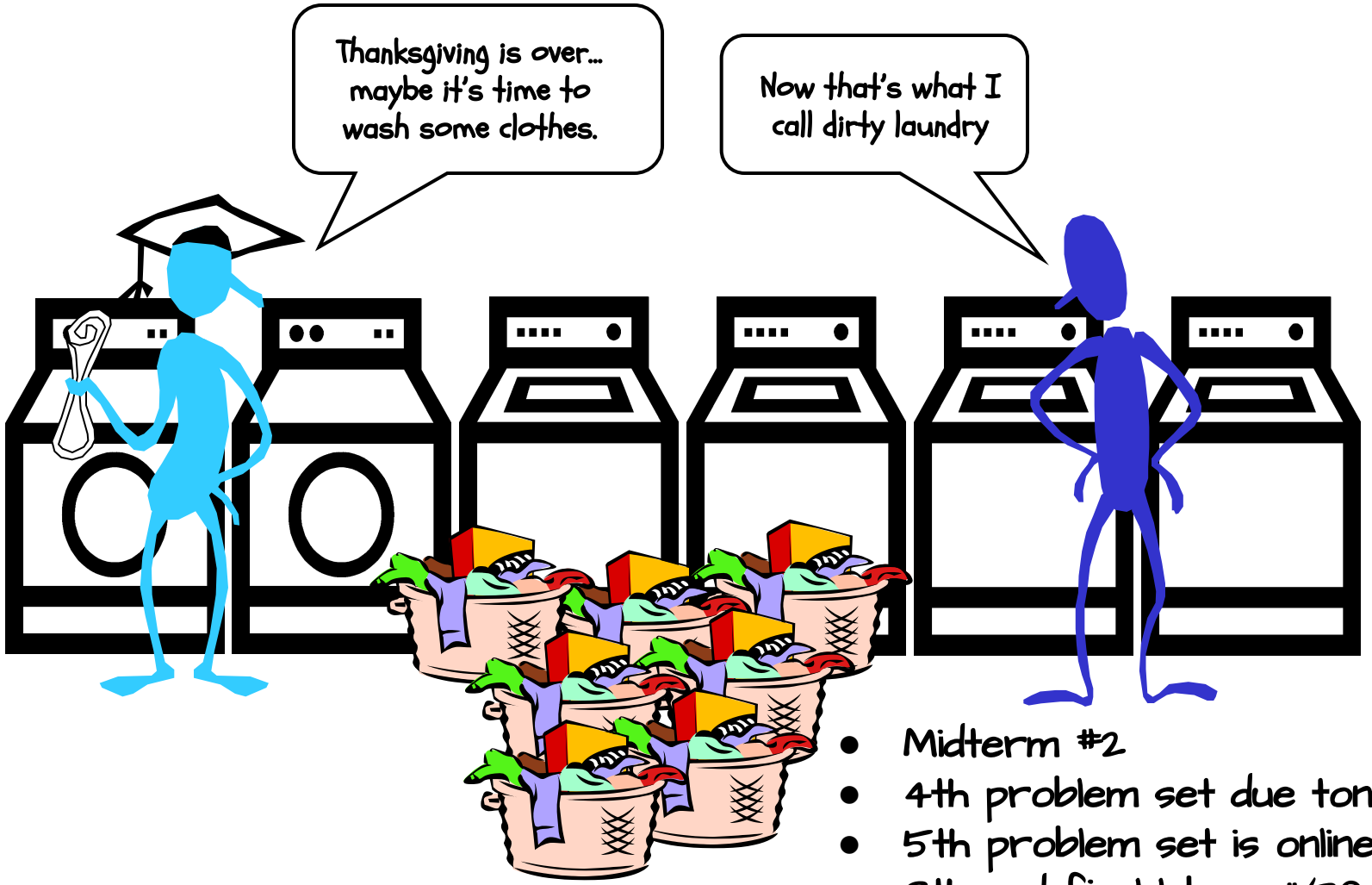
Pipelining



- Midterm #2
- 4th problem set due tonight
- 5th problem set is online
- 8th and final lab on 11/30

The Goal of Pipelining

- Recall our measure of processor performance

Millions of Instructions per Second

Frequency in Hz

$$\text{MIPS} = \frac{1}{10^6} \cdot \frac{\text{clocks / second}}{\text{clocks / instruction}}$$

CPI (Average Clocks Per Instruction)

- How can we turn up the clock rate?

