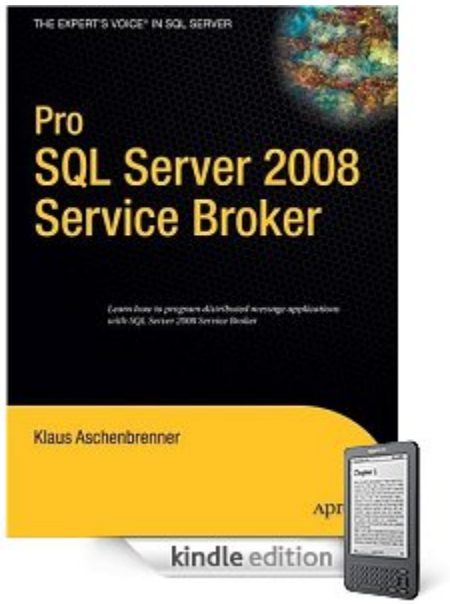
Latches, Spinlocks, and Lock Free Data Structures

Klaus Aschenbrenner
Microsoft Certified Master SQL Server 2008
www.SQLpassion.at
Twitter: @Aschenbrenner

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!

Caution!

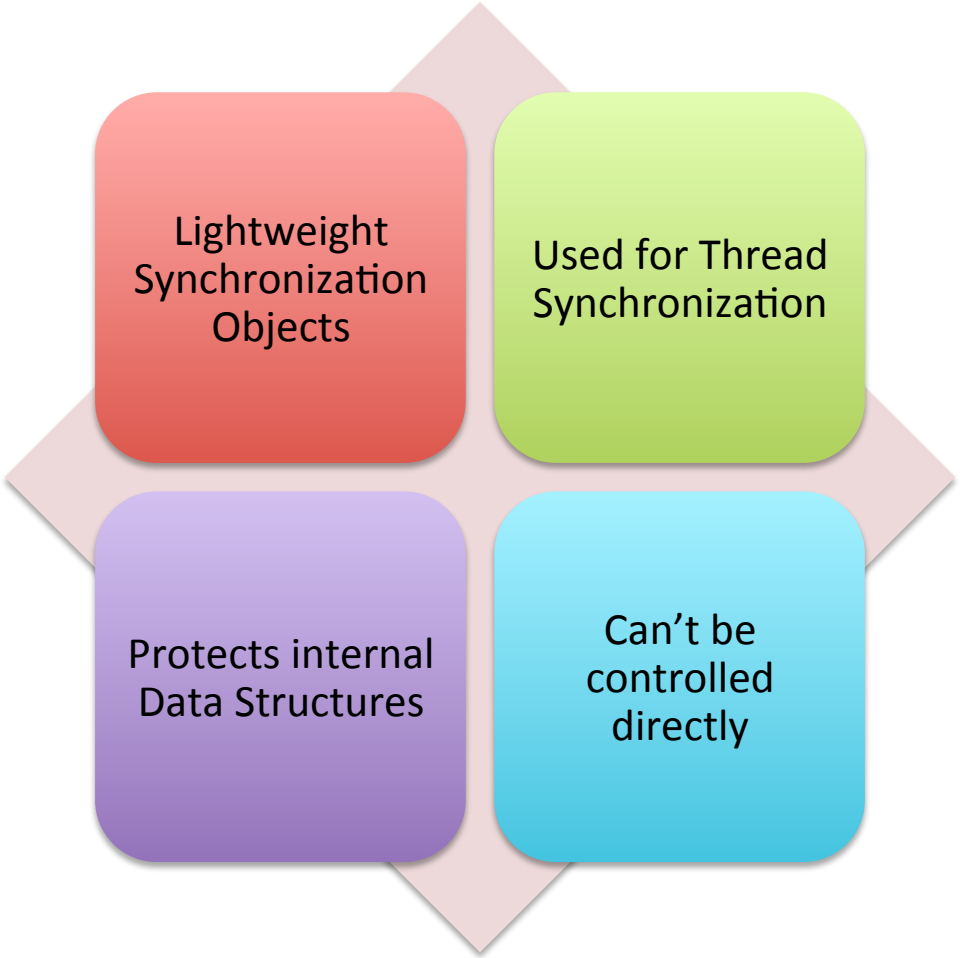
Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

