Lecture 10: Pipelining and Sequential Design

Based on material prepared by prof. Visvesh S. Sathe

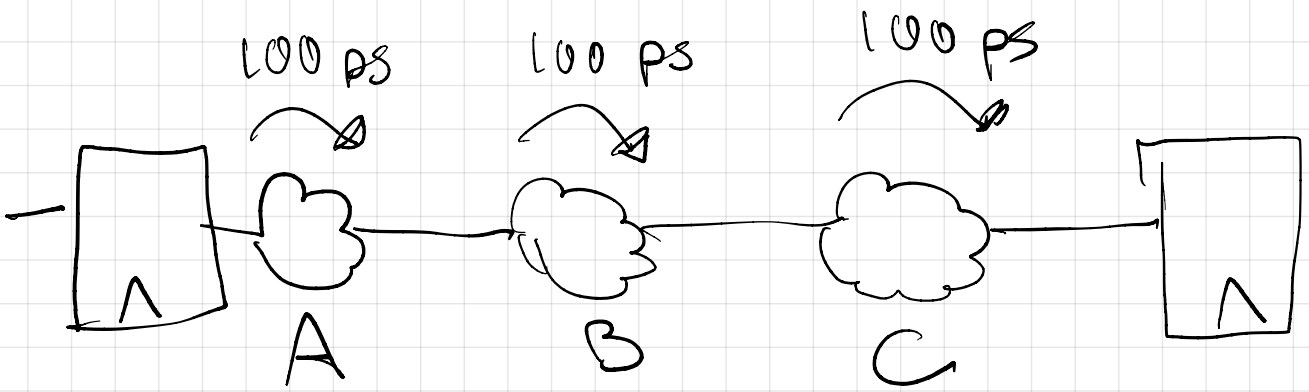
ELECTRICAL & COMPUTER ENGINEERING

Acknowledgements

All class materials (lectures, assignments, etc.) based on material prepared by Prof. Visvesh S. Sathe, and reproduced with his permission

Visvesh S. Sathe
Associate Professor
Georgia Institute of Technology
https://psylab.ece.gatech.edu

UW (2013-2022)
GaTech (2022-present)



100ps

200ps

300ps

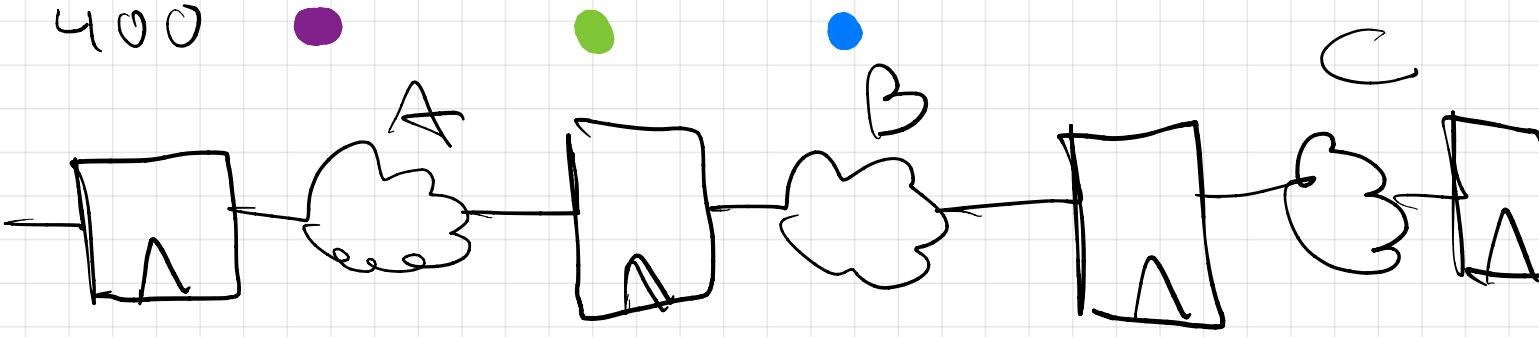
Pipeline

100ps

200

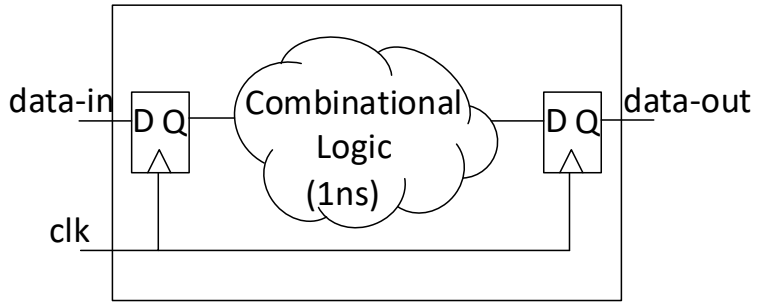
300

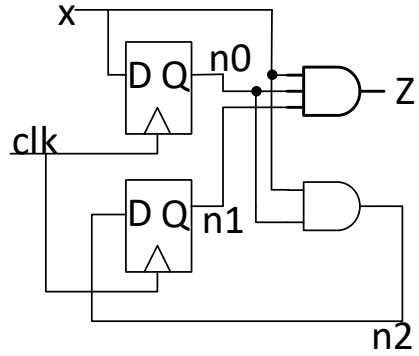
400

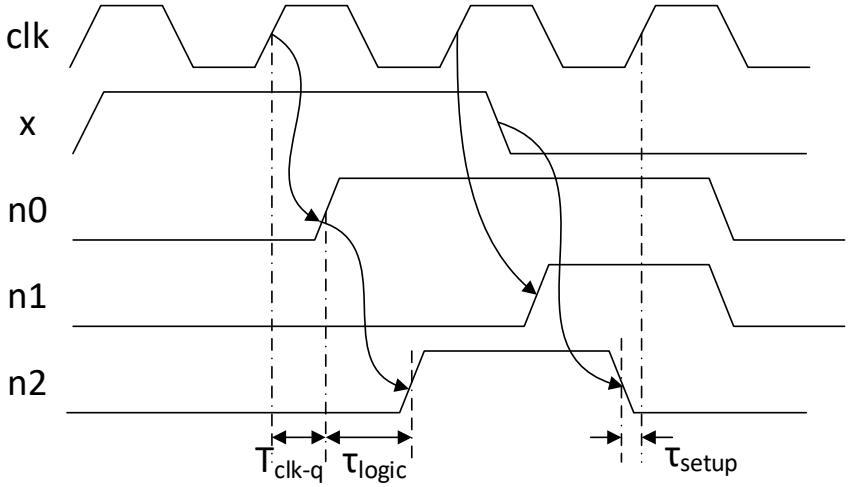


Two Broad Categories

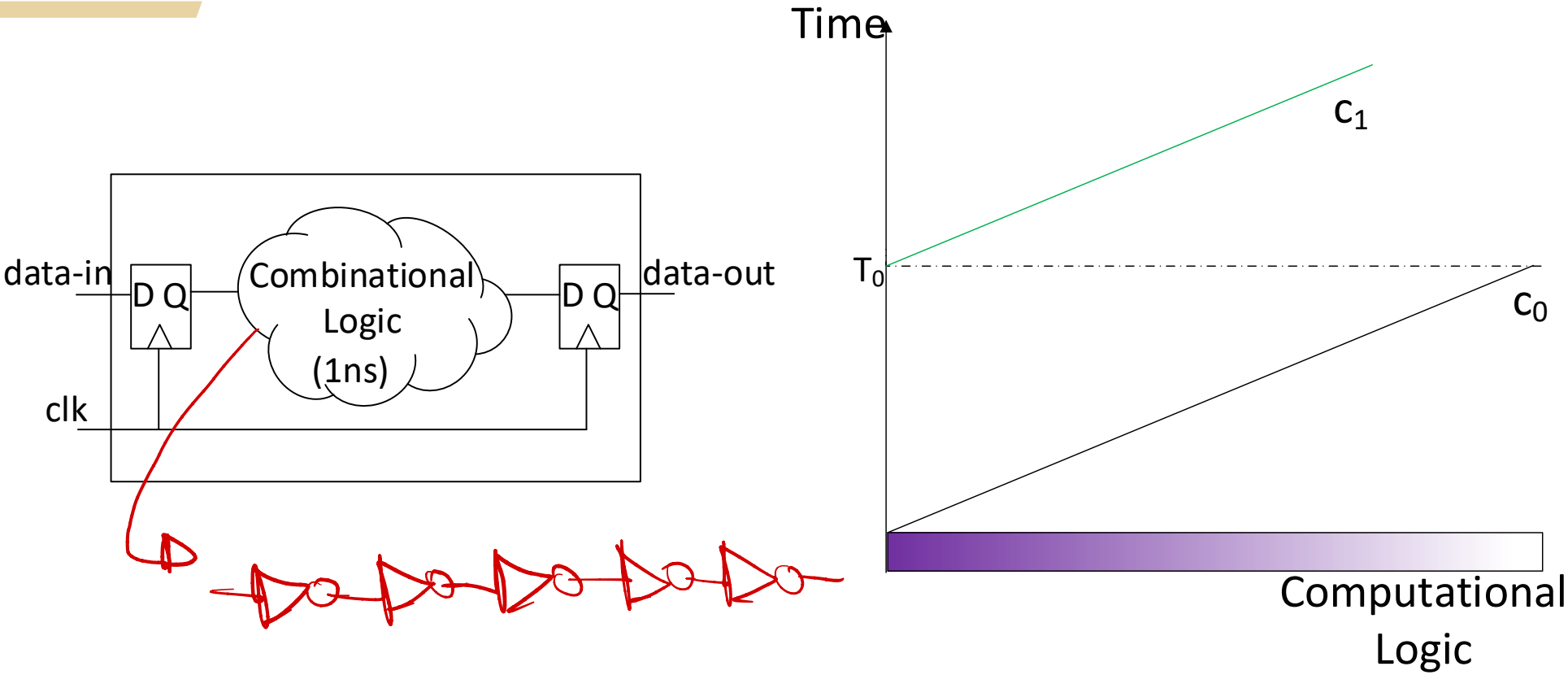
- Combinational logic
  - Output depends only in current inputs
- Sequential design
  - Output depends on a combination of existing and previous inputs
  - Consider a sequential system of 1 output M inputs. Arbitrary dependence on history of past N results leads to equivalent system of inputs.
  - In most practical applications, trace of previous inputs influences system “makeup” (or state), which is a stored quantity.
  - Output depends on current inputs and current state





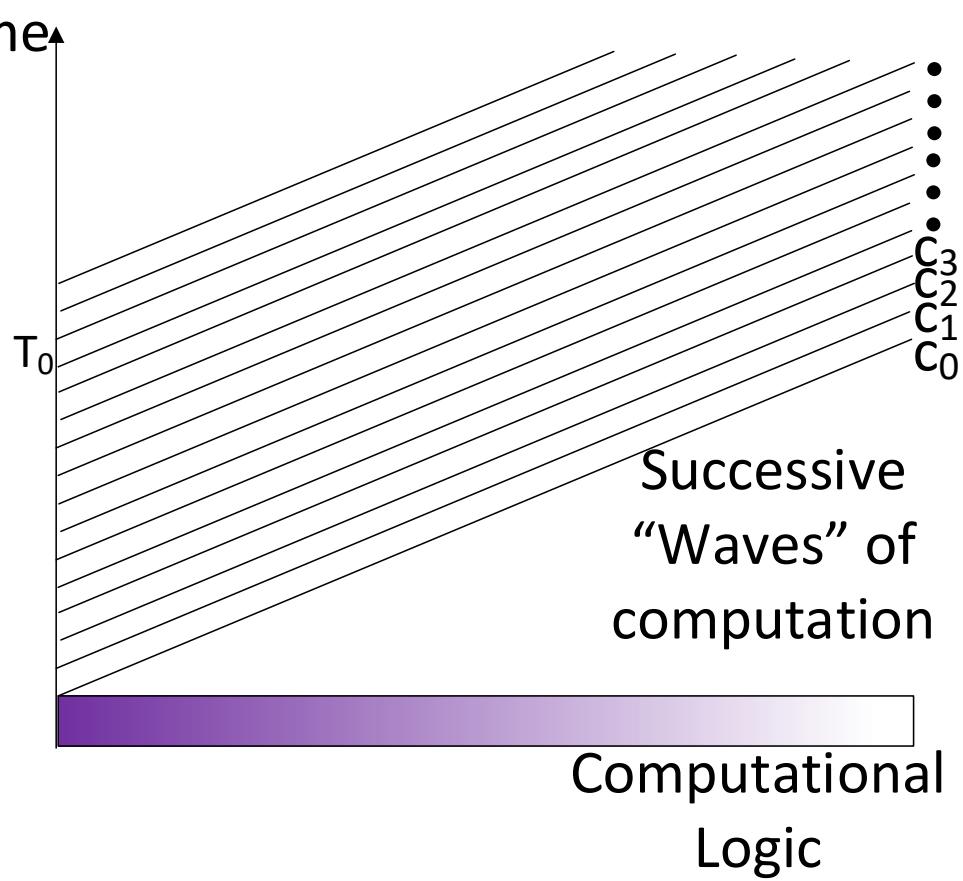


Pipelines In a Little More Detail….



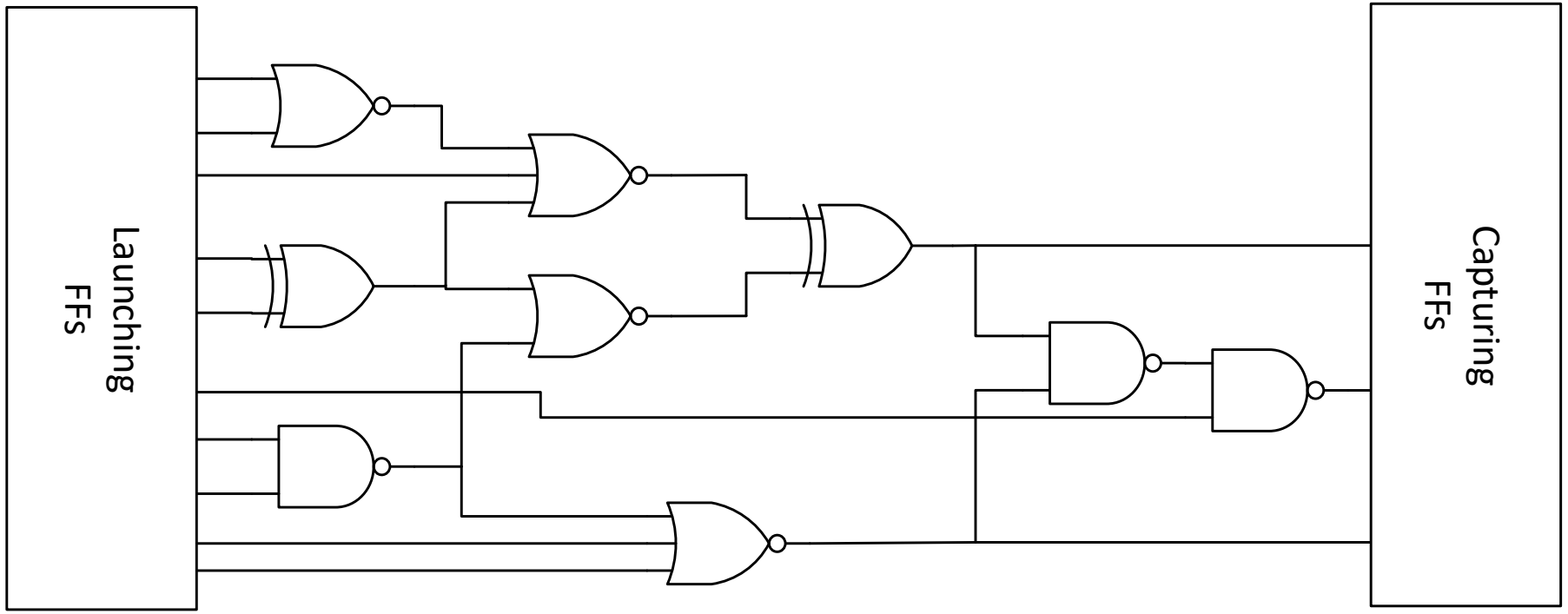
- Synchronous designs operate at a rate determined by a clock signal
  - Clocks control flip-flops to capture and propagate data
- If we wait for computation completion before launching new data:
  - Duration (or **latency**) of logic evaluation sets processing rate (throughput)
  - Latency L ➔ Throughput T=1/L

Why Wait?



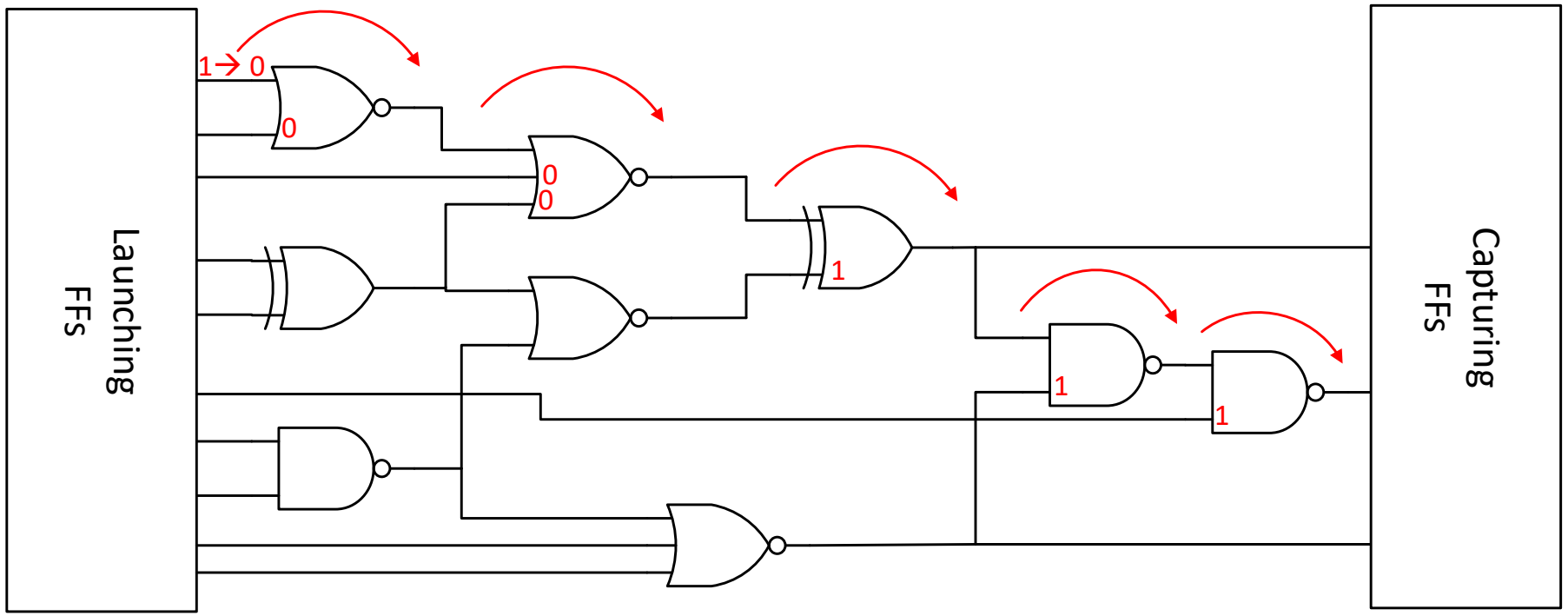
- Don't wait for a computation to end before starting the next one
  - Process data in "waves" within logic
  - Use a flip-flop to capture and store data (avoid waves from colliding)
  - Wave pipelining

Quick Detour: Max and Min-delays



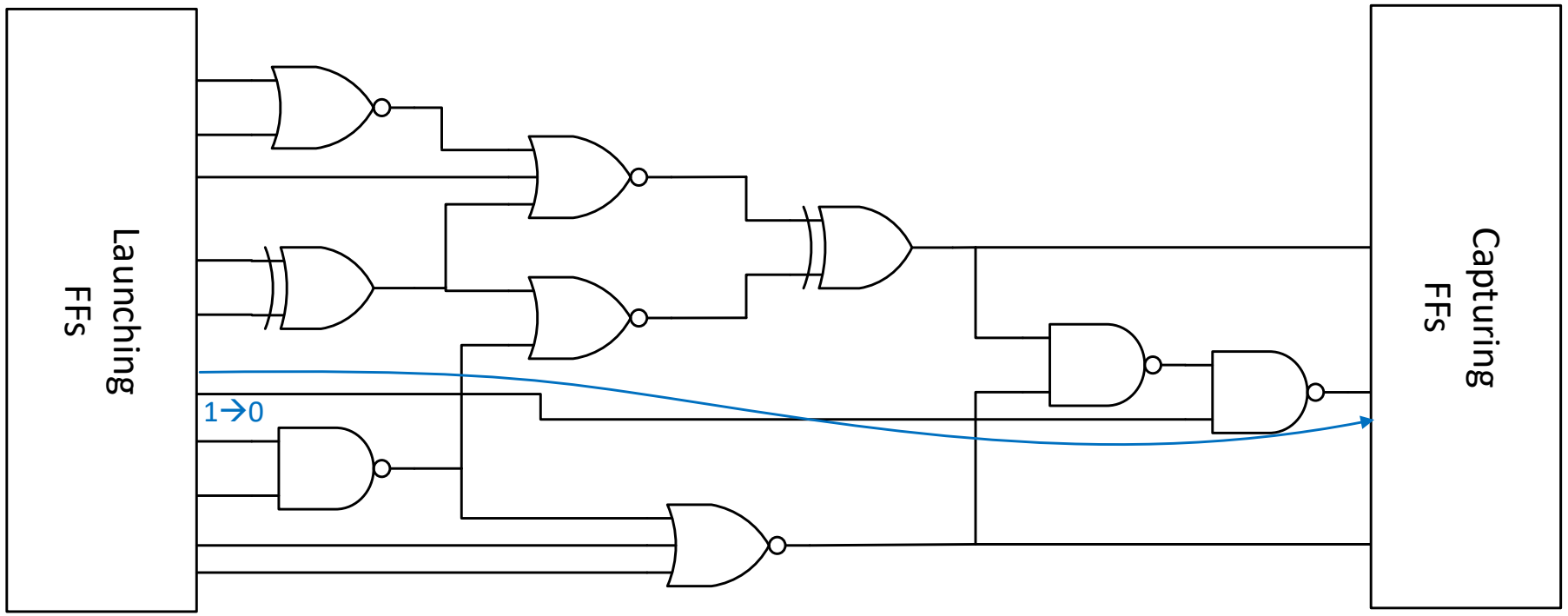
- Combinational logic evaluation latency occurs over a range of time
  - Latency (Duration) of the computation depends on input data selected
  - **Not every input combination excites the critical path**
  - Max-delay: Data inputs traverse critical path of the logic
  - Min-delay : Data inputs traverse shortest path (in a delay sense) of logic

Quick Detour: Max and Min-delays



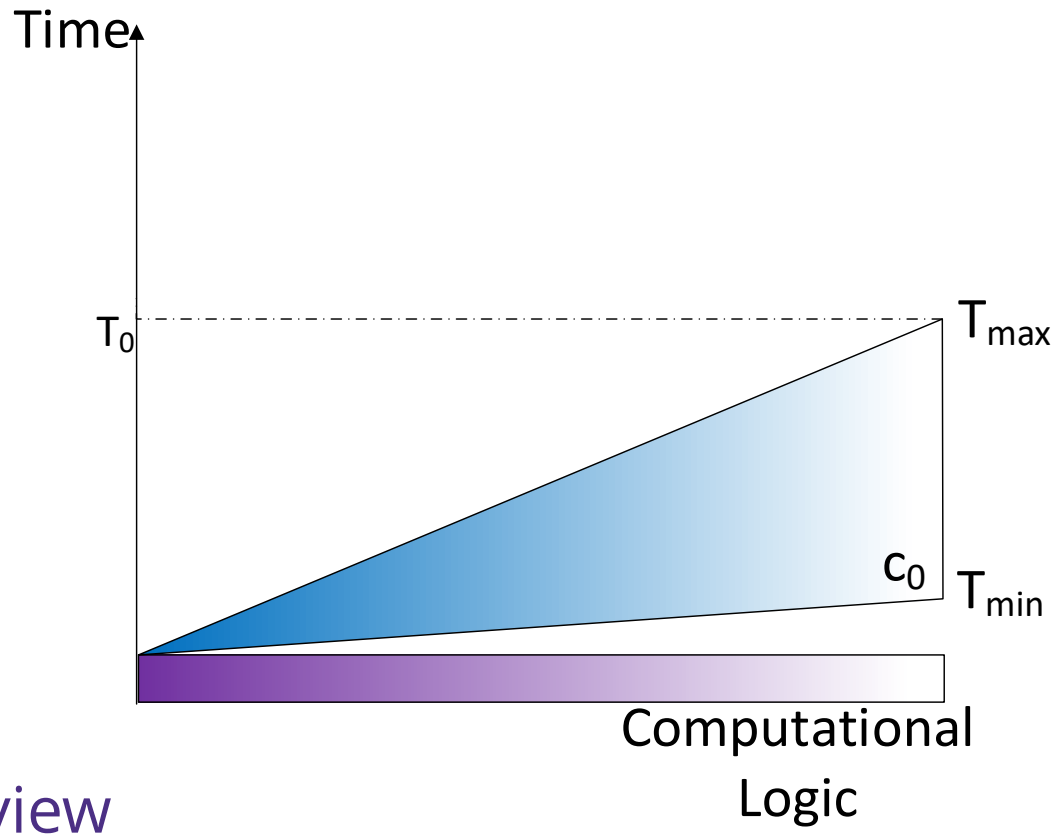
- Combinational logic evaluation latency occurs over a range of time
  - Latency (Duration) of the computation depends on input data selected
  - **Not every input combination excites the critical path**
  - Max-delay: Data inputs traverse critical path of the logic
  - Min-delay : Data inputs traverse shortest path (in a delay sense) of logic

