Chapter 2 Process, Thread and Scheduling

—— *Scheduler Class and Priority*

XIANG Yong
xyong@tsinghua.edu.cn

opensolaris

Outline

- **Scheduling Class and Priority**
- Dispatch Queues & Dispatch Tables
- Thread Priorities & Scheduling
- Turnstiles & Priority Inheritance

Scheduling Class and Priority

- **Solaris supports multiple scheduling classes**
  - Allows for the co-existence of different priority schemes andscheduling algorithms (policies) within the kernel
  - Each scheduling class provides a class-specific function to manage thread priorities, administration, creation, termination, etc.
- **The dispatcher is the kernel subsystem**
  - **Manages the dispatch queues (run queues), handles thread selection, context switching, preemption, etc**

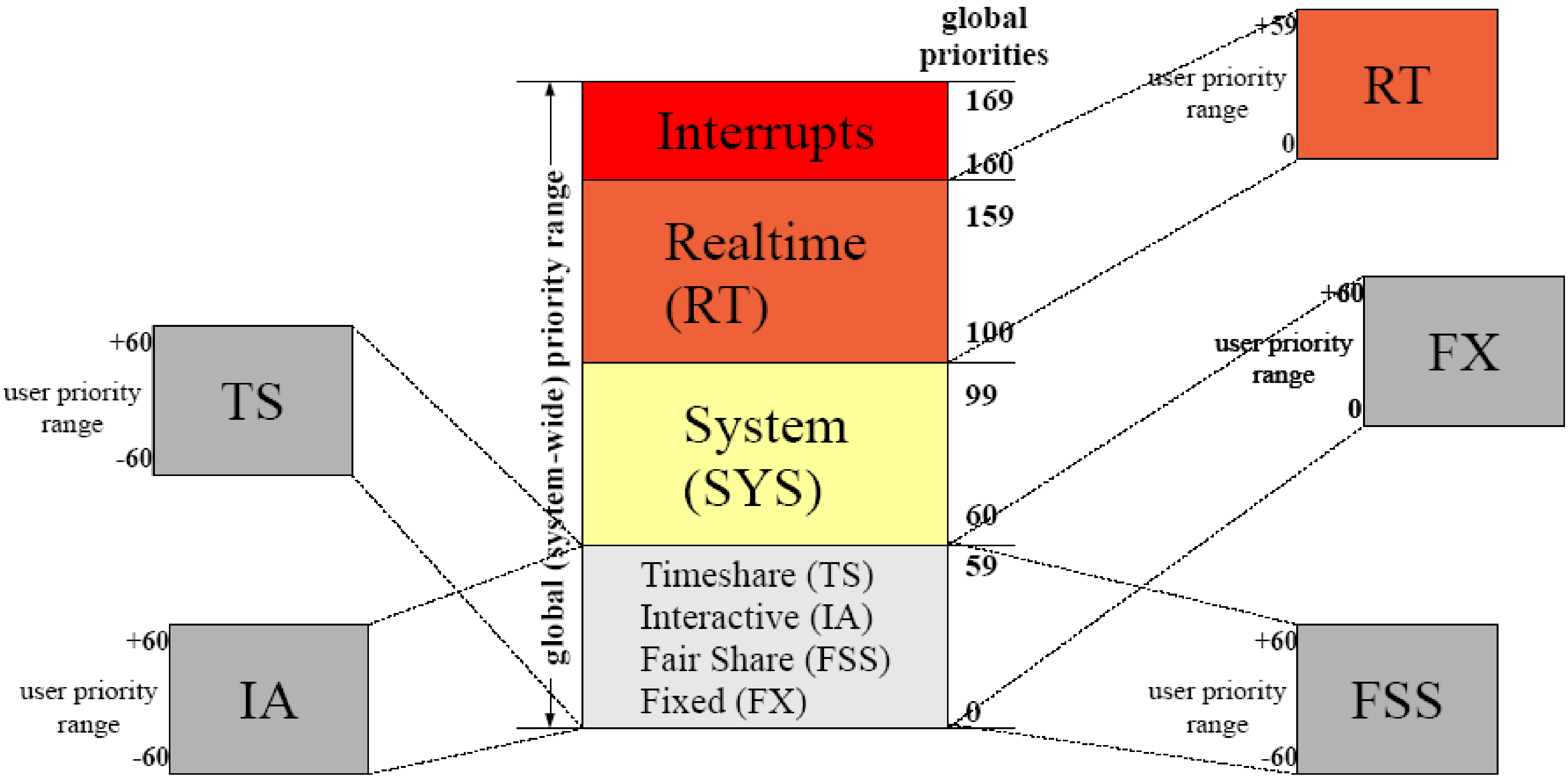
Scheduling Classes

- Traditional Timeshare (TS) class
  - attempt to give every thread a fair shot at execution time
- Interactive (IA) class
  - Desktop only
  - Boost priority of active (current focus) window
  - Same dispatch table as TS
- System (SYS)
  - Only available to the kernel, for OS kernel threads
- Realtime (RT)
  - Highest priority scheduling class
  - Will preempt kernel (SYS) class threads
  - Intended for realtime applications

