EDA322
Digital Design

Lecture 16:

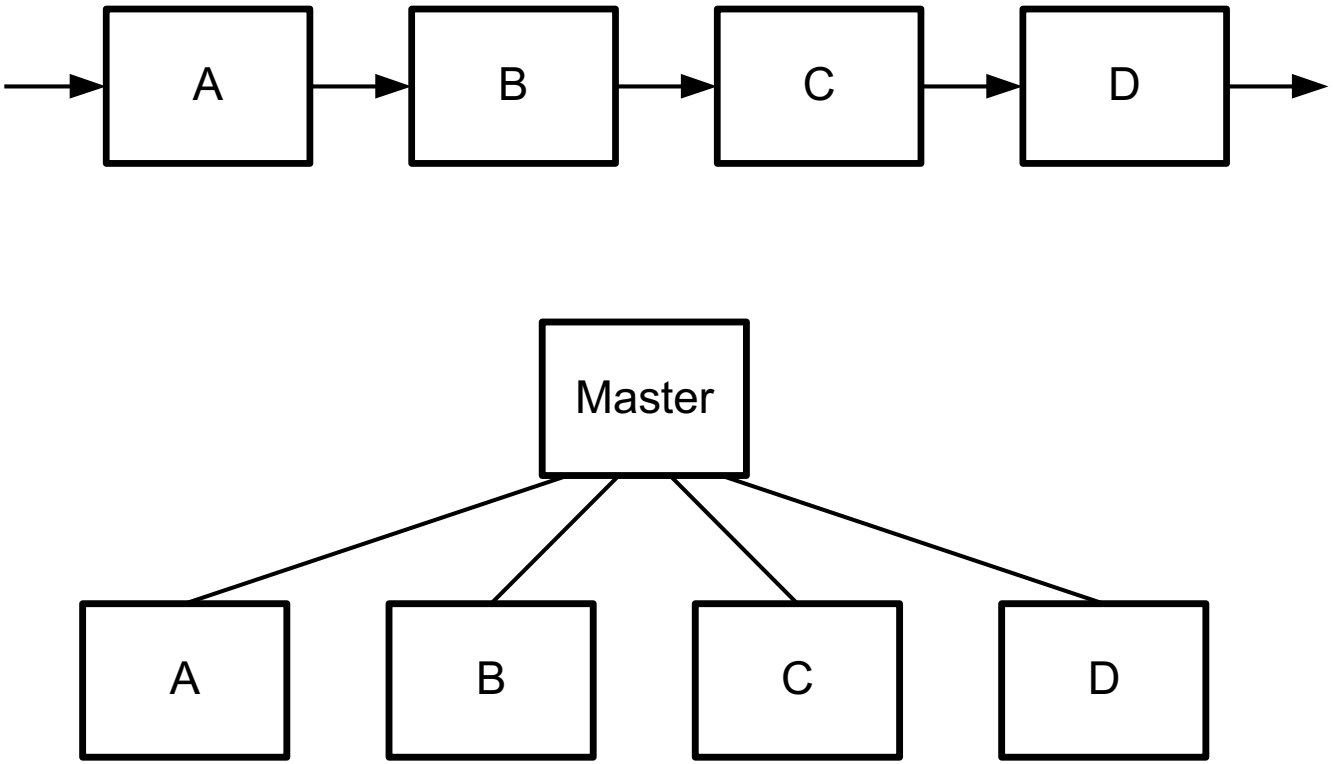
Pipelining

Ioannis Sourdis

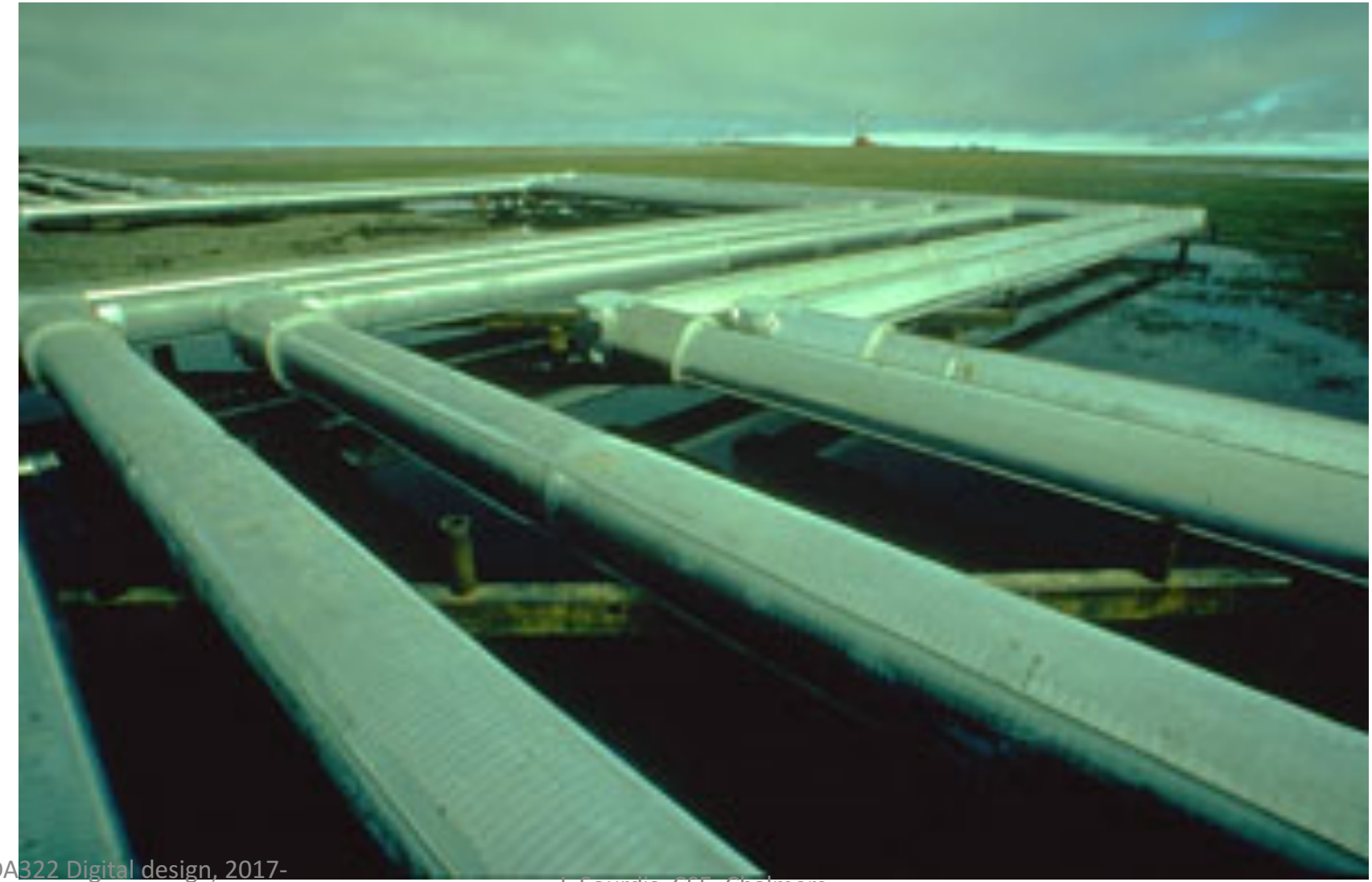
Outline of Lecture 16

- Pipelining
- Latency & Throughput
- Examples
  - Ripple carry adder
  - Microprocessor
  - Graphics
  - Network processing
- Load balancing
- Stalling

Two ways to make things faster: pipeline and parallel



Pipelines

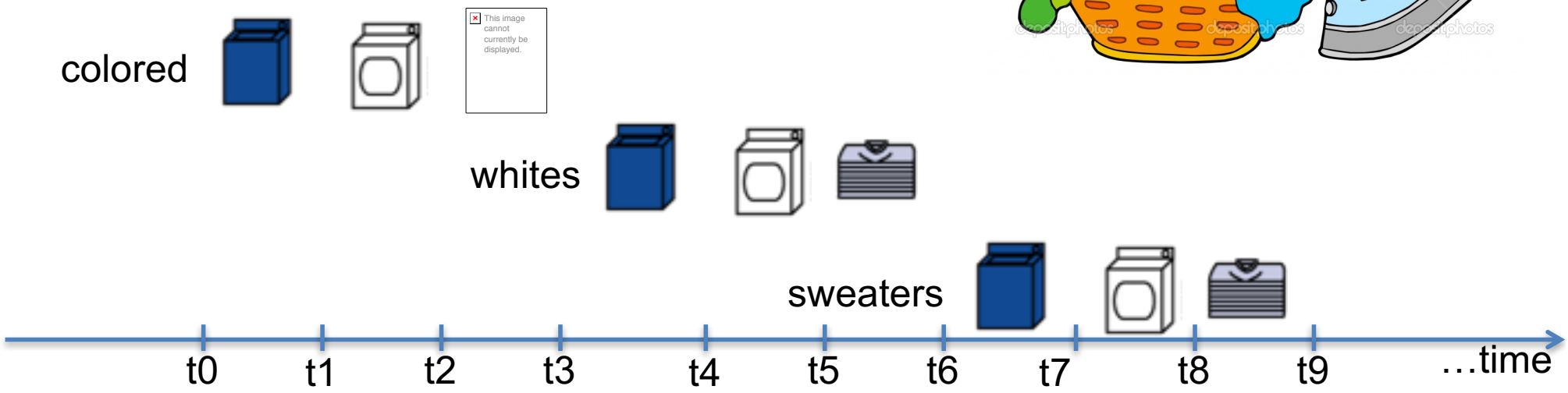


More Like an Assembly Line

Analogy: doing laundry

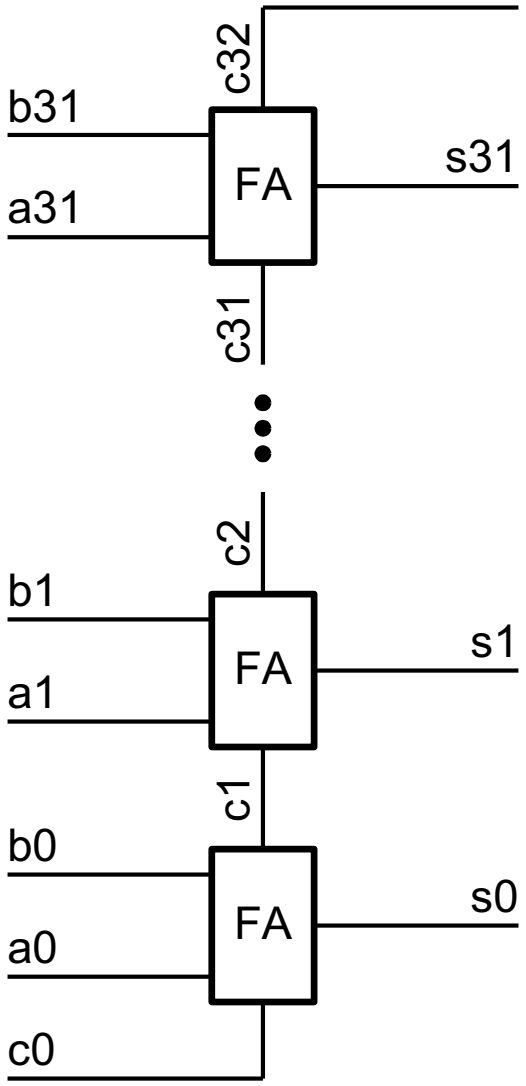
- Doing laundry:
  - Wash
  - Dry
  - Iron

- Can we do any better?

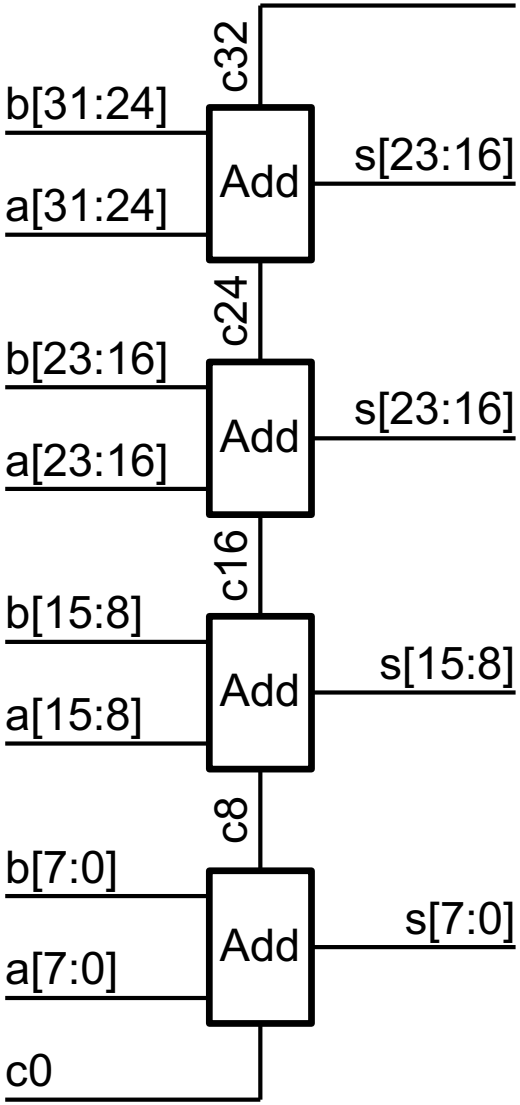


Pipelines

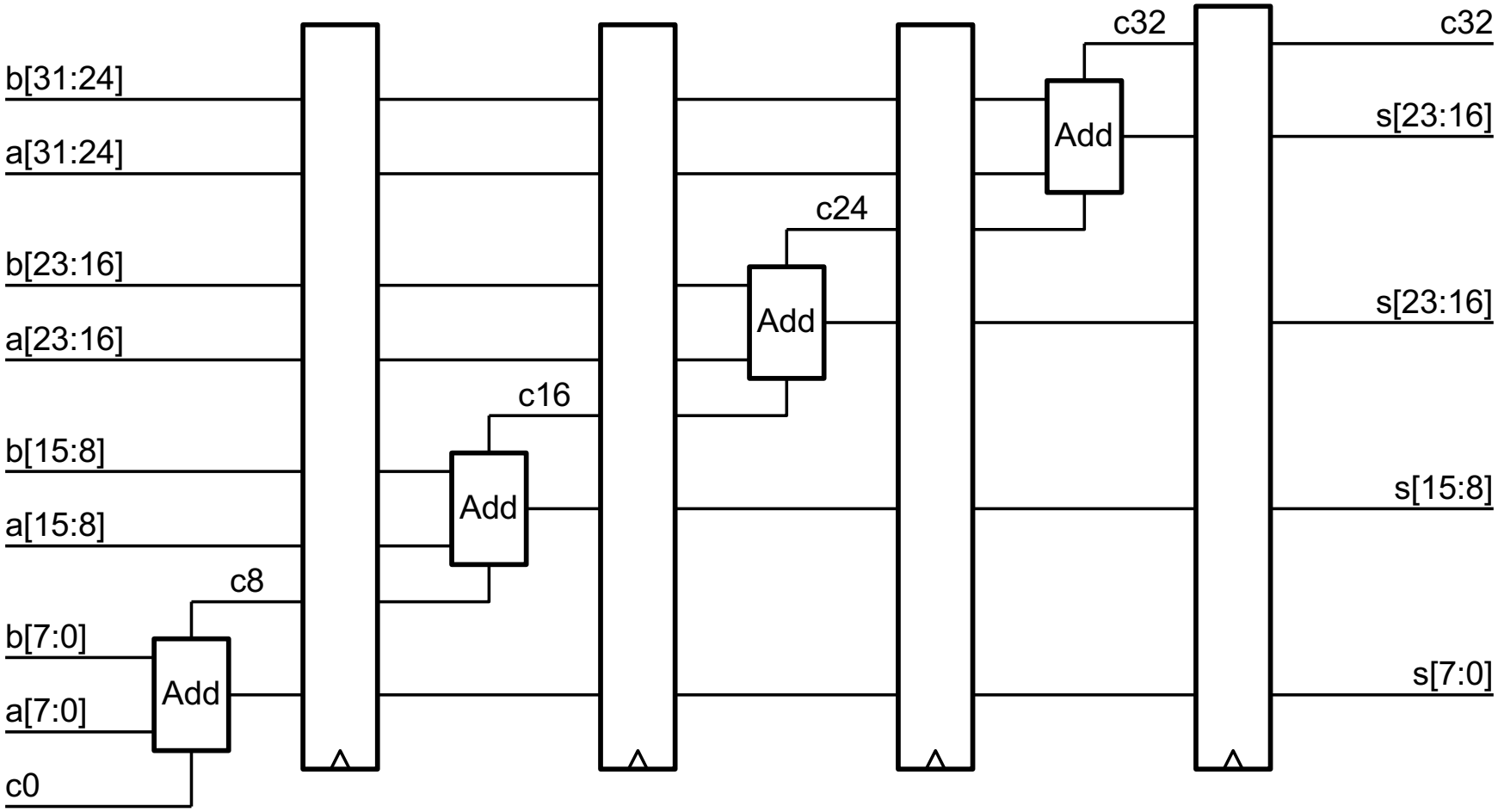
- Like an assembly line – each pipeline stage does part of the work and passes the ‘workpiece’ to the next stage
- Example 1: Pipelined 32b Adder



