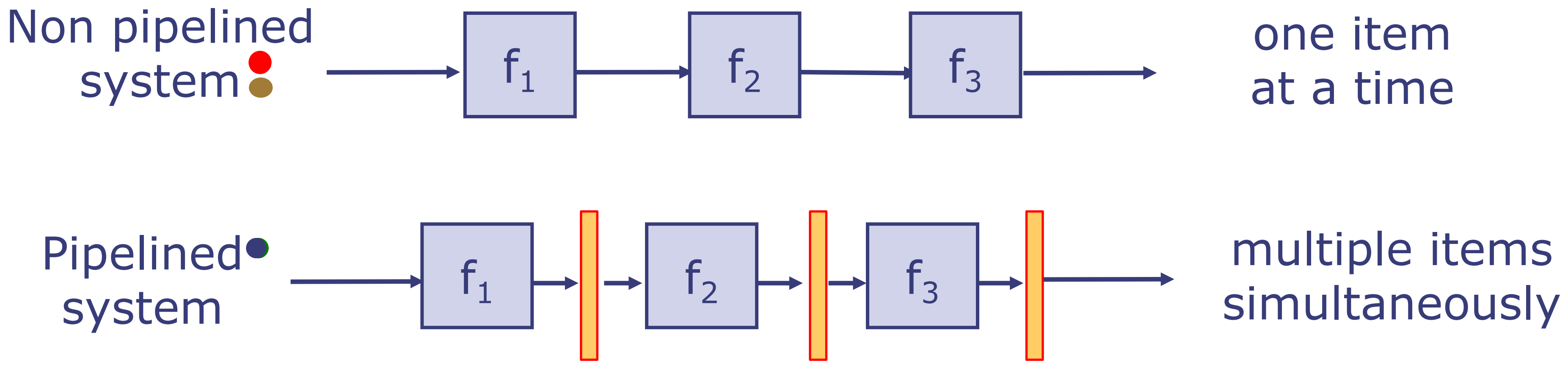
Introduction to Pipelining

Arvind
Computer Science & Artificial Intelligence Lab
M.I.T.

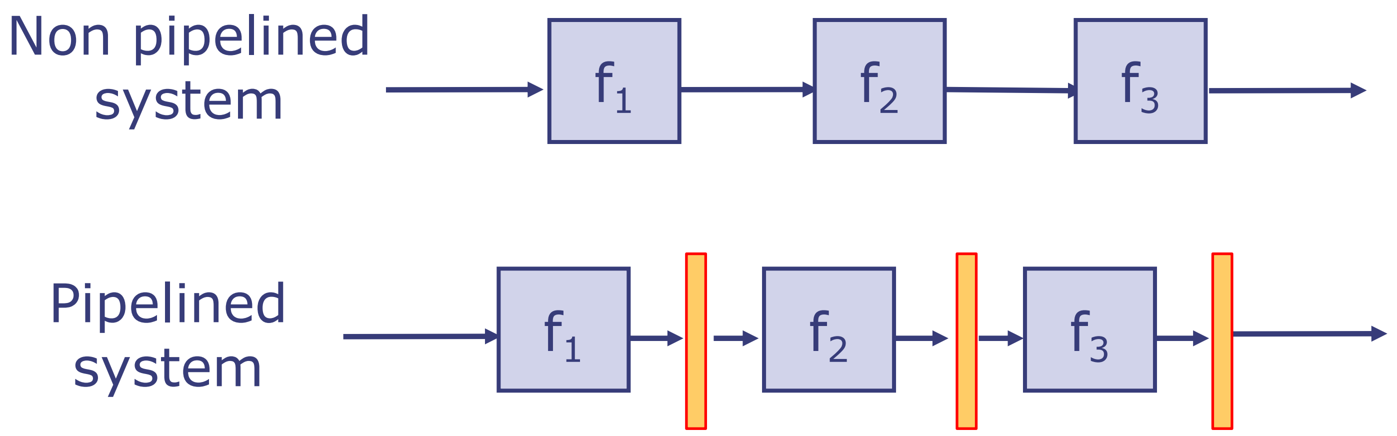
Pipelining: A technique to increase the throughput of a system



