YASA - A Framework for Validation, Test, and Analysis of Real-Time Scheduling Algorithms

Jan Blumenthal, Jens Hildebrandt, Frank Golatowski, and Dirk Timmermann

University of Rostock, Germany

5th Real-Time Linux Workshop

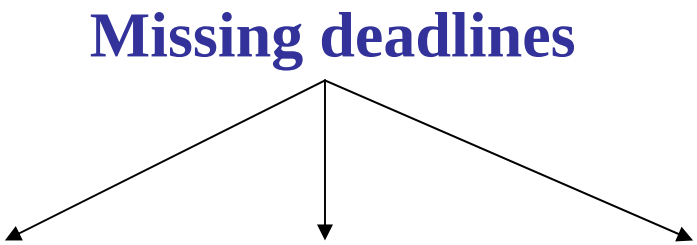
Valencia, 2003/11/13

Outline

- Preliminaries
- YASA Framework
  - Introduction & vision
  - Integration into target systems
  - Configuration & evaluation
- Conclusion

Design Goals in Real-Time Systems

Design Goals: - Proof of functionality
- Proof of timeliness



Economical disadvantage

Physical damage

Human injury

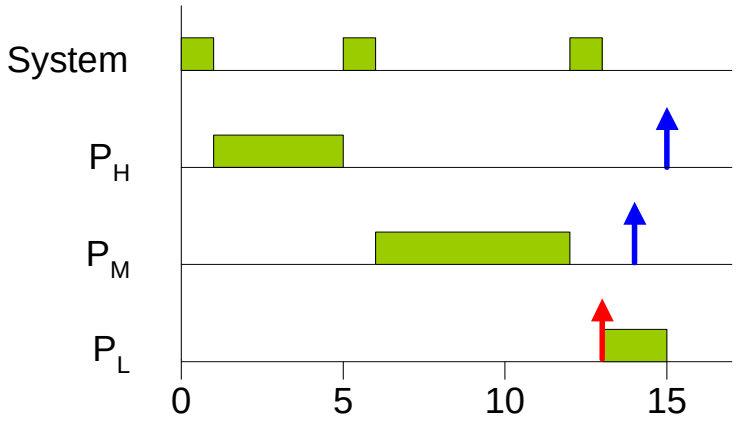
Our goal: Reduction of deadline misses

➢ Concentration on optimization of scheduling behavior

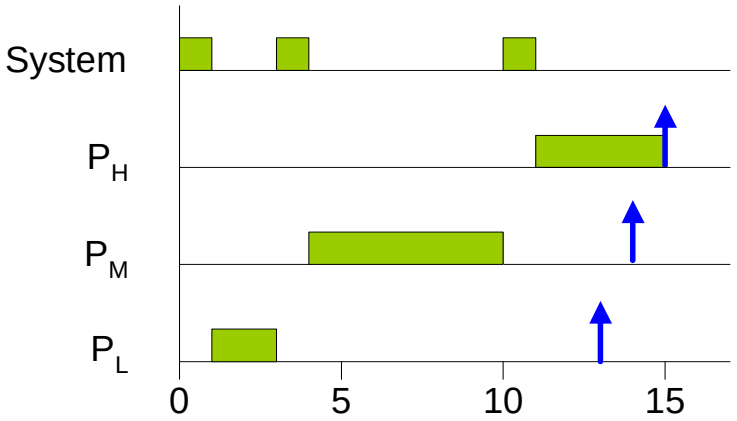
Scheduling

Task: Assignment of processes to CPU

Priority based:



Earliest Deadline First:



- **Analysis required**
- **Exchange of schedulers would be favorable**

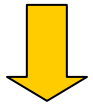
Synchronization Protocols

Semaphores:

- Protect critical areas
- Typical tasks:
  - Data exchange between processes
  - Request for resources

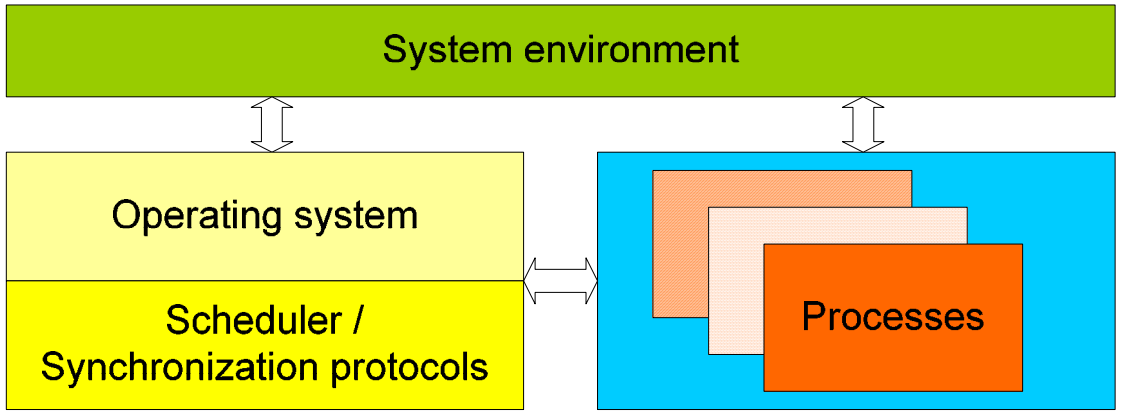
Synchronization protocols:

- Prevent while entering critical sections
  - Priority inversion
  - Deadlocks
- Often tightly coupled with schedulers



Exchange of synchronization protocols required, too !

Current Systems



- Operating system and system scheduler are tightly coupled
- Dynamic schedulers or synchronization protocols are not supported
- Unknown scheduling behavior

YASA

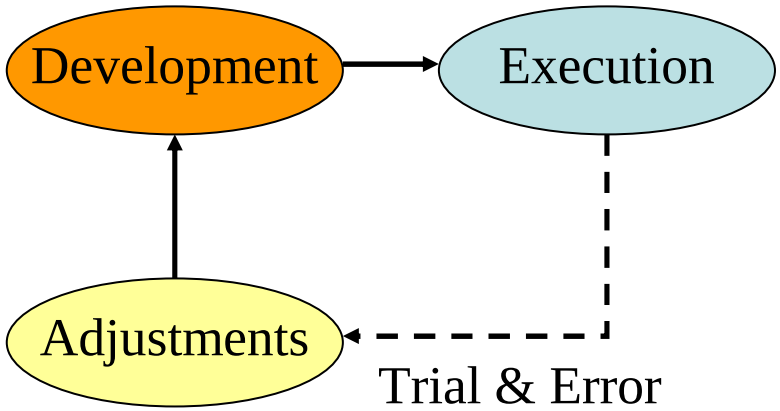
Goal: Optimization of scheduling behavior in real-time operating systems

Features:

- Analysis of scheduling behavior
- Exchange of schedulers
- Support of additional static and dynamic synchronizations protocols

