# Robust Query Processing through Progressive Optimization

*Appeared in SIGMOD 2004*


Volker Markl, Vijayshankar Raman,
David Simmen, Guy Lohman,
Hamid Pirahesh, Miso Cilimdzic

*raja@cse.iitb.ac.in*

# Motivation

- Current optimizers depend heavily upon the cardinality estimations
- What if there errors in those estimations?
- Errors can occur due to ...
  - Inaccurate statistics
  - Invalid assumptions (e.g. attribute independence)

# Overview of talk ...

- What's this all about
- Contribution of the paper
- Related work
- Progressive Query Optimization(POP)
- CHECK and its variants
- Performance analysis
- A real-world experiment results

# Contribution

- Concept of CHECK and its various flavors
- Method for determining validity ranges for QEPs
- Performance analysis of prototype of POP

# Evaluating a re-optimization scheme

- Risk Vs Opportunity
- Risk:
  - Extent to which re-optimization is not worthwhile
- Opportunity:
  - Refers to the aggressiveness

# Background

- KD98
- Tukwila
- Telegraph
- Parametric optimization

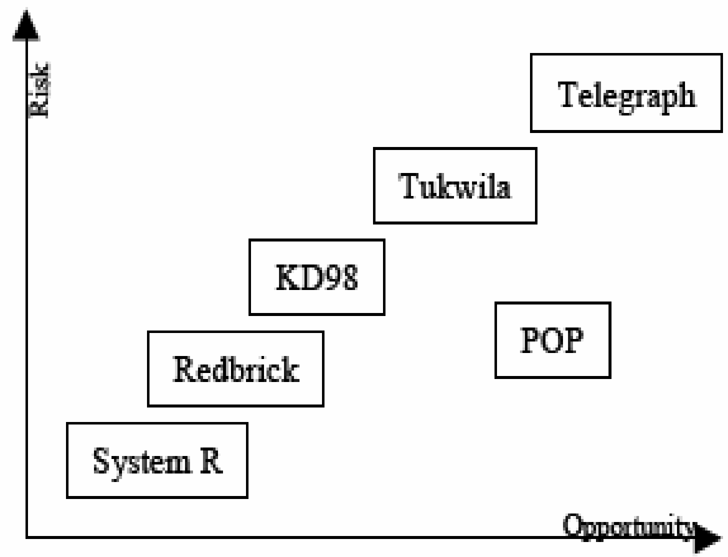
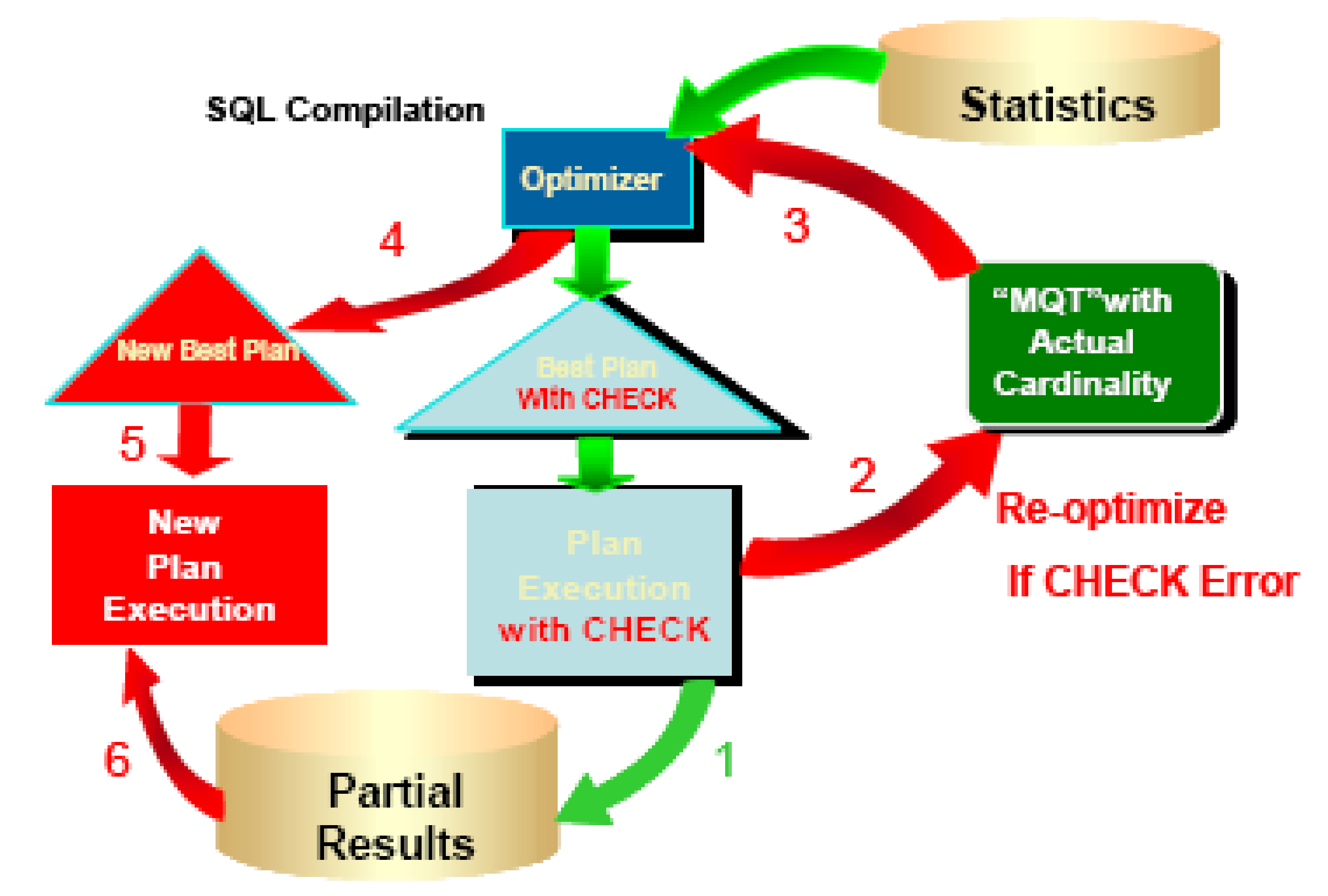



