

Contextual Concurrency Control

Sujin Park

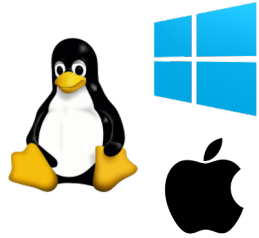
Irina Calciu

Taesoo Kim

Sanidhya Kashyap



Locks in everywhere!



Operating systems



Cloud services



Data processing systems



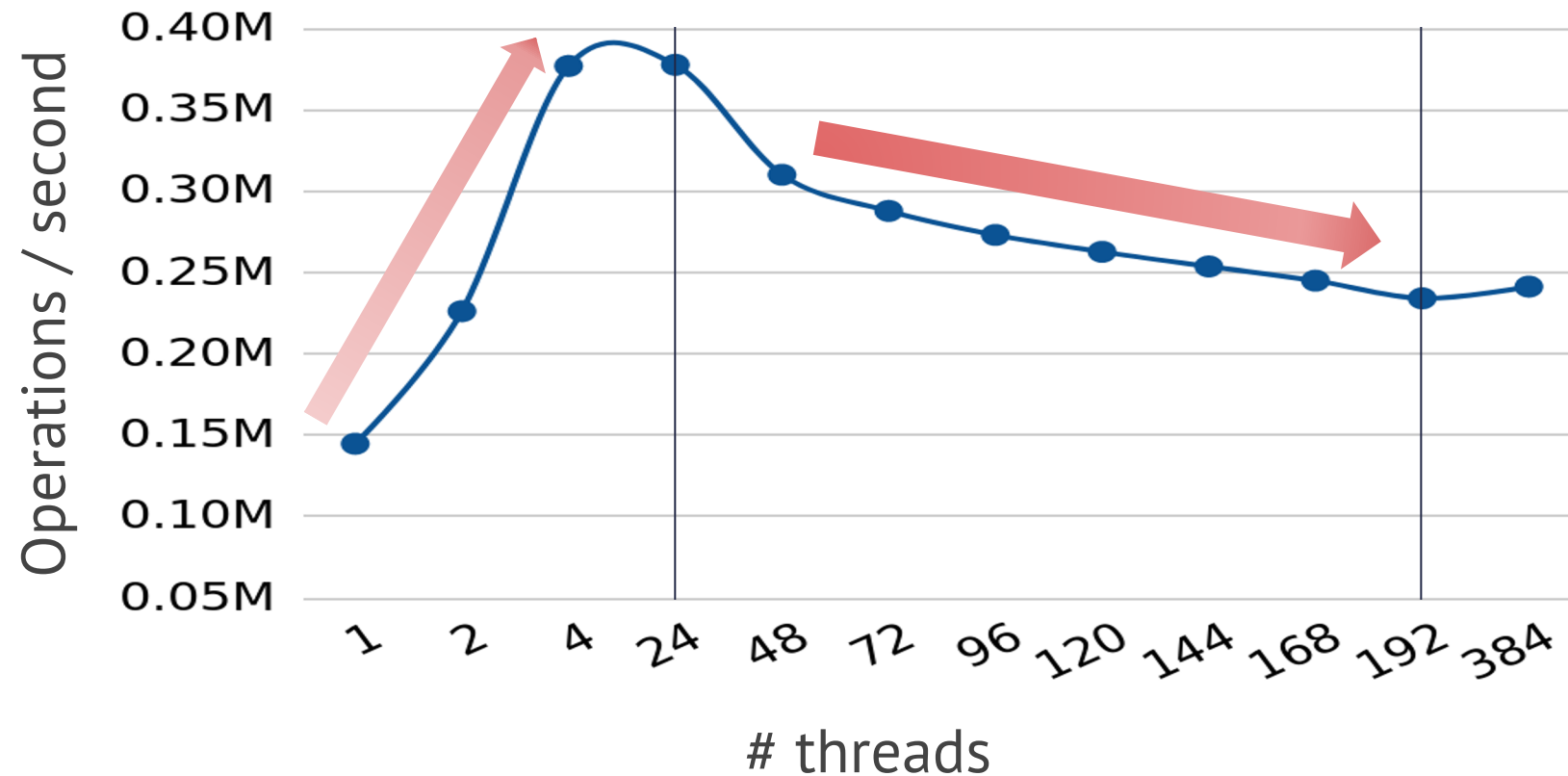
Databases

Synchronization mechanisms

Basic building block for designing applications

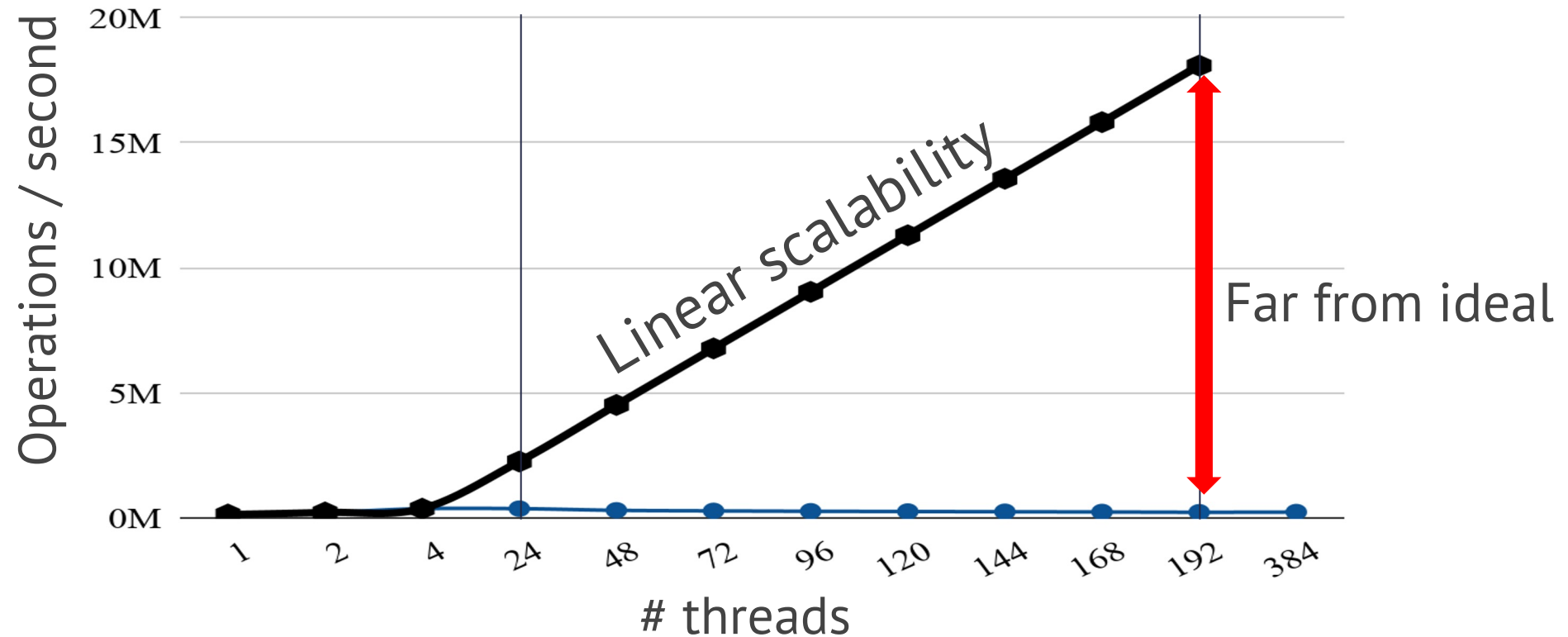
Locks are critical for application performance

