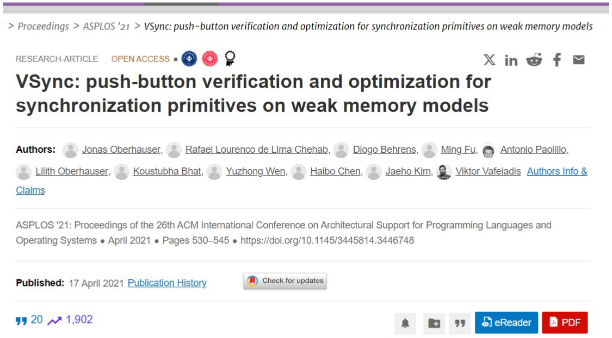
# VSync: Enabling Performance on Modern Hardware with Practical Verification

Diogo Behrens

## Some years ago, in this very same conference. . . finally open sourced with support of OpenHarmony!



> Proceedings > ASPLOS '21 > VSync: push-button verification and optimization for synchronization primitives on weak memory models

RESEARCH-ARTICLE OPEN ACCESS •

**VSync: push-button verification and optimization for synchronization primitives on weak memory models**

**Authors:** Jonas Oberhauser, Rafael Lourenco de Lima Chehab, Diogo Behrens, Ming Fu, Antonio Paolillo, Lilith Oberhauser, Koustubha Bhat, Yuzhong Wen, Haibo Chen, Jaeho Kim, Viktor Vafeiadis Authors Info & Claims

ASPLOS '21: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems • April 2021 • Pages 530–545 • https://doi.org/10.1145/3445814.3446748

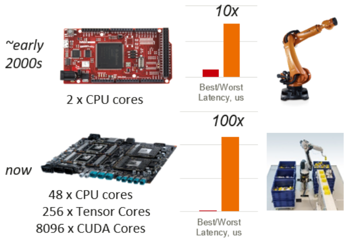
**Published:** 17 April 2021 Publication History

## As the hardware evolves, so does the concurrency control

### Challenges of modern hardware

- Many-cores everywhere

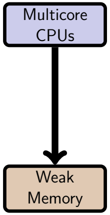
Multicore CPUs

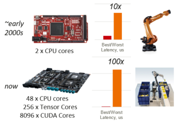


# As the hardware evolves, so does the concurrency control

## Challenges of modern hardware

- Many-cores everywhere
- Weak Memory Models, eg, RISC-V, ARMv8



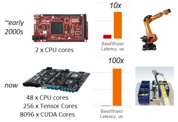


# As the hardware evolves, so does the concurrency control

## Challenges of modern hardware

- Many-cores everywhere
- Weak Memory Models, eg, RISC-V, ARMv8

$$\begin{tikzcd}\boxed{\text{Multicore CPUs}} \arrow[d] \\ \boxed{\text{Weak Memory}}\end{tikzcd}$$



Init

```
msg = 0; ready = false;
```

Thread 1 (WMM reorder)

```
msg = 42;
ready = true;
```

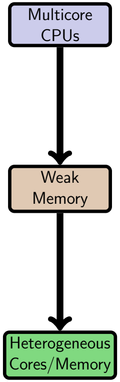
Thread 2

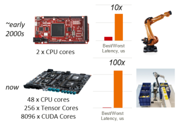
```
while(!ready) {}
assert(msg == 42); ✗
```

# As the hardware evolves, so does the concurrency control

## Challenges of modern hardware

- Many-cores everywhere
- Weak Memory Models, eg, RISC-V, ARMv8





Init

```
msg = 0; ready = false;
```

WMM reorder

Thread 1

```
msg = 42;
ready = true;
```

Thread 2

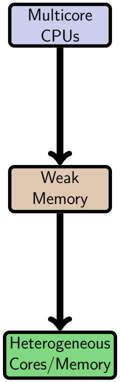
```
while(!ready) {}
assert(msg == 42);
```

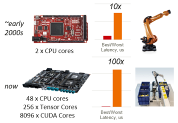
✗

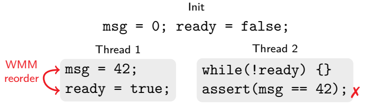
# As the hardware evolves, so does the concurrency control

## Challenges of modern hardware

- Many-cores everywhere
- Weak Memory Models, eg, RISC-V, ARMv8
- Deep NUMA hierarchies







Init

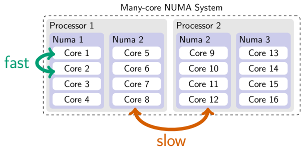
```
msg = 0; ready = false;
```

Thread 1

```
msg = 42;
ready = true;
```

Thread 2

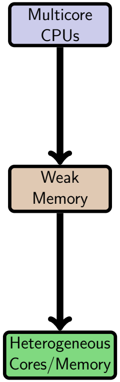
```
while(!ready) {}
assert(msg == 42); ✗
```

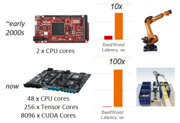


# As the hardware evolves, so does the concurrency control

## Challenges of modern hardware

- Many-cores everywhere
- Weak Memory Models, eg, RISC-V, ARMv8
- Deep NUMA hierarchies
- Heterogeneous cores, eg, big.LITTLE





Init

```
msg = 0; ready = false;
```

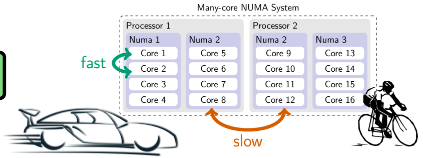
Thread 1

WMM reorder

```
msg = 42;
ready = true;
```

Thread 2

```
while(!ready) {}
assert(msg == 42); ✗
```



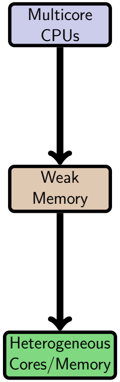
# As the hardware evolves, so does the concurrency control

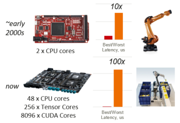
## Challenges of modern hardware

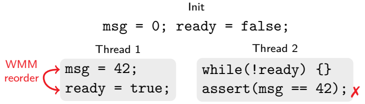
- Many-cores everywhere
- Weak Memory Models, eg, RISC-V, ARMv8
- Deep NUMA hierarchies
- Heterogeneous cores, eg, big.LITTLE

## Consequences to concurrent software

- Smarter concurrency is more complex







Init

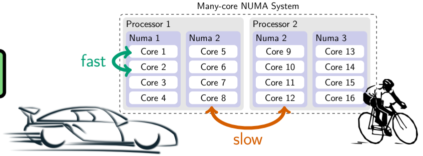
```
msg = 0; ready = false;
```

Thread 1

```
msg = 42;
ready = true;
```

Thread 2

```
while(!ready) {}
assert(msg == 42);
```



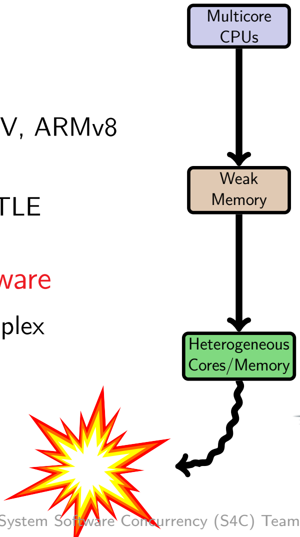
# As the hardware evolves, so does the concurrency control

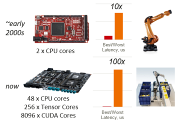
## Challenges of modern hardware

- Many-cores everywhere
- Weak Memory Models, eg, RISC-V, ARMv8
- Deep NUMA hierarchies
- Heterogeneous cores, eg, big.LITTLE

## Consequences to concurrent software

- Smarter concurrency is more complex
- Complexity gets out of control!
- Safety compromised:
  crashes, data corruption, . . .





