Lightweight Real-Time Synchronization under P-EDF on Symmetric and Asymmetric Multiprocessors

Alessandro Biondi[*†] and Björn B. Brandenburg[*]

[*] *MPI-SWS, Kaiserslautern, Germany*

[†] *Scuola Superiore Sant'Anna, Pisa, Italy*

Artifact * ECRTS * AE * Evaluated * Consistent * Complete * Well Documented * Easy to Reuse *

Max Planck Institute for Software Systems

Retis Real-Time Systems Laboratory

THIS WORK

Revisit lightweight synchronization under P-EDF

1. Generic analysis framework for P-EDF to cope with ***synchronization delays***.

2. Analysis with a state-of-the-art technique of ***lock-free synchronization*** and ***spin locks***.

3. Large-scale experimental study that considers both **symmetric** and ***asymmetric*** multiprocessors.

MOTIVATION

- No analysis for **lock-free synchronization** on multiprocessors published to date
- New, much less pessimistic blocking analysis for **spin locks** recently proposed, but **limited** to **P-FP**
- Synchronization in **asymmetric** multiprocessors not studied so far

FINDINGS

With the new analysis

- **FIFO spin locks** confirmed to **perform best** on **symmetric** multiprocessors
- **Lock-free synchronization** found to offer **significant advantages** on **asymmetric** multiprocessors

ESSENTIAL BACKGROUND

WHY P-EDF?

Partitioned EDF (**P-EDF**) is a pragmatically good choice for multiprocessor real-time systems:

- Very accurate **schedulability analysis**;
- Empirical **good performance** at high utilizations;
- **Low** runtime **overhead**;
- Good **scalability** *(#cpus, #tasks)*;
- Used as the basic mechanism for powerful **semi-partitioned** scheduling mechanisms (e.g., C=D splitting).
- Today **available in RTOSs** (e.g., **SCHED_DEADLINE** in Linux and **ERIKA Enterprise**);

PURE SCHEDULING IS NOT ENOUGH!

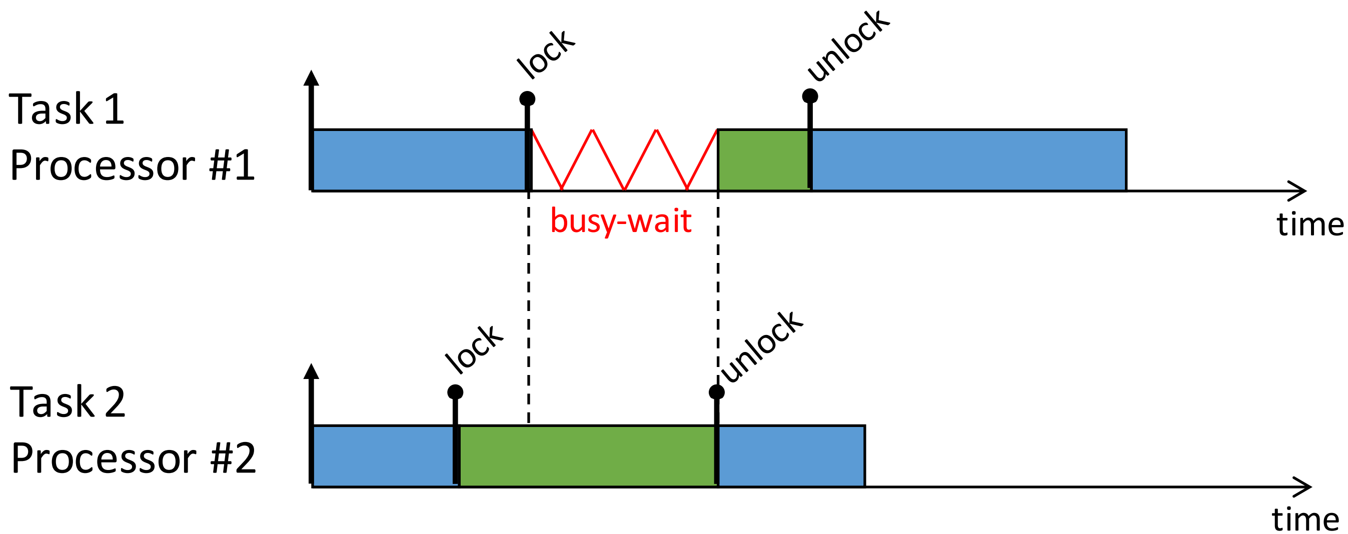
- **Real-word** applications **share resources** (buffers, data structures,…).
- Need for **predictable** and **efficient** **synchronization mechanisms**.
- How to **synchronize** under P-EDF has received considerable attention in prior work

Non-preemptive FIFO **spin locks** perform **best**:

- Highly **predictable**;
- **Lightweight** (low runtime overhead);
- Analytically **well-understood**.

SPIN LOCKS

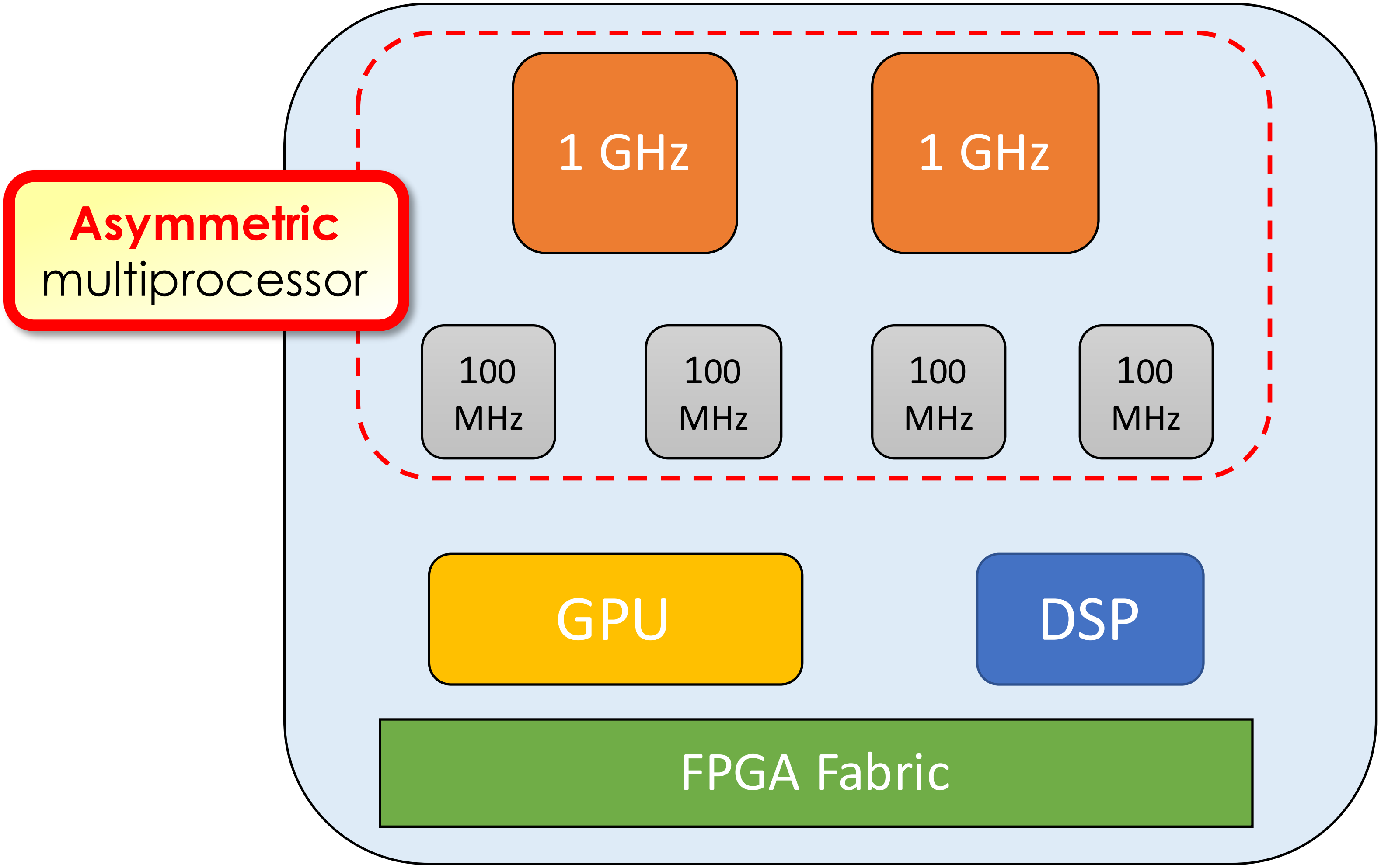
- Tasks **busy-wait** by executing **spin loop** until access to the resource is granted



