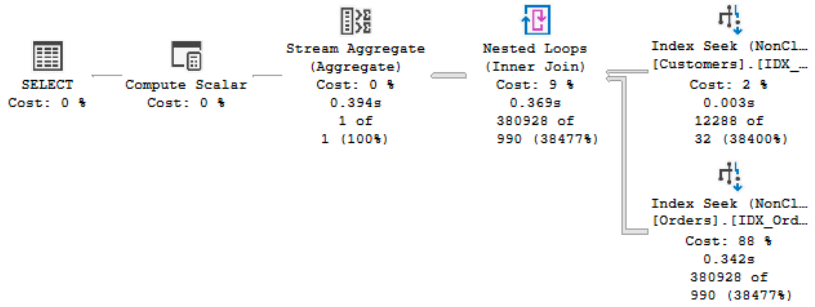
Red and Red in

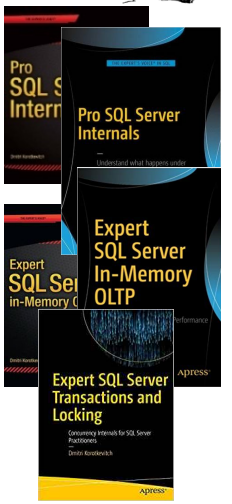


Dmitri Korotkevitch
http://aboutsqlserver.com

Hello!

20+ years of experience in IT
15+ years of experience working with SQL Server
Microsoft Data Platform MVP
Microsoft Certified Master
Author

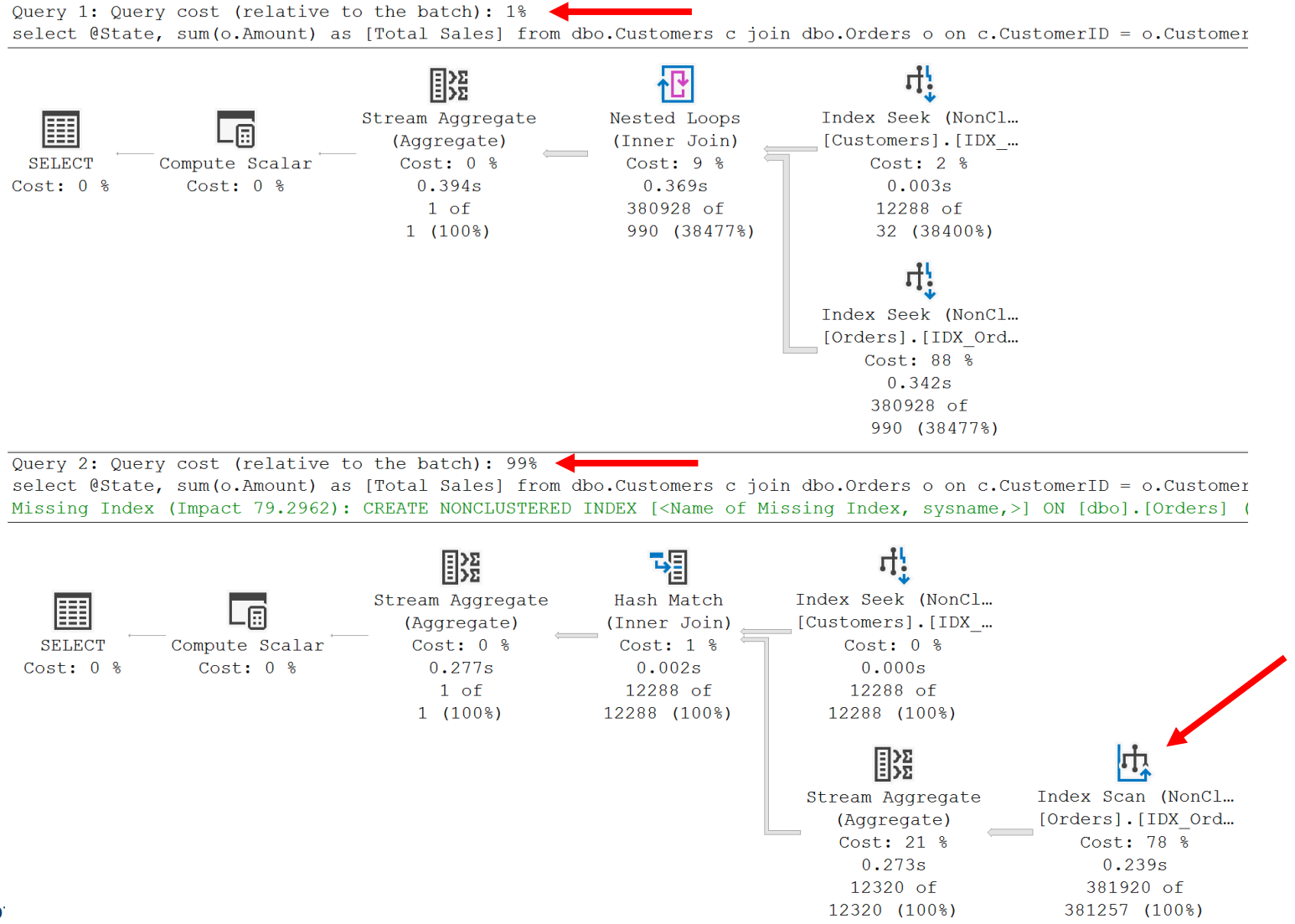
- Pro SQL Server Internals (v1-2)
- Expert SQL Server In-Memory OLTP (v1-2)
- Expert SQL Server Transactions and Locking



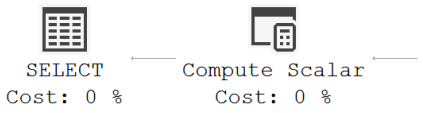
Blog: http://aboutsqlserver.com
Email: dk@aboutsqlserver.com
YouTube: AboutSQLServer

Time for Quiz

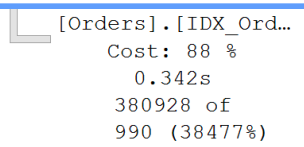
```
select @State, sum(o.Amount) as [Total Sales]
from dbo.Customers c join dbo.Orders o on
    c.CustomerID = o.CustomerID
where
    c.CustomerState = @State and
    o.OrderDate between @StartDate and @EndDate
```


Query 1: Query cost (relative to the batch): 1%
select @State, sum(o.Amount) as [Total Sales] from dbo.Customers c join dbo.Orders o on c.CustomerID = o.Customer
SELECT
Cost: 0 %
Compute Scalar
Cost: 0 %
Stream Aggregate
(Aggregate)
Cost: 0 %
0.394s
1 of
1 (100%)
Nested Loops
(Inner Join)
Cost: 9 %
0.369s
380928 of
990 (38477%)
Index Seek (NonCl…
[Customers].[IDX_…
Cost: 2 %
0.003s
12288 of
32 (38400%)
Index Seek (NonCl…
[Orders].[IDX_Ord…
Cost: 88 %
0.342s
380928 of
990 (38477%)
Query 2: Query cost (relative to the batch): 99%
select @State, sum(o.Amount) as [Total Sales] from dbo.Customers c join dbo.Orders o on c.CustomerID = o.Customer
Missing Index (Impact 79.2962): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Orders] (
SELECT
Cost: 0 %
Compute Scalar
Cost: 0 %
Stream Aggregate
(Aggregate)
Cost: 0 %
0.277s
1 of
1 (100%)
Hash Match
(Inner Join)
Cost: 1 %
0.002s
12288 of
12288 (100%)
Index Seek (NonCl…
[Customers].[IDX_…
Cost: 0 %
0.000s
12288 of
12288 (100%)
Stream Aggregate
(Aggregate)
Cost: 21 %
0.273s
12320 of
12320 (100%)
Index Scan (NonCl…
[Orders].[IDX_Ord…
Cost: 78 %
0.239s
381920 of
381257 (100%)

```
Query 1: Query cost (relative to the batch): 1%
select @State, sum(o.Amount) as [Total Sales] from dbo.Customers c join dbo.Orders o on c.CustomerID = o.Customer
```

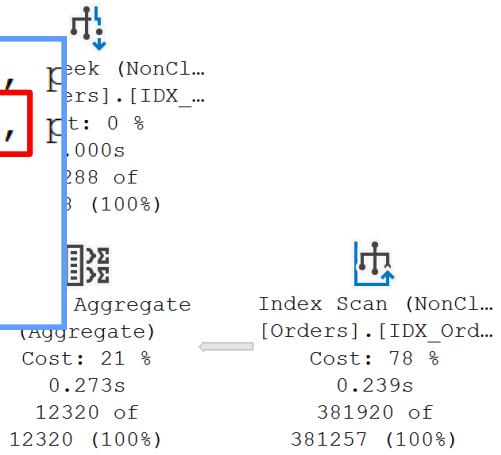



```
Table 'Orders'. Scan count 12288, logical reads 66038,
Table 'Customers'. Scan count 1, logical reads 48, phys

 SQL Server Execution Times:
   CPU time = 391 ms,  elapsed time = 395 ms.
```




```
Query 2: Query cost (relative to the batch): 99%
select @State, sum(o.Amount) as [Total Sales] from dbo.Customers c join dbo.Orders o on c.CustomerID = o.Customer
Missing Index (Impact 79.2962): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Orders] (
```

```
Table 'Customers'. Scan count 1, logical reads 48, p
Table 'Orders'. Scan count 1, logical reads 14532, p

 SQL Server Execution Times:
   CPU time = 196 ms,  elapsed time = 209 ms.
```



