SAC-593T

New Techniques to Develop Low Latency Network Apps

Ben Schultz
Osman Ertugay
Ed Briggs
Microsoft Corporation

Pay

Pre-Pay

Agenda

- What’s new and why it matters
- How we improved I/O
- How to use Registered I/O (RIO)

You’ll leave with examples of how to

- Create low latency apps with Windows Server 8

The Problem: Low Latency and Predictability …At Scale

- Today’s multicast aggregate stock market feed: 5 million updates/second
- 300,000 packets per second
- 600,000 packets per second
- Traffic rates are increasing

For many performance critical apps
…every microsecond saved means money

In the past…

Approach	Application type	Compatibility with UDP/TCP	Modifications
Sockets	General Apps	Yes	N/A
Hardware Acceleration	Low Latency Apps	No	Extensive

Windows Server 8

Approach	Application Type	Compatibility with UDP/TCP	Modifications
Sockets	General Apps	Yes	N/A
RIO Sockets	Low Latency Apps	Yes	Moderate
Acceleration	Extreme Low Latency Apps		

Your app must move data quickly
…but you don’t want to rewrite your entire
app or depend on custom hardware

Windows Server 8 delivers

Lower Latency

RIO ~15 - **30% reduction** in latency

Better Predictability

Variability (stdev) reduced by a **factor of 7**

Maximum values reduced by a **factor of 5**

Higher Throughput

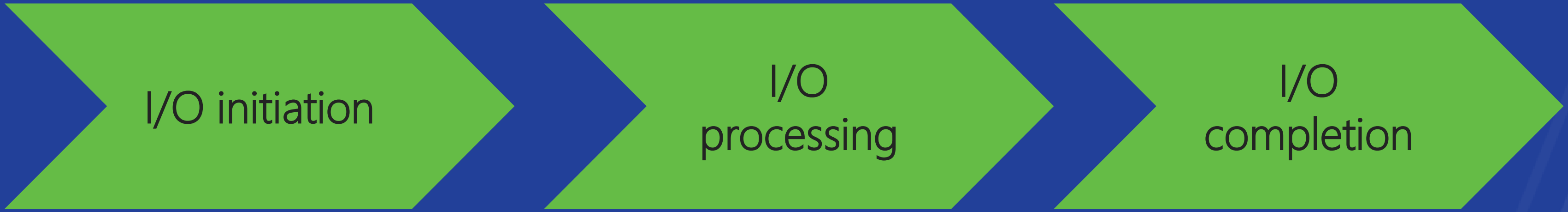
Windows Server 2008R2 sustains ~2 Million datagrams per second

With RIO, we have seen **double** the datagrams per second

Winsock I/O Model

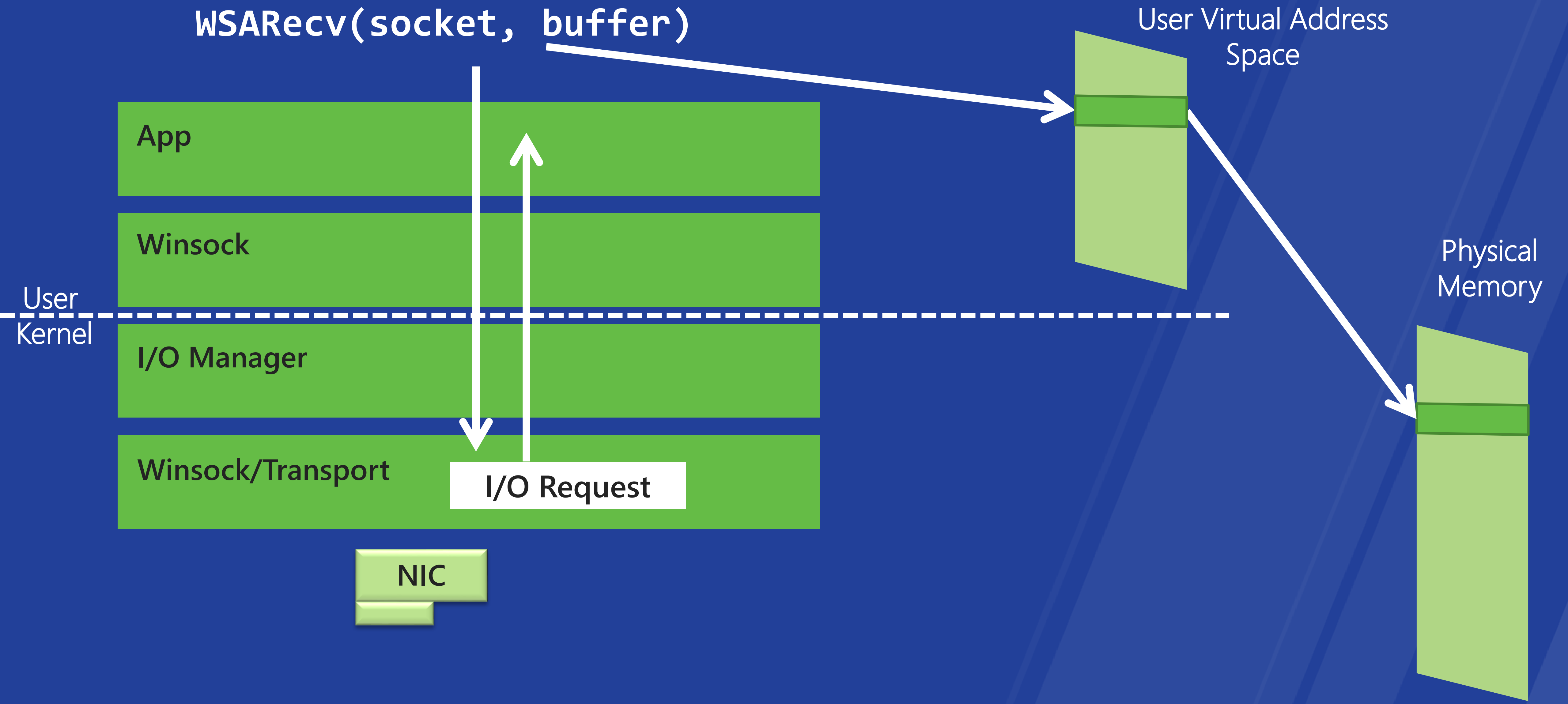
I/O Model, Phases, Elements

- Windows Overlapped I/O Model



- Socket handles → True OS handles
- Data buffers provided by the app
- Multiple completion methods supported by the OS