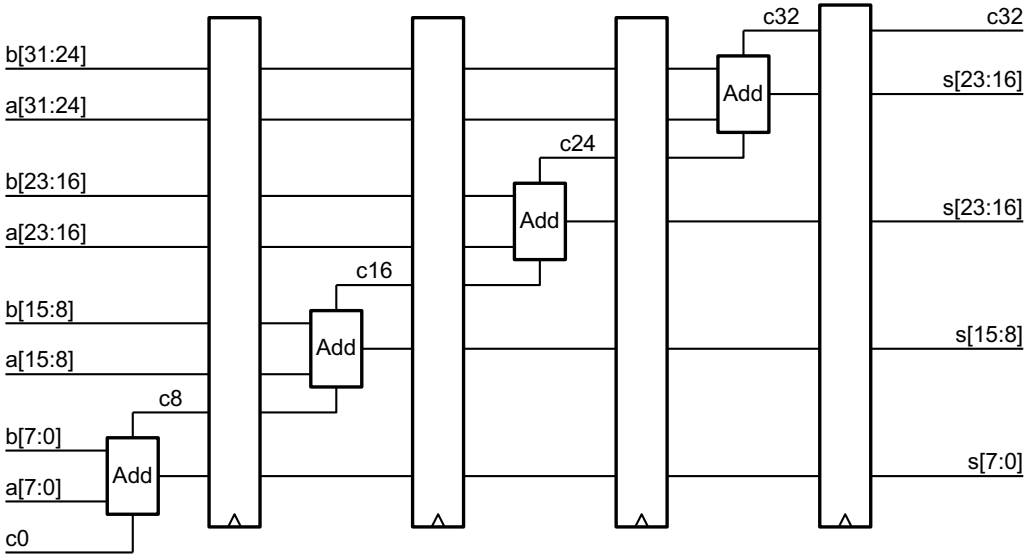
Split into 4 8-bit adders

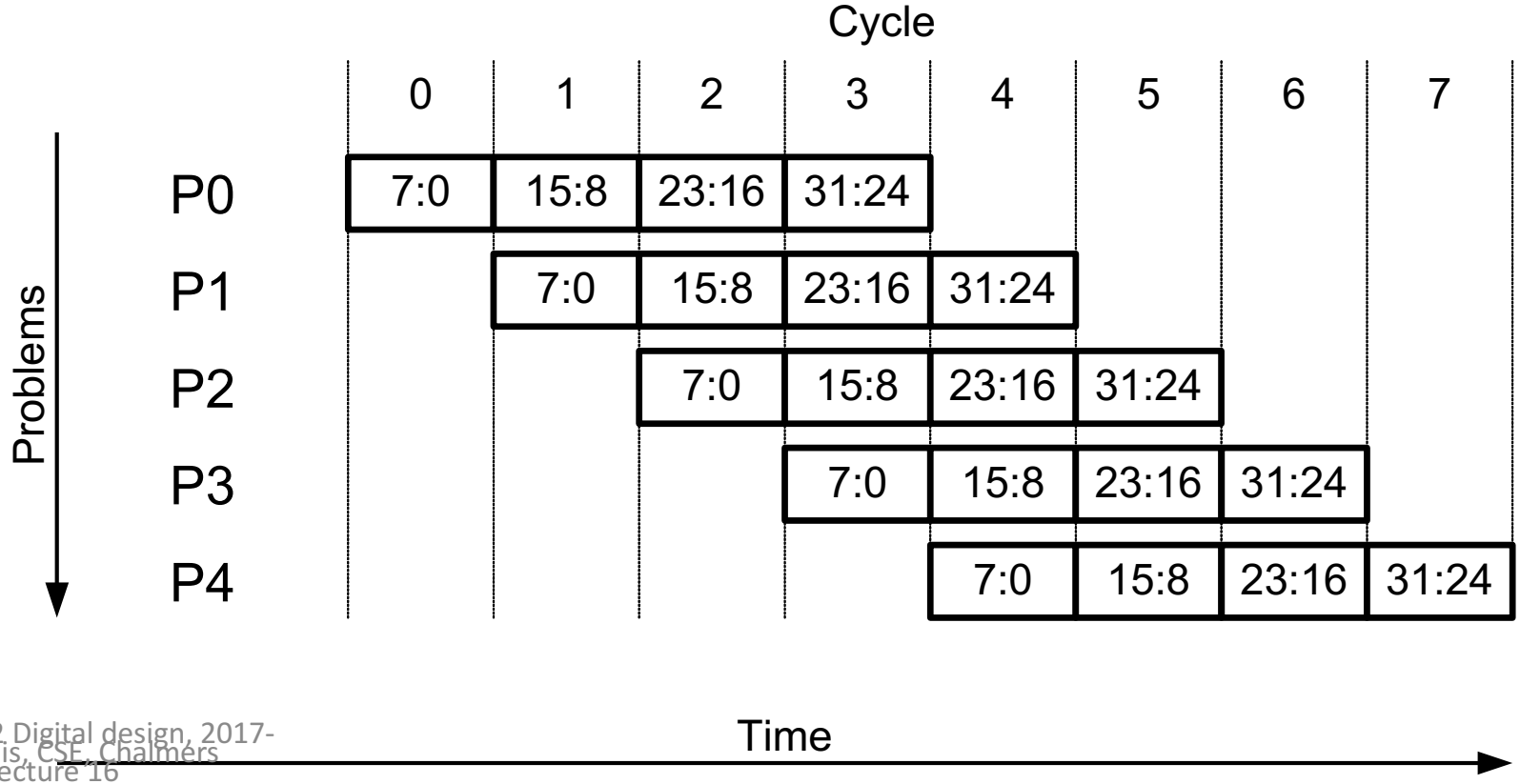


Split into stages
4 problems ‘in process’ at once

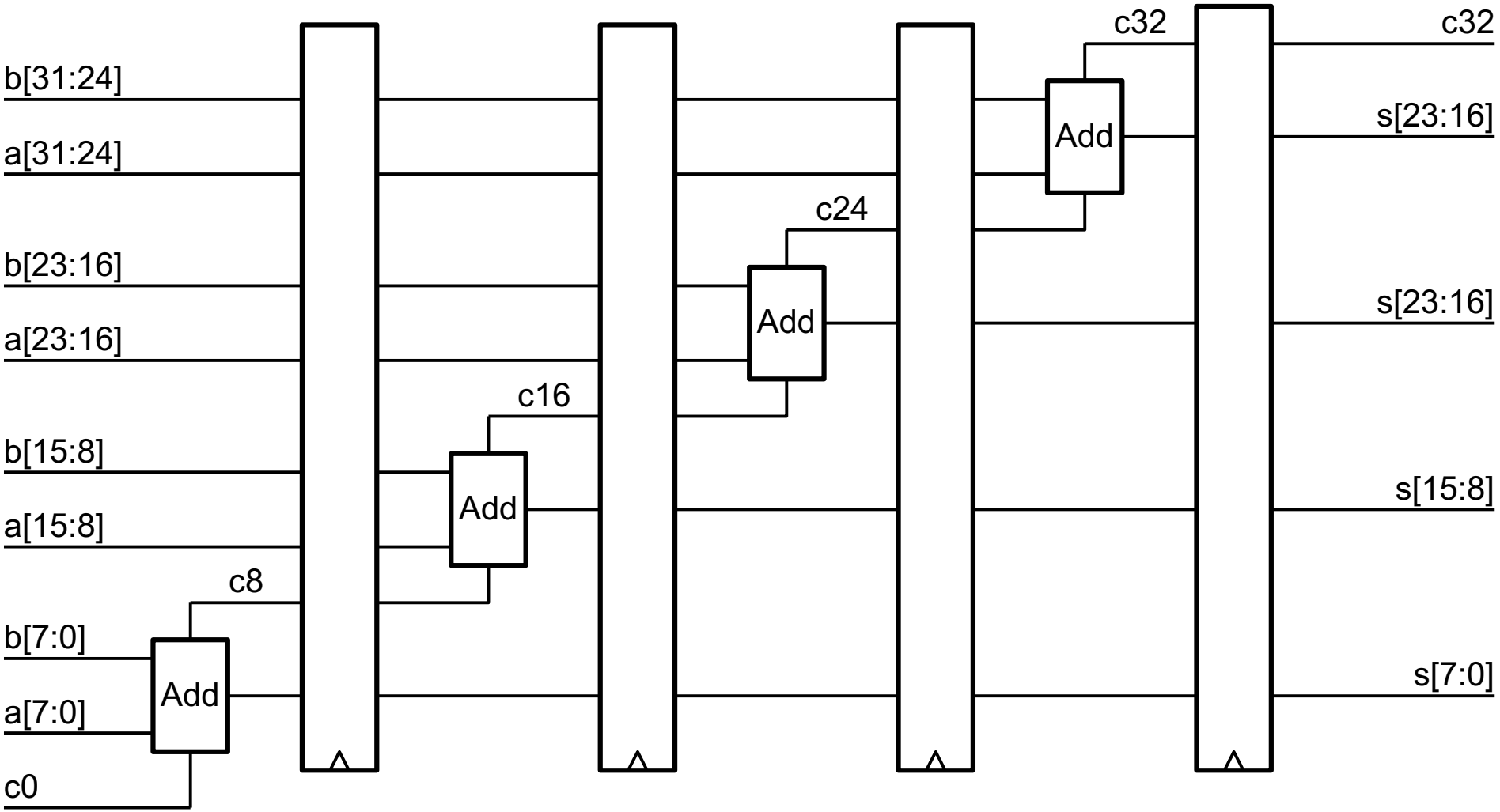


Pipeline Diagram Illustrates pipeline timing

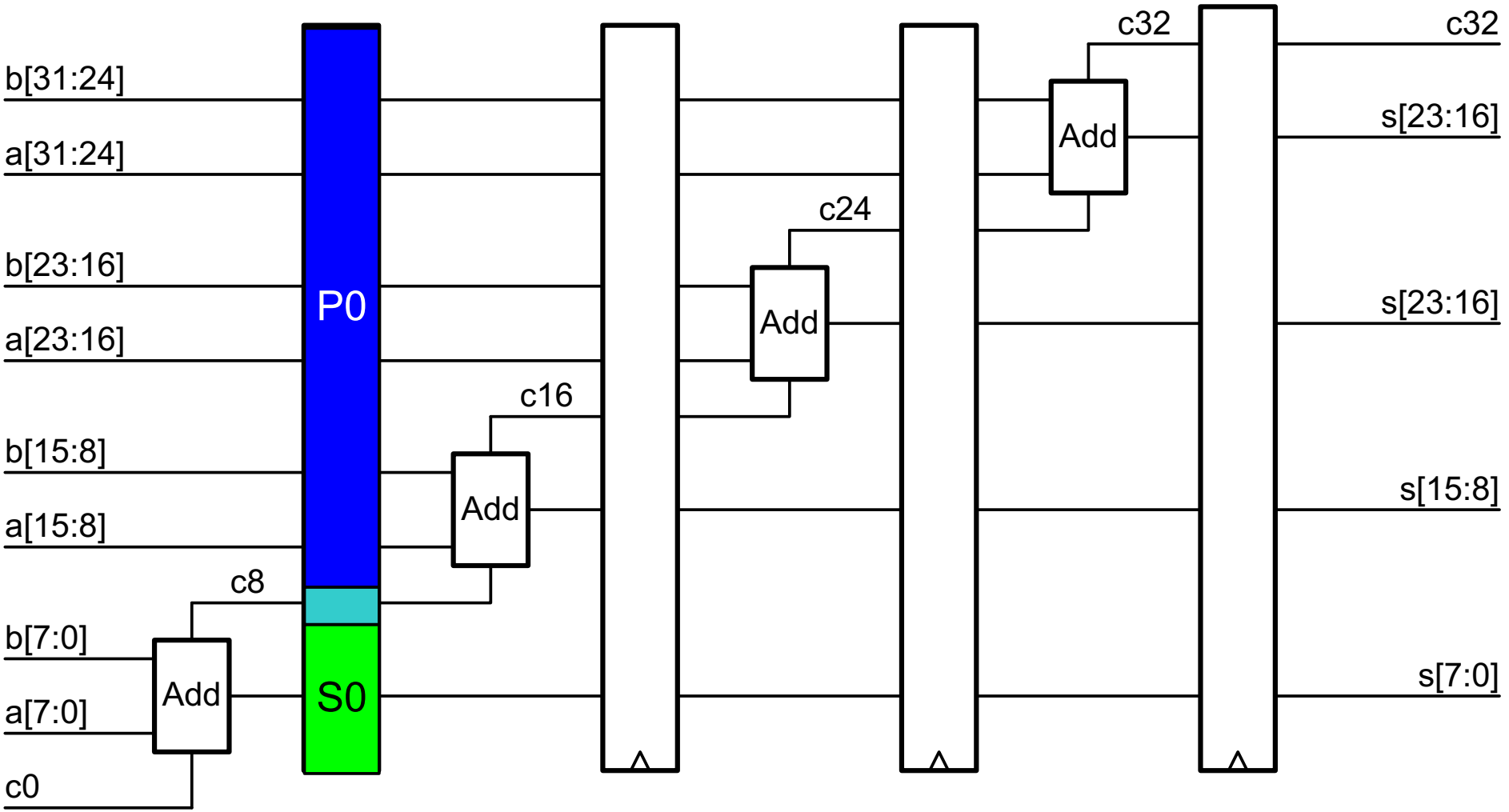




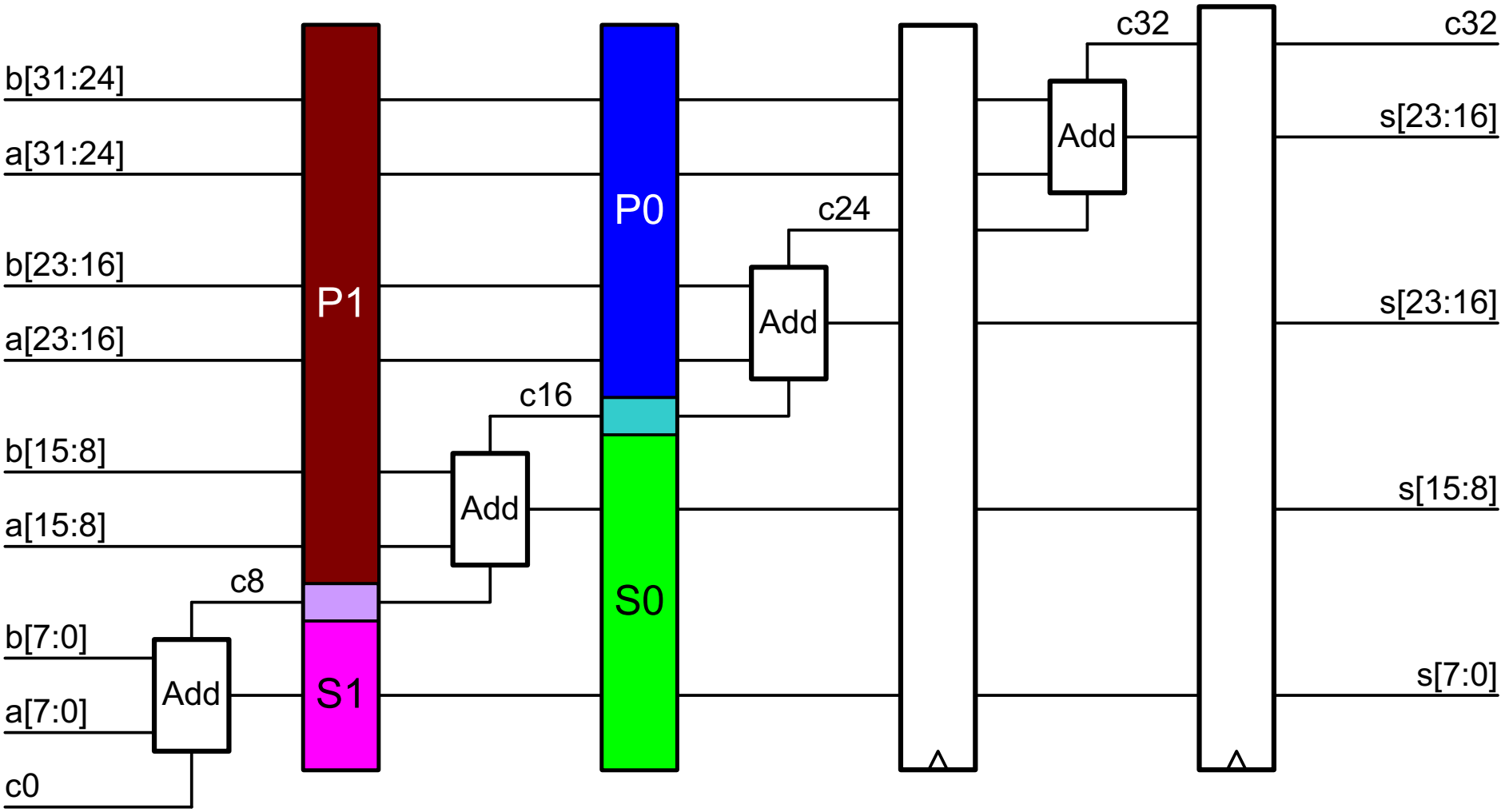
Movie


b[31:24]
a[31:24]
b[23:16]
a[23:16]
b[15:8]
a[15:8]
b[7:0]
a[7:0]
c0
c8
c16
c24
c32
Add
Add
Add
Add
c32
s[23:16]
s[23:16]
s[15:8]
s[7:0]

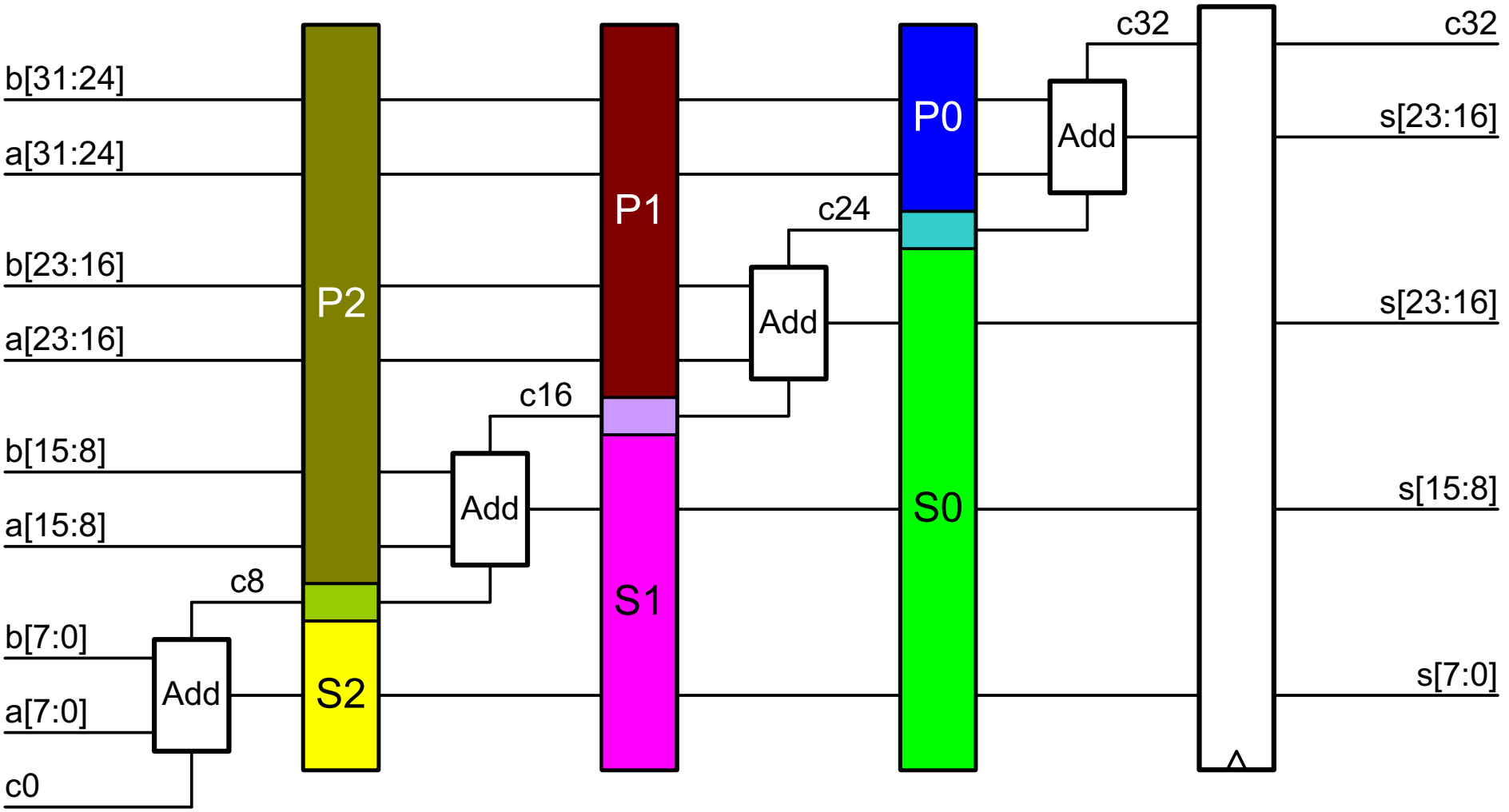
Cycle 1


b[31:24]
a[31:24]
b[23:16]
a[23:16]
b[15:8]
a[15:8]
b[7:0]
a[7:0]
c0
c8
c16
c24
c32
Add
Add
Add
Add
P0
S0
c32
s[23:16]
s[23:16]
s[15:8]
s[7:0]

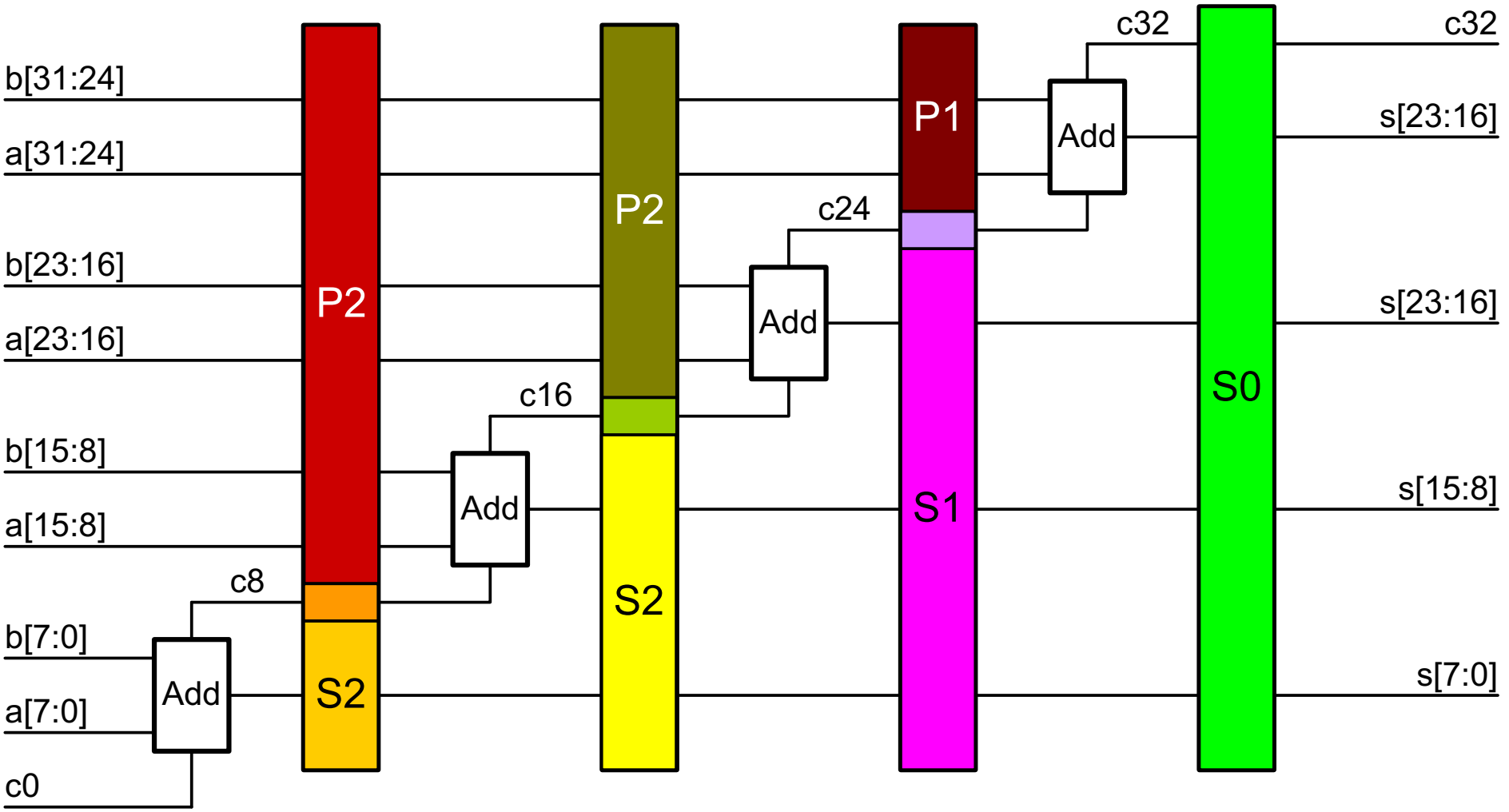
Cycle 2


b[31:24]
a[31:24]
b[23:16]
a[23:16]
b[15:8]
a[15:8]
b[7:0]
a[7:0]
c0
c8
c16
c24
c32
P1
S1
P0
S0
Add
Add
