SICV – Snapshot Isolation with Co-Located Versions

TECHNISCHE UNIVERSITÄT DARMSTADT

TPCTC 2011
3rd International Conference on Performance Evaluation and Benchmarking

Robert Gottstein,
Ilia Petrov,
Alejandro Buchmann
{lastname}@dvs.tu-darmstadt.de

Introduction

- FlashyDB
- MVCC
- Snapshot Isolation

- Co-Located Versions
- Block Pre-Allocation
- Tuple Permutation

- Leverage Flash Memory
- Delay Knee-Point
  - Average Response Times lower
  - Throughput Higher

Structure of the presentation

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Differences: SSD – HDD

2. Snapshot Isolation
   - Algorithm

3. Transaction Management
   - Algorithmic Description
   - Example

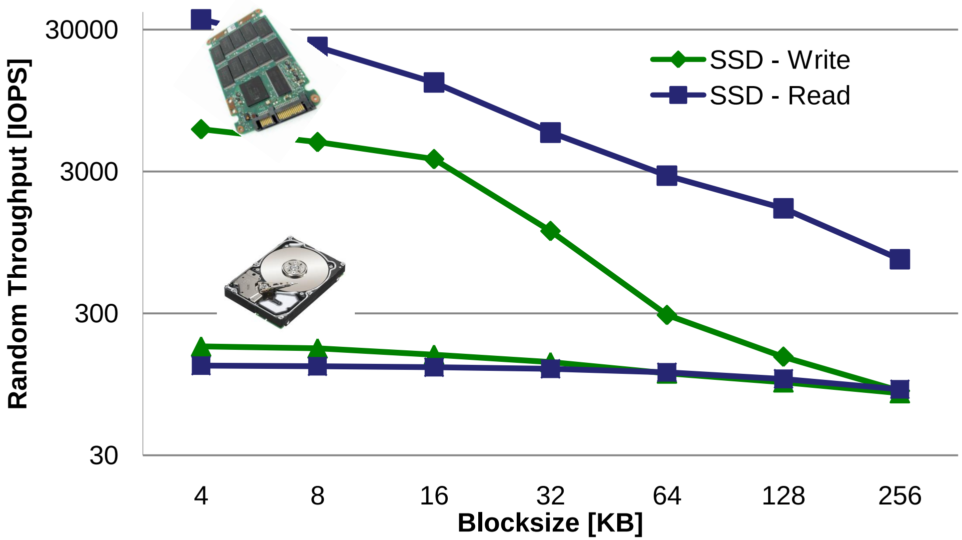
4. Experimental Results

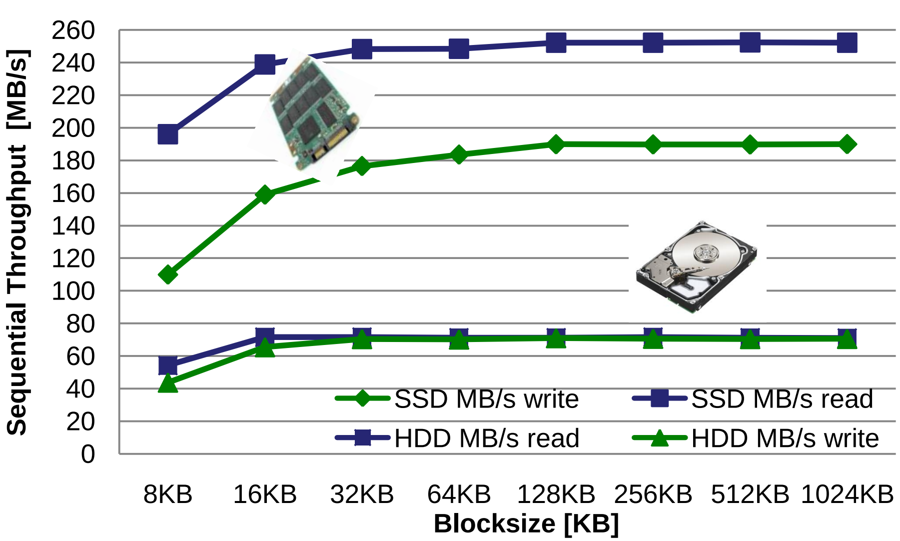
5. Summary

1. Differences: SSD – HDD

TECHNISCHE UNIVERSITÄT DARMSTADT

Flash Storage vs. Magnetic Storage Performance





- **HDD**: ***symmetric***; ***high Latency***; big block; rotational moving parts
- **SSD**: ***asymmetric***; ***low Latency***; FTL; ***No InPlace Updates***; small block; access patterns; ***Intrinsic Parallelism***; IOPS/$ vs. GB/$…

Impact on algorithmic and architectural DBMS assumptions?

