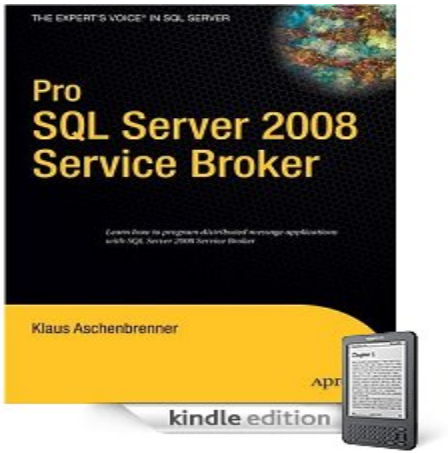

PASS SQLRally
Microsoft
PASS
Latches, Spinlocks, and Lock-Free Data Structures

About me

- CEO & Founder SQLpassion
- International Speaker, Blogger, Author
- SQL Server 2008 MCM
- "Pro SQL Server 2008 Service Broker"
- Twitter: @Aschenbrenner
- SQLpassion Academy
  - http://www.SQLpassion.at/academy
  - Free Newsletter, Training Videos



Caution!

If you are latched or spinlocked by the session, there is always a way to back-off: apply a lock-free operation!

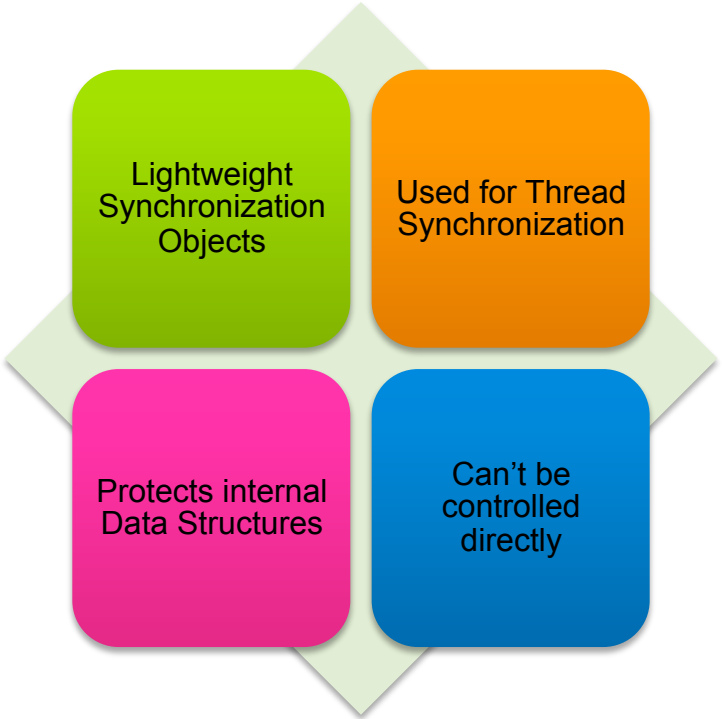
Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

