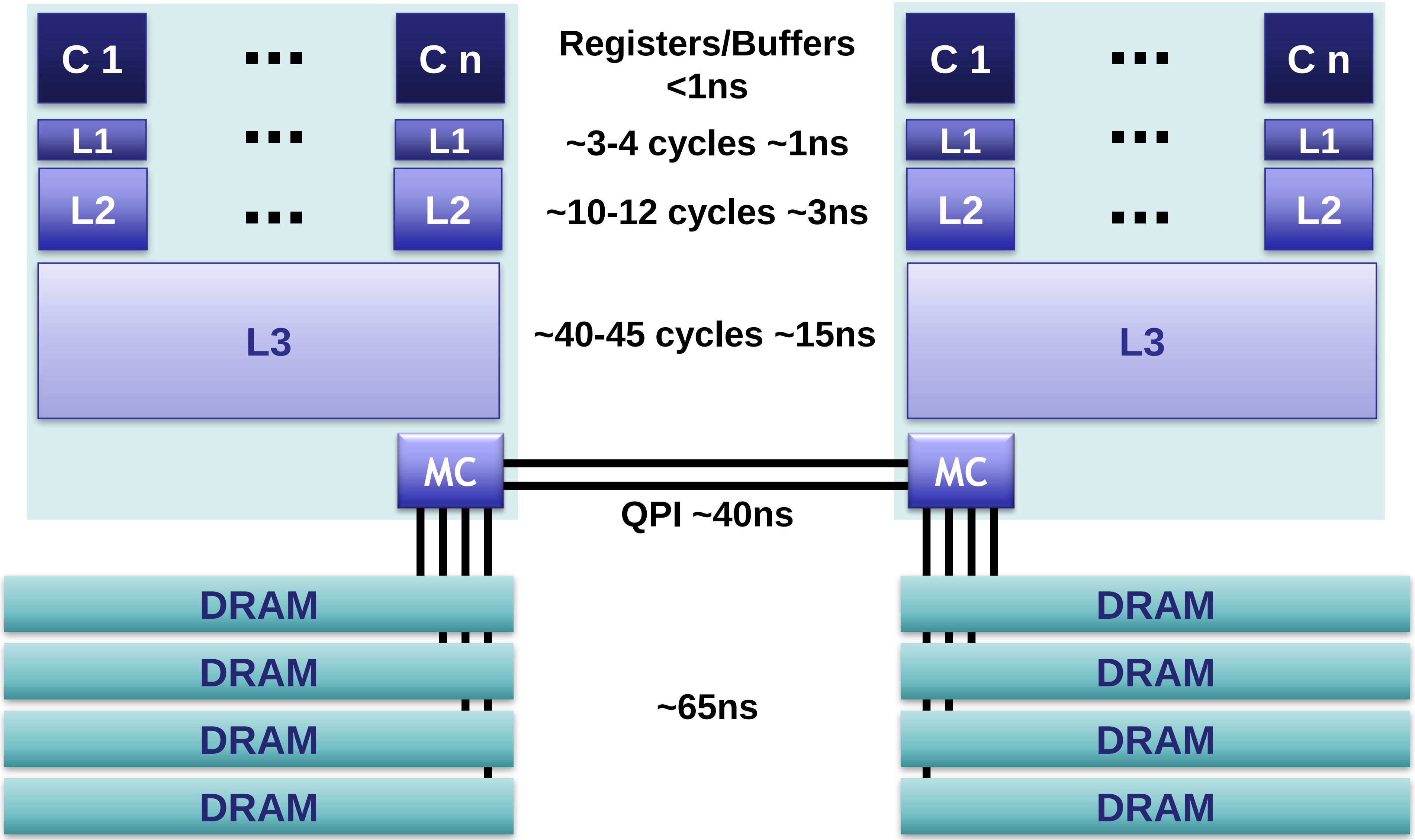
Real Logic
ACCELERATING SOFTWARE

Lock-Free Algorithms For Ultimate Performance