I/O Initiation



```
WSARecv(socket, buffer)
```

App

Winsock

User

Kernel

I/O Manager

Winsock/Transport

I/O Request

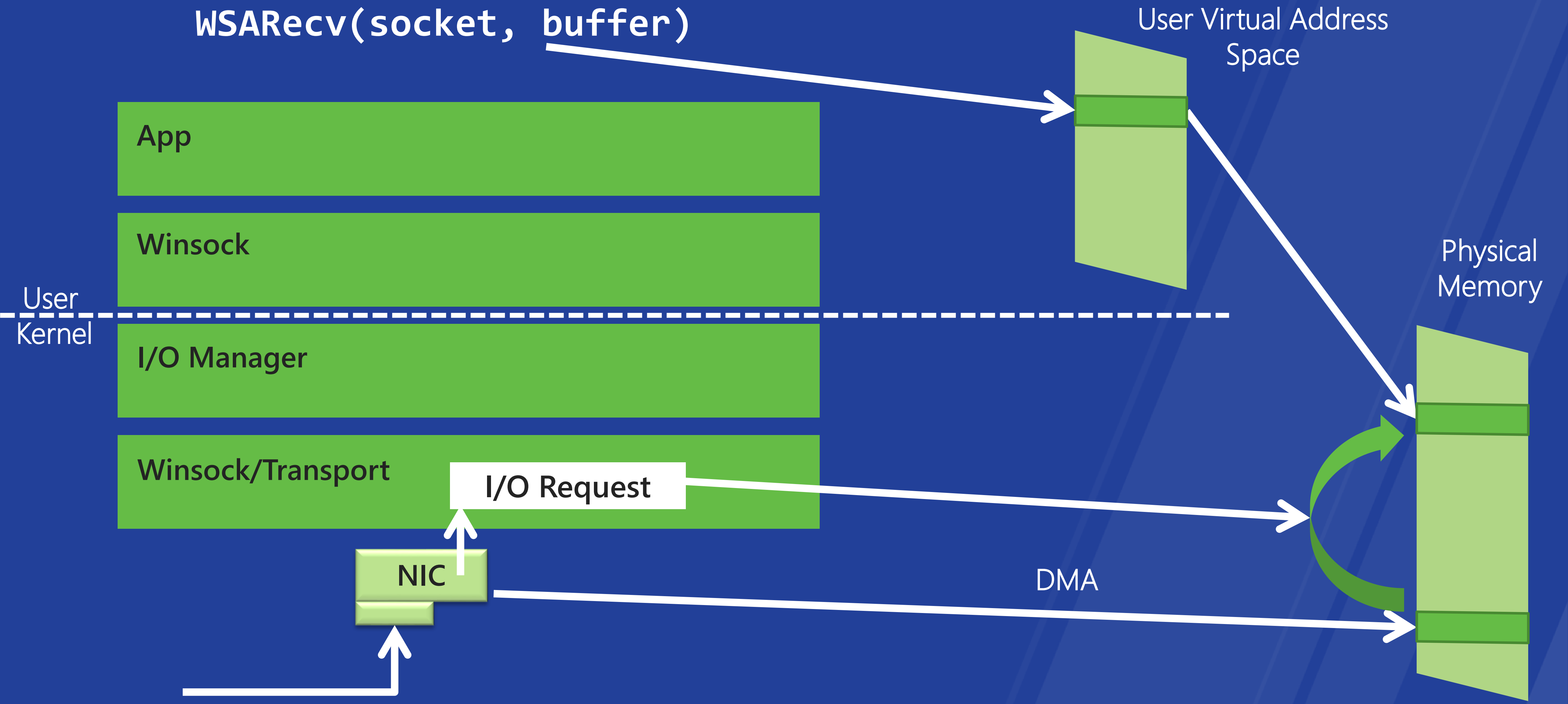
NIC

User Virtual Address Space

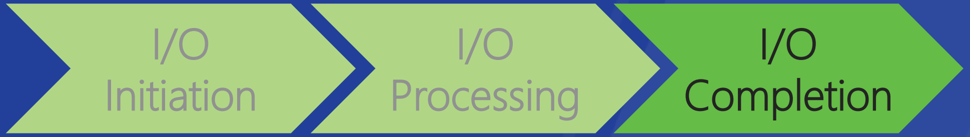
Physical Memory

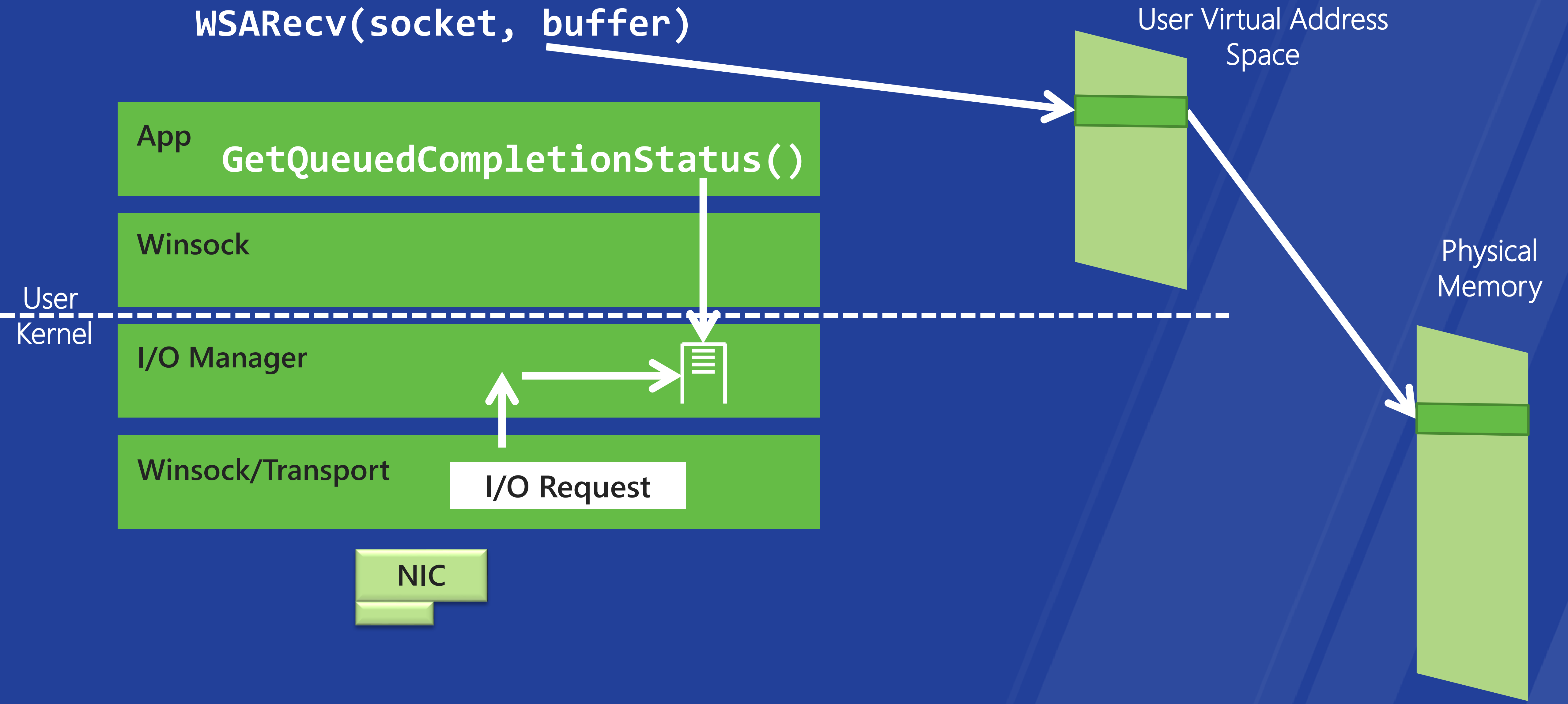
I/O Processing

I/O Initiation → I/O Processing → I/O Completion