Synchronization delay strictly depends on the **duration** of conflicting critical sections

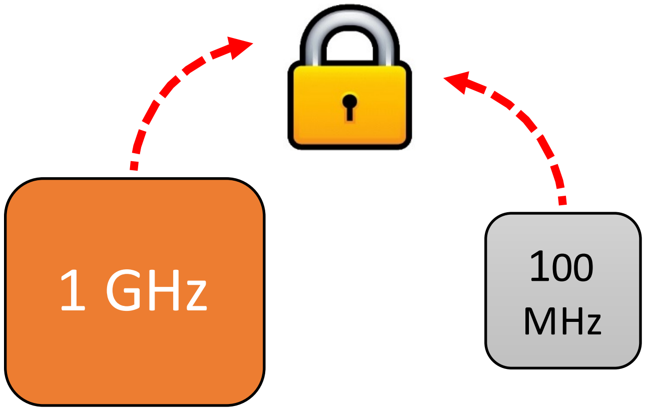
HETEROGENEOUS PLATFORMS

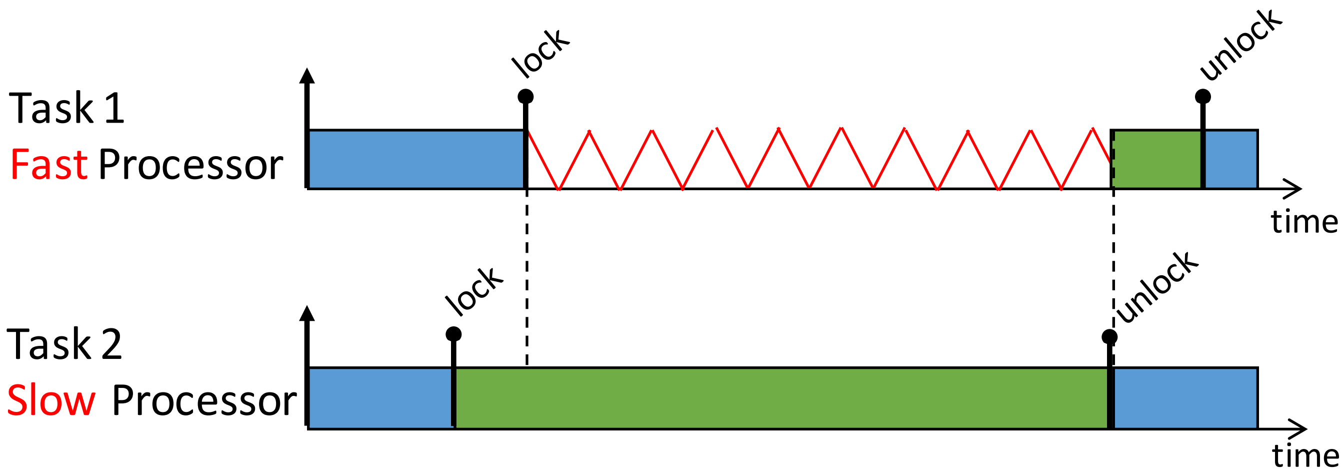
- Emerging in the **embedded** domain



ASYMMETRIC MULTIPROCESSORS

- Locks in **asymmetric** multiprocessors: possible disadvantages





ARE LOCKS THE BEST CHOICE (*from a worst-case perspective*) FOR ASYMMETRIC MULTIPROCESSORS?

What about **lock-free** synchronization mechanisms?

LOCK-FREE SYNCHRONIZATION

- Each task works on a **local copy** of (*a part of*) the shared resource and **tries** to perform an **atomic commit** to **publicize** its changes.
- If the commit **fails**, the task **retries**.

Commit loop

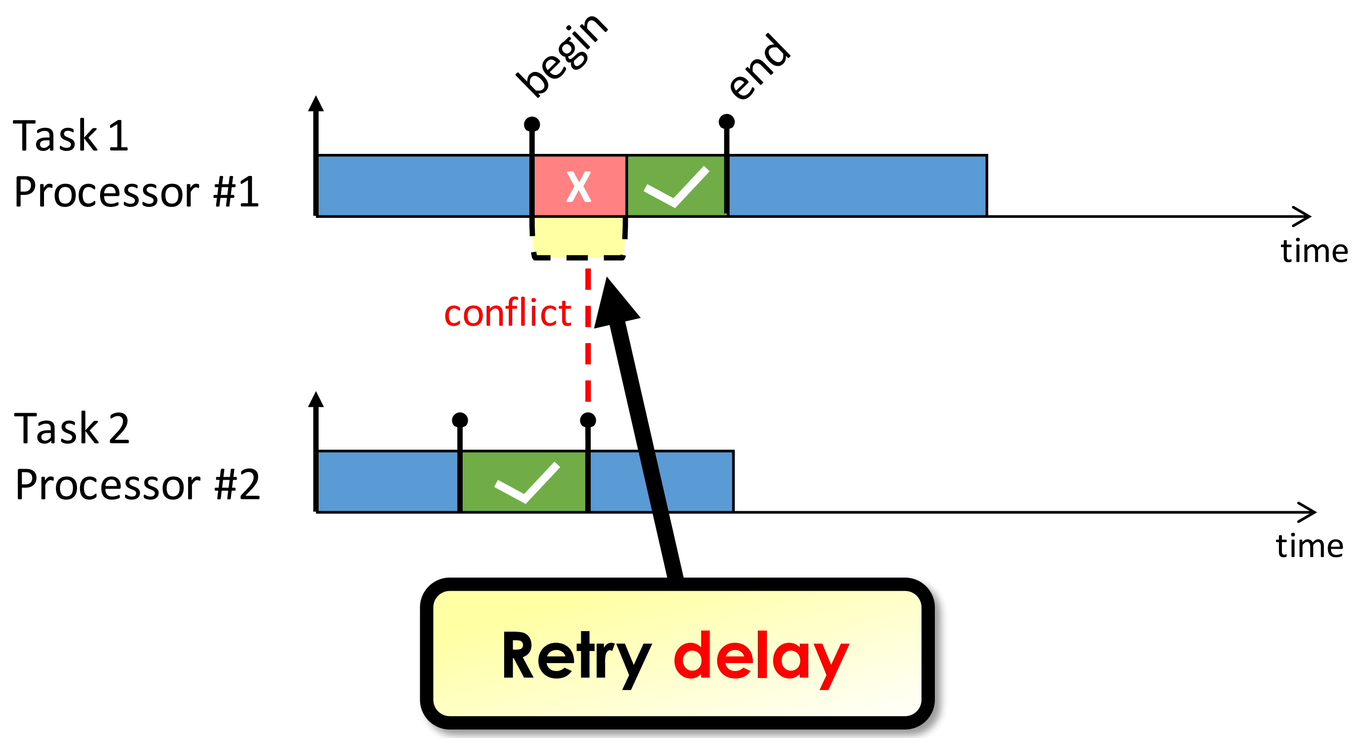
```
do
{
        <copy shared copy to local copy>
        <modify local copy>

} while not(atomic_commit())
```

