Latch: CC-BY Rberteig@flickr

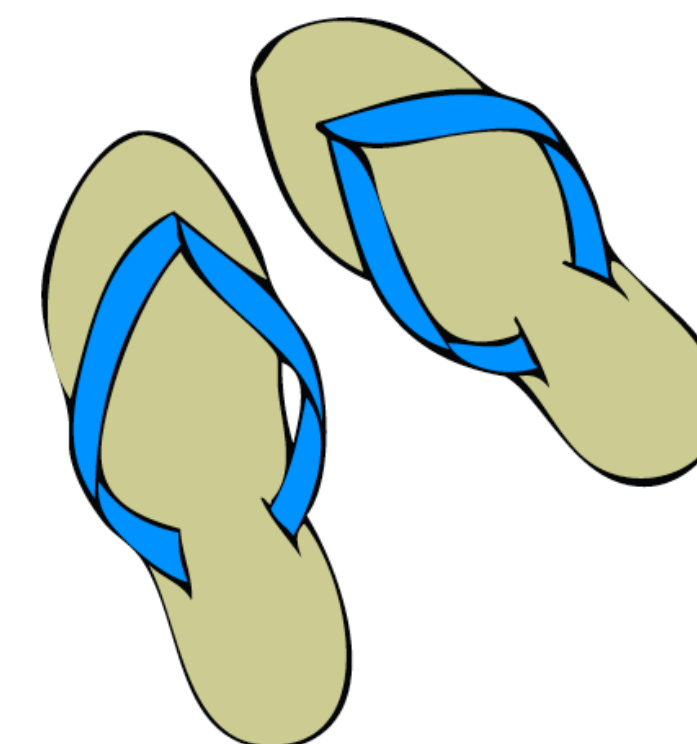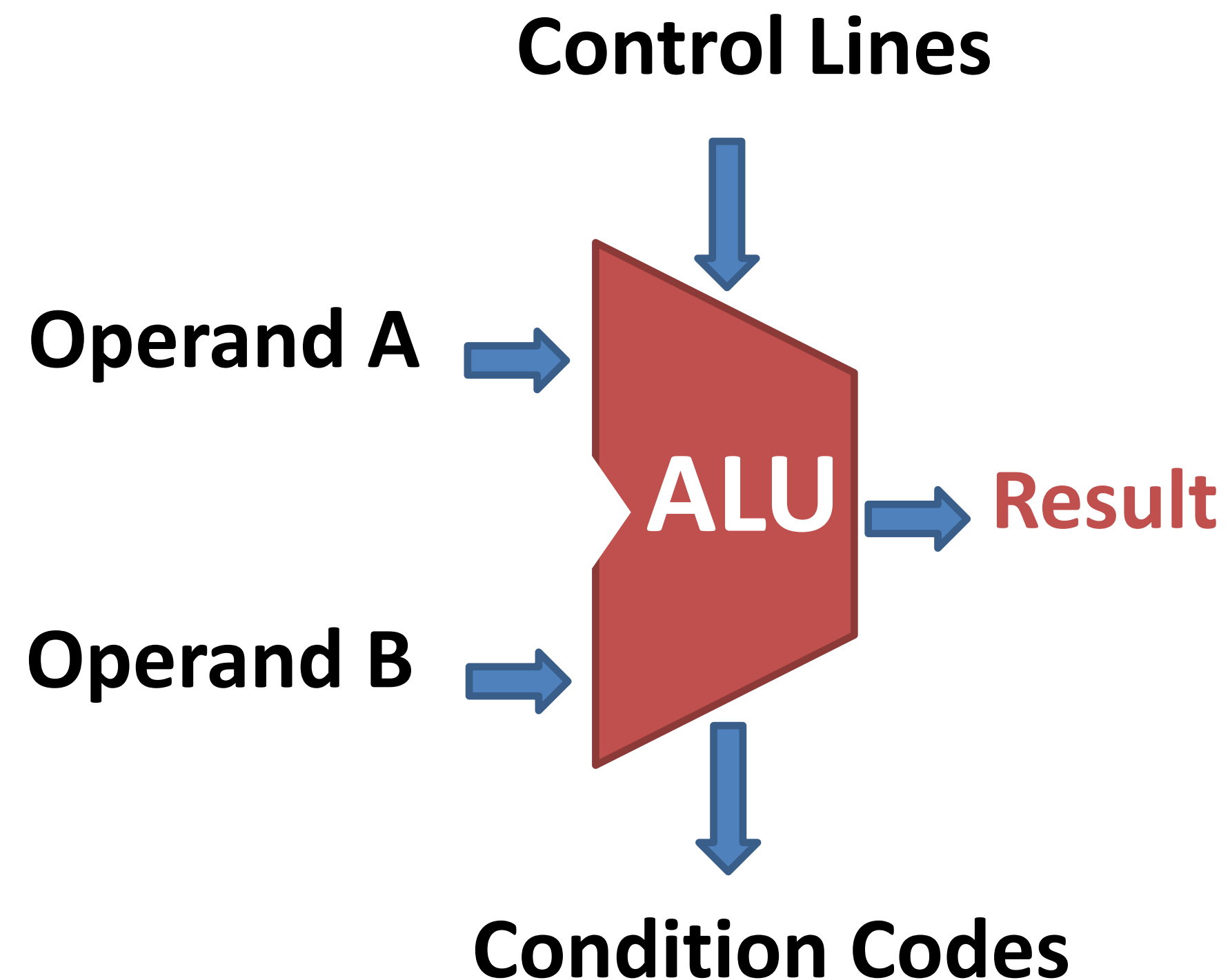# Sequential Logic and State

Output depends on inputs *and stored values*.

(vs. combinational: output depends only on inputs)

Elements to store values: latches, flip-flops, registers, memory

Motivation

Now that we have ALUs to perform computations, how do we store the results?

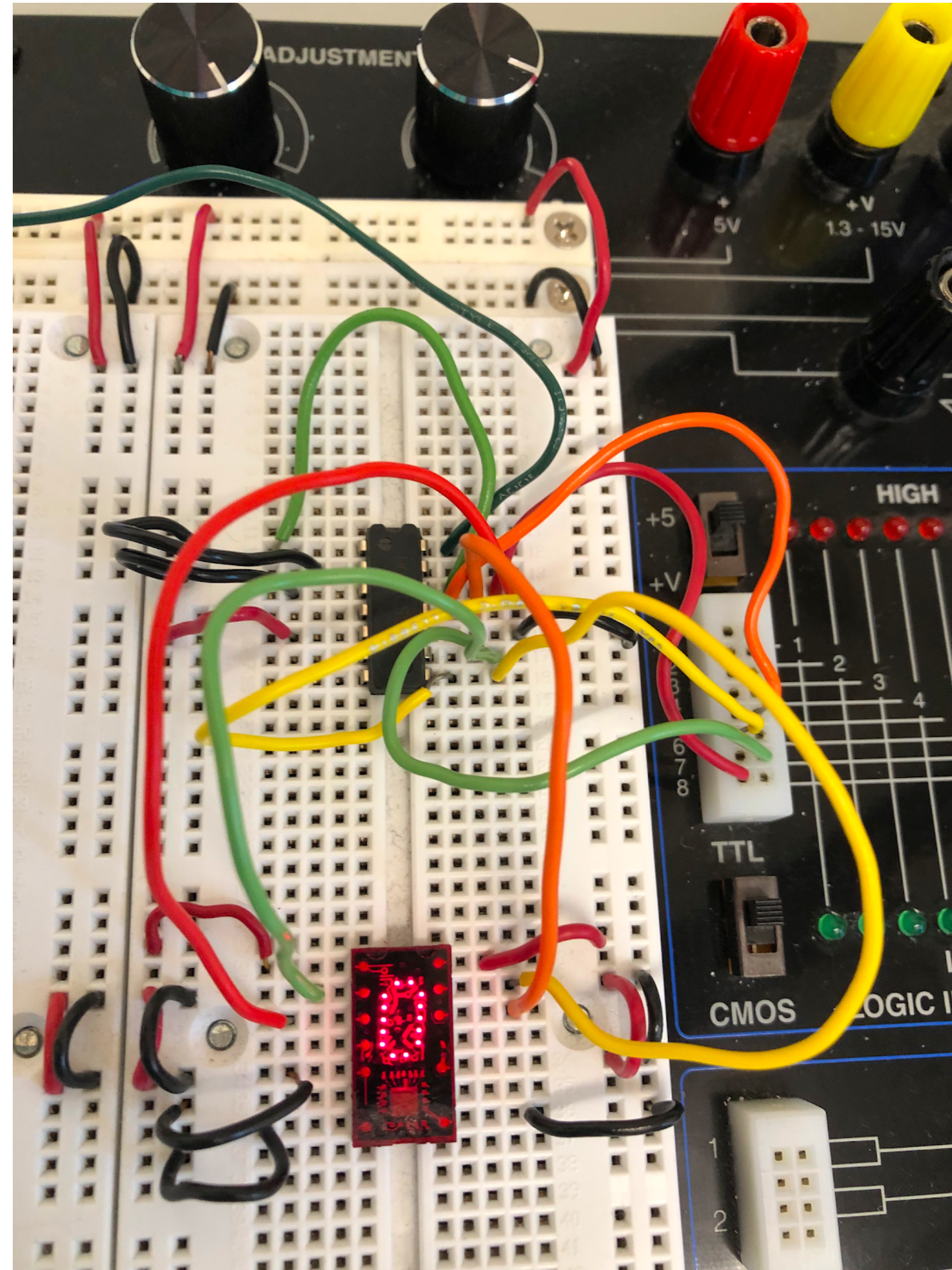How do we calculate different results over time?

Answer: we need circuits that depend not just on inputs, but also on *prior state* = *Sequential Logic*

Can you think of an example from lab of a sequential circuit you used?
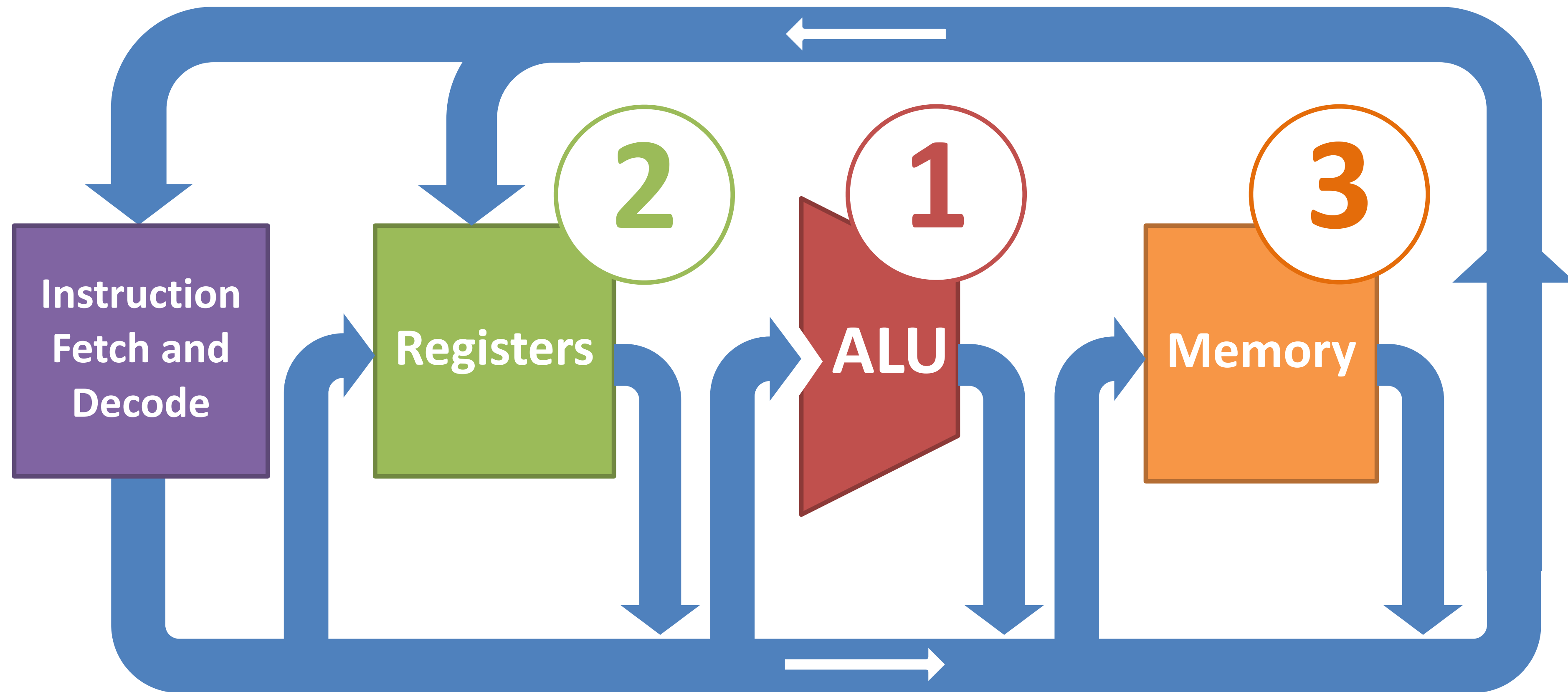# Hint: previous button pushes are past state.

Nobody has responded yet.

Hang tight! Responses are coming in.

