UNIVERSITY OF CAMBRIDGE

Lynx: Using OS and Hardware Support for Fast Fine-Grained Inter-Core Communication

Konstantina Mitropoulou, Vasileios Porpodas,
Xiaochun Zhang and Timothy M. Jones

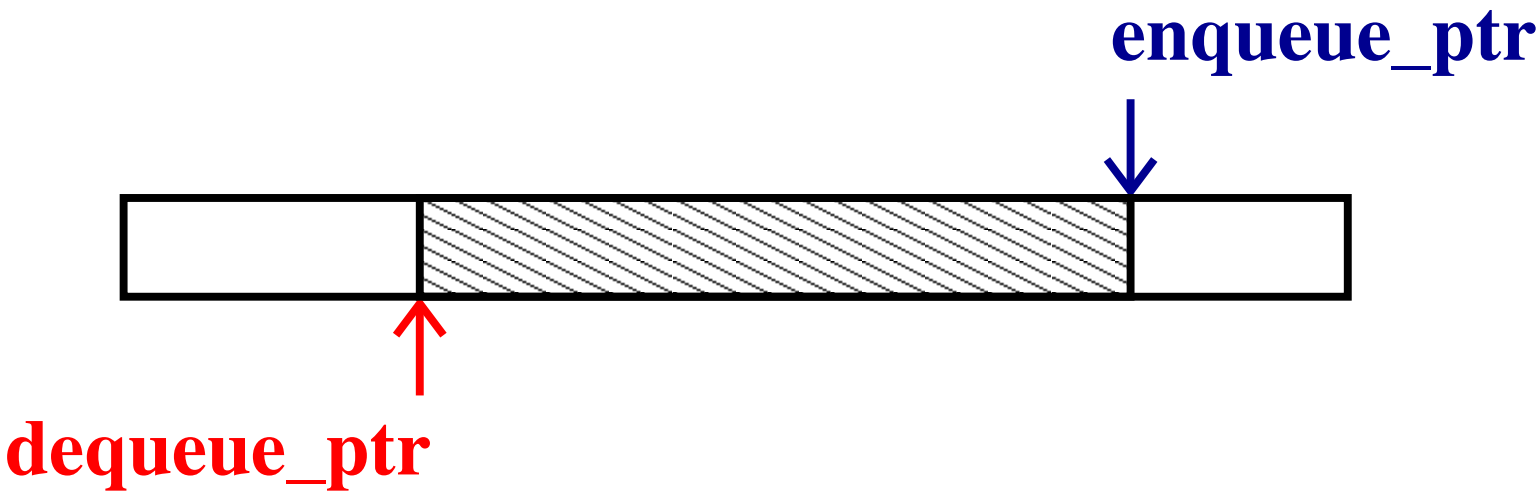
Computer Laboratory

ICS 2016, Istanbul

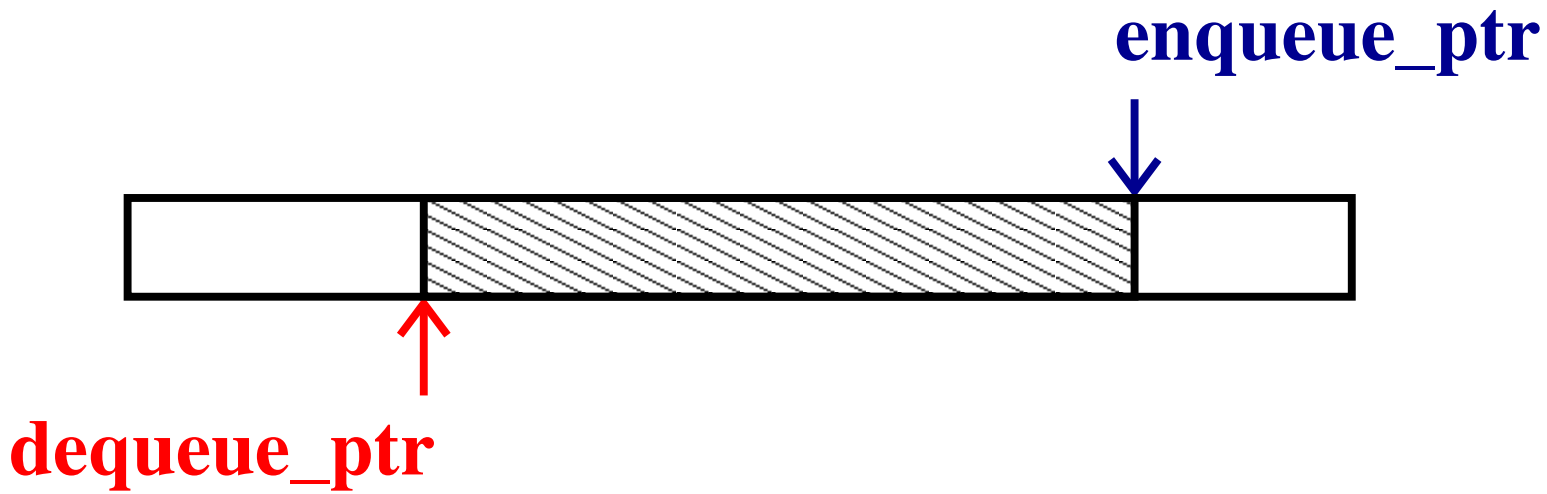
Outline

- Background:
  - Lamport's queue
  - Multi-section queue
- Lynx queue
- Performance evaluation

Lamport's Queue Bottlenecks

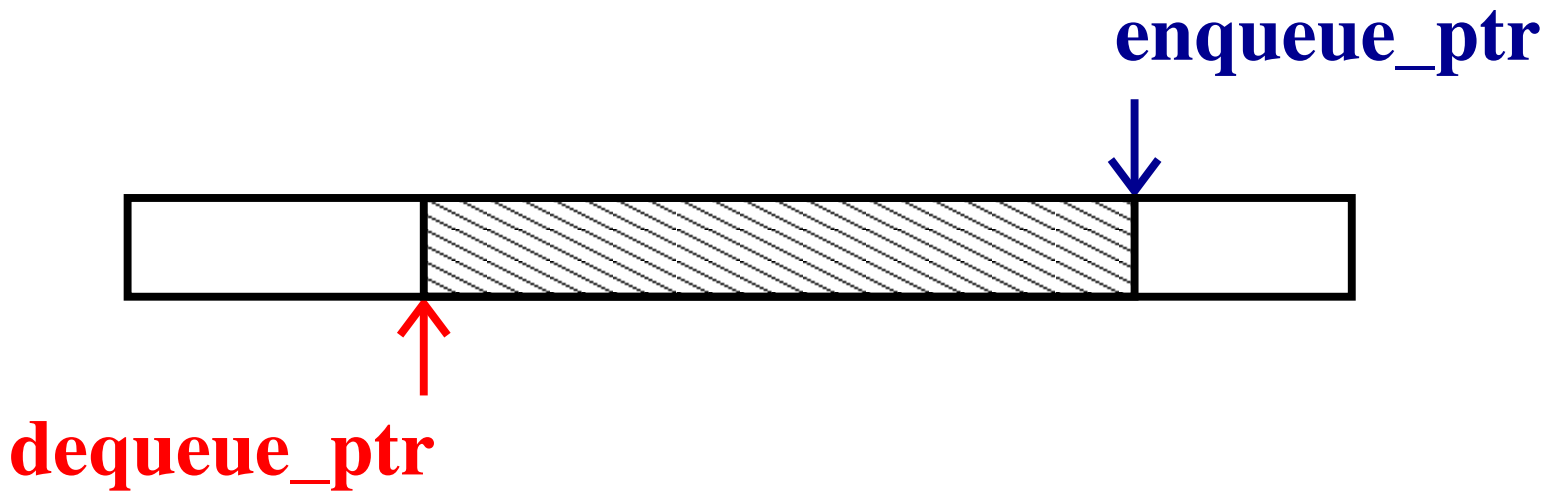


Lamport's Queue Bottlenecks




```
while(next_enqueue_ptr == dequeue_ptr){;}
```

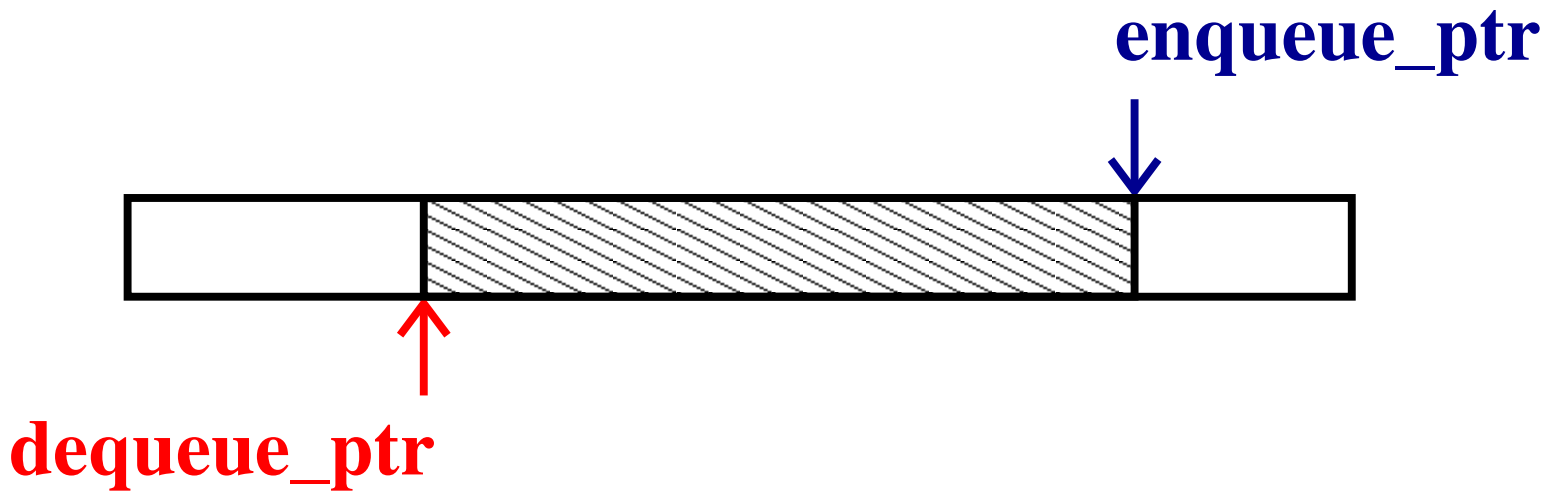
Lamport's Queue Bottlenecks



while(**next_enqueue_ptr** == **dequeue_ptr**){;}

Performance degradation due to:

Lamport's Queue Bottlenecks

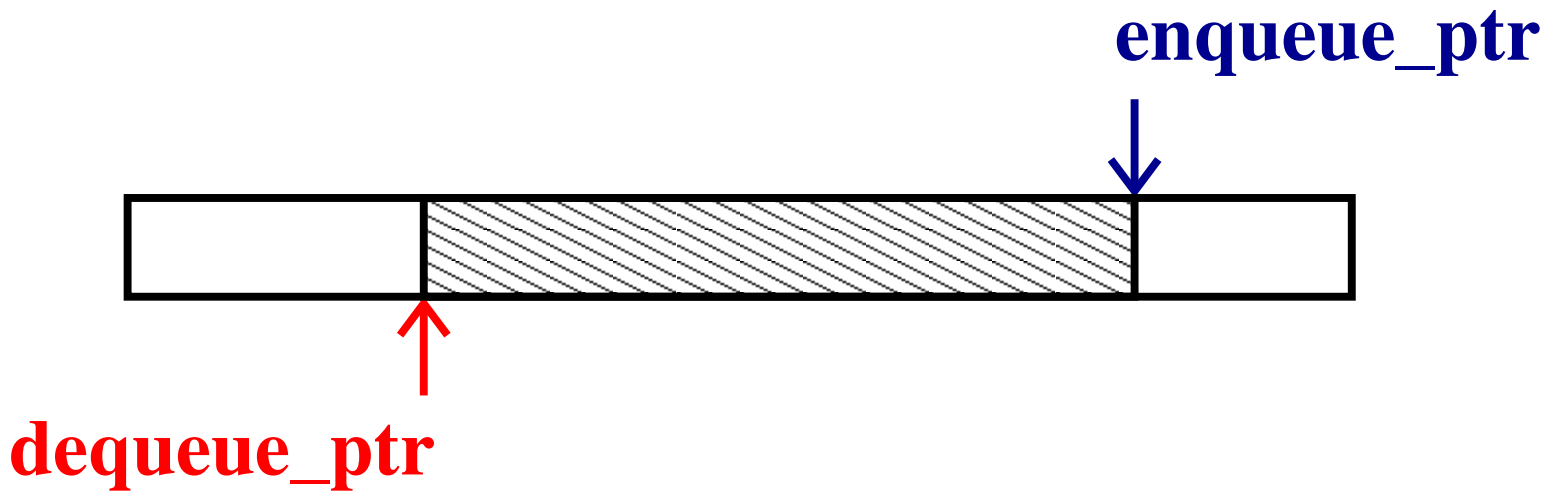


while(**next_enqueue_ptr** == **dequeue_ptr**){;}

Performance degradation due to:

- Frequent thread synchronisation

Lamport's Queue Bottlenecks

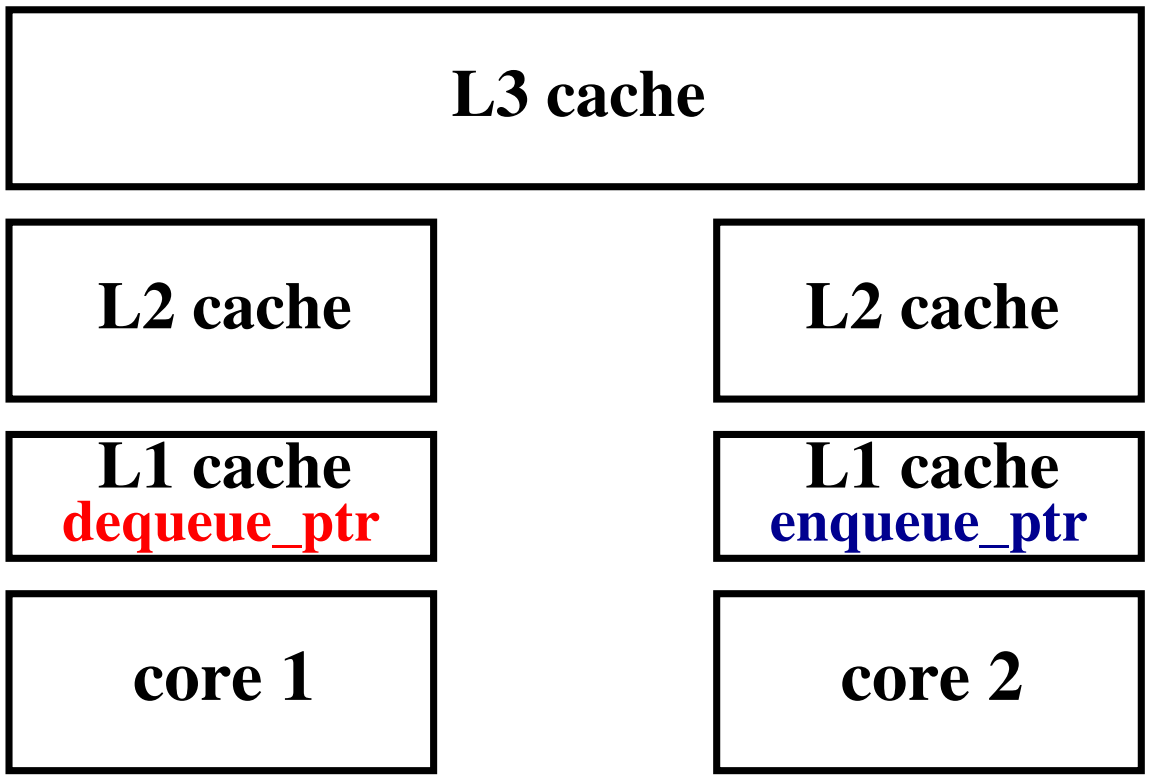


while(**next_enqueue_ptr** == **dequeue_ptr**){;}

Performance degradation due to:

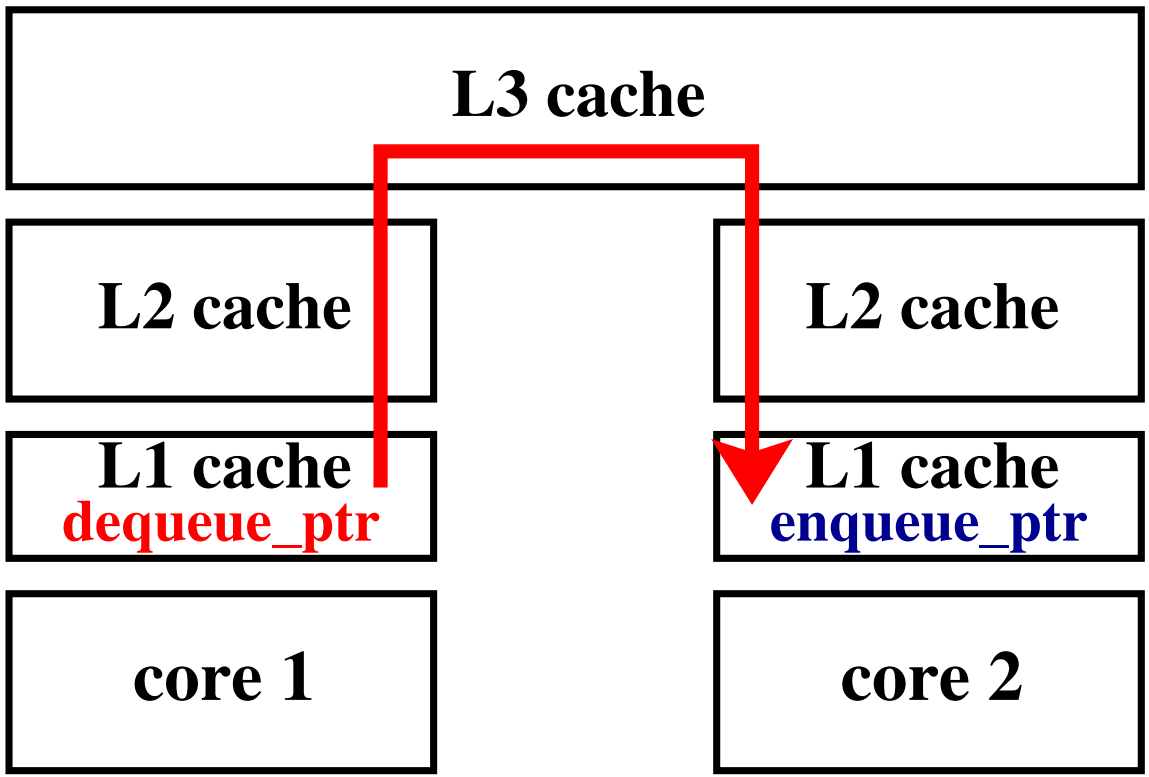
- Frequent thread synchronisation
- Cache ping-pong

Cache Ping-Pong




```
while(next_enqueue_ptr == dequeue_ptr){;}
```

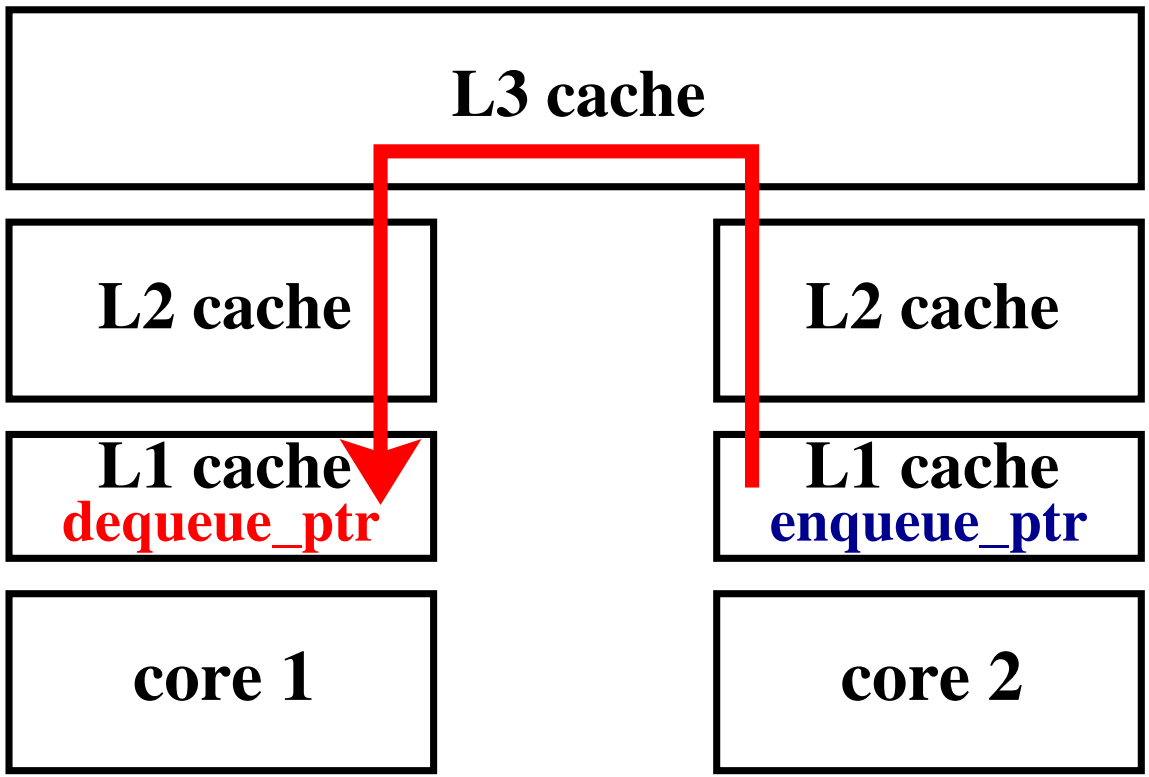
Cache Ping-Pong



while(**next_enqueue_ptr** == **dequeue_ptr**){;}

- Queue pointers ping-pong across cache hierarchy

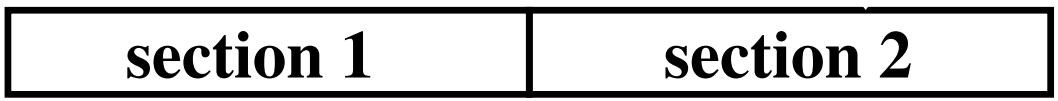
Cache Ping-Pong



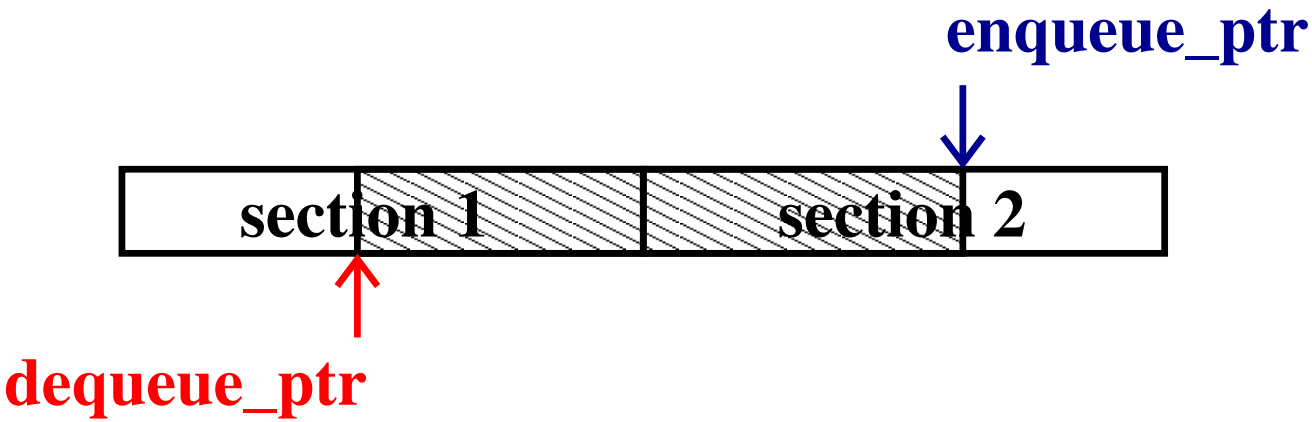
while(**next_dequeue_ptr** == **enqueue_ptr**){;}

- Queue pointers ping-pong across cache hierarchy

Multi-Section Queue(MSQ): state-of-the-art

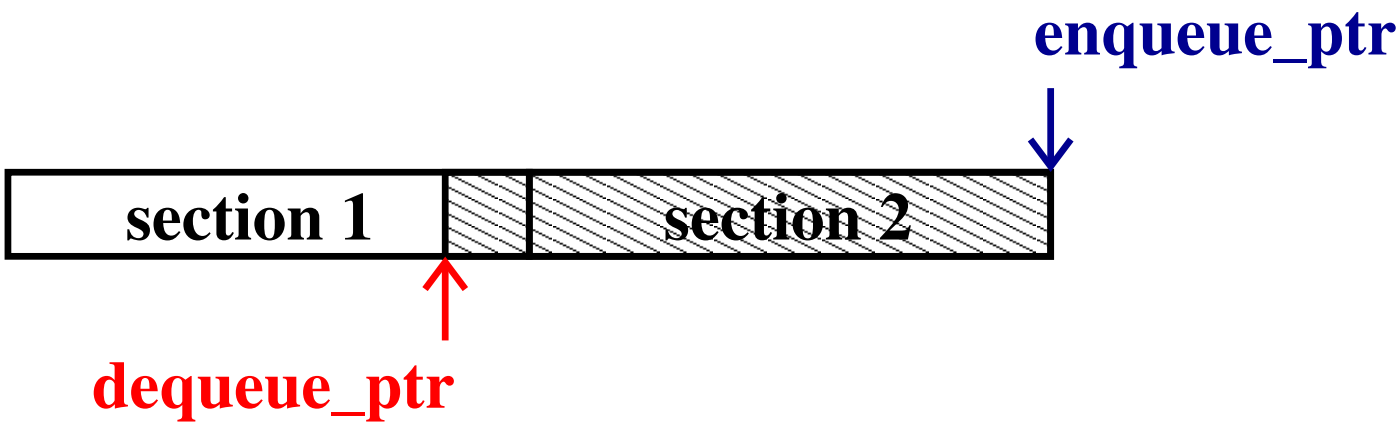


Multi-Section Queue(MSQ): state-of-the-art



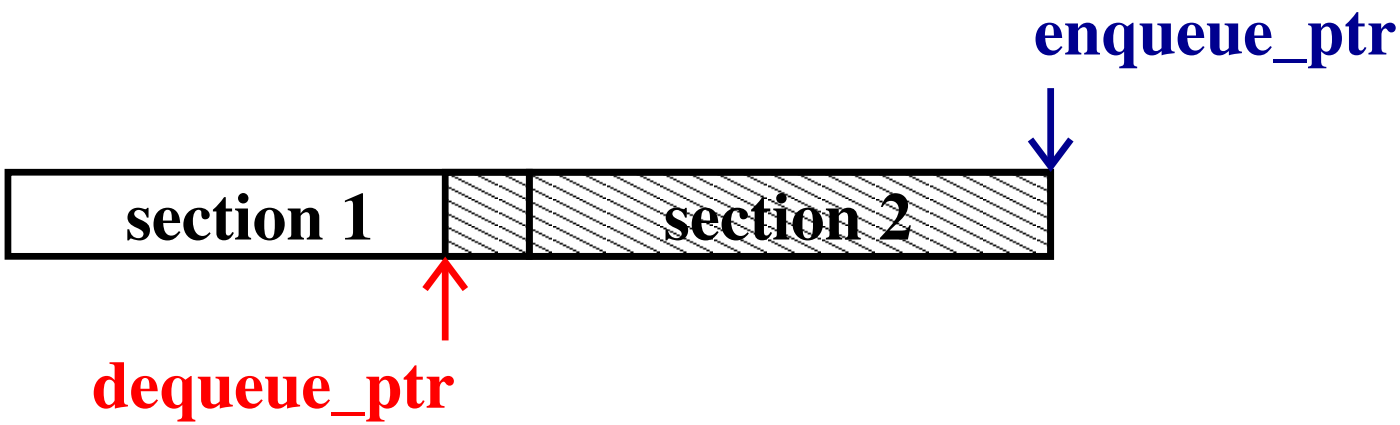
- Each section is exclusively used by one thread

Multi-Section Queue(MSQ): state-of-the-art



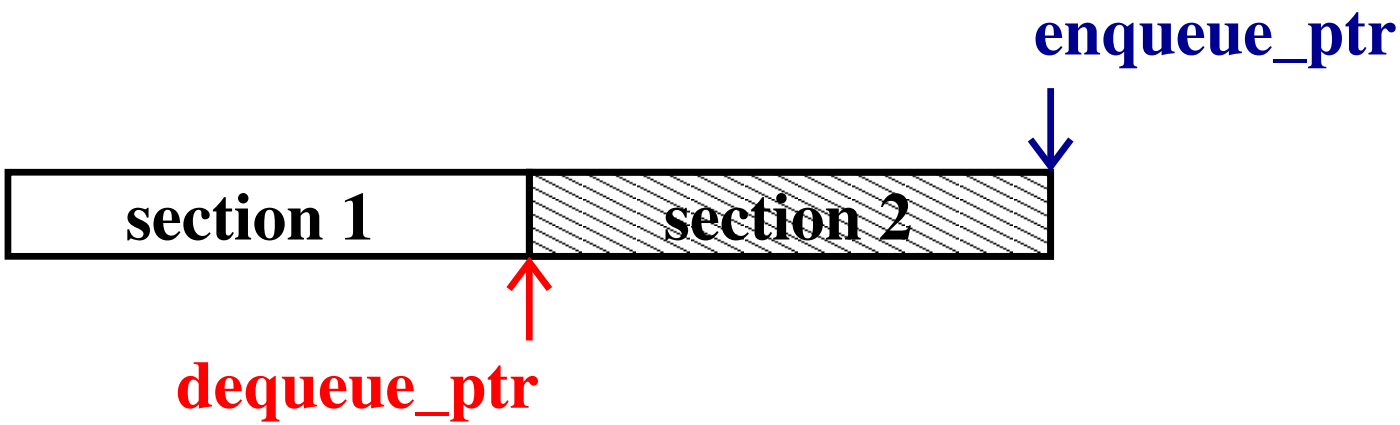
- Enqueue thread cannot access section 1 because dequeue thread still uses it

Multi-Section Queue(MSQ): state-of-the-art



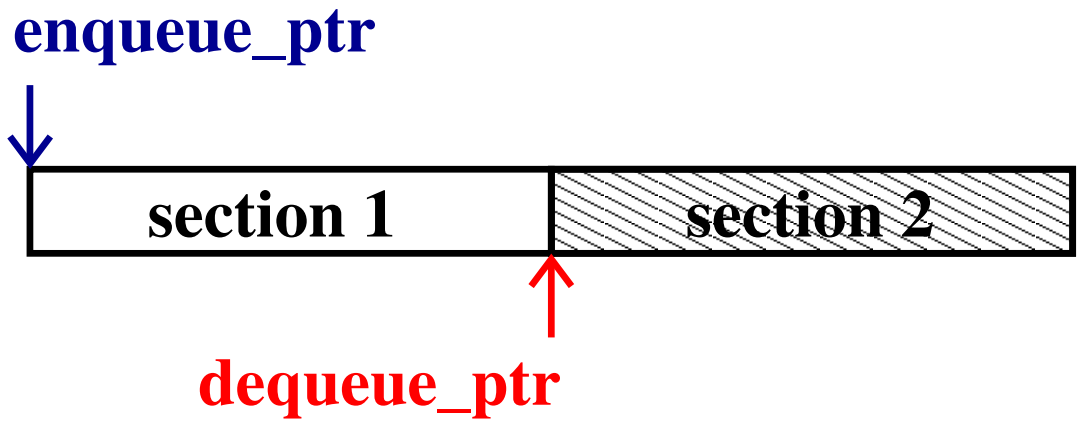
- Enqueue thread cannot access section 1 because dequeue thread still uses it
- Enqueue thread waits (spins) at the end of section 2

Multi-Section Queue(MSQ): state-of-the-art



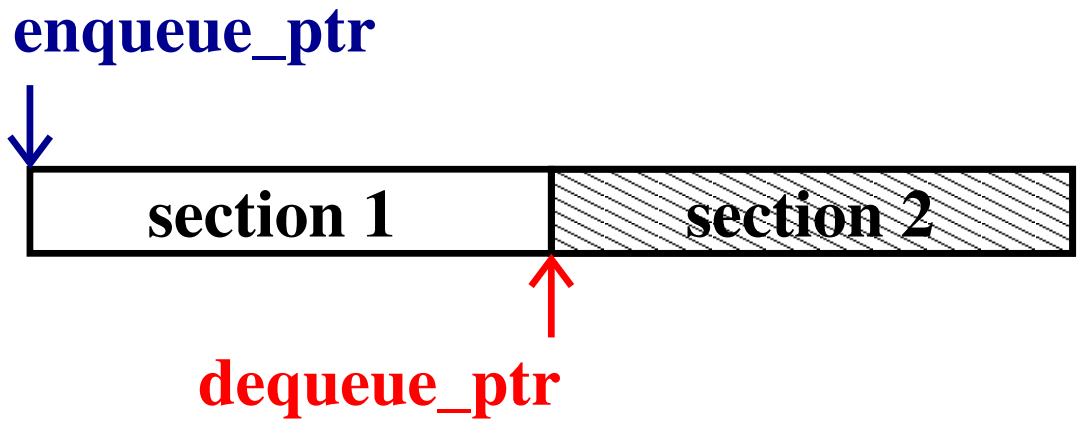
- Dequeue thread reached the end of section 1

Multi-Section Queue(MSQ): state-of-the-art



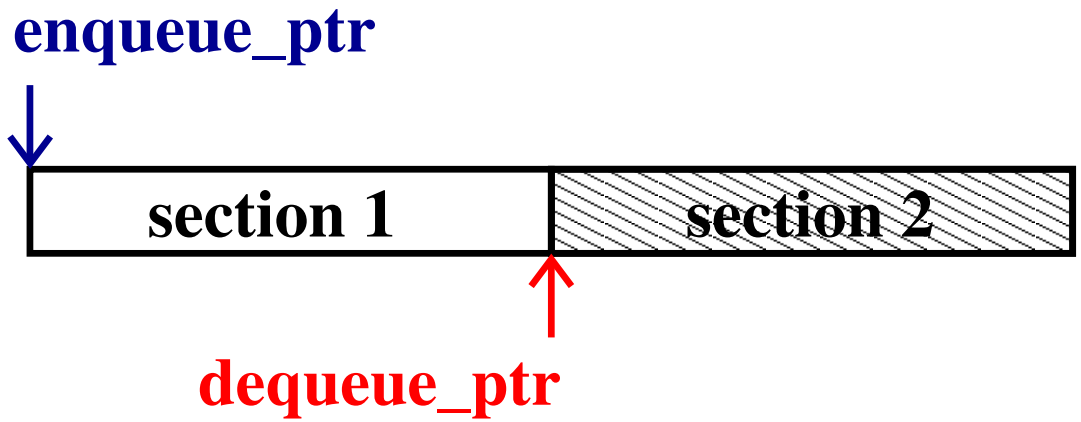
- Dequeue thread reached the end of section 1
- Enqueue thread enters section 1