Introduce **retry delay**

Typically implemented with **Compare-and-Swap**

LOCK-FREE SYNCHRONIZATION

- Example: two tasks running on two processors and sharing a resource subject to **lock-free synchronization**



LOCK-FREE SYNCHRONIZATION

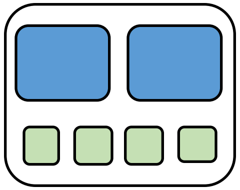
The delay is independent of the **duration** of the conflicting commit loops

Allows **decoupling** time domains in asymmetric multiprocessors

Weak progress mechanism (*unordered*): in the worst-case, every overlapping request **conflicts**

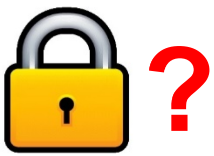
Tends to **perform worse** in the *worst case* compared to other mechanisms (e.g., FIFO-ordered locks)

LIMITATIONS OF THE SoA



Only **symmetric** multiprocessors have been considered so far.

What about emerging platforms that include **asymmetric** multiprocessors?



No up-to-date analysis of **lock-free synchronization** for multiprocessor systems in the published literature.

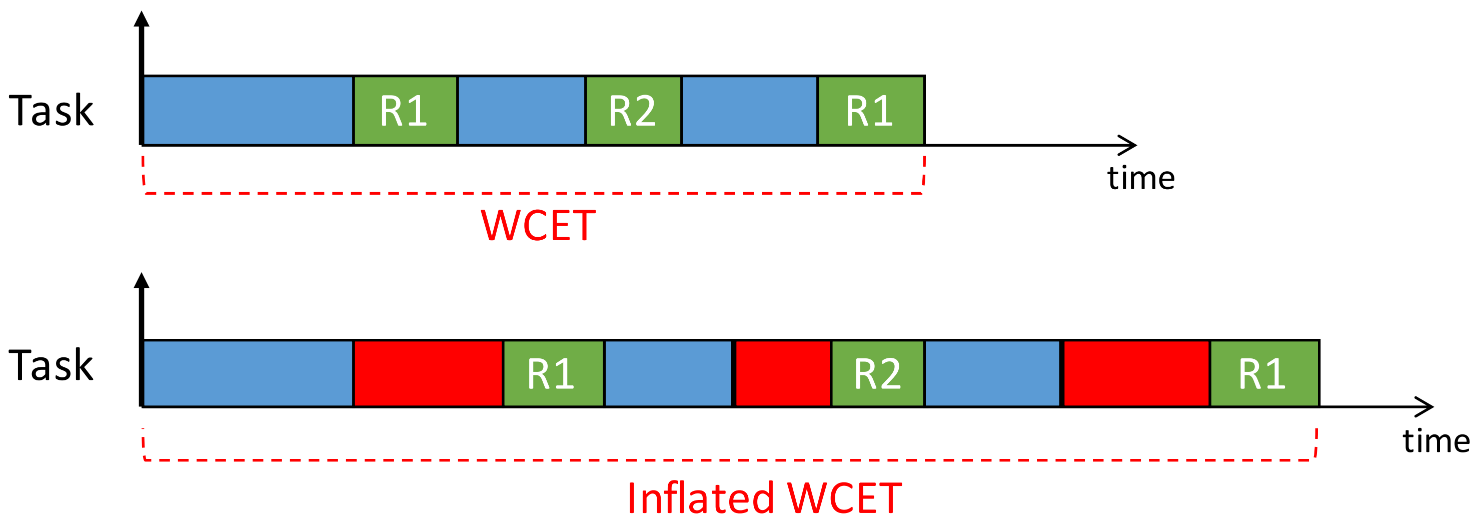
Are spin locks the best choice even with today's analysis techniques?

New, significantly improved blocking **analysis techniques** proposed in recent years, but limited to P-FP.

P-EDF has regrettably fallen behind the SoA in terms of real-time synchronization support.

INFLATION-BASED ANALYSIS

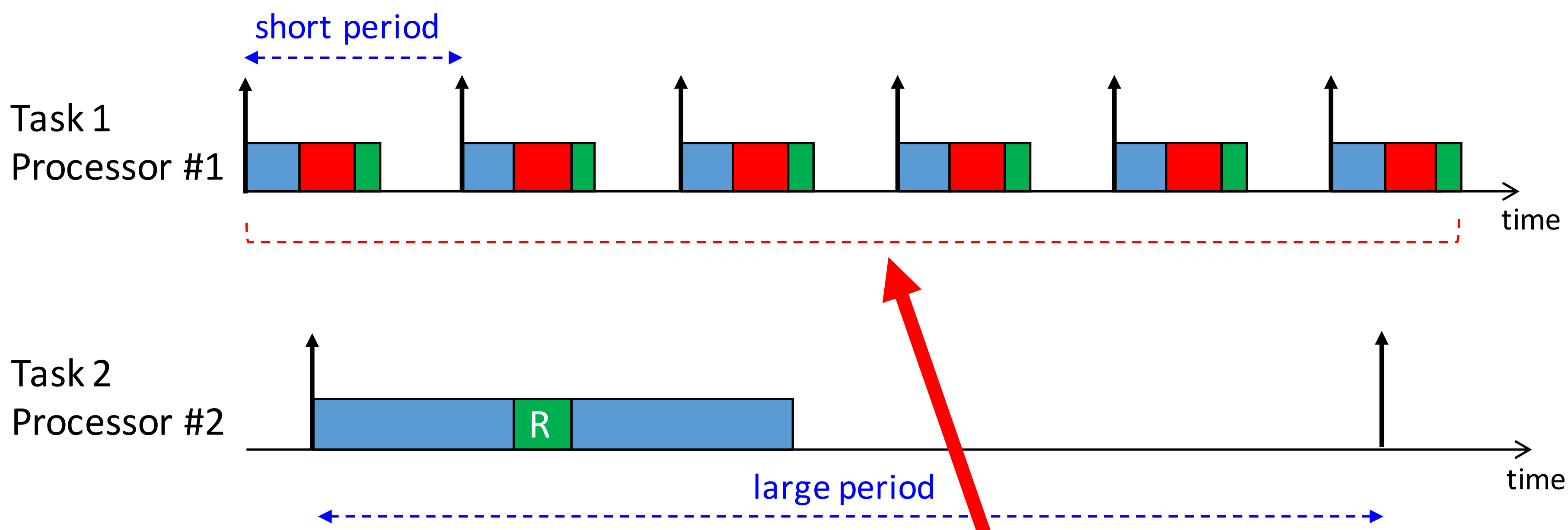
- Approach in **previous works**: **inflation** of task's WCET with a *coarse bound* on the synchronization delay.