Design Flow Comparison

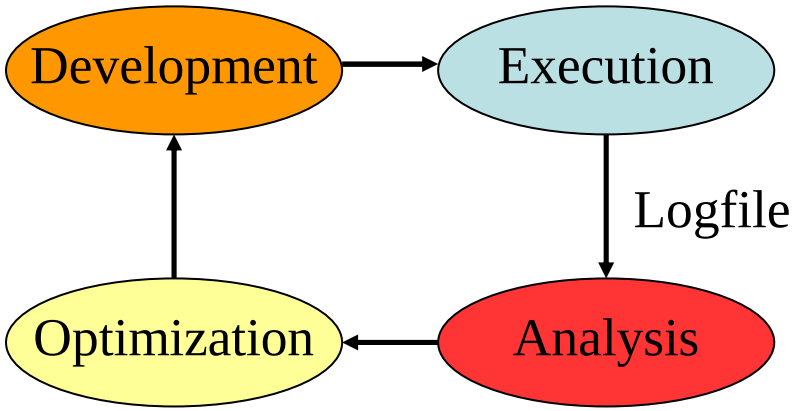
Current flow:



Characteristics:

- Unknown scheduling behavior
- No analysis
- Poor failure detection

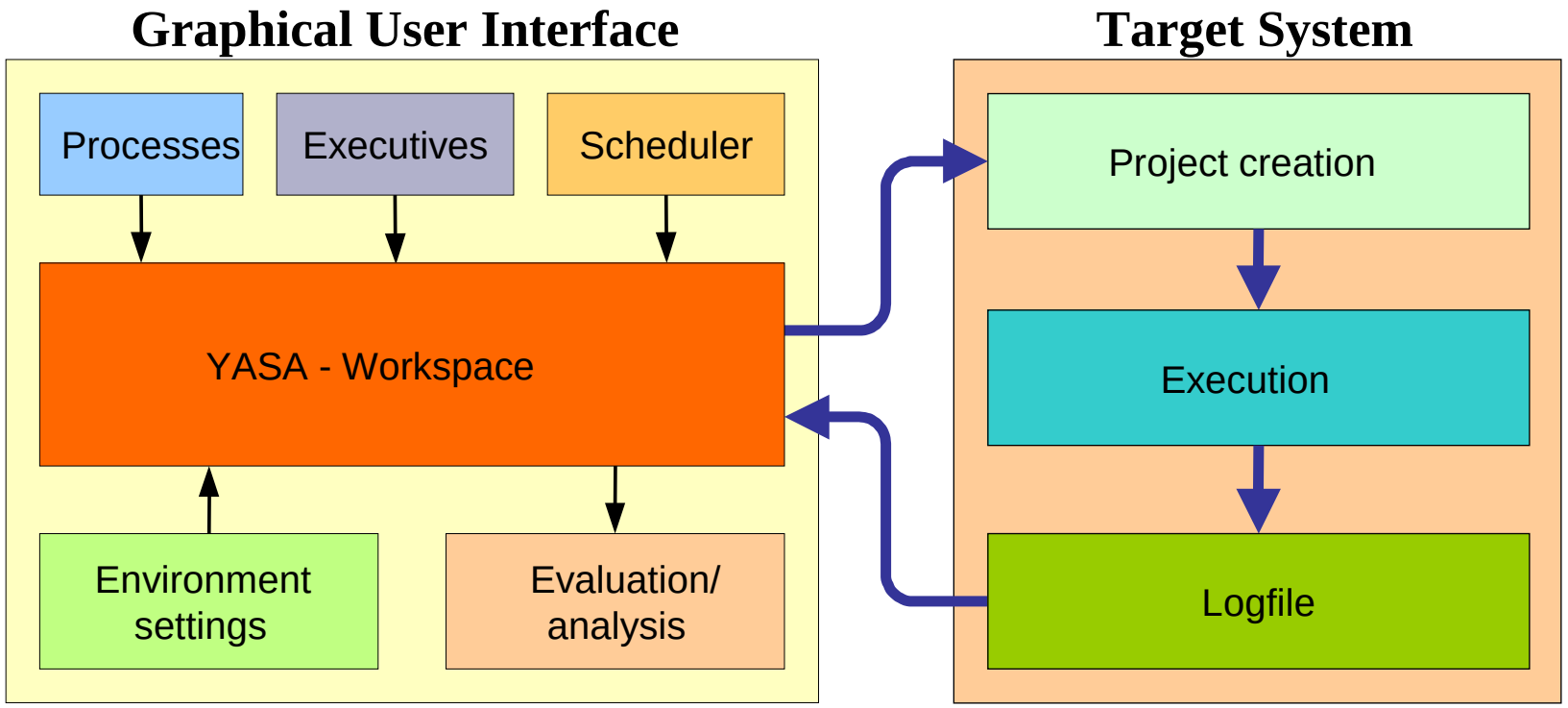
YASA 2:



Characteristics:

- Logged scheduling behavior
- Analysis of scheduling behavior
- Better failure detection