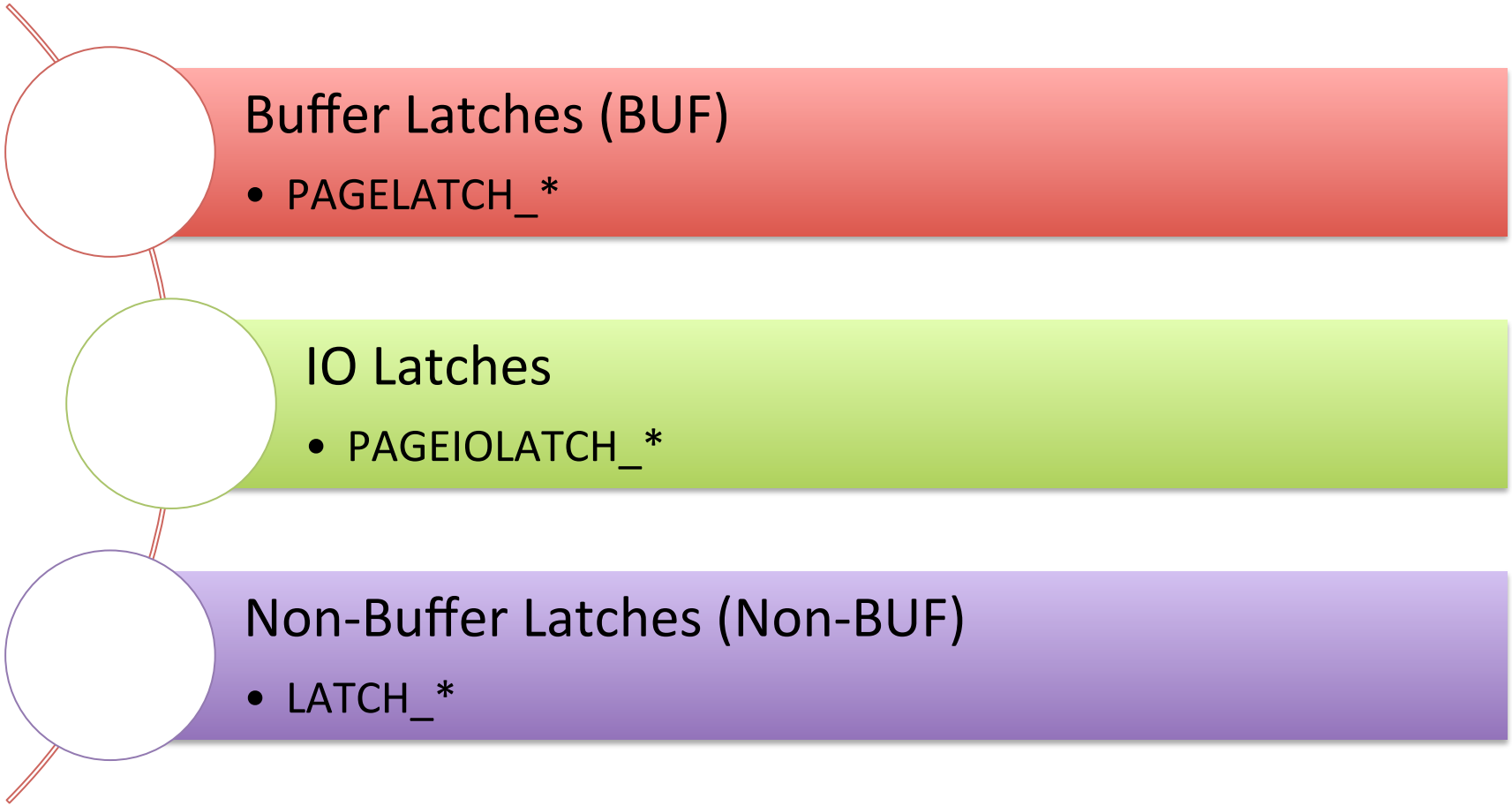
Latches – what are they?



