Unit 3: Concurrency

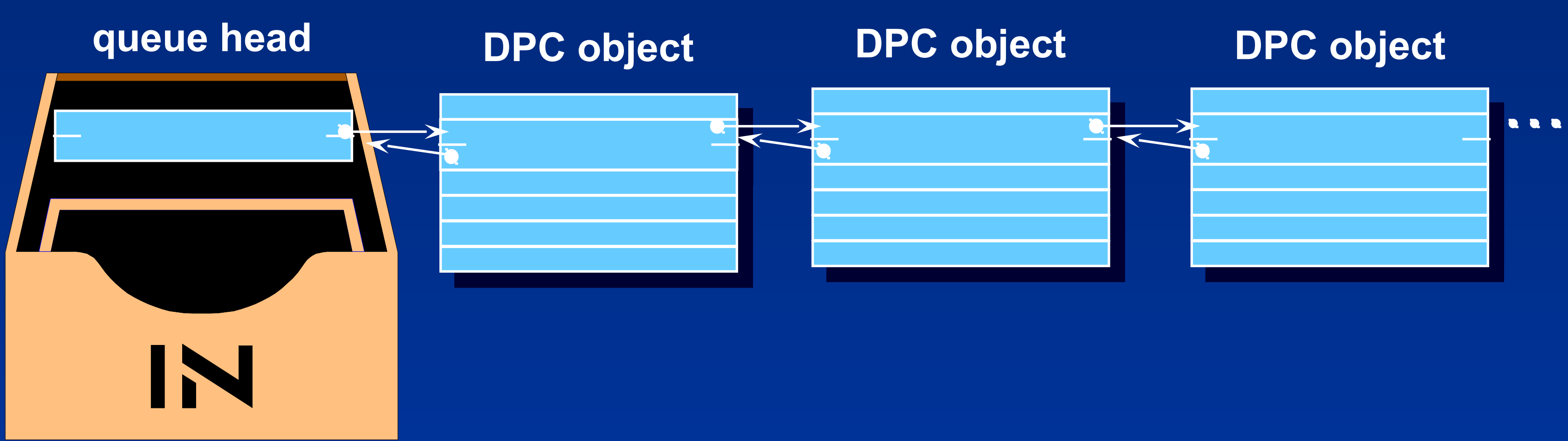
3.3. Advanced Windows Synchronization

Roadmap for Section 3.3.

- Deferred and Asynchronous Procedure Calls
- IRQLs and CPU Time Accounting
- Wait Queues & Dispatcher Objects

