Parameter Sniffing Problem with Stored Procedures

Milos Radivojevic

About Me

- DI Milos Radivojevic, Vienna, Austria
- Data Platform Architect

- Database Developer
- MCTS SQL Server Development

SQL Server 2005
SQL Server 2008, Database Development

- Contact:
- MRadivojevic@SolidQ.com
- www.solidq.com

About SolidQ

- Solid Quality Mentors
  - trusted global provider of advanced consulting, mentoring and education solutions for the Microsoft Data, BI, Collaboration and Development platforms
- SolidQ Journal

The SolidQ Journal, March 2011

Authors: Herbert Albert, Gianluca Hotz, Salvador Ramos, Itzik Ben-Gan, Pablo A. Ahumada, Greg Low, Fernando G. Guerrero
Section: Full Magazine

- http://www.solidq.com/sqj/Pages/Home.aspx

Agenda

- Symptoms of Parameter Sniffing
- What is Parameter Sniffing?
- When and Why It is a Problem?
- Case Study: Stored procedure with optional parameters
- Conclusion

Symptoms of Parameter Sniffing

- What?
  - Execution of a stored procedure within an application is suddenly slow
  - A lot of I/O reads
- When and How?
  - For some parameter combinations it performs well
  - The same procedure on some other installation or in the SSMS works perfect!
  - Suddenly

What is Parameter Sniffing?

▶ Execution of a stored procedure

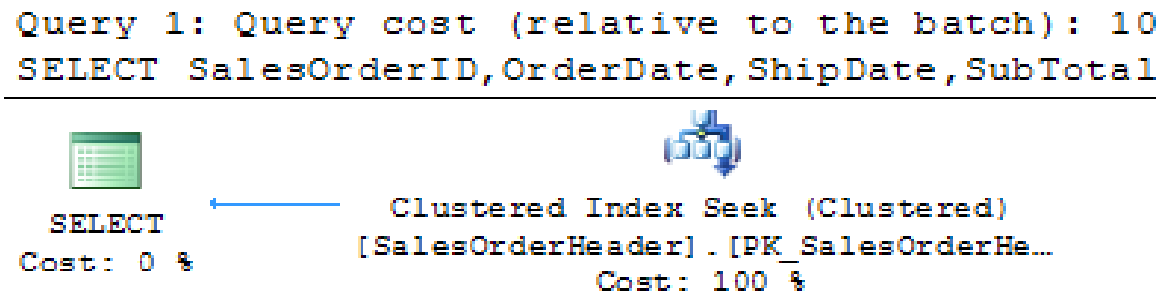
- The optimizer tries to reuse an exec. plan
- Optimizer considerations:
  - Table size
  - Data distribution
  - Selectivity of columns in the JOIN and WHERE clauses
  - Value of submitted parameters for the first invocation
- Submitted parameter values often decides about cardinality

What is Parameter Sniffing?

▶ Creating Sample Stored Procedure

```
CREATE PROCEDURE dbo.getSalesOrderHeader
@SalesOrderID int
AS
        SELECT SalesOrderID,OrderDate,ShipDate,SubTotal
        FROM Sales.SalesOrderHeader
        WHERE SalesOrderID = @SalesOrderID
```

- Execution plan: Clustered Index Scan



What is Parameter Sniffing?

▶ Sniffing

```
EXEC dbo.getSalesOrderHeader 45671
```

```
<ParameterList>
  <ColumnReference Column="@SalesOrderID" ParameterCompiledValue="(45671)" ParameterRuntimeValue="(45671)" />
</ParameterList>
```

```
EXEC dbo.getSalesOrderHeader 56781
```

```
<ParameterList>
  <ColumnReference Column="@SalesOrderID" ParameterCompiledValue="(45671)" ParameterRuntimeValue="(56781)" />
</ParameterList>
```

- PS is not a problem here
- Only one execution plan is possible, actually two
- Submitted parameter value is not important for the way how to generate a result set

What is Parameter Sniffing?

▶ Creating Sample Stored Procedure

```
CREATE PROCEDURE dbo.getSalesOrderHeader
@OrderDate datetime
AS
        SELECT SalesOrderID,OrderDate,ShipDate,SubTotal
        FROM Sales.SalesOrderHeader
        WHERE OrderDate >= @OrderDate
```

- Two reasonable execution plans: for parameter values causing high selectivity and for lowly selective parameter values

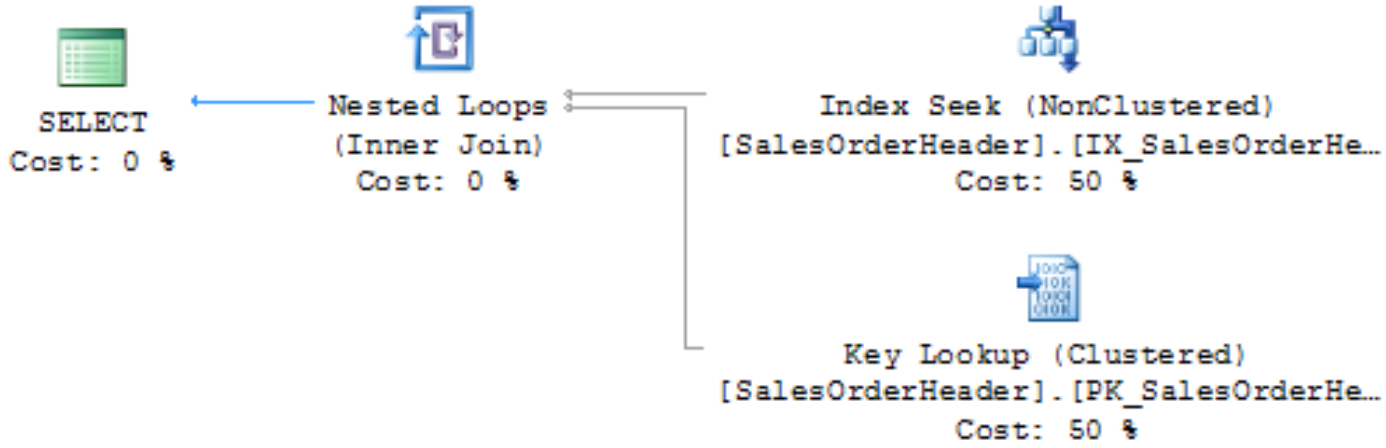
What is Parameter Sniffing?

▶ Execution of a stored procedure

- First call is with a high selective parameter

```
EXEC dbo.getSalesOrderHeader '20050731'
```

```
Table 'SalesOrderHeader'. Scan count 1, logical reads 5,
physical reads 0…
```



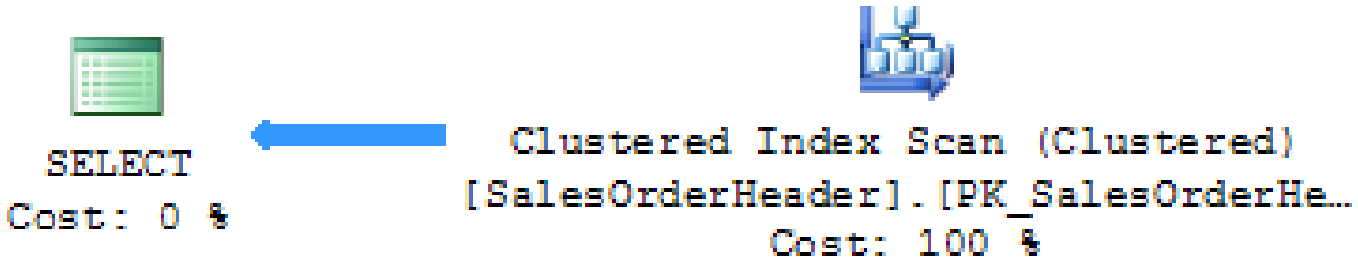
What is Parameter Sniffing?

