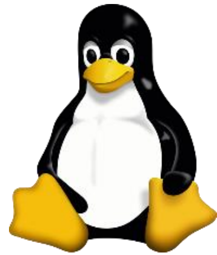
CLoF:
A Compositional Lock Framework for Multi-level NUMA Systems

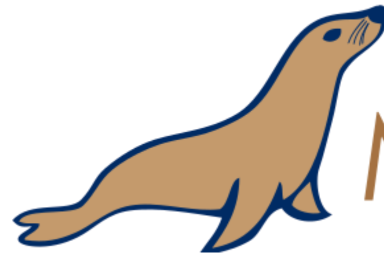
Rafael Chehab Antonio Paolillo Diogo Behrens Ming Fu Hermann Härtig Haibo Chen

March 18th, 2022

Concurrency is Everywhere

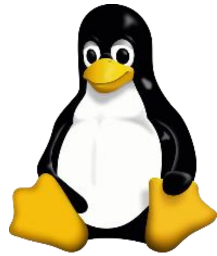
Modern operating systems, databases & applications resort to **multi-core concurrency** to achieve high performance.

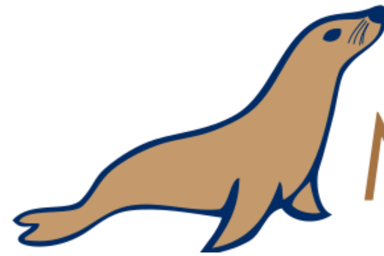




Concurrency is Everywhere

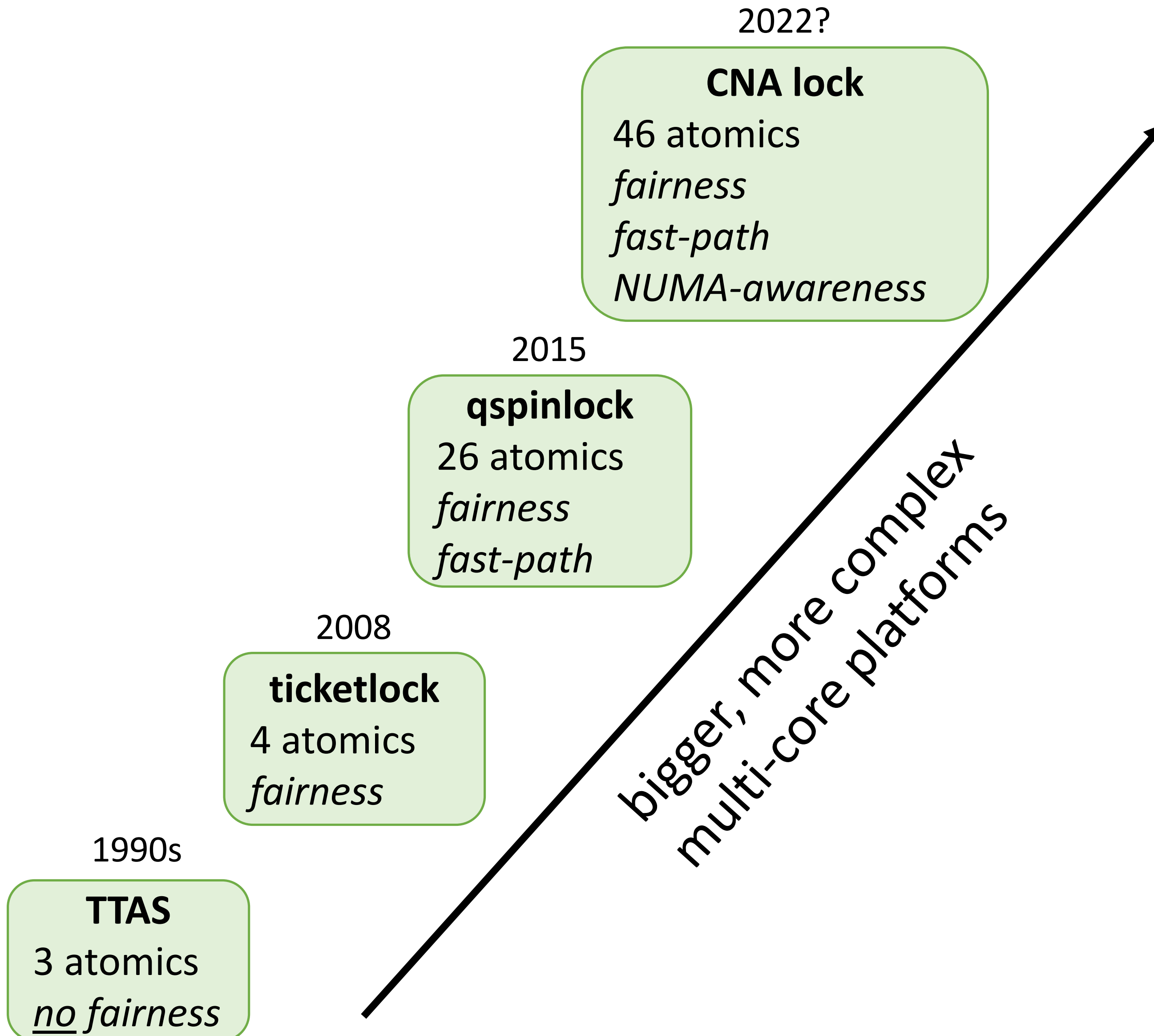
Modern operating systems, databases & applications resort to **multi-core concurrency** to achieve high performance.



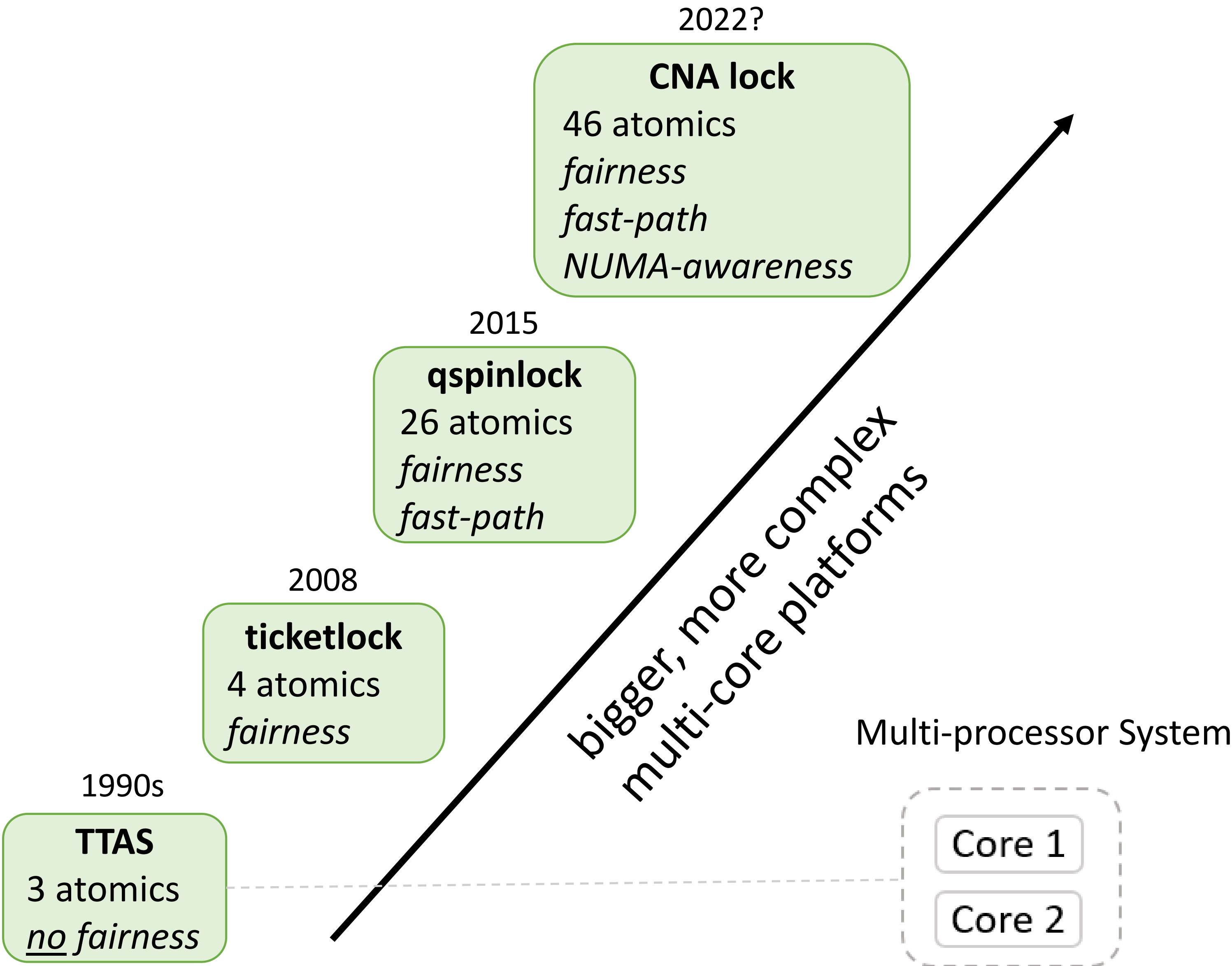


Multi-core concurrency:
One of its most important tasks is to *synchronize* access to shared variables

Linux spinlock evolution



Linux spinlock evolution



Linux spinlock evolution

2022?

CNA lock
46 atomics
fairness
fast-path
NUMA-awareness

2015

qspinlock
26 atomics
fairness
fast-path

2008

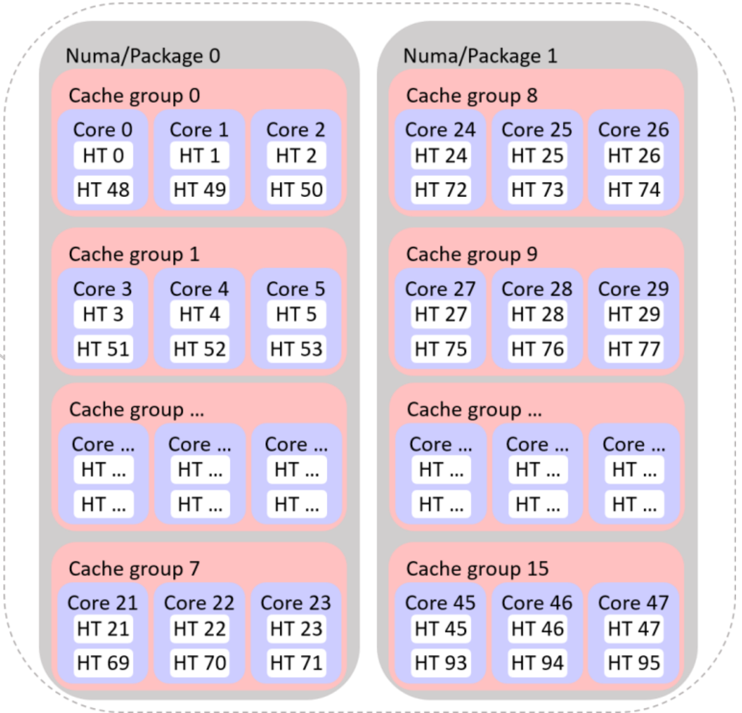
ticketlock
4 atomics
fairness

1990s

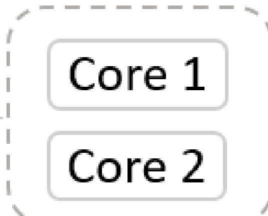
TTAS
3 atomics
no fairness

bigger, more complex
multi-core platforms

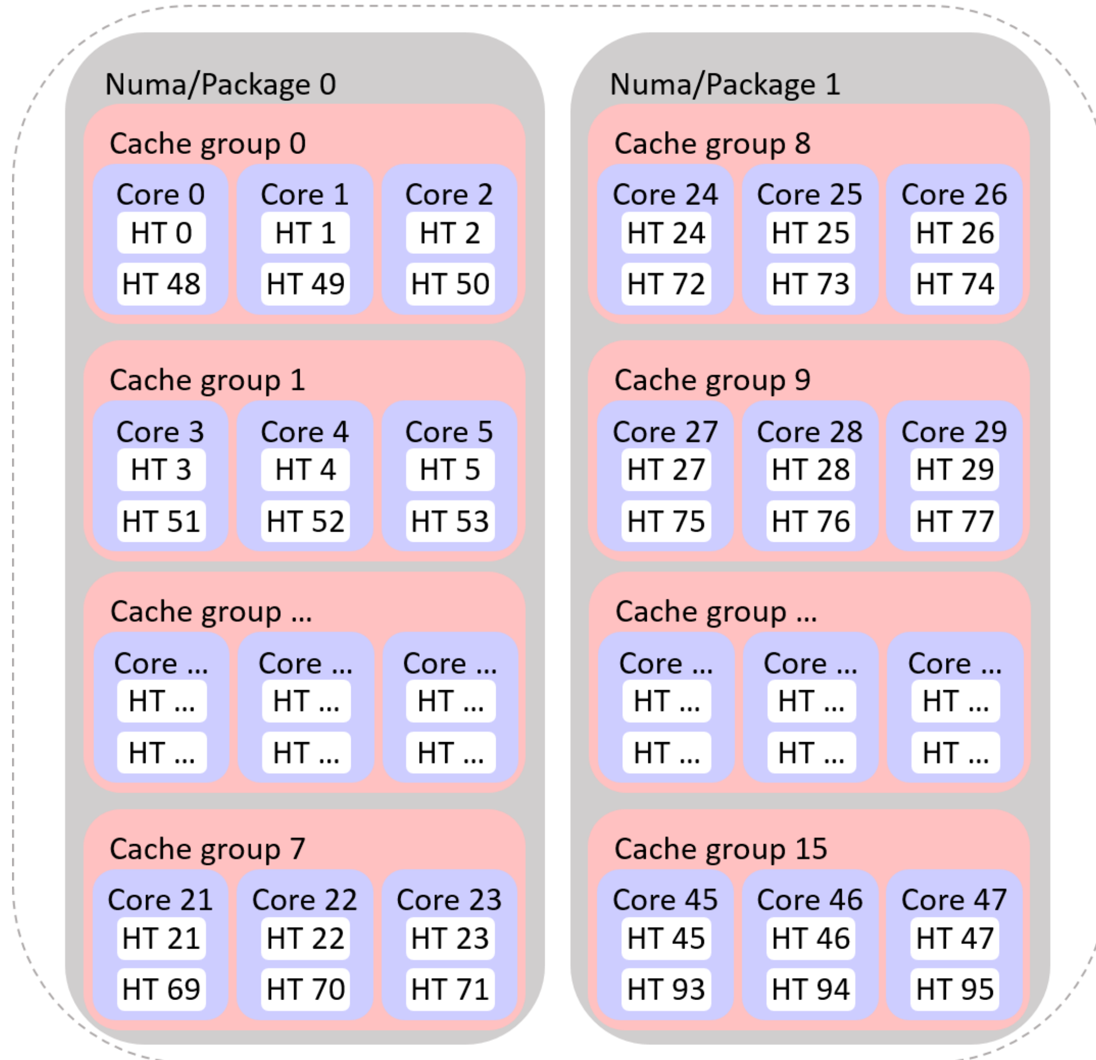
Many-core NUMA system



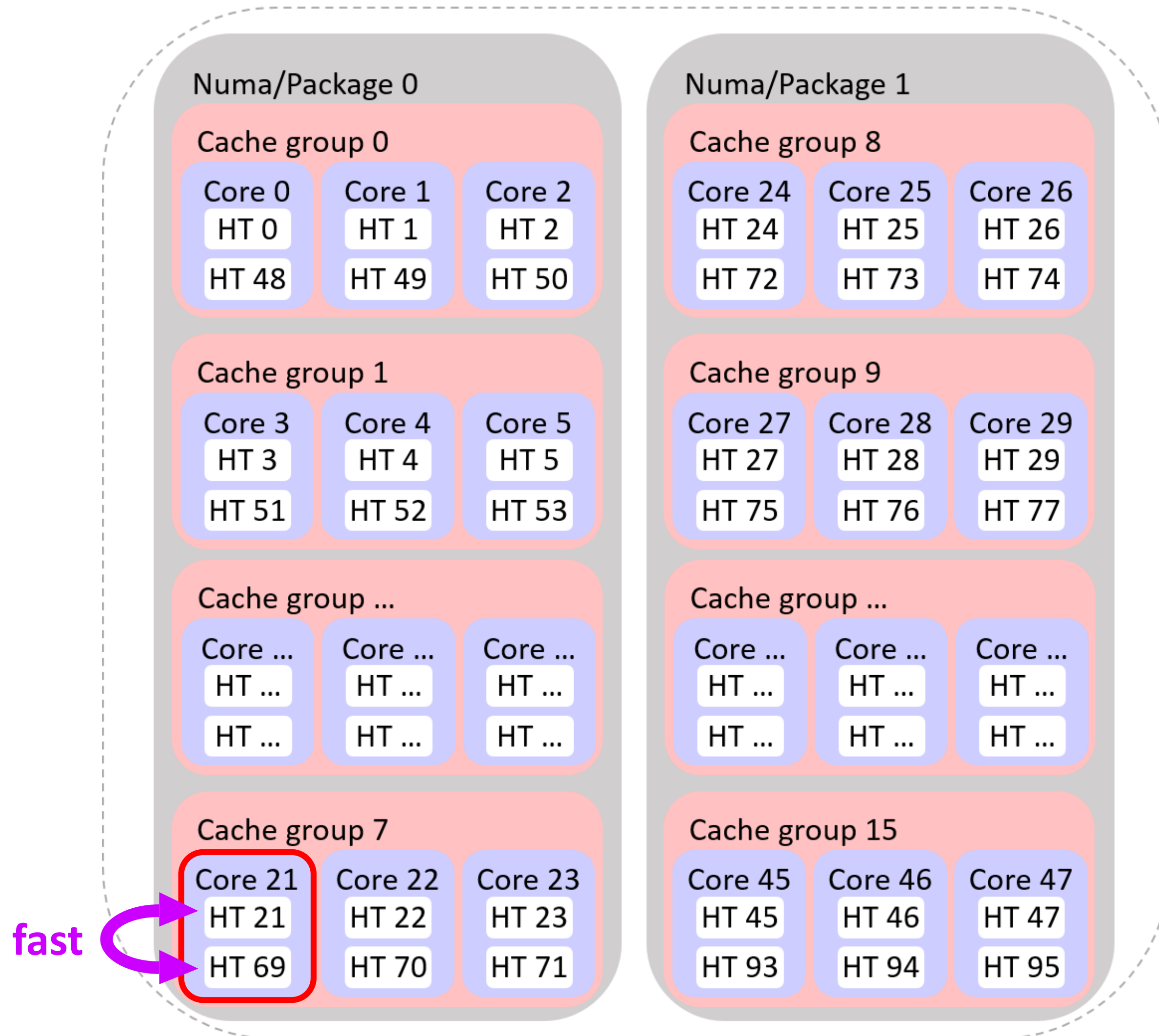
Multi-processor System



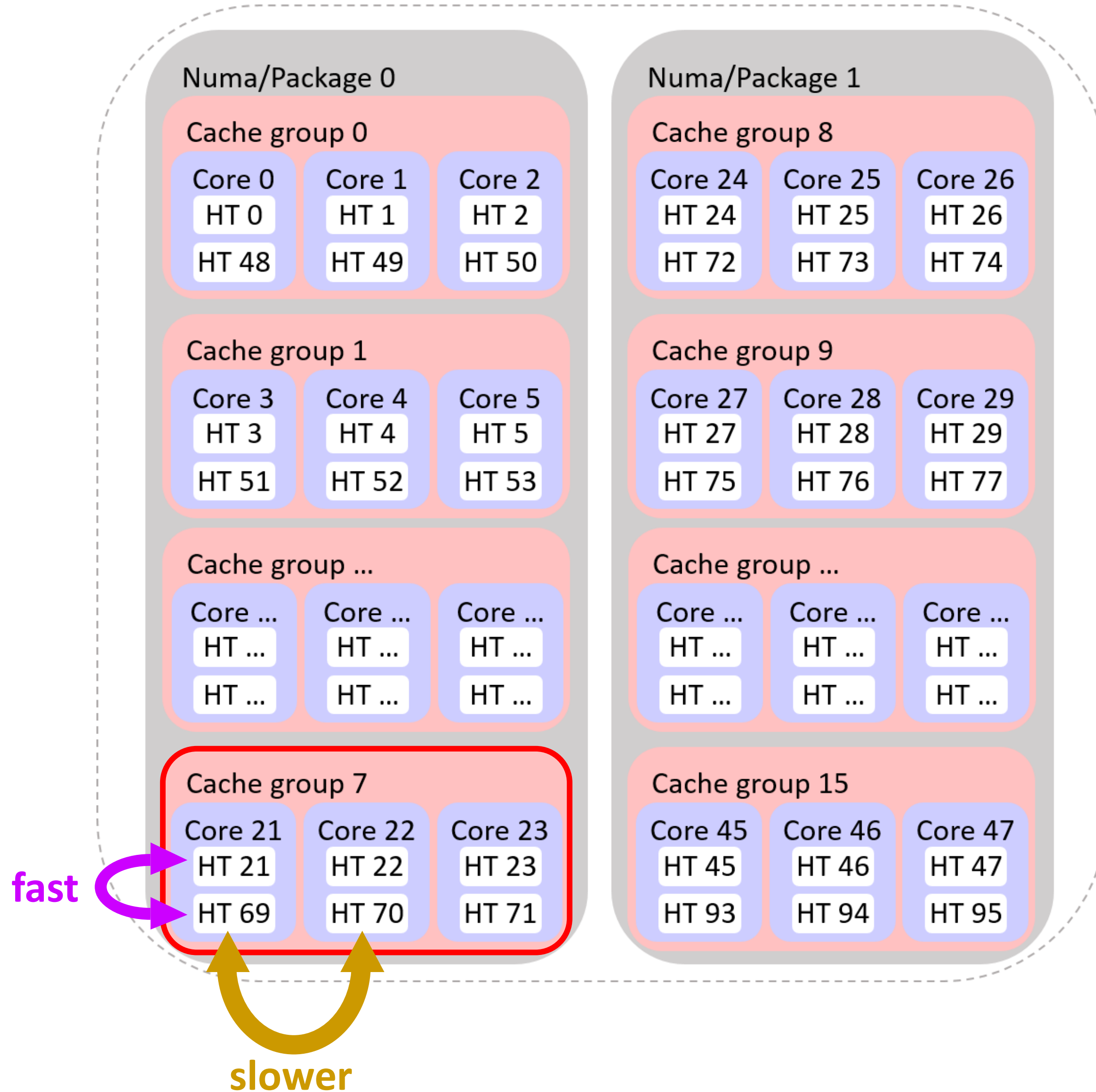
Deep-hierarchy NUMA systems



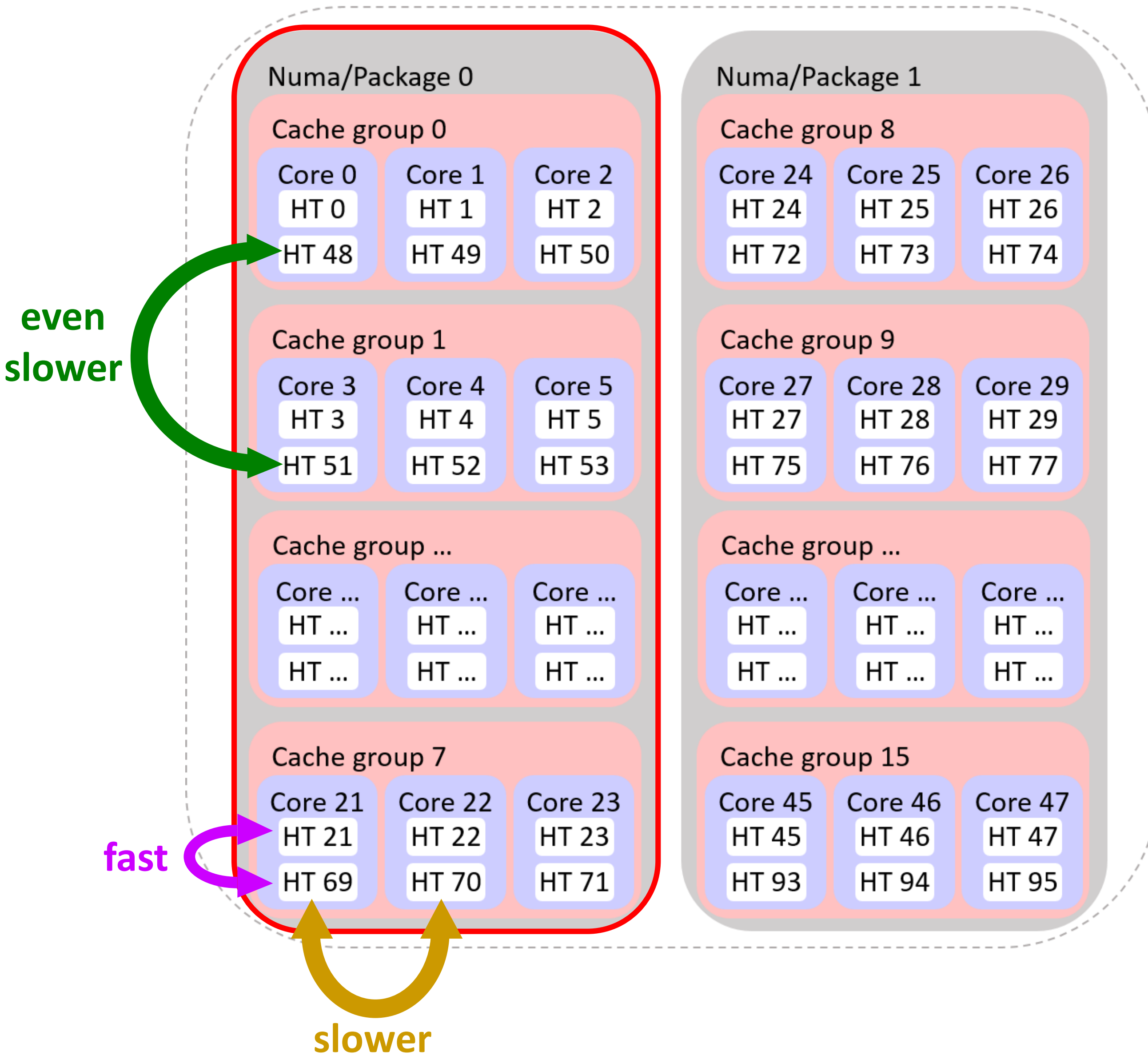
Deep-hierarchy NUMA systems



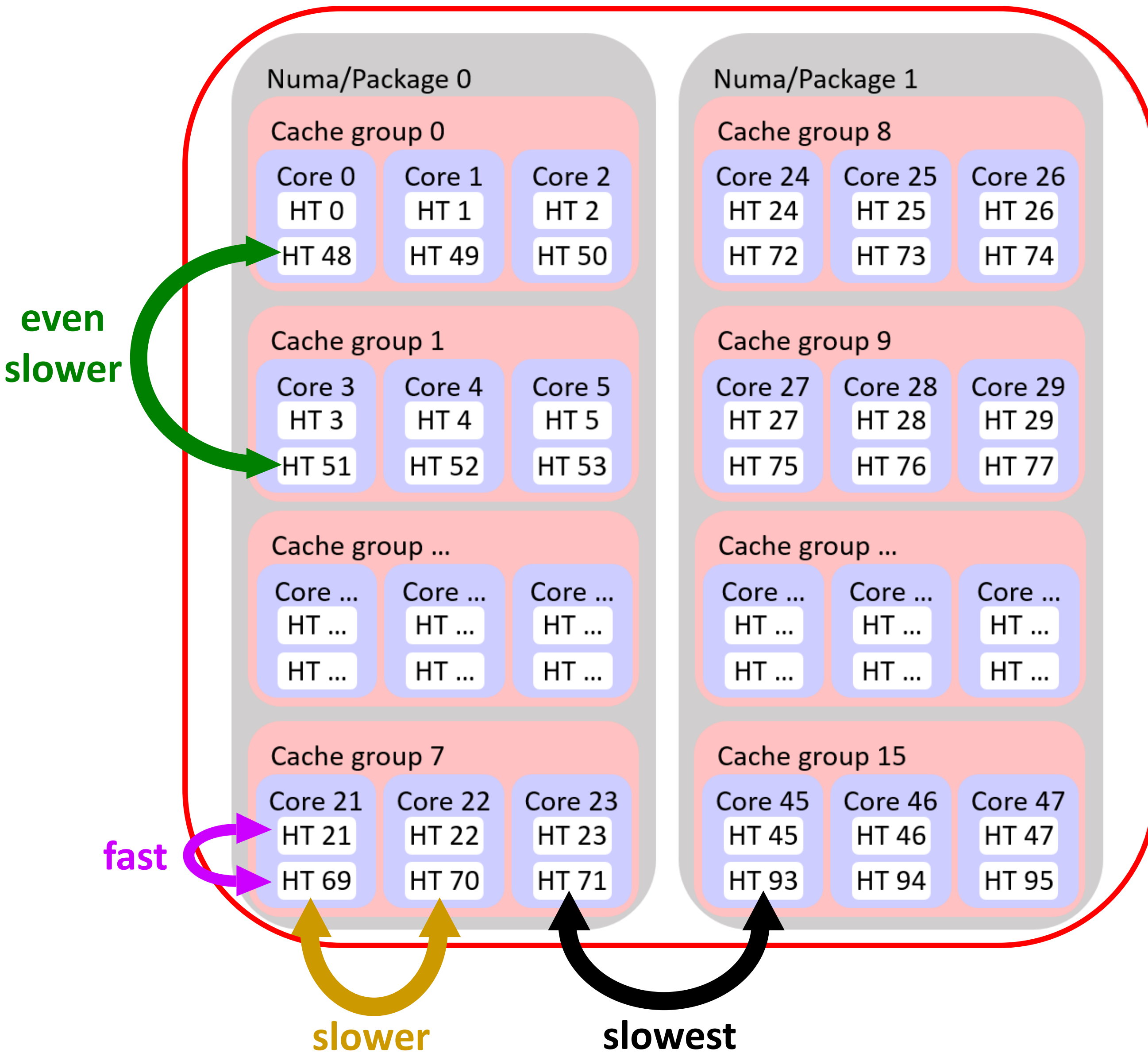
Deep-hierarchy NUMA systems



Deep-hierarchy NUMA systems



Deep-hierarchy NUMA systems



Deep-hierarchy NUMA systems

Many-core NUMA system

Numa/Package 0

Cache group 0
Core 0 HT 0 HT 48
Core 1 HT 1 HT 49
Core 2 HT 2 HT 50

Cache group 1
Core 3 HT 3 HT 51
Core 4 HT 4 HT 52
Core 5 HT 5 HT 53

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 7
Core 21 HT 21 HT 69
Core 22 HT 22 HT 70
Core 23 HT 23 HT 71