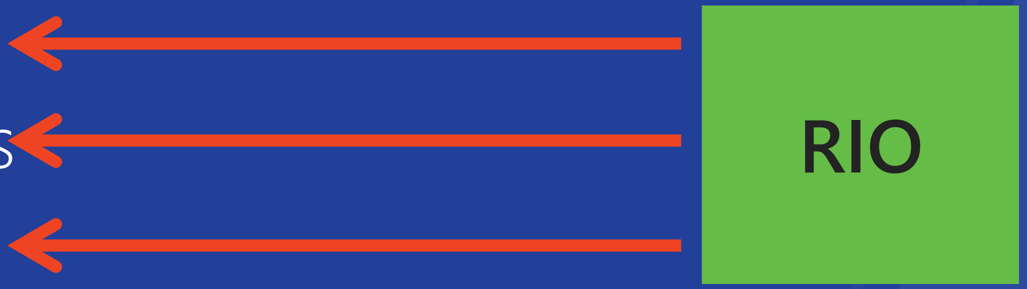
I/O Completion





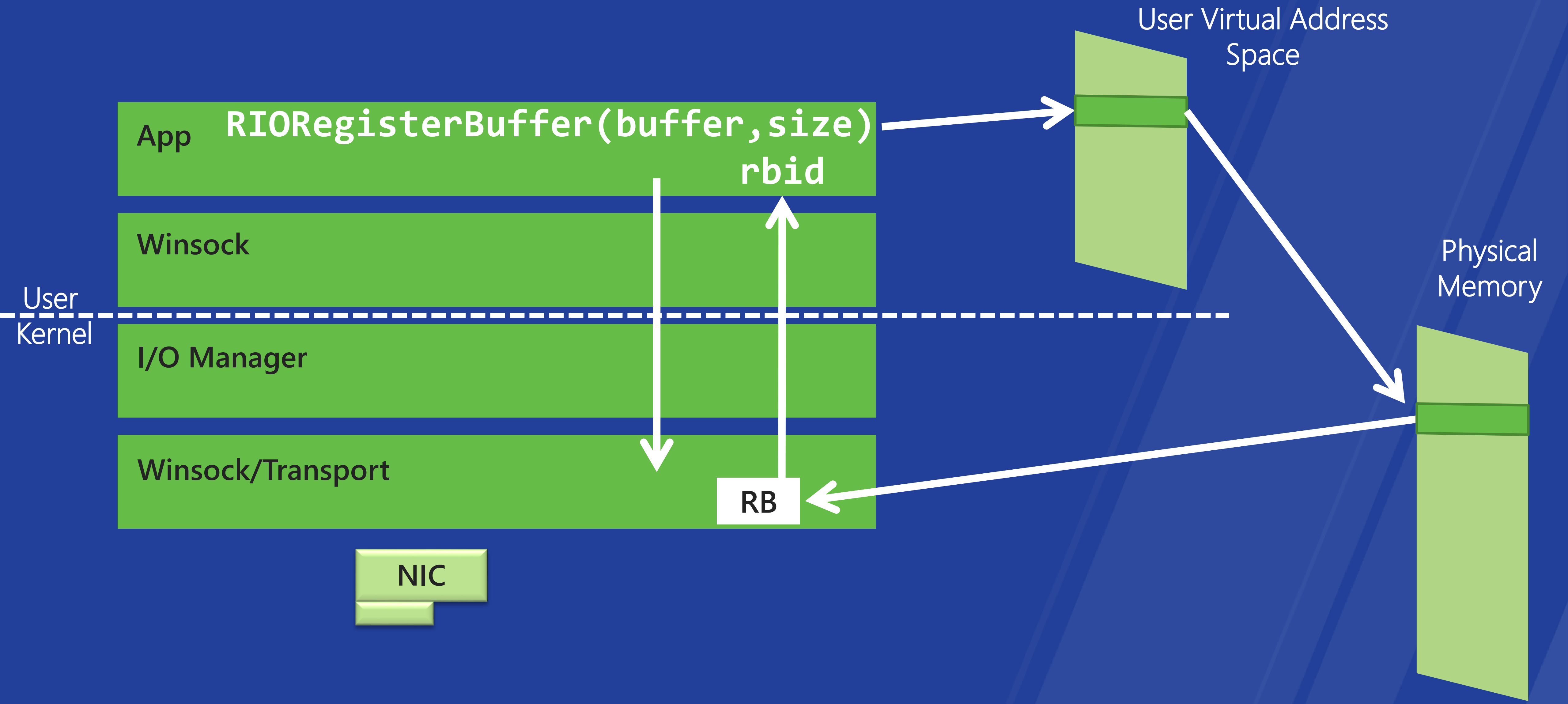
Summary

- Single I/O request may involve
  - ~~Memory lock & unlock~~
  - ~~Multiple handle look-ups~~
  - ~~Multiple system calls~~
- Works fine for general Windows I/O
- Can do much better for demanding apps

