Coding for Performance in Progress

Sean Overby

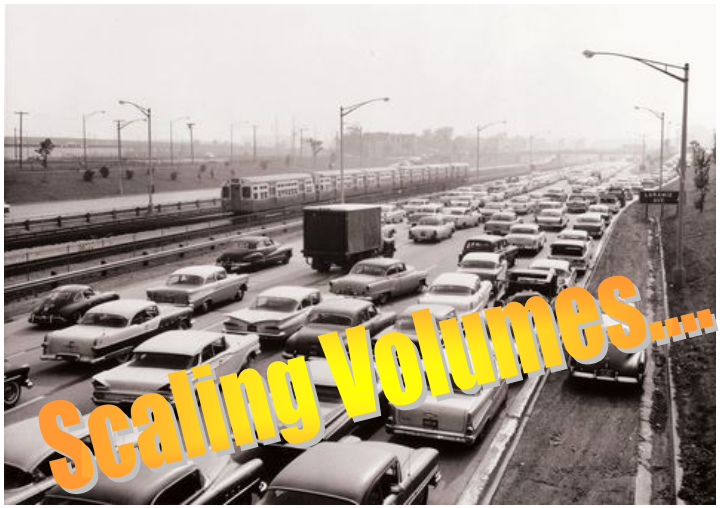
Senior Technologist, Solvepoint Corporation

SOLVEPOINT

SOLVEPOINT

Overview

- The lifeblood of a company is growth. Corporate growth puts increasing load on IT infrastructures.
- Managing increasing loads well is a key challenge to IT.
- All companies have a set of key algorithms and processes that are both computationally intensive and core to the functioning of the business.



Core Intensive Algorithms

- Material Requirement Planning runs.
- Financial Reporting.
- Item Pricing.
- Examples of your own?

When is 30ms too long?

- How many milliseconds are in 24 hours?
- How many 30ms operations can you complete in 24 hours?
- What if you have more operations to perform than there are hours in the day?

SOLVEPOINT

Core Algorithms

- Computational load of core algorithms will increase with scaling business volumes and require special care and handling.
- Consider your core algorithms, particularly ones that are, or may be, sensitive to load.
- Taking the time to reflect on future growth and it’s effects on your core processes now will save you major headaches and fire fighting later on.
- It’s not a question of if, it’s a question of when.
- If you actually get the time to do this (without making the time on your own) I’ll be stunned.

SOLVEPOINT

Caveats

- Your mileage not only can vary, it **will**. You must run your tests on your code on your servers.

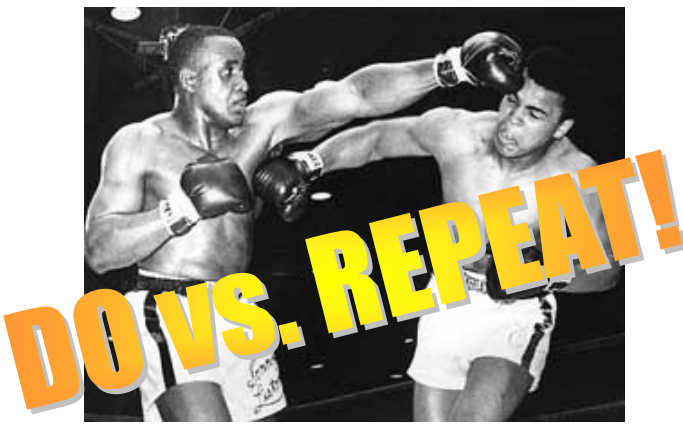
- While some of the next findings may be indicative of a persistent keyword variance, don't count on it. You may test examples shown here on your Linux server running Progress 9.1E05.04 (yes that's a feeble joke) and find out the exact opposite of what you see here.

- Most of the variances we will look at are small, remember that small variances add up over hours and days. Ask yourself how many repeat operations are performed enterprise wide during a 24 hour period in your IT infrastructure.

