



# **Computers as Real-time Data Recorders and Players. Eliminating the Variables...**

1501 South Sunset Street, Suite C, Longmont, CO 80501  
Telephone: 303.485.2721 Fax: 303.485.5104  
[www.conduant.com](http://www.conduant.com)

## **Computers as Real-time Data Recorders and Players. Eliminating the Variables...**

The personal computer is a wonderful general purpose tool. For the way most people use it, getting the correct result in a reasonable amount of time is all that counts. Variations in time to complete a job and an uneven distribution of processor time across the job go largely unnoticed.

The recording and playback of real-time data to and from disk storage, however, requires minimum average sustained performance levels at all nodes throughout the data path. A delay at one time in the job can not always be made up for by simply running faster later in the job. When the required minimum performance is not maintained, the result is corrupt or missing data in the recorded data stream.

While this paper is written in the context of recording real-time data, the highlighted variables also apply to real-time playback. In real-time playback, a digital stream from disk is sent to the appropriate devices that reproduce the real-time event. Even if data is recorded properly, failure to deliver stored data to the reproduction equipment in a timely manner produces a flawed reproduction of the event.

Many variables affect the bandwidth, reliability, and repeatability that can be attained when using a PC for real-time data recording and playback. These include:

- Disk drive access patterns (dependent on OS and file system),
- Contentions in the data path (based on user implementation, hardware speed, and PC hardware architecture),
- Dependence of data movement on timely OS and application software execution (based on user implementation, PC hardware speed, OS, and user application software), and
- Data buffer sizes (dependent on selected hardware).

Some of the above variables are constant for a given system while others vary from recording to recording on the same system. Whether or not a recording solution is on the edge of becoming unreliable depends on the desired data rates and the variables listed above. These variables make it impossible to produce a single formula that quantifies the minimum sustainable performance of a system. However, **different architectures can reduce or even eliminate the variables that can compromise a successful recording and playback session.**

## Real-time data from sensors to disk...

A real-time event is one where its characteristics change over an elapsed period of time. Repeated sampling of the event by sensors produces a real-time data stream that represents the event in an electronic form. The recorder has no control over the real-time event. It can not cause the event to pause or slow down when there are temporary delays in storing data to magnetic media. The data either gets recorded when it happens or gets lost because real-time data recording is an open-loop process. Thus, the worst performance of a real-time data recorder across the recording must exceed the highest performance level of the real-time data stream or data loss and corruption will occur.

Different sensors (such as CCDs for imaging, microphones for audio, thermocouples for temperature, and so on) are designed to capture different aspects of an event. The user might combine (Figure 1) the data collected from many different sensors to capture a more complete description of the event. Data from these sensors is temporarily stored in data buffers while it is also being removed in a FIFO (First-In-First-Out) order and stored to disk. When the storage system fails to accept data from the buffer at a rate sufficient to keep the buffer from overflowing (Figure 2), the recording becomes corrupt with incorrect or missing data.

