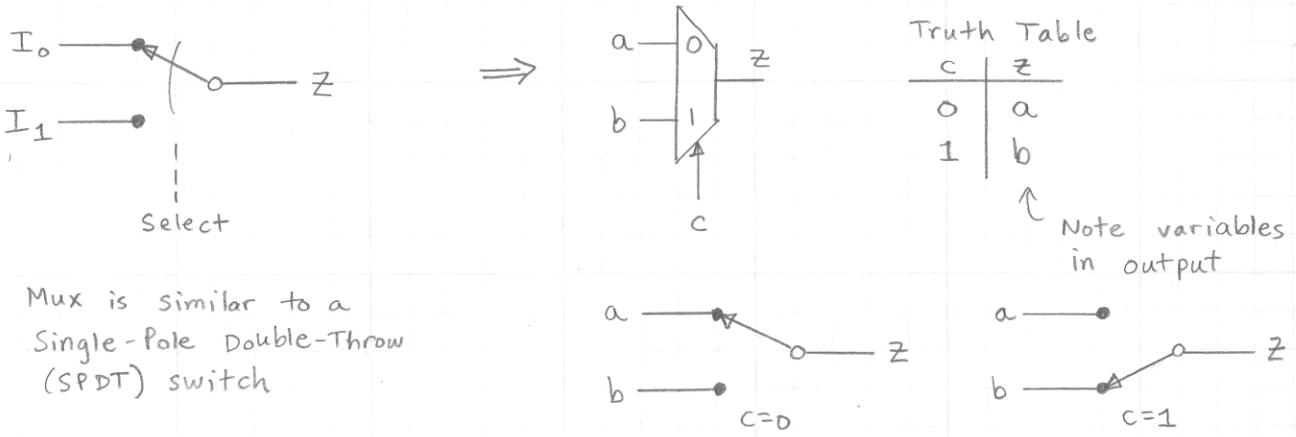
Digital Chip Classifications

			# transistors/chip	Examples	Cost (High Volume)
SSI	Small Scale	Integration	~ 10s	INV, NAND	$1
MSI	Medium	"	~ 10s – 100s	adder, counter	$1.50
LSI	Large	"	~ 1M	simple micro	
VLSI	Very Large	"	~ 1–10 M	processors, mem	
ULSI	Ultra Large	"	~ 10M – 1B	system-on-chip (SoC), large CPUs	$10 $50

Cost is more closely linked to number of chips (volume) than number of transistors.

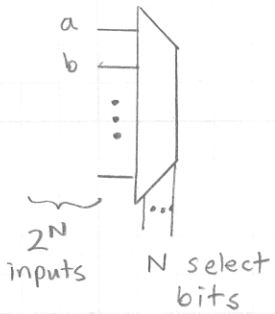
9.2 Multiplexers (Mux)

Multiplexer selects one of its inputs and routes it to the output



Mux is similar to a Single-Pole Double-Throw (SPDT) switch

In general,

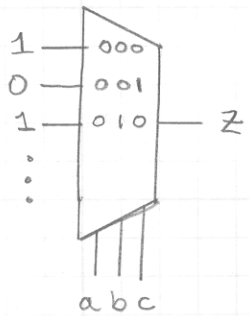


Ex: Implementing a truth table using a multiplexer...

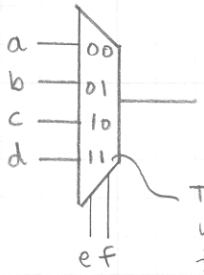
a	b	c	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	⋮	⋮	
⋮			

a b c ≡ mux select bits
Z ≡ output of mux

values of Z ≡ mux inputs



Mux Marking



These values indicate which input is connected to the output when the control input(s) equal these values.

