Computer Organization and Architecture

Module 7

Prof. Indranil Sengupta

Dr. Sarani Bhattacharya

Department of Computer Science and Engineering

IIT Kharagpur

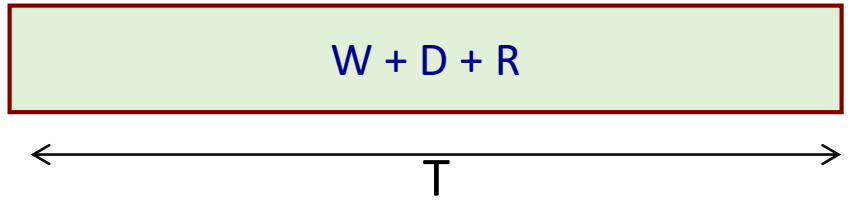
Pipelining

What is Pipelining?

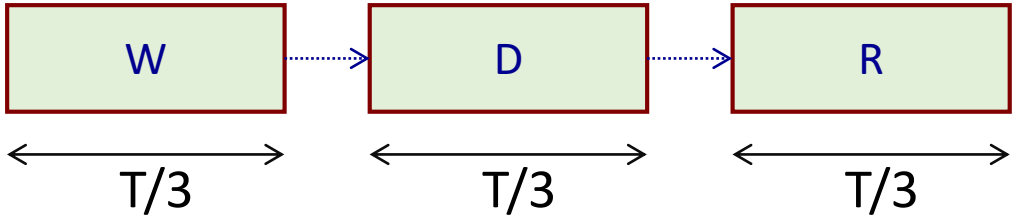
- A mechanism for overlapped execution of several input sets by partitioning some computation into a set of *k* sub-computations (or stages).
  - Very nominal increase in the cost of implementation.
  - Very significant speedup (ideally, *k*).
- Where are pipelining used in a computer system?
  a) *Instruction execution*: Several instructions executed in some sequence.
  b) *Arithmetic computation*: Same operation carried out on several data sets.
  c) *Memory access*: Several memory accesses to consecutive locations are made.

A Real-life Example

- Suppose you have built a machine *M* that can wash (*W*), dry (*D*), and iron (*R*) clothes, one cloth at a time.
  - Total time required is *T*.
- As an alternative, we split the machine into three smaller machines M_W, M_D and M_R, which can perform the specific task only.
  - Time required by each of the smaller machines is *T/3* (say).

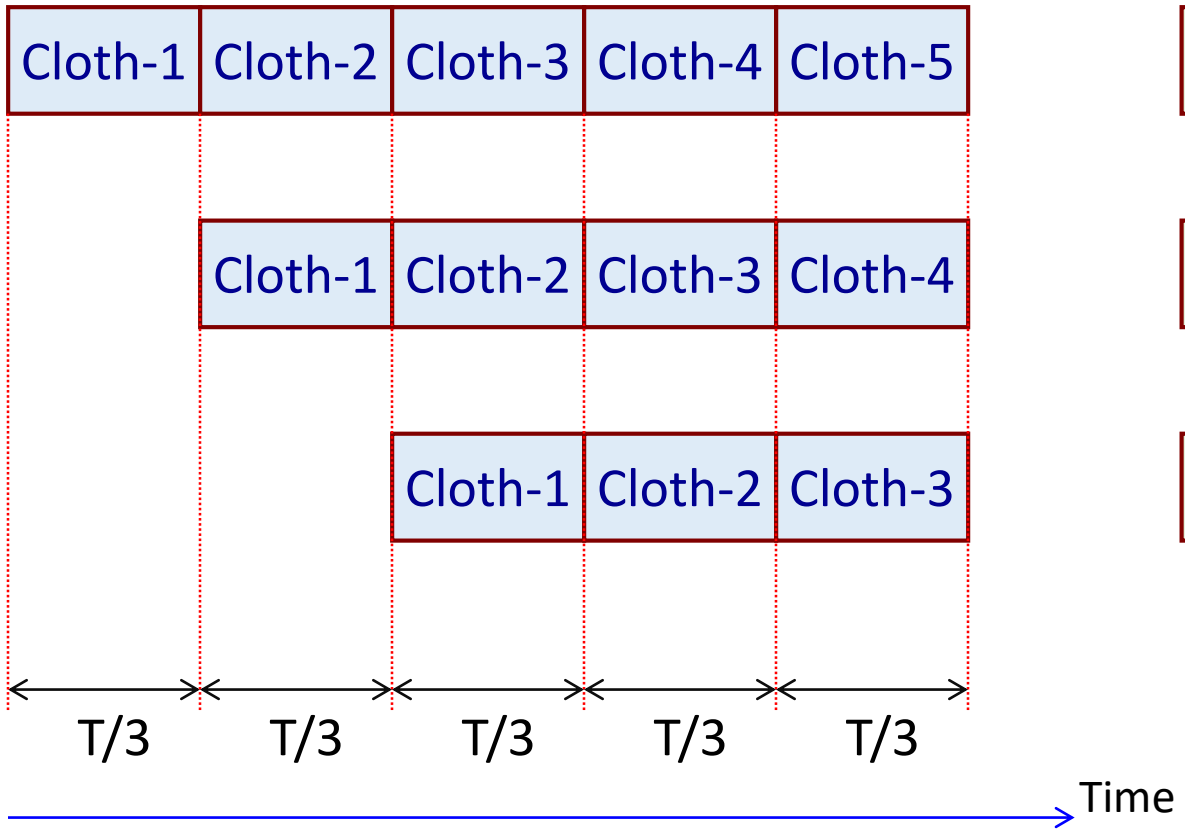


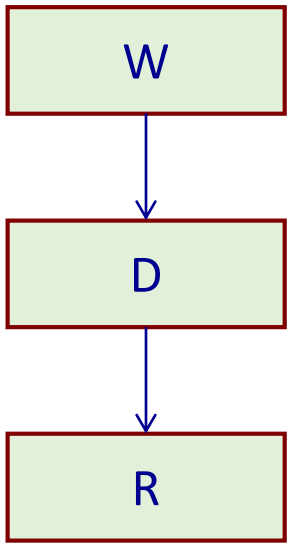
For N clothes, time $T_1 = N.T$



For N clothes, time $T_3 = (2 + N).T/3$

How does the pipeline work?