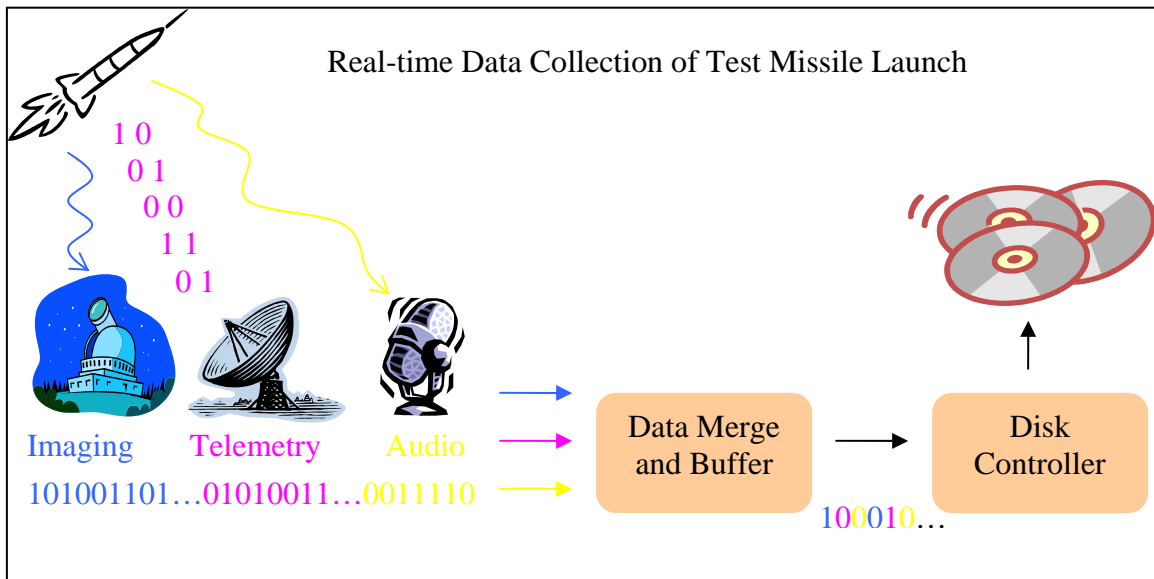


Figure 1 Real-time data recording example.

## Variability of Different Hardware Architectures

When constructing a real-time recording system from a PC, the user may want to consider portability from one computer to another. Dependence on internal PC hardware (chip sets, solid state memory, bus widths, processors, and clock speeds on all of the above) and the OS might result in a configuration that runs well on one computer while intermittently failing on a different computer.

Figure 5 shows a PC based on the Intel 875 chipset. In this architecture, data from the PCI bus is routed through two hubs to system memory. The reverse path is taken when moving data out to the disk controller on the PCI bus. Each hub aggregates data streams for other system functions, thus sharing the real-time data path with many other data processing streams that move into and out of system memory. Though the bus bandwidth increases as one looks higher in the architecture, some of the competitive paths include high-bandwidth traffic such as multiple disk storage ports, multiple USB 2.0 ports, and gigabit Ethernet ports. Most demanding of all are the instruction and data accesses of a high-performance processor to system memory. The impact on the real-time data stream by this other traffic is dependent on the parametrics of this particular hardware architecture, the OS, and what other applications and services are running in the background. As these variables change from one computer to the next, so may the repeatability of the behavior.

Increasingly faster chipsets, memory, and processors can make a marginal recorder a reliable recorder at a given data rate. However, many users already record at rates lower than desired due to the hardware limitations. These users often apply the increased hardware performance to faster data rates rather than improved margin. Thus, the problem remains unsolved.

One system design goal is to **optimize the data path by reducing the number of shared data path segments and thus the number of variables**. This may also reduce buffering requirements and will lead to a more portable solution.

