



Architettura degli Elaboratori I

Corso di Laurea Triennale in Informatica

Università degli Studi di Milano

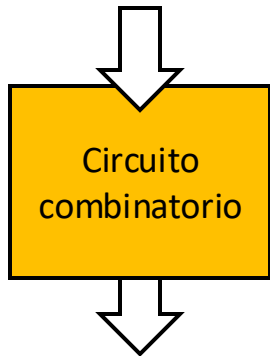
Dipartimento di Informatica "Giovanni Degli Antoni"

Edizione 2 (Cognomi H-Z), A.A. 2024-2025, Nicola.Basilico@unimi.it

Circuiti sequenziali

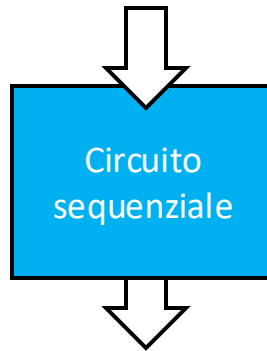
Dai circuiti combinatori a quelli sequenziali

Sequenza di input
 $\dots, x^{(t-2)}, x^{(t-1)}, x^{(t)}, \dots$



Output al tempo t
(in realtà $t + C \times \Delta_p$)
Dipende solo da $x^{(t)}$

Sequenza di input
 $\dots, x^{(t-2)}, x^{(t-1)}, x^{(t)}, \dots$



Output al tempo t
Può dipendere da tutta
o parte della **sequenza**,
non solo da $x^{(t)}$

- Per poter far dipendere l'uscita dalla sequenza, e quindi anche dagli input letti nel passato, il circuito sequenziale deve avere una capacità fondamentale: **ricordare**
- I circuiti sequenziali hanno una memoria interna che si indica con il termine di **stato**
- Lo stato codifica ciò che il circuito ricorda del passato, cioè della sequenza di input che ha visto
- L'uscita quindi dipenderà dall'input $x^{(t)}$ (il presente) e dallo stato (il passato)

- Un circuito combinatorio può essere anche visto come un caso particolare di circuito sequenziale dove la sequenza di input da cui far dipendere lo stato è di lunghezza unitaria
- Spesso l'uscita è pari al valore dello stato (lo stato viene messo in uscita per poter essere letto)

Dai circuiti combinatori a quelli sequenziali

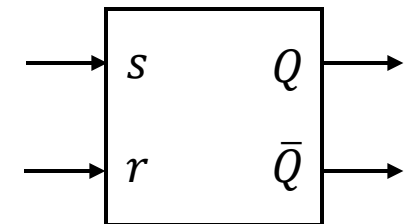
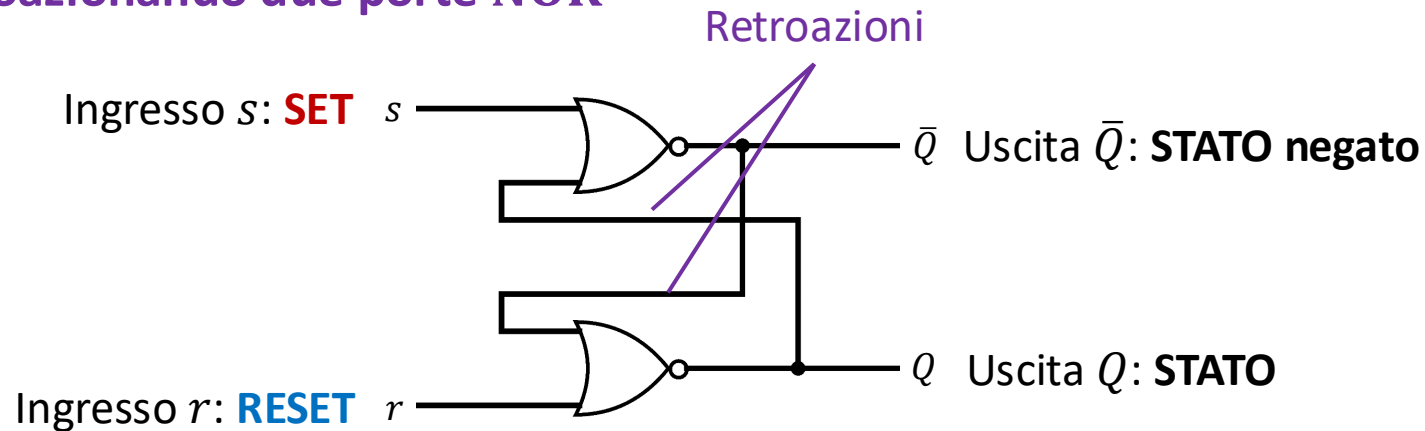
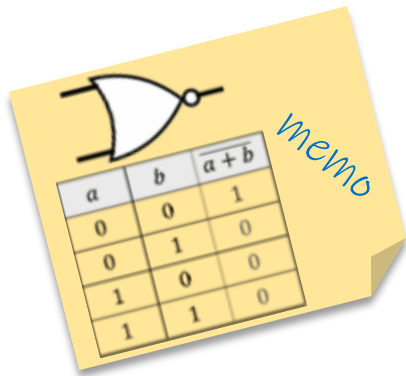
- Sulla carta possiamo scrivere che in un circuito combinatorio $y \leftarrow f(x)$, dato un input x posso determinare l'uscita y e a parità di input l'uscita è sempre la stessa
- In un circuito sequenziale invece $\langle y, s_{next} \rangle \leftarrow f(x, s)$, cioè dati input e stato corrente:
 - L'uscita y dipende sia dall'input x che dallo stato corrente s : a fronte di uno stesso input posso ottenere uscite diverse se lo stato corrente del circuito è diverso
 - Oltre a calcolare un'uscita, il circuito effettua una transizione di stato passando dallo stato corrente s allo stato successivo s_{next} (in certi casi può succedere che $s_{next} = s$)
- Per poter ricordare qualcosa, un circuito deve avere ≥ 2 stati possibili
- Se un circuito avesse 1 solo stato possibile, quello stato sarebbe implicito e potrebbe essere cancellato dalla notazione: si ottiene un circuito combinatorio!

