Sequential Circuits: Memory, Finite State Machines

Becker/Molitor, Chapter 11.3.1
Harris/Harris, Chapters 3.3, 3.4, 5.5

Jan Reineke
Universität des Saarlandes

Motivation: Sequential circuits

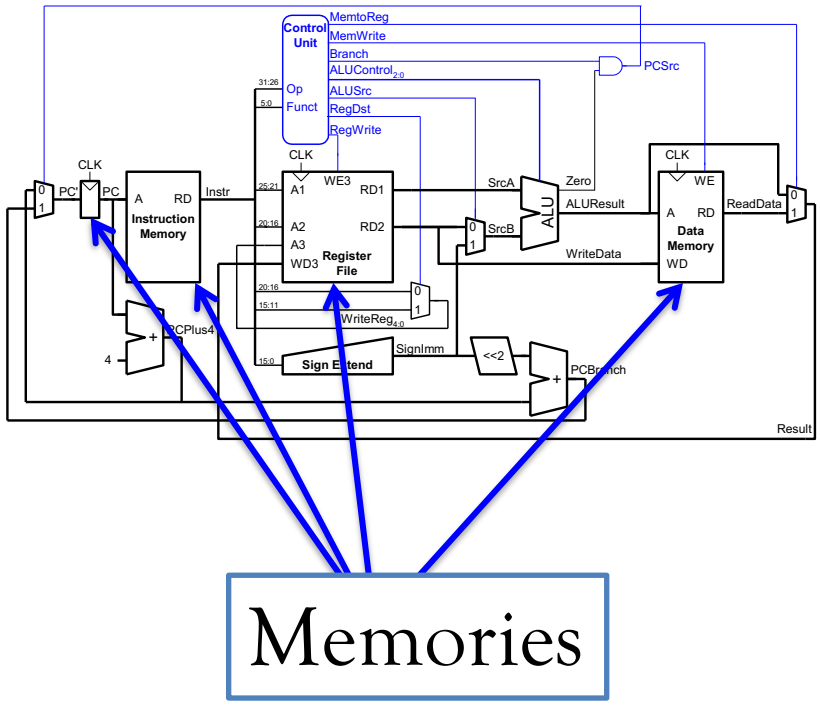
So far: combinatorial circuits = acyclic circuits

- Compute the same fixed Boolean function
- **Not** powerful enough to compute all computable functions

→ Computers process programs step-by-step:

- Fetch-decode-execute cycle
- Need memory to:
  - store programs and data
  - store intermediate results

Roadmap: Computer architecture



1. Combinatorial circuits: Boolean Algebra/Functions/Expressions/Synthesis
2. Number representations
3. Arithmetic Circuits: Addition, Multiplication, Division, ALU

4. **Sequential circuits: Flip-Flops, Registers, SRAM, Moore and Mealy automata**
5. Verilog

6. Instruction Set Architecture
7. Microarchitecture

8. Performance: RISC vs. CISC, Pipelining, Memory Hierarchy

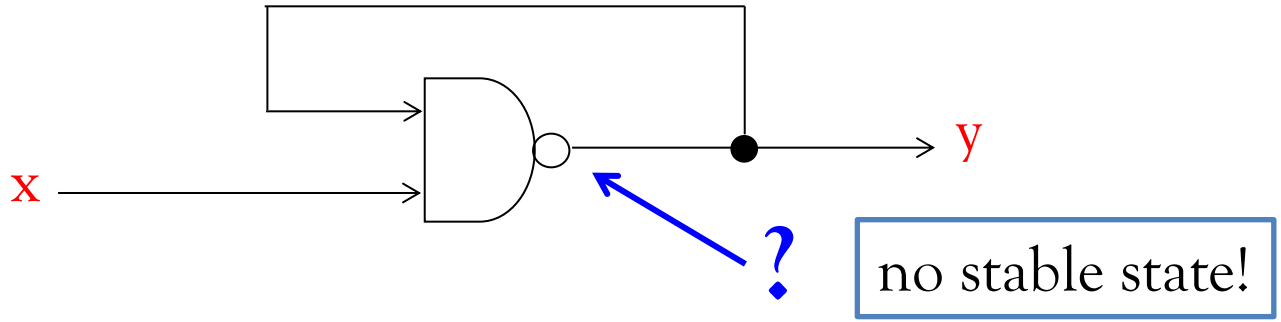
From combinatorial to sequential circuits

So far only **combinatorial circuits**:

$$C = (X_n, G, typ, IN, Y_m),$$

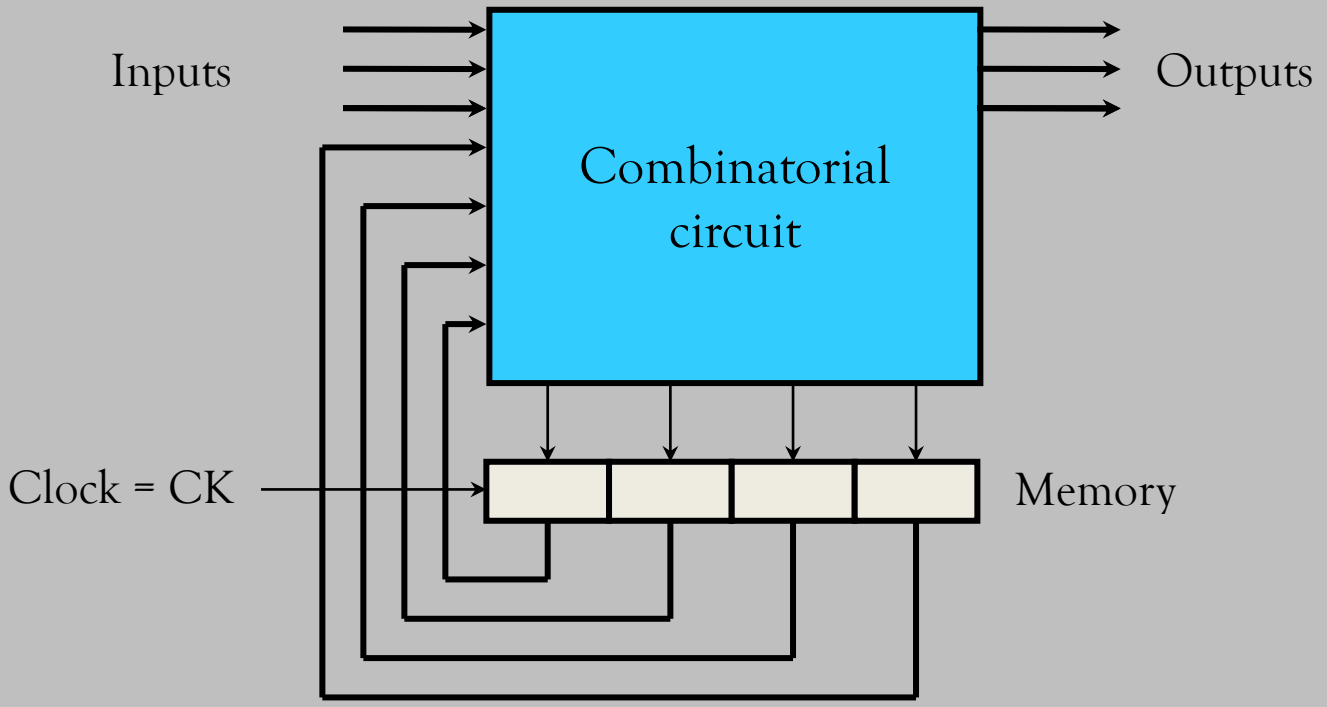
i.e., G was **acyclic**.

What happens if G is **cyclic**?



Circuits like this one are required to build storage elements!

Sequential circuits = (combinatorial) circuits + memory

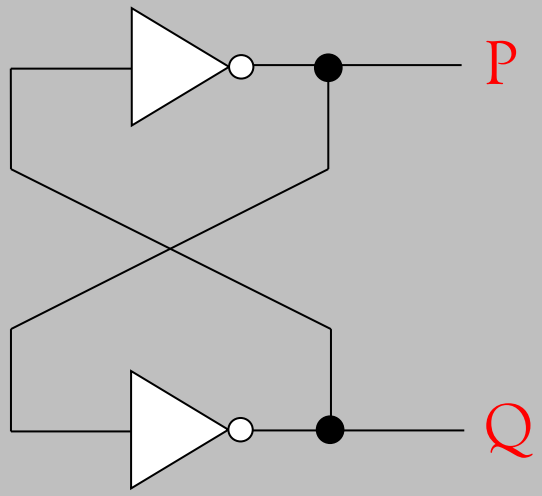


Properties:

- Every cycle contains storage element
- Separation between circuits and storage elements
- Implement finite state automata (Moore or Mealy, more on these later)

MEMORY CELLS

Example: Cyclic circuit

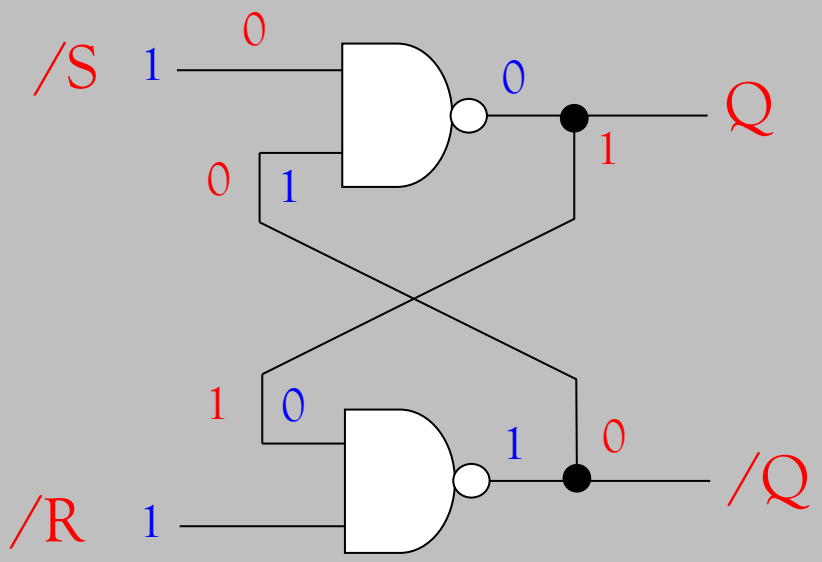


Which values can P and Q take?

How can the values of P and Q be changed?

(NAND) SR latch

Transition: Stable state Q = 0 → stable state Q = 1:



1. At time t_0 **lower** /S and at $t_0 + x$ **raise** it again (we call this a pulse)
2. After propagation delay $t_{P/SQ}$ we have Q = 1.
3. After propagation delay $t_{P/S/Q}$ we have /Q = 0.

