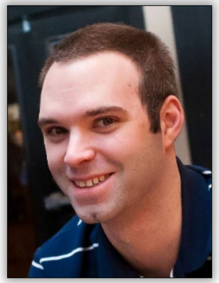
Locks, Blocks, and Snapshots: Maximizing Database Concurrency

Bob Pusateri

About Bob Pusateri



Microsoft
CERTIFIED
Master

SQL Server 2008

@bobp.bsky.social
@sqlbob
bob@bobpusateri.com
bobpusateri.com
bobpusateri

MVP
Microsoft®
Most Valuable Professional

Passions:

- Performance Tuning & Troubleshooting
- Very Large Databases
- Storage Engine Internals
- Big Data
- Cloud Architecture
- Teaching & Helping
- #BobFacts

ChiSQL

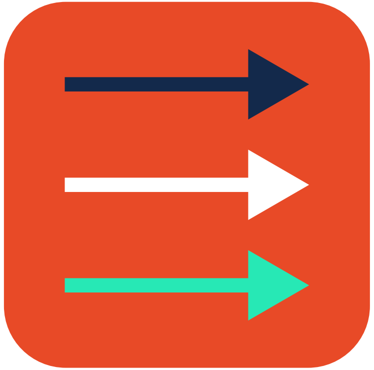
FRIEND OF
redgate
2023

Agenda

- Concurrency Basics
- Isolation Levels
- In-Memory OLTP

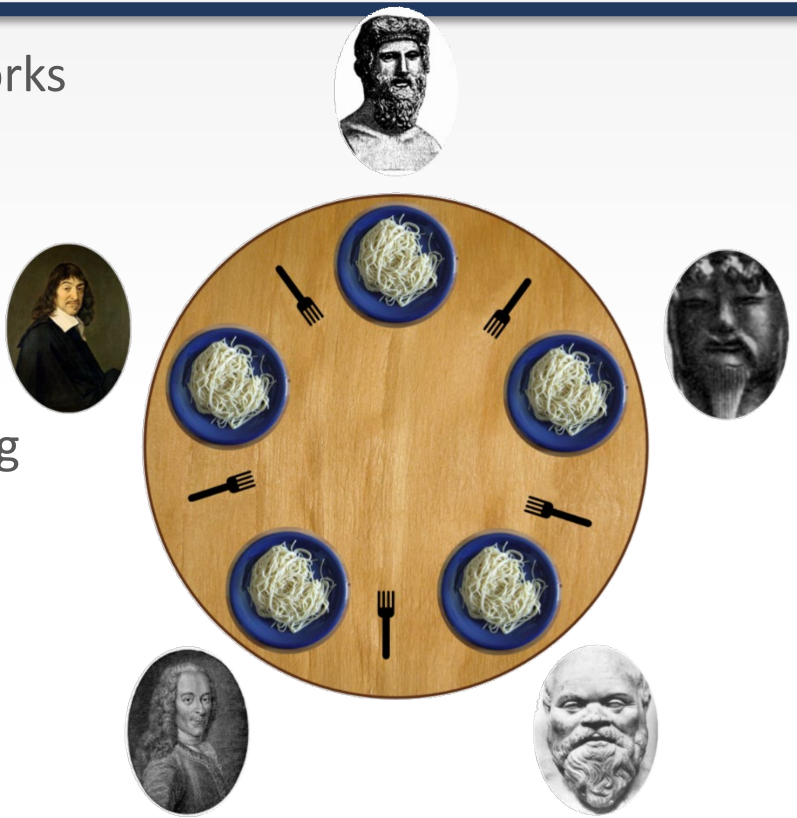
Concurrency Basics: What Is Concurrency?

- The ability for an operation to be broken up into multiple parts that can be worked on independently
- The ability for multiple operations to access or modify a shared resource at the same time
- More parts/users == more concurrency!
- Until a limiting factor appears…



The Dining Philosophers Problem

- 5 Philosophers, bowls of spaghetti, and forks
- To eat, 2 forks are required
- Can pick up one fork at a time
- Once finished, both forks must be returned to the table
- When not eating, a philosopher is thinking



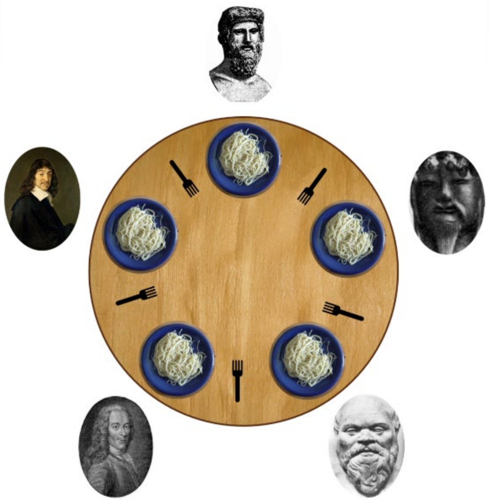
https://en.wikipedia.org/wiki/File:An_illustration_of_the_dining_philosophers_problem.png

The Dining Philosophers Problem

What if everyone picks up one fork?

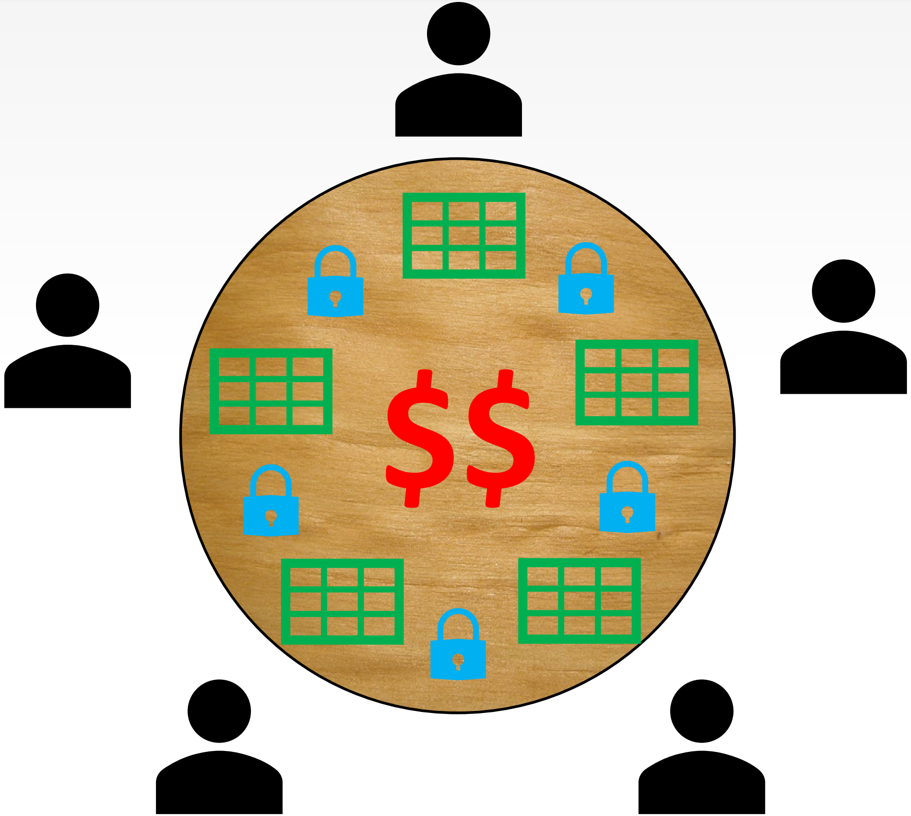
What if there's a time limit?

What if someone never gets to eat?



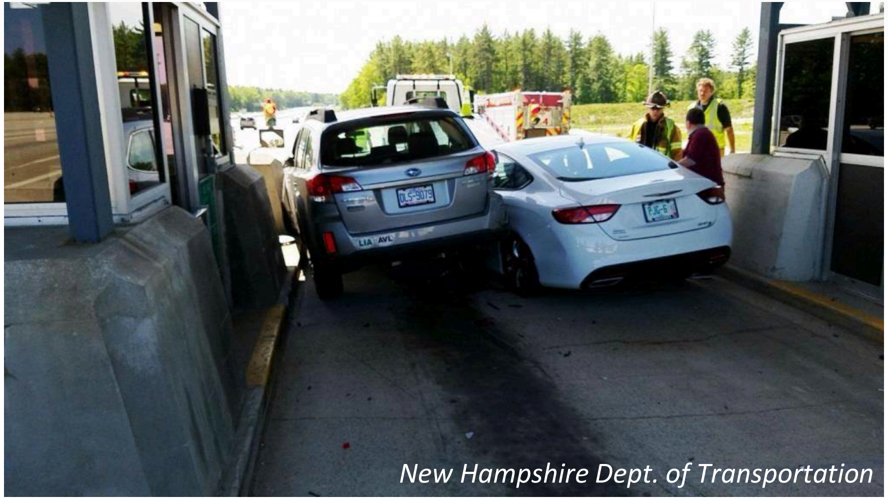
https://en.wikipedia.org/wiki/File:An_illustration_of_the_dining_philosophers_problem.png

The ~~Dining Philosophers~~ DATABASE Problem



Concurrency Conflicts

- Preventable Read Phenomena (ANSI-defined*)
  - Dirty Reads
  - Non-Repeatable Reads
  - Phantom Reads
- Lost Updates
- Deadlocks