**Figure 1:** Risk/Opportunity Tradeoff of Various Re-Optimization Schemes

# Progressive Query Optimization(POP)



# Architecture of POP

- Find out valid ranges
- Location of CHECKs
- Executing CHECKs
- Interpret CHECK
- Exploit intermediate results

# Computation on Validity Ranges

- Validity range: is an upper and lower bound which when violated, guarantees that the current plan is sub-optimal wrt to the optimizers cost model
- No need to enumerate all possible optimal plans beforehand
- Uses modified Newton-Raphson method to find validity ranges

# Exploiting Intermediate Results

- All the intermediate results are stored as temporary MVs
- Not necessarily written out to disk
- In the end, all these temporary MVs needs to be deleted (extra overhead?)

# Variants of CHECK

- Lazy checking
- Lazy checking with eager materialization
- Eager checking without compensation
- Eager checking with buffering
- Eager checking with deferred compensation

# Variants of CHECK (contd.)

- LC:
  - Adding CHECKs above a materialization point (SORT, TEMP etc)
  - As, no results have been output yet
  - And materialized results can be re-used
- LCEM:
  - Insert materialization point if it does not exists already
  - Typically done only for nested-loop join

# Eager Checking (EC)

- EC without Compensation:
  - CHECK is pushed down the MP
- EC with buffering
  - CHECK and buffer

# Eager Checking with pipelining

- EC with Deferred Compensation
  - Only SPJ queries
  - Identifier of all rows returned to the user are stored in a table S, which is used later in the new plan for anti-join with the new-result stream