Goal of Pipelining

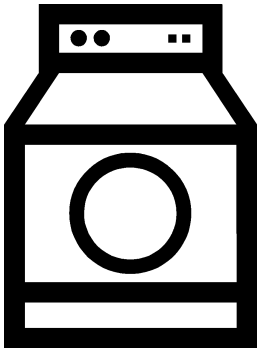
INPUT:
dirty laundry

Device: Washer

Function: Fill, Agitate, Spin

$Washer_{PD}$ = 30 mins

OUTPUT:
4 more weeks



Device: Dryer

Function: Heat, Spin

$Dryer_{PD}$ = 60 mins

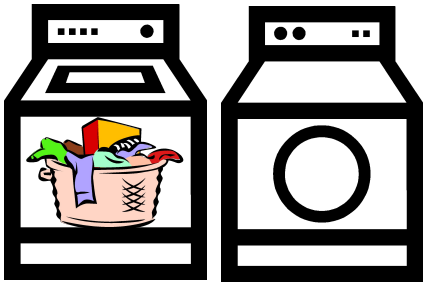
One Load at a Time

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

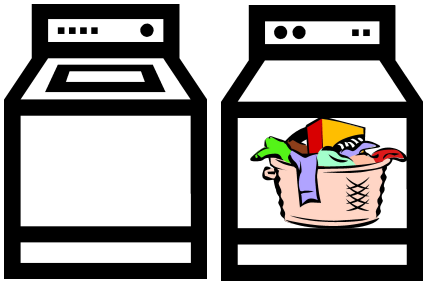
The fact is, doing laundry one load at a time is not smart.

(Sorry Mom, but you were wrong about this one!)

Step 1:



Step 2:



$$\text{Total} = \text{Washer}_{PD} + \text{Dryer}_{PD}$$

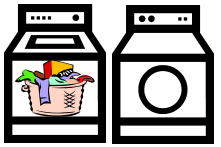
$$= \underline{\quad 90 \quad} \text{ mins}$$

Doing N Loads of Laundry

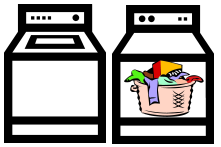
Here's how they do laundry at Duke, the "combinational" way.

(Actually, this is just an urban legend. No one at Duke actually does laundry. The butler's all arrive on Wednesday morning, pick up the dirty laundry and return it all pressed and starched by dinner)

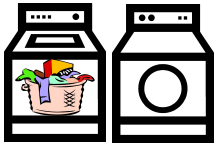
Step 1:



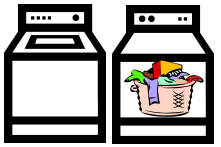
Step 2:



Step 3:



Step 4:



...

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

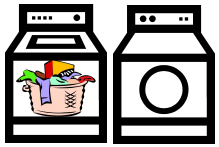
$$= \frac{N*90}{} \text{ mins}$$

Doing N Loads... the UNC way

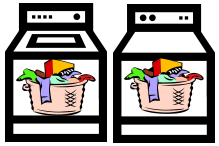
UNC students "pipeline" the laundry process.

That's why we wait!

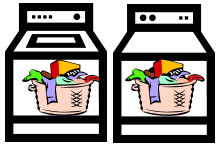
Step 1:



Step 2:



Step 3:



...

Actually, it's more like N*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.

