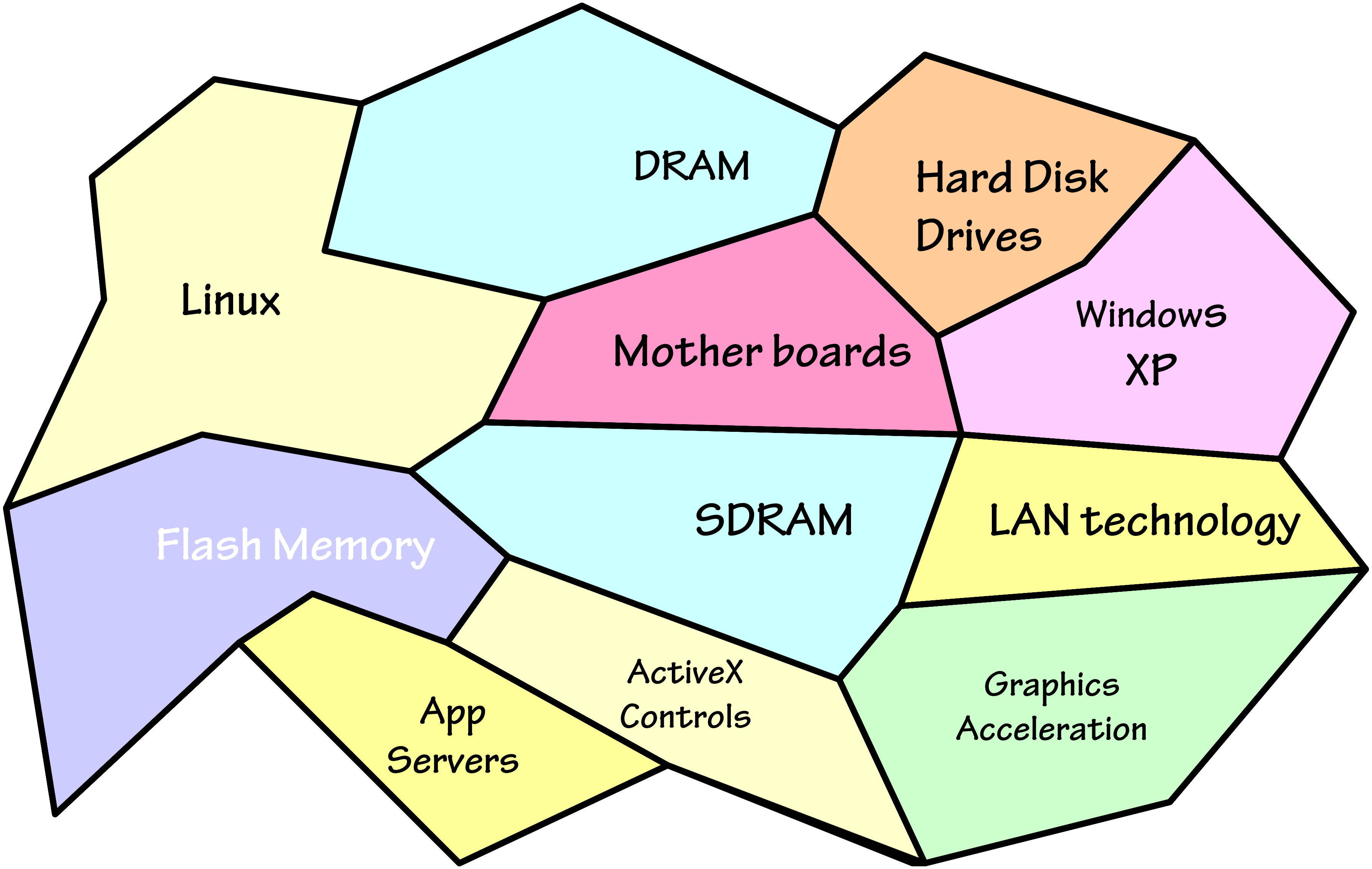
Interconnect & Communication

Space, Time, & stuff…

Lab 8 due Tomorrow (Wednesday)!

Computer System Technologies

What's the most important part of this picture?



DRAM

Hard Disk Drives

Linux

Mother boards

Windows XP

Flash Memory

SDRAM

LAN technology

ActiveX Controls

App Servers

Graphics Acceleration

Technology comes & goes; interfaces last forever

Interfaces typically *deserve* more engineering attention than the technologies they interface…

- **Abstraction:** should outlast many technology generations
- **Often "virtualized"** to extend beyond original function (e.g. memory, I/O, services, machines)
- **Represent more potential value** to their proprietors than the technologies they connect.

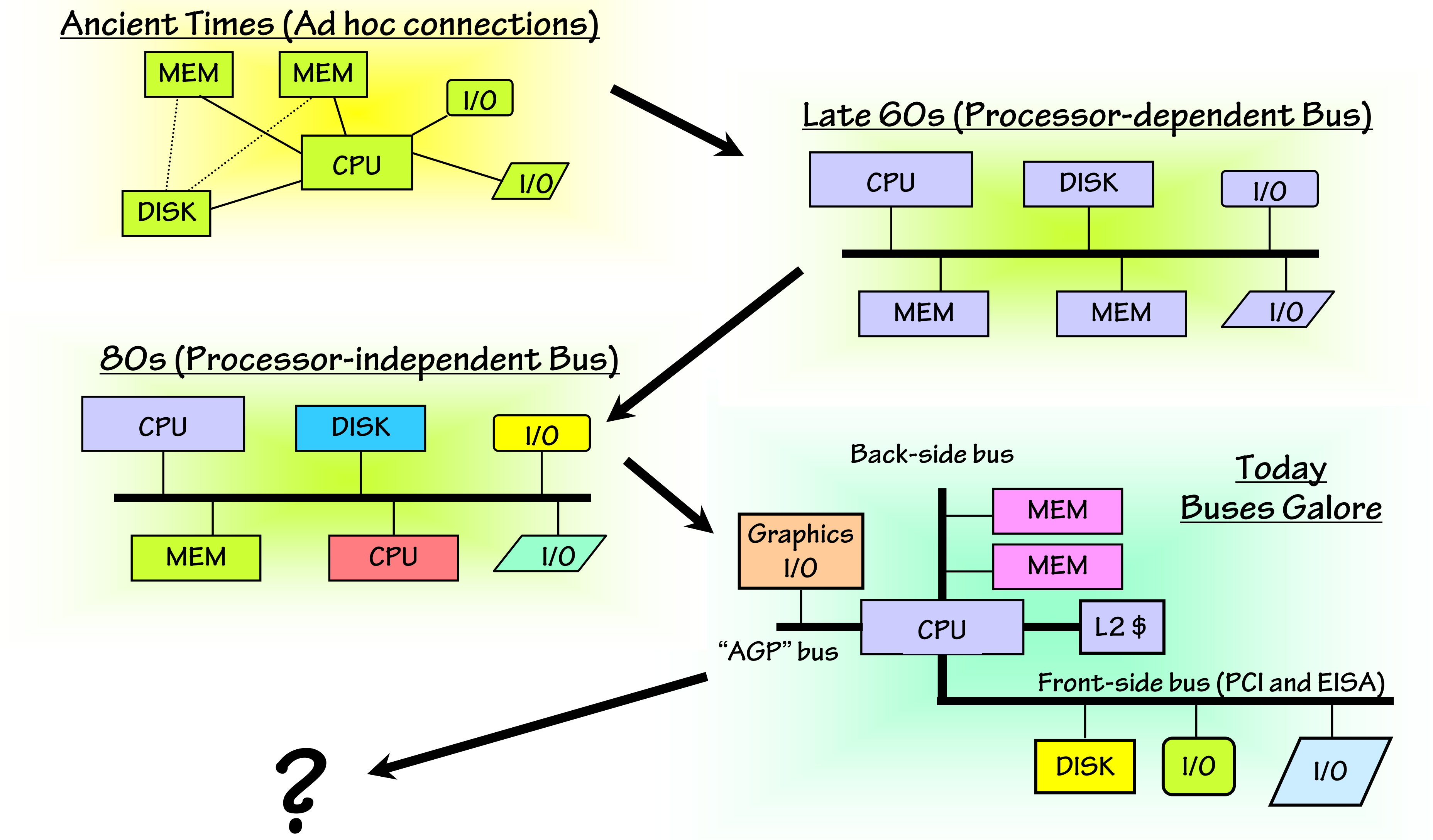
Interface sob stories:

- **Interface "warts":** Windows "aux.c" bug, Big/little Endian wars
- **IBM PC *debacle***

... and many success stories:

- **IBM 360** Instruction set architecture; Postscript; Compact Flash; ...
- **Backplane buses**

System Interfaces & Modularity



Ancient Times (Ad hoc connections)

Late 60s (Processor-dependent Bus)

80s (Processor-independent Bus)

Today
Buses Galore

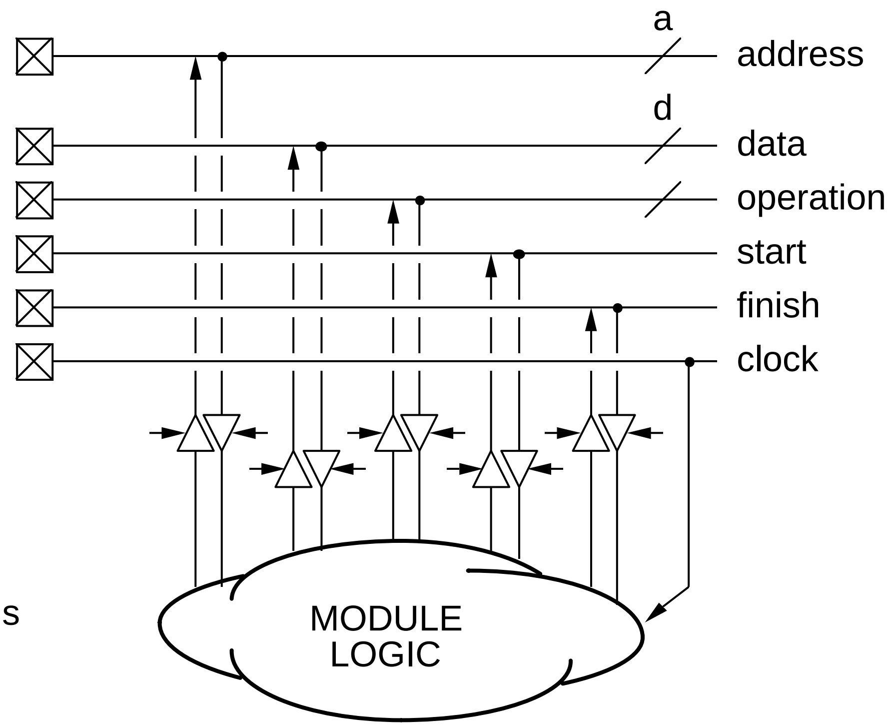
Interface Standard: Backplane Bus

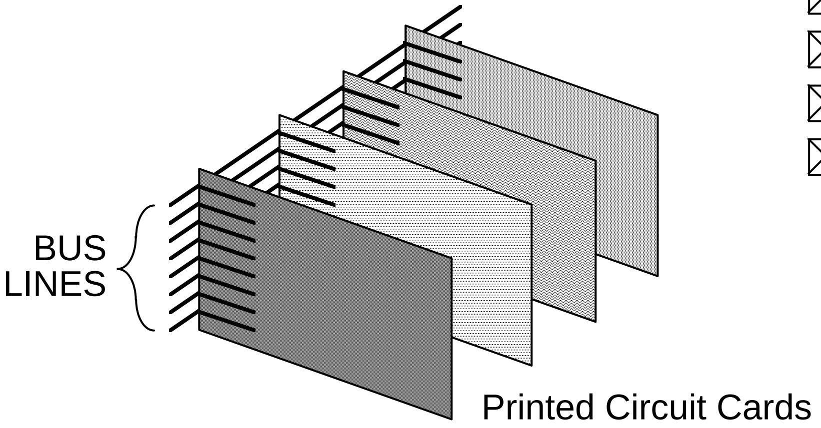
Modular cards that plug into a common backplane:

- CPUs
- Memories
- Bulk storage
- I/O devices
- S/W?

The backplane provides:

- Power
- Common system clock
- Wires for communication



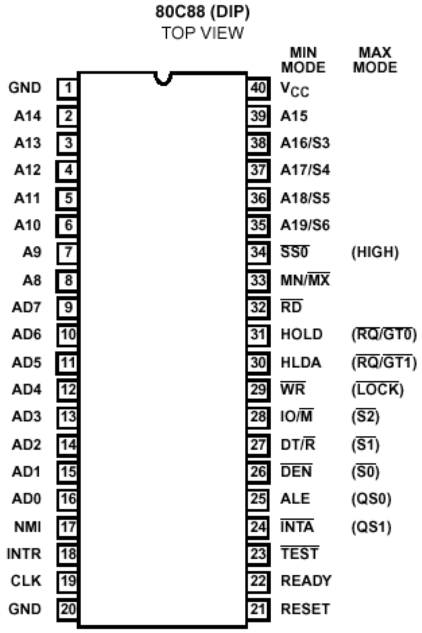


The Dumb Bus: ISA & EISA

Original primitive approach --

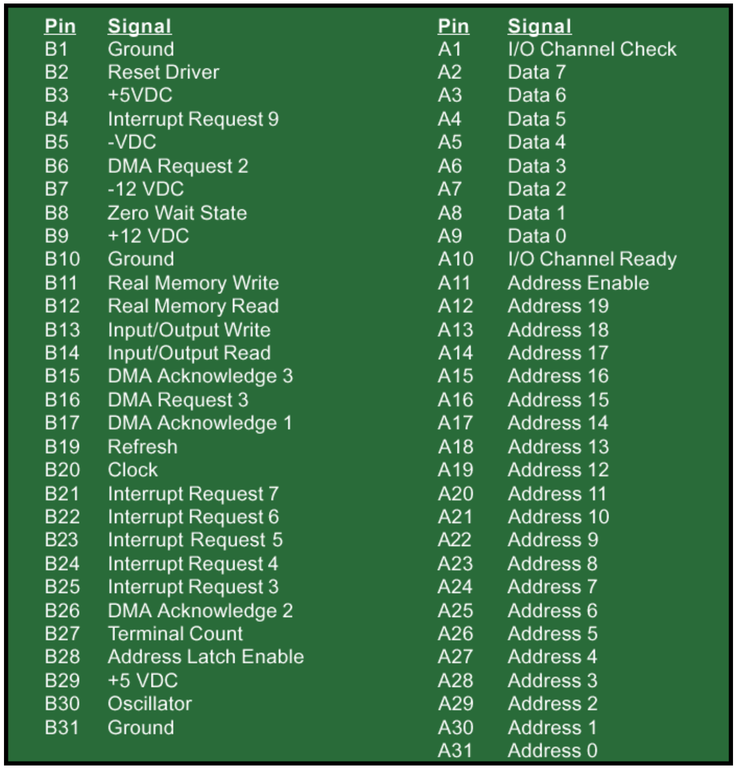
Just take the control signals and data bus from the CPU module, buffer it, and call it a bus.

Ah, you forget, Unibus, S-100, SWTP SS-50, STB, MultiBus, Apple 2E, …



ISA bus (Original IBM PC bus) -

Pin out and timing is nearly identical to the 8088 spec.



