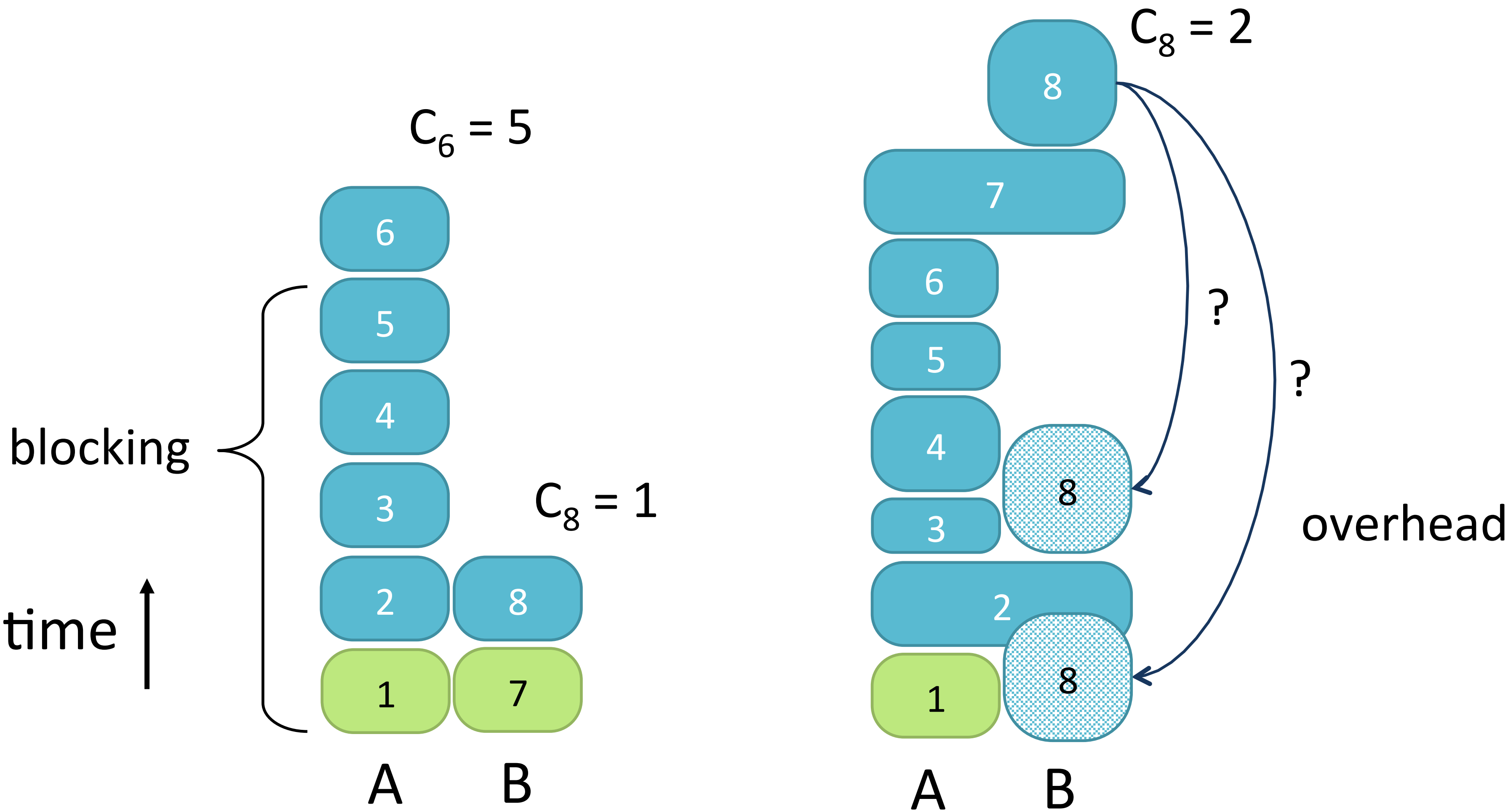
Using Lock Servers to Scale Real-Time Locking Protocols:
Chasing Ever-Increasing Core Counts

CATHERINE E. NEMITZ, TANYA AMERT, JAMES H. ANDERSON

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Contention-Sensitive Access



Using Lock Servers to Scale Real-Time Locking Protocols:

Chasing Ever-Increasing Core Counts

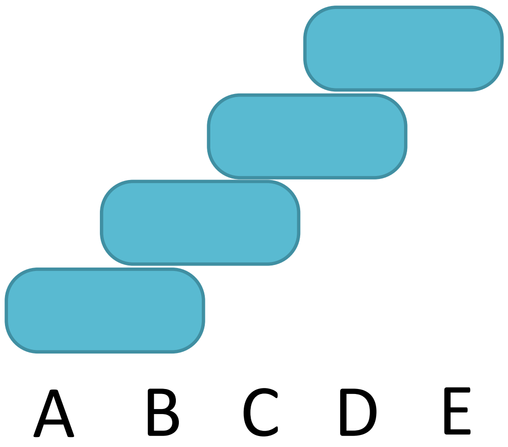
Challenge: Blocking Chains

\# of processors:

m=4

Processing capacity lost:

75%



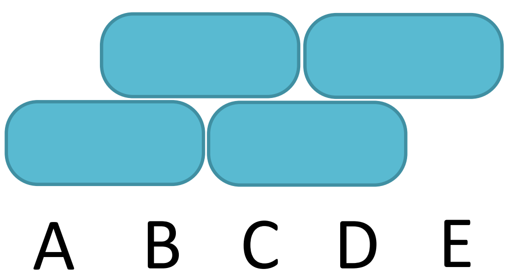
Challenge: Blocking Chains

of processors:

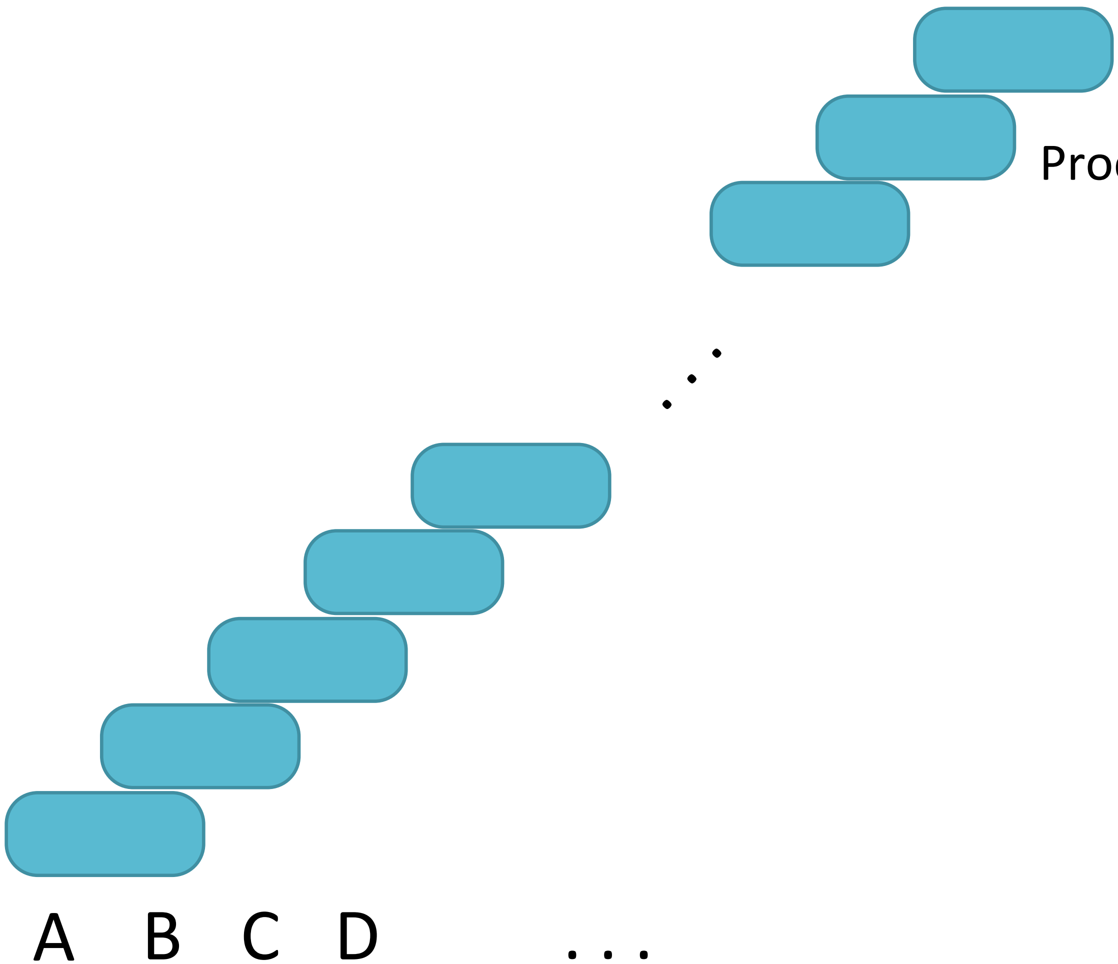
m=4

Processing capacity lost:

75%



Challenge: Blocking Chains



# of processors:

m=32

Processing capacity lost:

97%

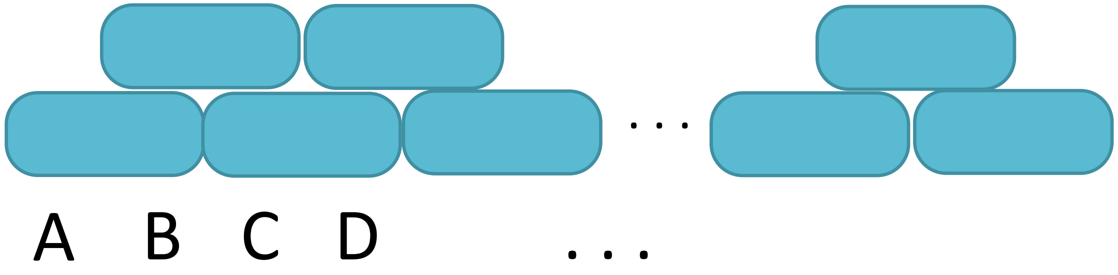
Challenge: Blocking Chains

of processors:

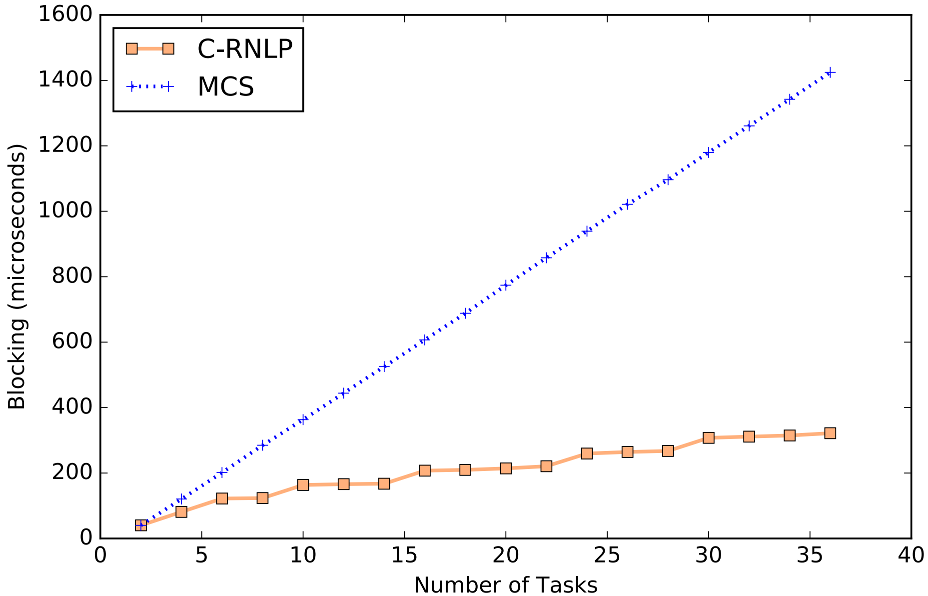
m=32

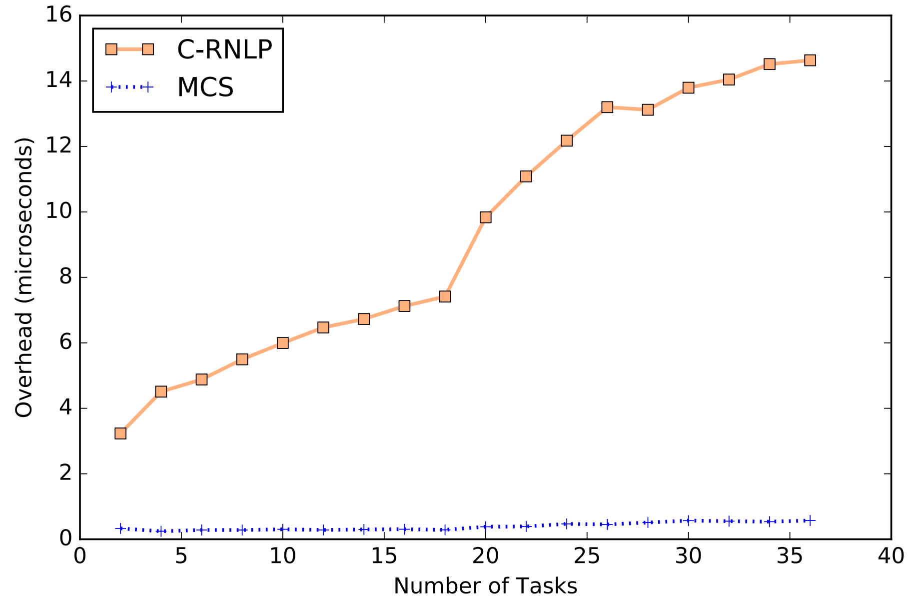
Processing capacity lost:

97%



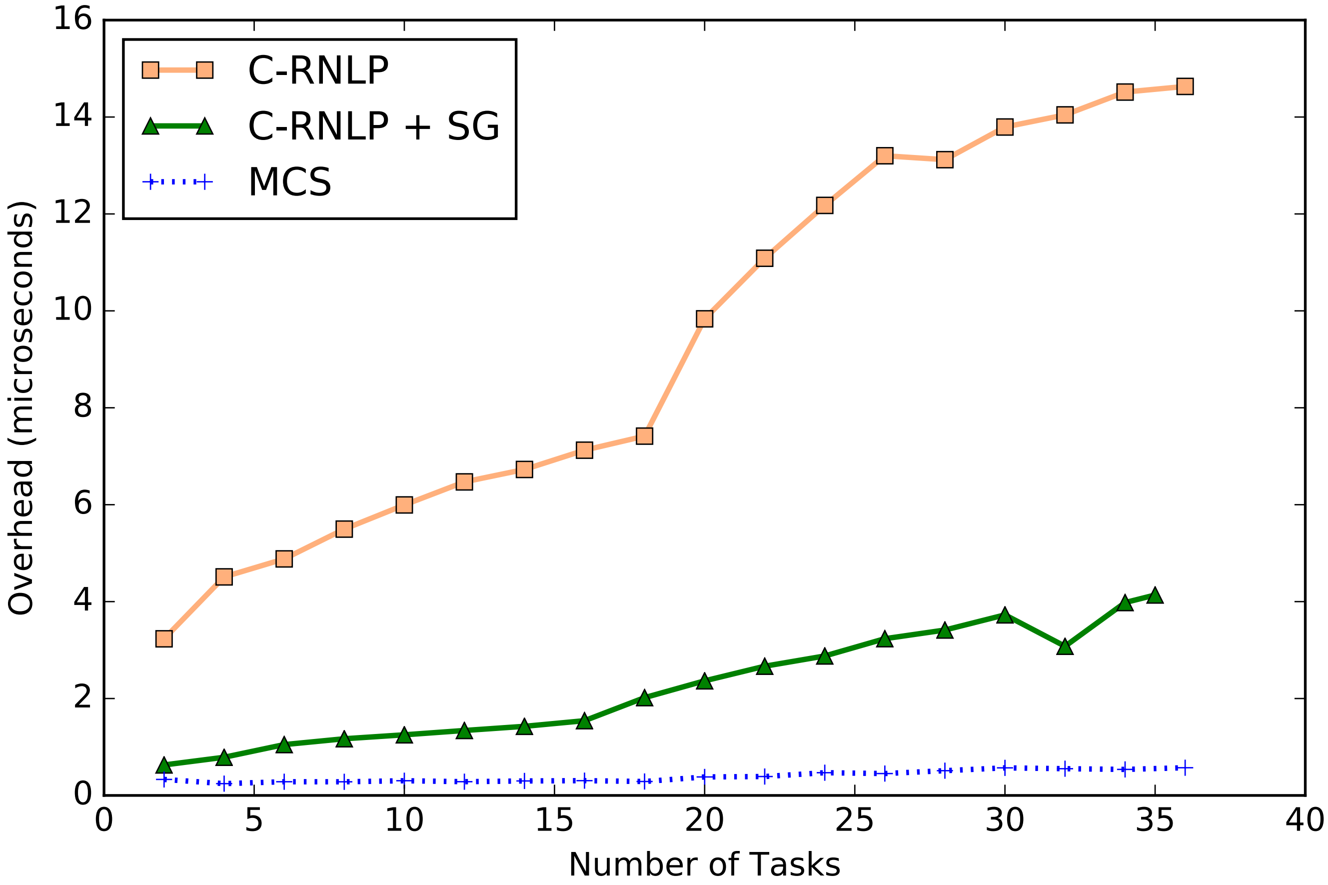
Cost of low blocking: high overhead





C-RNLP: C. Jarrett, B. Ward, and J. Anderson. A contention-sensitive fine-grained locking protocol for multiprocessor real-time systems. *RTNS 2015*.

MCS: J. Mellor-Crummey and M. Scott. Algorithms for scalable synchronization of shared-memory multiprocessors. *Transactions on Computer Systems*, 9(1), 1991.



Proposed Solution: Lock Servers

Four lock server paradigms

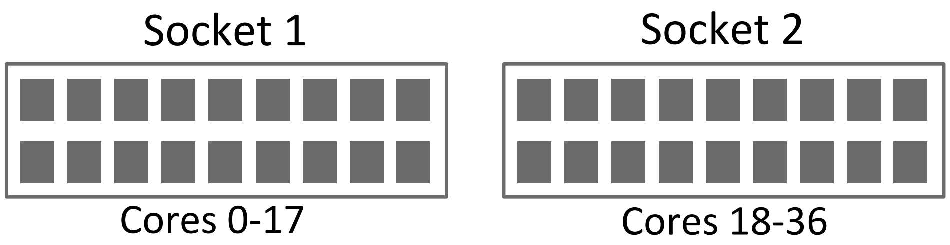
- Implementation
- Evaluation

Lock server coordination protocol

Contributions

Platform Description

dual-socket,
18-cores-per-socket
Intel Xeon E5-2699



L1 Data | L1 Instr.

L2 (shared between two cores)

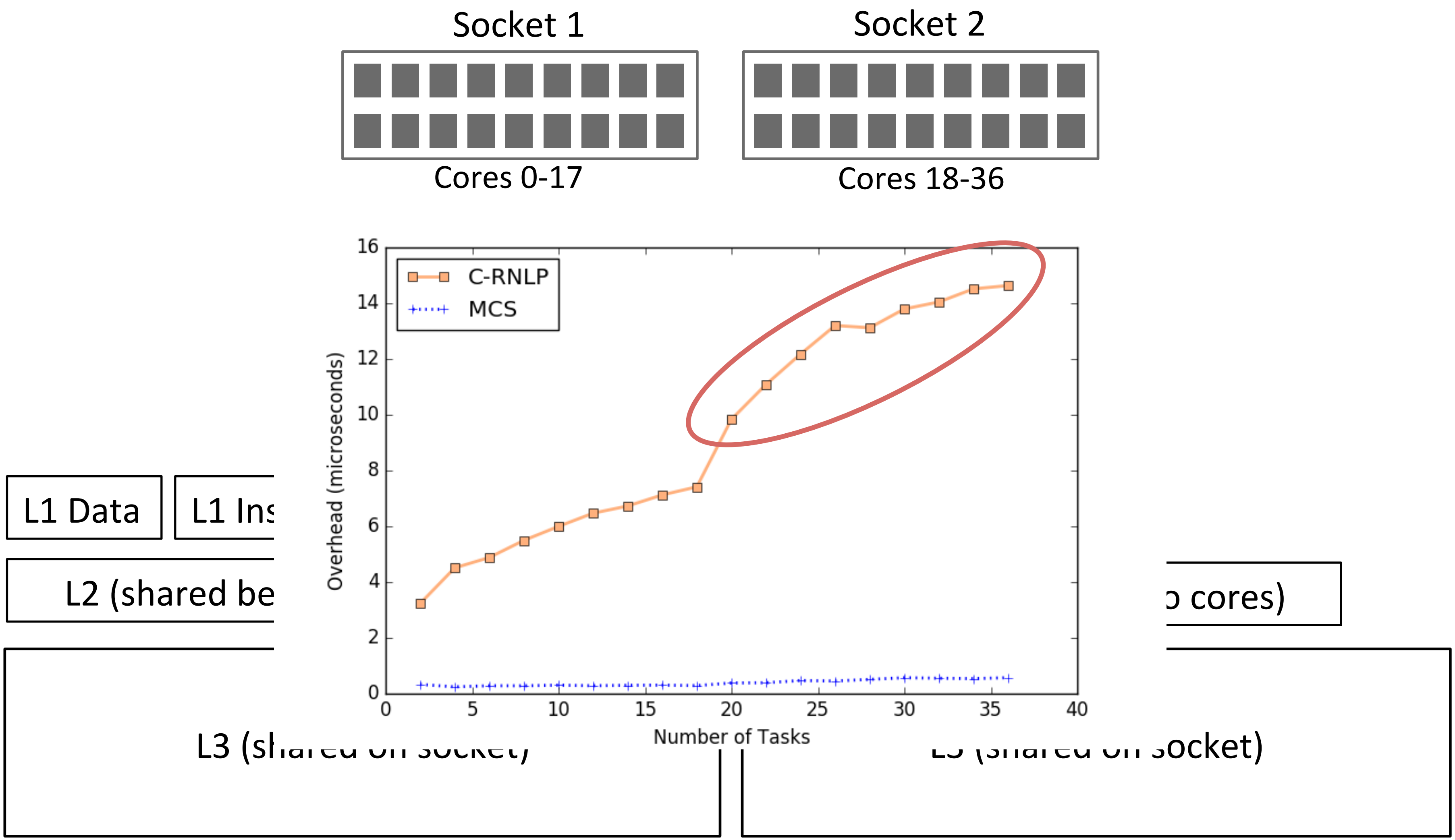
L3 (shared on socket)

L1 Data | L1 Instr.

