PASS SUMMIT2019

Deep Dive Into Blocking and Deadlocks Troubleshooting

Dmitri Korotkevitch
Director, Database Services
chewy.com

Please silence cell phones

Explore

everything PASS has to offer

Free Online Resources

Newsletters

PASS.org

24HOURS OF PASS

Free online webinar events

PASS LOCAL GROUPS

Local user groups around the world

PASS SQLSATURDAY

Free 1-day local training events

PASS MARATHON

Online special interest user groups

PASS VIRTUAL GROUPS

Business analytics training

PASS VOLUNTEERS

Get involved

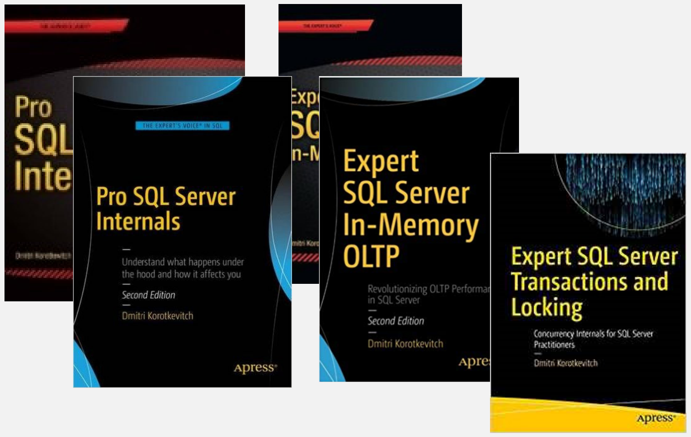
Dmitri Korotkevitch

Director, Database Services
chewy.com

/dmitri-korotkevitch-0b79805

@aboutsqlserver

dmitri@aboutsqlserver.com


Pro SQL Server Internals
Understand what happens under the hood and how it affects you
Second Edition
Dmitri Korotkevitch
Apress
Expert SQL Server In-Memory OLTP
Revolutionizing OLTP Performance in SQL Server
Second Edition
Dmitri Korotkevitch
Expert SQL Server Transactions and Locking
Concurrency Internals for SQL Server Practitioners
Dmitri Korotkevitch
Apress

Agenda

Overview of SQL Server Concurrency Model

Troubleshooting Techniques

Slides and Demos: https://aboutsqlserver.com/presentations

Disclaimers

Internal implementation is vaguely documented

- Documentation focuses on logical consistency rather than internal implementation

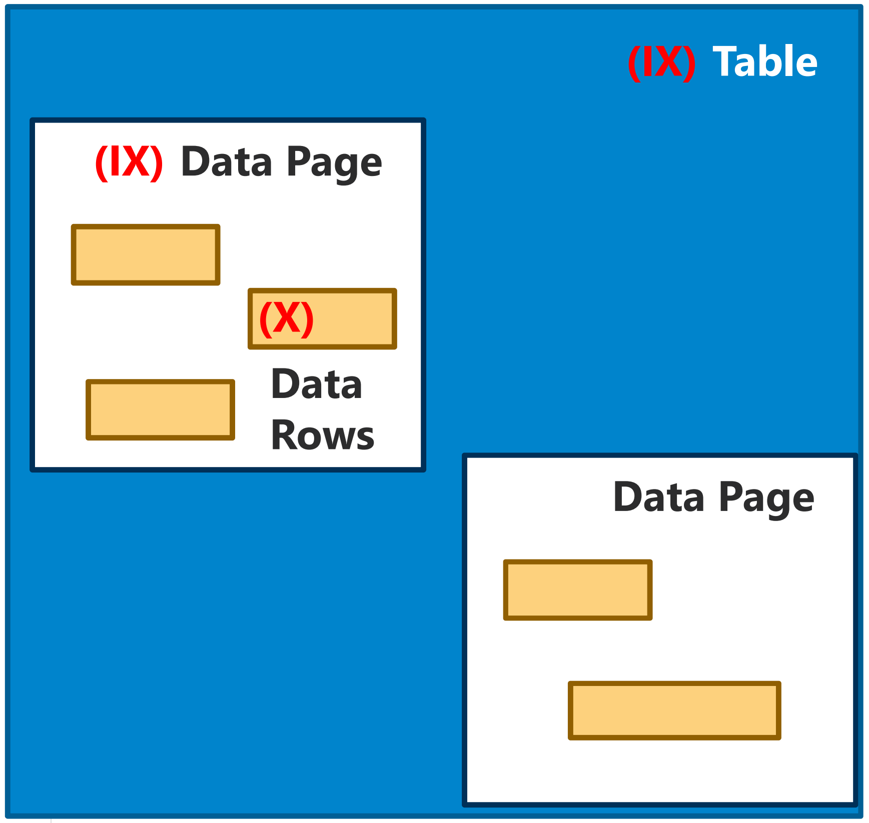
Locking behavior may slightly vary on case-by-case basis

- Lock compatibility rules always apply!

Session focuses on locking in disk-based B-Tree tables

- Columnstore indexes behave *somewhat* similar
- In-Memory OLTP behaves very differently

Lock Types



Full Locks:

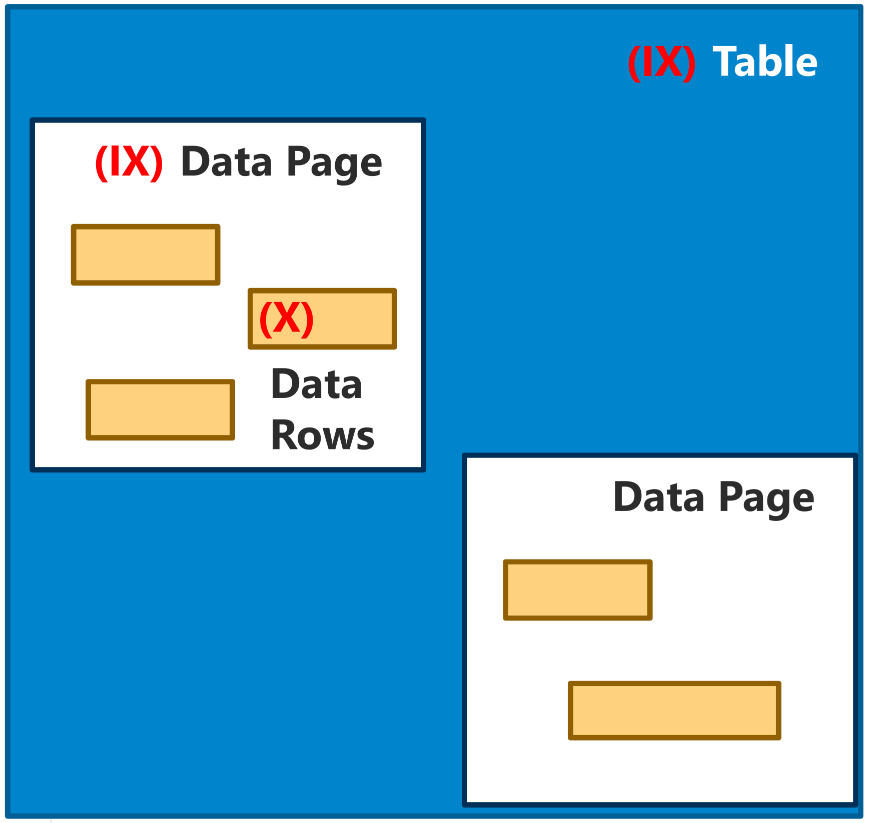
- Exclusive (X) – data had been modified
  - Always acquired by writers (INS, UPD, DEL)
  - **Always** held until the end of transaction
- Update (U) – validating if data needs to be modified
  - Always acquired except in SNAPSHOT
  - Converted to (X) or released
- Shared (S) – reading the data (SELECT)
  - Sometimes acquired (more later)

Intent Locks:

- Locks on the child objects: (IS), (IX), (IU)

UPDATE T WHERE ID = ?

Lock Granularity

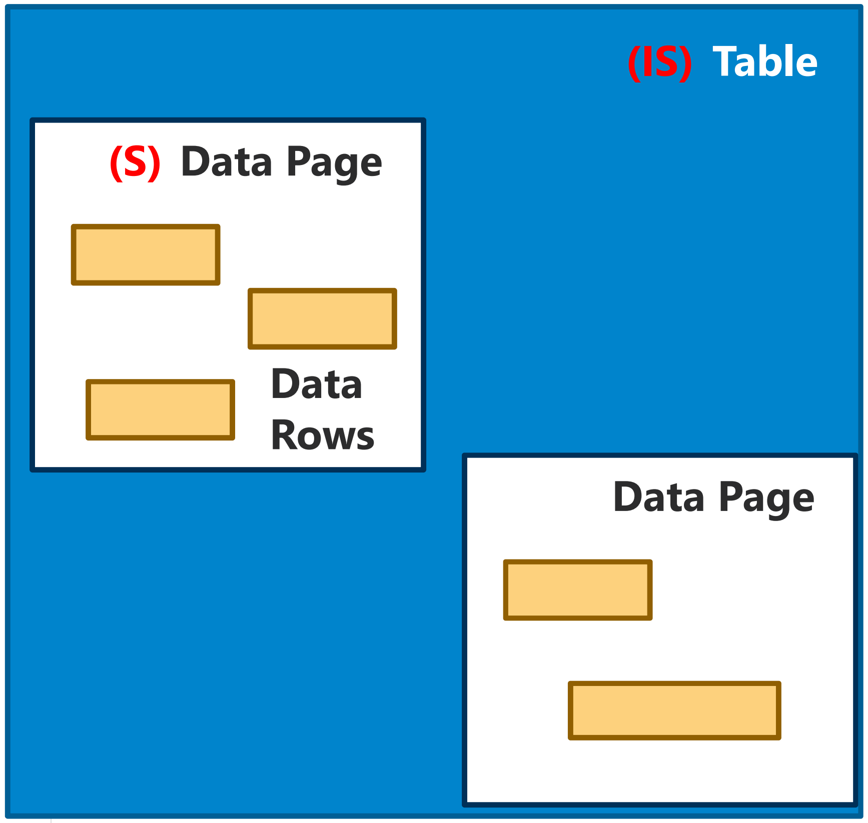


Usually row-level locking:

- Full lock on the data row
- Intent locks on the data page and table

UPDATE T WHERE ID = ?

Lock Granularity



Usually row-level locking:

- Full lock on the data row
- Intent locks on the data page and table

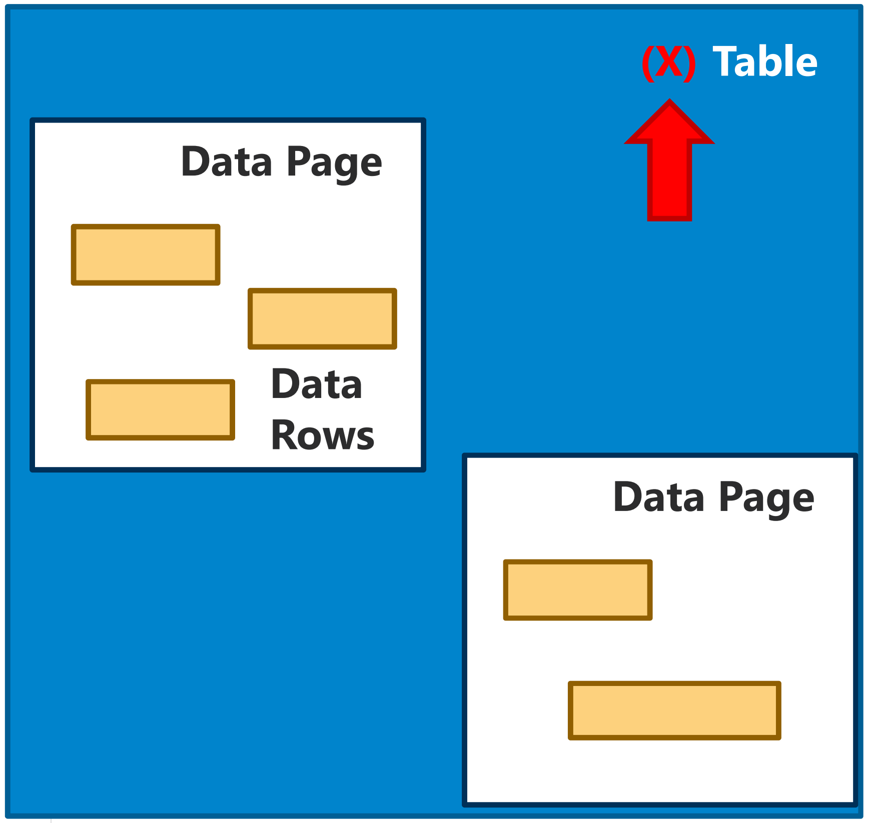
UPDATE T WHERE ID = ?

