Reducing Context Switching with Preemption-Threshold™

John Carbone
Express Logic, Inc.

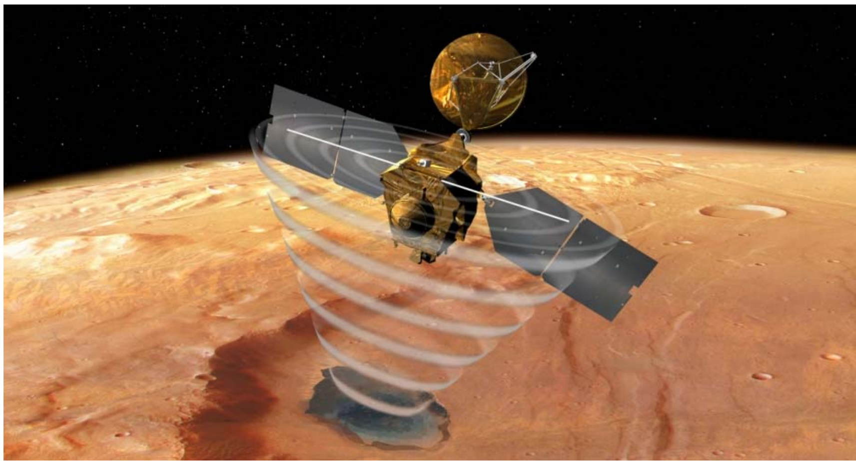
Presenter: John A. Carbone

- VP, Marketing, Express Logic, Inc.
  - Responsible for product and corporate marketing, partner relationships, technical articles, and technical training.
  - Presenter at various industry conferences
  - Authored technical papers on real-time multithreading, certification, and measurement of real-time performance

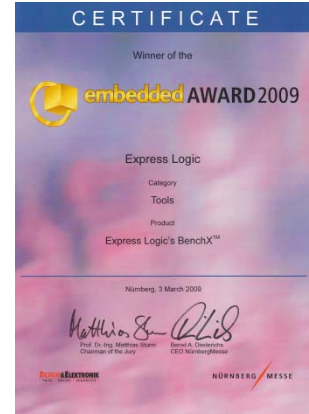
PREVIOUS EXPERIENCE:

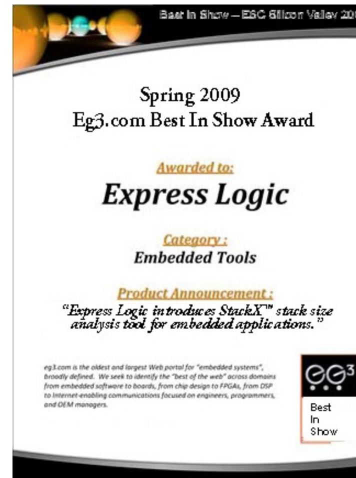
- VP, Marketing at Green Hills Software
- Embedded developer and FAE
- Member of the IEEE
- BA, Mathematics, Boston College

Express Logic Innovation



embedded AWARD 2009



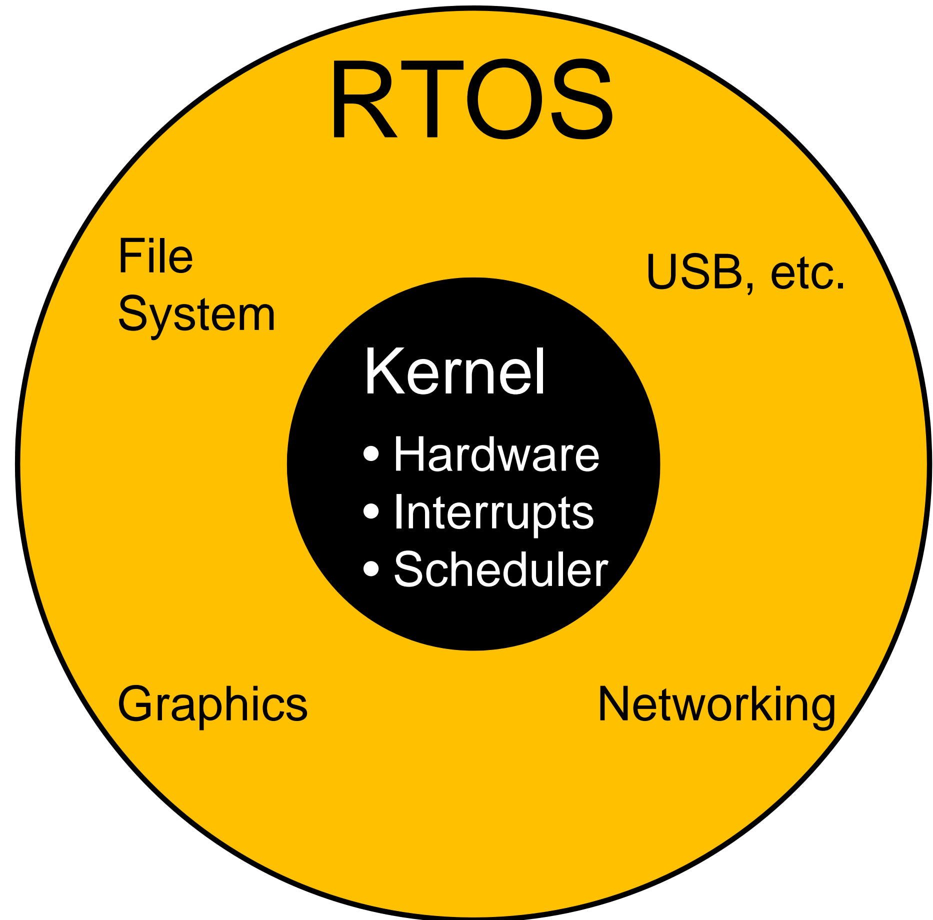


Agenda

- Part I - Key Concepts
  - What Is An RTOS
  - Benefits of an RTOS
  - RTOS services
  - Types of Scheduling
  - Multithreading
  - Preemptive Scheduling
  - Preemption-Threshold™
- Part II – An Example
  - ThreadX® RTOS
  - A Test To See The Effect of Preemption Threshold
  - Assessing The Results
  - Summary and Conclusion
- Q/A
- More Information

What Is An RTOS?

- RTOS
  - What is an RTOS?
    - Kernel + X, Y, Z, ...
  - What does an RTOS do for us?
    - Manages real-time applications
  - RTOS Services
    - Scheduler
    - Threads
    - Timers
    - Message Queues
    - Semaphores
    - Mutexes
    - Memory Pools



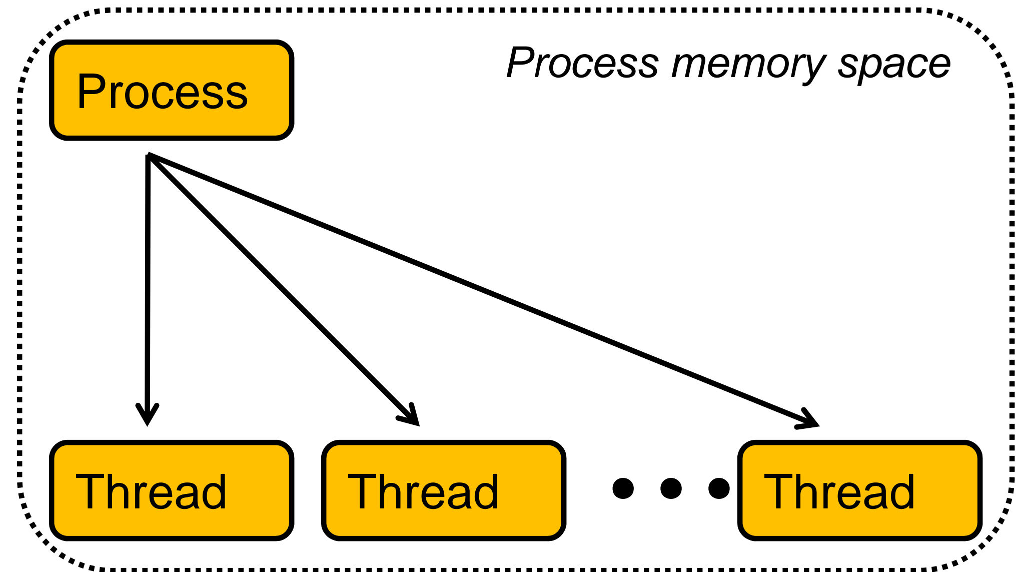
expresslogic

Benefits of An RTOS

- Reclaim CPU cycles – lower overhead
  - Polling keeps CPU at 100%
  - Sleep(), Intra-thread activation, Interrupts reclaim almost all polling cycles
- Easily add new threads
  - Modular expansion through threads
- Provides platform for adding middleware
  - TCP/IP Stack
  - USB Stack
  - Graphics
  - Event Trace
- See Article, "*Multitasking Mysteries*"

Threads and Priorities

- Threads
  - What is a thread?
    - Semi-independent program segment
    - Share same memory space
    - Run "concurrently"
  - How are threads used?
    - Modularize a program
    - Minimize stalls
  - Thread Services
    - Create, Suspend, Relinquish, Terminate, Exit, Prioritize
  - Thread States
    - READY, RUNNING, SUSPENDED, TERMINATED



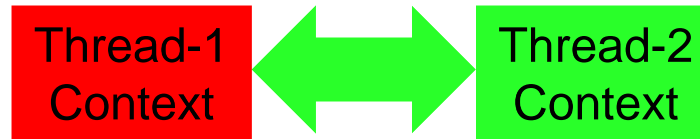
- Thread Priorities
  - Often 0-n, with 0 highest
  - Dynamic or Static
  - Equal priorities
    - Multiple threads at same priority
  - Unique priorities
    - Each thread has unique priority