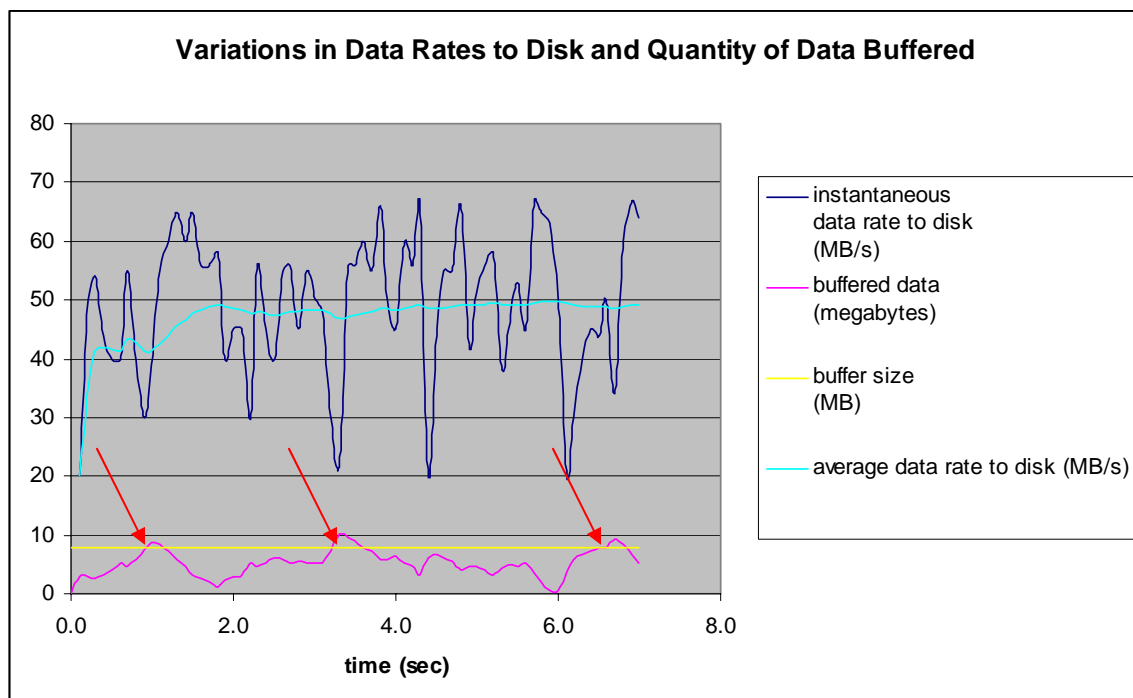
## **Variability of Software Managed Data Paths**

CPU time is a shared and unpredictable resource. Processor execution cycles for other applications and OS services compete with the recorder data path for system memory bandwidth. A real-time design that relies on the timely execution of a general purpose OS and user application introduces unpredictability into the system. To improve repeatable behavior, the user should shut down applications and services that are not relevant to the recording process.

A common implementation is to have the host processor program the DMA engine on a data acquisition card. When the burst is complete, the DMA chip generates an interrupt which is serviced by the OS and results in the programming of the next burst. This non-deterministic implementation relies on timely OS and application servicing of the interrupts. This variable might be somewhat mitigated with additional buffering on the acquisition card.

## Speed-Matching Data Buffers Helps Absorb Variability

Speed matching buffers are safety valves that facilitate a smooth data flow between successive nodes, each of which may have different bus clock speeds and duty cycles. Multiple buffers might be deployed in a system that has many different data path segments. If the data acquisition device is on a PCI card, there is likely a buffer on the card between the A/D converter and the PCI bus. This is because the PCI bus interface is shared and there are times when the bus is not available for the acquisition card to send data. If system memory is part of the data path, then it is also a data buffer between disk and the data source or target (depending on the direction of transfer). It is also common that a disk controller may have a data buffer between its PCI bus connection and the disk drives. This compensates for the “bursty” nature of disk transfers.



**Figure 2** Given an 8MB buffer and a 50MB/s data input rate, the chart shows how instantaneous variations of data rates to disk media affect the amount of data held in a buffer. Red arrows show where failure to move data to disk in a timely manner caused buffer overflows and, thus, lost real-time data.

The required buffer size at any point is dependent on the predictability of the incoming and outgoing data streams around that buffer. For instance, data paths that are being scheduled for transfer by software (OS or application) may require more buffer than those that are hardware controlled. Disk drives that are managed by the OS or application software will be less predictable than those managed with hardware engines. Also, disk drives that are subject to fragmentation (typical of those managed by an OS and file system) will also exhibit wider swings in instantaneous performance. Controlling or, better yet, eliminating the hardware and software variables generally results in the need for fewer and/or smaller buffers.