Sometimes page-level locking:

- Full lock on the data page
- Intent locks on the table
- Usually when all data on the page needs to be scanned

SELECT * FROM T

Lock Granularity



Usually row-level locking:

- Full lock on the data row
- Intent locks on the data page and table

UPDATE T WHERE ID = ?

Sometimes page-level locking:

- Full lock on the data page
- Intent locks on the table
- Usually when all data on the page needs to be scanned

SELECT * FROM T

Sometimes table-level locking:

- Full lock on the table
- Hints, SERIALIZABLE, Lock Escalation, etc (more later)

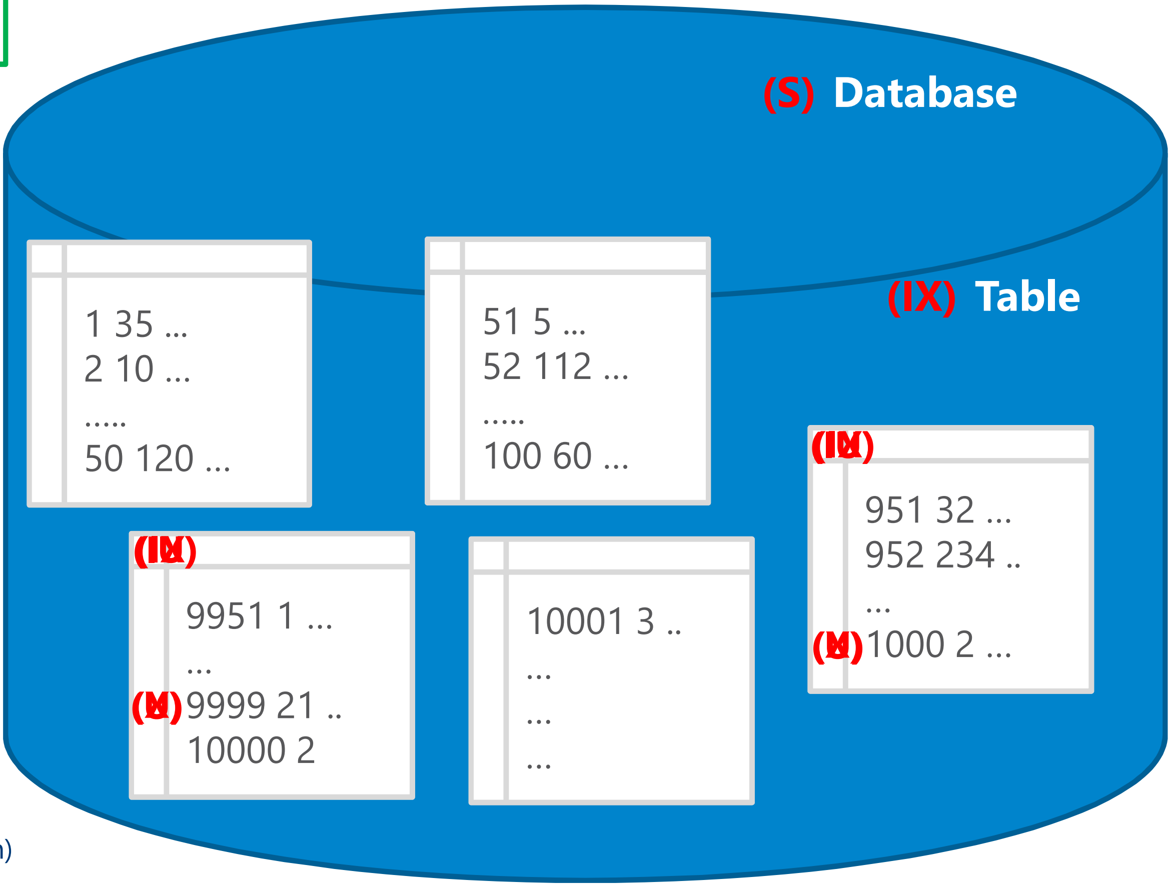
SELECT * FROM T WITH (TABLOCKX)

How It Works..

```
create table dbo.Orders
(
    OrderId int not null,
    CustomerId int not null,
    /* … */
    constraint PK_Orders
    primary key clustered(OrderId)
);
```

```
begin tran
    update dbo.Orders
    set Processed = 1
    where OrderId in (1000, 9999)
commit
```

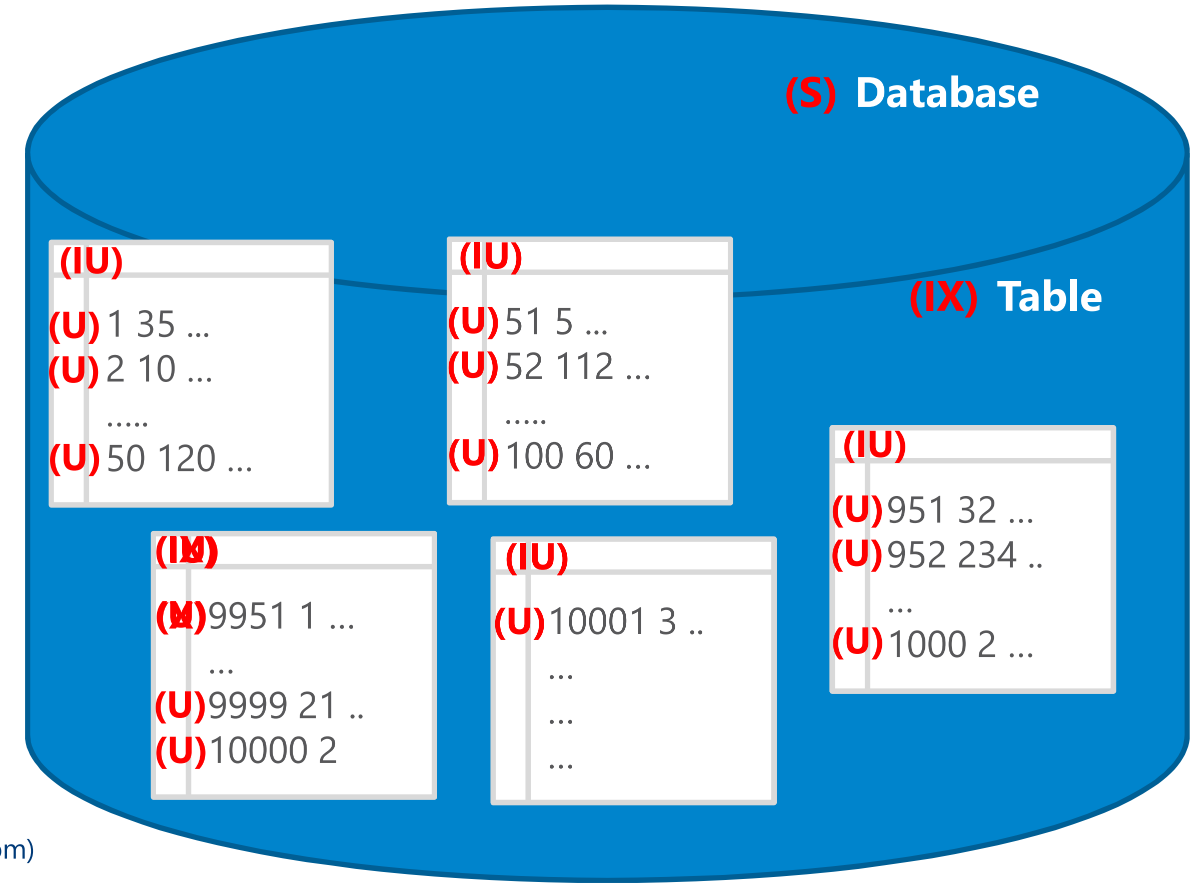
How It Works...



How It Works…

```
begin tran
    update dbo.Orders
    set Approved = 1
    where CustomerId = 1
```

```
commit
```



SQL Server does not know if row needs to be updated until it reads the row. (U) lock is acquired to evaluate *CustomerID = 1* predicate.

(X), (U) and (S) Lock Compatibility

	(S)	(U)	(IU) / (IX)	(X)
(S)	🟩	🟩	☹	☹
(U)	🟩	☹	☹	☹
(IU) / (IX)	☹	☹		☹
(X)	☹	☹	☹	☹

Exclusive (X) locks are held till the end of transaction

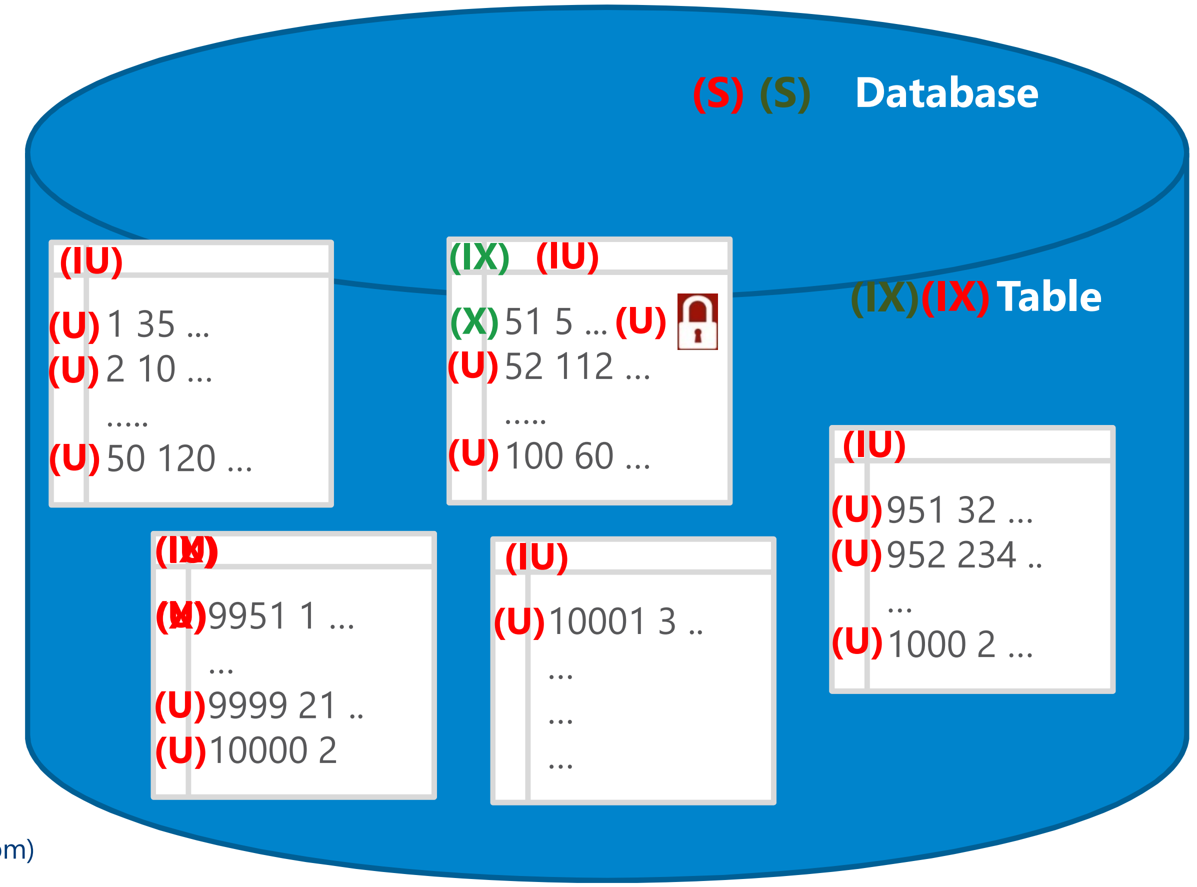
How It Works…

```
begin tran
    update dbo.Orders
    set Approved = 1
    where OrderId = 51
```

```
commit
```

```
begin tran
    update dbo.Orders
    set Approved = 1
    where CustomerId = 1
```

```
commit
```