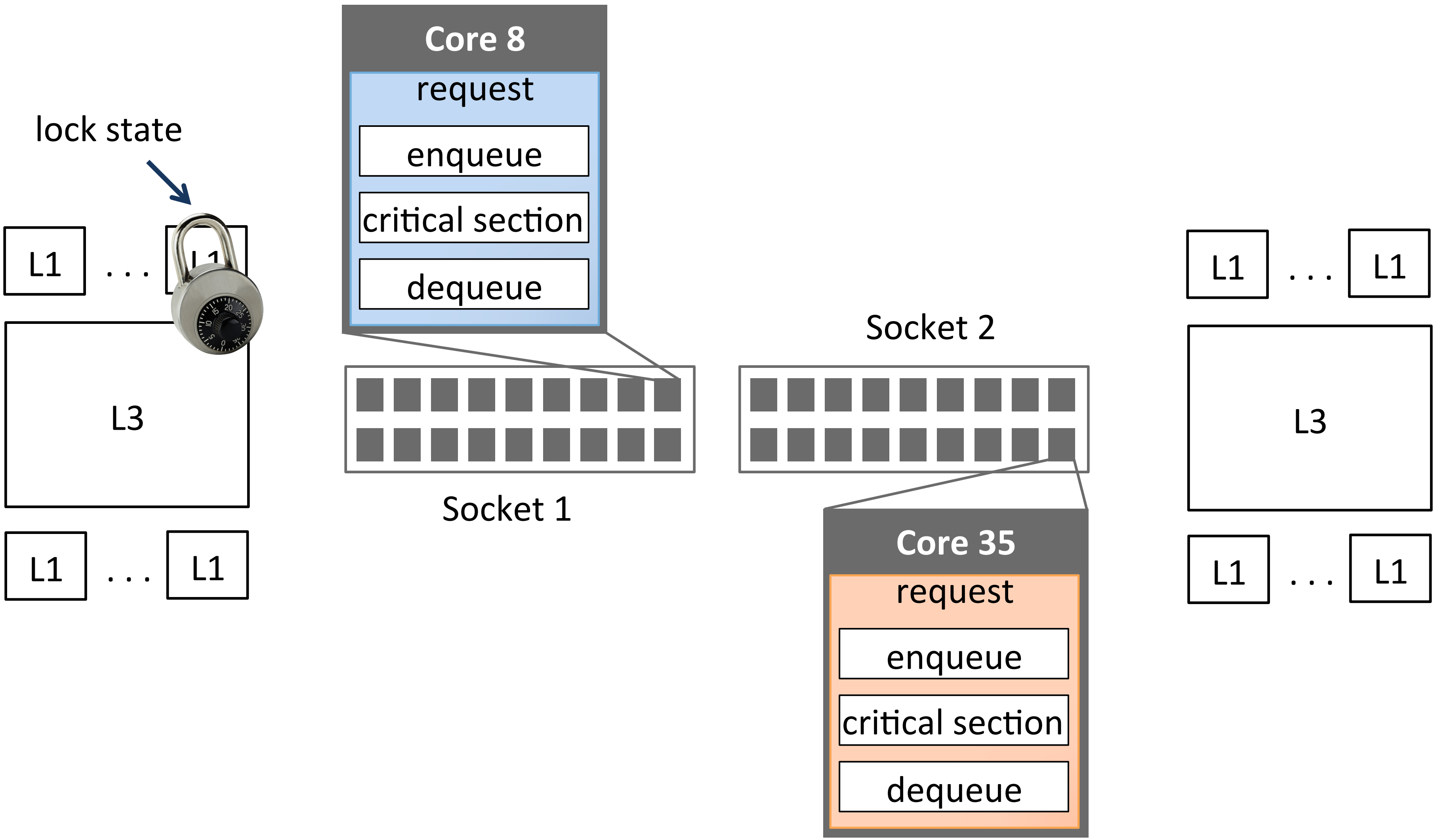
L2 (shared between two cores)

L3 (shared on socket)

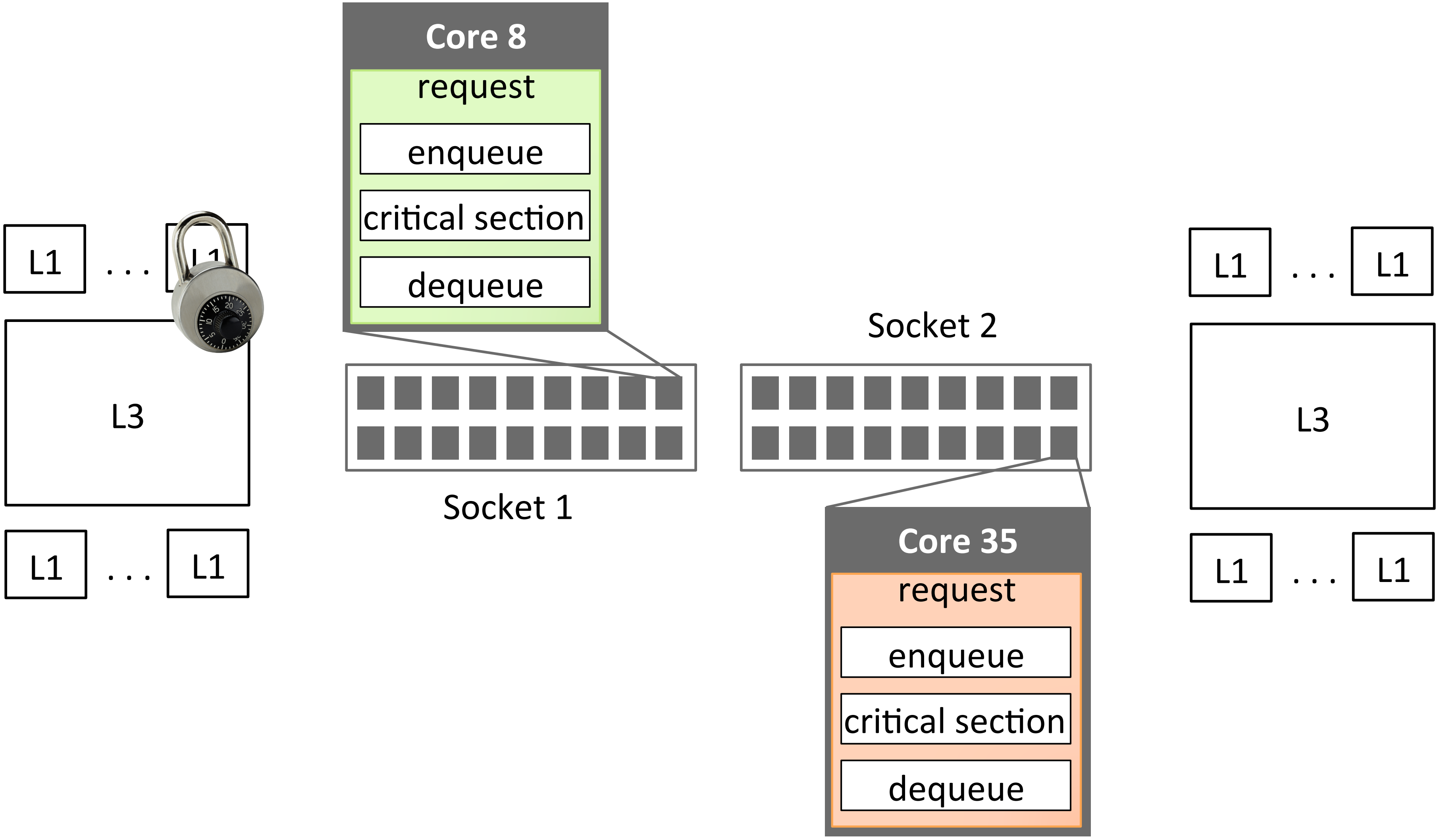
Platform Description



Standard C-RNLP



Standard C-RNLP



The Idea

Remote Core Locking

- used to reduce critical-section lengths [1]

We developed: lock servers

- used to reduce overhead

Lock server: a process dedicated to performing lock and unlock functions

[1] J. Lozi, F. David, G. Thomas, J. Lawall, and G. Muller. Remote core locking: migrating critical-section execution to improve the performance of multithreaded applications. *USENIX ATC 2012*.