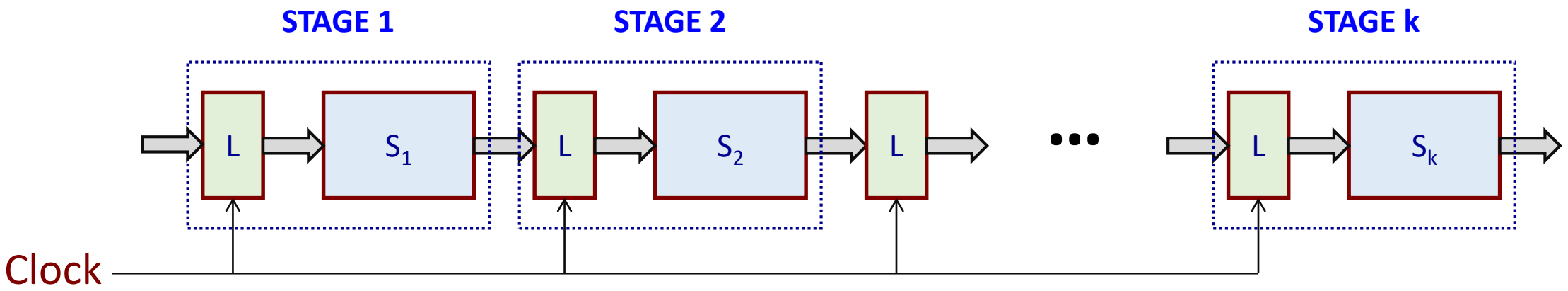
Finishing times:

- Cloth-1 – 3.T/3
- Cloth-2 – 4.T/3
- Cloth-3 – 5.T/3
- …
- Cloth-N – (2 + N).T/3

Extending the Concept to Processor Pipeline

- The same concept can be extended to *hardware pipelines*.
- Suppose we want to attain *k* times speedup for some computation.
  - *Alternative 1*: Replicate the hardware *k* times → cost also goes up *k* times.
  - *Alternative 2*: Split the computation into *k* stages → very nominal cost increase.
- Need for *buffering*:
  - In the washing example, we need a *tray* between machines (W & D, and D & R) to keep the cloth temporarily before it is accepted by the next machine.
  - Similarly in hardware pipeline, we need a *latch* between successive stages to hold the intermediate results temporarily.

Model of a Synchronous k-stage Pipeline



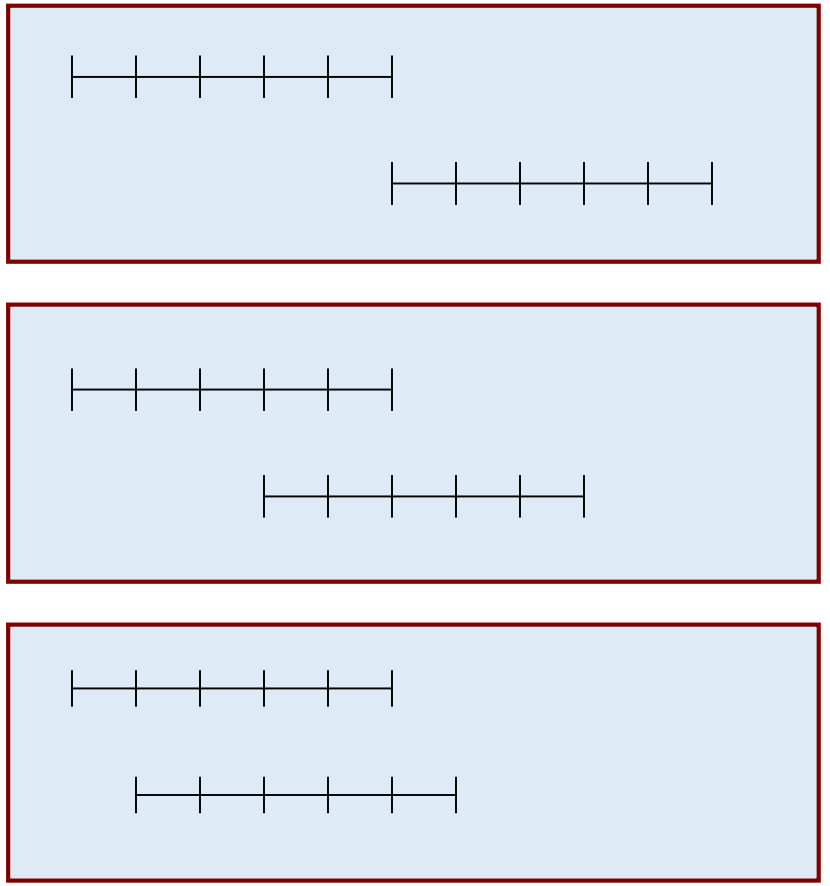
- The latches are made with master-slave flip-flops, and serve the purpose of isolating inputs from outputs.
- The pipeline stages are typically combinational circuits.
- When *Clock* is applied, all latches transfer data to the next stage simultaneously.

