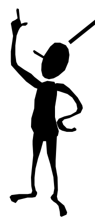
Interconnect & Communication

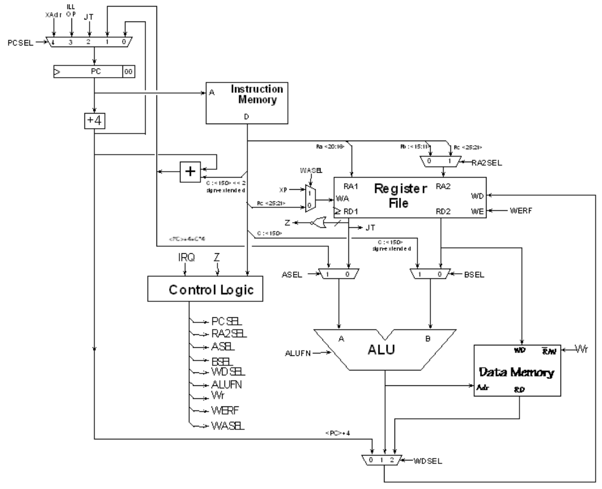
What is the big deal with these things?
Cool Bus
WARD & HALSTEAD
6.004 NERD KIT
I don't see what is so exciting about the "back-side" either.

Handouts: Lecture Slides

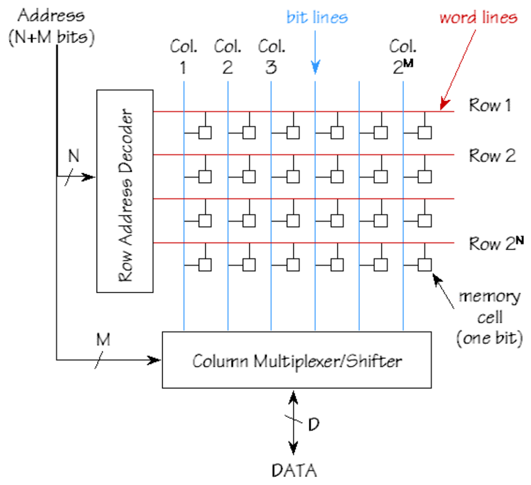

Don't forget the QUIZ tonight in Walker Gym (50-340). You can attend either of 2 sessions, 5:30-7:30 or 7:30-9:30. More later.

Reminder: It's all About Information

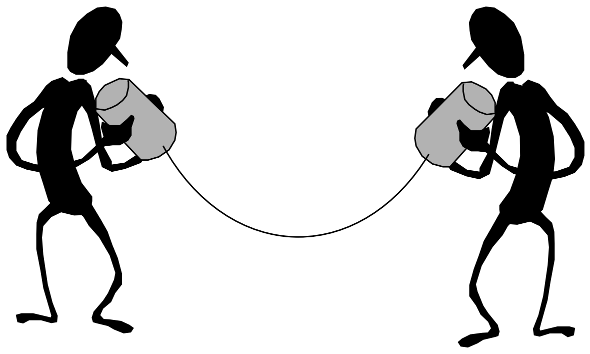
Transforming it - Computation



Storing it - Memory



Transporting it - Communication



Goal #1: Modularity

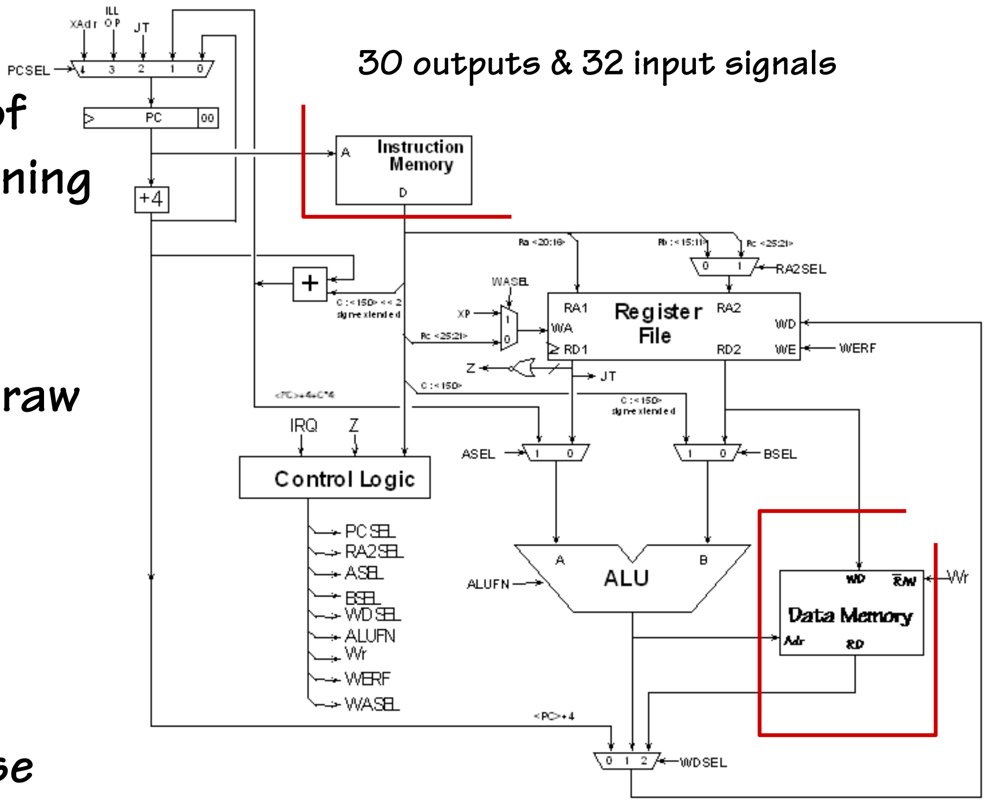
We can't fit everything onto a single chip (yet).

One of the major challenges of computer architecture is defining intermodule interfaces.

The tricky bit is… where to draw the lines?

- Minimize I/O
- Minimize cost
- Maximize expandability
- general vs. special-purpose
- shared vs. point-to-point

30 outputs & 32 input signals

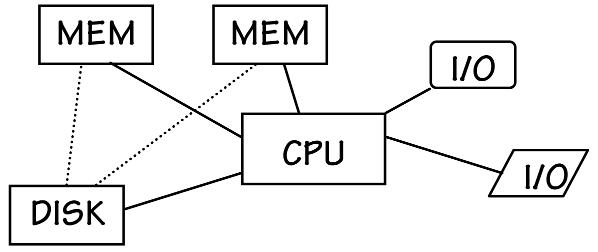


30 + 32 outputs,
32 input signals,
& 1 control signal

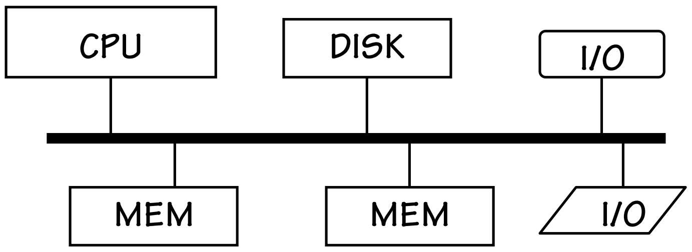
Goal #2: Expansion

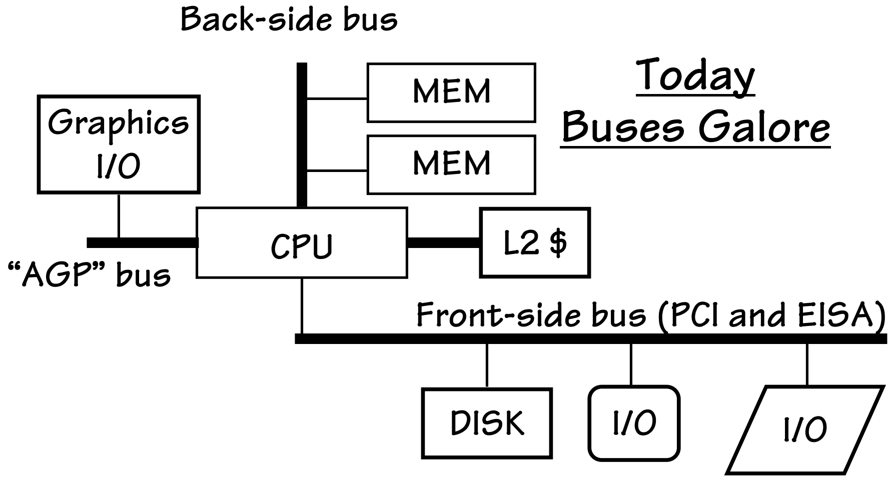
How do we add new, processors, memory, and I/O devices to our system?