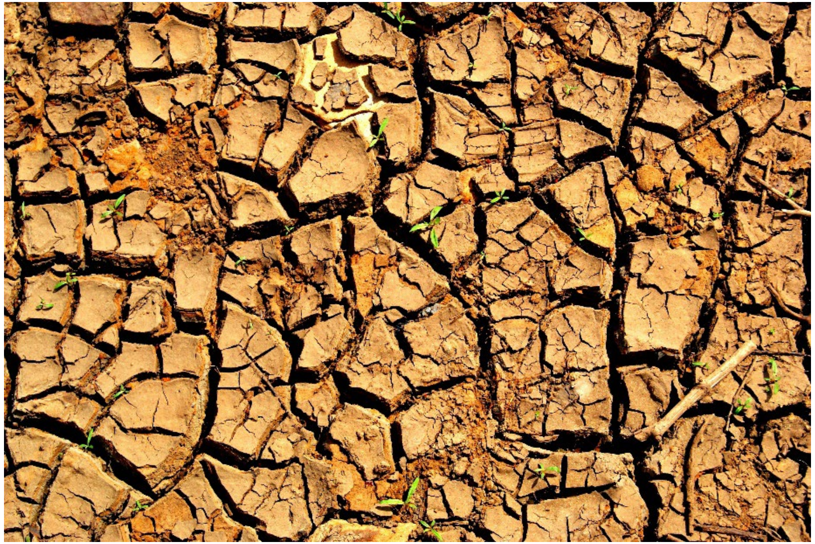


New Hampshire Dept. of Transportation

**ANSI specifies which behaviors to allow at each level, but not how to implement them*

Dirty Reads

- Reading data that is not yet committed
- Changes are still “in flight” in another process
- You can read data multiple times or not at all
- “But it’s faster!”



https://flic.kr/p/8ihVag

Non-repeatable Reads

- A.K.A. “Inconsistent Analysis”
- Multiple queries **in the same transaction** get differing results
- Cause: A different transaction commits changes between reads

Phantom Reads

- Only affects queries with a predicate (WHERE clause)
- Membership in the result set changes
- Multiple queries **using the same predicate** in the same transaction return differing results

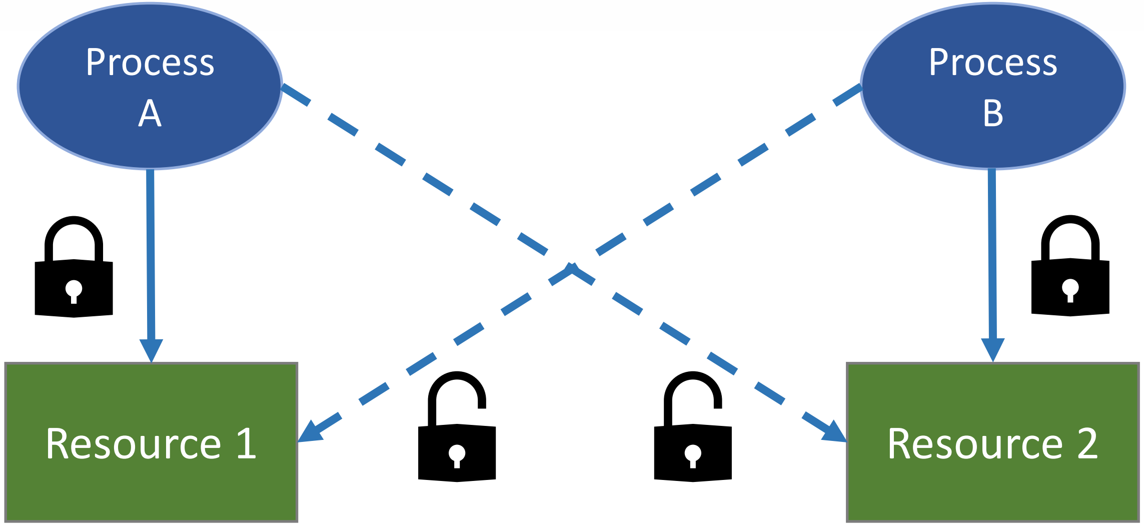
Lost Updates

- “Update Conflict”
- One user’s update overwrites another user’s (simultaneous) update
- Appears as though the first update never happened

**SQL Server will not permit lost updates in any isolation level*

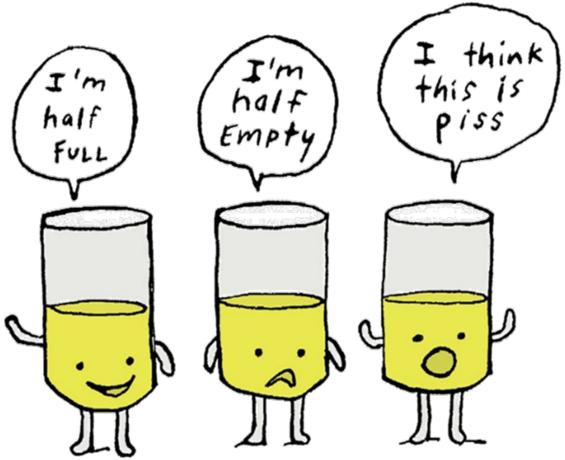
Deadlocks

- Two or more tasks block each other
- Each has a lock on a resource that the other task needs a lock on
- SQL Server detects and resolves these by choosing a victim
- Victim is rolled back, releasing all its locks



Ways to Address These Issues

- **Pessimistic Concurrency**
  - Conflicts are expected; locks taken to prevent them
  - Readers block writers, writers block readers
  - Only option available pre-2005
- **Optimistic Concurrency**
  - Conflicts are considered possible, but unlikely
  - Row versioning means <u>**less**</u> locking



Agenda

- Concurrency Basics
- Isolation Levels
- In-Memory OLTP

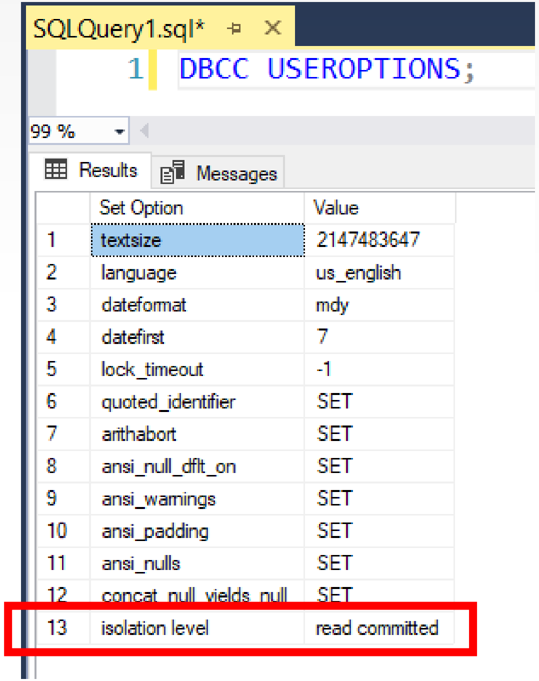
Isolation Levels

- *How isolated is my transaction from the effects of other transactions?*
- **Pessimistic Isolation Levels**
  - Read Committed [*default* default]
  - Read Uncommitted
  - Repeatable Read
  - Serializable
- **Optimistic Isolation Levels**
  - Snapshot
  - Read Committed Snapshot [*alternate* default]

https://flic.kr/p/7LjDDL

Finding the Current Isolation Level

- DBCC USEROPTIONS



```
DBCC USEROPTIONS;
```

	Set Option	Value
1	textsize	2147483647
2	language	us_english
3	dateformat	mdy
4	datefirst	7
5	lock_timeout	-1
6	quoted_identifier	SET
7	arithabort	SET
8	ansi_null_dflt_on	SET
9	ansi_warnings	SET
10	ansi_padding	SET
11	ansi_nulls	SET
12	concat_null_yields_null	SET
13	isolation level	read committed

Changing Isolation Levels

- Can be set at the **connection** or **query** level
- Cannot be changed server-wide
- *Default* default isolation level is READ COMMITTED
- To change at connection level:

```
SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED
| READ COMMITTED | REPEATABLE READ | SNAPSHOT
| SERIALIZABLE } [;]
```

- Change applies to all queries for the current connection

Changing Isolation Levels

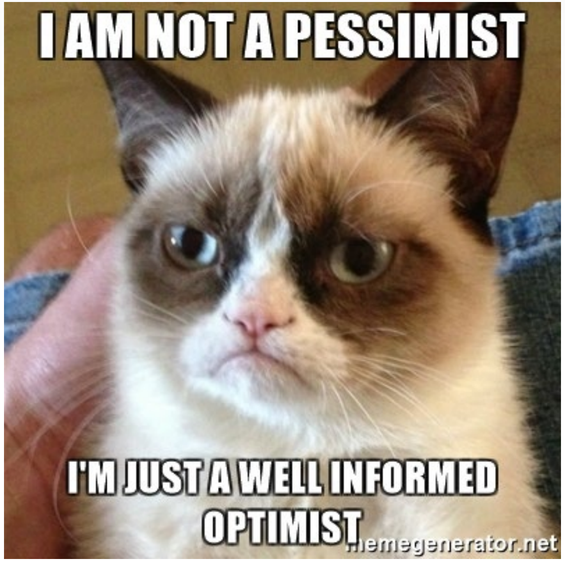
- To change at the query level, use table hints

```
SELECT column
FROM table WITH (NOLOCK);
```