Numa/Package 1

Cache group 8
Core 24 HT 24 HT 72
Core 25 HT 25 HT 73
Core 26 HT 26 HT 74

Cache group 9
Core 27 HT 27 HT 75
Core 28 HT 28 HT 76
Core 29 HT 29 HT 77

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 15
Core 45 HT 45 HT 93
Core 46 HT 46 HT 94
Core 47 HT 47 HT 95

even slower

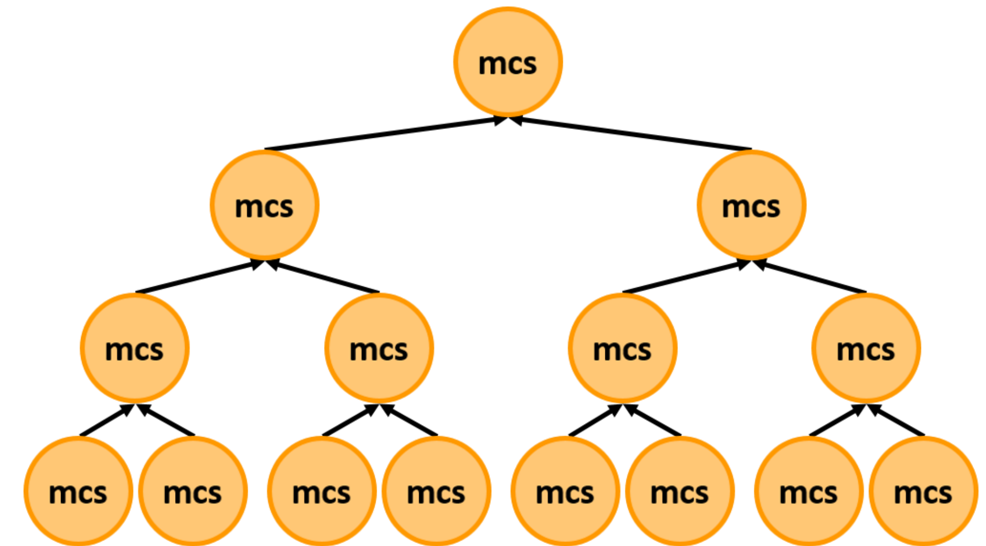
fast

slower

slowest

Multi-level NUMA-aware Locks

- HMCS creates a hierarchy of MCS locks
  - Arbitrary number of levels



Deep-hierarchy NUMA systems

Many-core NUMA system

Numa/Package 0

Cache group 0
Core 0 HT 0 HT 48
Core 1 HT 1 HT 49
Core 2 HT 2 HT 50

Cache group 1
Core 3 HT 3 HT 51
Core 4 HT 4 HT 52
Core 5 HT 5 HT 53

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 7
Core 21 HT 21 HT 69
Core 22 HT 22 HT 70
Core 23 HT 23 HT 71

Numa/Package 1

Cache group 8
Core 24 HT 24 HT 72
Core 25 HT 25 HT 73
Core 26 HT 26 HT 74

Cache group 9
Core 27 HT 27 HT 75
Core 28 HT 28 HT 76
Core 29 HT 29 HT 77

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 15
Core 45 HT 45 HT 93
Core 46 HT 46 HT 94
Core 47 HT 47 HT 95

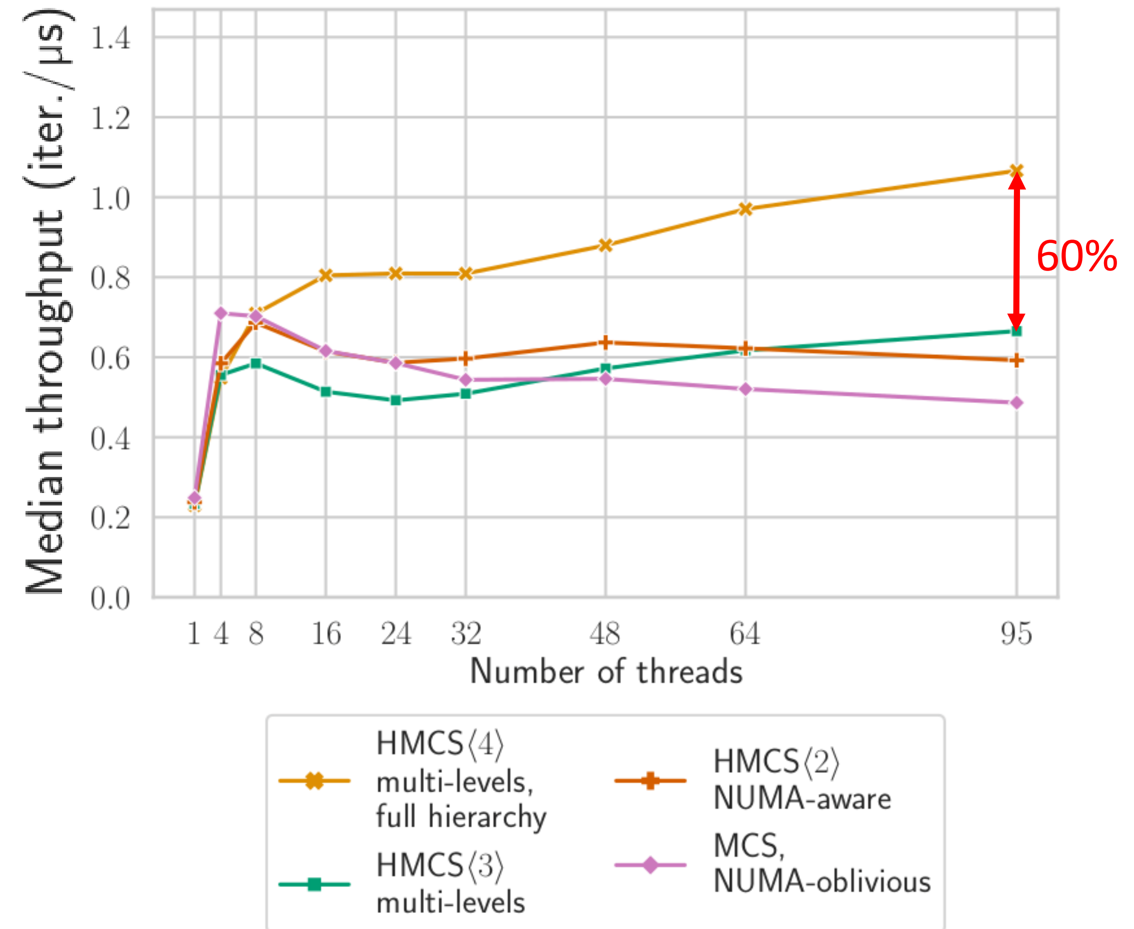
even slower

fast

slower

slowest

Multi-level NUMA-aware Locks



Deep-hierarchy NUMA systems

Many-core NUMA system

Numa/Package 0

Cache group 0
Core 0 HT 0 HT 48
Core 1 HT 1 HT 49
Core 2 HT 2 HT 50

Cache group 1
Core 3 HT 3 HT 51
Core 4 HT 4 HT 52
Core 5 HT 5 HT 53

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 7
Core 21 HT 21 HT 69
Core 22 HT 22 HT 70
Core 23 HT 23 HT 71

Numa/Package 1

Cache group 8
Core 24 HT 24 HT 72
Core 25 HT 25 HT 73
Core 26 HT 26 HT 74

Cache group 9
Core 27 HT 27 HT 75
Core 28 HT 28 HT 76
Core 29 HT 29 HT 77

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 15
Core 45 HT 45 HT 93
Core 46 HT 46 HT 94
Core 47 HT 47 HT 95

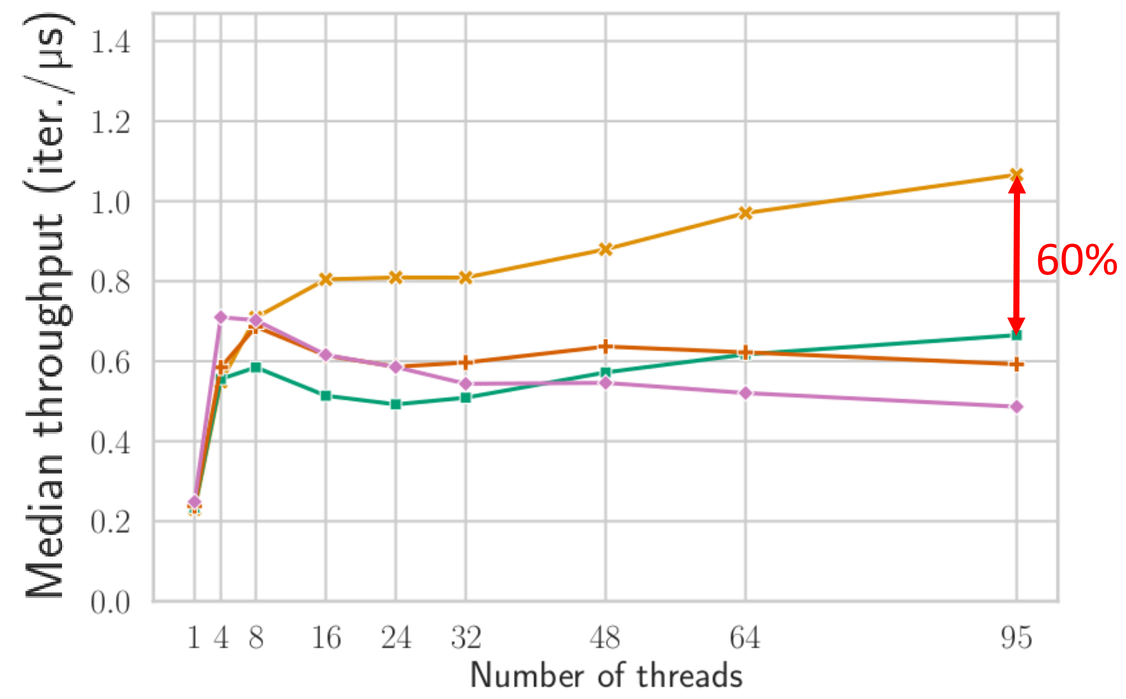
even slower

fast

slower

slowest

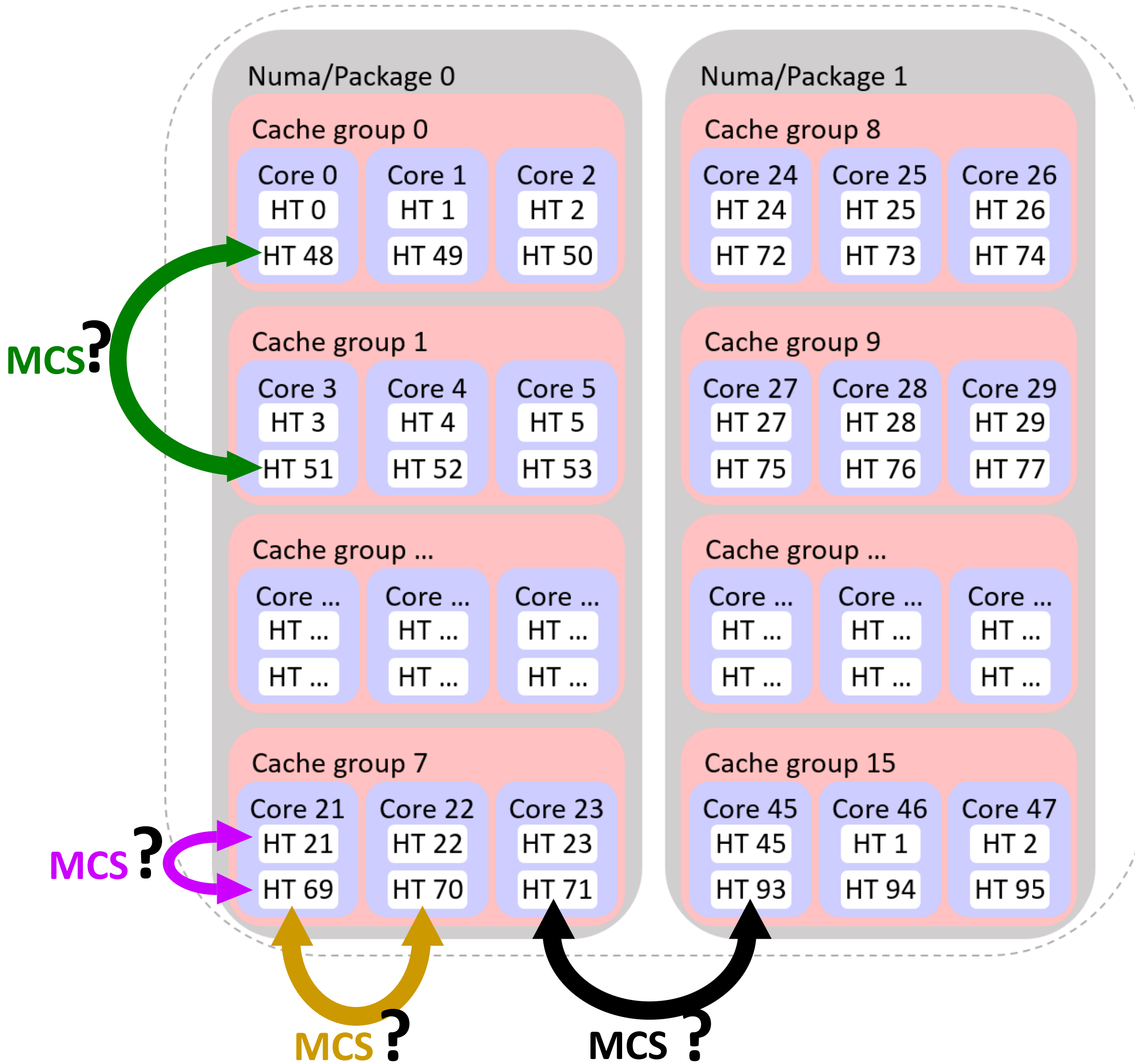
Multi-level NUMA-aware Locks



Multi-Level:

Utilizing the full deep-hierarchy in a lock improves performance

Level-heterogeneity



Level-heterogeneity

Experiment - Heterogeneity

Many-core NUMA system

Numa/Package 0

Cache group 0
Core 0 HT 0 HT 48
Core 1 HT 1 HT 49
Core 2 HT 2 HT 50

Cache group 1
Core 3 HT 3 HT 51
Core 4 HT 4 HT 52
Core 5 HT 5 HT 53

Cache group …
Core … HT … HT …
Core … HT … HT …
Core … HT … HT …

Cache group 7
Core 21 HT 21 HT 69
Core 22 HT 22 HT 70
Core 23 HT 23 HT 71

Numa/Package 1

Cache group 8
Core 24 HT 24 HT 72
Core 25 HT 25 HT 73
Core 26 HT 26 HT 74

Cache group 9
Core 27 HT 27 HT 75
Core 28 HT 28 HT 76
Core 29 HT 29 HT 77

Cache group …
Core … HT … HT …
Core … HT … HT …
Core … HT … HT …

Cache group 15
Core 45 HT 45 HT 93
Core 46 HT 1 HT 94
Core 47 HT 2 HT 95

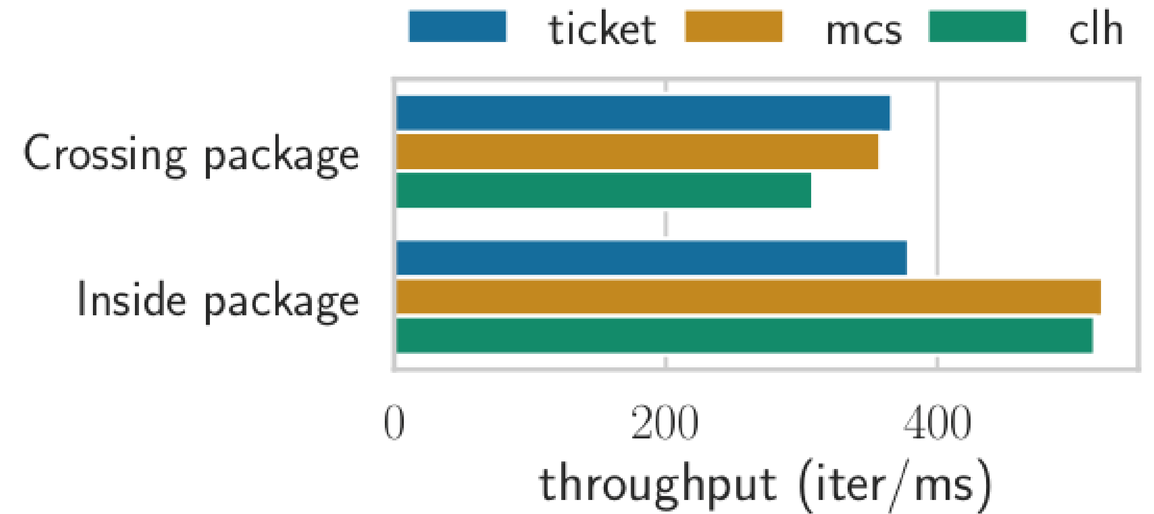
MCS?

MCS?

MCS?

MCS ?

x86 server – execution of classical locks in isolation



Level-heterogeneity

Many-core NUMA system

Numa/Package 0

Cache group 0
Core 0 HT 0 HT 48
Core 1 HT 1 HT 49
Core 2 HT 2 HT 50

Cache group 1
Core 3 HT 3 HT 51
Core 4 HT 4 HT 52
Core 5 HT 5 HT 53

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 7
Core 21 HT 21 HT 69
Core 22 HT 22 HT 70
Core 23 HT 23 HT 71

Numa/Package 1

Cache group 8
Core 24 HT 24 HT 72
Core 25 HT 25 HT 73
Core 26 HT 26 HT 74

Cache group 9
Core 27 HT 27 HT 75
Core 28 HT 28 HT 76
Core 29 HT 29 HT 77

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 15
Core 45 HT 45 HT 93
Core 46 HT 1 HT 94
Core 47 HT 2 HT 95

MCS?

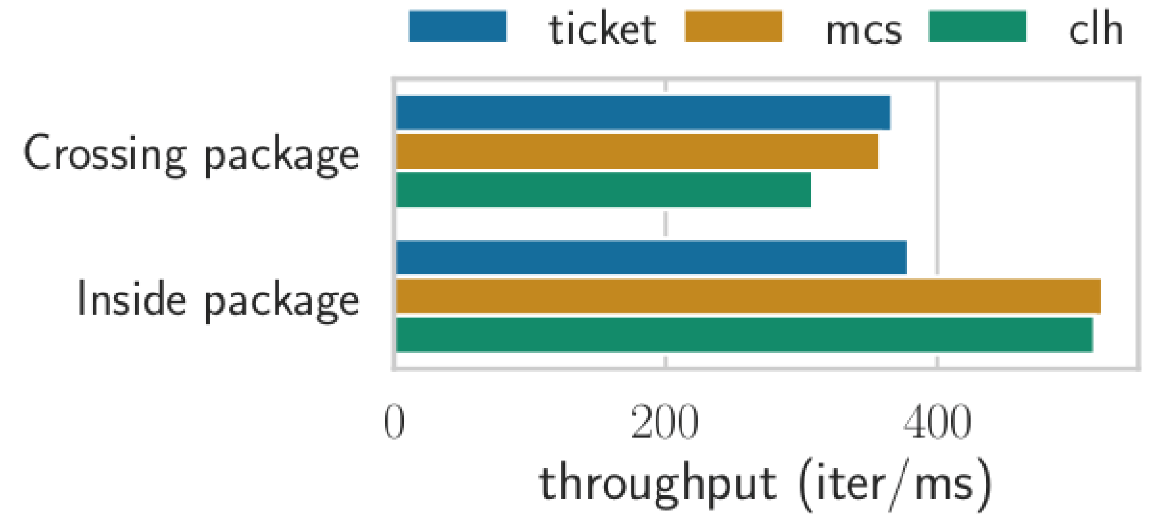
MCS?

MCS?

MCS ?

Experiment - Heterogeneity

x86 server – execution of classical locks in isolation



Level-heterogeneity

Many-core NUMA system

Numa/Package 0

Cache group 0: Core 0 (HT 0, HT 48), Core 1 (HT 1, HT 49), Core 2 (HT 2, HT 50)

Cache group 1: Core 3 (HT 3, HT 51), Core 4 (HT 4, HT 52), Core 5 (HT 5, HT 53)

Cache group ...: Core ... (HT ..., HT ...), Core ... (HT ..., HT ...), Core ... (HT ..., HT ...)

Cache group 7: Core 21 (HT 21, HT 69), Core 22 (HT 22, HT 70), Core 23 (HT 23, HT 71)

Numa/Package 1

Cache group 8: Core 24 (HT 24, HT 72), Core 25 (HT 25, HT 73), Core 26 (HT 26, HT 74)

Cache group 9: Core 27 (HT 27, HT 75), Core 28 (HT 28, HT 76), Core 29 (HT 29, HT 77)

Cache group ...: Core ... (HT ..., HT ...), Core ... (HT ..., HT ...), Core ... (HT ..., HT ...)

Cache group 15: Core 45 (HT 45, HT 93), Core 46 (HT 1, HT 94), Core 47 (HT 2, HT 95)

MCS?

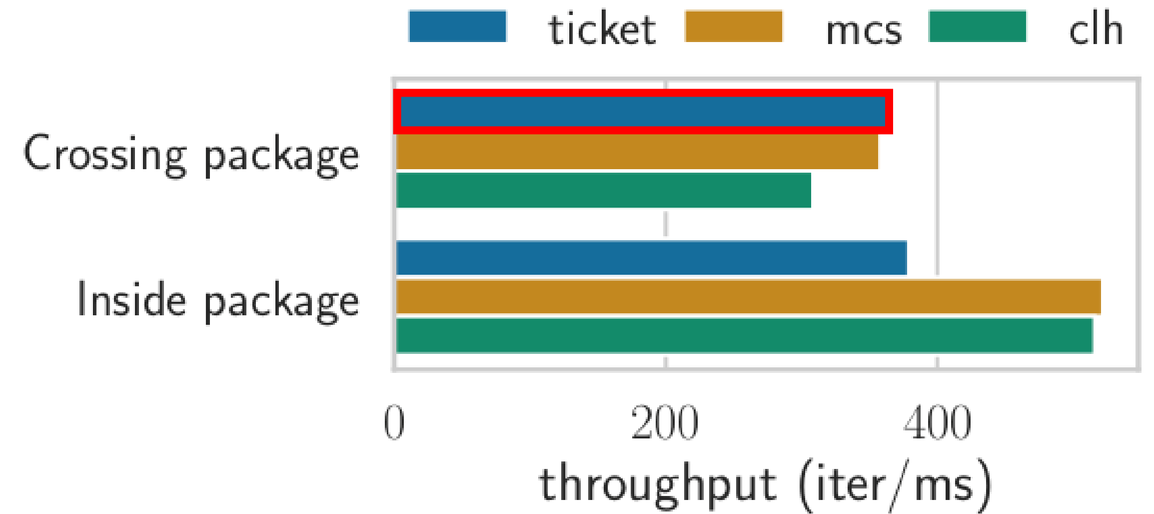
MCS?

MCS?

MCS ticketlock

Experiment - Heterogeneity

x86 server – execution of classical locks in isolation



Level-heterogeneity

Many-core NUMA system

Numa/Package 0

Cache group 0
Core 0 HT 0 HT 48
Core 1 HT 1 HT 49
Core 2 HT 2 **HT 50**

Cache group 1
Core 3 HT 3 HT 51
Core 4 HT 4 HT 52
Core 5 HT 5 **HT 53**

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... **HT ...**

Cache group 7
Core 21 HT 21 HT 69
Core 22 HT 22 HT 70
Core 23 HT 23 **HT 71**

Numa/Package 1

Cache group 8
Core 24 HT 24 HT 72
Core 25 HT 25 HT 73
Core 26 HT 26 HT 74

Cache group 9
Core 27 HT 27 HT 75
Core 28 HT 28 HT 76
Core 29 HT 29 HT 77

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 15
Core 45 HT 45 HT 93
Core 46 HT 1 HT 94
Core 47 HT 2 HT 95

MCS?

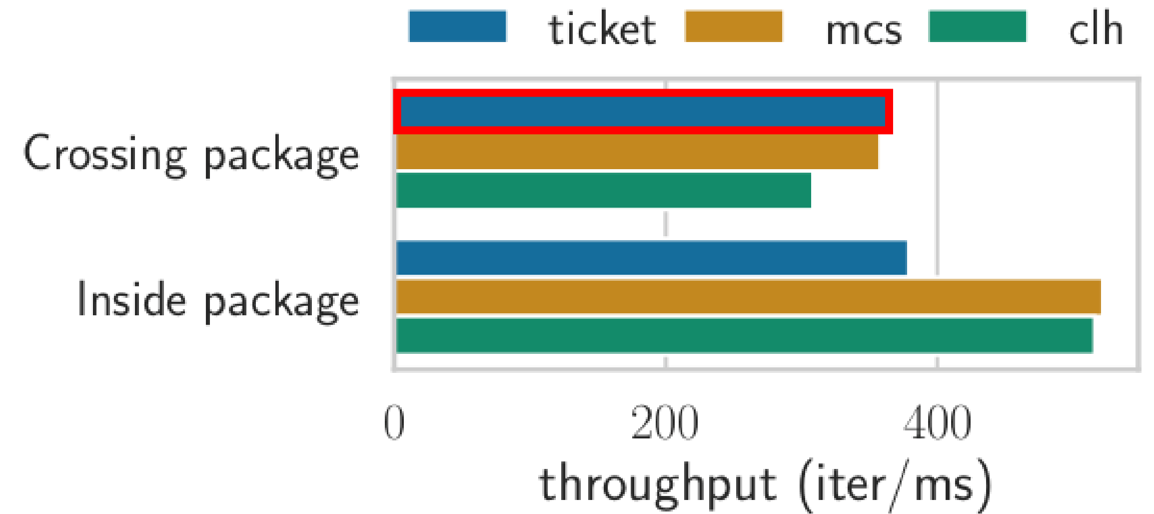
MCS?

MCS?

~~MCS~~ ticketlock

Experiment - Heterogeneity

x86 server – execution of classical locks in isolation



Level-heterogeneity

Many-core NUMA system

Numa/Package 0

Cache group 0
Core 0 HT 0 HT 48
Core 1 HT 1 HT 49
Core 2 HT 2 **HT 50**

Cache group 1
Core 3 HT 3 HT 51
Core 4 HT 4 HT 52
Core 5 HT 5 **HT 53**

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... **HT ...**

Cache group 7
Core 21 HT 21 HT 69
Core 22 HT 22 HT 70
Core 23 HT 23 **HT 71**

Numa/Package 1

Cache group 8
Core 24 HT 24 HT 72
Core 25 HT 25 HT 73
Core 26 HT 26 HT 74

Cache group 9
Core 27 HT 27 HT 75
Core 28 HT 28 HT 76
Core 29 HT 29 HT 77

Cache group ...
Core ... HT ... HT ...
Core ... HT ... HT ...
Core ... HT ... HT ...

Cache group 15
Core 45 HT 45 HT 93
Core 46 HT 1 HT 94
Core 47 HT 2 HT 95

MCS

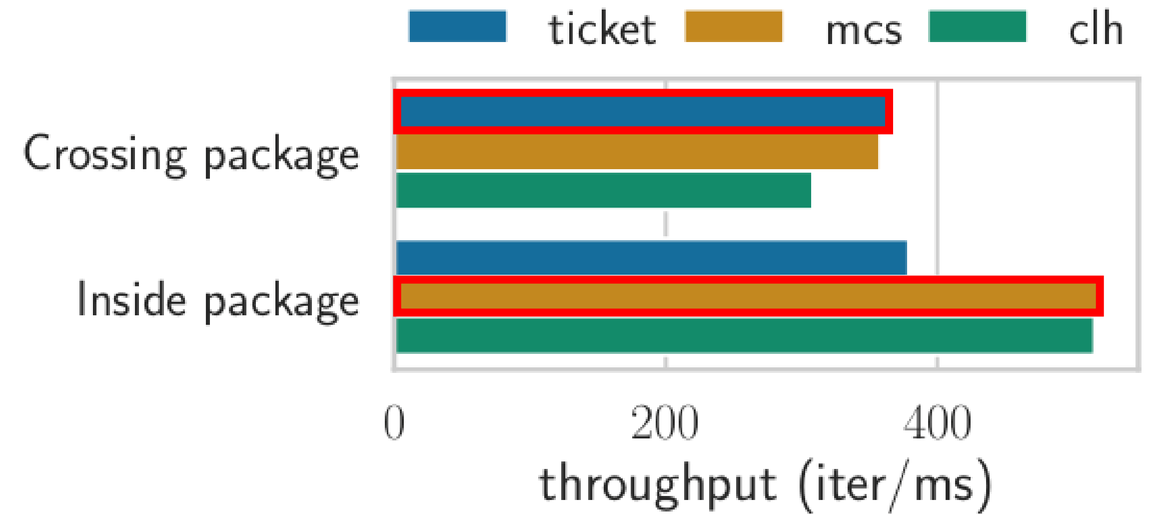
MCS ?

MCS ?

~~MCS~~ ticketlock

Experiment - Heterogeneity

x86 server – execution of classical locks in isolation



Level-heterogeneity

Many-core NUMA system

Numa/Package 0

Cache group 0: Core 0 (HT 0, HT 48), Core 1 (HT 1, HT 49), Core 2 (HT 2, **HT 50**)

Cache group 1: Core 3 (HT 3, HT 51), Core 4 (HT 4, HT 52), Core 5 (HT 5, **HT 53**)

Cache group …: Core … (HT …, HT …), Core … (HT …, HT …), Core … (HT …, **HT …**)

Cache group 7: Core 21 (HT 21, HT 69), Core 22 (HT 22, HT 70), Core 23 (HT 23, **HT 71**)

Numa/Package 1

Cache group 8: Core 24 (HT 24, HT 72), Core 25 (HT 25, HT 73), Core 26 (HT 26, HT 74)

Cache group 9: Core 27 (HT 27, HT 75), Core 28 (HT 28, HT 76), Core 29 (HT 29, HT 77)

Cache group …: Core … (HT …, HT …), Core … (HT …, HT …), Core … (HT …, HT …)

Cache group 15: Core 45 (HT 45, HT 93), Core 46 (HT 1, HT 94), Core 47 (HT 2, HT 95)

MCS

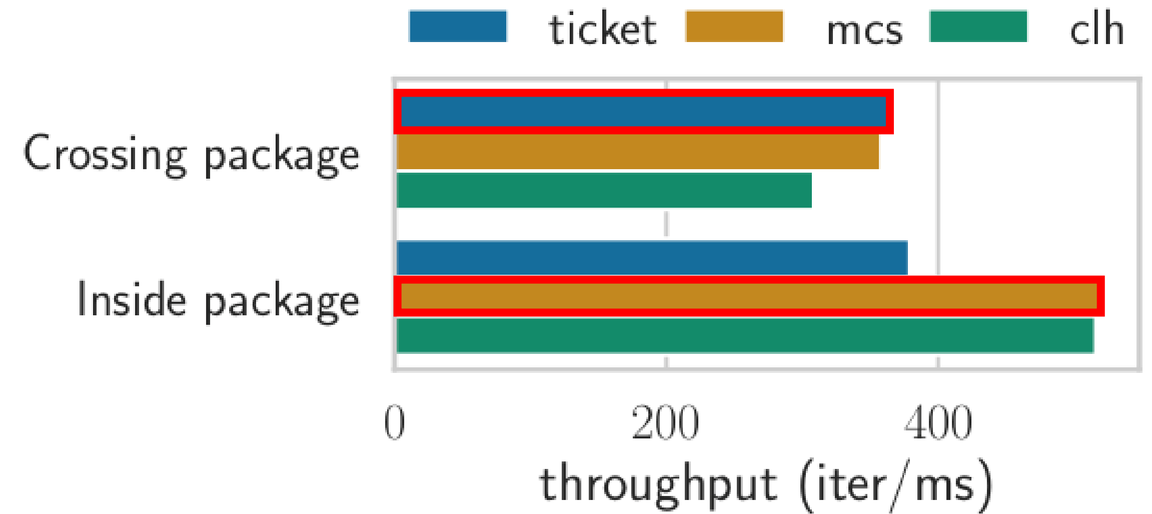
MCS ?

