Unit 3: Processes and Threads

3.5. Windows 2000 Thread Scheduling

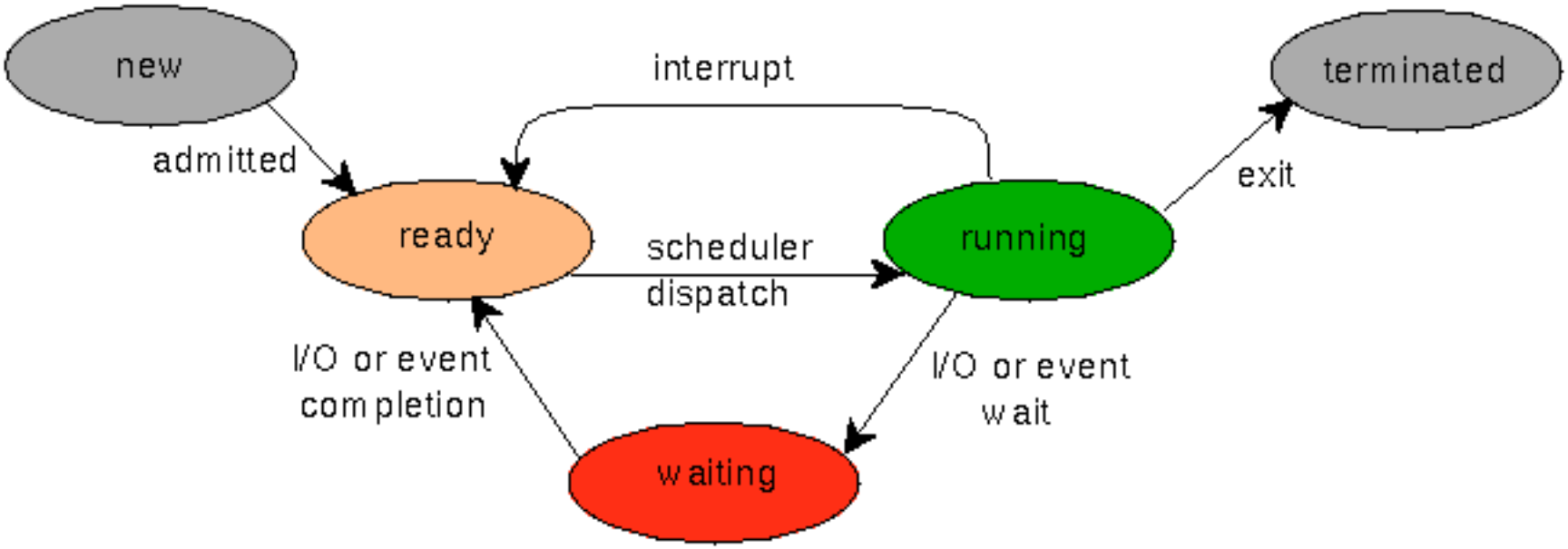
Windows 2000 Thread Scheduling

- Priority-driven, preemptive scheduling system
- Highest-priority runnable thread always runs
- Restricted by *processor affinity*
- Thread runs for time amount of *quantum*
- No single scheduler – event-based scheduling code spread across the kernel

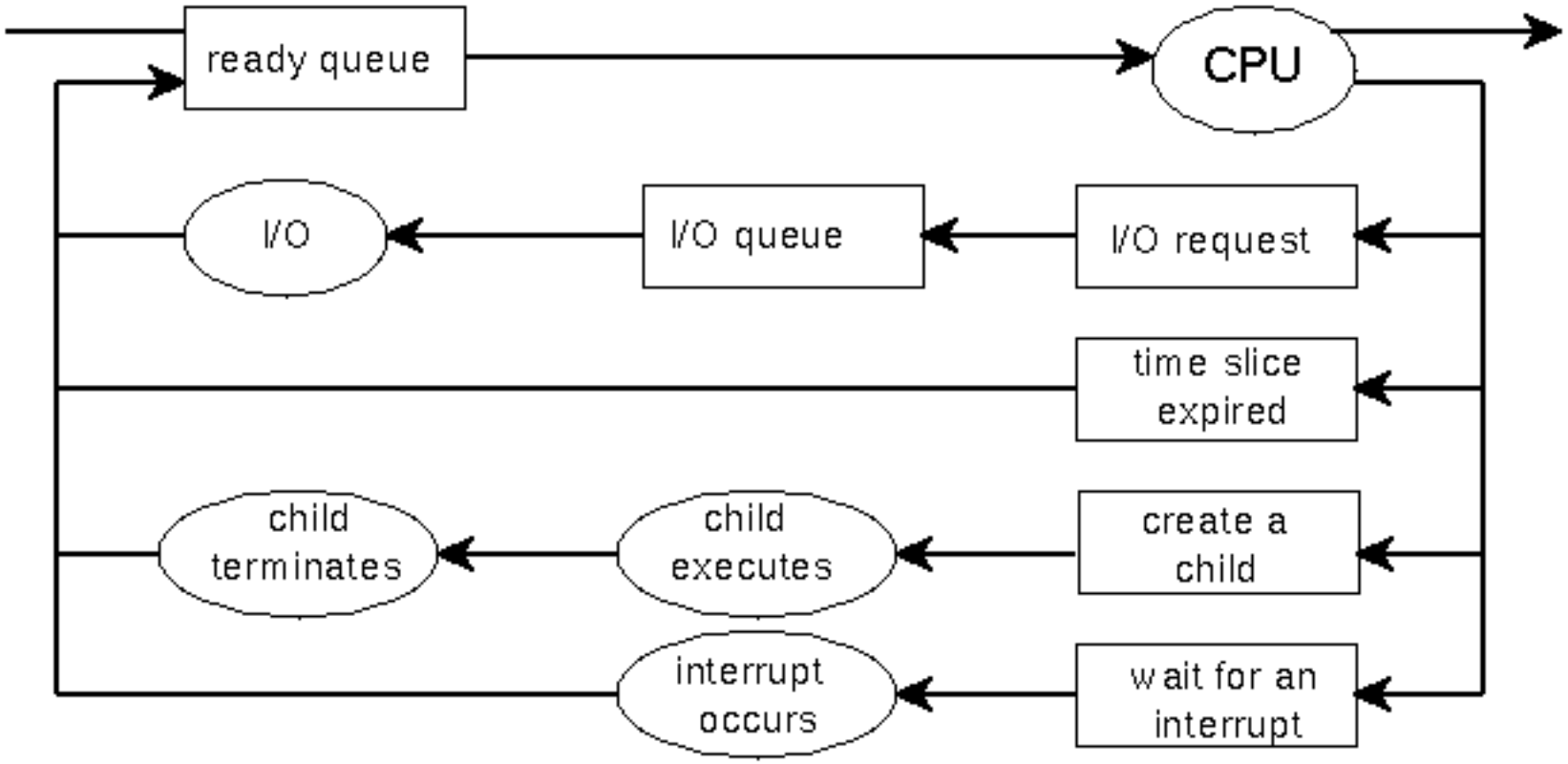
  Dispatcher routines triggered by the following events:
  - Thread becomes ready for execution
  - Thread leaves running state (quantum expires, wait state)
  - Thread‘s priority changes (system call/NT activity)
  - Processor affinity of a running thread changes