- This is on a per-table basis

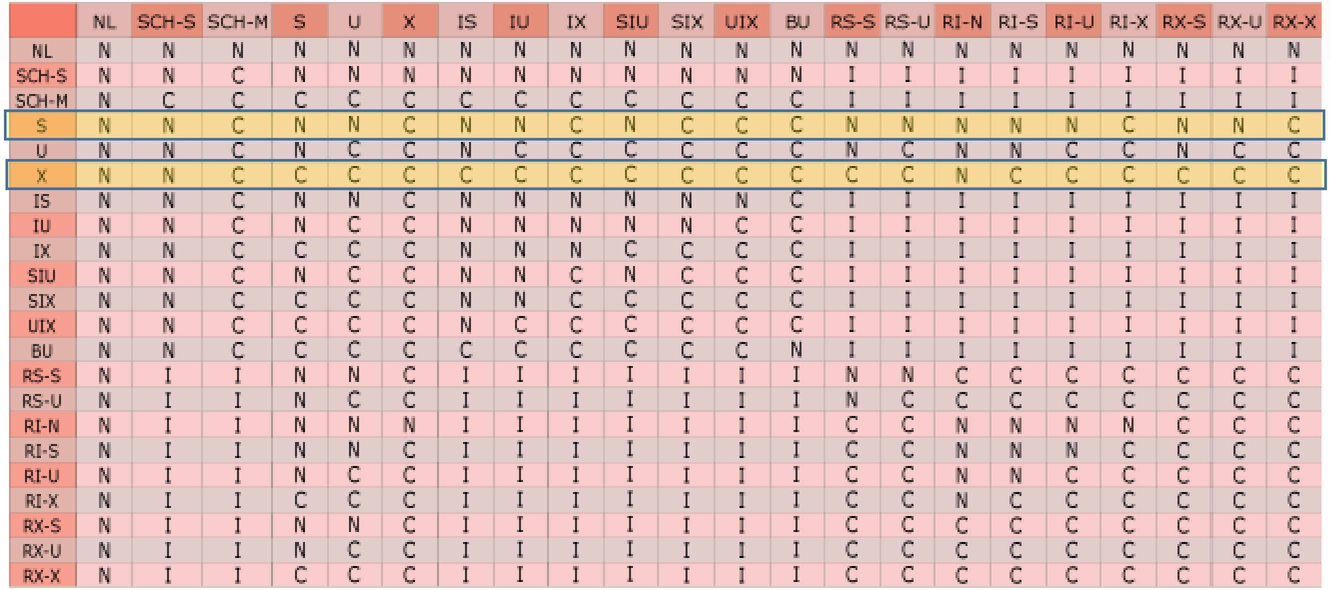
Pessimistic Concurrency

- Uses locking to prevent concurrency conflicts
- Classic locking model
  - Readers **don't** block readers
  - Readers block writers
  - Writers block readers
  - Writers block writers
- PESSIMISTIC – we're expecting problems



Lock Modes

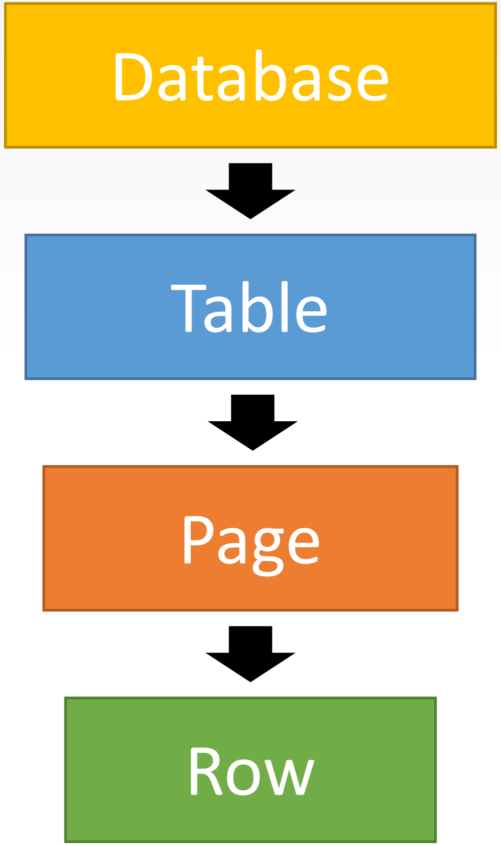
- Many different lock modes exist
- S = Shared
- X = eXclusive



https://technet.microsoft.com/en-us/library/ms186396(v=sql.105).aspx

	NL	SCH-S	SCH-M	S	U	X	IS	IU	IX	SIU	SIX	UIX	BU	RS-S	RS-U	RI-N	RI-S	RI-U	RI-X	RX-S	RX-U	RX-X
NL	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SCH-S	N	N	C	N	N	N	N	N	N	N	N	N	N	I	I	I	I	I	I	I	I	I
SCH-M	N	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
S	N	N	C	N	N	C	N	N	C	N	C	C	C	N	N	N	N	N	C	N	N	C
U	N	N	C	N	C	C	N	C	C	C	C	C	C	N	C	N	N	C	C	N	C	C
X	N	N	C	C	C	C	C	C	C	C	C	C	C	C	C	N	C	C	C	C	C	C
IS	N	N	C	N	N	C	N	N	N	N	N	N	C	I	I	I	I	I	I	I	I	I
IU	N	N	C	N	C	C	N	N	N	N	N	C	C	I	I	I	I	I	I	I	I	I
IX	N	N	C	C	C	C	N	N	N	C	C	C	C	I	I	I	I	I	I	I	I	I
SIU	N	N	C	N	C	C	N	N	C	N	C	C	C	I	I	I	I	I	I	I	I	I
SIX	N	N	C	C	C	C	N	N	C	C	C	C	C	I	I	I	I	I	I	I	I	I
UIX	N	N	C	C	C	C	N	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
BU	N	N	C	C	C	C	C	C	C	C	C	C	N	I	I	I	I	I	I	I	I	I
RS-S	N	I	I	N	N	C	I	I	I	I	I	I	I	N	N	C	C	C	C	C	C	C
RS-U	N	I	I	N	C	C	I	I	I	I	I	I	I	N	C	C	C	C	C	C	C	C
RI-N	N	I	I	N	N	N	I	I	I	I	I	I	I	C	C	N	N	N	N	C	C	C
RI-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	N	N	N	C	C	C	C
RI-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	N	N	C	C	C	C	C
RI-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	N	C	C	C	C	C	C
RX-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C

Lock Hierarchy 101



Database

Table

Page

Row

Lock Hierarchy 101: Read Operations

Database	[S] Shared Lock
Table	[IS] Intent Shared Lock
Page	[IS] Intent Shared Lock
Row	[S] Shared Lock

Lock Hierarchy 101: Write Operations

Database	[S] Shared Lock
↓ Table	[IX] Intent Exclusive Lock **OR** [IU] Intent Update Lock
↓ Page	[IX] Intent Exclusive Lock **OR** [IU] Intent Update Lock
↓ Row	[X] Exclusive Lock **OR** [U] Update Lock

Read Uncommitted

Dirty	Nonrepeatable	Phantom
Yes	Yes	Yes

- A.K.A “NOLOCK”
- May read rows multiple times
- May read rows zero times
- May return results that were NEVER true!
- **Only applies to SELECT queries**
  - Ignored if used for data modification queries
  - Could cause index corruption if you tried it (since fixed)

Read Uncommitted

(public domain) https://commons.wikimedia.org/wiki/File:8_ball_break_time_lapse.jpg

Read Uncommitted Myths

- “No locks are taken”
  - WRONG!
  - No *shared* locks are taken when reading data
  - Other locks are still taken as normal
- “It makes this query faster”
  - WRONG(ish)!
  - Only true if query had a lot of blocking on SELECT statements

Demo

Read Uncommitted

- Not a terrible setting, it exists for a reason
- BUT make sure you understand the risks and consequences
- Make sure the business knows this too

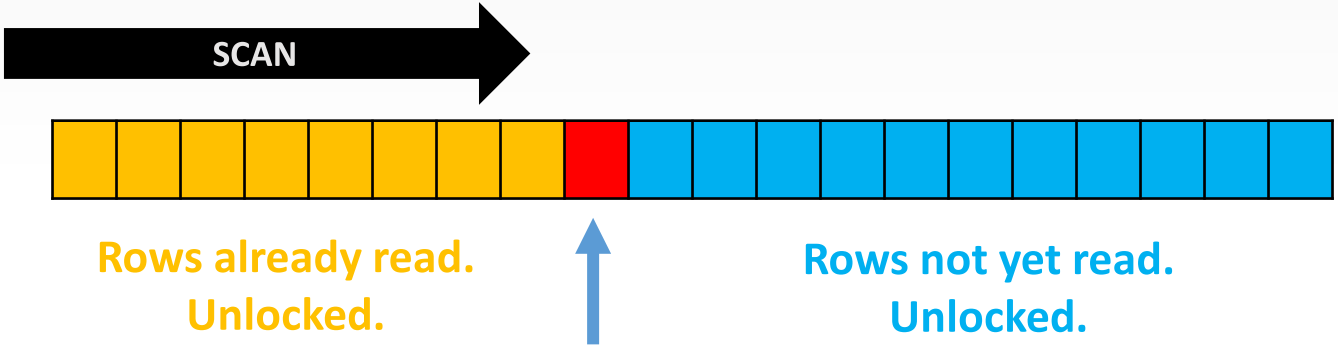
Read Committed

Dirty	Nonrepeatable	Phantom
No	Yes	Yes

- The *Default* Default Isolation level
- Guarantee: Data that is read is guaranteed to be committed.
  - No Dirty Reads
  - No other guarantees

Read Committed

- Ensure only committed data is read by locking
- (Locks only last as long as the read, released immediately after)