Init

```
msg = 0; ready = false;
```

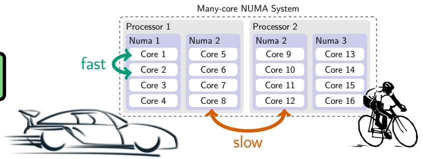
Thread 1

WMM reorder

```
msg = 42;
ready = true;
```

Thread 2

```
while(!ready) {}
assert(msg == 42); ✗
```



## What can the industry do?

### Keep-it-simple and Overprotect?

- Simplify design as much as possible
- Spray code with memory barriers and locks
- Risk: performance impact

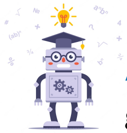
### Rely on Expert Optimizations?

- Exclusively rely on highly-skilled engineers
- Carefully design, implement and optimize
- Risk: error-prone, low maintainability

# Our approach!

Batteries Included

`github.com/open-s4c/libvsync`



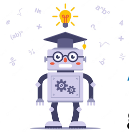
Automated Experts

`github.com/open-s4c/vsyncer`

## Our approach!

Batteries Included

github.com/open-s4c/libvsync



Automated Experts

github.com/open-s4c/vsyncer

Don't rely on a minimal set of overprotected and inefficient concurrent components.

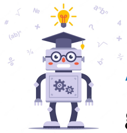
## Our approach!

Batteries Included

github.com/open-s4c/libvsync

Don't rely on a minimal set of overprotected and inefficient concurrent components.

Instead, provide an efficient and verified library of practical components.



Automated Experts

github.com/open-s4c/vsyncer

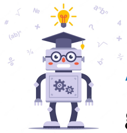
## Our approach!

Batteries Included

github.com/open-s4c/libvsync

Don't rely on a minimal set of overprotected and inefficient concurrent components.

Instead, provide an efficient and verified library of practical components.



Automated Experts

github.com/open-s4c/vsyncer

Avoid always relying on concurrency experts!

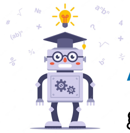
# Our approach!

## Batteries Included

`github.com/open-s4c/libvsync`

Don't rely on a minimal set of overprotected and inefficient concurrent components.

Instead, provide an efficient and verified library of practical components.



## Automated Experts

`github.com/open-s4c/vsyncer`

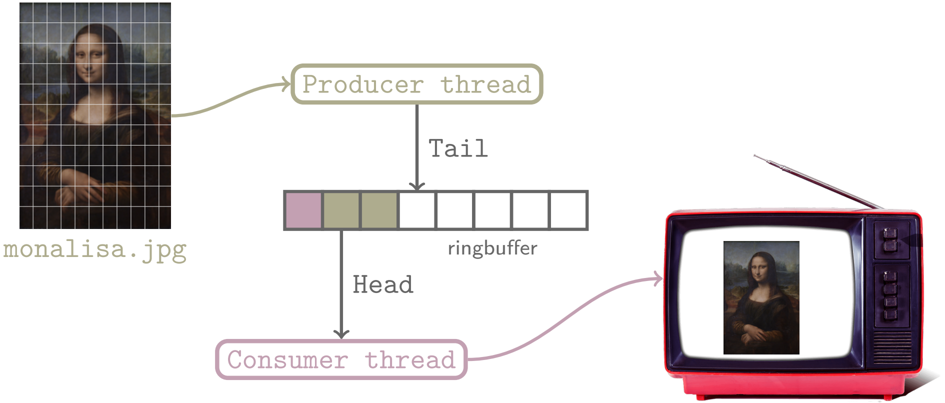
Avoid always relying on concurrency experts!

Instead, enable normal developers to develop concurrent code supporting verification tools.

TODO: Agenda

# Let's implement a `mycat.c`

```
$ mycat monalisa.jpg | viu
```

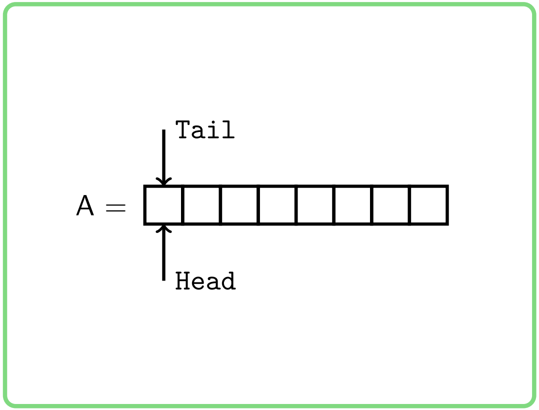


## Coding session #1

Implementing an SPSC ringbuffer. . .

## So, what is the problem, again?

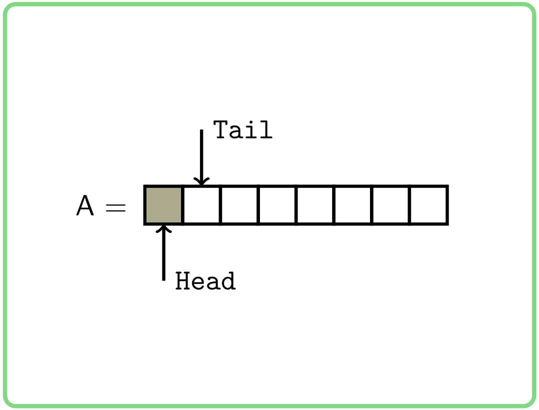
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}
```

## So, what is the problem, again?

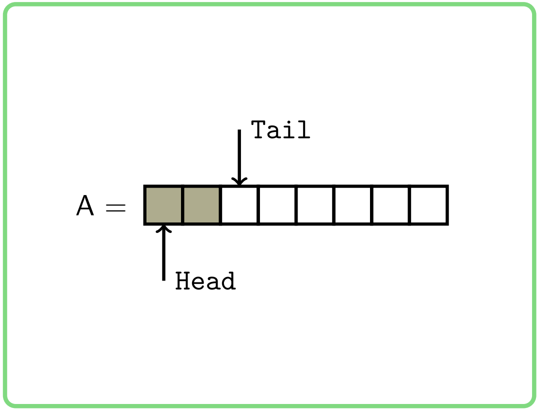
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}
```