Types of Pipelined Processors

- Can be classified based on various parameters:
  a) Degree of overlap
     - Serial, overlapped or pipelined
  b) Depth of the pipeline
     - Shallow or Deep
  c) Structure of the pipeline
     - Linear or Non-linear
  d) How the operations are scheduled in the pipeline?
     - Static or Dynamic

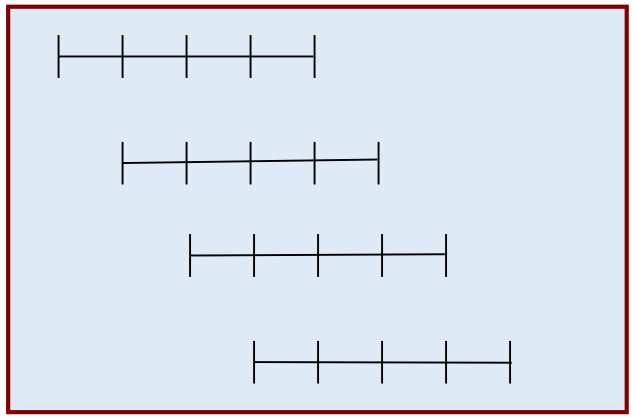
(a) Degree of Overlap

- Serial
  - The next operation can start only after the previous operation finishes.
- Overlapped
  - There is some overlap between successive operations.
- Pipelined
  - Fine-grain overlap between successive operations.

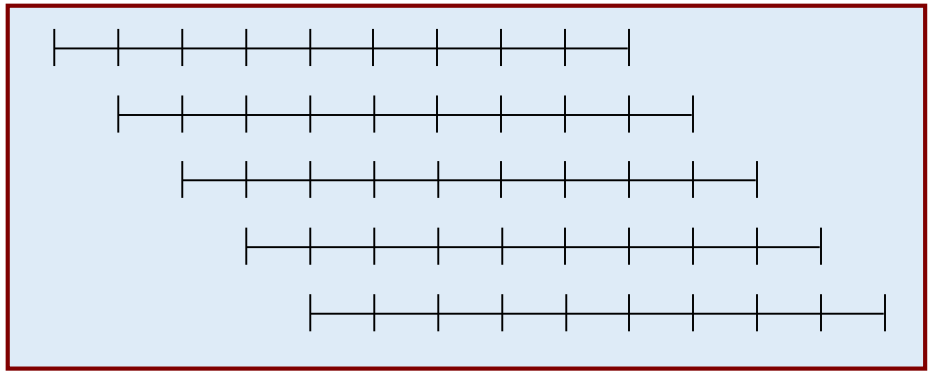


(b) Depth of the Pipeline

- Performance of a pipeline depends on the number of stages and how they can be utilized without conflict.
- Shallow pipeline is one with fewer number of stages.
  - Individual stages more complex.
- Deep pipeline is one with larger number of stages.
  - Individual stages simpler.



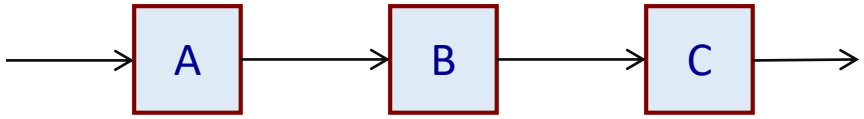
Shallow

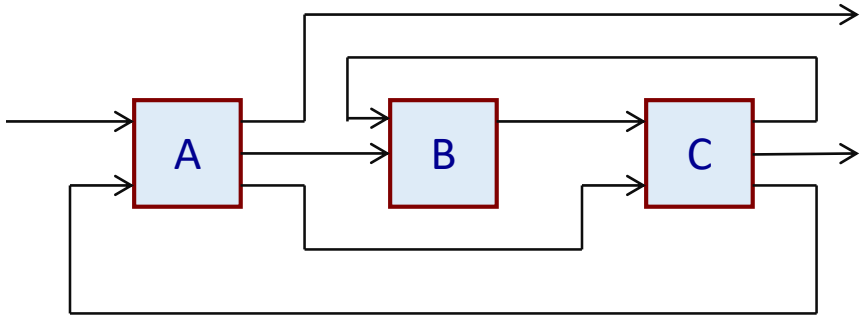


Deep

(c) Structure of the Pipeline

- **Linear Pipeline**: The stages that constitute the pipeline are executed one by one in sequence (say, from left to right).
- **Non-linear Pipeline**: The stages may not execute in a linear sequence (say, a stage may execute more than once for a given data set).





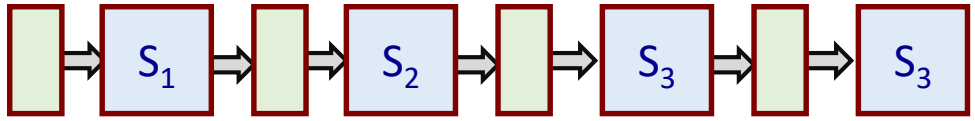
A possible sequence: A, B, C, B, C, A, C, A

(d) Scheduling Alternatives

- **Static Pipeline:**
  - Same sequence of pipeline stages are executed for all data / instructions.
  - If one data / instruction stalls, all subsequent ones also gets delayed.
- **Dynamic Pipeline:**
  - Can be reconfigured to perform variable functions at different times.
  - Allows feedforward and feedback connections between stages.

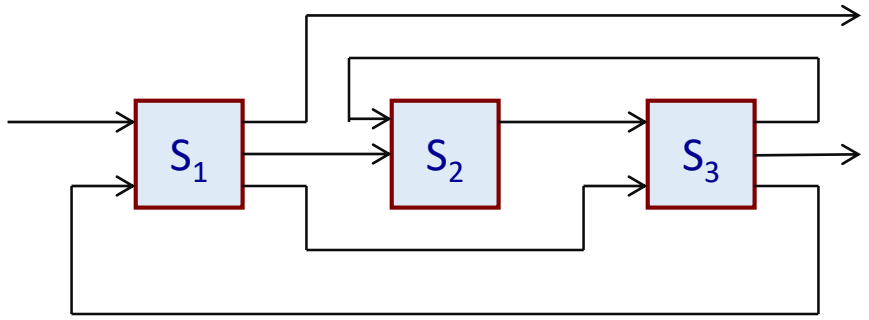
Reservation Table

- The Reservation Table is a data structure that represents the utilization pattern of successive stages in a synchronous pipeline.
  - Basically a *space-time diagram* of the pipeline that shows precedence relationships among the stages.
    - X-axis shows the time steps
    - Y-axis shows the stages
  - Number of columns give evaluation time.
  - The reservation table for a 4-stage linear pipeline is shown.