Agenda

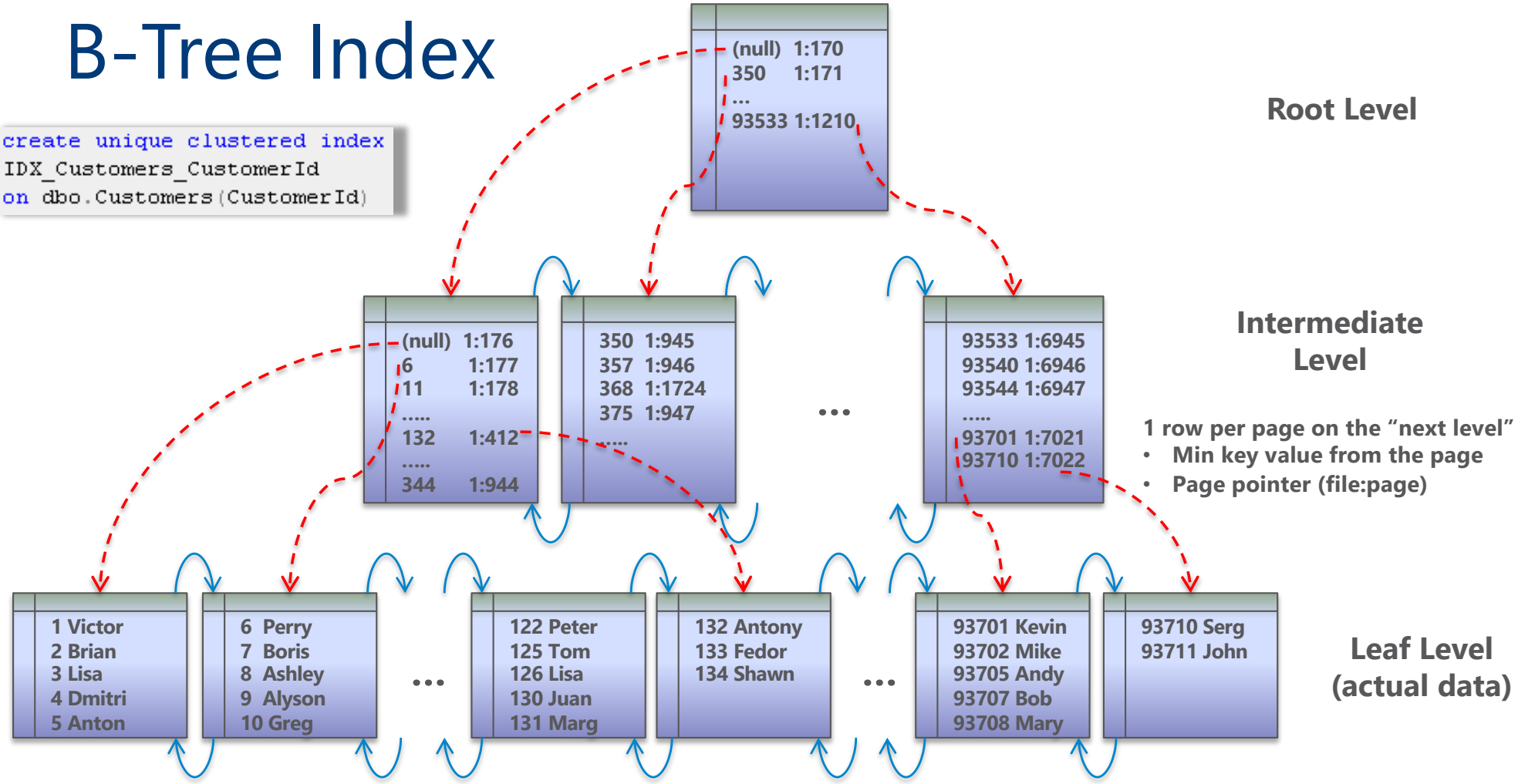
Seeks and Scans

Joins

Key Lookups

B-Tree Index

```
create unique clustered index
IDX_Customers_CustomerId
on dbo.Customers(CustomerId)
```



Root Level

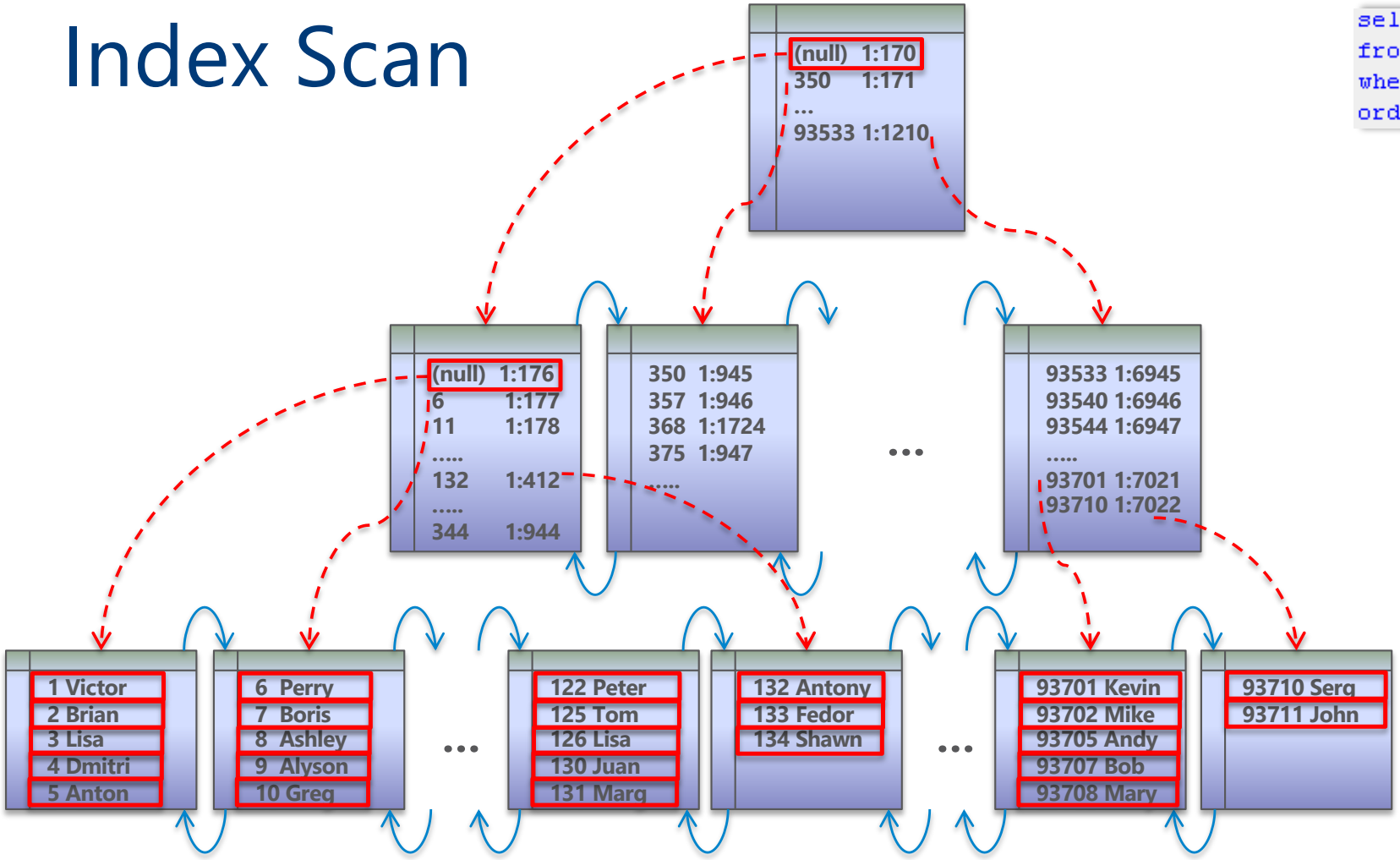
Intermediate Level

1 row per page on the "next level"
- **Min key value from the page**
- **Page pointer (file:page)**

Leaf Level (actual data)

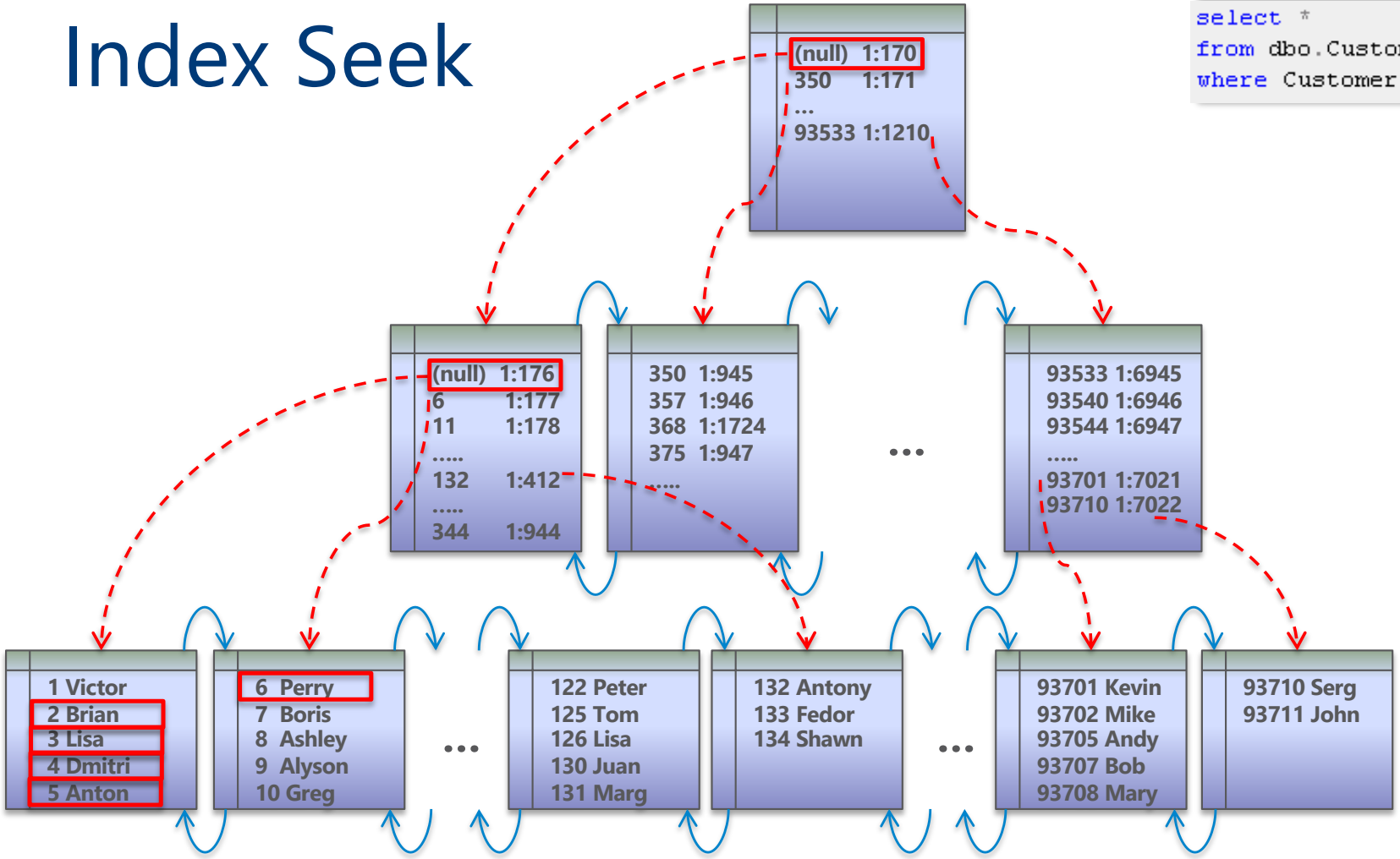
Index Scan

```
select *
from dbo.Customers
where Name = N'Boris'
order by CustomerId
```