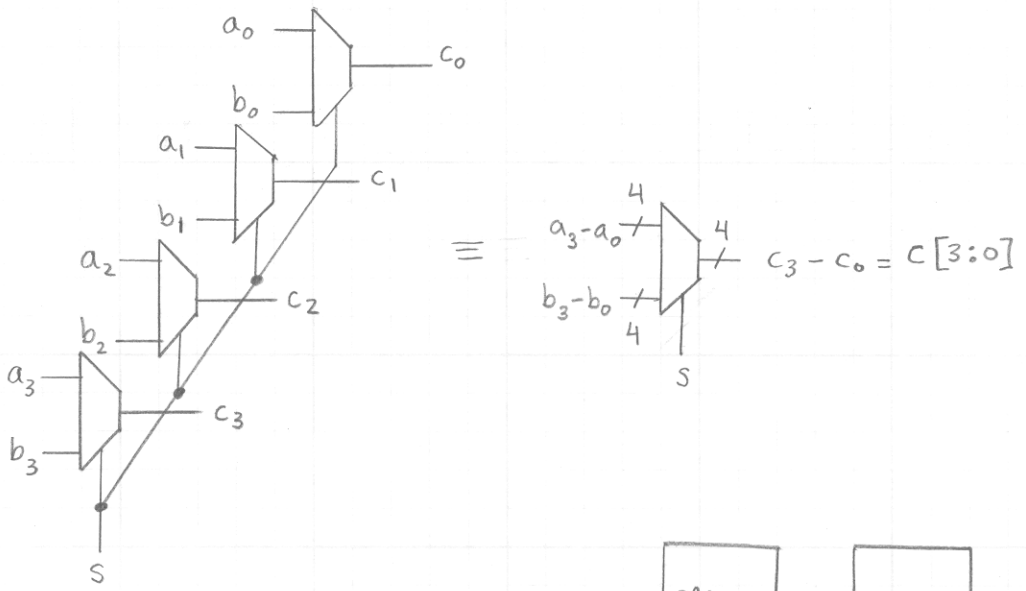
e	f	z
0	0	a
0	1	b
1	0	c
1	1	d

Bus Notation A bus is a "bundle" of wires which are conceptually related (e.g., an 8-bit binary number on a set of 8 wires, each bit on its own wire)

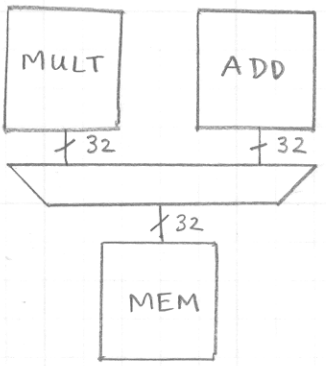
Shorthand bus notation:

is exactly equal to

Ex: We can stack up muxes in parallel to create multi-bit muxes

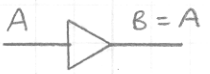


Ex: Routing results in a CPU

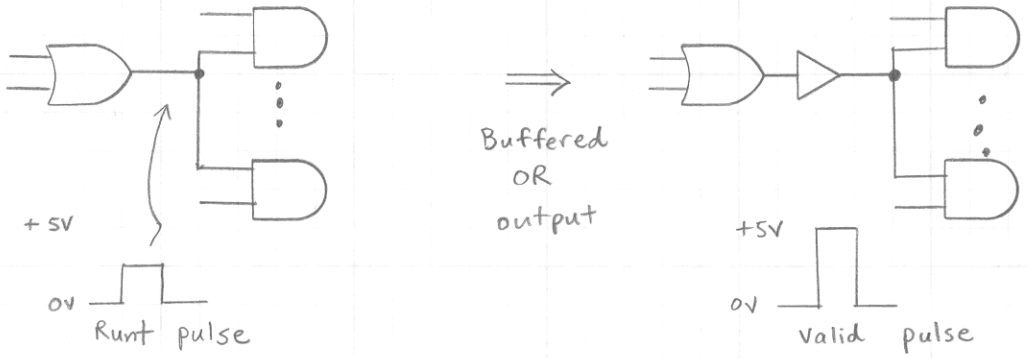


9.3 Three-State Buffers

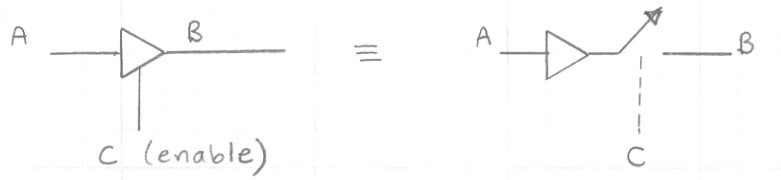
A <u>buffer</u> simply passes its inputs to its outputs:



Logically, it does nothing, but electrically it can help drive a large number of loads:

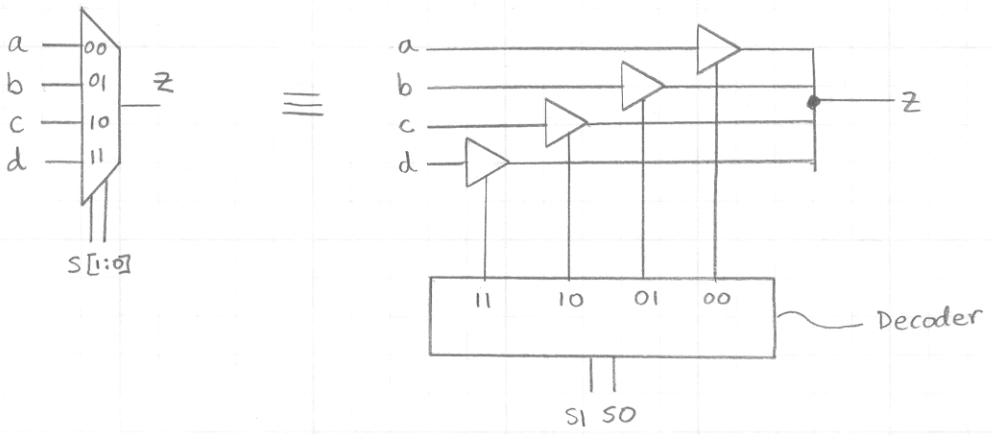


A <u>three-state (tri-state)</u> buffer has three output states: 0, 1, high impedance (disconnected)

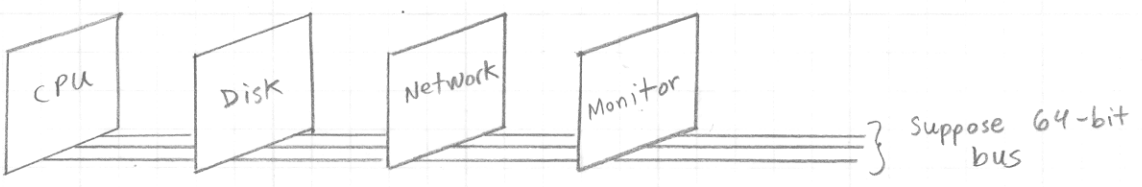


C	B
0	High impedance (Hi-Z)
1	A

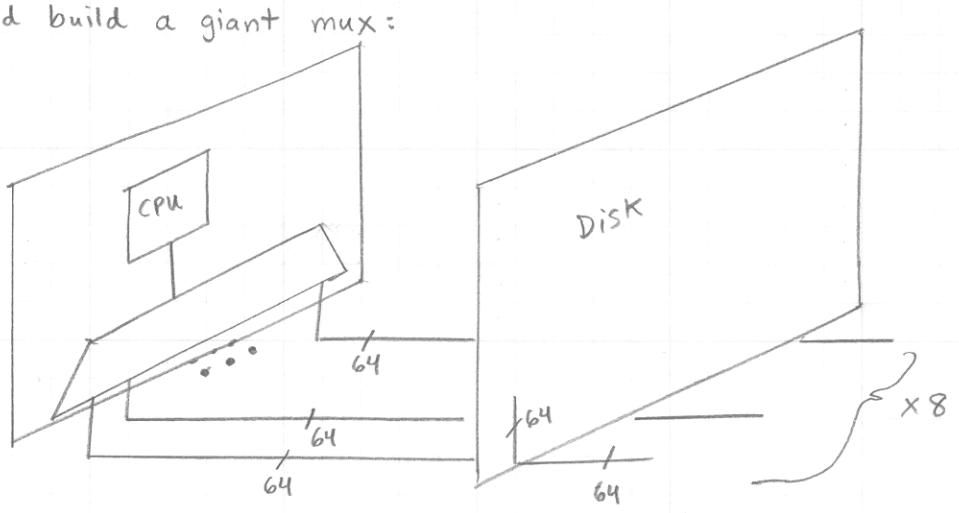
<u>Ex:</u> Use tri-state buffers to build multiplexers:



<u>Ex:</u> Suppose we need to communicate with one module at a time where there are from 1-8 modules. For example, on the PCI bus inside your computer:

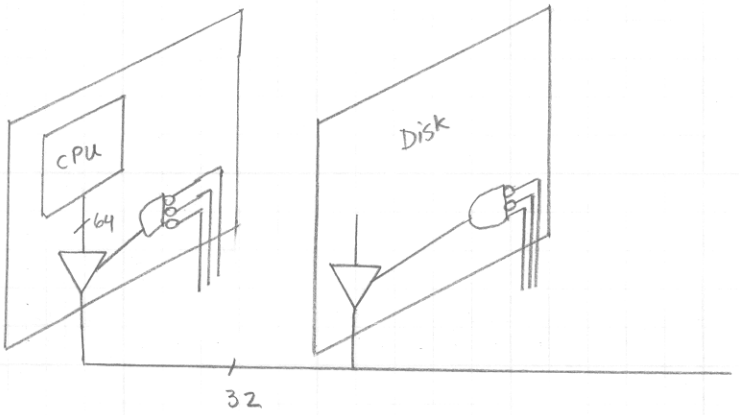


Could build a giant mux:



but the motherboard requires 64 x 8 = 512 signal wires!

Better to use tri-state buffers on each board and activate them when the three bit address matches:



Now we only have 64 + 3 = 67 motherboard wires!

9.4 Decoders and Encoders

For some applications (like building a mux out of tri-state buffers), we need a box to "decode" selector bits into single enables:

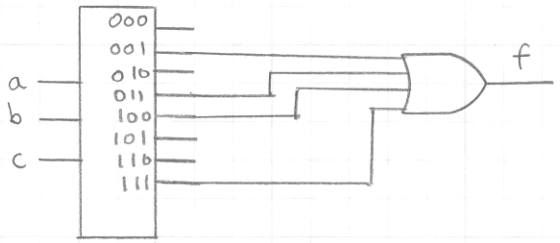
a	b	w	x	y	z
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Only one output is high at a time.

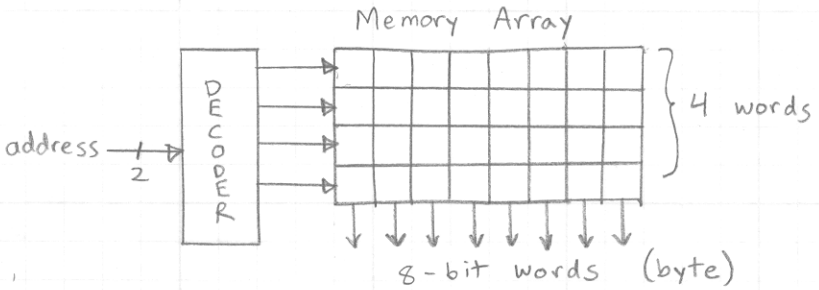
Called a <u>decoder.</u>

Each row generates exactly one minterm.

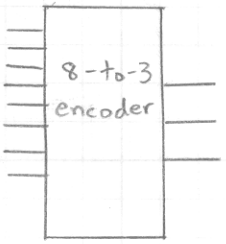
Ex: Use a decoder to generate a boolean function: $f(a,b,c) = \sum m(1,3,4,7)$



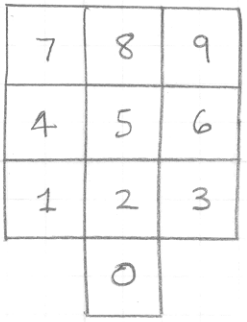
Ex: Also, an address decoder is an essential part of any multi-word memory:



An encoder performs the inverse operation of a decoder:



Ex: A touch-tone keypad:



$\Longrightarrow$ 10 inputs, coded to 4-bit number

Also need an extra "valid" bit to indicate when the output is valid (i.e., a key is being pressed).

Encoder does this (valid is just OR of inputs), but what if more than 1 button is pushed simultaneously?

10 input bits ($k_0 - k_9$) $\Rightarrow$ 1024 combinations
4 output bits $\Rightarrow$ 16 combinations

Possibility #1: Output is invalid if more than 1 key is pressed:

K_0	K_1	K_2	...	K_q	Output	Valid
0	0	0		0	XXXX	0
1	0	0		0	0000	1
0	1	0		0	0001	1
0	0	1			0010	1
	⋮			⋮	⋮	⋮
0	0	0	...	1	1001	1
1	1	0	...	0	XXXX	0
1	0	1	...	0	XXXX	0
1	0	0	...	1	XXXX	0
		⋮				

Need to specify many states!