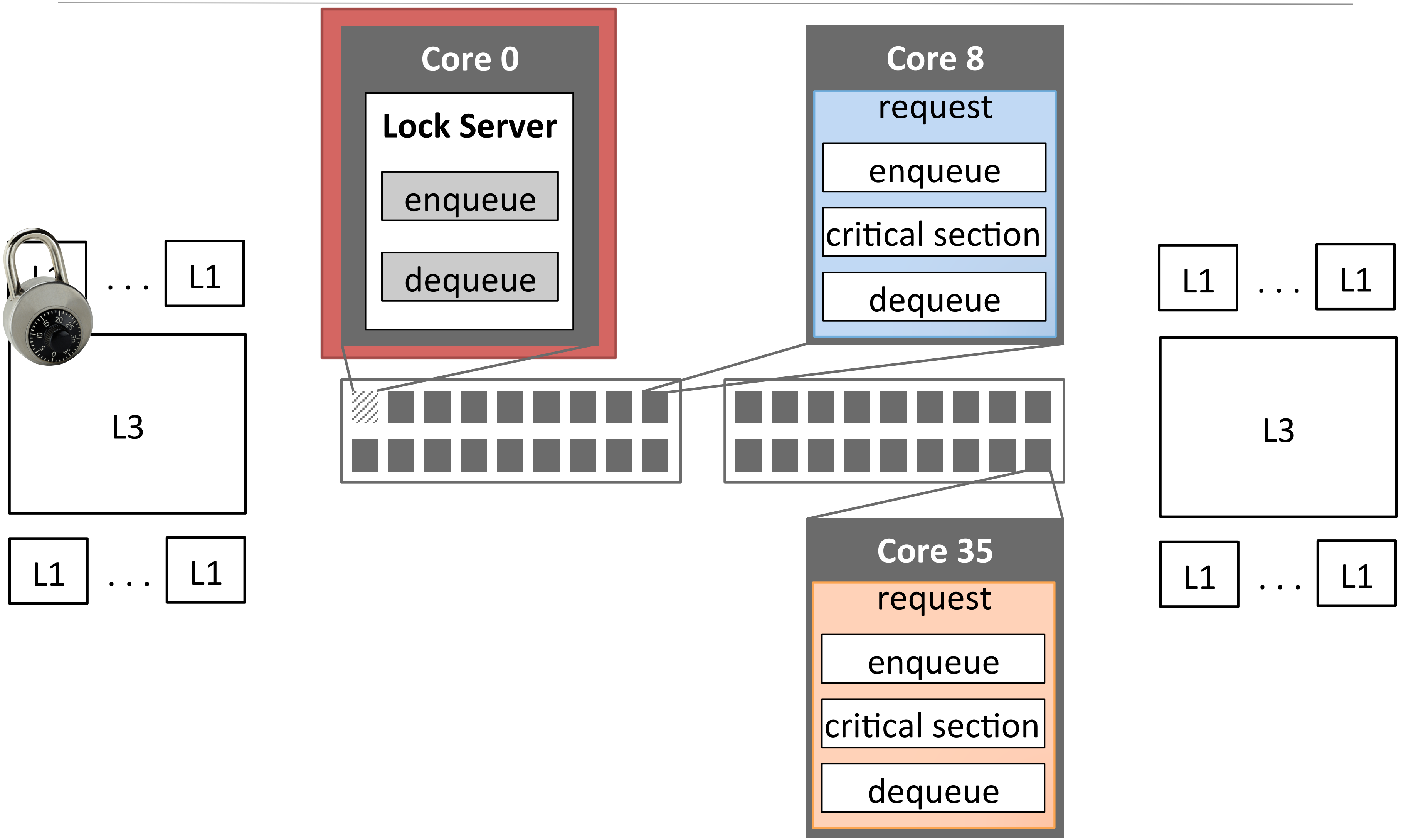
Lock Server Paradigms

locality

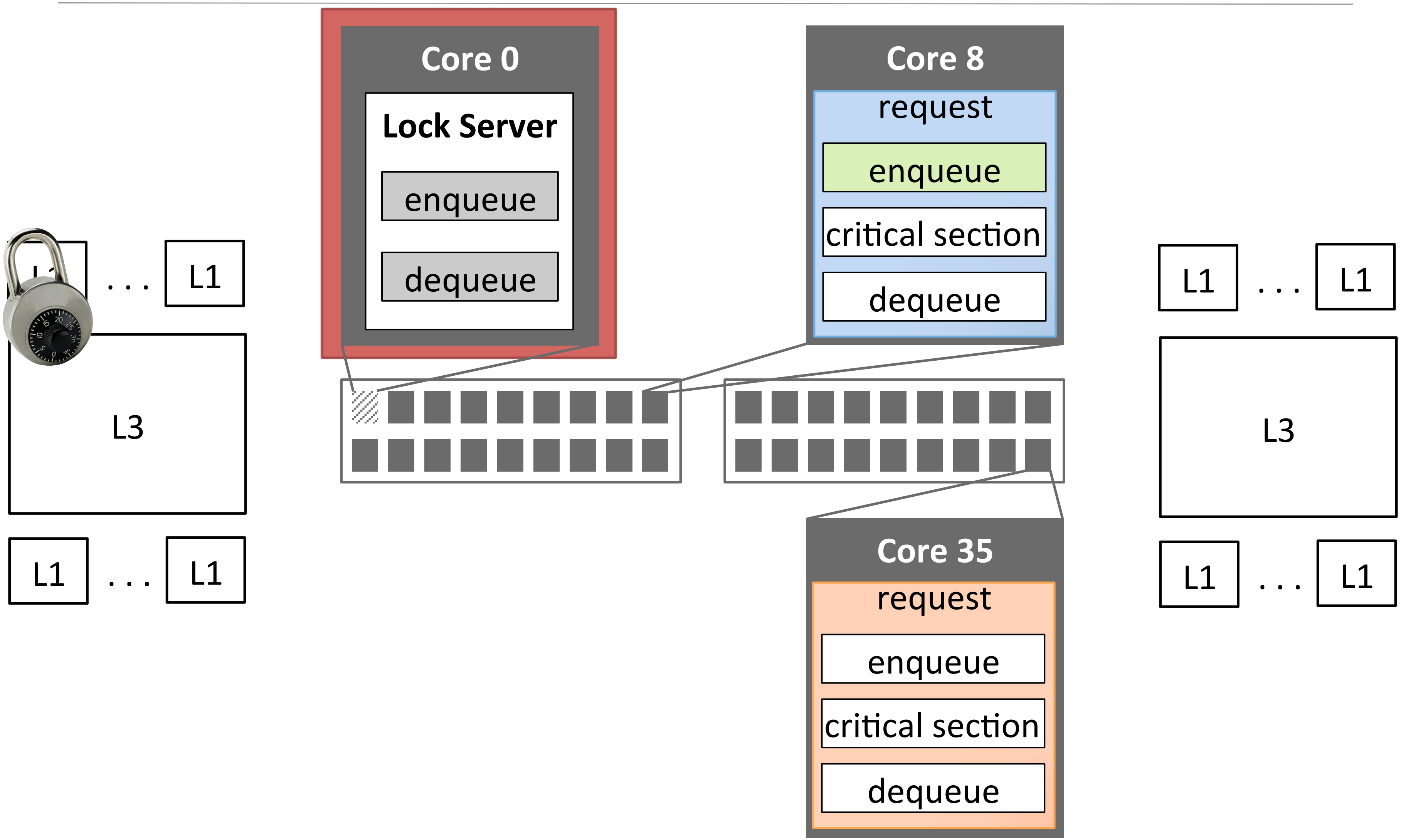
mobility

	Global	Local
Static	#1	
Floating		

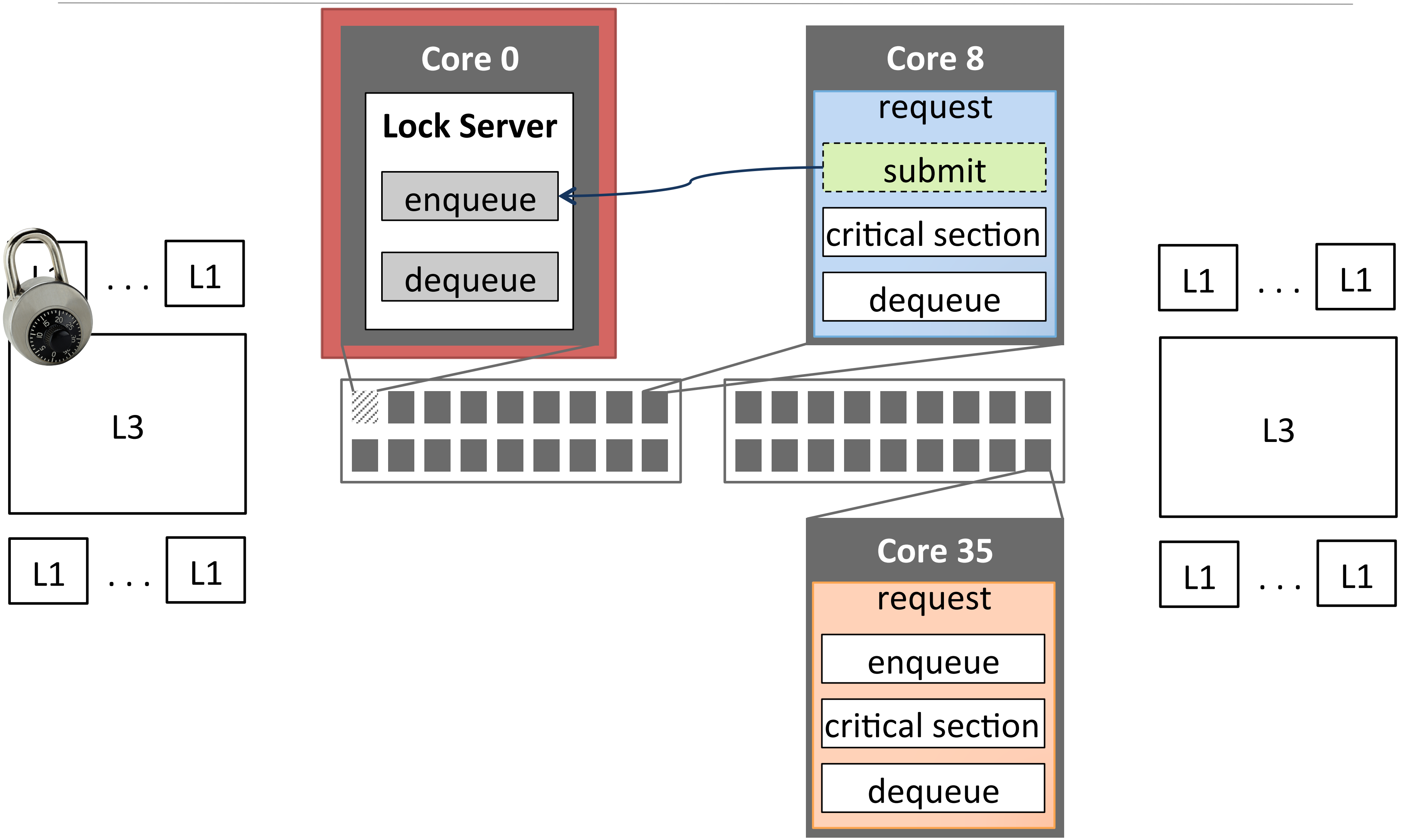
Paradigm #1: Static Global



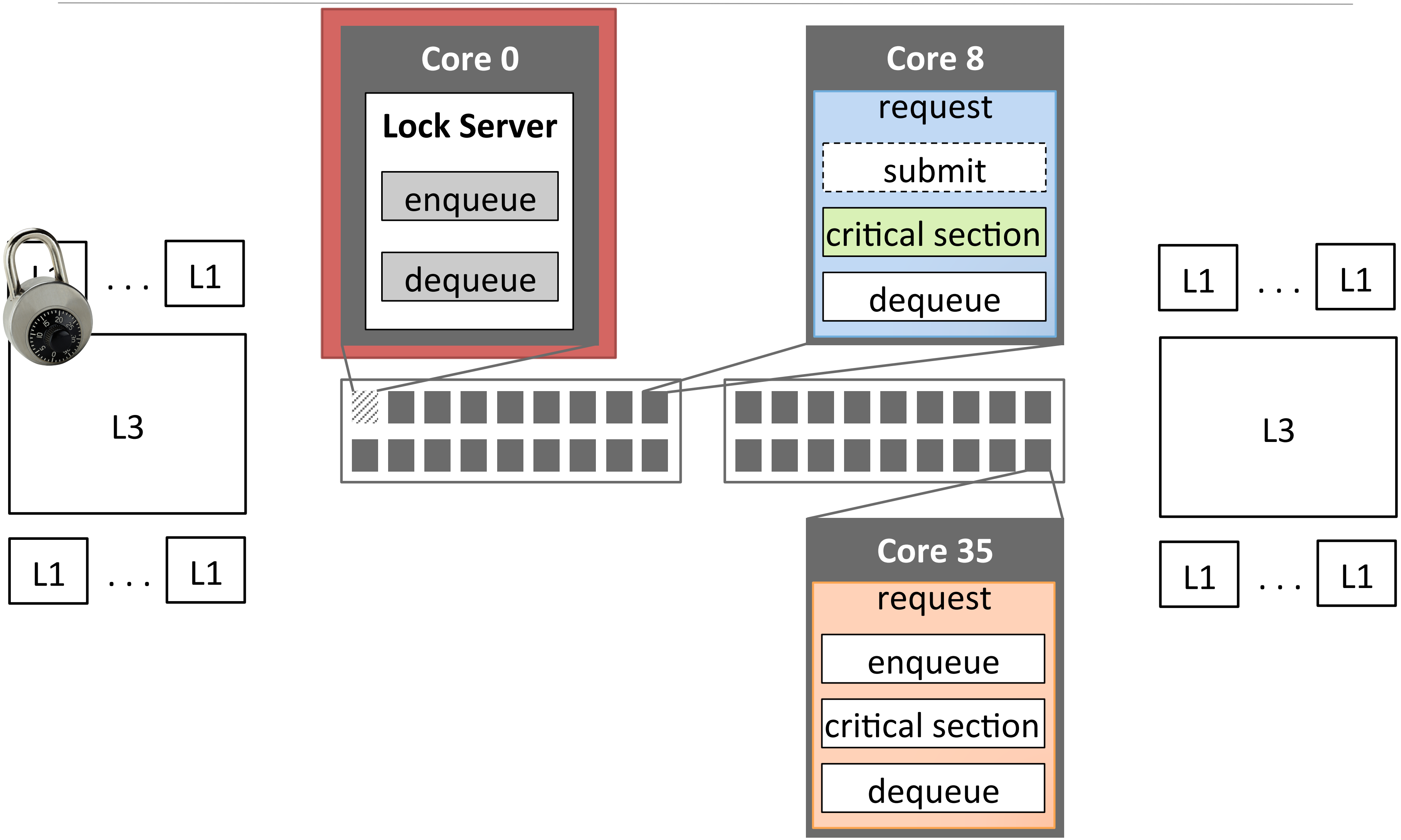
Paradigm #1: Static Global



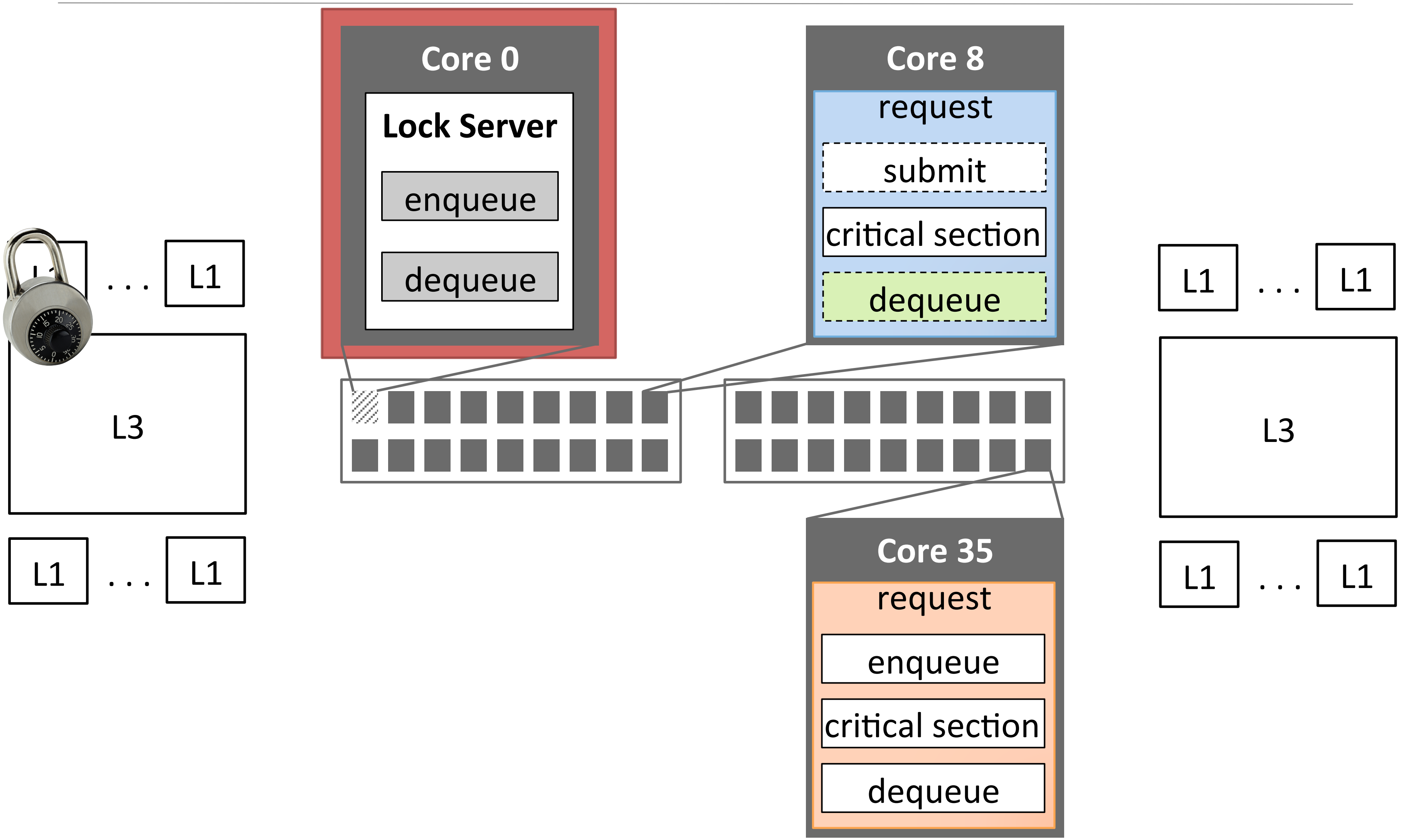
Paradigm #1: Static Global



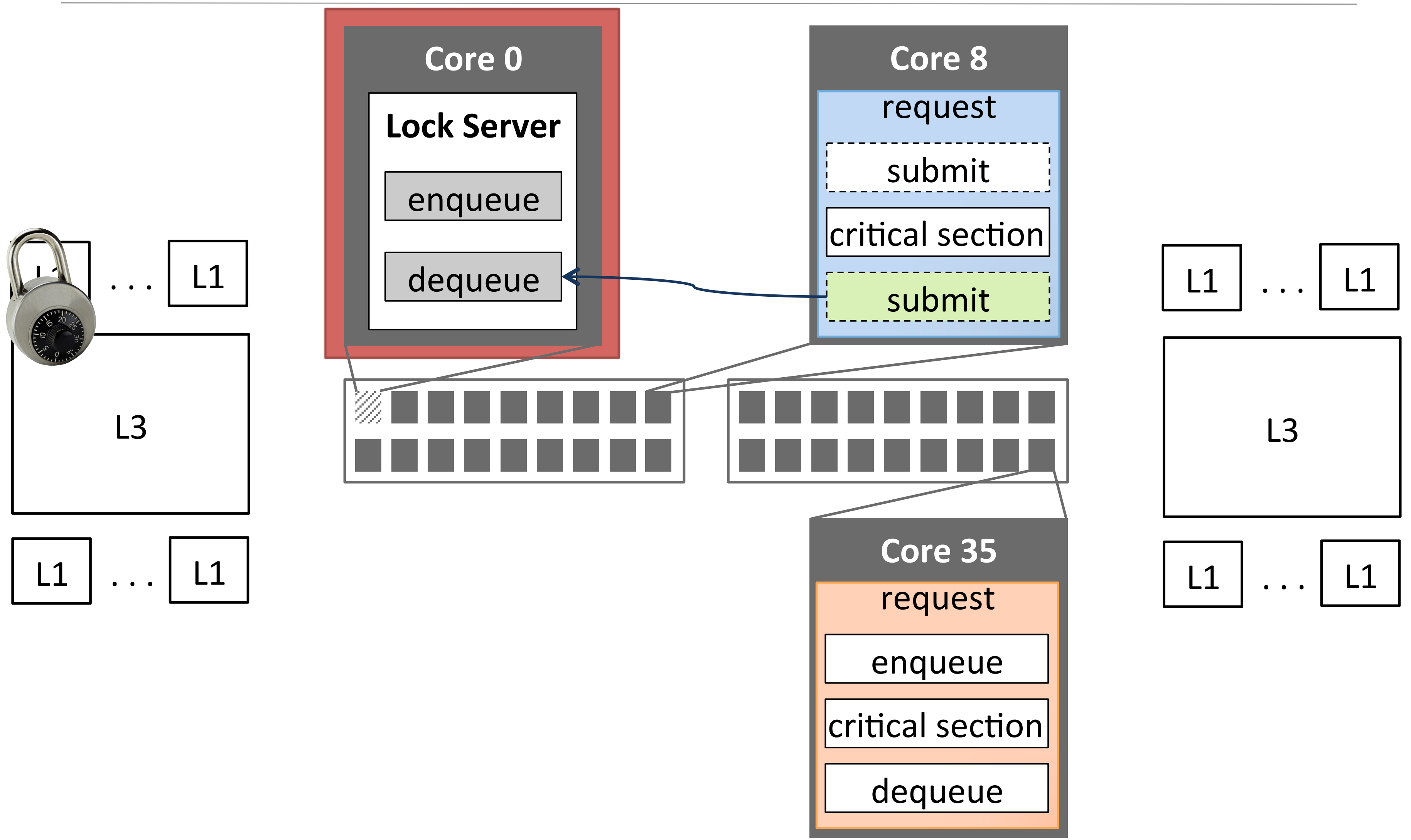
Paradigm #1: Static Global



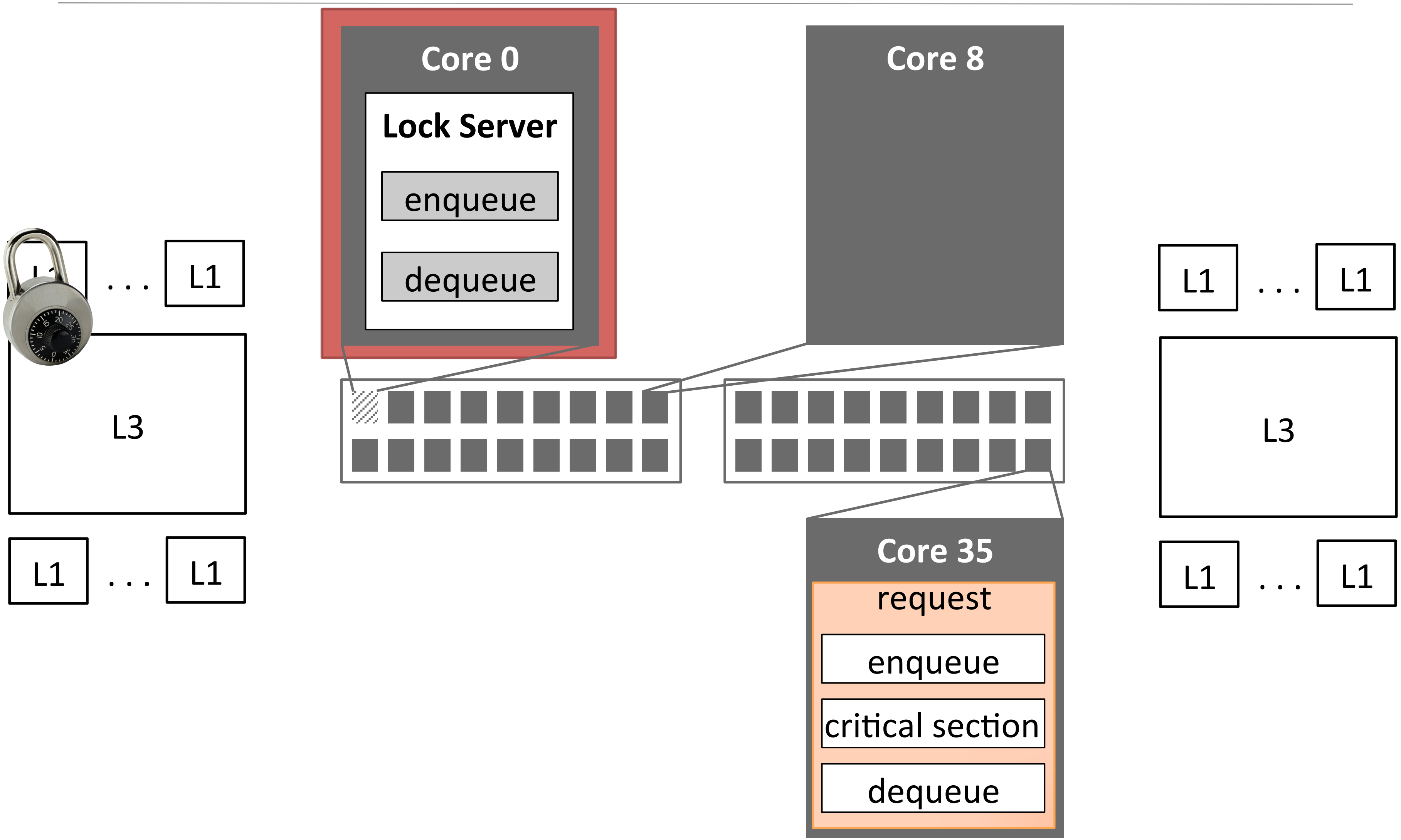
Paradigm #1: Static Global



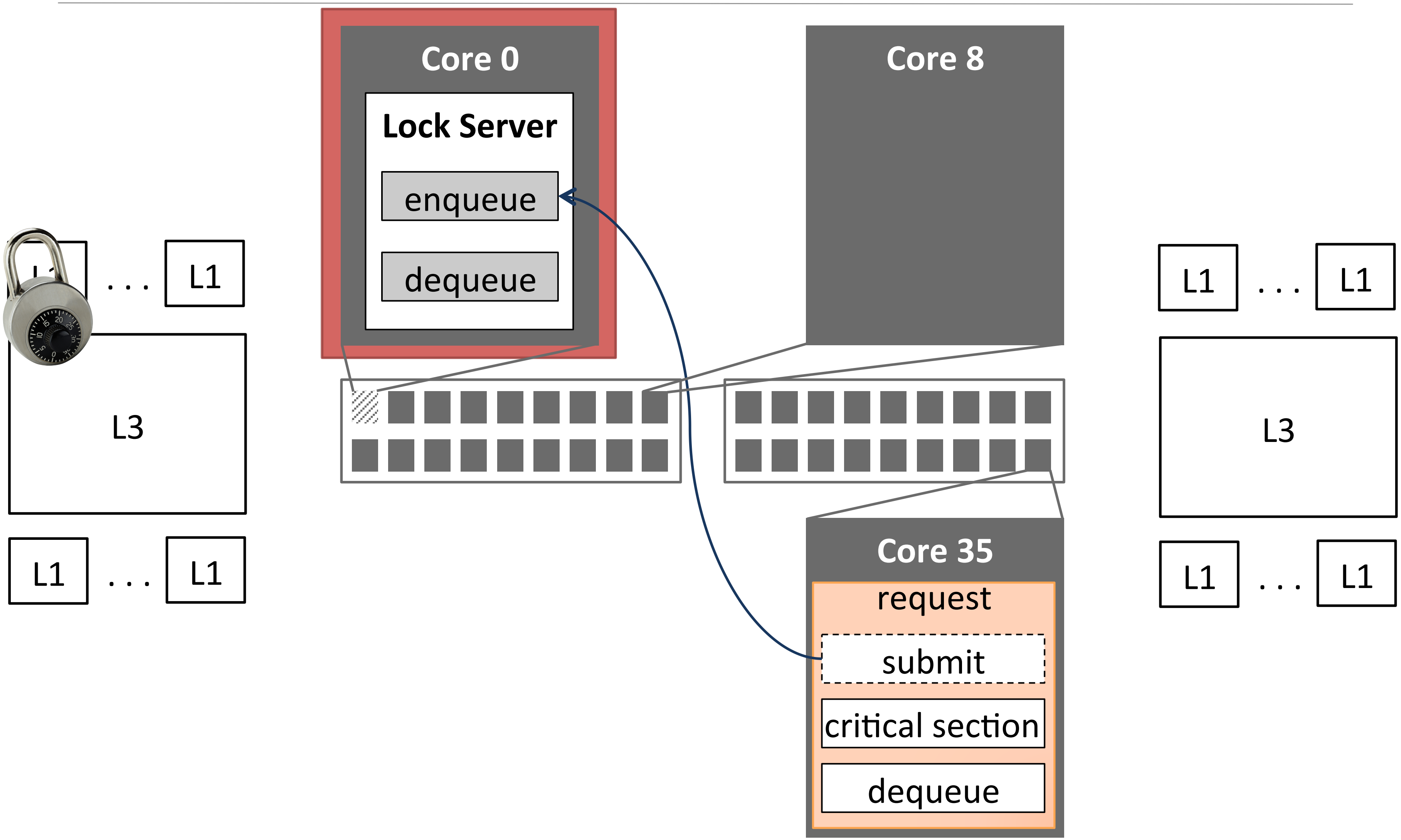
Paradigm #1: Static Global



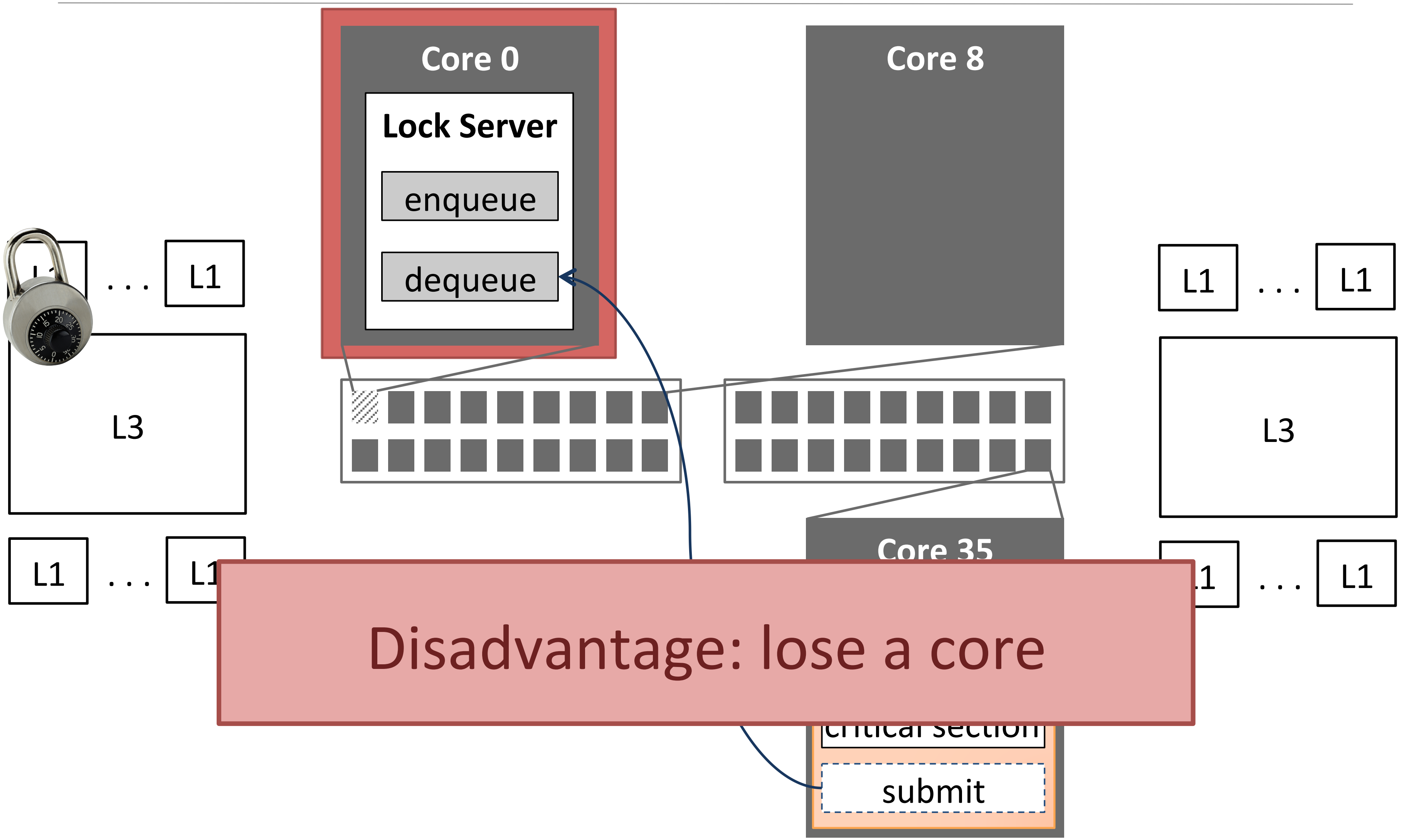
Paradigm #1: Static Global



Paradigm #1: Static Global



Paradigm #1: Static Global



Disadvantage: lose a core

Lock Server Paradigms

locality

mobility

	Global	**Local**
Static	#1	
Floating	#2	

Lock Server Paradigms

locality

mobility

	Global	**Local**
Static	- Lose 1 core + L1 cache affinity	