Dai circuiti combinatori a quelli sequenziali

- Da un punto di vista progettuale che cosa cambia in un circuito sequenziale rispetto ad uno combinatorio? Cosa ha di diverso un circuito sequenziale?
- Quando abbiamo descritto i circuiti combinatori abbiamo detto che essi hanno due proprietà fondamentali (legate tra loro)
 - L'uscita dipende solo dagli input
 - **L'elaborazione procede in un senso solo: da sinistra a destra (dagli input verso gli output)**
- Dal primo punto si origina una differenza funzionale tra i due tipi di circuiti
- Dal secondo si ha una differenza implementativa: nei circuiti sequenziali il flusso di elaborazione può anche procedere da destra verso sinistra in un passaggio chiamato **retroazione**

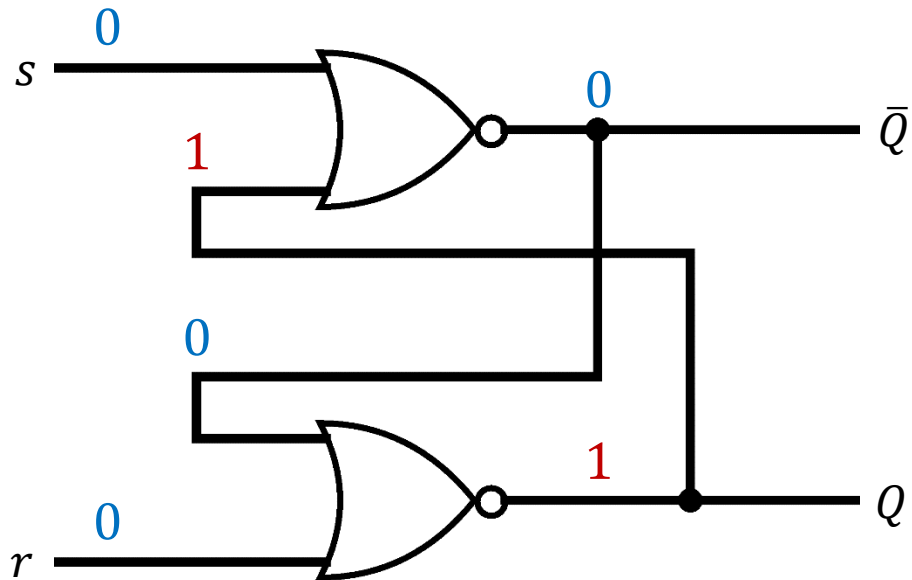
Il bistabile

- Partiamo dalle due caratteristiche che abbiamo dedotto
 1. Servono almeno 2 stati
 2. Serve la retroazione
- Possono essere combinate nel modo più semplice possibile per ottenere il circuito sequenziale che sta alla base di tutti gli altri: il **bistabile SR**
- Si ottiene **retroazionando due porte NOR**

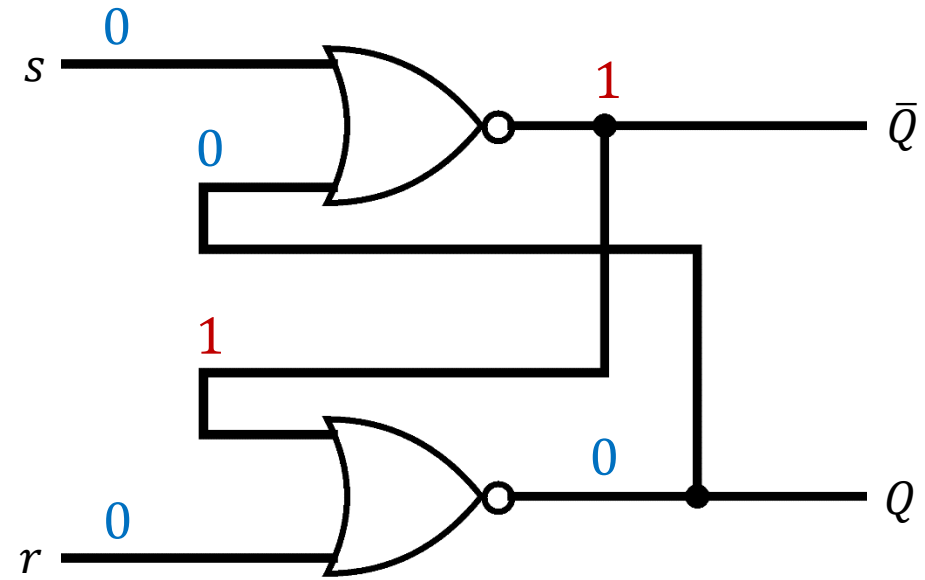


- Questo circuito ha 2 stati, è in grado di ricordare 1 bit! Vediamo perché ...