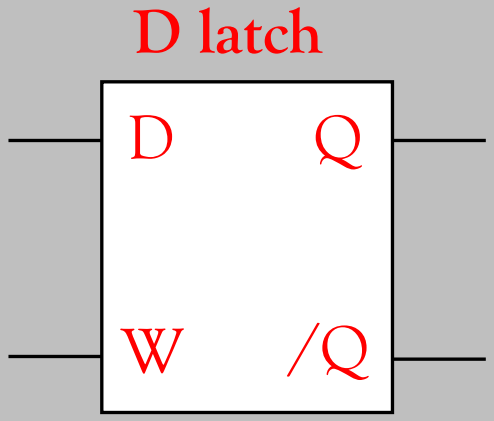
"**active low**" terminology:
/S and /R **activated**
at **low** input voltage

Circuit with **two stable states**,
adequate to store $\log_2 2 = 1$ **bit**.

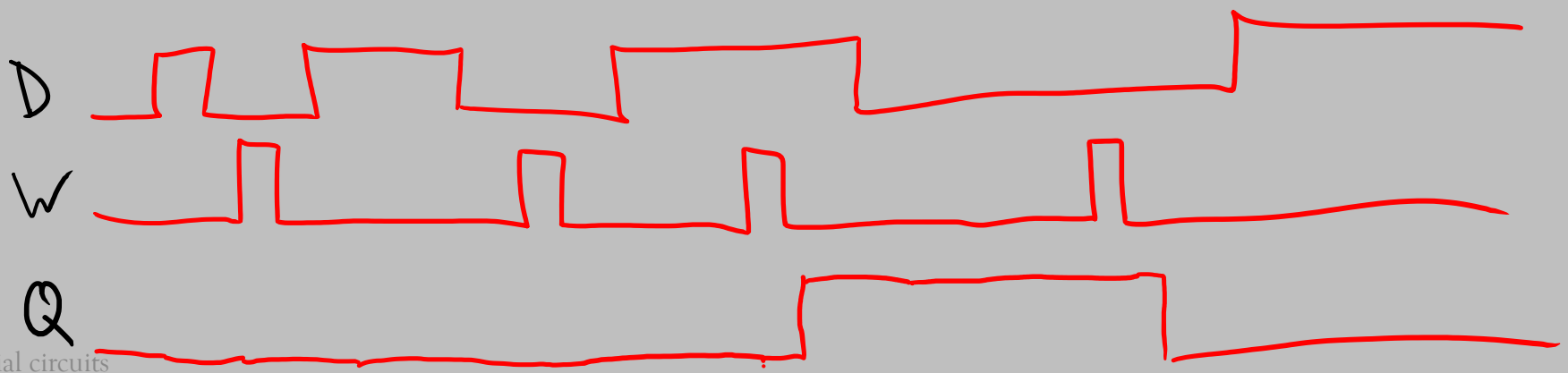
D latch: Behavior

To store an incoming data value D
via a pulse (interval between raising and lowering) at W.

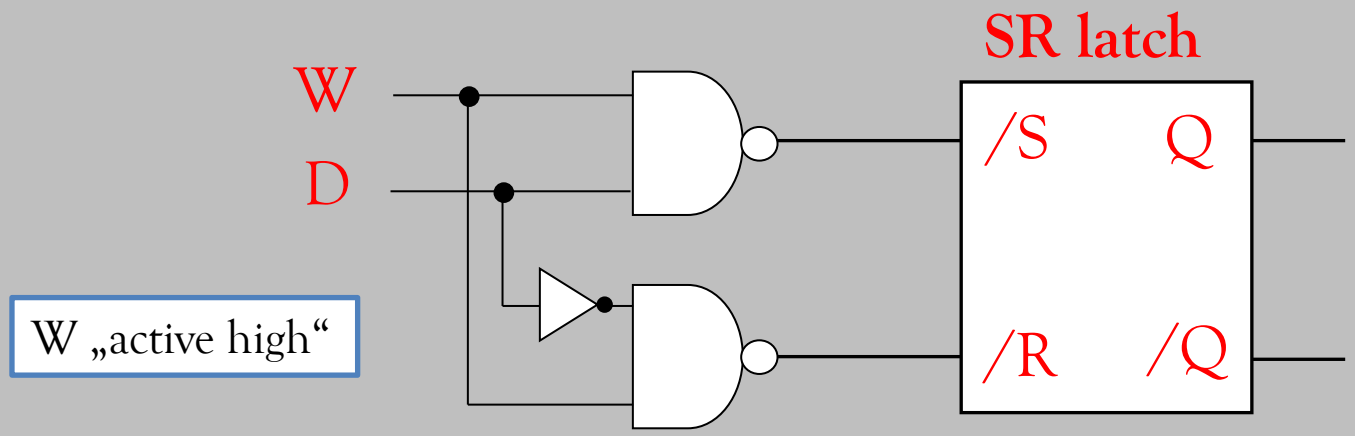
Symbol:



Behavior:



D latch: Implementation



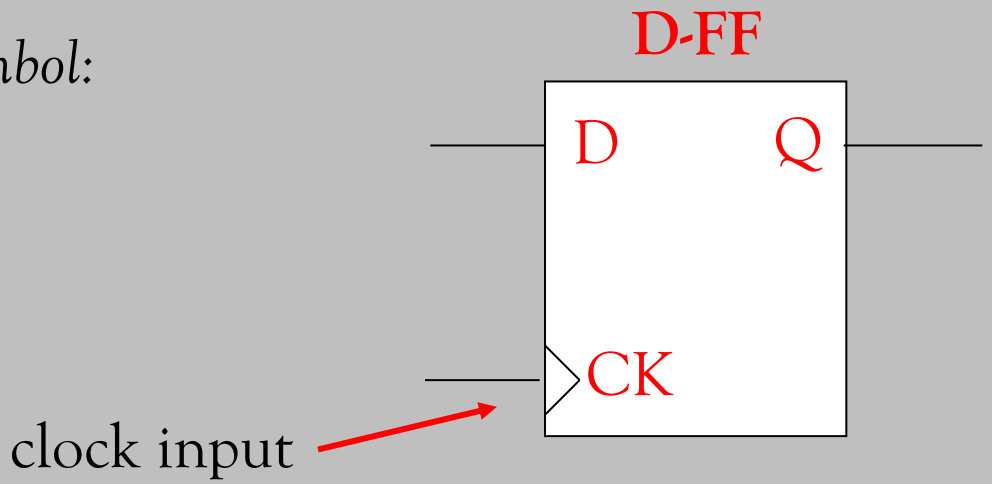
1. Case: W = 0 → /S = /R = 1

2. Case: W = 1 → /S = NAND(1, D) = D‘
and /R = NAND(1,D‘) = D

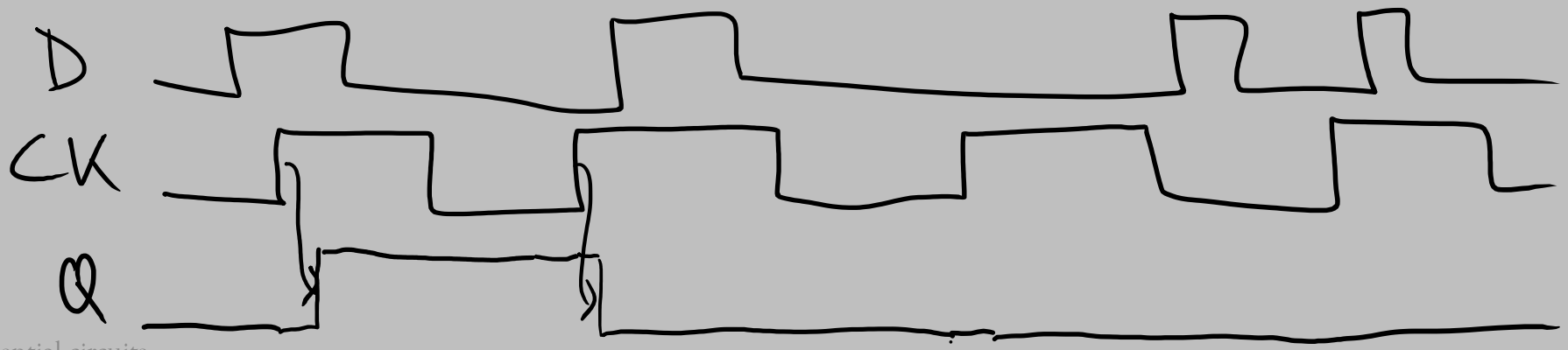
D flip-flop: Edge-triggered

Controlled via a **rising edge** of a signal (usually of a clock):

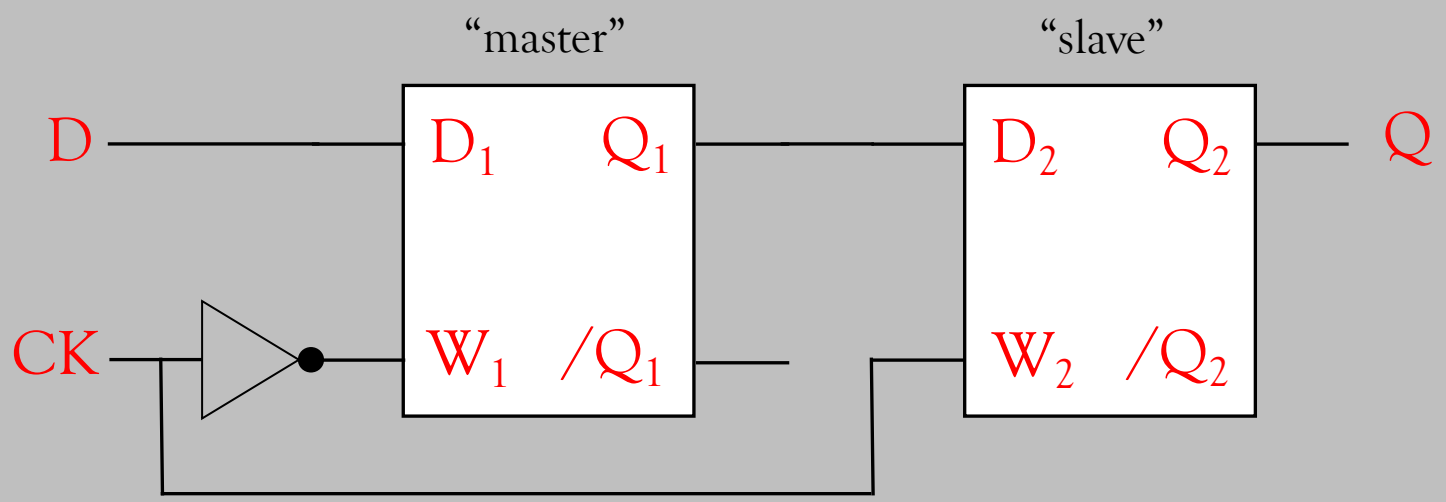
Symbol:

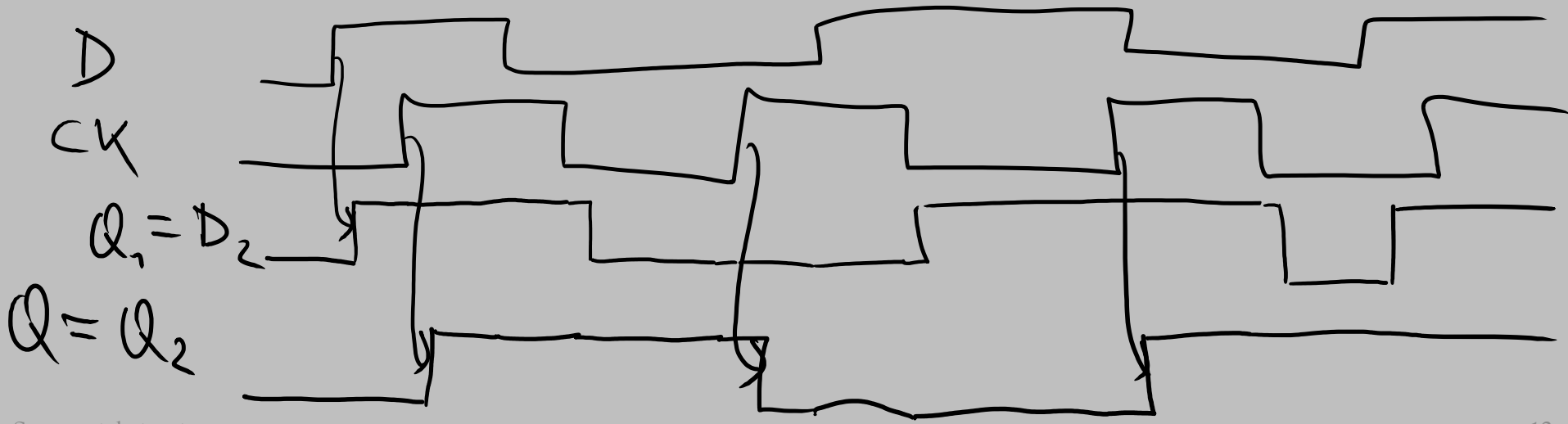


Behavior:



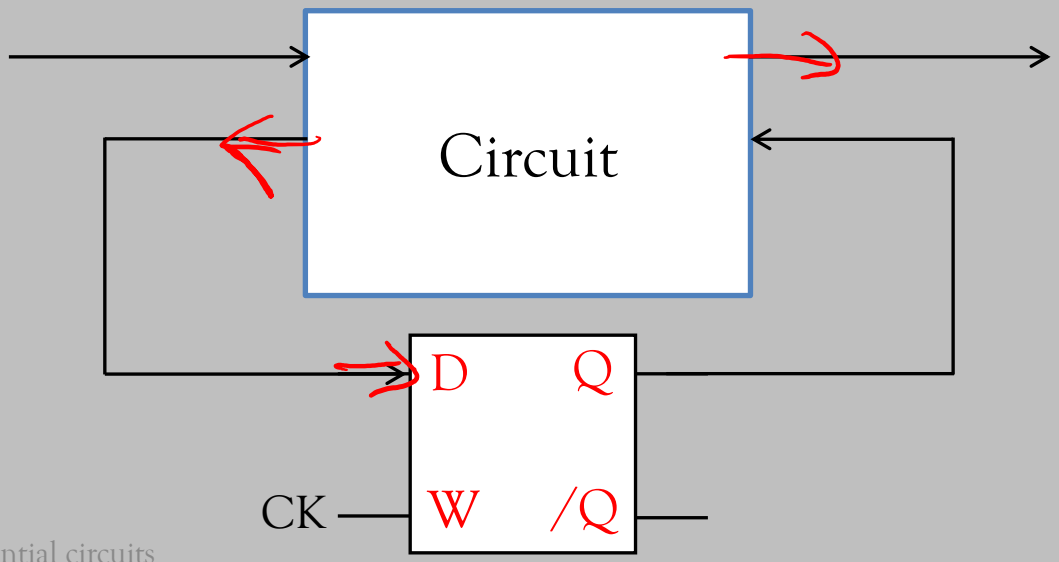
D flip-flop: Implementation





Latch vs Flip-flop

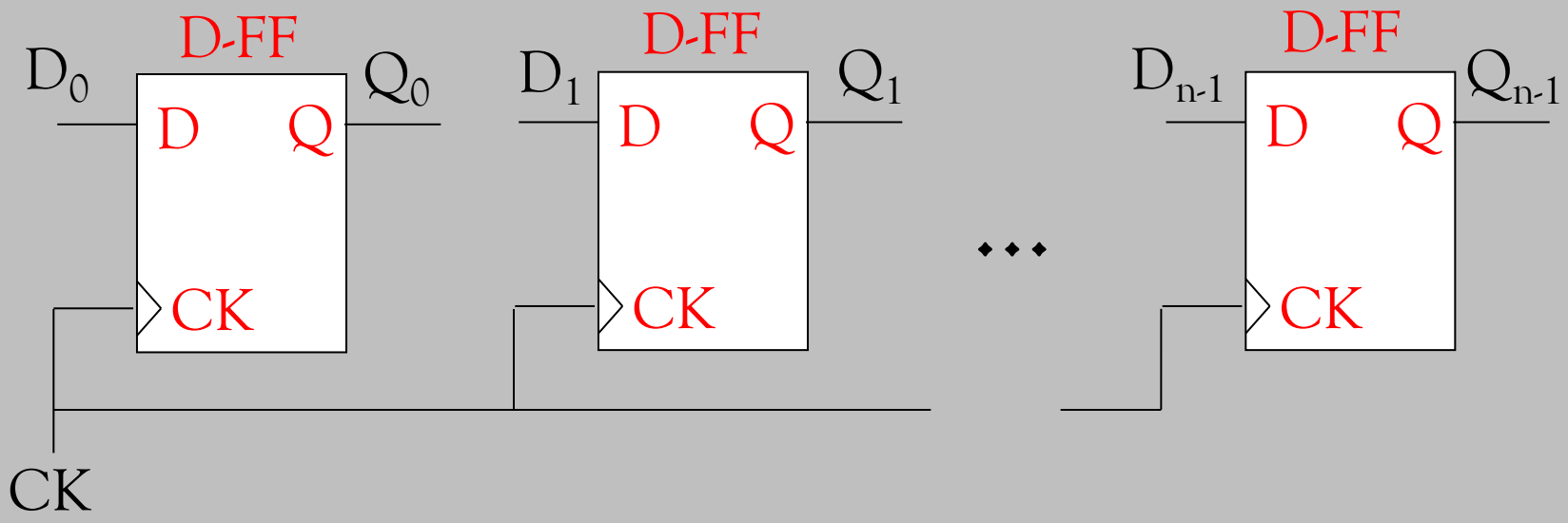
- **Latch** = level-triggered
- **Flip-flop** = edge-triggered
  - *Advantage*:
    More predictable in circuits with feedback



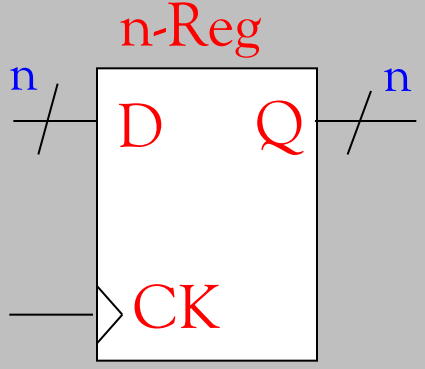
With a **latch** the behavior depends on the precise timing of the circuit!

With a **flip-flop** the circuit just has to be "fast enough".

Derived circuit: n-bit register

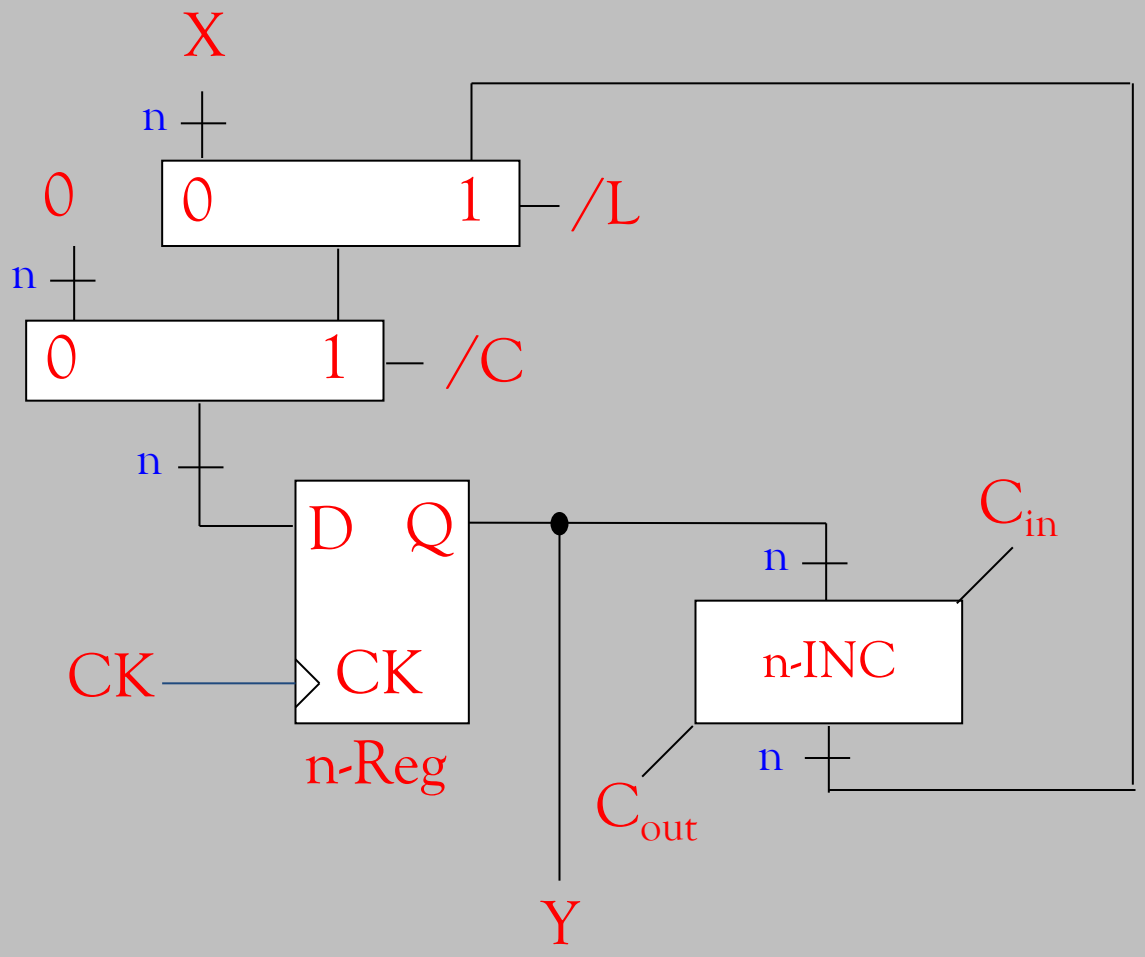


Symbol:



A simple sequential circuit: An n-bit counter

/C clear, /L load, X input, Y output

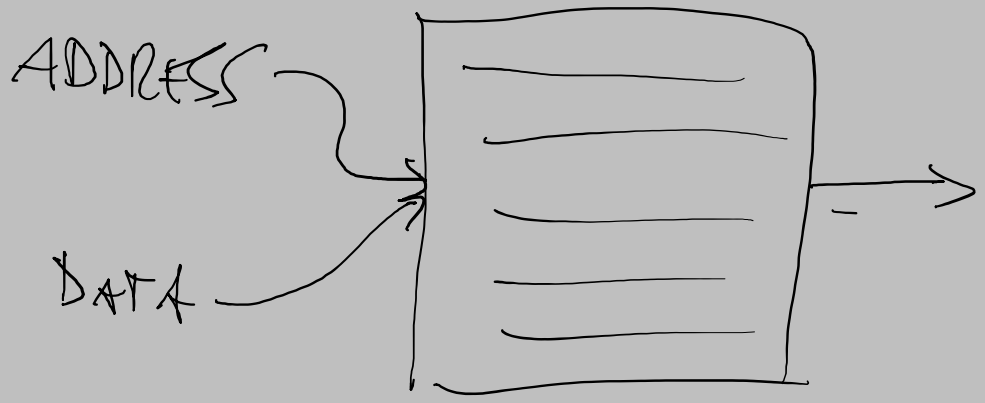


STATIC RAM AND DYNAMIC RAM

Derived circuit: Random access memory (RAM)

Characteristics:

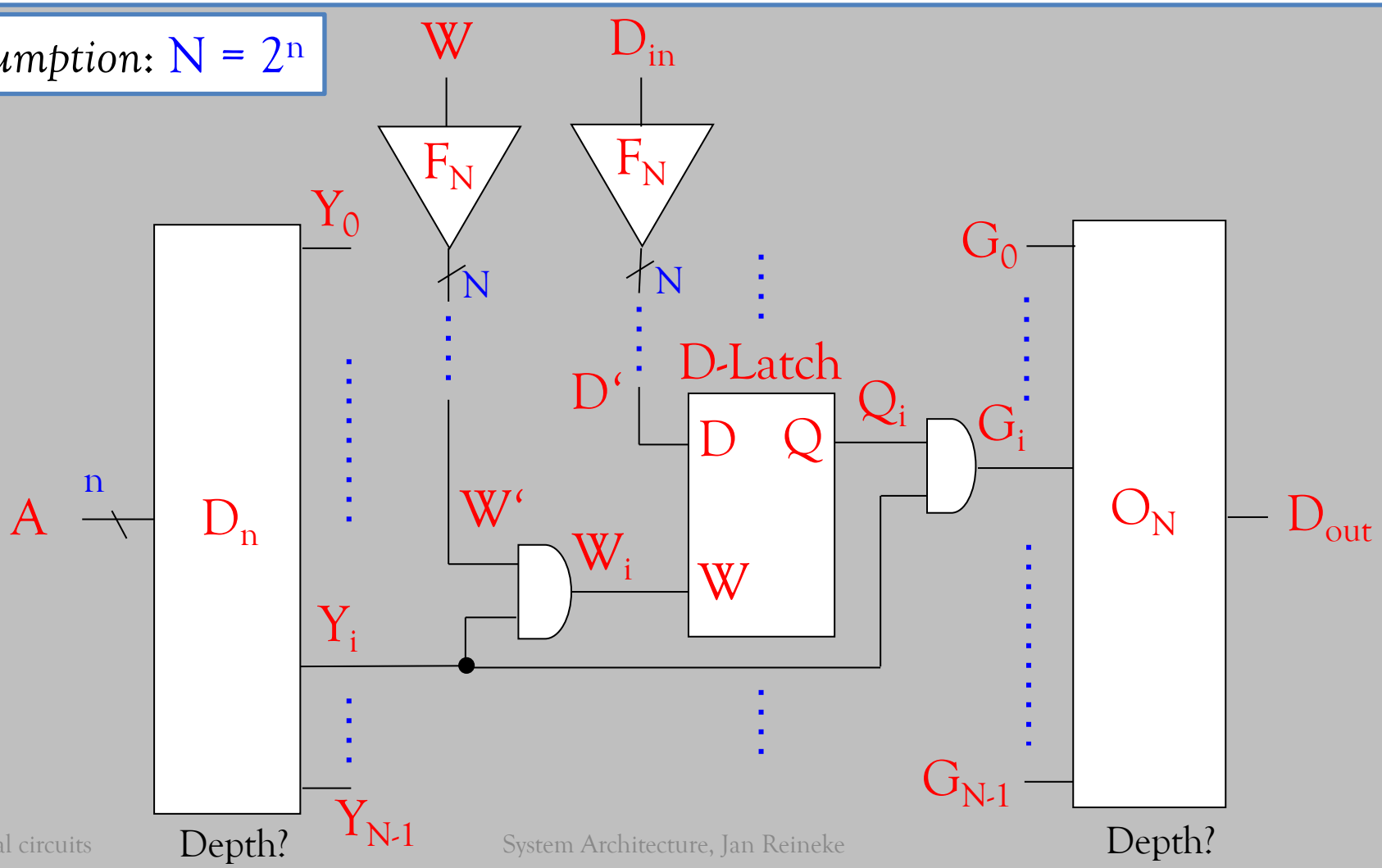
- linear array of storage cells
- single storage cell selected by **adress**
- *Reading* and *writing* possible
- **volatile** = not persistent = loses its state if power is off.



Schematic of an SRAM (a single addressed bit)

Inputs: A address (i = <A>), W write, D_{in} data input,
Output: D_{out} data output *Components*: D_n decoder, O_N or gate

Assumption: $N = 2^n$



Decoder

Definition:

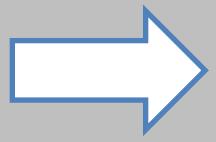
An **n-bit decoder** is a circuit that computes the following Boolean function $f : B^n \rightarrow B^N$, with $N = 2^n$:

$$y_i = f(x_{n-1} \ldots x_0)_i \Leftrightarrow \left(\langle x_{n-1} \ldots x_0 \rangle = i \right)$$

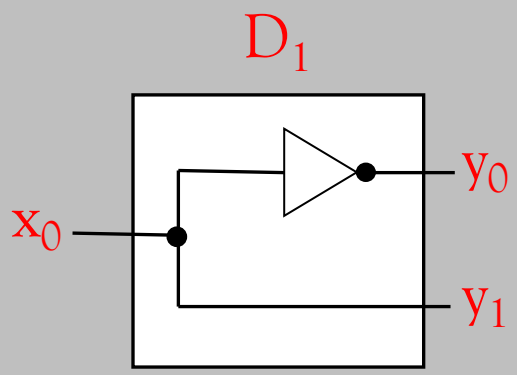
Base case: 1-bit decoder

Truth table:

x_0	y_0	y_1
0	1	0
1	0	1



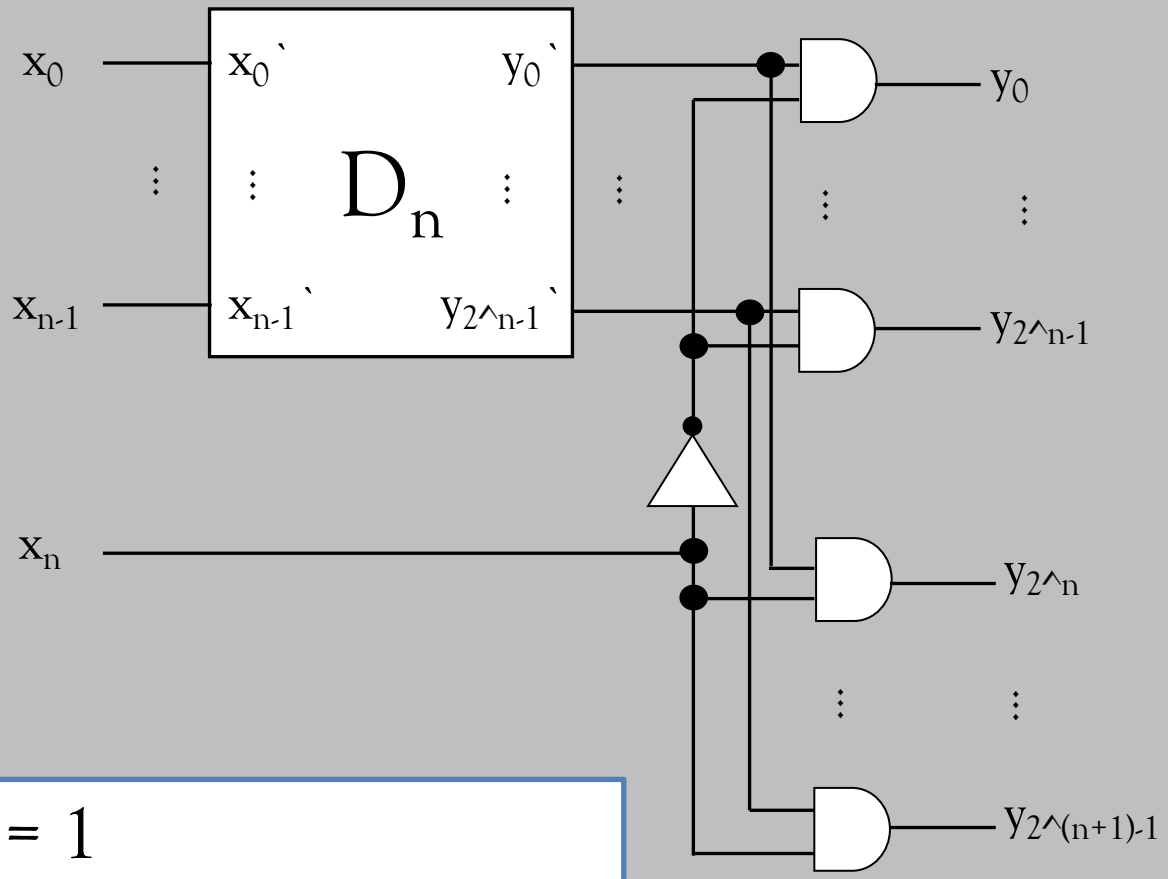
Circuit:



Properties:

- 1 input, $2^1 = 2$ outputs
- *Depth*: depth(D_1) = 1
- *Cost*: C(D_1) = 1

n-Bit decoder: Recursive construction (from D_n to D_{n+1})



Cost?