## So, what is the problem, again?

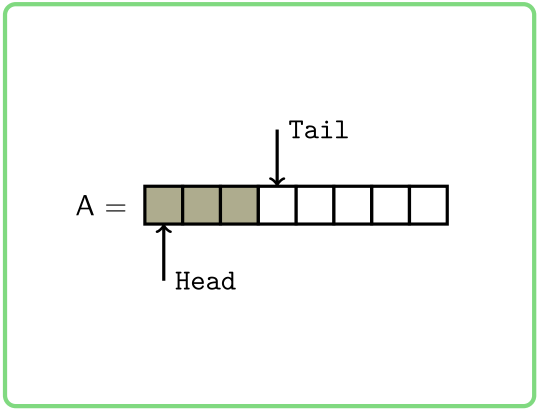
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}
```

## So, what is the problem, again?

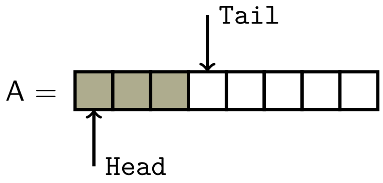
```c
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```c
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}
```

## So, what is the problem, again?

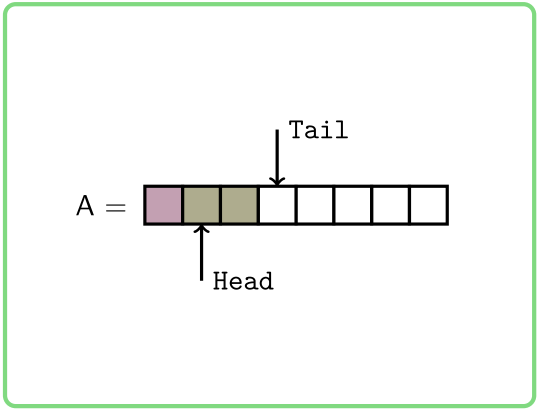
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

## So, what is the problem, again?

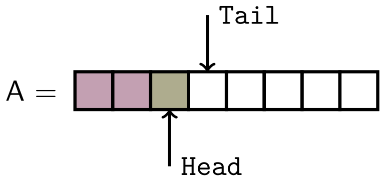
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

# So, what is the problem, again?

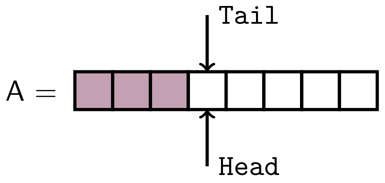
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

## So, what is the problem, again?

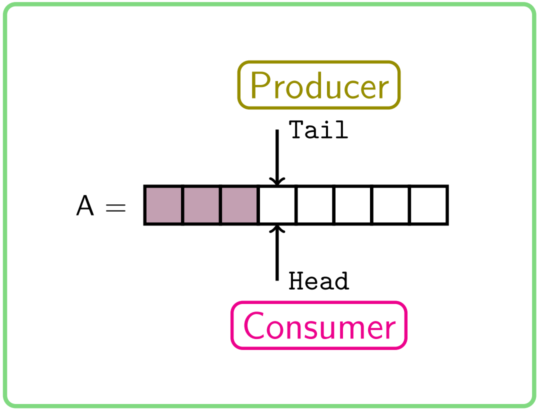
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

## So, what is the problem, again?

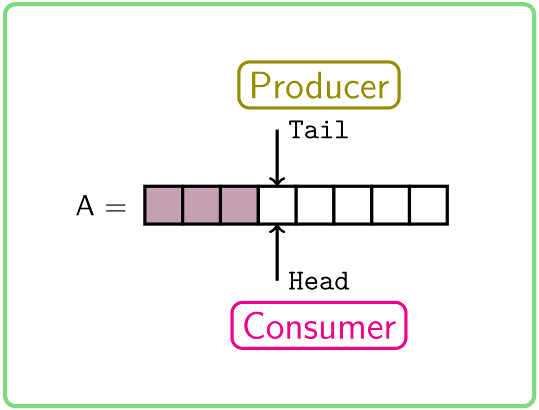
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

## So, what is the problem, again?

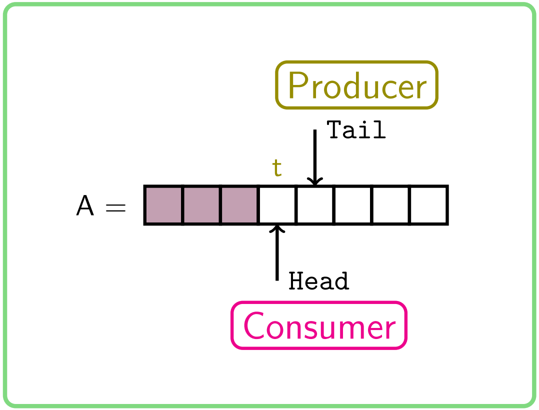
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
→   uint t = Tail++;
    A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

# So, what is the problem, again?

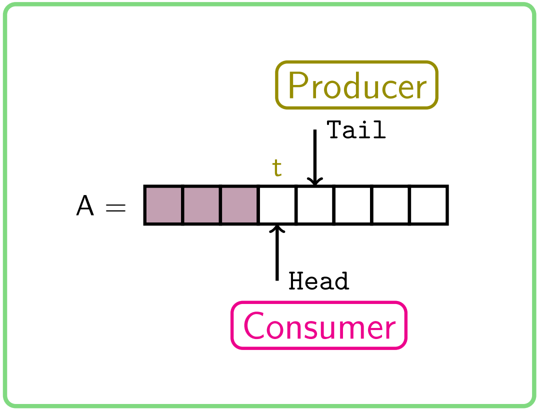
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
→   A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

## So, what is the problem, again?

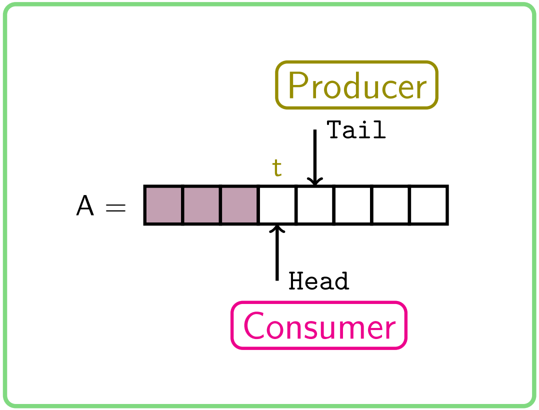
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
--> A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
--> if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

## So, what is the problem, again?

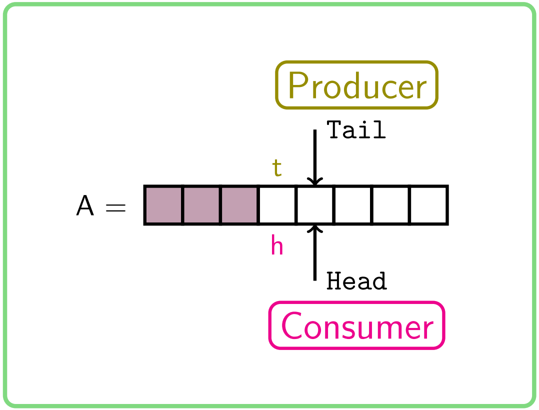
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
--> A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
--> uint h = Head++;
    item_t *i = A[h % N];
    return i;
}
```

# So, what is the problem, again?

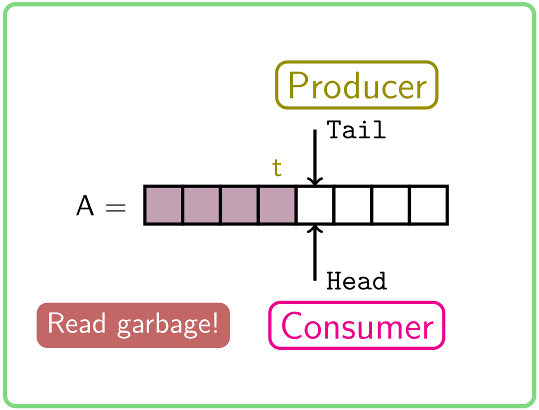
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
--> A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
--> item_t *i = A[h % N];
    return i;
}
```

## So, what is the problem, again?

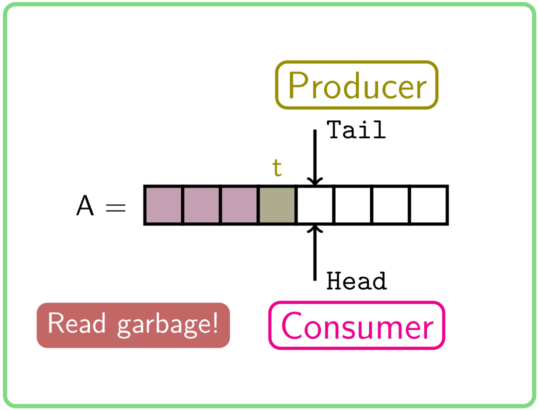
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
--> A[t % N] = item;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
--> return i;
}
```

# So, what is the problem, again?

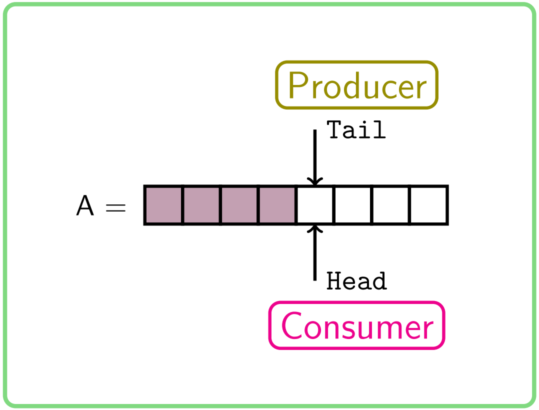
```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail++;
    A[t % N] = item;
→   return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head++;
    item_t *i = A[h % N];
-->  return i;
}
```

## So, what is the problem, again?

```
#define N 8
item_t *A[N];
uint Tail = 0, Head = 0;
```



```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail;
    A[t % N] = item;
    Tail = t + 1;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head;
    item_t *i = A[h % N];
    Head = h + 1;
    return i;
}
```

## Coding session #2

Would this work on Raspberry Pi?