Priority	
0	*Highest*
1	
2	
...	
n	*Lowest*

expresslogic

ThreadX RTOS Services

- Context Switch

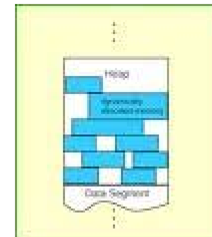


- Timers
  - Unlimited, one-shot, repeatable

- Message Passing
  - Queues, send, receive, pend

- Semaphores/Mutexes
  - Priority inheritance optional

- Memory Management
  - Byte and block pool allocation



expresslogic

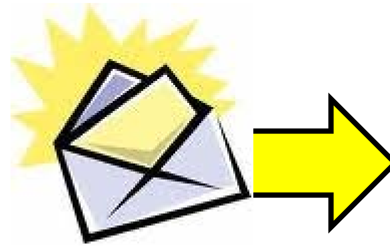
Context Switch

- Thread Context
  - Information critical to thread's operation
  - Register Contents, Program Counter, Stack Pointer
  - Saved when thread is preempted
  - Restored when thread is resumed

- Context Switch
  - Interrupt running thread and do something else
  - Result of preemption, interrupt, or cooperative service
- What's involved in a context switch?
  - See

Step	Operation	Cycles
1	Save the current thread's context (ie: GP and FP register values and PC) on the stack.	20 - 100
2	Save the current stack pointer in the thread's control block.	2 - 20
3	Switch to the system stack pointer.	2 - 20
4	Return to the scheduler.	2 - 20
5	Find the highest priority thread that is ready to run.	2 - 50
6	Switch to the new thread's stack.	2 - 50
7	Recover the new thread's context.	20 - 100
8	Return to the new thread at its previous PC.	2 - 40
9	Other processing	0 - 100
	TOTAL	**50 - 500**

Message Queues



- What is a Message Queue?
  - Data structure that holds messages

messages inserted
at rear of queue

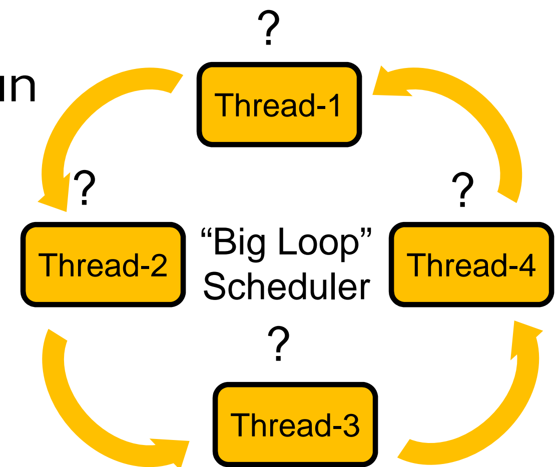
messages removed
from front of queue

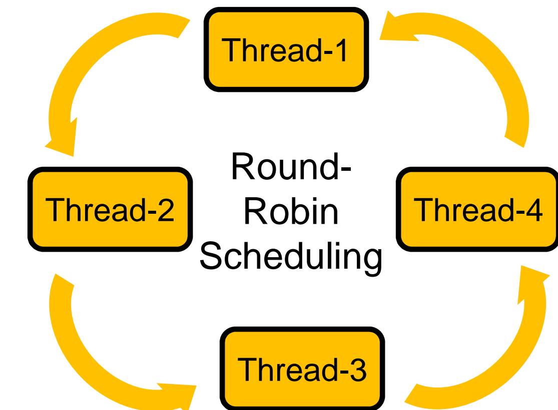
  - A message is a 32-bit word, or a pointer to a larger array of information
  - Means of message-passing among threads
  - Messages usually are inserted at rear of queue (FIFO) but can be inserted at front of queue if desired (LIFO)
  - Messages are removed from front of queue
  - Public resource—any thread can access any queue
- Why Use Message Queues?
  - To send data from thread to thread
  - To notify a thread that an event has occurred
  - Threads will suspend on queue full and queue empty

expresslogic

Types of Schedulers

- Big Loop Scheduling
  - Each thread is polled to see if it needs to run
  - Polling proceeds sequentially, or in priority order
  - Inefficient, lacks responsiveness
- RTOS Scheduler
  - Controls which thread is allowed to run
  - Performs context switches
  - Provides thread services
    - Sleep
    - Relinquish
    - Terminate
- Round-Robin Scheduling
  - Cycle through multiple "READY" threads
  - And/or impose "time-slice" for each thread
  - A bit better than the loop, but still not very responsive

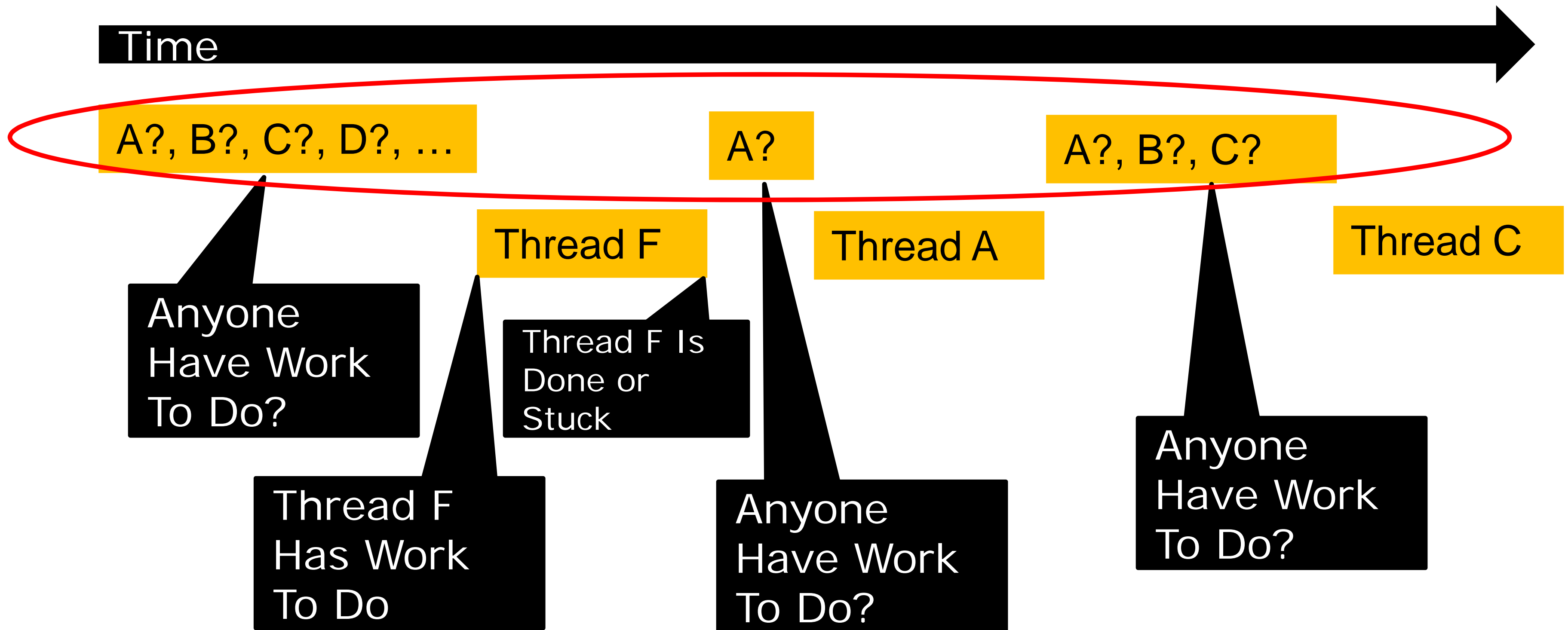




expresslogic

The Big Loop

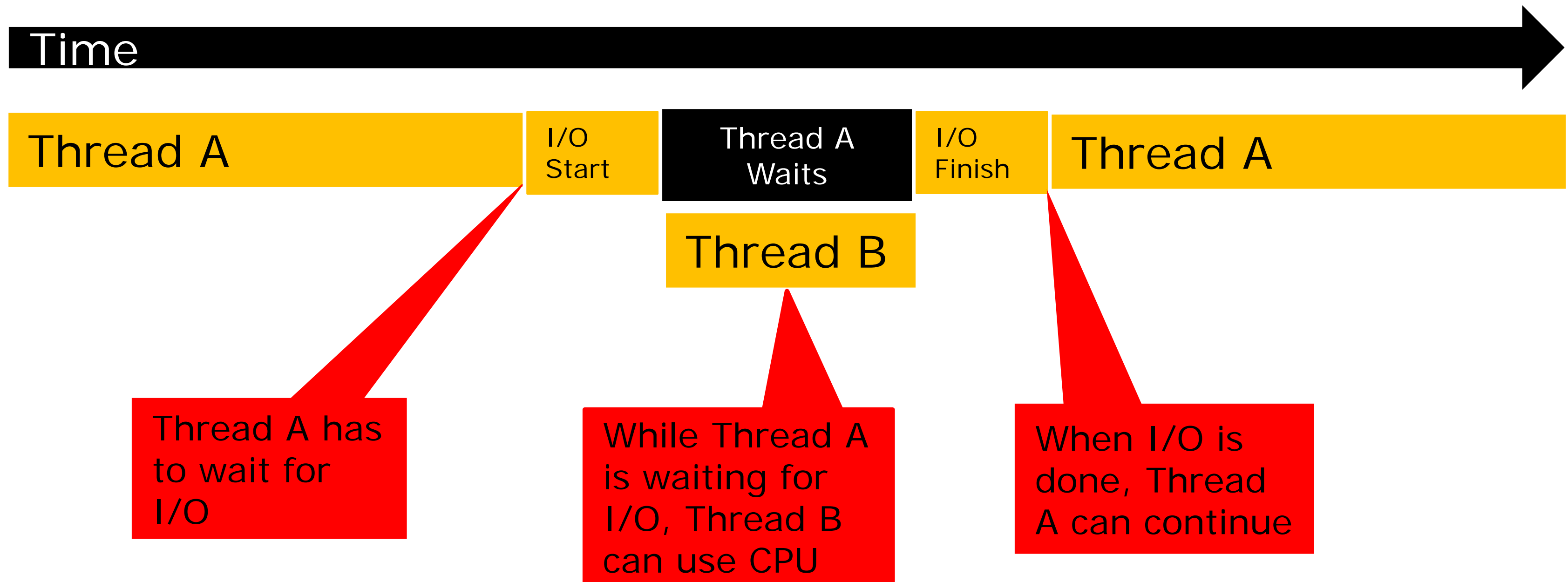
- CPU time is spent checking to see if any activity (thread) has work to do



