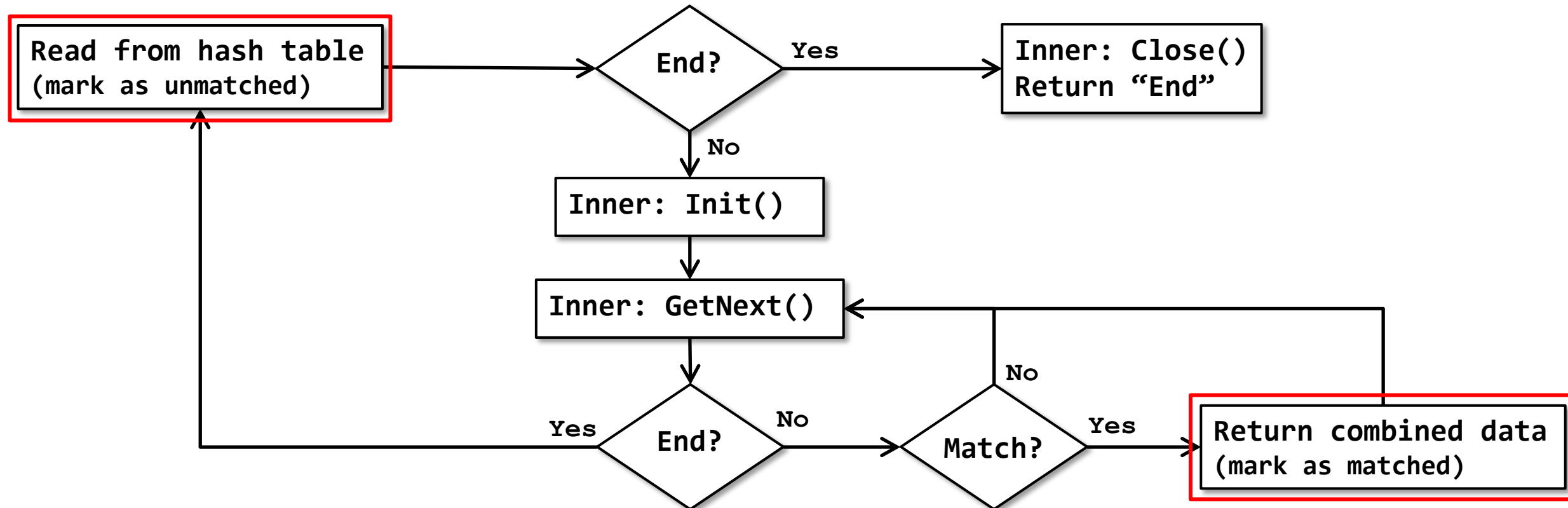
# Adaptive Join (inner join, no spill)

Modified Nested Loops phase



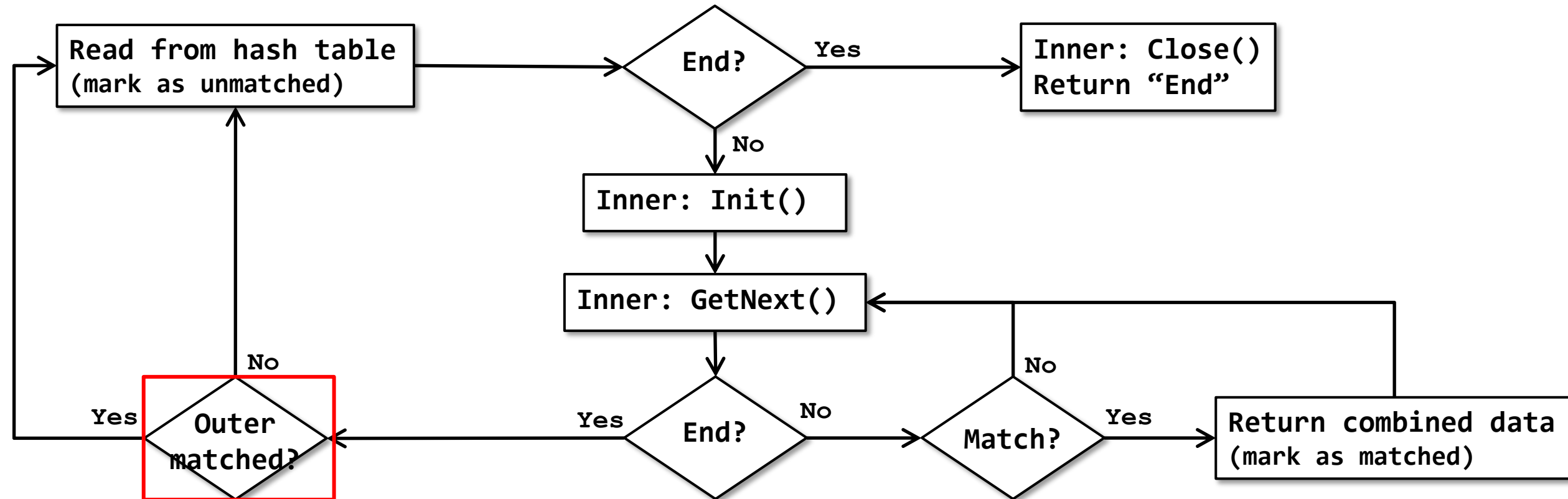
# Adaptive Join (all operations, no spill)