Safe, but substantial **pessimism**

INFLATION-BASED ANALYSIS

Example



Only **one** of these jobs can **conflict** with the critical section of Task 2

INFLATION-FREE ANALYSIS FOR P-EDF

How to **bound** synchronization delay **without** inflating task WCETs?

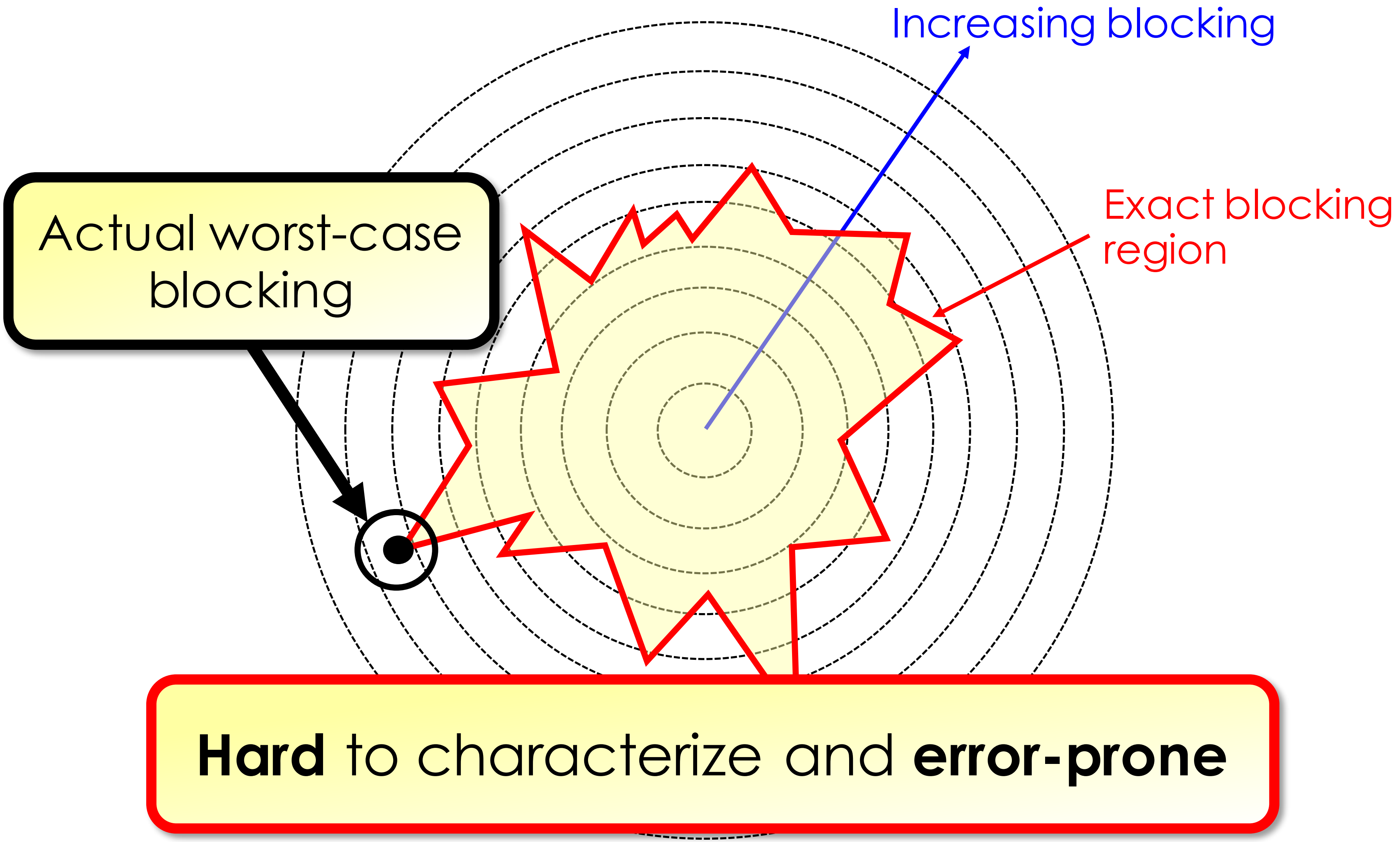
INFLATION-FREE ANALYSIS

- Do **not** inflate tasks's WCET but **explicitly** account for **synchronization delay**

Approach outline

- **Identify** all the conflicting requests;
- **Never** count a critical section more than once as contribution to synchronization delay;
- Characterize the **maximum** synchronization delay in every possible schedule.