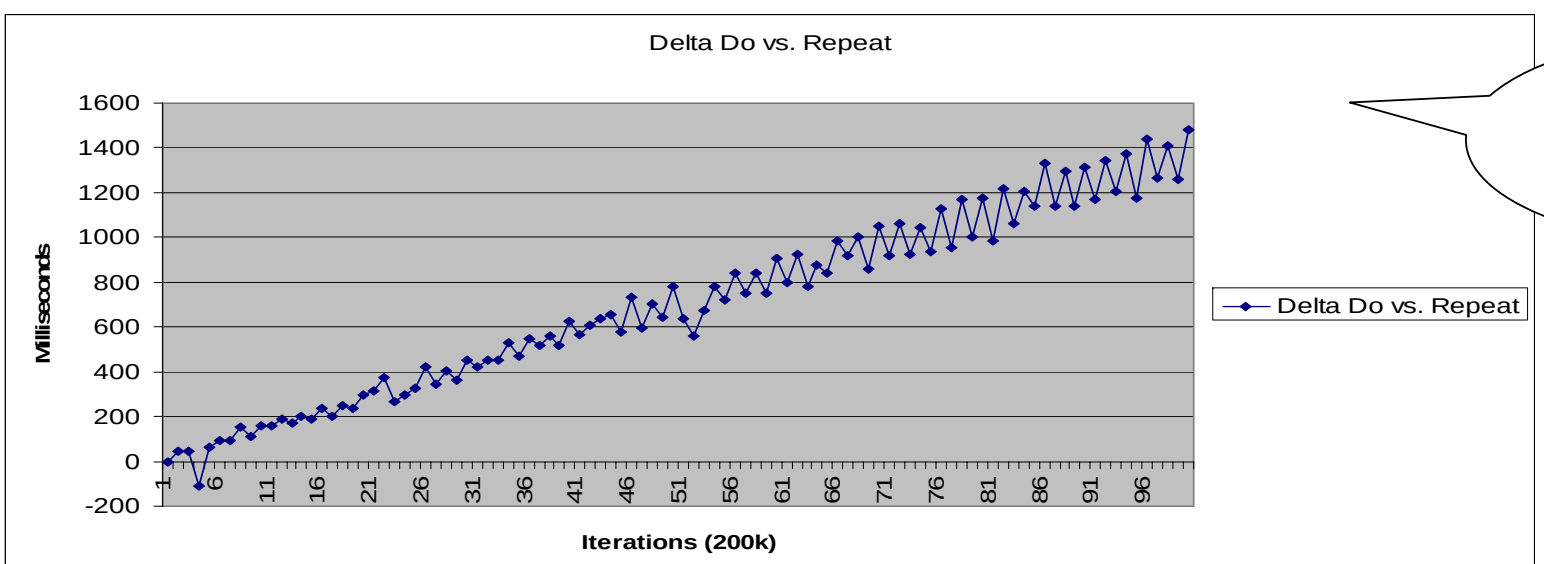
Block Processing

DO or REPEAT?

DO vs. REPEAT



- DO has **no** implicit transaction and buffer scoping, or undo handling.
- REPEAT has implicit transaction, buffer, and undo handling.
- How many block iterations take place in your Enterprise each day?



Increasing time delta, with repeat block taking an increasing amount of time

Code Blocks

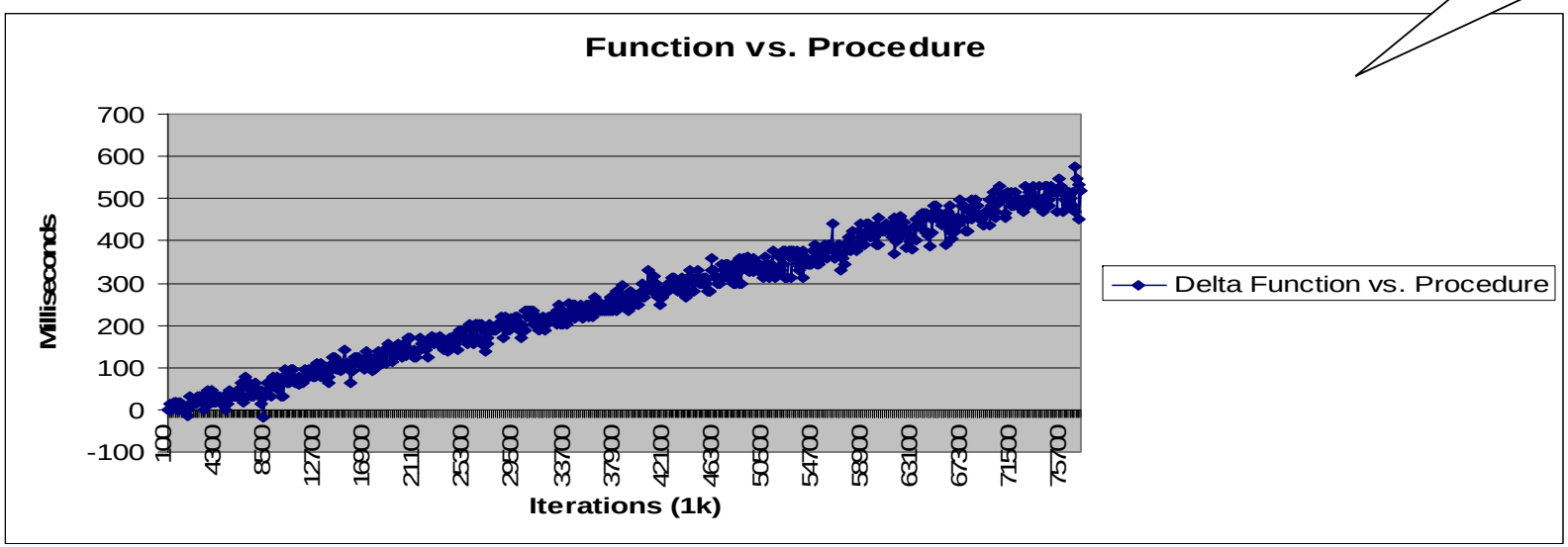
Procedure or Function?

Function vs. Procedure

- Functions have a single possible return value.
- Using a Function instead of a Procedure simply for the slight performance gain where a procedure is technically the correct answer…

Bonus Question: True or False, you can RETURN ERROR from a function?

