# A Configurable Hardware Scheduler (CHS) for Real-Time Systems

**Pramote Kucharoen, Mohamed A. Shalan and Vincent J. Mooney III**

***Center for Research on Embedded Systems and Technology***
***School of Electrical and Computer Engineering***
***Georgia Institute of Technology***
***Atlanta, Georgia, USA***

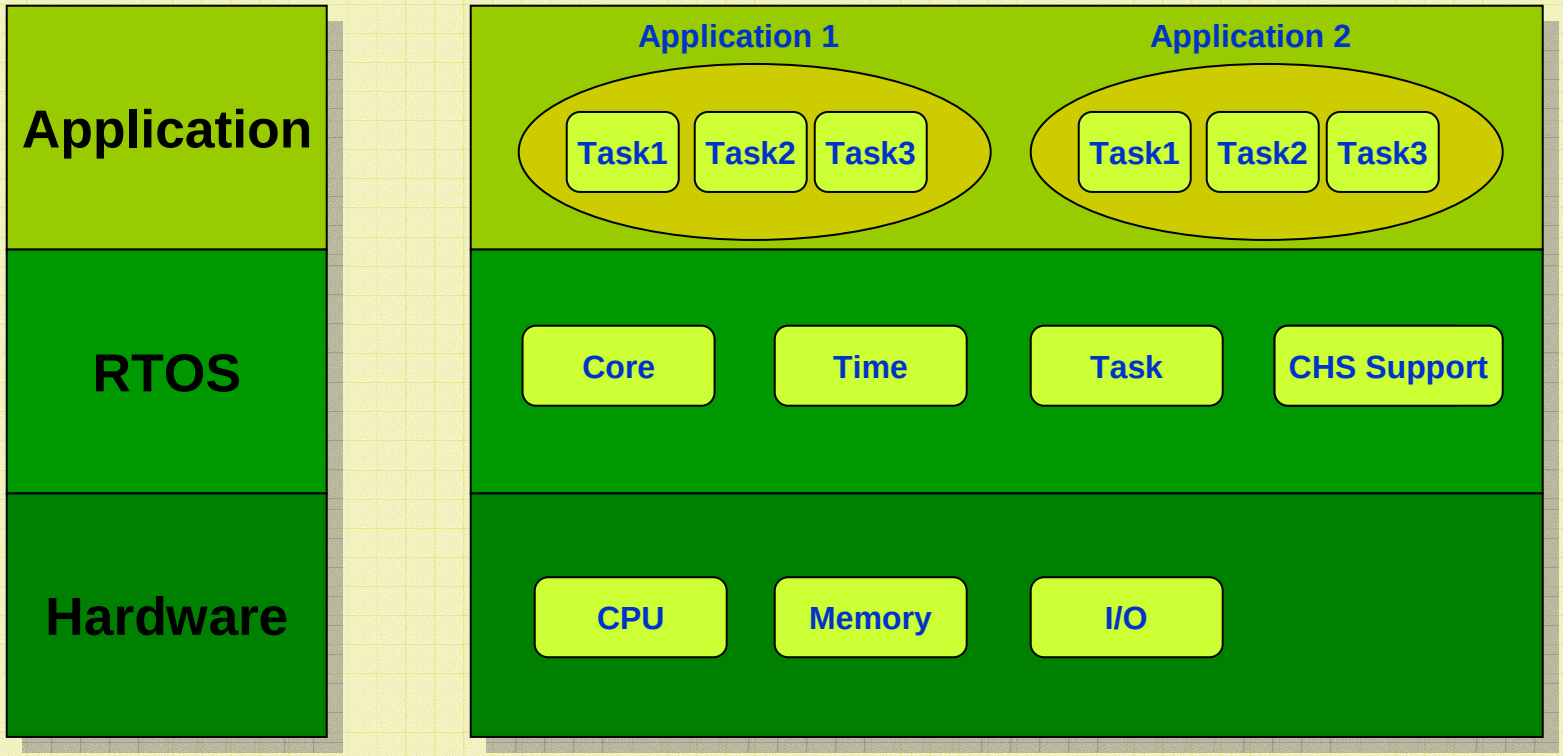
**23 June 2003**


# Outline

- Introduction
- Related work
- CHS architecture
- CHS commands
- CHS interface
- Software support
- Automatic customization of CHS
- Experiments and results
- Conclusion