Add
Add
c32
s[23:16]
s[23:16]
s[15:8]
s[7:0]

Cycle 3


b[31:24]
a[31:24]
b[23:16]
a[23:16]
b[15:8]
a[15:8]
b[7:0]
a[7:0]
c0
c8
c16
c24
c32
P2
S2
P1
S1
P0
S0
Add
Add
Add
Add
c32
s[23:16]
s[23:16]
s[15:8]
s[7:0]

Cycle 4


b[31:24]
a[31:24]
b[23:16]
a[23:16]
b[15:8]
a[15:8]
b[7:0]
a[7:0]
c0
c8
c16
c24
c32
P2
P2
P1
S2
S2
S1
S0
Add
Add
Add
Add
c32
s[23:16]
s[23:16]
s[15:8]
s[7:0]

Latency and Throughput of a Pipeline

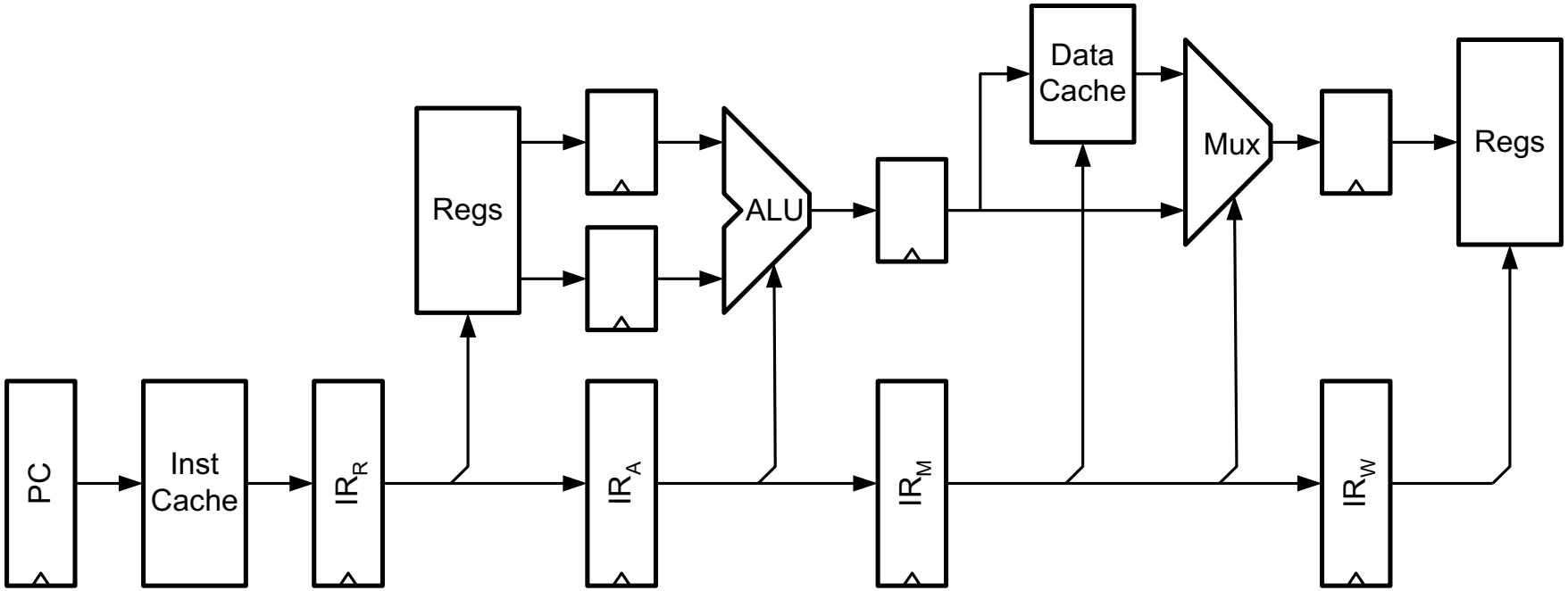
- Suppose before pipelining the delay of our 32b adder is 3200ps (100ps per bit) and this adder can do one problem each 3200ps for a *throughput* of 1/3200ps = 312 Millions of operations (additions) per second (Mops)

- What is the delay (latency, t_{pipe}) and throughput (Θ) of the adder with pipelining?
  Suppose td_{CQ} = 120ps, t_{setup} = 80ps, (200ps Overhead)