Quick Detour: Max and Min-delays



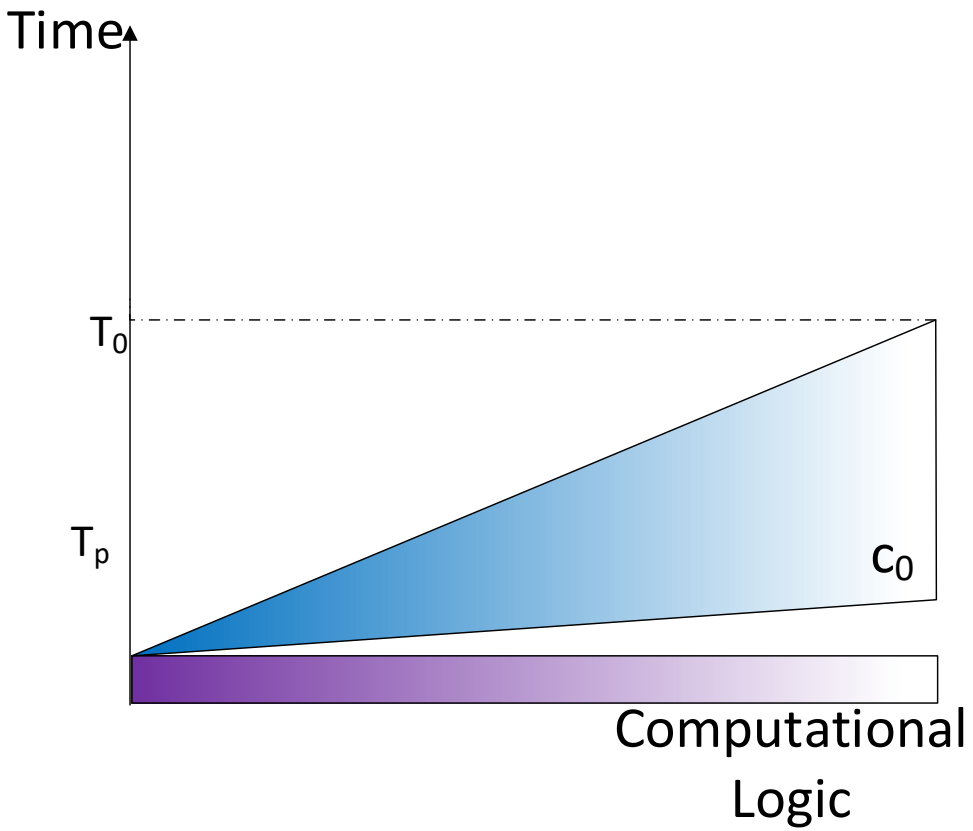
- Combinational logic evaluation latency occurs over a range of time
  - Latency (Duration) of the computation depends on input data selected
  - **Not every input combination excites the critical path**
  - Max-delay: Data inputs traverse critical path of the logic
  - Min-delay : Data inputs traverse shortest path (in a delay sense) of logic

Delay Distribution in Logic



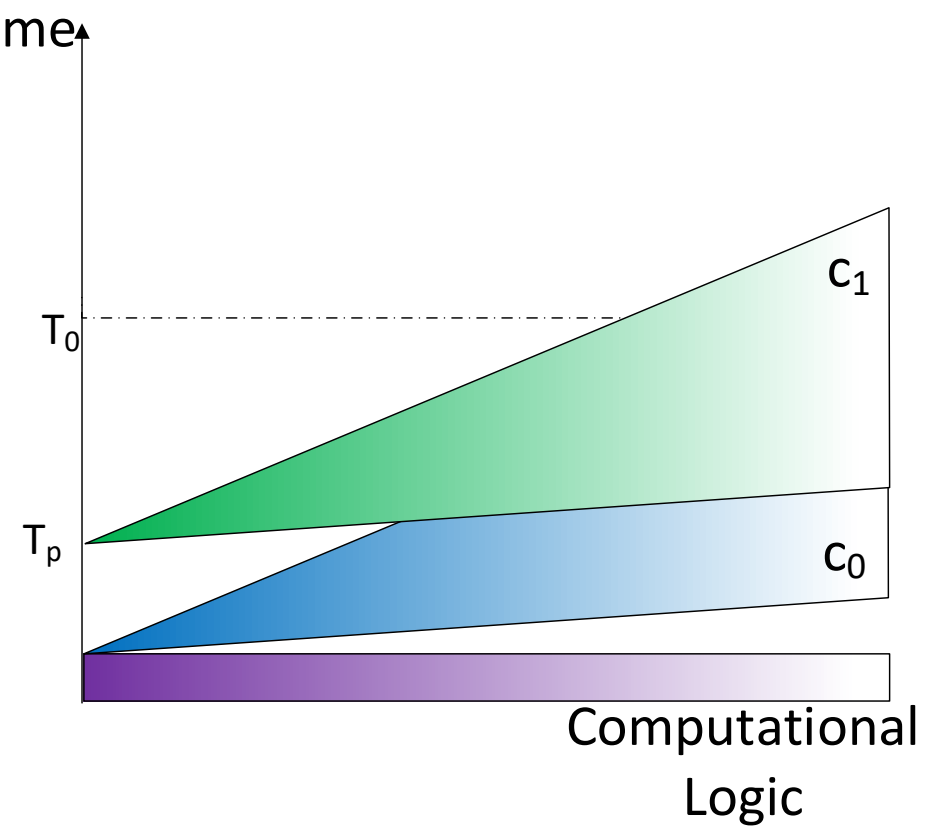
- A more realistic view
  - Logic progression through the combinational block occurs over a range of time
  - Latency depends on the nature of inputs computed
  - Typically, critical paths are excited with a low probability **(but must still be designed for!)**
  - **Note: This figure (showing continuous progression) is still a simplification!**

Wave Collision



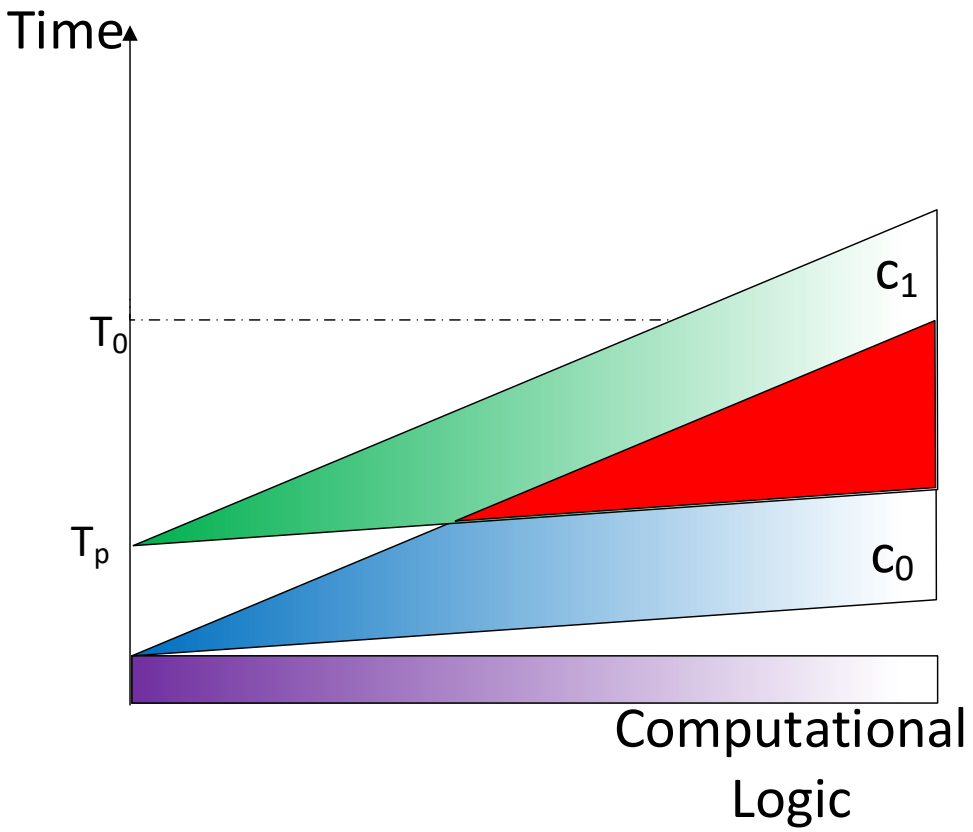
- Prematurely launching new wave before previous wave is “sufficiently far along” risks wave collision (and therefore failure)
- Max/Min delay spread for **each** output node must be considered

Wave Collision



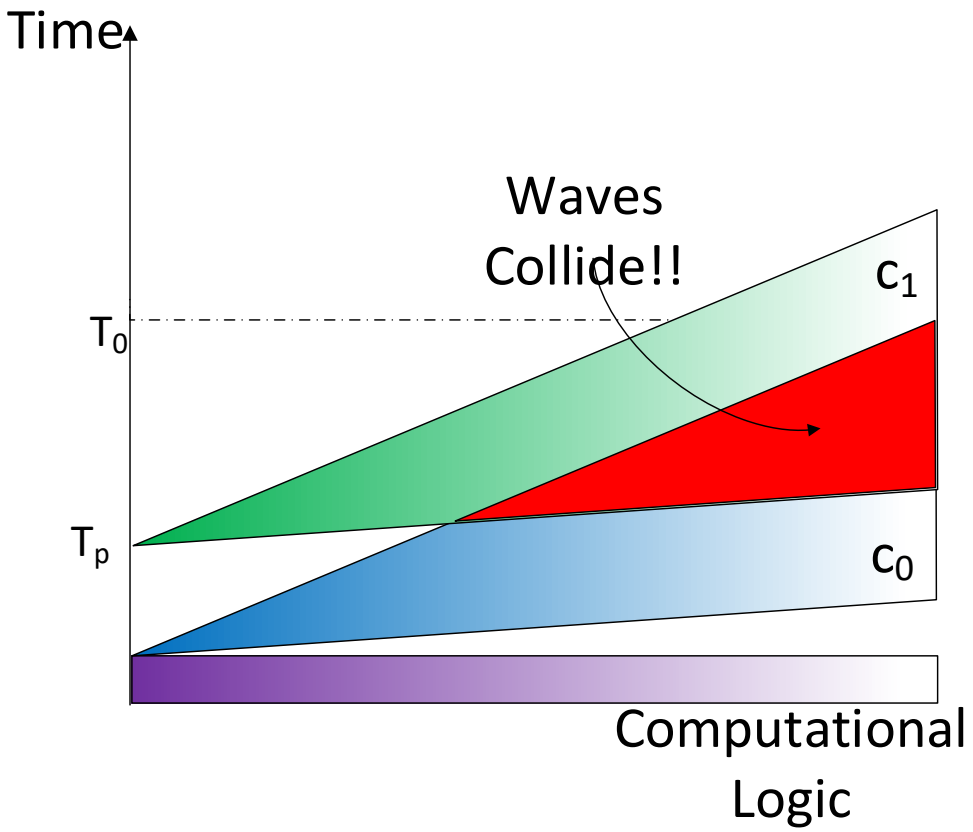
- Prematurely launching new wave before previous wave is “sufficiently far along” risks wave collision (and therefore failure)
- Max/Min delay spread for **each** output node must be considered

Wave Collision



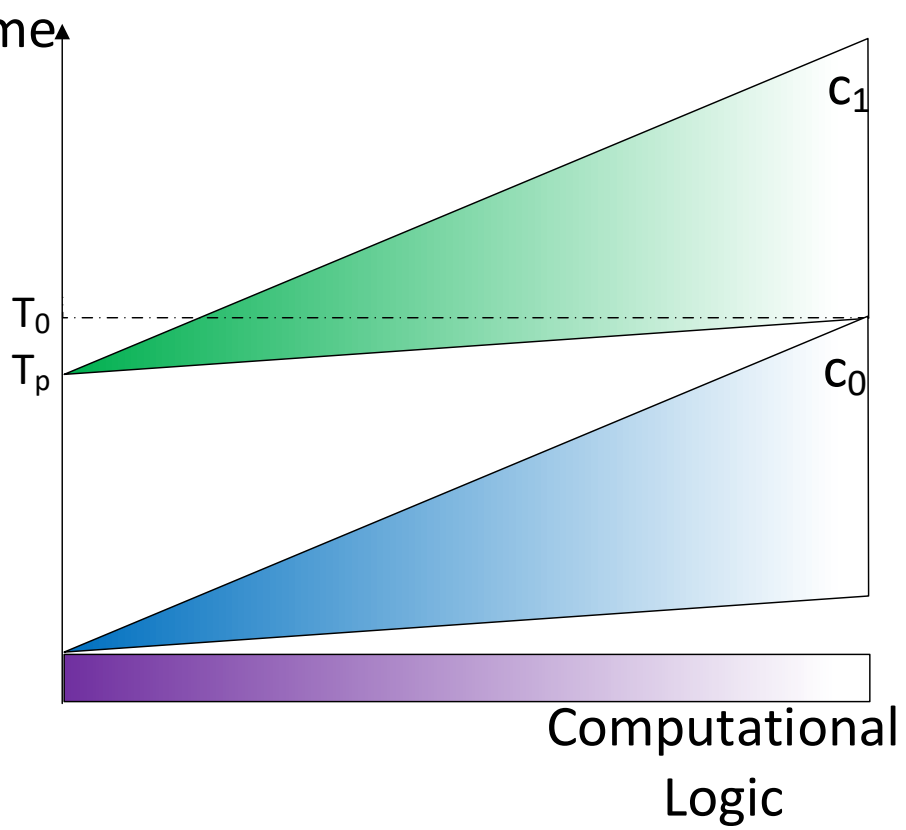
- Prematurely launching new wave before previous wave is “sufficiently far along” risks wave collision (and therefore failure)
- Max/Min delay spread for **each** output node must be considered

Wave Collision



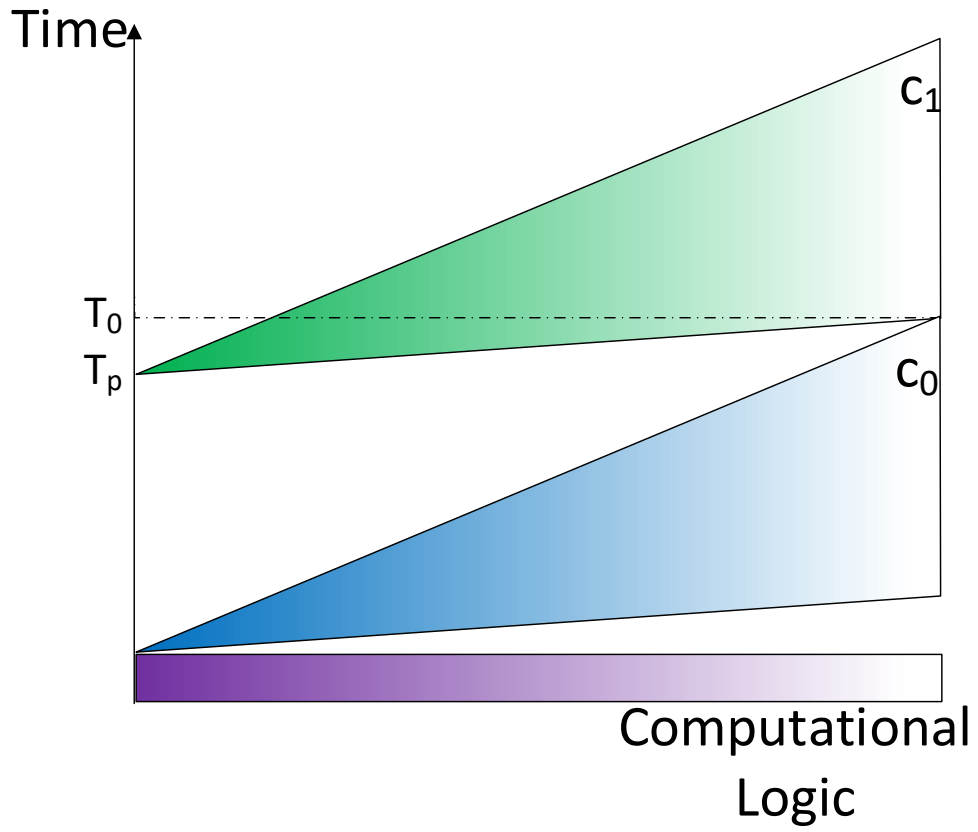
- Prematurely launching new wave before previous wave is “sufficiently far along” risks wave collision (and therefore failure)
- Max/Min delay spread for **each** output node must be considered

Maximum Throughput



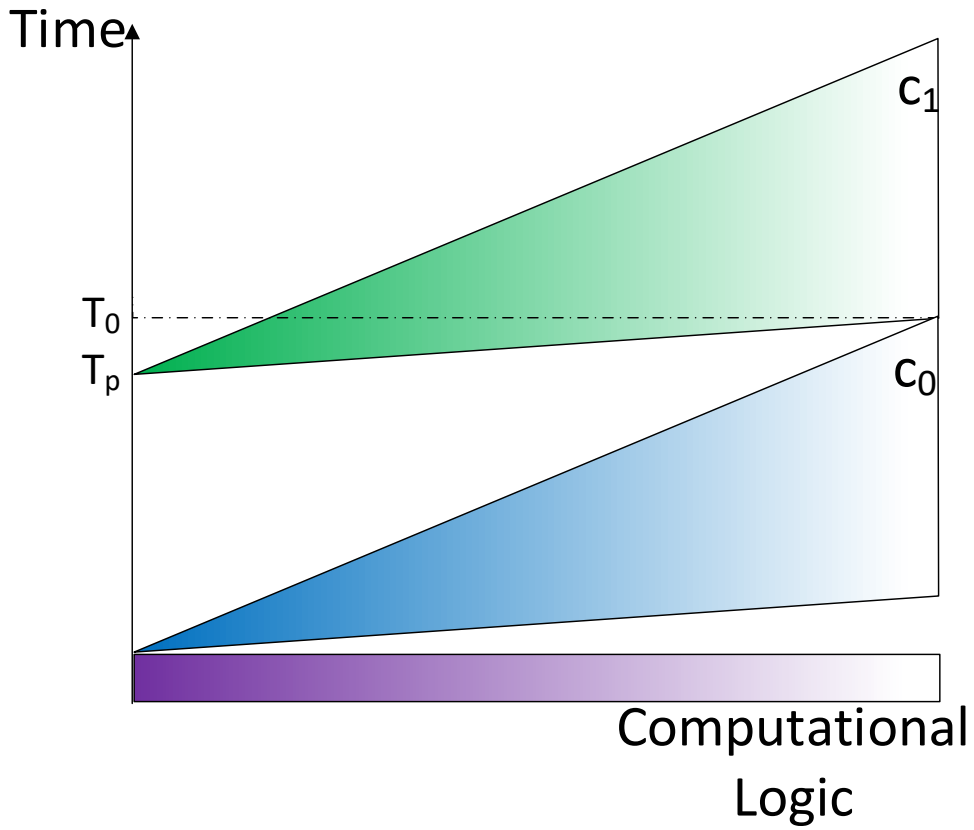
- Need to wait for at least before launching new data

Maximum Throughput



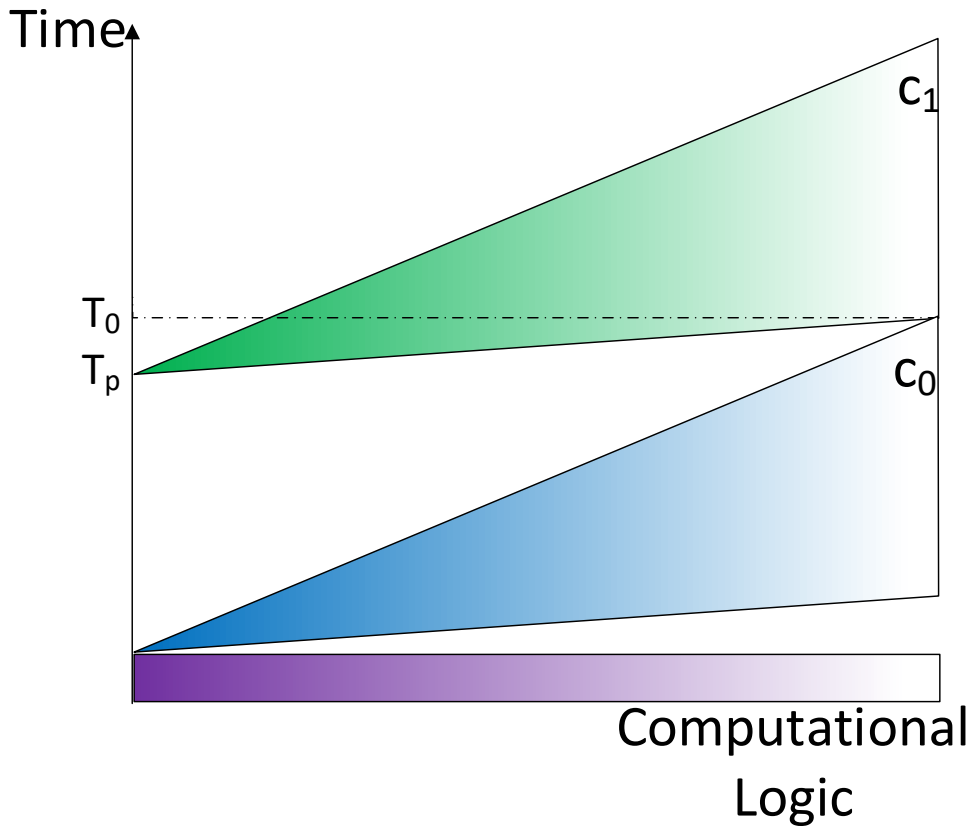
- Need to wait for at least T_{max}-T_{min} before launching new data

Maximum Throughput



- Need to wait for at least T_{max}-T_{min} before launching new data
- Max, min latency **for any point** sets the absolute highest achievable throughput

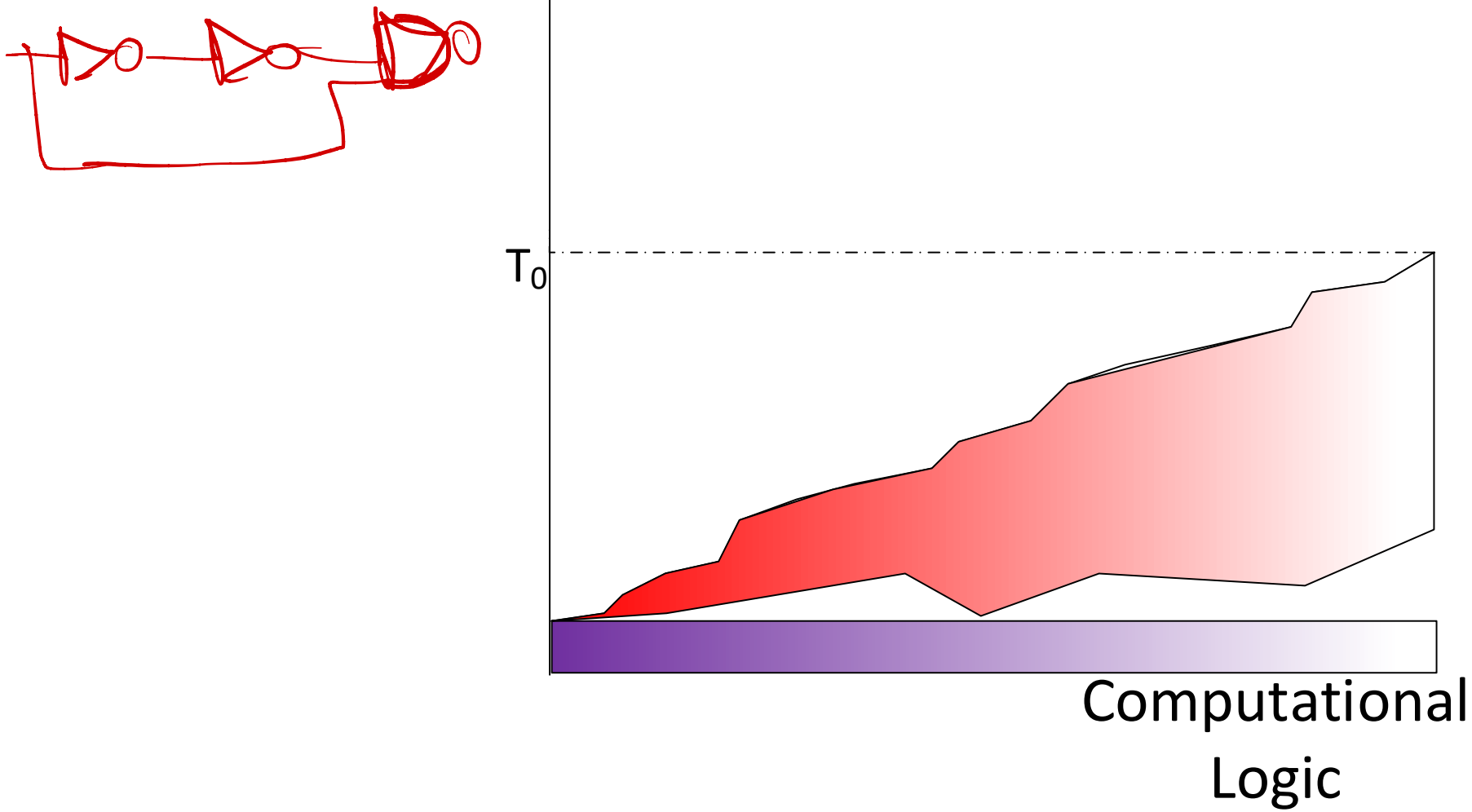
Maximum Throughput



OK. T_{max}-T_{min}, but where?