Increasing time delta, with procedure calls taking an increasing amount of time



Variables

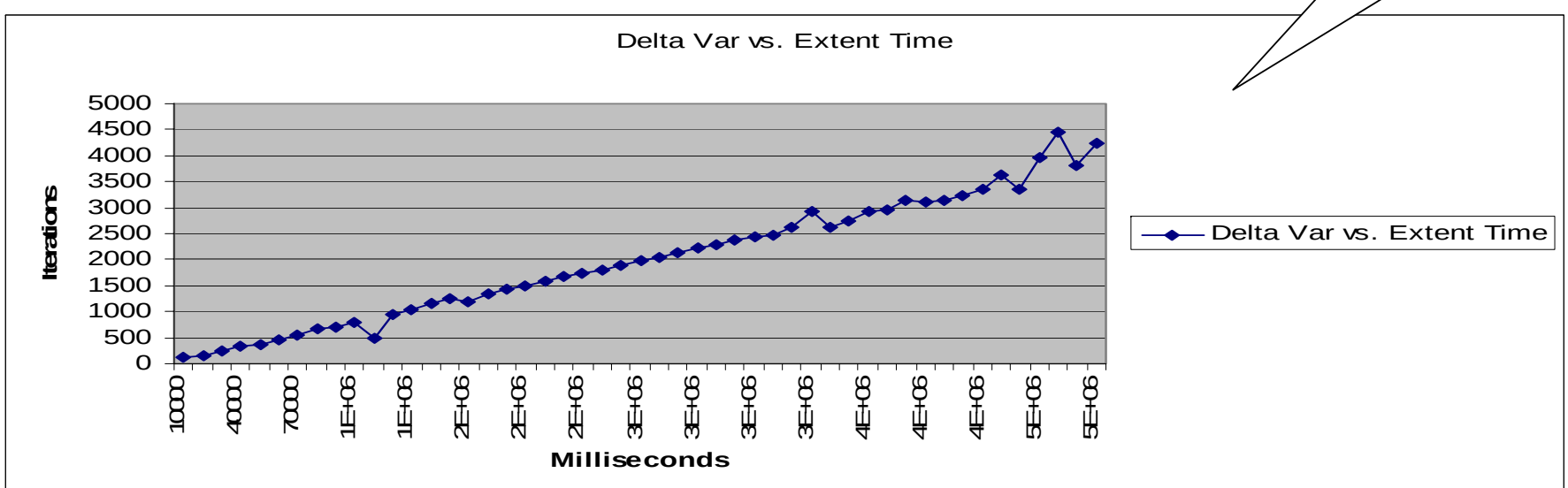
Array or Variable?

Variables

Variable Vs. Arrays

- During assignment, a regular variable will perform faster than an array element.
- Consider replacing array elements with single variables.

Increasing time delta, with the assignment of array extents taking an increasing amount of time



Functions as loop terminators

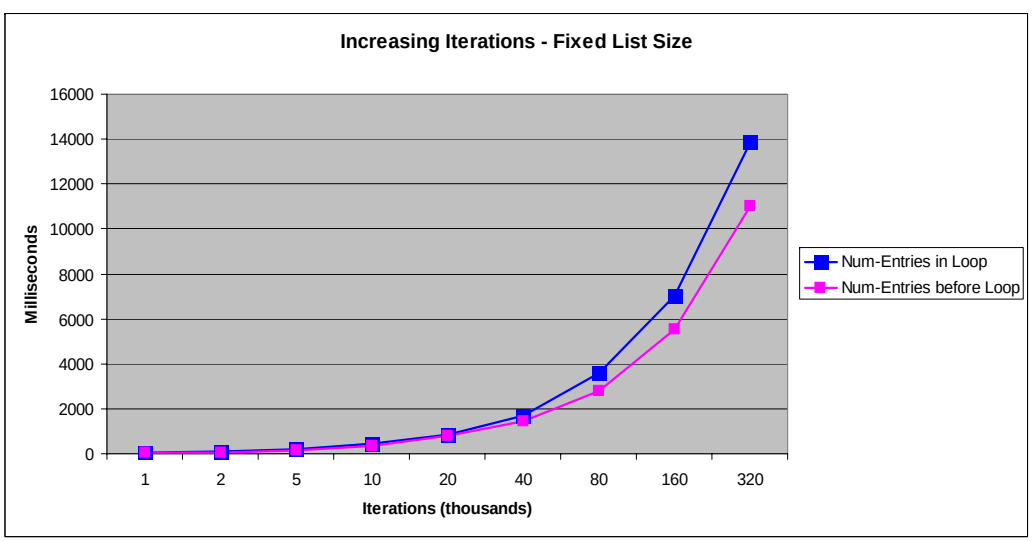
- DO cnt = 1 TO NUM-ENTRIES()