▶ Execution of a stored procedure

- First call is with a low selective parameter

```
EXEC dbo.getSalesOrderHeader '20010701'
```

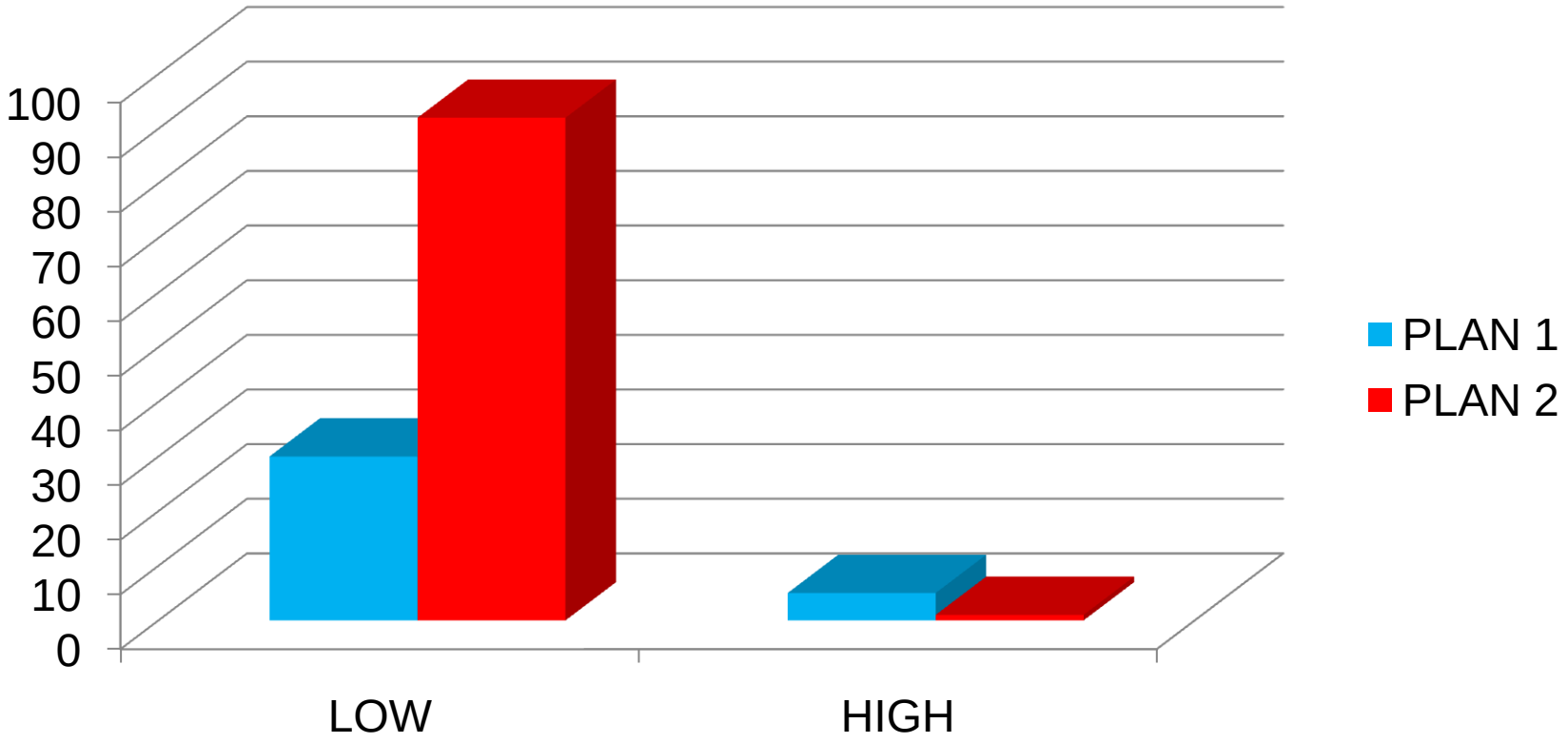
```
Table 'SalesOrderHeader'. Scan count 1, logical reads
1406, physical reads 0…
```



What is Parameter Sniffing?

▶ Execution of a stored procedure

- Execution time in milliseconds



When Parameter Sniffing Is a Problem?

- PS could be a problem when several execution plans are possible: some parameter combinations use a wrong plan
- When it is a problem? When the combinations are important or frequently used
- An exec. plan can be always removed from cache – we don’t have a control over this process

Parameter Sniffing Challenge

▶ What we can do?

- SQL Server Optimizer:
  - Consider input parameters
  - Reuse of execution plan
- What we can do? We can suggest to the optimizer:
  - Do not consider input parameters!
  - Do not “blindly” reuse a plan!

