A Contention-Sensitive Fine-Grained Locking Protocol for Multiprocessor Real-Time Systems

CATHERINE E. JARRETT, BRYAN C. WARD, AND JAMES H. ANDERSON

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Outline

Problem

Wait-For Graph Model

C-RNLP

Results

Future Work

Outline

Problem

Wait-For Graph Model

C-RNLP

Results

Future Work

Multiprocessor Platform

Considering: nested lock requests

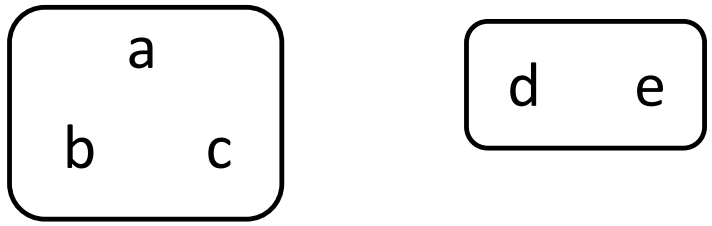
Implementation: *coarse-grained locking* or *fine-grained locking*

Multiprocessor Platform

Considering: nested lock requests

Implementation: ***coarse-grained locking*** or *fine-grained locking*

```
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```



Multiprocessor Platform

Considering: nested lock requests

Implementation: ***coarse-grained locking*** or *fine-grained locking*

```
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```

Multiprocessor Platform

Considering: nested lock requests

Implementation: *coarse-grained locking* or ***fine-grained locking***

```
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```

a c e

b d

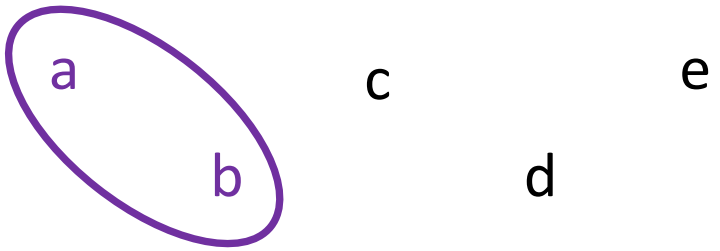
dynamic group locking

Multiprocessor Platform

Considering: nested lock requests

Implementation: *coarse-grained locking* or ***fine-grained locking***

```
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```



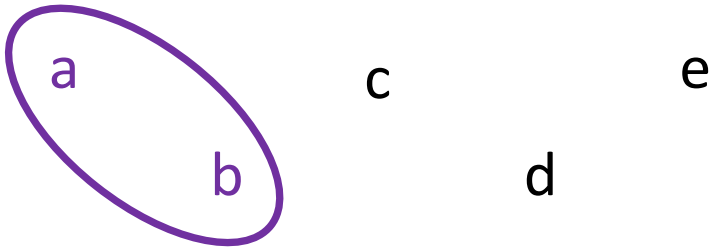
dynamic group locking

Multiprocessor Platform

Considering: nested lock requests

Implementation: *coarse-grained locking* or ***fine-grained locking***

```
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```




```
lock(c);
...
lock(d);
...
unlock(d);
...
unlock(c);
```

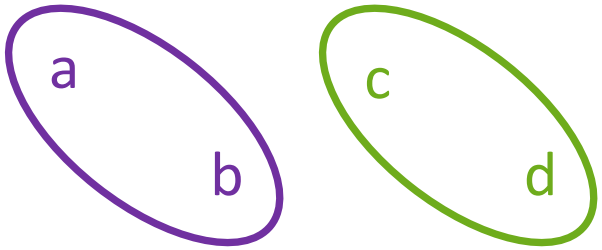
dynamic group locking

Multiprocessor Platform

Considering: nested lock requests

Implementation: *coarse-grained locking* or ***fine-grained locking***

```
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```



e

```
lock(c);
...
lock(d);
...
unlock(d);
...
unlock(c);
```

dynamic group locking

Real-time Nested Locking Protocol (RNLP)

Processor count, *m*

Task count, *n*

Real-time Nested Locking Protocol (RNLP)

Processor count, *m*

Task count, *n*

Minimize: priority-inversion blocking (pi-blocking)

Real-time Nested Locking Protocol (RNLP)

Processor count, *m*

Task count, *n*

Minimize: priority-inversion blocking (pi-blocking)

Coarse-grained locking: *O(m)* or *O(n)* pi-blocking per request

Real-time Nested Locking Protocol (RNLP)

Processor count, *m*

Task count, *n*

Minimize: priority-inversion blocking (pi-blocking)

Coarse-grained locking: *O(m)* or *O(n)* pi-blocking per request

RNLP: *O(m)* or *O(n)* pi-blocking per request

Real-time Nested Locking Protocol (RNLP)

Contention: the number of existing requests that access the same resources as the request of interest

Coarse-grained locking: *O(m)* or *O(n)* pi-blocking per request

RNLP: *O(m)* or *O(n)* pi-blocking per request

Real-time Nested Locking Protocol (RNLP)

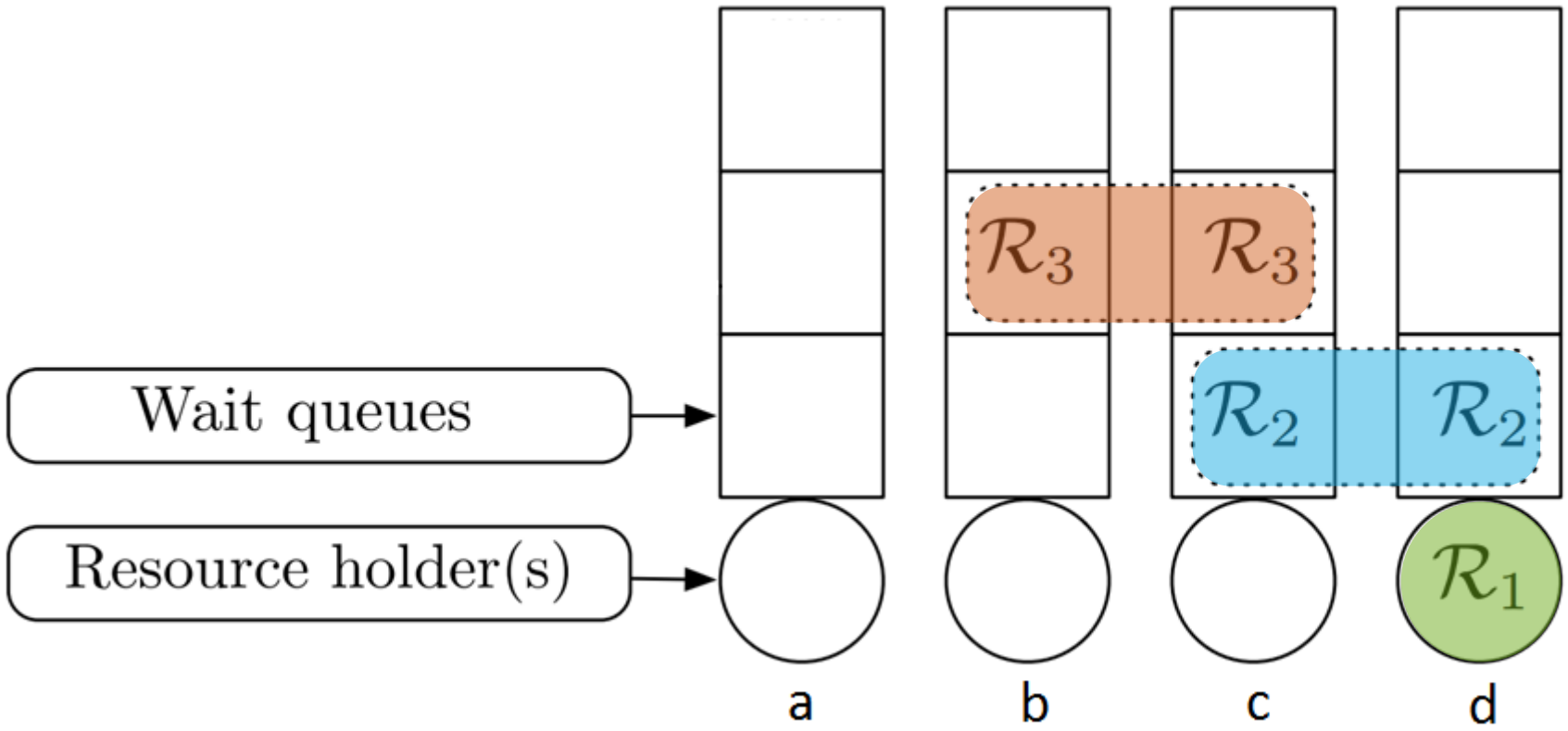
Contention: the number of existing requests that access the same resources as the request of interest

Coarse-grained locking: *O(m)* or *O(n)* pi-blocking per request

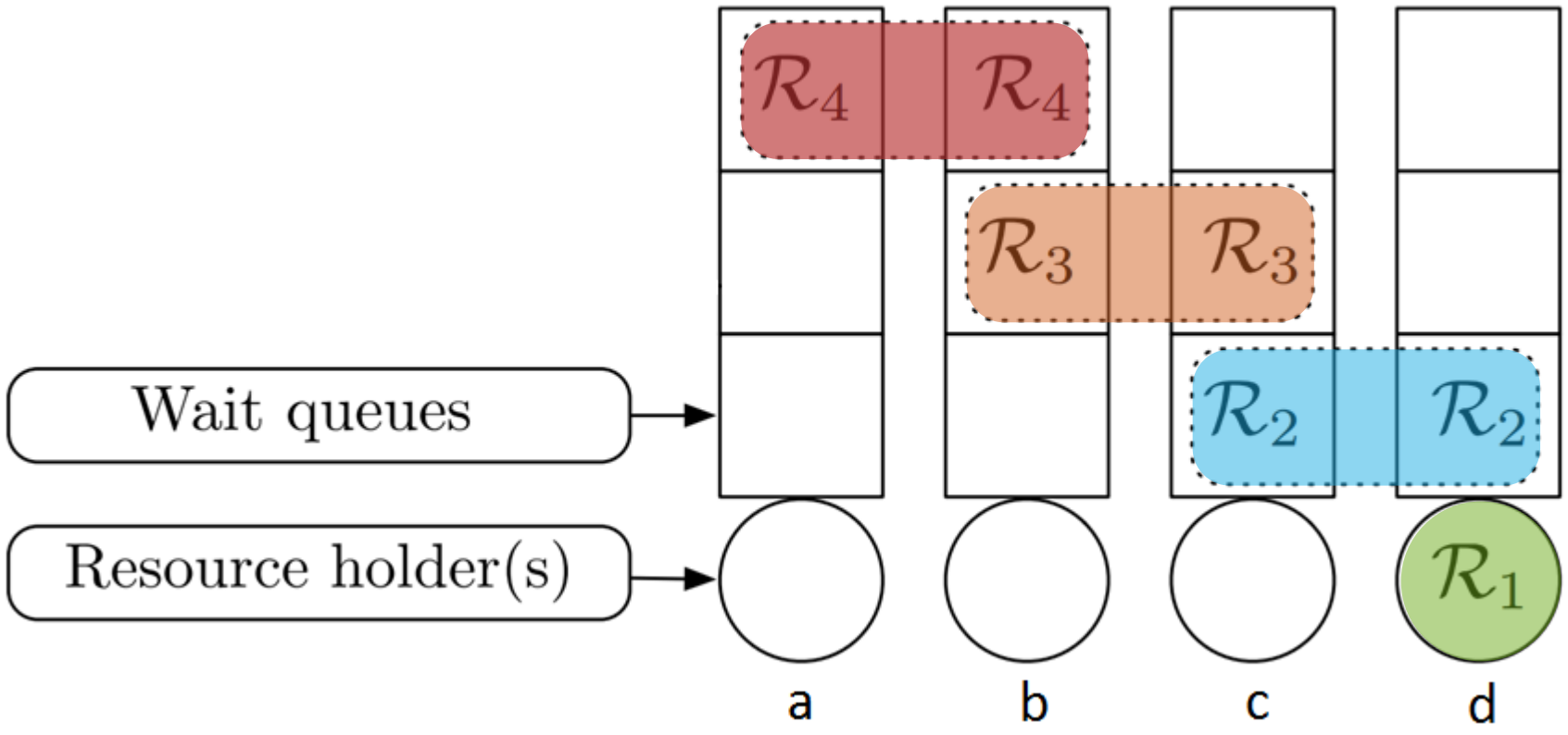
RNLP: *O(m)* or *O(n)* pi-blocking per request

C-RNLP Goal: contention sensitive pi-blocking

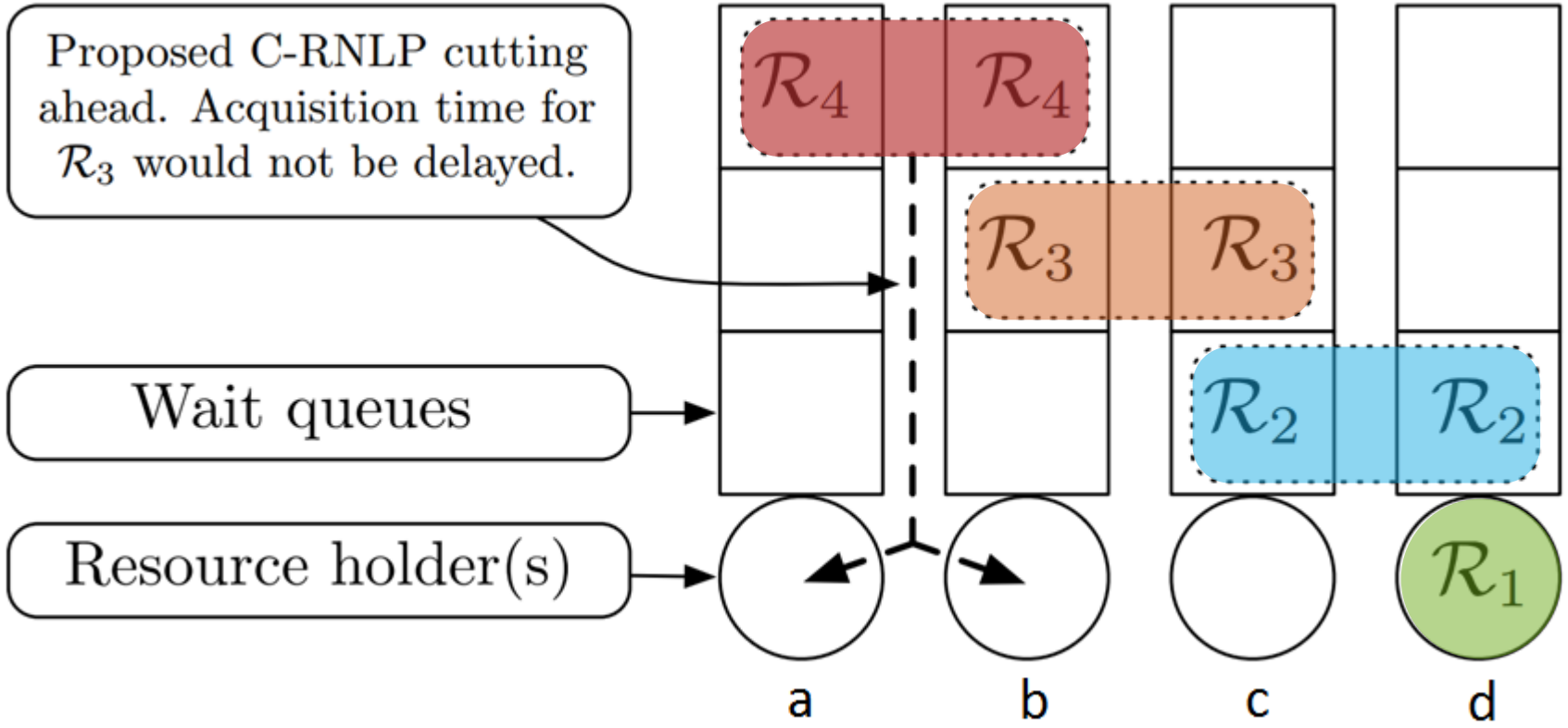
Key Problem: transitive blocking chains



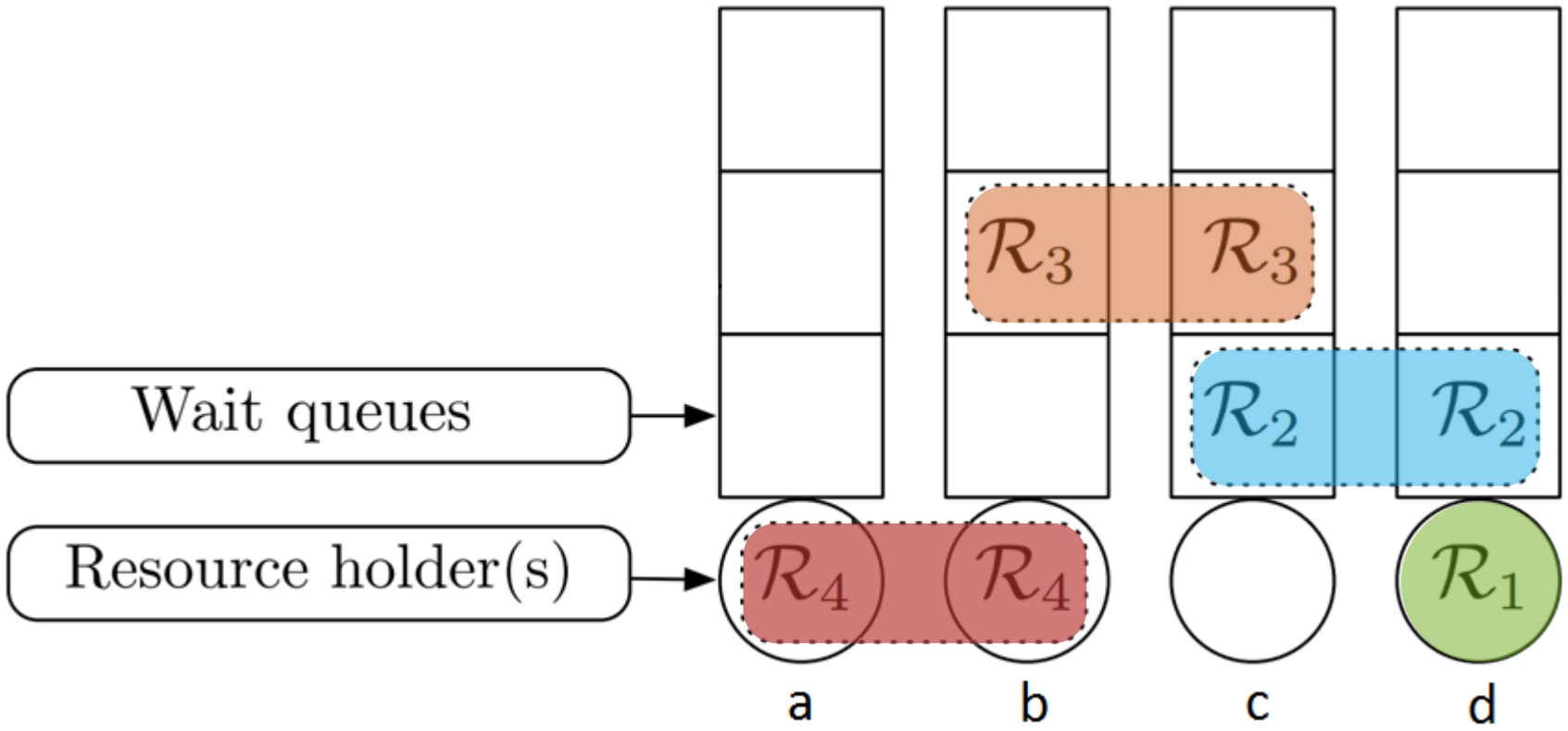
Key Problem: transitive blocking chains



Key Problem: transitive blocking chains



Key Problem: transitive blocking chains



Outline

Problem

Wait-For Graph Model

C-RNLP

Results

Future Work

Outline

Problem

Wait-For Graph Model

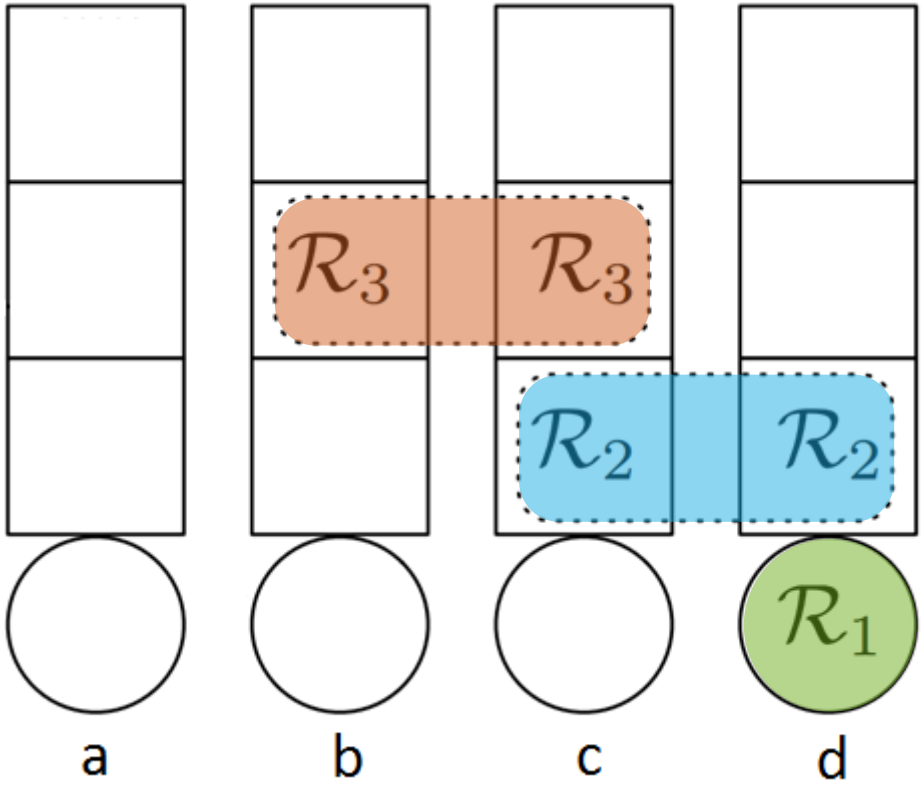
C-RNLP

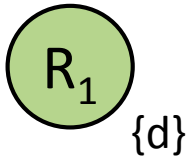
Results

Future Work

Wait-for graph model

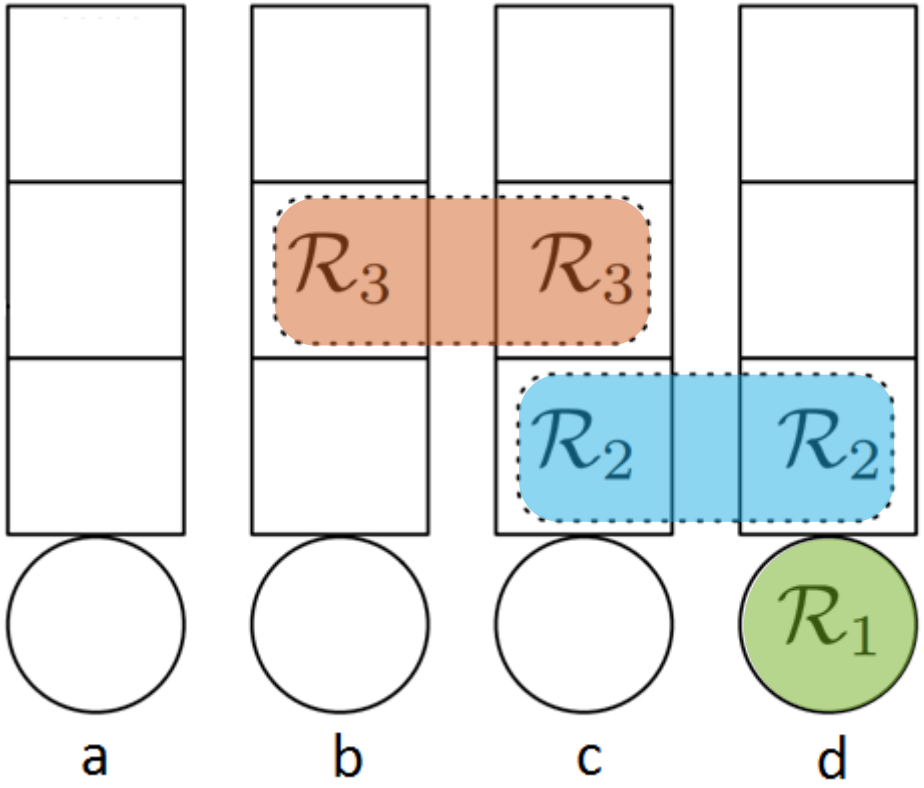
When is a request satisfied?