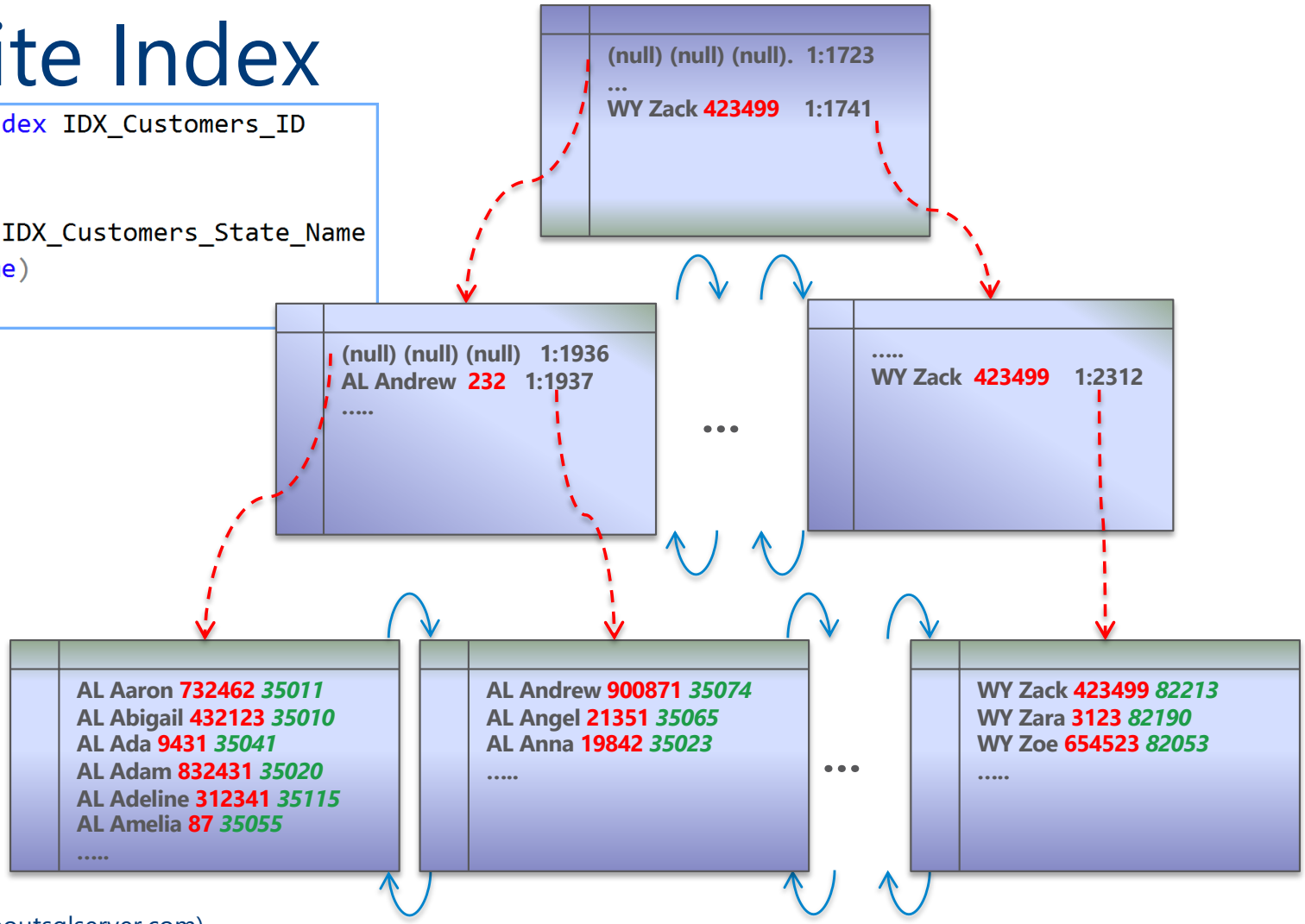
Index Seek

```
select *
from dbo.Customers
where CustomerId between 2 and 6
```



Composite Index

```
create unique clustered index IDX_Customers_ID
on dbo.Customers(ID);

create nonclustered index IDX_Customers_State_Name
on dbo.Customers(State,Name)
include(ZipCode);
```



Index Seek

Demo

Index Seek

Cardinality estimation error

Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Number of Rows Read	4497408
Actual Number of Rows	380928
Estimated Number of Rows	31
Estimated Number of Rows to be Read	366

Predicate
[SQLServerInternals].[dbo].[Orders].[OrderDate] as [o].[OrderDate]>=[@StartDate] AND [SQLServerInternals].[dbo].[Orders].[OrderDate] as [o].[OrderDate]<=[@EndDate]

Object
[SQLServerInternals].[dbo].[Orders].[IDX_Orders_CustomerId] [o]

Output List
[SQLServerInternals].[dbo].[Orders].Amount

Seek Predicates
Seek Keys[1]: Prefix: [SQLServerInternals].[dbo].[Orders].CustomerID = Scalar Operator ([SQLServerInternals].[dbo].[Customers].[CustomerID] as [c].[CustomerID])

# of rows read / processed

Large # of reads or large delta – is index selective enough?

# of rows selected

Additional filter on all selected rows

Can those column(s) be added as the index key(s)?

Filtering the range of the keys

Is it selective enough?

Joins

Three logical joins

- (Nested) Loop
- Hash
- Merge

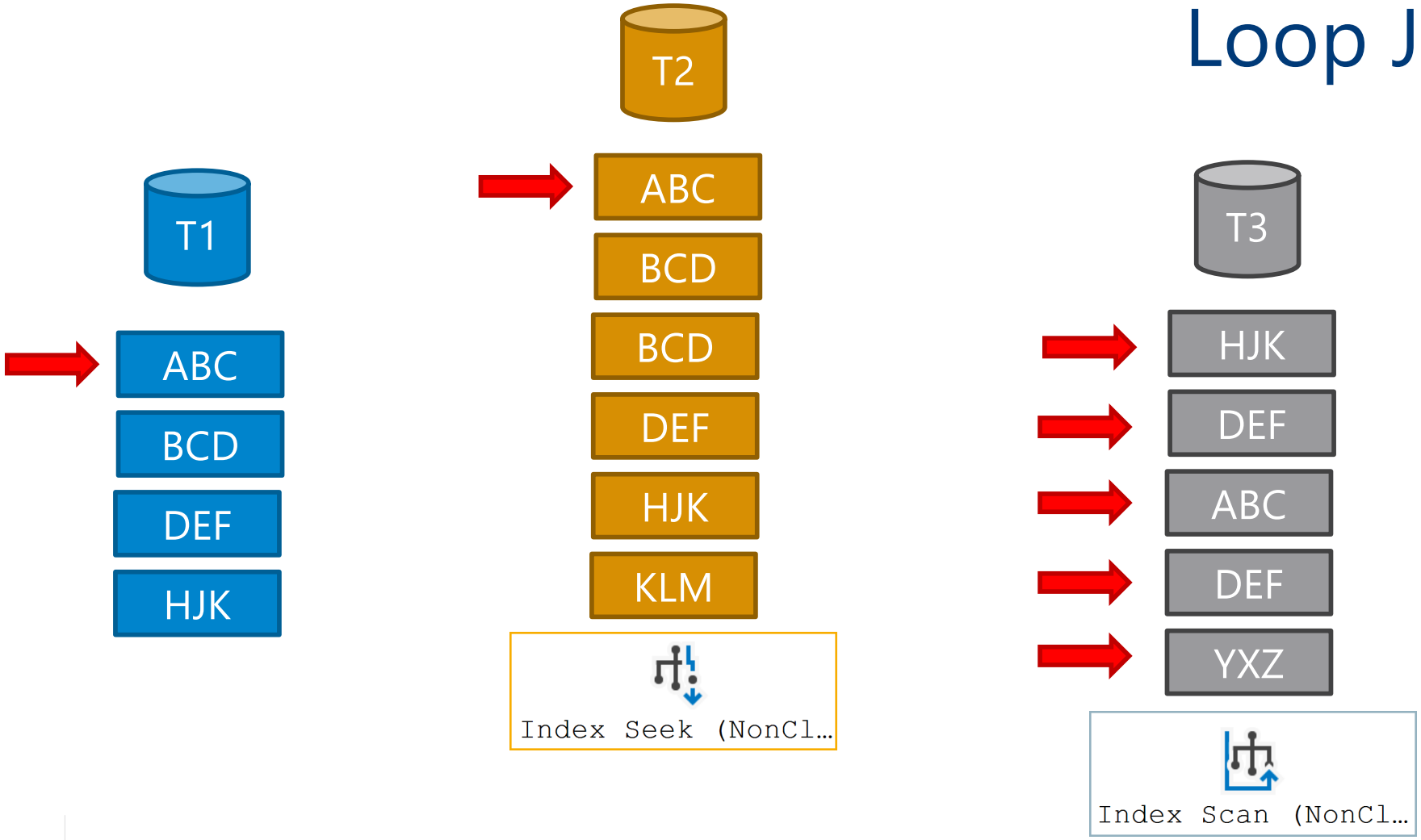
Multiple physical joins

Joins

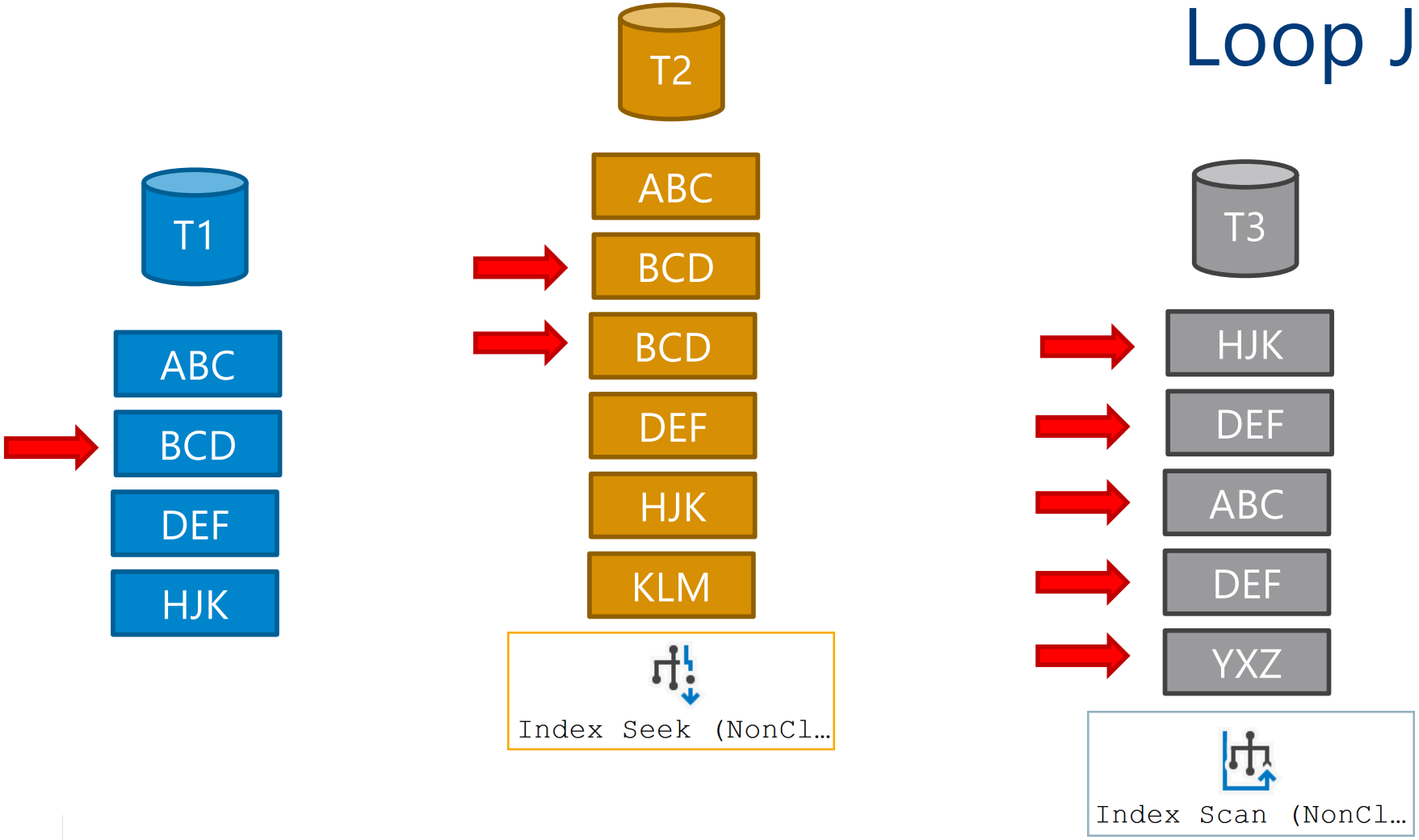
```
create table T1
(
    Col varchar,
    index IDX_T1(Col)
)

create table T2
(
    Col varchar,
    index IDX_T2(Col)
)

create table T3
(
    Col varchar
)
```

```
select *
from
    T1 join T2 on
        T1.Col = T2.Col
    join T3 on
        T1.Col = T3.Col
```