$C(D_1) = 1$

$C(D_{n+1}) = C(D_n) + 2^{n+1} + 1$

→ $C(D_n) = 2^{n+1}+n-4$

Depth?

$depth(D_1) = 1$

$depth(D_{n+1}) = depth(D_n)+1$

→ $depth(D_n) = n$

Correctness of an n-bit decoder

Proof by induction over n:

Base case (n=1): ✓

Induction step (n → n+1):

Need to show for all $0 \le i < 2^{n+1}$:

Case distinction:

$$y_i \Leftrightarrow \left(\left\langle x_n x_{n-1} \ldots x_0 \right\rangle = i\right)$$

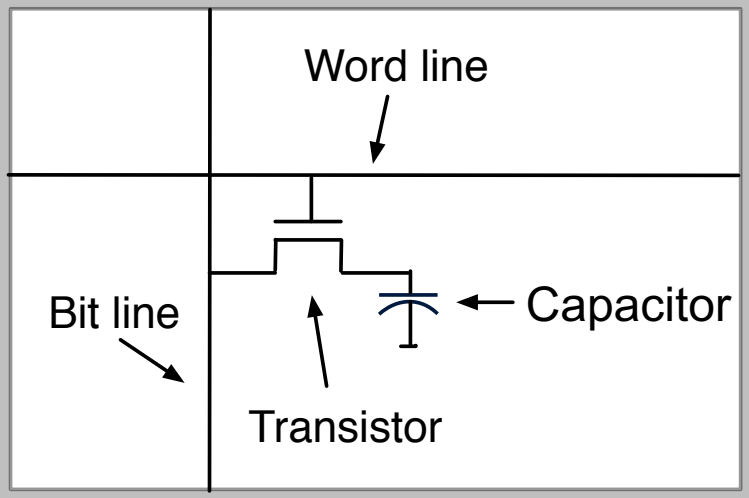
1. $0 \le i < 2^n$: by construction we have:

$$y_i \Leftrightarrow y'_i \wedge \overline{x}_n \overset{\text{I.H.}}{\Leftrightarrow} \left(\left\langle x_{n-1} \ldots x_0 \right\rangle = i\right) \wedge \overline{x}_n \Leftrightarrow \left(\left\langle x_n x_{n-1} \ldots x_0 \right\rangle = i\right)$$

2. $2^n \le i < 2^{n+1}$: by construction we have:

$$y_i \Leftrightarrow y'_{i-2^n} \wedge x_n \overset{\text{I.H.}}{\Leftrightarrow} \left(\left\langle x_{n-1} \ldots x_0 \right\rangle = i - 2^n\right) \wedge x_n \Leftrightarrow \left(\left\langle x_n x_{n-1} \ldots x_0 \right\rangle = i\right)$$

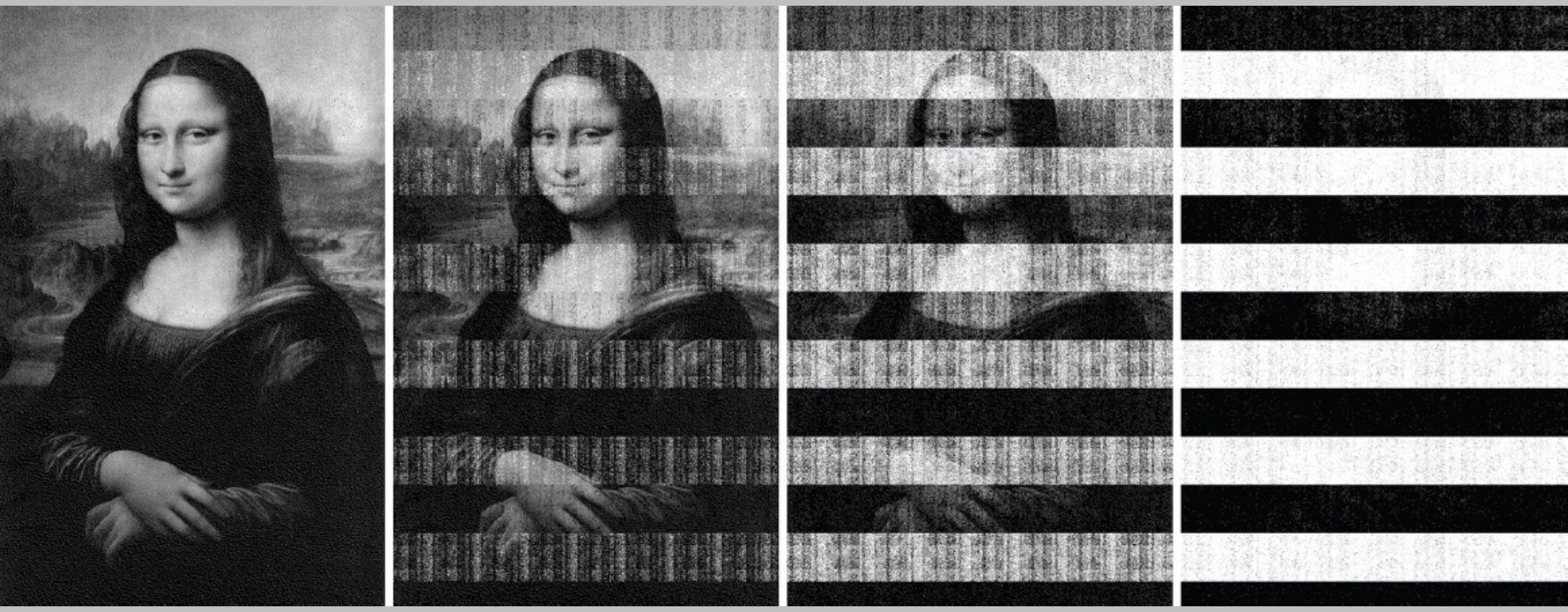
Alternative: Dynamic RAM (DRAM)



- **Word line** active, when bit is read or written to, transistor transmits
- *Writing*:
  - Voltage on **bit line**
    - high for 1, low for 0
  - Pulse on **word line**
    - transfers charge to capacitor
- *Reading*:
  - Charge of capacitor is transferred via the **bit line** to a sense amplifier
    - compares with reference value to detect 0 or 1

- Capacitor "leaks" charge.
  State must be refreshed periodically; according to standard every 64 ms → thus **"dynamic"** RAM
  - temperature-dependent
  - less leakage the colder the transistor is

Anecdote: “Memory attacks”



Lest We Remember: Cold-Boot Attacks on Encryption Keys
Communications of the ACM, 2009

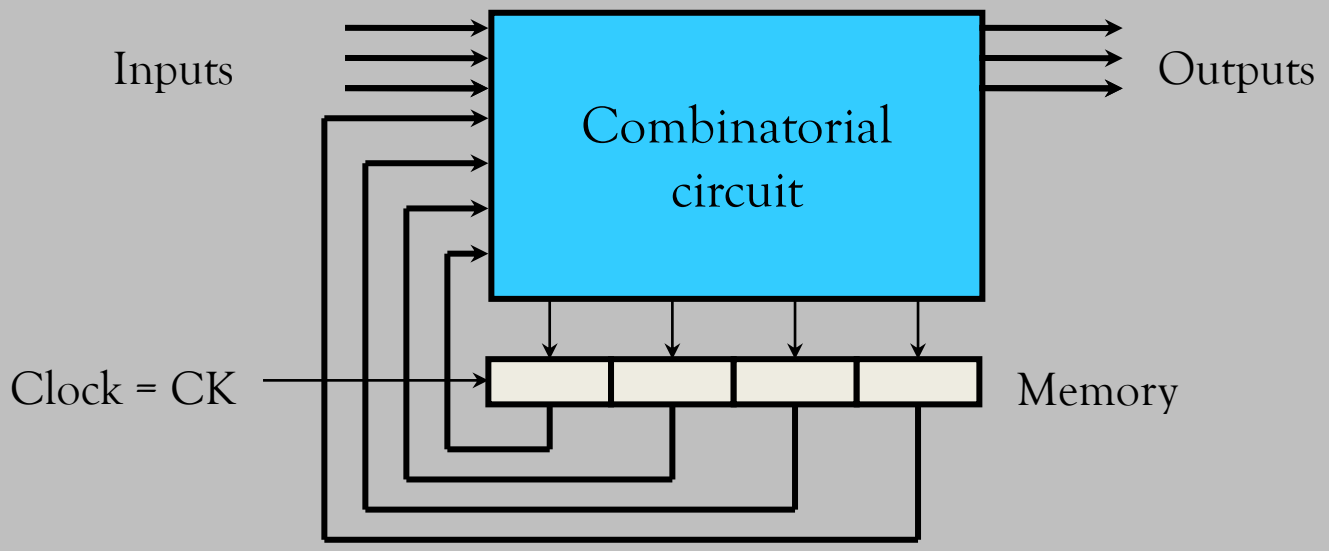
Comparison: SRAM vs DRAM

Both are **volatile**:
i.e. must expend energy to keep data

Static RAM	Dynamic RAM
requires 6 transistors per bit → lower density	requires 1 transistor and 1 capacitor per bit → higher density
faster accesses	slower accesses
	requires refresh
→ used in caches	→ used in main memory