Multi-Section Queue(MSQ): state-of-the-art



Performance optimisations:

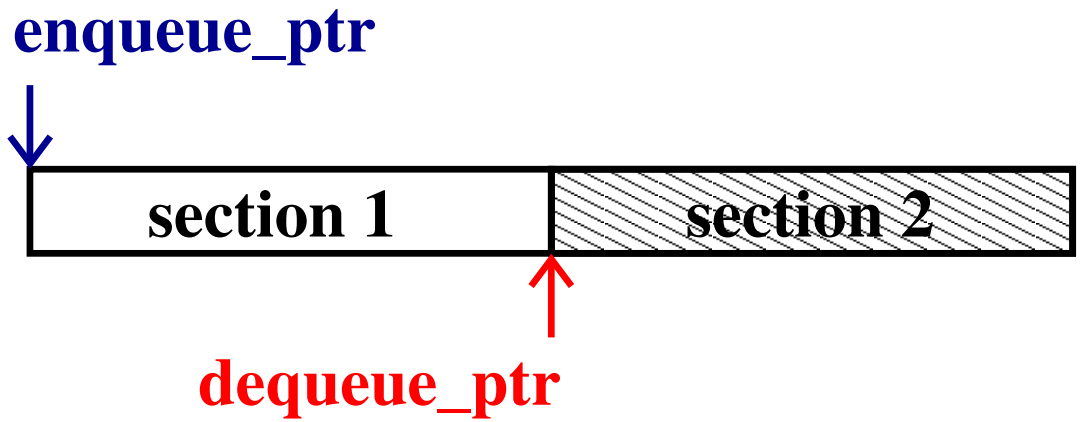
Multi-Section Queue(MSQ): state-of-the-art



Performance optimisations:

- Infrequent boundary checks (less frequent synchronisation)

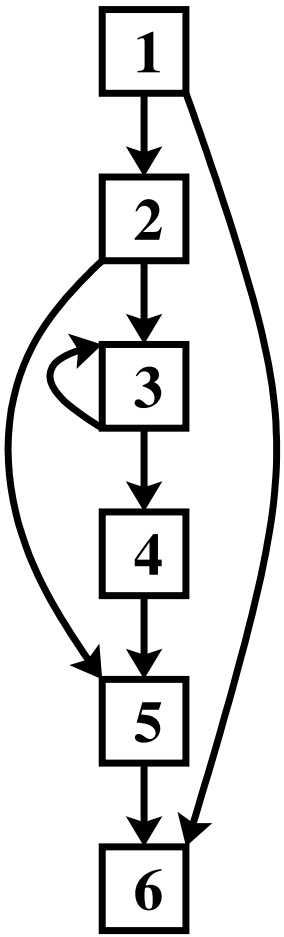
Multi-Section Queue(MSQ): state-of-the-art



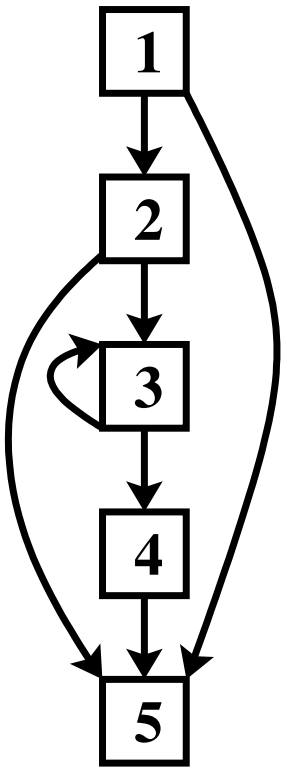
Performance optimisations:

- Infrequent boundary checks (less frequent synchronisation)
- Reduced cache ping-pong

MSQ Control-Flow Graph and Internals

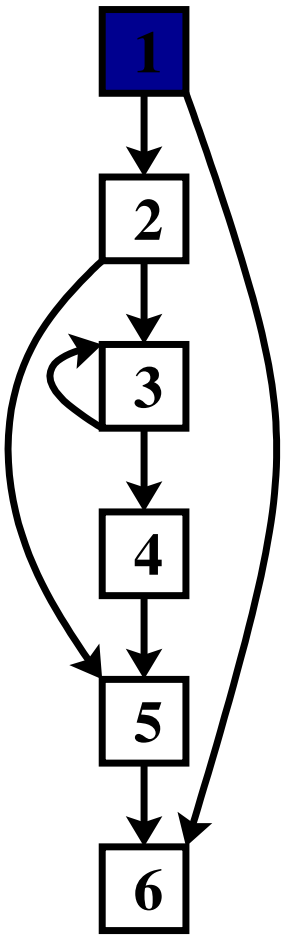


enqueue function



dequeue function

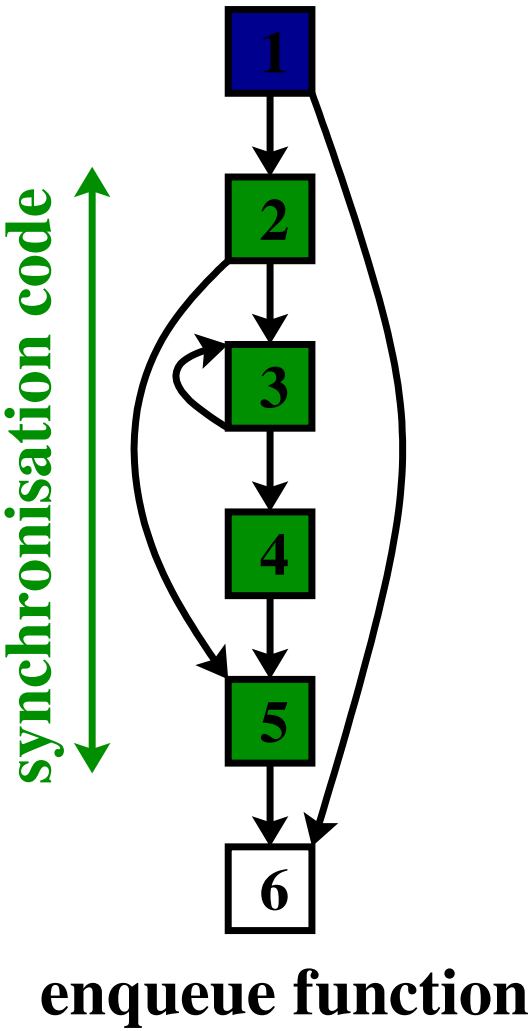
MSQ Control-Flow Graph and Internals



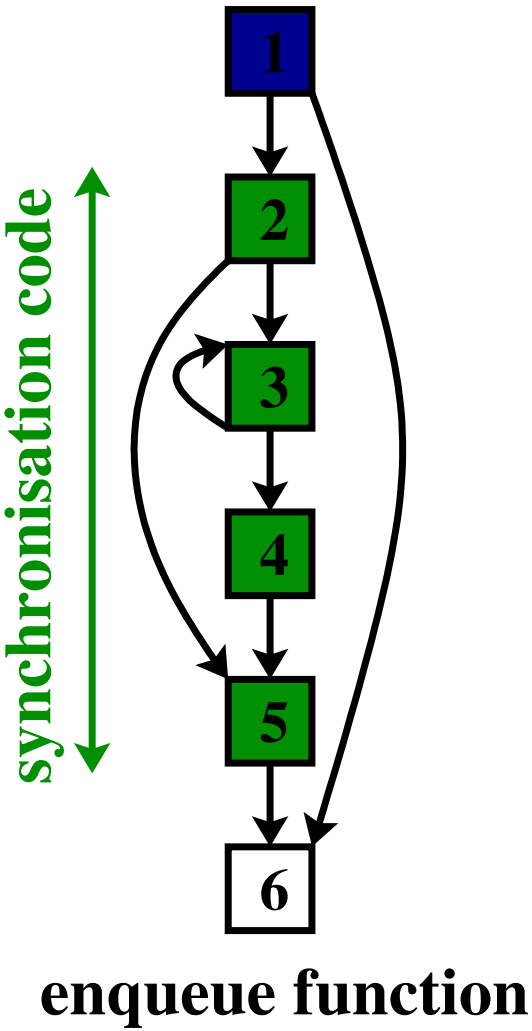
enqueue function

enqueue

MSQ Control-Flow Graph and Internals



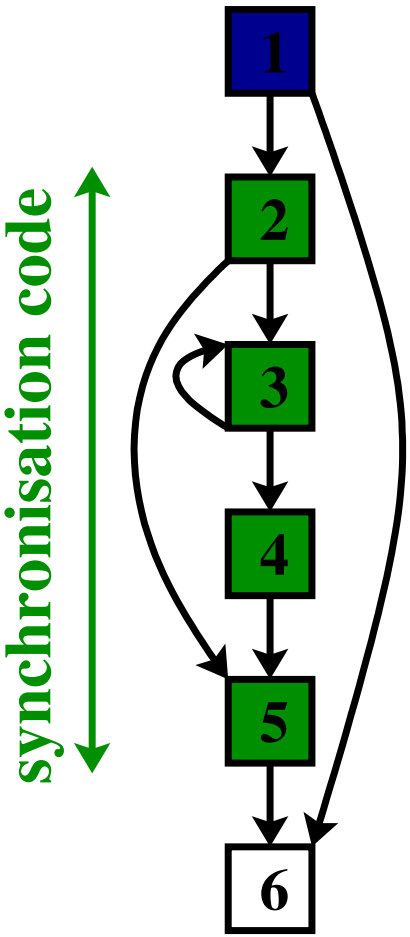
MSQ Control-Flow Graph and Internals



enqueue

checks if next section is free

MSQ Control-Flow Graph and Internals



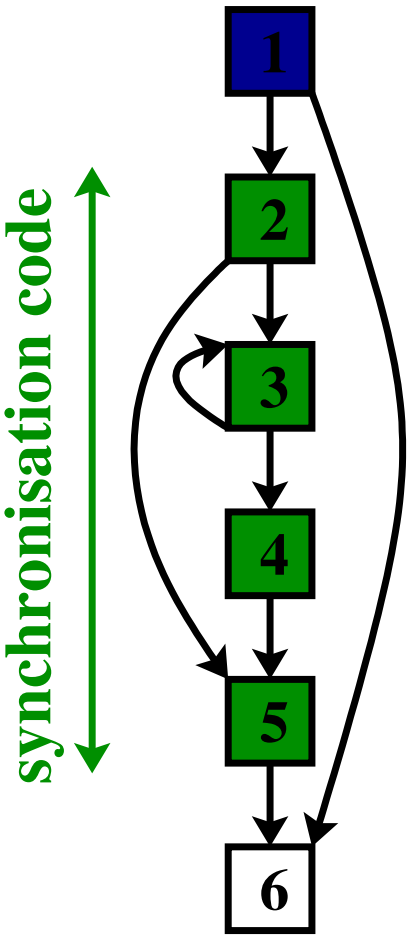
enqueue function

enqueue

checks if next section is free

spin loop

MSQ Control-Flow Graph and Internals



enqueue function

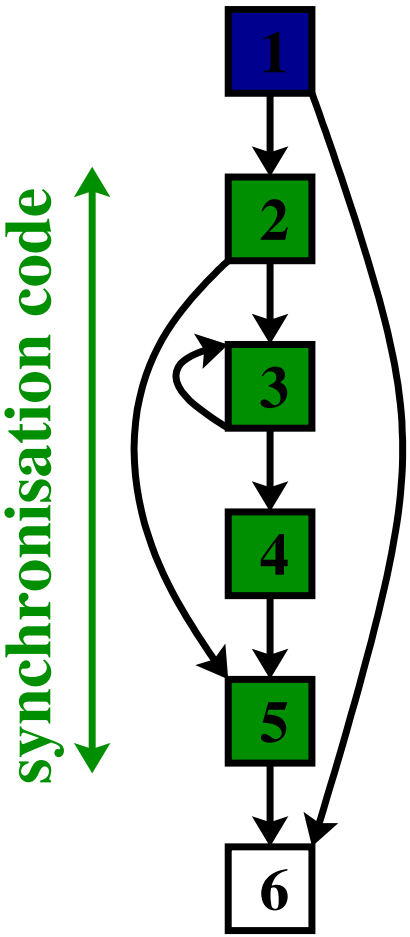
enqueue

checks if next section is free

spin loop

update local variables

MSQ Control-Flow Graph and Internals



enqueue function

enqueue

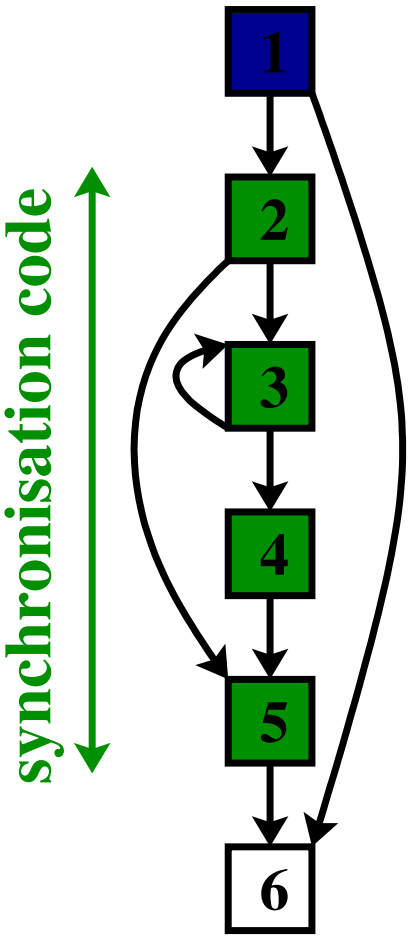
checks if next section is free

spin loop

update local variables

update shared variable

MSQ Control-Flow Graph and Internals



enqueue function

enqueue

checks if next section is free

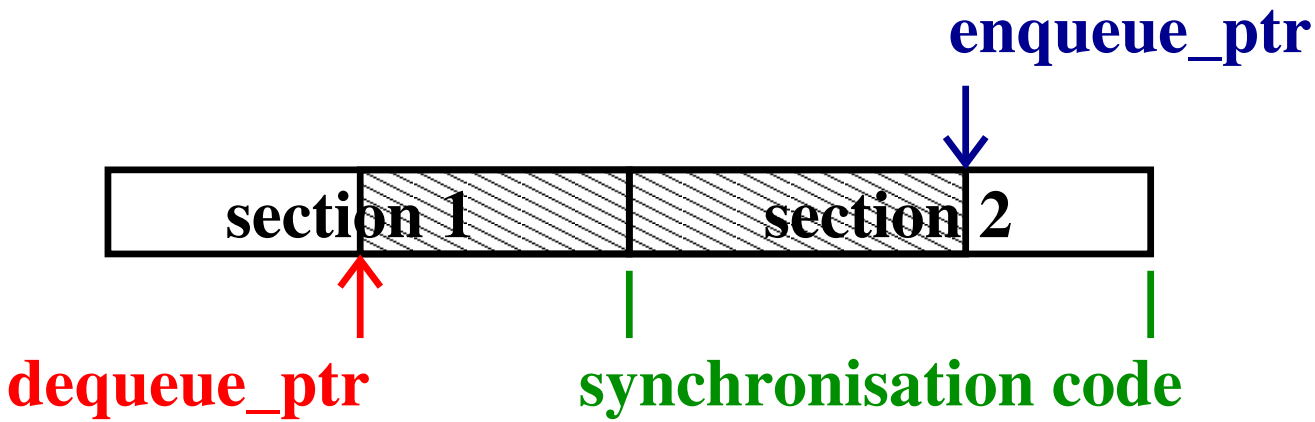
spin loop

update local variables

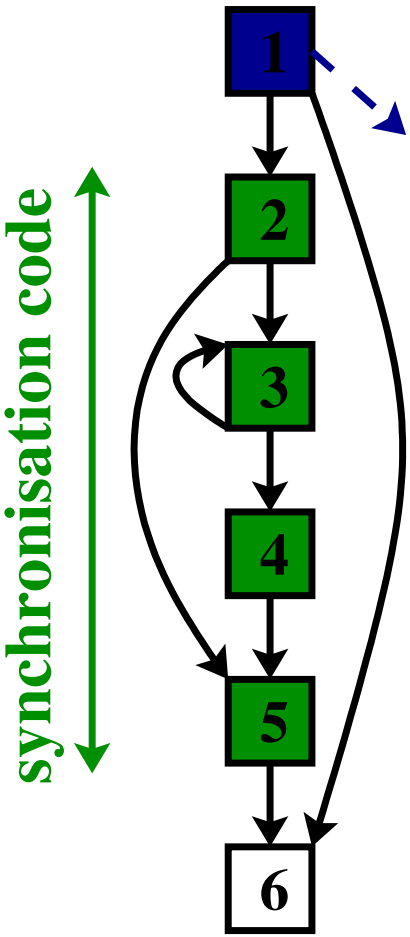
update shared variable

join basic–block

MSQ Control-Flow Graph and Internals



MSQ Control-Flow Graph and Internals

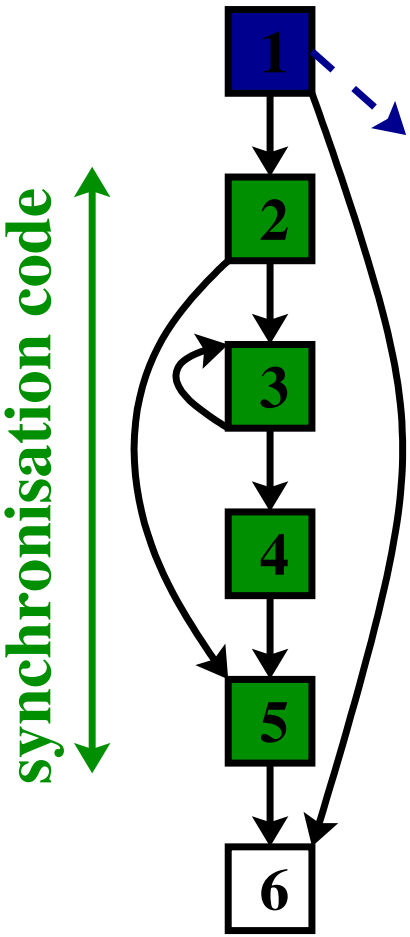


enqueue

```
lea rax, [rdx+8]

mov QWORD PTR [rdx], rcx

mov rdx, rax

and rdx, ROTATE MASK

test eax, SECTION_MASK

jne .L2
```

