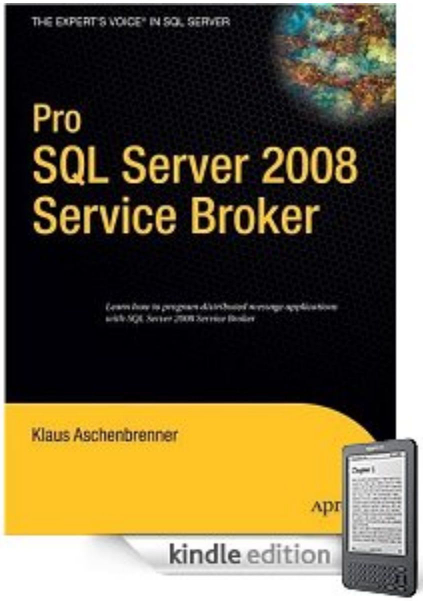
Latches, Spinlocks, and Lock-Free Data Structures

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

About me

- CEO & Founder SQLpassion
- International Speaker, Blogger, Author
- Microsoft Certified Master SQL Server
- „Pro SQL Server 2008 Service Broker“
- Twitter: @Aschenbrenner
- SQLpassion Academy
  - https://www.SQLpassion.at/academy
  - Free Newsletter, Training Videos



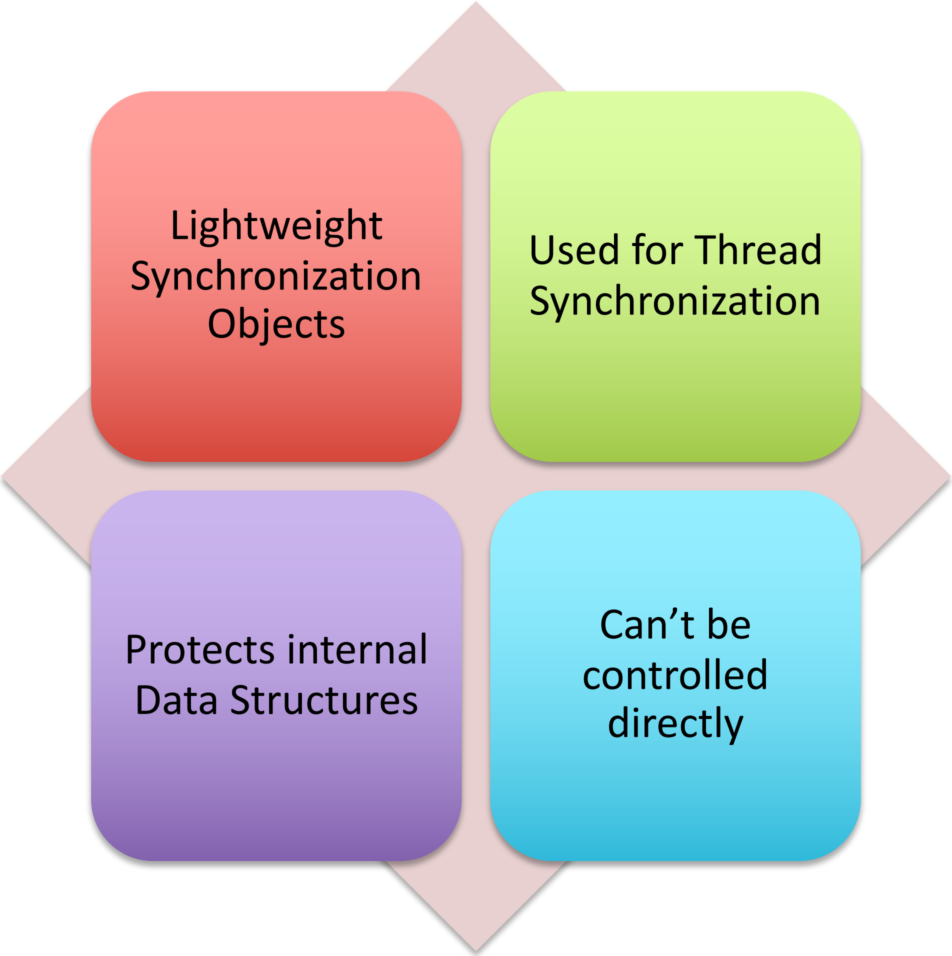
Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

Latches – what are they?


