Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Grupo de Investigación en Arquitectura de Computadores (gaZ)
Universidad Zaragoza

HiPEAC COMPILATION ARCHITECTURE
European Network on High Performance and Embedded Architecture and Compilation

Exposing Abstraction-Level Interactions with a Parallel Ray Tracer

Alejandro Valero*, Darío Suárez Gracia*, Rubén Gran Tejero*, Luis M.Ramos, **Agustín Navarro-TorresΦ**, Adolfo Muñoz, Joaquín Ezpeleta, José Luis Briz, Ana C. Murillo, Eduardo Montijano, Javier Resano, María Villarroya-Gaudó, Jesús Alastruey-Benedé, Enrique Torres, Pedro Álvarez, Pablo Ibáñez, and Víctor Viñals

* Corresponding authors: {alvabre,dario,rgran}@unizar.es
Φ **agusnt@unizar.es**

22/06/2019

Outline

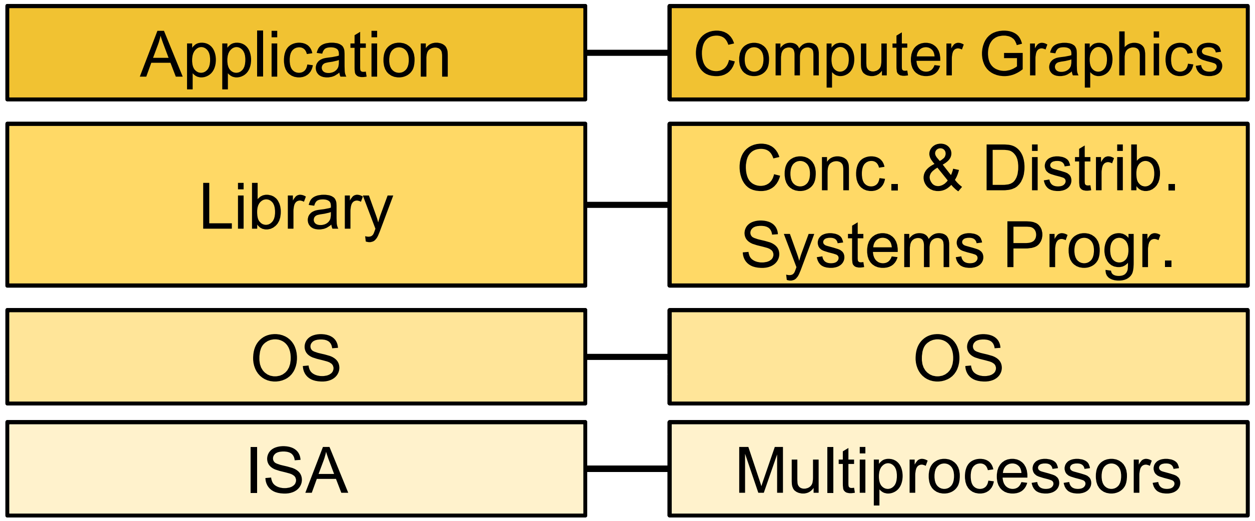
- Motivation and Overview
- Context
- Cross-Cutting Project
- Experimental Environment & Results
- Conclusions and Future Work

Outline

- **Motivation and Overview**
- Context
- Cross-Cutting Project
- Experimental Environment & Results
- Conclusions and Future Work

Motivation and Overview

- **Abstraction boundaries**

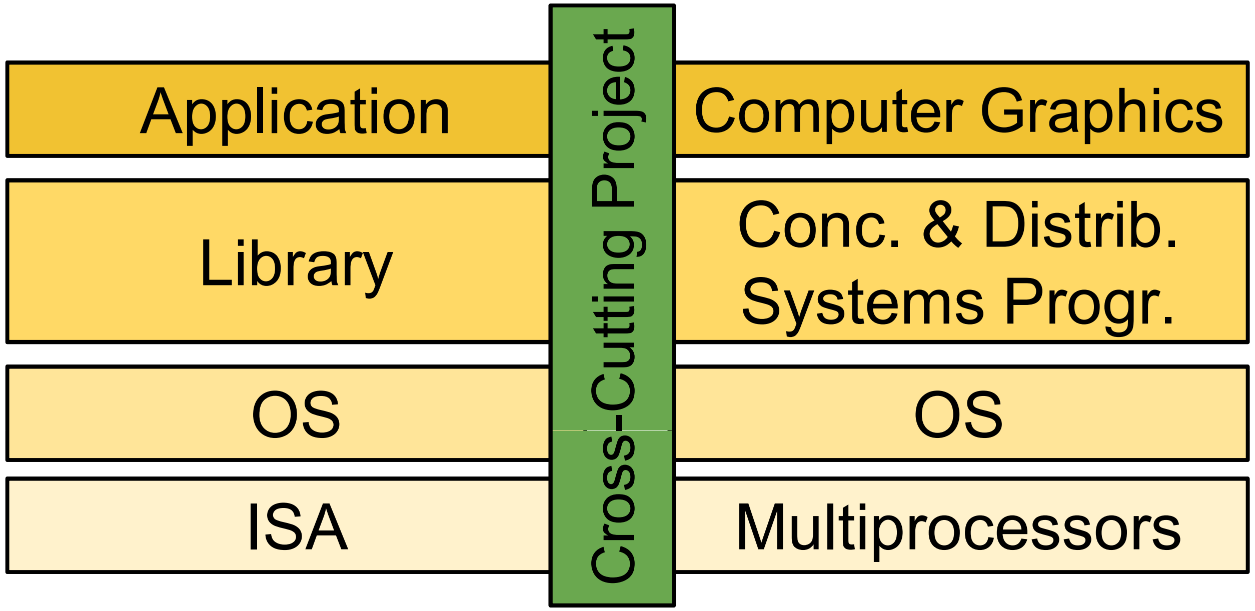


- **Computer Engineering (CE)**: establish boundaries across courses
  - Strengthen the learning process
  - Lost the overall vision of a computer system

- **Goal**: educate professionals and researchers with a global vision

Motivation and Overview

- **Abstraction boundaries**



- **Computer Engineering (CE)**: establish boundaries across courses
  - Strengthen the learning process
  - Lost the overall vision of a computer system

