Adaptability, Extensibility, and Flexibility in Real-Time Operating Systems: the Georgia Tech DRTOS

Pramote Kuacharoen, Tankut Akgul,

Professor Vincent Mooney and Professor Vijay K. Madisetti

{pramote,tankut,mooney,vkm}@ece.gatech.edu

School of Electrical and Computer Engineering

Georgia Institute of Technology

This research is funded by the State of Georgia under the Yamacraw initiative and by NSF.

July 31, 2001

yamacraw futureforward

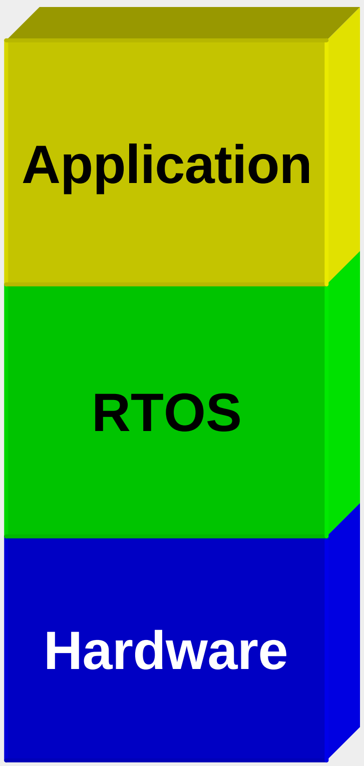
Georgia Institute of Technology

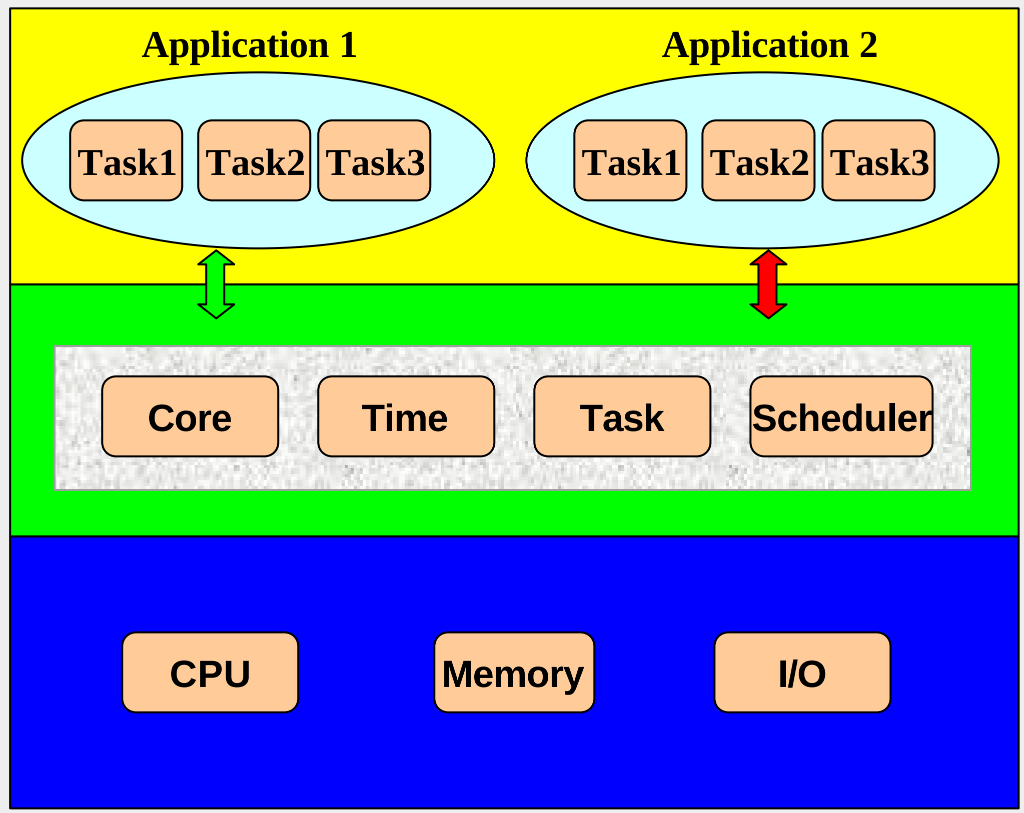
Outline

- Introduction
- Related Work
- Technical Approach
- Experiments and Results
- Conclusion

Introduction (continued)

Embedded Systems





Related Work

- **SPIN (University of Washington) 1996**
  - A general purpose operating system that provides extensibility, safety and good performance.
  - SPIN OS consists of a set of extension services and core system services that execute within the kernel's virtual address space
  - Extensions can be loaded into kernel at any time. Once loaded, they integrate themselves into the existing infrastructure and provide system service specific to the application that require them.
  - User space and kernel space are kept separate.
  - Single processor.
  - The core system services cannot be changed.