Bistabile Set-Reset (SR)



- Pongo a **1** il segnale di set s
- L'uscita Q passa a **1** (\bar{Q} passa a zero **0**)
- Riporto s a **0**: l'uscita Q resta **stabile** a **1**!
- Come se ci fosse un 1 «impresso» nella struttura intrinseca del circuito: è lo **stato**: il circuito sta **ricordando** l'evento di set avvenuto nel passato



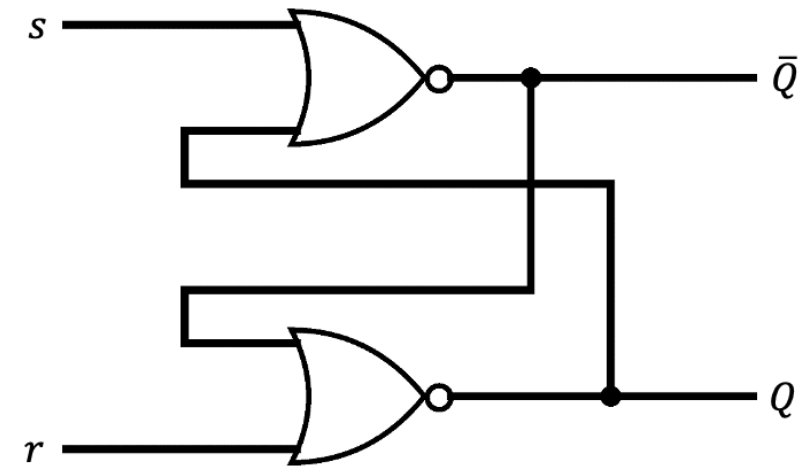
- Pongo a **1** il segnale di reset r
- L'uscita Q passa a **0** (\bar{Q} passa a **1**)
- Riporto r a **0**: l'uscita Q resta **stabile** a **0**!
- Come se ci fosse uno 0 «impresso» nella struttura intrinseca del circuito: è lo **stato**: il circuito sta **ricordando** l'evento di reset avvenuto nel passato

Bistabile Set-Reset (SR)

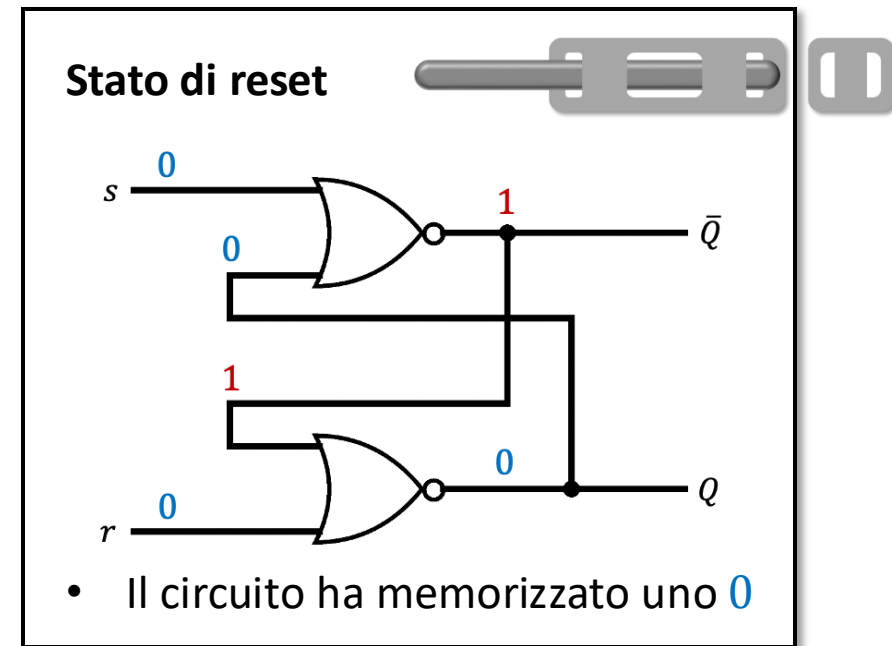
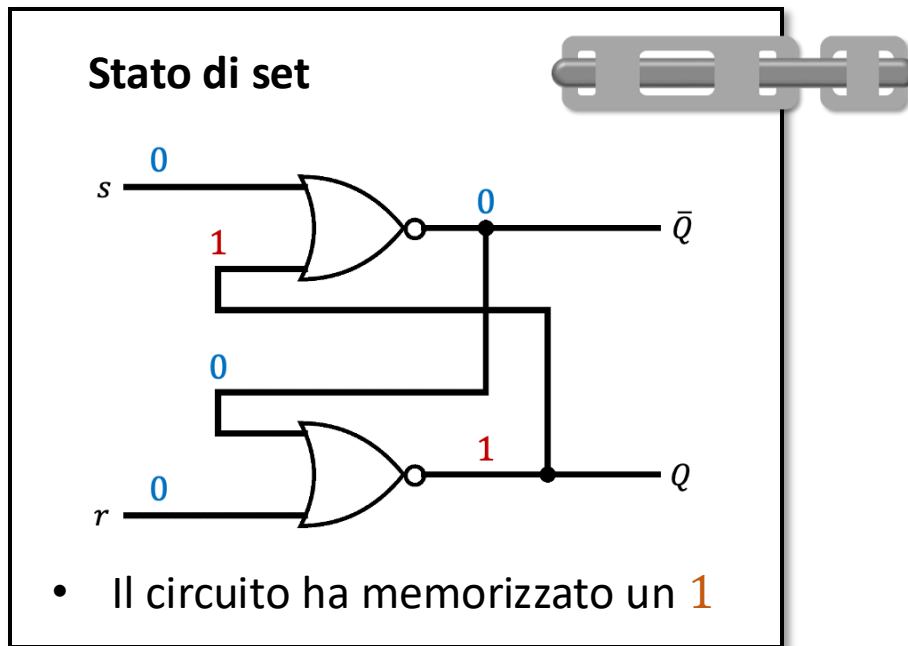
- E se poniamo a 1 entrambi i segnali s e r contemporaneamente?
 - Fintanto che $s = r = 1$, entrambe le uscite valgono a 0
 - $Q = \bar{Q} = 0$ perdiamo la semantica dei nomi delle uscite!
-
- Ripetendo lo stesso ragionamento di prima, cosa succede quando torno a $s = r = 0$?
 - I due segnali non commutano contemporaneamente, c'è sempre un piccolo Δt tra i due:
 - Se $s \rightarrow 0$ prima che $r \rightarrow 0$, ho un reset ($Q \rightarrow 0$, $\bar{Q} \rightarrow 1$)
 - Se $r \rightarrow 0$ prima che $s \rightarrow 0$, ho un set ($Q \rightarrow 1$, $\bar{Q} \rightarrow 0$)

Lo stato di arrivo è imprevedibile!