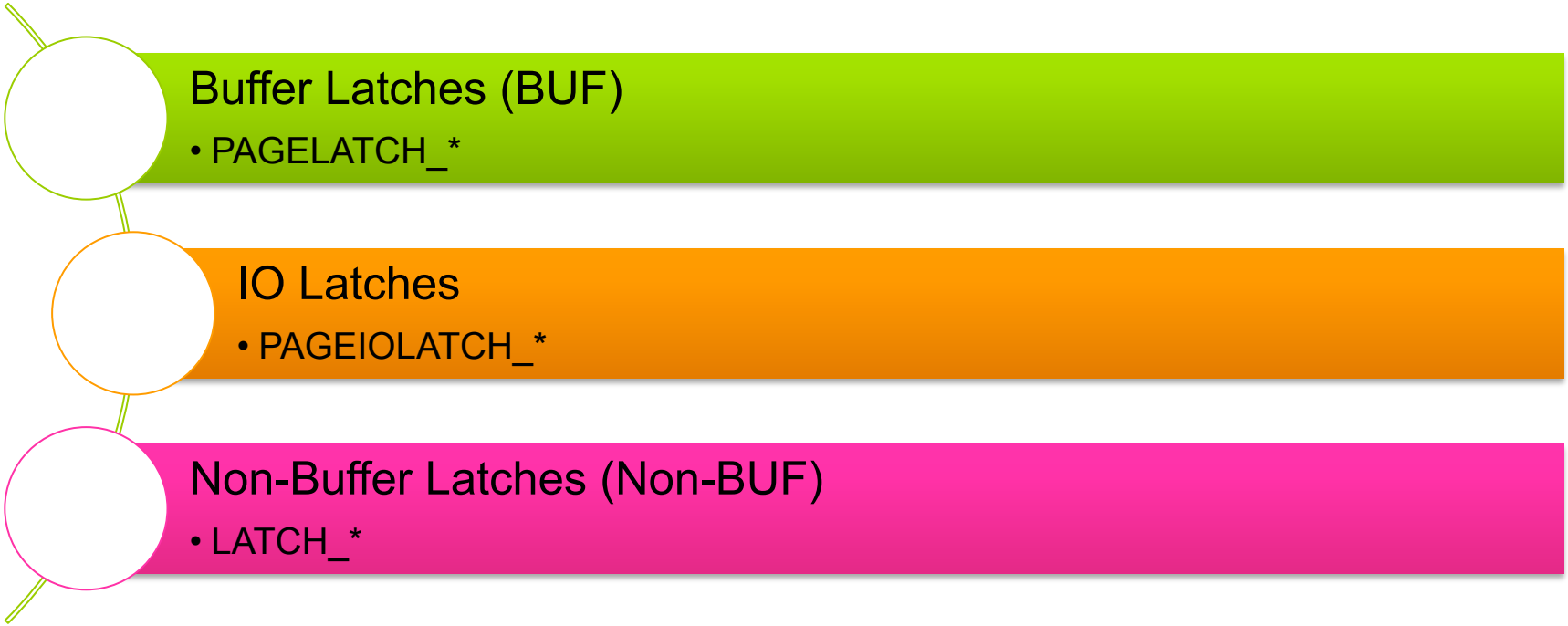
Latches – what are they?



Locks vs. Latches

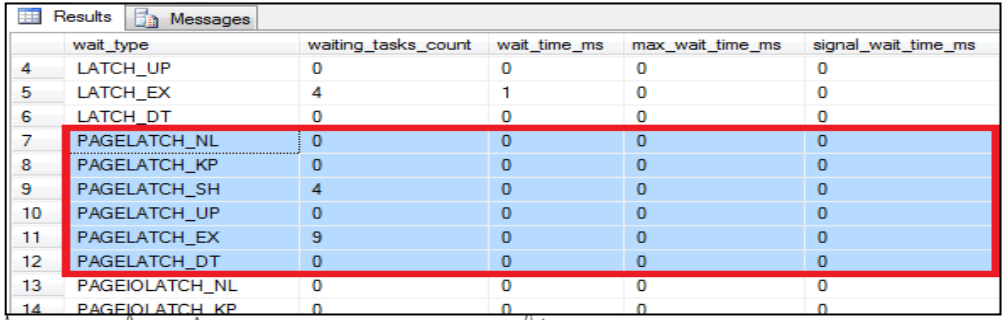
	Locks	Latches
Controls…	Transactions	Threads
Protects…	Database content	In-Memory Data Structures
During…	Entire transaction	Critical section
Modes…	Shared, Update, Exclusive, Intention	Keep, Shared, Update, Exclusive, Destroy
Deadlock…	Detection & Resolution	Avoidance through careful coding techniques
Kept in…	Lock Manager's Hashtable	Protected Data Structure

Latch Types


Buffer Latches (BUF)
• PAGELATCH_*
IO Latches
• PAGEIOLATCH_*
Non-Buffer Latches (Non-BUF)
• LATCH_*

BUF-Latches

- Protect all kinds of pages when they are accessed from the Buffer Pool
  - Data Pages/Index Pages
  - PFS/SGAM/GAM Pages
  - IAM Pages
- PAGELATCH_*
- Accessible through sys.dm_os_wait_stats