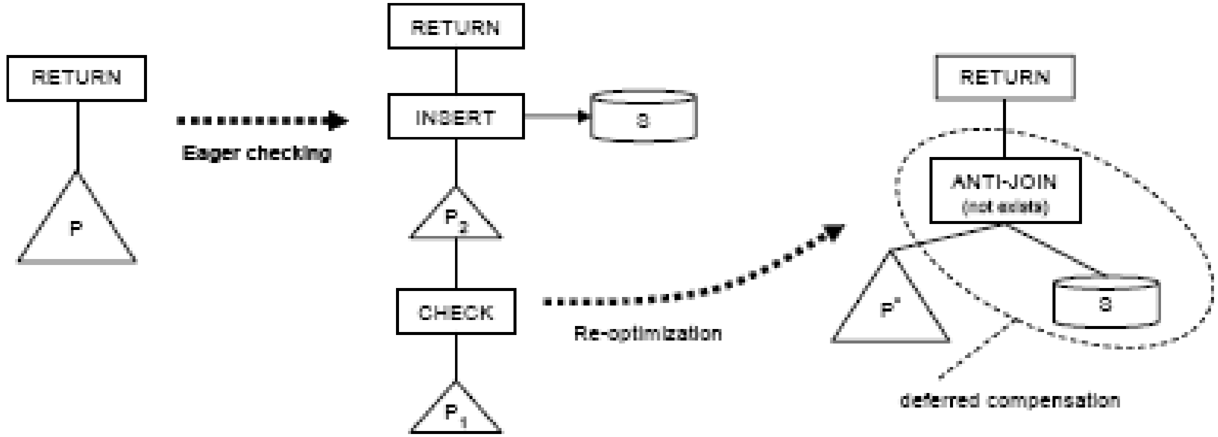



**Figure 9** Eager checking with deferred compensation

**Table 1:** Placement, Risk and Opportunity for various flavors of checkpoints

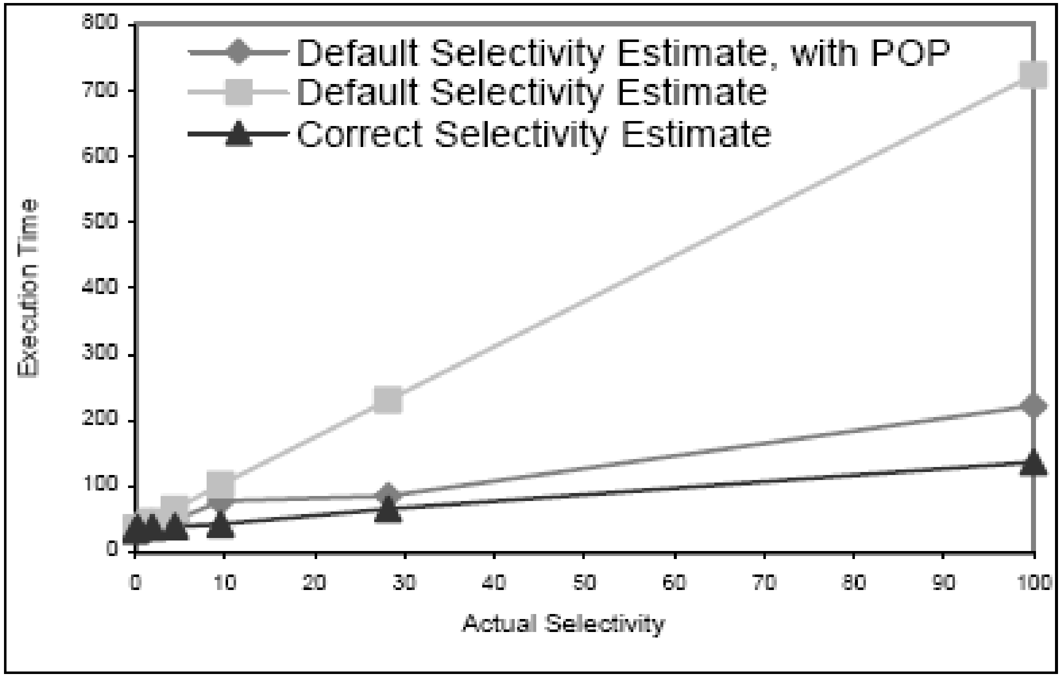
| Checkpoint Type | Placement | Risk | Opportunity |
|---|---|---|---|
| Lazy Check (LC) | CHECK Above materialization points | Very Low -- only context switching | Low, only at materialization points |
| Lazy Check with Eager Materialization (LCEM) | CHECK-Materialization pairs on outer of NLJN | Context Switching + materialization overhead | Materialization points and NLJN outers |
| Eager Check with Buffering (ECB) | BUFCHECK on outer of NLJN. | High – exact cardinality of subplan below ECB not available | Can reoptimize anytime during materialization |
| Eager Check without compensation (ECWC) | CHECK below materialization points | High – may throw away arbitrary amount of work during reoptimization | Anywhere below a materialization point |
| Eager Check with deferred compensation (ECDC) | CHECK and INSERT before reoptimization; anti-join afterwards | High – may throw away arbitrary amount of work during reoptimization | Anywhere in the plan of an SPJ-query |

