BRENT OZAR
UNLIMITED®

Solving LCK Waits with Isolation Levels & Replicas

The RCSI/SI version store, Always On Availability Groups, and my personal arch nemesis: replication.

2.4 p1

3 ways to fix concurrency issues

1. Have enough indexes to make your queries fast, but not so many that they slow down DUIs, making them hold more locks for longer times.
   (I cover this in Mastering Index Tuning.)
2. Tune your transactional code.
   (I cover this in Mastering Query Tuning.)
3. Use the right isolation level for your app's needs.
   (This module focuses on this topic.)

Agenda

Isolation levels:
like readable secondaries in TempDB

When that's not enough:
real readable secondaries

2.4 p3

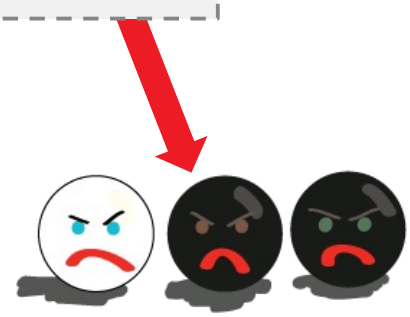
SQL Server defaults to pessimistic

2.4 p4

You've got 3 marbles

An update starts

```
UPDATE dbo.Marbles
SET Color='White'
WHERE MarbleID=2;
```



2.4 p6

It takes out locks

```
UPDATE dbo.Marbles
SET Color='White'
WHERE MarbleID=2;
```

2.4 p7

Before it completes, a read starts

```
UPDATE dbo.Marbles
SET Color='White'
WHERE MarbleID=2;
```

```
SELECT COUNT(*)
FROM dbo.Marbles
WHERE Color='Black'
```

2.4 p8

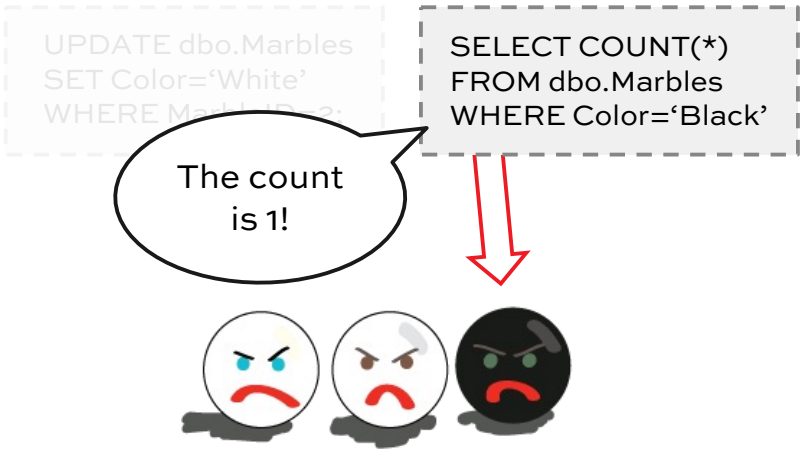
The writer blocks the reader

```
UPDATE dbo.Marbles
SET Color='White'
WHERE MarbleID=2;
```

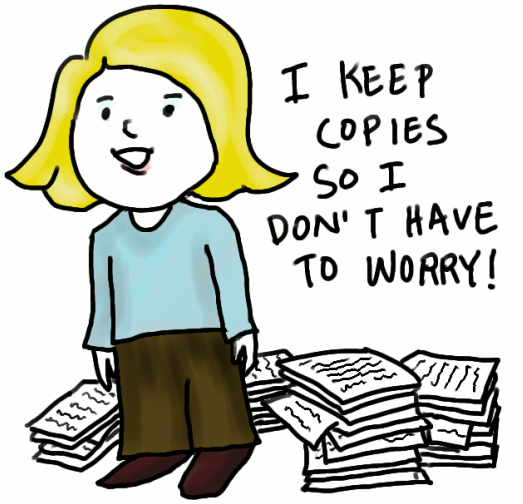
```
SELECT COUNT(*)
FROM dbo.Marbles
WHERE Color='Black'
```

2.4 p9

The writer finishes, releases locks



You can choose optimistic locking

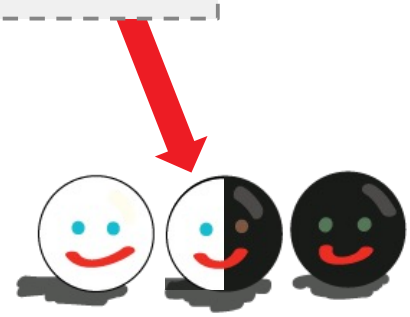


2.4 p11

You've got 3 marbles

An update starts

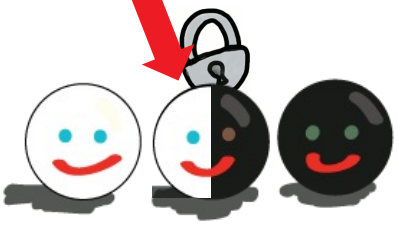
```
UPDATE dbo.Marbles
SET Color='White'
WHERE MarbleID=2;
```



2.4 p13

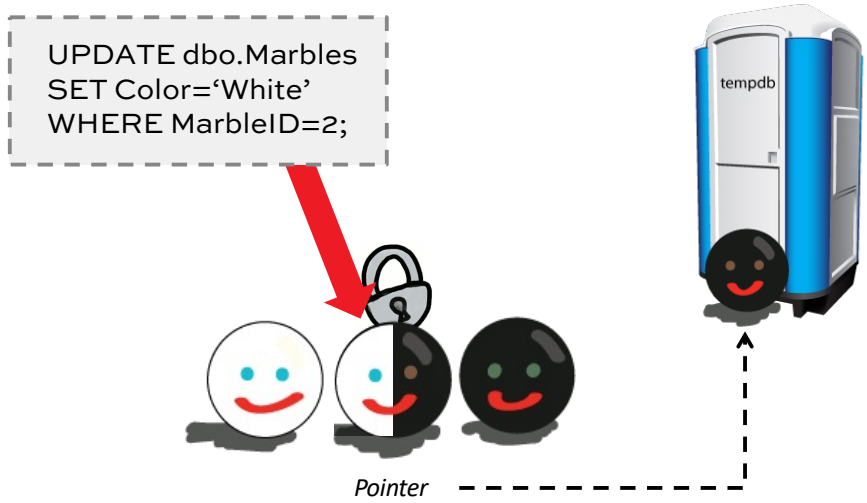
The writer still takes out locks

```
UPDATE dbo.Marbles
SET Color='White'
WHERE MarbleID=2;
```