Lightweight Synchronization Objects

Used for Thread Synchronization

Protects internal Data Structures

Can’t be controlled directly

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

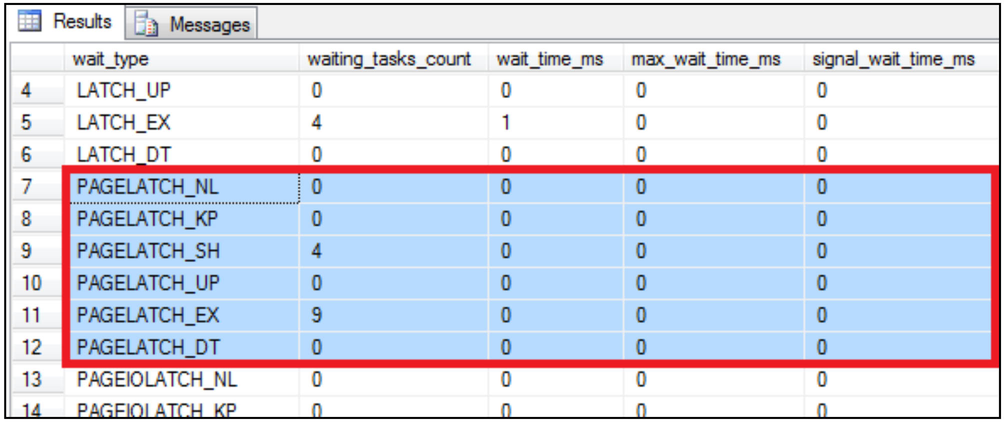
- PAGEIOLATCH_*

Non-Buffer Latches (Non-BUF)

- LATCH_*

BUF-Latches

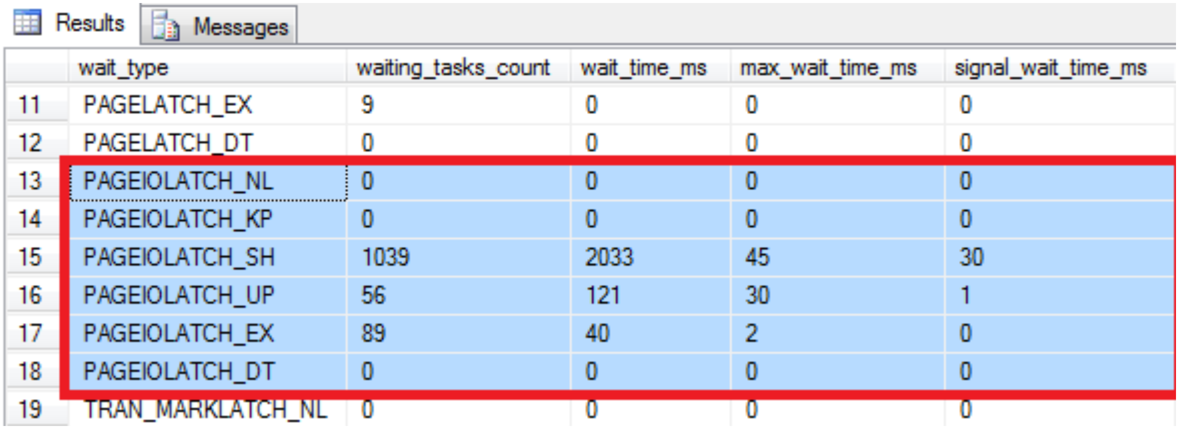
- Protect all kinds of pages when they are accessed from the Buffer Pool
  - Data Pages/Index Pages
  - PFS/SGAM/GAM Pages
  - IAM Pages
- PAGELATCH_*
- Accessible through sys.dm_os_wait_stats



