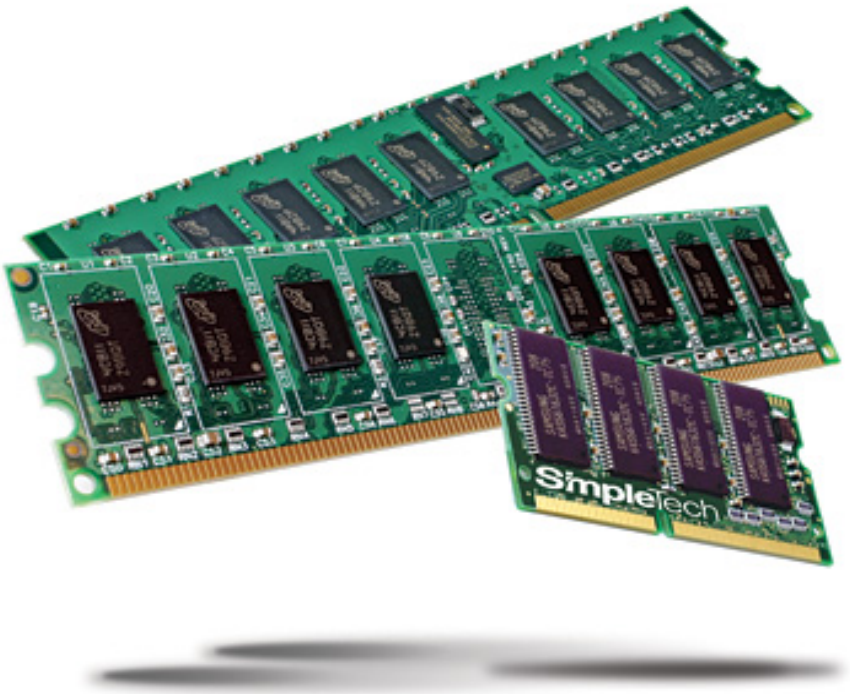
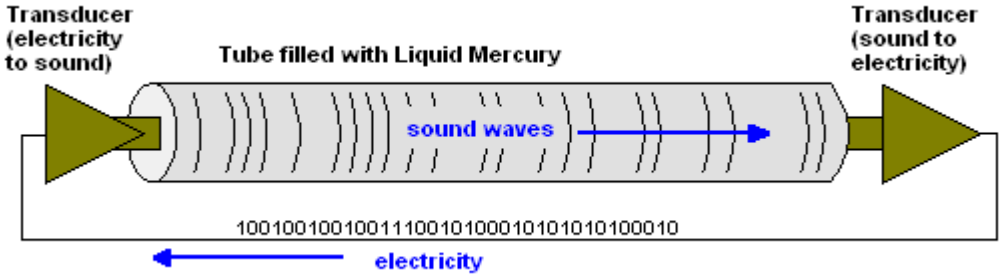
Logic Design:

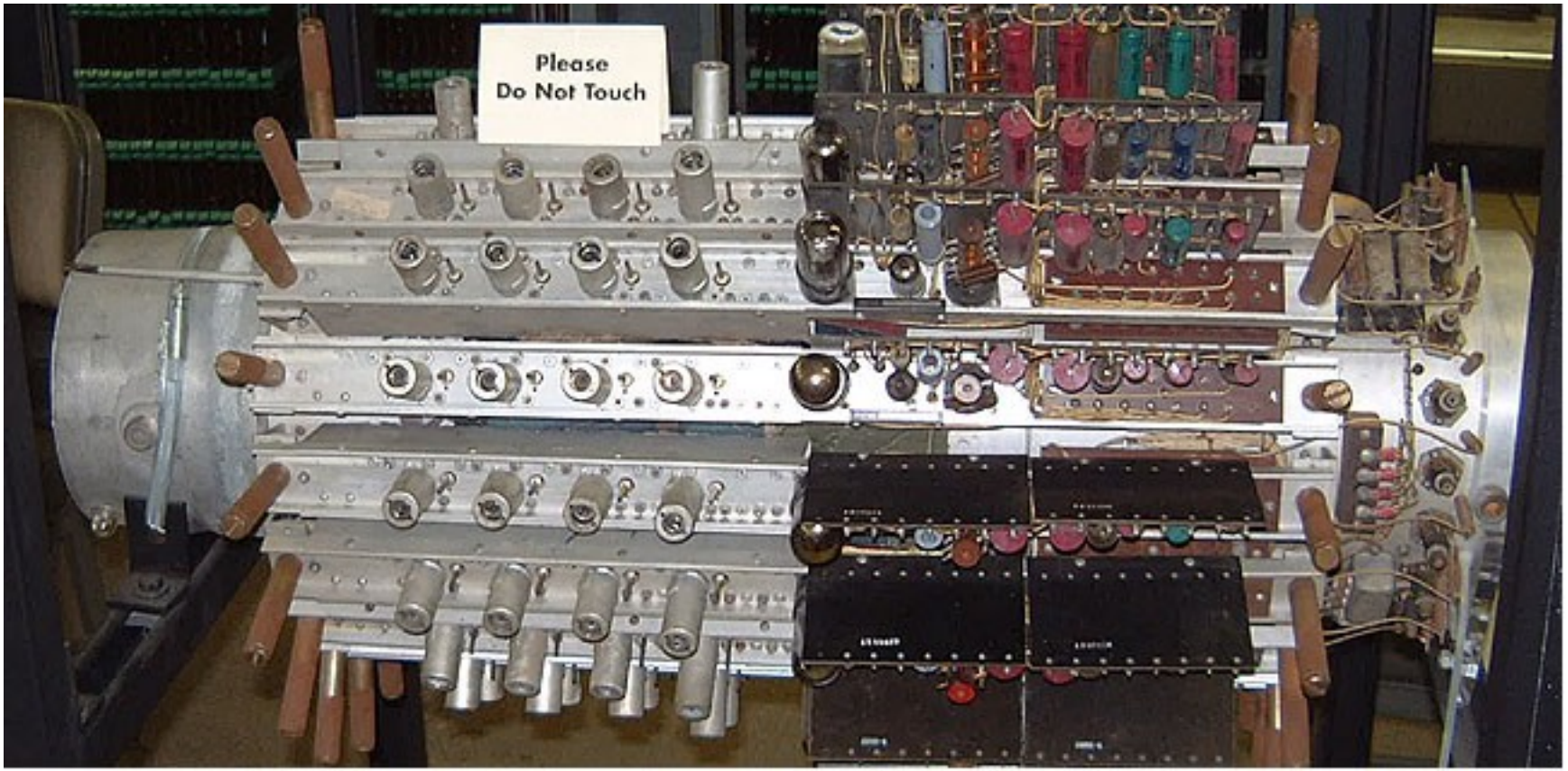
Implementing **Memory**

Memory: an organism's ability to **store**, **retain** and **recall** information