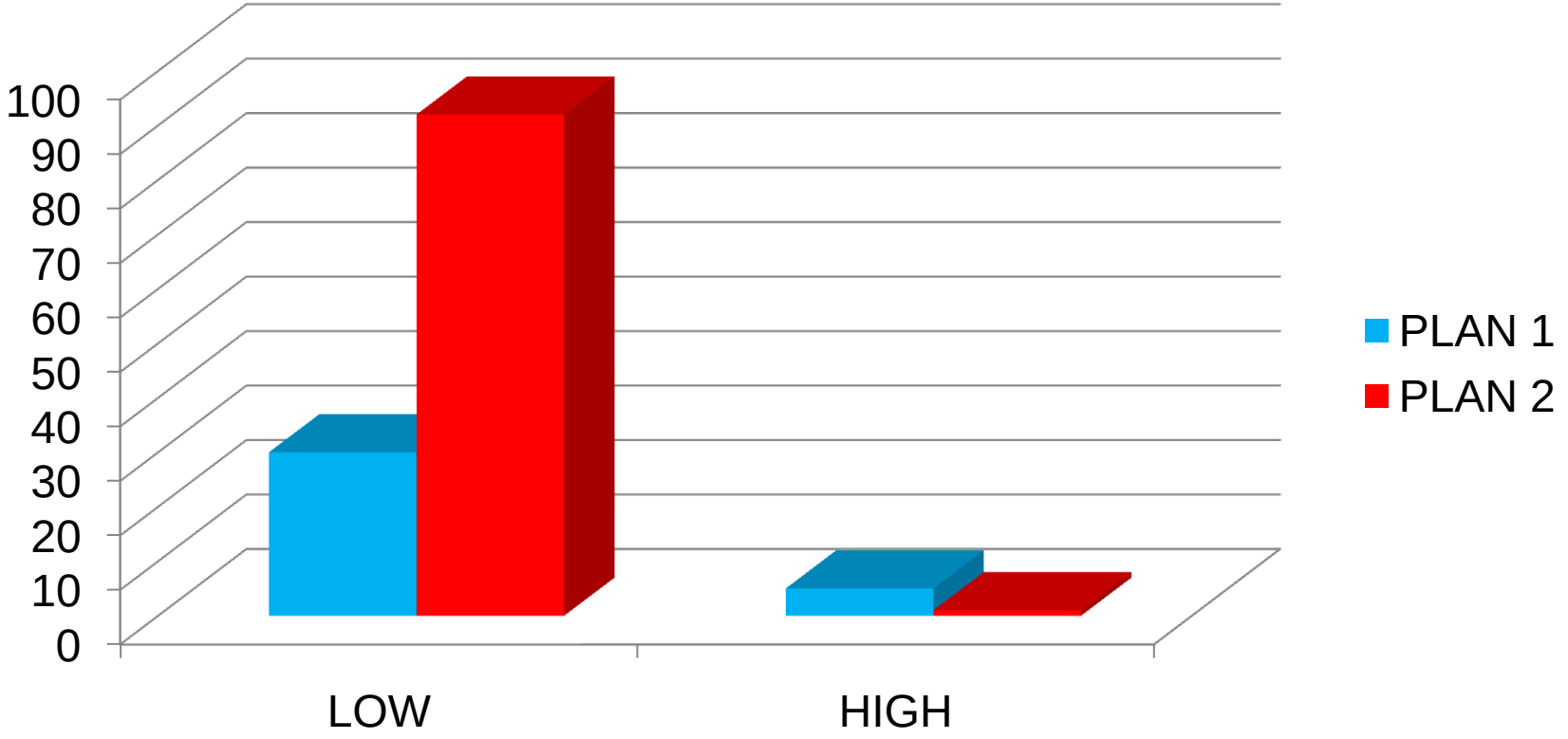
Parameter Sniffing Challenge

▶ Goals of Parameter Sniffing Optimization

- Two approaches:
  - Force the optimizer to generate one plan and use it for all parameter combinations
  - Leave the optimizer to generate a plan, but for every parameter combination use an optimal plan
- Which one to use?
  - Depends on Business Requirements