Flash Storage vs. Magnetic Storage Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- Algorithms for ***Transactional Management*** are ***build on HDD properties***
  - ***Suitable*** for SSD but ***not optimal*** (HDD: "Rand. Reads as fast as Rand. Writes")

- Multi Version Concurrency Control (MVCC)

[1] Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., and O'Neil, P. 1995. A critique of ANSI SQL isolation levels. In *Proc. The ACM SIGMOD'05 (San Jose, California, United States, May 22 - 25, 1995)*

  - Snapshot Isolation [1] (SI)
  - „In SI a Transaction T_i executes against ist own snapshot (view) of the database" Comprised of committed data (before Start of Ti) and its own data
    - Implemented in ***Oracle, Postgres, SQL Server***…
    - → Reads are never blocked
    - → Leverage SSD read performance

Optimization at which points?

How does SI work?

2. Snapshot Isolation

TECHNISCHE UNIVERSITÄT DARMSTADT

Snapshot Isolation Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- Timestamps on **Transactions** and **Tuples**
  - BOT_i = timestamp(Begin_T_i) (assume = ***TID_i***)
    - $R_i[X]$; $W_i[Y]$; $W_i[X]$; $R_i[Y]$
  - EOT_i = Commit → timestamp(End_T_i)

Tuple X
X.V_m(t_xmin=123, t_xmax=134)

Tuple X
X.V_o(t_xmin=134, t_xmax=null)

- R_i [X] – read last version of X committed before T_i started
  - NO READ locks
  - X.V_i→***t_xmin*** ≤ BOT_i
  - If T_i already modified a data item → sees its own version e.g. X.V_o rather than X.V_i

- W_i [X]–Concurrent transactions, modifying the same data item cannot commit
  - ***First-Committer-Wins-Rule*** (compare writesets) or
  - ***First-Updater-Wins-Rule*** (X-Locks)
  - Update a tuple → create a new Version and invalidate the old version (***t_xmax***)

Snapshot Isolation
Co-Located Versions

TECHNISCHE UNIVERSITÄT DARMSTADT

- ***Extend*** SI's ***transaction management*** to ***create*** a ***tuple permutation*** that:
  - better fits the properties of the SSD
  - reduces random writes that are the result of the concurrent execution

→ Extension of the transaction management to redistribute tuples through a ***pre-allocation of buffer pages*** (blocks) ***per transaction*** (permutation)