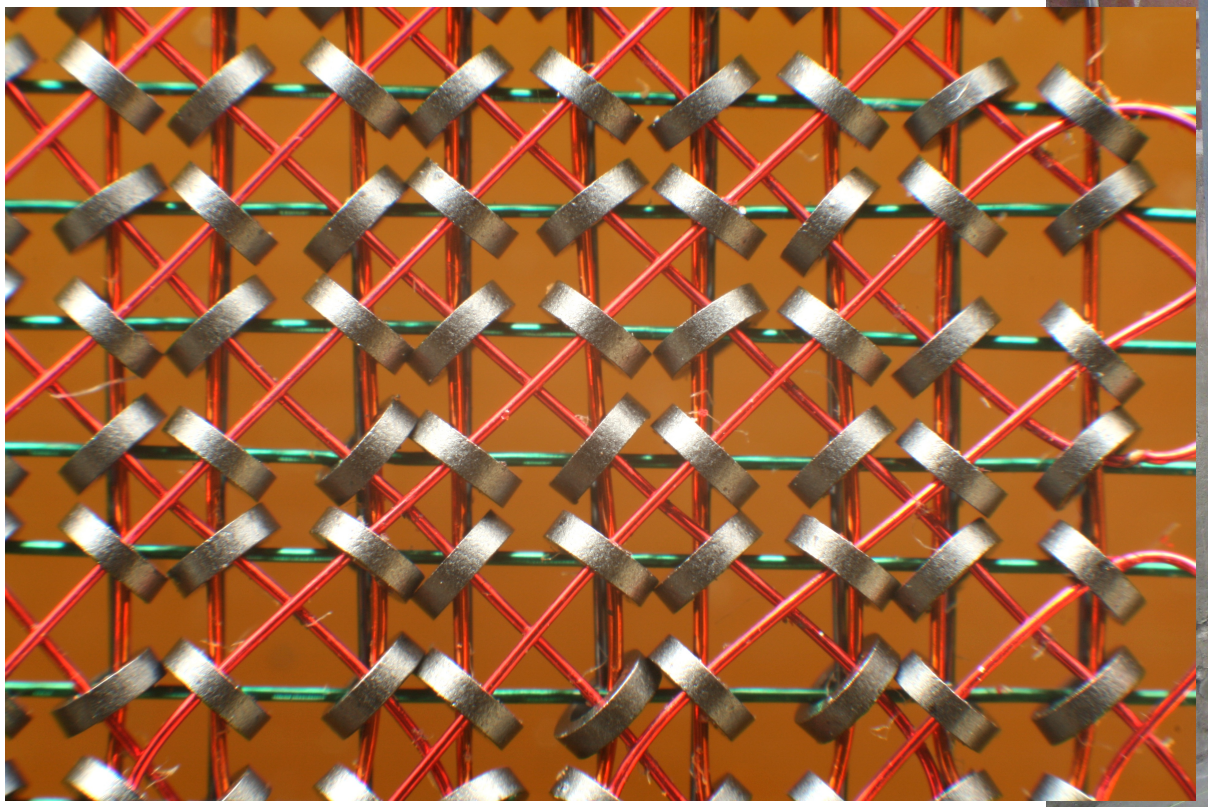


Delay Line memory

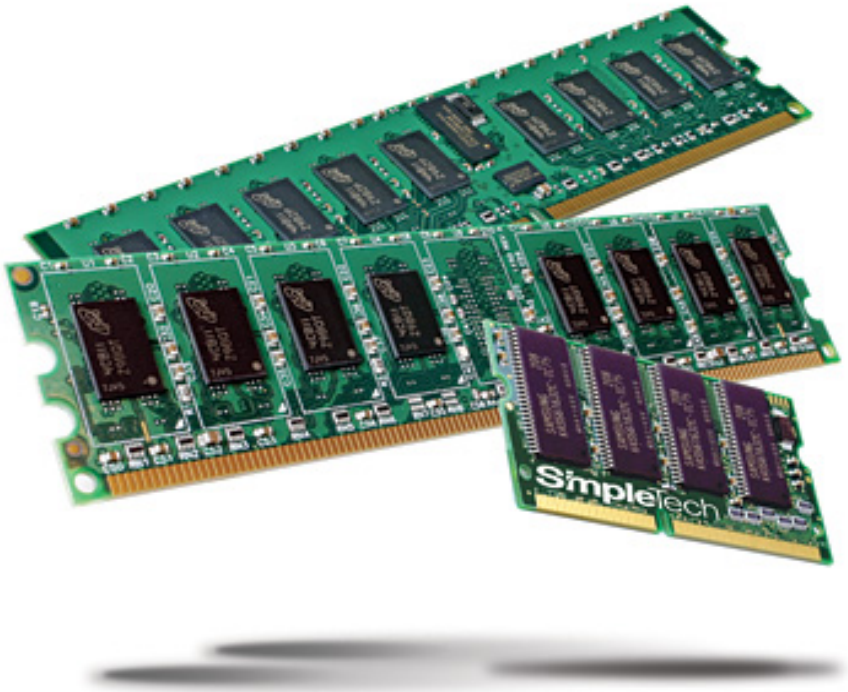




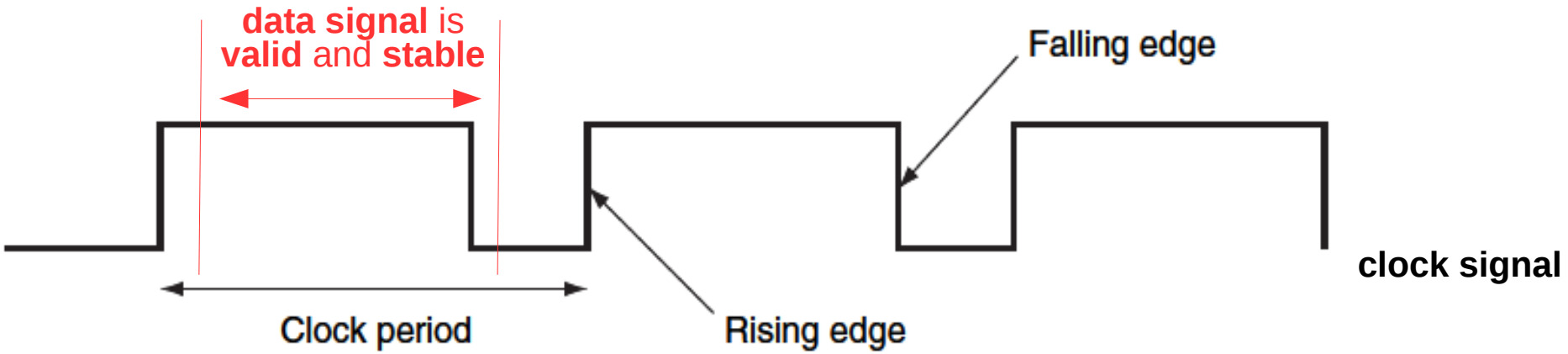
Magnetic Core memory



Modern memory bank



Clock Signal



Clocking **methodology**:
when data is **valid** and **stable** relative to **clock**

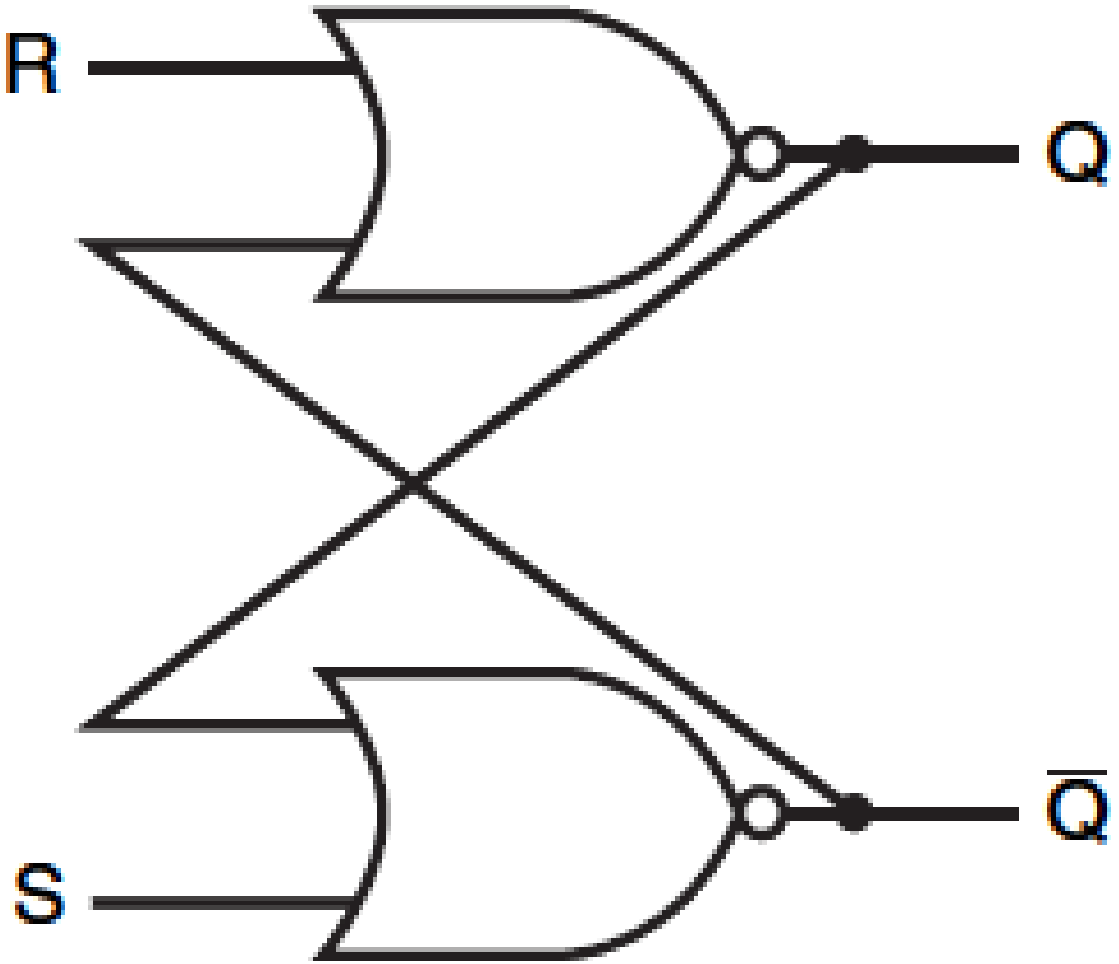
Edge-triggered clocking:
all **state changes** occur on a **clock edge**

State Elements: “memory”

State Element – Combinational Logic – State Element
(function/computation)

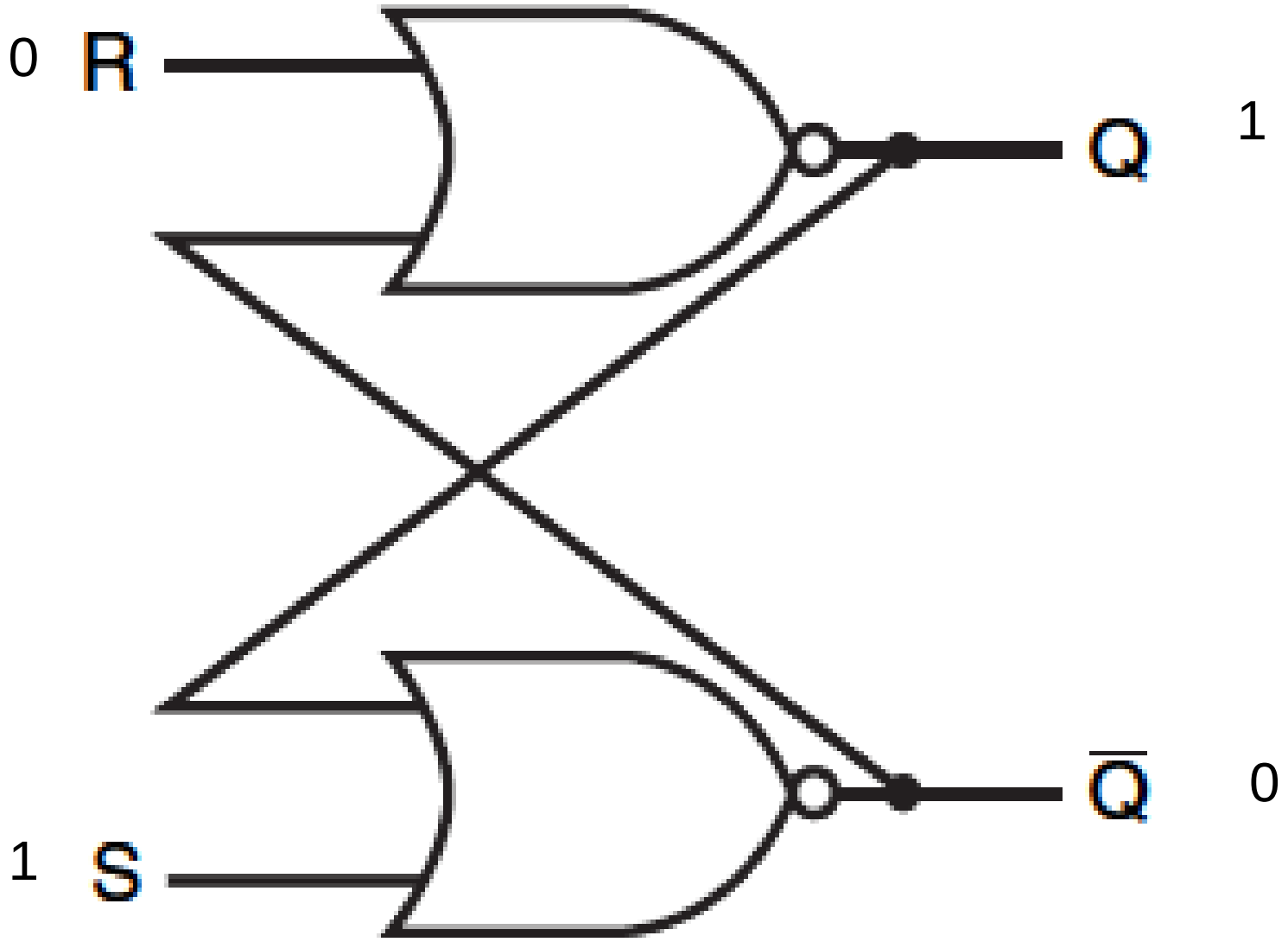

Latch: store value

S-R Latch (unclocked): store value

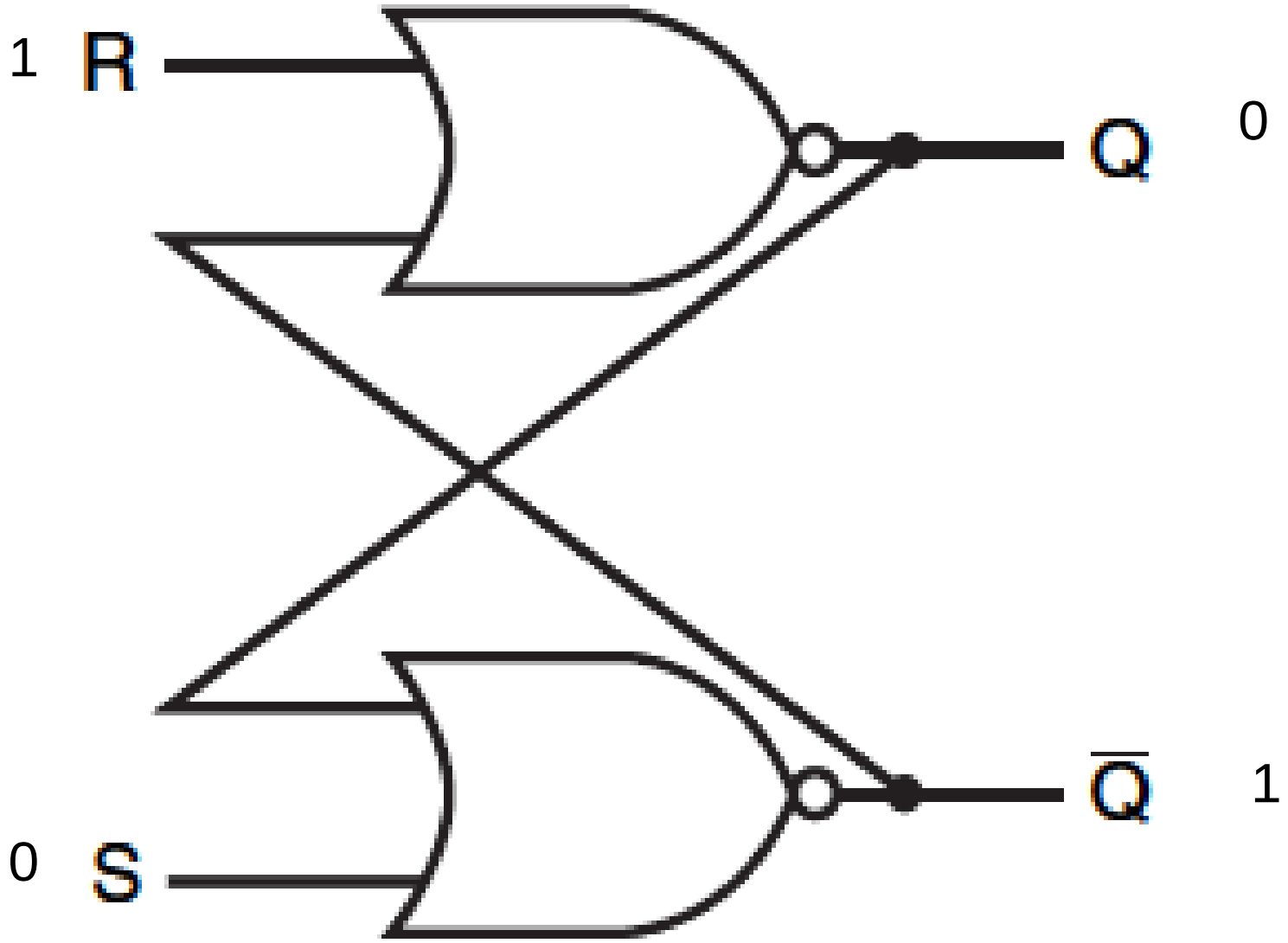


http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/16-flipflops/10-srff/srff.html

