

# Pipeline Design

Computer Architecture  
Dr. Guy Tel-Zur

## מטרות המצגת

- ביסוס הבנת מושג ה-Pipeline
- מדדים המשפיעים על הביצועים
- מונחים מרכזיים:
- דרגות ה-Pipeline, חוקיו, עקרונות התכנון
- מתן דוגמה

# This presentation is based on:

- MIT Open course ware "**Computation Structures**" by Dr. Christopher J. Terman.
- References:
  - MIT Open course ware  
MIT6.004 2017
  - Slides 13-15 : Slides by Prof. Shmuel Wimer Technion/BIU

Chris Terman. 6.004 Computation Structures. Spring 2017. Massachusetts Institute of Technology: MIT OpenCourseWare, <https://ocw.mit.edu>. License: Creative Commons BY-NC-SA.

In particular we will discuss material from:

<https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-004-computation-structures-spring-2017/c7/c7s1/#1>

See 7.2.2, 7.2.3, 7.2.4 @ MIT open course ware site

# Forget circuits... Let's Solve a Real Problem

INPUT:  
dirty laundry



OUTPUT:  
6 more weeks



Device: Washer

Function: Fill, Agitate, Spin

Washer<sub>PD</sub> = 30 mins



Device: Dryer

Function: Heat, Spin

Dryer<sub>PD</sub> = 60 mins

PD = Propagation delay, Period

# One Load At a Time

Everyone knows that the real reason that we put off doing laundry so long is not because we procrastinate, are lazy, or even have better things to do.

The fact is, doing one load at a time is not smart.

Step 1:



Step 2:



$$\begin{aligned}\text{Total}_{\text{PD}} &= \text{Washer}_{\text{PD}} + \text{Dryer}_{\text{PD}} \\ &= \underline{\quad 90 \quad} \text{ mins}\end{aligned}$$

# Doing N Loads of Laundry

Here's how they do laundry at Harvard, the “combinational” way.

Of course, this is just an urban legend. No one at Harvard actually *does* laundry. The butlers all arrive on Wednesday morning, pick up the dirty laundry and return it all pressed and starched in time for afternoon tea.



Step 1:



Step 2:



Step 3:



Step 4:



...

$$\begin{aligned} \text{Total}_{PD} &= N * (\text{Washer}_{PD} + \text{Dryer}_{PD}) \\ &= \underline{N*90} \text{ mins} \end{aligned}$$

# Doing N Loads... The 6.004 Way

6.004 students  
“pipeline” the laundry  
process.

That’s why we wait!

Actually, it’s more like  $N \cdot 60 + 30$  if we account for the startup transient correctly. When doing pipeline analysis, we’re mostly interested in the “steady state” where we assume we have an infinite supply of inputs.



$$\text{Total}_{PD} = N * \text{Max}(\text{Washer}_{PD}, \text{Dryer}_{PD})$$

$$= \underline{N * 60} \text{ mins}$$

לפי האיטי מבין השניים

# Performance Measures

Latency:

The delay from when an input is established until the output associated with that input becomes valid.

$$\text{Harvard Laundry} = \frac{90}{120} \text{ mins}$$

$$6.004 \text{ Laundry} = \frac{120}{120} \text{ mins}$$

Assuming that the wash is started as soon as possible and waits (wet) in the washer until dryer is available.

Throughput:

The *rate* at which inputs or outputs are processed.

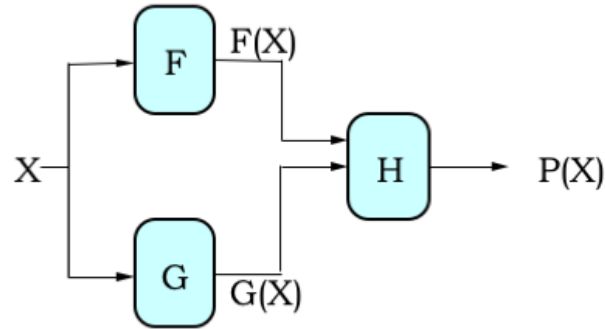
$$\text{Harvard Laundry} = \frac{1/90}{1/60} \text{ outputs/min}$$

$$6.004 \text{ Laundry} = \frac{1/60}{1/60} \text{ outputs/min}$$

פריט יוצא כל  
60 דקות



# Okay, Back To Circuits...

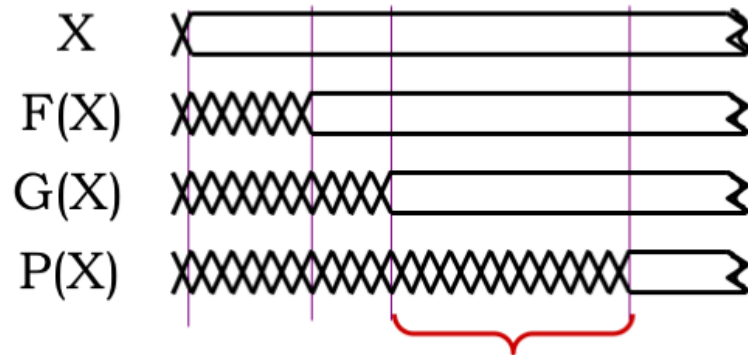


For combinational logic:

latency =  $t_{PD}$ ,

throughput =  $1/t_{PD}$ .

We can't get the answer faster,  
but are we making effective use  
of our hardware at all times?