Ancient Times (Ad hoc expansion)



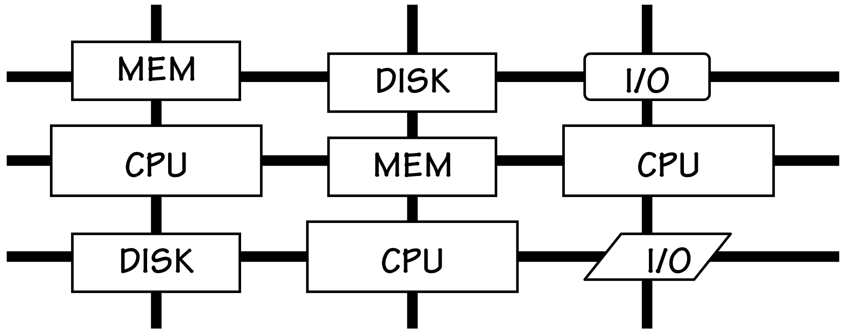
Yesterday (Centralized Bus)





Today
Buses Galore

Tomorrow



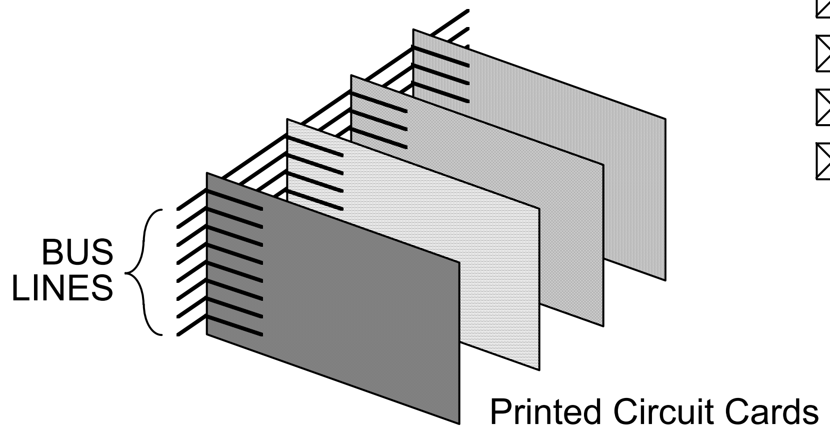
Realization: Backplane Bus

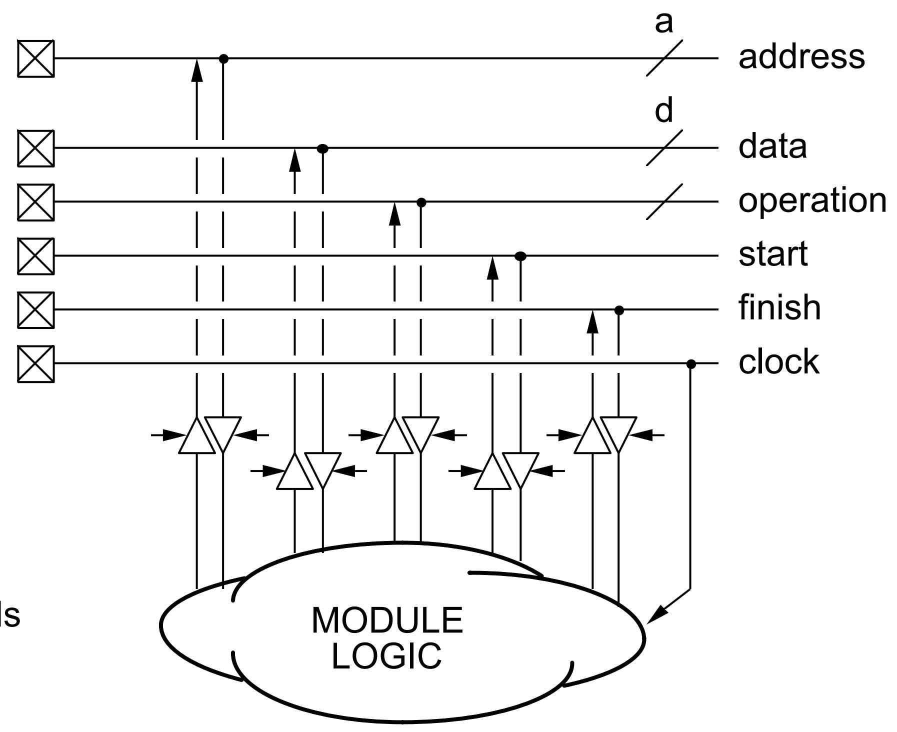
Modular cards that plug into a common backplane:

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

The backplane provides:

- Power
- Common system clock
- Wires for communication





Dawn of the Dumb Bus: ISA & EISA

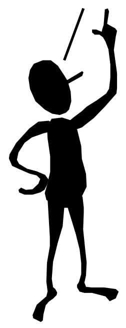
Philosophy (or lack thereof)-

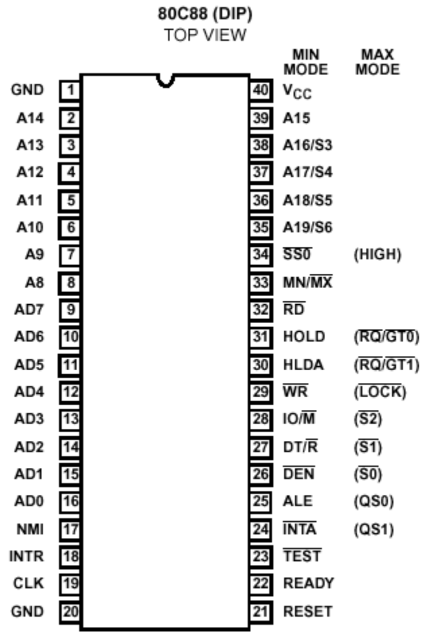
Just take the control signals and data bus from the CPU module, buffer it, and call it a bus.