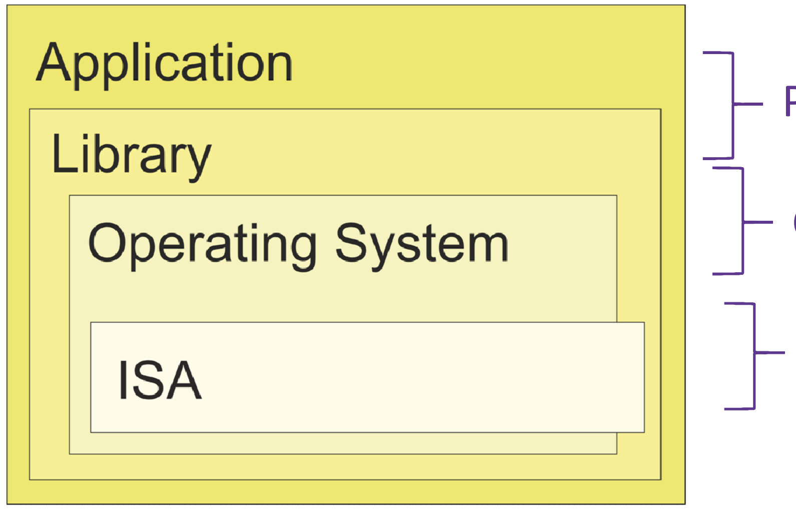
- **Goal**: educate professionals and researchers with a global vision

Motivation and Overview

- **Proposal**: solve a problem with shared resources and cover multiples levels of a computer system
  - Atomicity, consistency, parallelism & concurrency

- **Case of study**: implementation of a parallel ray-tracing algorithm that uses a concurrent queue to assign tasks

Requirements:



Parallel ray-tracing → concurrent queue

Concurrent queue → OS system calls

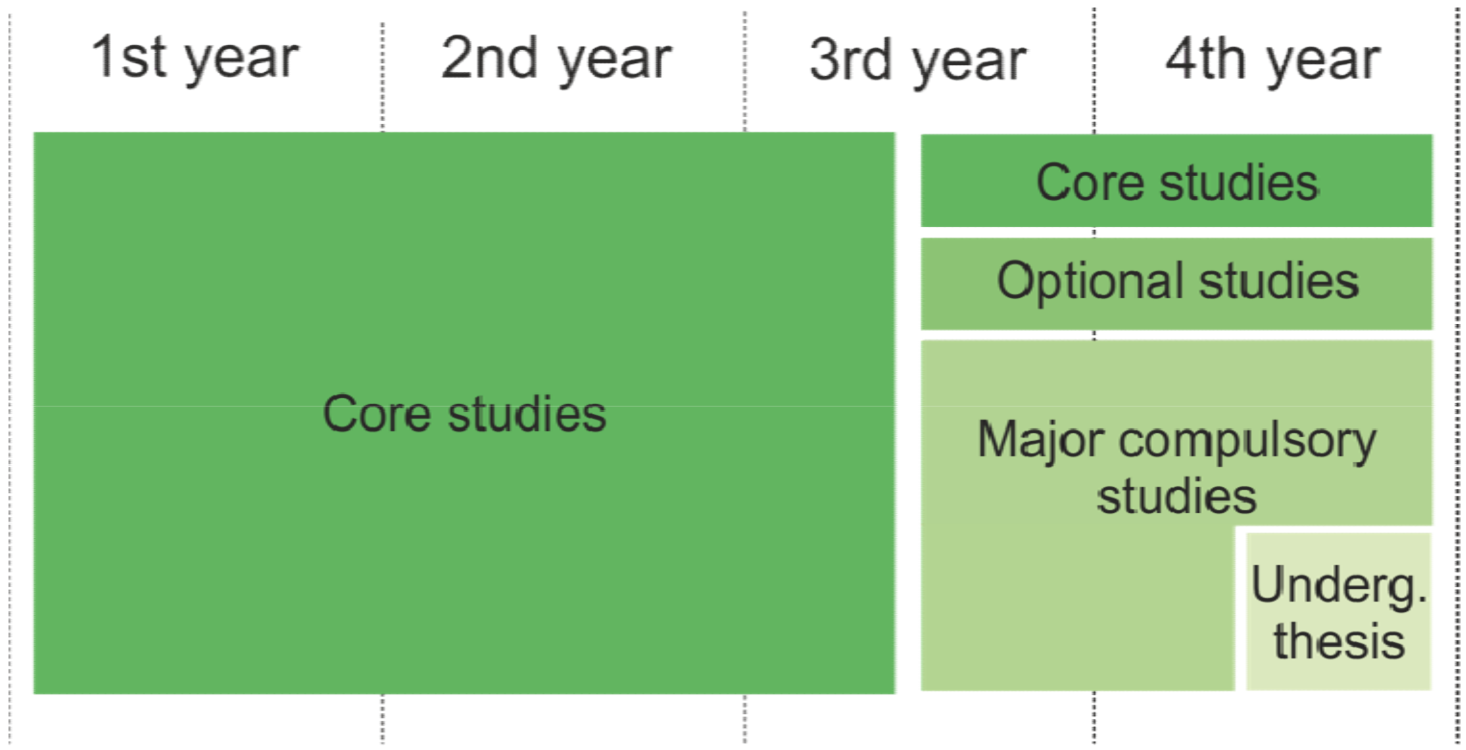
OS system calls → synchronization & atomic instructions

Outline

- Motivation and Overview
- **Context**
- Cross-Cutting project
- Experimental Environment & Results
- Conclusions and Future Work