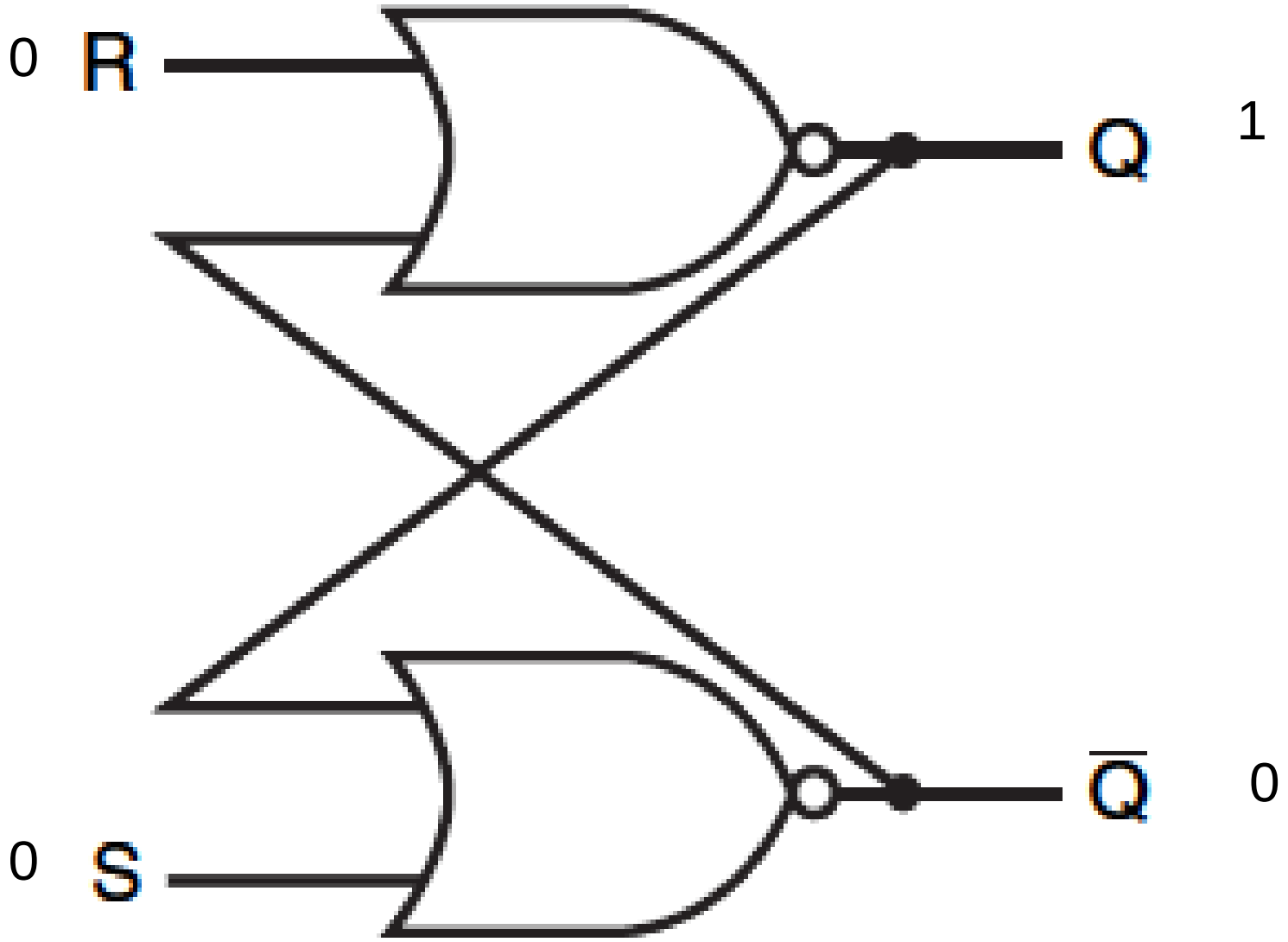
S-R Latch - set



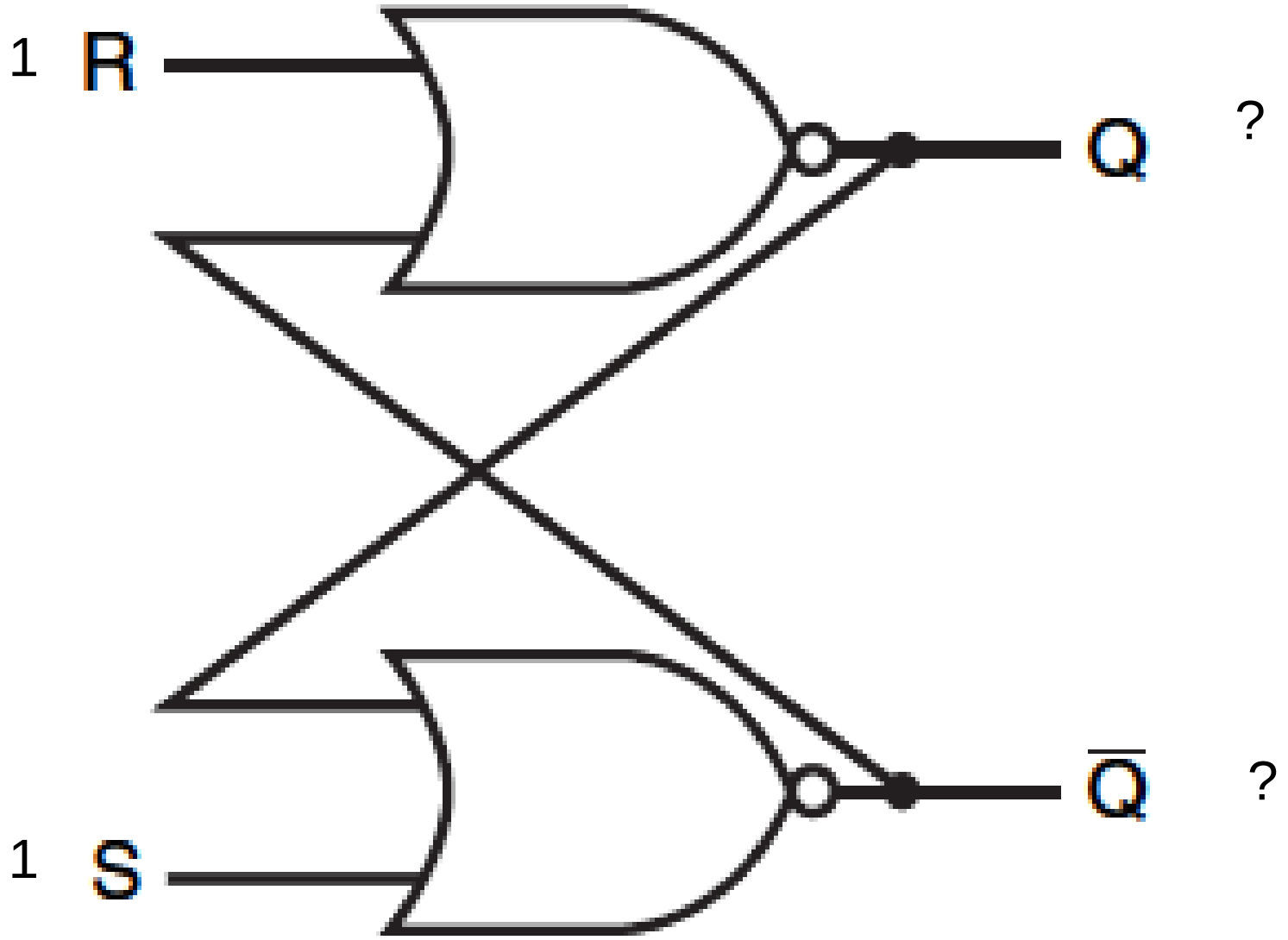
S-R Latch - reset



S-R Latch - remember



S-R Latch - meta-stable



S-R Latch

circuit →
Q = not(R or not(Q)); not(Q) = not(S or Q)
→ (substitute) →
not(R or not(Q)) = not(not(S or Q))
→ (simplify) →
(not(R) and Q) = (S or Q)

S	**R**	**formula**	**Q**	**not(Q)**
0	1	0 and Q = 0 or Q → 0 = Q	0	1
1	0	1 and Q = 1 or Q → Q = 1	1	0
0	0	1 and Q = 0 or Q → Q = Q	**Q**	**not(Q) ← memory**
1	1	0 and Q = 1 or Q → 0 = 1	?	?

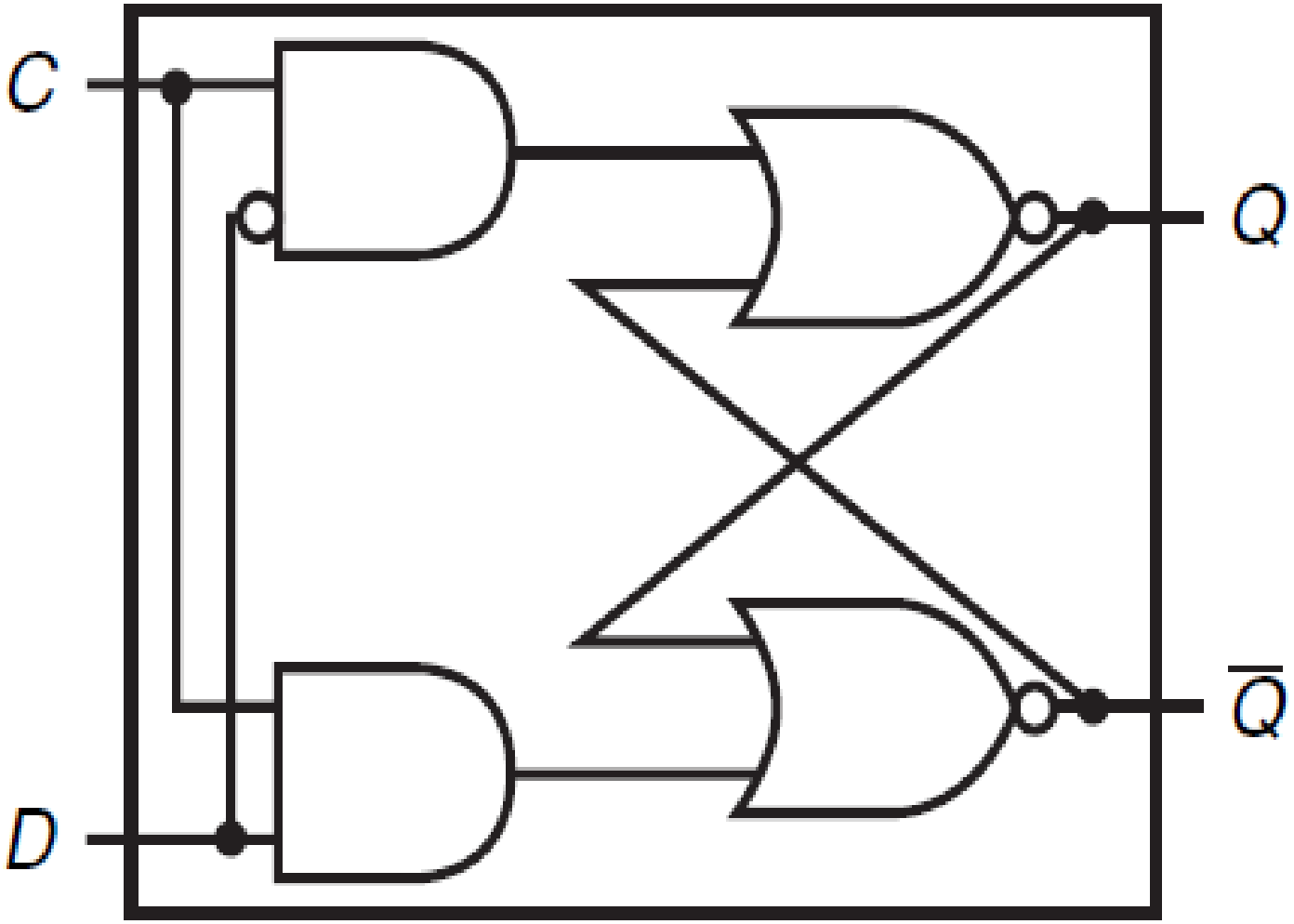
Latch:

change **while** clock asserted

Flip-Flop:

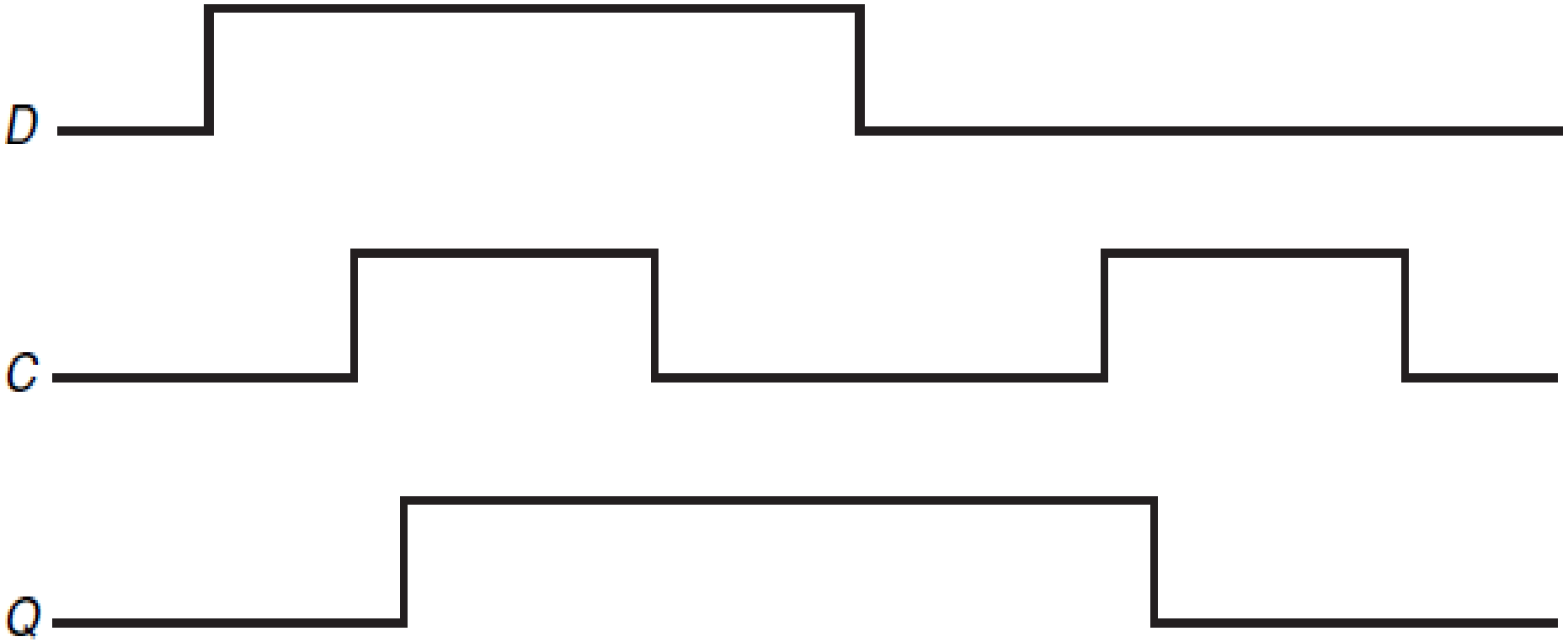
change **on** clock **edge**

D latch



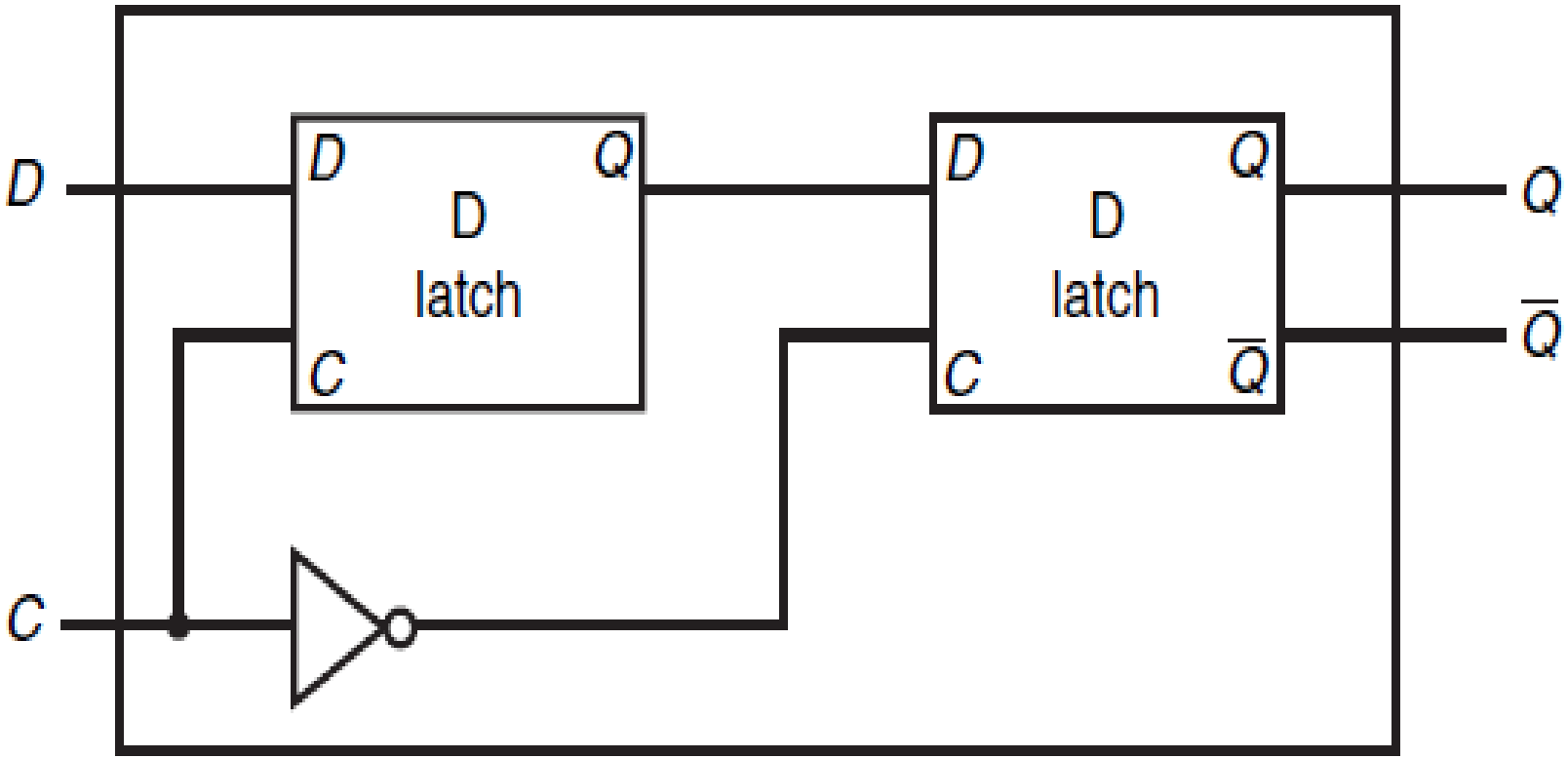
http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/16-flipflops/20-dlatch/dlatch.html

D latch operation



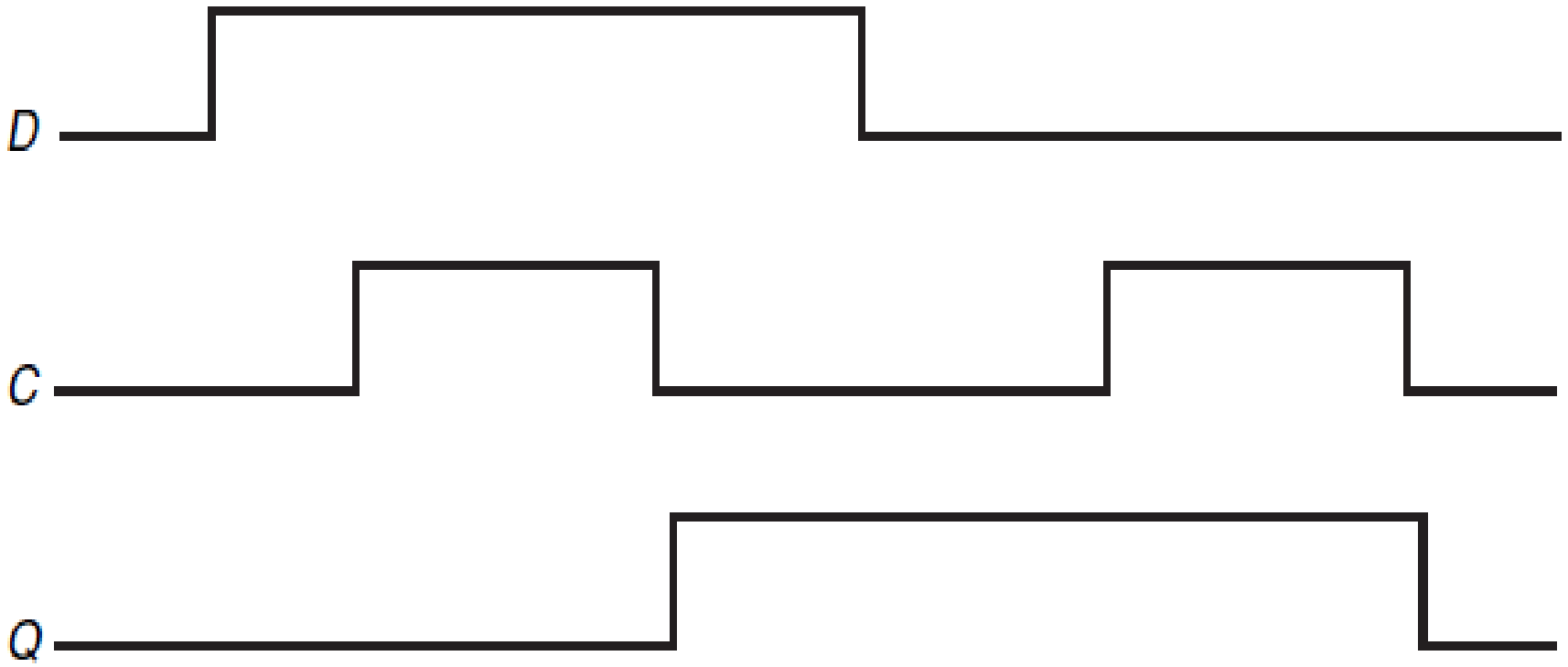
What if D changes during C asserted?