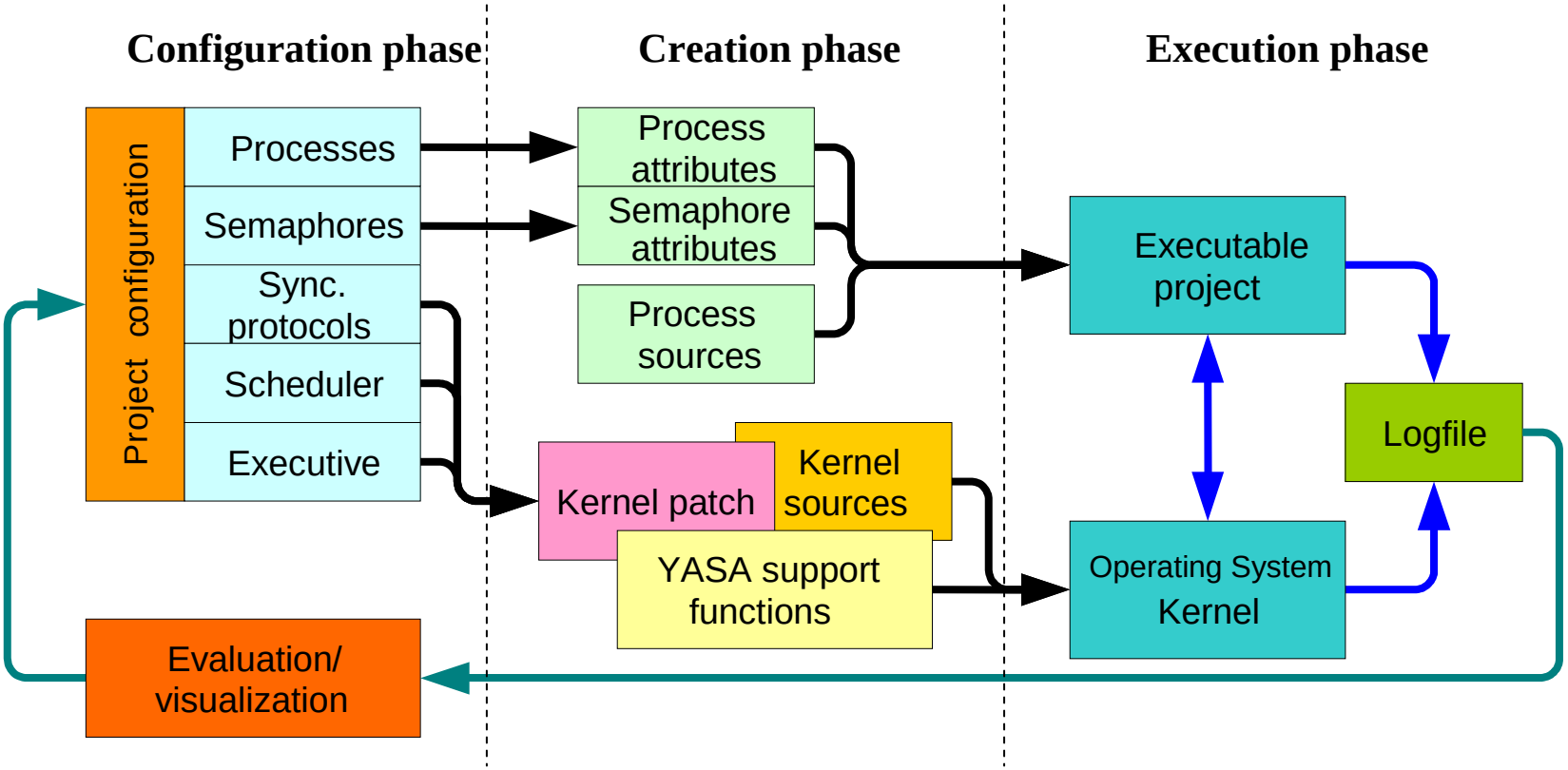
Design Flow in YASA



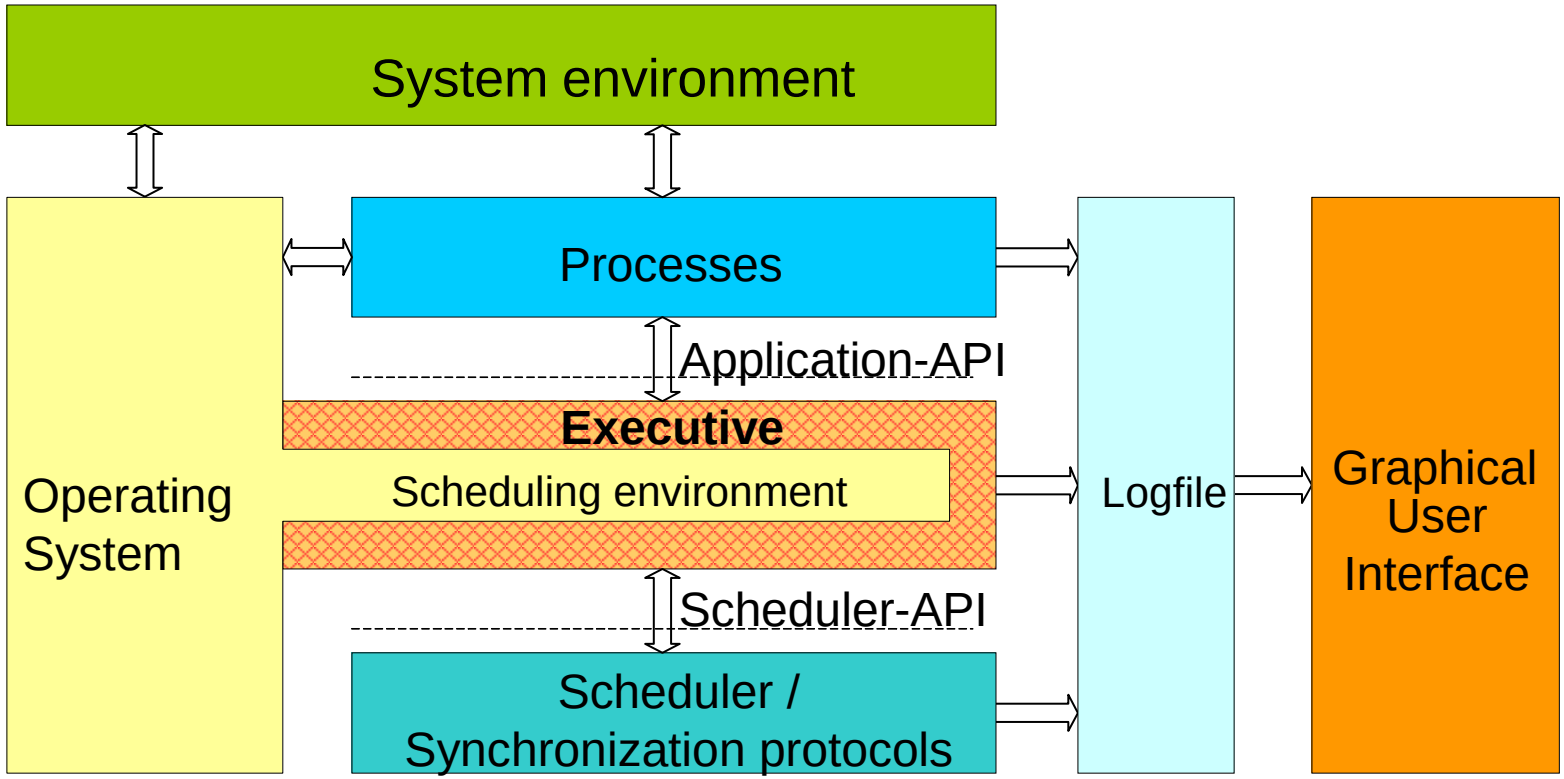
- Project management & configuration
- Source code generation
- Evaluation