Results | Messages

	wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms	signal_wait_time_ms
4	LATCH_UP	0	0	0	0
5	LATCH_EX	4	1	0	0
6	LATCH_DT	0	0	0	0
7	PAGELATCH_NL	0	0	0	0
8	PAGELATCH_KP	0	0	0	0
9	PAGELATCH_SH	4	0	0	0
10	PAGELATCH_UP	0	0	0	0
11	PAGELATCH_EX	9	0	0	0
12	PAGELATCH_DT	0	0	0	0
13	PAGEIOLATCH_NL	0	0	0	0
14	PAGEIOLATCH_KP	0	0	0	0

I/O Latches

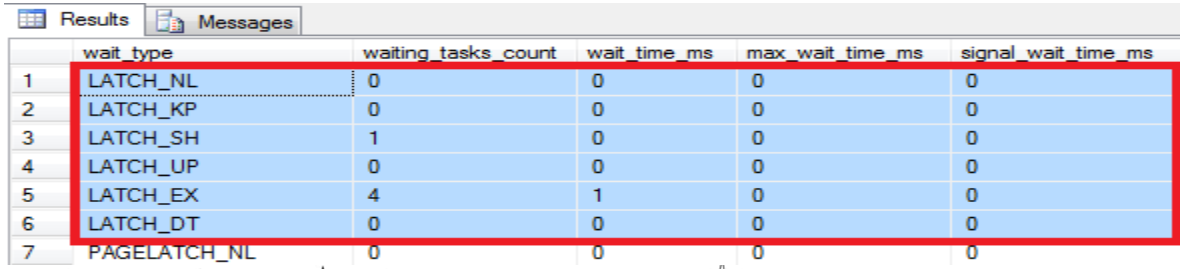
- Subset of BUF Latches
- Used when outstanding I/O operations are done against pages in the Buffer Pool
  - Disk to Memory Transfers (Reading)
  - Memory to Disk Transfers (Writing)
- SQL Server is waiting on the I/O subsystem
- PAGEIOLATCH_*

Results | Messages

	wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms	signal_wait_time_ms
11	PAGELATCH_EX	9	0	0	0
12	PAGELATCH_DT	0	0	0	0
13	PAGEIOLATCH_NL	0	0	0	0
14	PAGEIOLATCH_KP	0	0	0	0
15	PAGEIOLATCH_SH	1039	2033	45	30
16	PAGEIOLATCH_UP	56	121	30	1
17	PAGEIOLATCH_EX	89	40	2	0
18	PAGEIOLATCH_DT	0	0	0	0
19	TRAN_MARKLATCH_NL	0	0	0	0

Non-BUF Latches

- Guarantees the consistency of any other in-memory structures other than Buffer Pool pages
- LATCH_*
- Detailed breakdown in sys.dm_os_latch_stats