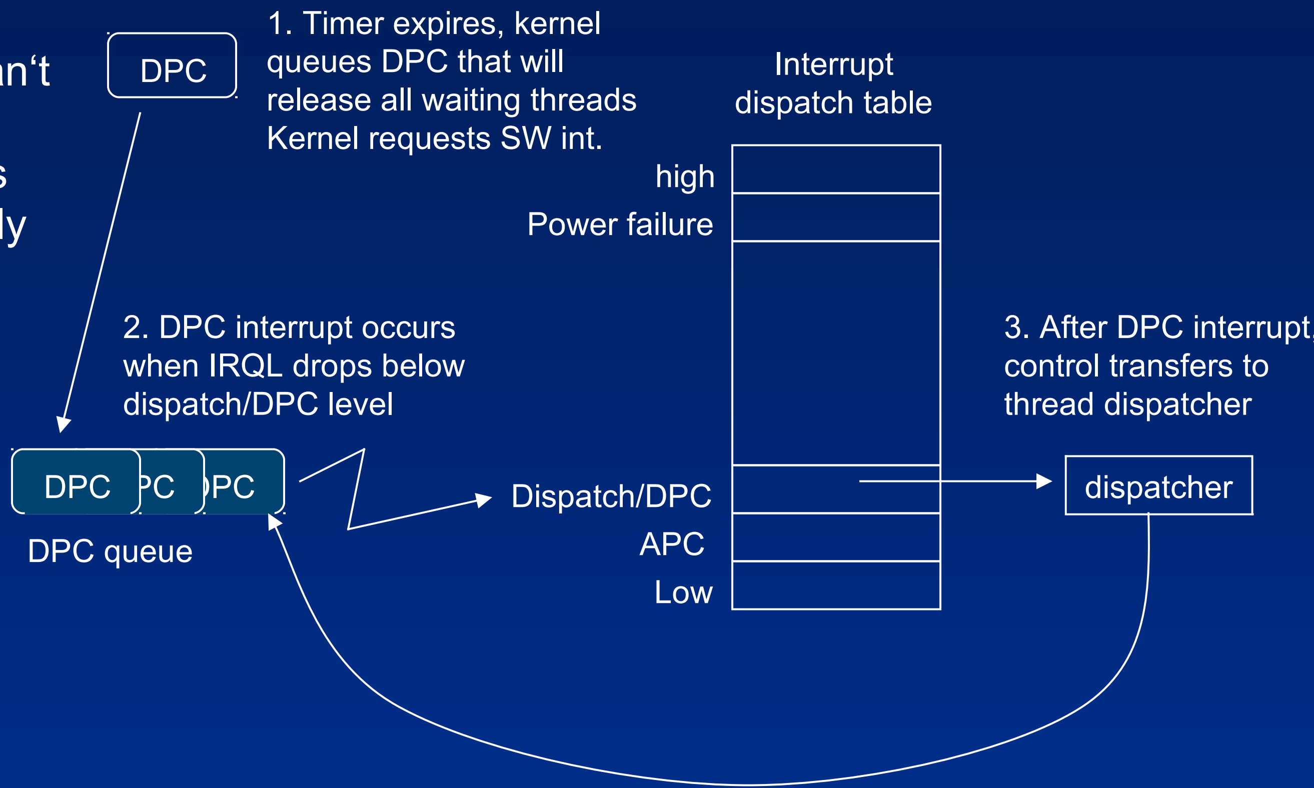
Deferred Procedure Calls (DPCs)

- Used to defer processing from higher (device) interrupt level to a lower (dispatch) level
  - Also used for quantum end and timer expiration
- Driver (usually ISR) queues request
  - One queue per CPU. DPCs are normally queued to the current processor, but can be targeted to other CPUs
  - Executes specified procedure at dispatch IRQL (or “dispatch level”, also “DPC level”) when all higher-IRQL work (interrupts) completed
  - Maximum times recommended: ISR: 10 usec, DPC: 25 usec
    - See http://www.microsoft.com/whdc/driver/perform/mmdrv.mspx



Delivering a DPC

DPC routines can't assume what process address space is currently mapped



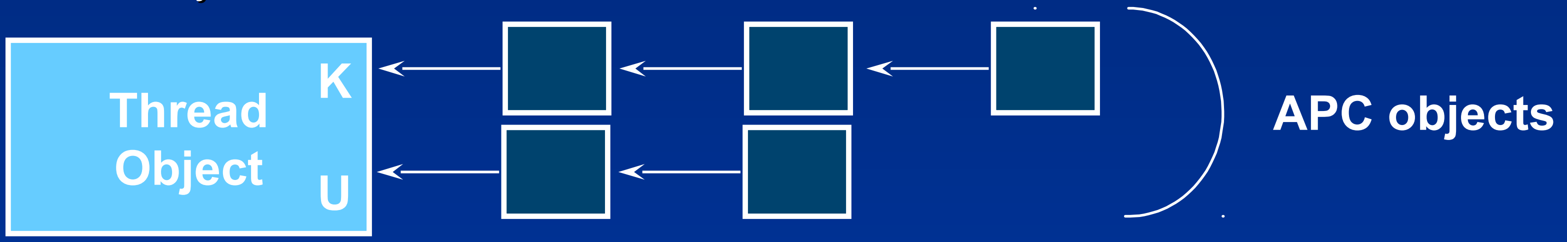
DPC routines can call kernel functions but can't call system services, generate page faults, or create or wait on objects

Asynchronous Procedure Calls (APCs)

- Execute code in context of a particular user thread
  - APC routines can acquire resources (objects), incur page faults, call system services
- APC queue is thread-specific
- User mode & kernel mode APCs
  - Permission required for user mode APCs
- Executive uses APCs to complete work in thread space
  - Wait for asynchronous I/O operation
  - Emulate delivery of POSIX signals
  - Make threads suspend/terminate itself (environment subsystems)
- APCs are delivered when thread is in alertable wait state
  - WaitForMultipleObjectsEx(), SleepEx()