D flip-flop
with falling-edge trigger

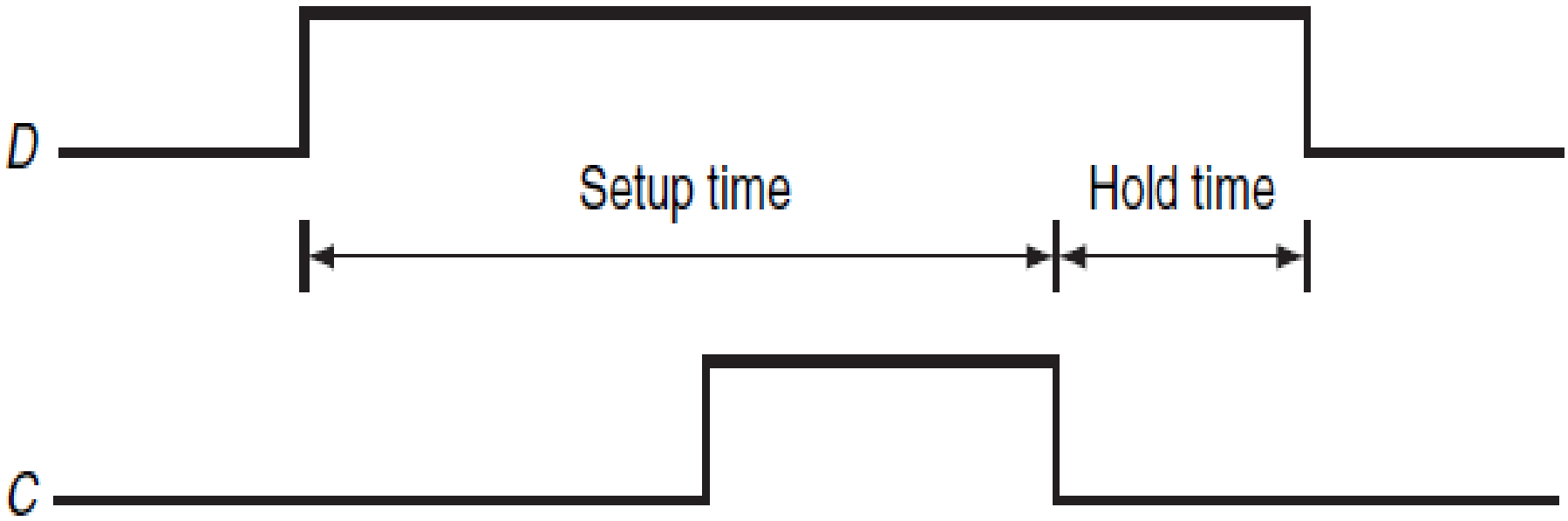


http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/16-flipflops/20-dlatch/dff.html