	1	2	3	4
S_1	X			
S_2		X		
S_3			X	
S_4				X

- Reservation table for a 3-stage dynamic multi-function pipeline is shown.
  - Contains feedforward and feedback connections.
  - Two functions *X* and *Y*.
- Some characteristics:
  - *Multiple X's in a row* :: repeated use of the same stage in different cycles.
  - *Contiguous X's in a row* :: extended use of a stage over more than one cycles.
  - *Multiple X's in a column* :: multiple stages are used in parallel during a clock cycle.



	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

	1	2	3	4	5	6
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y

Speedup and Efficiency

Some notations:

τ :: clock period of the pipeline

t_i :: time delay of the circuitry in stage S_i

d_L :: delay of a latch

Maximum stage delay $\tau_m = max\ \{t_i\}$

Thus, $\tau = \tau_m + d_L$

Pipeline frequency $f = 1 / \tau$

If one result is expected to come out of the pipeline every clock cycle, f will represent the maximum throughput of the pipeline.

- The total time to process *N* data sets is given by

$$T_k = [(k - 1) + N].\tau$$

(k – 1) τ time required to fill the pipeline

1 result every τ time after that → total N.τ

- For an equivalent non-pipelined processor (i.e. one stage), the total time is

$$T_1 = N.k.\tau$$ (ignoring the latch overheads)

- Speedup of the *k*-stage pipeline over the equivalent non-pipelined processor:

$$S_k = \frac{T_1}{T_k} = \frac{N.k.\tau}{k.\tau + (N - 1).\tau} = \frac{N.k}{k + (N - 1)}$$

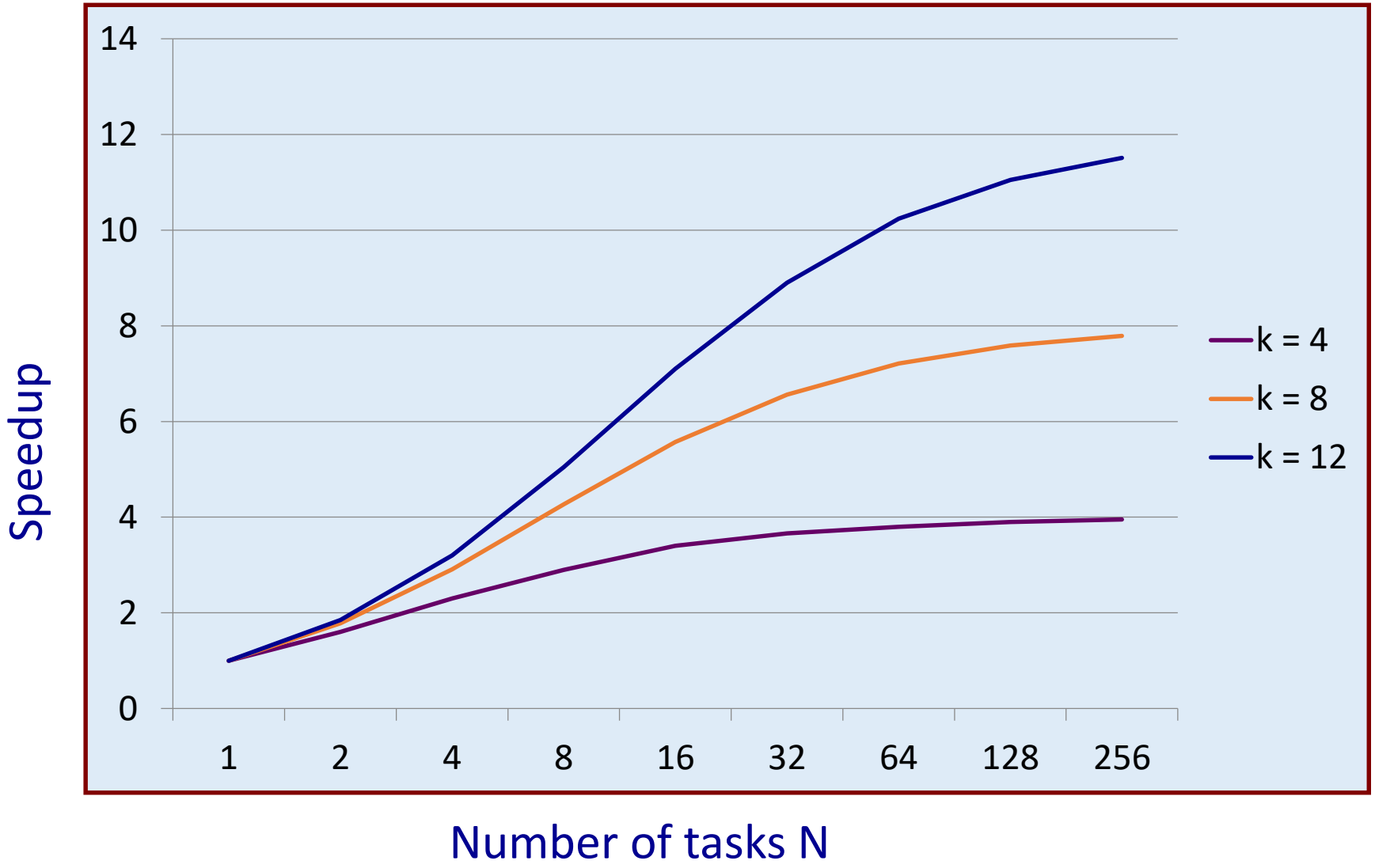
As $N \rightarrow \infty$, $S_k \rightarrow k$

- **Pipeline efficiency**:
  - How close is the performance to its ideal value?

$$E_k = \frac{S_k}{k} = \frac{N}{k + (N - 1)}$$

- **Pipeline throughput:**
  - Number of operations completed per unit time.