enqueue function

MSQ Control-Flow Graph and Internals

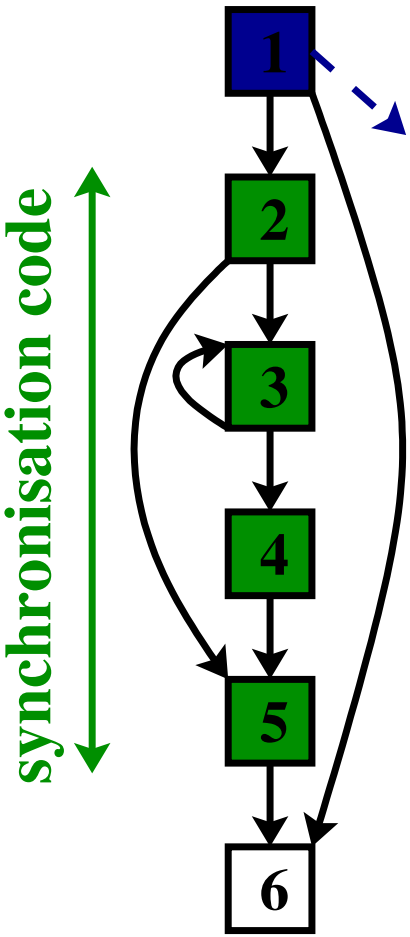


enqueue

```
lea rax, [rdx+8]                    incr pointer
mov QWORD PTR [rdx], rcx
mov rdx, rax
and rdx, ROTATE MASK
test eax, SECTION_MASK
jne .L2
```

enqueue function

MSQ Control-Flow Graph and Internals

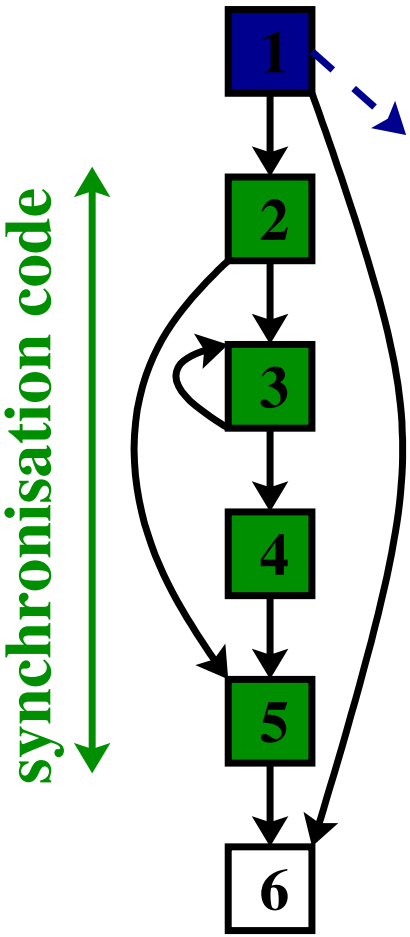


enqueue

```
lea rax, [rdx+8]                    incr pointer
mov QWORD PTR [rdx], rcx            store
mov rdx, rax
and rdx, ROTATE MASK
test eax, SECTION_MASK
jne .L2
```

enqueue function

MSQ Control-Flow Graph and Internals

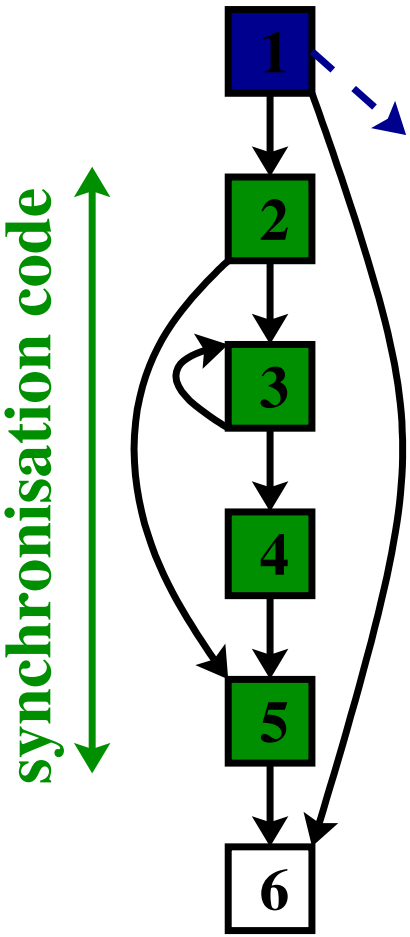


enqueue function

enqueue

lea rax, [rdx+8]	**incr pointer**
mov QWORD PTR [rdx], rcx	**store**
mov rdx, rax	**compiler's copy**
and rdx, ROTATE MASK	
test eax, SECTION_MASK	
jne .L2	

MSQ Control-Flow Graph and Internals

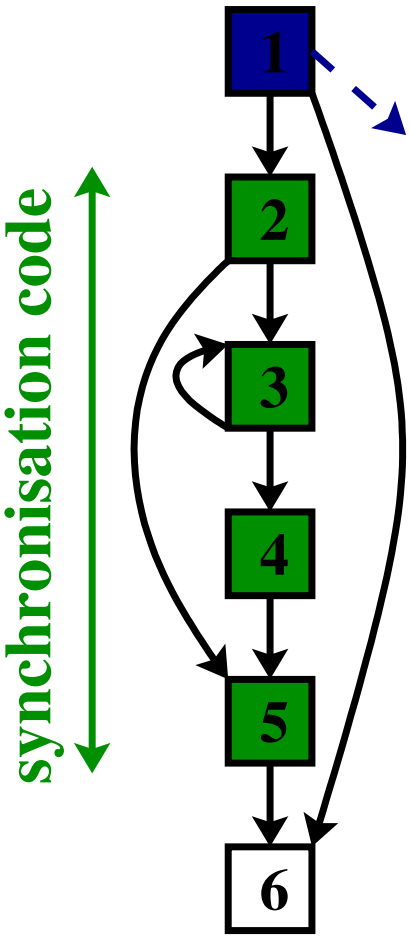


enqueue function

enqueue

lea rax, [rdx+8]	**incr pointer**
mov QWORD PTR [rdx], rcx	**store**
mov rdx, rax	**compiler's copy**
and rdx, ROTATE MASK	**rotate pointer**
test eax, SECTION_MASK	
jne .L2	

MSQ Control-Flow Graph and Internals

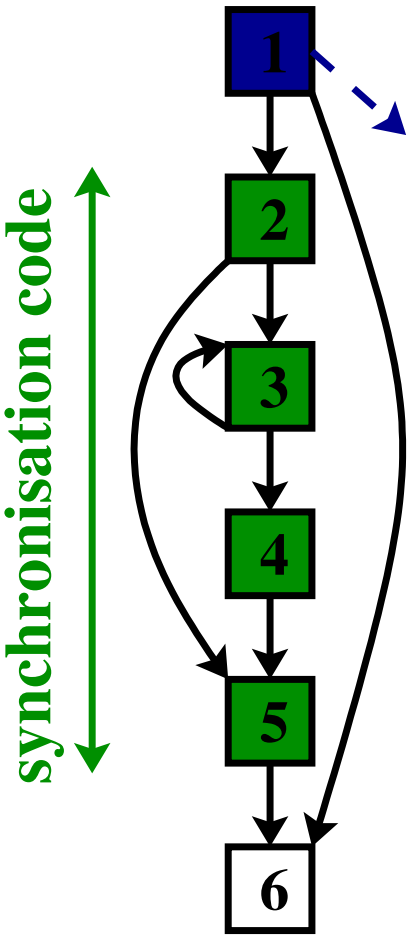


enqueue function

enqueue

lea rax, [rdx+8]	**incr pointer**
mov QWORD PTR [rdx], rcx	**store**
mov rdx, rax	**compiler's copy**
and rdx, ROTATE MASK	**rotate pointer**
test eax, SECTION_MASK	**end of section**
jne .L2	

MSQ Control-Flow Graph and Internals

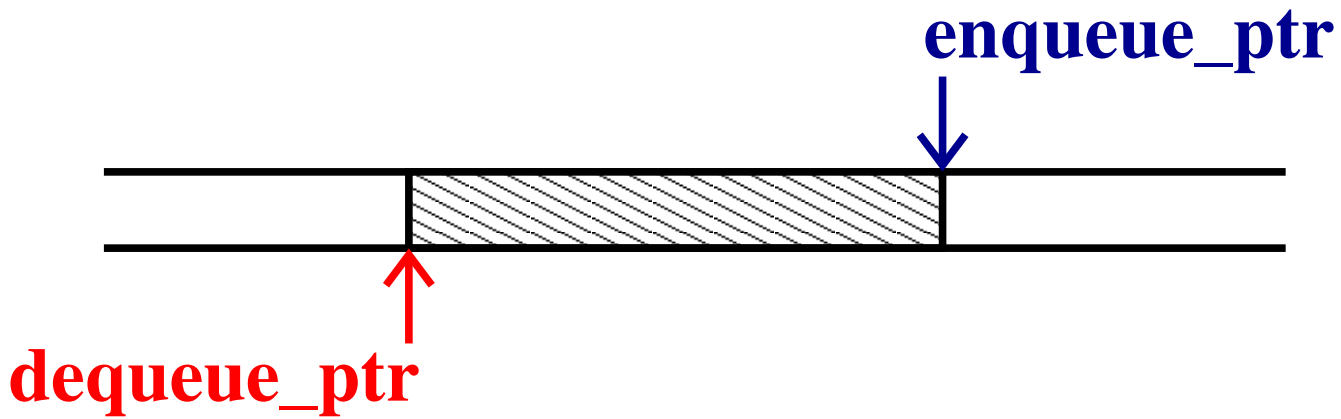


enqueue function

enqueue

lea rax, [rdx+8]	**incr pointer**
mov QWORD PTR [rdx], rcx	**store**
mov rdx, rax	**compiler's copy**
and rdx, ROTATE MASK	**rotate pointer**
test eax, SECTION_MASK	**end of section**
jne .L2	**skip sync code**

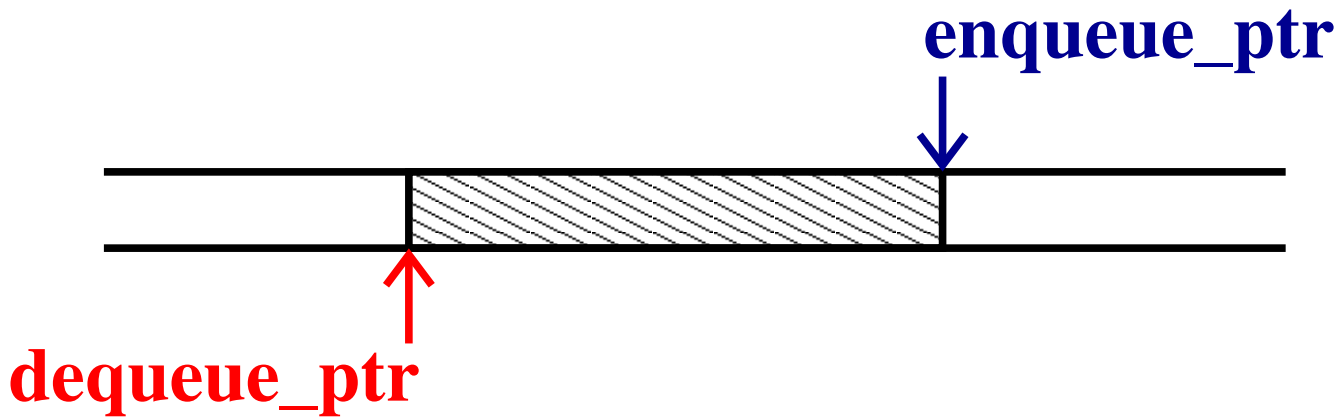
Optimal Queue



Optimal queue features:

- infinite size

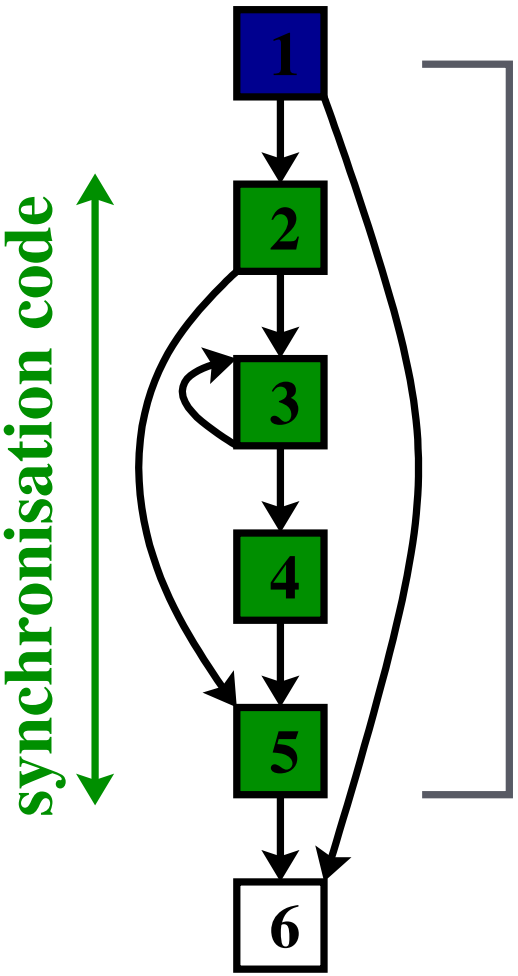
Optimal Queue



Optimal queue features:

- infinite size
- 2 instructions overhead
  1. pointer increment
  2. store into the queue