- Compilation
- Execution
- Logfile creation

Project Cycle in YASA



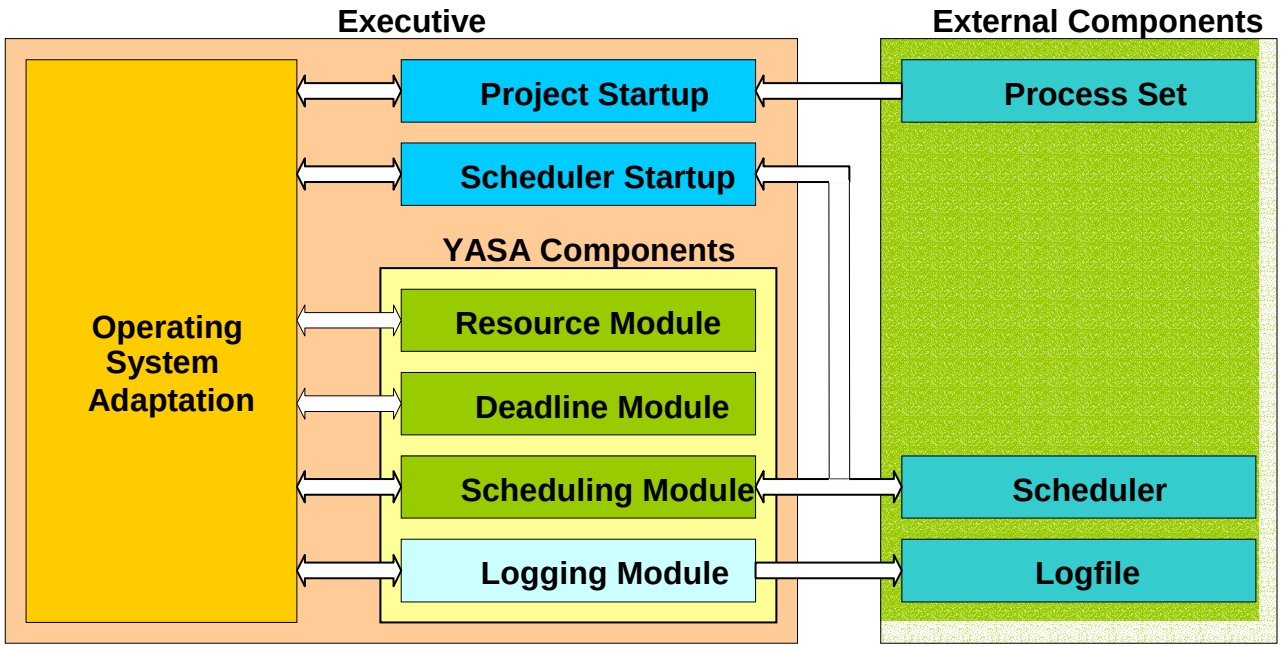
Integration of YASA



➡ Encapsulate scheduling environment into Executives !

Design of an Executive

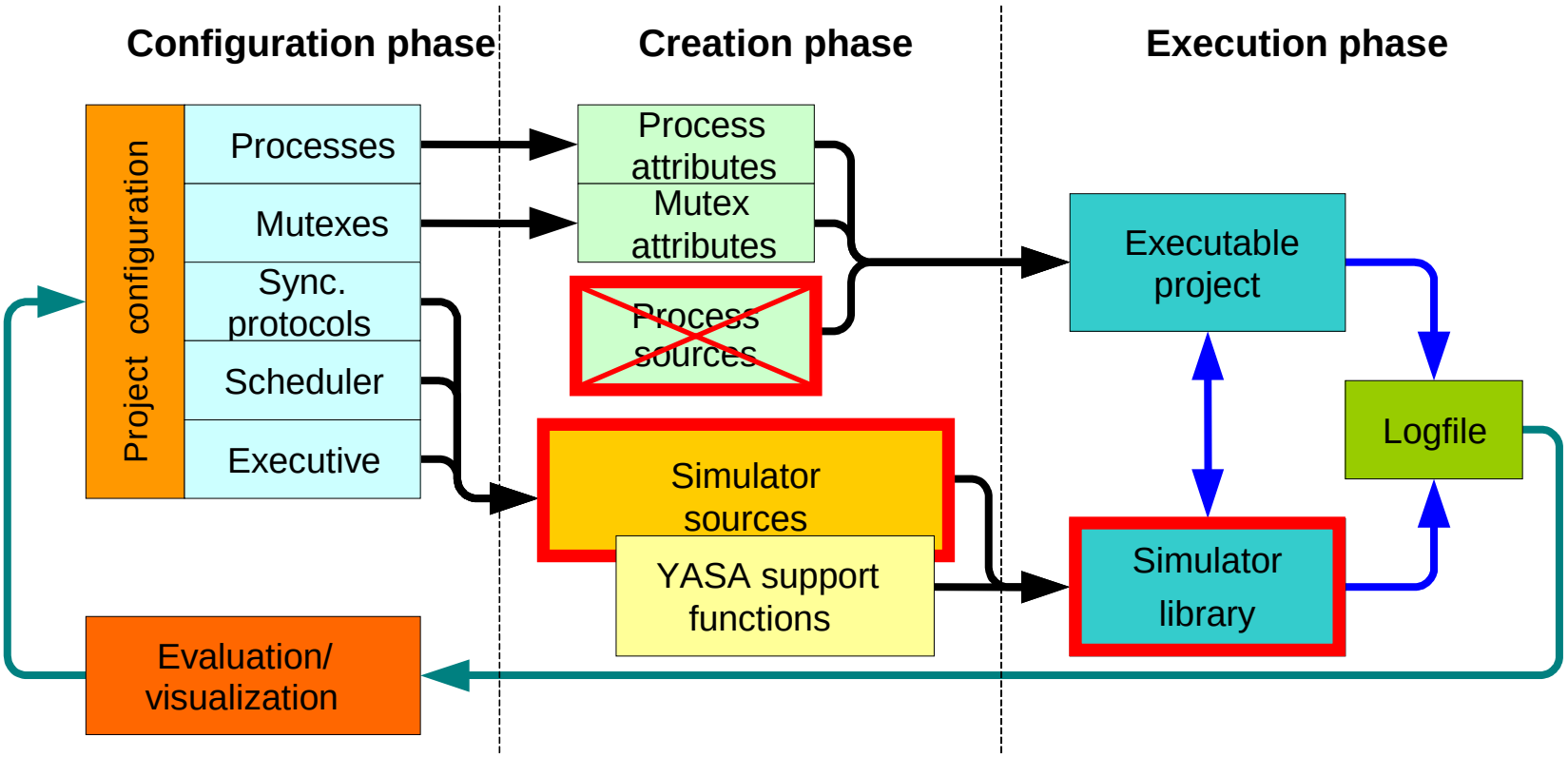
- Virtual scheduling environment
- Interface between scheduler and processes
- Support for platform independent schedulers



Simulator-Executive

- Execution of applications with synthetic work-load
- Virtual program flow
- Simulation of
  - Process states
  - Resource conditions
- Multiprocessing capable
- Platform independent
- Application fields:
  - Proof of theoretical analysis
  - Development and test of new schedulers