MCS ?

ticketlock

Experiment - Heterogeneity

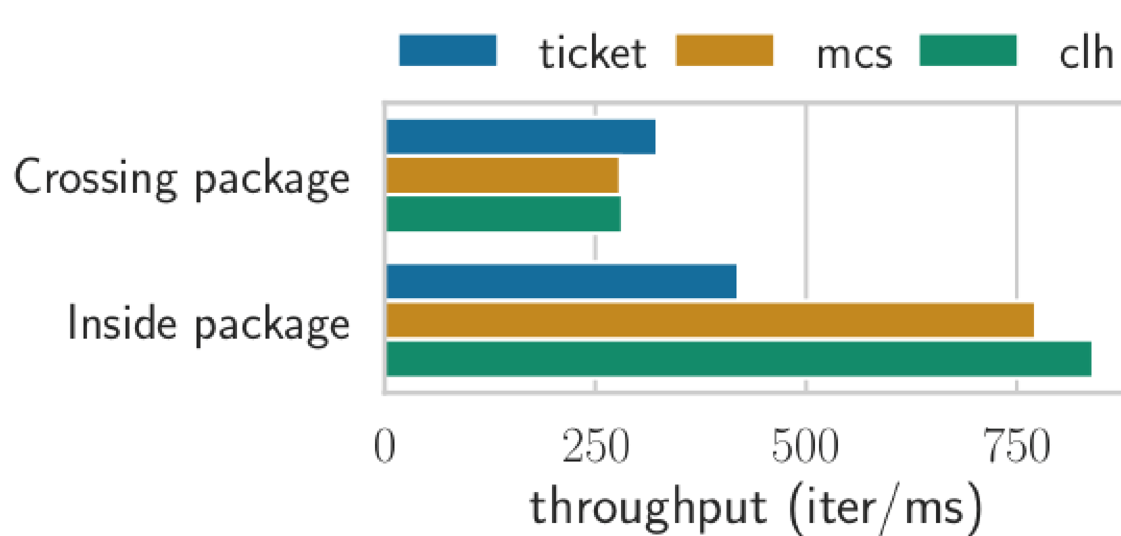
x86 server – execution of classical locks in isolation



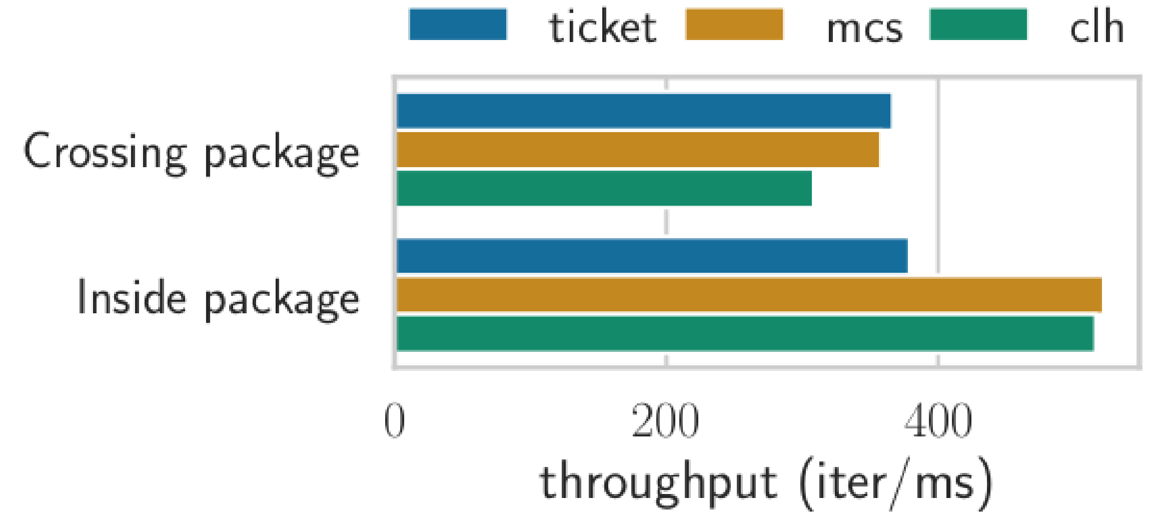
Level-heterogeneity:
For different levels,
the best lock may differ

Experiment – Platform Optimization   Experiment - Heterogeneity

Arm server – execution of classical locks in isolation



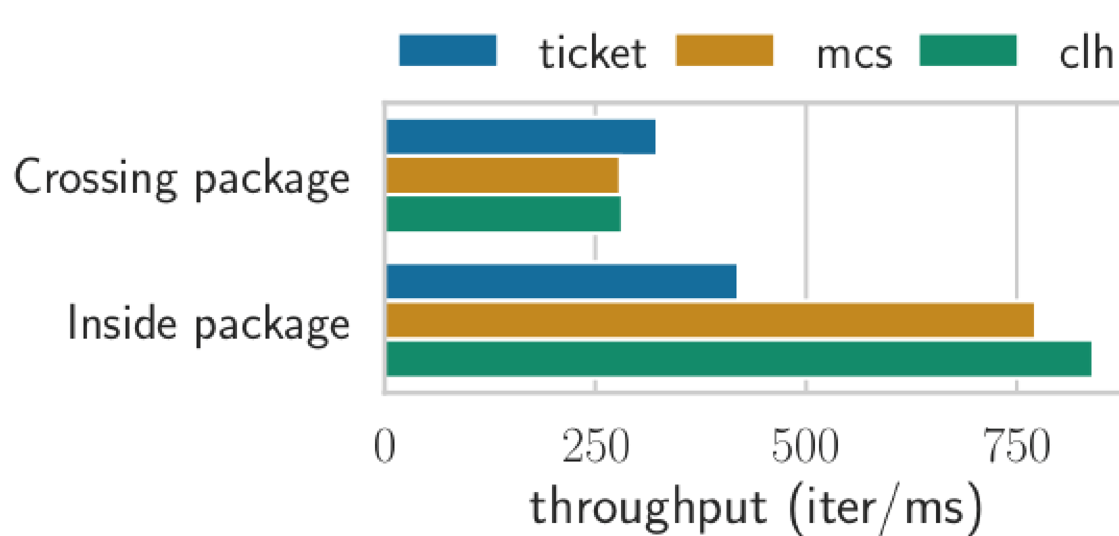
x86 server – execution of classical locks in isolation



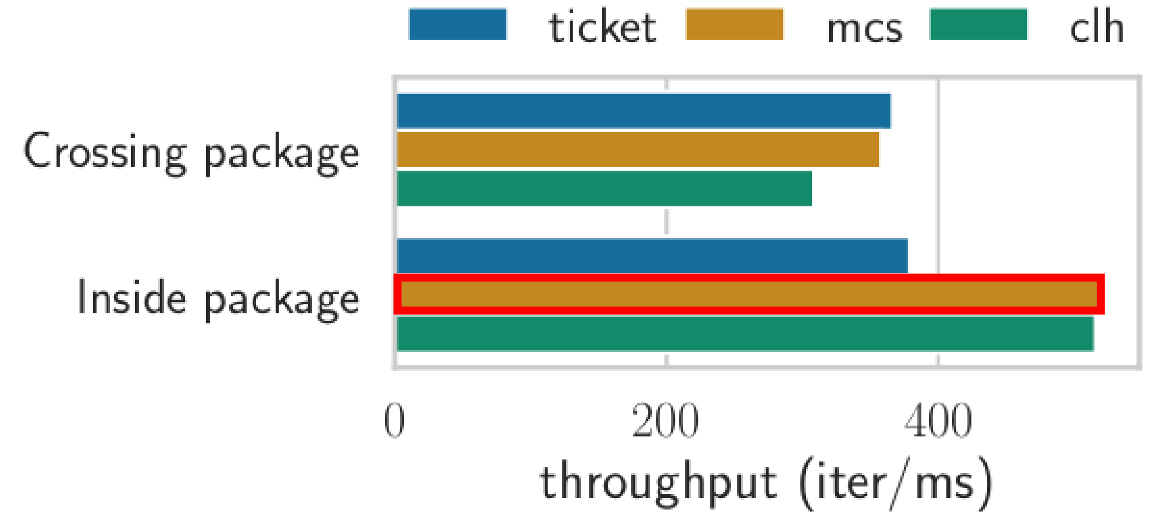
Level-heterogeneity:
For different levels,
the best lock may differ

Experiment – Platform Optimization   Experiment - Heterogeneity

Arm server – execution of classical locks in isolation



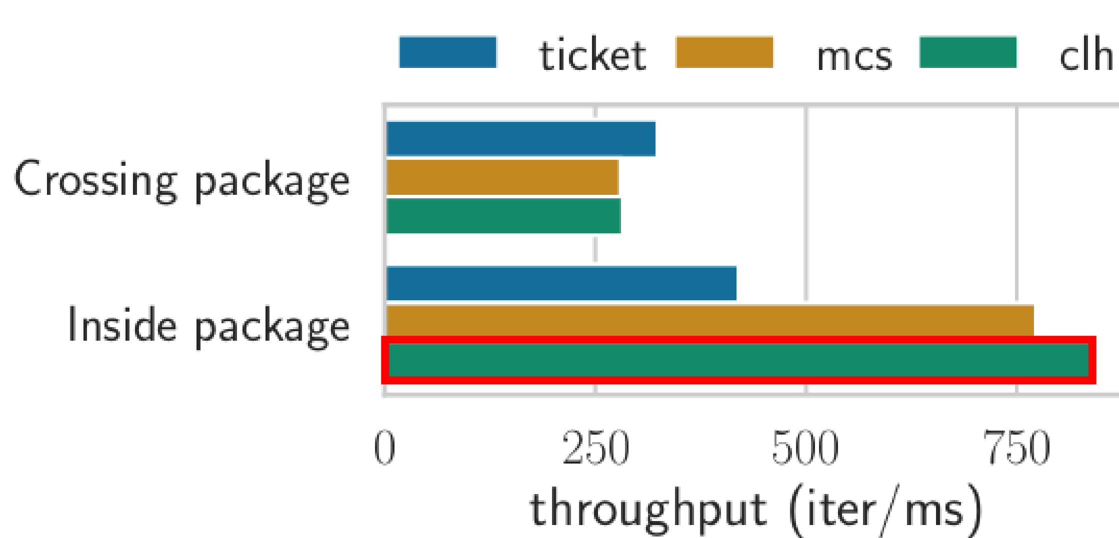
x86 server – execution of classical locks in isolation



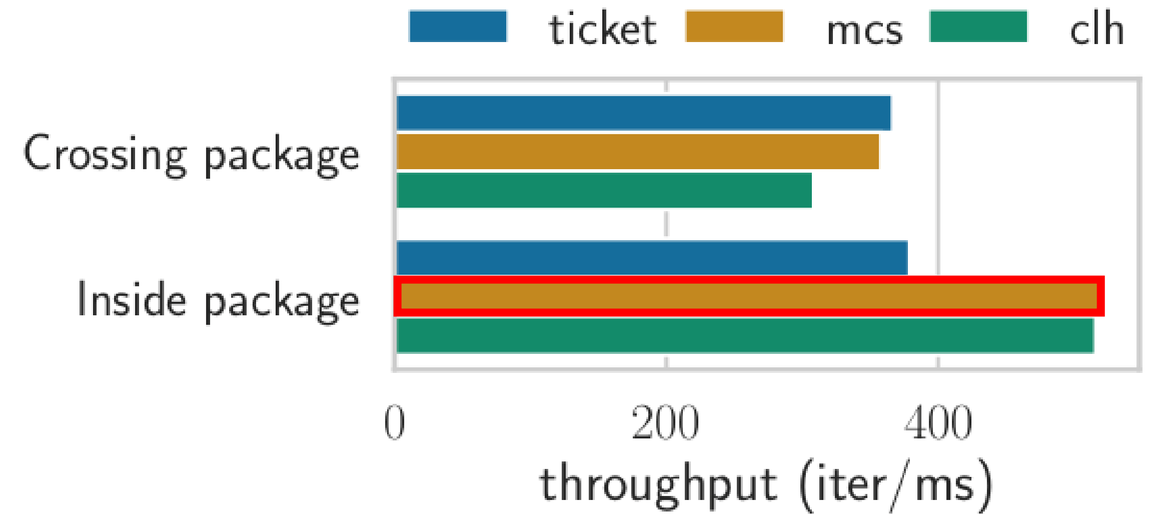
Level-heterogeneity:
For different levels,
the best lock may differ

Experiment – Platform Optimization   Experiment - Heterogeneity

Arm server – execution of classical locks in isolation



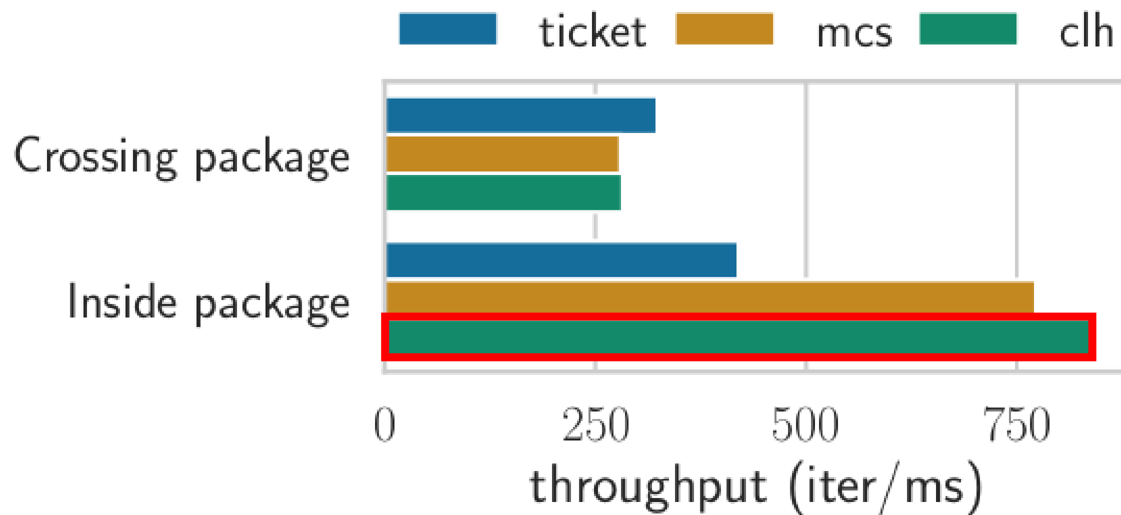
x86 server – execution of classical locks in isolation



Level-heterogeneity:
For different levels,
the best lock may differ

Experiment – Platform Optimization

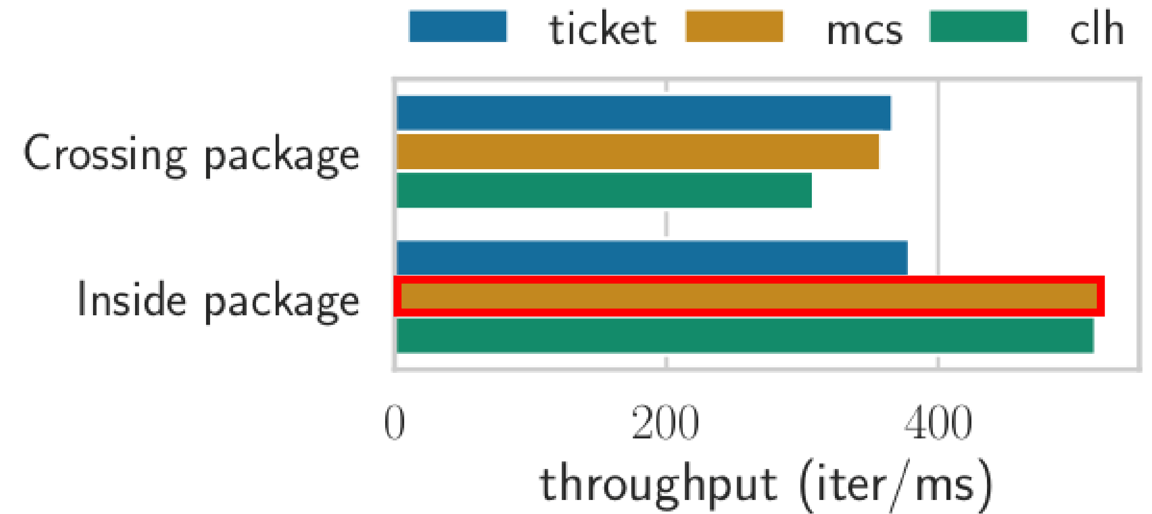
Arm server – execution of classical locks in isolation



Platform Optimization:
For different platforms,
the best lock for a level may differ

Experiment - Heterogeneity

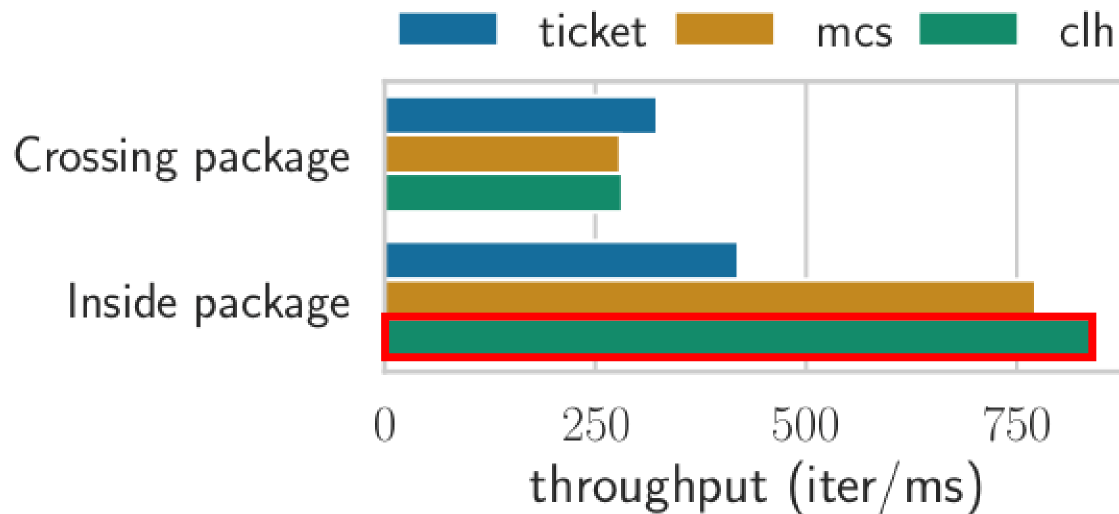
x86 server – execution of classical locks in isolation



Level-heterogeneity:
For different levels,
the best lock may differ

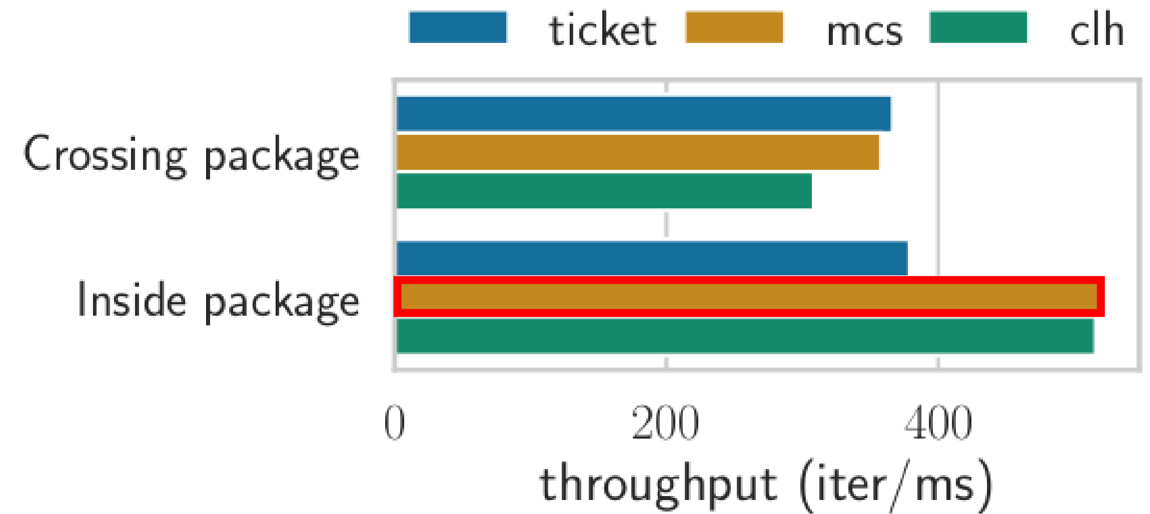
Experiment – Platform Optimization   Experiment - Heterogeneity

Arm server – execution of classical locks in isolation



Platform Optimization:
For different platforms,
the best lock for a level may differ

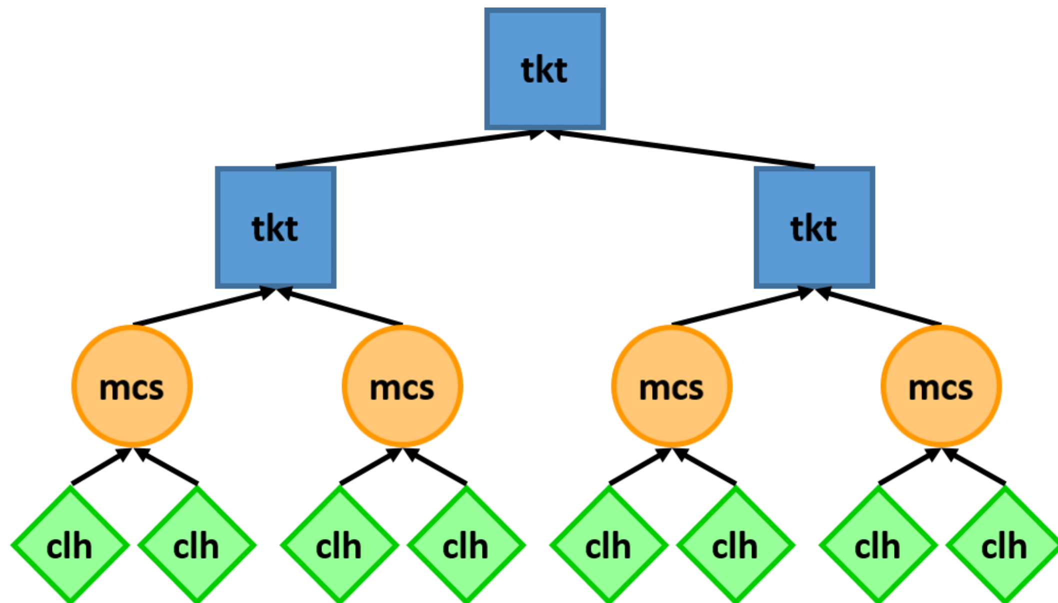
x86 server – execution of classical locks in isolation



Level-heterogeneity:
For different levels,
the best lock may differ

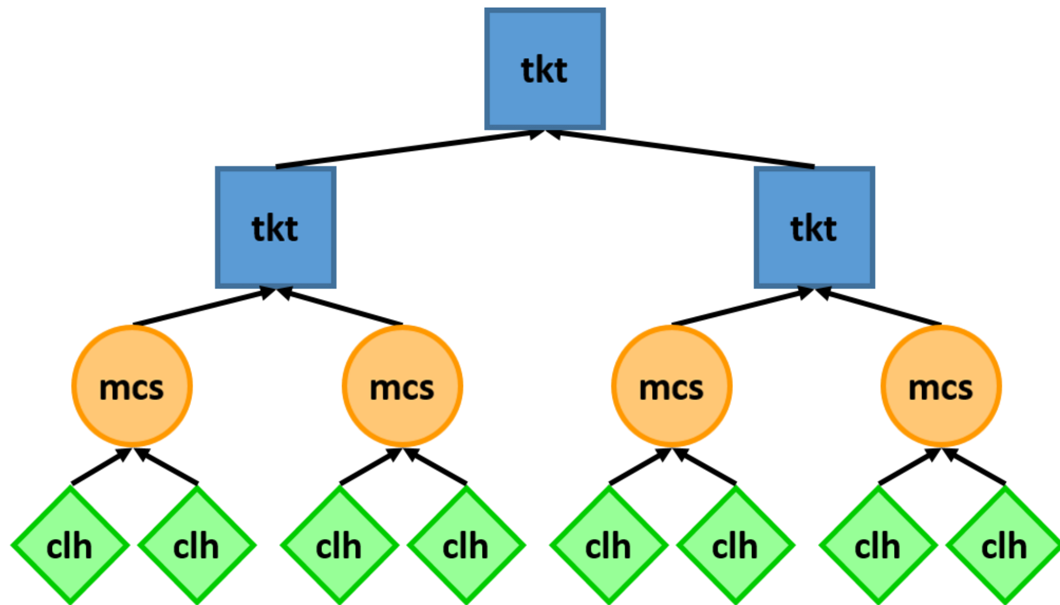
So, how does this affects our lock design?

Our Desired NUMA-aware lock



- Multi-Level
- Level-Heterogeneous
- Configurability for Platform Optimization

Our Desired NUMA-aware lock



- Multi-Level
- Level-Heterogeneous
- Configurability for Platform Optimization

- Showing lock correctness is challenging
  - Weak Memory Models (WMMs) make it even more complicated

Our contribution: CLoF

We propose CLoF, a framework to generate locks for a target platform:

- that support an arbitrary hierarchy;
- for each level, the lock implementation may be different;
- that are *correct-by-construction* on Weak Memory Models.

Our contribution: CLoF

We propose CLoF, a framework to generate locks for a target platform:

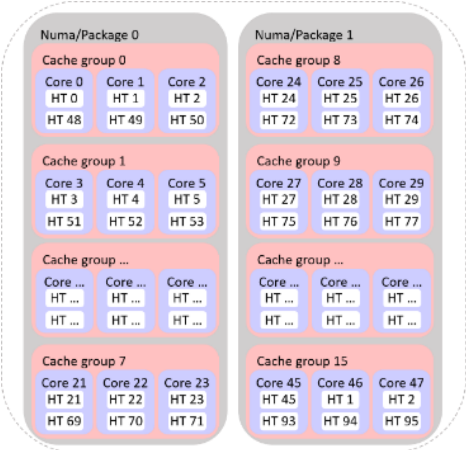
- that support an arbitrary hierarchy;
- for each level, the lock implementation may be different;
- that are *correct-by-construction* on Weak Memory Models.

NUMA-aware locks		Correctness on WMMs	Level heterogeneity & Architecture optimization	Multi-Level
lock cohorting	PPPoPP'12	✗	✓	✗
HMCS	PPoPP'15	✗[1]	✗	✓
CNA lock	EuroSys'19	✗	✗	✗
ShflLock	SOSP'19	✗	✗	✗
CLoF	SOSP'21	✓	✓	✓

[1] Insufficient barriers, fixed in Oberhauser *et al.*, Verifying and Optimizing the HMCS Lock for Arm Servers, NETYS'2021

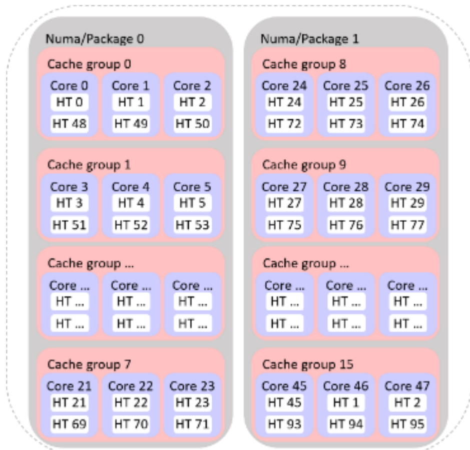
CLoF Workflow

Discover Memory Hierarchy



CLoF Workflow

Discover Memory Hierarchy



Verify correctness on WMMs

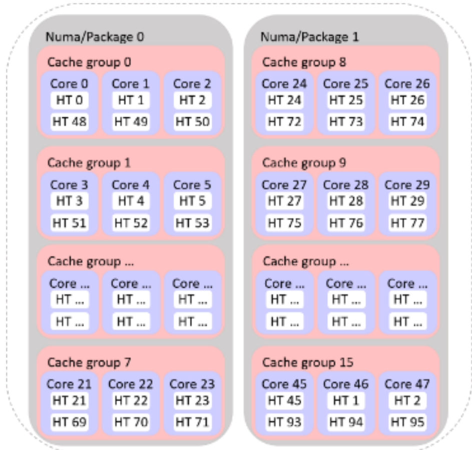
tkt

clh

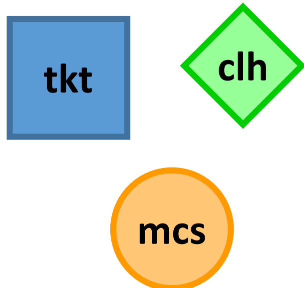
mcs

CLoF Workflow

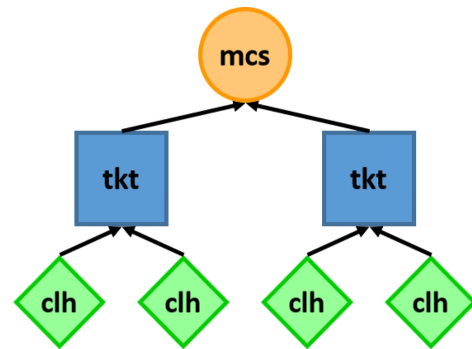
Discover Memory Hierarchy



Verify correctness on WMMs

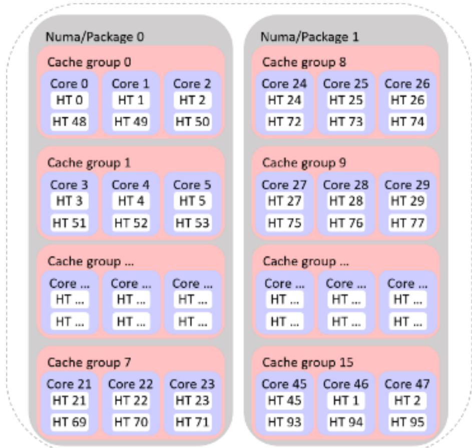


CLoF Lock Generator

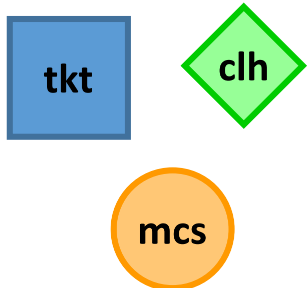


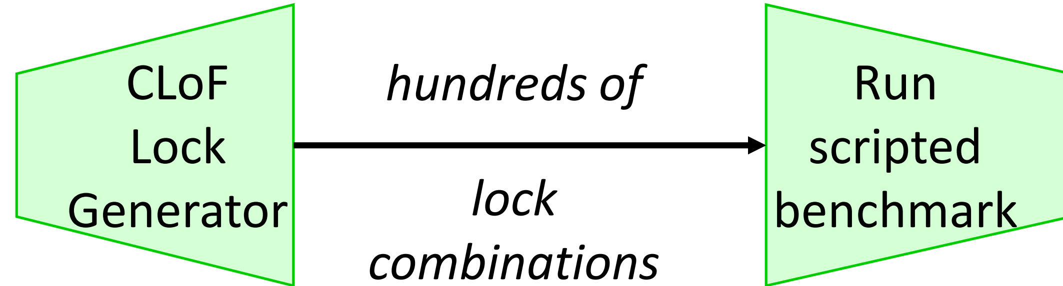
CLoF Workflow

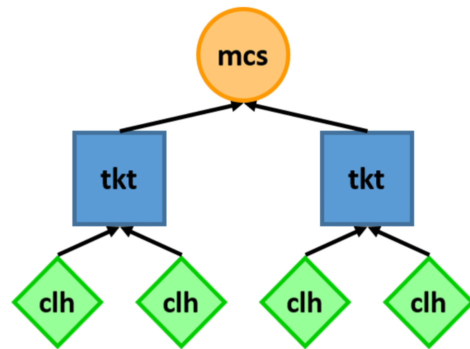
Discover Memory Hierarchy

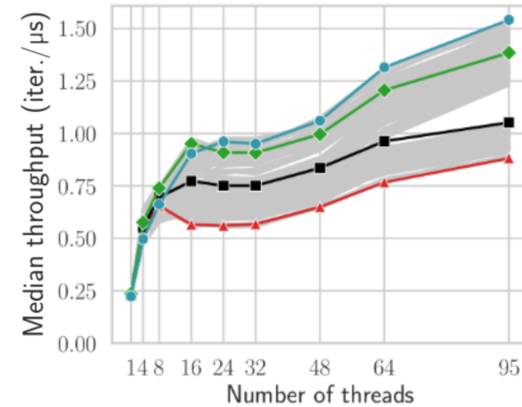


Verify correctness on WMMs



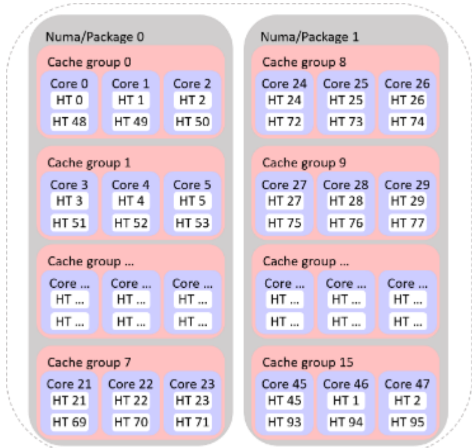




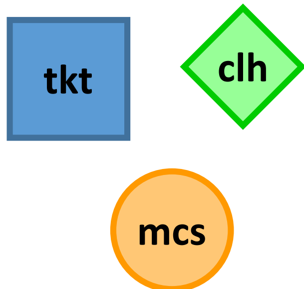


CLoF Workflow

Discover Memory Hierarchy



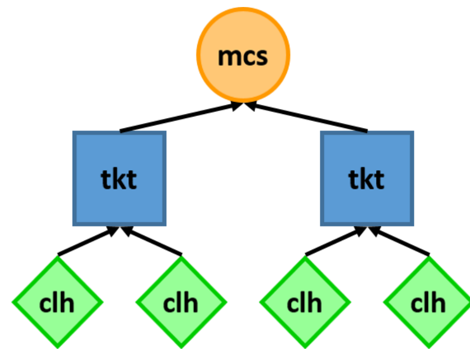
Verify correctness on WMMs



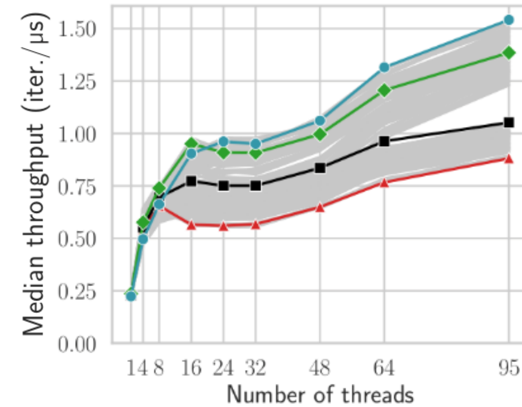
CLoF Lock Generator

hundreds of

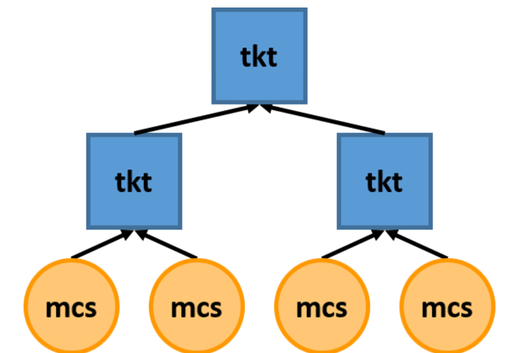
lock combinations



Run scripted benchmark



Select Best Lock



Agenda

- How to figure out the hierarchy we need to use?
- How does our CLoF Lock Generator works?
- How do we know it is correct?
- How do we pick the best lock for the target platform?