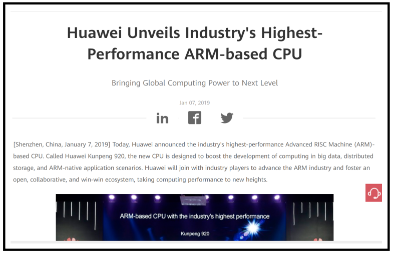
TODO: Agenda

## Weak Memory Consistency Models (WMMs)

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

## Weak Memory Consistency Models (WMMs)

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


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

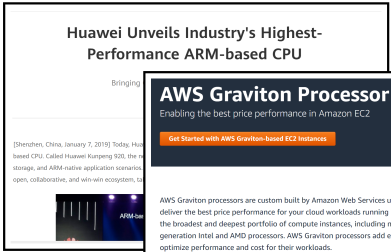
Bringing Global Computing Power to Next Level

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.

# Weak Memory Consistency Models (WMMs)

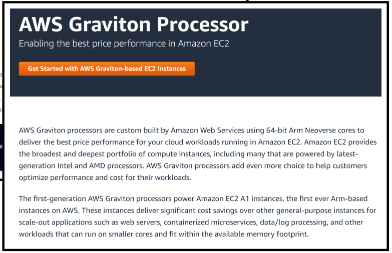
- Modern architectures becoming popular
  eg, Arm, RISC-V



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

Bringing

[Shenzhen, China, January 7, 2019] Today, Hua
based CPU. Called Huawei Kunpeng 920, the n
storage, and ARM-native application scenarios.
open, collaborative, and win-win ecosystem, ta



AWS Graviton Processor

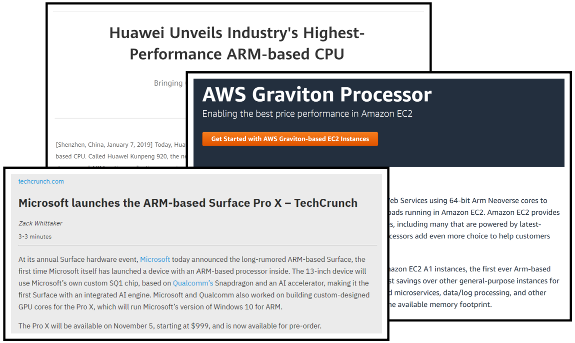
Enabling the best price performance in Amazon EC2

AWS Graviton processors are custom built by Amazon Web Services using 64-bit Arm Neoverse cores to deliver the best price performance for your cloud workloads running in Amazon EC2. Amazon EC2 provides the broadest and deepest portfolio of compute instances, including many that are powered by latest-generation Intel and AMD processors. AWS Graviton processors add even more choice to help customers optimize performance and cost for their workloads.

The first-generation AWS Graviton processors power Amazon EC2 A1 instances, the first ever Arm-based instances on AWS. These instances deliver significant cost savings over other general-purpose instances for scale-out applications such as web servers, containerized microservices, data/log processing, and other workloads that can run on smaller cores and fit within the available memory footprint.

# Weak Memory Consistency Models (WMMs)

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


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

Bringing

[Shenzhen, China, January 7, 2019] Today, Hua
based CPU. Called Huawei Kunpeng 920, the n

AWS Graviton Processor
Enabling the best price performance in Amazon EC2

eb Services using 64-bit Arm Neoverse cores to
oads running in Amazon EC2. Amazon EC2 provides
s, including many that are powered by latest-
cessors add even more choice to help customers

azon EC2 A1 instances, the first ever Arm-based
t savings over other general-purpose instances for
d microservices, data/log processing, and other
e available memory footprint.

techcrunch.com

Microsoft launches the ARM-based Surface Pro X – TechCrunch

*Zack Whittaker*

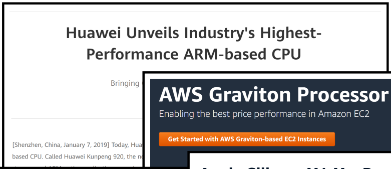
3-3 minutes

At its annual Surface hardware event, Microsoft today announced the long-rumored ARM-based Surface, the first time Microsoft itself has launched a device with an ARM-based processor inside. The 13-inch device will use Microsoft's own custom SQ1 chip, based on Qualcomm's Snapdragon and an AI accelerator, making it the first Surface with an integrated AI engine. Microsoft and Qualcomm also worked on building custom-designed GPU cores for the Pro X, which will run Microsoft's version of Windows 10 for ARM.

The Pro X will be available on November 5, starting at $999, and is now available for pre-order.

# Weak Memory Consistency Models (WMMs)

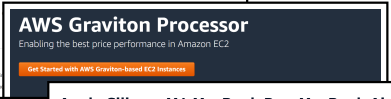
- Modern architectures becoming popular
  eg, Arm, RISC-V



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

Bringing

[Shenzhen, China, January 7, 2019] Today, Hua
based CPU. Called Huawei Kunpeng 920, the n



AWS Graviton Processor

Enabling the best price performance in Amazon EC2

techcrunch.com

Microsoft launches the ARM-based Surf

*Zack Whittaker*

3-3 minutes

At its annual Surface hardware event, Microsoft today announced
first time Microsoft itself has launched a device with an ARM-base
use Microsoft's own custom SQ1 chip, based on Qualcomm's Snap
first Surface with an integrated AI engine. Microsoft and Qualcomm
GPU cores for the Pro X, which will run Microsoft's version of Wind

The Pro X will be available on November 5, starting at $999, and is

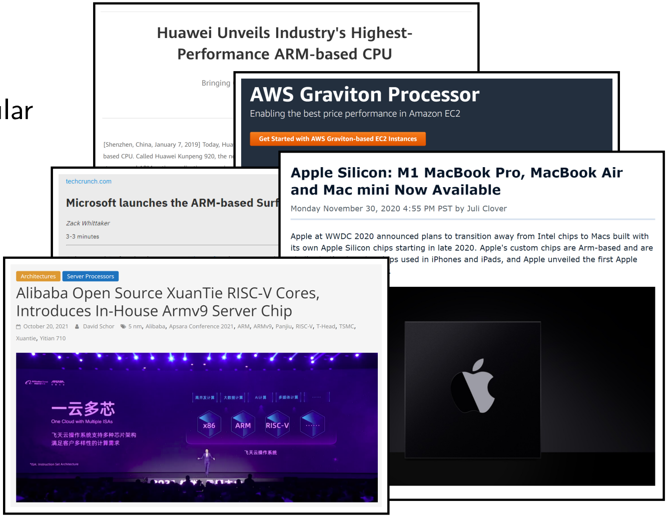
Apple Silicon: M1 MacBook Pro, MacBook Air and Mac mini Now Available

Monday November 30, 2020 4:55 PM PST by Juli Clover

Apple at WWDC 2020 announced plans to transition away from Intel chips to Macs built with its own Apple Silicon chips starting in late 2020. Apple's custom chips are Arm-based and are similar to the A-series chips used in iPhones and iPads, and Apple unveiled the first Apple Silicon Macs in November.

# Weak Memory Consistency Models (WMMs)

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



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

Bringing

[Shenzhen, China, January 7, 2019] Today, Hua
based CPU. Called Huawei Kunpeng 920, the n

AWS Graviton Processor

Enabling the best price performance in Amazon EC2

techcrunch.com

Microsoft launches the ARM-based Surf

*Zack Whittaker*

3-3 minutes

Apple Silicon: M1 MacBook Pro, MacBook Air and Mac mini Now Available

Monday November 30, 2020 4:55 PM PST by Juli Clover

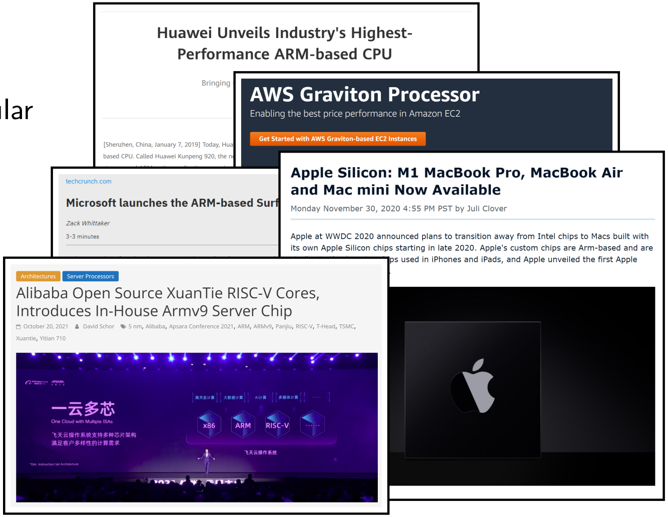
Apple at WWDC 2020 announced plans to transition away from Intel chips to Macs built with its own Apple Silicon chips starting in late 2020. Apple's custom chips are Arm-based and are
ps used in iPhones and iPads, and Apple unveiled the first Apple