## **Disk Drive Data Organization: The Biggest Variable of All**

At the time the disk drive is manufactured, each data sector on it is assigned a LBA (Logical Block Addresses). LBAs are assigned as an incrementing number from the first sector at the outer diameter to the last sector at the inner diameter. The order in which they are assigned is generally the order that allows them to be most rapidly accessed from one to the next.

A read or write command from the disk controller specifies a starting LBA and the number of incrementally successive sectors to transfer. Once the command is received, the disk electronics selects the correct head, seeks that head to the track with the starting sector, and then waits for the disk to rotate around to the position where the first requested sector should be. The data transfer begins when the starting sector spins under the head and continues for each successive sector until the length specified in the command has been satisfied. This reading of successive (sequential) sectors is the most efficient manner in which a drive can transfer data to/from the magnetic media. It is common that a single command might request the last several sectors on one track and the beginning sectors on the next track. In this case, a track-to-track seek is executed between the transfers of the last sector on the previous track and the first sector on the next track. The duration of the seek is “wasted” time in that no data is transferring to or from the media. However, track-to-track seeks are a necessity of disk technology. To minimize the wasted time, the manufacturer rotationally skews the first sector on the next track from last sector on the previous track by the time it takes to complete the track-to-track seek. This time is typically from 0.5 to 1.5 milliseconds with current generation disk drives.

When new read and write commands set the starting point for the current access at the sector following the completion of the previous transfer, this is referred to as sequential access. Continuous use of sequential commands, the assistance of read-ahead and write caching, and a very fast command dispatcher in the disk controller make it possible for the drive to continuously operate at its theoretical maximum speed from the outer to inner diameters of the disks. In this mode, data is transferring between the media and the head for most of every revolution, pausing only for track-to-track delays. It doesn't get any better than that.