Possibility #2: Highest number pressed wins

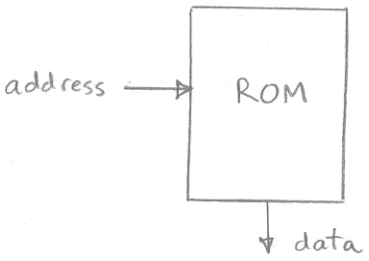
K_0	K_1	K_2	...	K_q	output	Valid
(single bit high cases same as above)					⋮	⋮
		⋮		⋮		
0	0	0	...	1	1001	1
X	1	0	...	0	0001	1
X	X	1	...	0	0010	1
X	X	X	...	1	1001	1
		⋮			⋮	⋮

Higher position inputs take priority over lower position inputs ⇒

called a priority encoder

9.5 Read-Only Memories (ROMs)

A Read-Only Memory (ROM) is a memory whose contents are permanent or semi-permanent (non-volatile)

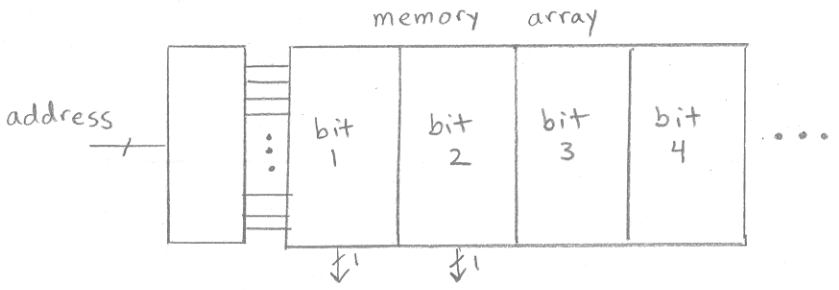


- Can only read it (no writes during normal operation)
- For n address bits, there can be up to 2^n words in the memory

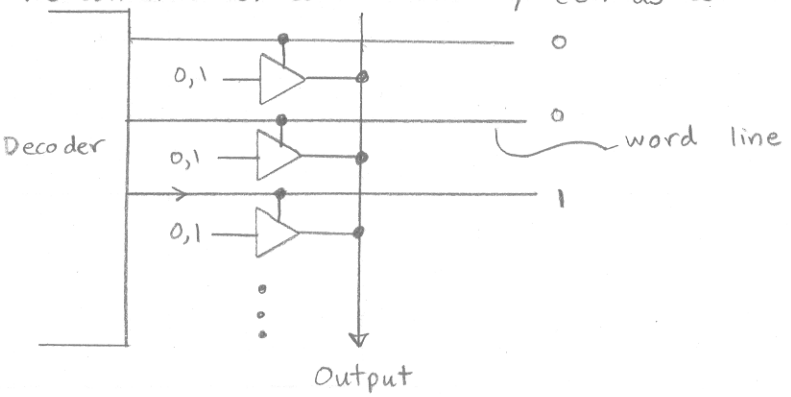
Specify contents with a truth table:

address	output
000	0
001	1
010	0
011	1
⋮	

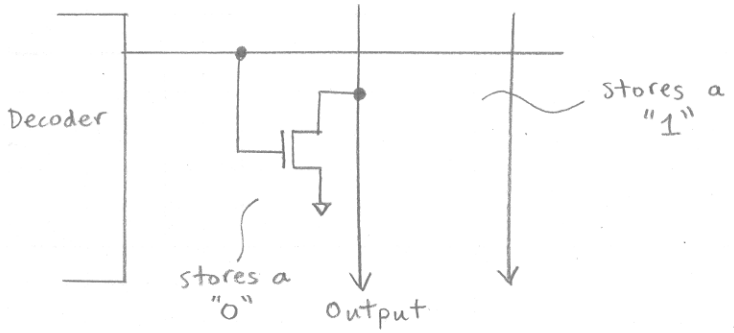
} completely arbitrary values – we choose



We can think of a ROM memory cell as a tri-state buffer: (enabled by decoder output)



We can also think of the memory cells as transistors:



Think of the address decoder as a big mux which selects the activated memory row.