Related Work (continued)

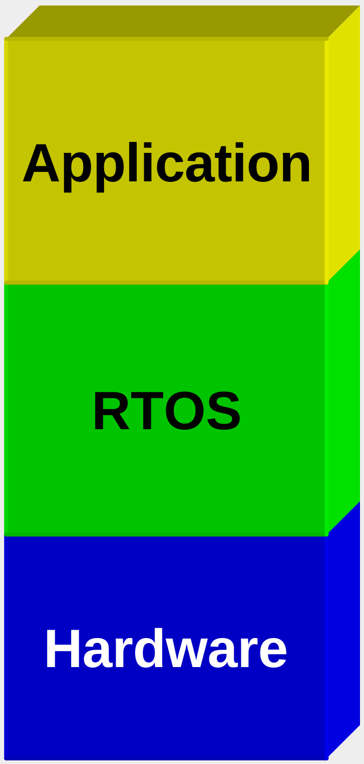
- **Exokernel (MIT) 1997**
  - General purpose operating system.
  - Exokernel's sole function is to allocate, de-allocate, and multiplex physical resources in a secure way (very good protection is provided).
  - The lower level interface allows flexible user-level implementations of traditionally rigidly defined OS services.
  - Single processor.
  - The core kernel code cannot be changed.

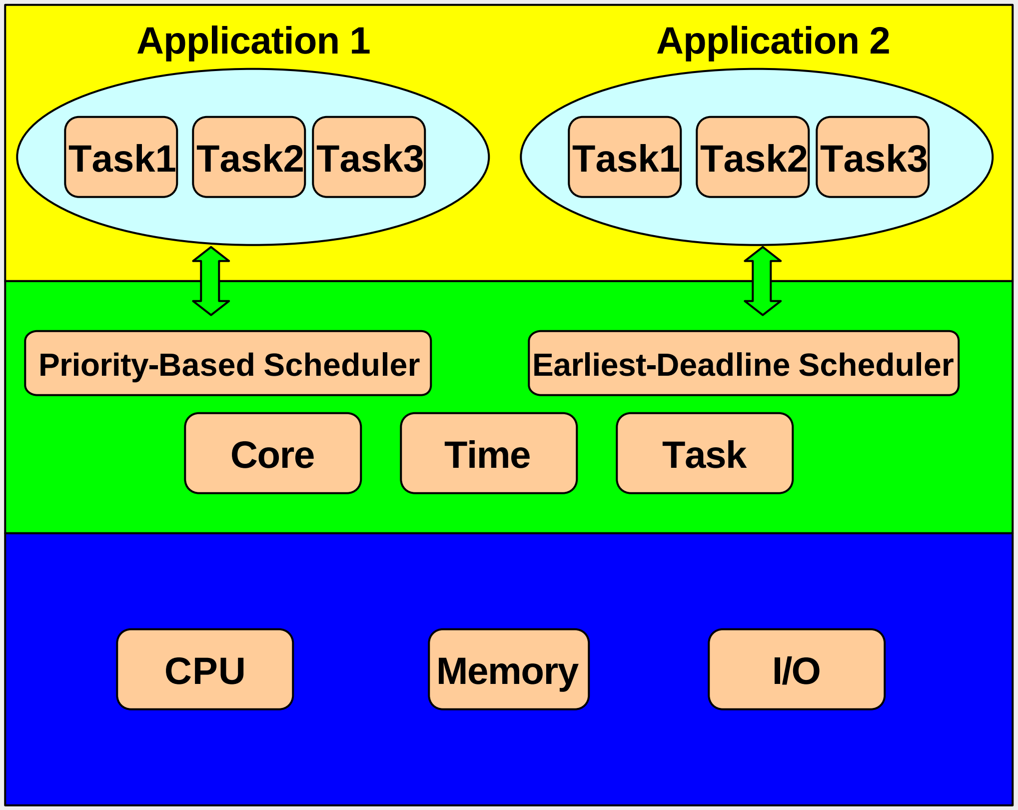
Why is it important to be able to dynamically change the core?