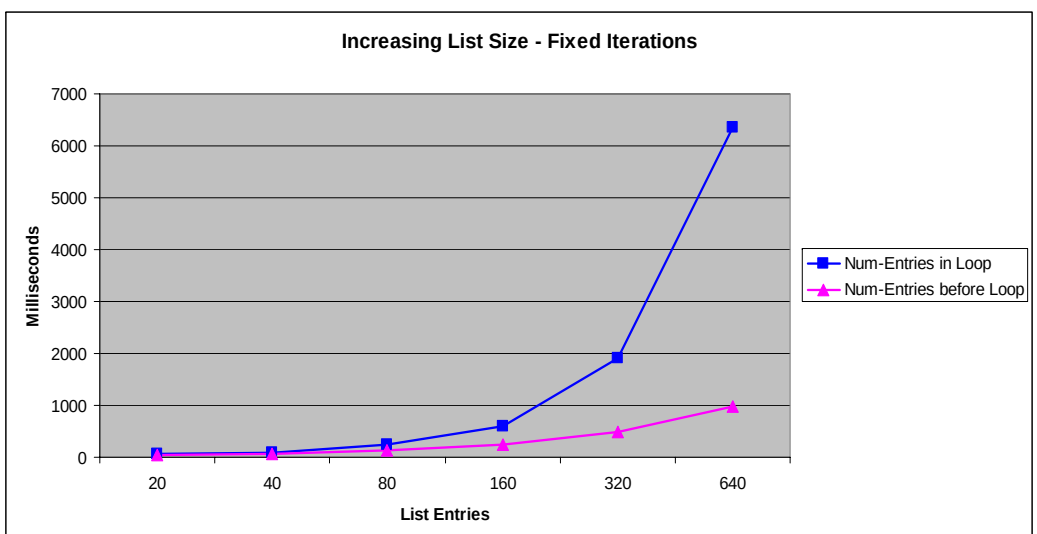
SOLVEPOINT

Functions as loop terminators

The Dreaded "Num-Entries" Loop

- Functions that do not need to be evaluated every loop should never ever go in the block statement.
- DO cnt = 1 to NUM-ENTRIES(someList) is the poster child...





SOLVEPOINT

Short Circuiting

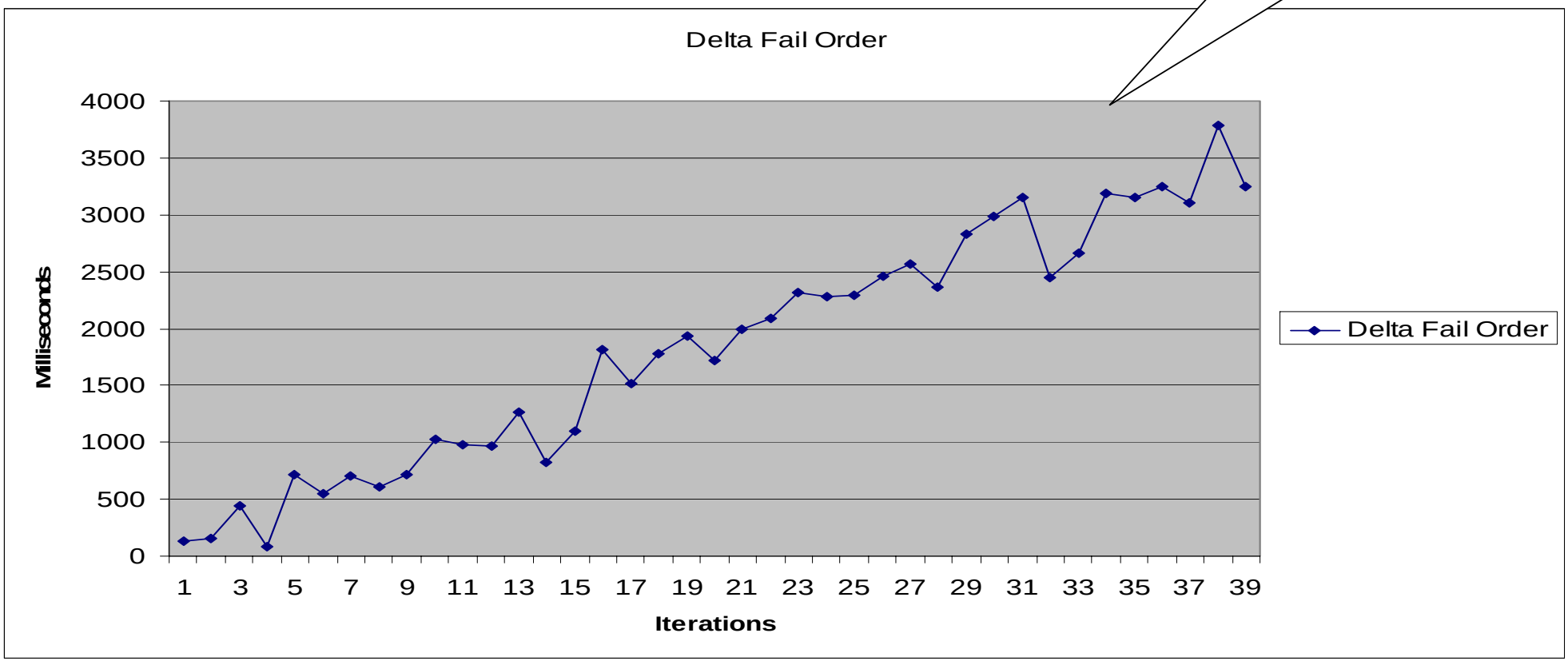
Ensure least expensive tests come first for short circuiting