Scheduling Classes (Con’d)

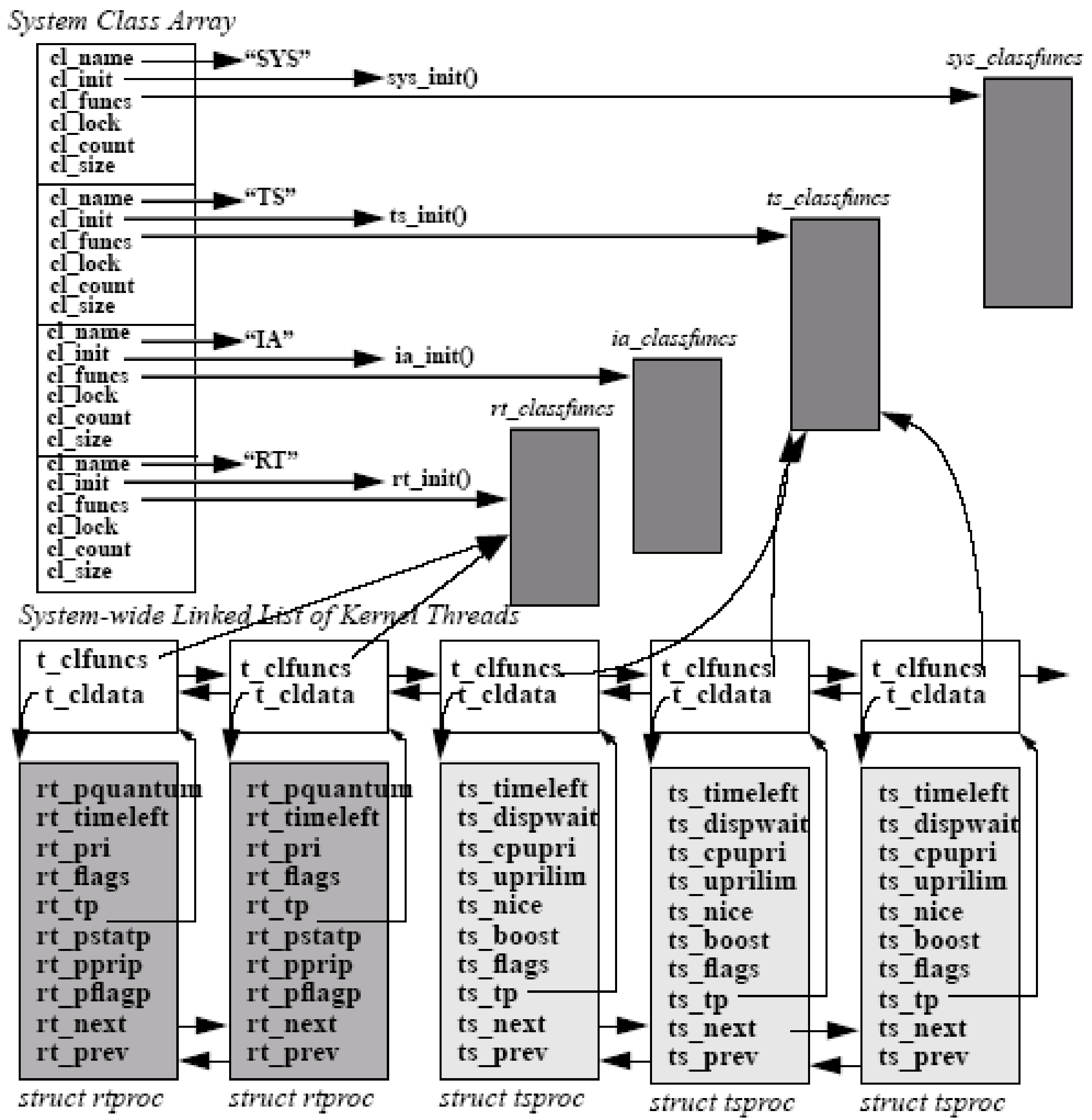
- Fair Share Scheduler (FSS) Class
  - Same priority range as TS/IA class
  - CPU resources are divided into shares
  - Shares are allocated (projects/tasks) by administrator
  - Scheduling decisions made based on shares allocated and used, not dynamic priority changes
- Fixed Priority (FX) Class
  - The kernel will not change the thread's priority
  - A “batch” scheduling class

