Computer systems bring together many technologies and harness them to provide fast execution of programs.

Some of these technologies are relatively new, others have been with us for decades.

Each of the system components comes with a detailed specification of their functionality and interface.

The expectation is that system designers can engineer the system based on the component specifications without having to know the details of the implementations of each component.

This is good since many of the underlying technologies change, often in ways that allow the components to become smaller, faster, cheaper, more energy efficient, and so on.

Assuming the new components still implement same interfaces, they can be integrated into the system with very little effort.

The moral of this story is that the important part of the system architecture is the interfaces.

Our goal is to design interface specifications that can survive many generations of technological change.

One approach to long-term survival is to base the specification on a useful abstraction that hides most, if not all, of the low-level implementation details.

Operating systems provide many interfaces that have remained stable for many years.

For example, network interfaces that reliably deliver streams of bytes to named hosts, hiding the details of packets, sockets, error detection and recovery, etc.

Or windowing and graphics systems that render complex images, shielding the application from details about the underlying graphics engine.

Or journaled file systems that behind-the-scenes defend against corruption in the secondary storage arrays.

Basically, we're long since past the point where we can afford to start from scratch each time the integrated circuit gurus are able to double the number of transistors on a chip, or the communication wizards figure out how to go from 1GHz networks to 10GHz networks, or the memory mavens are able to increase the size of main memory by a factor of 4.

The interfaces that insulate us from technological change are critical to ensure that improved technology isn't a constant source of disruption.

There are some famous examples of where an apparently convenient choice of interface has had embarrassing long-term consequences.

For example, back in the days of stand-alone computing, different ISAs made different choices on how to store multi-byte numeric values in main memory.

IBM architectures store the most-significant byte in the lowest address (so called "big endian"), while Intel's x86 architectures store the least-significant byte first (so called "little endian").

But this leads to all sorts of complications in a networked world where numeric data is often transferred from system to system.

This is a prime example of a locally-optimal choice having an unfortunate global impact.

As the phrase goes: "a moment of convenience, a lifetime of regret." Another example is the system-level communication strategy chosen for the first IBM PC, the original personal computer based Intel CPU chips.

IBM built their expansion bus for adding I/O peripherals, memory cards, etc., by simply using the interface signals provided by then-current x86 CPU.

So the width of the data bus, the number of address pins, the data-transfer protocol, etc. where are exactly as designed for interfacing to that particular CPU.

A logical choice since it got the job done while keeping costs as low a possible.

But that choice quickly proved unfortunate as newer, higher-performance CPUs were introduced, capable of addressing more memory or providing 32-bit instead of 16-bit external data paths.

So system architects were forced into offering customers the Hobson's choice of crippling system throughput for the sake of backward compatibility, or discarding the networking card they bought last year since it was now incompatible with this year's system.

But there are success stories too.

The System/360 interfaces chosen by IBM in the early 1960s carried over to the System/370 in the 70's and 80's and to the Enterprise System Architecture/390 of the 90's.

Customers had the expectation that software written for the earlier machines would continue to work on the newer systems and IBM was able to fulfill that expectation.

Maybe the most notable long-term interface success is the design the TCP and IP network protocols in the early 70's, which formed the basis for most packet-based network communication.

A recent refresh expanded the network addresses from 32 to 128 bits, but that was largely invisible to applications using the network.

It was a remarkably prescient set of engineering choices that stood the test of time for over four decades of exponential growth in network connectivity.

Today's lecture topic is figuring out the appropriate interface choices for interconnecting system components.

In the earliest systems these connections were very ad hoc in the sense that the protocols and physical implementation were chosen independently for each connection that had to be made.

The cable connecting the CPU box to the memory box (yes, in those days, they lived in separate 19" racks!) was different than the cable connecting the CPU to the disk.

Improving circuit technologies allowed system components to shrink from cabinet-size to board-size and system engineers designed a modular packaging scheme that allowed users to mix-and-match board types that plugged into a communication backplane.

The protocols and signals on the backplane reflected the different choices made by each vendor - IBM boards didn't plug into Digital Equipment backplanes, and vice versa.

This evolved into some standardized communication backplanes that allowed users to do their own system integration, choosing different vendors for their CPU, memory, networking, etc.

Healthy competition quickly brought prices down and drove innovation.

However this promising development was overtaken by rapidly improving performance, which required communication bandwidths that simply could not be supported across a multi-board backplane.

These demands for higher performance and the ability to integrate many different communication channels into a single chip, lead to a proliferation of different channels.

In many ways, the system architecture was reminiscent of the original systems - ad-hoc purpose-built communication channels specialized to a specific task.

As we'll see, engineering considerations have led to the widespread adoption of general-purpose unidirectional point-to-point communication channels.

There are still several types of channels depending on the required performance, the distance travelled, etc., but asynchronous point-to-point channels have mostly replaced the synchronous multi-signal channels of earlier systems.

Most system-level communications involve signaling over wires, so next we'll look into some the engineering issues we've had to deal with as communication speeds have increased from kHz to GHz.