Single Processor Scheduling

- Scheduling on a per-thread basis

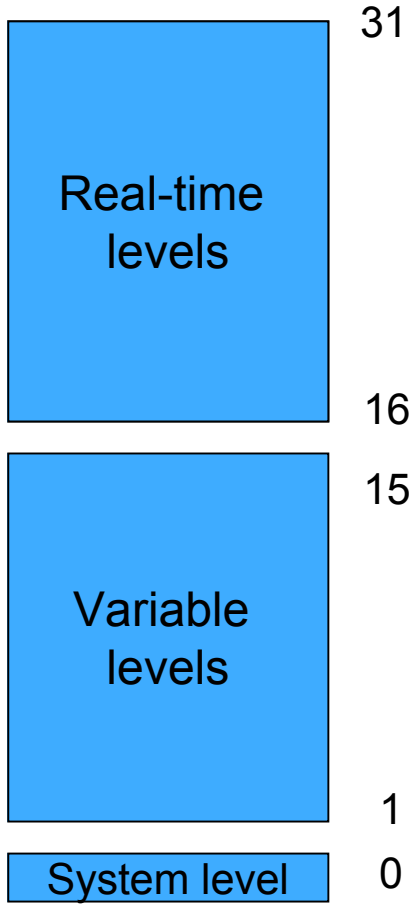


Single-Processor Scheduling queuing diagram



Windows 2000 priority levels

- 16-32: static priorities (real-time)
- 1-15: variable priorities
- 0: MemManager: zero-page thread
- 0: Idle: „lower than 0“, no thread no
- Idle Threads: one per CPU (in proc with PID 0)



Win32 vs. NT Kernel Priorities (thread base priorities)

Win32 priority classes

Win32 thread priorities	Real-time	High	Normal	Idle
Time critical	31	15	15	15
Highest	26	15	10	6
Above normal	25	14	9	5
Normal	24	13	8	4
Below normal	23	12	7	3
Lowest	22	11	6	2
Idle	16	1	1	1

Win32 priorities

- Thread priority is based on combination of process priority class and relative thread priority
- Row „normal“ is base priority for the priority classes
- Base priority can be changed (*SetProcessPriority()*)
  - Default base priority is „normal“ (24, 13, 8, 4)
  - NT system proc. have higher base prio. than default for normal class (session manager, service controller, local authentication server)
- Thread priority is adjusted by NT (variable levels)
- Real-time priorities are never adjusted

Scheduling rules (1)

- Preemptive, priority driven
  - FIFO ready queues per priority level
  - A ready thread either runs or is inserted at end of queue
  - ...a preempted thread is inserted at head of queue
- Time-sliced round-robin per priority level
  - Threads have base priority & varying current priority
  - Process has only base priority (starting prio for threads in this process)
- On multiprocessor systems:
  - Tries to keep thread on same CPU
  - Lowest priority thread is preempted
  - Any processor can interrupt another processor to schedule a thread

Scheduling rules (2)

- Voluntary switch:
  - Enters wait state
  - SwitchToThread (NtYield)
  - Explicit priority decrease (SetThreadPriority – NtSetInformationThread)
  - Action: the next_ready thread is run, previous goes to right/state list
- Preemption: (fixed prio vs. variable prio)
  - A higher priority thread becomes ready
  - Action: lowest priority thread is preempted (goes to head of queue)
  - Preemption is immediate (fixed prio) or at quantum end (variable prio) (Win 2000)
- Quantum end:
  - Next_ready thread is run, previous has prio decrease by 1 (no lower than base priority)

Scheduling rules (3)

- Quantum duration:
  - Workstation (Professional): 2 clock ticks
  - Server: 12 clock ticks
  - Clock tick: 10 ms (7.5-15 ms)
- Quantum stretching (favouring foreground applications)
  - Longer quantum: 2, 4, 6 clock ticks
  - HKLM\CCS\Control\PriorityControl\W32PrioritySeparation=0,1,2
- Priority boosting (keep the I/O system busy)
  - After a wait (e.g. I/O) is satisfied a priority boost is given (not over 15)
  - Default boost values:
    - 1 for disk, CD-ROM, parallel, video, semaphore
    - 2 for serial, network, named pipe, mailslot
    - 6 for keyboard or mouse
  - Amount can be specified by driver or executive

CPU Starvation Avoidance

- Balance Manager runs every second and:
  - Scan ready_queues for threads that were ready for 4 sec or more
  - Attempts to avoid „priority inversion deadlock“:
    - Thread A waits at priority 14 for resource owned by
    - Thread B which is ready at priority 4 and has no chance to run (load)
- Actions (since NT 4):
  - Thread B is boosted at priority 15 for a double quantum
  - After quantum ends, priority decreases to initial value (4)
- Preventing overhead:
  - Not more than 16 read_threads per queue per second
  - Not more than 10 priority boots per second