2.4 p14

SQL copies the prior version into the version store



The version store lives in TempDB

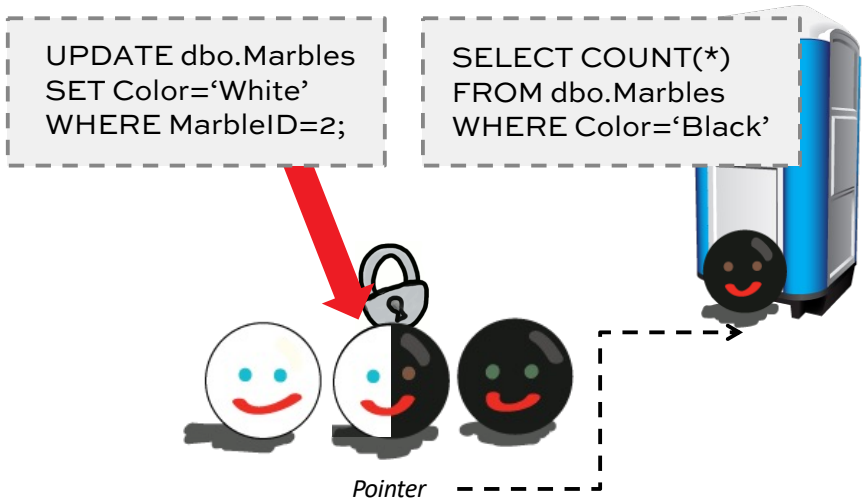
It's completely system-managed:
you don't control growth or purging.

This does impact TempDB performance.

In SQL Server 2019, you can also choose
to put it in the user database itself.
(Accelerated Database Recovery)

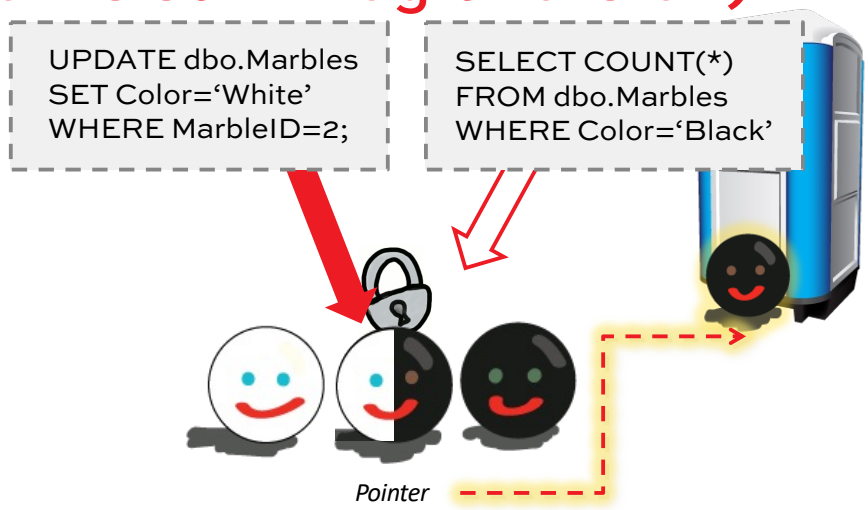
We'll touch on all this later.

A reader query starts

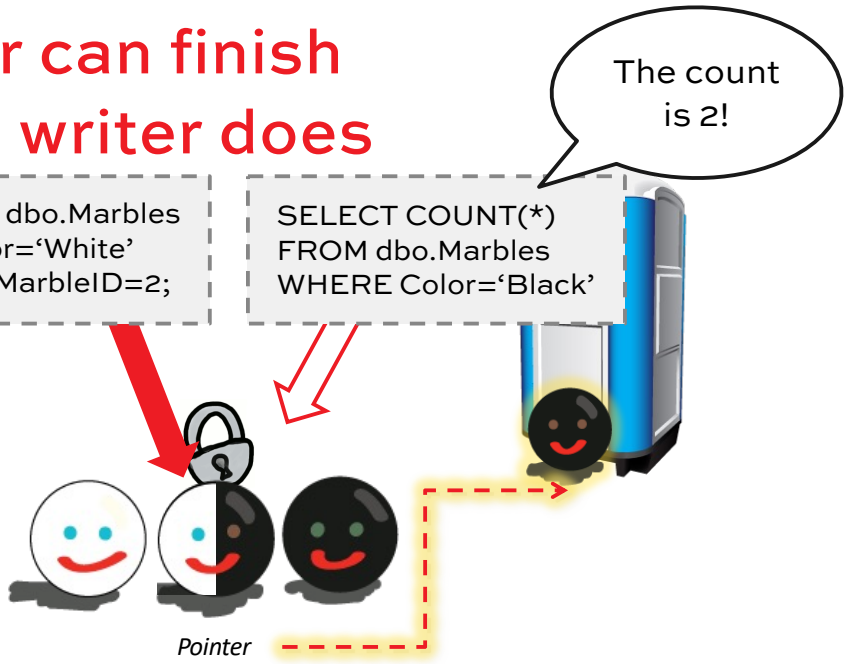


2.4 p17

The reader gets the old version (which is still a legit version)