Priorities


global
priorities
169
160
159
100
99
60
59
0
Interrupts
Realtime
(RT)
System
(SYS)
Timeshare (TS)
Interactive (IA)
Fair Share (FSS)
Fixed (FX)
global (system-wide) priority range
TS
+60
user priority
range
-60
IA
+60
user priority
range
-60
RT
+59
user priority
range
0
FX
+60
user priority
range
0
FSS
+60
user priority
range
-60

Scheduling Class Structures

- The kernel maintains an array of sclass structures for each loaded scheduling class
- Thread pointer to the class functions array, and perthread class-specific data structure
- Scheduling class operations vectors and CL_XXX macros allow a single, central dispatcher to invoke scheduling-class specific functions


System Class Array
cl_name
cl_init
cl_funcs
cl_lock
cl_count
cl_size
"SYS"
sys_init()
sys_classfuncs
"TS"
ts_init()
ts_classfuncs
"IA"
ia_init()
ia_classfuncs
"RT"
rt_init()
rt_classfuncs
System-wide Linked List of Kernel Threads
t_clfuncs
t_cldata
rt_pquantum
rt_timeleft
rt_pri
rt_flags
rt_tp
rt_pstatp
rt_pprip
rt_pflagp
rt_next
rt_prev
struct rtproc
ts_timeleft
ts_dispwait
ts_cpupri
ts_uprilim
ts_nice
ts_boost
ts_flags
ts_tp
ts_next
ts_prev
struct tsproc

Scheduling Class Specific Functions

- Implemented via macros
  - `#define CL_ENTERCLASS(t, cid, clparmsp, credp, bufp) \`
  - `(sclass[cid].cl_funcs->thread.cl_enterclass) (t, cid, \`
  - `(void *)clparmsp, credp, bufp)`
- Class management and priority manipulation functions
  - xx_admin, xx_getclinfo, xx_parmsin, xx_parmsout, xx_getclpri, xx_enterclass, xx_exitclass, xx_preempt, xx_sleep, xx_tick, xx_trapret, xx_fork, xx_parms[get|set], xx_donice, xx_yield, xx_wakeup

Outline

- Scheduling Class and Priority
- **Dispatch Queues & Dispatch Tables**
- Thread Priorities & Scheduling
- Turnstiles & Priority Inheritance