Pin	Signal	Pin	Signal
B1	Ground	A1	I/O Channel Check
B2	Reset Driver	A2	Data 7
B3	+5VDC	A3	Data 6
B4	Interrupt Request 9	A4	Data 5
B5	-VDC	A5	Data 4
B6	DMA Request 2	A6	Data 3
B7	-12 VDC	A7	Data 2
B8	Zero Wait State	A8	Data 1
B9	+12 VDC	A9	Data 0
B10	Ground	A10	I/O Channel Ready
B11	Real Memory Write	A11	Address Enable
B12	Real Memory Read	A12	Address 19
B13	Input/Output Write	A13	Address 18
B14	Input/Output Read	A14	Address 17
B15	DMA Acknowledge 3	A15	Address 16
B16	DMA Request 3	A16	Address 15
B17	DMA Acknowledge 1	A17	Address 14
B19	Refresh	A18	Address 13
B20	Clock	A19	Address 12
B21	Interrupt Request 7	A20	Address 11
B22	Interrupt Request 6	A21	Address 10
B23	Interrupt Request 5	A22	Address 9
B24	Interrupt Request 4	A23	Address 8
B25	Interrupt Request 3	A24	Address 7
B26	DMA Acknowledge 2	A25	Address 6
B27	Terminal Count	A26	Address 5
B28	Address Latch Enable	A27	Address 4
B29	+5 VDC	A28	Address 3
B30	Oscillator	A29	Address 2
B31	Ground	A30	Address 1
		A31	Address 0

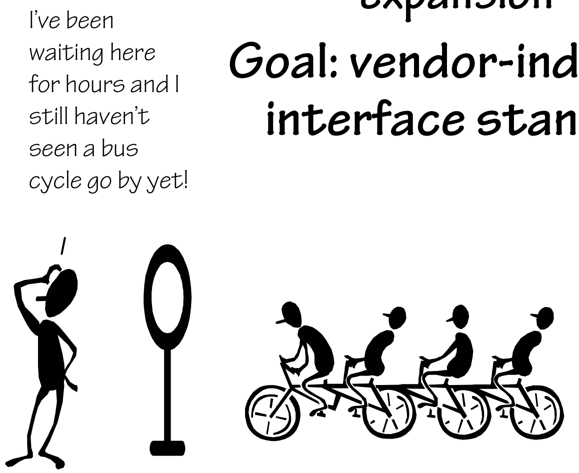
Smarter "Processor Independent" Buses

NuBus, PCI…

Isolate basic communication primitives from processor architecture:

- Simple read/write protocols
- Symmetric: any module can become "Master" (smart I/O, multiple processors, etc)
- Support for "plug & play" expansion

Goal: vendor-independent interface standard



TERMINOLOGY –

BUS MASTER – a module that initiates a bus transaction. (CPU, disk controller, etc.)

BUS SLAVE – a module that responds to a bus request. (Memory, I/O device, etc.)

BUS CYCLE – The period from when a transaction is requested until it is served.

Buses, Interconnect…
what's the big deal?

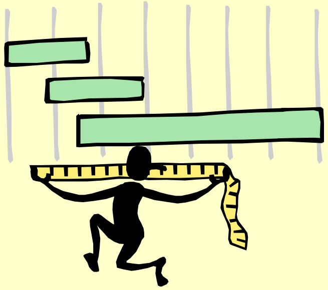
Aren't buses simply logic circuits with long wires?

Wires: circuit theorist's view:

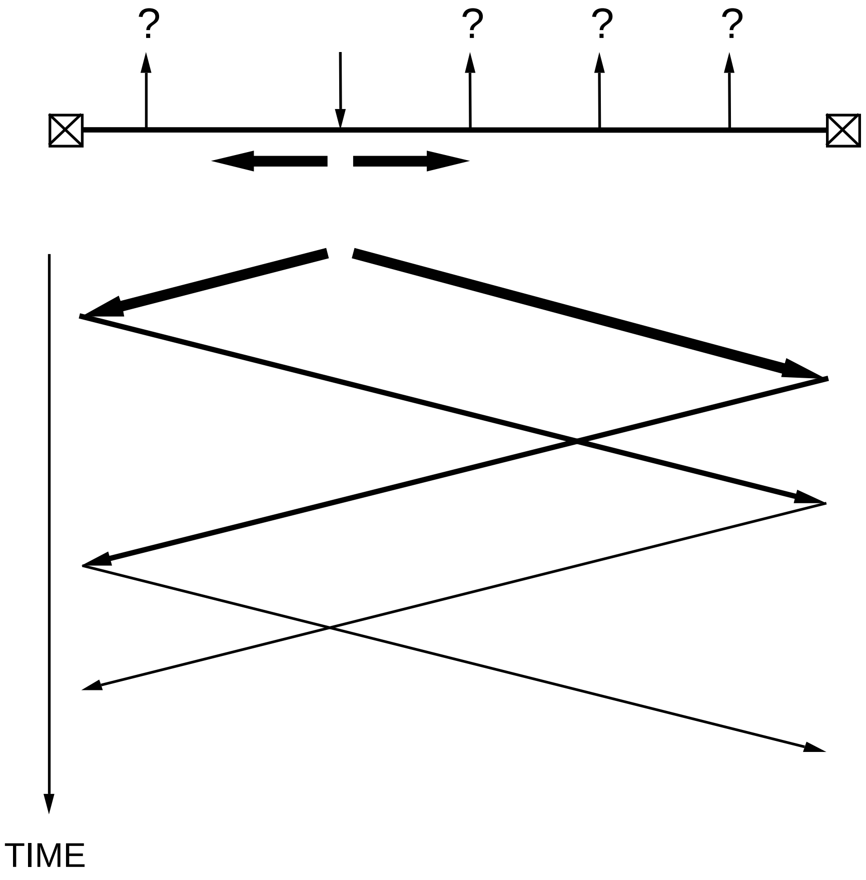
- Equipotential "nodes" of a circuit.
- Instant propagation of v, i over entire node.
- "space" abstracted out of design model.
- Time issues dictated by RLC elements; wires are timeless.

Wires: interconnect engineer's view:

- Transmission lines.
- Finite signal propagation velocity.
- Space matters.
- Time matters.
- Reality matters.



Bus Lines as Transmission Lines



ANALOG ISSUES:

- Propagation times
  - Light travels about 1 ft / ns (about 7”/ns in a wire)
- Skew
  - Different points along the bus see the signals at different times
- Reflections & standing waves
  - At each interface (places where the propagation medium changes) the signal may reflect if the impedances are not matched.
  - Make a transition on a long line – may have to wait many transition times for echos to subside.

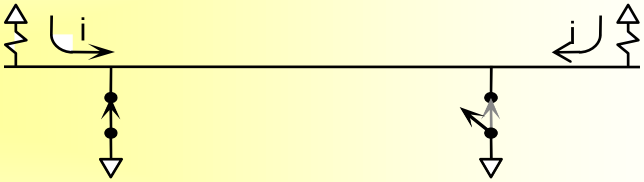
Coping with Analog Issues...

We'd like our bus to be *technology independent*...

- *Self-timed* protocols allow bus transactions to accommodate varying response times;
- *Asynchronous* protocols avoid the need to pick a (technology-dependent) clock frequency.

BUT... asynchronous protocols are vulnerable to analog-domain problems, like the infamous

WIRED-OR GLITCH: what happens when a switch is opened???