Locks vs. Latches

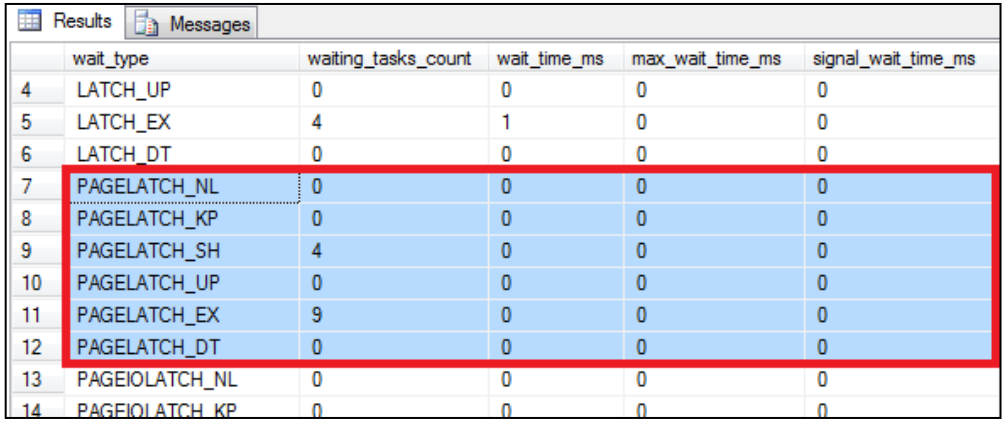
	Locks	Latches
Controls…	Transactions	Threads
Protects…	Database content	In-Memory Data Structures
During…	Entire transaction	Critical section
Modes…	Shared, Update, Exclusive, Intention	Keep, Shared, Update, Exclusive, Destroy
Deadlock…	Detection & Resolution	Avoidance through careful coding techniques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