- $s = r = 1$ si può fare ma perdiamo l'interpretazione di stato e affidiamo al caso parte del comportamento del circuito. **Quindi eviteremo di farlo!**



Bistabile Set-Reset (SR)



- Il circuito si chiama **bistabile** perché riesce a mantenere stabilmente uno tra due diversi stati (set e reset), viene anche chiamato **latch** (chiavistello)
- Notiamo nelle due figure: stesso circuito, stessi input, **ma uscite diverse!** Sarebbe impossibile con un circuito combinatorio (senza retroazioni)
- Abbiamo costruito una memoria a 1 bit, il bit memorizzato è visibile in uscita (Q)

Bistabile Set-Reset (SR)

- Un circuito sequenziale ammette una tabella di verità?
- La tabella di verità non è più sufficiente, dobbiamo fornire un formalismo più generale: tabella delle transizioni

		Input sr			
		00	01	10	11
Stato Q	0	0	0	1	<i>non usato</i>
	1	1	0	1	<i>non usato</i>

- In alternativa possiamo considerare in modo esplicito il tempo e vedere lo stato prossimo come una funzione logica di input e stato corrente
- Possiamo in questo modo comporre una tabella delle transizioni fatta **come una tabella di verità**
- **Attenzione!** Q e Q_{next} sono lo stesso segnale ma in tempi diversi (corrente, successivo)

s	r	Q	Q_{next}
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	x
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	x

Bistabile Set-Reset (SR)

- Usando questa interpretazione combinatoria della transizione da stato corrente a stato prossimo, posso sintetizzare la funzione stato prossimo $Q_{next} = T(s, r, Q)$
- Dato input (s e r) e stato corrente (Q) calcola lo stato prossimo (Q_{next})
- Come se Q e Q_{next} fossero segnali diversi, anche se sappiamo che non lo sono

s	r	Q	Q_{next}	
0	0	0	0	
0	1	0	0	
1	0	0	1	$\Rightarrow s\bar{r}\bar{Q}$
1	1	0	$x \rightarrow 1$	$\Rightarrow sr\bar{Q}$
0	0	1	1	$\Rightarrow \bar{s}\bar{r}Q$
0	1	1	0	
1	0	1	1	$\Rightarrow s\bar{r}Q$
1	1	1	$x \rightarrow 1$	$\Rightarrow srQ$

$$\begin{aligned}
 T(s, r, Q) &= s\bar{r}\bar{Q} + sr\bar{Q} + \bar{s}\bar{r}Q + s\bar{r}Q + srQ \\
 &= s\bar{r}\bar{Q} + sr\bar{Q} + \bar{s}\bar{r}Q + s\bar{r}Q + srQ + srQ \\
 &= s(\bar{r}\bar{Q} + r\bar{Q} + \bar{r}Q + rQ) + \bar{r}Q(s + \bar{s}) \\
 &= s + \bar{r}Q
 \end{aligned}$$

Utilizzo della logica sequenziale

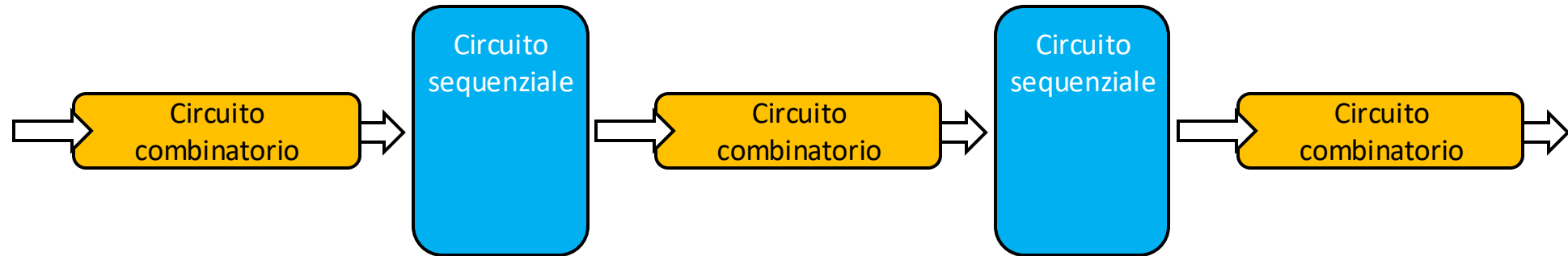
- Prima di sviluppare circuiti sequenziali più complessi a partire dal nostro latch SR, chiediamoci a cosa servono in generale questi circuiti
- **Primo utilizzo:** conservare risultati di alcune elaborazioni, ad esempio risultati intermedi prodotti dalla ALU, sono di fatto una memoria
- **Secondo utilizzo:** affrontare il problema del cammino critico in circuiti combinatori molto complessi
- **Approccio:** segmentare un circuito combinatorio in diversi sotto-circuiti e attuare una sincronizzazione tra i vari sotto-circuiti

Architetture sincrone

- Un circuito combinatorio complesso presenta un cammino critico elevato: prima che le uscite siano stabili deve passare molto tempo