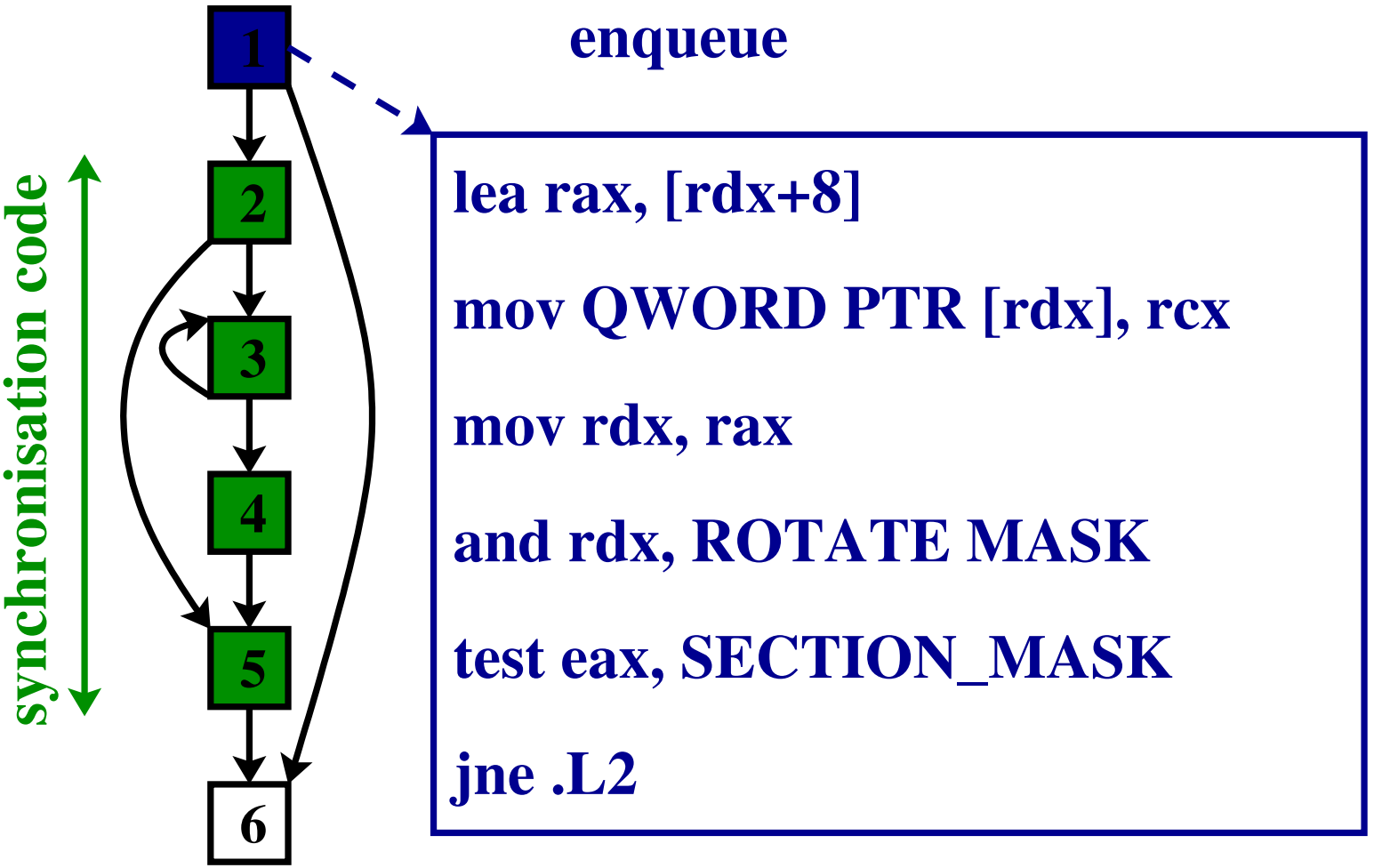
Lynx: Just 2 instructions overhead



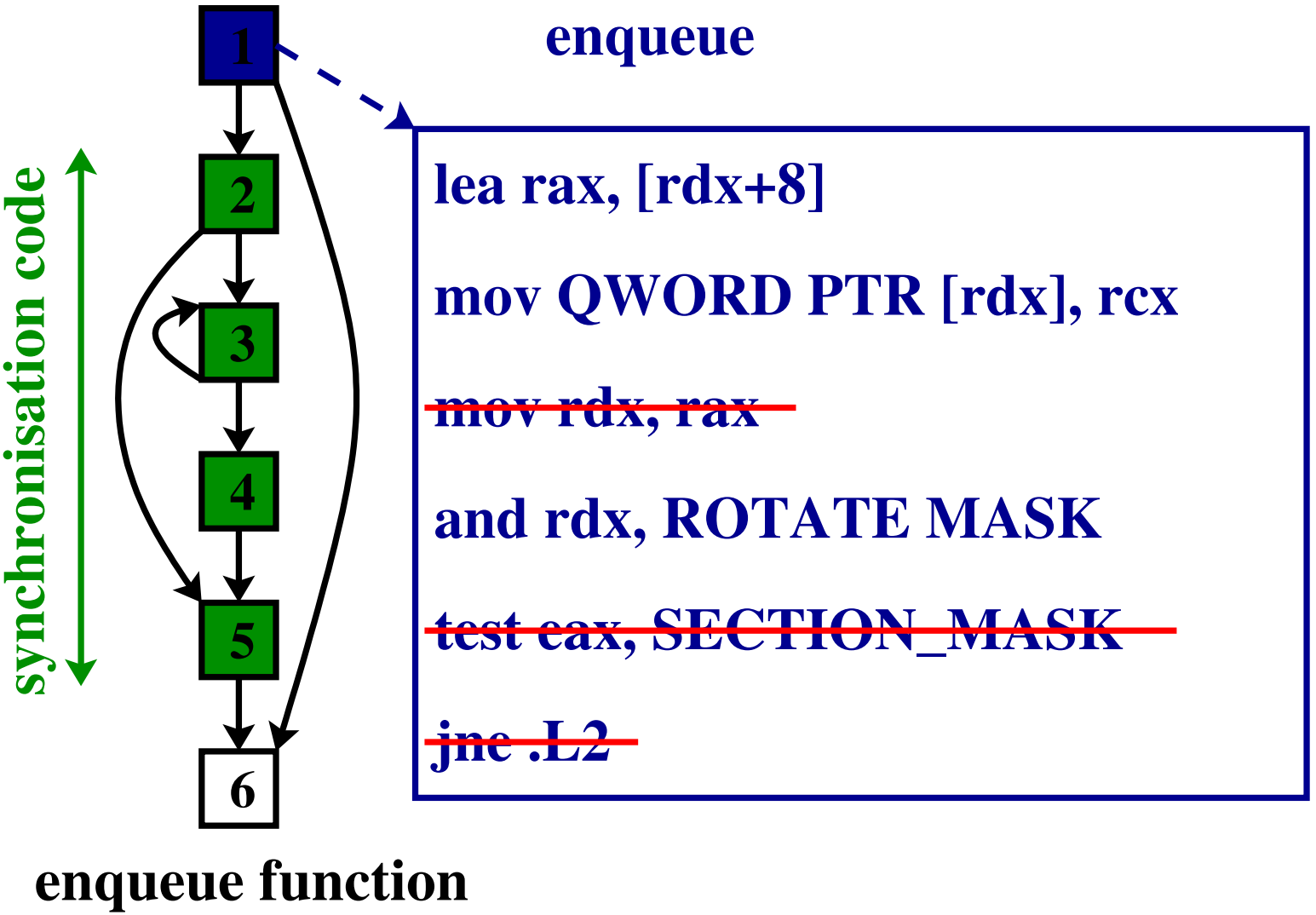
enqueue function

Lynx removes part of enqueue (boundary checks) and all the synchronisation overhead off the critical path

Lynx(1): H/W triggered Synchronisation

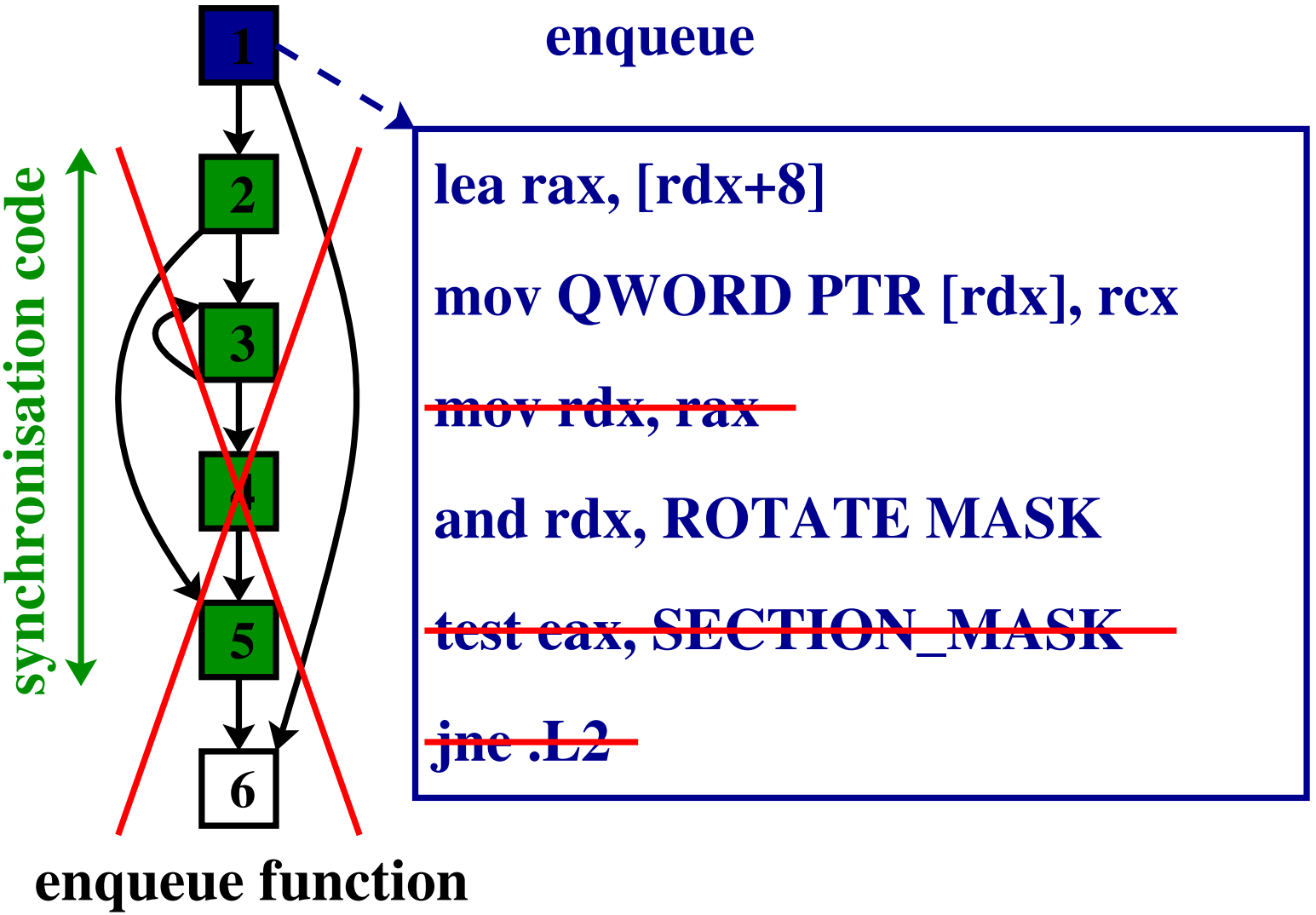


Lynx(1): H/W triggered Synchronisation

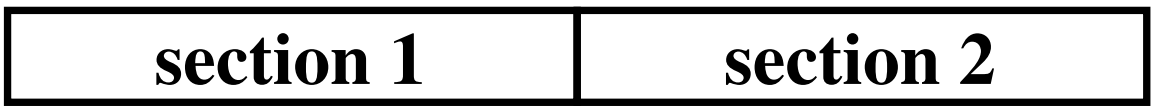


enqueue function

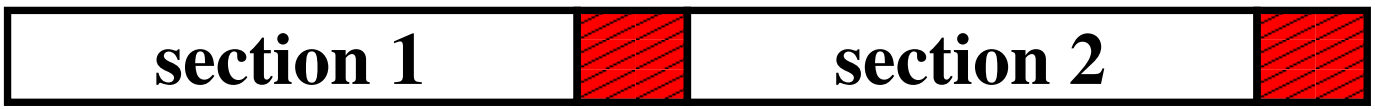
Lynx(1): H/W triggered Synchronisation



Lynx(1): H/W triggered Synchronisation



Lynx(1): H/W triggered Synchronisation

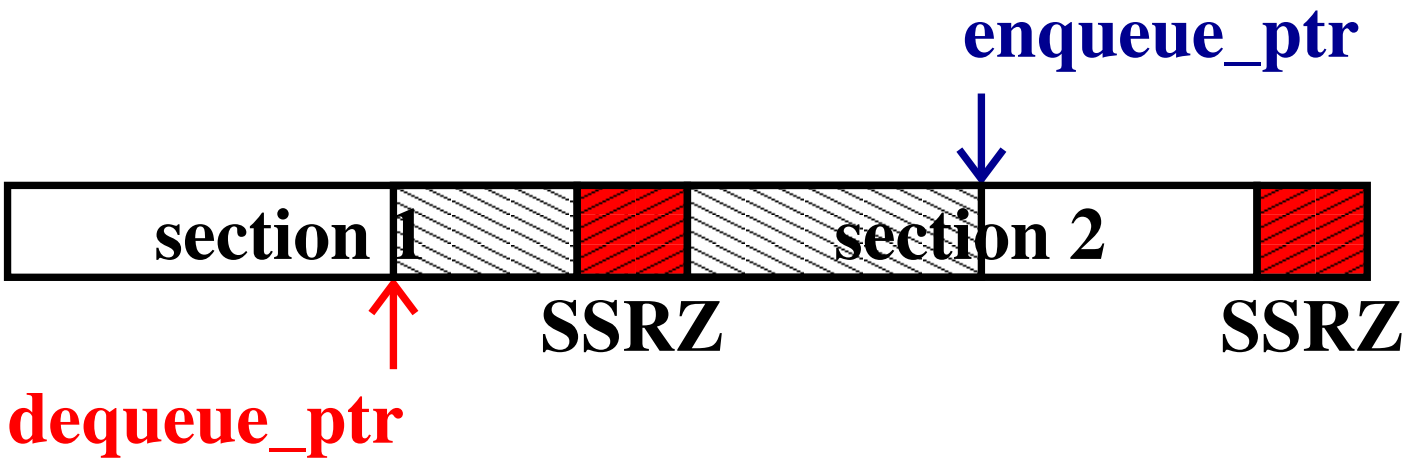


- A red zone is a non-read and non-write part of memory

Lynx(1): H/W triggered Synchronisation

- SSRZ: Section Synchronisation Red-Zone

Lynx(1): H/W triggered Synchronisation



Lynx(1): H/W triggered Synchronisation

Lynx(1): H/W triggered Synchronisation

Lynx(1): H/W triggered Synchronisation

Lynx's handler checks:

Lynx(1): H/W triggered Synchronisation

Lynx's handler checks:

- whether the SIG_SEGV is from the queue or the system

Lynx(1): H/W triggered Synchronisation

Lynx's handler checks:

- whether the SIG_SEGV is from the queue or the system
- which thread raised the exception

Lynx(1): H/W triggered Synchronisation

Lynx's handler checks:

- whether the SIG_SEGV is from the queue or the system
- which thread raised the exception
- if the thread is in section 1 or 2

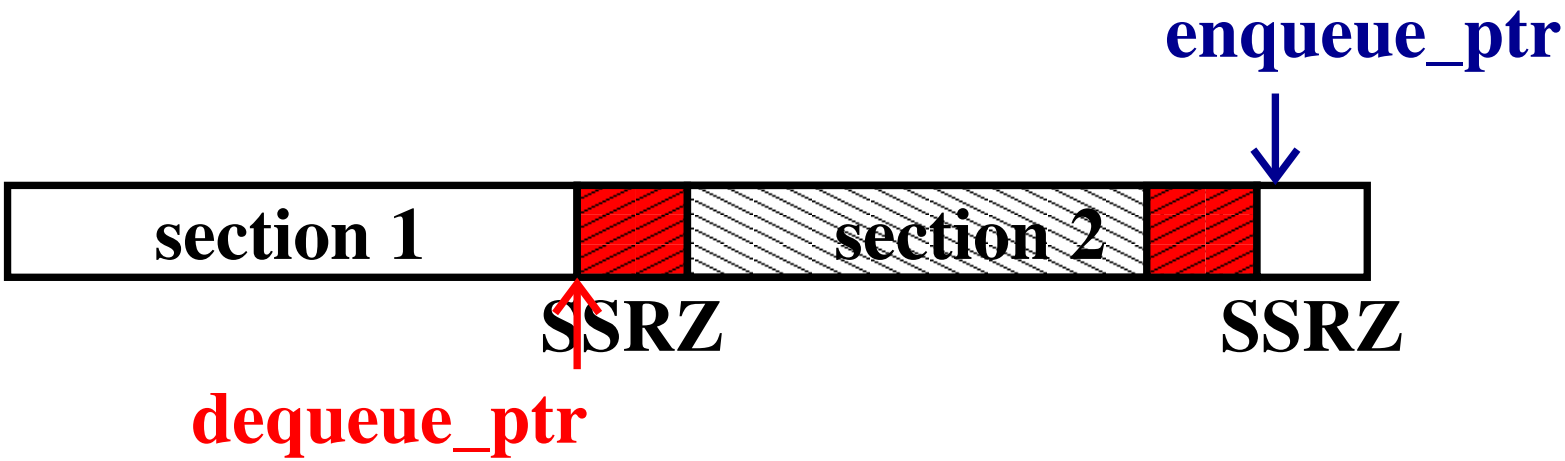
Lynx(1): H/W triggered Synchronisation

Lynx's handler checks:

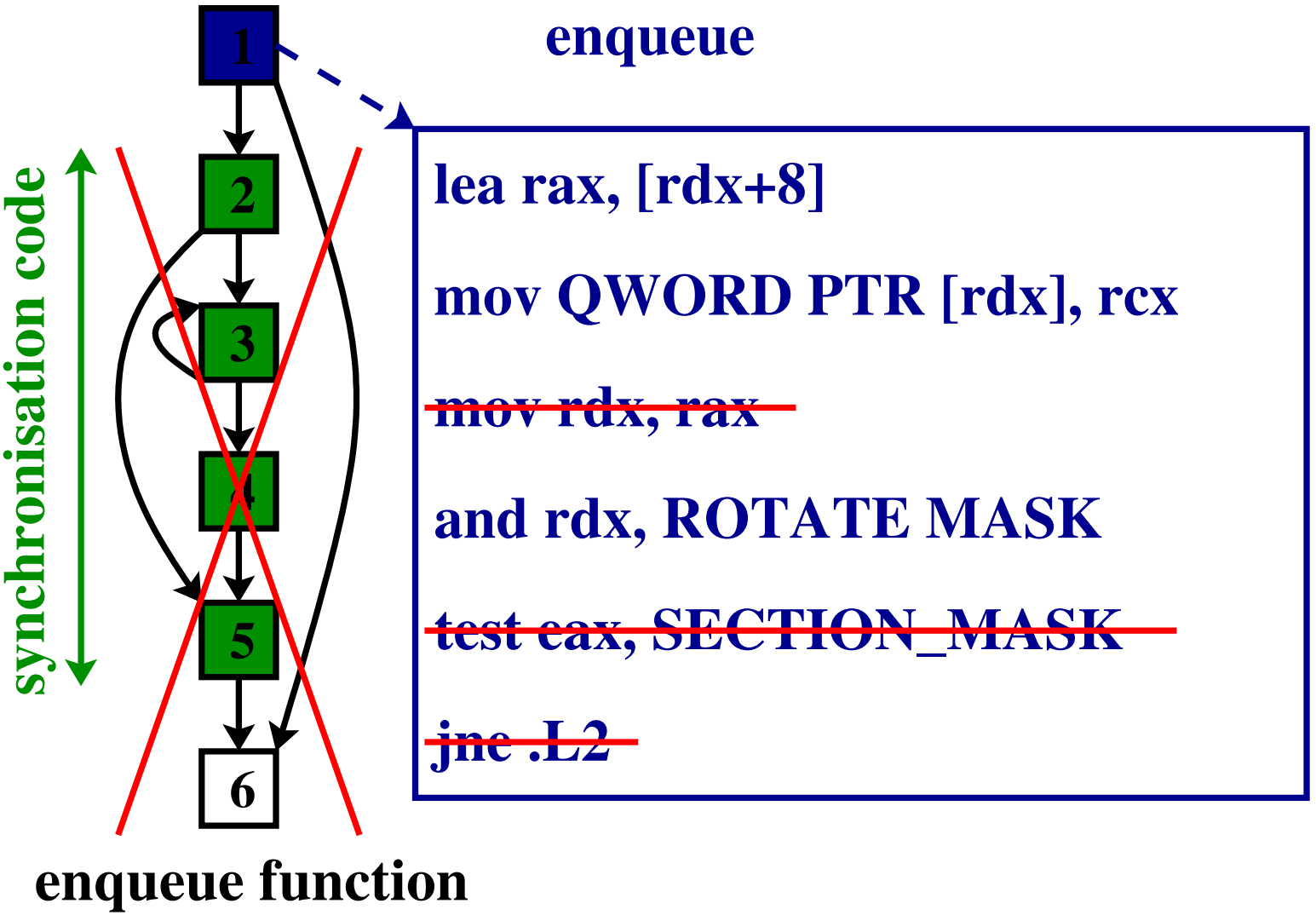
- whether the SIG_SEGV is from the queue or the system
- which thread raised the exception
- if the thread is in section 1 or 2
- if the next section is free