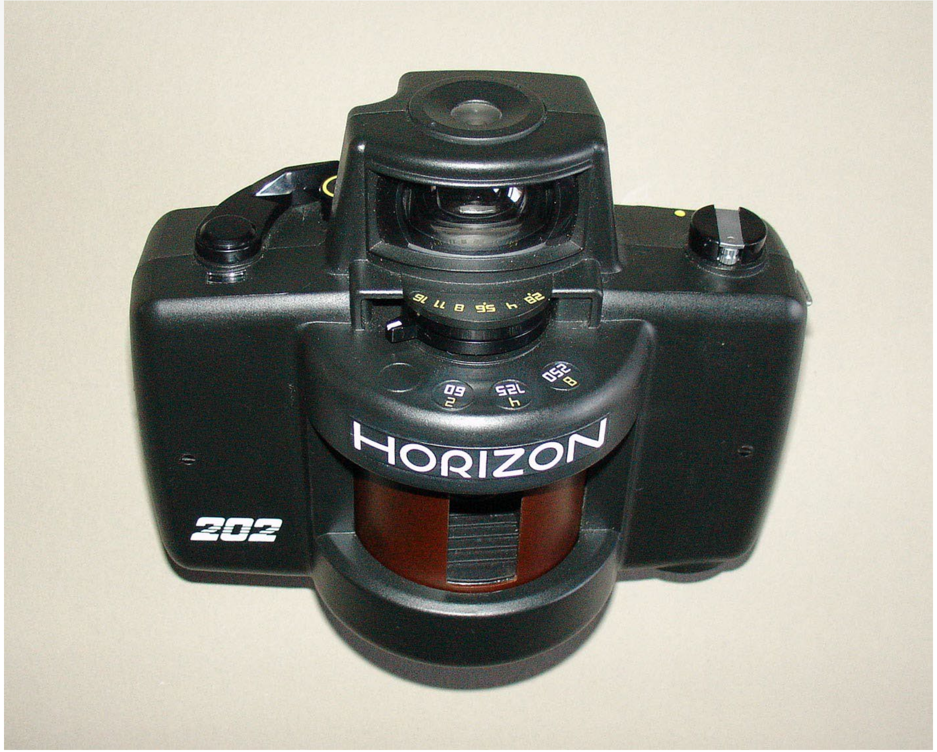


Rows already read.
Unlocked.

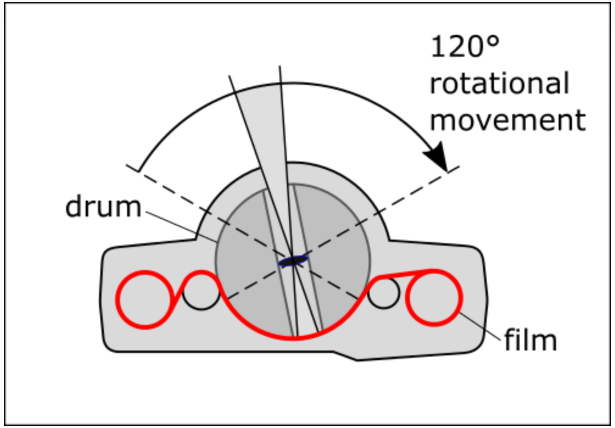
Rows not yet read.
Unlocked.

Row currently being
read. Locked.

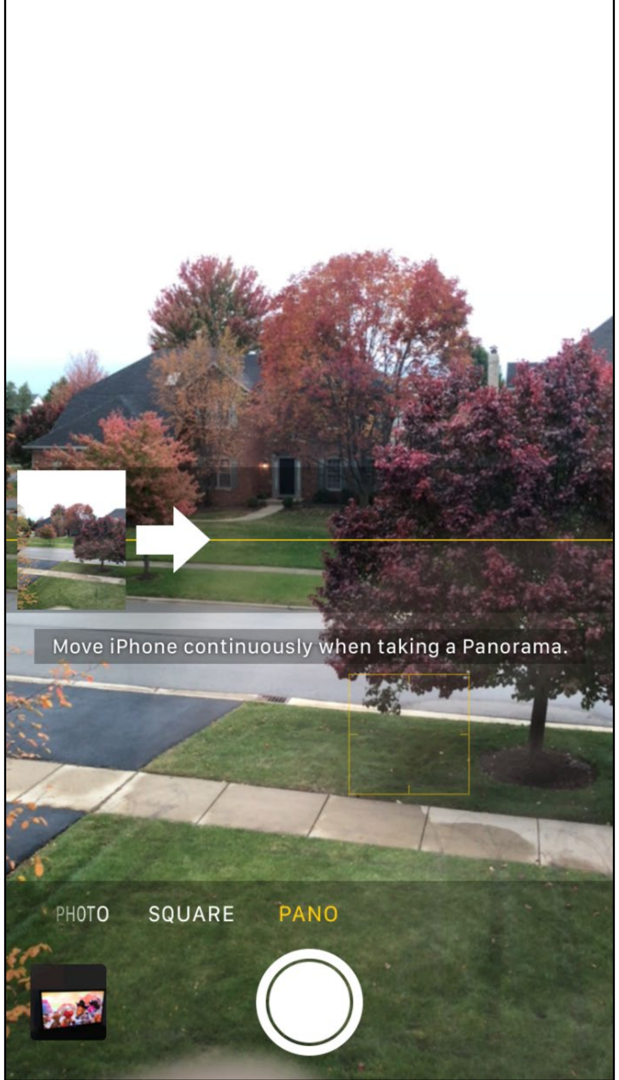
Let’s Talk Photography: Swing Lens Cameras



https://commons.wikimedia.org/wiki/File:Horizon202.jpg



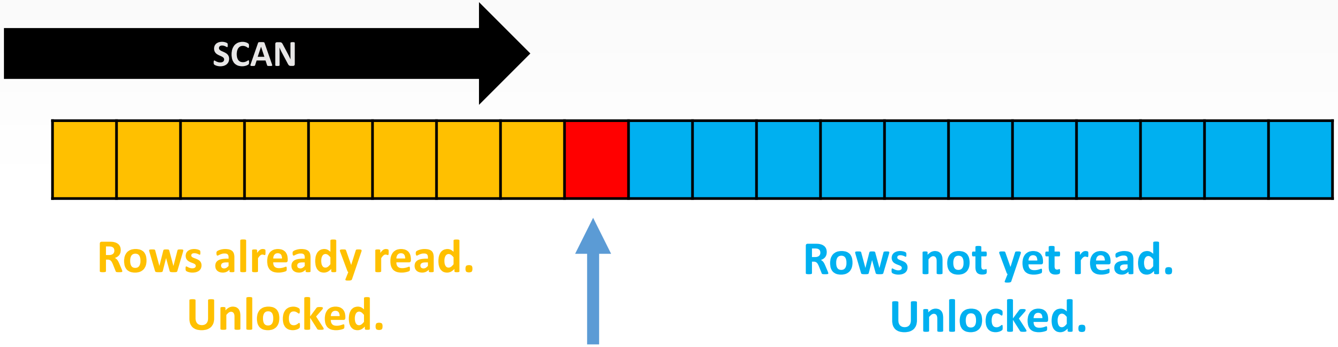
https://commons.wikimedia.org/wiki/File:Horizon202sketch.png



Move iPhone continuously when taking a Panorama.

Read Committed

- Unlocked rows can move at any time



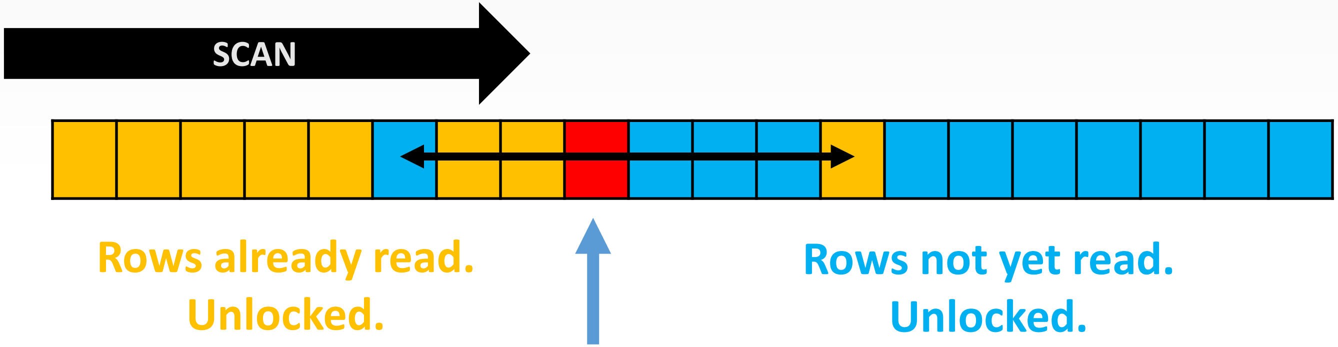
Rows already read. Unlocked.

Rows not yet read. Unlocked.

Row currently being read. Locked.

Read Committed

- Unlocked rows can move at any time



Rows already read.
Unlocked.

Rows not yet read.
Unlocked.

Row currently being
read. Locked.