Floating	#2	

Lock Server Paradigms

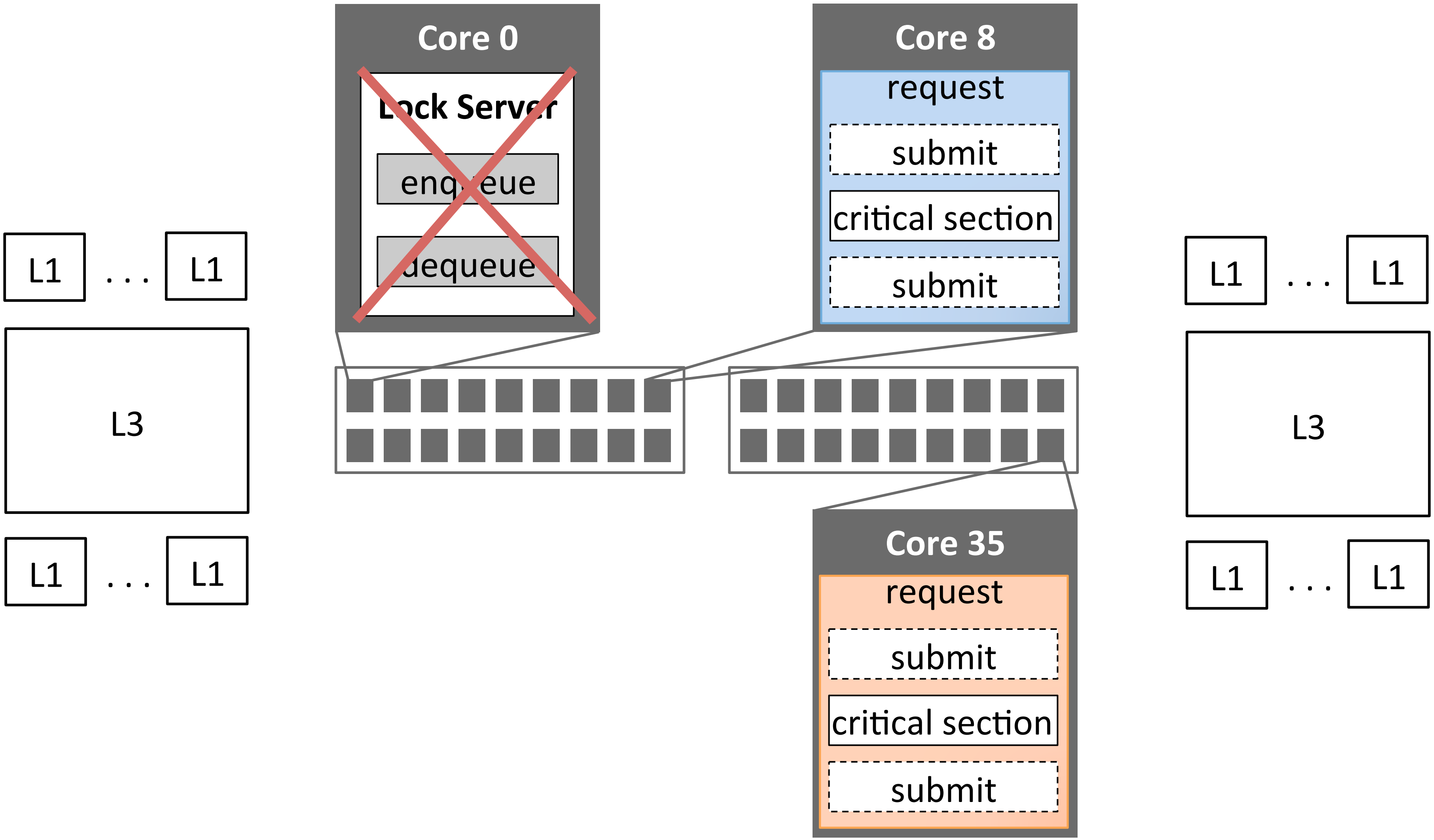
locality

	Global	**Local**
Floating	#2	

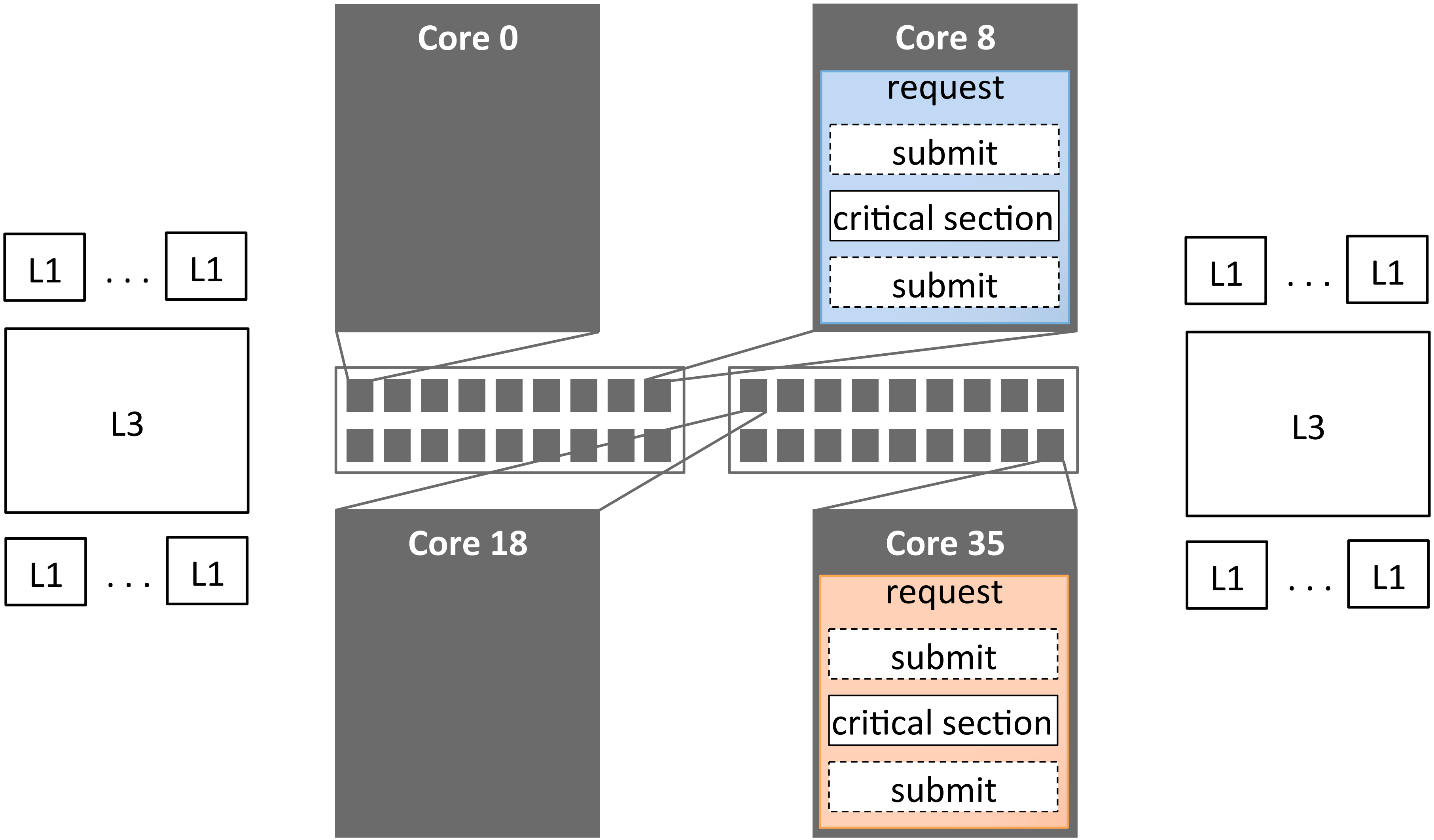
mobility

Key insight: blocked requests are busy-waiting, using CPU

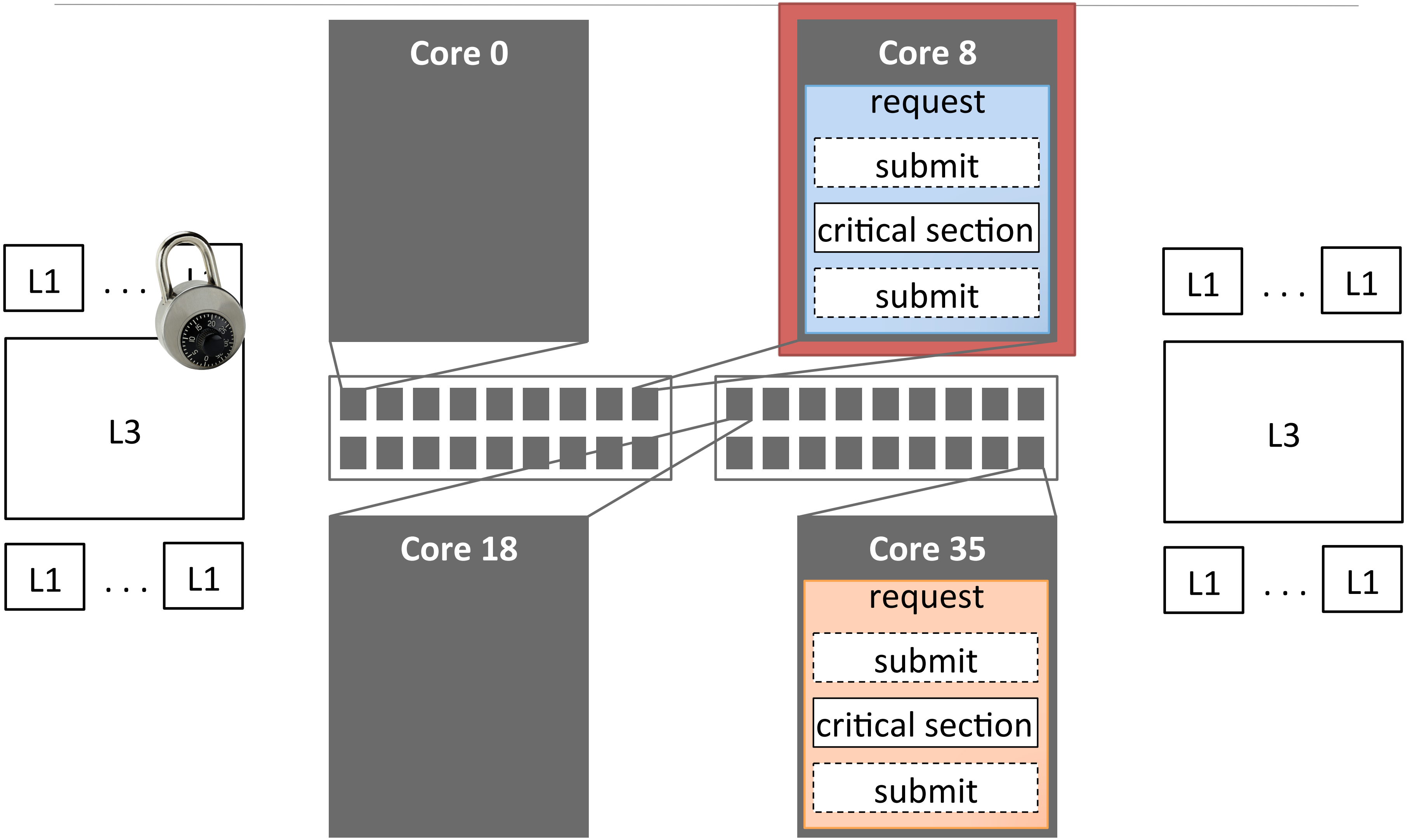
Paradigm #2: Floating Global



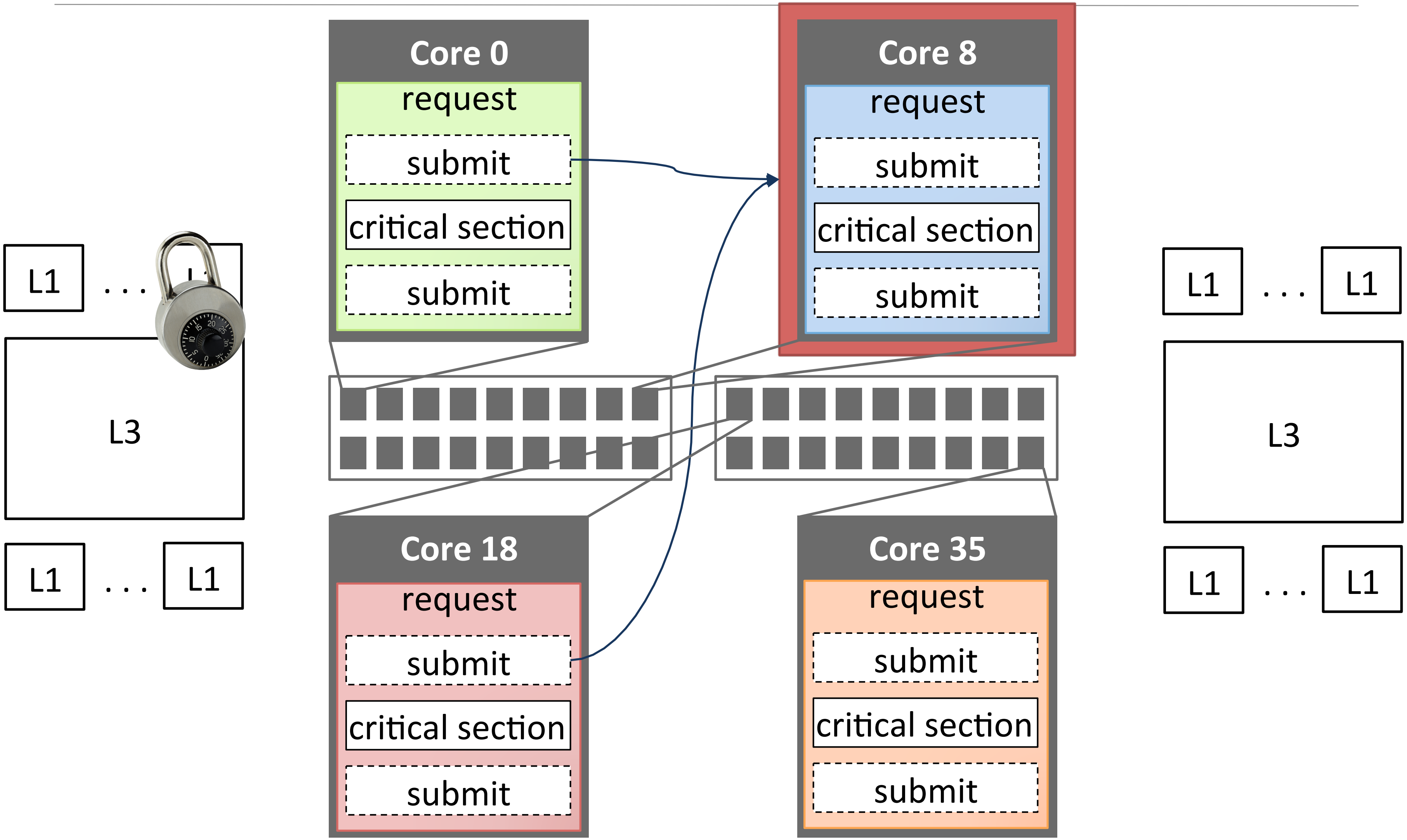
Paradigm #2: Floating Global



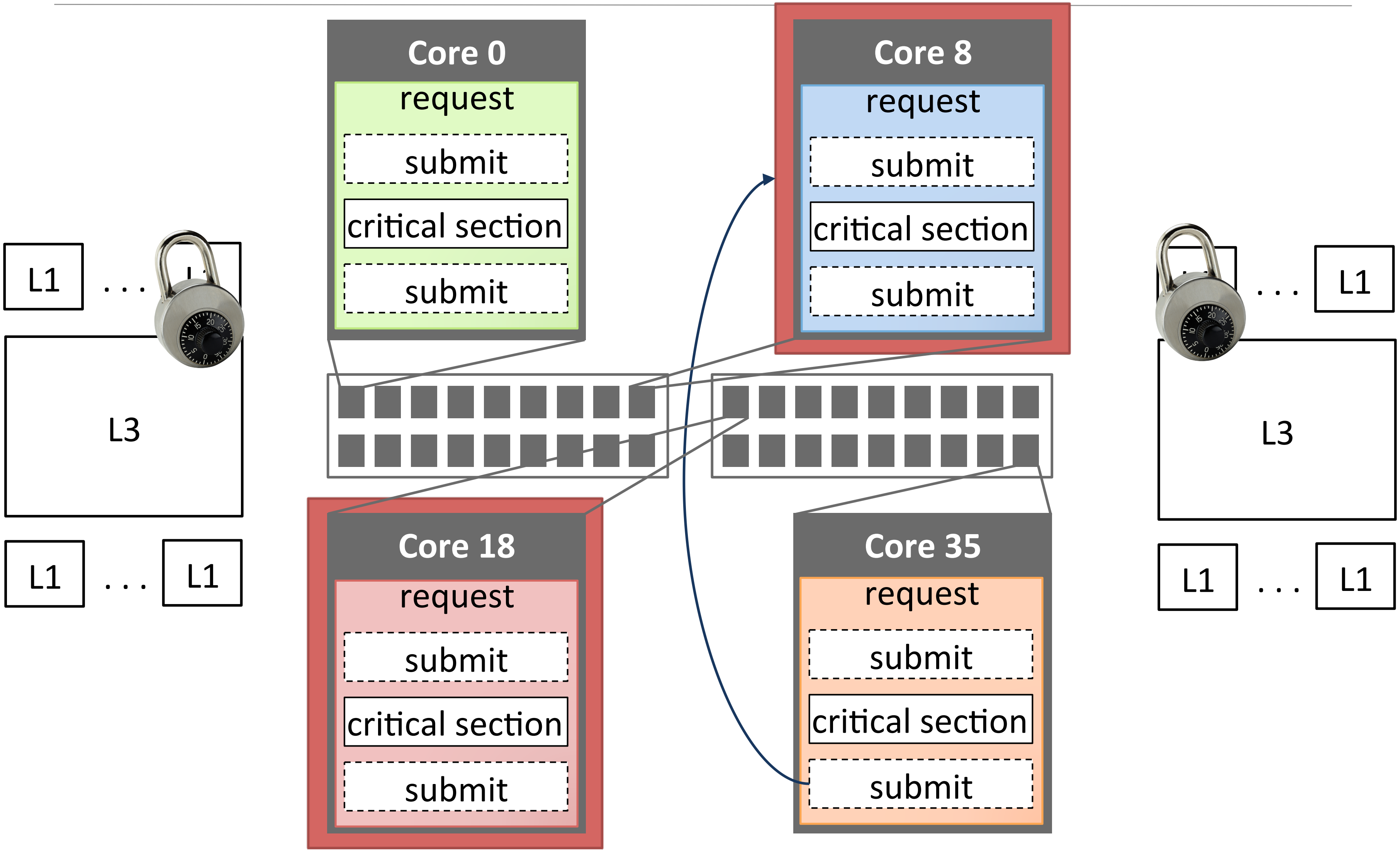
Paradigm #2: Floating Global



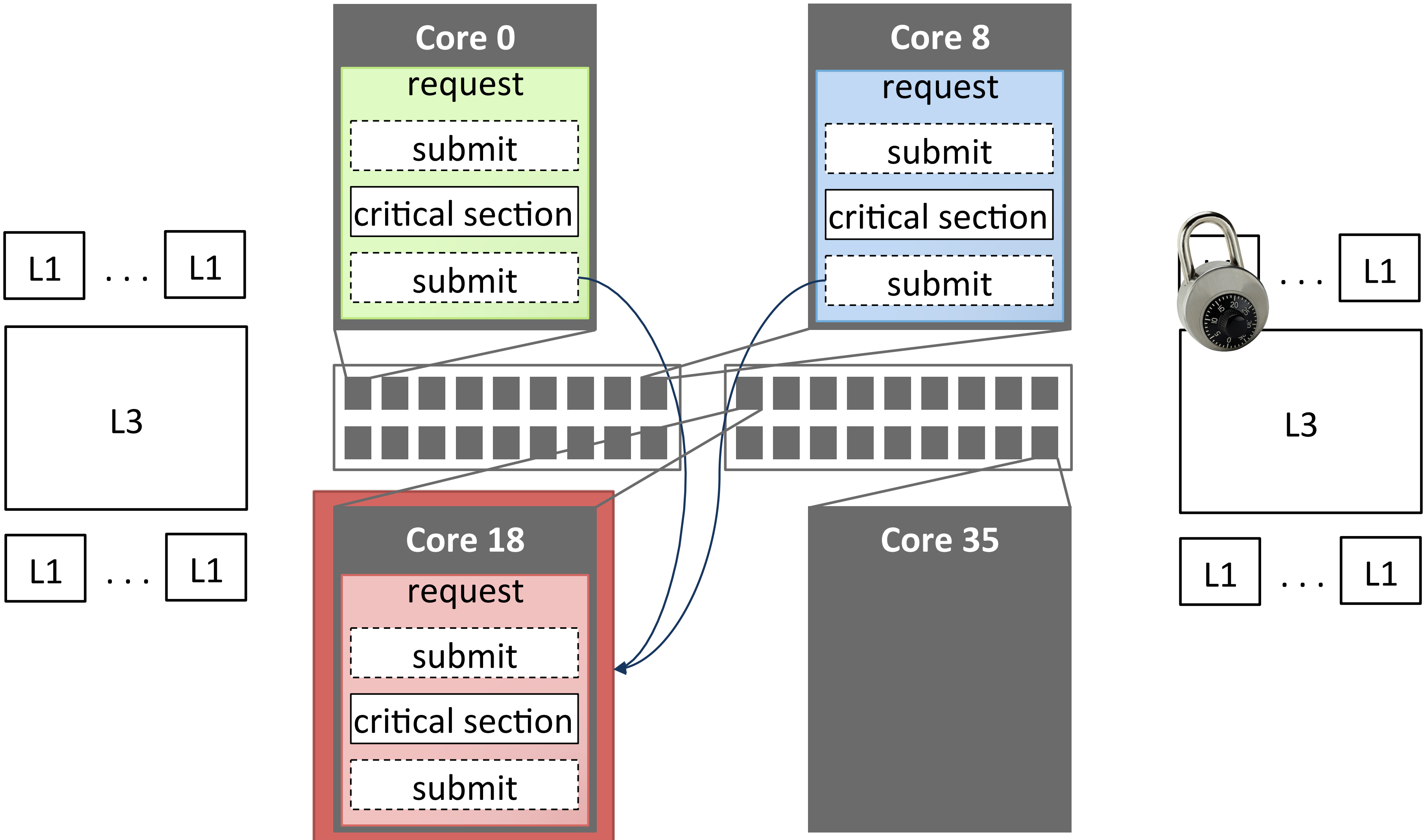
Paradigm #2: Floating Global



Paradigm #2: Floating Global



Paradigm #2: Floating Global



Lock Server Paradigms

locality

mobility	Global	Local
Static	- Lose 1 core + L1 cache affinity	
Floating	#2	

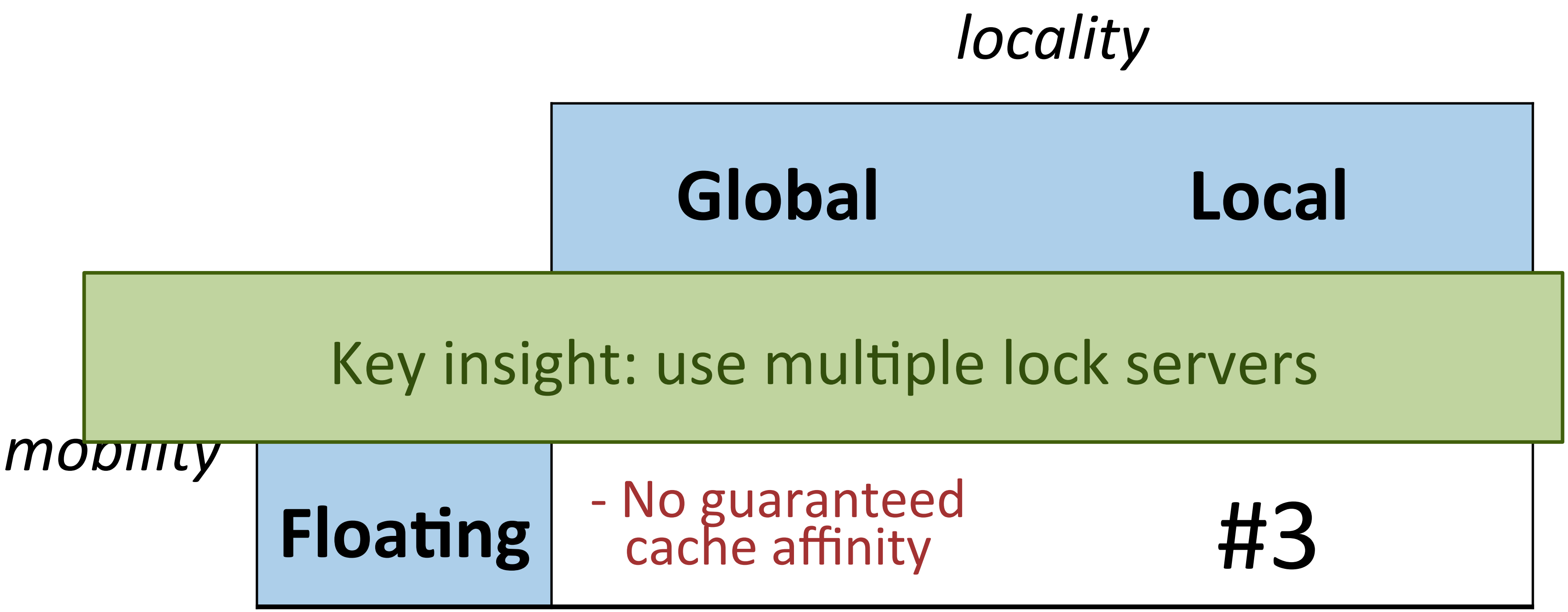
Lock Server Paradigms

locality

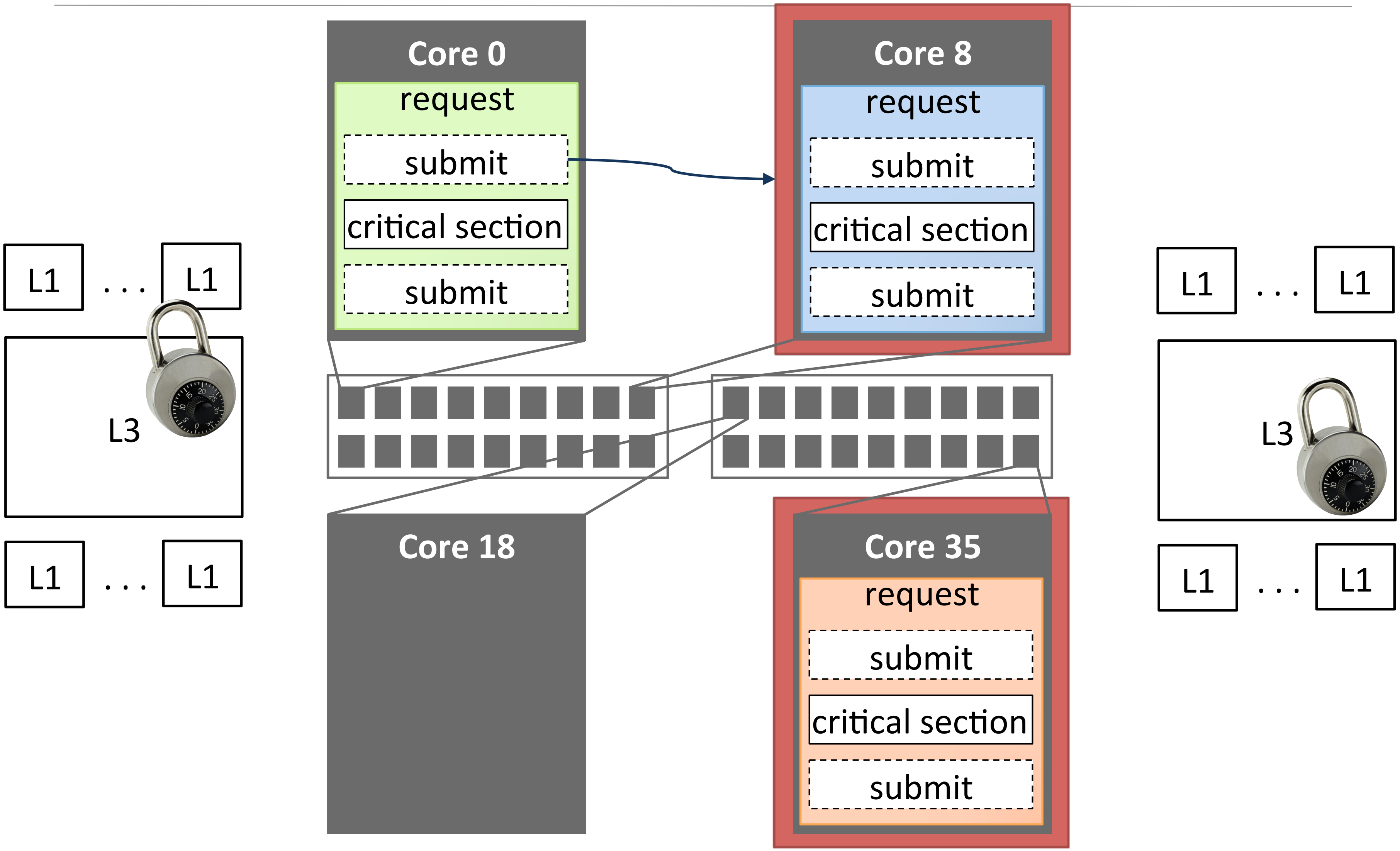
mobility

	Global	**Local**
Static	- Lose 1 core + L1 cache affinity	
Floating	- No guaranteed cache affinity	#3