Alibaba Open Source XuanTie RISC-V Cores, Introduces In-House Armv9 Server Chip

October 20, 2021 David Schor 5 nm, Alibaba, Apsara Conference 2021, ARM, ARMv9, Panjiu, RISC-V, T-Head, TSMC, Xuantie, Yitian 710

# Weak Memory Consistency Models (WMMs)

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



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

Bringing

[Shenzhen, China, January 7, 2019] Today, Hua
based CPU. Called Huawei Kunpeng 920, the n

AWS Graviton Processor

Enabling the best price performance in Amazon EC2

Get Started with AWS Graviton-based EC2 Instances

techcrunch.com

Microsoft launches the ARM-based Surf

Zack Whittaker

3-3 minutes

Apple Silicon: M1 MacBook Pro, MacBook Air and Mac mini Now Available

Monday November 30, 2020 4:55 PM PST by Juli Clover

Apple at WWDC 2020 announced plans to transition away from Intel chips to Macs built with its own Apple Silicon chips starting in late 2020. Apple's custom chips are Arm-based and are ps used in iPhones and iPads, and Apple unveiled the first Apple

Architectures Server Processors

Alibaba Open Source XuanTie RISC-V Cores, Introduces In-House Armv9 Server Chip

October 20, 2021 David Schor 5 nm, Alibaba, Apsara Conference 2021, ARM, ARMv9, Panjiu, RISC-V, T-Head, TSMC, Xuantie, Yitian 710

## WMM and mycat

TODO: explain how atomics fix the weak memory issues in mycat

- Independent memory accesses can be reordered
- For example, reorders writes to `A[t % N]` and `Tail`
- Ringbuffer is still broken!
- We need to add barriers
- TOO MANY?

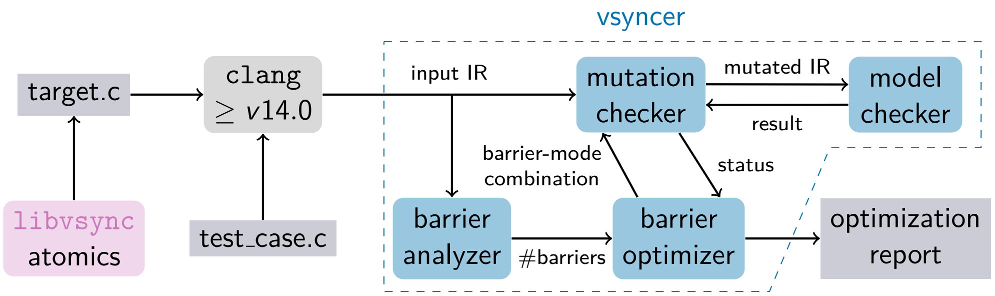
```
bool enqueue(item_t *item) {
    // space to enqueue?
    if (Tail - Head == N)
        return false;
    uint t = Tail;
    A[t % N] = item;
    Tail = t + 1;
    return true;
}

item_t *dequeue() {
    // item to dequeue?
    if (Tail - Head == 0)
        return NULL;
    uint h = Head;
    item_t *i = A[h % N];
    Head = h + 1;
    return i;
}
```

## Coding session #3

How to relax barriers without breaking the code?

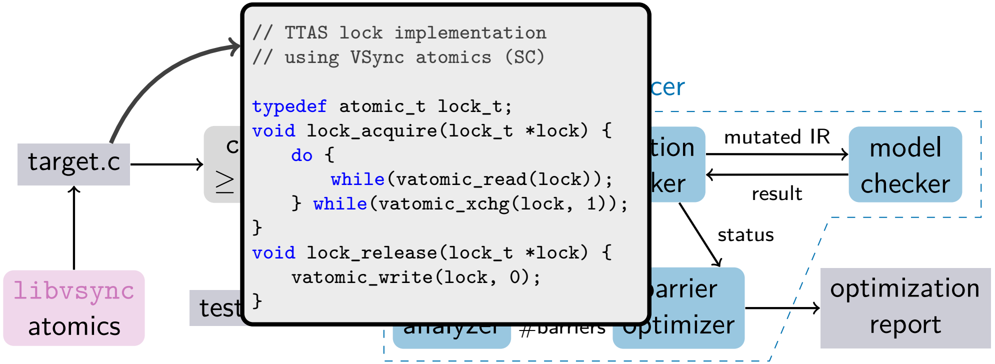
# Safe and Fast Concurrency on Arm processors

VSync: Push-button Verification and Optimization on WMMs — Distinguished paper at ASPLOS'21



## Safe and Fast Concurrency on Arm processors

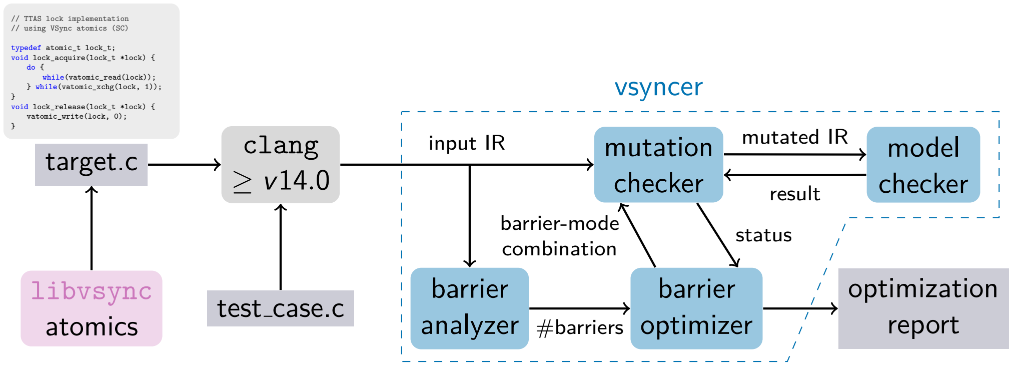
VSync: Push-button Verification and Optimization on WMMs — Distinguished paper at ASPLOS'21



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

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(vatomic_read(lock));
    } while(vatomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    vatomic_write(lock, 0);
}
```

# Safe and Fast Concurrency on Arm processors

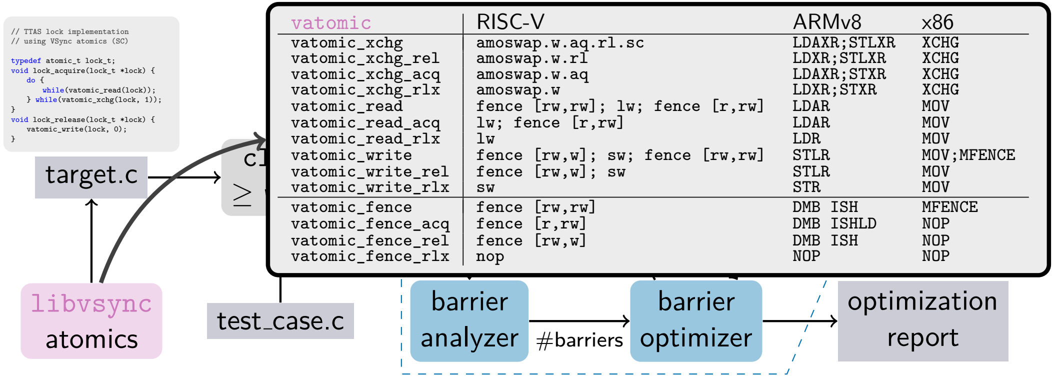
VSync: Push-button Verification and Optimization on WMMs — Distinguished paper at ASPLOS'21



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

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(vatomic_read(lock));
    } while(vatomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    vatomic_write(lock, 0);
}
```