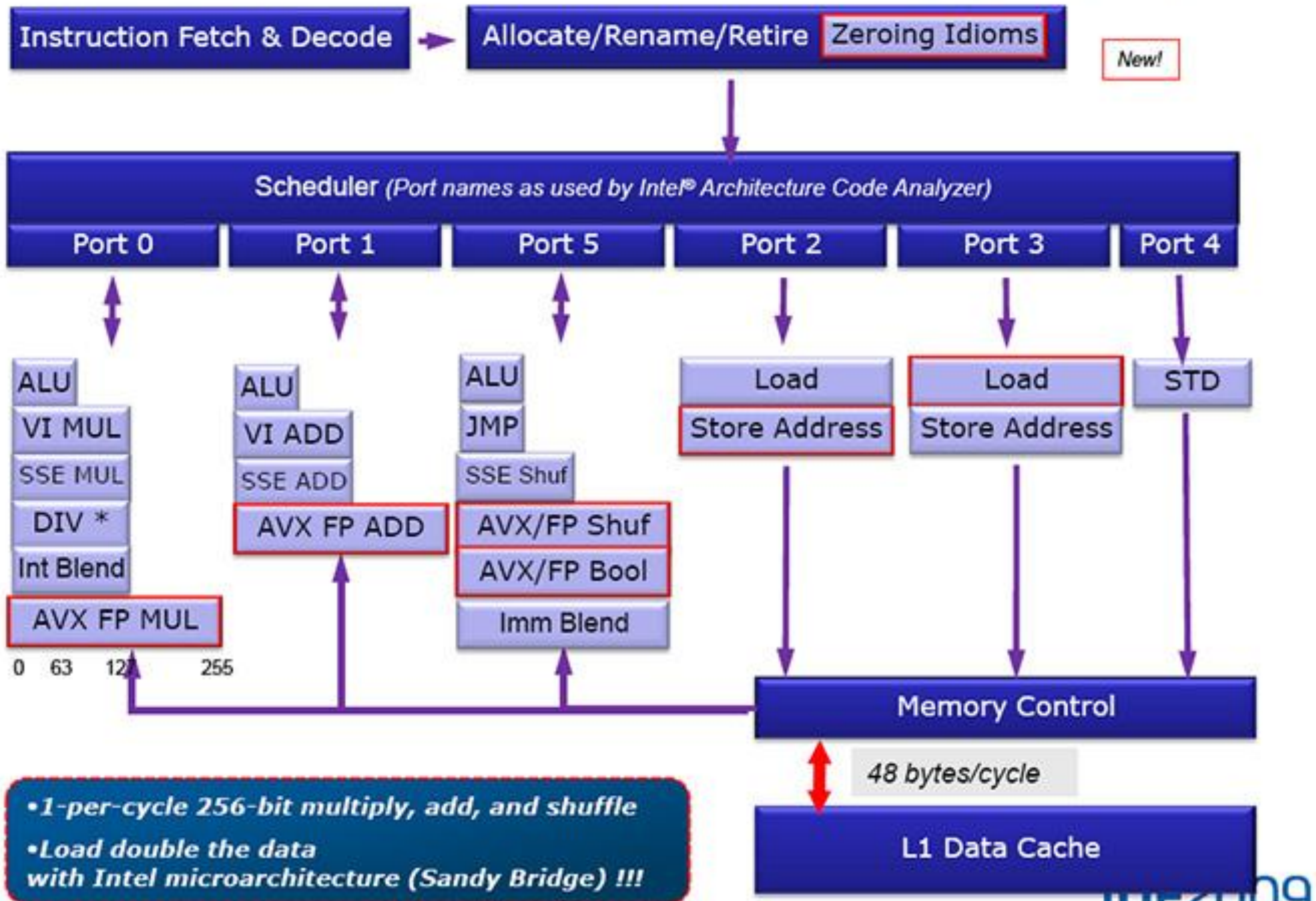
Martin Thompson - @mjpt777

Modern Hardware Overview

Modern Hardware (Intel Sandy Bridge)