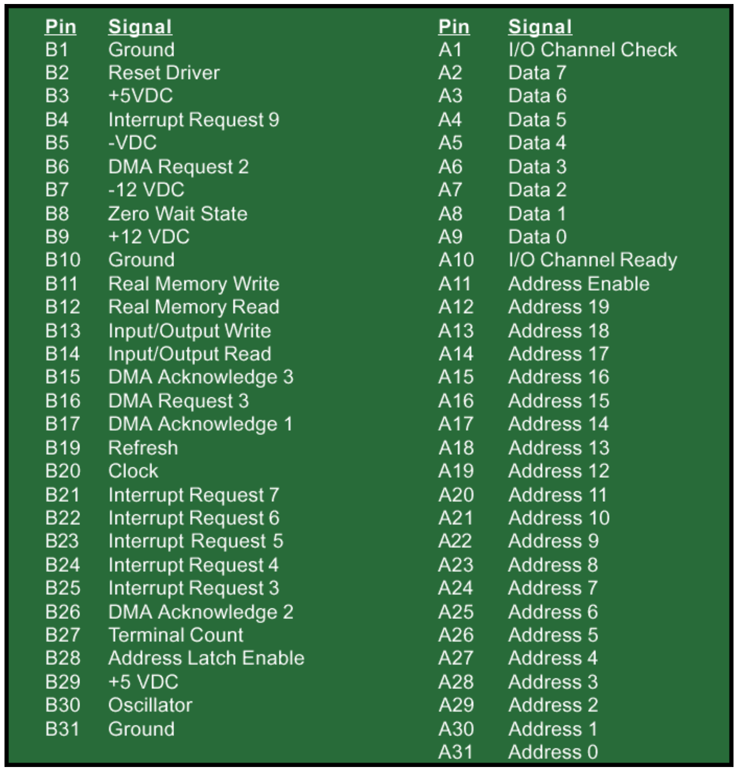
ISA bus (Original IBM PC bus) -

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

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







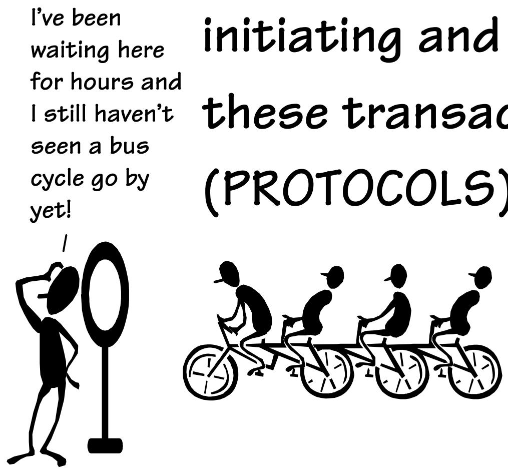
Pin	Signal	Pin	Signal
B1	Ground	A1	I/O Channel Check
B2	Reset Driver	A2	Data 7
B3	+5VDC	A3	Data 6
B4	Interrupt Request 9	A4	Data 5
B5	-VDC	A5	Data 4
B6	DMA Request 2	A6	Data 3
B7	-12 VDC	A7	Data 2
B8	Zero Wait State	A8	Data 1
B9	+12 VDC	A9	Data 0
B10	Ground	A10	I/O Channel Ready
B11	Real Memory Write	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

VME, NuBus, PCI

The function that buses serve is fairly simple:

1) They allow the movement of data from point-to-point via specific transactions (Reading, Writing, etc.)
2) They define rules for initiating and completing these transactions (PROTOCOLS)


I've been waiting here for hours and I still haven't seen a bus cycle go by yet!

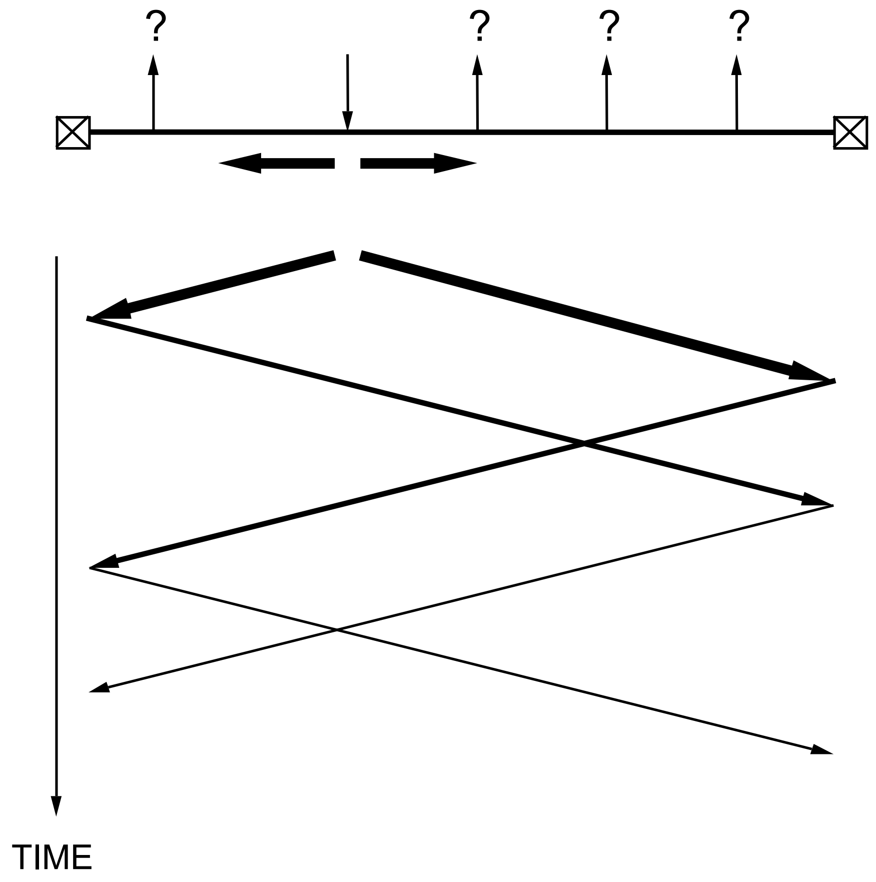
TERMINOLOGY –

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

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

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

Bus Lines as Transmission Lines



ANALOG ISSUES:

- <u>Propagation times</u>
  Light travels about 1 ft / ns
  (about 7"/ns in a wire)
- <u>Skew</u>
  Different points along the bus see the signals at different times
- <u>Reflections & standing waves</u>
  At each interface (places where the propagation medium changes) the signal may reflect if the impedances are not matched.

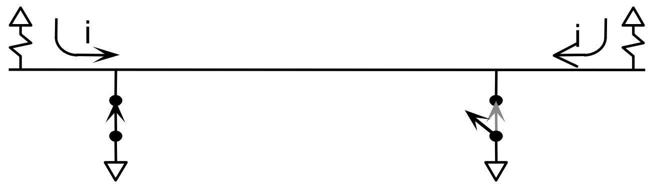
Coping with Analog Issues...

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???

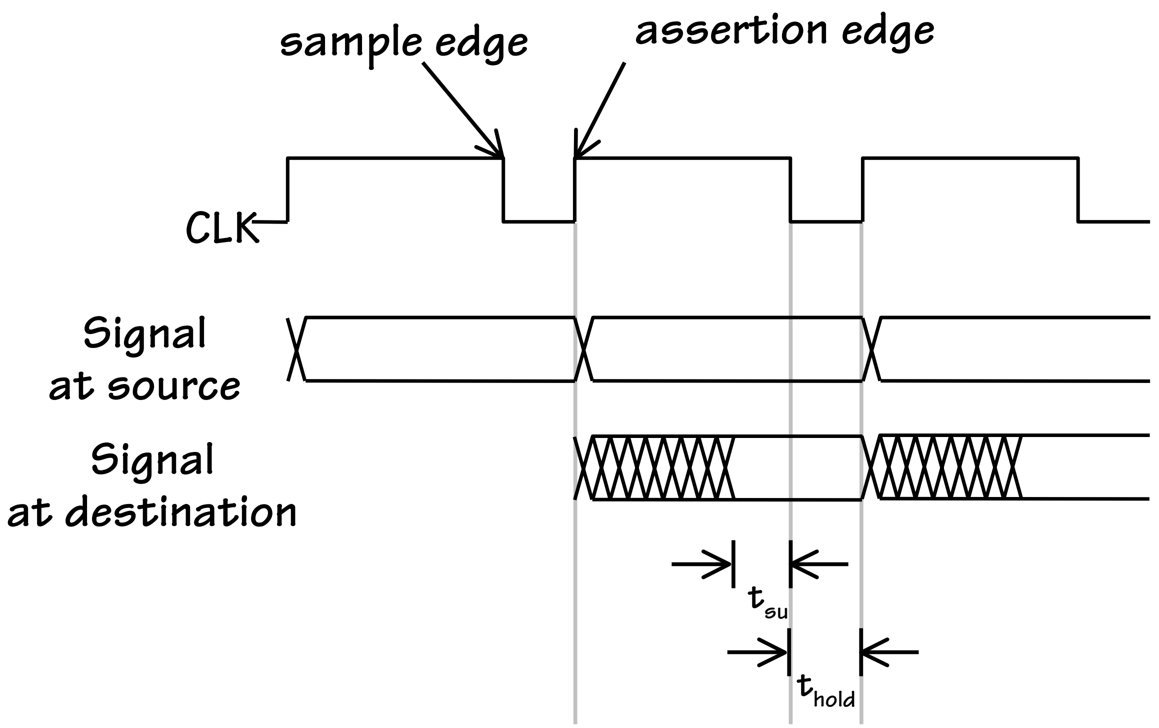


COMMON COMPROMISE: Synchronous, Self-Timed protocols

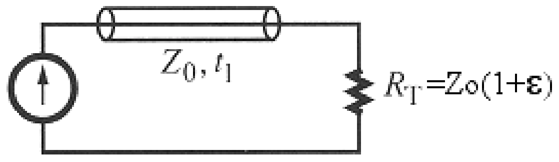
- Broadcast bus clock
- Signals sampled at "safe" times