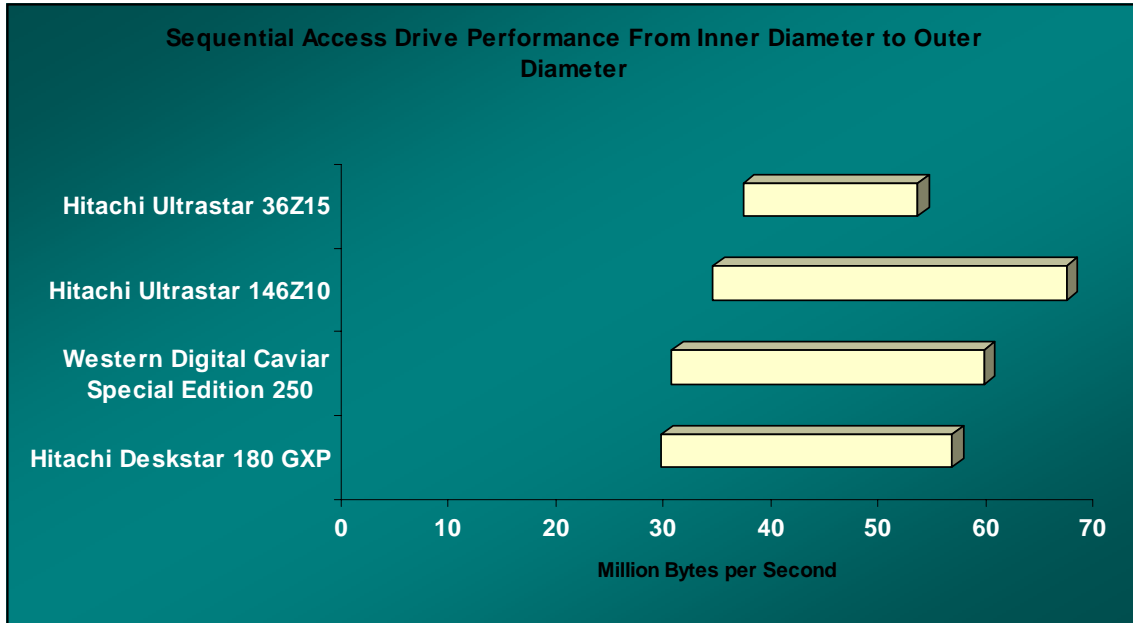


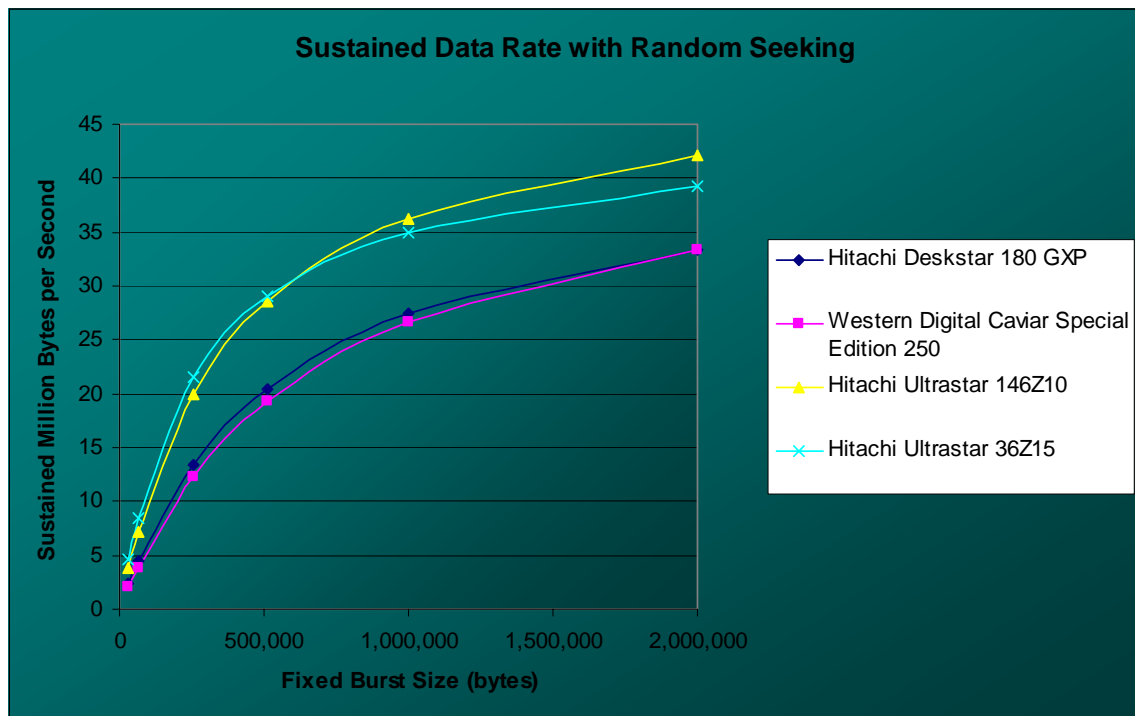
Figure 3 Shows the distribution of sequential performance as heads move from the OD (Outer Diameter, higher values) to the ID (Inner Diameter, lower values) on four different models of disk drives.

A random seek occurs when the starting sector on a new command is not immediately following the last sector specified in the previous command. Random seeks are common when disk storage is managed by an OS and a file system. This is because the OS manages thousands of data files on a disk drive that are commonly added, deleted, or modified in size while the surrounding files remain in place. This behavior creates pockets of unused space between other pockets of used space. When a new file is created (such as one intended to record real-time data), the OS creates that new file by linking together many of the unused pockets of space to make a single logical file. Thus, the reading or writing of such a file consists of non-sequential (or random) seeks to complete the transfer. File systems also reserve portions of the disk space for file management. For recording areas that span reserved areas, random seeks will be required to skip from one portion of the recording to another. Unlike track-to-track seeks which are an inherent necessity of disk drive technology, random seeks are imposed by software (primarily the OS) and file management systems that are not designed, understandably, to run the disk drive in a physically sequential manner.