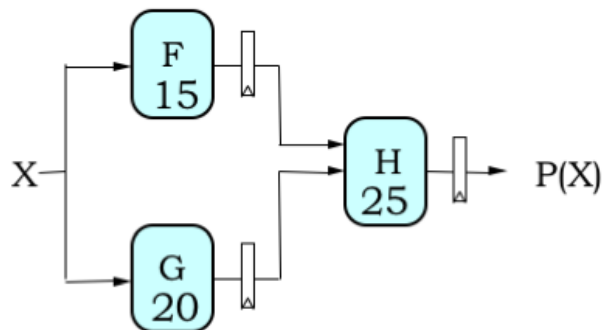


F & G are “idle”, just holding their outputs  
stable while H performs its computation





# Pipelined Circuits

use registers to hold H's input stable!

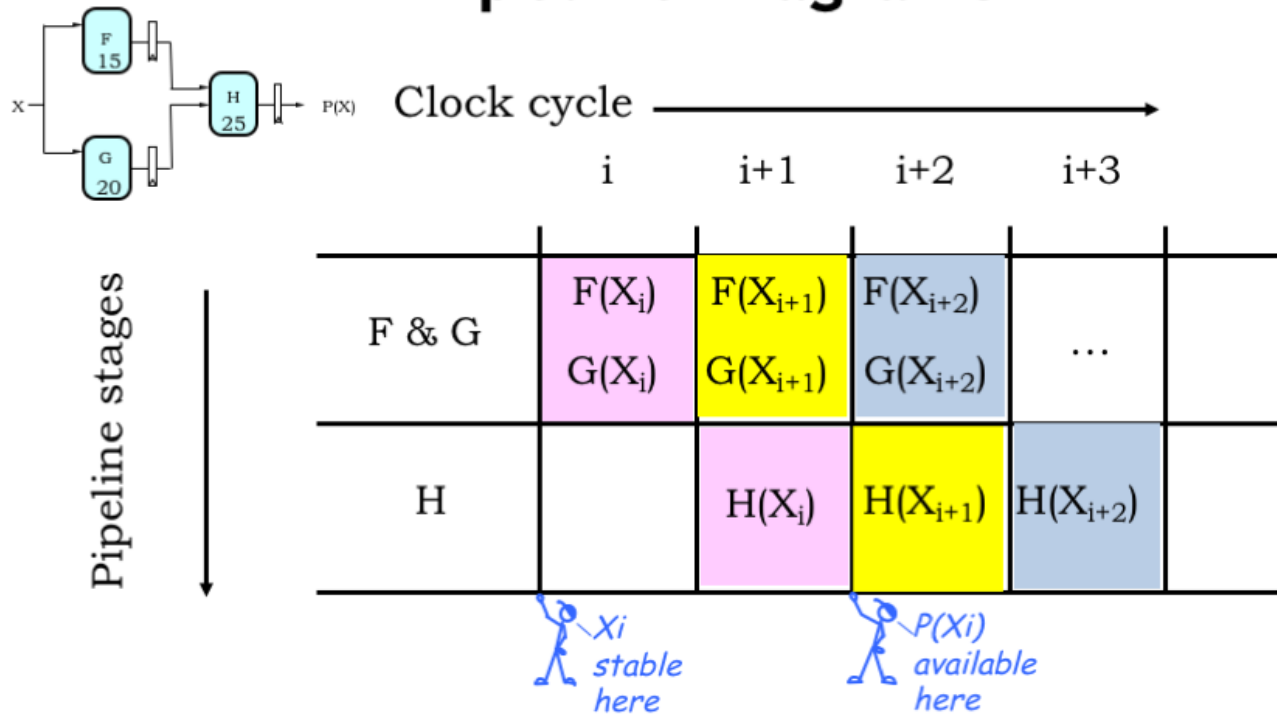


Now F & G can be working on input  $X_{i+1}$  while H is performing its computation on  $X_i$ . We've created a 2-stage *pipeline*: if we have a valid input X during clock cycle j, P(X) is valid during clock j+2.

Suppose F, G, H have propagation delays of 15, 20, 25 ns and we are using ideal zero-delay registers:

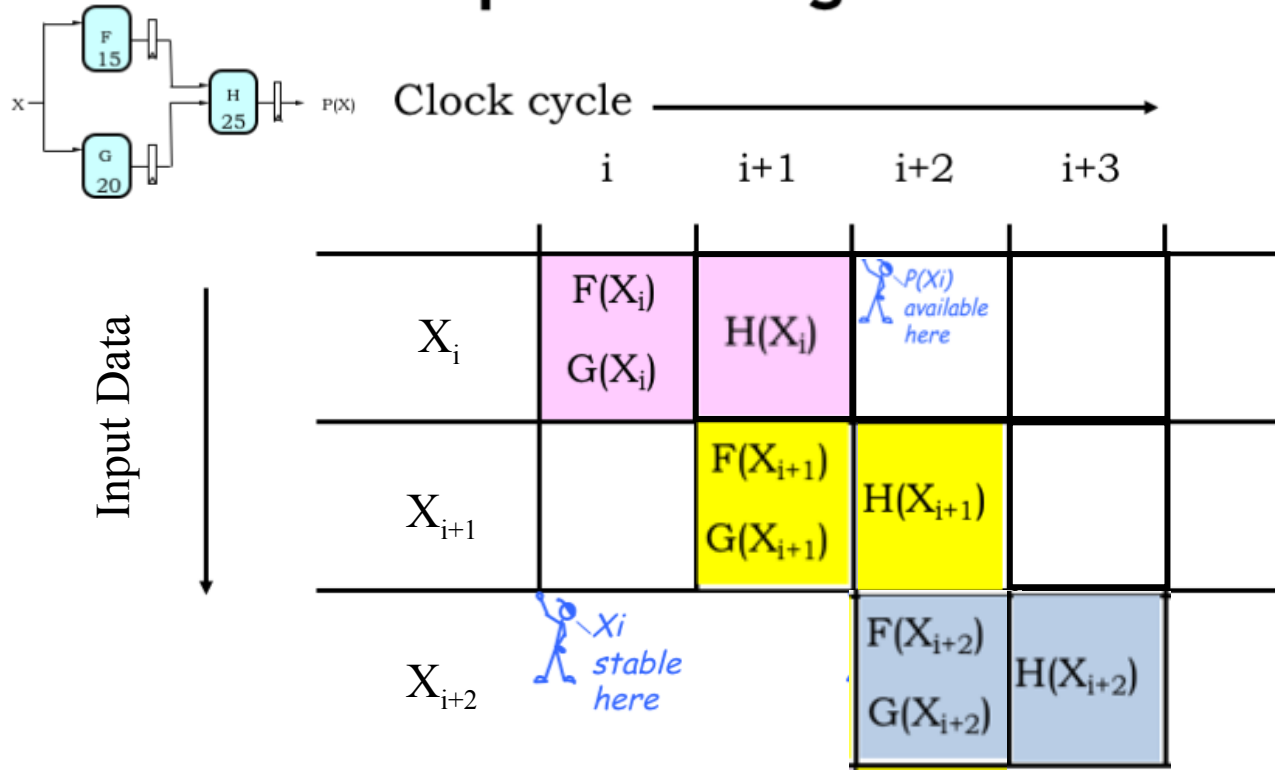
	<u>latency</u>	<u>throughput</u>
unpipelined	45	1/45
2-stage pipeline	<u>50</u> worse 	<u>1/25</u> better 

# Pipeline Diagrams



The results associated with a particular set of input data moves *diagonally* through the diagram, progressing through one pipeline stage each clock cycle.

# Pipeline Diagrams



The results associated with a particular set of input data moves *diagonally* through the diagram, progressing through one pipeline stage each clock cycle.

# Pipeline Conventions

## DEFINITION:

A well-formed *K-Stage Pipeline* (“K-pipeline”) is an acyclic circuit having exactly K registers on *every* path from an input to an output.

a COMBINATIONAL CIRCUIT is thus an 0-stage pipeline.

## COMPOSITION CONVENTION:

Every pipeline stage, hence every K-Stage pipeline, has a register on its *OUTPUT* (not on its input).

## ALWAYS:

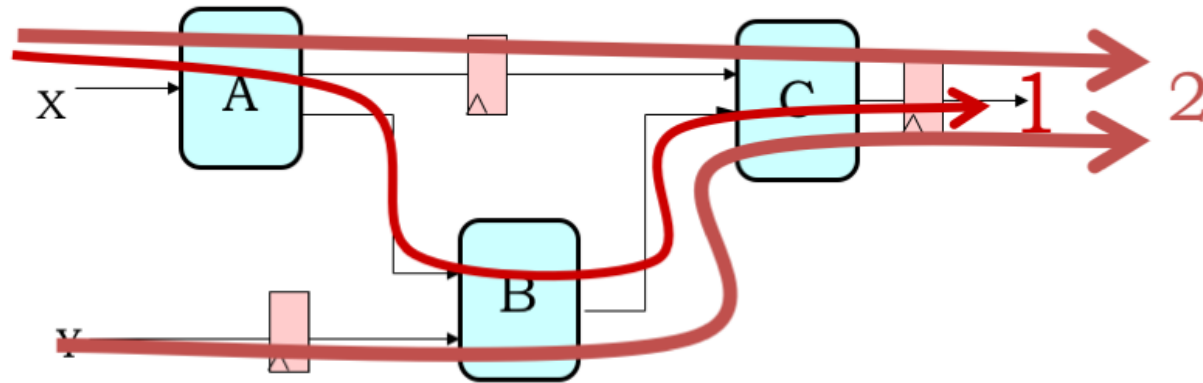
The CLOCK common to all registers must have a period sufficient to cover propagation over combinational paths PLUS (input) register  $t_{PD}$  PLUS (output) register  $t_{SETUP}$ .

The LATENCY of a K-pipeline is K times the period of the system's clock.

The THROUGHPUT of a K-pipeline is the frequency of the clock.

# Ill-formed Pipelines

Consider a BAD job of pipelining:



For what value of K is the following circuit a K-Pipeline?

