

# Tapped Delay

---

Copyright (c) 2024 - 2019 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).  
This document was produced by using LibreOffice.

# Based on

---

Introduction to Signal Processing

S. J. Ofranidis

The necessities in DSP C Programming

FIR Filter (A.pdf) 20191114

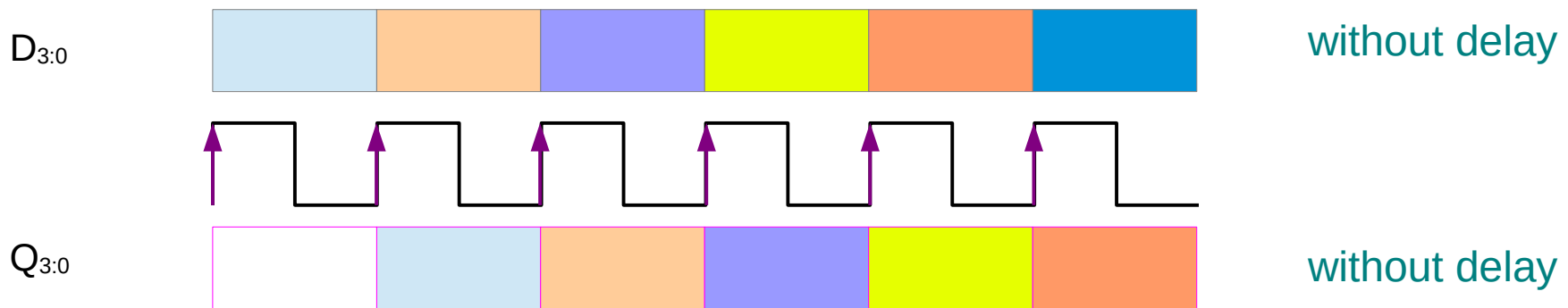
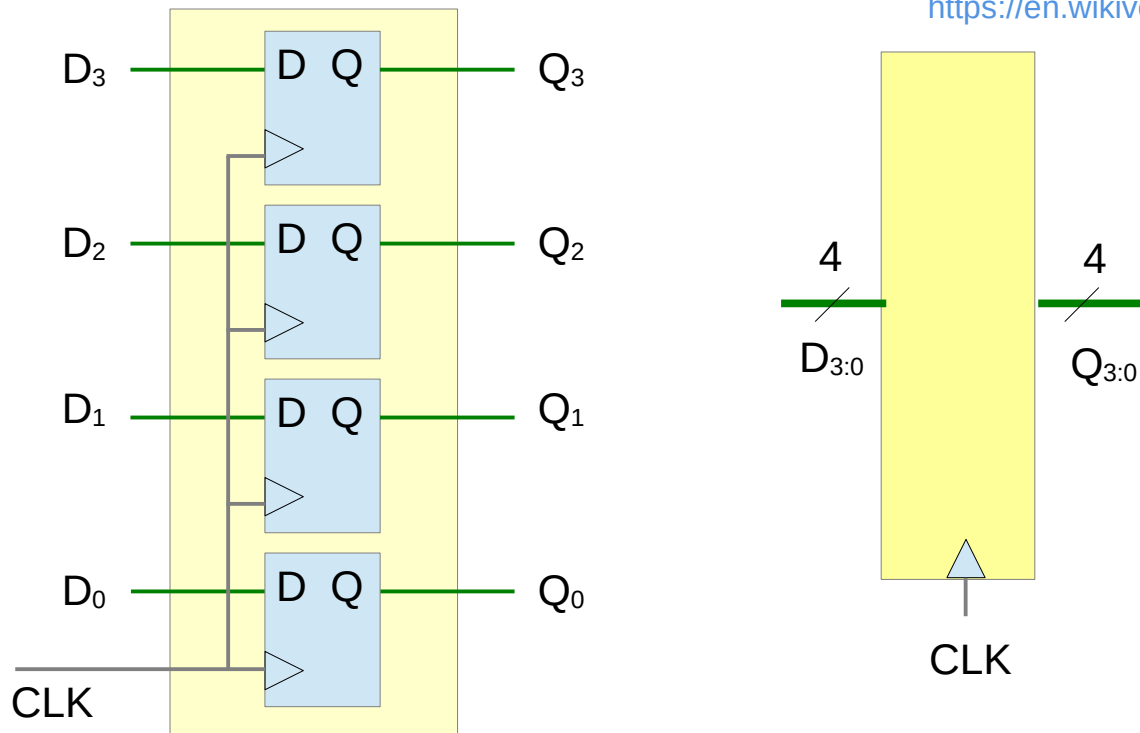
# D Flip Flop

---

Considering the widely used  
Edge triggered  
D-type Flip Flops

# Register

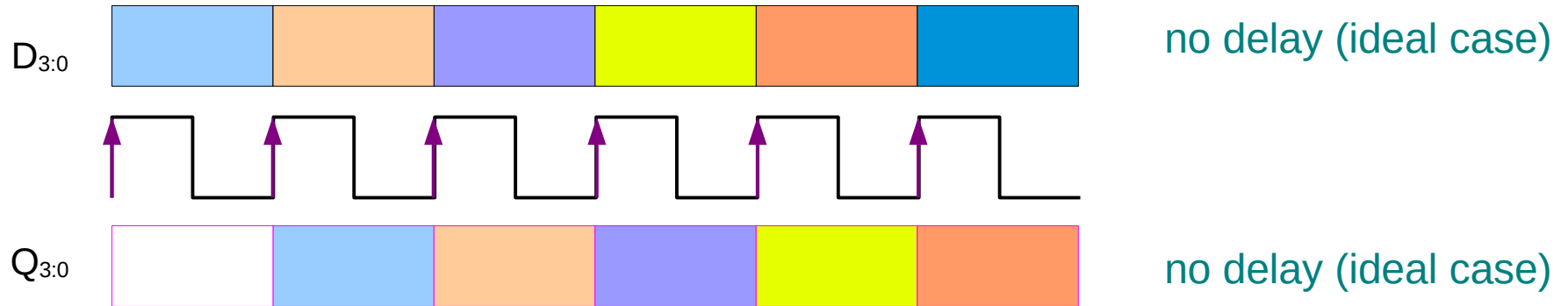
[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)



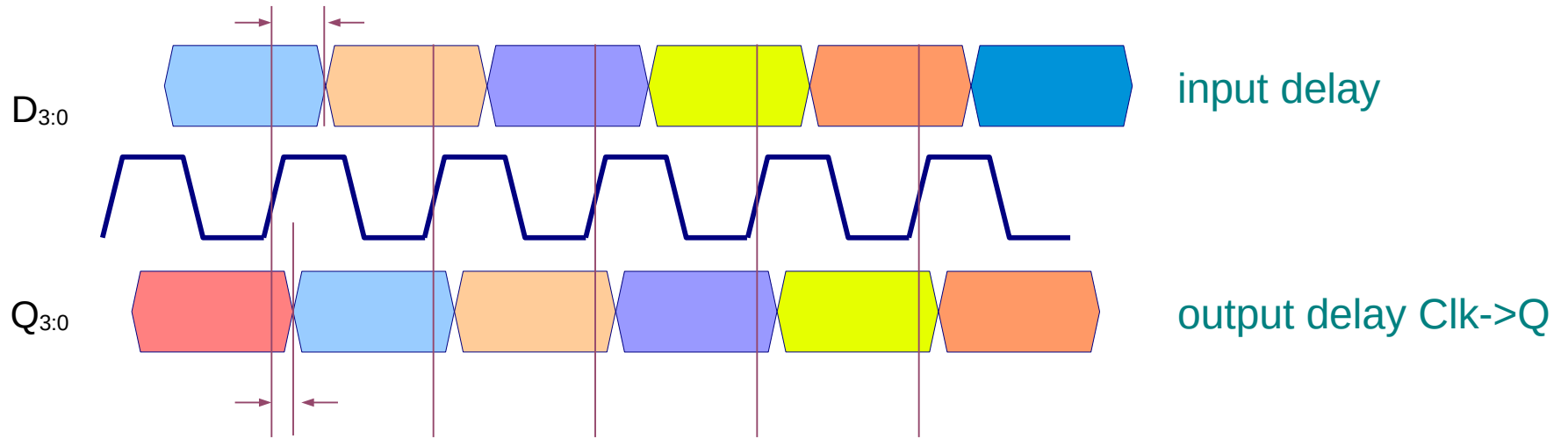
# Types of Timing Diagrams

## a timing diagram without delays

[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)

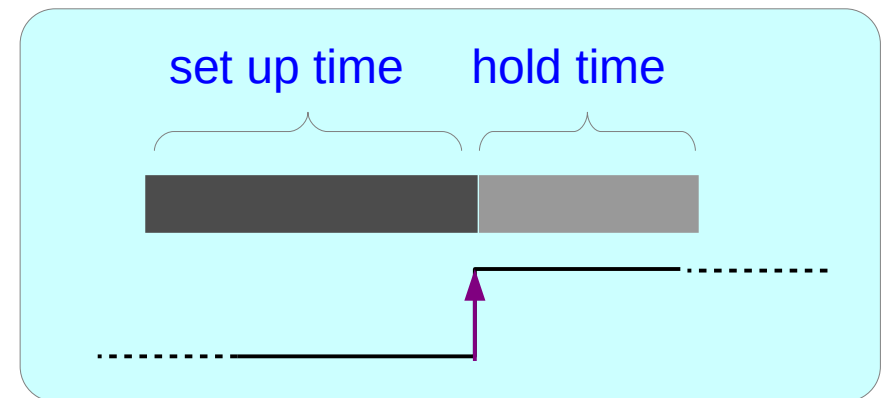
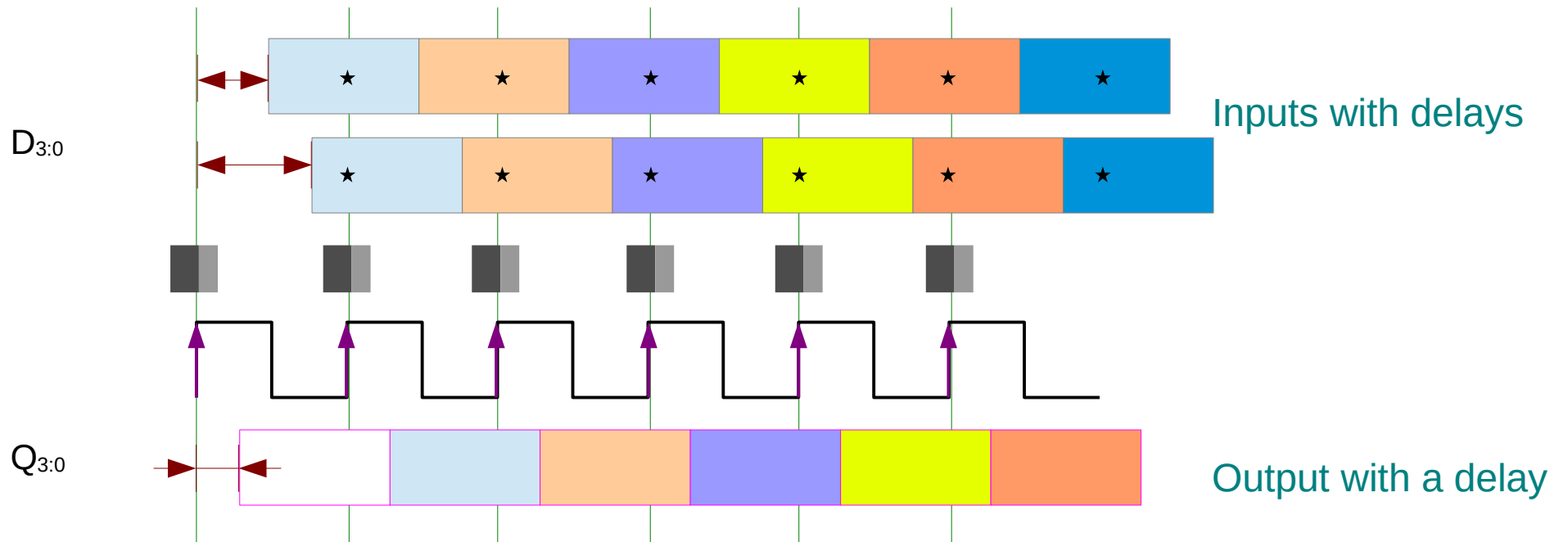


## a timing diagram with delays



# Setup & Hold Time (1)

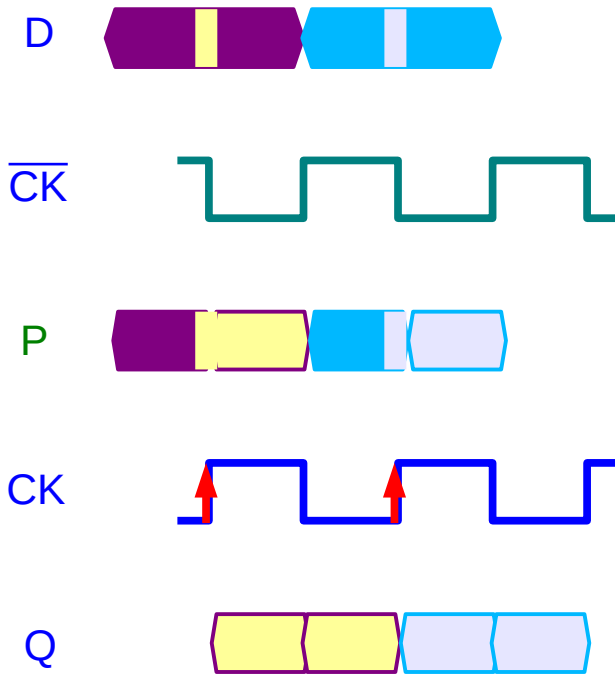
[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)



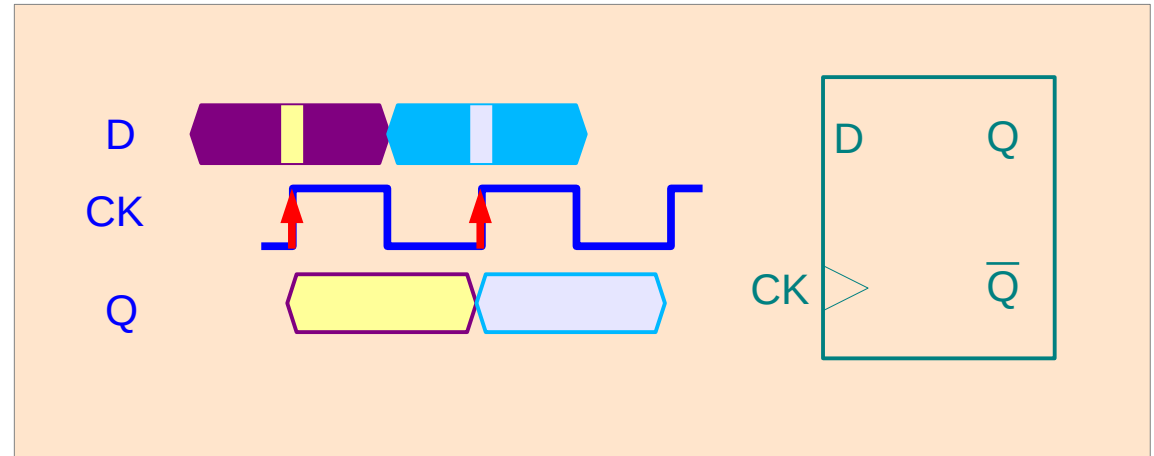
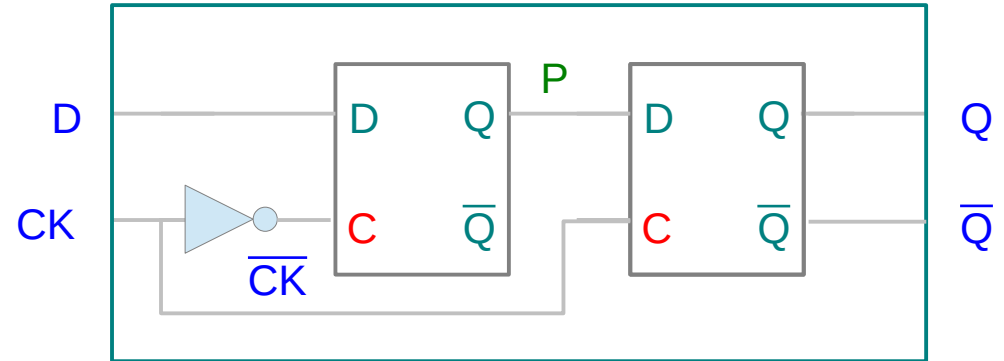
# Master-Slave D FlipFlop – Rising Edge