Example from previous lab
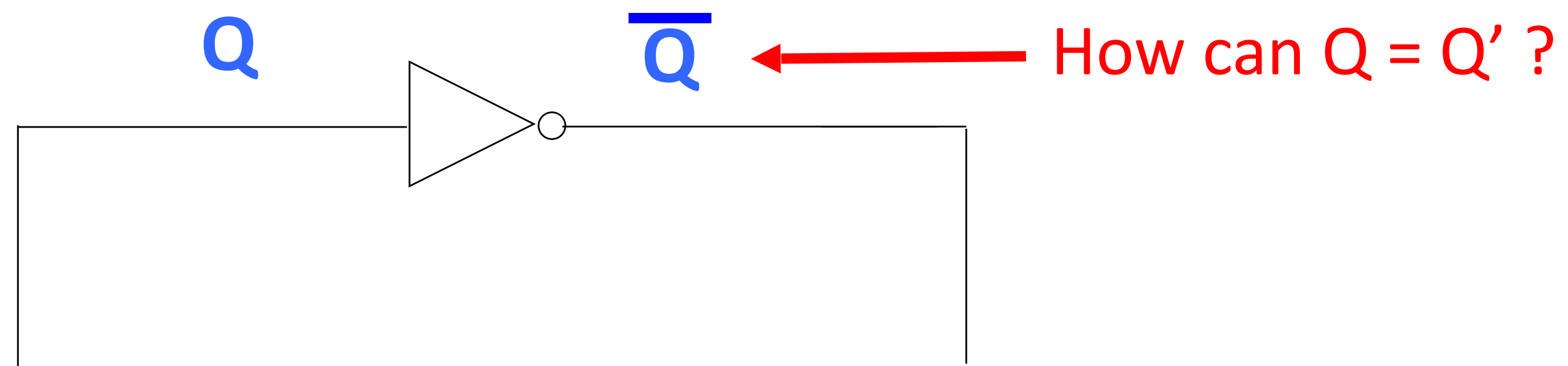
Processor: Data Path Components

Goal for this section

Design a circuit state that **holds** a state over time

- We should be able to set the value to 0 or 1
- We should be able to read the value off the circuit
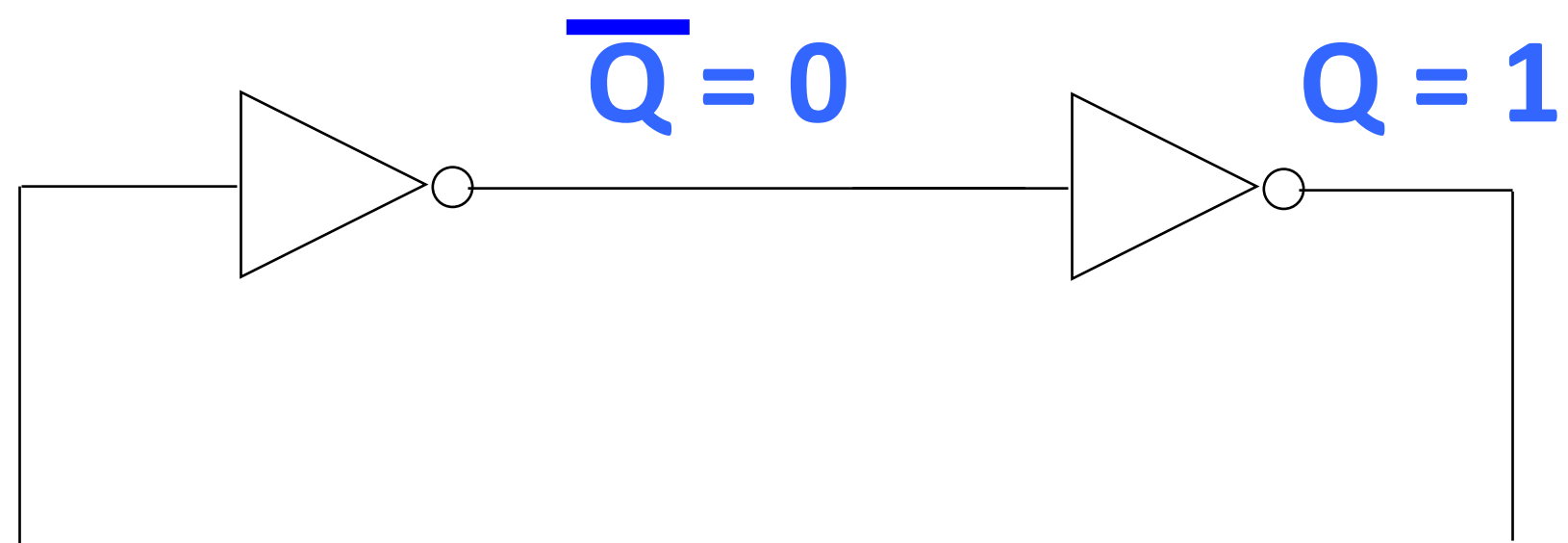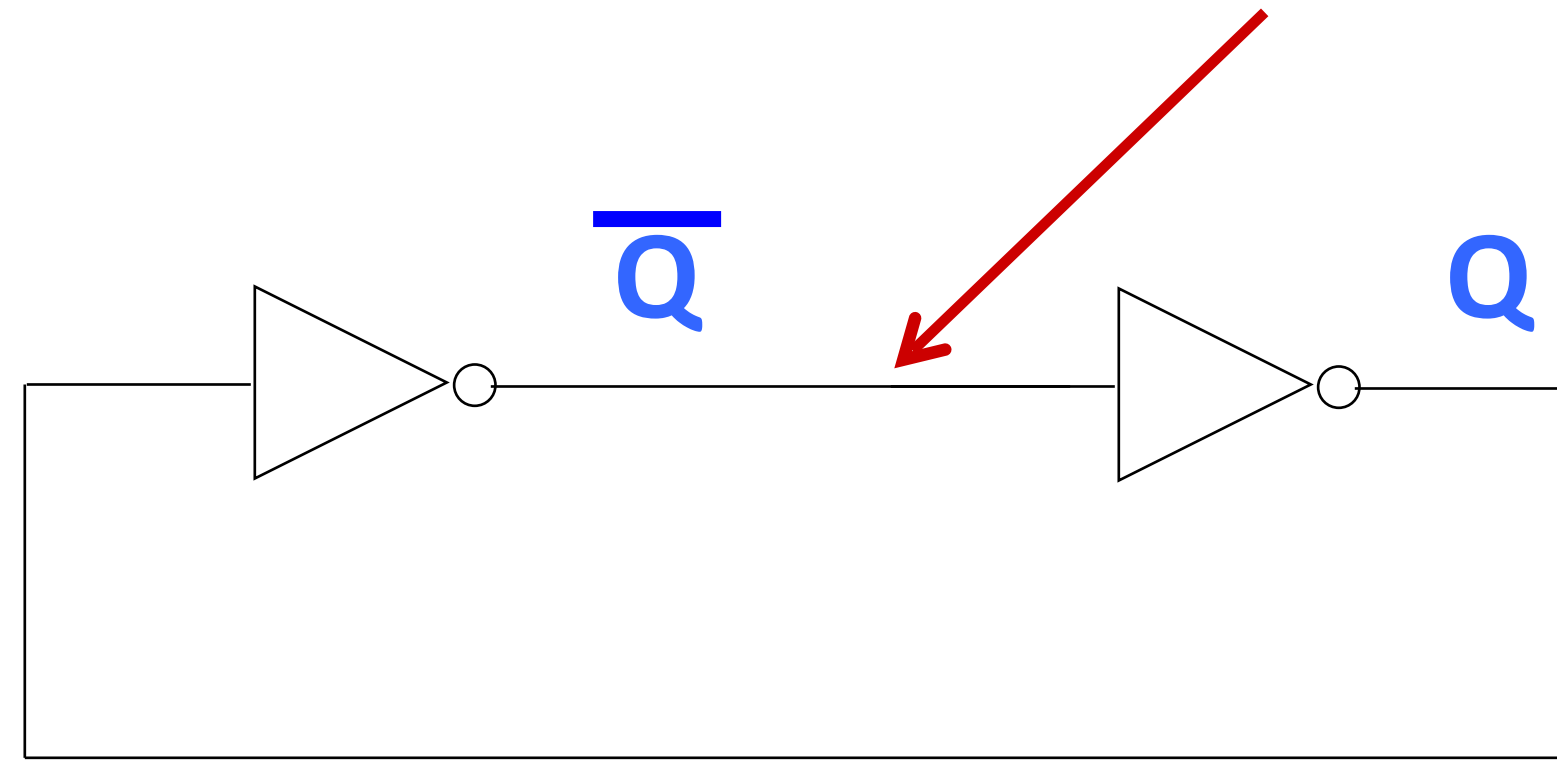
First attempt: Unstable circuit

Q $\overline{Q}$ ← How can Q = Q' ?

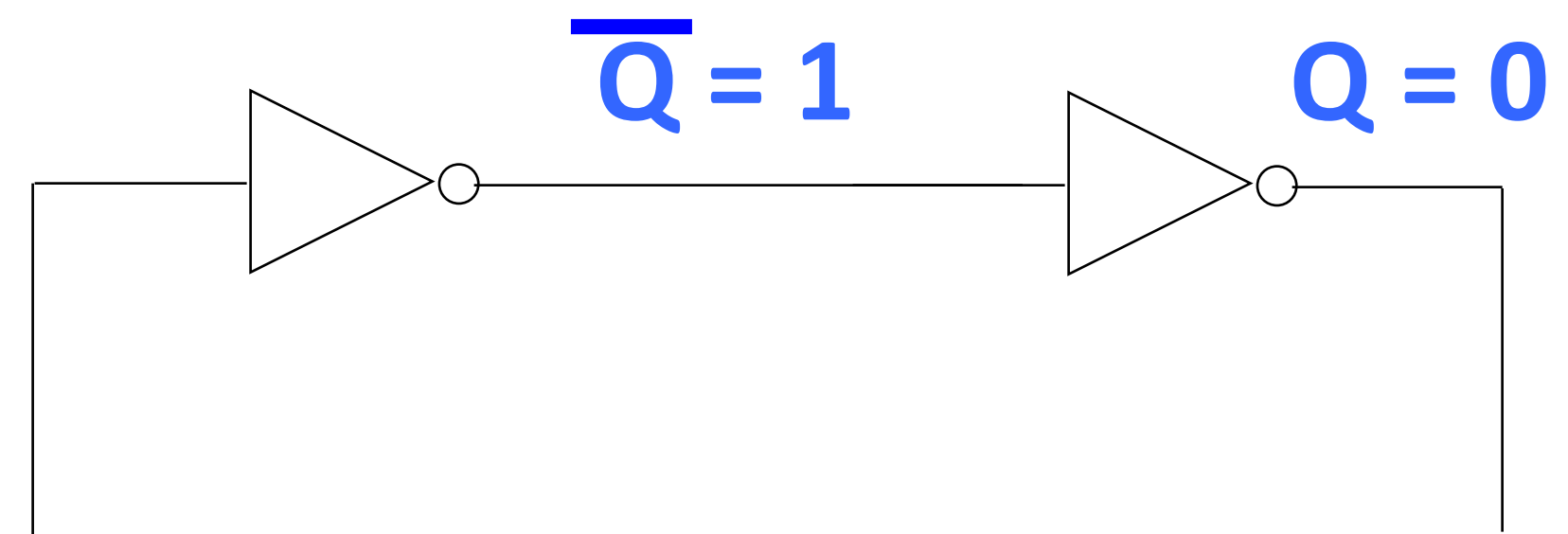Have this issue with any odd number of inverters in a loop.

Second attempt: stable circuit?

Things are more sensible with an even number of inverters in a loop.

Suppose we somehow get a 1 (or a 0?) on here.

$\overline{Q}$ Q

or

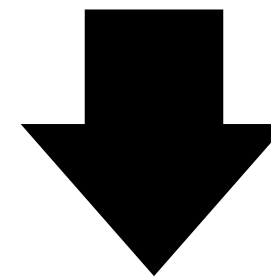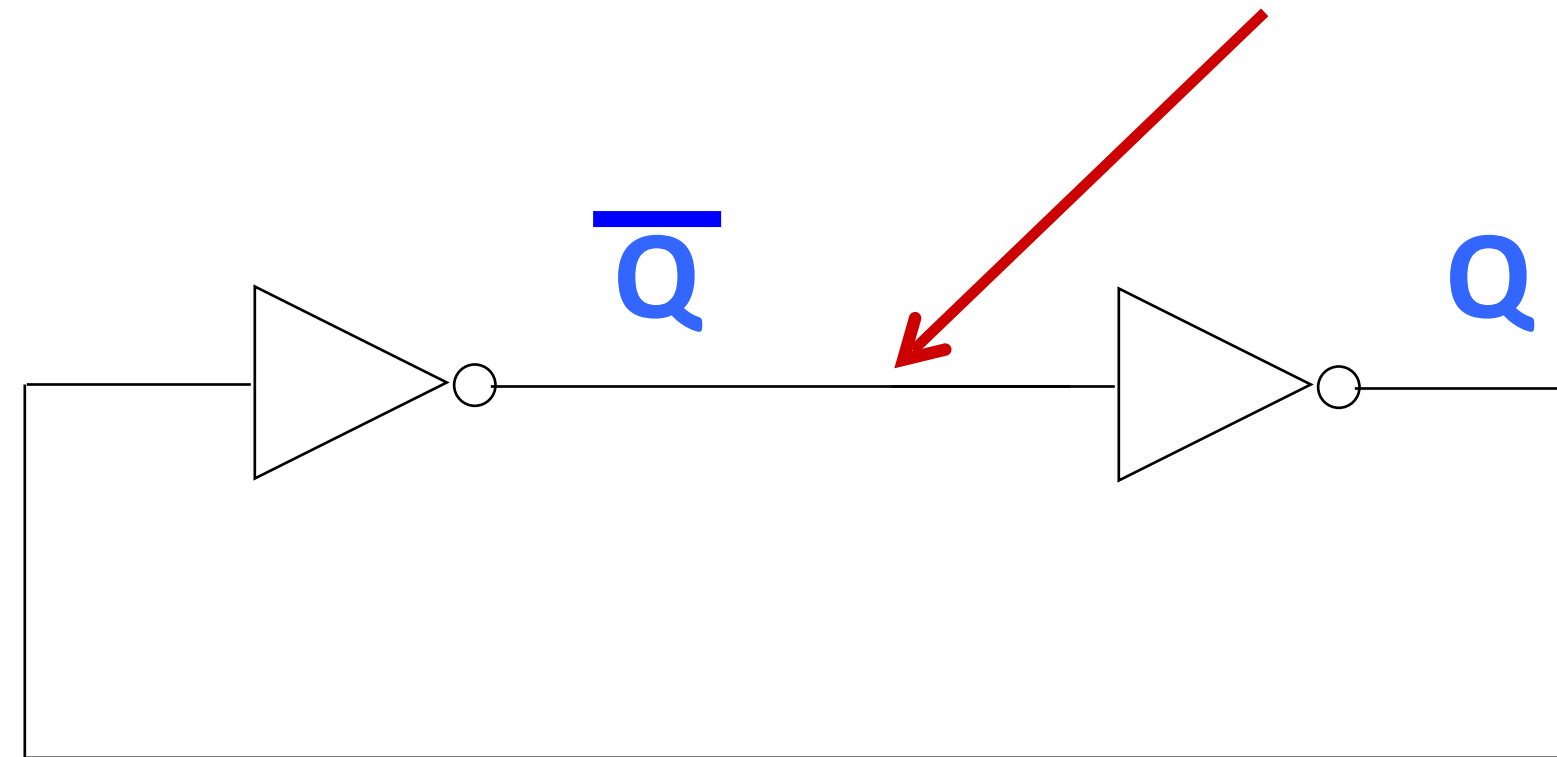$\overline{Q}$ = 0   Q = 1 $\overline{Q}$ = 1   Q = 0

Now stable, but how do we set the value?