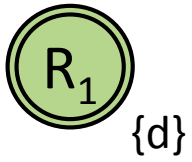




Wait-for graph model

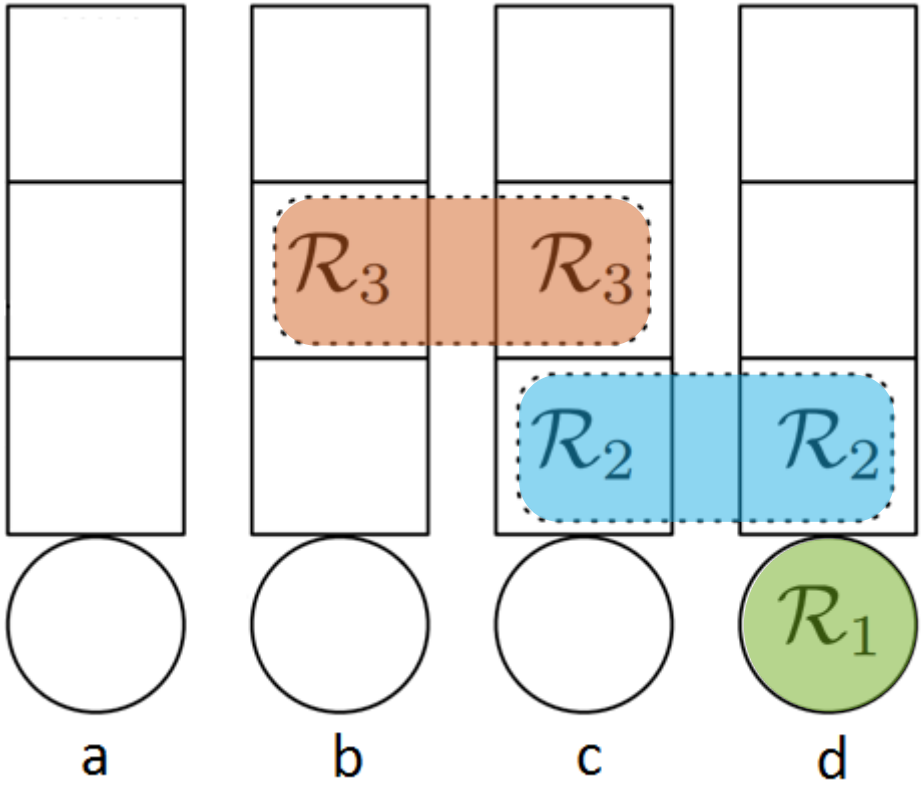
When is a request satisfied?

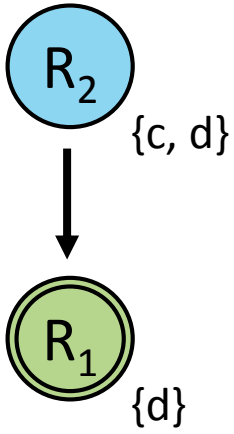




Wait-for graph model

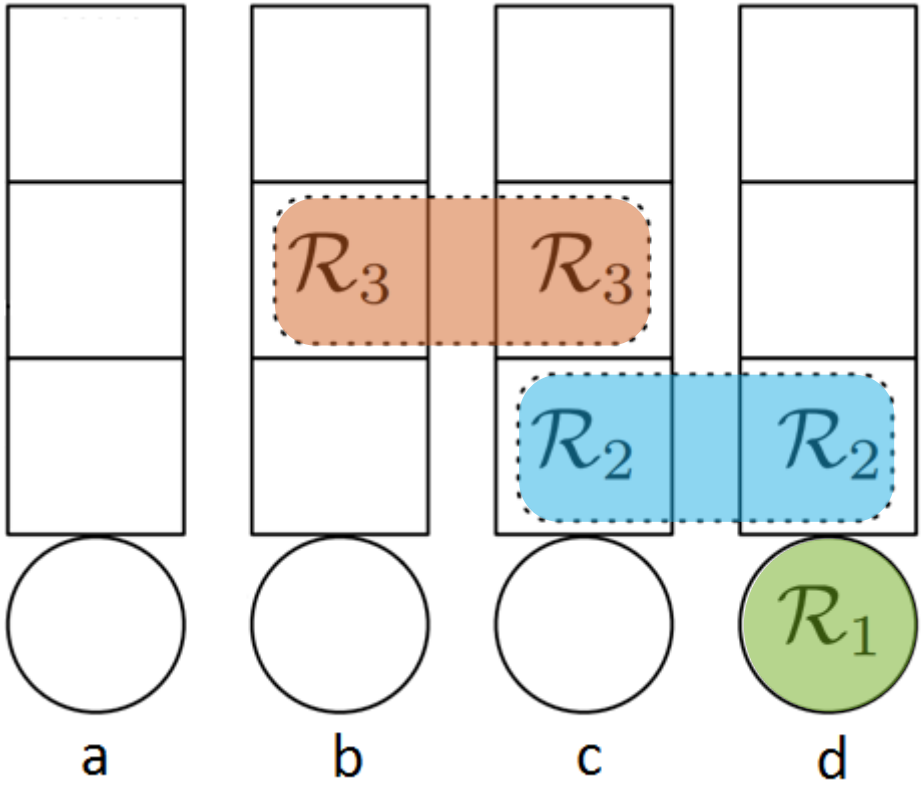
When is a request satisfied?

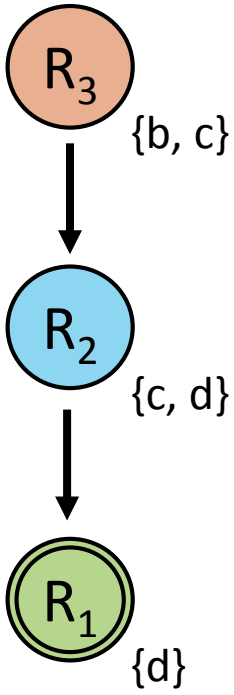




Wait-for graph model

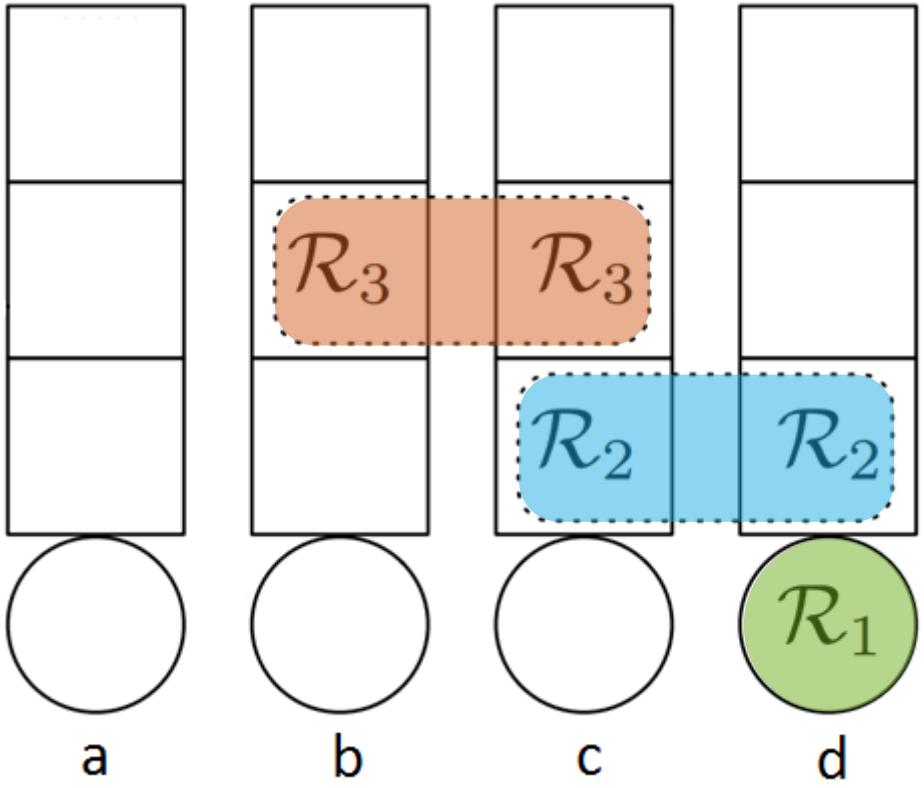
When is a request satisfied?

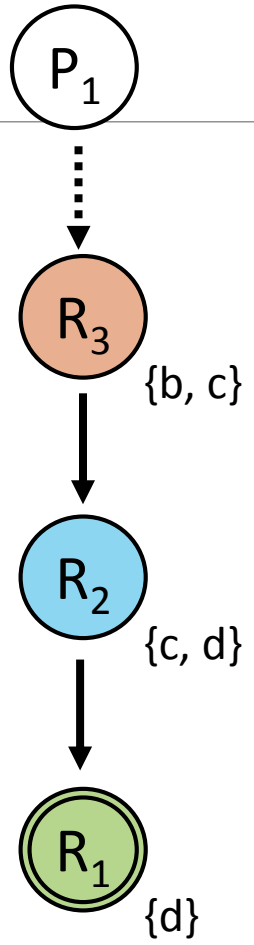




Wait-for graph model

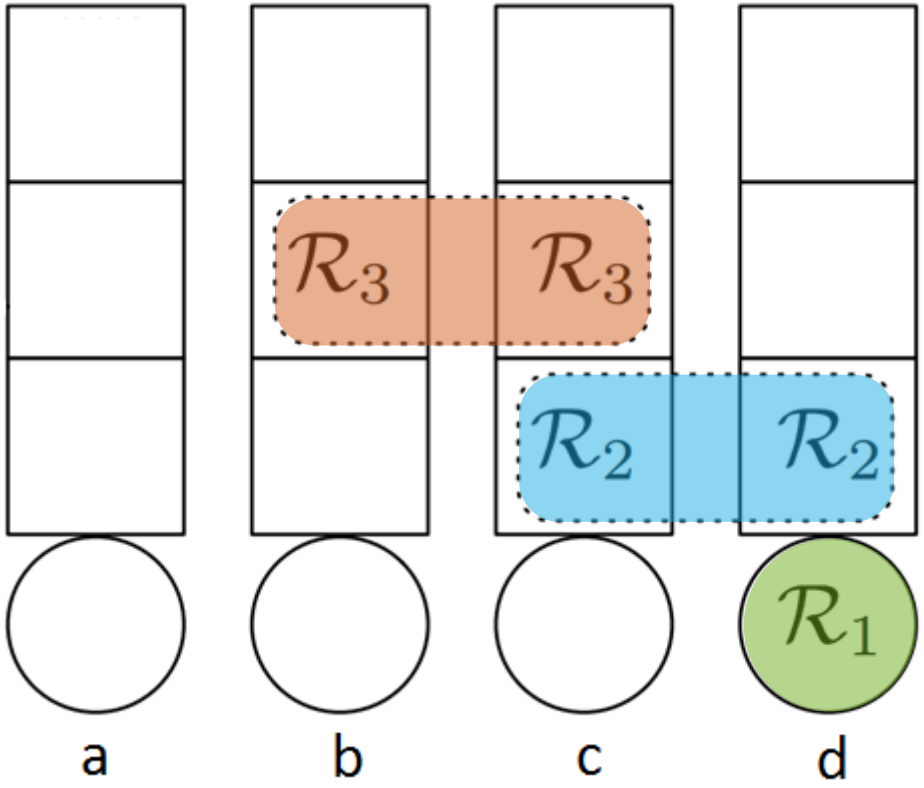
When is a request satisfied?

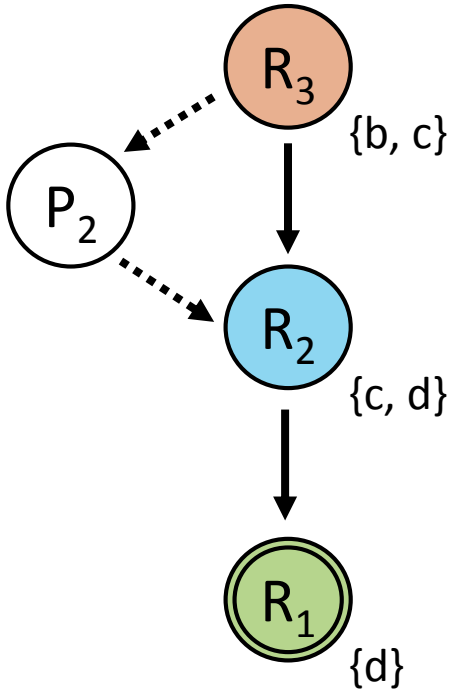




Wait-for graph model

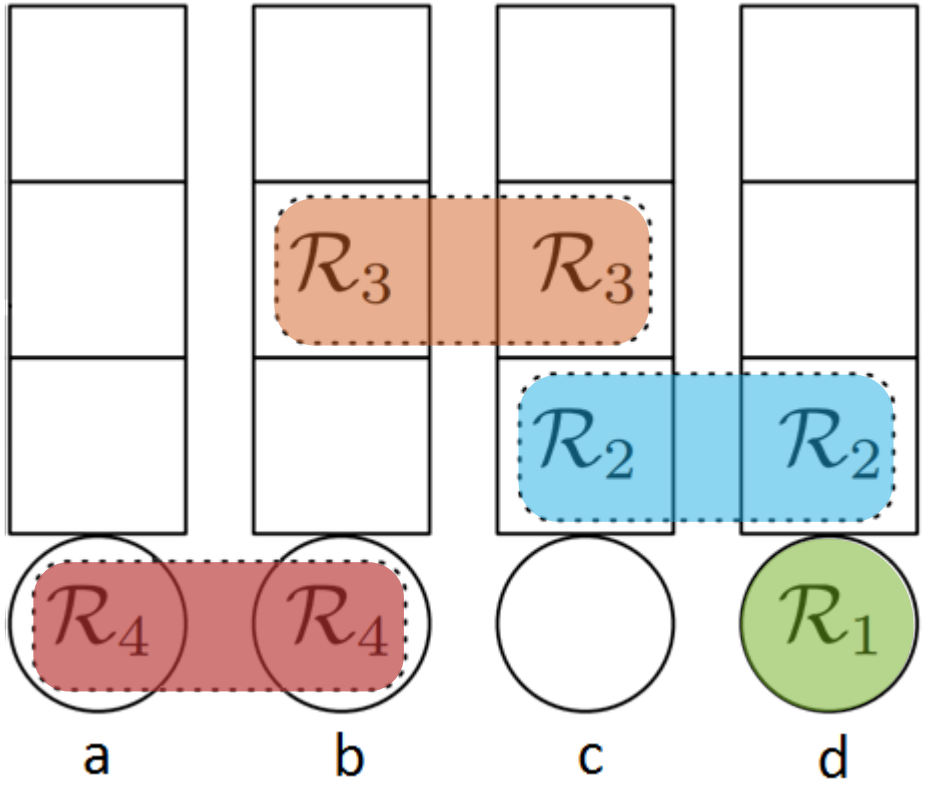
When is a request satisfied?

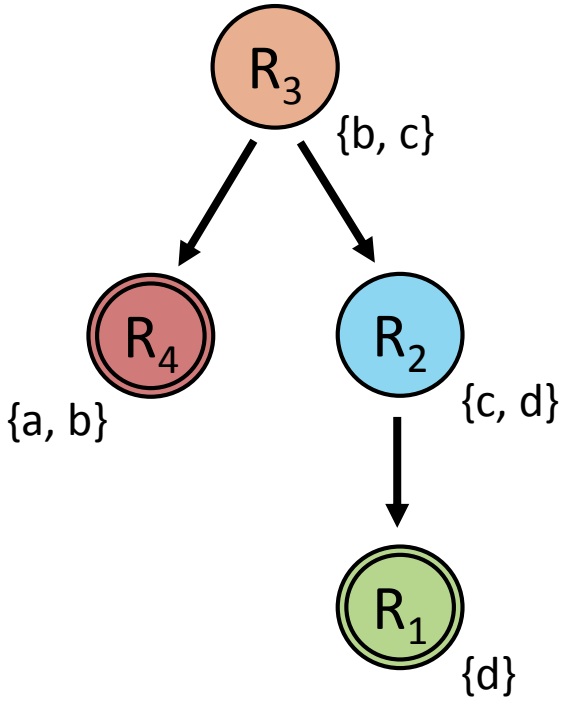




Wait-for graph model

When is a request satisfied?





Outline

Problem

Wait-For Graph Model

C-RNLP

Results

Future Work

Outline

Problem

Wait-For Graph Model

C-RNLP

Results

Future Work

Outline

Problem

Wait-For Graph Model

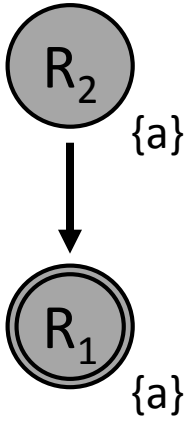
C-RNLP

- Safety
- Delay Preservation
- Capacity of a Position

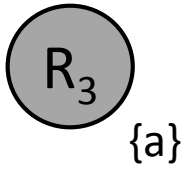
Results

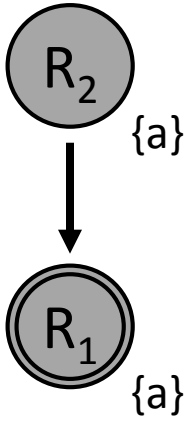
Future Work

Safety



Safety

Add: 

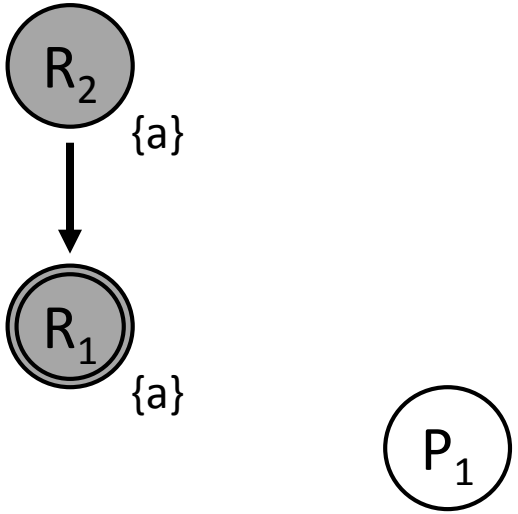


Safety

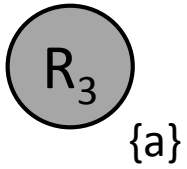
Safety

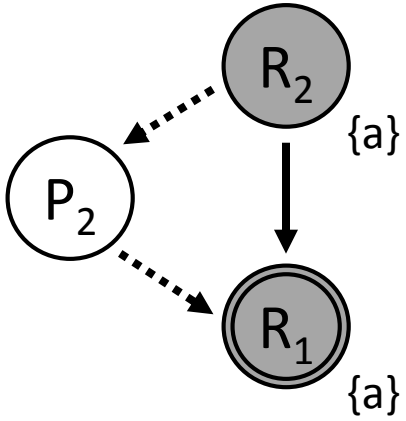
Add: R_3 {a}

Not safe!



Safety

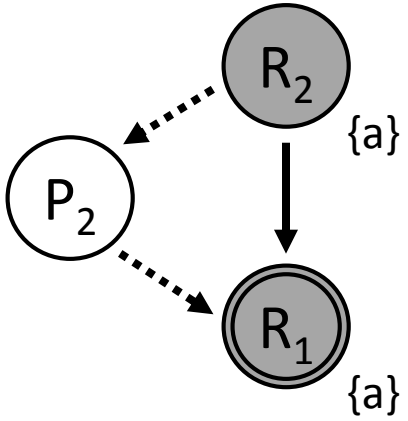
Add: 



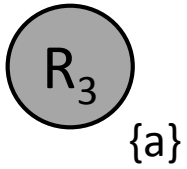
Safety

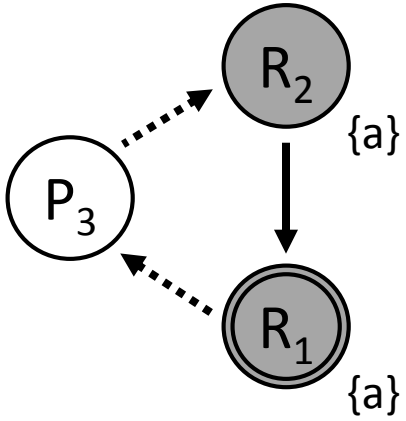
Add: R_3 {a}

Safe!

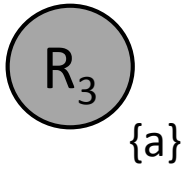


Safety

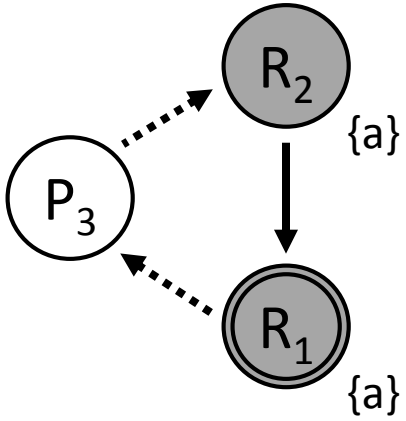
Add: 



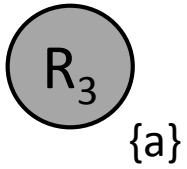
Safety

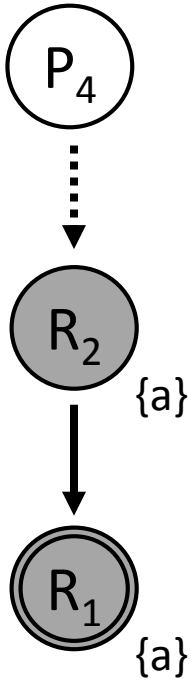
Add: 

Deadlock!
Not safe!



Safety

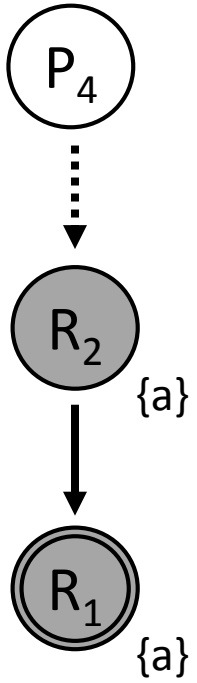
Add: 



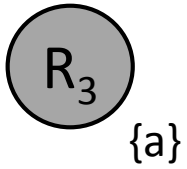
Safety

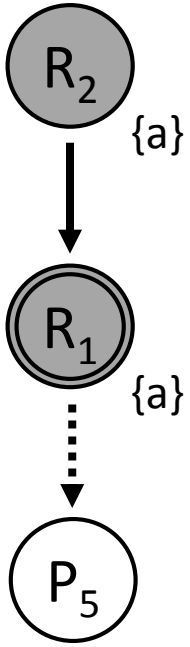
Add: R_3 {a}

Safe!



Safety

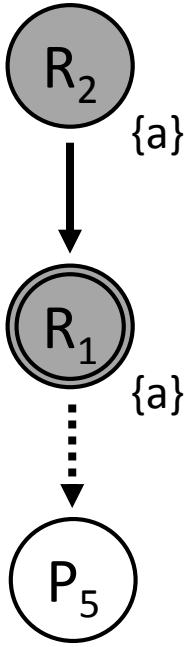
Add: 



Safety

Add: R_3 {a}

Not safe!



Outline

Problem

Wait-For Graph Model

C-RNLP

- Safety
- Delay Preservation
- Capacity of a Position

Results

Future Work

Safety: There must be a unique ordering per resource and a new request may not cut ahead of a satisfied request.