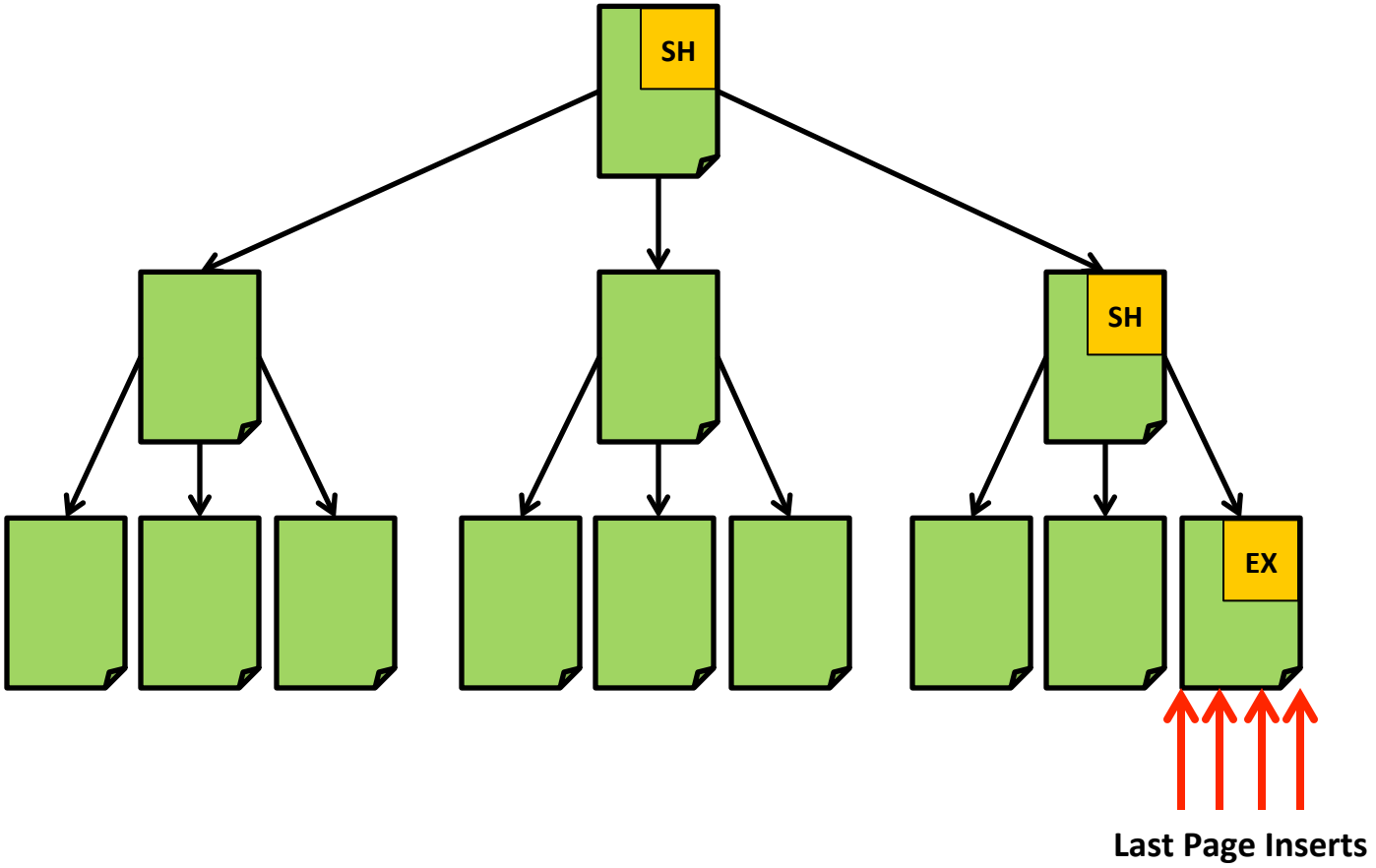
Results | Messages

	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+

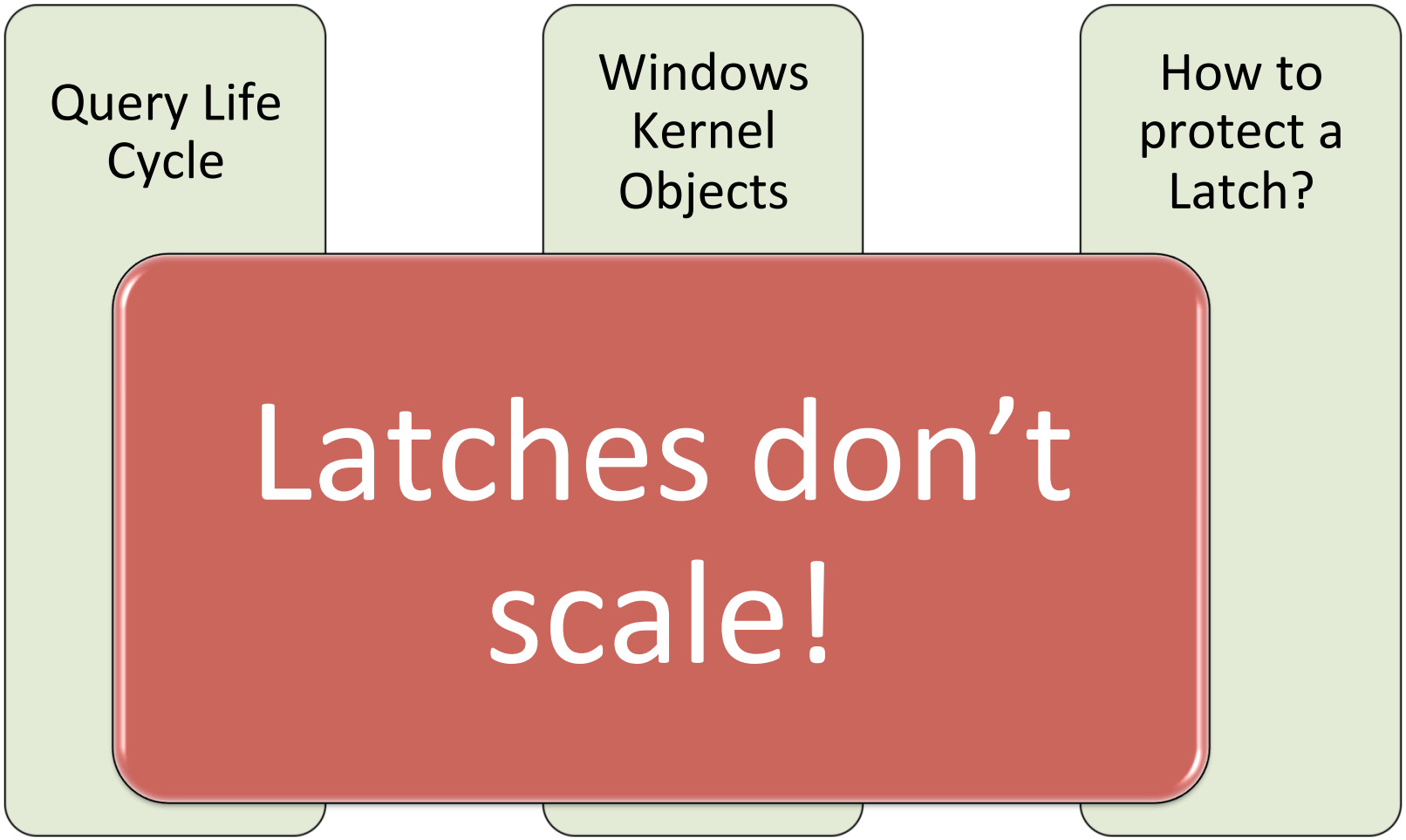
Demo

Last Page Insert Latch Contention

Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

Why Spinlocks?



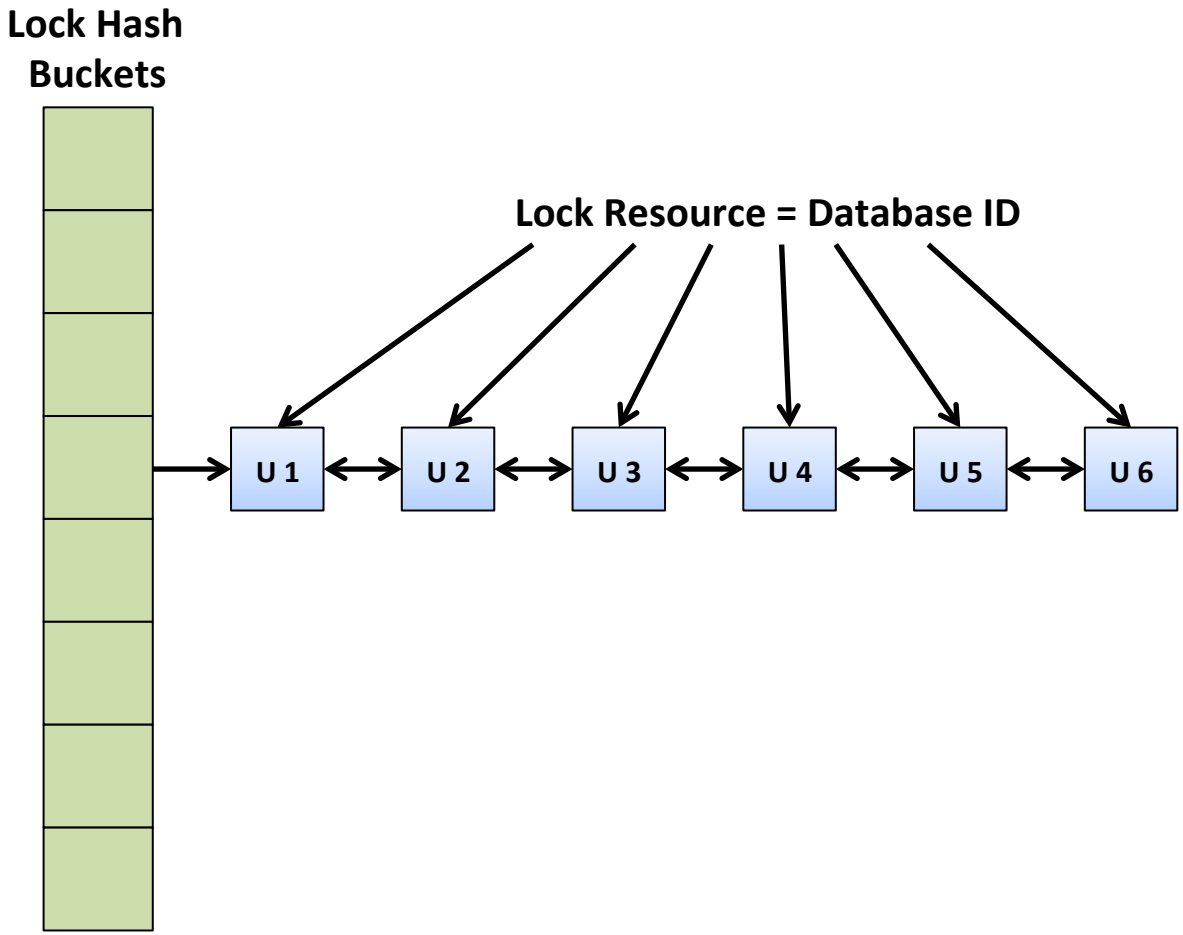
Spinlock Internals

- 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)