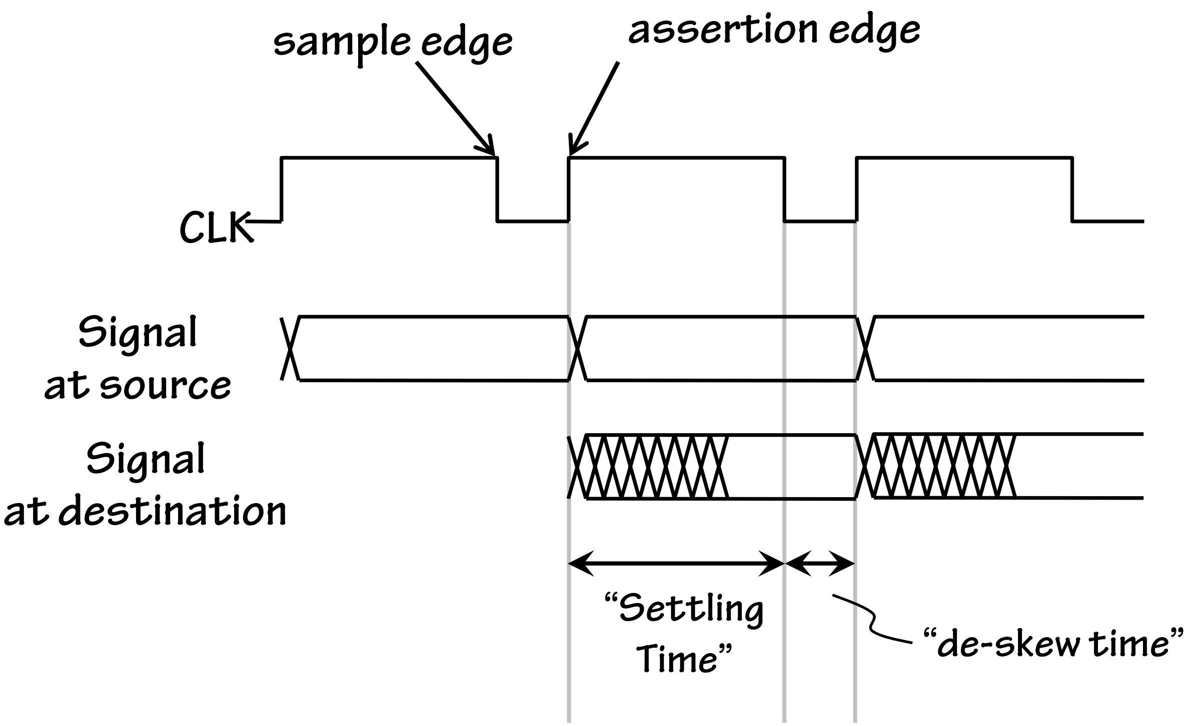


COMMON COMPROMISE: Synchronous, Self-Timed protocols

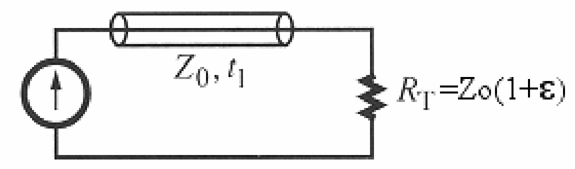
- Broadcast bus clock
- Signals sampled at "safe" times

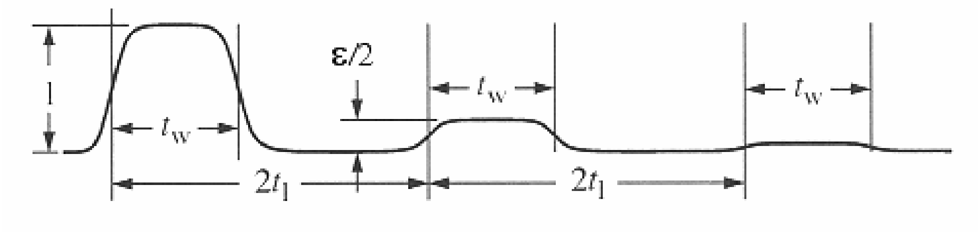
* DEAL WITH: noise, clock skew (wrt signals)

Synchronous Bus Clock Timing

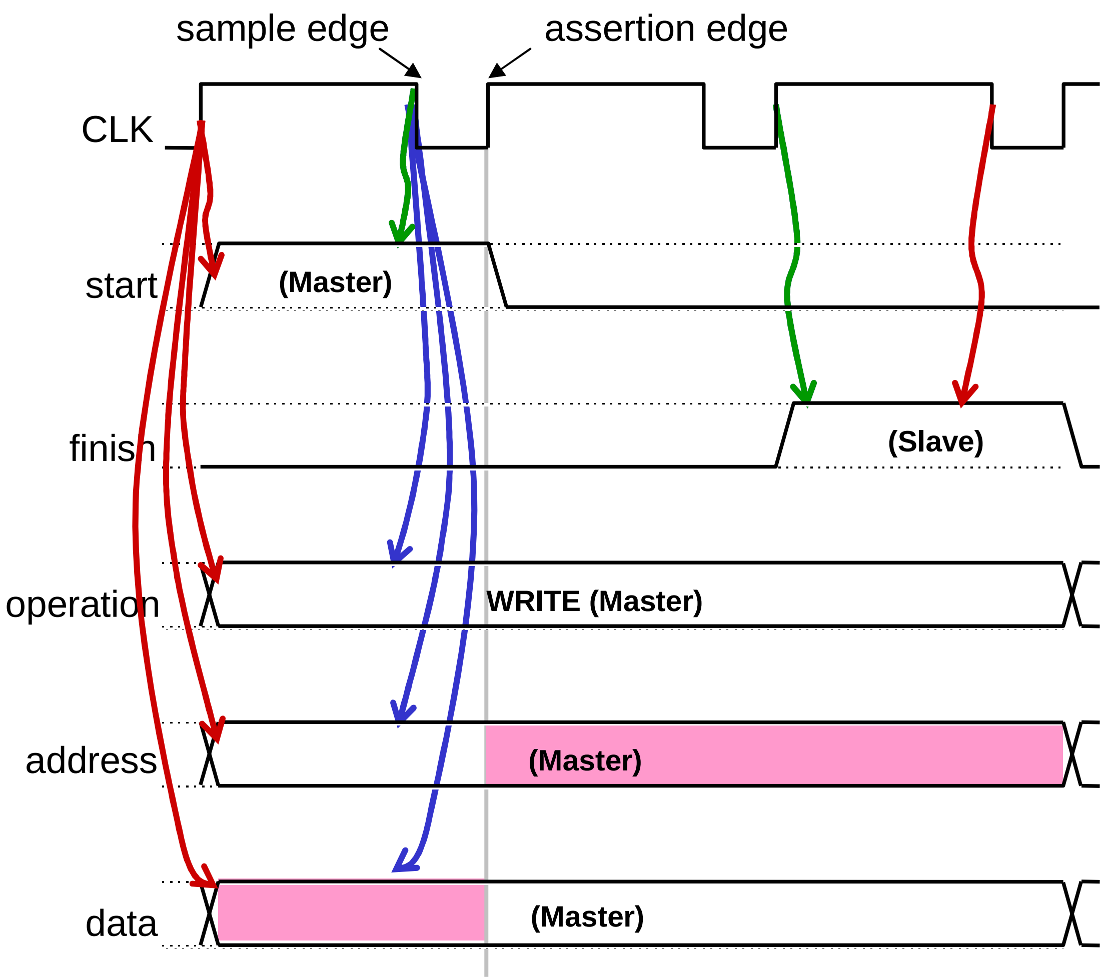


Allow for several "round-trip" bus delays so that ringing can die down.