Outline

Problem

Wait-For Graph Model

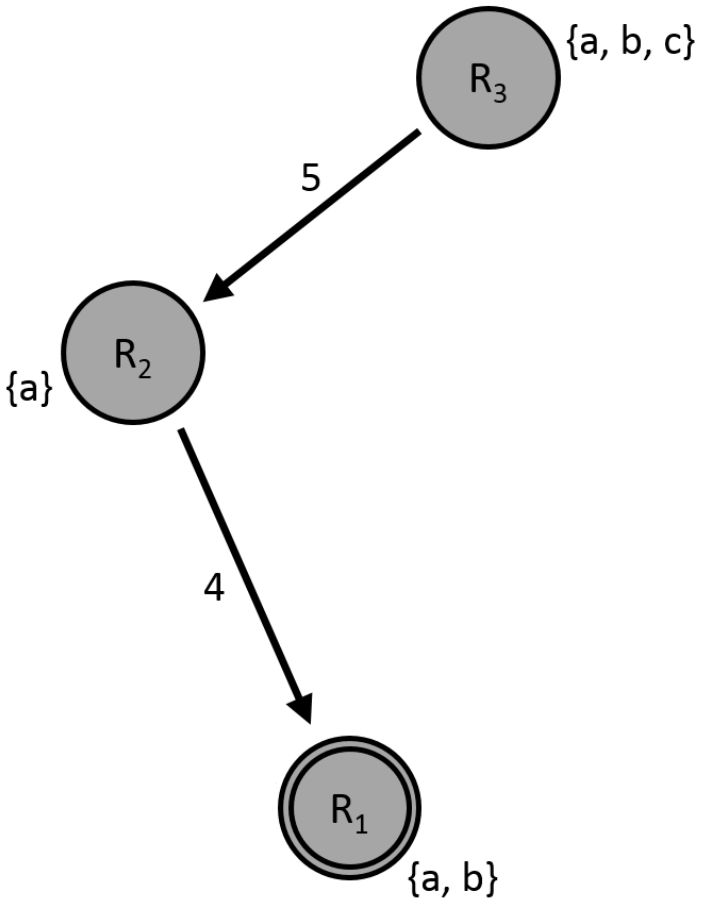
C-RNLP

- Safety
- Delay Preservation
- Capacity of a Position

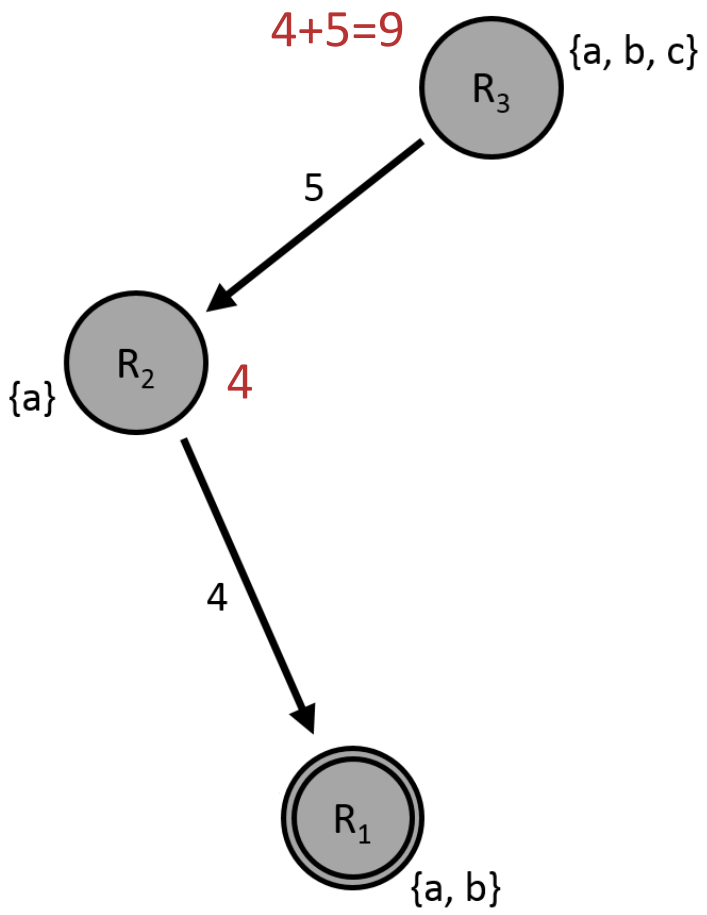
Results

Future Work

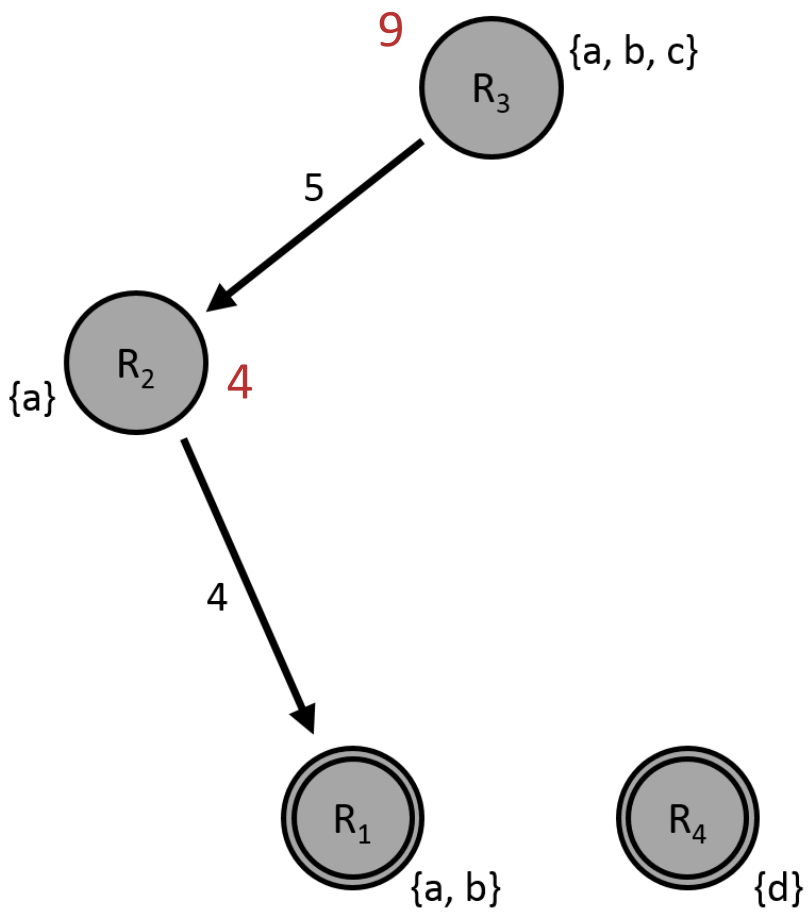
Delay preservation



Delay preservation

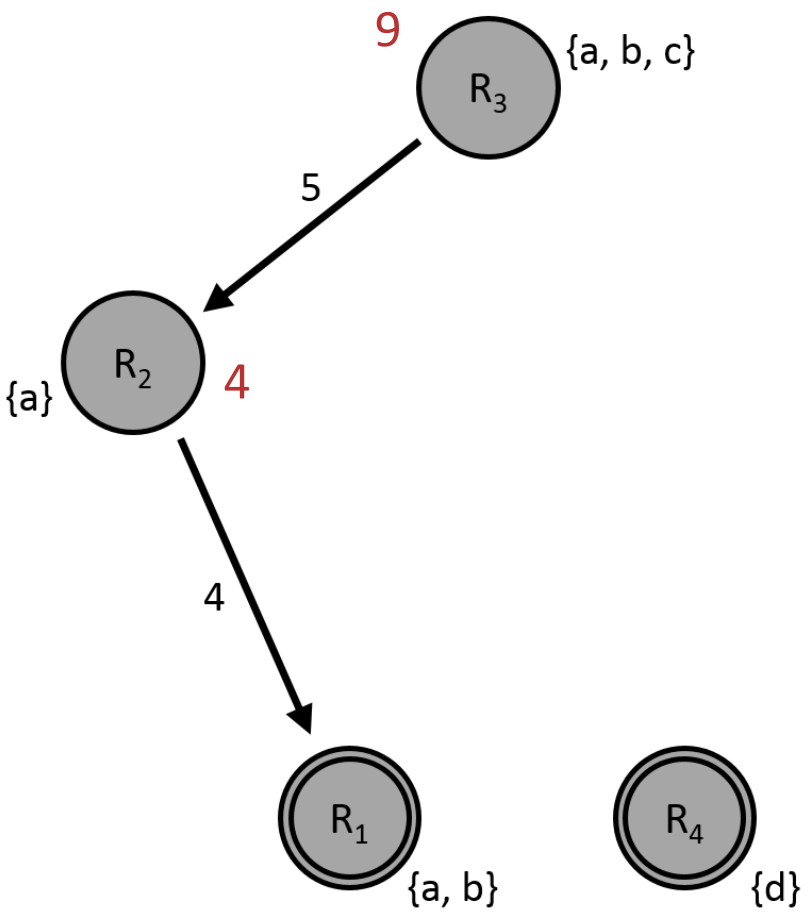


Delay preservation

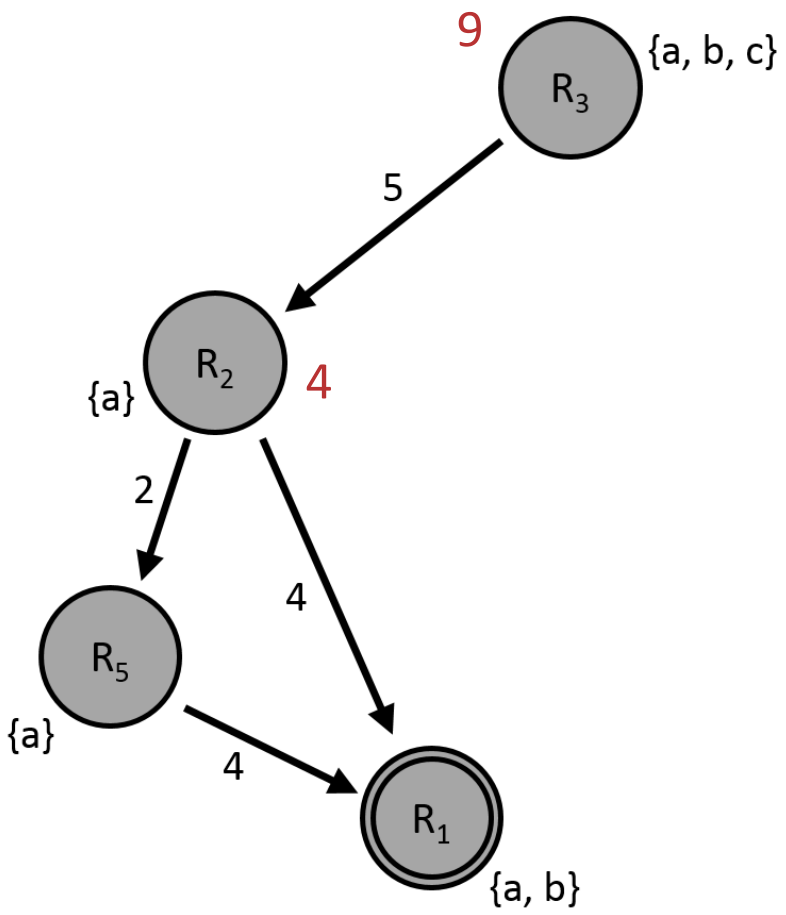


Delay preservation

Delay preserving!



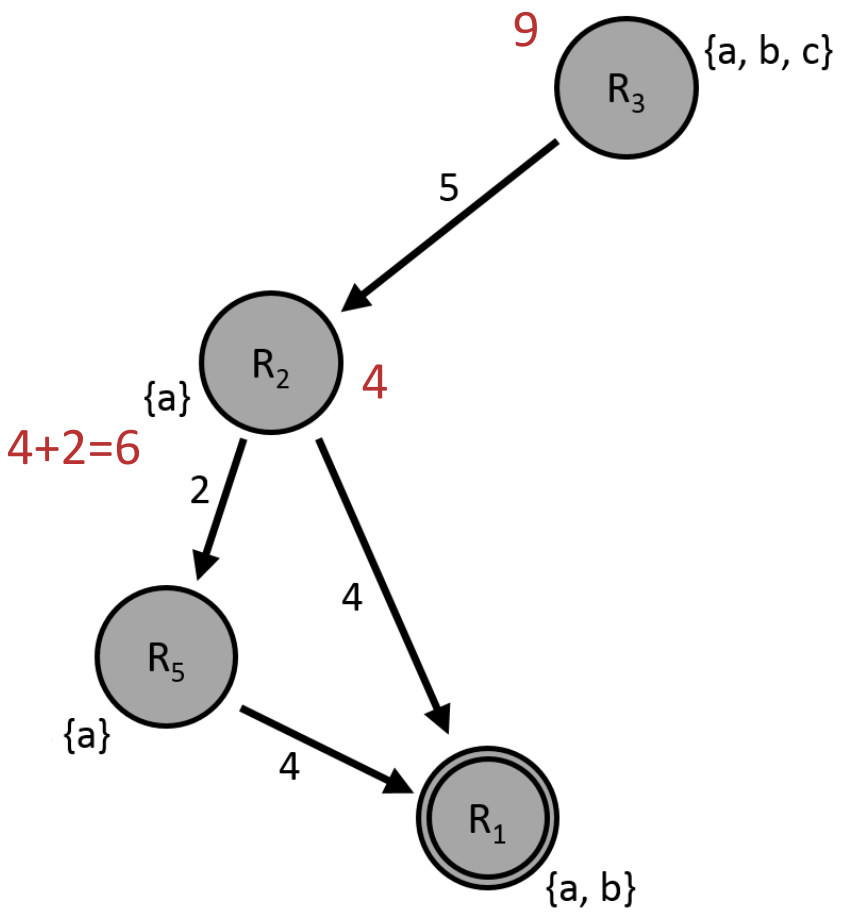
Delay preservation



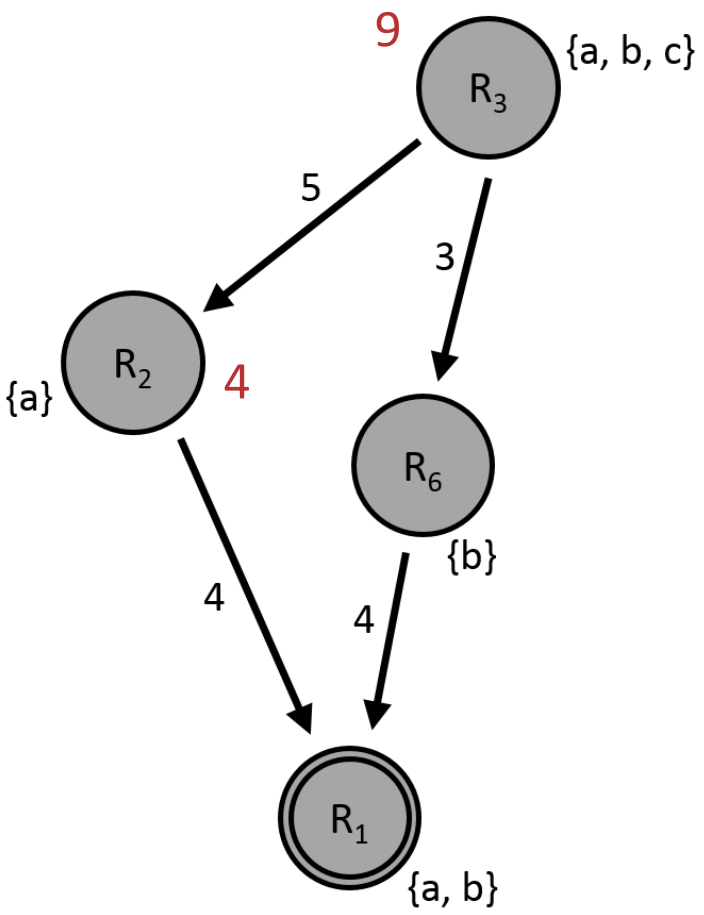
Delay preservation

Not delay preserving!

R_2 has increased worst-case blocking.

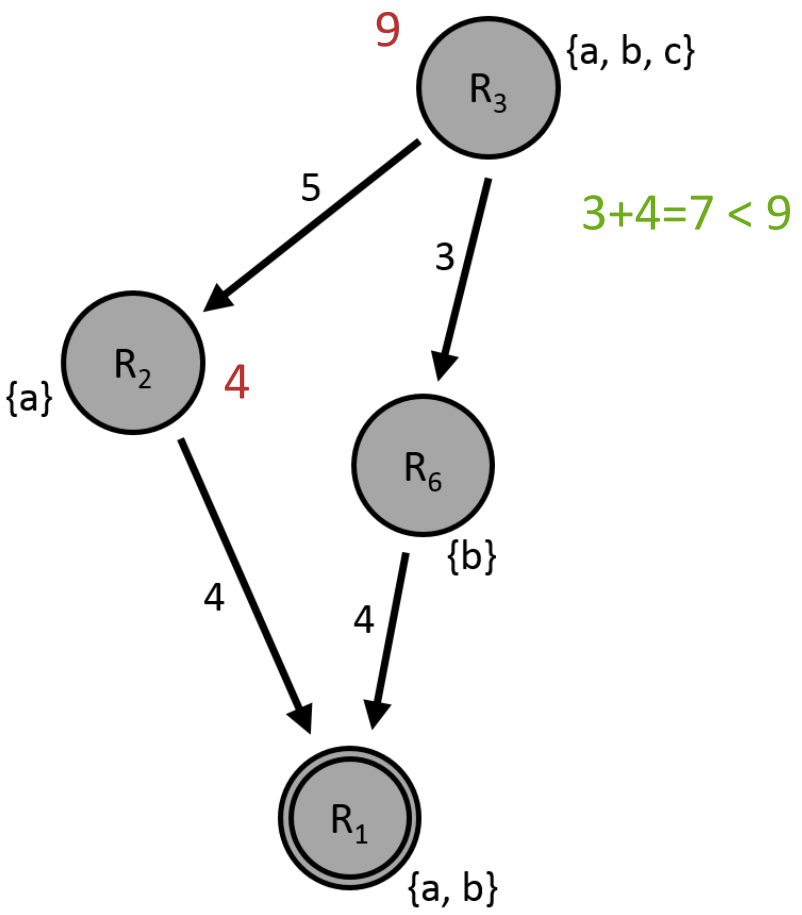


Delay preservation

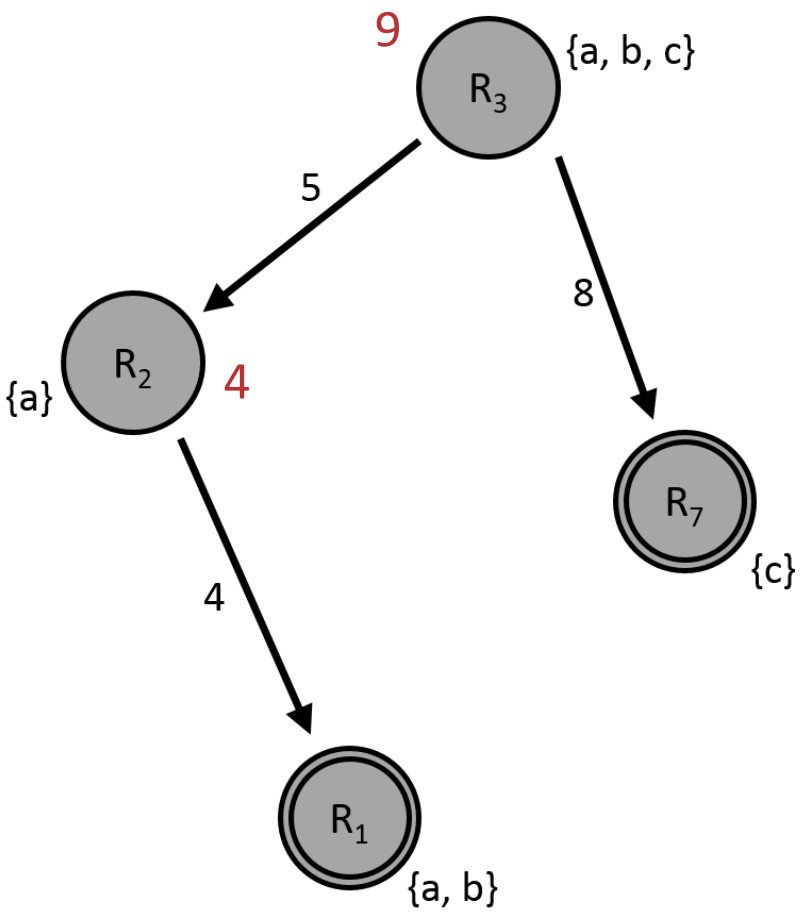


Delay preservation

Delay preserving!



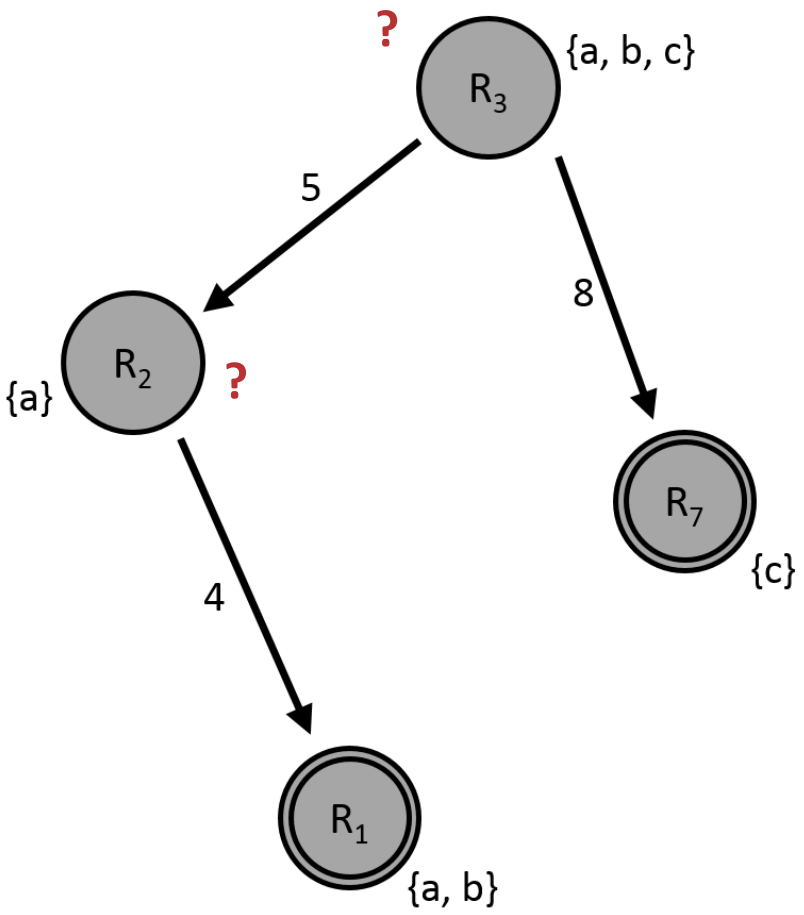
Delay preservation



Delay preservation

Delay preserving?

Worst-case blocking for R_3 depends on execution of R_1.



Outline

Problem

Wait-For Graph Model

C-RNLP

- Safety
- Delay Preservation
- Capacity of a Position

Results

Future Work

Delay Preservation: The insertion of a new request must not increase the worst-case blocking of any request.

Outline

Problem

Wait-For Graph Model

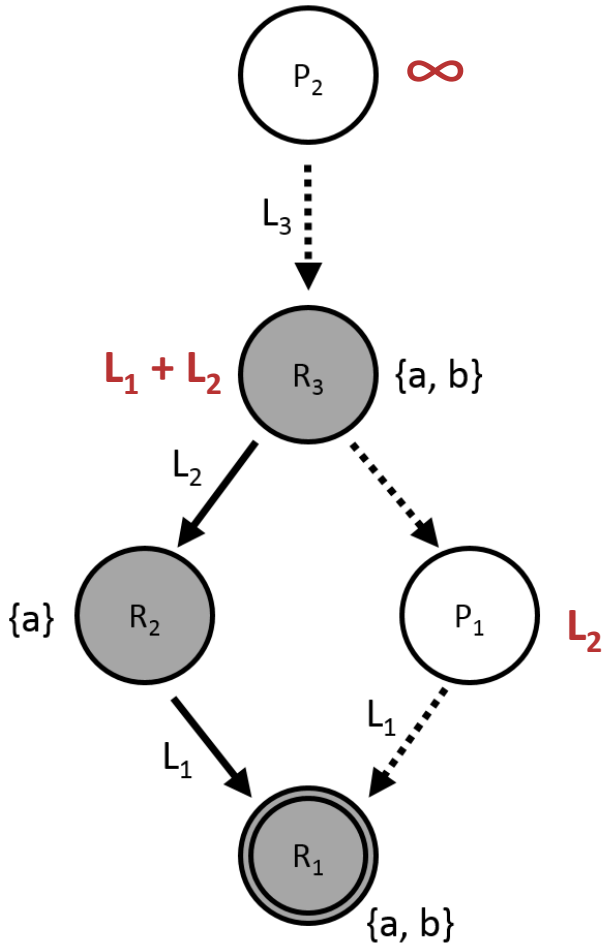
C-RNLP

- Safety
- Delay Preservation
- Capacity of a Position

Results

Future Work

Capacity of a Position



Outline

Problem

Wait-For Graph Model

C-RNLP

Results

Future Work

Outline

Problem

Wait-For Graph Model

C-RNLP

Results

Future Work

Outline

Problem

Wait-For Graph Model

C-RNLP

Results

- Theoretical
- Experimental

Future Work

Theorem

A request R_i is satisfied within $O(\min(m, c_i))$ time units.