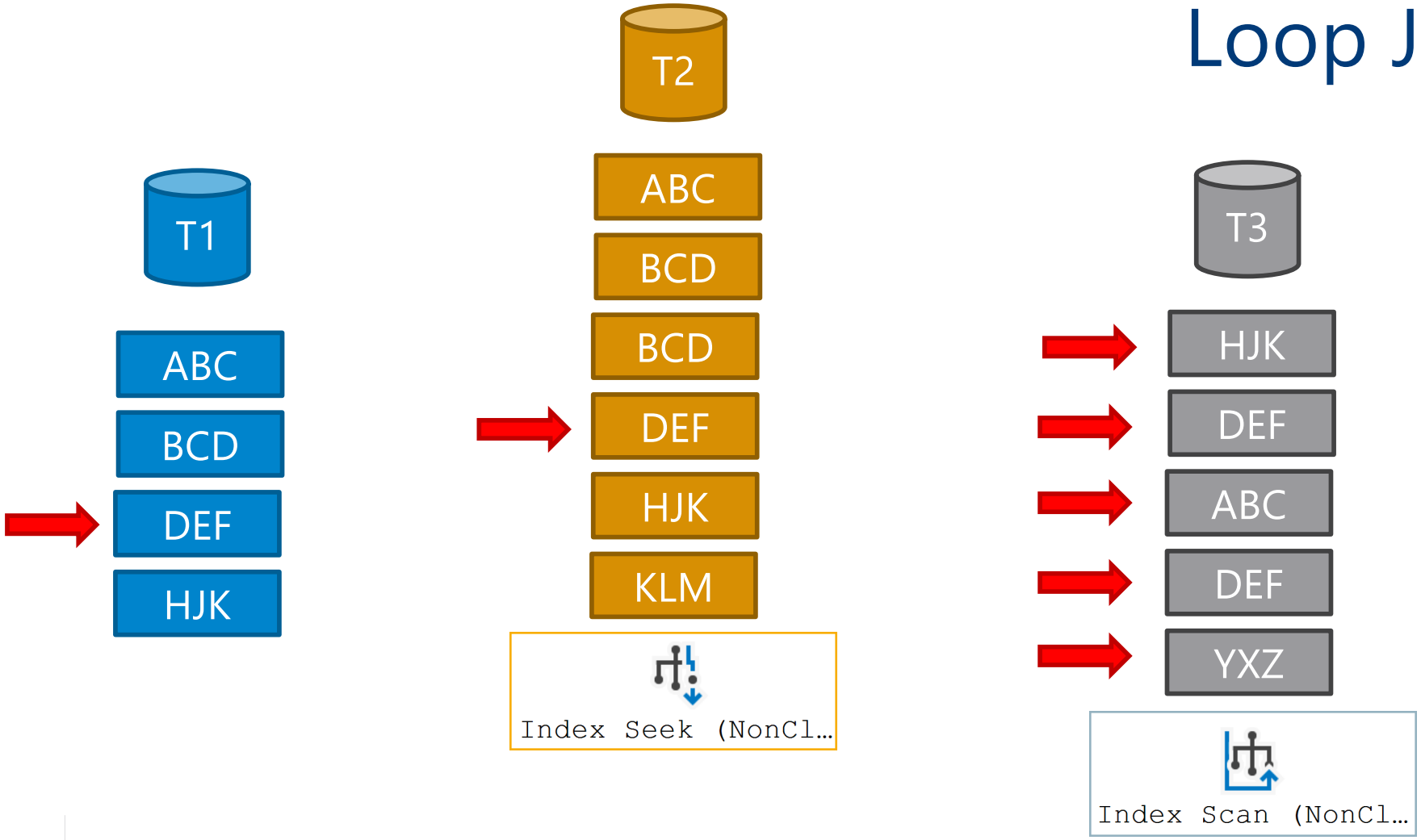
Loop Join


T1
ABC
BCD
DEF
HJK
T2
ABC
BCD
BCD
DEF
HJK
KLM
Index Seek (NonCl…
T3
HJK
DEF
ABC
DEF
YXZ
Index Scan (NonCl…

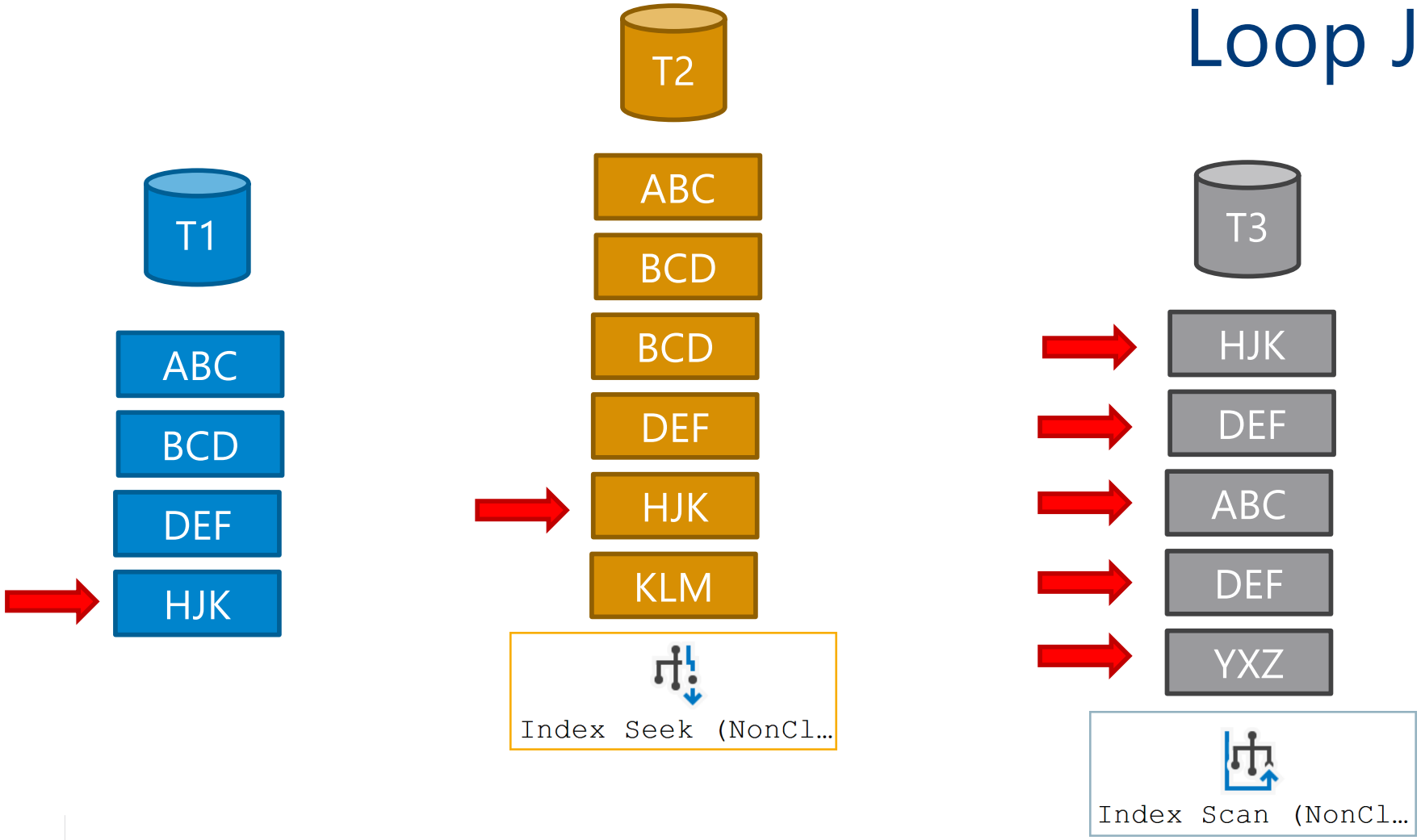
Loop Join


T1
ABC
BCD
DEF
HJK
T2
ABC
BCD
BCD
DEF
HJK
KLM
Index Seek (NonCl…
T3
HJK
DEF
ABC
DEF
YXZ
Index Scan (NonCl…

Loop Join


T1
ABC
BCD
DEF
HJK
T2
ABC
BCD
BCD
DEF
HJK
KLM
Index Seek (NonCl…
T3
HJK
DEF
ABC
DEF
YXZ
Index Scan (NonCl…