- Need to wait for at least T_{max}-T_{min} before launching new data
- Max, min latency **for any point** sets the absolute highest achievable throughput

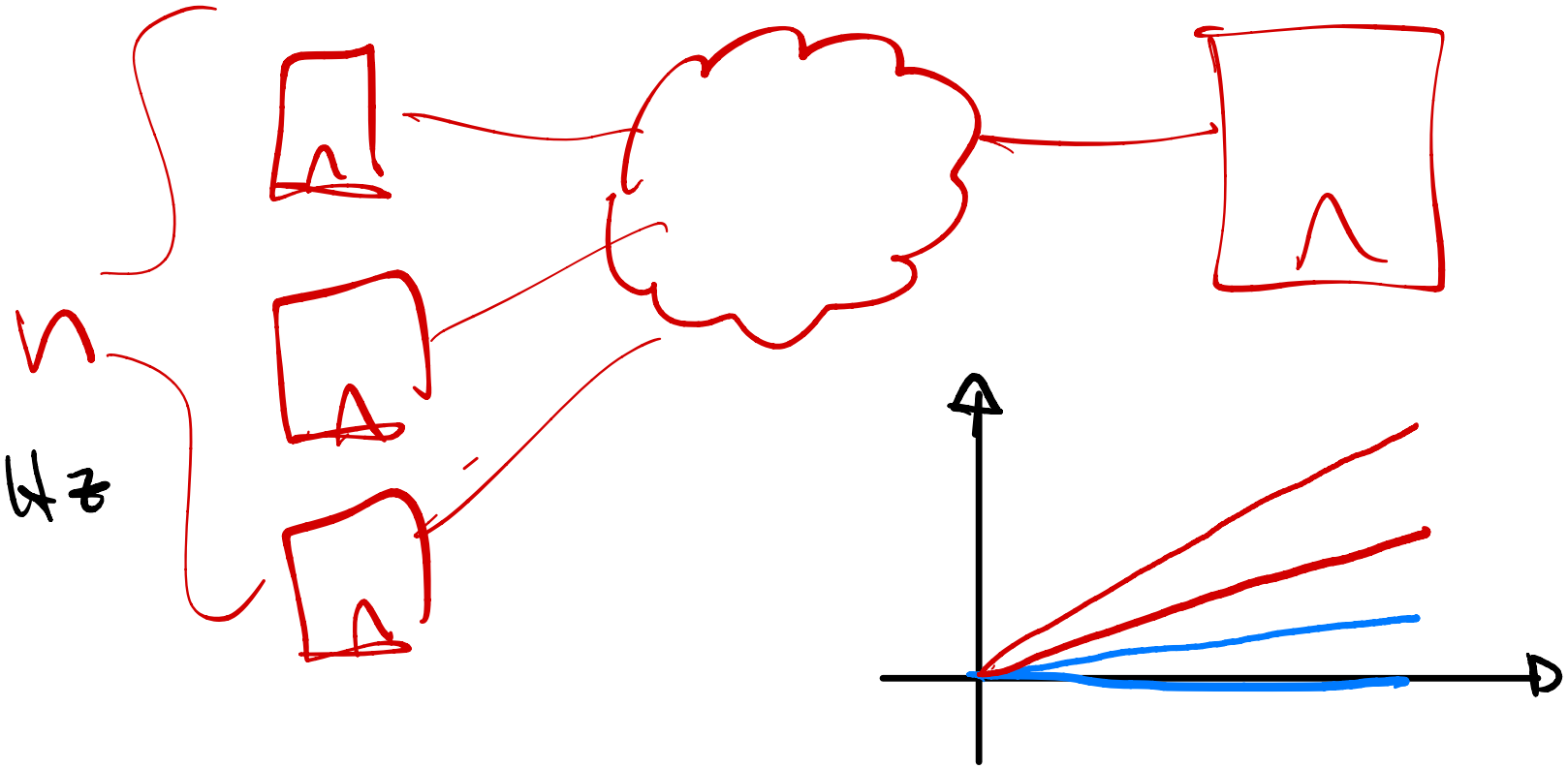
Typical Max and Min Delay Profile



Question

- Given: A combinational logic with n inputs and 1 output. T_{max} range (1.5ns-2.5ns) , T_{min}(0,0.5ns). What is the highest achievable throughput using this “wave-pipelining” approach.
  - 3.33 GHz
  - 0.67 GHz
  - 0.4 GHz
  - 0.5 GHz

$\frac{1}{2.5ns} = 0.4\,GHz$

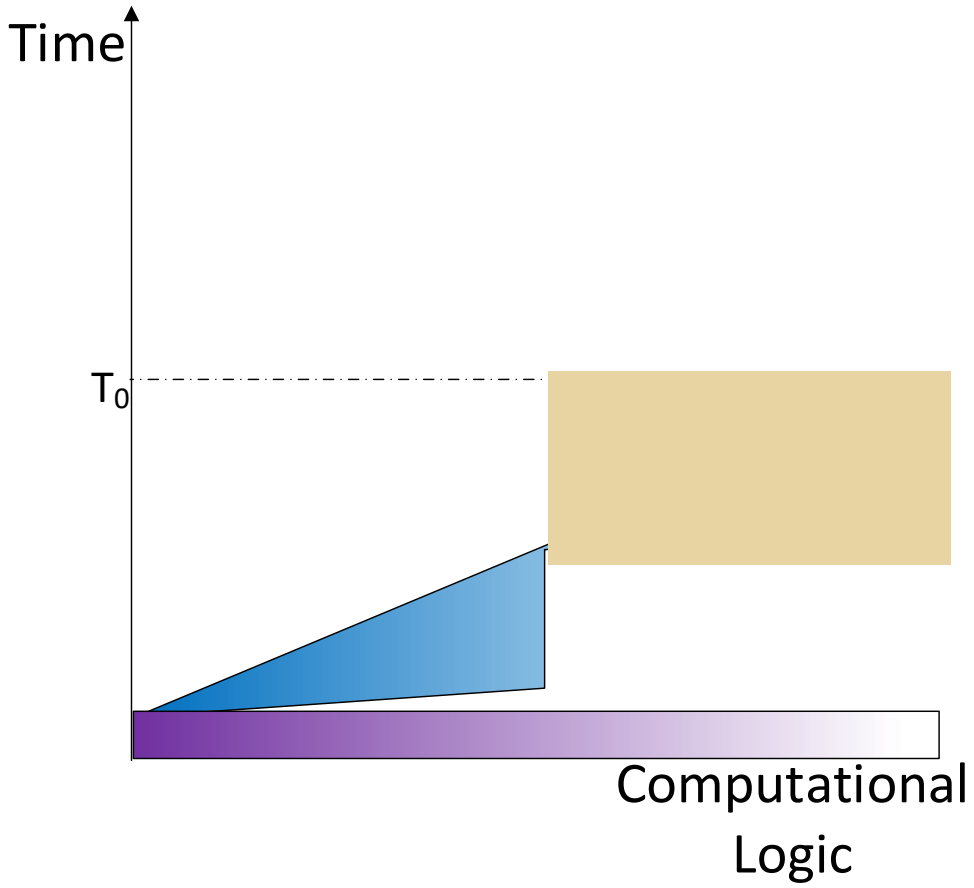


Question

- Given: A combinational logic with n inputs and 1 output. T_{max} range (1.5ns-2.5ns) , T_{min}(0,0.5ns). What is the highest achievable throughput using this “wave-pipelining” approach.
  - 3.33 GHz
  - 0.67 GHz
  - 0.4 GHz
  - 0.5 GHz

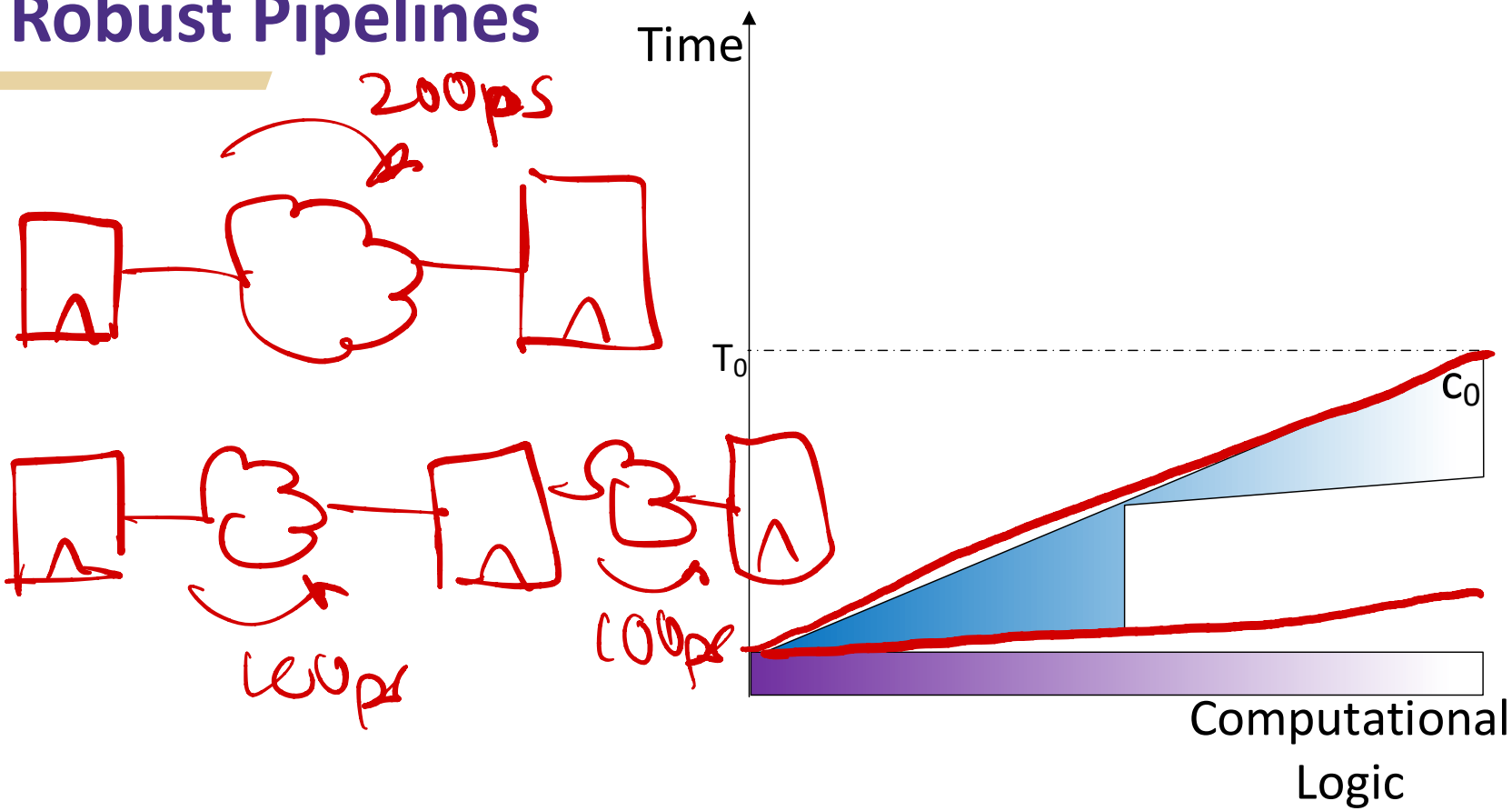
- Can a general statement be made about computational latency, and the achievable speedup using this approach?

Robust Pipelines



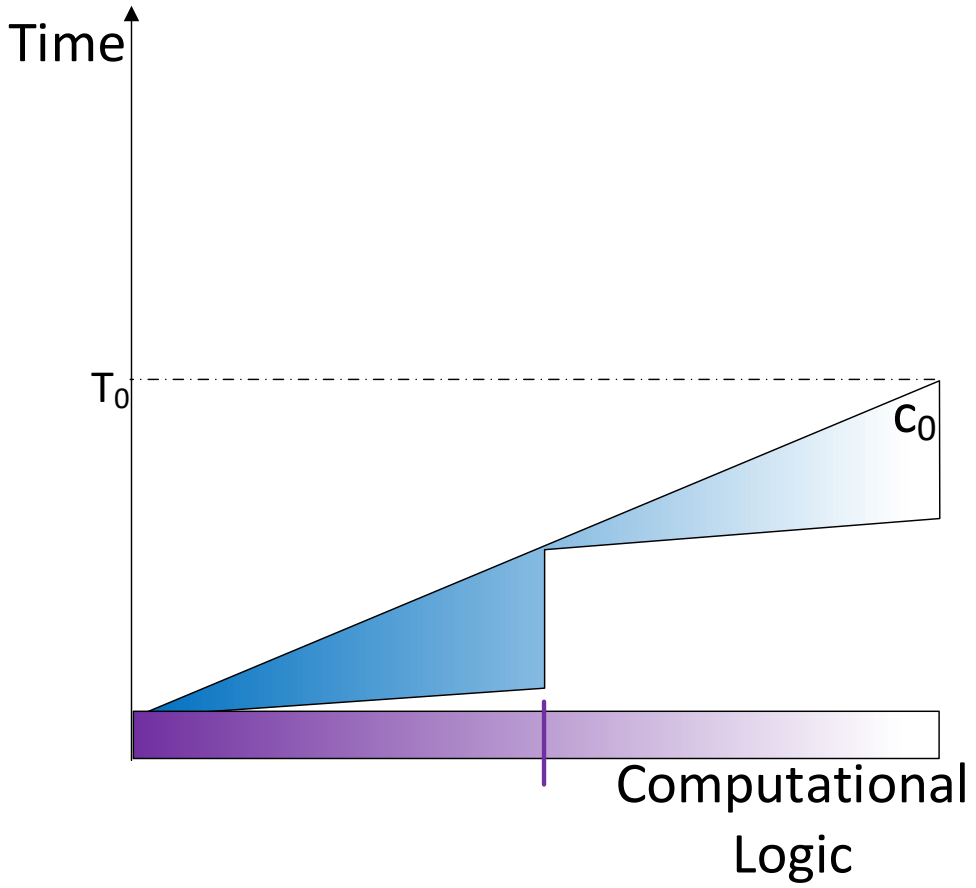
- FF-based pipelines (2-stage example)
  - Insert FFs at points in the logic corresponding to T/2 peak delay

Robust Pipelines



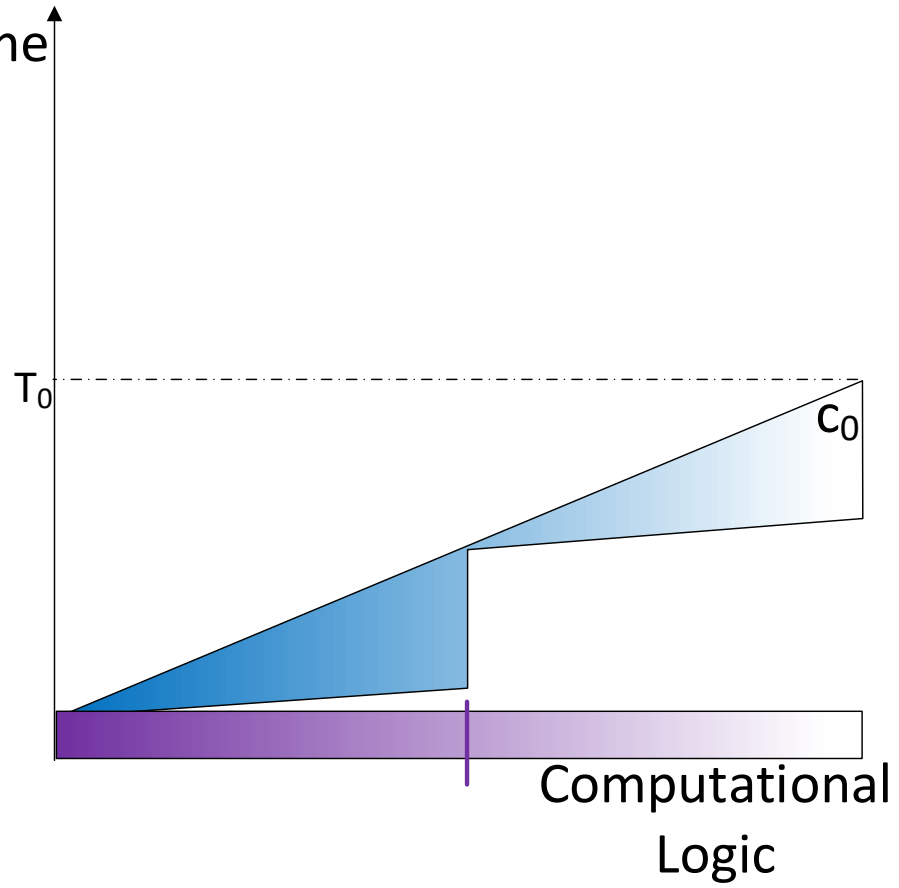
- FF-based pipelines (2-stage example)
  - Insert FFs at points in the logic corresponding to T/2 peak delay

Robust Pipelines



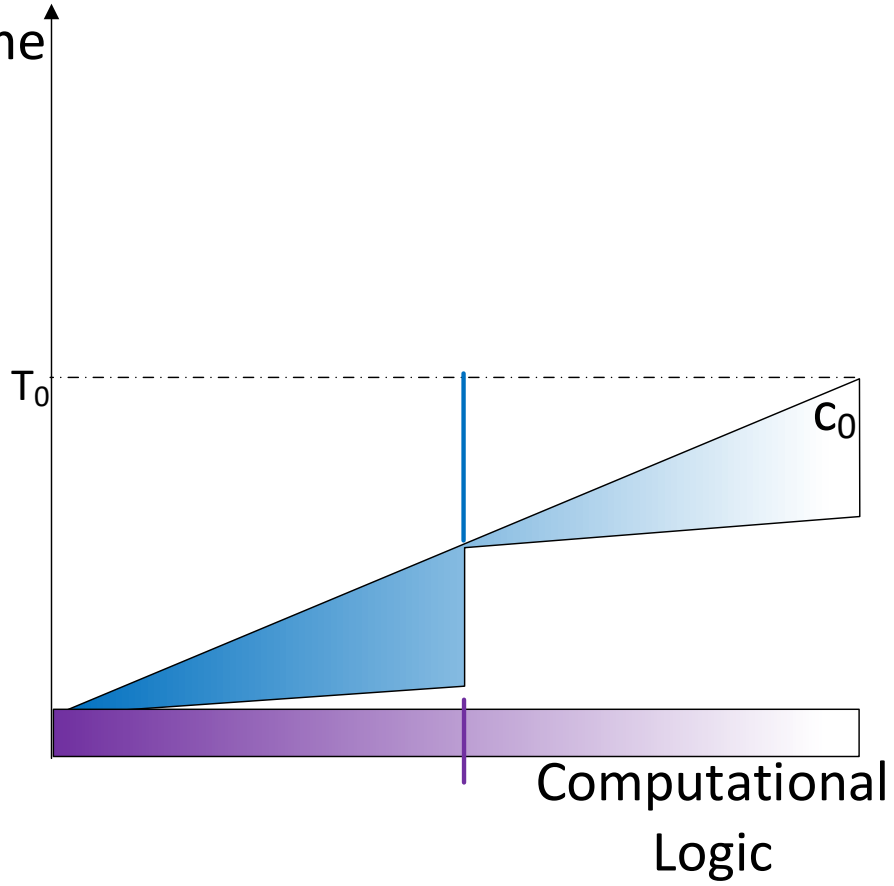
- FF-based pipelines (2-stage example)
  - Insert FFs at points in the logic corresponding to T/2 peak delay

Robust Pipelines



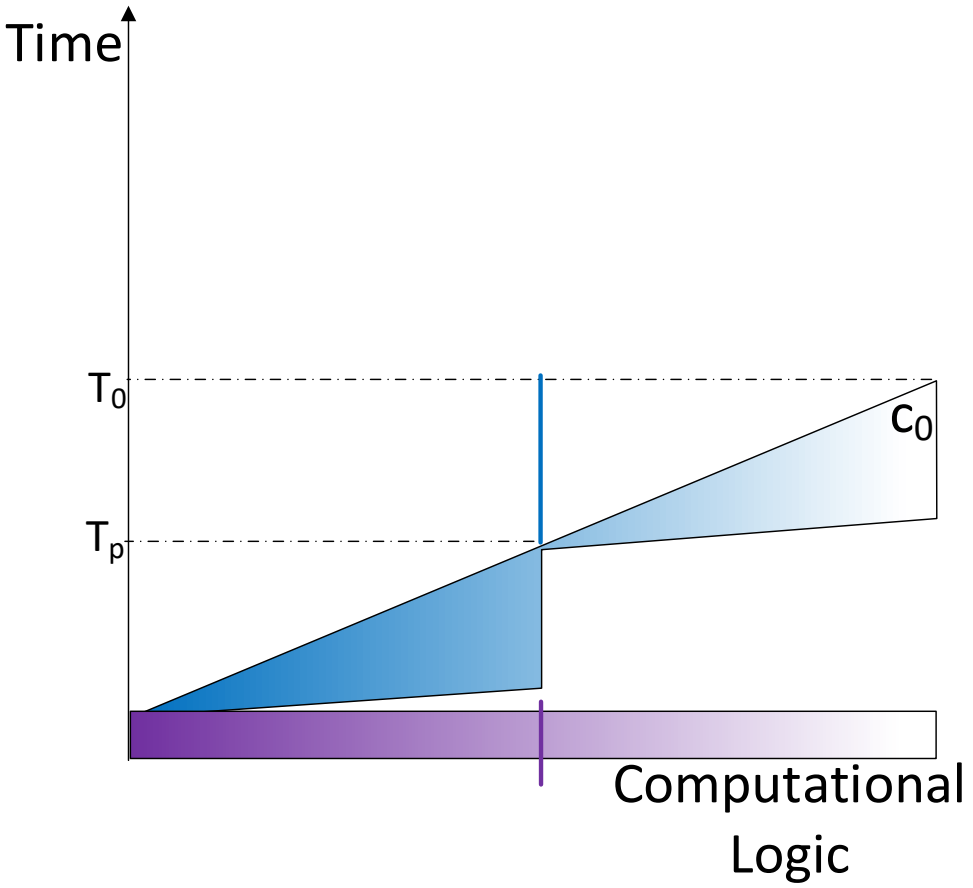
- FF-based pipelines (2-stage example)
  - Insert FFs at points in the logic corresponding to T/2 peak delay
  - FFs capture and store the values, avoiding successive fast-paths from colliding (or even interfering) with them (also referred to as a “race”)

Robust Pipelines



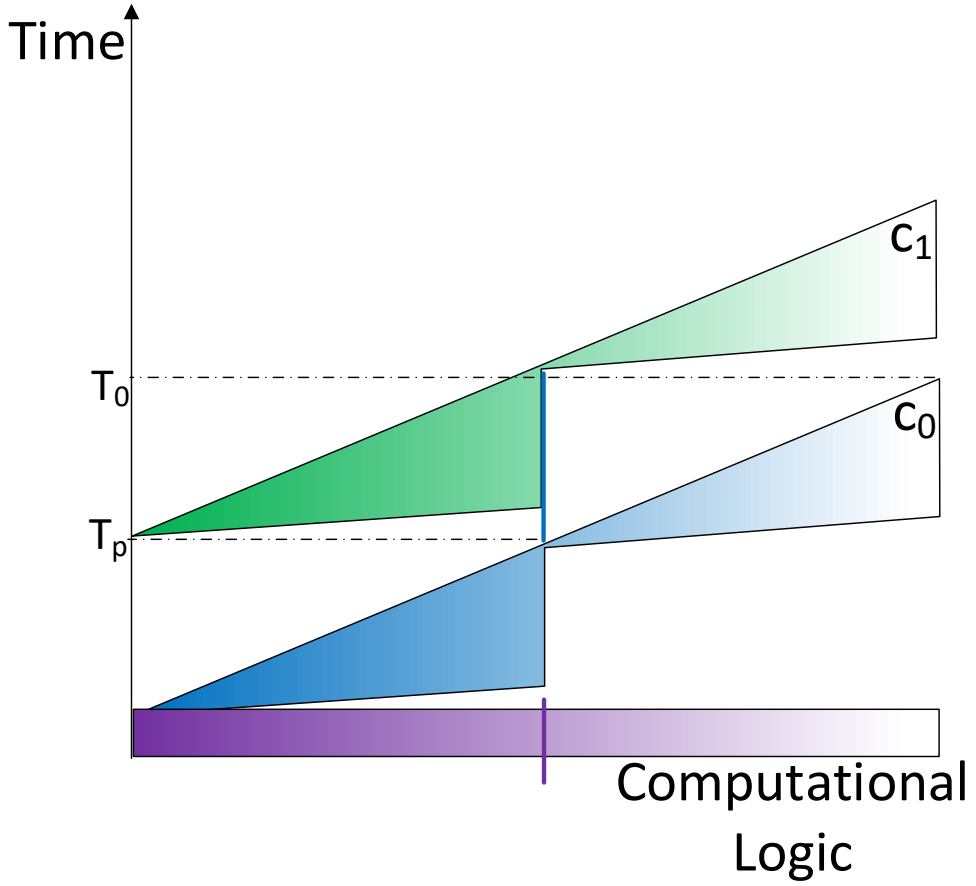
- FF-based pipelines (2-stage example)
  - Insert FFs at points in the logic corresponding to T/2 peak delay
  - FFs capture and store the values, avoiding successive fast-paths from colliding (or even interfering) with them (also referred to as a “race”)