Parameter Sniffing Challenge

- Execution of a stored procedure

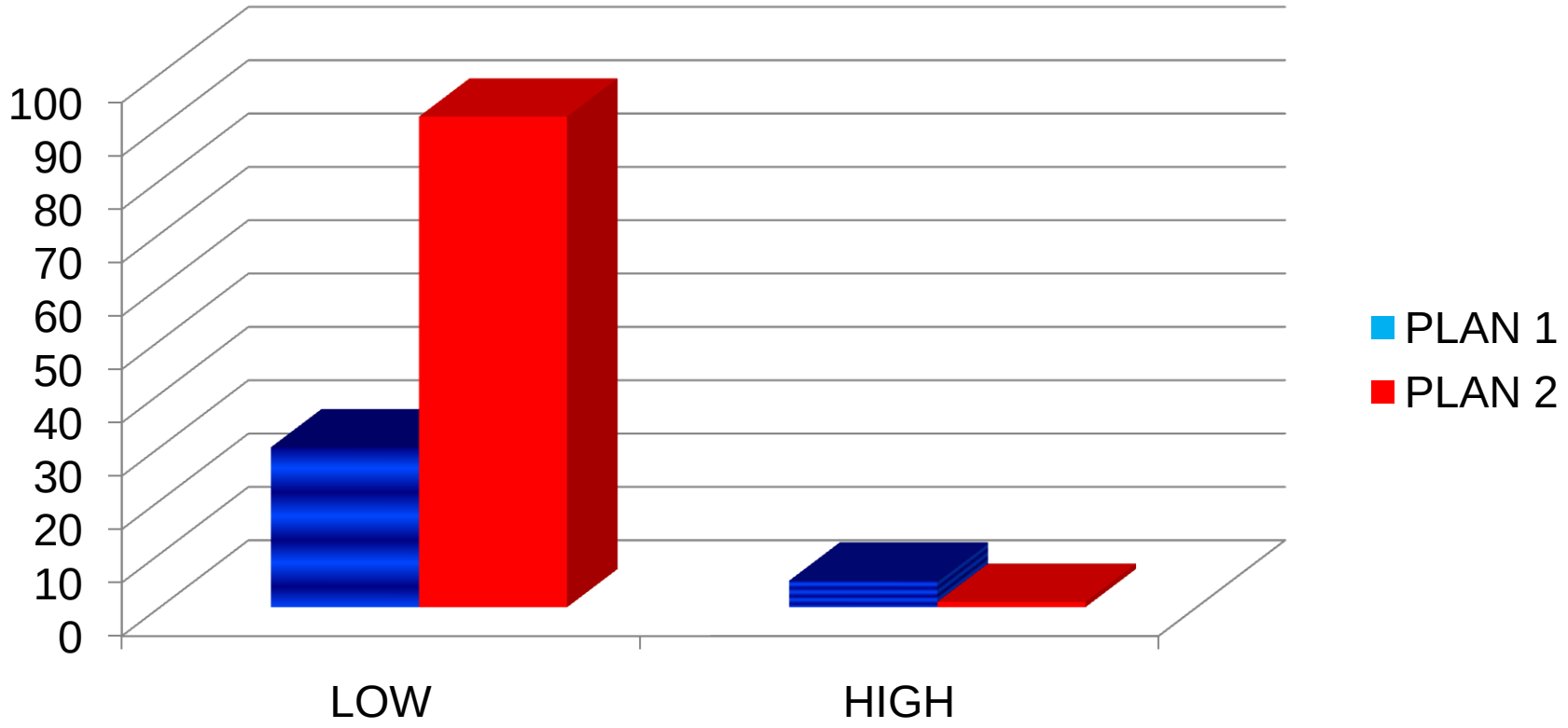


1. Blue plan
2. Red plan
3. Blue for LOW and Red for HIGH

Parameter Sniffing Challenge

▶ Option 1 – Using an Average Execution Plan

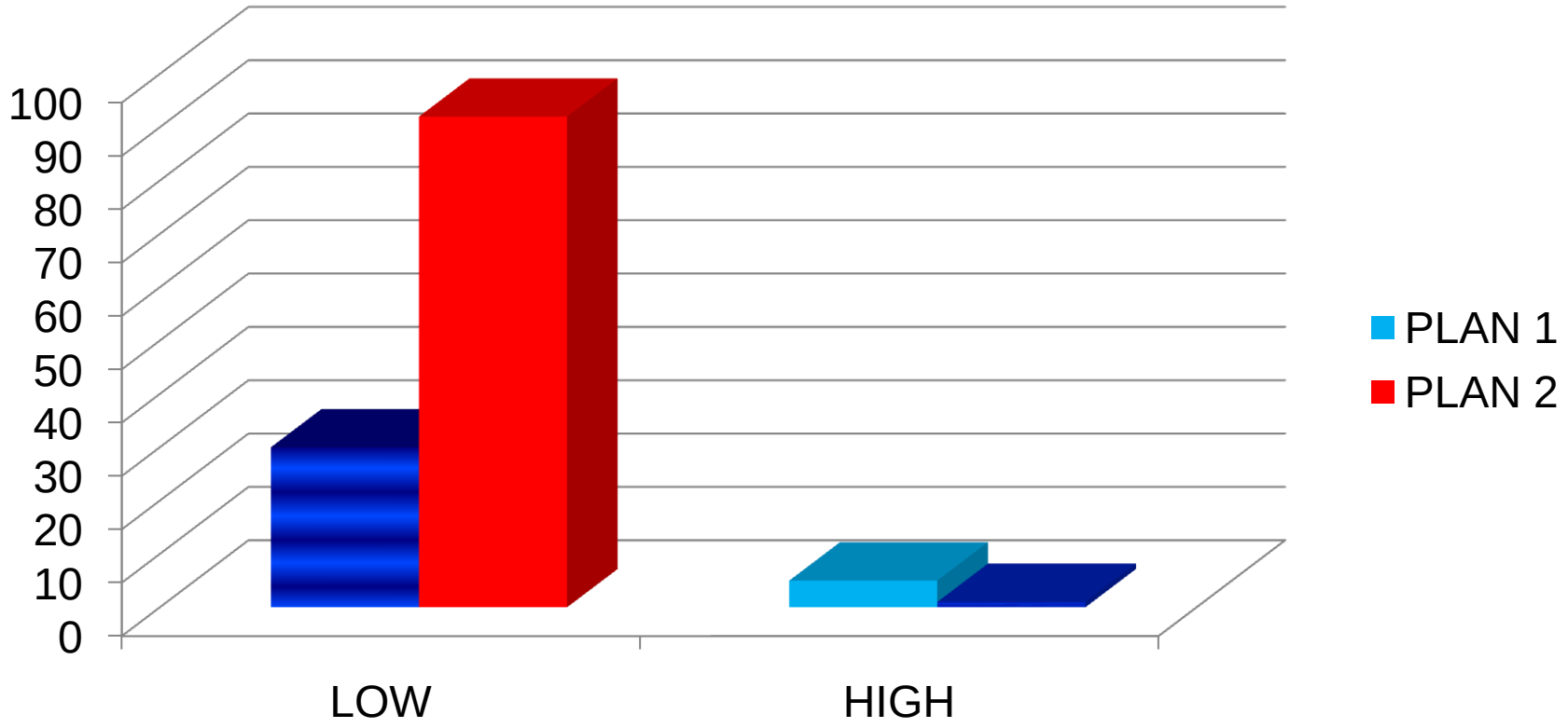
- Use the PLAN 1 for both parameter



Parameter Sniffing Challenge

▶ Option 2 – Using an Optimal Execution Plan

- Use a best possible plan for all parameters



Parameter Sniffing?

▶ Execution of a stored procedure

- Not only limited to stored procedures
  - Static parameterized queries
  - Dynamic queries executed with **sp_executesql**
- Stored procedures prone to parameter sniffing
  - stored procedures with parameters participating in range operators
  - stored procedures with optional parameters

SP with optional parameters

- A typical example is a web search page with optional search criteria
- The result set can be very large, but also empty
- It is challenge to optimize the implementation for all search requests
- Too many search criteria sometimes – application design problem

SP with optional parameters

- Sample table **`dbo.StudentExams`**

StudentExams

	Column Name	Data Type	Allow Nulls
🔑	exam_number	int	☐
	student_id	int	☐
	exam_id	tinyint	☐
	exam_note	tinyint	☐
	exam_date	datetime	☐
	exam_comment	char(500)	☑
			☐

Indexes: clustered on the `exam_number`

non-clustered on the `student_id` and `exam_date`

SP with optional parameters

- Requirements:
  - Two input parameters: **`student_id`** and **`exam_date`**
  - Both parameters are optional
  - The result set should contain only 20 rows sorted by **`exam_number`**

SP with Optional Parameters

▶ Solution 1

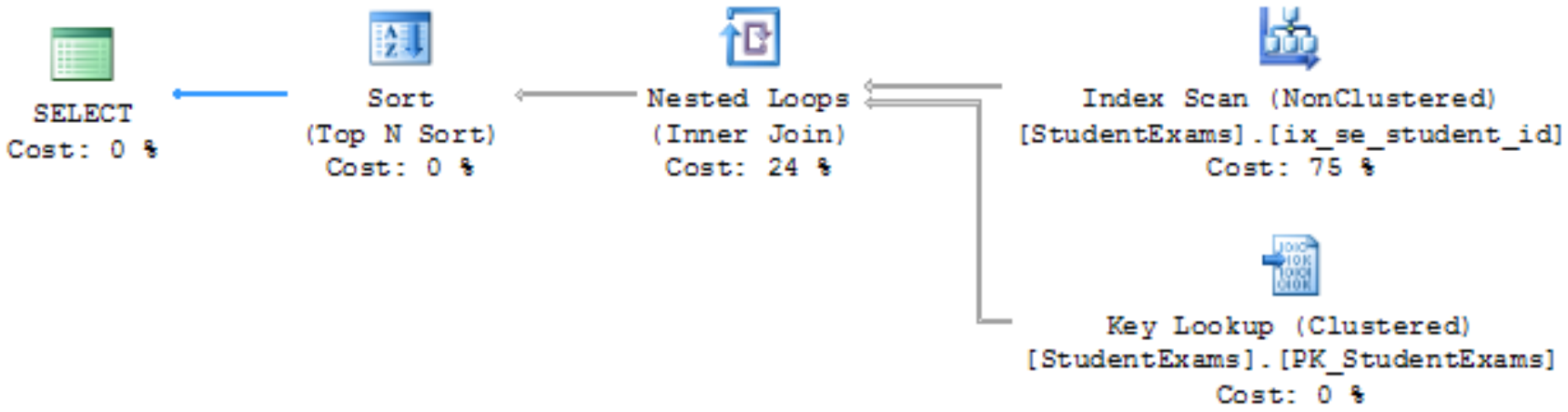
```
CREATE PROCEDURE dbo.getExams
@student_id int = NULL, @exam_date datetime = NULL
AS
BEGIN
        SELECT TOP (20) student_id, exam_number, exam_date
        FROM dbo.StudentExams se
        WHERE (student_id=@student_id OR @student_id IS NULL)
        AND (exam_date=@exam_date OR @exam_date IS NULL)
        ORDER BY exam_number
END
```

- Dynamic search conditions
- Three possible execution plans

SP with Optional Parameters

▶ Execution plan 1

```
DBCC FREEPROCCACHE
EXEC dbo.getExams 1
```

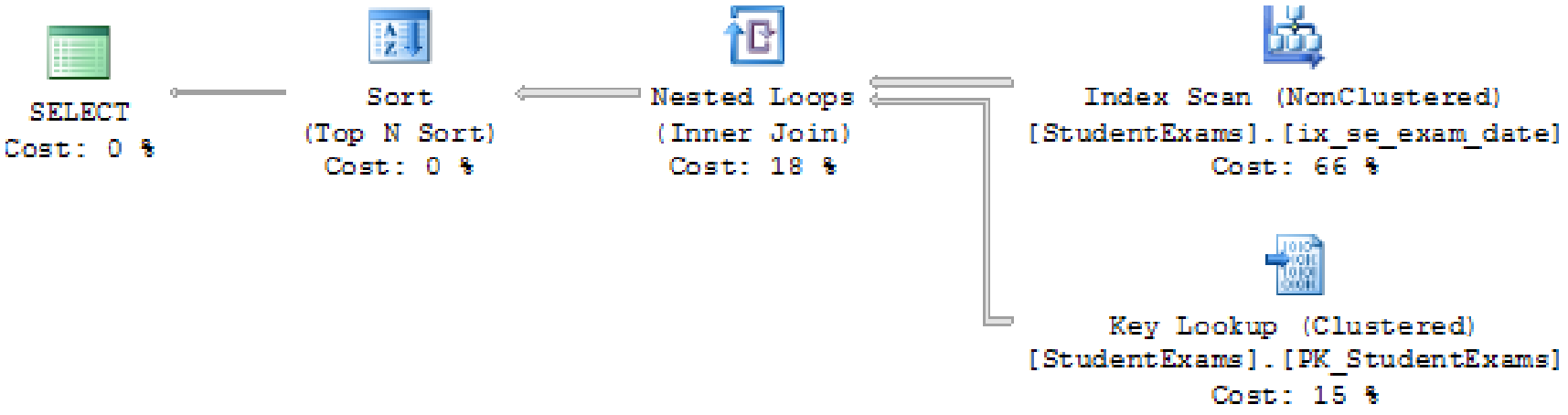



```
Table 'StudentExams'. Scan count 1, logical reads 1448, physical reads 0, read-ahead reads 0,
```

