HUAWEI

Enabling Performance on Modern Hardware with Practical Verification

Diogo Behrens

Huawei Dresden Research Center & Huawei Central Software Institute

Agenda

- Modern Hardware and Concurrent Programs
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  - VSync: dealing with Relaxed Memory Models (RMMs)
  - CNA on RMM: verifying next-gen Linux spinlock
  - CLoF: dealing with NUMA hierarchies and heterogeneity
  - BBQ: building highly-efficient ringbuffers
- Wrap up and Outlook

Agenda

- Modern Hardware and Concurrent Programs
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  - VSync: dealing with Relaxed Memory Models (RMMs)
  - CNA on RMM: verifying next-gen Linux spinlock
  - CLoF: dealing with NUMA hierarchies and heterogeneity
  - BBQ: building highly-efficient ringbuffers
- Wrap up and Outlook

Concurrency is everywhere

Operating systems, databases and server applications resort to **multicore concurrency** to achieve high performance.

Concurrency is everywhere

Operating systems, databases and server applications resort to **multicore concurrency** to achieve high performance.

. . . but brings challenges!

- **Notoriously hard to get right**
  - Non-deterministic thread interleaving

Concurrency is everywhere

Operating systems, databases and server applications resort to **multicore concurrency** to achieve high performance.

. . . but brings challenges!

- **Notoriously hard to get right**
  - Non-deterministic thread interleaving
- **Traditional testing isn't sufficient**
  - Testing is **unlikely** to trigger subtle concurrency bugs
  - Production is **likely** to trigger them, → many instances and long executions
- **Reproducing concurrency bugs is hard**

Modern hardware makes matters worse

Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V

Modern hardware makes matters worse

Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V

Huawei Unveils Industry's Highest-Performance ARM-based CPU

Bringing Global Computing Power to Next Level

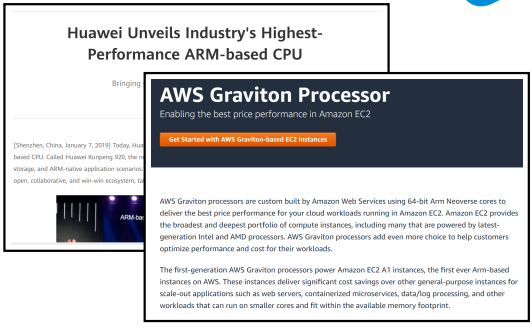
Jan 07, 2019

(Shenzhen, China, January 7, 2019) Today, Huawei announced the industry's highest-performance Advanced RISC Machine (ARM)-based CPU. Called Huawei Kunpeng 920, the new CPU is designed to boost the development of computing in big data, distributed storage, and ARM-native application scenarios. Huawei will join with industry players to advance the ARM industry and foster an open, collaborative, and win-win ecosystem, taking computing performance to new heights.

Modern hardware makes matters worse

Relaxed memory models (RMMs)

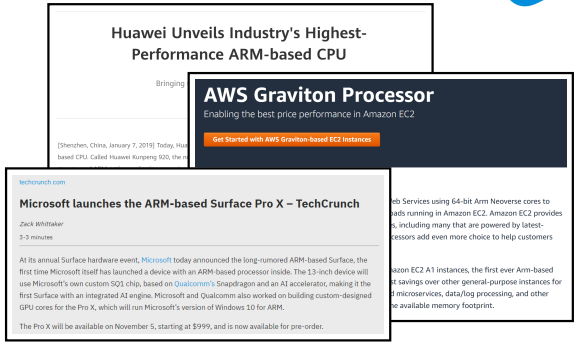
- Modern architectures becoming popular
  eg, Arm, RISC-V



Modern hardware makes matters worse

Relaxed memory models (RMMs)

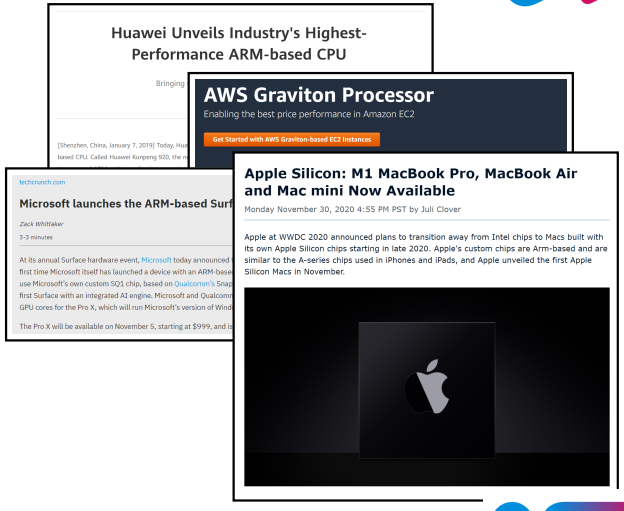
- Modern architectures becoming popular
  eg, Arm, RISC-V



Modern hardware makes matters worse

Relaxed memory models (RMMs)

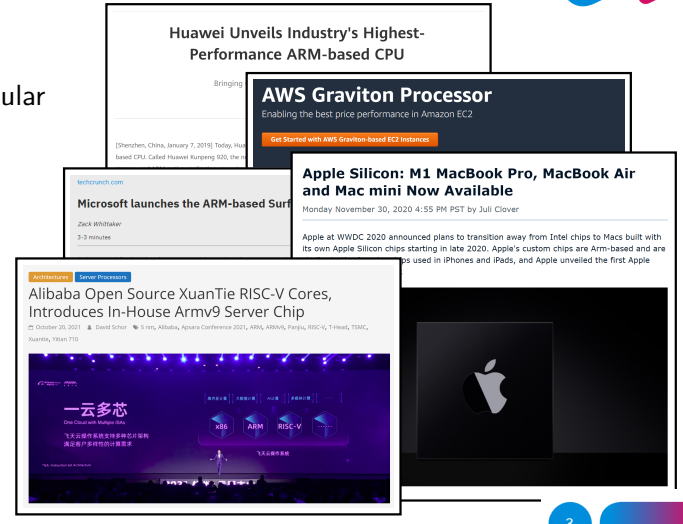
- Modern architectures becoming popular
  eg, Arm, RISC-V



Modern hardware makes matters worse

Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V



Modern hardware makes matters worse

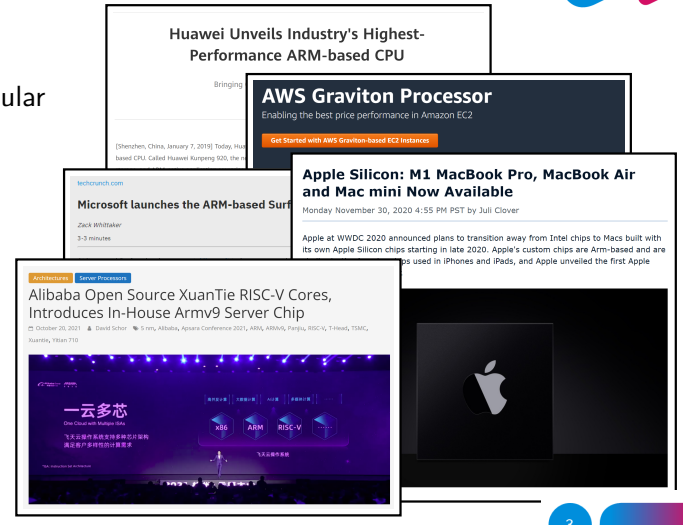
Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V
- **Aggressive reorderings** to improve performance
- **Much higher non-determinisim**
- Careful use of **memory barriers**
  (neither too many, nor too few)

Init

```
data = ctrl = 0;
```

	Thread 1	**Thread 2**
RMM reorder	`data = 1;`	`while(!ctrl) {}`
	`ctrl = 1;`	`assert(data == 1);` ✗



Do RMM bugs really happen?

Bugs found with our tools

Mutual exclusion bug: seL4 CLH lock

The seL4 microkernel is a flagship of formal verification. The big kernel lock implementation, however, was not verified and missed one acquire-release barrier.

Potential hang: DPDK MCS lock

The lock had one missing release barrier. An Arm engineer replied to our patch: “Unfortunately, memory ordering questions are hard topics. I have been discussing this internally [...], hope to conclude soon.” More than 3 months to accept the 1-line patch.

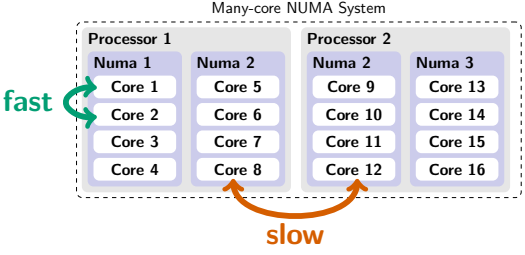
Crash: MariaDB lockfree hashtable

Due to missing barriers for Arm, data of a node can be accessed after the node has been deleted. That causes SEGFAULT crashes on high load workloads.

Diogo Behrens — diogo.behrens@huawei.com

Modern hardware makes matters worse

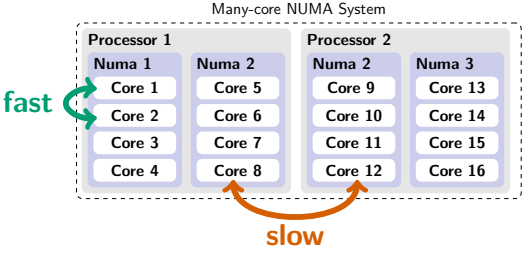
Modern hardware makes matters worse



Deep NUMA hierarchies

- Core **distance affects** shared-memory communication **performance**

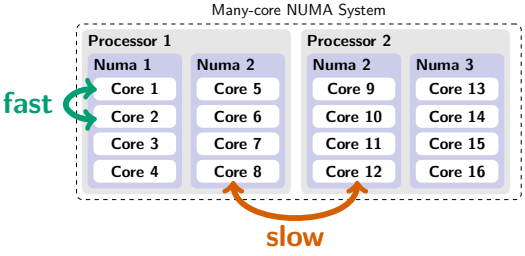
Modern hardware makes matters worse



Deep NUMA hierarchies

- Core **distance affects** shared-memory communication **performance**
- Concurrent algorithms must exploit that!

