Fast on Average, Predictable in the Worst Case

Exploring Real-Time Futexes in LITMUS^{RT}

Roy Spliet, **Manohar Vanga**, Björn Brandenburg, Sven Dziadek

IEEE Real-Time Systems Symposium 2014
December 2-5, 2014
Rome, Italy

Max
Planck
Institute
for
Software Systems

Real-Time Locking API

Mutex API

```
kernel_do_lock();
// critical section
kernel_do_unlock();
```

Real-Time Locking API

Mutex API

```
kernel_do_lock();
// critical section
kernel_do_unlock();
```

No other task is currently holding the lock

No other task is waiting for the lock

Operations are typically ***uncontended*** at runtime

Futexes: Fast Userspace Mutexes

A **mechanism (API) in the Linux** kernel

- For optimizing the uncontended case of locking protocol implementations

- Avoid kernel invocation during uncontended operations

Futexes: Fast Userspace Mutexes

A **mechanism (API) in the Linux** kernel

- For optimizing the uncontended case of locking protocol implementations
- Avoid kernel invocation during uncontended operations

Why optimize the uncontended case?

- Improved throughput for soft real-time workloads
- Increasing available slack time in the system

Real-Time Locking Implementations

	PRIO_INHERIT (Priority Inheritance Protocol)
Futex Implementation	✓

↑

Linux

Real-Time Locking Implementations

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)
Futex Implementation	✓	✗

Linux

Real-Time Locking Implementations

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✗	✗	✗

Linux

LITMUS^{RT}

Real-Time Locking Implementations

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✗	✗	✗

Linux

LITMUS^{RT}

Choice between futex implementation and better analytical properties

Real-Time Locking Implementations

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✗	✗	✗

Linux

LITMUS^{RT}

Choice between futex implementation and better analytical properties

Why is it challenging to have both?

Real-Time Locking Implementations

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✗	✗	✗

Reactive

Anticipatory

Real-Time Locking Protocol Dichotomy

Reactive Locking Protocols	Anticipatory Locking Protocols
React to contention on a lock **only when it occurs**.	**Anticipate problem scenarios** and take measure to minimize priority-inversion (pi) blocking.

Real-Time Locking Protocol Dichotomy

Reactive Locking Protocols	Anticipatory Locking Protocols
React to contention on a lock **only when it occurs**.	**Anticipate problem scenarios** and take measure to minimize priority-inversion (pi) blocking.
Simple futex implementation	**Tricky** to implement futexes without **violating semantics**

Contributions

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✗	✗	✗

Contributions

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✓	✓	✓

Design and implementation of futexes for **three anticipatory real-time locking protocols**

Contributions

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✓	✓	✓
Observed overhead reduction:			**92%**	**85%**	**84%**

Design and implementation of futexes for **three anticipatory real-time locking protocols**

Contributions

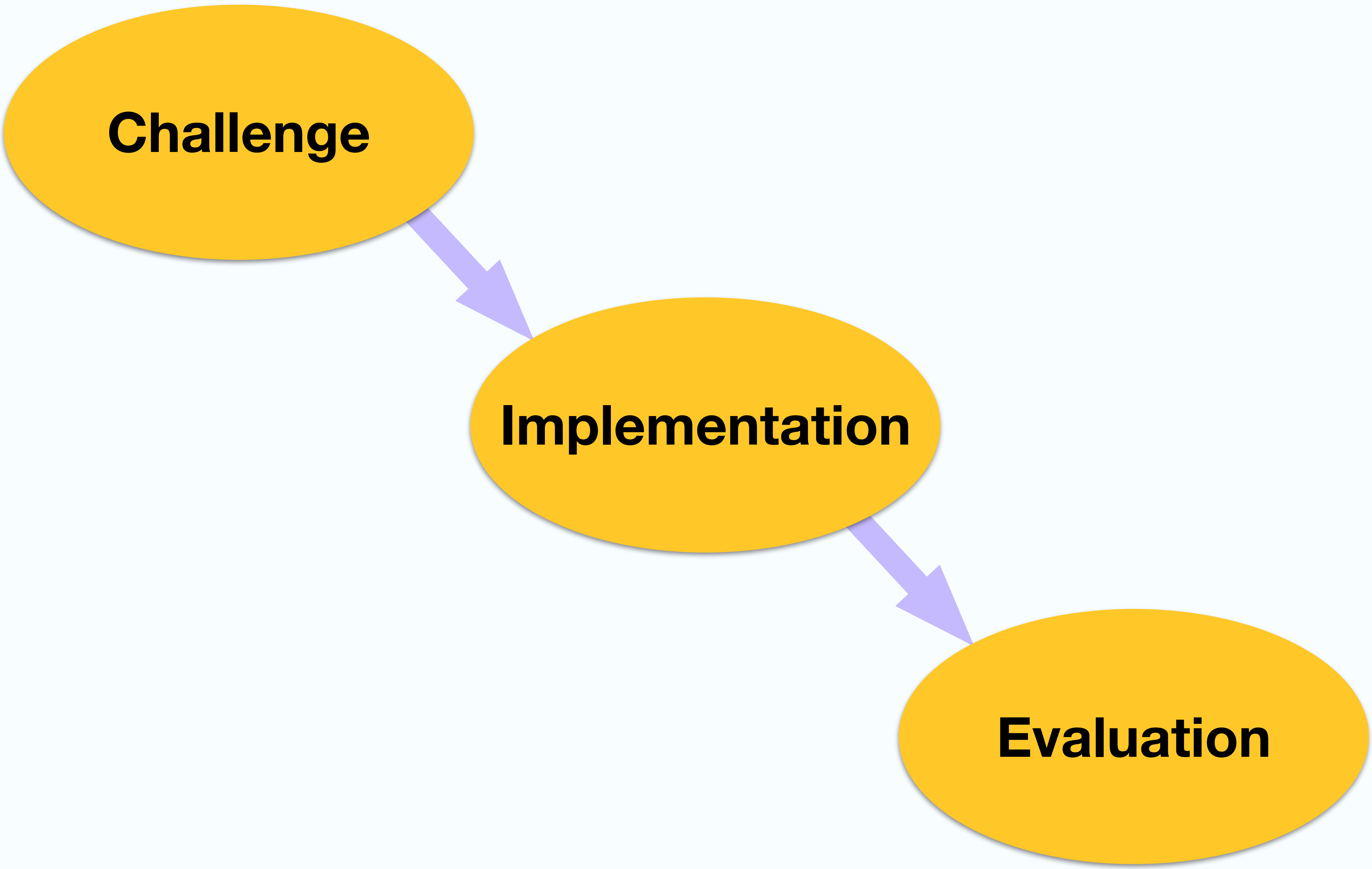
	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✓	✓	✓
Observed overhead reduction:			**92%**	**85%**	**84%**

Design and implementation of futexes for **three anticipatory real-time locking protocols**

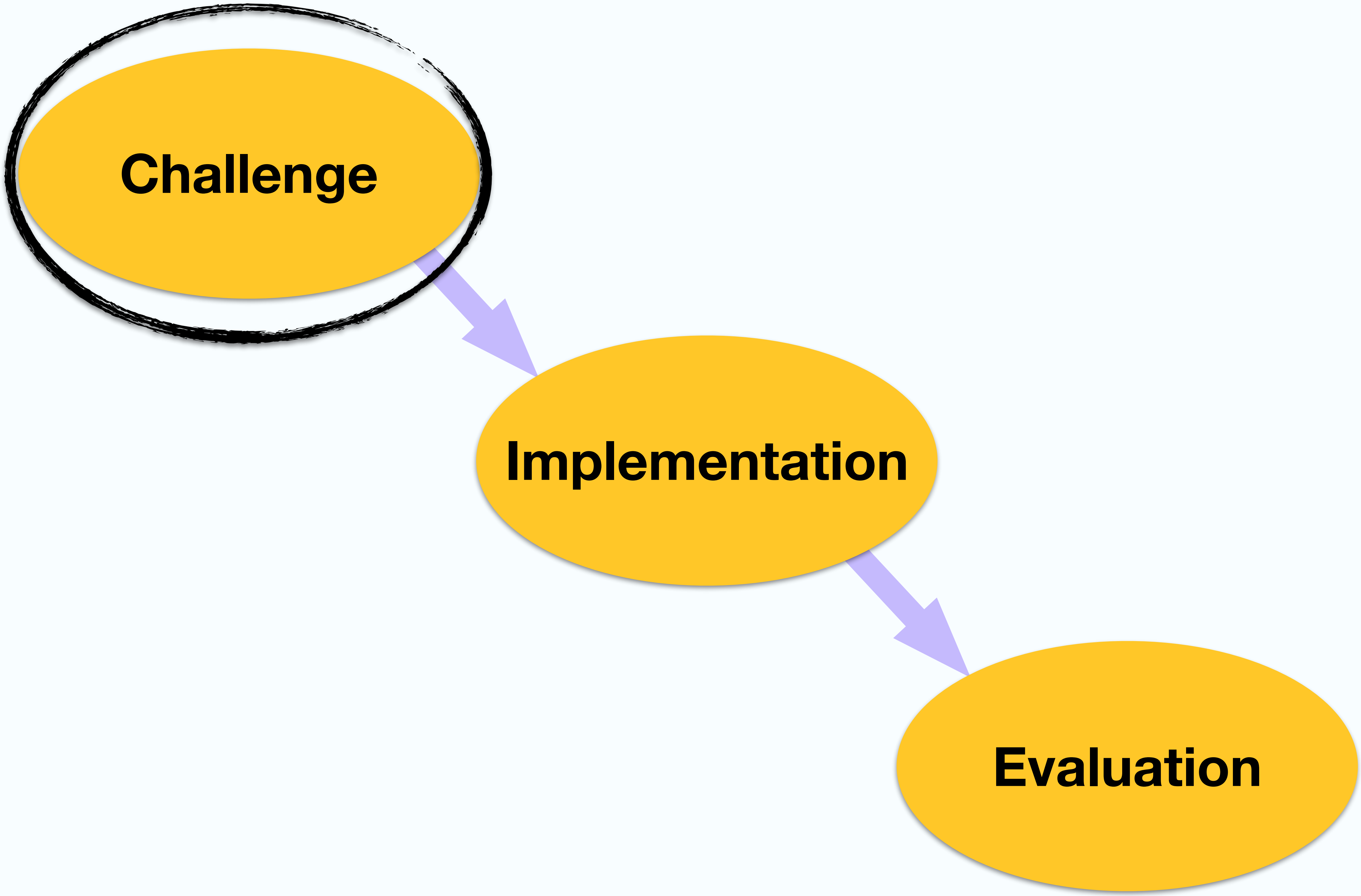
Method of **using page faults** to implement futexes

Overview of Talk

Overview of Talk



Overview of Talk



Futexes — Overview and Intuition

The problem with the vanilla approach

Mutex API

```
kernel_do_lock();
// critical section
kernel_do_unlock();
```

Futexes — Overview and Intuition

The problem with the vanilla approach

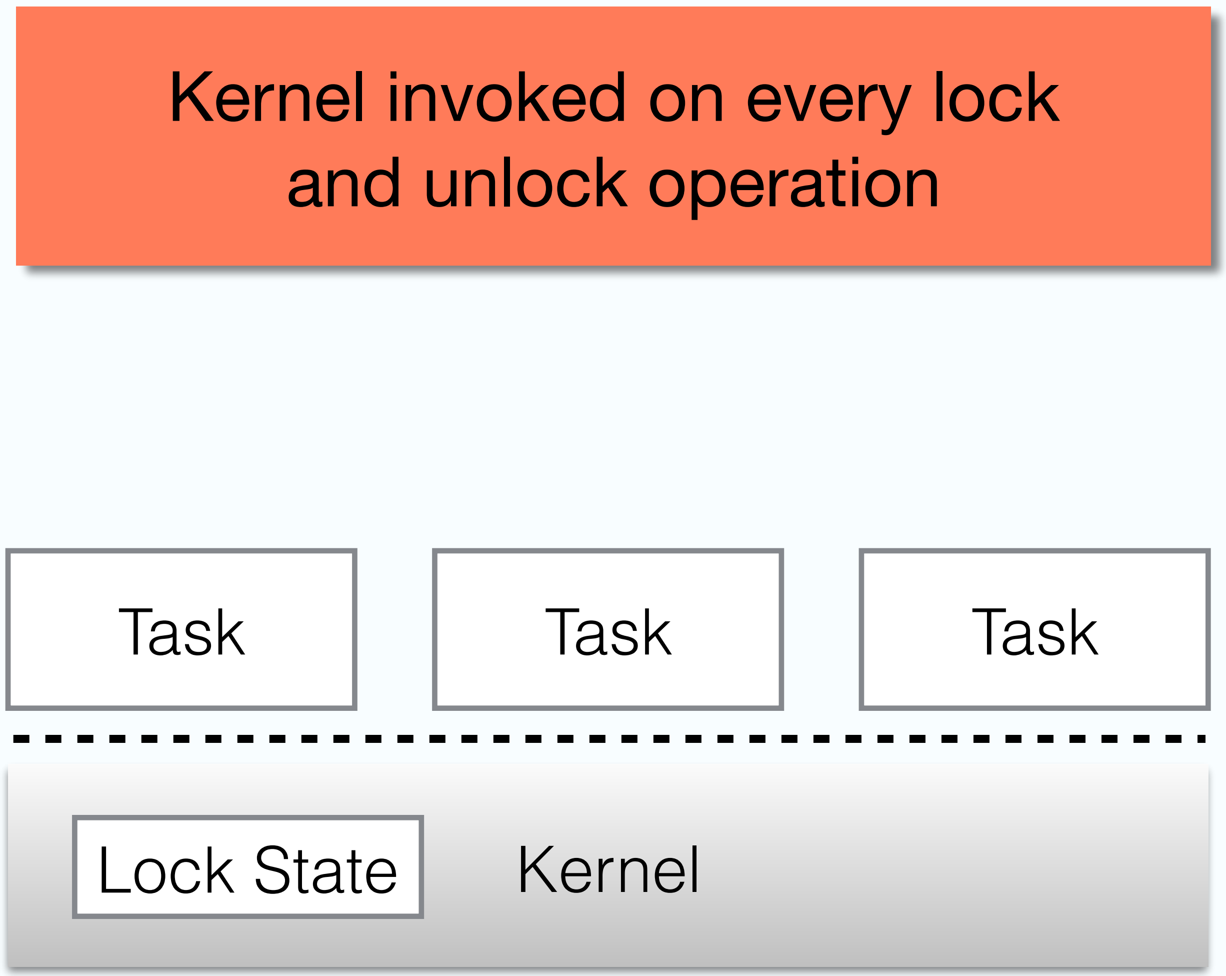
Kernel invoked on every lock and unlock operation

Mutex API

```
kernel_do_lock();
// critical section
kernel_do_unlock();
```

Futexes — Overview and Intuition

The problem with the vanilla approach

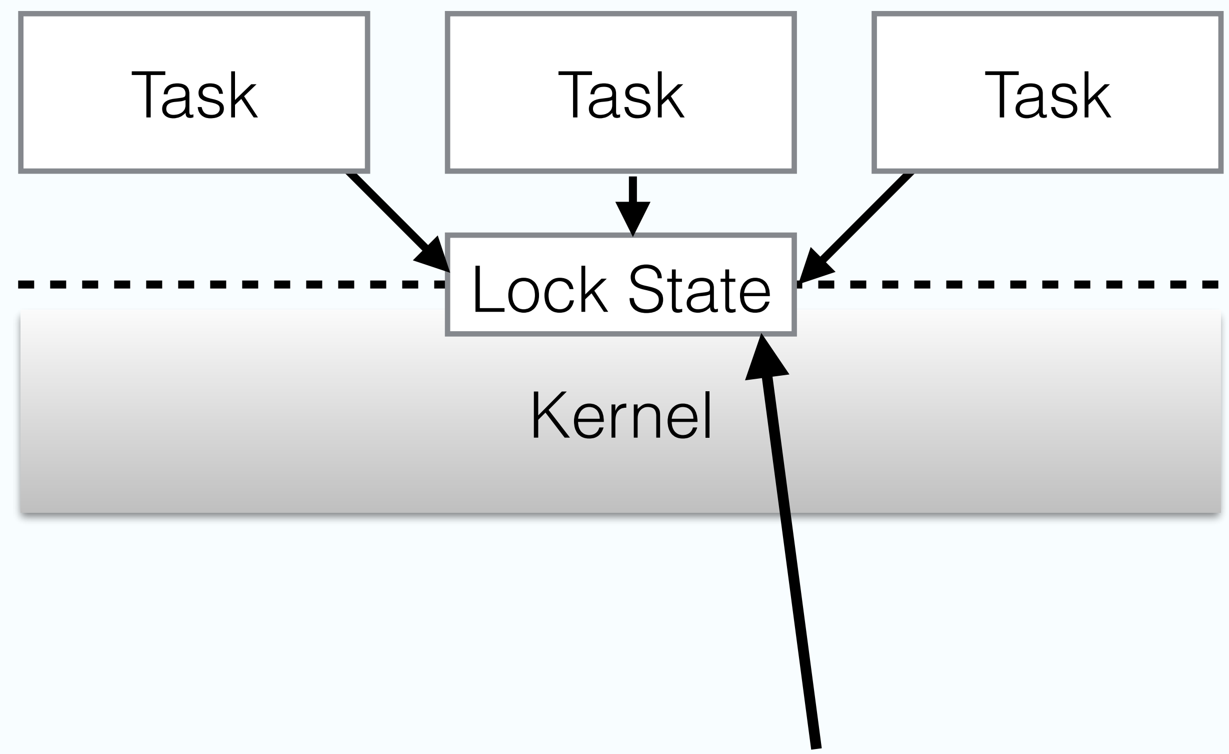


Mutex API

```
kernel_do_lock();
// critical section
kernel_do_unlock();
```

Futexes — Overview and Intuition

Exporting Lock State to Userspace Processes



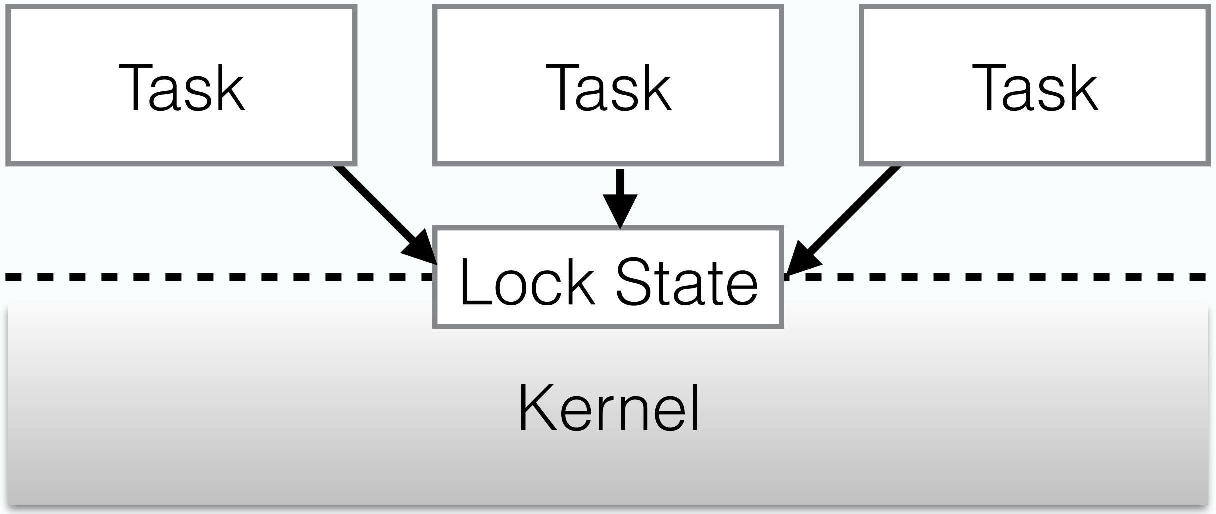
Shared lock state between userspace and kernel