	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

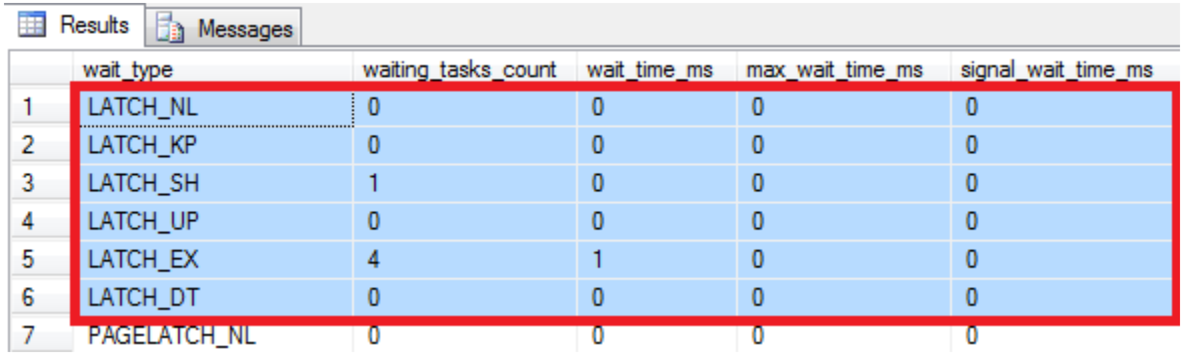
- Subset of BUF Latches
- Used when outstanding I/O operations are done against pages in the Buffer Pool
  - Disk to Memory Transfers (Reading)
  - Memory to Disk Transfers (Writing)
- SQL Server is waiting on the I/O subsystem
- PAGEIOLATCH_*



