What is "Latch Contention”?

And why should I care?

Tom Bascom, White Star Software

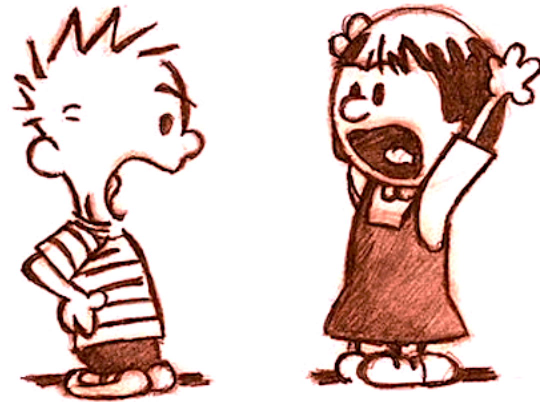
Thursday 9:00am

Abstract: Come to this session to learn what a latch is, how it enables reliable and highly concurrent access to your database and why you should be wary of "too many cores"!

White Star
software

What is “Latch Contention”?

And why should I care?



Tom Bascom, White Star Software
tom@wss.com

White Star software

What is a “latch” anyway?

- A lock on shared memory
- Also known as a MUTEX (Mutual Exclusion) Lock
- Or a “spin lock”
- Latches must be implemented as an *atomic* operation and they protect activities that must behave as atomic actions
- At the machine code level this is often a “Test and Set” or a “Compare and Swap” instruction

White Star
software

Why are latches important?

- Latches coordinate access to shared memory
- The use of latches enables highly performant concurrent access
  - Alternate methods of locking shared memory (such as semaphores) are *much* slower
  - (Semaphores et al involve context switches, sleeping, and waking up)
- Because latches are shared:
  - They can be a source of contention and performance bottlenecks
  - They can also be a source of database crashes

White Star
software

When Might I See “Latch Contention”

- Usually not until after you have eliminated IO as a bottleneck
- The amount of memory in servers has grown enormously
  - Terabyte memory in servers is becoming common
  - Which means that it is often possible to fit your whole DB into -B
  - Even without doing that, the “working set” of a DB is usually much less than the full db size
- It is becoming increasingly common to see latch contention bottlenecks

White Star software

I’m just doing FIND NO-LOCK

- You still need to pay attention
- When you access data, whether to read or write, you access shared memory
- Latches keep shared memory consistent and have nothing to do with record locks
  - (although latches are used to keep the lock table consistent in support of record locks)

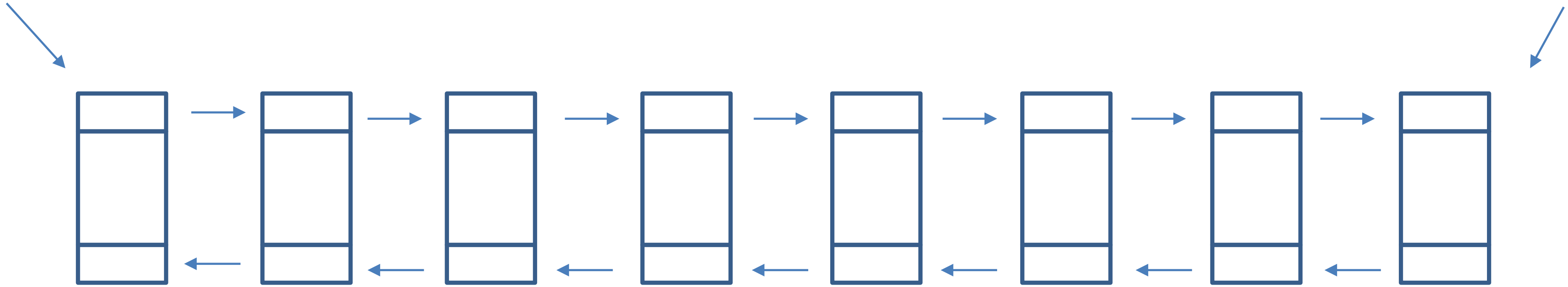
White Star software

An Example: The LRU Chain

- LRU = Least Recently Used
- This is how the DB Engine decides which block to replace when it needs to read a new block from disk
- Whenever any block is referenced it is moved to the head of the LRU chain (a ”linked list”)
  - This includes FIND NO-LOCK
- This results in unused blocks gradually moving to the tail of the chain and frequently used blocks staying towards the head