expresslogic

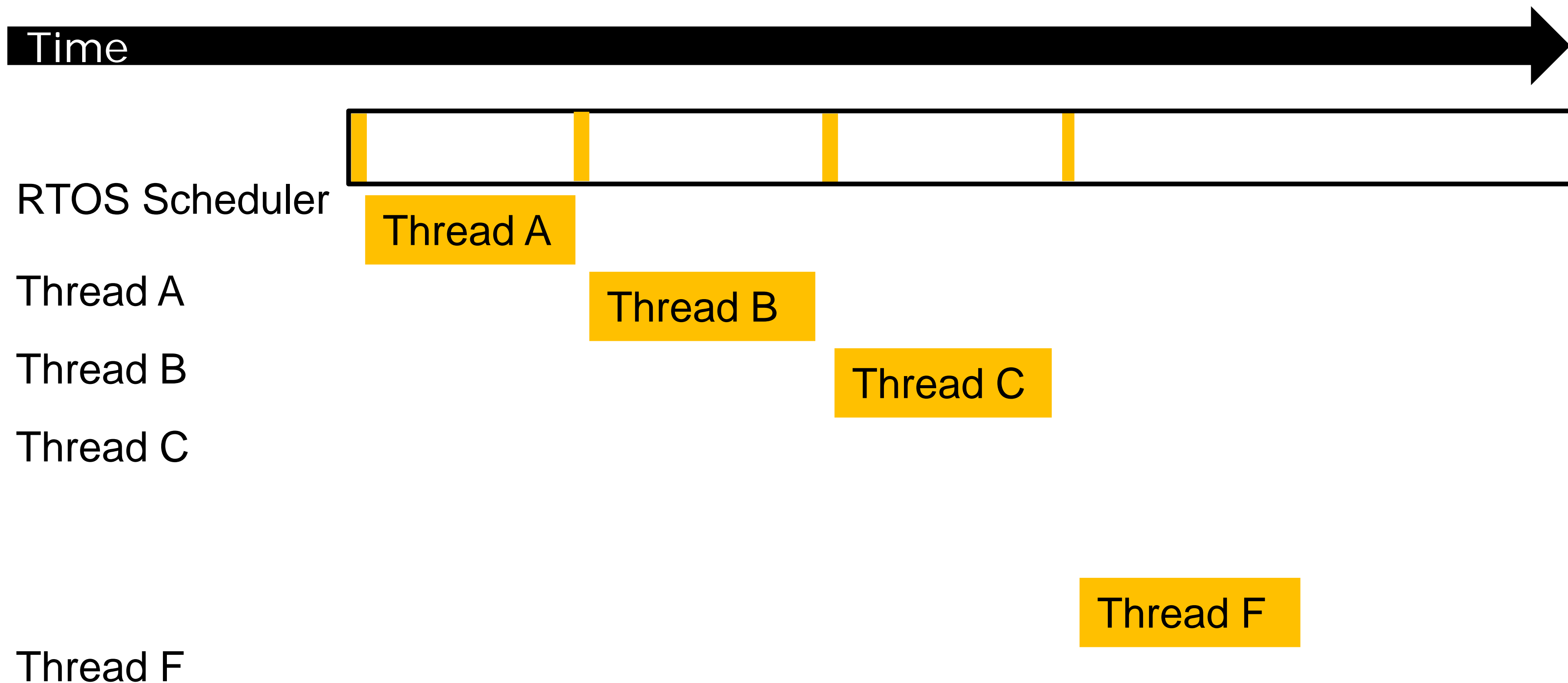
Multithreading

- Enabling an activity to use the CPU while other activities don't need it – I/O Delay



RTOS Scheduling

- Implement multithreading by keeping track of thread states and activate threads with work to do



expresslogic

"RTOS-izing" Code

- Stand-alone code generally uses "event loops" to run functions

```
While(1) {
            if (condition_1) {
               function_1()
            else if (condition_2)
               function_2()
            else if (condition_3)
               function_3()
            …
            …
            …
            endif;
}
```

Time to evaluate each "condition_n" expression plus decide and branch, adds up – delays response to "condition_x". Plus, any new conditions or functions change timing of loop.

- With an RTOS
  - Run highest priority function (task/thread)
  - When that thread must wait for an event, thread suspends until "event" occurs
  - Enables other threads to get CPU cycles
  - Event triggers interrupt. ISR calls scheduler
  - Scheduler performs context switch
- Result is faster response and better use of time

expresslogic

"RTOS-izing" Code

- Stand-alone code often uses "Delay Loops"

```
For i=1, i<10000, i++ {
            ...........
            end;
}
```

Spin loop occupies CPU 100% for duration of delay period.

- Replace with call to tx_thread_sleep(n)

```
tx_thread_sleep (1);
```

Sleep call frees up CPU for other threads or for low-power operation.

- Enables other (READY) threads to get CPU cycles
- In "n" timer ticks (can be any user-defined duration), suspended thread re-awakens
- Result is better use of delay time by other threads

expresslogic

Results With Big Loop and Spin-Loop Delays – No RTOS

File View Help

500 | 0 | Event | 0 Delta Ticks

Sequential View | Time View

Context Summary
Event Summary
Time (Ticks)

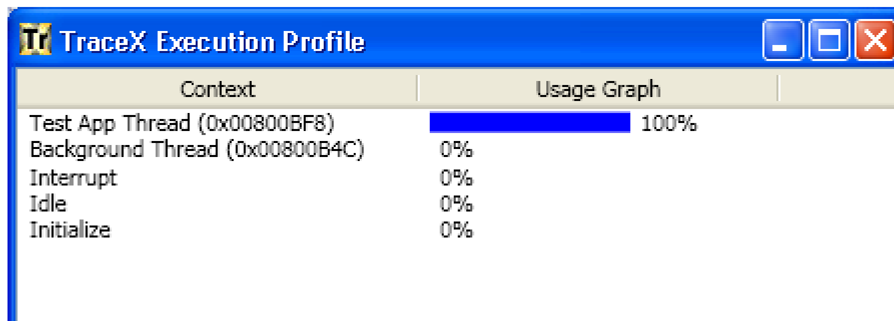
Interrupt

0 28800 57600 86400 115200 144000 172800 201600 230400 259200

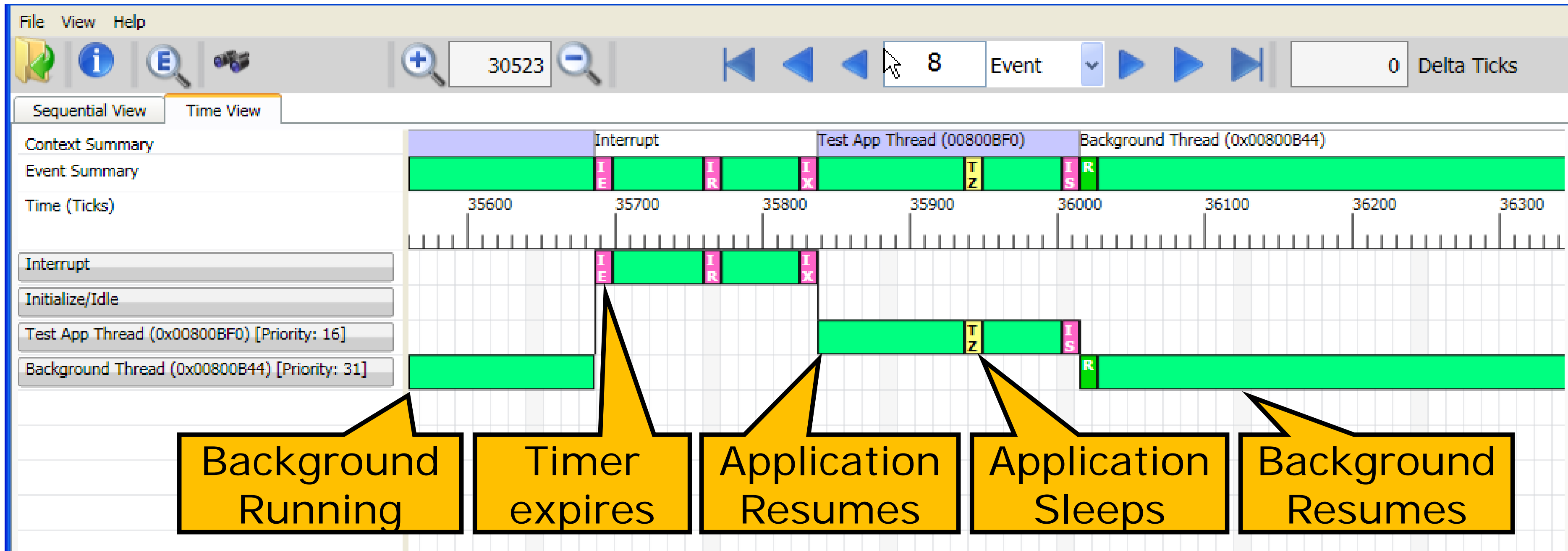
Core 0
Interrupt
Initialize/Idle
Test App Thread (0x00800BF8) [Priority: 16]
Background Thread (0x00800B4C) [Priority: 31]