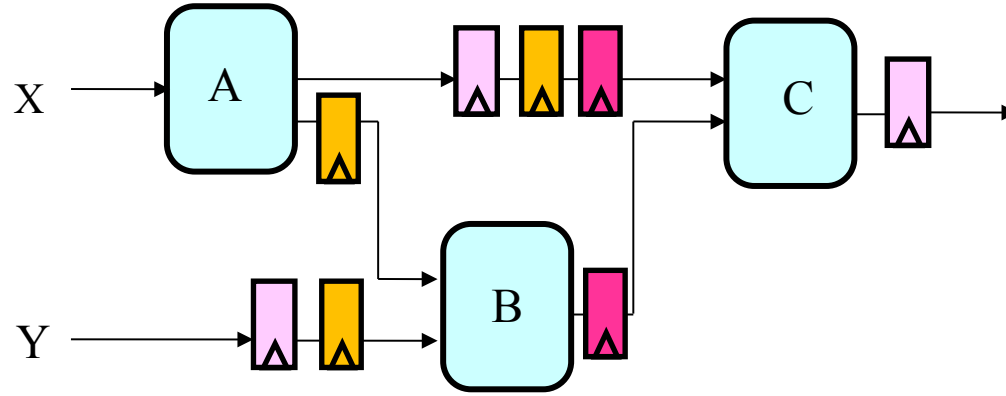
ANS: none

Problem:

*Successive inputs get mixed: e.g.,  $B(A(X_{i+1}), Y_i)$ . This happened because some paths from inputs to outputs have 2 registers, and some have only 1!*

**This CAN'T HAPPEN on a well-formed K pipeline!**

# fixed Pipeline



This is mainly a 4-pipeline with a twist of 3-pipeline to allow mixing  $A(X_{i+1})$  and  $Y_i$  at the input of B.

# A Pipelining Methodology

Step 1:

Draw a line that crosses every output in the circuit, and mark the endpoints as terminal points.

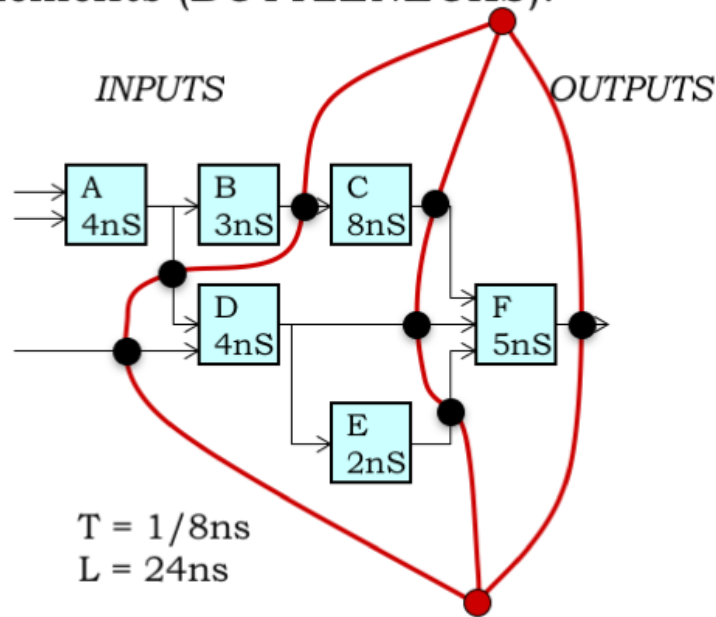
Step 2:

Continue to draw new lines between the terminal points across various circuit connections, ensuring that every connection crosses each line in the same direction. These lines demarcate *pipeline stages*.

Adding a pipeline register at every point where a separating line crosses a connection will always generate a valid pipeline.

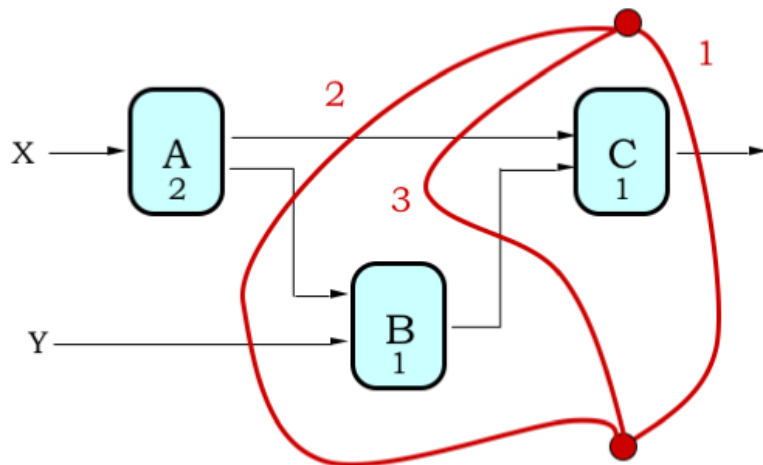
STRATEGY:

Focus your attention on placing pipelining registers around the slowest circuit elements (BOTTLENECKS).





# Pipeline Example



## OBSERVATIONS:

- 1-pipeline improves neither L or T.
- T improved by breaking long combinational paths, allowing faster clock.
- Too many stages cost L, don't improve T.
- Back-to-back registers are often required to keep pipeline well-formed.