Robust Pipelines



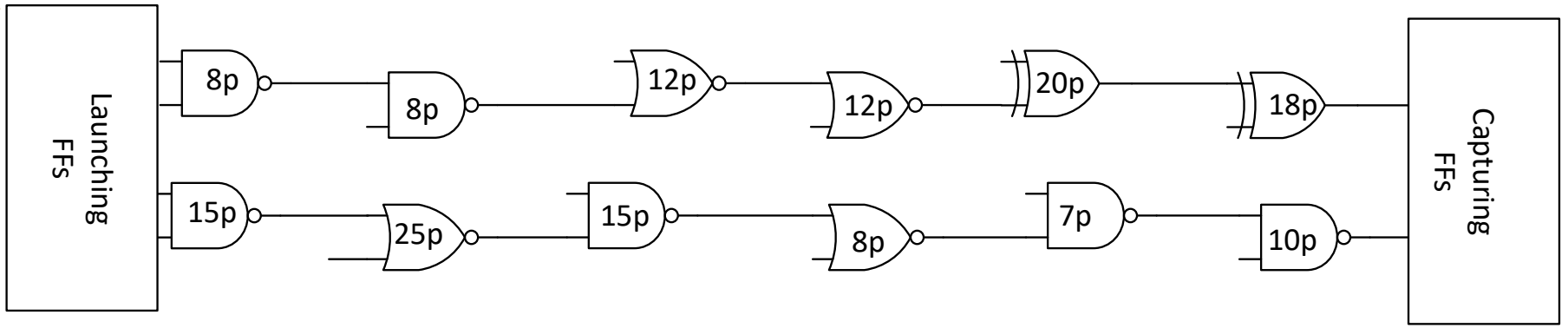
- FF-based pipelines (2-stage example)
  - Insert FFs at points in the logic corresponding to T/2 peak delay
  - FFs capture and store the values, avoiding successive fast-paths from colliding (or even interfering) with them (also referred to as a “race”)

Robust Pipelines



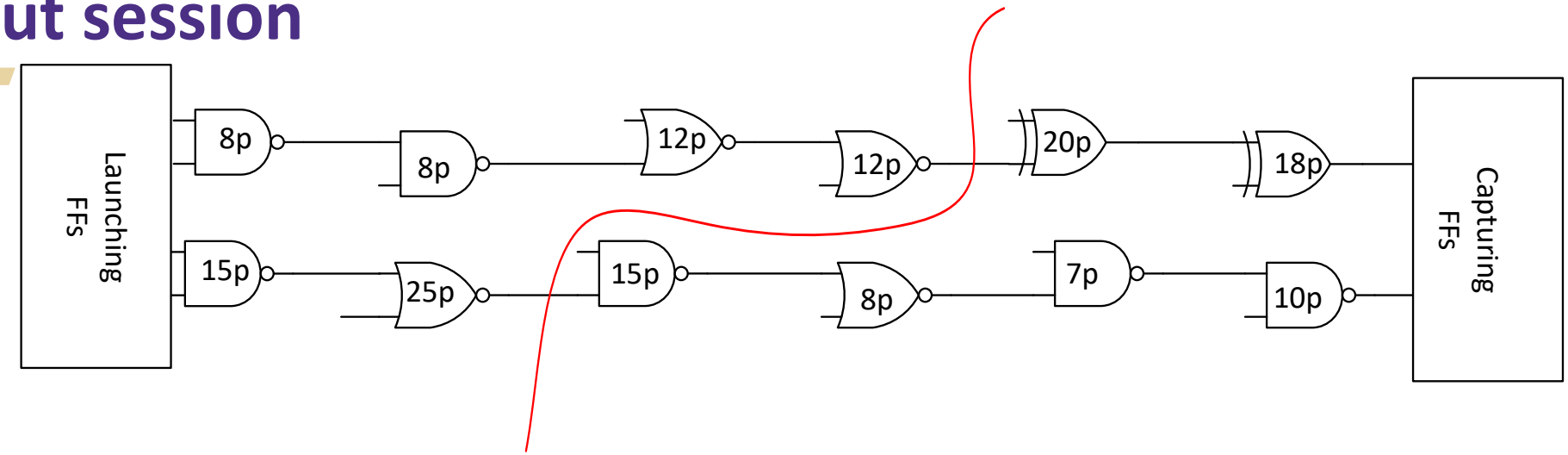
- FF-based pipelines (2-stage example)
  - Insert FFs at points in the logic corresponding to T/2 peak delay

Break-out session



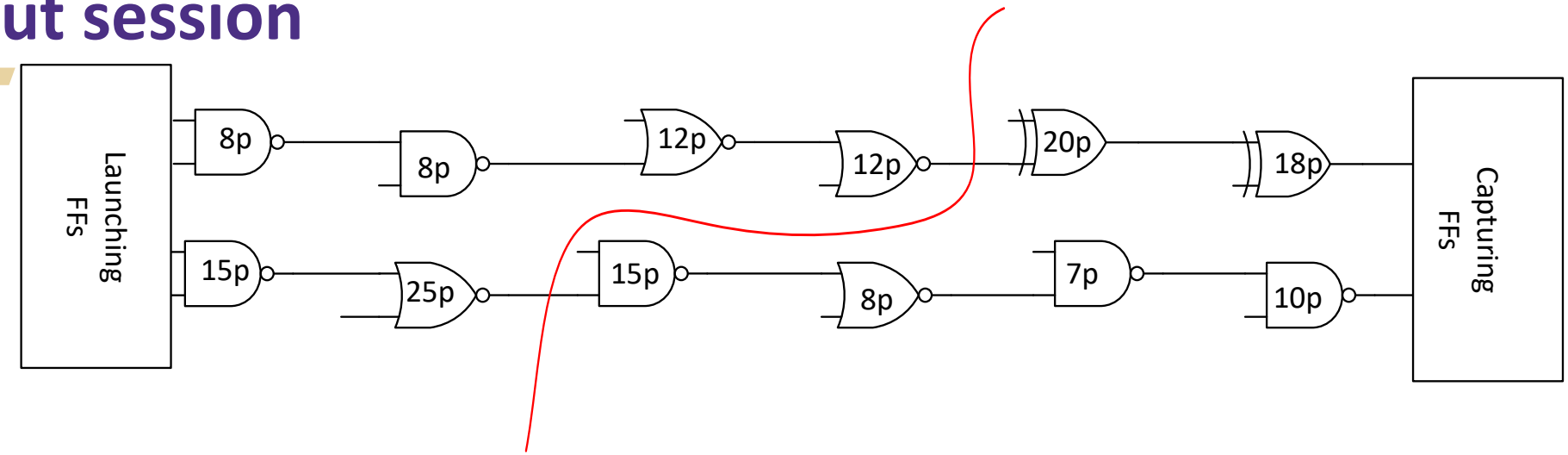
- Consider the following logic paths. Where should flip-flops be inserted to best improve performance for a 2-state design

Break-out session



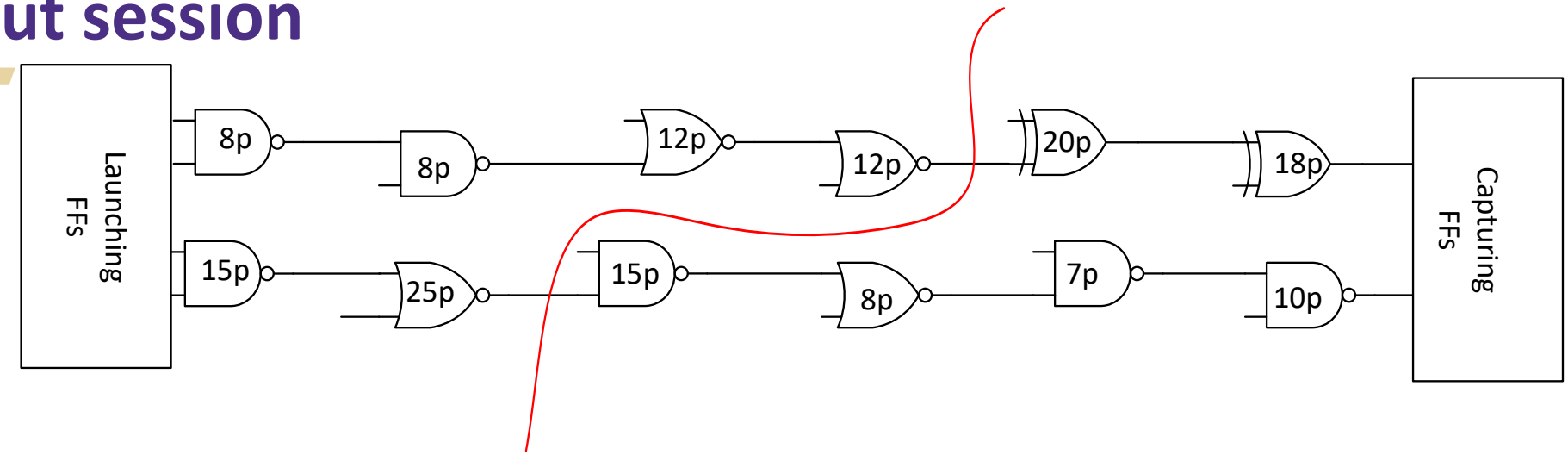
- Consider the following logic paths. Where should flip-flops be inserted to best improve performance for a 2-state design

Break-out session



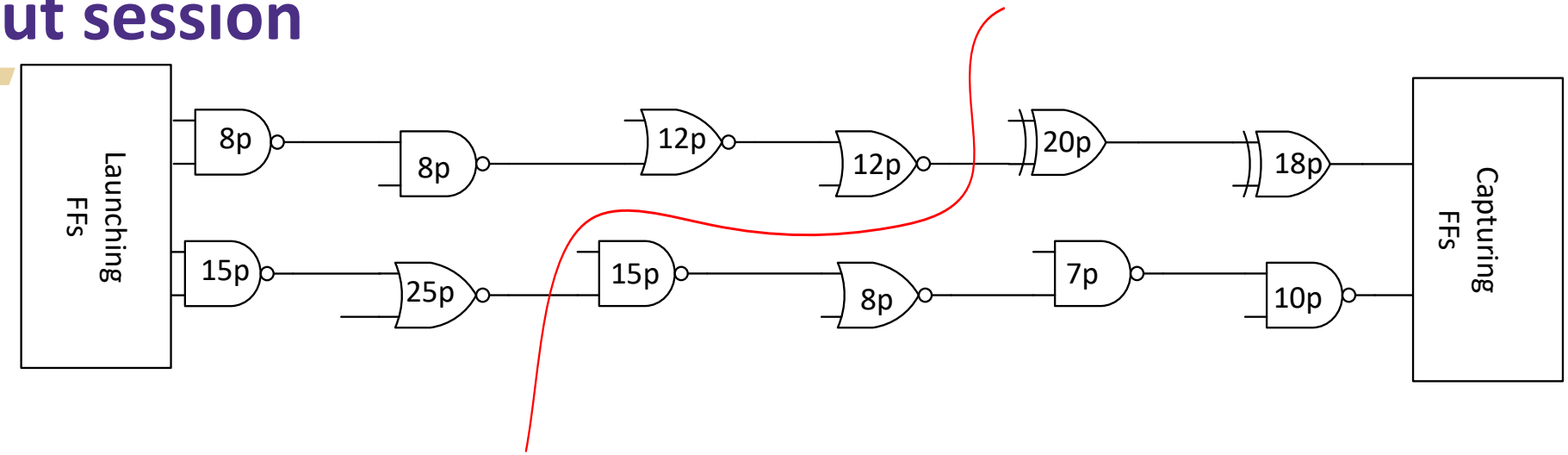
- Consider the following logic paths. Where should flip-flops be inserted to best improve performance for a 2-state design
  - FF insertion at nodes with max. delay of ~T/n
    - T = Total, maximum computational latency
    - n = Number of pipeline stages

Break-out session



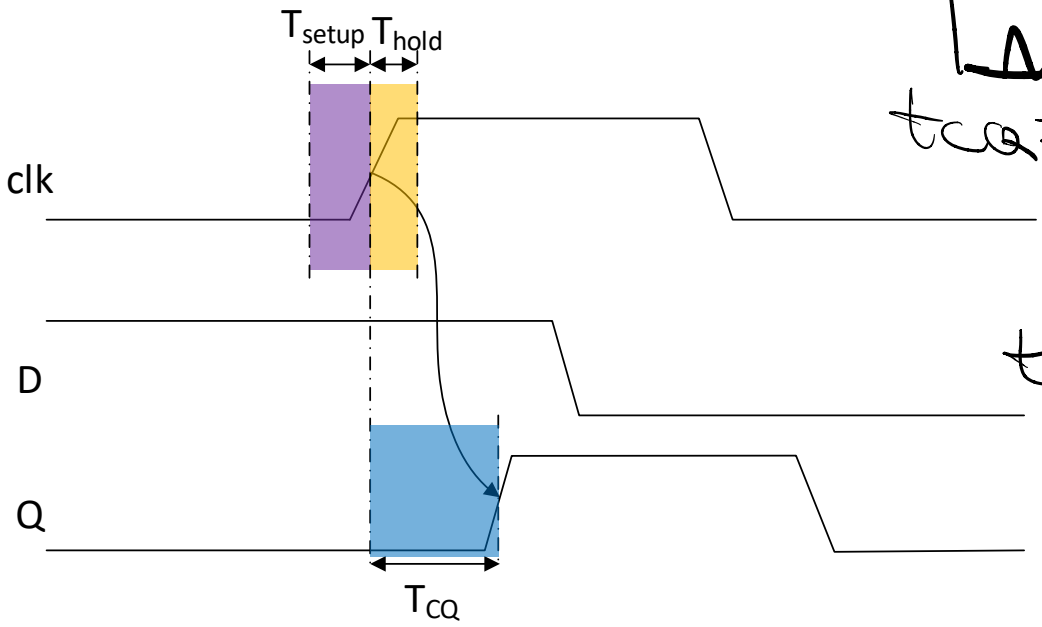
- Consider the following logic paths. Where should flip-flops be inserted to best improve performance for a 2-state design
  - FF insertion at nodes with max. delay of ~T/n
    - T = Total, maximum computational latency
    - n = Number of pipeline stages
- What will be the peak performance for a 3-stage design?

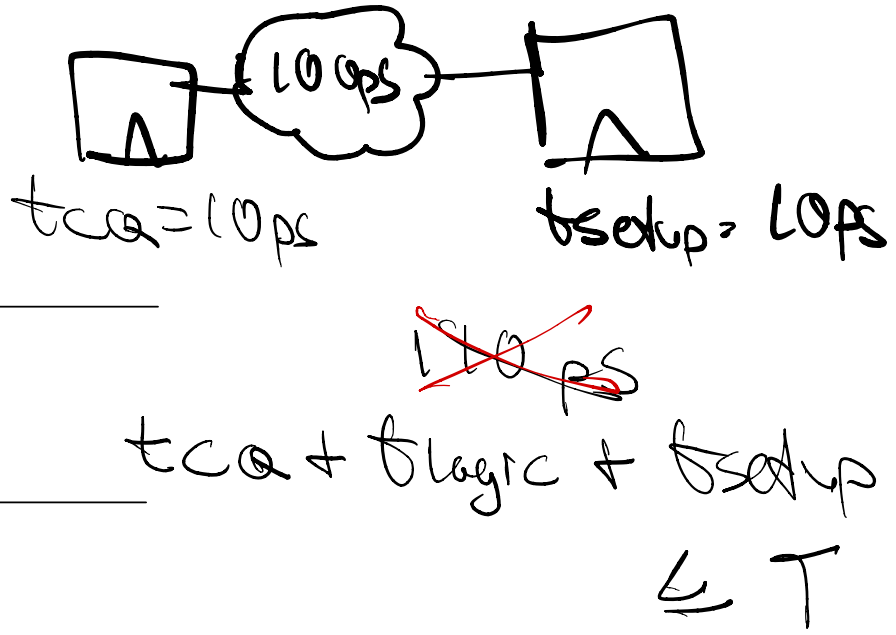
Break-out session



- Consider the following logic paths. Where should flip-flops be inserted to best improve performance for a 2-state design
  - FF insertion at nodes with max. delay of ~T/n
    - T = Total, maximum computational latency
    - n = Number of pipeline stages
- What will be the peak performance for a 3-stage design?
  - Delay-granularity is a consideration, especially for heavily pipelined designs

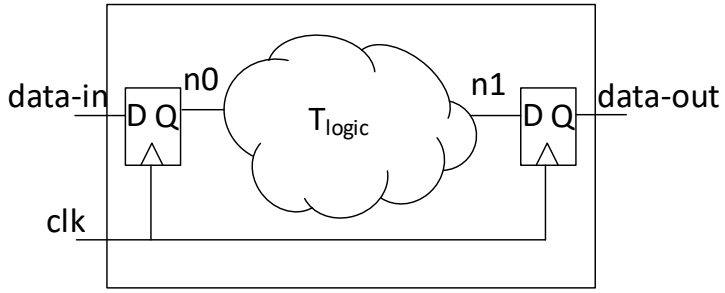
Flip-Flop Timing Properties





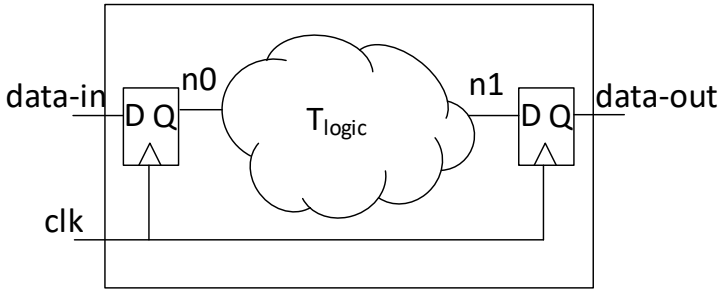
- FFs impose timing constraints on input data
  - Data to be latched must arrive no later than T_{setup} before clock edge
  - Data to be latched must be held stable for T_{hold} after clock edge
    - Early-arriving data from previous pipeline stage could degrade timing properties
  - Sampled input data does not get transmitted to the output instantly (T_{CQ})
- $T_{setup} + T_{CQ}$ a good metric for FF **timing overhead**
- $T_{CQ} - T_{hold}$ a good metric for **race immunity** (more on this later)

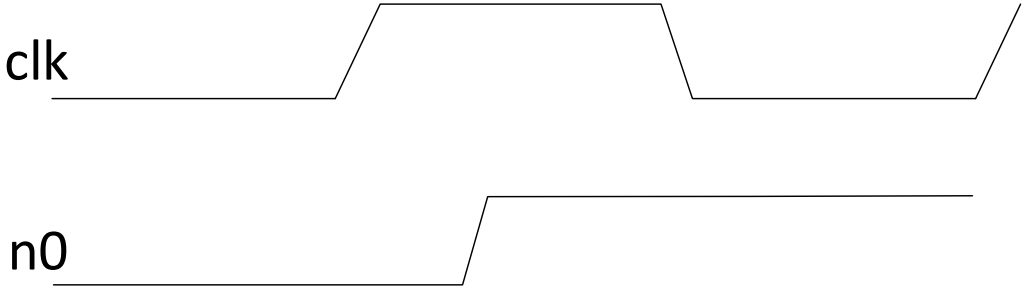
Pipelined Designs: Basic Timing Analysis



- Cycle-time (and throughput) depend on FF parameters and logic
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup}$
- A design can consist of multiple such synchronous blocks
  - Blocks synchronized by the same clock said to be in the same **clock domain**
  - Cycle time of the design is determined by the slowest block.

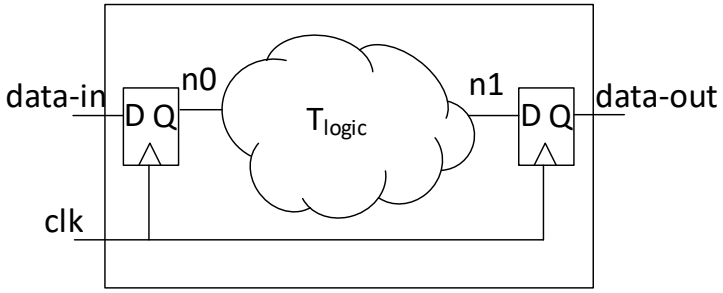
Pipelined Designs: Basic Timing Analysis

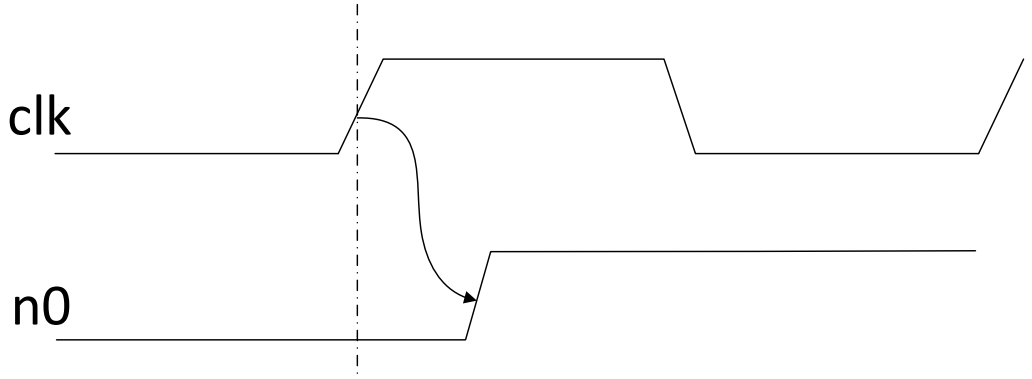




- Cycle-time (and throughput) depend on FF parameters and logic
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup}$
- A design can consist of multiple such synchronous blocks
  - Blocks synchronized by the same clock said to be in the same **clock domain**
  - Cycle time of the design is determined by the slowest block.

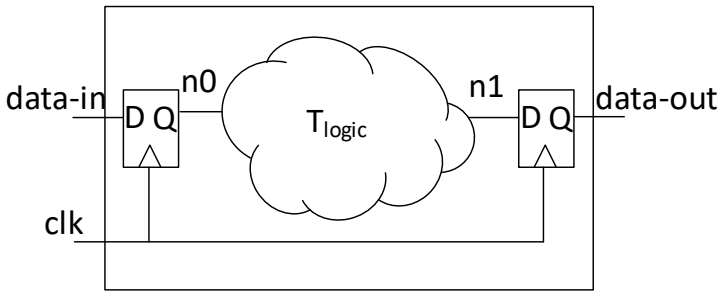
Pipelined Designs: Basic Timing Analysis

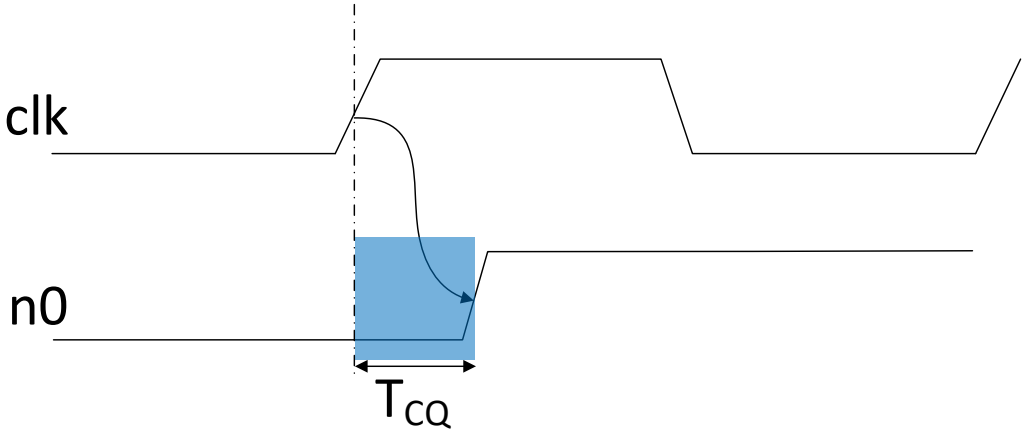




- Cycle-time (and throughput) depend on FF parameters and logic
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup}$
- A design can consist of multiple such synchronous blocks
  - Blocks synchronized by the same clock said to be in the same **clock domain**
  - Cycle time of the design is determined by the slowest block.