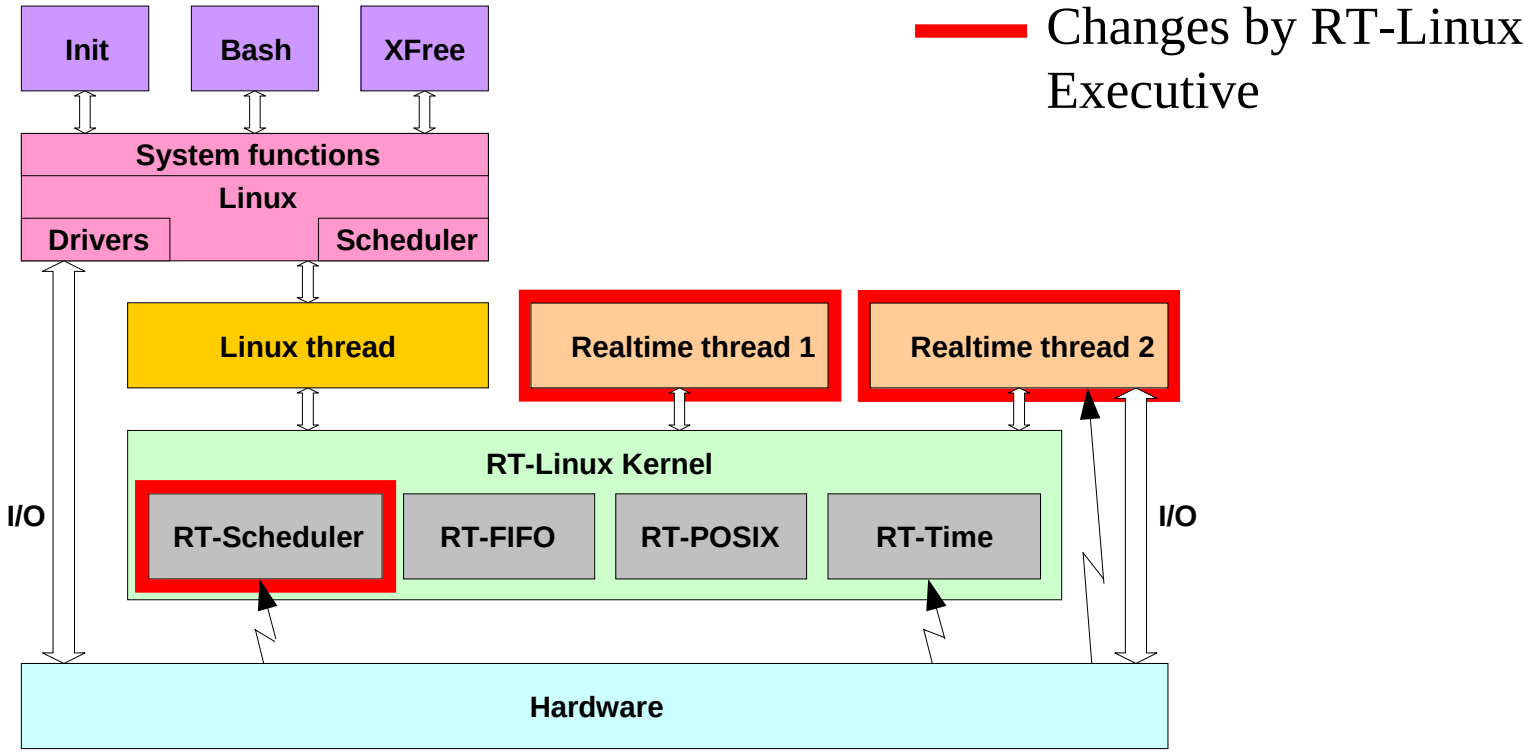
Project Cycle Using Simulator



Particularities of the Simulator-Executive

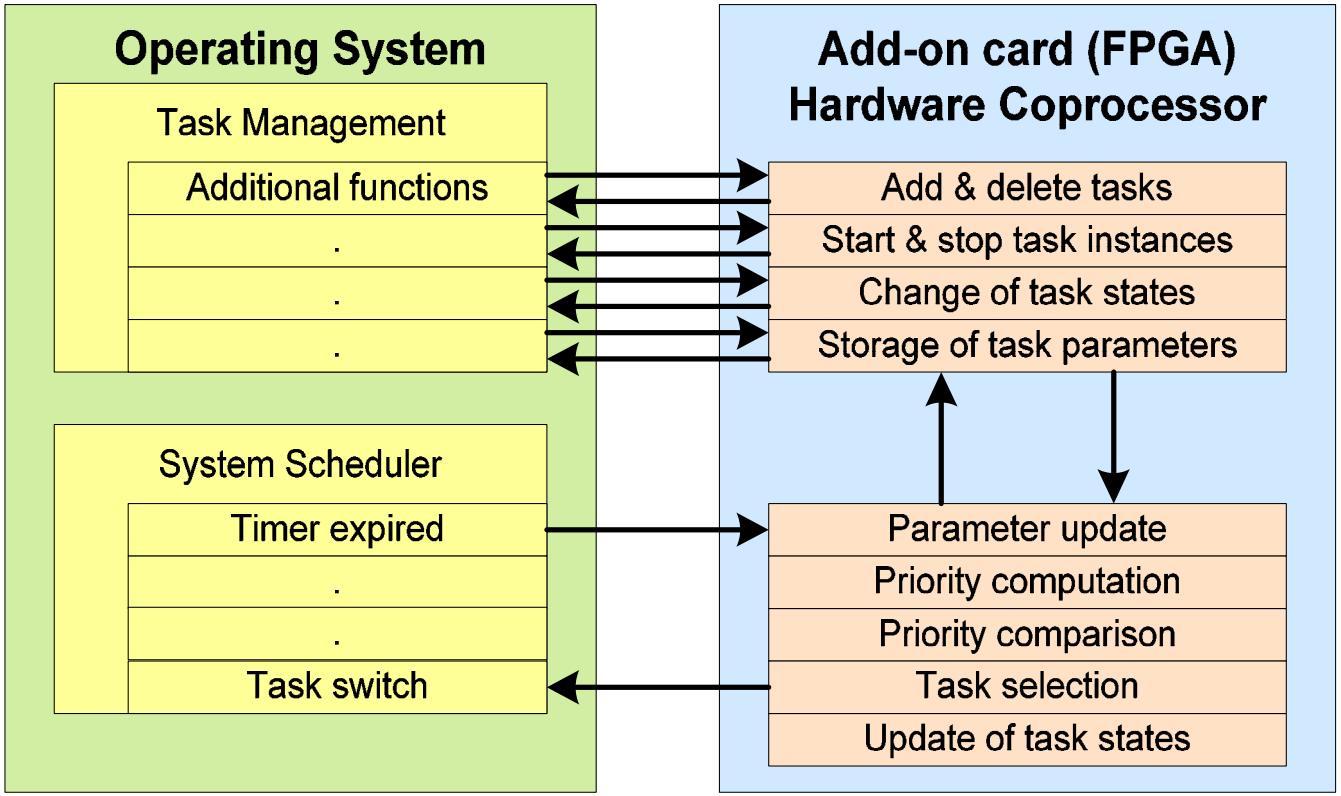
RT-Linux Executive

- Adaptation layer to RT-Linux
- Execution of real applications
- Enhancements of POSIX-API