Modern hardware makes matters worse

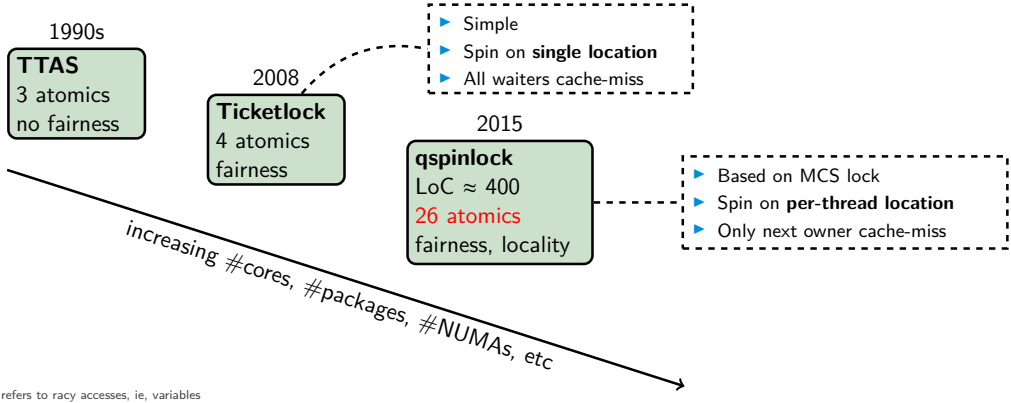


Deep NUMA hierarchies

- Core **distance affects** shared-memory communication **performance**
- Concurrent algorithms must exploit that!
- **Makes code even more complex**

As the hardware evolves, so does the concurrency control

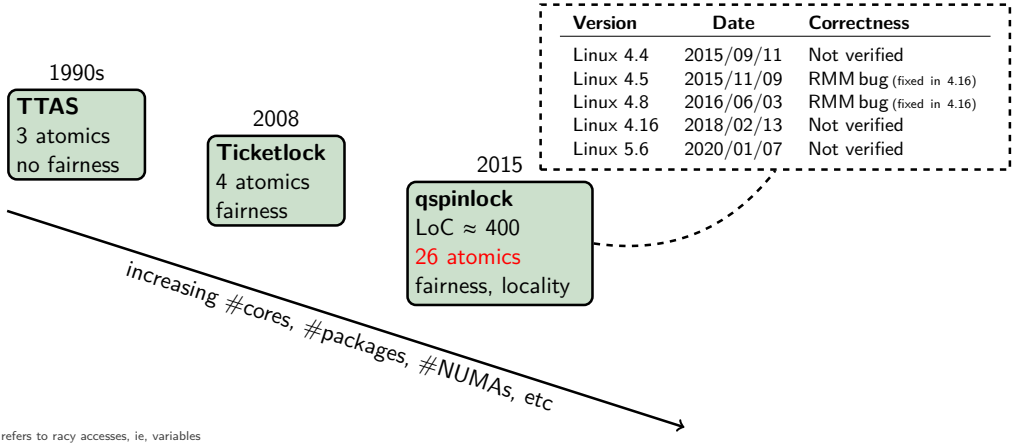
Example of Linux spinlock over the years



"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

As the hardware evolves, so does the concurrency control

Example of Linux spinlock over the years

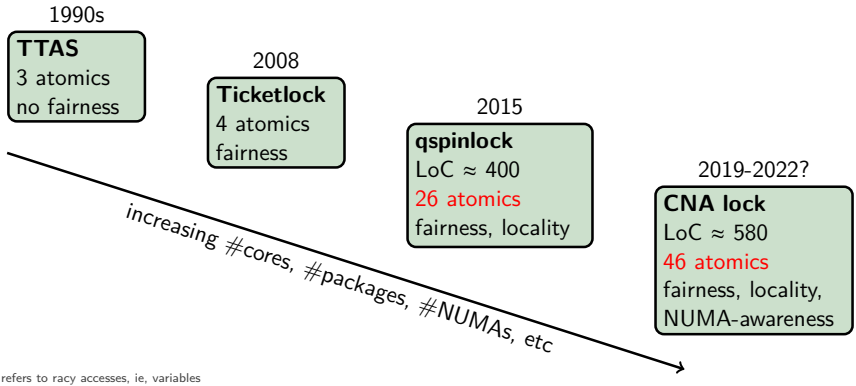


Version	Date	Correctness
Linux 4.4	2015/09/11	Not verified
Linux 4.5	2015/11/09	RMM bug (fixed in 4.16)
Linux 4.8	2016/06/03	RMM bug (fixed in 4.16)
Linux 4.16	2018/02/13	Not verified
Linux 5.6	2020/01/07	Not verified

"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

As the hardware evolves, so does the concurrency control

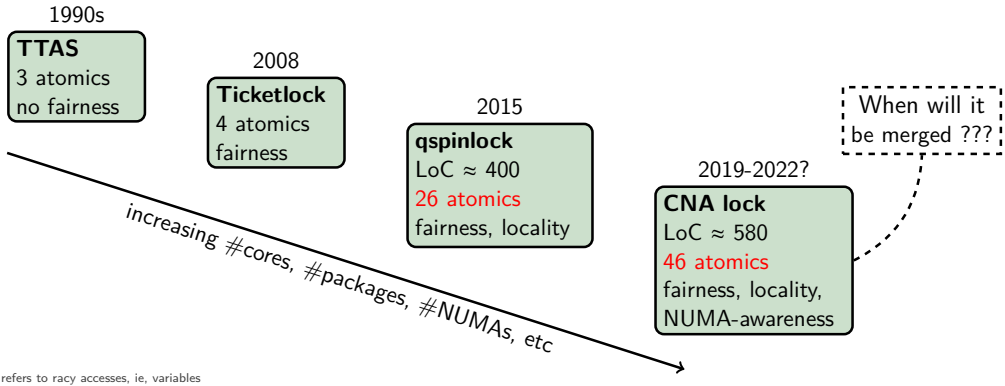
Example of Linux spinlock over the years



"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

As the hardware evolves, so does the concurrency control

Example of Linux spinlock over the years



"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

Diogo Behrens — diogo.behrens@huawei.com

What can the industry do?

Solution 1: overprotection

- Spray the code with memory barriers
- Simplify design as much as possible
- Cost: **performance impact**

Solution 2: expert design

- Hire highly-skilled engineers
- Carefully design and implement
- Cost: **error-prone, low maintainability**

Solution 3:
Enabling Performance with **Practical** Verification

What is **Practical** Verification?

In short: we don't know!

What is **Practical** Verification?

We know what formal verification is!

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking

What is **Practical** Verification?

We know what formal verification is!

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking
- In contrast to testing, **exhaustive:**
  - All possible interleavings
  - All possible reorderings
  - All possible contexts, ...

What is **Practical** Verification?

We know what formal verification is!

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking
- In contrast to testing, **exhaustive:**
  - All possible interleavings
  - All possible reorderings
  - All possible contexts, ...
- Problems:
  - **High level of expertise** required
  - Often **unrealistic assumptions** (eg, hardware simplifications)

What is **Practical** Verification?

We know what formal verification is!

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking
- In contrast to testing, **exhaustive:**
  - All possible interleavings
  - All possible reorderings
  - All possible contexts, ...
- Problems:
  - **High level of expertise** required
  - Often **unrealistic assumptions** (eg, hardware simplifications)

And practical verification?

- Goal: **increase confidence**
- Exhaustive as long **as it pays off**
- A concurrency counterpart of testing
- **Ideally any developer** can use it:
  - Push-button, fully-automated
  - No expertise required

What is **Practical** Verification?

We know what formal verification is!

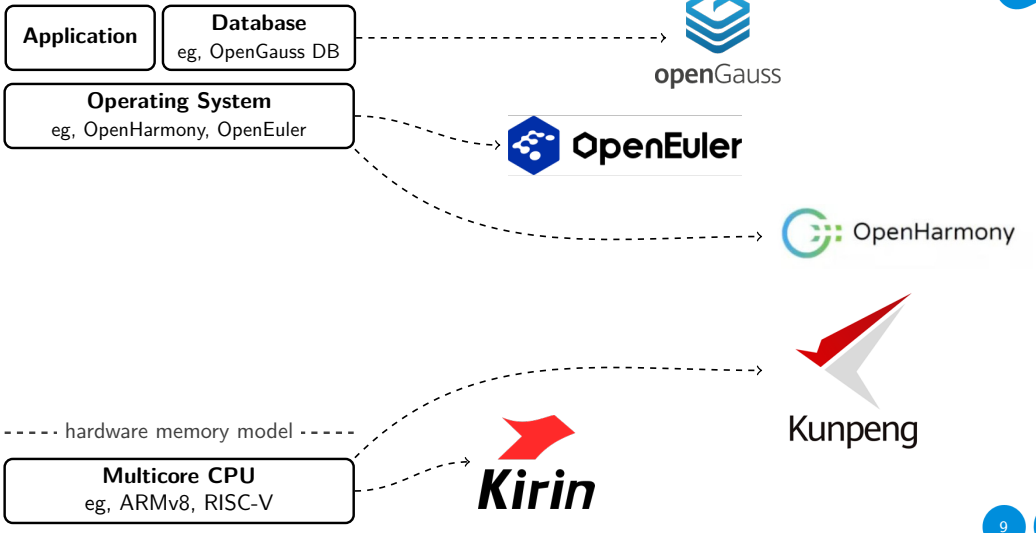
- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking
- In contrast to testing, **exhaustive:**
  - All possible interleavings
  - All possible reorderings
  - All possible contexts, ...
- Problems:
  - **High level of expertise** required
  - Often **unrealistic assumptions** (eg, hardware simplifications)

And practical verification?