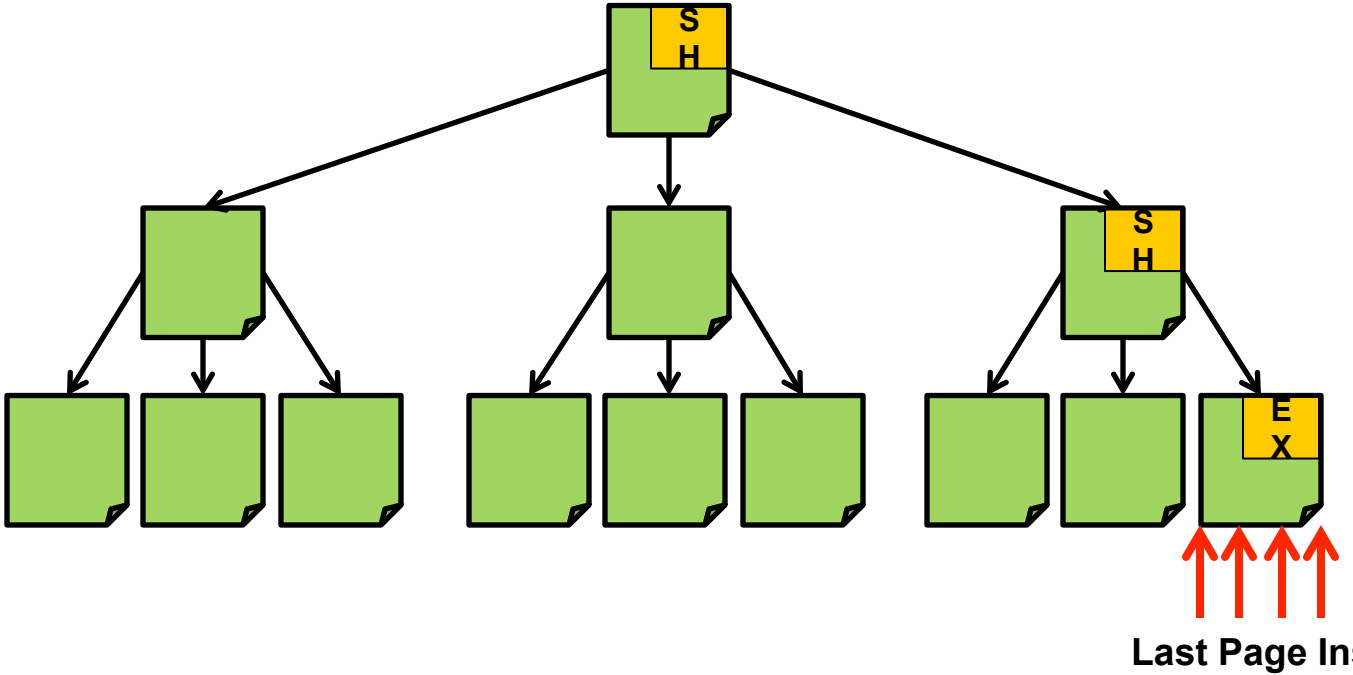
Results | Messages

	wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms	signal_wait_time_ms
1	LATCH_NL	0	0	0	0
2	LATCH_KP	0	0	0	0
3	LATCH_SH	1	0	0	0
4	LATCH_UP	0	0	0	0
5	LATCH_EX	4	1	0	0
6	LATCH_DT	0	0	0	0
7	PAGELATCH_NL	0	0	0	0

Demo

Exploring Latches

Last Page Insert Latch Contention



Current Solutions

- Random Clustered Keys
  - UNIQUEIDENTIFIER
  - Distributes the INSERTs across the Leaf Level
  - Larger Lookup Values in Non-Clustered Indexes…
  - Index Fragmentation
- Hash Partitioning
  - Distribute INSERTs across different partitions
  - Every CPU core has its own partition
  - You can't additionally partition your table…
  - Partition Elimination is almost impossible…
- In-Memory OLTP
  - SQL Server 2014+

Demo

Last Page Insert Latch Contention

TempDb Latch Contention

- Big amount of creation and destruction of many objects
  - Temp Tables
  - Table Variables
- Can lead to Latch Contention
  - PFS Page
  - GAM/SGAM Page

Creating a temp table means...

1. Reading the SGAM page (2:1:3) to find a mixed extent with free space
   - SQL Server uses an exclusive latch on the SGAM page while updating the page
2. Reading the PFS page (2:1:1) to find a free page within the extent
   - SQL Server also uses an exclusive latch on the PFS page while updating the page
3. SQL Server will report a PAGELATCH wait type with the appropriate resource description
   - 2:1:3 for SGAM page
   - 2:1:1 for PFS page

Resolving Latch Contention 1/3

- Multiple TempDB data files
  - ¼ to ½ data files of the CPU cores you have (HT cores should be included in the calculation!)
  - Allocation of new objects is done round-robin between the data files
  - All data files must have the same size to be effective
- Don‘t use the Microsoft recommendation – 1:1 mapping between data files and CPU cores!
  - http://www.sqlskills.com/BLOGS/PAUL/post/A-SQL-Server-DBA-myth-a-day-(1230)-tempdb-should-always-have-one-data-file-per-processor-core.aspx

Demo

Resolving Latch Contention 1/3

Resolving Latch Contention 2/3

- Temporary Object Reuse
  - SQL Server can cache temporary objects instead of recreating them again and again
  - 1 IAM page and 1 Extent are cached
  - http://sqlblog.com/blogs/paul_white/archive/2012/08/17/temporary-object-caching-explained.aspx
- Caching is possible when
  - Named constraints are not created
  - DDL statements are not used that effect the object like
    - CREATE INDEX
    - CREATE STATISTICS
  - Object is not created dynamically, e.g. through sp_executesql
  - Object is created inside another object
    - Stored Procedure, Trigger, UDF