# Safe and Fast Concurrency on Arm processors

## VSync: Push-button Verification and Optimization on WMMs — Distinguished paper at ASPLOS'21

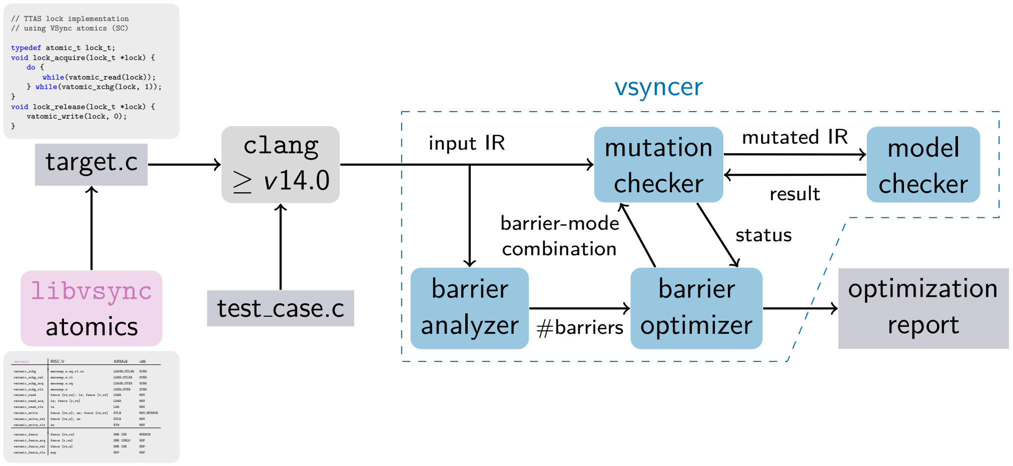


```c
// TTAS lock implementation
// using VSync atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(vatomic_read(lock));
    } while(vatomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    vatomic_write(lock, 0);
}
```

| vatomic | RISC-V | ARMv8 | x86 |
|---|---|---|---|
| vatomic_xchg | amoswap.w.aq.rl.sc | LDAXR;STLXR | XCHG |
| vatomic_xchg_rel | amoswap.w.rl | LDXR;STLXR | XCHG |
| vatomic_xchg_acq | amoswap.w.aq | LDAXR;STXR | XCHG |
| vatomic_xchg_rlx | amoswap.w | LDXR;STXR | XCHG |
| vatomic_read | fence [rw,rw]; lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_acq | lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_rlx | lw | LDR | MOV |
| vatomic_write | fence [rw,w]; sw; fence [rw,rw] | STLR | MOV;MFENCE |
| vatomic_write_rel | fence [rw,w]; sw | STLR | MOV |
| vatomic_write_rlx | sw | STR | MOV |
| vatomic_fence | fence [rw,rw] | DMB ISH | MFENCE |
| vatomic_fence_acq | fence [r,rw] | DMB ISHLD | NOP |
| vatomic_fence_rel | fence [rw,w] | DMB ISH | NOP |
| vatomic_fence_rlx | nop | NOP | NOP |

# Safe and Fast Concurrency on Arm processors

## VSync: Push-button Verification and Optimization on WMMs — 🎖 Distinguished paper at ASPLOS'21

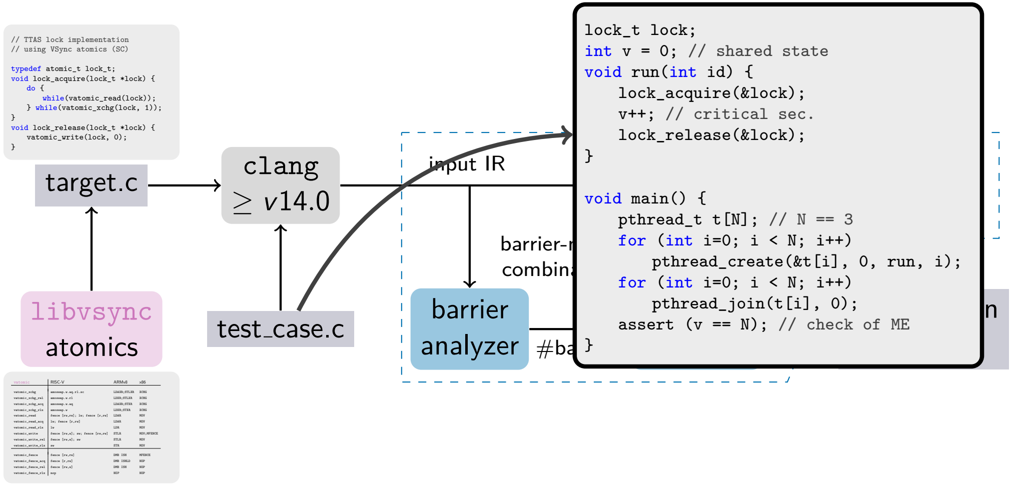


```
// TTAS lock implementation
// using VSync atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(vatomic_read(lock));
    } while(vatomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    vatomic_write(lock, 0);
}
```

| vatomic | RISC-V | ARMv8 | x86 |
|---|---|---|---|
| vatomic_xchg | amoswap.w.aq.rl.sc | LDAXR;STLXR | XCHG |
| vatomic_xchg_rel | amoswap.w.rl | LDXR;STLXR | XCHG |
| vatomic_xchg_acq | amoswap.w.aq | LDAXR;STXR | XCHG |
| vatomic_xchg_rlx | amoswap.w | LDXR;STXR | XCHG |
| vatomic_read | fence [rw,rw]; lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_acq | lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_rlx | lw | LDR | MOV |
| vatomic_write | fence [rw,w]; sw; fence [rw,rw] | STLR | MOV;MFENCE |
| vatomic_write_rel | fence [rw,w]; sw | STLR | MOV |
| vatomic_write_rlx | sw | STR | MOV |
| vatomic_fence | fence [rw,rw] | DMB ISH | MFENCE |
| vatomic_fence_acq | fence [r,rw] | DMB ISHLD | NOP |
| vatomic_fence_rel | fence [rw,w] | DMB ISH | NOP |
| vatomic_fence_rlx | nop | NOP | NOP |

# Safe and Fast Concurrency on Arm processors

## VSync: Push-button Verification and Optimization on WMMs — Distinguished paper at ASPLOS’21



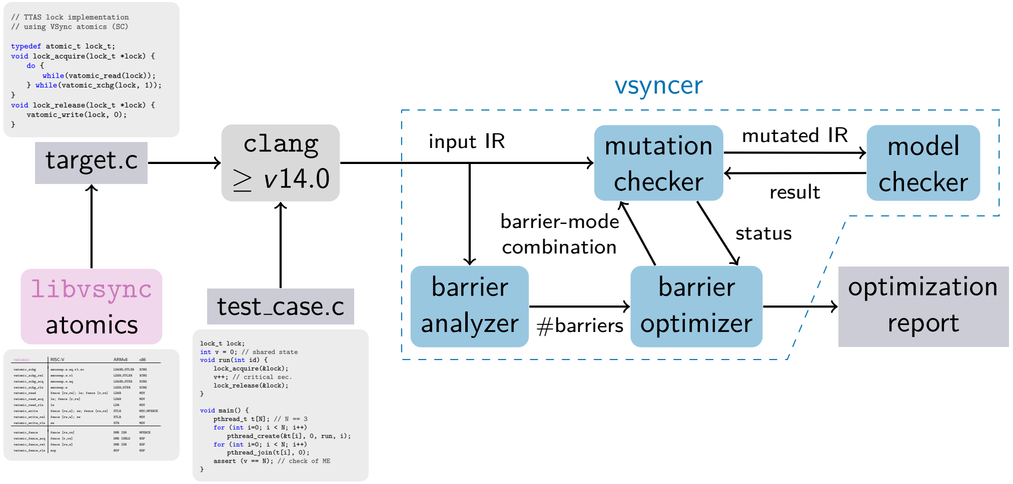
```
// TTAS lock implementation
// using VSync atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(vatomic_read(lock));
    } while(vatomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    vatomic_write(lock, 0);
}
```

| vatomic | RISC-V | ARMv8 | x86 |
|---|---|---|---|
| vatomic_xchg | amoswap.w.aq.rl.sc | LDAXR;STLXR | XCHG |
| vatomic_xchg_rel | amoswap.w.rl | LDXR;STLXR | XCHG |
| vatomic_xchg_acq | amoswap.w.aq | LDAXR;STXR | XCHG |
| vatomic_xchg_rlx | amoswap.w | LDXR;STXR | XCHG |
| vatomic_read | fence [rw,rw]; lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_acq | lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_rlx | lw | LDR | MOV |
| vatomic_write | fence [rw,w]; sw; fence [rw,rw] | STLR | MOV;MFENCE |
| vatomic_write_rel | fence [rw,w]; sw | STLR | MOV |
| vatomic_write_rlx | sw | STR | MOV |
| vatomic_fence | fence [rw,rw] | DMB ISH | MFENCE |
| vatomic_fence_acq | fence [r,rw] | DMB ISHLD | NOP |
| vatomic_fence_rel | fence [rw,w] | DMB ISH | NOP |
| vatomic_fence_rlx | nop | NOP | NOP |

```
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
}
```

