ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering and
Networks Laboratory

SF3P: A Framework to Explore and Prototype Hierarchical Compositions of Real-Time Schedulers

Andres Gomez, Lars Schor, Pratyush Kumar, Lothar Thiele

Intl. Symp. on Rapid System Prototyping
New Delhi, October 16-17, 2014

Motivation

- Highly integrated real-time systems are showing:
  - Increasingly complex functionality
  - Need for sophisticated scheduling techniques (mixed-criticality)
- Scheduler designers need to validate at early design stages
  - Prototype schedulers on different HW platforms
- Prototyping platforms should:
  - Offer a high level of abstraction (extendable)
  - Have minimal system requirements
  - Inexpensive to execute (low overhead)

Software Options in Real-Time Systems

- Unix-like OS
  - ✓ High HW/SW compatibility
  - – Limited scheduling options
- Modified Kernel Space
  - ✓ High HW compatibility
  - ✓ Customizable scheduling options
  - – Limits SW compatibility/portability

Faggioli, et al. (2009)

Asberg, et al. (2012)

Palopoli, et al. (2009)

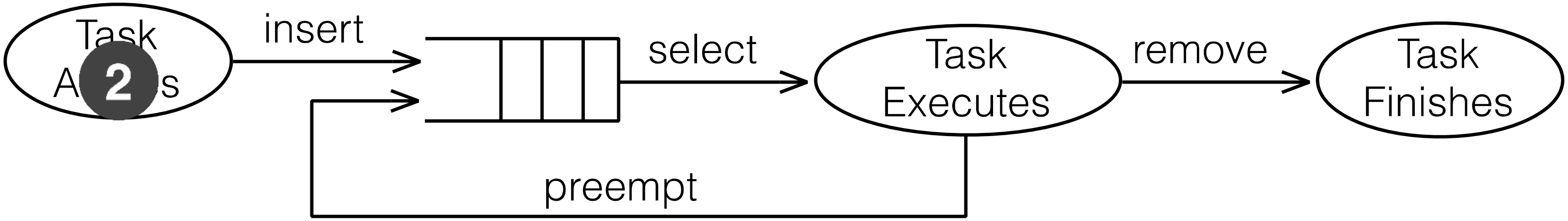
- Custom RTOS
  - ✓ Finely tuned scheduler
  - – Limited HW/SW compatibility

Buttazzo, et al (1993)