$$H_k = \frac{N}{T_k} = \frac{N}{[k + (N - 1)].\tau}$$


Speedup
14
12
10
8
6
4
2
0
1
2
4
8
16
32
64
128
256
k = 4
k = 8
k = 12
Number of tasks N

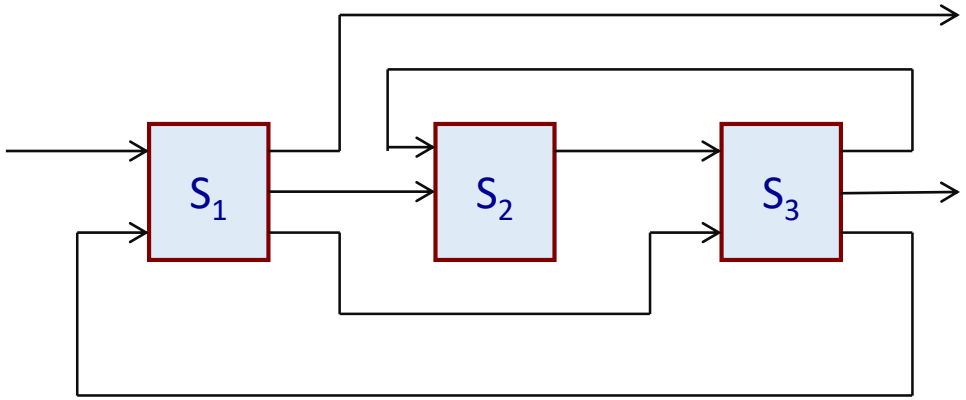
Clock Skew / Jitter / Setup time

- The minimum clock period of the pipeline must satisfy the inequality:

$$\tau \geq t_{skew+jitter} + t_{logic+setup}$$

- Definitions:
  - *Skew*: Maximum delay difference between the arrival of clock signals at the stage latches.
  - *Jitter*: Maximum delay difference between the arrival of clock signal at the same latch.
  - *Logic delay*: Maximum delay of the slowest stage in the pipeline.
  - *Setup time*: Minimum time a signal needs to be stable at the input of a latch before it can be captured.

Scheduling of Non-linear Pipelines



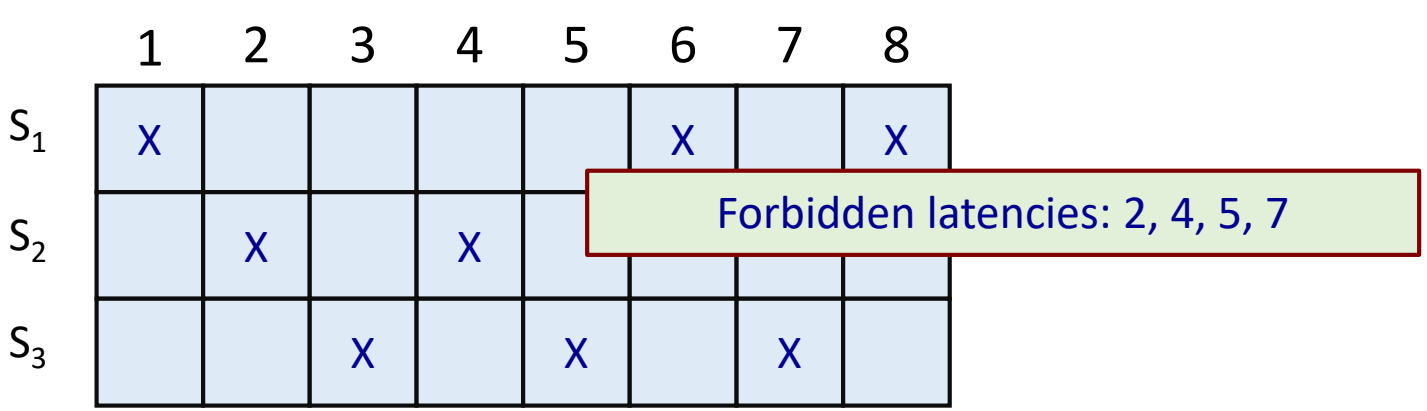
	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

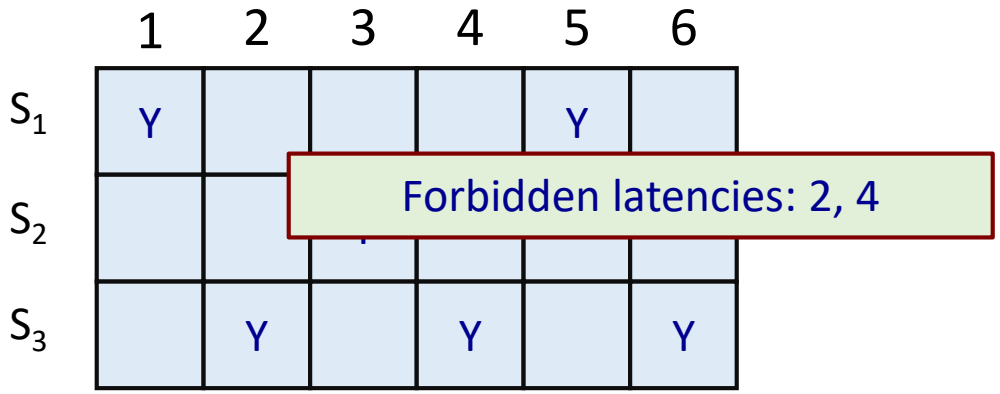
Two operations X and Y

- **X: 8 time steps to complete**
- **Y: 6 time steps to complete**

	1	2	3	4	5	6
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y

- **Latency Analysis**:
  - The number of time units between two initiations of a pipeline is called the *latency* between them.
  - Any attempt by two or more initiations to use the same pipeline stage at the same time will cause a *collision*.
  - The latencies that can cause collision are called *forbidden latencies*.
    - Distance between two X's in the same row of the reservation table.