	LATENCY	THROUGHPUT
0-pipe:	4	1/4
1-pipe:	4	1/4
2-pipe:	4	1/2
3-pipe:	6	1/2

+ increase throughput  
- increase latency  
- "bottleneck" problem

3 דרגות, כל אחת 2 נ"ש

2 דרגות, כל אחת 2 נ"ש

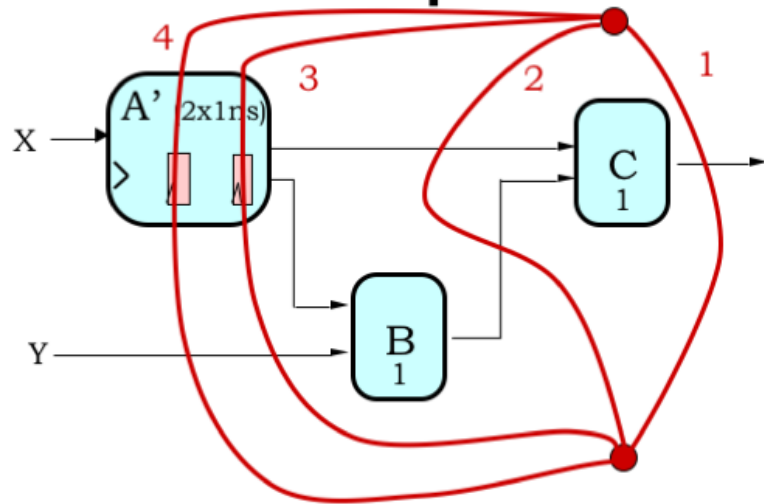
מערכת צרופים בלבד

קו 1

קווים 2+1

קווים 3+2+1

# Pipelined Components



4-stage pipeline, throughput=1

Pipelined systems  
can be hierarchical:

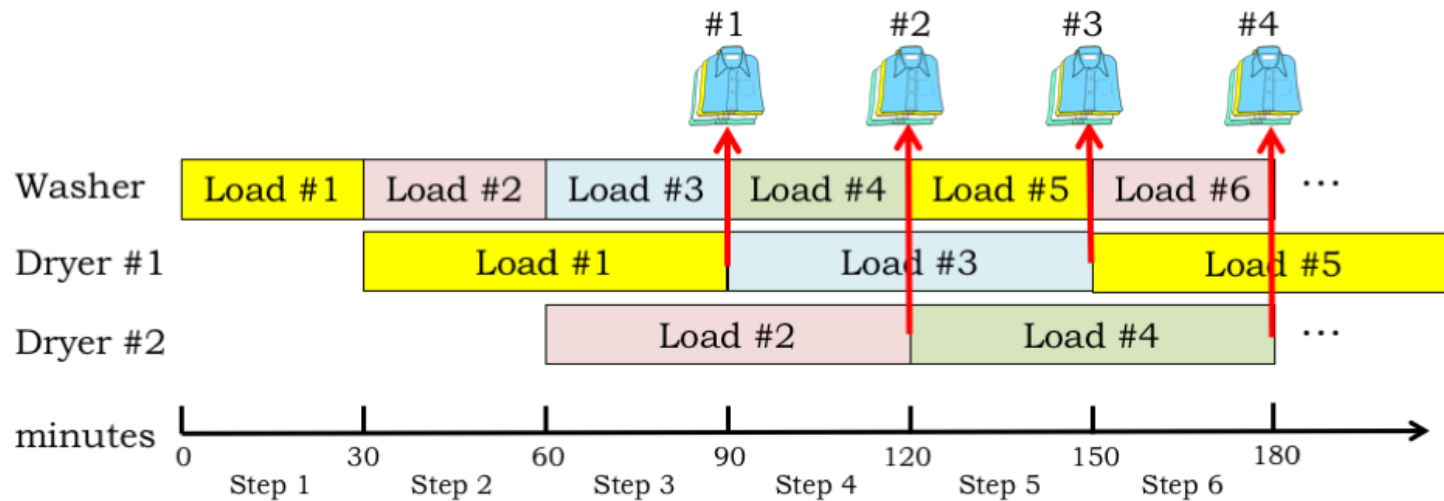
- Replacing a slow combinational component with a k-pipe version may let us decrease the clock period
- Must account for new pipeline stages in our plan



*but... but...  
How can I pipeline  
a clothes dryer???*

# How Do 6.004 Students Do Laundry?

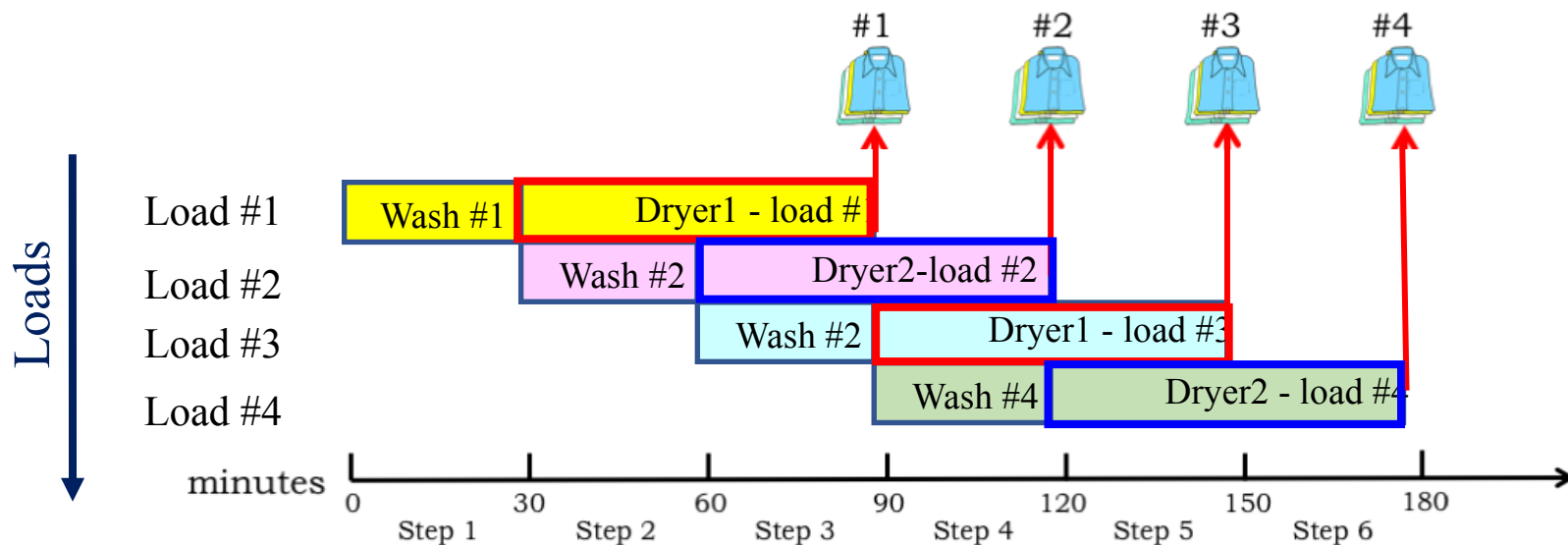
They work around the bottleneck. First, they find a laundromat with two dryers for every washer. Then they use dryer #1 for odd-numbered wash loads and dryer #2 for even-numbered wash loads.