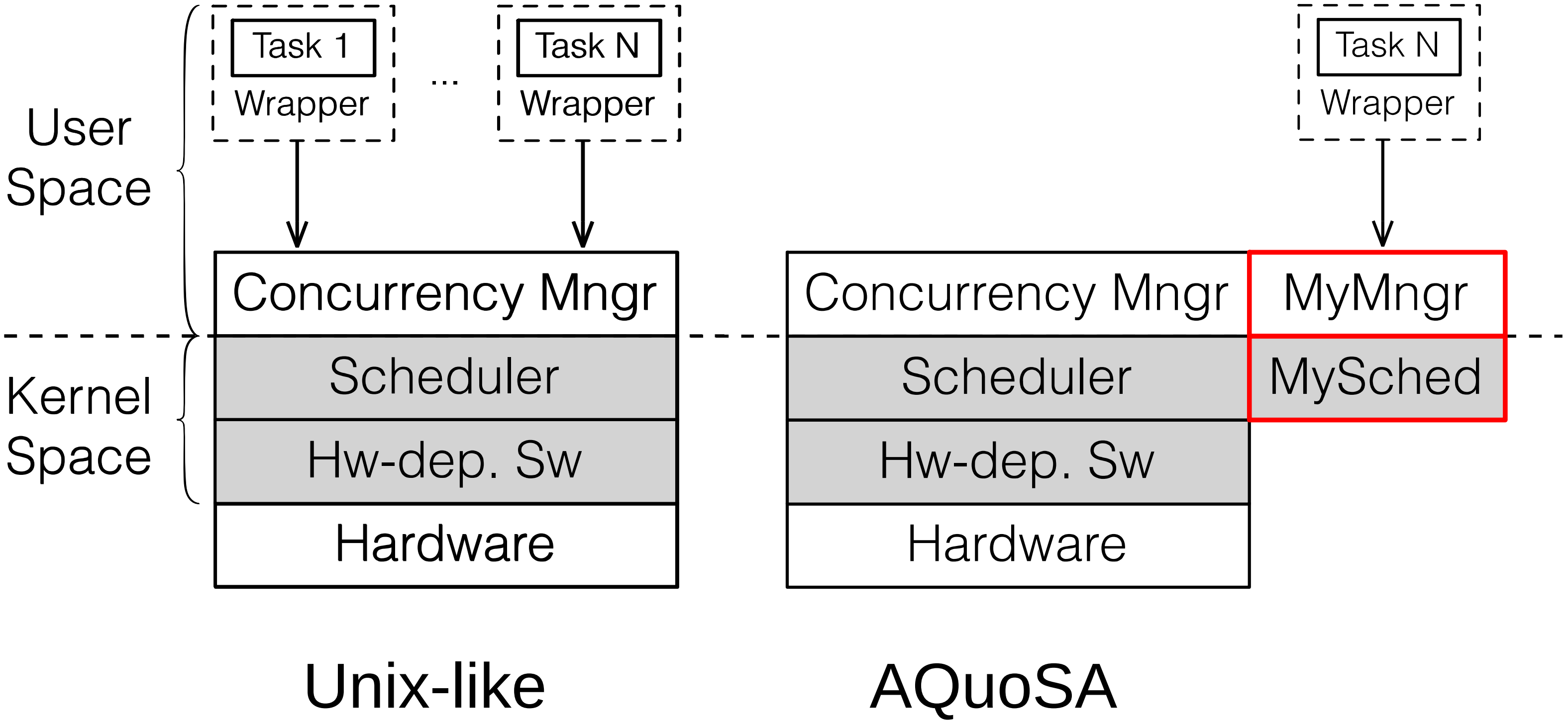
<u>Our proposal:</u>

- Add **flexible** scheduling layer on top of a **standard** kernel

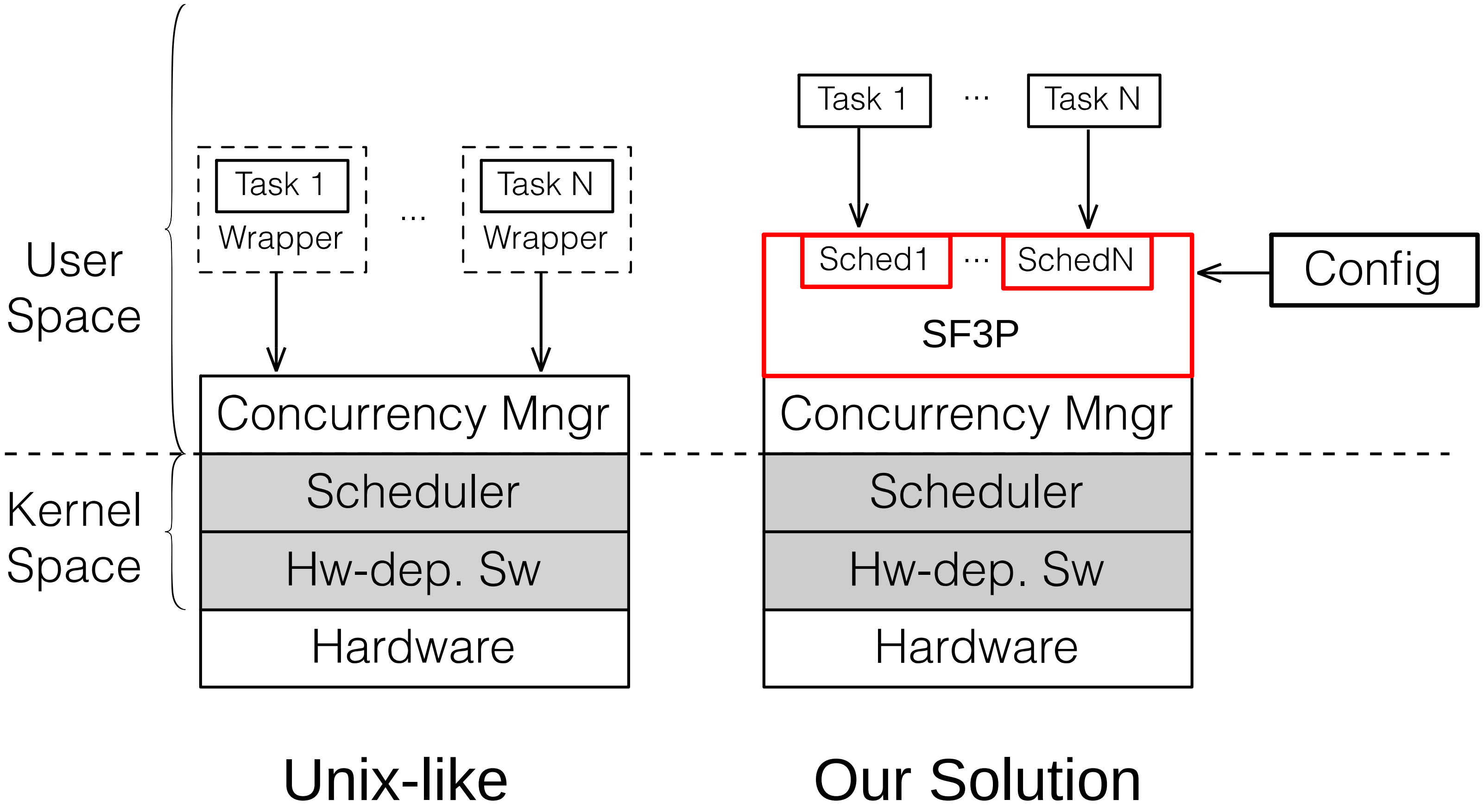
Our Scheduling Model



Scheduling in Unix-like Operating Systems



Scheduling Framework for Fast Prototyping (SF3P)



Our Goals

- We can add a scheduling layer in the User Space

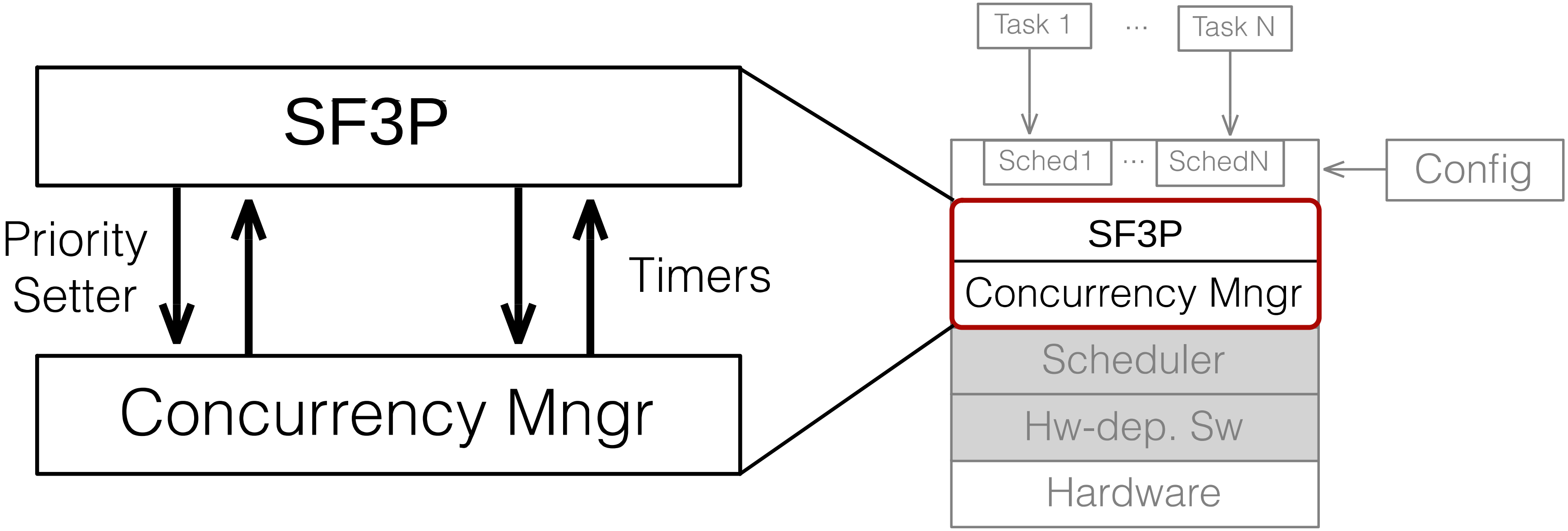
1. Portable to different platforms with no cost
2. Extendable to new schedulers with low cost
3. Low Overhead

Our Goals

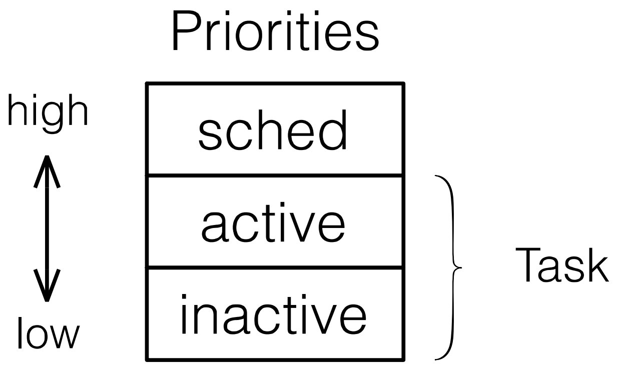
- We can add a scheduling layer in the User Space

1. Portable to different platforms with no cost
2. Extendable to new schedulers with low cost
3. Low Overhead

SF3P – Concurrency Manager Interaction



Basic Concept – How does SF3P Schedule?