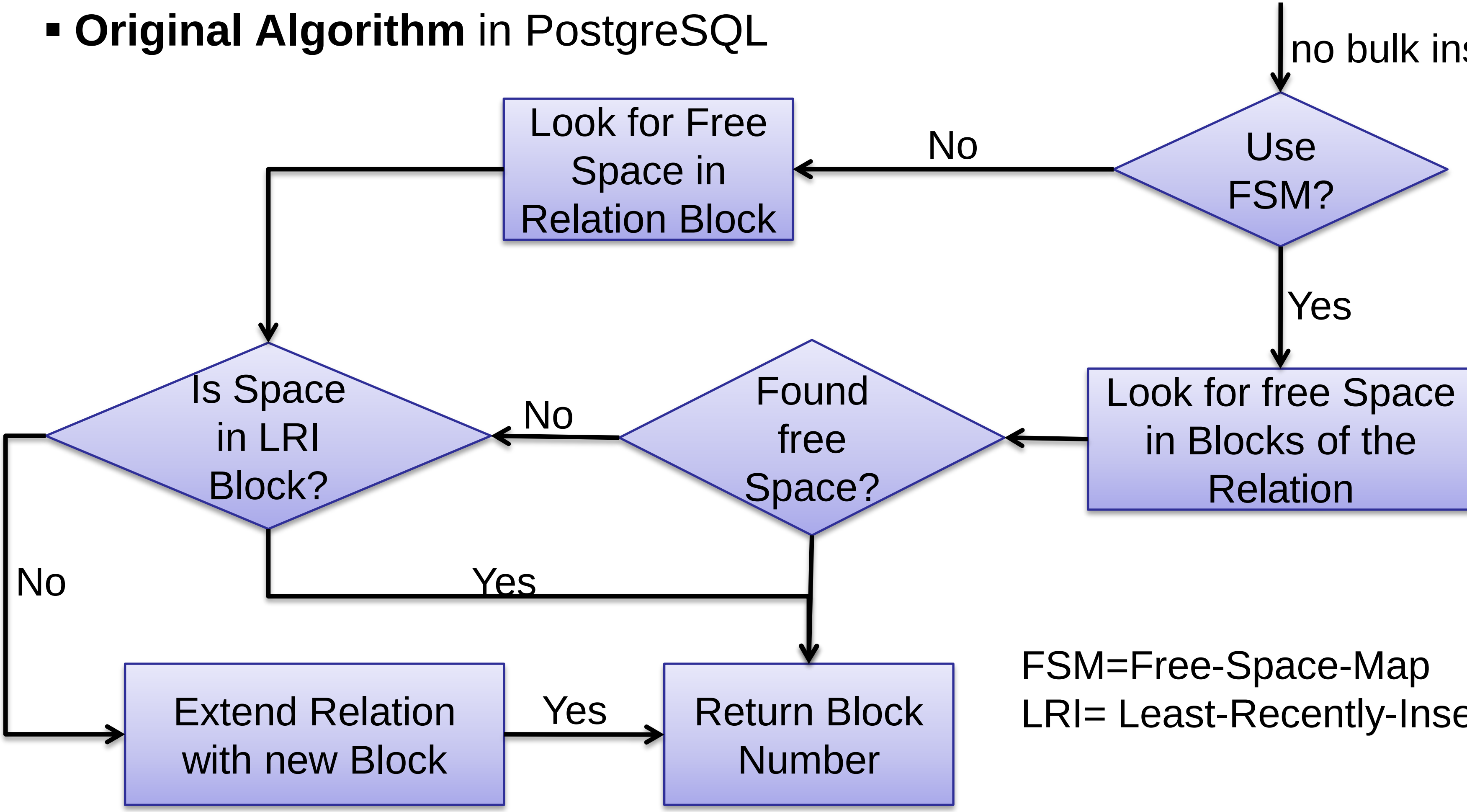
  - Avoid unnecessary random writes which are based on the concurrent execution of multiple transactions without restricting concurrency

Multi Transaction Processing?