## Modified Nested Loops phase



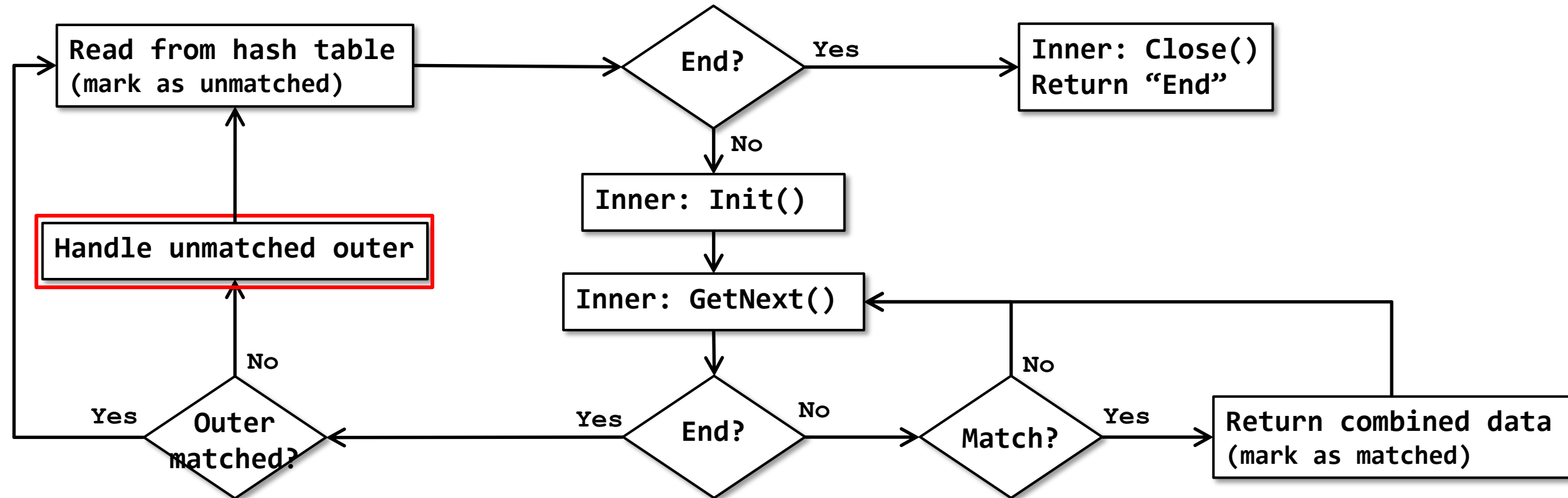
# Adaptive Join (all operations, no spill)

