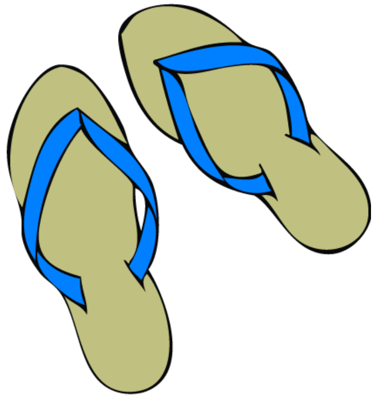


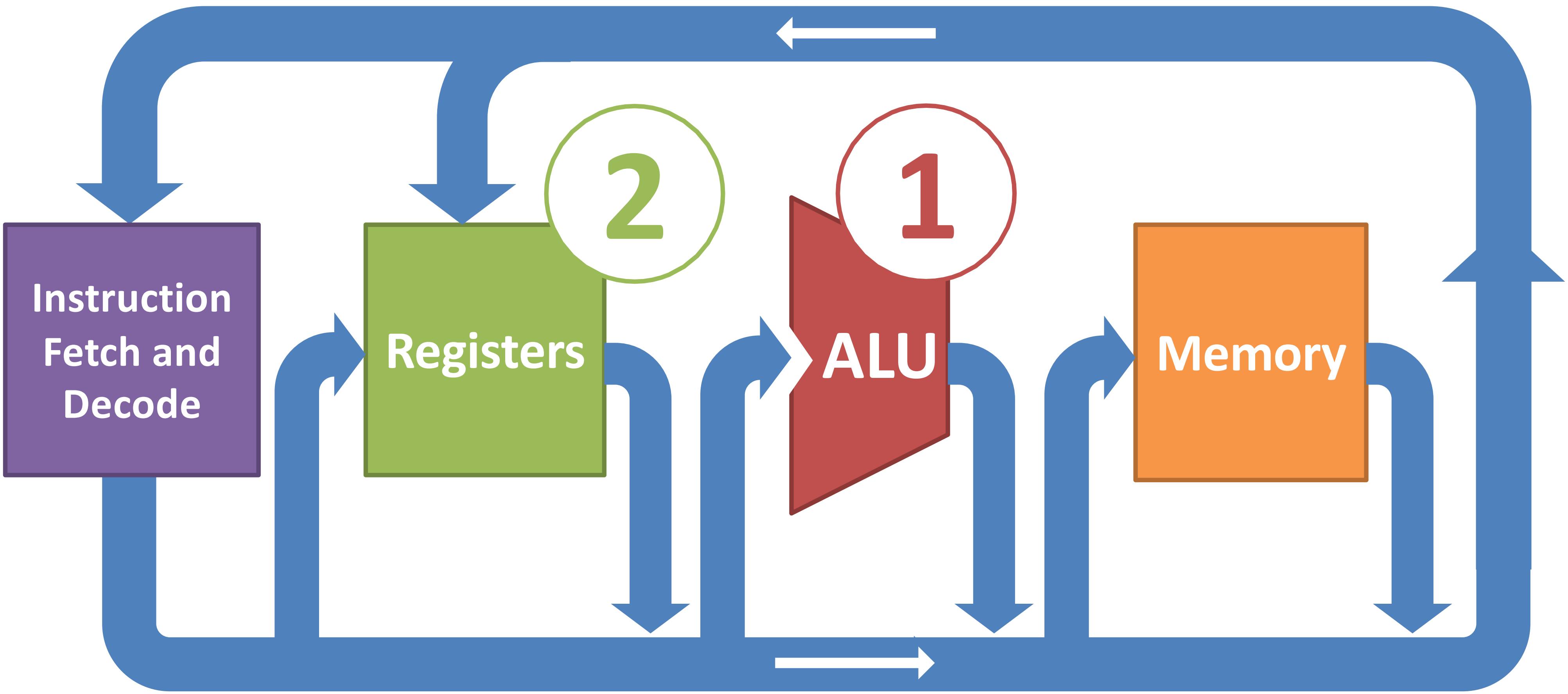
Latches, Flip-flops, and Registers

Sequential logic: fundamental elements to store values

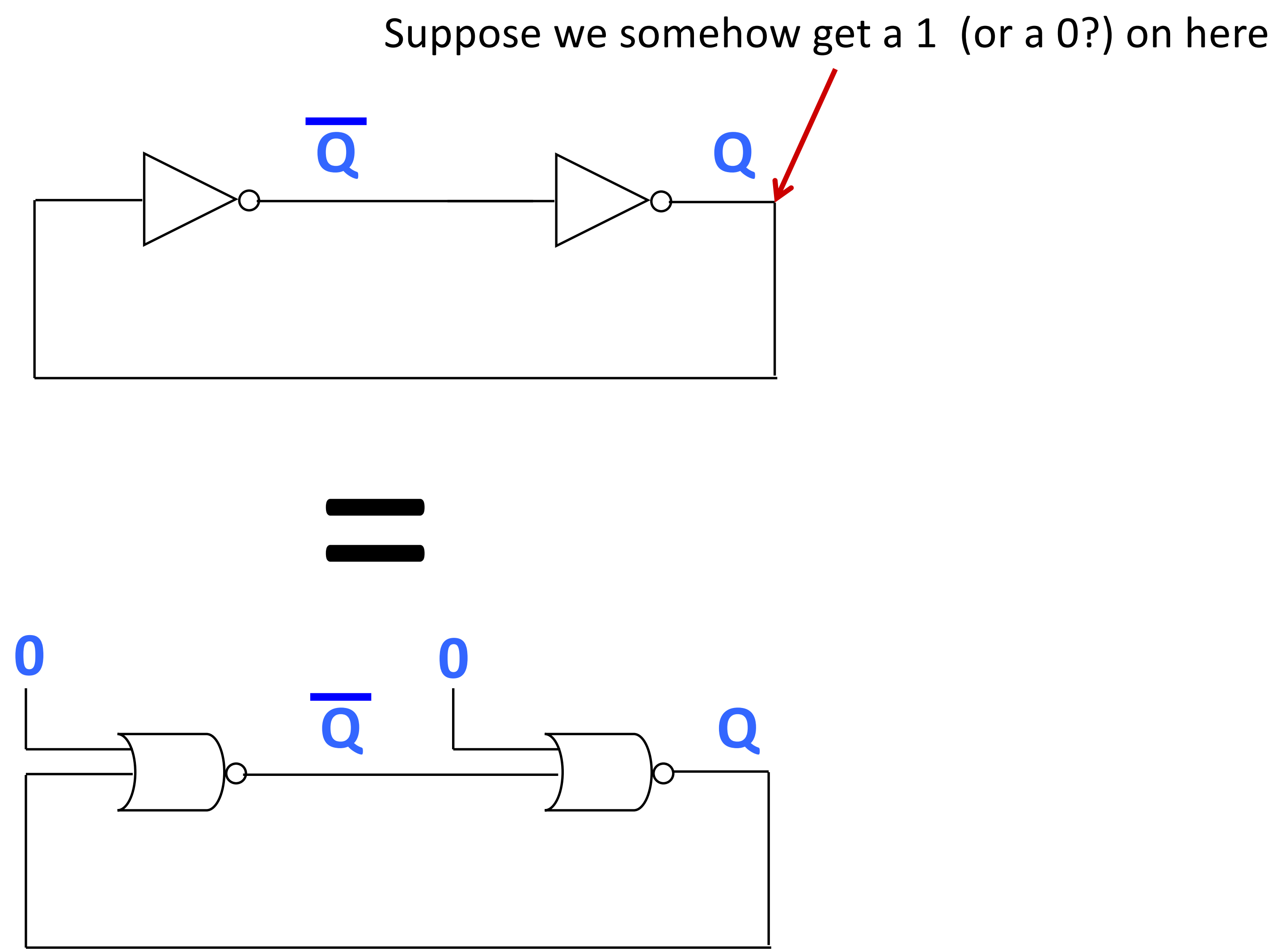
Output depends on inputs *and stored values*.

(vs. combinational logic: output depends only on inputs)