- All stages of a pipeline run at the same clock speed
- *Throughput:* Rate at which items are processed
- *Latency:* Total time to process 1 item
- The notion of an item and associated answer depends on the application, e.g.,
  - Arithmetic pipeline for function f: input is number x, output is f(x)
  - Processor: item is an instruction
- The concept of pipelining is applicable to both combinational and sequential circuits, i.e., each f_i could be a combinational or sequential circuit

we will first study combinational pipelines

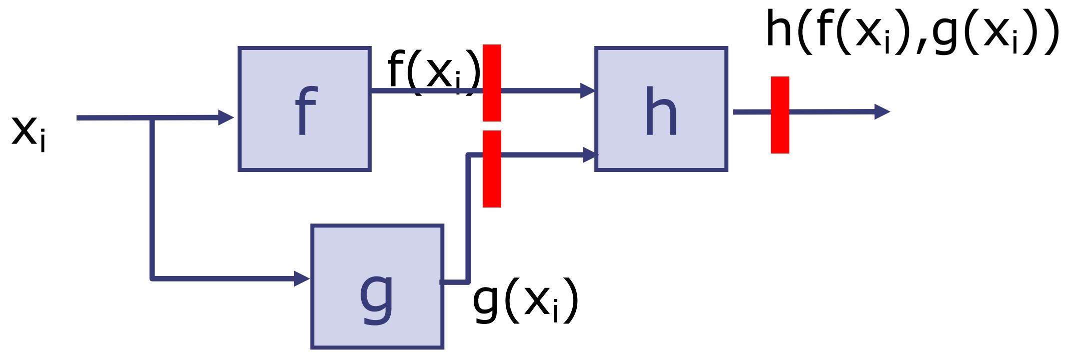
Example:
Latency and Throughput



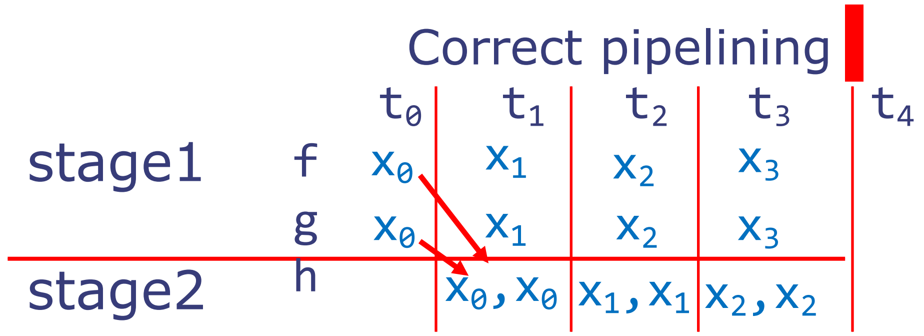
Suppose the latency of
- f_1 and f_3 is 5ns
- f_2 is 6ns
- register is 0ns

	Non pipelined	Pipelined
Clock period (ns)	5+6+5=16	6
Throughput (1/ns)	1/16	1/6
Latency (ns)	16	3*6 = 18

Pipelining combinational circuits



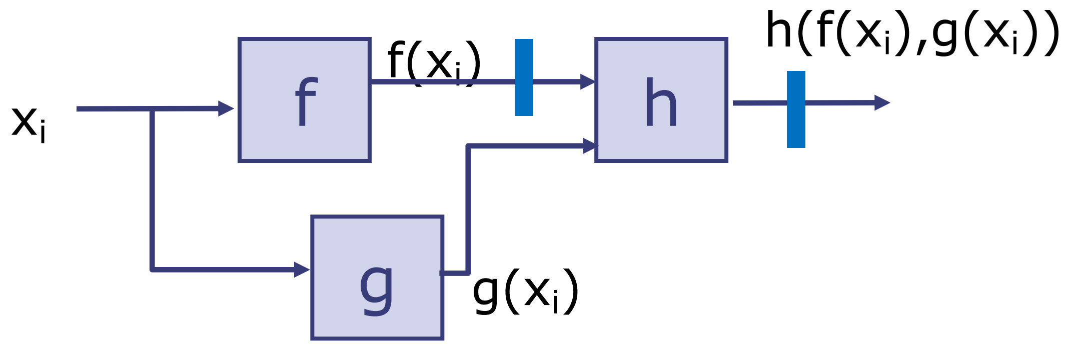
- Functionality of a combinational circuit can be specified as a pure function of inputs
- Correct pipelining must preserve the functionality



		t_0	t_1	t_2	t_3	t_4
stage1	f	x_0	x_1	x_2	x_3	
	g	x_0	x_1	x_2	x_3	
stage2	h		x_0,x_0	x_1,x_1	x_2,x_2	

output = $h(f(x_i),g(x_i))$

Incorrect Pipelining



Incorrect pipelining

	t_0	t_1	t_2	t_3
f	x_0	x_1		
g	x_0	x_1		
h	$(?,x_0)$	(x_0,x_1)		

output = $h(f(x_i),g(x_{i+1}))$

Every input to output path must have the same number of pipeline registers otherwise the computation would not divide into stages properly!