Throughput =  $\frac{1}{30}$  loads/min, Latency = 90 mins/load

# How Do 6.004 Students Do Laundry?

They work around the bottleneck. First, they find a laundromat with two dryers for every washer. Then they use dryer #1 for odd-numbered wash loads and dryer #2 for even-numbered wash loads.



Throughput =  $\frac{1}{30}$  loads/min, Latency = 90 mins/load

# Back To Our Bottleneck...

Recall our earlier example...

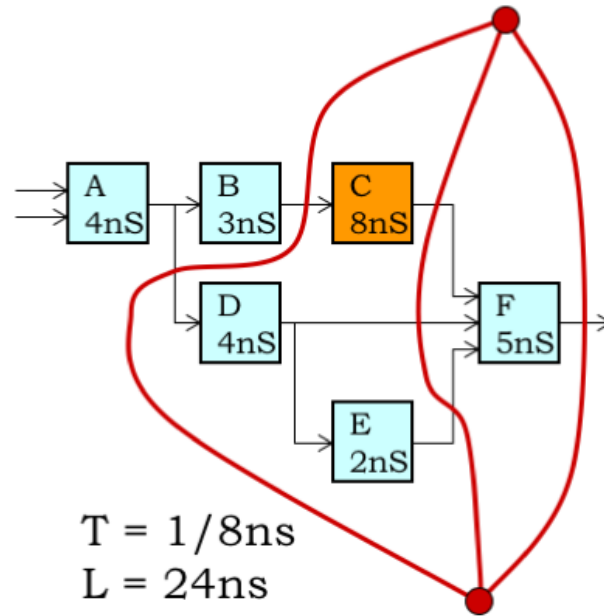
- C – the slowest component – limits clock period to 8 ns.
- HENCE throughput limited to  $1/8\text{ns}$ .

We could improve throughput by

- Finding a pipelined version of C;

OR ...

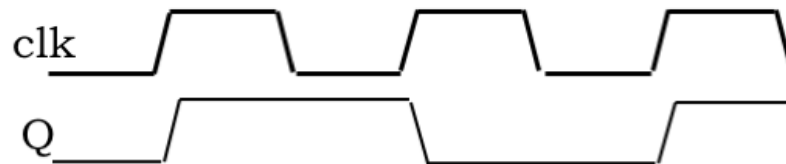
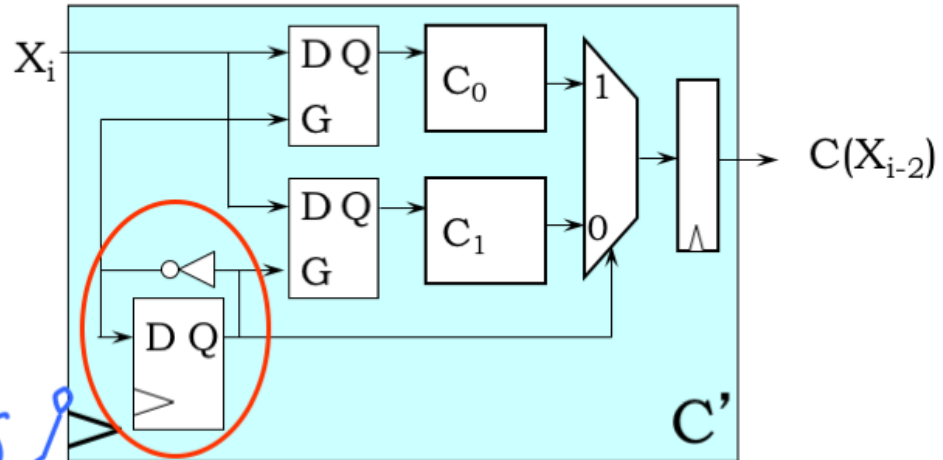
- *interleaving* multiple copies of C!