The reader can finish before the writer does



2.4 p19

The reader finishes faster

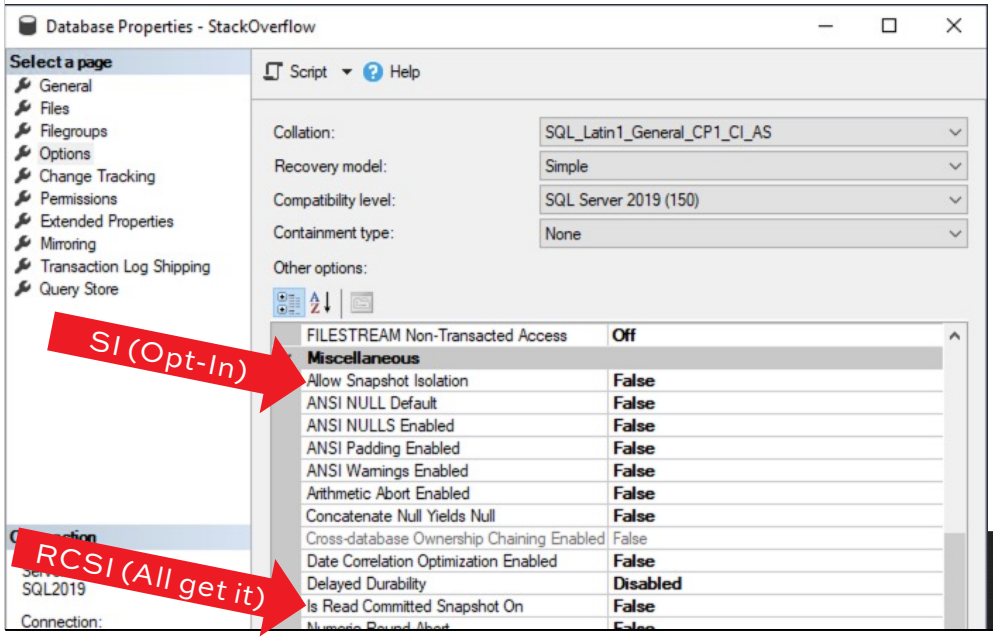
But could see different results depending on how quickly the other transaction committed and released its locks:

- Pessimistic locking: 1 black marble
- Optimistic locking: 2 black marbles

Two ways to implement: BrentOzar.com/go/rcsi

	Read Committed Snapshot Isolation (RCSI)	Snapshot Isolation
What does it do?	Changes the default isolation level for an entire database to optimistic locking	Allows individual transactions to use optimistic locking
What version does a statement read?	The most recent committed version	The committed version consistent with when the transaction began
Requires code changes	No	Yes
Requires testing	Yes	Yes

Changing it is super-easy



Choose RCSI when:

You're building a new app from scratch

An existing app was designed for Oracle or Postgres, and it just "also" works (slowly) on SQL Server (because it was probably designed & tested for MVCC/RCSI)

You have an existing app, and you can test the whole code base pretty confidently to make sure it's not affected by different results due to blocking

Choose snapshot isolation when:

You have a big code base that you can't test

You only have a handful of queries that are involved in the blocking

The queries are all readers (like reports)

You can change the reader queries to start with `SET TRANSACTION ISOLATION LEVEL SNAPSHOT`

With either RCSI or SI...

If your queries ask for dirty reads:

- WITH (NOLOCK)
- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

Then they'll still get dirty reads.

Query-level hints override everything.

You'll want to take those hints out of your queries after you implement RCSI or SI.

Preparing for the new overhead

Both RCSI and SI have two impacts on wherever the version store is kept (normally TempDB):

1. **Size:** long-running transactions mean the data in the version store may hang around longer than expected, so the file grows larger than it did before
2. **Performance:** they cause more disk reads & writes, and storage might have already been slow

You'll need to prepare for both & monitor them.

Preparing for the new overhead

You're gonna ask:

- "How much more space will TempDB need?"
- "How much activity will happen in TempDB?"

And I'm gonna ask:

- "How big are your transactions, as in, how much data are you changing in one transaction?"
- "How long will you hold the transactions open?"

And you're not gonna know.

How I advise clients

TempDB size: expect it to be ~25% of the size of your total databases, if not larger.

If you do batch loads in a single transaction (BEGIN TRAN / COMMIT), and they take more than 30 minutes today, it's probably not a good fit. Fix the batch loads to avoid transactions first.

If your TempDB storage is averaging 100ms or slower for writes, fix that first. Any additional workload here would be a bad idea.

If you can put TempDB on cheap local mirrored solid state storage, do it. 2TB SSDs just aren't a big deal.

Monitoring the new overhead

Perfmon counters in SQLServer:Transactions:

- Free Space in tempdb (KB)
- Version Store Size (KB)
- Longest Running Transaction Time
  (but this one only reports data when you're actually using RCSI or SI)

Learn more:
https://www.brentozar.com/archive/2015/03/monitoring-snapshot-isolation-with-perfmon-in-sql-server-video/

Troubleshooting the version store

"Why is the version store growing?"
Run sp_WhoIsActive or sp_BlitzWho and look for open transactions.

"Which database is using version store space?"
Diagnostic query: https://sqlrambling.net/2017/11/14/monitoring-the-version-store-in-sql-server-2017/

"Why can't SQL Server clean it out?"
Read this post from Microsoft:
https://blogs.msdn.microsoft.com/sqlserverstorageengine/2008/12/31/managing-tempdb-in-sql-server-tempdb-basics-version-store-growth-and-removing-stale-row-versions/

Or, put it in the user database.

SQL Server 2019's Accelerated Database Recovery can put the version store in the user database files.

Also enables near-instant rollbacks because the original versions of rows are already in the data file, and don't need to be pulled from the log file.

Learn more:
https://www.mssqltips.com/sqlservertip/5971/accelerated-database-recovery-in-sql-server-2019/

2.4 p31

But it has big drawbacks.

Your user database data files can grow large quickly since they hold the version store, on all replicas.

The differential backups can grow too.

Your user database storage is under more workload, and you may not be able to use fast local SSDs like you do TempDB.