Registered to: Express Logic

Seats Purchased: 0



Using an RTOS for Multithreading



Results With RTOS Multithreading

File View Help

1050 0 Event 0 Delta Ticks

Sequential View | Time View

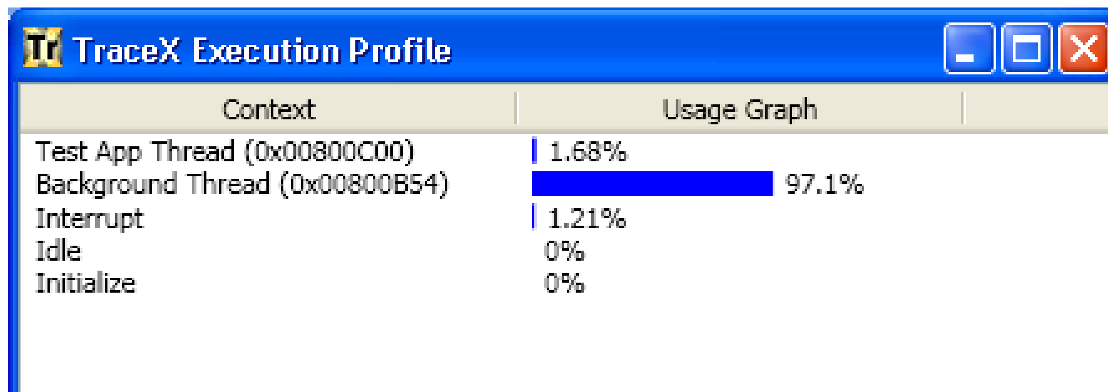
Context Summary
Event Summary
Time (Ticks)

c0:Background Thread (0x00800B4C)
c0:Background Thread (0x00800B4C)

10000 12500 15000 17500 20000 22500 25000 27500 30000 32500

Core 0
Interrupt
Initialize/Idle
Test App Thread (0x00800BF8) [Priority: 16]
Background Thread (0x00800B4C) [Priority: 31]

Registered to: Express Logic
Seats Purchased: 0

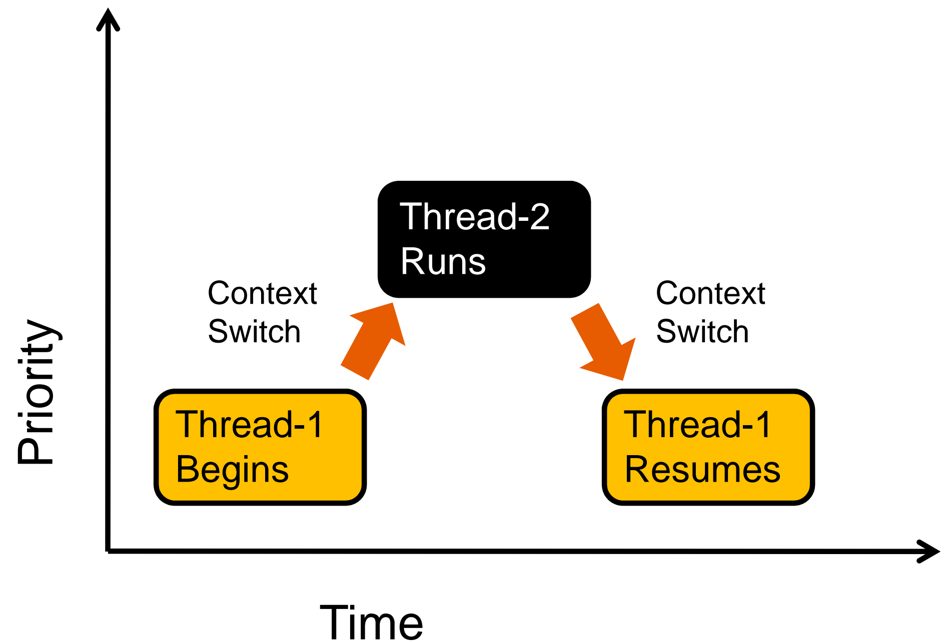


TraceX Execution Profile

Context	Usage Graph
Test App Thread (0x00800C00)	1.68%
Background Thread (0x00800B54)	97.1%
Interrupt	1.21%
Idle	0%
Initialize	0%

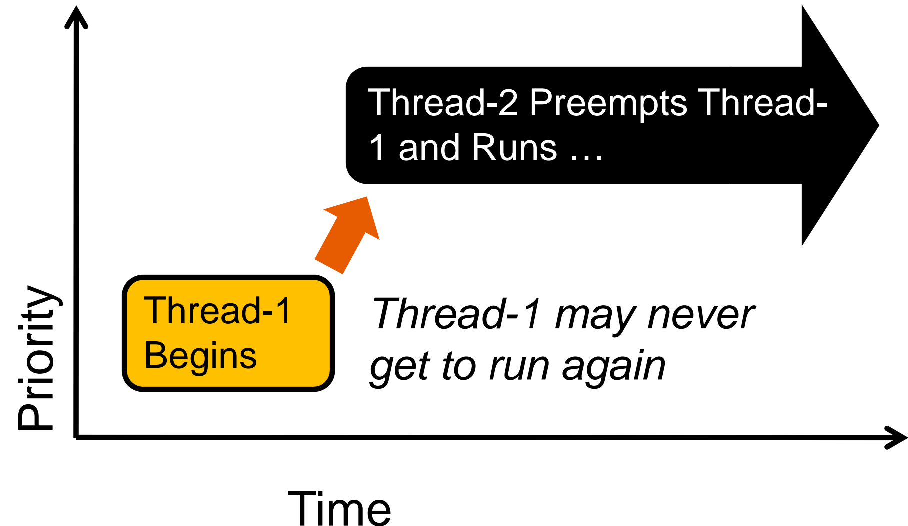
Preemptive Scheduling

- Preemption
  - Interruption for higher-priority activity
    - Interrupt
    - Thread
- Preemptive Scheduling
  - Always run highest priority thread that is READY to run
    - Maximum responsiveness
    - No Polling, so more efficient
    - Always results in a context switch



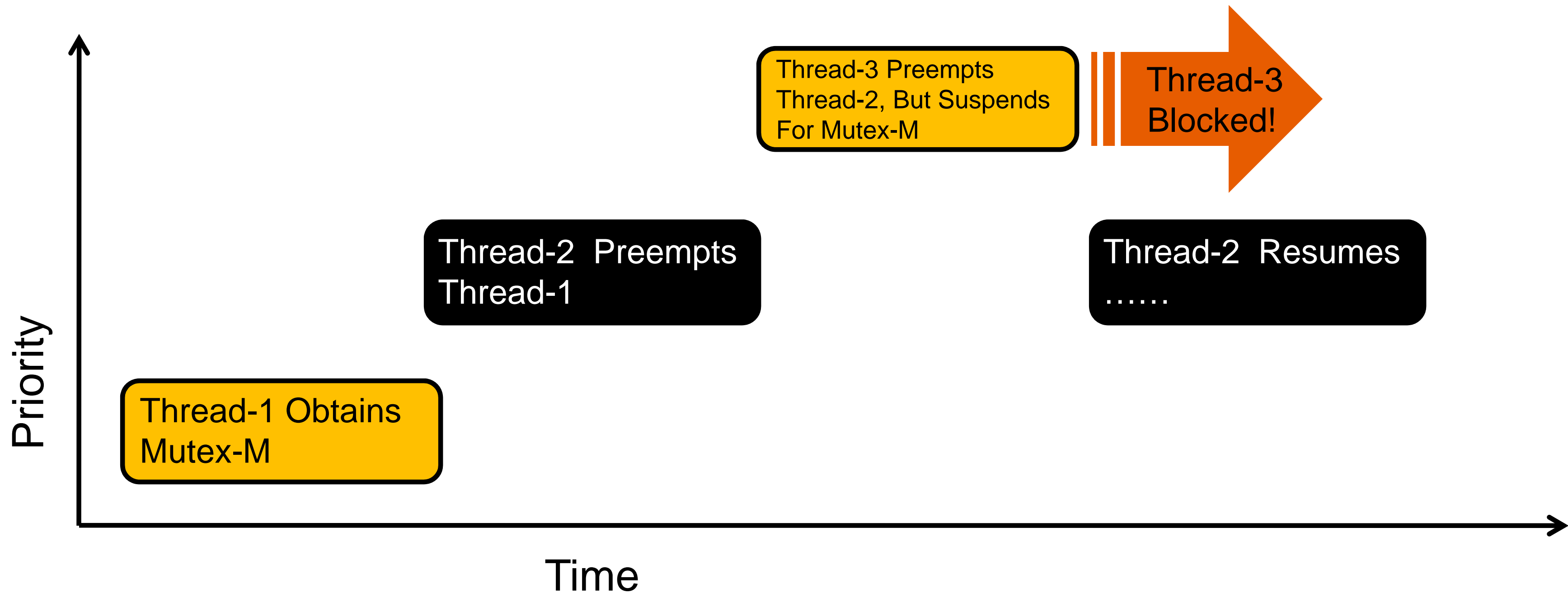
Preemptive Problems

- Thread Starvation
  - If a higher-priority thread is always ready, the lower priority threads never execute
- Excessive Overhead
  - From context switching
  - The subject of our demo
- Priority Inversion
  - Higher-priority thread can be suspended because a lower-priority thread has a needed resource – see following …