Processor: Data Path Components

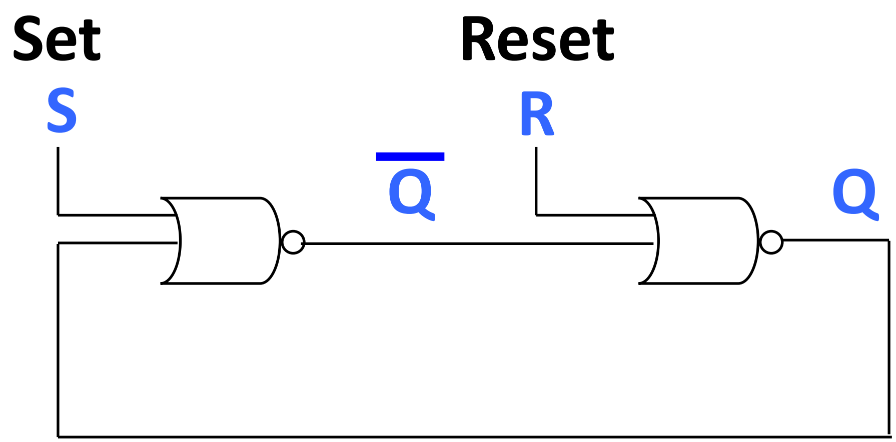


Bistable latches

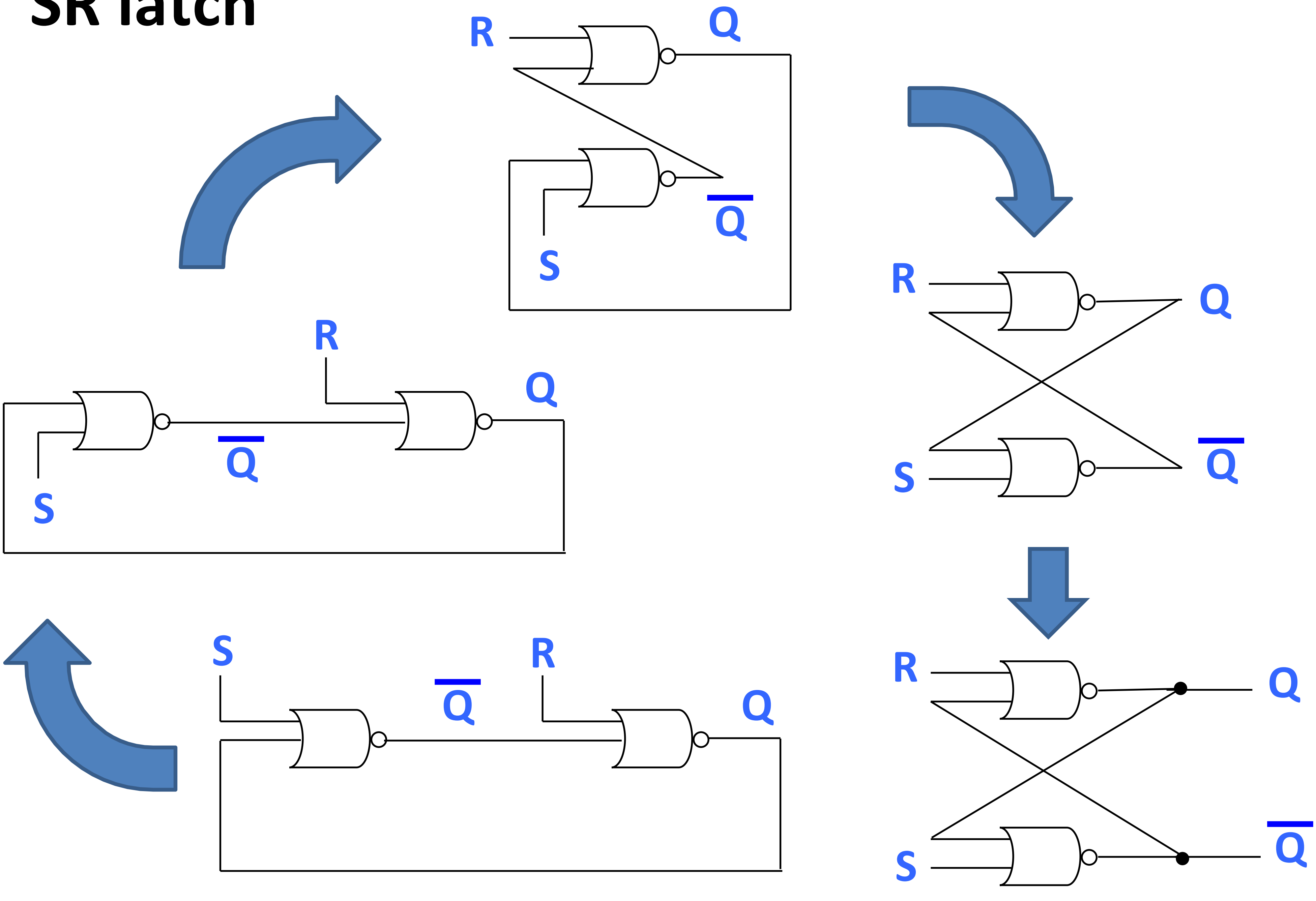


SR latch

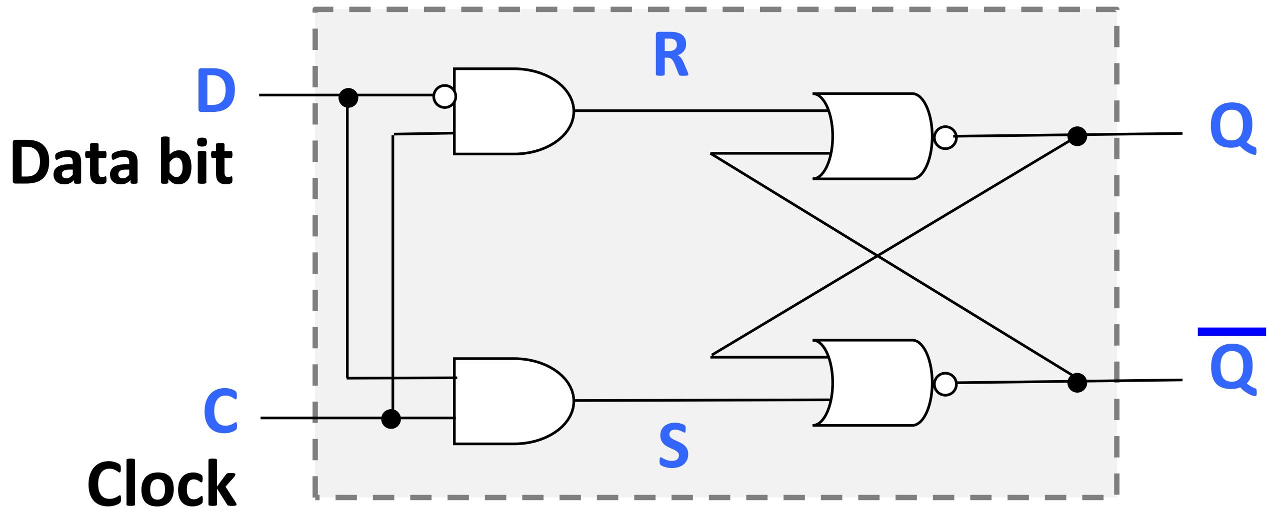
S	R	Q	Q'	Q (stable)	Q' (stable)
0	0	0	0	?	?
0	**0**	**0**	**1**	**0**	**1**
0	**0**	**1**	**0**	**1**	**0**
0	0	1	1	?	?
1	**0**	**?**	**?**	**1**	**0**
0	**1**	**?**	**?**	**0**	**1**



SR latch


R
Q
S
$\overline{Q}$
R
Q
S
$\overline{Q}$