A Simple Bus Transaction



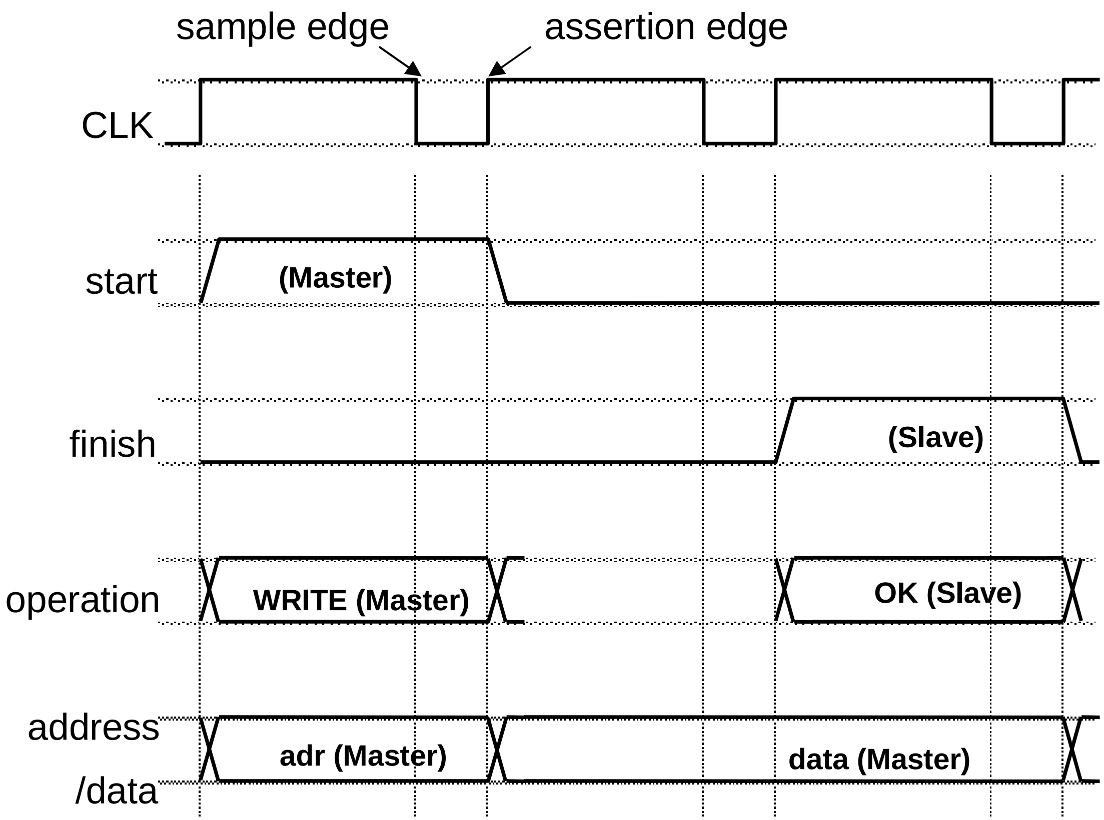
MASTER:
1) Chooses bus operation
2) Asserts an address
3) Waits for a slave to answer.

SLAVE:
1) Monitors start
2) Check address
3) If meant for me
    a) look at bus operation
    b) do operation
    c) signal finish of cycle

BUS:
1) Monitors start
2) Start count down
3) If no one answers before counter reaches 0 then "time out"

Multiplexed Bus: Write Transaction

More efficient use of shared wires

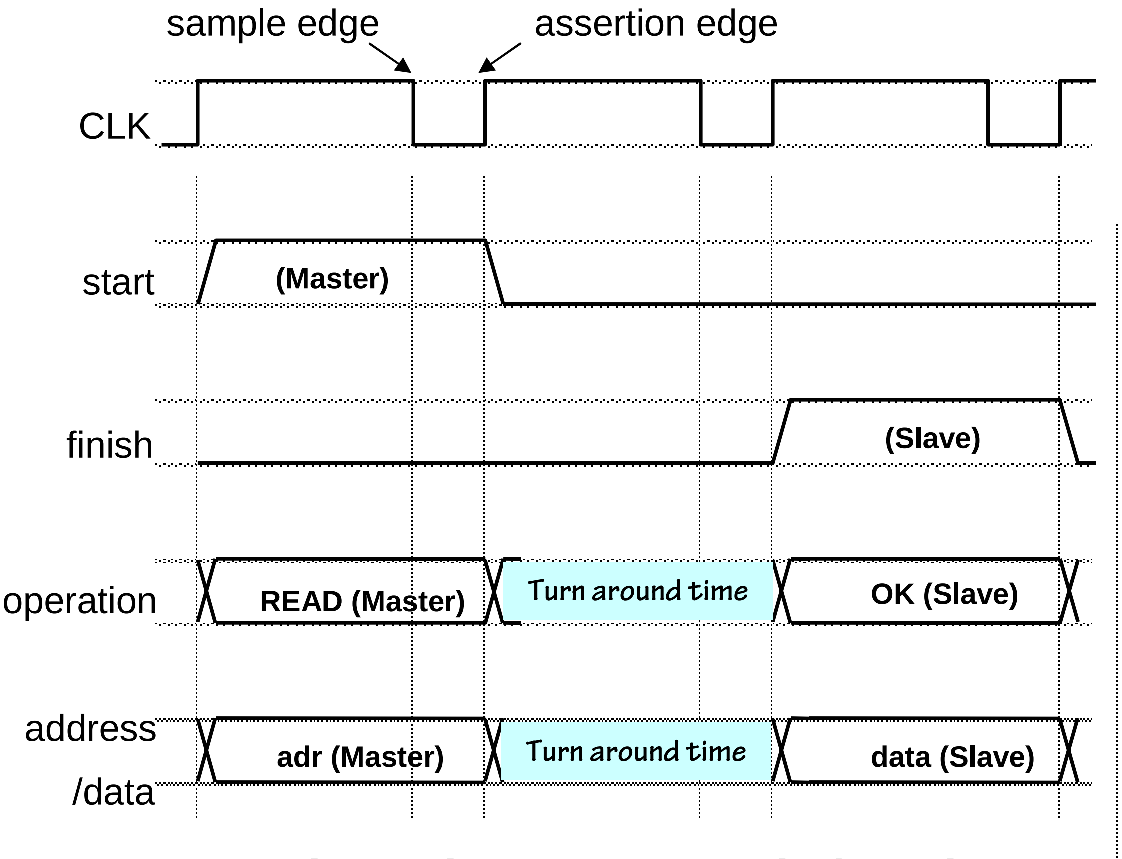


We let the address and data buses share the same wires.

Slave sends a status message by driving the operation control signals when it finishes. Possible indications:
- request succeeded
- request failed
- try again

A slave can stall the write by waiting several cycles before asserting the finish signal.