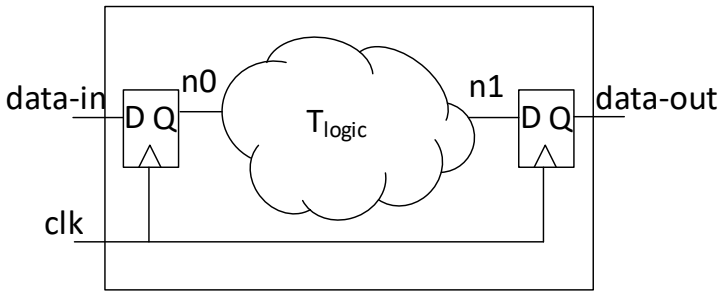
Pipelined Designs: Basic Timing Analysis

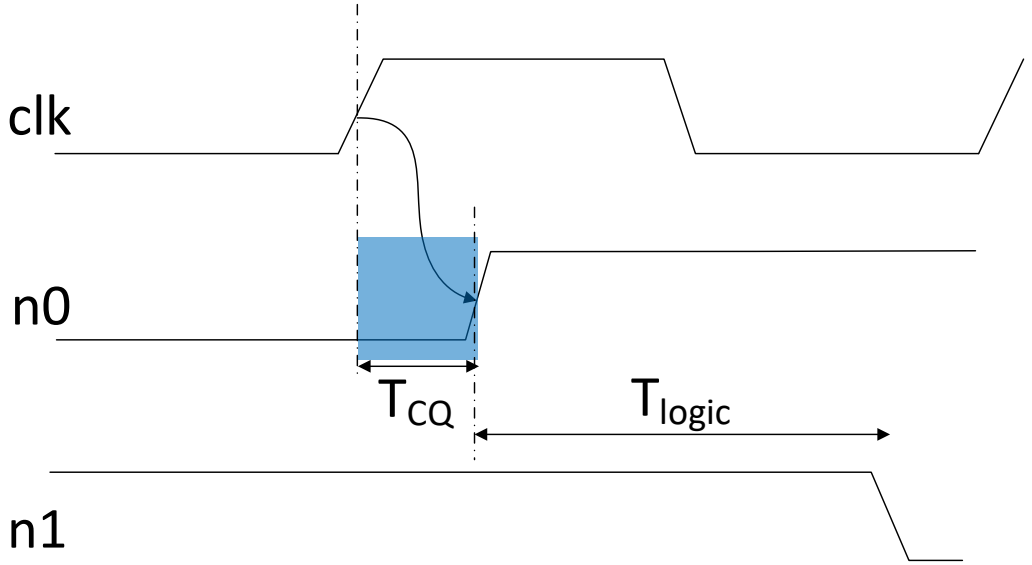




- Cycle-time (and throughput) depend on FF parameters and logic
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup}$
- A design can consist of multiple such synchronous blocks
  - Blocks synchronized by the same clock said to be in the same **clock domain**
  - Cycle time of the design is determined by the slowest block.

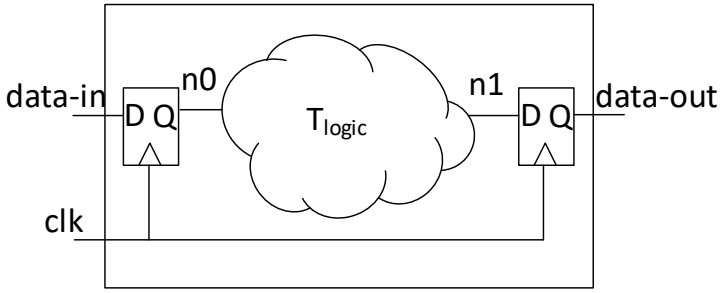
Pipelined Designs: Basic Timing Analysis

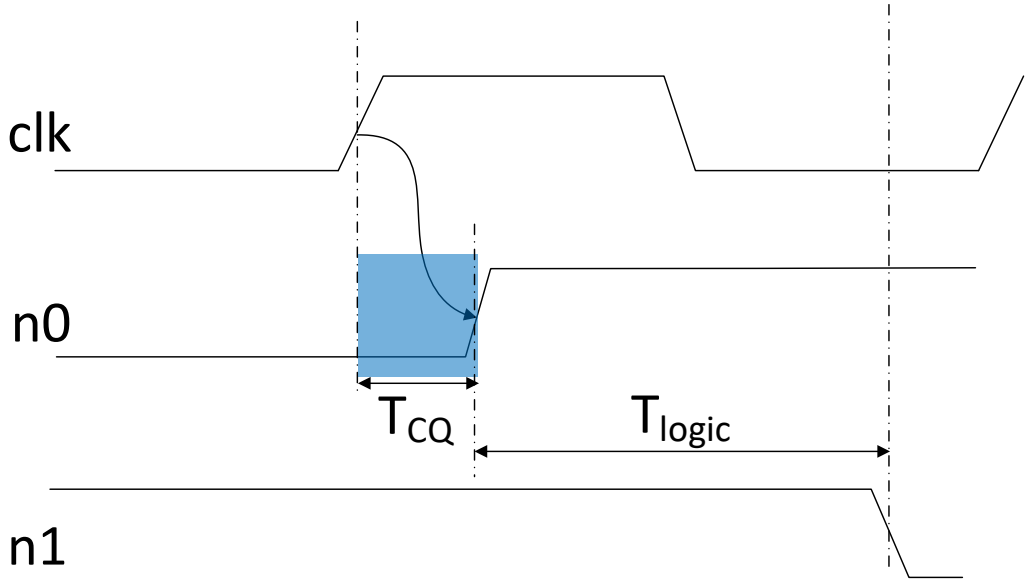




- Cycle-time (and throughput) depend on FF parameters and logic
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup}$
- A design can consist of multiple such synchronous blocks
  - Blocks synchronized by the same clock said to be in the same **clock domain**
  - Cycle time of the design is determined by the slowest block.

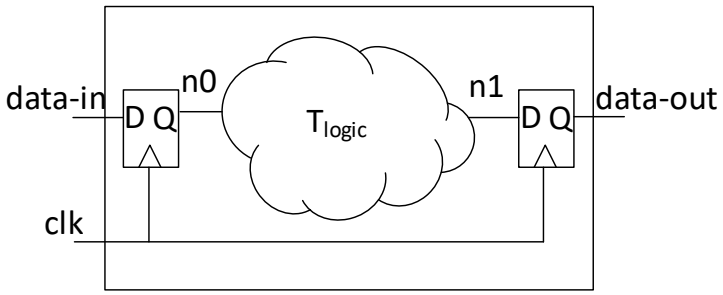
Pipelined Designs: Basic Timing Analysis

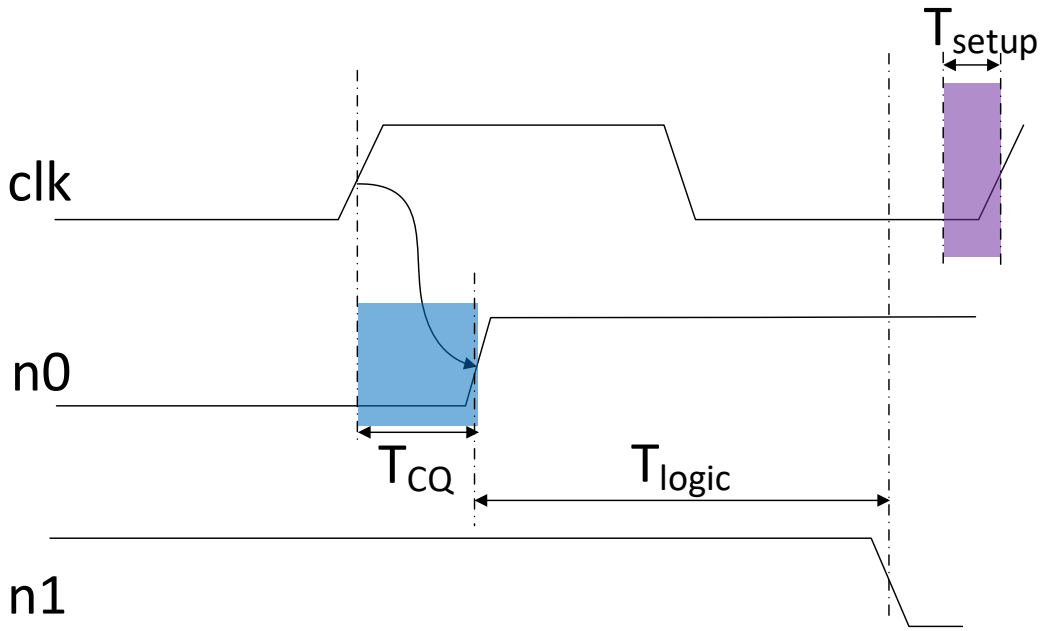




- Cycle-time (and throughput) depend on FF parameters and logic
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup}$
- A design can consist of multiple such synchronous blocks
  - Blocks synchronized by the same clock said to be in the same **clock domain**
  - Cycle time of the design is determined by the slowest block.

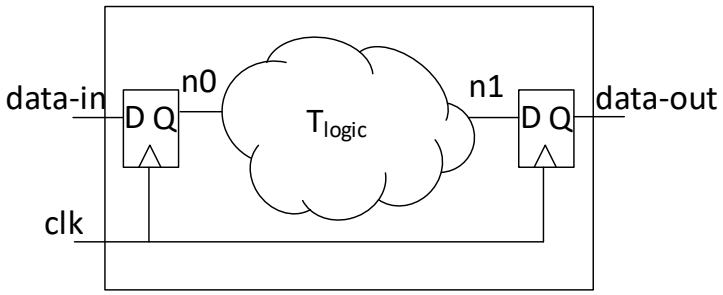
Pipelined Designs: Basic Timing Analysis

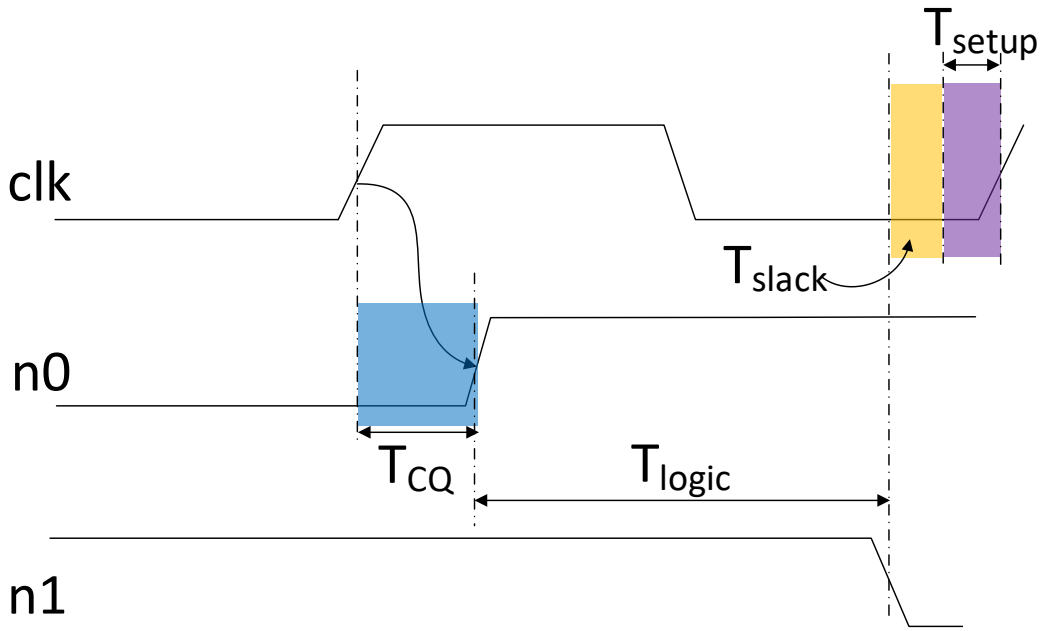




- Cycle-time (and throughput) depend on FF parameters and logic
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup}$
- A design can consist of multiple such synchronous blocks
  - Blocks synchronized by the same clock said to be in the same **clock domain**
  - Cycle time of the design is determined by the slowest block.

Pipelined Designs: Basic Timing Analysis

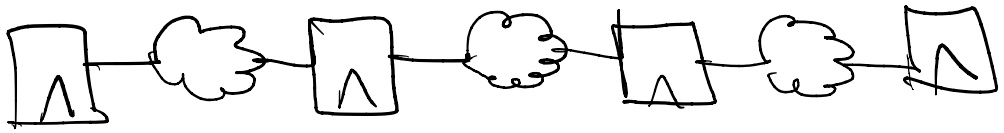




- Cycle-time (and throughput) depend on FF parameters and logic
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup}$
- A design can consist of multiple such synchronous blocks
  - Blocks synchronized by the same clock said to be in the same **clock domain**
  - Cycle time of the design is determined by the slowest block.

Performance

- For a combinational block, addition of N-pipeline stages impacts several parameters.
  - Total Latency (Ignoring area impact)
    - Absolute Time : $T_{logic} + N*(T_{setup}+T_{cq})$ (↑)
    - $N*T_{clk}$ in cycles
  - Throughput improvement (Ignoring area* impact)
    - Perf. Improvement = $\frac{T_{cycle,nom}}{T_{cycle,N-stage}} = \frac{T_{logic}+T_{CQ}+T_{setup}}{\frac{T_{logic}}{N}+T_{CQ}+T_{setup}}$
    - Assumes logic can be equally divided. Remember **Longest pipe-stage sets cycle-time**
  - Area
    - $A_N = A_{logic} + \sum_1^N A_i$
  - Power*
    - $P_N = P + \sum_1^N P_i$



$\frac{T_{logic}}{N} + T_{setup} + T_{cq}$

- **Exercise**. Consider an unpipelined design with T_{logic} = 2ns. Sketch the plot for cycle time vs pipeline depth, N with T_{setup}=T_{cq}=200ps

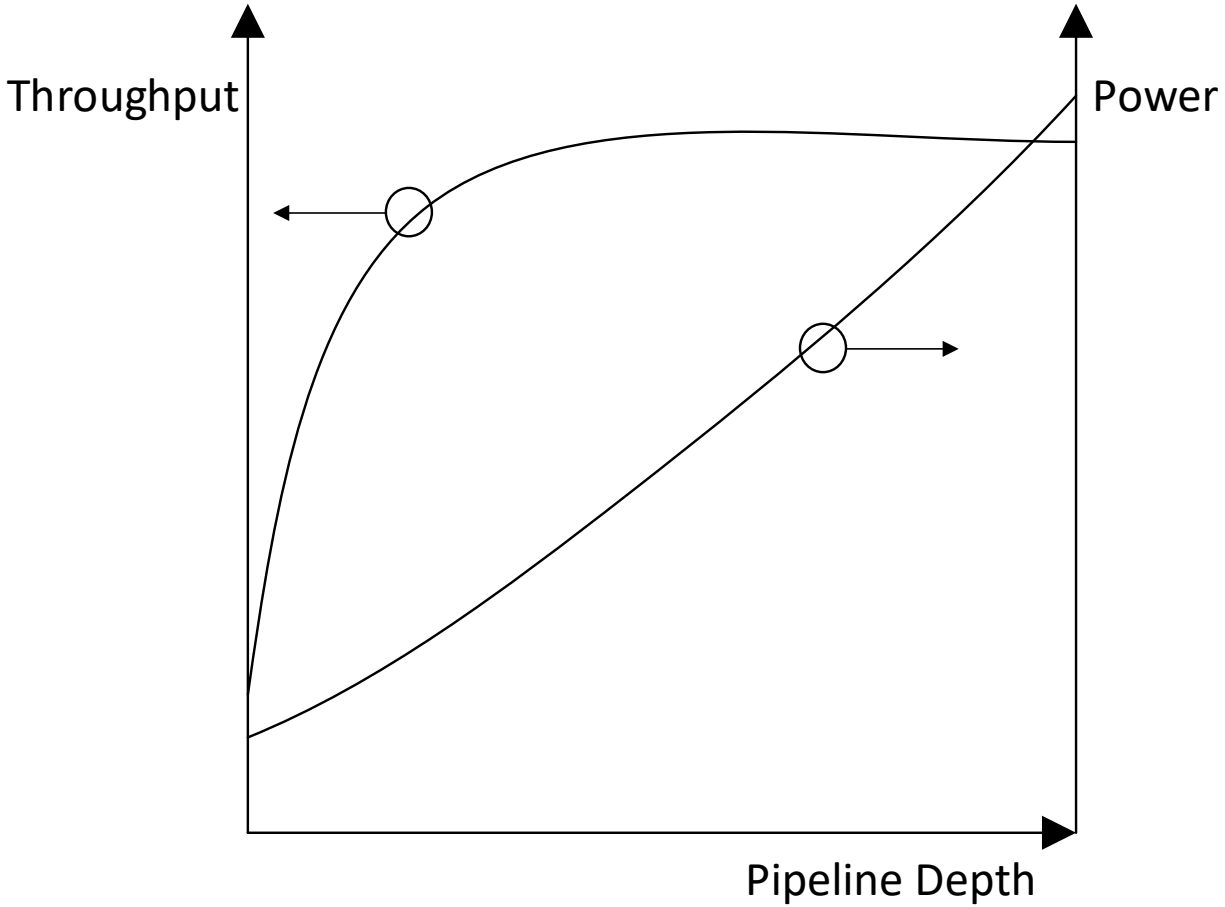
Performance

- For a combinational block, addition of N-pipeline stages impacts several parameters.
  - Total Latency (Ignoring area impact)
    - Absolute Time : $T_{logic} + N*(T_{setup}+T_{cq})$ (↑)
    - $N*T_{clk}$ in cycles
  - Throughput improvement (Ignoring area* impact)
    - Perf. Improvement = $\frac{T_{cycle,nom}}{T_{cycle,N-stage}} = \frac{T_{logic}+T_{CQ}+T_{setup}}{\frac{T_{logic}}{N}+T_{CQ}+T_{setup}}$
    - Assumes logic can be equally divided. Remember **Longest pipe-stage sets cycle-time**
  - Area
    - $A_N = A_{logic} + \sum_i^N A_i$
  - Power*
    - $P_N = P + \sum_i^N P_i$

↑ wire/driver cap due to larger area
↑ clock load. ↑ FF power

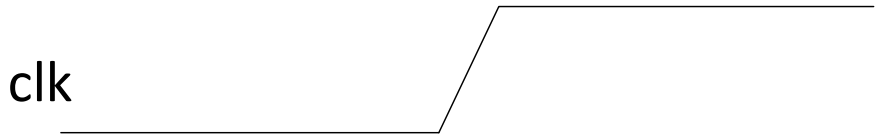
- **Exercise**. Consider an unpipelined design with T_{logic} = 2ns. Sketch the plot for cycle time vs pipeline depth, N with T_{setup}=T_{cq}=200ps

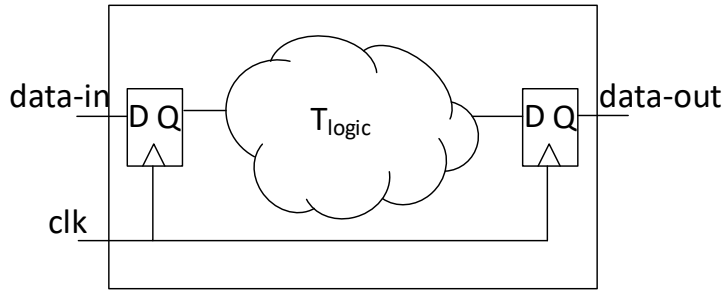
Pipelining Limits



- Ideal flip-flops (Timing and power) → Pipelining is free of cost
- Real flip-flops → Pipelining provides benefits only up to a point
  - Non-zero delays, capacitive loading, power and area count diminish returns

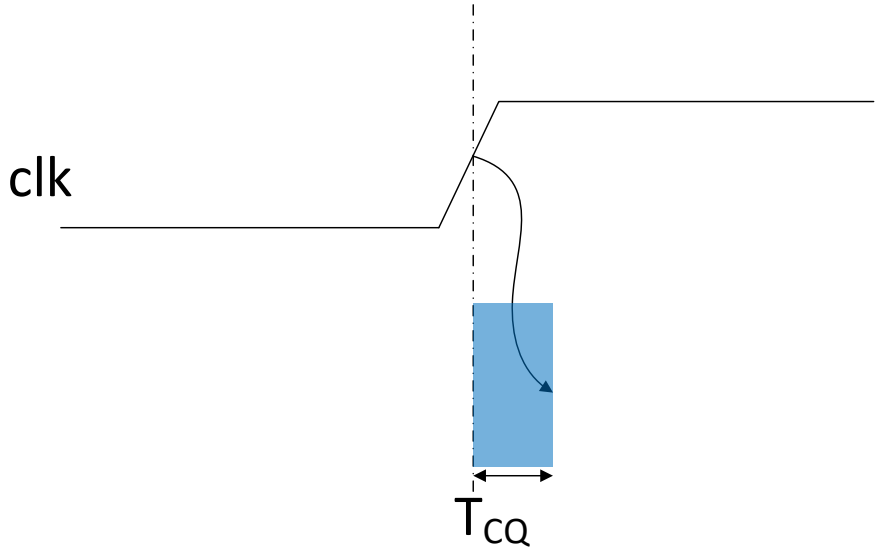
Hold Time Analysis (Race)

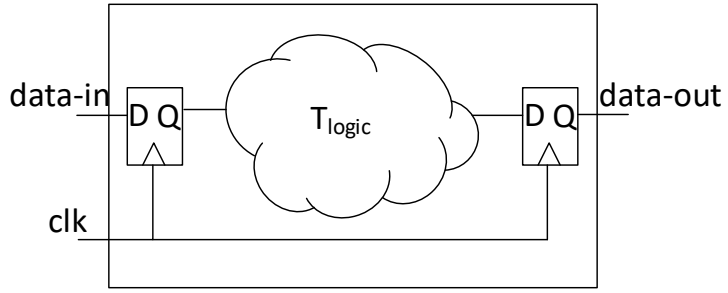




- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

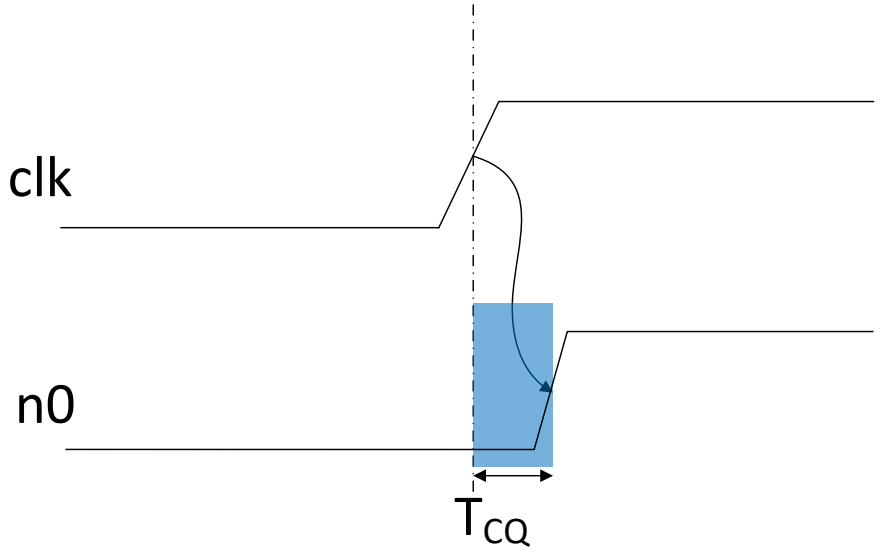
Hold Time Analysis (Race)

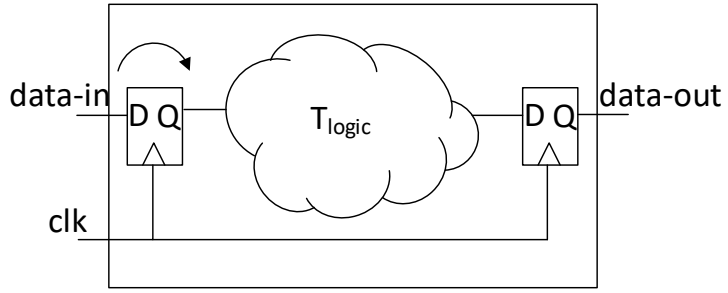




- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

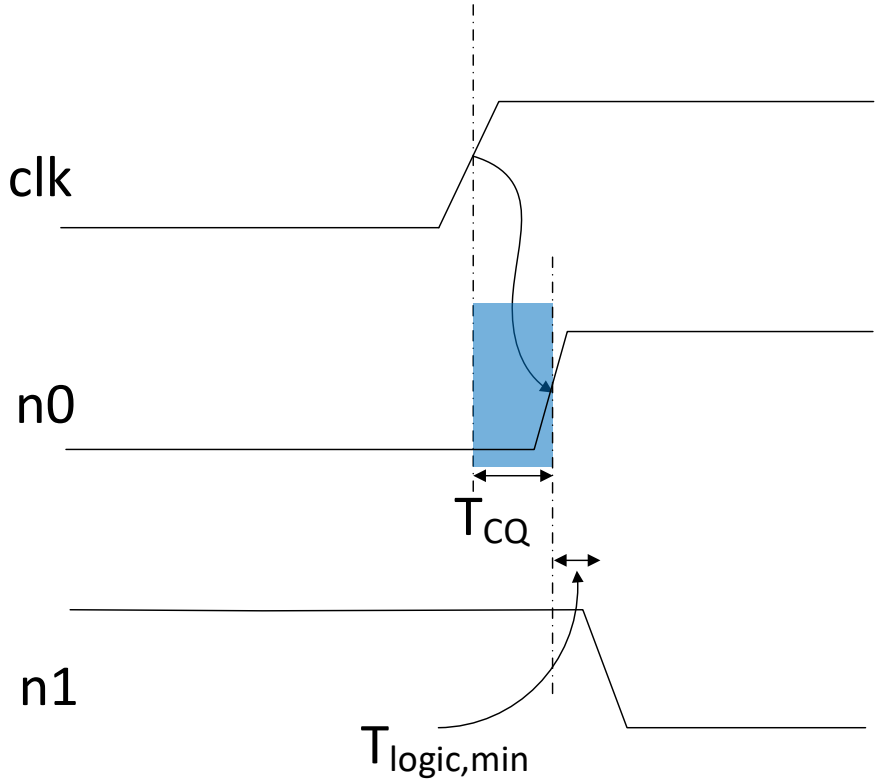
Hold Time Analysis (Race)