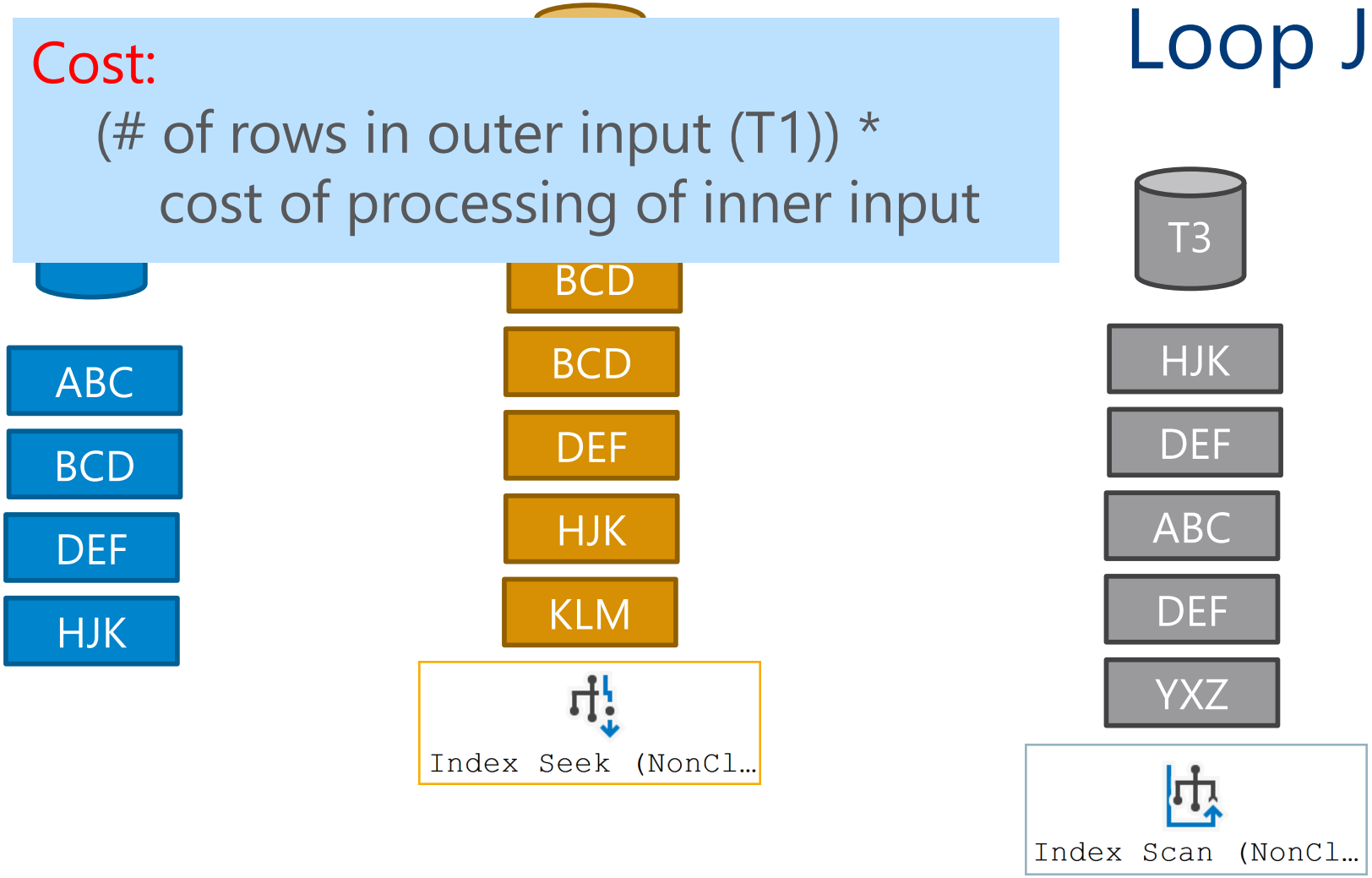
Loop Join


T1
ABC
BCD
DEF
HJK
T2
ABC
BCD
BCD
DEF
HJK
KLM
Index Seek (NonCl…
T3
HJK
DEF
ABC
DEF
YXZ
Index Scan (NonCl…

Loop Join

Cost:

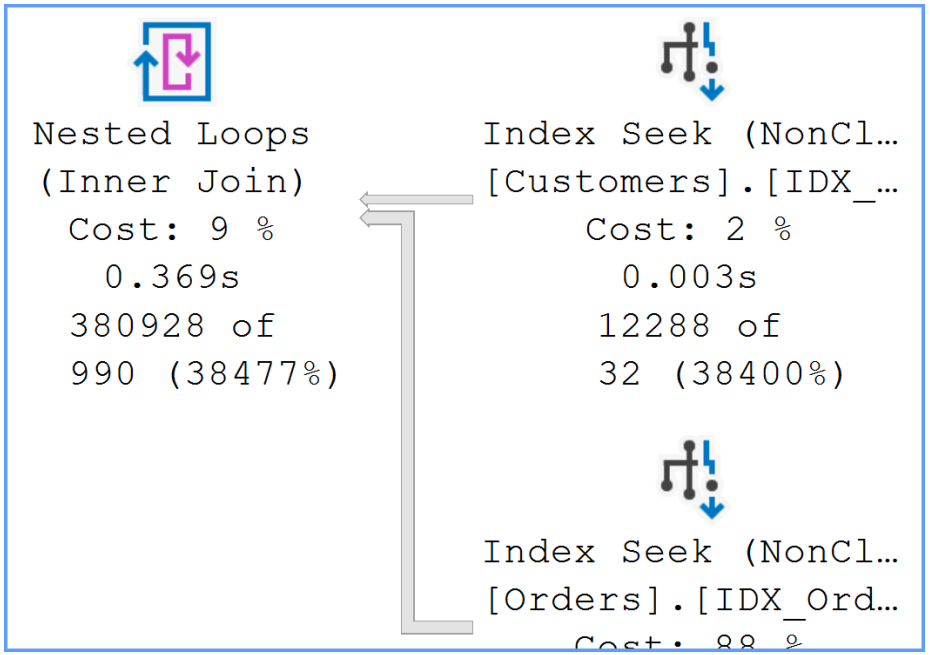
(# of rows in outer input (T1)) *
cost of processing of inner input



Loop Join

Cost:

(# of rows in outer input (T1)) *
cost of processing of inner input



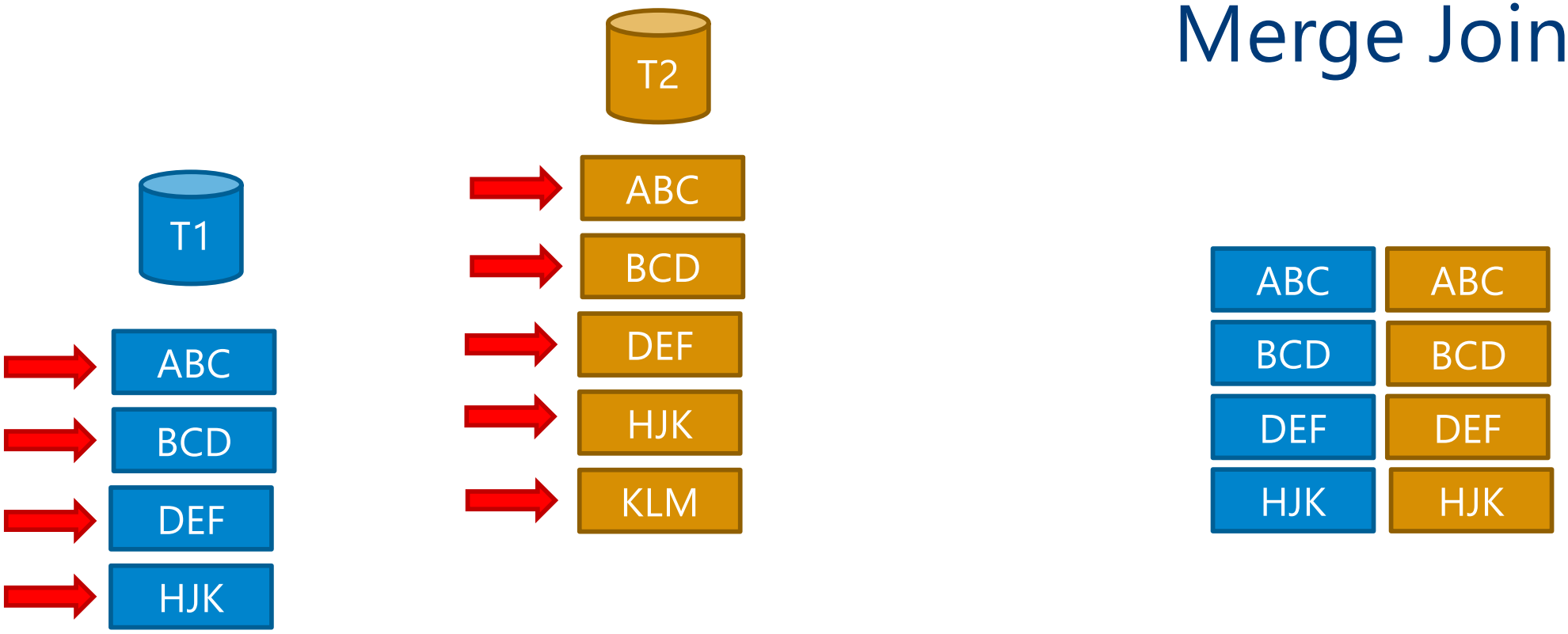
Index Seek (N
Scan a particular range of rows index.

Actual Number of Rows	12288
Estimated Number of Rows	32

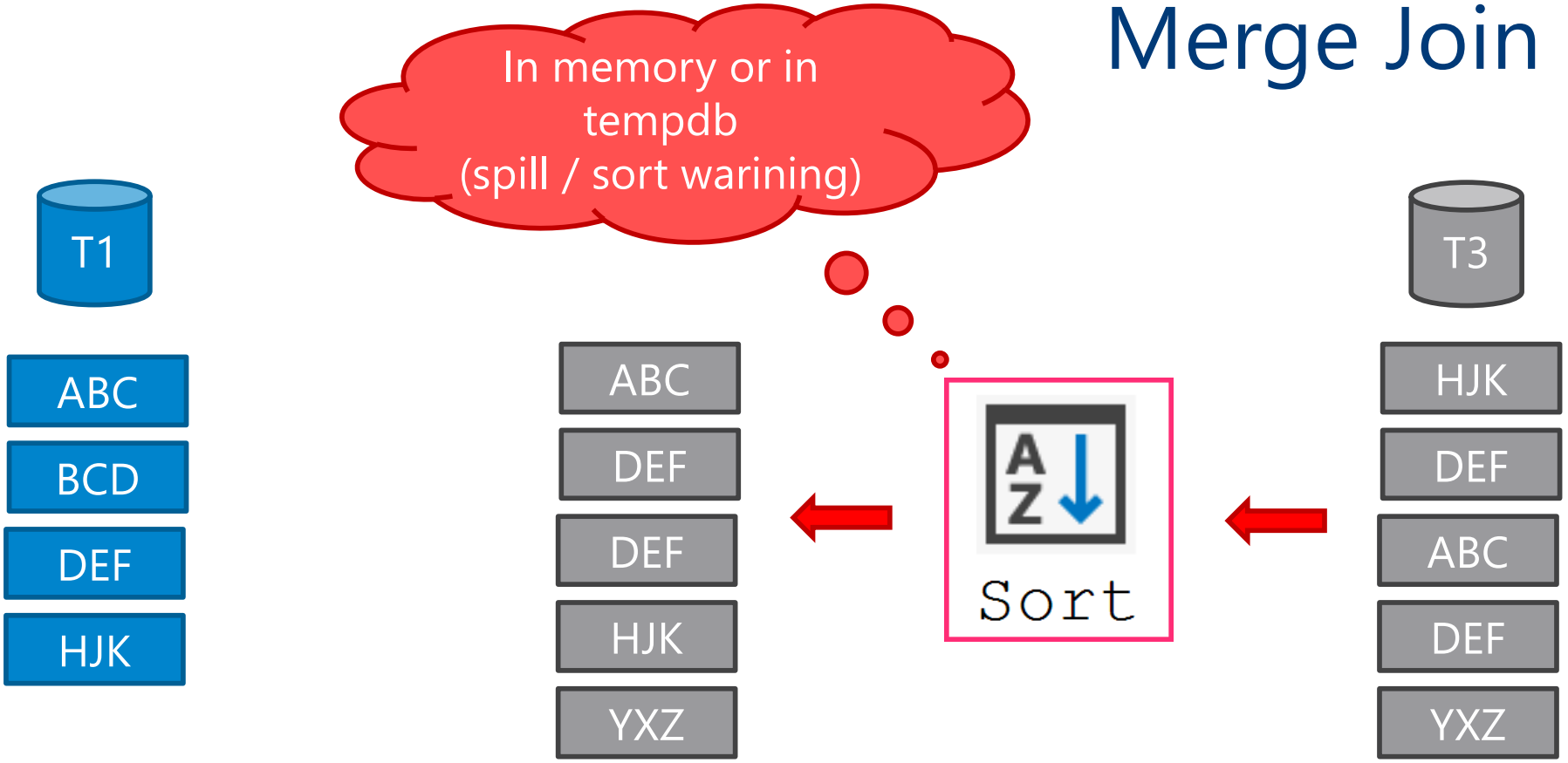
Index Seek (NonClu
Scan a particular range of rows from ndex.