expresslogic

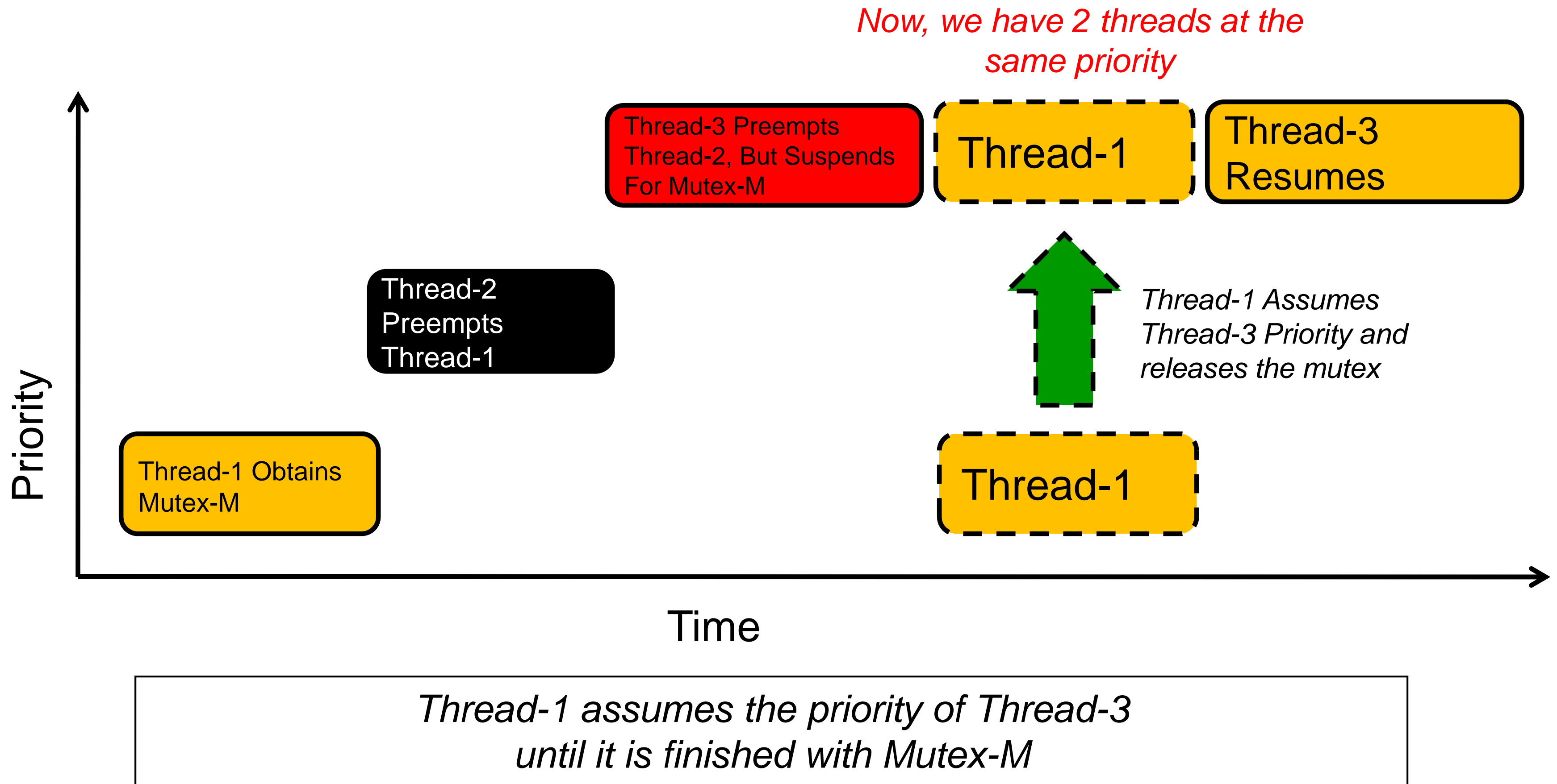
Priority Inversion



Even though Thread-3 has the highest priority, it must wait for Thread-2. Thus, priorities have become inverted.

expresslogic

Priority Inheritance



Thread-1 assumes the priority of Thread-3 until it is finished with Mutex-M

expresslogic

Preemption-Threshold™

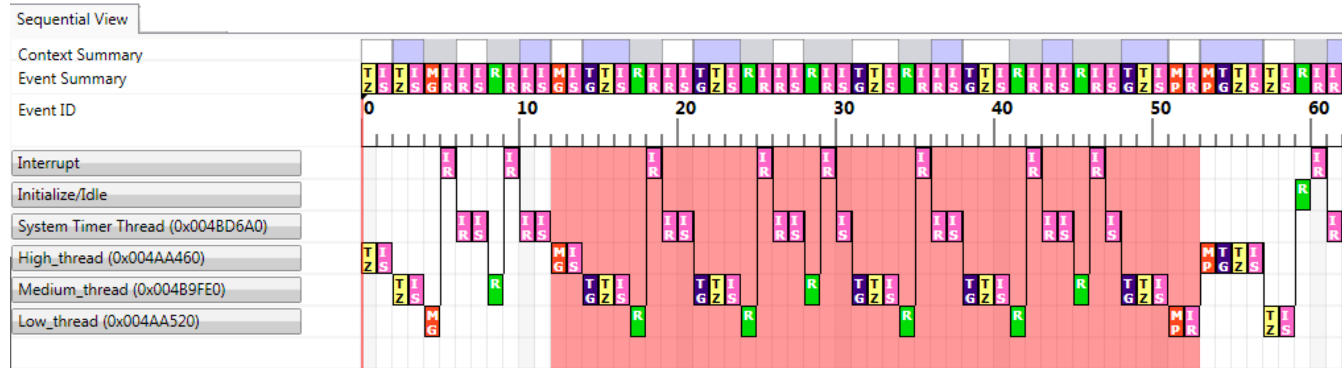
- A technique to reduce context switches
- **Preemption-Threshold** establishes a priority ceiling for disabling preemption – preemption requires a priority higher (lower number) than the ceiling
- For example, assume a thread's priority is 20, and its preemption threshold is set to 15
- Threads with priority lower than (larger number) 14 , even if higher than (smaller number) the running thread's priority (20), will not preempt the running thread

Priority	Comment
0	Preemption allowed for threads with priorities from 0 to 14 (inclusive)
:	
14	
15	Thread is **assigned Preemption-threshold** = **15** [this has the effect of disabling preemption for threads with priority values from 15 to 19 (inclusive)]
:	
19	
20	Thread is assigned **Priority** = **20**
:	
31	

expresslogic

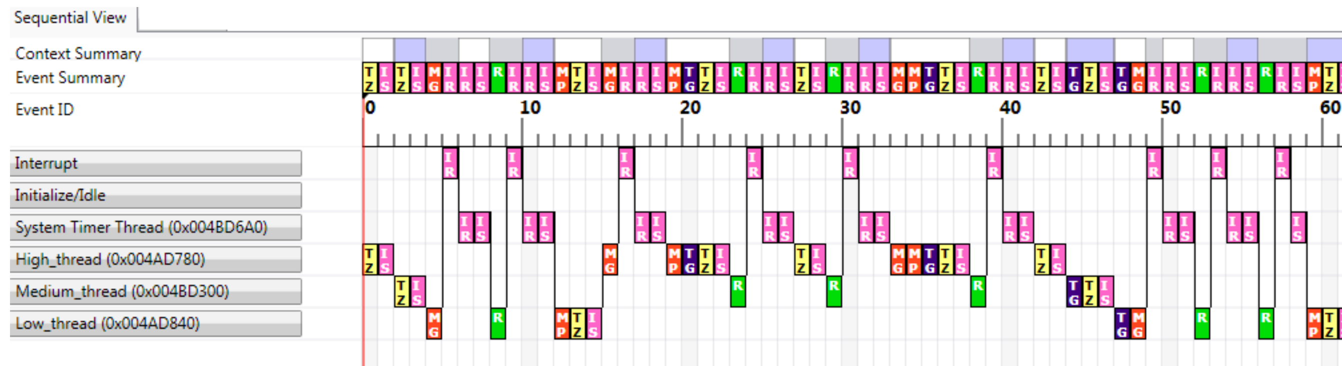
Preemption-Threshold Can Help Avoid Priority Inversion

- Without Preemption-Threshold



The shaded area identifies priority inversion
Without Preemption-Threshold, High_thread must wait for Medium_thread, which has a lower priority

- With Preemption-Threshold



With Preemption-Threshold, no priority inversion has been detected
High_thread obtains the mutex without waiting for Medium_thread

Preemption-Threshold Improves Efficiency

Relative Time Between Mutex_Get and Mutex_Put Pairs for High_thread

	Average Time	Minimum Time	Maximum Time	Number of Get-Put Pairs
Without Preemption-Threshold	40.1	2	104	17
With Preemption-Threshold	2.0	2	2	52

Number of Priority Inversions For Test Case

	Number of Non-Deterministic Priority Inversions
Without Preemption-Threshold	9
With Preemption-Threshold	0

For more information, see academic research:
http://www.cs.utah.edu/~regehr/reading/open_papers/preempt_thresh.pdf

© 2010 Express Logic, Inc.

