Verifying Data Race Freedom of Kernel APIs in a Real Time Operating System

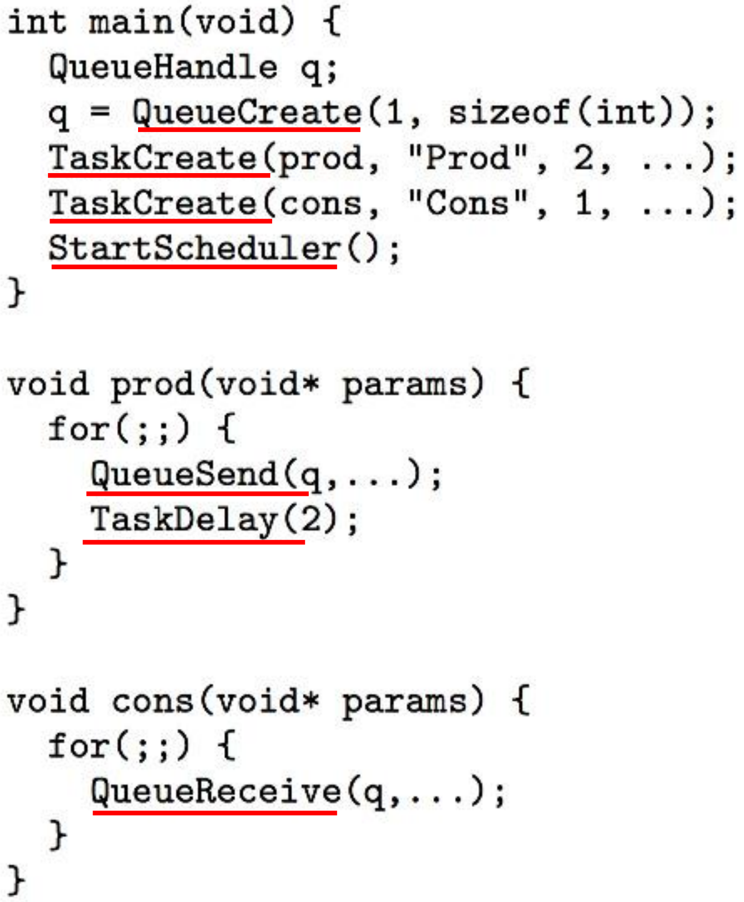
Suvam Mukherjee
Advisor: Prof. Deepak D'Souza
Computer Science and Automation, Indian Institute of Science

OUTLINE

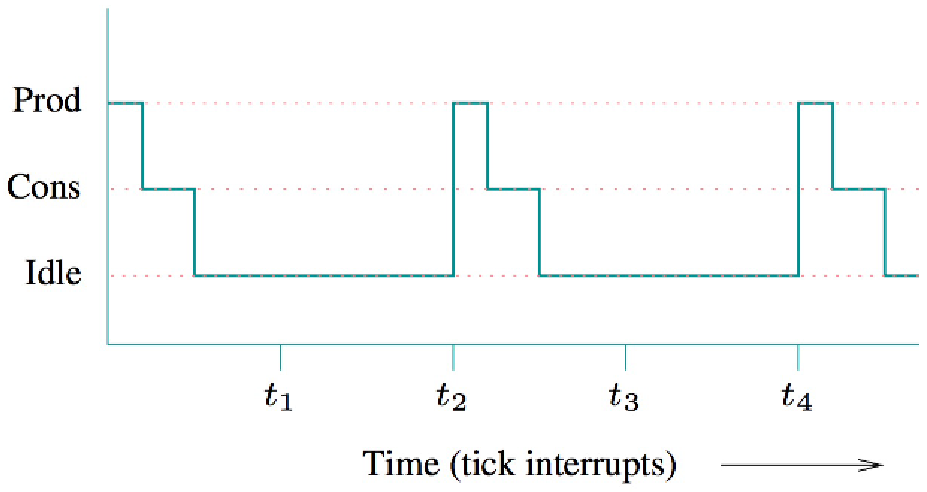
- Problem Definition
- Proposed Solution
- A Case Study: FreeRTOS
- Experimental Evaluation
- Conclusion and Future Work