Futexes — Overview and Intuition

Exporting Lock State to Userspace Processes

Futex Pseudocode

```
try_to_acquire_lock(lock);
if (lock is contended)
    kernel_do_lock(lock);
// critical section
release_lock(lock);
if (there are blocked tasks)
    kernel_do_unlock();
```

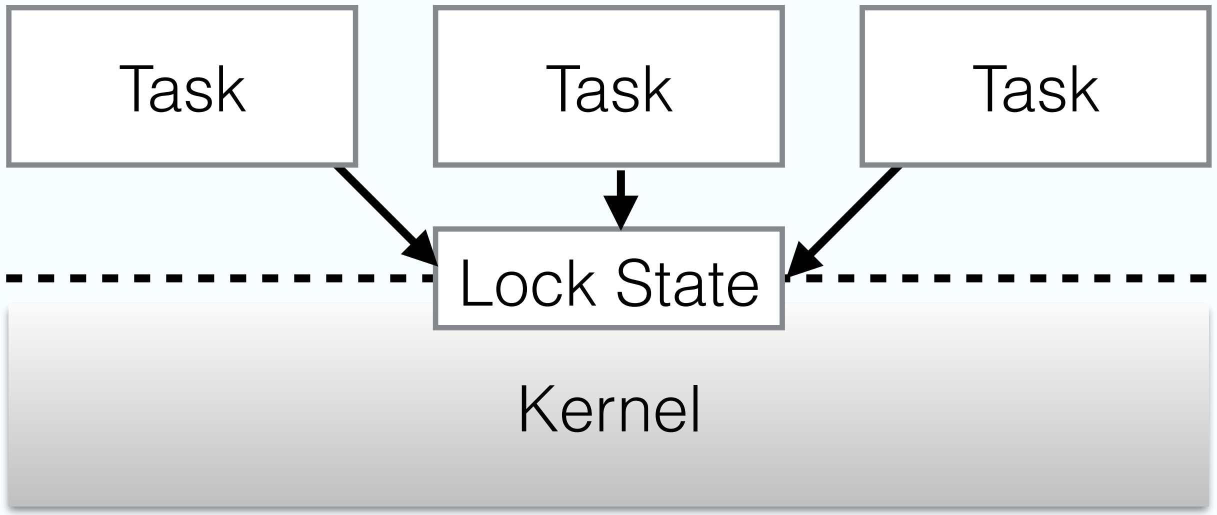


Futexes — Overview and Intuition

Fast-Path for Uncontended Operations

Futex Pseudocode

```
try_to_acquire_lock(lock);
if (lock is contended)
    kernel_do_lock(lock);
// critical section
release_lock(lock);
if (there are blocked tasks)
    kernel_do_unlock();
```



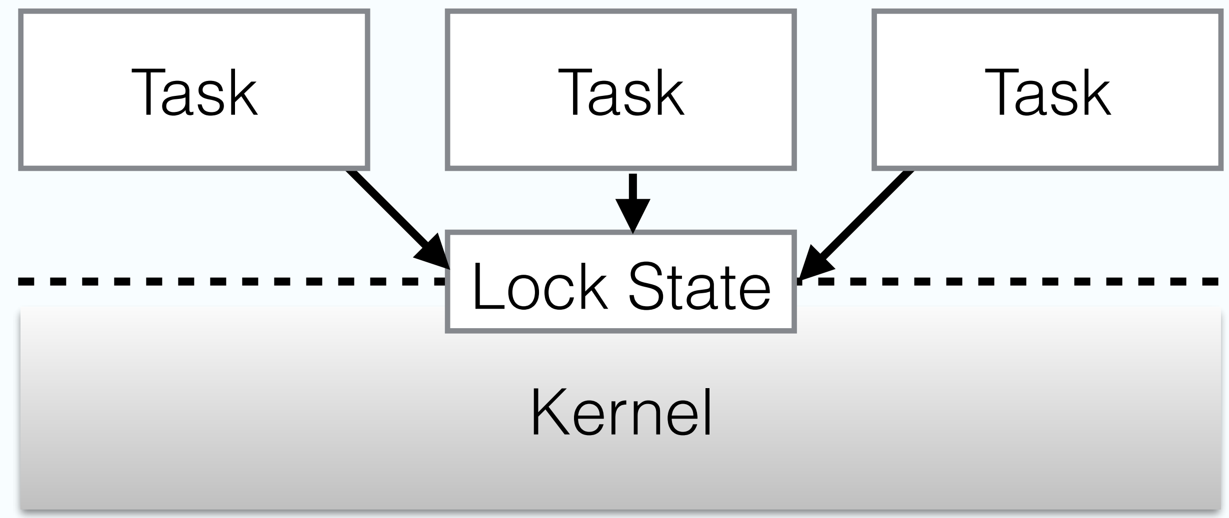
Fast-path when operation is uncontended (kernel not invoked)

Futexes — Overview and Intuition

Slow-Path for Contended Operations

Futex Pseudocode

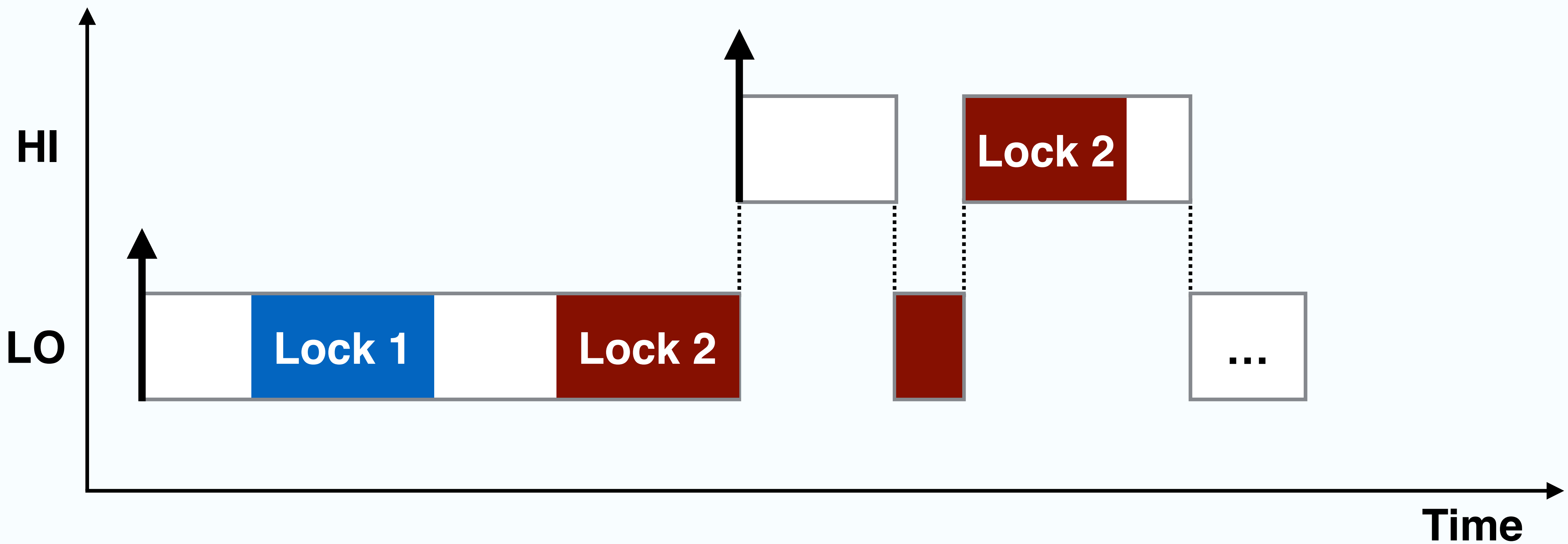
```
try_to_acquire_lock(lock);
if (lock is contended)
    kernel_do_lock(lock);
// critical section
release_lock(lock);
if (there are blocked tasks)
    kernel_do_unlock();
```



Kernel invoked **only when lock or unlock operation is contended**

Priority Inheritance Protocol

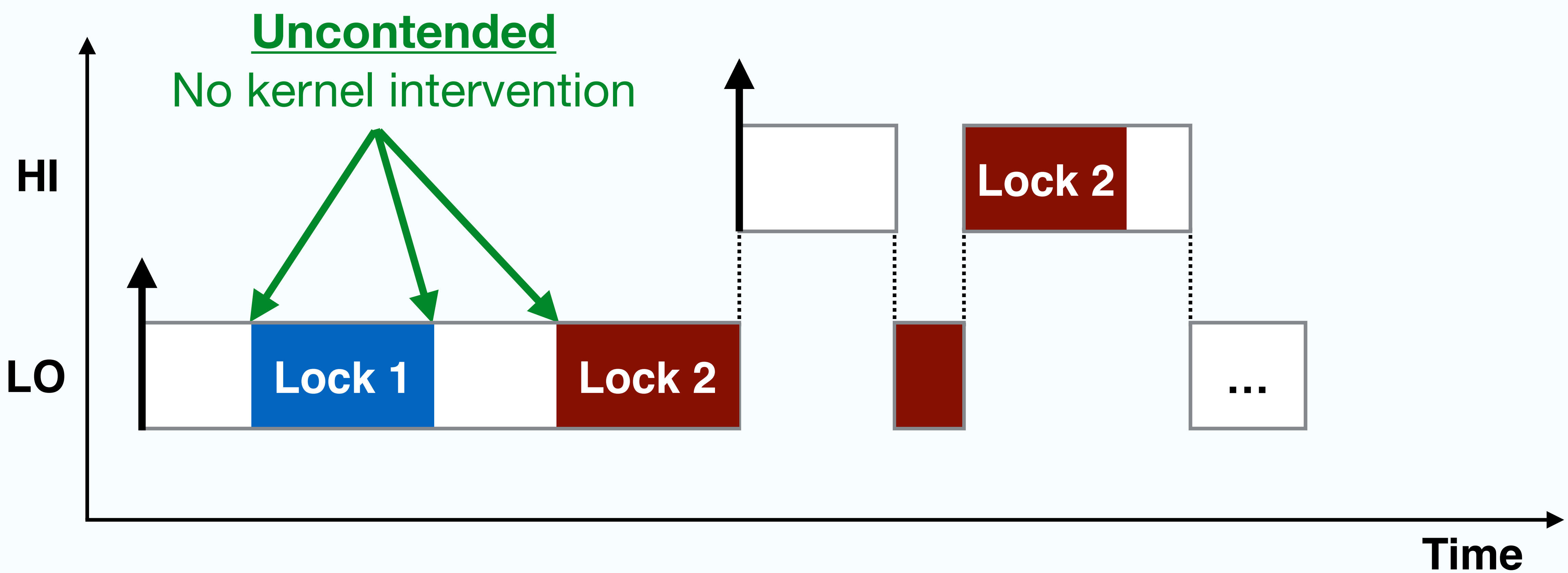
Futexes for reactive locking protocols



Fixed-Priority Scheduling

Priority Inheritance Protocol

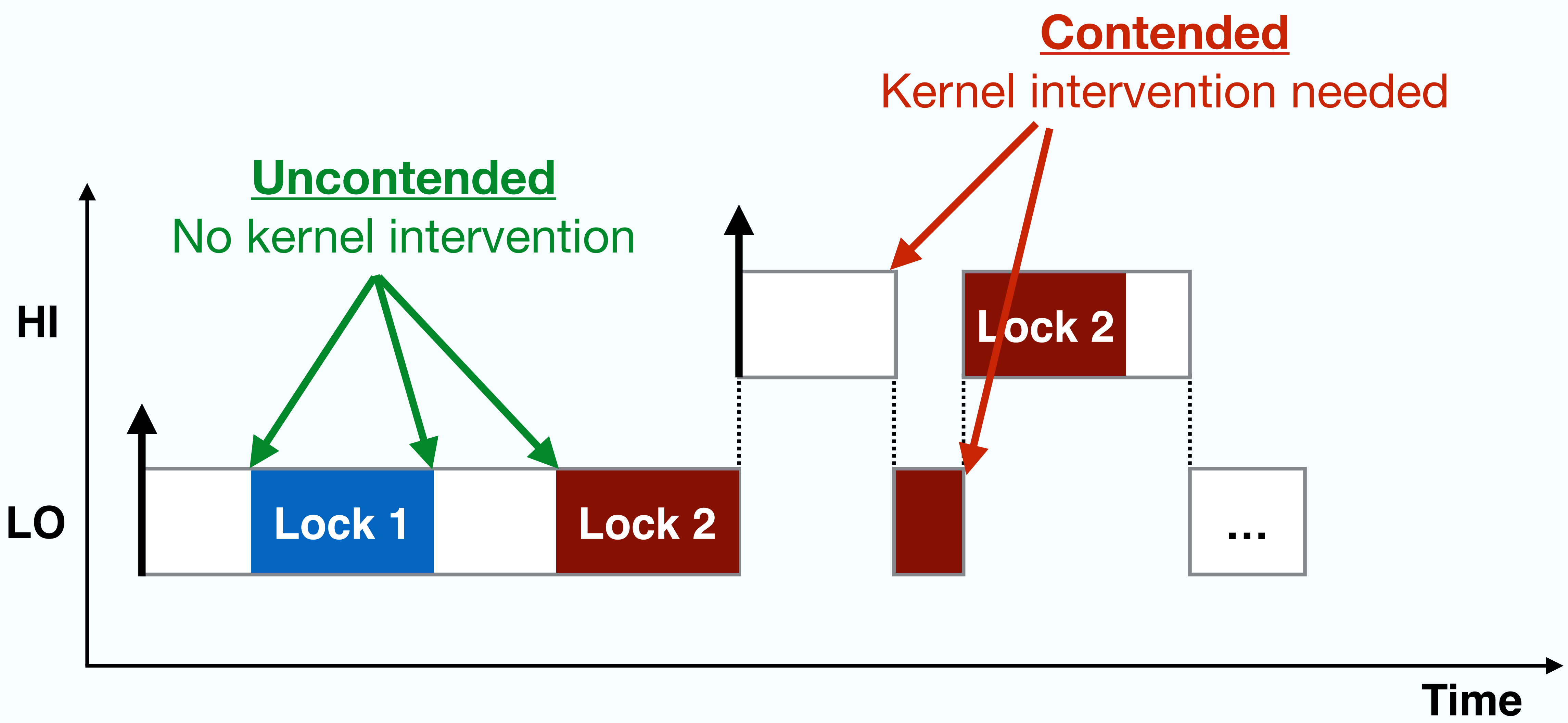
Futexes for reactive locking protocols



Fixed-Priority Scheduling

Priority Inheritance Protocol

Futexes for reactive locking protocols



Fixed-Priority Scheduling

Priority Ceiling Protocol (PCP)

- Each lock has a **priority ceiling** — the highest priority of all tasks that can acquire that lock.