THE BLOCKING SPACE



Hard to characterize and **error-prone**

LP-BASED ANALYSIS

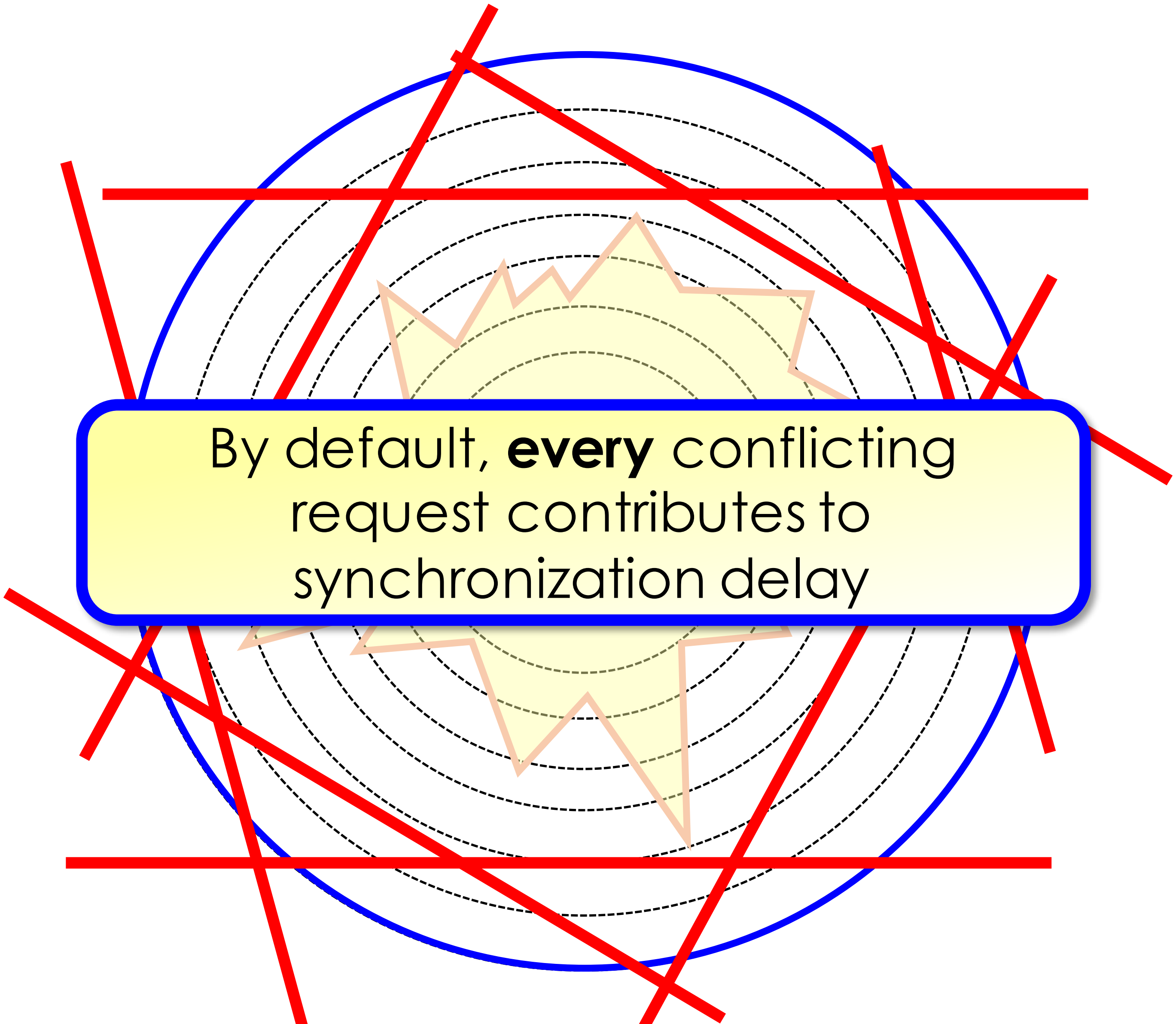
Model contribution of requests for shared resources to synchronization delay as **variables** of a **linear program**

Constraints are enforced to **exclude impossible scenarios** (e.g., encoding protocol invariants)

Maximize LP

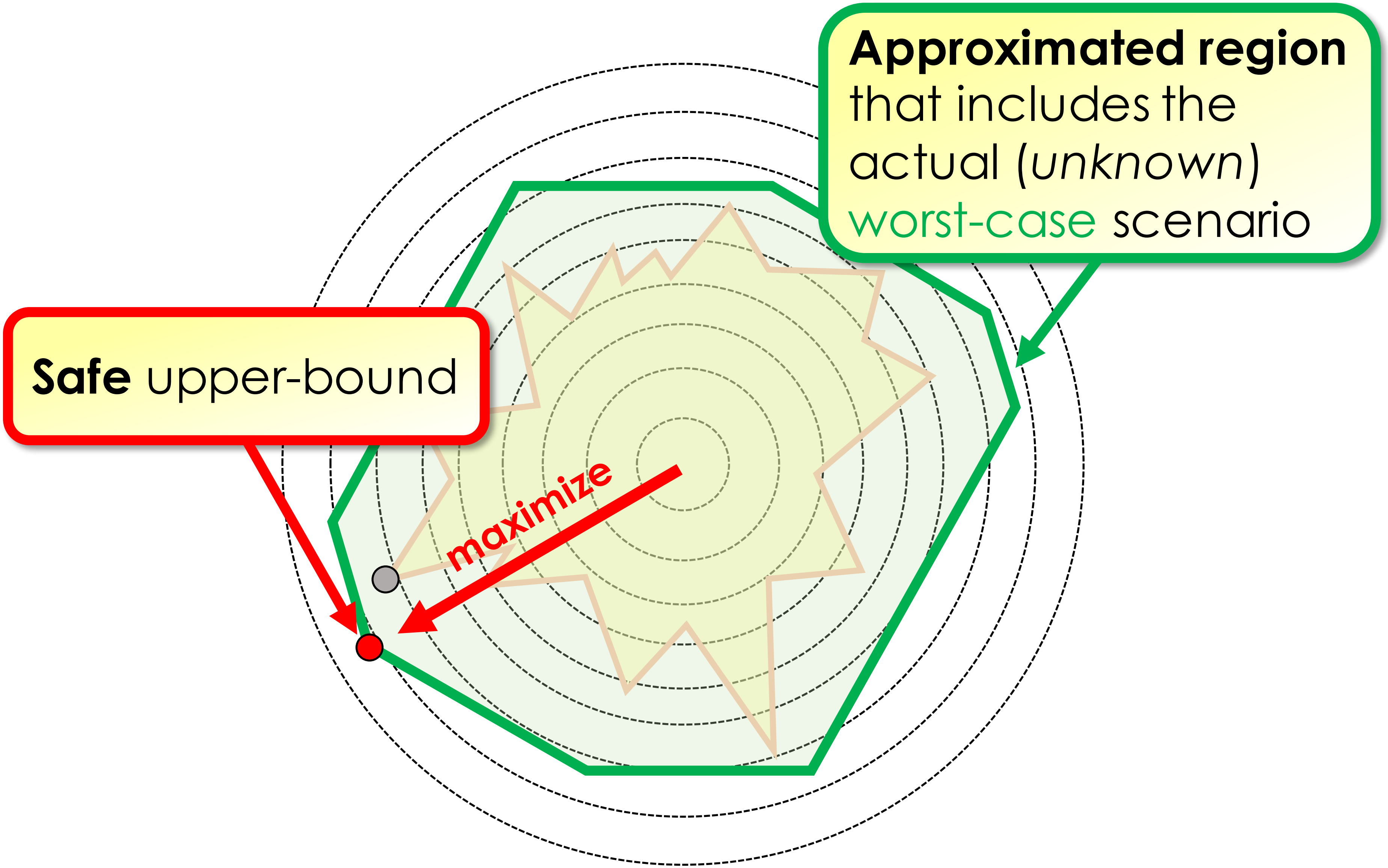
Safe upper-bound on all not-excluded scenarios, including the **actual worst-case**