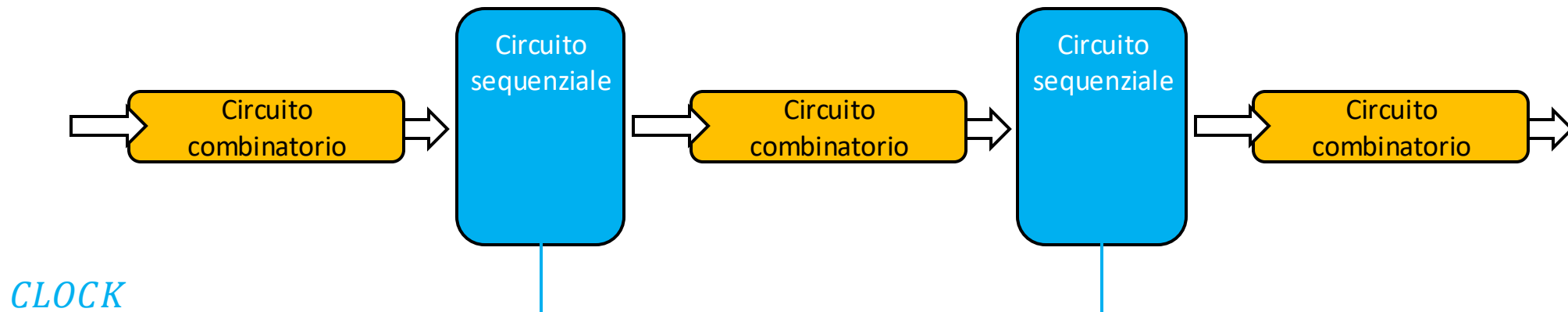
- **Idea:** segmentare il circuito combinatorio in circuiti più semplici che, collegati in serie, siano equivalenti al circuito originale
- Eseguire l'elaborazione per passi, un sotto-circuito dopo l'altro in sequenza: quando un sotto circuito ha completato (le sue uscite sono stabili), il successivo legge in input le uscite del precedente e procede
- Come si ottiene questa elaborazione «a staffetta»? **Salvando** i risultati intermedi e **sincronizzando** i passaggi



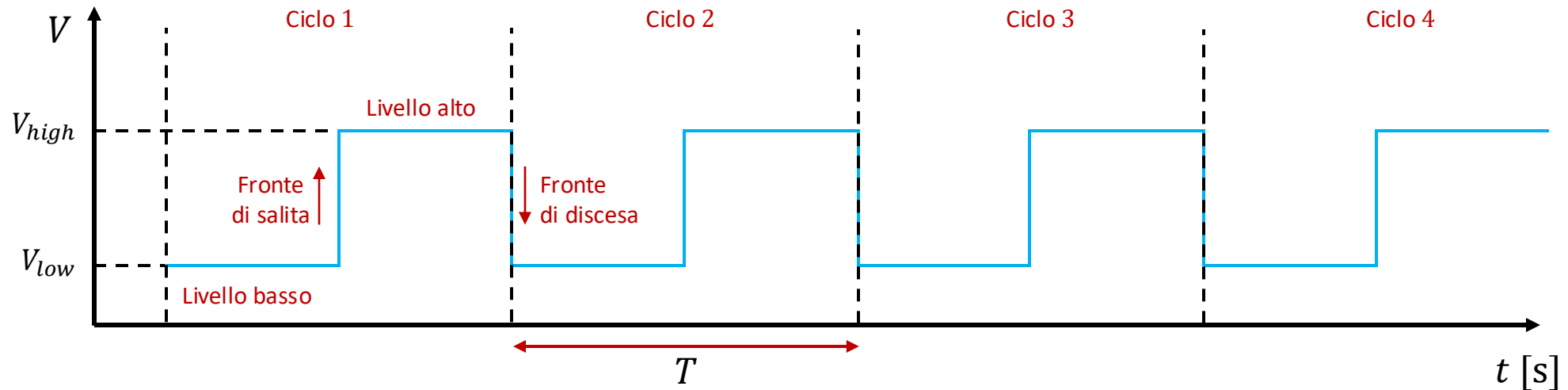
- Reparti di stoccaggio/smistamento: memorizzano il risultato proveniente da sinistra e lo rendono disponibile a destra come in una catena di montaggio

Architetture sincrone

- Per poter svolgere questa «staffetta» i vari circuiti devono coordinarsi o **sincronizzarsi** tra di loro
- **Segnale di clock**: dirige i passaggi, sincronizzando le varie fasi
- I circuiti sequenziali possono essere immaginati come dei reparti di stoccaggio e smistamento con due cancelli automatici, uno a destra e uno a sinistra
- Il clock è un segnale in grado di aprire e chiudere i cancelli
- Sincronizza l'elaborazione dicendo a ciascun reparto
 - quando aprire il cancello di sinistra e ricevere un dato dal circuito combinatorio
 - quando aprire quello di destra per passare il dato al circuito combinatorio successivo



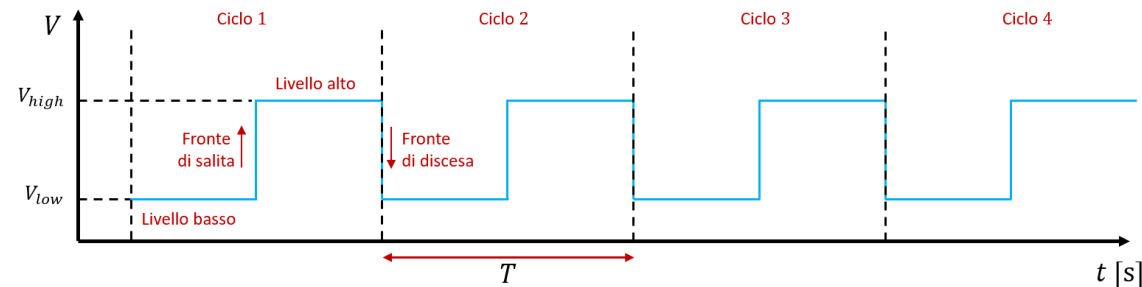
Il segnale di clock



- Il periodo T (in secondi s) misura la lunghezza temporale di un ciclo di clock, va dimensionato in modo che la logica combinatoria abbia il tempo necessario per commutare in modo stabile le uscite
- La frequenza di clock (in hertz Hz, cicli al secondo) è l'inverso del periodo $f = \frac{1}{T}$
- I livelli alto e basso del clock corrispondono a valori alti (V_{high}) e bassi (V_{low}) di tensione del segnale che rappresentano l'**1** e lo **0** logico