Agenda

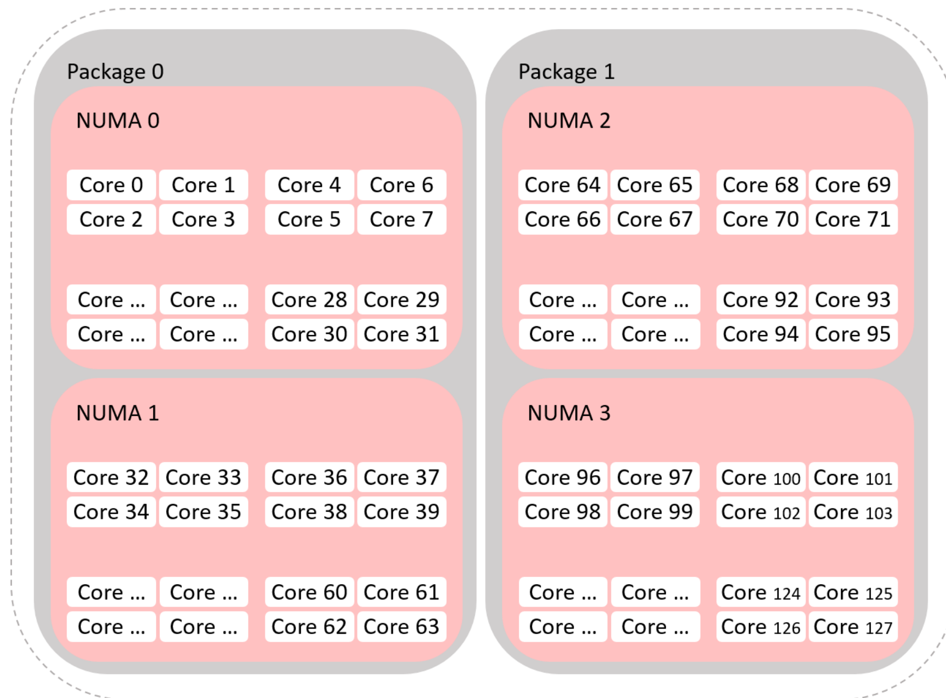
- How to figure out the hierarchy we need to use?
- How does our CLoF Lock Generator works?
- How do we know it is correct?
- How do we pick the best lock for the target platform?

Discovering the Memory Hierarchy

Operating systems know the hierarchy

- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8

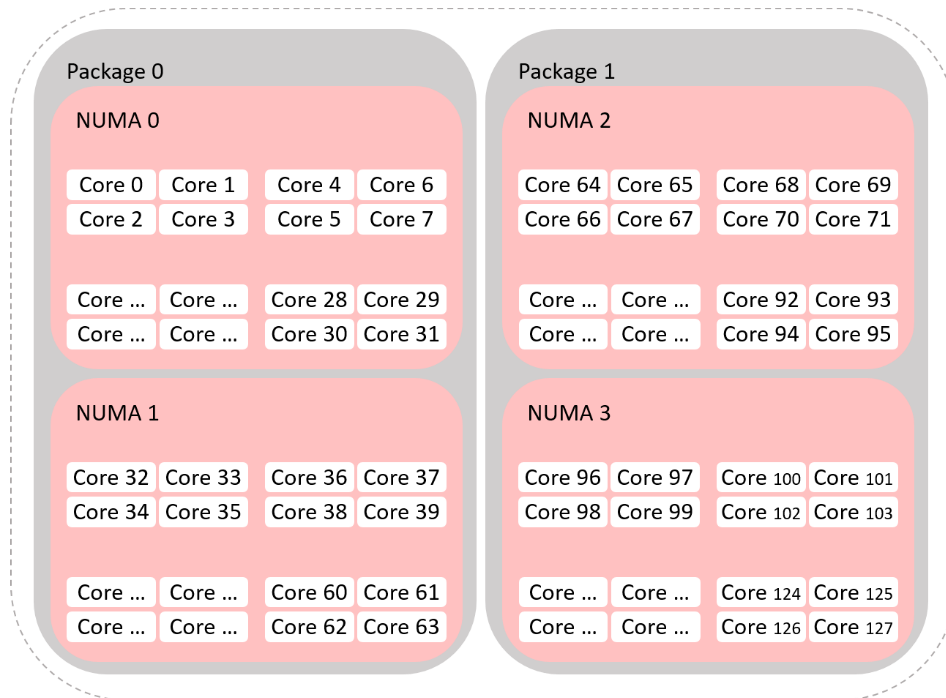


Discovering the Memory Hierarchy

Operating systems know the hierarchy

- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



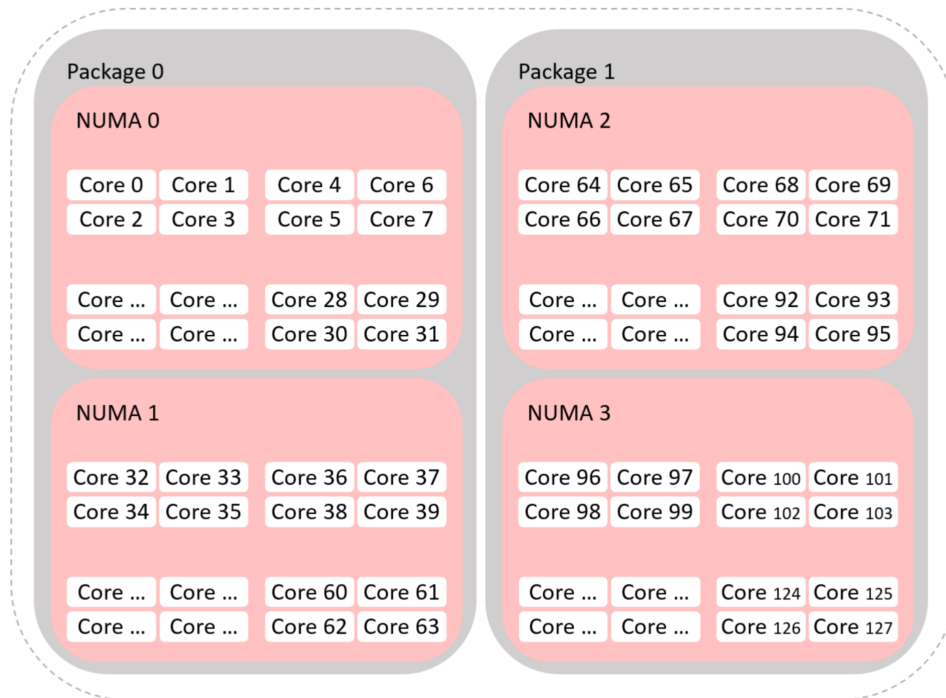
Is this the **full** hierarchy?

Discovering the Memory Hierarchy

Operating systems know the hierarchy

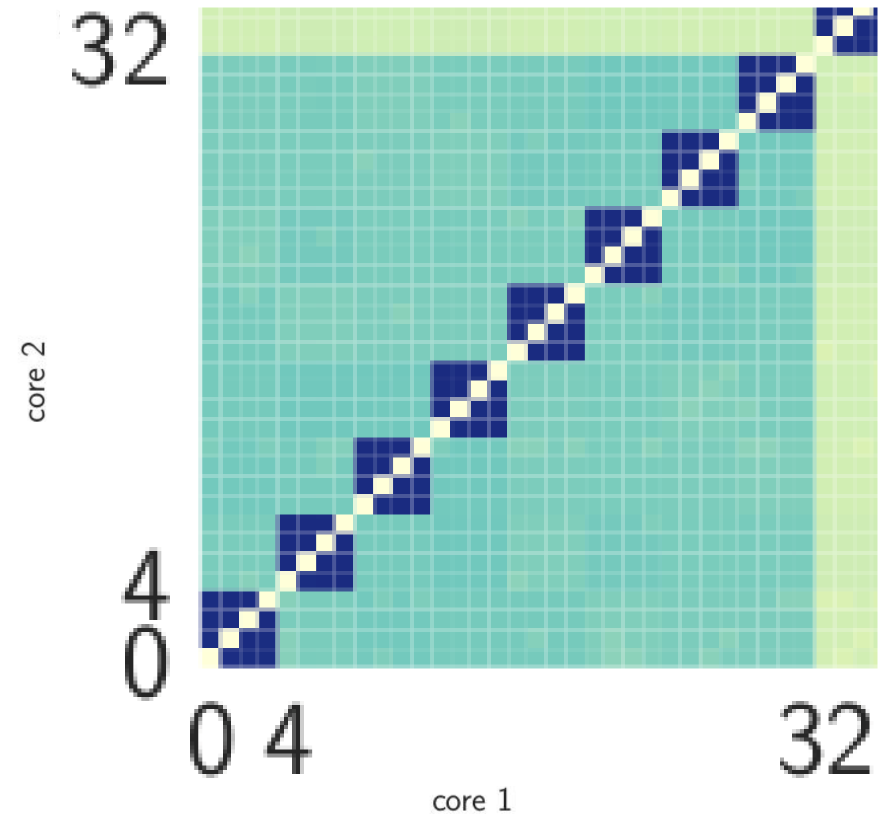
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

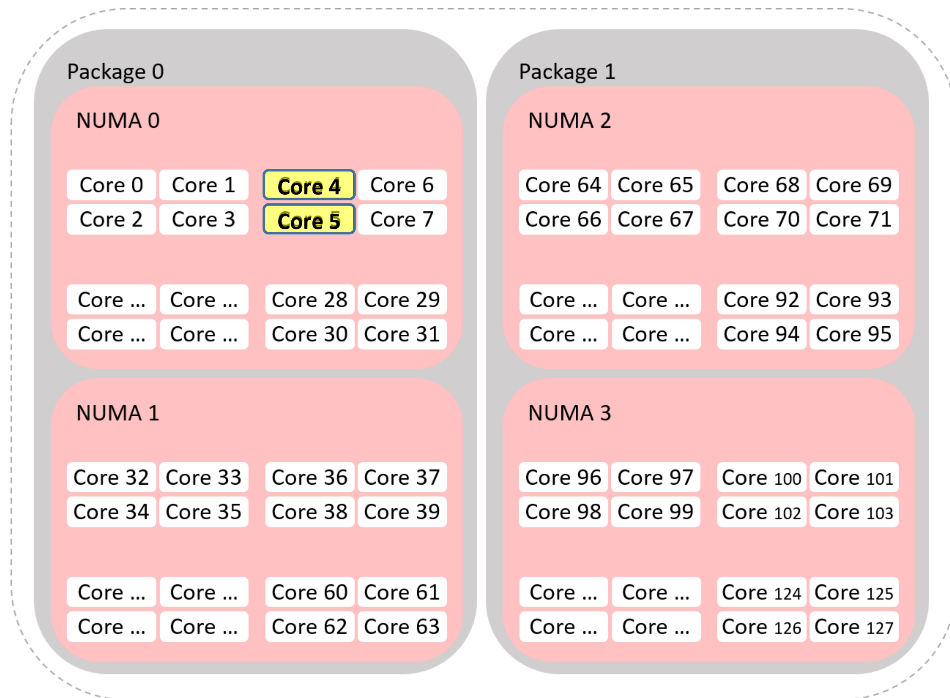


Discovering the Memory Hierarchy

Operating systems know the hierarchy

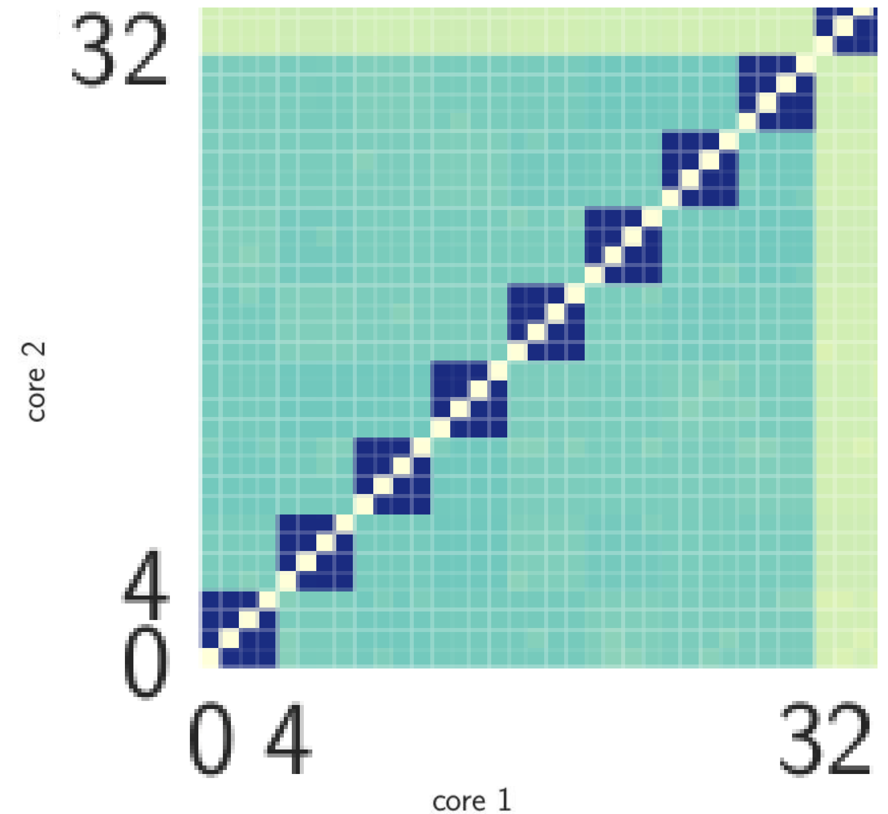
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

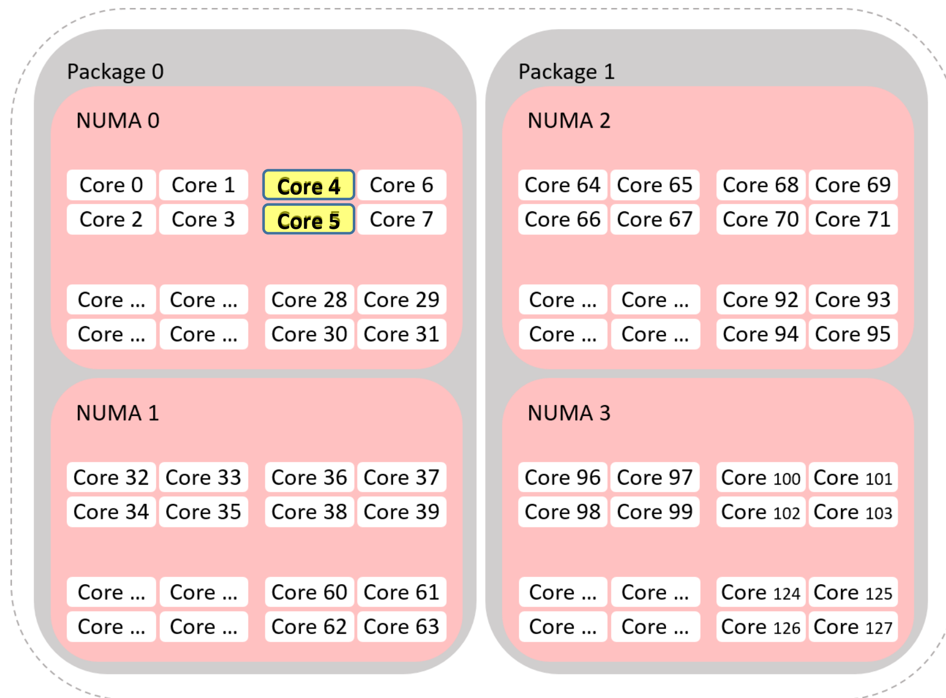


Discovering the Memory Hierarchy

Operating systems know the hierarchy

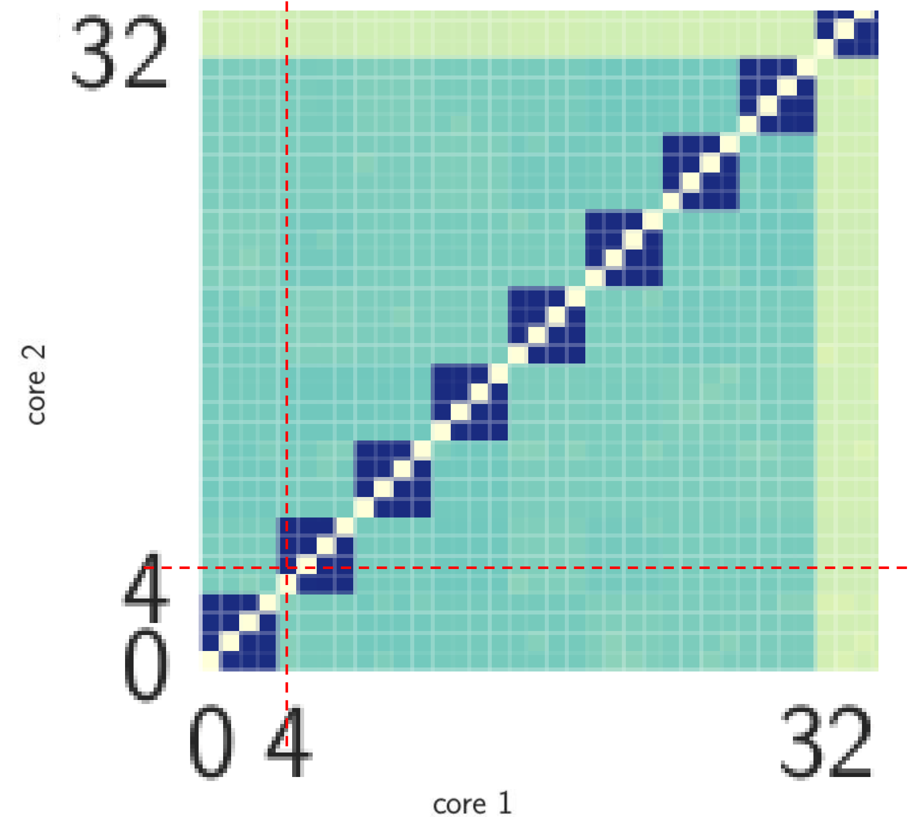
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

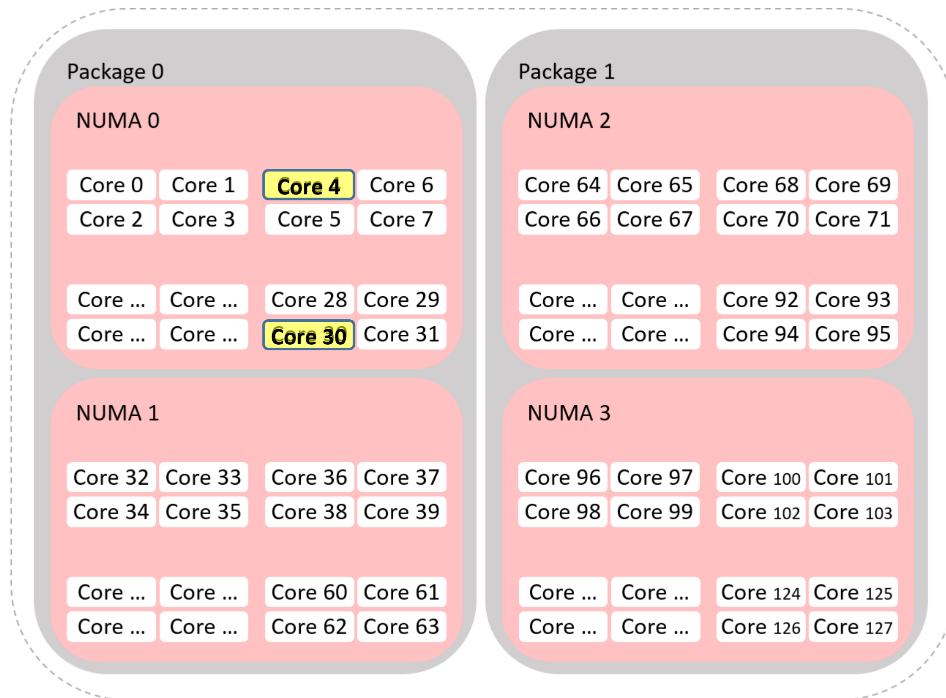


Discovering the Memory Hierarchy

Operating systems know the hierarchy

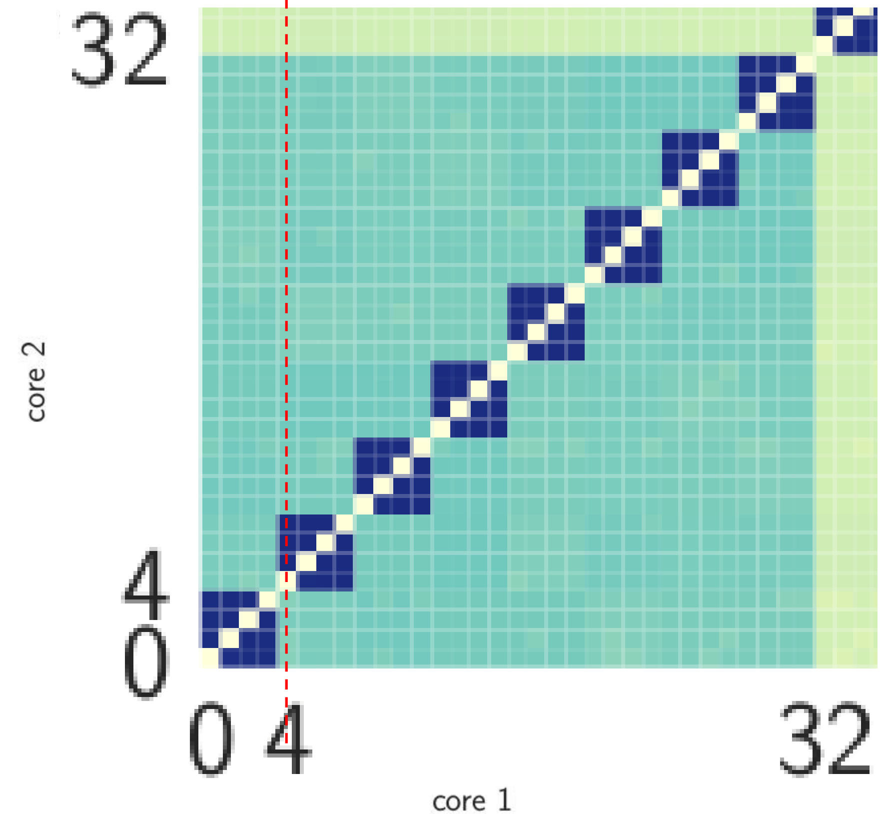
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

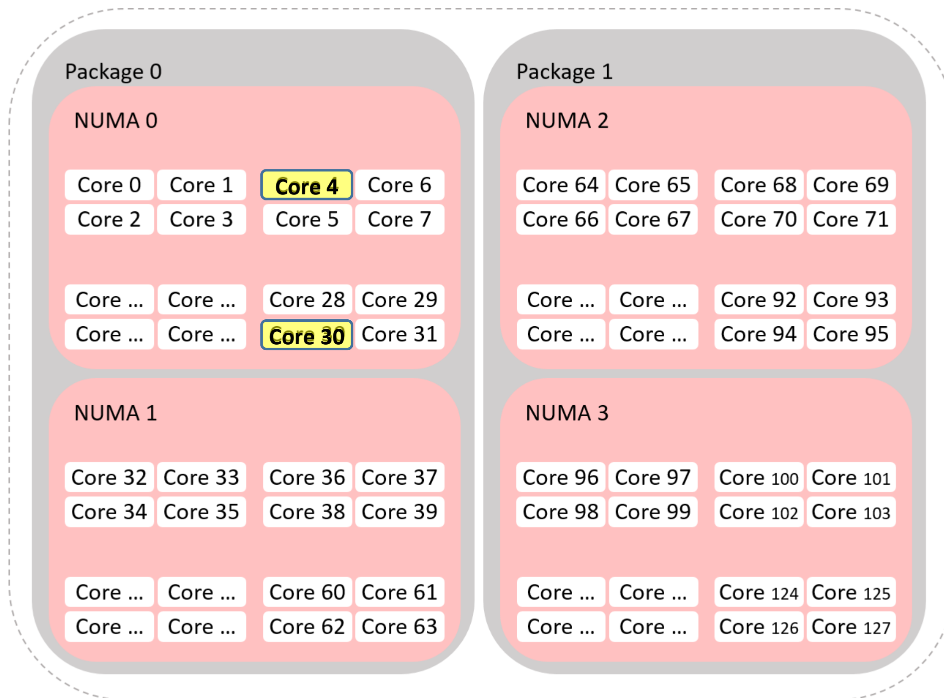


Discovering the Memory Hierarchy

Operating systems know the hierarchy

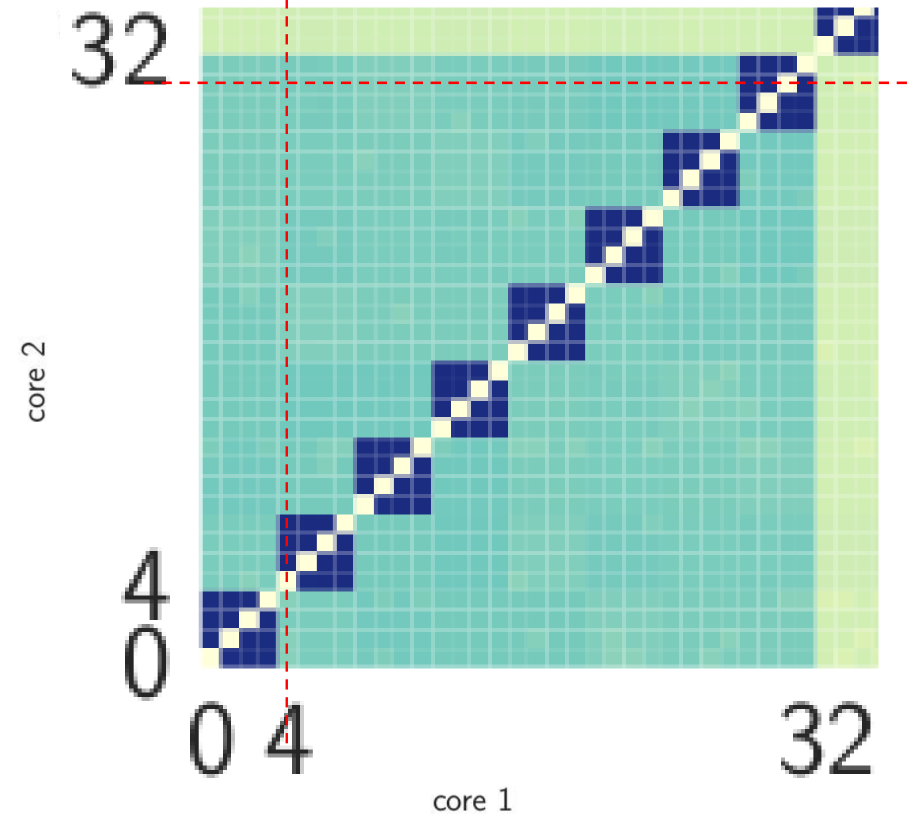
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