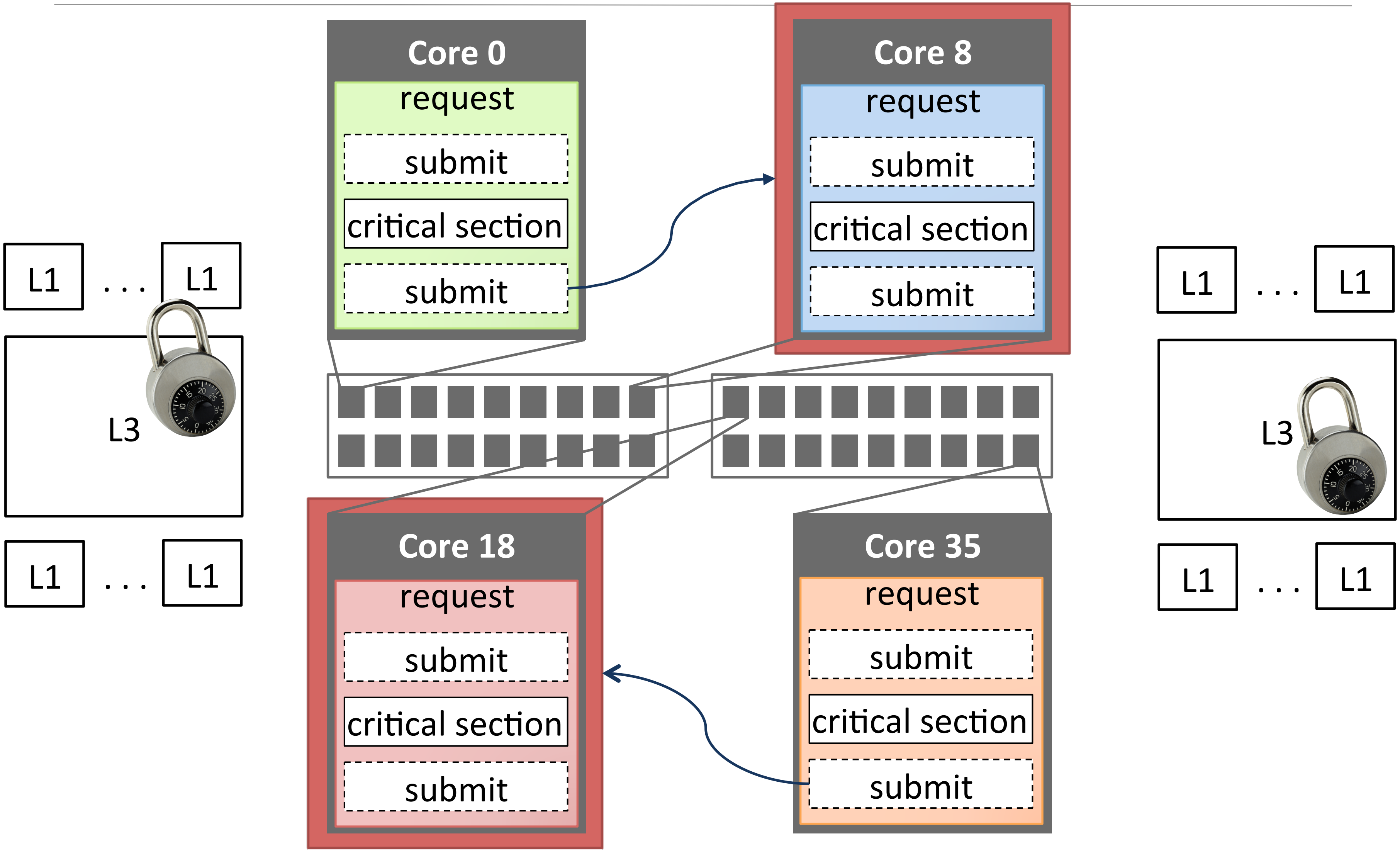
Lock Server Paradigms



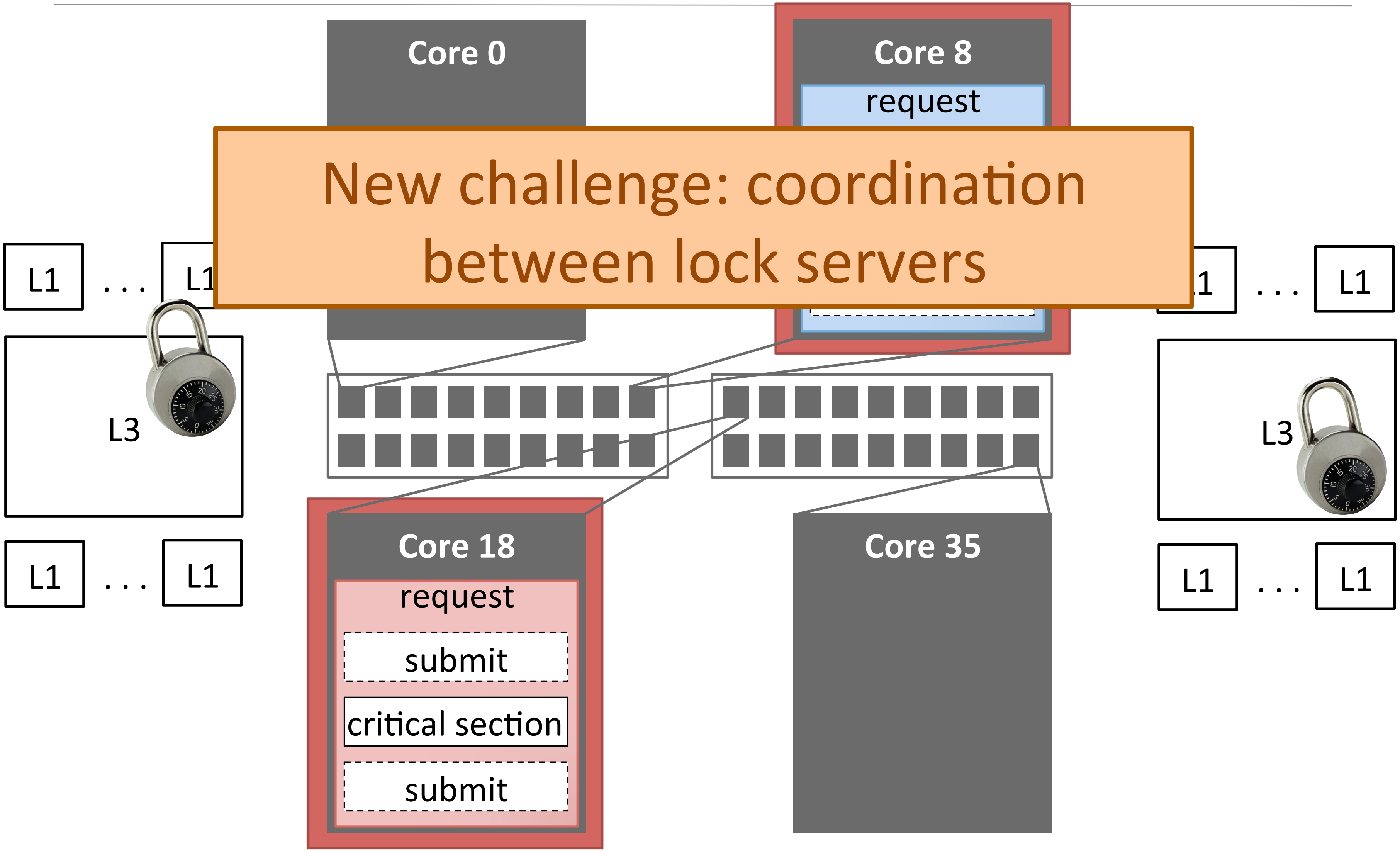
Paradigm #3: Floating Local



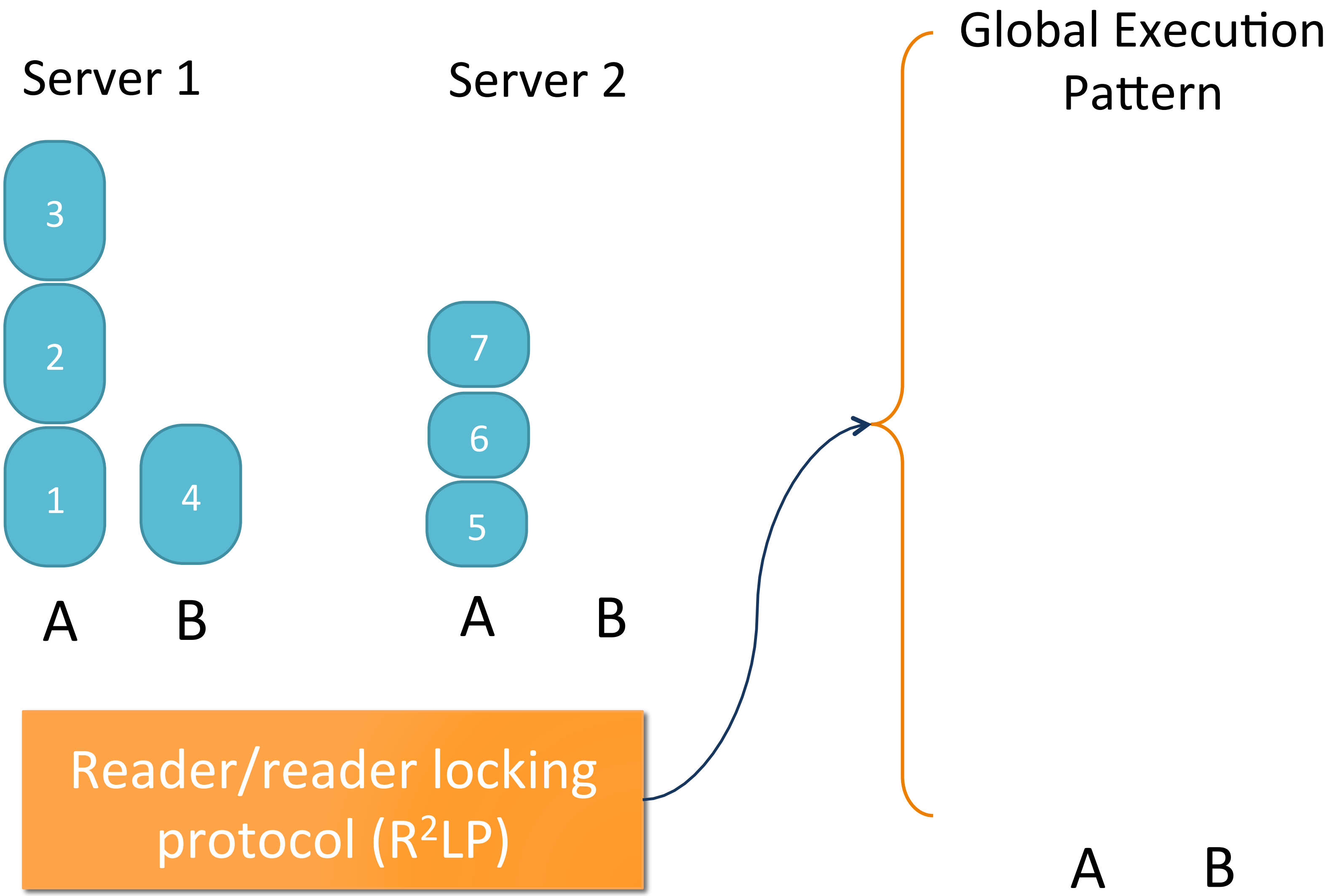
Paradigm #3: Floating Local



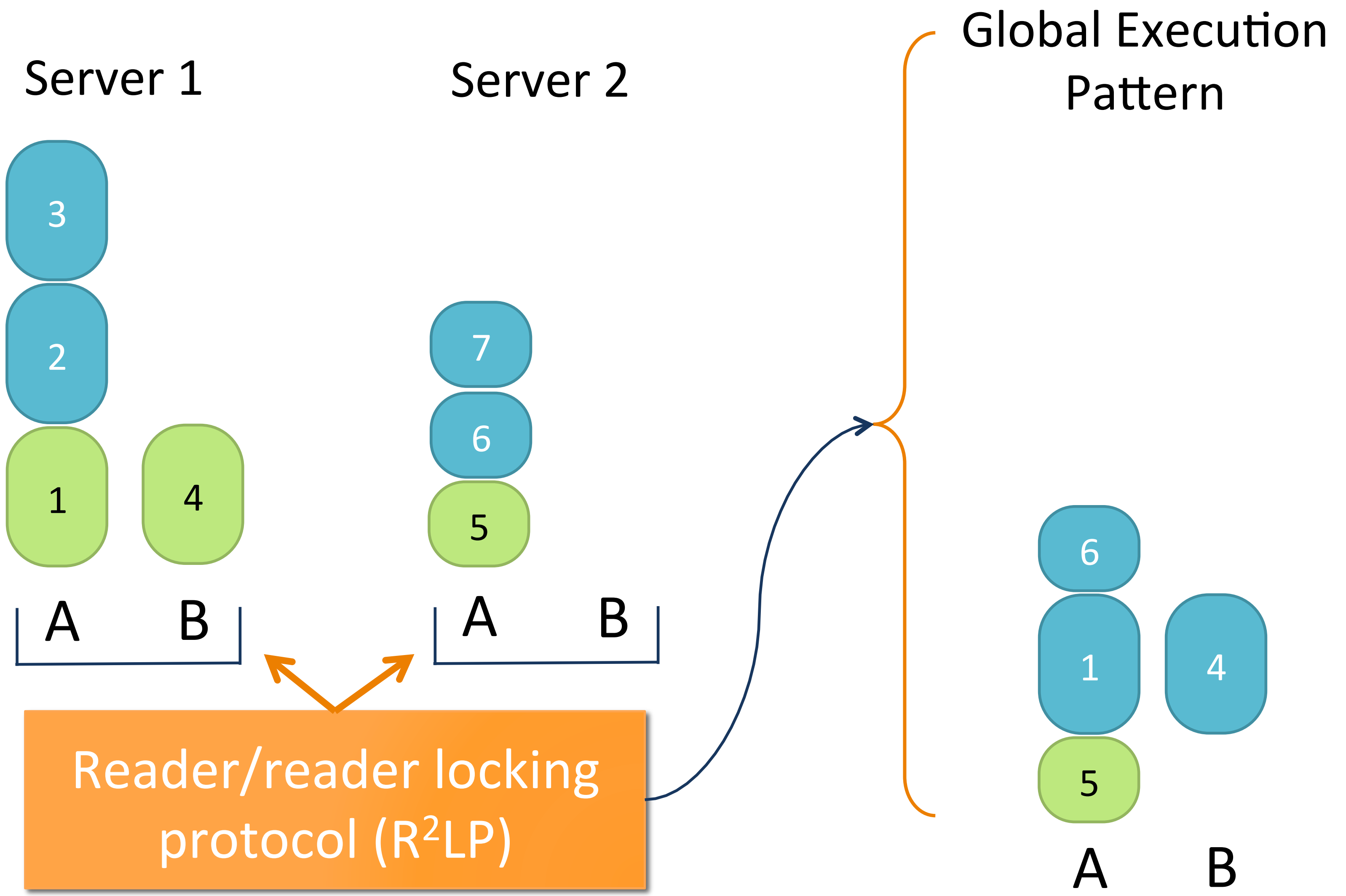
Paradigm #3: Floating Local



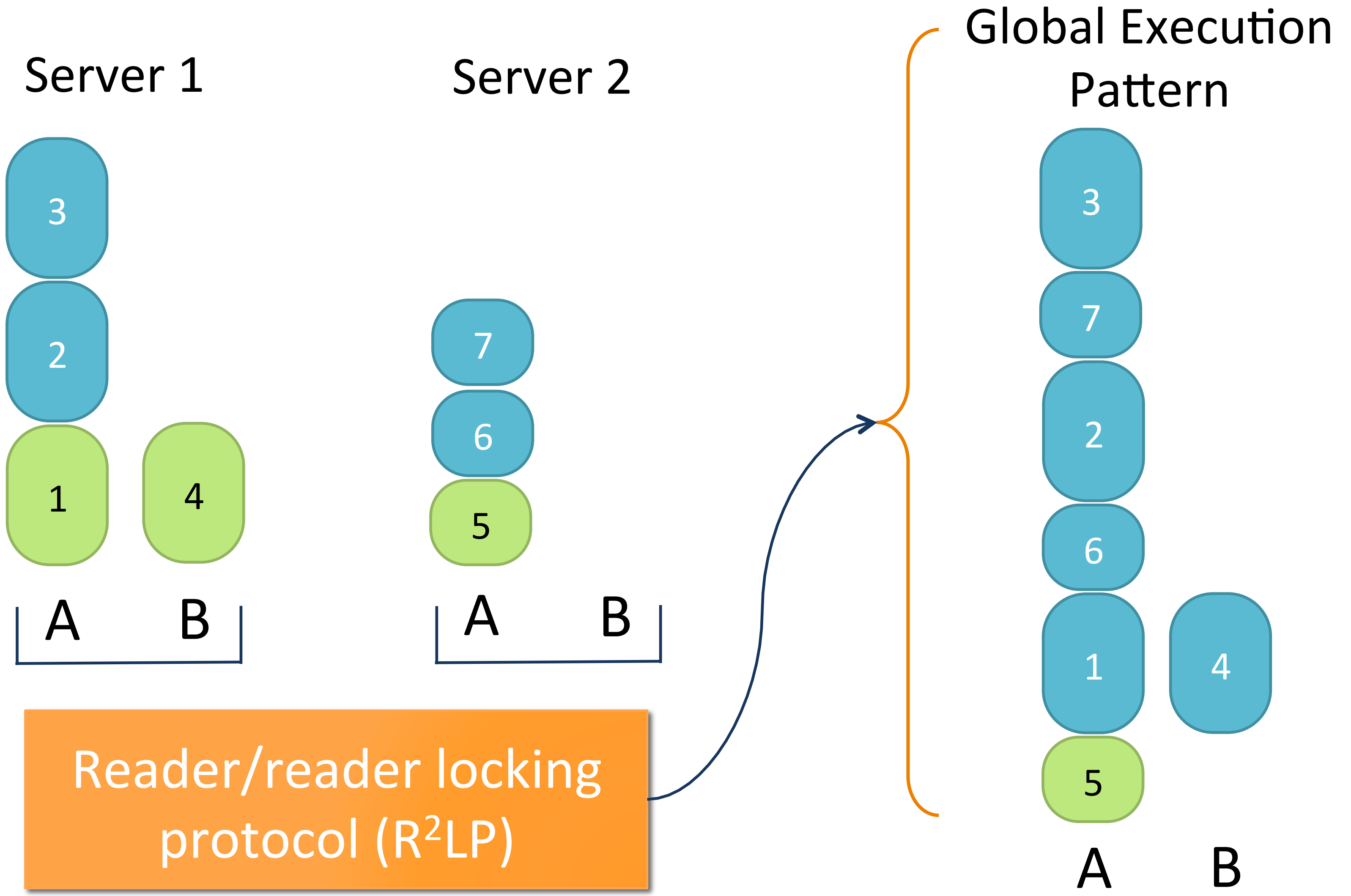
Lock Server Coordination



Lock Server Coordination

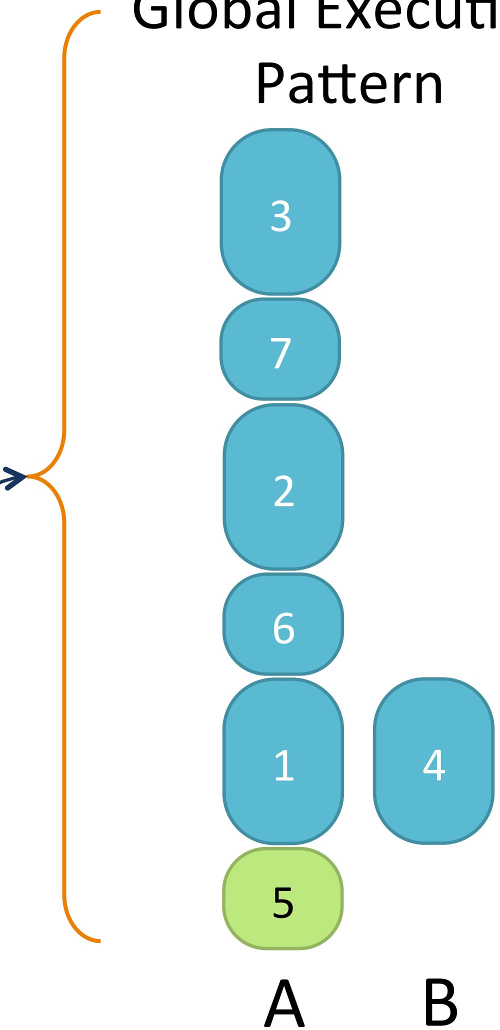


Lock Server Coordination



Lock Server Coordination

Global Execution Pattern



Original C-RNLP bound:

$(c_i + 1)L_{MAX}$

Reader/reader locking protocol (R^2LP)

Lock Server Coordination

Original C-RNLP bound:

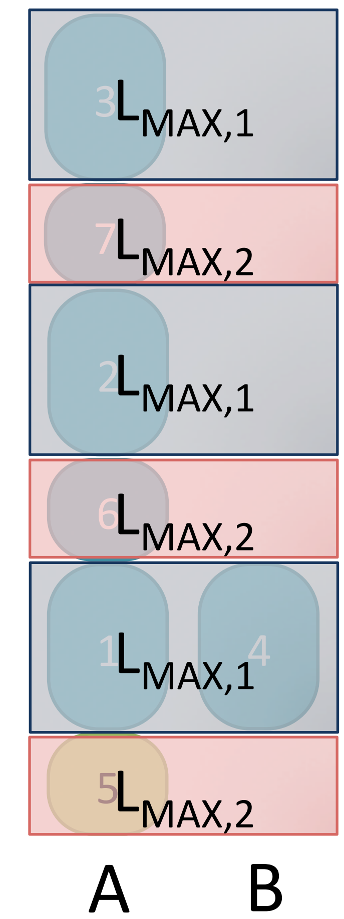
$$(c_i + 1)L_{MAX}$$

Two-server C-RNLP bound with R^2LP:

$$(c_{i,s} + 1)(L_{MAX,1} + L_{MAX,2})$$

Reader/reader locking protocol (R^2LP)

Global Execution Pattern



Lock Server Paradigms

locality / *mobility*

	Global	**Local**
Static	- Lose 1 core + L1 cache affinity	
Floating	- No guaranteed cache affinity	+ L3 cache affinity

Lock Server Paradigms

locality

mobility

	Global	**Local**
Static	- Lose 1 core + L1 cache affinity	- Lose multiple cores + L1 cache affinity
Floating	- No guaranteed cache affinity	+ L3 cache affinity

coordination with R^2LP, additional blocking considerations

Experimental Evaluation

Measured **overhead** and **blocking**

One task per core issuing 10,000 random requests

64 resources, each task requests 4 of these, critical-section lengths = 40μs

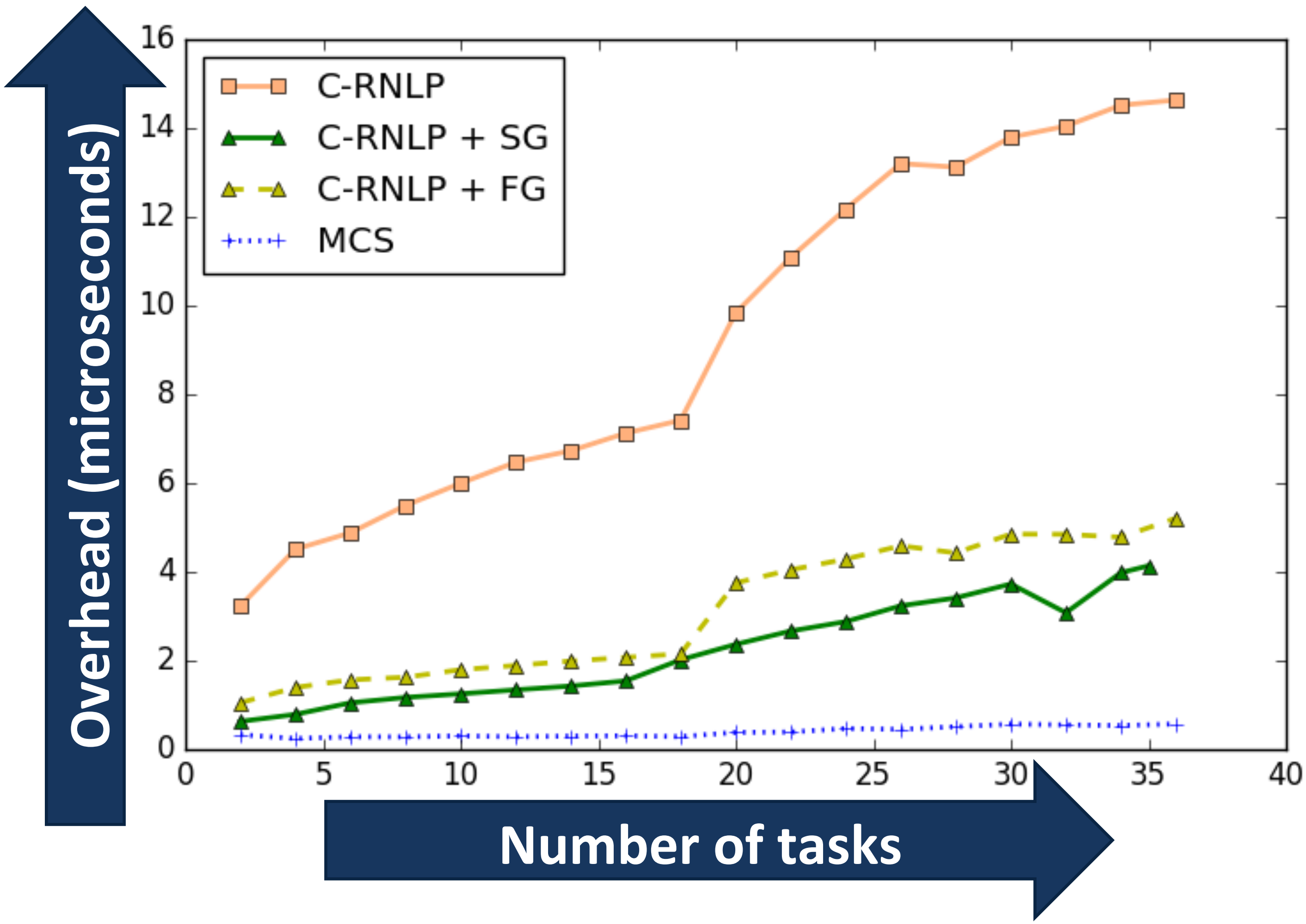
Experimental Evaluation

By how much can a lock server paradigm reduce worst-case overhead?

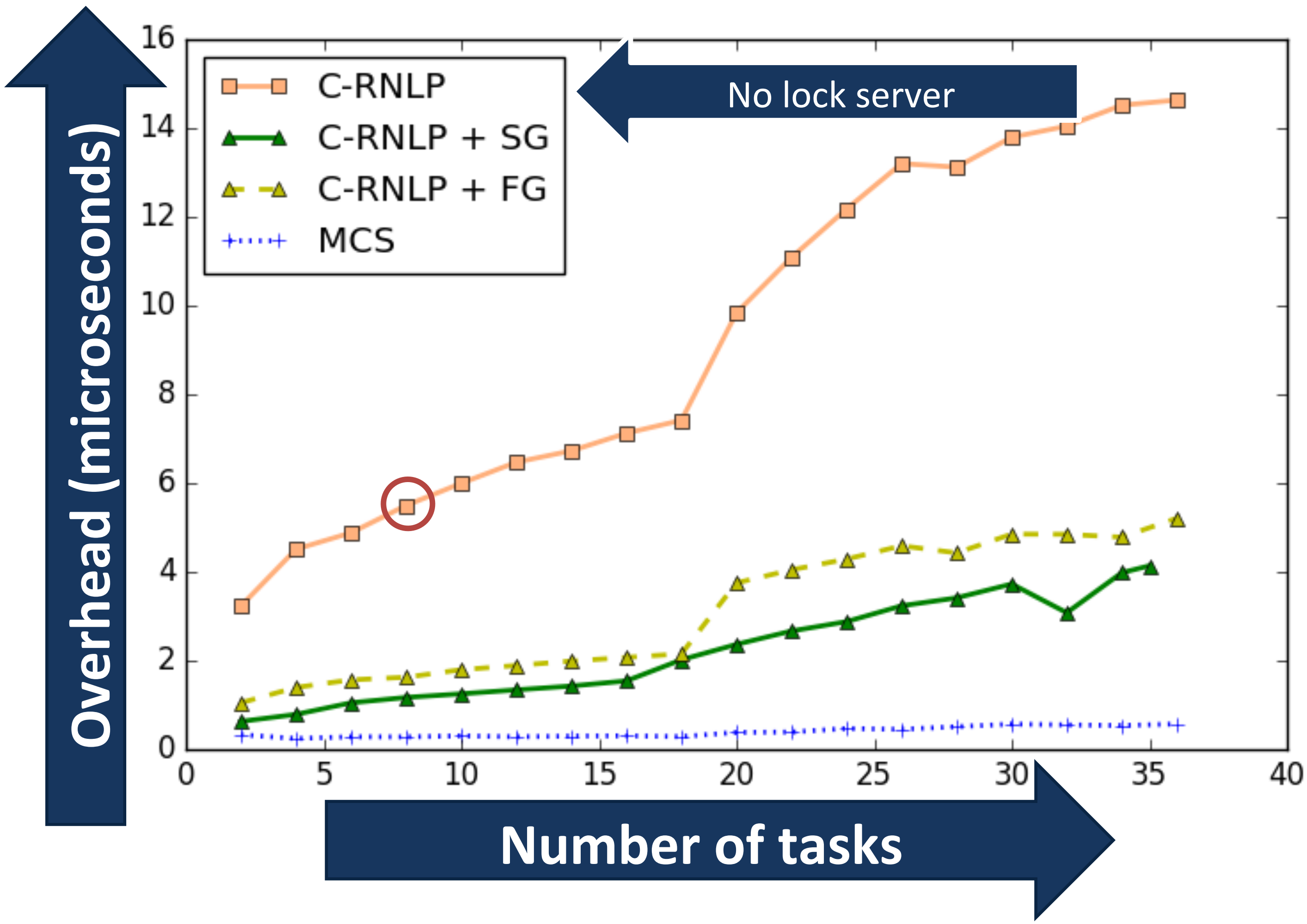
How do **overhead** and **blocking** differ between static and floating lock servers?