LP-BASED ANALYSIS



By default, **every** conflicting request contributes to synchronization delay

LP-BASED ANALYSIS

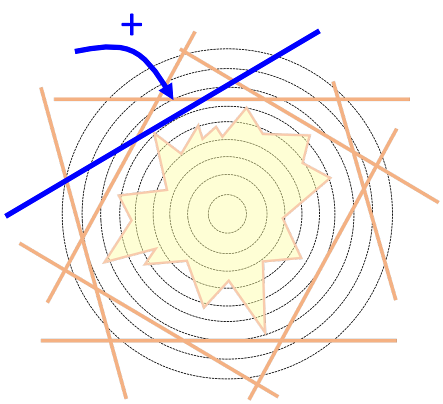


LP-BASED ANALYSIS

Other benefits of the approach

Compositionality

Every constraint can be proven **independently**

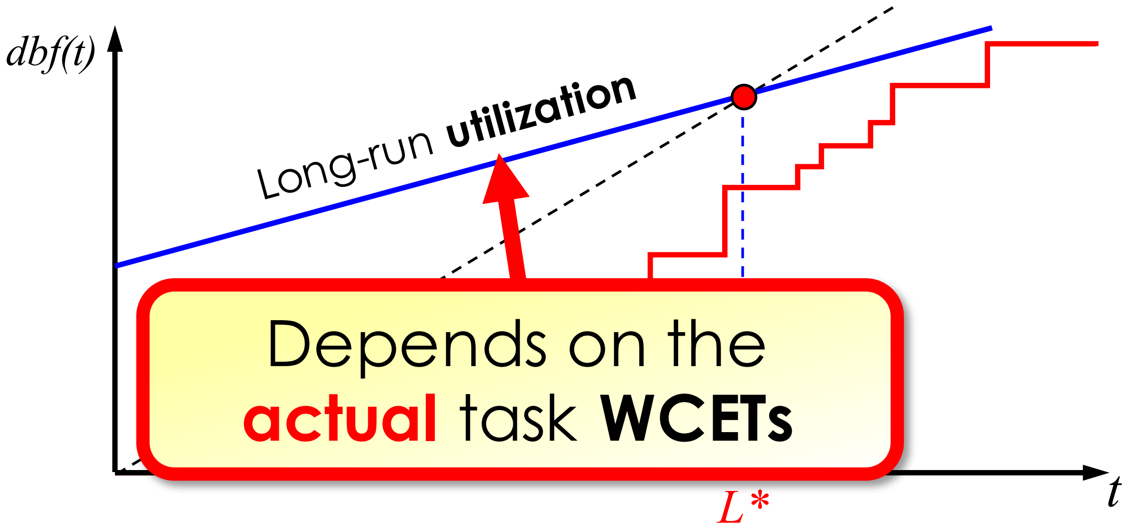


Extensibility

By “plugging in” new **application-specific** constraints (e.g., a resource is accessed only every k jobs,…)

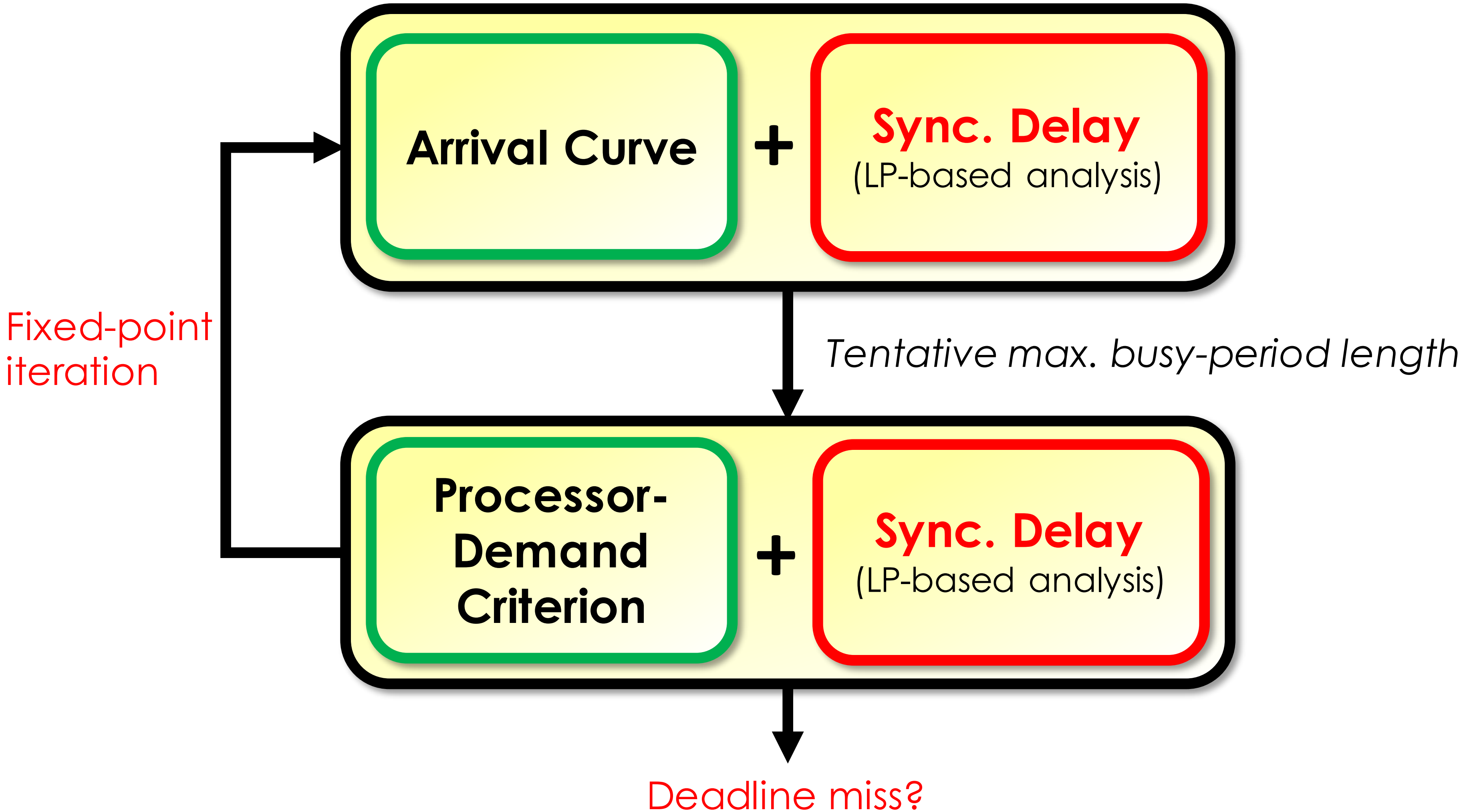
ANALYSIS FRAMEWORK

- Processor-demand Criterion (*Baruah '06*) **+** Synchronization Delay (LP-based analysis)
- How to **bound** the maximum **busy-period** length?



ANALYSIS FRAMEWORK

For each processor…



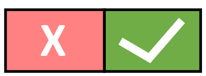
LP-BASED ANALYSIS

- We applied this approach to **4** synchronization mechanisms:
  - **Lock-free algorithms** with **preemptive** commit loops
  - **Lock-free algorithms** with **non-preemptive** commit loops
  - **Non-preemptive** FIFO **Spin Locks**
  - **Preemptive** FIFO **Spin Locks**

LP-BASED ANALYSIS

Different **modeling strategies** are required for lock-free algorithms and spin locks because of their fundamental structural differences

Lock-free Algorithms



Retry delay composed of a **number of events** (i.e., retries), modeled with *integer* variables of an *Integer Linear Program* (**ILP**).