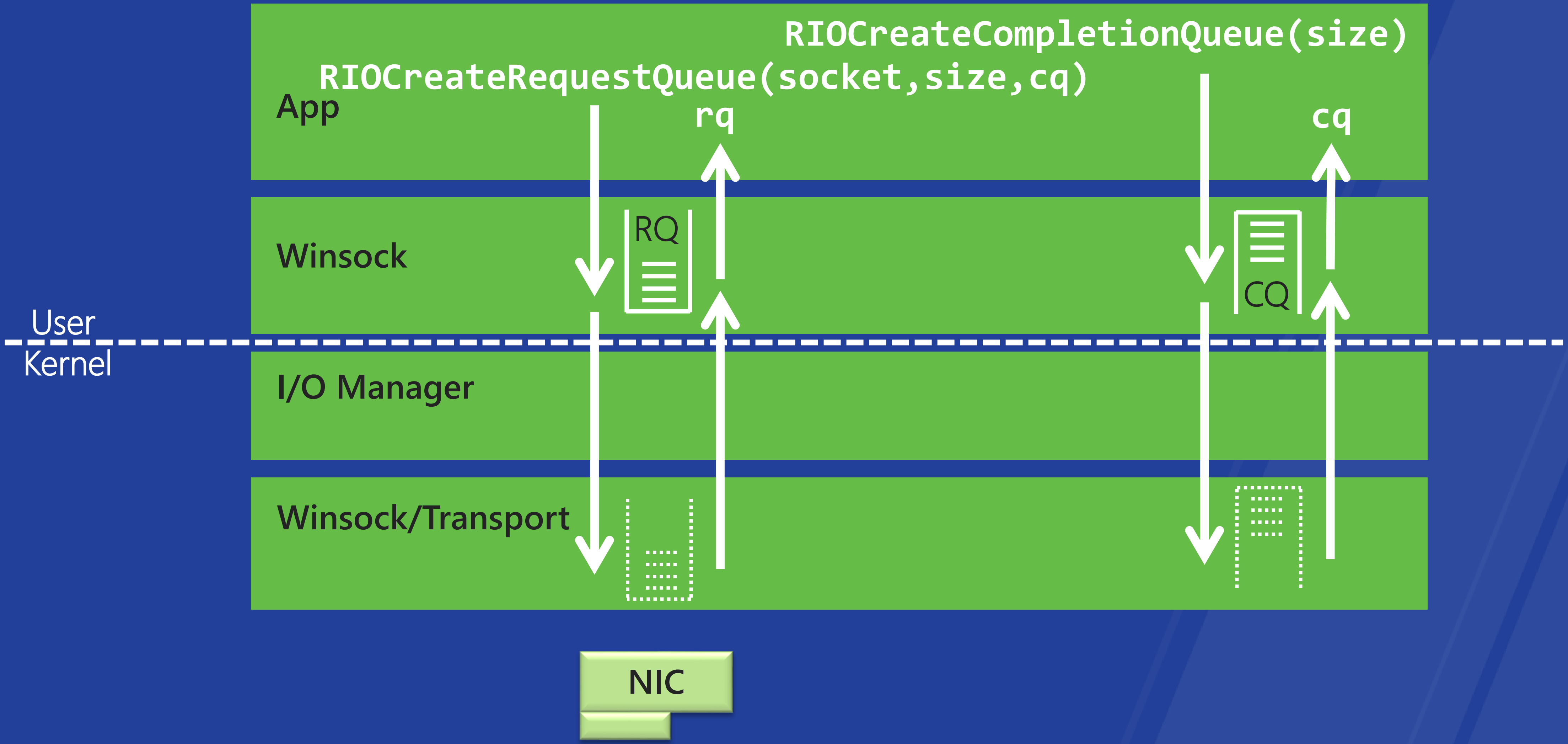


RIO

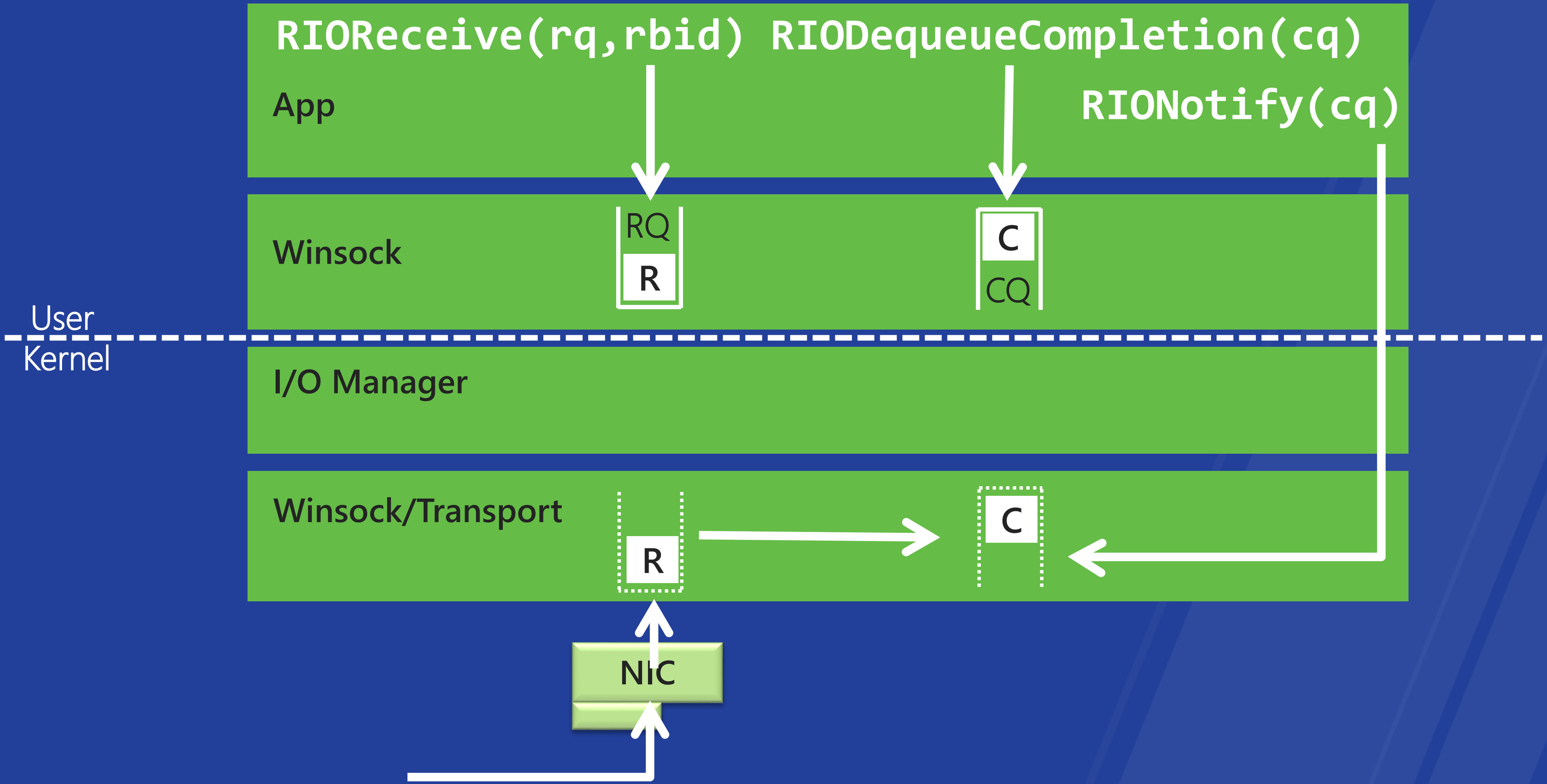
RIO Buffer Handling



RIO Queues


RIOCreateCompletionQueue(size)
RIOCreateRequestQueue(socket,size,cq)
App
rq
cq
Winsock
RQ
CQ
User
Kernel
I/O Manager
Winsock/Transport
NIC