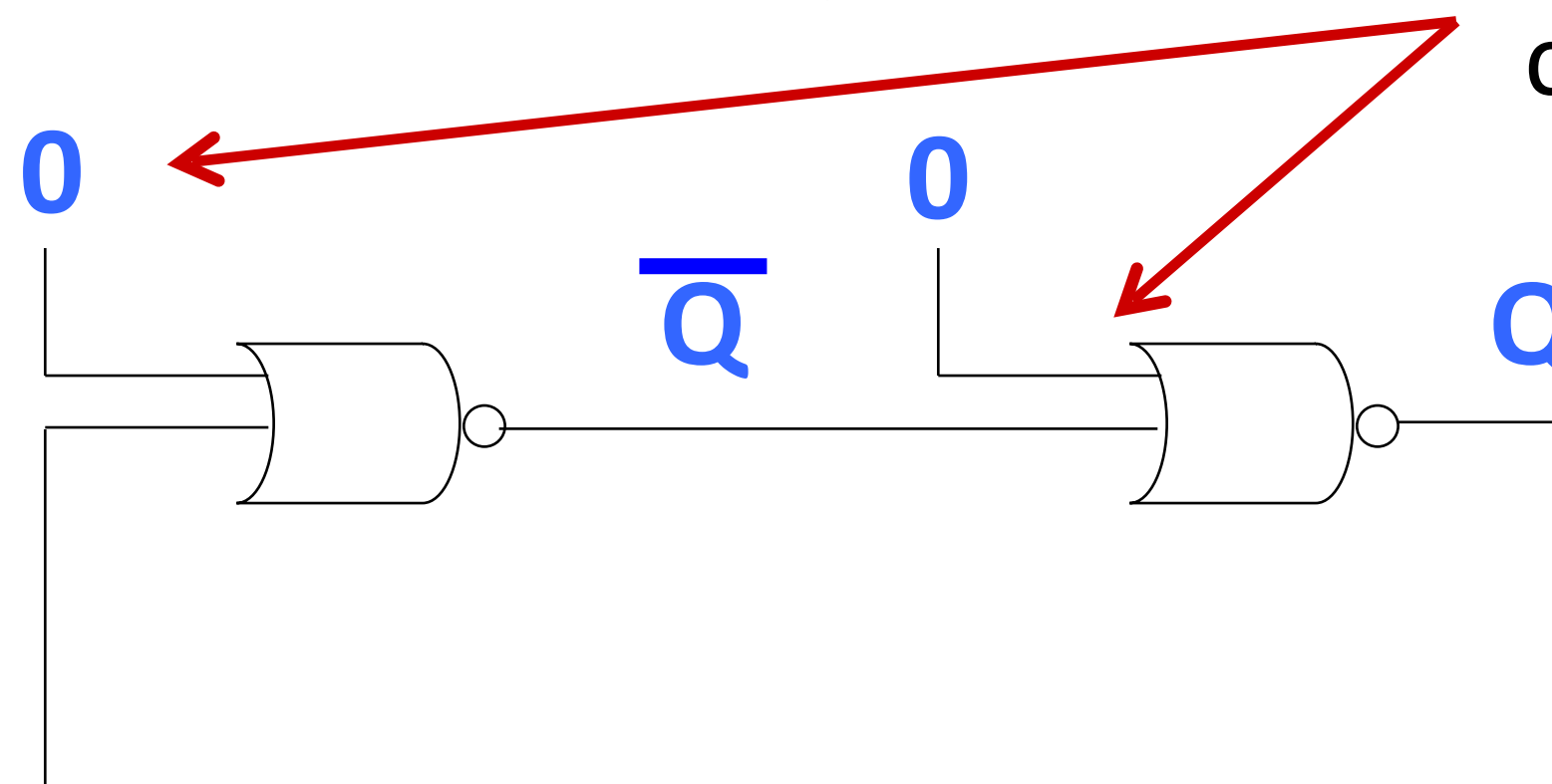# Bistable latches

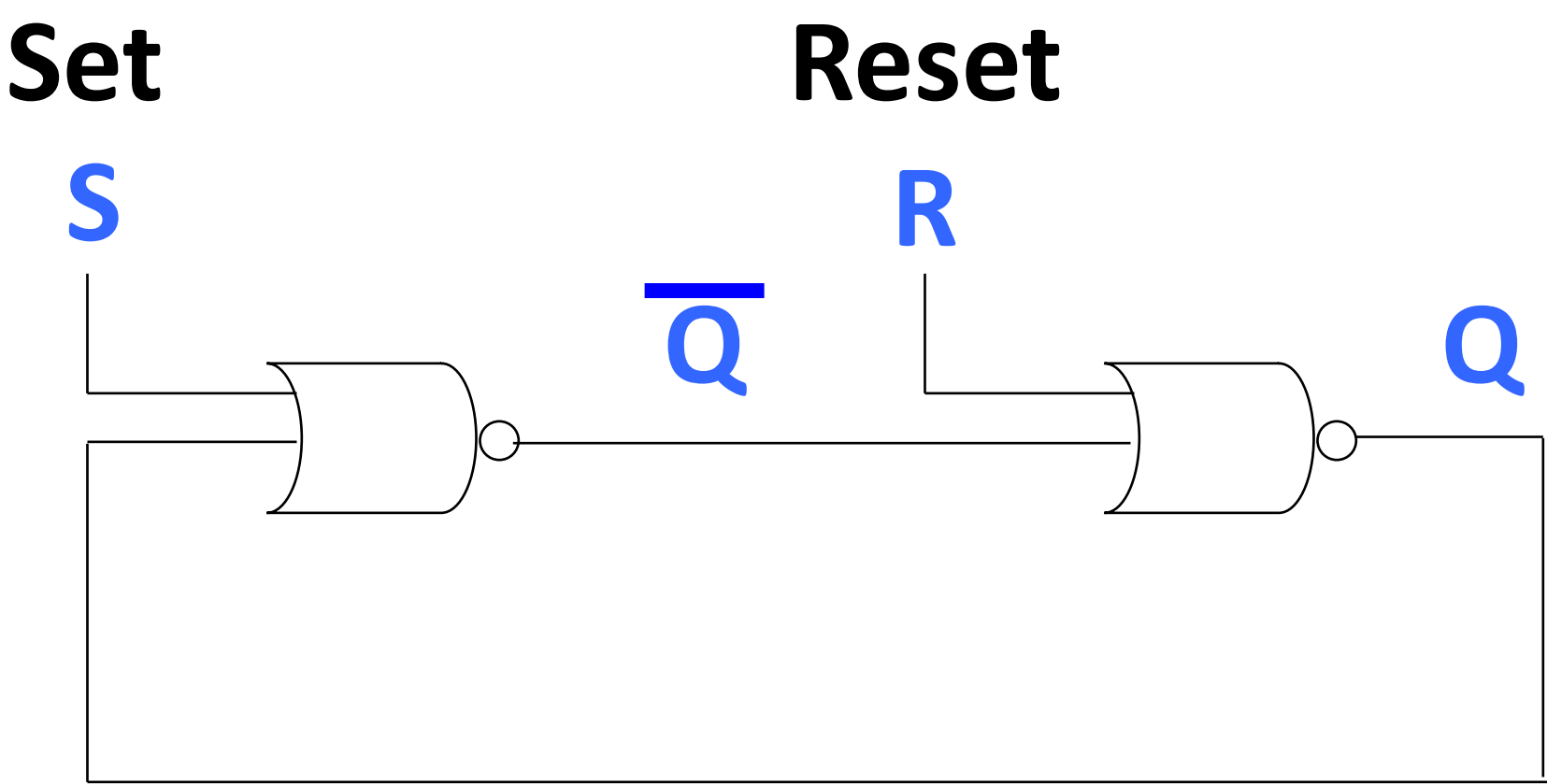Things are more sensible with an even number of inverters in a loop.

Suppose we somehow get a 1 (or a 0?) on here.

$\overline{Q}$ Q

Change to a 2-input gates so that we can **set** updated values to be stored

0   0

$\overline{Q}$ Q

SR latch

Set   Reset

S   **R**

$\overline{Q}$ **Q**

S	R	Q_{prev}	Q'_{prev}	Q_{next} (stable)	Q'_{next} (stable)
0	0	0	1	0	1
0	0	1	0	1	0
1	0	*any*	*any*	1	0
0	1	*any*	*any*	0	1
1	**1**	***any***	***any***	**0**	**0**

Violates invariant that
Q and Q' are inverses!

SR latch

Move from the circuit we built to the canonical form

SR latch

R — Q

S — $\overline{Q}$

Figure 3.3 **SR latch schematic**

Meets our goals:
- Able to set the value to 0 or 1
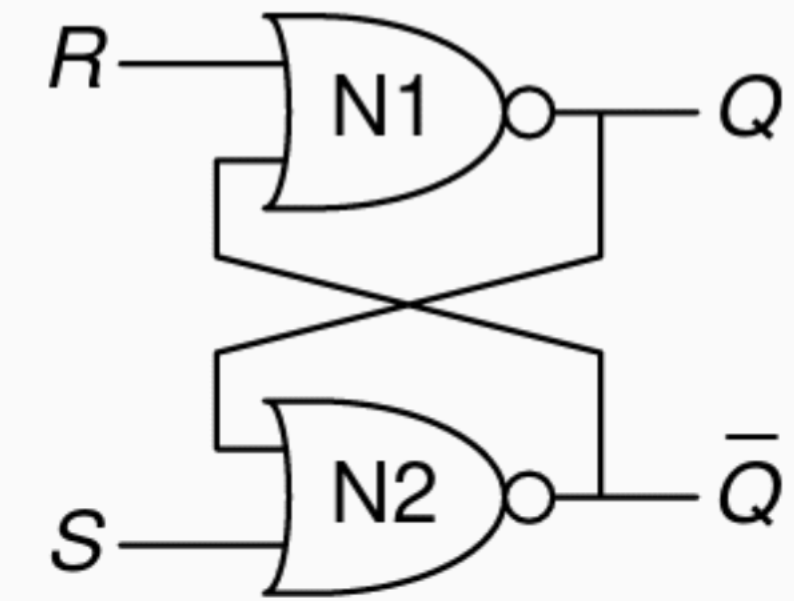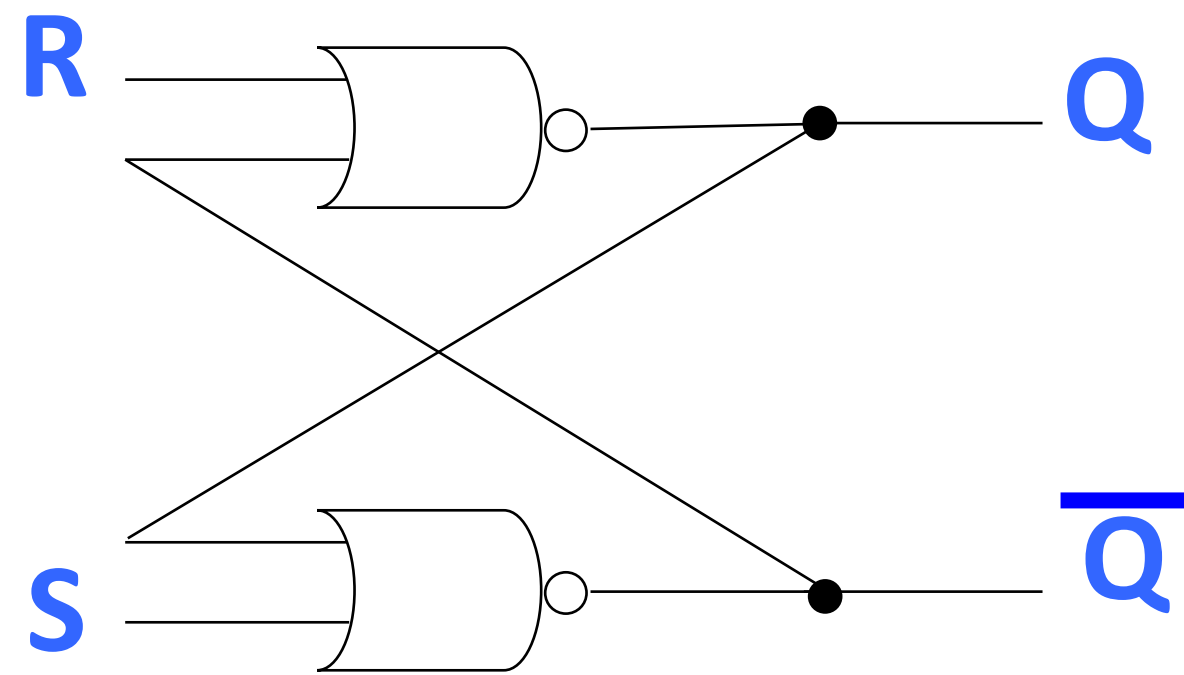- Able to read the value off the circuit

How do we set Q to 1?