Every random seek incurs two different performance penalties. The time for a random seek depends on the distance from its starting point to its ending point. In general, a longer seek takes more time. The **average** (some will be longer and some will be shorter) seek times of the four drives shown in Figure 4 range from 10.9 down to 4.2 milliseconds. The second penalty relates to rotational orientation. Once the heads land on the desired track, the desired starting sector may not be under the head. On the average, the drive will incur a half revolution delay before the starting sector rotates under the head to start the transfer. The spin speeds of the drives in Figure 4 range from 7200 to 15,000 RPM, thus producing an average latency of 4.2 down to 2.0 milliseconds.

However, some instances will incur a full revolution delay while others will incur almost no rotational delay. The wide range of seek times and rotational delays contribute to the non-deterministic nature of OS controlled storage.

The plots of Figure 4 show the rapid decay of instantaneous random-seeking drive performance as the block size (or fragments) becomes smaller. The general block size used by the OS, disk caching, tagged queuing by the disk controller, fragmentation, and the speed with which the disk controller dispatches the next command after the previous one completes all affect where the drive performance operates on these plots. Because fragmentation changes over time, the repeatability of this recording system will also vary over time.



**Figure 4 is a comparison of random-seek drive performance as the burst size in a disk command varies. Plots assume average seek times and average rotational latency but individual instances could be worse. Sequential commands that do not use command queuing or a fast disk controller with disk write caching will have performance more like random seeks than sequential access due to wasted disk revolutions between commands.**

If one has made the decision to live with random access, then faster spinning and faster seeking drives perform better than slower drives. However, a comparison of Figures 3 and 4 clearly shows that the continuous performance of the slowest drive running in sequential mode is much faster than the fastest spinning and seeking drive as it transfers data through a series of small, non-sequential file fragments. Without sufficient unused buffering space, slow disk performance through these fragmented areas could result in buffer overflows which cause data loss and corruption. It is also worth noting that the “slower” drives have a much lower cost/GB and are available in significantly higher storage capacities.

	Hitachi Deskstar 180 GXP	Western Digital Caviar Special Edition 250	Hitachi Ultrastar 146Z10	Hitachi Ultrastar 36Z15
Drive Model	180 GXP	Western Digital Caviar Special Edition 250	Hitachi Ultrastar 146Z10	Hitachi Ultrastar 36Z15
Capacity (GB)	180	250	146	18
Buffer Size (MB)	2	8	8	4
Spin Speed (RPM)	7200	7200	10000	15000
Interface	ATA	ATA	SCSI	SCSI
Cost/GB**	\$1.25	\$1.60	\$6.69	\$14.50
Track-to-track Seek (ms)	0.00110	0.00200	0.00050	0.00065
Average Seek (ms)	0.00880	0.01090	0.00470	0.00380
Avg Latency (ms)	0.00417	0.00417	0.00300	0.00200
OD Data per Track (bytes)*	528,267	609,667	433,550	245,520
ID Data per Track (bytes)*	273,567	310,000	219,700	170,190
Average Bytes per Track*	400,917	459,833	326,625	207,855
* indicates non-published data extrapolated from public information				
** best prices located in a random sampling of online resellers on 4/01/2003				

**Table 1 Parametric information regarding different disk drives.**

### **Different Disk Controllers are Different**

While the raw capabilities of the disk drive make a difference, a disk controller specifically designed for real-time recording also makes a big difference. It is the disk controller that can enforce physical sequentiality if it can operate independently of the OS. When coupled with disk write caching, a good disk controller can dispatch new sequential commands fast enough to continue writing data for the next command without wasting a disk revolution between commands. A good controller can smoothly deliver a cumulative bandwidth that is the sum of the sequential bandwidths of each of the individual drives. Lastly, a disk controller designed for real-time recording can shift the load normally carried by one disk drive to the remaining disk drives in the event that a drive suffers a temporary or permanent failure. This feature, called Dynamic Storage Allocation (DSA), favors getting the customer data onto the magnetic media in the least amount of time rather than waiting for a drive to recover before proceeding. This capability prevents backup of data into the buffers which could lead to an overflow condition.