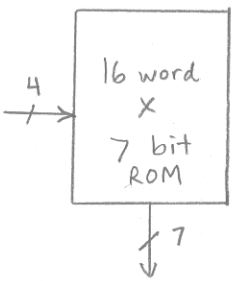
ROMs are the simplest implementation of a truth table - in fact, a truth table is how we specify a ROM!

A B C D	E F G H I J K
0 0 0 0	0 1 0 1 0 1 1
0 0 0 1	1 1 0 0 0 0 1
⋮	⋮
1 1 1 1	0 0 1 0 1 1 0

4 address bits

output bits = 7

words in memory = 16

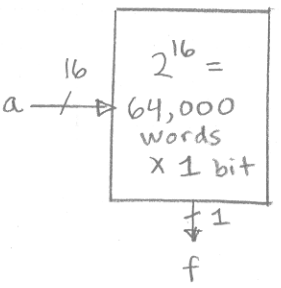


ROMs are a completely general solution for implementing combinational logic, but the solution depends on "randomness" of function, among other things.

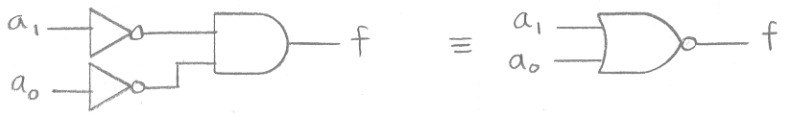
Ex: Implement f is high when a 16-bit number $\{a_{15}, a_{14}, \ldots a_0\}$ = a[15:0] is a factor of 4

ROM:

a_{15}	a_{14}	...	a_3	a_2	a_1	a_0	f
0	0		0	0	0	0	1
			0	0	0	1	0
			0	0	1	0	0
			0	0	1	1	0
			0	1	0	0	1
			0	1	0	1	0



Logic: f is 1 when $a_1 = 0$ and $a_0 = 0$



Some types of ROMs:

1. Permanent (mask programmable): bits set when chip is made
2. One-time programmable (OTP) ROM, PROM
3. Electrically Erasable programmable ROM: EEPROM, flash memory, nonvolatile memory
4. Ultraviolet light (UV) erasable programmable ROM

9.8 Field-Programmable Gate Arrays

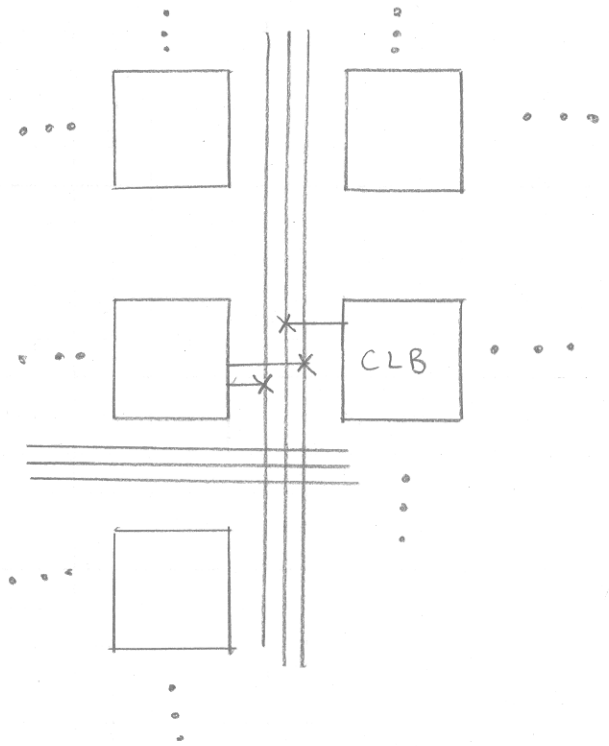
Making custom integrated circuits is expensive and time consuming:

Cost to design and pay for masks (similar to a photographic negative) for each layer of an I.C. can be $1000 → $25,000 each, usually 10-30 masks.

Gate Array covers a chip with logic gates and you pay to customize a few top layers of metal wiring

Field-Programmable Gate Array (FPGA)

Last layer of connections done with a reconfigurable network circuit (through software).



User programs:

- Small logic blocks, configurable logic blocks (CLBs)
- Connections between blocks