Intel® Microarchitecture (Sandy Bridge) Highlights

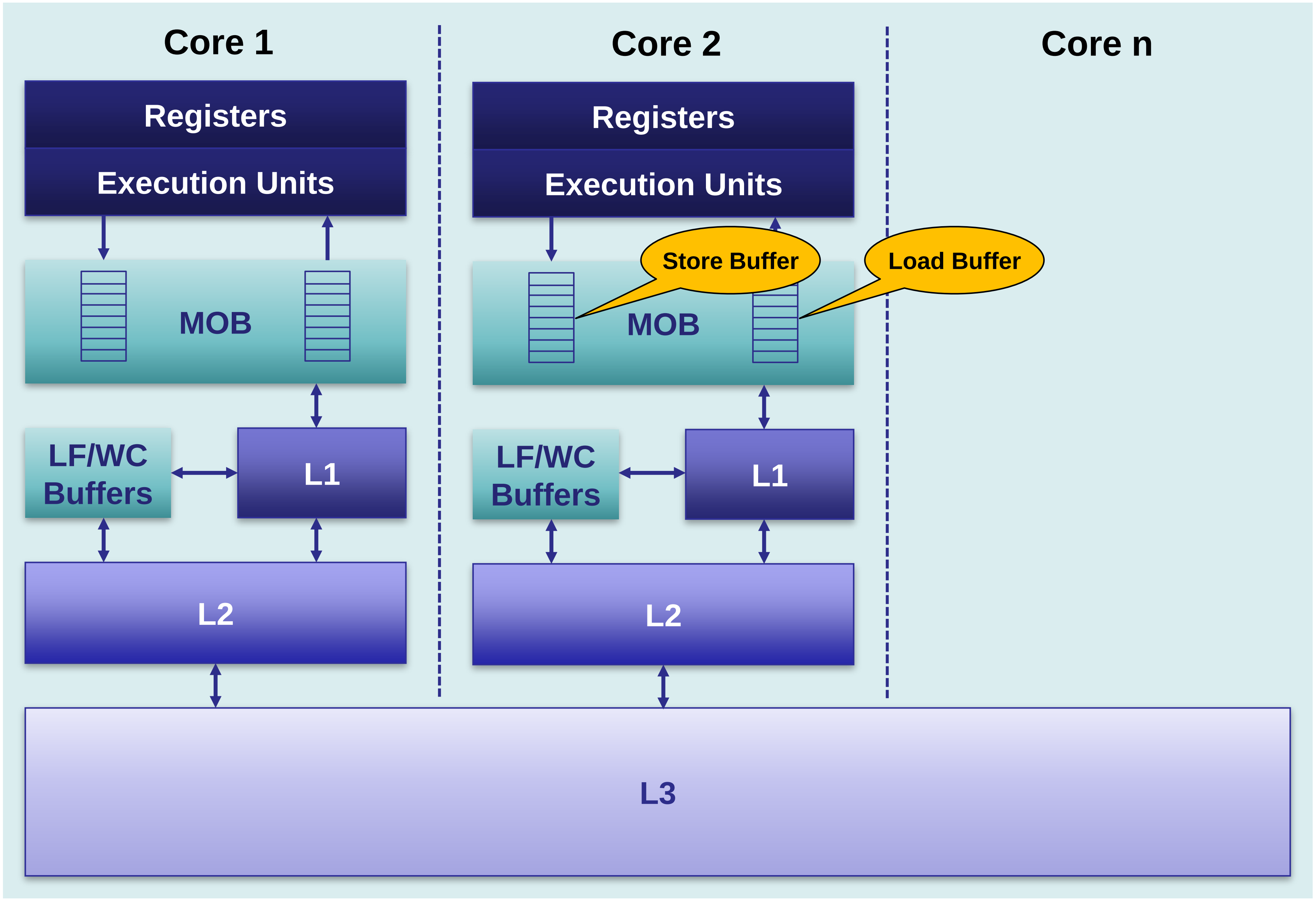


- *1-per-cycle 256-bit multiply, add, and shuffle*
- *Load double the data with Intel microarchitecture (Sandy Bridge) !!!*