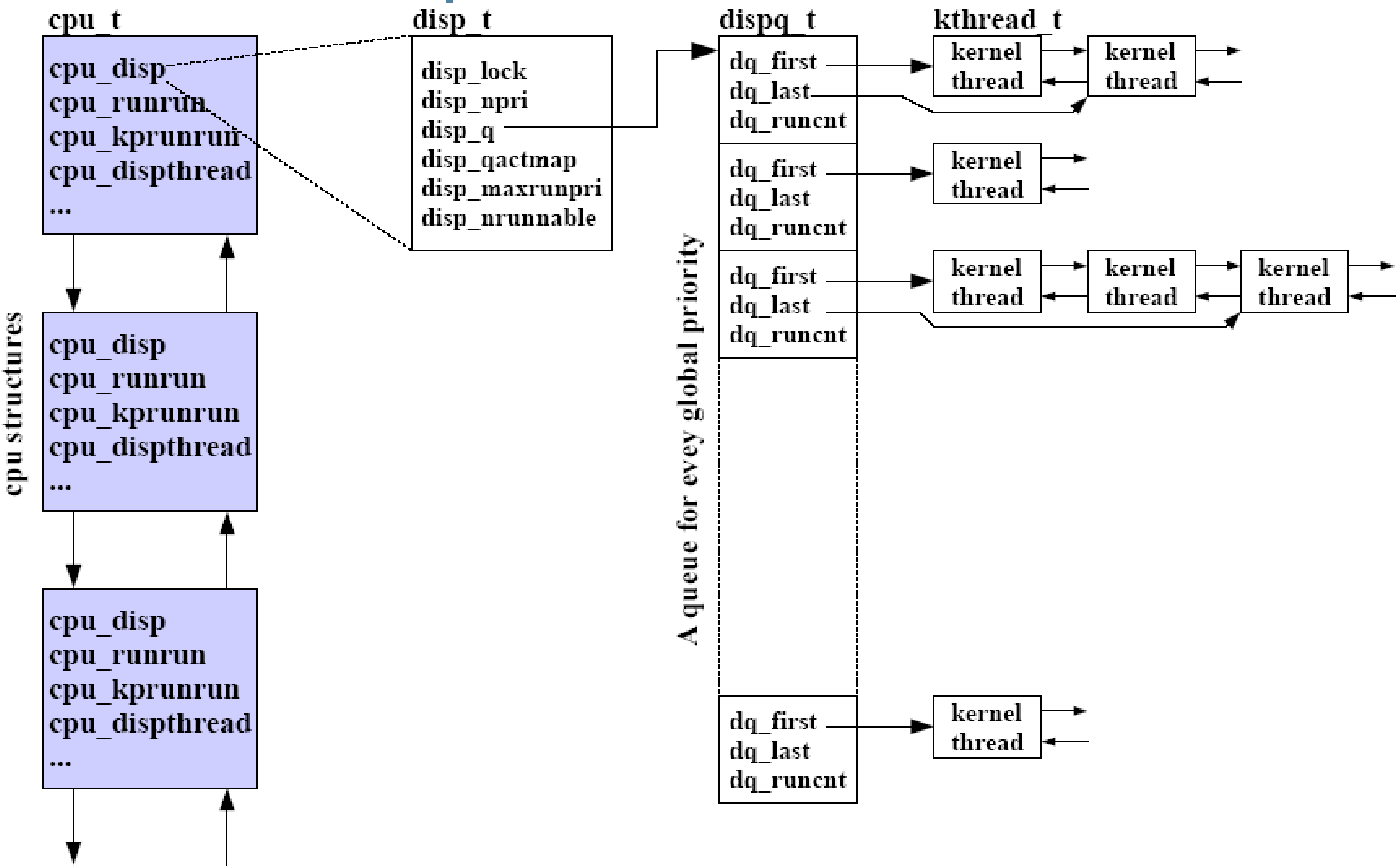
Dispatcher

- The kernel subsystem that manages the dispatch queues (run queues), handles preemption, finding the next runnable thread, the idle loop, initiating context switching, etc
- Solaris implements per-processor dispatch queues - actually a queue of queues
- Several dispatcher-related variables maintained in the CPU structure as well
  - cpu_runrun - preemption flag - do it soon
  - cpu_kprunrun - kernel preemption flag - do it now!
  - cpu_disp - dispatcher data and root of queues
  - cpu_chosen_level - priority of next selected thread
  - cpu_dispthread - kthread pointer
- A system-wide (or per-processor set) queue exists for realtime threads

Dispatch Queues

- Per-CPU run queues
  - Actually, a queue of queues
- Ordered by thread priority
- Queue occupation represented via a bitmap
- For Realtime threads, a system-wide kernel preempt queue is maintained
  - Realtime threads are placed on this queue, not the per-CPU queues
  - If processor sets are configured, a kernel preempt queue exists for each processor set

Per-CPU Dispatch Queues



Dispatch Tables

- Per-scheduling class parameter tables
- Time quantums and priorities
- tuneable via dispadmin(1M)