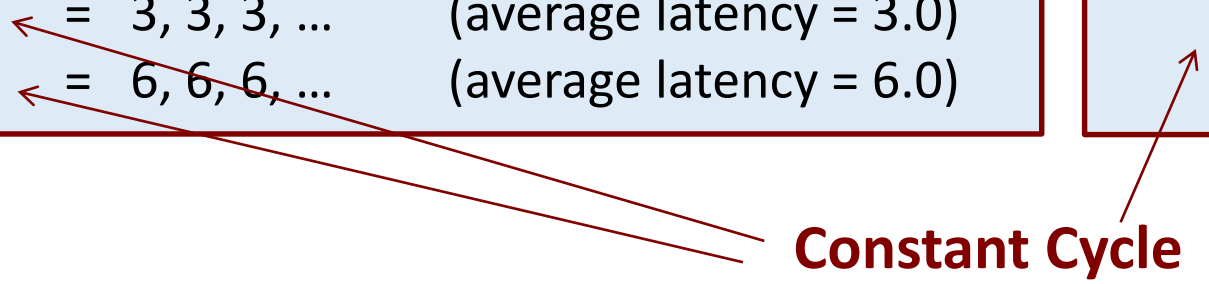
- A *latency sequence* is a sequence of permissible non-forbidden latencies between successive task initiations.
- A *latency cycle* is a latency sequence that repeats the same subsequence.

Function X

- Forbidden latencies: 2, 4, 5, 7
- Possible latency cycles:
  (1, 8) = 1, 8, 1, 8, … (average latency = 4.5)
  (3) = 3, 3, 3, … (average latency = 3.0)
  (6) = 6, 6, 6, … (average latency = 6.0)

Function Y

- Forbidden latencies: 2, 4
- Possible latency cycles:
  (1, 5) = 1, 5, 1, 5, … (average latency = 3.0)
  (3) = 3, 3, 3, … (average latency = 3.0)
  (3, 5) = 3, 5, 3, 5, … (average latency = 4.0)