- **Unsafe or not convenient to reboot**
- **Adaptability and Flexibility**
  - Example: interrupt handling
    - Case 1: very fast handling of interrupts (always stop current interrupt)
    - Case 2: non-interruption of a particular interrupt
  - There may be no way to predict all the additional cases which could come up

DRTOS **Technical Approach**

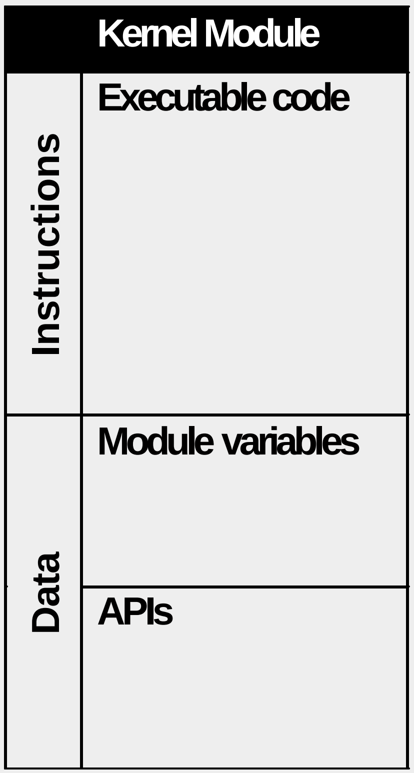
Embedded Systems

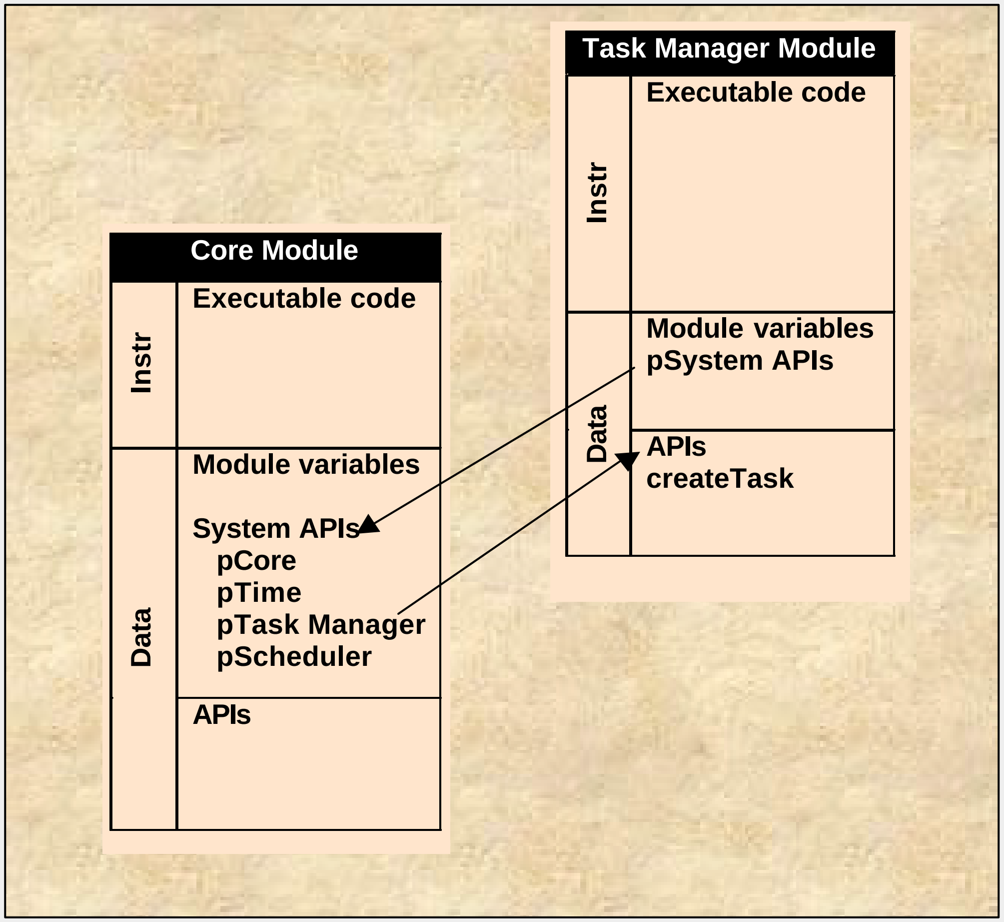




DRTOS **Technical Approach (continued)**

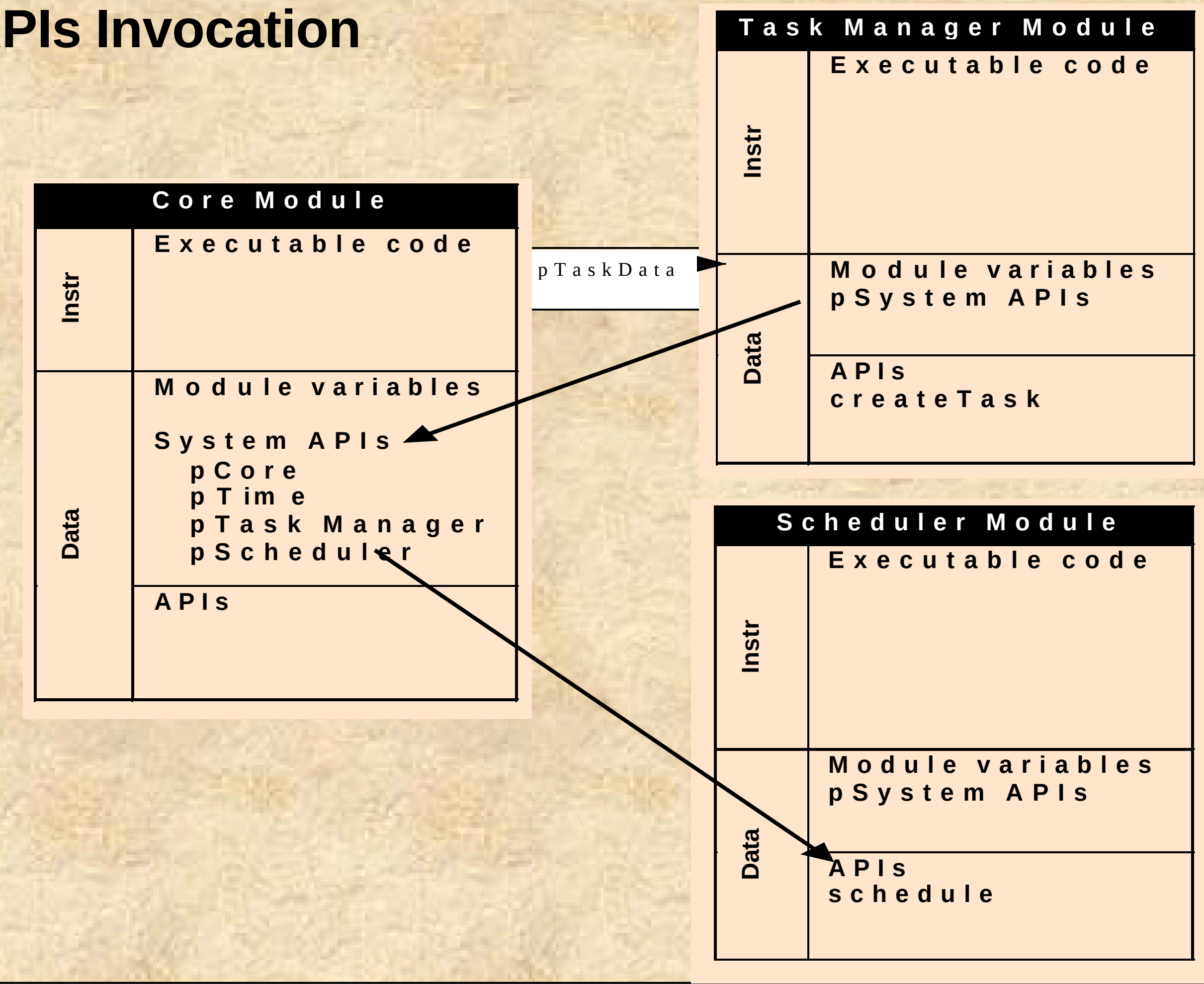
Module Installation





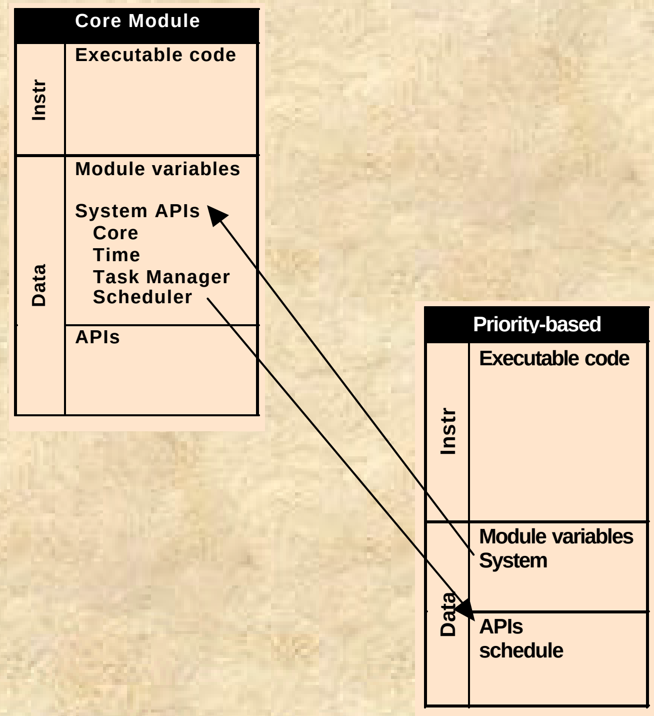
DRTOS **Technical Approach (continued)**