Monitoring & troubleshooting practices aren't well-defined yet.

When isolation levels aren't enough

Real Readable Replicas

2.4 p33

Replicas need 2 connection strings

1. Writes and real-time reads (your current default)
   (Extremely expensive/difficult to scale out)

2. Reads that can tolerate older data
   (point at readable secondaries)

2.4 p34

SQL Server can't offload reads.

Your developers have to understand when they need to query the readable secondary tier.

Your app just sees two different SQL Servers.
(Yes, this is more work for ORMs, but that's why this is a last resort.)

When you get started or do development, they can all point to the exact same SQL Server.

But as you implement readable replicas, then you can point them at different groups of servers later.

2.4 p35

3 ways to get readable replicas

1. "I'm cheap, have a lot of read-only queries, and they could live with a snapshot of data as of this morning."
   Log shipped secondaries
2. "I need nearly up-to-date data, and I absolutely require different indexes or data on each replica."
   Transactional replication
3. "I need nearly up-to-date data, and I need to offload backups or CHECKDB, and I'm willing to pay for Enterprise Edition everywhere."
   Always On Availability Groups

* But #2 and #3 both require separate connection strings for read-only reports, just like AGs.

2.4 p36

Log shipped secondaries

The idea: you're already taking log backups anyway

Just start restoring them onto another server

No performance overhead on the primary

Just use RESTORE WITH STANDBY
to leave the database in a readable state

More info & gotchas:
https://www.brentozar.com/archive/2015/01/reporting-log-shipping-secondary-standby-mode/

2.4 p37

Benefit: can be Standard Edition

Save a ton of money on licensing
(because all queried replicas have to be licensed)

The primary is Enterprise Edition? No worries: since SQL 2016 SP1, Standard Edition has had the same developer features, like:

- Partitioning
- Columnstore
- Compression

2.4 p38

Downside A: restores eject you

Users can't be in a database when you restore the latest set of transaction log backups

Two ways to work around that:

- Write a job to check for no user activity for 30-60 seconds, and then restore the latest logs, or
- Only run restores 8PM-6AM, and all day long, people see a readable copy of data as of 6AM

2.4 p39

Downside B: read-only data files

The log shipped database is exactly the same as the primary: same data, same indexes

You can't add custom indexes

You can't delete or change data

If you need those, it's time for option #2: transactional replication

Let's talk option #2

1. "I'm cheap, have a lot of read-only queries, and they could live with a snapshot of data as of this morning."
   Log shipped secondaries

2. "I need nearly up-to-date data, and I absolutely require different indexes or data on each replica."
   Transactional replication

3. "I need nearly up-to-date data, and I need to offload backups or CHECKDB, and I'm willing to pay for Enterprise Edition everywhere."
   Always On Availability Groups

* But #2 and #3 both require separate connection strings for read-only reports, just like AGs.

2.4 p41

Replication is a good fit when...

You're willing to custom code your own replication stored procedures to deal with deletes, conflicts

You're good at learning things yourself in production: a lot of the problems aren't well-documented

Your servers are two different versions, or Azure SQL DB

The number of tables is relatively low (say, <100)

The table structures don't change

The data doesn't completely change, quickly

2.4 p42

Replication is not-so-good when

Table structures change frequently:
new columns, changed columns, normalized

The data changes en masse, like
“we reloaded two years of sales last night”

You only have 1 person on the team who knows it
(or nobody on the team)

“I never want to work with replication again.”

Most people who work with replication

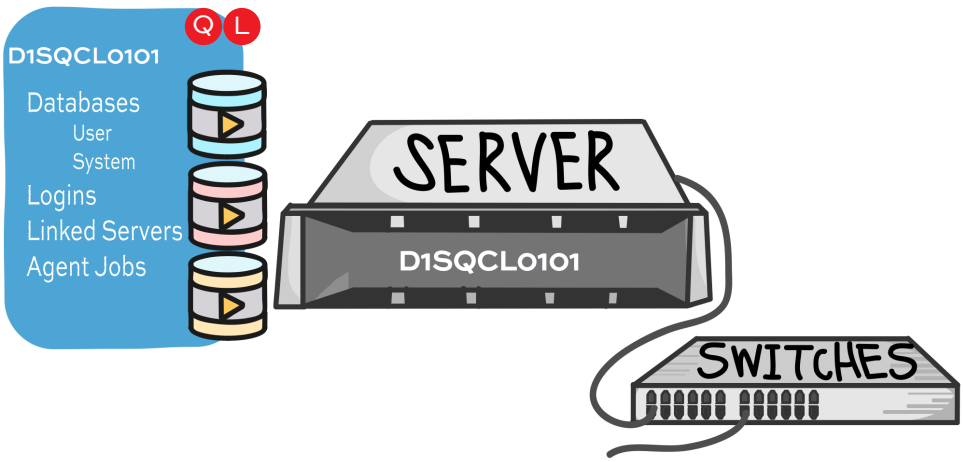
2.4 p44

Let's talk option #3

1. "I'm cheap, have a lot of read-only queries, and they could live with a snapshot of data as of this morning."
   Log shipped secondaries
2. "I need nearly up-to-date data, and I absolutely require different indexes or data on each replica."
   Transactional replication
3. "I need nearly up-to-date data, and I need to offload backups or CHECKDB, and I'm willing to pay for Enterprise Edition everywhere."
   Always On Availability Groups

* But #2 and #3 both require separate connection strings for read-only reports, just like AGs.