Priorities

high

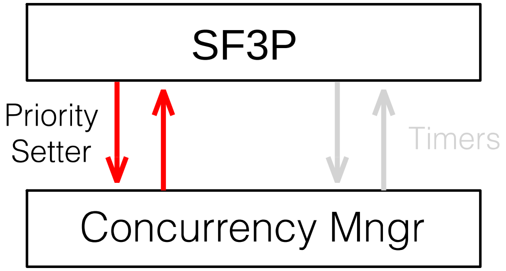
low

sched

active

inactive

Task

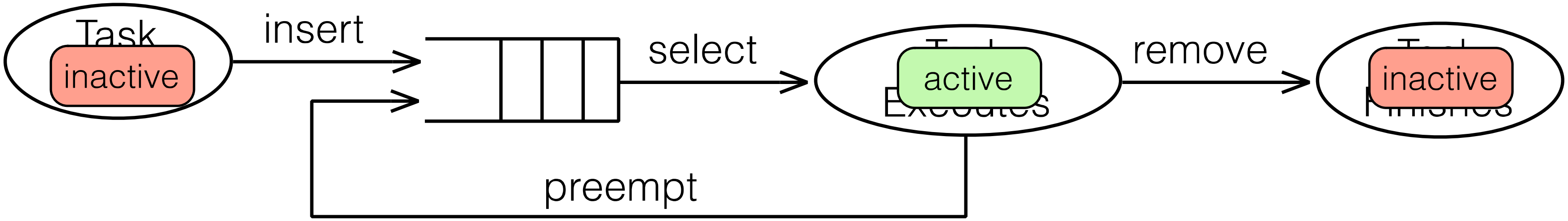


SF3P

Priority Setter

Timers

Concurrency Mngr



Task

inactive

insert

select

Task Executes

active

remove

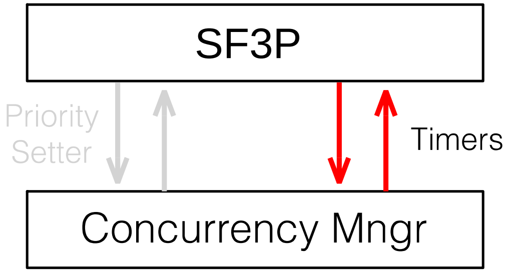
Task Finishes

inactive

preempt

Time Triggered Scheduling

- Time Division Multiple Access

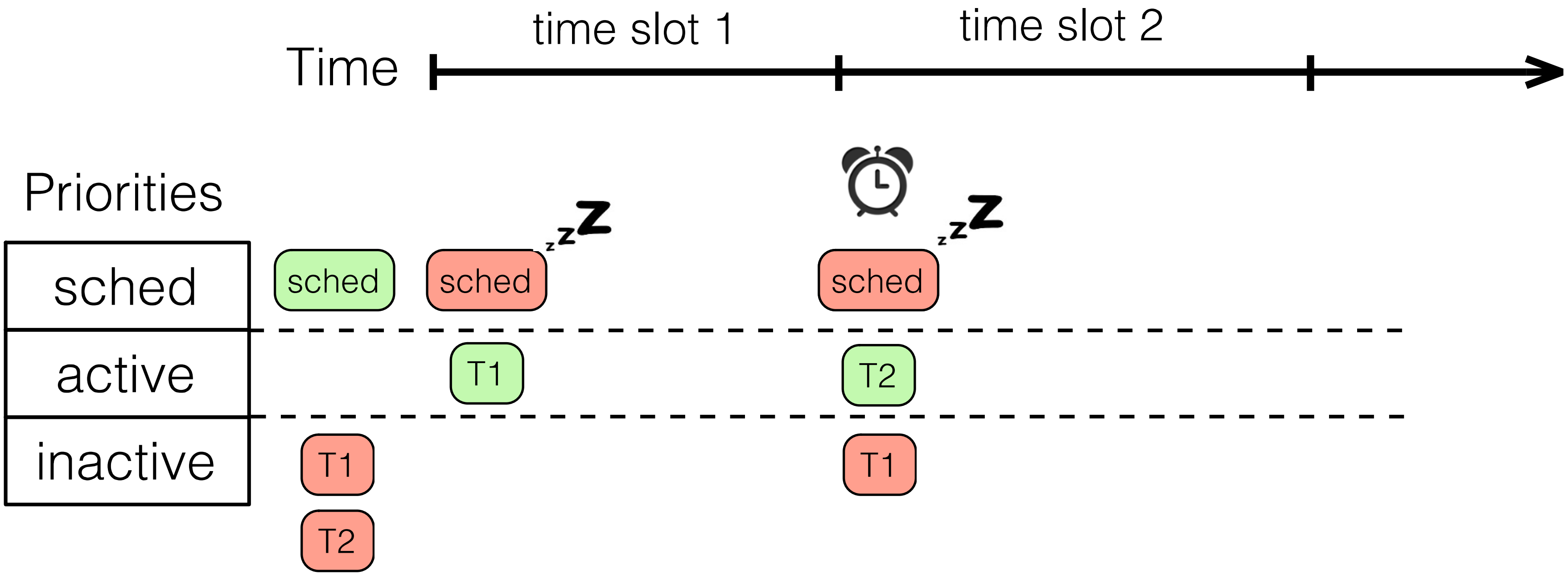


SF3P

Priority Setter

Timers

Concurrency Mngr



Time

time slot 1

time slot 2

Priorities

sched

active

inactive

sched

T1

T2

sched

T1

sched

T2

T1

Our Goals

- We can add a scheduling layer in the User Space

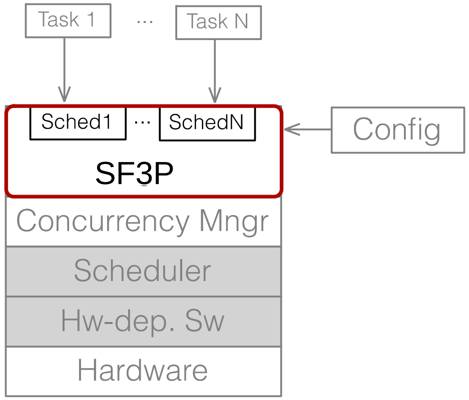
1. Portable to different platforms with no cost ✓
2. Extendable to new schedulers with low cost
3. Low Overhead

Our Goals

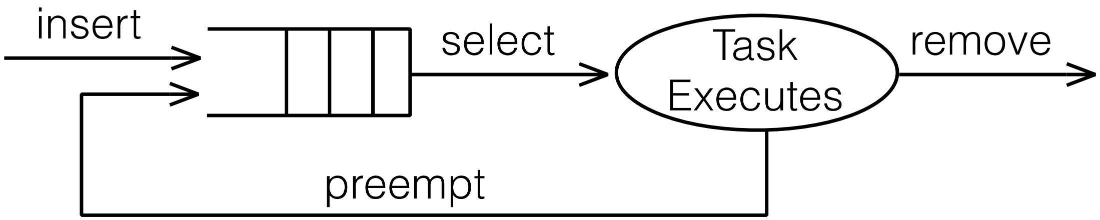
- We can add a scheduling layer in the User Space