Priority Ceiling Protocol (PCP)

- Each lock has a **priority ceiling** — the highest priority of all tasks that can acquire that lock.
- **System ceiling** — Currently held lock with highest ceiling

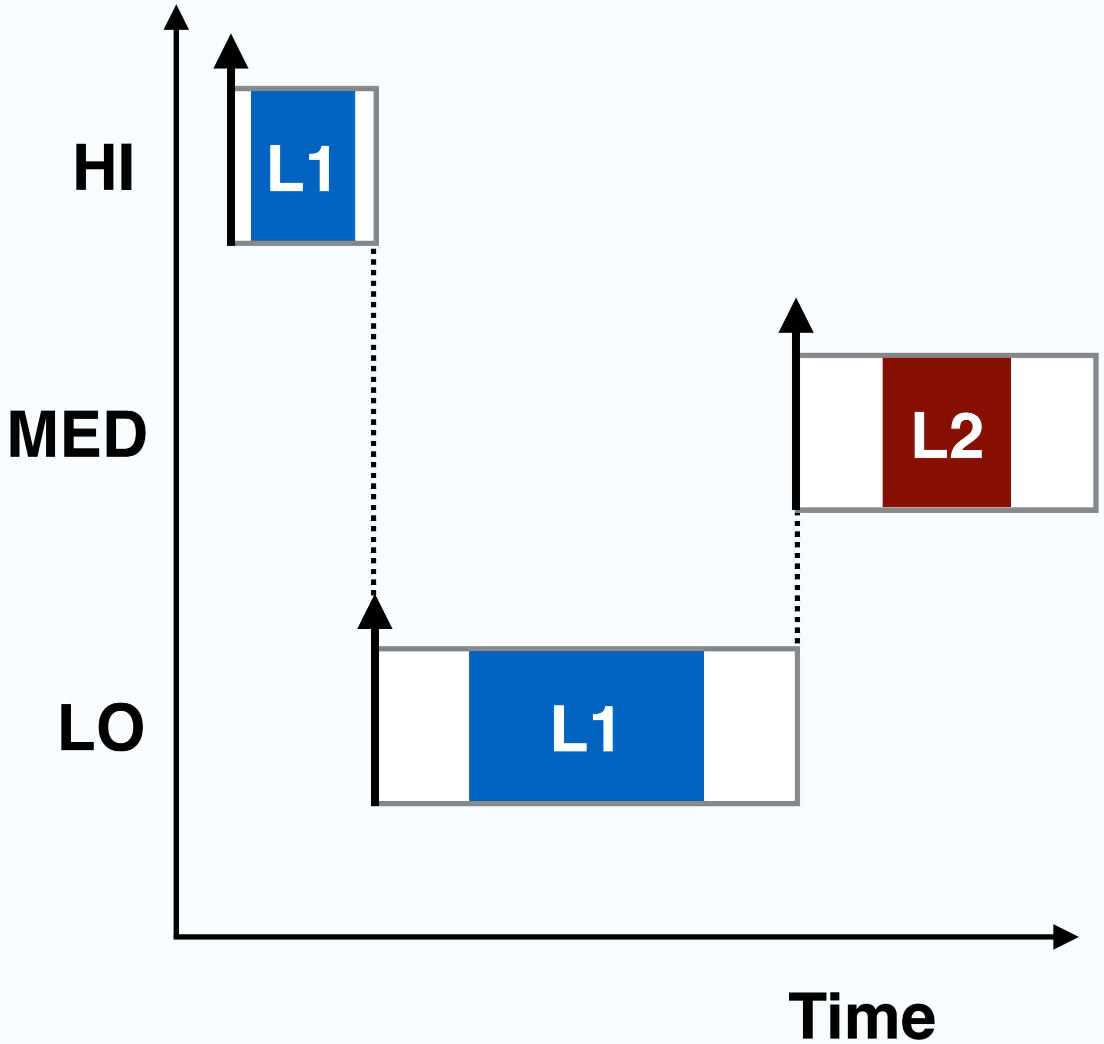
Priority Ceiling Protocol (PCP)

- Each lock has a **priority ceiling** — the highest priority of all tasks that can acquire that lock.
- **System ceiling** — Currently held lock with highest ceiling
- A lock may be acquired by a task only if:
    - **Task priority > System Ceiling Priority**
    - Or the task is **holding the system ceiling**

Priority Ceiling Protocol (PCP)

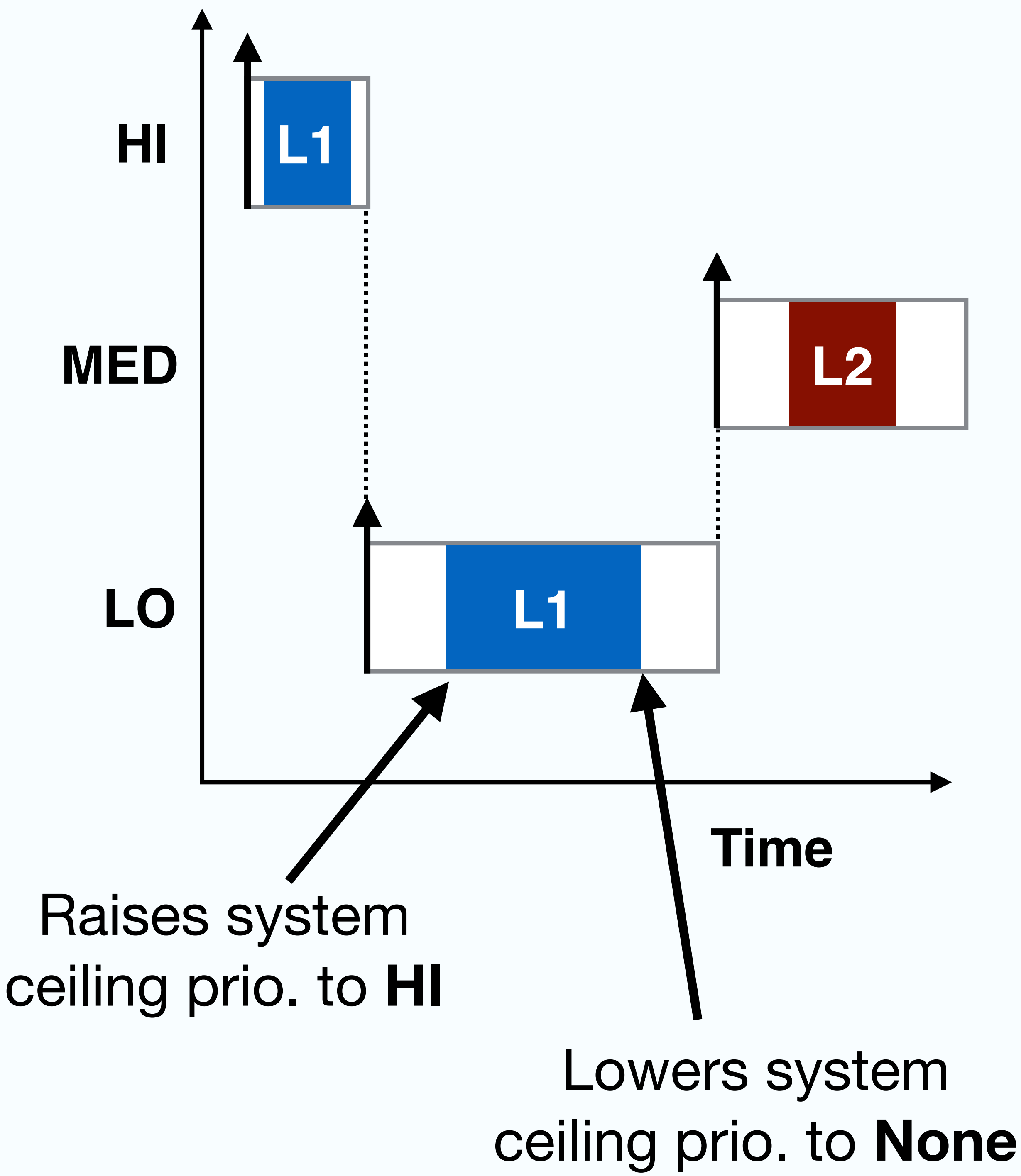
- Each lock has a **priority ceiling** — the highest priority of all tasks that can acquire that lock.
- **System ceiling** — Currently held lock with highest ceiling
- A lock may be acquired by a task only if:
    - **Task priority > System Ceiling Priority**
    - Or the task is **holding the system ceiling**
- When **under contention, priority inheritance** (raises priority of lower priority task to that of higher priority task).