## Modified Nested Loops phase



# Adaptive Join (all operations, no spill)

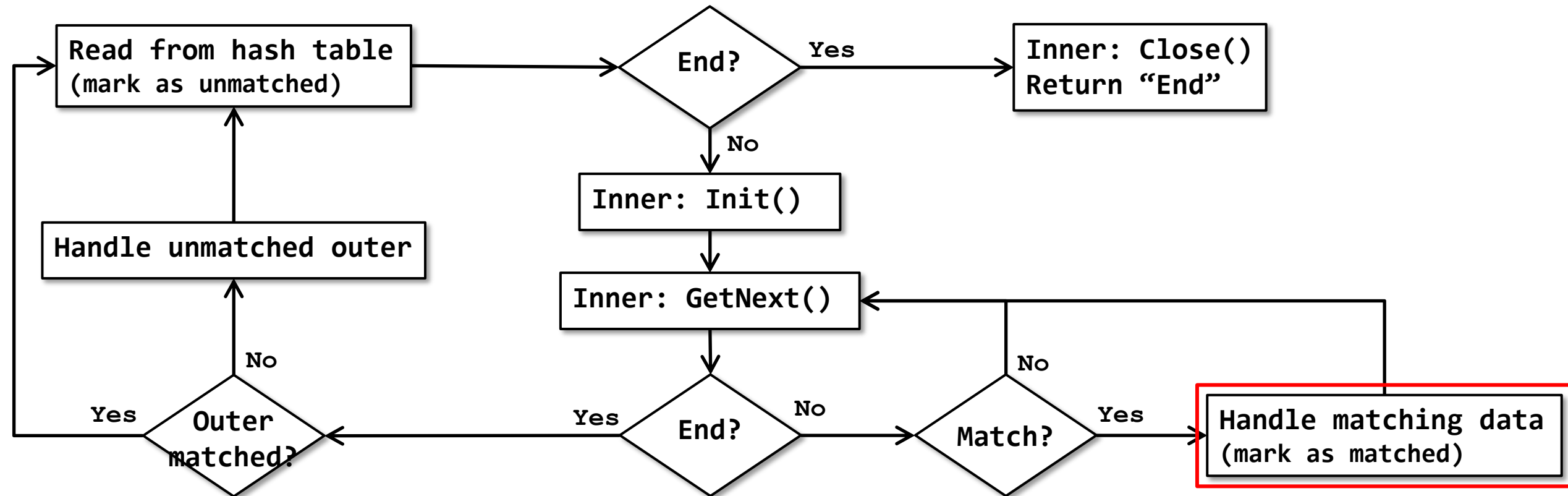
## Modified Nested Loops phase





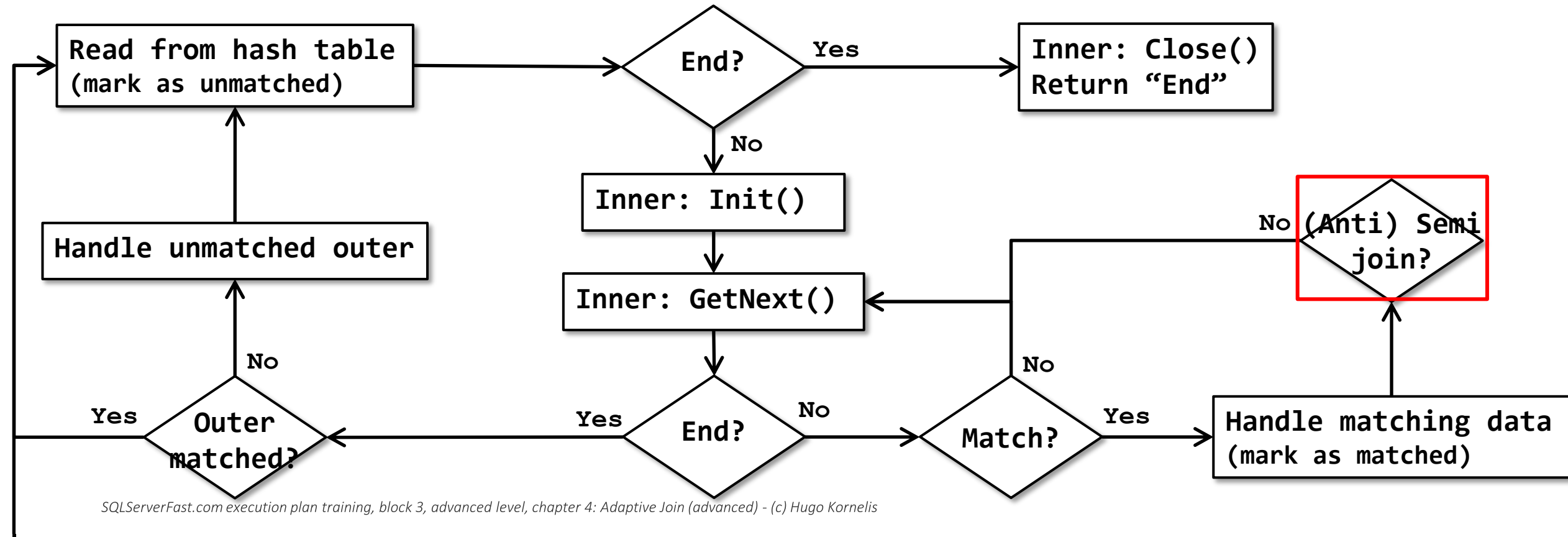
# Adaptive Join (all operations, no spill)