Multiplexed Bus: Read Transaction

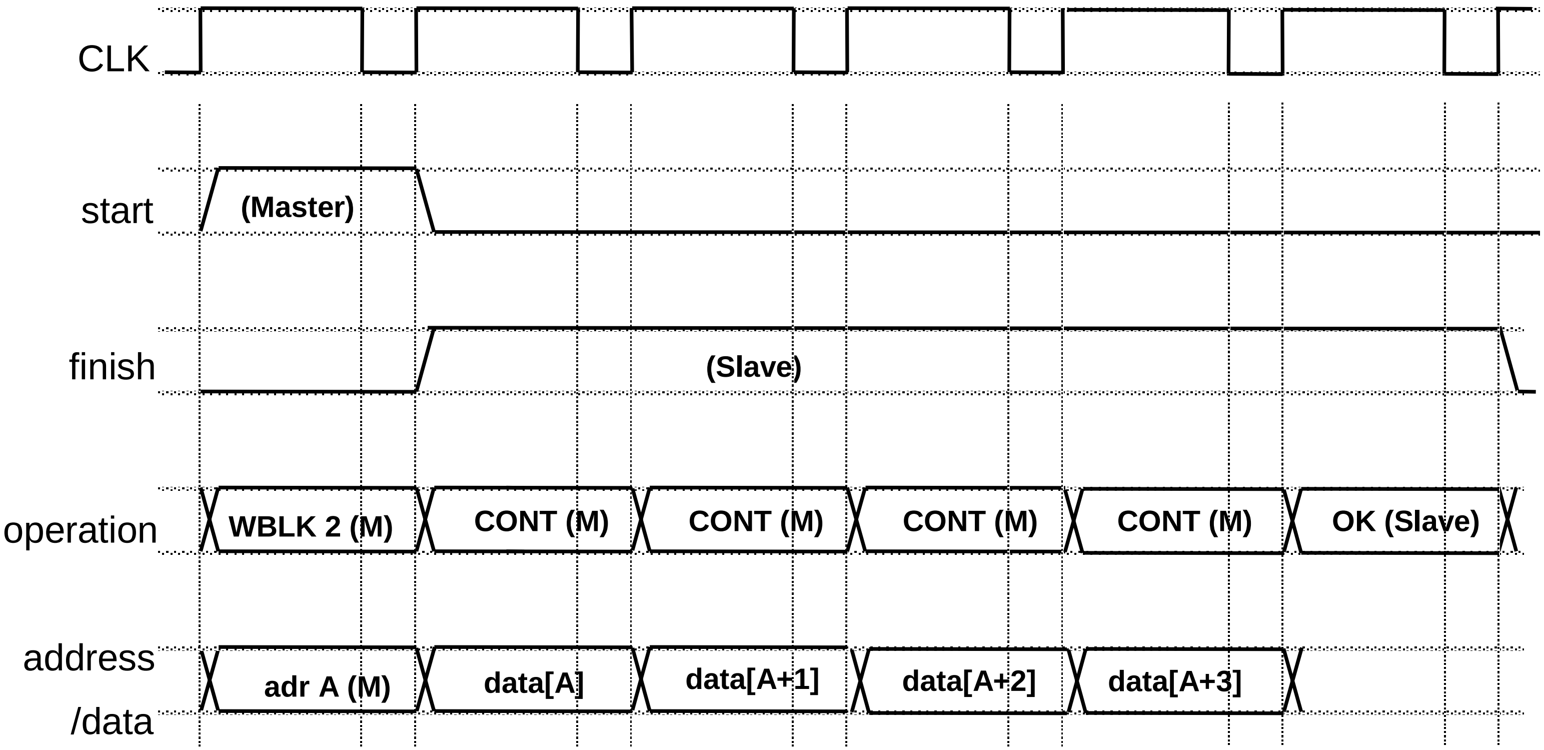


Throughput: 3+ Clocks/word

On reads, we allot one cycle for the bus to "turn around" (stop driving and begin receiving). It generally takes some time to read data anyway.

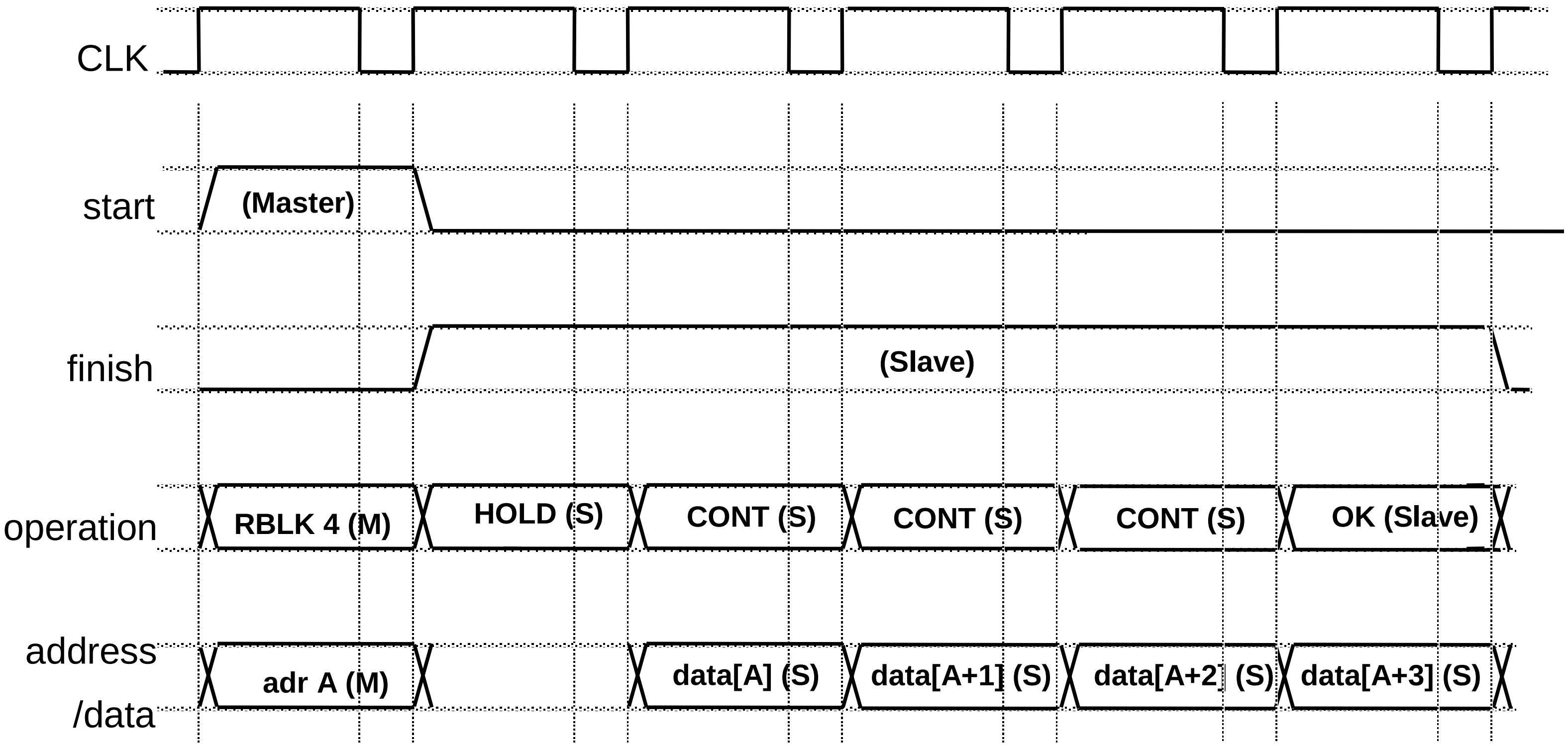
A slave can stall the read (for instance if the device is slow compared to the bus clock) by waiting several clocks before asserting the finish signal. These delays are sometimes called "WAIT-STATES"

Block Write Transfers



Block transfers are the way to get peak performance from a bus. A throughput of nearly 1 Clock/word is achievable on large blocks. Slaves must generate sequential addresses.

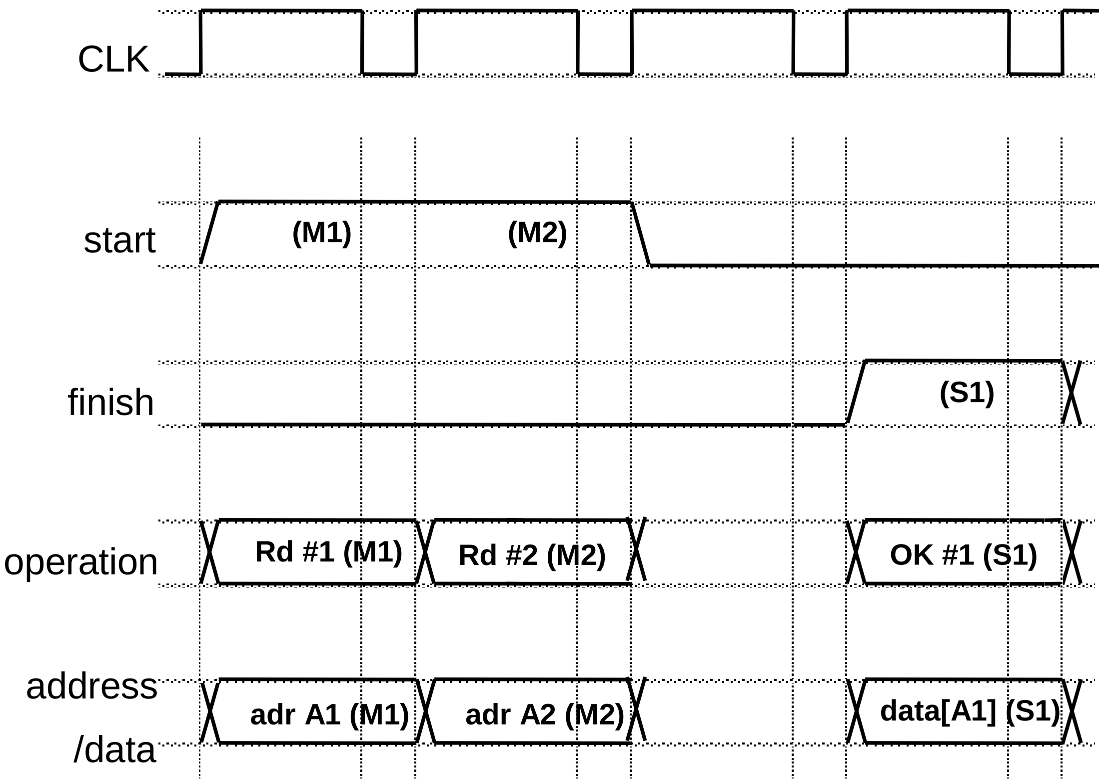
Block Read Transfers



Block read transfers still require at least one cycle to turn-around the bus. More WAIT-STATES can be added if initial latency is high. The throughput is nearly 1 Clock/word on large blocks. Great for reading long cache lines!