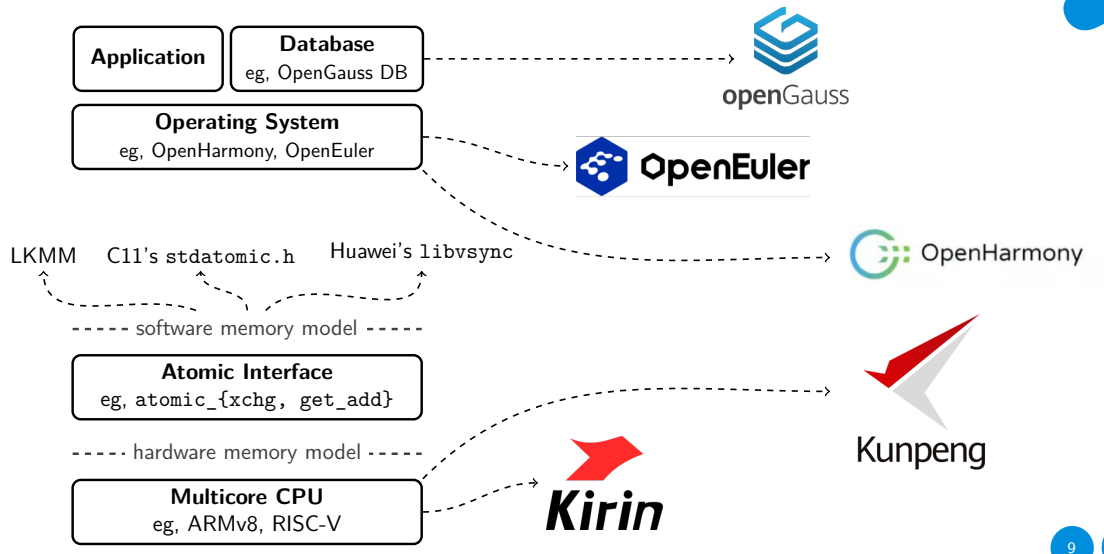
- Goal: **increase confidence**
- Exhaustive as long **as it pays off**
- A concurrency counterpart of testing
- **Ideally any developer** can use it:
  - Push-button, fully-automated
  - No expertise required
- **In practice, a few experts** must help
  - Defining what to check (properties)
  - Improving tool **scalability**
  - Making methods **more realistic**

Where do we apply practical verification?

Practical Verification Scope in Huawei



Practical Verification Scope in Huawei



Practical Verification Scope in Huawei

Application

Database
eg, OpenGauss DB

Operating System
eg, OpenHarmony, OpenEuler

Synchronization Primitives
eg, spinlock, mutex, RCU

----- software memory model -----

Atomic Interface
eg, `atomic_{xchg, get_add}`

----- hardware memory model -----

Multicore CPU
eg, ARMv8, RISC-V

Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();


    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Practical Verification Scope in Huawei

Application

Database
eg, OpenGauss DB

Operating System
eg, OpenHarmony, OpenEuler

Synchronization Primitives
eg, spinlock, mutex, RCU

----- software memory model -----

Atomic Interface
eg, `atomic_{xchg, get_add}`

----- hardware memory model -----

Multicore CPU
eg, ARMv8, RISC-V

Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();


    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

DPDK
DATA PLANE DEVELOPMENT KIT

seL4
Security. Performance. Proof.

Practical Verification Scope in Huawei

Application

Database
eg, OpenGauss DB

Operating System
eg, OpenHarmony, OpenEuler

Synchronization Primitives
eg, spinlock, mutex, RCU

Lockless Data Structures
eg, list, queue, hashtable, etc

----- software memory model -----

Atomic Interface
eg, `atomic_{xchg, get_add}`

----- hardware memory model -----

Multicore CPU
eg, ARMv8, RISC-V

Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

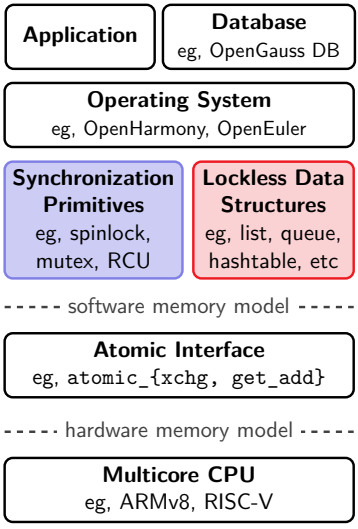
Asynchronous with data structures

```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

DPDK
DATA PLANE DEVELOPMENT KIT

seL4
Security. Performance. Proof.

Practical Verification Scope in Huawei



Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();


    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Asynchronous with data structures

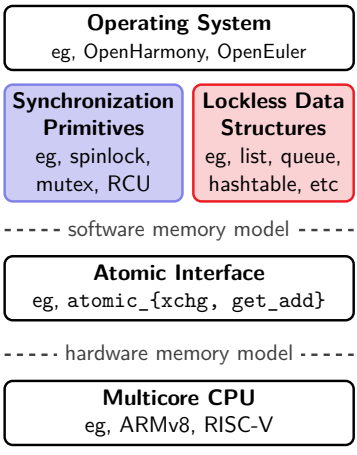
```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

MariaDB

DPDK
DATA PLANE DEVELOPMENT KIT

seL4
Security. Performance. Proof.

Practical Verification Scope in Huawei



Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();


    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Asynchronous with data structures

```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

Adhoc synchronization

```
if (atomic_add(counter) > MAX)
    do_something(data);
```


Diogo Behrens — diogo.behrens@huawei.com

Practical Verification Scope in Huawei

Operating System
eg, OpenHarmony, OpenEuler

Synchronization Primitives
eg, spinlock, mutex, RCU

Lockless Data Structures
eg, list, queue, hashtable, etc

----- software memory model -----

Atomic Interface
eg, `atomic_{xchg, get_add}`

----- hardware memory model -----

Multicore CPU
eg, ARMv8, RISC-V

Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Asynchronous with data structures

```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

Adhoc synchronization

```
if (atomic_add(counter) > MAX)
    do_something(data);
```


Diogo Behrens — diogo.behrens@huawei.com

Practical Verification Scope in Huawei

- **Sequential code**
  - It can be tested! Good news!
  - Even the critical section code
- **Concurrent code**
  - Cannot be easily tested
  - Barrier placement is hard
  - Hardware-awareness is hard
  - Scope of practical verification!

Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();


    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Asynchronous with data structures

```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

Adhoc synchronization

```
if (atomic_add(counter) > MAX)
    do_something(data);
```

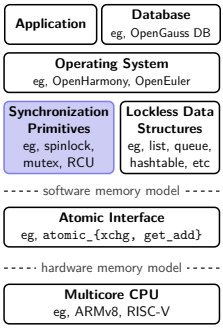

Diogo Behrens — diogo.behrens@huawei.com

Agenda

- Modern Hardware and Concurrent Programs
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  - VSync: dealing with Relaxed Memory Models (RMMs)
  - CNA on RMM: verifying next-gen Linux spinlock
  - CLoF: dealing with NUMA hierarchies and heterogeneity
  - BBQ: building highly-efficient ringbuffers
- Wrap up and Outlook

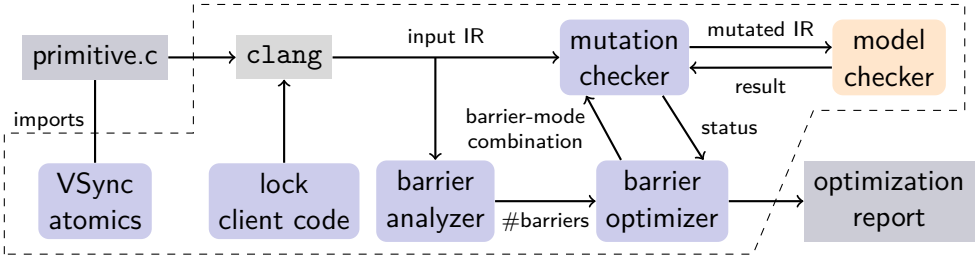
Agenda

- Modern Hardware and Concurrent Programs
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  - VSync: dealing with Relaxed Memory Models (RMMs)
  - CNA on RMM: verifying next-gen Linux spinlock
  - CLoF: dealing with NUMA hierarchies and heterogeneity
  - BBQ: building highly-efficient ringbuffers
- Wrap up and Outlook

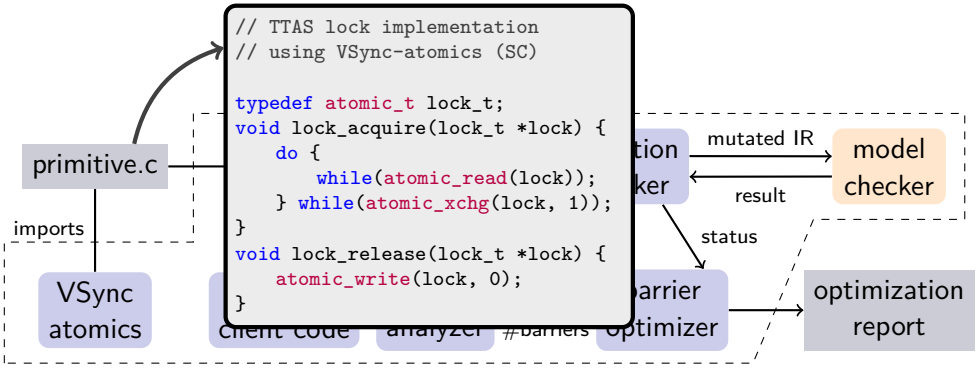


VSync: Push-button Verification/Optimization on RMMs

- VSync: https://dl.acm.org/doi/abs/10.1145/3445814.3446748 — **Best paper** @ ASPLOS'21
  by Oberhauser, Chehab, Behrens, Fu, Paolillo, Oberhauser, Bhat, Wen, Chen, Kim, Vafeiadis
- Making relaxed memory models fair: https://dl.acm.org/doi/abs/10.1145/3485475 — **Best paper** @ OOPSLA'21
  by Lahav, Namakonov, Oberhauser, Podkopaev, Vafeiadis

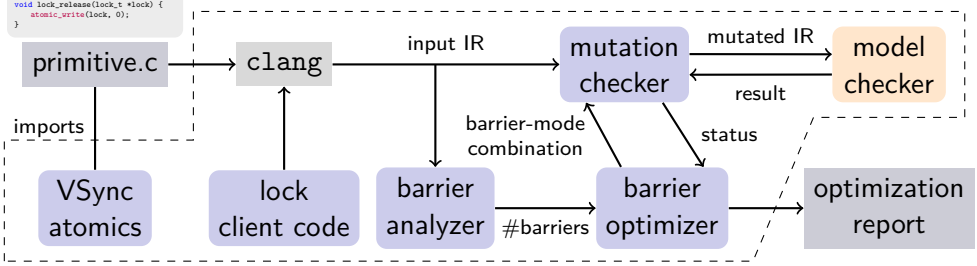


VSync: Push-button Verification/Optimization on RMMs



VSync: Push-button Verification/Optimization on RMMs