Conversion and Range Locks

Conversion Locks:

- Combination of full and intent locks on the page- or table-level
- **(SIX) = (S) + (IX); (UIX) = (U) + (IX); (SIU) = (S) + (IU)**
- Compatibility rules apply to both lock types

Range Locks (**LCK_M_Range*)**

- Protect interval of the rows in SERIALIZABLE isolation level

Isolation Levels and (S) Locks Behavior

	(S) Locks Behavior	Table Hint
READ UNCOMMITTED	(S) locks are not acquired	NOLOCK
READ COMMITTED	(S) locks are acquired and released immediately	READCOMMITTED
REPEATABLE READ	(S) locks are held until the end of transaction	REPEATABLEREAD
SERIALIZABLE	Range (S) locks are held until the end of transaction	HOLDLOCK
READ COMMITTED SNAPSHOT, SNAPSHOT	(S) locks are not acquired (except for FK checks) (more later)	

Troubleshooting Blocking Issues

Understand “who is blocking whom and why”

Now:

- **sys.dm_tran_locks** – currently lock requests and their statuses
- **sys.dm_os_waiting_tasks** – wait_type (LCK_M_*), blocking_session_id
- **sys.dm_exec_requests** – wait_type (LCK_M_*), blocking_session_id

Later:

- Blocked Process Report

Troubleshooting Blocking Issues

Demo

Blocked Process Report

Can be collected with:

- xEvent and SQL Traces, Event Notifications

Challenges:

- Execution plans and statements may not be available when you troubleshoot
- One blocking condition may generate many reports

Blocking Monitoring Framework:

- Captures and parses blocked process report in real time (Event Notification-based solution)
- Download from my blog: http://aboutsqlserver.com/bmframework

Blocking Monitoring Framework

Demo

Lock Escalation

SQL Server escalates locks to the table/partition level

- After ~5,000 locks per statement per object
- If failed – after ~1,250 new locks per statement per object

It is completely normal unless it is not.. ☺

Pattern: batch operation triggered lock escalation. All other sessions that tried to obtain incompatible intent lock on the object were blocked.

Lock Escalation

Demo

Lock Escalation

Troubleshooting

- *Lock Escalation* events in xEvents and SQL Trace
- *Table Lock Escalations/Sec* performance counter

Disabling Lock Escalation

- SQL Server 2008+: *ALTER TABLE .. SET LOCK_ESCALATION*
- TF 1211 / 1224

Schema Locks

DML statements acquire Schema Stability (Sch-S) locks that prevent alteration of underlying objects

- In some cases (Sch-S) can be replaced with (I*) locks

DDL statements acquire Schema Modification (Sch-M) locks and held them until the end of transaction

- Beware schema comparison tools!

Typical issues

- Partition function alteration especially with the data movement
- Index rebuild

Schema Locks

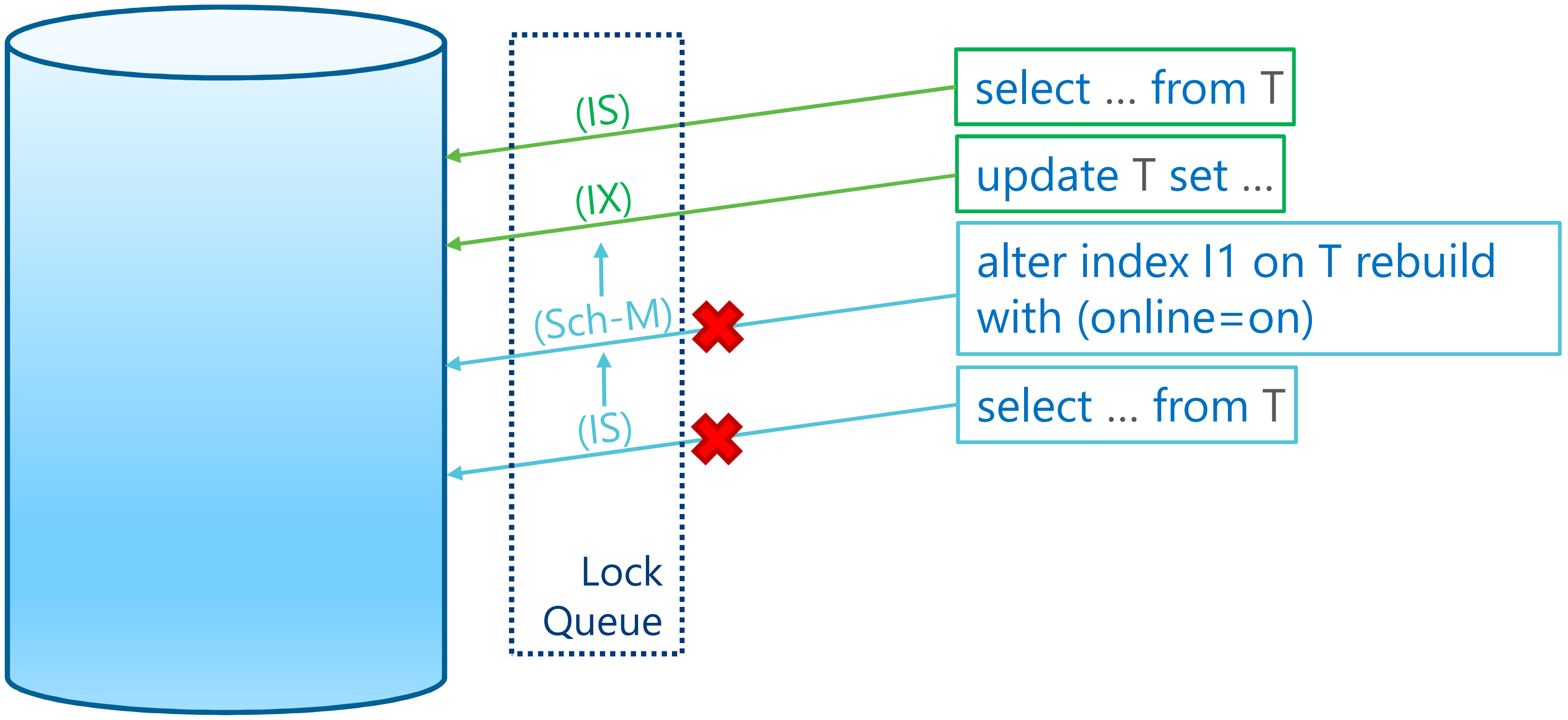
Demo

Locking Queues & Low Priority Locks

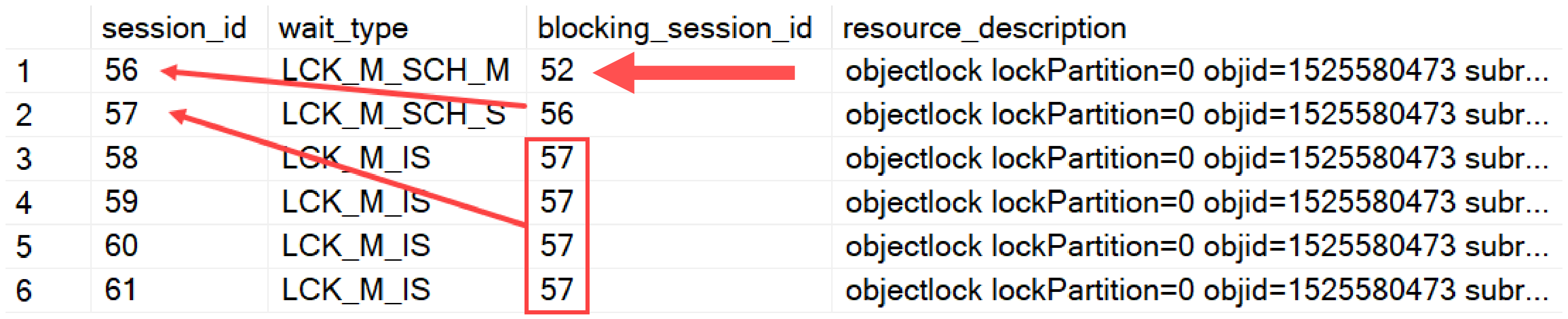
Lock needs to be compatible with all locks in the queue (granted or waited) in order to be granted

SQL Server 2014+ allows to use separate locking queue for online index rebuild and partition switch

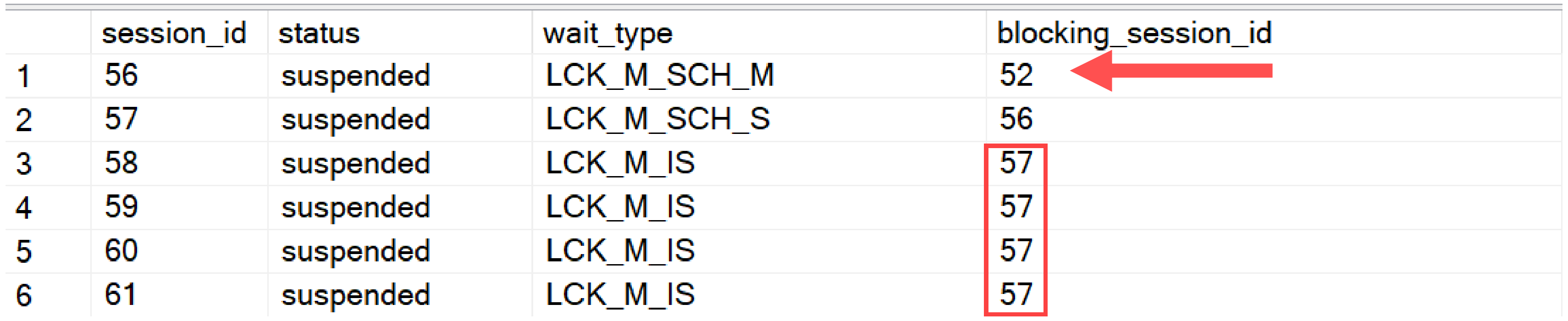
Lock Queue


(IS)
(IX)
(Sch-M)
(IS)
Lock Queue
select ... from T
update T set ...
alter index I1 on T rebuild with (online=on)
select ... from T