SP with Optional Parameters

- Execution plan 2

```
DBCC FREEPROCCACHE
EXEC dbo.getExams NULL, '20050731'
```

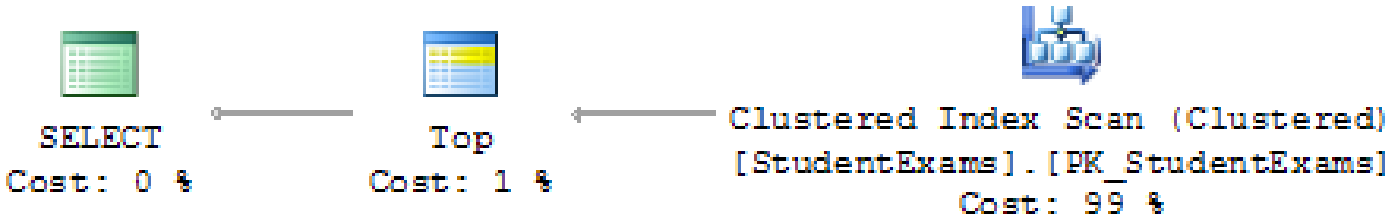



```
Table 'StudentExams'. Scan count 1, logical reads 2468, physical reads 0, re
```

SP with Optional Parameters

- Execution plan 3

```
DBCC FREEPROCCACHE
EXEC dbo.getExams
```




```
Table 'StudentExams'. Scan count 1, logical reads 9, physical reads 0,
```

SP with Optional Parameters

▶ What we can do?

```
DBCC FREEPROCCACHE
EXEC dbo.getExams 1
EXEC dbo.getExams
GO
DBCC FREEPROCCACHE
EXEC dbo.getExams
GO
```

```
Table 'StudentExams'. Scan count 1, logical reads 3001367, physical reads 0, read-ahead reads 0,
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
Table 'StudentExams'. Scan count 1, logical reads 12, physical reads 0, read-ahead reads 0, lob
```

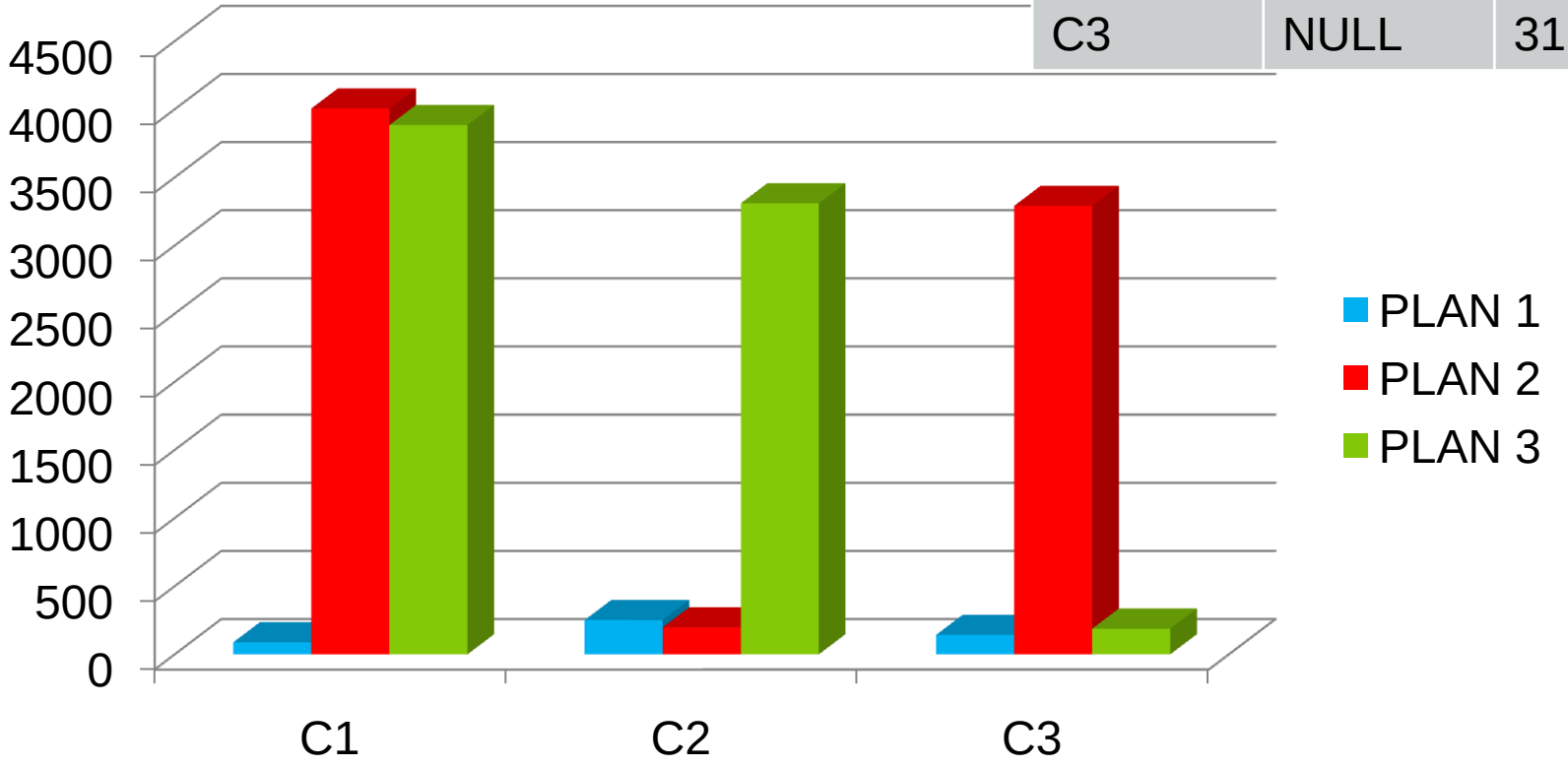
- If first call was with `student_id` than all invocations without `student_id` performs terrible (9 vs. 3 Mio logical reads!)
- Similar for the first invocation with `exam_date`
- First call without parameters seems to be acceptable