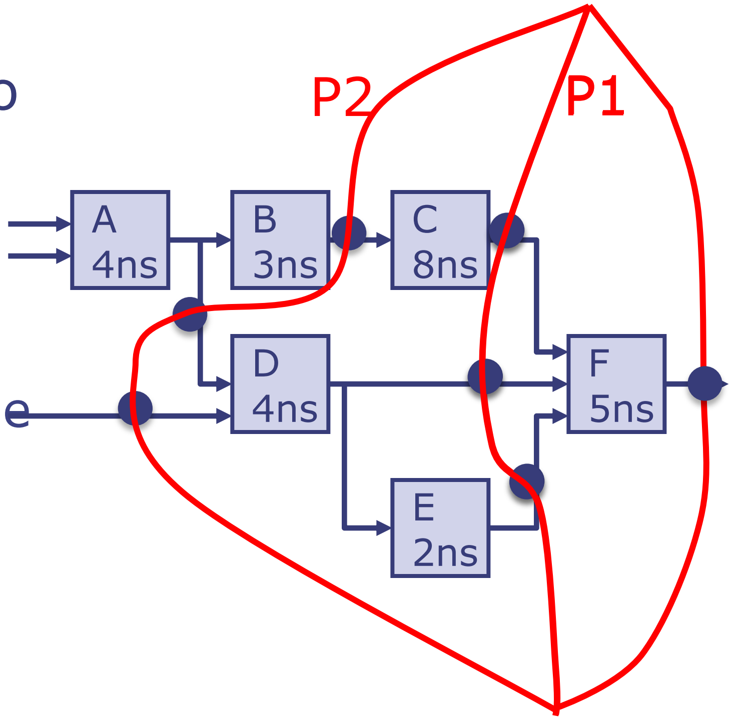
A methodology to pipeline combinational circuits

1. Draw a line crossing all outputs
2. Draw a line to partition the graph into two parts;
   - all the arrows must cross the partition line in the same direction
   - add a pipeline register at every point where the partition line crosses an edge

   P1: t_{CLK} = max{4+3+8,4+4,4+4+2,5} = 15

   P2: t_{CLK} = max{4+3,8+5,4+5,4+2+5} = 13
3. To increase the number of pipeline stages, further partition any partition using step 2

   P1+P2: t_{CLK} = max{4+3,8,4+2,5} = 8
4. For the fastest possible pipeline make the slowest box be the determinant of the clock speed, i.e., the bottleneck



Homework:
Draw another three stage pipeline that does not increase the clock period

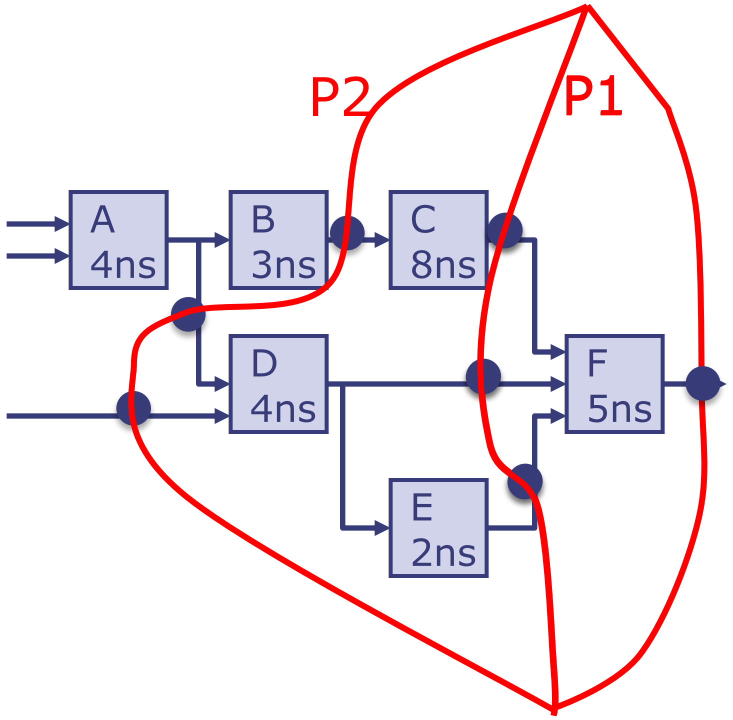
Latency and throughput of pipelined combinational circuit

Clock Period = 8 ns

Latency through full circuit:

3*8 = 24ns

Throughput = 1/8ns

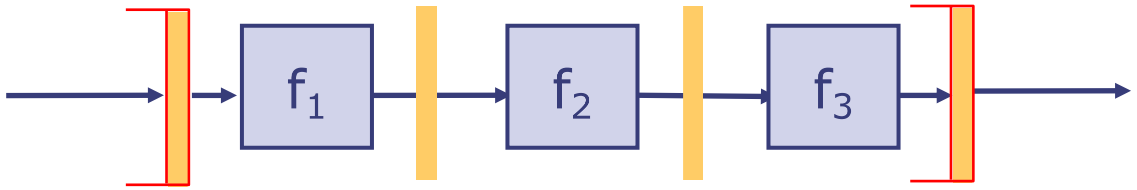


Tools for pipelining combinational circuits

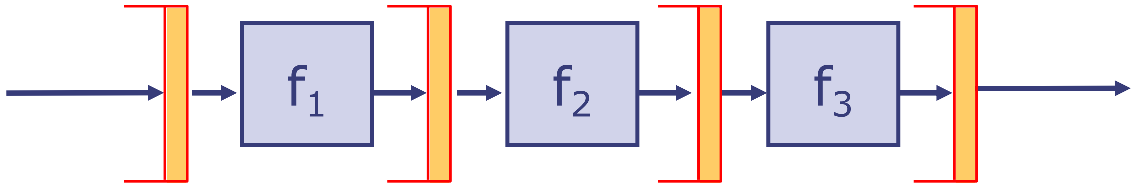
- Once we specify the number of stages or the clock period, modern tools can pipeline combinational circuits to generate fairly optimal pipelines
  - The method tools follow is similar to the one shown on the previous slide except that the tools work with networks of primitive gates

We won't be using such tools in this class!

Elastic vs Inelastic pipeline



Inelastic: Uses registers to separate pipeline stages;
All pipeline stages move synchronously;
It is complicated to start and stop a synchronous pipeline

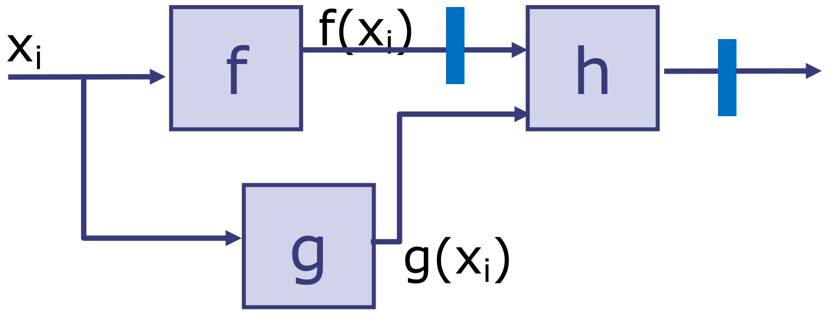


Elastic: Uses Fifos to separate pipeline stages;
A stage can process data if its input FIFO is not empty and output FIFO is not Full

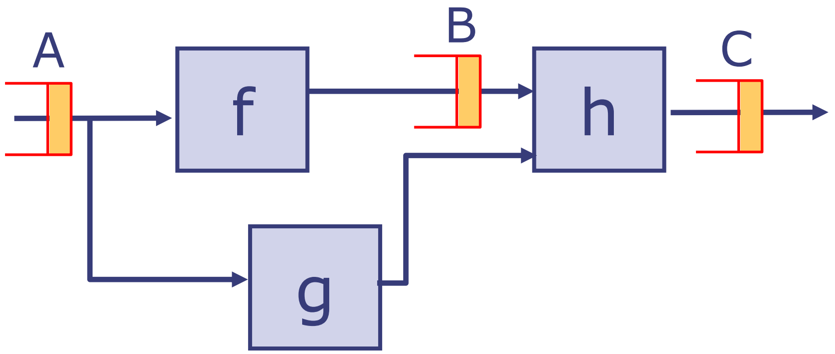
In this course we will focus exclusively on Elastic pipelines
Why?

Inelastic vs Elastic Pipelining

Incorrect synchronous pipeline
output = h(f(x_i),g(x_{i+1}))



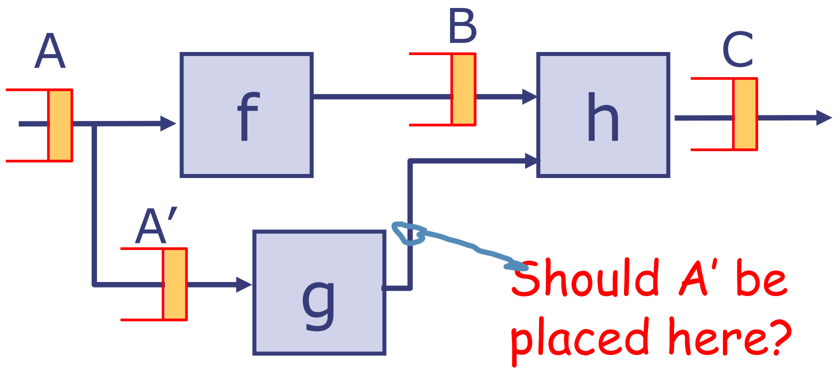
Assuming fifo B is empty initially, a rule describing this circuit will never fire