1. Portable to different platforms with no cost ✓
2. Extendable to new schedulers with low cost
3. Low Overhead

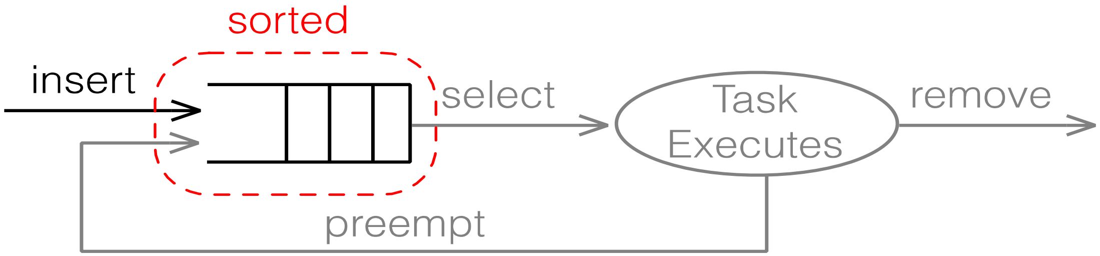
Adding a New Scheduler



- Generic Scheduler

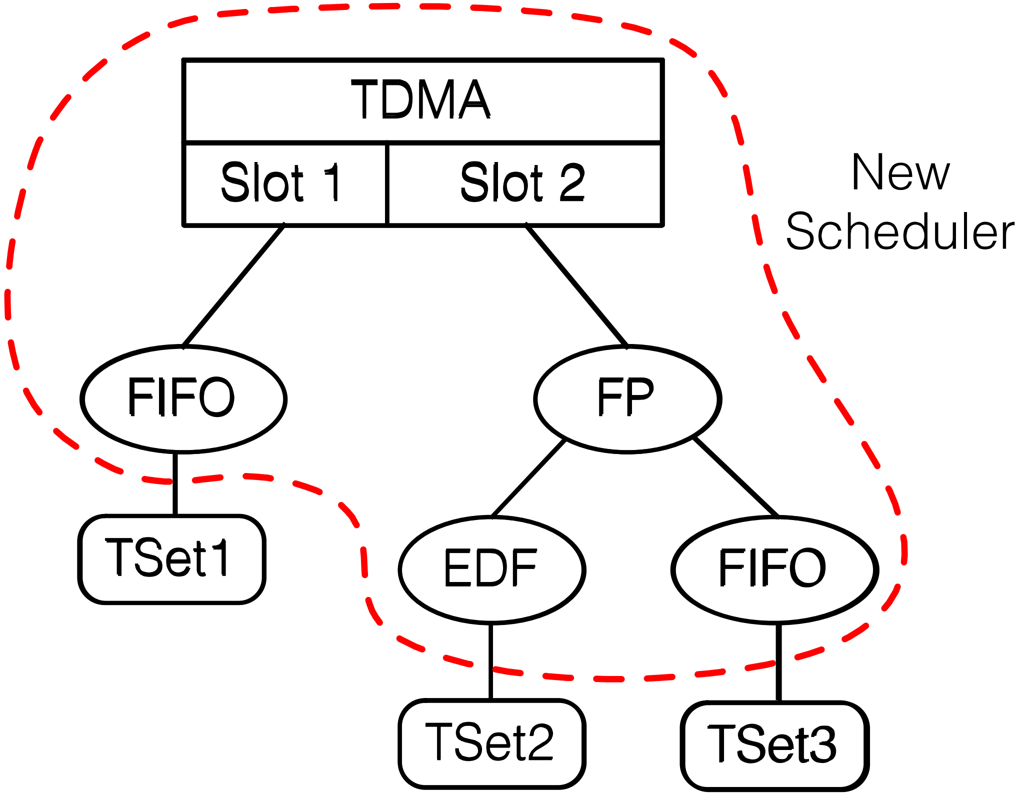


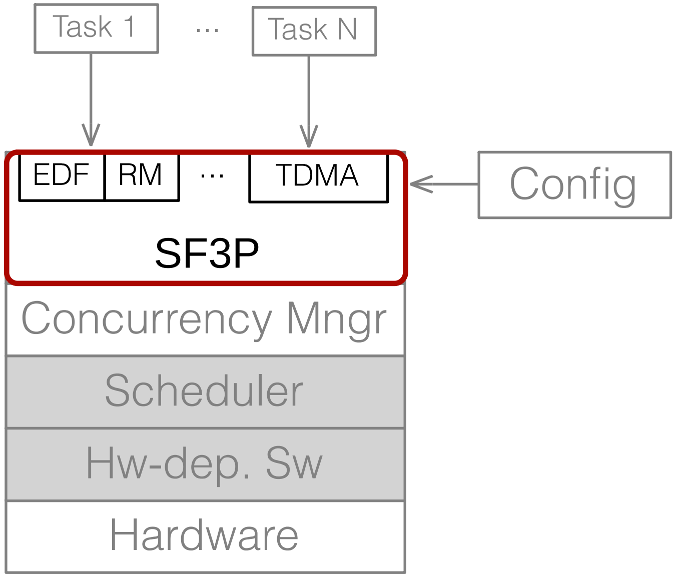
- Decoupled Insertion



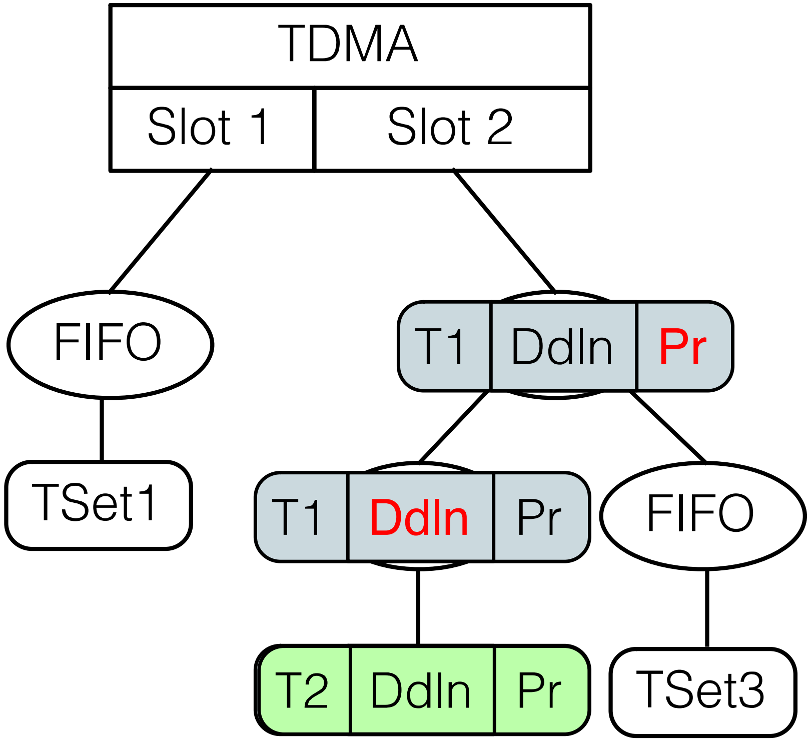
- Implemented: FIFO, FP, EDF, RM, TDMA

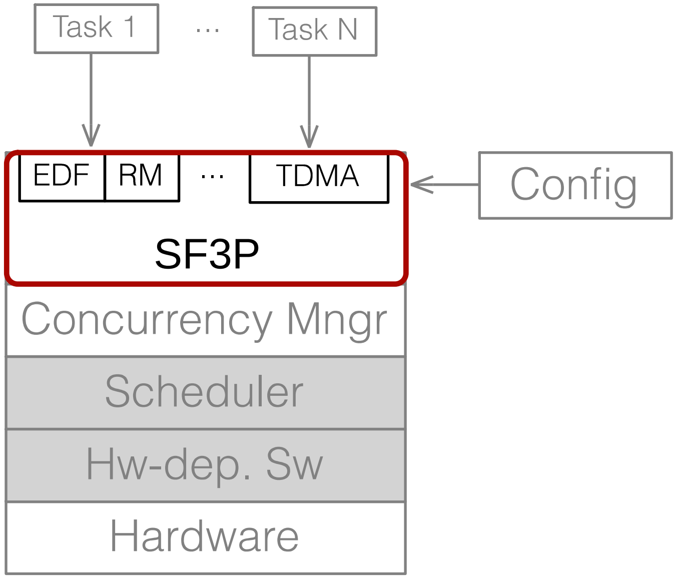
More Hierarchical Scheduling





Criteria Inheritance





Our Goals

- We can add a scheduling layer in the User Space

1. Portable to different platforms with no cost ✓
2. Extendable to new schedulers with low cost ✓
3. Low Overhead

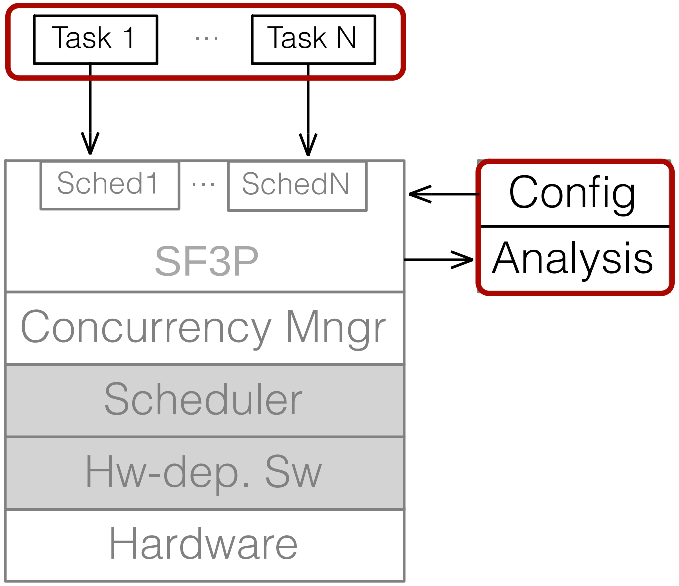
Our Goals