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

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

Recall Our Performance Measures

Latency:

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

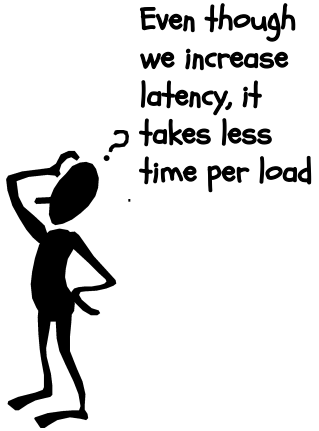
(Duke Laundry = ~~90~~ _____ mins)
(UNC Laundry = ~~120~~ _____ mins)

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

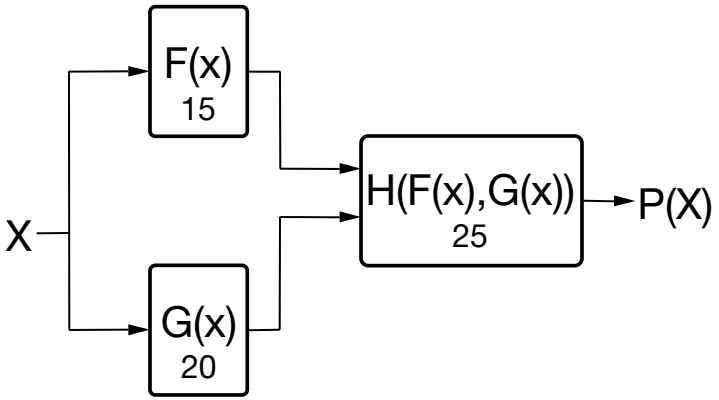
Throughput:

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

(Duke Laundry = $\frac{1}{90}$ outputs/min)
(UNC Laundry = $\frac{1}{60}$ outputs/min)



Okay, Back to Circuits...

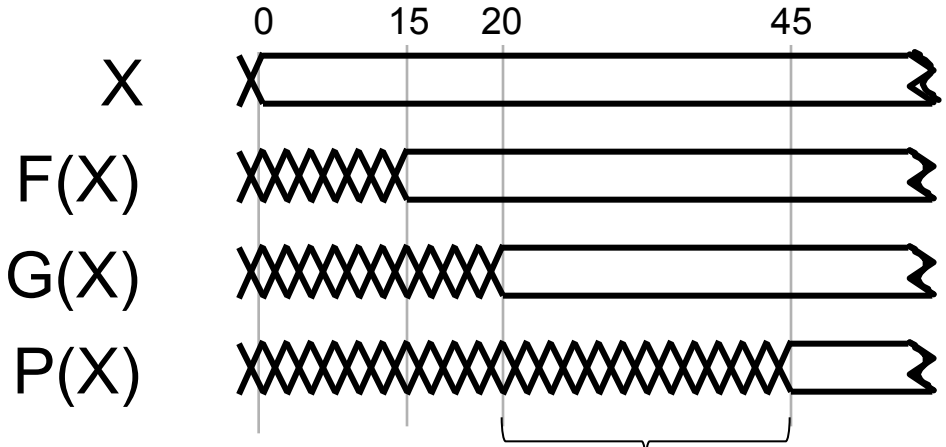


For combinational logic:

latency = t_{PD}

throughput = $1/t_{PD}$

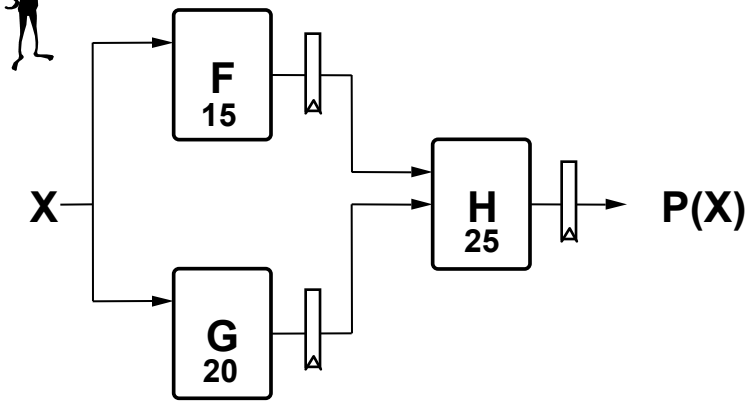
We can't get the answer faster, but are we making effective use of our hardware at all times?