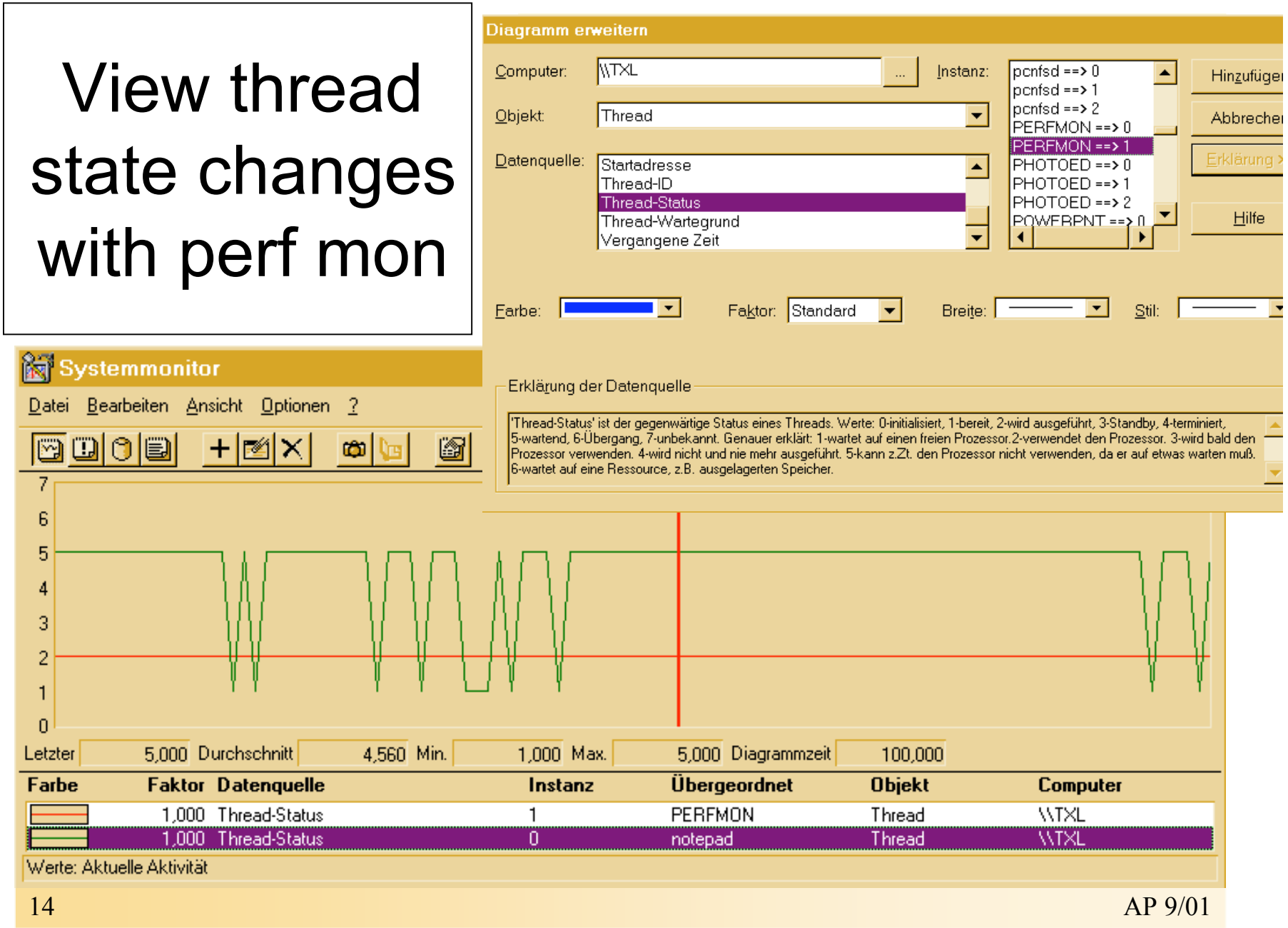
Win32 Scheduling APIs

Suspend/ResumeThread	Suspends or resumes a paused thread from execution
Get/SetPriorityClass	Returns or sets a process‘s priority class (base prio)
Get/SetThreadPriority	Returns or sets a thread‘s priority (relative to its process base priority
Get/SetProcessAffinityMask	Returns or sets a process‘s affinity mask
SetThreadAffinityMask	Sets a thread‘s affinity mask (subset of process‘s affinity mask) for a particular set of processors
Get/SetThreadPriorityBoost	Returns or sets ability for NT to boost priority of a thread
SetThreadIdealProcessor	Establishes preferred processor (not restricting to that p.)
Get/SetProcessPriorityBoost	Returns/sets default priority boost control state
SwitchToThread	Yields execution for one quantum to another ready thread
Sleep	Puts thread in wait state for n msec (0: give up quantum)
SleepEx	Wait until I/O completion, or APC, or time interval ends

Relevant Tools

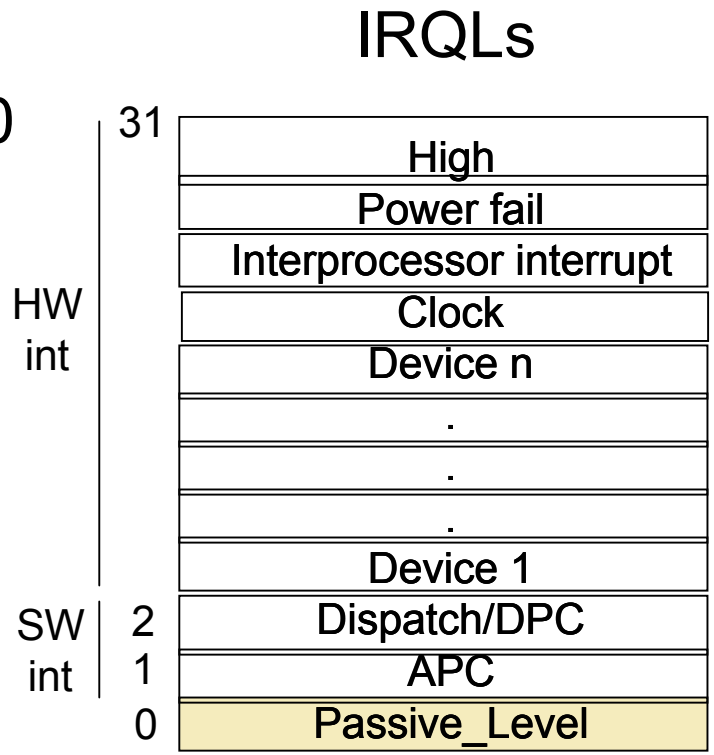
- View (and change) process base priority with:
  - TaskManager, Pview, Pviewer
- View numeric process base priority with:
  - PerfMon, pstat
- View thread priorities with
  - PerfMon, Pview, Pviewer, Pstat
- No general utility to change relative thread priority levels
- Need *increase scheduling priority* privilege
  - Important NT kernel threads run in real-time priority class
  - Be careful with threads spending excessive time in RT prio range