Estimated Number of Executions	32
Number of Executions	12288

Merge Join


T2
ABC
BCD
DEF
HJK
KLM
T1
ABC
BCD
DEF
HJK
ABC
ABC
BCD
BCD
DEF
DEF
HJK
HJK

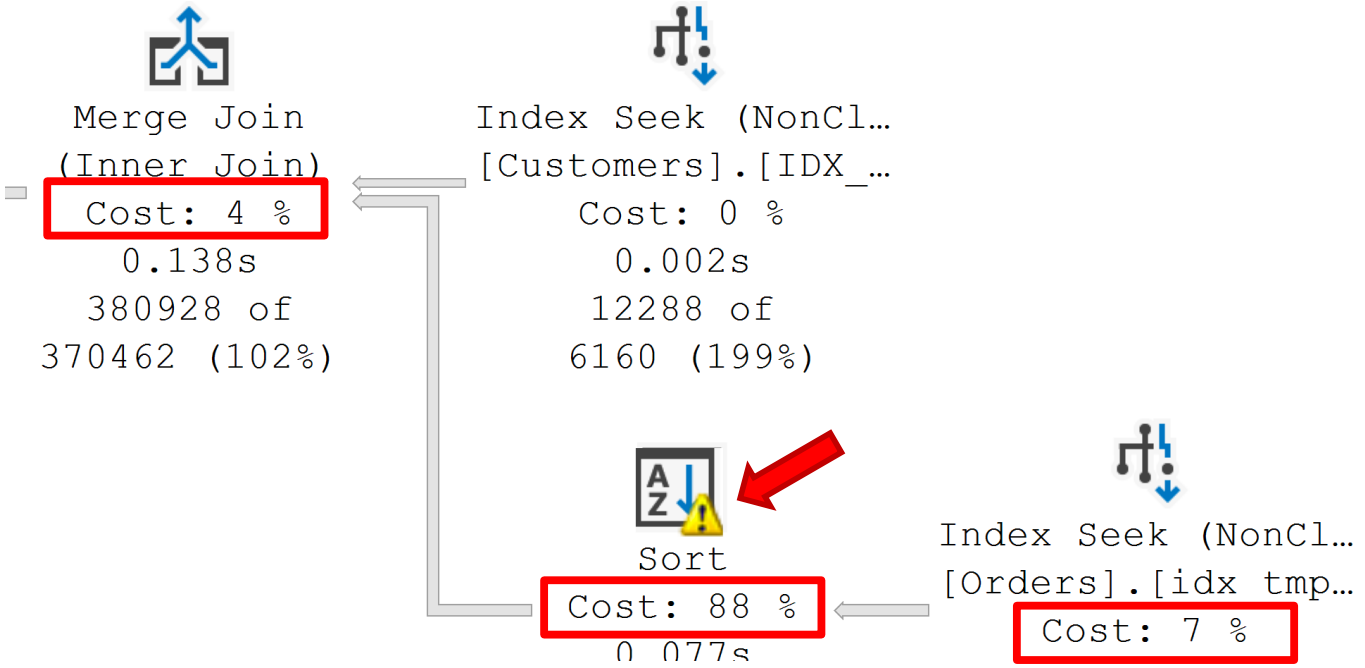
Merge Join


In memory or in tempdb (spill / sort warining)
T1
ABC
BCD
DEF
HJK
ABC
DEF
DEF
HJK
YXZ
Sort
T3
HJK
DEF
ABC
DEF
YXZ

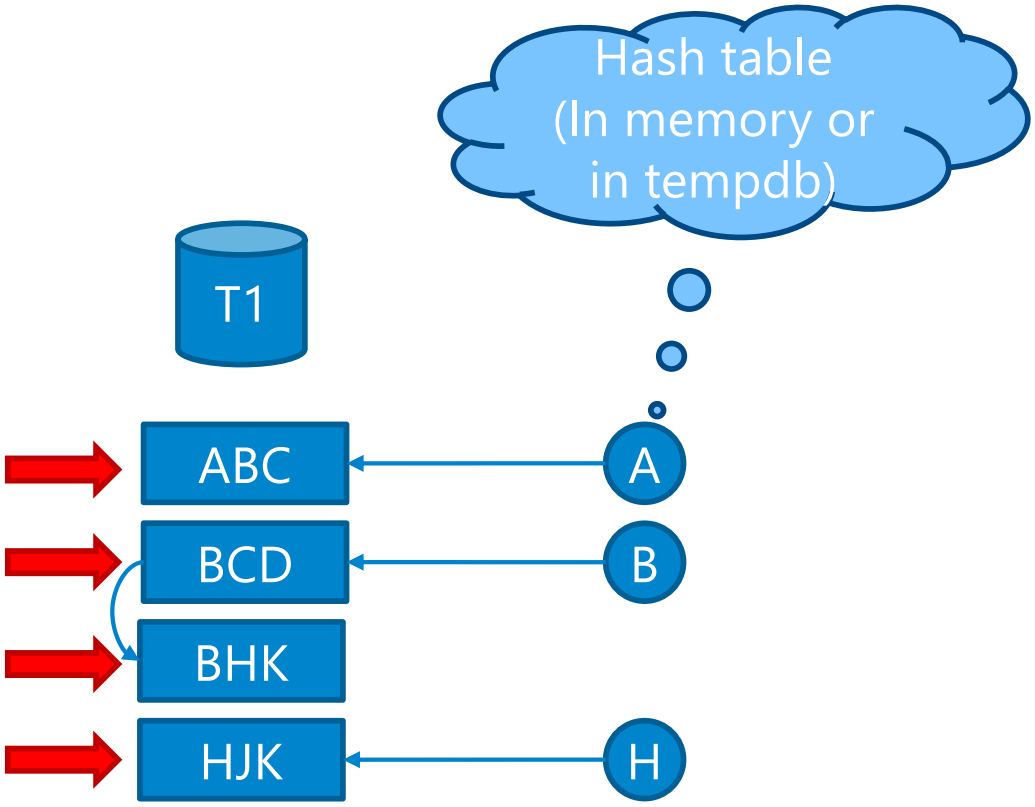
Merge Join

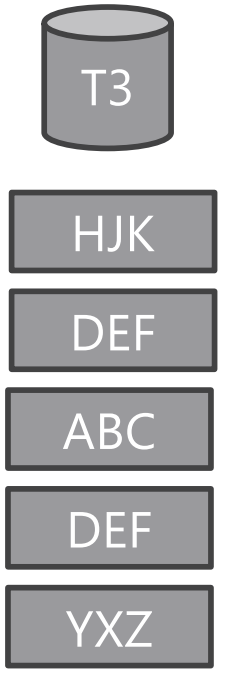
Cost:

cost of processing of input 1 +
cost of processing of input 2 +
cost of sort (if required)