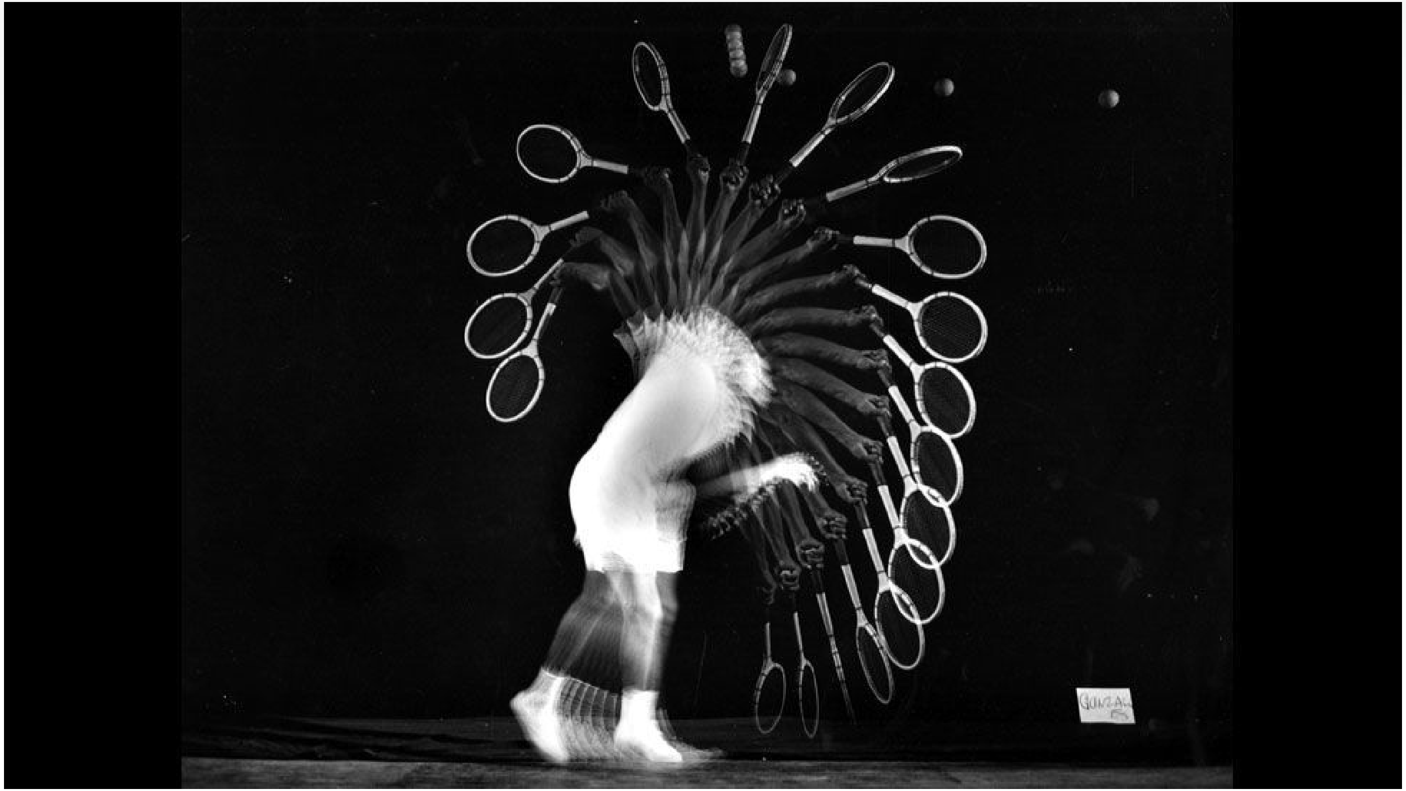
Demo

Repeatable Read

Dirty	Nonrepeatable	Phantom
No	No	Yes

- Builds on READ COMMITTED
- If a query is repeated within the same transaction, records read the first time will not change
- Once a row is read, locks are held for length of the transaction
  - Even rows that don’t qualify as results
- These locks will not stop additional rows from being added or included in subsequent queries

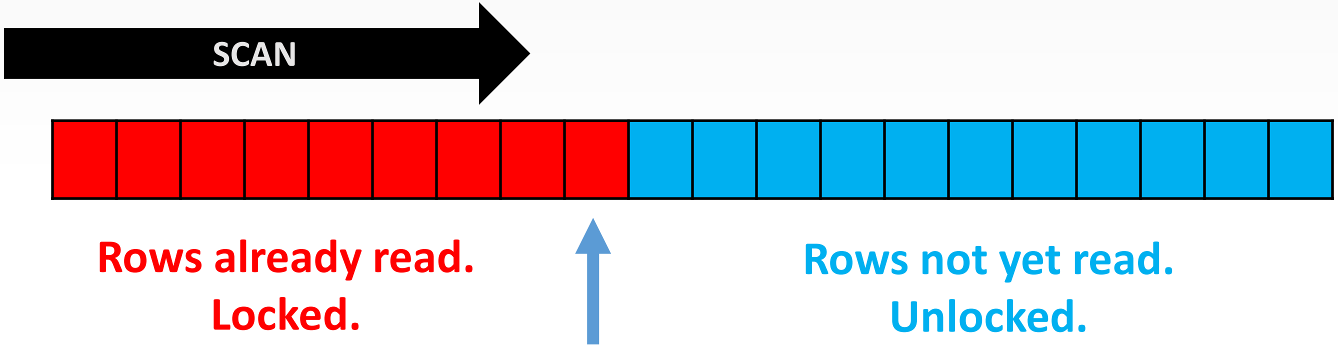
Repeatable Read



Harold Edgerton Archive, MIT

Repeatable Read

- Once read, rows are locked for duration of transaction



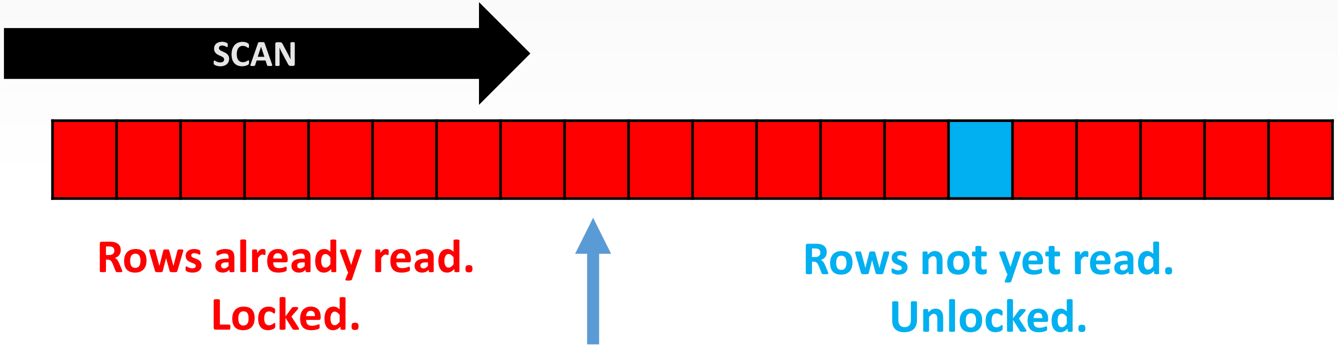
Rows already read. Locked.

Row currently being read. Locked.

Rows not yet read. Unlocked.

Repeatable Read

- On a second scan, new rows may enter



SCAN

Rows already read. Locked.

Rows not yet read. Unlocked.

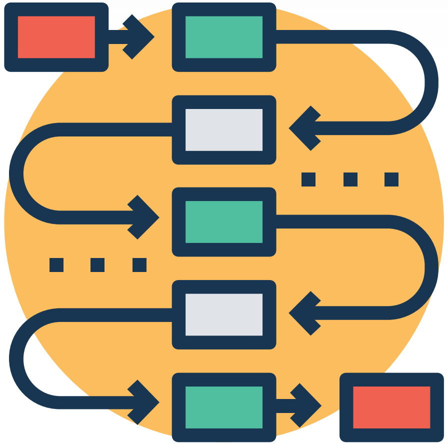
Row currently being read. Locked.

Demo

Serializable

Dirty	Nonrepeatable	Phantom
No	No	No

- Builds on REPEATABLE READ
- If a query is repeated within the same transaction, results will be the same
- No data seen previously will change; no new results will appear
- We now need to lock data that doesn't exist!

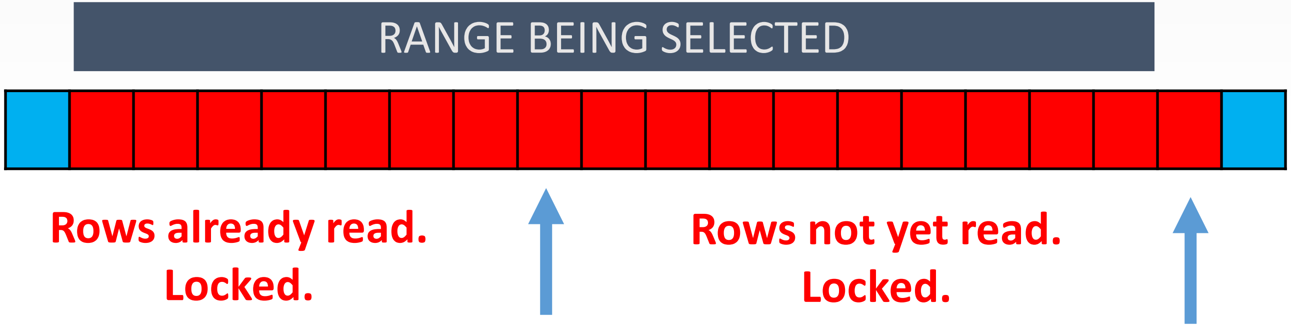


Serializable

- Key-range locks
- Prevent phantom reads by defining a range that other transactions cannot insert rows within
- If you select a row/range that doesn't exist, that gets locked too

Serializable

- Range Locks cover the entire range AND the first row outside it



RANGE BEING SELECTED

Rows already read. Locked.

Rows not yet read. Locked.

Row currently being read. Locked.

First key outside range. Locked.