RIO I/O Initiation & Completion

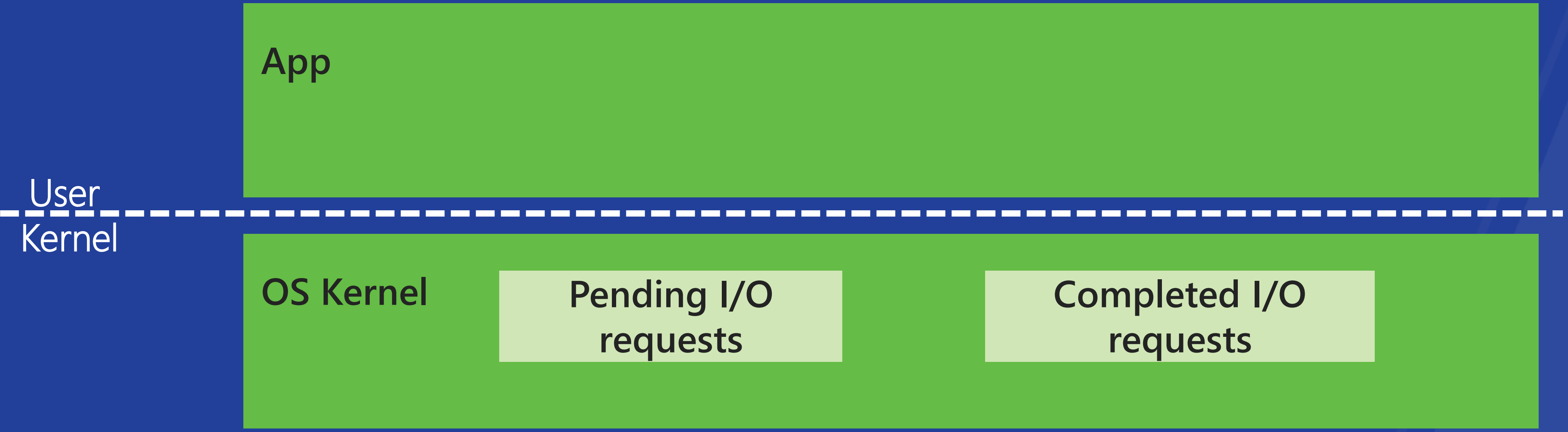


RIO Summary

- Separated I/O buffer handling from actual I/O

RIO Summary

- Separated I/O buffer handling from actual I/O



- Reduced I/O cost, improved predictability

Key RIO Programming Considerations

- Buffer handling moved to app
  - SO_SNDBUF, SO_RCVBUF are not applicable for RIO
  - Ensure to prepost enough requests
    - To avoid dropping incoming packets
    - To avoid stalling the send pipe
- RIO buffers, RQs, CQs always locked in physical memory
  - Trading memory for reduced CPU cycles per I/O

Developing apps with the RIO socket API

RIO Sockets are an extension to Winsock

- Same as regular sockets
  - bind
  - listen
  - accept
  - connect
  - join multicast groups
  - setsockopt (not all options apply to RIO)
  - IOCTL (not all IOCTLs apply to RIO)

- What is New for RIO
  - RIOSend, RIOSendEx, RIOReceive, RIOReceiveEx
  - RIOCreateRequestQueue (and related)
  - RIOCreateCompletionQueue (and related)
  - RIODequeueCompletion
  - RIONotify
  - RIORegisterBuffer

RIO Socket API Concepts

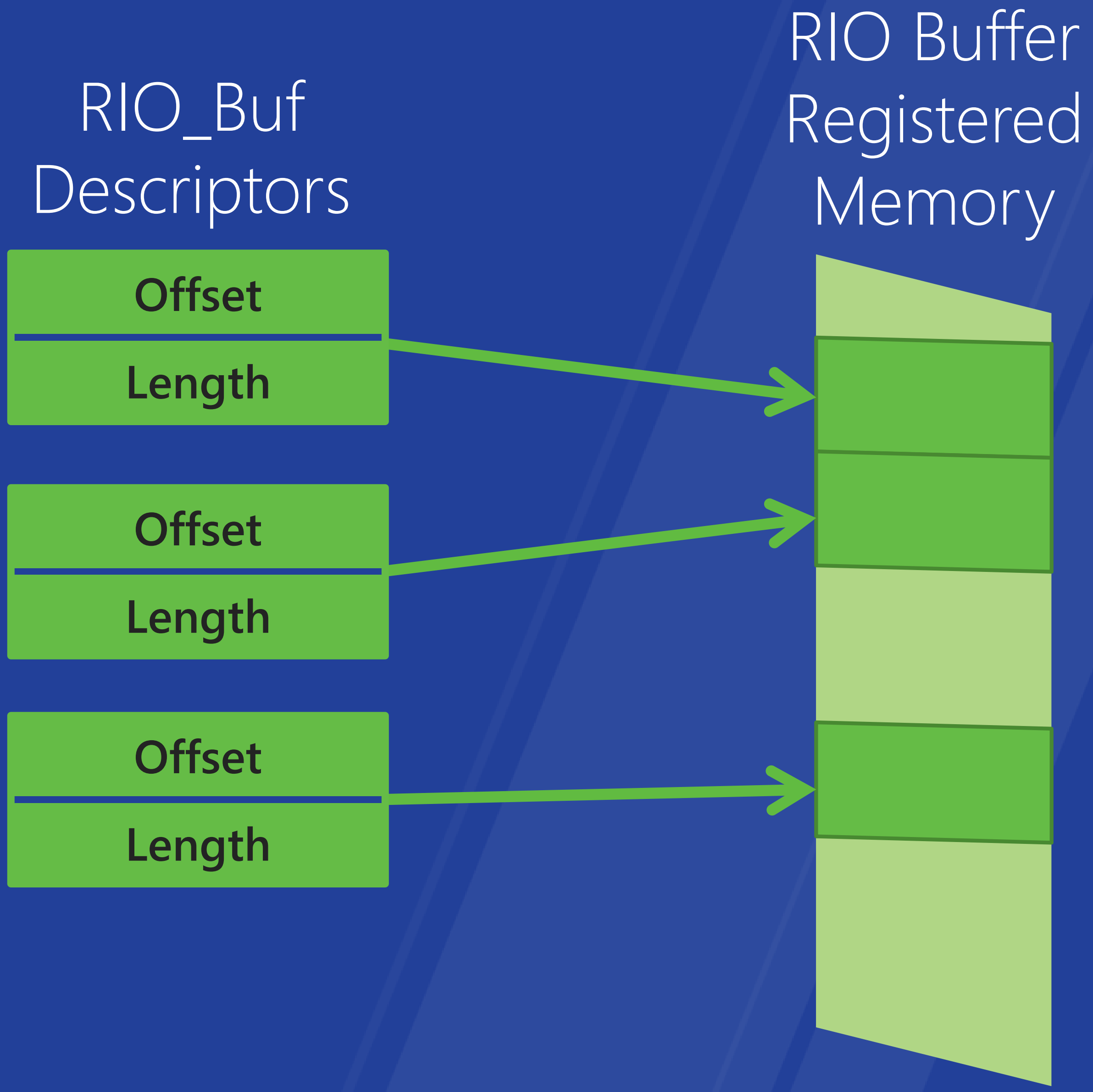
- All RIO IO is performed with Registered Buffers
- Typical use:
  - Allocate a large buffer area
    - Malloc, new(), HeapAlloc, VirtualAllocExNuma()
  - Register it with **RIORegisterBuffer()**
  - Carve it up with RIO_BUF descriptors
  - When you are all finished (e.g. at shutdown)
    - RIODeregisterBuffer()