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

Pipelined Circuits

use registers to hold H's input stable!



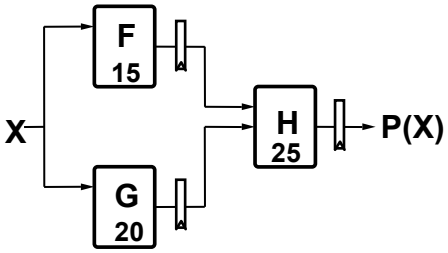
Now F & G can be working on input X_{i+1} while H is performing its computation on X_i. We've created a 2-stage *pipeline*: if we have a valid input X during clock cycle j, P(X) is valid during clock j+2.

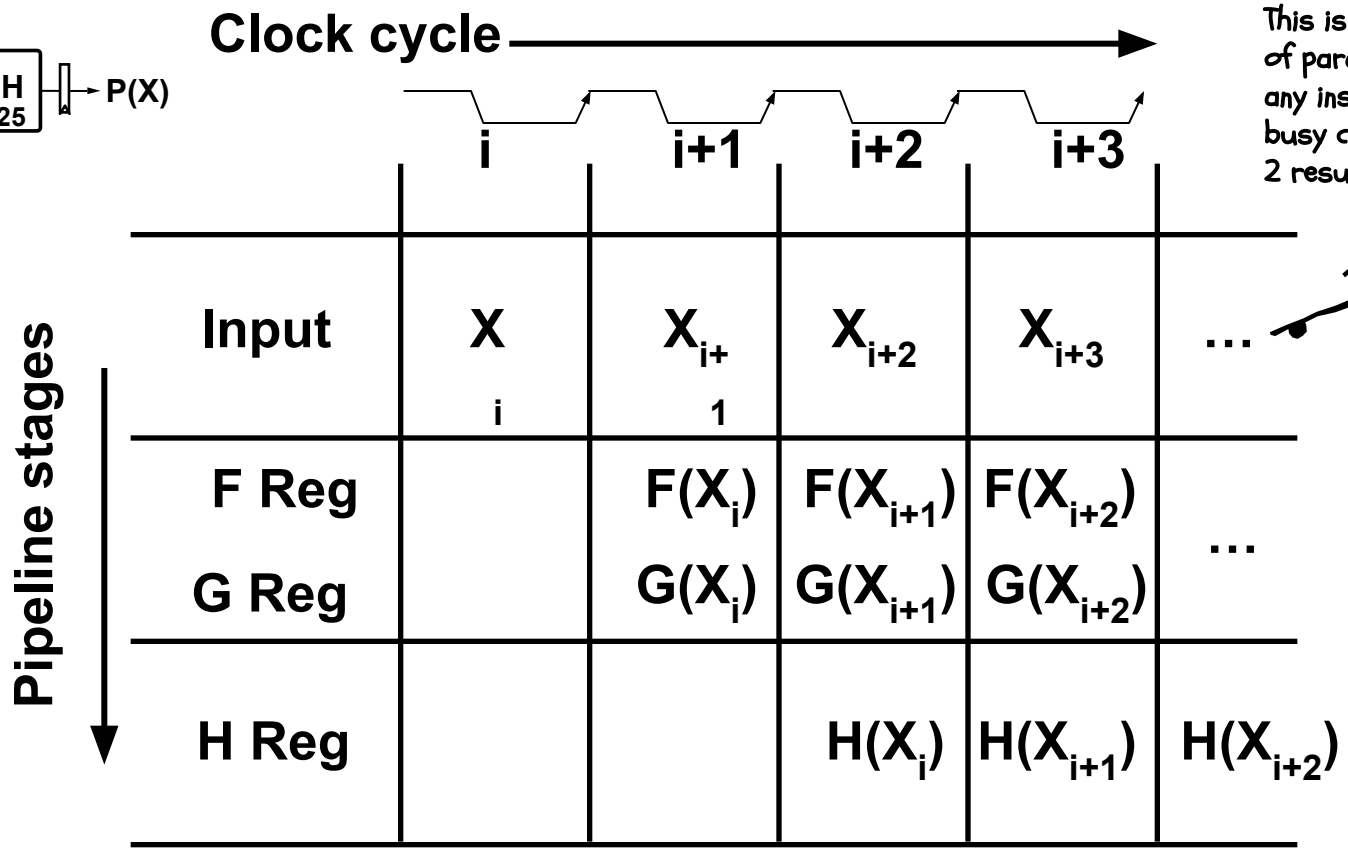
Suppose F, G, H have propagation delays of 15, 20, 25 ns and we are using **ideal zero-delay registers** ($t_s = 0$, $t_{pd} = 0$):