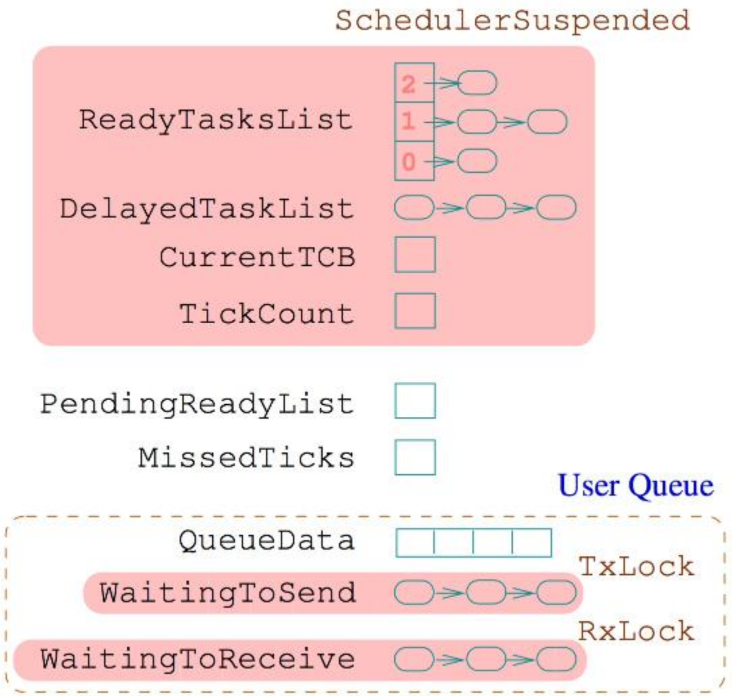
Problem Definition

Verifying Data Race Freedom of **Kernel APIs** in a Real Time Operating System



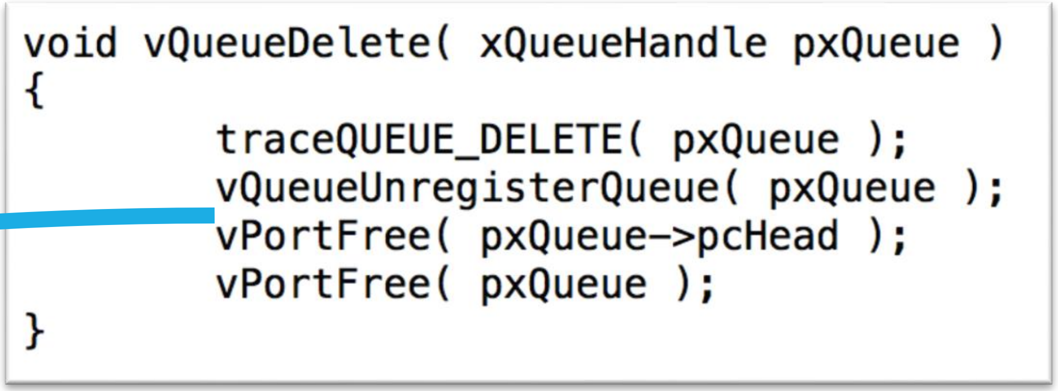
```
int main(void) {
  QueueHandle q;
  q = QueueCreate(1, sizeof(int));
  TaskCreate(prod, "Prod", 2, ...);
  TaskCreate(cons, "Cons", 1, ...);
  StartScheduler();
}

void prod(void* params) {
  for(;;) {
    QueueSend(q,...);
    TaskDelay(2);
  }
}

void cons(void* params) {
  for(;;) {
    QueueReceive(q,...);
  }
}
```