[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)

Master D Latch



most of D data in a time slot (dark shaded area) is not used but only the data around the rising edge of the clock CK

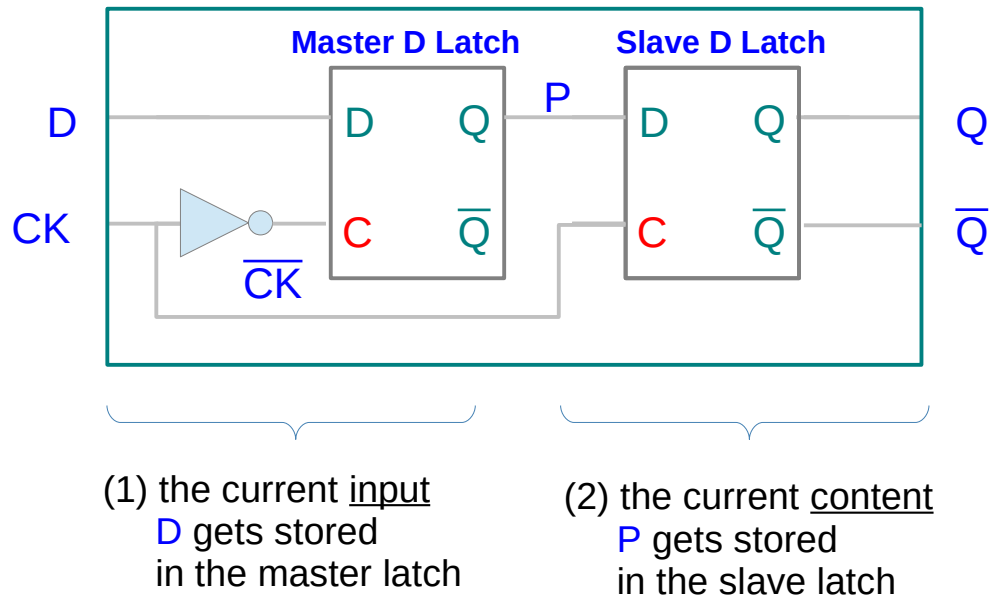


Slave D Latch

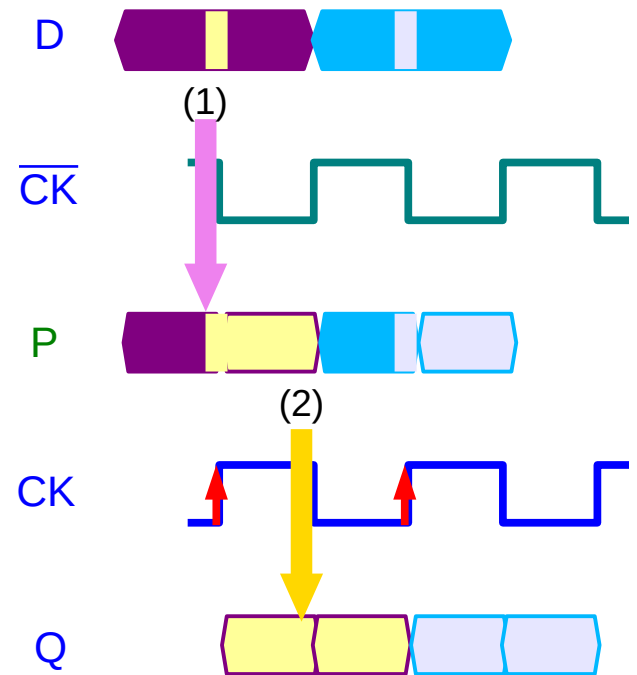


# Master-Slave D FlipFlop – Rising Edge

[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)



Master D Latch



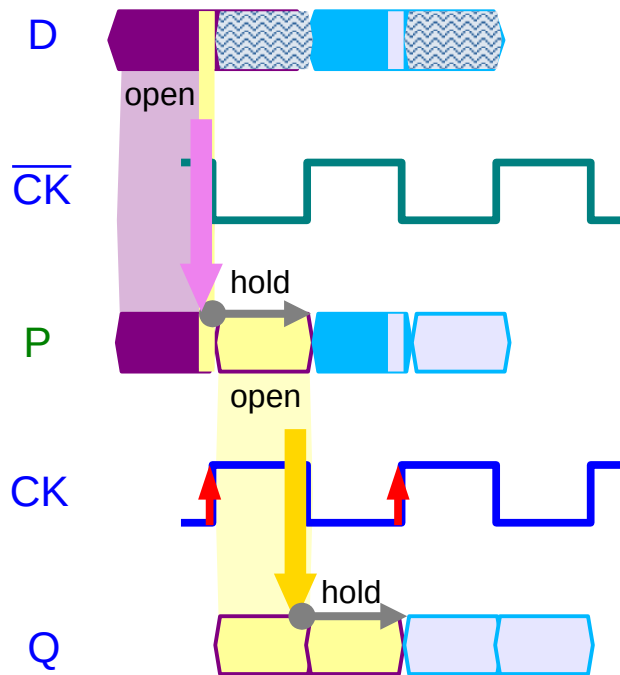
Slave D Latch

Using **inverted clocks enable**  
(1) and (2) to be executed sequentially


# Master-Slave D FlipFlop – open and hold

[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)

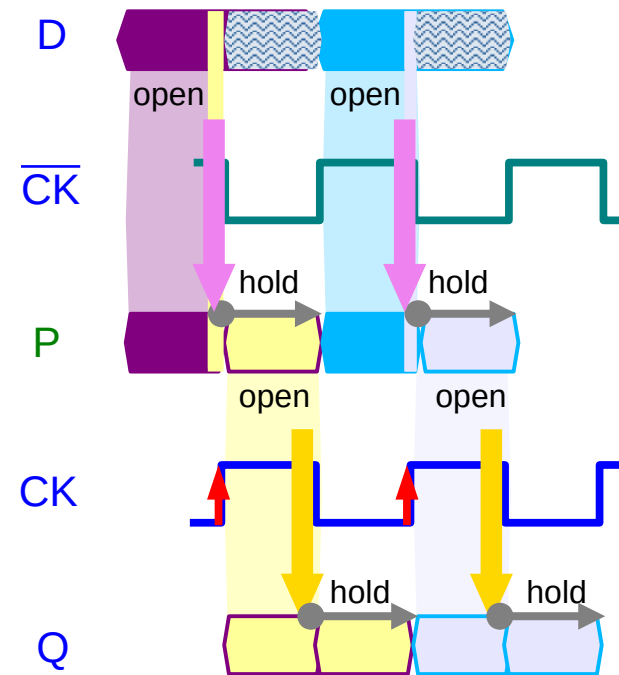
Master D Latch




Slave D Latch

 shaded area has no effect in operation

Master D Latch

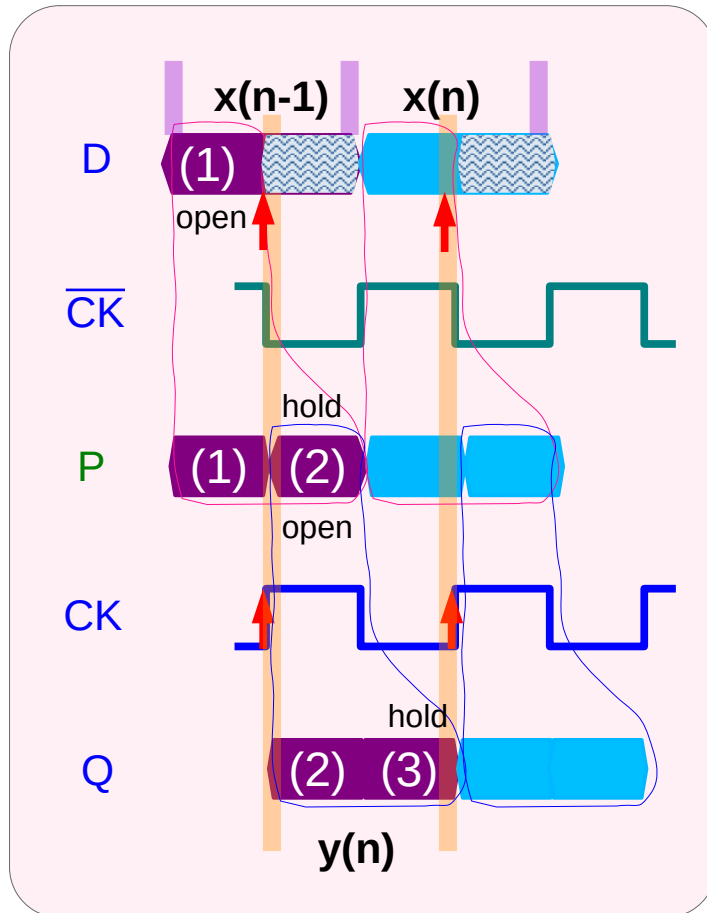


Slave D Latch

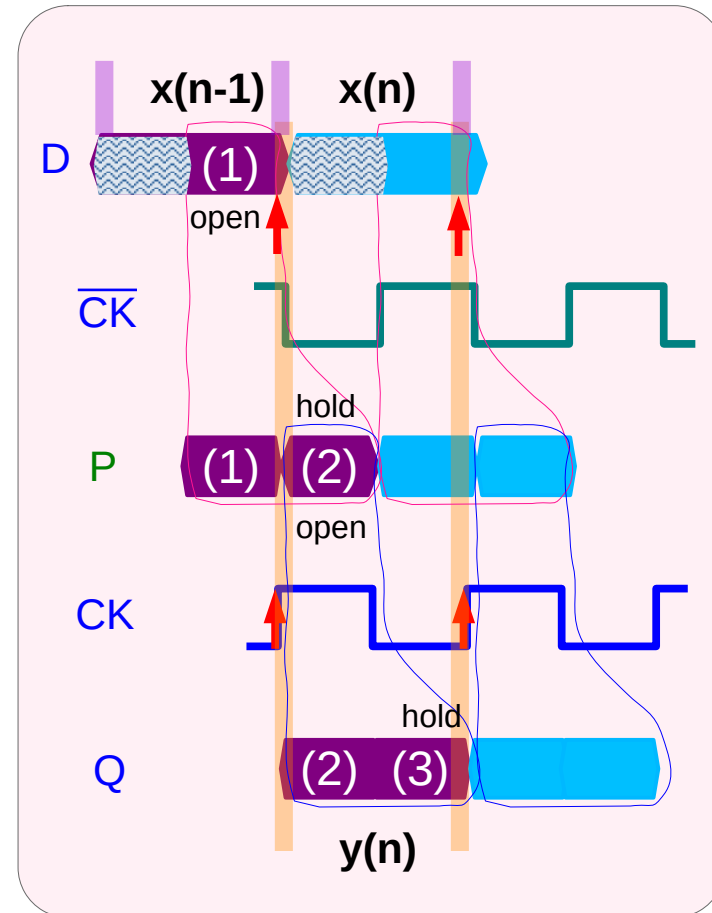
 shaded area has no effect in operation

# Master-Slave D FlipFlop – typical vs. ideal timing (1)

## Typical Timing

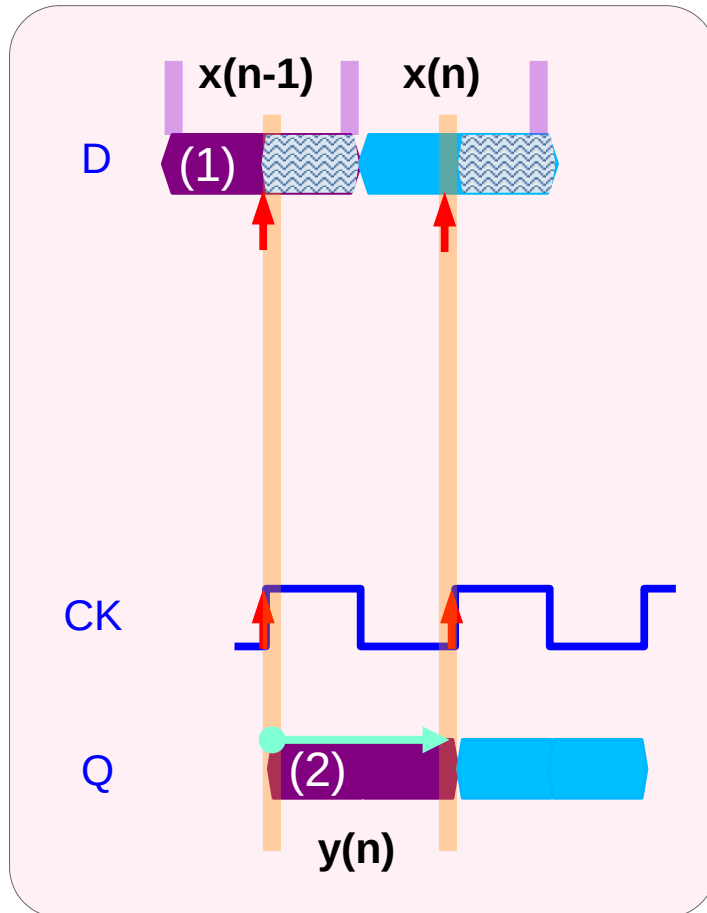


## Ideal Timing

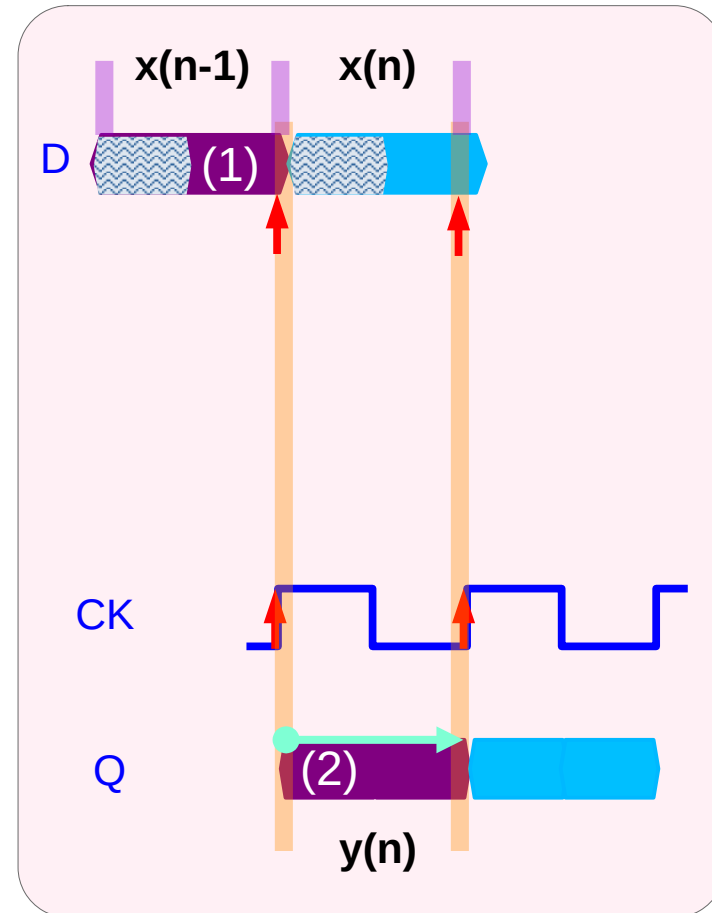


# Master-Slave D FlipFlop – typical vs. ideal timing (2)

## Typical Timing

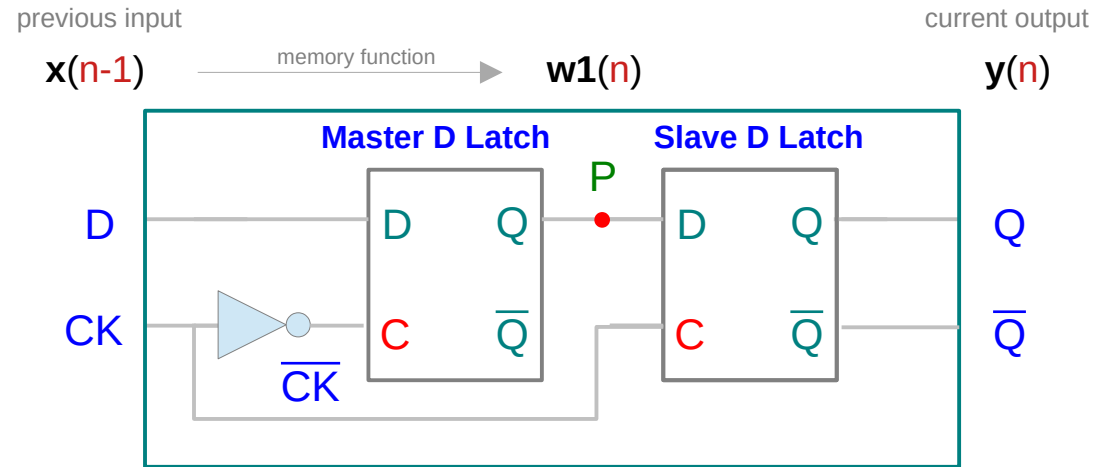
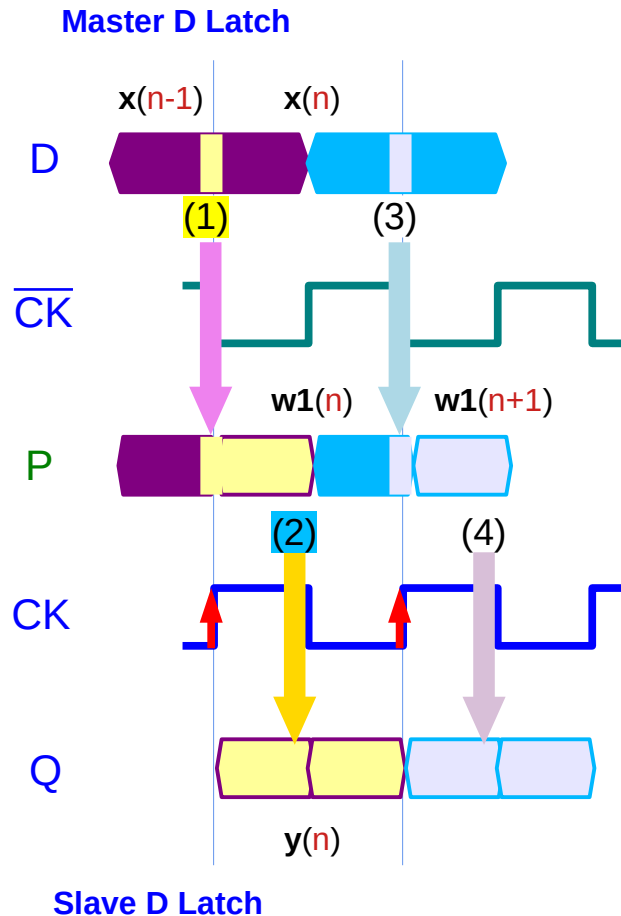