Context

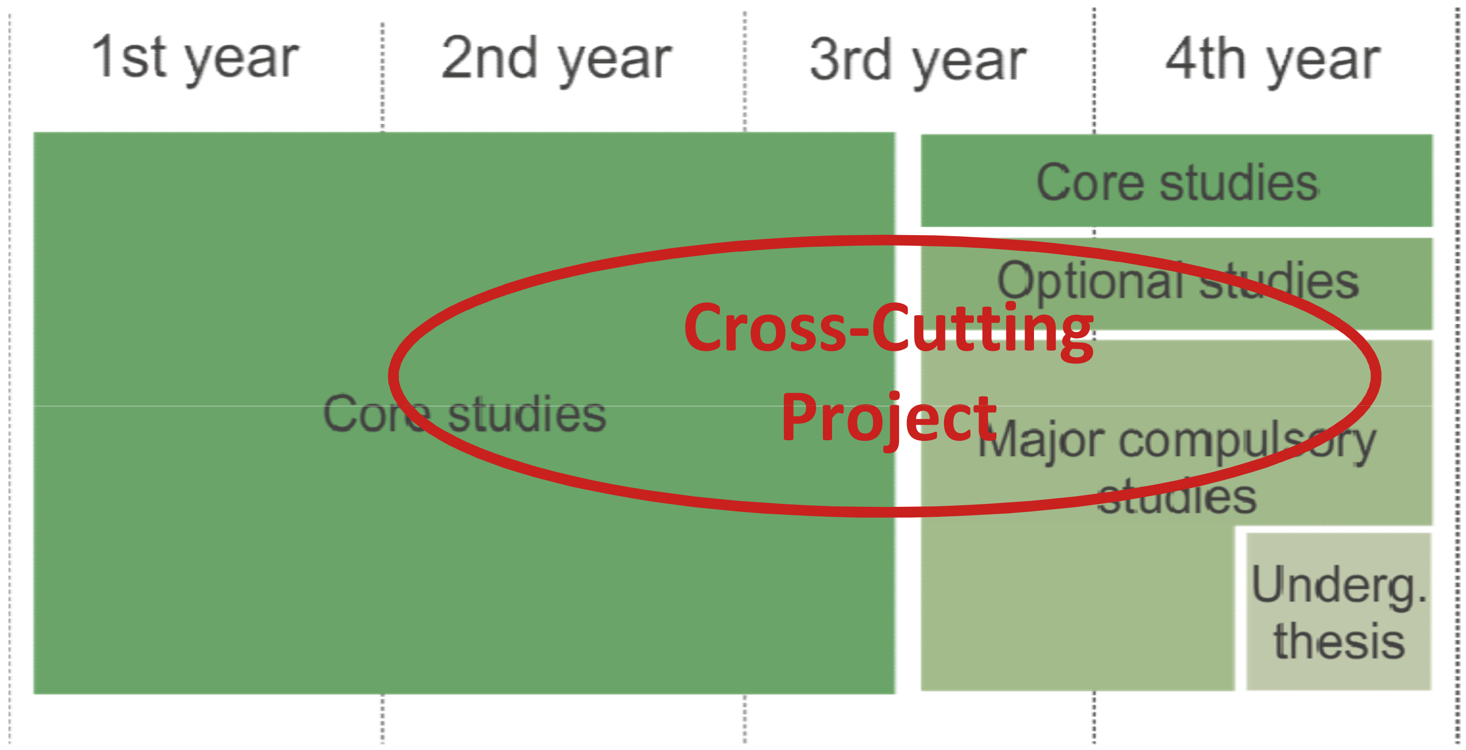
Computer Engineering Program at the University of Zaragoza



- Five Majors:
  - Computer Science
  - Computer Engineering
  - Information Systems
  - Information Technology
  - Software Engineering

Context

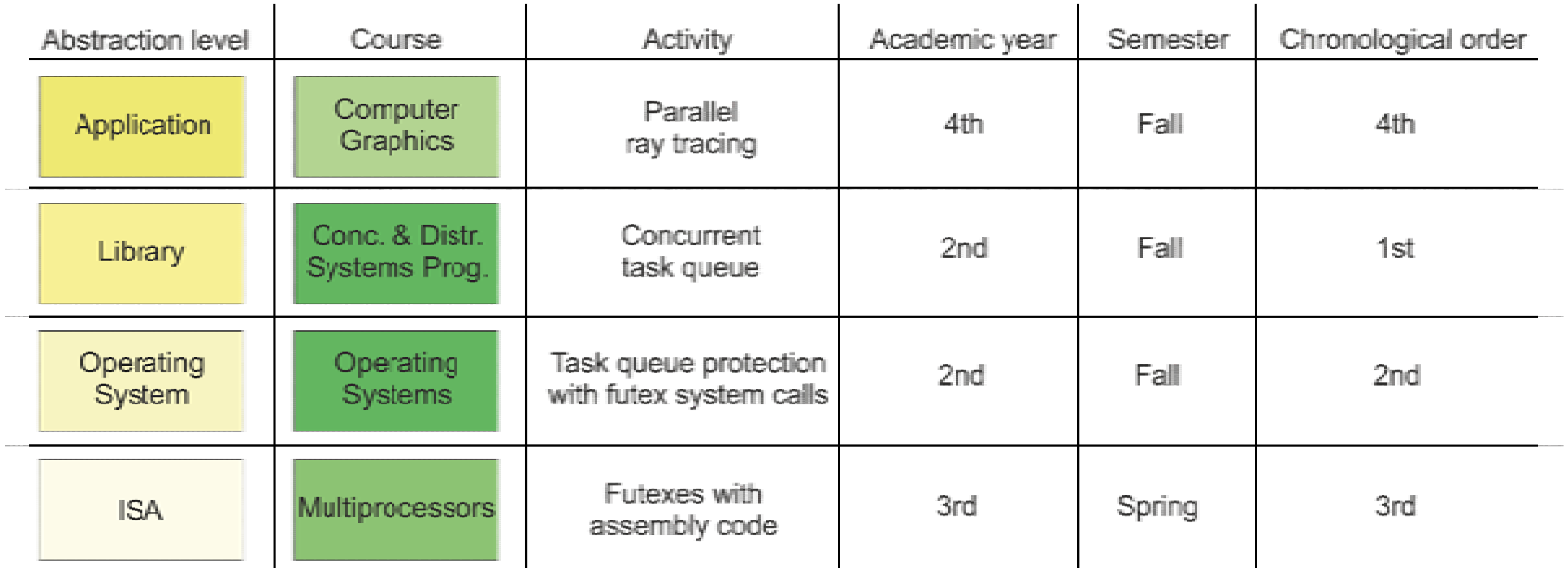
Computer Engineering Program at the University of Zaragoza