Quick Review

RIO API Description

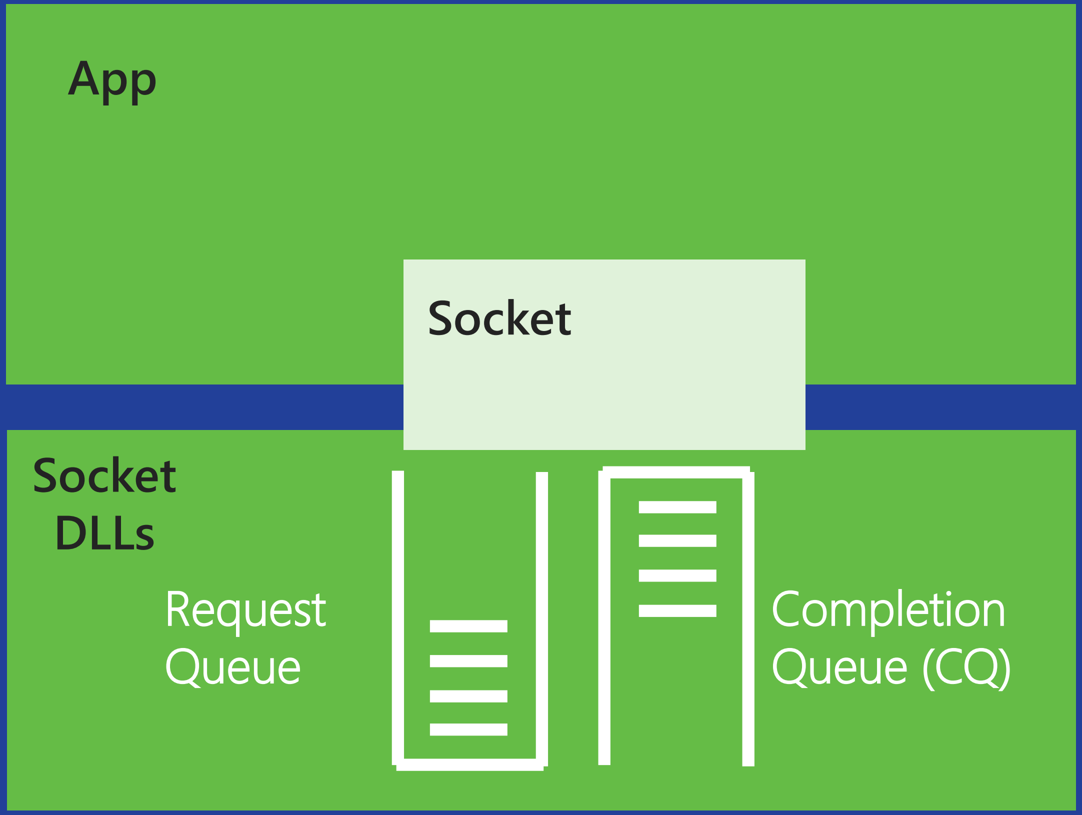
RIO_BUF descriptors are used to carve up the large RIOBUFFER which is ‘locked down’