- We can add a scheduling layer in the User Space

1. Portable to different platforms with no cost ✓
2. Extendable to new schedulers with low cost ✓
3. Low Overhead

Evaluation Mechanism

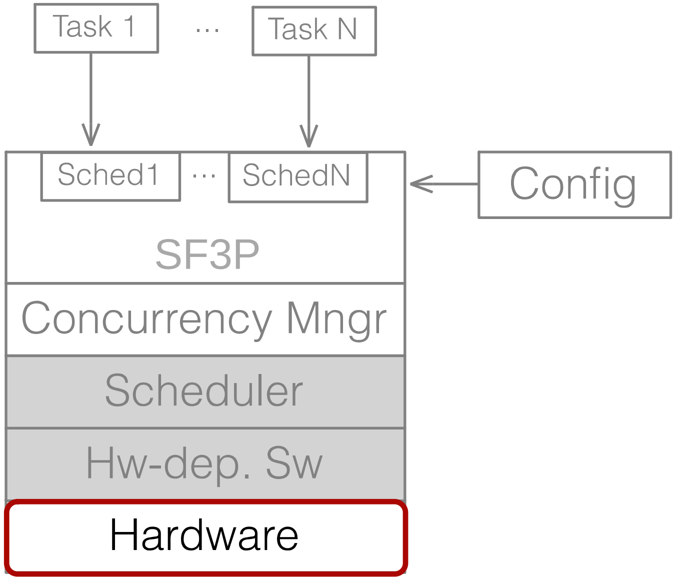
- Configuration File
  - Specify schedulers, tasks, criteria
- Dispatcher Library
  - Simulate task arrivals
- Analysis Tools
  - Calculate metrics



Experimental Evaluation

- Desktop Testing Environment

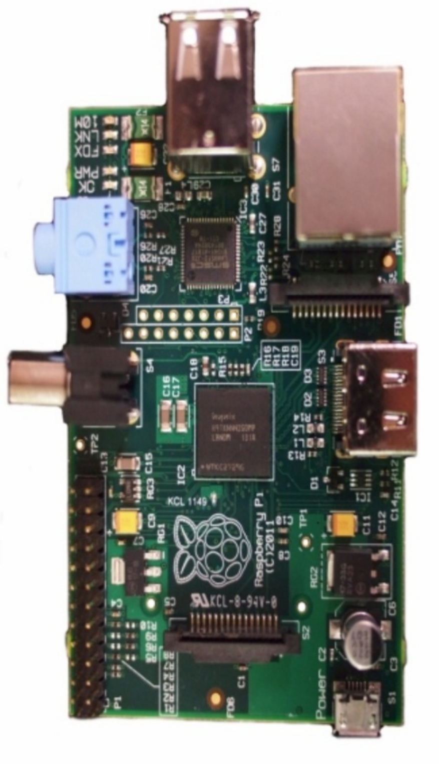
Linux Kernel:	3.2
Processor:	Intel i7 @ 3.4GHz
Memory:	16 GB RAM
Linux Runlevel:	1



Experimental Evaluation (II)

- Embedded Testing Environment (Raspberry Pi)

Linux Kernel:	2.6
Processor:	ARM V6 @ 700MHz
Memory:	512 MB RAM
Linux Runlevel:	1