SEQUENTIAL CIRCUITS

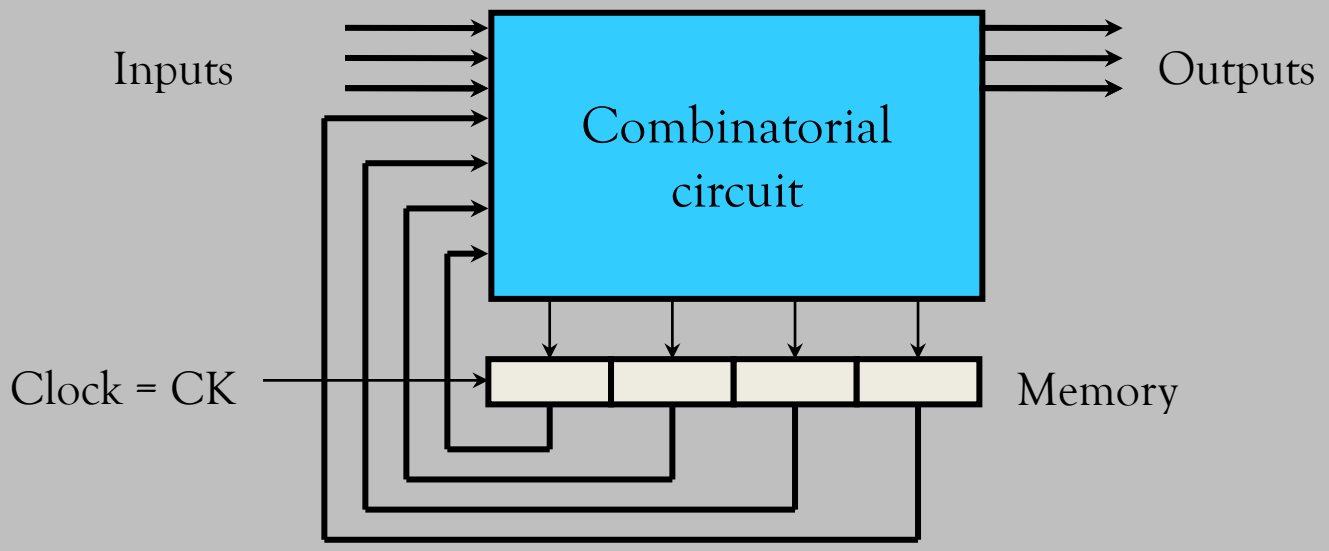
Sequential circuits = (combinatorial) circuits + memory



Properties:

- Every cycle contains storage element
- Separation between circuits and storage elements

Sequential circuits = (combinatorial) circuits + memory



Wanted:

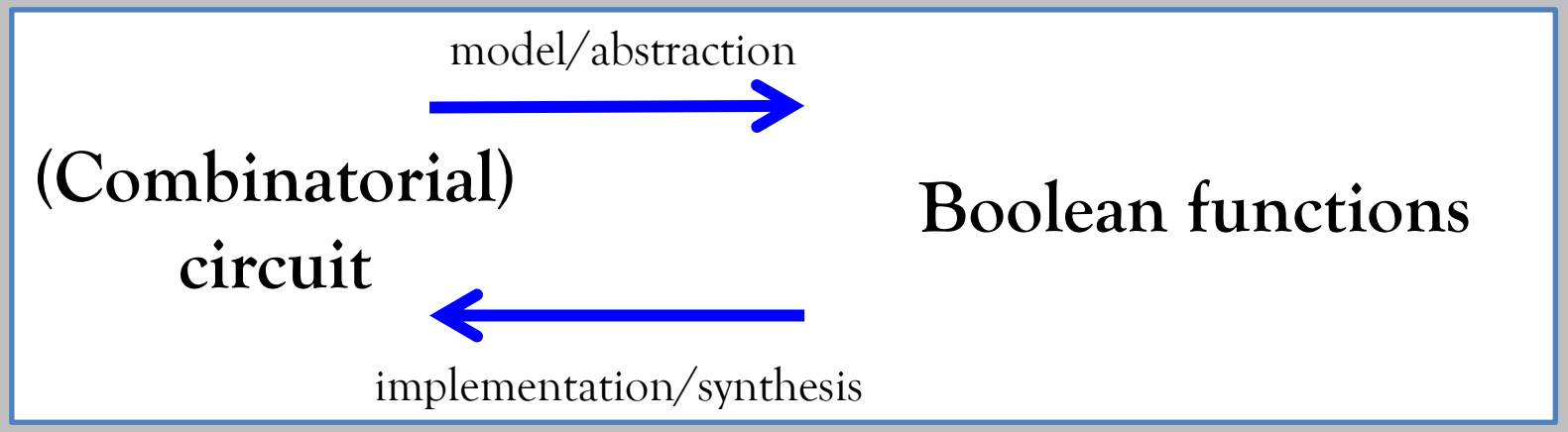
- Predictable, deterministic behavior:
  - deterministic: same output sequence on same input sequence
  - predictable: can mathematically capture input/output behavior

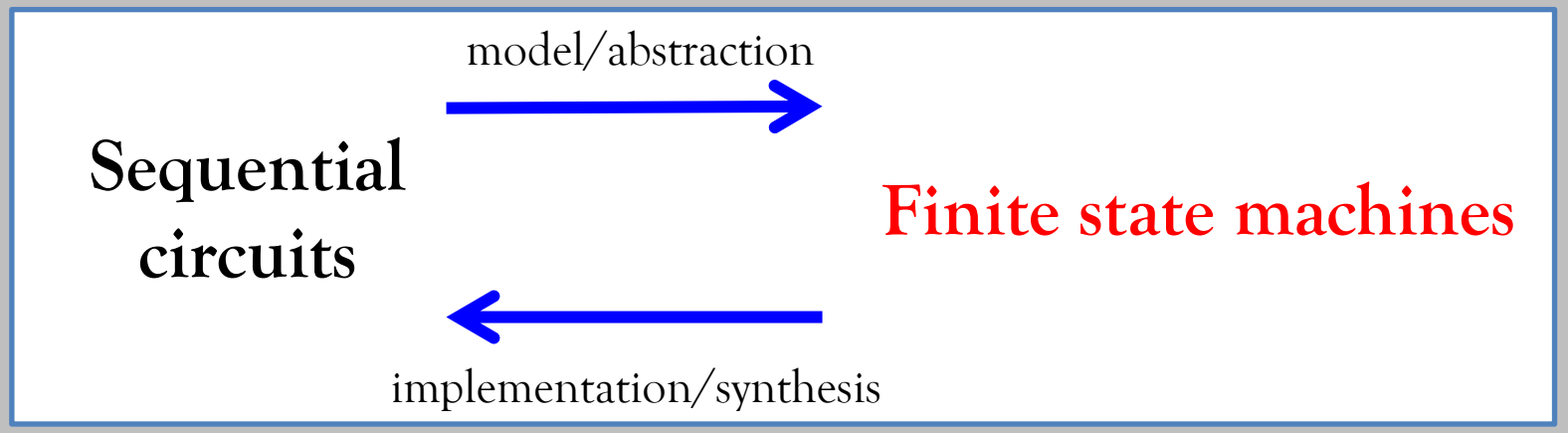
Required for that:

- Depth of combinatorial circuit < length of a clock cycle

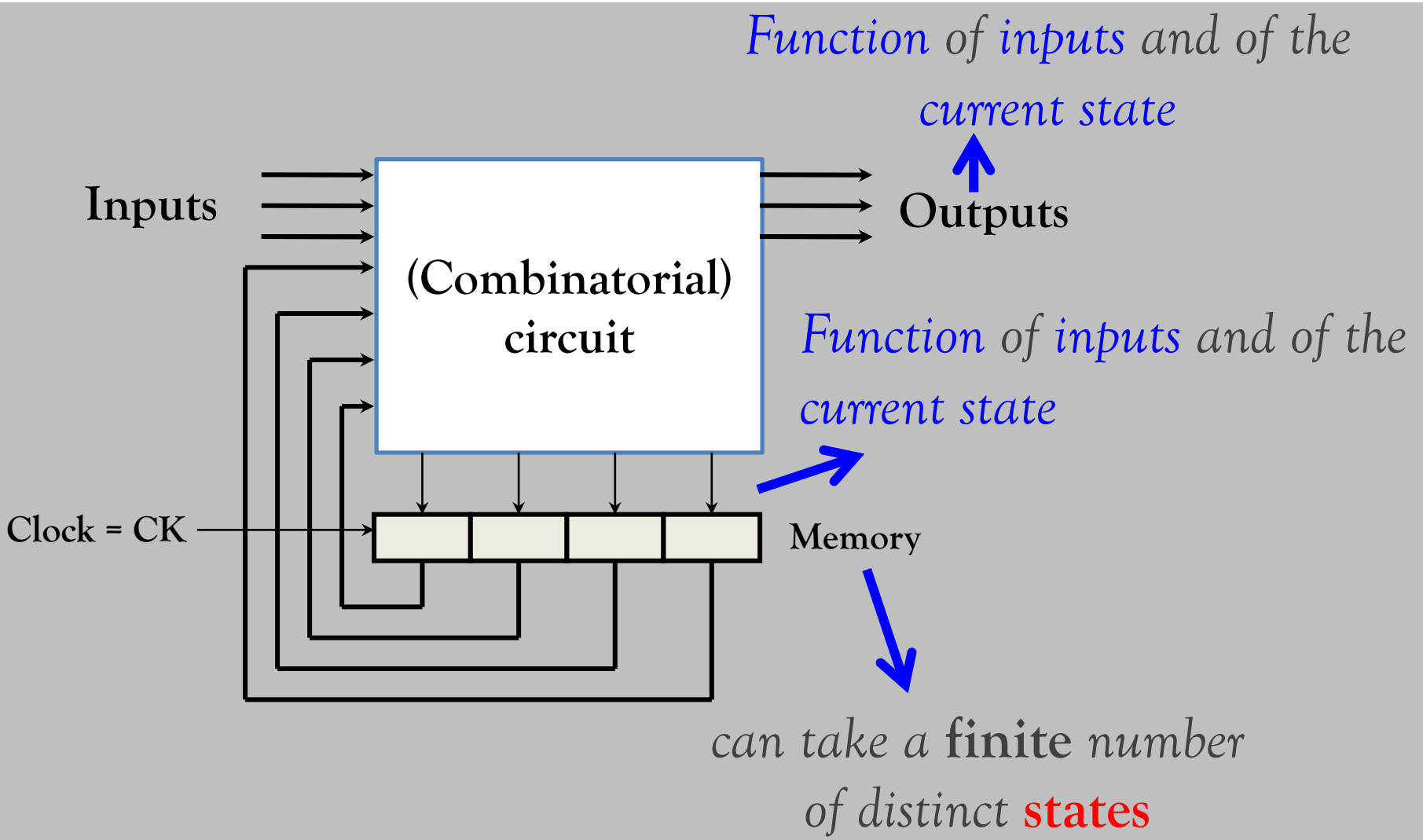
FINITE STATE MACHINES

Finite state machines as models of sequential circuits





Sequential circuits: Abstraction



Finite state machine, Mealy machine

Definition:
A Mealy machine is a 6-tuple $M=(Q, q_0, I, O, \delta, \lambda)$:

- Q is a *finite*, *non-empty* set of states,
- $q_0 \in Q$ is the initial state,
- I is a *finite*, *non-empty* input alphabet,
- O is a *finite*, *non-empty* output alphabet,
- $\delta : Q \times I \rightarrow Q$ is the transition function,
- $\lambda : Q \times I \rightarrow O$ is the output function.