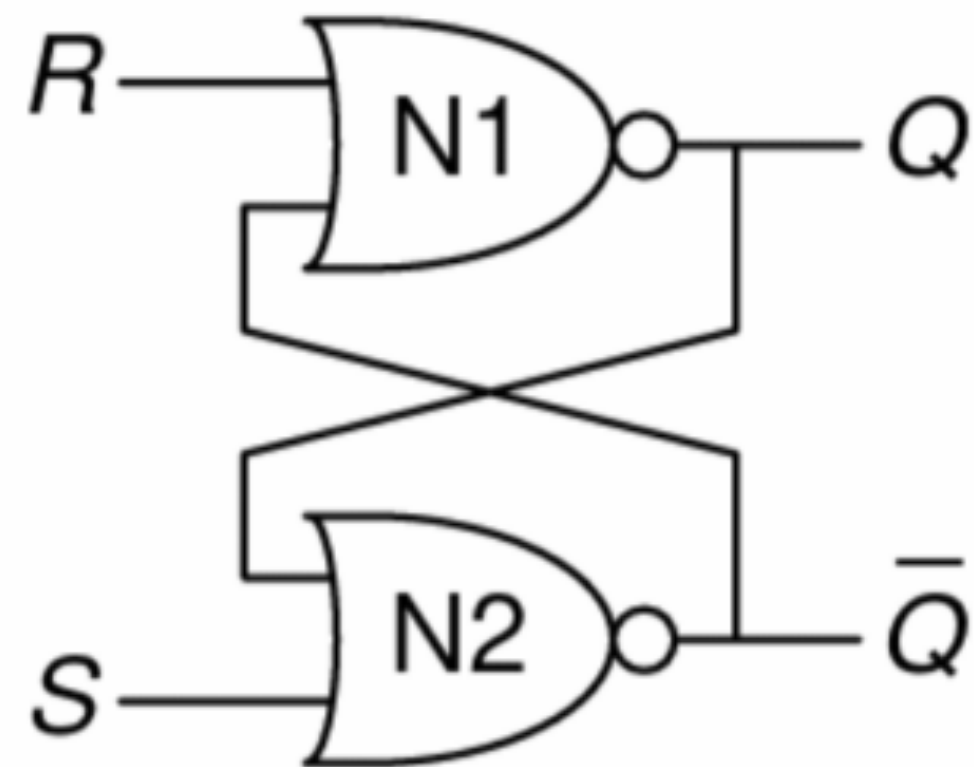Figure 3.3 SR latch schematic

S = 0; R = 0

S = 1; R = 0

S = 0; R = 1

S = 1; R = 1

None of the above

How do we set Q to 1?

Figure 3.3 SR latch schematic

S = 0; R = 0

0%

S = 1; R = 0

0%

S = 0; R = 1

0%

S = 1; R = 1

0%

None of the above

0%

How do we set Q to 1?
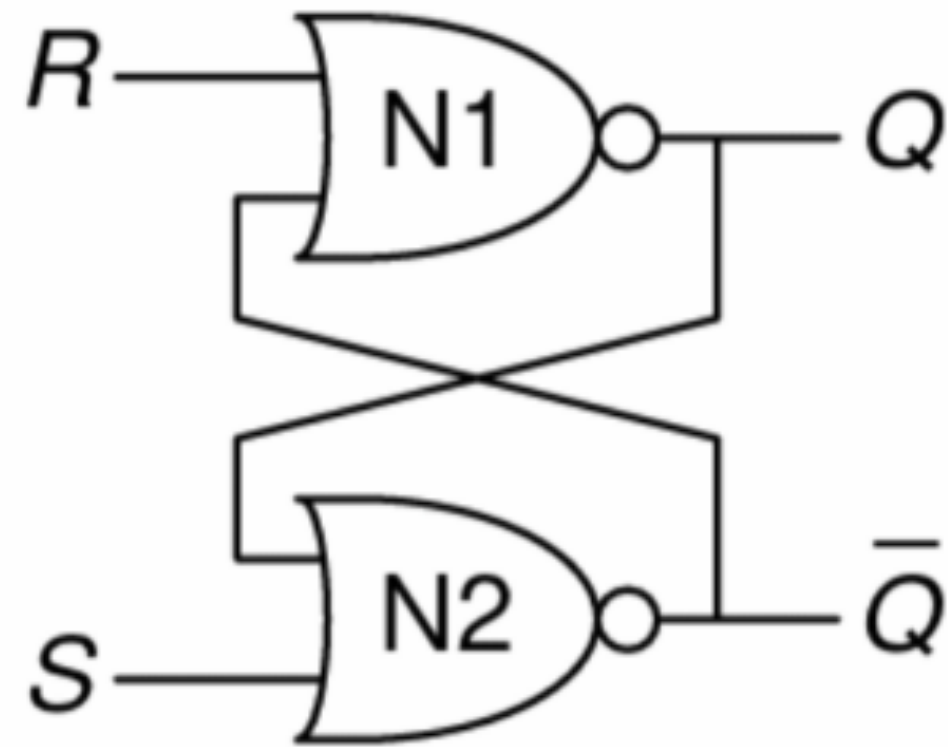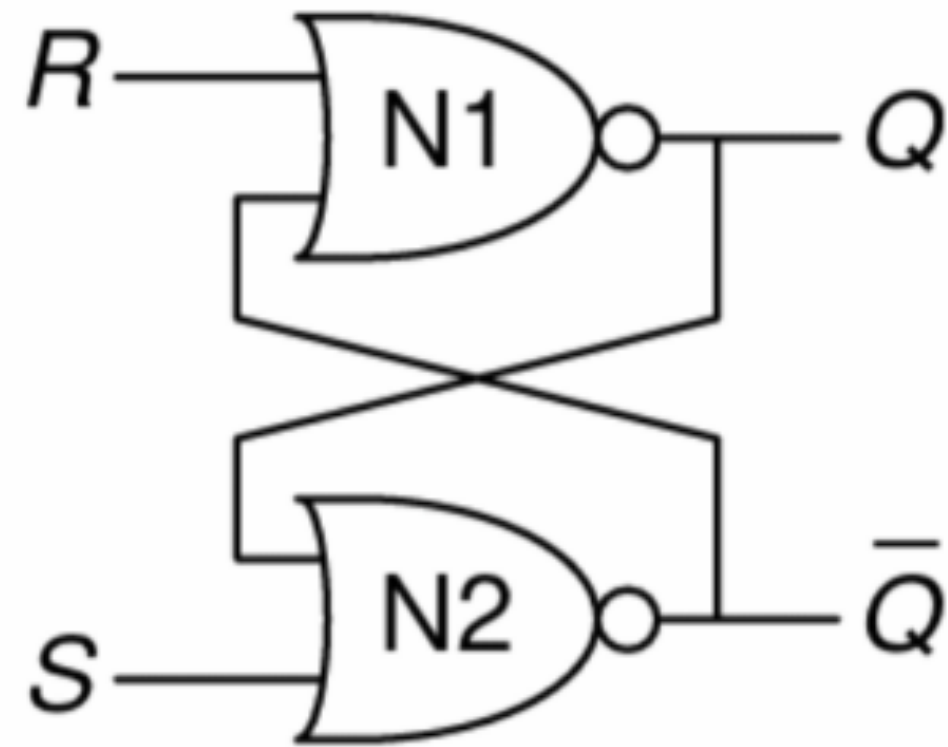

Figure 3.3 SR latch schematic

S = 0; R = 0

0%

S = 1; R = 0

0%

S = 0; R = 1

0%

S = 1; R = 1

0%

None of the above

0%

SR latch
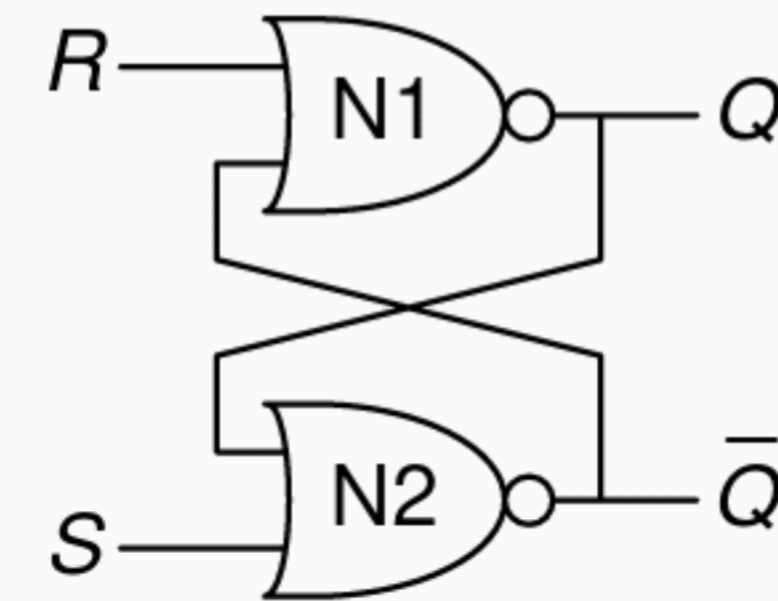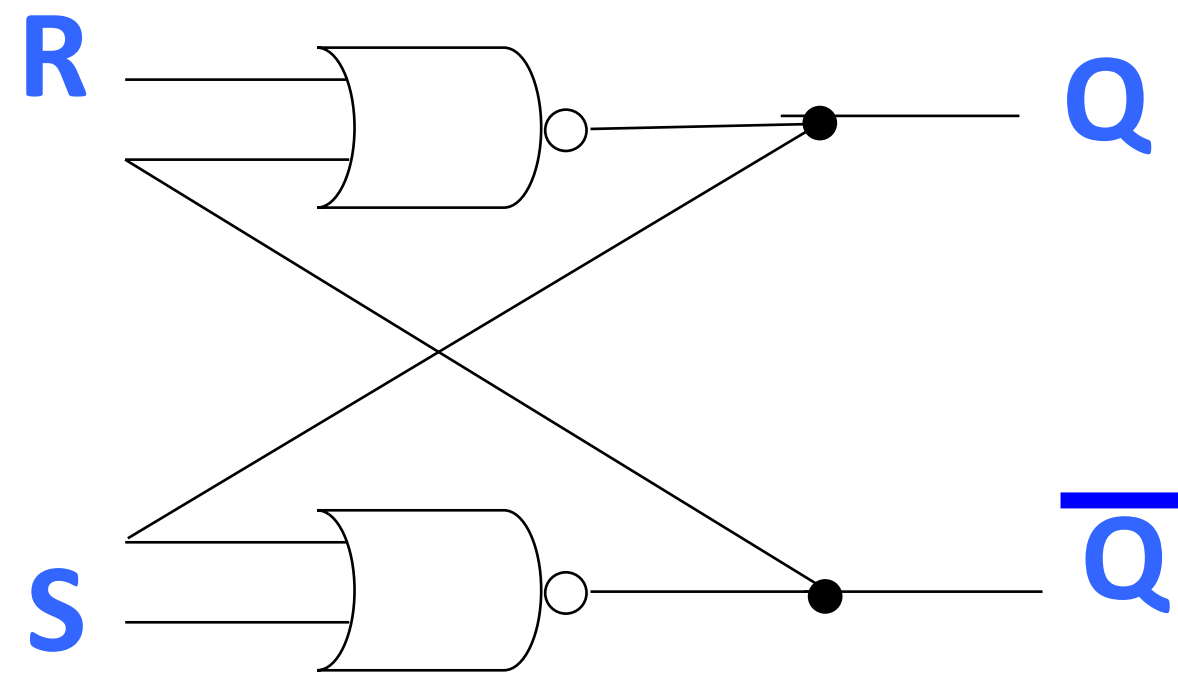
Figure 3.3 SR latch schematic

Meets our goals:
- Able to set the value to 0 or 1
- Able to read the value off the circuit

But:
- Ambiguous when S = 1 and R = 1
- No distinction between new value and timing

D latch

Goals:
- Only 1 bit for data
- Control over timing
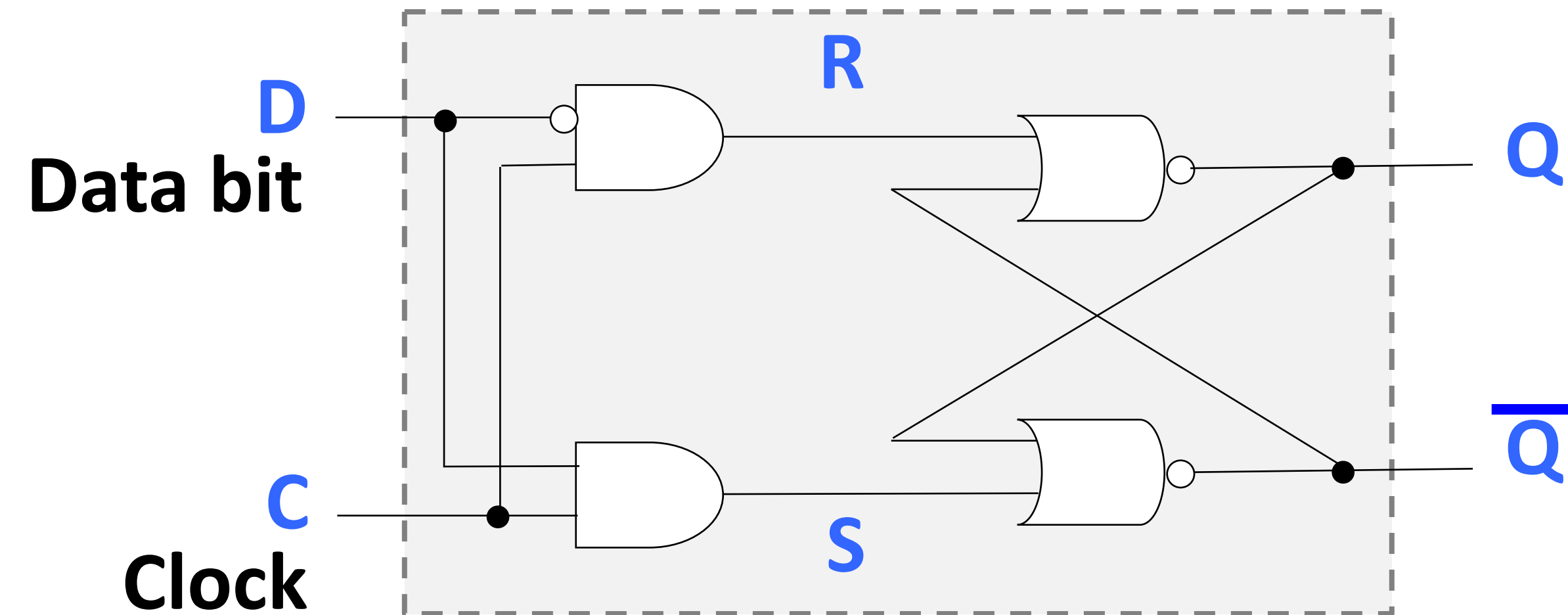
if C = 0, then SR latch stores current value of Q.