View thread state changes with perf mon

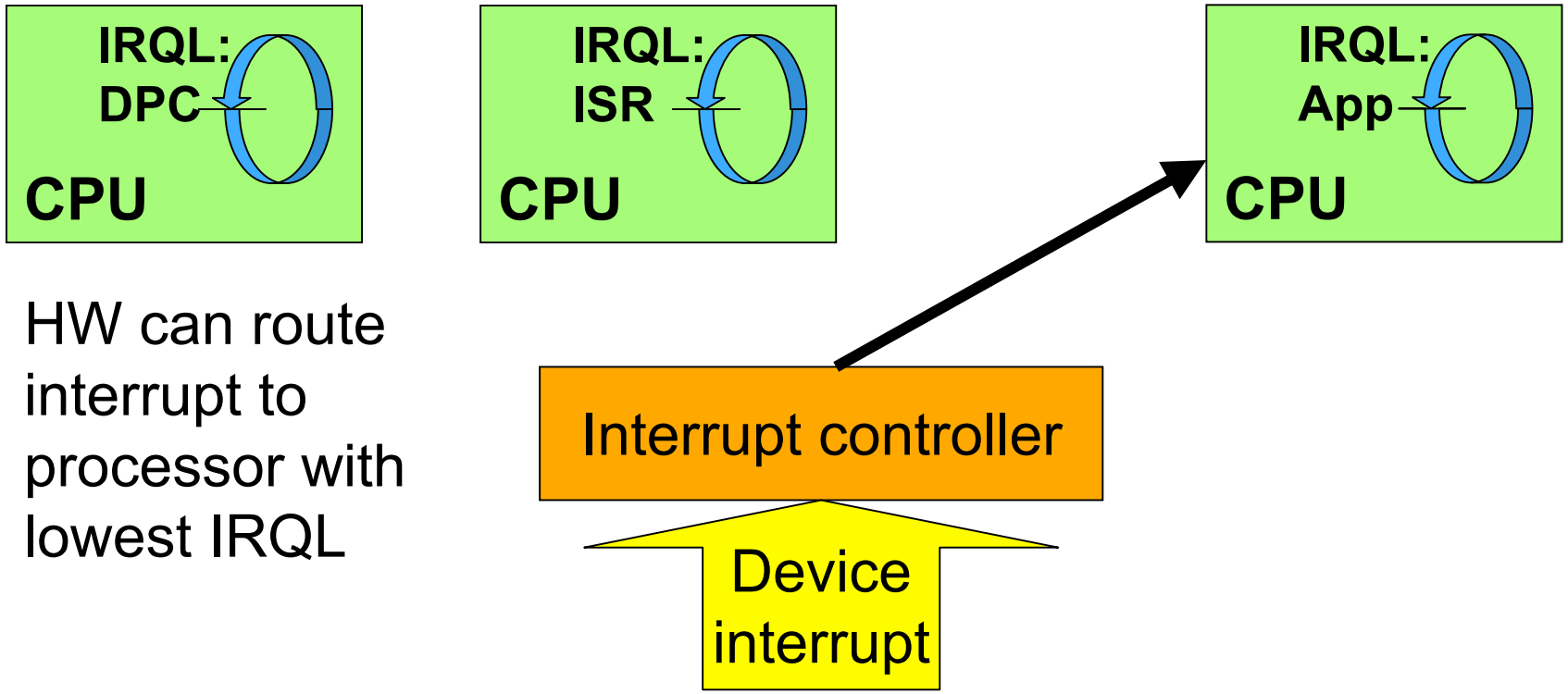


Interrupt levels vs. Priority levels

- All threads run at IRQL 0 or 1
- Threads normally run at IRQL 0
- Only kernel mode APCs (asynch. proc. calls) run at IRQL 1
- No thread ever blocks hardware interrupts
- Thread scheduling at IRQL 2 (dispatching)
- Spinlock synchronizes access to scheduling data on MP system (*KiDispatcherLock*)

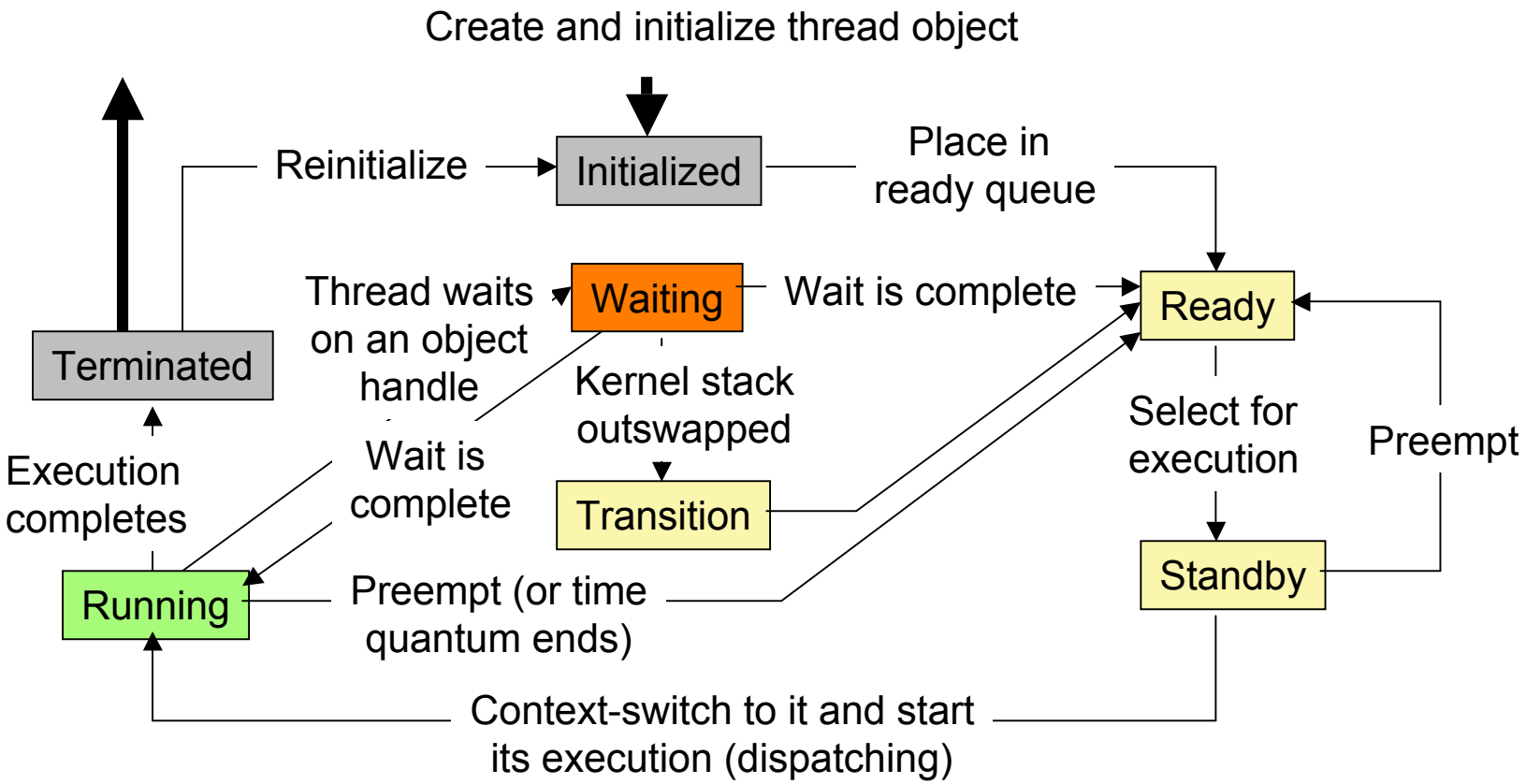


Interrupts on MP system



Different routines of a driver may execute in parallel (ISR, DPC, Dispatch Function)

Thread States



Thread states (contd.)