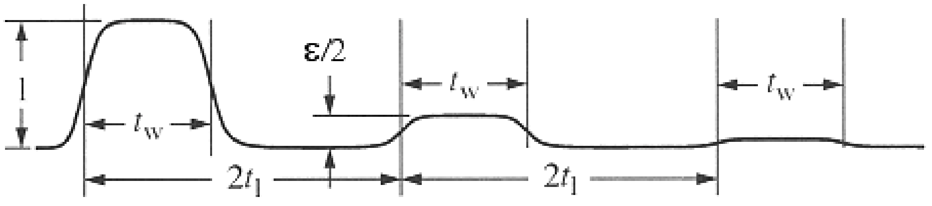
* DEAL WITH: noise, clock skew (wrt signals)

Synchronous Bus Clock Timing

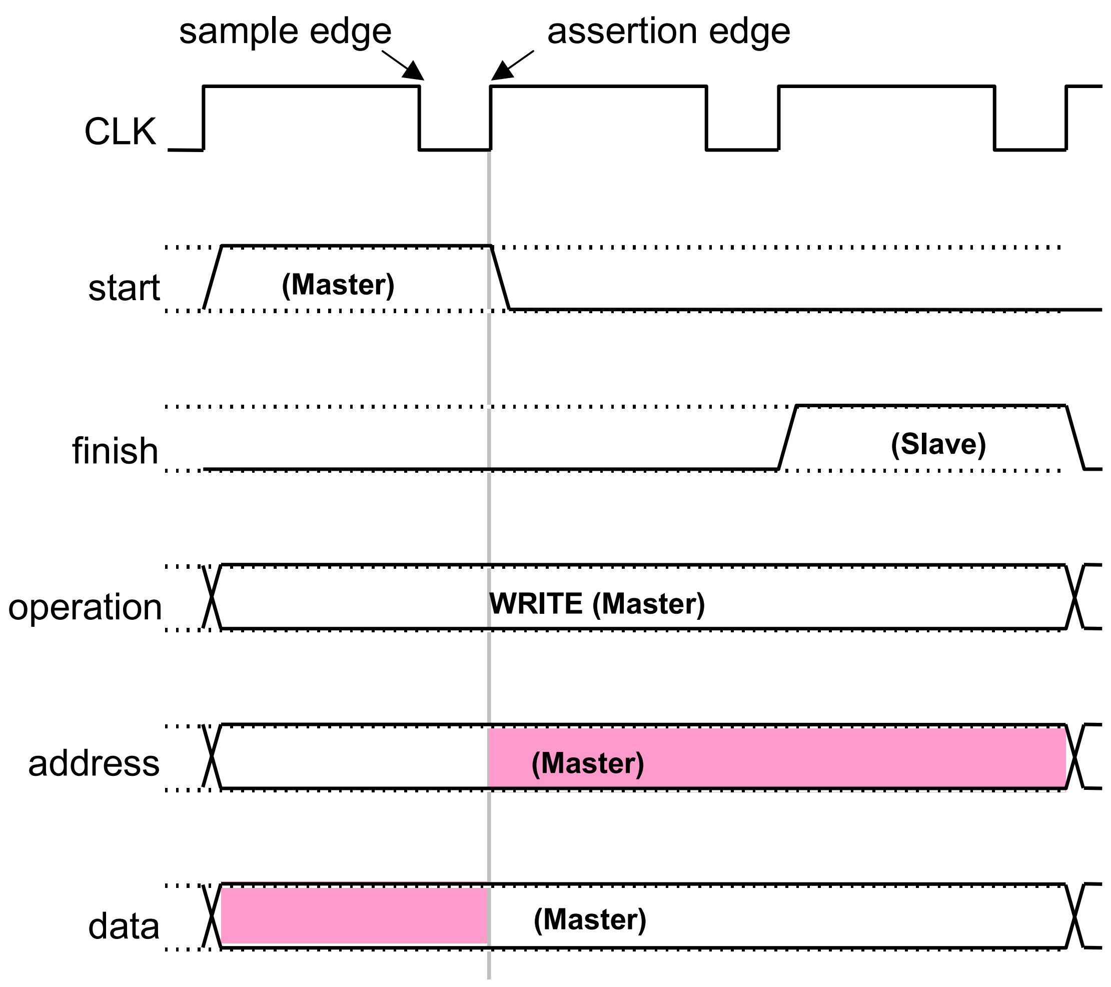


Allow for several "round-trip" bus delays so that ringing can die down.





A Simple Bus Transaction

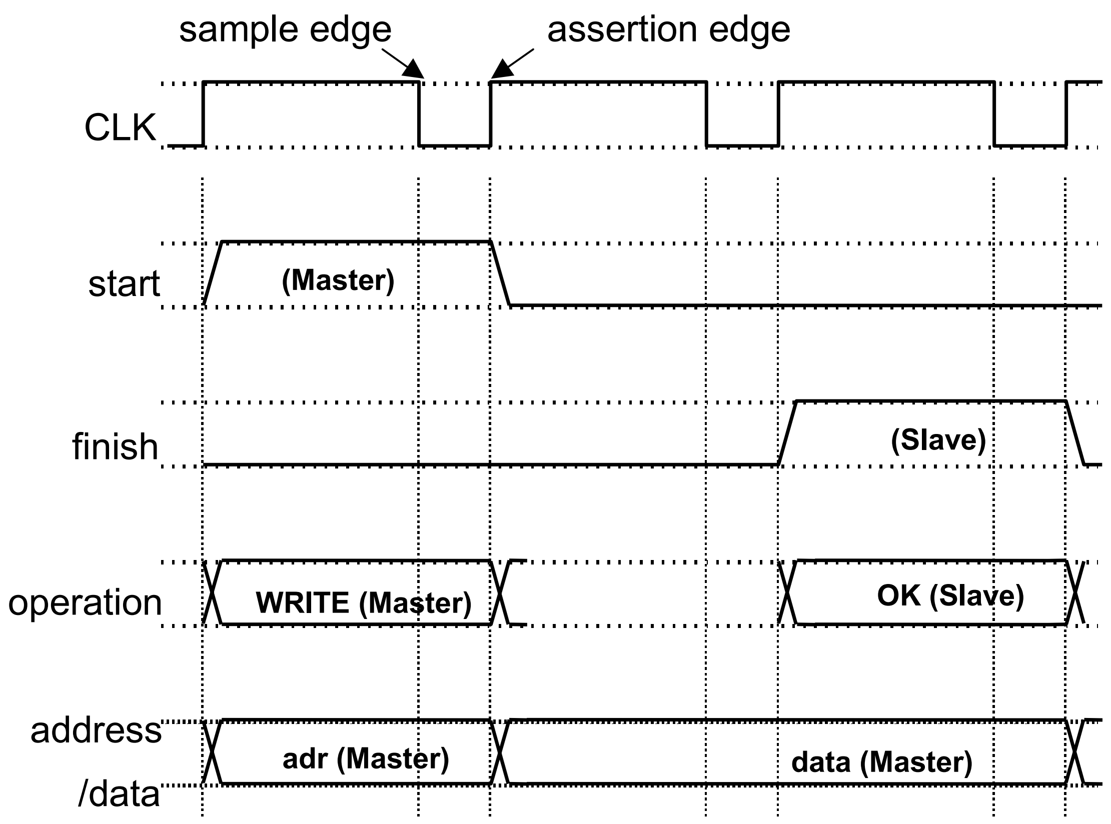


MASTER:
1) Chooses bus operation
2) Asserts an address
3) Waits for a slave to answer.