Serializable

Demo

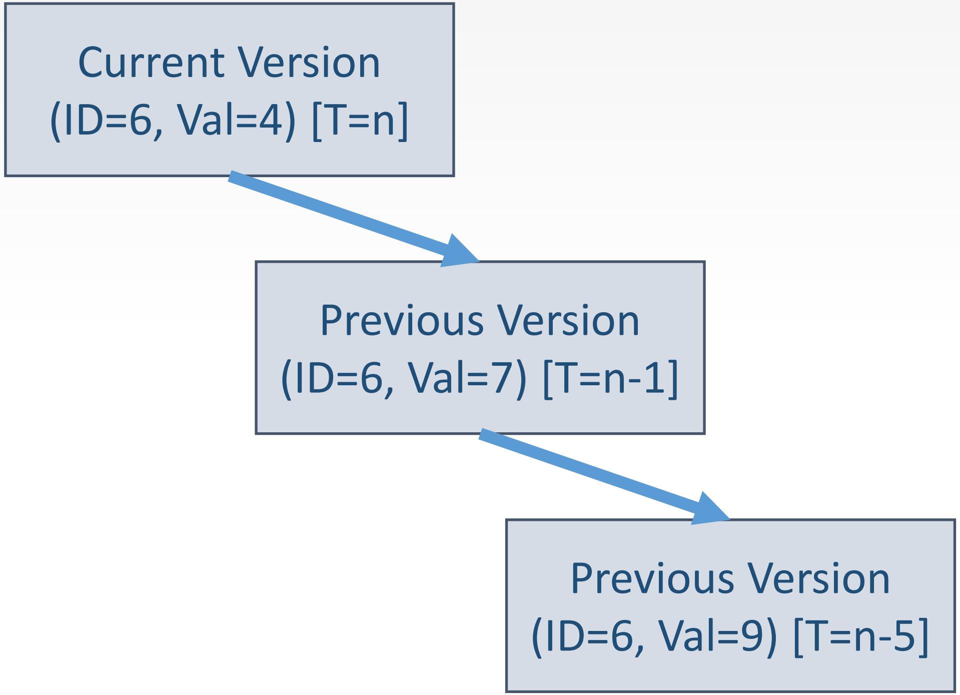
Optimistic Concurrency

- Uses row versioning to prevent concurrency conflicts
- Fewer locks are needed, so blocking is reduced
  - Readers no longer block writers
  - Writers no longer block readers
  - Writers still block writers
- Uses a *version store* to do this
- Version Store lives in tempdb
- Remember, it's OPTIMISTIC!

Row Versioning

- Whenever a row is updated, previous version is stored in the version store
- New version of row has a pointer to the previous version
- Versions are stored for as long as operations exist that might need them
  - All versions of rows modified by a transaction must be kept for as long as that transaction is open

Row Versioning

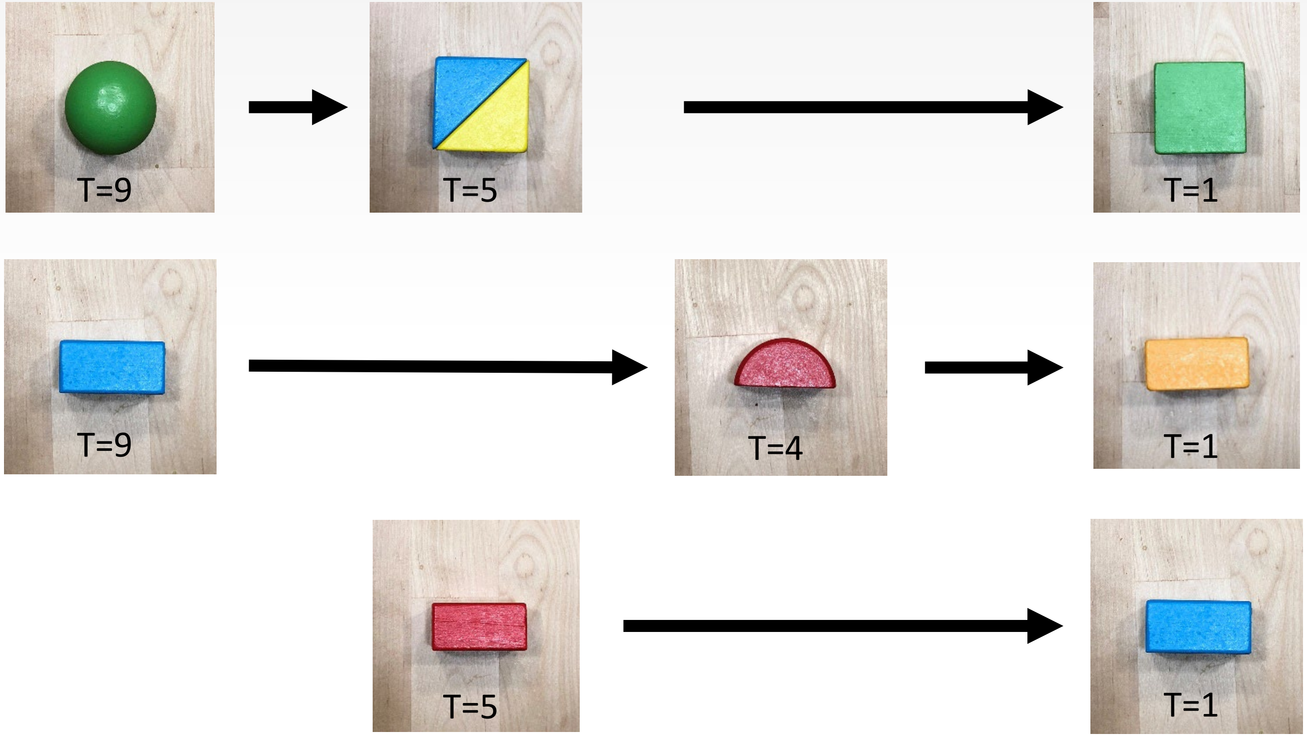


Current Version
(ID=6, Val=4) [T=n]

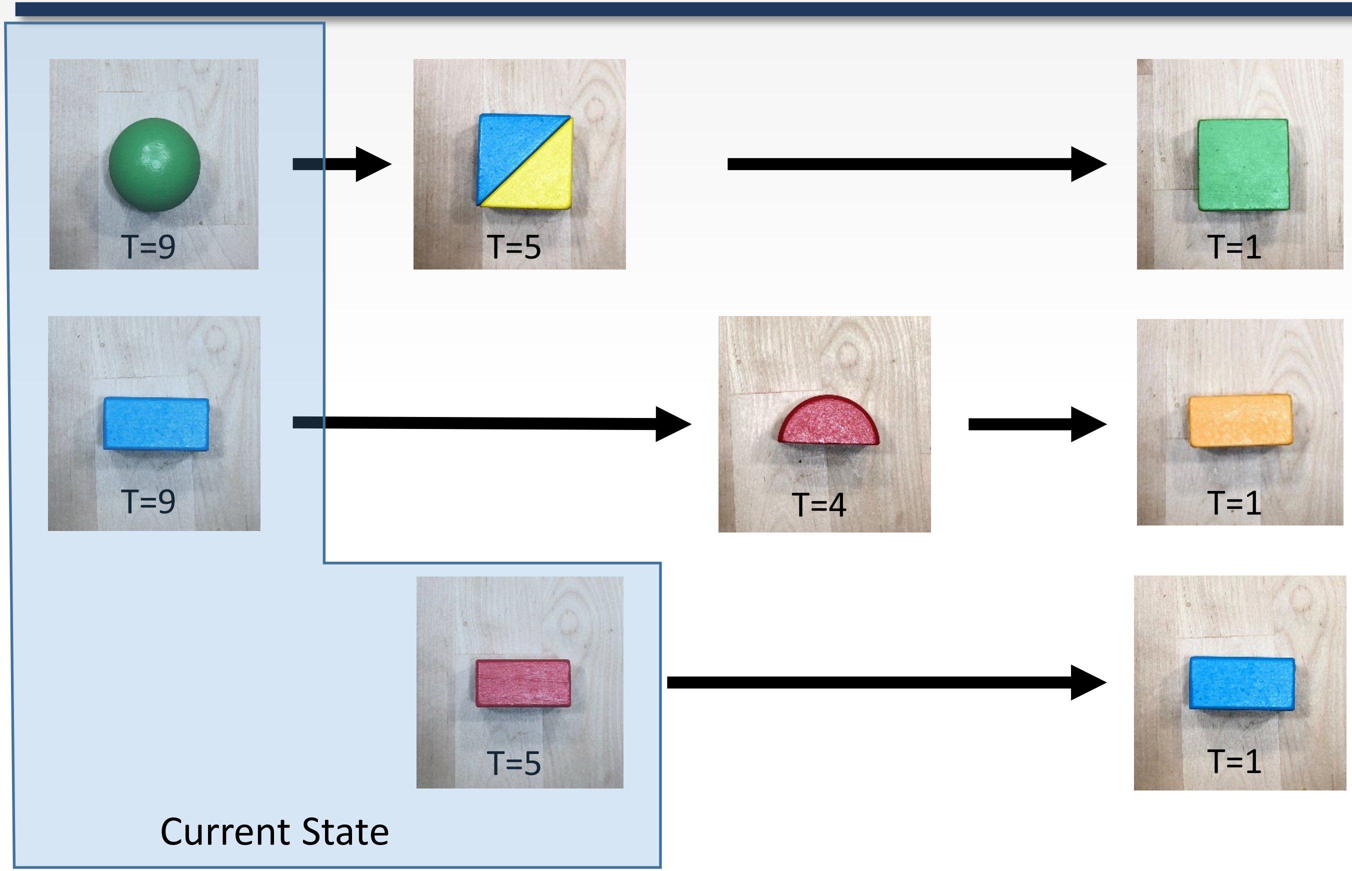
Previous Version
(ID=6, Val=7) [T=n-1]

Previous Version
(ID=6, Val=9) [T=n-5]