Asynchronous Procedure Calls (APCs)

- Special kernel APCs
  - Run in kernel mode, at IRQL 1
  - Always deliverable unless thread is already at IRQL 1 or above
  - Used for I/O completion reporting from “arbitrary thread context”
  - Kernel-mode interface is linkable, but not documented
- “Ordinary” kernel APCs
  - Always deliverable if at IRQL 0, unless explicitly disabled (disable with KeEnterCriticalRegion)
- User mode APCs
  - Used for I/O completion callback routines (see ReadFileEx, WriteFileEx); also, QueueUserApc
  - Only deliverable when thread is in “alertable wait”

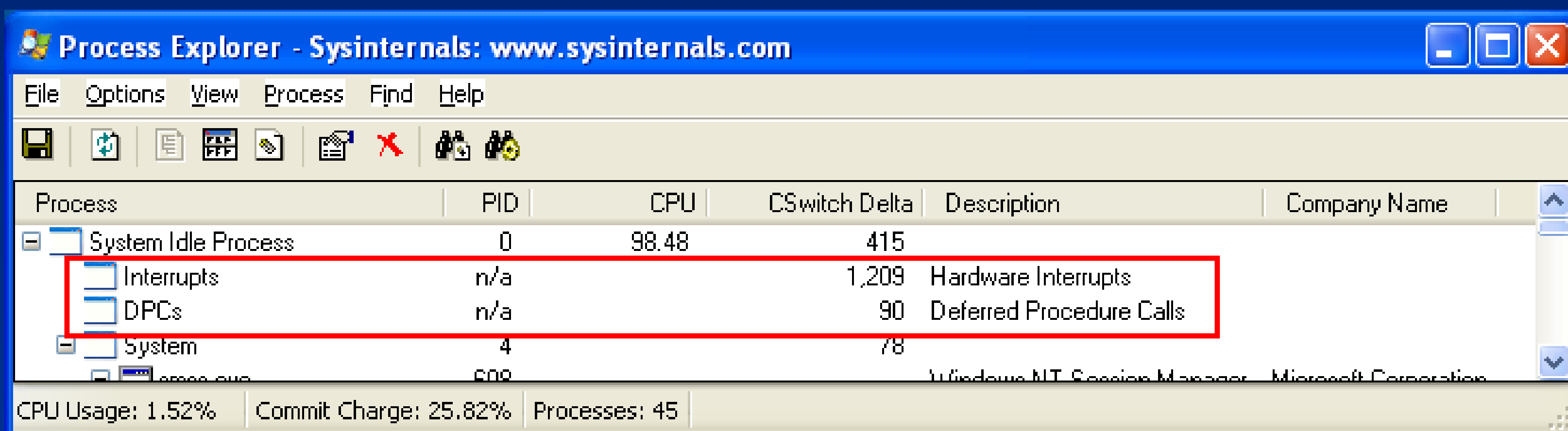


IRQLs and CPU Time Accounting

- System's clock is maintained either by the old PIT (Programmable Interrupt Timer) or the newer RTC (Real Time Clock) device
- PIT uses a 1.193182 MHz clock signal, converted by HAL in 1-15 ms intervals; RTC runs at 32.768 Khz, and the APIC MP HAL converts it in 15.6ms intervals, which is configurable by Windows through special APIs and mechanisms.
- System's clock interval timer, through its ISR, keeps track of time
- Clock's ISR runs at IRQL CLOCK_LEVEL and does time accounting:
  - If IRQL<2, charge to thread's user or kernel time
  - If IRQL=2 and processing a DPC, charge to DPC time
  - If IRQL=2 and not processing a DPC, charge to thread kernel time
  - If IRQL>2, charge to interrupt time
- Since time servicing interrupts are NOT charged to interrupted thread, if system is busy but no process appears to be running, must be due to interrupt-related activity
  - Note: time at IRQL 2 or more is charged to the current thread's quantum (to be described)