## Ideal Timing



# Master-Slave D FlipFlop – Rising Edge Sampling

[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)



**(1)** the input  $x(n-1)$  gets stored in the master latch

$$w1(n) = x(n-1)$$

**WR**  $w1(n)$

first, write

at the end of the previous time slot  $n-1$ ,  $x(n-1)$  gets stored in  $w1(n)$  for the current time slot  $n$

**(2)** the content of  $w1(n)$  gets stored in the slave latch

$$y(n) = w1(n)$$

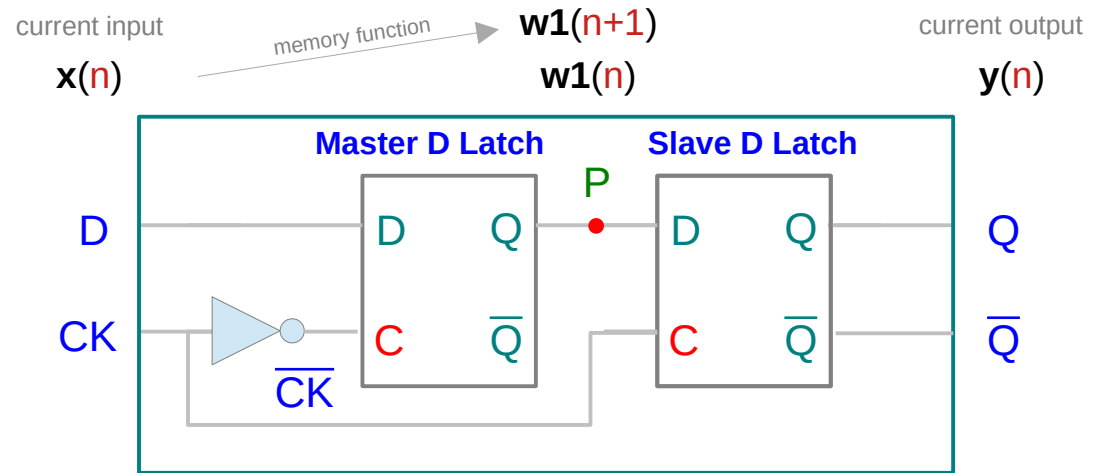
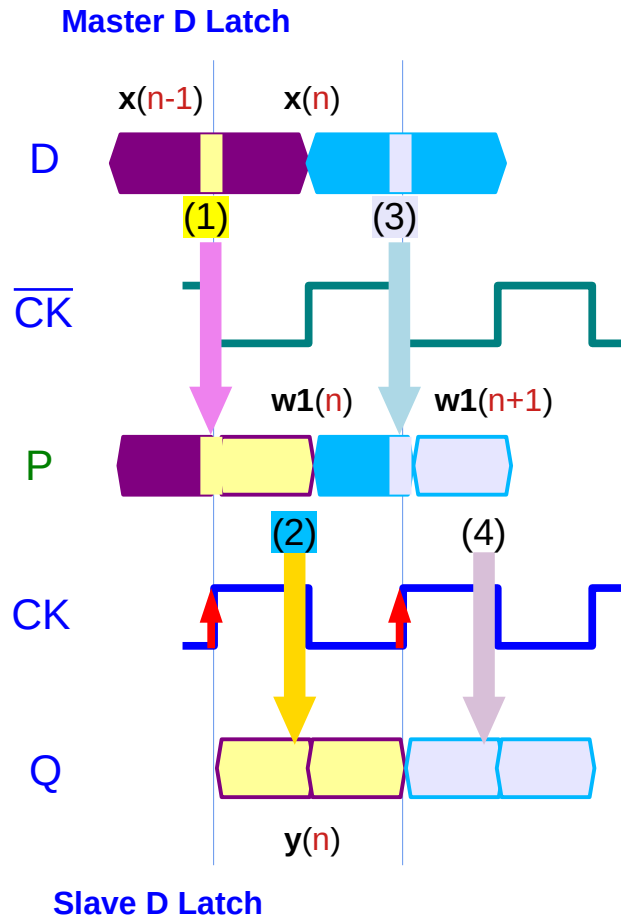
**RD**  $w1(n)$

then, read

during the current time slot  $n$ ,  $y(n)$  holds the the value of  $x(n-1)$  at the end of the previous time slot  $n-1$

# Master-Slave D FlipFlop – Rising Edge Sampling

[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)



(2) the content of  $w1(n)$  gets stored in the slave latch

$$y(n) = w1(n)$$

**RD**  $w1(n)$

first, read

during the current time slot  $n$ ,  $y(n)$  hold the the value of  $x(n-1)$  at the end of the previous time slot  $n-1$

(3) the input  $x(n)$  gets stored in the master latch

$$w1(n+1) = x(n)$$

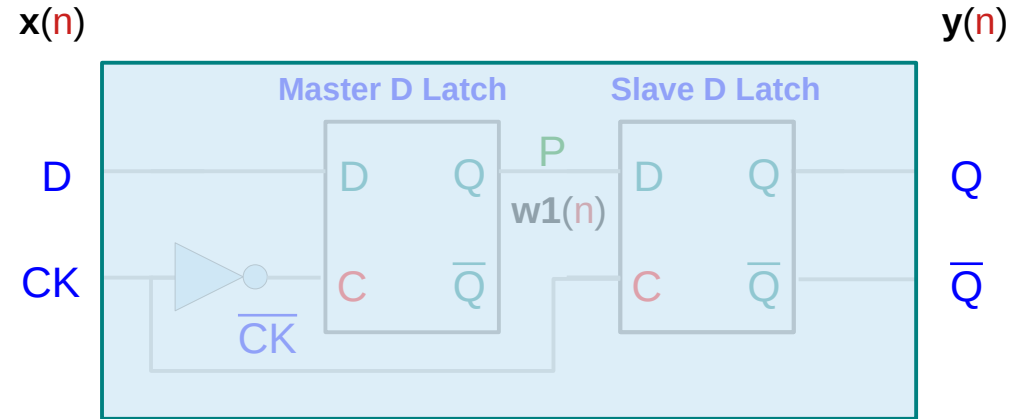
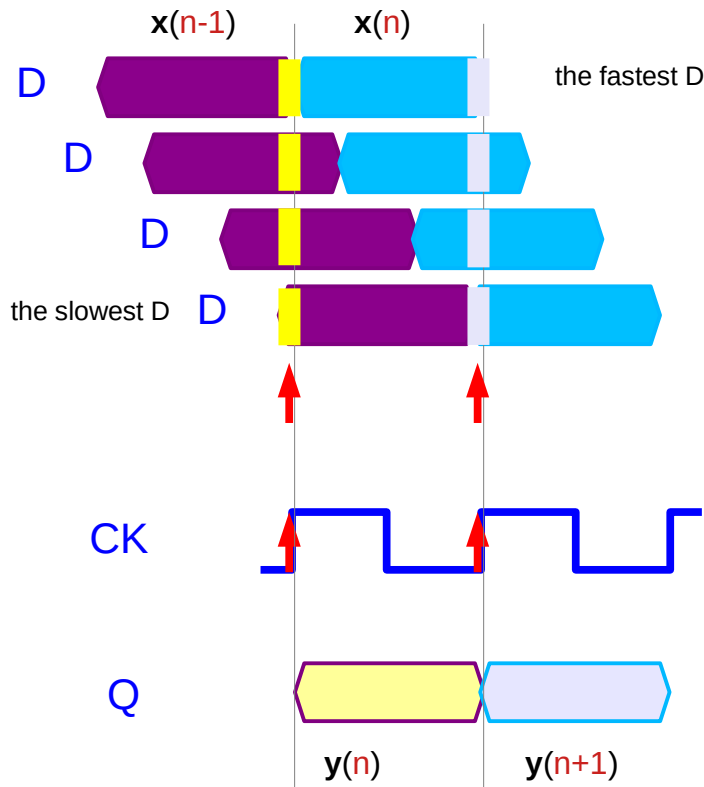
**WR**  $w1(n+1)$

then, write

at the end of the current time slot  $n$ ,  $x(n)$  gets stored in  $w1(n+1)$  for the next time slot  $n+1$

# Master-Slave D FlipFlop – Rising Edge Sampling

[https://en.wikiversity.org/wiki/The\\_necessities\\_in\\_Digital\\_Design](https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design)



$$y(n) = w1(n) \quad RD \ w1(n)$$

$$w1(n+1) = x(n) \quad WR \ w1(n+1)$$

first **RD**, then **WR**  
no **RAW** (read after write) hazards

# Fixed point representation

fractional numbers

**Floating Point Representation**

**Fixed Point Representation**

integer + implicit fixed scaling factor

a 2's complement number  $(11110101)_2 = -11$

could represent  $-11 \cdot 2^{-3} = -88$ ,  
 $-11 \cdot 2^{-5} = -0.34375$   
 $-11 \cdot 2^{-12} = -0.002685546875$

with implied scaling factors -3, -5, -12

fractional numbers

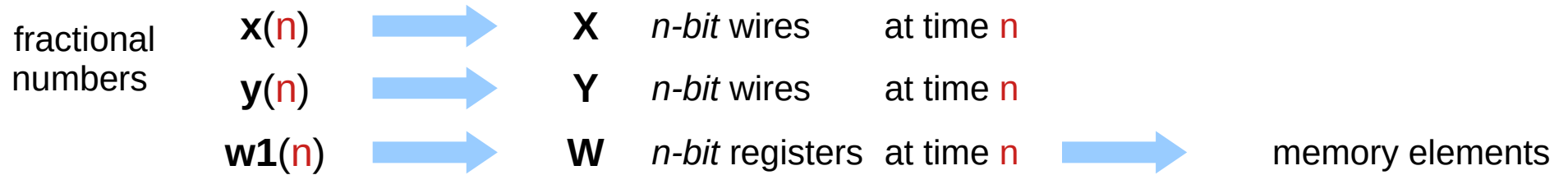
$x(n)$   **X** *n-bit wires* at time *n*

$y(n)$   **Y** *n-bit wires* at time *n*

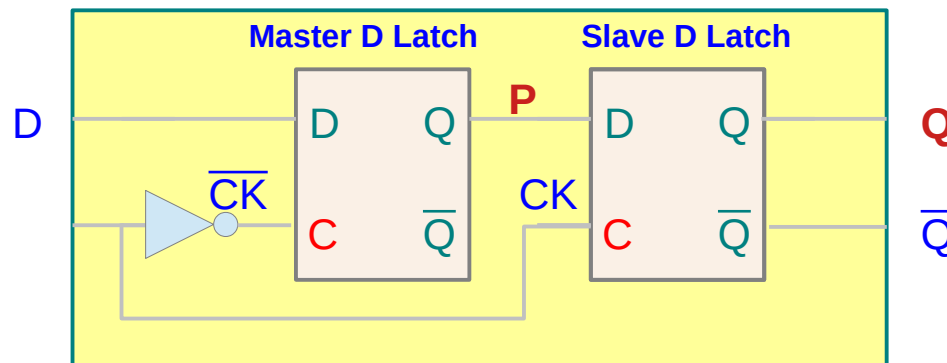
$w1(n)$   **W** *n-bit registers* at time *n*



# Memory Elements

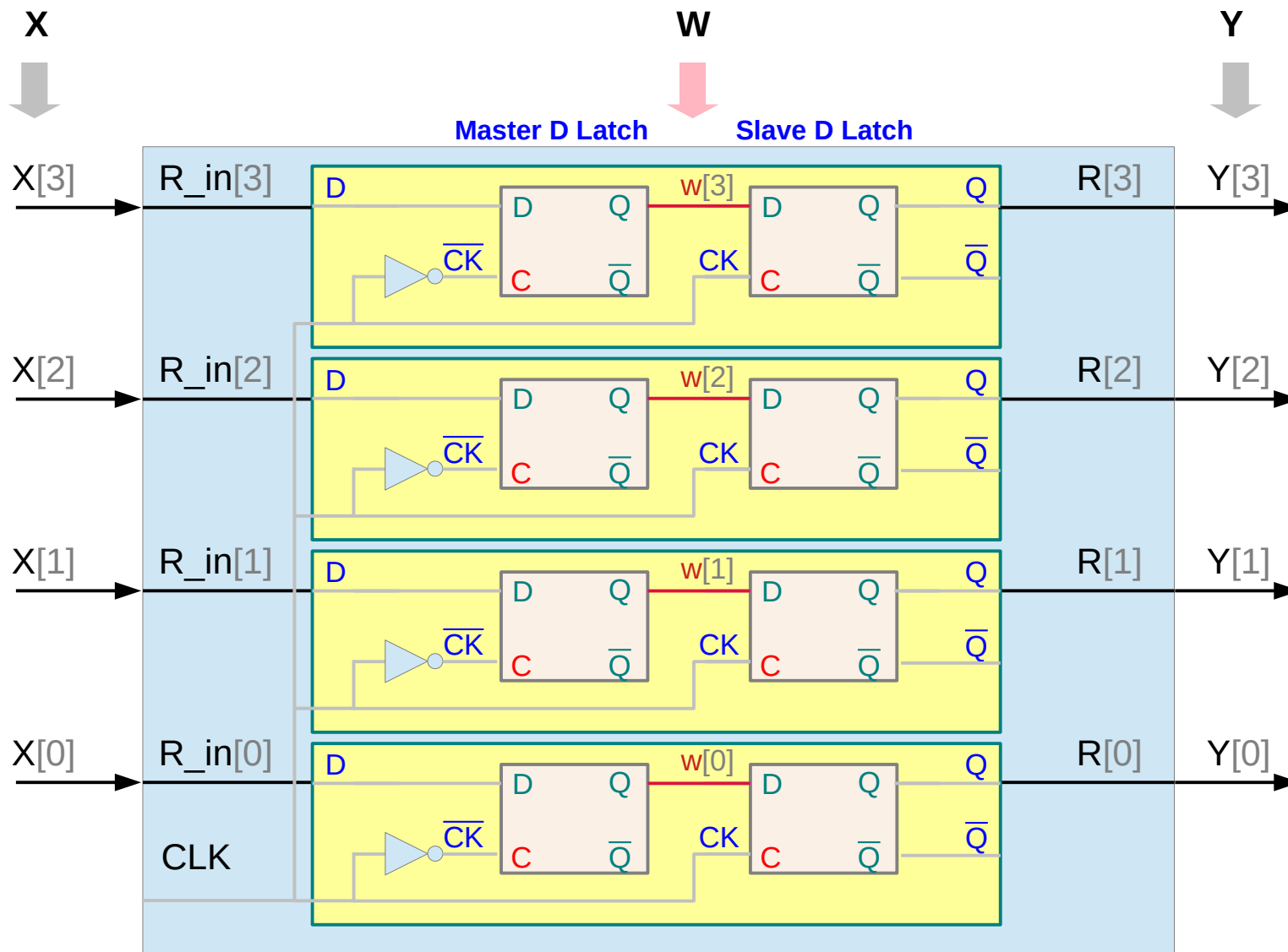


1. using the **P** outputs of **master latches** as the name for **memory elements**
2. using the **Q** outputs of **flipflops** as the name for **memory elements**