APIs Invocation



DRTOS **Technical Approach (continued)**

Updating Scheduler module



Load a priority-based scheduler

Link the priority-based scheduler to the core module

Unlink the round robin scheduler

The round robin scheduler can be deleted from the memory

DRTOS **Technical Approach (continued)**

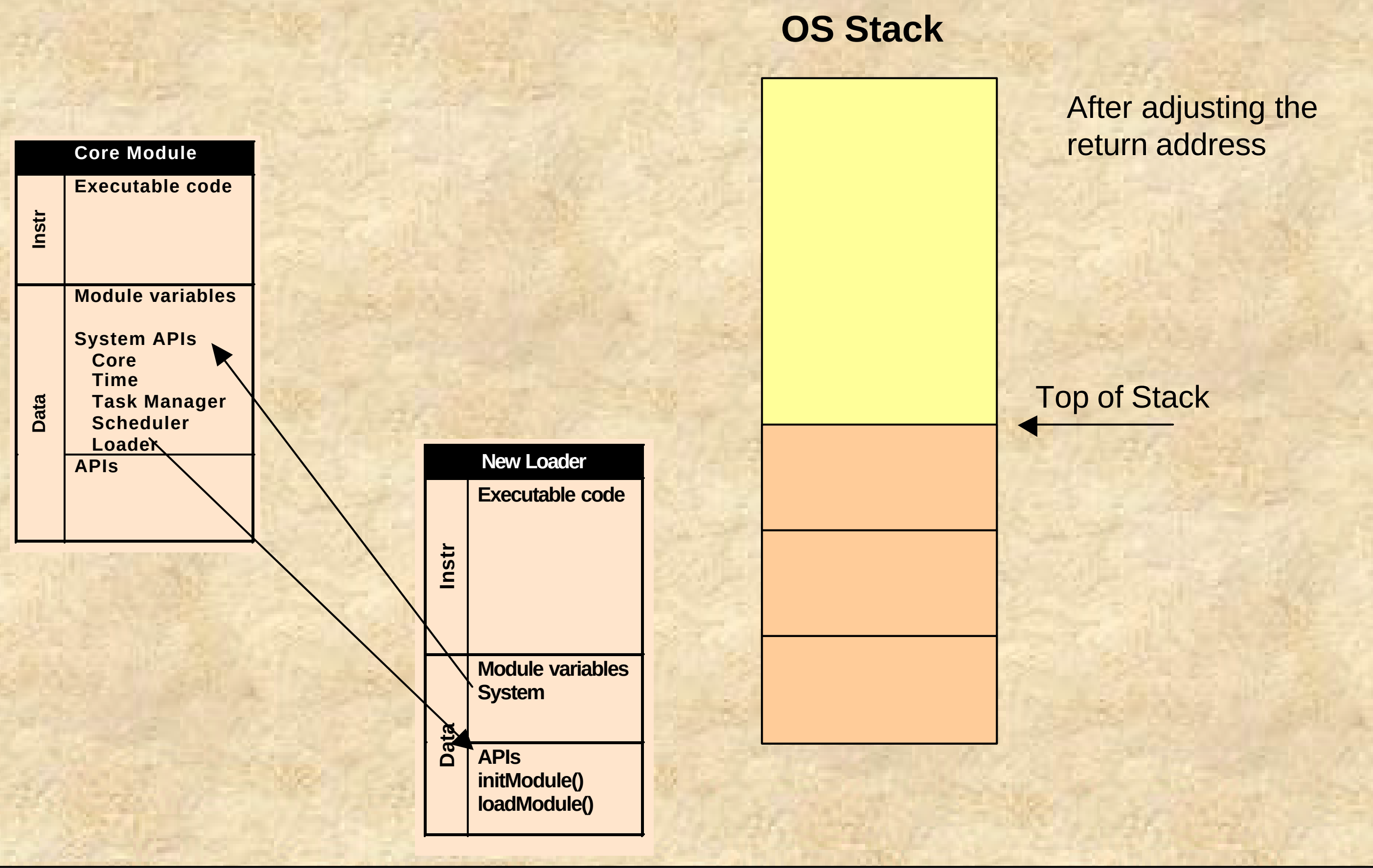
- Function pointers are used for APIs
- Invocation of a scheduler API
  - `System *pSys = (System *) pTaskData->pSystem;`
  - `SchedulerMethod *scheduler = (SchedulerMethod *) pSys->scheduler;`
  - `scheduler->schedule(pNewTask, TASK_READY);`
- Updating Core module
  - Either (1) System variable must either be in the same location as before
  - Or (2) each module must be notified when Core module is updated (all modules’ System variables must point to the new location for the System variable).