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

Notes:

- Data bit D replaces S & R: it's the bit value we want to store when Clock = 1
  - Internally, Data bit D prevents bad case of S = R = 1
- This logic is **level-triggered**; as long as Clock = 1, changes to D flow to outputs

Time matters!

D

C

Q
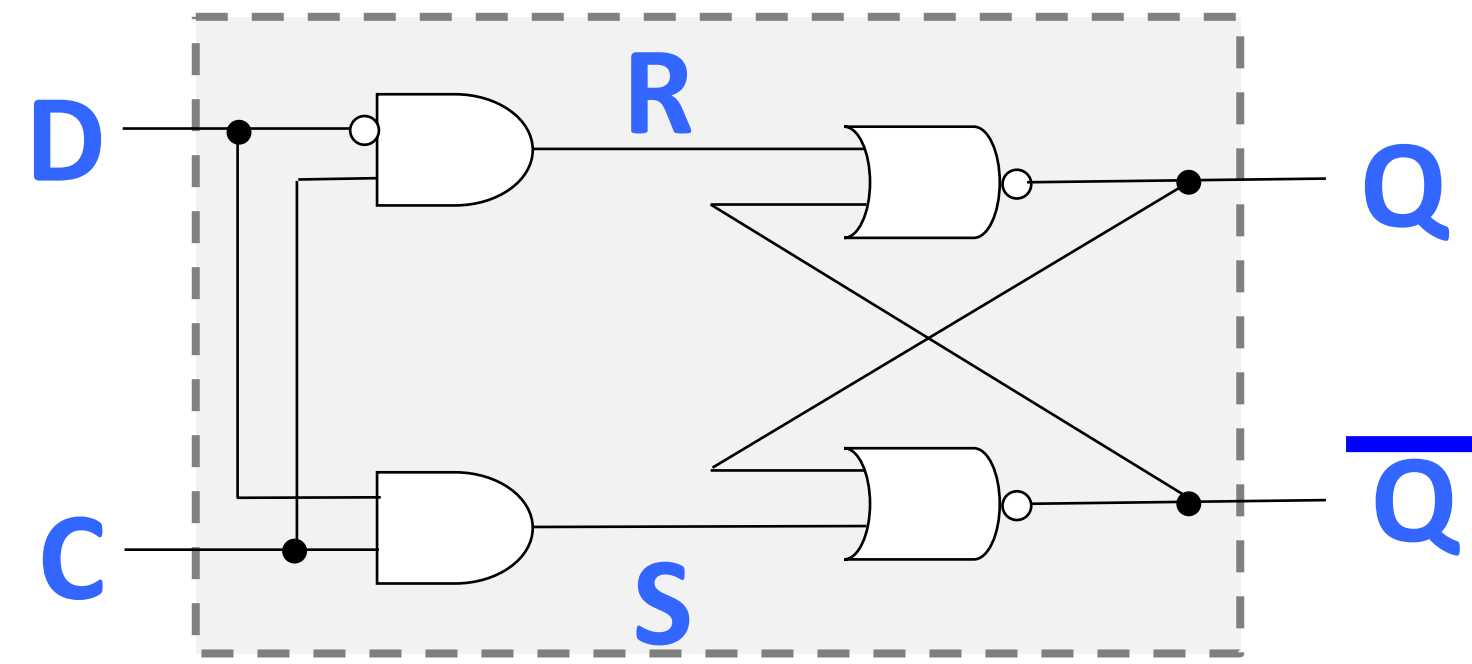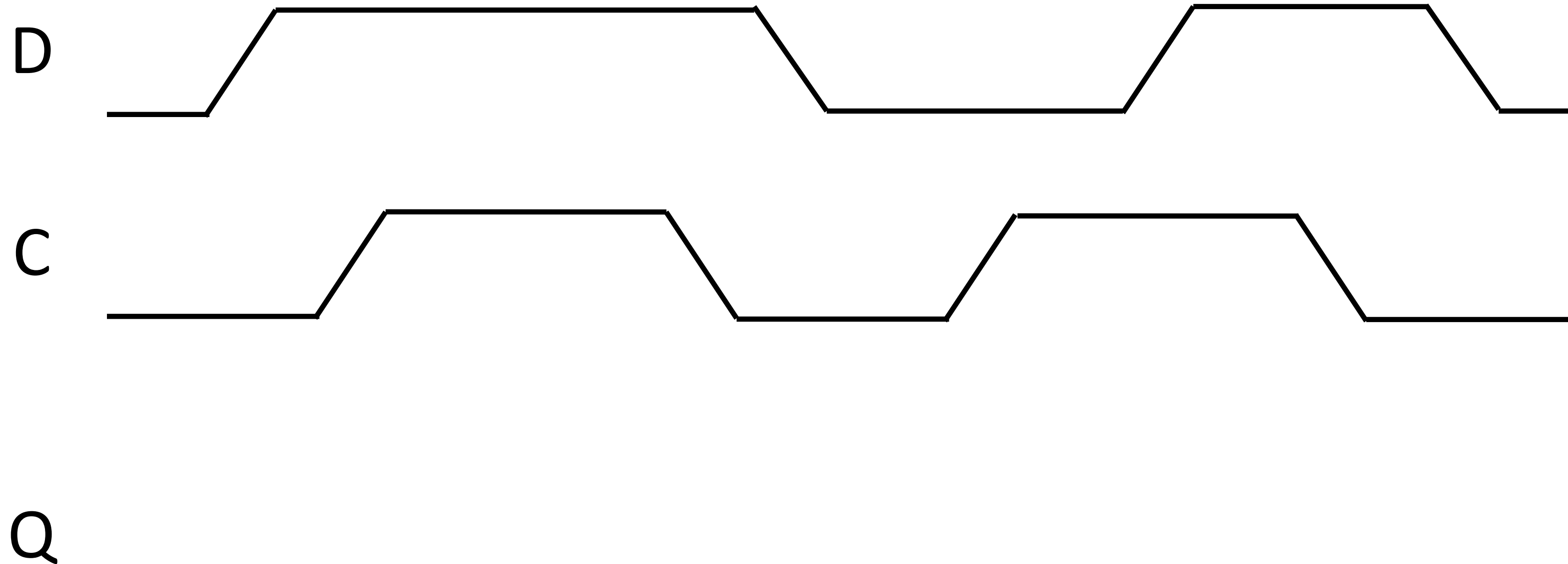
Assume Q has an initial state of 0

In general: clocks

Clock: free-running signal

with fixed cycle time = clock period = T.

Clock frequency = 1 / clock period

Falling edge

Clock period   Rising edge
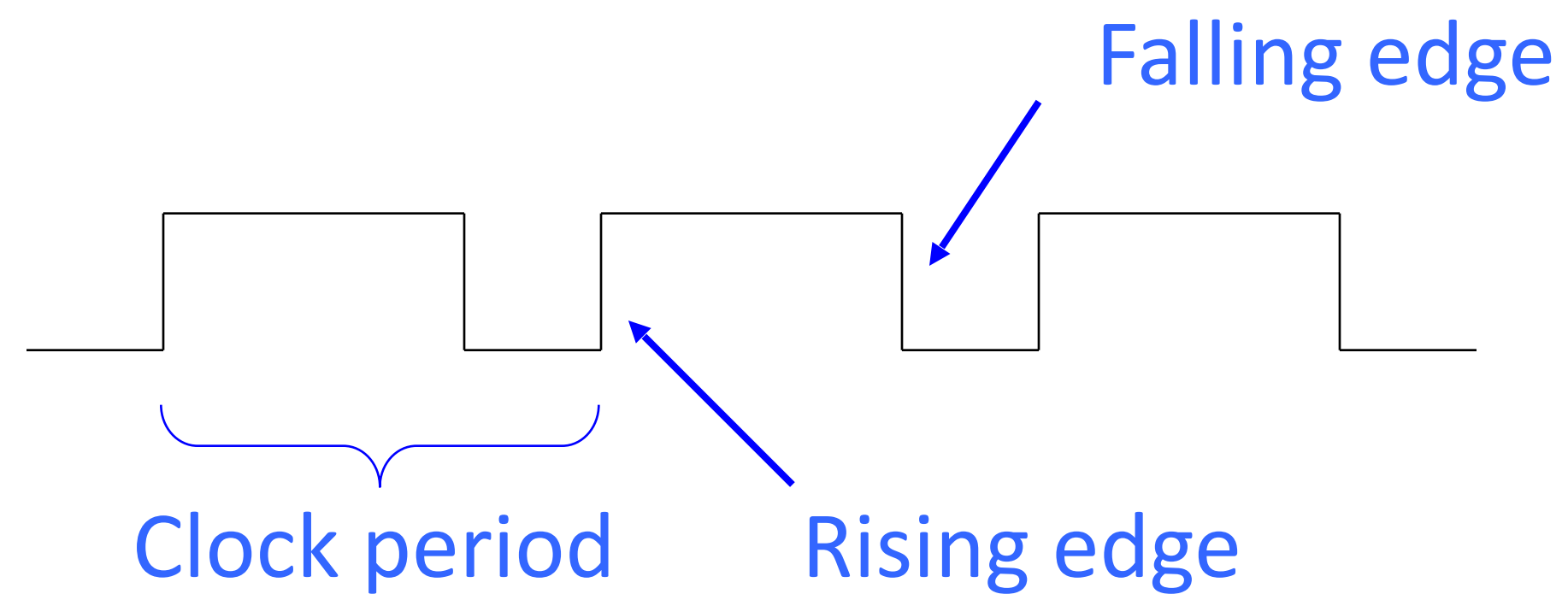
A clock controls when to update a sequential logic element's state.

Aside: "Clock frequency"

Clock frequency

= 1 / period = 1 / s = Hz

Typical CPU: 3-4 GHz

Microprocessor Clock Speed

Logarithmic Plot

$r^2 = 0.9699$

(Hz) vs Year

Doubling time: 3 years
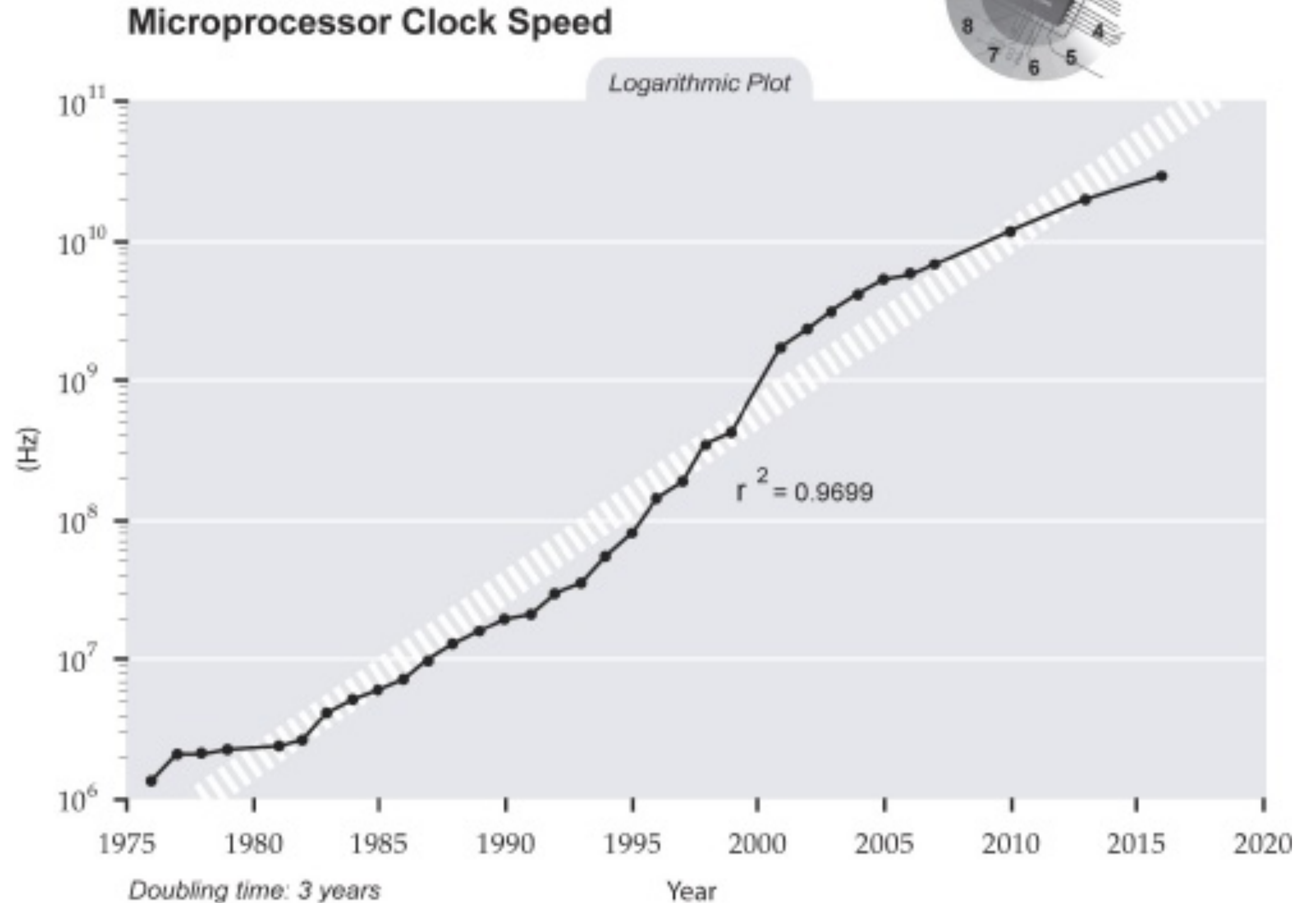
Synchronous systems