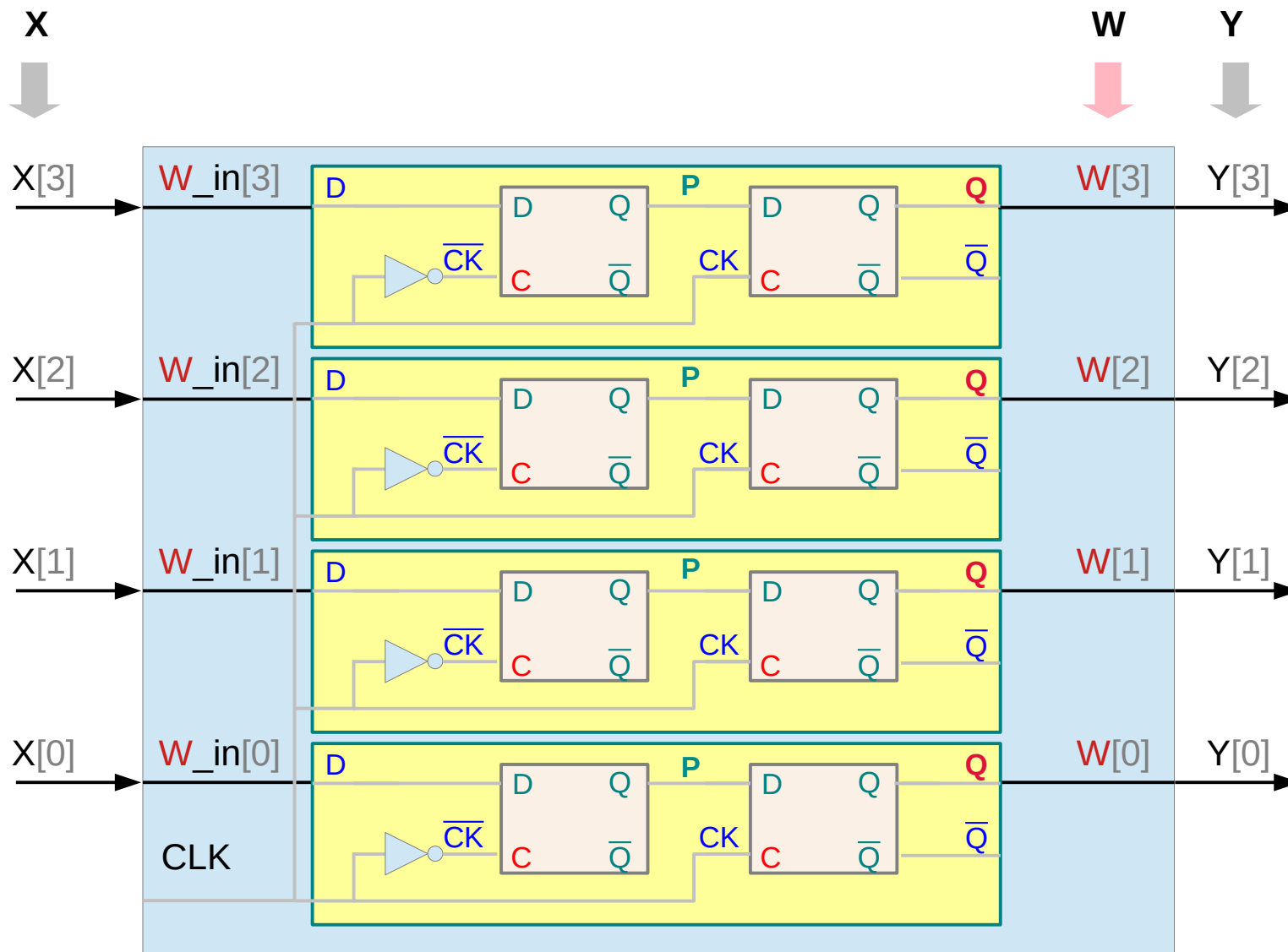


D FlipFlop using Master and Slave D Latches

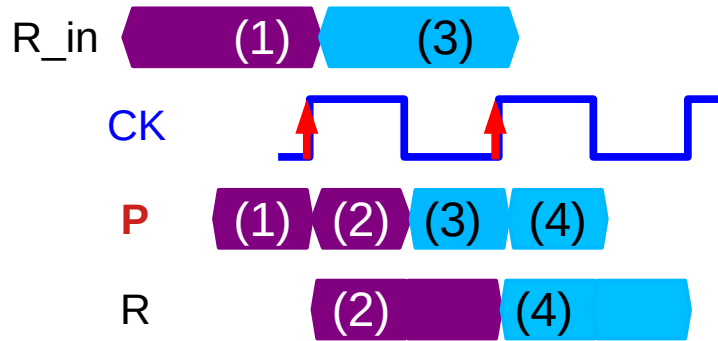
# (1) Using master latch outputs P



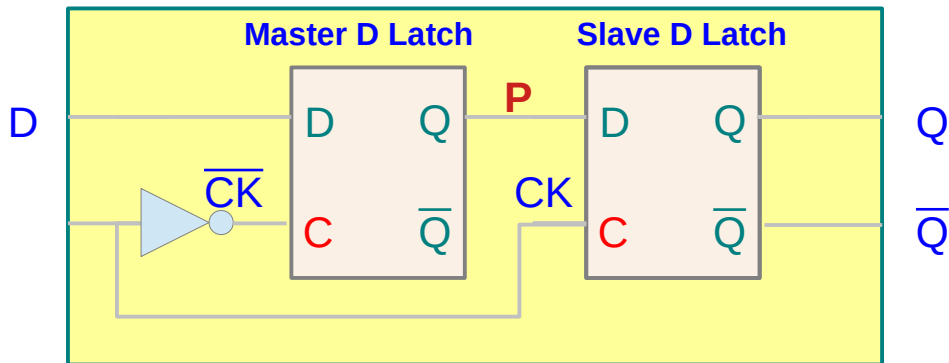
## (2) Using flipflop outputs Q



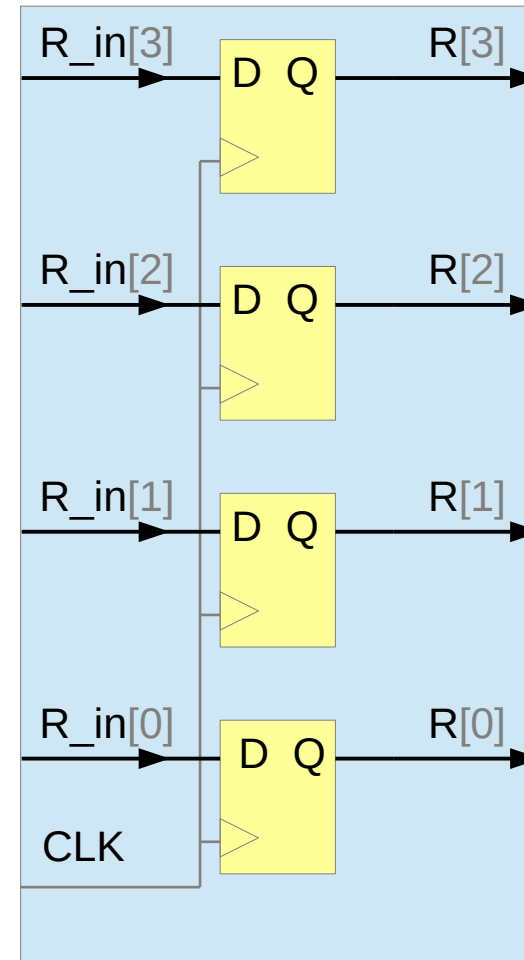
# (1) 4-bit Register Using master latch outputs **P**



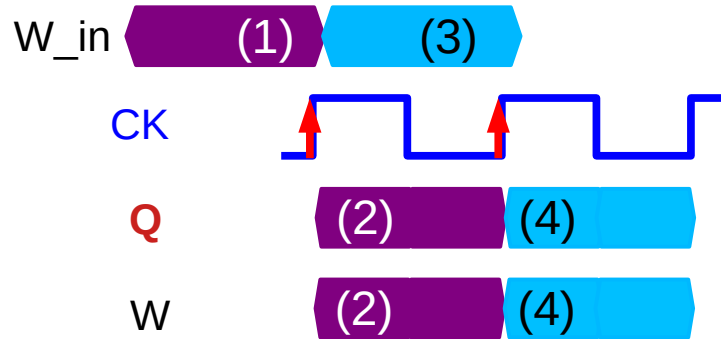
D FlipFlop using Master and Slave D Latches



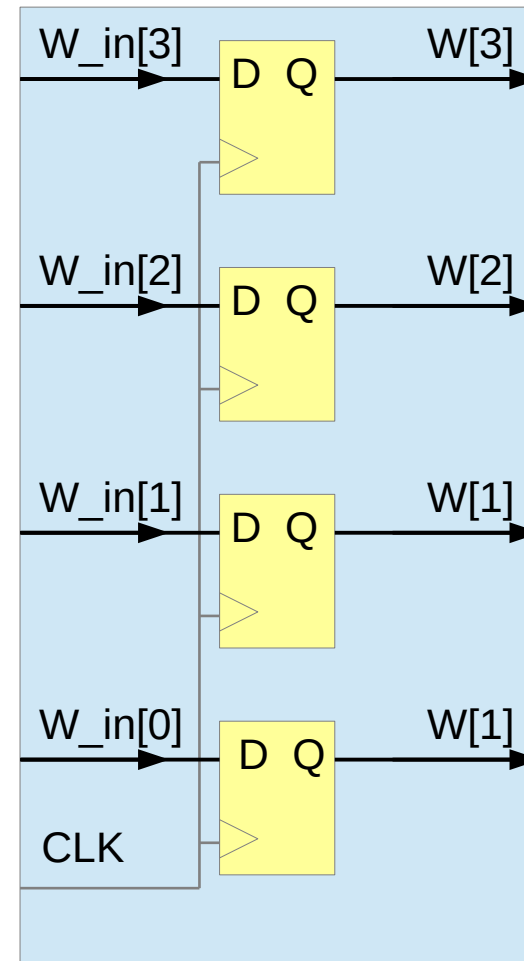
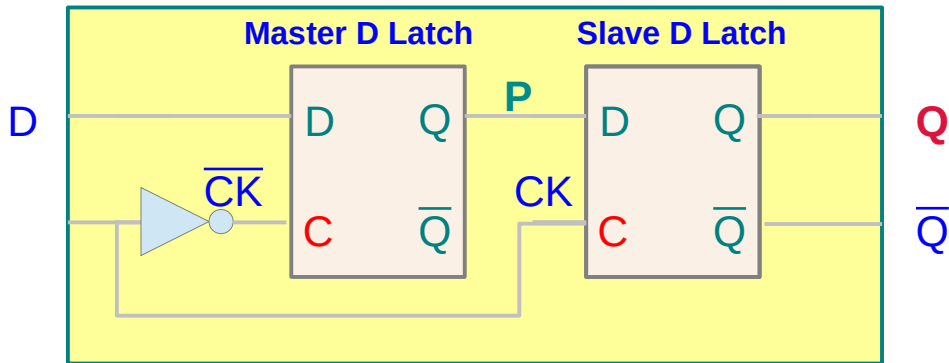
4-bit Register using 4 D flipflops



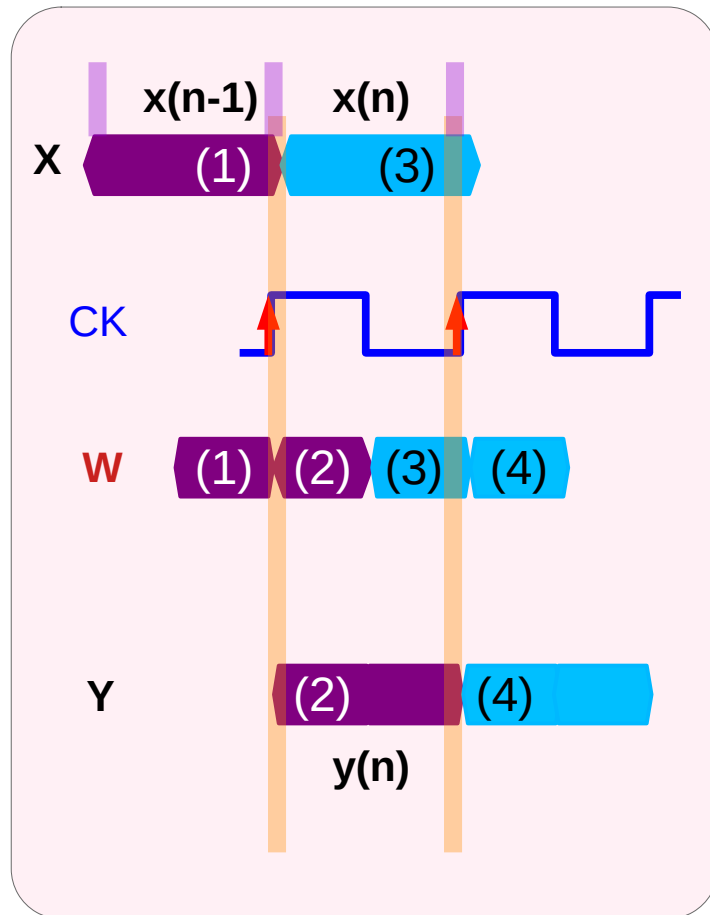
## (2) 4-bit Register Using flipflop outputs Q