SP with Optional Parameters

▶ Suboptimal execution plans

- Execution time

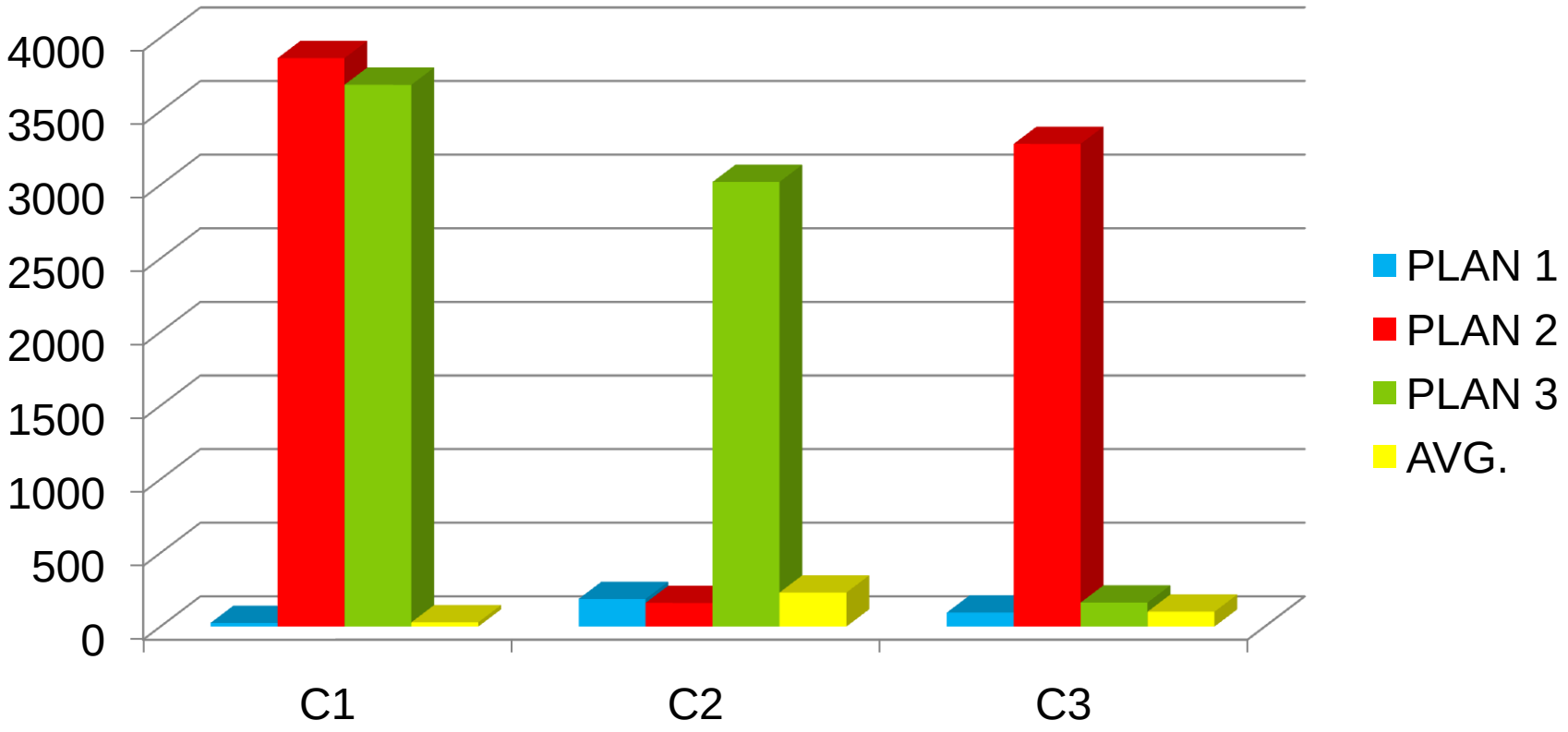
	student_id	exam_date
C1	NULL	NULL
C2	1	NULL
C3	NULL	31.07.05



SP with Optional Parameters

▶ What we can do?

- Use an “average” plan for all combinations



SP with Optional Parameters

▶ How to achieve it?

- Disable parameter sniffing
  - Using the OPTIMIZE query hint
  - Rewrite queries
  - Use trace flag 4136
- Optimize the execution for specific parameter combination

SP with Optional Parameters

▶ Disable Parameter Sniffing - Solution 1

- Disable parameter sniffing by using the OPTIMIZE query hint

```
CREATE PROCEDURE dbo.getExams
@student_id int = NULL, @exam_date datetime = NULL
AS
BEGIN
        SELECT TOP (20) student_id, exam_number, exam_date
        FROM dbo.StudentExams se
        WHERE (student_id=@student_id OR @student_id IS NULL)
        AND (exam_date=@exam_date OR @exam_date IS NULL)
        ORDER BY exam_number
        OPTION (OPTIMIZE FOR UNKNOWN)
END
```

SP with Optional Parameters

▶ Disable Parameter Sniffing - Solution 2

- Disable parameter sniffing by using local variables

```
CREATE PROCEDURE dbo.getExams
@student_id int = NULL, @exam_date datetime = NULL
AS
BEGIN
        DECLARE @s_id int = @student_id,
        DECLARE @e_date datetime = @exam_date
        SELECT TOP (20) student_id, exam_number, exam_date
        FROM dbo.StudentExams se
        WHERE (student_id=@s_id OR @s_id IS NULL)
        AND (exam_date=@e_date OR @e_date IS NULL)
        ORDER BY exam_number
END
```

SP with Optional Parameters

▶ Disable Parameter Sniffing - Solution 3

- Disable parameter sniffing by using trace flag 4136

- KB980653 http://support.microsoft.com/kb/980653/
- SQL Server 2008 R2 Cumulative Update 2
- SQL Server 2008 Service Pack 1 (SP1) Cumulative Update 7
- SQL Server 2005 Service Pack 3 (SP3) Cumulative Update 9

```
--at the connection level
DBCC TRACEON (4136)
--at the server level
DBCC TRACEON (4136, -1)
--check status
DBCC TRACESTATUS(4136)
```

SP with Optional Parameters

▶ Optimize for specific parameters - Solution 4

- Optimize execution for specific parameters

```
CREATE PROCEDURE dbo.getExams
@student_id int = NULL, @exam_date datetime = NULL
AS
BEGIN
        SELECT TOP (20) student_id, exam_number, exam_date
        FROM dbo.StudentExams se
        WHERE (student_id=@student_id OR @student_id IS NULL)
        AND (exam_date=@exam_date OR @exam_date IS NULL)
        ORDER BY exam_number
        OPTION (OPTIMIZE FOR (@student_id=NULL,@exam_date=NULL))
END
```

SP with Optional Parameters

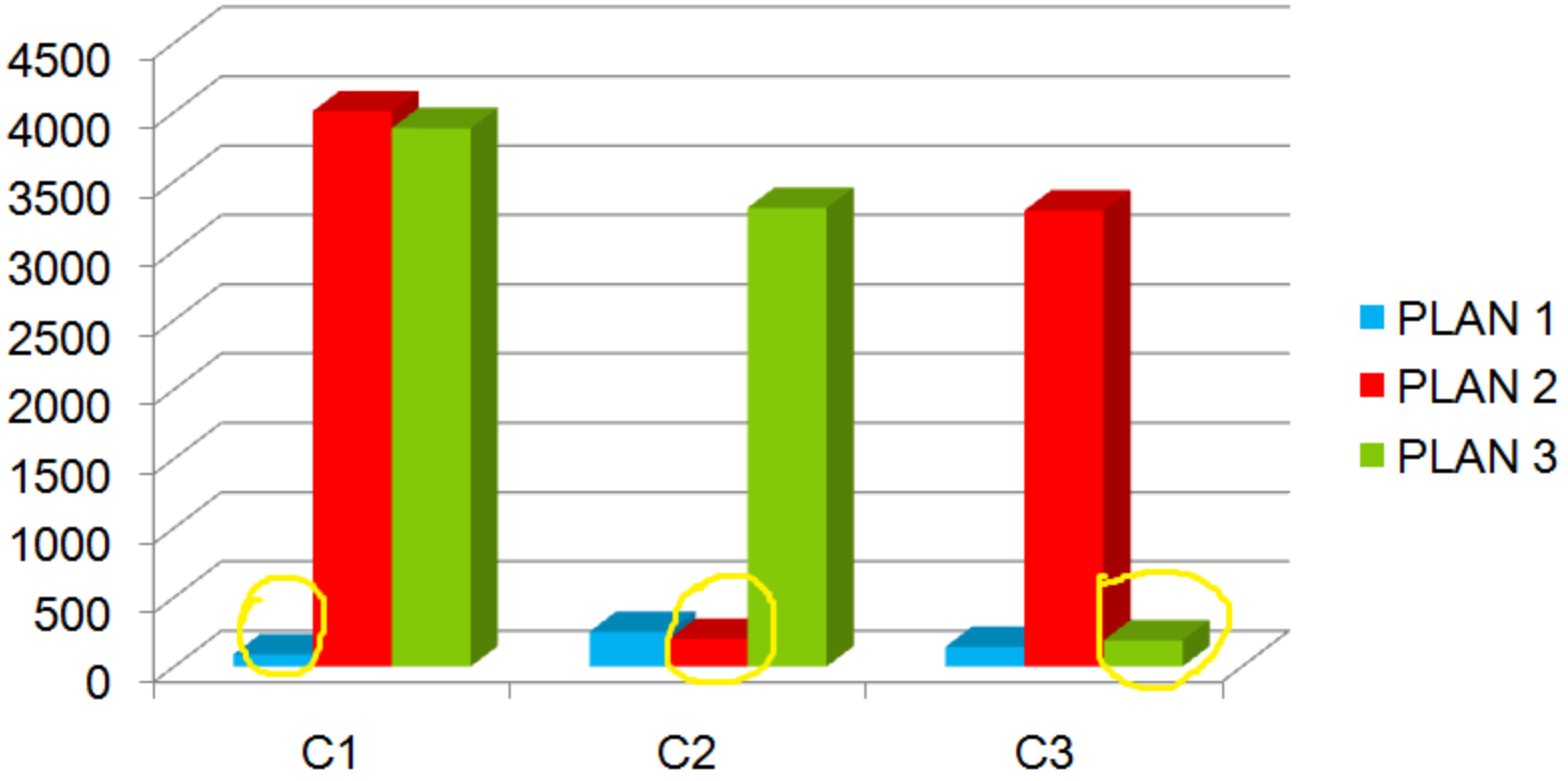
▶ Disable Parameter Sniffing Solutions – Which one to use?