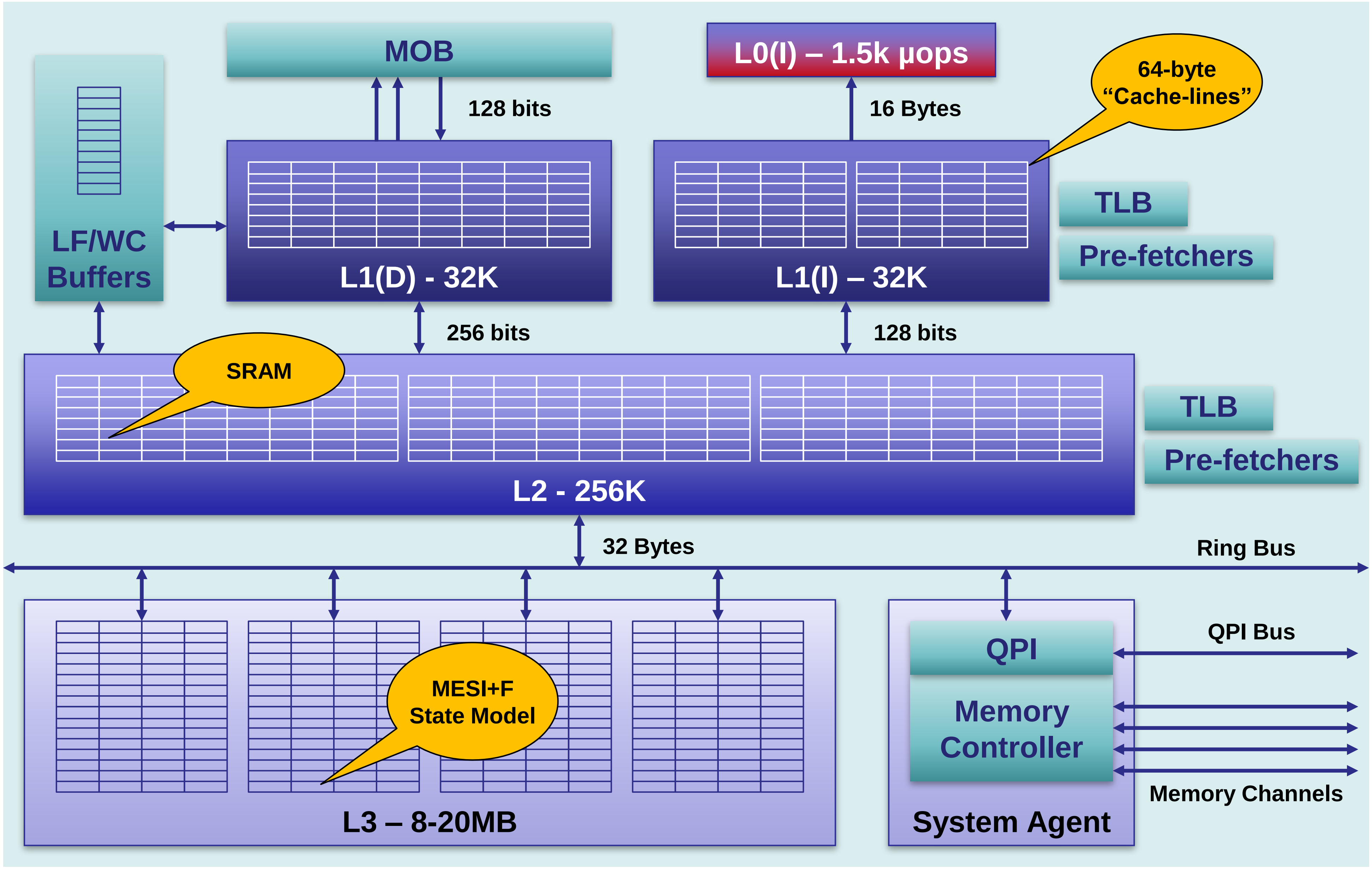
* Not fully pipelined

IDF2009 INTEL DEVELOPER FORUM

Memory Ordering



Cache Structure & Coherence



Memory Models

Hardware Memory Models

Memory consistency models describe how threads may interact through shared memory consistently.

- **Program Order (PO) for a single thread**
- **Sequential Consistency (SC) [Lamport 1979]**
  - **What you expect a program to do! (for race free)**
- **Strict Consistency (Linearizability)**
  - **Some special instructions**
- **Total Store Order (TSO)**
  - **Sparc model that is stronger than SC**
- **x86/64 is TSO + (Total Lock Order & Causal Consistency)**
  - **http://www.youtube.com/watch?v=WUfvvFD5tAA**
- **Other Processors can have weaker models**

Intel x86/64 Memory Model

http://www.multicoreinfo.com/research/papers/2008/damp08-intel64.pdf

http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html

1. **Loads are not reordered with other loads.**
2. **Stores are not reordered with other stores.**
3. **Stores are not reordered with older loads.**
4. **Loads may be reordered with older stores to different locations but not with older stores to the same location.**
5. **In a multiprocessor system, memory ordering obeys causality (memory ordering respects transitive visibility).**
6. **In a multiprocessor system, stores to the same location have a total order.**
7. **In a multiprocessor system, locked instructions have a total order.**
8. **Loads and stores are not reordered with locked instructions.**

Language/Runtime Memory Models

Some languages/Runtimes have a well defined memory model for portability:

- **Java Memory Model (Java 5)**
- **C++ 11**
- **Erlang**
- **Go**