# Safe and Fast Concurrency on Arm processors

## VSync: Push-button Verification and Optimization on WMMs — Distinguished paper at ASPLOS'21



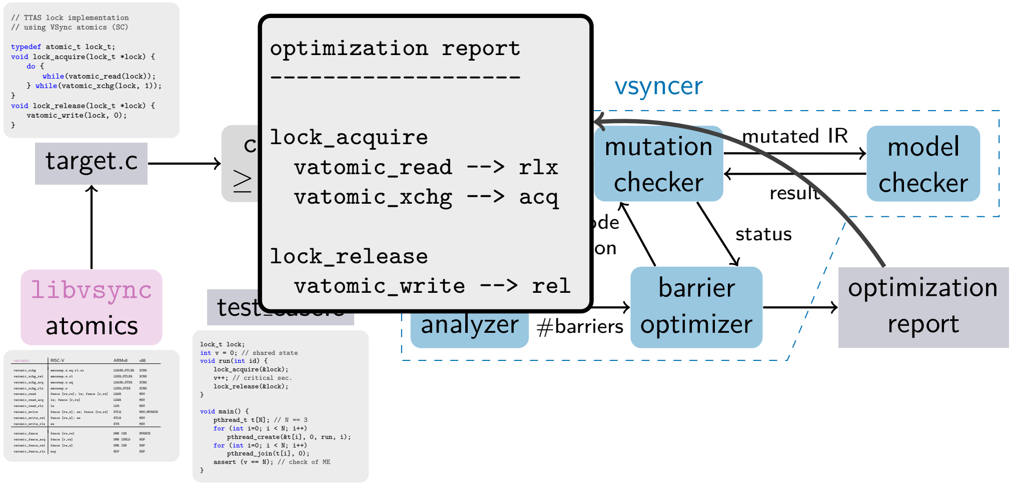
```
// TTAS lock implementation
// using VSync atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(vatomic_read(lock));
    } while(vatomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    vatomic_write(lock, 0);
}
```

| vatomic | RISC-V | ARMv8 | x86 |
|---|---|---|---|
| vatomic_xchg | amoswap.w.aq.rl.sc | LDAXR;STLXR | XCHG |
| vatomic_xchg_rel | amoswap.w.rl | LDXR;STLXR | XCHG |
| vatomic_xchg_acq | amoswap.w.aq | LDAXR;STXR | XCHG |
| vatomic_xchg_rlx | amoswap.w | LDXR;STXR | XCHG |
| vatomic_read | fence [rw,rw]; lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_acq | lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_rlx | lw | LDR | MOV |
| vatomic_write | fence [rw,w]; sw; fence [rw,rw] | STLR | MOV;MFENCE |
| vatomic_write_rel | fence [rw,w]; sw | STLR | MOV |
| vatomic_write_rlx | sw | STR | MOV |
| vatomic_fence | fence [rw,rw] | DMB ISH | MFENCE |
| vatomic_fence_acq | fence [r,rw] | DMB ISHLD | NOP |
| vatomic_fence_rel | fence [rw,w] | DMB ISH | NOP |
| vatomic_fence_rlx | nop | NOP | NOP |

```
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
}
```

# Safe and Fast Concurrency on Arm processors

## VSync: Push-button Verification and Optimization on WMMs — Distinguished paper at ASPLOS'21



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

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(vatomic_read(lock));
    } while(vatomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    vatomic_write(lock, 0);
}
```

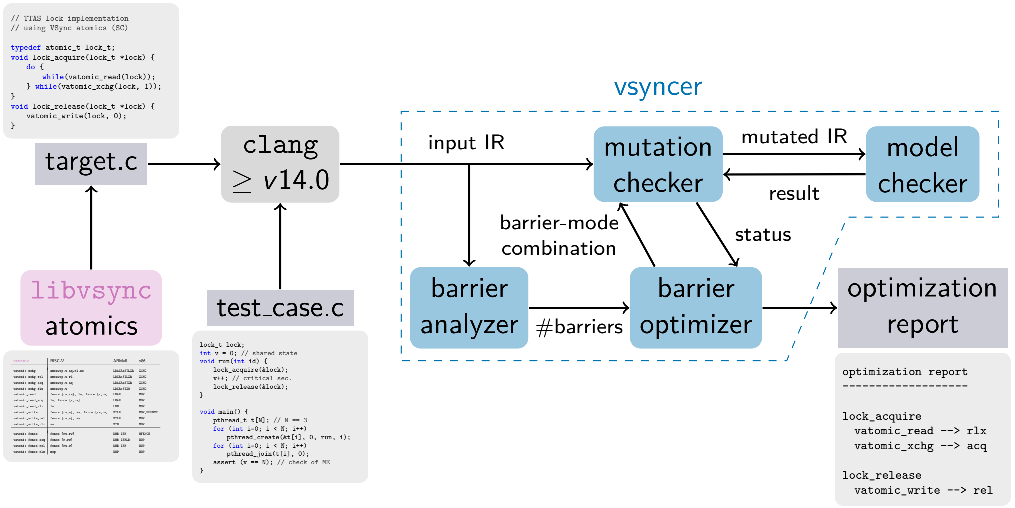
```
optimization report
--------------------

lock_acquire
  vatomic_read --> rlx
  vatomic_xchg --> acq

lock_release
  vatomic_write --> rel
```

| vatomic | RISC-V | ARMv8 | x86 |
|---|---|---|---|
| vatomic_xchg | amoswap.w.aq.rl.sc | LDAXR;STLXR | XCHG |
| vatomic_xchg_rel | amoswap.w.rl | LDXR;STLXR | XCHG |
| vatomic_xchg_acq | amoswap.w.aq | LDAXR;STXR | XCHG |
| vatomic_xchg_rlx | amoswap.w | LDXR;STXR | XCHG |
| vatomic_read | fence [rw,rw]; lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_acq | lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_rlx | lw | LDR | MOV |
| vatomic_write | fence [rw,w]; sw; fence [rw,rw] | STLR | MOV;MFENCE |
| vatomic_write_rel | fence [rw,w]; sw | STLR | MOV |
| vatomic_write_rlx | sw | STR | MOV |
| vatomic_fence | fence [rw,rw] | DMB ISH | MFENCE |
| vatomic_fence_acq | fence [r,rw] | DMB ISHLD | NOP |
| vatomic_fence_rel | fence [rw,w] | DMB ISH | NOP |
| vatomic_fence_rlx | nop | NOP | NOP |

```
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
}
```

# Safe and Fast Concurrency on Arm processors

## VSync: Push-button Verification and Optimization on WMMs — 🎗 Distinguished paper at ASPLOS'21



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

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(vatomic_read(lock));
    } while(vatomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    vatomic_write(lock, 0);
}
```

| vatomic | RISC-V | ARMv8 | x86 |
|---|---|---|---|
| vatomic_xchg | amoswap.w.aq.rl.sc | LDAXR;STLXR | XCHG |
| vatomic_xchg_rel | amoswap.w.rl | LDXR;STLXR | XCHG |
| vatomic_xchg_acq | amoswap.w.aq | LDAXR;STXR | XCHG |
| vatomic_xchg_rlx | amoswap.w | LDXR;STXR | XCHG |
| vatomic_read | fence [rw,rw]; lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_acq | lw; fence [r,rw] | LDAR | MOV |
| vatomic_read_rlx | lw | LDR | MOV |
| vatomic_write | fence [rw,w]; sw; fence [rw,rw] | STLR | MOV;MFENCE |
| vatomic_write_rel | fence [rw,w]; sw | STLR | MOV |
| vatomic_write_rlx | sw | STR | MOV |
| vatomic_fence | fence [rw,rw] | DMB ISH | MFENCE |
| vatomic_fence_acq | fence [r,rw] | DMB ISHLD | NOP |
| vatomic_fence_rel | fence [rw,w] | DMB ISH | NOP |
| vatomic_fence_rlx | nop | NOP | NOP |

```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
}
```