```
// TTAS lock implementation
// using VSync-atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(atomic_read(lock));
    } while(atomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    atomic_write(lock, 0);
}
```



VSync: Push-button Verification/Optimization on RMMs

```
// TTAS lock implementation
// using VSync-atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(atomic_read(lock));
    } while(atomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    atomic_write(lock, 0);
}
```

primitive.c → cl

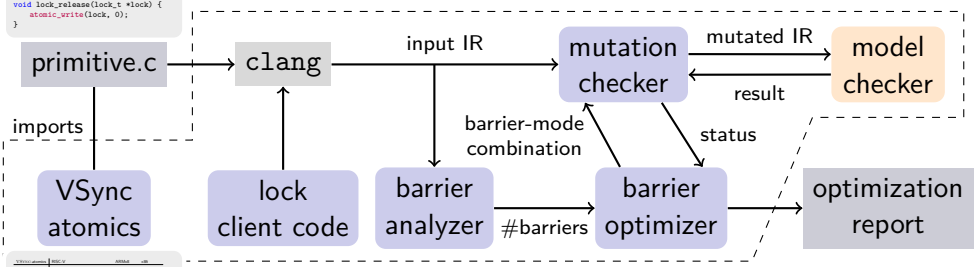
imports

VSync-atomics	RISC-V	ARMv8	x86
atomic_xchg	amoswap.w.aq.rl.sc	LDAXR;STLXR	XCHG
atomic_xchg_rel	amoswap.w.rl	LDXR;STLXR	XCHG
atomic_xchg_acq	amoswap.w.aq	LDAXR;STXR	XCHG
atomic_xchg_rlx	amoswap.w	LDXR;STXR	XCHG
atomic_read	fence [rw,rw]; lw; fence [r,rw]	LDAR	MOV
atomic_read_acq	lw; fence [r,rw]	LDAR	MOV
atomic_read_rlx	lw	LDR	MOV
atomic_write	fence [rw,w]; sw; fence [rw,rw]	STLR	MOV;MFENCE
atomic_write_rel	fence [rw,w]; sw	STLR	MOV
atomic_write_rlx	sw	STR	MOV
atomic_fence	fence [rw,rw]	DMB ISH	MFENCE
atomic_fence_acq	fence [r,rw]	DMB ISHLD	NOP
atomic_fence_rel	fence [rw,w]	DMB ISH	NOP
atomic_fence_rlx	nop	NOP	NOP

VSync atomics | lock client code | barrier analyzer → #barriers → barrier optimizer → optimization report

VSync: Push-button Verification/Optimization on RMMs

```
// TTAS lock implementation
// using VSync-atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(atomic_read(lock));
    } while(atomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    atomic_write(lock, 0);
}
```

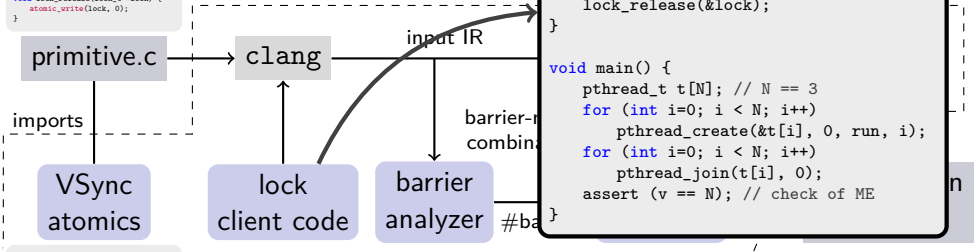


Diogo Behrens — diogo.behrens@huawei.com

VSync: Push-button Verification/Optimization on RMMs

```c
// TTAS lock implementation
// using VSync-atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(atomic_read(lock));
    } while(atomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    atomic_write(lock, 0);
}
```

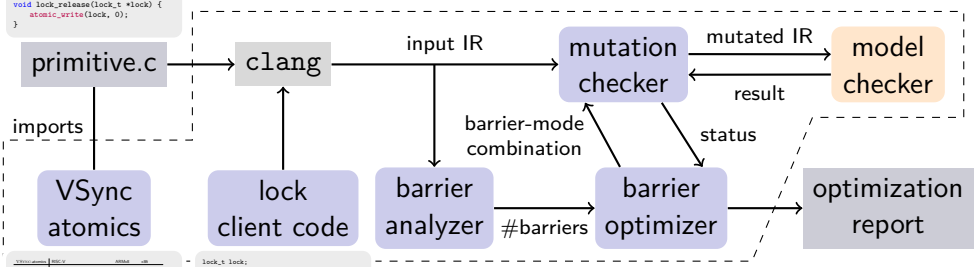
```c
lock_t lock;
int v = 0; // shared state
void run(int id) {
    lock_acquire(&lock);
    v++; // critical sec.
    lock_release(&lock);
}

void main() {
    pthread_t t[N]; // N == 3
    for (int i=0; i < N; i++)
        pthread_create(&t[i], 0, run, i);
    for (int i=0; i < N; i++)
        pthread_join(t[i], 0);
    assert (v == N); // check of ME
}
```



Diogo Behrens — diogo.behrens@huawei.com

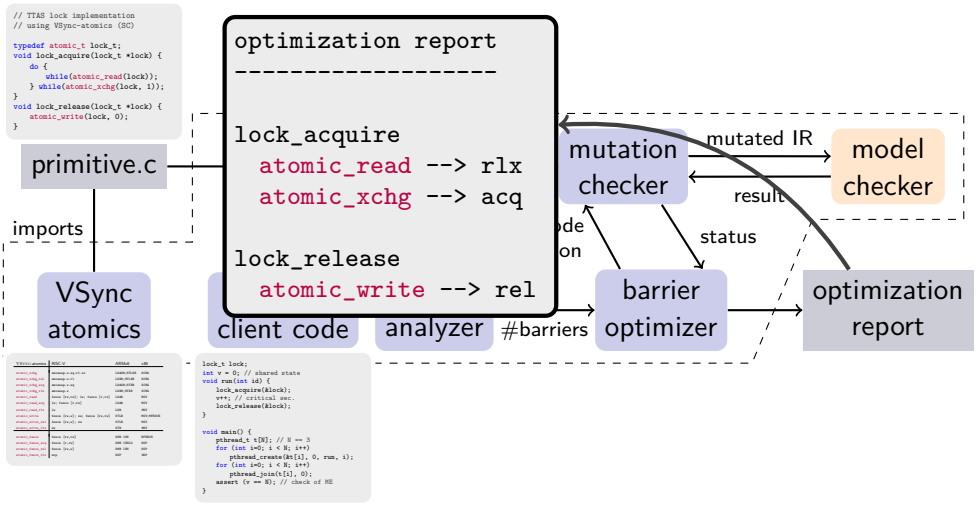
VSync: Push-button Verification/Optimization on RMMs

```
// TTAS lock implementation
// using VSync-atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(atomic_read(lock));
    } while(atomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    atomic_write(lock, 0);
}
```




```
lock_t lock;
int v = 0; // shared state
void run(int id) {
    lock_acquire(&lock);
    v++; // critical sec.
    lock_release(&lock);
}

void main() {
    pthread_t t[N]; // N == 3
    for (int i=0; i < N; i++)
        pthread_create(&t[i], 0, run, i);
    for (int i=0; i < N; i++)
        pthread_join(t[i], 0);
    assert (v == N); // check of ME
}
```

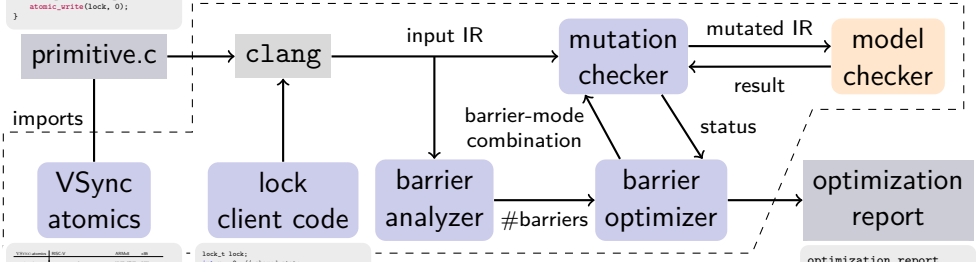
VSync: Push-button Verification/Optimization on RMMs



VSync: Push-button Verification/Optimization on RMMs

```
// TTAS lock implementation
// using VSync-atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(atomic_read(lock));
    } while(atomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    atomic_write(lock, 0);
}
```

Problem: Some optimizations cause hangs on Arm CPUS! Why?

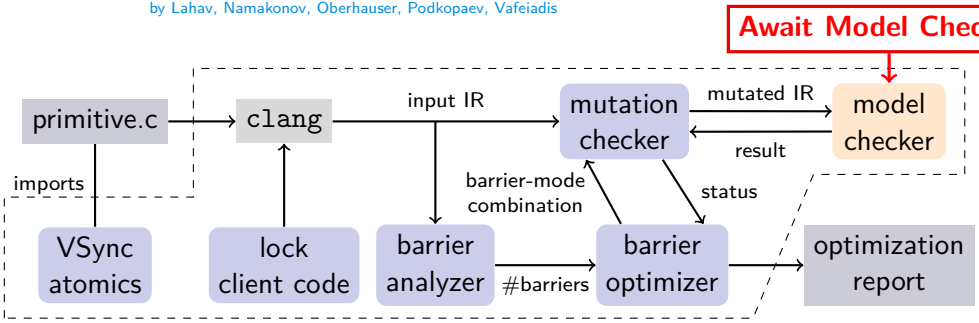