Blocking Chains

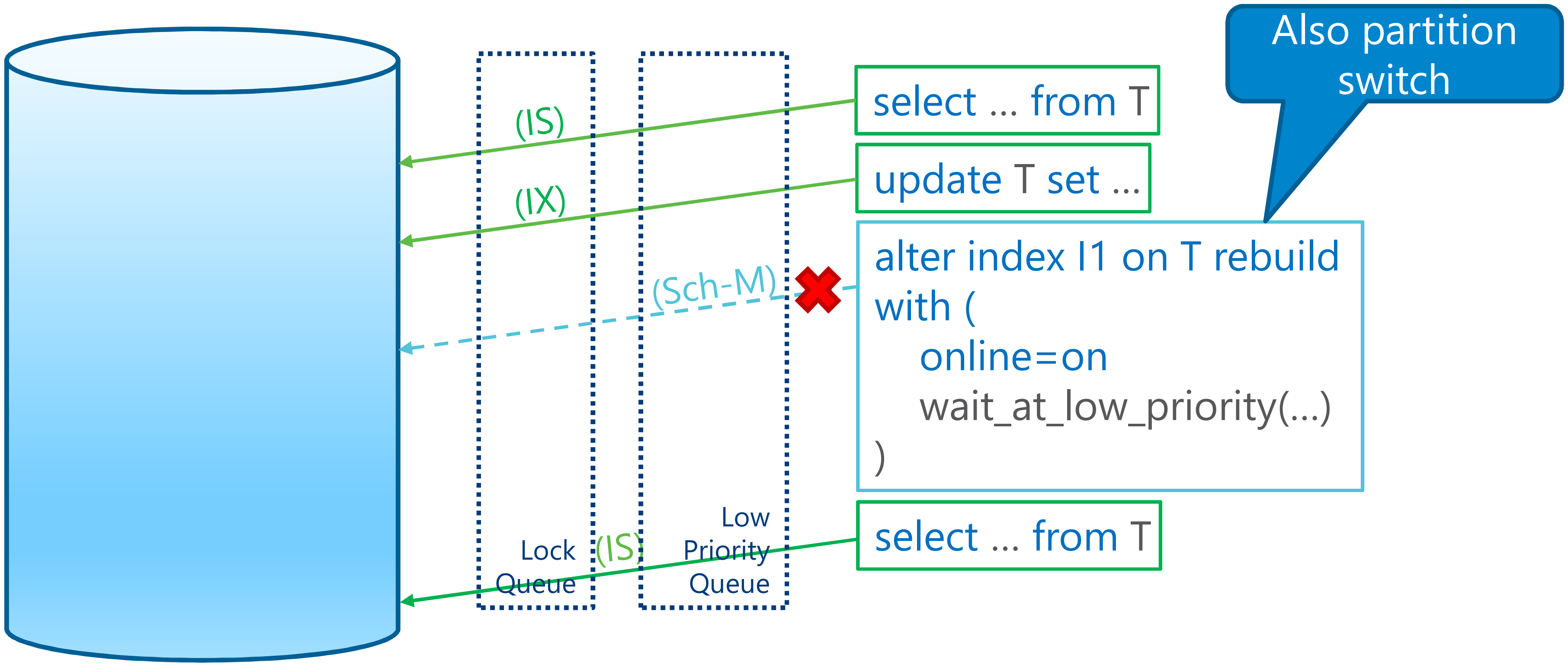


	session_id	wait_type	blocking_session_id	resource_description
1	56	LCK_M_SCH_M	52	objectlock lockPartition=0 objid=1525580473 subr...
2	57	LCK_M_SCH_S	56	objectlock lockPartition=0 objid=1525580473 subr...
3	58	LCK_M_IS	57	objectlock lockPartition=0 objid=1525580473 subr...
4	59	LCK_M_IS	57	objectlock lockPartition=0 objid=1525580473 subr...
5	60	LCK_M_IS	57	objectlock lockPartition=0 objid=1525580473 subr...
6	61	LCK_M_IS	57	objectlock lockPartition=0 objid=1525580473 subr...



	session_id	status	wait_type	blocking_session_id
1	56	suspended	LCK_M_SCH_M	52
2	57	suspended	LCK_M_SCH_S	56
3	58	suspended	LCK_M_IS	57
4	59	suspended	LCK_M_IS	57
5	60	suspended	LCK_M_IS	57
6	61	suspended	LCK_M_IS	57

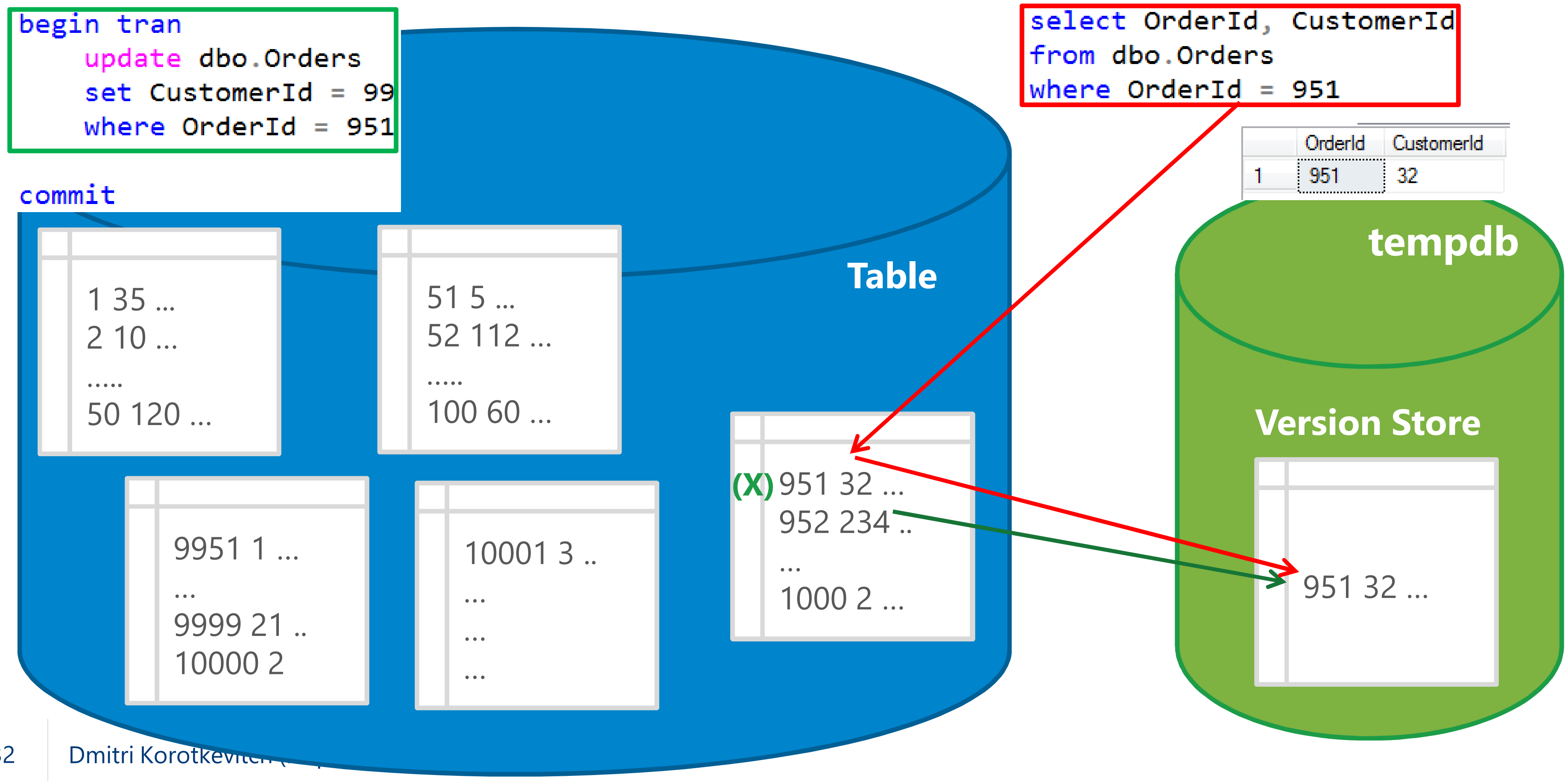
Low Priority Locks (SQL Server 2014+)



Low Priority Locks

Demo

Read Committed Snapshot



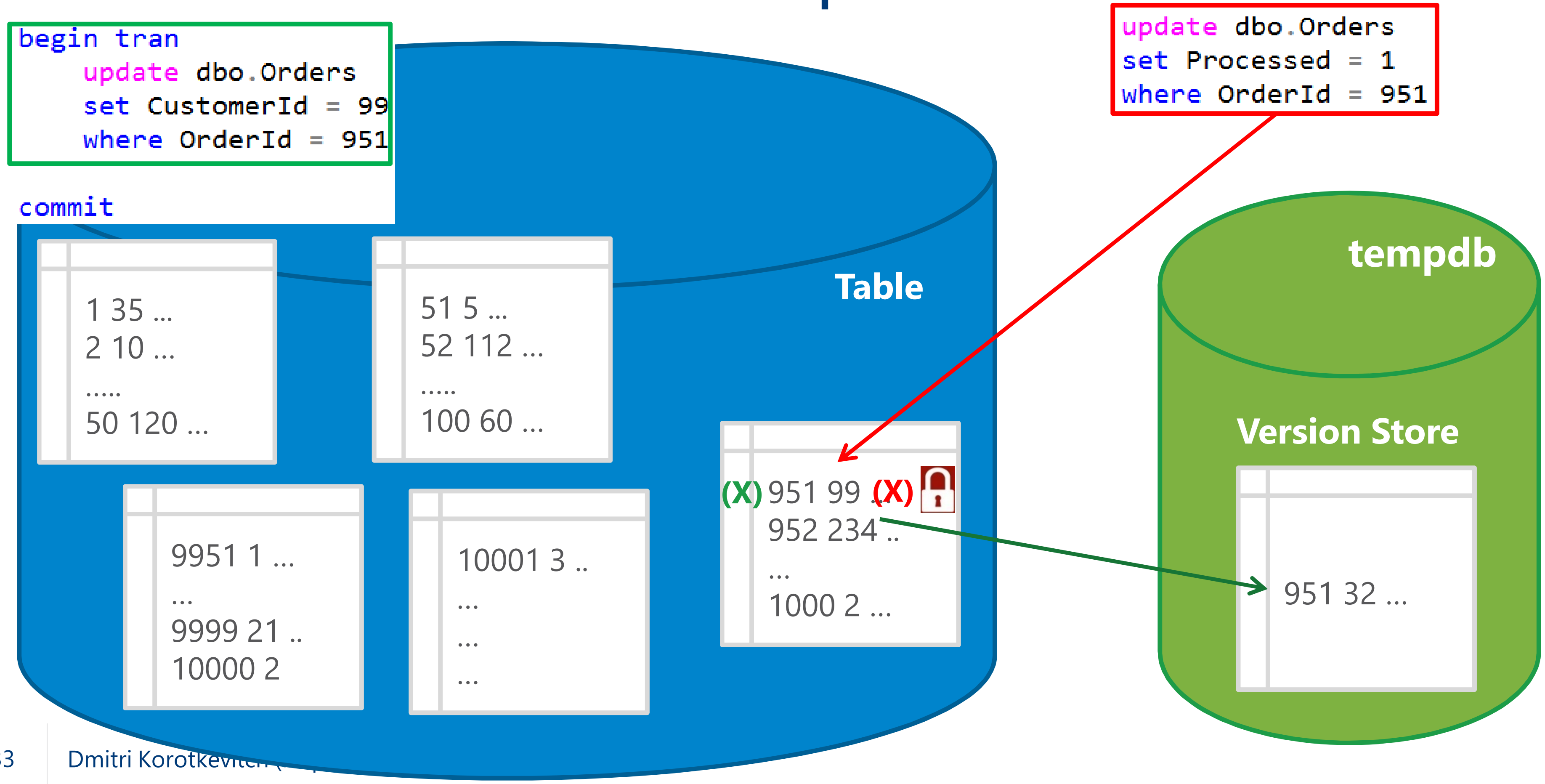
```
begin tran
    update dbo.Orders
    set CustomerId = 99
    where OrderId = 951

commit
```

```
select OrderId, CustomerId
from dbo.Orders
where OrderId = 951
```

	OrderId	CustomerId
1	951	32

Read Committed Snapshot



```
begin tran
    update dbo.Orders
    set CustomerId = 99
    where OrderId = 951

commit
```

