6.004 students work around the dryer bottleneck by finding a laundromat that has two dryers for every washer.

Looking at the timeline you can see the plan, which is divided into 30-minute steps.

The washer is in use every step, producing a newly-washed load every 30 minutes.

Dryer usage is interleaved, where Dryer #1 is used to dry the odd-numbered loads and Dryer #2 is used to dry the even-numbered loads.

Once started, a dryer runs for a duration of two steps, a total of 60 minutes.

Since the dryers run on a staggered schedule, the system as a whole produces a load of clean, dry laundry every 30 minutes.

The steady-state throughput is 1 load of laundry every 30 minutes and the latency for a particular load of laundry is 90 minutes.

And now here's the take-home message from this example.

Consider the operation of the two-dryer system.

Even though the component dryers themselves aren't pipelined, the two-dryer interleaving system is acting like a 2-stage pipeline with a clock period of 30 minutes and a latency of 60 minutes.

In other words, by interleaving the operation of 2 unpipelined components we can achieve the effect of a 2-stage pipeline.

Returning to the example of the previous section, we couldn't improve the throughput of our pipelined system past 1/8 ns because the minimum clock period was set by the 8 ns latency of the C module.

To improve the throughput further we either need to find a pipelined version of the C component or use an interleaving strategy to achieve the effect of a 2-stage pipeline using two instances of the unpipelined C component.

Let's try that… Here's a circuit for a general-purpose two-way interleaver, using, in this case, two copies of the unpipelined C component, C_0 and C_1.

The input for each C component comes from a D-latch, which has the job of capturing and holding the input value.

There's also a multiplexer to select which C-component output will be captured by the output register.

In the lower left-hand corner of the circuit is a very simple 2-state FSM with one state bit.

The next-state logic is a single inverter, which causes the state to alternate between 0 and 1 on successive clock cycles.

This timing diagram shows how the state bit changes right after each rising clock edge.

To help us understand the circuit, we'll look at some signal waveforms to illustrate its operation.

To start, here are the waveforms for the CLK signal and our FSM state bit from the previous slide.

A new X input arrives from the previous stage just after the rising edge of the clock.

Next, let's follow the operation of the C_0 component.

Its input latch is open when FSM Q is low, so the newly arriving X_1 input passes through the latch and C_0 can begin its computation, producing its result at the end of clock cycle #2.

Note that the C_0 input latch closes at the beginning of the second clock cycle, holding the X_1 input value stable even though the X input is starting to change.

The effect is that C_0 has a valid and stable input for the better part of 2 clock cycles giving it enough time to compute its result.

The C_1 waveforms are similar, just shifted by one clock cycle.

C_1's input latch is open when FSM Q is high, so the newly arriving X_2 input passes through the latch and C_1 can begin its computation, producing its result at the end of clock cycle #3.

Now let's check the output of the multiplexer.

When FSM Q is high, it selects the value from C_0 and when FSM Q is low, it selects the value from C_1.

We can see that happening in the waveform shown.

Finally, at the rising edge of the clock, the output register captures the value on its input and holds it stable for the remainder of the clock cycle.

The behavior of the interleaving circuit is like a 2-stage pipeline: the input value arriving in cycle i is processed over two clock cycles and the result output becomes available on cycle i+2.

What about the clock period for the interleaving system?

Well, there is some time lost to the propagation delays of the upstream pipeline register that supplies the X input, the internal latches and multiplexer, and the setup time of the output register.

So the clock cycle has to be just a little bit longer than half the propagation delay of the C module.

We can treat the interleaving circuit as a 2-stage pipeline, consuming an input value every clock cycle and producing a result two cycles later.

When incorporating an N-way interleaved component in our pipeline diagrams, we treat it just like a N-stage pipeline.

So N of our pipelining contours have to pass through the component.

Here we've replaced the slow unpipelined C component with a 2-way interleaved C-prime component.

We can follow our process for drawing pipeline contours.

First we draw a contour across all the outputs.

Then we add contours, ensuring that two of them pass through the C-prime component.

Then we add pipeline registers at the intersections of the contours with the signal connections.

We see that the contours passing through C-prime have caused extra pipeline registers to be added on the other inputs to the F module, accommodating the 2-cycle delay through C-prime.

Somewhat optimistically we've specified the C-prime minimum t_CLK to be 4 ns, so that means that the slow component which determines the system's clock period is now the F module, with a propagation delay of 5 ns.

So the throughput of our new pipelined circuit is 1 output every 5 ns, and with 5 contours, it's a 5-pipeline so the latency is 5 times the clock period or 25 ns.

By running pipelined systems in parallel we can continue to increase the throughput.

Here we show a laundry with 2 washers and 4 dryers, essentially just two copies of the 1-washer, 2-dryer system shown earlier.

The operation is as described before, except that at each step the system produces and consumes two loads of laundry.

So the throughput is 2 loads every 30 minutes for an effective rate of 1 load every 15 minutes.

The latency for a load hasn't changed; it's still 90 minutes per load.

We've seen that even with slow components we can use interleaving and parallelism to continue to increase throughput.

Is there an upper bound on the throughput we can achieve?

Yes!

The timing overhead of the pipeline registers and interleaving components will set a lower bound on the achievable clock period, thus setting an upper bound on the achievable throughput.

Sorry, no infinite speed-up is possible in the real world.