# Introduction

## Real-time system layers

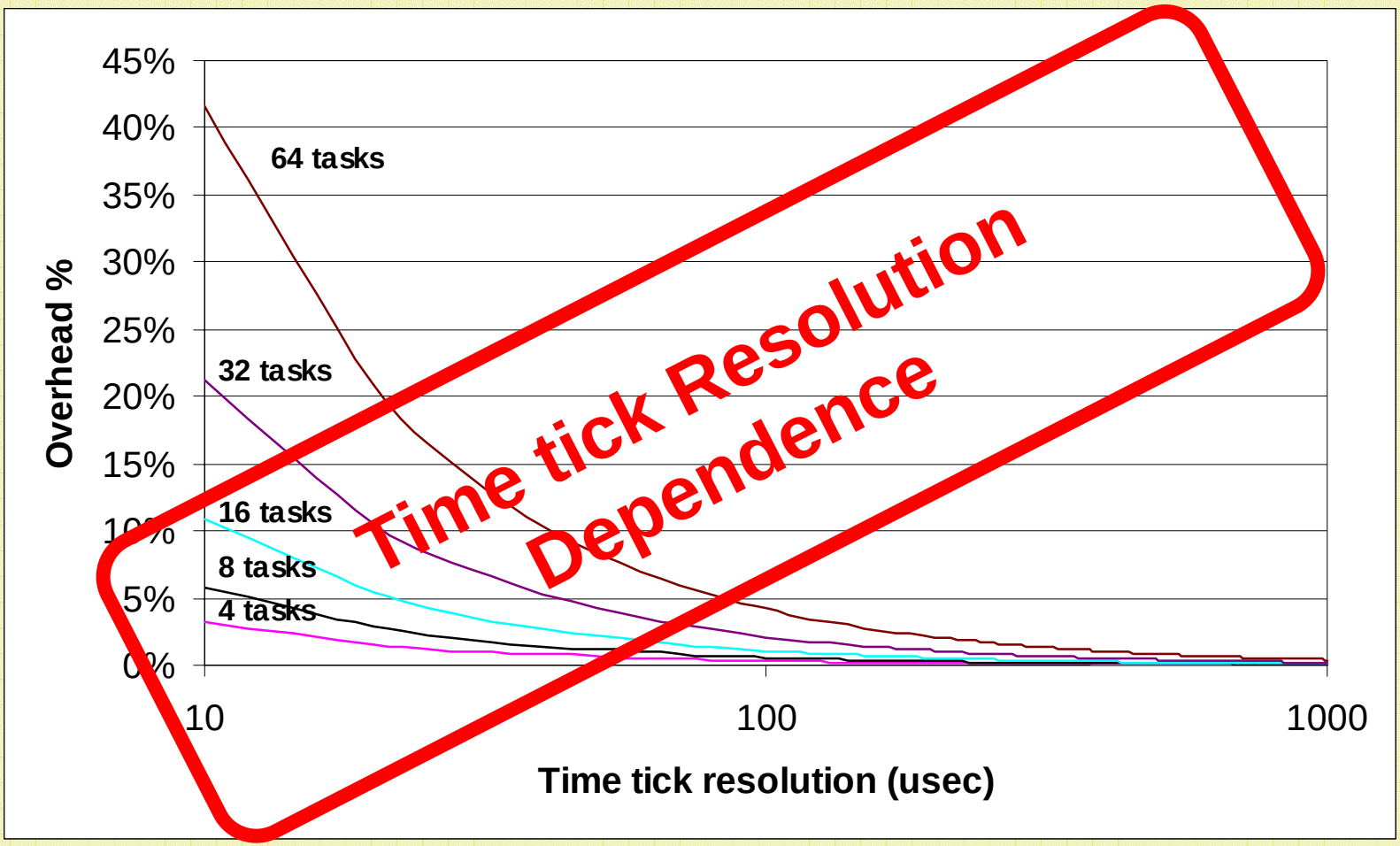


# µC/OS II Background Processing

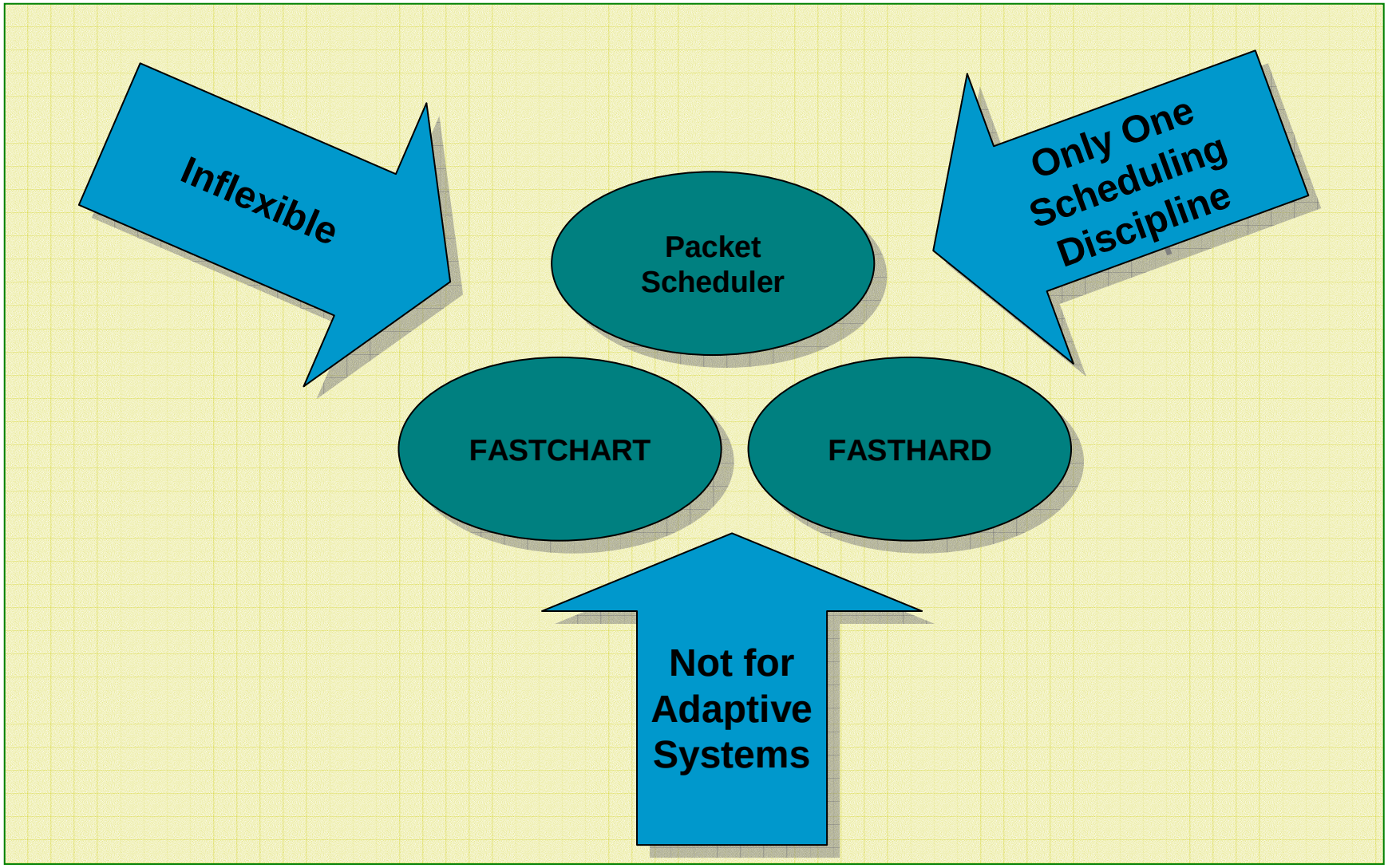
```
ptcb = OSTCBList;                          /* Point at first TCB in TCB list */
while (ptcb->OSTCBId != OS_TASK_IDLE_ID) { /* Go through all TCBs in TCB list */
    OS_ENTER_CRITICAL();
    if (ptcb->OSTCBDly != 0) {             /* Delayed or waiting for event with TO     */
      if (--ptcb->OSTCBDly == 0) {         /* Decrement nbr of ticks to end of delay   */
        if (!(ptcb->OSTCBStat & OS_STAT_SUSPEND)) { /* Is task suspended? */
          OSSched(ptcb,RDY);
        else                               /* Yes, Leave 1 tick to prevent loosing */
          ptcb->OSTCBDly = 1;              /* the task when the suspension is removed. */
      }
    }
    ptcb = ptcb->OSTCBNext;                /* Point at next TCB in TCB list            */
    OS_EXIT_CRITICAL();
  }
```