R
Q
S
$\overline{Q}$
R
Q
S
$\overline{Q}$
S
R
$\overline{Q}$
Q

D latch



if **C = 0**, then SR latch stores current value of Q.

if **C = 1**, then D flows to Q:

if **D = 0**, then **R = 1** and S = 0, **Q = 0**

if **D = 1**, then R = 0 and **S = 1**, **Q = 1**

Time matters!

D

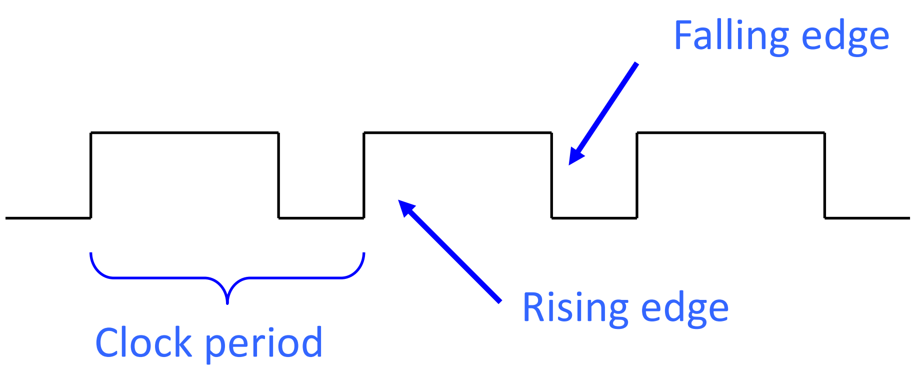
C

Q

Clocks

Clock: free-running signal
with fixed **cycle** time = **clock period** = T.