4 courses across the last 3 years of the Program

Outline

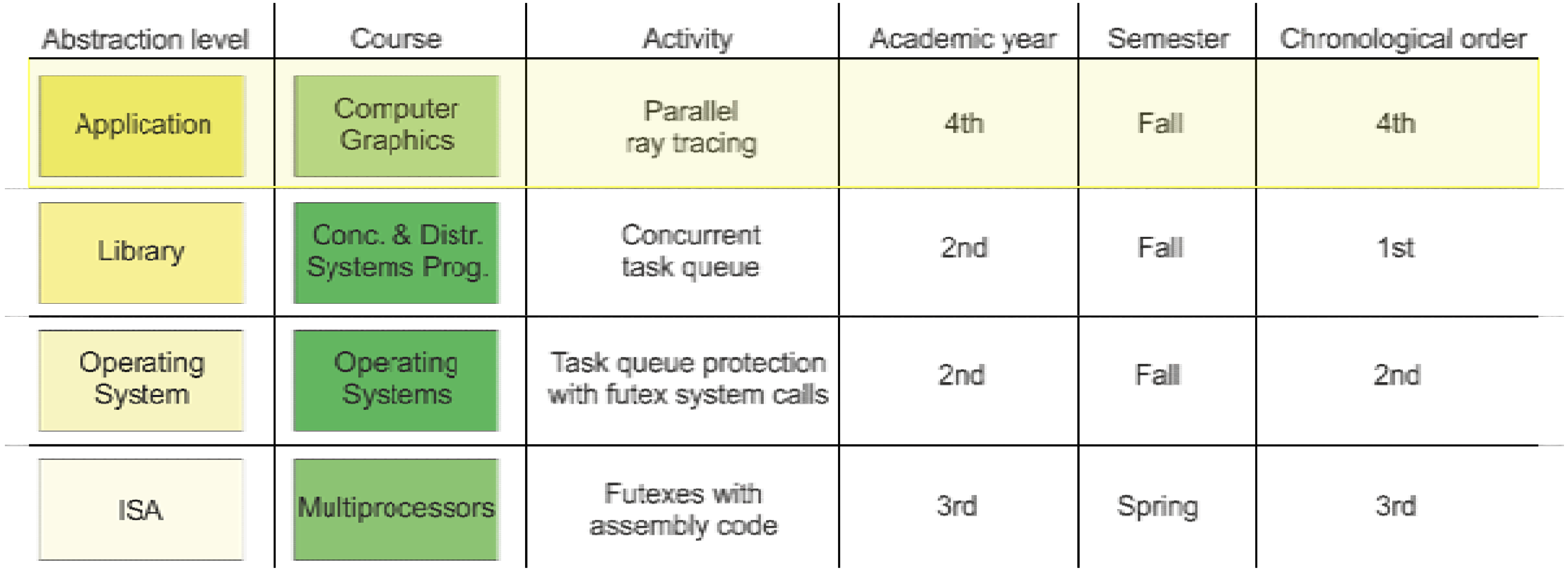
Project Overview



Abstraction level	Course	Activity	Academic year	Semester	Chronological order
Application	Computer Graphics	Parallel ray tracing	4th	Fall	4th
Library	Conc. & Distr. Systems Prog.	Concurrent task queue	2nd	Fall	1st
Operating System	Operating Systems	Task queue protection with futex system calls	2nd	Fall	2nd
ISA	Multiprocessors	Futexes with assembly code	3rd	Spring	3rd

2-hour laboratories to implement each level and relate it with the other levels

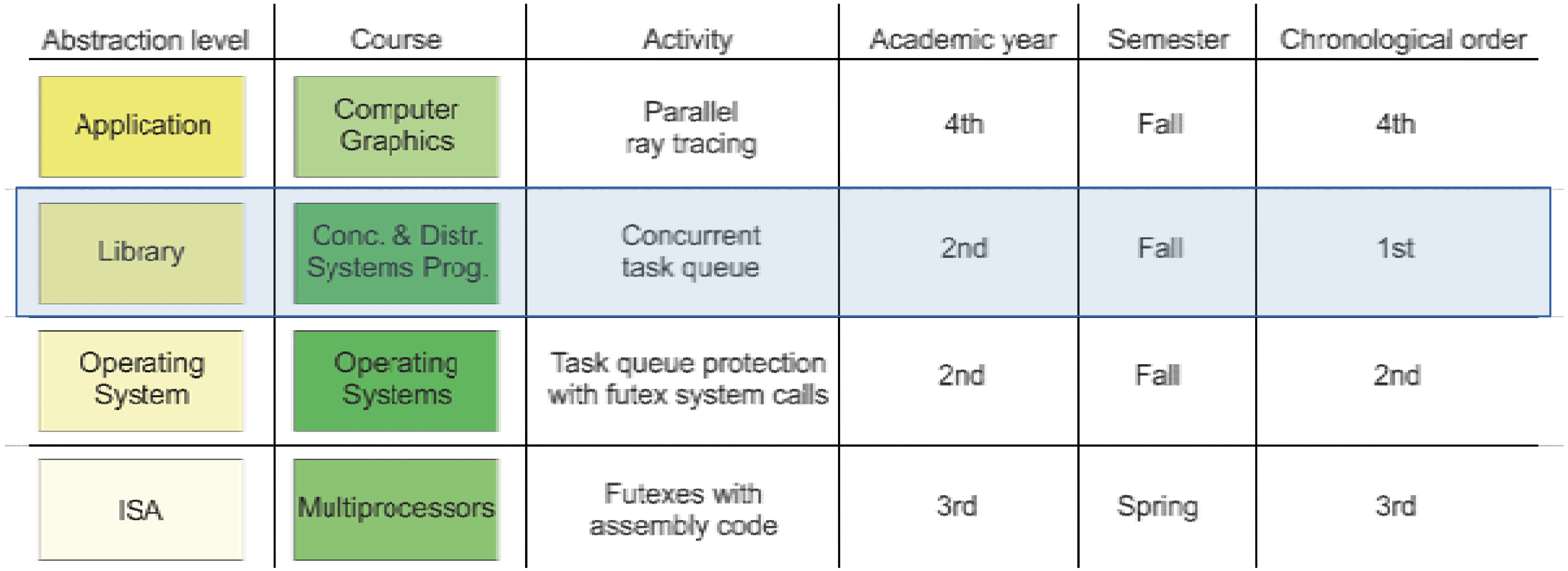
Project Overview



Abstraction level	Course	Activity	Academic year	Semester	Chronological order
Application	Computer Graphics	Parallel ray tracing	4th	Fall	4th
Library	Conc. & Distr. Systems Prog.	Concurrent task queue	2nd	Fall	1st
Operating System	Operating Systems	Task queue protection with futex system calls	2nd	Fall	2nd
ISA	Multiprocessors	Futexes with assembly code	3rd	Spring	3rd

Parallel path-tracing by assigning different tasks to execution threads using a concurrent task queue