- **Ready**: waiting for execution
- **Standby**: selected to run next on particular CPU
  (only one per CPU)
- **Running**: executed until kernel preempts thread to run higher prio thread, quantum ends, termination, wait
- **Waiting**: synchronization, I/O, suspended by environment subsystem
  (depending on priority, thread may move to ready state after wait)
- **Transition**: ready for exec. but kernel stack paged out
- **Terminated**: execution finished,
  (thread object may be re-used)

Quantum

Amount of time a thread gets to run before NT checks for other threads

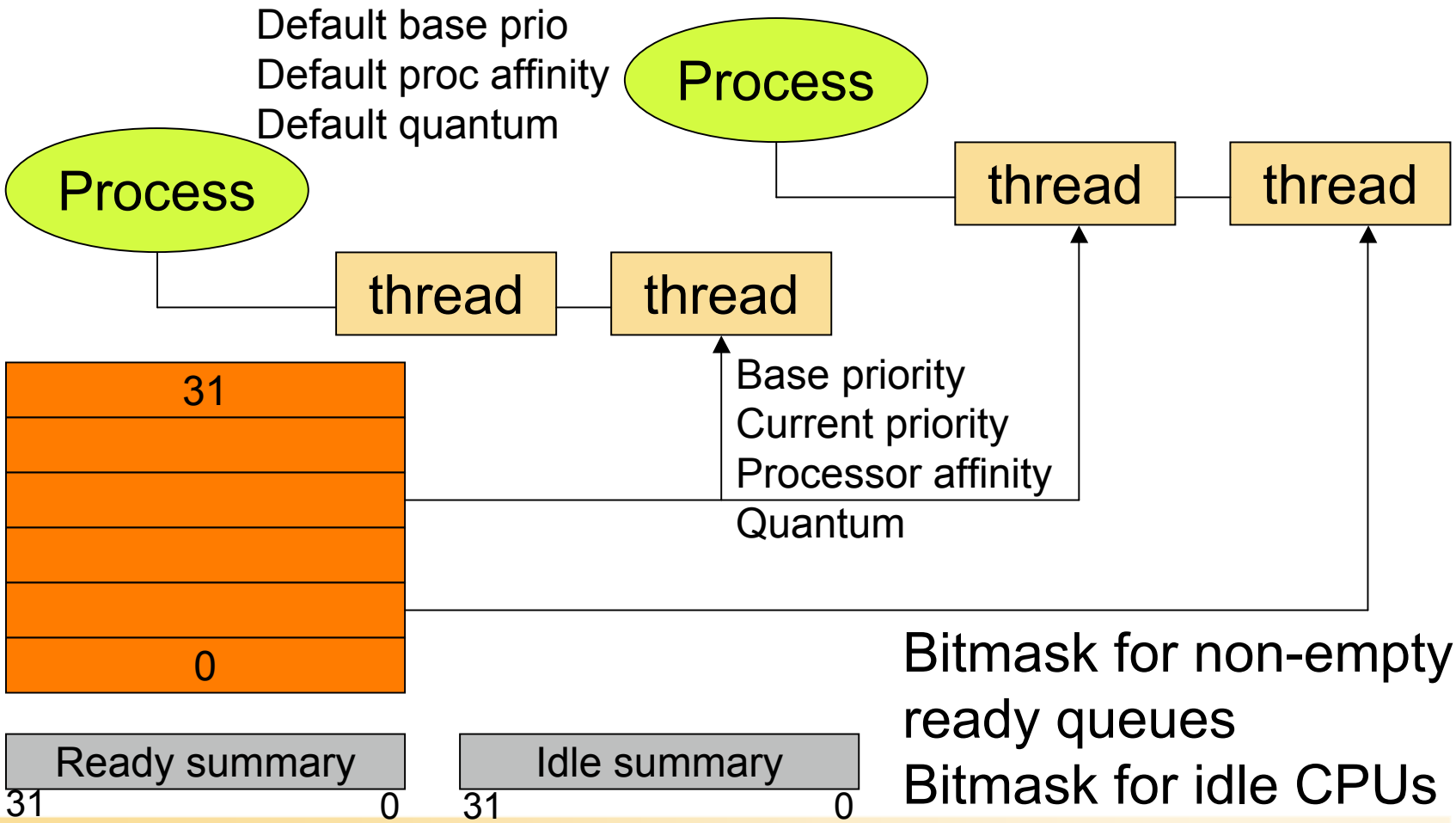
- How long is a quantum? How does NT compute it?
- Each thread runs for QU (*quantum units:* integer val)
  - Threads start with QU = 6 on NT WS, QU = 36 on NT S
  - NT S should be able to answer every request in one quantum
- Each clock-int reduces QU by 3; QU == 0: dispatching
  - Length of clock interval depends on HW (HAL)
- NT WS: Quantum may be increased for foreground app
- QU is doubled when NT boosts thread priority
  - Avoid priority inversion / starvation
- QU is adjusted when thread comes out of wait state
  - QU = initial for real-time threads
  - QU := QU-1 for variable prio threads
  - QU == 0: reset QU to process default value; adjust priority (decrease boosted)

Default Quantum for Different Architectures

Processor	Clock Interval	Default Quantum on NT WS	Default Quantum on NT S
Typical Uniprocessor 486	10 ms	20 ms	120 ms
Typical uniproc. Pentium/PPro	15 ms	30 ms	180 ms
Some Multi-processor 486	10 ms	20 ms	120 ms
Other Multiproc. Intel systems	15 ms	30 ms	180 ms
DEC Alpha AXP	7.8125 ms	15.6 ms	93.6 ms