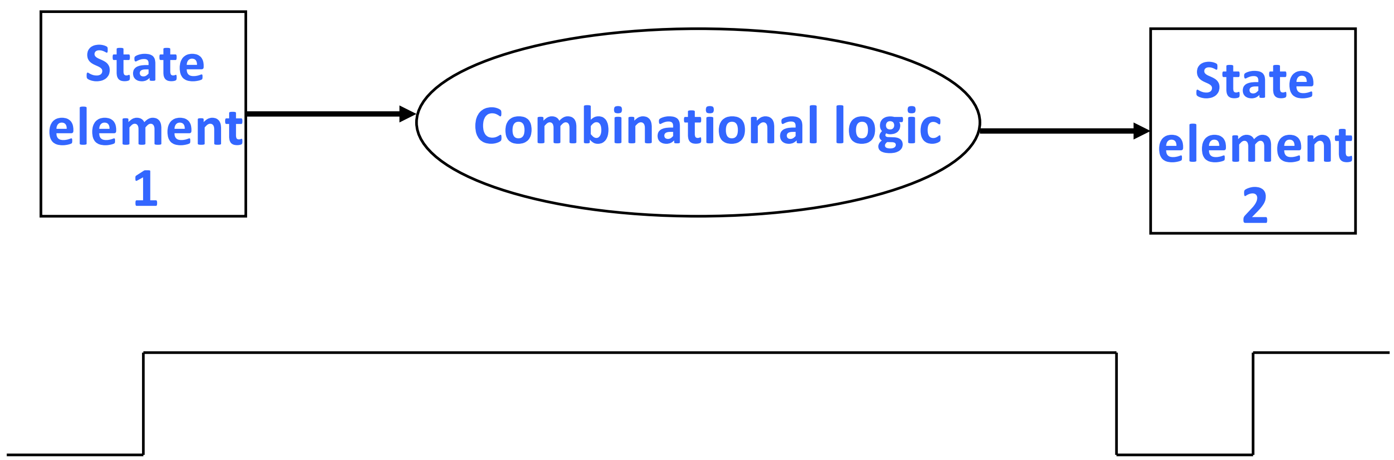
Clock frequency = 1 / clock period



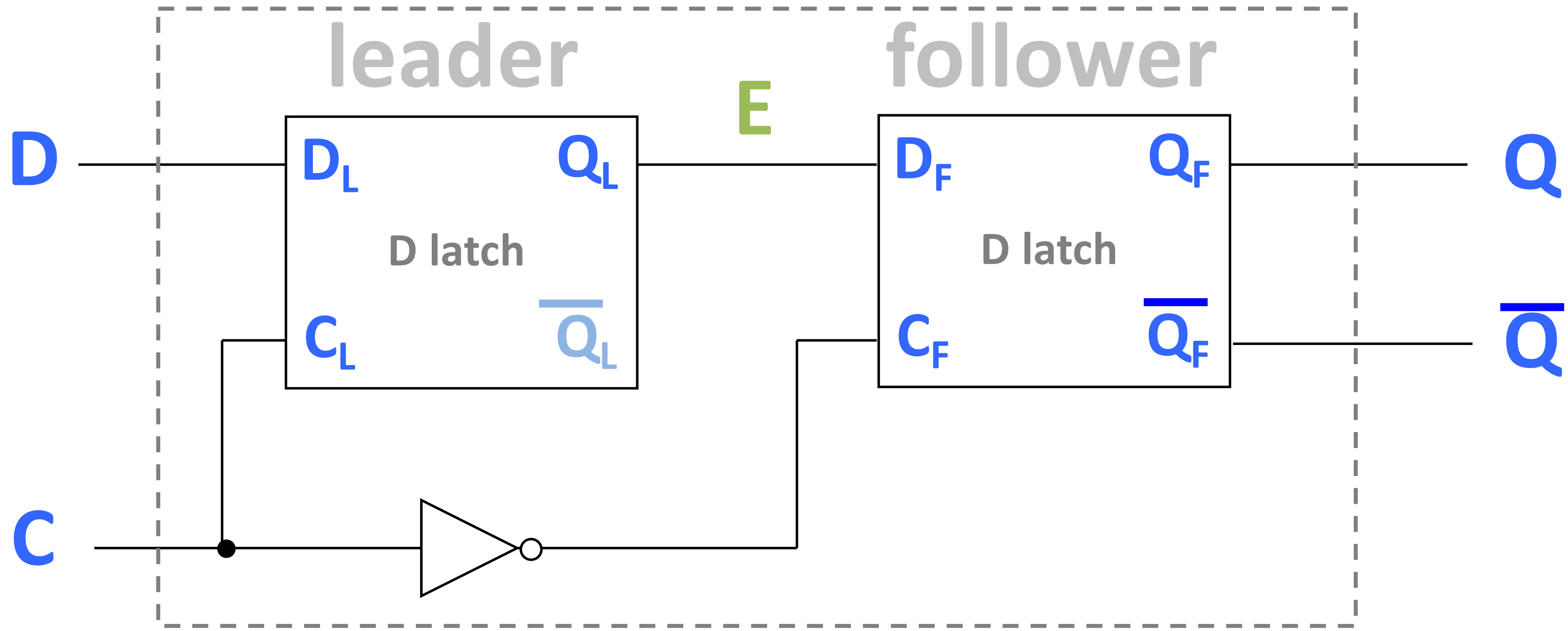
A clock controls when to update
a sequential logic element's state.