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



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

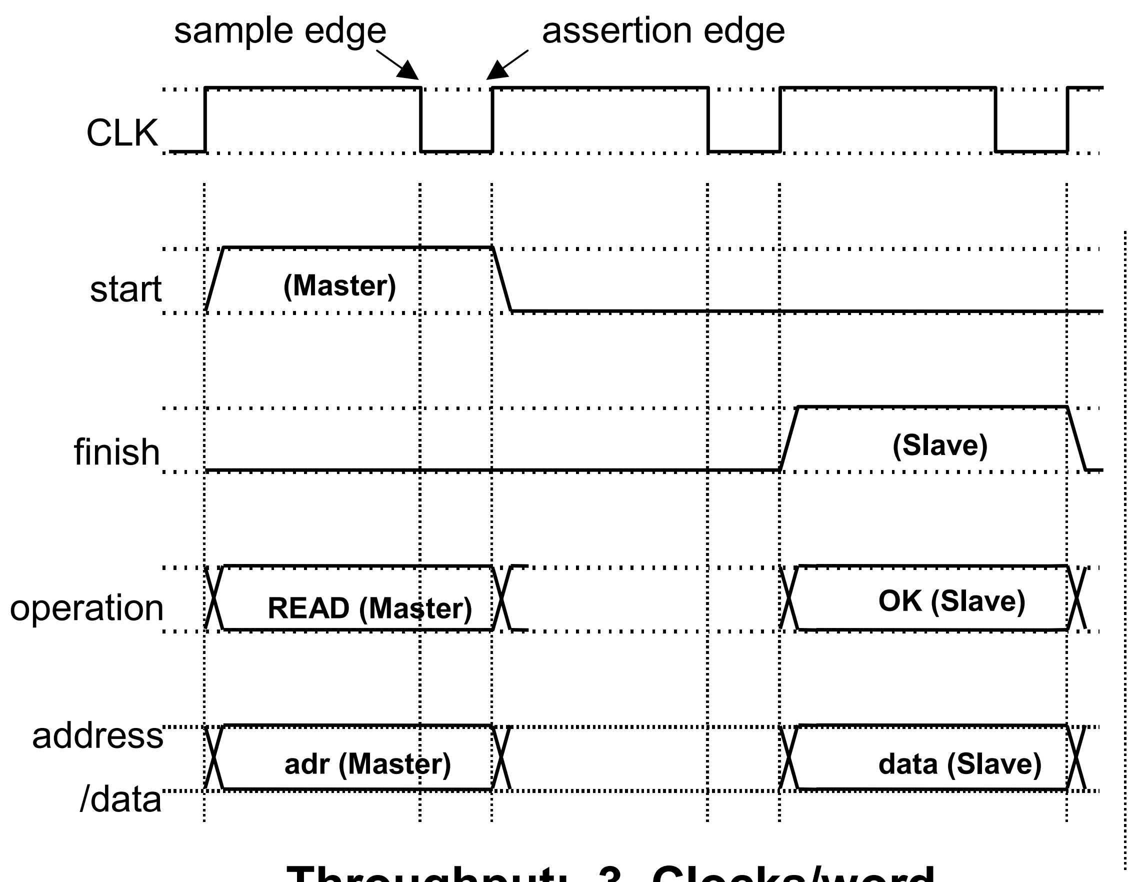
Two ways of thinking about it:
- fewer wires
- more bits of data

Slave can send an "out-of-band" 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

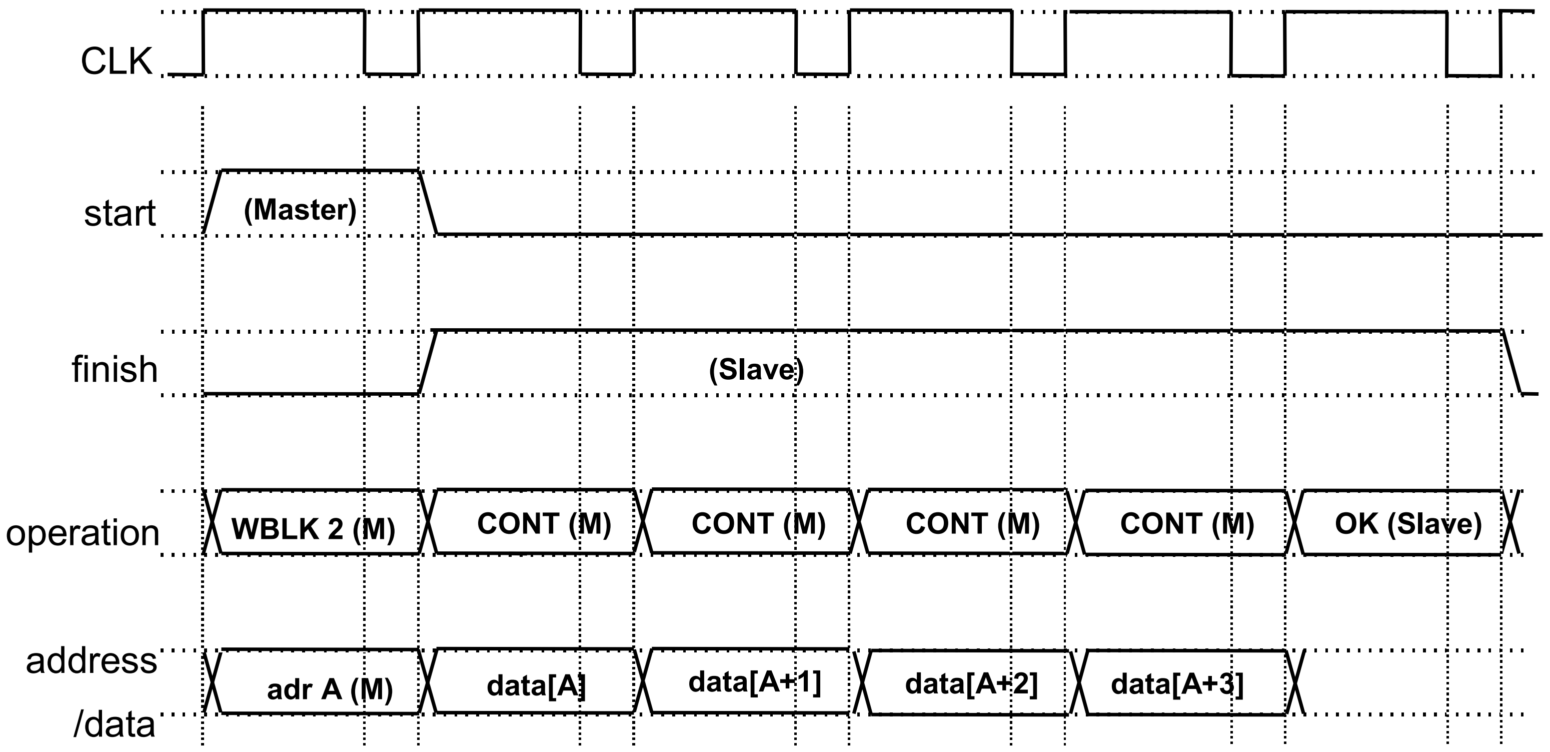


Throughput: 3 Clocks/word

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.