Lynx(1): H/W triggered Synchronisation

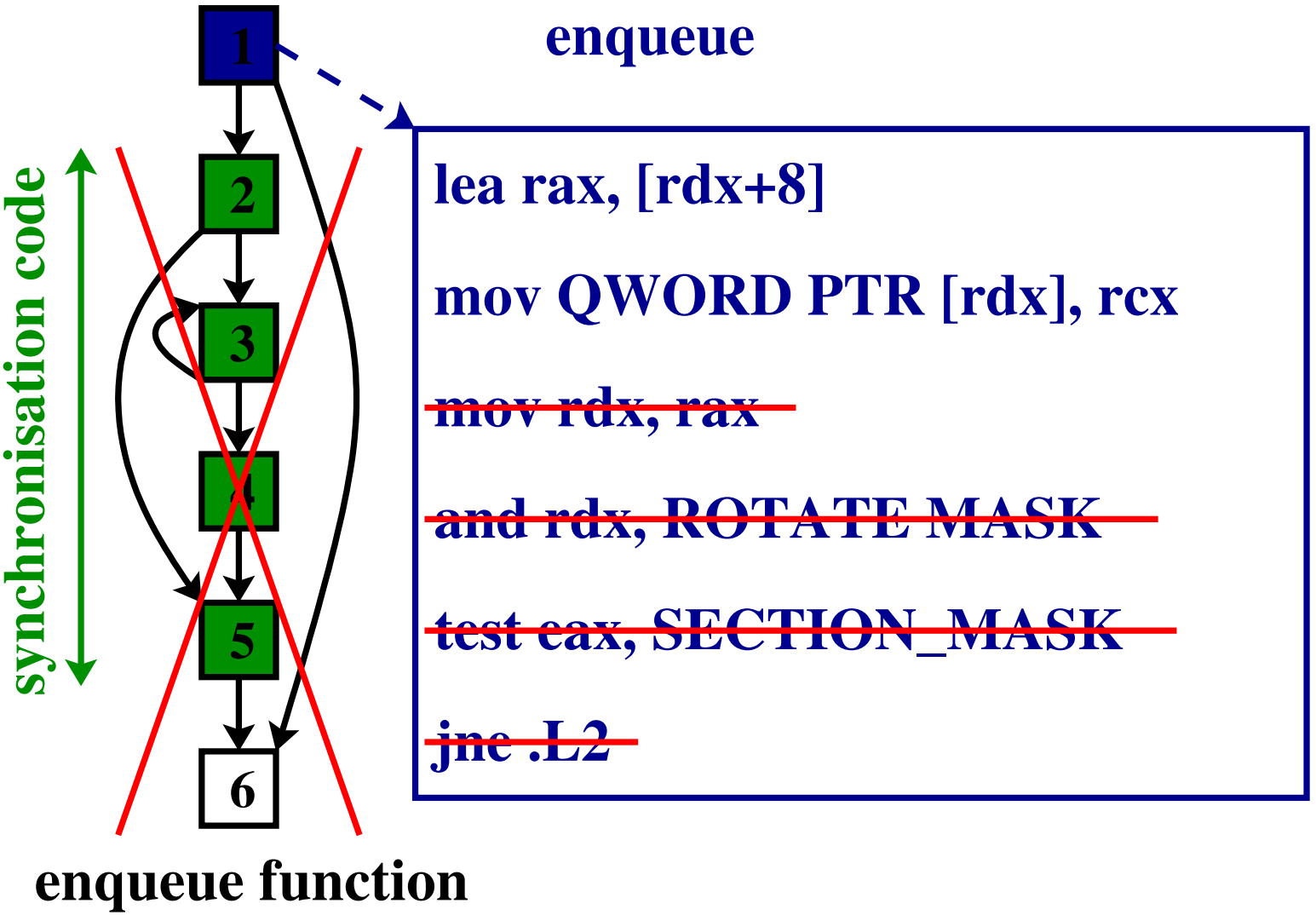
Lynx(1): H/W triggered Synchronisation



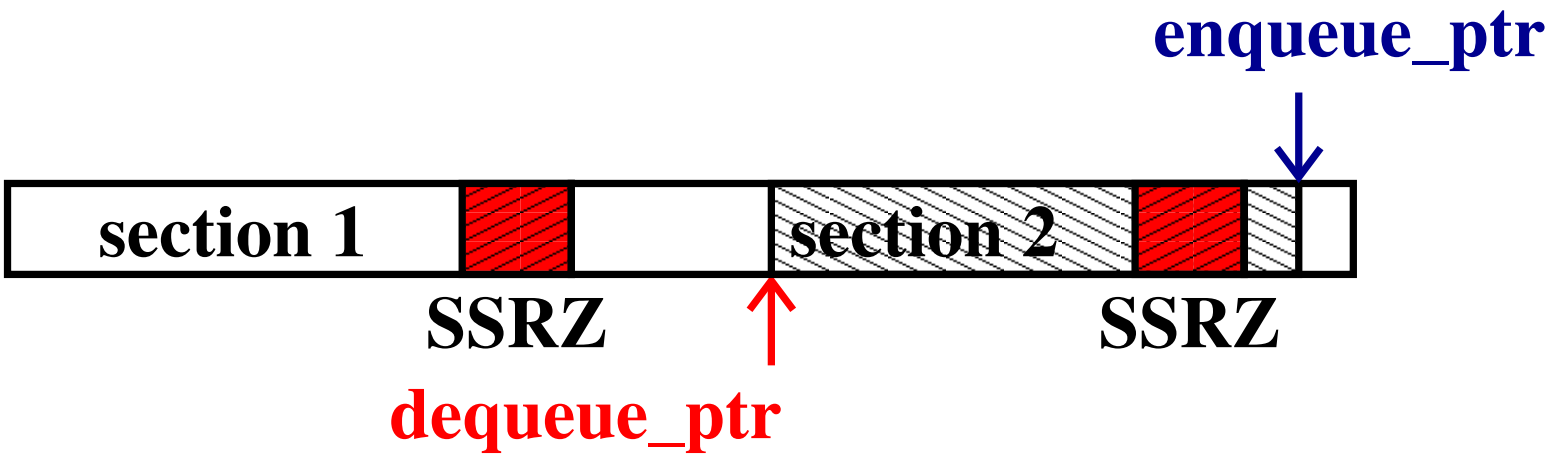
Lynx(2): H/W triggered Pointer Rotation



Lynx(2): H/W triggered Pointer Rotation

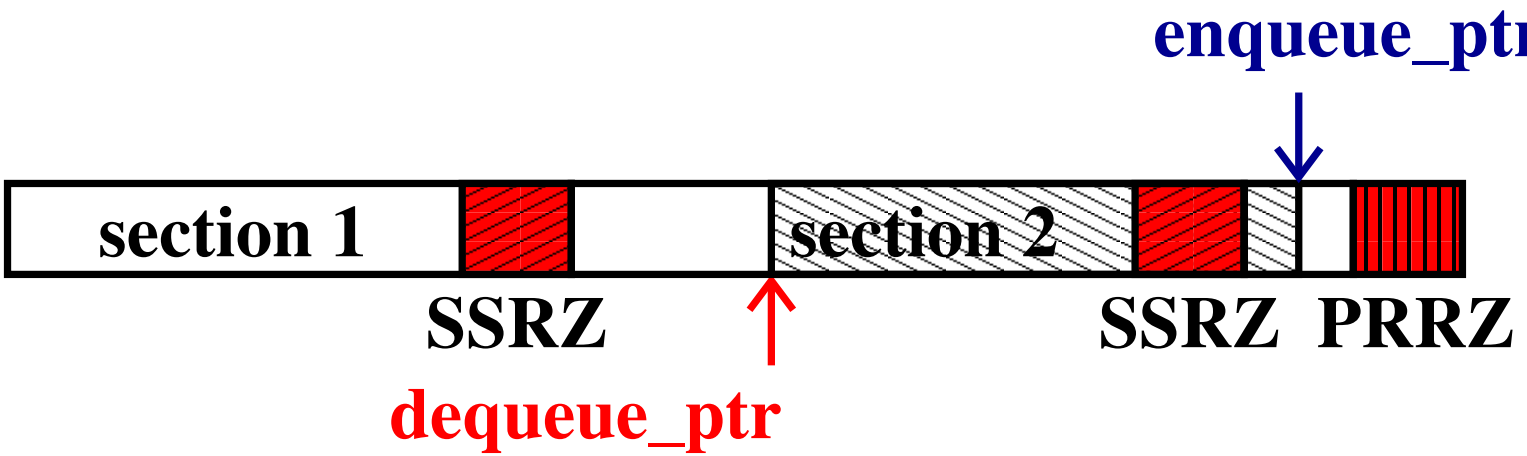


Lynx(2): H/W triggered Pointer Rotation



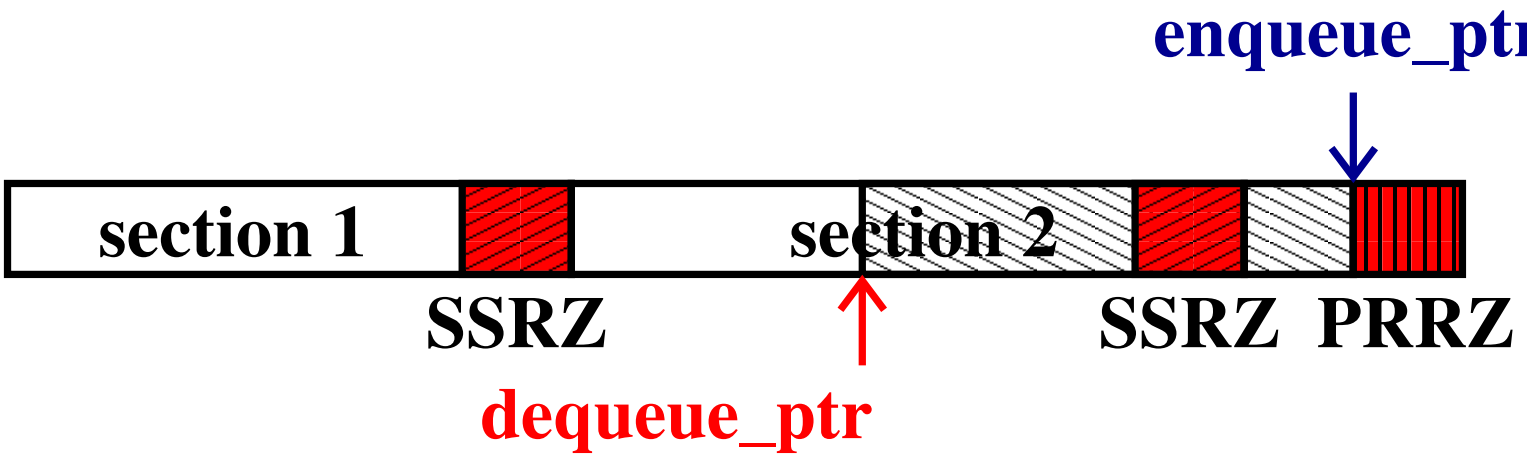
- SSRZ: Section Synchronisation Red-Zone

Lynx(2): H/W triggered Pointer Rotation



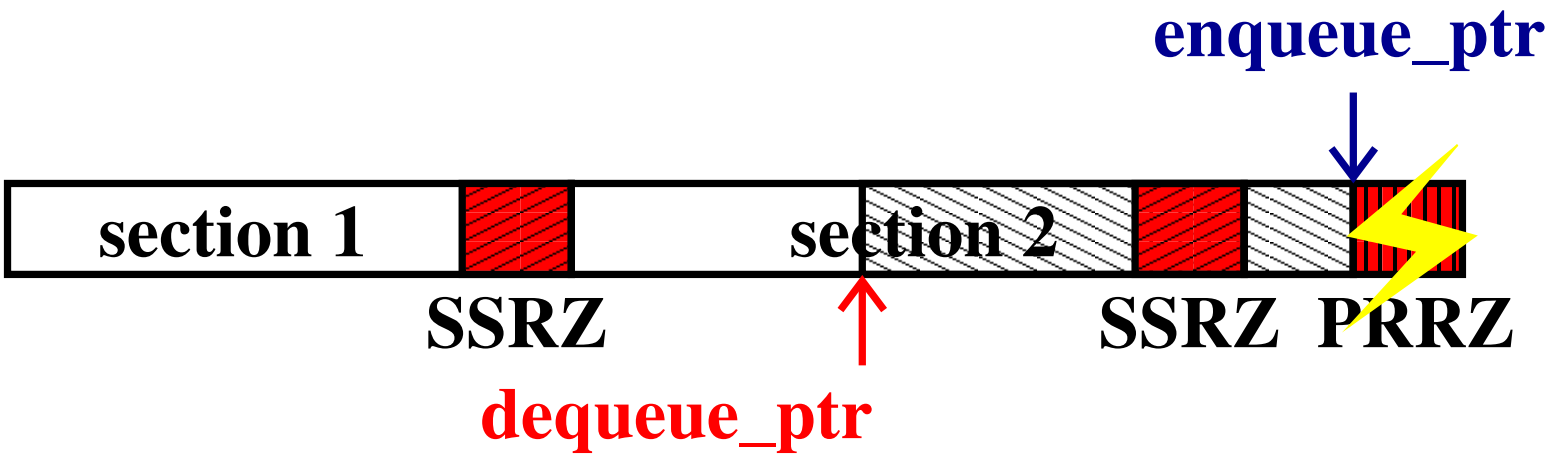
- SSRZ: Section Synchronisation Red-Zone
- PRRZ: Pointer Rotation Red-Zone

Lynx(2): H/W triggered Pointer Rotation



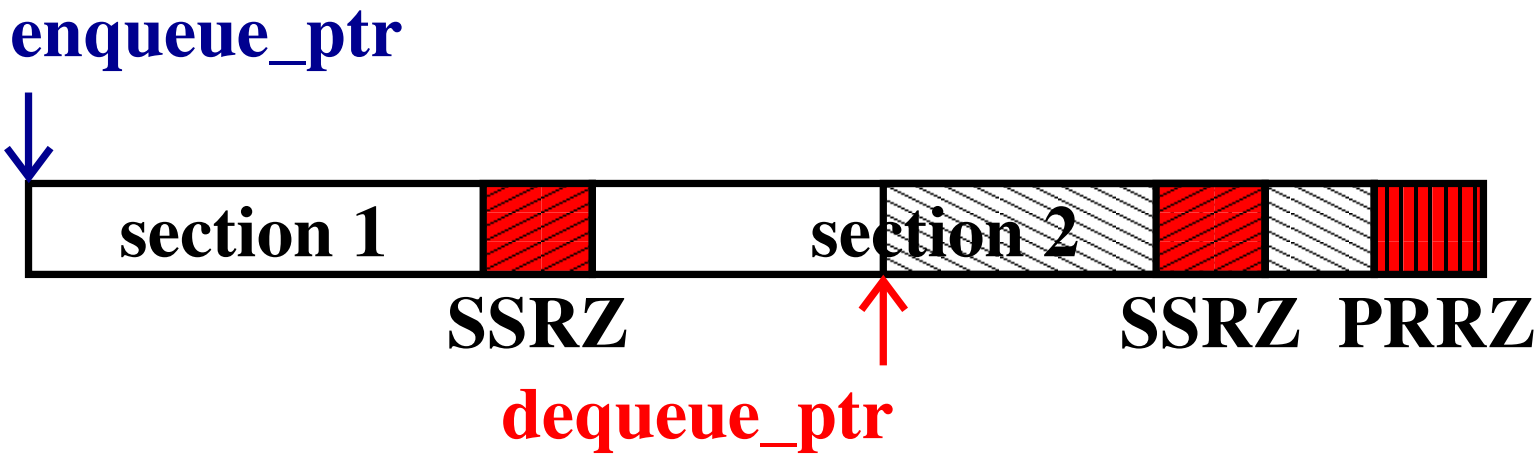
- SSRZ: Section Synchronisation Red-Zone
- PRRZ: Pointer Rotation Red-Zone