- This is one of the first ‘low hanging’ fruit I look for after query structures, it’s been that significant in improving performance in production programs.
- Least expensive can be by operation type, e.g. CAN-FIND, or by highest likelihood of failure.
- ‘Fail fast’, boolean operation in and’ed groups should be in order of highest chance of failure from left to right.
- IF FailMost() AND ExpensiveDBOperation() OR FailMoreButNotAsMuch() AND EvenNastierExpensiveDBOperation() THEN

SOLVEPOINT

Short Circuiting

Increasing time delta, with least fail first taking an increasing amount of time



SOLVEPOINT

Short Circuiting

```
PROCEDURE test1:
   DO cnt = 1 TO limit:
      IF funcOne() AND funcTwo() AND funcThree()  THEN .
   END.
END PROCEDURE.

PROCEDURE test2:
   DO cnt = 1 TO limit:
      IF funcThree() AND funcTwo() AND funcOne()  THEN .
   END.
END PROCEDURE.
```

Bonus Question: How do you tell Progress to use the alternate random number generator?

```
FUNCTION funcOne RETURNS LOGICAL:
   DEFINE VARIABLE ist AS LOGICAL    NO-UNDO.
   DEFINE VARIABLE cnt AS INTEGER    NO-UNDO.
   DO cnt = 1 TO 10000:
   END.
   /** Return true 80% of the time **/
   ASSIGN ist = (RANDOM(1,100) <= 80).
   RETURN ist.
END FUNCTION.

FUNCTION funcTwo RETURNS LOGICAL:
   DEFINE VARIABLE cnt AS INTEGER    NO-UNDO.
   DEFINE VARIABLE ist AS LOGICAL    NO-UNDO.
   DO cnt = 1 TO 10000:
   END.
   /** Return true 60% of the time **/
   ASSIGN ist = (RANDOM(1,100) <= 60).
   RETURN ist.
END FUNCTION.
```