RIOSend/RIOReceive calls use RIO_BUF descriptors to perform I/O



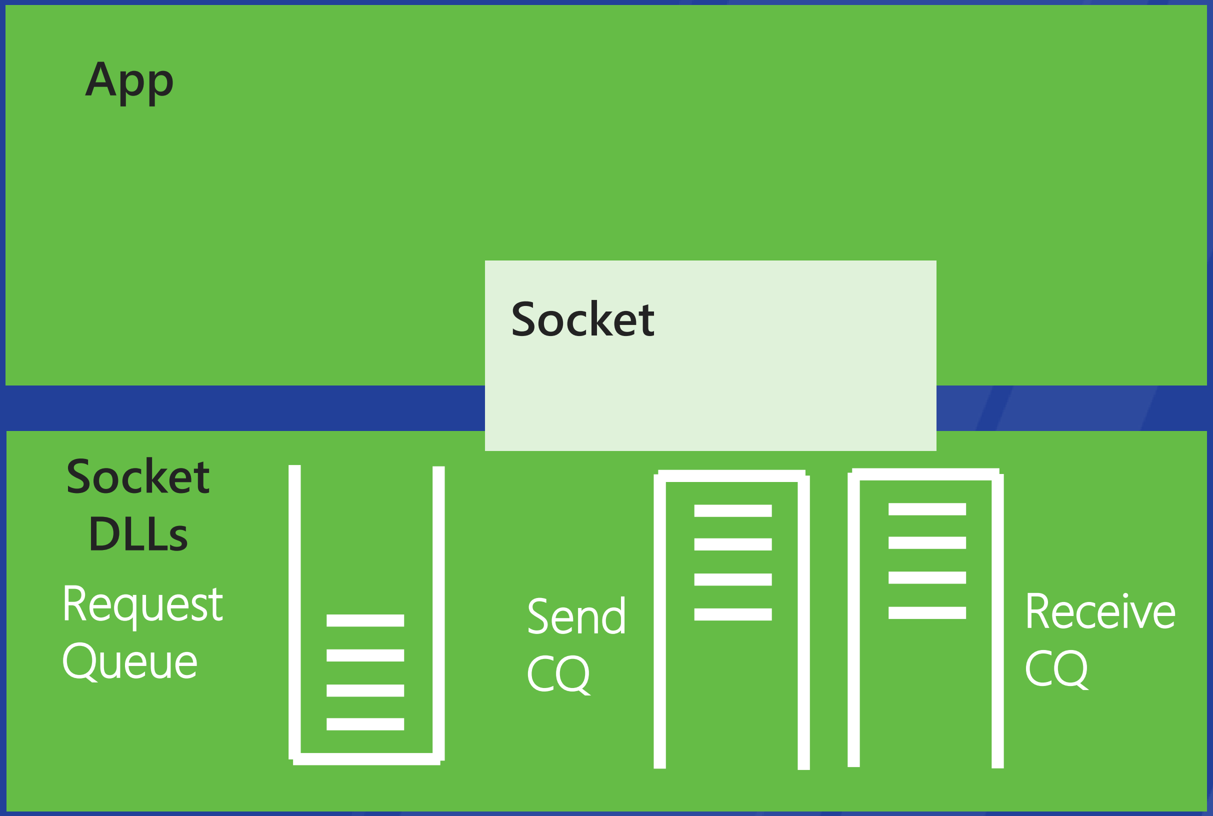
RIO Request and Completion Queues

Each RIO socket has dedicated request queue, and a completion queue



OR

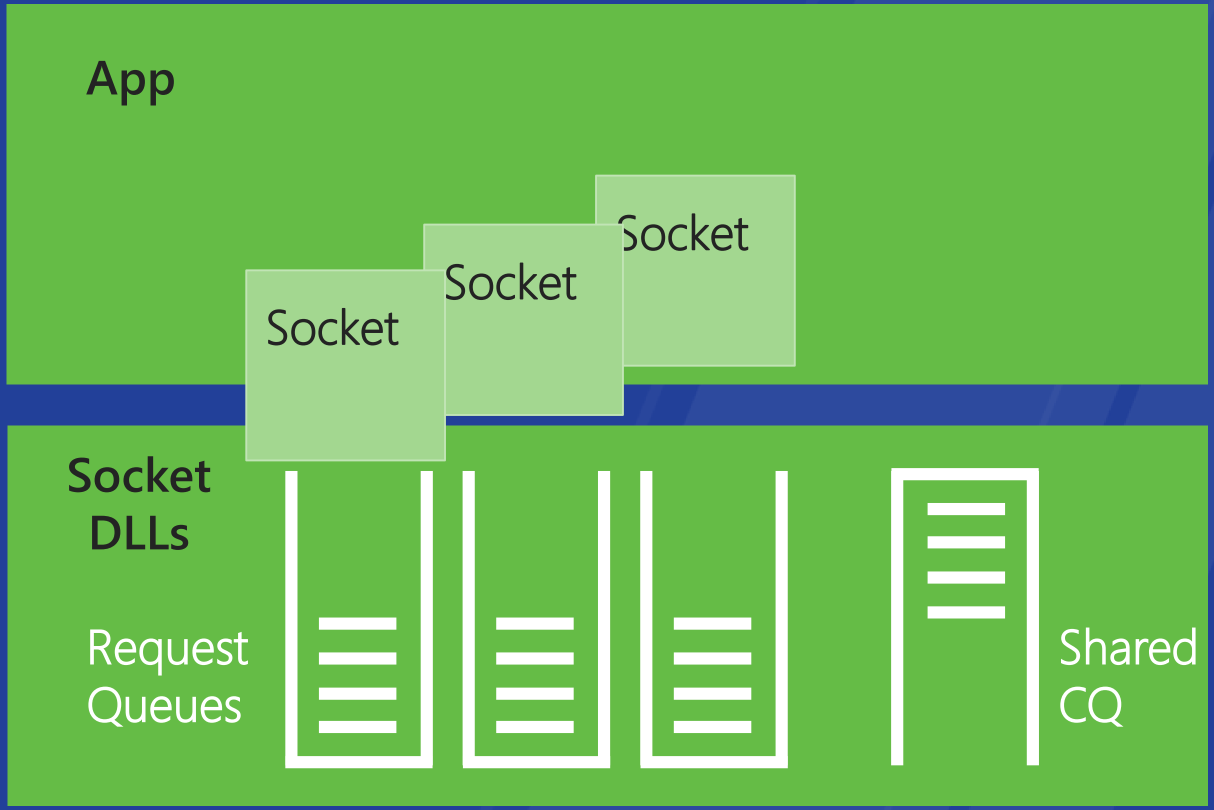
Each RIO socket has separate completion queues for SEND and RECV



RIO Request and Completion Queues

- Completion Queues may be shared, making it easy to handle multiple sockets.

- Alternatively, separate completion queues make it easy to segregate completions to different cores or NUMA nodes.



Creating a RIO Socket

```
SOCKET SocketHandle = WSASocket(
               AF_INET,
               SOCK_DGRAM,
               IPPROTO_UDP,
               NULL,
               0,
               WSA_FLAG_REGISTERED_IO);
```

Creating a RIO Completion Queue

```
RRIO_CQ  RIOCreateCompletionQueue(
          DWORD QueueSize,
          PRIO_NOTIFICATION_COMPLETION CompletionNotificationType)
```

The CompletionNotificationType allows you specify which sort of notification you'd like when and I/O completes:

```
     None – (used when polling)
     Windows IO Completion Port
     Windows Event
```

Creating a RIO Request Queue

```
RIO_RQ CQ = RIOCreateRequestQueue(
            SocketHandle,
            MaxOutstandingReceiveRequests,
            Reserved,
            MaxOutstandingSendRequests,
            Reserved,
            CompletionQueueForReceiveCompletions,
            CompletionQueueForSendCompletions,
            UserSpecifiedPerSocketContextInformation);
```

Send a Message

```
BOOL RIOSend( RIO_RQ SocketQueue,
                PRIO_BUF pData,
                ULONG Reserved,
                DWORD Flags,
                PVOID RequestContext);

//RioSendEx allows you specify other parameters  (e.g. destination address etc.)
```

First Parameter is a Request Queue – not a socket

Send a Message

```
BOOL RIOSend( RIO_RQ SocketQueue,
                PRIO_BUF pData,
                ULONG Reserved,
                DWORD Flags,
                PVOID RequestContext);

//RioSendEx allows you specify other parameters  (e.g. destination address etc.)
```

The data you send is described by a RIO_BUF

Send a Message

```
 BOOL RIOSend( RIO_RQ SocketQueue,
                PRIO_BUF pData,
                ULONG Reserved,
                DWORD Flags,
                PVOID RequestContext);

//RioSendEx allows you specify other parameters  (e.g. destination address etc.)
```

Send a Message

```
 BOOL RIOSend( RIO_RQ SocketQueue,
                 PRIO_BUF pData,
                 ULONG Reserved,
                 DWORD Flags,
                 PVOID RequestContext);

//RioSendEx allows you specify other parameters  (e.g. destination address etc.)
```

I/O completion alternatives

- Polling the RIOCompletionQueue
  - May provide the highest performance and lowest latency, but high CPU utilization
- I/O CompletionPort Notification
  - Easy to integrate RIO with existing IOCP based code
- Windows Event Notification
  - Easy to integrate with existing code that uses Windows Events.

Polling the Completion Queue

```
ULONG NResults = 0;
 RIORESULT Results[MaxResults];

// Poll the completion queue for completions


while (0 == (NResults = RIODequeueCompletion(CQ,
&Results[0], MaxResults))) {

    YieldProcessor();
}
```

Use RIO with IOCP and Polling

```
// Wait for one or more completions, and
// get them all in one operation

  GetQueuedCompletionStatus(IocpHandle …)

  NResults = RIODequeueCompletion(CQ, &Results[0], MaxResults);
```

Wiring up a completion port to RIO Sockets

```
RIO_NOTIFICATION_COMPLETION NotificationCompletion;

NotificationCompletion.Type = RIO_IOCP_COMPLETION;
NotificationCompletion.Iocp.IocpHandle = Iocp;
NotificationCompletion.Iocp.Overlapped = &Overlapped;
NotificationCompletion.Iocp.CompletionKey = NULL;

CQ = Rio.RIOCreateCompletionQueue(QueueSize, &NotificationCompletion);
```