# CHECK Placement

- LCEM and ECB – outer side of nested-loop join
- LC – above materialization points
- ECWC and ECDC – anywhere

# Performance Analysis

- Robustness



# Performance Analysis cont …

- Risk Analysis

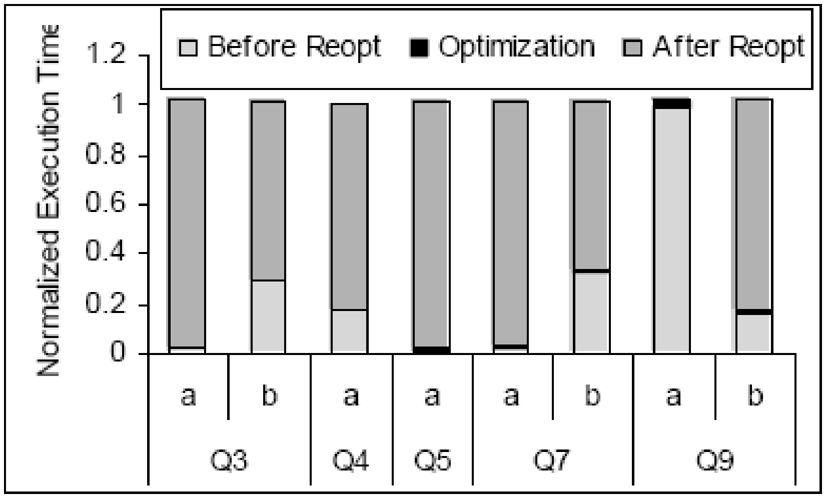



**Figure 12**: Normalized Execution time with LC re-optimization (1 is the execution time without re-optimization)

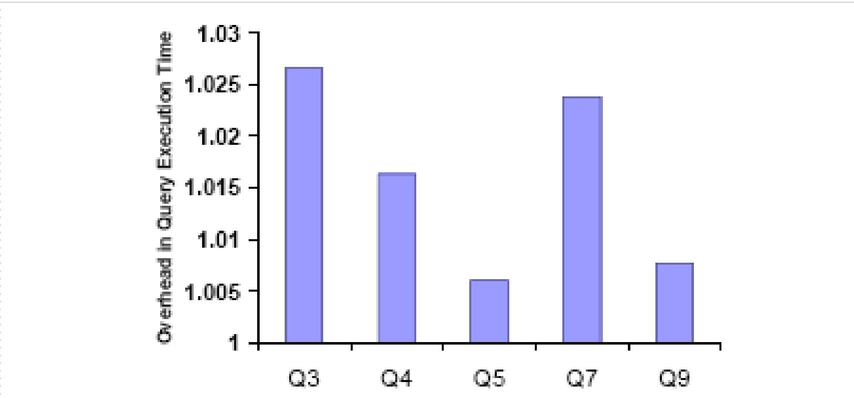



**Figure 13**: Cost of Lazy Checking with Eager Materialization

# Opportunity Analysis

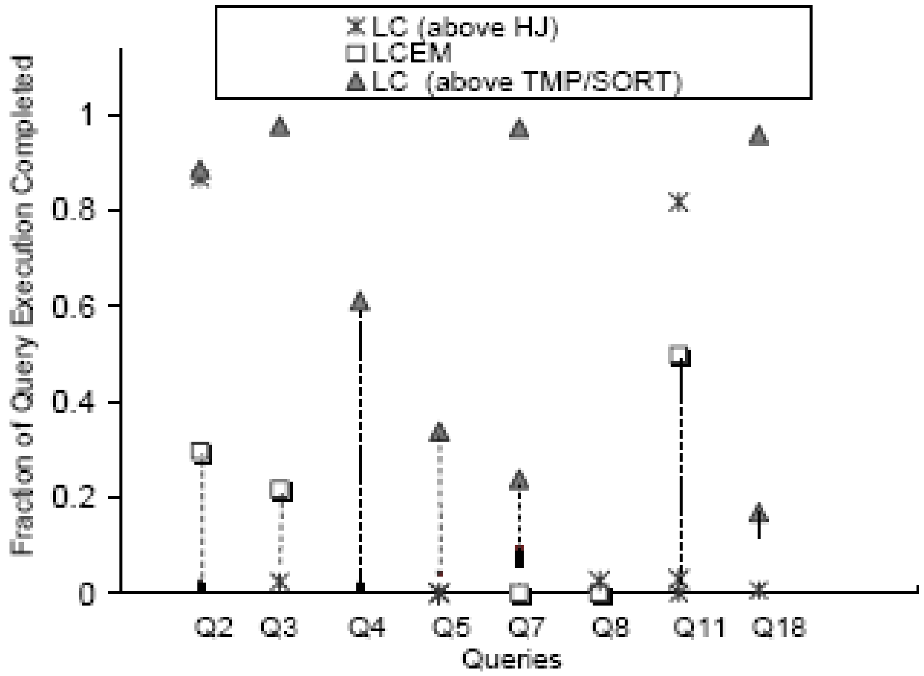