For most other languages we are at the mercy of the compiler

- **Instruction reordering**
- **C “volatile” is inadequate**
- **Register allocation for caching values**
- **No mapping to the hardware memory model**
- **Fences/Barriers need to be applied**

Measuring What Is Going On

Model Specific Registers (MSR)

Many and varied uses

- Timestamp Invariant Counter
- Memory Type Range Registers

Performance Counters!!!

- L2/L3 Cache Hits/Misses
- TLB Hits/Misses
- QPI Transfer Rates
- Instruction and Cycle Counts
- Lots of others....

Accessing MSRs

```
void rdmsr(uint32_t msr, uint32_t* lo, uint32_t* hi)
{
    asm volatile("rdmsr" : "=a" lo, "=d" hi : "c" msr);
}

void wrmsr(uint32_t msr, uint32_t lo, uint32_t hi)
{
    asm volatile("wrmsr" :: "c" msr, "a" lo, "d" hi);
}
```

Accessing MSRs On Linux

```
f = new RandomAccessFile(“/dev/cpu/0/msr”, “rw”);
ch = f.getChannel();
buffer.order(ByteOrder.nativeOrder());
ch.read(buffer, msrNumber);
long value = buffer.getLong(0);
```

Accessing MSRs Made Easy!

Intel VTune

- http://software.intel.com/en-us/intel-vtune-amplifier-xe

Linux "perf stat"

- http://linux.die.net/man/1/perf-stat

likwid - Lightweight Performance Counters

- http://code.google.com/p/likwid/

Biggest Performance Enemy

"Contention!"

Contention

- Managing Contention
  - > Locks
  - > CAS Techniques
- Little’s & Amdahl’s Laws
  - > $L = \lambda W$
  - > *Sequential Component Constraint*

- Single Writer Principle
- Shared Nothing Designs

Locks

Software Locks

- Mutex, Semaphore, Critical Section, etc.
  - > What happens when un-contended?
  - > What happens when contention occurs?
  - > What if we need condition variables?
  - > What are the cost of software locks?
  - > Can they be optimised?