PCP Futexes — Challenge



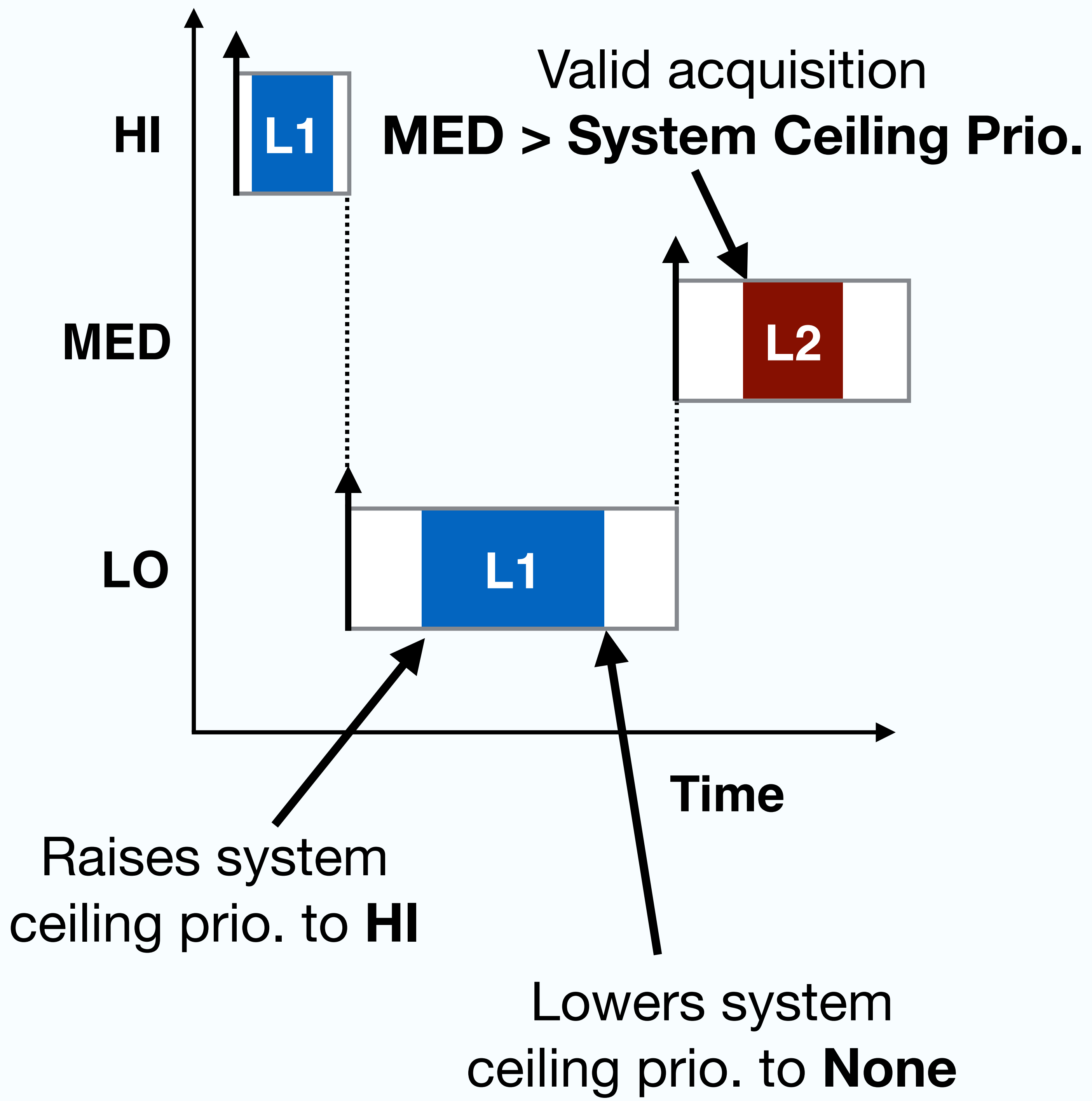
Fixed-Priority Scheduling

PCP Futexes — Challenge

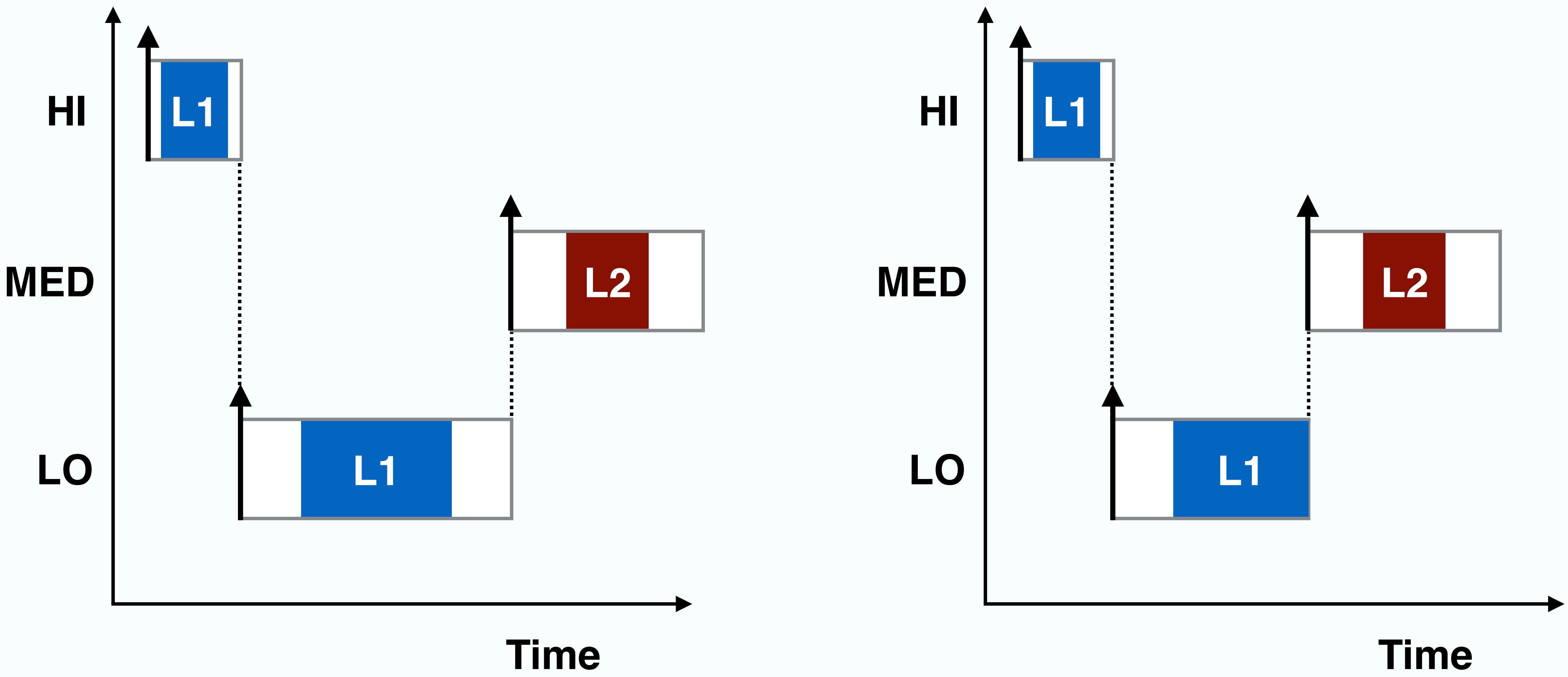


Fixed-Priority Scheduling

PCP Futexes — Challenge

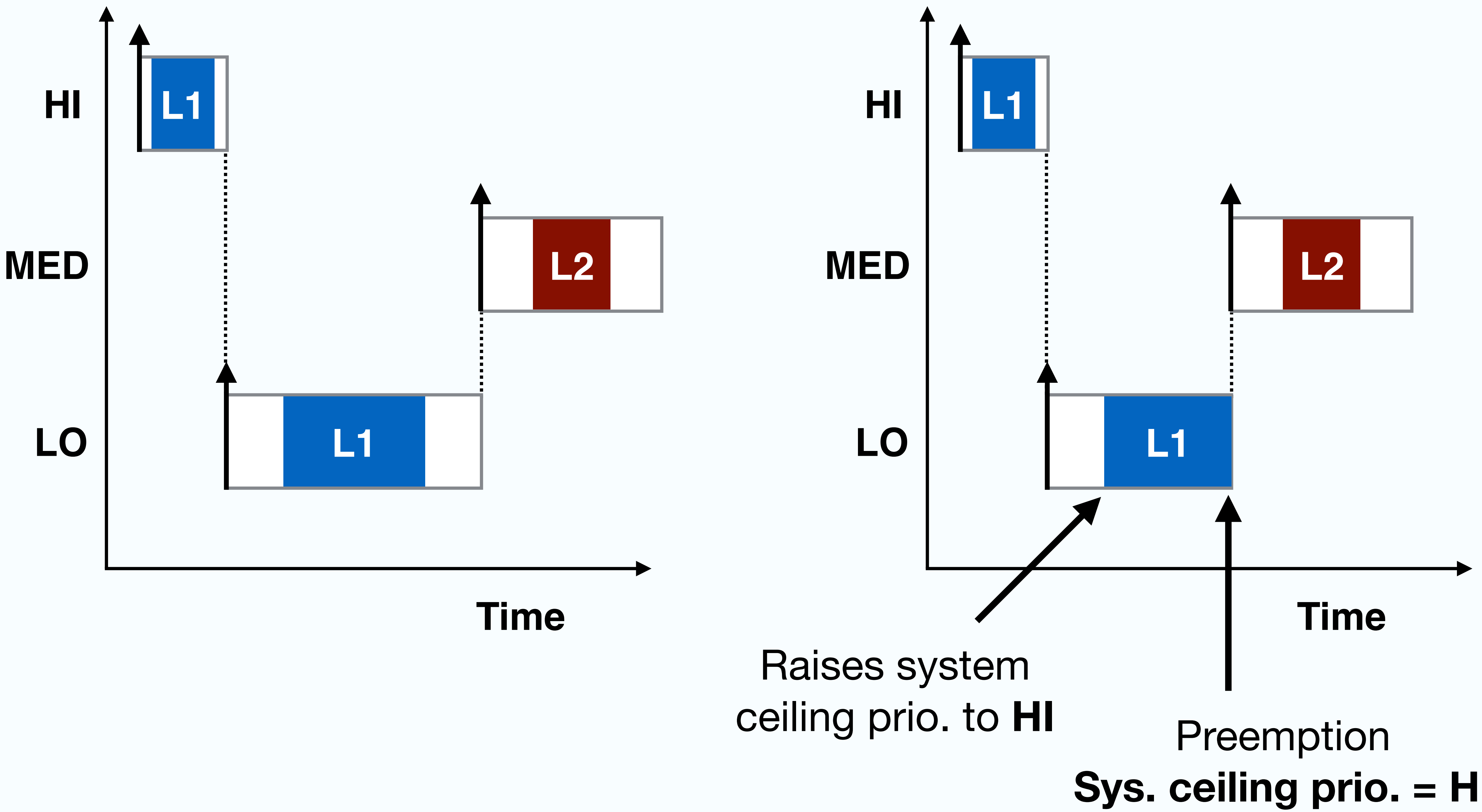


PCP Futexes — Challenge



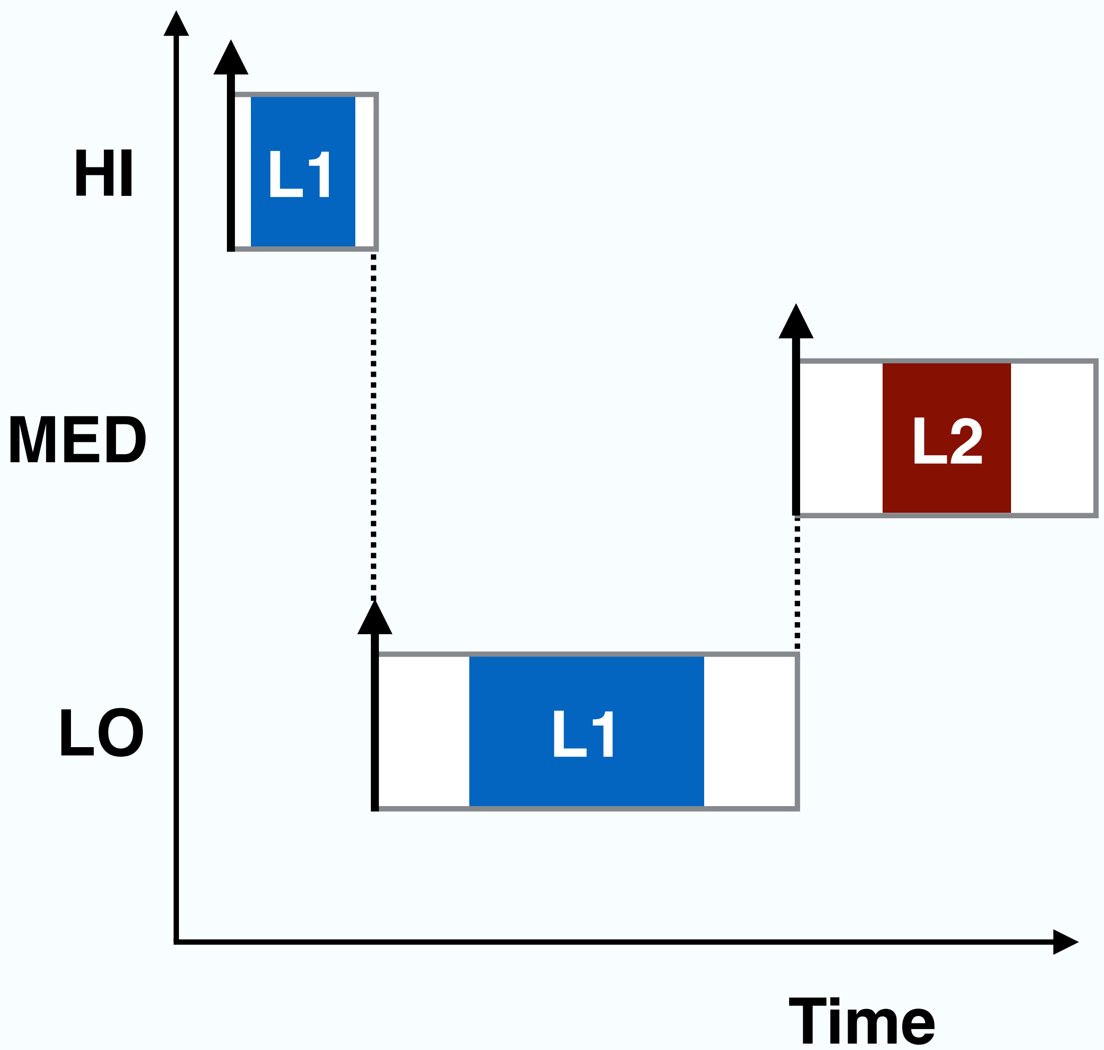
Fixed-Priority Scheduling

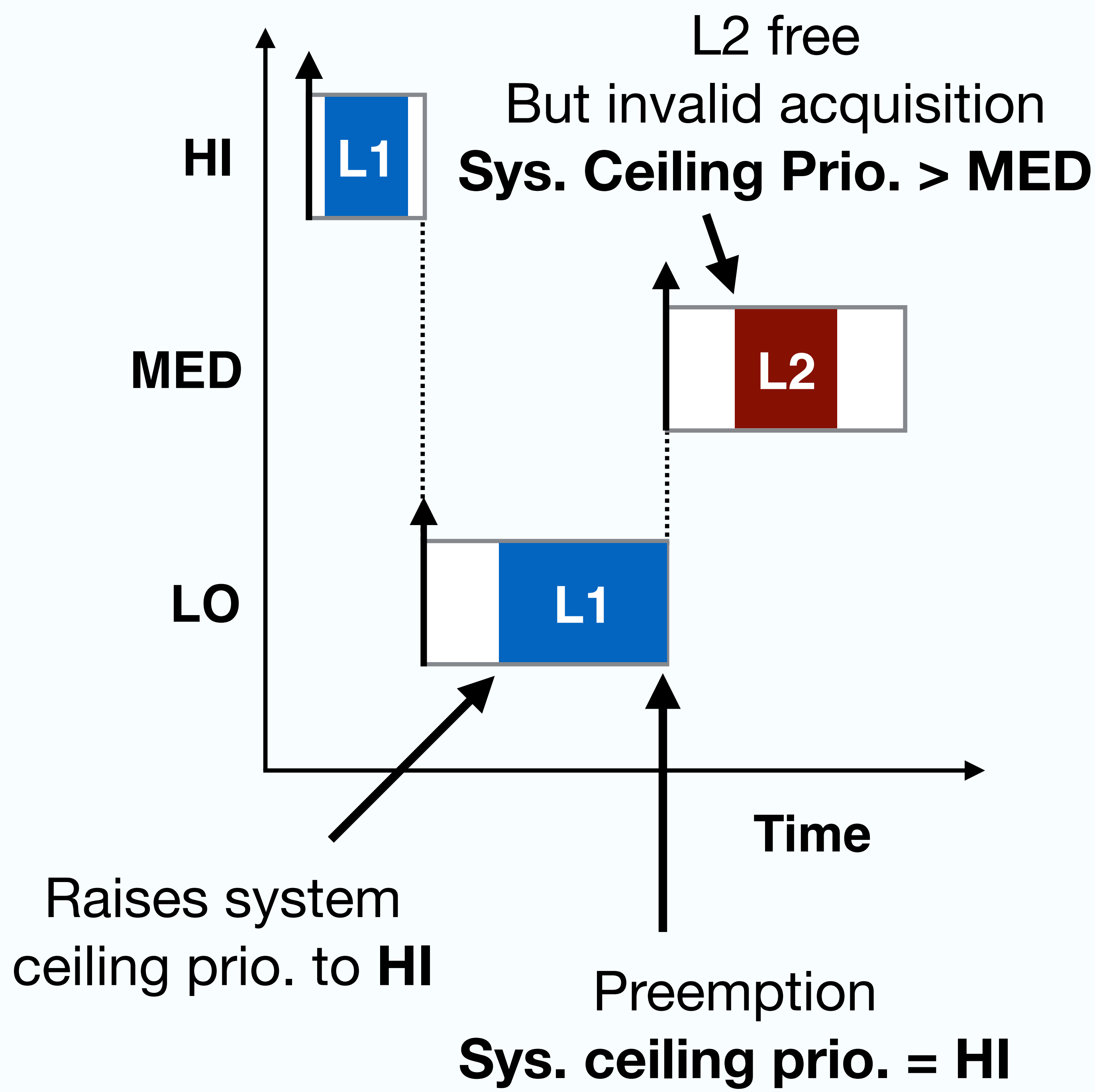
PCP Futexes — Challenge



Fixed-Priority Scheduling

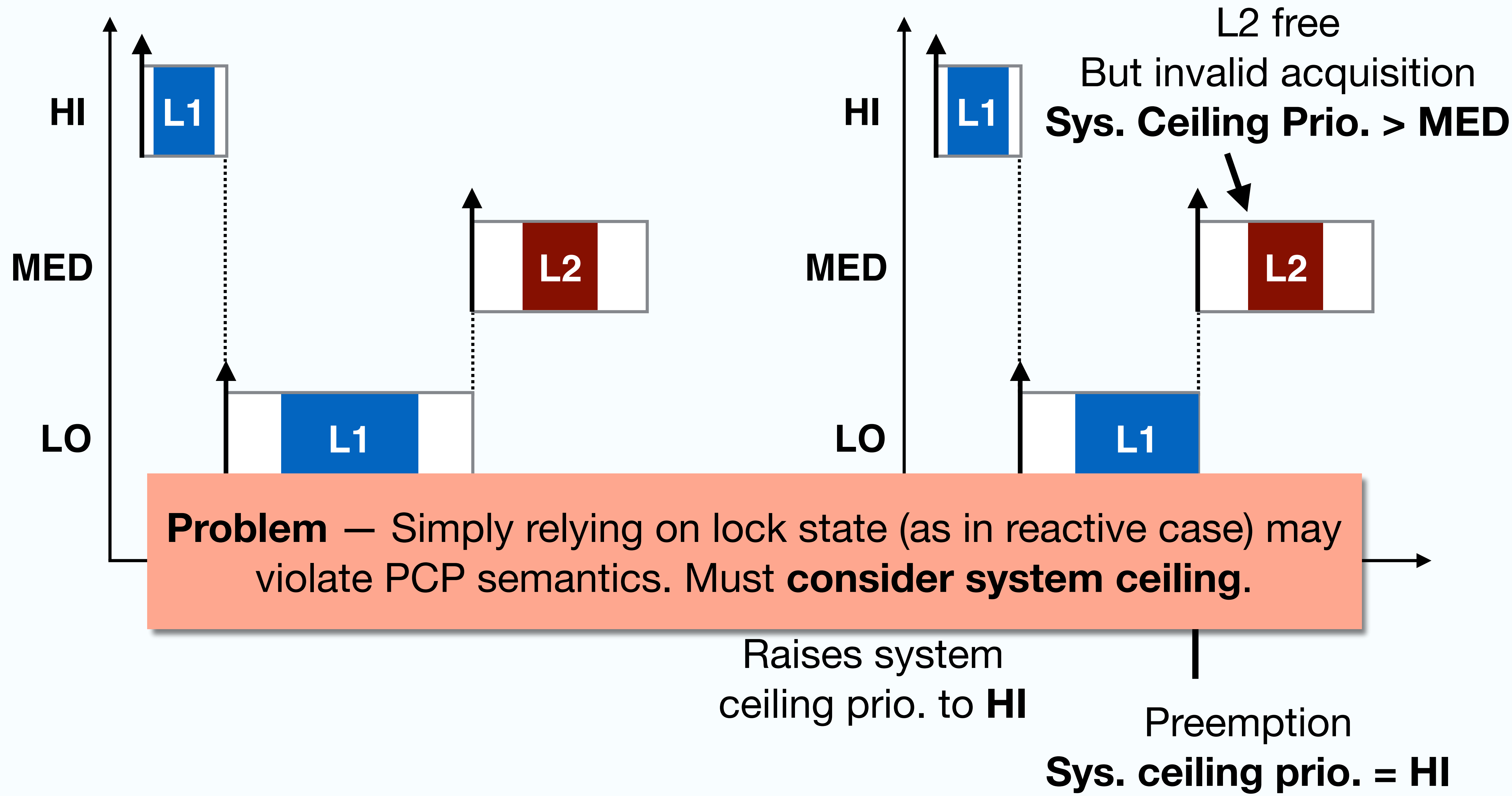
PCP Futexes — Challenge





Fixed-Priority Scheduling

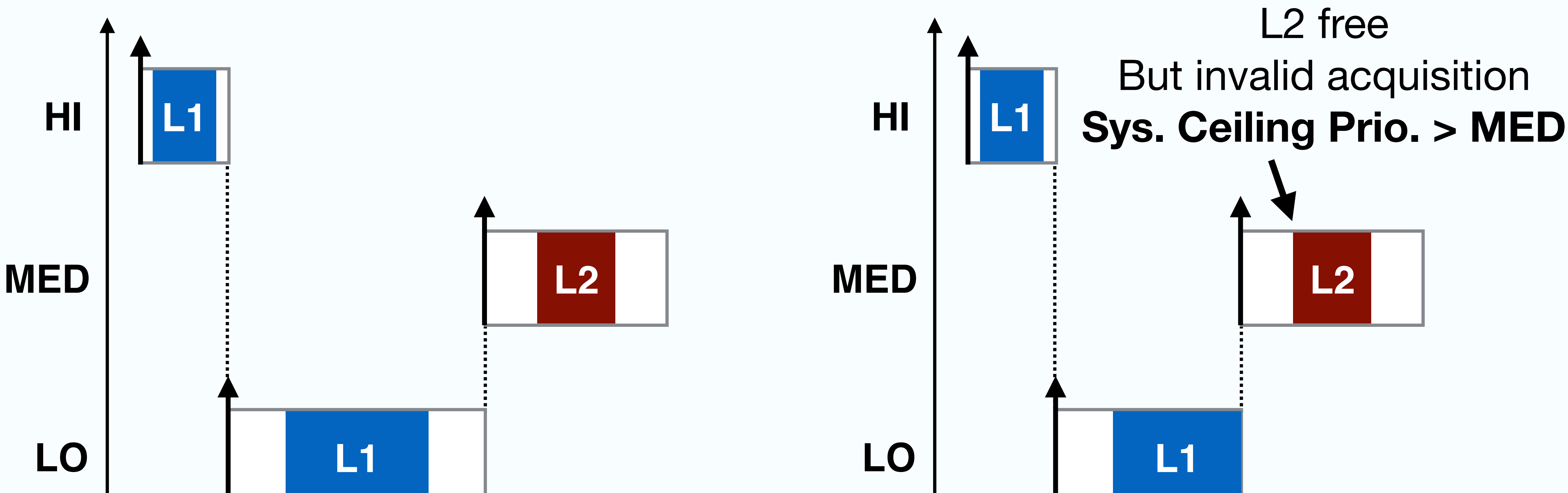
PCP Futexes — Challenge



Problem — Simply relying on lock state (as in reactive case) may violate PCP semantics. Must **consider system ceiling.**

Fixed-Priority Scheduling

PCP Futexes — Challenge



Problem — Simply relying on lock state (as in reactive case) may violate PCP semantics. Must **consider system ceiling**.

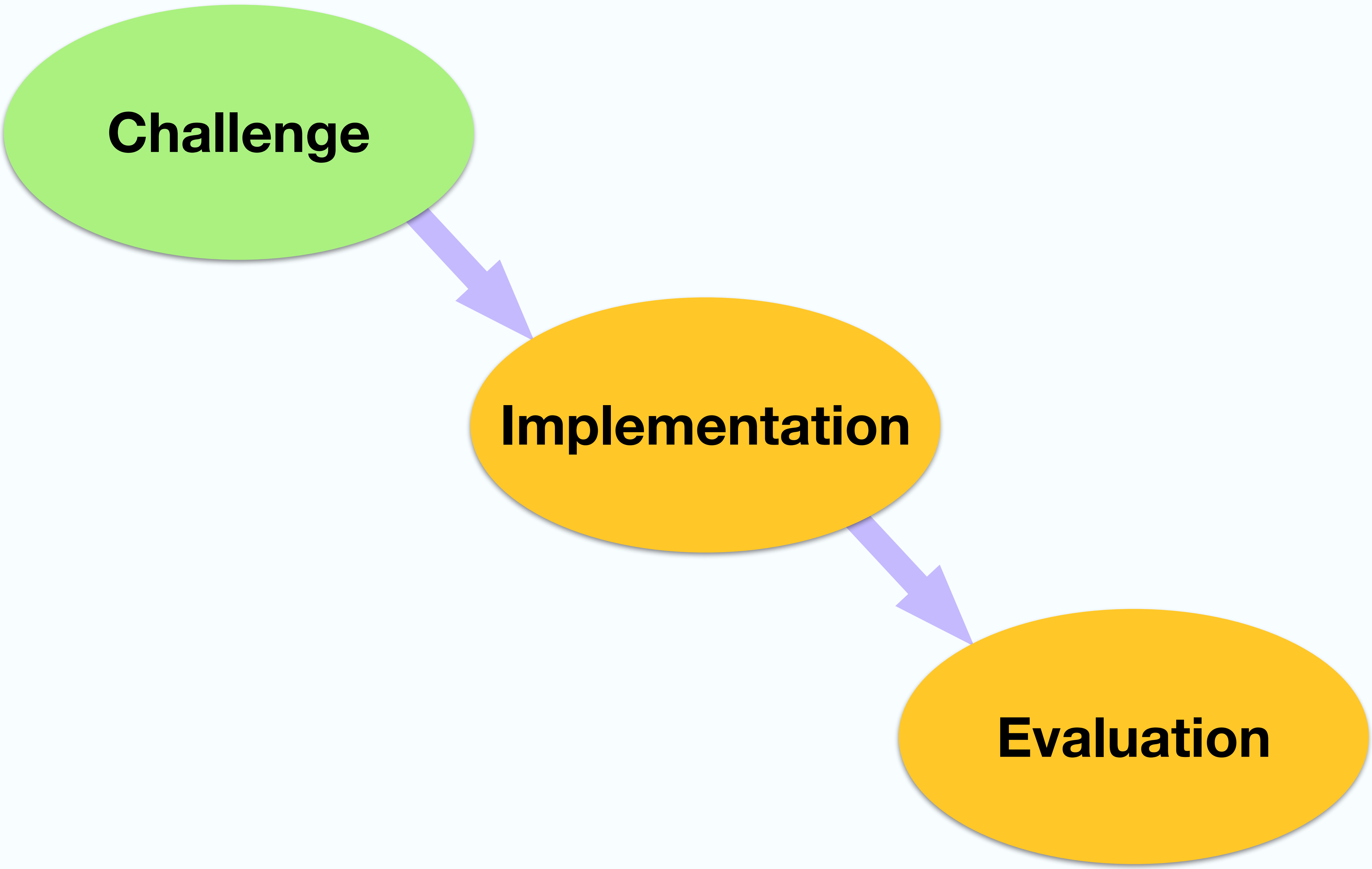
Raises system

Problem — system ceiling potentially updated at every lock acquisition and release — must **always invoke kernel** to avoid protocol violations