Outline

Problem

Wait-For Graph Model

C-RNLP

Results

- Theoretical
- Experimental

Future Work

Outline

Problem

Wait-For Graph Model

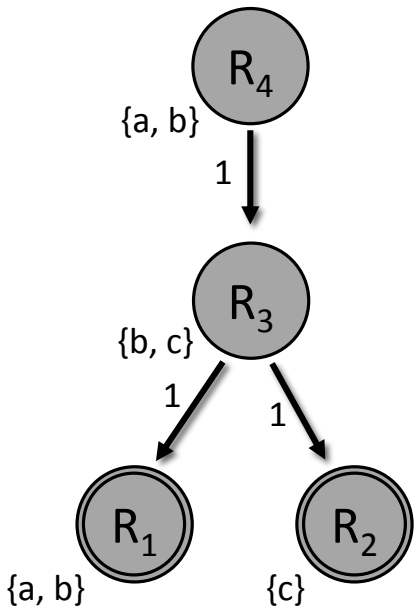
C-RNLP

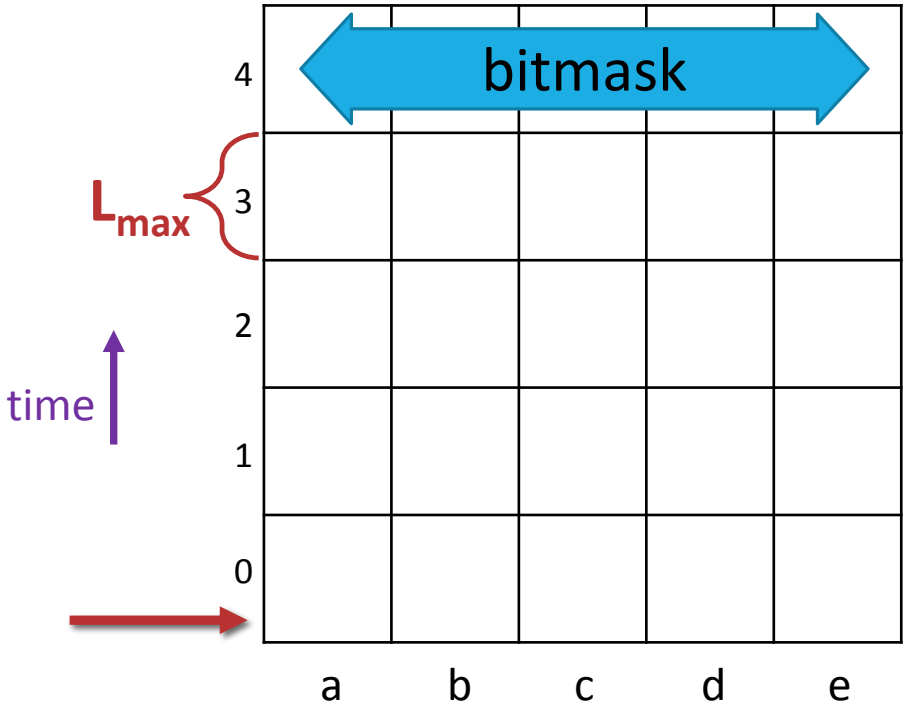
Results

- Theoretical
- Experimental

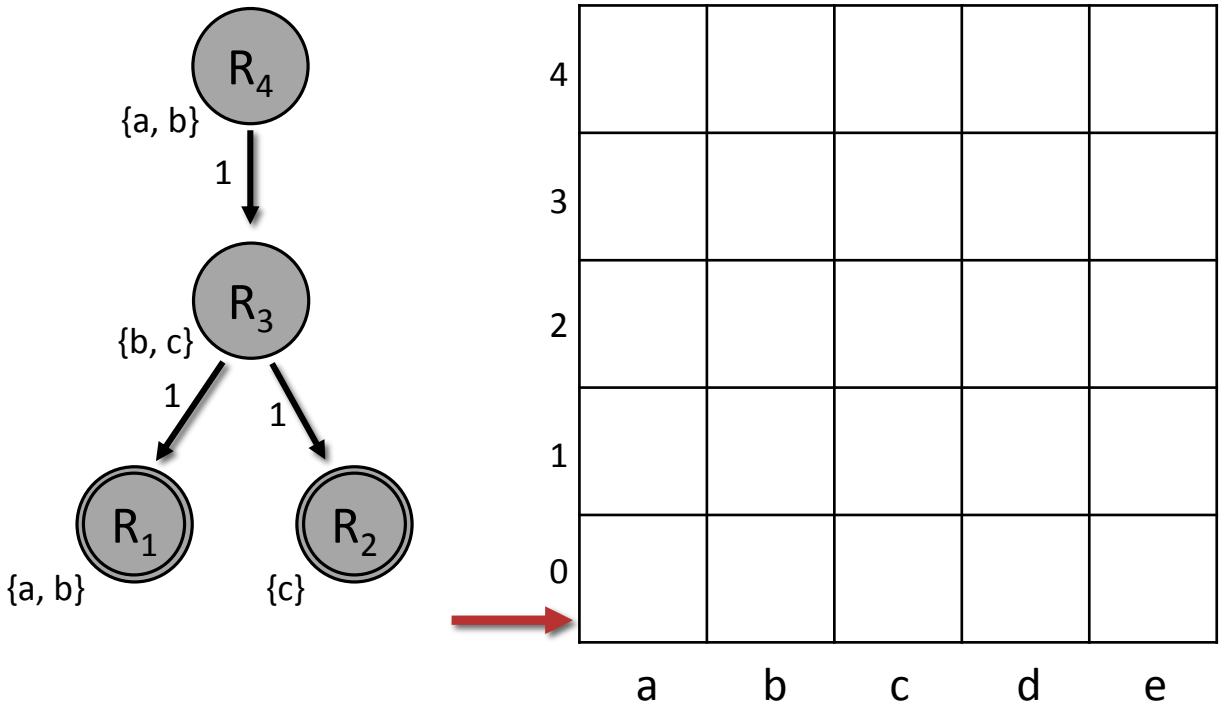
Future Work

Implementation of C-RNLP

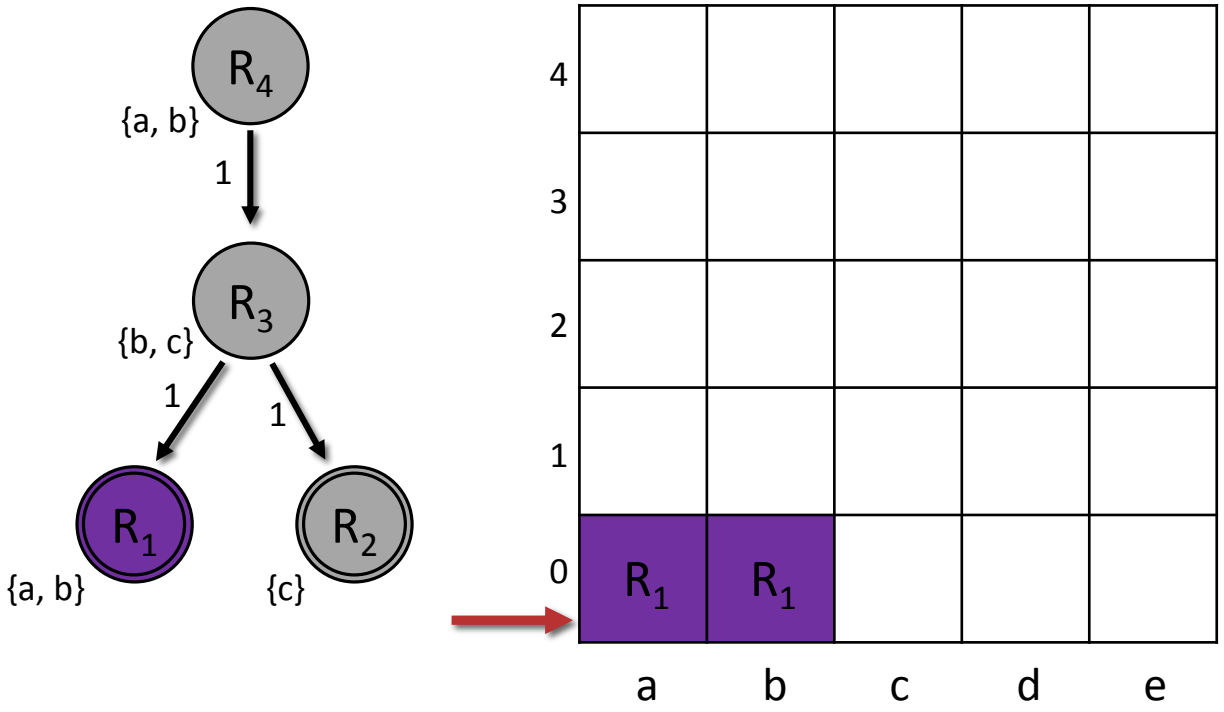




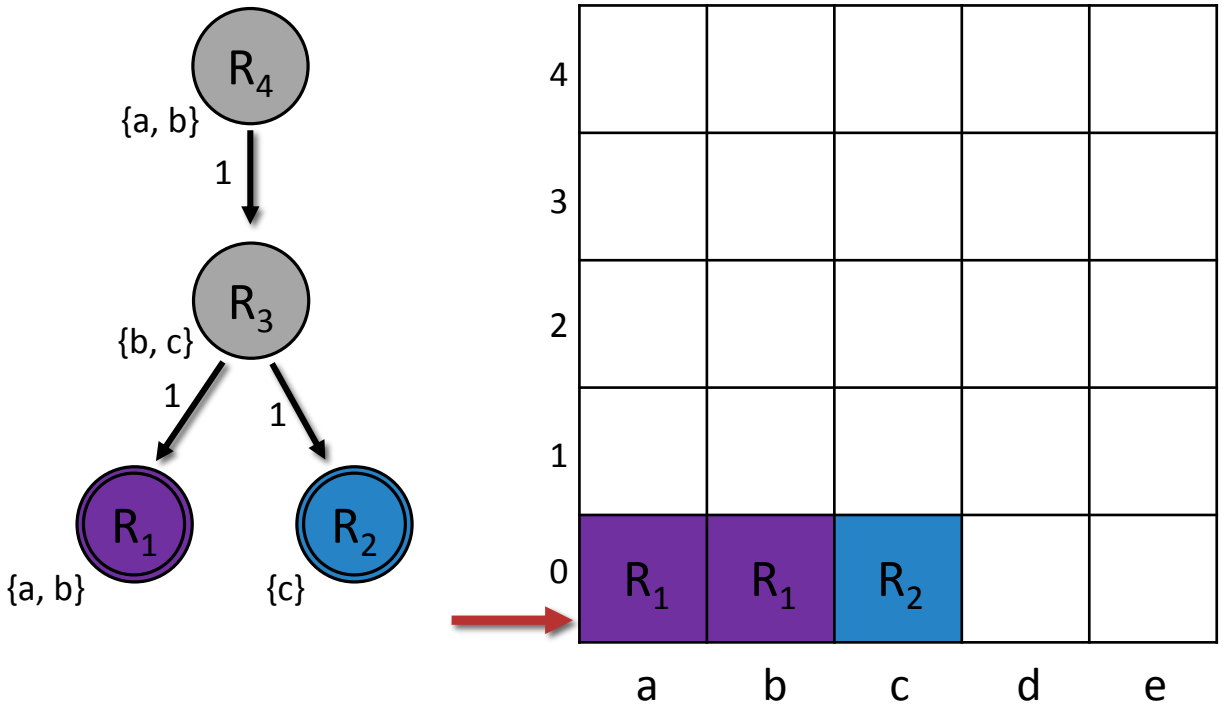
Implementation of C-RNLP



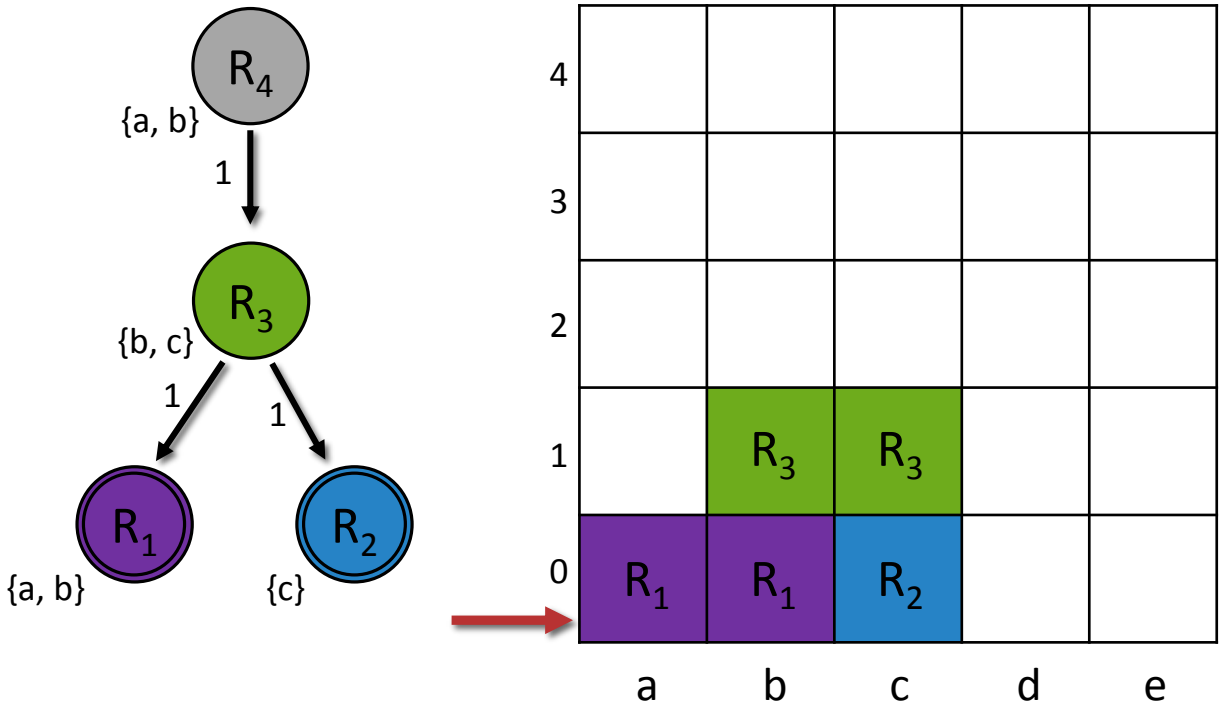
Implementation of C-RNLP



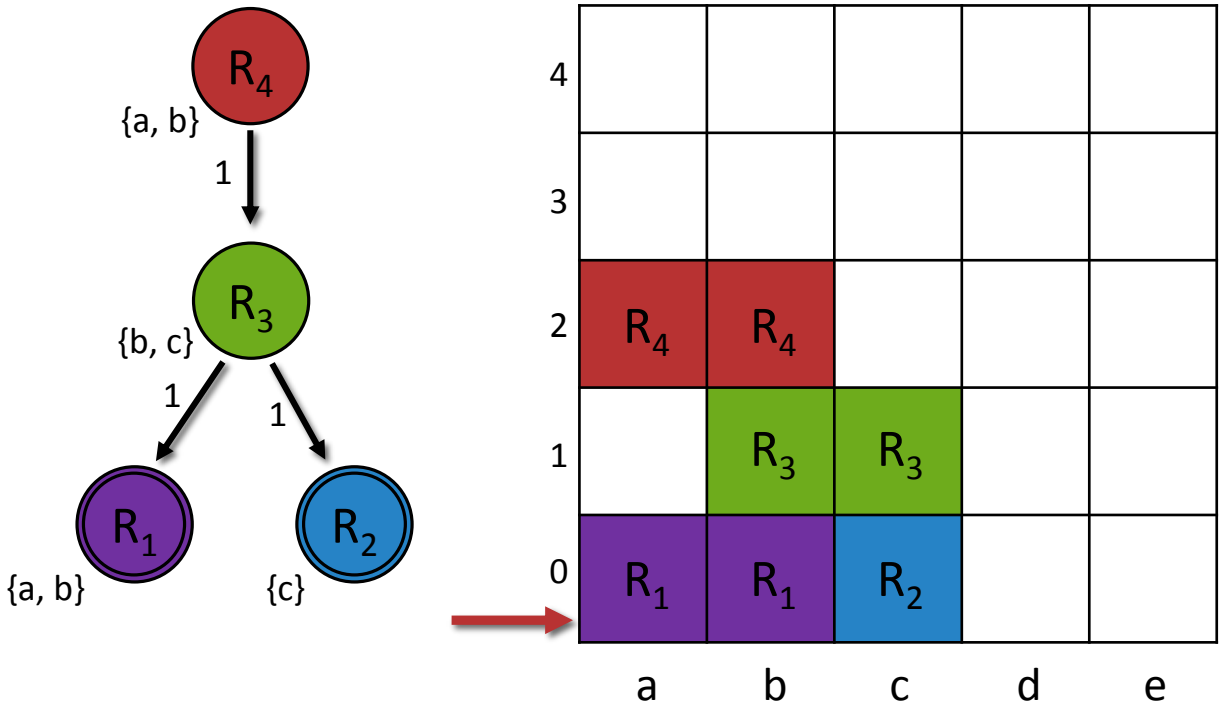
Implementation of C-RNLP



Implementation of C-RNLP



Implementation of C-RNLP



Experimental Evaluation

Dual-socket, 18-cores-per-socket Intel Xeon E5-2699 platform

Comparison with:

- RNLP
- Mellor-Crummey and Scott's queue lock (MCS)

Experimental Evaluation

Tasks {2, 4, …, 36}

Resources {1, 2, 4, 6, …, 64}

Requested resources {1, 2, 4, 8, 12, …,48}

Generated task sets to request resources as often as possible

Analyzing: overhead blocking and protocol blocking

Experimental Evaluation

overhead blocking

protocol blocking

C-RNLP Lock

```
procedure C-RNLP LOCK(requested)
    lock(Sublock)
    if Pending_requests > 0 then
        start ← (Head + 1) mod SIZE
        while (Table[start] & requested) ≠ 0 do
            start ← (start + 1) mod SIZE
        end while
    else
        start ← Head
    end if
    next ← (start + 1) mod SIZE
    Blocked[next] ← Blocked[next] + 1
    Pending_requests ← Pending_requests + 1
    Table[start] ← Table[start] | requested
    unlock(Sublock)
    while Enabled[start] ≠ 1 do
        /* null */
    end while
end procedure
```

Experimental Evaluation

overhead blocking

protocol blocking

C-RNLP Lock

```
procedure C-RNLP LOCK(requested)
    lock(Sublock)
    if Pending_requests > 0 then
        start ← (Head + 1) mod SIZE
        while (Table[start] & requested) ≠ 0 do
            start ← (start + 1) mod SIZE
        end while
    else
        start ← Head
    end if
    next ← (start + 1) mod SIZE
    Blocked[next] ← Blocked[next] + 1
    Pending_requests ← Pending_requests + 1
    Table[start] ← Table[start] | requested
    unlock(Sublock)
    while Enabled[start] ≠ 1 do
        /* null */
    end while
end procedure
```

Experimental Evaluation

overhead blocking

protocol blocking

C-RNLP Lock

```
procedure C-RNLP LOCK(requested)
    lock(Sublock)
    if Pending_requests > 0 then
        start ← (Head + 1) mod SIZE
        while (Table[start] & requested) ≠ 0 do
            start ← (start + 1) mod SIZE
        end while
    else
        start ← Head
    end if
    next ← (start + 1) mod SIZE
    Blocked[next] ← Blocked[next] + 1
    Pending_requests ← Pending_requests + 1
    Table[start] ← Table[start] | requested
    unlock(Sublock)
    while Enabled[start] ≠ 1 do
        /* null */
    end while
end procedure
```

Experimental Evaluation

overhead blocking

protocol blocking

C-RNLP Lock

```
procedure C-RNLP LOCK(requested)
    lock(Sublock)
    if Pending_requests > 0 then
        start ← (Head + 1) mod SIZE
        while (Table[start] & requested) ≠ 0 do
            start ← (start + 1) mod SIZE
        end while
    else
        start ← Head
    end if
    next ← (start + 1) mod SIZE
    Blocked[next] ← Blocked[next] + 1
    Pending_requests ← Pending_requests + 1
    Table[start] ← Table[start] | requested
    unlock(Sublock)
    while Enabled[start] ≠ 1 do
        /* null */
    end while
end procedure
```