D flip-flop with falling-edge trigger operation

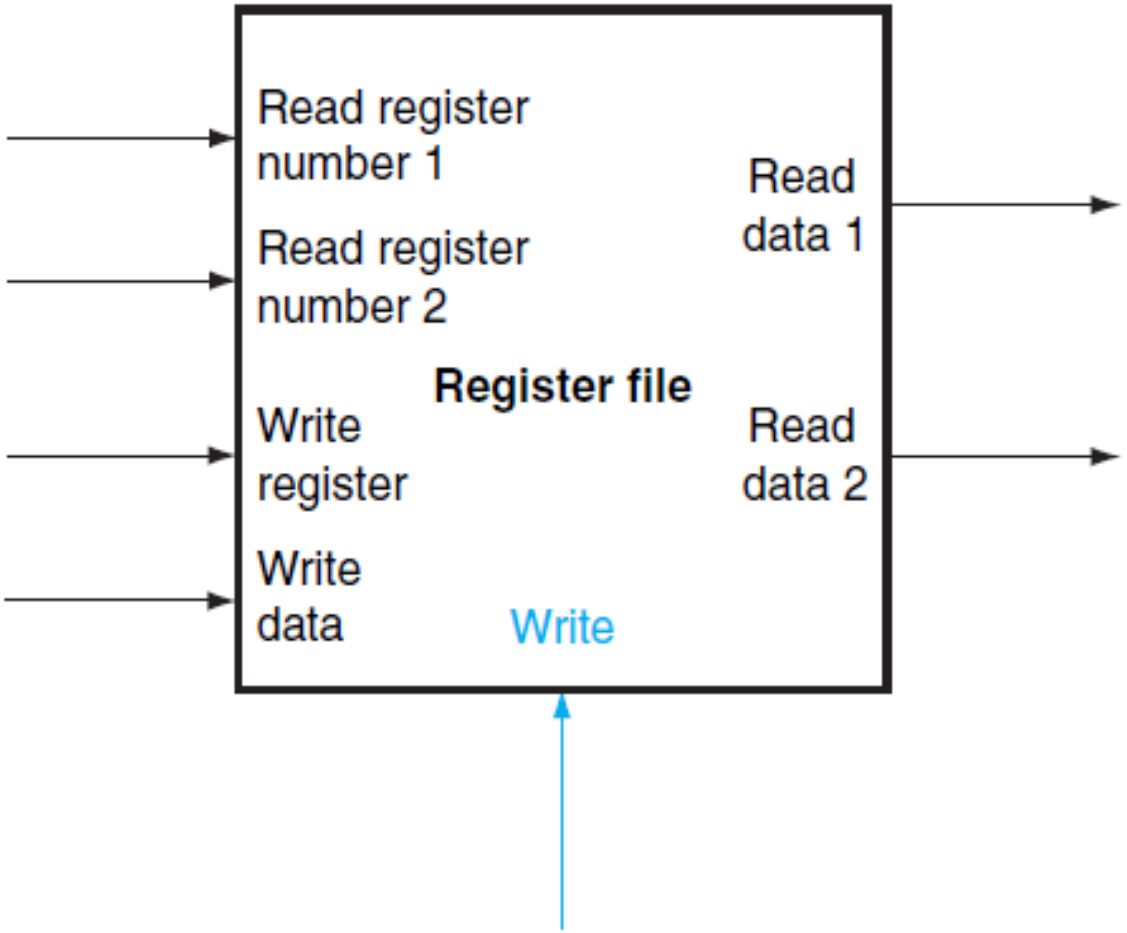


D flip-flop timing constraints



Register Files

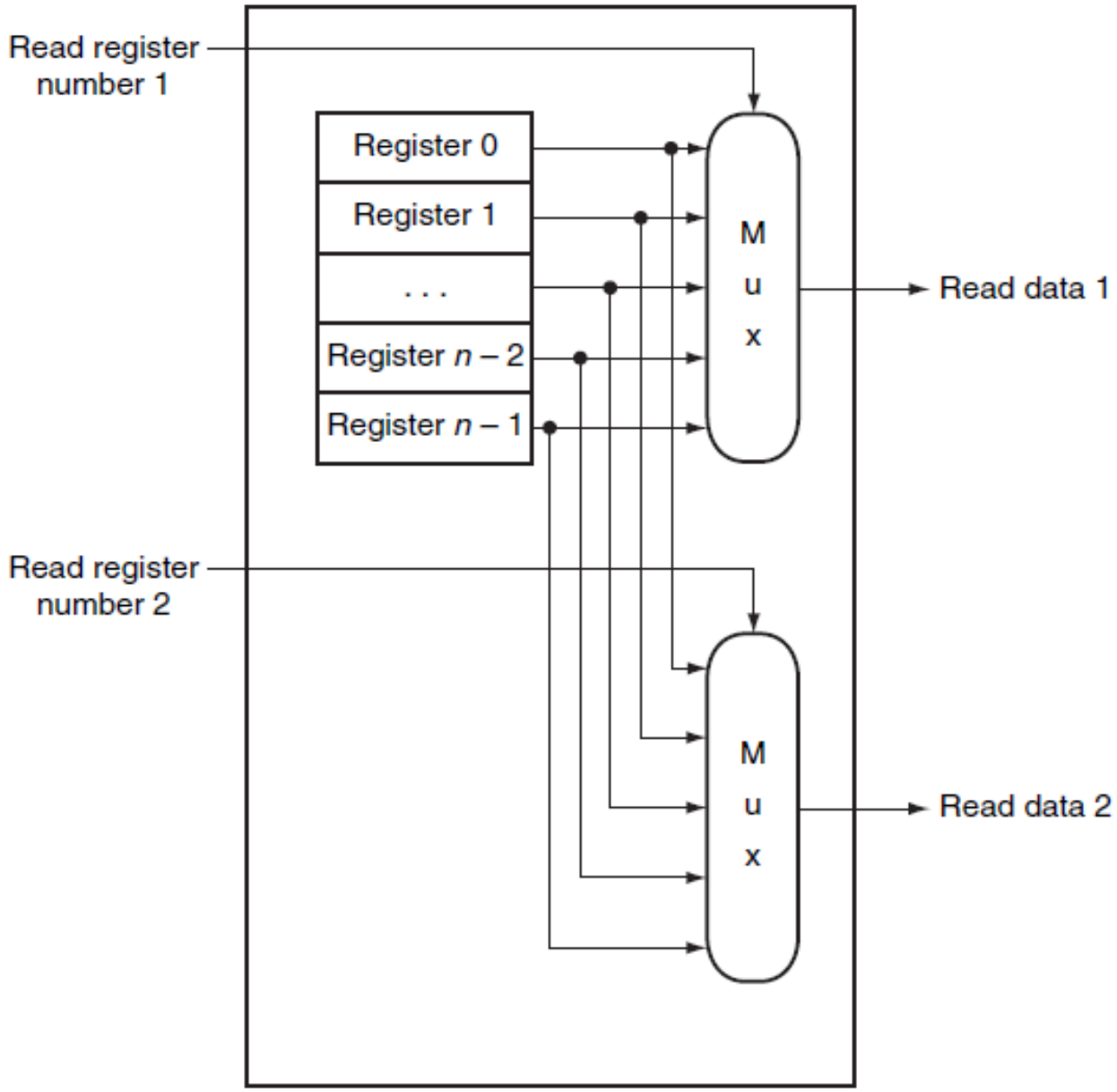
N-bit registers



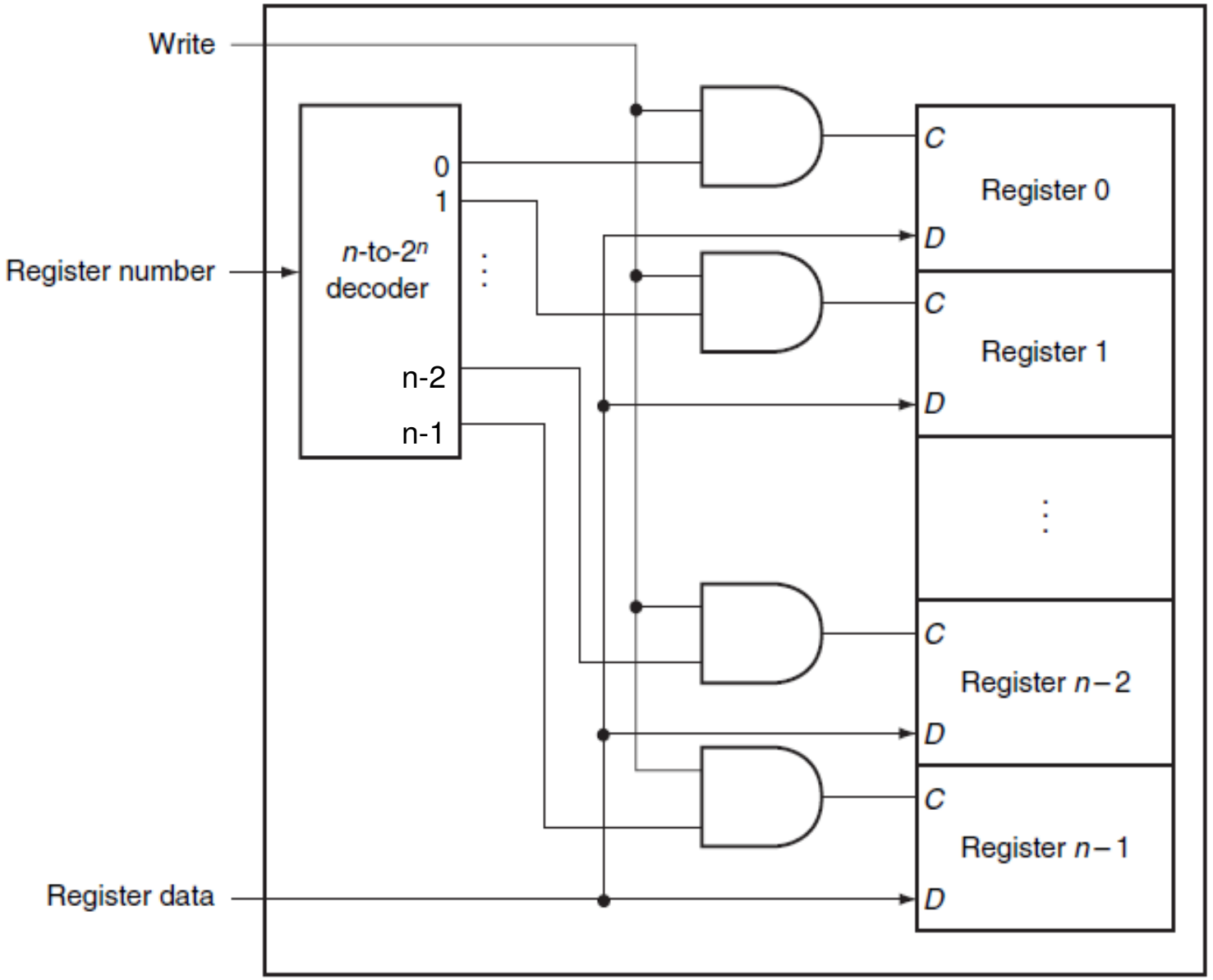
and clock ...

Register Files: read

width?



Register Files: write

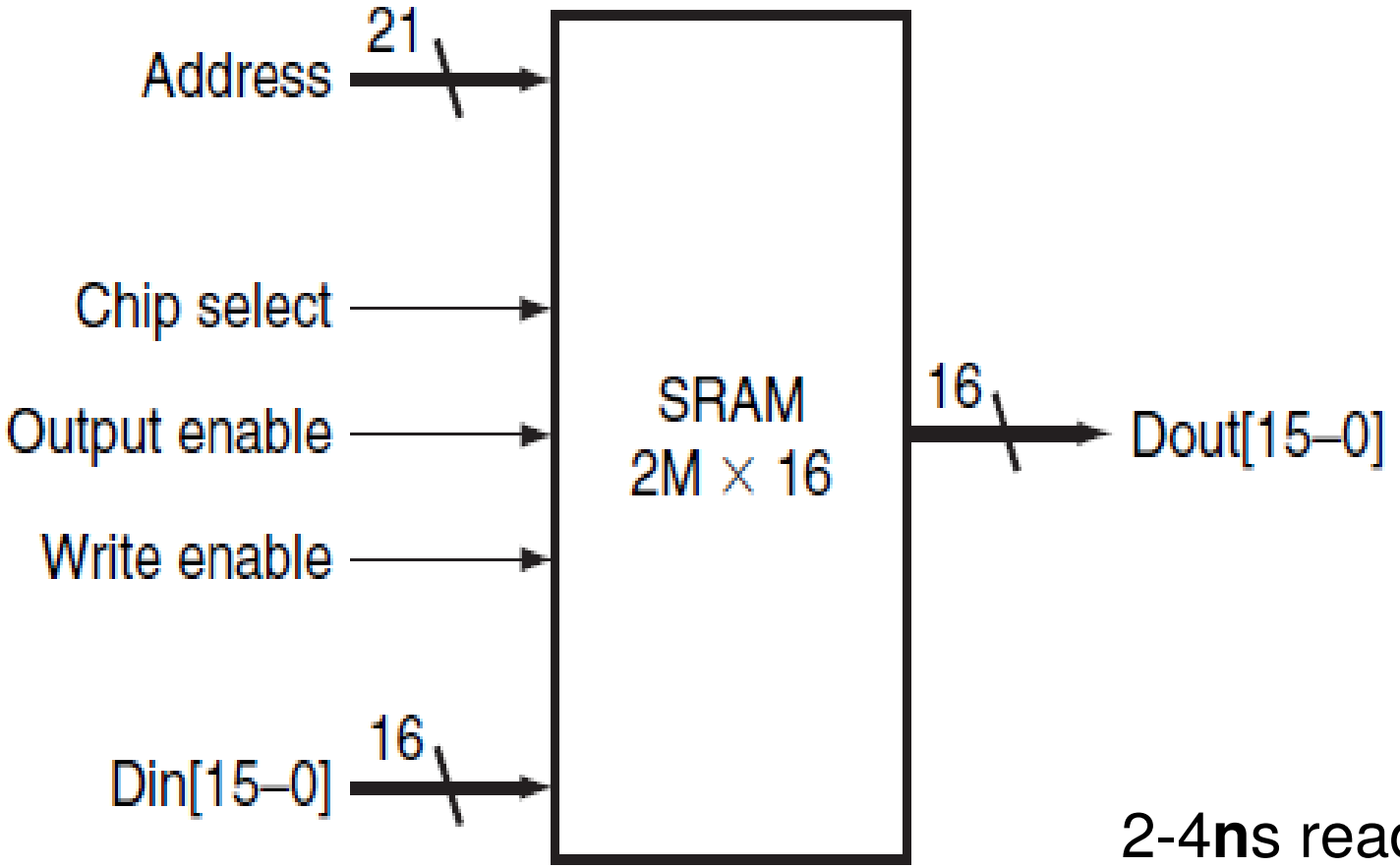


and clock ...

Large Memory:

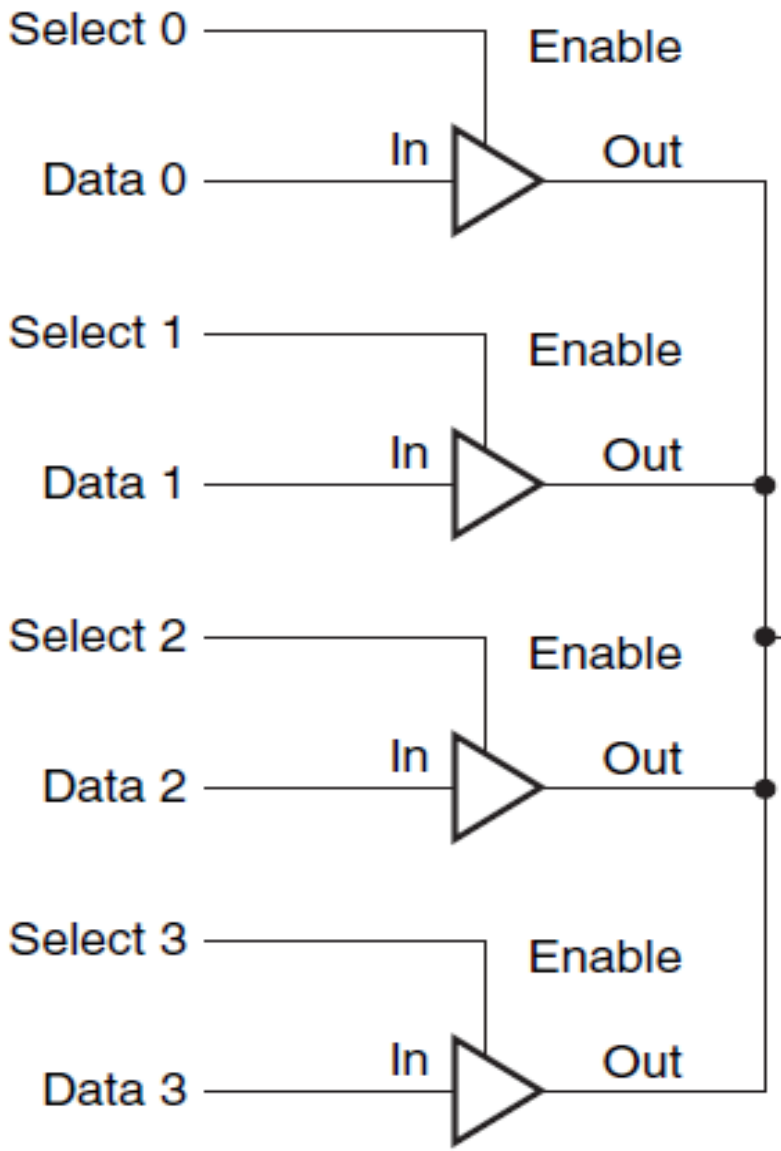
Static Random Access Memory (SRAM)

Dynamic Random Access Memory (DRAM)



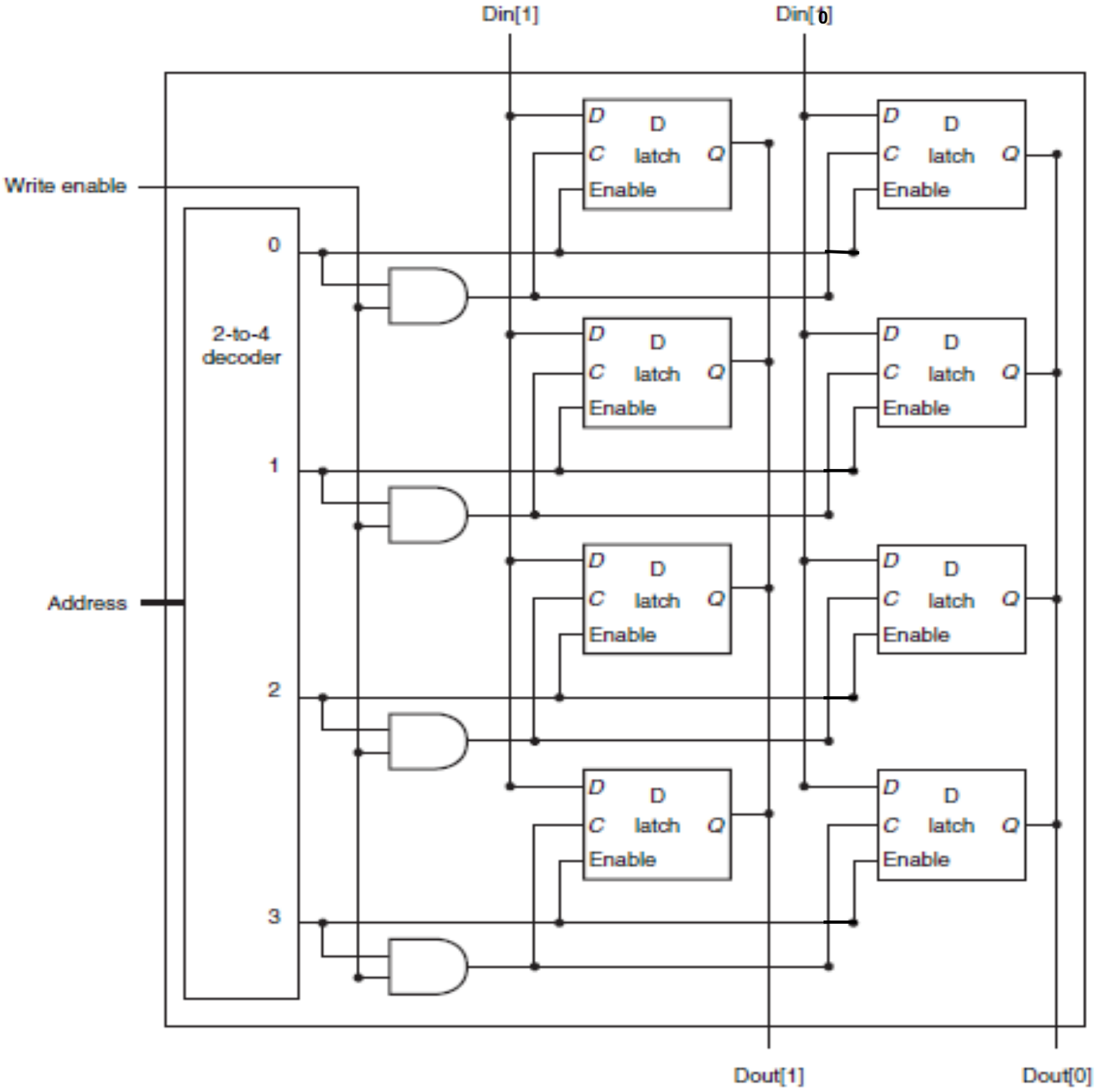
2-4**n**s read time

Three-state buffers (~multiplex)