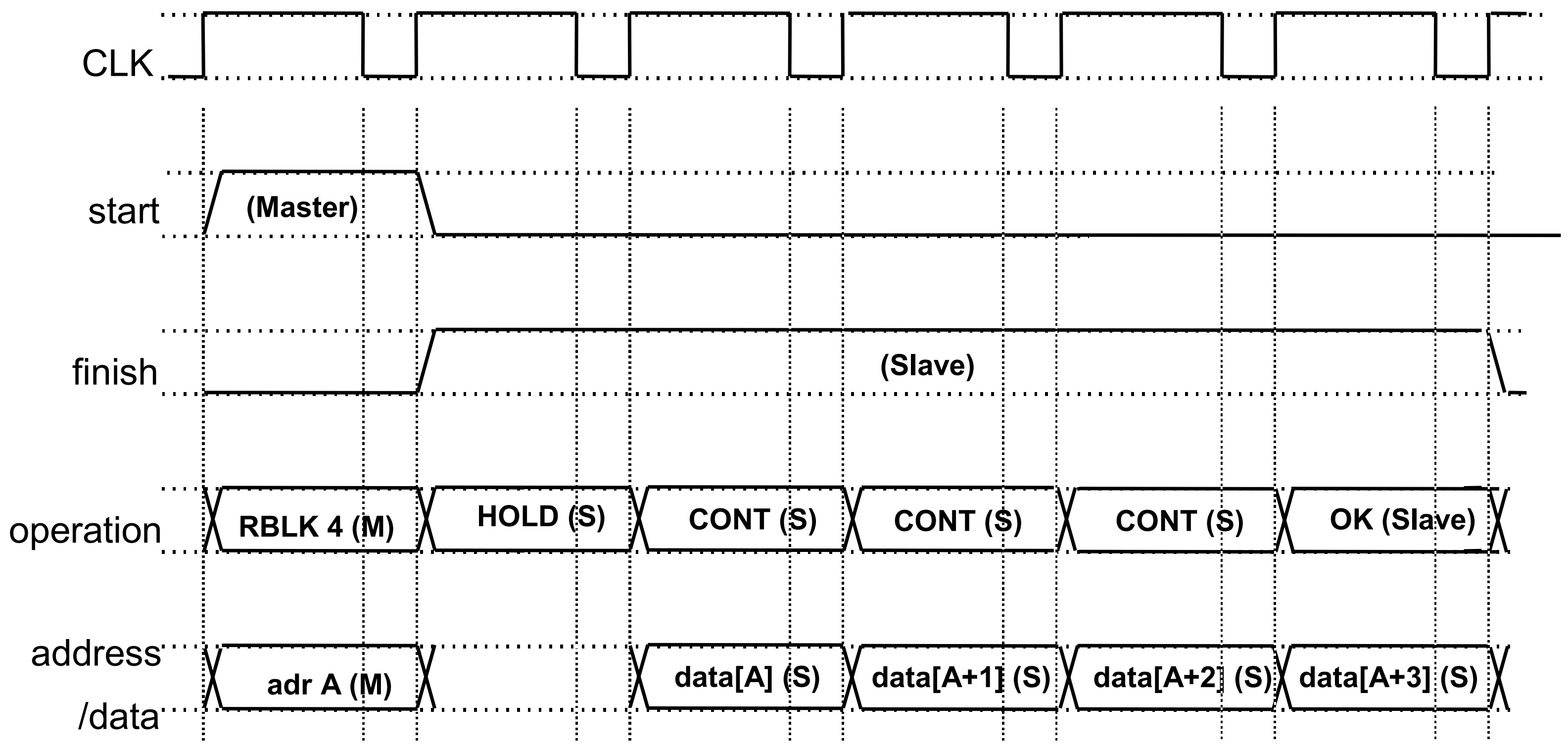
A slave can stall the read (for instance if the device is slow compared to the bus clock) by waiting several clocks before asserting the finish signal. These delays are sometimes called "WAIT-STATES"

Block Write Transfers



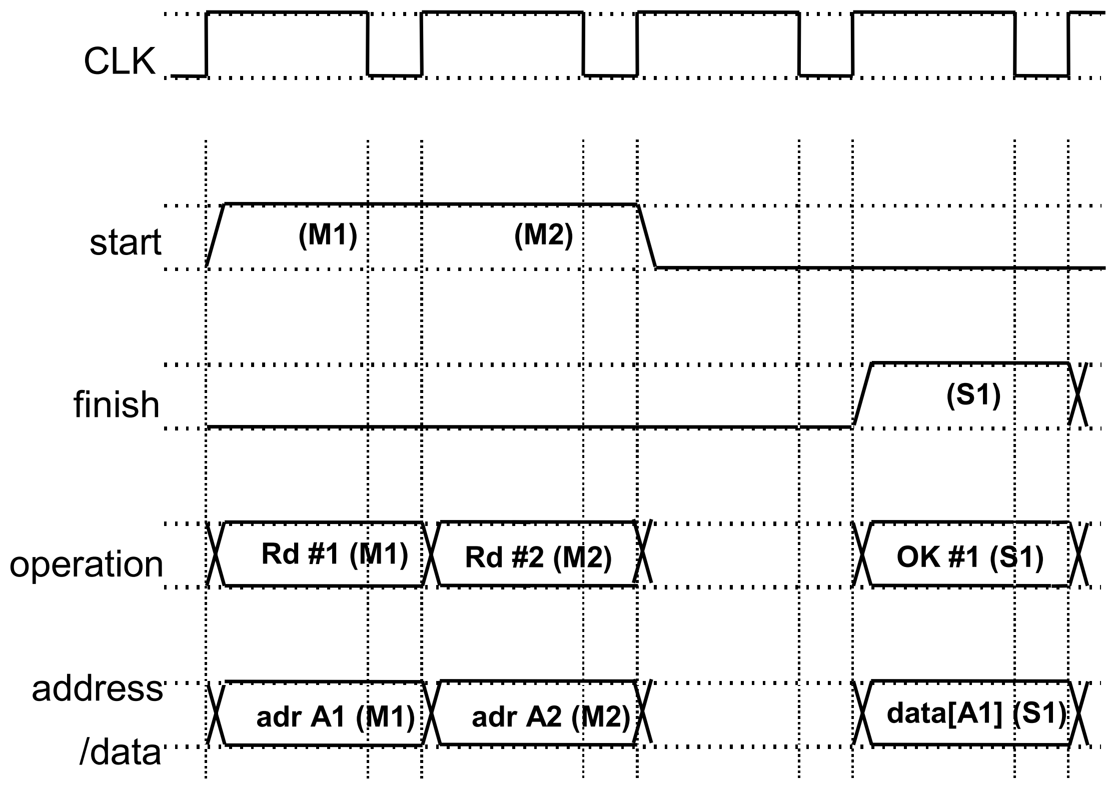
Block transfers are the way to get peak performance from a bus. A throughput of nearly 1 Clock/word is achievable on large blocks. Slaves must generate sequential addresses.

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 cache lines!