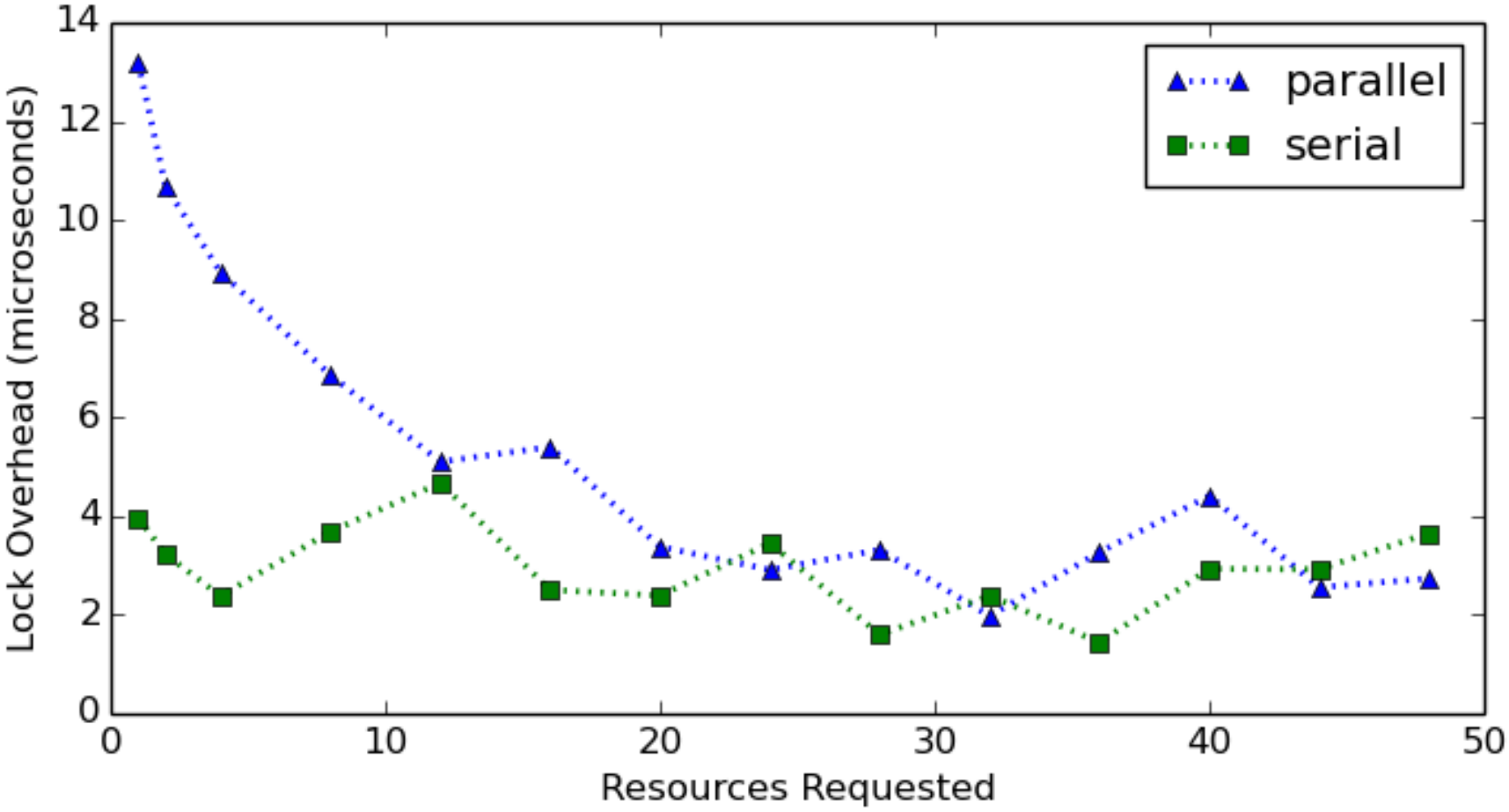
Experimental Evaluation

overhead blocking = lock overhead

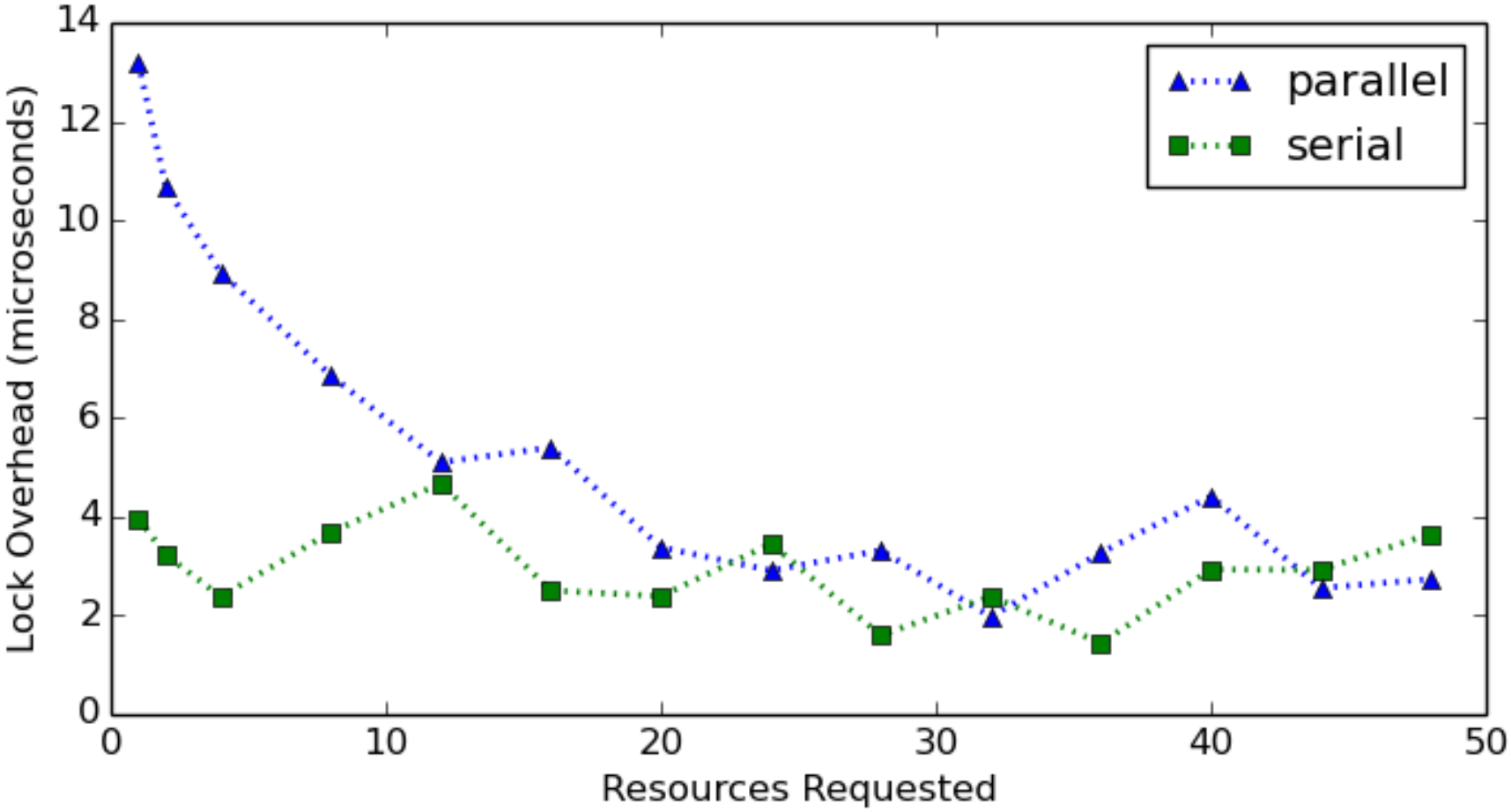
protocol blocking

C-RNLP Lock

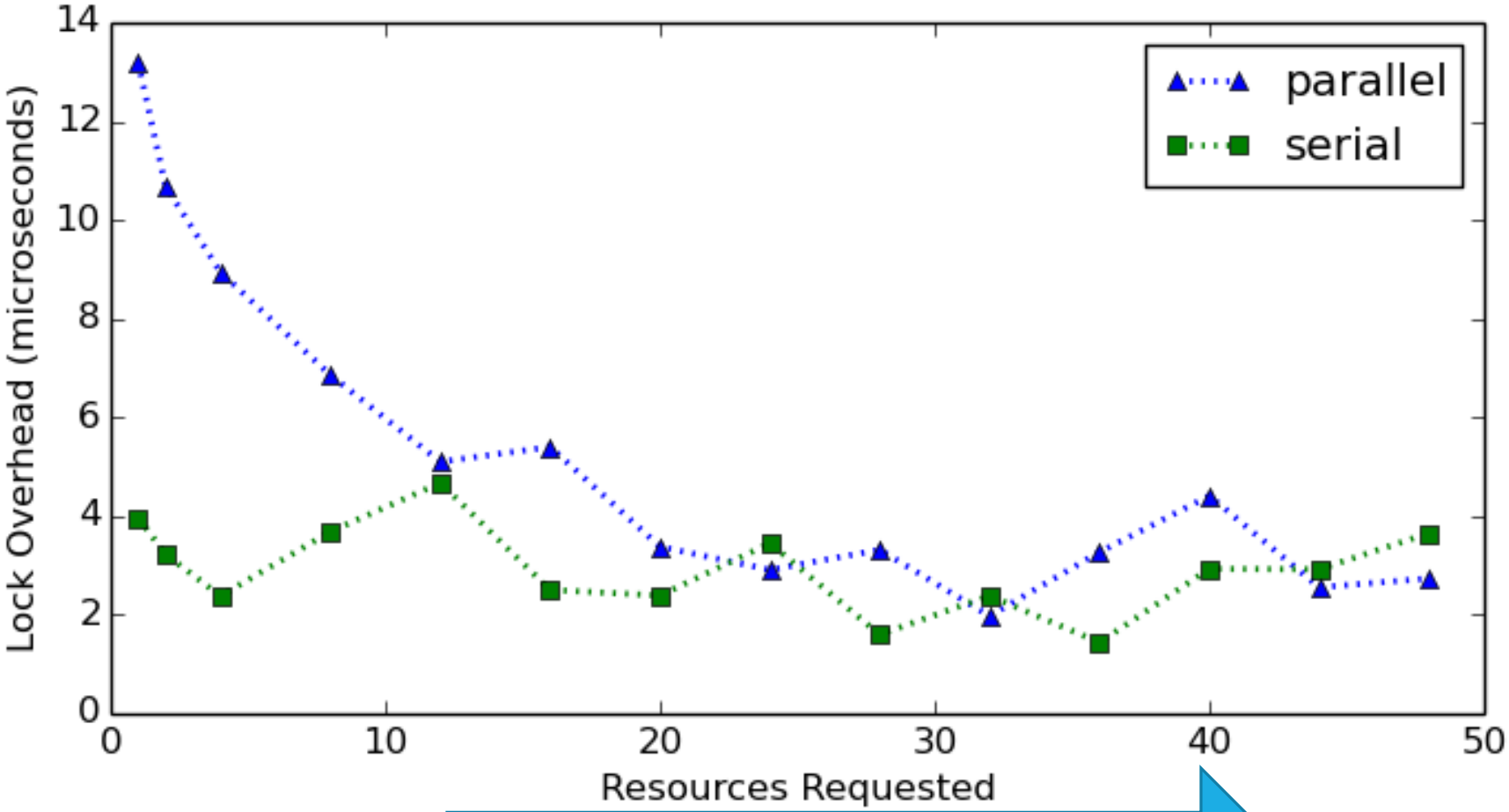
```
procedure C-RNLP LOCK(requested)
    lock(Sublock)
    if Pending_requests > 0 then
        start ← (Head + 1) mod SIZE
        while (Table[start] & requested) ≠ 0 do
            start ← (start + 1) mod SIZE
        end while
    else
        start ← Head
    end if
    next ← (start + 1) mod SIZE
    Blocked[next] ← Blocked[next] + 1
    Pending_requests ← Pending_requests + 1
    Table[start] ← Table[start] | requested
    unlock(Sublock)
    while Enabled[start] ≠ 1 do
        /* null */
    end while
end procedure
```


parallel
serial
Lock Overhead (microseconds)
Resources Requested
0
2
4
6
8
10
12
14
0
10
20
30
40
50