# Circuit Interleaving

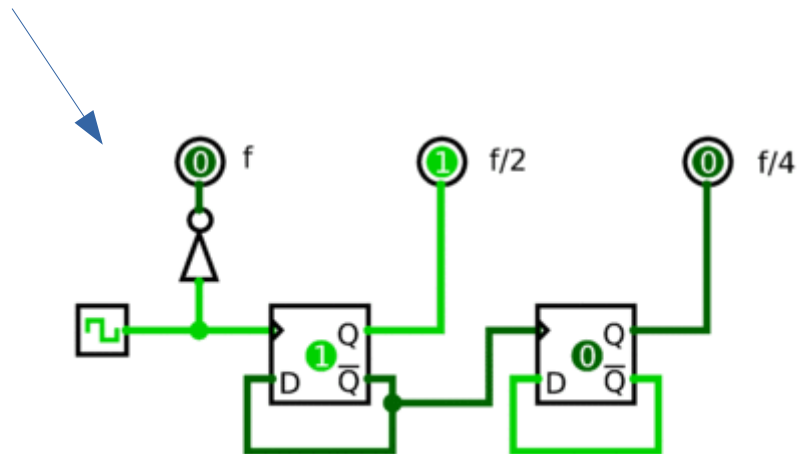
We can simulate a pipelined version of a slow component by replicating the critical element and alternate inputs between the various copies.

*This is a simple 2-state FSM that alternates between 0 and 1 on each clock*



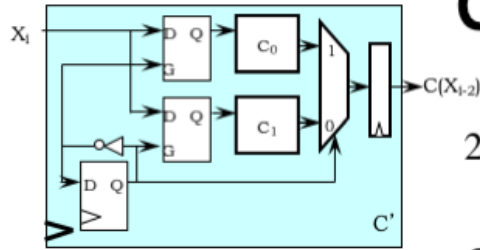
# מחלק תדר

Click for animation

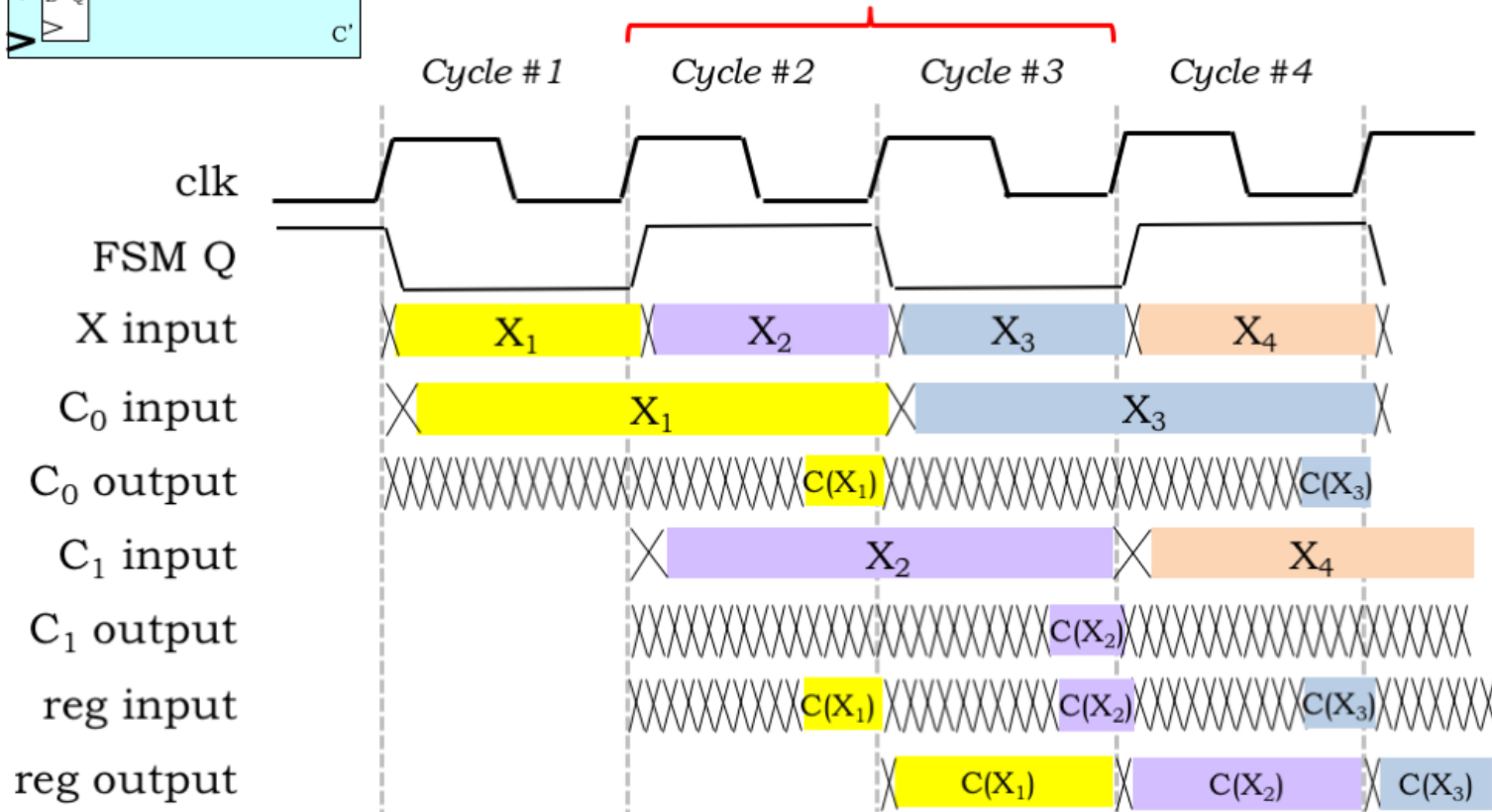


Credit: [https://en.wikipedia.org/wiki/Frequency\\_divider](https://en.wikipedia.org/wiki/Frequency_divider)