D FlipFlop using Master and Slave D Latches



# (1) Timing diagrams with master latch outputs **P**



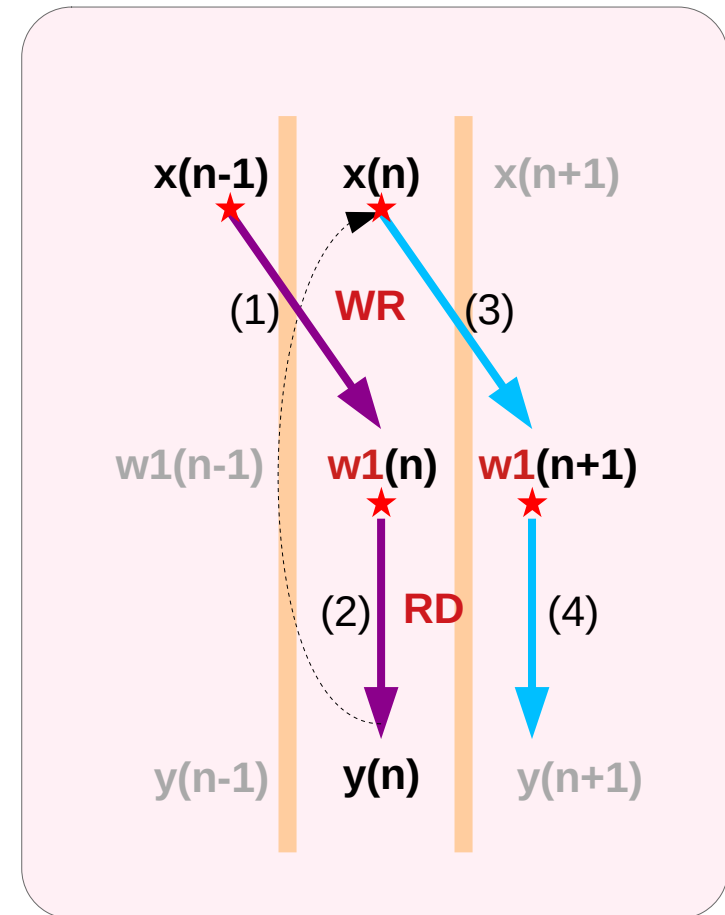
Hardware model

data centered view

Input

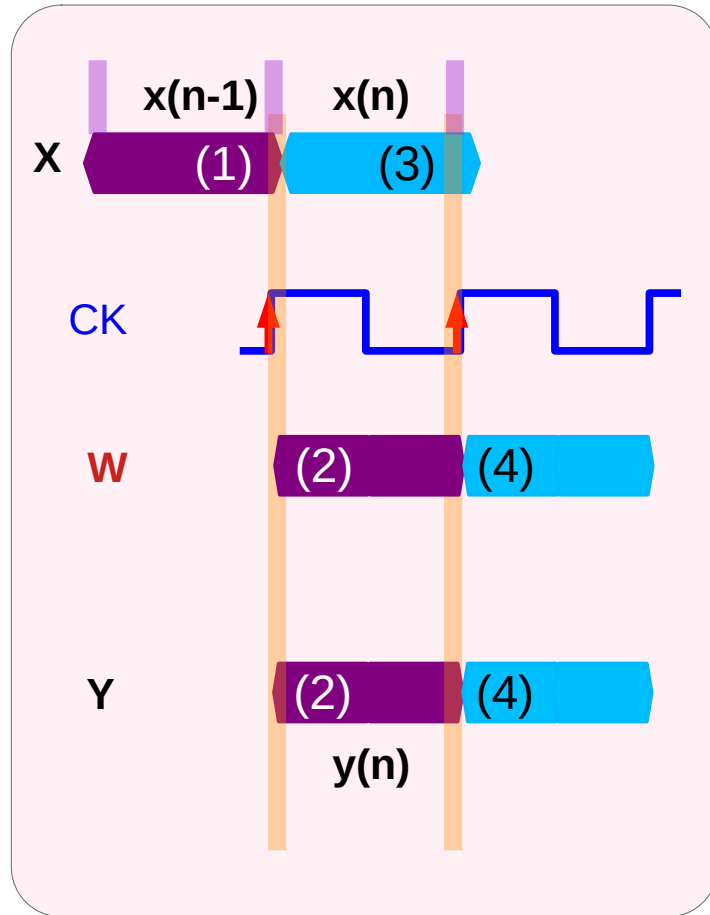
Internal State

Output



The same DSP C model

## (2) Timing diagrams with flipflop outputs Q



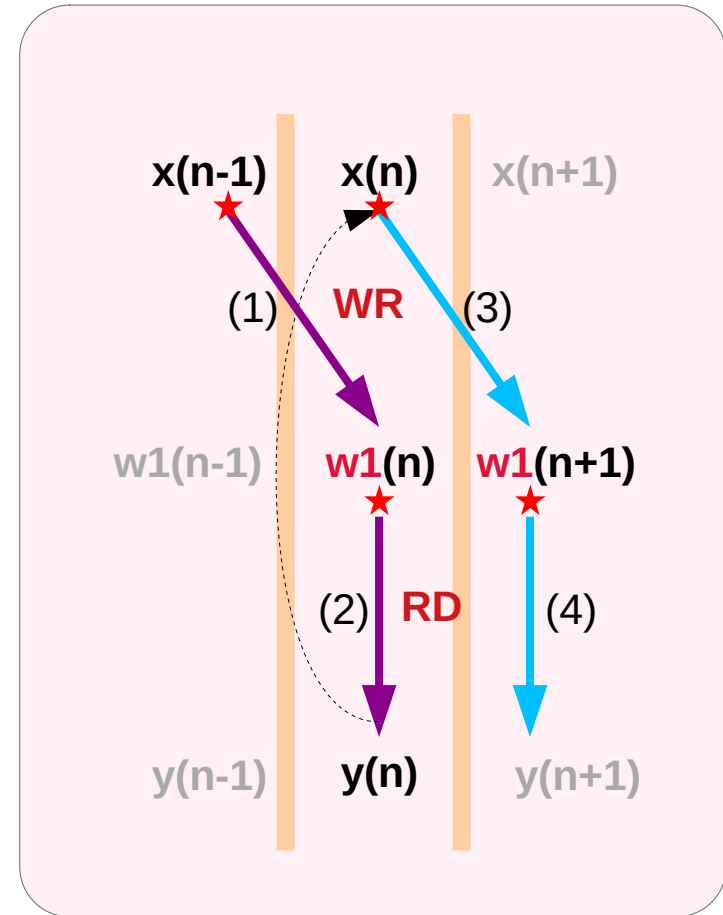
Hardware model

data centered view

Input

Simplified Internal State

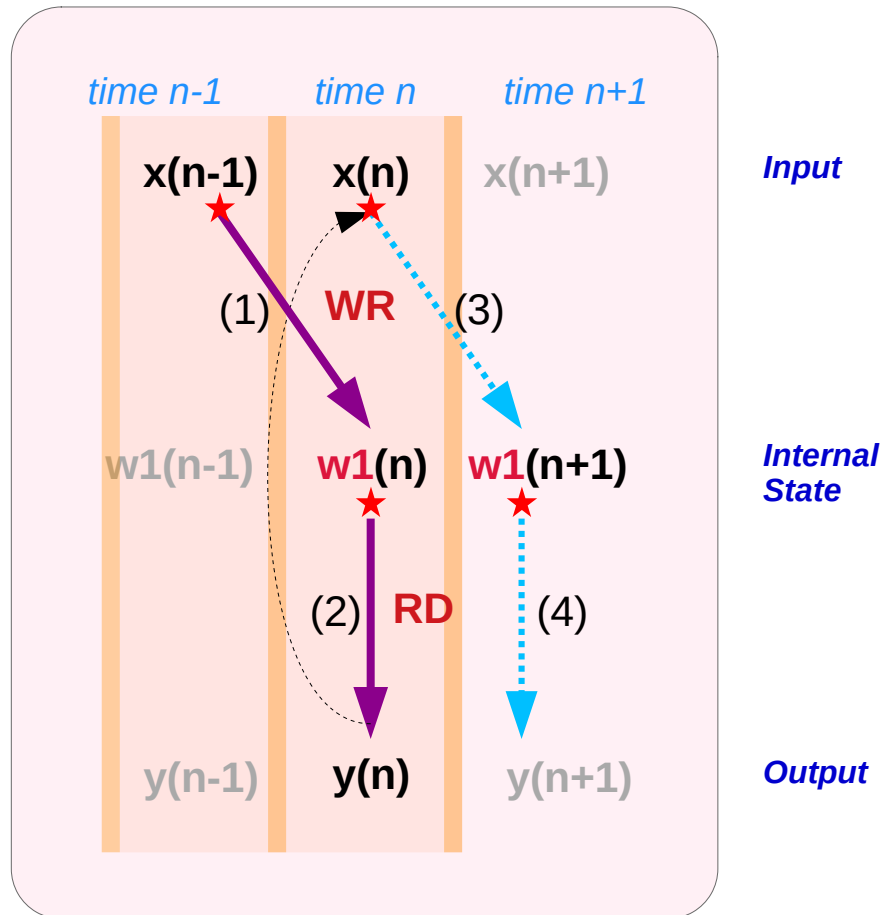
Output



The same DSP C model

# Data centered operations

## data centered operations



## DSP C model

operation spans two time steps ( $n-1, n$ )

- (1)  $w1(n) = x(n-1)$       *WR*  $w1(n)$   
 (2)  $y(n) = w1(n)$       *RD*  $w1(n)$

first **WR**, then **RD**  
 must avoid **RAW (read after write)** hazards

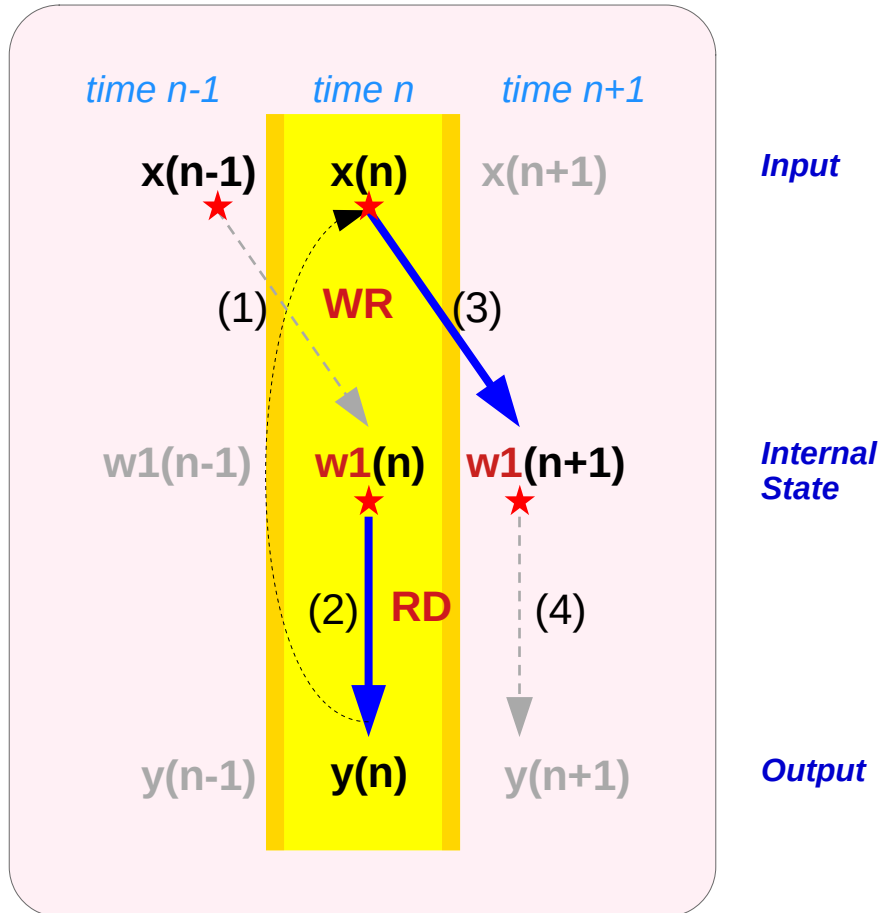
here, because the written data is read out,  
 there is no **RAW (read after write)** hazards

$$\begin{array}{l}
 \underline{y(n-1) = w1(n-1)} \quad \underline{w1(n) = w1(n-1)} \\
 \underline{w1(n) = x(n-1)} \\
 \underline{y(n) = w1(n)} \quad \underline{w1(n+1) = w1(n)} \\
 \underline{w1(n+1) = x(n)} \\
 \underline{y(n+1) = w1(n+1)} \quad \underline{w1(n+2) = w1(n+1)} \\
 \underline{w1(n+2) = x(n+1)} \\
 \underline{y(n+2) = w1(n+2)}
 \end{array}$$



# Time centered operations

## time centered operations



## DSP C model

## operation spans only one time step (n)

- (2)  $y(n) = w1(n)$       *RD*  $w1(n)$   
 (3)  $w1(n+1) = x(n)$       *WR*  $w1(n)$

first **RD**, then **WR**  
 no **RAW (read after write)** hazards

$y(n-1) = w1(n-1)$	$w1(n) = w1(n-1)$
$w1(n) = x(n-1)$	
$y(n) = w1(n)$	$w1(n+1) = w1(n)$
$w1(n+1) = x(n)$	
$y(n+1) = w1(n+1)$	$w1(n+2) = w1(n+1)$
$w1(n+2) = x(n+1)$	
$y(n+2) = w1(n+2)$	

# Data vs. time centered operations (1)

## data centered operations

## time centered operations

$y(n-1) = w1(n-1)$ <i>output from w1(n-1)</i> <hr/> $w1(n) = x(n-1)$ <i>input to w1(n)</i>	$w1(n) = w1(n-1)$	$y(n-1) = w1(n-1)$ <i>output from w1(n-1)</i> <hr/> $w1(n) = x(n-1)$ <i>input to w1(n)</i>	$w1(n) = w1(n-1)$	} <i>time n-1</i> } <i>time n</i> } <i>time n+1</i> } <i>time n+2</i>
$y(n) = w1(n)$ <i>output from w1(n)</i> <hr/> $w1(n+1) = x(n)$ <i>input to w1(n+1)</i>	$w1(n+1) = w1(n)$	$y(n) = w1(n)$ <i>output from w1(n)</i> <hr/> $w1(n+1) = x(n)$ <i>input to w1(n+1)</i>	$w1(n+1) = w1(n)$	
$y(n+1) = w1(n+1)$ <i>output from w1(n+1)</i> <hr/> $w1(n+2) = x(n+1)$ <i>input to w1(n+2)</i>	$w1(n+2) = w1(n+1)$	$y(n+1) = w1(n+1)$ <i>output from w1(n+1)</i> <hr/> $w1(n+2) = x(n+1)$ <i>input to w1(n+2)</i>	$w1(n+2) = w1(n+1)$	
$y(n+2) = w1(n+2)$ <i>output from w1(n+2)</i>		$y(n+2) = w1(n+2)$ <i>output from w1(n+2)</i>		

- (1)  $w1(n) = x(n-1)$       *WR w1(n)*  
 (2)  $y(n) = w1(n)$       *RD w1(n)*

- (2)  $y(n) = w1(n)$       *RD w1(n)*  
 (1)  $w1(n) = x(n)$       *WR w1(n)*

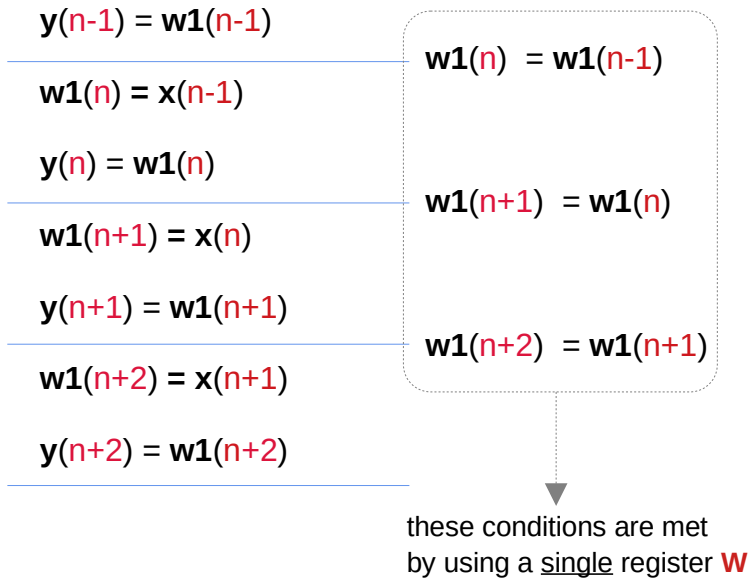
$W1 = X$	<i>WR W1</i>
$Y = W1$	<i>RD W1</i>

$Y = W1$	<i>RD W1</i>
$W1 = X$	<i>WR W1</i>

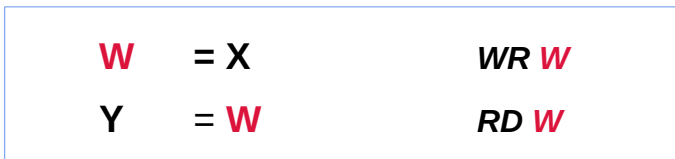
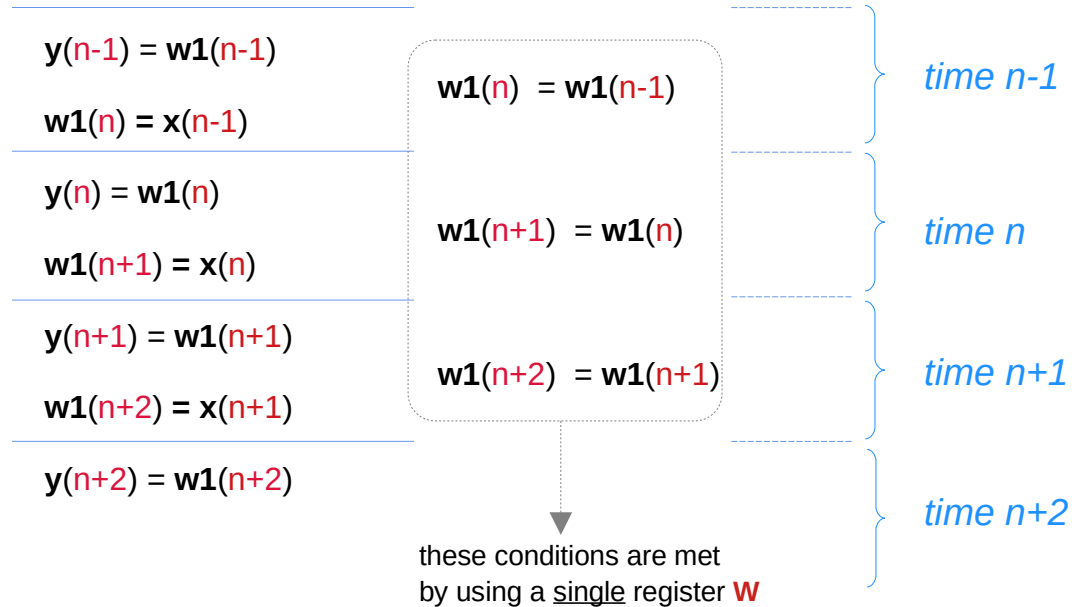
← *Register Transfer Level*