Typical application performance on a multicore machine

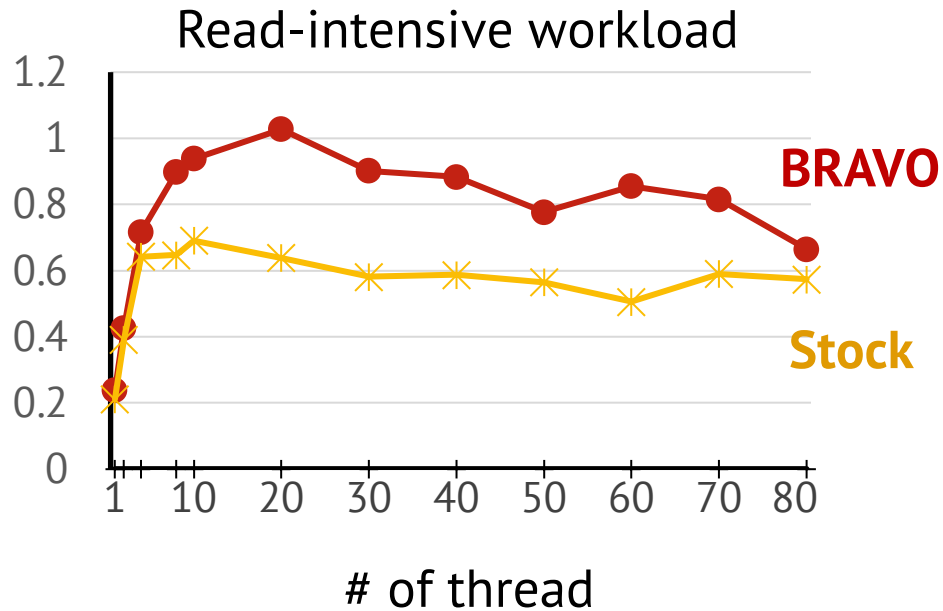


Locks are critical for application scalability

Typical application performance on a manycore machine



One lock cannot rule all of them!



Evolving hardware



Various applications & requirements



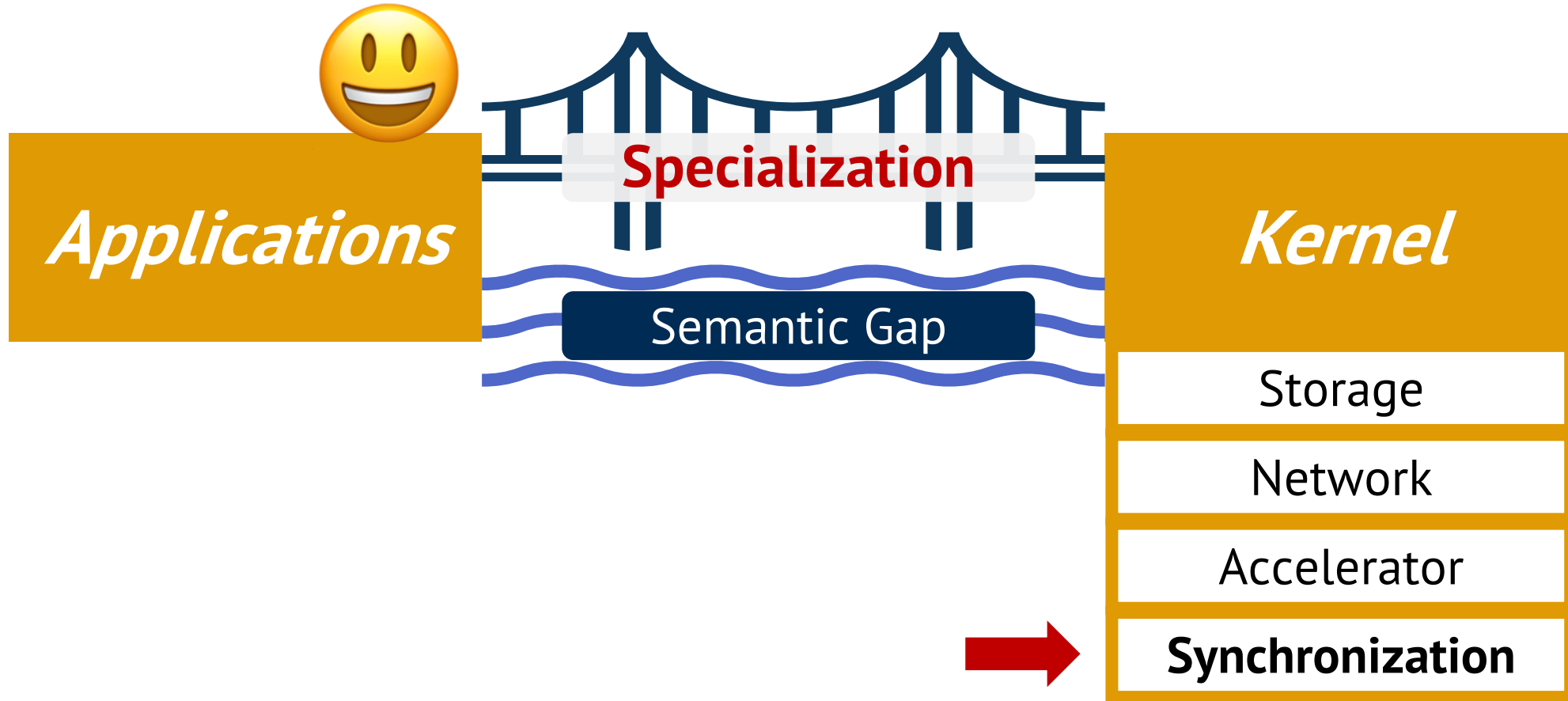
mongoDB.