Architetture sincrone

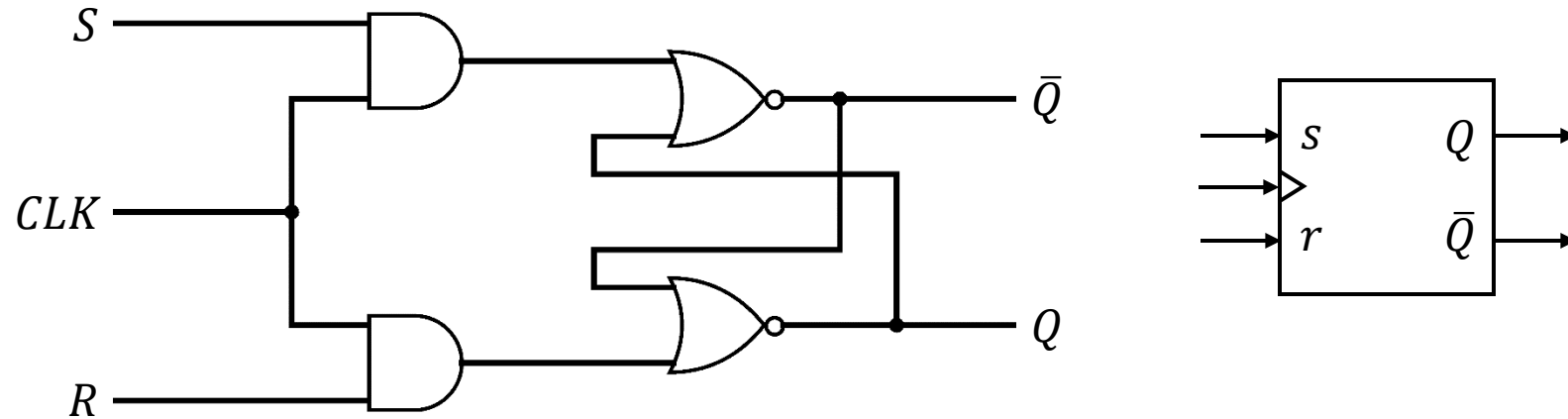
- In una architettura sincrona i circuiti «obbediscono» al segnale di clock
- Significa che le variazioni di stato possono avvenire solo quando una certa condizione sul clock è verificata



- Architetture **sensibili ai livelli**:
 - gli stati cambiano quando il livello del clock è alto (o basso),
 - quando il clock è basso (o alto) non avvengono variazioni di stato, i segnali restano stabili e si propagano nella logica combinatoria
 - Essendo i segnali stabili, la logica combinatoria può svolgere le sue elaborazioni senza problemi
- Architetture **sensibili ai fronti**:
 - Più restrittivo, le variazioni di stato possono avvenire solo sui fronti: nell'attimo in cui il clock passa da **1** a **0** o vice versa

Bistabile Set-Reset (SR) sincrono

- Progettiamo una variante del bistabile SR che «obbedisce» al clock (CLK)



- Sensibile al livello
 - Se $CLK = 0$ si forza lo stato di riposo, lo stato non può cambiare, il cancello è chiuso
 - Se $CLK = 1$ allora si può fare un set o un reset per cambiare lo stato, il cancello è aperto

Bistabile Set-Reset (SR) sincrono

- Sintesi della funzione di stato prossimo $T(s, r, Q, CLK)$

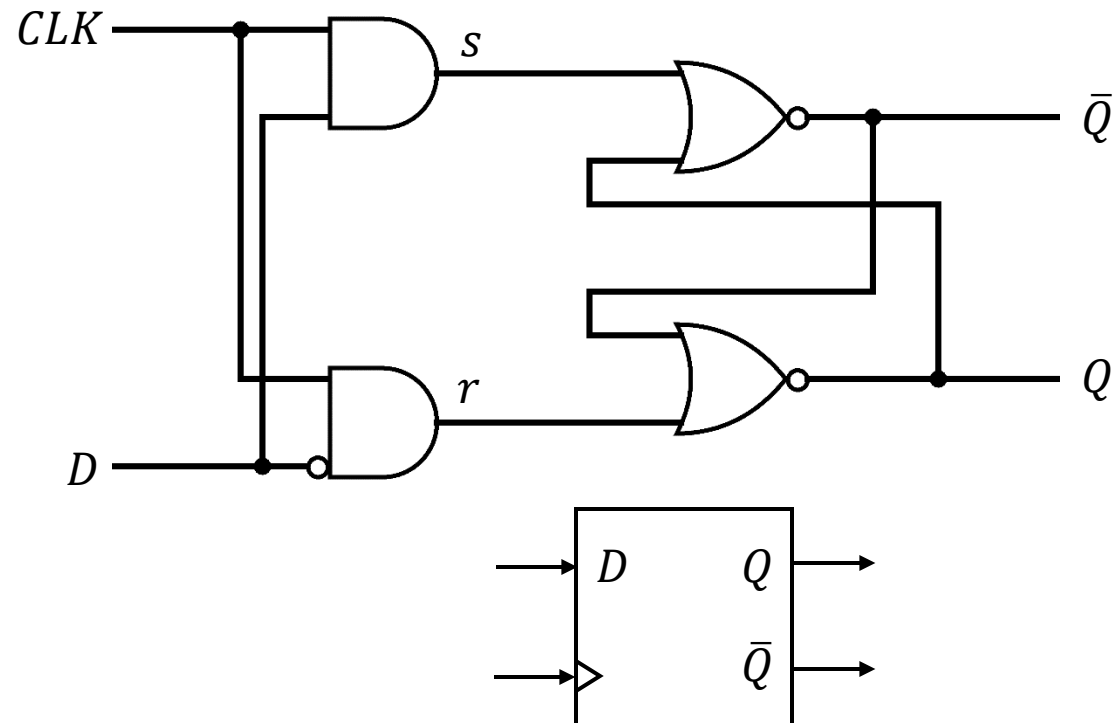
		Input sr			
		00	01	11	10
Stato Q Clock CLK	00	0	0	0	0
	01	1	1	1	1
	11	1	0	1	1
	10	0	0	1	1