## Modified Nested Loops phase



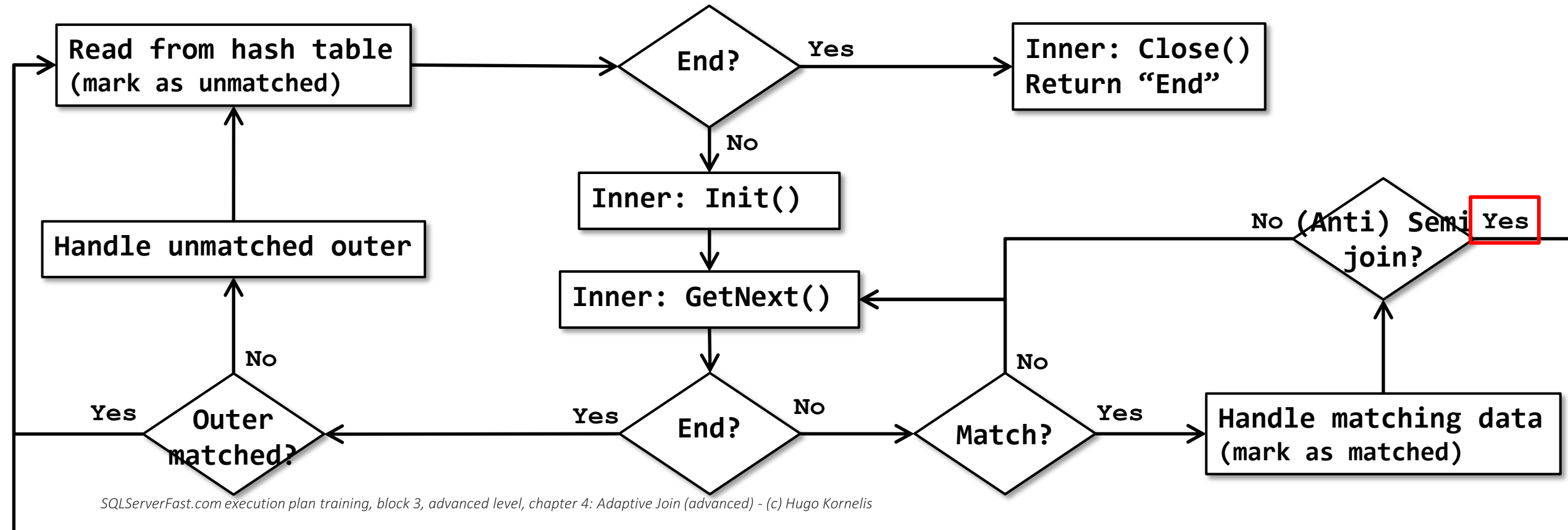
# Adaptive Join (all operations, no spill)

## Modified Nested Loops phase



# Adaptive Join (all operations, no spill)

## Modified Nested Loops phase




# Adaptive Join

## Properties

### Optimized Nested Loops

Optimized Nested Loops optimization  
(when #rows < Adaptive Threshold Rows)

Properties	
Adaptive Join	
	
Misc	
Adaptive Threshold Rows	986,802
BitmapCreator	True
Defined Values	[[ContosoRetailDW].[dbo].[Fact
Description	Chooses dynamically between
Estimated CPU Cost	0,0000902
Estimated Execution Mod	Batch
Estimated I/O Cost	0
Estimated Join Type	HashMatch
Estimated Number of Exe	1
Estimated Number of Row	4509,17
Estimated Number of Row	4509,17
Estimated Operator Cost	0 (0%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	173 B
Estimated Subtree Cost	3,53729
Hash Keys Build	[ContosoRetailDW].[dbo].[Fact
Hash Keys Probe	[ContosoRetailDW].[dbo].[Dim
Is Adaptive	True
Logical Operation	Inner Join
Node ID	0
Optimized	False
Outer References	[ContosoRetailDW].[dbo].[Fact
Output List	[ContosoRetailDW].[dbo].[Fact
Parallel	False
Physical Operation	Adaptive Join
Probe Residual	[ContosoRetailDW].[dbo].[Fact
<b>Adaptive Threshold Rows</b>	
If this is an adaptive operator, the cardinality at which it adapts.	