3. Transaction Management

Transaction Management
Snapshot Isolation

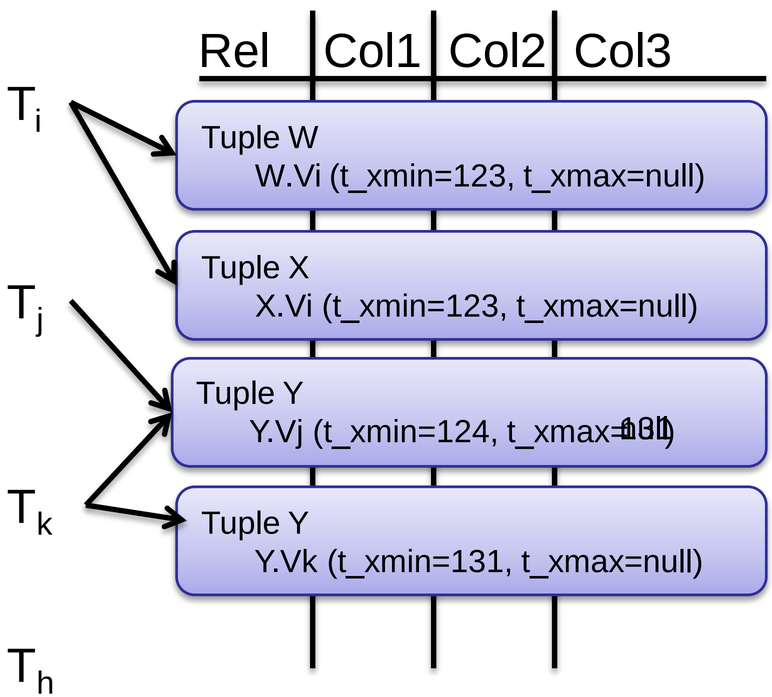
- **Original Algorithm** in PostgreSQL



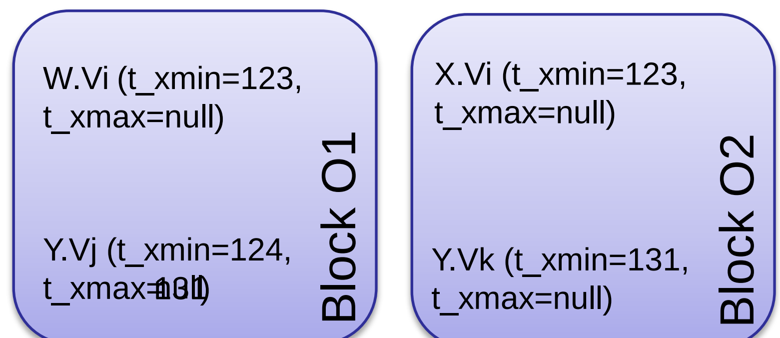
FSM=Free-Space-Map
LRI= Least-Recently-Inserted

Transaction Management
Example Snapshot Isolation

Start(T_i),Start(T_j),
W_i[W],W_j[Y],Commit[T_j],W_i[X],Commit[T_i],Start[T_h],Start[T_k],R_k[W],R_k[X],R_k[Y],W_k[Y],Commit[T_k],R_h[Y],Commit[R_h]



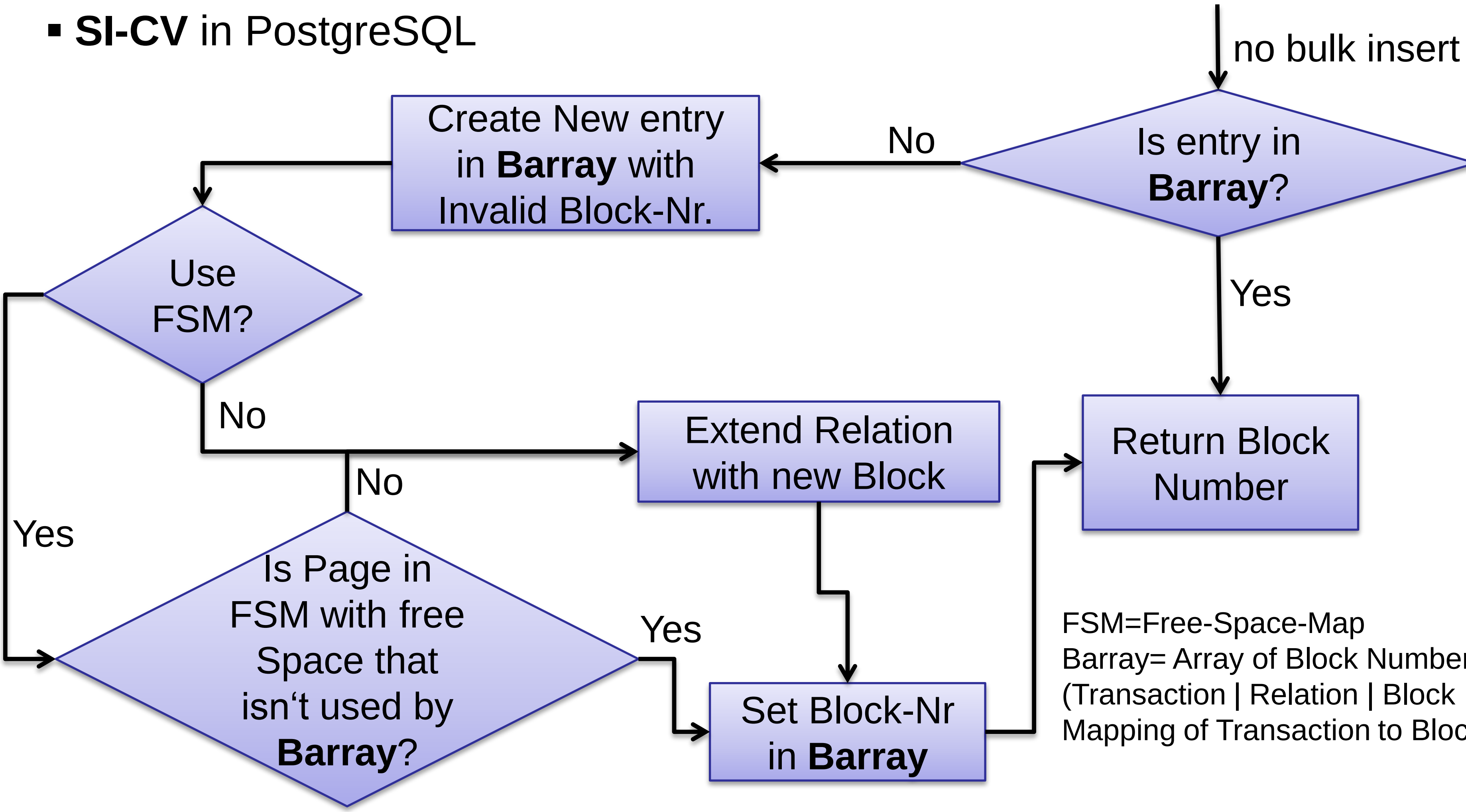
Transaction	TID	Query
T_i	123	INSERT INTO Rel (col1, col2, col3) VALUES (4, Lufthansa, London), (5, Lufthansa, Seattle);
T_j	124	INSERT INTO Rel (col1, col2, col3) VALUES(6, Lufthansa, Frankfurt);
T_h	129	SELECT * FROM Rel WHERE col3=Frankfurt;
T_k	131	UPDATE col2=Condor WHERE col3=Frankfurt;