Specialization bridges the semantic gap



Specialization bridges the semantic gap



Can we tune lock policy on the fly?

Contextual Concurrency Control

New paradigm to tune synchronization mechanism
from user space

Need for user-defined locks on the fly

Lock implementations are application agnostic

Only few locks contend for a given application

May need a variant of a lock based on the workload

CONCORD Framework

Lock implementations are application agnostic

→ Let application developers to tune locks in the kernel on the fly

Only few locks contend for a given application

→ Modify set of locks at various granularities

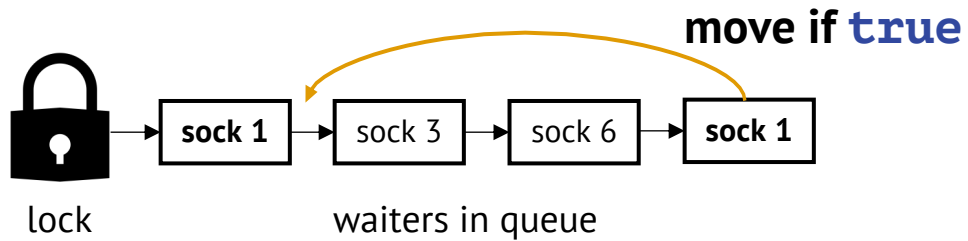
May need a variant of a lock based on the workload