Interrupt Time Accounting

- Task Manager includes interrupt and DPC time with the Idle process time
- Since interrupt activity is not charged to any thread or process, Process Explorer shows these as separate processes (not really processes)
  - Context switches for these are really number of interrupts and DPCs



Time Accounting Quirks

- Looking at total CPU time for each process may not reveal where system has spent its time
- CPU time accounting is driven by programmable interrupt timer
  - Normally 10 milliseconds (15 ms on some MP Pentiums)
- Thread execution and context switches between clock intervals NOT accounted
  - E.g., one or more threads run and enter a wait state before clock fires
  - Thus threads may run but never get charged
- View context switch activity with Process Explorer
  - Add Context Switch Delta column

Looking at Waiting Threads

- For waiting threads, user-mode utilities only display the wait reason
- Example: pstat

Command Prompt

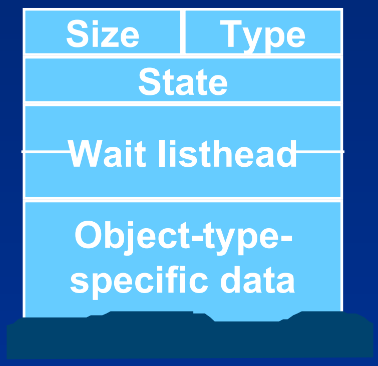
```
C:\WINDOWS\SYSTEM32>pstat
Pstat version 0.3:  memory: 130480 kb  uptime:  0 21:24:36.734
      :
pid:  0 pri: 0 Hnd:    0 Pf:      1 Ws:     16K Idle Process
 tid pri Ctx Swtch StrtAddr    User Time   Kernel Time  State
   0   0   2845450        0 0:00:00.000 20:55:56.375 Running
   0   0   3056193        0 0:00:00.000 21:09:33.234 Running

pid:  2 pri: 8 Hnd:  221 Pf:   1875 Ws:    200K System
 tid pri Ctx Swtch StrtAddr    User Time   Kernel Time  State
   1   0     21214 801c3f6c 0:00:00.000  0:00:39.687 Wait:FreePage
   3  16        51 8010ba7a 0:00:00.000  0:00:00.000 Wait:EventPairLow
   4  16     45518 8010ba7a 0:00:00.000  0:00:00.906 Wait:EventPairLow
      :
pid: 9e pri: 8 Hnd:   78 Pf:   8711 Ws:   1140K Explorer.exe
 tid pri Ctx Swtch StrtAddr    User Time   Kernel Time  State
  48  14    122844 77f052ec 0:00:04.703  0:00:26.312 Wait:UserRequest
  64   8       826 77f052e0 0:00:00.015  0:00:00.140 Wait:UserRequest
  a5  14     23048 77f052e0 0:00:04.140  0:00:11.562 Wait:UserRequest
  a6  14      4976 77f052e0 0:00:00.203  0:00:00.921 Wait:UserRequest
  a7  14      1378 77f052e0 0:00:00.000  0:00:00.000 Wait:LpcReceive
```