Tasks: 36
Available resources: 64

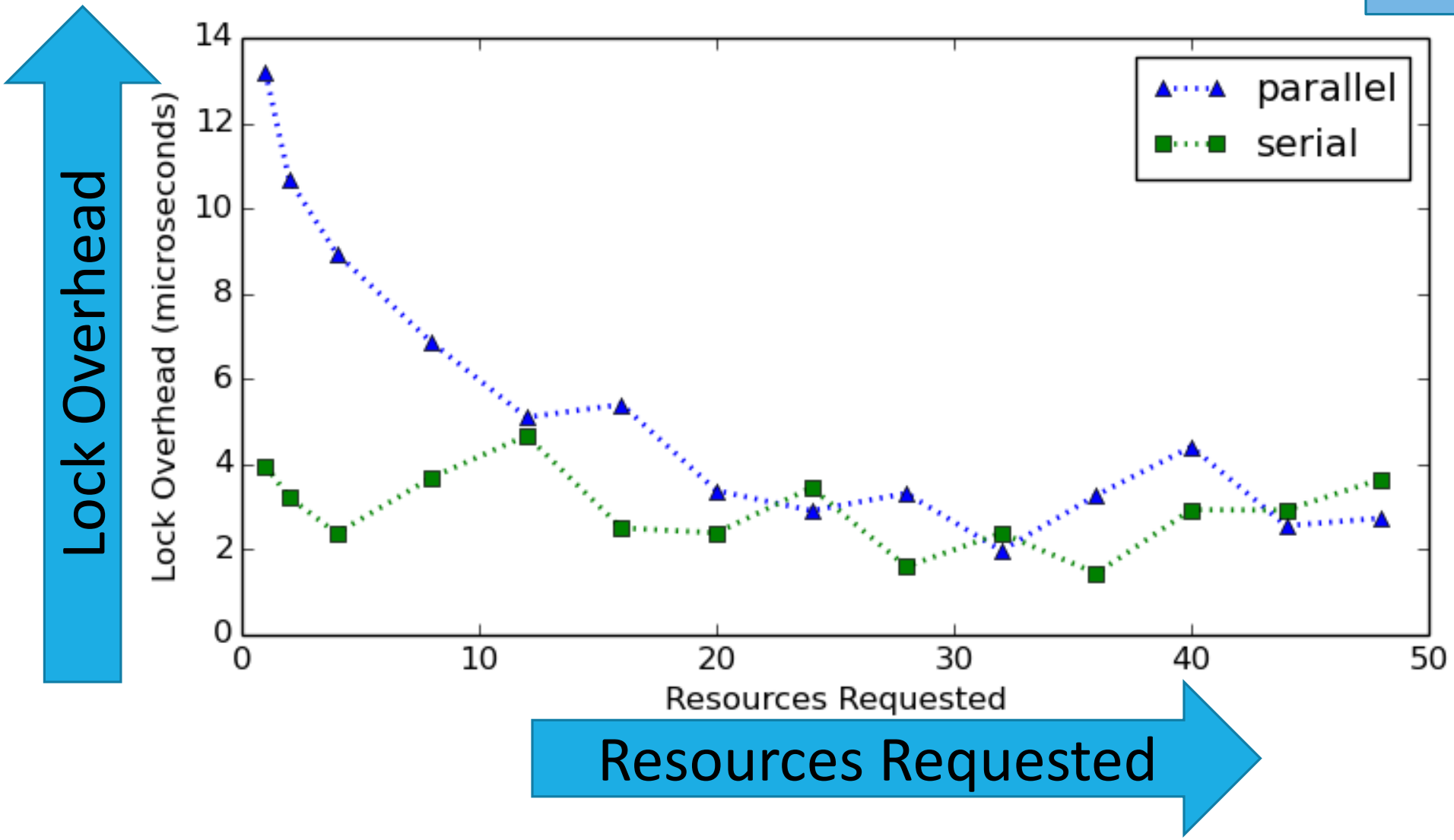


Tasks: 36
Available resources: 64

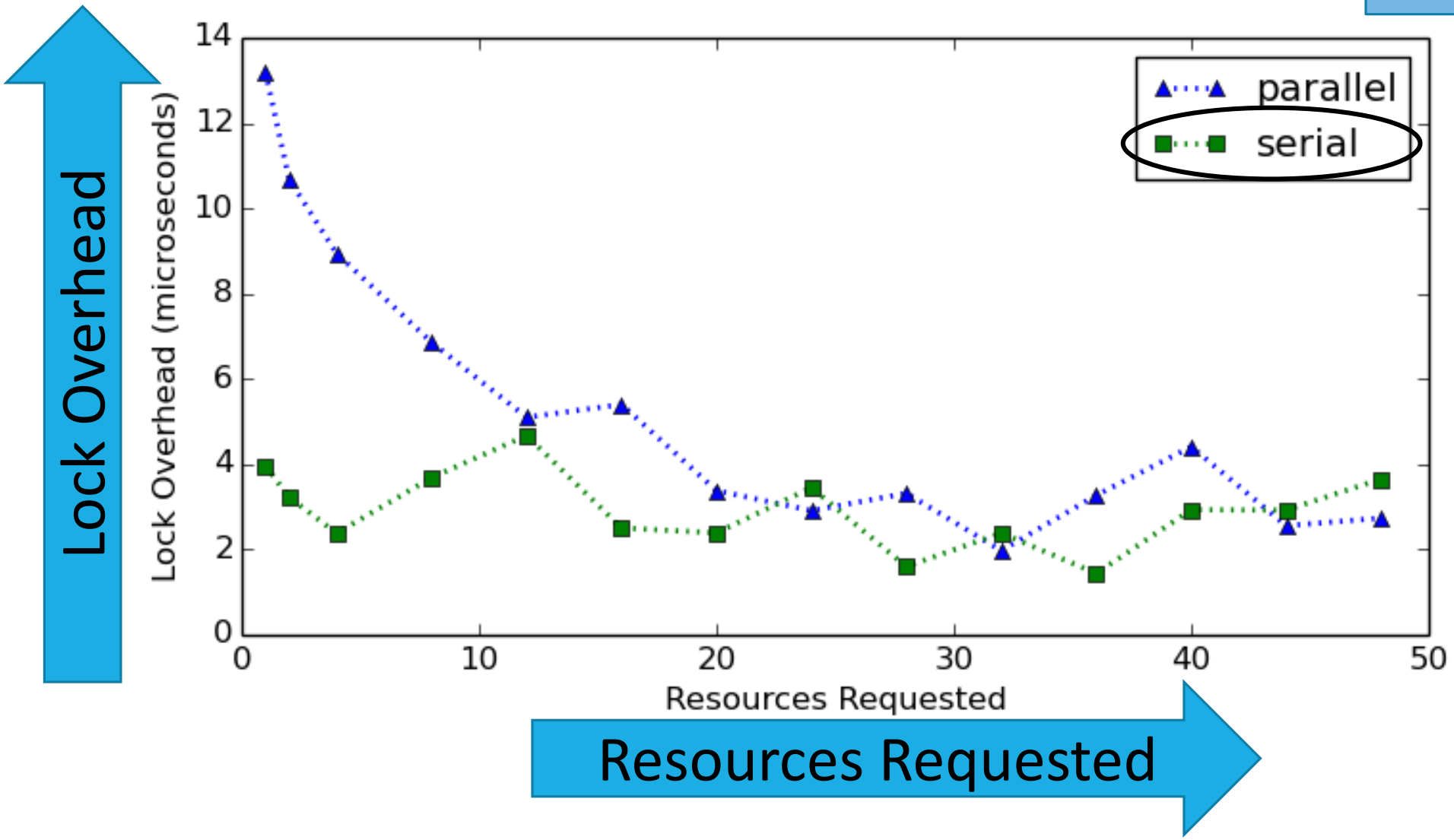


Resources Requested

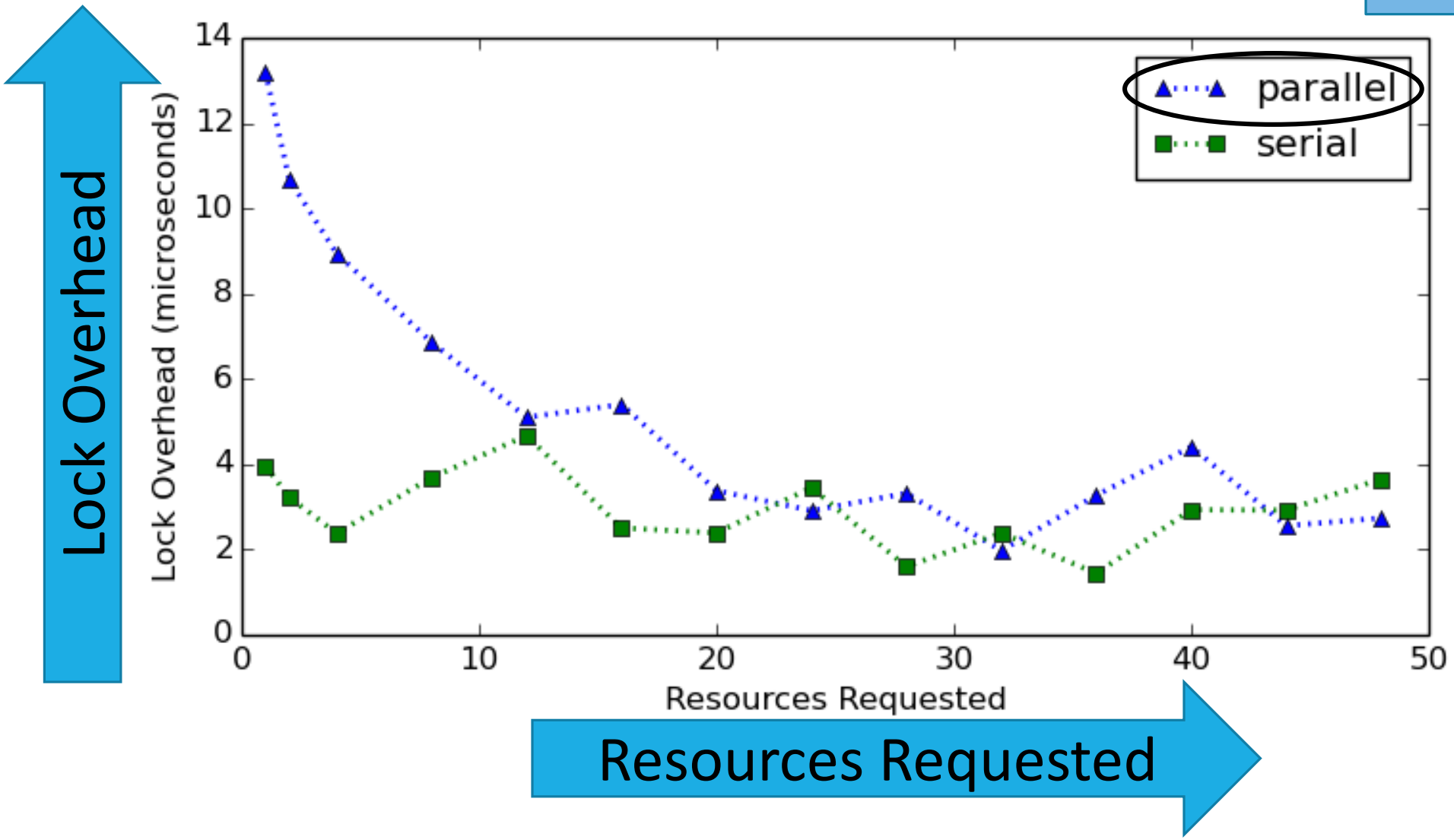
Tasks: 36
Available resources: 64



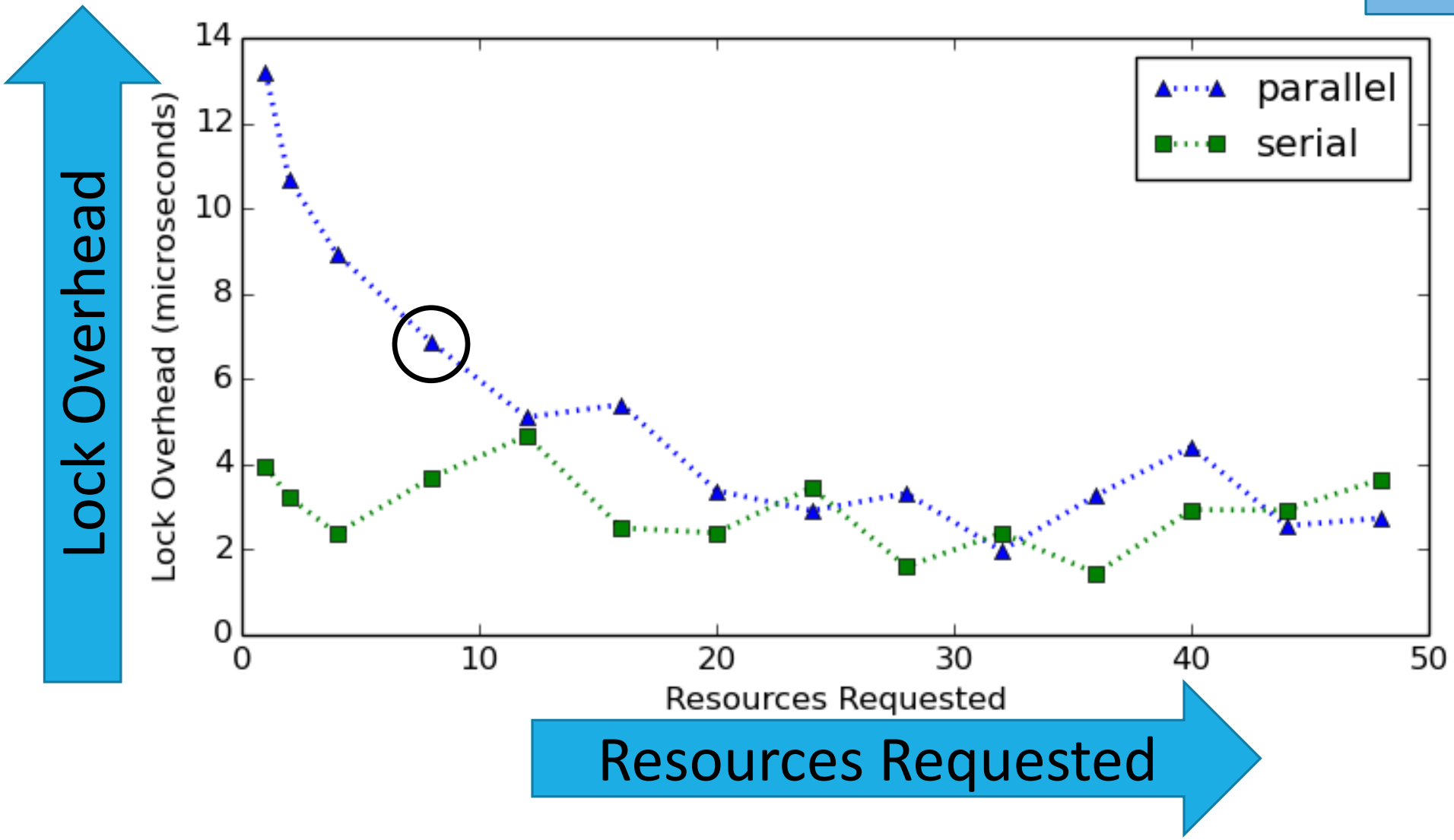
Tasks: 36
Available resources: 64



Tasks: 36
Available resources: 64

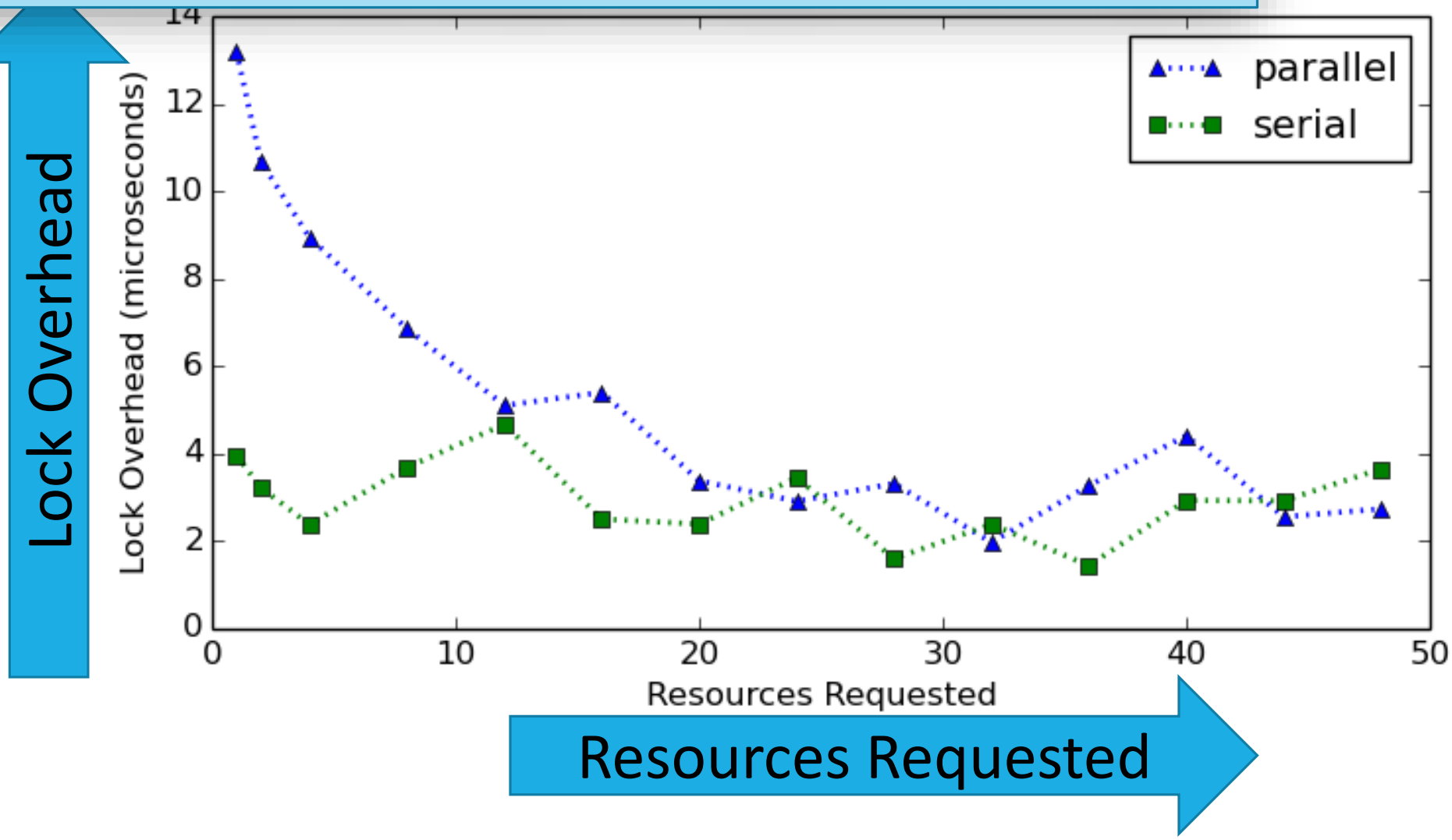


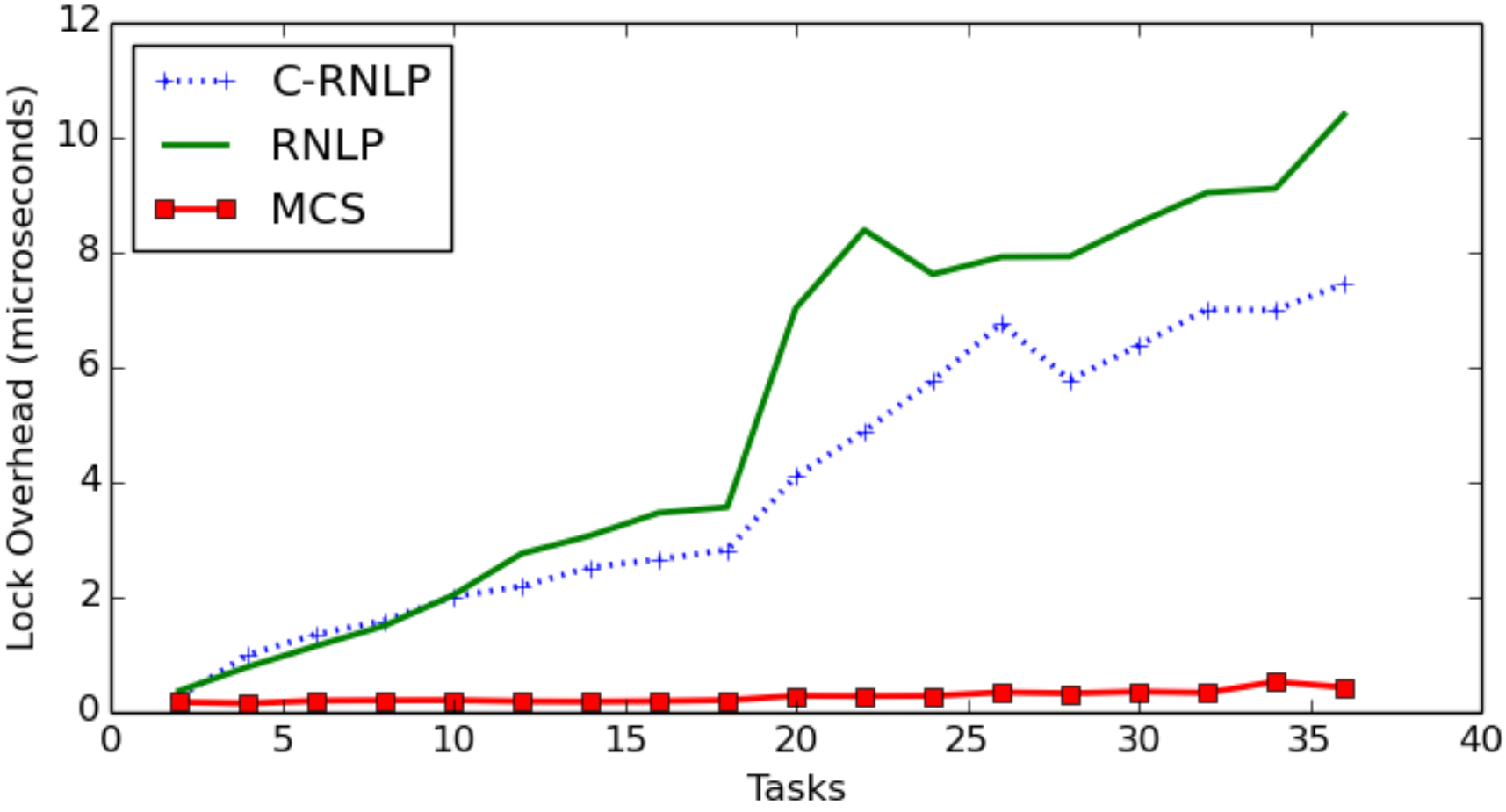
Tasks: 36
Available resources: 64



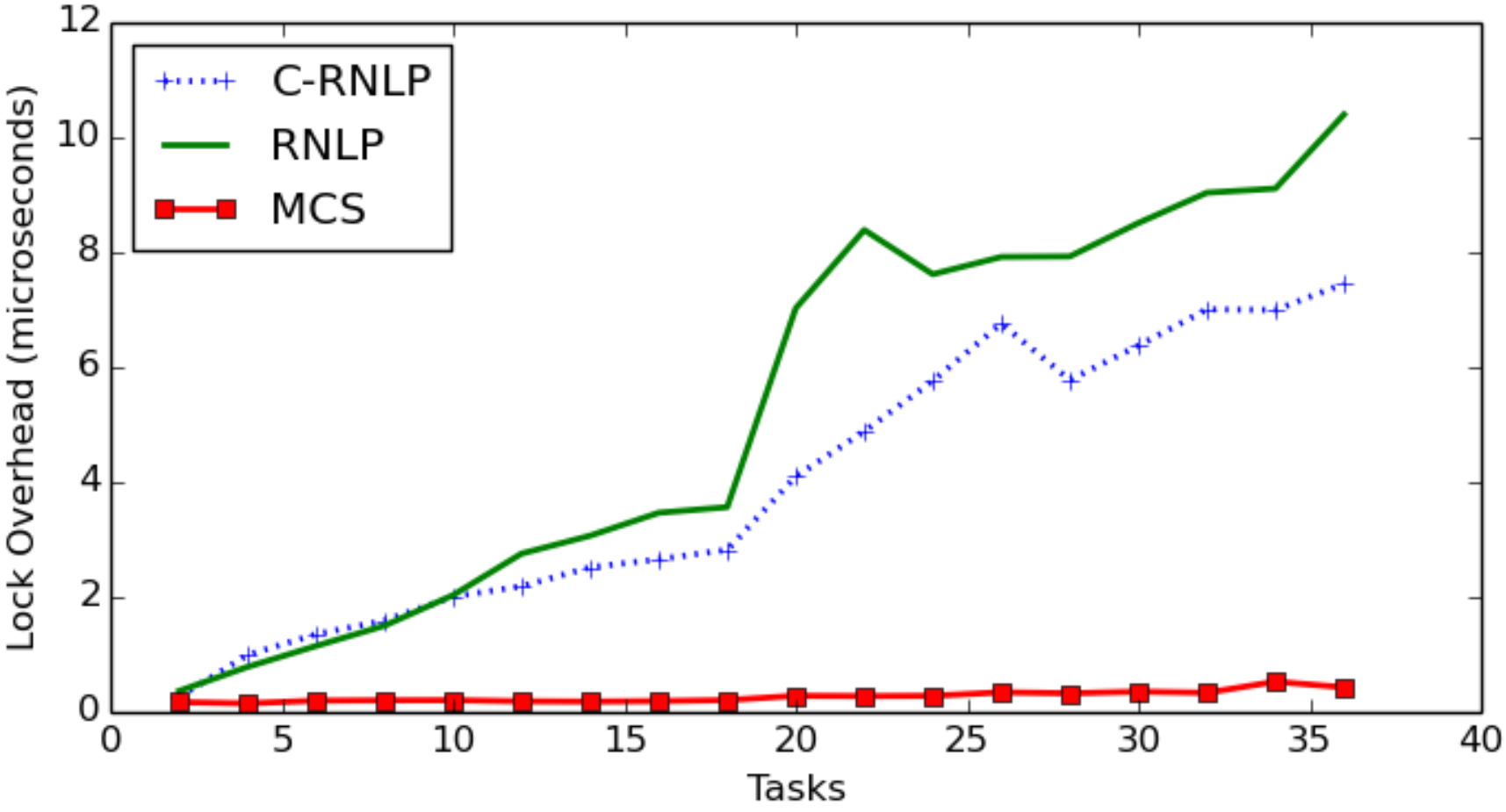
Observation 1. Runtime parallelism can increase worst-case lock and unlock overheads for the C-RNLP.

Tasks: 36
Available resources: 64

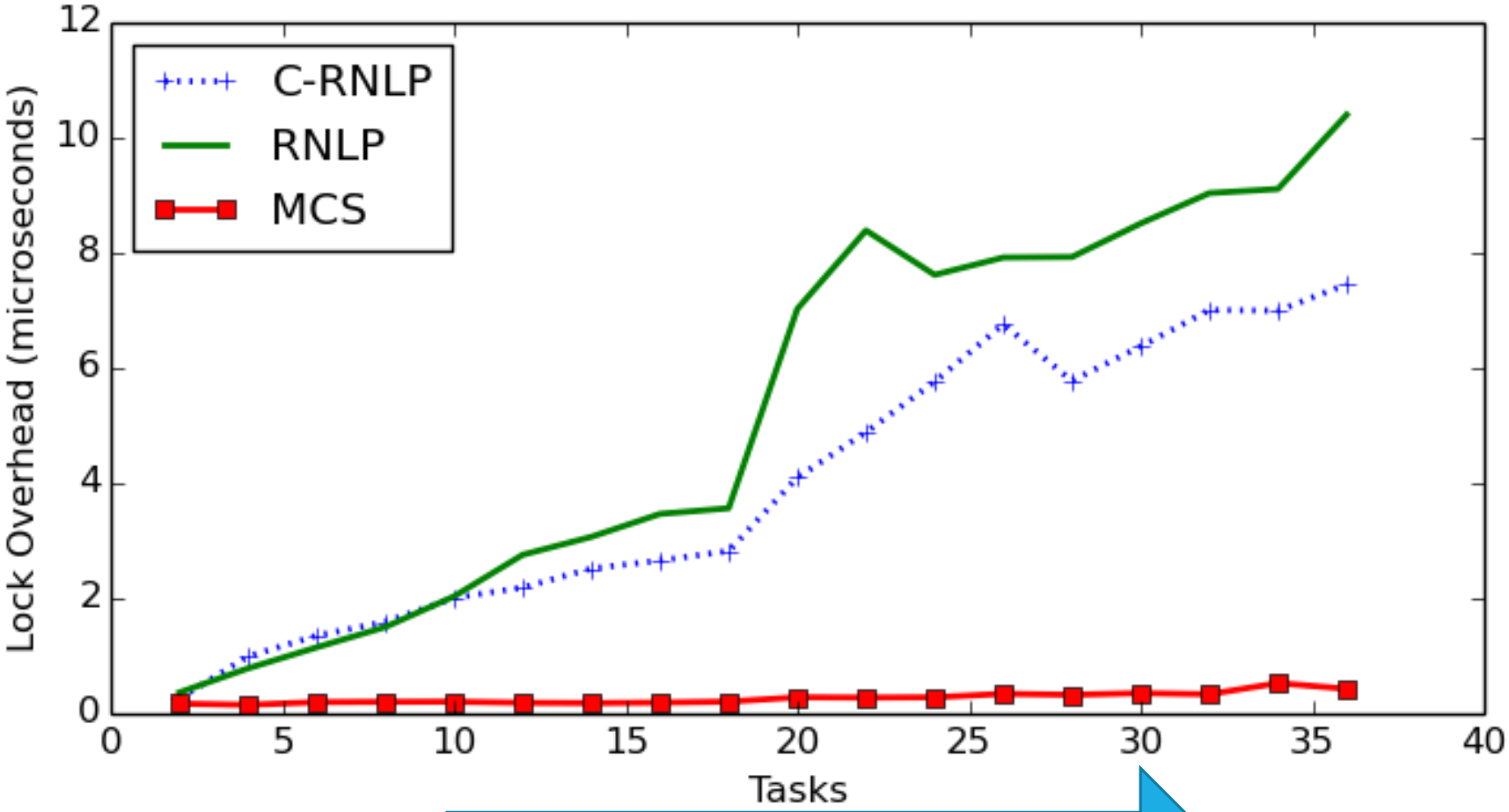



C-RNLP
RNLP
MCS
Lock Overhead (microseconds)
Tasks
0
2
4
6
8
10
12
0
5
10
15
20
25
30
35
40

Available resources: 64
Requested resources: 4

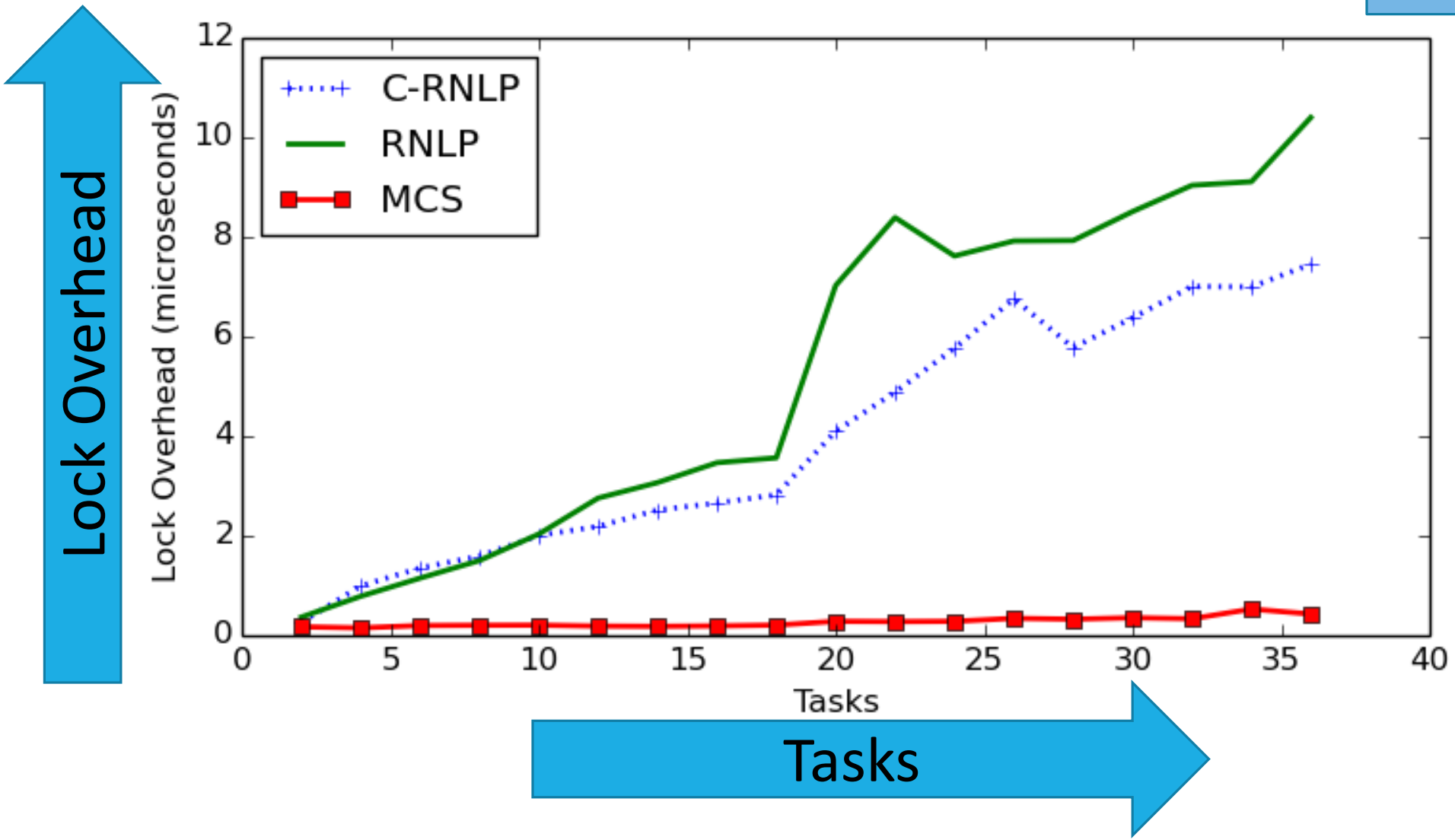


Available resources: 64
Requested resources: 4

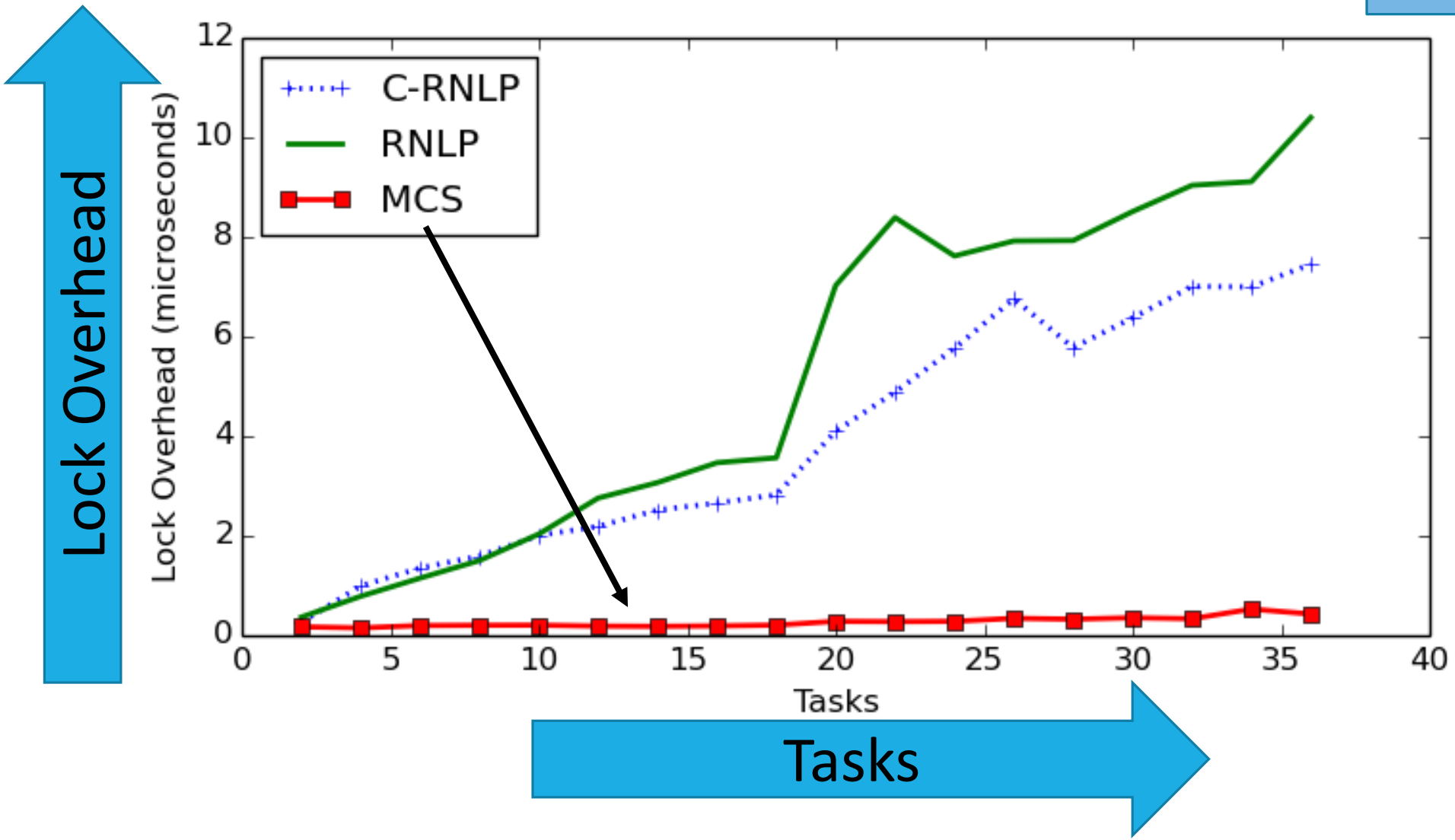


Tasks

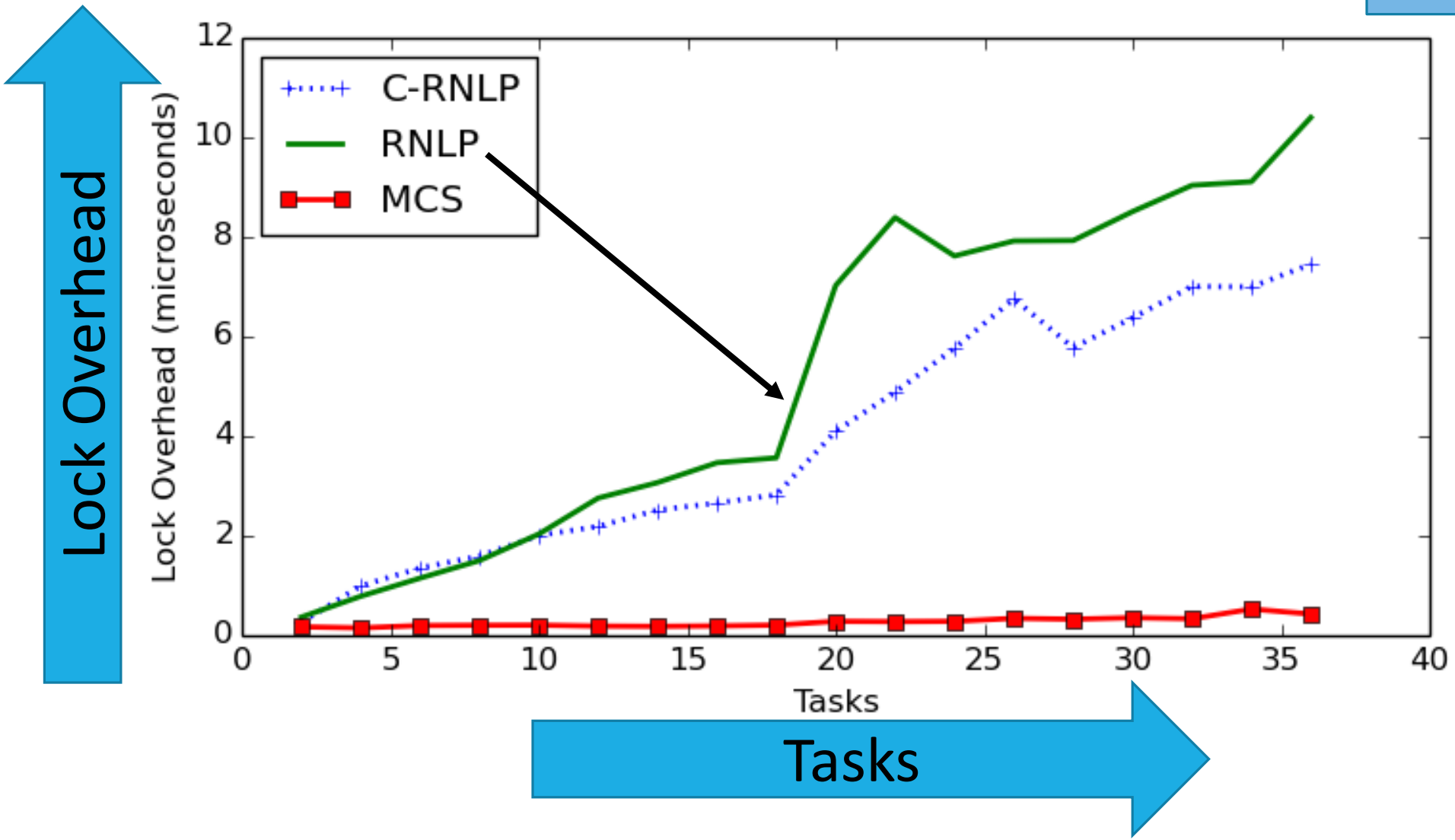
Available resources: 64
Requested resources: 4