$$\begin{aligned}
 T(s, r, q, CLK) &= \overline{CLK}Q + CLKs + CLKQ\bar{r} \\
 &= \overline{CLK}Q + CLK(s + \bar{r}Q)
 \end{aligned}$$

- Se il clock è basso si conserva lo status quo, Q
- Se il clock è alto abbiamo lo stesso comportamento del bistabile SR asincrono

Latch D

- Introduciamo una semplice miglione nel bistabile SR sincrono
- Nel bistabile abbiamo due input s e r , ma sappiamo che ne usiamo sempre uno per volta
- Possiamo raggrupparli in un unico input D



- $D = 1$ corrisponde a SET
- $D = 0$ corrisponde a RESET
- D sta per «dato», quando il clock è alto D viene scritto dentro il bistabile
- Anche in questo caso abbiamo sensibilità sul livello: per tutto il tempo in cui il clock è alto lo stato può cambiare

Latch D

- Sintesi della funzione di stato prossimo $T(D, Q, CLK)$

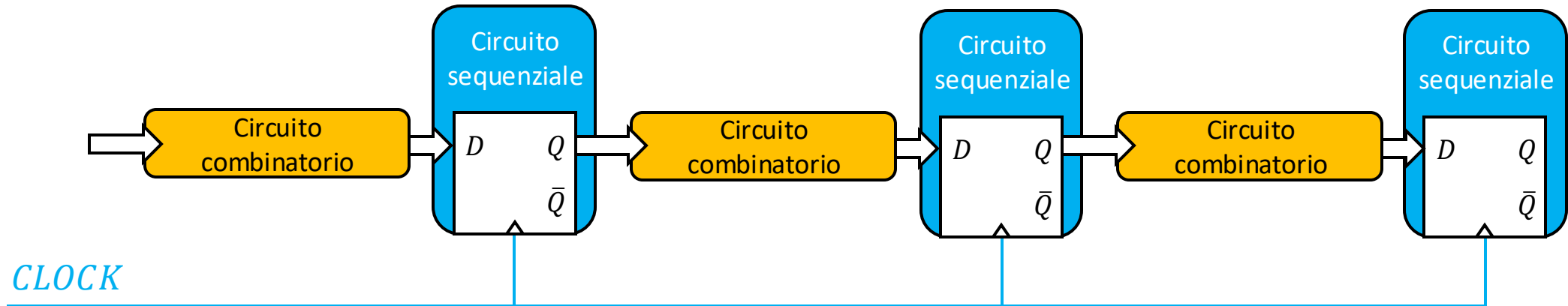
		Input D	
		0	1
Stato Q Clock CLK	00	0	0
	01	1	1
	11	0	1
	10	0	1

$$T(D, q, CLK) = \overline{CLK}Q + CLKD$$

- Se il clock è basso si conserva lo status quo, Q
- Se il clock è alto D viene scritto nel bistabile

Architettura sincrona con latch D

- Possiamo utilizzare il latch D nell'architettura sincrona che abbiamo introdotto?



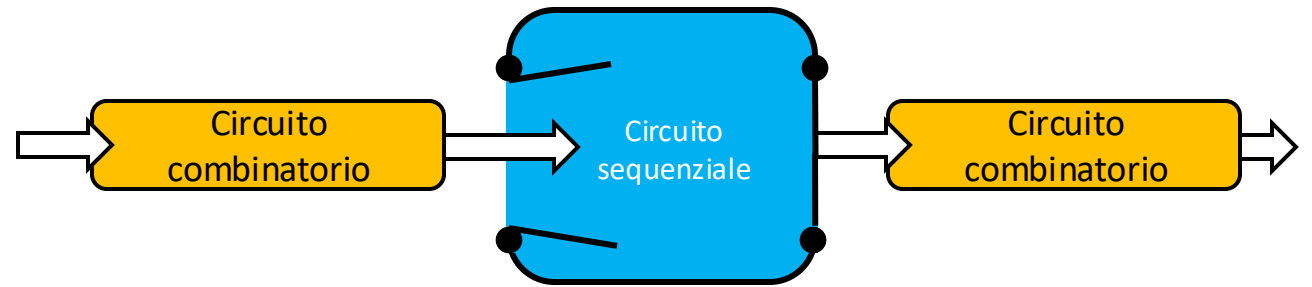
- La sensibilità sul livello crea il problema della **trasparenza**
- Quando il clock vale **0** tutti i latch mantengono lo stato corrente, niente cambia, le uscite sono stabili
- Quando il clock vale **1** tutti i latch sono sensibili a cambiamenti di stati
- Tutti i cancelli sono aperti! Perdiamo la separazione tra stadi e il segnale può attraversare tutto il circuito da sinistra a destra: riotteniamo il problema del cammino critico