Demo

Resolving Latch Contention 2/3

Resolving Latch Contention 3/3

- Trace Flag 1118
  - Introduced with SQL Server 2000
  - Not really needed in SQL Server 2008, because of already improved algorithm when allocating space in Mixed Extents
- Disables all Mixed Extent allocations in TempDb
- Every object that is created, is created in an Uniform Extent
  - Every temp table therefore needs at least 64kb storage
- Should be only enabled when you have contention on the SGAM page (2:1:3)

Demo

Auto Growth – Transaction Log

Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

Why Spinlocks?

Query Life Cycle

How to protect a Latch?

Latches don’t scale!

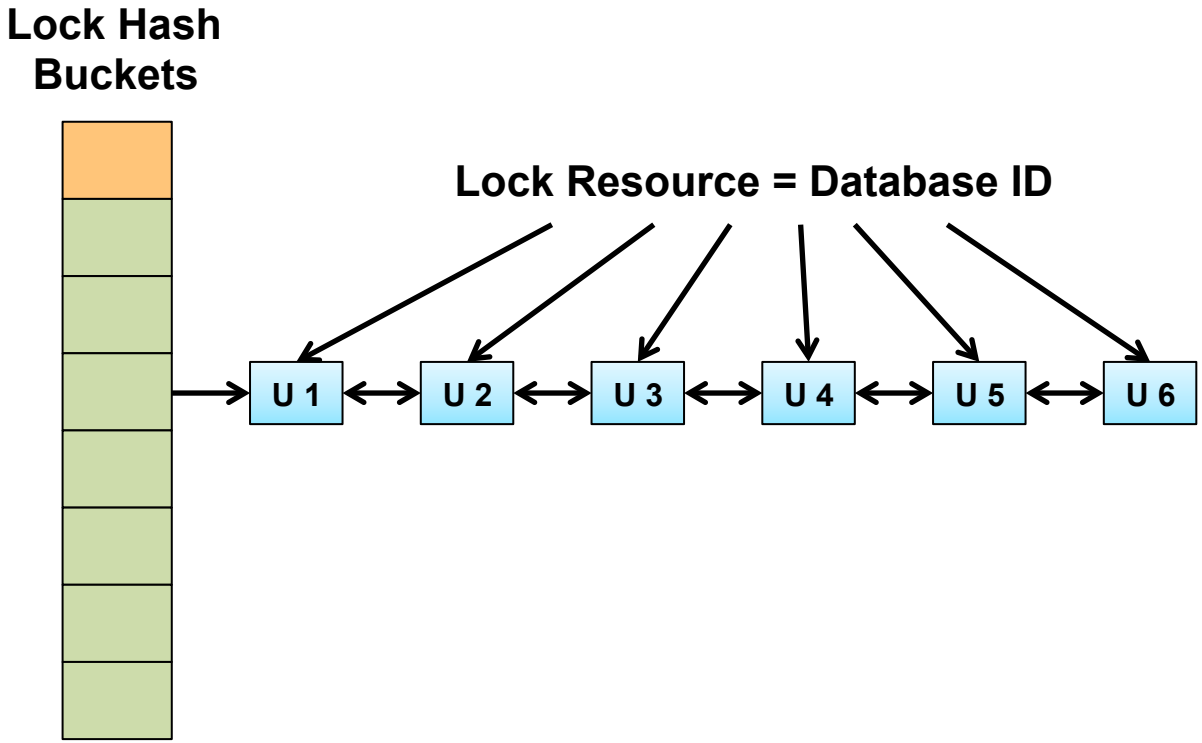
Spinlock Internals

- It's a Mutex (Mutual Exclusion)
  - No waiting list
  - No compatibility matrix
  - You hold the spinlock, or not!
- Used to protect "busy" data structures
  - Read or written very frequently
  - Held for a short amount of time
  - E.g. Lock Manager (LOCK_HASH)