- Favorite parameter combination => use the **`OPTIMIZE FOR`** query hint for this combination
- No favorite combination => disable parameter sniffing
  - Trace flag 4136 as last option
  - The query hints approach has advantage over the option with local variables
    - can be combined with plan guides and implemented without changing application code!
    - doesn't change business logic (can be done by DBA)
- **`OPTIMIZE FOR UNKNOWN`** is available from SQL Server 2008

SP with Optional Parameters

▶ What we can do?

- Use an optimal plan for every parameter combination



SP with Optional Parameters

▶ How to achieve it?

- Recompile
  - Using WITH RECOMPILE option
  - Using OPTION (RECOMPILE) query hint
- Rewrite Queries
  - Using decision tree stored procedures
  - Using dynamic sql

SP with Optional Parameters

▶ Solving Parameter Sniffing with Recompile - Solution 1

- Recompile when defining a stored procedure WITH OPTION

```
CREATE PROCEDURE dbo.getExams
@student_id int = NULL, @exam_date datetime = NULL
WITH RECOMPILE
AS
BEGIN
        SELECT TOP (20) student_id, exam_number, exam_date
        FROM dbo.StudentExams se
        WHERE (student_id=@student_id OR @student_id IS NULL)
        AND (exam_date=@exam_date OR @exam_date IS NULL)
        ORDER BY exam_number
END
```

SP with Optional Parameters

▶ Solving Parameter Sniffing with Recompile - Solution 2

- Recompile with the OPTION (RECOMPILE) query hint

```
CREATE PROCEDURE dbo.getExams
@student_id int = NULL, @exam_date datetime = NULL
AS
BEGIN
        SELECT TOP (20) student_id, exam_number, exam_date
        FROM dbo.StudentExams se
        WHERE (student_id=@student_id OR @student_id IS NULL)
        AND (exam_date=@exam_date OR @exam_date IS NULL)
        ORDER BY exam_number
        OPTION (RECOMPILE)
END
```

SP with Optional Parameters

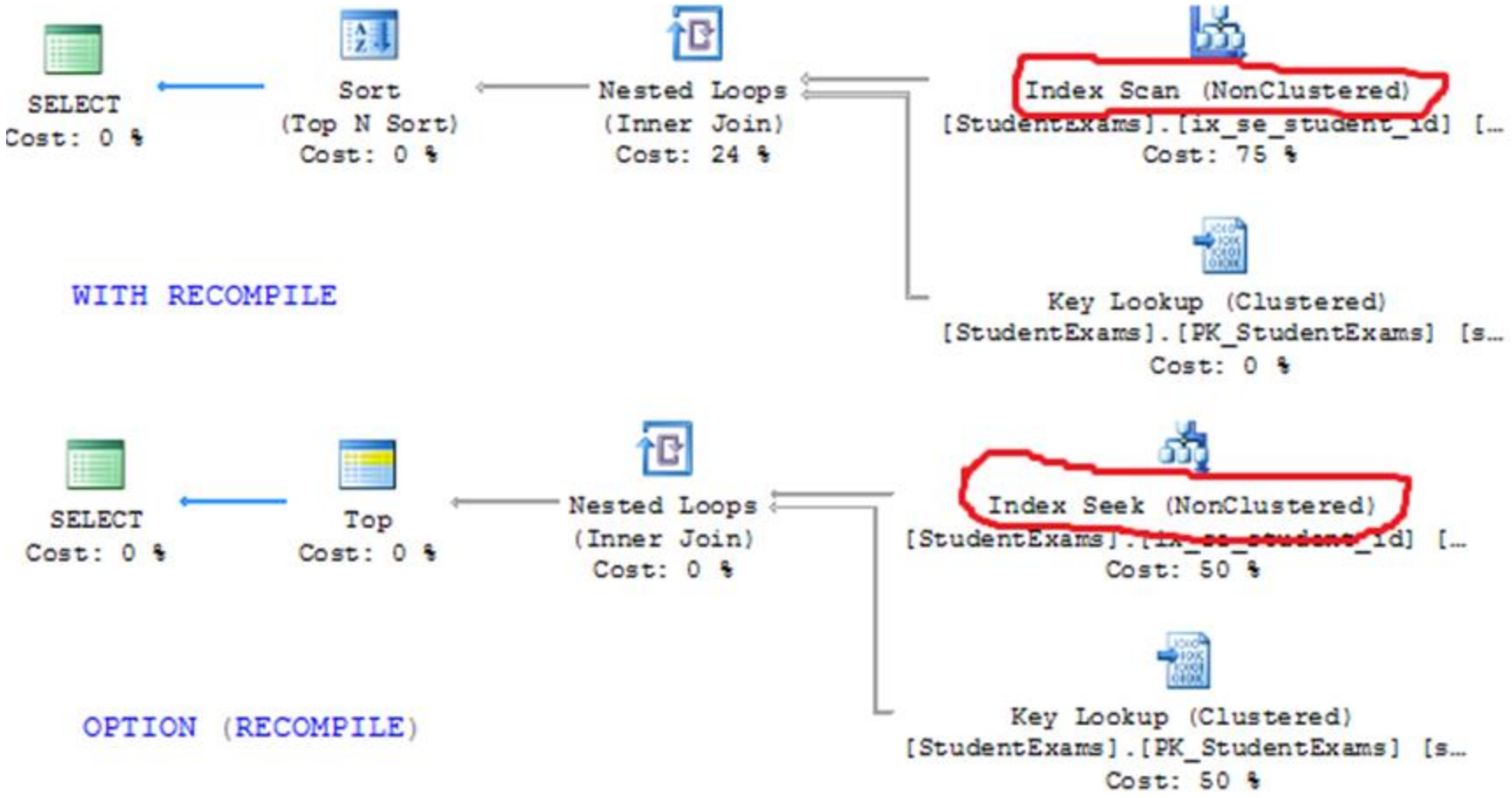
▶ How to achieve it?