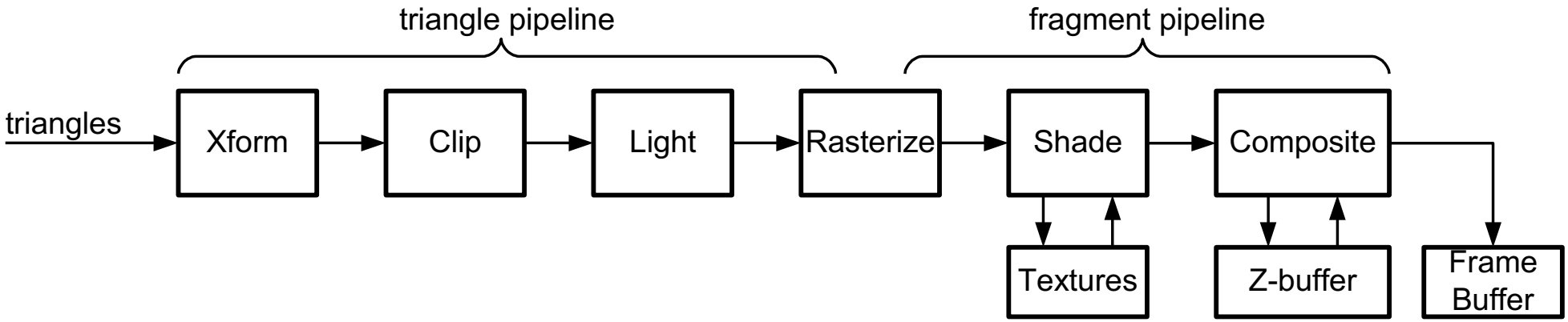
$t_{pipe} = n(t_{stage} + t_{dCQ} + t_s) = 4(800 + 200) = 4000$

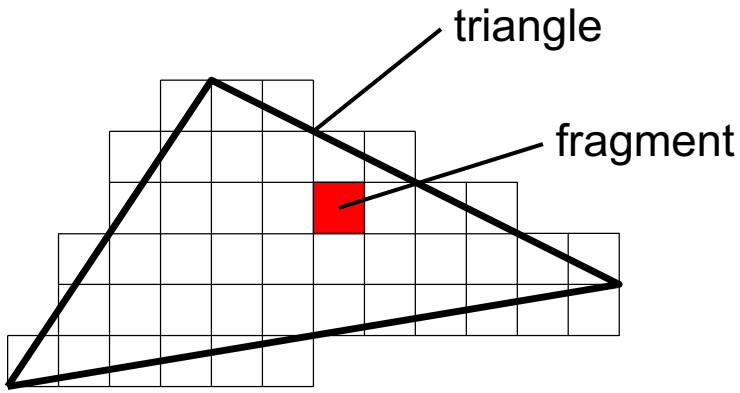
$\Theta = 1/(t_{stage} + t_{dCQ} + t_s) = 1/1000 = 1\text{Gops}$

Example 2: Processor Pipeline

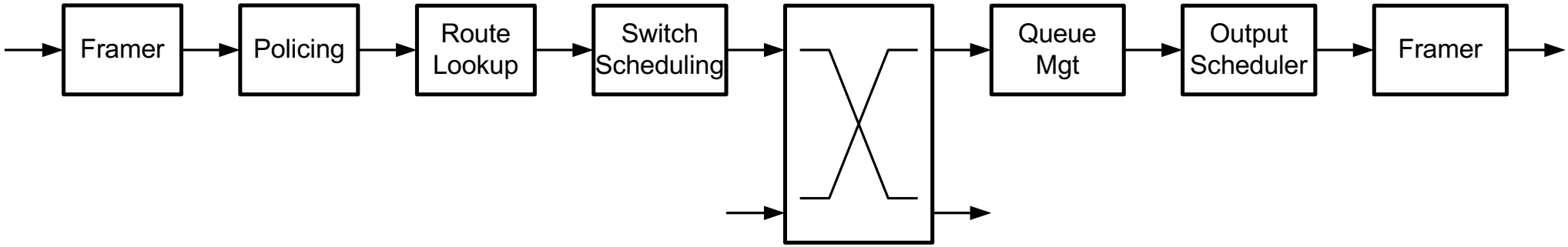


Example 3: Graphics rendering pipeline





Example 4 – Packet Processing Pipeline



And each of these modules is internally pipelined

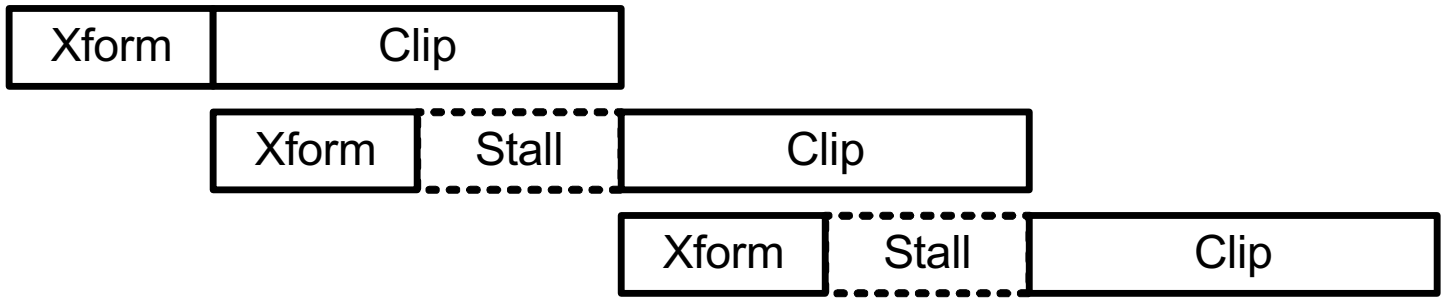
You get the idea. Lots of systems are organized this way.

Issues with pipelines (all deal with time per stage)

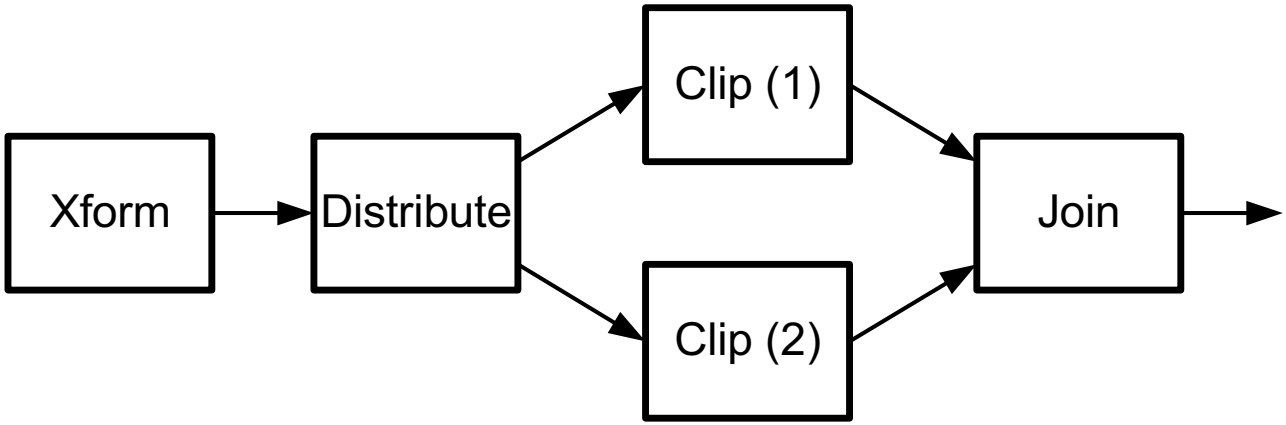
- Load balance (across stages)
  - one stage takes longer to process each input than the others – becomes a 'bottleneck'
  - Example
    - Rasterizing an 'average' triangle in a graphics pipeline takes more time than 'lighting' its vertices.
- Variable load (across data)
  - A given stage takes more time on some inputs than others
  - Example
    - The the time needed to rasterize a triangle is proportional to the number of fragments in the triangle. The average triangle may contain 20 fragments, but triangles range from 0 to over 1M
- Long latency
  - A stage may require a long latency operation (e.g., texture access)

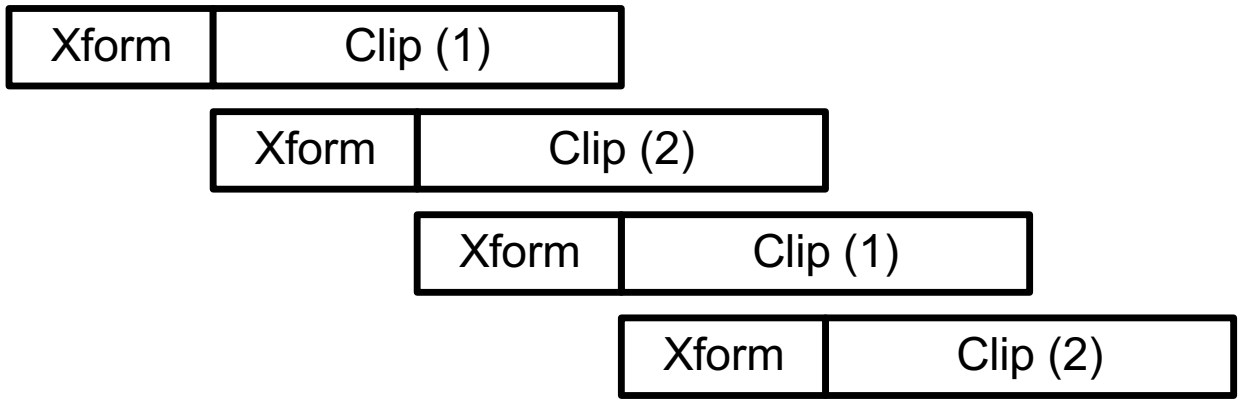
Load Balancing Pipelines

- Suppose transform takes 2 cycles and clip 4 cycles
- Clip is a ‘bottleneck’ pipeline stage
- Xform unit is busy only half the time

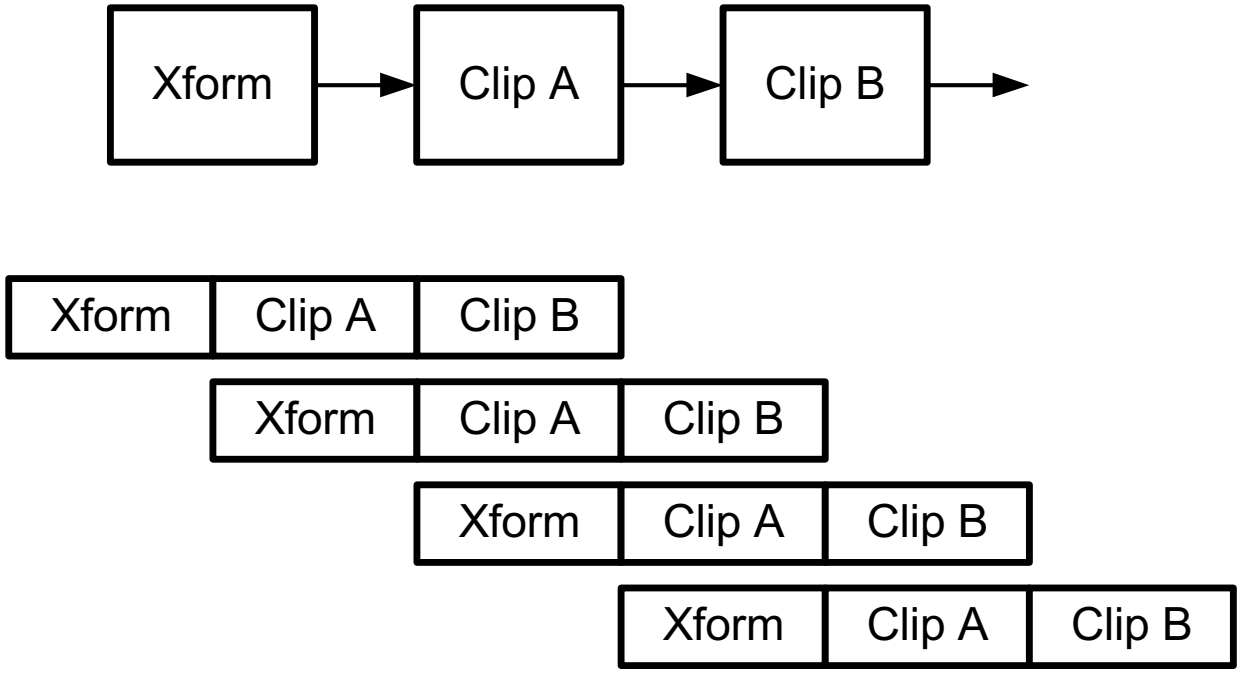


Load Balancing Solutions
1 – Parallel copies of slow unit

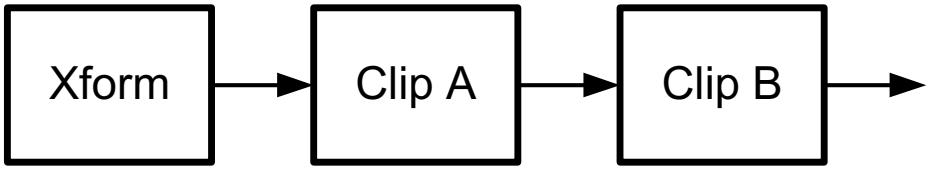


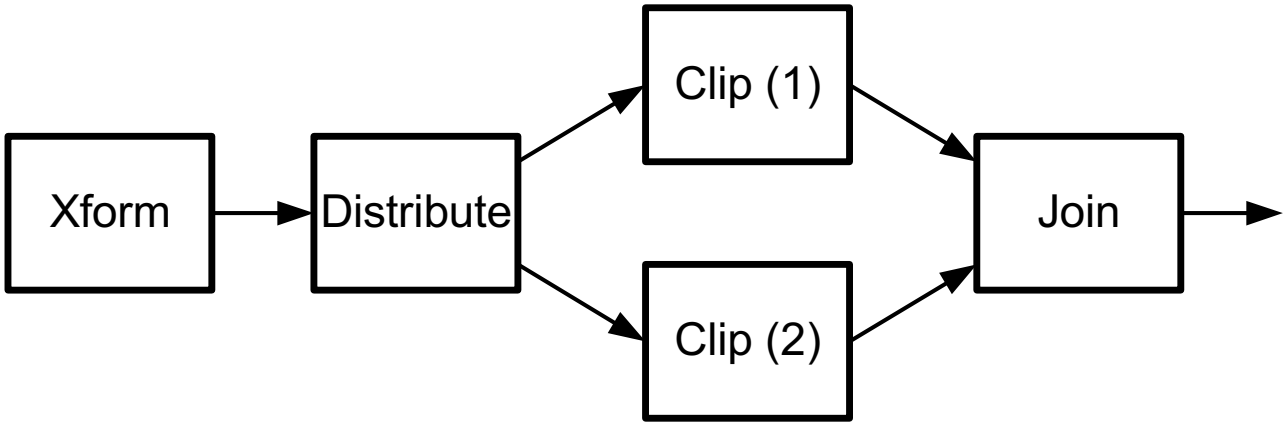


Load Balancing Solutions
2 – Split slow pipeline stage



When is it better to split? To copy? Throughput and latency are the same



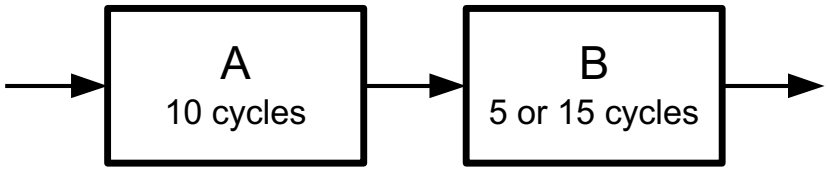


Variable load

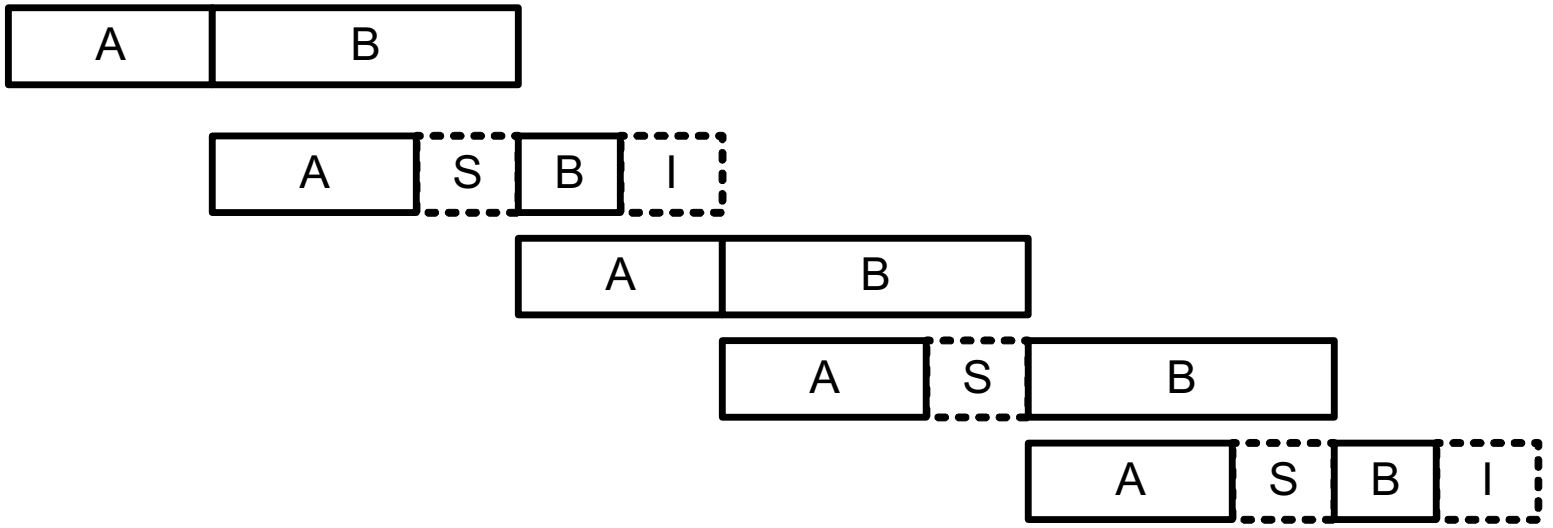
Stage A always takes 10 cycles.