Split-Transaction Bus Operation



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:

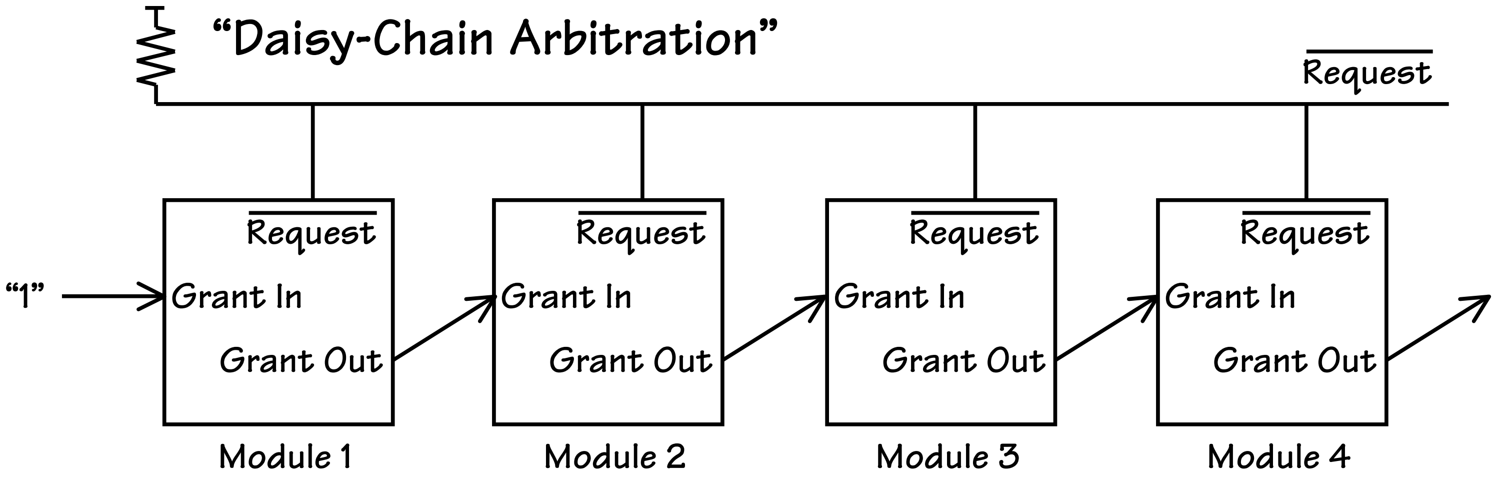
Fairness - Given uniform requests, bus cycles should be divided evenly among modules (to each, according to their needs)

Bounded Wait - There should be an upper bound on how long a module has to wait between requesting and receiving a grant

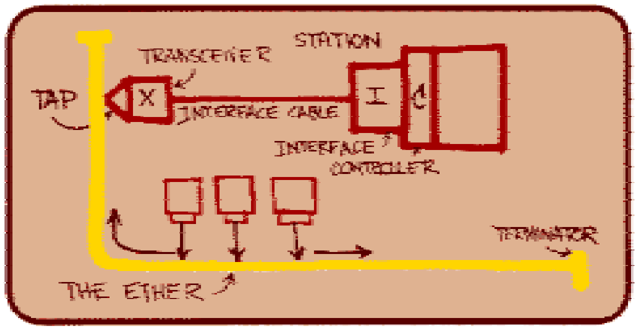
Utilization - Arbitration scheme should allow for maximum bus performance

Scalability - Fixed-cost per module (both in terms of arbitration H/W and arbitration time.

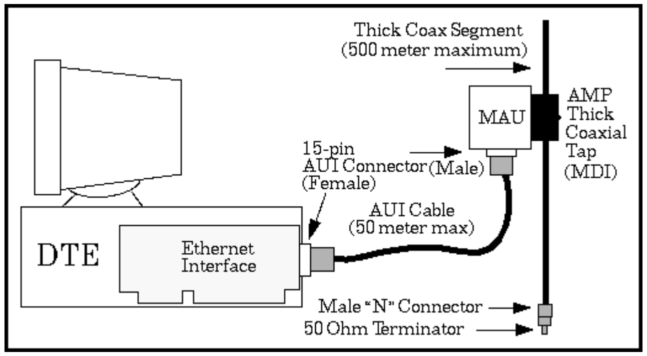
“Daisy-Chain Arbitration”


$\overline{\text{Request}}$
“1”
$\overline{\text{Request}}$
Grant In
Grant Out
Module 1
$\overline{\text{Request}}$
Grant In
Grant Out
Module 2
$\overline{\text{Request}}$
Grant In
Grant Out
Module 3
$\overline{\text{Request}}$
Grant In
Grant Out
Module 4

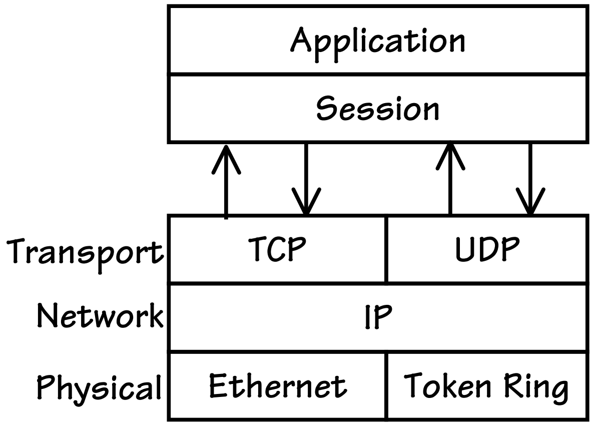
A Bus with Staying Power

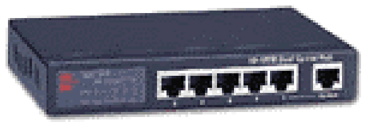


In the mid-70's Bob Metcalf (at Xerox PARC, an MIT alum) devised a bus for networking computers together.

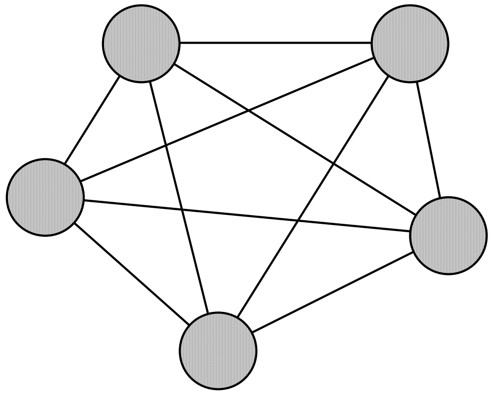


KEY IDEA: Buses are about high-level protocols, not physical interfaces.





Beyond Buses: Communication Topologies

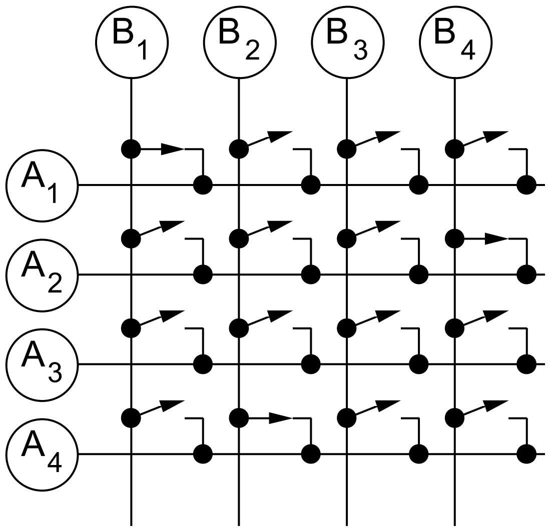


COMPLETE GRAPH:

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

CROSSBAR SWITCH:

switch dedicated between each pair of nodes; each A can be connected to one B at any time.

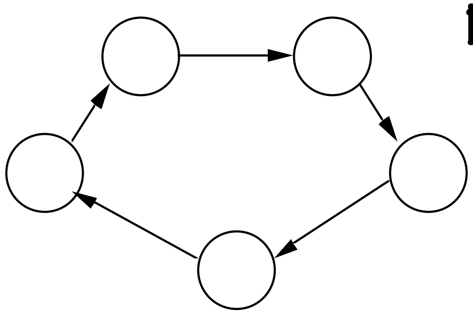


Special cases:

- A = processors, B = memories.
- A and B are same type.

DRAWBACK: Quadratic Cost!