expresslogic

Part-II

- An example to show how Preemption-Threshold can reduce context switches
- Based on ThreadX RTOS
- And TraceX analysis tool

expresslogic

Express Logic's ThreadX RTOS

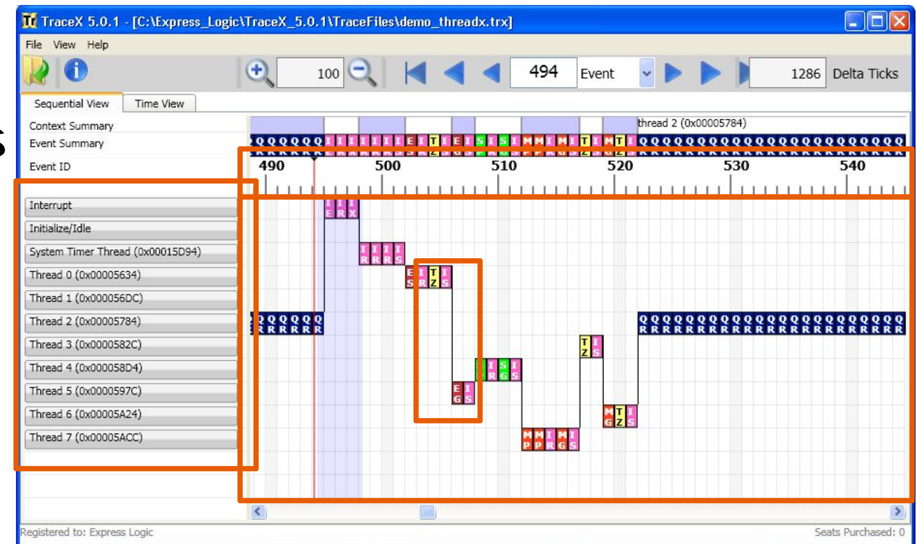
- Small, fast, easy-to-use RTOS for hard real-time applications. Widely used in Medical, Consumer, Industrial markets. FDA 501(k) and IEC-62304 approved.
- Footprint: 2KB – 15KB
  - Automatically scales based on services used
- Speed: ~ 1-2ms context switch
  - Most services 50-200 cycles
- API: Intuitive, 66 functions in 8 categories

ThreadX Technology

- Picokernel™ Architecture
  - Non-layered implementation for size & speed
  - Deterministic processing, not affected by number of application objects
  - Automatic scaling
- Preemption-Threshold™
  - Scheduling technology that reduces overhead
- Performance metrics
  - Counts various system events and operations (context switches, etc.)
  - Execution Profile Kit
- File system, Network stack, USB stack host/device/OTG, Graphics
  - Full featured, fully integrated
- TraceX and StackX development tools
  - Innovative tools for real-time systems
- Optimized Interrupt Processing
  - Only scratch registers saved/restored if no preemption
  - No idle thread, hence no context save/restore when system is idle
  - Most of API available directly from ISRs
  - Optional timer thread or direct timer processing in ISR

TraceX

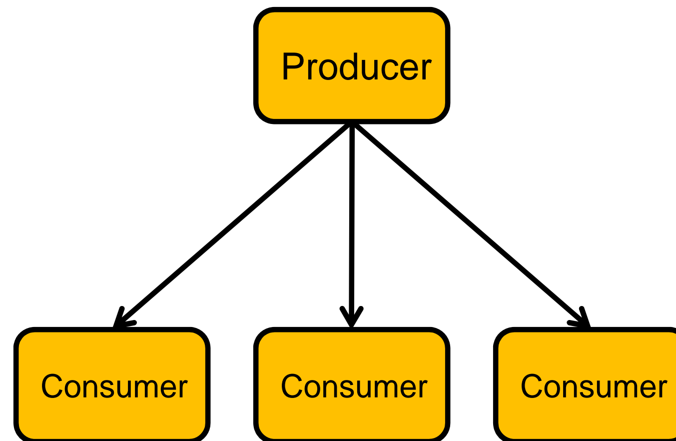
- Analysis tool to examine system “events”
- RTOS logs “events” in trace buffer in target memory
  - Events include RTOS services like “queue_send”, “queue_receive”
  - Also internal RTOS operations like “internal_suspend”
- Upload Trace Buffer to host as a file
  - Binary
  - Hex
  - Table
- TraceX reads file and converts to graphical representation
  - Shows all threads
  - Shows all logged events
  - Shows time-ticks
  - Shows context switches



The Application

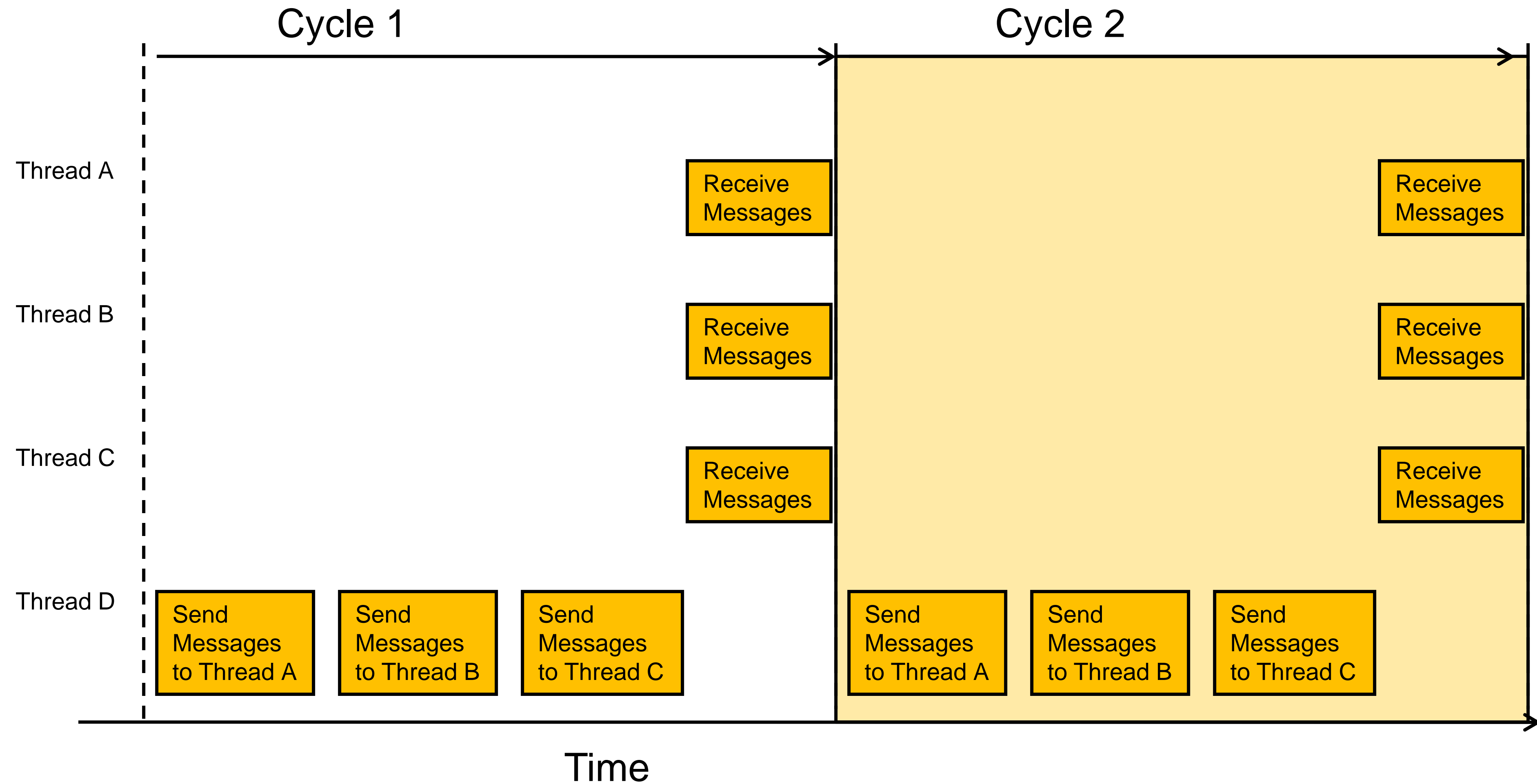
- Fully Preemptive or Preemption-Threshold: Does it matter?
  - What are the consequences of each with respect to context switching?

- Construct a system to run and observe
  - Producer-consumer application
  - One producer, three consumers
  - Continuous operation
  - Log events
  - Run to breakpoint
  - View events
  - Count context switches
  - Measure throughput
  - Draw conclusions