2.4 p45



Normal

Nodes: 1
Instances: 1
Licensed cores: 8

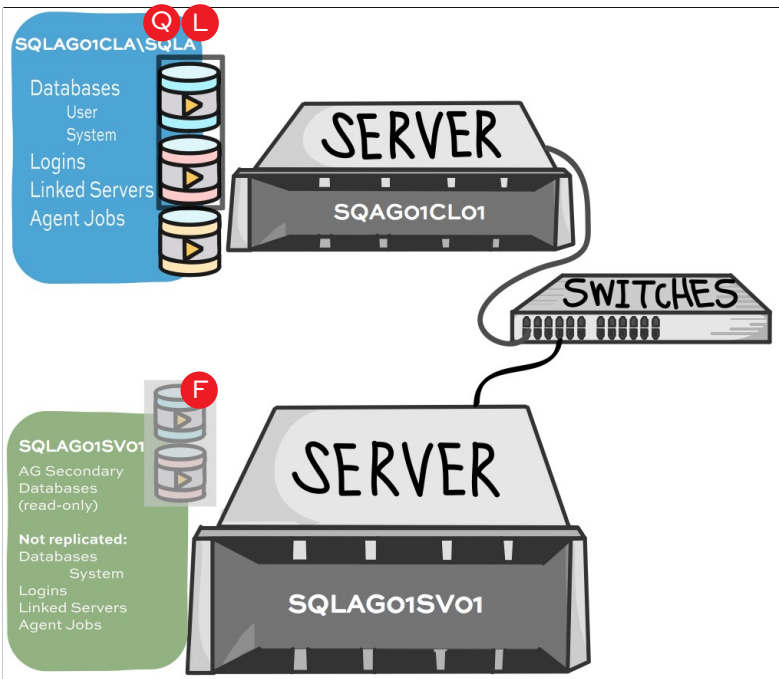
Simple local installation on a standalone server.

Connected to the network so you can query it.

Licenses required: However many cores are in the physical node, D1SQCL0101.

This instance is Queried.

This instance is Licensed.



AlwaysOn AG

Nodes: 2
Instances: 2
Licensed cores: 8 EE w/SA, 16 w/o SA.

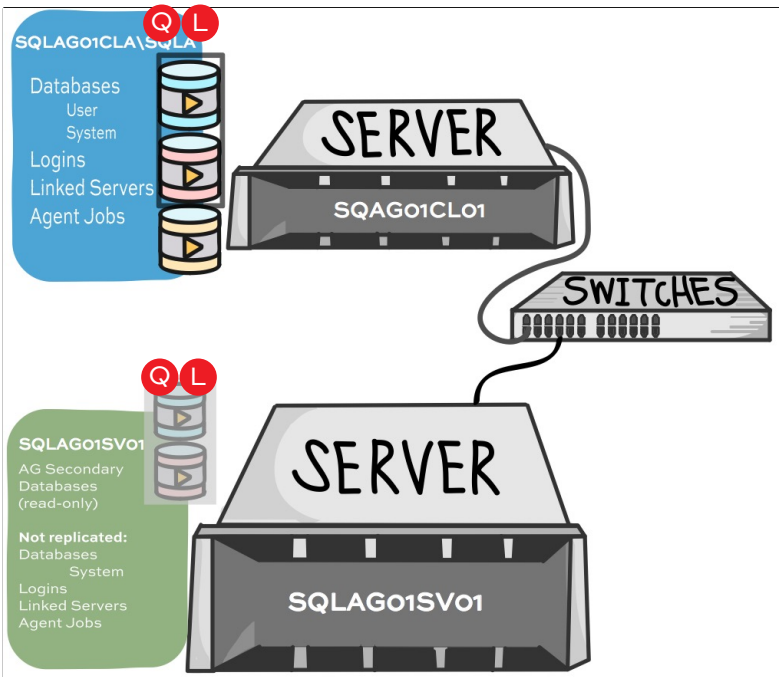
One primary server that we pay for.

One secondary server. Doesn't matter whether it's sync or async, and in the same data center or elsewhere.

As long as we have SA and we don't query, back up, or DBCC the secondary, there's no charge for it.

Gains automatic page repair, too!

2.4 p47



AlwaysOn AG

Nodes: 2
Instances: 2
Licensed cores: 16 EE.

The instant we start querying our replica, or running backups or DBCC on it, then we have to license it.

This is the scenario we're covering today: using AGs to scale out reads to one (or multiple) replicas.

2.4 p48

Small: StackOverflow (circa 2012)

Web site network for questions & answers:
StackOverflow.com, DBA.StackExchange.com, etc.

High traffic: >8mm visitors and >75mm pages/month

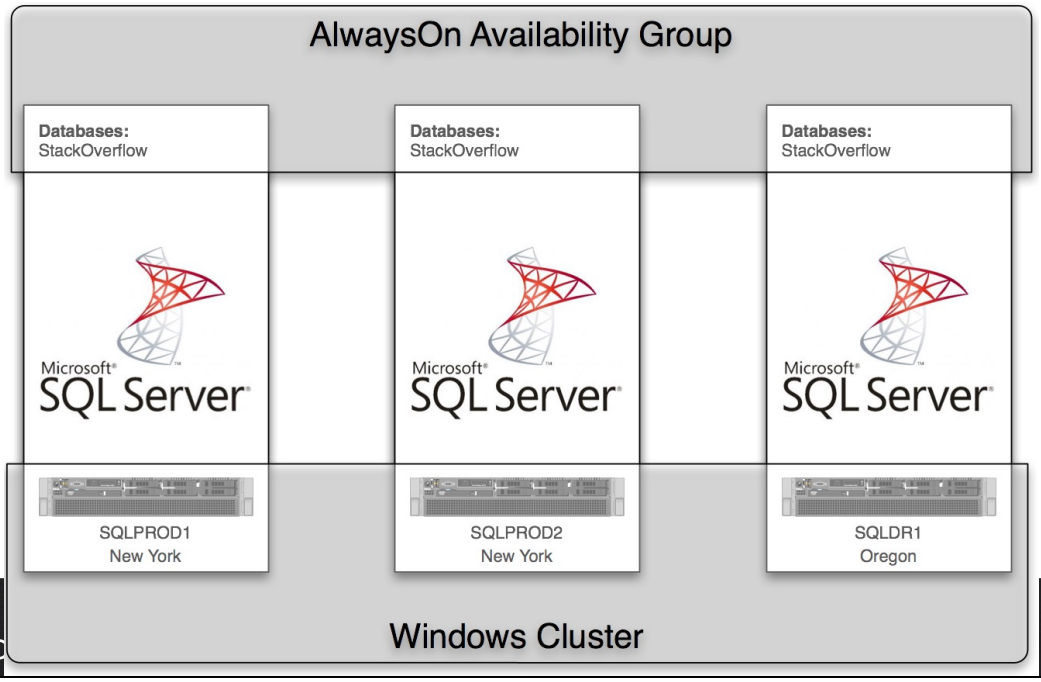
No dedicated full time DBAs, but several very hard-core sysadmins that write their own monitoring tools