NOT FIXED-CYCLE OPERATIONS
Number of Tasks Dependent

# Overhead in µC/OS II Scheduler



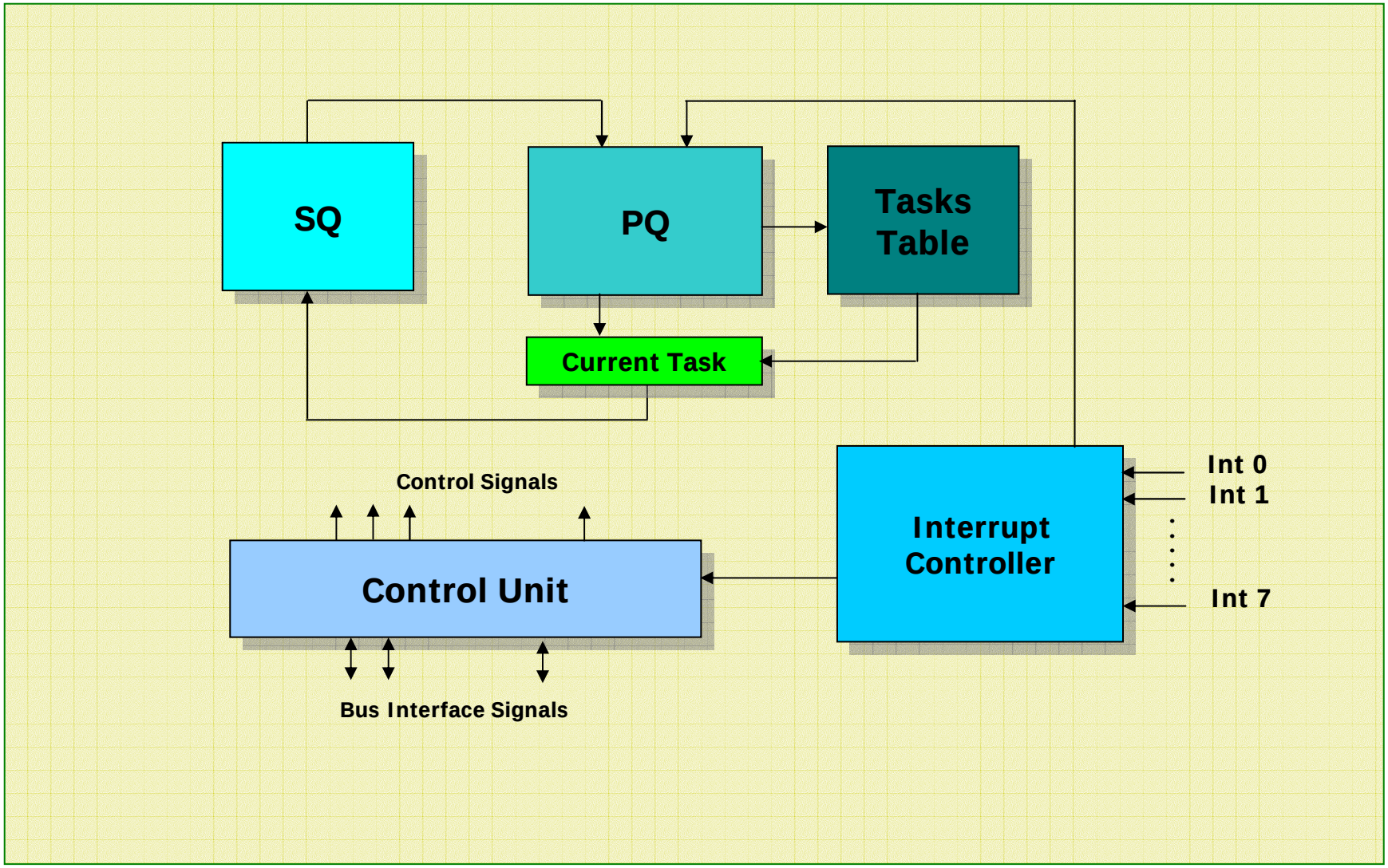
# Related Work



# Why do we need the CHS?

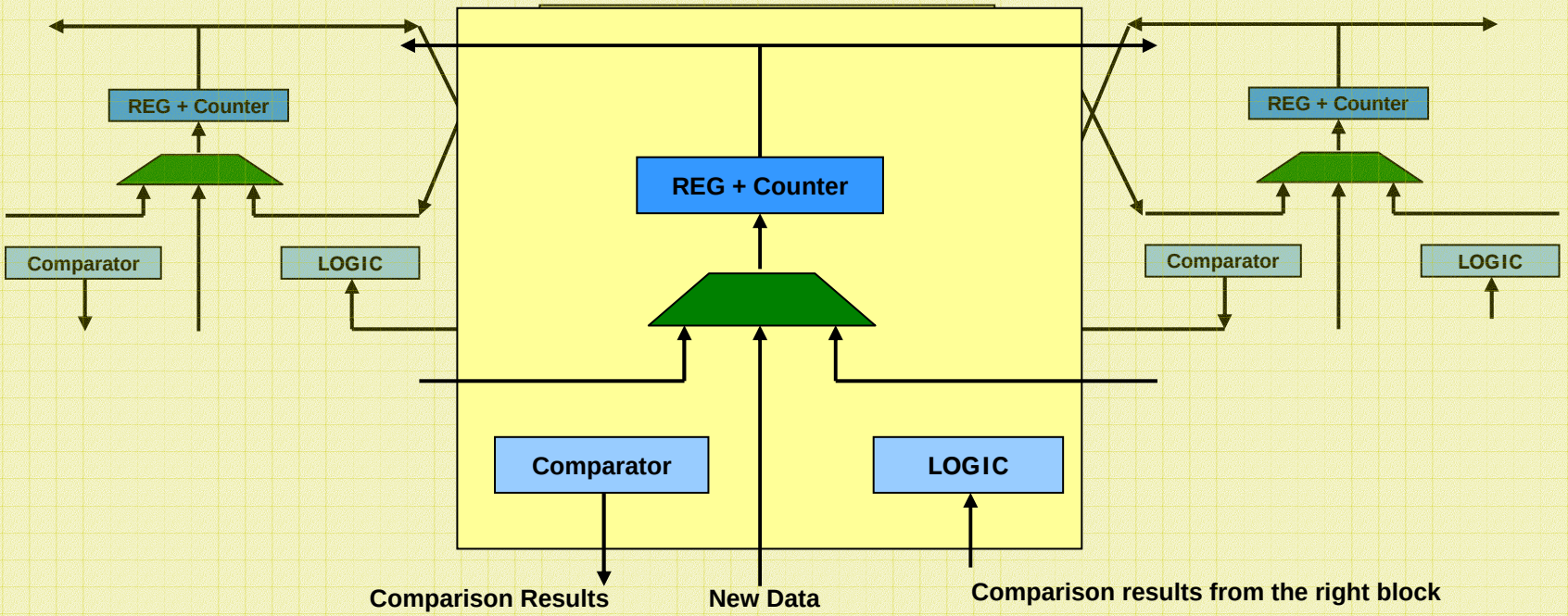
- To reduce the scheduling overhead from the real-time operating system; hence, improve the system response time
- To support a wide range of applications by supporting multiple scheduling disciplines that can be changed during system execution time.
  - Priority
  - Earliest Dead Line First (EDF)
  - Rate Monotonic (RM)

# CHS Architecture (1)



# CHS Architecture (2)

## Priority Queue (Ready Queue)




| ID | Register | Counter |
|---|---|---|

# CHS Architecture (3)

**Sleep Queue**

- Used to store the Sleeping Tasks (YIELD/SLEEP).
- The Tasks are sorted according to their remaining sleep time.
- Once The Sleep Time expires it is moved to the PQ.

| ID | Counter |
|---|---|

# CHS Architecture (4)

**Task Table**

- **Store Information about the existing tasks**
- **Indexed by the Task ID**

| PRI | Period | WCET | TYPE | PRE | STATUS |
|---|---|---|---|---|---|

# CHS Commands

| | Command | # of Cycles |
|---|---|---|
| Scheduler Related | STOP | 1 |
| | RUN | 1 |
| | CONFIGURE | 1 |
| Task Related | CREATE Task | 1 |
| | MODIFY Task | 2 |
| | SLEEP | 2 |
| | SSLEEP | 1 |
| | YIELD | 1 |
| | SUSPEND | 1 |
| | RESUME | 1 |
| | DELETE | 1 |

# CHS Interface

The CHS Hardware is designed to be able to interface easily to any microprocessor core:

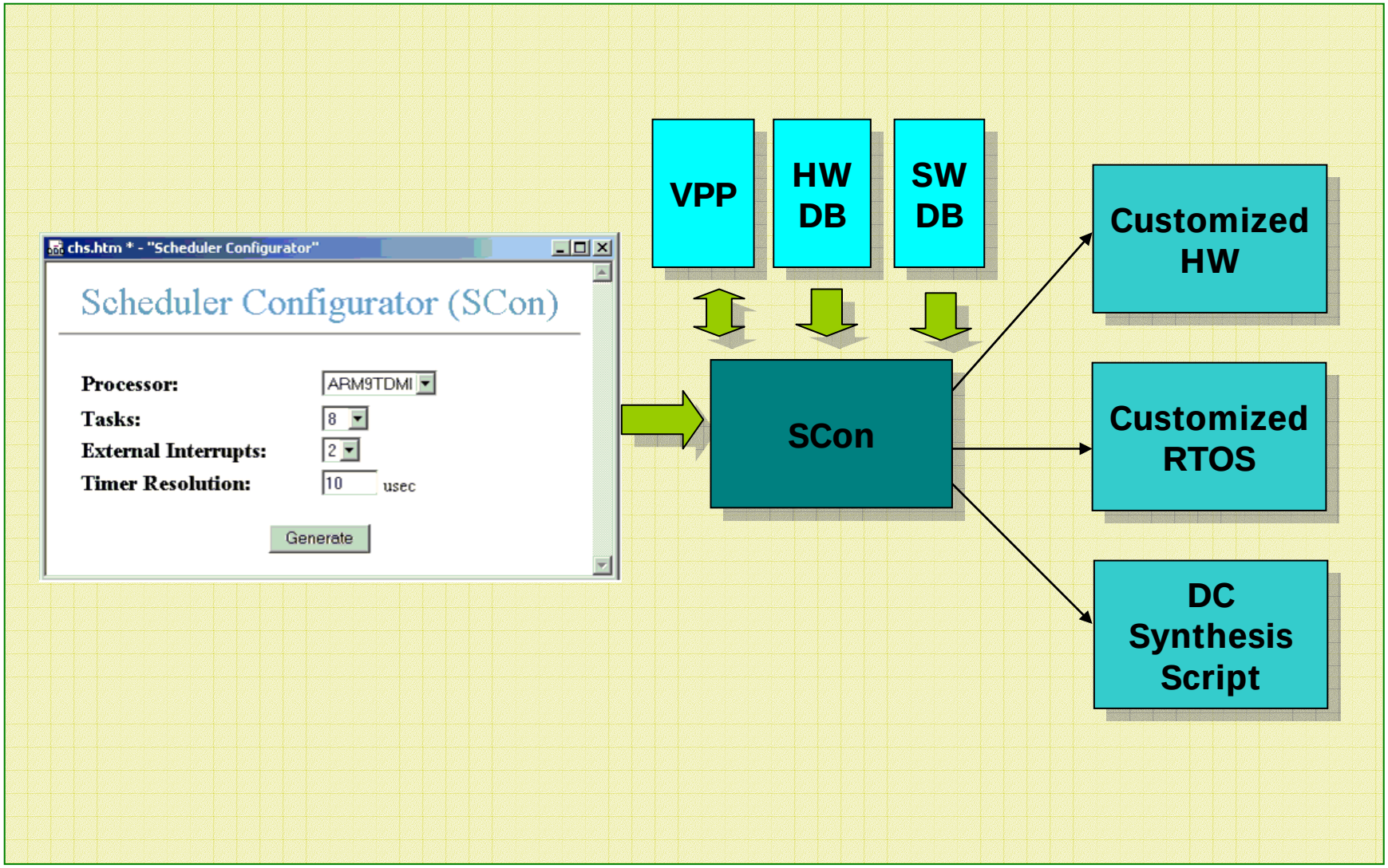
- As a memory mapped I/O Port,
- As a co-processor, or
- As instruction-set accelerator