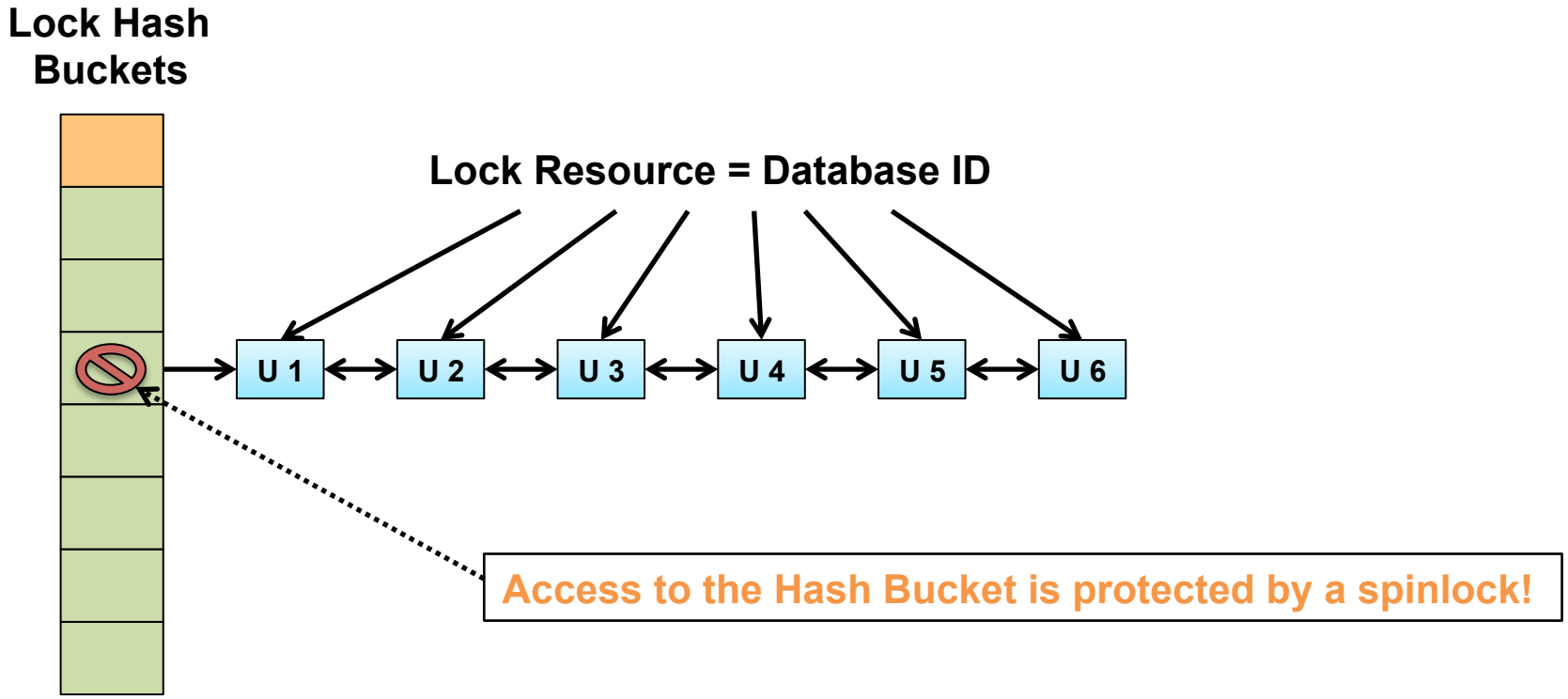
Spinlock Contention

- Problems
  - Tight spinning around a busy data structure
  - Short waits are expected!
  - Exponential back-off since SQL Server 2008 R2+
- Symptoms
  - High CPU usage without performing useful work
  - High “backoffs” in sys.dm_os_spinlock_stats