Requirements:

- Easy high availability and disaster recovery
- Some data loss and downtime tolerable
- Avoid expensive shared storage
- **Scale-out is a bonus for things like API read requests**

2.4 p49

StackExchange (circa 2012)



2.4 p5C

Large: Discovery Education

Complex 24/7 web application

Microsoft SQL Server, MySQL, other technologies

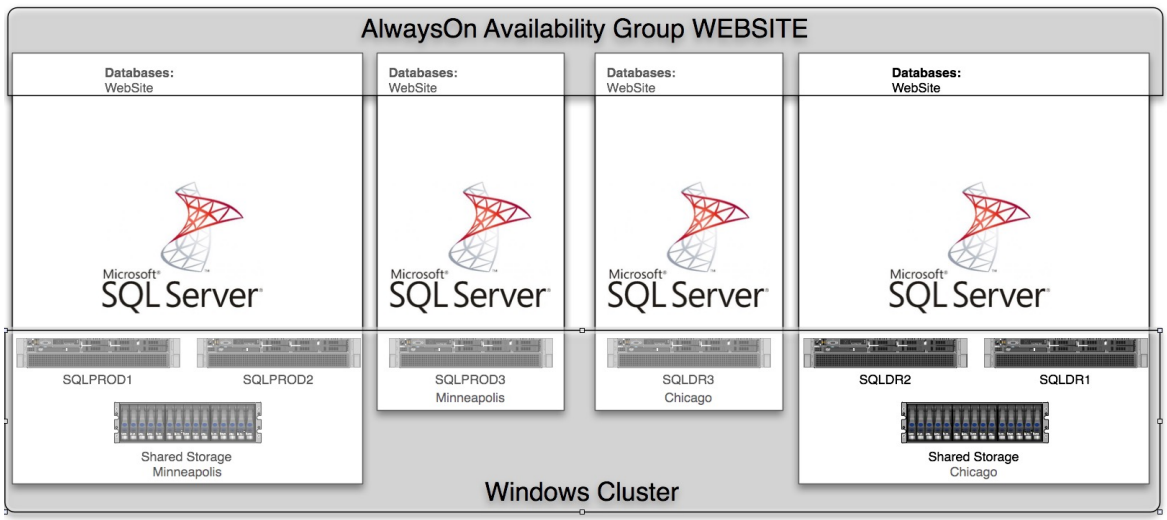
Team of dedicated DBAs comfortable with failover clustering and shared storage

Requirements:

- Automatic failover between servers
- Manual failover between datacenters with some data loss, but need full power in either datacenter
- Ability to serve traffic from two datacenters
- **Wanted simpler reporting processes**

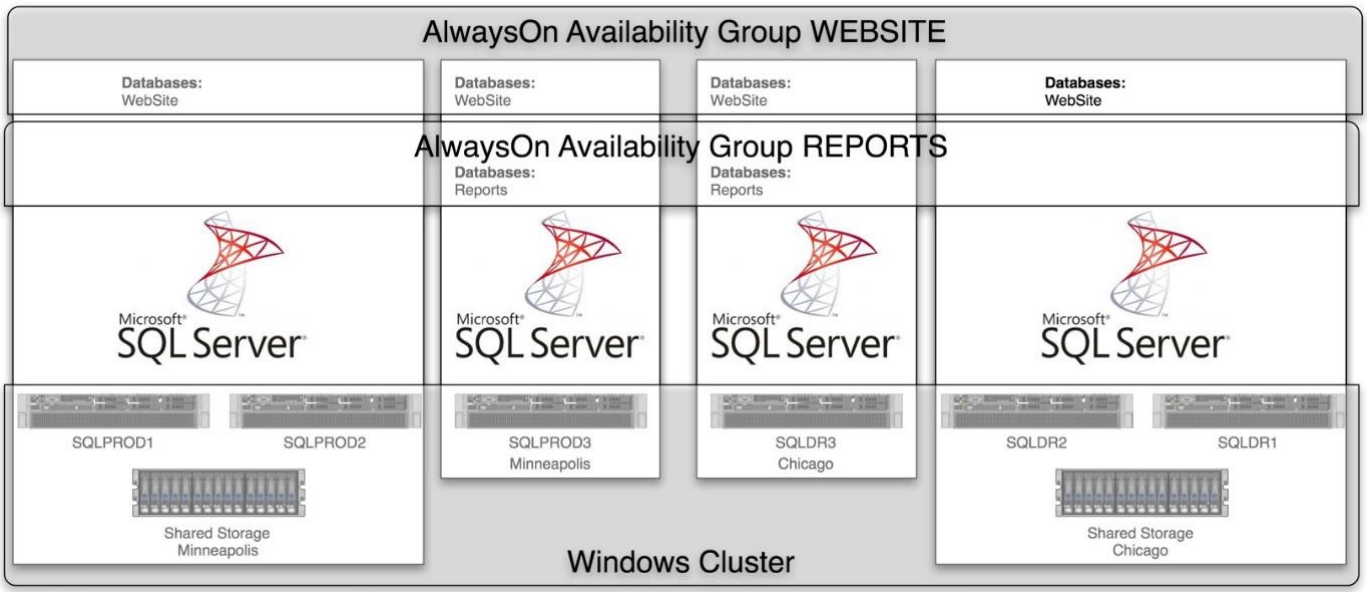
2.4 p51

The “simple” version



2.4 p52

Well actually, multiple AGs in 1 cluster, and each one could have a different primary at any given time.