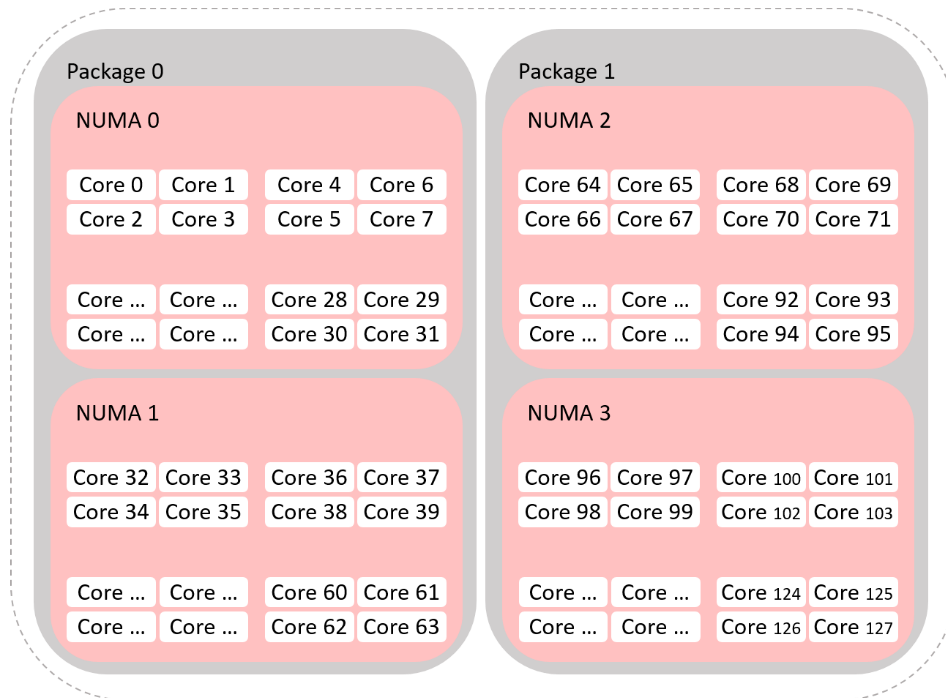


Discovering the Memory Hierarchy

Operating systems know the hierarchy

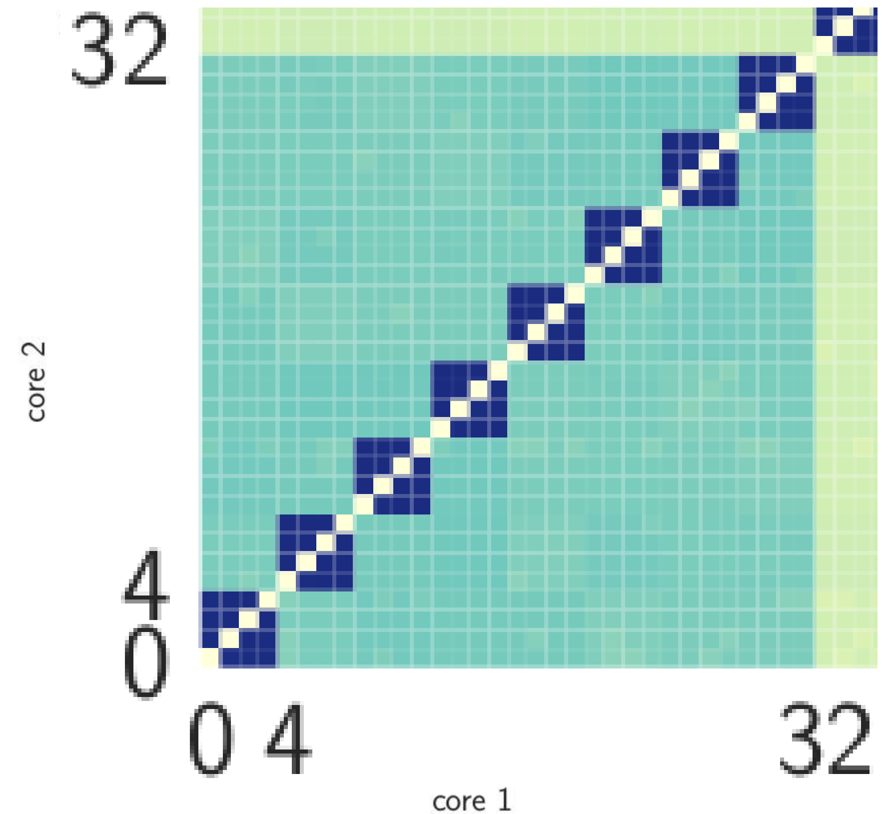
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

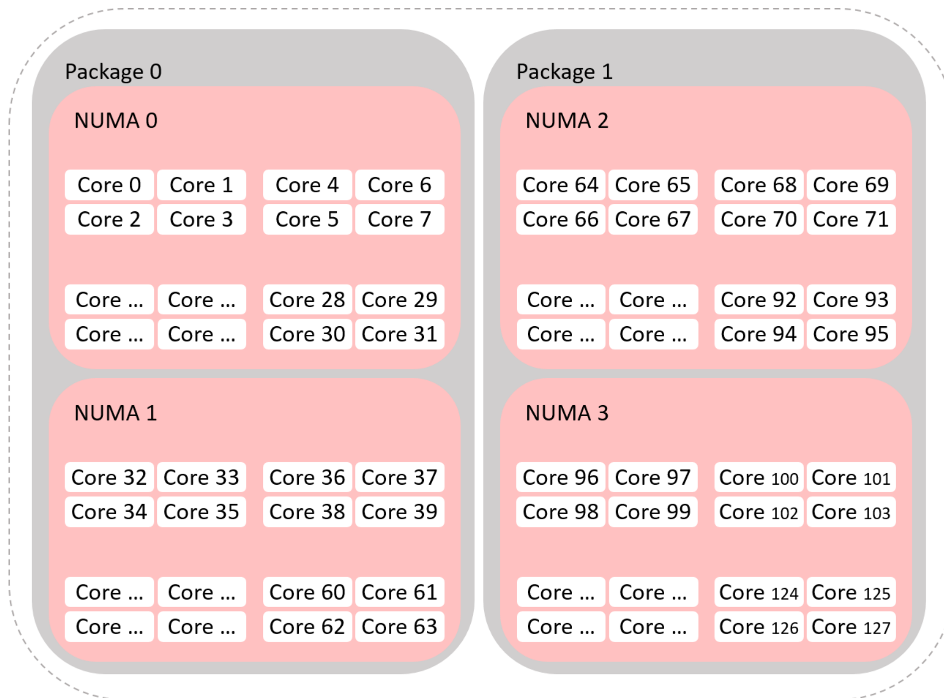


Discovering the Memory Hierarchy

Operating systems know the hierarchy

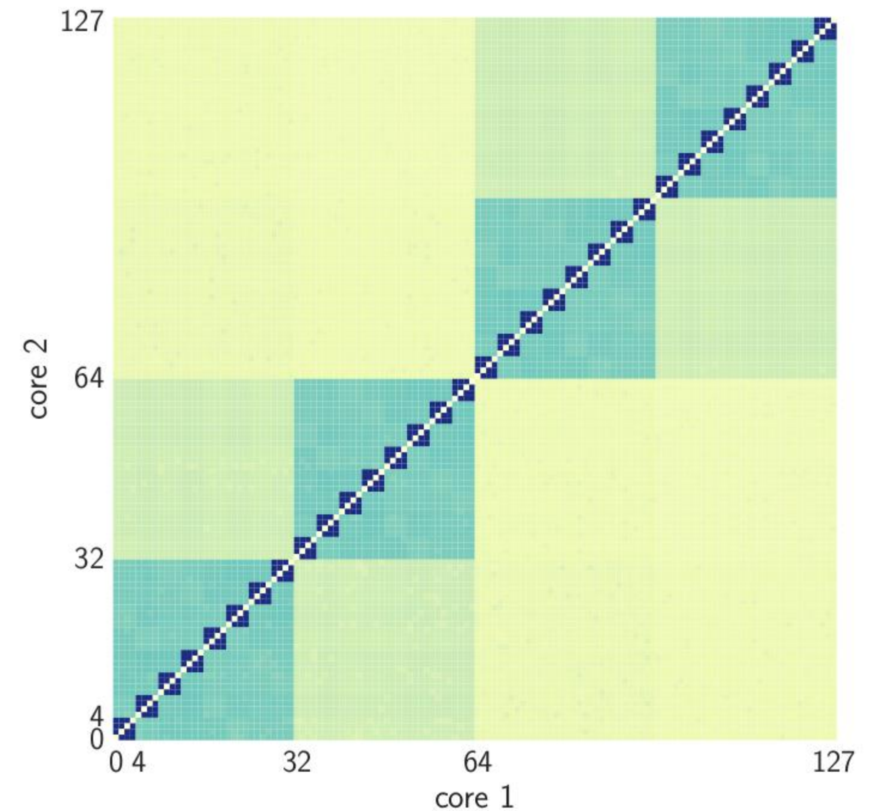
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

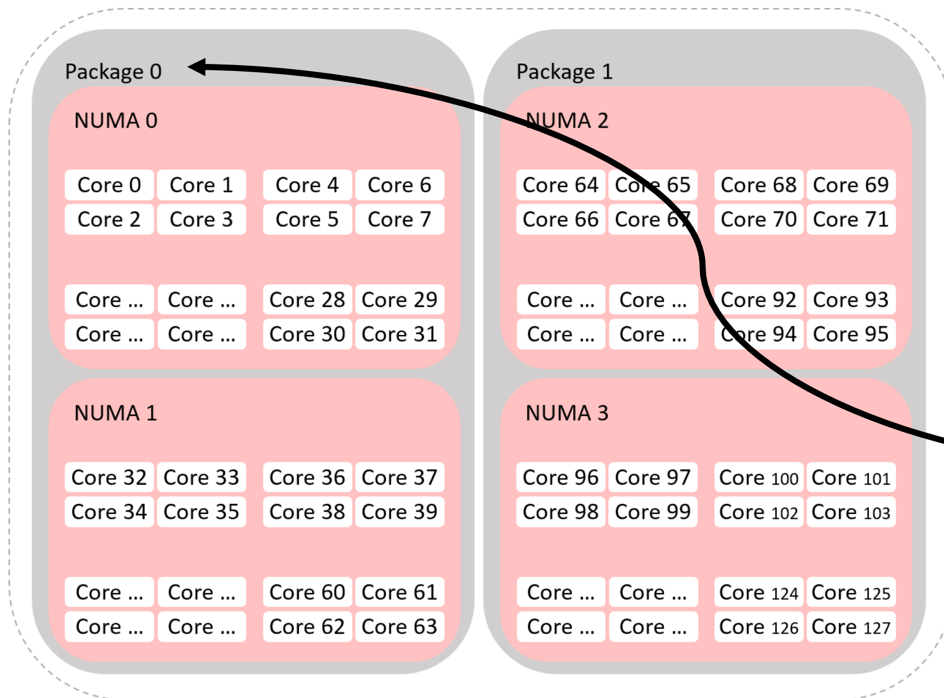


Discovering the Memory Hierarchy

Operating systems know the hierarchy

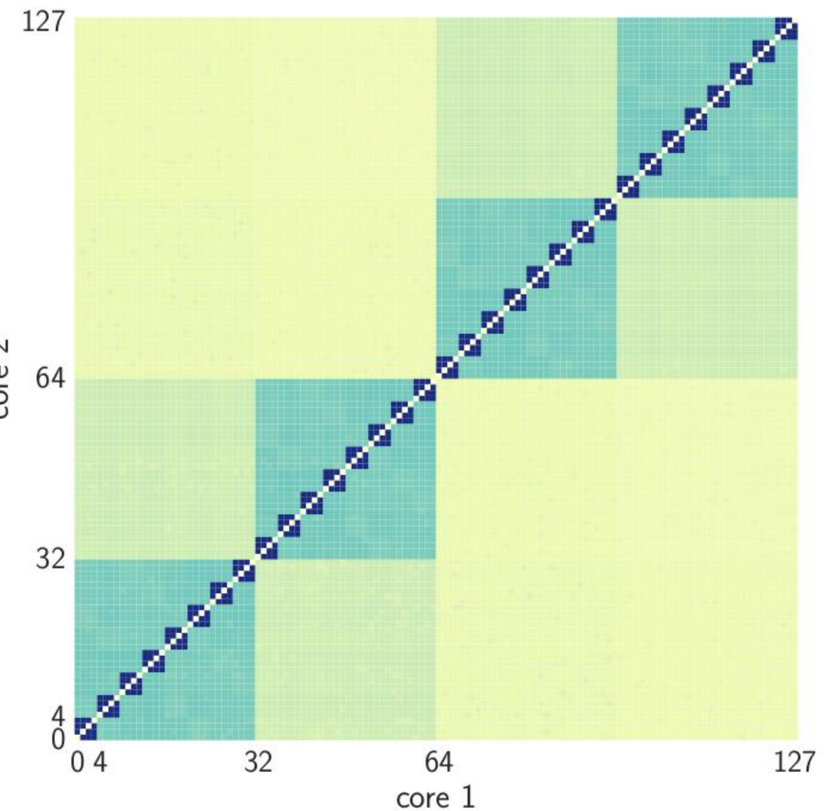
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

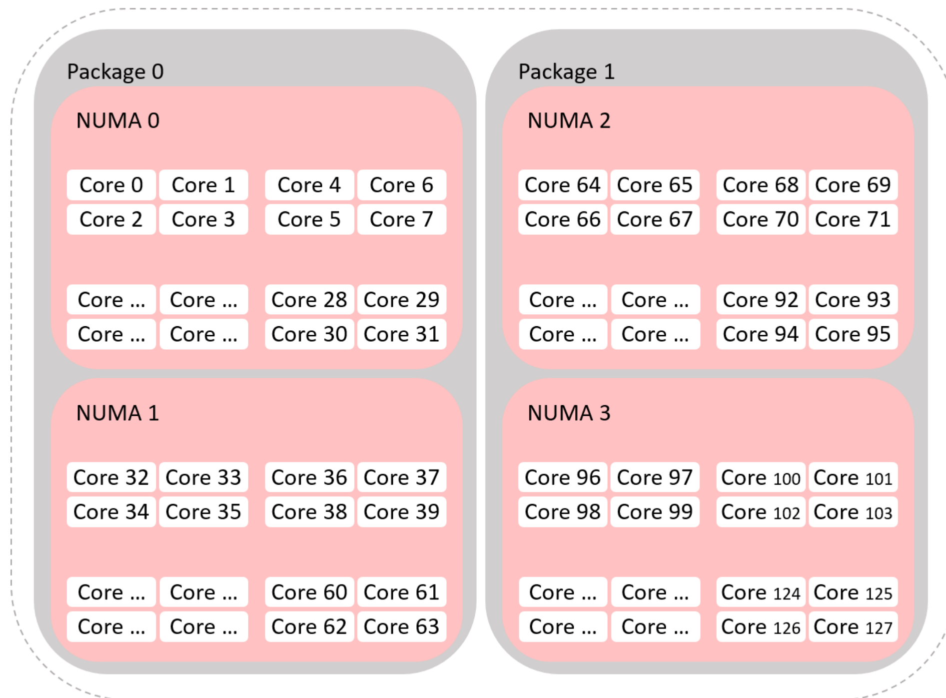


Discovering the Memory Hierarchy

Operating systems know the hierarchy

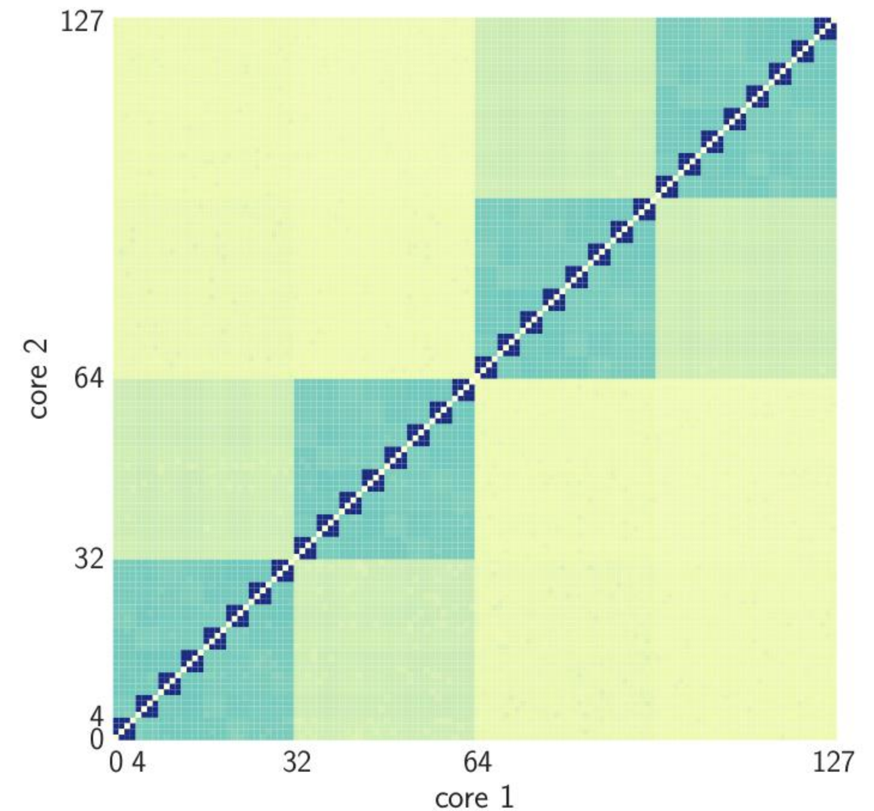
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

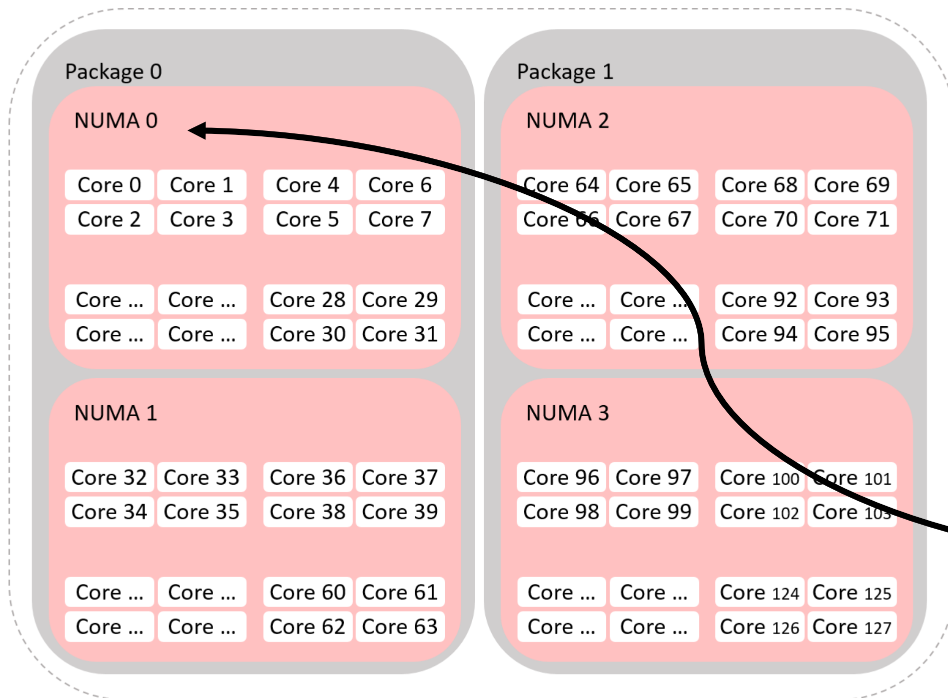


Discovering the Memory Hierarchy

Operating systems know the hierarchy

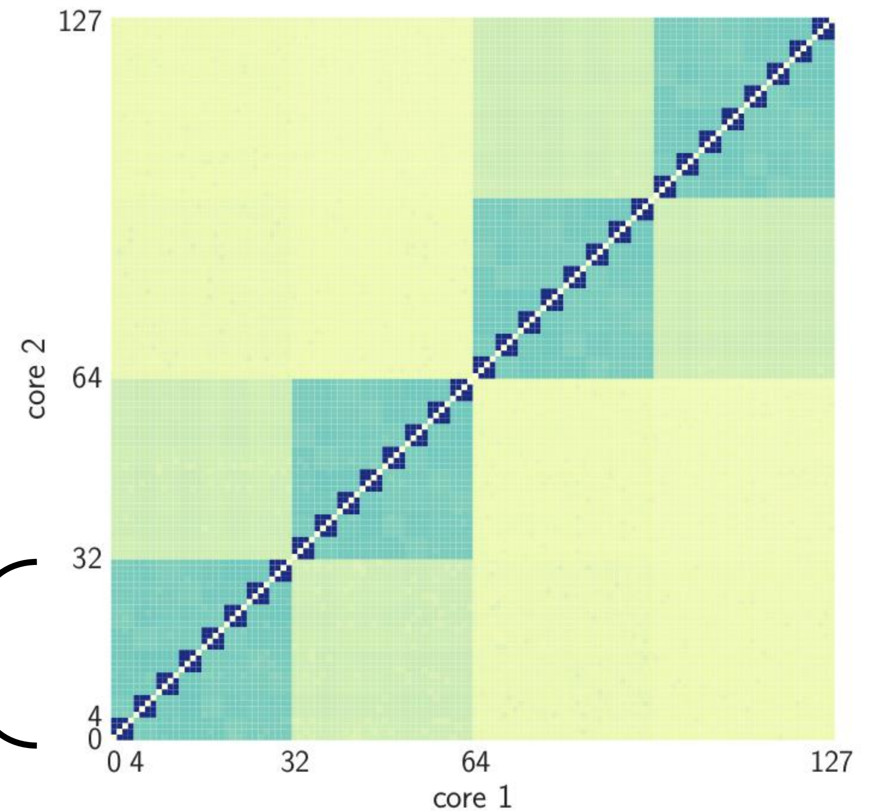
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

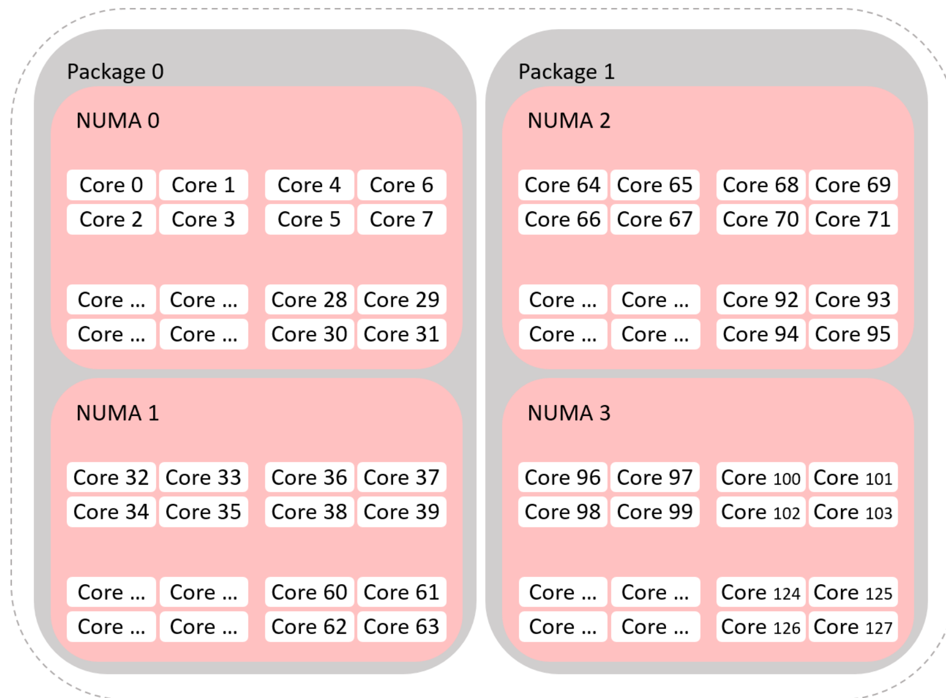


Discovering the Memory Hierarchy

Operating systems know the hierarchy

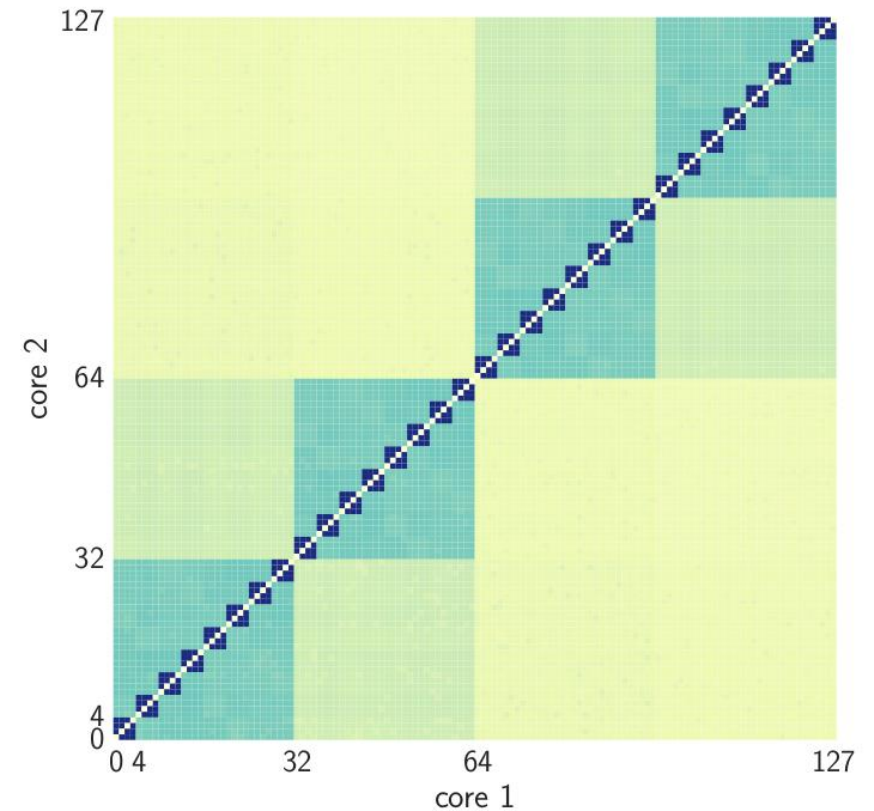
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

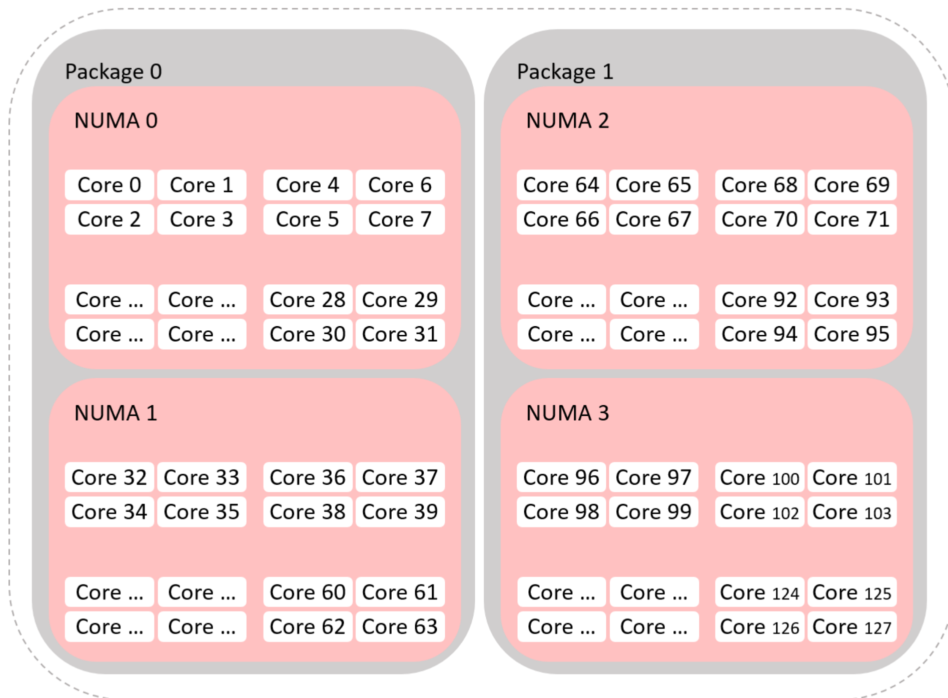


Discovering the Memory Hierarchy

Operating systems know the hierarchy

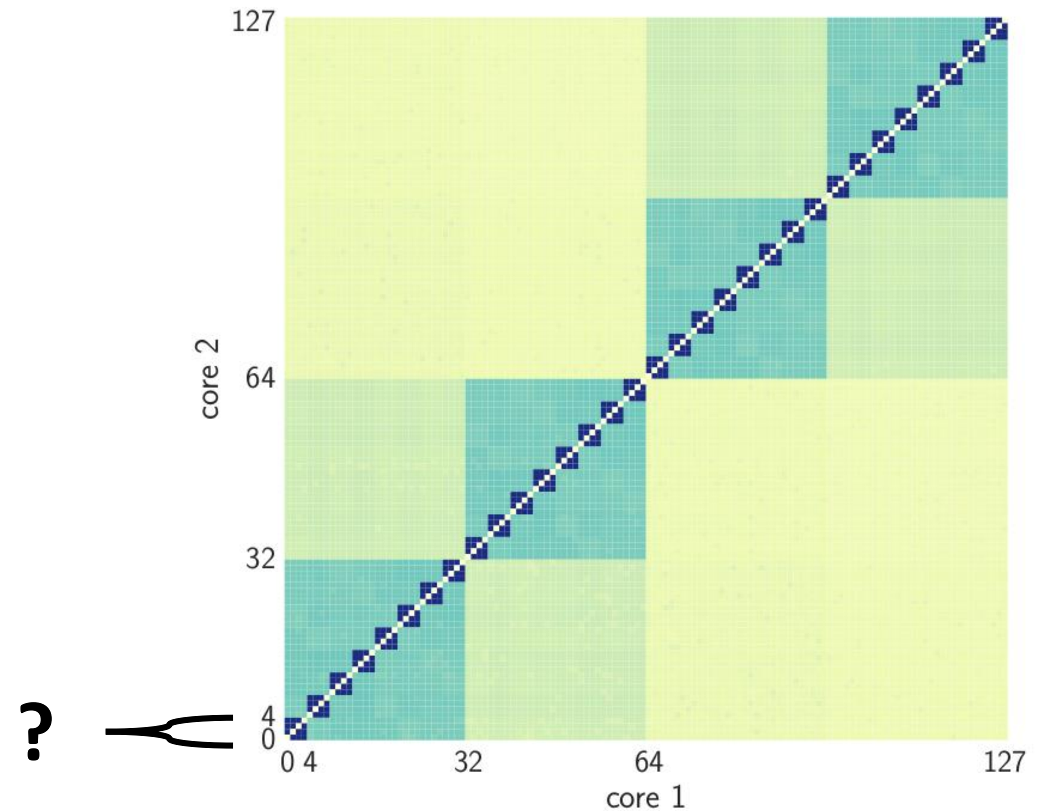
- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



Experimental Discovery

- 2 threads alternately increment a shared counter
- Darker colors => Higher throughput

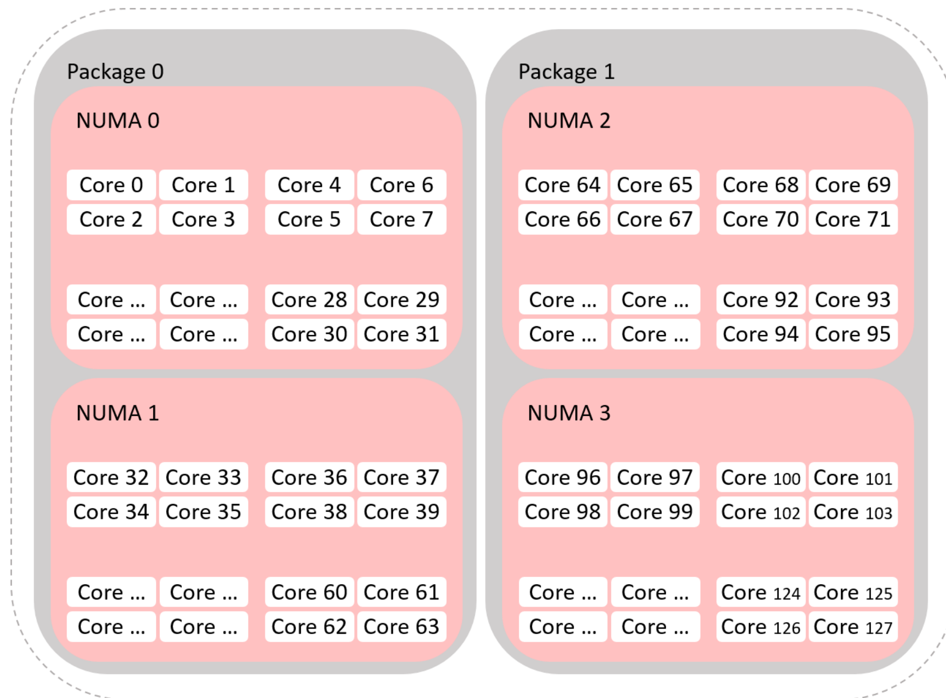


Discovering the Memory Hierarchy

Operating systems know the hierarchy

- Visible with Linux's lscpu and lstopo

Taishan 200 server – Armv8



- Not shown by lscpu/lstopo
- Information is shown in processors' datasheet
  - Not efficient
  - Remained unstudied and unused
    - HMCS<4> includes hidden cache group level

Agenda

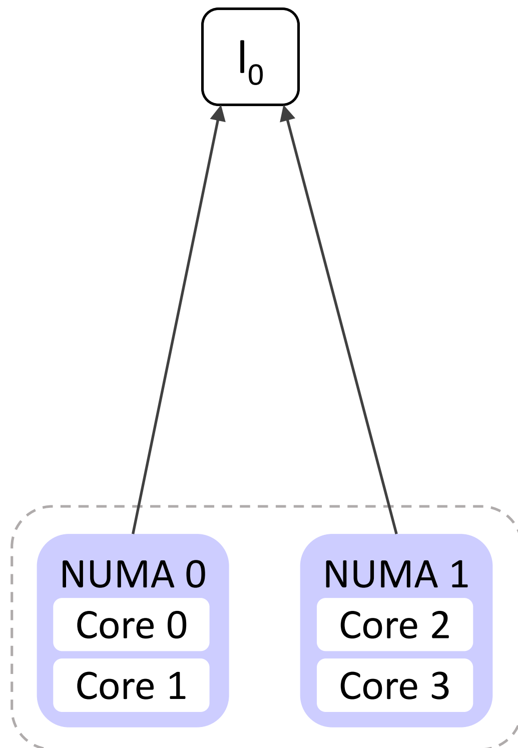
- How to figure out the hierarchy we need to use?
- How does our CLoF Lock Generator works?
- How do we know it is correct?
- How do we pick the best lock for the target platform?