SOLVEPOINT

Conditional Assignment

```
IF this THEN ASSIGN x = that ELSE x = this.
```

OR

```
ASSIGN x = IF this THEN this ELSE that.
```

OR

```
ASSIGN x = this WHEN this
       x = that WHEN that.
```

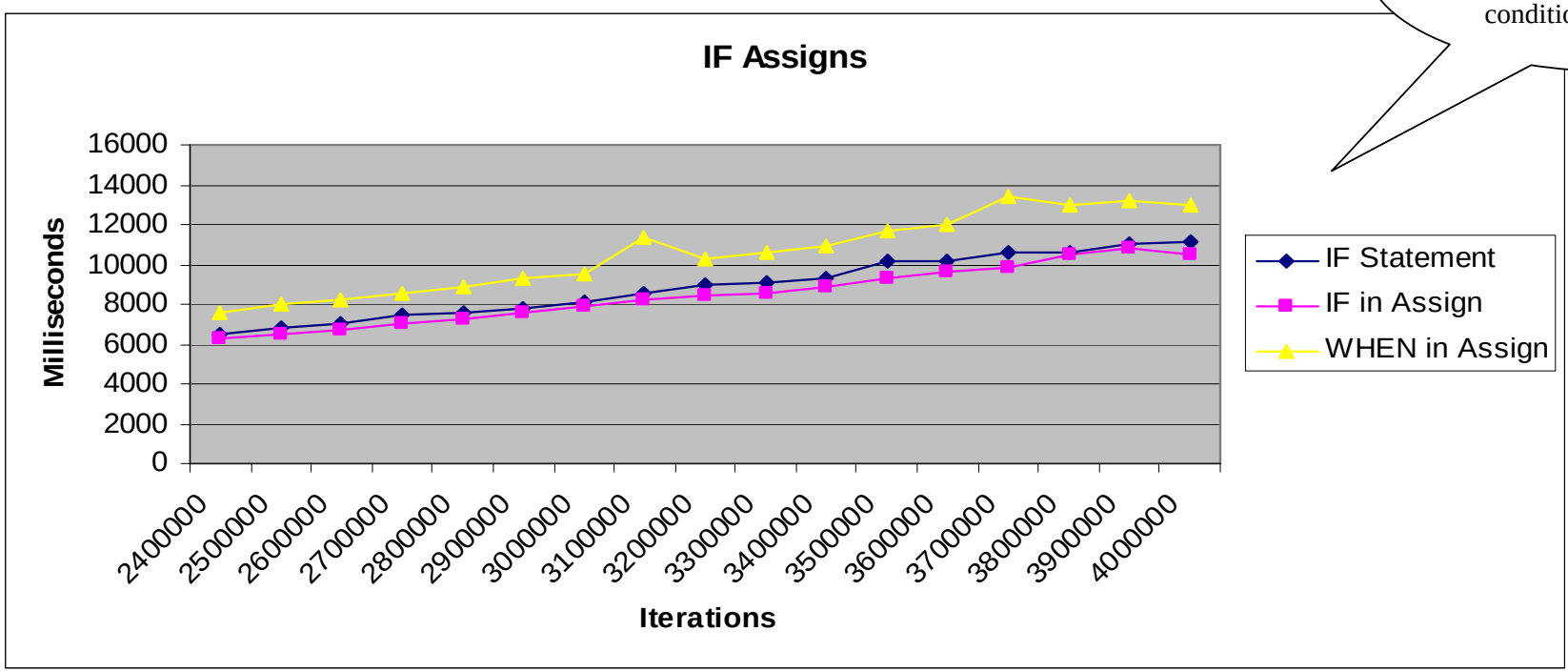
SOLVEPOINT

Conditional Assignment

- IF Statement - IF boolean operation THEN ASSIGN this ELSE ASSIGN that.
- IF in Assign - ASSIGN variable = IF this THEN this ELSE that.
- WHEN in Assign – ASSIGN variable = this WHEN some condition.

- The results of this surprised me.

No Delta here, just raw time scores showing the difference between various conditional assignment options



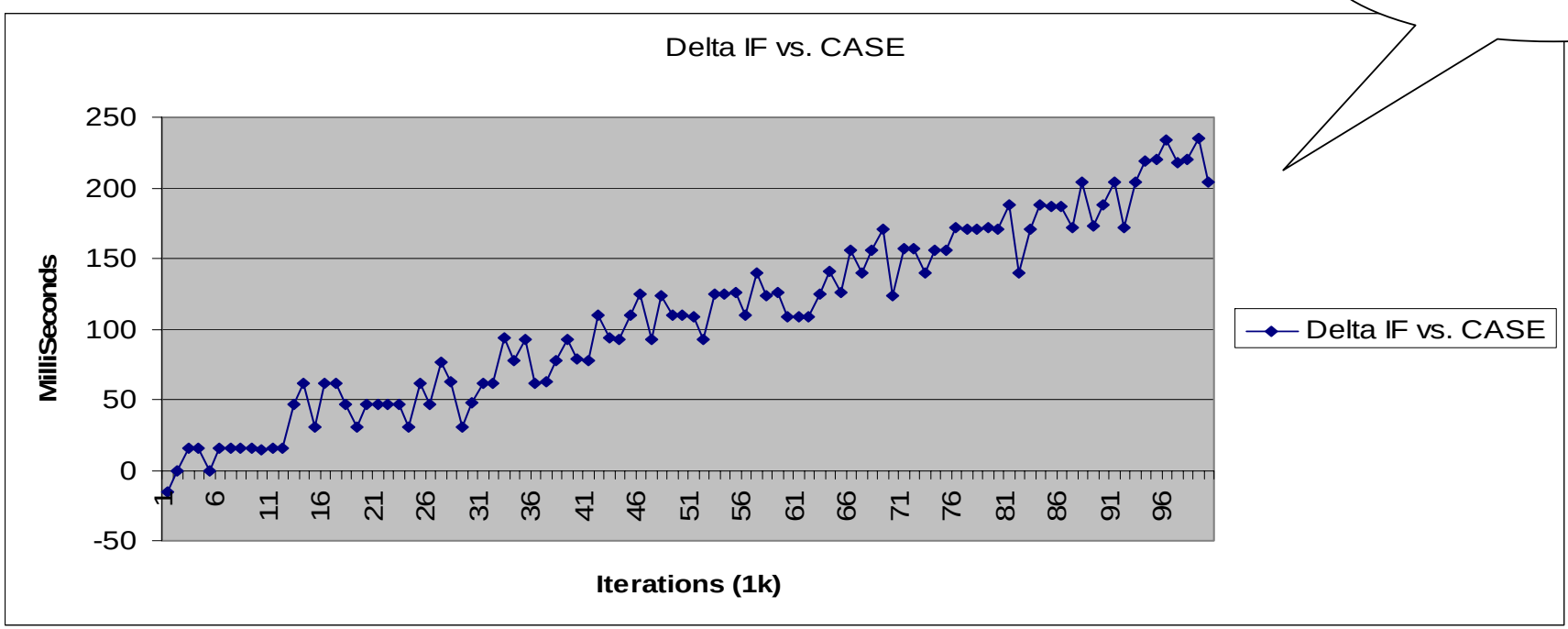
Conditional Evaluation

IF or CASE?