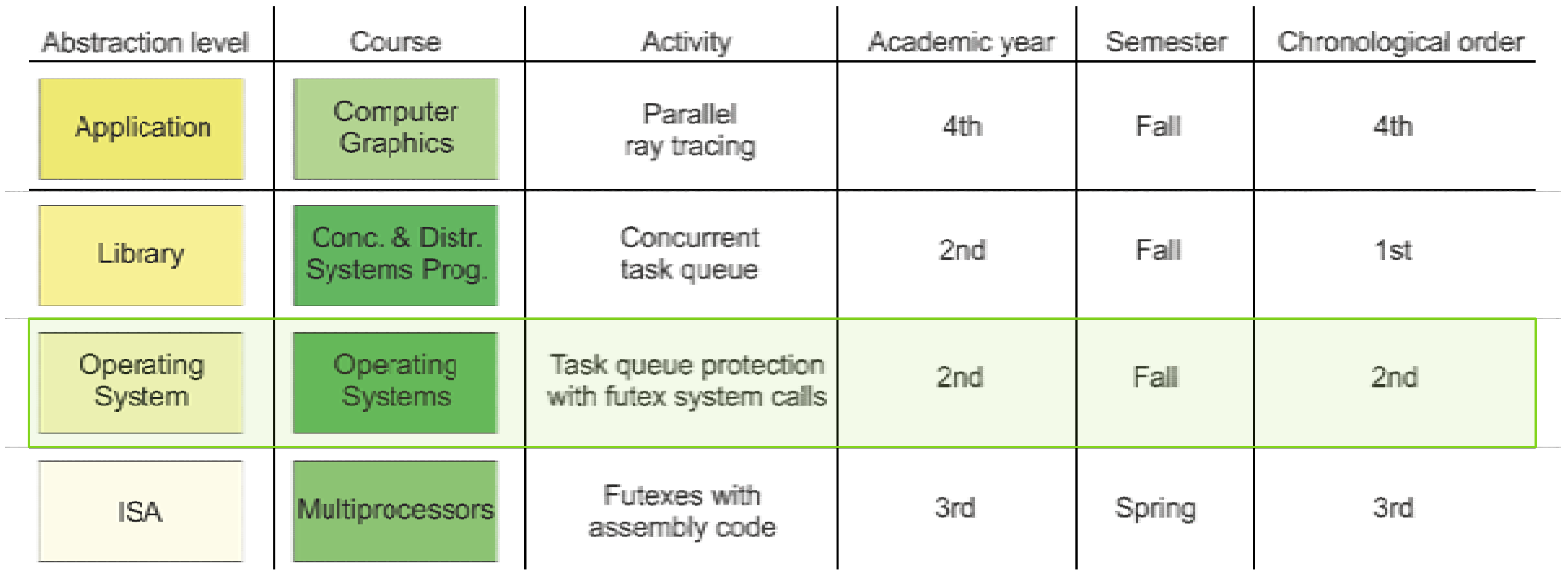
Project Overview



Abstraction level	Course	Activity	Academic year	Semester	Chronological order
Application	Computer Graphics	Parallel ray tracing	4th	Fall	4th
Library	Conc. & Distr. Systems Prog.	Concurrent task queue	2nd	Fall	1st
Operating System	Operating Systems	Task queue protection with futex system calls	2nd	Fall	2nd
ISA	Multiprocessors	Futexes with assembly code	3rd	Spring	3rd

Implementation of a concurrent queue with mutual exclusion among the execution threads, in order to preserve data integrity

Project Overview



Abstraction level	Course	Activity	Academic year	Semester	Chronological order
Application	Computer Graphics	Parallel ray tracing	4th	Fall	4th
Library	Conc. & Distr. Systems Prog.	Concurrent task queue	2nd	Fall	1st
Operating System	Operating Systems	Task queue protection with futex system calls	2nd	Fall	2nd
ISA	Multiprocessors	Futexes with assembly code	3rd	Spring	3rd

Implementation of a concurrent queue with futex System Calls

Project Overview

Abstraction level	Course	Activity	Academic year	Semester	Chronological order
Application	Computer Graphics	Parallel ray tracing	4th	Fall	4th
Library	Conc. & Distr. Systems Prog.	Concurrent task queue	2nd	Fall	1st
Operating System	Operating Systems	Task queue protection with futex system calls	2nd	Fall	2nd
ISA	Multiprocessors	Futexes with assembly code	3rd	Spring	3rd

Implementation of futexes with assembly code

Project Overview

Abstraction level	Course	Activity	Academic year	Semester	Chronological order
Application	Computer Graphics	Parallel ray tracing	4th	Fall	4th
Library	Conc. & Distr. Systems Prog.	Concurrent task queue	2nd	Fall	1st
Operating System	Operating Systems	Task queue protection with futex system calls	2nd	Fall	2nd
ISA	Multiprocessors	Futexes with assembly code	3rd	Spring	3rd

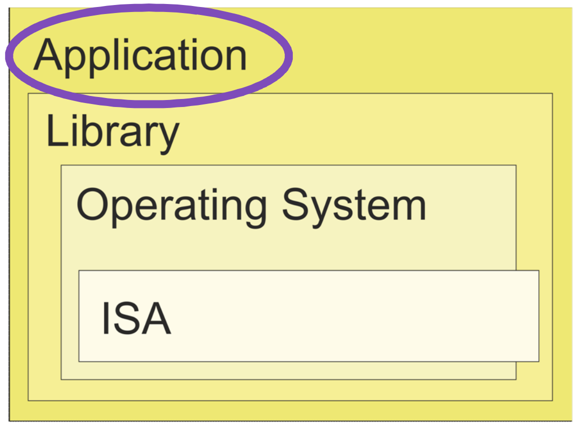
The chronological distribution of the courses demands the following order:

Library → OS → ISA → Application

Application: Ray-Tracing

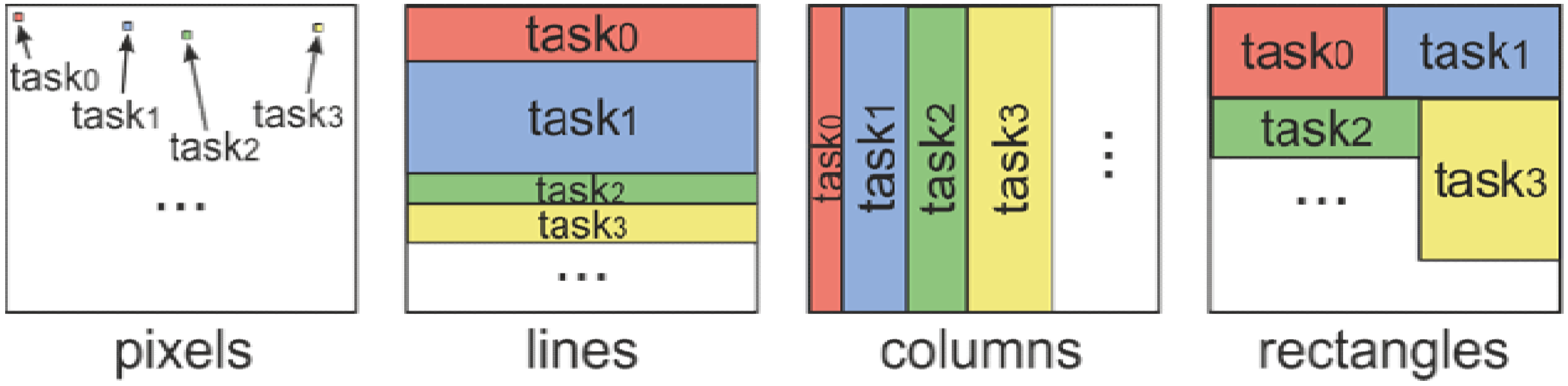
- Development and implementation of a ray-tracing from a pin-hole camera

3D Representation → 2D Image

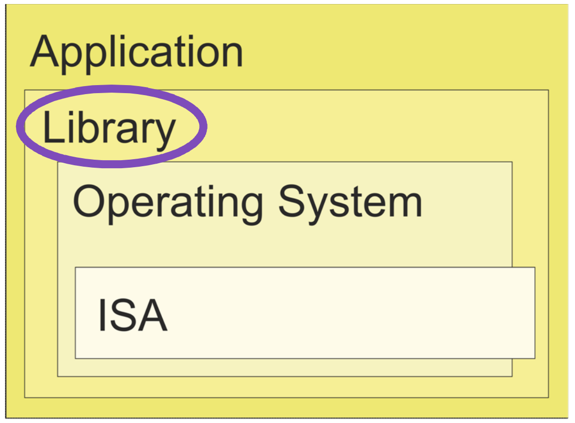


The color pixel generation is independent → Parallelism

Workload Distribution → Split image in sections:



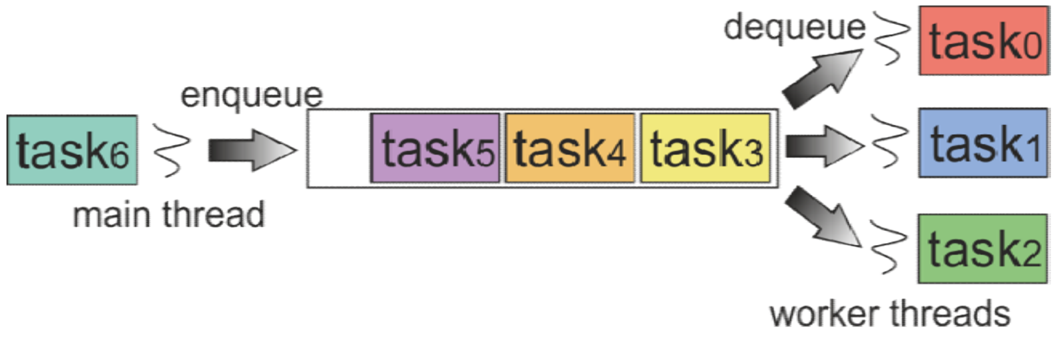
Library: Concurrent Queue



- Implement a size fixed queue which allows concurrent access for data insertion and extraction

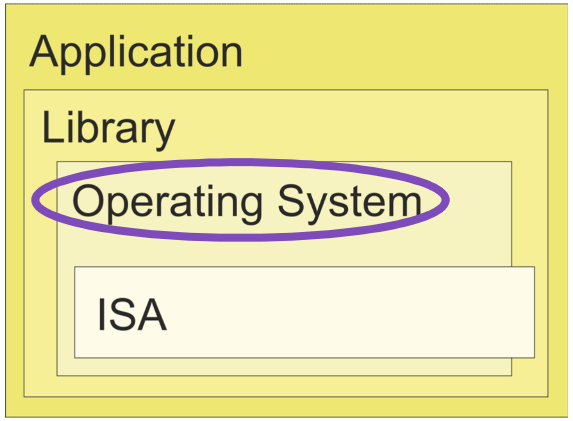
- Fundamental synchronization aspects:
  - Mutual Exclusion
  - Conditional Waiting

- Use *pthread* library: `pthread_mutex_init, pthread_mutex_lock, pthread_mutex_unlock, pthread_mutex_destroy`

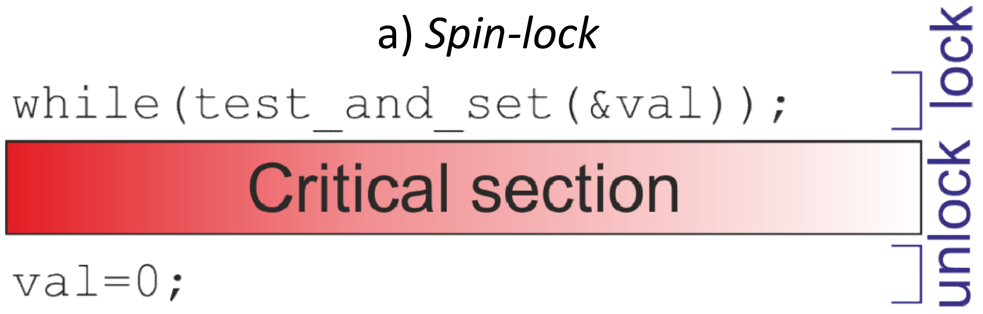


OS: Mutual Exclusion with System Calls

- Implementation of a synchronization mechanism
- U.Drepper Mutex lock and unlock [1]
- Multiple mutex versions:
  - Spin-lock
  - Basic sleep
  - Advanced sleep