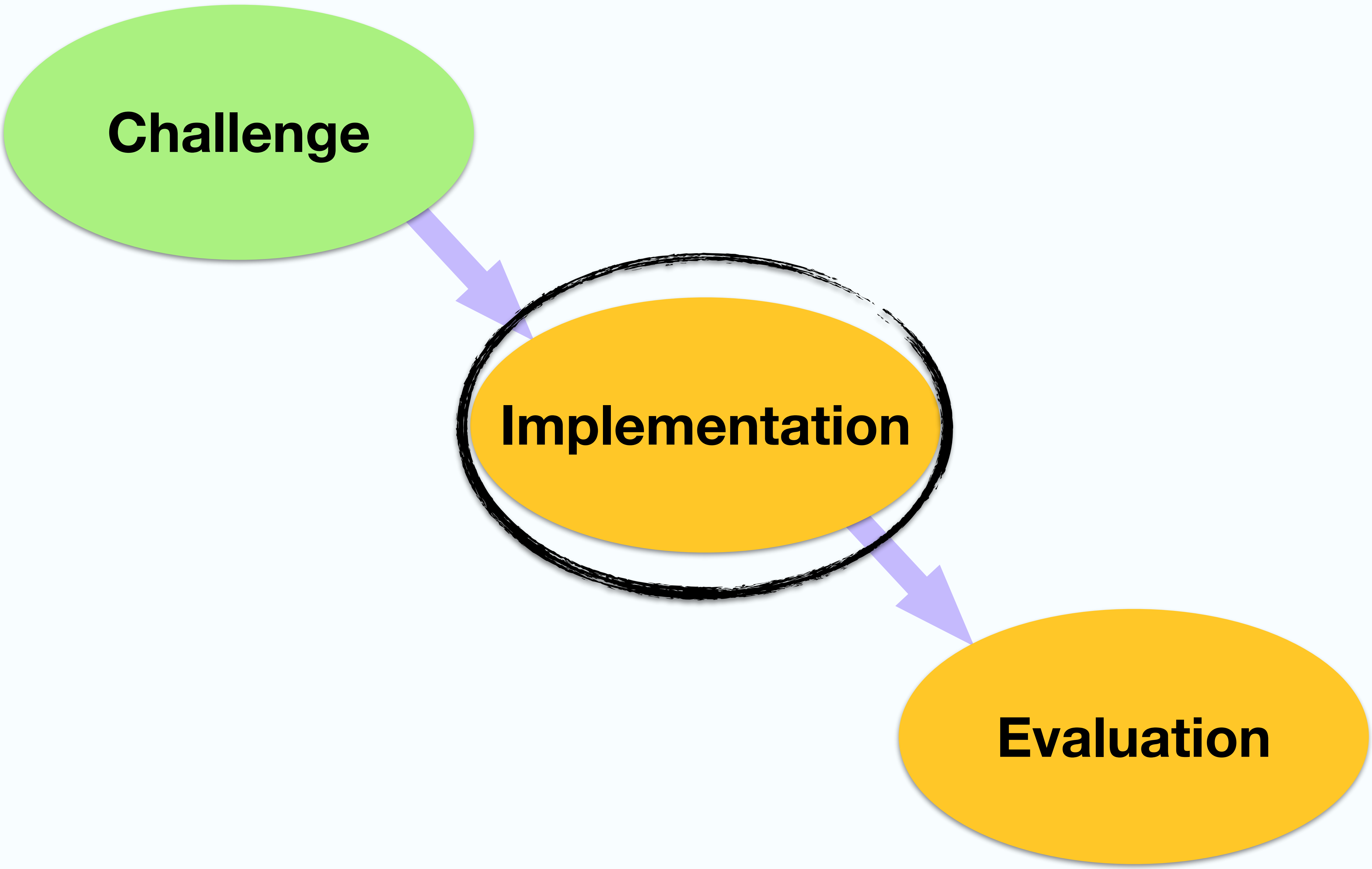
= **HI**

Fixed-Priority Scheduling

Overview of Talk



Overview of Talk



PCP Futexes — Our Approach

PCP Futex Pseudocode

```
if (lock is contended)
    kernel_do_lock(lock);
else
   acquire_lock_locally(lock)
// critical section
release_lock(lock);
if (there are blocked tasks)
    kernel_do_unlock();
```

PCP Futexes — Our Approach

PCP Futex Pseudocode

```
if (lock is contended)
    kernel_do_lock(lock);
else
   acquire_lock_locally(lock)
// critical section
release_lock(lock);
if (there are blocked tasks)
    kernel_do_unlock();
```

Whether lock acquisition is contended is determined in PCP by the **current system ceiling**

PCP Futexes — Our Approach

PCP Futex Pseudocode

```
if (lock is contended)
    kernel_do_lock(lock);
else
   acquire_lock_locally(lock)
// critical section
release_lock(lock);
if (there are blocked tasks)
    kernel_do_unlock();
```

Whether lock acquisition is contended is determined in PCP by the **current system ceiling**

The presence of blocked tasks is determined by the **size of the wait-queue** corresponding to the lock.

PCP Futexes — Our Approach

PCP Futex Pseudocode

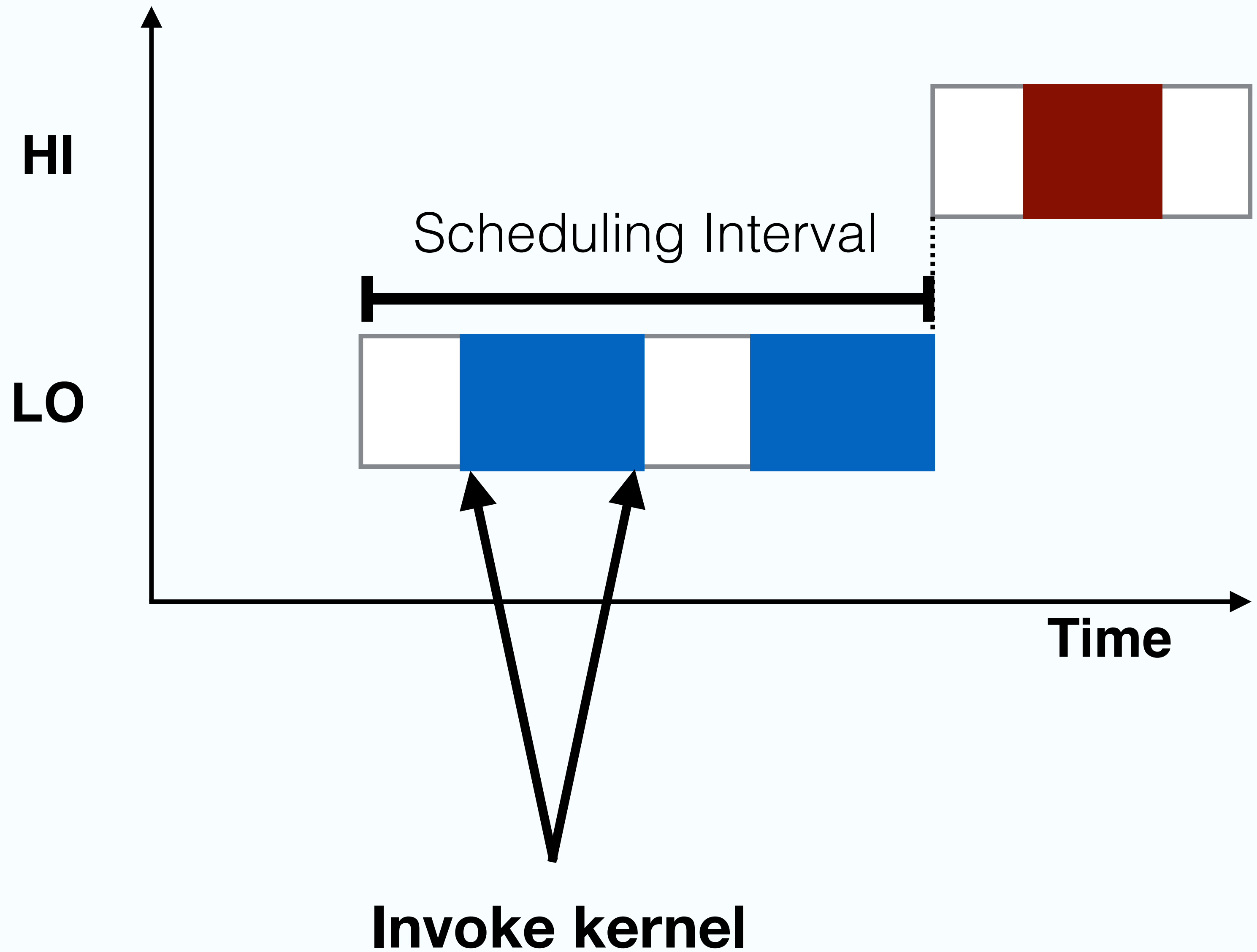
```
if (lock is contended)
    kernel_do_lock(lock);
else
   acquire_lock_locally(lock)
// critical section
release_lock(lock);
if (there are blocked tasks)
    kernel_do_unlock();
```

Whether lock acquisition is contended is determined in PCP by the **current system ceiling**

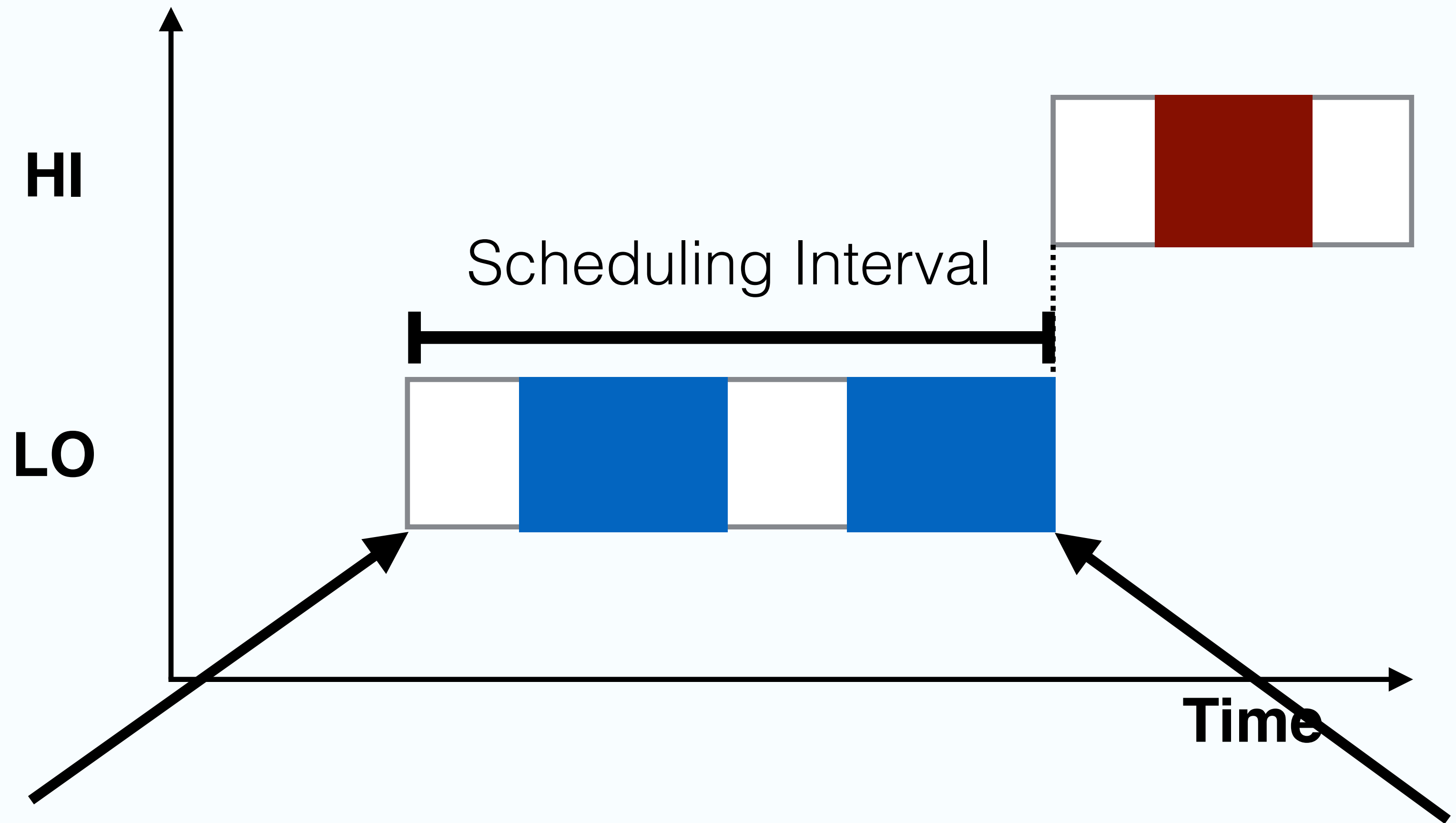
The presence of blocked tasks is determined by the **size of the wait-queue** corresponding to the lock.

Cannot change **while a task is scheduled** on a uniprocessor!

PCP Futexes — Deferred Updates



PCP Futexes — Deferred Updates

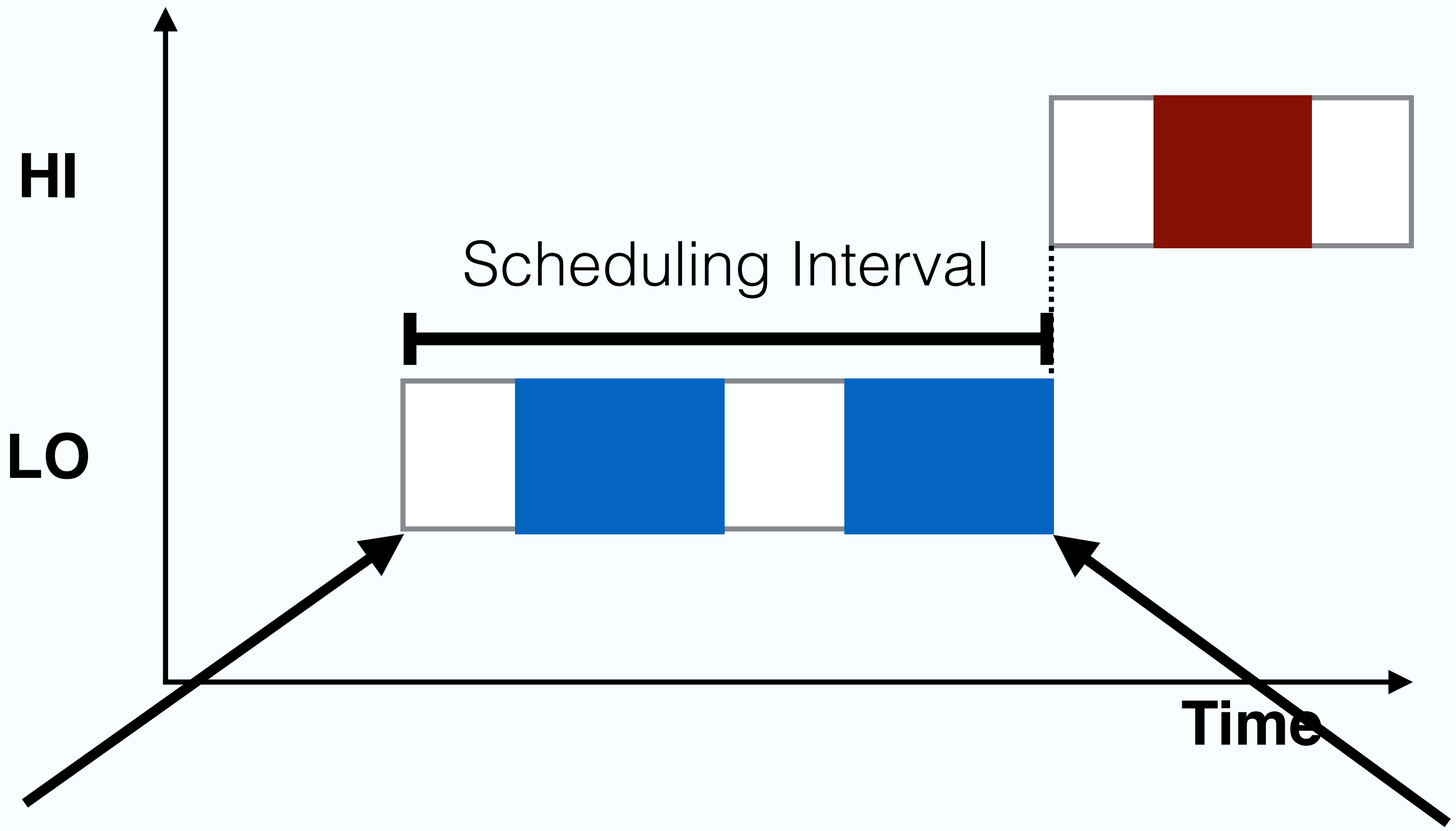


Predetermine contention conditions and communicate them to task

Make changes to **local copy of lock states** (if uncontended)

Update kernel state based on copy **at context switch**

PCP Futexes — Deferred Updates



Predetermine contention conditions and communicate them to task

Make changes to **local copy of lock states** (if uncontended)

Update kernel state based on copy **at context switch**

Requires a **bidirectional communication channel** between tasks and the kernel

Bidirectional Communication — Lock Page

bitmap: lock_states
bool: tasks_blocked

Lock Page

Per-process **page of memory** writable by both the process and kernel.

Bidirectional Communication — Lock Page

Lock-state bitmap used to communicate acquisitions and releases of locks to kernel

bitmap: lock_states
bool: tasks_blocked

Lock Page

Per-process **page of memory** writable by both the process and kernel.

Bidirectional Communication — Lock Page

Lock-state bitmap used to communicate acquisitions and releases of locks to kernel

bitmap: lock_states
bool: tasks_blocked

Boolean set by kernel to indicate the **presence of blocked tasks** (indicates that release operation is contended)

Lock Page

Per-process **page of memory** writable by both the process and kernel.

Bidirectional Communication — Lock Page

Lock-state bitmap used to communicate acquisitions and releases of locks to kernel

bitmap: lock_states
bool: tasks_blocked

Boolean set by kernel to indicate the **presence of blocked tasks** (indicates that release operation is contended)

Lock Page

Per-process **page of memory** writable by both the process and kernel.

A “permission bit” set by the kernel to indicate **whether lock acquisition is allowed** (system ceiling check)

Communicating the Permission Bit

PCP Futex Pseudocode

```
if (permission bit is set)
    set_bit_in_bitmap(lock)
else
    kernel_do_lock(lock);
// critical section
release_lock(lock);
if (tasks_blocked is set)
    kernel_do_unlock();
```

Communicating the Permission Bit

PCP Futex Pseudocode

```
if (permission bit is set)
    set_bit_in_bitmap(lock)
else
    kernel_do_lock(lock);
// critical section
release_lock(lock);
if (tasks_blocked is set)
    kernel_do_unlock();
```

Challenge:

Checking for permission and setting lock bit **must be atomic.**

Why? Preemption between them may change the permission.

Communicating the Permission Bit

We proposed two approaches for the Classic PCP

Page Faults (PCP-DU-PF)

Uses **page faults** to invoke kernel when locking is prohibited

CMPXCHG (PCP-DU-BOOL)

Boolean for permission bit.