Typical UNIX system: 10 ms

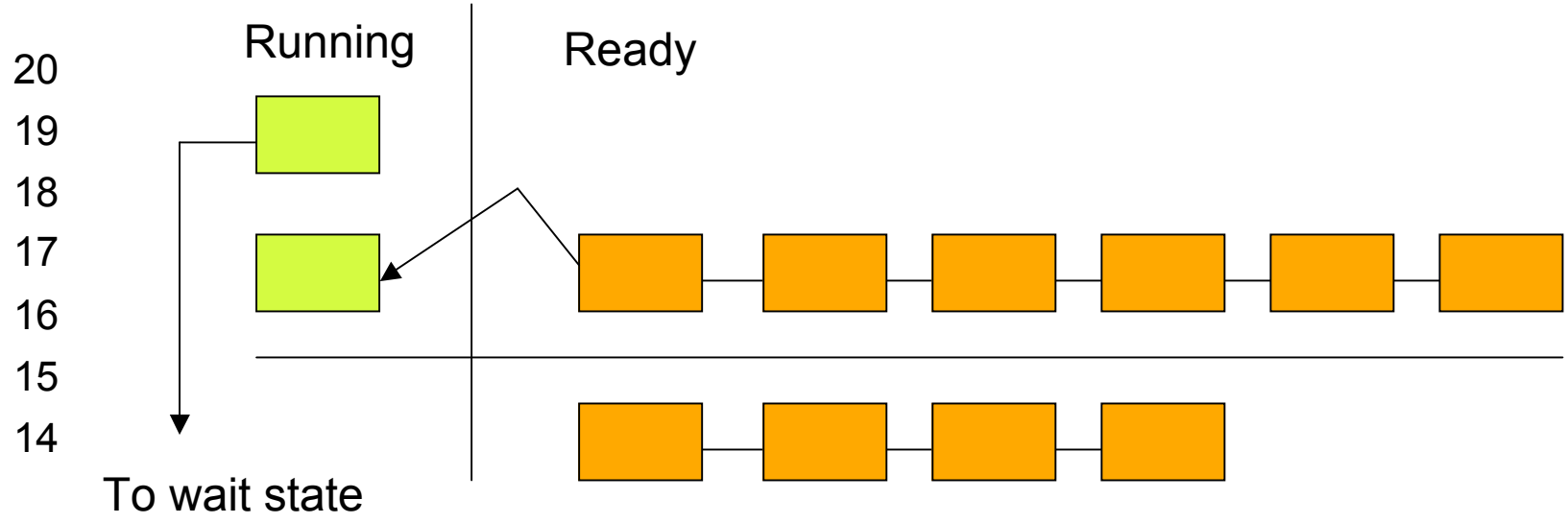
Scheduling Data Structures



Scheduling System Variables

Variable	Type	Description
KiDispatcherLock	Spinlock	Dispatcher spinlock
KeNumberProcessors	Byte	Number of active processors
KeActiveProcessors	Bitmask (32 bits)	Bitmask of active processors
KiIdleSummery	Bitmask (32 bits)	Bitmask of idle processors
KiReadySummary	Bitmask (32 bits)	Bitmask of priority levels that have one ore more ready threads
KiDispatcherReady-ListHead	Array of 32 list entreis	List heads for the 32 ready queues
PspForeground-Quantum	Array of schar	[0] – default QU = 6, [1] – NT WS: QU = 12, [2] – NT WS: QU = 18

Scheduling Scenarios – voluntary switch

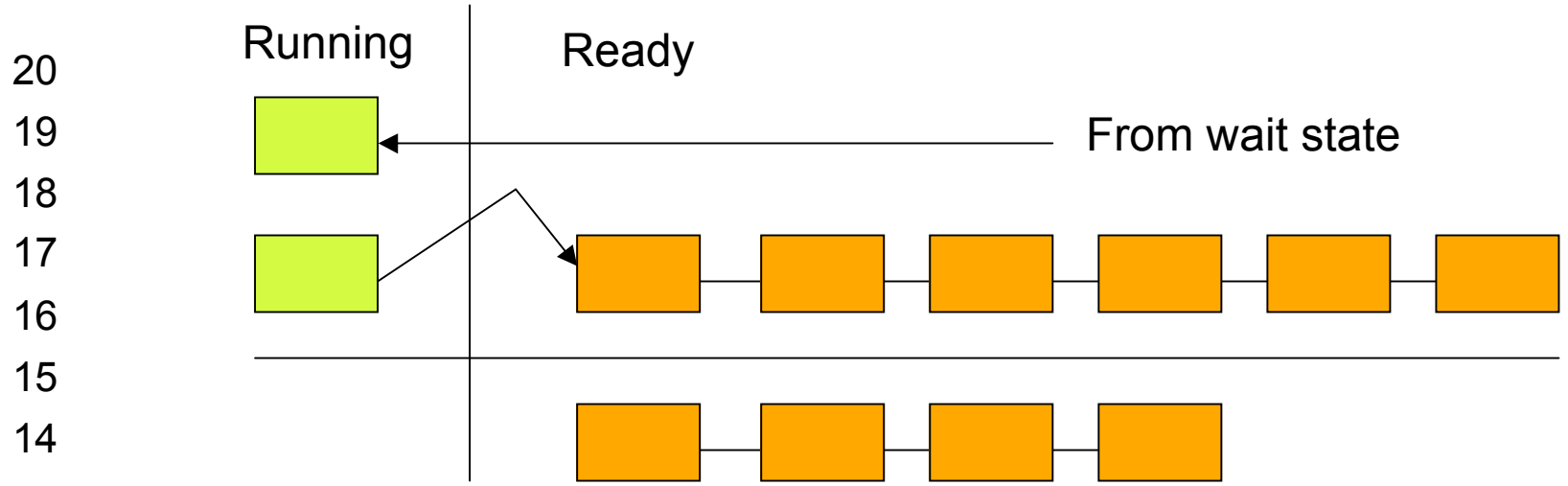


Thread gives up CPU (wait for event, mutex, semaphore, I/O completion port, process, thread, window message

- Priority of relinquishing thread is not reduced
- Quantum value is decremented by 1 (when wait satisfied)

Scheduling Scenarios – preemption

- A higher-priority thread‘s wait completes
- A thread priority is increased or decreased
- Preempted thread is put at head of ready queue

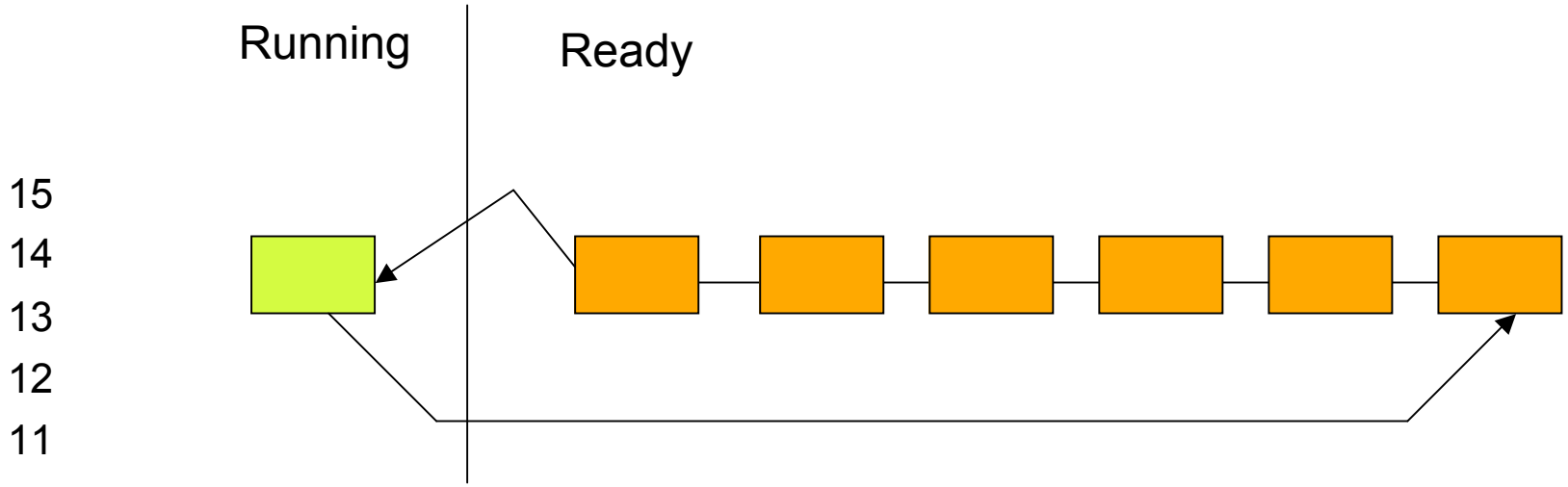


No priority boost for threads in real-time range

Scheduling Scenarios – Quantum end

Running thread exhausts CPU quantum:

- Should the thread‘s priority be decremented?
- Should another thread be run on the processor?