Hardware Locks

- Atomic Instructions
  - Compare And Swap/Set
  - Lock instructions on x86
    - `LOCK XADD` is a bit special

- Used to update *sequences* and *pointers*
- What are the costs of these operations?

- Guess how software locks are created?

- TSX (Transactional Synchronization Extensions)

Let’s Look At A Lock-Free Algorithm

OneToOneQueue – Take 1

```
public final class OneToOneConcurrentArrayQueue<E>
    implements Queue<E>
{
    private final E[] buffer;

    private volatile long tail = 0;
    private volatile long head = 0;

    public OneToOneConcurrentArrayQueue(int capacity)
    {
        buffer = (E[])new Object[capacity];
    }
```

OneToOneQueue – Take 1

```
public boolean offer(final E e)
{
    final long currentTail = tail;
    final long wrapPoint = currentTail - buffer.length;
    if (head <= wrapPoint)
    {
        return false;
    }

    buffer[(int)(currentTail % buffer.length)] = e;

    tail = currentTail + 1;

    return true;
}
```

OneToOneQueue – Take 1

```
public E poll()
{
    final long currentHead = head;
    if (currentHead >= tail)
    {
        return null;
    }

    final int index =
        (int)(currentHead % buffer.length);
    final E e = buffer[index];
    buffer[index] = null;

    head = currentHead + 1;

    return e;
}
```

Concurrent Queue Performance Results

	Ops/Sec (Millions)	Mean Latency (ns)
LinkedBlockingQueue	4.3	~32,000 / ~500
ArrayBlockingQueue	3.5	~32,000 / ~600
ConcurrentLinkedQueue	13	NA / ~180
ConcurrentArrayQueue	13	NA / ~150

Note: None of these tests are run with thread affinity set, Sandy Bridge 2.4 GHz

Latency: Blocking - put() & take() / Non-Blocking - offer() & poll()

Let's Apply Some *"Mechanical Sympathy"*

Mechanical Sympathy In Action

Knowing the cost of operations

- Remainder Operation
- Volatile writes and lock instructions

Why so many cache misses?

- False Sharing
- Algorithm Opportunities
  - > “Smart Batching”
- Memory layout

Operation Costs

Signalling

```
// Lock
pthread_mutex_lock(&lock);
sequence = i;
pthread_cond_signal(&condition);
pthread_mutex_unlock(&lock);

// Soft Barrier
asm volatile(“” ::: “memory”);
sequence = i;

// Fence
asm volatile(“” ::: “memory”);
sequence = i;
asm volatile(“lock addl $0x0,(%rsp)”);
```

Signalling Costs

	Lock	Fence	Soft
Million Ops/Sec	9.4	45.7	108.1
L2 Hit Ratio	17.26	28.17	13.32
L3 Hit Ratio	0.78	29.60	27.99
Instructions	12846 M	906 M	801 M
CPU Cycles	28278 M	5808 M	1475 M
Ins/Cycle	0.45	0.16	0.54

OneToOneQueue – Take 2

```
public final class OneToOneConcurrentArrayQueue2<E>
    implements Queue<E>
{
    private final int mask;
    private final E[] buffer;

    private final AtomicLong tail = new AtomicLong(0);
    private final AtomicLong head = new AtomicLong(0);

    public OneToOneConcurrentArrayQueue2(int capacity)
    {
        capacity = findNextPositivePowerOfTwo(capacity);
        mask = capacity - 1;
        buffer = (E[])new Object[capacity];
    }
```

OneToOneQueue – Take 2

```
public boolean offer(final E e)
{
    final long currentTail = tail.get();
    final long wrapPoint = currentTail - buffer.length;
    if (head.get() <= wrapPoint)
    {
        return false;
    }

    buffer[(int)currentTail & mask] = e;
    tail.lazySet(currentTail + 1);

    return true;
}
```