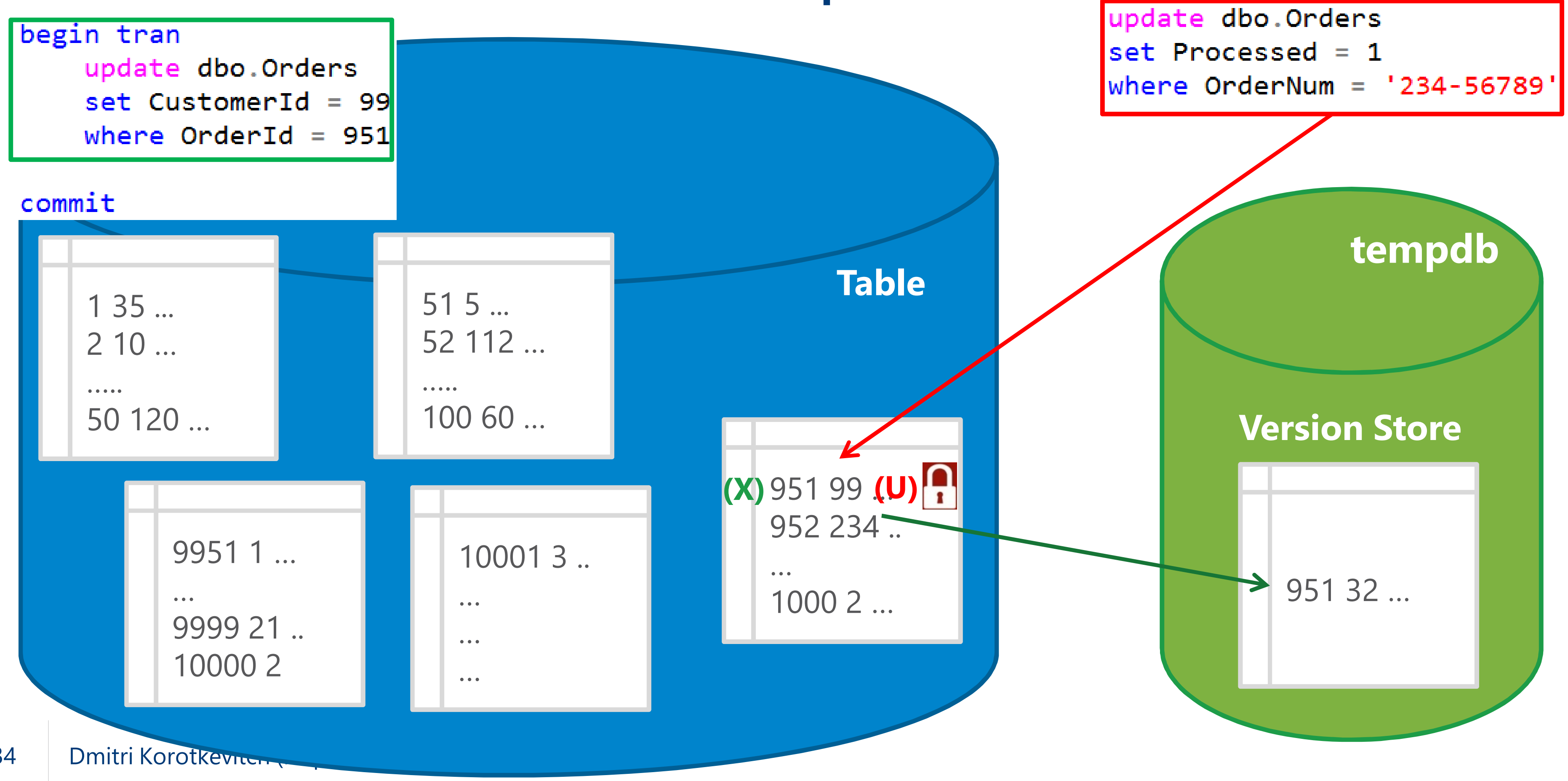
```
update dbo.Orders
set Processed = 1
where OrderId = 951
```

Read Committed Snapshot

```
begin tran
    update dbo.Orders
    set CustomerId = 99
    where OrderId = 951

commit
```

```
update dbo.Orders
set Processed = 1
where OrderNum = '234-56789'
```



Read Committed Snapshot (Statement-Level Consistency)

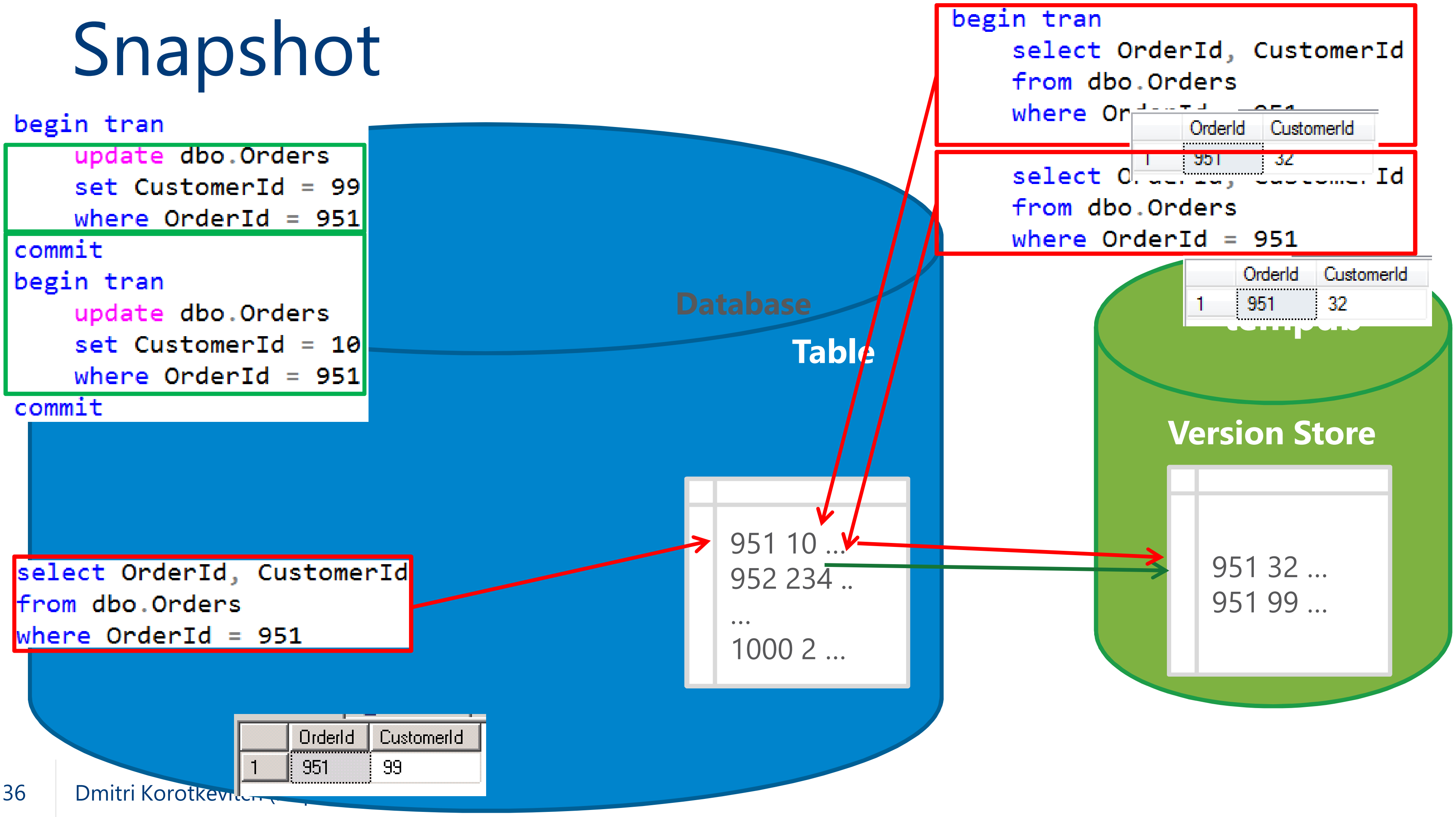
Database option that changes *readers* behavior in READ COMMITTED transaction isolation level

Removes *readers/writers* blocking. Does not remove *writers/writers* blocking

Great emergency technique if system suffers from locking

Lightweight row-versioning is used with readable AG secondaries even if not enabled

Snapshot



```
begin tran
    update dbo.Orders
    set CustomerId = 99
    where OrderId = 951
commit
begin tran
    update dbo.Orders
    set CustomerId = 10
    where OrderId = 951
commit
```

```
select OrderId, CustomerId
from dbo.Orders
where OrderId = 951
```

	OrderId	CustomerId
1	951	99

```
begin tran
    select OrderId, CustomerId
    from dbo.Orders
    where Or
```

	OrderId	CustomerId
1	951	32

```
    select O                    Id
    from dbo.Orders
    where OrderId = 951
```

	OrderId	CustomerId
1	951	32

Snapshot

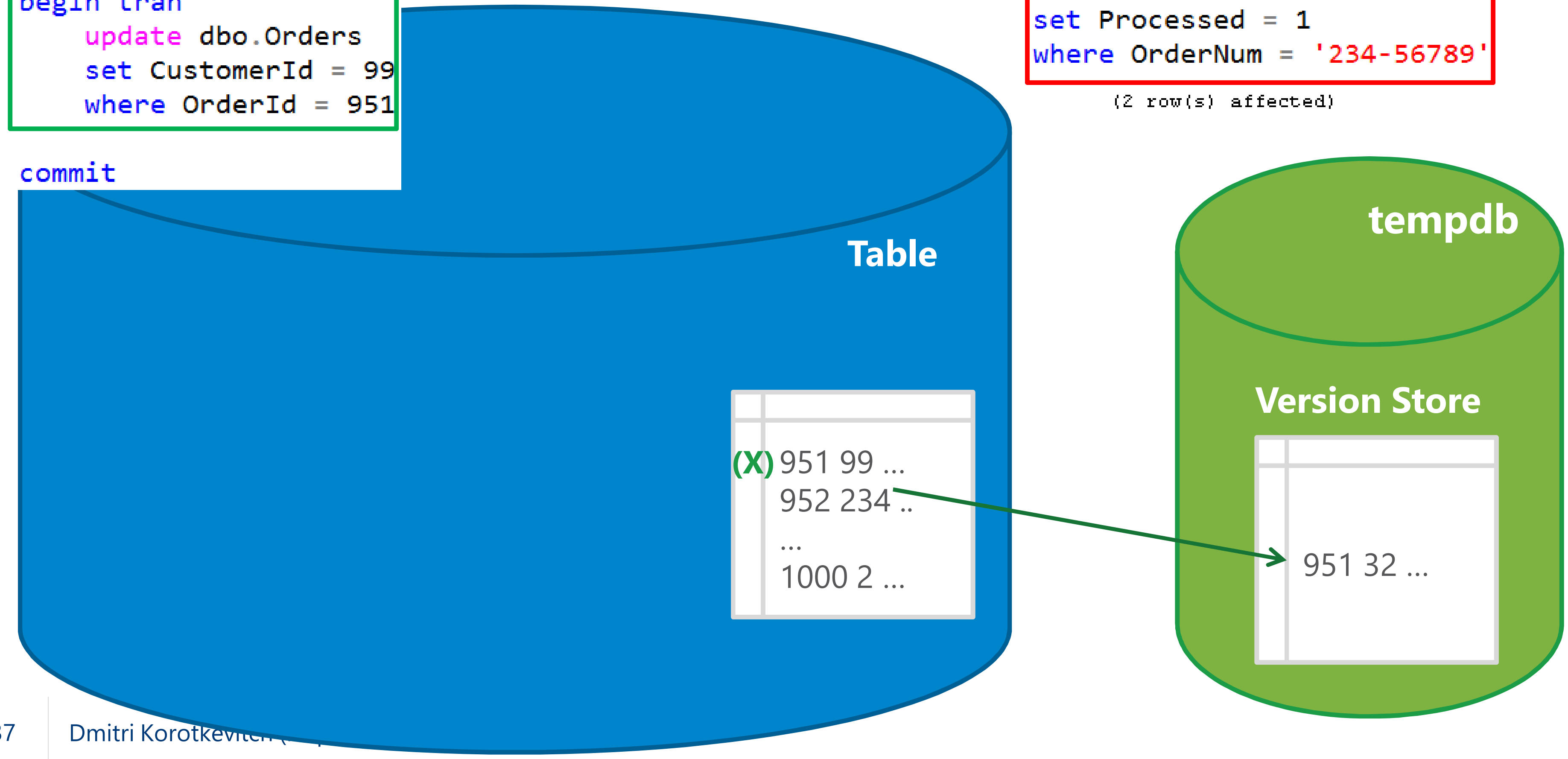
```
begin tran
    update dbo.Orders
    set CustomerId = 99
    where OrderId = 951

commit
```