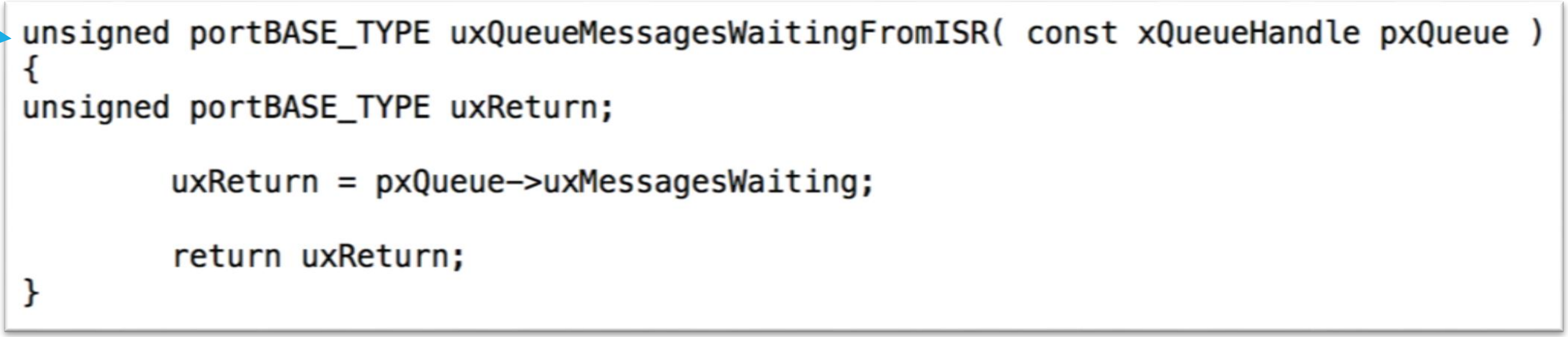


Problem Definition

Verifying **Data Race Freedom** of Kernel APIs in a Real Time Operating System



```
void vQueueDelete( xQueueHandle pxQueue )
{
        traceQUEUE_DELETE( pxQueue );
        vQueueUnregisterQueue( pxQueue );
        vPortFree( pxQueue->pcHead );
        vPortFree( pxQueue );
}
```



```
unsigned portBASE_TYPE uxQueueMessagesWaitingFromISR( const xQueueHandle pxQueue )
{
unsigned portBASE_TYPE uxReturn;

        uxReturn = pxQueue->uxMessagesWaiting;

        return uxReturn;
}
```

Problem Definition

Verifying Data Race Freedom of Kernel APIs in a Real Time Operating System

❑ Guarantees for any application with an arbitrary number of tasks

(unlike bug-finding)