# Circuit Interleaving



$$2 \cdot t_{CLK} \geq (t_{PD,upstreamREG} + t_{PD,LATCH} + t_{PC,C} + t_{PD,MUX} + t_{SETUP,REG})$$





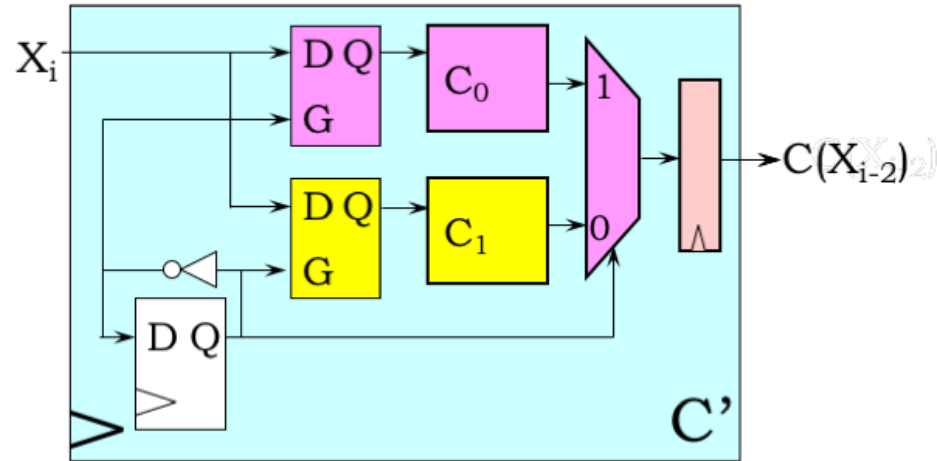
# Circuit Interleaving

## 2-Clock Martinizing

“In by  $t_i$ , out by  $t_{i+2}$ ”

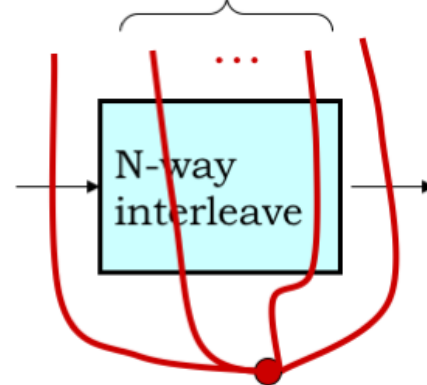
Throughput = 1/clock

Latency = 2 clocks



N registers

N-way interleaving  
is equivalent to  
N pipeline Stages...

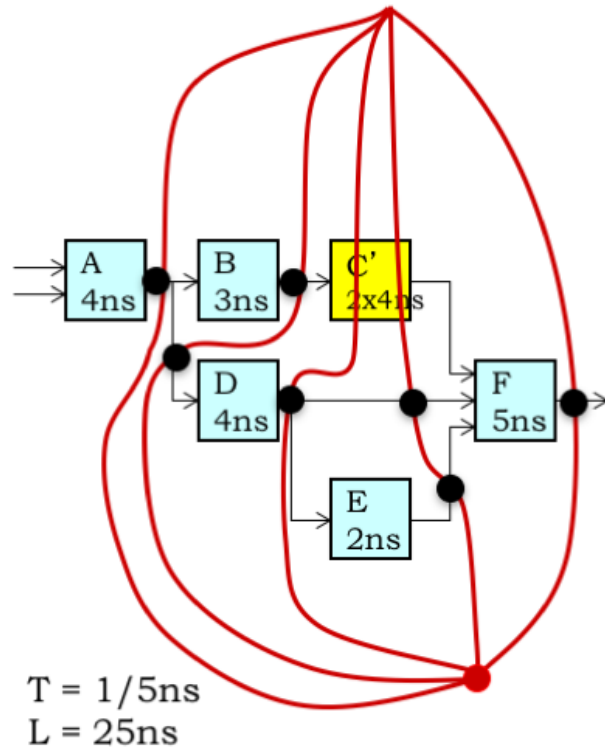


# Combine Techniques

We can combine interleaving and pipelining. Here, C' interleaves two C elements and has an effective tCLK of 4 ns and a latency of 8 ns.

Since C' behaves as a 2-stage pipeline, two of our pipelining contours must pass through the C' component.

By combining interleaving with pipelining we move the bottleneck from the C element to the F element.



# Summary

Latency (L) = time it takes for given input to arrive at output

Throughput (T) = rate at which new outputs appear

For combinational circuits:  $L = t_{PD}$  of circuit,  $T = 1/L$

For K-pipelines ( $K > 0$ ):

- always have register on output(s)
- K registers on every path from input to output
- Inputs available shortly after clock i, outputs available shortly after clock (i+K)
- $t_{CLK} = t_{PD,REG} + t_{PD}$  of slowest pipeline stage +  $t_{SETUP}$
- $T = 1/t_{CLK}$ 
  - more throughput  $\Rightarrow$  split slowest pipeline stage(s)
  - use replication/interleaving if no further splits possible
- $L = K \cdot t_{CLK} = K / T$ 
  - pipelined latency  $\geq$  combinational latency

עד כאן מצגת זו