	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

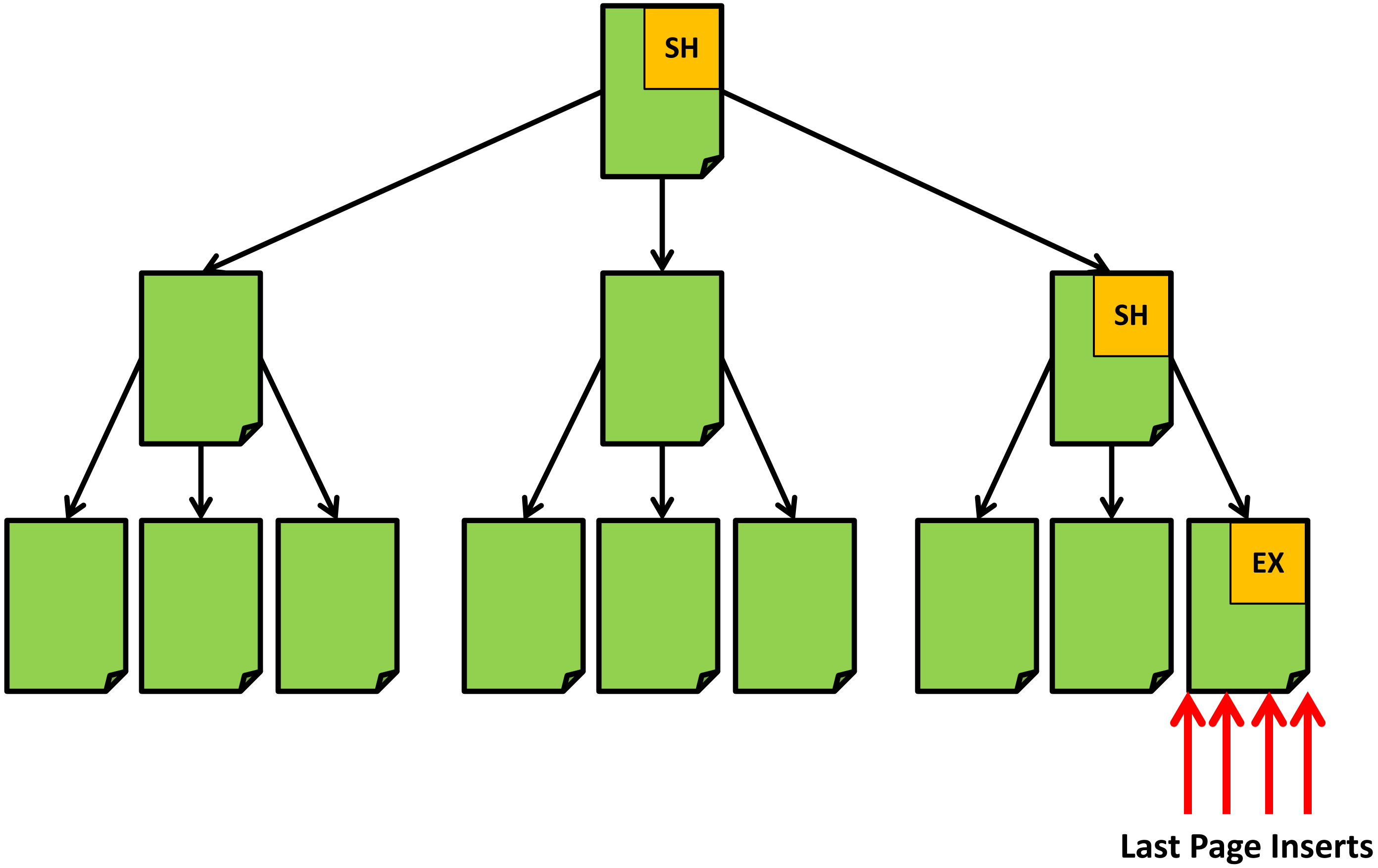


	wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms	signal_wait_time_ms
1	LATCH_NL	0	0	0	0
2	LATCH_KP	0	0	0	0
3	LATCH_SH	1	0	0	0
4	LATCH_UP	0	0	0	0
5	LATCH_EX	4	1	0	0
6	LATCH_DT	0	0	0	0
7	PAGELATCH_NL	0	0	0	0

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

Demo

Non-BUF Latches

Agenda

- Latches
- Spinlocks
- Lock Free Data Structures

Spinlock

- Spinlock
  - Lightweight Synchronization Object that protects In-Memory Data Structures, like
  - Lock Manager (LOCK_HASH)
  - Security Caches (SOS_CACHESTORE)
- Will be acquired when the resource will be held for a very short duration
  - Thread will not yield
  - Yielding is too expense because of Context Switching
  - The other threads must wait on the same resource

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

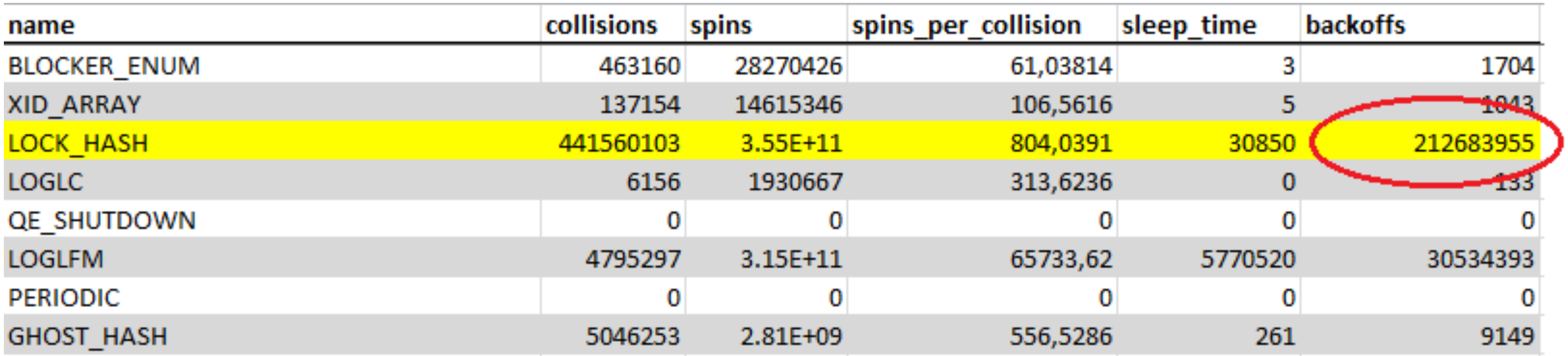
- Very high CPU usage
  - SOS_SCHEDULER_YIELD Wait Type
- Very high Concurrency
  - > 24 CPUs
  - > 32 CPUs
- Troubleshooting
  - sys.dm_os_spinlock_stats
  - Extended Events
    - Analyze Backoff Events
    - Provides you the Code Path in SQL Server where Contention occurs

Spinlock Contention Sample

- Service Broker should process 1 Mio messages per hour
  - CPU was 100%
  - No work was done anymore
- Contention
  - LOCK_HASH Spinlock
  - LCK_M_IS Wait Type
- SSB Activation Stored Procedure
  - Retrieved configuration settings from a Config table
  - Leaded to Spinlock Contention in LOCK_HASH
- Resolution (temporary)
  - WITH (NOLOCK)