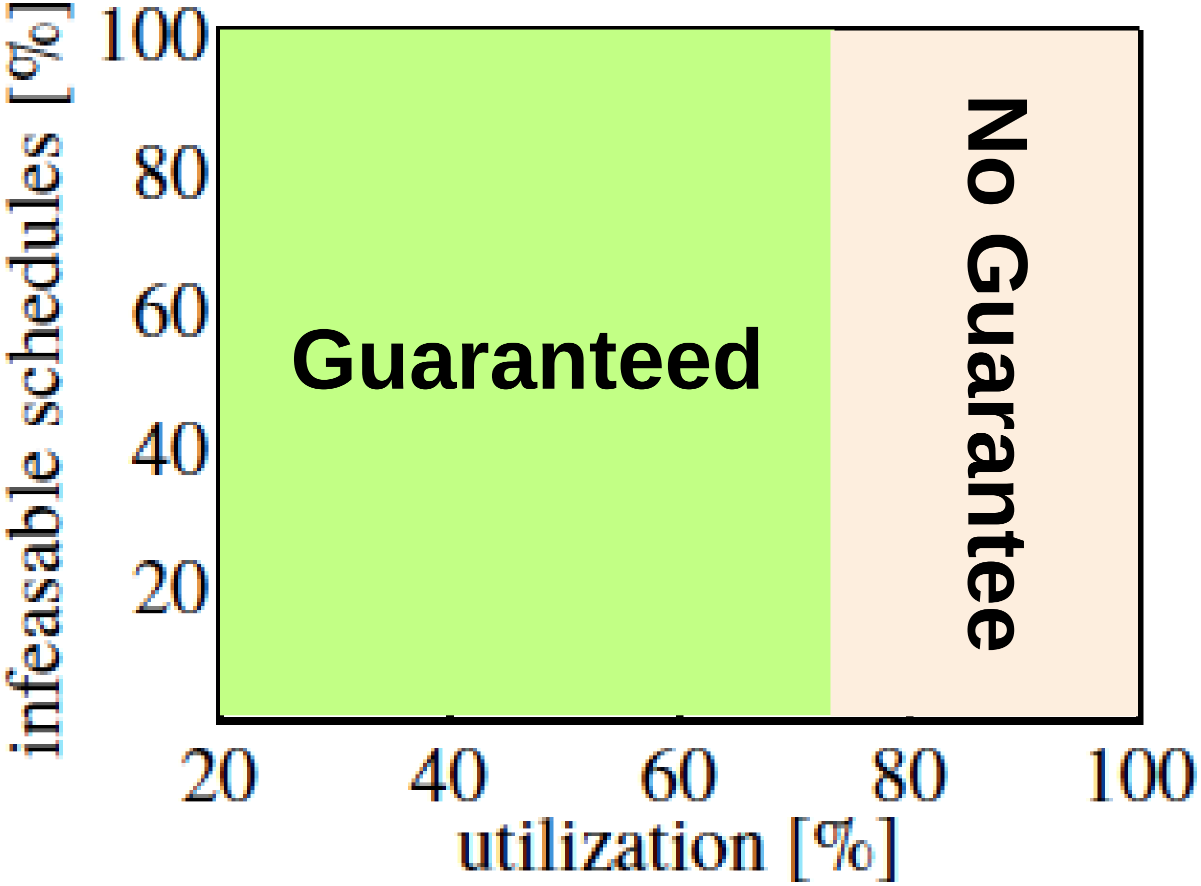
Schedulability Analysis

- A schedule is feasible if tasks meet **all** of their deadlines

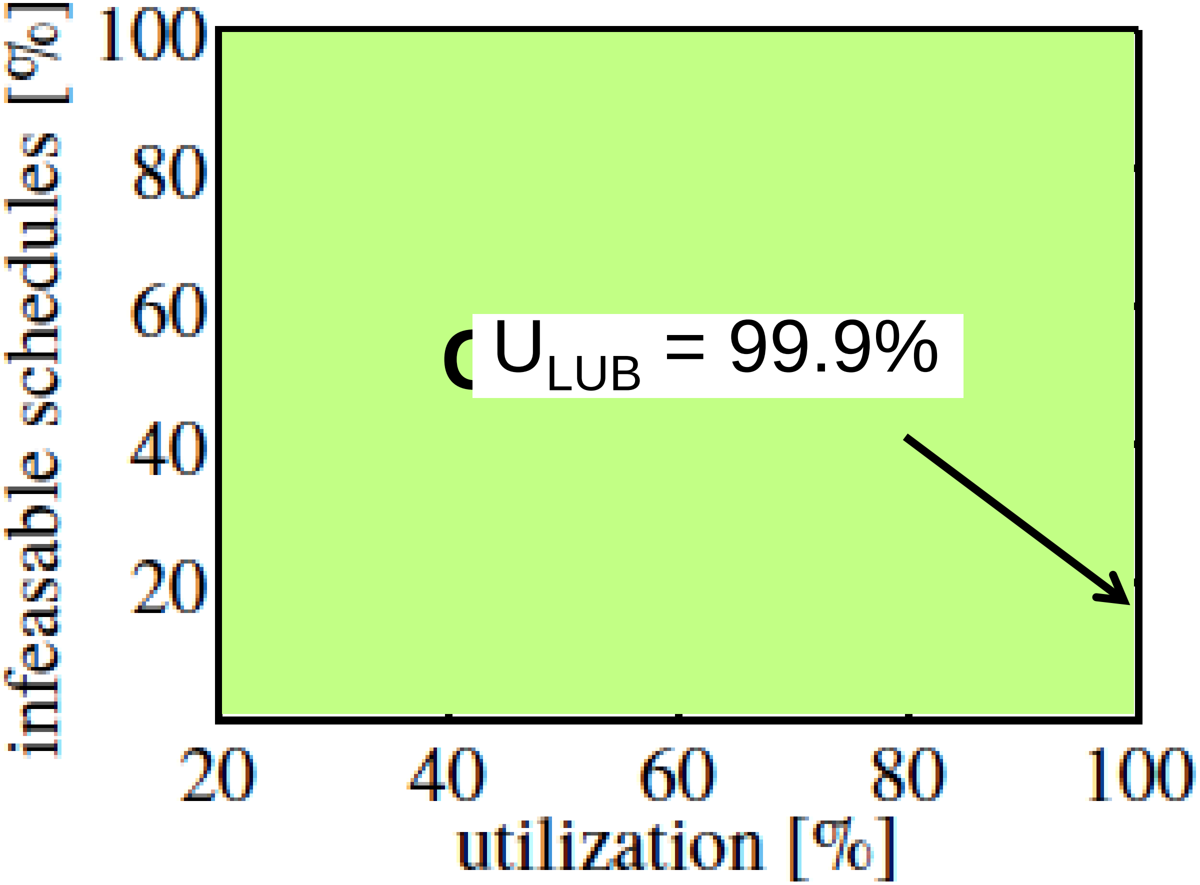
- In classical algorithms:
  - Utilization test $$U = \sum_i \frac{C_i}{T_i}$$
  - If $U < U_{LUB}$ then the schedule is feasible

- Generate (random) schedules and verify feasibility
  - $N_{tasks} \in [5,50]$ $U \in [20,100]\%$
  - $C_{long} \in [40,50]ms$ $C_{short} \in [5,10]ms$

Rate Monotonic Schedulability (Desktop)

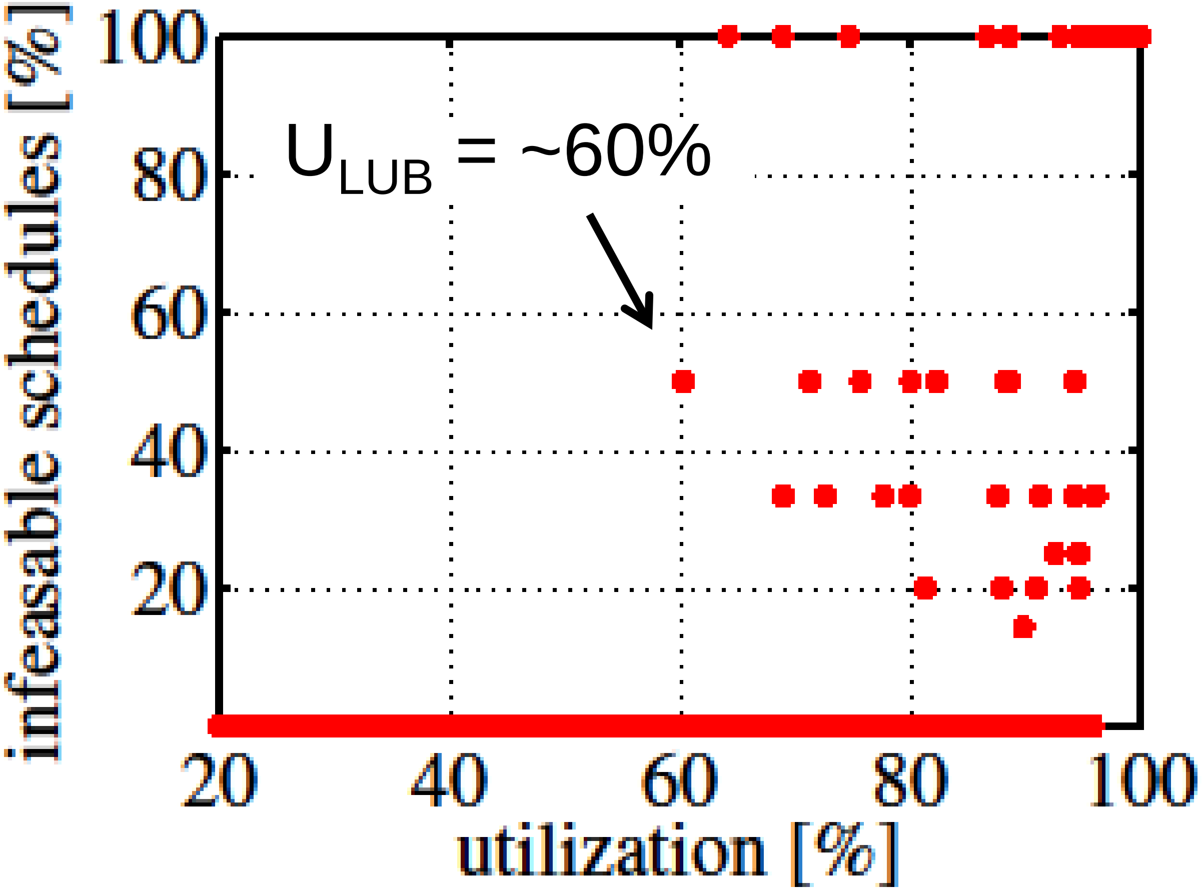


EDF Schedulability (Desktop)



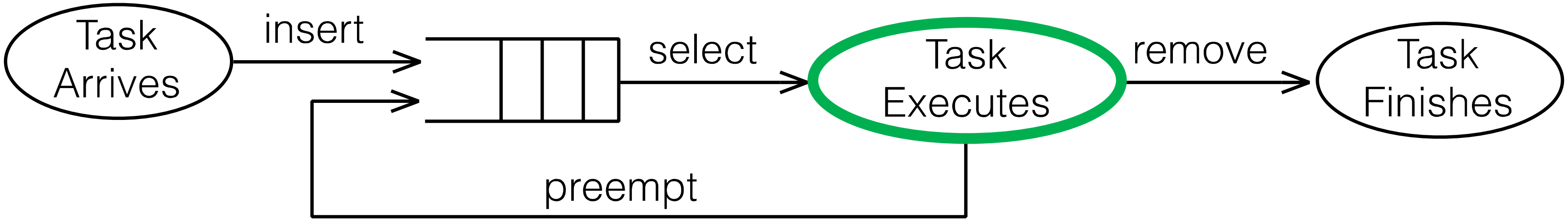
U_{LUB} = 99.9%

EDF Schedulability (RPI)