Named after:
Mealy, George H. (1955), "A *method for synthesizing sequential circuits*", Bell System Technical Journal

Mealy machine: Example vending machine

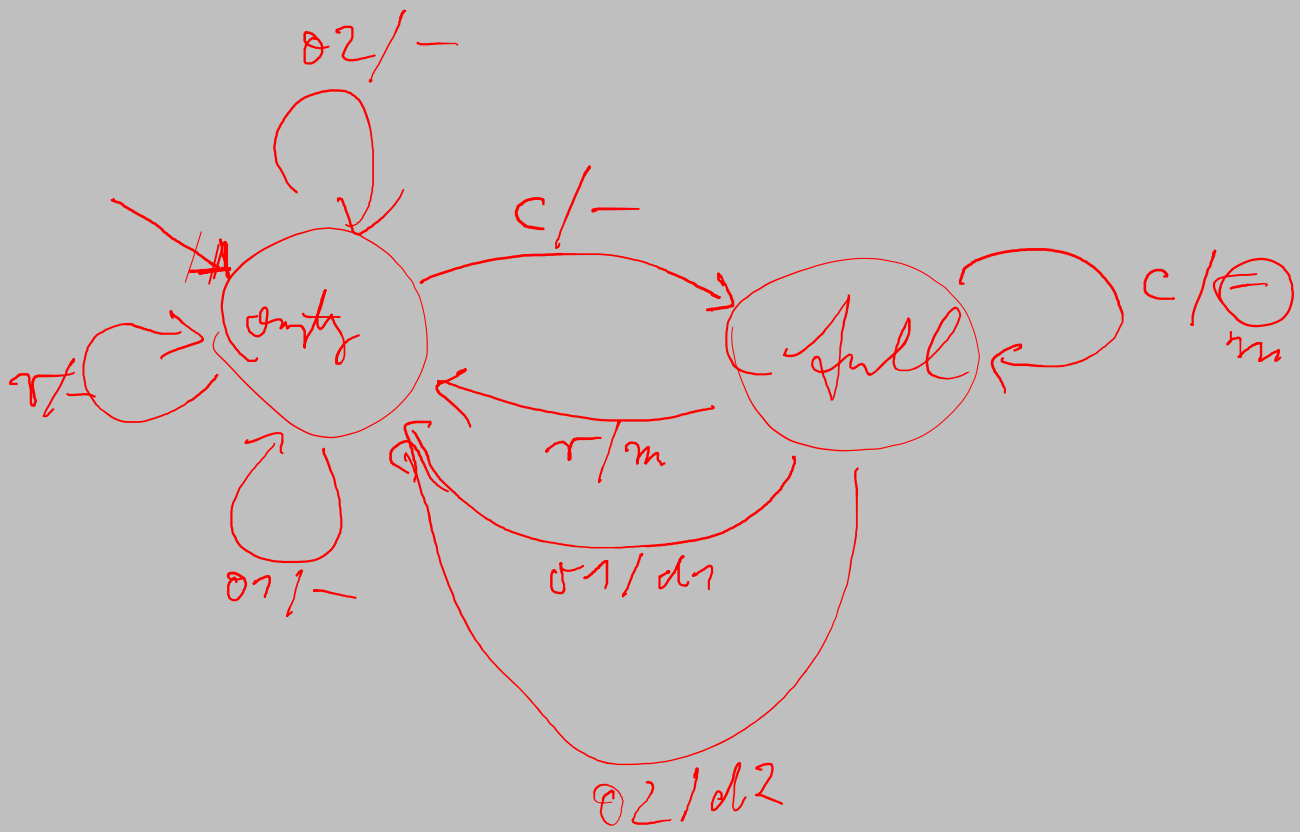
- A (simple) vending machine:
  1. Pay one euro,
  2a. Then choose one among two drinks.
  2b. Or press the return button to retrieve the money
- Other behavior is ignored by the vending machine.

Example: Vending machine

- Set of states Q?
- Input alphabet I?
- Output alphabet O?

- Q = {empty, full}, q_0 = empty
- I = {coin (c), return (r), option 1 (o1), option 2 (o2)}
- O = {no output (-), emission of money (m), emission of drink 1 (d1), emission of drink 2 (d2)}

Example: Vending machine
Transition and output function graphically



Example: Vending machine
Transition and output function

Transition function:

Q	I	δ
empty	c	full
empty	r	empty
empty	o1	empty
empty	o2	empty
full	c	full
full	r	empty
full	o1	empty
full	o2	empty

Output function:

Q	I	λ
empty	c	-
empty	r	-
empty	d1	-
empty	d2	-
full	c	-
full	r	m
full	d1	o1
full	d2	o2

From Mealy machines to Sequential circuits

How can we turn transition and output functions into Boolean functions?

From Mealy machines to Sequential circuits

1. Fix **encoding** of *states*, *inputs*, and *outputs*
2. Synthesize **circuits** for
   transition function and *output function*

Example: Vending machine
1. Encoding

- 2 states, i.e., we require at least one bit.
  Assume *empty* → 0, *full* → 1.
- 4 inputs and 4 outputs, and so we require at least 2 bits, as $4 = 2^2$.

For example:

I	X1	X2
c	0	0
r	0	1
o1	1	0
o2	1	1

O	Y1	Y2
-	0	0
m	0	1
d1	1	0
d2	1	1

Example: Vending machine
2. Synthesis of circuits

Truth table of transition function follows from encoding:

Transition function:

Q	I	δ
empty	c	full
empty	r	empty
empty	o1	empty
empty	o2	empty
full	c	full
full	r	empty
full	o1	empty
full	o2	empty

Q	X1	X2	δ
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Example: Vending machine
2. Synthesis of circuits

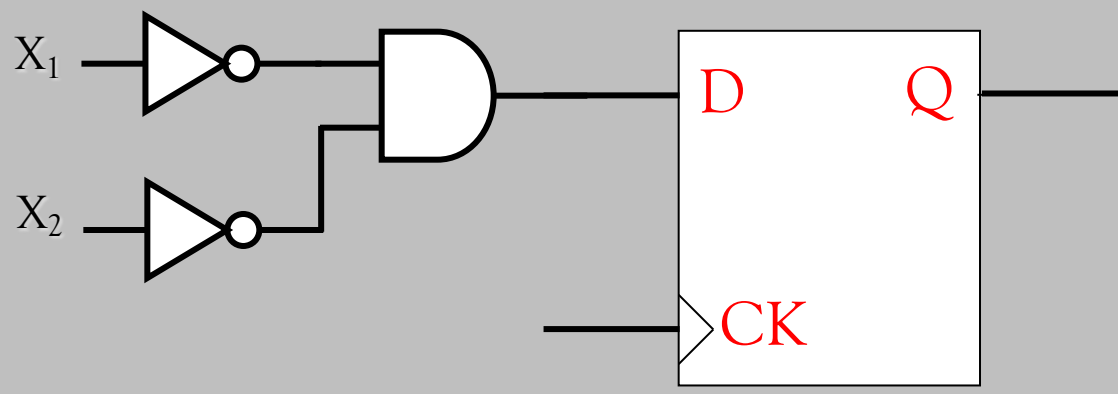
Synthesis of a circuit for the transition function:

Transition function:

Q	X1	X2	δ
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Minterms:
Q‘·X1‘·X2‘ and Q·X1‘·X2‘

Prime implicant(s):
X1‘·X2‘

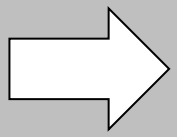


Example: Vending machine
2. Synthesis of circuits

Truth table of output function follows from encoding:

Output function:

Q	I	λ
empty	c	-
empty	r	-
empty	o1	-
empty	o2	-
full	c	-
full	r	m
full	o1	d1
full	o2	d2



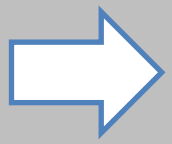
Q	X1	X2	Y1	Y2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Example: Vending machine
2. Synthesis of circuits

Synthesis of a circuit for the output function:

Output function:

Q	X1	X2	Y1	Y2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1



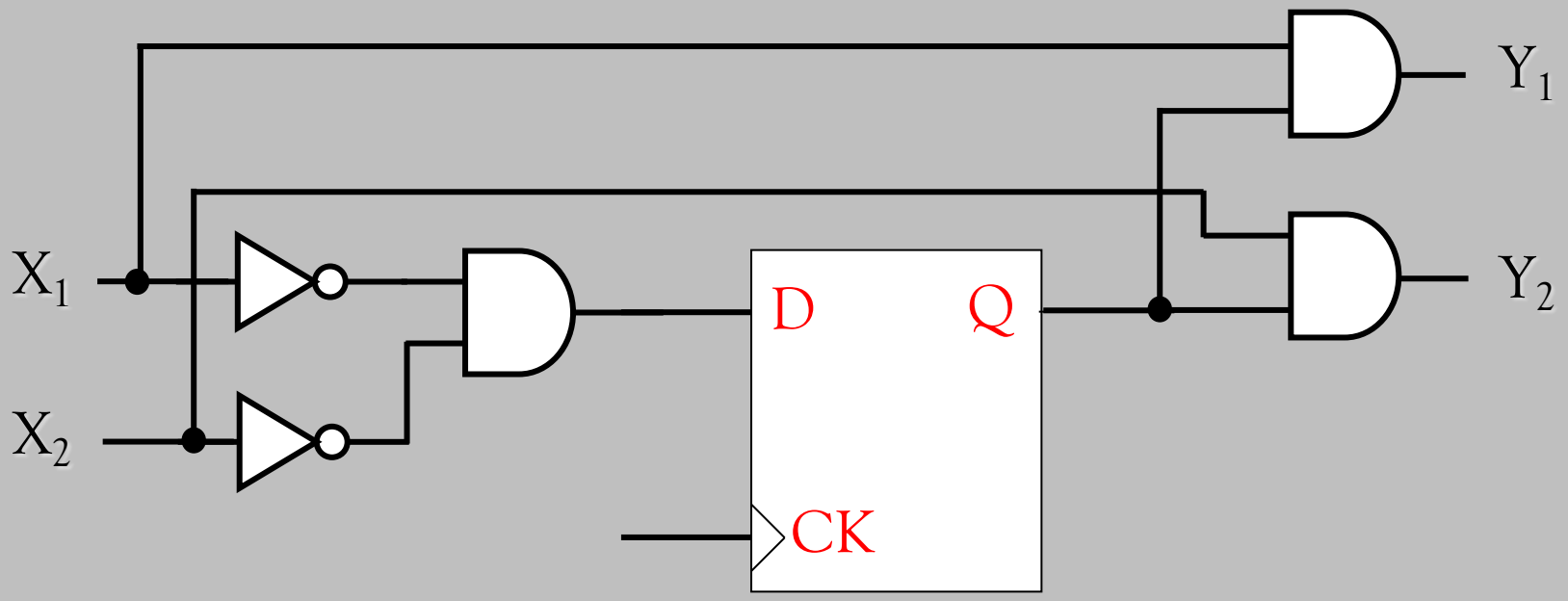
Minterms for Y1:
Q·X1·X2‘ and Q·X1·X2

Prime implicant(s):
Q·X1

Minterms for Y2:
Q·X1‘·X2 and Q·X1·X2

Prime implicant(s):
Q·X2

Example: Vending machine Sequential circuit



From Mealy machines to Sequential circuits

Encoding may strongly influence cost and depth of resulting circuits!