OneToOneQueue – Take 2

```
public E poll()
{
    final long currentHead = head.get();
    if (currentHead >= tail.get())
    {
        return null;
    }

    final int index = (int)currentHead & mask;
    final E e = buffer[index];
    buffer[index] = null;
    head.lazySet(currentHead + 1);

    return e;
}
```

Concurrent Queue Performance Results

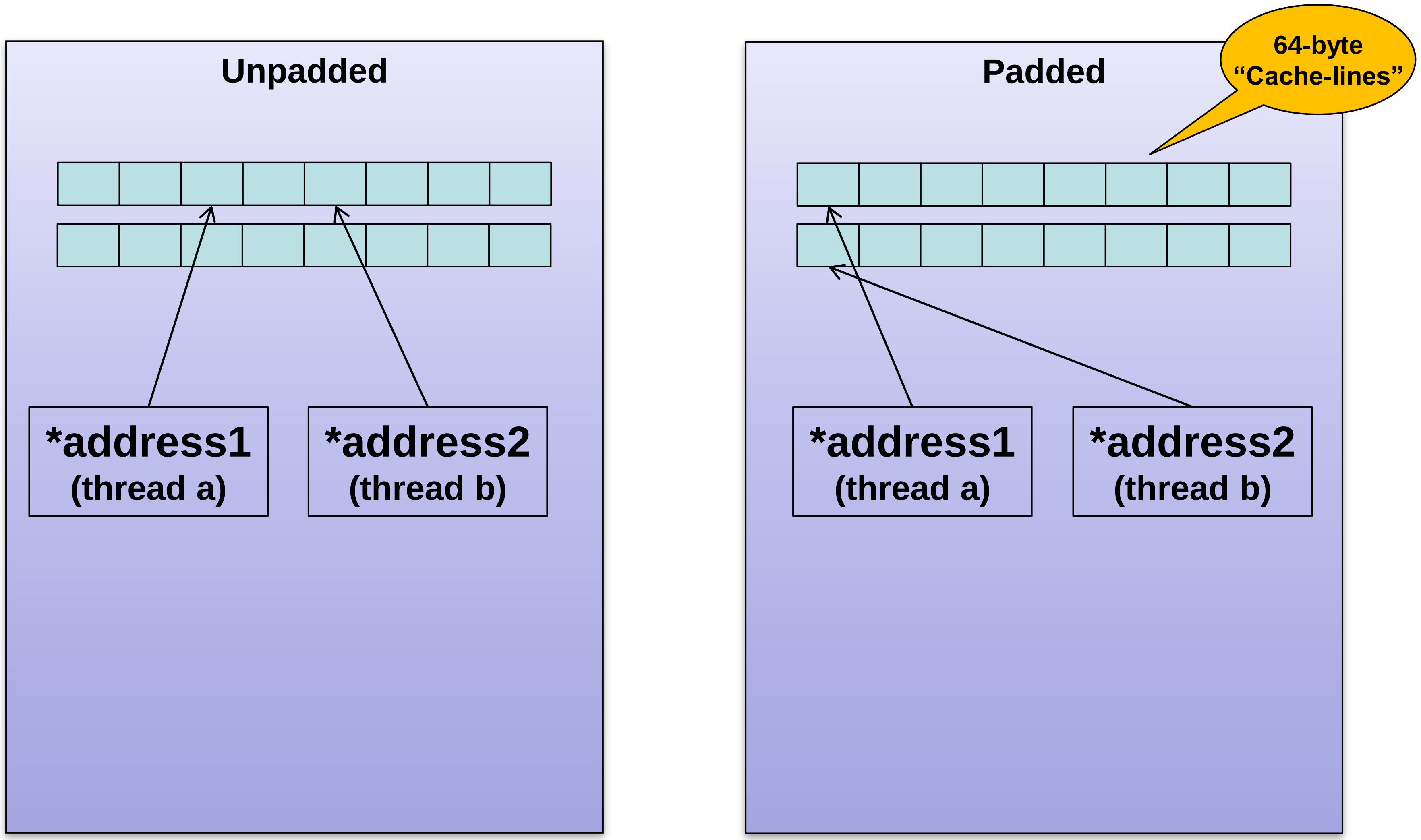
	Ops/Sec (Millions)	Mean Latency (ns)
LinkedBlockingQueue	4.3	~32,000 / ~500
ArrayBlockingQueue	3.5	~32,000 / ~600
ConcurrentLinkedQueue	13	NA / ~180
ConcurrentArrayQueue	13	NA / ~150
ConcurrentArrayQueue2	45	NA / ~120