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

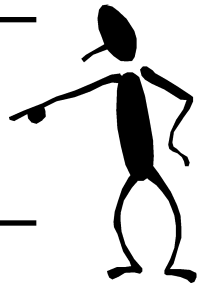
Pipelining uses registers to improve the throughput of combinational circuits

Pipeline Diagrams





This is an example of parallelism. At any instant we are busy computing 2 results.



A pipeline diagram is just a depiction of what inputs are being processed during a given clock period. The results associated with a particular set of input data move *diagonally* through the diagram, progressing through one pipeline stage on each clock cycle.

Pipeline Conventions

DEFINITION:

A *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 a 0-stage pipeline.

CONVENTION:

Every pipeline stage, hence every K-Stage pipeline, has a register on its *OUTPUTS* (as opposed to, alternatively, its inputs).

ALWAYS:

The CLOCK common to all registers *must* have a period sufficient to allow for the propagation delays of all combinational paths PLUS (input) register's t_{PD} PLUS (output) register's t_{SETUP}.

The LATENCY of a K-pipeline is K times the period of the clock common to all registers.

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

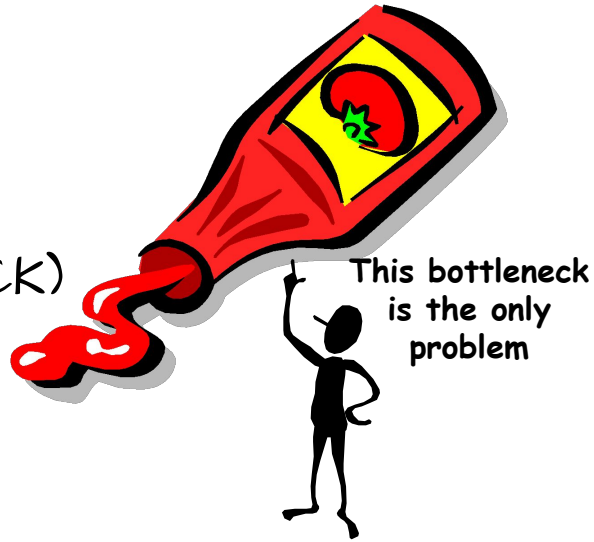
Pipelining Summary

Advantages:

- Higher throughput than combinational system
- Different parts of the logic work on different parts of the problem…