# Adaptive Join

Properties

Optimized Nested Loops

Prefetching

(Ordered or Unordered)

BitmapCreator

```
-<xsd:complexType name="AdaptiveJoinType">
  -<xsd:annotation>
    -<xsd:documentation>
      The Adaptive Join element replaces a adaptive concat with Hash Join and Nested loops as inputs. This element will have 3
      showplan element.
    -</xsd:documentation>
  -</xsd:annotation>
  -<xsd:complexContent>
    -<xsd:extension base="shp:RelOpBaseType">
      -<xsd:sequence>
        <xsd:element name="HashKeysBuild" type="shp:ColumnReferenceListType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="HashKeysProbe" type="shp:ColumnReferenceListType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="BuildResidual" type="shp:ScalarExpressionType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="ProbeResidual" type="shp:ScalarExpressionType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="StarJoinInfo" type="shp:StarJoinInfoType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="Predicate" type="shp:ScalarExpressionType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PassThru" type="shp:ScalarExpressionType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="OuterReferences" type="shp:ColumnReferenceListType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PartitionId" type="shp:SingleColumnReferenceType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="RelOp" type="shp:RelOpType" minOccurs="3" maxOccurs="3"/>
      -</xsd:sequence>
      <xsd:attribute name="BitmapCreator" type="xsd:boolean" use="optional"/>
      <xsd:attribute name="Optimized" type="xsd:boolean" use="required"/>
      <xsd:attribute name="WithOrderedPrefetch" type="xsd:boolean" use="optional"/>
      <xsd:attribute name="WithUnorderedPrefetch" type="xsd:boolean" use="optional"/>
    -</xsd:extension>
  -</xsd:complexContent>
-</xsd:complexType>
```

<https://schemas.microsoft.com/sqlserver/2004/07/showplan/sql2022/showplanxml.xsd>



Adaptive Join  
(Inner Join)

# Adaptive Join

## Out of memory

### Batch mode

May request additional memory during execution

Will eventually fail → hash spill

Same process as spilling Hash Match

Hybrid / grace hash join  
Recursive hash join

} Data stored in tempdb!

Bit-vector filtering

Dynamic role reversal

Bail-out



Adaptive Join  
(Inner Join)

# Adaptive Join

## Out of memory

### Batch mode

May request additional memory during execution

Will eventually fail → hash spill

Same process as spilling Hash Match

What if #rows < Adaptive Threshold Rows?

Change modified Nested Loops algorithm

Repeat for each partition

Read from spill files directly



Adaptive Join  
(Inner Join)

# Adaptive Join

## Out of memory

### Batch mode

May request additional memory during execution

Will eventually fail → hash spill

Same process as spilling Hash Match

What if #rows < Adaptive Threshold Rows?

~~Change modified Nested Loops algorithm~~

~~Repeat for each partition~~

~~Read from spill files directly~~

Continue with Hash Match algorithm (regardless of #rows)



Adaptive Join  
(Inner Join)



# Adaptive Join

Role in the execution plan

- Batch mode execution plans only

  - Columnstore index

  - Batch Mode on Rowstore

- Top input with unreliable estimate

Two strategies

  - Hash Match

  - Nested Loops



Adaptive Join  
(Inner Join)

# Adaptive Join

Role in the execution plan

- Batch mode execution plans only

  - Columnstore index

  - Batch Mode on Rowstore

- Top input with unreliable estimate

Two strategies

  - Hash Match, efficient for “many” rows

  - Nested Loops, efficient for “few” rows



Adaptive Join  
(Inner Join)

# Adaptive Join

## Role in the execution plan

### Two strategies

Hash Match, efficient for “many” rows

- Single pass

- All required rows

- Some “extra” rows acceptable

Nested Loops, efficient for “few” rows

- Multiple executions

- Low cost per execution

- Return only relevant rows for current top rows

  - Based on *Outer References* property



Adaptive Join  
(Inner Join)