DRTOS **Technical Approach (continued)**

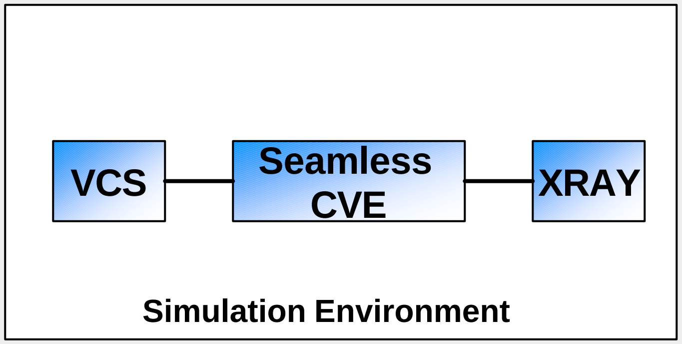
- Updating Loader module
  - The current loader module is called to update to the new loader module.
  - The *initModule()* function of the new loader module is invoked.
  - The return address from the *initModule()* function must be adjusted to the location which calls the update API of the old loader module by clearing the stack to ignore the call from the old loader module.
  - The old loader module can be deleted from the memory.

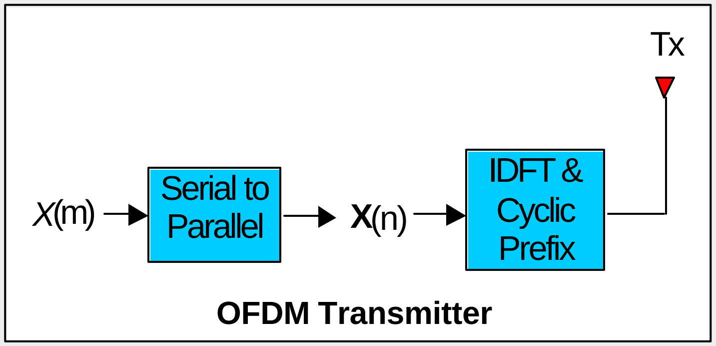
DRTOS **Technical Approach (continued)**