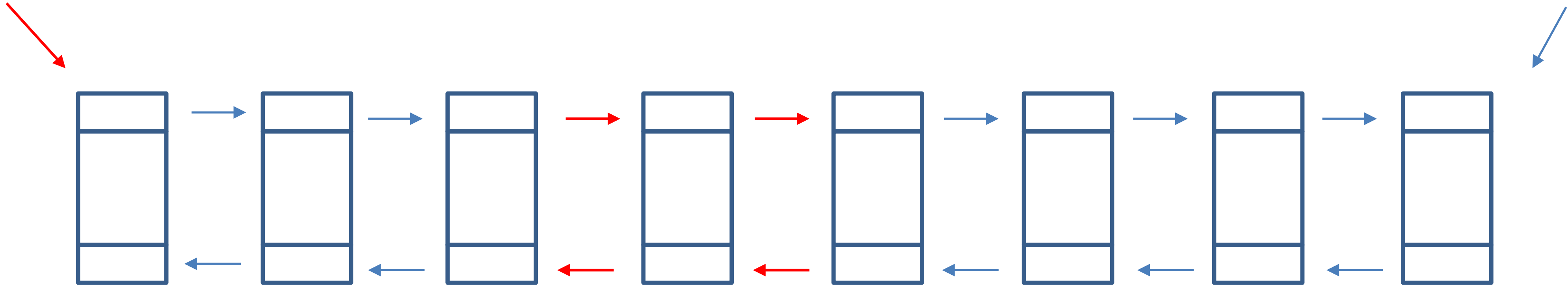
White Star
software

Updating the LRU Chain



White Star software

Updating the LRU Chain



LRU Chain

- To move an entry you need to update multiple pointers in shared memory
- If someone else updates the list at the same time, that will corrupt the list
- So in order to safely update the list it must be locked!
- Because many people need to do this, the lock forms a bottleneck

White Star
software

So what does -spin have to do with it?

- To obtain a latch you "test and set" until you succeed
- This can consume a lot of CPU cycles
- But it takes much less time and fewer resources than waiting for a semaphore
- The -spin value is the number of times to try before giving up
- If you "spin out" then your process will nap before trying again
- The length of your nap increases progressively from -nap to -napmax until you eventually succeed and get the latch!

White Star software

The spin in “spin lock”

- High performance is obtained by testing the latch continuously until it is available
- Excessive resource consumption is avoided by limiting the number of attempts (-spin)
- After the -spin value is reached the process naps for a short period (-nap)
- The more times it fails to get the lock, the longer the nap (-napmax)
- There are additional optimizations possible ;)

White Star software

Latch Requests & Waits (naps): ProTop “w”

Latch Activity

Latch	Type	Hlder	QHold	Requests	Waits v	Lock%
>BF1	Spin	-1	-1	1813487	8	100.00%
LRU	Spin	14	-1	906745	4	100.00%
BHT	Spin	-1	-1	523120	2	100.00%
USR	Queue	12	12	1	0	100.00%
RPL	Spin	0	-1	0	0	?
SEC	Spin	-1	-1	0	0	?
CDC	Spin	-1	-1	0	0	?
INC	Spin	-1	-1	0	0	?
BF4	Spin	-1	-1	0	0	?
BF3	Spin	-1	-1	0	0	?
BF2	Spin	-1	-1	0	0	?
LRU2	Spin	-1	-1	0	0	?
CPQ	Spin	12	-1	0	0	?
PWQ	Spin	-1	-1	0	0	?
BFP	Spin	17	-1	0	0	?
LKF	Spin	15	-1	0	0	?
EC	Spin	-1	-1	0	0	?
TXQ	Spin	12	-1	0	0	?
AIB	Spin	12	-1	0	0	?
SEQ	Spin	0	-1	0	0	?
LHT4	Spin	-1	-1	0	0	?
LHT3	Spin	-1	-1	0	0	?
LHT2	Spin	-1	-1	0	0	?
LHT	Spin	-1	-1	0	0	?
TXT	Spin	12	-1	0	0	?
GST	Spin	15	-1	0	0	?
LKP	Spin	12	-1	0	0	?
SCC	Queue	15	15	0	0	?
BIB	Spin	12	-1	0	0	?
OM	Spin	12	-1	0	0	?
MTX	Spin	12	-1	0	0	?