# Data vs. time centered operations (2)

## data centered operations



## time centered operations



# Data centered operations

## data centered operations

## in the Register Transfer Level

$y(n-1) = w1(n-1)$ <small>output <math>y(n-1)</math> from <math>w1(n-1)</math></small> $w1(n) = x(n-1)$ <small>input <math>x(n-1)</math> to <math>w1(n)</math></small>	$w1(n) = w1(n-1)$	$Y = W1$ <small>output <math>y(n-1)</math> from <math>W1</math></small> $W1 = X$ <small>input <math>x(n-1)</math> to <math>W1</math></small>	} time n-1
$y(n) = w1(n)$ <small>output <math>y(n)</math> from <math>w1(n)</math></small> $w1(n+1) = x(n)$ <small>input <math>x(n)</math> to <math>w1(n+1)</math></small>	$w1(n+1) = w1(n)$	$Y = W1$ <small>output <math>y(n)</math> from <math>W1</math></small> $W1 = X$ <small>input <math>x(n)</math> to <math>W1</math></small>	
$y(n+1) = w1(n+1)$ <small>output <math>y(n+1)</math> from <math>w1(n+1)</math></small> $w1(n+2) = x(n+1)$ <small>input <math>x(n+1)</math> to <math>w1(n+2)</math></small>	$w1(n+2) = w1(n+1)$	$Y = W1$ <small>output <math>y(n+1)</math> from <math>W1</math></small> $W1 = X$ <small>input <math>x(n+1)</math> to <math>W1</math></small>	} time n+1
$y(n+2) = w1(n+2)$ <small>output <math>y(n+2)</math> from <math>w1(n+2)</math></small>		$Y = W1$ <small>output <math>y(n+2)</math> from <math>W1</math></small>	
(1) $w1(n) = x(n-1)$ $WR\ w1(n)$ (2) $y(n) = w1(n)$ $RD\ w1(n)$	$W1 = X$ $WR\ W1$ $Y = W1$ $RD\ W1$		

# Time centered operations

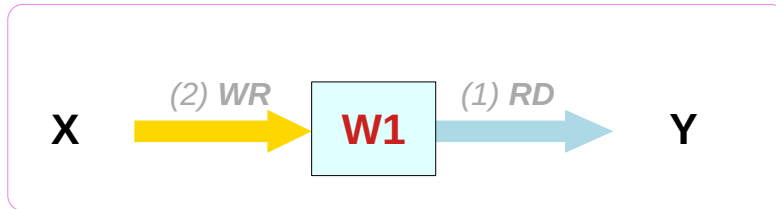
## time centered operations

## in the Register Transfer Level

$y(n-1) = w1(n-1)$ <small>output from <math>w1(n-1)</math></small> $w1(n) = x(n-1)$ <small>input to <math>w1(n)</math></small>	$w1(n) = w1(n-1)$	$Y = W1$ <small>output <math>y(n-1)</math> from <math>W1</math></small> $W1 = X$ <small>input <math>x(n-1)</math> to <math>W1</math></small>	} time n-1 } time n } time n+1 } time n+2
$y(n) = w1(n)$ <small>output from <math>w1(n)</math></small> $w1(n+1) = x(n)$ <small>input to <math>w1(n+1)</math></small>	$w1(n+1) = w1(n)$	$Y = W1$ <small>output <math>y(n)</math> from <math>W1</math></small> $W1 = X$ <small>input <math>x(n)</math> to <math>W1</math></small>	
$y(n+1) = w1(n+1)$ <small>output from <math>w1(n+1)</math></small> $w1(n+2) = x(n+1)$ <small>input to <math>w1(n+2)</math></small>	$w1(n+2) = w1(n+1)$	$Y = W1$ <small>output <math>y(n+1)</math> from <math>W1</math></small> $W1 = X$ <small>input <math>x(n+1)</math> to <math>W1</math></small>	
$y(n+2) = w1(n+2)$ <small>output from <math>w1(n+2)</math></small>		$Y = W1$ <small>output <math>y(n+2)</math> from <math>W1</math></small>	
(2) $y(n) = w1(n)$ <span style="margin-left: 100px;"><math>RD\ w1(n)</math></span>		$Y = W1$ <span style="margin-left: 100px;"><math>RD\ W1</math></span>	
(1) $w1(n) = x(n)$ <span style="margin-left: 100px;"><math>WR\ w1(n)</math></span>		$W1 = X$ <span style="margin-left: 100px;"><math>WR\ W1</math></span>	

# Simultaneous RD and WR actions

## Register Transfer Level



at time  $n$

at time  $n+1$

$$Y = W1$$

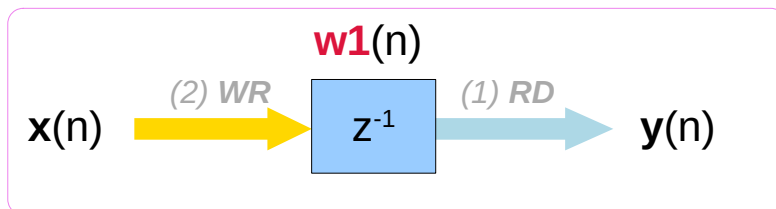
$$W1 = X$$

$$Y = W1$$

$$W1 = X$$

(1) RD  
(2) WR

## DSP C Model



at time  $n$

at time  $n+1$

$$y(n) = w1(n)$$

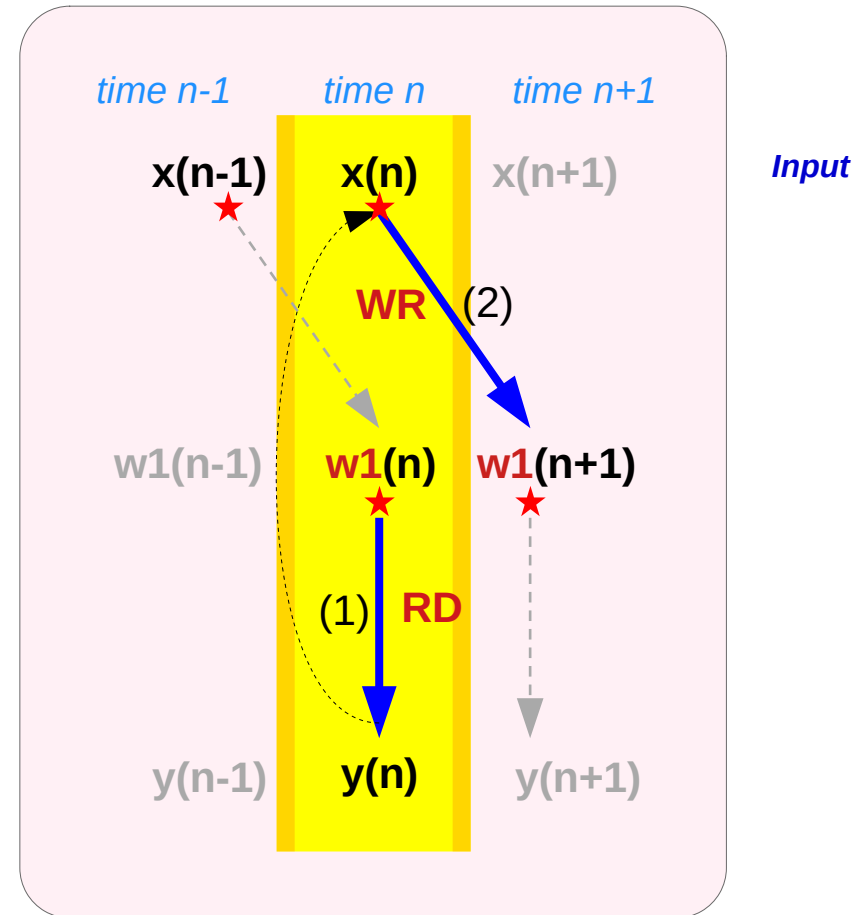
$$w1(n+1) = x(n)$$

$$y(n+1) = w1(n+1)$$

$$w1(n+2) = x(n+1)$$

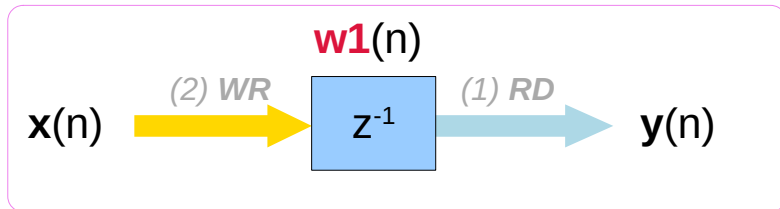
(1) RD  
(2) WR

## time centered operations



## DSP C model

# Simultaneous **RD** and **WR** actions



current content	<b>w1(n)</b>	= <b>x(n-1)</b>
next content	<b>w1(n+1)</b>	= <b>x(n)</b>

at time  $n$

$$y(n) = w1(n)$$

$$w1(n+1) = x(n)$$

at time  $n+1$

$$y(n+1) = w1(n+1)$$

$$w1(n+2) = x(n+1)$$

- (1) **RD**    *read internal state*  
 (2) **WR**    *update internal state*

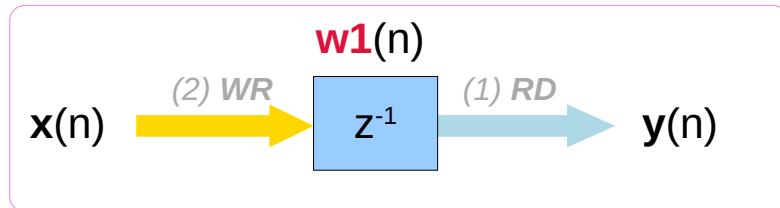
at time  $n$ ,

- 1) the content of the register **w1(n)**  
becomes the output **y(n)**
- 2) the input **x(n)** is saved and  
becomes the new content **w1(n+1)**

**RD** access of **w1(n) = x(n-1)**

**WR** access of **w1(n+1) = x(n)**

# Current content $w1(n)$ and current input $x(n)$

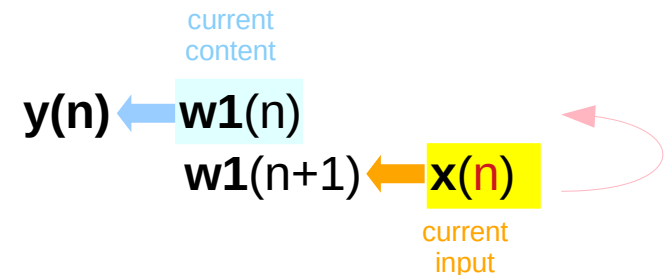


current content	$w1(n)$	$= x(n-1)$
next content	$w1(n+1)$	$= x(n)$

a register holding the previous input sample  $x(n-1)$

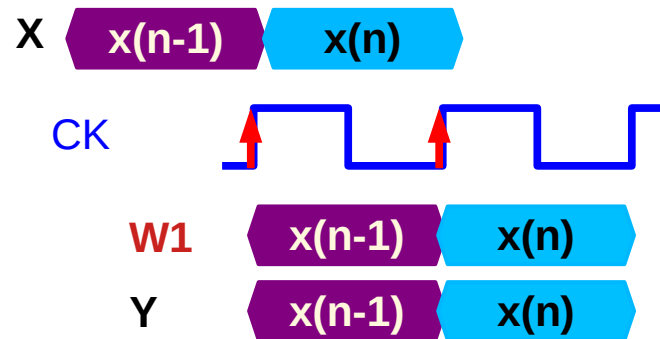
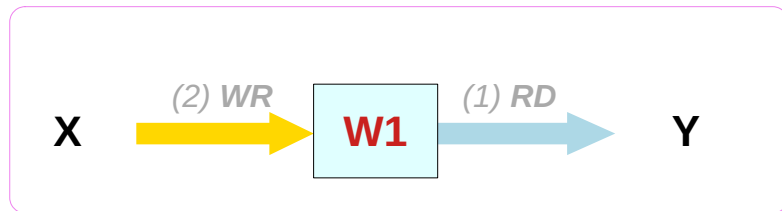
- (1) the **current content**  $x(n-1)$  is clocked out to the output
- (2) the **current input**  $x(n)$  gets stored in the register

It will be held for one sampling instant and become the output at the next time  $n+1$





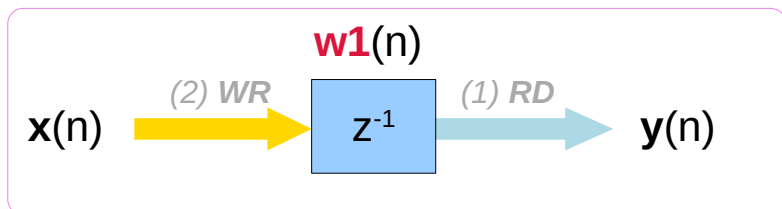
# Current content $w1(n)$ and current input $x(n)$



*simulate a clocked hardware  
ignoring delay constraints in hardware*

*zero-delay simulation*

# Delay element modeling



$w1(n) \longrightarrow y(n)$	$y(n) = w1(n)$ (1) RD old w1
$x(n) \longrightarrow w1(n+1)$	$w1(n+1) = x(n)$ (2) WR new w1

The content of the delay register at time  $n$  as the **internal state** of the filter by

internal state at time  $n$

$$w1(n) = x(n-1)$$

internal state at time  $n+1$

$$w1(n+1) = x(n)$$

output at time  $n$

$$y(n) = w1(n)$$

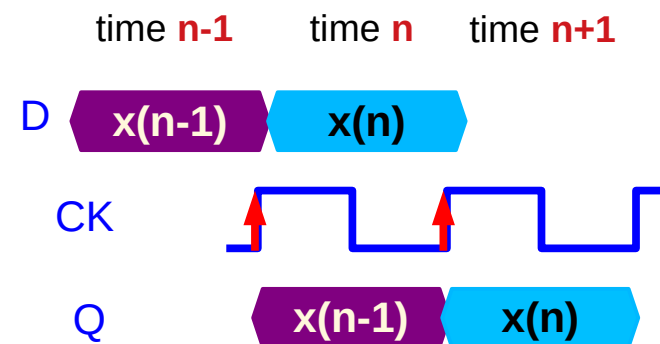
RD before WR

WAR (Write after Read) Access

WR at time  $n-1$

WR at time  $n$

RD at time  $n$



*simulate a clocked hardware  
ignoring delay constraints in hardware*

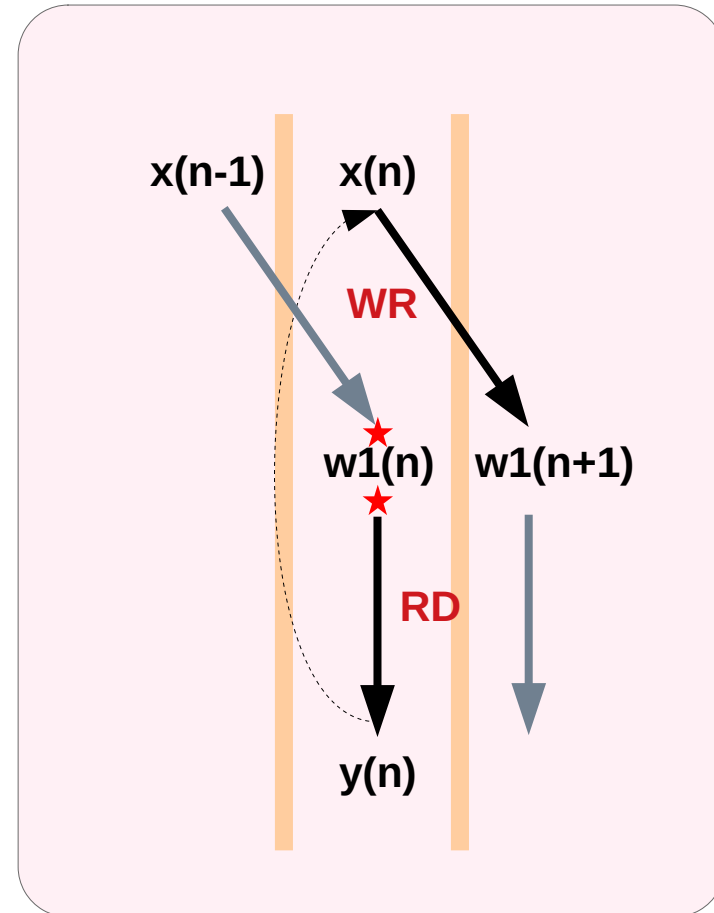
*zero-delay simulation*

# WAR (Write after Read)

$y(n) = w1(n)$	(1) RD	old w1
$w1(n+1) = x(n)$	(2) WR	new w1
$y(n+1) = w1(n+1)$	(1) RD	old w1
$w1(n+2) = x(n+1)$	(2) WR	new w1