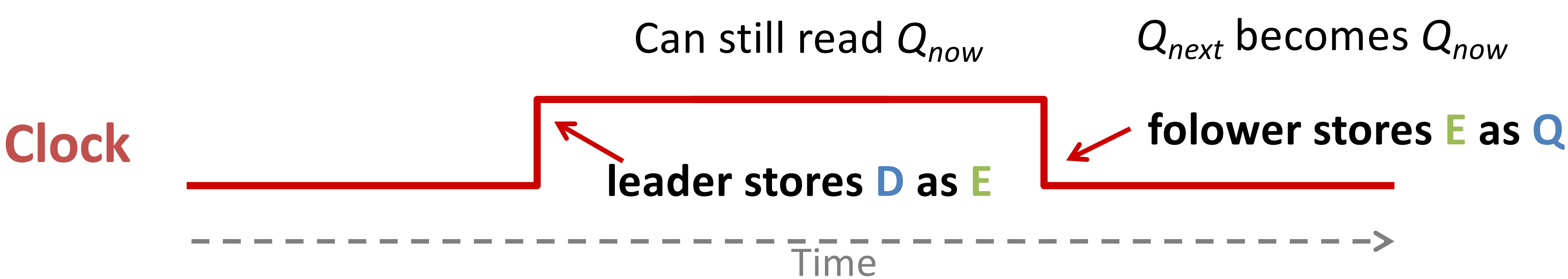
Synchronous systems

Inputs to state elements must be **valid** on active clock edge.



D flip-flop with falling-edge trigger





Time matters!

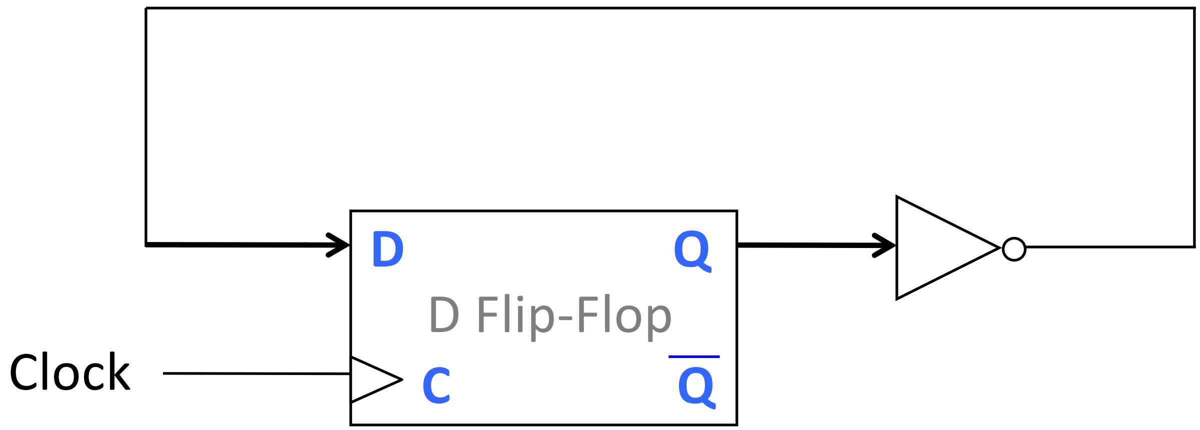
D

C

E

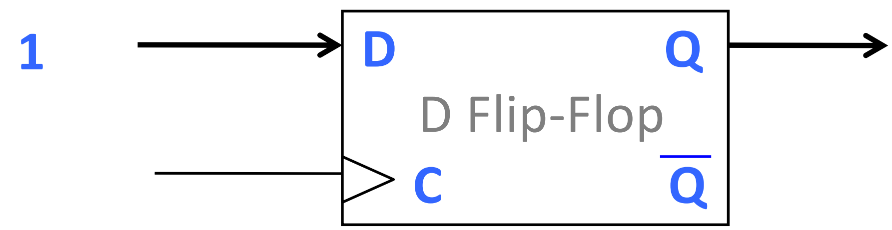
Q

Reading and writing in the same cycle



Assume Q is initially 0.