Split-Transaction Bus Operation

… you knew we'd work pipelining in somehow!



The bus master can post several read requests before the first request is served.

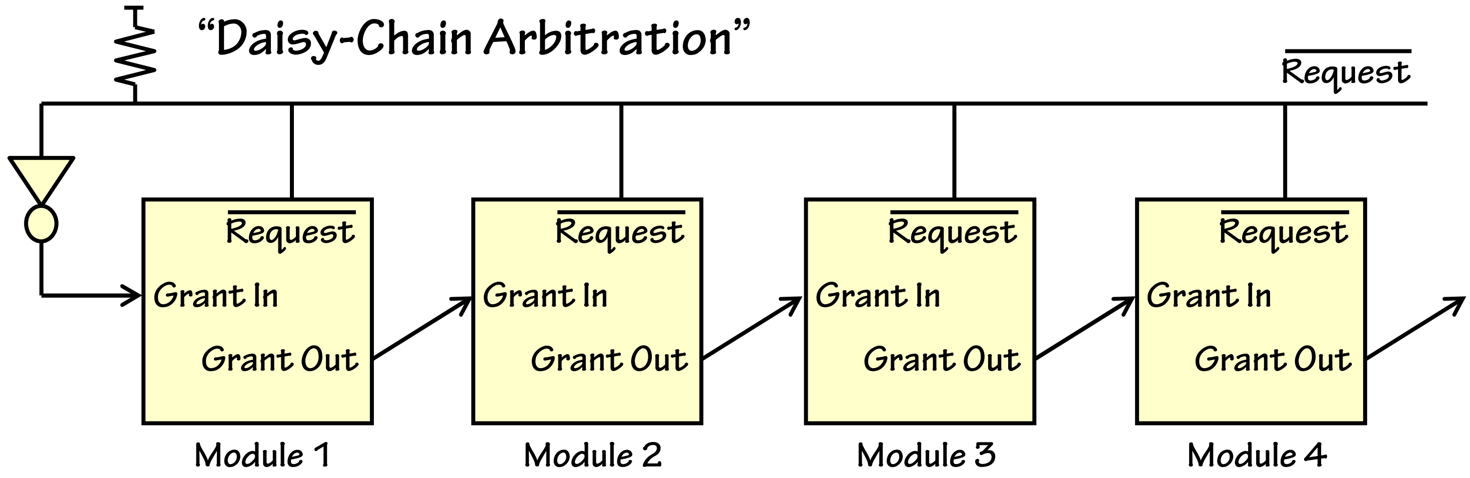
Generally, accesses are served in the same order that they are requested.

Slaves must queue up multiple requests, until master releases bus.

The master must keep track of outstanding requests and their status.

Throughput: 2 Clocks/word, *independent of read latency*

Bus Arbitration: Multiple Bus Masters



ISSUES:

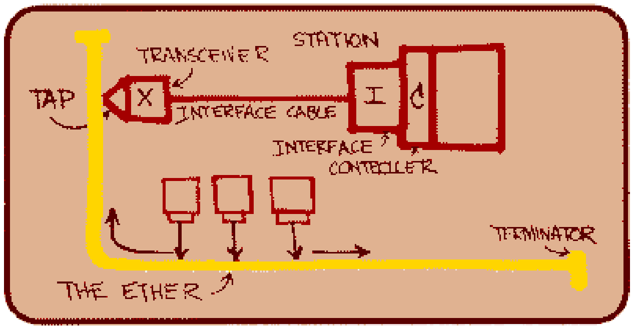
- Fairness - Given uniform requests, bus cycles should be divided evenly among modules (to each, according to their needs…)
- Bounded Wait – An upper bound on how long a module has to wait between requesting and receiving a grant
- Utilization - Arbitration scheme should allow for maximum bus performance
- Scalability - Fixed-cost per module (both in terms of arbitration H/W and arbitration time.

STATE OF THE ART ARBITRATION: N masters, log N time, log N wires.

Outside the box…

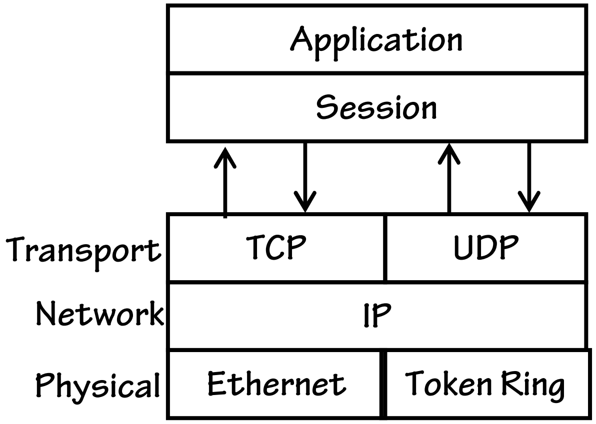
The Network as an interface standard

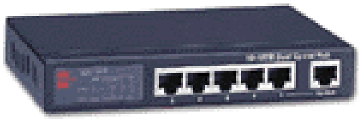
ETHERNET: In the mid-70's Bob Metcalf (at Xerox PARC, an MIT alum) devised a bus for networking computers together.



- Bit-serial (optimized for long wires)
- Asynchronous (no clock distribution)
- Variable-length "packets"

EMERGING IDEA: Protocol "stacks" that isolate application-level interface from low-level physical devices:





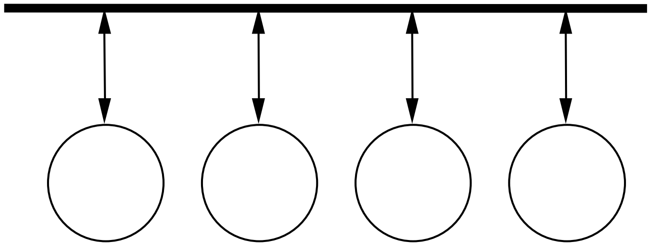
Generalizing Buses…

Communication Topologies

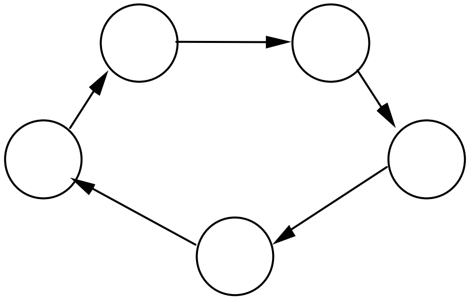
1-dimensional approaches:

"Low cost networks" – constant cost/node

BUS



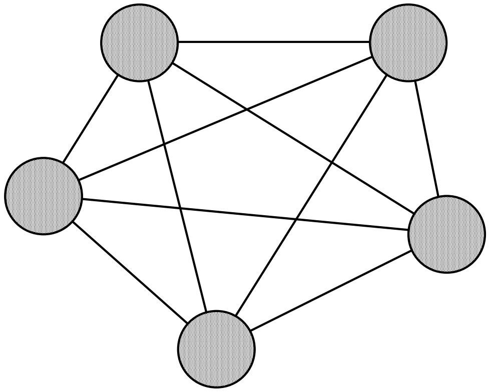
ONE step for random message delivery (but only one message at a time)