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

D flip-flop with *falling*-edge trigger

leader   follower

E

D D_L Q_L D_F Q_F Q

D latch   D latch

C_L $\overline{Q_L}$ C_F $\overline{Q_F}$ $\overline{Q}$

C

Assume

Q = Q_{now}

Q still = Q_{now}

follower stores

E = Q_{next} **as Q**

Clock

leader stores

D = Q_{next} **as E**

Q is now Q_{next}
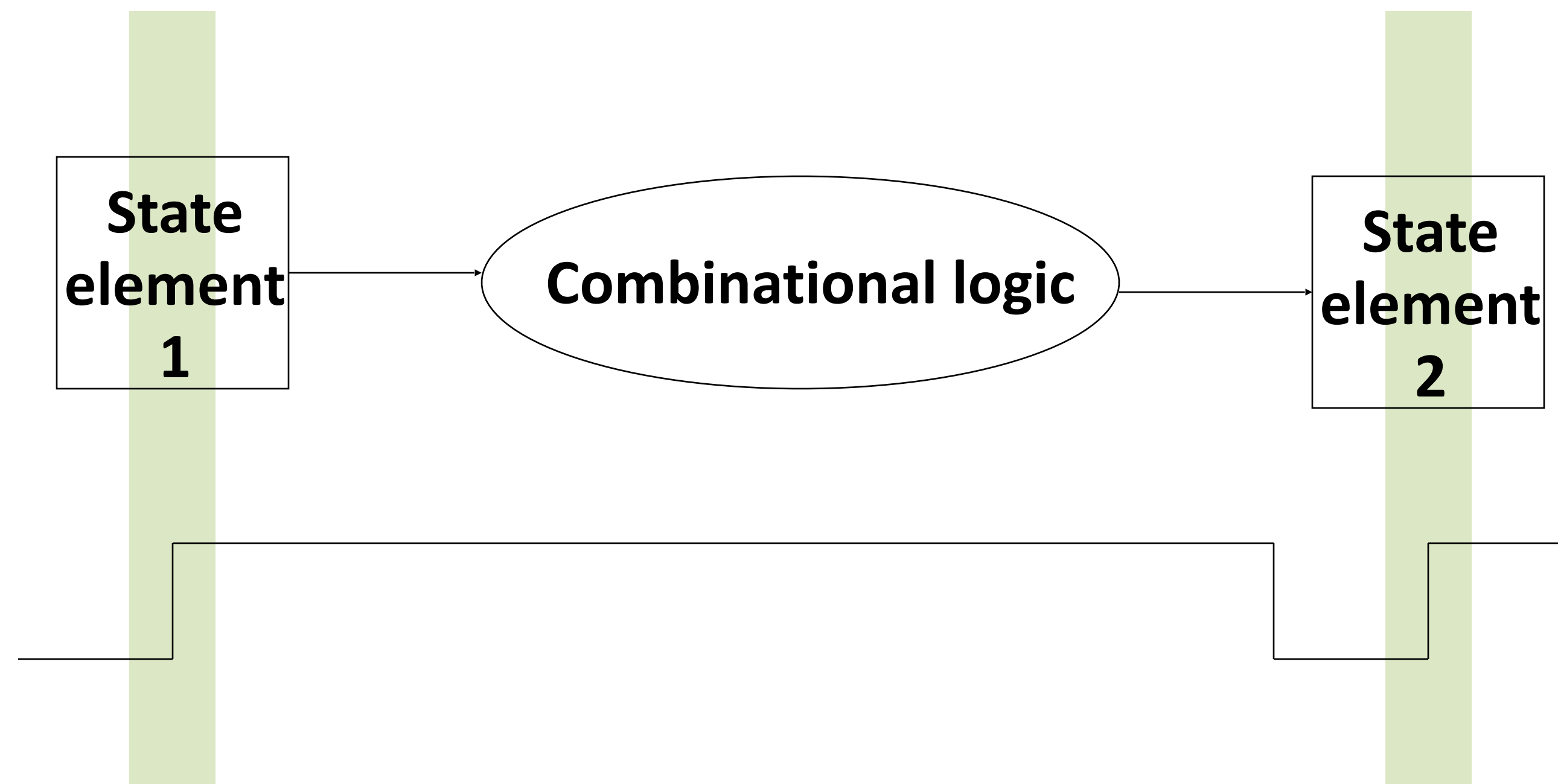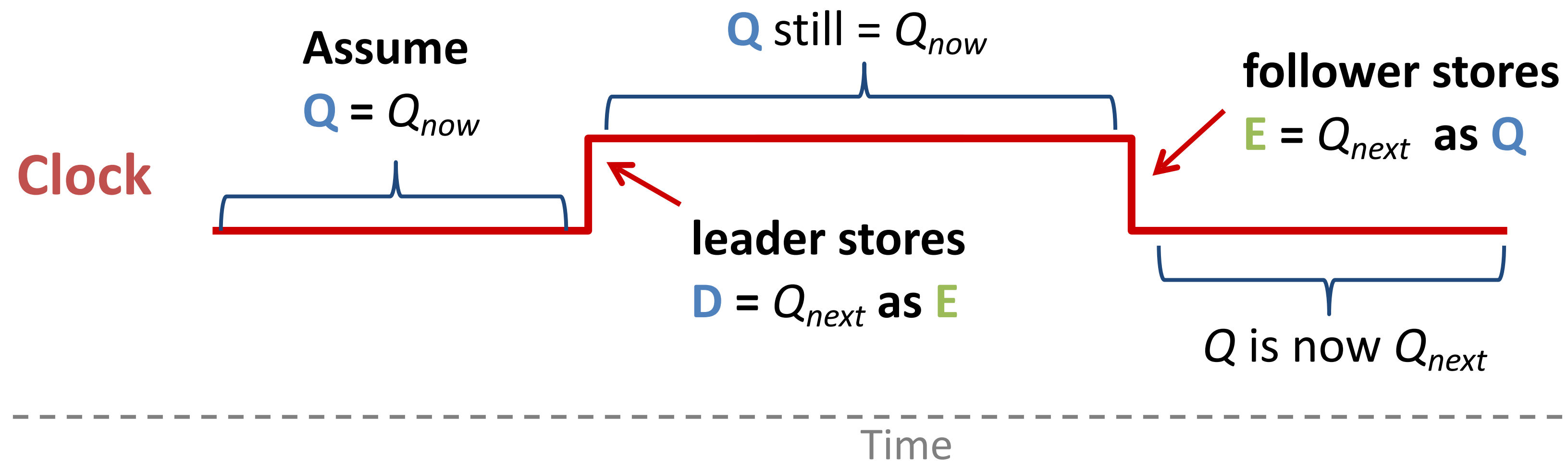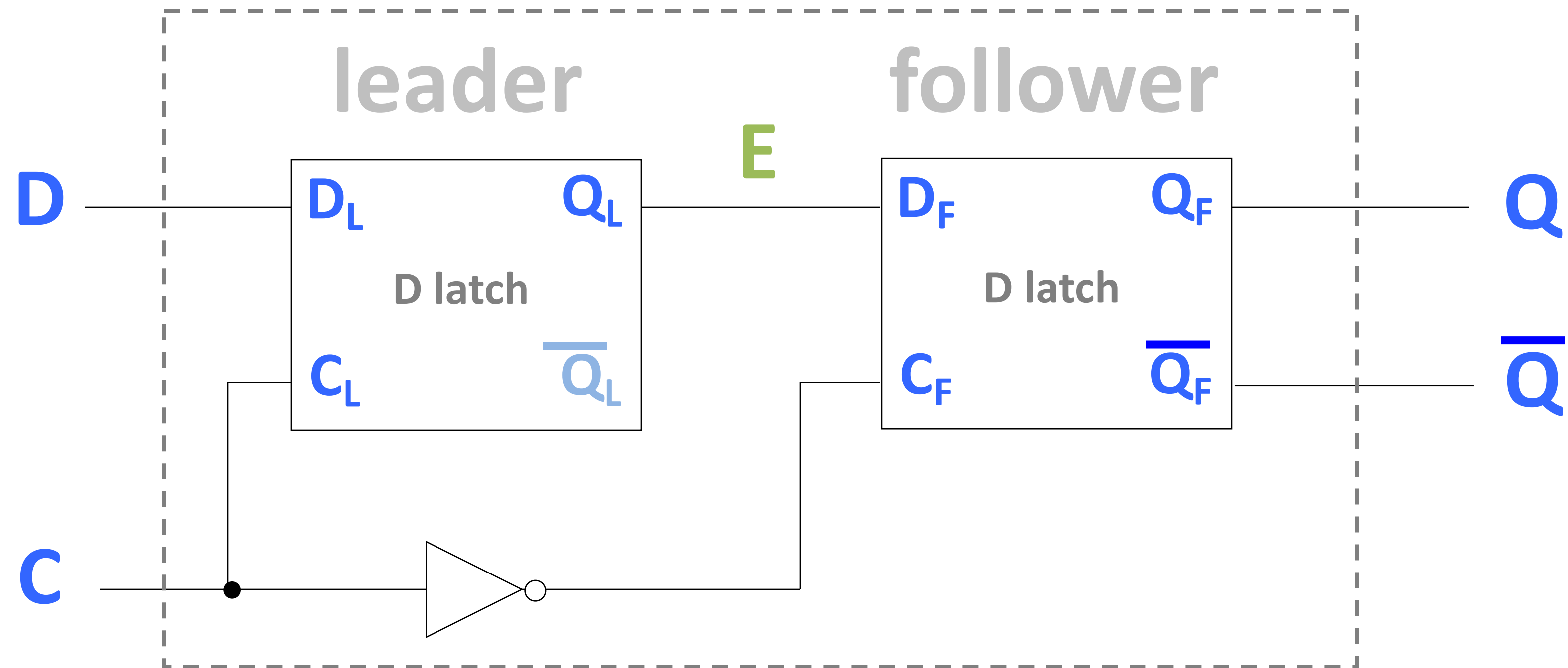
Time

Time matters! D flip-flop with *falling*-edge trigger

D

C

E

Q
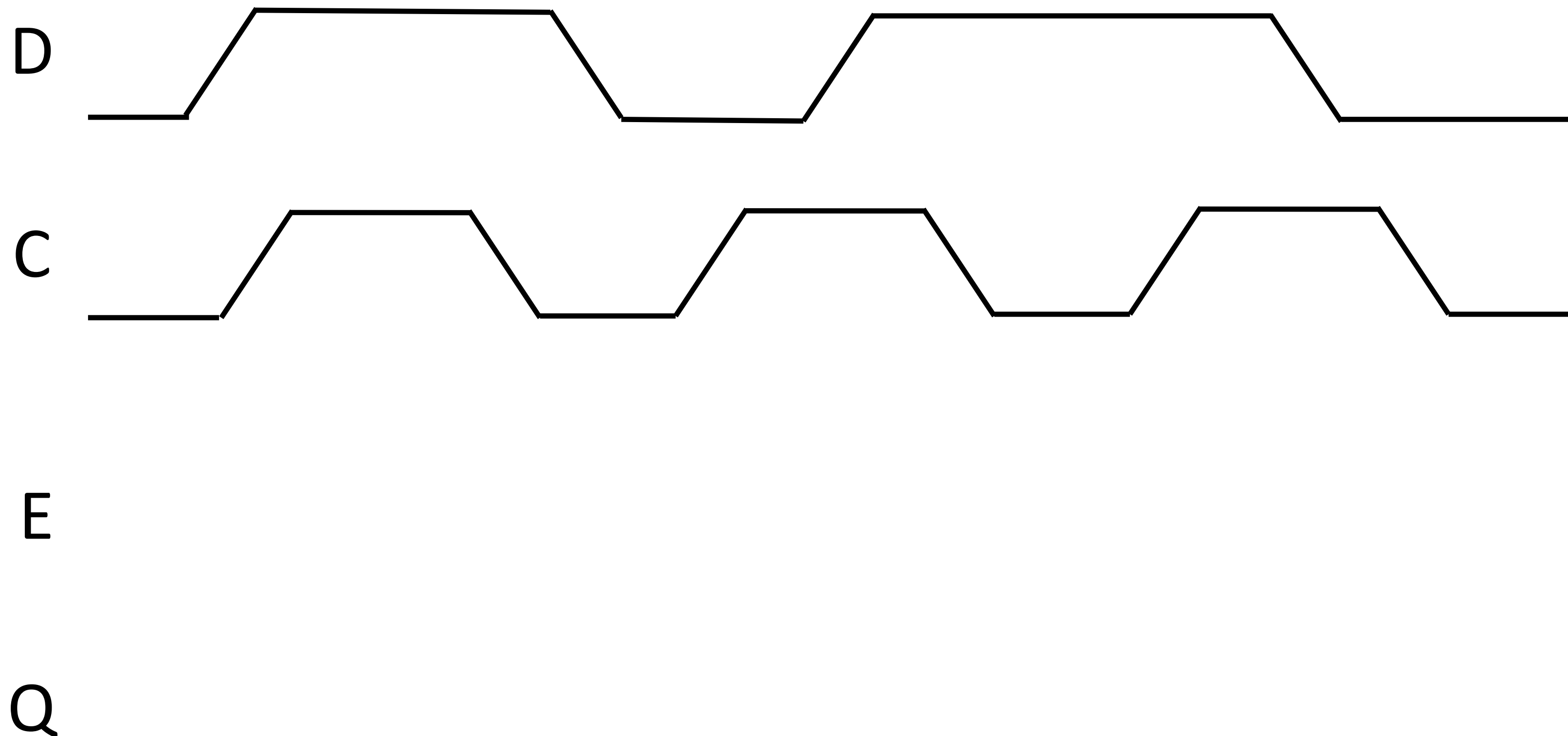
Assume Q and E have an initial state of 0