Resource Waits

Resource	Requests	Waits v	Lock%
>DB Buf S Lock	898845	0	100.00%
Statement cach	0	0	?
Encryption buf	0	0	?
DB Svc enqueue	0	0	?
Repl TEND Ack	0	0	?
TXE Excl Lock	0	0	?
TXE Commit Loc	0	0	?
TXE Update Loc	0	0	?
TXE Share Lock	0	0	?
AI Buf Write	0	0	?
AI Buf Read	0	0	?
BI Buf Write	0	0	?
BI Buf Read	0	0	?
DB Buf Write L	0	0	?
DB Buf X Lock	0	0	?
DB Buf S Lock	0	0	?
DB Buf Avail	0	0	?
DB Buf X Lock	0	0	?
DB Buf Backup	0	0	?
DB Buf Write	0	0	?
DB Buf Read	0	0	?
Record Get	0	0	?
DB Buf I Lock	0	0	?
Trans Commit	0	0	?
Schema Lock	0	0	?
Record Lock	0	0	?
Shared Memory	0	0	?

TXE Locks & Waits

TXE Type	Locks	Waits v	Lock%	ConLk%	ConLk	ConWt
>Record Cr	0	0	?	?		
Record Mo	0	0	?	?		
Record De	0	0	?	?		
Key Add	0	0	?	?		
Key Delet	0	0	?	?		
Sequence	0	0	?	?		
Other	0	0	?	?		
Update Lo	0	0	?	?		
Commit Lo	0	0	?	?		

White Star
software

Single Threaded Execution

- Only one process can hold a specific latch at a time
- So access to any particular latch forms a bottleneck equivalent to what a single thread can execute
- You cannot process more latch requests for any one latch than one CPU core can handle

Single Threaded Execution

- Only one process can hold a latch at a time
- So access to a latch forms a single-threaded bottleneck
- You cannot process more latch requests for any one latch than one CPU core can handle

- The only solutions are:
  - Request fewer latches
  - Spread locking over more latches
  - Get faster cores

White Star software

Request Fewer Latches

- -lruskips (later)
- Fix your code ;)

White Star
software

More Latches

- Upgrade!
  - Progress frequently improves the underlying algorithms to better spread things around
- -hashLatchFactor
  - Adjusts the number of BHT latches relative to -hash
- Use more blocks
  - BF* latches are actually one per block, smaller RPB spreads data over more blocks and thus more latches, but wastes a lot of space and memory
  - This really only works for very small yet very active tables

White Star
software

Faster Cores

- Since the bottleneck is what a single core can execute you need faster cores to get more work done in less time
- ADDITIONAL CORES WILL NOT HELP
- Lots of slow cores helps even less
- There is a trade off between number of cores per socket and CPU speed, to get faster cores you need fewer of them

The Impact of Too Many Cores

- Latches are frequently accessed memory locations
- Which means that they are in the CPU cache…
- … of every active core
- Coordinating those caches across the bus is complex and expensive:
  - Testing a value (reading it) is relatively cheap
  - Updating it (setting it to a new value) is extremely expensive
  - It is dramatically more expensive when the caches are on different chips or different CPU boards

White Star software

Too Many Cores II

- You want to have your cores physically close to each other
- Ideally in the same chip
- Less ideally on the same board
- The more distance signals have to travel and the more handoffs across the bus the worse performance will be
- Modern operating systems try to ensure that the smallest number of cores are used and that they are "close" together – but virtualization often defeats this

White Star
software

Some Famous “Too Many Cores” Servers

- HP Superdome
- SUN “Niagra”
- IBM P790

White Star
software

Some Famous “Too Many Cores” Servers

- HP Superdome
- SUN “Niagra”
- IBM P790
- Any Intel Server with more than 8 cores

White Star
software

Some Famous “Too Many Cores” Servers

- HP Superdome
- SUN “Niagra”
- IBM P790
- Any Intel Server with more than 8 cores

- Virtualization of these environments does not magically fix them!

White Star software

Back to OpenEdge…

Some Important OpenEdge Latches

- LRU - Least Recently Used block chain
- MTX - Micro-transaction
- BHT - Buffer Hash Table
- OM - Object Manager
- BF# - Buffer control
- SEQ - Sequence Updates
- LHT - Lock Hash Table (-L)
- LKF - Lock Table Free List
- LKP - Lock Purge Table

What to do when you have naps on LRU?