For lack of time we do not further consider this topic in this course.

Alternative: Moore machine

Definition:

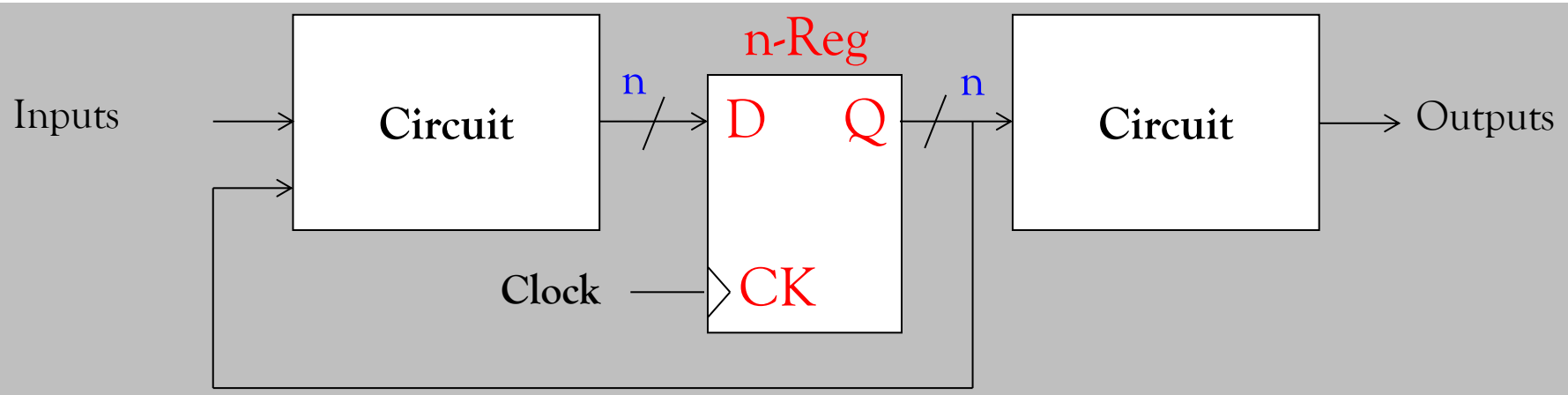
A Moore machine is a tuple $M=(Q, q_0, I, O, \delta, \lambda)$:

- Q is a *finite*, *non-empty* set of states,
- $q_0 \in Q$ is the initial state,
- I is a *finite*, *non-empty* input alphabet,
- O is a *finite*, *non-empty* output alphabet,
- $\delta : Q \times I \rightarrow Q$ is the transition function,
- $\lambda : Q \rightarrow O$ is the output function.

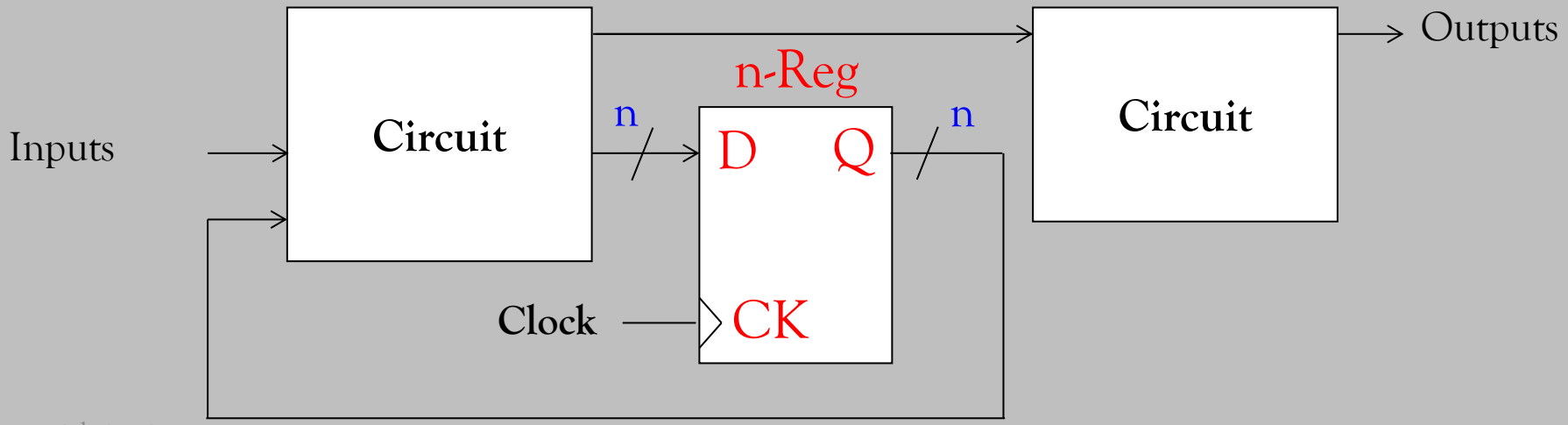
Named after:

Moore, Edward F. (1956), "*Gedanken-experiments on Sequential Machines*", Automata Studies

From Moore machines to Sequential circuits

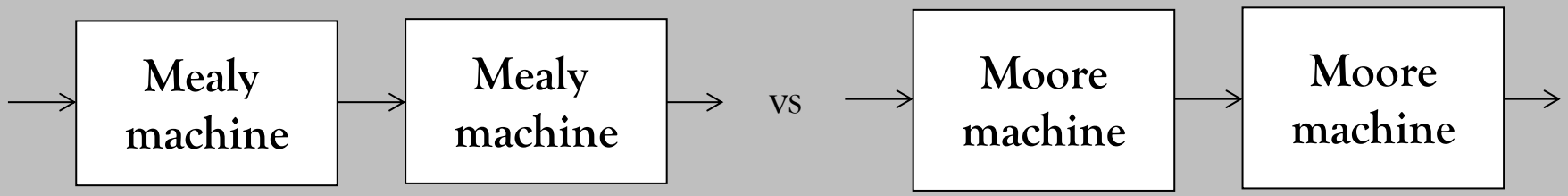


In contrast to Mealy automata:



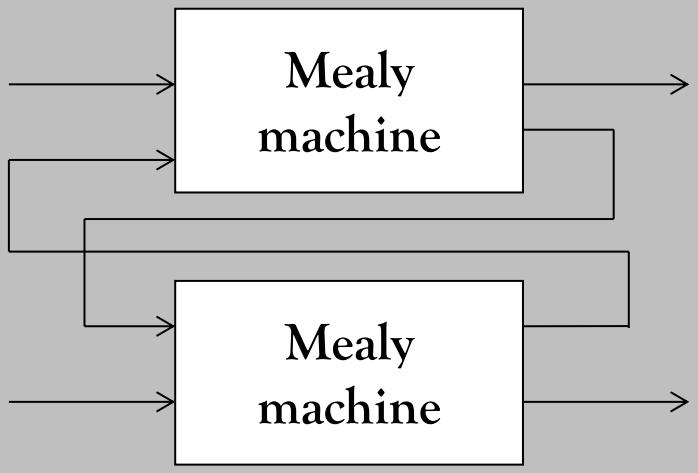
Moore- vs Mealy machines

- Mealy machines *react faster* on inputs:
  reaction in the same cycle
- Mealy machines often require *fewer states*:
  Need not store current input
- Moore machines can be more *"safely"* composed:
  "Serial composition":



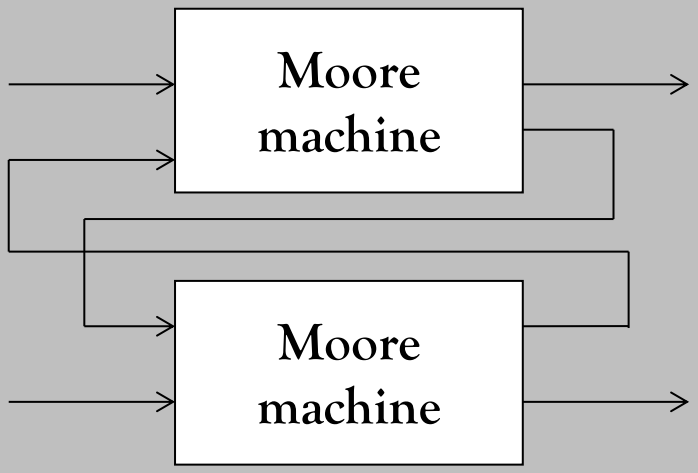
Moore- vs Mealy machines

- Mealy machines *react faster* on inputs:
  reaction in the same cycle
- Mealy automata often require *fewer states*:
  Need not store current input
- Moore automata can be more *"safely"* composed:
  "Feedback composition":



Problem: may introduce cycles!

vs



No problem!

Summary

- **Cyclic circuits** are necessary to implement **storage elements**:
  - **Latches** are level-triggered
  - **Flip-flops** are edge-triggered
- **Memory technologies**:
  - **SRAM**: fast, but low density
  - **DRAM**: slower, but higher density
- **Mathematical models for sequential circuits**:
  - **Mealy machine**:
    Output depends on current state and current input
  - **Moore machine**:
    Output depends only on current state