Reading and writing in the same cycle

Clock

D Flip-Flop

D   Q

C   Q̄

Assume Q is initially 0.
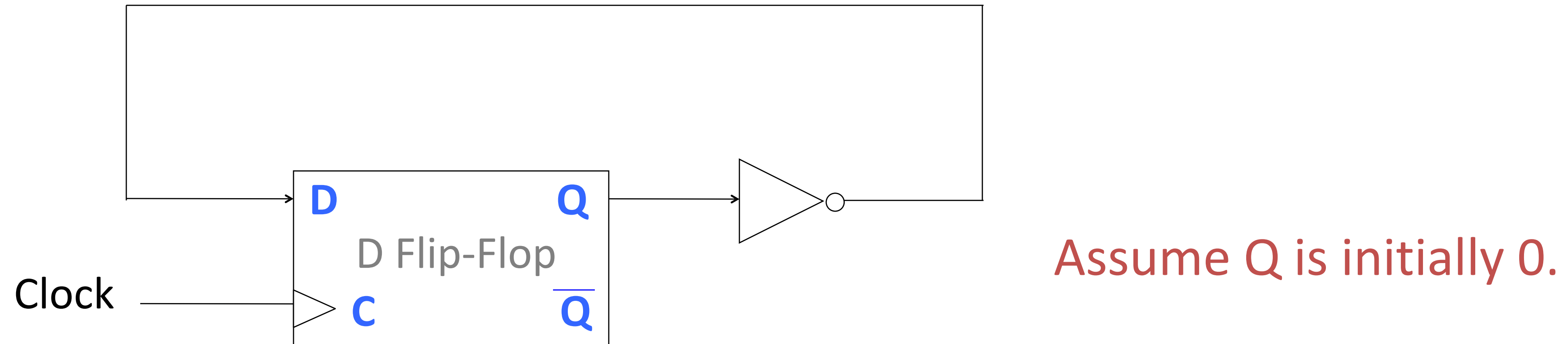
Moral: It's OK to use the current output Q of a flip-flop as part of the the next data input D to the same flip-flop.

D flip-flop = one bit of storage

The bit value of D when C has a falling edge is remembered at Q until the next falling edge of C.

Registers
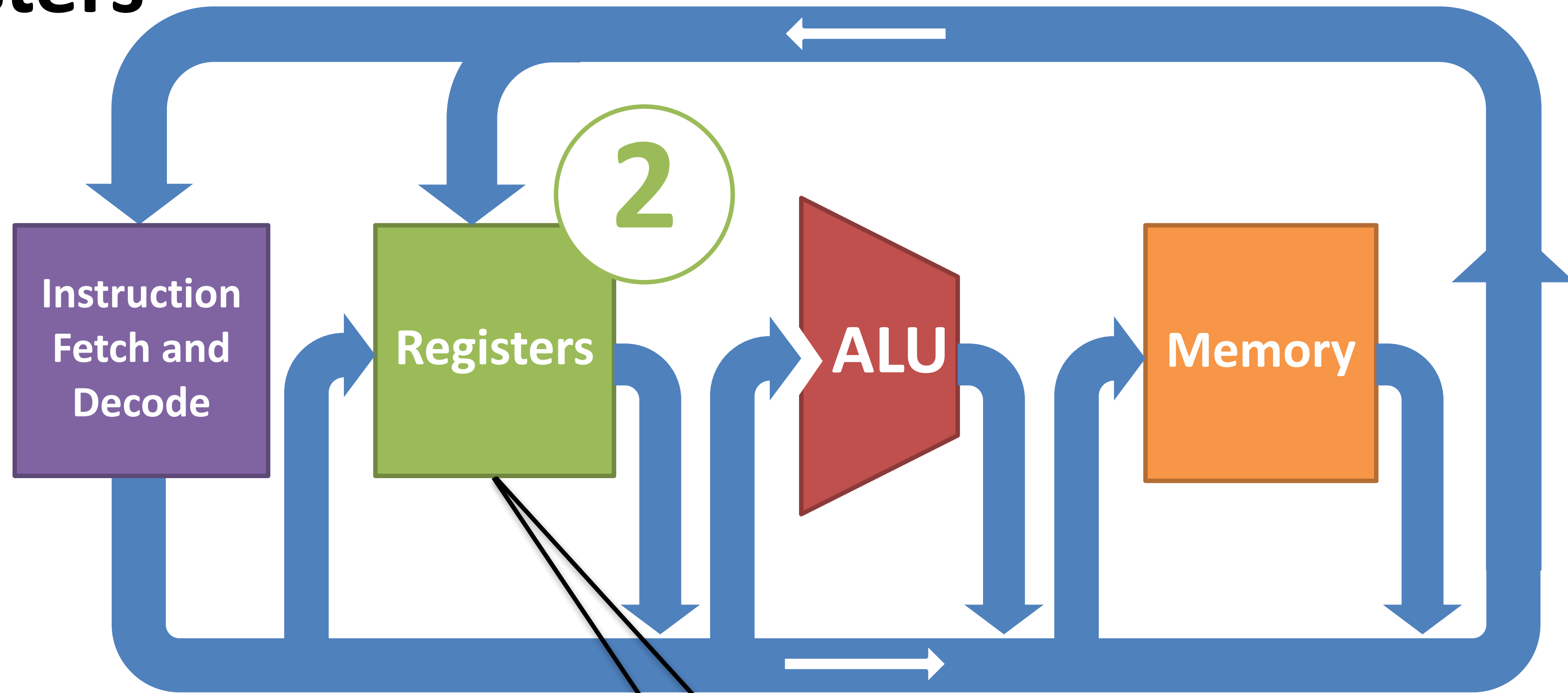
Assembly code (later this semester):   **addq** **%rdi**,**%rsi**

A 1-nybble* register

Half a byte!

(a 4-bit hardware storage cell)

Write value

0

D   **Q**

D Flip-Flop

C $\overline{\mathbf{Q}}$

1

D   **Q**

D Flip-Flop

C $\overline{\mathbf{Q}}$

0

D   **Q**

D Flip-Flop

C $\overline{\mathbf{Q}}$

1

D   **Q**

D Flip-Flop

C $\overline{\mathbf{Q}}$
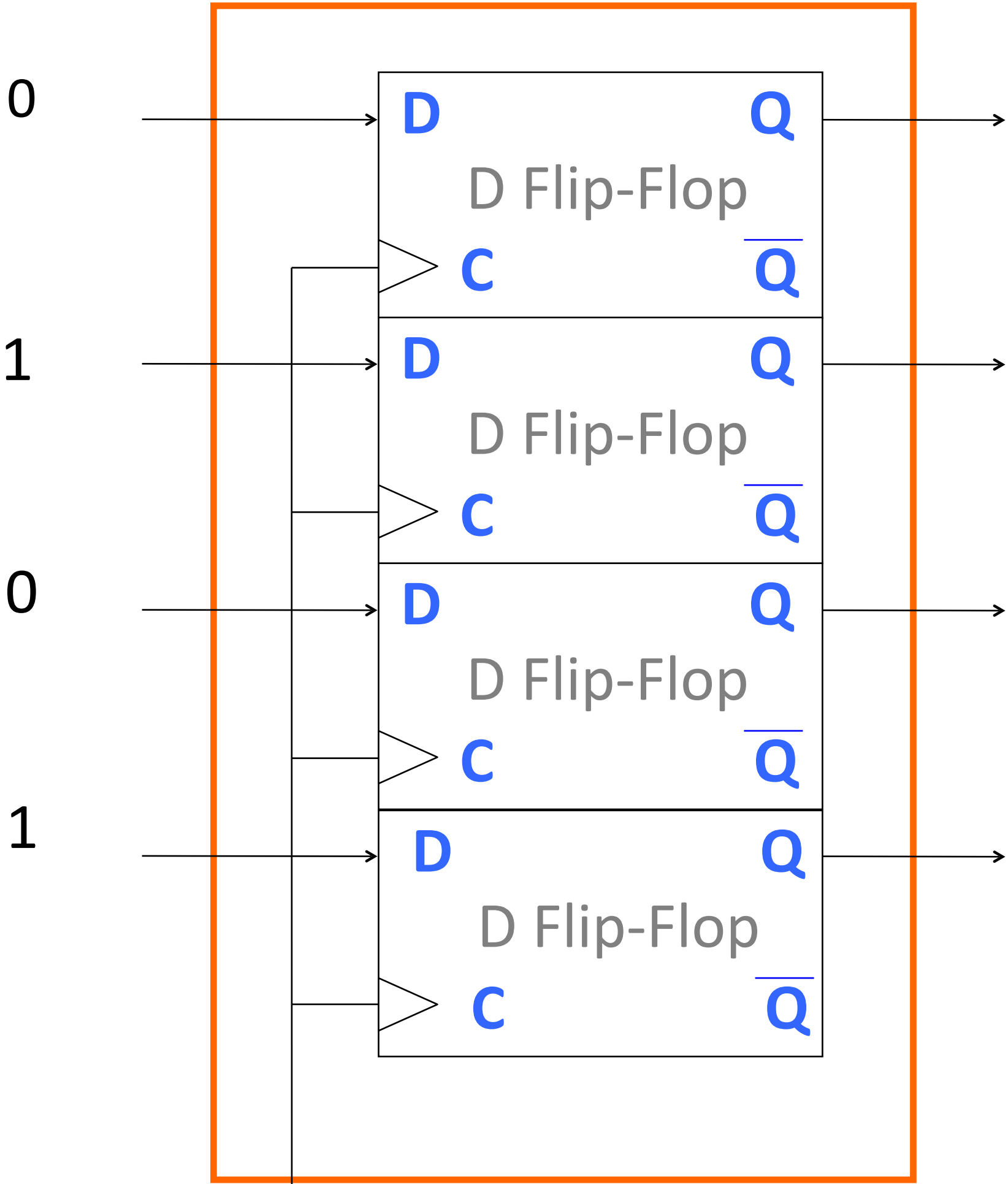
Shared clock

write control

Clock

4-bit register

C

4   4

Clock line may be indicated

Register file

Register address #1 — r

Register address #2 — r

Register address #3 — r

Write data — w

Write port

Write?
0 = read
1 = write

Read data 1 — w

Read data 2 — w

**Read ports
Why 2?**

r = log$_2$ number of registers
w = bits in word
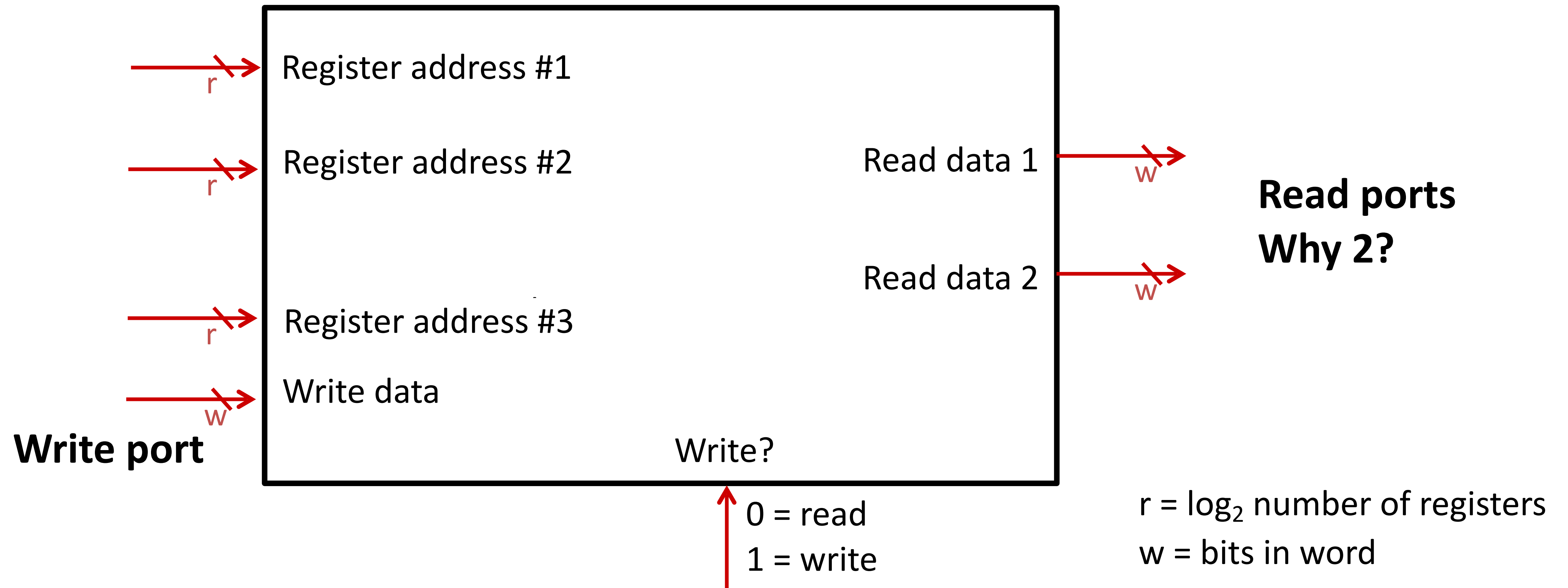
Array of registers, with register selectors, write/read control,
input port for writing data, output ports for reading data.

Read ports (data out)

Register address #1
($log_2 \, k$ bits)

Register 0

Register 1

. . .