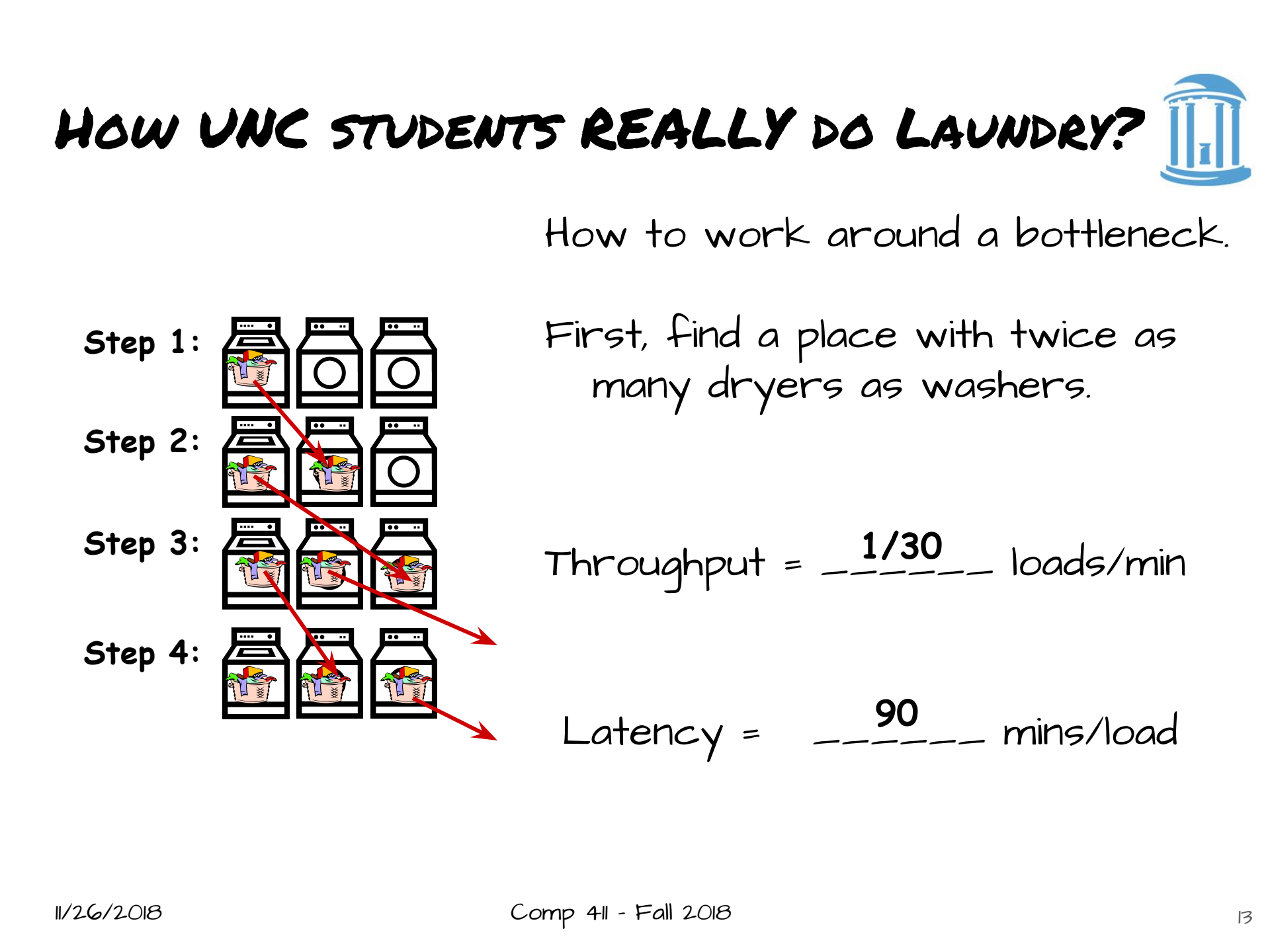
Disadvantages:

- Generally, increases latency
- Only as good as the *weakest* link (often called the pipeline's BOTTLENECK)



Isn't there a way around this "weak link" problem?

How UNC students REALLY do Laundry?



How to work around a bottleneck.

First, find a place with twice as many dryers as washers.