CLoF Lock Generator

Two NUMA node example

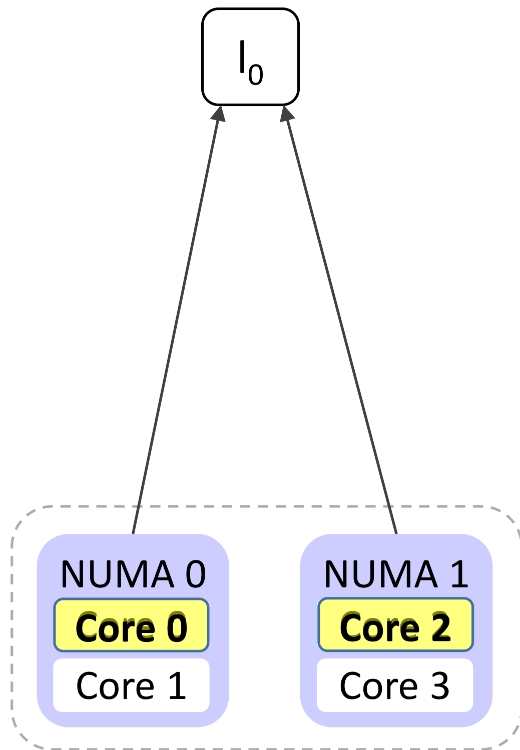
CLoF Lock Generator

Two NUMA node example



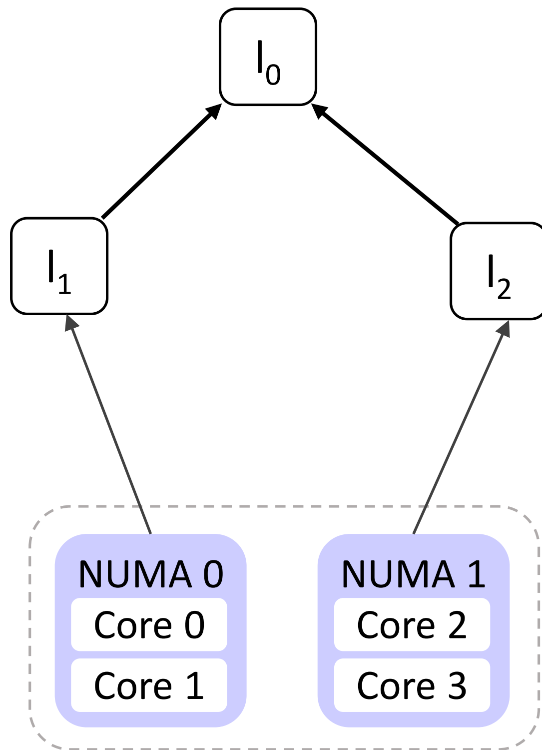
CLoF Lock Generator

Two NUMA node example



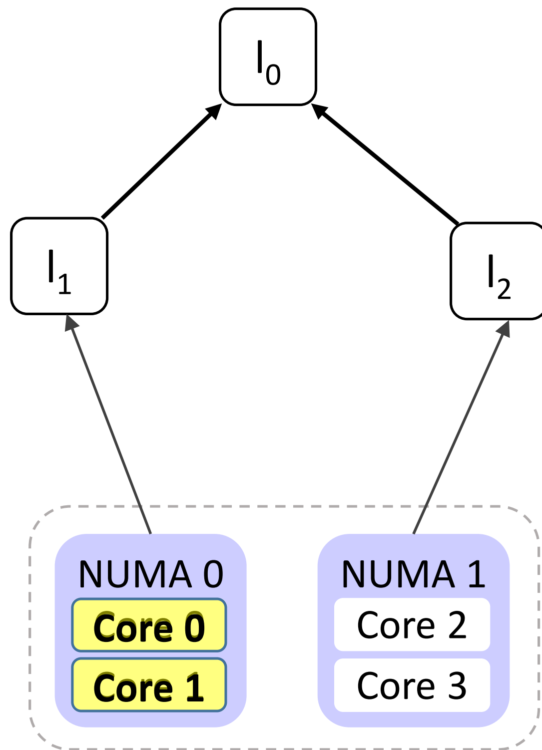
CLoF Lock Generator

Two NUMA node example



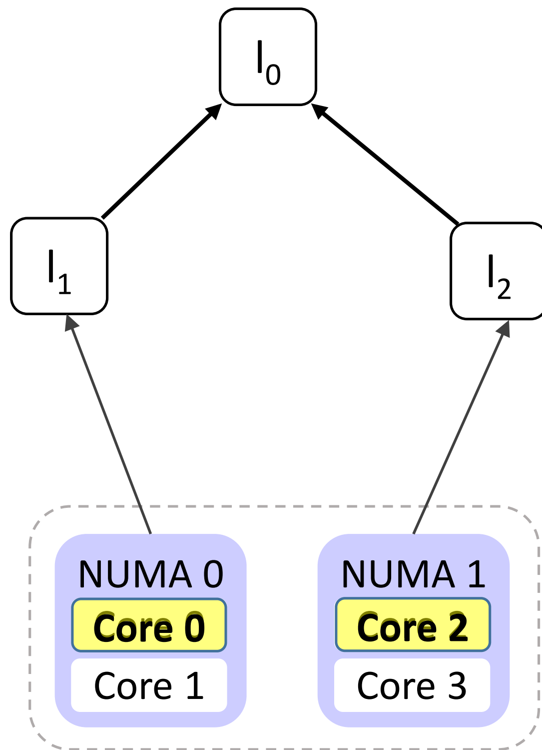
CLoF Lock Generator

Two NUMA node example



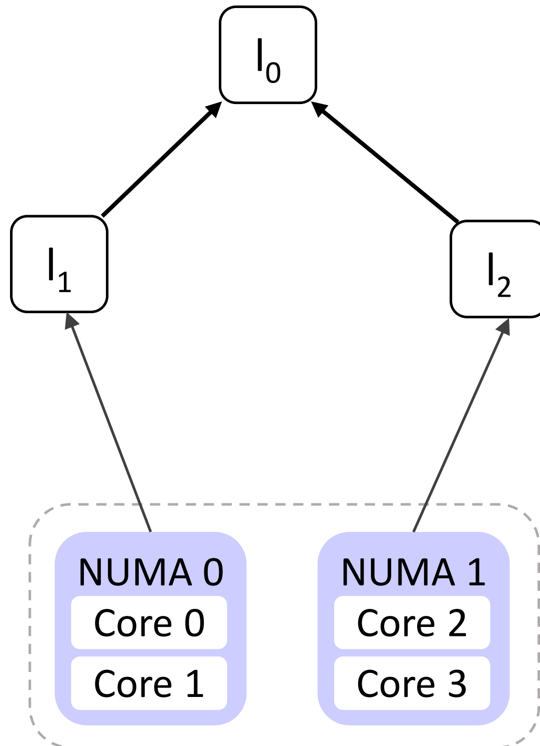
CLoF Lock Generator

Two NUMA node example



CLoF Lock Generator

Two NUMA node example



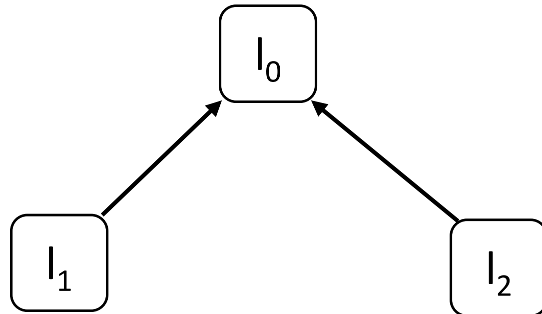
Simplified Code

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

CLoF(*l*, *L*)::release =
if(someone is waiting **and**
others won't starve)
release *l*;
else
release *L*;
release *l*;

CLoF Lock Generator

Compile-Time Syntactic Recursion



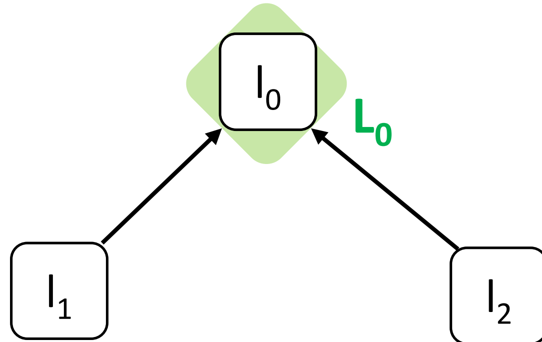
Simplified Code

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

CLoF(*l*, *L*)::release =
  if(someone is waiting **and**
    others won't starve)
    release *l*;
  else
    release *L*;
    release *l*;

CLoF Lock Generator

Compile-Time Syntactic Recursion



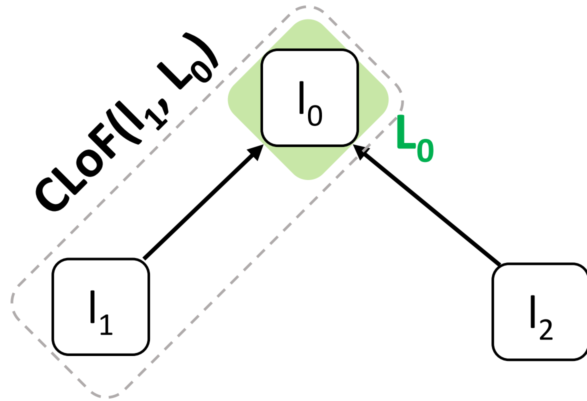
Simplified Code

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

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

CLoF Lock Generator

Compile-Time Syntactic Recursion



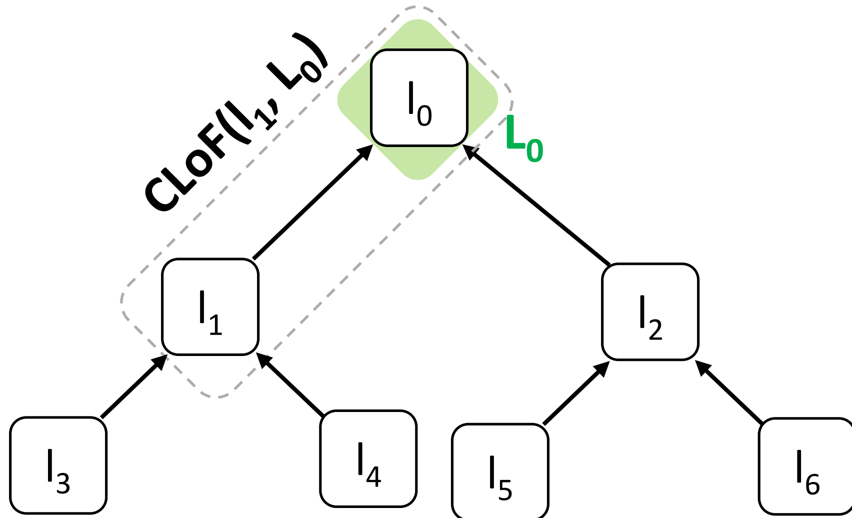
Simplified Code

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

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

CLoF Lock Generator

Compile-Time Syntactic Recursion



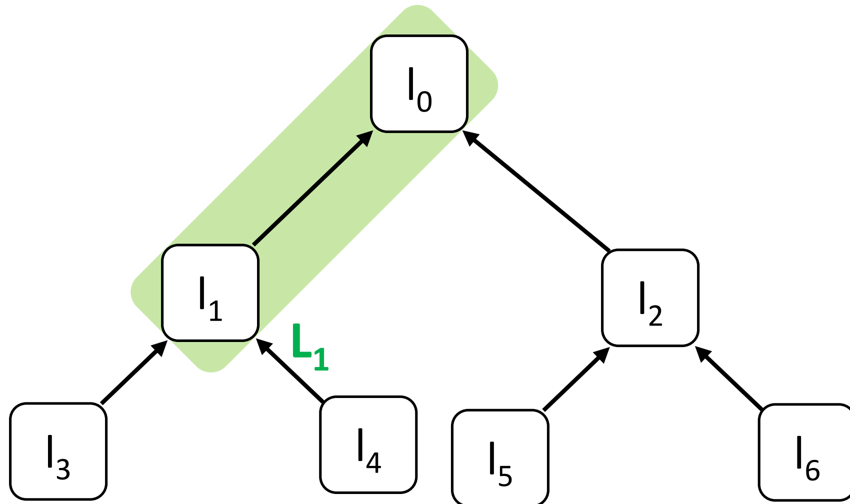
Simplified Code

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

CLoF(*l*, *L*)::release =
if(someone is waiting **and** others won't starve)
release *l*;
else
release *L*;
release *l*;

CLoF Lock Generator

Compile-Time Syntactic Recursion



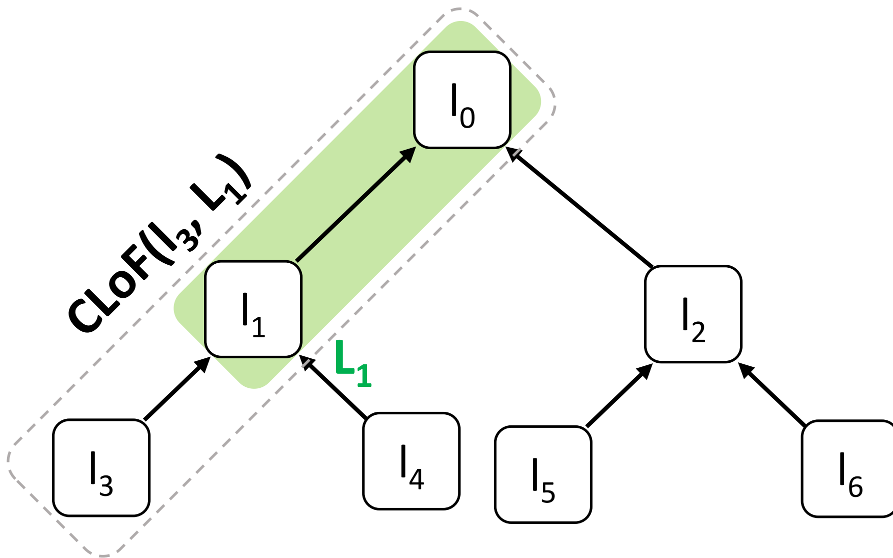
Simplified Code

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

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

CLoF Lock Generator

Compile-Time Syntactic Recursion



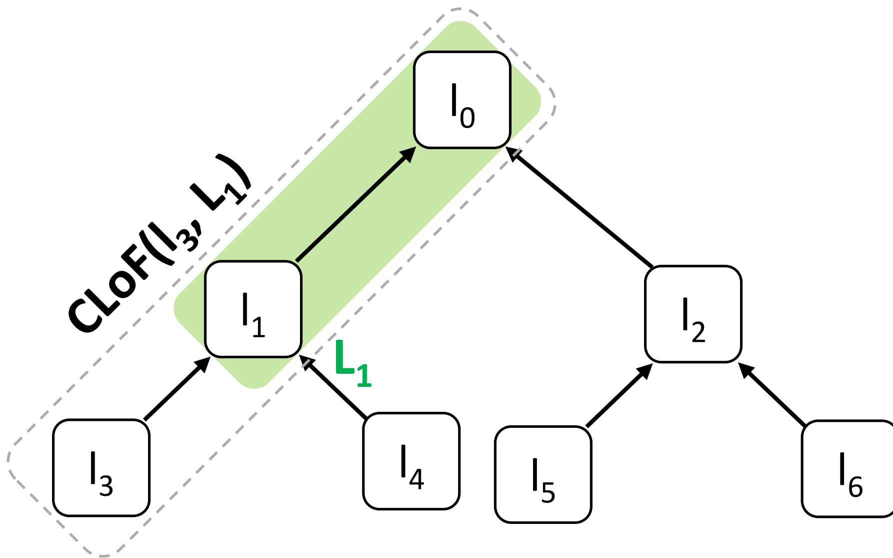
Simplified Code

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

CLoF(*l*, *L*)::release =
if(someone is waiting **and** others won't starve)
release *l*;
else
release *L*;
release *l*;

CLoF Lock Generator

Compile-Time Syntactic Recursion



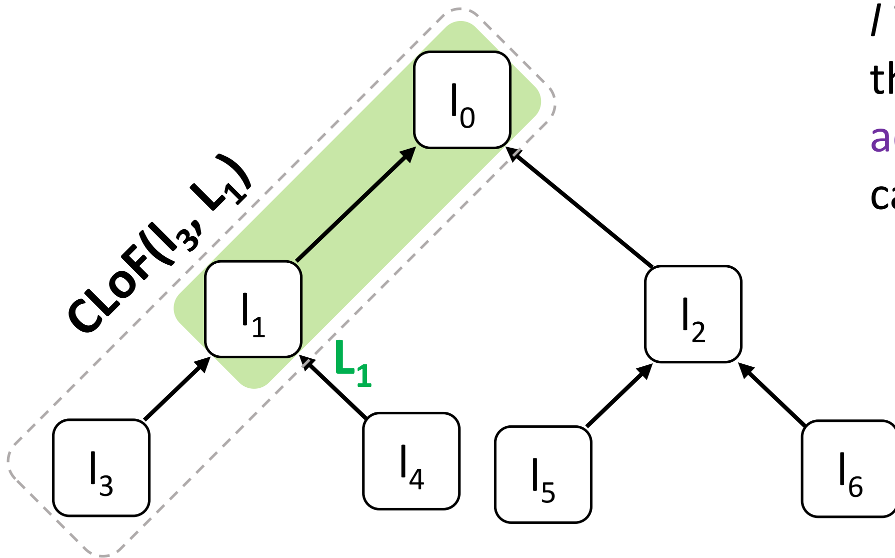
Lock Abstraction

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

CLoF(*l*, *L*)::release =
if(someone is waiting **and** others won't starve)
release *l*;
else
release *L*;
release *l*;

CLoF Lock Generator

Compile-Time Syntactic Recursion



l is only accessed through acquire/release calls

Lock Abstraction

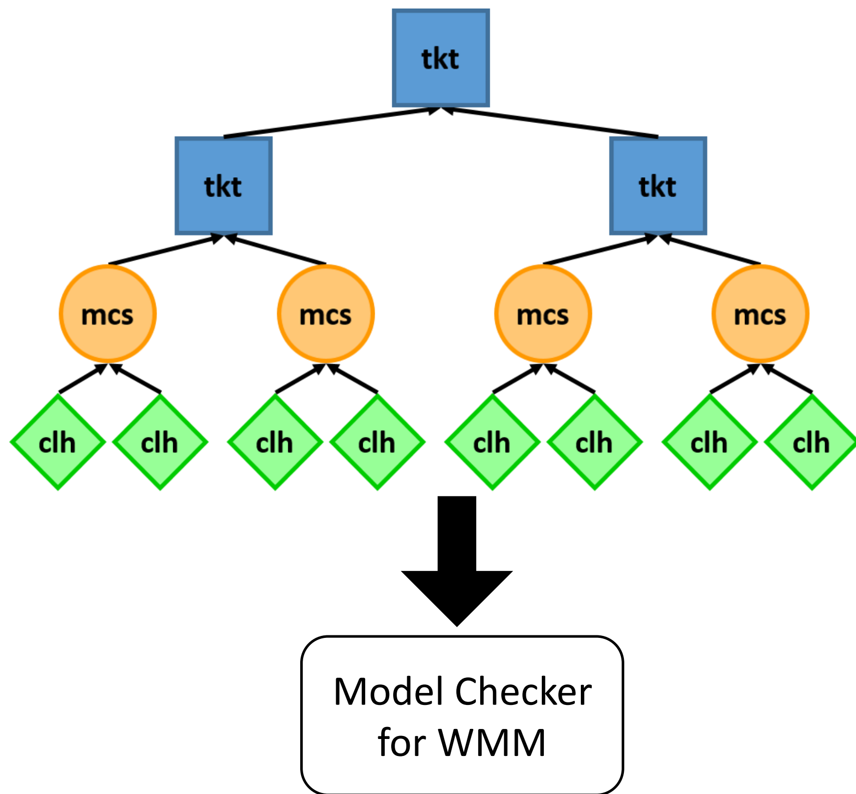
CLoF(*l*, *L*)::acquire =
acquire *l*;
if(¬already has *L*)
acquire *L*;

CLoF(*l*, *L*)::release =
if(someone is waiting **and** others won't starve)
release *l*;
else
release *L*;
release *l*;

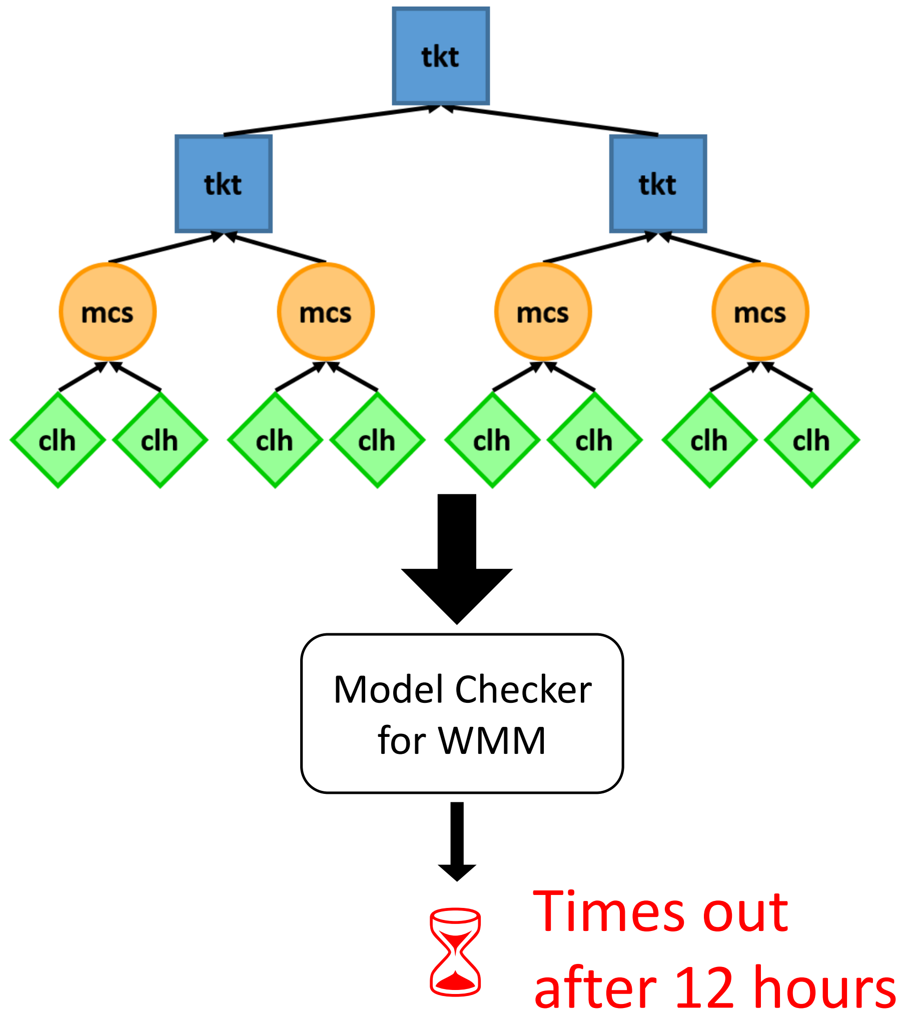
Agenda

- How to figure out the hierarchy we need to use?
- How does our CLoF Lock Generator works?
- How do we know it is correct?
- How do we pick the best lock for the target platform?