Write Count Requests	**BlockO1**	**BlockO2**
	1 3	1 2

Transaction Management
Snapshot Isolation with Co-Located Versions

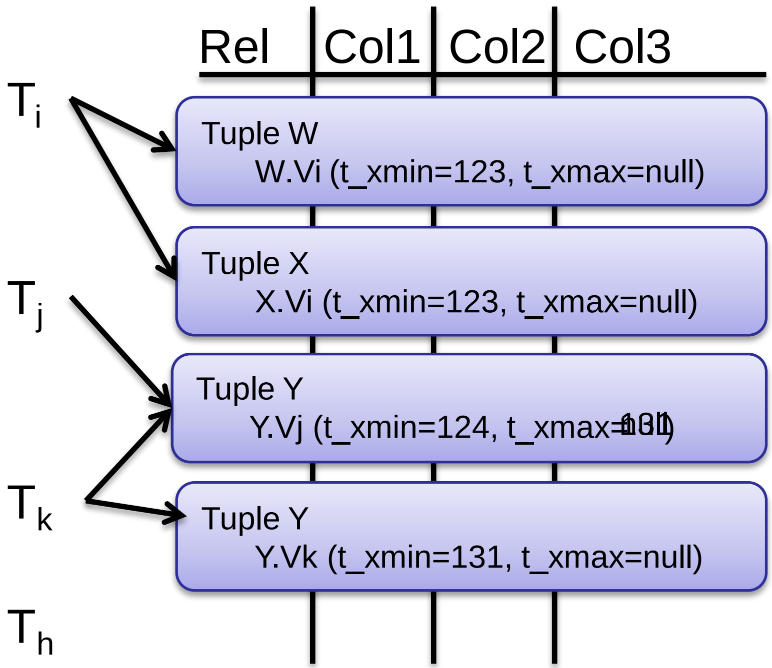
- **SI-CV** in PostgreSQL



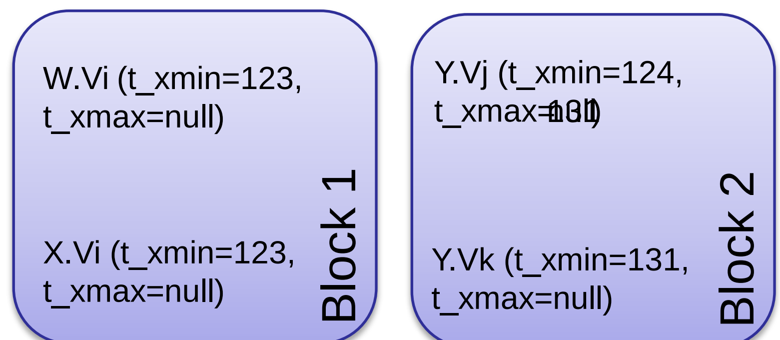
FSM=Free-Space-Map
Barray= Array of Block Numbers
(Transaction | Relation | Block Nr)
Mapping of Transaction to Block