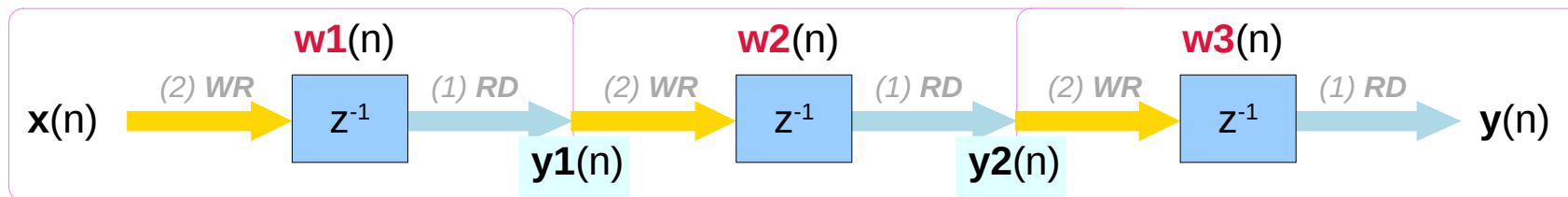
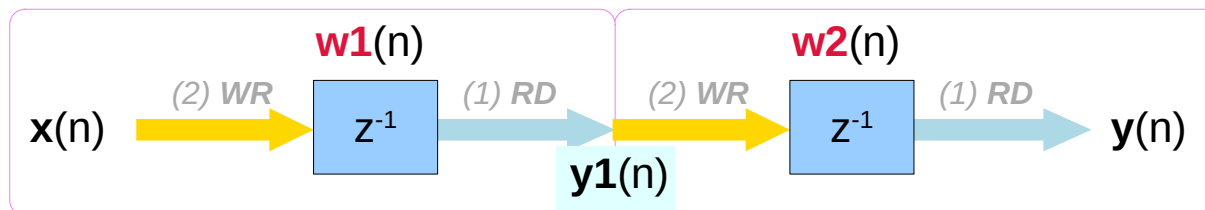
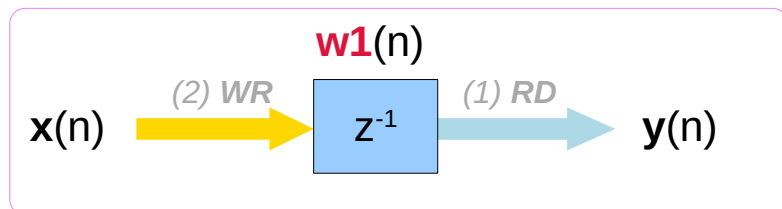
$w1(n+1) = x(n)$	(2) WR	new w1
$y(n) = w1(n)$	(1) RD	old w1
$w1(n+2) = x(n+1)$	(2) WR	new w1
$y(n+1) = w1(n+1)$	(1) RD	old w1

WAR (Write after Read) Violation



DSP C model

# Single, Double, Triple Delay – Summary



single delay

$$\begin{aligned} y(n) &= w1(n) && \text{output} \\ w1(n+1) &= x(n) && \text{input} \end{aligned}$$

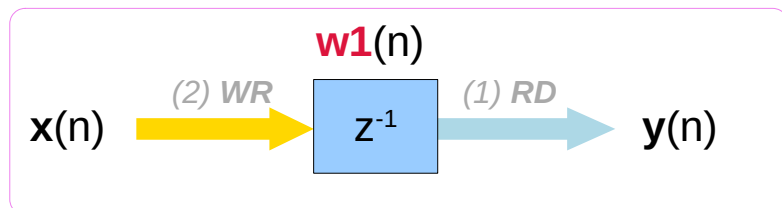
double delay

$$\begin{aligned} y(n) &= w2(n) && \text{output} \\ w2(n+1) &= w1(n) && \\ w1(n+1) &= x(n) && \text{input} \end{aligned}$$

triple delay

$$\begin{aligned} y(n) &= w3(n) && \text{output} \\ w3(n+1) &= w2(n) && \\ w2(n+1) &= w1(n) && \\ w1(n+1) &= x(n) && \text{input} \end{aligned}$$

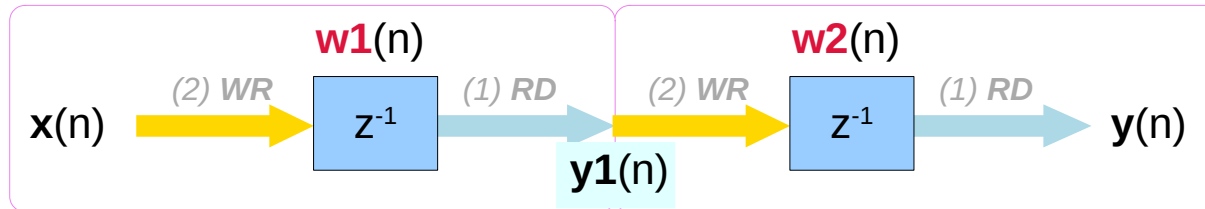
# Single Delay – IO Equations



single delay

$y(n) = w1(n)$	output
$w1(n+1) = x(n)$	input

# Double Delay – IO Equations



$$\begin{aligned} \mathbf{y1}(n) &= \mathbf{w1}(n) \\ \mathbf{w1}(n+1) &= \mathbf{x}(n) \end{aligned}$$

$$\mathbf{w1}(n+1) = \mathbf{x}(n)$$



$$\begin{aligned} \mathbf{y}(n) &= \mathbf{w2}(n) \\ \mathbf{w2}(n+1) &= \mathbf{y1}(n) \end{aligned}$$

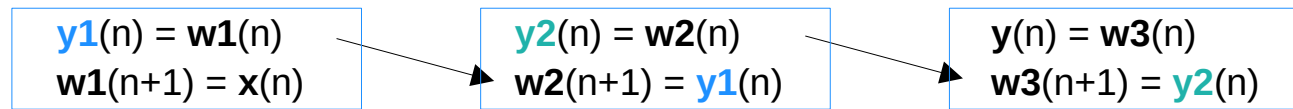
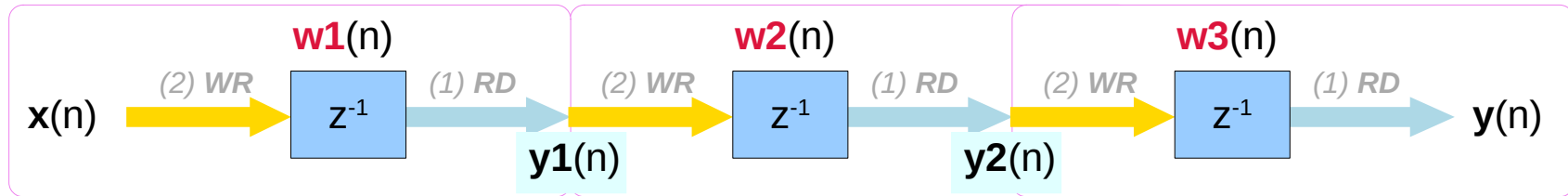
$$\begin{aligned} \mathbf{y}(n) &= \mathbf{w2}(n) \\ \mathbf{w2}(n+1) &= \mathbf{w1}(n) \end{aligned}$$

$$\begin{aligned} \mathbf{y}(n) &= \mathbf{w2}(n) \\ \mathbf{w2}(n+1) &= \mathbf{w1}(n) \\ \mathbf{w1}(n+1) &= \mathbf{x}(n) \end{aligned}$$

output

input

# Triple Delay – IO Equations



$$w_1(n+1) = x(n)$$

$$w_2(n+1) = w_1(n)$$

$$y(n) = w_3(n)$$

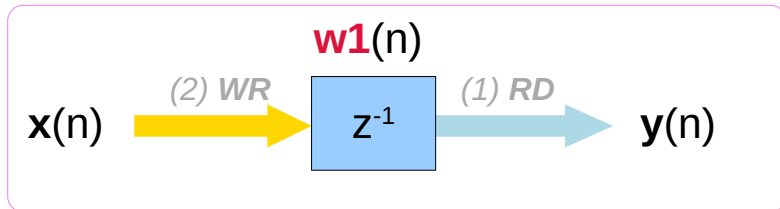
$$w_3(n+1) = w_2(n)$$

$y(n) = w_3(n)$   
 $w_3(n+1) = w_2(n)$   
 $w_2(n+1) = w_1(n)$   
 $w_1(n+1) = x(n)$

output

input

# Single Delay

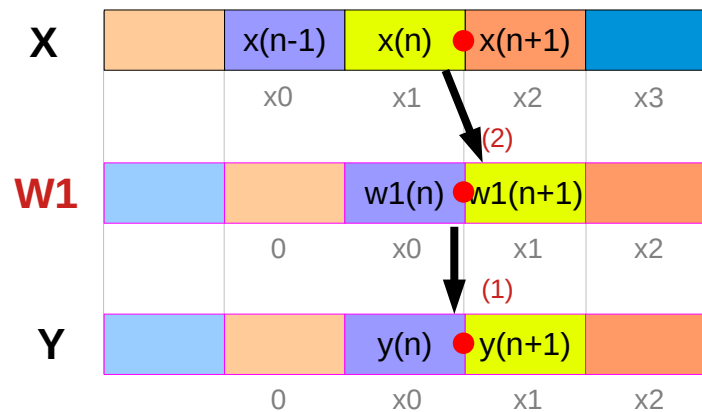


single delay

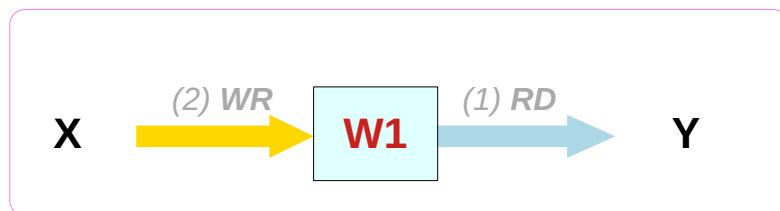
$$y(n) = w1(n)$$

$$w1(n+1) = x(n)$$

output  
input

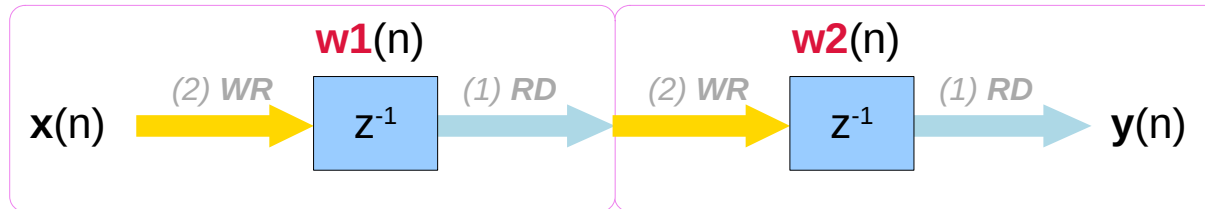


n	x(n)	w1(n)	y(n)
0	x0	0	0
1	x1	x0	x0
2	x2	x1	x1
3	x3	x2	x2
4	x4	x3	x3



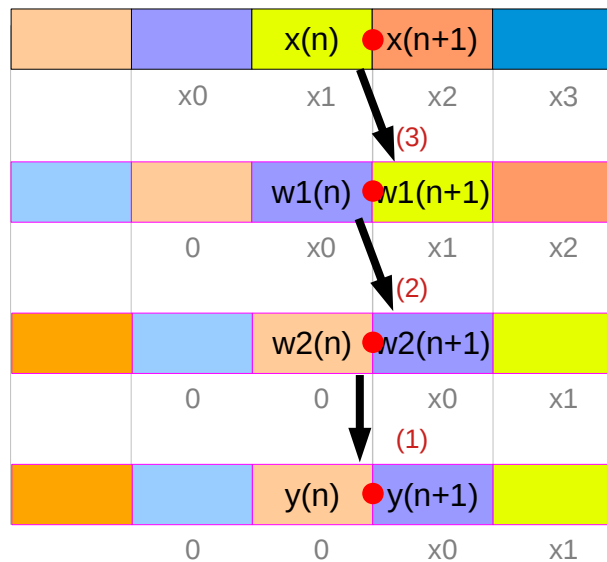


# Double Delay



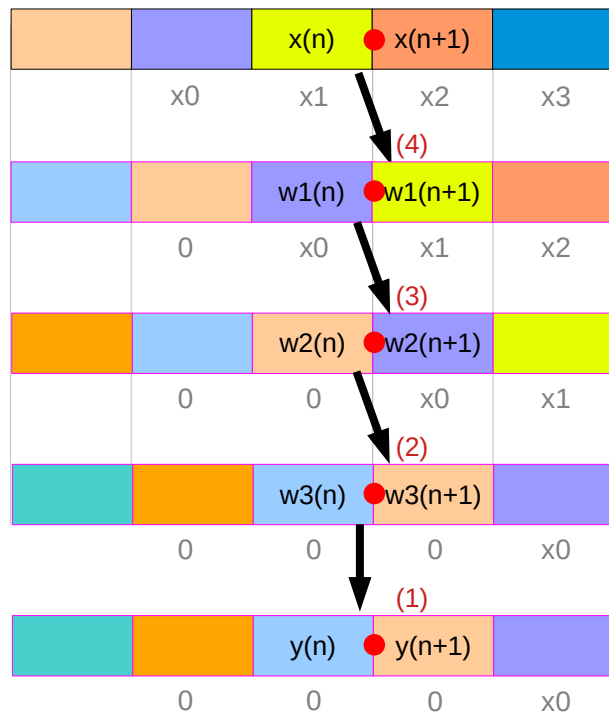
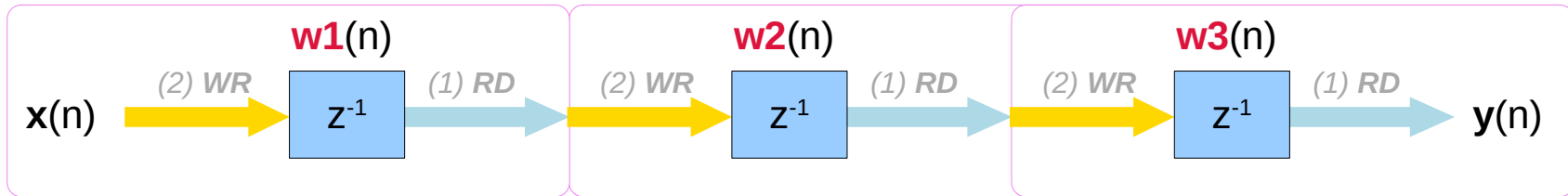
double delay

$$\begin{aligned}
 y(n) &= w2(n) && \text{output} \\
 w2(n+1) &= w1(n) \\
 w1(n+1) &= x(n) && \text{input}
 \end{aligned}$$



n	x(n)	w1(n)	w2(n)	y(n)
0	x0	0	0	0
1	x1	x0	0	0
2	x2	x1	x0	x0
3	x3	x2	x1	x1
4	x4	x3	x2	x2

# Triple Delay



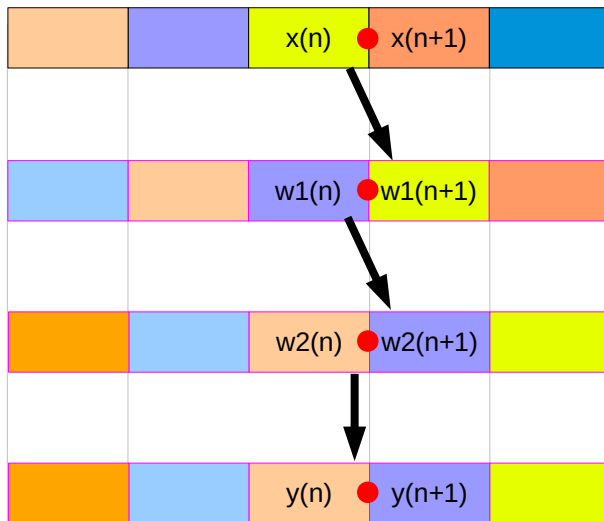
triple delay

$$\begin{aligned}
 y(n) &= w3(n) && \text{output} \\
 w3(n+1) &= w2(n) \\
 w2(n+1) &= w1(n) \\
 w1(n+1) &= x(n) && \text{input}
 \end{aligned}$$

n	x(n)	w1(n)	w2(n)	w3(n)	y(n)
0	x0	0	0	0	0
1	x1	x0	0	0	0
2	x2	x1	x0	0	0
3	x3	x2	x1	x0	x0
4	x4	x3	x2	x1	x1