SOLVEPOINT

CASE vs. IF

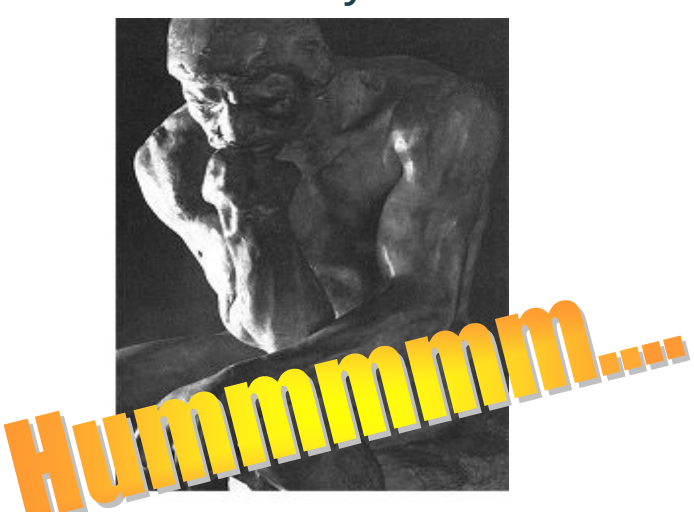
- Comparing last possible for seven choices.
- In this example use of an IF shows an increasing delta time versus the CASE statement.
- But… (queue the critical thinking lobe)



SOLVEPOINT

Critical Thinking

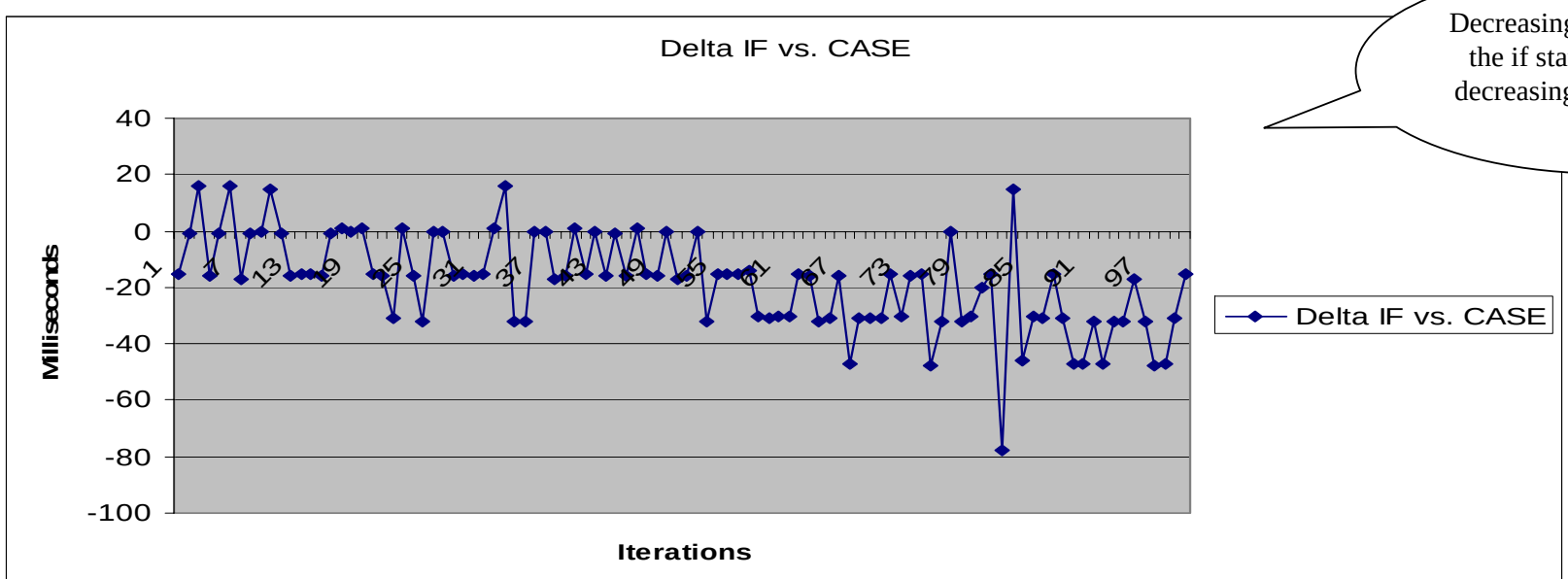
- Question our assumptions.
- Question our methods.
- Question our results.
- What are some issues with testing IF vs. CASE this way?




```
PROCEDURE doIf:
    IF cString = "ABC" THEN .
    ELSE IF cString = "BCD" THEN .
    ELSE IF cString = "CDE" THEN .
    ELSE IF cString = "DEF" THEN .
    ELSE IF cString = "EFG" THEN .
    ELSE IF cString = "FGH" THEN .
    ELSE IF cString = "GHI" THEN .
    ELSE IF cString = "HIJ" THEN .
END PROCEDURE.

PROCEDURE doCase:

    CASE cString:
      WHEN "ABC" THEN .
      WHEN "BCD" THEN .
      WHEN "CDE" THEN .
      WHEN "DEF" THEN .
      WHEN "EFG" THEN .
      WHEN "FGH" THEN .
      WHEN "GHI" THEN .
      WHEN "HIJ" THEN .
    END CASE.

END PROCEDURE.
```

Critical Thinking

- Highly unlikely that real-world code will have seven levels of IF/THEN/ELSE (we hope).
- Only testing the last condition is only testing it's most extreme point.
- As it turns out comparing the first condition gives the IF statement an edge over the CASE.