Compare-and-exchange operation to check and acquire locks atomically

Communicating the Permission Bit

We proposed two approaches for the Classic PCP

Page Faults (PCP-DU-PF)

Uses **page faults** to invoke kernel when locking is prohibited

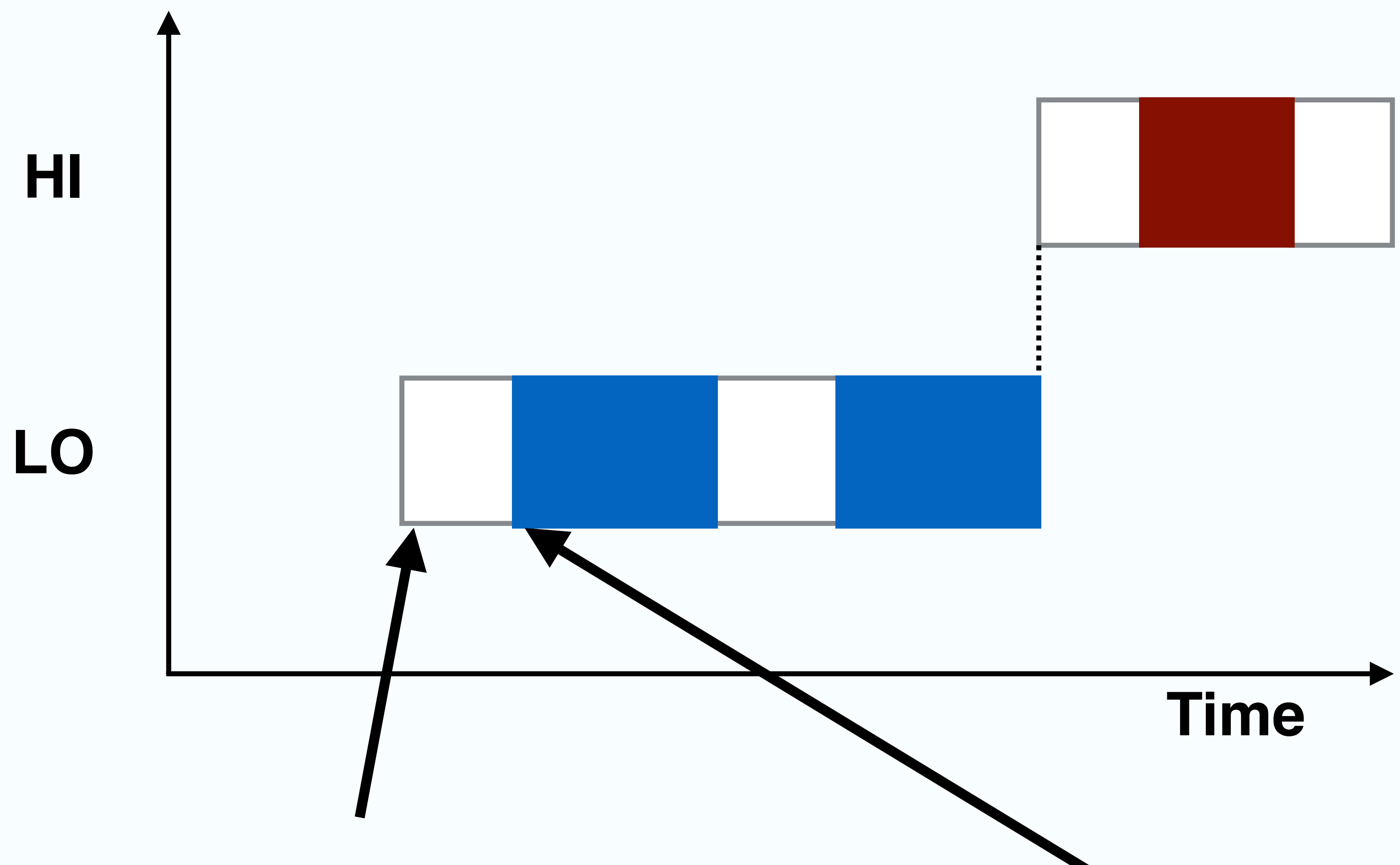
CMPXCHG (PCP-DU-BOOL)

See paper for details!

Boolean

Compare-and-exchange operation to check and acquire locks atomically

PCP Futexes — Page Fault Approach



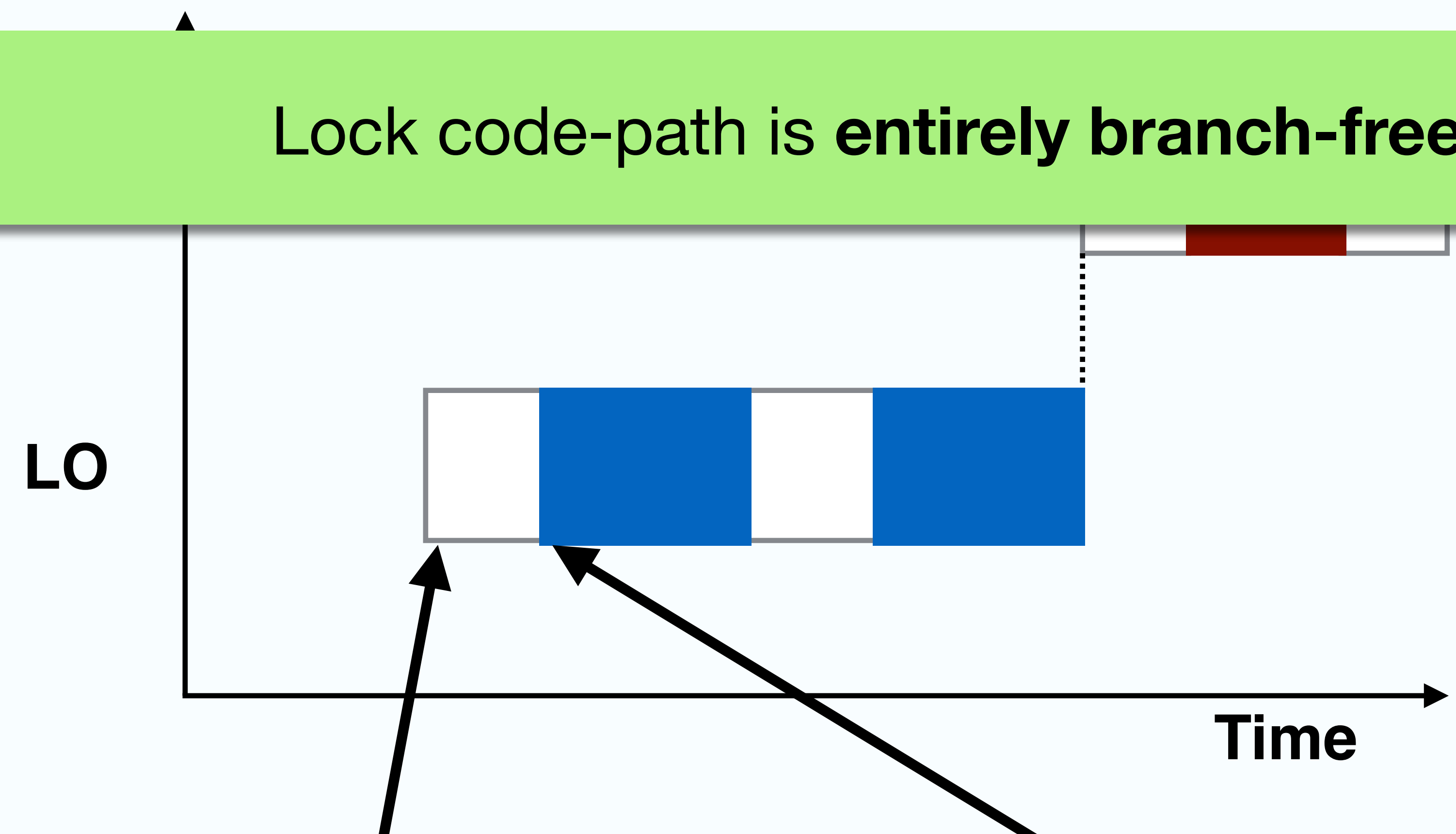
If (locking is allowed)
    set lock page writable
else
    set lock page unwritable

Attempting to write lock bit **will invoke kernel automatically** if locking is not permitted

PCP Futexes — Page Fault Approach

Can be implemented using **segmentation faults** as well

Lock code-path is **entirely branch-free**



If (locking is allowed)
    set lock page writable
else
    set lock page unwritable

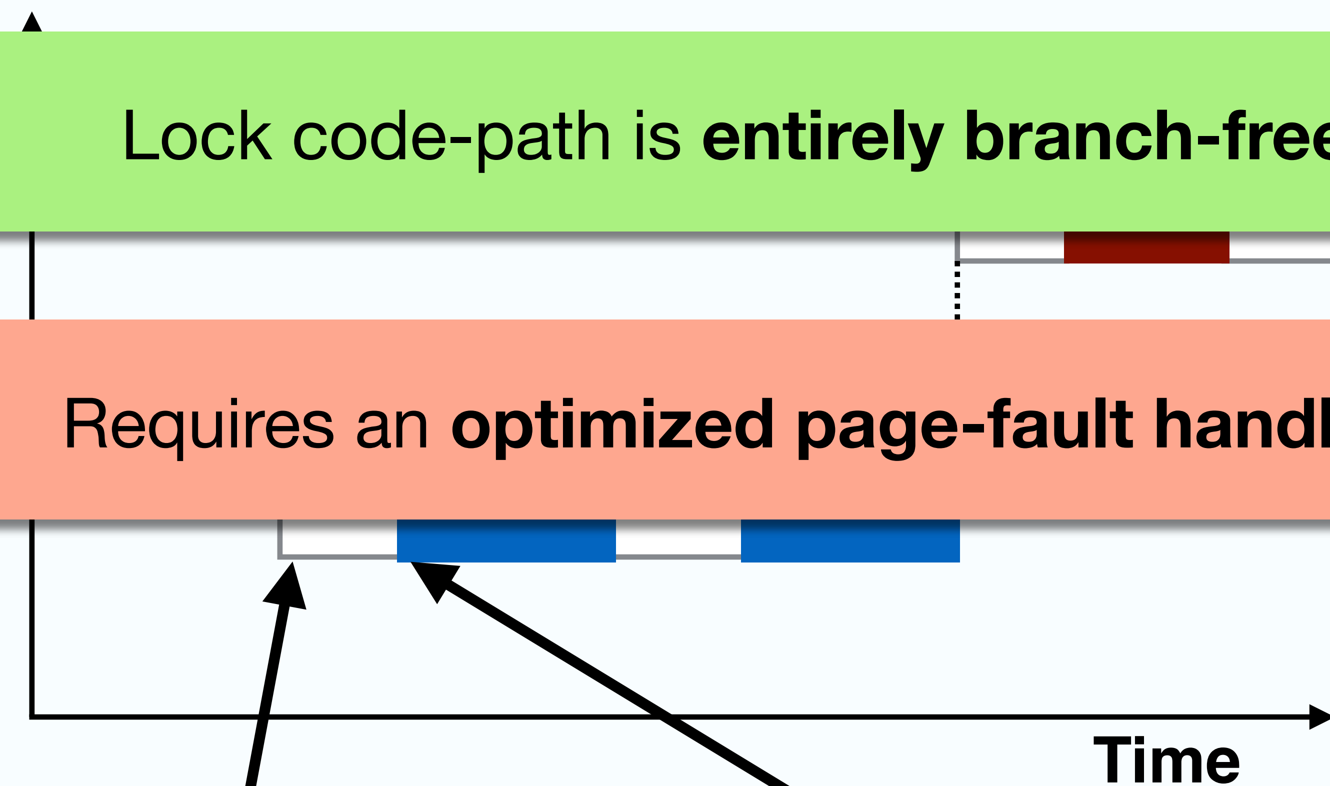
Attempting to write lock bit **will invoke kernel automatically** if locking is not permitted

PCP Futexes — Page Fault Approach

Can be implemented using **segmentation faults** as well

Lock code-path is **entirely branch-free**

Requires an **optimized page-fault handler**



If (locking is allowed)
    set lock page writable
else
    set lock page unwritable

Attempting to write lock bit **will invoke kernel automatically** if locking is not permitted

Communicating the Permission Bit

We proposed two approaches for the Classic PCP

Page Faults (PCP-DU-PF)

Uses **page faults** to invoke kernel when locking is prohibited

Lock code-path is entirely branch-free

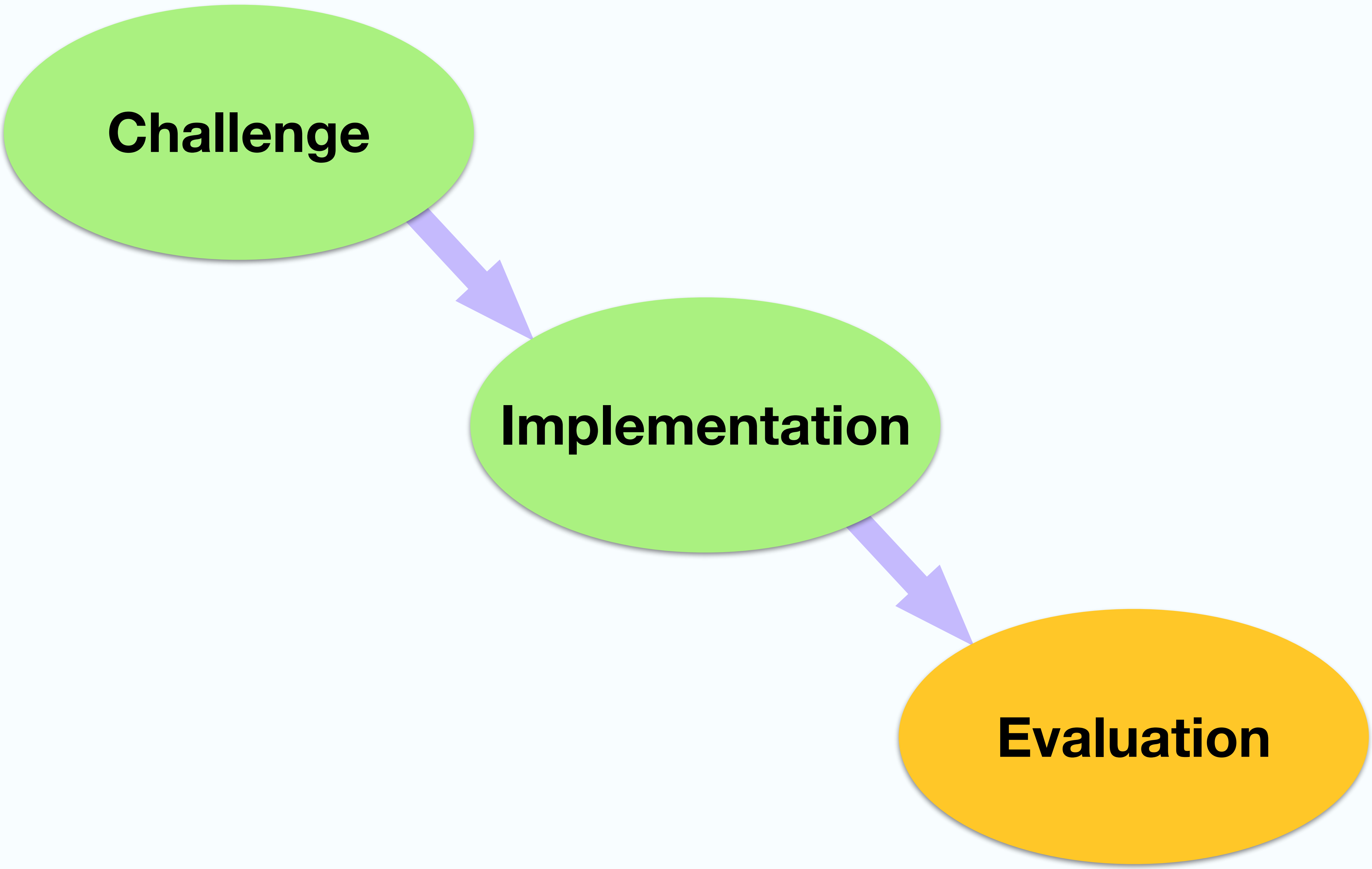
CMPXCHG (PCP-DU-BOOL)

Boolean for permission bit.

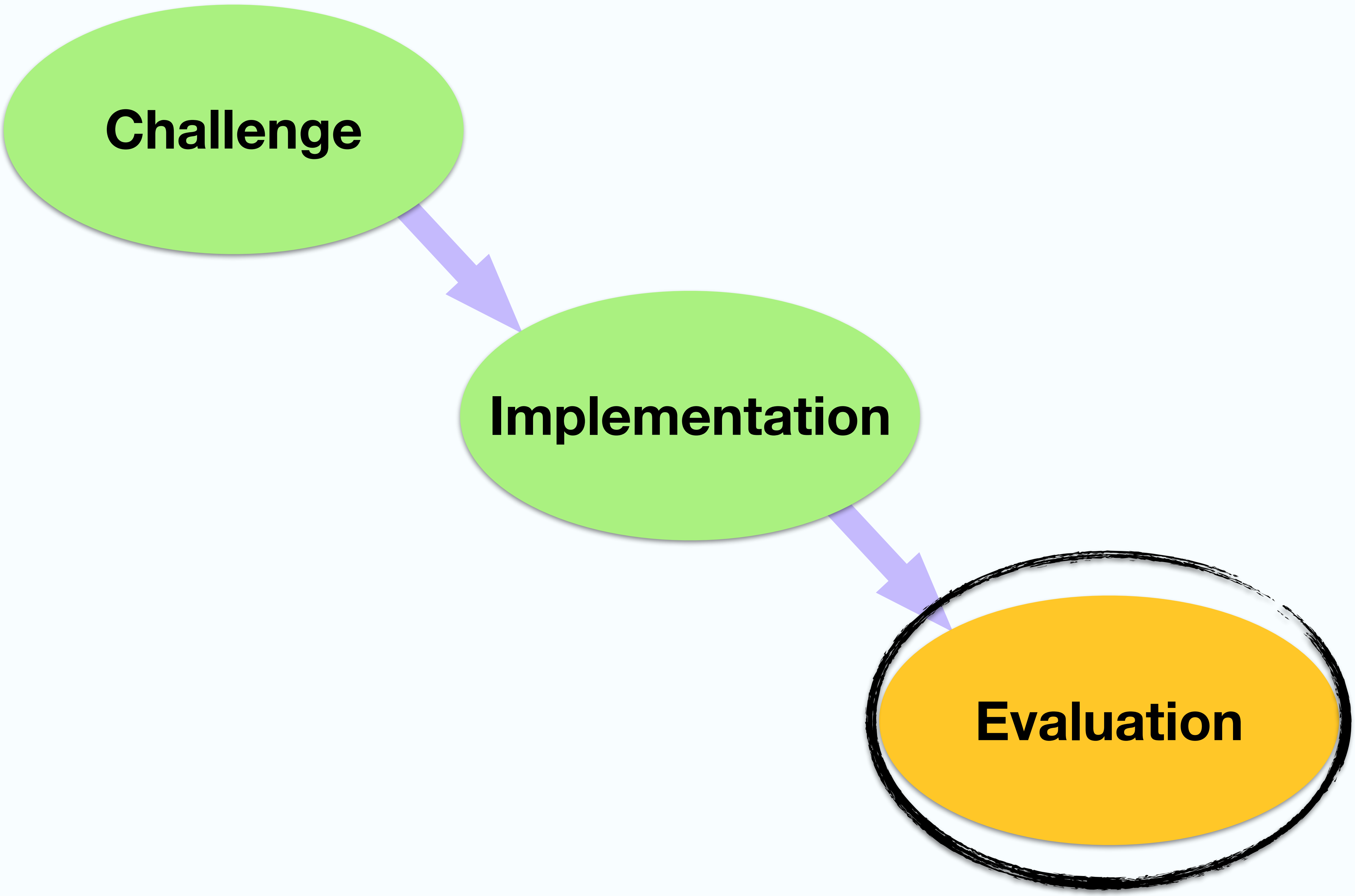
Compare-and-exchange operation to check and acquire locks atomically