→ Exposes set of APIs to modify lock algorithms

CONCORD Overview

User

1 User create
lock policy



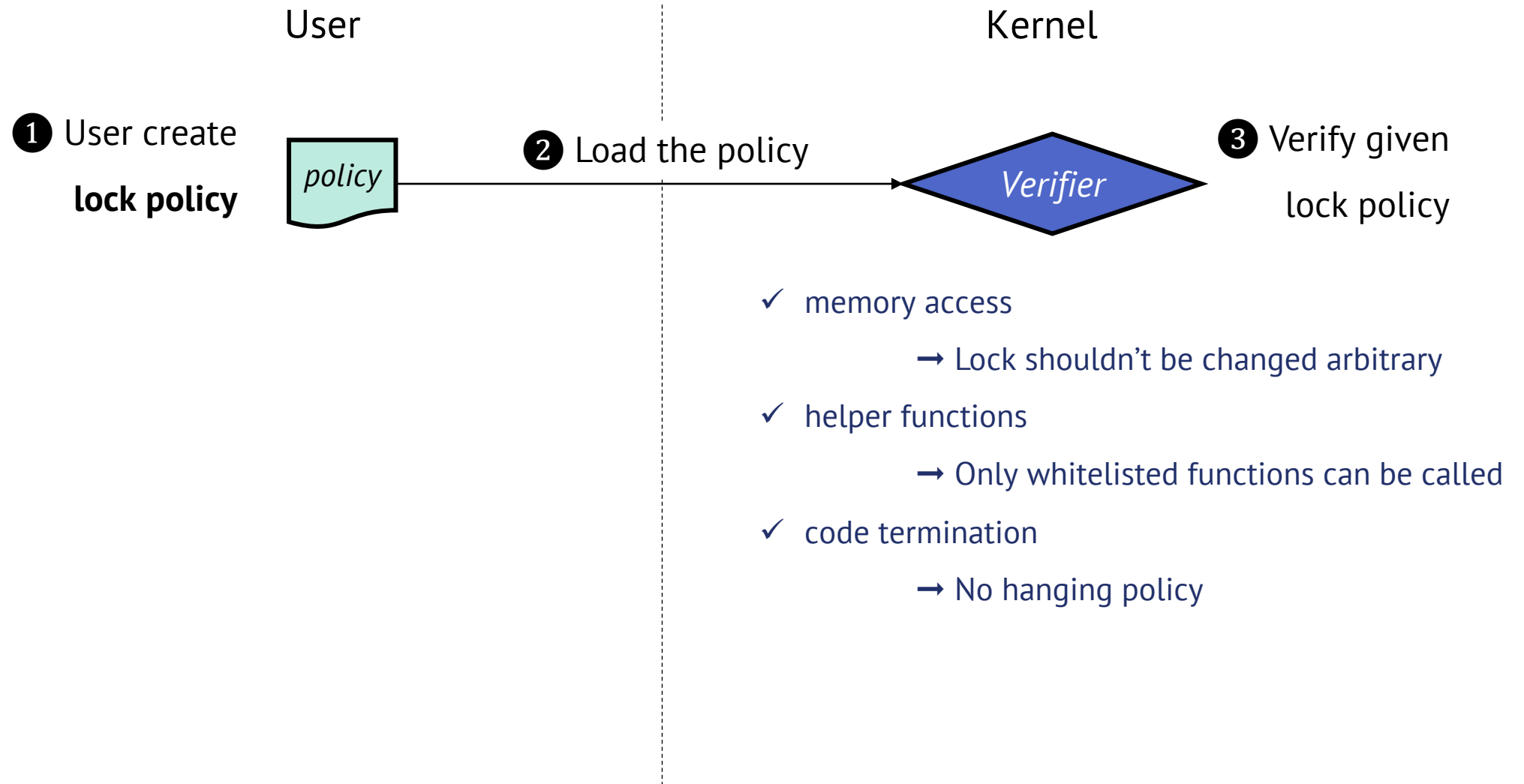
Example :

```
bool cmp_node(node* pre, node* cur){  
    return (pre->sock == cur->sock) ;  
}
```