Lynx(2): H/W triggered Pointer Rotation



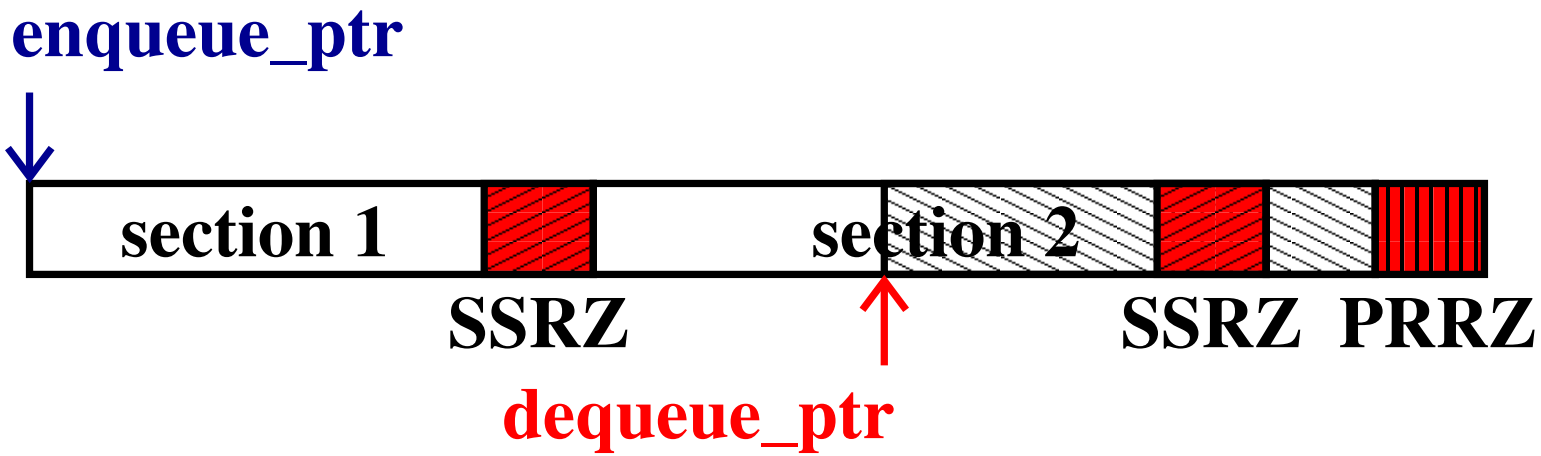
- SSRZ: Section Synchronisation Red-Zone
- PRRZ: Pointer Rotation Red-Zone

Lynx(2): H/W triggered Pointer Rotation



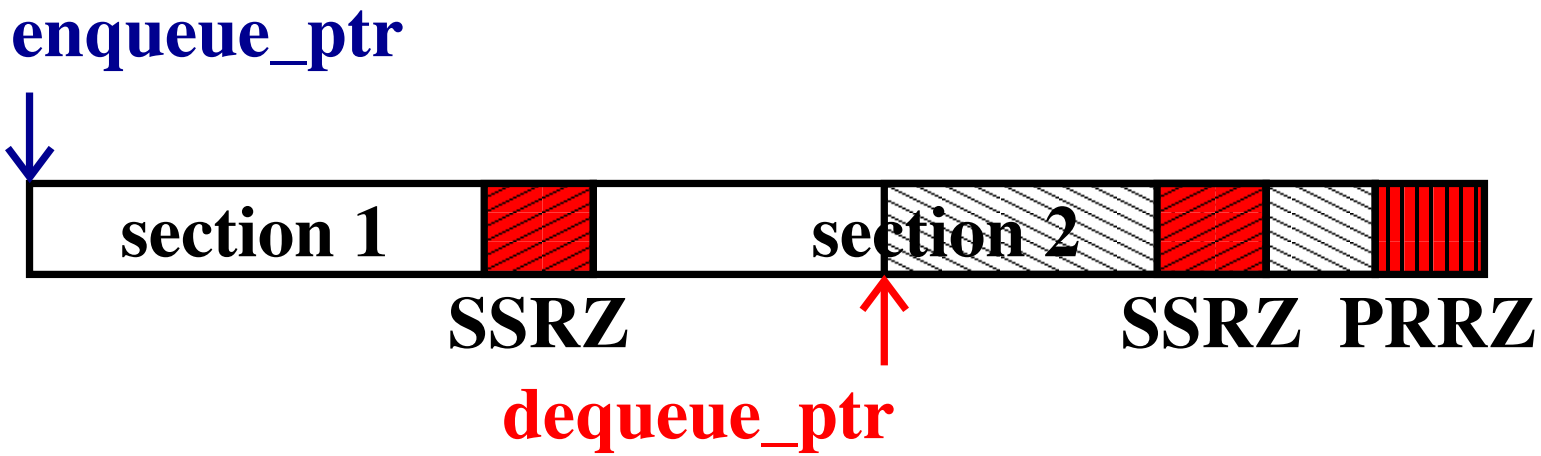
- SSRZ: Section Synchronisation Red-Zone
- PRRZ: Pointer Rotation Red-Zone

Lynx(2): H/W triggered Pointer Rotation



Two types of red-zones:

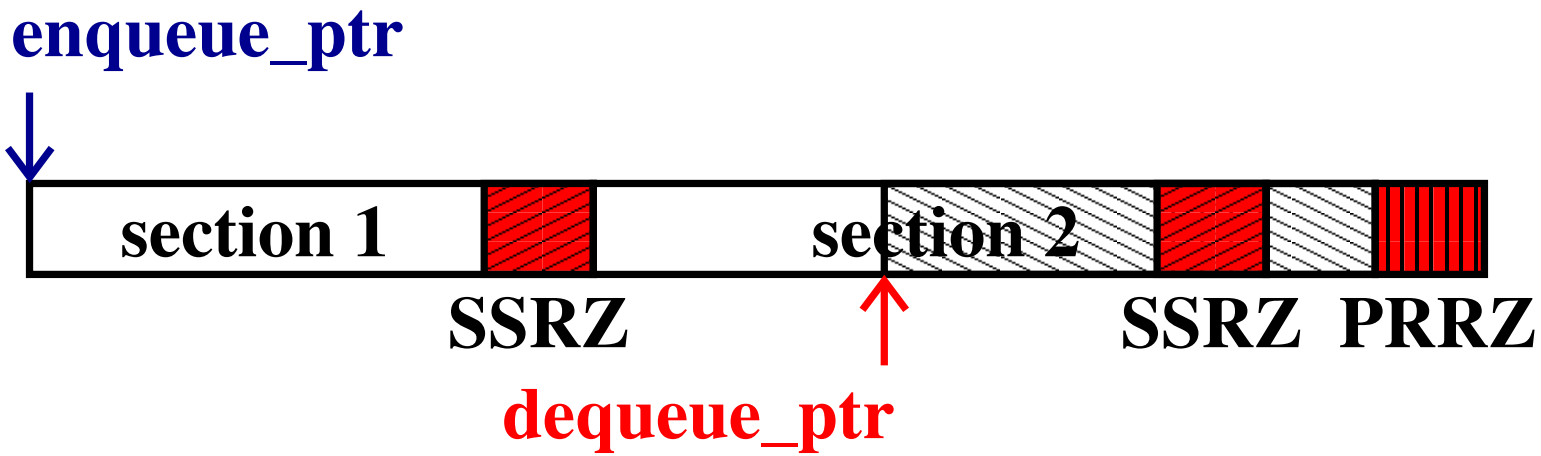
Lynx(2): H/W triggered Pointer Rotation



Two types of red-zones:

1. moving red-zone: SSRZ (Section Synchronisation Red-Zone)

Lynx(2): H/W triggered Pointer Rotation



Two types of red-zones:

1. moving red-zone: SSRZ (Section Synchronisation Red-Zone)
2. fixed red-zone: PRRZ (Pointer Rotation Red-Zone)

Experimental Setup

- Implementation in C++ with inline assembly

Experimental Setup

- Implementation in C++ with inline assembly
- Evaluation on severals machines: from embedded SOCs to server CPUs

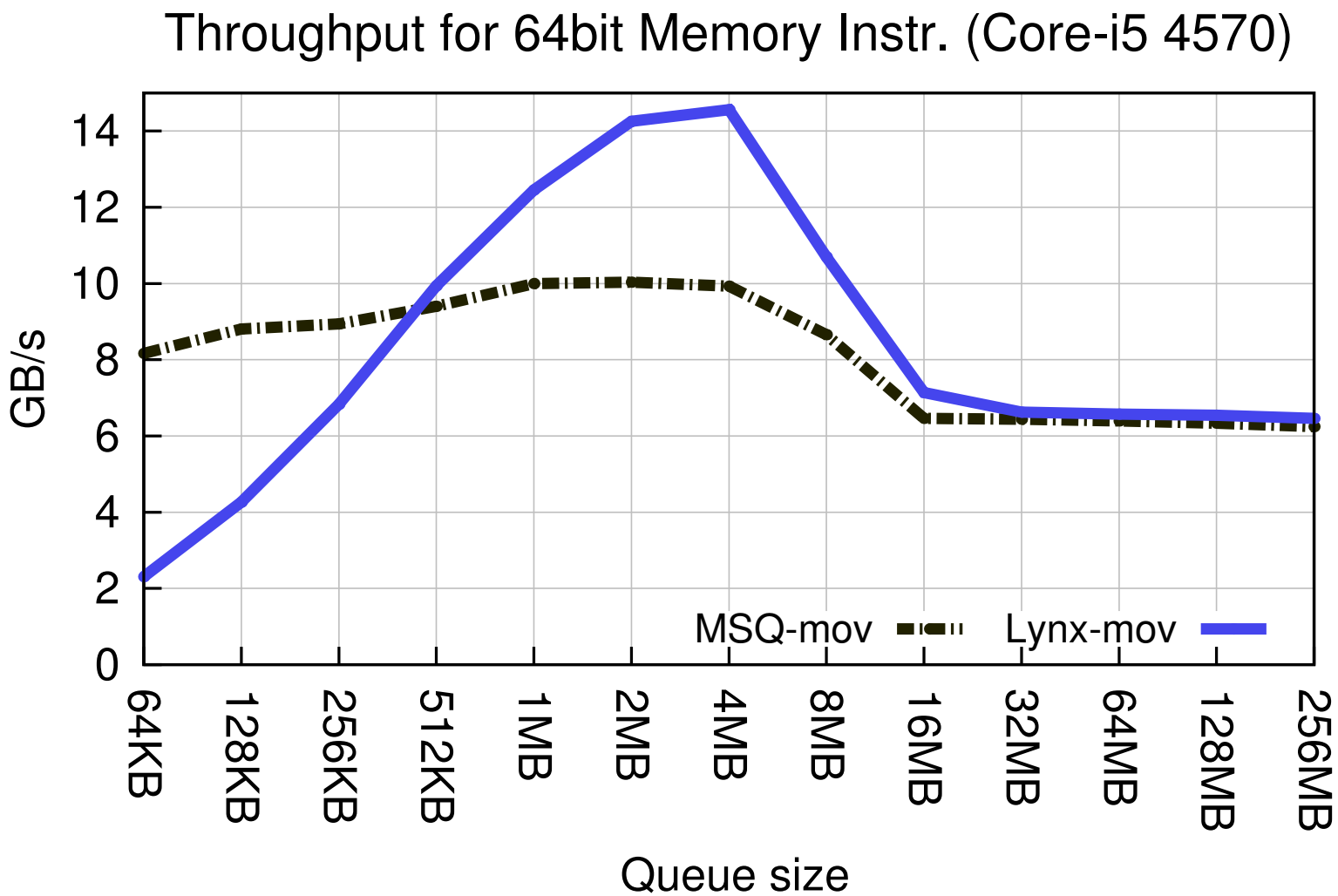
Experimental Setup

- Implementation in C++ with inline assembly
- Evaluation on severals machines: from embedded SOCs to server CPUs
- Throughput experiments for a wide range of queue sizes

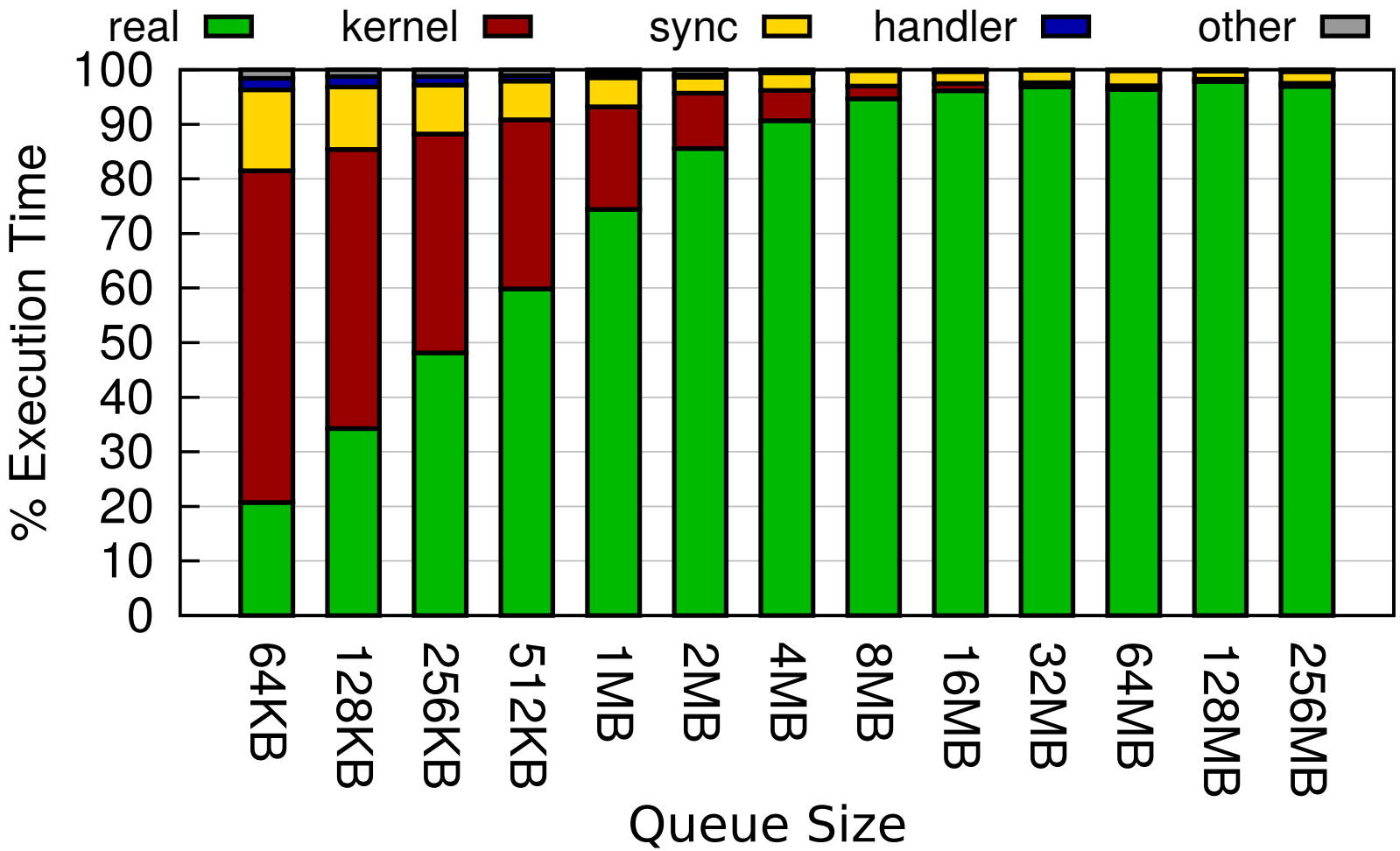
Experimental Setup

- Implementation in C++ with inline assembly
- Evaluation on severals machines: from embedded SOCs to server CPUs
- Throughput experiments for a wide range of queue sizes
- Absolute throughput performance in GB/s