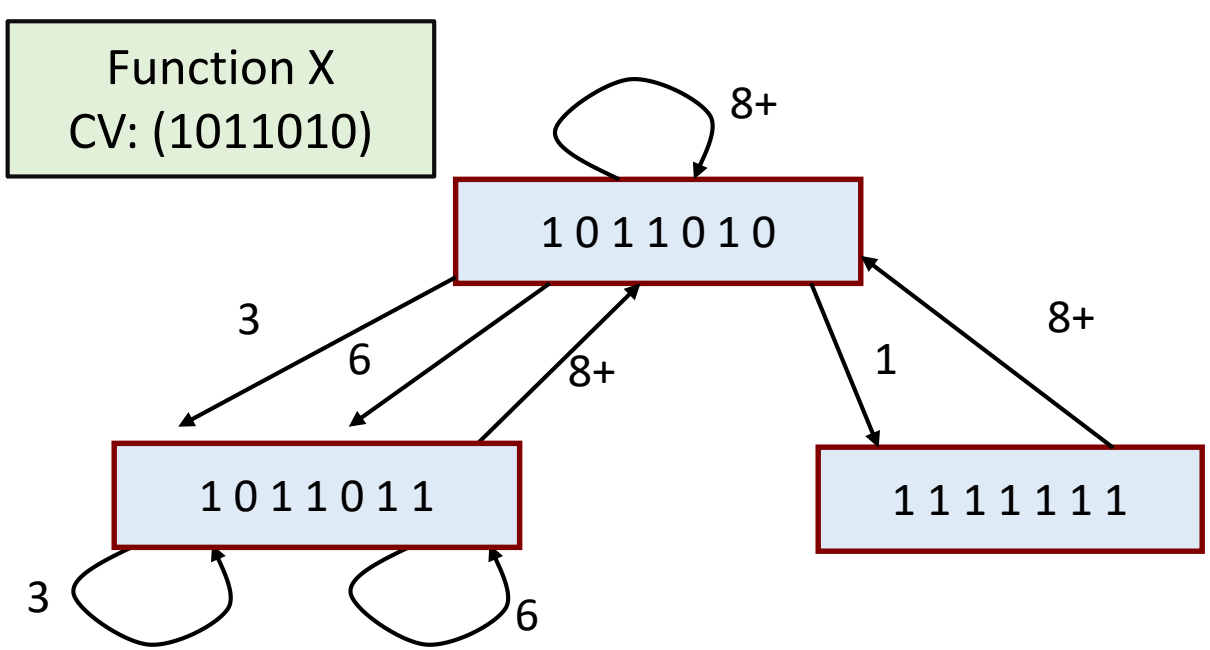


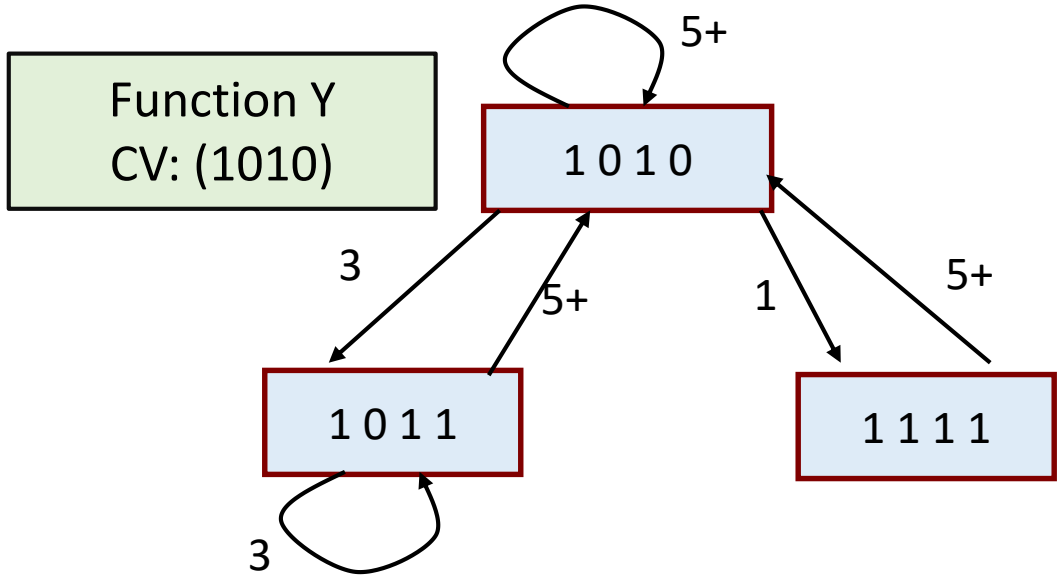
Collision Free Scheduling

- Main objective:
  - Obtain the shortest average latency between initiations without causing collisions.
- We define a *collision vector*.
  - If the reservation table has n columns, the maximum forbidden latency is $m \leq n-1$.
  - The permissible latencies p will satisfy: $1 \leq p \leq m-1$.
  - The collision vector is an m-bit binary vector $C = (C_m\ C_{m-1} \ldots C_2\ C_1)$, where $C_i = 1$ if latency i causes collision, and $C_i = 0$ otherwise.
  - C_m is always 1.

Function X: $C_X = (1011010)$
Function Y: $C_Y = (1010)$

- From the collision vector (CV), we can construct a *state diagram* specifying the permissible state transitions among successive initiations.
  - The collision vector corresponds to the initial state of the pipeline.
  - The next state at time *t+p* is obtained by shifting the present state *p*-bits to the right and OR-ing with the initial collision vector *C*.





- From the state diagram, we can determine latency cycles that result in *minimum average latency (MAL)*.
  - In a *simple cycle*, a state appears only once.
  - Some of the simple cycles are *greedy cycles*, which are formed only using outgoing edges with minimum latencies.

Function X:

- Simple cycles: (3), (6), (8), (1,8), (3,8), (6,8)
- Greedy cycles: (3), (1,8)

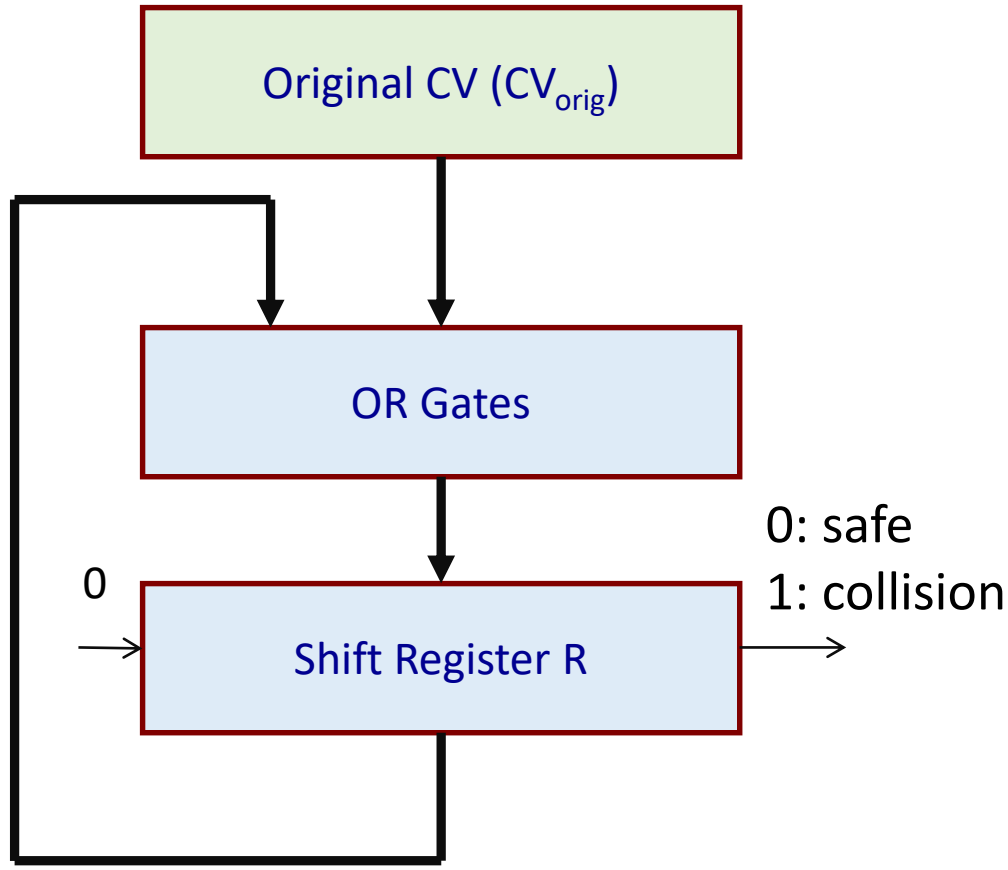
Function Y:

- Simple cycles: (3), (5), (1,5), (3,5)
- Greedy cycles: (3), (1,5)

MAL = 3 for both X and Y

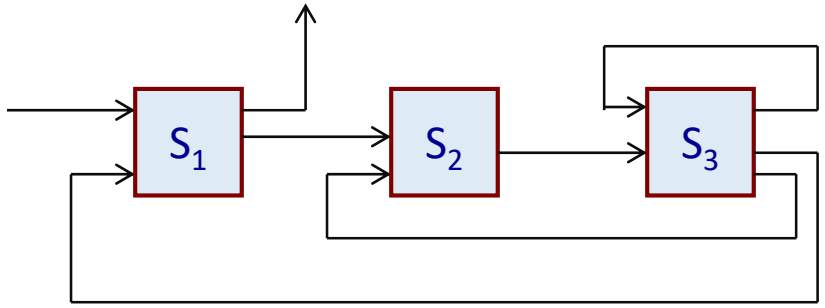
The Scheduling Algorithm

```
Load collision vector (CV) in a shift register R;
If (LSB of R is 1) then
    begin
        Do not initiate an operation;
        Shift R right by one position with 0 insertion;
    end
else
    begin
        Initiate an operation;
        Shift R right by one position with 0 insertion;
        R = R OR CV_orig;       // Logical OR with original CV
    end
```

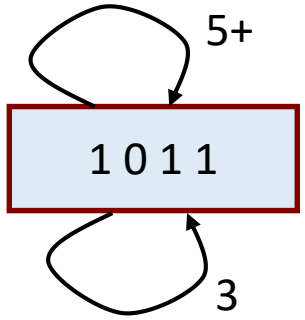


Optimizing a Pipeline Schedule

- We can insert non-compute (dummy) delay stages into the original pipeline.
  - This will modify the reservation table, resulting in a new collision vector.
  - Possibly a shorter MAL.