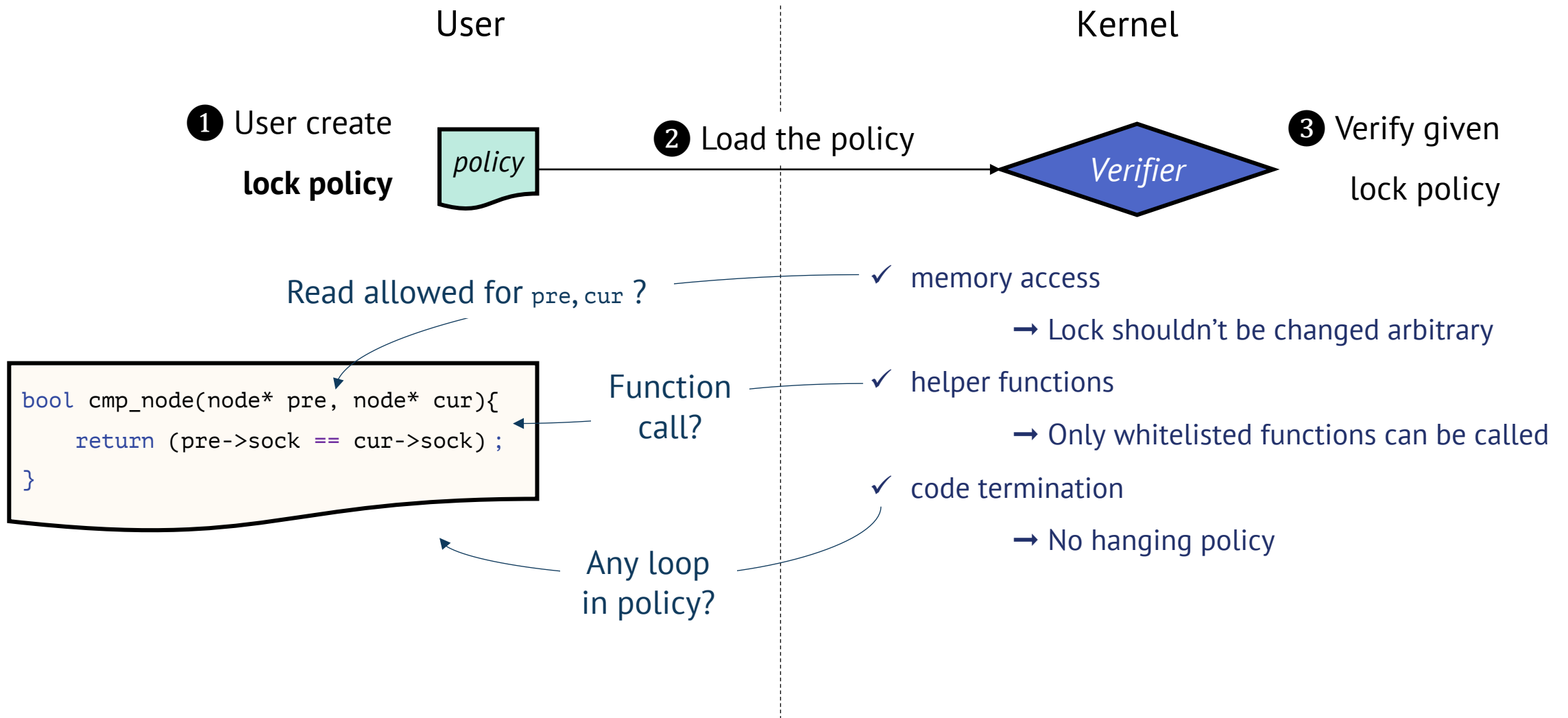
Grouping node from same socket

Kernel

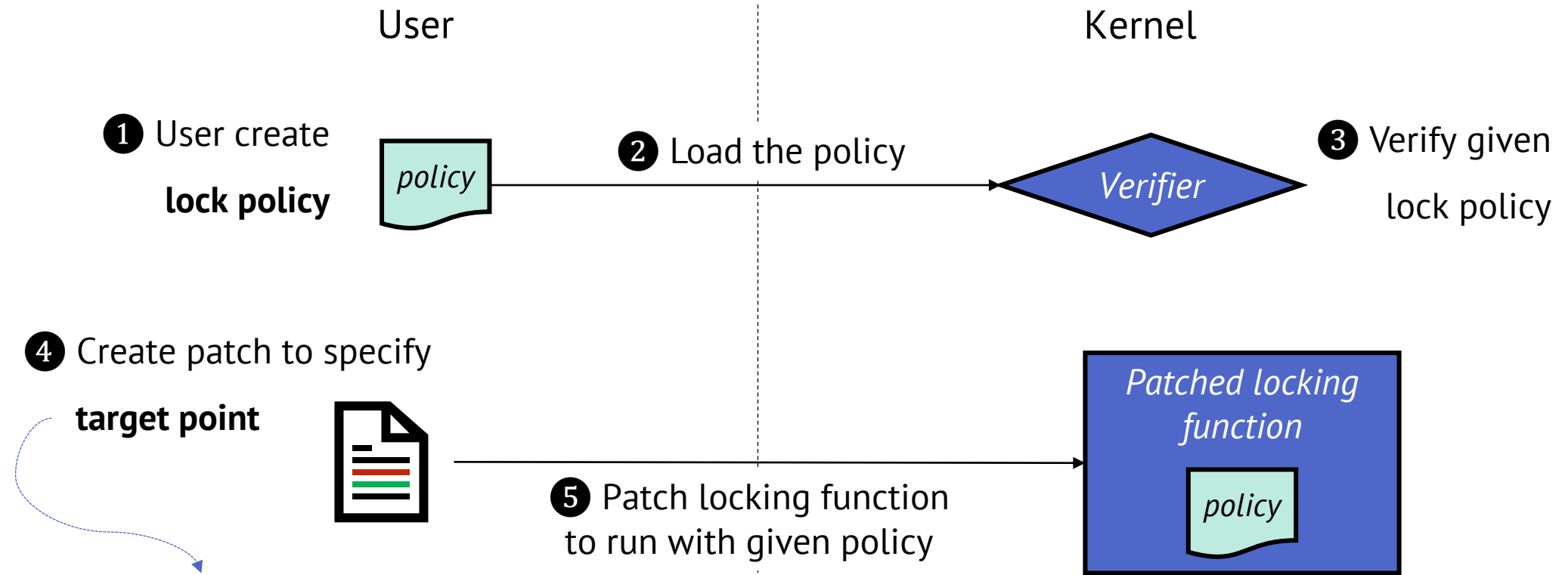
CONCORD Overview



CONCORD Overview



CONCORD Overview



- All spinlocks in the kernel
- Spinlocks used in filesystem
- A spinlock used in an inode