Avoid page faults at cost of atomic op + branch

Overview of Talk



Overview of Talk



Evaluation — Platform

- Boundary Devices Sabre Lite Board
  - Freescale I.MX6Q Quad Core SoC
  - ARM Cortex A9 (1 GHz)
- Implemented in LITMUS^{RT} 2013.1 (based on Linux 3.10.5)

Evaluation — Benchmarking Methodology

- Test program **locks and unlocks once every period**
    - Uniprocessor tests: 5 threads
- **Randomly assigned parameters** for threads
    - *Critical section length — 25-45μs*
    - *Execution time — 25-65μs*
    - *Period — 600-800μs*
- Measured using processor **timestamp counters**

Evaluation — Benchmarking Methodology

- Test program **locks and unlocks once every period**
  - Uniprocessor tests: 5 threads
- **Randomly assigned parameters** for threads
  - *Critical section length — 25-45μs*
  - *Execution time — 25-65μs*
  - *Period — 600-800μs*

Demanding workload — thousands of operations per second

- Measured using processor **timestamp counters**

PCP Futex Evaluation — Baseline

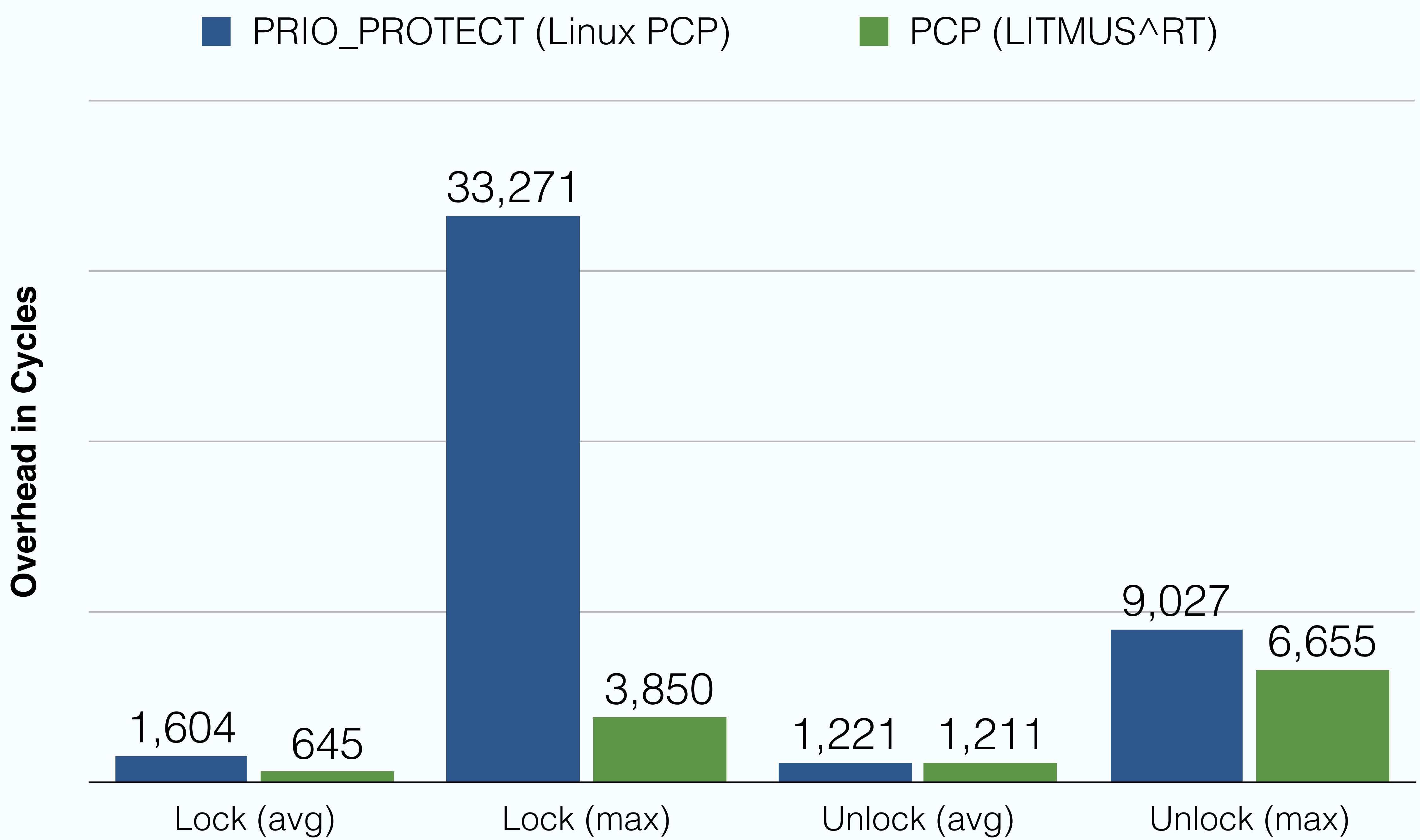
We compare all futex implementations against the ***non-futex PCP implementation*** *in LITMUS^{RT}*

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✗	✗	✗

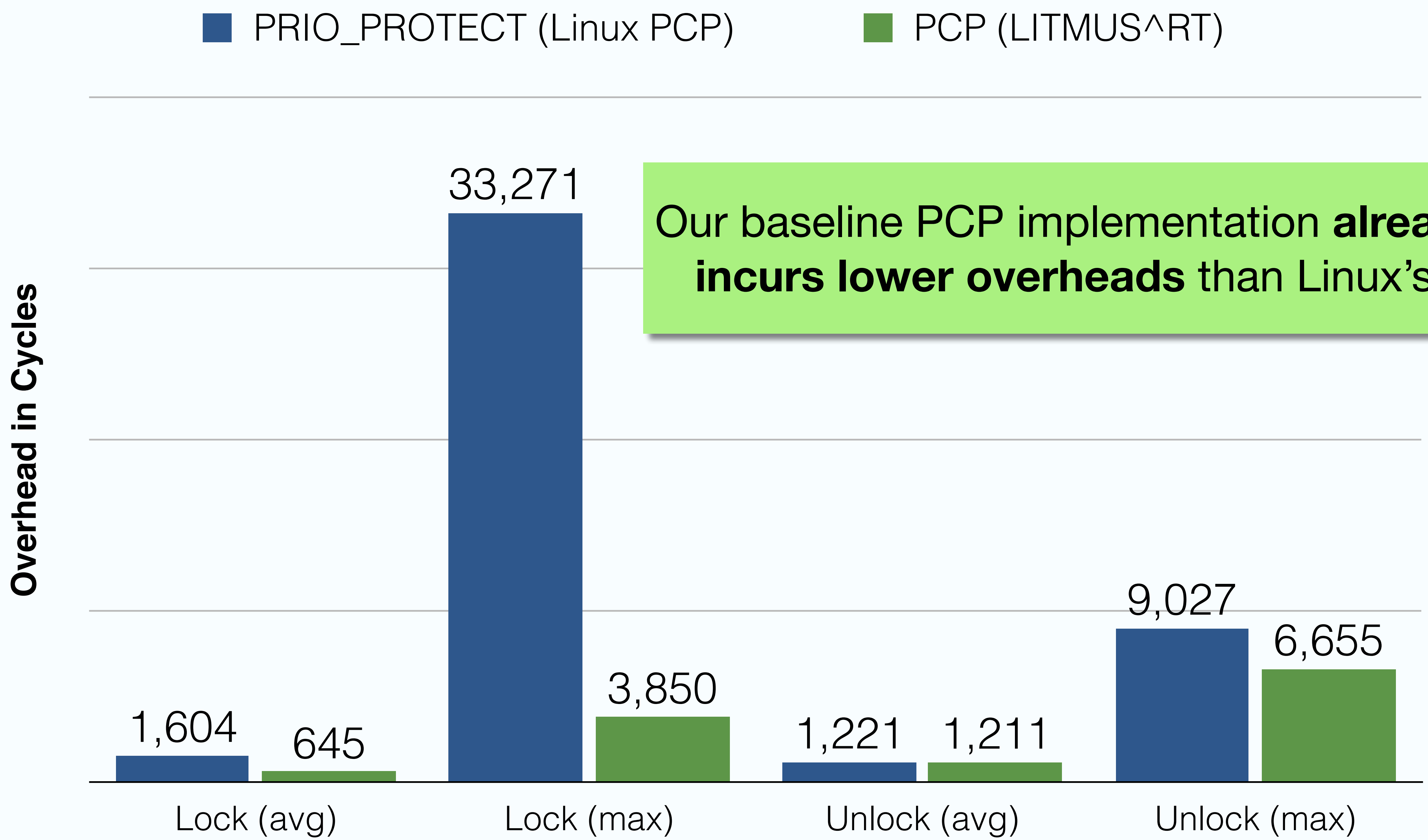
Linux

LITMUS^{RT}

PCP Futex Evaluation — Baseline



PCP Futex Evaluation — Baseline

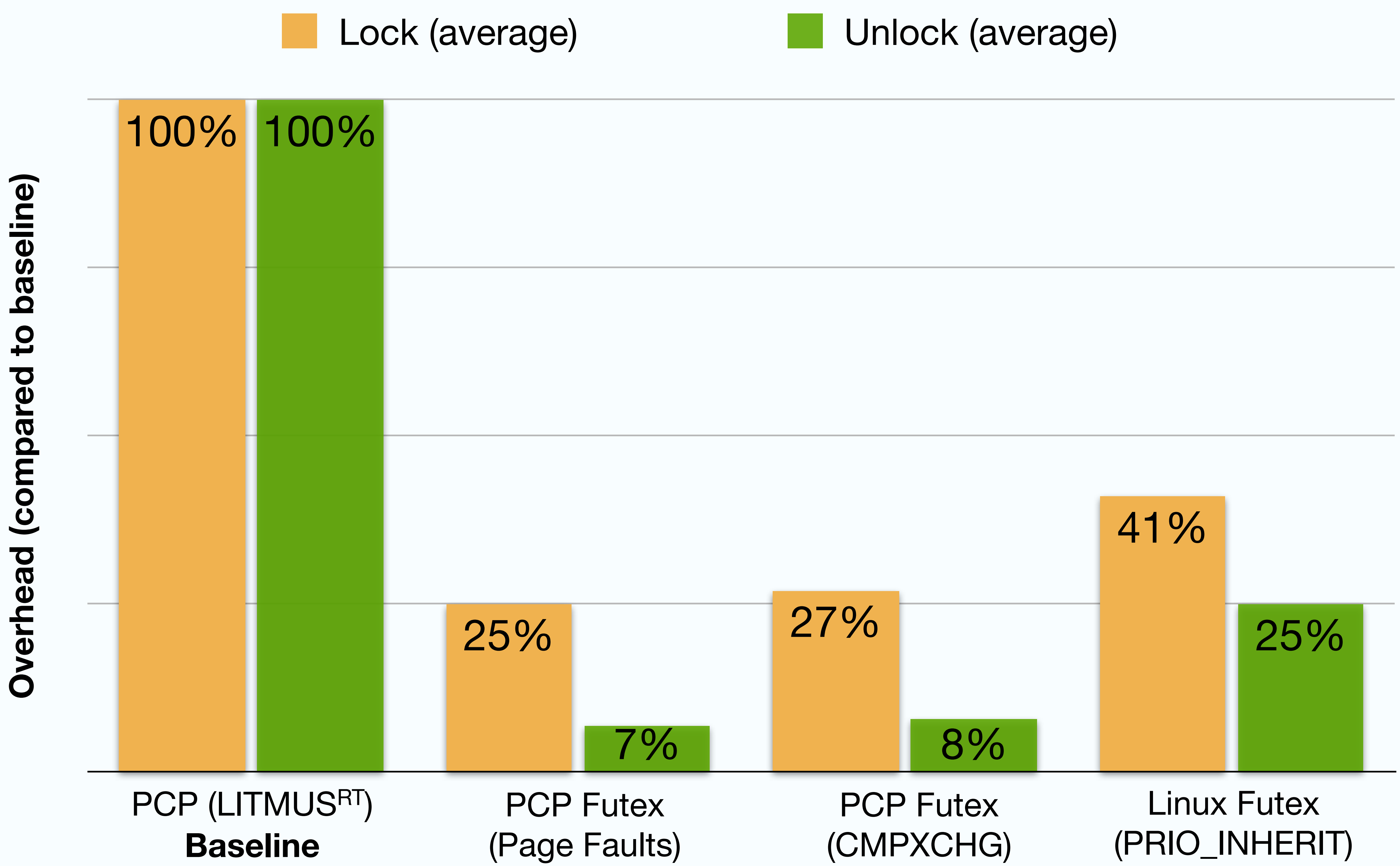


Our baseline PCP implementation **already incurs lower overheads** than Linux's.

PCP Futex Evaluation — Uncontended Case

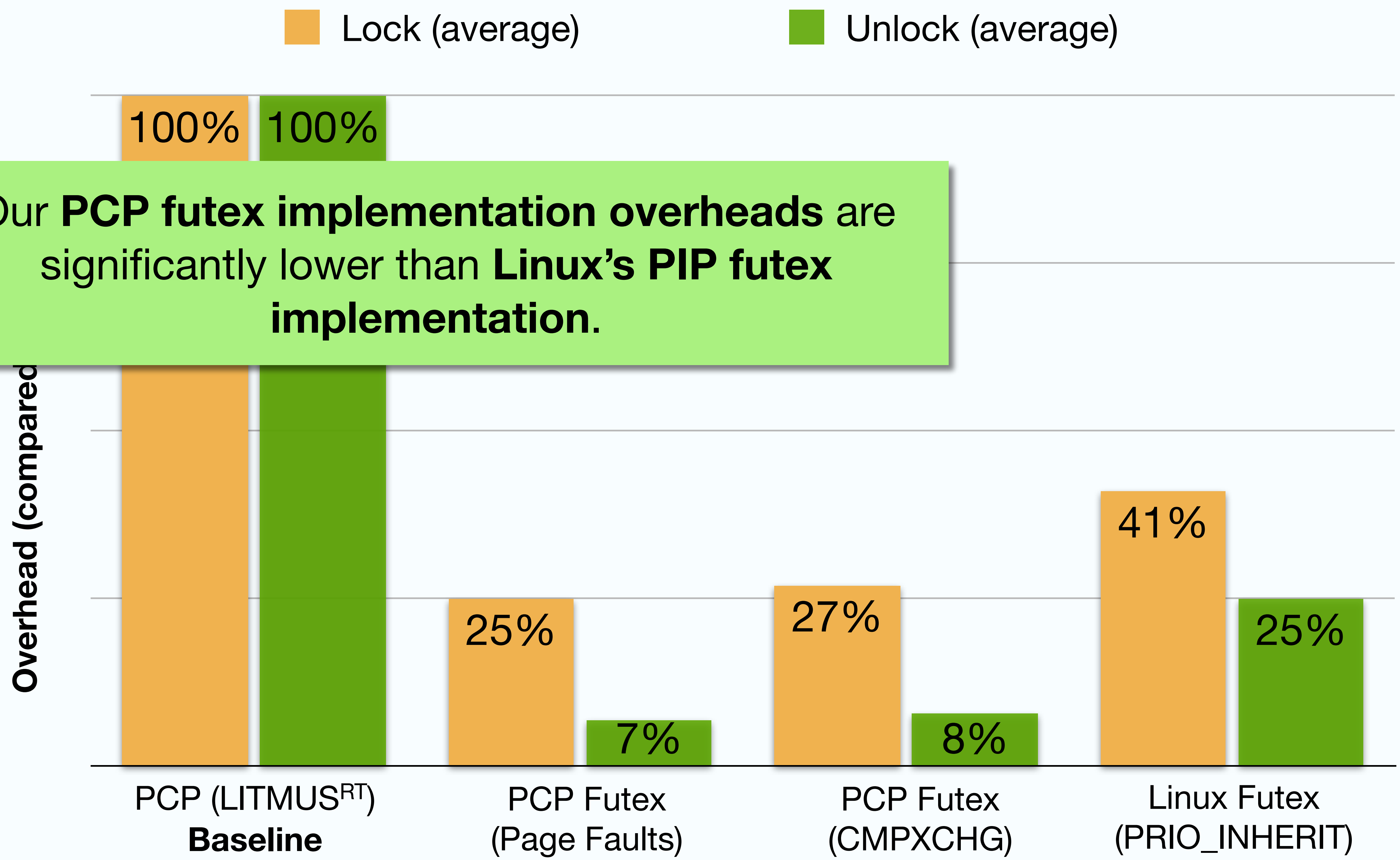
How much reduction do we see in average uncontended-case overheads?