- To find out what a thread is waiting on, must use kernel debugger

Wait Internals 1:
Dispatcher Objects

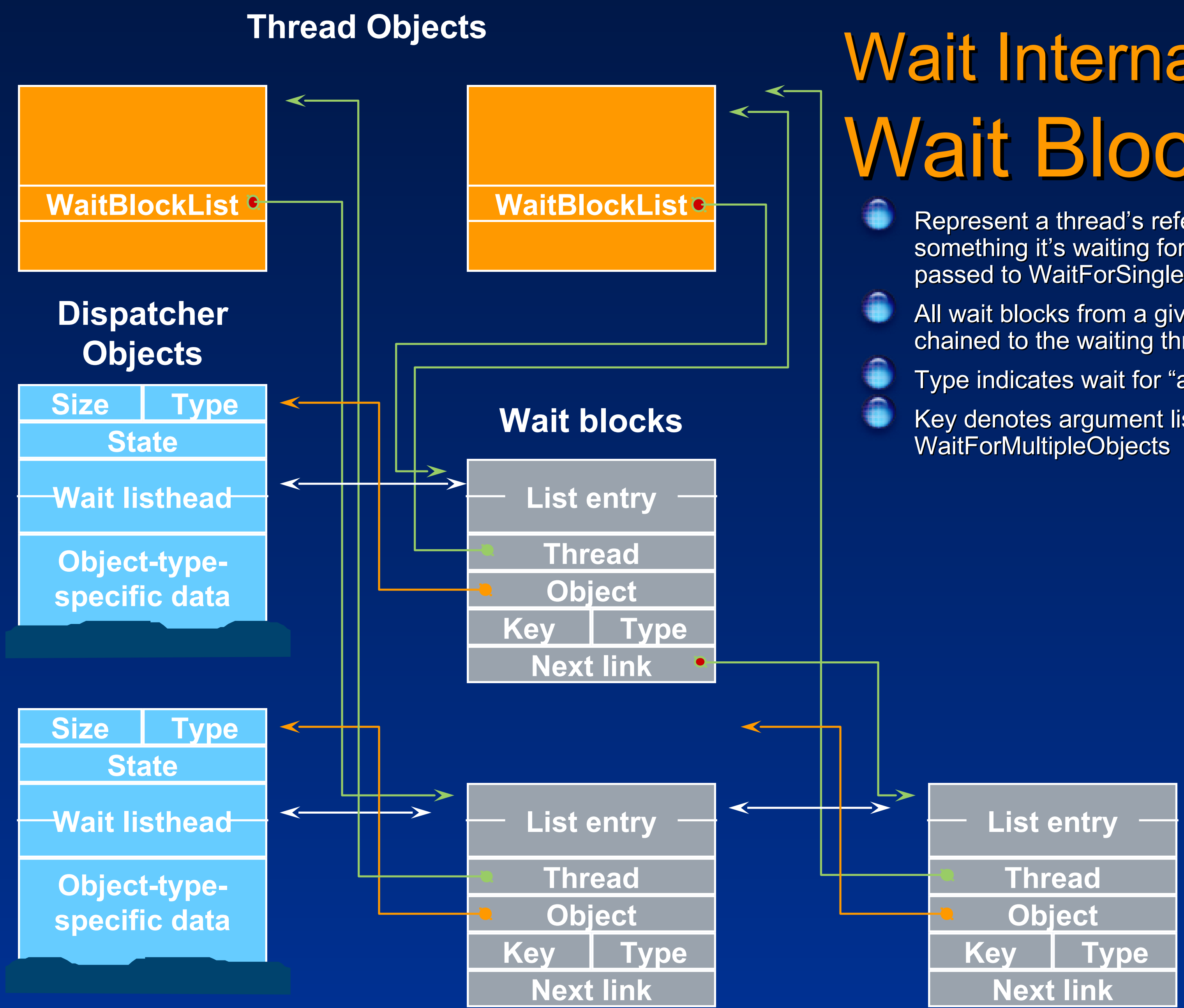
- Any kernel object you can wait for is a "dispatcher object"
  - some exclusively for synchronization
    - e.g. events, mutexes ("mutants"), semaphores, queues, timers
  - others can be waited for as a side effect of their prime function
    - e.g. processes, threads, file objects
  - non-waitable kernel objects are called "control objects"
- All dispatcher objects have a common header
- All dispatcher objects are in one of two states
  - "signaled" vs. "nonsignaled"
  - when signalled, a wait on the object is satisfied
  - different object types differ in terms of what changes their state
  - wait and unwait implementation is common to all types of dispatcher objects

Dispatcher Object



(see \ntddk\inc\ddk\ntddk.h)

Wait Internals 2: Wait Blocks



- Represent a thread's reference to something it's waiting for (one per handle passed to WaitForSingle/Multiple…)
- All wait blocks from a given wait call are chained to the waiting thread
- Type indicates wait for "any" or "all"
- Key denotes argument list position for WaitForMultipleObjects

Further Reading

- Mark E. Russinovich, David A. Solomon and Alex Ionescu,

  "*Windows Internals*", 6th Edition, Microsoft Press, 2012.
- Chapter 3 - System Mechanisms
  - DPC interrupts (from pp. 104)
  - APC interrupts (from pp. 110)
  - Low-IRQL Syncronization (from pp. 183)
  - Kernel Event Tracing (from pp. 220)

  Remark: this chapter will be in part 2 of 7th edition!