```
lock_t lock;
int v = 0; // shared state
void run(int id) {
    lock_acquire(&lock);
    ++v; // critical sec.
    lock_release(&lock);
}

void main() {
    pthread_t t[N]; // N == 3
    for (int i=0; i < N; i++)
        pthread_create(&t[i], 0, run, i);
    for (int i=0; i < N; i++)
        pthread_join(t[i], 0);
    assert (v == N); // check of ME
}
```

```
optimization report
-------------------

lock_acquire
  atomic_read --> rlx
  atomic_xchg --> acq

lock_release
  atomic_write --> rel
```

VSync: Push-button Verification/Optimization on RMMs

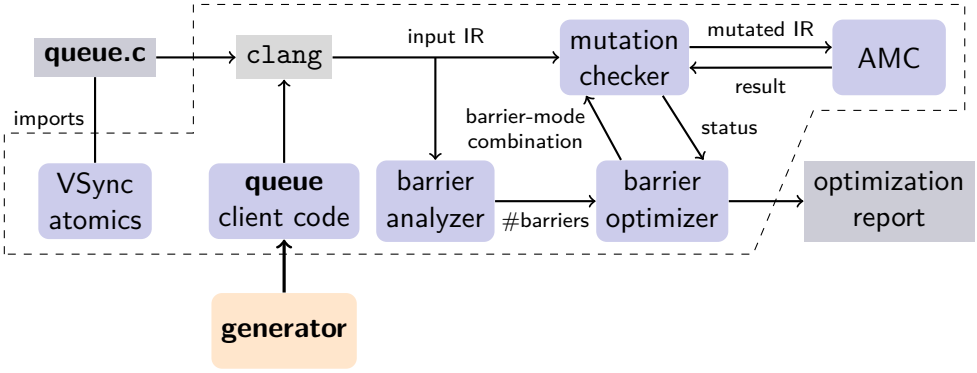
- VSync: https://dl.acm.org/doi/abs/10.1145/3445814.3446748 — **Best paper** @ ASPLOS'21
  by Oberhauser, Chehab, Behrens, Fu, Paolillo, Oberhauser, Bhat, Wen, Chen, Kim, Vafeiadis
- Making relaxed memory models fair: https://dl.acm.org/doi/abs/10.1145/3485475 — **Best paper** @ OOPSLA'21
  by Lahav, Namakonov, Oberhauser, Podkopaev, Vafeiadis



No scalable model checker for RMM was capable of checking termination of spinloops!

VSync: Push-button Verification/Optimization on RMMs

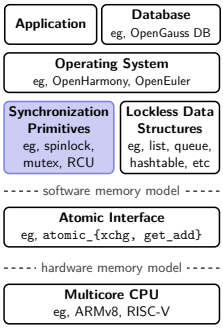
Current work: extend VSync to data structures
eg, queues, lists, stacks



Diogo Behrens — diogo.behrens@huawei.com

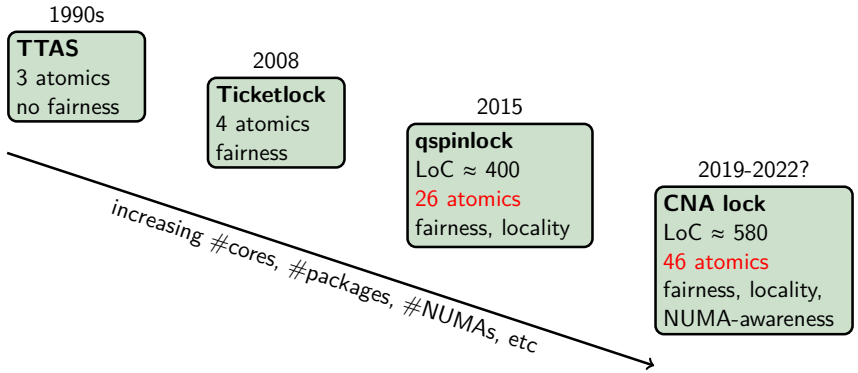
Agenda

- Modern Hardware and Concurrent Programs
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  - VSync: dealing with Relaxed Memory Models (RMMs)
  - CNA on RMM: verifying next-gen Linux spinlock
  - CLoF: dealing with NUMA hierarchies and heterogeneity
  - BBQ: building highly-efficient ringbuffers
- Wrap up and Outlook



Verifying CNA-based Linux qspinlock on LKMM

LKMM = Linux Kernel Memory Model



"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

Diogo Behrens — diogo.behrens@huawei.com

Verifying CNA-based Linux qspinlock on LKMM

Technical report @ arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer
https://arxiv.org/abs/2111.15240

- **Goal**:
  - Support merge of CNA patch (Oracle)
- **Verification:**
  - Linux Kernel Memory Model (LKMM)
  - Armv8 memory model
  - Power memory model
- **Verification time:**
  - With 4 threads, from 12h to 15h

Verifying CNA-based Linux qspinlock on LKMM

Technical report @ arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer
https://arxiv.org/abs/2111.15240

- **Goal**:
  - Support merge of CNA patch (Oracle)
- **Verification:**
  - Linux Kernel Memory Model (LKMM)
  - Armv8 memory model
  - Power memory model
- **Verification time:**
  - With 4 threads, from 12h to 15h

- **Linux qspinlock and CNA:**
  - **correct on Armv8**
  - correct on Power

Verifying CNA-based Linux qspinlock on LKMM

Technical report @ arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer
https://arxiv.org/abs/2111.15240

- **Goal**:
  - Support merge of CNA patch (Oracle)
- **Verification:**
  - Linux Kernel Memory Model (LKMM)
  - Armv8 memory model
  - Power memory model
- **Verification time:**
  - With 4 threads, from 12h to 15h

- **Linux qspinlock and CNA:**
  - **correct on Armv8**
  - correct on Power
  - **incorrect on LKMM**!
- **Lessons learned**
  - Was the model checker broken? No!
  - LKMM mismatches reality!
    (**expert support**)

Diogo Behrens — diogo.behrens@huawei.com

Verifying CNA-based Linux qspinlock on LKMM

Technical report @ arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer
https://arxiv.org/abs/2111.15240

- **Goal**:
  - Support merge of CNA patch (Oracle)
- **Verification:**
  - Linux Kernel Memory Model (LKMM)
  - Armv8 memory model
  - Power memory model