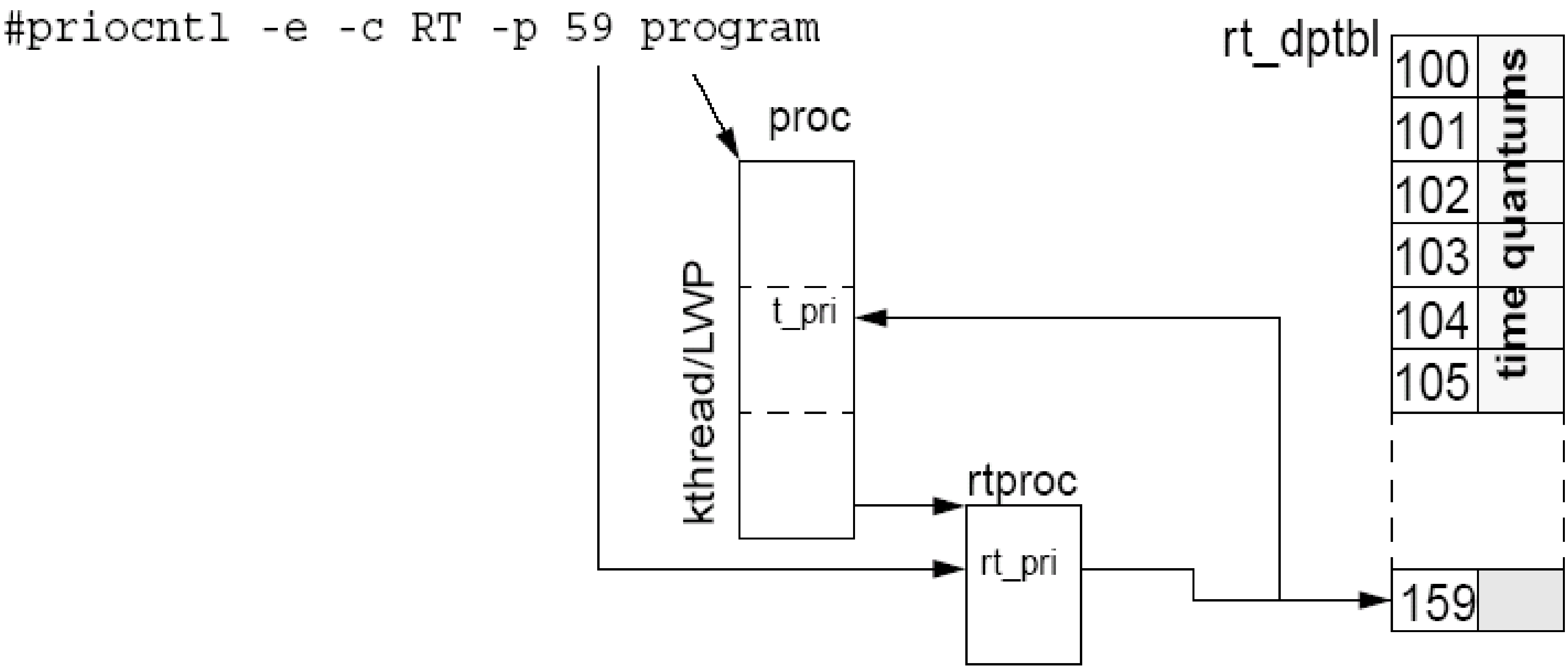
TS Dispatch Table

- TS and IA class share the same dispatch table
  - RES. Defines the granularity of ts_quantum
  - ts_quantum. CPU time for next ONPROC state
  - ts_tqexp. New priority if time quantum expires
  - ts_slpret. New priority when state change from TS_SLEEP to TS_RUN
  - ts_maxwait. “waited too long” ticks
  - ts_lwait. New priority if “waited too long”

RT, FX & FSS Dispatch Tables

- RT
  - Time quantum only
  - For each possible priority
- FX
  - Time quantum only
  - For each possible priority
- FSS
  - Time quantum only
  - Just one, not defined for each priority level
  - Because FSS is share based, not priority based
- SYS
  - No dispatch table
  - Not needed, no rules apply
- INT
  - Not really a scheduling class

Setting A RT Thread's Priority



Dispatch Queue Placement

- Queue placement is based a few simple parameters
  - The thread priority
  - Processor binding/Processor set
  - Processor thread last ran on: Warm affinity
  - Depth and priority of existing runnable threads
  - Memory Placement Optimization (MPO) enabled will keep thread in defined locality group (lgroup)

Dispatch Queue Manipulation

- setfrontdq(),
- setbackdq()
- A thread will be placed on either the front of back of the appropriate dispatch queue depending on

Outline

- Scheduling Class and Priority
- Dispatch Queues & Dispatch Tables
- **Thread Priorities & Scheduling**
- Turnstiles & Priority Inheritance

Thread Priorities & Scheduling

- Every thread has 2 priorities; a global priority, derived based on its scheduling class, and (potentially) and inherited priority
- Priority inherited from parent, alterable via priocntl(1) command or system call
- Typically, threads run as either TS or IA threads
  - IA threads created when thread is associated with a windowing system
- RT threads are explicitly created
- SYS class used by kernel threads, and for TS/IA threads when a higher priority is warranted
  - A temporary boost when an important resource is being held
- Interrupts run at interrupt priority

Thread Selection

- The kernel dispatcher implements a select-and-ratify thread selection algorithm
  - disp_getbest(). Go find the highest priority runnable thread, and select it for execution
  - disp_ratify(). Commit to the selection. Clear the CPU preempt flags, and make sure another thread of higher priority did not become runnable
    - If one did, place selected thread back on a queue, and try again
- Warm affinity is implemented
  - Put the thread back on the same CPU it executed on last
    - Try to get a warm cache
  - rechoose_interval kernel parameter
    - Default is 3 clock ticks

Thread Preemption

- Two classes of preemption
  - User preemption
    - A higher priority thread became runnable, but it's not a realtime thread
    - Flagged via cpu_runrun in CPU structure
    - Next clock tick, you're outta here
  - Kernel preemption
    - A realtime thread became runnable. Even OS kernel threads will get preempted
    - Poke the CPU (cross-call) and preempt the running thread now