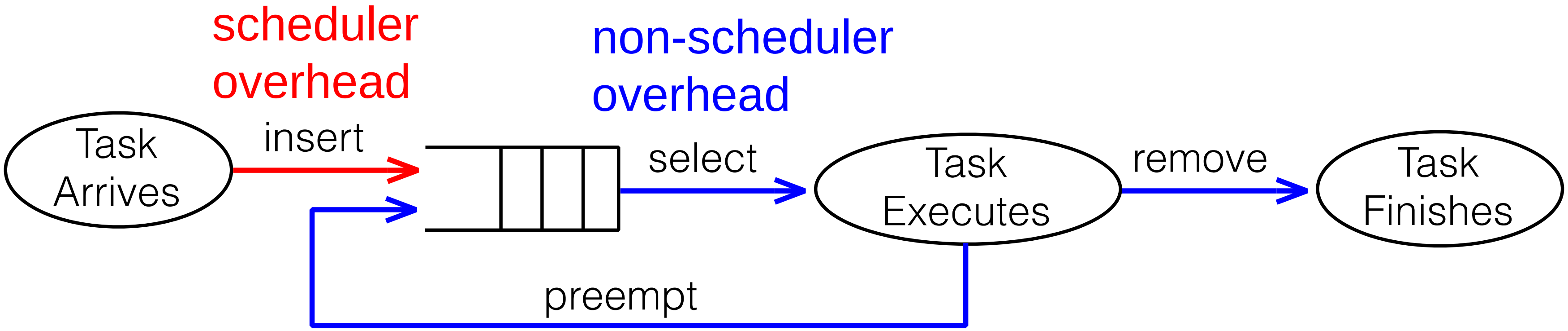
U_{LUB} = ~60%

SF3P Overhead



Overhead: time spent executing *anything* other than tasks

SF3P Overhead



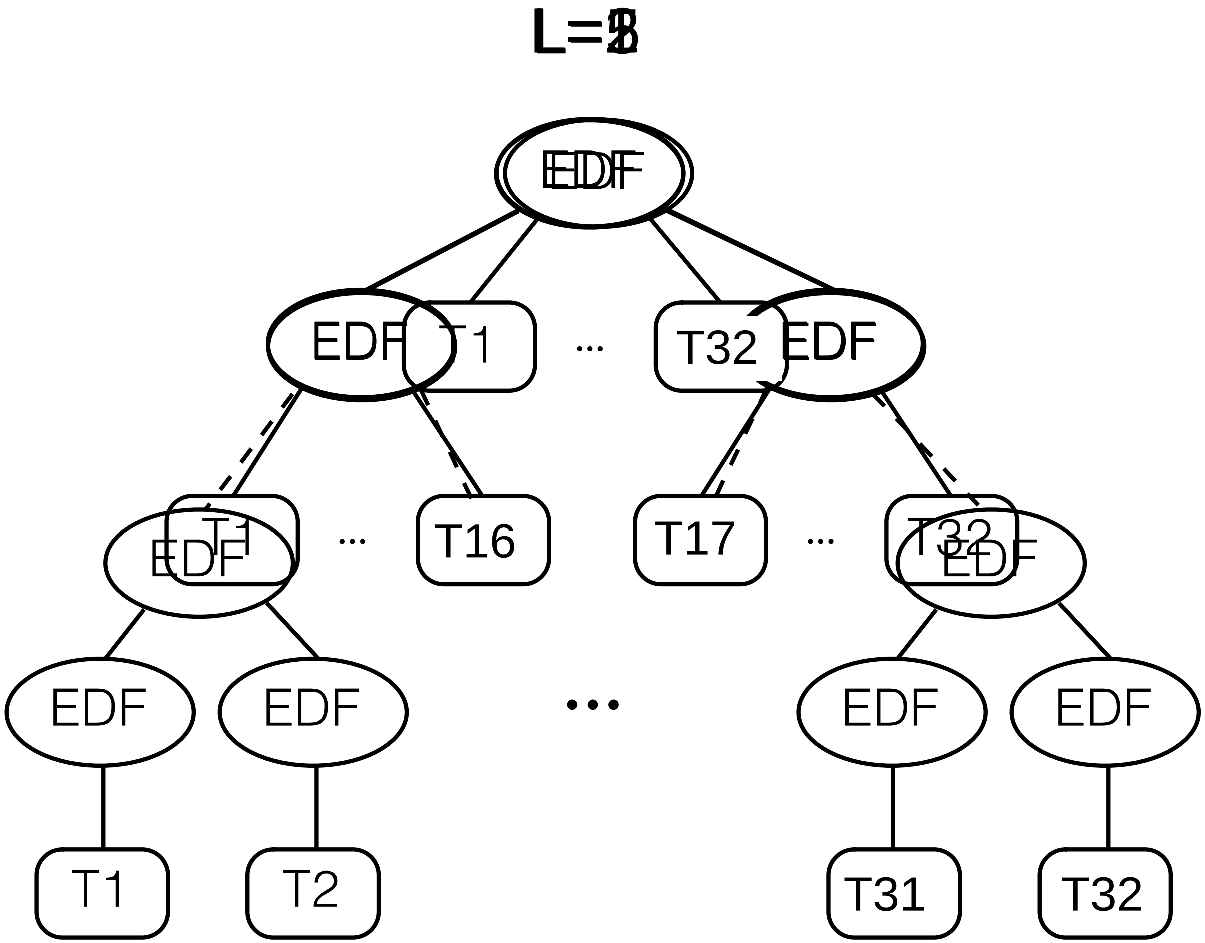
Scheduler Overhead

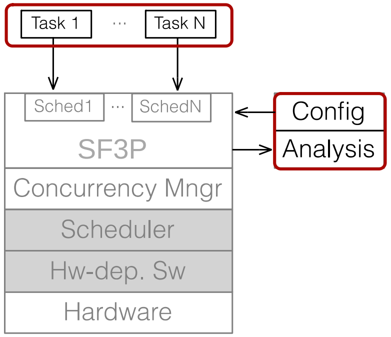
- Algorithm-dependent

Non-Scheduler Overhead

- Platform-dependent

Increasing the Levels of Hierarchy (L)