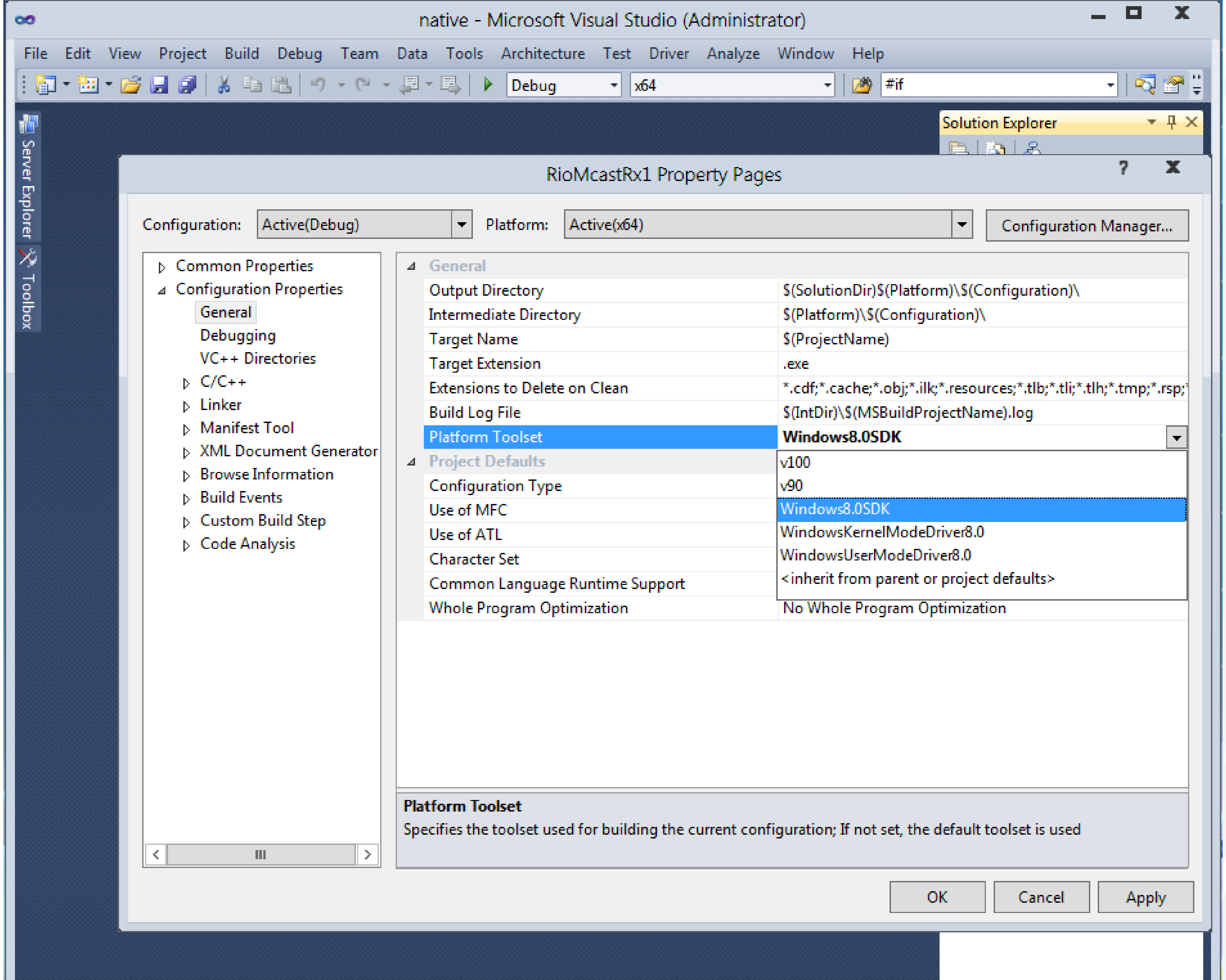
RIO API Summary

- RIO Sockets
  - Extension of WinSock
  - Use Request Queues and Completion Queues
  - Use previously Registered Buffers for I/O
  - Allow for fast polling of I/O completions
  - Can be used with IOCP and Win32 Events for efficient waits

Adding RIO Sockets to Your App

Adding RIO to Your App

- Install Windows Server 8
- Install with Windows 8 SDK
- Install Visual Studio 2010
- Set the Visual Studio C++ Project settings to use new SDK headers/libs
  - Change “Platform Toolset” to Windows 8 SDK
- Write your code
- Compile


native - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test Driver Analyze Window Help
Debug
x64
#if
Solution Explorer
Server Explorer
Toolbox

RioMcastRx1 Property Pages

Configuration: Active(Debug) Platform: Active(x64)

General

Output Directory	$(SolutionDir)$(Platform)\$(Configuration)\
Intermediate Directory	$(Platform)\$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.ilk;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;
Build Log File	$(IntDir)\$(MSBuildProjectName).log
Platform Toolset	Windows8.0SDK

Project Defaults

Configuration Type	
Use of MFC	
Use of ATL	
Character Set	
Common Language Runtime Support	
Whole Program Optimization	No Whole Program Optimization

v100
v90
Windows8.0SDK
WindowsKernelModeDriver8.0
WindowsUserModeDriver8.0
<inherit from parent or project defaults>

Platform Toolset

Specifies the toolset used for building the current configuration; If not set, the default toolset is used

Adding RIO to Your App

Determine if your system supports RIO at run-time

- Check Windows version information.
- Attempt to create a RIO socket. This will fail if RIO is not supported.
- Attempt to retrieve the RIO Function Extensions. This will fail if RIO is not supported.

You can include RIO code in your app safely ...even if the runtime platform doesn’t support RIO

- RIO Functionality won’t work – of course
- But no Runtime Linkage problems
- This also means you could develop your code on Windows 7 and copy it to Windows Server 8 machine for testing.

Adding RIO Sockets to Your App

- Determine the style of completion you want
  - Poll, IOCP, Event
- Determine the sort of buffering you require
  - Varies by receive rate
  - May want to consider lock free lists, or per-NUMA node pools
- Arrange for more asynchronous I/O where necessary
  - For UDP receive, you can't rely on a large socket receive buffer with RIO
  - You many need to post multiple asynchronous receives to create the necessary buffering for high rates of receive traffic
- Instrument your code with Concurrency Visualizer markers and/or ETW

Recap

For many performance critical apps
…every microsecond saved means money.

Windows Server 8 delivers

Lower Latency

RIO ~15 - **30% reduction** in latency

Better Predictability

Variability (stdev) reduced by a **factor of 7**

Maximum values reduced by a **factor of 5**

Higher Throughput

Windows Server 2008R2 sustains ~2 Million datagrams per second

With RIO, we have seen **double** the datagrams per second

Related sessions

- [417] Windows Server 8 Performance and Improvements
- [433] Network Acceleration & Other NIC Technologies for the Datacenter
- [565] Windows Networking with PowerShell:
  A Foundation for Datacenter Management

Microsoft®

The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

//build/

windows