## A Comparison of Three Real-time Data Recorders

Figure 5 shows a PC with three cards (data acquisition, disk controller, data playback) added to the PCI bus. Table 2 compares three different solutions for moving data from the acquisition card to the disk controller and the disk controller to the data playback card.

Solution A (red arrows) in Figure 5 shows a common approach to moving data from the data acquisition card through system memory to a disk controller. As shown in table 2, solution A is subject to many bottlenecks and variables that reduce performance or make performance unpredictable from computer to computer or from run to run on the same computer. This is largely the result of using shared data paths, using the OS and application software to manage data flow, and using the OS to manage disk storage. It is not uncommon that users of this solution find this solution a tuning and tweaking nightmare. They often find it necessary to re-examine their original requirements after their significant monetary expenditure and finally settle for what the system actually delivers compared to what was expected.

Solution B (light green) creates a point-to-point connection across the PCI bus between the data source and disk controller. Except for bus arbitration cycles and other traffic not eliminated by the user, nearly the entire bandwidth of the PCI bus is available. This solution eliminates all hardware contention at the hubs and system memory. It also functions consistently from computer to computer regardless of processor speed and OS predictability. Furthermore, the specialized disk controller forces data to be written in a physically sequential manner, thus eliminating the random variable introduced by file systems and fragmentation. Lastly, solution B can move a data word from source to disk in a single PCI bus clock cycle whereas solution A requires 2 clock cycles per word, one to and one from system memory. For this reason, solution B can sustain double the data rate as solution A.

Solution C (dark green) has all the benefits of solution B except that it eliminates the variable of PCI bus contention. The disk controller has external data ports (FPDP, Channel Link, and so on) that allow the user to move data into and out of the disk controller through a non-arbitrated and thus unshared data bus. The dedicated data path can, for some users, also eliminate the need for data buffering by running the external bus clock at the inherent speed of the data acquisition and playback hardware.