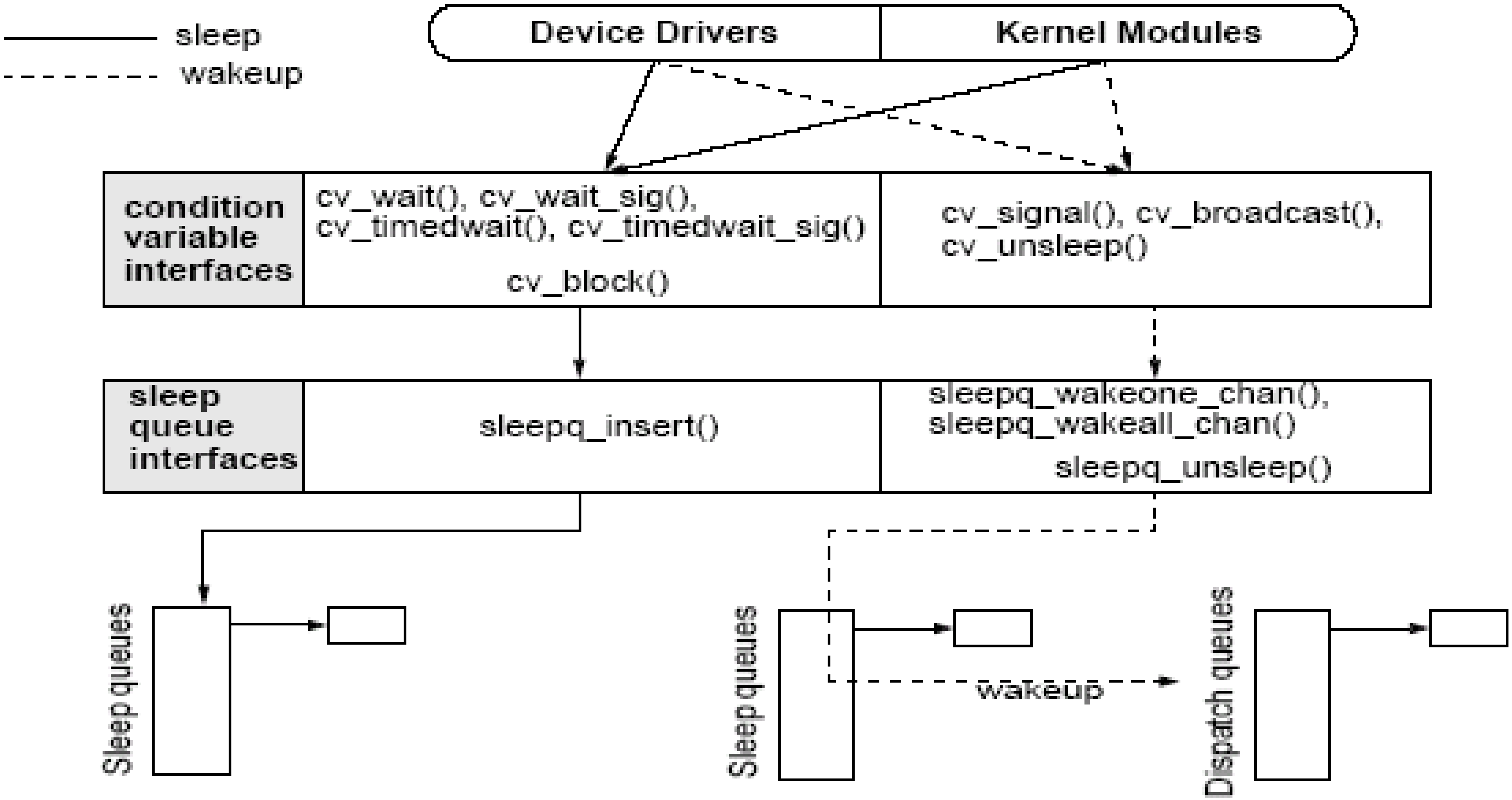
Thread Execution

- Run until
  - A preemption occurs
    - Transition from S_ONPROC to S_RUN
    - placed back on a run queue
  - A blocking system call is issued
    - e.g. read(2)
    - Transition from S_ONPROC to S_SLEEP
    - Placed on a sleep queue
  - Done and exit
    - Clean up
  - Interrupt to the CPU you're running on
    - pinned for interrupt thread to run
    - unpinned to continue

Sleep & Wakeup

- Condition variables used to synchronize thread sleep/wakeup
  - A block condition (waiting for a resource or an event) enters the kernel cv_xxx() functions
  - The condition variable is set, and the thread is placed on a sleep queue
  - Wakeup may be directed to a specific thread, or all threads waiting on the same event or resource
    - One or more threads moved from sleep queue, to run queue