# Software Support

**APIs**

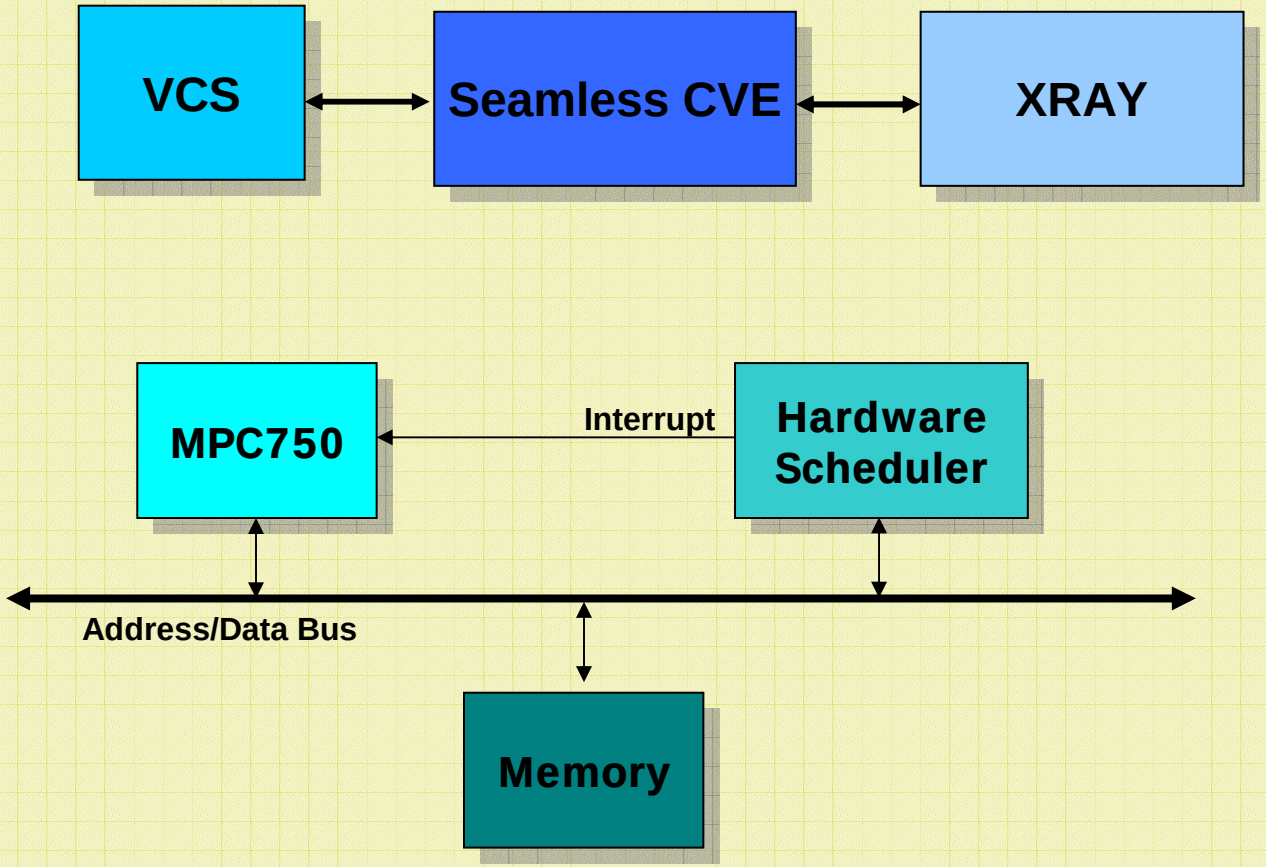
- Task
  - createTask
  - suspendTask, resumeTask
  - changePriority, changeWCET, changePeriod
  - Yield
  - ssleep, sleep
- Scheduler
  - configureScheduler
  - enableScheduler, disableScheduler

# Automatic Customization of CHS



# Experiments and Results (1)

## Simulation Environment



# Experiments and Results (2)

## Assembly instruction execution comparison

| | Micro C/OS II | Hardware Scheduler |
|---|---|---|
| Scheduler* | 69 | 0 |
| Time-tick processing | 47+47*(number of tasks) | 0 |