- **Verification time:**
  - With 4 threads, from 12h to 15h

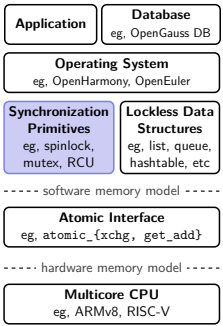
- **Linux qspinlock and CNA:**
  - **correct on Armv8**
  - correct on Power
  - **incorrect on LKMM**!
- **Lessons learned**
  - Was the model checker broken? No!
  - LKMM mismatches reality! (**expert support**)
  - Having 2 model checkers is helpful!

https://github.com/hernanponcedeleon/Dat3M
https://github.com/MPI-SWS/genmc

Diogo Behrens — diogo.behrens@huawei.com

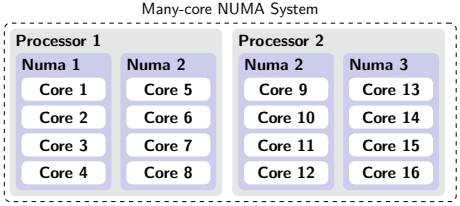
Agenda

- Modern Hardware and Concurrent Programs
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  - VSync: dealing with Relaxed Memory Models (RMMs)
  - CNA on RMM: verifying next-gen Linux spinlock
  - CLoF: dealing with NUMA hierarchies and heterogeneity
  - BBQ: building highly-efficient ringbuffers
- Wrap up and Outlook



Why would locks be too complex to model check?

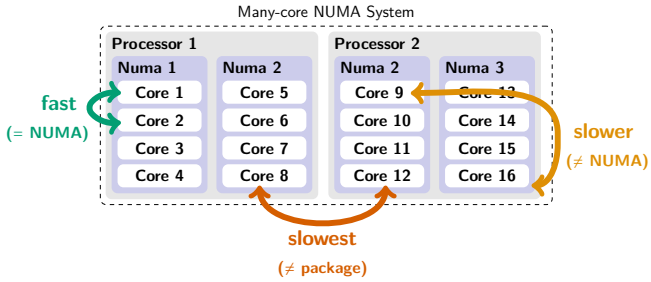
Modern lock-design challenges



Why would locks be too complex to model check?

Modern lock-design challenges

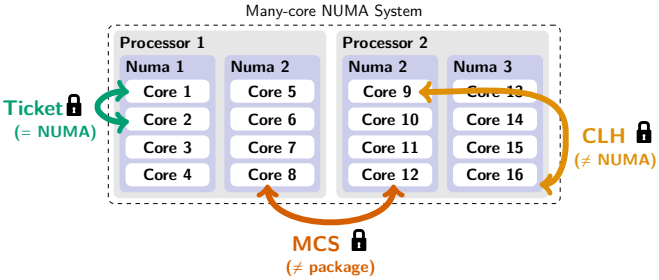
- **Deep NUMA hierarchies**
  eg, packages, NUMA nodes, L3 cache partitions



Why would locks be too complex to model check?

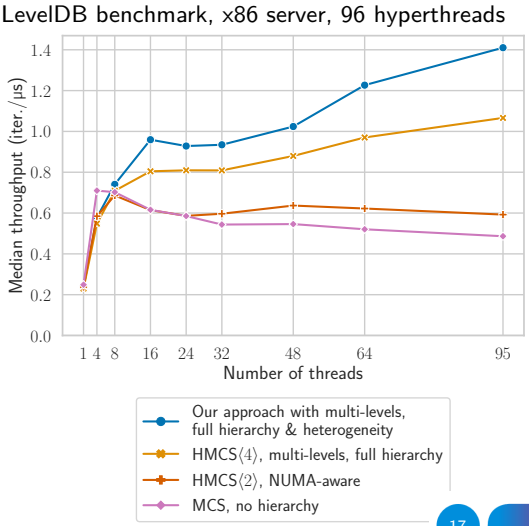
Modern lock-design challenges

- **Deep NUMA hierarchies**
  eg, packages, NUMA nodes, L3 cache partitions
- **Lock heterogenity**
  Different locks perform better in different contexts, eg, cores with shared cache or not



Why would locks be too complex to model check?

What are the potential improvements?

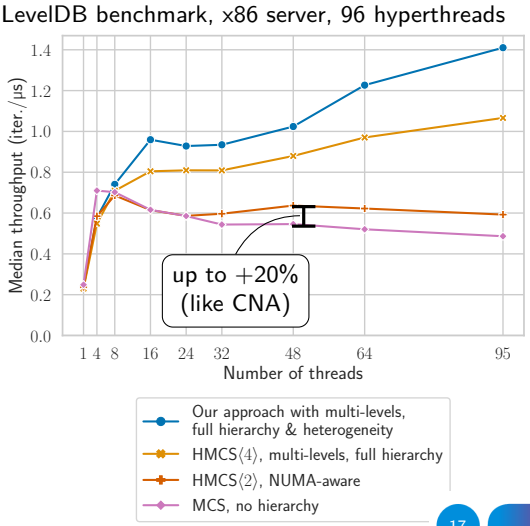


Diogo Behrens — diogo.behrens@huawei.com

Why would locks be too complex to model check?

What are the potential improvements?

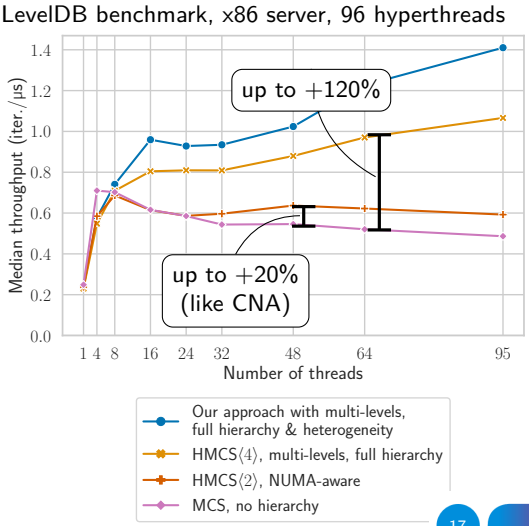
- 2-level is good



Why would locks be too complex to model check?

What are the potential improvements?

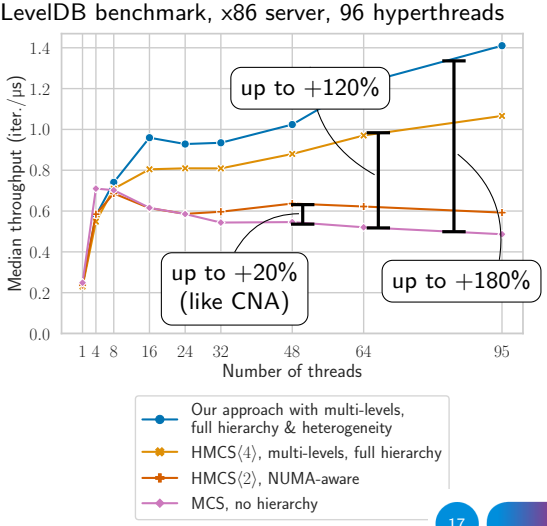
- 2-level is good
- Multi-level is great



Why would locks be too complex to model check?

What are the potential improvements?

- 2-level is good
- Multi-level is great
- Multi-level+heterogeneity is awesome!



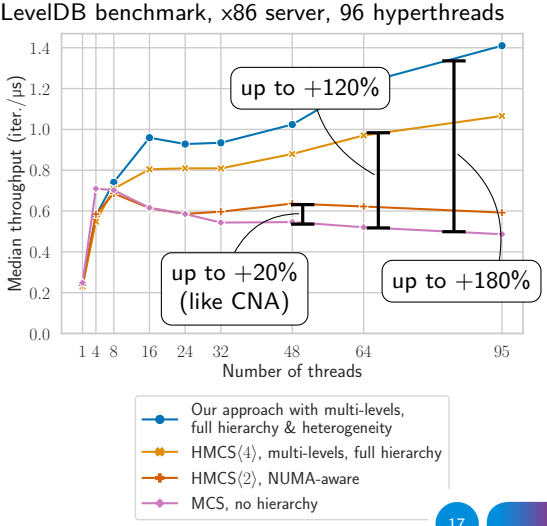
Why would locks be too complex to model check?

What are the potential improvements?

- 2-level is good
- Multi-level is great
- Multi-level+heterogeneity is awesome!

But model checkers can't handle that

- Example multi-level HMCS:



Why would locks be too complex to model check?

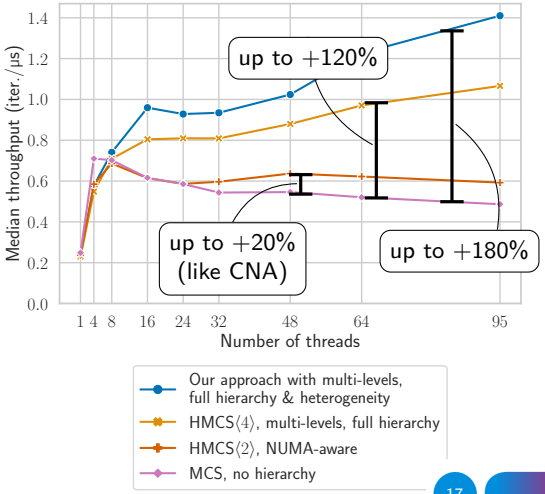
What are the potential improvements?

- 2-level is good
- Multi-level is great
- Multi-level+heterogeneity is awesome!

But model checkers can't handle that

- Example multi-level HMCS:
  - 2 levels: 2s
  - 3 levels: 10s
  - 4 levels: timeout after 24h

LevelDB benchmark, x86 server, 96 hyperthreads



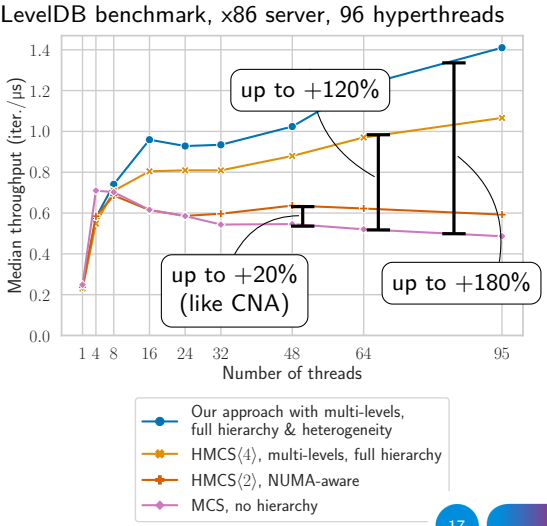
Why would locks be too complex to model check?

What are the potential improvements?

- 2-level is good
- Multi-level is great
- Multi-level+heterogeneity is awesome!

But model checkers can't handle that

- Example multi-level HMCS:
  - 2 levels: 2s
  - 3 levels: 10s
  - 4 levels: timeout after 24h
- **Enter CLoF**!



CLoF: Compositional Lock Framework

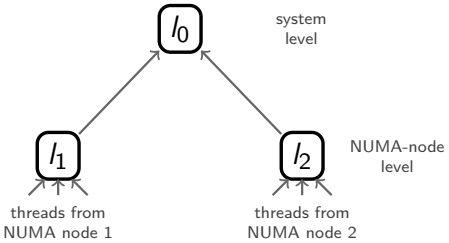
Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

Two NUMA-node example

CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

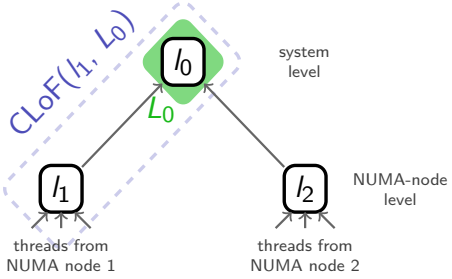
Two NUMA-node example



CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

Two NUMA-node example



Simplified algorithm