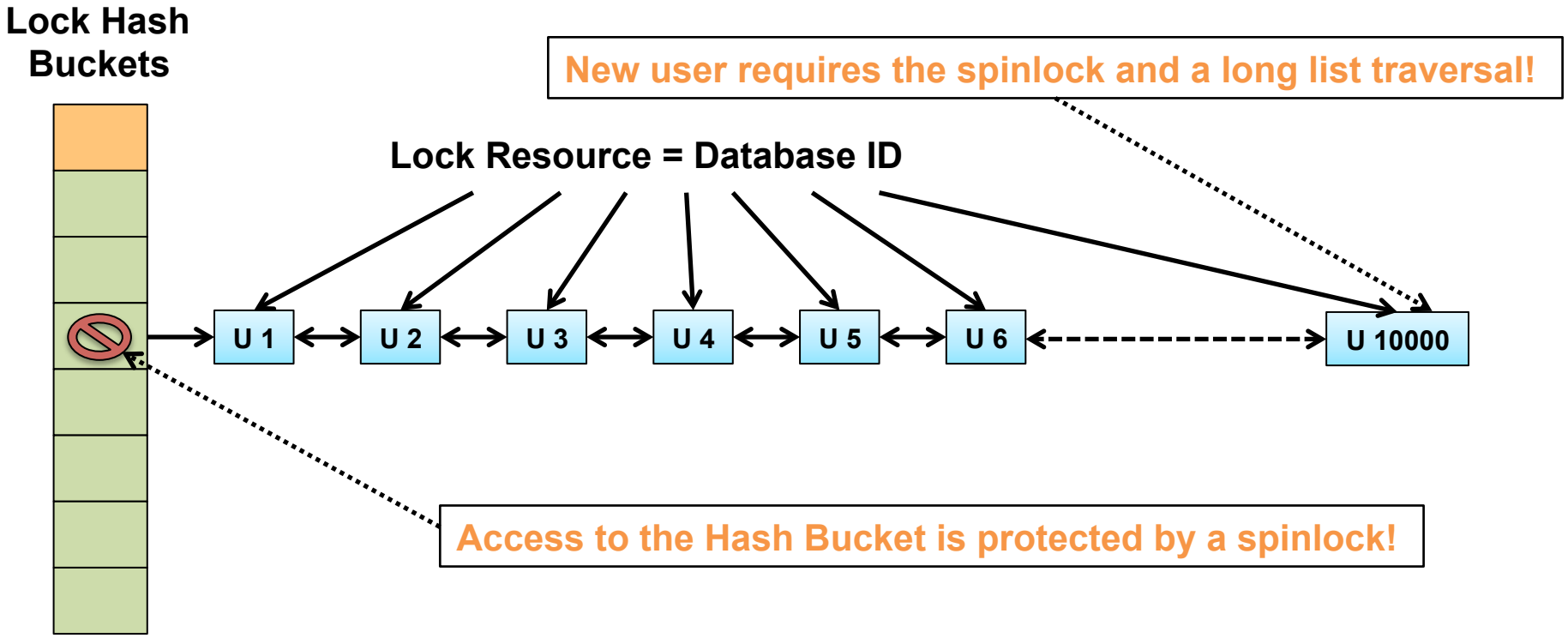
LOCK_HASH Example



LOCK_HASH Example



LOCK_HASH Example



Demo

Debugging Spinlock Contention

Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

Non-Blocking Algorithms

*"A **non-blocking algorithm** ensures that threads competing for a shared resource do not have their execution indefinitely postponed by mutual exclusion. A non-blocking algorithm is **lock-free** if there is guaranteed system-wide progress regardless of scheduling."*

Source: http://en.wikipedia.org/wiki/Non-blocking_algorithm

Traditional Spinlocks

```
int compare_and_swap(int *value, int expected, int newValue) {
   int temp = *value;

   if (*value == expected)
      *value = newValue;

   return temp;
}

void Foo() {
   do {
      while (compare_and_swap(&lock, UNLOCKED, LOCKED) != 0)
         ; /* Do nothing */

      /* Critical section */
      val = val + 5;

      lock = UNLOCKED;
   } while (true);
}
```


Traditional Spinlocks

```
int compare_and_swap(int *value, int expected, int newValue) {
   int temp = *value;

   if (*value == expected)
      *value = newValue;

   return temp;
}

void Foo() {
   do {
      while (compare_and_swap(&lock, UNLOCKED, LOCKED) != 0)
         ; /* Do nothing */

      /* Critical section */
      val = val + 5;

      lock = UNLOCKED;
   } while (true);
}
```

We want to execute this code in a thread-safe manner!