Sleep/Wakeup Kernel Subsystem



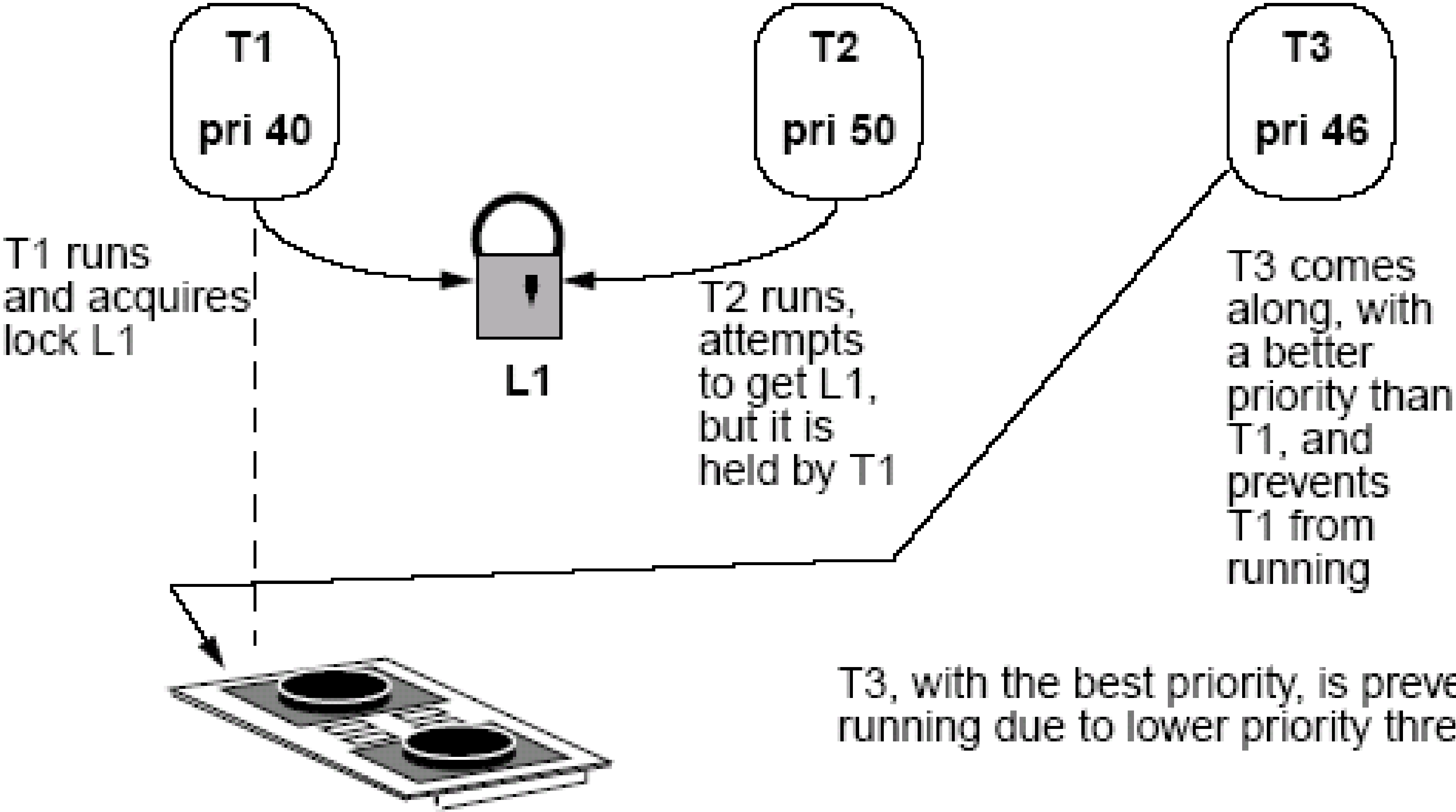
Outline

- Scheduling Class and Priority
- Dispatch Queues & Dispatch Tables
- Thread Priorities & Scheduling
- **Turnstiles & Priority Inheritance**

Turnstiles & Priority Inheritance

- Turnstile - A special set of sleep queues for kernel threads blocking on mutex or R/W locks
- Priority inversion - a scenerio where a thread holding a lock is preventing a higher priority thread from running, because the higher priority thread needs the lock.
- Priority inheritance - a mechanism whereby a kernel thread may inherit the priority of the higher priority kernel thread
- Turnstiles provide sleep/wakeup, with priority inheritance, for synchronization primitives

Priority Inversion



T3, with the best priority, is prevented from running due to lower priority threads