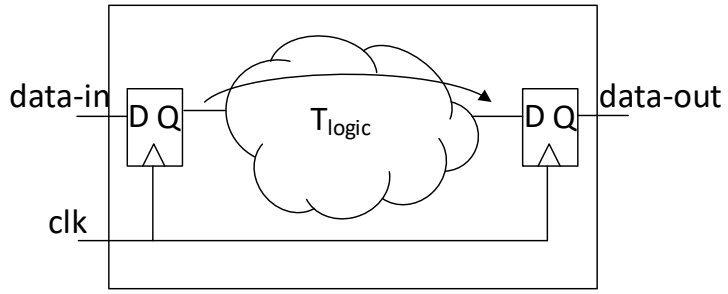




- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

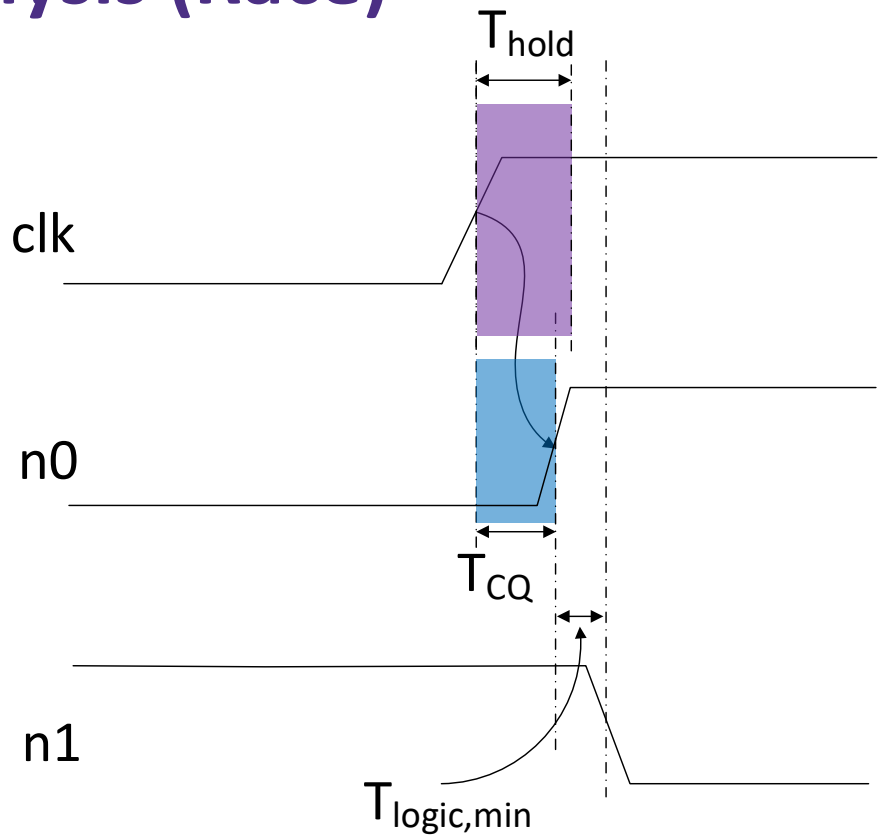
Hold Time Analysis (Race)

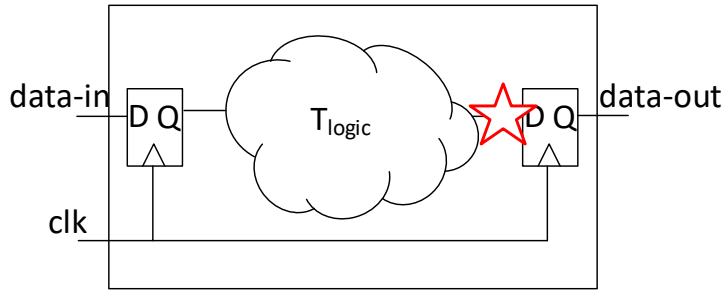




- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

Hold Time Analysis (Race)

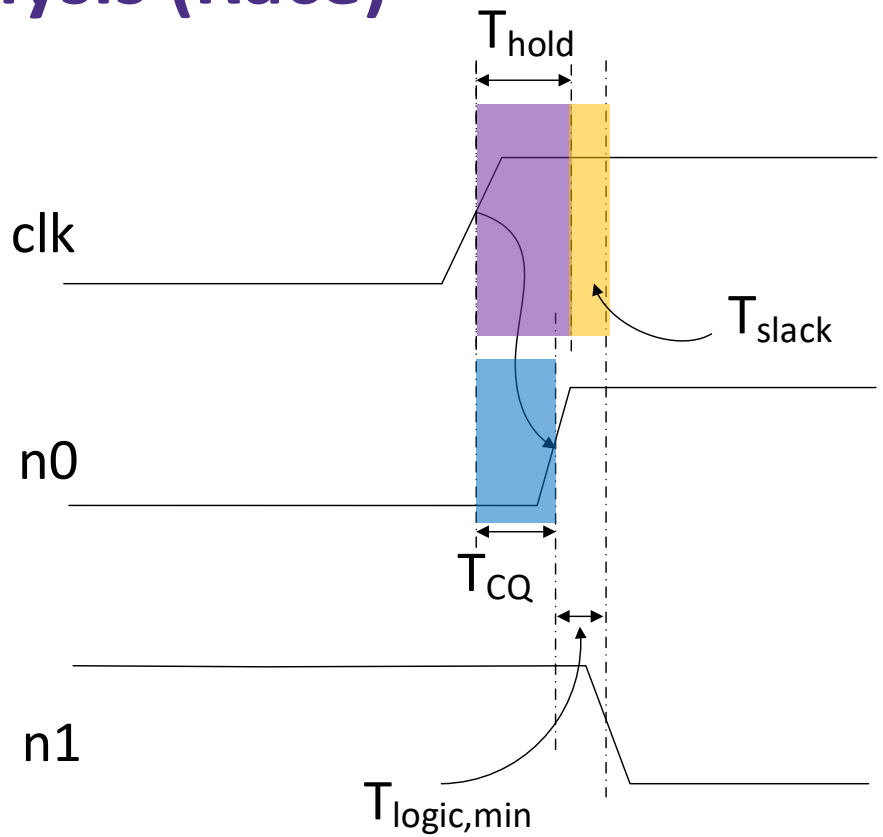


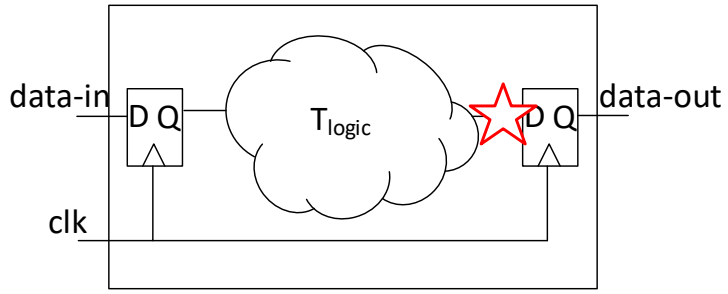


$T_{cq} + t_{contamination} \geq T_{hold}$

- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

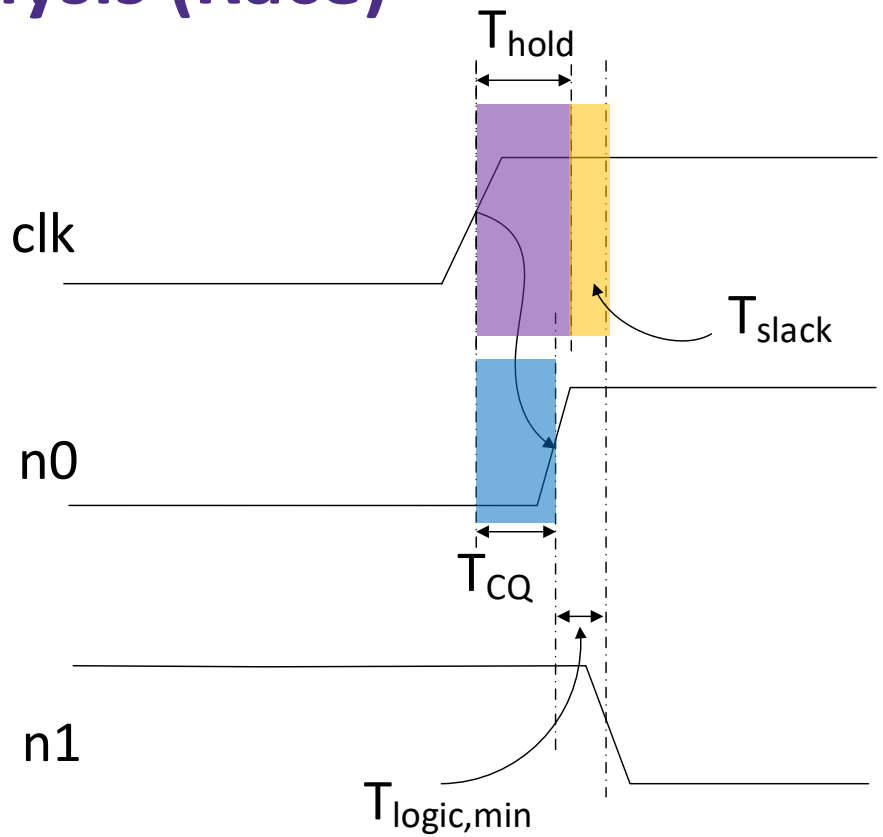
Hold Time Analysis (Race)

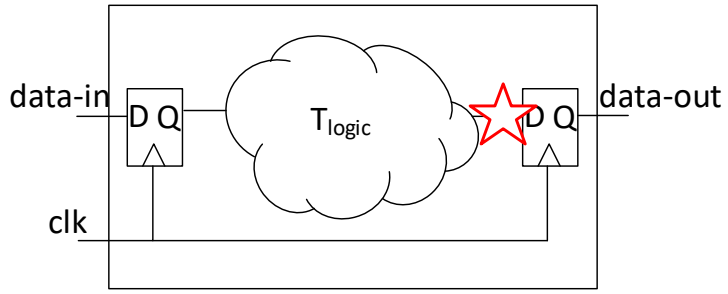




- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

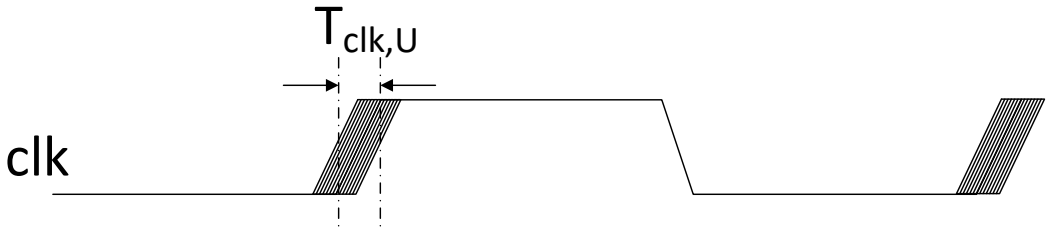
Hold Time Analysis (Race)





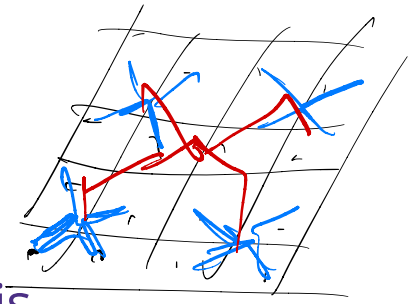
- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF
- $T_{hold} \leq T_{CQ} + T_{logic,min}$

Clock Uncertainty

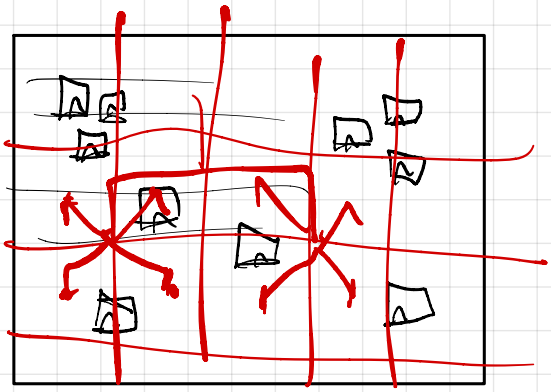


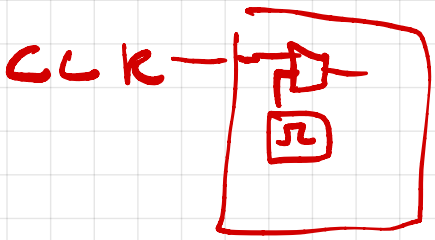
CLK

- 2 components: Skew and jitter
- Clock skew
  - Earlier analysis assumed that *clk* arrives at all flops simultaneously
  - Difficult to do in reality
  - In real designs, *clk* arrives at different times
  - Time invariant component (invariant relative to clock propagation time) is referred to as clock skew: T_{skew}
    - Time-invariant (sort of) component: T_{skew}



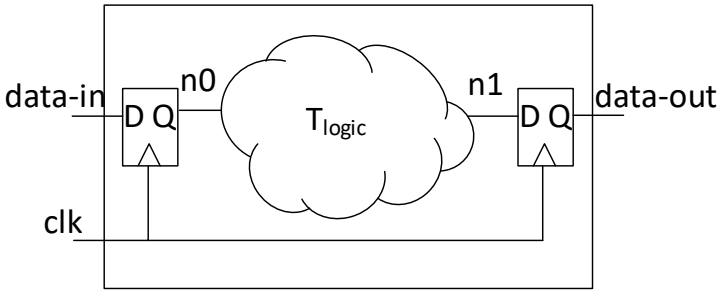
- Clock jitter
  - Time-variant component : T_{jitter}
  - Accurate analysis of impact on timing is a little more involved (T_{hold} particularly)
- Simplifying assumption: Worst case clock uncertainty everywhere
  - $T_{clk,U} = T_{skew} + T_{jitter}$




CLK

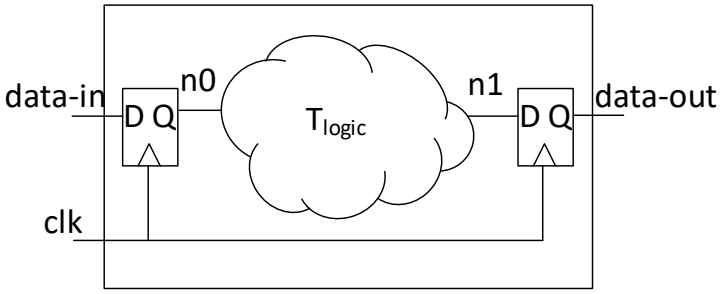
CLK
Root
Leaf
CLK_root
leaf
insertion delay
CLKA
CLKB
CLKA
CLKB

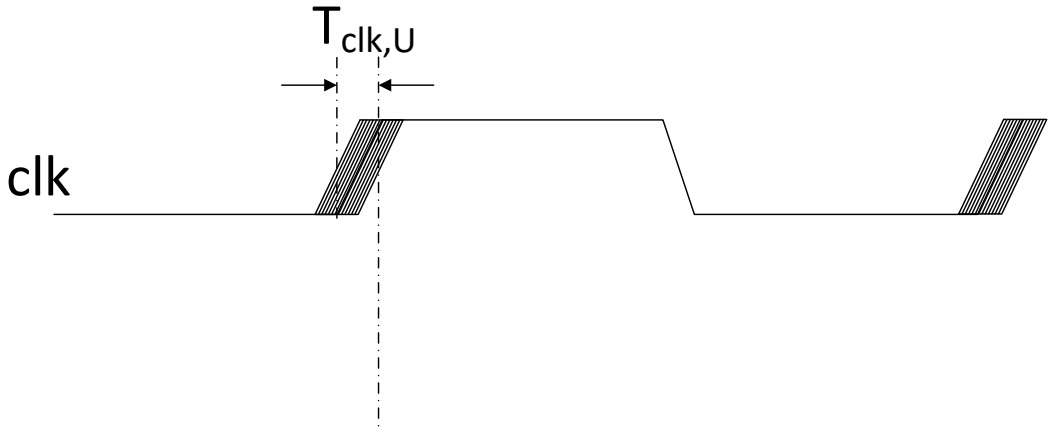
Pipelined Designs: Clock Uncertainty (Setup)



- Setup time analysis: Late launch, early capture
- Cycle-time
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup} + T_{clk,U}$

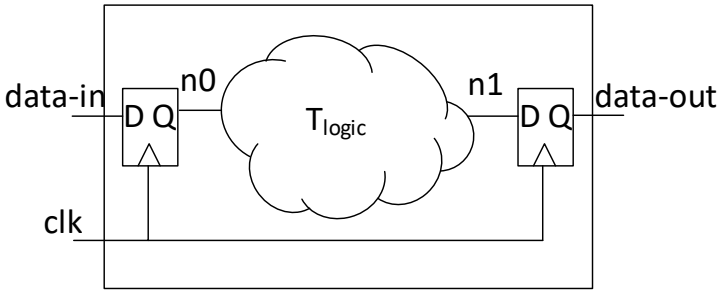
Pipelined Designs: Clock Uncertainty (Setup)

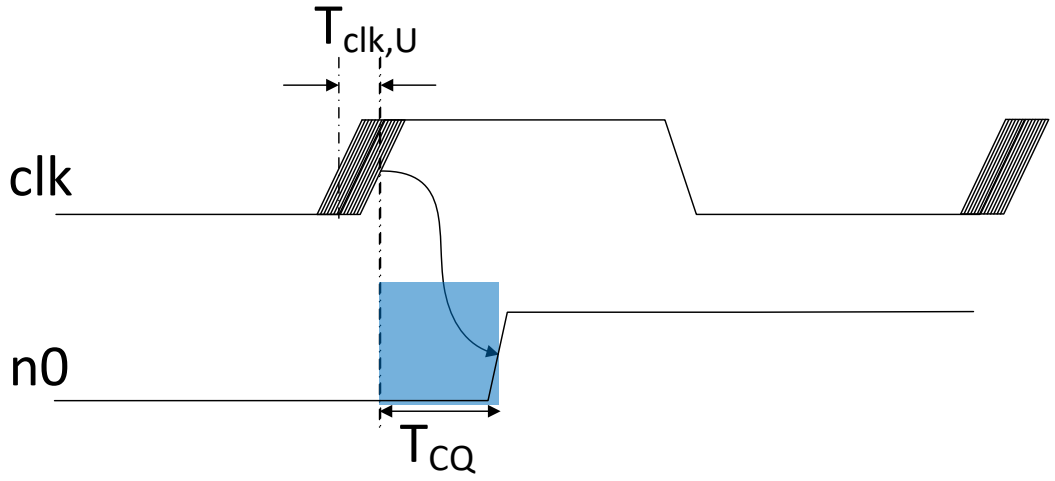




- Setup time analysis: Late launch, early capture
- Cycle-time
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup} + T_{clk,U}$

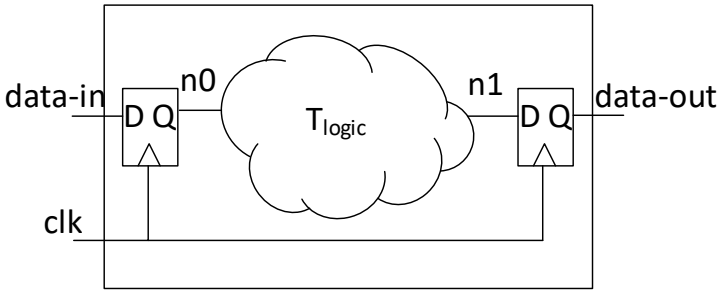
Pipelined Designs: Clock Uncertainty (Setup)

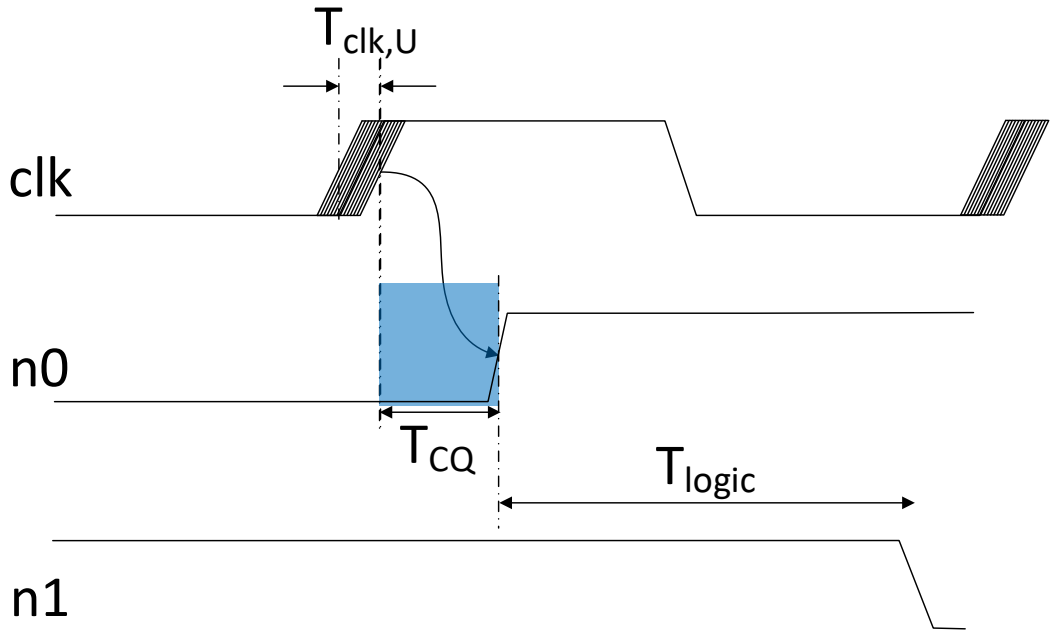




- Setup time analysis: Late launch, early capture
- Cycle-time
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup} + T_{clk,U}$

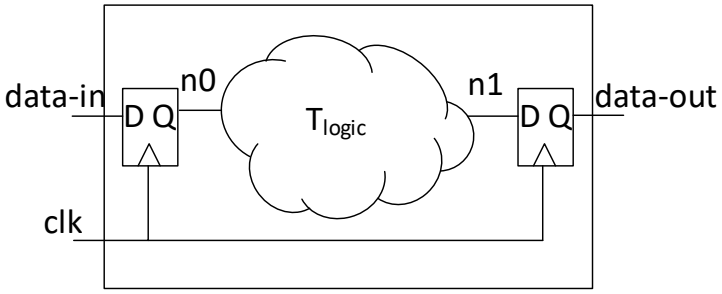
Pipelined Designs: Clock Uncertainty (Setup)

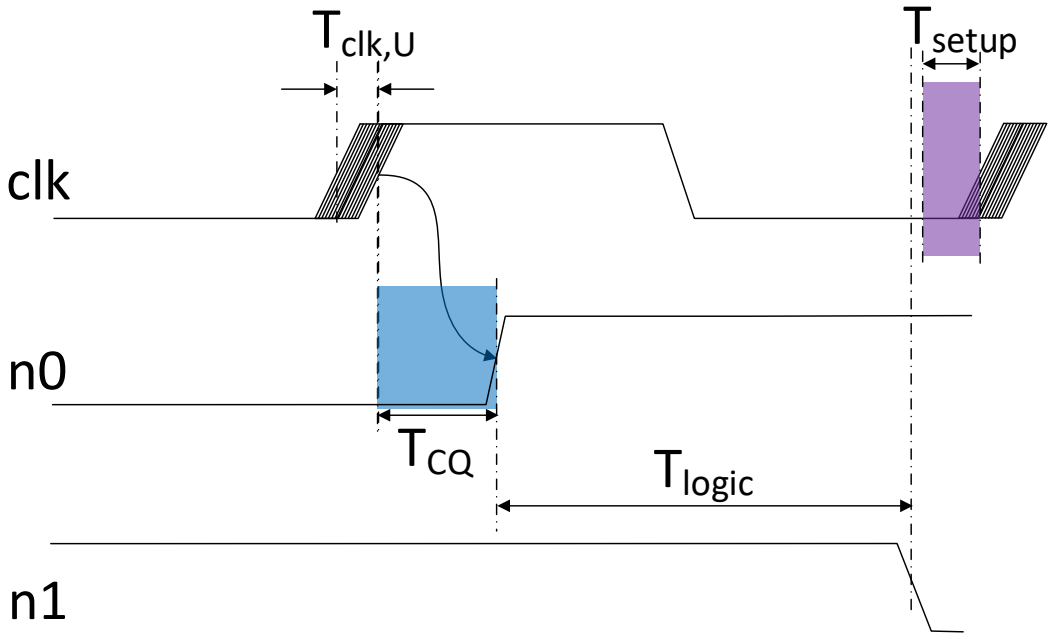




- Setup time analysis: Late launch, early capture
- Cycle-time
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup} + T_{clk,U}$