```
CLoF(l, L)::acquire =
    acquire l;
    if (¬already has L)
        acquire L;
```

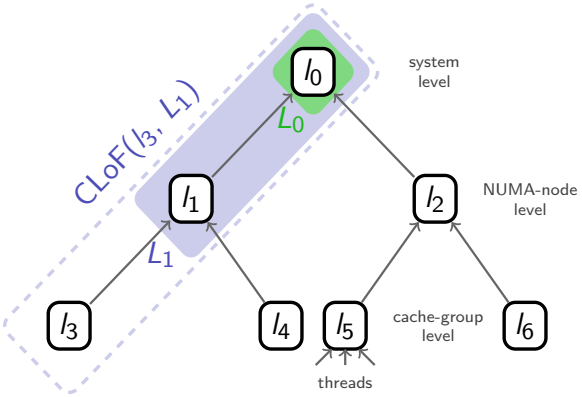
```
CLoF(l, L)::release =
    if (others won't starve)
        release l;
    else
        release L;
        release l;
```


Diogo Behrens — diogo.behrens@huawei.com

CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

Two NUMA-node example



Simplified algorithm

```
CLoF(l, L)::acquire =
    acquire l;
    if (¬already has L)
        acquire L;
```

```
CLoF(l, L)::release =
    if (others won't starve)
        release l;
    else
        release L;
        release l;
```

CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

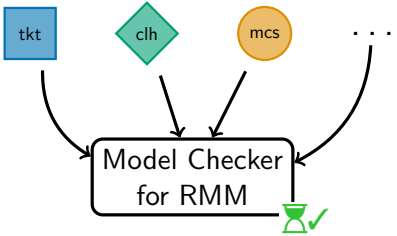
Base Step

Induction Step

CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

Base Step



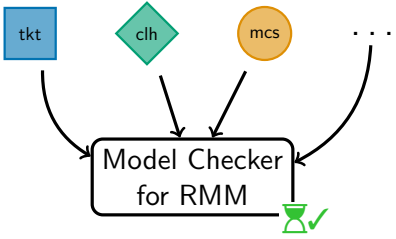
As in VSync paper

Induction Step

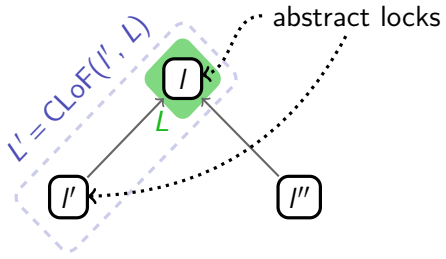
CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

Base Step



Induction Step

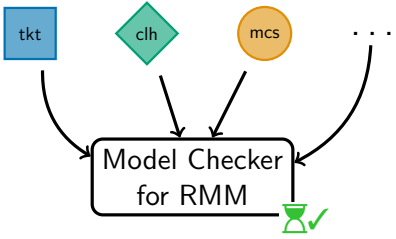


Diogo Behrens — diogo.behrens@huawei.com

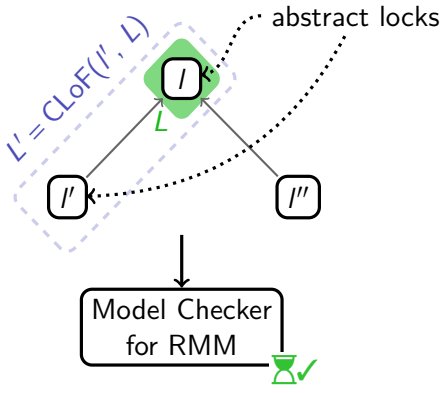
CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

Base Step

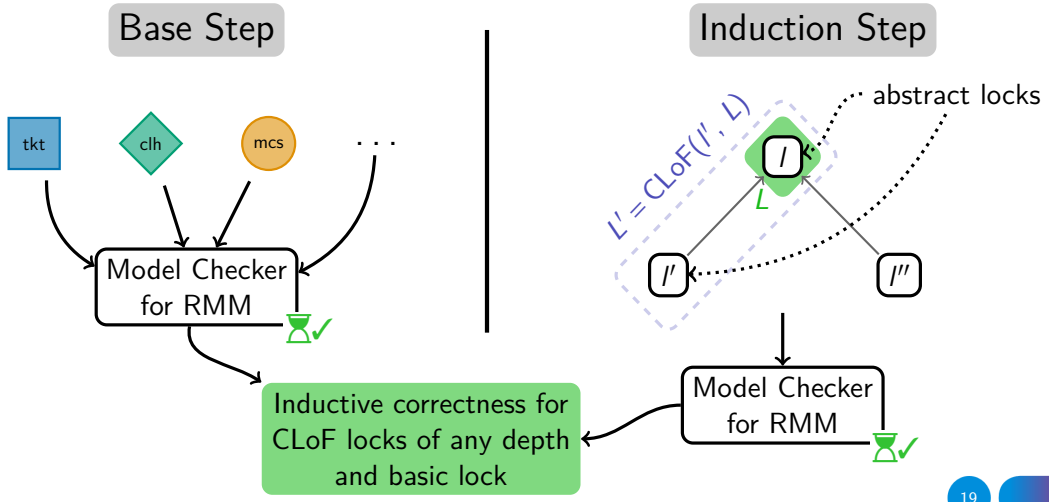


Induction Step



CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557



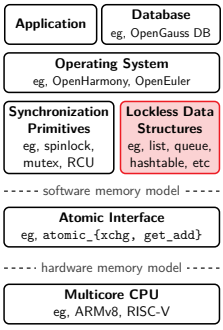
CLoF Learnings

- **Concurrency must be smart and hardware tailored**
  Otherwise we miss big opportunities
- **Modularity is essential for model checking**
  Ideally by design, not in hinsight
- **Support proof sketches are faster, scale, and often OK**
  Goodbye fully-automated verification ☹

Diogo Behrens — diogo.behrens@huawei.com

Agenda

- Modern Hardware and Concurrent Programs
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  - VSync: dealing with Relaxed Memory Models (RMMs)
  - CNA on RMM: verifying next-gen Linux spinlock
  - CLoF: dealing with NUMA hierarchies and heterogeneity
  - BBQ: building highly-efficient ringbuffers
- Wrap up and Outlook



What about concurrent data structures?

What about concurrent data structures?

Ringbuffers are pervasive

What about concurrent data structures?

Ringbuffers are pervasive

What about concurrent data structures?

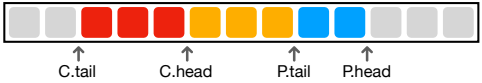
Ringbuffers are pervasive

What about concurrent data structures?

Ringbuffers are pervasive

BBQ: Block-based Bounded Queue

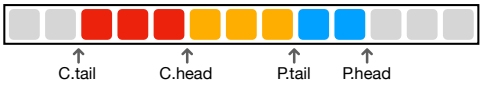
- **Conventional designs:**
  - most **favor simplicity**
  - performance ∼ interference enq/deq



BBQ: Block-based Bounded Queue

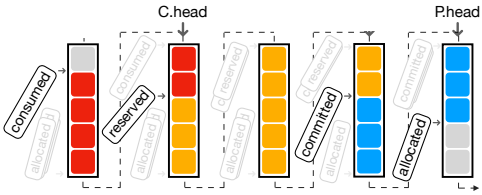
In 6 days @ USENIX ATC'22 — by Wang, Behrens, Fu, Oberhauser, Oberhauser, Lei, Chen, Härtig, Chen

- **Conventional designs:**
  - most **favor simplicity**
  - performance ~ interference enq/deq



- **Block Approach:**
  - Split ringbuffer and metadata in blocks
  - Drastically reduce enq/deq interference
  - 1.5x to 50x higher throughput
  - **Verified with VSync**

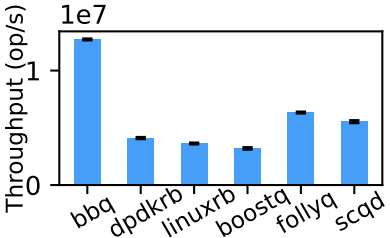
BBQ:

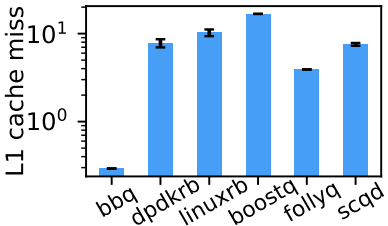


Diogo Behrens — diogo.behrens@huawei.com

BBQ: Performance Experiments

SPSC Microbenchmarks





2x throughput of Meta's FollyQ

See paper for DPDK, Disruptor, io_uring, multiple modes, etc.

BBQ Learnings

The cost of increased performance

DPDK-like algorithm

```
1  enqueue(data){
2  again:
3    ph = LOAD(P.head);
4    pn = ph + 1;
5    if (pn > LOAD(C.tail) + SZ)
6      return FULL;
7    if (!CAS(P.head, ph, pn))
8      goto again;
9    entry[pn % SZ] = data;
10   while(LOAD(P.tail) != ph);
11   STORE(P.tail, pn);
12   return OK;
13 }
```

```
14 dequeue(){
15 again:
16   ch = LOAD(C.head);
17   cn = ch + 1;
18   if (cn > LOAD(P.tail))
19     return EMPTY;
20   if (!CAS(C.head, ch, cn))
21     goto again;
22   data = entry[cn % SZ];
23   while(LOAD(C.tail) != ch);
24   STORE(C.tail, cn);
25   return data;
26 }
```

≈ 10 atomics

Diogo Behrens — diogo.behrens@huawei.com

BBQ Learnings

The cost of increased performance

DPDK-like algorithm

```
enqueue(data){
again:
  ph = LOAD(P.head);
  pn = ph + 1;
  if (pn > LOAD(C.tail) + SZ)
    return FULL;
  if (!CAS(P.head, ph, pn))
    goto again;
entry[pn % SZ] = data;
while(LOAD(P.tail) != ph);
STORE(P.tail, pn);
return OK;
}
```

```
dequeue(){
again:
  ch = LOAD(C.head);
  cn = ch + 1;
  if (cn > LOAD(P.tail))
    return EMPTY;
  if (!CAS(C.head, ch, cn))
    goto again;
data = entry[cn % SZ];
while(LOAD(C.tail) != ch);
STORE(C.tail, cn);
return data;
}
```

≈ 10 atomics

increased complexity

Part of BBQ

```
<Head, Block> BBQ<T>::get_phead_and_block(){
  ph = LOAD(phead);
  return {ph, blocks[ph.idx]};
}
state BBQ<T>::allocate_entry(Block blk){
  if (LOAD(blk.allocated).off >= BLOCK_SIZE)
    return BLOCK_DONE;
  old = FAA(blk.allocated, 1).off;
  if (old >= BLOCK_SIZE)
    return BLOCK_DONE;
  return ALLOCATED{EntryDesc{.block=blk, .offset=old}};
}
void BBQ<T>::commit_entry(EntryDesc e, T data){
  e.block.entries[e.offset] = data;
  ADD(e.block.committed, 1);
}
state BBQ<T>::advance_phead(Head ph) {
  nblk = blocks[(ph.idx + 1) % BLOCK_NUM];
  cons = LOAD(nblk.consumed);
  if (cons.vsn < ph.vsn ||
      (cons.vsn == ph.vsn && cons.off != BLOCK_SIZE)) {
    reserved = LOAD(nblk.reserved);
    if (reserved.off == cons.off) return NO_ENTRY;
    else return NOT_AVAILABLE;
  }
  cmtd = LOAD(nblk.committed);
  if (cmtd.vsn == ph.vsn && cmtd.off != BLOCK_SIZE)
    return NOT_AVAILABLE;
  MAX(nblk.committed, Cursor{.vsn=ph.vsn + 1});
  MAX(nblk.allocated, Cursor{.vsn=ph.vsn + 1});
  MAX(phead, ph + 1);
  return SUCCESS;
}
class BBQ<T> {
  shared<Head> phead, chead;
  Block<T>[] blocks;
}
class Block<T> {
  shared<Cursor> allocated, committed;
  shared<Cursor> reserved, consumed;
  T[] entries;
}
class EntryDesc {
  Block block; Offset offset; Version version; }
```