# Adaptive Join

Role in the execution plan

Two strategies

Hash Match, efficient for “many” rows  
(Clustered) Index Scan

Nested Loops, efficient for “few” rows  
(Clustered) Index Seek

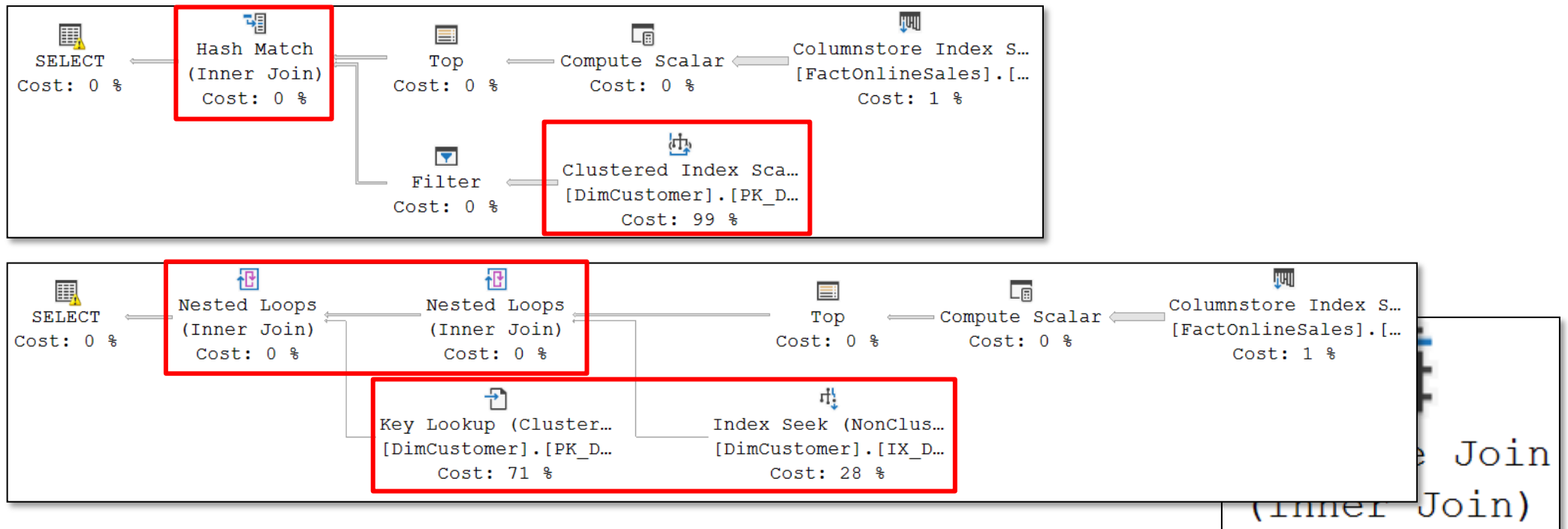


Adaptive Join  
(Inner Join)