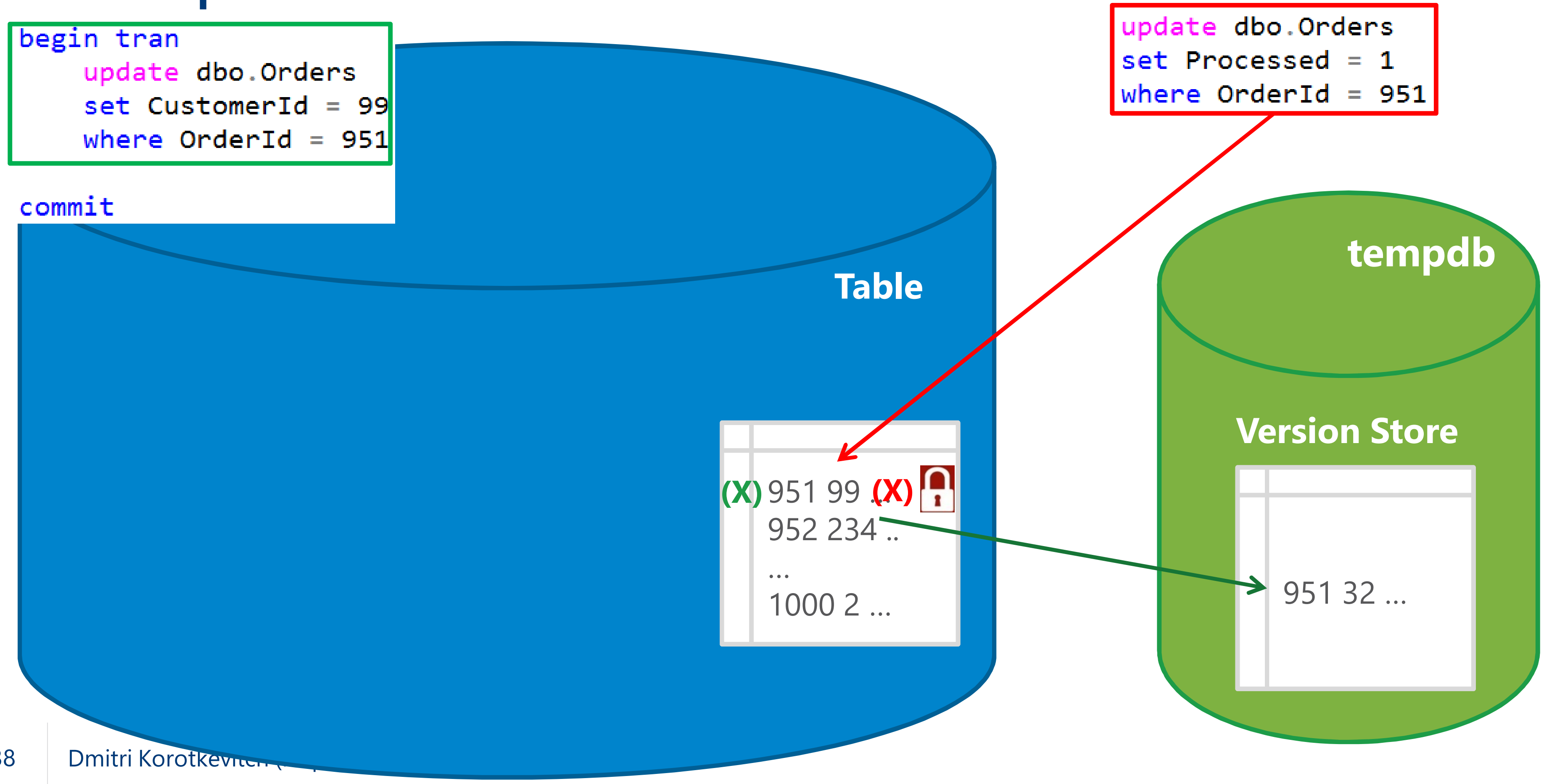
```
update dbo.Orders
set Processed = 1
where OrderNum = '234-56789'
```

```
(2 row(s) affected)
```

No (U) locks



Snapshot

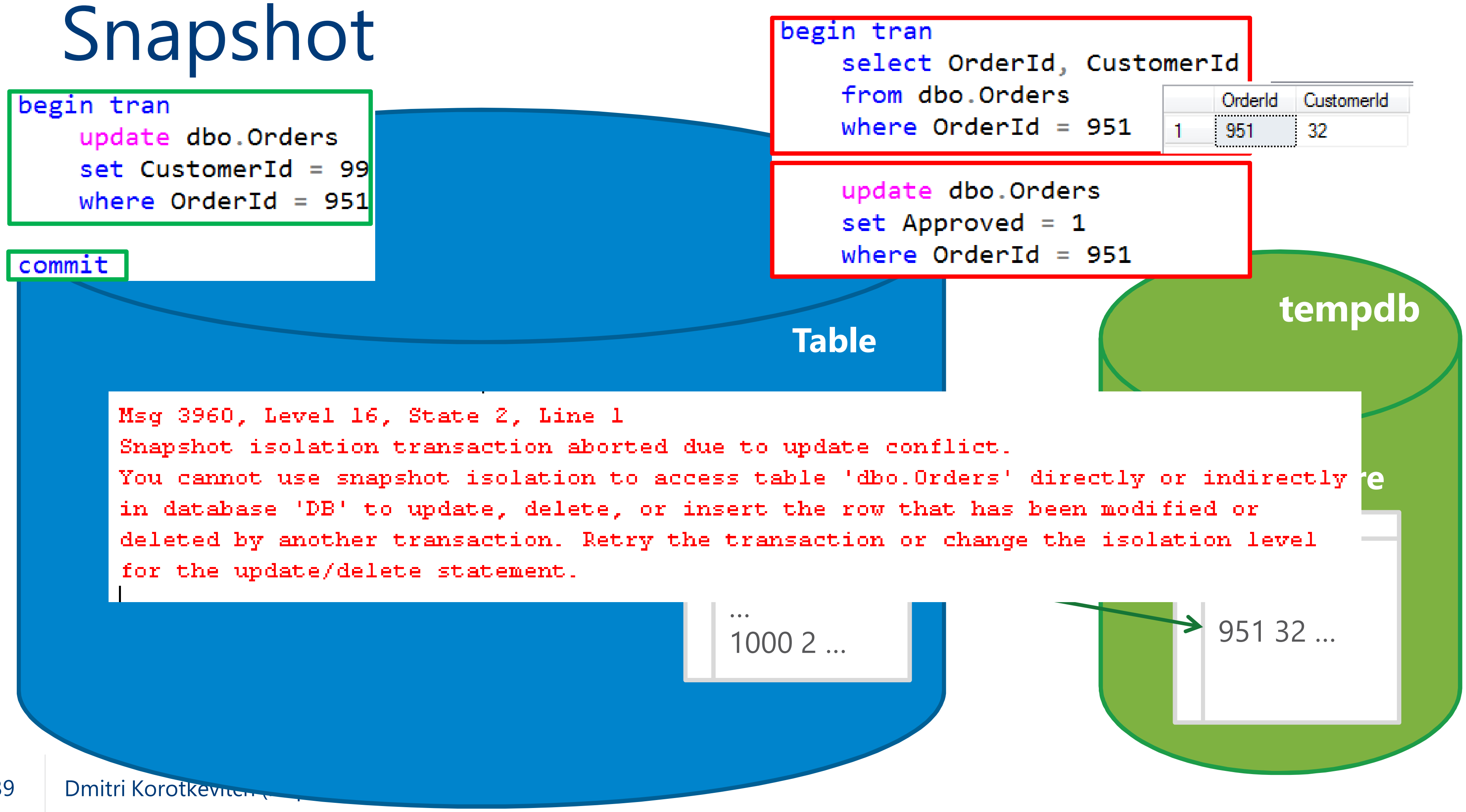


```
begin tran
    update dbo.Orders
    set CustomerId = 99
    where OrderId = 951

commit
```

```
update dbo.Orders
set Processed = 1
where OrderId = 951
```

Snapshot



```
begin tran
    update dbo.Orders
    set CustomerId = 99
    where OrderId = 951
```

```
commit
```

```
begin tran
    select OrderId, CustomerId
    from dbo.Orders
    where OrderId = 951
```

	OrderId	CustomerId
1	951	32

```
    update dbo.Orders
    set Approved = 1
    where OrderId = 951
```

```
Msg 3960, Level 16, State 2, Line 1
Snapshot isolation transaction aborted due to update conflict.
You cannot use snapshot isolation to access table 'dbo.Orders' directly or indirectly
in database 'DB' to update, delete, or insert the row that has been modified or
deleted by another transaction. Retry the transaction or change the isolation level
for the update/delete statement.
```

Snapshot (Transaction-Level Consistency)

New Transaction Isolation Level. Should also be enabled on the database level

Removes *readers/writers* and *writers/writers* blocking

TANSTAAFL!

There Ain't No Such Thing As A Free Lunch!

Old version(s) of the rows copied to the *version store*

- Increases tempdb load

14-byte version pointers added to the modified rows

- Increases index fragmentation. Do not use FILLFACTOR=100

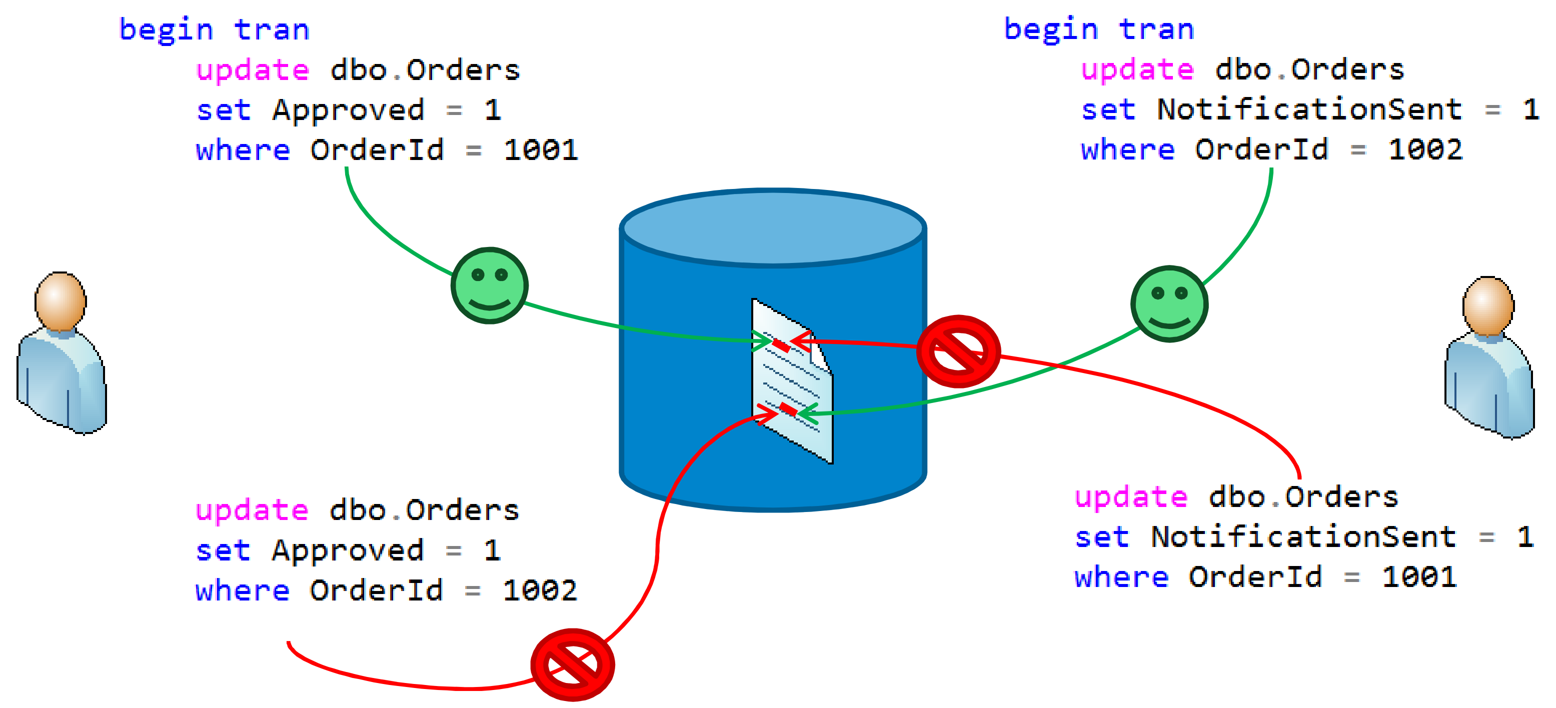
Different system behavior

- Referential integrity based on triggers does not work
- Error 3960 in SNAPSHOT
- Possible side effects due to different data consistency behavior in SNAPSHOT