Stage B takes 5 or 15 cycles – averages 10 cycles

Pipeline averages ____ cycles per element

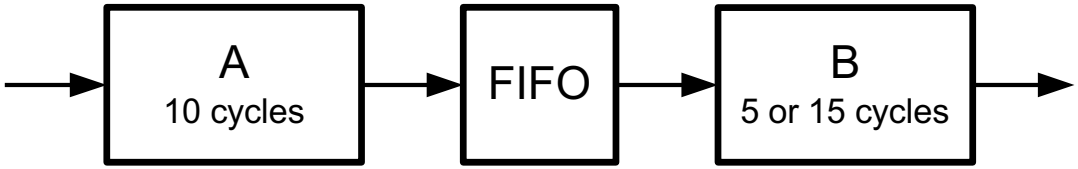


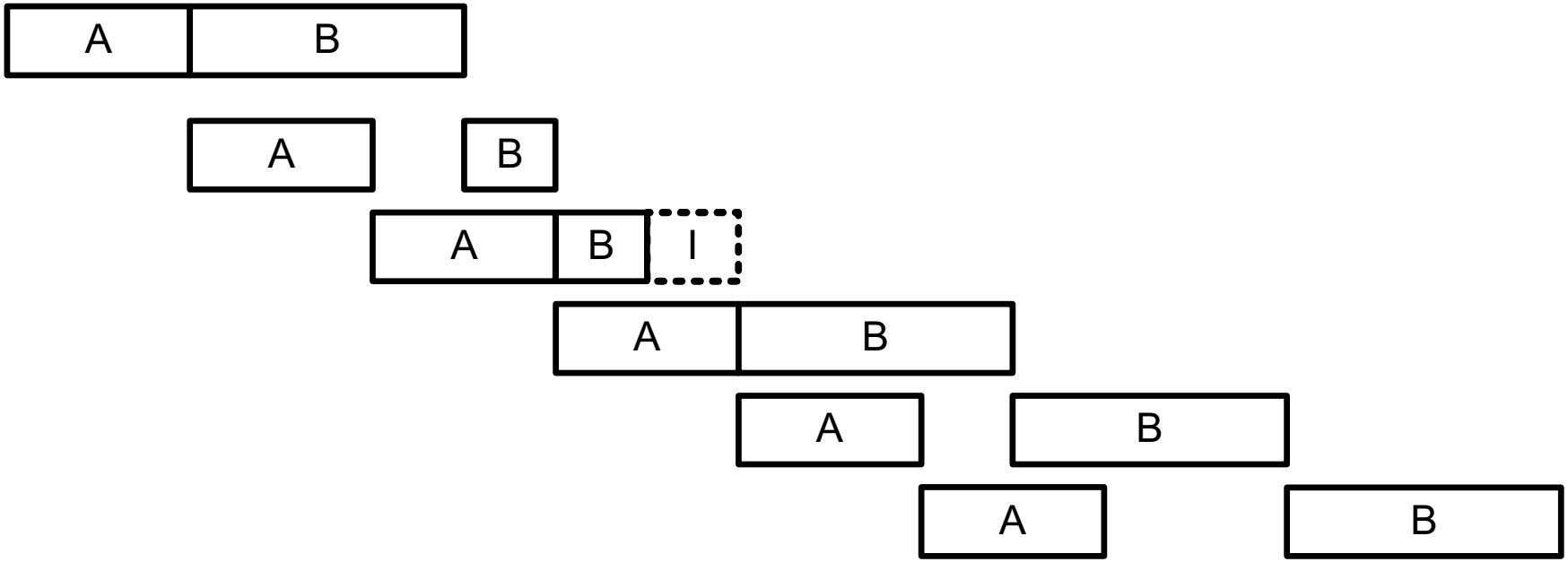
S: stall
I: idle



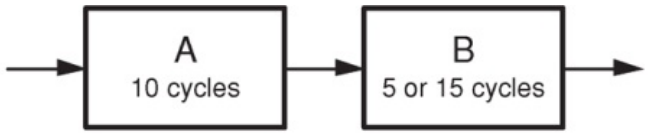
Elastic Pipelines

A FIFO between stages decouples timing
Allows stages to operate at their 'average' speed

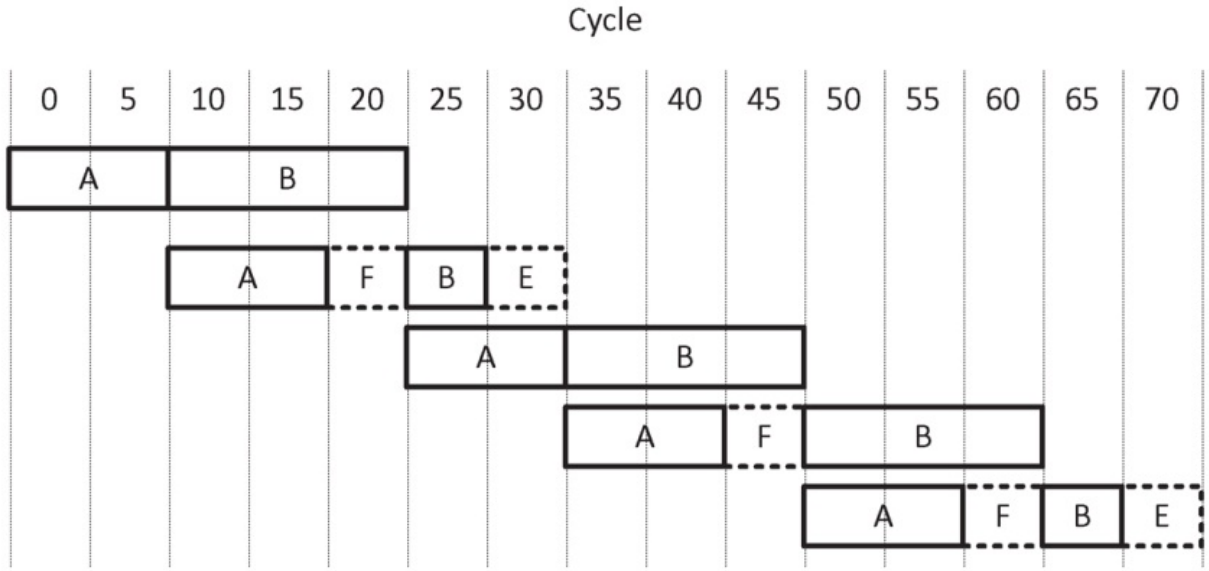




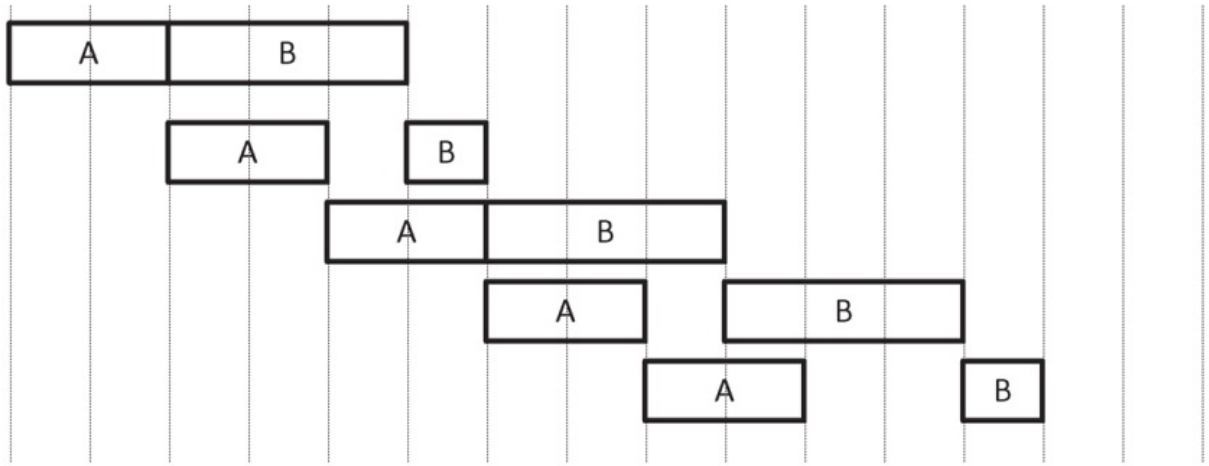
Variable Load



(a)

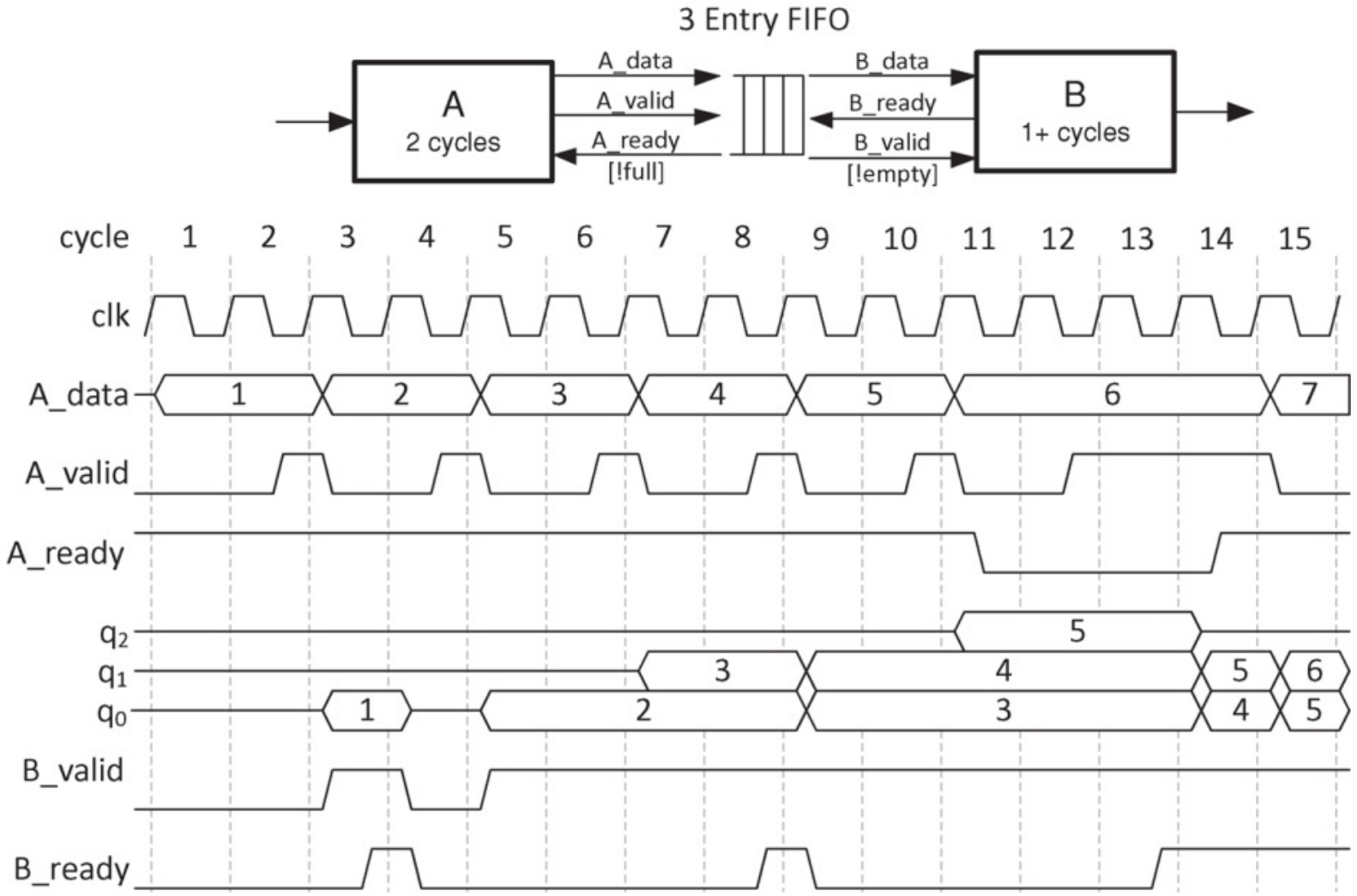


(b)



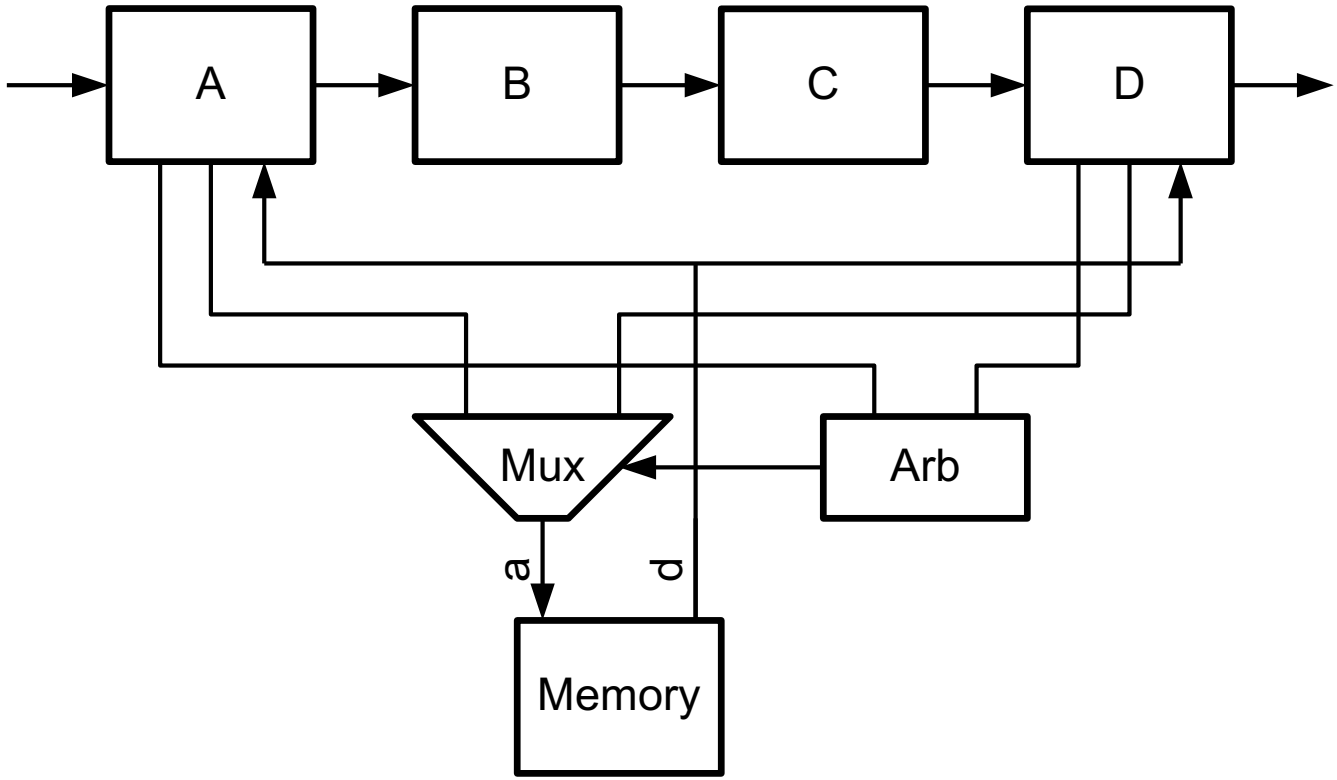
(c)