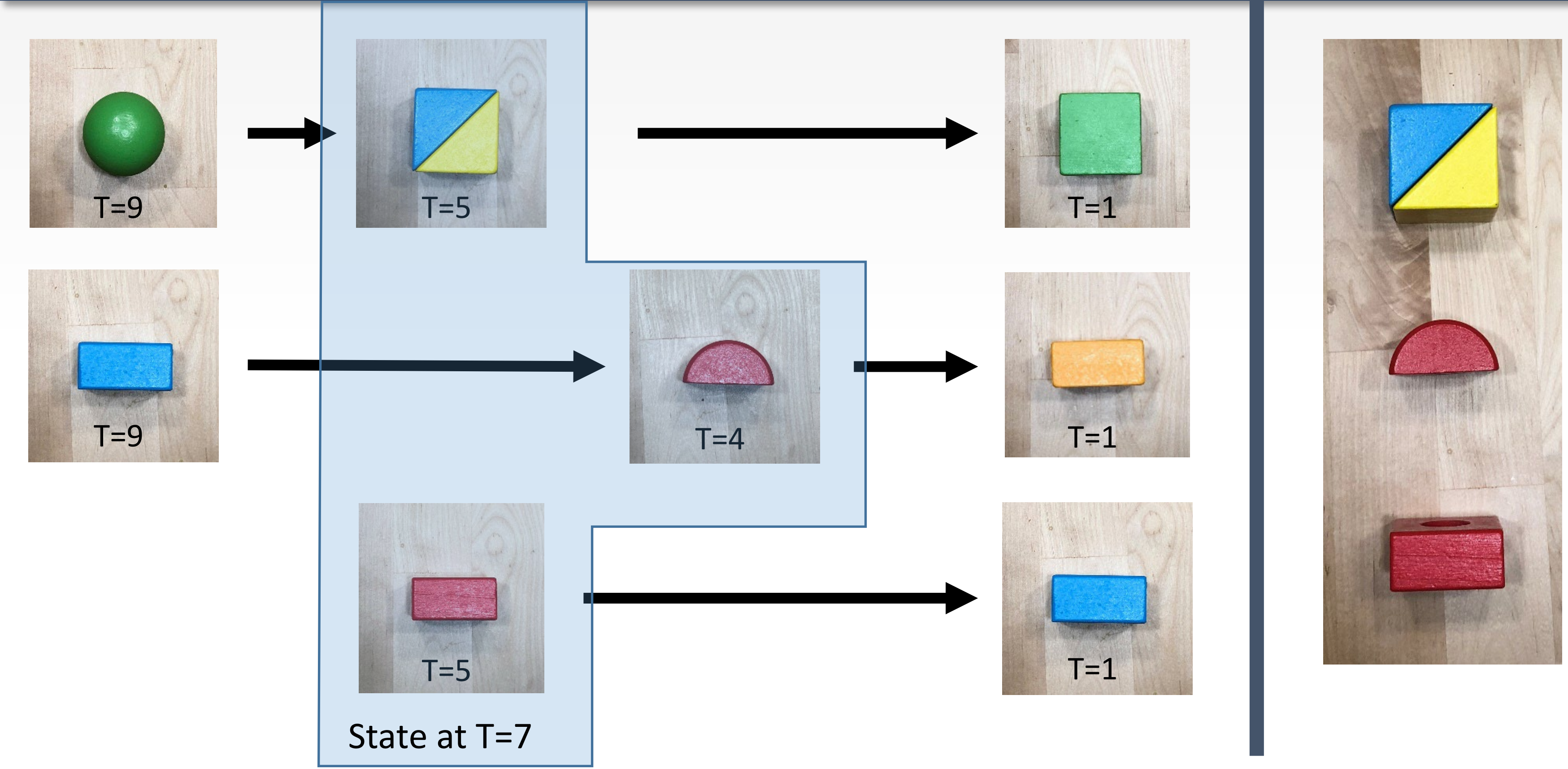
Row Versioning



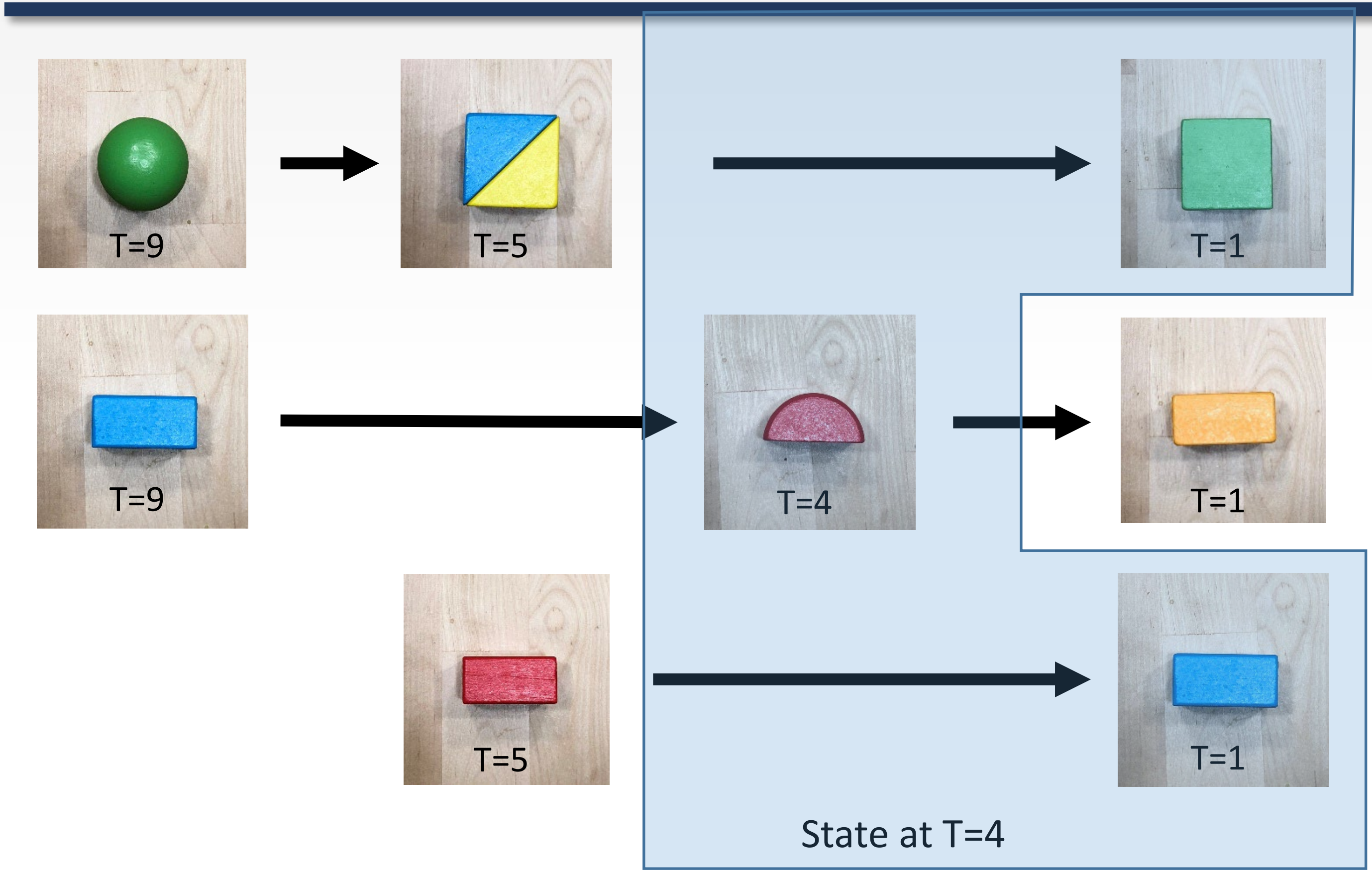
Row Versioning



Row Versioning



Row Versioning

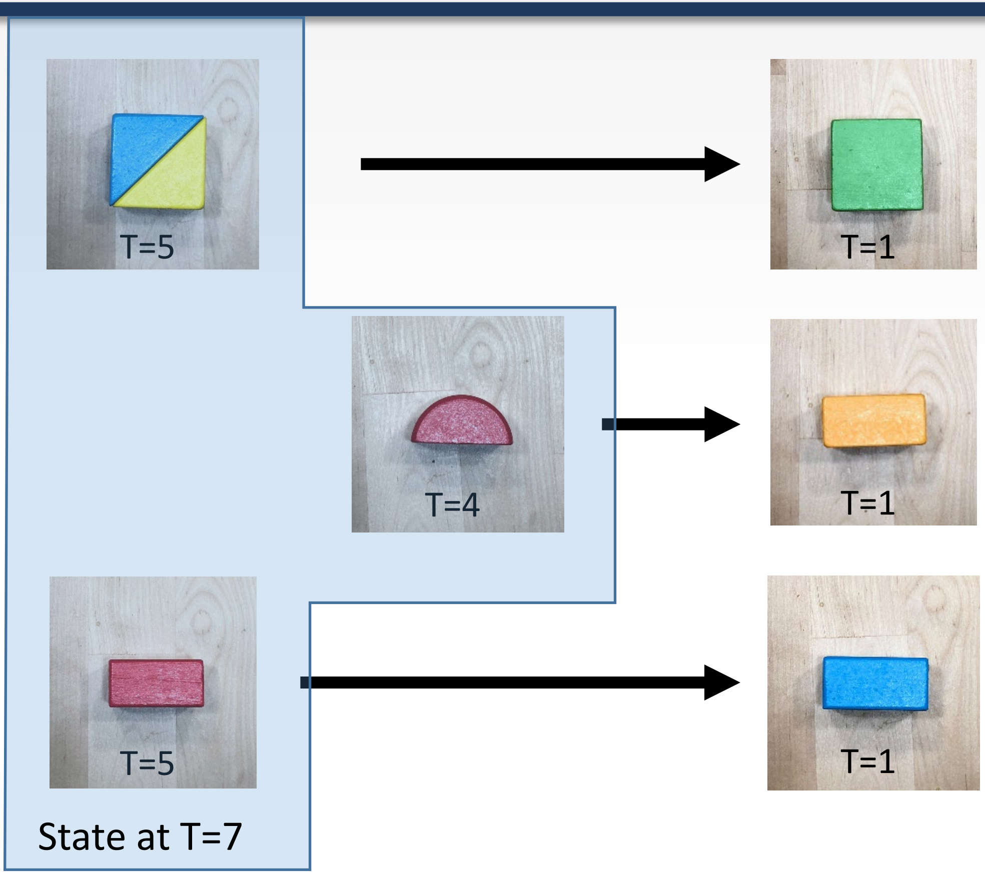


Read Committed Snapshot

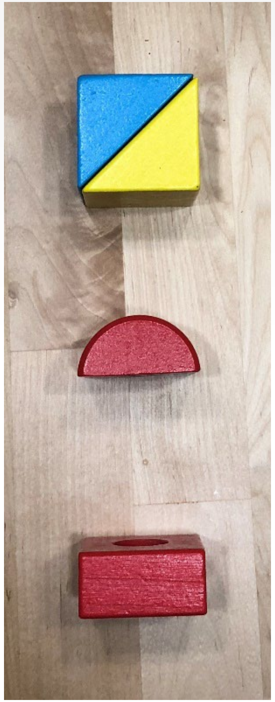
Dirty	Nonrepeatable	Phantom
No	Yes	Yes

- Same guarantees as READ COMMITTED, just an optimistic implementation
- **Statement-level** snapshot isolation
  - Queries will see the most recent committed values as of the beginning of that statement (not the transaction)

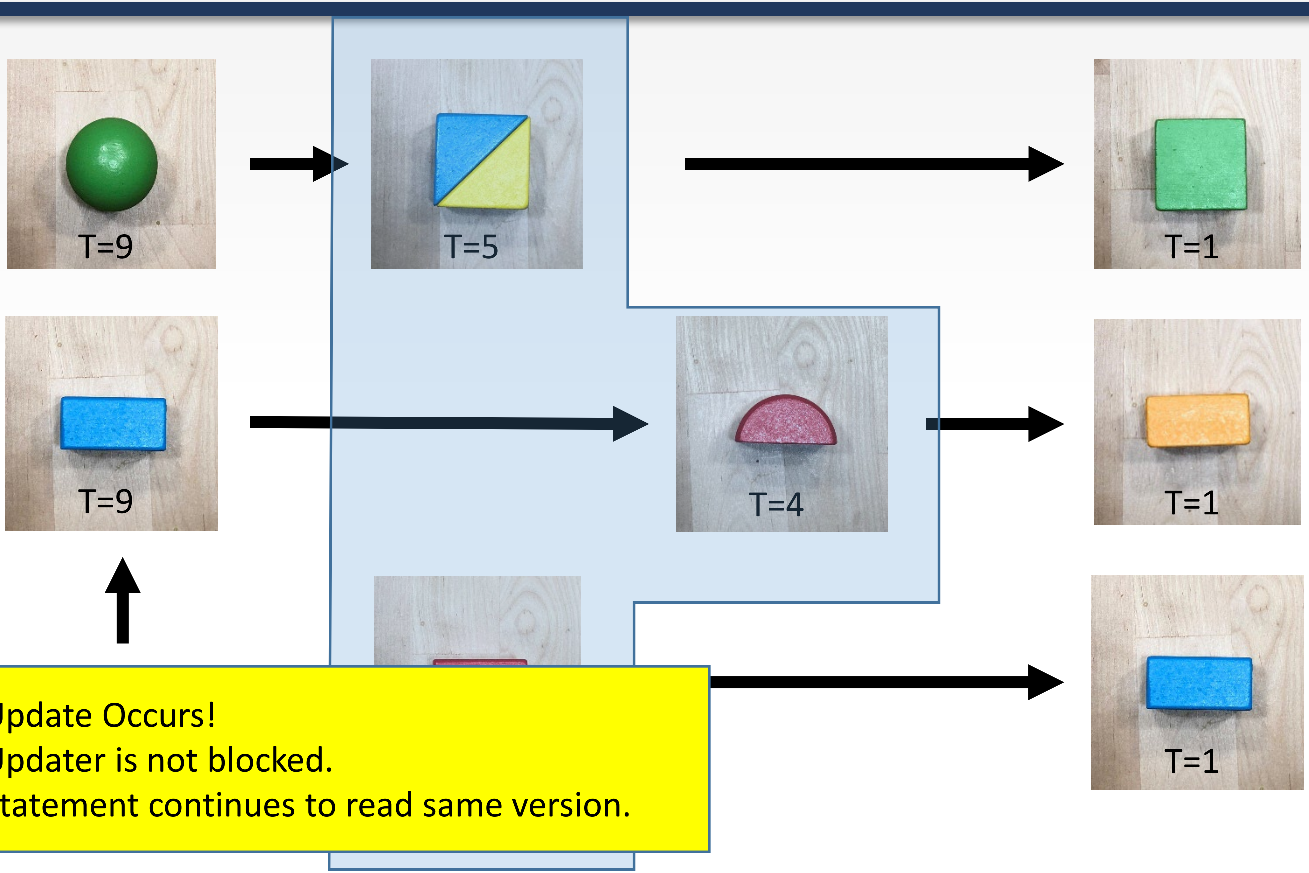
Read Committed Snapshot



Statement sees this:



Read Committed Snapshot



Update Occurs!
Updater is not blocked.
Statement continues to read same version.

Statement sees this:

Read Committed Snapshot

Dirty	Nonrepeatable	Phantom
No	Yes	Yes

```
ALTER DATABASE <DB_NAME>
SET READ_COMMITTED_SNAPSHOT ON
[WITH (NO_WAIT | ROLLBACK IMMEDIATE)];
```

- When enabled, RCSI becomes the default isolation level for this database.
- Command will block if DB has other connections
  - NO_WAIT will prevent blocking and just fail instead
  - ROLLBACK_IMMEDIATE will rollback other transactions

Demo

Snapshot

Dirty	Nonrepeatable	Phantom
No	No	No

- Same guarantees as SERIALIZABLE, just an optimistic implementation
- **Transaction-level** snapshot isolation
  - Queries will see the most recent committed values as of the beginning of that transaction (the first data read in it)

Snapshot

Dirty	Nonrepeatable	Phantom
No	No	No

```
ALTER DATABASE <DB_NAME>
SET ALLOW_SNAPSHOT_ISOLATION ON;
```

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

- First statement merely *allows* snapshot isolation

Snapshot Update Conflicts

- Process 1 reads data in a transaction, does not commit
- Process 2 reads/updates same data, does not commit
- Process 1’s snapshot does not see Process 2’s update