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

lock_acquire
  vatomic_read --> rlx
  vatomic_xchg --> acq

lock_release
  vatomic_write --> rel
```

## Safe and Fast Concurrency on Arm processors

VSync: Push-button Verification and Optimization on WMMs — Distinguished paper at ASPLOS'21

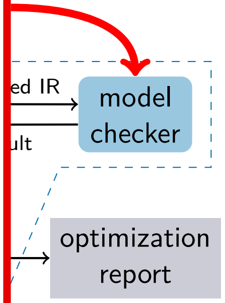
Under the hook the best model checkers for WMMs

GenMC by MPI-SWS

- `https://github.com/MPI-SWS/genmc`
- First to verify liveness of spinloops based on our work.

Dartagnan by TU Braunschweig and Huawei DRC

- `https://github.com/hernanponcedeleon/Dat3M`
- Hernan Ponce de Leon (maintainer) joined our team in 2022
- We are transforming it from an academic into a practical tool
- Gold medal at SV-COMP 2023
- Two gold medals at SV-COMP 2024



## Coding session #4

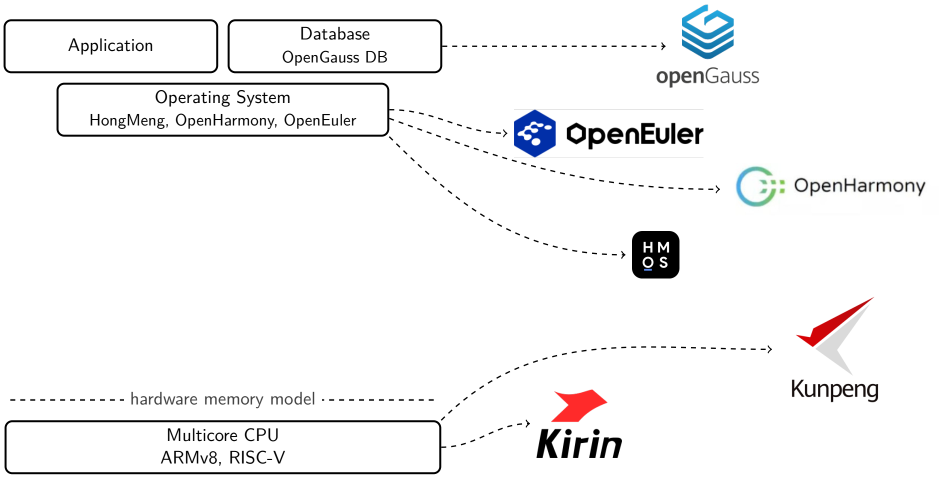
What if MPMC? ARMv8? . . .

Expected questions:

Do I always have to do all this work?

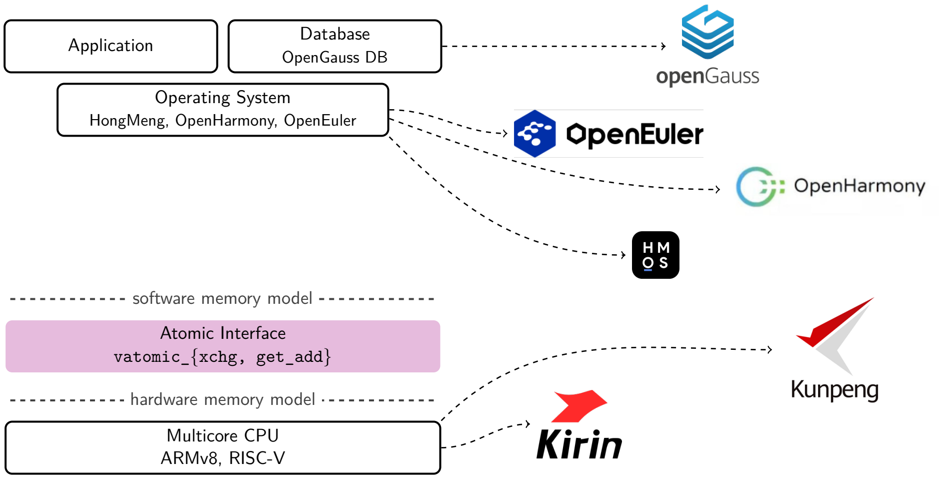
# libvsync: Solid concurrency foundations

github.com/open-s4c/libvsync



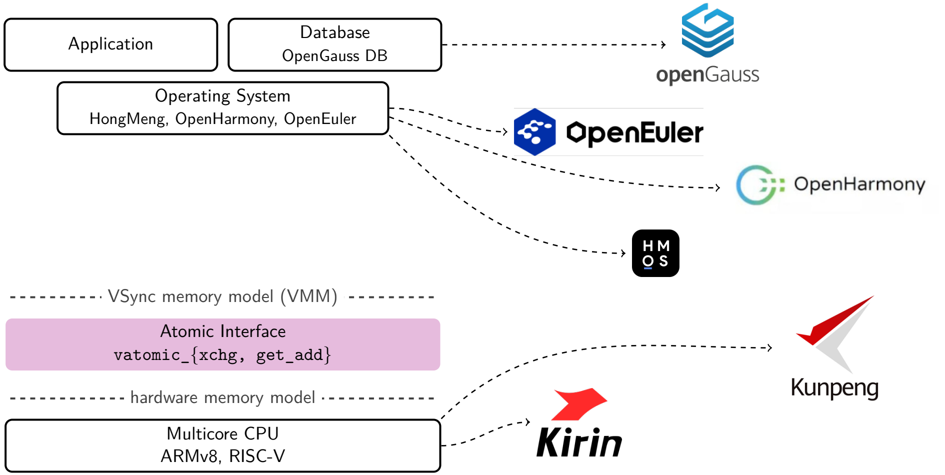
# libvsync: Solid concurrency foundations

github.com/open-s4c/libvsync



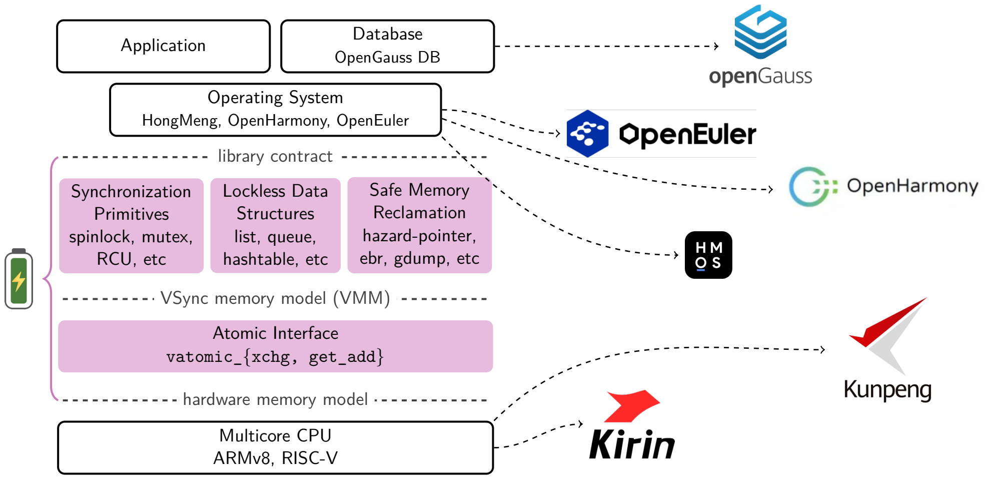
# libvsync: Solid concurrency foundations

github.com/open-s4c/libvsync



# libvsync: Solid concurrency foundations

github.com/open-s4c/libvsync



HUAWEI

THANK YOU

非常感谢你