Context Switching

Thread‘s context and context switching are arch-specific

- Context switch requires saving/loading of these data:
  - Program counter
  - Processor status register
  - Other register contents
  - User and kernel stack pointers
  - Pointer to address space in which thread runs (page table directory)

Kernel saves this info by pushing it on kernel stack

- Kernel saves kernels stack pointer in KTHREAD block and loads new thread‘s kernel stack address
- Loads new thread‘s context& page tab.dir.; flushes TLB
- Pending kernel APCs are delivered (IRQL 1)
- Control passes to new thread‘s PC; execution resumes

Idle Thread

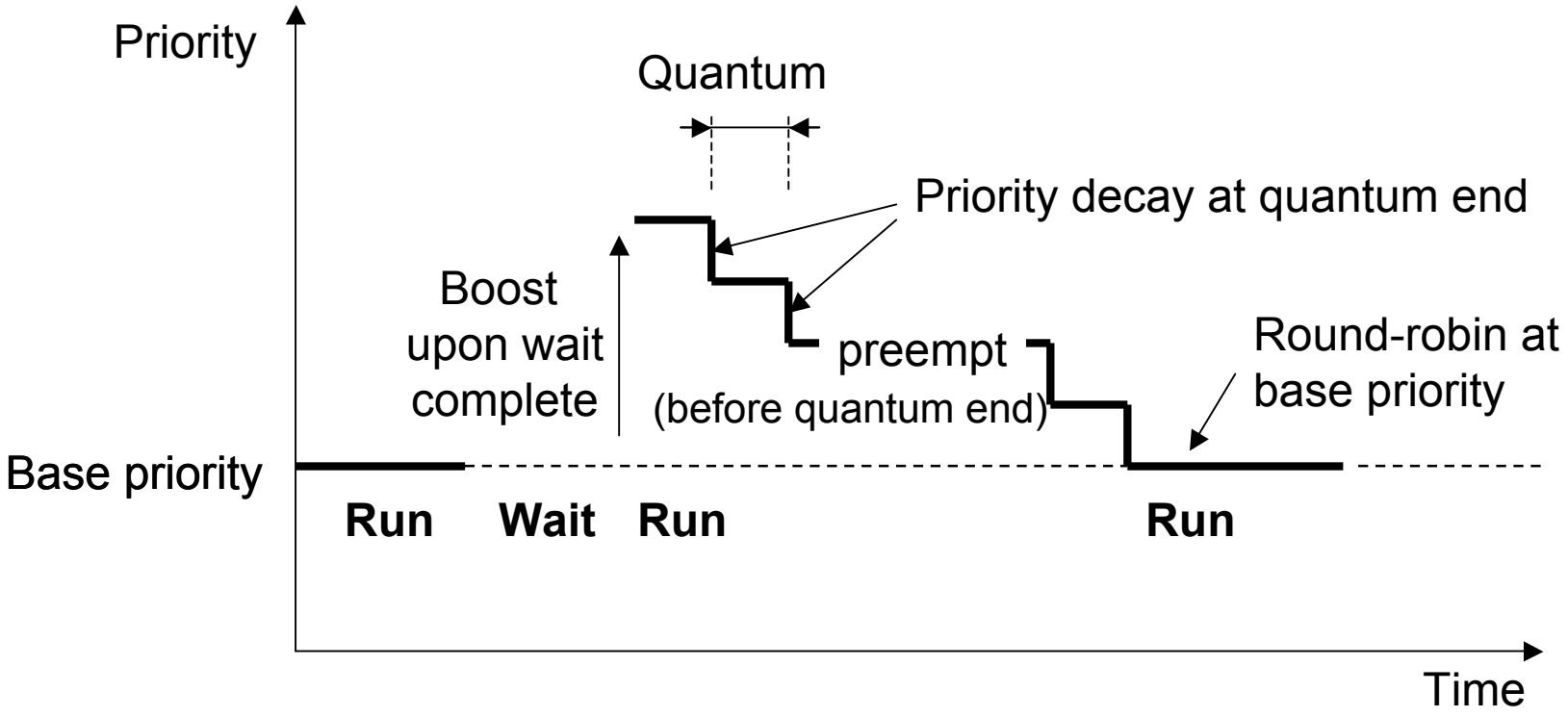
- NT dispatches idle thread when no runnable thread exists on a CPU
- Idle thread has no priority (reported as 0); runs at IRQL 2
- Control flow of idle thread:
  - Enable/disable interrupts (allow pending ints to be delivered)
  - Checks, whether DPCs are pending
  - Checks, whether thread has been selected to run next on CPU; if so: dispatches that thread
  - Calls HAL idle routine (to perform power management)

- Varying names: *System Idle Process* (TaskManager), *Idle* (Pview/Pviewer), *Idle Process* (Pstat), *System Process* (Tlist/Qslice)