expresslogic

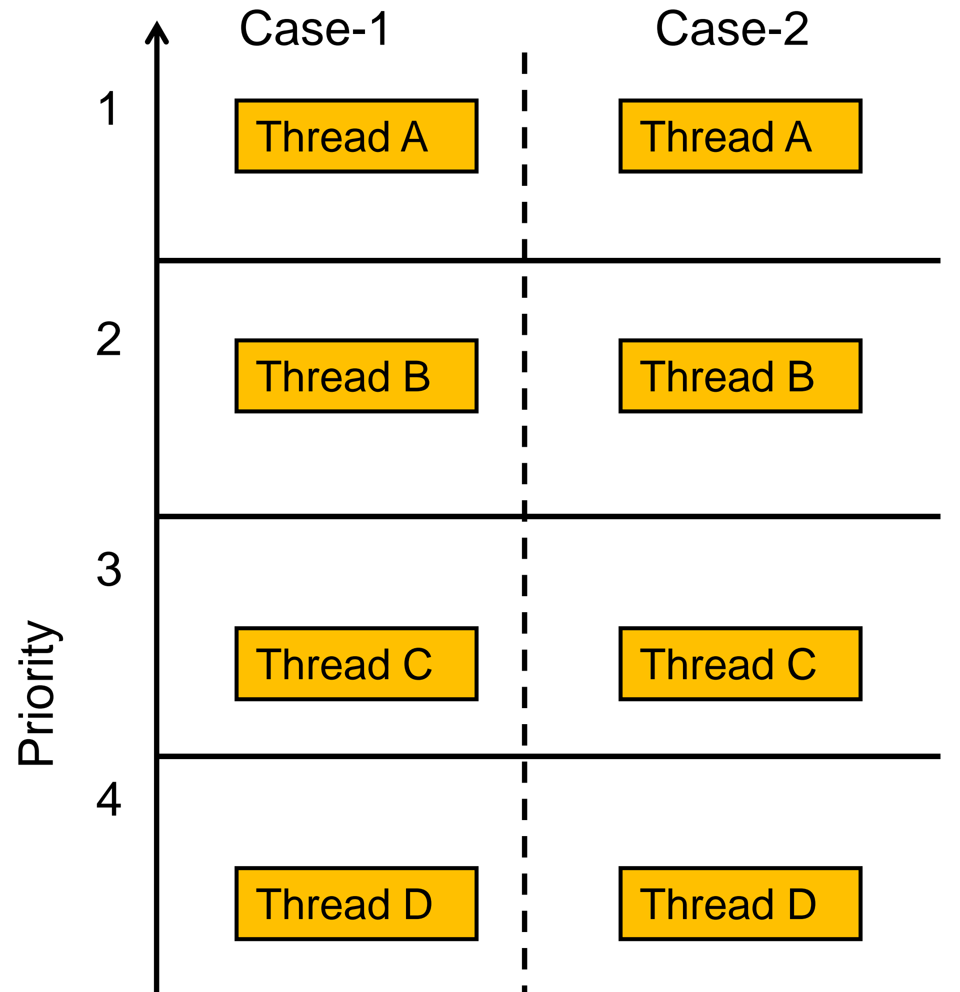
A Test To See The Impact



expresslogic

Priority Assignments

- Case-1
  - Unique Priorities
  - Thread A = 1
  - Thread B = 2
  - Thread C = 3
  - Thread D = 4
  - Preemptive Scheduling
- Case-2
  - Unique Priorities
  - Thread A = 1
  - Thread B = 2
  - Thread C = 3
  - Thread D = 4
  - Preemption-Threshold



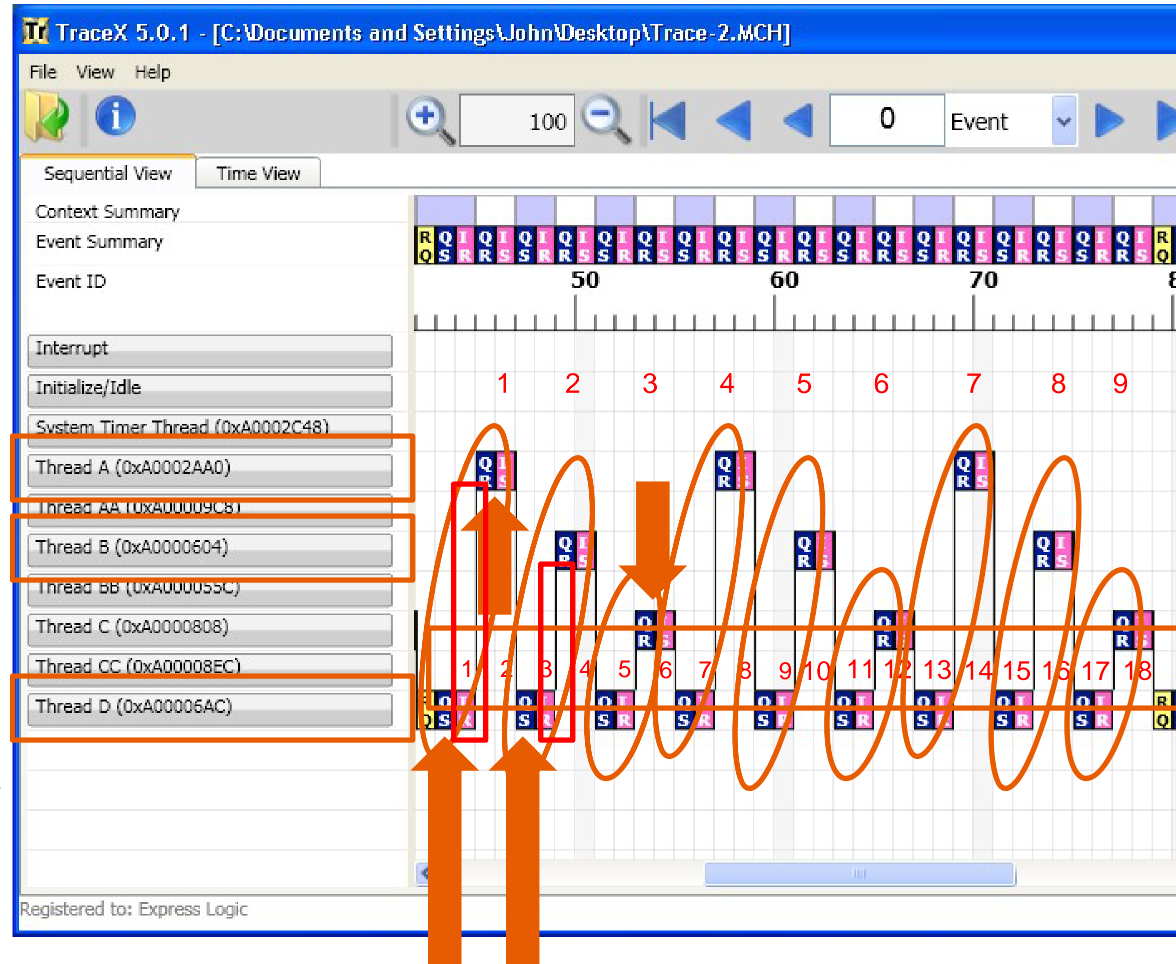
expresslogic

Building And Running the Test Cases

- Case-1
  - No PTS
  - Build
  - Download
  - Set Breakpoint
  - Run to Breakpoint
    - Use TraceX to view activity
    - Use TraceX to measure timer ticks
    - Other TraceX information
- Case-2
  - With PTS
  - Modify code
  - Set Thread-D Preemption-Threshold to 1
    - This means A, B, and C cannot preempt Thread-D
  - Re-build, etc.

Examining The Events

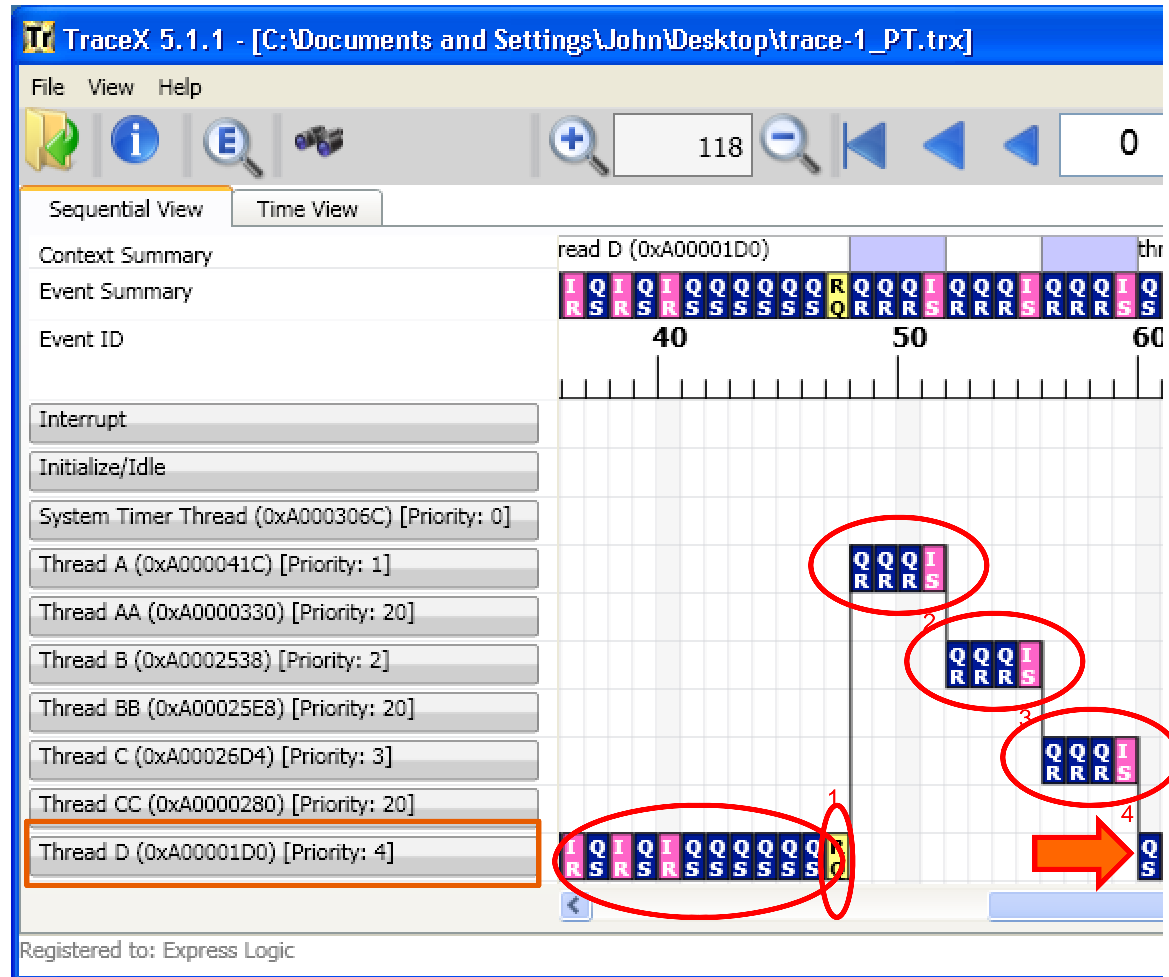
- Case-1: Unique Priorities, No PTS
  - Thread D sends a message to Thread A
  - Thread A preempts
  - Thread A reads its message then suspends (queue empty)
  - Thread D sends message to Thread B
  - Thread B preempts
  - Similarly for Thread C
  - 9 Messages
  - 18 Context Switches