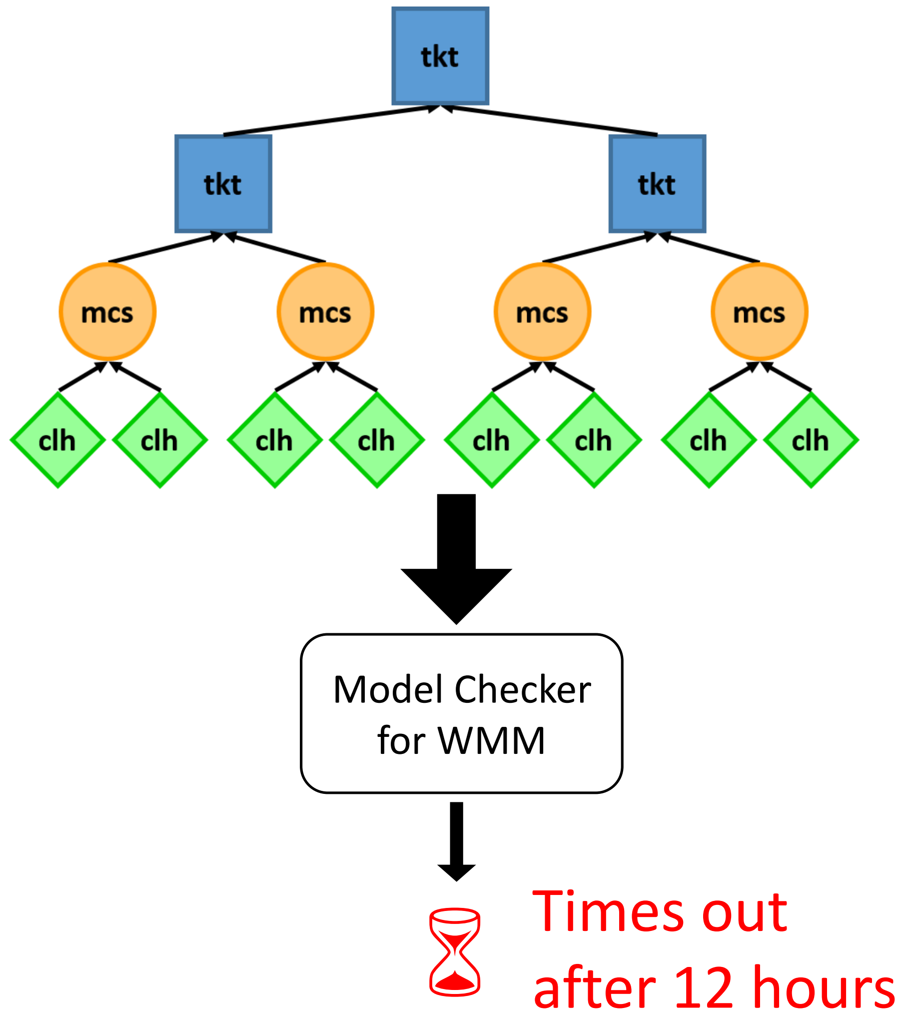
CLoF Correctness



CLoF Correctness



CLoF Correctness

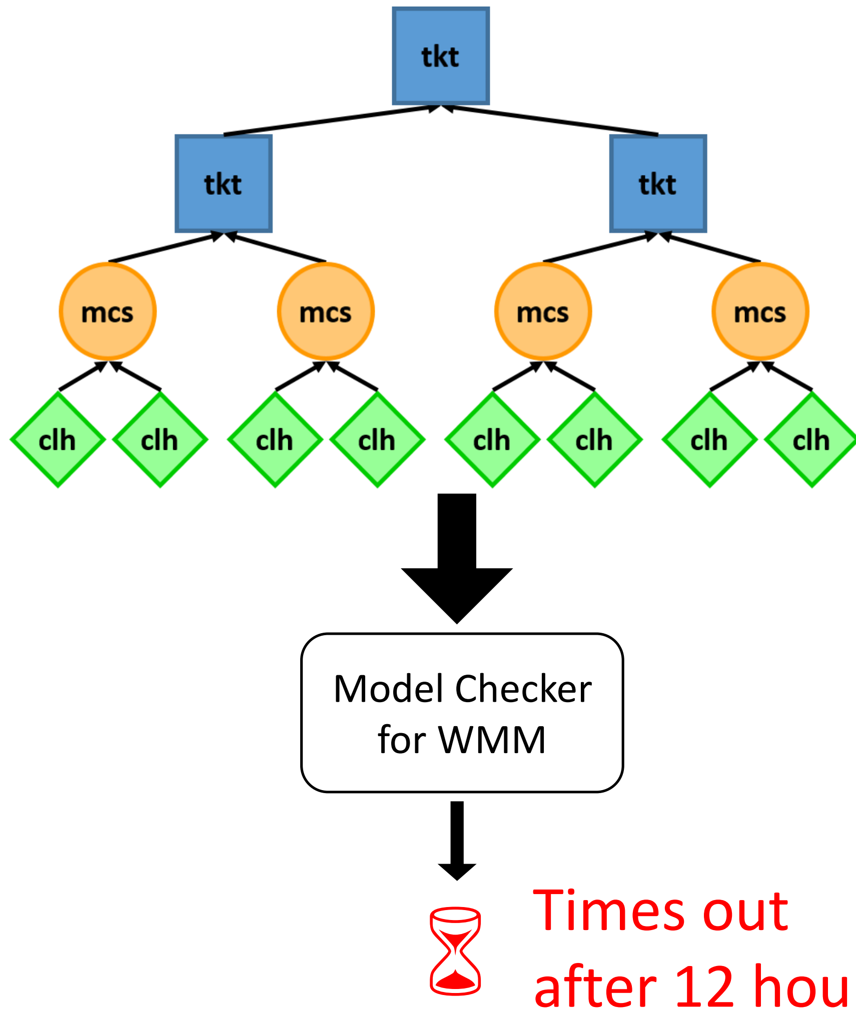


Inductive Approach

Base Step

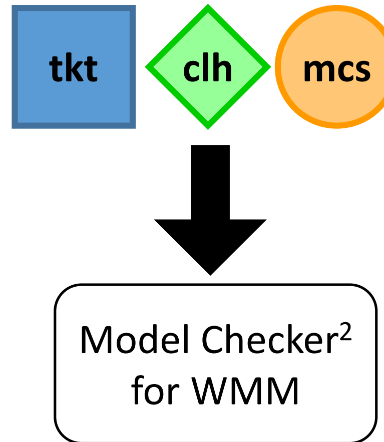
Inductive Step

CLoF Correctness



Inductive Approach

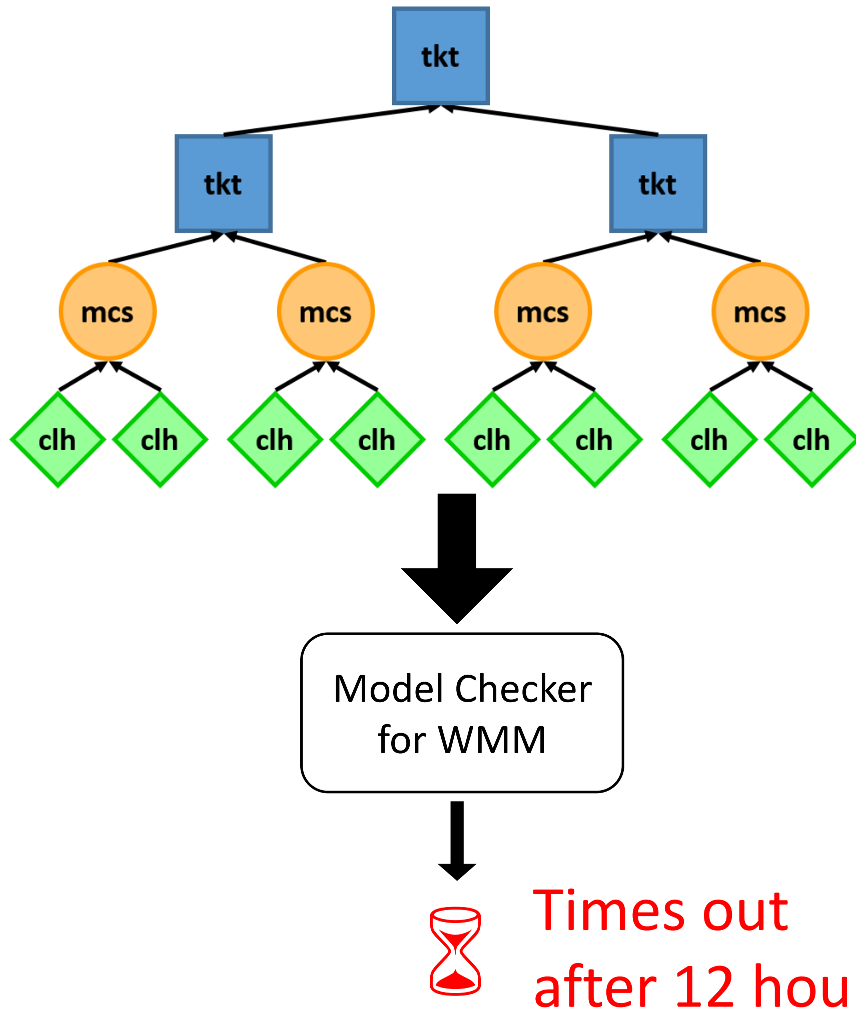
Base Step



Inductive Step

2 Oberhauser *et al.*, VSync: push-button verification and optimization for synchronization primitives on weak memory models, ASPLOS'2021

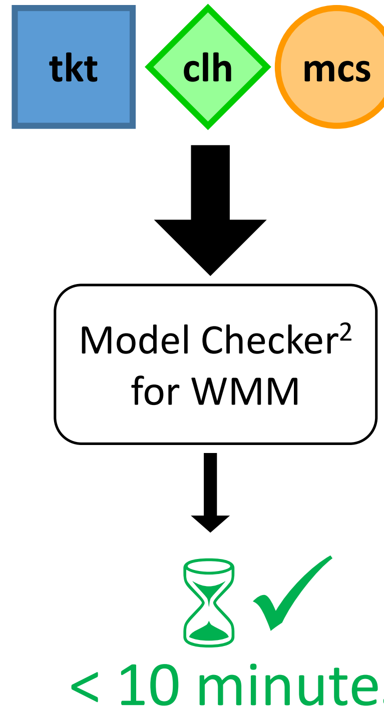
CLoF Correctness



Times out after 12 hours

Inductive Approach

Base Step

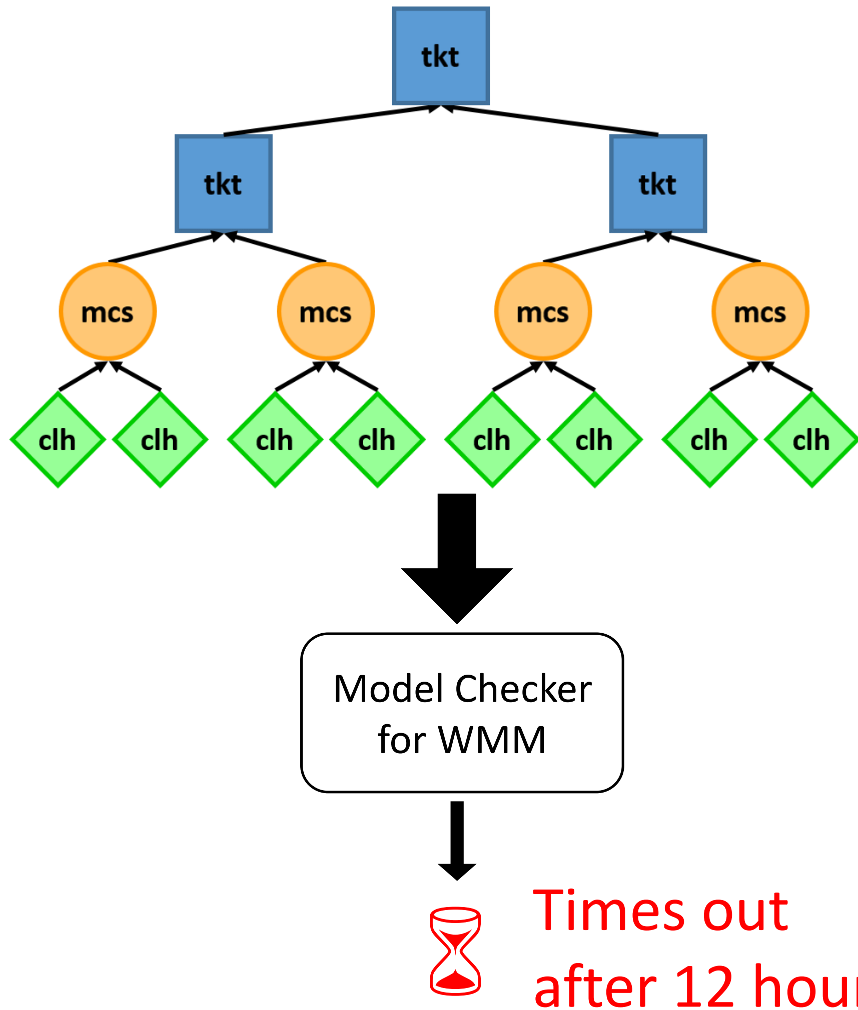


< 10 minutes

Inductive Step

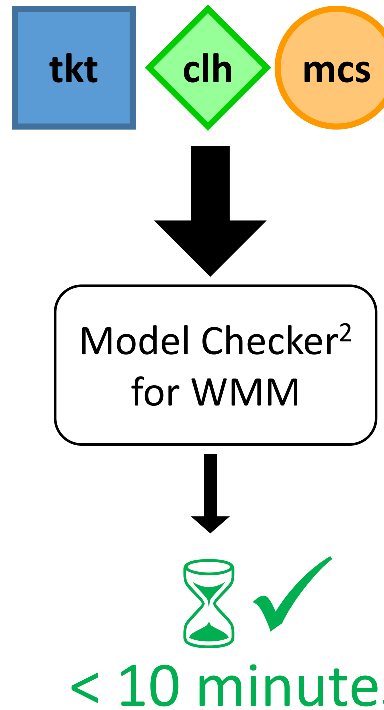
[2] Oberhauser *et al.*, VSync: push-button verification and optimization for synchronization primitives on weak memory models, ASPLOS'2021

CLoF Correctness

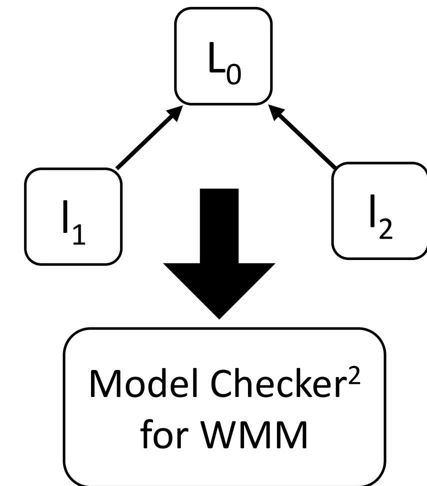


Inductive Approach

Base Step

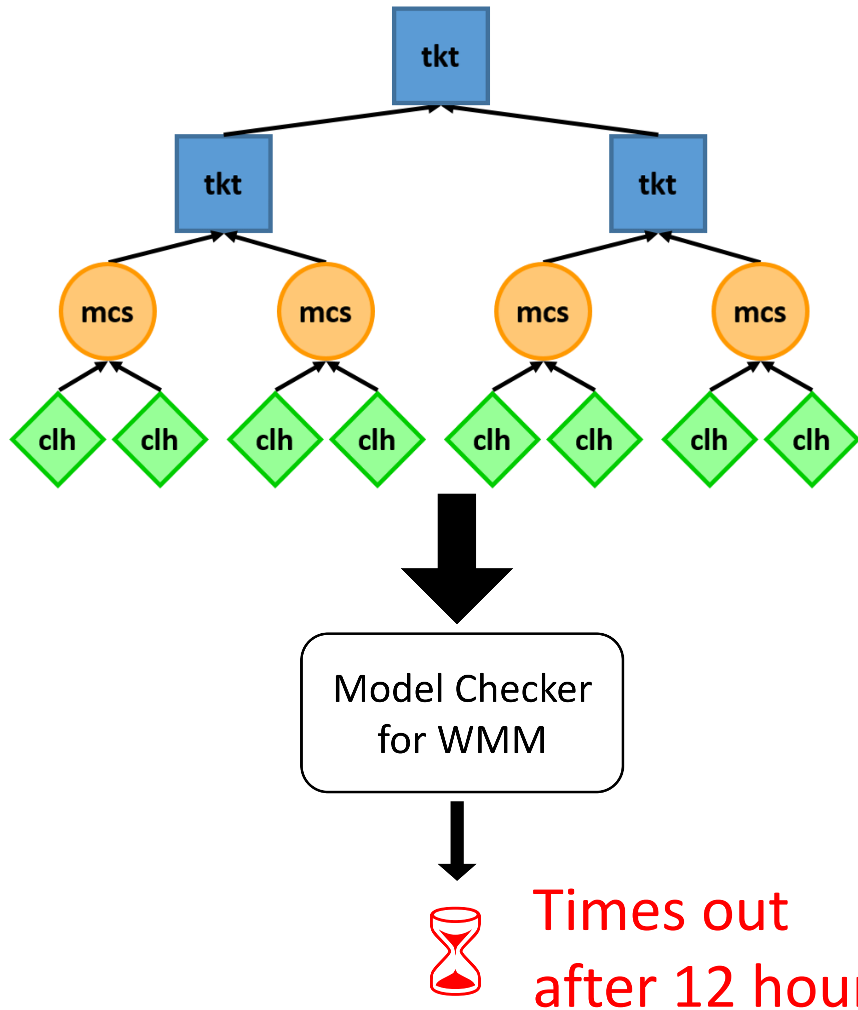


Inductive Step



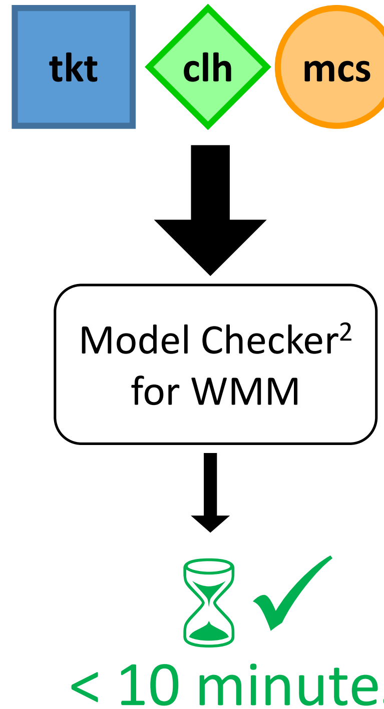
[2] Oberhauser *et al.*, VSync: push-button verification and optimization for synchronization primitives on weak memory models, ASPLOS'2021

CLoF Correctness

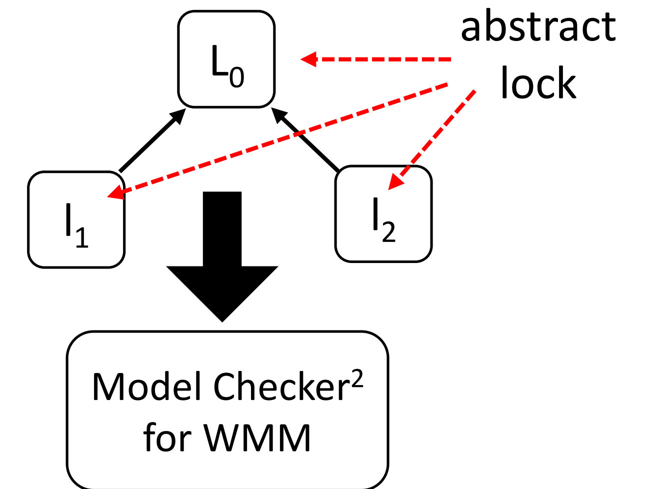


Inductive Approach

Base Step

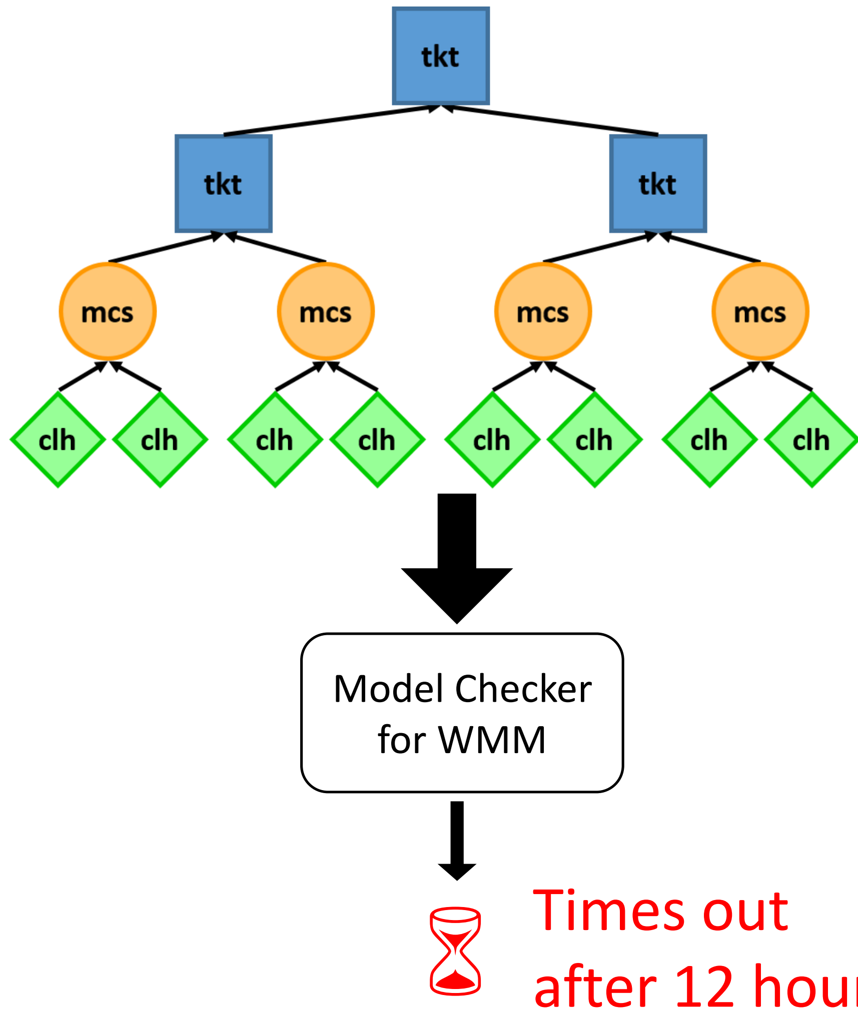


Inductive Step



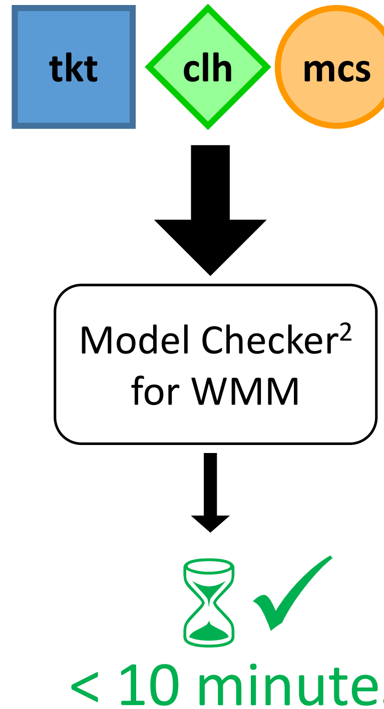
[2] Oberhauser *et al.*, VSync: push-button verification and optimization for synchronization primitives on weak memory models, ASPLOS'2021

CLoF Correctness

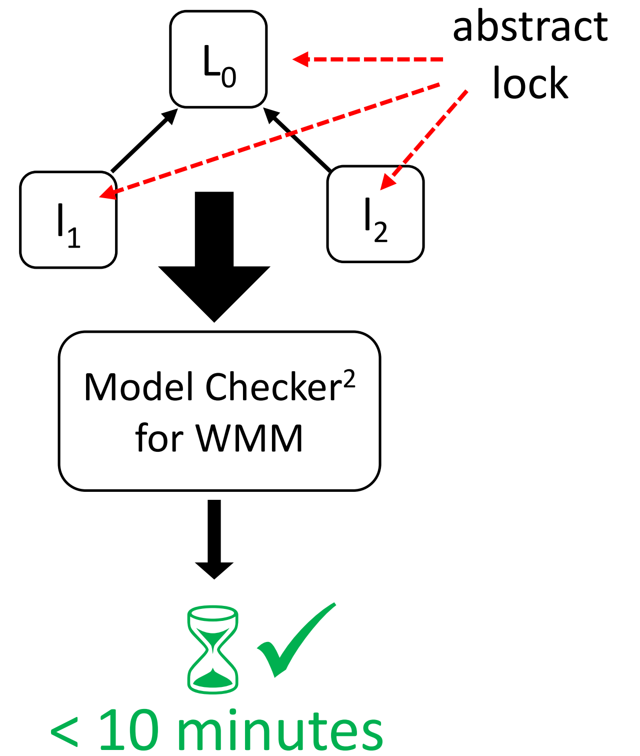


Inductive Approach

Base Step



Inductive Step



[2] Oberhauser *et al.*, VSync: push-button verification and optimization for synchronization primitives on weak memory models, ASPLOS'2021

Agenda

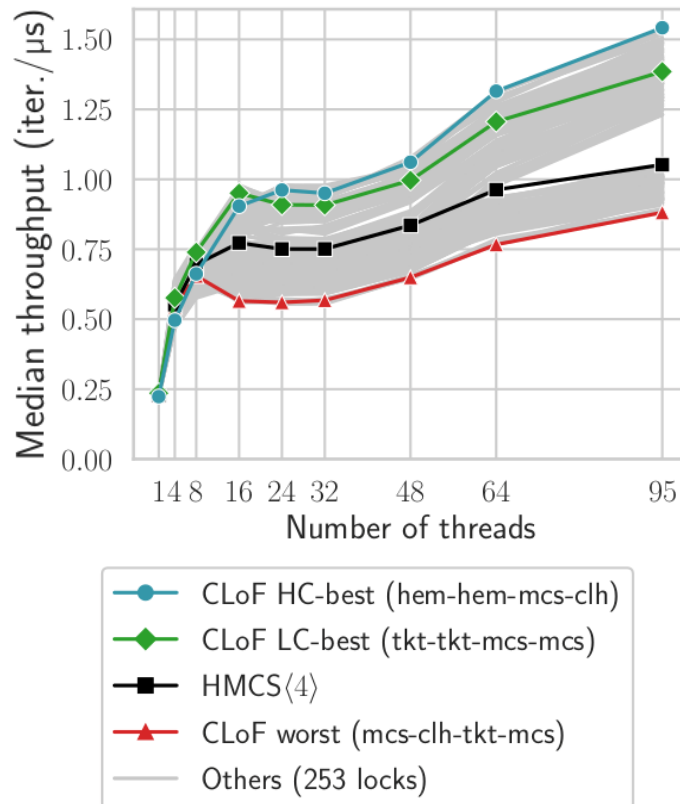
- How to figure out the hierarchy we need to use?
- How does our CLoF Lock Generator works?
- How do we know it is correct?
- How do we pick the best lock for the target platform?

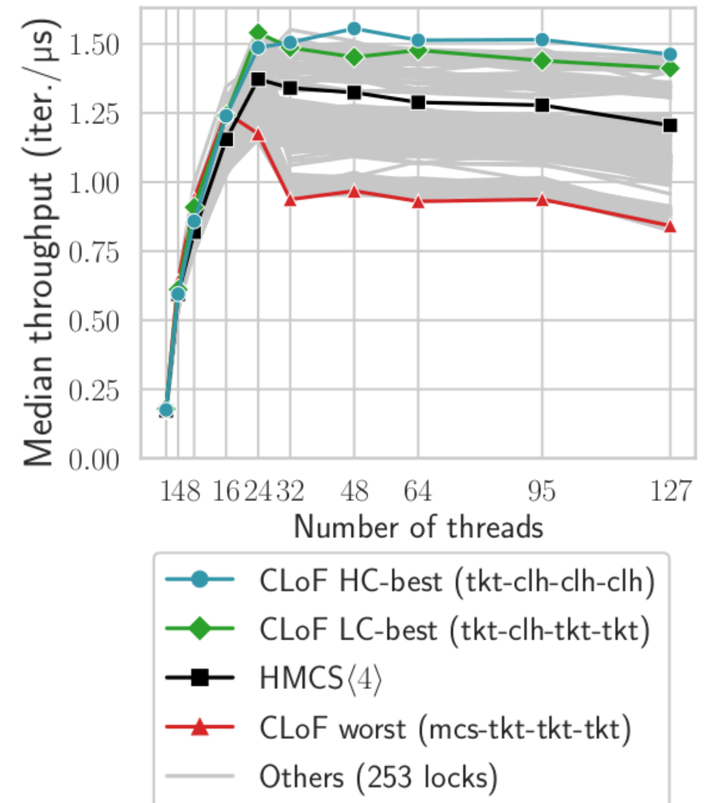
Run Scripted Benchmark

- Run all lock combinations composed by the CLoF lock generator
- For B = 4 locks and L = 4 levels, we have $B^L = 4^4 = 256$ combinations
  - Up to 1 hour in a platform with 128 cores

Run Scripted Benchmark

- Run all lock combinations composed by the CLoF lock generator
- For B = 4 locks and L = 4 levels, we have $B^L = 4^4 = 256$ combinations
  - Up to 1 hour in a platform with 128 cores

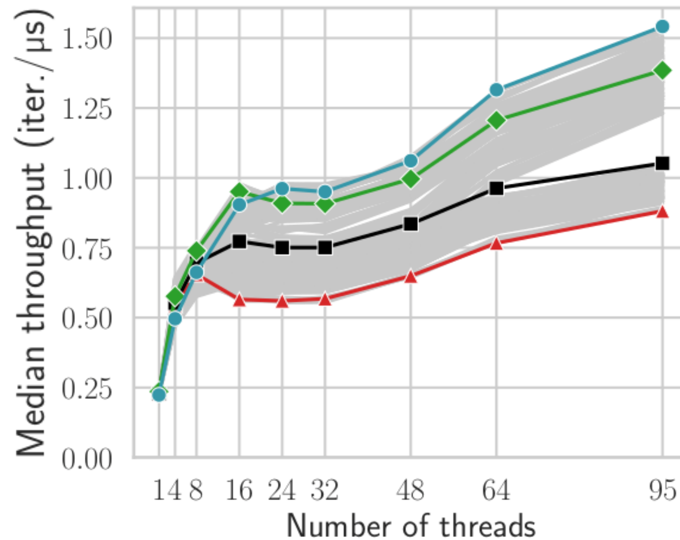


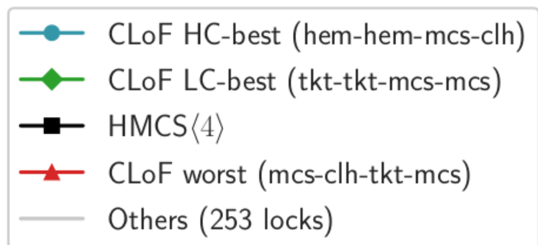


Run Scripted Benchmark

- Run all lock combinations composed by the CLoF lock generator
- For B = 4 locks and L = 4 levels, we have $B^L = 4^4 = 256$ combinations
  - Up to 1 hour in a platform with 128 cores

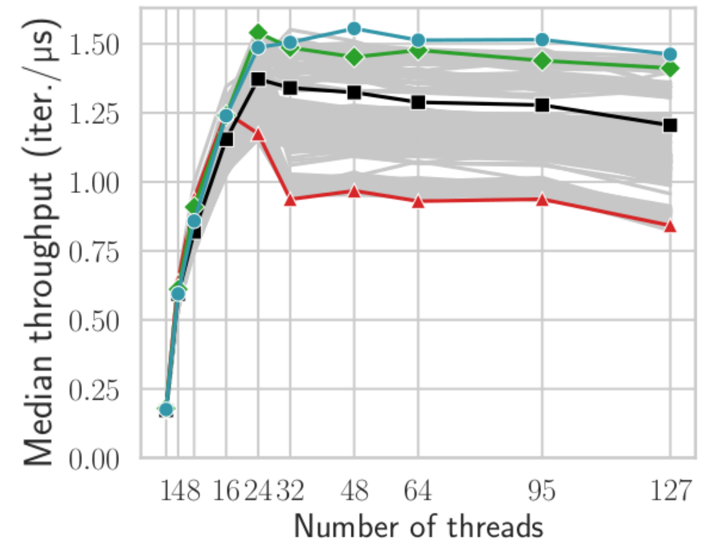
x86 server – levelDB readrandom benchmark

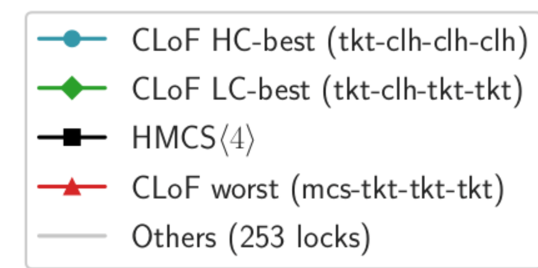




No combination is better for all contention levels

arm server – levelDB readrandom benchmark

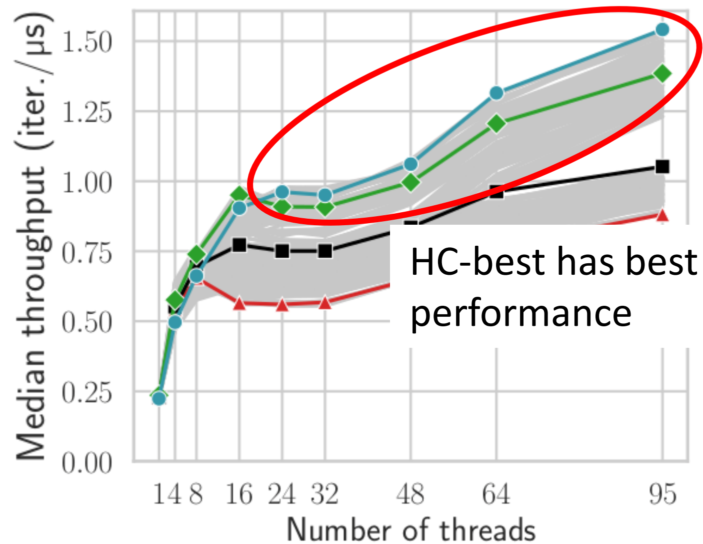


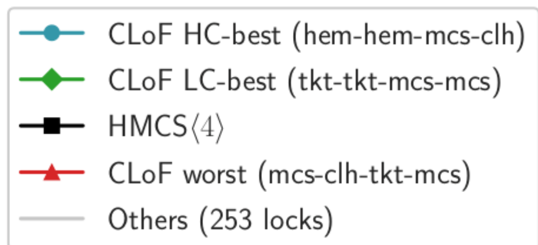


Run Scripted Benchmark

- Run all lock combinations composed by the CLoF lock generator
- For B = 4 locks and L = 4 levels, we have $B^L = 4^4 = 256$ combinations
  - Up to 1 hour in a platform with 128 cores