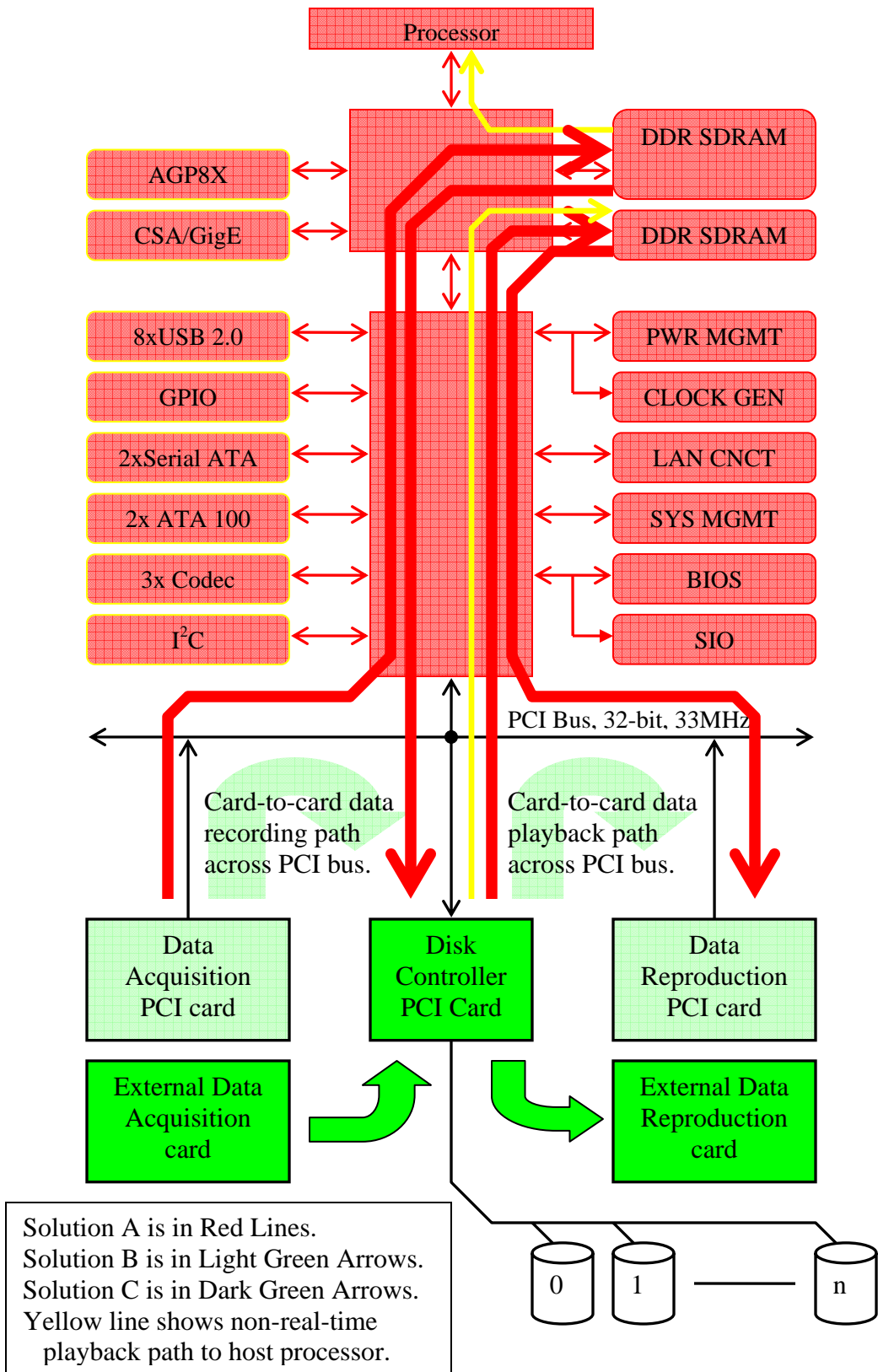


Figure 5 A common computer architecture with different data path possibilities.

Bottle-neck/ Variable		Solution A	Solution B	Solution C
1	Number of bus clock cycles required to move data from data acquisition card to disk storage: PCI bus: Interhub bus: System bus:	2 2 2	1 0 0	0 0 0
2	Data path and disk controller performance independent of variable host CPU, OS, and system memory performance.	NO	YES	YES
3	Maximum disk performance with the fewest number of disks by managing disk storage independent of OS in a physically sequential organization.	NO	YES	YES
4	Performance independent of other PCI bus traffic.	NO	NO	YES
5	Load carried by problem disk drive is dynamically picked up by remaining disk drives. (Dynamic Storage Allocation)	NO	YES	YES
6	Disk Controller Used	SCSI or RAID Controller	Conduant StreamStor	Conduant StreamStor

**Table 2 Comparison different data recording architectures.**

### **Data Playback**

While the initial discussion has been focused on the recording of data, the architecture holds true for real-time data playback. It is often desired to reproduce actual or computer-generated (for simulation purposes) signals in laboratory environments and in other instances such as the projection of digital cinematography, for example. For the same reasons that this architecture eliminates variables during the recording process, it does so for playback as well.

## **Conclusion**

In summary, desktop computers are great tools for a lot of things including data recording and playback for consumer applications. Where data rates get higher or missing data is a concern, understanding the requirements, limitations of the equipment, and what alternatives are available is the key to designing a consistently reliable system.

## **Rights and Trademarks**

StreamStor<sup>™</sup> is a trademark of Conduant Corporation.

## **Revision History**

A. May 22, 2003 - Initial Release.