Introduction of an extra fifo A′ will remove the deadlock

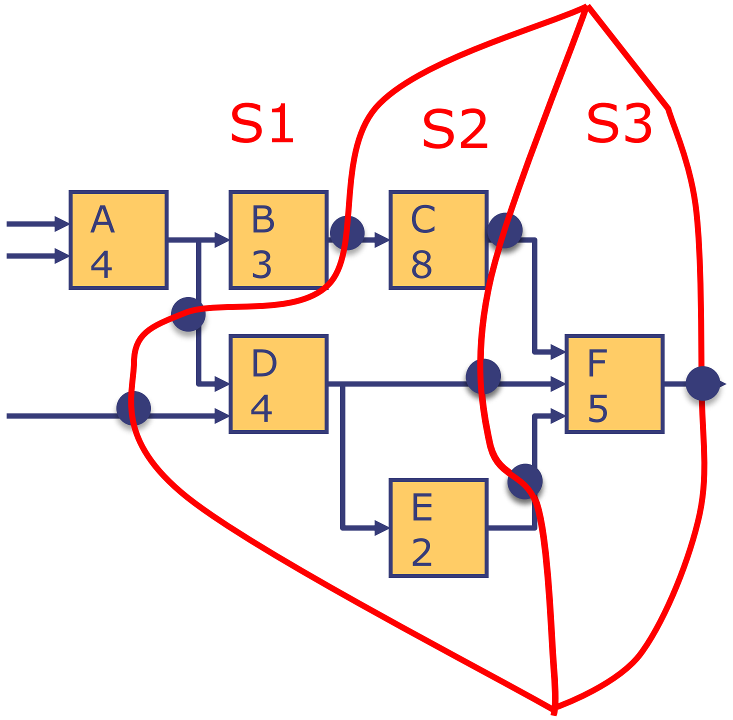
- rule 1 will enq in A′ and B
- rule 2 will enq in C

It is easier to preserve the correctness using fifos but balancing is still an issue



Pipelining sequential modules

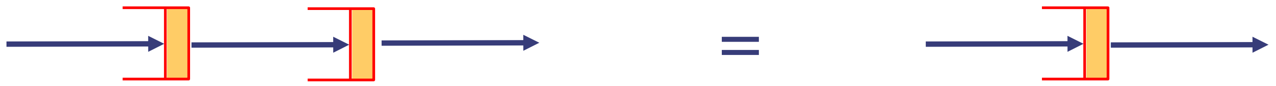
- Now suppose each box is sequential circuit (aka module) instead of a combinational circuit
- Let the number in each module represent the number of clock cycles that module takes to produce the output given an input
- Pipelined circuit still makes sense



```
Stage S1: 7 cycles
Stage S2: 8 cycles
Stage S3: 5 cycles
```

However, on an average we can't process inputs any faster than the slowest stage. After some time the output fifo of fast stage will get blocked by following slow stage

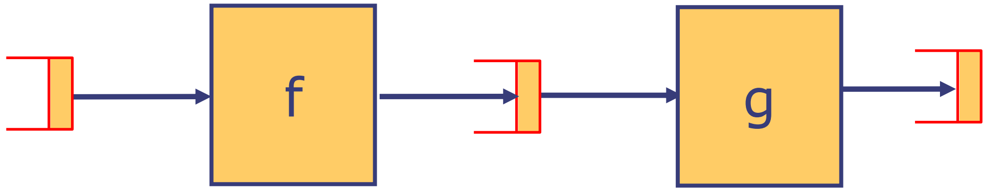
Cascaded Fifos



- When we draw a fifo, we usually don’t indicate the number of elements in it.
- Thus, cascaded fifo’s can be represented by one fifo

- Also for analysis purpose we will assume that the input Fifos of the system are never empty and output Fifos are never full

Pipeline bubbles



- Suppose f takes 1 or 2 cycles depending upon the input data, and g always takes one cycle
- Suppose input data to this system are x_0, x_1, x_2, x_3, where x_0, x_2 take one cycle each and x_1, x_3 take 2 cycles each