Spin Locks

Spin delay composed of **fractions of time**, modeled with *real* variables of a *Mixed-Integer Linear Program* (**MILP**).

EXPERIMENTAL STUDY

QUESTIONS

- Are lock-free algorithms preferable (from a worst-case perspective) to spin locks on **asymmetric** multiprocessors?
- Are spin locks still preferable on **symmetric** multiprocessors, even with up-to-date lock-free analysis techniques?

OUR STUDY

- Large-scale, based on synthetic workload
- **Symmetric** multiprocessors
- **Asymmetric** multiprocessors with two types, tested a wide range of relative speeds (from **1x** to **10x**)
- **>3000** configurations have been tested to investigate the questions

ANSWERS

- Are lock-free algorithms **preferable** (from a worst-case perspective) to spin locks on **asymmetric** multiprocessors?

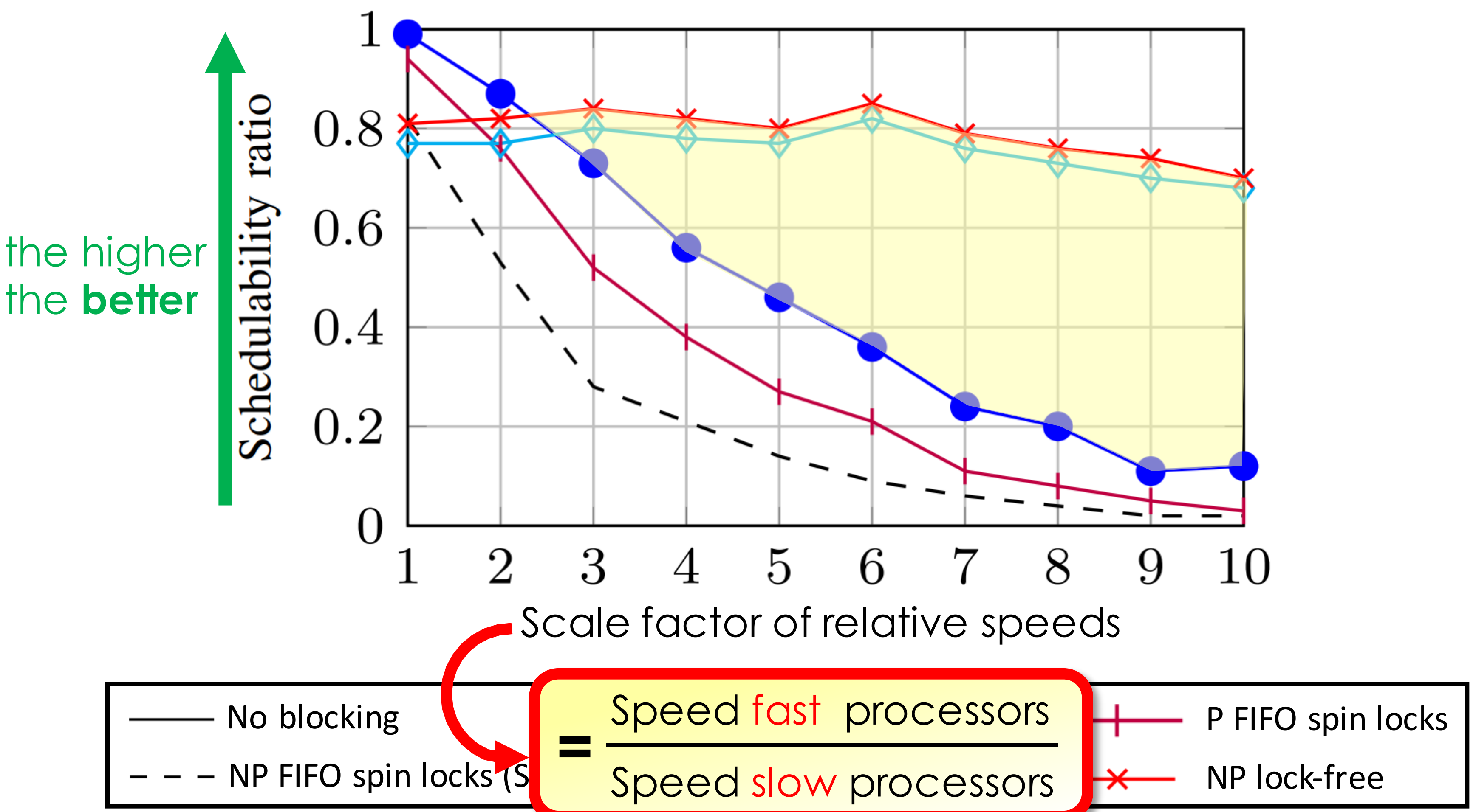
  Yes, especially in the presence of low contention and high difference in relative processors speeds.

- Are spin locks still **preferable** on **symmetric** multiprocessors, even with up-to-date lock-free analysis techniques?

  Yes, in all the tested scenarios and especially in the presence of high contention.