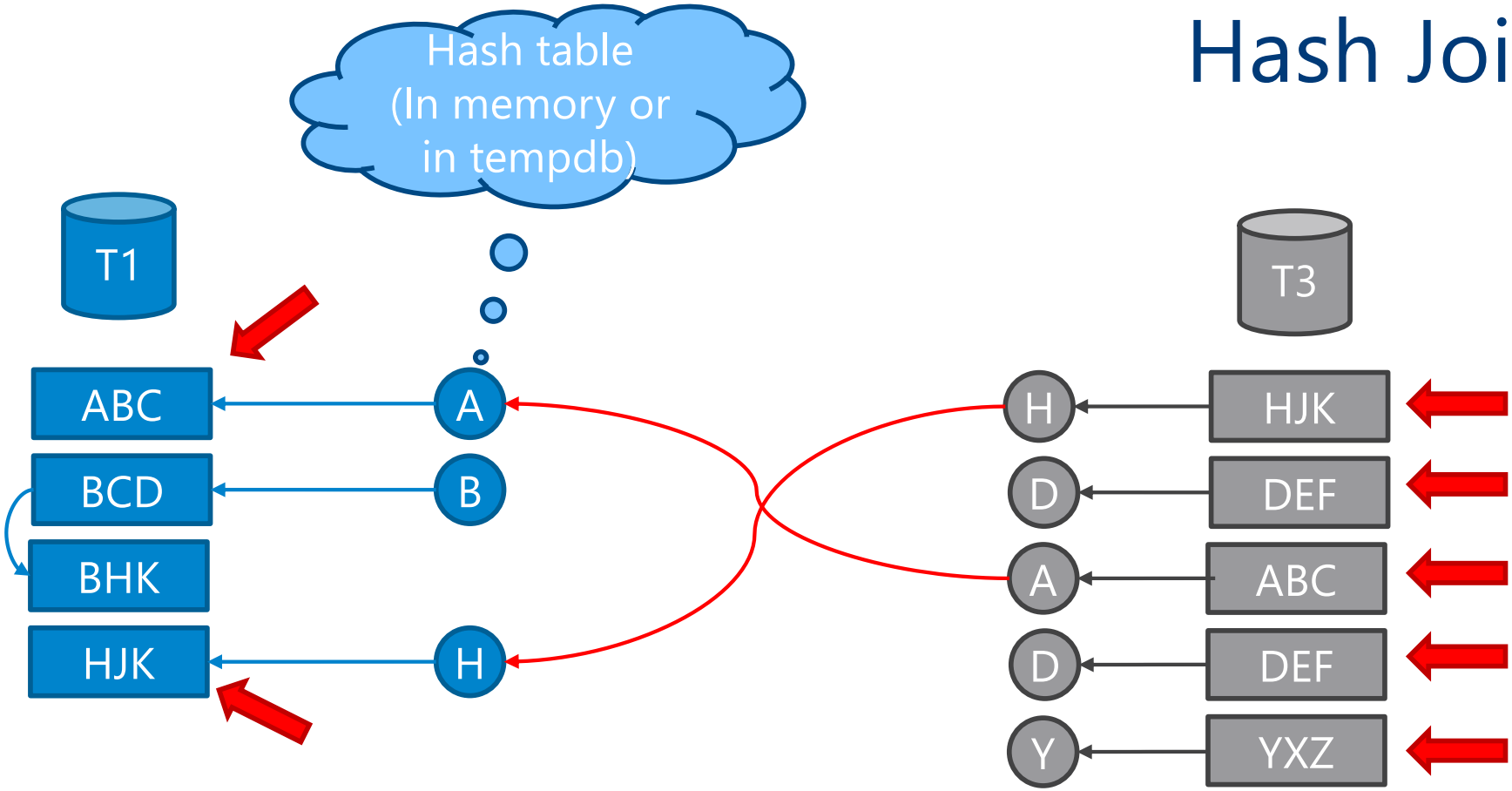


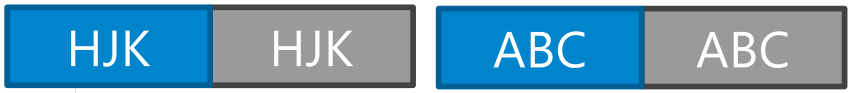
Hash Join


Hash table
(In memory or
in tempdb)
T1
ABC
BCD
BHK
HJK
A
B
H


T3
HJK
DEF
ABC
DEF
YXZ

Hash Join

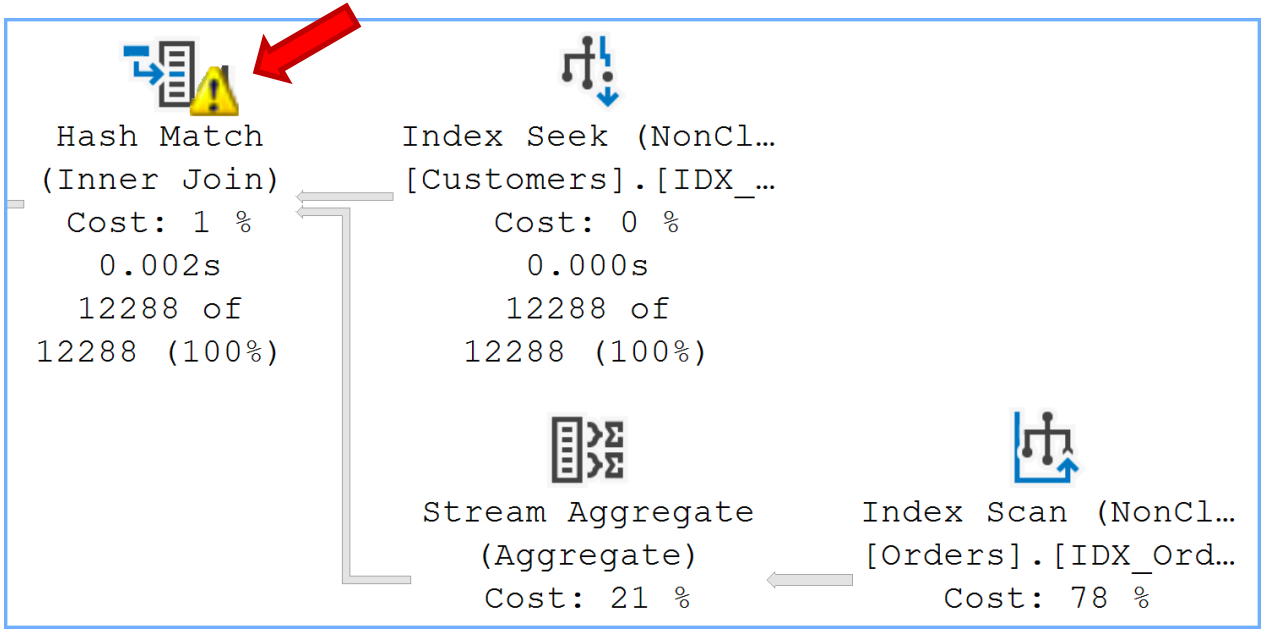

Hash table
(In memory or
in tempdb)
T1
ABC
BCD
BHK
HJK
A
B
H
T3
HJK
DEF
ABC
DEF
YXZ
H
D
A
D
Y


HJK
HJK
ABC
ABC

Hash Join

Cost:

cost of processing of input 1 +
cost of processing of input 2 +
cost of hash operations



Joins

	Loop	Hash	Merge
Use-case	Small outer input Low cost of inner input scan	Large unsorted inputs	Mid-size sorted inputs
Cost	Depends on cost of inner input scan and # of executions	Startup overhead with linear dependency on size of inputs	Low cost but may introduce expensive *Sort*
tempdb	No	May lead to spills	*Sort* may lead to spills
Blocking	No	Yes (hash table creation)	Yes if *Sort* is required

Joins

Demo

Cardinality Estimation Errors (Estimated vs. Actual)

Outdated statistics

- Redesign statistics maintenance strategy (TF2371, DB Compat Level, etc)

Parameter-sensitive plans / Parameter Sniffing

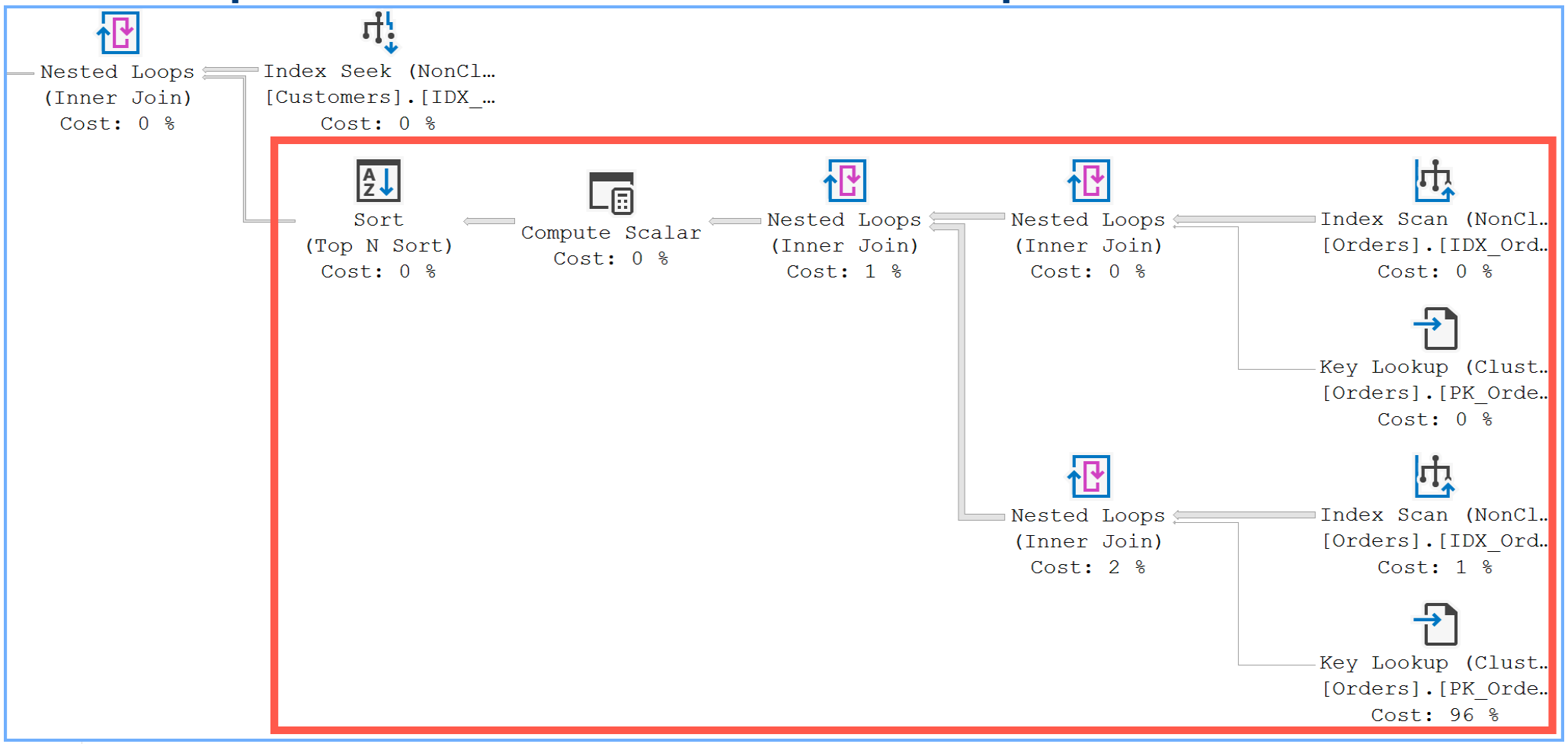
Model issues

- Simplify / split queries
- Use temporary tables
- Remove multi-statement UDFs
- Change cardinality estimation model
- … find the root cause