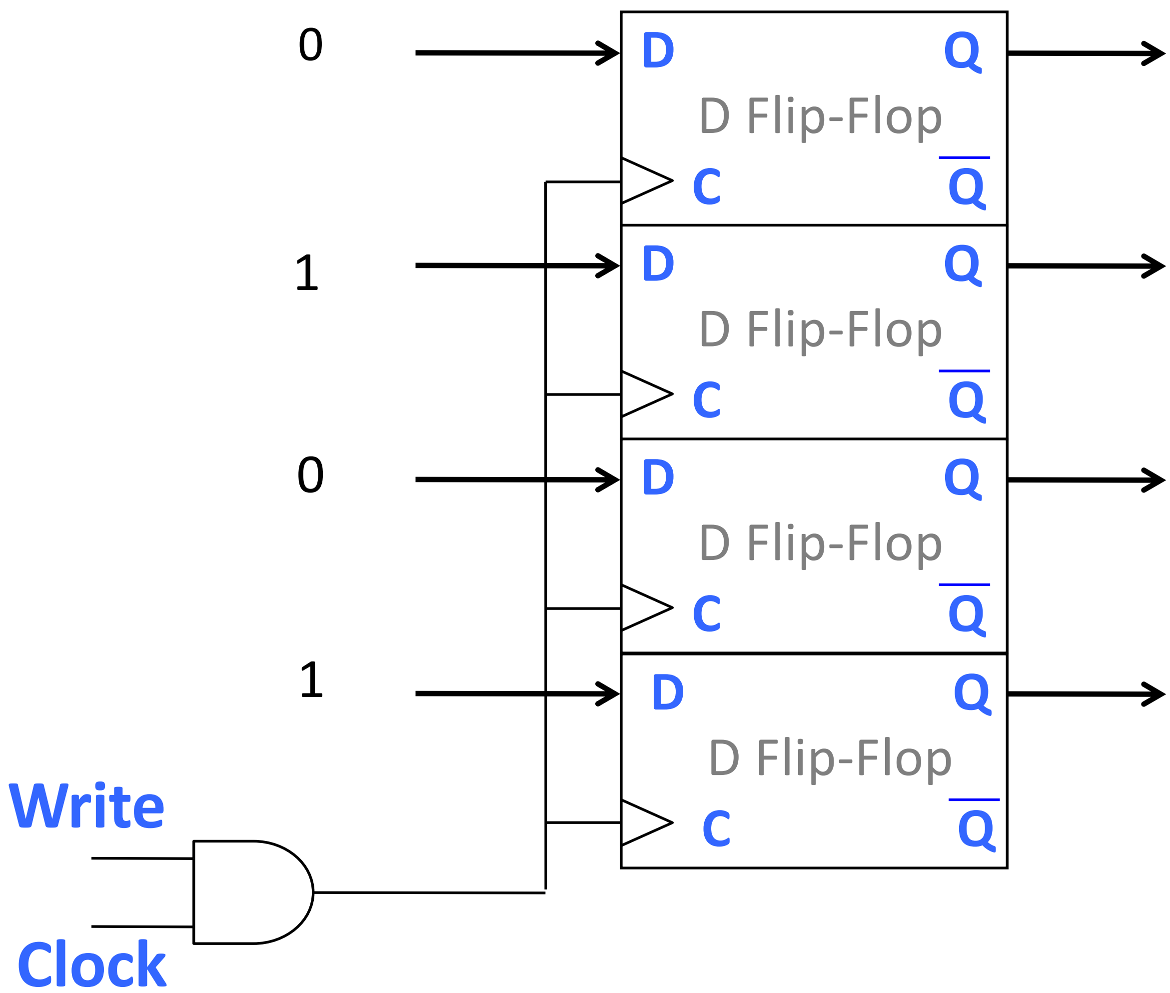
D flip-flop = one bit of storage



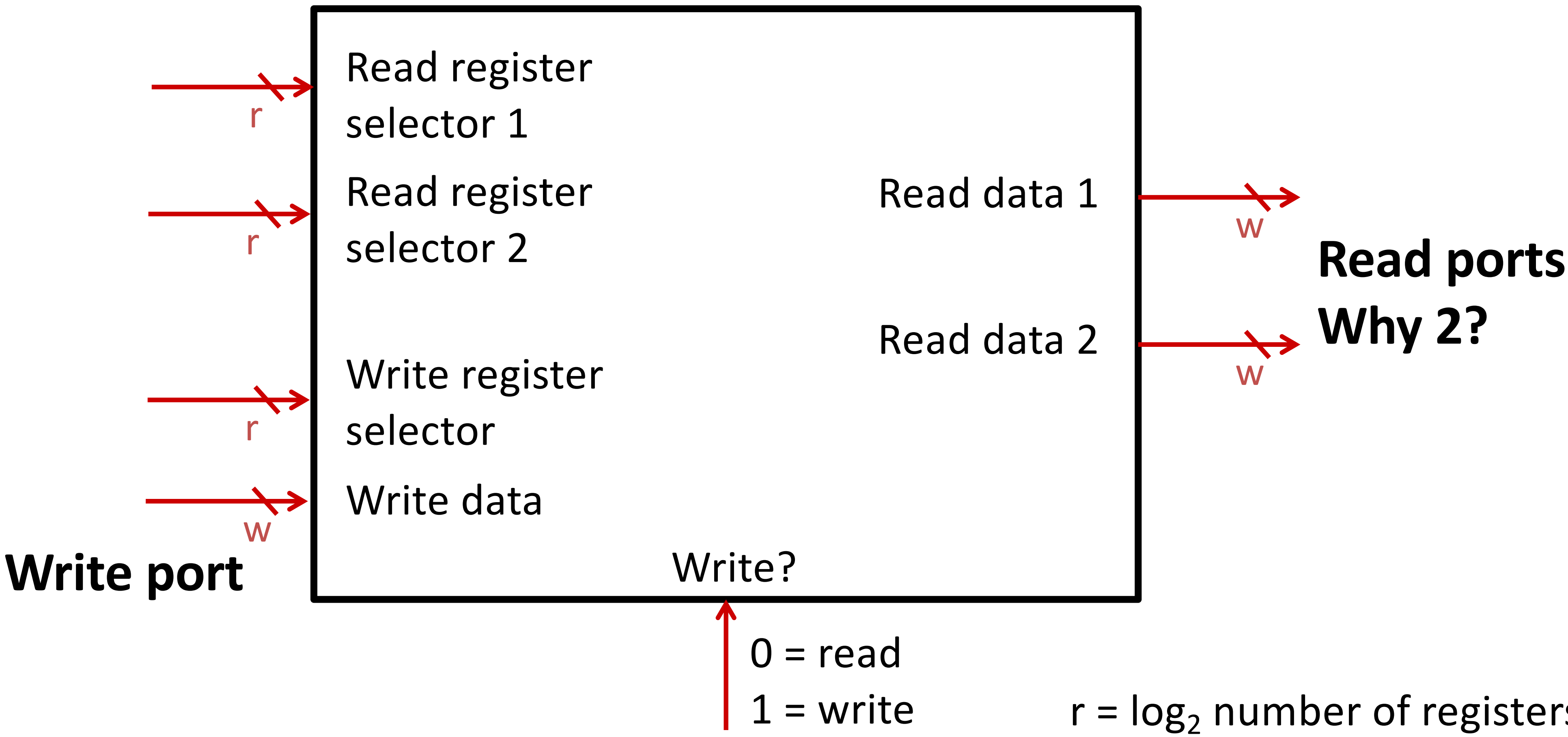
*Half a byte!