Timing of an elastic pipeline



Resource Sharing

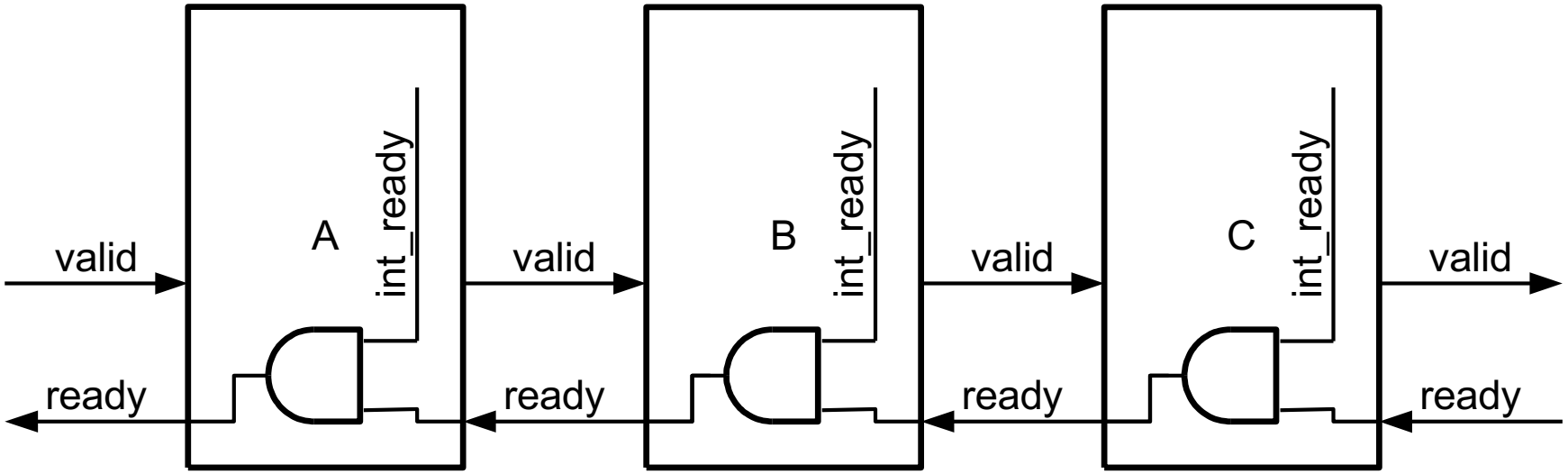
Suppose two pipeline stages need to access the same memory



How would you set the priority on the arbiter?

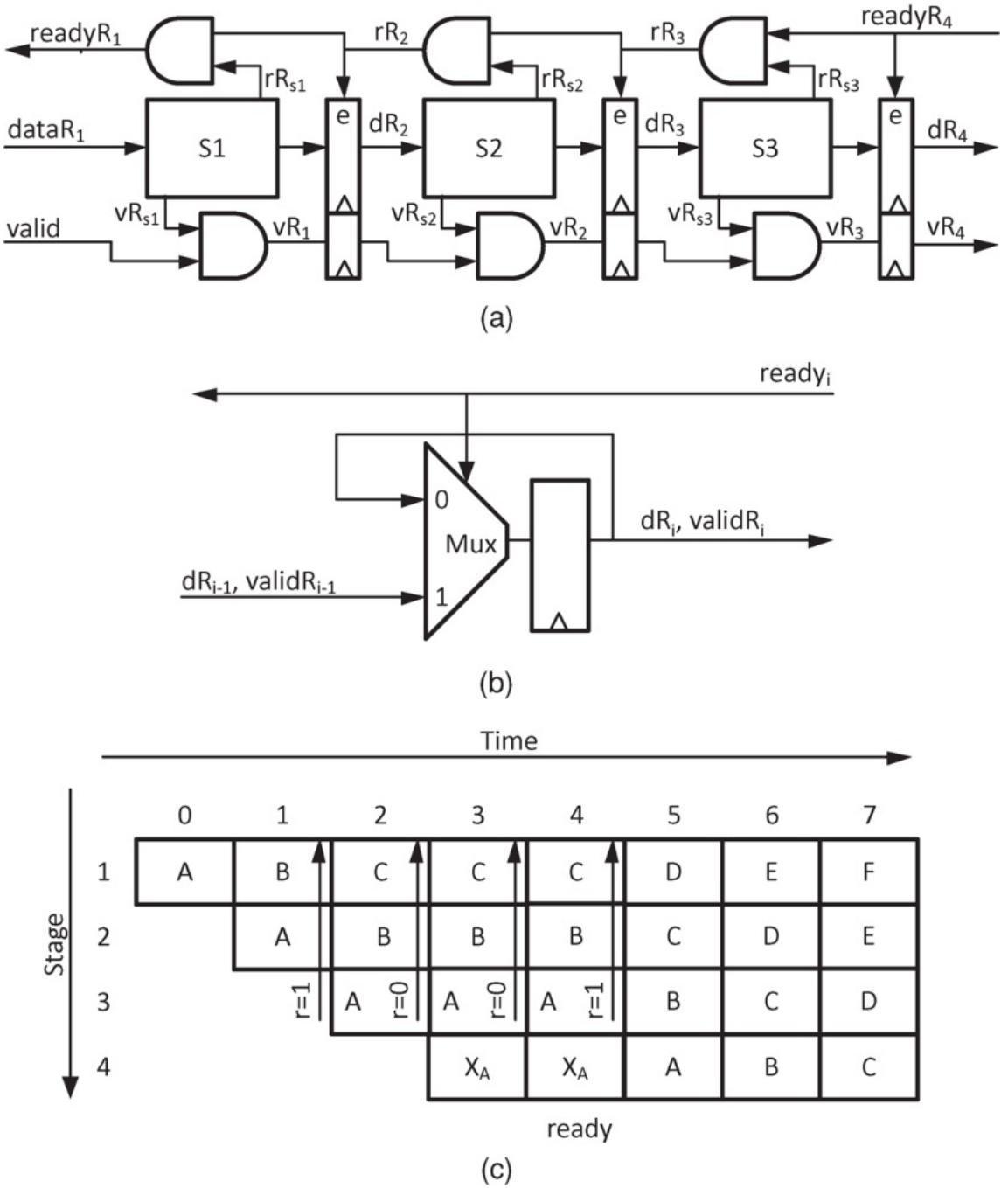
Stalling a *rigid* pipeline

A stall in any stage halts *all* stages upstream of the stall point *instantly* (on the next clock)



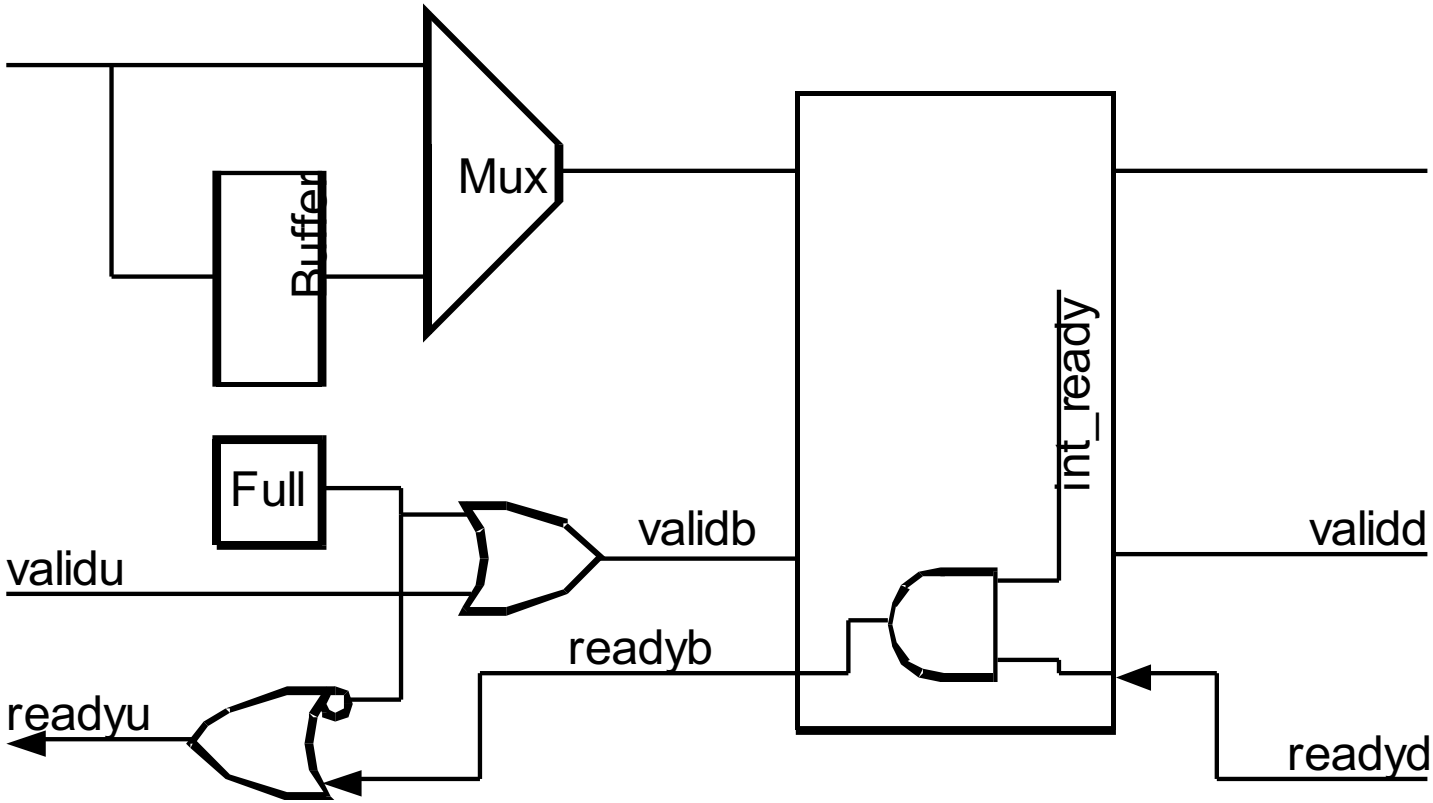
What if we stopped all stages, not just upstream stages? How does the delay of this structure *scale* with the number of stages?

Stalling a *rigid* pipeline

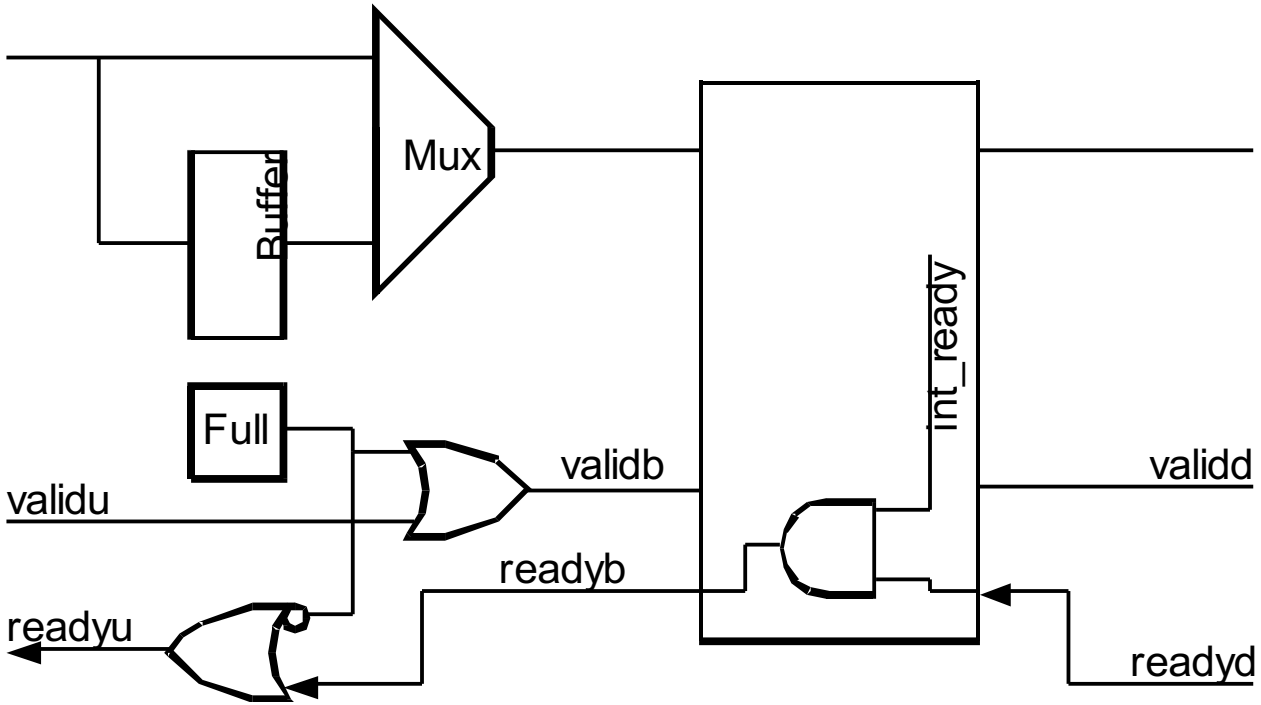


Double Buffer

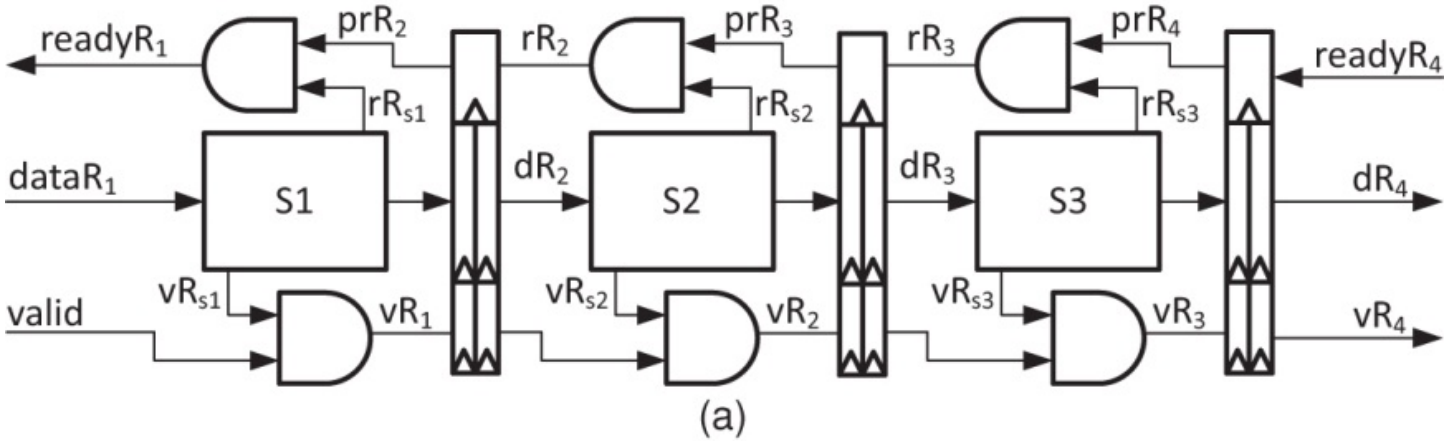
Add an extra buffer to each stage that is filled during the first cycle of a stall.