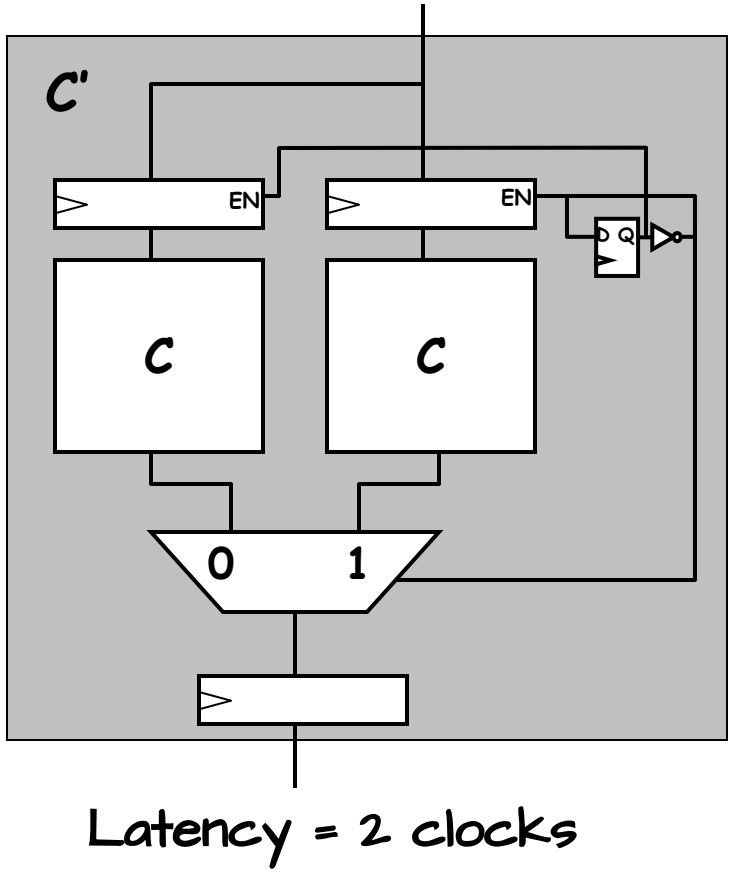
Throughput = 1/30 loads/min

Latency = 90 mins/load

This is called "Interleaving"

One way to overcome a pipeline bottleneck is to **replicate the critical element** as many times as needed and *alternate* inputs between the various copies.

N-way interleaving is equivalent to how many pipeline stages? _N_



Latency = 2 clocks

Better Yet... Parallelism

We can combine interleaving and pipelining with parallelism.