Architettura sincrona con latch D

Idea: costruire dei circuiti sequenziali con cancello doppio

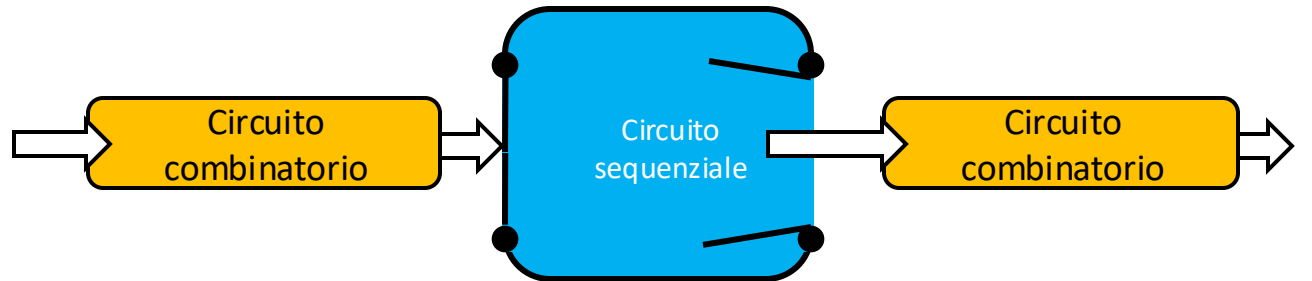
Primo step (clock alto):

- il cancello di sinistra si apre
- Il dato proveniente dal primo circuito combinatorio entra nel circuito sequenziale e viene memorizzato
- Il cancello di destra è chiuso! La separazione tra stadi tiene



Secondo step (clock basso):

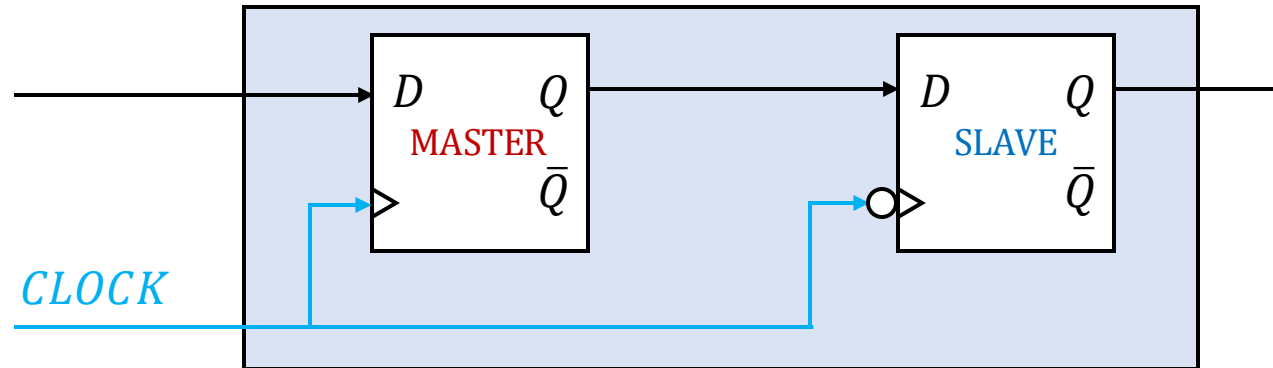
- il cancello di destra si apre
- Il dato che era stato memorizzato precedentemente viene mandato in uscita, il secondo circuito combinatorio lo riceve in input
- Il cancello di sinistra è chiuso! Il circuito precedente completa la sua elaborazione stabilizzandosi



Il circuito sequenziale che implementa questa idea si chiama **Flip Flop** e combina 2 latch D

Flip Flop

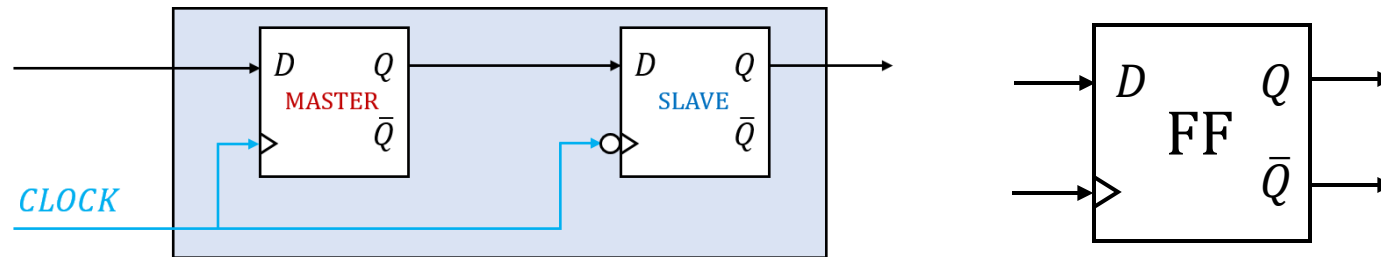
- Due latch D collegati in serie in una configurazione master-slave, dove il latch slave riceve il clock negato



- $CLOCK \rightarrow 1$: **FLIP**
 - **master aperto**, il dato in input viene scritto
 - **slave isolato**, continua a mettere in uscita il suo stato corrente (spoiler: è il dato che era contenuto nel master prima del FLIP)
- $CLOCK \rightarrow 0$: **FLOP**
 - **master isolato**, mette in uscita il suo stato corrente (il dato memorizzato durante il FLIP)
 - **slave aperto**, in un tempo quasi istantaneo lo stato del master viene copiato nello slave che inizia subito a porlo stabilmente in uscita

Flip Flop

- Il Flip-Flop è un circuito sensibile ai fronti



- L'uscita commuta in modo istantaneo sul fronte di discesa del clock e resta stabile fino al prossimo fronte di discesa quindi per tutto il ciclo di clock
- L'architettura sincrona procede per step, uno per ogni ciclo di clock