	Global	Local
Static	- Lose 1 core + L1 cache affinity	- Lose multiple cores + L1 cache affinity
Floating	- No guaranteed cache affinity	+ L3 cache affinity

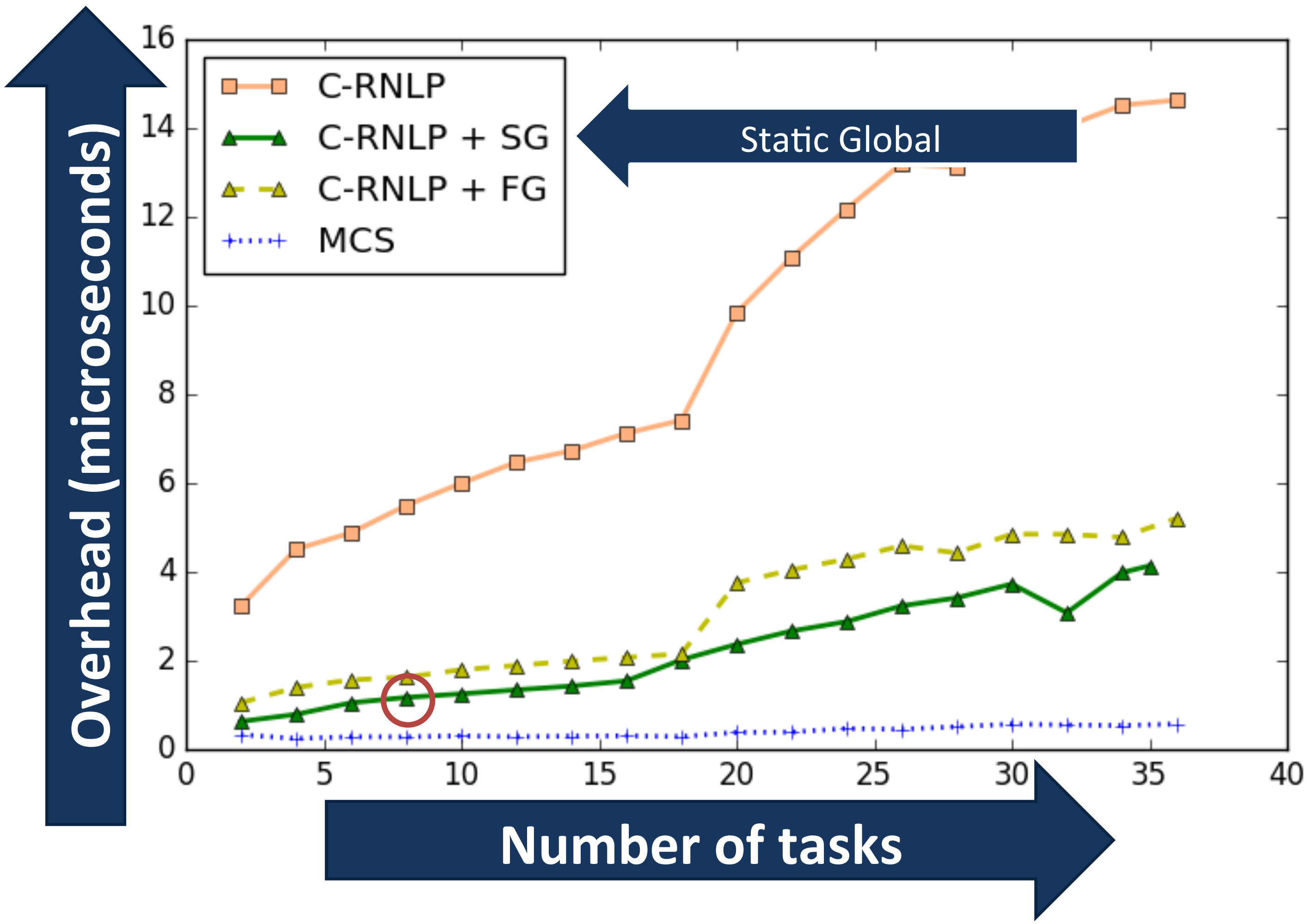
Experimental Results



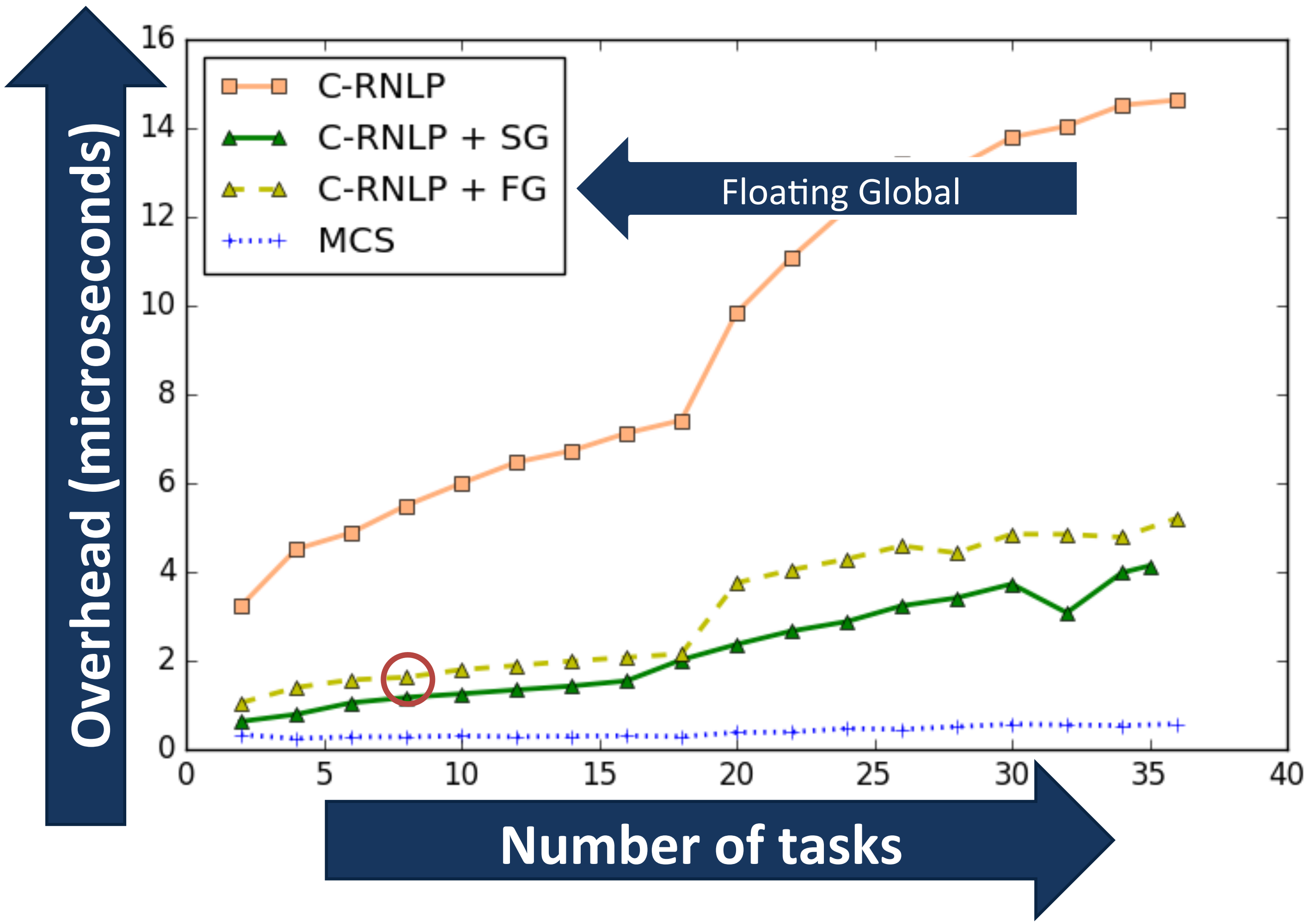
Experimental Results



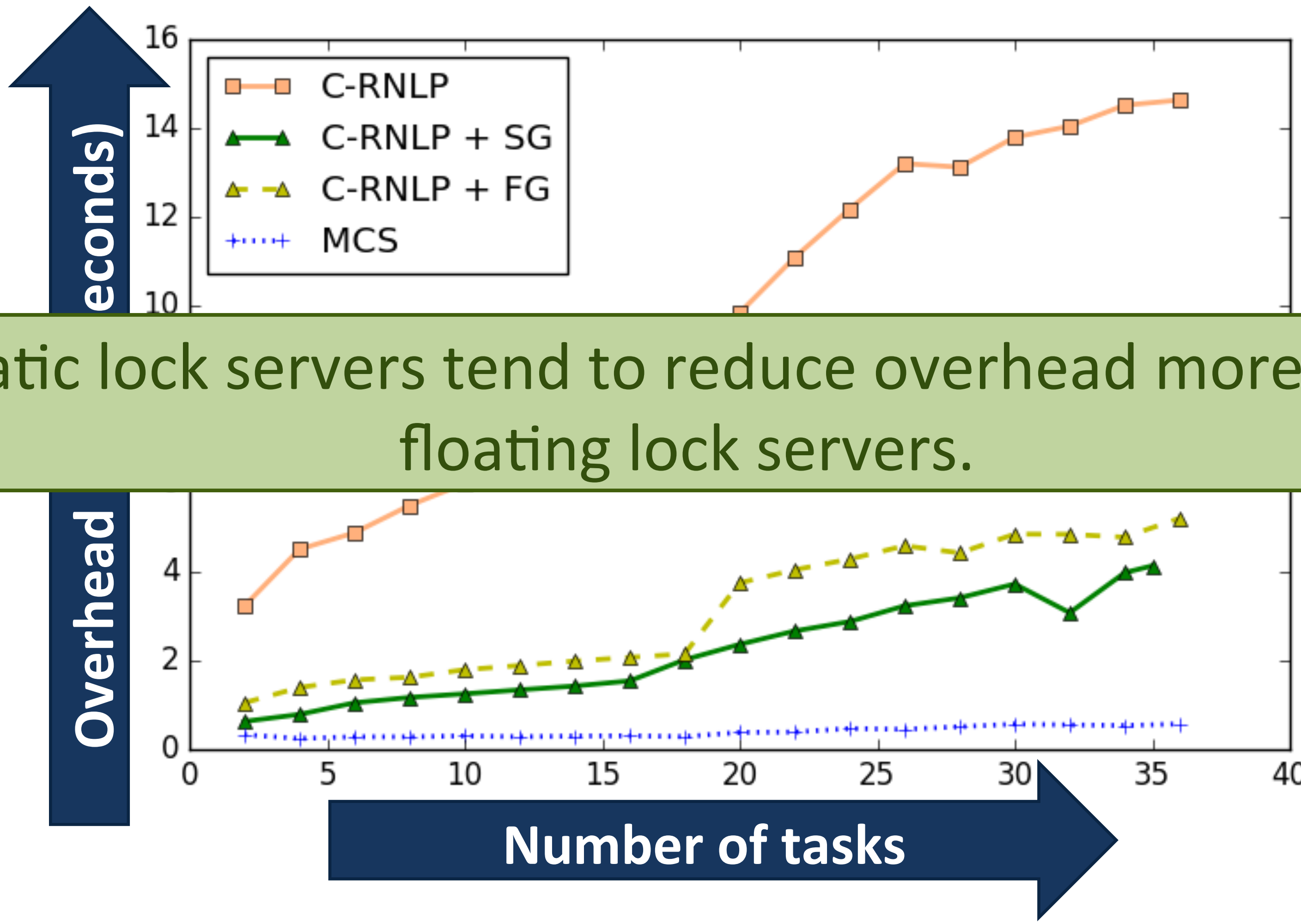
Experimental Results



Experimental Results

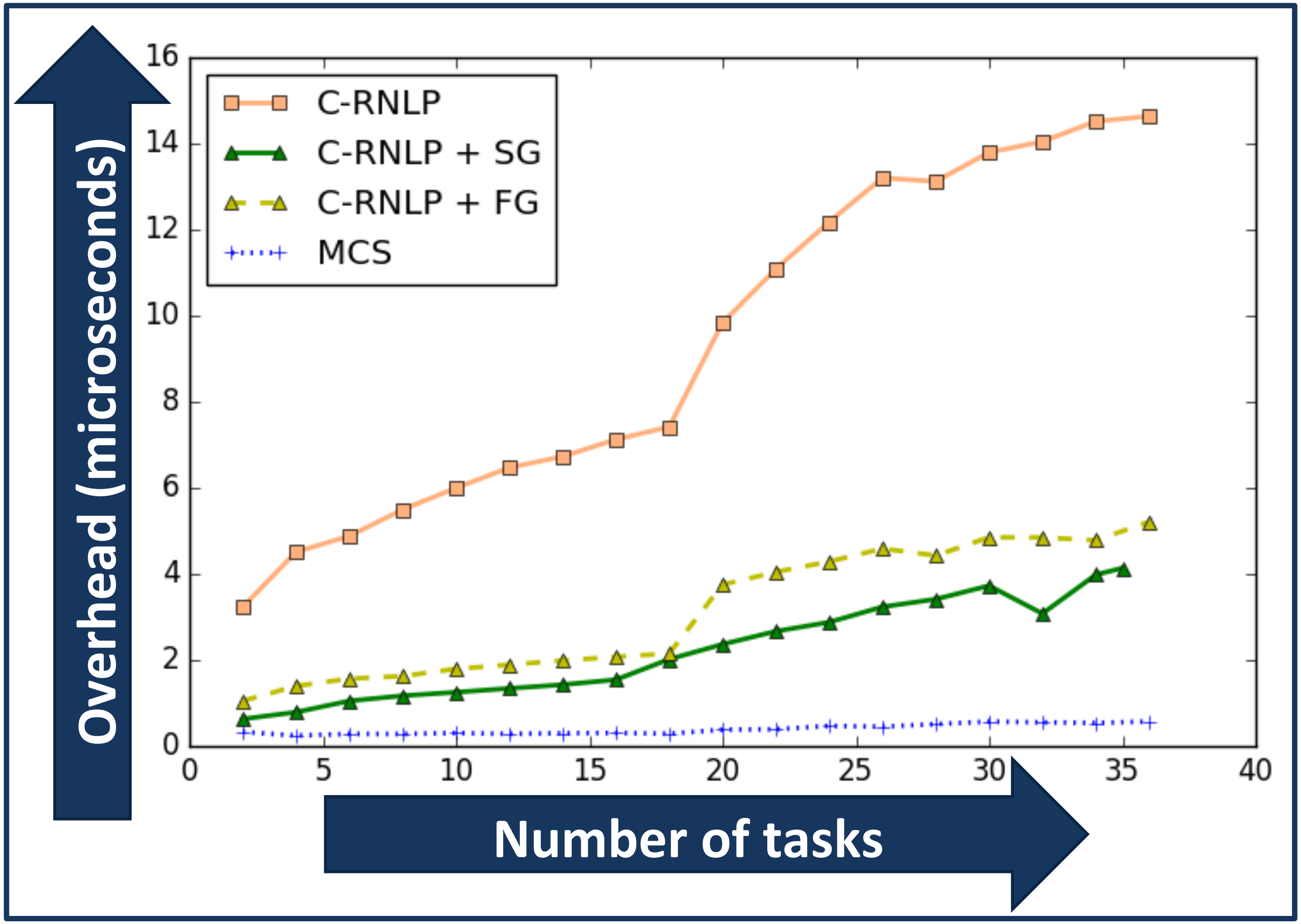


Experimental Results

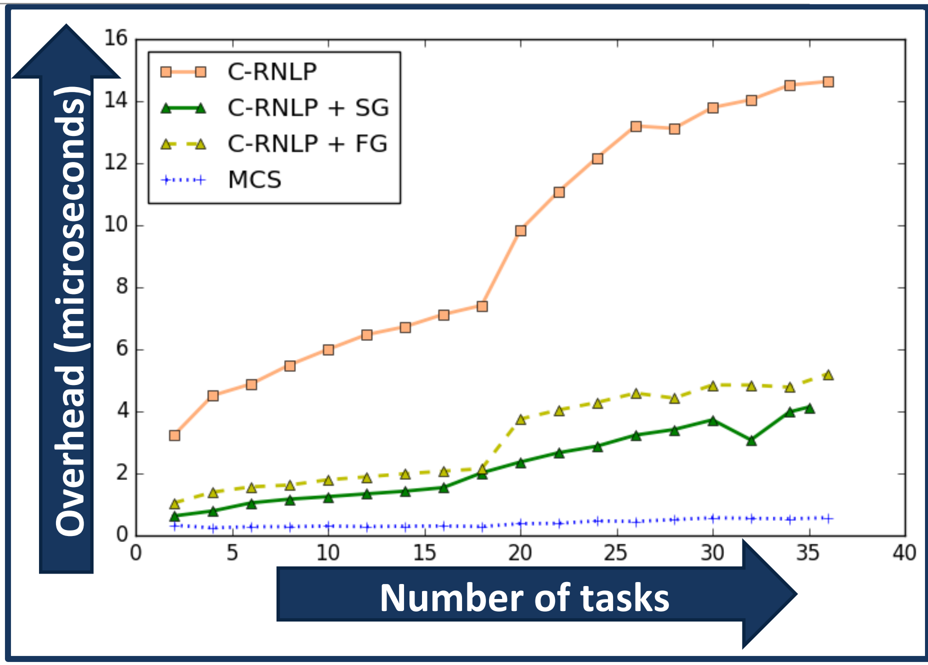


Static lock servers tend to reduce overhead more than floating lock servers.