Safety and APIs

Reordering waiters

- `bool cmp_node(lock, node, node){}`
- `bool skip_shuffle(lock, node){}`

 Flexibility to change lock on the fly

 Fairness

Profiling

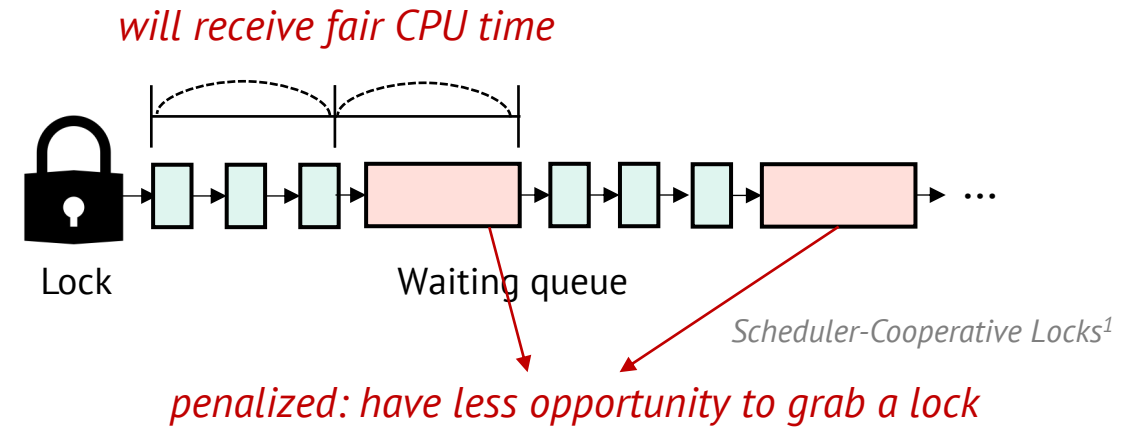
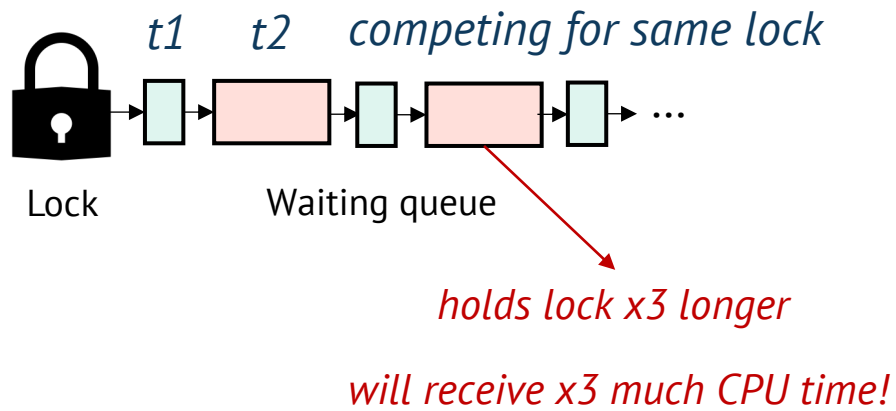
- `void lock_acquire(lock){}`
- `void lock_contended(lock){}`
- `void lock_acquired(lock){}`
- `void lock_release(lock){}`