Adjusting Thread Scheduling

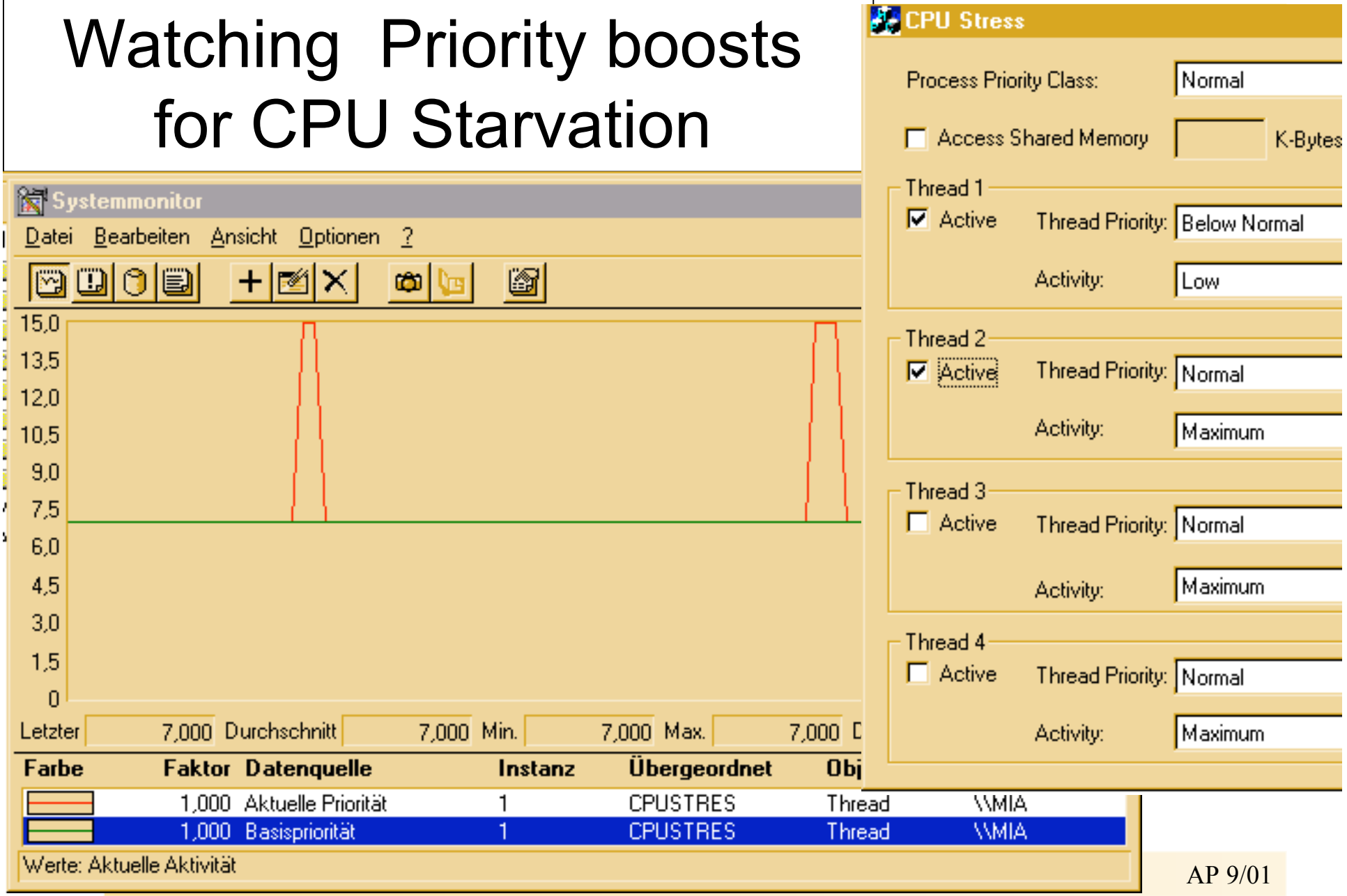
- Quantum stretching for threads in foreground process
  - New to Windows NT 4.0
  - Before: increase base priority for foreground threads; problem: starvation of background processes
- Boosting priority upon wait completion
  - Suggested values in \ddk\include\ntddk.h
  - Event: 1, Disk/CD: 1, Network: 2, Keyboard/Mouse: 6, Sound: 8
  - Priority drops slowly, one level per quantum
- Boosting priority for threads entering a wait state
  - CSRSS boosts GUI thread waiting for windows message to 14
  - Priority drops immediately; thread runs for double quantum at high prio
- Boosting priority for threads not getting CPU time

Priority boosting and decay



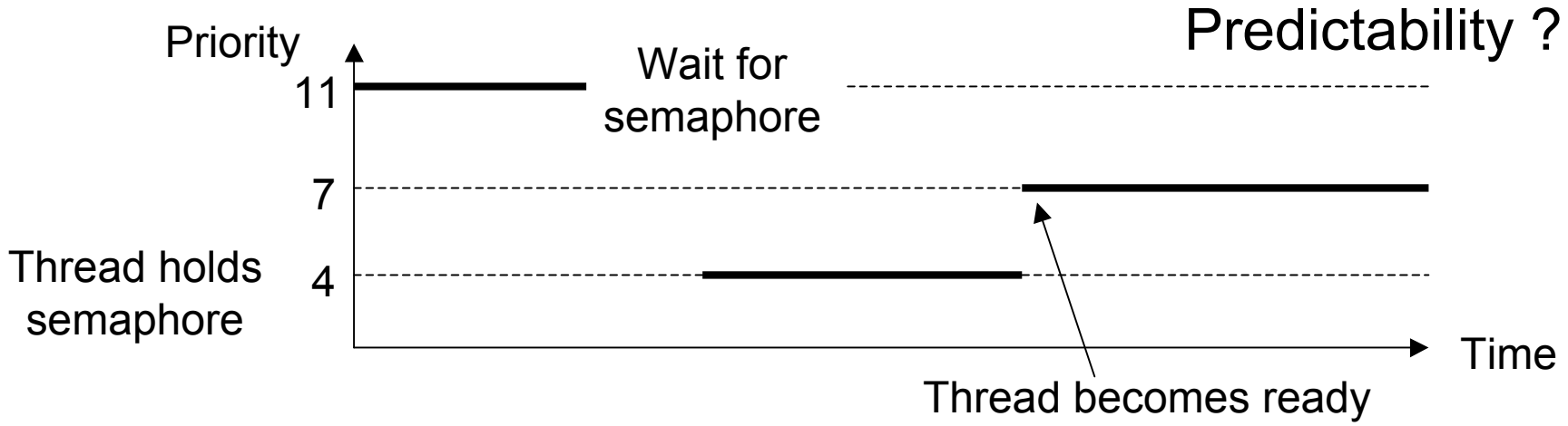
- Boosted thread can still be preempted by another thread
- Threads will never be boosted into real-time priority range

Watching Priority boosts for CPU Starvation



Boosted thread can still be preempted

Priority boosts for CPU starvation – priority inversion



Lower priority thread blocks high priority thread

- Balance set manager scans ready queue for threads that have not run for longer than 300 clock ticks (1/sec)
- Boosts prio to 15; double quantum; no more than 16 threads checked; no more than 10 threads boosted

Scheduling on MP system

- NT attempts to schedule highest priority thread on all available CPUs
- Affinity (SetProcessAffinityMask/SetThreadAffinityMask)
- Ideal processor/next processor
  - Stored in kernel thread block
  - Ideal processor is choosen randomly; not changed by NT
- Selecting a new thread for a CPU:
  - Ran last on specified processor
  - Has ideal processor set to specified processor
  - Has been waiting for longer than 2 quantums
  - Has priority greater than or equal to 24
- Threads are not moved in order to free up CPU