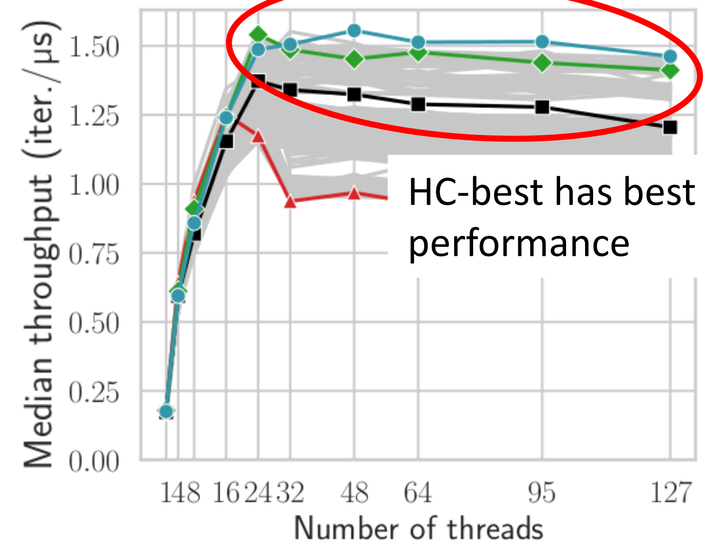
x86 server – levelDB readrandom benchmark

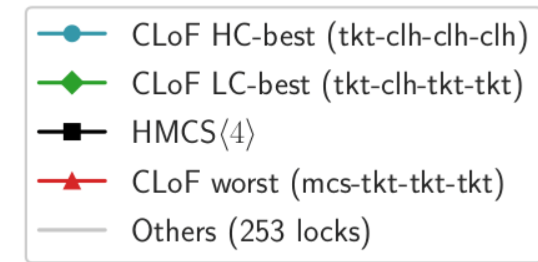




No combination is better for all contention levels

arm server – levelDB readrandom benchmark

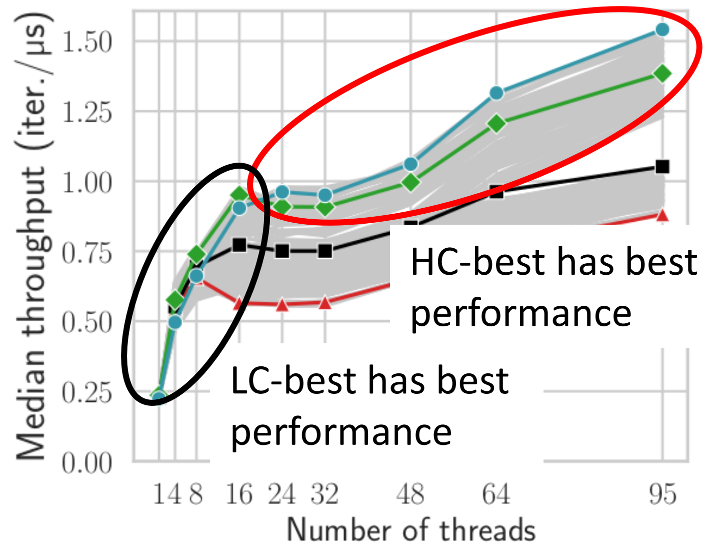


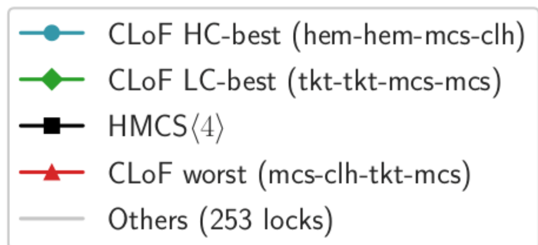


Run Scripted Benchmark

- Run all lock combinations composed by the CLoF lock generator
- For B = 4 locks and L = 4 levels, we have $B^L = 4^4 = 256$ combinations
  - Up to 1 hour in a platform with 128 cores

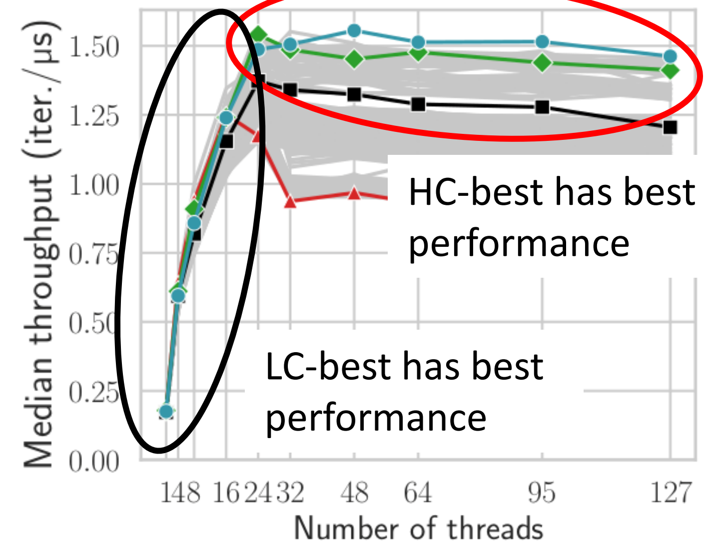
x86 server – levelDB readrandom benchmark

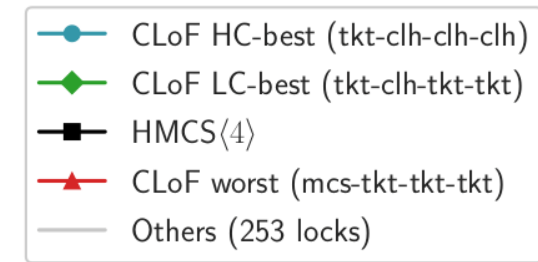




No combination is better for all contention levels

arm server – levelDB readrandom benchmark

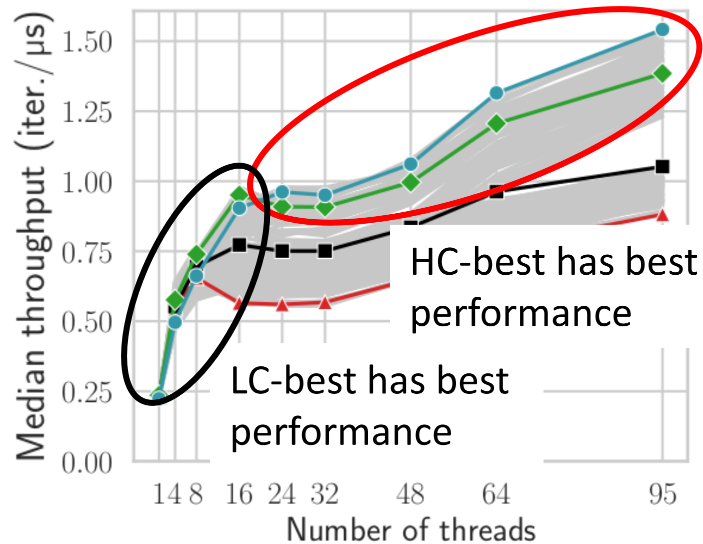




Run Scripted Benchmark

- Run all lock combinations composed by the CLoF lock generator
- For B = 4 locks and L = 4 levels, we have $B^L = 4^4 = 256$ combinations
  - Up to 1 hour in a platform with 128 cores

x86 server – levelDB readrandom benchmark

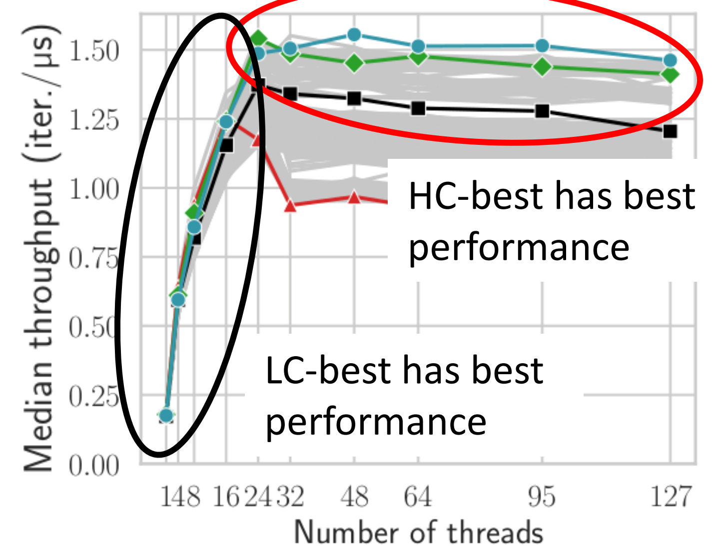


- CLoF HC-best (hem-hem-mcs-clh)
- CLoF LC-best (tkt-tkt-mcs-mcs)
- HMCS⟨4⟩
- CLoF worst (mcs-clh-tkt-mcs)
- Others (253 locks)

No combination is better for all contention levels

Which one should be chosen?

arm server – levelDB readrandom benchmark

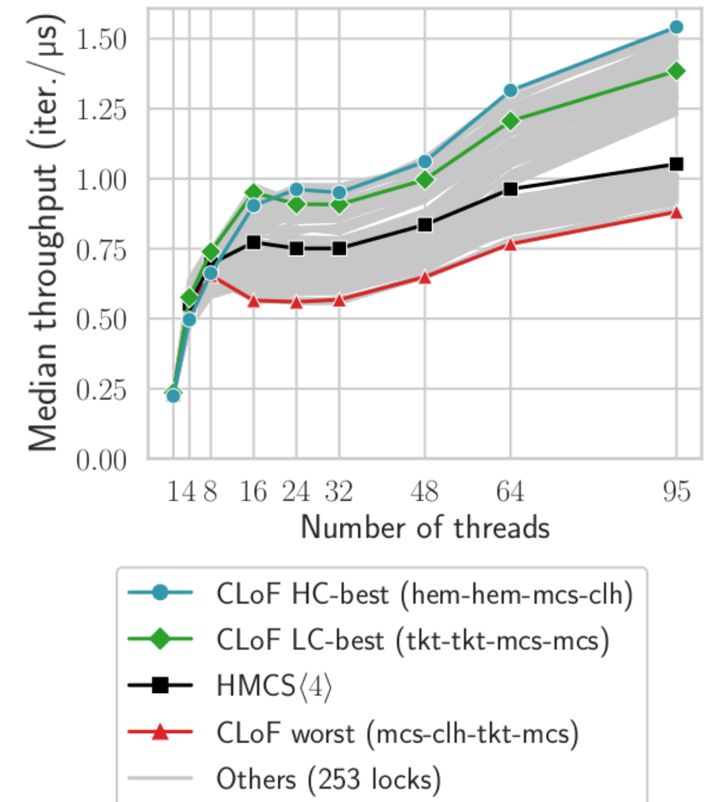


- CLoF HC-best (tkt-clh-clh-clh)
- CLoF LC-best (tkt-clh-tkt-tkt)
- HMCS⟨4⟩
- CLoF worst (mcs-tkt-tkt-tkt)
- Others (253 locks)

Tuning Point: Choose Selection Policy

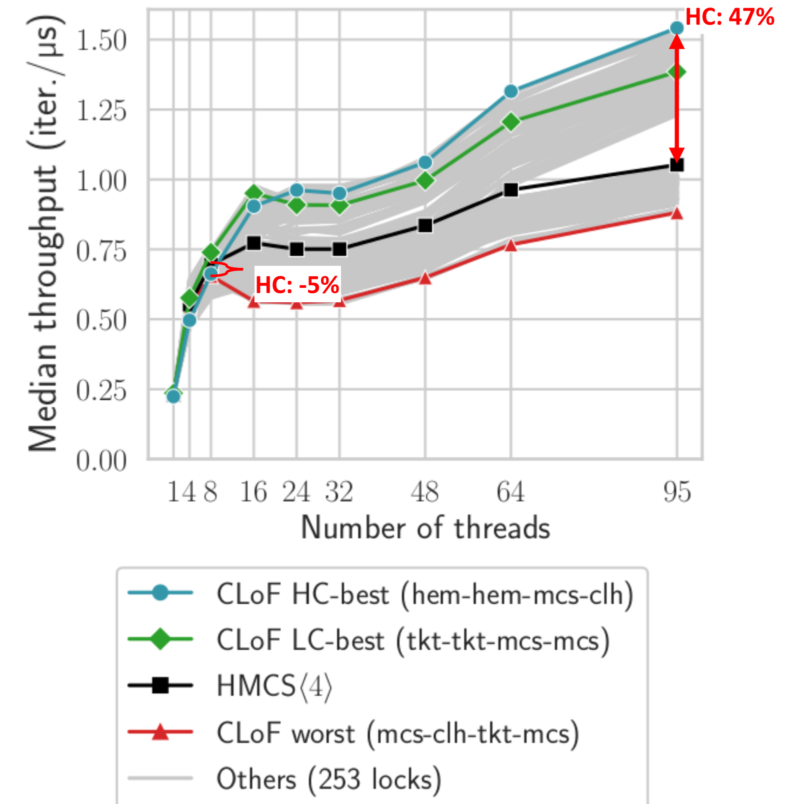
- CLoF stablishes 2 selection policies for the best lock:
  - HC-best prioritizes performance at high contention
  - LC-best prioritizes performance at low contention

x86 server – levelDB readrandom benchmark



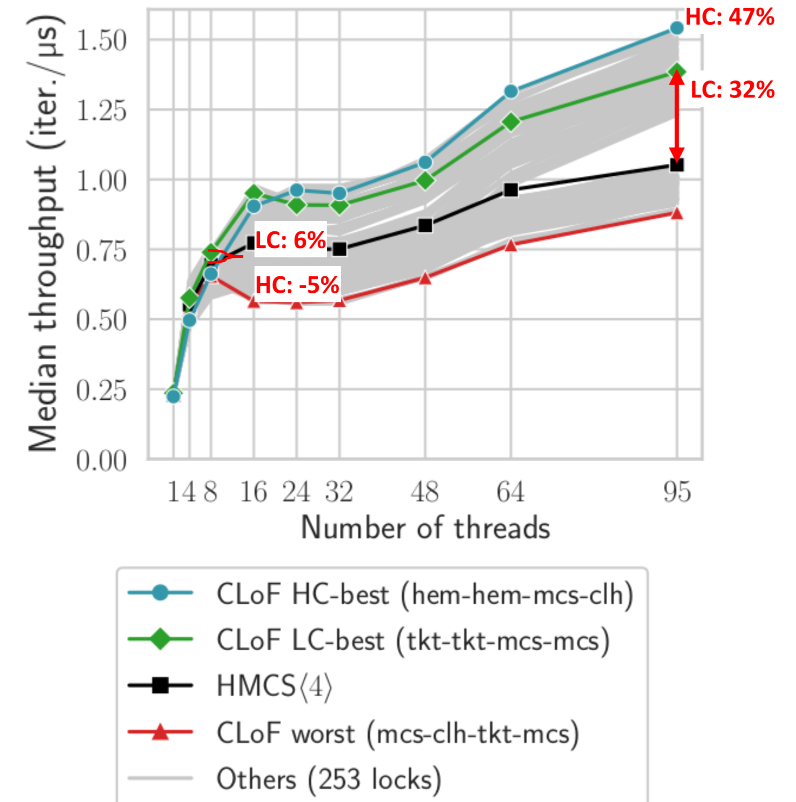
Tuning Point: Choose Selection Policy

- CLoF stablishes 2 selection policies for the best lock:
  - HC-best prioritizes performance at high contention
  - LC-best prioritizes performance at low contention



Tuning Point: Choose Selection Policy

- CLoF stablishes 2 selection policies for the best lock:
  - HC-best prioritizes performance at high contention
  - LC-best prioritizes performance at low contention

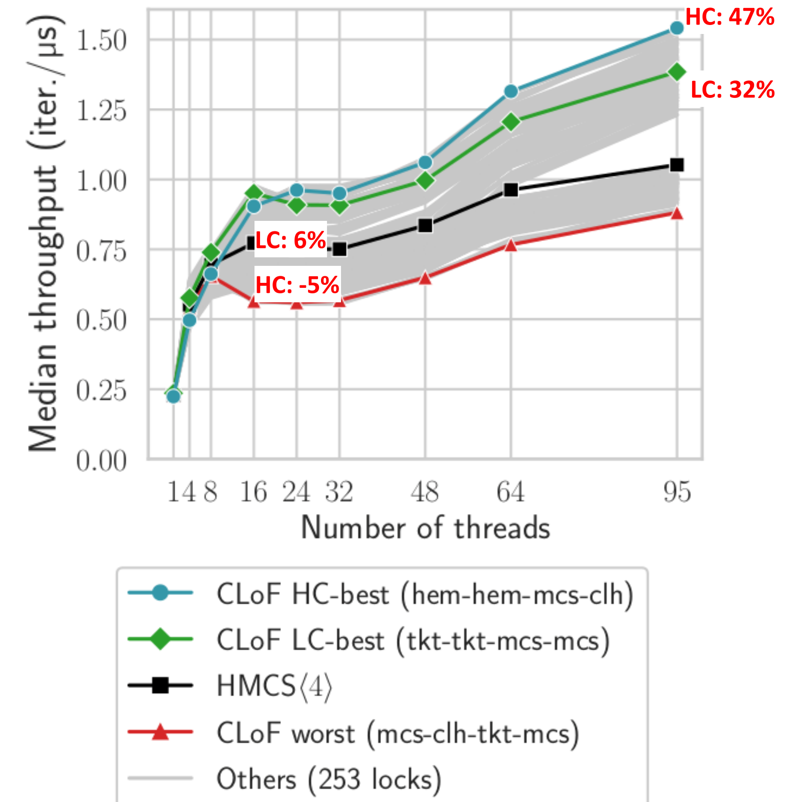


Tuning Point: Choose Selection Policy

- CLoF stablishes 2 selection policies for the best lock:
  - HC-best prioritizes performance at high contention
  - LC-best prioritizes performance at low contention

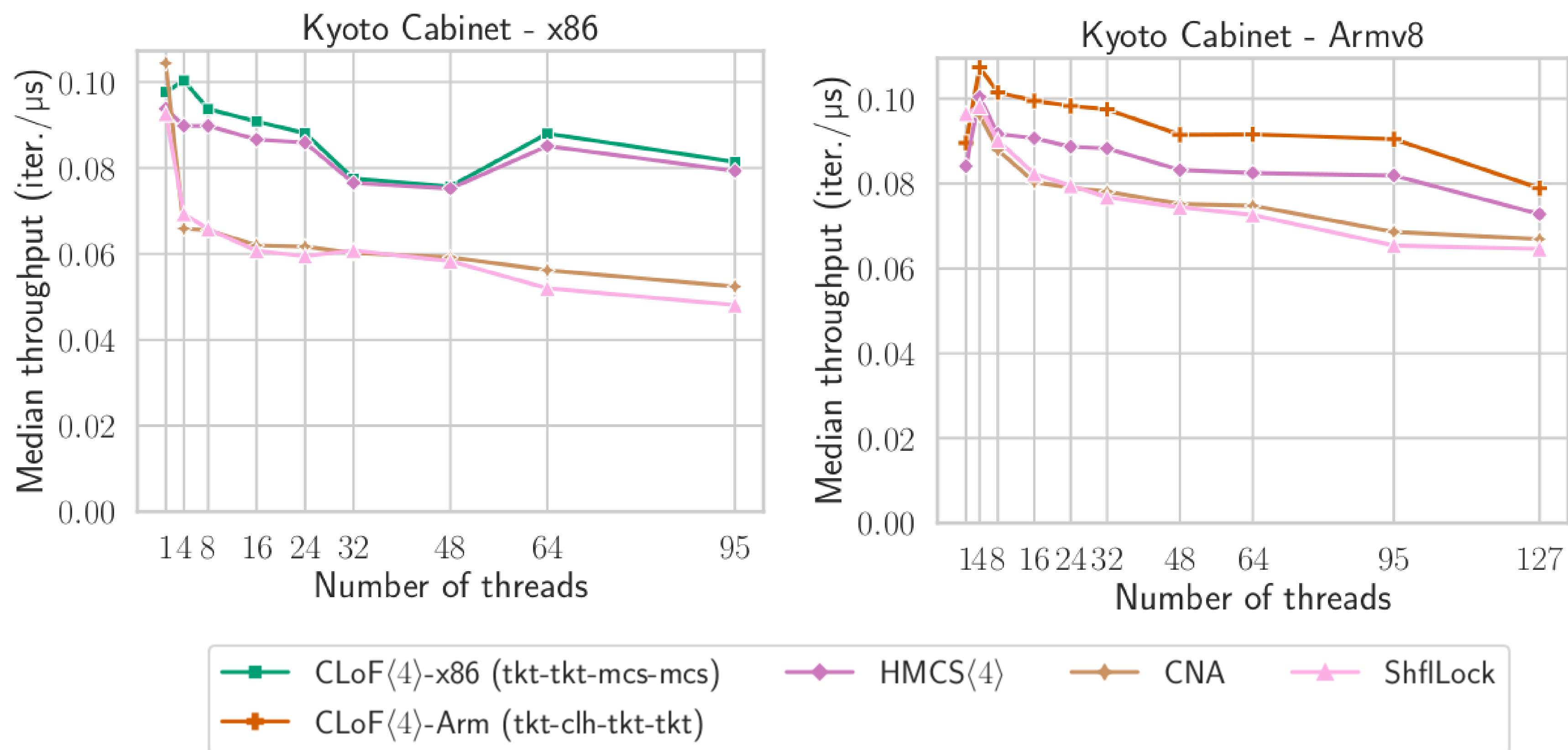
User can tune which selection policy is desired

- Without the need to re-run the benchmark



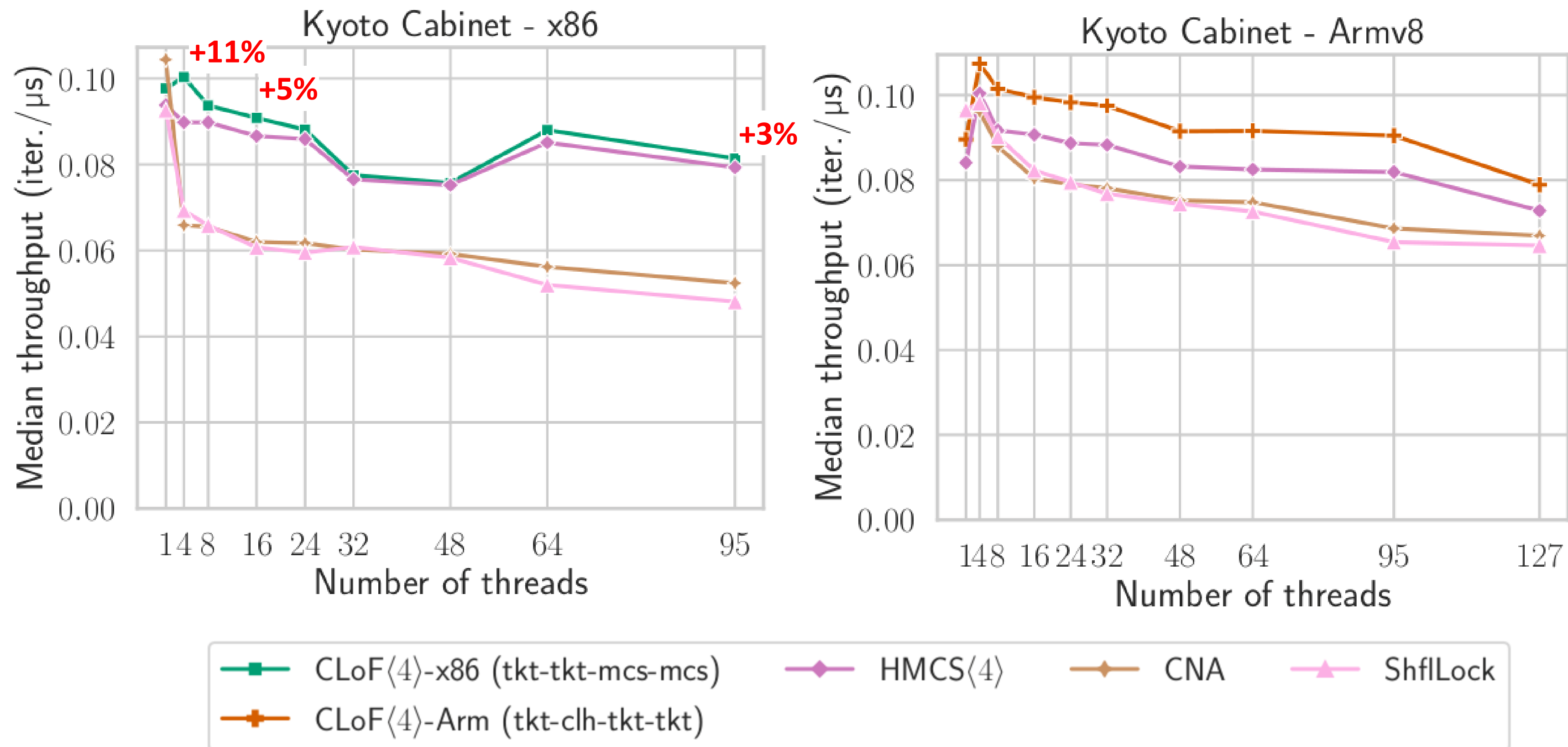
Evaluation

- Another benchmark, called Kyoto Cabinet, is used to cross-validate results
  - Display LC-best



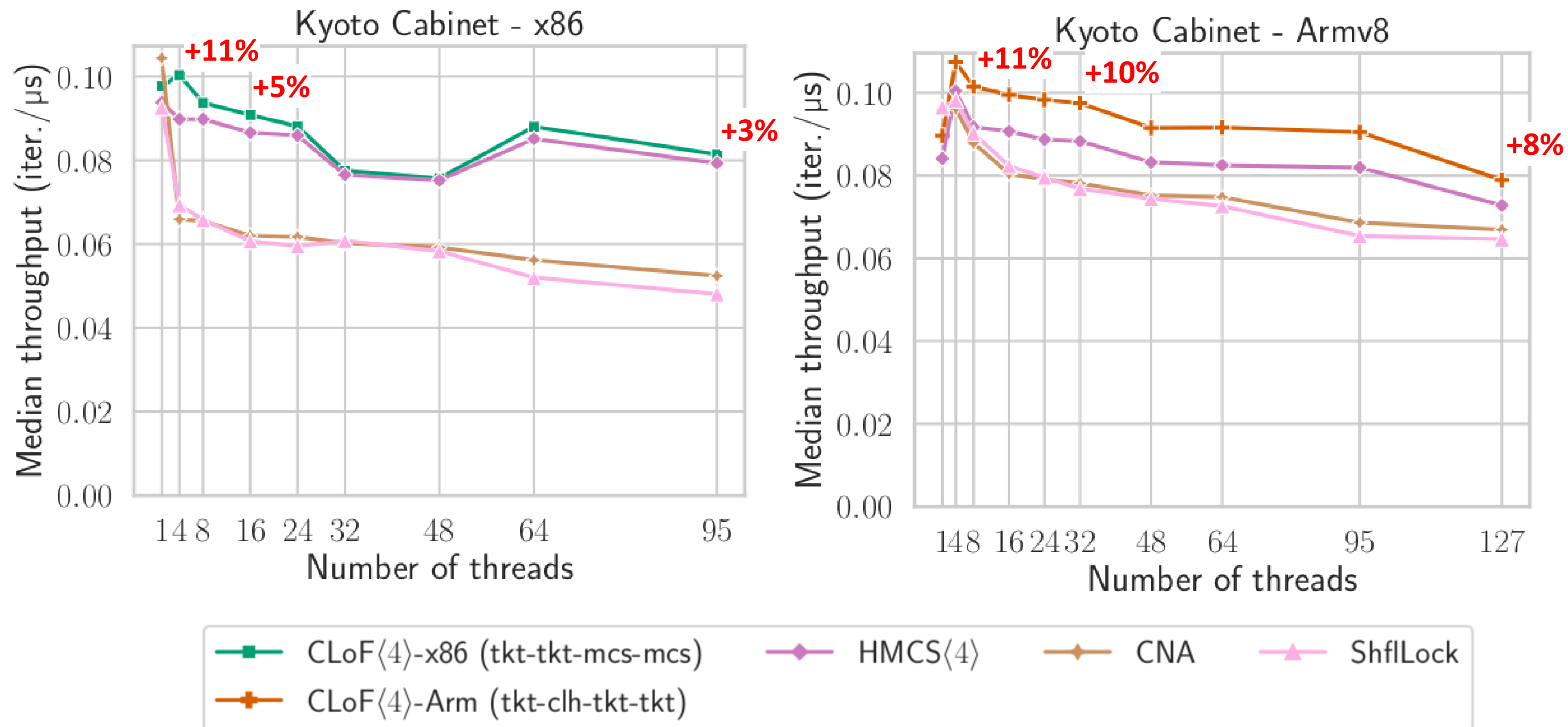
Evaluation

- Another benchmark, called Kyoto Cabinet, is used to cross-validate results
  - Display LC-best



Evaluation

- Another benchmark, called Kyoto Cabinet, is used to cross-validate results
  - Display LC-best

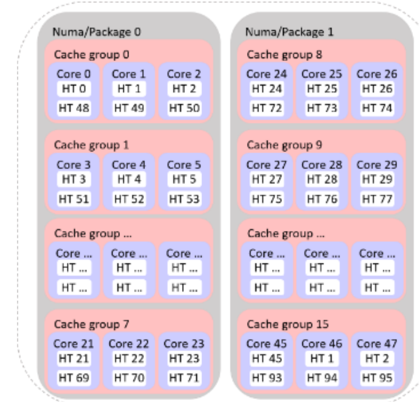


Conclusion and Future Work

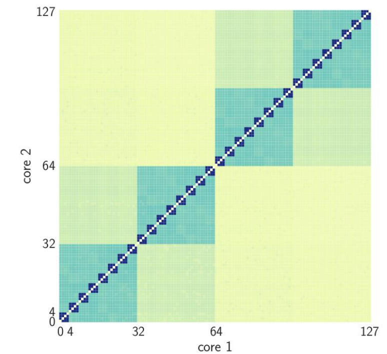
- CLoF locks
  - fully leverage deep hierarchy
  - are level heterogeneity
  - can be optimized for target platform
  - are correct-by-construction on Weak Memory Models
- Don't miss the details!
  - platform-specific optimizations
  - analysis of lock combinations
  - …
- Future work
  - CLoF in the Linux kernel
  - big.LITTLE platforms

Thanks!

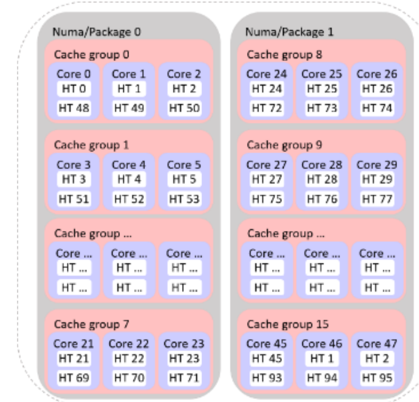
Tuning Point: Choosing the Hierarchy Levels



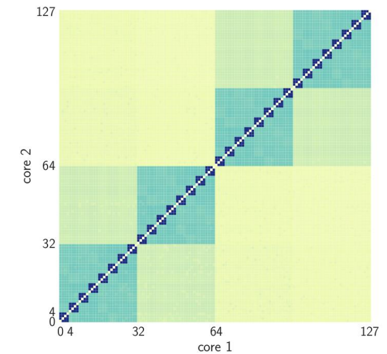
- Not all levels will always be used
  - Application can disable hyperthreads – x86 server
  - Some levels may have small improvement – package level on Kunpeng 920



Tuning Point: Choosing the Hierarchy Levels



- Not all levels will always be used
  - Application can disable hyperthreads – x86 server
  - Some levels may have small improvement – package level on Kunpeng 920



- User can tune levels that are wanted
  - Include/Remove levels found at discovery