$$N = 32$$

$$L \in [1,5]$$

$$U \in [50,90] \quad \%$$

$$C \in [10,40] \text{ ms}$$

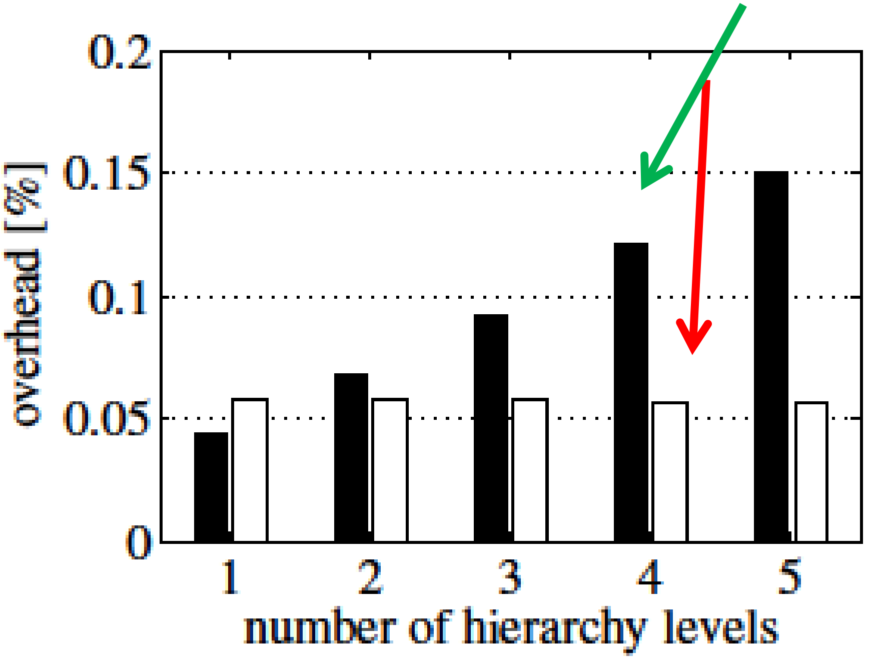
Overhead vs Levels of Hierarchy

■ Scheduler overhead

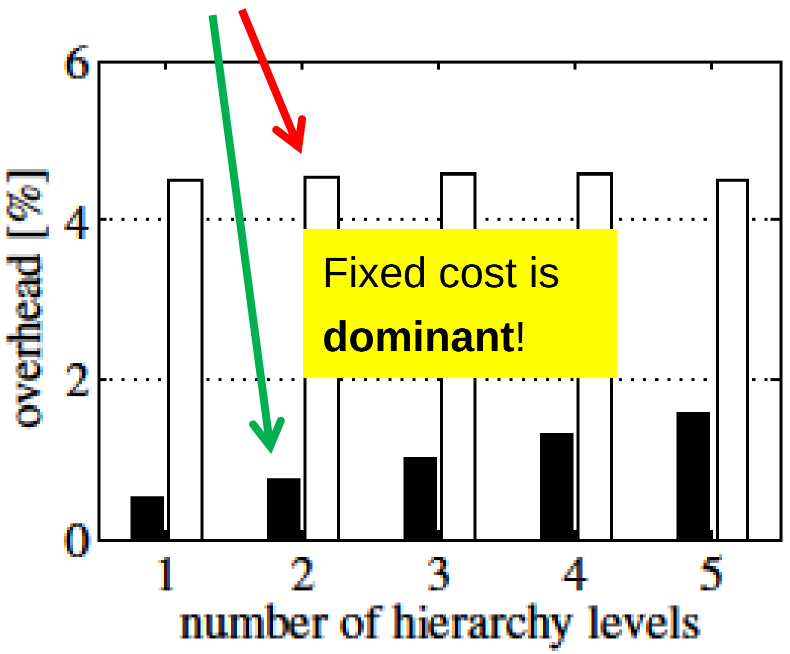
□ Non Scheduler overhead

Scheduler overhead increases **linearly!**

Non Sched. Overhead is constant!



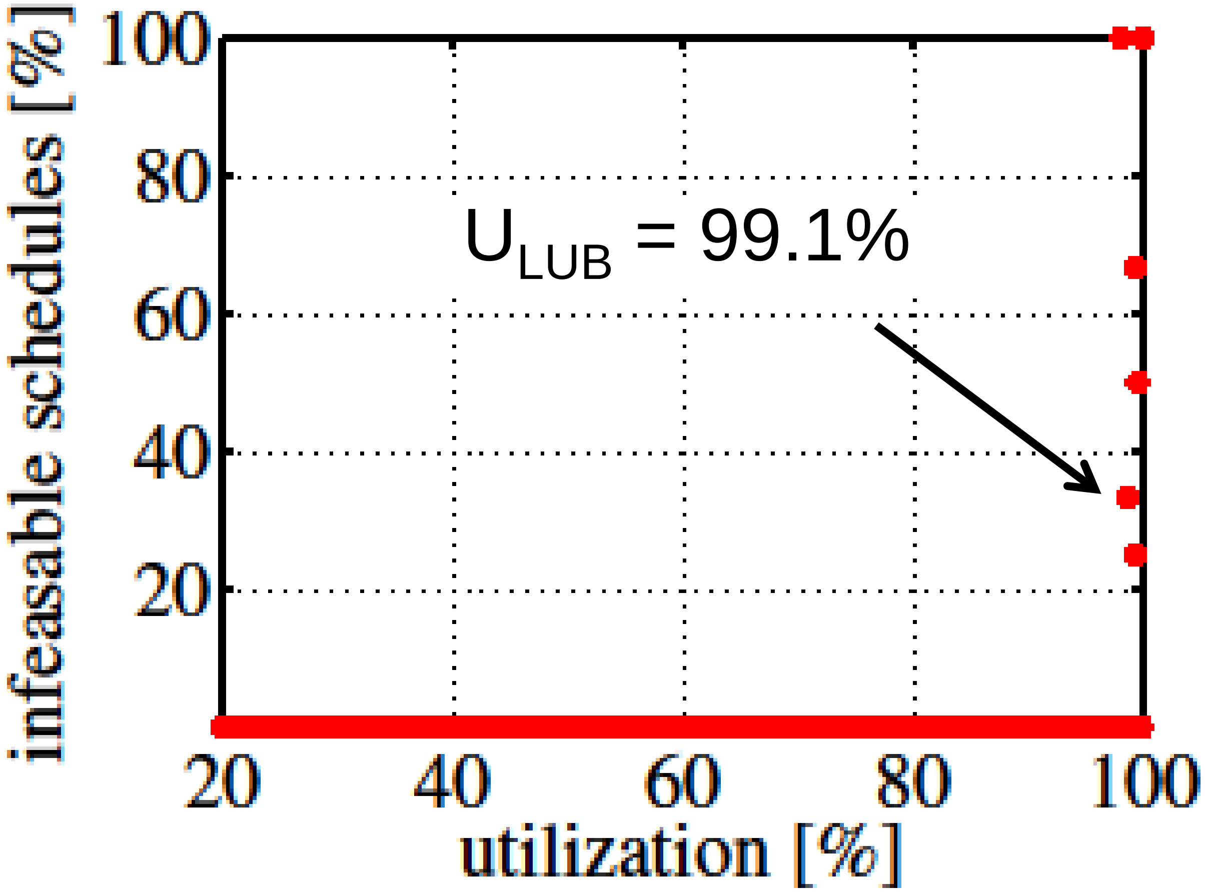
Desktop



Fixed cost is **dominant**!

RPI

Re-running EDF with long (10x) Tasks on RPI



U_{LUB} = 99.1%

Our Goals

- We can add a scheduling layer in the User Space

1. Portable to different platforms with no cost ✓
2. Extendable to new schedulers with low cost ✓
3. Low Overhead ✓

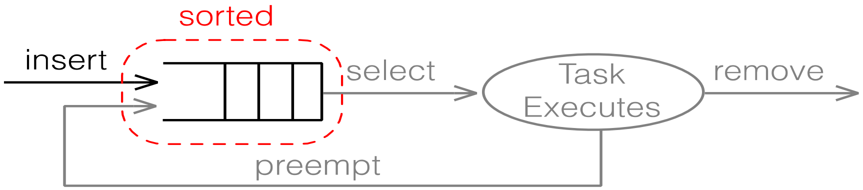
Our Goals

- We can add a scheduling layer in the User Space

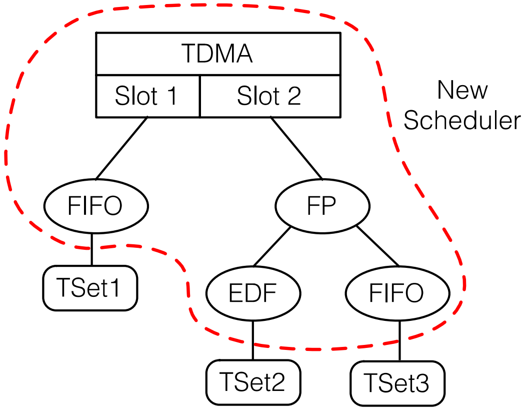
1. Portable to different platforms with no cost ✓
2. Extendable to new schedulers with low cost ✓
3. Low Overhead ✓

SF3P Summary

- Framework for fast prototyping of real-time schedulers
  - Modular, extendable, composable



- New hierarchical schedulers
  - Suitable for complex scheduling needs
- Low overhead



Available at: http://www.tik.ee.ethz.ch/~euretile/scheduling