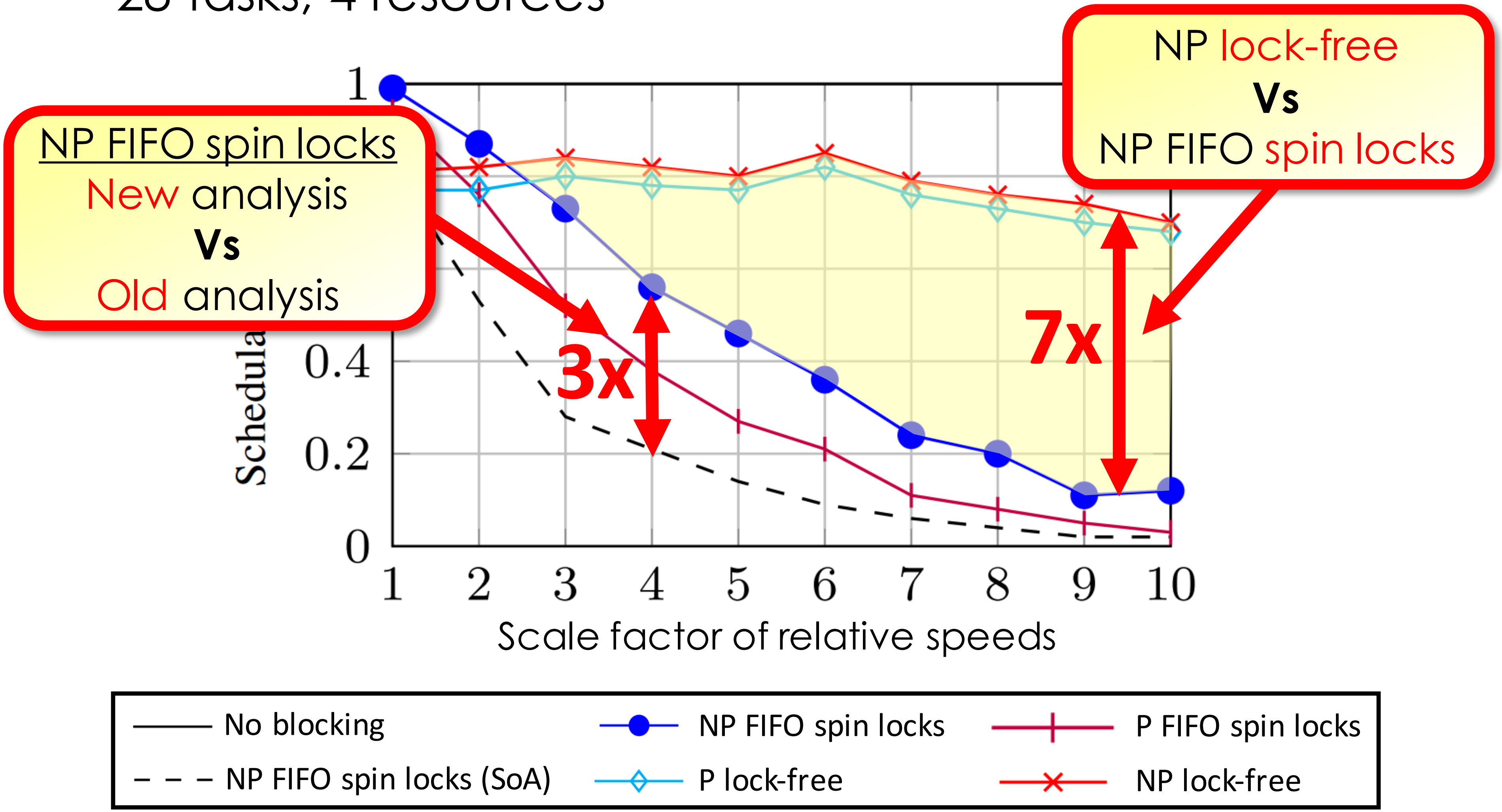
EEXPERIMENTAL RESULTS

- Asymmetric multiprocessor: **2** fast and **2** slow
  28 tasks, 4 resources



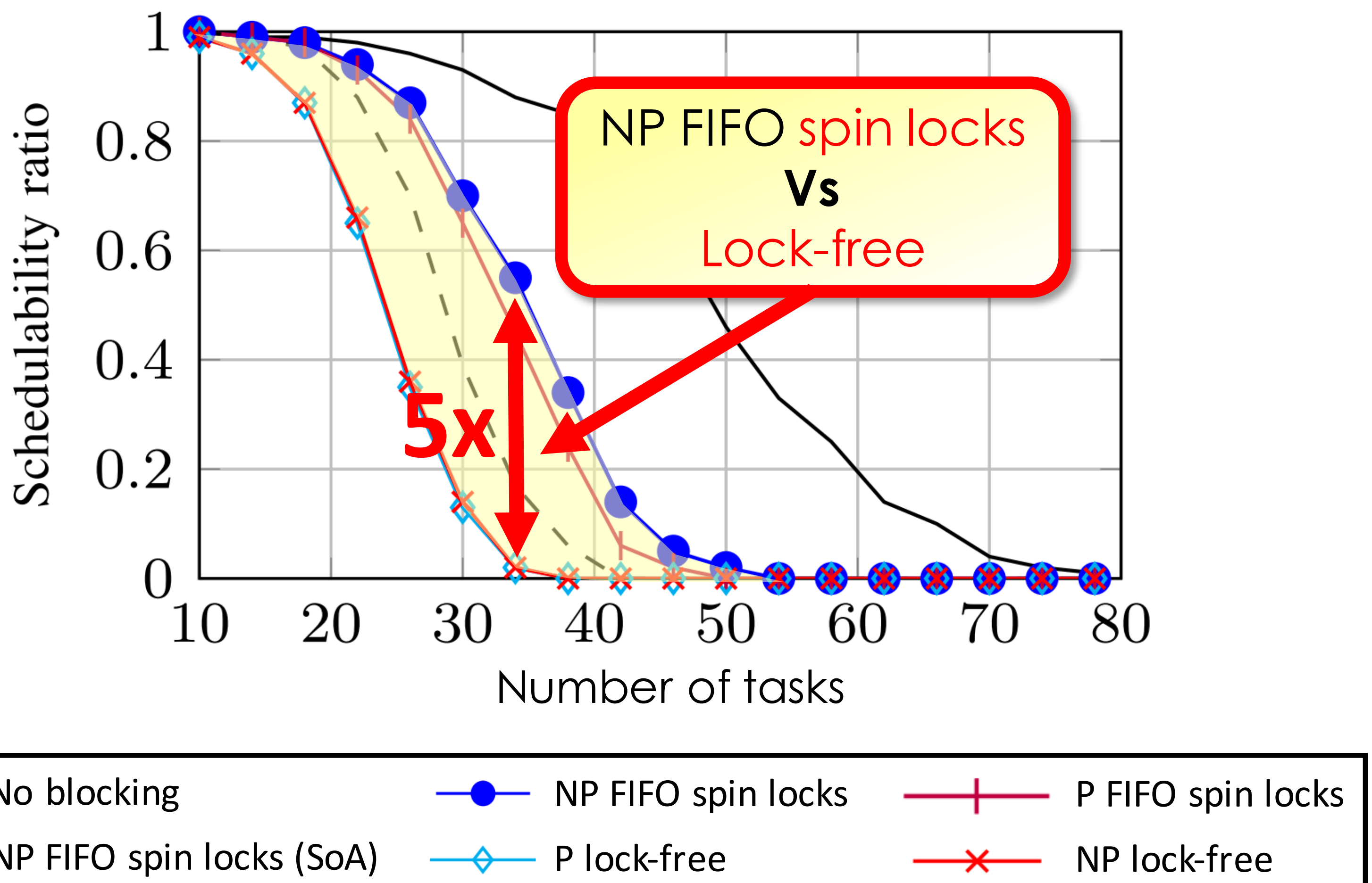
EEXPERIMENTAL RESULTS

- Asymmetric multiprocessor: **2** fast and **2** slow
  28 tasks, 4 resources



EXPERIMENTAL RESULTS

- Symmetric multiprocessor with **8** processors



CONCLUSIONS

- We took a **fresh look** at **lightweight synchronization** under **P-EDF**
- **Lock-free synchronization** and **FIFO spin locks** analyzed with a state-of-the-art technique (inflation-free analysis)
- **Experimental study** considering both **symmetric** and **asymmetric** multiprocessors

Take-away messages

- **FIFO spin locks** perform **best** on **symmetric** multiprocessors, even under **P-EDF**
- **Lock-free synchronization** offers **significant advantages** for **asymmetric** platforms

FUTURE WORK

- **Synchronization** mechanisms for semi-partitioned scheduling with **C=D**;
- **Extension** to other synchronization mechanisms (MrsP, SRP-based commit loops, wait-free,…);
- Investigation on the use of lock-free algorithms for **component-based** software design.

Thank you!

Alessandro Biondi
alessandro.biondi@sssup.it

Max
Planck
Institute
for
Software Systems