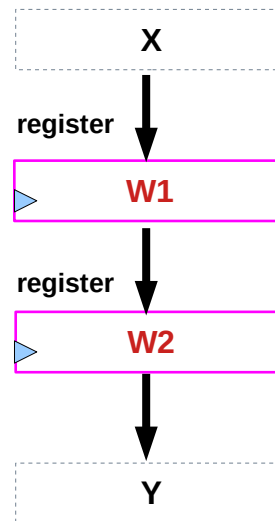
# Delay C Model

Timing Chart



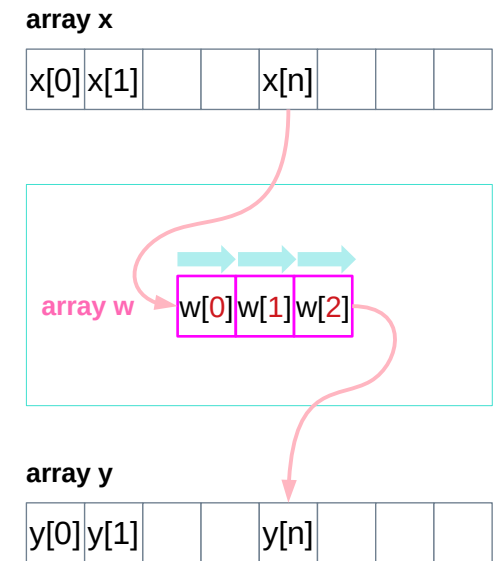
$$\begin{aligned}
 y(n) &= w2(n) \\
 w2(n+1) &= w1(n) \\
 w1(n+1) &= x(n)
 \end{aligned}$$

Register Transfer



$$\begin{aligned}
 Y &= W2 \\
 W2 &= W1 \\
 W1 &= X
 \end{aligned}$$

DSP C Model for simulation



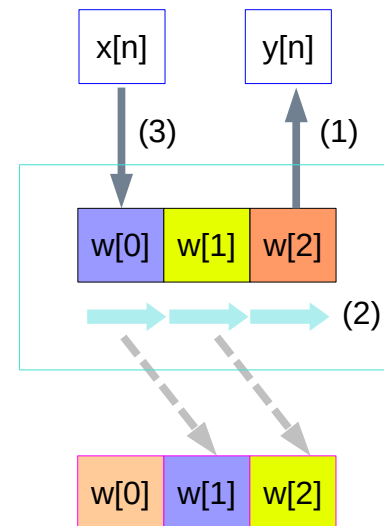
$$\begin{aligned}
 y[n] &= w[2] \\
 w[2] &= w[1] \\
 w[1] &= w[0] \\
 w[0] &= x[n]
 \end{aligned}$$

# IO Equations for the Triple Delay (1)

$$\begin{aligned}y(n) &= w_2(n) \\w_2(n+1) &= w_1(n) \\w_1(n+1) &= w_0(n) \\w_0(n+1) &= x(n)\end{aligned}$$

$$D = 2, 1$$

$$\begin{aligned}y[n] &= w[2] && // (1) \text{ get the output} \\w[2] &= w[1] && // (2) \text{ shift} \\w[1] &= w[0] && // (2) \text{ shift} \\w[0] &= x[n] && // (3) \text{ put the input}\end{aligned}$$



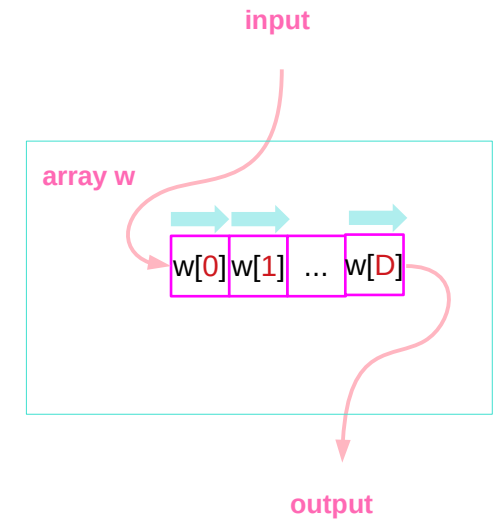
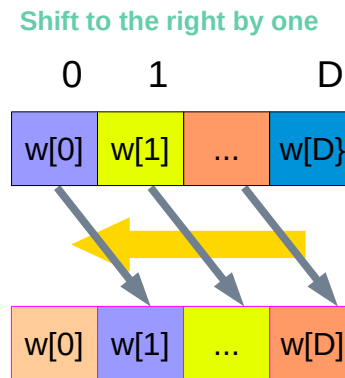
# delay.c

```
/* delay.c - delay by D time samples */
/* w[0] = input, w[D] = output */

void delay(int D, double *w)
{
    int i;

    for (i=D; i>=1; i--)
        w[i] = w[i-1];

    // reverse-order updating
}
```



order of execution

$$\begin{aligned} w[D] &= w[D-1] \\ \dots & \\ w[2] &= w[1] \\ w[1] &= w[0] \end{aligned}$$

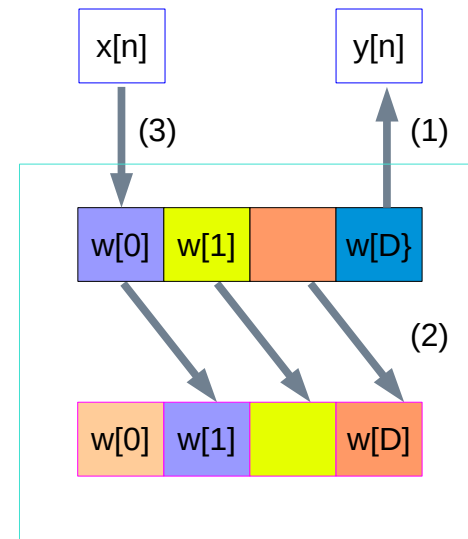
# Using the delay function

```
double *w;  
w = (double *) calloc(D+1, sizeof(double)); // (D+1)-dimensional
```

```
for (n = 0; n < Ntot; n++) {  
    y[n] = w[D]; // (1) get output y[n]  
    w[0] = x[n]; // (3) read input x[n]  
    delay(D, w); // (2) update delay line  
}
```

```
/* delay.c - delay by D time samples */  
/* w[0] = input, w[D] = output */
```

```
void delay(int D, double *w)  
{  
    int i;  
  
    for (i=D; i>=1; i--)  
        w[i] = w[i-1];  
  
    // reverse-order updating  
}
```



# Delay Functions

$$y(n) = w_1(n)$$

$$w_1(n+1) = x(n)$$

$$y(n) = w_2(n)$$

$$w_2(n+1) = w_1(n)$$

$$w_1(n+1) = x(n)$$

$$y(n) = w_3(n)$$

$$w_3(n+1) = w_2(n)$$

$$w_2(n+1) = w_1(n)$$

$$w_1(n+1) = x(n)$$

$$y(n) = w_D(n)$$

$$w_0(n) = x(n)$$

$$w_i(n+1) = w_{i-1}(n),$$

$$i = D, D-1, \dots, 2, 1$$

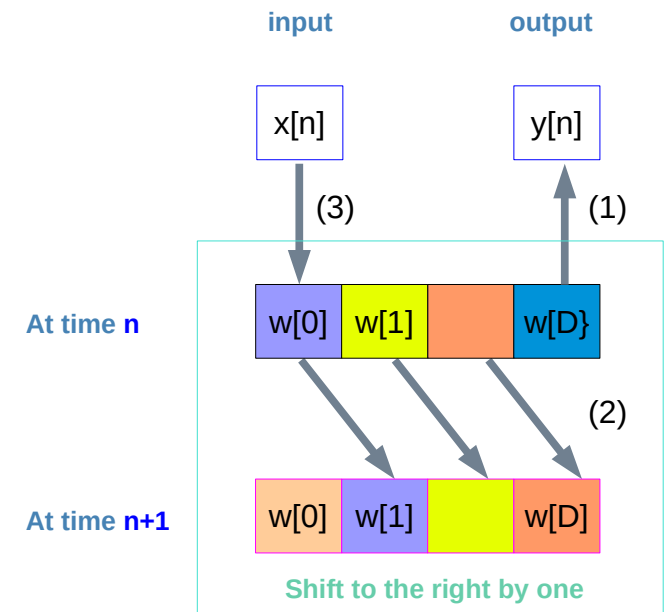
time index :  $n$

memory location :  $W_i$

memory index :  $i$

$$w_i(n+1) = w_{i-1}(n)$$

the current value at  $w_{i-1}$   
will become  
the next value at  $w_i$



# Holding a delayed input sequence

$$w_0(n) = x(n)$$

$$w_1(n) = x(n-1) = w_0(n-1)$$

$$w_2(n) = x(n-2) = w_1(n-1)$$

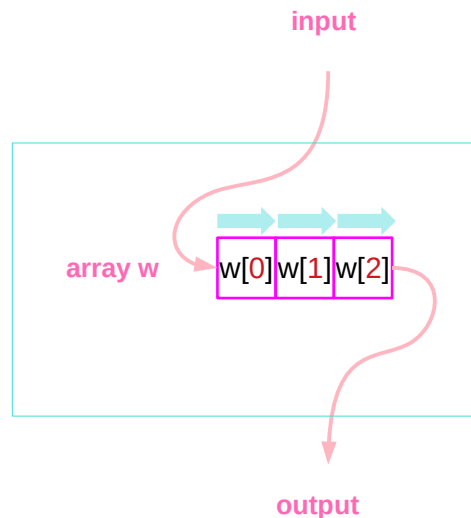
$$w_3(n) = x(n-3) = w_2(n-1)$$

$$y(n) = w_3(n)$$

$$w_3(n+1) = w_2(n)$$

$$w_2(n+1) = w_1(n)$$

$$w_1(n+1) = x(n)$$



$$y(n) = w_3(n) = w_2(n-1) = w_1(n-2) = x(n-4)$$

$$w_3(n) = w_2(n-1) = w_1(n-2) = x(n-3)$$

$$w_2(n) = w_1(n-1) = x(n-2)$$

$$w_1(n) = x(n-1)$$

$$y(n) = w_3(n)$$

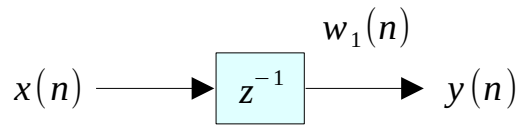
$$w_3(n) = w_2(n-1)$$

$$w_2(n) = w_1(n-1)$$

$$w_1(n) = x(n-1)$$



# Single Delay (1)



$$w_1(n) = x(n-1)$$

(internal state at time n)



$$w_1(n+1) = x(n)$$

(internal state at time n+1)

$$y(n) = w_1(n)$$

$$w_1(n+1) = x(n)$$

$$y(n+1) = w_1(n+1)$$

$$w_1(n+2) = x(n+1)$$

$$w_1(0) = 0$$

$$[x_0, x_1, x_2, x_3, \dots] \rightarrow [0, x_0, x_1, x_2, \dots]$$

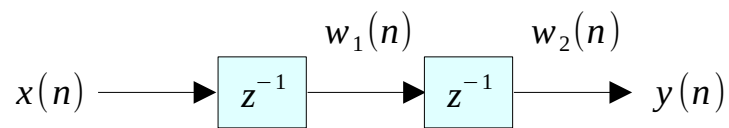
for each input sample  $x$  do:

$$y := w_1$$

$$w_1 := x$$

$n$	$x(n)$	$w_1(n)$	$y(n)$
0	$x_0$	0	0
1	$x_1$	$x_0$	$x_0$
2	$x_2$	$x_1$	$x_1$
3	$x_3$	$x_2$	$x_2$
4	$x_4$	$x_3$	$x_3$

# Double Delay (1)



$$w_2(n) = w_1(n-1) = x((n-1)-1) = x(n-2)$$

$$w_1(n) = x(n-1)$$



$$w_2(n+1) = w_1(n)$$

$$w_1(n+1) = x(n)$$

$$y(n) = w_2(n)$$

$$w_2(n+1) = w_1(n)$$

$$w_1(n+1) = x(n)$$

$$[x_0, x_1, x_2, x_3, \dots] \rightarrow [0, 0, x_0, x_1, \dots]$$

for each input sample  $x$  do:

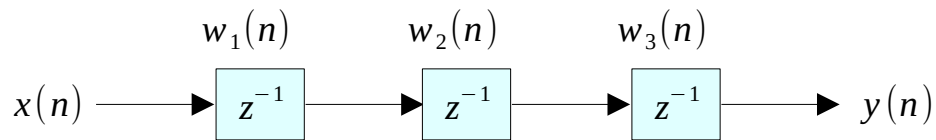
$$y := w_2$$

$$w_2 := w_1$$

$$w_1 := x$$

$n$	$x(n)$	$w_1(n)$	$w_2(n)$	$y(n)$
0	$x_0$	0	0	0
1	$x_1$	$x_0$	0	0
2	$x_2$	$x_1$	$x_0$	$x_0$
3	$x_3$	$x_2$	$x_1$	$x_1$
4	$x_4$	$x_3$	$x_2$	$x_2$

# Triple Delay (1)



$$\begin{aligned}
 w_3(n) &= w_2(n-1) & = w_1(n-2) & = x(n-3) \\
 w_2(n) &= w_1(n-1) \\
 w_1(n) &= x(n-1)
 \end{aligned}$$



$$\begin{aligned}
 w_3(n+1) &= w_2(n) \\
 w_2(n+1) &= w_1(n) \\
 w_1(n+1) &= x(n)
 \end{aligned}$$

$$y(n) = w_3(n)$$

$$\begin{aligned}
 w_3(n+1) &= w_2(n) \\
 w_2(n+1) &= w_1(n) \\
 w_1(n+1) &= x(n)
 \end{aligned}$$

$$[x_0, x_1, x_2, x_3, \dots] \rightarrow [0, 0, 0, x_0, \dots]$$

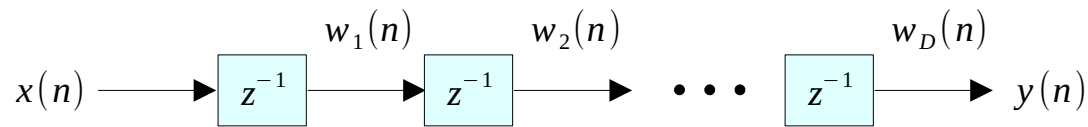
for each input sample  $x$  do:

```

y := w3
w3 := w2
w2 := w1
w1 := x
    
```

$n$	$x(n)$	$w_1(n)$	$w_2(n)$	$y(n)$
0	$x_0$	0	0	0
1	$x_1$	$x_0$	0	0
2	$x_2$	$x_1$	$x_0$	$x_0$
3	$x_3$	$x_2$	$x_1$	$x_1$
4	$x_4$	$x_3$	$x_2$	$x_2$

# D Unit Delay (1)



$$w_i(n) = w_{i-1}(n-1) \quad \text{for } i = 1, 2, \dots, D$$

$$[x_0, x_1, x_2, x_3, \dots] \rightarrow [0, x_0, x_1, x_2, x_3, \dots]$$

for each input sample  $x$  do:

$y := w_D$

$w_0 := x$

$w_0 := x$

for  $i = D, D-1, \dots, 1$  do:

$w_i := w_{i-1}$

for each input sample  $w_0$  do:

for  $i = D, D-1, \dots, 1$  do:

$w_i := w_{i-1}$

# D Unit Delay (1)

```
/* delay.c - delay by D time samples */
void delay(int D, double *w)      w[0] = input, w[D] = output
{
    int i;

    for (i=D; i>=1; i--)          reverse-order updating
        w[i] = w[i-1];
}
```

# dot

```
/* dot.c - dot product of two length-(M+1) vectors */
double dot(int M, double *h, double *w)          Usage: y = dot(M, h, w);
{
    int i;                                       h = filter vector, w = state vector
    double y;                                   M = filter order

    for (y=0, i=0; i<=M; i++)                   compute dot product
        y += h[i] * w[i];

    return y;
}
```

$$y = h_0 w_0 + h_1 w_1 + \dots + h_M w_M = [h_0, h_1, \dots, h_M] \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} = \mathbf{h}^T \mathbf{w}$$

# Direct Form

Considering the widely used  
Edge triggered  
D-type Flip Flops

$$H(z) = \frac{N(z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

$$y_n = -a_1 y_{n-1} - a_2 y_{n-2} + b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2}$$

---

## References

- [1] S. J. Ofranidis , Introduction to Signal Processing