PCP Futex Evaluation — Uncontended Case



Based on 6.6 million samples per protocol

PCP Futex Evaluation — Uncontended Case



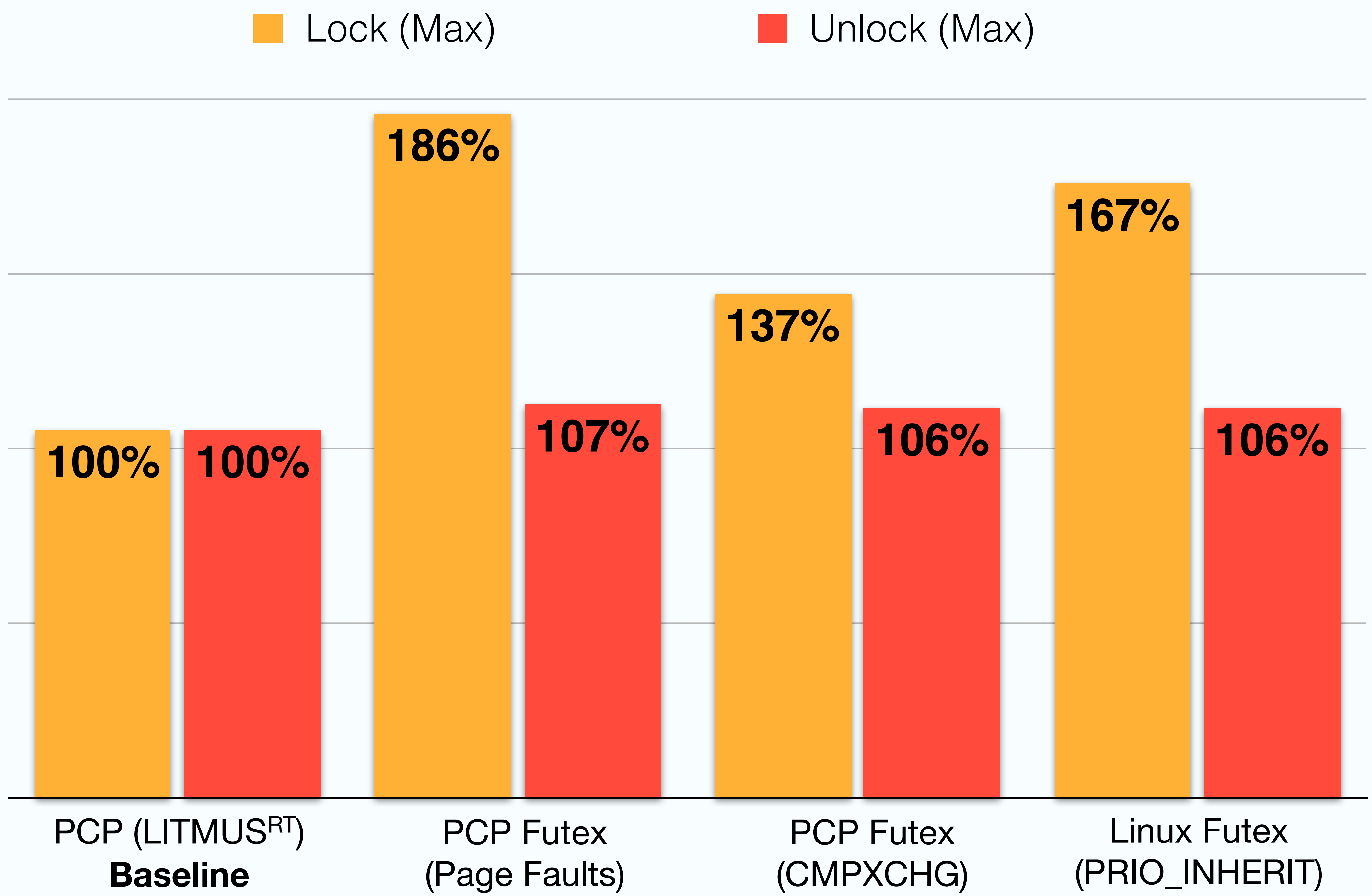
Our **PCP futex implementation overheads** are significantly lower than **Linux's PIP futex implementation.**

Based on 6.6 million samples per protocol

PCP Futex Evaluation —Contended Case

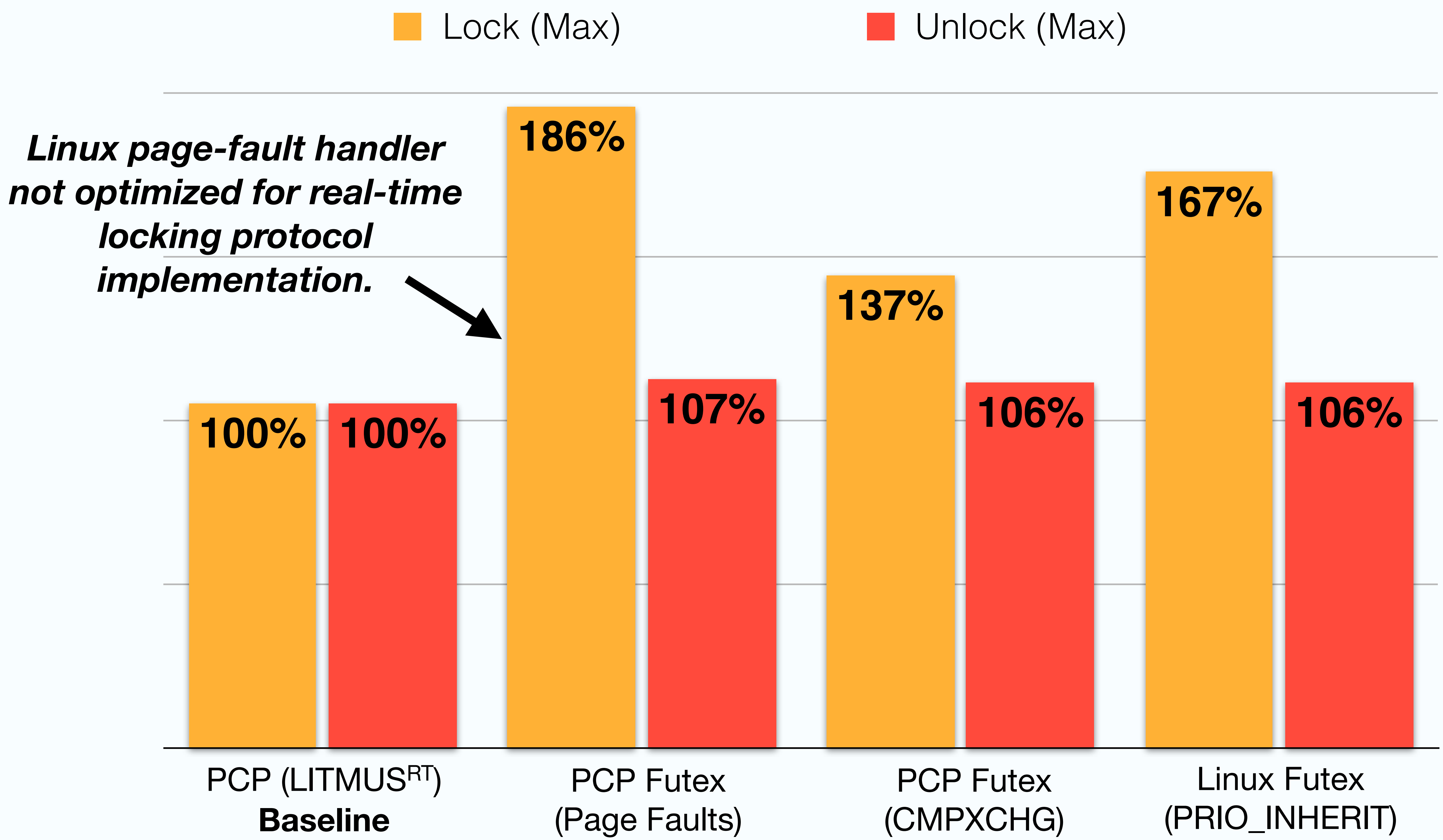
How much additional overhead do we incur in the maximum contended-case overheads?

PCP Futex Evaluation —Contended Case



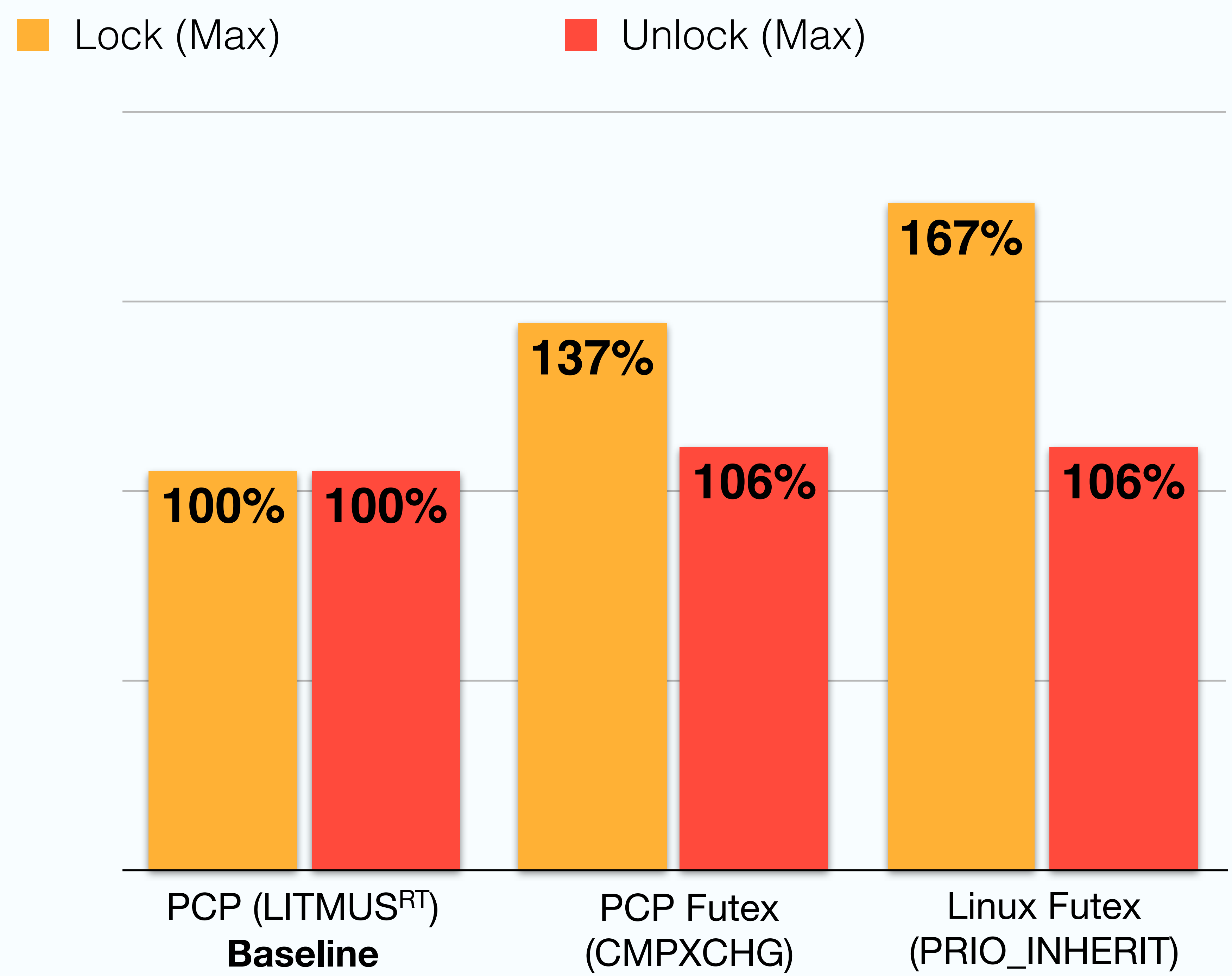
Based on 900K samples per protocol

PCP Futex Evaluation — Contended Case



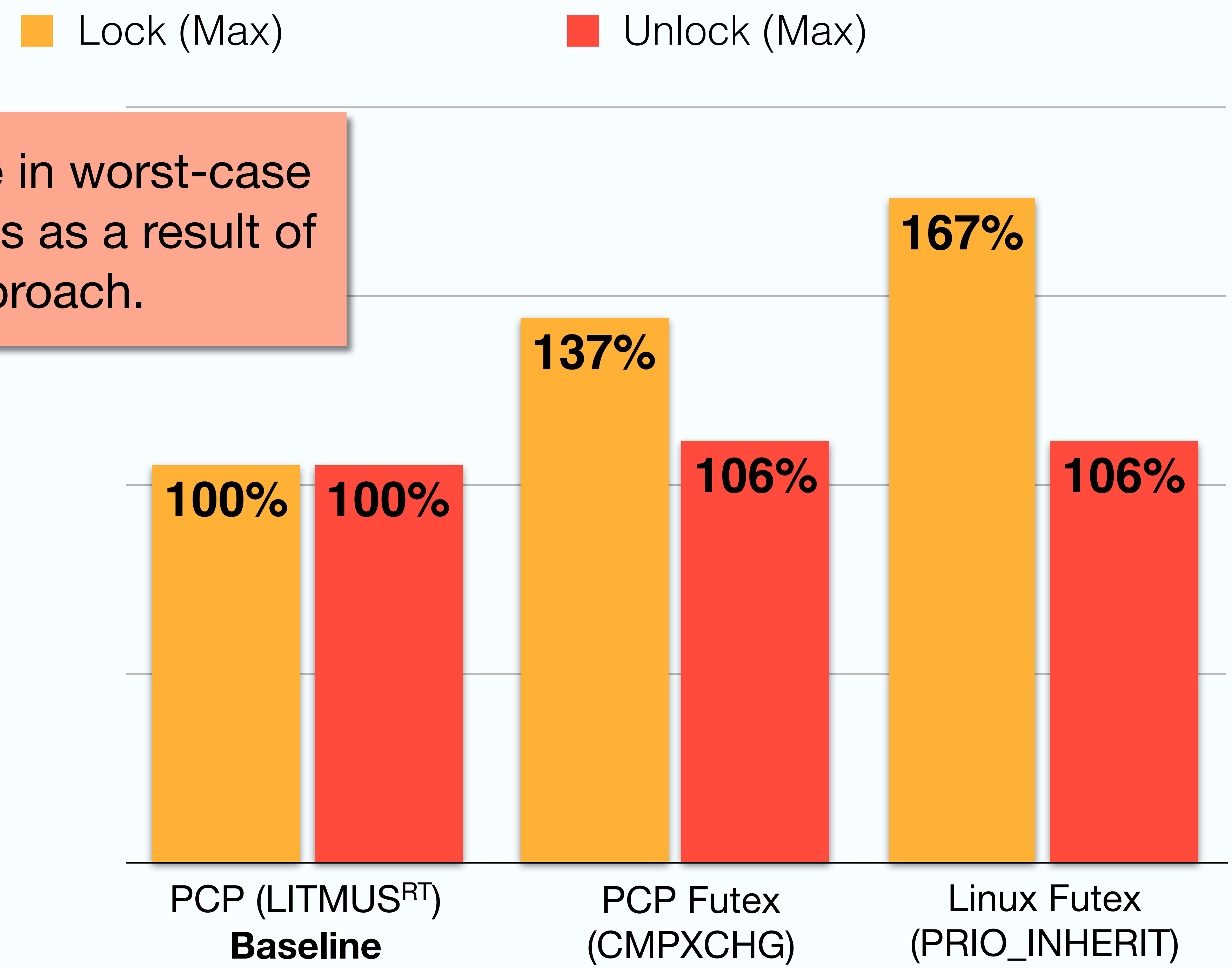
Based on 900K samples per protocol

PCP Futex Evaluation — Contended Case



Based on 900K samples per protocol

PCP Futex Evaluation — Contended Case



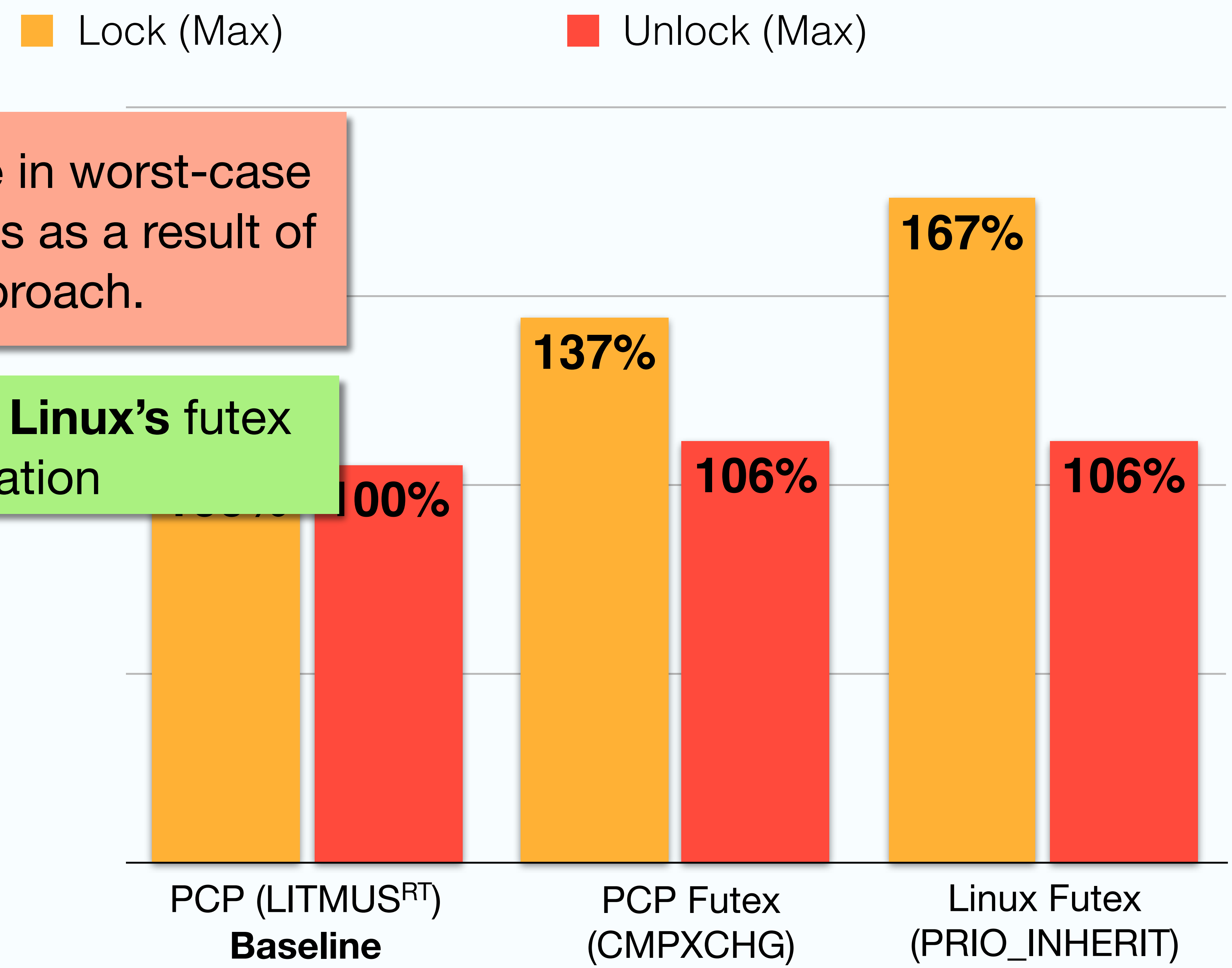
Up to **37% increase** in worst-case contended overheads as a result of the futex approach.

Based on 900K samples per protocol

PCP Futex Evaluation — Contended Case

Up to **37% increase** in worst-case contended overheads as a result of the futex approach.

But **no worse than Linux's** futex implementation



Based on 900K samples per protocol

PCP Futex Evaluation — Summary

Up to **92% reduction in average uncontended overheads** at a cost of 37% increase in maximum contended overheads (**no worse than Linux**).

Page faults can be used to implement futexes for anticipatory real-time locking protocols.

Summary

	PRIO_INHERIT (Priority Inheritance Protocol)	PRIO_PROTECT (Immediate Priority Ceiling Protocol)	Classic Priority Ceiling Protocol (PCP)	Multiprocessor PCP (MPCP)	FMLP
Futex Implementation	✓	✗	✓	✓	✓
Observed overhead reduction:			**92%**	**85%**	**84%**

Design and implementation of futexes for **three anticipatory real-time locking protocols**

Method of **using page faults** to implement futexes

Thanks!

LITMUS RT

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

All source code available at: http://www.litmus-rt.org/