A 1-nybble* register

(a 4-bit hardware storage cell)



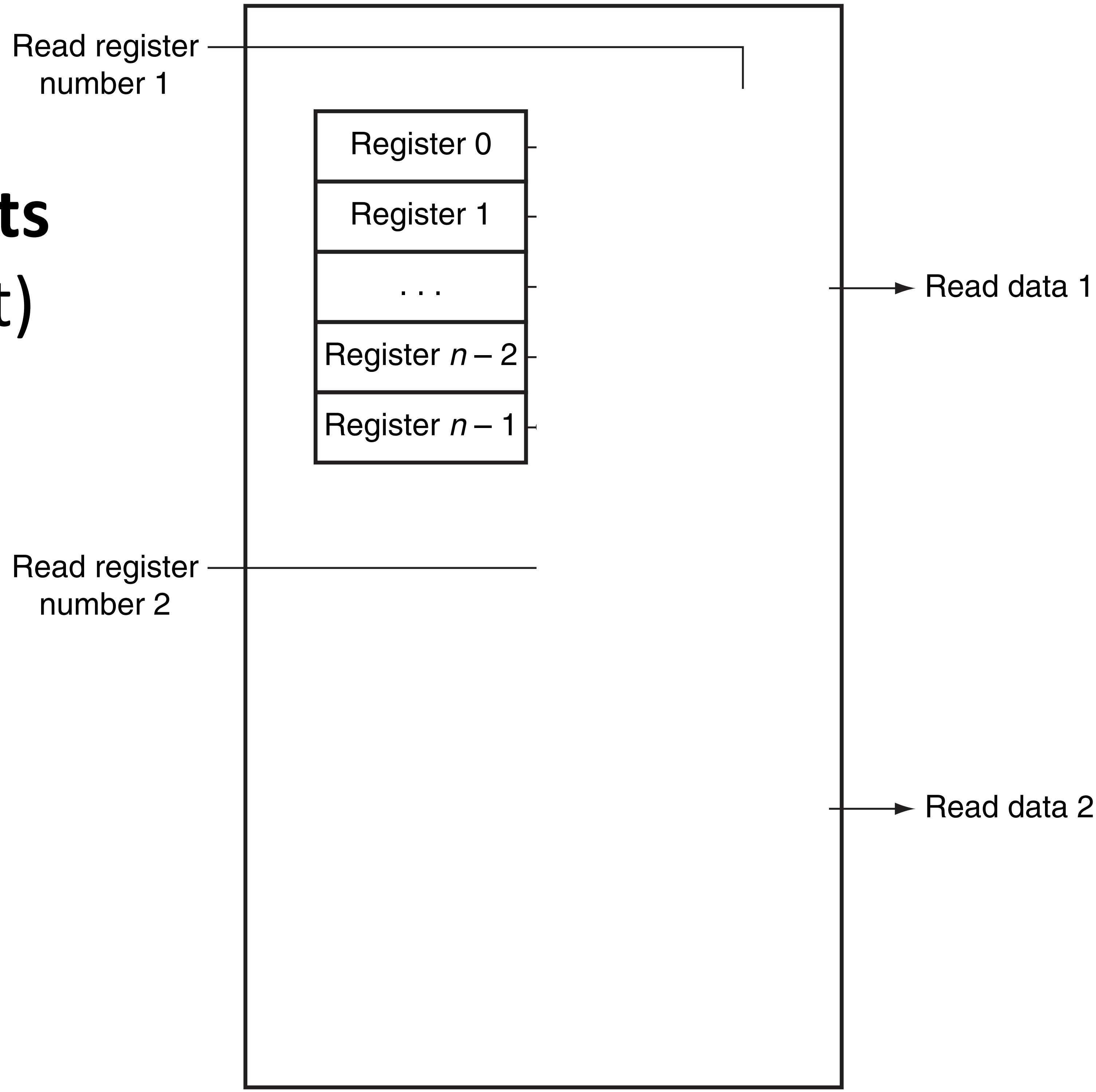
Register file



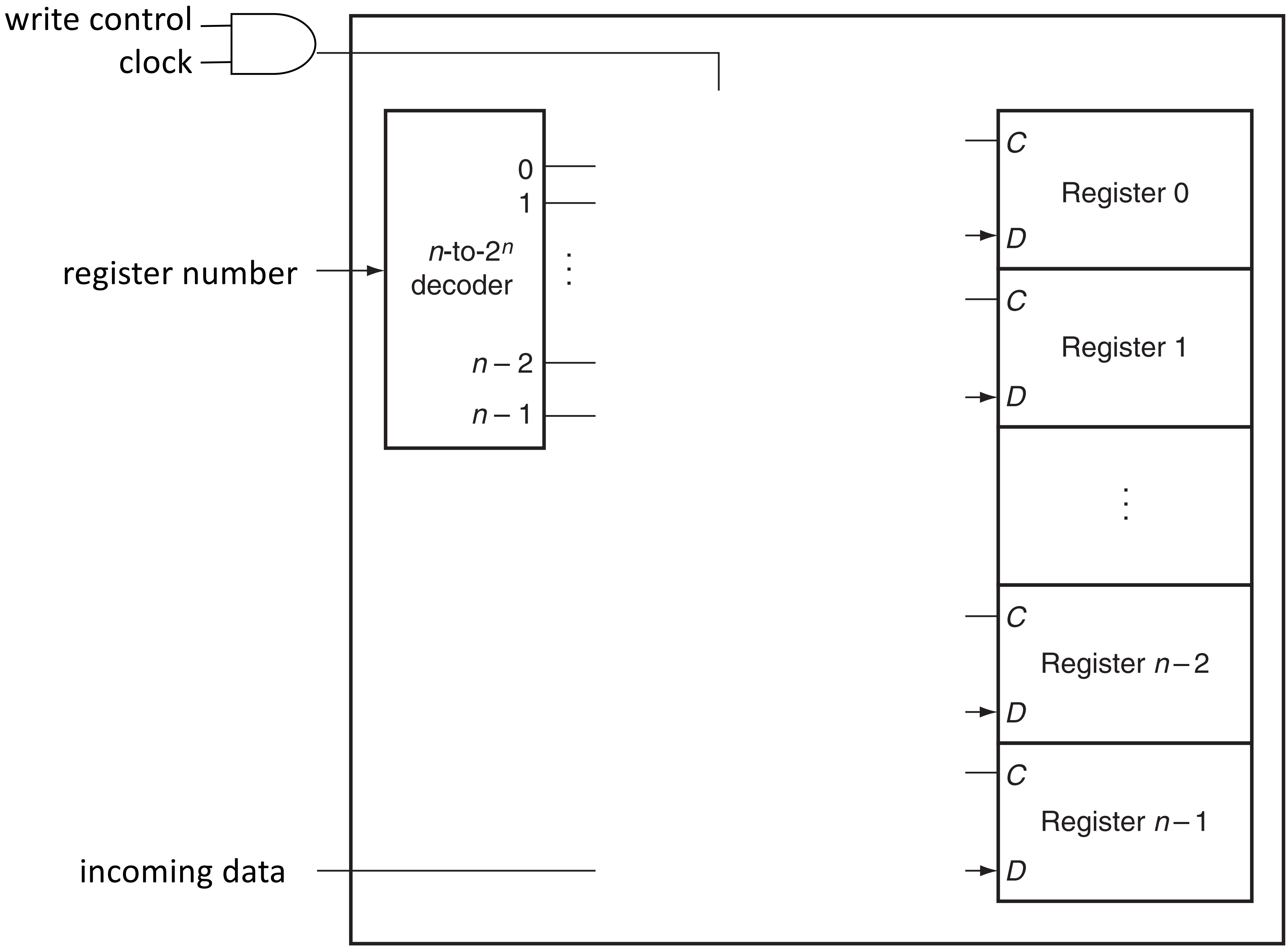
r = $\log_2$ number of registers
w = bits in word

Array of registers, with register