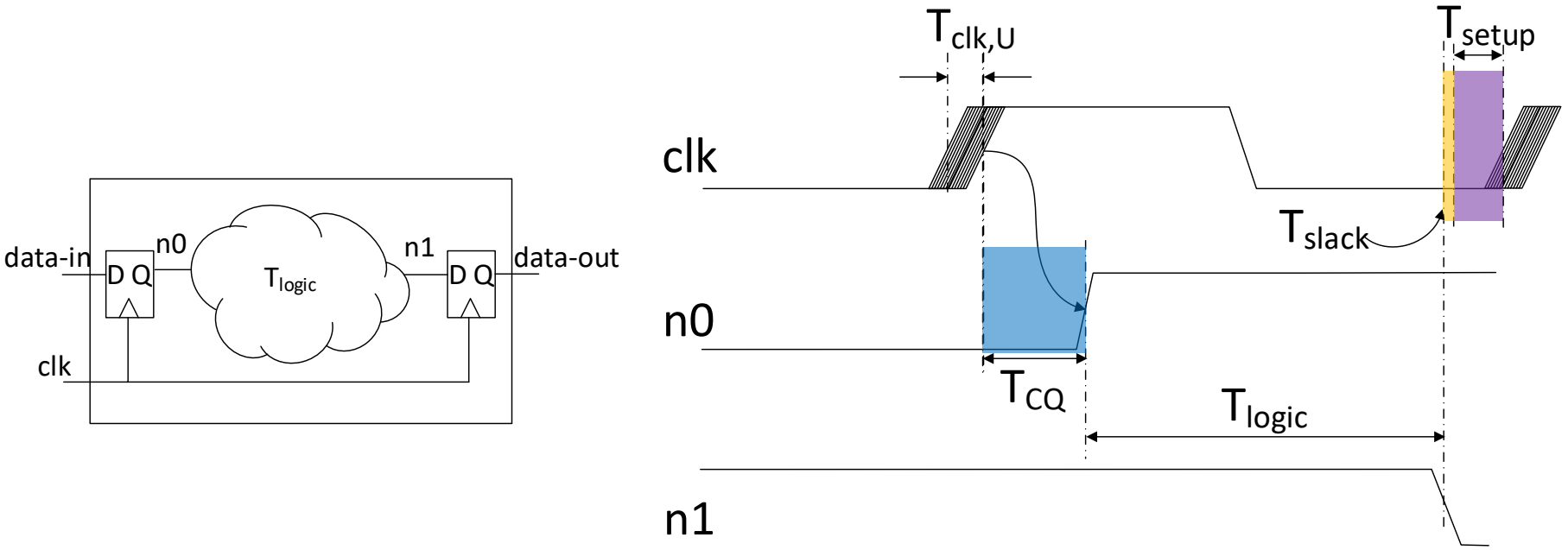
Pipelined Designs: Clock Uncertainty (Setup)





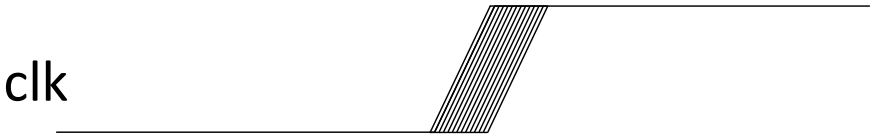
- Setup time analysis: Late launch, early capture
- Cycle-time
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup} + T_{clk,U}$

Pipelined Designs: Clock Uncertainty (Setup)



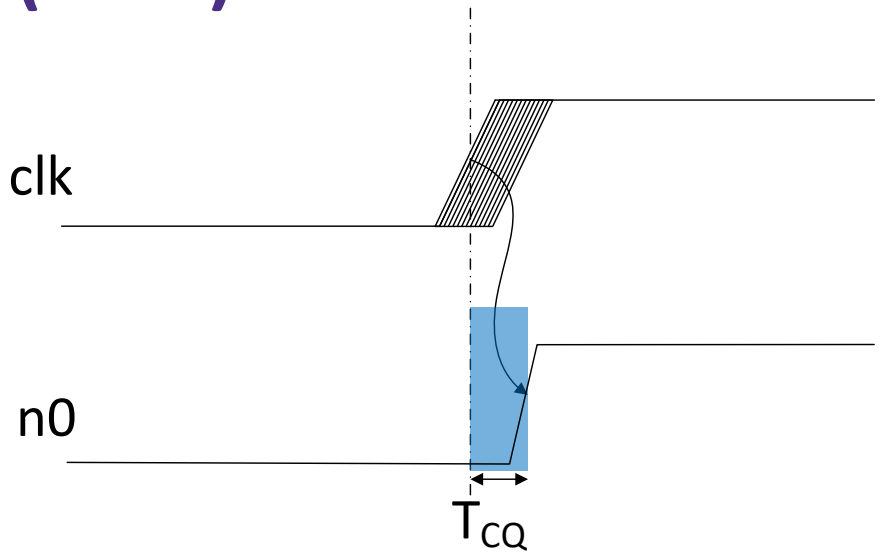
- Setup time analysis: Late launch, early capture
- Cycle-time
  - $T_{cycle} \geq T_{cq} + T_{logic,max} + T_{setup} + T_{clk,U}$

Clock Uncertainty (Hold)



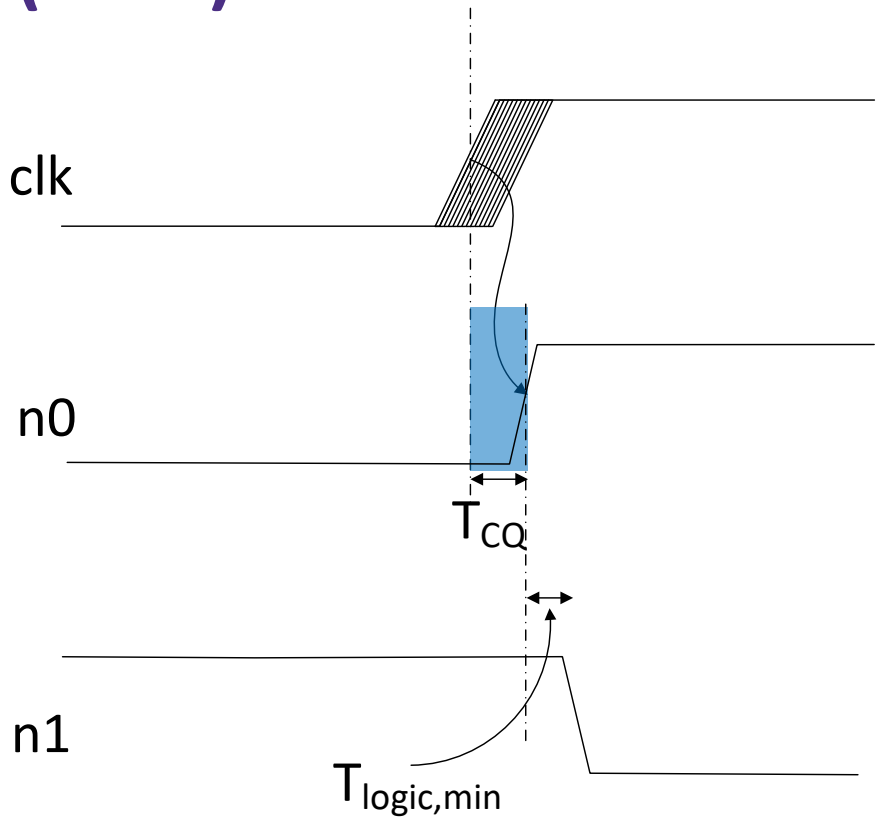
- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

Clock Uncertainty (Hold)



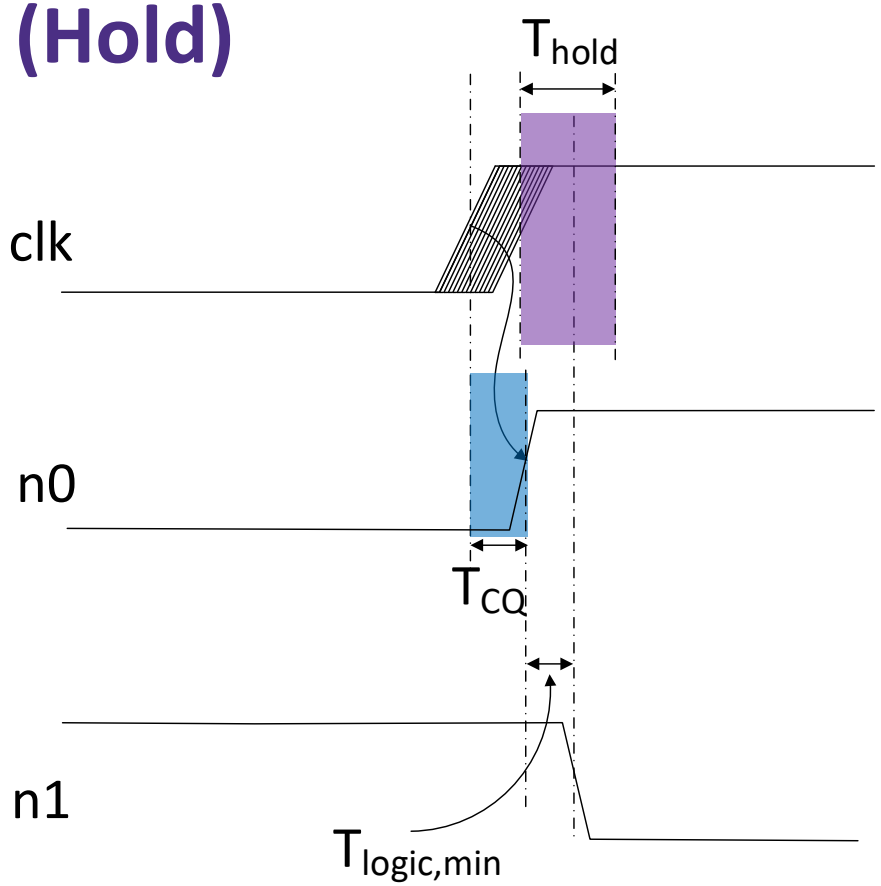
- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

Clock Uncertainty (Hold)



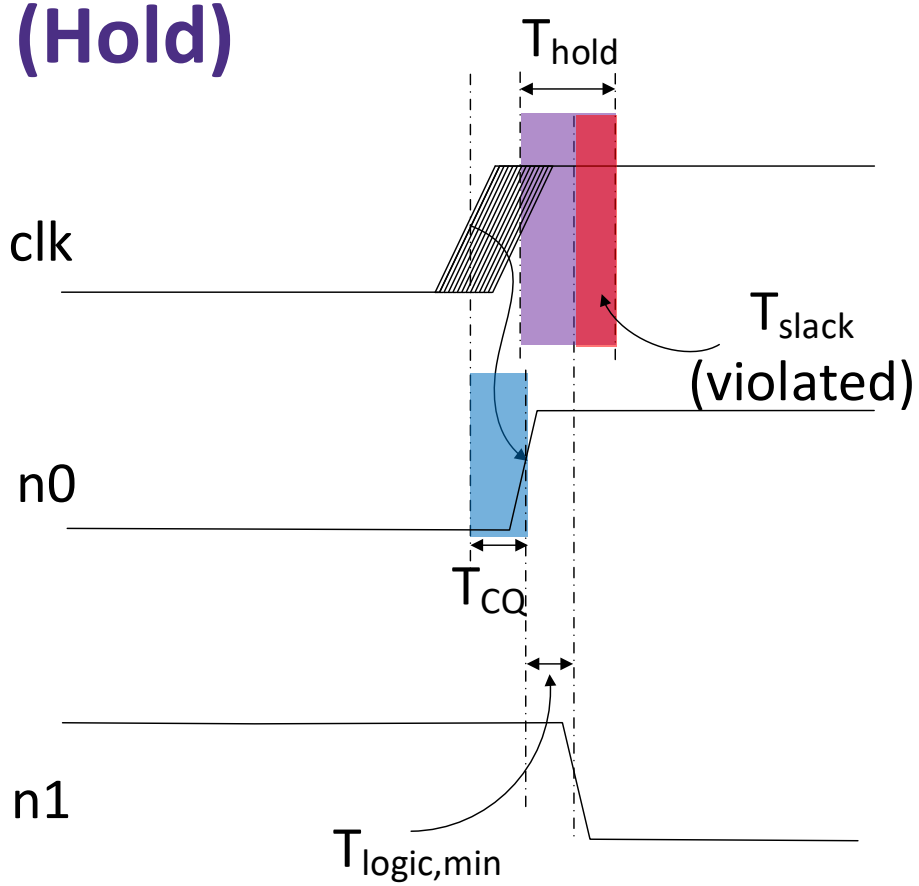
- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

Clock Uncertainty (Hold)



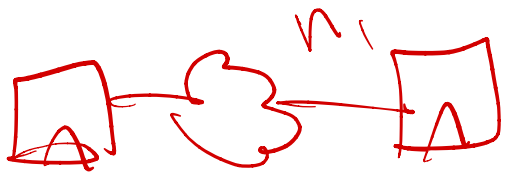
- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

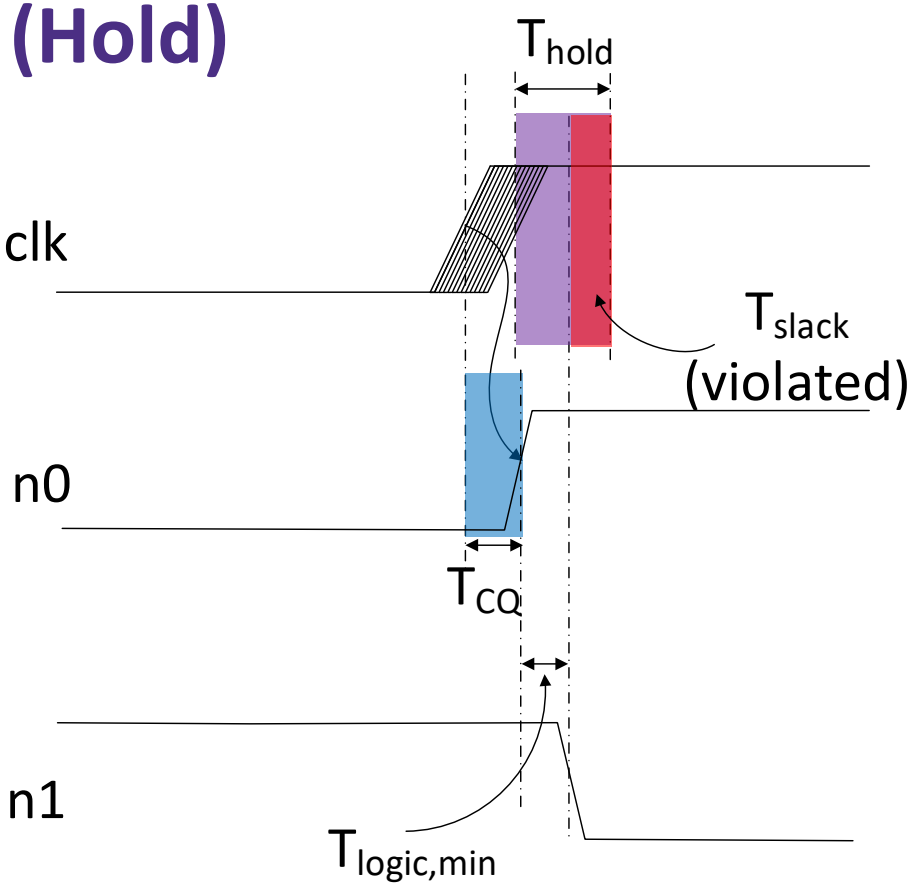
Clock Uncertainty (Hold)

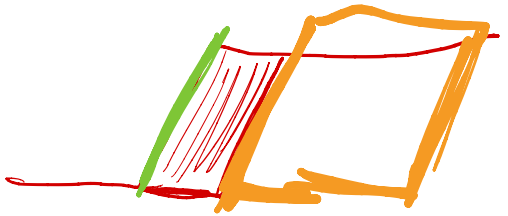


- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF

Clock Uncertainty (Hold)



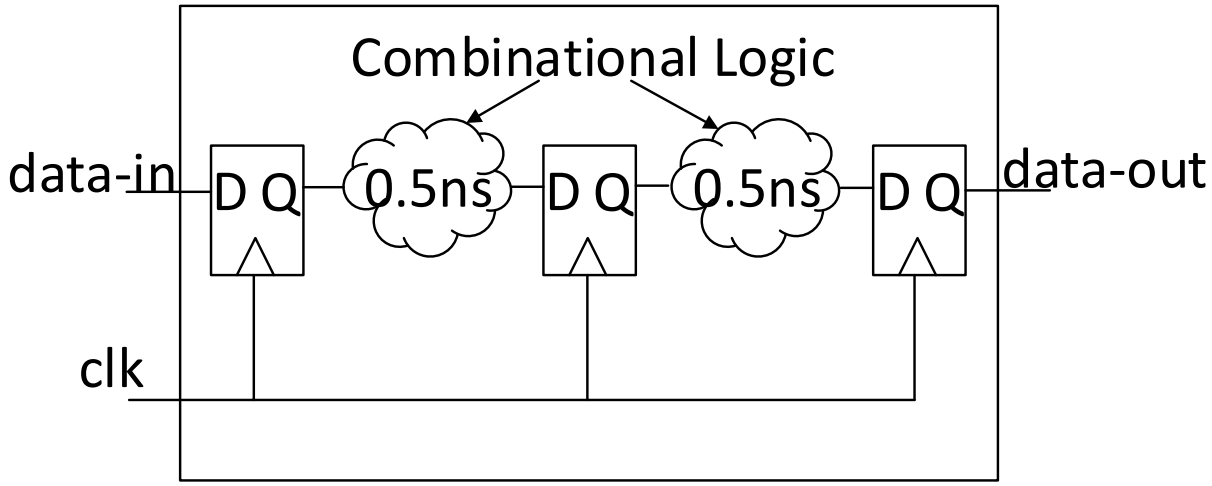




- Analysis examines susceptibility to hold-time violations in FFs
- Sets a minimum delay between launching clock the next FF's A.T
- Consider early arriving data at input of given FF
- $T_{hold} \leq T_{CQ} + T_{logic,min} - T_{clk,U}$

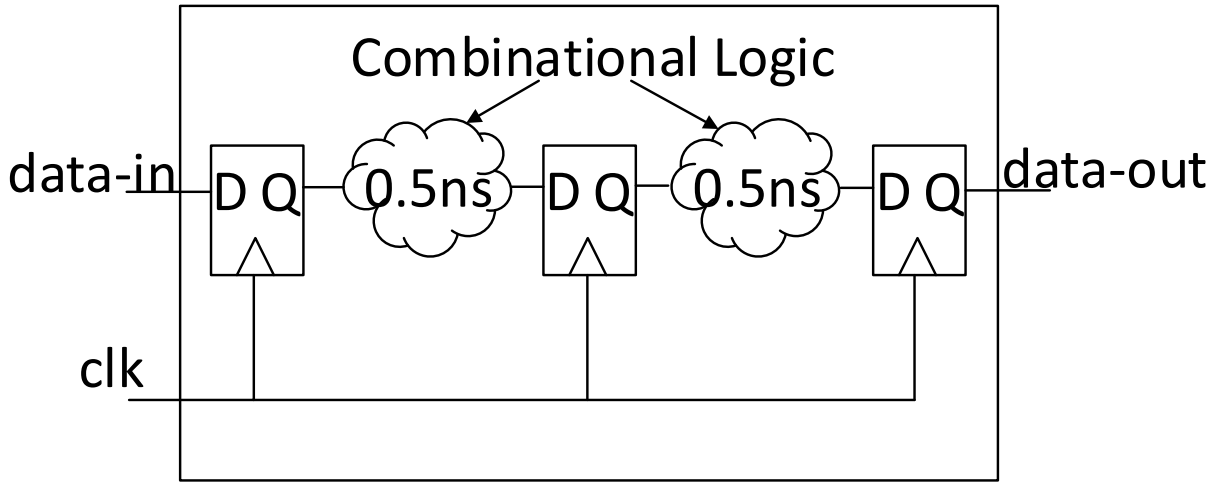
Exercise

- Consider the following pipeline design
  - Combinational logic is fine-grained (Can be divided into stages easily)
  - T_{setup} = 100ps, T_{cq} = 100ps, T_{hold} = 100ps, $T_{clk,U}$=50ps
- Determine current cycle time
- Determine $T_{logic,min}$ required for robust operation
- Determine cycle time if one extra pipeline stage were added.



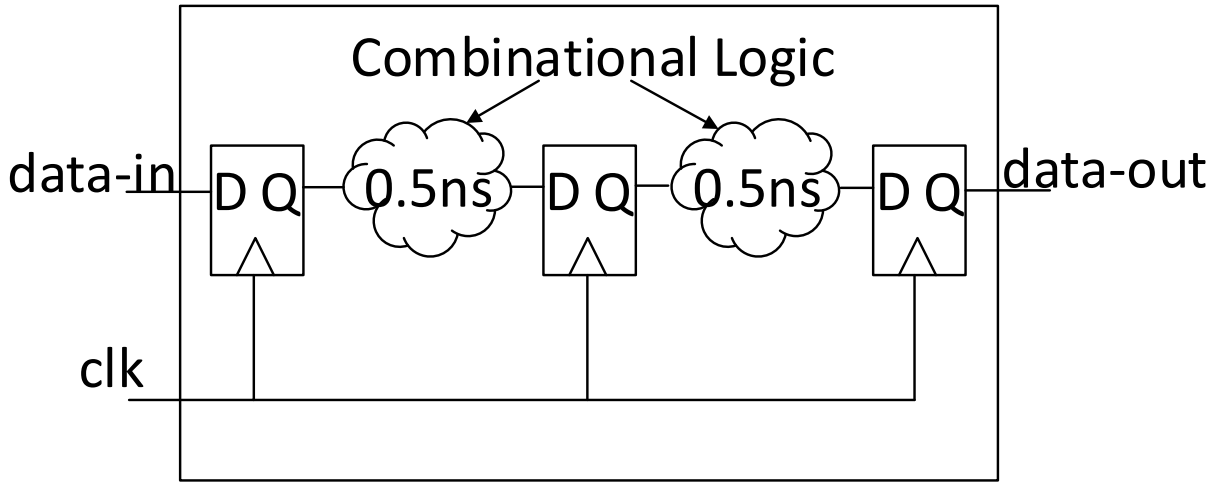
Exercise

- Consider the following pipeline design
  - Combinational logic is fine-grained (Can be divided into stages easily)
  - T_{setup} = 100ps, T_{cq} = 100ps, T_{hold} = 100ps, $T_{clk,U}$=50ps
- Determine current cycle time ($T_{cq} + T_{logic} + T_{setup} + T_{clk,U}$)
- Determine $T_{logic,min}$ required for robust operation
- Determine cycle time if one extra pipeline stage were added.



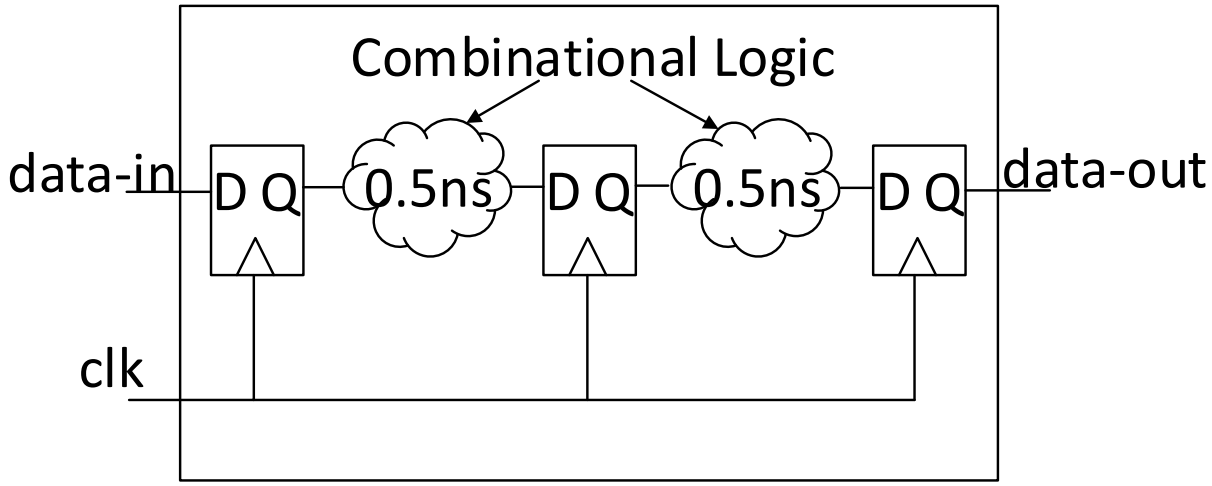
Exercise

- Consider the following pipeline design
  - Combinational logic is fine-grained (Can be divided into stages easily)
  - T_{setup} = 100ps, T_{cq} = 100ps, T_{hold} = 100ps, $T_{clk,U}$=50ps
- Determine current cycle time ($T_{cq} + T_{logic} + T_{setup} + T_{clk,U}$) = 750ps
- Determine $T_{logic,min}$ required for robust operation
- Determine cycle time if one extra pipeline stage were added.