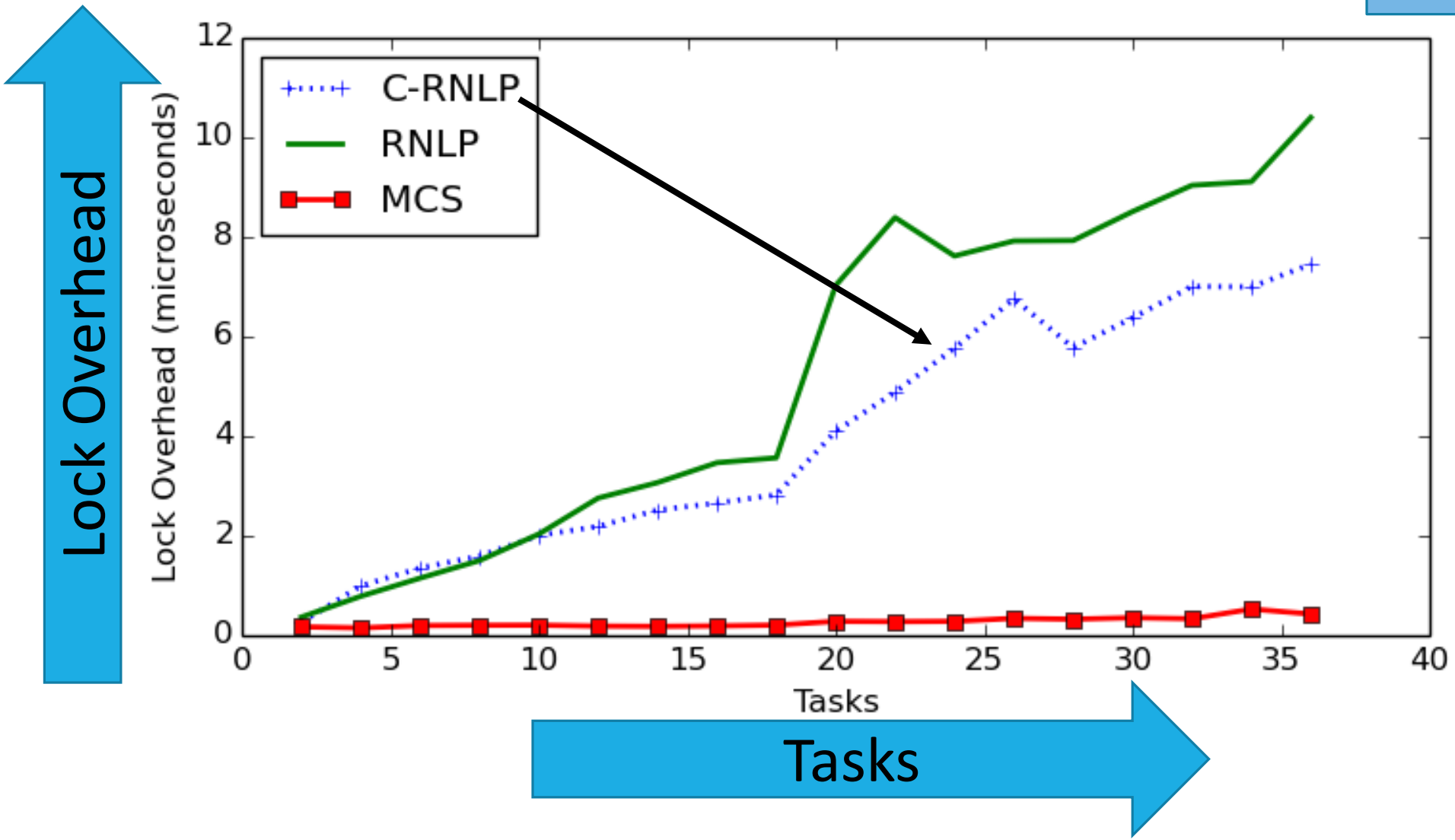
Available resources: 64
Requested resources: 4



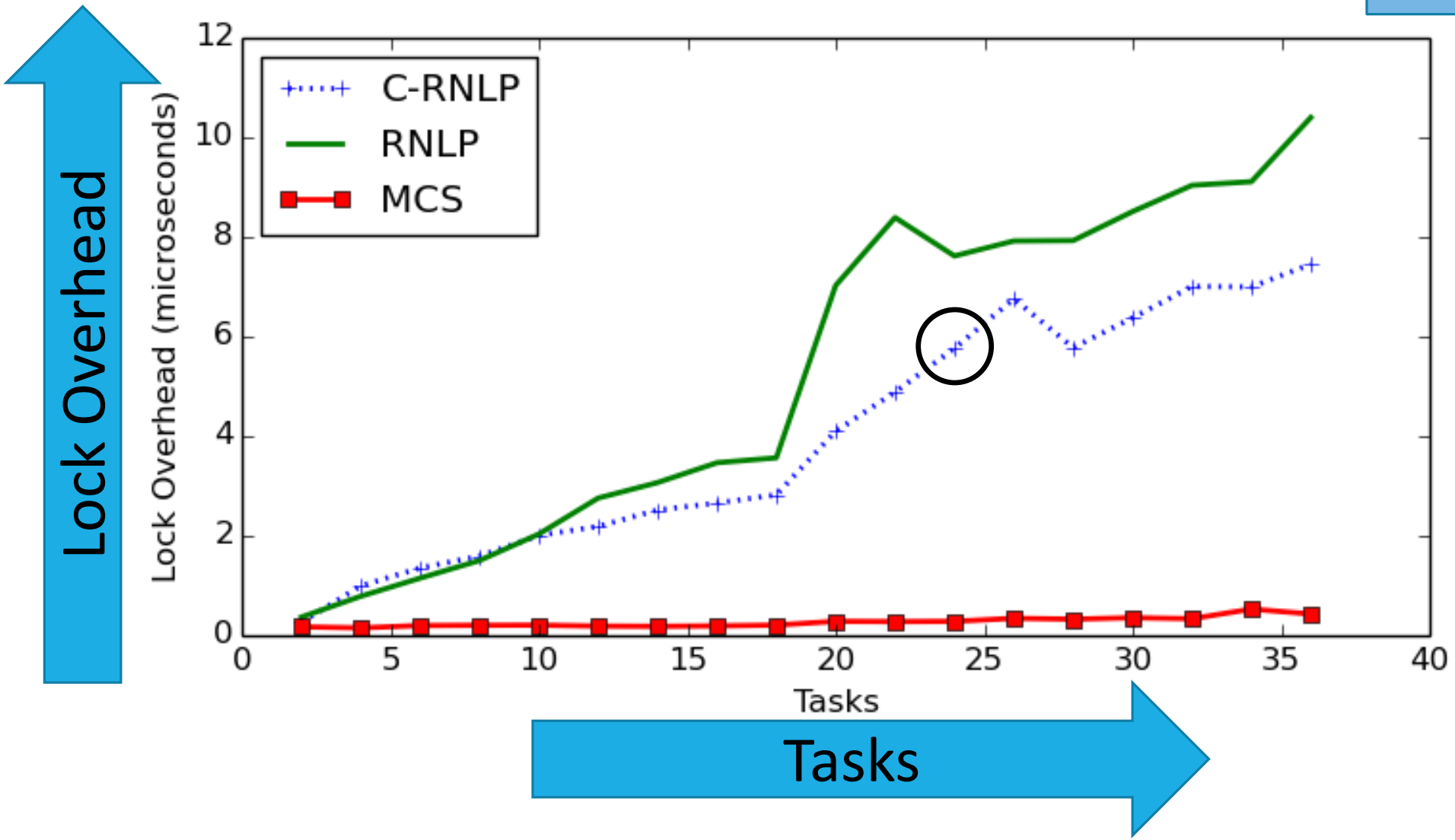
Available resources: 64
Requested resources: 4



Available resources: 64
Requested resources: 4

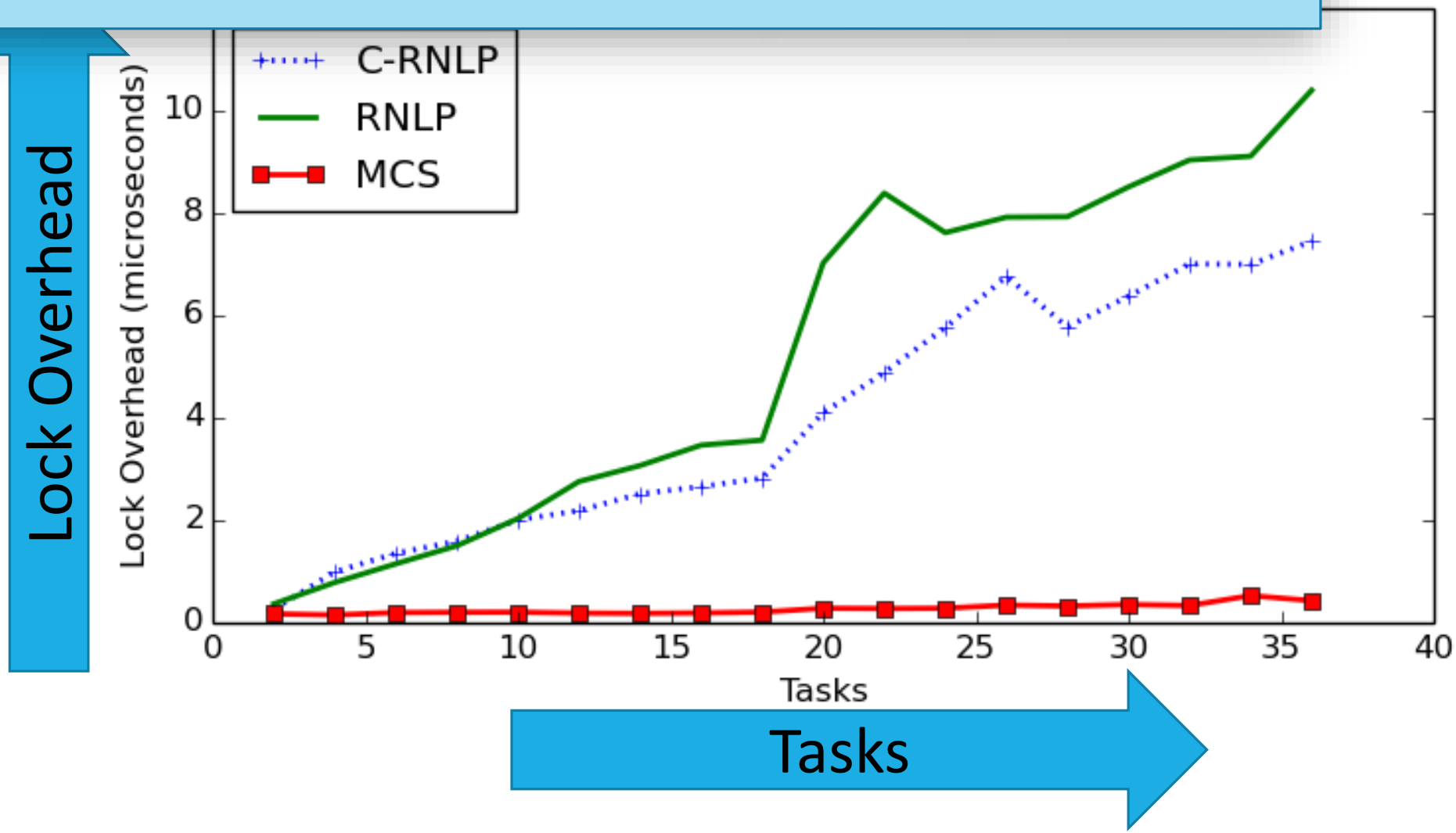


Available resources: 64
Requested resources: 4



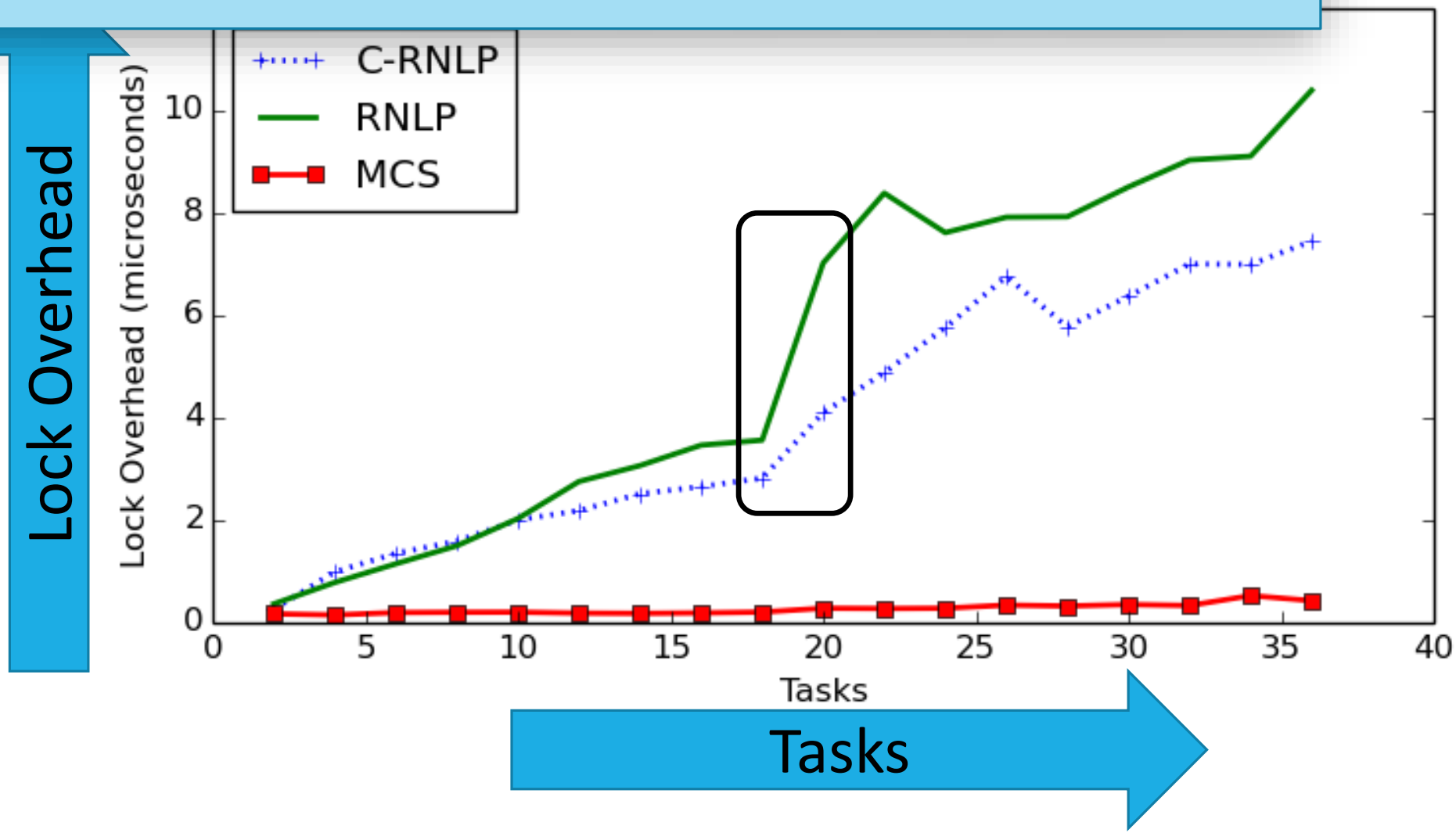
Observation 2. For the C-RNLP, observed overheads increase dramatically when resources are shared across sockets.

Available resources: 64
Requested resources: 4



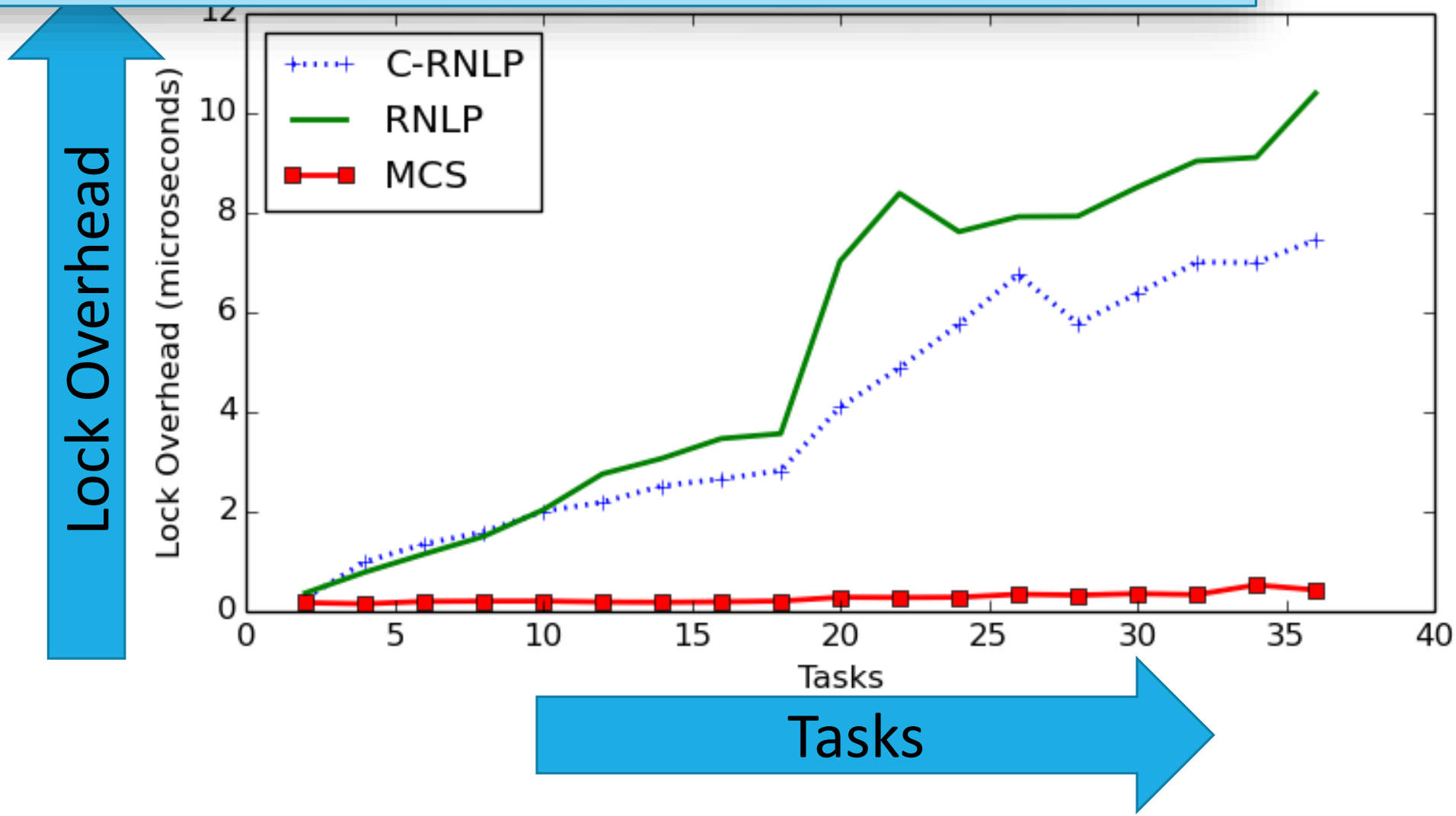
Observation 2. For the C-RNLP, observed overheads increase dramatically when resources are shared across sockets.

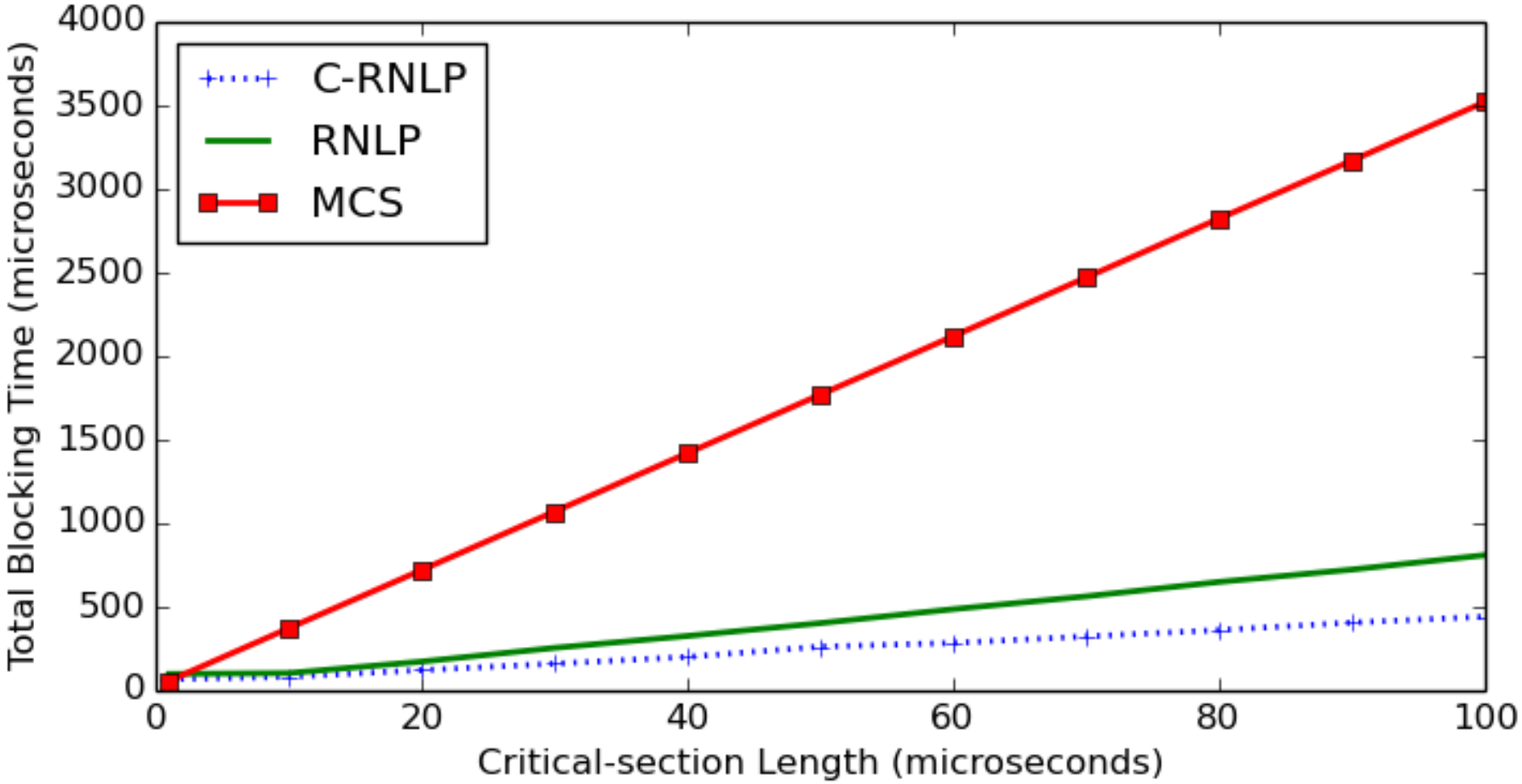
Available resources: 64
Requested resources: 4



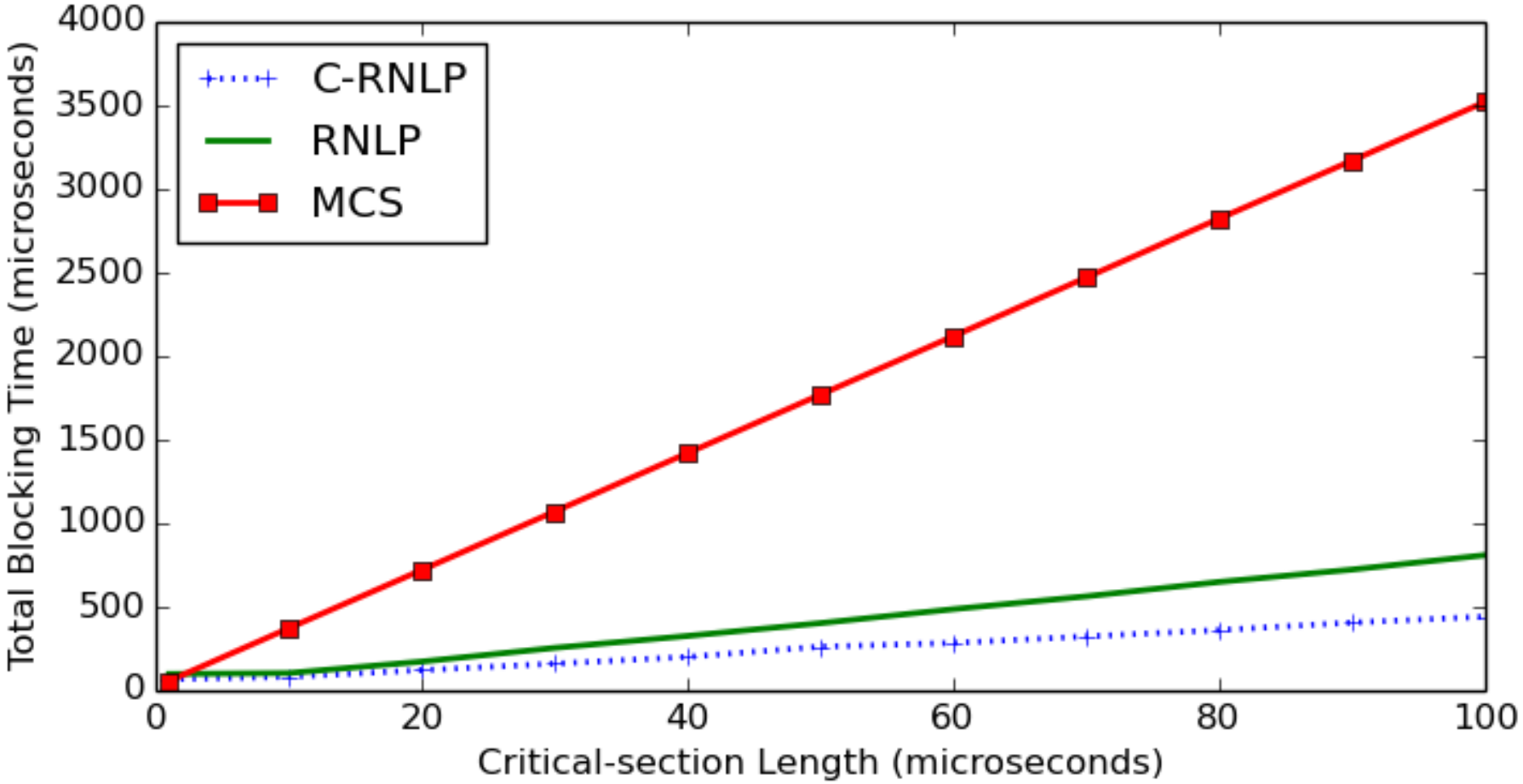
Observation 3. Fine-grained locking protocols have higher overhead than coarse-grained ones.