selectors, write/read control, input port for writing data, output ports for reading data.

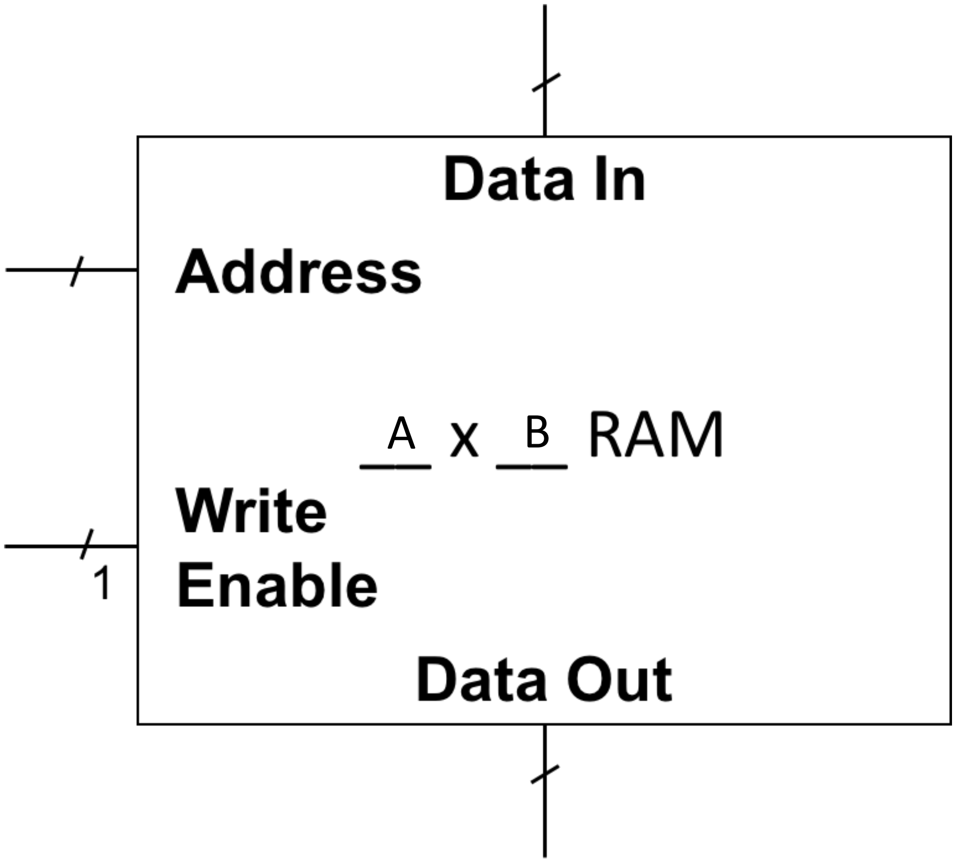
Read ports
(data out)



Write port (data in)



RAM (Random Access Memory)



Similar to register file, except…

16 x 4 RAM