Microsoft invested over time

	Basic Availability Groups (BAGs)	Availability Groups (AGs)	Distributed Availability Groups (DAGs)
Available in:	2016 Standard	2012, 2014, 2016 Enterprise	2016 Enterprise
# of databases	1	Many	Many
# of replicas	1	6-8	Many
Can query, back up on the replicas	NO	Yes	Yes
Replica list can be modified	NO	Yes	Yes
Supports either synchronous or async	Yes	Yes	Yes
Cross-database transactions	N/A	2012, 2014 – No 2016 – Yes*	

2.4 p54

No matter what replica technology you use...

Performance tuning gets harder.

2.4 p55

Uptime monitoring is hard...

Is the primary accepting writeable connections?

Are all of the read-only replicas up and running?

Is load distributed between replicas?

Are any of the replicas running behind?

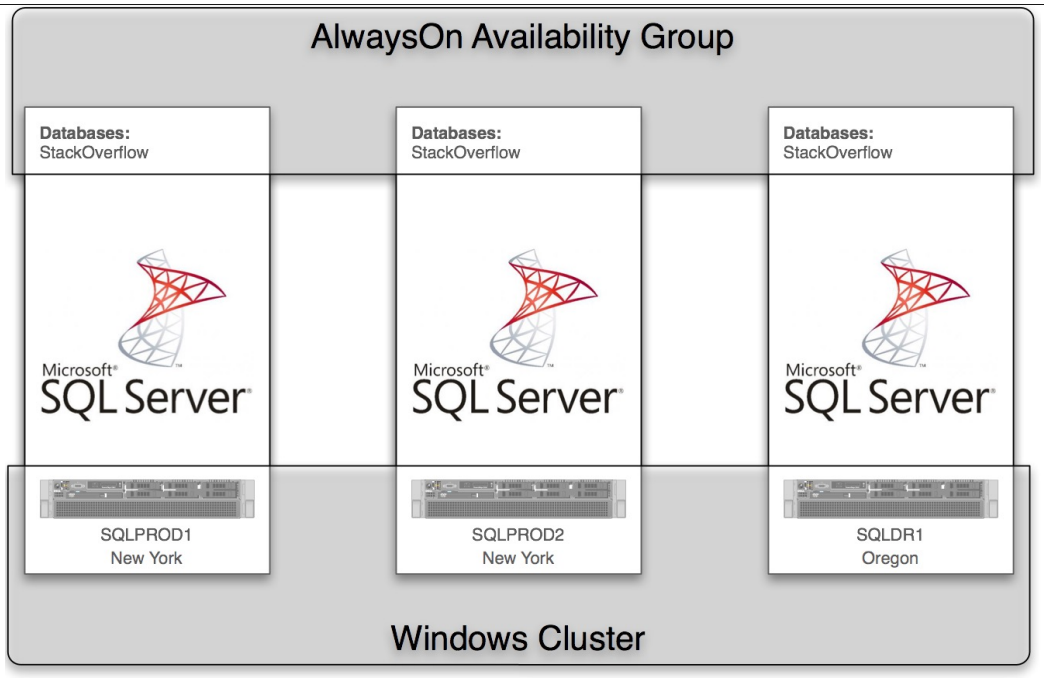
Does our 3rd party monitoring app
cover our chosen replication tool well?

But performance monitoring is HARD.

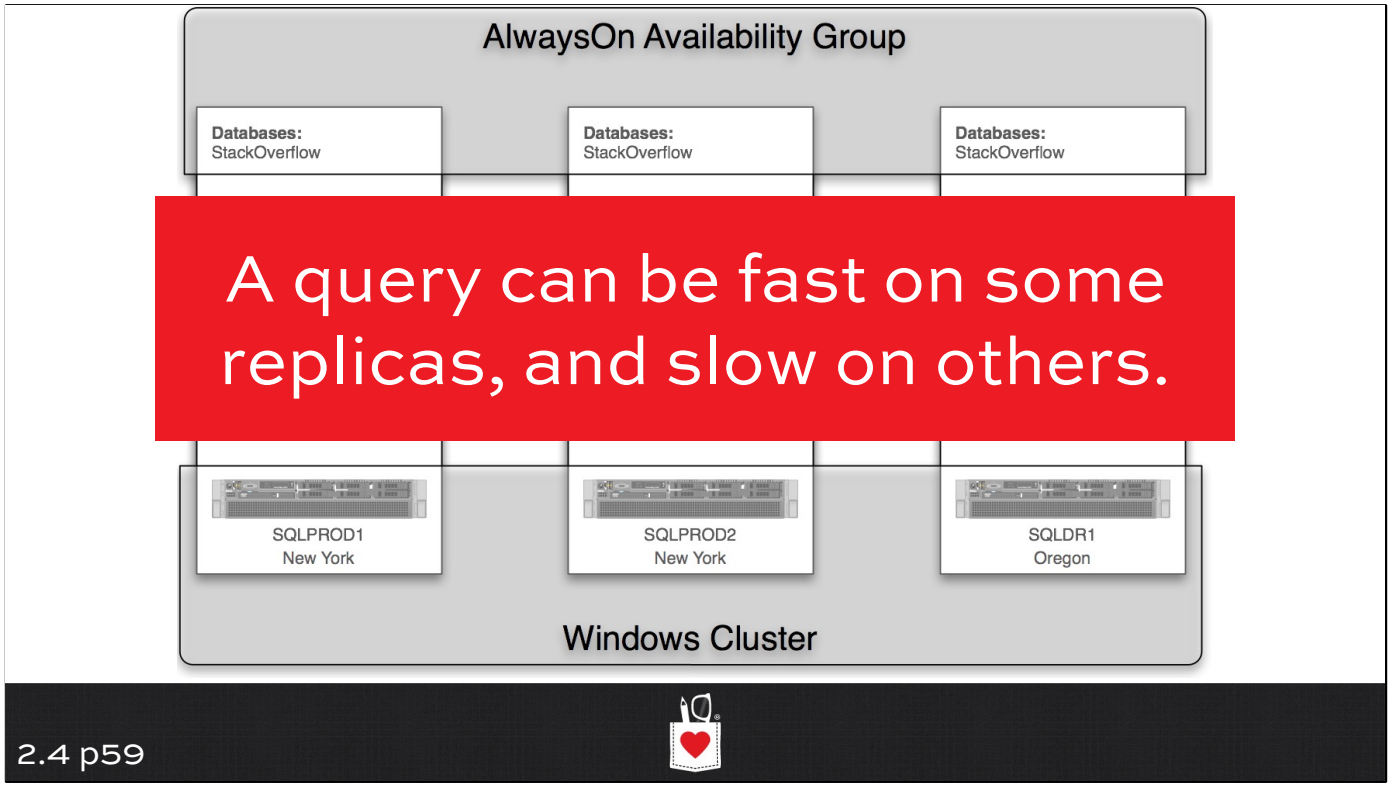
Each replica has its own:

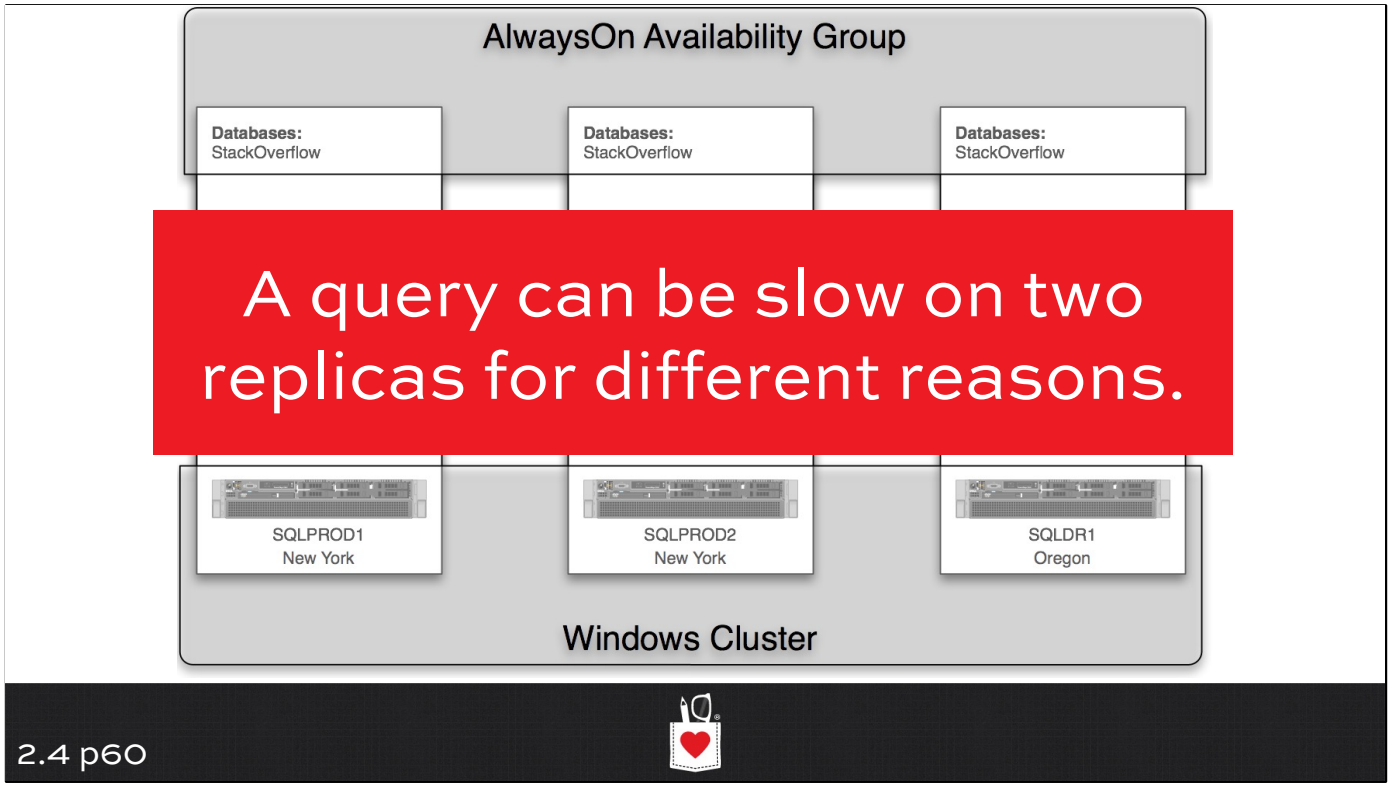
- DMV data (index usage, plan cache)
- Statistics
- Query plans
- Hardware
- Different databases

2.4 p57



2.4 p58





Build AGs into your routine

Which replica are we having the problem on?

Can we replicate the same problem on other replicas?

Can we compare performance between two replicas to help identify why one of them is slower?

Are any environment differences between replicas contributing to the problem? (Different hardware, sp_configure settings, statistics, performance loads)

Recap

2.4 p62

3 ways to fix concurrency issues

1. Have enough indexes to make your queries fast, but not so many that they slow down DUIs, making them hold more locks for longer times.
   (I cover this in Mastering Index Tuning.)
2. Tune your transactional code.
   (I cover this in Mastering Query Tuning.)
3. Use the right isolation level for your app's needs, and only when that isn't enough, scale out.

RCSI & SI: like a TempDB replica.

Optimistic concurrency aka multi-version concurrency control (MVCC): store row versions to avoid blocking.

Read Committed Snapshot Isolation:
everybody uses the version store by default.

Snapshot Isolation: defaults to pessimistic, but individual queries can opt into using the version store:
`SET TRANSACTION ISOLATION LEVEL SNAPSHOT`

Both cause new overhead in TempDB.