Parameter Sniffing

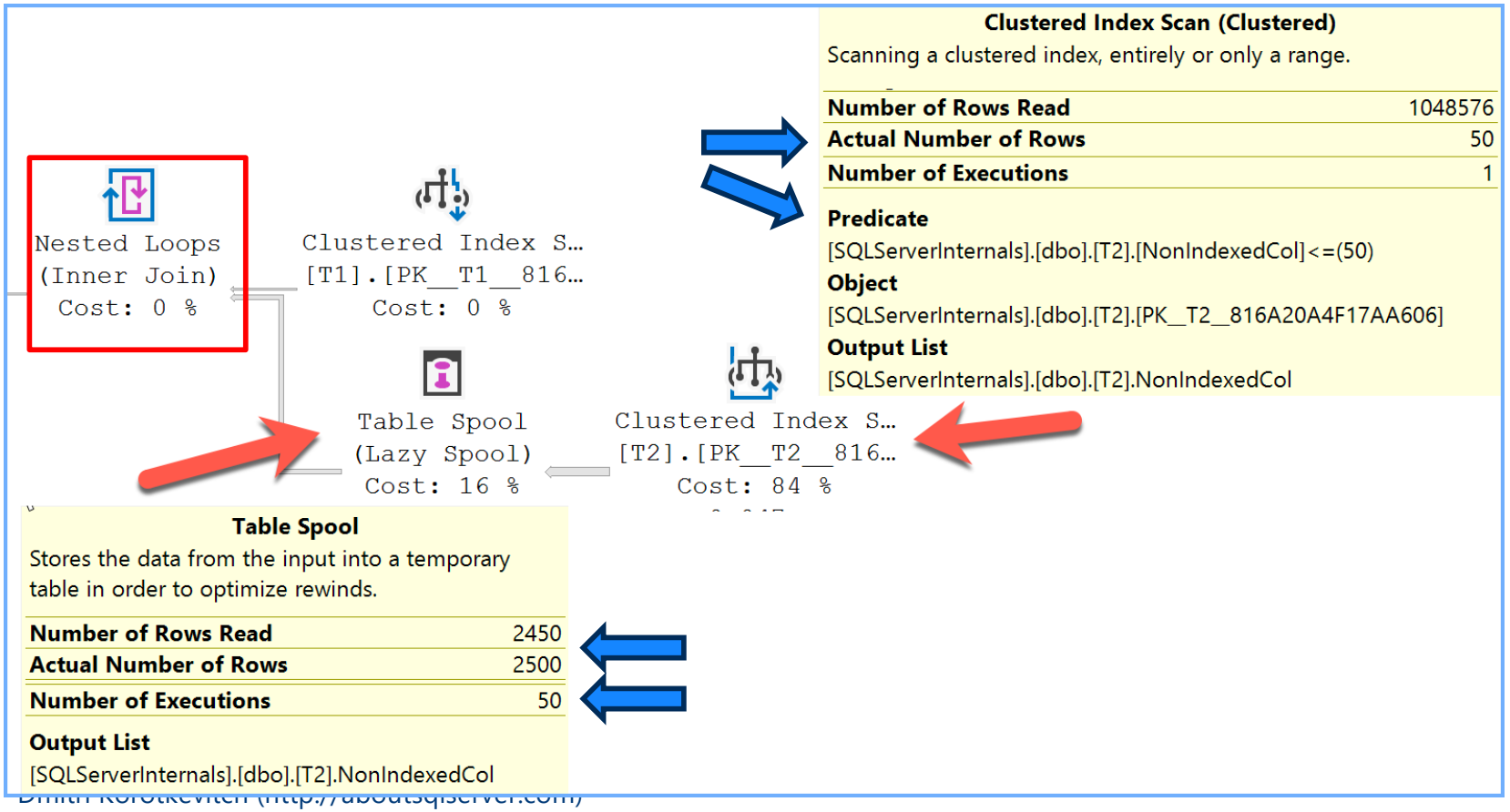
Demo

Loop Join: Cost of Inner Input

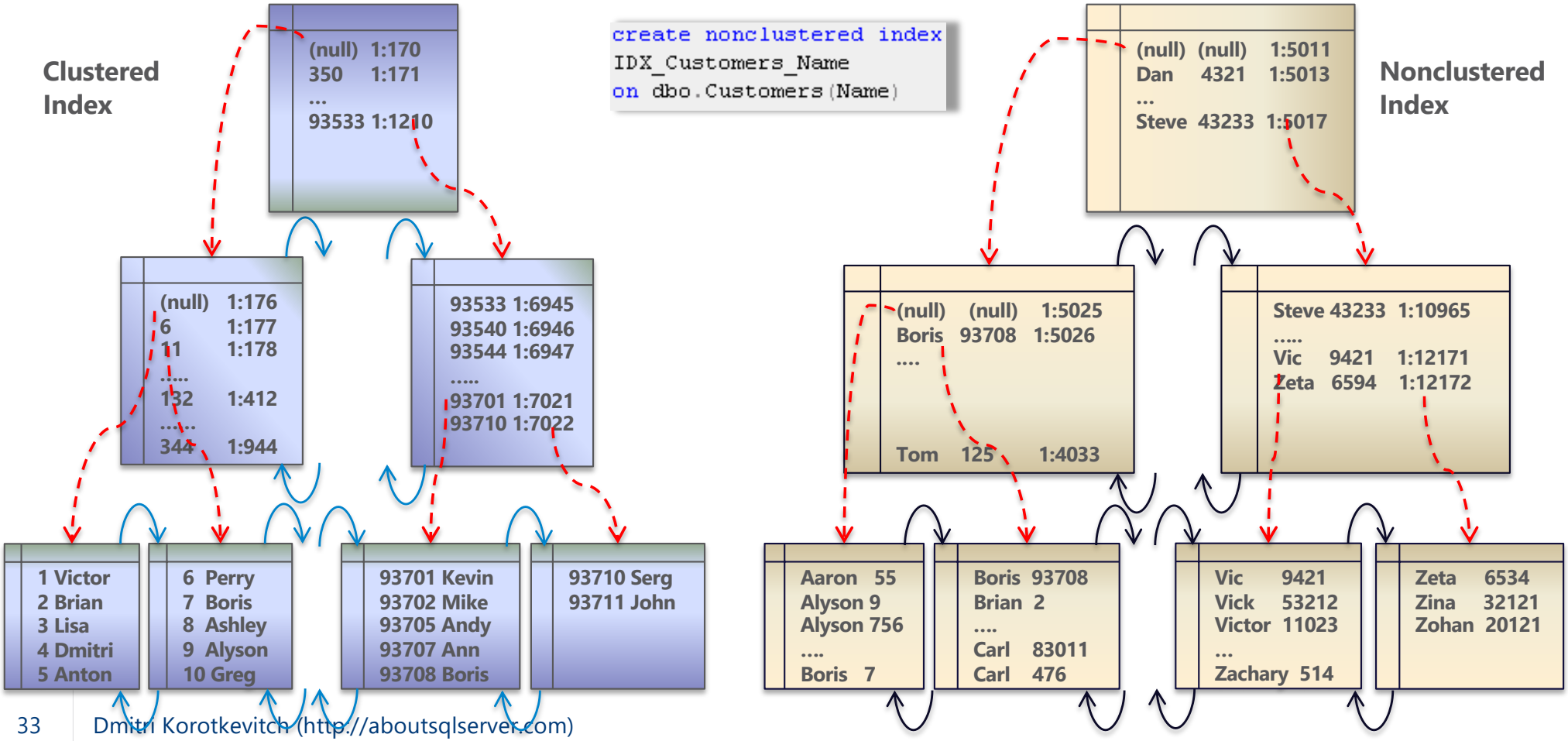


Spools

```
select count(*)
from dbo.T1 inner loop join dbo.T2 on
    T1.IndexedCol = T2.NonIndexedCol
where
    T1.IndexedCol <= 50
```

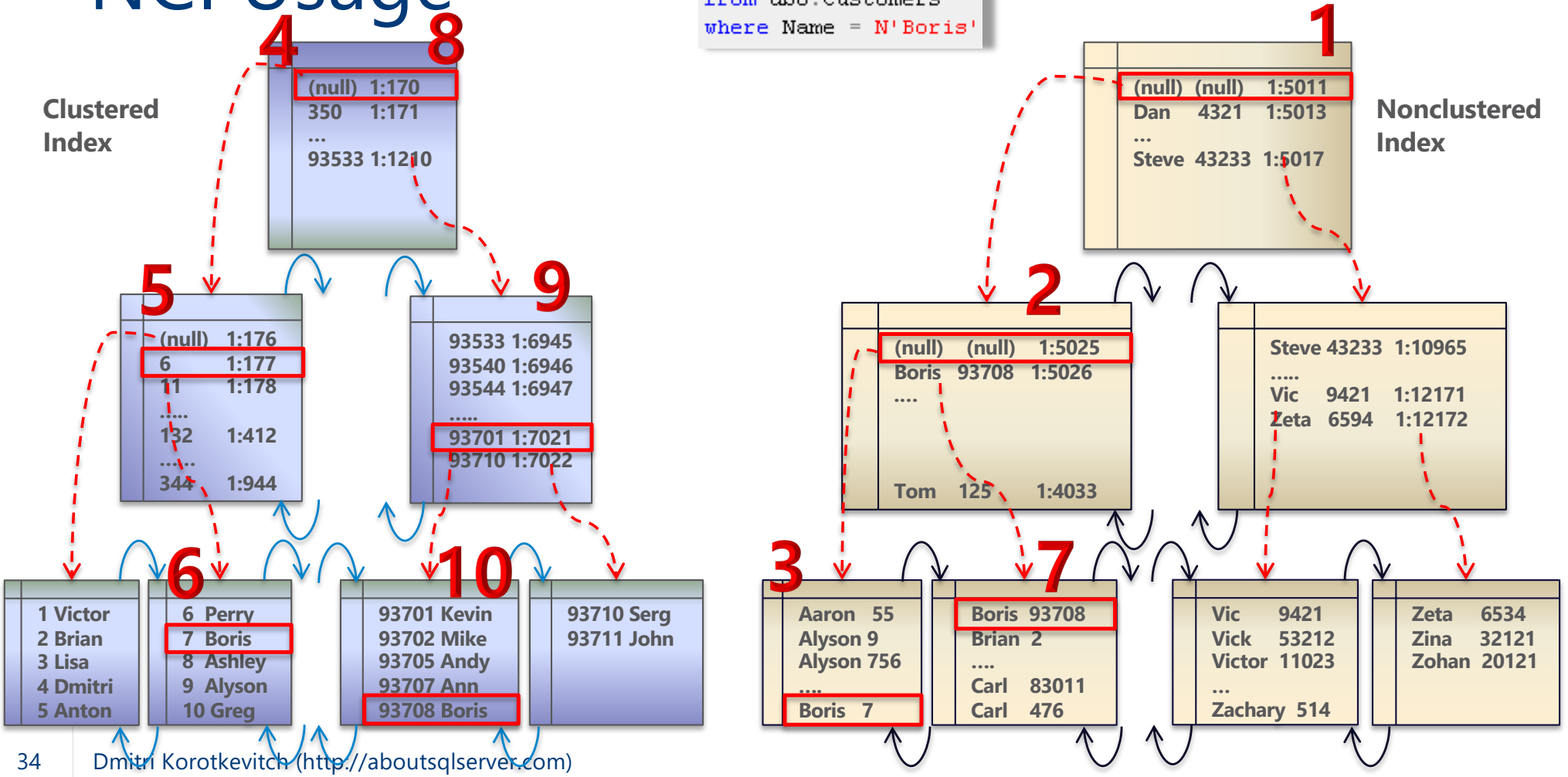


Nonclustered Index



NCI Usage

```
select *
from dbo.Customers
where Name = N'Boris'
```



Key Lookup

Demo

Key Lookups

Key Lookups are Nested Loop Joins

- Fine and very efficient with small # of executions
- Inefficient and not used with large # of executions
- Beware of Cardinality Estimation Errors!

Can be avoided with included columns

- Remember about the overhead!

A Couple Tricks

Demo

Estimated vs. Actual Execution Plans

AKA -> Plans with and without run-time execution statistics

Plans without run-time execution statistics

- Will be affected by cardinality estimation errors
- Learn the data!

Estimated vs. Actual Execution Plans

AKA -> Plans with and without run-time execution statistics

Plans without run-time execution statistics

- Will be affected by cardinality estimation errors
- Learn the data!

Key Points

Analyze efficiency and selectivity of the Index Seek

- Seek Predicate, Predicate, Number of rows read vs. Actual number of rows, etc

Pay attention to join type choice

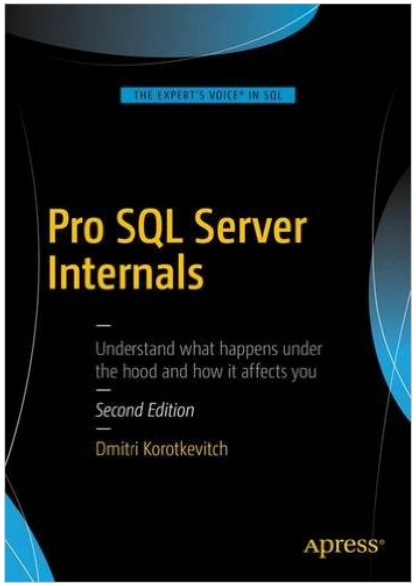
- Estimated vs. actual # of executions; cardinality estimation errors
- Cost of inner input in loop join

Analyze and address excessive Key Lookups

- Small # of Key Lookups is perfectly normal
- Remember the overhead of included columns

Beware of cardinality estimation errors!

Q&A



Blog: http://aboutsqlserver.com
E-mail: dk@aboutsqlserver.com
Slides and demos: http://aboutsqlserver.com/presentations

Thank You