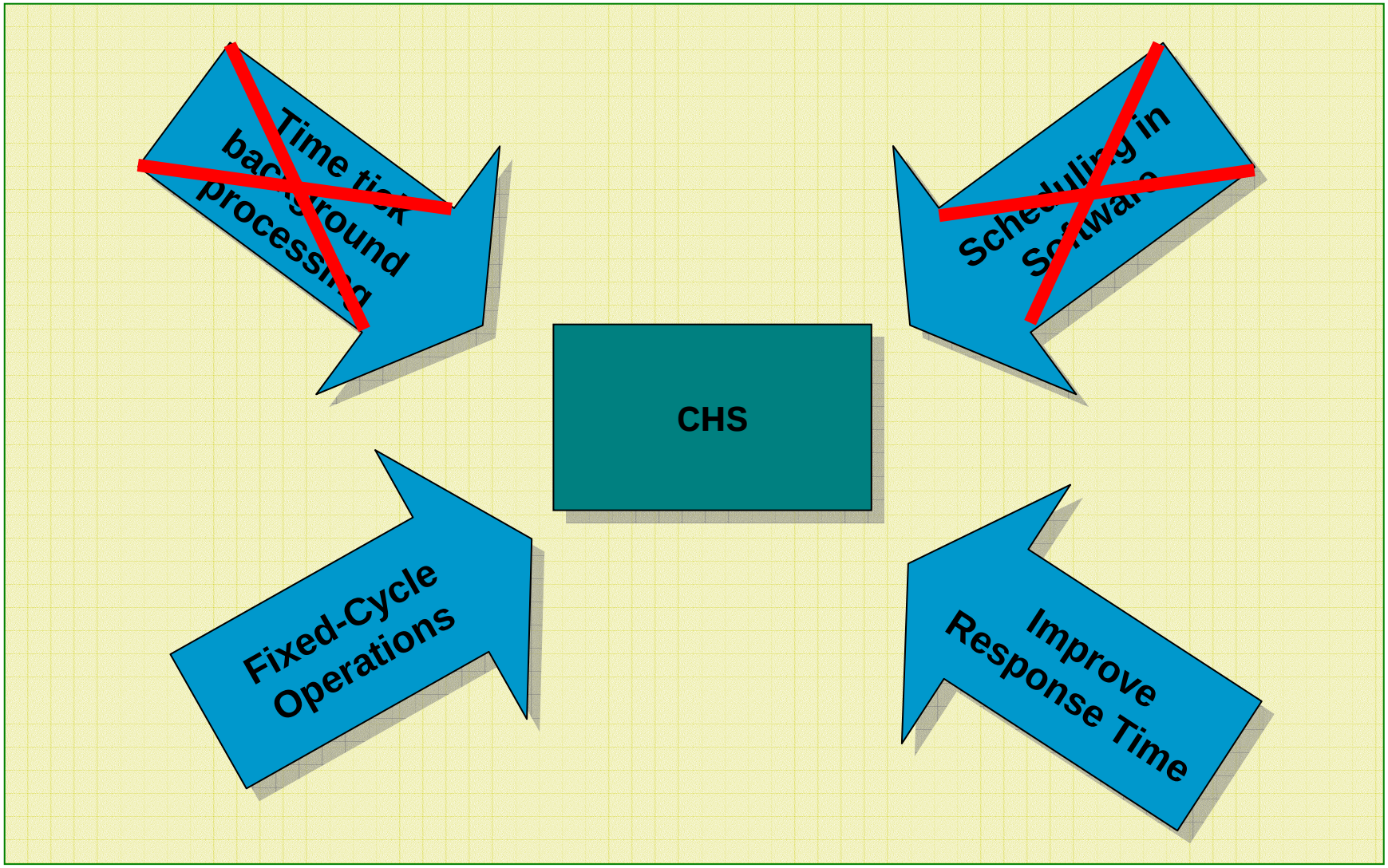
* Priority Scheduler

## Number of PowerPC instruction of the APIs

| API | # of PPC Assembly Instructions | WCET (# of cycles) |
|---|---|---|
| configureScheduler | 37 | 230 |
| SuspendTask | 21 | 125 |

CHS Requires One PPC Instruction to be Configured and One Instruction to Suspend a Task which means over 100x Speedup.

# Experiments and Results (3)



# CHS Synthesis Results

| Number of standard cells | Area ($mm^2$) |
|---|---|
| 1115 | 0.24 |

Using HP 0.35µ process

| Number of Logic Elements | Number of Registers |
|---|---|
| 421 | 564 |

Using Altera Quartus II for EP20K

The Synthesized CHS Supports

- 16 Tasks and
- up to 8 interrupt sources

# Conclusion

- We implemented a configurable hardware scheduler that supports 3 scheduling algorithms
- We developed software interface for the configurable hardware scheduler and a tool to generate a customized synthesizable CHS
- The configurable hardware scheduler eliminated the time spent by the processor for background time tick processing and scheduling

# Questions?