Throughput = 1/15 load/min

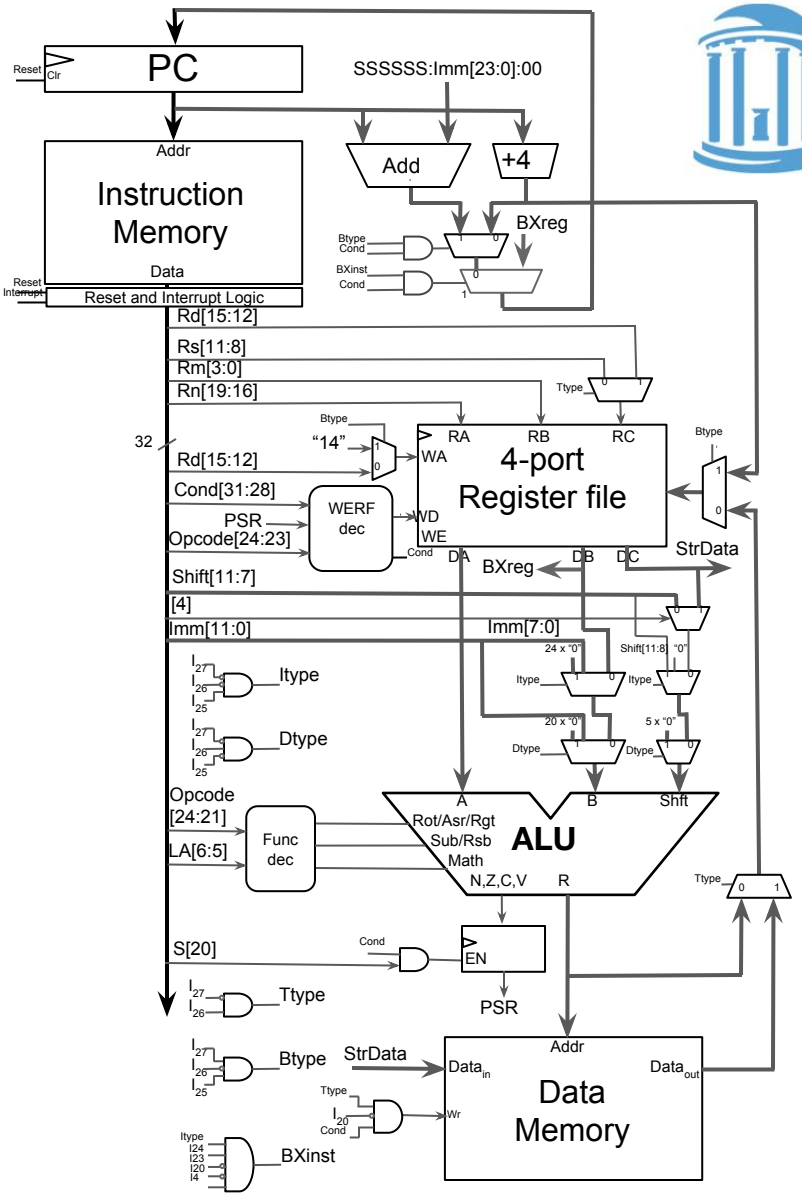
Latency = 90 min

How to Pipeline This?

A CPU is a digital circuit like any other. Thus, we should be able to pipeline it to increase its throughput.

However, there are a few tricky issues.

- It already has registers that get updated on each clock (register file, PSR, and PC)
- It has feedback, the ALU or Data Memory outputs are routed back to the register file



Our Goal

A simple 3-stage pipeline:

Fetch:

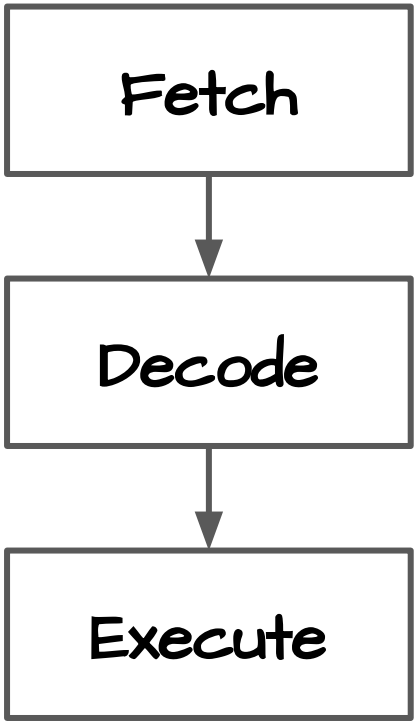
- Instruction memory access

Decode:

- Decode instructions
- Get register operands

Execute:

- ALU operation
- Write-back register



How Instructions Flow

Consider the following instruction sequence:

Progress in a three-stage pipeline

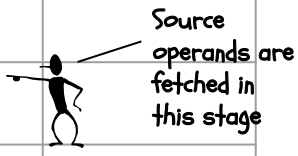
Once filled, at every clock there are 3 instructions at various stages of execution.

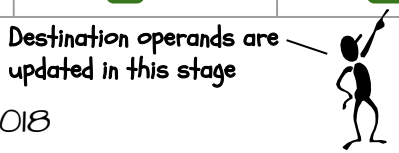
```
...
sub     r1,r1,r2
add     r2,r2,#2
and     r0,r0,#1
cmp     r1,r2
```

Time (in clock cycles)

Pipeline

	i	i+1	i+2	i+3	i+4	i+5
Fetch	sub r1,r1,r2	add r2,r2,#2	and r0,r0,#1	cmp r1,r2		
Decode		sub r1,r1,r2	add r2,r2,#2	and r0,r0,#1	cmp r1,r2	
Execute			sub r1,r1,r2	add r2,r2,#2	and r0,r0,#1	cmp r1,r2





Next Time

- Three pipeline registers on every datapath from the instruction memory's output to the register file's write data port.
- How much faster?