# Adaptive Join

Role in the execution plan

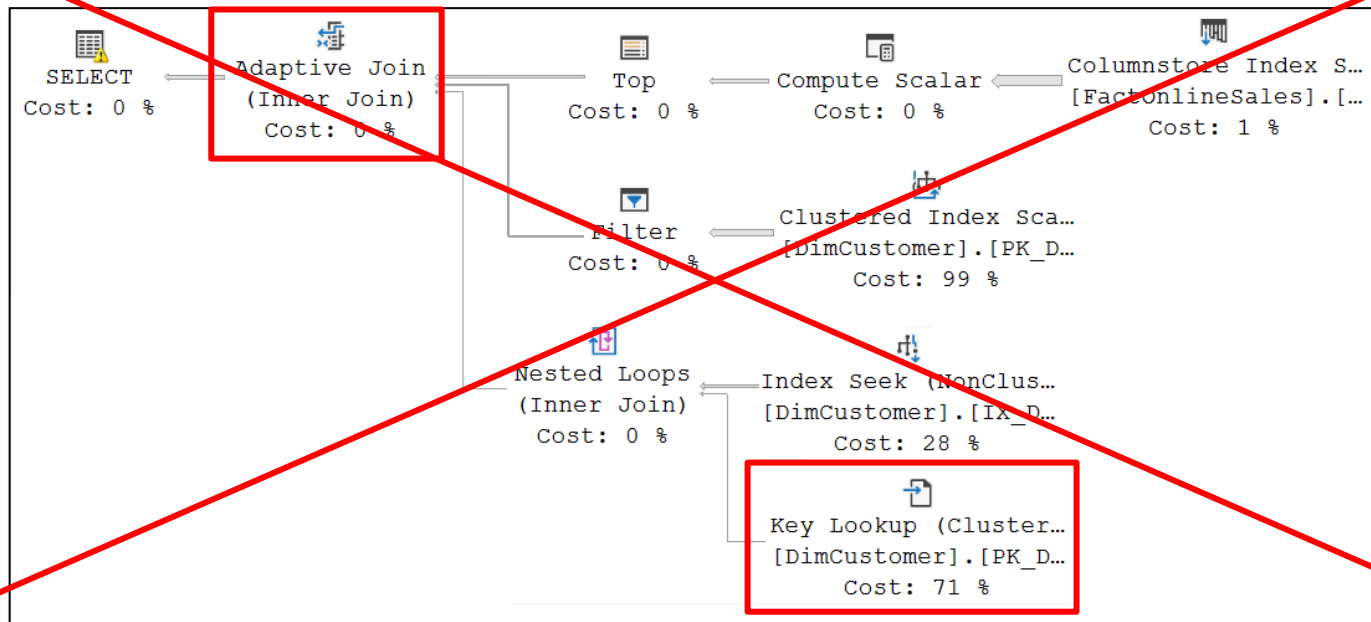
Two strategies



# Adaptive Join

Role in the execution plan

Two strategies

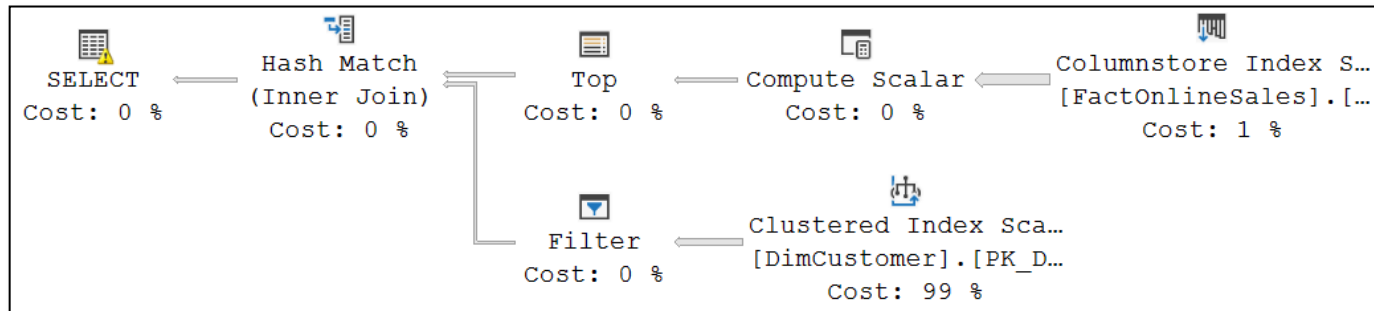
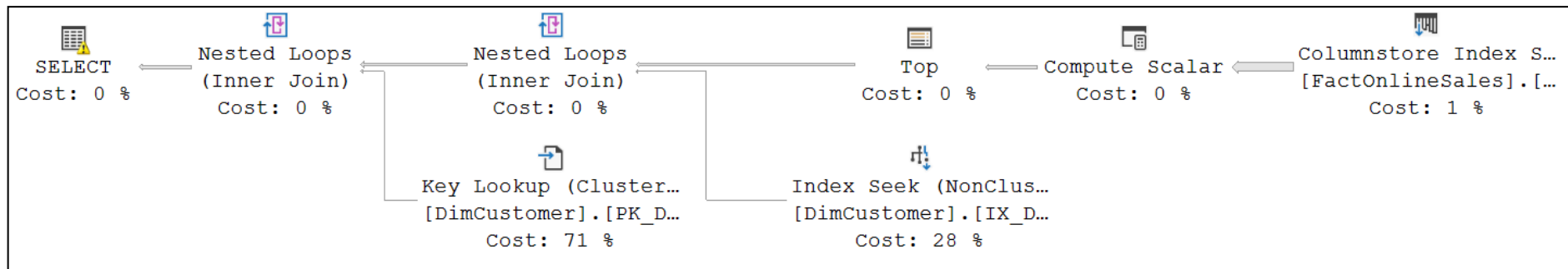


Adaptive Join  
(Inner Join)