❑ Helps to create a version of the RTOS certified against data races

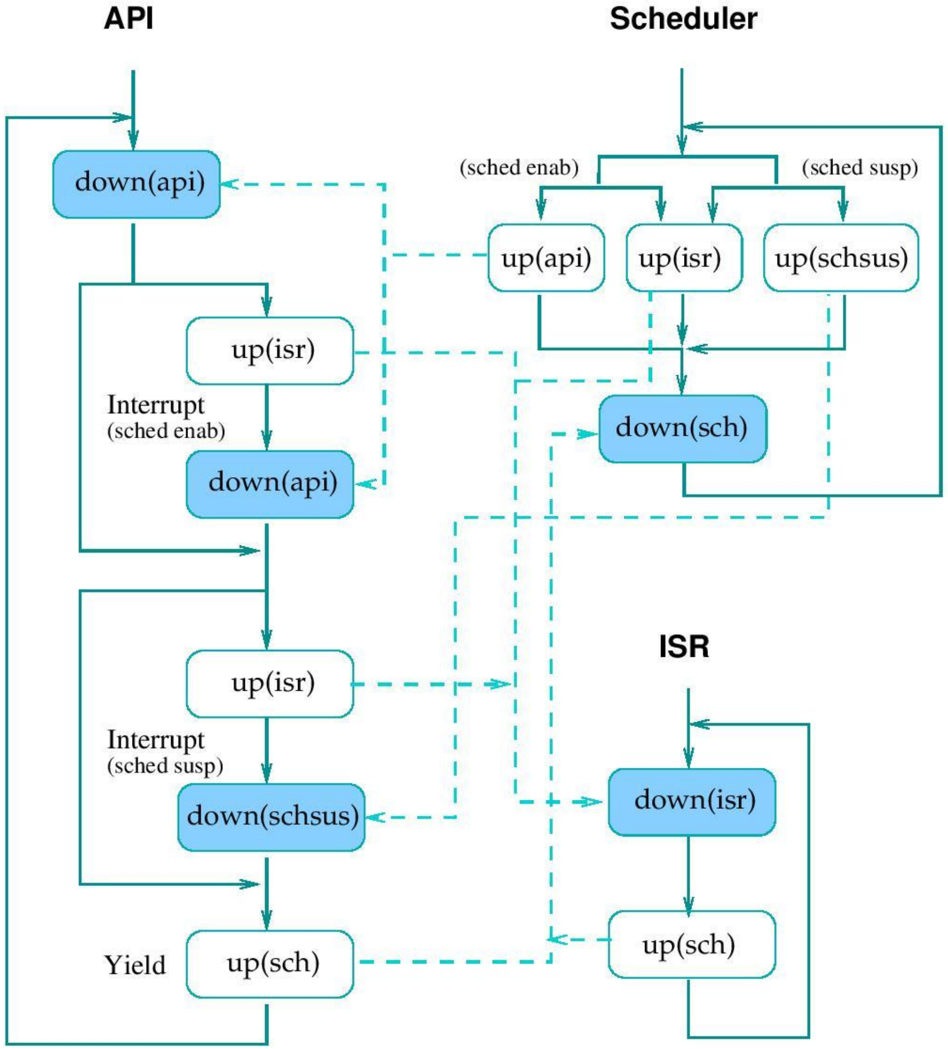
Proposed Solution

1. Model control flow
2. Model accesses to shared data structures
3. Perform suitable abstractions
4. Model check a small number of *reduced* models
    - Enhances scalability
    - Preserves soundness guarantees

A Case Study: FreeRTOS

- One of the most popular real time operating systems
- Over 100,000 downloads in 2014 alone
- Uses a preemptive flag-based and priority-based scheduling policy
- Rich set of APIs performing a wide variety of operations
  - Creating tasks,
  - Creating queues,
  - Communication between tasks, and many more
- Presence of interrupts
  - Specific set of functions which interrupt handlers can invoke

A Case Study: FreeRTOS

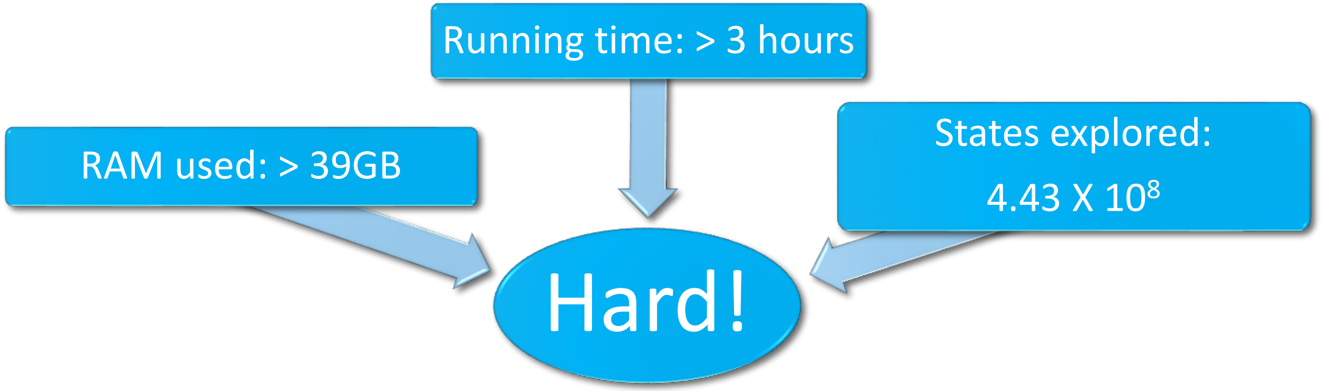


Courtesy: Prof. Deepak D'Souza

Experimental Evaluation

Model Checking M2

- On a system with 128GB RAM, 2 X (8-core Intel Xeon Haswell 2.6GHz) system
- With SPIN optimizations enabled



Running time: > 3 hours

RAM used: > 39GB

States explored: 4.43 X 10^8

Hard!

Model Checking with Reduction

- Reduced model
  - Process 1: API
  - Process 2: API
  - Process 3: ISR
  - Process 4: Tick Interrupt
  - Process 5: Scheduler
- 2023 Reduced Models (17 APIs, 7 ISRs)
- System Used: 32 GB RAM, Intel Core i7 Quad-Core 3.40GHz, Ubuntu 14.04

Iteration	# Violations	FP	Harmful	Benign	Time (hrs)
1	40	10	24	6	1.5
2	0	-	-	-	1.35

Conclusion and Future Work

- Proposed an approach to model and exhaustively check a library of Kernel APIs in an RTOS for data races
- The proposed steps:
  - Model control flow and access to shared data structures
  - Perform suitable abstractions
  - For scalability, model check a small number of reduced models
- Concrete instantiation of our approach
  - Modelled concurrency behaviors of FreeRTOS Kernel APIs and ISRs
  - Model checked 2023 reduced models in under 2 hours
  - Detected 30 data races and classified them as harmful or benign.
  - Created a certified race-free version of FreeRTOS

- Carry out further instantiations, for example, OSEK, `java.util.concurrent` etc.
- Identify general patterns which allow model checking of small set of reduced models

---