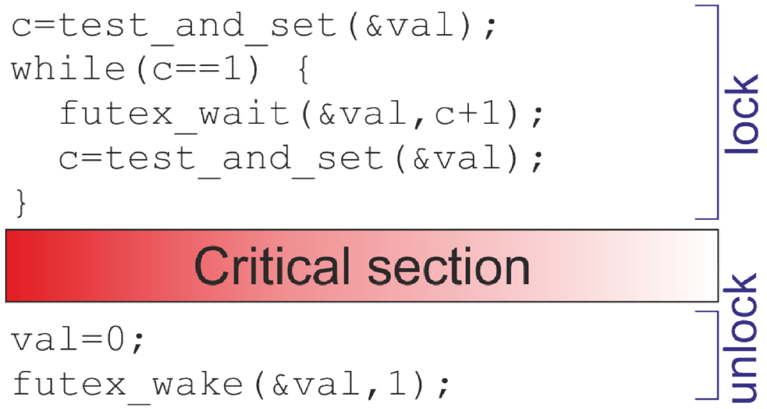
a) *Spin-lock*



```
while(test_and_set(&val));
```

```
val=0;
```

b) *Basic sleep*



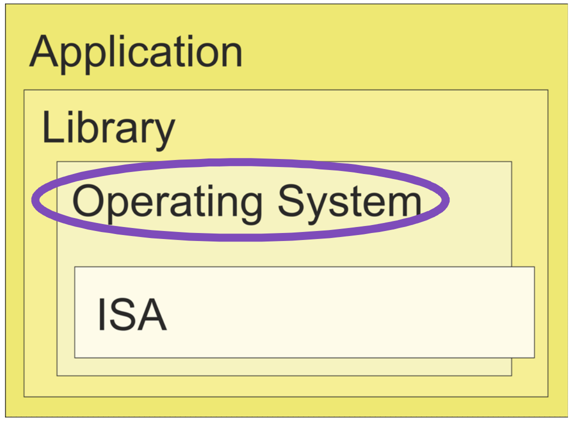
```
c=test_and_set(&val);
while(c==1) {
  futex_wait(&val,c+1);
  c=test_and_set(&val);
}
```

```
val=0;
futex_wake(&val,1);
```

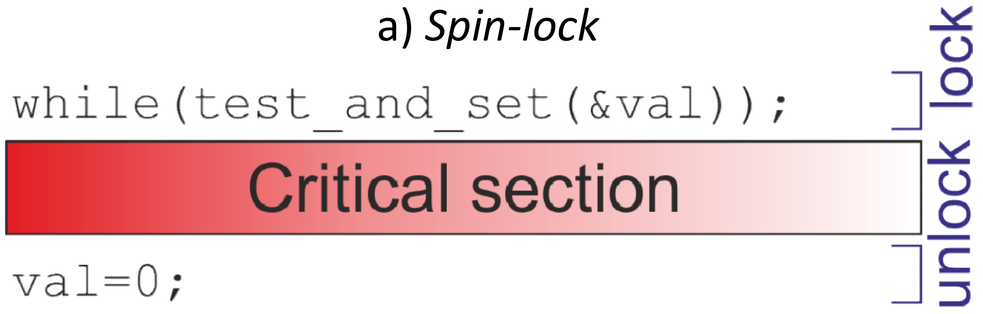
[1] U. Drepper, “Futexes Are Tricky”, 2011, http://people.redhat.com/drepper/futex.pdf

OS: Mutual Exclusion with System Calls

- Implementation of a synchronization mechanism
- U.Drepper Mutex lock and unlock [1]
- Multiple mutex versions:
    - Spin-lock
    - Basic sleep
    - Advanced sleep



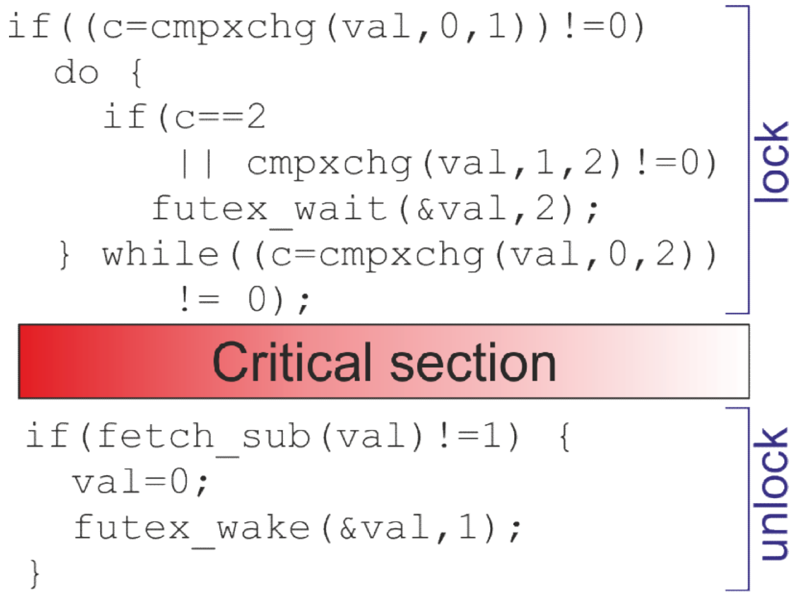
a) *Spin-lock*



```
while(test_and_set(&val));
```

```
val=0;
```

c) *Advanced sleep*



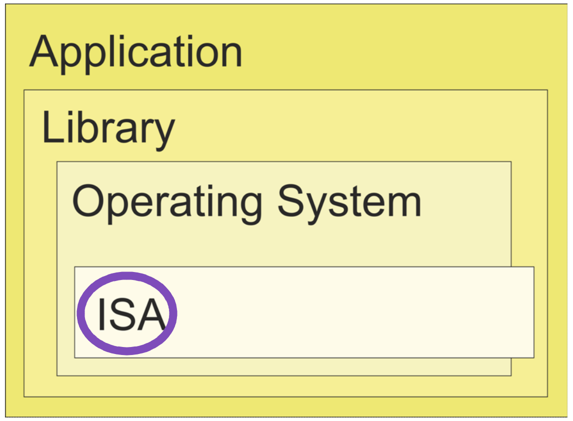
```
if((c=cmpxchg(val,0,1))!=0)
  do {
    if(c==2
       || cmpxchg(val,1,2)!=0)
      futex_wait(&val,2);
  } while((c=cmpxchg(val,0,2))
       != 0);
```

```
if(fetch_sub(val)!=1) {
  val=0;
  futex_wake(&val,1);
}
```

[1] U. Drepper, “Futexes Are Tricky”, 2011, http://people.redhat.com/drepper/futex.pdf