Decreasing time delta, with the if statement taking a decreasing amount of time

I/O

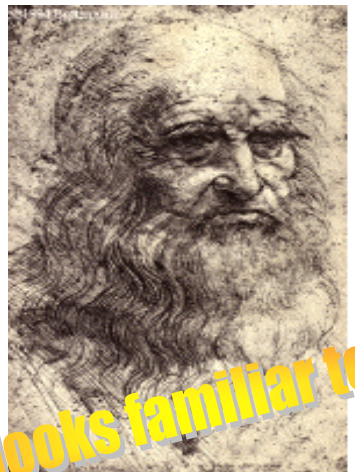
- What's the slowest part of the computer?
- No unbuffered output.
- No network drives.
- Preferably no internal File I/O whatsoever.
- Monitor temp file sizes and access.
  - Watch for growing/large SRT files – sorting on the client, check your indexes.
  - Growing large DBI files – Large Temp Tables, consider the –Bt startup parameter.
  - Growing large LBI files - Ensure NO-UNDO option on variables, check for many sub-transactions.

Bonus Question: What's the second slowest part of the computer?

SOLVEPOINT

The Scientific Method

- 1. Observation and description of a phenomenon or group of phenomena. **Test the code and gather performance statistics.**
- 2. Formulation of an hypothesis to explain the phenomena. In physics, the hypothesis often takes the form of a causal mechanism or a mathematical relation. **Create a metrics base-line.** ok I'm *reallyyyyy reaching here*
- 3. Use of the hypothesis to predict the existence of other phenomena, or to predict quantitatively the results of new observations. **Maintain metric base-lines through code revisions.**



SOLVEPOINT

Applied to Code Performance

- Question every piece of code in detail.
- Monitor code performance, CPU, I/O, Memory usage.
  - Compare keyword performance variances
    - DO vs. REPEAT
    - FUNCTION vs. PROCEDURE
    - IF vs. CASE
- Create base-line performance statistics.
- Update performance base-lines when deploying changes and keep a history.
- Repeat your experiments for every change.

SOLVEPOINT

What To Test

- Underlying keyword implementations may vary.
- For instance, DO vs. REPEAT. Do has **no** implicit transaction and buffer scoping or error retry behaviors. There is small but increasing delta between execution times under scaling loads.
- These deltas typically are tiny, for normal processing can be insignificant (but it still adds up). However for core business algorithms which need to scale under immense load they can add up to hours of CPU time.
- The delta is the important measurement, therefore underlying performance variations will be irrelevant. This is in contrast to measuring absolute performance where the underlying variations become a complicated variable in the measurement.

How to Test

- Tools
  - Windows: www.sysinternals.com, Process Explorer, FileMon
  - Unix: iostat, vmstat (or flavor specific)
- System clock resolution –be aware that your system clock resolution may not be what you think it is.
- Create a test harness program to allow for consistent test runs.
- There are two basic ways to simulate increasing load:
  - Increase Number of Iterations.
  - Increase Data Load (e.g. number of records).
  - Combining the two will give different results.

SOLVEPOINT

Applied to Code Performance

- Question every piece of code in detail.
- Monitor code performance, CPU, I/O, Memory usage.
  - Compare keyword performance variances
    - DO vs. REPEAT
    - FUNCTION vs. PROCEDURE
    - IF vs. CASE
- Create base-line performance statistics.
- Update performance base-lines when deploying changes and keep a history.
- Repeat your experiments for every change.

SOLVEPOINT

So what if it’s still to slow?

- After all possible tweaks and tunes have been made it’s still just not fast enough due to growth or other reasons.
- There are other options, such as taking the divide and conquer approach.
- But that’s a different presentation ;p.

Bonus Question: How many individual operations can a single Progress session perform simultaneously?

SOLVEPOINT

Summary

Key Points

- Identify and document processes that may not scale well under load or are subject to load increases based on business volume.
- Create baseline performance metrics.
- Run performance metrics after all changes, no matter how minor.
  - Performance on ‘live’ code that grows over-time can introduce a slow, subtle and hard to repair downward performance curve.
- Be wary of taking away any hard and fast rules from your observations as they may change with a patch, OS, or moon phase.

SOLVEPOINT

Summary

Key Points

- Question your methods, question your assumptions, question your results.
- Performance can and will be affected by outside influences.
  - Someone adds latest greatest new super report that just happens to attach the same database server your on.

SOLVEPOINT

Question & Answer

Thank you Pete!

Questions or Comments?

SOLVEPOINT

Thank You!

Thank you very much for your time and attention!

Sean A. Overby
Senior Technologist

Solvepoint Corporation
882 South Matlack Street, Suite 110
West Chester, PA 19382

Email: soverby@solvepoint.com

http://www.solvepoint.com

Building strong customer relationships through excellent service and delivery of advanced technology solutions.

SOLVEPOINT

Bibliography and Sources

- Introduction to the Scientific Method: http://teacher.nsrl.rochester.edu/phy_labs/AppendixE/AppendixE.html
- SysInternals Tools: http://www.sysinternals.com