Note: None of these tests are run with thread affinity set, Sandy Bridge 2.4 GHz

Latency: Blocking - put() & take() / Non-Blocking - offer() & poll()

Cache Misses

False Sharing and Cache Lines



False Sharing Testing

```
int64_t* address = seq->address

for (int i = 0; i < ITERATIONS; i++)
{
    int64_t value = *address;
    ++value;
    *address = value;
    asm volatile("lock addl 0x0,(%rsp)");
}
```

False Sharing Test Results

	Unpadded	Padded
Million Ops/sec	12.4	104.9
L2 Hit Ratio	1.16%	23.05%
L3 Hit Ratio	2.51%	39.18%
Instructions	4559 M	4508 M
CPU Cycles	63480 M	7551 M
Ins/Cycle Ratio	0.07	0.60

OneToOneQueue – Take 3

```
public final class OneToOneConcurrentArrayQueue3<E>
    implements Queue<E>
{
    private final int capacity;
    private final int mask;
    private final E[] buffer;

    private final AtomicLong tail = new PaddedAtomicLong(0);
    private final AtomicLong head = new PaddedAtomicLong(0);

    public static class PaddedLong
    {
        public long value = 0, p1, p2, p3, p4, p5, p6;
    }

    private final PaddedLong tailCache = new PaddedLong();
    private final PaddedLong headCache = new PaddedLong();
```

OneToOneQueue – Take 3

```
public boolean offer(final E e)
{
    final long currentTail = tail.get();
    final long wrapPoint = currentTail - capacity;
    if (headCache.value <= wrapPoint)
    {
        headCache.value = head.get();
        if (headCache.value <= wrapPoint)
        {
            return false;
        }
    }

    buffer[(int)currentTail & mask] = e;
    tail.lazySet(currentTail + 1);

    return true;
}
```

OneToOneQueue – Take 3

```java
public E poll()
{
    final long currentHead = head.get();
    if (currentHead >= tailCache.value)
    {
        tailCache.value = tail.get();
        if (currentHead >= tailCache.value)
        {
            return null;
        }
    }

    final int index = (int)currentHead & mask;
    final E e = buffer[index];
    buffer[index] = null;
    head.lazySet(currentHead + 1);

    return e;
}
```

Concurrent Queue Performance Results

	Ops/Sec (Millions)	Mean Latency (ns)
LinkedBlockingQueue	4.3	~32,000 / ~500
ArrayBlockingQueue	3.5	~32,000 / ~600
ConcurrentLinkedQueue	13	NA / ~180
ConcurrentArrayQueue	13	NA / ~150
ConcurrentArrayQueue2	45	NA / ~120
ConcurrentArrayQueue3	150	NA / ~100

Note: None of these tests are run with thread affinity set, Sandy Bridge 2.4 GHz

Latency: Blocking - put() & take() / Non-Blocking - offer() & poll()

How Far Can We Go With Lock Free Algorithms?

Further Adventures With Lock-Free Algorithms

- State Machines
- CAS operations
- Wait-Free in addition to Lock-Free algorithms
- Thread Affinity
- x86 and busy spinning and back off

Questions?

Blog: http://mechanical-sympathy.blogspot.com/
Code: https://github.com/mjpt777/examples
Twitter: @mjpt777

"The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry."

- Henry Peteroski