Transaction Management
Example SI with Co-Located Versions

Start(T_i),Start(T_j),
W_i[W],W_j[Y],Commit[T_j],W_i[X],Commit[T_i],Start[T_h],Start[T_k],R_k[W],R_k[X],R_k[Y],W_k[Y],Commit[T_k],R_h[Y],Commit[R_h]



Transaction	TID	Query
T_i	123	INSERT INTO Rel (col1, col2, col3) VALUES (4, Lufthansa, London), (5, Lufthansa, Seattle);
T_j	124	INSERT INTO Rel (col1, col2, col3) VALUES(6, Lufthansa, Frankfurt);
T_h	129	SELECT * FROM Rel WHERE col3=Frankfurt;
T_k	131	UPDATE col2=Condor WHERE col3=Frankfurt;



Transaction	Relation - Block Nr.
T_k 123 / 131	Rel - Block 1 / 2
T_j 124	Rel - Block 2

Write Count Requests	Block 1	Block 2
	1	1 / 2

4. Experimental Results

System Setup

TECHNISCHE UNIVERSITÄT DARMSTADT

- PostgreSQL 8.4.2 on Linux Server, Ubuntu 64bit
- Intel Core 2 Duo 3GHz with 512MB Ram
- Intel ***X25-E***/64GB SSD and Hitachi HDS72161 7200RPM SATA2 HDD
- On Disk Write Cache enabled
- IO Scheduling noop for SSD; deadline for HDD; No Swapping

- DBT2 ***TPC-C*** Benchmark
  - Nominal DB Size ~ 31 GB after data generation and import
  - 20 DB Connections and 20 Terminals per Warehouse
  - increasing amount of Warehouses
    - Intention: **Increasing Concurrency with each run**
  - 2 hour duration for each test