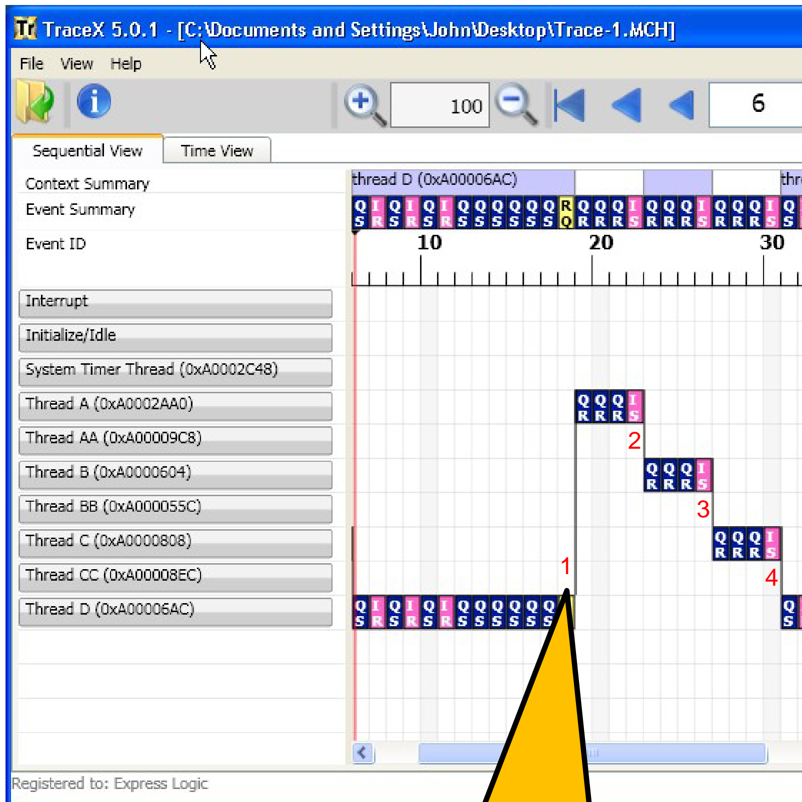
expresslogic

Examining The Events

- Case-2: Preemption-Threshold
  - Thread D sends 3 messages to each queue
  - Thread D then suspends (“relinquish”)
  - Thread A reads its 3 messages then suspends (queue empty)
  - Similarly, for Threads B and C
  - Thread D then writes another set of messages
  - 9 Messages
  - 4 Context Switches

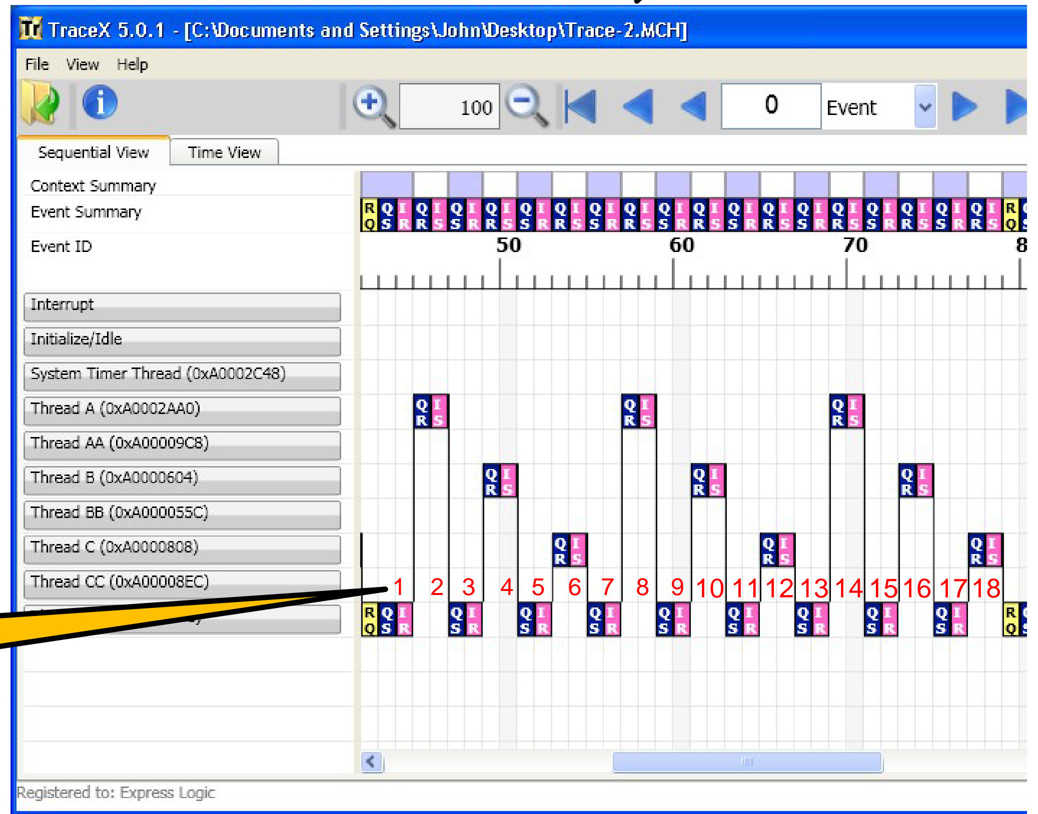


Compare Context Switches



Case-2
With PTS

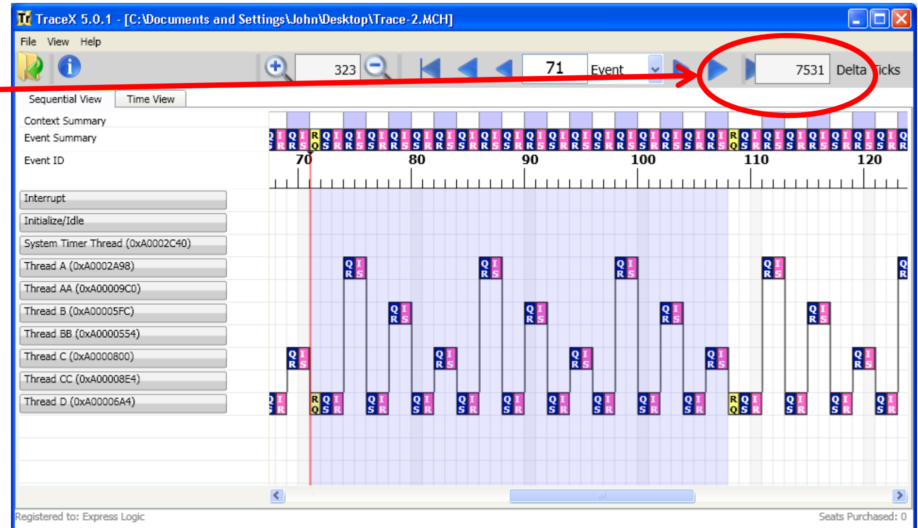
Case-1
No PTS



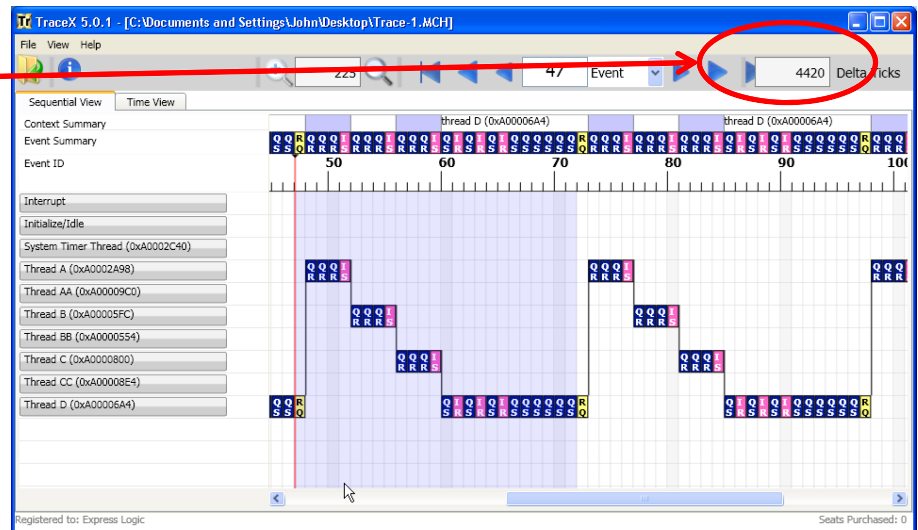
Context Switches

Compare Timing

- Case-1 shows 7,531 ticks in a cycle



- Case-2 shows 4,420 ticks in a cycle



Assessing The Results

- Context Switches

Case	Messages	Context Switches
Case-1: Unique Priorities	9	18
Case-2: Preemption-Threshold	9	4

- Throughput

Measurement	Case-1 (Unique Priorities)	Case-2 (Preemption -Threshold)	Ratio (Case 1 vs Case-2)
Context Switches	18	4	450%
Elapsed Time	7,531 ticks	4,420 ticks	170%
Messages Sent	9	9	No Change
Messages Received	9	9	No Change

Summary And Conclusions

- Using an RTOS is easy and efficient
  - And, enables use of other tools
- Preemption-Threshold can reduce the number of context switches an application must perform
  - Preemption is necessary for maximum responsiveness, but may have a negative impact on throughput
  - Preemption-threshold might offer a solution to excessive context switches, while retaining maximum responsiveness

Q/A

- Q/A

- For further information
  - Contact Express Logic, Inc.
  - 1-888-THREADX (1-858-613-6640)
  - info@expresslogic.com
  - www.rtos.com