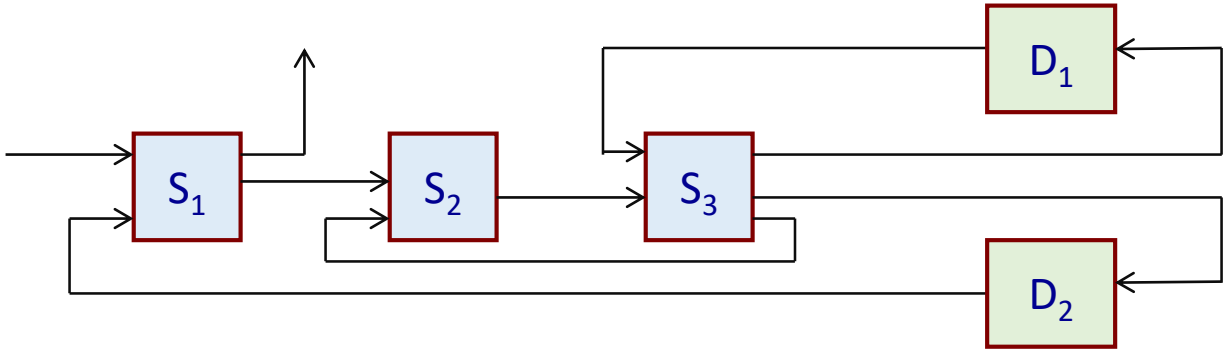


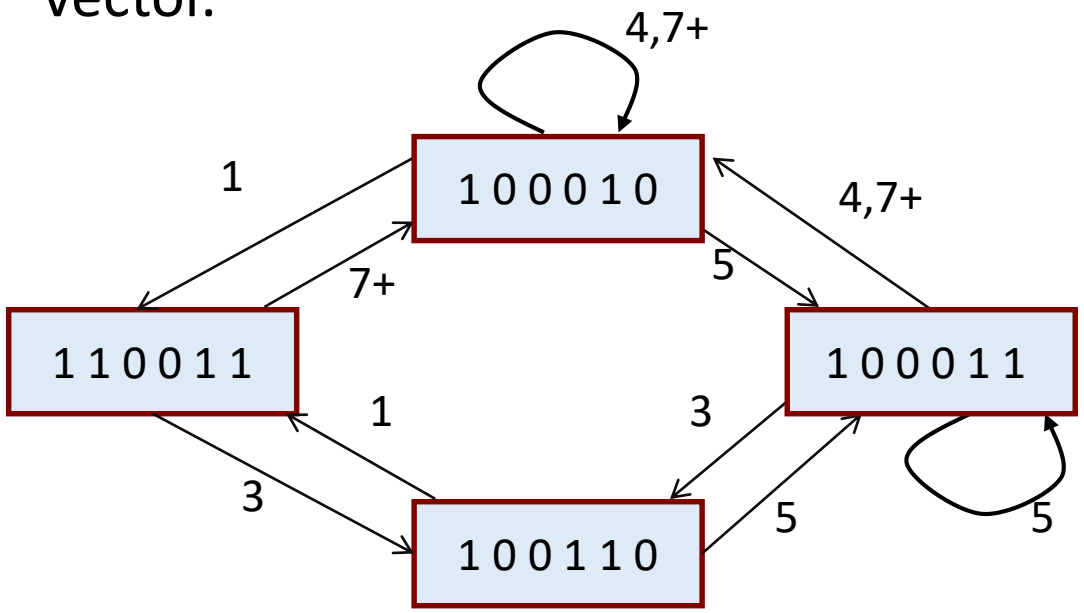
	1	2	3	4	5
S_1	X				X
S_2		X		X	
S_3			X	X	



MAL = 3

- Suppose we insert delay elements D_1 and D_2 as shown.
  - This will modify the reservation table, resulting in a new collision vector.





MAL = 2 (for the greedy cycle (1,3))

	1	2	3	4	5	6	7
S_1	X				→	→	X_2
S_2		X		X			
S_3			X	→	X_1		
D_1				X			
D_2						X	

Exercise 1

- For the following reservation tables,
  a) What are the forbidden latencies?
  b) Show the state transition diagram.
  c) List all the simple cycles and greedy cycles.
  d) Determine the optimal constant latency cycle, and the MAL.
  e) Determine the pipeline throughput, for τ = 20 ns.

	1	2	3	4
S_1	X			X
S_2		X		
S_3			X	

	1	2	3	4	5	6	7
S_1	X		X				X
S_2		X			X		
S_3				X		X	

Exercise 2

- A non-pipelined processor X has a clock frequency of 250 MHz and an average CPI of 4. Processor Y, an improved version of X, is designed with a 5-stage linear instruction pipeline. However, due to latch delay and clock skew, the clock rate of Y is only 200 MHz.

  If a program consisting of 5000 instructions are executed on both processors, what will be the speedup of processor Y as compared to processor X?

Exercise 3

- Consider a 5-stage pipeline with stage delays 25ns, 18ns, 15ns, 17ns and 16ns respectively, and the delay of a pipeline latch stage is 5ns. For processing 1500 data items, what will be the total time required (ignoring pipeline stalls)?

  Repeat the calculation for the case where the first stage is split into two simpler stages with stage delays 15ns and 16ns respectively.