Available resources: 64
Requested resources: 4

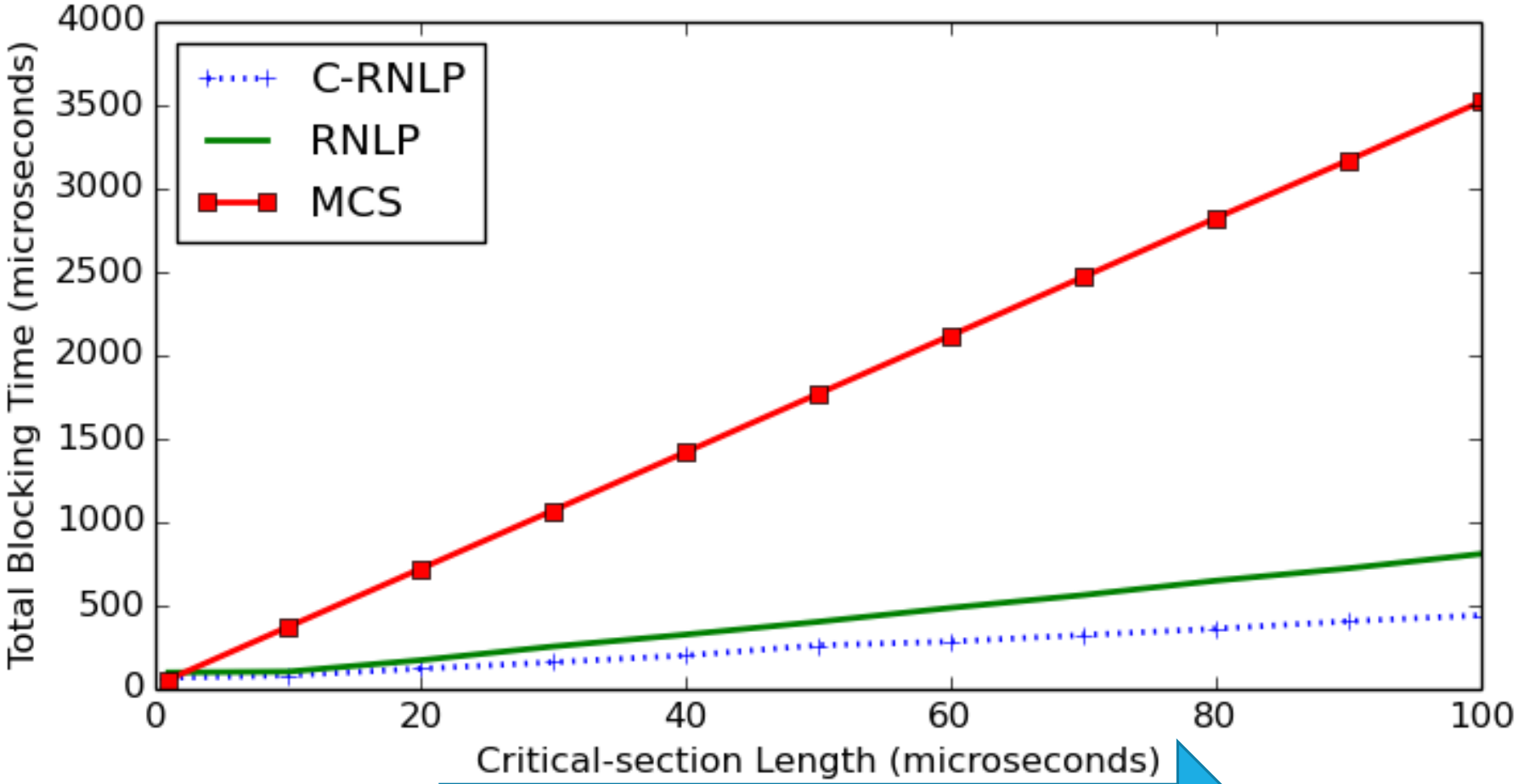



C-RNLP
RNLP
MCS
Total Blocking Time (microseconds)
Critical-section Length (microseconds)
0
500
1000
1500
2000
2500
3000
3500
4000
0
20
40
60
80
100

Tasks: 36
Available resources: 64
Requested resources: 2

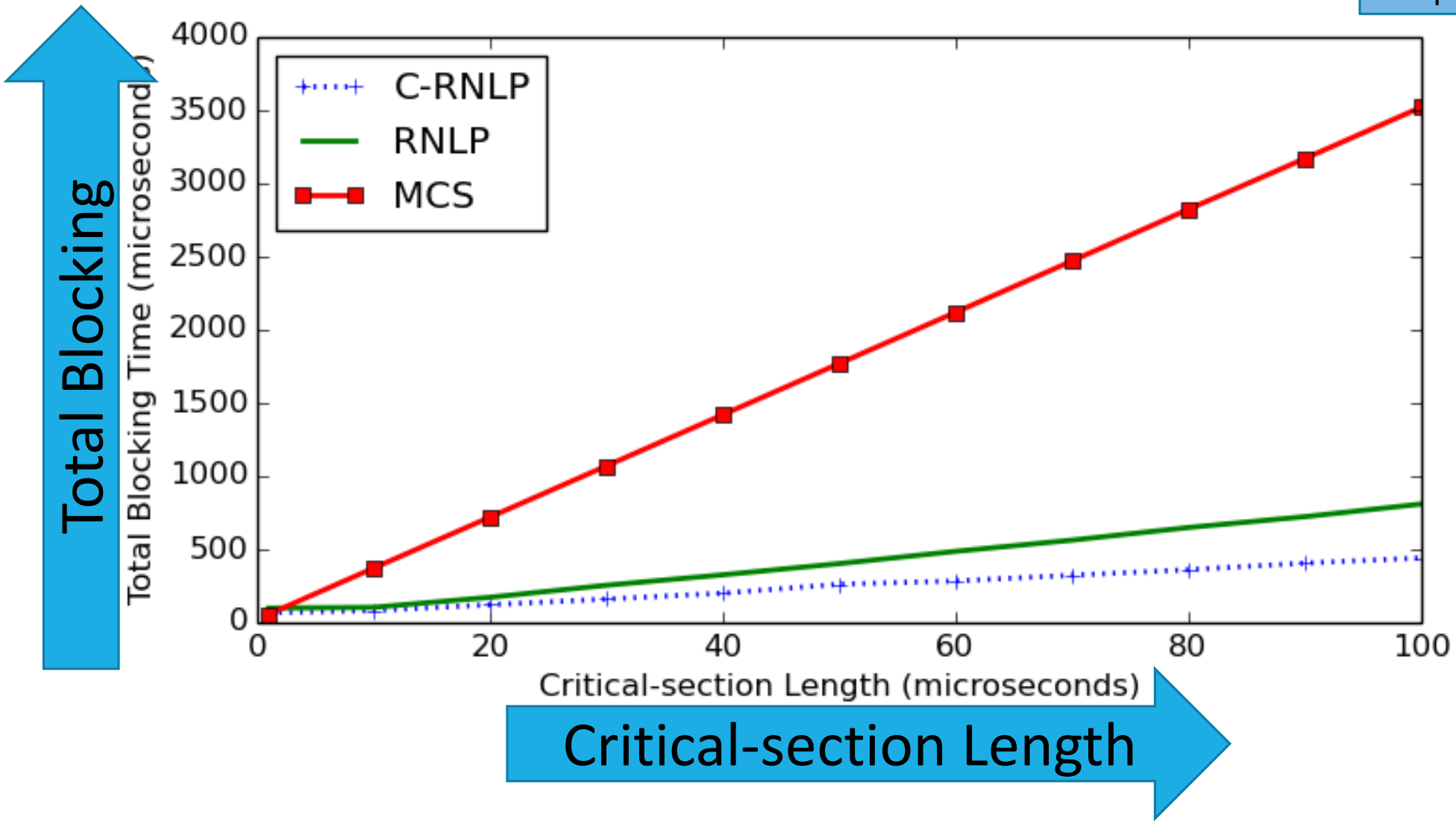


Tasks: 36
Available resources: 64
Requested resources: 2

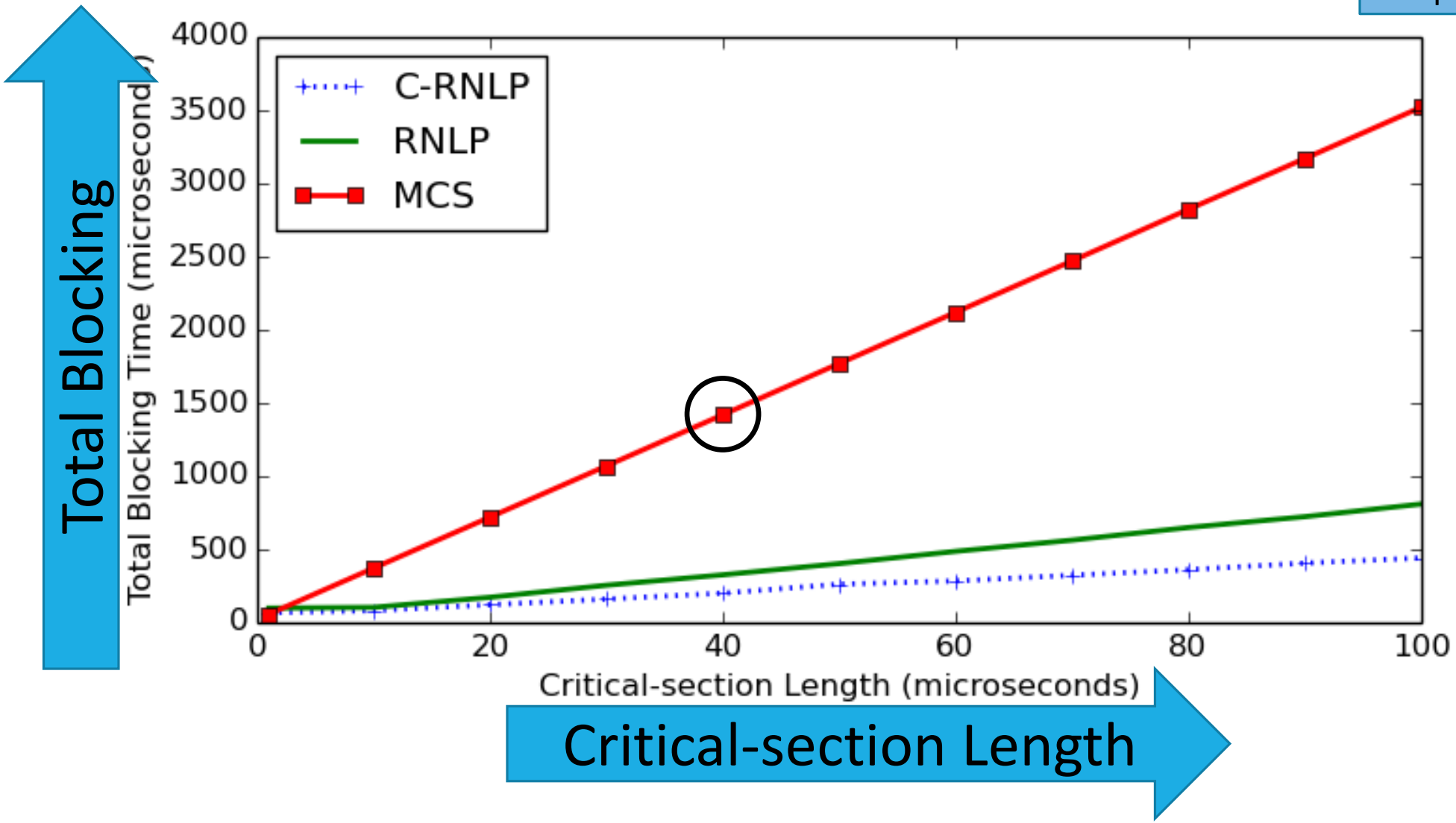


Critical-section Length

Tasks: 36
Available resources: 64
Requested resources: 2

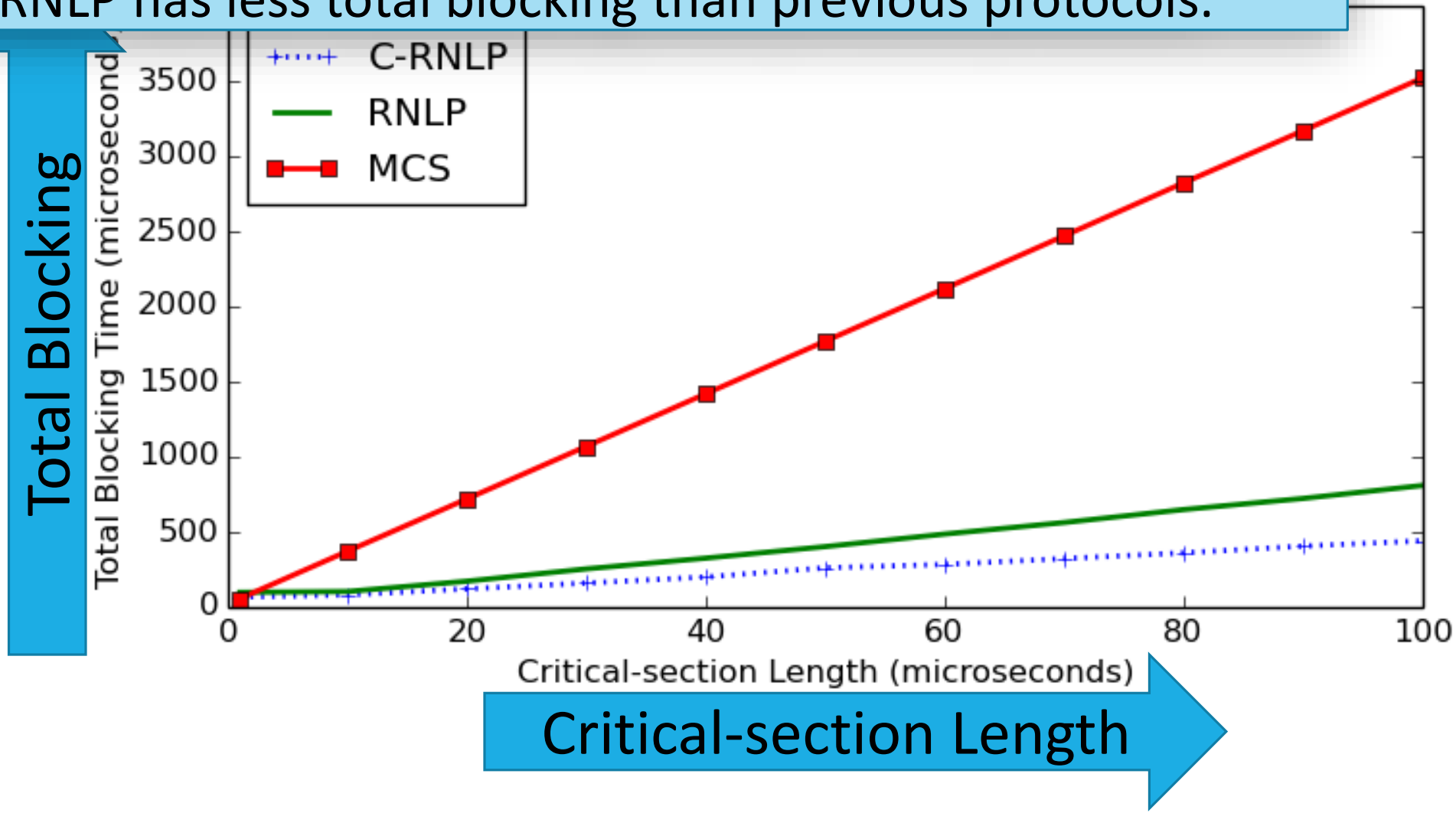


Tasks: 36
Available resources: 64
Requested resources: 2



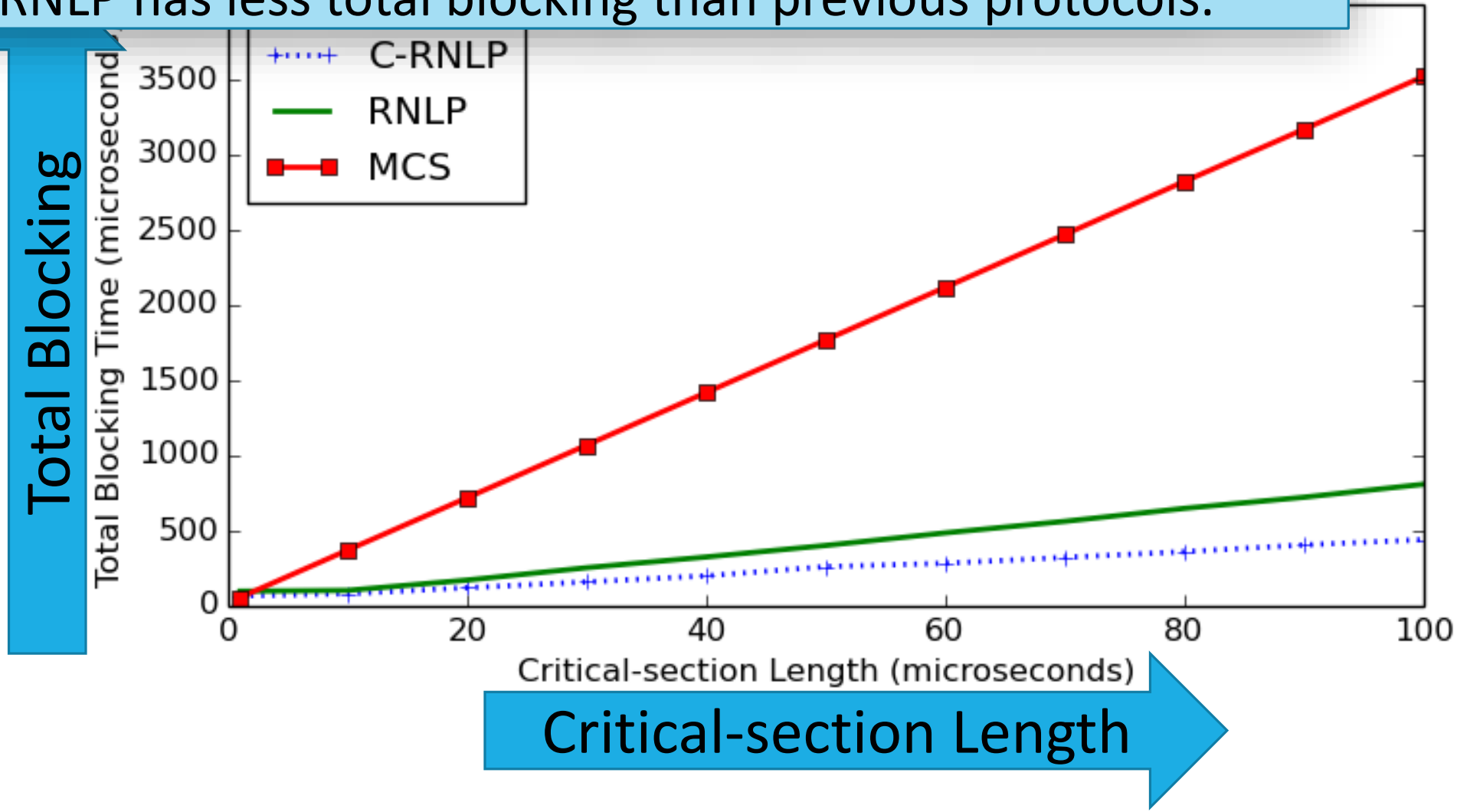
Observation 4. When critical-section lengths are greater than several microseconds and some parallelism is possible, the C-RNLP has less total blocking than previous protocols.

Tasks: 36
Available resources: 64
Requested resources: 2



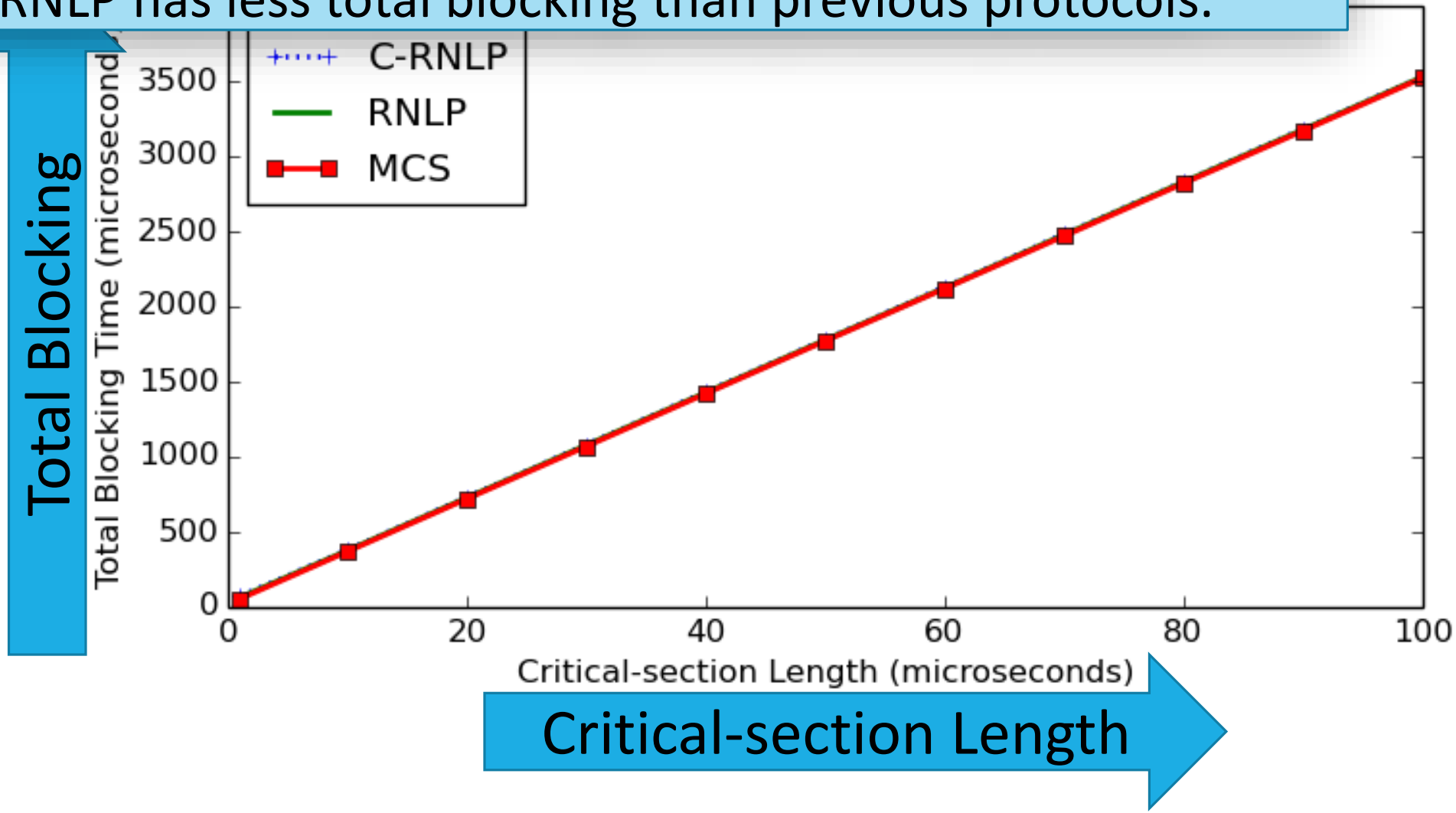
Observation 4. When critical-section lengths are greater than several microseconds and some parallelism is possible, the C-RNLP has less total blocking than previous protocols.

Tasks: 36
Available resources: 64
Requested resources: 2



Observation 4. When critical-section lengths are greater than several microseconds and some parallelism is possible, the C-RNLP has less total blocking than previous protocols.

Tasks: 36
Available resources: 64
Requested resources: 16



Contributions

C-RNLP, a contention-sensitive nested locking protocol

- Under certain conditions far out-performs existing protocols
- Otherwise does no worse than existing protocols

Future Work

Resource renaming

Lock server

C-RNLP variants corresponding to all RNLP variants

- short vs. long critical sections
- reader/writer

Questions?