RING

$\Theta(n)$ steps for random message delivery

Quadratic-cost Topologies

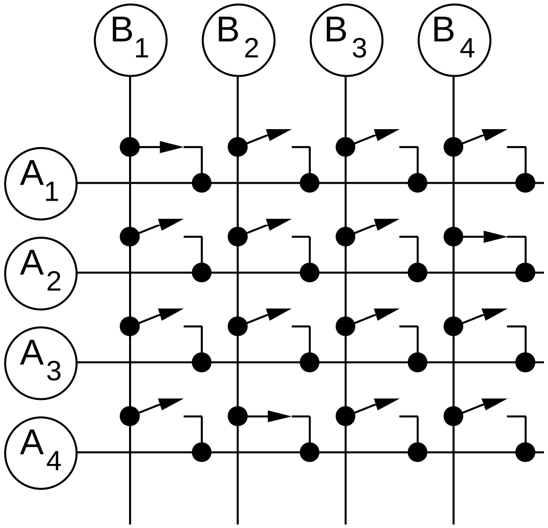


COMPLETE GRAPH:

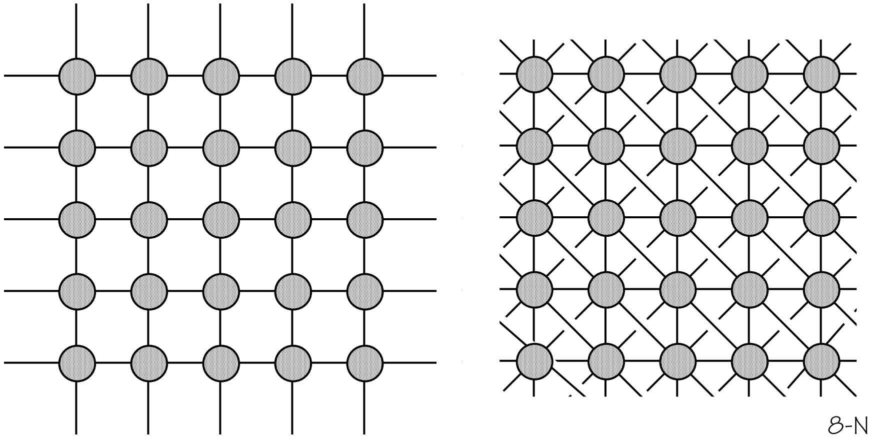
Dedicated lines connecting each pair of communicating nodes. $\Theta(n)$ simultaneous communications.

CROSSBAR SWITCH:

- Switch dedicated between each pair of nodes
- Each A_i can be connected to one B_j at any time
- Special cases:
  - A = processors, B = memories
  - A, B same type of node
  - A, B same nodes (complete graph)



Mesh Topologies



4-Neighbor

8-Neighbor

2-Dimensional Meshes

$\Theta(n)$	Thruput
$\Theta(\sqrt{n})$	Latency
$\Theta(n)$	Cost

Nearest-neighbor connectivity:

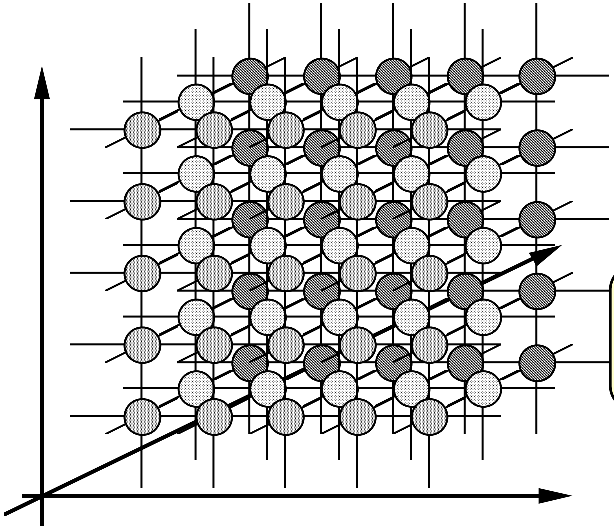
Point-to-point interconnect

- minimizes delays

- minimizes “analog” effects

Store-and-forward

(some overhead associated with communication routing)



3-D, 6-Neighbor Mesh

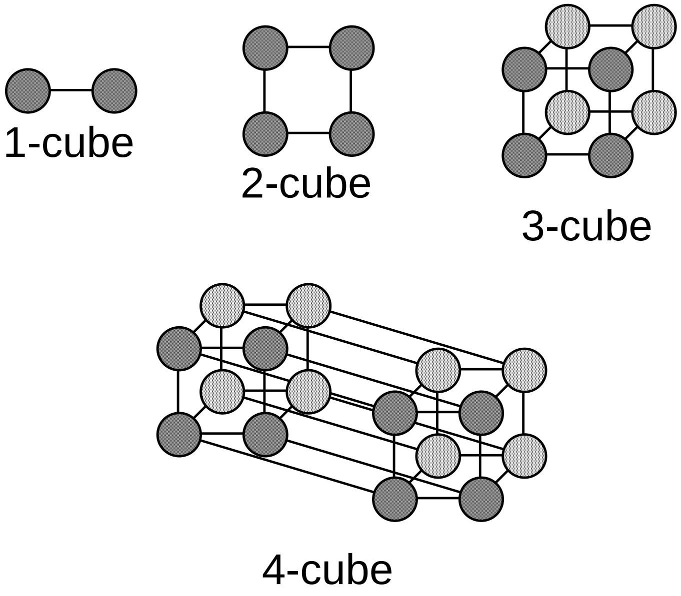
$\Theta(n)$	Thruput
$\Theta(\sqrt[3]{n})$	Latency
$\Theta(n)$	Cost

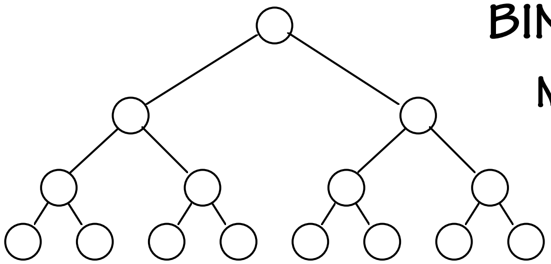
Logarithmic Latency Networks

HYPERCUBE (n-cube):

Cost = $\Theta(n \log n)$

Worst-case path length = $\Theta(\log n)$





BINARY TREE:

Maximum path length is $\Theta(\log n)$ steps;

Cost/node constant.

Communication Topologies: Latency

Theorist's view:

- Each point-to-point link requires one hardware unit.
- Each point-to-point communication requires one time unit.

Topology	$	Theoretical Latency	Actual Latency
Complete Graph	$\Theta(n^2)$	~~$\Theta(1)$~~ →	$\geq \Theta(\sqrt[3]{n})$
Crossbar	$\Theta(n^2)$	~~$\Theta(1)$~~ →	$\Theta(n)$
1D Bus	$\Theta(n)$	~~$\Theta(1)$~~ →	$\Theta(n)$
2D Mesh	$\Theta(n)$	$\Theta(\sqrt{n})$	
3D Mesh	$\Theta(n)$	$\Theta(\sqrt[3]{n})$	
Tree	$\Theta(n)$	~~$\Theta(\log n)$~~ →	$\geq \Theta(\sqrt[3]{n})$
N-cube	$\Theta(n \log n)$	~~$\Theta(\log n)$~~ →	$\geq \Theta(\sqrt[3]{n})$

IS IT REAL?

- Speed of Light: ~ 1 ns/foot (typical bus propagation: 5 ns/foot)
- Density limits: can a node shrink forever? How about Power, Heat, etc … ?

OBSERVATION: Links on Tree, N-cube must grow with n; hence time/link must grow.

Communications Futures

Backplane Buses – *standard for peripherals*

+ easy hardware configurability
+ vendor-independent standards
- serialized communications
- bottleneck as systems scale up

FireWire ISA EISA RDRAM NuBus PCI SDRAM SBUS USB

Specialized buses for memory, graphics, …

New-generation communications…

- **Log networks (trees, hypercubes, …)**
- **2D Meshes (IWARP, ...)**
- **3D Meshes …**
  - **4-neighbor, 3D mesh (NuMesh Diamond lattice)**

Space: *the final frontier?*