# Adaptive Join

Role in the execution plan

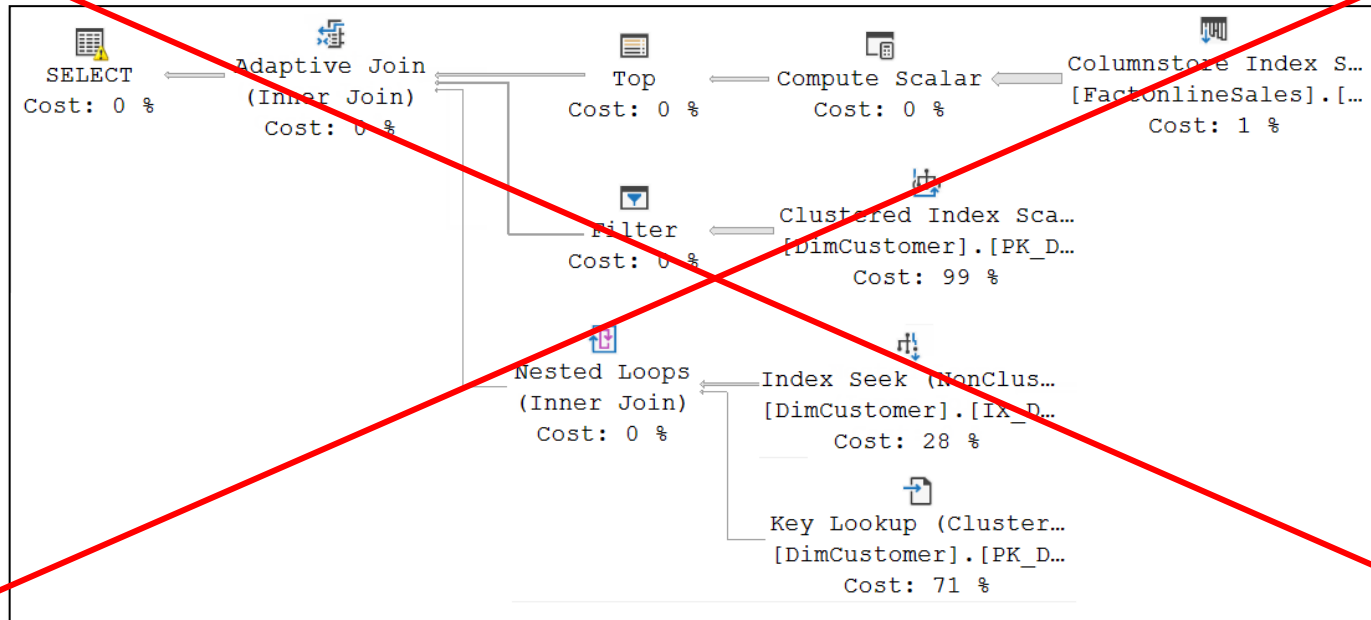
Two strategies



# Adaptive Join

Role in the execution plan

Two strategies



Adaptive Join  
(Inner Join)



# Adaptive Join

## Role in the execution plan

### Two strategies

Hash Match, efficient for “many” rows

(Clustered) Index Scan (+ Filter)

*Hash Match / Merge Join*

*Regular aggregations*

Nested Loops, efficient for “few” rows

(Clustered) Index Seek

*Nested Loops*

*Global aggregation with Stream Aggregate*

*Table Spool / Index Spool*

All potential matches

Single pass

More expensive

Only relevant rows

Based on *Outer References*

Cheap per execution



Adaptive Join  
(Inner Join)

# Summary

## Adaptive Join (advanced)

- Logic for all supported join types

- Nested Loops

  - Optimized Nested Loops

  - Prefetching

- Hash spills

- Use cases

  - (Currently?) limited to simple use cases only

# Summary

## Block 3, basic level

- Logical join types

- Nested Loops

- Merge Join

- Hash Match

- Adaptive Join

- Other combining operators

  - Concatenation, Sequence, Switch

## Block 3, advanced level

- Nested Loops (advanced)

- Merge Join (advanced)

- Hash Match (advanced)

- Adaptive Join (advanced)

# Next chapters

## Block 4: Sorting and grouping – basic level

### Sorting

- Sort

- Top N Sort

- Distinct Sort

### Aggregation

- Stream Aggregate

- Hash Match

### Segmenting

### Window Spool

# Next chapters

Block 4: Sorting and grouping – basic level

Block 4: Sorting and grouping – advanced level

- Local/global aggregation

- Advanced Hash Match aggregation

- Sort algorithms

- Window Spool performance