Register k - 2

Register k - 1

n

Data read from
register address #1
(n bits)

Register address #2
($log_2 \, k$ bits)

n

Data read from
register address #2
(n bits)

Read ports (data out)

Register address #1
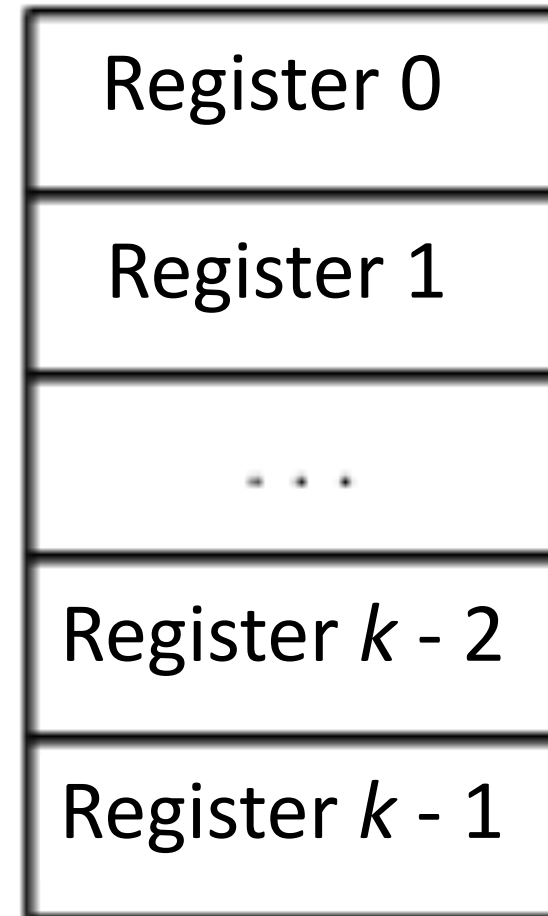($log_2 k$ bits)

Register 0

Register 1

. . .

Register k - 2

Register k - 1

Mux

n

Data read from register address #1
(n bits)

Register address #2
($log_2 k$ bits)

Mux

n

Data read from register address #2
(n bits)

Write port
(data in)

write control

clock

Address of register
to write to
($log_2 k$ bits)

m-to-2^m
decoder

0
1

k - 2

k - 1

C

Register 0

D

C

Register 1

D

⋮

C

Register k - 2

D

C

Register k - 1

D

Data to write
(n bits)

n

Write port (data in)

write control

clock

Address of register to write to ($log_2\ k$ bits)

m-to-2^m decoder

0
1
k - 2
k - 1

C
Register 0
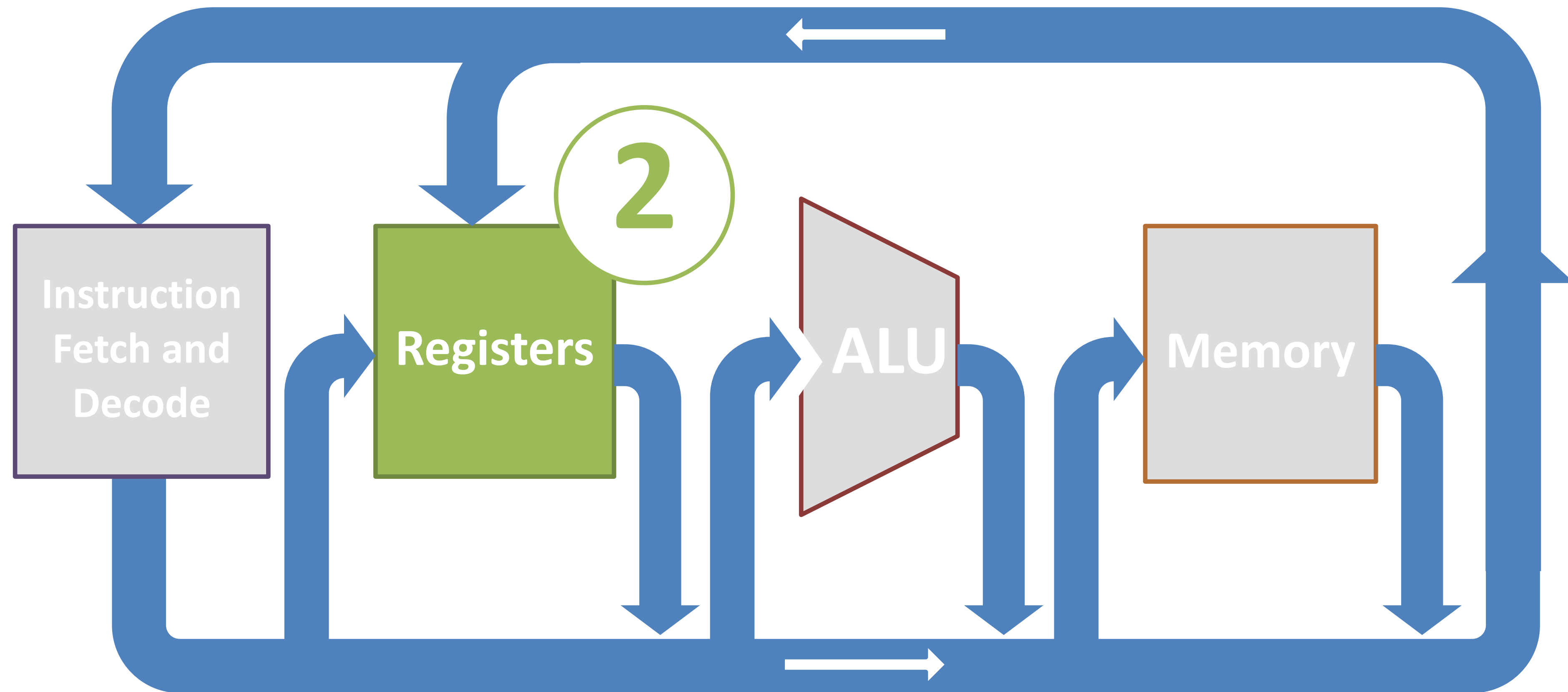D

C
Register 1
D

C
Register k - 2
D

C
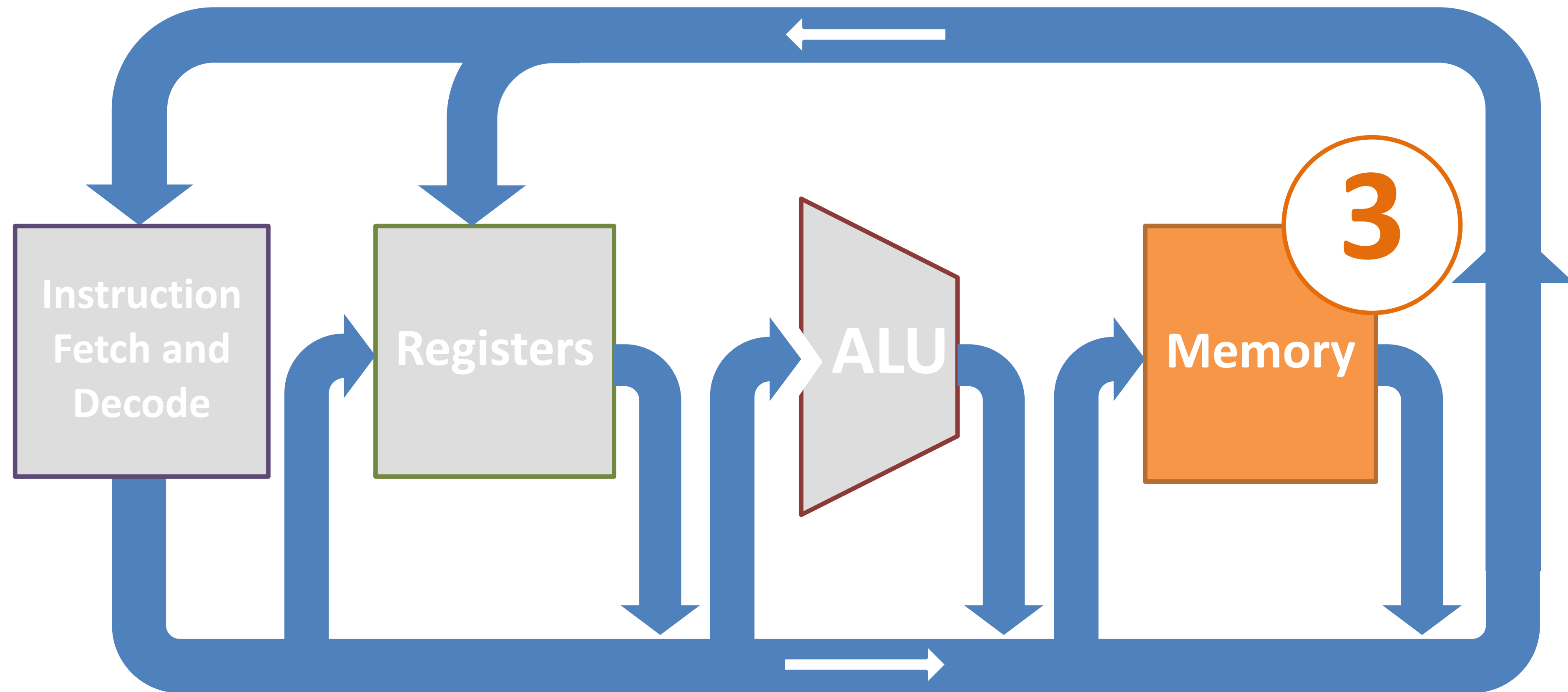Register k - 1
D

Data to write (n bits)

n

Registers summary

- For our purposes: implemented with flip-flops
- Very fast access
- Limited in size:
  - Need an m-to-2^m decoder
  - CPUs typically have ~10s of words of register storage

Registers summary

Instruction Fetch and Decode

Registers

ALU

Memory

3

- We'll think about at a higher level of abstraction
- Designed to handle a much larger amount of data
  - CPUs can have millions-billions of words of memory storage

RAM (Random Access Memory)

$\log_2 A$

B

Data In

Address

__A__ x __B__ RAM

Write

Enable
1

Data Out
B
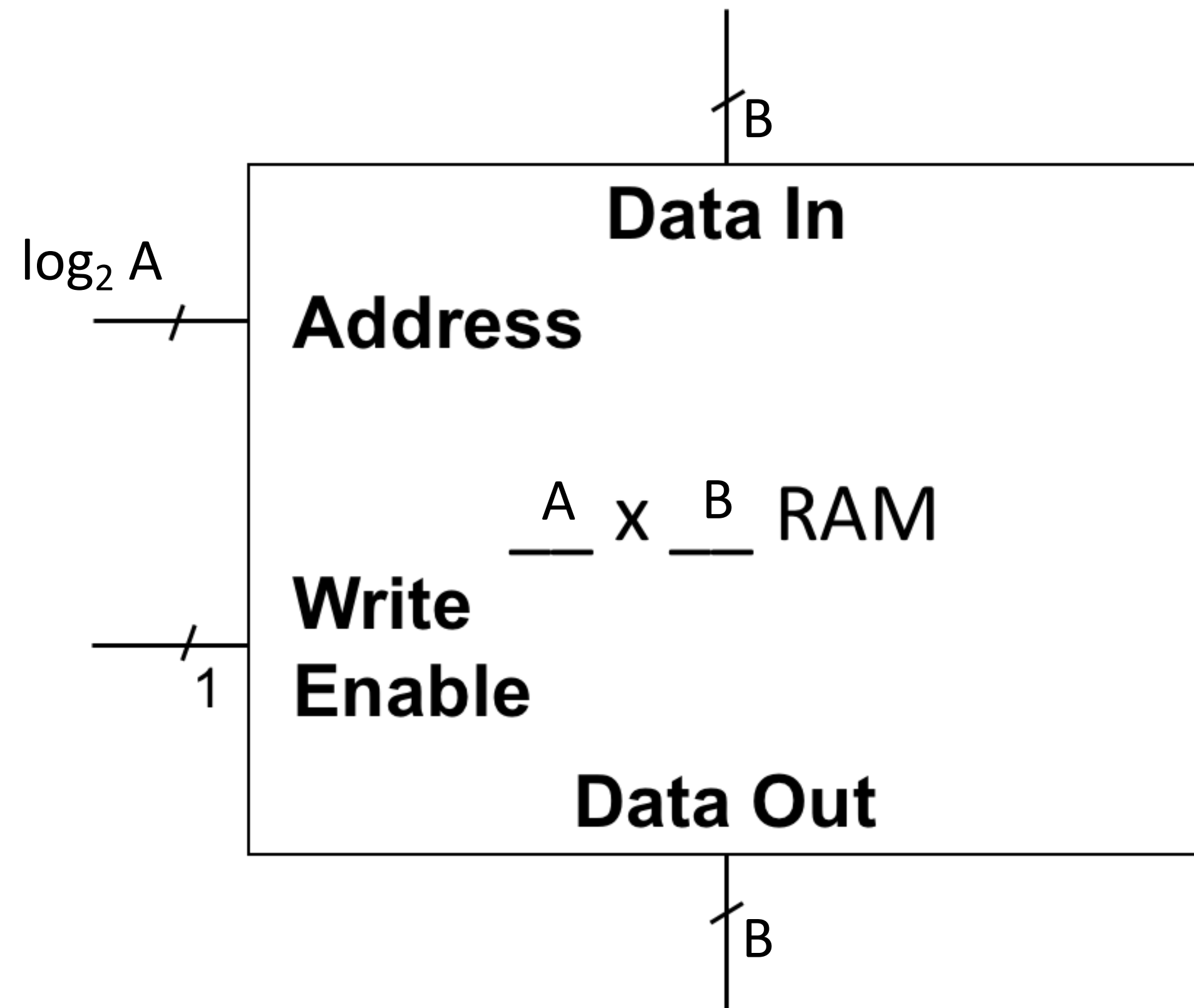
- A is number of words in RAM
- Specify the desired word by an address of size $\log_2 A$
- B is the width of each word (in bits)

16 x 4 RAM

4-bit address

1101

4 to 16 decoder

data out