- Process 1 tries to update, gets blocked
- As soon as Process 2 commits, Process 1 errors out
- This will raise error 3960 on process 1

https://flic.kr/p/4WHW81

Demo

Azure SQL Database

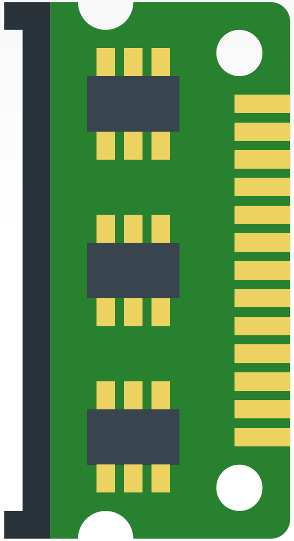
- Everything I’ve covered here behaves the same in Azure SQL Database
- *Exception:* RCSI is enabled by default

Agenda

- Concurrency Basics
- Isolation Levels
- In-Memory OLTP

Memory-Optimized Tables

- This could be its own presentation by itself
- Optimistic multi-version concurrency control
  - No locks required at any time
  - (Not even for data modification)
  - No waiting because of blocking!
  - No latches or spinlocks either
- Waits can still occur….
  - (Waiting for log writes to disk following a transaction)



Memory-Optimized Tables

- No existing row is ever modified
  - UPDATE creates a new version of a row
  - There may be multiple versions in play at once
- Transactions needing to read are presented with the correct version

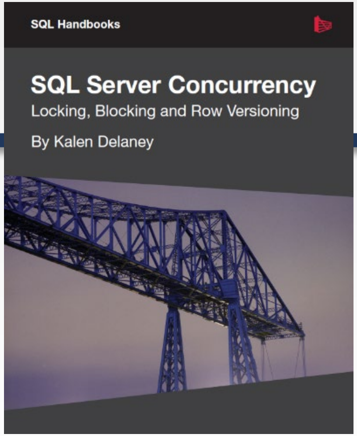
Memory-Optimized Tables

Begin Time	End Time	Data Columns	
10	<inf>	1	Red
10	<inf>	3	Green

Begin Time	End Time	Data Columns	
10	20	1	Red
10	20	3	Green
20	<inf>	3	Blue
20	<inf>	6	Pink

Now at time 20, let’s:
Delete (1, Red)
Update (3, Green) to (3, Blue)
Insert (6, Pink)

Resources


SQL Handbooks

SQL Server Concurrency

Locking, Blocking and Row Versioning

By Kalen Delaney

- Craig Freedman's posts on SQL Server Isolation Levels
  https://blogs.msdn.microsoft.com/craigfr/tag/isolation-levels/
- SQL Server Concurrency: Locking, Blocking, and Row Versioning (Kalen Delaney, Simple Talk Publishing)
- Myths and Misconceptions about Transaction Isolation Levels
  http://www.sqlpassion.at/archive/2014/01/21/myths-and-misconceptions-about-transaction-isolation-levels/

Questions?

@bobp.bsky.social

@sqlbob

bob@bobpusateri.com

bobpusateri.com

bobpusateri