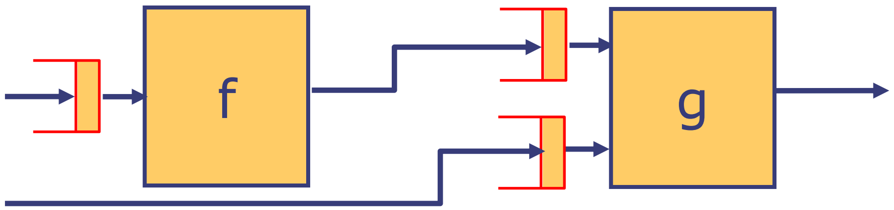
	t0	t1	t2	t3	t4	t5	t6	t7	t8
f	x_0	x_1 —	x_1	x_2	x_3 —	x_3			
g		x_0	☁	x_1	x_2	☁	x_3		

A bubble is introduced any time a pipeline stage has no work

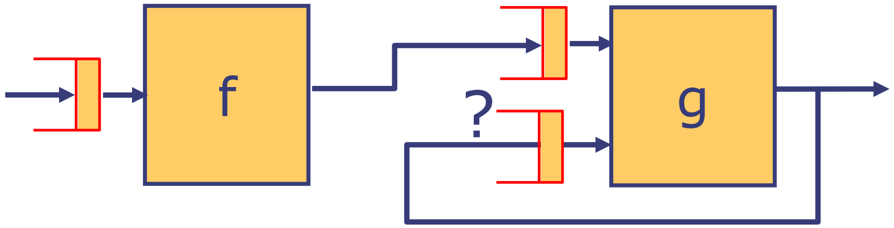
A bubble represents a loss of performance

Feed-Forward Pipelines

Feed-Forward pipeline



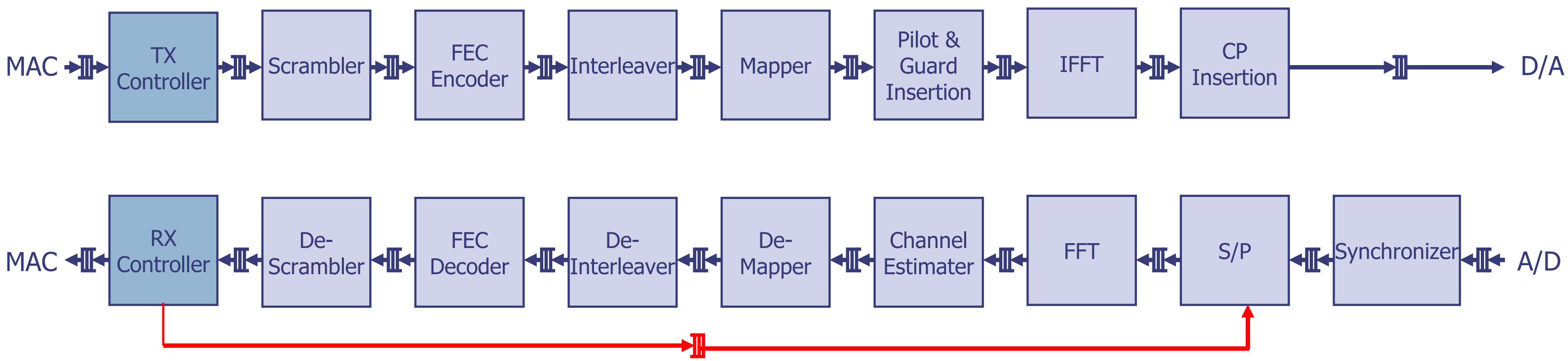
Pipeline with feedback



- Feed-forward pipelines have no feedback loops
  - Easy to insert pipeline stages and preserve functionality
- One can easily alter the functionality of a pipeline with feedback loops by inserting a fifo in the feed back path
- Pipelining a network of modules with loops requires careful analysis of what the pipeline is supposed to do
- Analysis is difficult because a module's interface does not say
  - if the module is internally pipelined
  - if it processes items in order

Some examples

Example of feed-forward pipeline
Baseband OFDM protocols



- WiFi, WiMax, WUSB are all examples of OFDM protocols
- All stages behave strictly in the first-in first-out manner
- Stages differ enormously in terms of complexity and area
- In spite of the similar structure, specific protocols have very different performance requirements, e.g.,
  - WiFi: 64pt @ 0.25MHz
  - WiMAX: 256pt @ 0.03MHz
  - WUSB: 128pt 8MHz