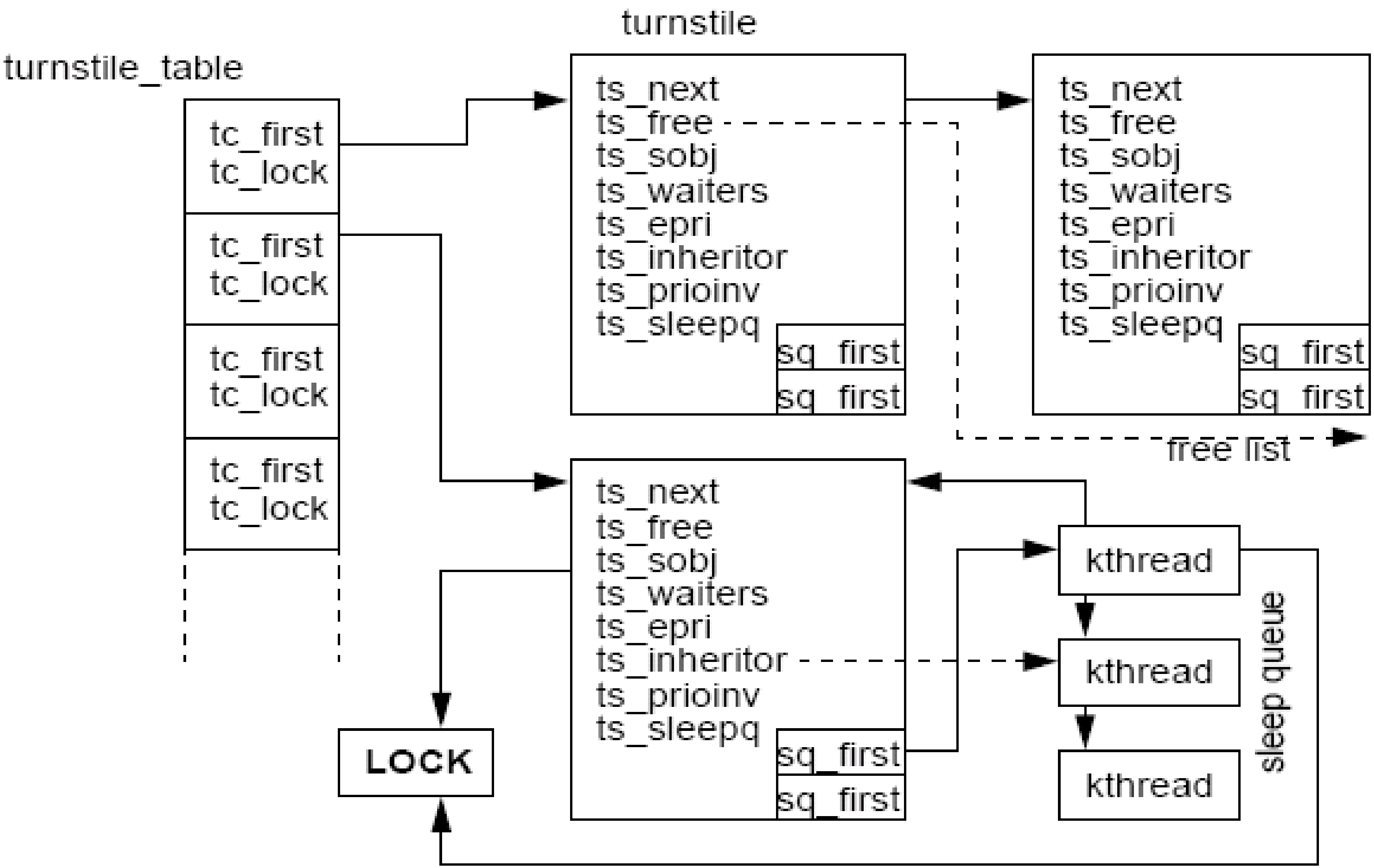
Turnstiles

- All active turnstiles reside in turnstile_table[], index via a hash function on the address of the synchronization object
- Each hash chain protected by a dispatcher lock, acquired by turnstile_lookup()
- Each kernel thread is created with a turnstile, in case it needs to block on a lock
- turnstile_block() - put the thread to sleep on the appropriate hash chain, and walk the chain, applying PI where needed

Turnstiles (con'd)

- turnstile_wakeup() - waive an inherited priority, and wakeup the specific kernel threads
- For mutex locks, wakeup is called to wake all kernel threads blocking on the mutex
- For R/W locks;
  - If no waiters, just release the lock
  - If a writer is releasing the lock, and there are waiting readers and writers, waiting readers get the lock if they are of the same or higher priority than the waiting writer
  - A reader releasing the lock gives priority to waiting writers

Turnstiles (con’d)



opensolaris

Reference

- Richard McDougall, James Mauro, "SOLARIS Kernel Performance, Observability & Debugging", USENIX'05, 2005, t2-solaris-slides.pdf
- Solaris internals and performance management, Richard McDougall, 2002, class0802.pdf

End

- 2006-02-19