Communication Topologies: Low-Cost Networks



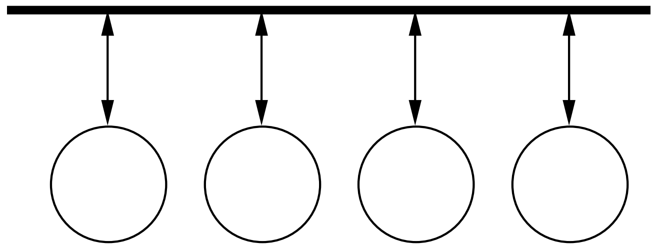
RING

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

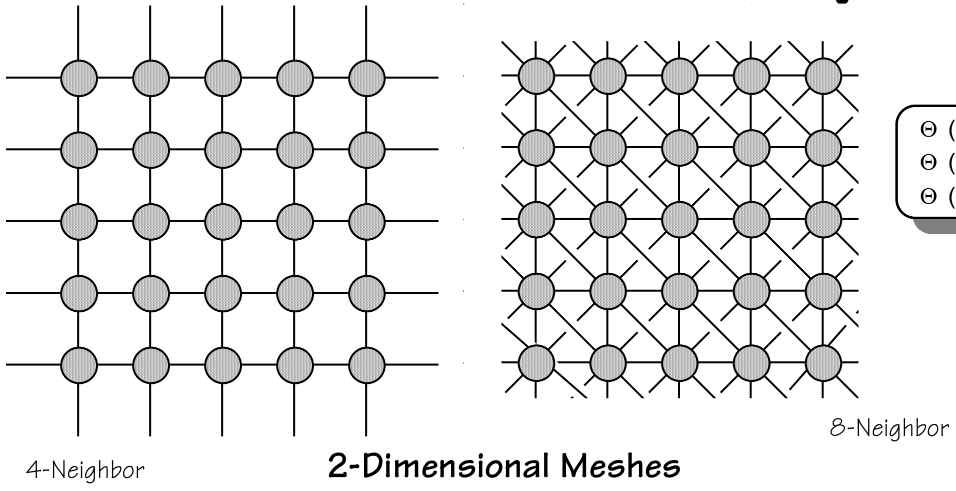
BUS

One step for random message delivery

(but only one message at a time!)

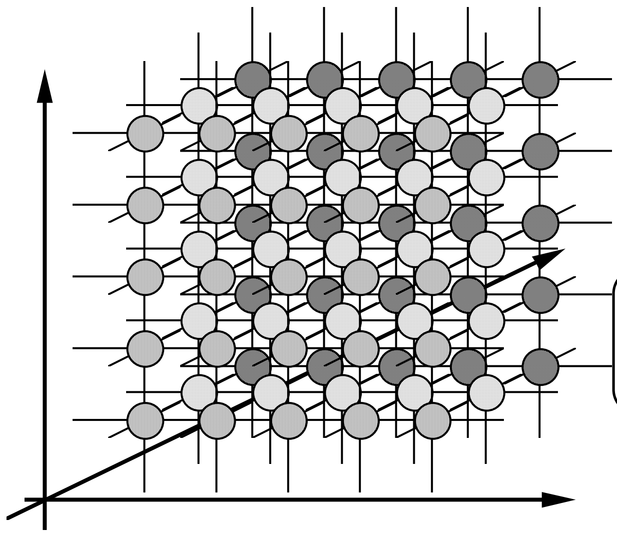


Mesh Topologies



2-Dimensional Meshes

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



3-D, 6-Neighbor Mesh

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

Nearest-neighbor connectivity:

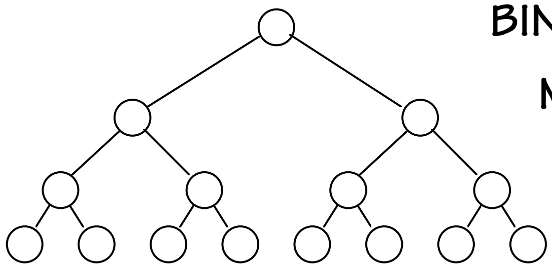
Point-to-point interconnect

- minimizes delays
- minimizes "analog" effects

Store-and-forward

(some overhead associated with communication routing)

Communication Topology: Logarithmic Latency Networks



BINARY TREE:

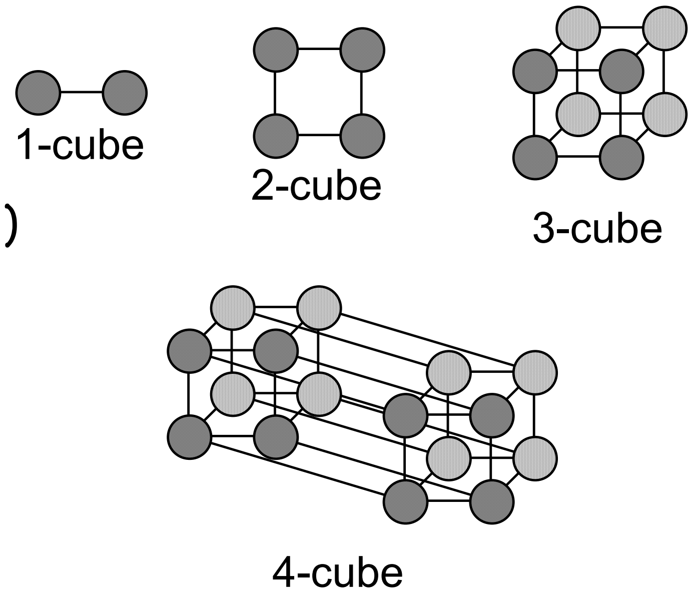
Maximum path length is $\Theta(\log n)$ steps;

Cost/node constant.

HYPERCUBE (n-cube):

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

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



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	$\Theta(n^2)$	~~$\Theta(1)$~~ $\rightarrow$	$\geq \Theta(\sqrt[3]{n})$
Crossbar	$\Theta(n^2)$	~~$\Theta(1)$~~ $\rightarrow$	$\Theta(n)$
1D Bus	$\Theta(n)$	~~$\Theta(1)$~~ $\rightarrow$	$\Theta(n)$
2D Mesh	$\Theta(n)$	$\Theta(\sqrt{n})$	
3D Mesh	$\Theta(n)$	$\Theta(\sqrt[3]{n})$	
Tree	$\Theta(n)$	~~$\Theta(\log n)$~~ $\rightarrow$	$\geq \Theta(\sqrt[3]{n})$
N-cube	$\Theta(n \log n)$	~~$\Theta(\log n)$~~ $\rightarrow$	$\geq \Theta(\sqrt[3]{n})$

IS IT REAL?

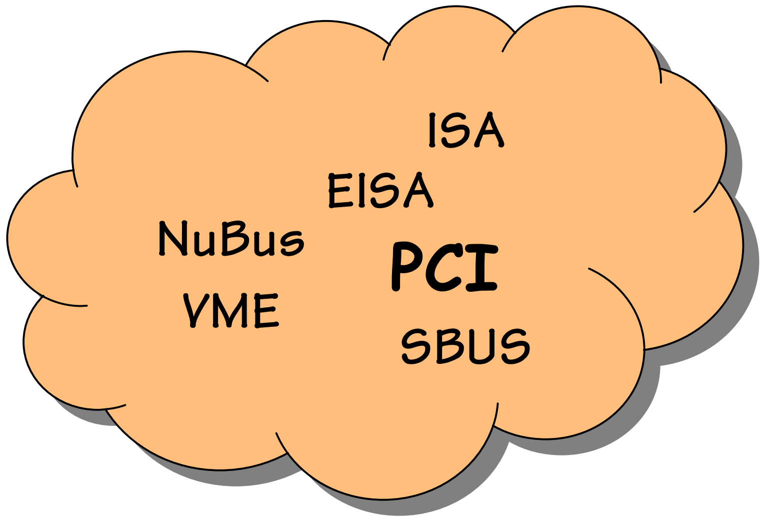
- Speed of Light: ~ 1 ns/foot (typical bus propagation: 5 ns/foot)
- Density limits: can a node shrink forever? How about Power, Heat, etc ... ?

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

Communications Futures

Backplane Buses - *still the standard*

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



New-generation communications...

- Log networks (Intel Hypercube, CMs)
- 2D Meshes (IWARP, ...)
- 3D Meshes (J Machine)
- 4-neighbor, 3D mesh (NuMesh Diamond lattice)

  6-neighbor, 3D mesh (cube cut on its diagonal)

  Nodes plug together like Legos!