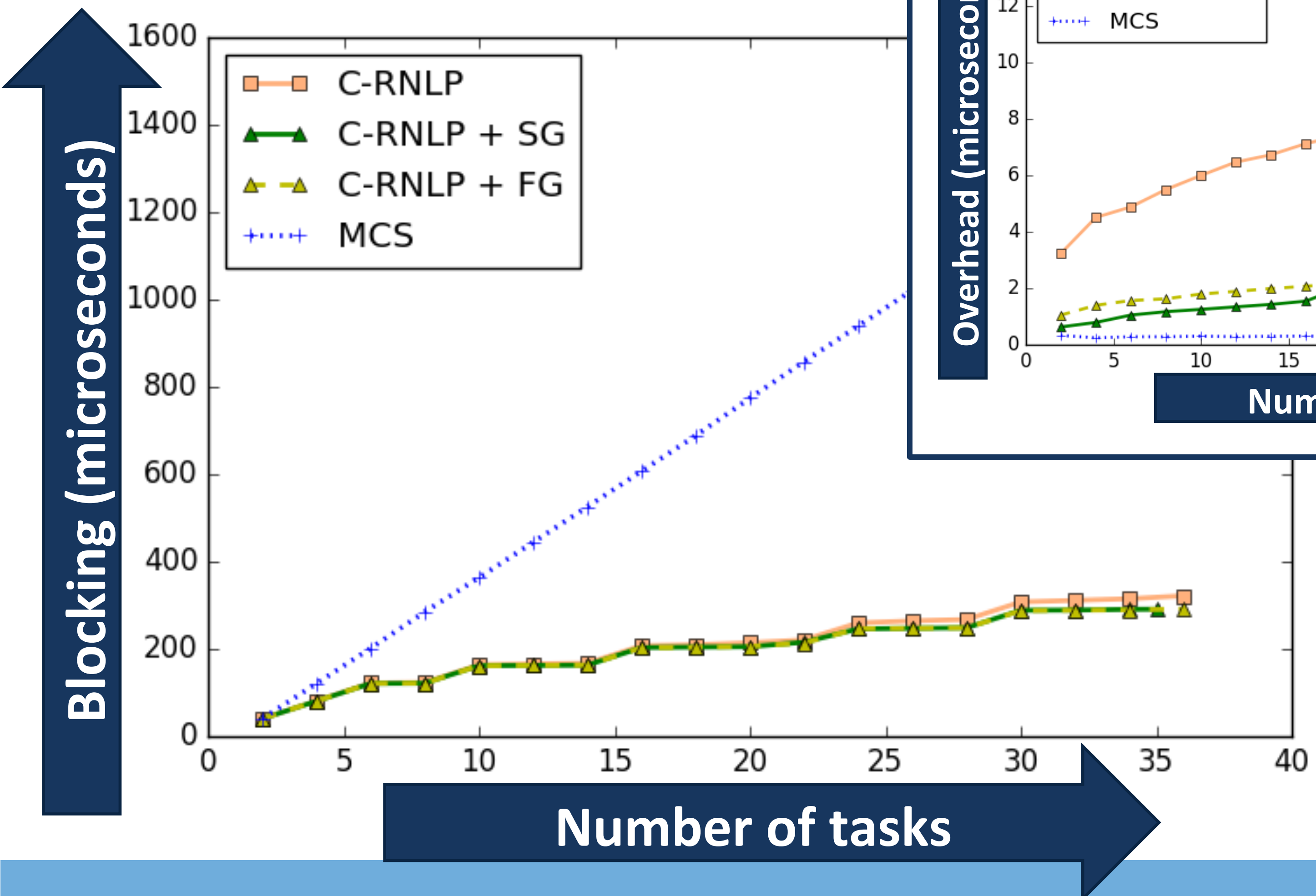
Experimental Results

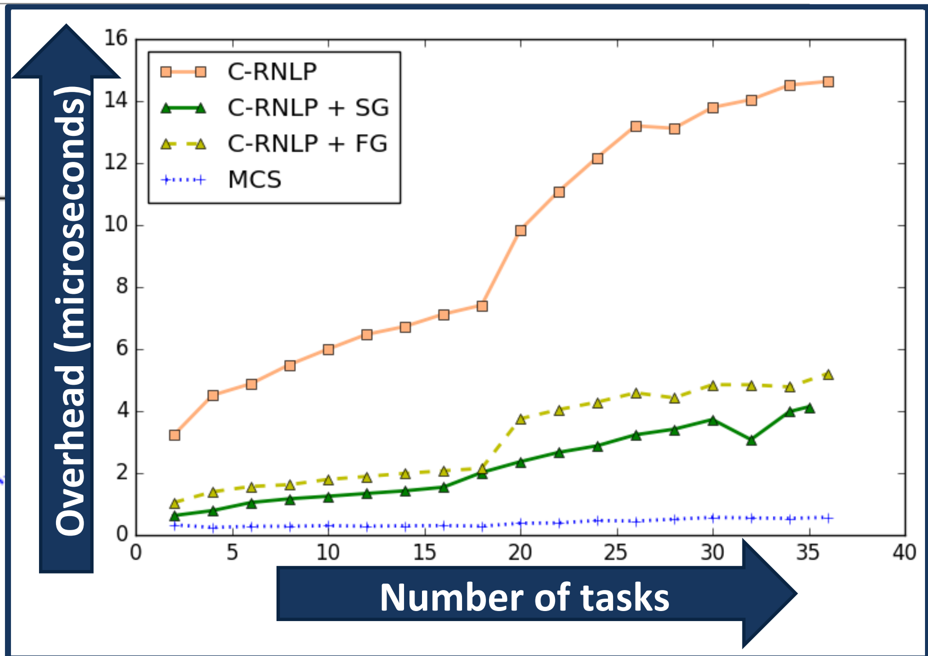


Experimental Results

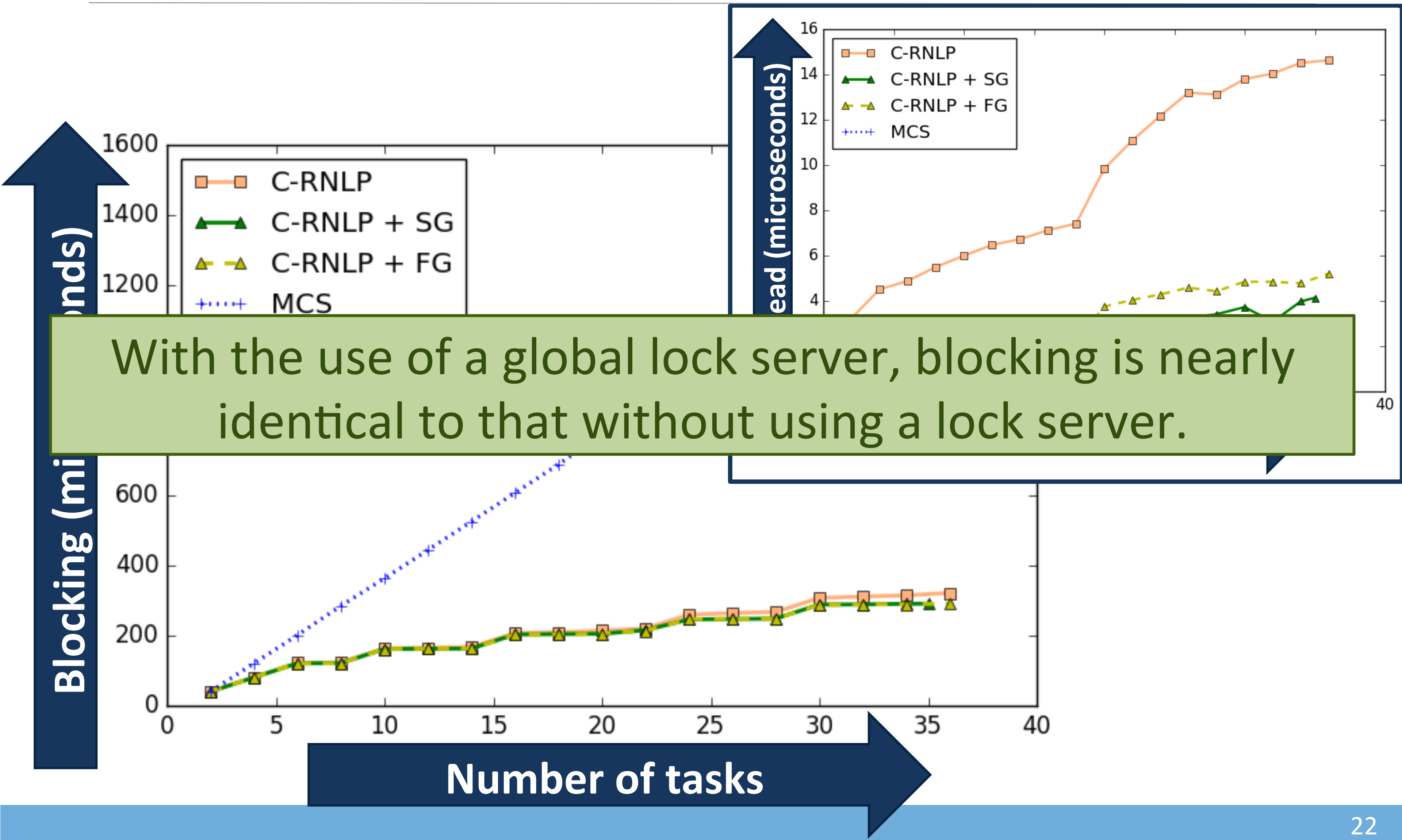


Experimental Results





Experimental Results



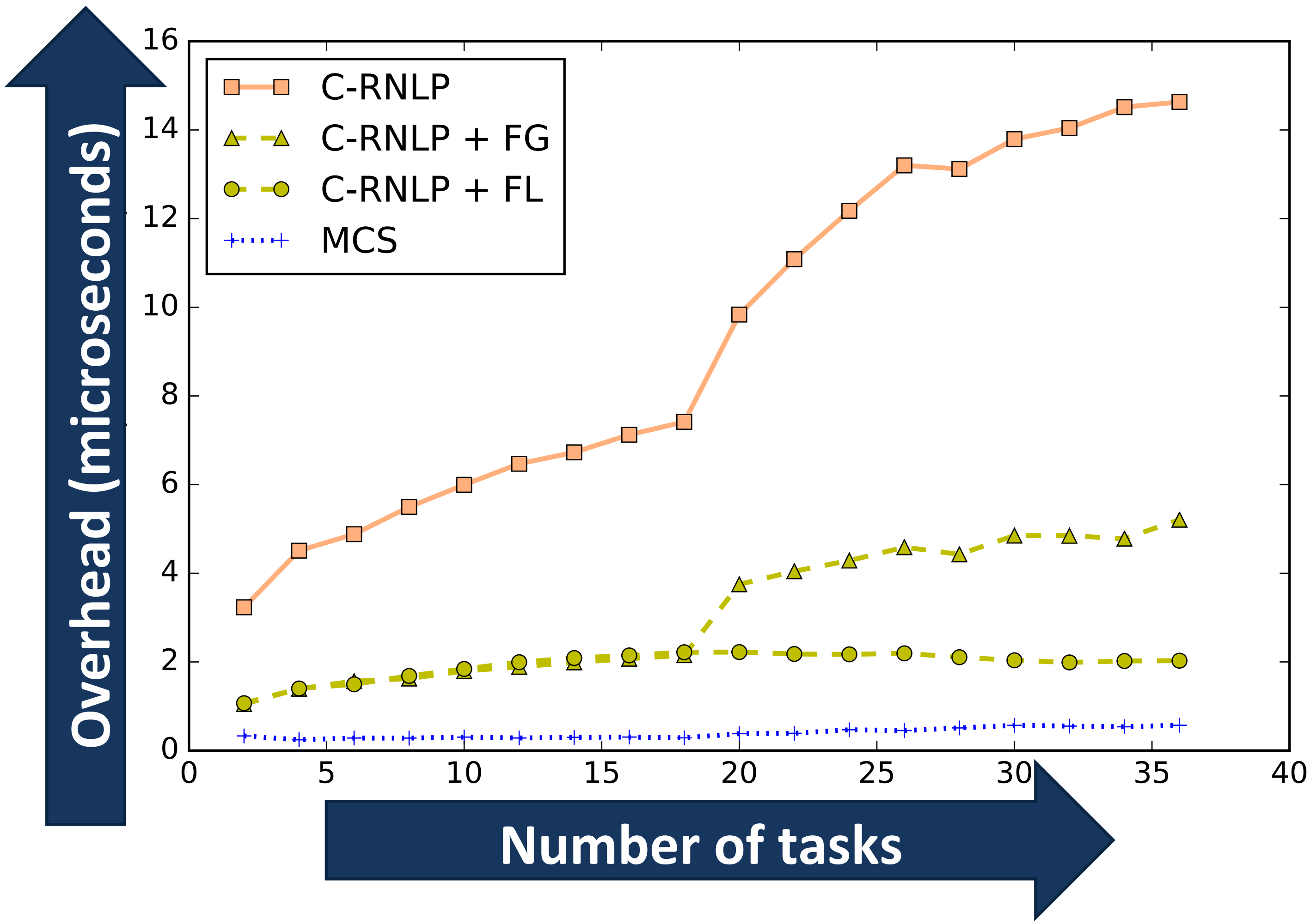
With the use of a global lock server, blocking is nearly identical to that without using a lock server.

Experimental Evaluation

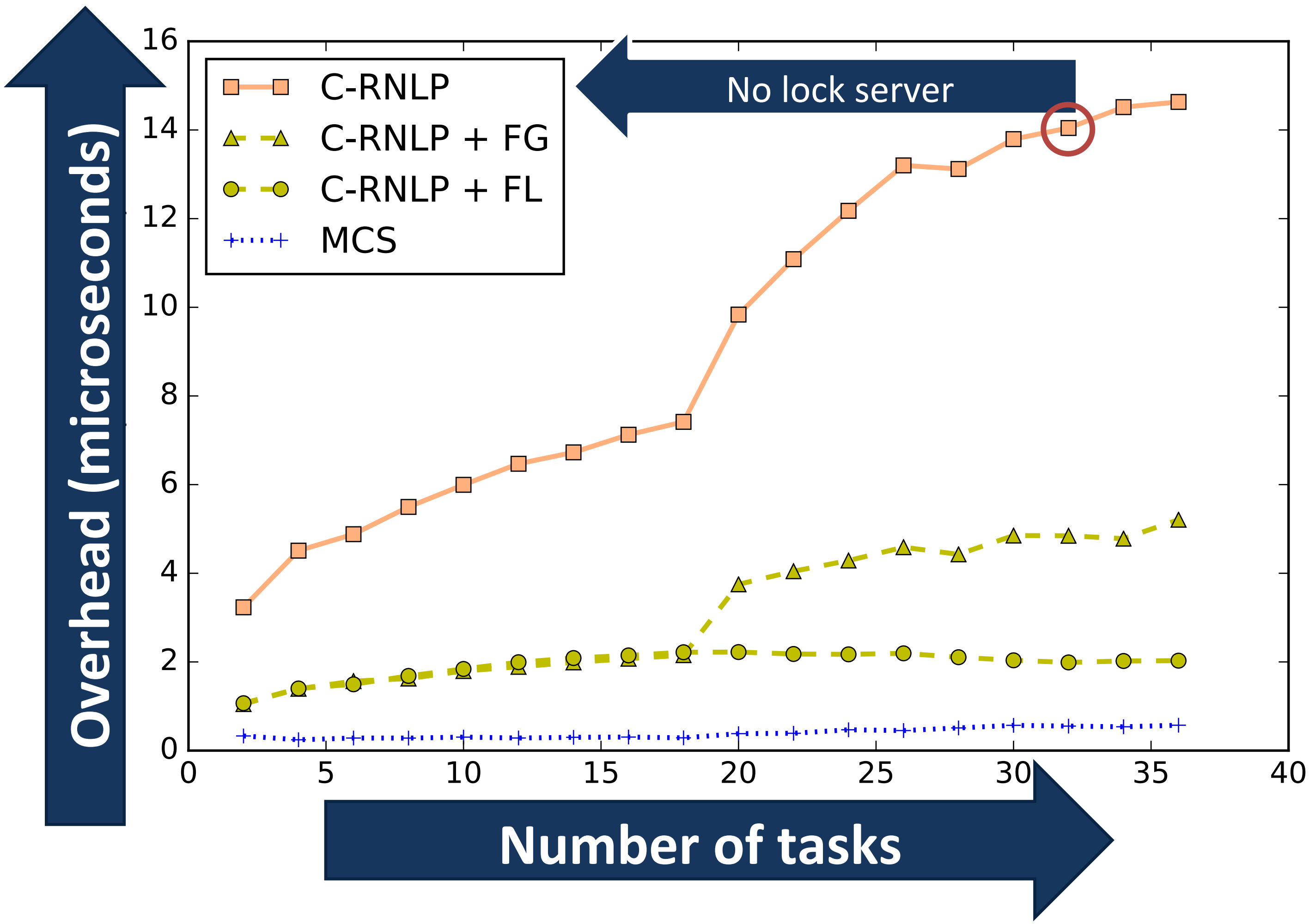
How do **overhead** and **blocking** differ between global and local lock servers?

	Global	**Local**
Static	- Lose 1 core + L1 cache affinity	- Lose multiple cores + L1 cache affinity
Floating	- No guaranteed cache affinity	+ L3 cache affinity

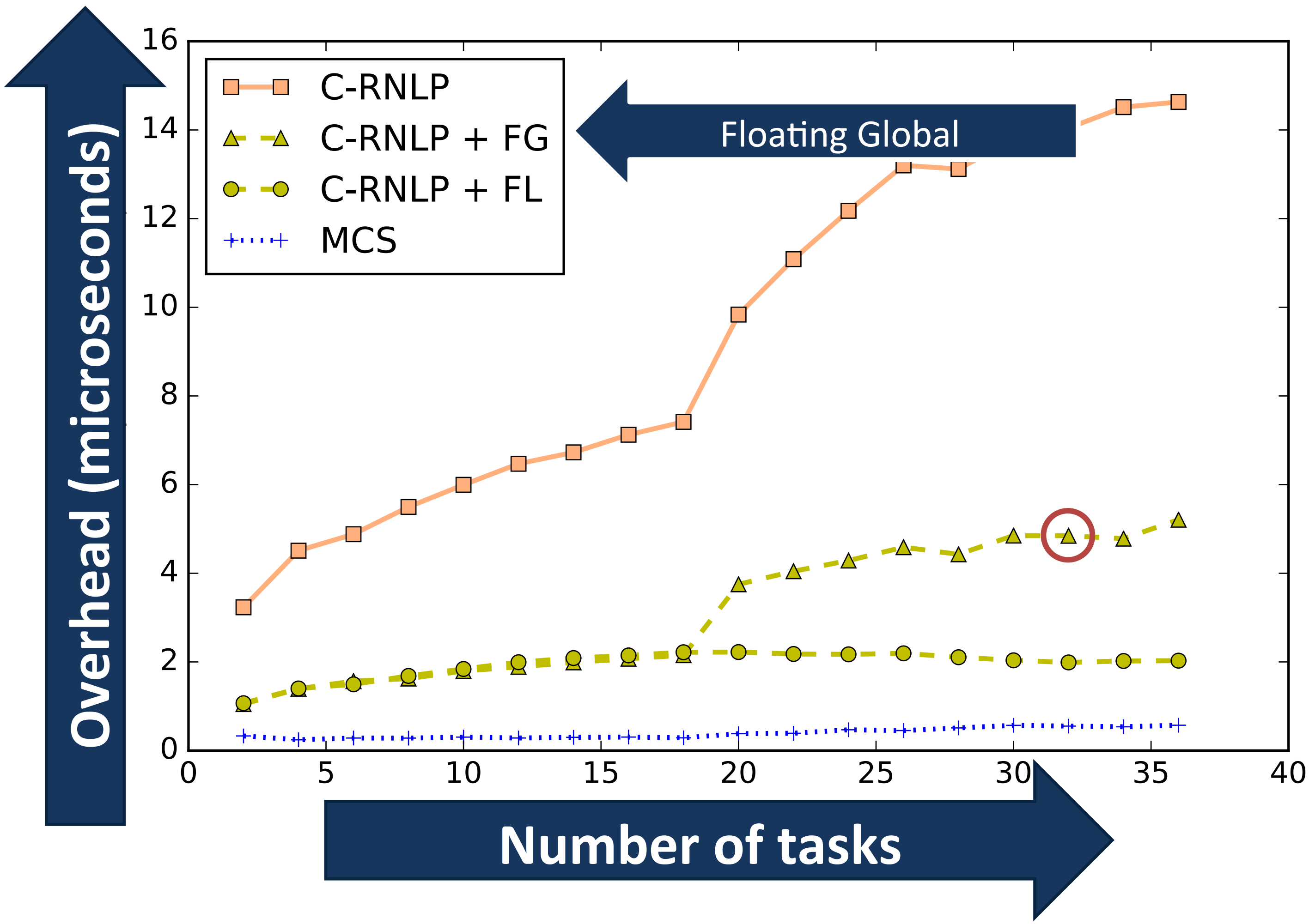
Experimental Results



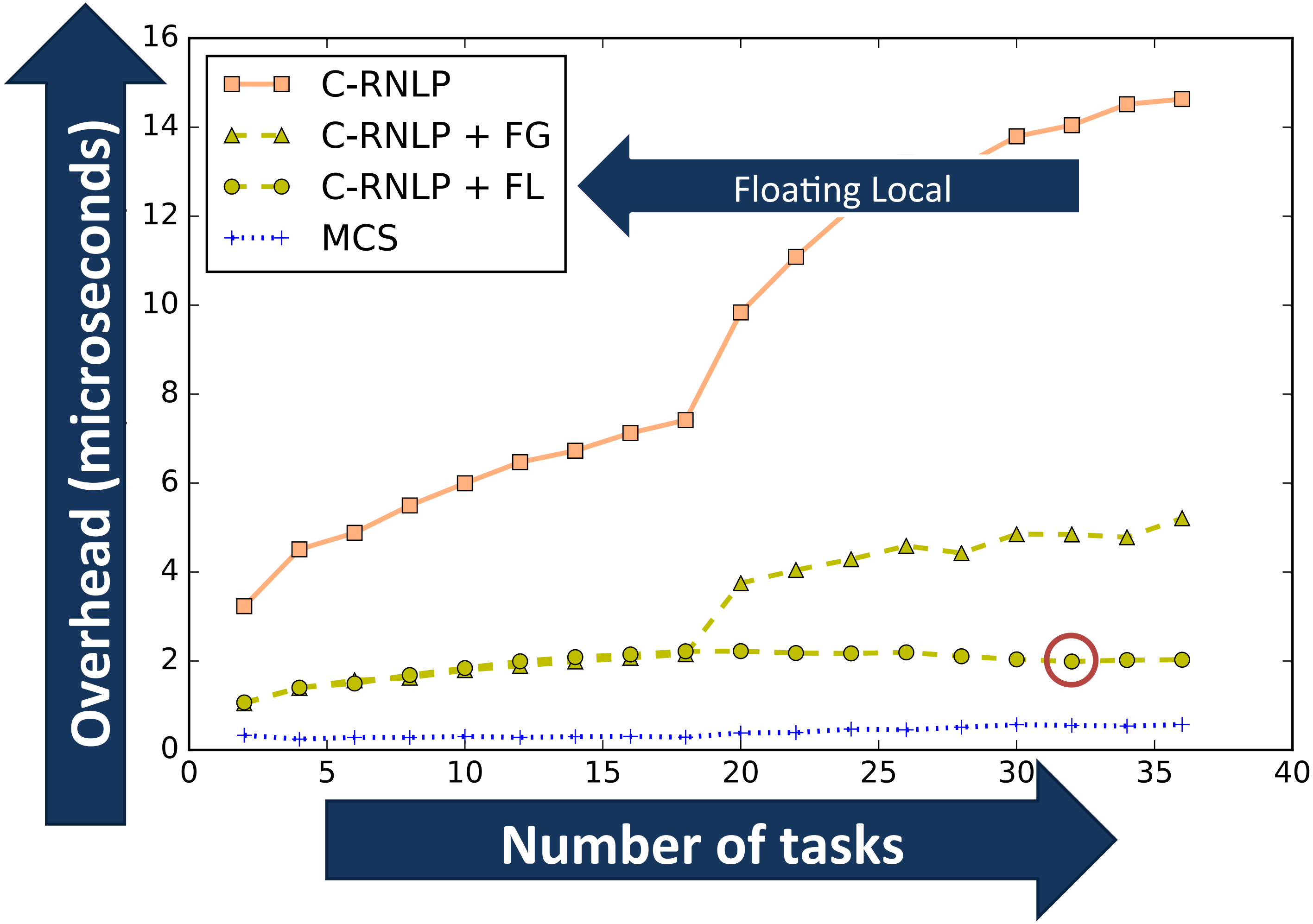
Experimental Results



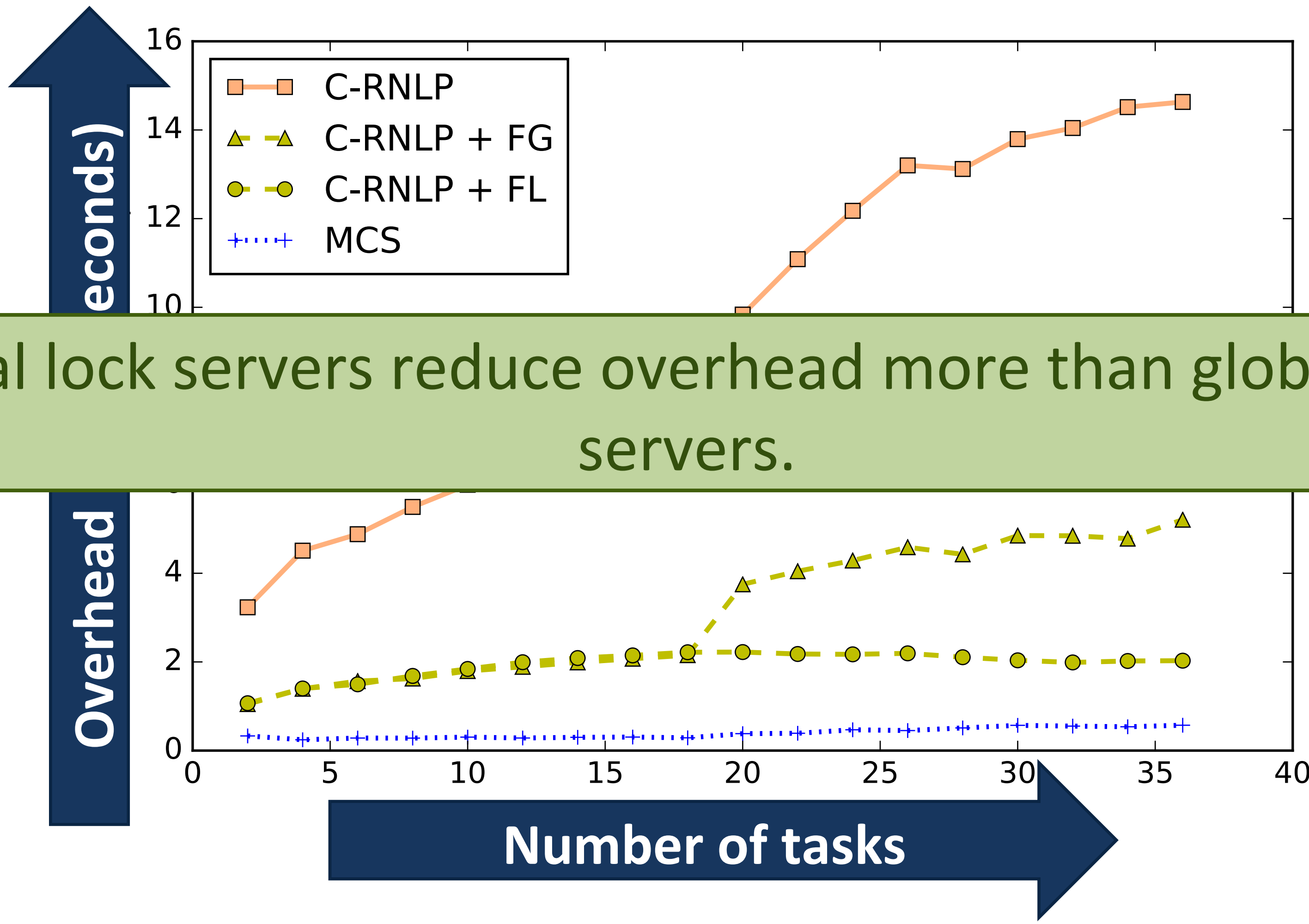
Experimental Results



Experimental Results

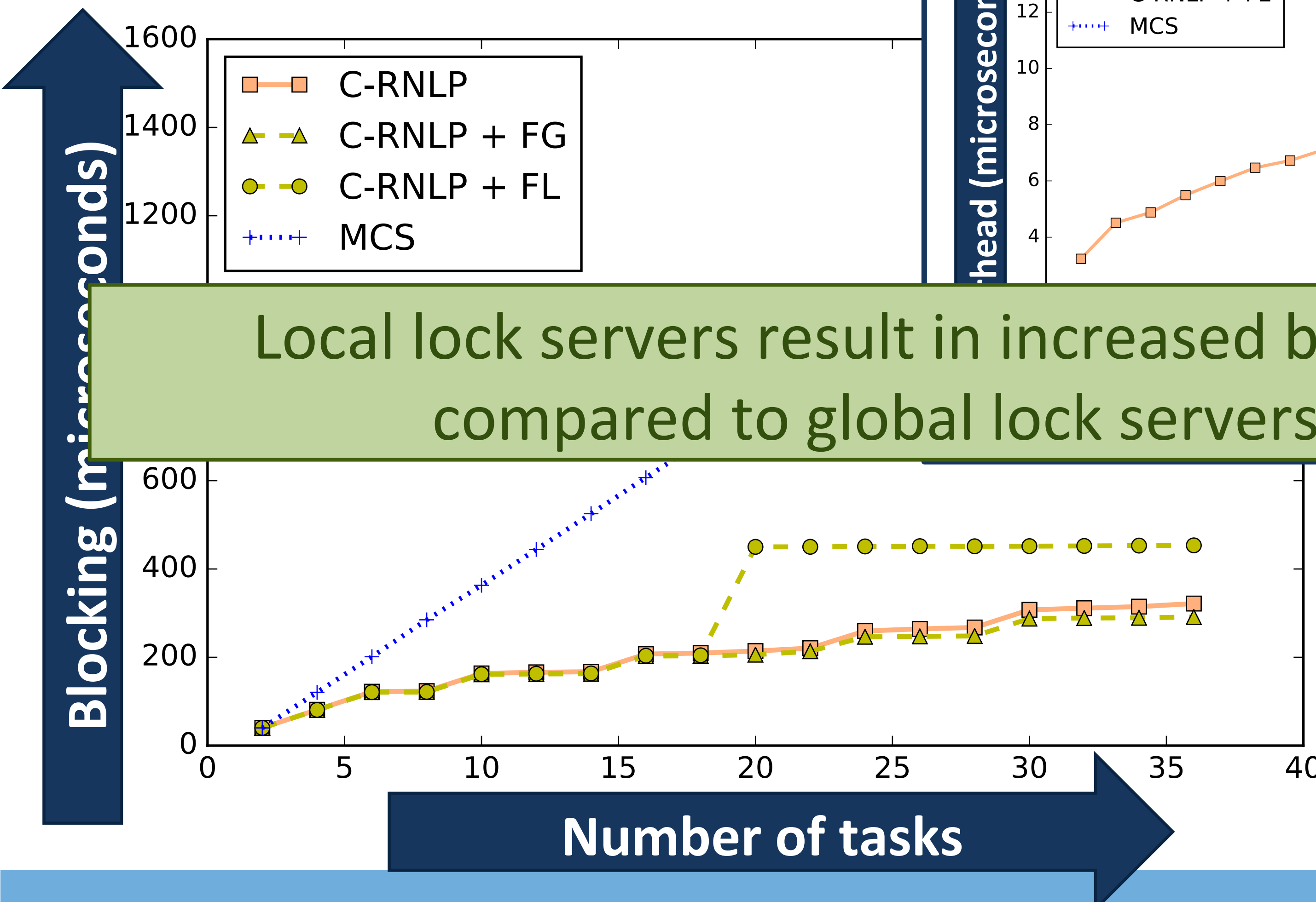


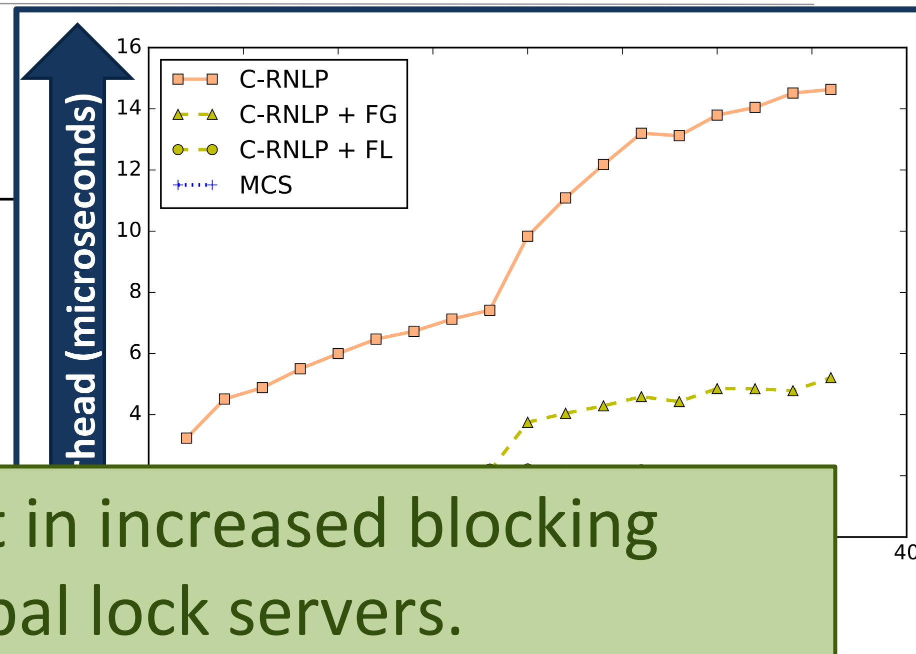
Experimental Results



Local lock servers reduce overhead more than global lock servers.