```
<Head, Block> BBQ<T>::get_chead_and_block(){
  ch = LOAD(chead);
  return {ch, blocks[ch.idx]};
}
state BBQ<T>::reserve_entry(Block blk){
again:
  reserved = LOAD(blk.reserved);
  if (reserved.off < BLOCK_SIZE) {
    committed = LOAD(blk.committed);
    if (reserved.off == committed.off)
      return NO_ENTRY;
    if (committed.off != BLOCK_SIZE) {
      allocated = LOAD(blk.allocated);
      if (allocated.off != committed.off)
        return NOT_AVAILABLE;
    }
    if (MAX(blk.reserved, reserved + 1) == reserved)
      return RESERVED{EntryDesc{.block=blk,
        .offset=reserved.off, .version=reserved.vsn}};
    else goto again;
  }
  return BLOCK_DONE{reserved.vsn};
}
BBQ<T>::consume_entry(EntryDesc e){
  data = e.block.entries[e.offset];
  ADD(e.block.consumed, 1);
  allocated = LOAD(e.block.allocated);
  if (allocated.vsn != e.version) return NULL;
  return data;
}
bool BBQ<T>::advance_chead(Head ch, Version vsn){
  nblk = blocks[(ch.idx + 1) % BLOCK_NUM];
  committed = LOAD(nblk.committed);
  if (committed.vsn != ch.vsn + 1)
    return false;
  MAX(nblk.consumed, Cursor{.vsn=ch.vsn + 1});
  MAX(nblk.reserved, Cursor{.vsn=ch.vsn + 1});
  if (committed.vsn < vsn + (ch.idx == 0))
    return false;
  MAX(nblk.reserved, Cursor{.vsn=committed.vsn});
  MAX(chead, ch + 1);
  return true;
}
```

retry-new mode | drop-old mode

≈ 20 atomics

Diogo Behrens — diogo.behrens@huawei.com

BBQ Learnings

How to ensure correctness?

- **Long stress testing**
  by engineers
- **Identification of corner cases**
  by experts and engineers
- **Model checking of corner cases**
  by engineers with expert support

- **Only a few corner cases necessary**
  queue full/empty, FIFO, wrap-around
- **3 bugs found model checking them**
  Not found while stress testing
- **Reproducible on real hardware**
  Test cases were built in retrospect

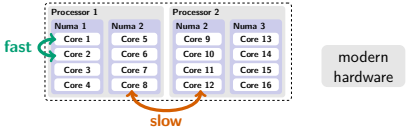
Agenda

- Modern Hardware and Concurrent Programs
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  - VSync: dealing with Relaxed Memory Models (RMMs)
  - CNA on RMM: verifying next-gen Linux spinlock
  - CLoF: dealing with NUMA hierarchies and heterogeneity
  - BBQ: building highly-efficient ringbuffers
- Wrap up and Outlook

Enabling Performance with Practical Verification

Modern hardware features

- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies



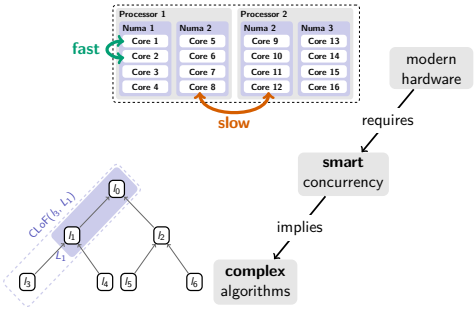
Enabling Performance with Practical Verification

Modern hardware features

- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies

Consequences to concurrency

- Must be **smarter** to boost performance



Diogo Behrens — diogo.behrens@huawei.com

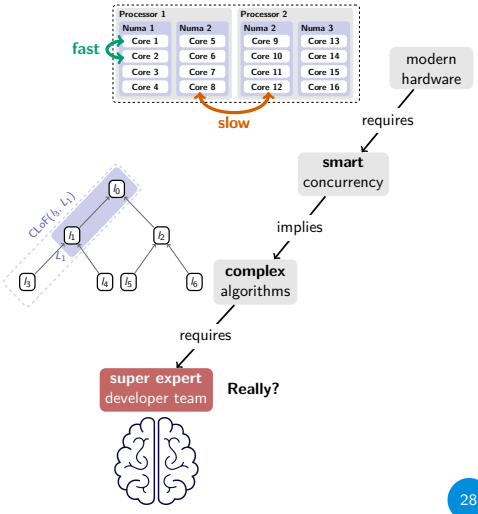
Enabling Performance with Practical Verification

Modern hardware features

- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies

Consequences to concurrency

- Must be **smarter** to boost performance
- But **complexity gets out of control!**



Enabling Performance with Practical Verification

Modern hardware features

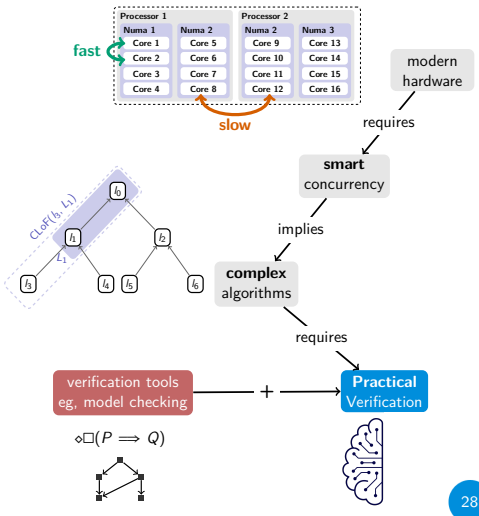
- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies

Consequences to concurrency

- Must be **smarter** to boost performance
- But **complexity gets out of control!**

Practical verification

- **Formal verification tools**
  ↗ confidence on code correctness



Enabling Performance with Practical Verification

Modern hardware features

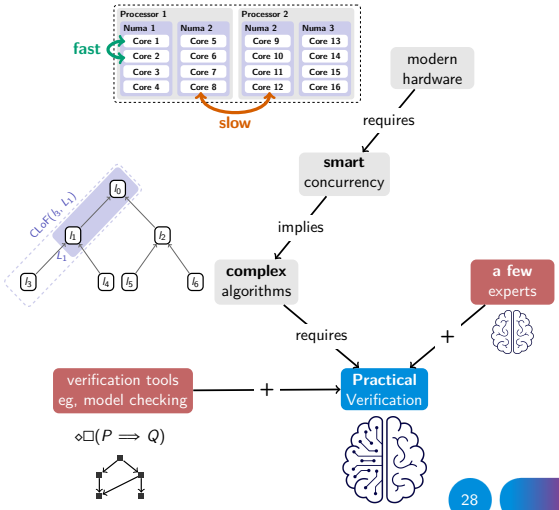
- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies

Consequences to concurrency

- Must be **smarter** to boost performance
- But **complexity gets out of control!**

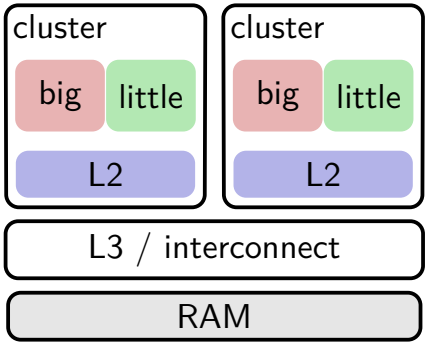
Practical verification

- **Formal verification tools**
  ↗ confidence on code correctness
- **A few experts**
  ↗ scalability and coverage of tools



Diogo Behrens — diogo.behrens@huawei.com

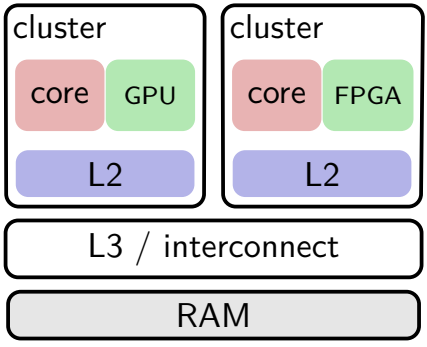
Outlook: A challenging future!



Deeper and more complex hierarchies

- Heterogeneous processing power
  eg, Arm big.LITTLE, Intel Alder Lake

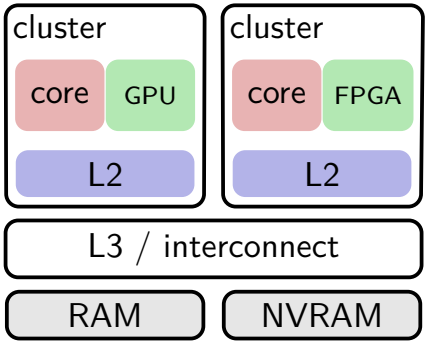
Outlook: A challenging future!



Deeper and more complex hierarchies

- Heterogeneous processing power
  eg, Arm big.LITTLE, Intel Alder Lake
- Accelerators on shared memory
  eg, GPUs, NPUs, FPGAs

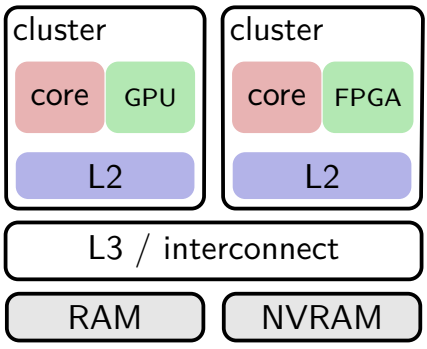
Outlook: A challenging future!



Deeper and more complex hierarchies

- Heterogeneous processing power
  eg, Arm big.LITTLE, Intel Alder Lake
- Accelerators on shared memory
  eg, GPUs, NPUs, FPGAs
- Non-volatile memories

Outlook: A challenging future!



Deeper and more complex hierarchies

- Heterogeneous processing power
  eg, Arm big.LITTLE, Intel Alder Lake
- Accelerators on shared memory
  eg, GPUs, NPUs, FPGAs
- Non-volatile memories

How to consider everythig together?

- Practical verification FTW!
- Great potential of HW-SW collaboration!

THANK YOU

非常感谢你