Optimistic Isolation Levels

Demo

Classic Deadlock



```
begin tran
    update dbo.Orders
    set Approved = 1
    where OrderId = 1001
```

```
begin tran
    update dbo.Orders
    set NotificationSent = 1
    where OrderId = 1002
```

```
    update dbo.Orders
    set Approved = 1
    where OrderId = 1002
```

```
update dbo.Orders
set NotificationSent = 1
where OrderId = 1001
```

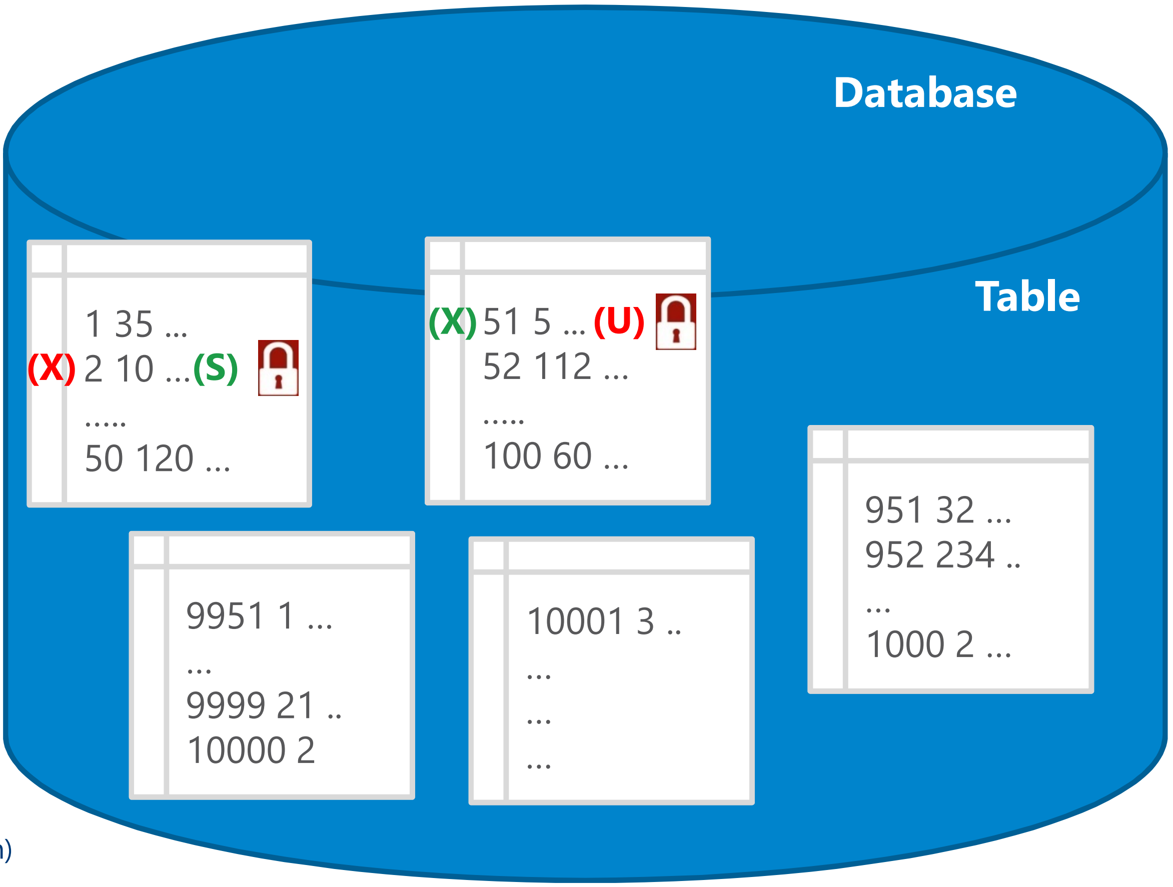
Common Deadlock

```
begin tran
    update dbo.Orders
    set Approved = 1
    where OrderId = 51
```

```
    select sum(Amount)
    from dbo.Orders
    where CustomerId = 5
```

```
begin tran
    update dbo.Orders
    set Cancelled = 1
    where OrderId = 2
```

```
    update dbo.Orders
    set Completed = 1
    where OrderNum = '234-56789'
```



Troubleshooting Deadlocks (Obtaining *Deadlock Graph*)

xEvents and SQL Traces

system_health xEvent session

TF1222 (Writing deadlock graph to SQL Server Error Log)

Event Notifications

Blocking Monitoring Framework

Troubleshooting Deadlocks

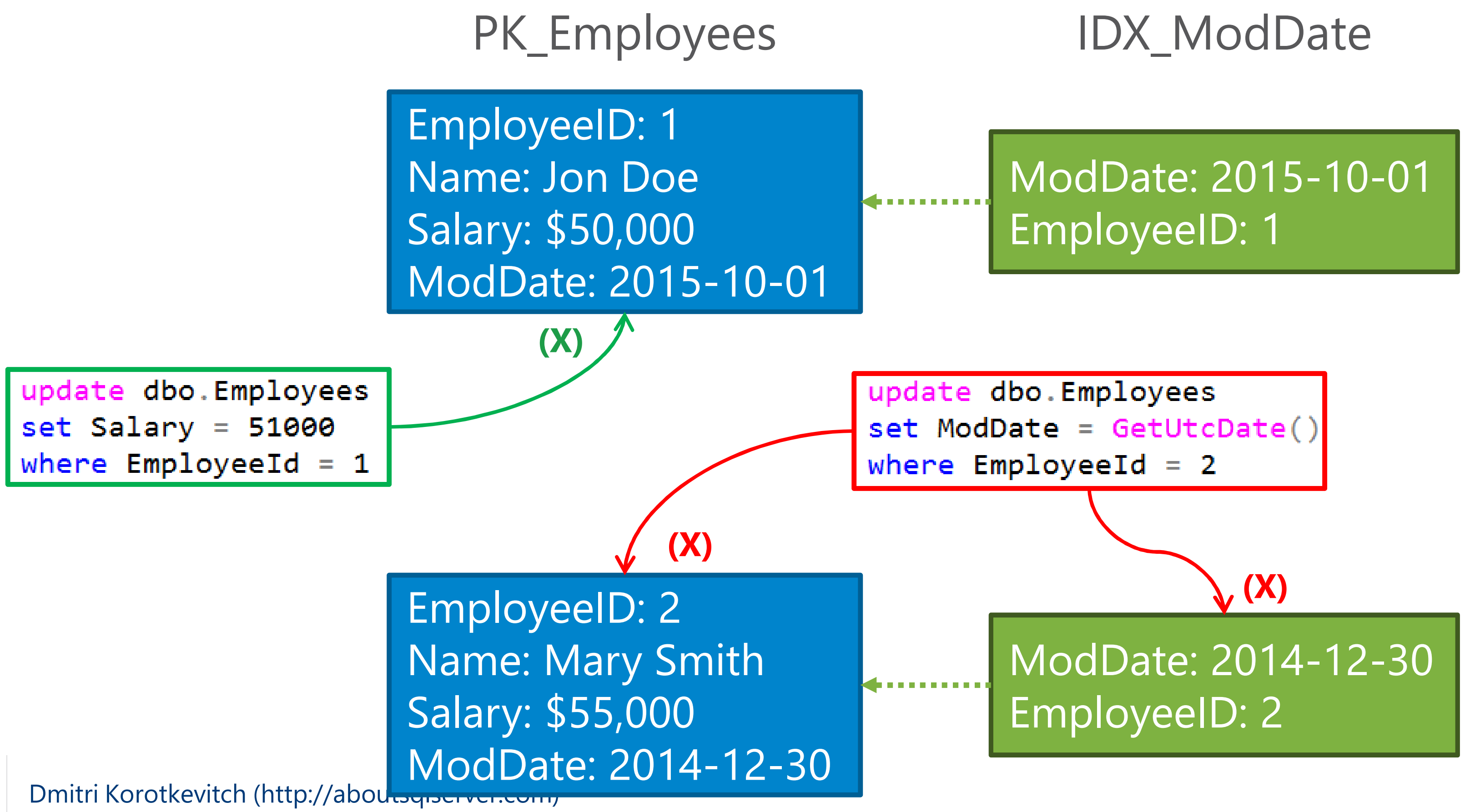
Demo

Locks and Indexes

```
create table dbo.Employees
(
     EmployeeId int not null,
     Name varchar(64) not null,
     Salary money not null,
     ModDate datetime not null,
     constraint PK_Employees
     primary key clustered(EmployeeId)
);

create nonclustered index IDX_ModDate
on dbo.Employees(ModDate);
```

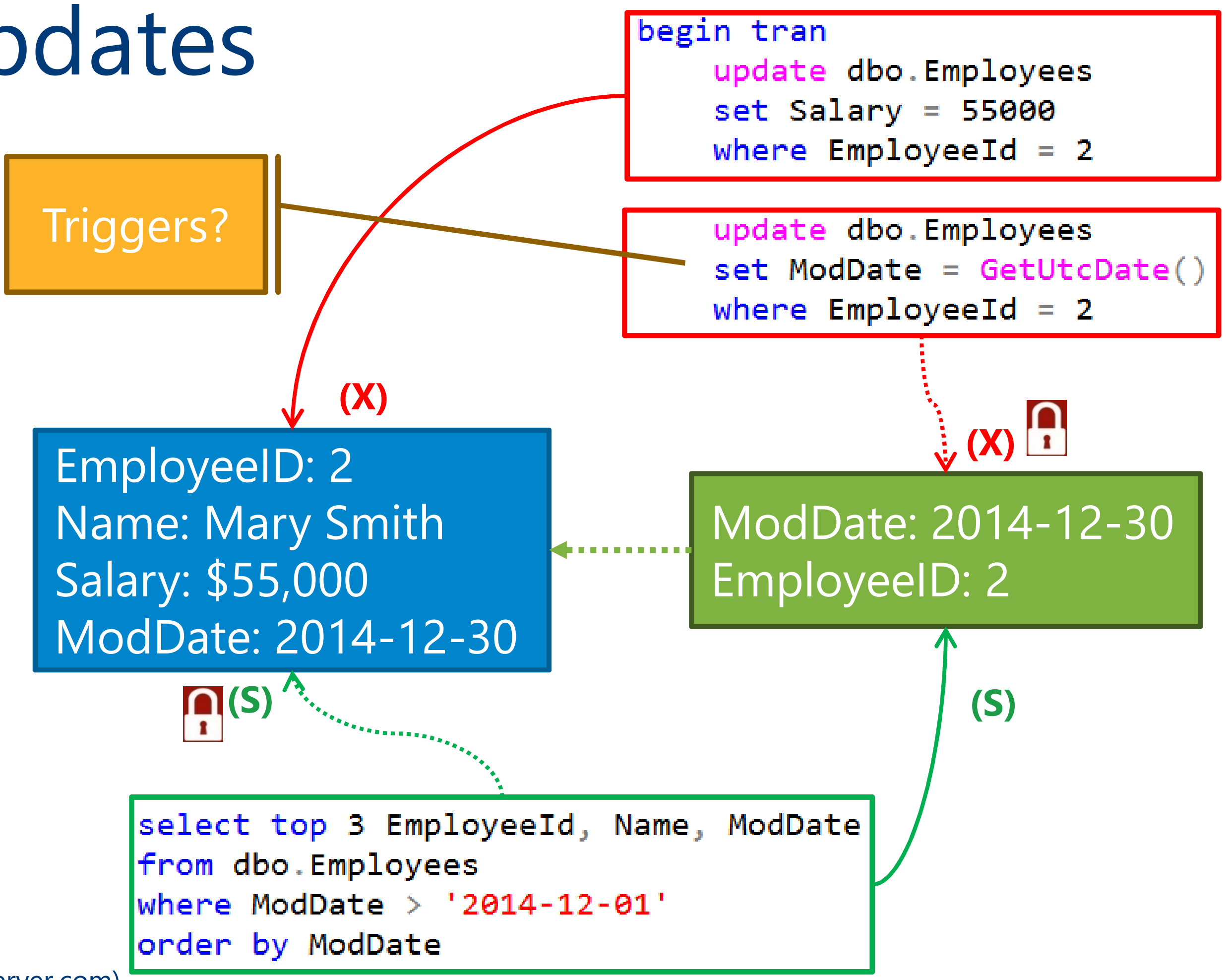
Locks and Indexes



```
update dbo.Employees
set Salary = 51000
where EmployeeId = 1
```

```
update dbo.Employees
set ModDate = GetUtcDate()
where EmployeeId = 2
```