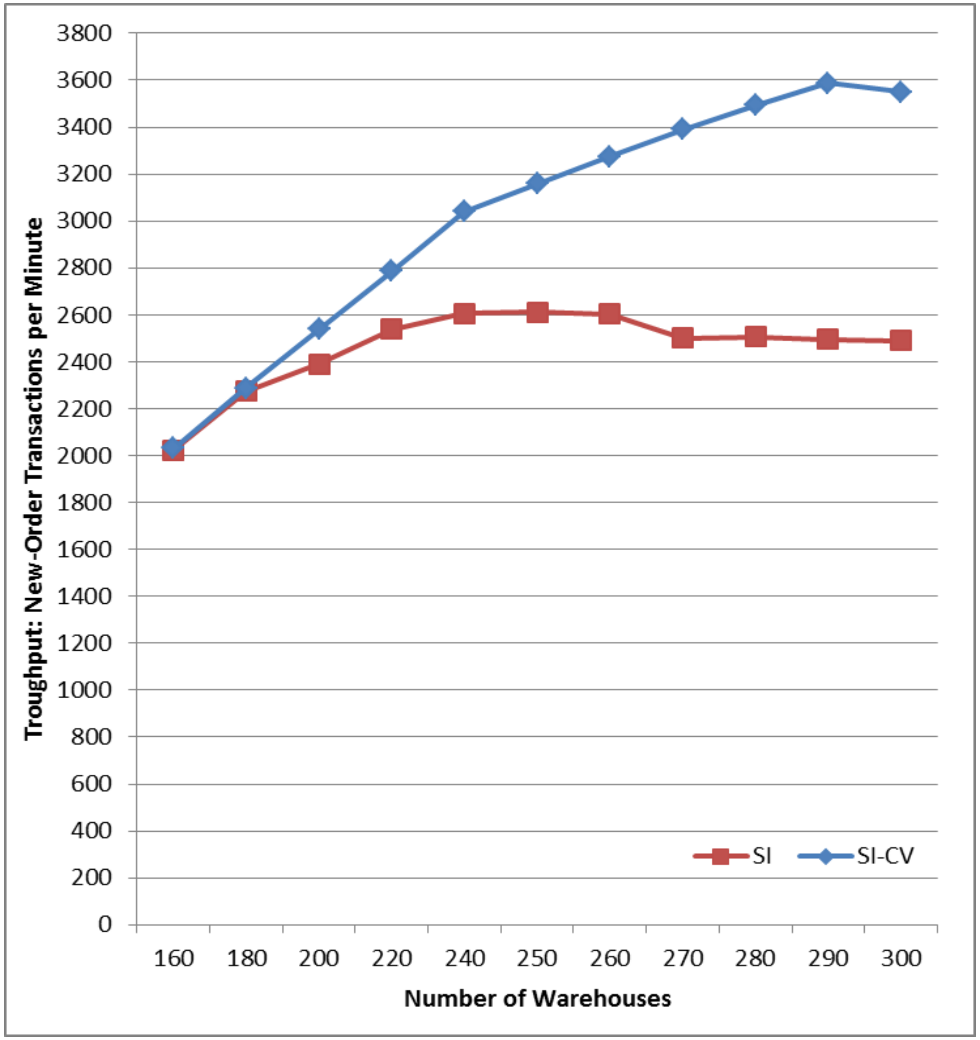
NOTPMs on SSD – SI vs. SI-CV

TECHNISCHE UNIVERSITÄT DARMSTADT

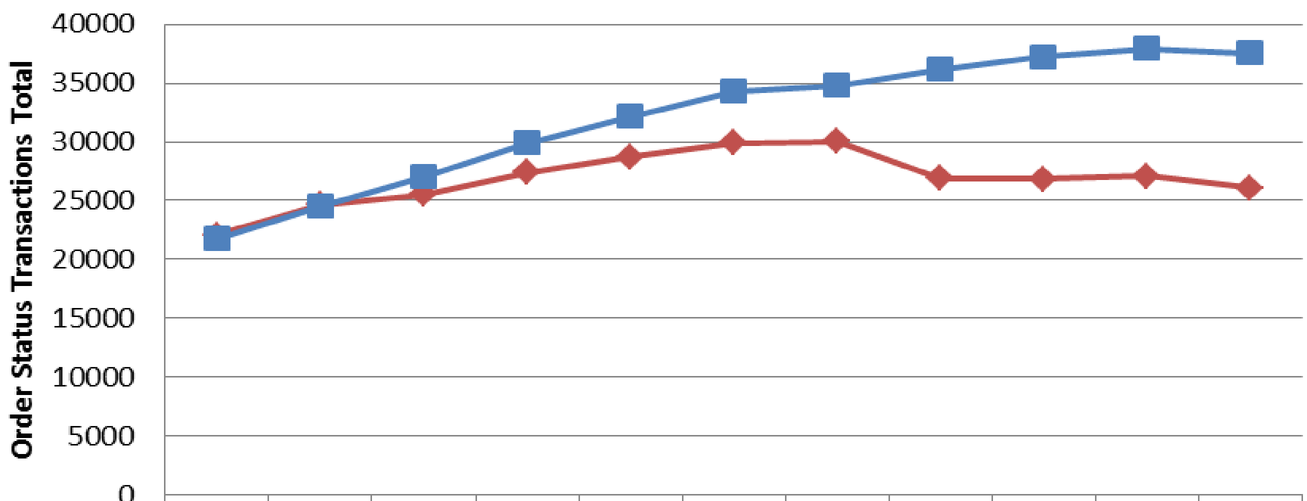
- Each Point = Average NOTPMs
- Range [160, 300] Warehouses
  - Increase transactional load after each run
  - more Transactions → larger effect of collocation/ preallocation
- Equal up to 180 Warehouses
  - Deterioration in Throughput above 240 Warehouses on SI
- Collocation saves random writes

SI-CV performs better under heavy loads.

Performance increases with higher amount of transactions.



Order Status Relation on SSD



	160	180	200	220	240	250	260	270	280	290	300
SI	22085	24678	25545	27399	28710	29900	30029	26908	26859	27115	26084
SICV	21763	24548	27012	29901	32149	34285	34811	36139	37246	37906	37540