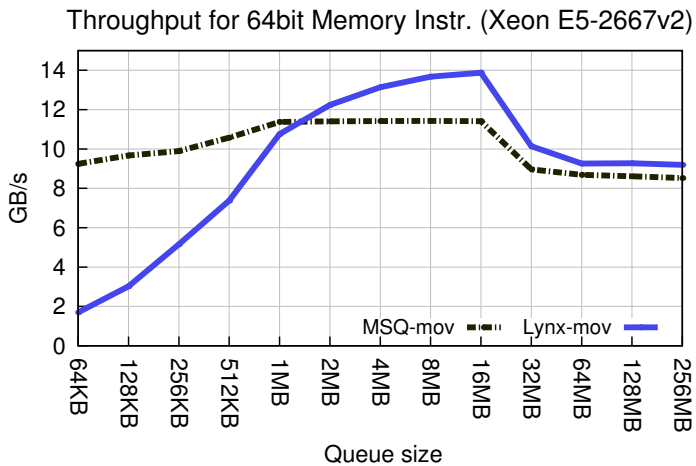
Throughput (GB/s) on Intel core-i5

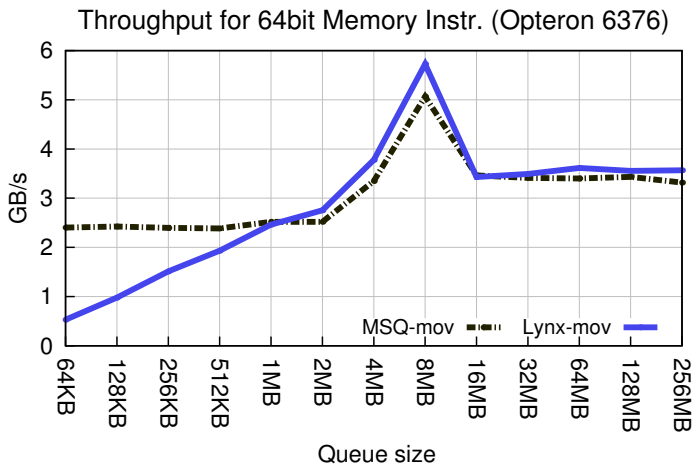


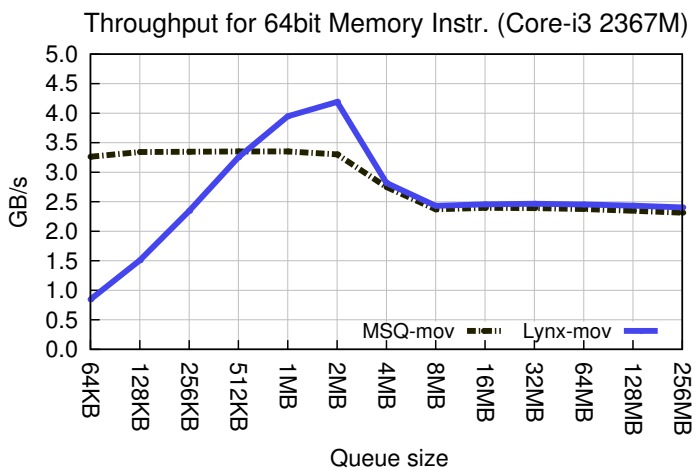
Breakdown of Lynx Overheads

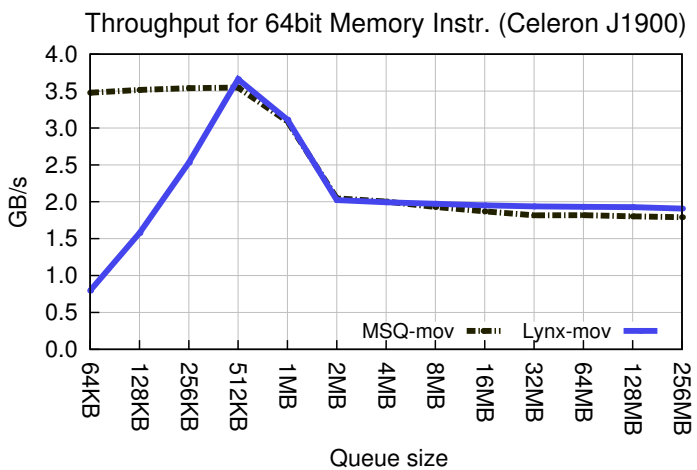


Throughput (GB/s) on Various Machines

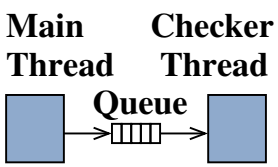


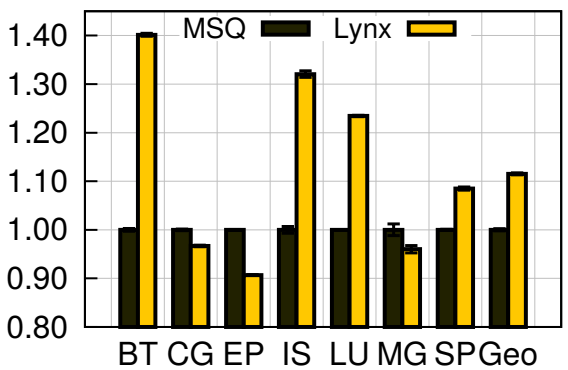




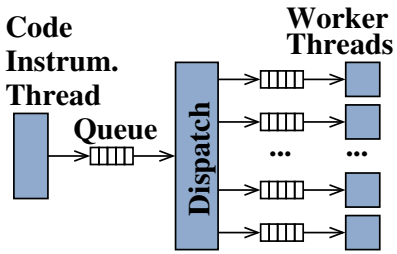


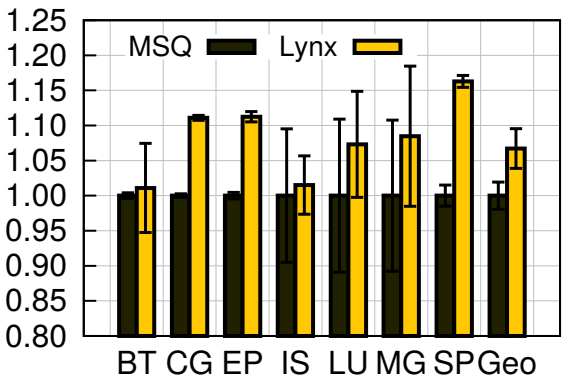
Real World Applications on Intel Xeon



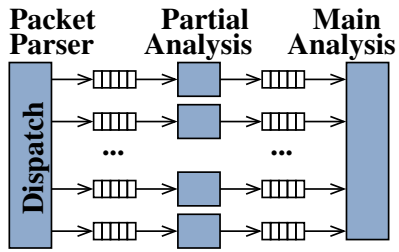


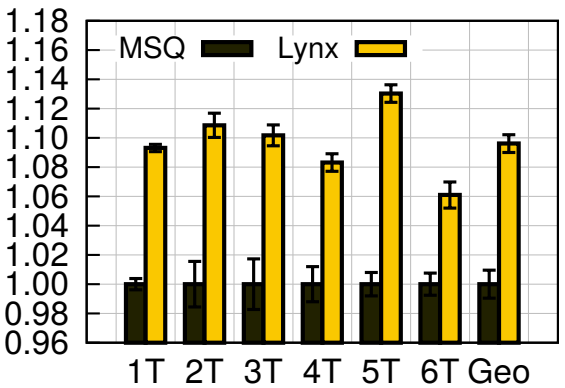
SRMT





SD3





NetworkAnalyser

- The best queue configuration with Lynx is better than the best with MSQ

Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead

Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
- Relies on existing commodity H/W and O/S support for memory protection

Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
- Relies on existing commodity H/W and O/S support for memory protection
- The overhead of synchronisation and boundary checking is moved to the exception handler

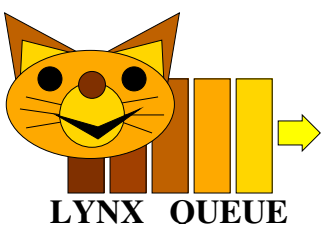
Conclusion

- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
- Relies on existing commodity H/W and O/S support for memory protection
- The overhead of synchronisation and boundary checking is moved to the exception handler
- Throughput increases by up to 57%

Conclusion

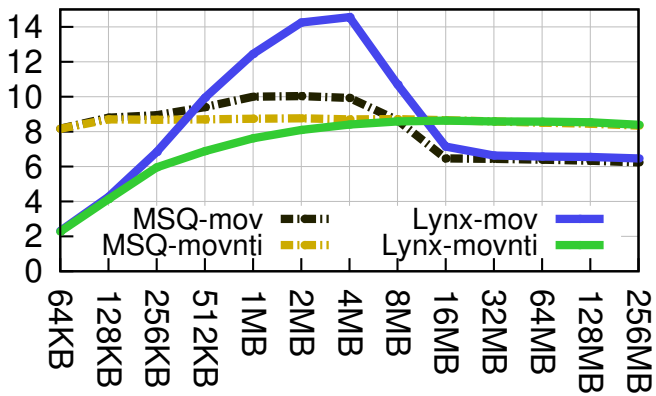
- Proposed Lynx: a lock-free SP/SC software queue with just 2 instructions overhead
- Relies on existing commodity H/W and O/S support for memory protection
- The overhead of synchronisation and boundary checking is moved to the exception handler
- Throughput increases by up to 57%

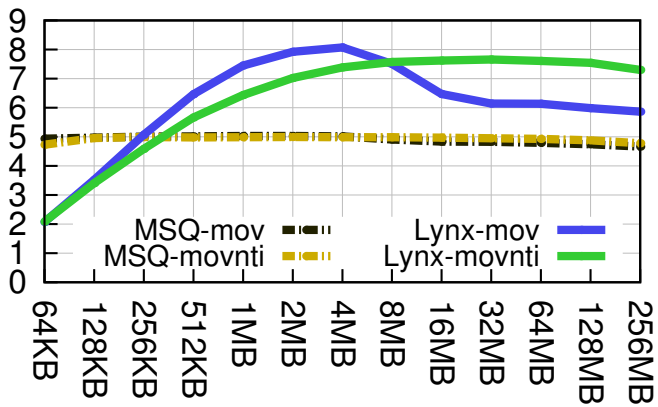
`https://www.cl.cam.ac.uk/~km647/papers/lynx/lynxQ.tar.bz2`

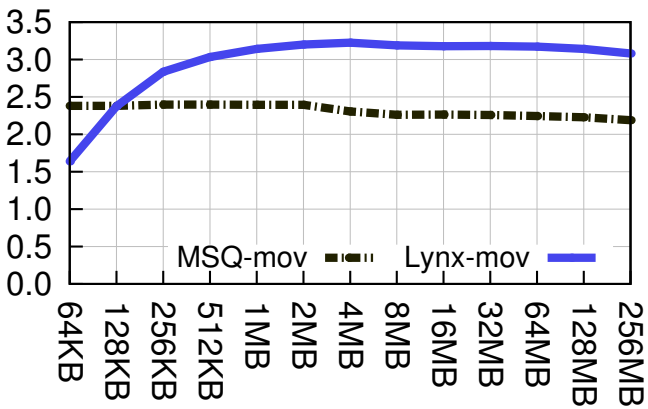


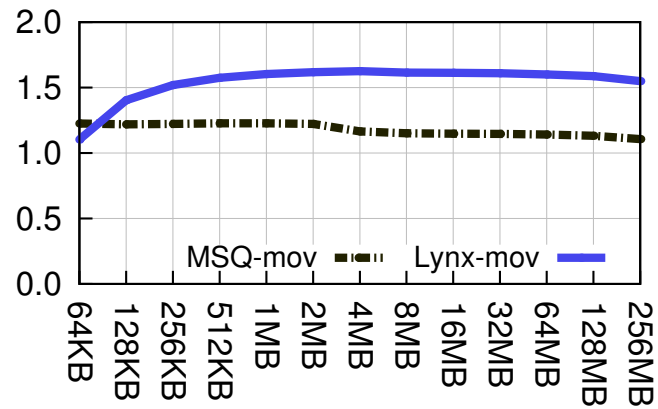
Back-up slides

Throughput (GB/s) on Intel core-i5

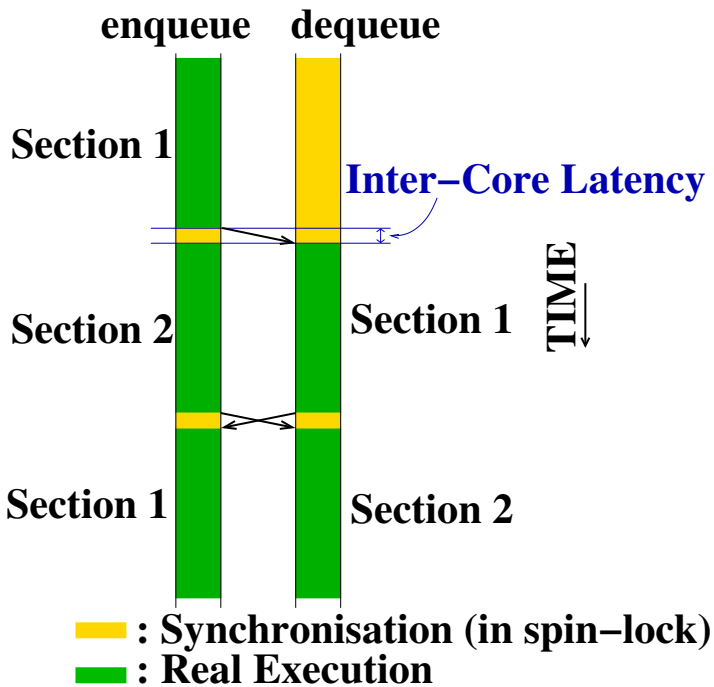


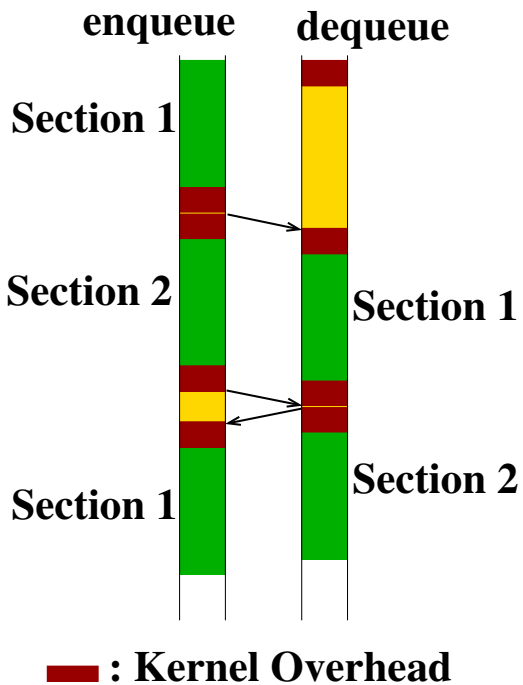




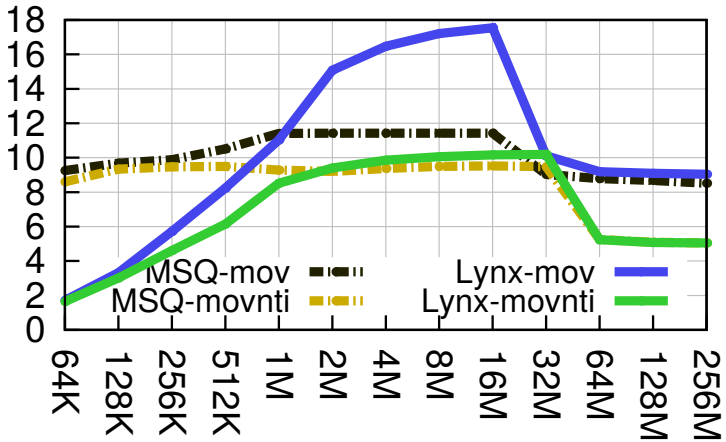


Breakdown of Synchronisation Overheads

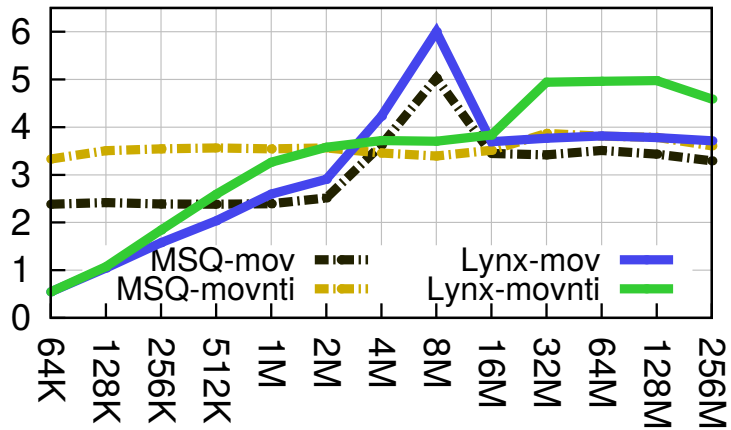




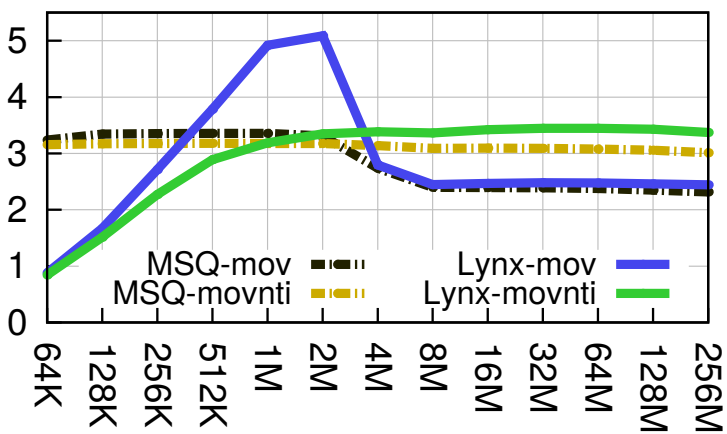
Throughput (GB/s) on Different Machines



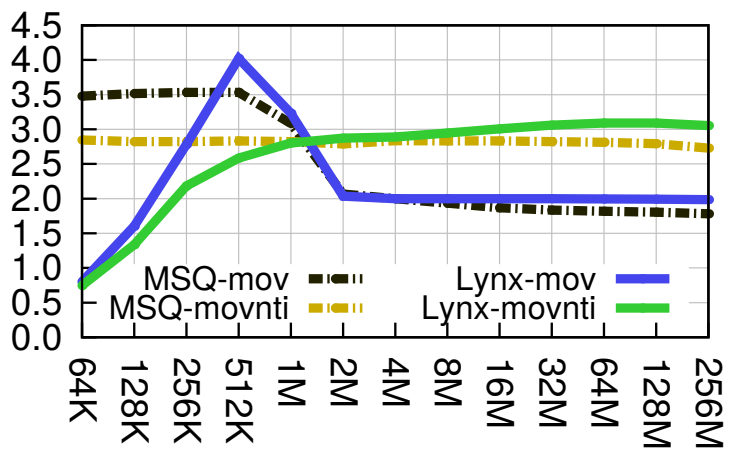
Intel Xeon E5-2667 v2



AMD Opteron 6376



Intel Core-i3 2367M



Intel Celeron J1900