Multiple Updates



```
begin tran
    update dbo.Employees
    set Salary = 55000
    where EmployeeId = 2
```

```
    update dbo.Employees
    set ModDate = GetUtcDate()
    where EmployeeId = 2
```

```
select top 3 EmployeeId, Name, ModDate
from dbo.Employees
where ModDate > '2014-12-01'
order by ModDate
```

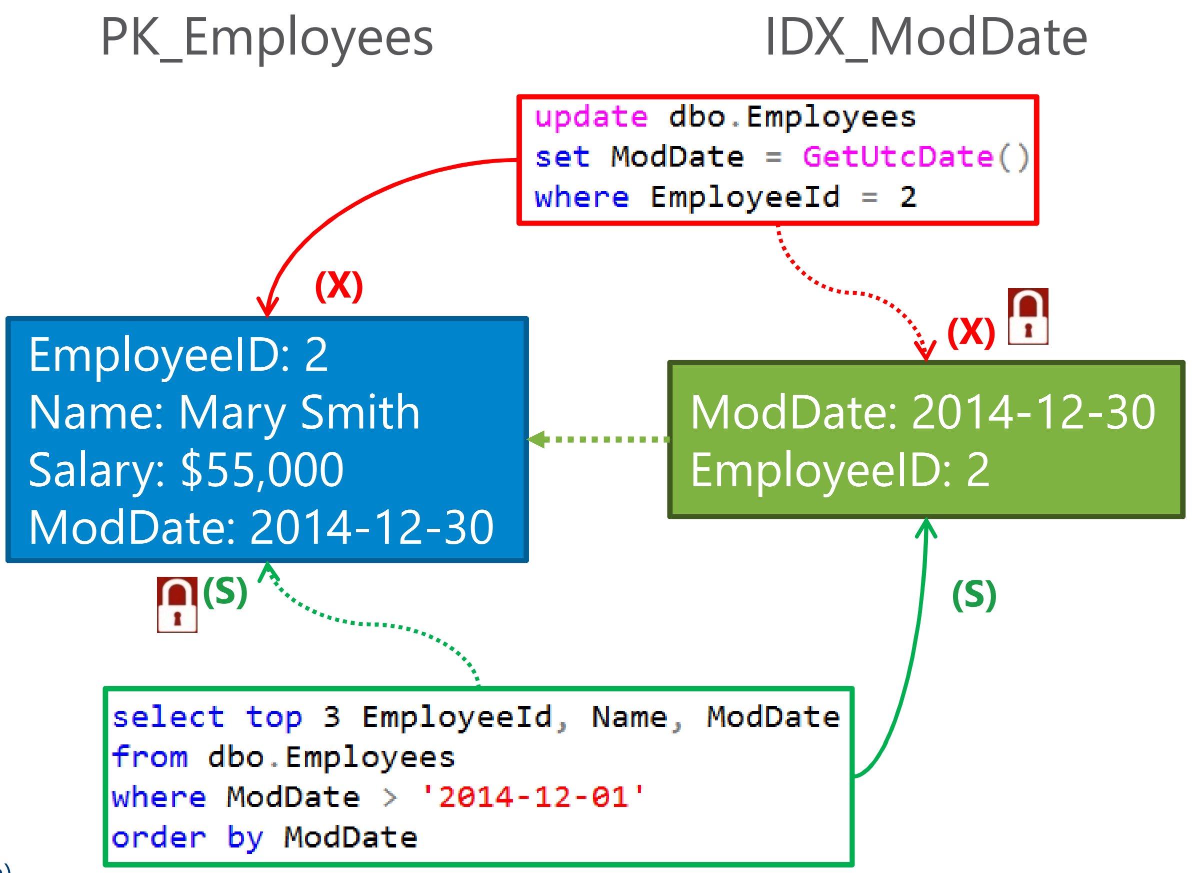
Multiple Updates Deadlock

Demo

Key Lookup Deadlock

Solutions:

- Covered Indexes
- Optimistic Isolation Levels



```
update dbo.Employees
set ModDate = GetUtcDate()
where EmployeeId = 2
```

```
select top 3 EmployeeId, Name, ModDate
from dbo.Employees
where ModDate > '2014-12-01'
order by ModDate
```

Key Lookup Deadlock

Demo

IGNORE_DUP_KEY index option

SQL Server uses SERIALIZABLE isolation level to protect index integrity with IGNORE_DUP_KEY = ON

Applies only to nonclustered indexes

IGNORE_DUP_KEY deadlock

Demo

Lock Partitioning (16+ CPUs)

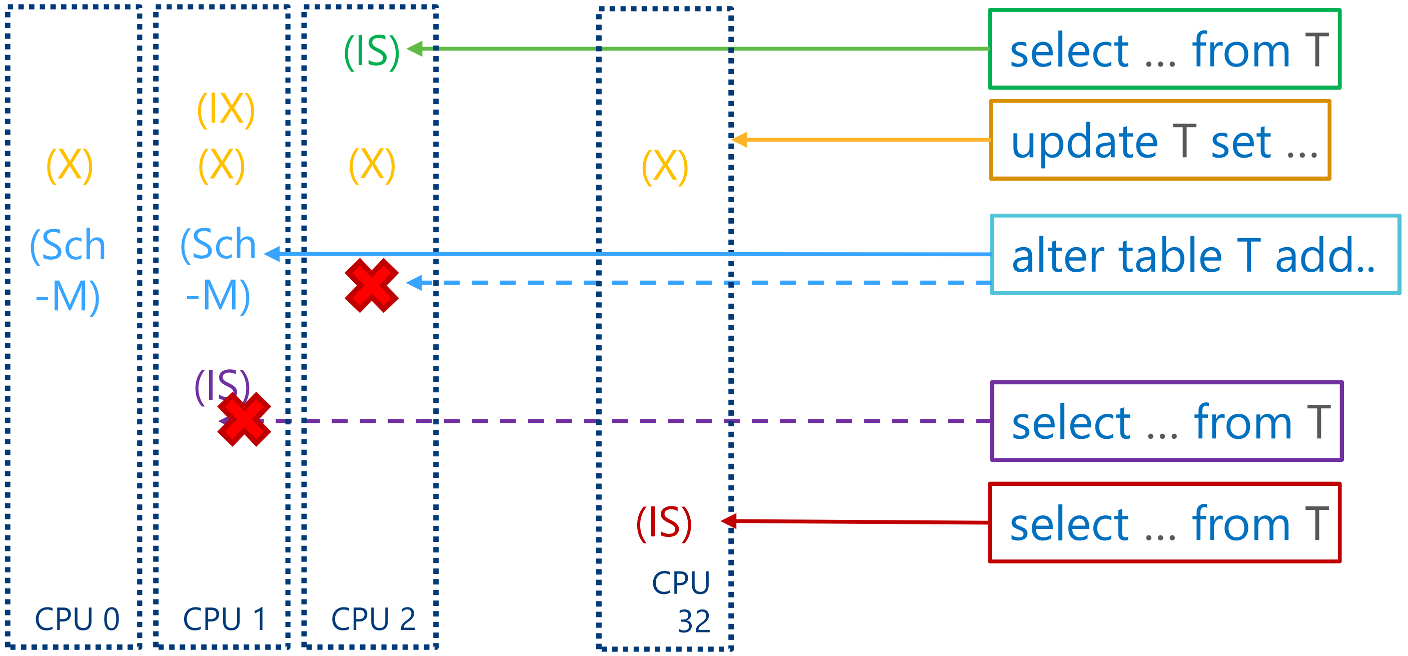
SQL Server partitions the locks on per-scheduler basis

- I*, Sch-S locks are kept in the single partition
- Other types are acquired in all partitions. Can significantly increase memory usage
- Undocumented T1229 disables it. Can lead to contention.

Can lead to the deadlocks:

- Mixing intent and full object locks in the same transaction
- Sometimes (rarely) multiple statements can use different lock partitions in the same transaction on busy servers

Lock Partitioning (16+ CPUs)



Lock Partitioning

Demo

What Isolation Level Should I Choose?

It depends ☺

Do not use READ UNCOMMITTED unless you do not care about data consistency

SNAPSHOT or RCSI is good for DW/Reporting/Analytical workload

For OLTP:

READ COMMITTED is *good enough* when queries are optimized

READ COMMITTED SNAPSHOT is *good* when *tempdb* throughput is sufficient

- Remember about extra index fragmentation in OLTP

Wait Types

Metric	Possible Causes
LCK_M_SCH_*	Index and/or partition maintenance, frequent schema modifications (app issues)
LCK_M_I*	Lock escalation, schema modifications (see LCK_M_SCH*)
LCK_M_RS*	SERIALIZABLE transactions, IGNORE_DUP_KEY option in NCI
LCK_M_U	Nonoptimized DML queries. Perform query tuning. Analyze transaction management
LCK_M_S	Nonoptimized SELECT queries. Perform query tuning. Consider RCSI to hide the problem.

sys.dm_db_index_operational_stats

- lock_count, lock_wait_count, lock_wait_ms on row and page levels
- Lock Escalation statistics

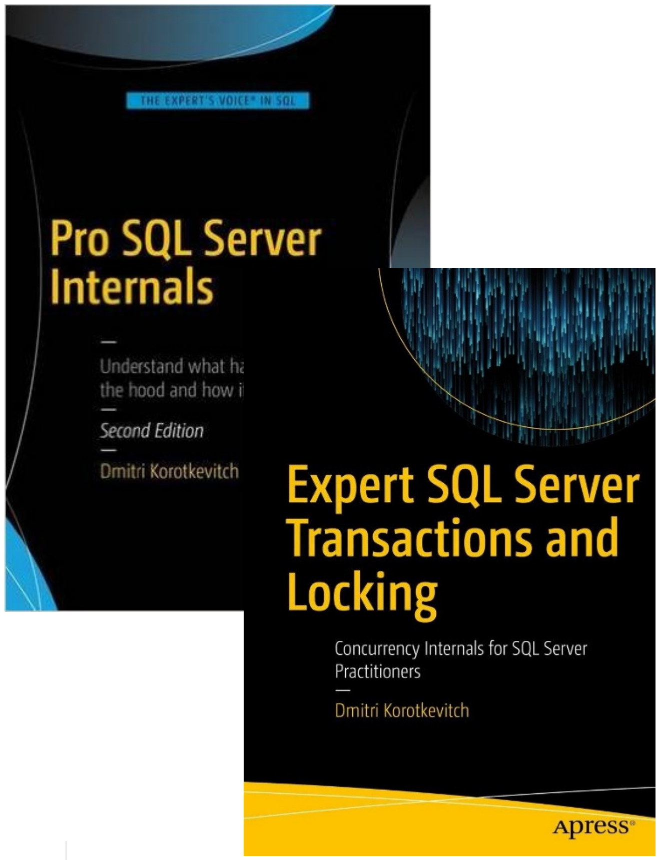
Key Points

1st rule of improving concurrency in the system – optimize queries
2nd rule of improving concurrency in the system – optimize queries
3rd rule of improving concurrency in the system – optimize queries
Choose appropriate isolation level
Use short transactions
Do not update entities multiple time in the same transaction
Be careful with ORM frameworks and SQL Generators

https://aboutsqlserver.com – a lot of stuff
"Expert SQL Server Transactions and Locking"
"Pro SQL Server Internals"

Slides and Demos: PASS Summit site & https://aboutsqlserver.com/presentations
Blocking monitoring framework: http://aboutsqlserver/bmframework

Q&A



Email me:
dk@aboutsqlserver.com

Slides and Demos:
https://aboutsqlserver.com/presentations

Session Evaluations

Submit by 5pm Friday, November 15th to win prizes.

3 WAYS TO ACCESS

- Go to PASSsummit.com
- Download the GuideBook App and search: PASS Summit 2019
- Follow the QR code link on session signage

Thank You

Dmitri Korotkevitch

@aboutsqlserver

dmitri@aboutsqlserver.com