ISA: Synchronization & Atomicity Instructions

- Avoid System Calls overhead using ISA primitives from the user level
- Relations between high and low memory models
- ARMv8: *lock/unlock*
- Two versions:



a) *Spin-Lock*

```
loop:
    ldaxr   w2,  [@lock]
    cbnz    w2,  loop
    mov     w3,  #1
    stxr    w4,  w3,  [@lock]
    cbnz    w4,  loop
```

lock

Critical section

```
    stlr    wzr,  [@lock]
```

unlock

b) *Sleep mode*

```
    sevl
loop:
    wfe
    ldaxr   w2,  [@lock]
    cbnz    w2,  loop
    mov     w3,  #1
    stxr    w4,  w3,  [@lock]
    cbnz    w4,  loop
```

lock

Critical section

```
    stlr    wzr,  [@lock]
```

unlock

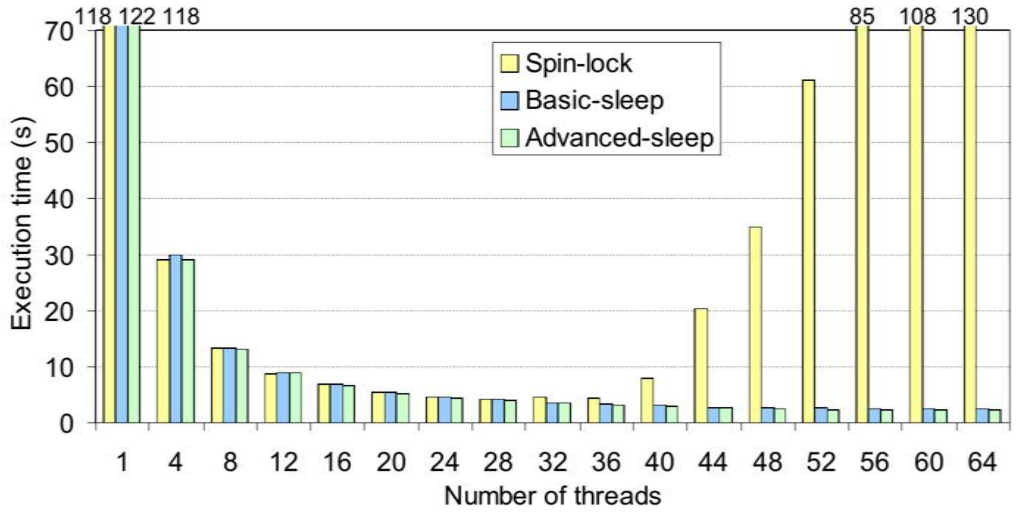
Outline

- Motivation and Overview
- Context
- Cross-Cutting Project
- **Experimental Environment & Results**
- Conclusions and Future Work

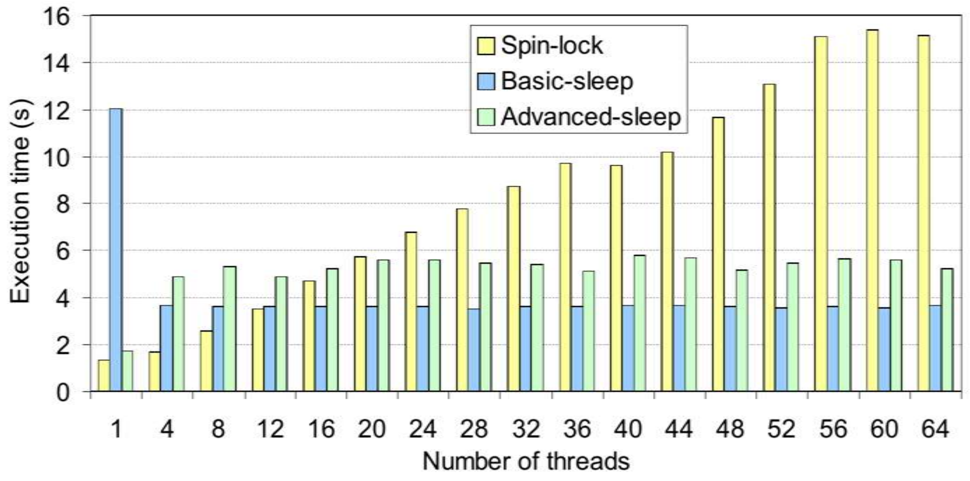
Experimental Environment

Type	Description	**Raspberry Pi**	DragonBoard	HiKey	BeagleBoard
H & S	Multiprocessor	✔	✔	✔	×
H	JTAG	~	×	×	✔
S	High Level O.S support	✔	✔	✔	✔
H & S	Bare metal	✔	×	✔	✔
H	Virtualization	✔	✔	✔	×
H & S	Low cost	✔	×	×	×

Experimental Results



Real contention



Synthetic contention

Student Learning Outcomes

- A new OS lab with 15% of the student's class (volunteers)
- Before the lab, 83% of the students perceive that the OS course strongly relates to computer architecture and parallel and distributed computing
- After the lab, this percentage rises to 92%
- The overall score of the lab was 4.42 (of 5)

Outline

Conclusions & Future Work

- The current structure of Computer Engineering program causes students to lose sight of the overall view of a computer system

- We present a cross-cutting project that covers multiple abstraction levels
  - Ray-Tracing
    - Application – Computer Graphics
    - Library – Distributed and Concurrent Systems Programming
    - OS – Operating Systems
    - ISA – Multiprocessors

- Feedback received by the students is encouraging

- Involve more courses and students

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Grupo de Investigación en Arquitectura de Computadores (gaZ)
Universidad Zaragoza

HiPEAC COMPILATION ARCHITECTURE
European Network on High Performance and Embedded Architecture and Compilation

Exposing Abstraction-Level Interactions with a Parallel Ray Tracer

Alejandro Valero*, Darío Suárez Gracia*, Rubén Gran Tejero*, Luis M.Ramos, **Agustín Navarro-TorresΦ**, Adolfo Muñoz, Joaquín Ezpeleta, José Luis Briz, Ana C. Murillo, Eduardo Montijano, Javier Resano, María Villarroya-Gaudó, Jesús Alastruey-Benedé, Enrique Torres, Pedro Álvarez, Pablo Ibáñez, and Víctor Viñals

* Corresponding authors: {alvabre,dario,rgran}@unizar.es
Φ **agusnt@unizar.es**

22/06/2019