Traditional Spinlocks

```
int compare_and_swap(int *value, int expected, int newValue) {
   int temp = *value;

   if (*value == expected)
      *value = newValue;

   return temp;
}

void Foo() {
   do {
      while (compare_and_swap(&lock, UNLOCKED, LOCKED) != 0)
         ; /* Do nothing */

      /* Critical section */
      val = val + 5;

      lock = UNLOCKED;
   } while (true);
}
```


Implemented through one atomic hardware instruction: CMPXCHG

Traditional Spinlocks

```
int compare_and_swap(int *value, int expected, int newValue) {
   int temp = *value;

   if (*value == expected)
      *value = newValue;

   return temp;
}

void Foo() {
   do {
      while (compare_and_swap(&lock, UNLOCKED, LOCKED) != 0)
         ; /* Do nothing */

      /* Critical section */
      val = val + 5;

      lock = UNLOCKED;
   } while (true);
}
```

Implemented through one atomic hardware instruction: CMPXCHG

There is a shared resource involved!

Traditional Spinlocks

```
int compare_and_swap(int *value, int expected, int newValue) {
    int temp = *value;

    if (*value == expected)
        *value = newValue;

    return temp;
}

void Foo() {
    do {
        while (compare_and_swap(&lock, UNLOCKED, LOCKED) != 0)
            ; /* Do nothing */

        /* Critical section */
        val = val + 5;

        lock = UNLOCKED;
    } while (true);
}
```

Implemented th
hardware in

If one thread holds the spinlock, and gets suspended, we get stuck in the loop!

There is a shared resource involved!

Lock Free Approach

```
int compare_and_swap(int *value, int expected, int newValue) {
   int temp = *value;

   if (*value == expected)
      *value = newValue;

   return temp;
}

void Foo() {
   do {
      val = *addr;
   }
   while (compare_and_swap(&addr, val, val + 5) != 0)
}
```

Lock Free Approach

```
int compare_and_swap(int *value, int expected, int newValue) {
    int temp = *value;

    if (*value == expected)
        *value = newValue;

    return temp;
}

void Foo() {
    do {
        val = *addr;
    }
    while (compare_and_swap(&addr, val, val + 5) != 0)
}
```

We just check if someone has modified "addr" before we make the atomic addition

Lock Free Approach

```
int compare_and_swap(int *value, int expected, int newValue) {
    int temp = *value;

    if (*value == expect
        *value = newValue

    return temp;
}

void Foo() {
    do {
        val = *addr;
    }
    while (compare_and_swap(&addr, val, val + 5) != 0)
}
```

There is no shared resource, no other thread can block us anymore!

We just check if someone has modified "addr" before we make the atomic addition

Lock Free Approach

```
int compare_and_swap(int *value, int expected, int newValue) {
    int temp = *value;

    if (*value == expect
        *value = newValue

    return temp;
}

void Foo() {
    do {
        val = *addr;
    }
    while (compare_and_swap(&addr, val, val + 5) != 0)
}
```

There is no shared res
no other thread ca
anymo

In-Memory OLTP installs page changes in the mapping table of the Bw-Tree with this technique ☺

ed
ake
tion

Demo

In-Memory OLTP

Summary

- Latches
- Spinlocks
- Lock Free Data Structures

SQL Server Performance Tuning & Troubleshooting Workshop

- Date & Location
  - June 1 – 5 in London/United Kingdom
- Agenda
  - Database Internals & Execution Plans
  - Indexing & Statistics
  - Locking, Blocking, Deadlocking
  - Performance Troubleshooting
- Further information
  - http://www.SQLpassion.at/academy

Explore Everything PASS Has to Offer

24 HOURS OF PASS

Free SQL Server and BI Web Events

PASS SQL saturday

Free 1-day Training Events

PASS SQLRally

Regional Event

PASS SUMMIT 2015

This is Community

PASS BUSINESS ANALYTICS CONFERENCE 2015

Business Analytics Training

PASS CHAPTERS

Local User Groups Around the World

Session Recordings

PASS CommunityCONNECTOR

PASS Newsletter

PASS VIRTUAL CHAPTERS

Free Online Technical Training