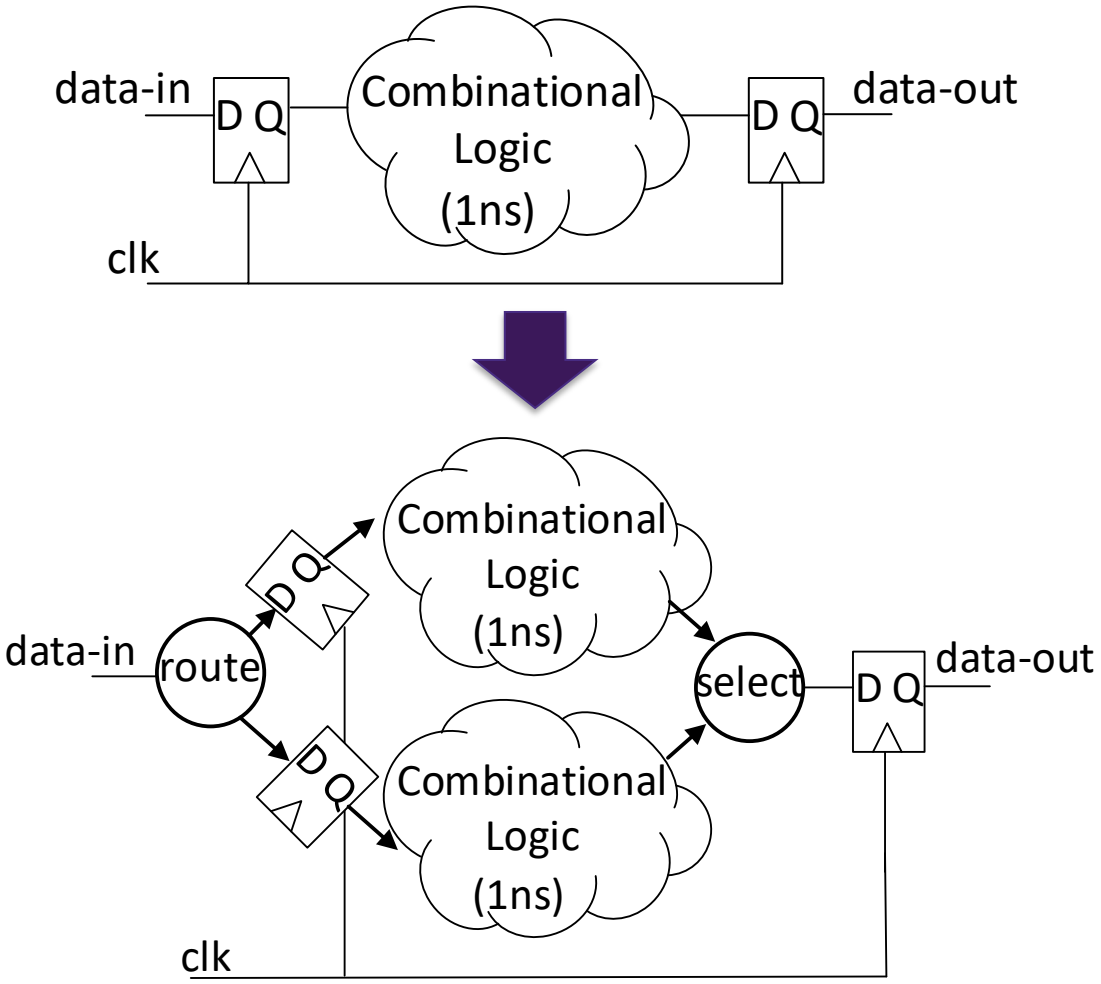


Exercise

- Consider the following pipeline design
  - Combinational logic is fine-grained (Can be divided into stages easily)
  - T_{setup} = 100ps, T_{cq} = 100ps, T_{hold} = 100ps, $T_{clk,U}$=50ps
- Determine current cycle time ($T_{cq} + T_{logic} + T_{setup} + T_{clk,U}$) = 750ps
- Determine $T_{logic,min}$ required for robust operation $T_{hold} - T_{cq} + T_{clk,U}$=50p
- Determine cycle time if one extra pipeline stage were added.

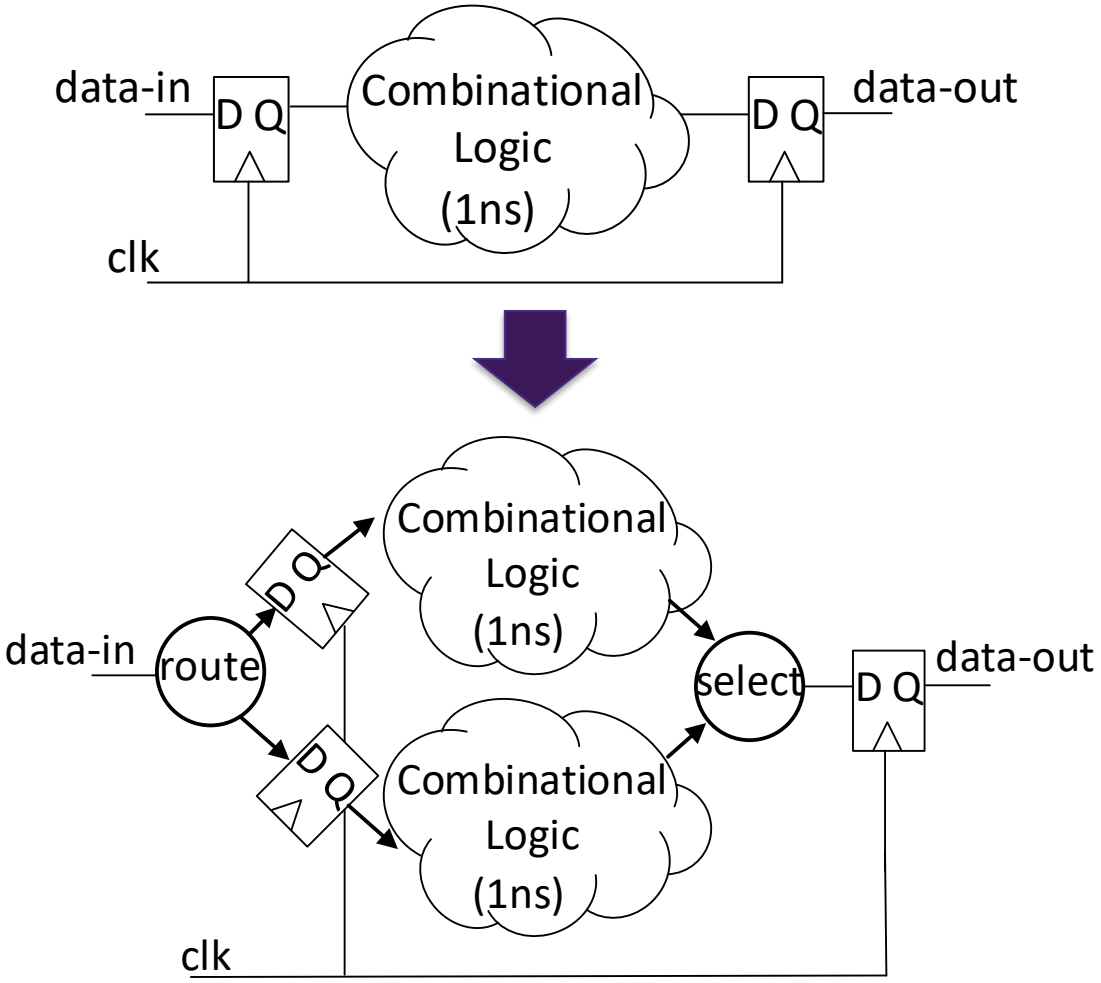


Parallel Processing



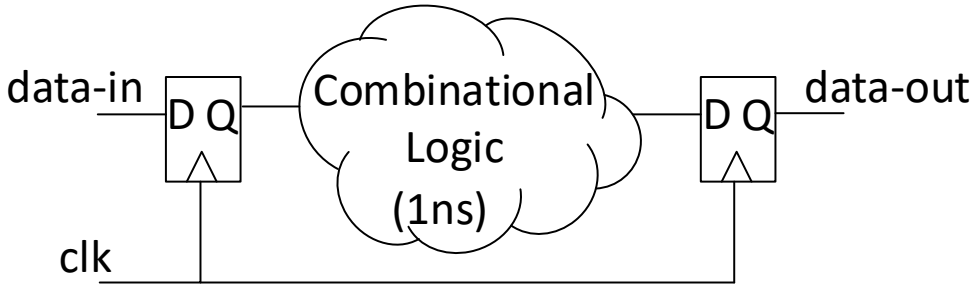
- An alternative approach to enhanced processing capability
- Conduct multiple computations in parallel. Combine results

Parallel Processing



- An alternative approach to enhanced processing capability
- Conduct multiple computations in parallel. Combine results
- There's a bug here …

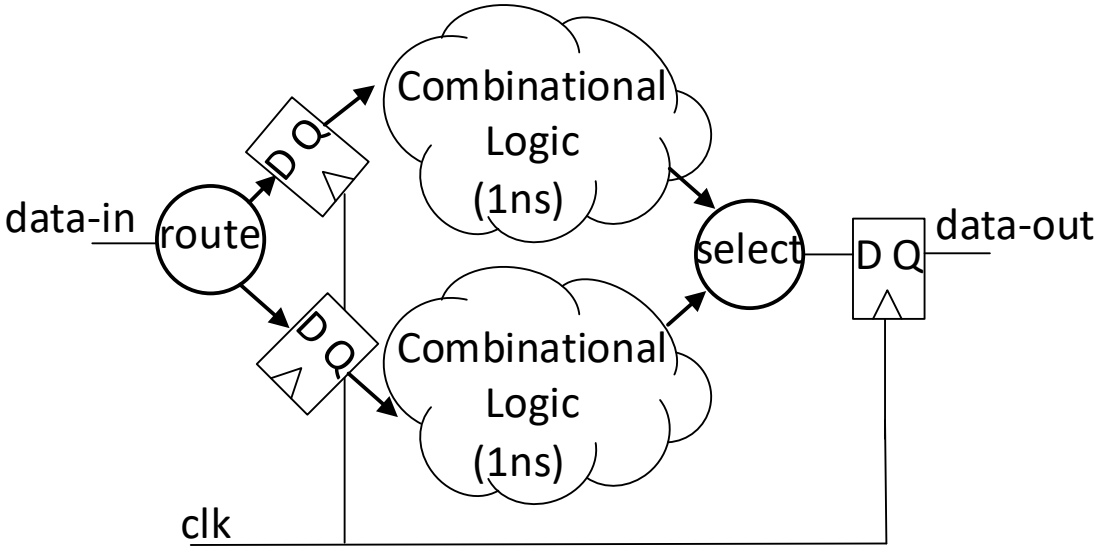
Parallel Processing



$\frac{1}{2} C V_{dd}^2 f s$

$f \rightarrow \frac{f}{2}$

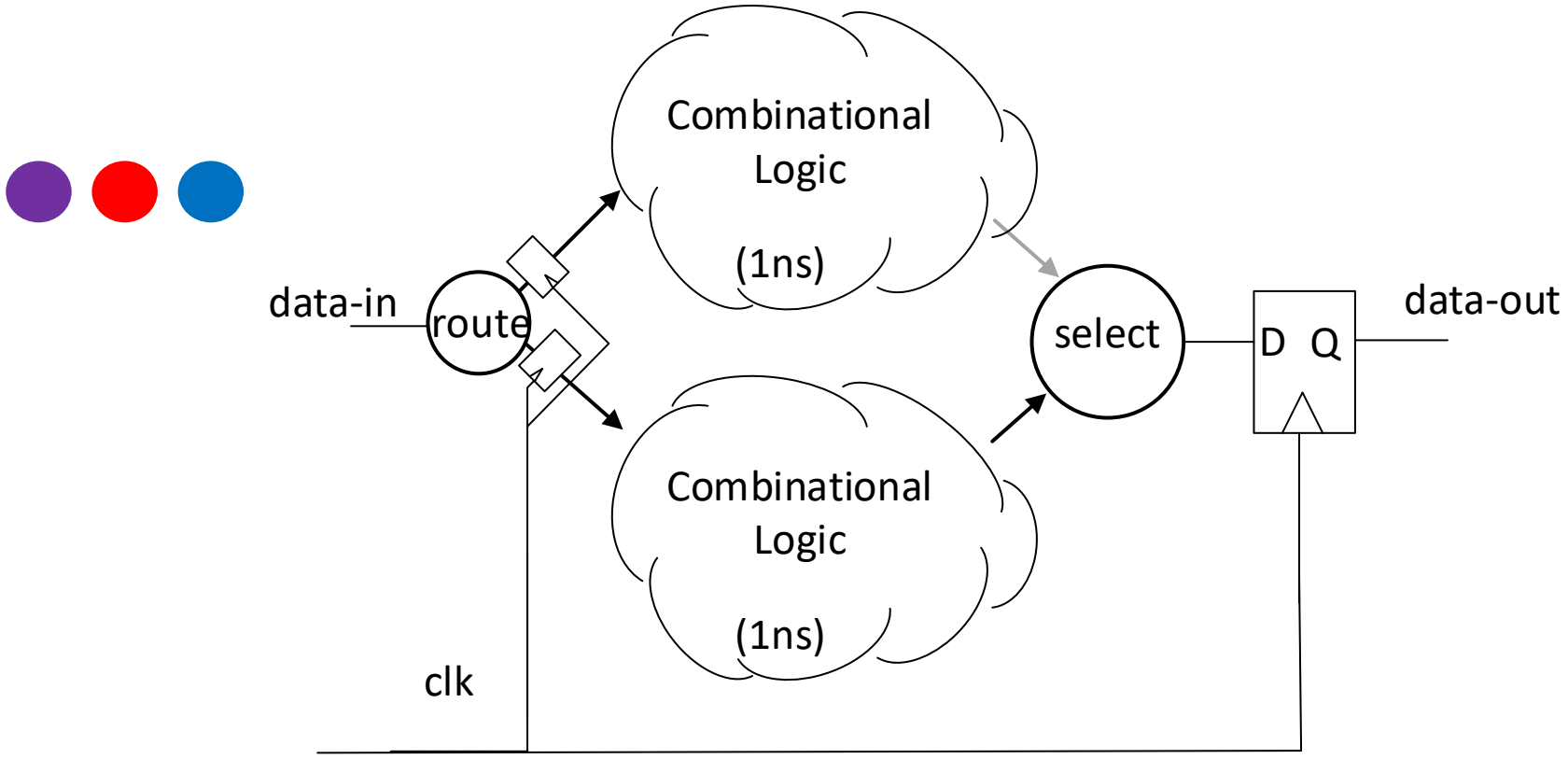
$C \rightarrow 2C$



Equal intervals
between channel change?

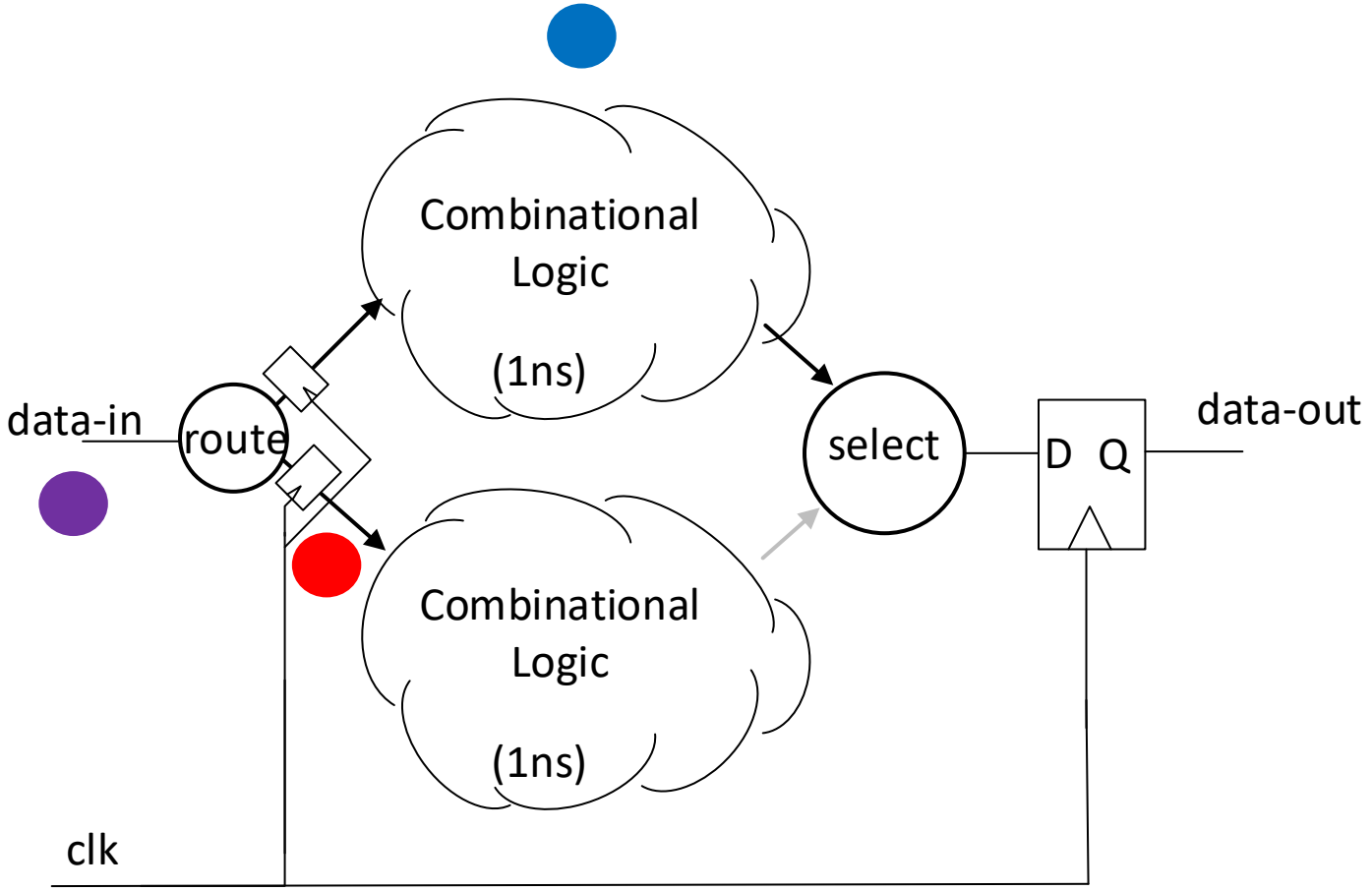
- An alternative approach to enhanced processing capability
- Conduct multiple computations in parallel. Combine results
- There's a bug here …

Throughput not determined only be combinational logic latency



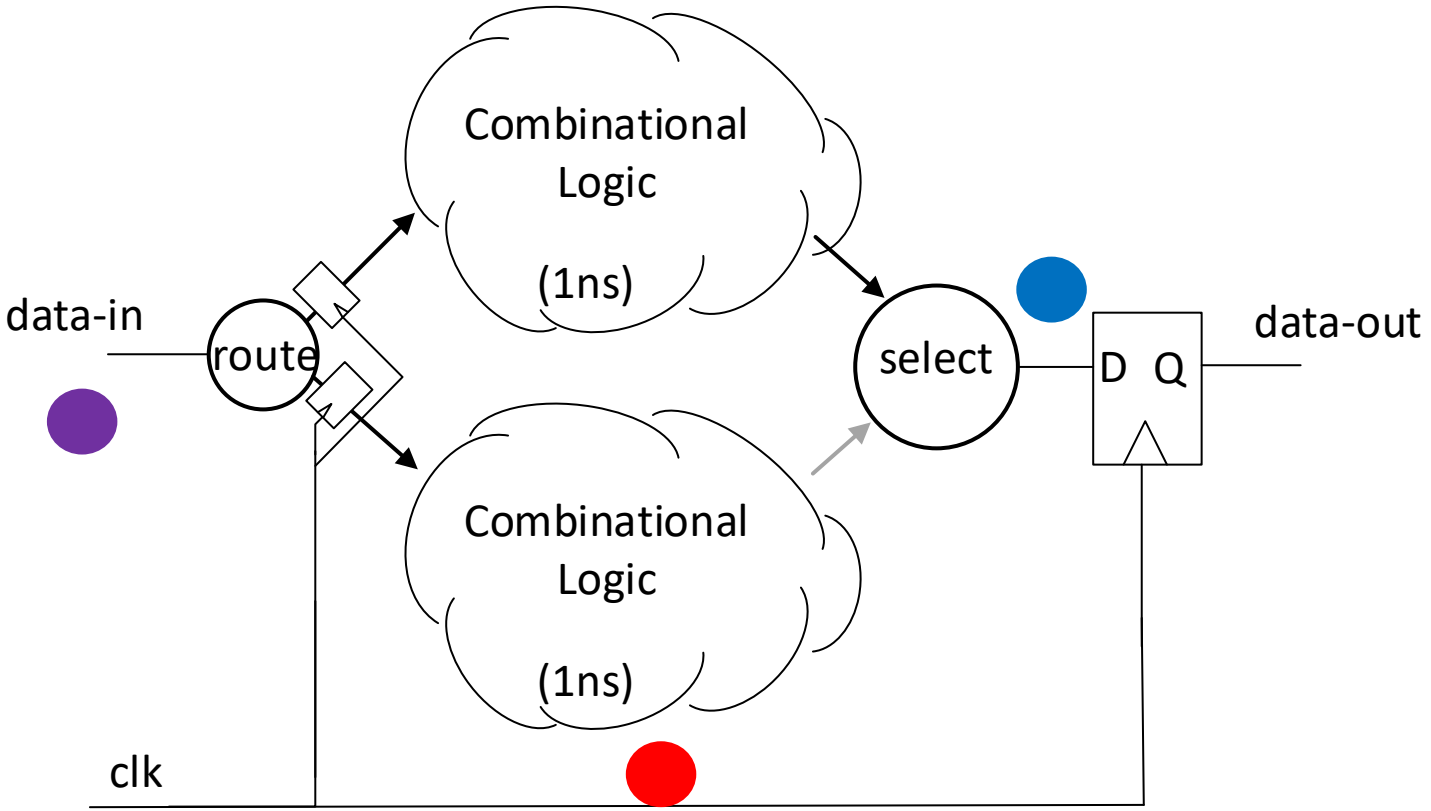
- Throughput not determined only be combinational logic latency

Parallel Processing : Walkthrough

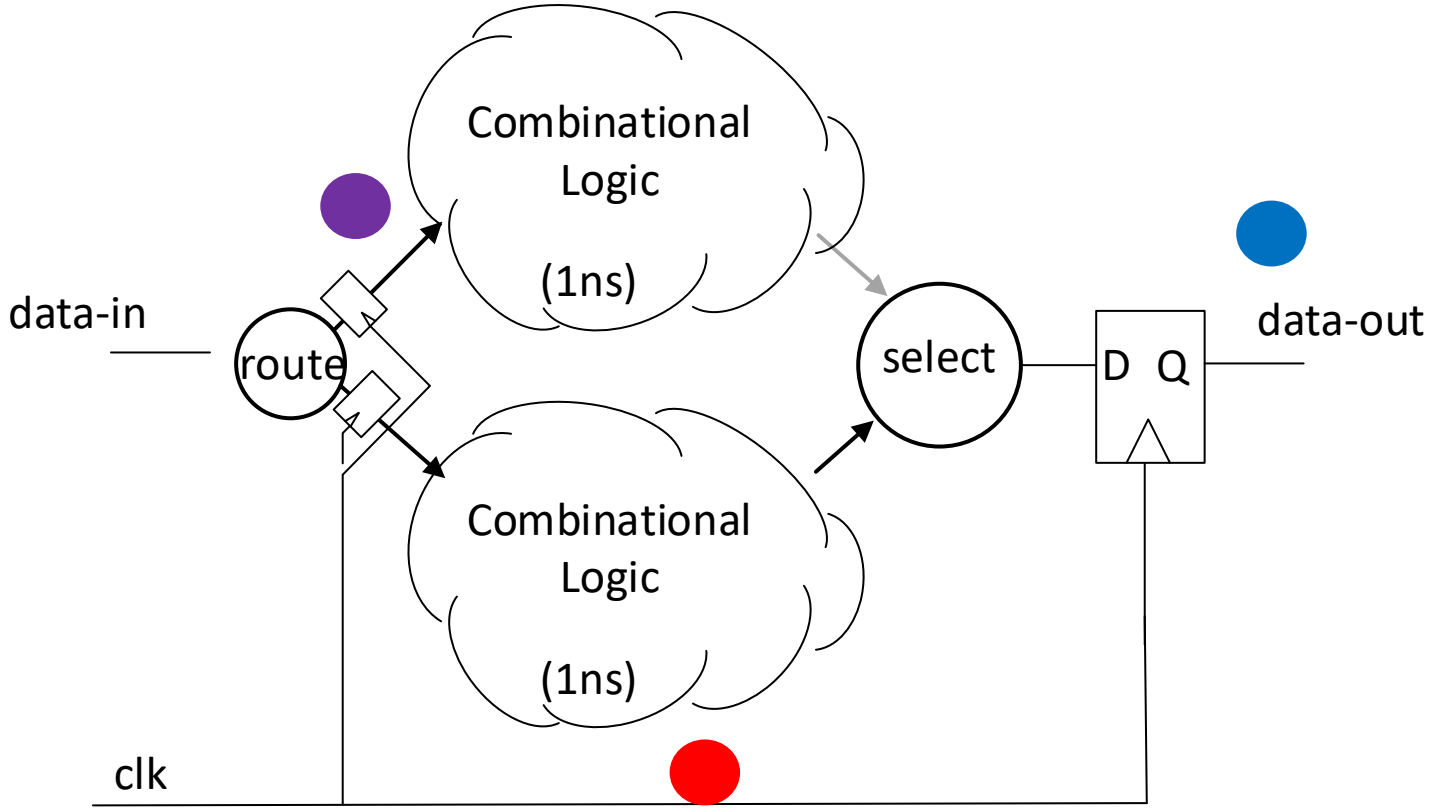


- After ~0.5ns, blue computation still being conducted
- Launch red computation through the other parallel logic

Parallel Processing : Walkthrough



Parallel Processing : Walkthrough



- At T~1ns, blue computation completed
- Ready to launch purple computation
- Red computation still being performed.

Parallel Processing : Gains

- Exercise: Given T_{logic}=1ns, T_{router} = 100ps, T_{select} = 100ps
  - What is the speedup of a 2-way parallel processing computation.

Parallel Processing Limitations

- Inter-computation dependencies
  - Many computations depend on each other (esp. microprocessors)
  - Parallel processing: Send, forget for n cycles, capture
  - Does not provide a mechanism for handling such dependencies

- Other limitations
  - Increased cost (die area)
  - Increased leakage power

Reading assignment

- W&H : 10.1-10.2.3
- Extra Reading: 10.3