**Figure 14**: Opportunities for various kinds of checkpoints

# POP in (in)action

- 22 Vs 17

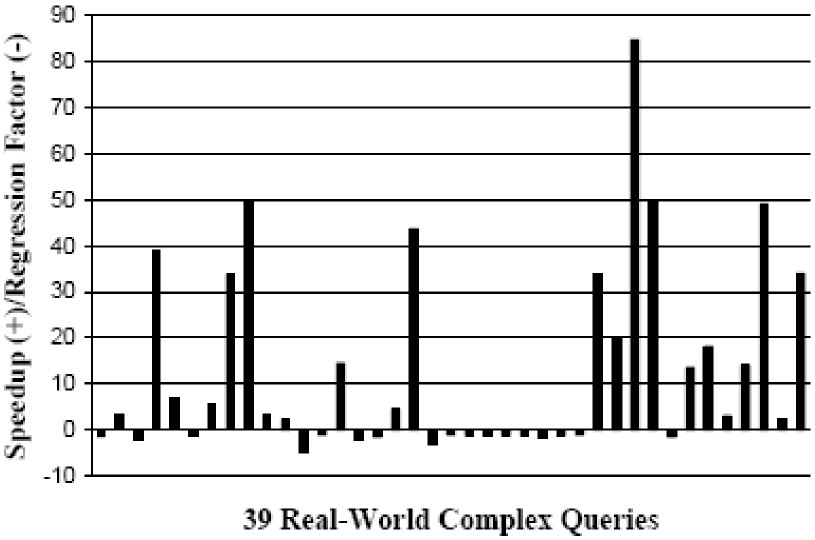



**Figure 16:** Speedup and Regression of each Query

# POP in (in)action (contd.)

- Re-optimization may result in the choice of worse plan due to:
  - Two estimation errors canceling out each other
  - Re-using intermediate results

# Conclusions

- POP gives us a robust mechanism for re-optimization through inserting of CHECK (in its various flavors)
- Higher opportunity at low risk