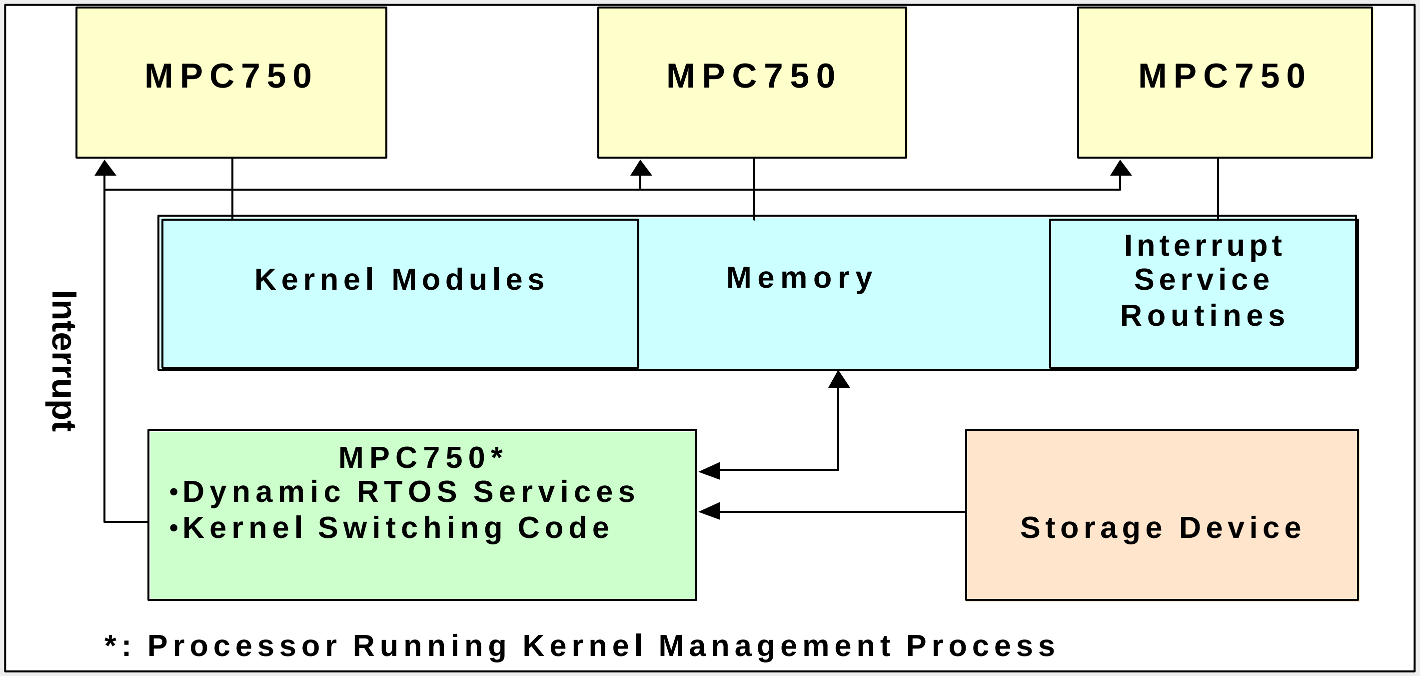
Updating Loader module



Experiment 1



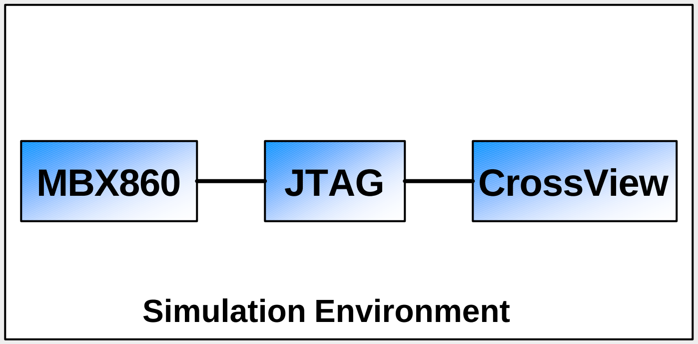


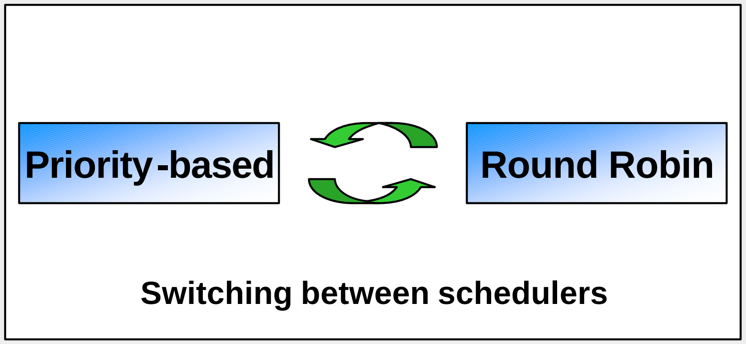


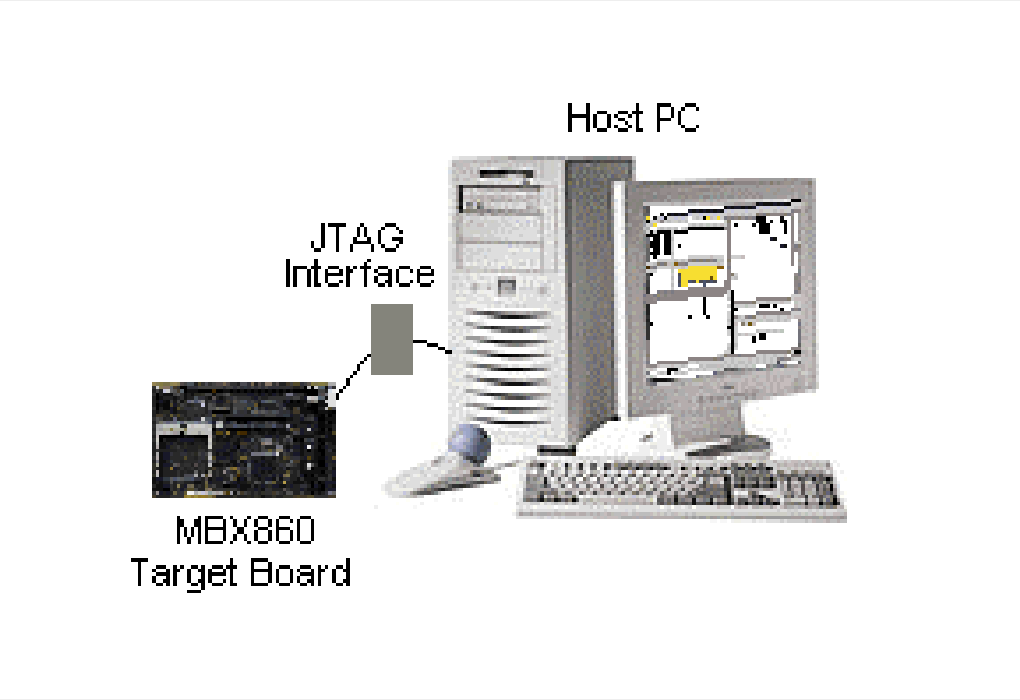
Experiment 1 (continued)