- assered
- de-asserted
- high-impedance

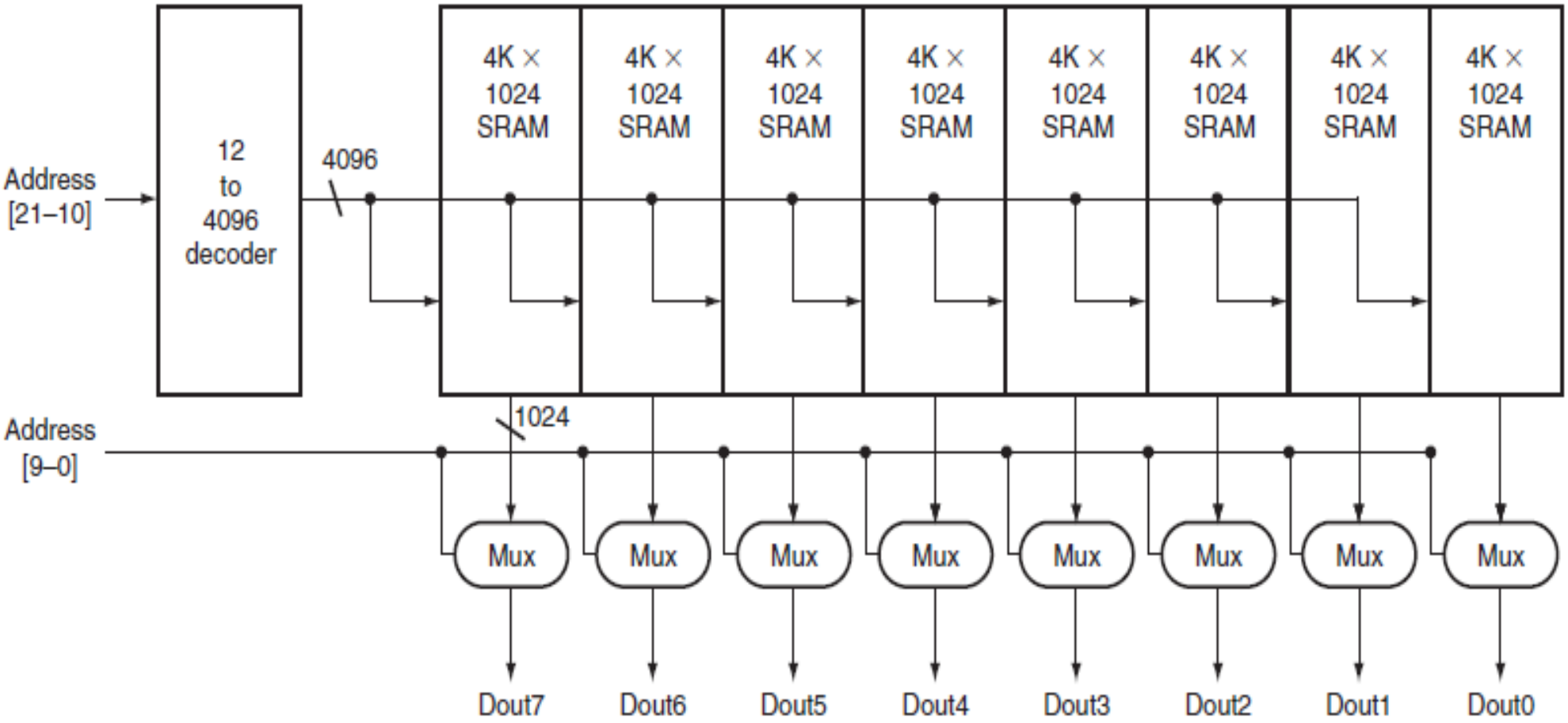
4x2 SRAM (without output enable, chip select)



No large multiplexor at output!

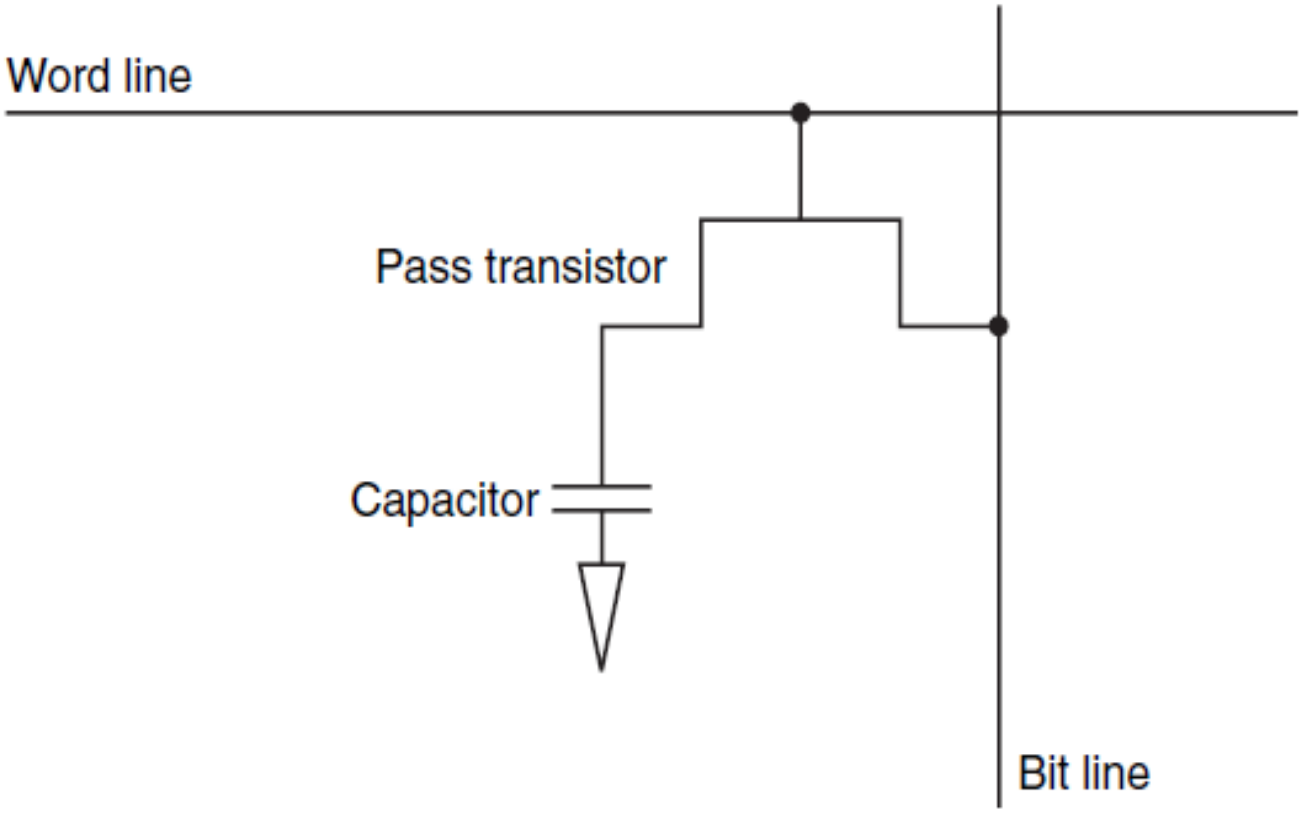
SRAM **array**

(2-step process
to remove need for large decoder)



DRAM: capacitor (need refresh)

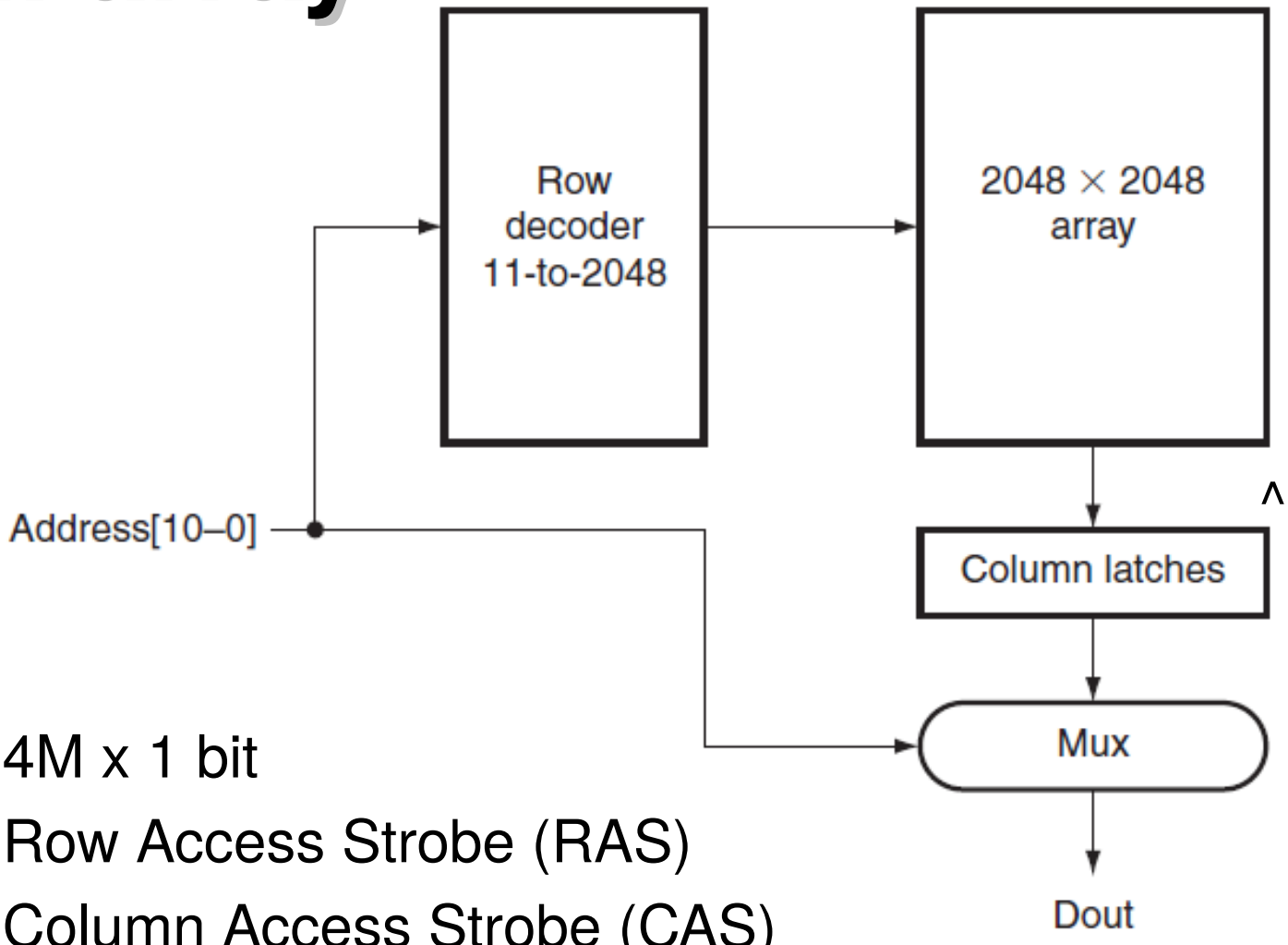
1 transistor per bit → denser and cheaper



read/write/refresh (rate = **ms**)

refresh by read and write-back

DRAM array



4M x 1 bit

Row Access Strobe (RAS)