no brownie points for running the protocol faster than the specs

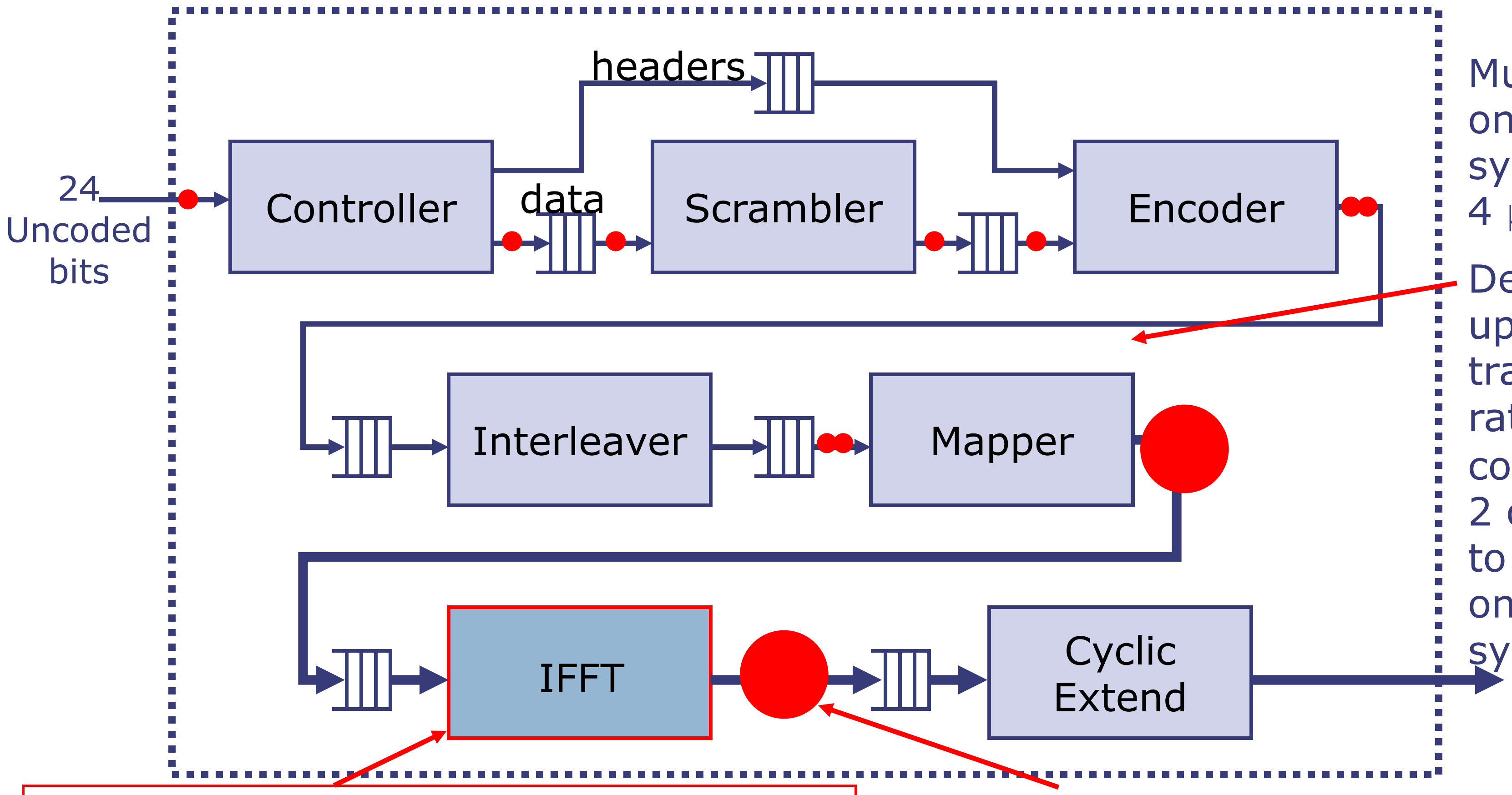
- There is a feedback path but it is only to shut down the pipeline

Designing feed-forward pipelines

- Requires optimizing area and clock speed to consume least amount of power while meeting the performance requirements
  - Some phases are much more complex computationally than others, and may have to be pipelined internally to meet the performance goals
  - Some phases can be turned into multicycle implementations to reduce area without affecting the performance
  - Some simple phases can be merged to save area

Design exploration is essential for good designs; Modules with Latency-Insensitive interfaces facilitate design exploration

802.11a Transmitter



Must produce one OFDM symbol every 4 μsec

Depending upon the transmission rate, consumes 1, 2 or 4 tokens to produce one OFDM symbol

IFFT Transforms 64 (frequency domain) complex numbers into 64 (time domain) complex numbers

One OFDM symbol (64 Complex Numbers)

🟦 accounts for 85% area

802.11a Transmitter Design:

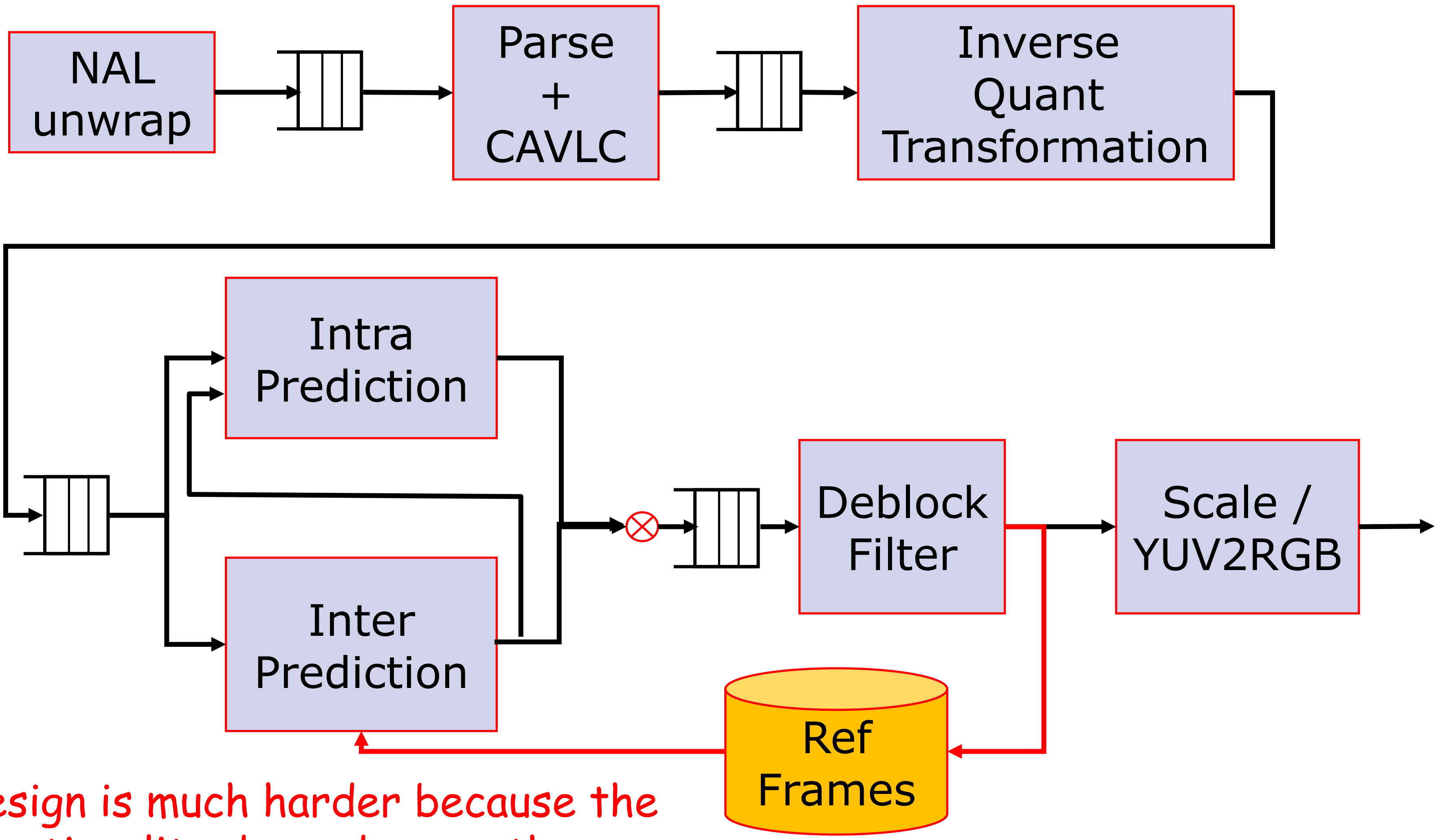
Design Block	Lines of Code (BSV)	Relative Area
Controller	49	0%
Scrambler	40	0%
Conv. Encoder	113	0%
Interleaver	76	1%
Mapper	112	11%
IFFT	95	85%
Cyc. Extender	23	3%

Complex arithmetic libraries constitute another 200 lines of code

[MEMOCODE 2006]

Pipelines with feedback

H.264 Decoder



Design is much harder because the functionality depends upon the feedback loop

Processor pipelines

- Pipelining processor provides the ultimate challenge in computer architecture
  - Stringent correctness requirements
  - Requires speculative execution of instructions to pipeline at all!
  - Requires dealing with a variety of feedbacks
  - The goal is always to achieve highest performance but within a given area and power budget

Next two lectures and your project will be on processor pipelines