- Initially running VUI code
  - uses round-robin scheduler
- Want to change to OFDM code
  - install new I/O code
  - install new priority scheduler code
- OFDM Code Size: 1600 lines of code
- Time to load (VUI still operational): 4 kbytes x 2 cycles/byte
- Time to switch to new DRTOS code: 60 cycles

Experiment 2



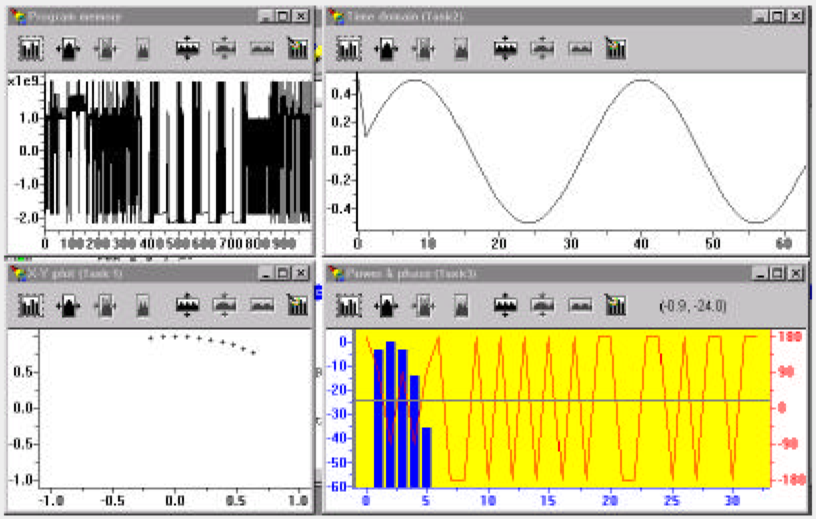




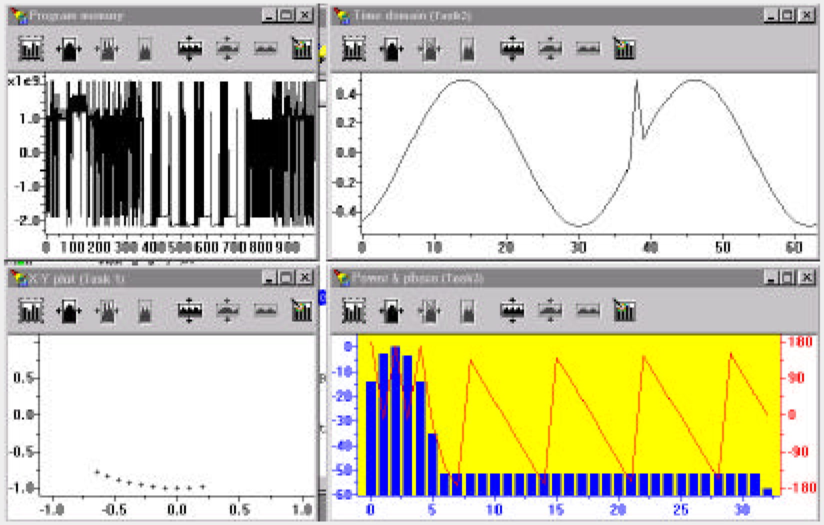
Experiment 2 (continued)

- Three tasks
  - DRTOS uses priority-based scheduler
- Change to round robin scheduler
  - install round robin scheduler code
  - migrate tasks from the previous scheduler
- Round Robin Code Size: 200 lines of code
- Switching Time: 60 + 8n assembly instructions (n = number of tasks currently in the system, the scheduler needs to poll each task to get its handle)

Experiment 2 (continued)



Priority-based scheduler



Round robin scheduler

Conclusion

- Existing real-time operating systems not fully dynamic
- The needs of a new real-time operating system architecture to support emerging applications
- Our approach: the Georgia Tech DRTOS
- Initial experiments and results