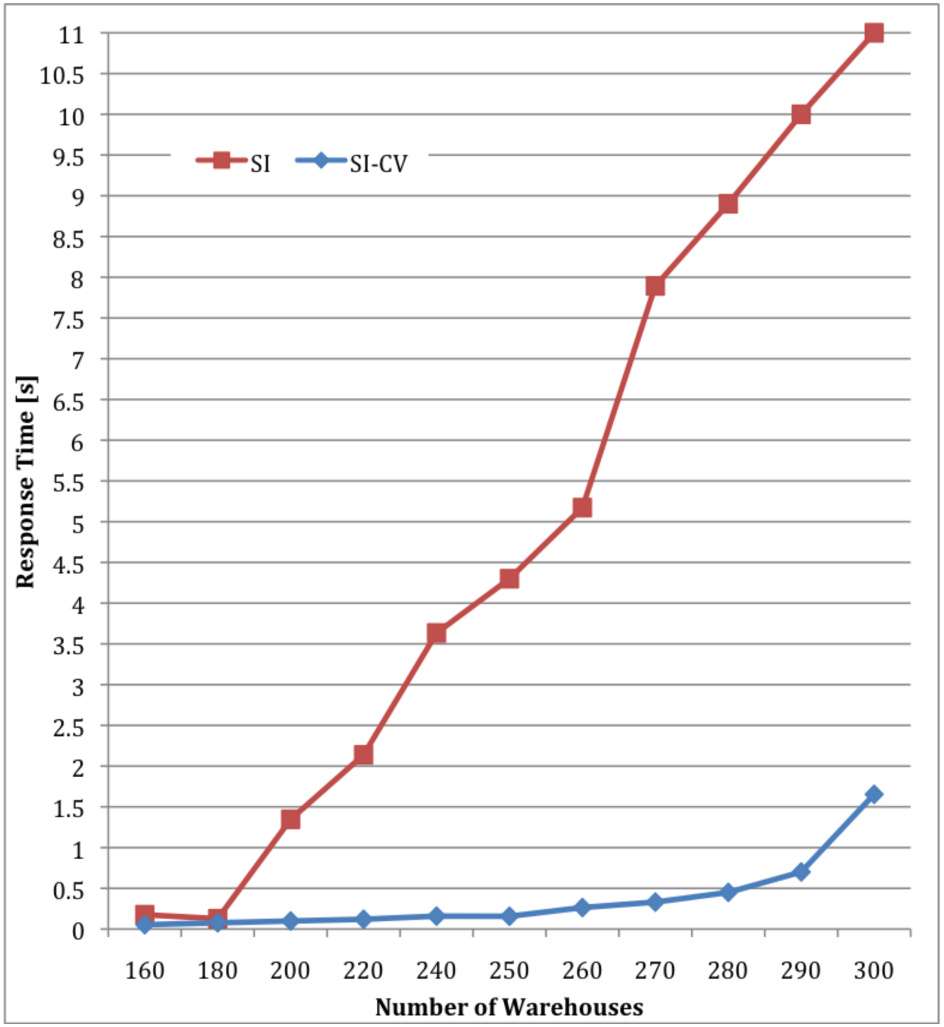
- Ordinate: Amount of order status transactions (absolute)
- Leverages SSD random read performance

Read Performance of SI-CV equally good or better.

Average Response Time on SSD

- Under-committed System
  - Enough free resources: SI & SI-CV perform equally well
    - ≤ 180 Warehouses
- Increase of Load bringt SI into thrashing
  - ≥ 230 Warehouses

- SI-CV able to maintain avg. resp. times <5sec for a wider band of warehouses
  - above the knee of SI



Resp. times in over-committed system significantly lower

Space Consumption

- Hypothesis: Preallocation uses/ needs more Space
  - Blocks may not be ***filled*** optimally

- ***Normalized*** „per Warehouse " ***Values***
  - Reason: NOTPM count of SI-CV is higher when using the same amount of warehouses, therefore space consumption per Warehouse alone is not meaningful

- Used the value that shows the highest difference at 280 Warehouses
  - Maximum increase in space utilization after 2 hours 0.0016% per Warehouse

  → Insertion of Bulk Loads not affected

SI-CV almost as space efficient as SI.

5. Summary

- SI-CV performs better under heavy load conditions, when the system is I/O-Bound → Up to 30%
- Relative performance of SI-CV increases with higher number of transactions
- Response time in over-committed system significantly lower than that of SI, therefore „shifting the knee“
- Pre-Allocation strategy per Transaction almost as space efficient as SI
  - Additional space utilization marginal → justified performance advantage
- Read performance of SI-CV in comparison to SI equally good or better

Thank You…

TECHNISCHE UNIVERSITÄT DARMSTADT

www.dvs.tu-darmstadt.de/research/flashydb