- OPTION(RECOMPILE) query hint performs better than WITH RECOMPILE option, **better than initial ("its own") plan!**

- OPTION (RECOMPILE) compiles at the statement level and handles proper dynamic search filters

SP with Optional Parameters

- Benefits of the OPTION (RECOMPILE) query hint



SP with Optional Parameters

▶ Solving Parameter Sniffing with Decision Tree Stored Procedure - Solution 3

- Decision tree stored procedure (static SQL)

```
CREATE PROCEDURE dbo.getExams
@student_id int = NULL, @exam_date datetime = NULL
AS
BEGIN
        IF @student_id IS NOT NULL
           IF @exam_date IS NOT NULL
                EXEC dbo.StudentExams1 @student_id, @exam_date
           ELSE
                EXEC dbo.StudentExams2 @student_id
        ELSE
           IF @exam_date IS NOT NULL
                EXEC dbo.StudentExams3 @exam_date
           ELSE
                EXEC dbo.StudentExams4
END
```

SP with Optional Parameters

▶ Solving Parameter Sniffing with Dynamic SQL - Solution 4

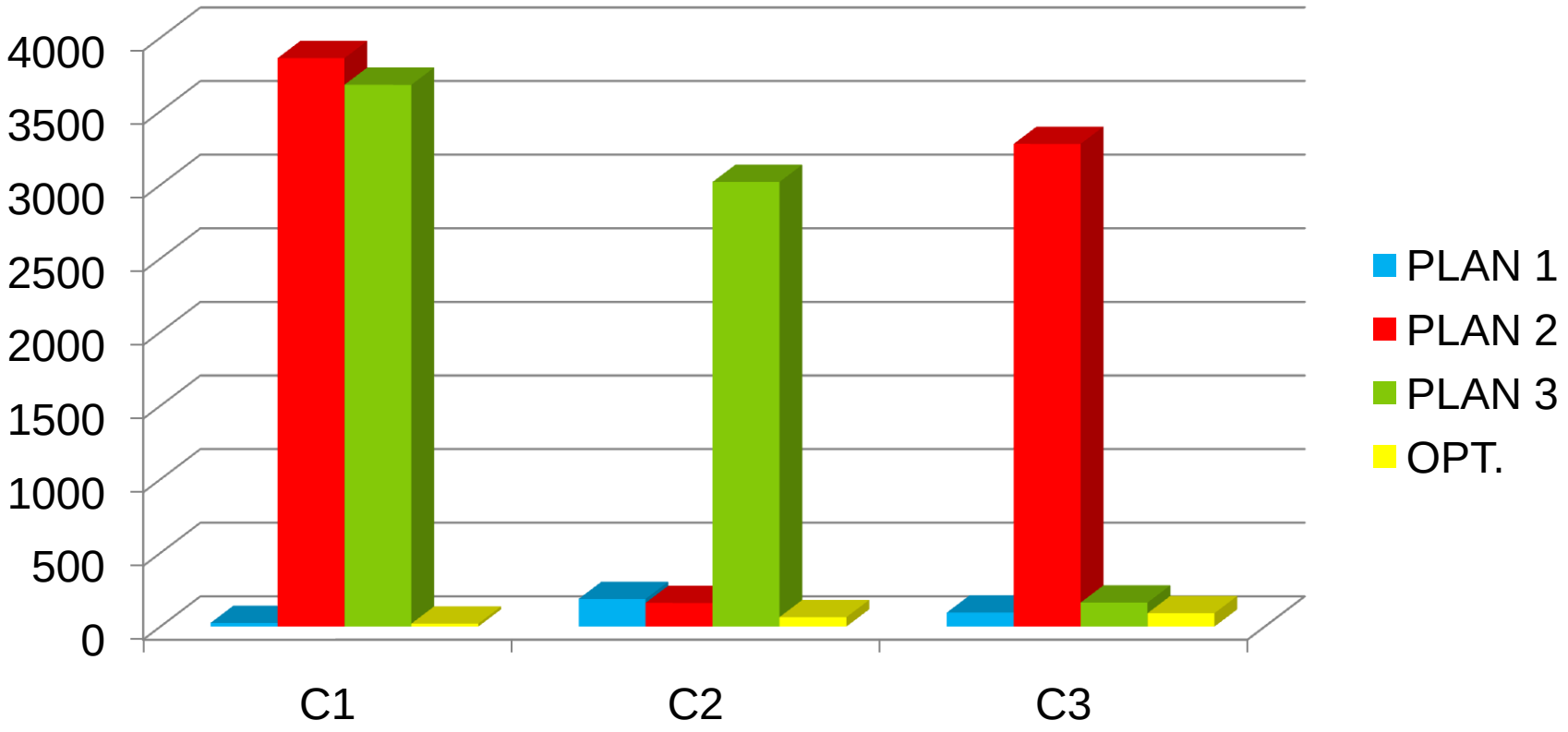
- Dynamic SQL

```
CREATE PROCEDURE dbo.getExams
@student_id int = NULL, @exam_date datetime = NULL
AS
BEGIN
        DECLARE @sql nvarchar(600) = N'SELECT TOP (20)
student_id, exam_number, exam_date FROM dbo.StudentExams WHERE
1=1  '
        IF @student_id IS NOT NULL
                SET @sql+=' AND student_id = ' + CAST(@student_id
AS nvarchar(8))
        IF @exam_date IS NOT NULL
                SET @sql+=' AND exam_date = ''' + CAST(@exam_date
AS nvarchar(8)) + ''''
        EXEC sp_executesql @sql, N'@sid int, @ed datetime',
@sid=@student_id, @ed=@exam_date;
END
```

SP with Optional Parameters

▶ What we can do?

- Use an optimal plan for all combinations



SP with Optional Parameters

- Which one to use?

	WITH RECOMPILE	OPTION (RECOMPILE)	DECISION TREE SP	DYNAMIC SQL
Optimal plan		x	x	x
Plan Reuse			x	x
Maintenance	x	x		
Additional Permissions				x
CPU Usage	x	x		

SP with Optional Parameters

▶ WITH RECOMPILE

- An execution plan for every parameter combination identical to initial one
- Manageable
- Requires CPU resources for every execution
- Execution plan is good, but not optimal

SP with Optional Parameters

▶ OPTION (RECOMPILE)

- Eliminate parameter sniffing problem
- Manageable
- An (really) optimal execution plan for all parameter combinations
- Can be combined with PLAN GUIDES
- Requires CPU resources for every execution

SP with Optional Parameters

▶ Decision Tree

- Optimal execution plan for every parameter combination
- Reuses an execution plan and doesn't need additional CPU resources
- Object maintenance problem:
  - Requires one sub-procedure for every execution plan we want to use
- Code maintenance problem:
  - Unmanageable for more than four possible execution plans

SP with Optional Parameters

▶ Dynamic SQL

- Optimal execution plan for every parameter combination
- Reuses an execution plan
- Can manage a lot of optional parameters
- Additional permissions required
- Code maintenance problem (error-friendly)
- Prone to SQL Injection

Parameter Sniffing

▶ Summary

- General recommendation: try to reuse an execution plan whenever is possible and makes sense
- Define the goal of the optimization
- Check business logic requirements
- Check application design

Parameter Sniffing

▶ Summary

- Use query hints, if it’s possible
- Recompiling use carefully
- OPTION (RECOMPILE) is better than WITH RECOMPILE
- Do not hesitate to combine both approaches
- Dynamic SQL is also one solution

Contact

- MRadivojevic@SolidQ.com

- milos.radivojevic@inhouse.wko.at

Parameter Sniffing with Stored Procedures

SOLID QUALITY MENTORS

Thank You!