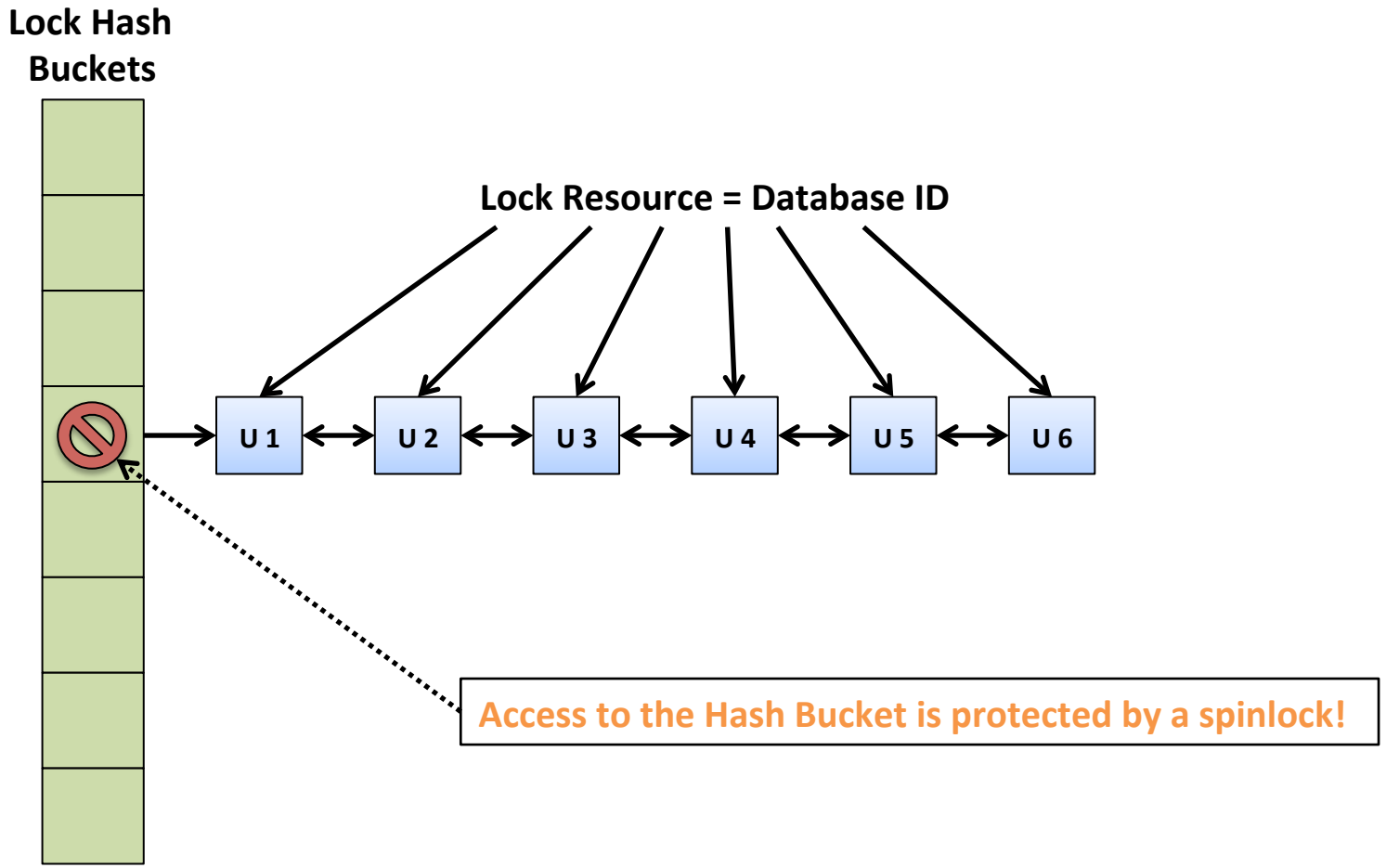
Spinlock Contention

- Problem
  - 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

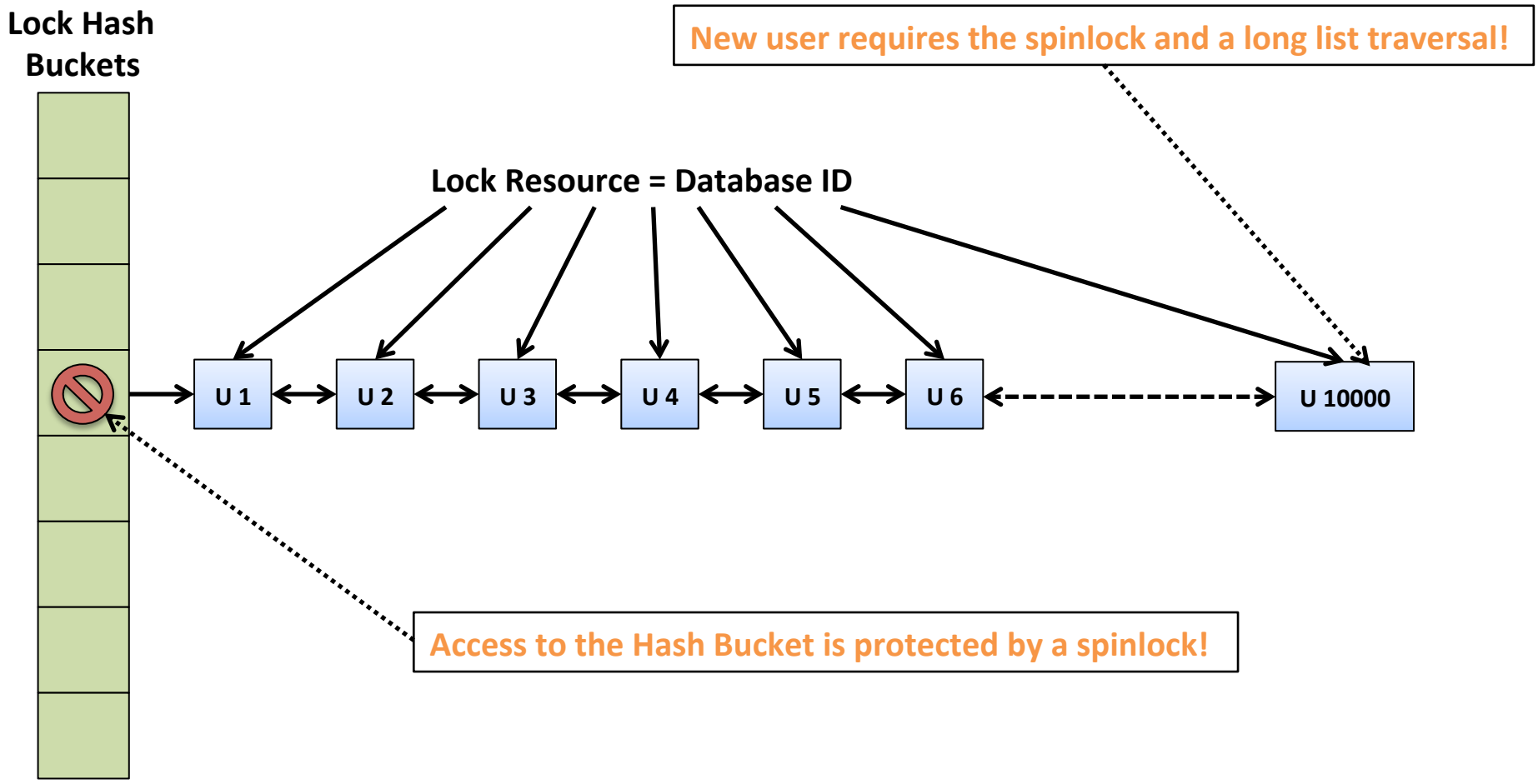
LOCK_HASH Example



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

hardware

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 = newValu

    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 = newValu

    return temp;
}

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

There is no shared resource, no c… thread can h… anymo…

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

…ke the …ddition

Summary

- Latches
- Spinlocks
- Lock Free Data Structures

SQL Server Query Tuning Workshop

- Date & Location
  - October 20 – 23 in London
- Agenda
  - How to write high performance T-SQL queries
  - Logical & physical query processing
  - Execution Plan Troubleshooting
  - Applying Indexing Strategies
  - Using In-Memory Technologies
- Further information
  - http://www.SQLpassion.at/academy