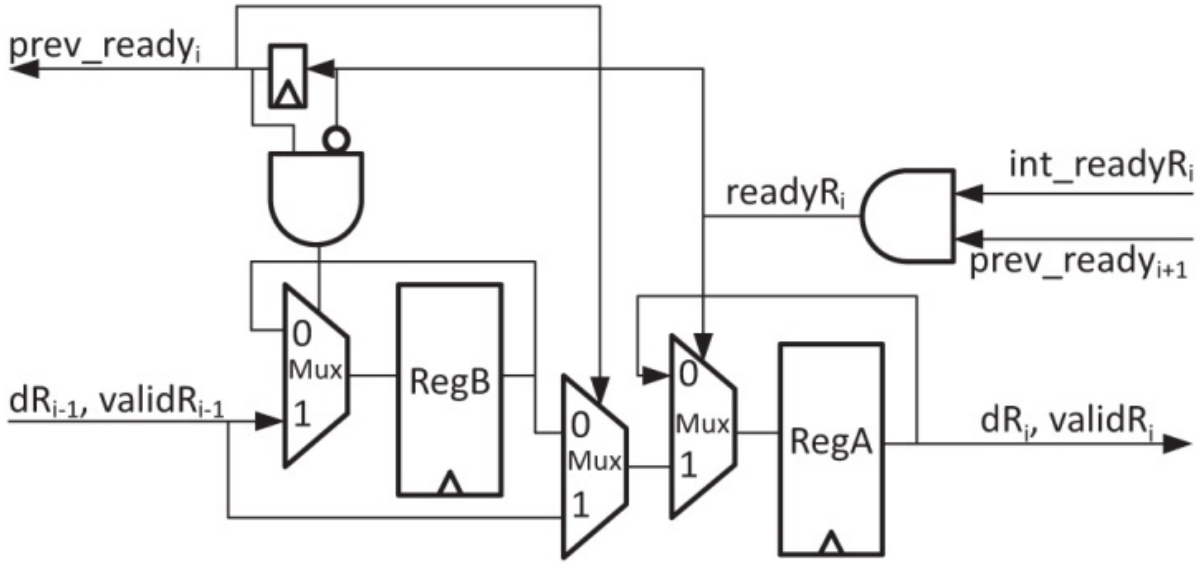


What is the logic equation for "next_full"? "next_buf"? "mux_sel"?