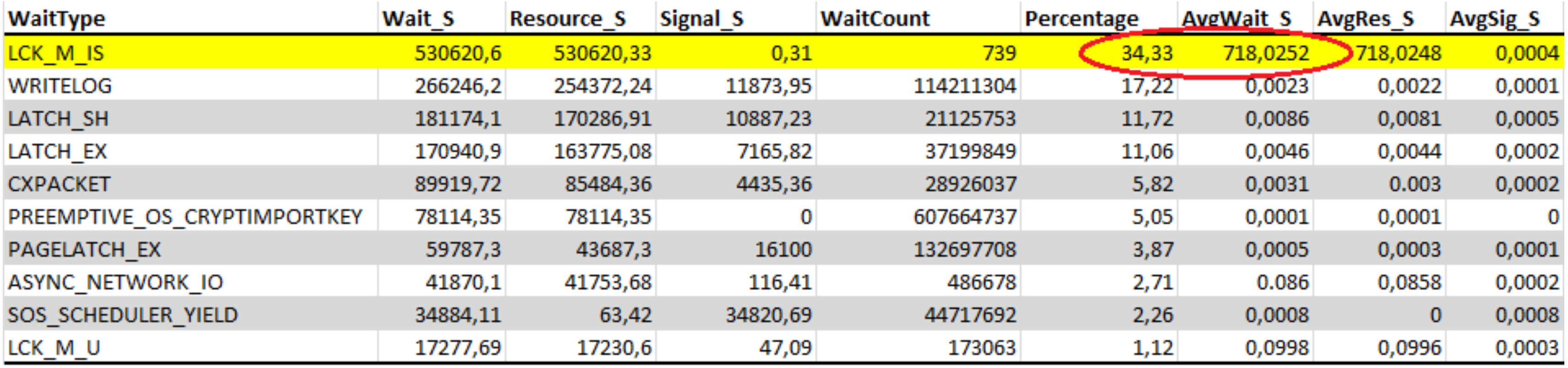
Spinlock Contention Sample

- Spinlock Stats



name	collisions	spins	spins_per_collision	sleep_time	backoffs
BLOCKER_ENUM	463160	28270426	61,03814	3	1704
XID_ARRAY	137154	14615346	106,5616	5	1043
LOCK_HASH	441560103	3.55E+11	804,0391	30850	212683955
LOGLC	6156	1930667	313,6236	0	133
QE_SHUTDOWN	0	0	0	0	0
LOGLFM	4795297	3.15E+11	65733,62	5770520	30534393
PERIODIC	0	0	0	0	0
GHOST_HASH	5046253	2.81E+09	556,5286	261	9149

- Wait Stats



WaitType	Wait_S	Resource_S	Signal_S	WaitCount	Percentage	AvgWait_S	AvgRes_S	AvgSig_S
LCK_M_IS	530620,6	530620,33	0,31	739	34,33	718,0252	718,0248	0,0004
WRITELOG	266246,2	254372,24	11873,95	114211304	17,22	0,0023	0,0022	0,0001
LATCH_SH	181174,1	170286,91	10887,23	21125753	11,72	0,0086	0,0081	0,0005
LATCH_EX	170940,9	163775,08	7165,82	37199849	11,06	0,0046	0,0044	0,0002
CXPACKET	89919,72	85484,36	4435,36	28926037	5,82	0,0031	0.003	0,0002
PREEMPTIVE_OS_CRYPTIMPORTKEY	78114,35	78114,35	0	607664737	5,05	0,0001	0,0001	0
PAGELATCH_EX	59787,3	43687,3	16100	132697708	3,87	0,0005	0,0003	0,0001
ASYNC_NETWORK_IO	41870,1	41753,68	116,41	486678	2,71	0.086	0,0858	0,0002
SOS_SCHEDULER_YIELD	34884,11	63,42	34820,69	44717692	2,26	0,0008	0	0,0008
LCK_M_U	17277,69	17230,6	47,09	173063	1,12	0,0998	0,0996	0,0003

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;
}
```

Implemented through one atomic hardware instruction: CMPXCHG

```
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;
}
```

Implemented through one atomic hardware instruction: CMPXCHG

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

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

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

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 o thread can h anymo

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

ke the
dition

Summary

- Latches
- Spinlocks
- Lock Free Data Structures