Column Access Strobe (CAS)

45-65**n**s read time

Synchronous RAM (**S**SRAM and **S**DRAM):

“burst” of data from a series of sequential addresses (use clock)

Error Detection Code

1 bit **parity**: **detect** 1 bit error

= distance-2 code

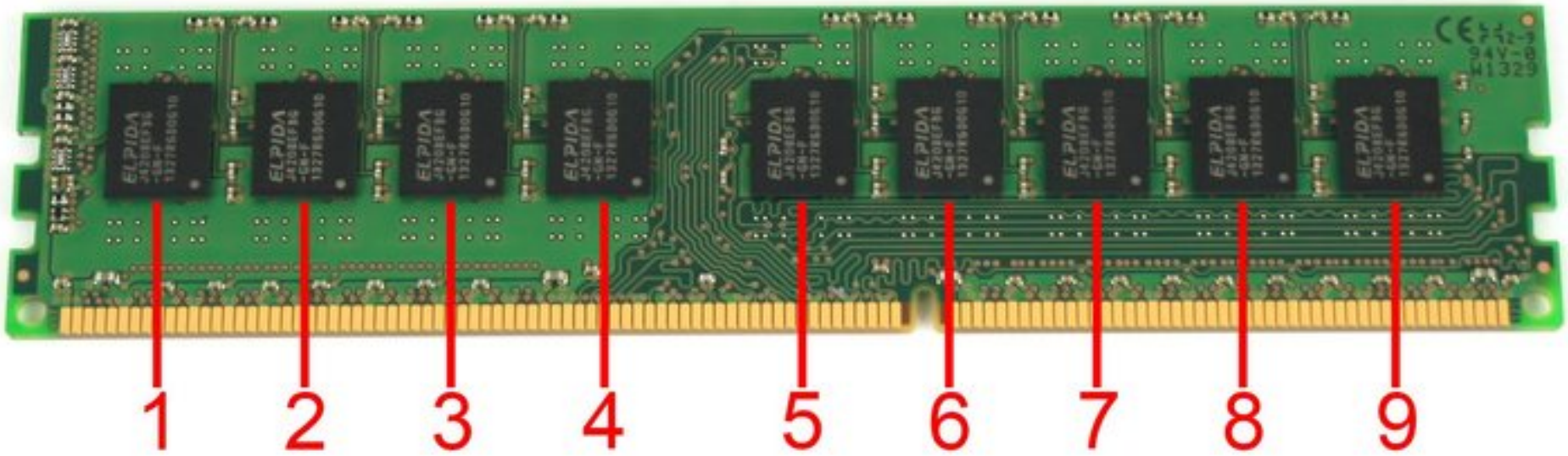
Error Correction Codes (ECC)

Data Word	Code bits	Data	Code bits
0000	000	1000	111
0001	011	1001	100
0010	101	1010	010
0011	110	1011	001
0100	110	1100	001
0101	101	1101	010
0110	011	1110	100
0111	000	1111	111

0110 - 1 bit error: 1110, 0010, 0100, 0111

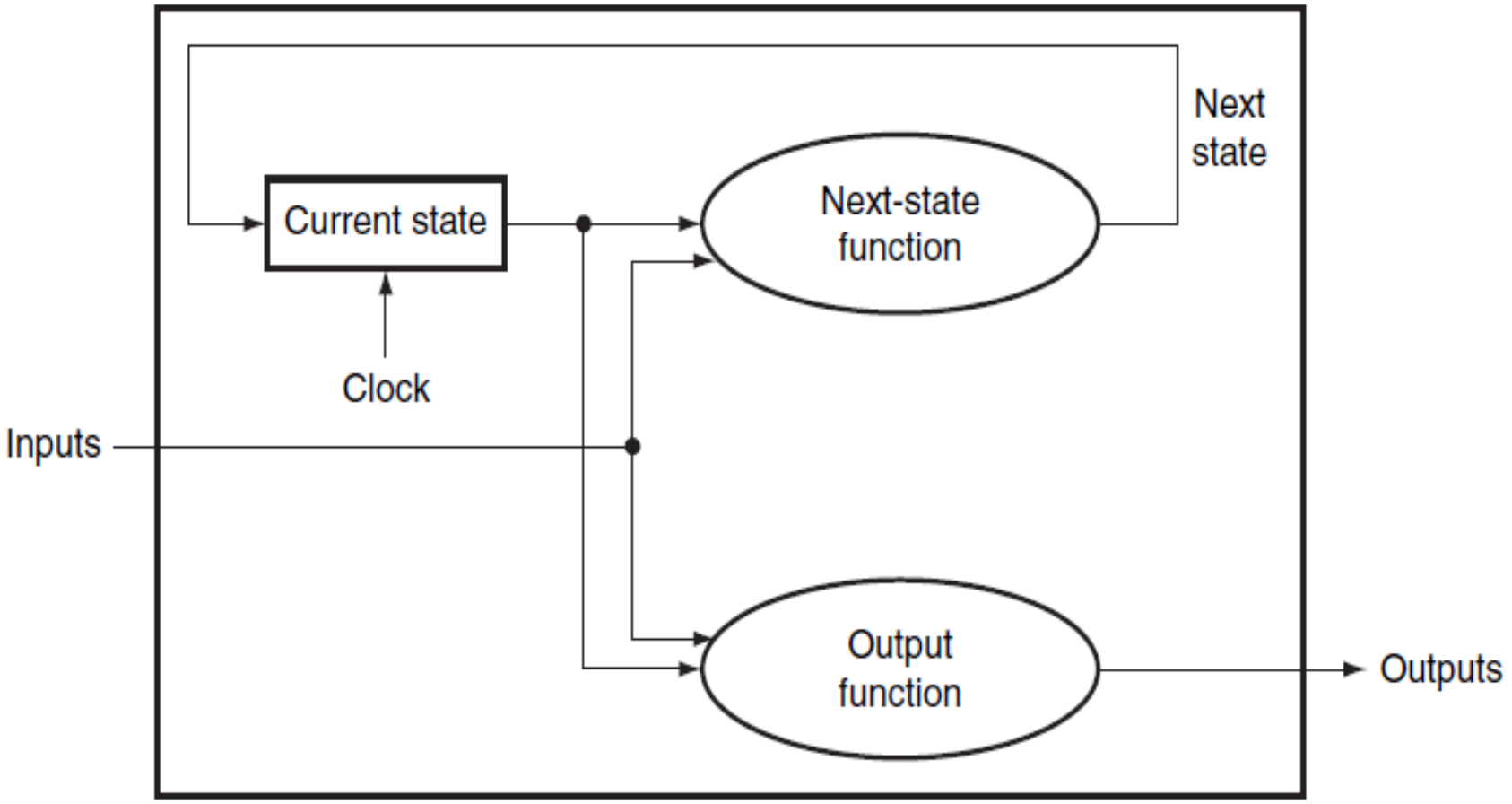
Code: 011 appears for 0110 and 0001

ECC RAM



Non-ECC RAM

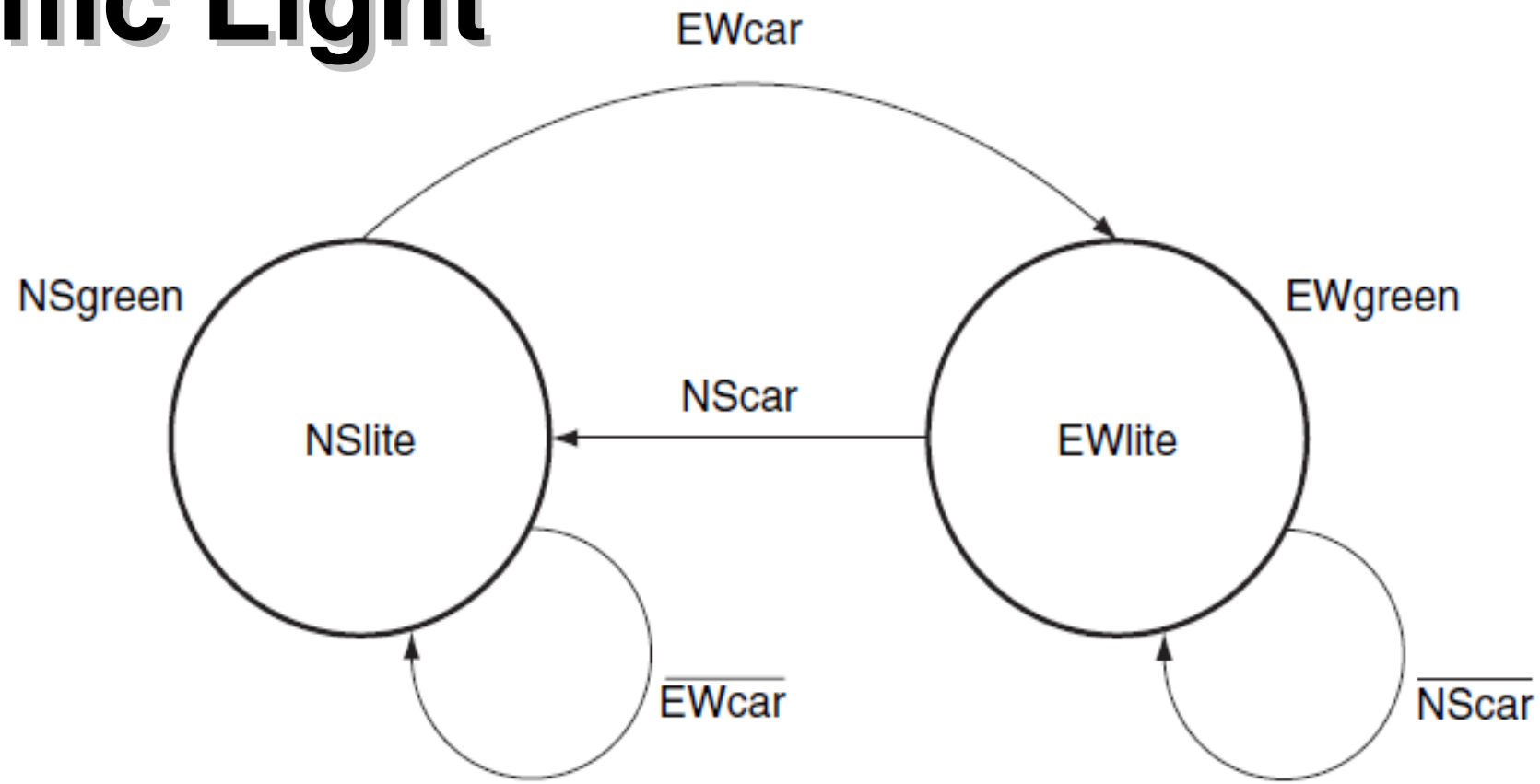
Finite State Machines



Discrete-Event (DE) vs. Discrete-Time (DT) System

Mealy vs. Moore machine

Traffic Light



encode `CurrentState (NSgreen or Ewgreen)` in 1 bit

```
NextState = (~CurrentState . EWcar) + (CurrentState.~NScar)
NSlite    = ~CurrentState
EWlite    = CurrentState
```

... add `after(delay)` ... ?

Traffic Light

	Inputs		
	NScar	EWcar	Next state
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

Initialize State Automata!

http://www.play-auto.net/2010/05/volvo-s60-brake-test-epic-fail/

Traffic Light