# Future work

- Lets decide ☺

- Extra Slides

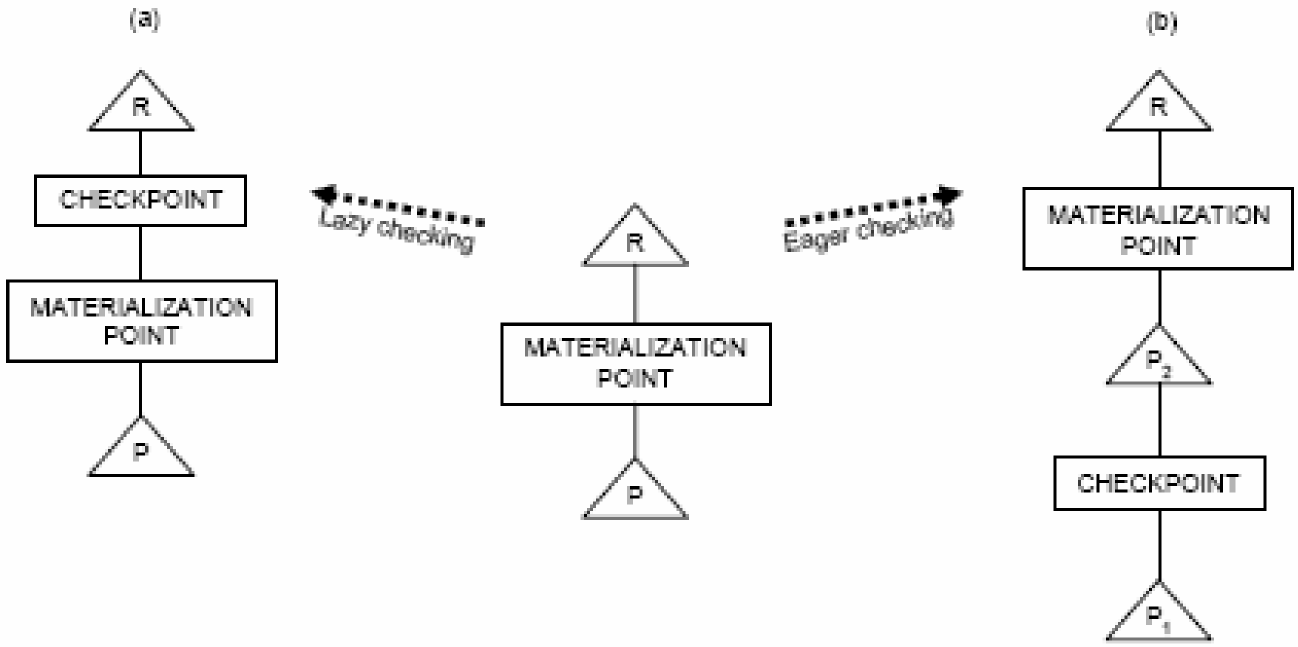



**Figure 7**: Lazy checking (LC) and eager checking without compensation (ECWC)

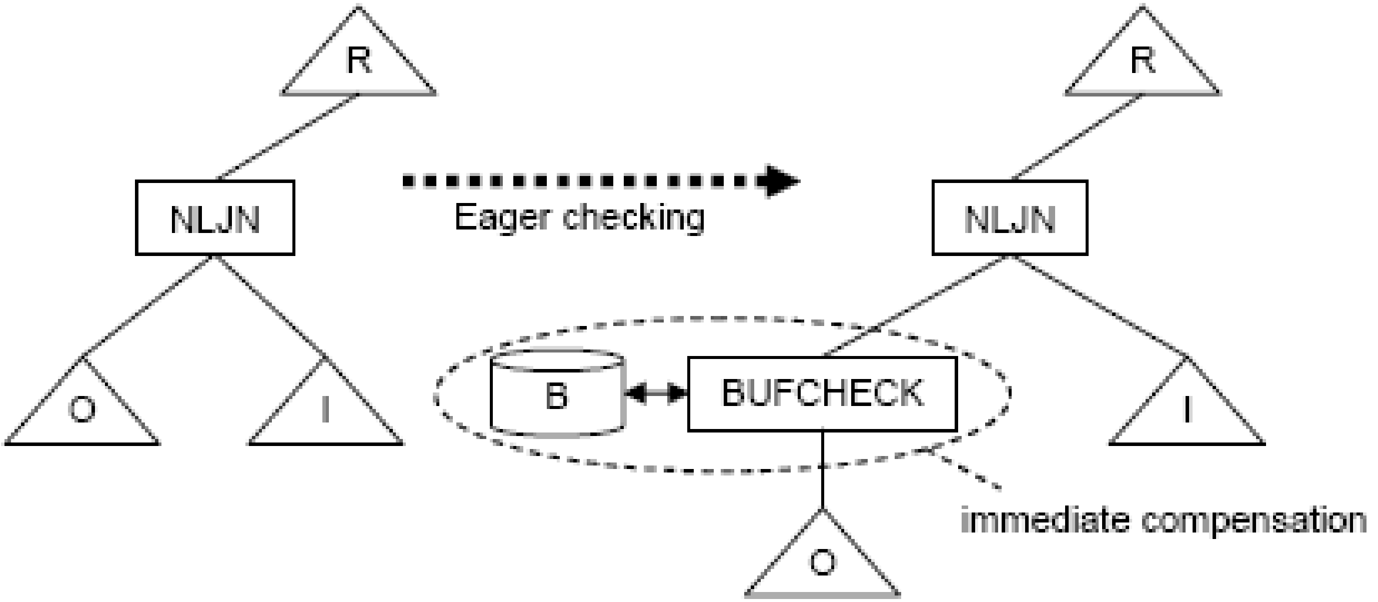



**Figure 8**: Eager checking with Buffering