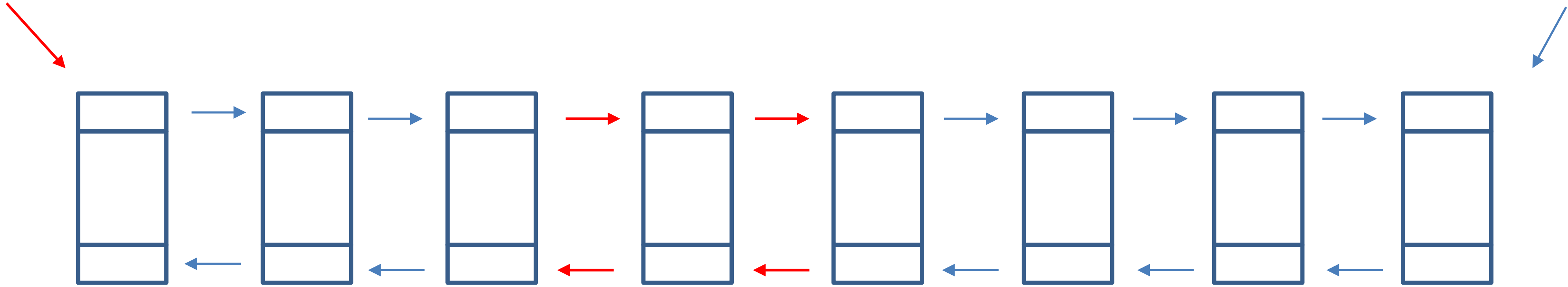
- -lruskips 10
- Only update the LRU chain every 10th time a block is accessed
- The order of the chain is no longer perfect
- But that is more than good enough for most purposes
- You quickly get to diminishing returns with large values
  - Values > 100 are probably not going to be very helpful
  - Very small -B and very large -lruskips is probably not a good combination

White Star
software

Updating the LRU Chain

Most Recently Used

Least Recently Used



Add a counter and only update every Nth block access

White Star
software

What to do when you have naps on MTX?

- MTX naps occur when committing data to disk
- They are most frequently seen on slow, external disk subsystems
  - SAN with RAID 5 for instance
- Or review your code and reduce the rate at which you commit transactions
  - Perhaps you have transaction scoping problems causing more commits than you expect
  - Or maybe you are doing lots of single record commits that could be grouped together

White Star
software

What to do when you have naps on BHT?

- You probably have some **very** high record read rates, millions per second
- This is often due to poor WHERE clauses or poor index support for queries
- If you cannot fix the code or add an index, faster cores are your best bet
- OE11.7+ -hashLatchFactor may help

White Star
software

What to do when you have naps on OM?

- Object Manager cache
- -omsize should be at least equal to the total number of tables, indexes and LOBs in your schema – including the metaschema

White Star
software

BF1 <> BF2 <> BF3 <> BF4

- The BF* latches are an accounting fiction
- There is actually one such latch for every block in -B
- But that would get messy to track so there are 4 "buckets"
  - Normally the values for all 4 buckets are "close"
  - If one of them consistently spikes then you may have a "hot spot"
  - The chances are fairly good that the driver of that hot spot is an unusually active user/table/index/sequence

Lots of Napping

- Many latch waits
- Relatively high %sys
- Spare %usr CPU capacity is available
- Increase -spin!

CPU Exhaustion

- You use up all of your CPU power
- The number of latch **requests** plummets
- But nobody is napping and users complain that the system is unresponsive
- -spin cannot complete before your process is rescheduled

- Reduce -spin

White Star
software

Suggested Value for -spin?

- Be very wary of “# CPUs * X”
  - That was invented when # was rarely more than 4
  - Now you can easily buy a server with # = 96
    - (IMHO only about 6 OE databases in the world need more than 8)

One –spin to Rule Them All?

- There is no One Spin To Rule Them All
  - I suggest that **most** people should start with 20,000
  - We have customers who are happy at 100,000 or more
  - One customer has 1,000,000 – but that is a **very** special case
  - Another customer had to drop it to 5,000 (to avoid CPU exhaustion)
  - Monitor your application and tune it up or down
    - Watch out for excess CPU utilization
    - Watch out for naps
    - Measure throughput, if increasing -spin consistently increases throughput then keep increasing

White Star software

Questions?

White Star
software

Thank You!

White Star
software

White Star
software