 Fine-grained lock profiling

 Increase critical section

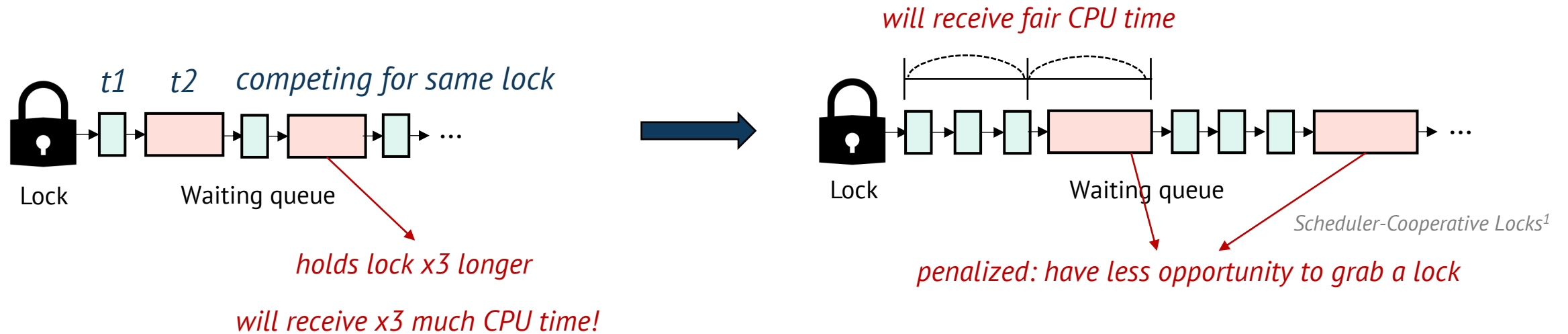
Ensure mutual exclusion & safe from crashing

Usecase



1. Avoiding Scheduler Subversion using Scheduler-Cooperative Locks. Eurosys'20

Usecase

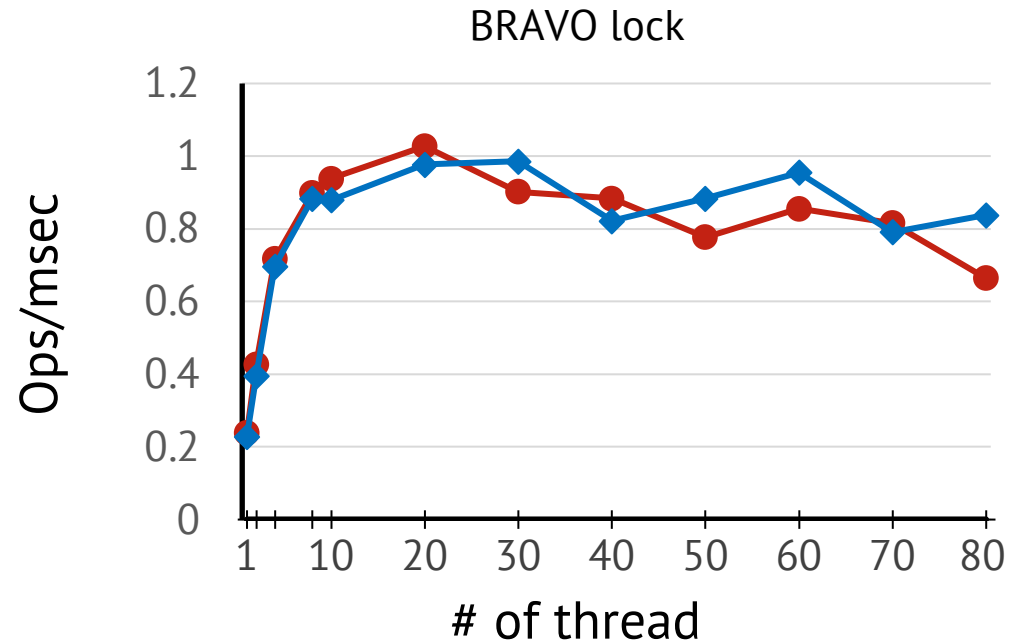


Will this fairness always be beneficial?

Let application developers enforce this fairness only when needed

1. Avoiding Scheduler Subversion using Scheduler-Cooperative Locks. Eurosys'20

Overhead of CONCORD



- Overhead of **CONCORD-lock** compared to **pre-compiled lock**
- Almost negligible overhead (And now **we can change lock on the fly!**)

Conclusions

- Kernel locks are critical for application performance and scalability
 - Out of the reach of application developers
- **C3 : Contextual Concurrency Control**
 - Let userspace application to fine tune concurrency control
- CONCORD Framework
 - Exposes a set of APIs
 - Apply to specific target locks (instead of all locks in the kernel)
 - Change locks on the fly with minimal overhead