Experimental Results





Local lock servers result in increased blocking compared to global lock servers.

Experimental Evaluation

Measured **overhead** and **blocking**

One task per core issuing 10,000 random requests

Parameter sweep:

- Number of tasks: {2,4,…,36}
- Total # of resources: {16, 32, 64}
- # resources per request: {1,2,4,6,8,10}
- Critical-section lengths: {1, 20, 40, …, 100} µs

Contributions

Four lock server paradigms

- Implementation
- Evaluation

Lock server coordination protocol

Lock Server Coordination

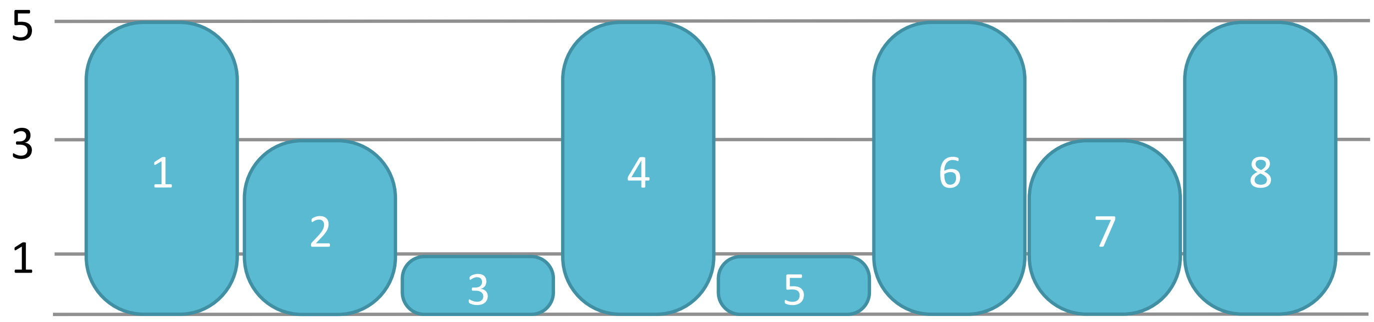
Original C-RNLP bound:

$$(c_i + 1)L_{MAX}$$

Two-server C-RNLP bound with R^2LP:

$$(c_{i,s} + 1)(L_{MAX,1} + L_{MAX,2})$$

C-RNLP bound



$(c_i + 1)L_{MAX} = (7 + 1)(5) = 40$ time units

40 time units

Arbitrary Split

Server 1 Server 2

Blocking

Server 1: (3 + 1)(5 + 5) = 40 — 40 time units

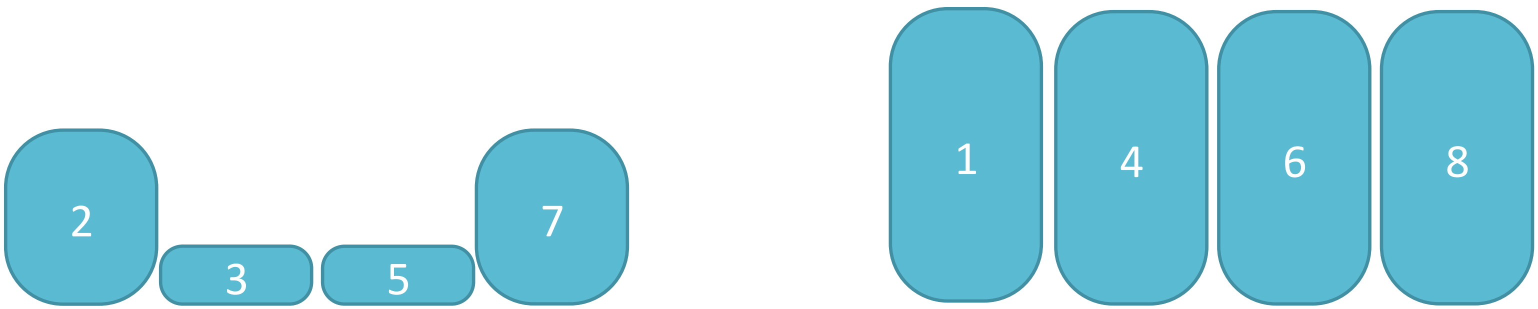
Server 2: (3 + 1)(5 + 5) = 40 — 40 time units

$$(c_{i,s} + 1)(L_{MAX,1} + L_{MAX,2})$$

Even Split by Critical-Section Length

Server 1

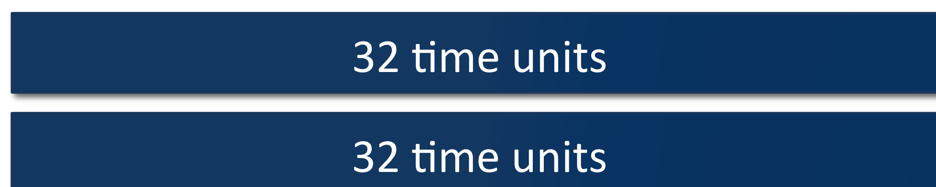
Server 2



Blocking

Server 1: $(3 + 1)(3 + 5) = 32$

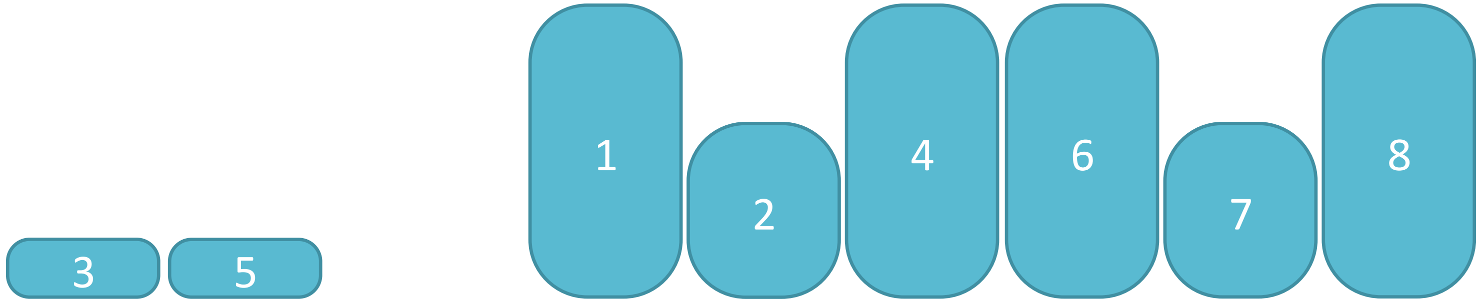
Server 2: $(3 + 1)(3 + 5) = 32$



$$(c_{i,s} + 1)(L_{MAX,1} + L_{MAX,2})$$

Uneven Split by Critical-Section Length

Server 1 Server 2



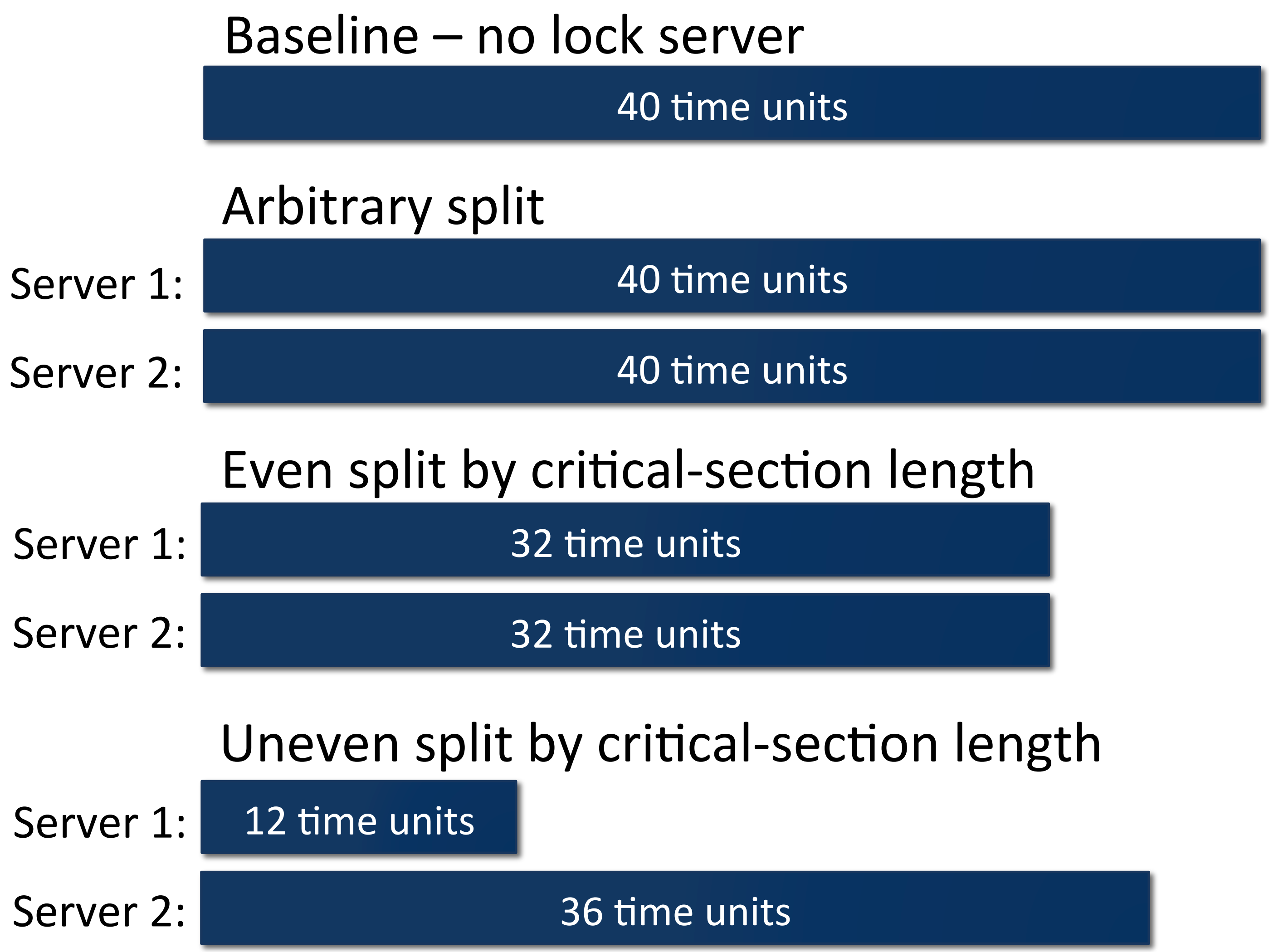
Blocking

Server 1: (1 + 1)(1 + 5) = 12

Server 2: (5 + 1)(1 + 5) = 36

$$(c_{i,s} + 1)(L_{MAX,1} + L_{MAX,2})$$

Blocking Bounds

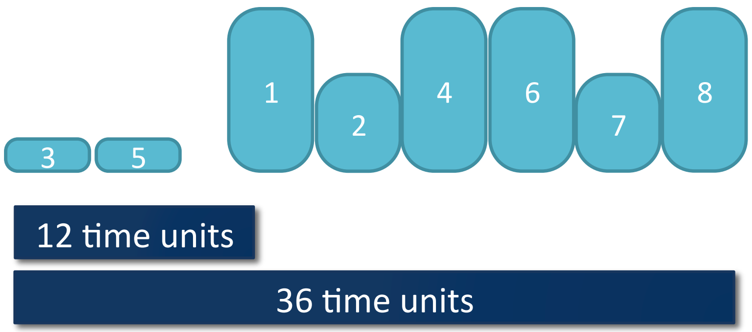


Future Work

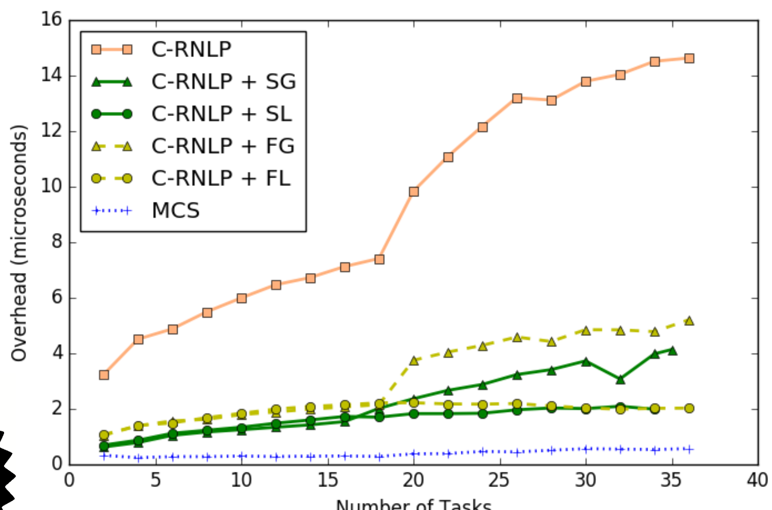
- Choose how to split tasks (based on requests) when using lock servers
- Explore accounting for static servers
  - Set server at highest priority, ensure lock state in cache
  - Use a dedicated IRQ-handling core
  - Treat as a special kind of interrupt
- Conduct a large-scale overhead-aware schedulability study

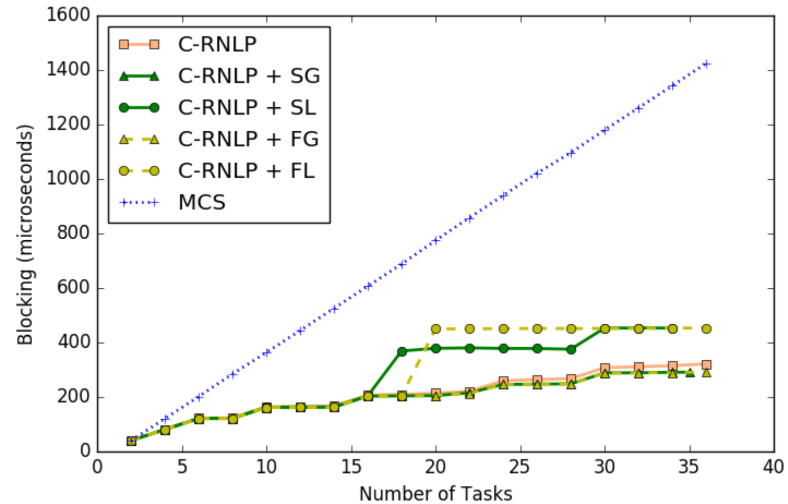
	Global	Local
Static	- Lose 1 core + L1 cache affinity	- Lose multiple cores + L1 cache affinity
Floating	- No guaranteed cache affinity	+ L3 cache affinity

Server Coordination: R^{2}LP



Questions?

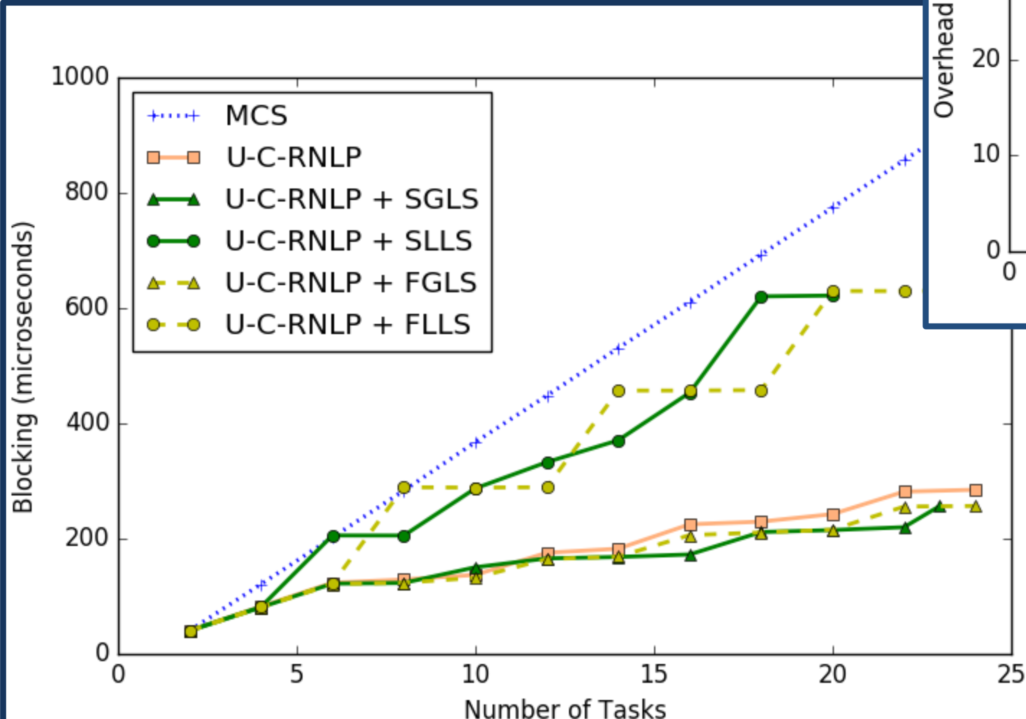


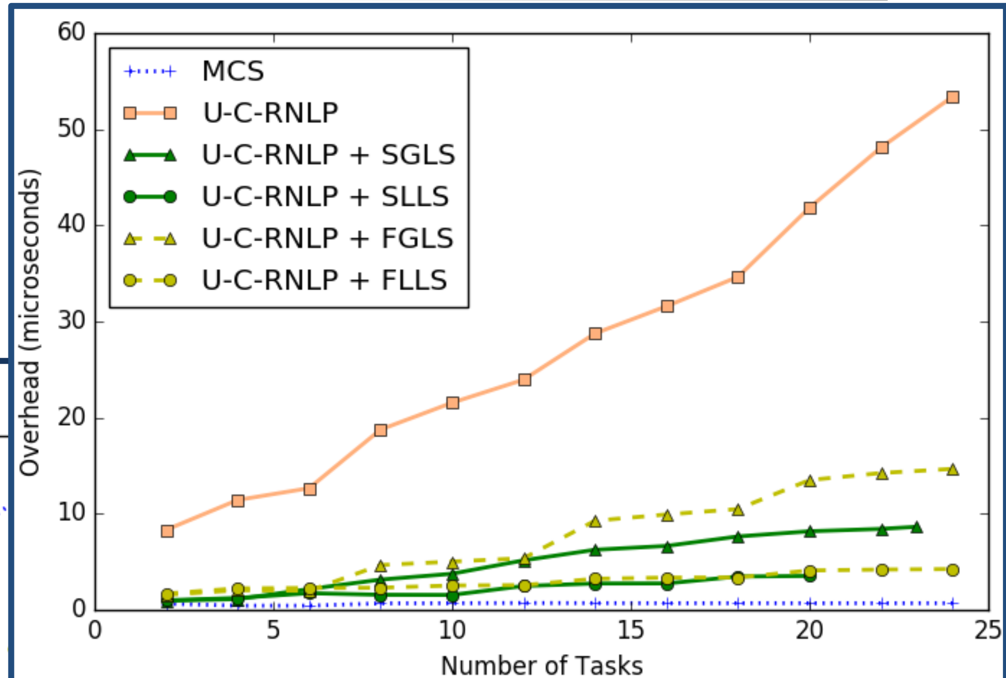


dx.doi.org/10.4230/DARTS.4.2.2

Case Study: Four Sockets

four-socket, 6-cores-per-socket
Intel Xeon L7455





L1 data: 32KB
L1 instruction: 32KB
L2: 3MB
L3: 12MB

Case Study: Profile of Requests

TABLE IV. RUNNABLE AVERAGE EXECUTION TIMES

Period	Average Execution Times in µs		
	Min.	Avg.	Max.
1 ms	0,34	5,00	30,11
2 ms	0,32	4,20	40,69
5 ms	0,36	11,04	83,38
10 ms	0,21	10,09	309,87
20 ms	0,25	8,74	291,42
50 ms	0,29	17,56	92,98
100 ms	0,21	10,53	420,43
200 ms	0,22	2,56	21,95
1000 ms	0,37	0,43	0,46

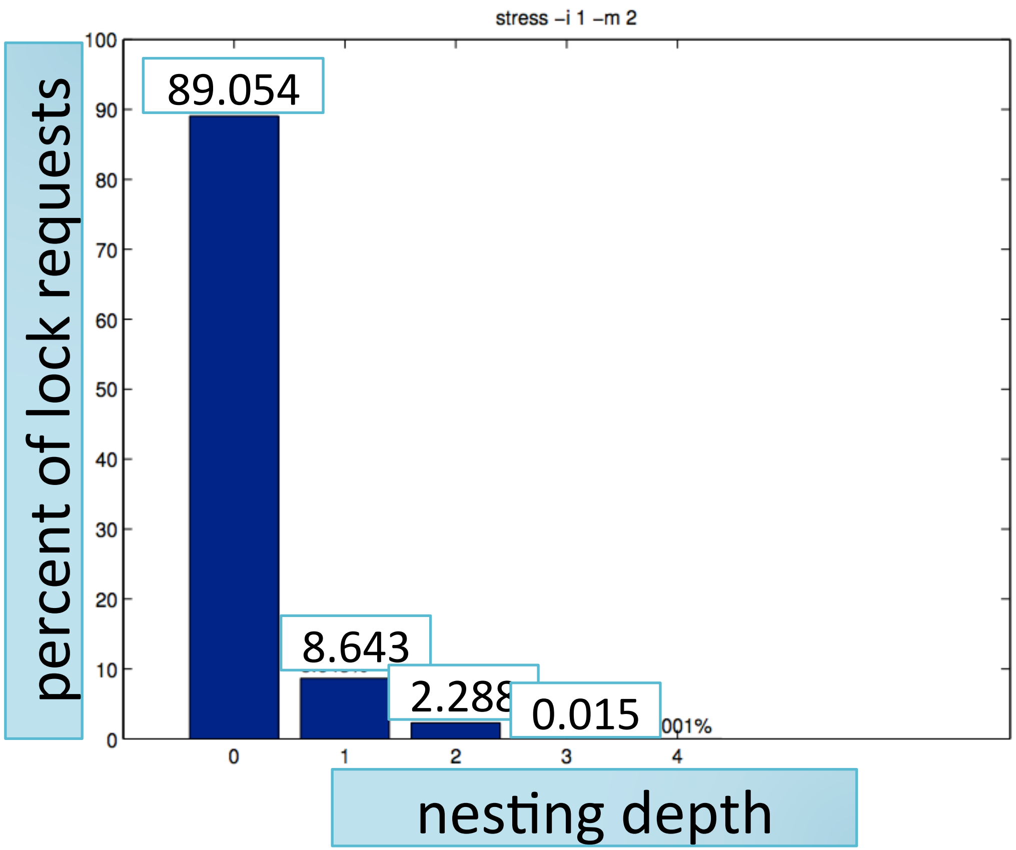
I	II	III	IV	V	VI
<10	10-50	51-100	100-500	501-1000	>1000

TABLE II. INTER-TASK COMMUNICATION

Period	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms	100 ms	200 ms	1000 ms	sync
1 ms				I	I		I			I
2 ms				I	I		I			
5 ms		I	IV	IV	II	II	I			
10 ms	II	II	II	VI	IV	II	IV	II	III	IV
20 ms	I	I	I	IV	VI	II	IV	I	II	IV
50 ms			II	II	II	III	I			
100 ms		I	I	V	IV	II	VI	II	III	IV
200 ms				I	I		I	I	I	
1000 ms				III	II		III	I	IV	I
Angle-sync	I	I	I	IV	IV	I	III	I	I	V

S. Kramer, D. Ziegenbein, and A. Hamann. Real world automotive benchmarks for free. *WATERS 2015.*

Case Study: Nested Requests



B. Brandenburg and J. Anderson. Feather-trace: A lightweight event tracing toolkit. *OSPERT 2007*.

Handling Nested Requests

Non-nested lock request

```
lock(A)
    //critical section
unlock(A)
```

With Dynamic Group Locks (DGLs)

```
lock(A)
    //critical section
unlock(A)
```

Nested lock request

```
lock(A)
    lock(B)
      //critical section
    unlock(B)
unlock(A)
```

```
lock(A,B)
    //critical section
unlock(A,B)
```

Additional Experimental Results

	U-C-RNLP	U-C-RNLP + SGLS	U-C-RNLP + SLLS	U-C-RNLP + FGLS	G-C-RNLP + SGLS
Total Firsts	0	92	0	23	12
Total Seconds	1	26	18	70	4
Total Thirds	68	2	17	20	8
Total	69	120	35	113	24

- **Figure 16** Results of total request time comparison.

Challenge: Blocking Chains

Sequential resource acquisition: critical-section length $O(m^D)$

H. Takada and K. Sakamura. Real-time scalability of nested spin locks. *RTCSA 1995*.

Dynamic group locks: no worst-case critical-section inflation, same asymptotic bounds

B. Ward and J. Anderson. Supporting nested locking in multiprocessor real-time systems. *ECRTS 2012.*