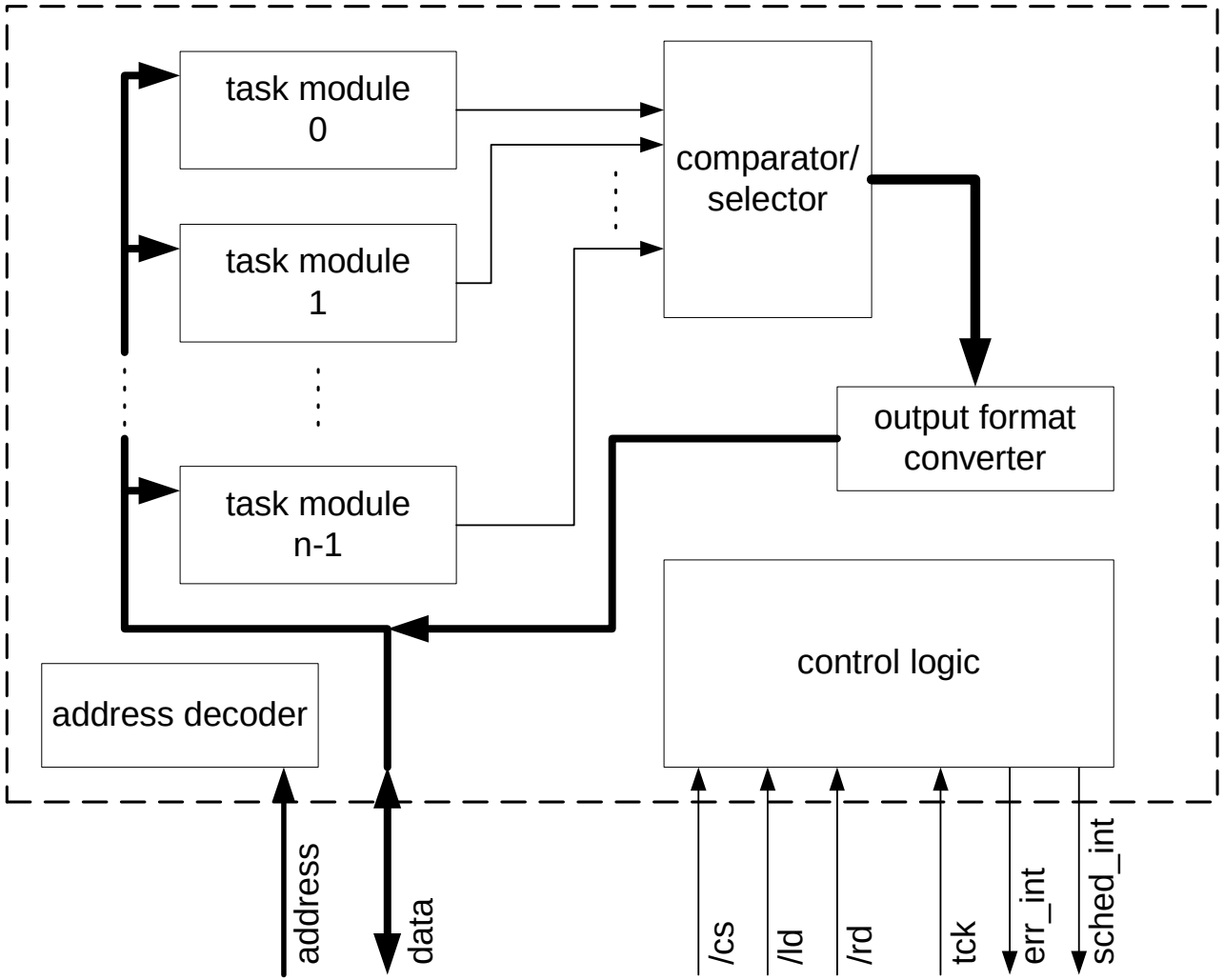
RT-Linux Coprocessor Executive

- Integration into OS by replacement / extension of several kernel functions
- **Parallel computation** (favorable for ELLF)



Executive not finished, yet!

Coprocessor – Internal Design



Development Environment

Scheduler

Properties:

- Replacement during runtime possible
- Implementation
  - Executive independent
  - Platform independent
  - No limitations in time resolution and process count

Available schedulers:

Static schedulers	Dynamic schedulers	Cyclic schedulers
Priority based	Earliest Deadline First	Round Robin
Rate Monotonic	Least Laxity First	First Come – First Serve
Deadline Monotonic	Enhanced Least Laxity First	

Synchronization Protocols

Properties:

- Synchronize operating resources
  - Reduce effect of priority inversion
  - Prevent deadlocks

Available synchronization protocols:

Static	Dynamic
Priority Inheritance Protocol (PIP)	Dynamic Priority Ceiling Protocol (DPCP)
Priority Ceiling Protocol (PCP)	Stack Resource Policy (SRP)
Ceiling Semaphore Protocol (CSP)	

Synchronization Protocols II

- **Decoupling synchronization protocols and schedulers**
- Supporting **dynamic priority types**
- Combining synchronization protocols with several schedulers at runtime
  - Dynamic synchronization protocols with every scheduler
  - Static synchronization protocols with every static scheduler

Scheduler	Type	Possible Synchronization Protocols
Priority Based Scheduler	Static	• Priority Inheritance Protocol • Priority Ceiling Protocol • Ceiling Semaphore Protocol • Dynamic Priority Ceiling Protocol • Stack Resource Policy
Earliest Deadline First	Dynamic	• Dynamic Priority Ceiling Protocol • Stack Resource Policy

Dynamic Priority Types

- Comparison methods and data types in synchronization protocols and schedulers must match
- Dynamic priority type:
  - depends on scheduler
  - controls comparison operation
  - affects data type size
- Access to data type through macros during compilation

Possible Priority Types:

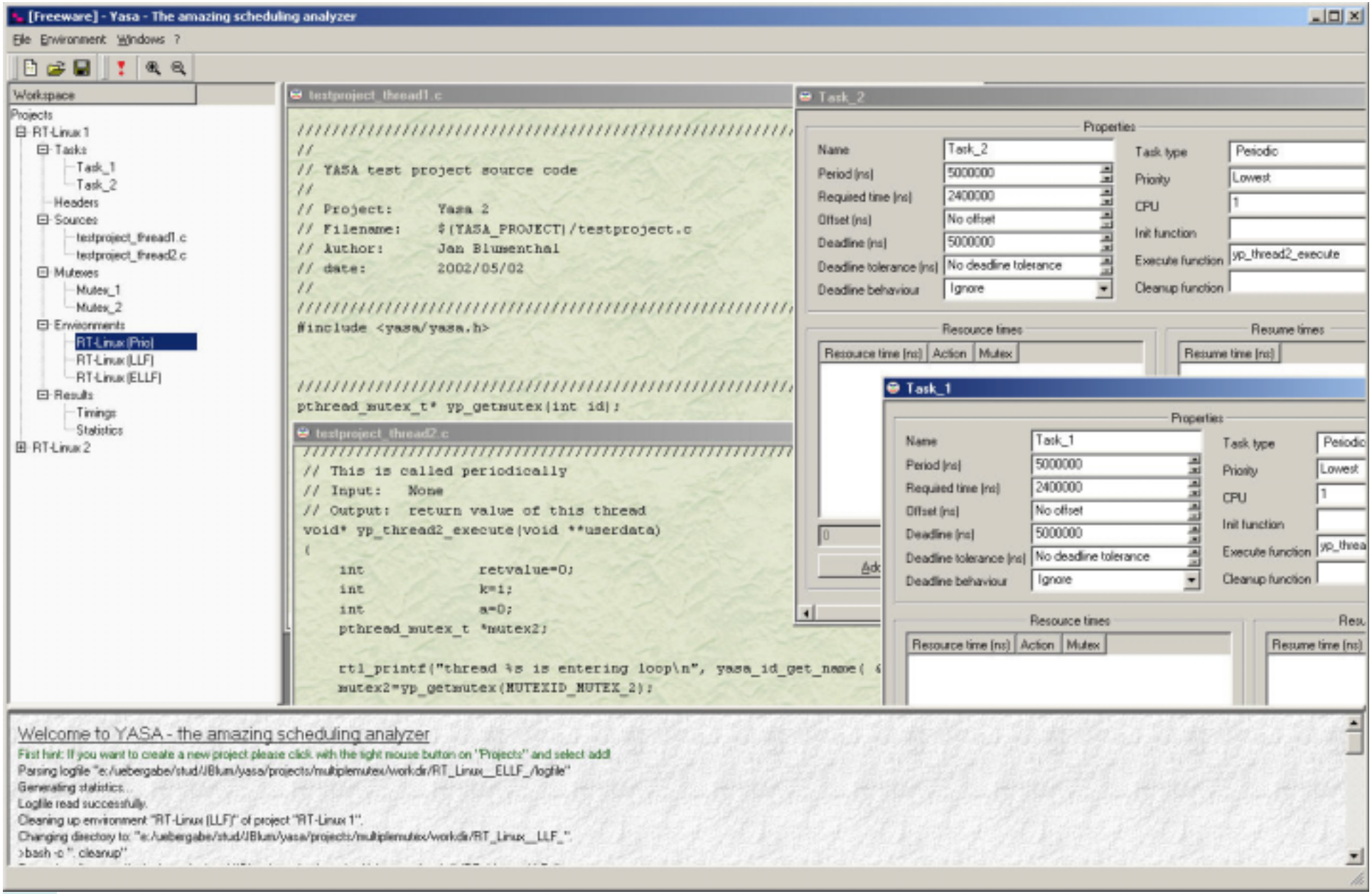
Name	Type	Data type	Operation
Priority	Static	int	$P_1 > P_2$
Period	Static	YASA_TIME	$P_1 < P_2$
Laxity	Dynamic	YASA_TIME	$P_1 < P_2$
Deadline	Dynamic	YASA_TIME	$P_1 < P_2$
Remaining runtime	Dynamic	YASA_TIME	$P_1 > P_2$

Graphical User-Interface YASA

- Object-oriented programming
- Using QT class library (261 classes)
- Platform independent
- Utilize Design Patterns
- Multilingual (UNICODE)
- Evaluate application logfiles in terms of

graphics → ***Task diagram*** ***Mutex diagram*** ***Processor diagram***

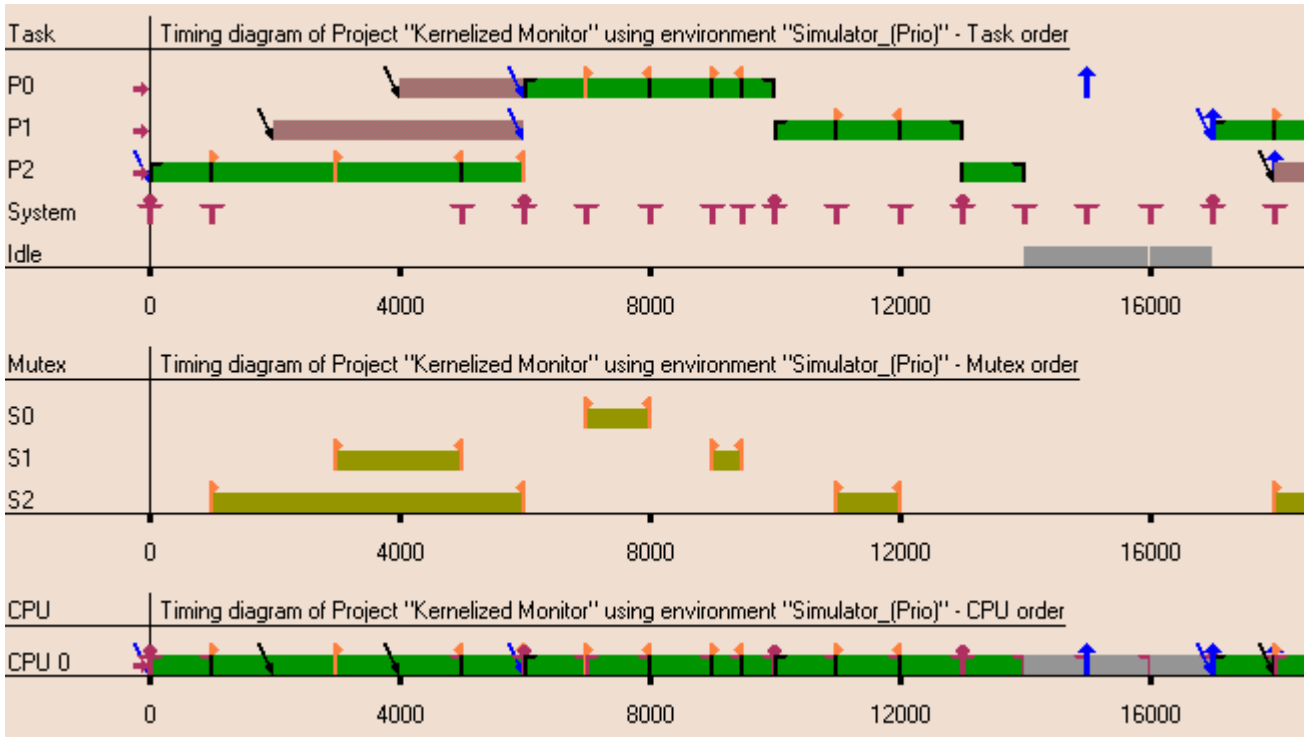
tables → ***Task table*** ***Mutex table*** ***Processor table***