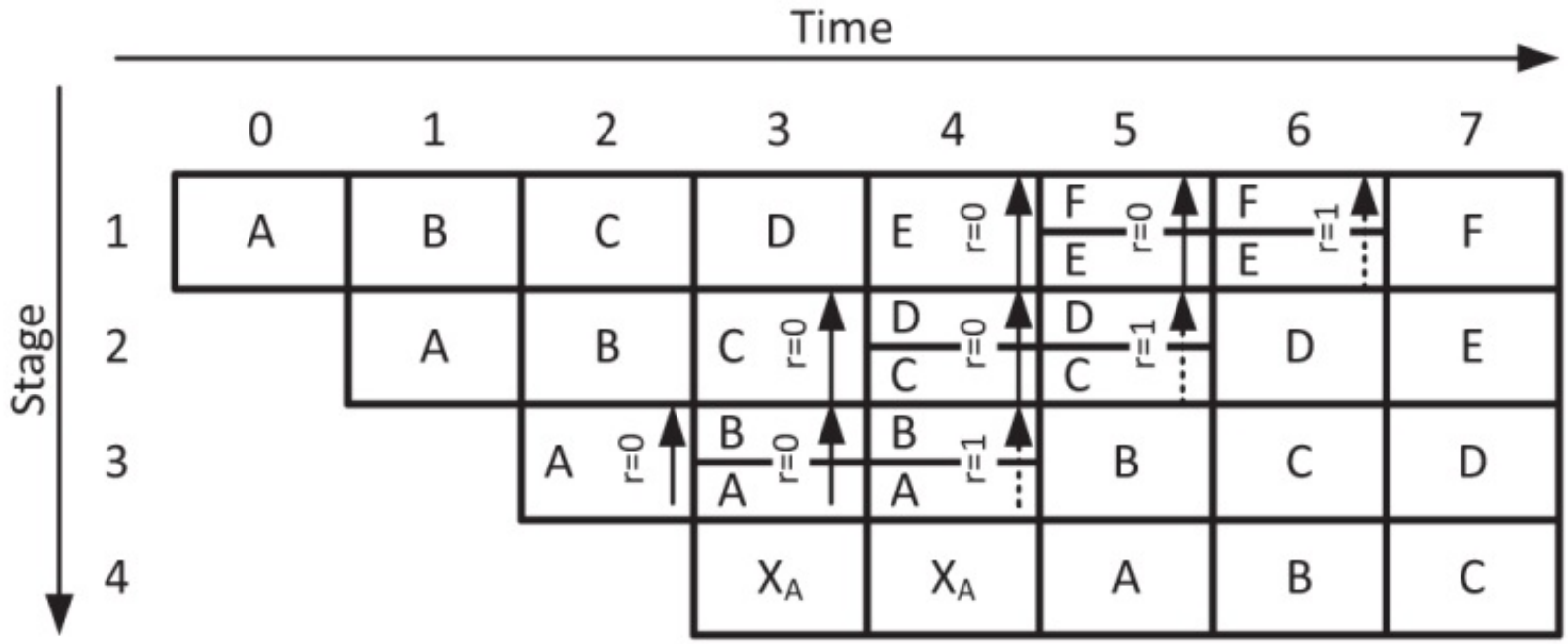


Double Buffering

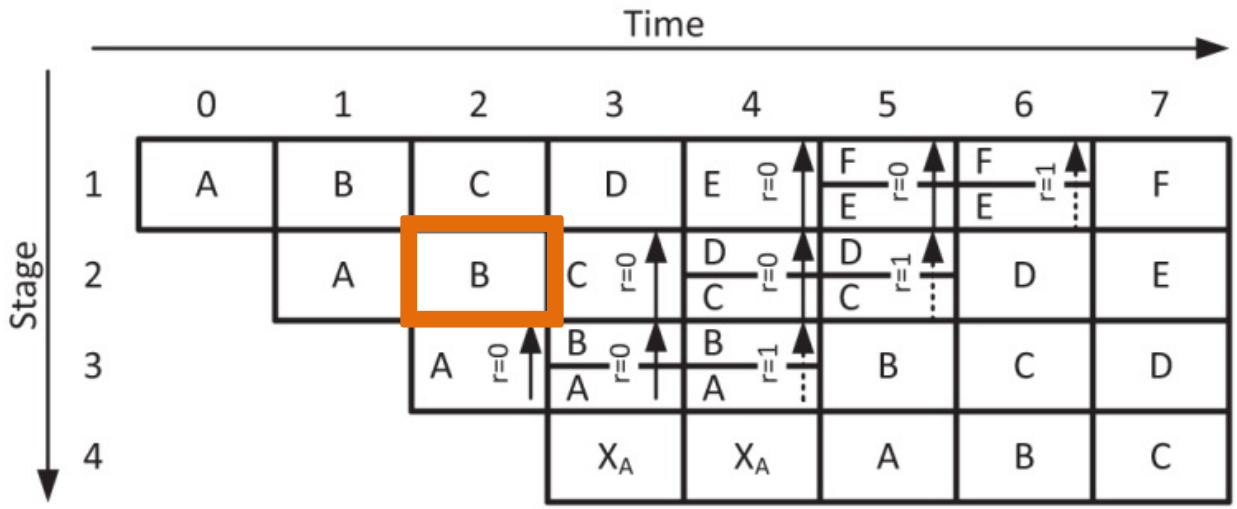




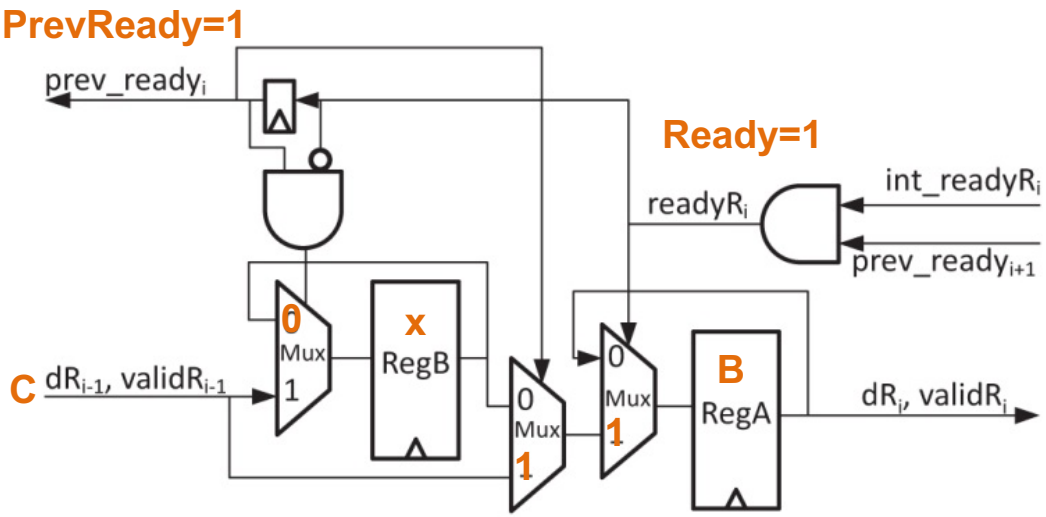
Double Buffer Timing



Double Buffer Timing

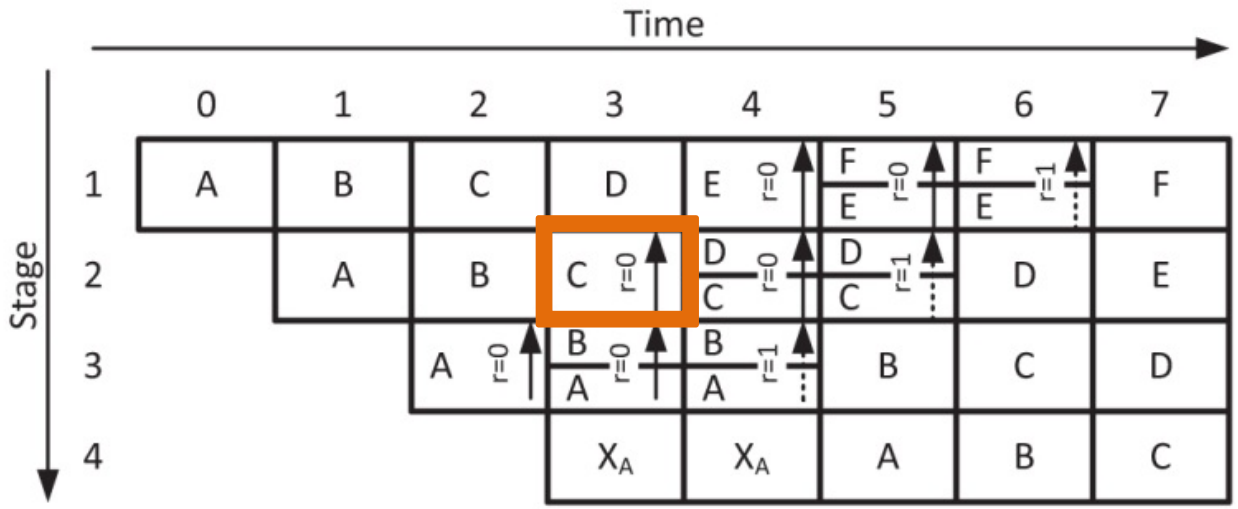


Cycle 2

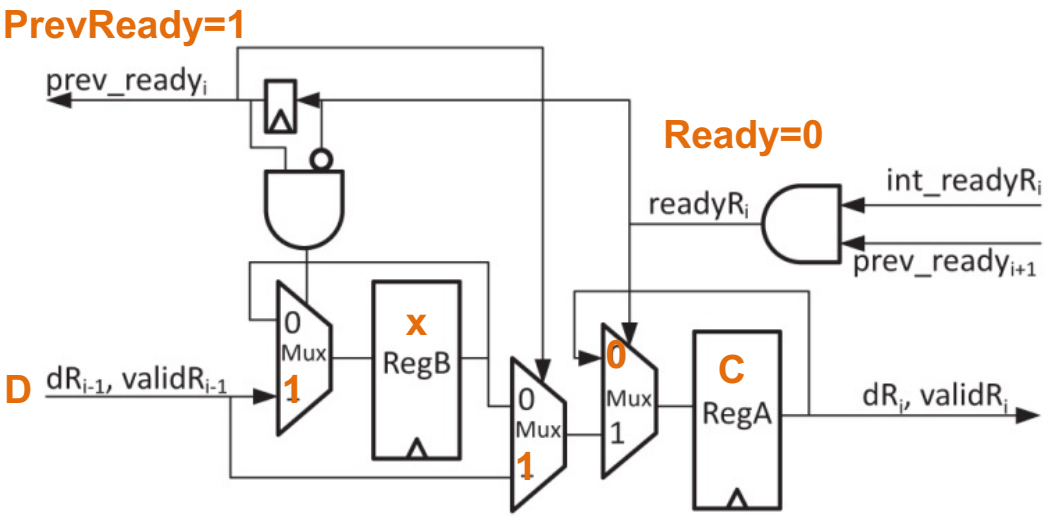


Stage-2 registers

Double Buffer Timing

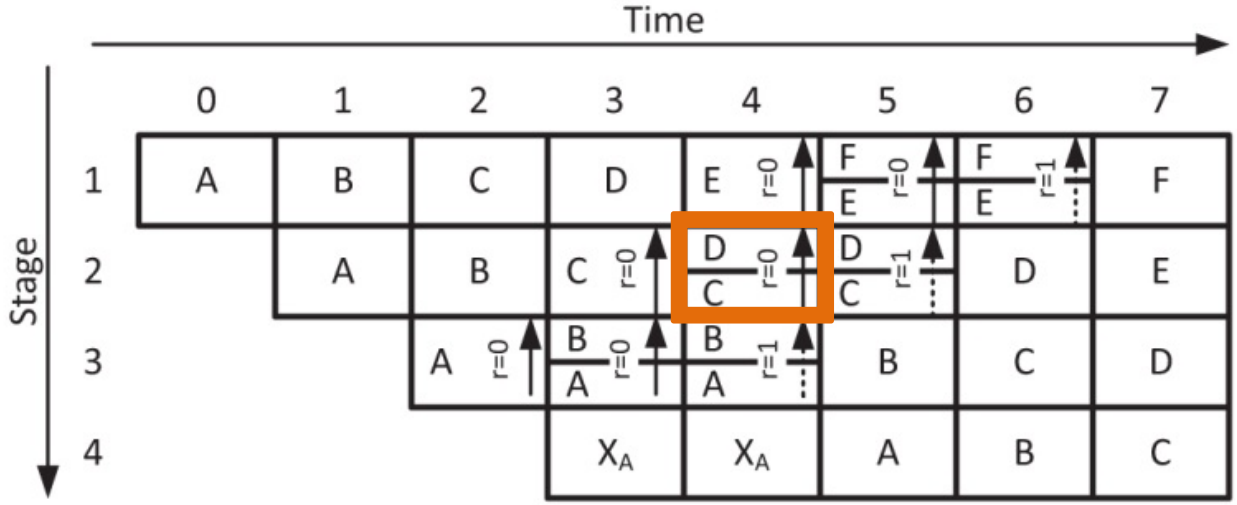


Cycle 3

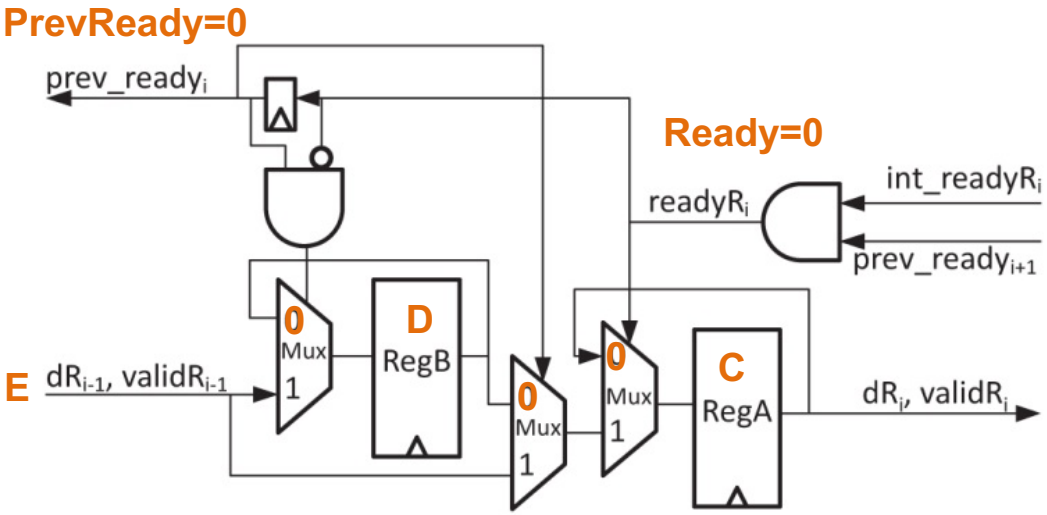


Stage-2 registers

Double Buffer Timing

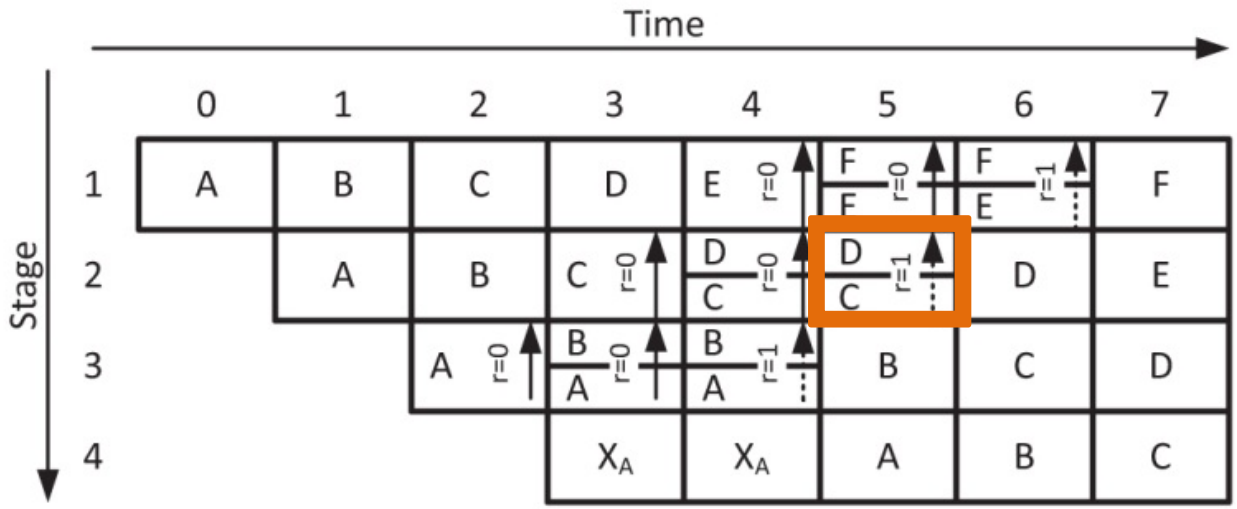


Cycle 4

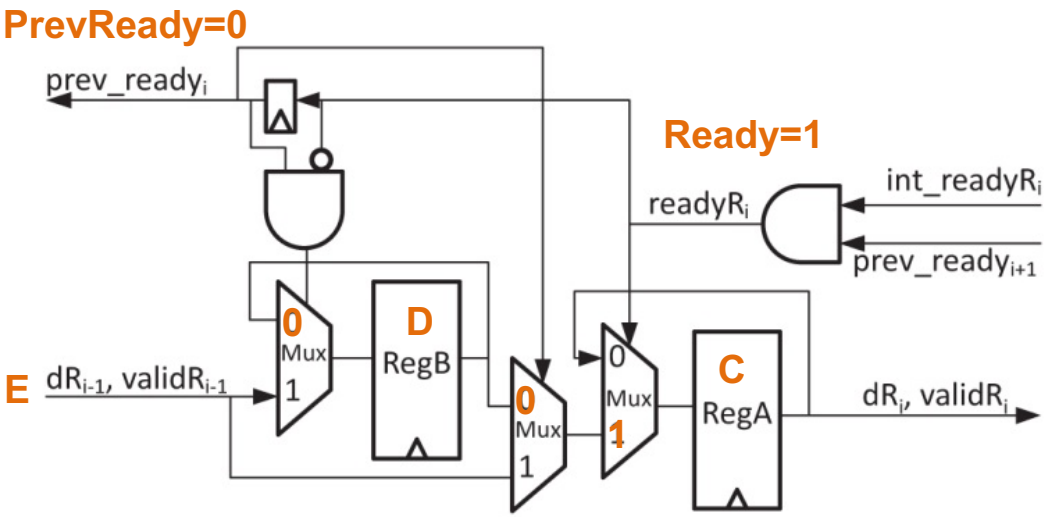


Stage-2 registers

Double Buffer Timing

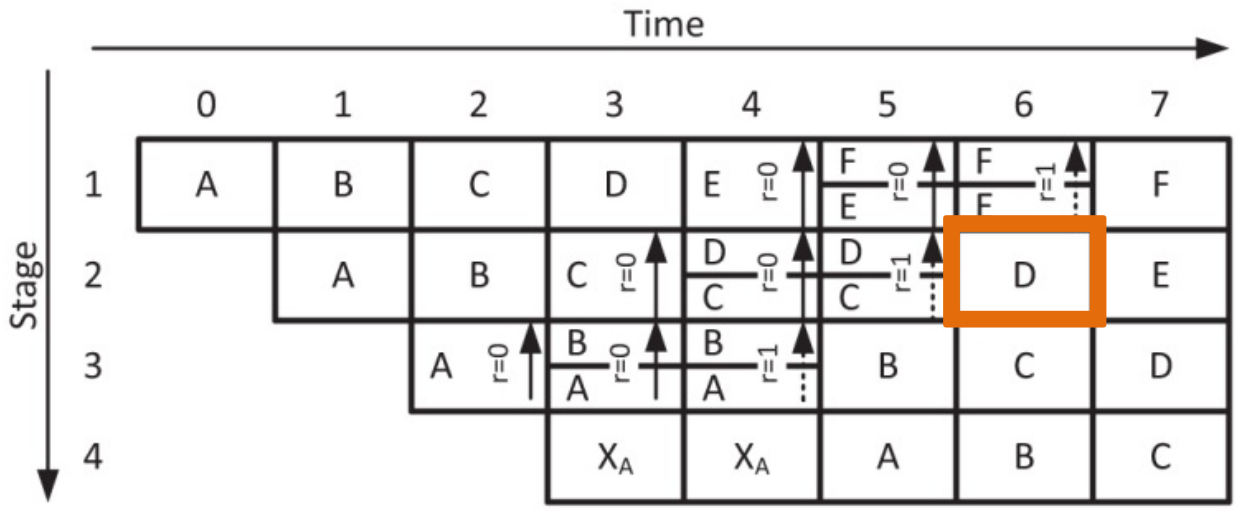


Cycle 5

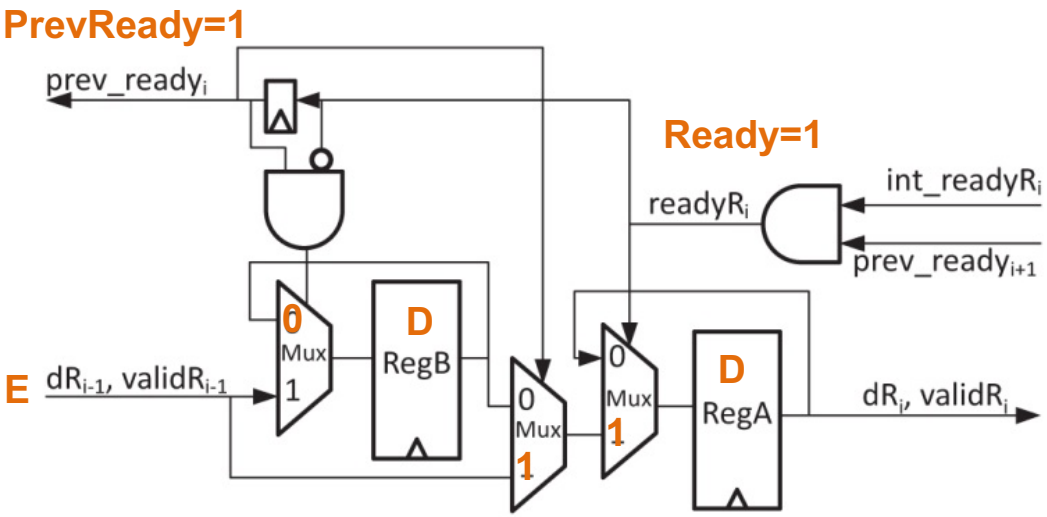


Stage-2 registers

Double Buffer Timing

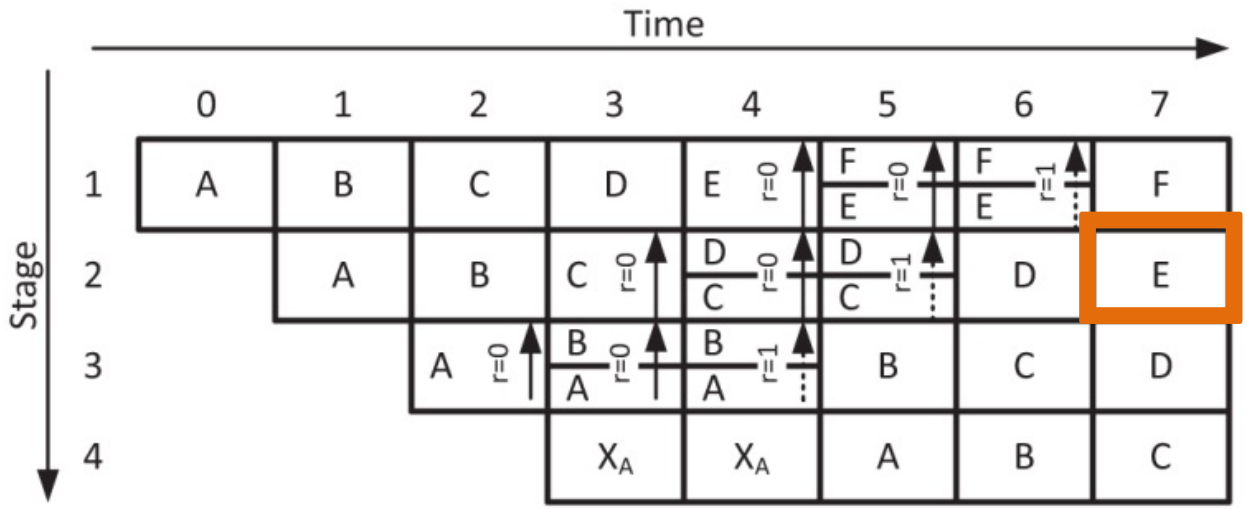


Cycle 6

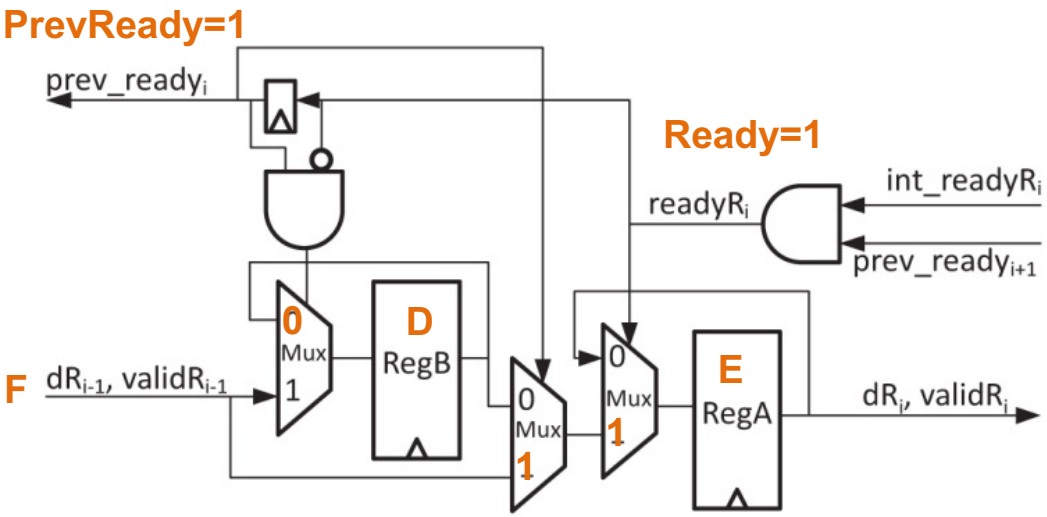


Stage-2 registers

Double Buffer Timing



Cycle 7



Stage-2 registers

Pipeline overview

- Divide large problem into *stages* assembly-line style
- Divide evenly or *load imbalance* will occur
  - Fix by splitting or copying *bottleneck* stage
- Variable load results in *stalls* and *idle* cycles on a rigid pipeline
  - Make pipeline *elastic* by adding FIFOs between key stages
- Rigid pipelines have no extra storage between stages
  - A *stall* on any stage halts all *upstream* stages
  - Hard to stop 100 stages at once
    - Make this scalable with double-buffering

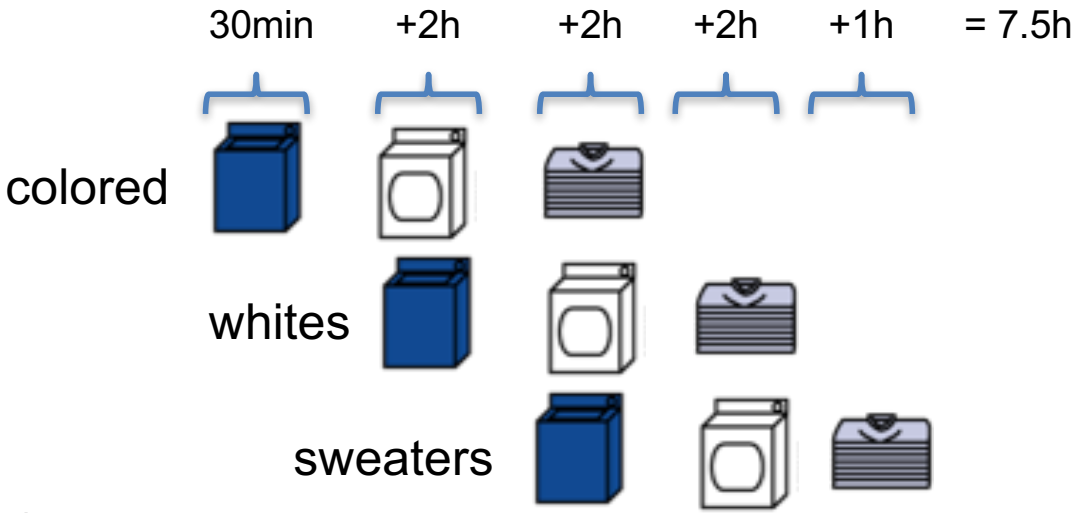
Quiz- 18-1

http://m.socrative.com/student/#joinRoom

room number: **713113**

- If washing takes 30 min, drying takes 2 hours and ironing takes 1 hour, how fast can you prepare 3 portions of cloths if I pipeline?
  - **a. 7.5 h**
  - **b. 4.5 h**
  - **c. 10 h**
  - **d. 10.5 h**

- **Answer:**



Summary

- Pipelining
- Latency & Throughput
- Examples
  - Ripple carry adder
  - Microprocessor
  - Graphics
  - Network processing
- Load balancing
- Stalling

- Reading:
  - Chapter 23
- Next Lecture:
  - Timing, Delay, Power