[Freeware] - Yasa - The amazing scheduling analyzer
File Environment Windows ?
Workspace
Projects
RT-Linux 1
Tasks
Task_1
Task_2
Headers
Sources
testproject_thread1.c
testproject_thread2.c
Mutexes
Mutex_1
Mutex_2
Environments
RT-Linux (Prio)
RT-Linux (LLF)
RT-Linux (ELLF)
Results
Timings
Statistics
RT-Linux 2
testproject_thread1.c
// YASA test project source code
// Project: Yasa 2
// Filename: $(YASA_PROJECT)/testproject.c
// Author: Jan Blumenthal
// date: 2002/05/02
#include <yasa/yasa.h>
pthread_mutex_t* yp_getmutex(int id);
testproject_thread2.c
// This is called periodically
// Input: None
// Output: return value of this thread
void* yp_thread2_execute(void **userdata)
{
int retvalue=0;
int k=1;
int a=0;
pthread_mutex_t *mutex2;
rtl_printf("thread %s is entering loop\n", yasa_id_get_name(
mutex2=yp_getmutex(MUTEXID_MUTEX_2);
Task_2
Properties
Name Task_2
Period [ns] 5000000
Required time [ns] 2400000
Offset [ns] No offset
Deadline [ns] 5000000
Deadline tolerance [ns] No deadline tolerance
Deadline behaviour Ignore
Task type Periodic
Priority Lowest
CPU 1
Init function
Execute function yp_thread2_execute
Cleanup function
Resource times
Resource time [ns] Action Mutex
Resume times
Resume time [ns]
Task_1
Properties
Name Task_1
Period [ns] 5000000
Required time [ns] 2400000
Offset [ns] No offset
Deadline [ns] 5000000
Deadline tolerance [ns] No deadline tolerance
Deadline behaviour Ignore
Task type Periodic
Priority Lowest
CPU 1
Init function
Execute function yp_threa
Cleanup function
Resource times
Resource time [ns] Action Mutex
Resume time [ns]
Welcome to YASA - the amazing scheduling analyzer
First hint: If you want to create a new project please click with the right mouse button on "Projects" and select add!
Parsing logfile "e:/uebergabe/stud/JBlum/yasa/projects/multiplemutex/workdir/RT_Linux__ELLF_/logfile"
Generating statistics...
Logfile read successfully.
Cleaning up environment "RT-Linux (LLF)" of project "RT-Linux 1".
Changing directory to: "e:/uebergabe/stud/JBlum/yasa/projects/multiplemutex/workdir/RT_Linux__LLF_"
>bash -c ". cleanup"

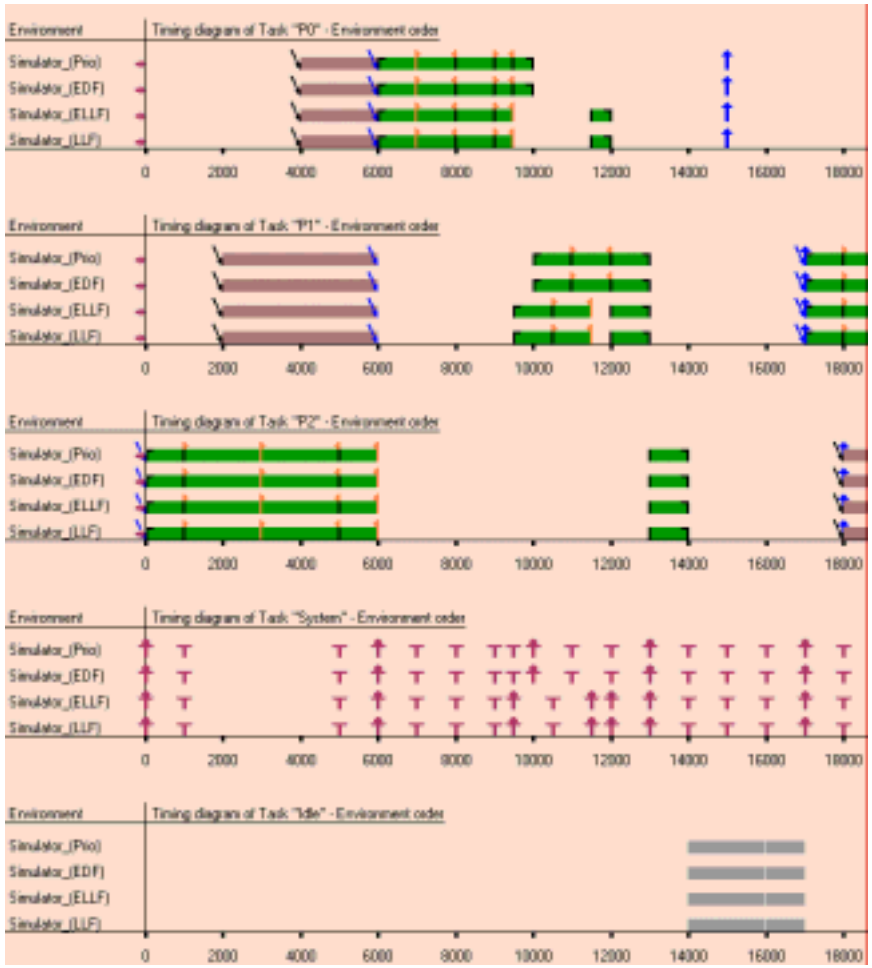
Graphical Evaluation I

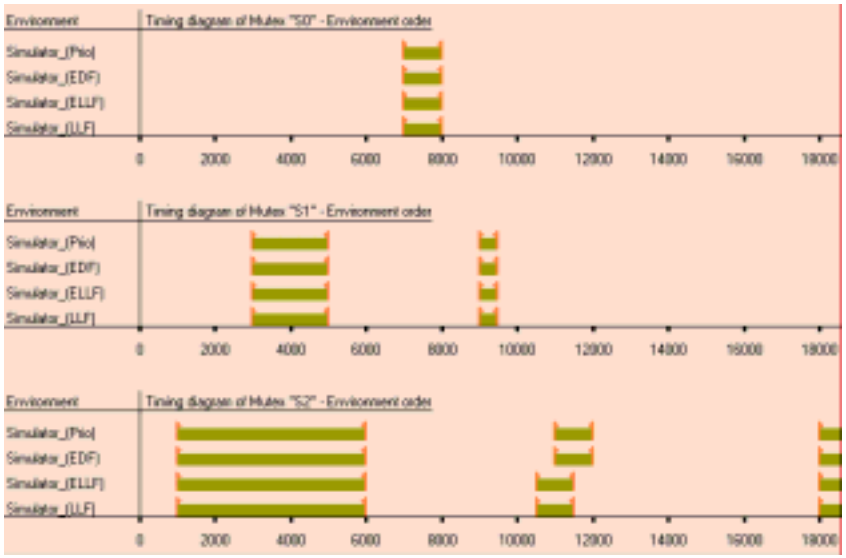
Represent timing diagrams in

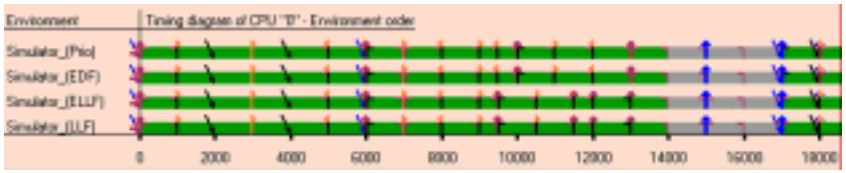
- Task order
- Mutex order
- Processor order



Graphical Evaluation II







Visualization in **Environment** order!

Conclusion

- Framework YASA
  - Scheduling analysis with different schedulers
  - Extends existing real-time operating systems
  - Platform independent schedulers and synchronization protocols
  - Simulation and execution of real programs
  - Graphical front-end for configuration and evaluation issues

- Homepage

  http://yasa.e-technik.uni-rostock.de

  http://sourceforge.net/projects/yasa

Thank you

Jan Blumenthal, Jens Hildebrandt, Frank Golatowski, and Dirk Timmermann

Universität Rostock

5th Real-Time Linux Workshop
Valencia, 2003/11/13