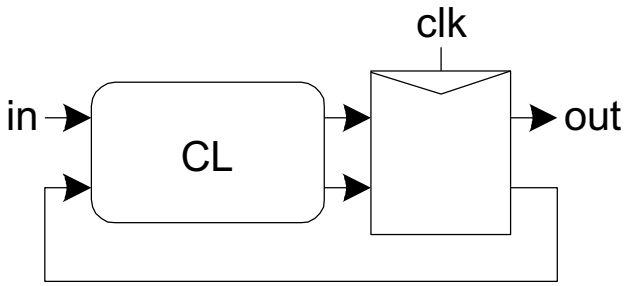
Lecture 10:
Sequential Elements (Latches and Flip Flops)

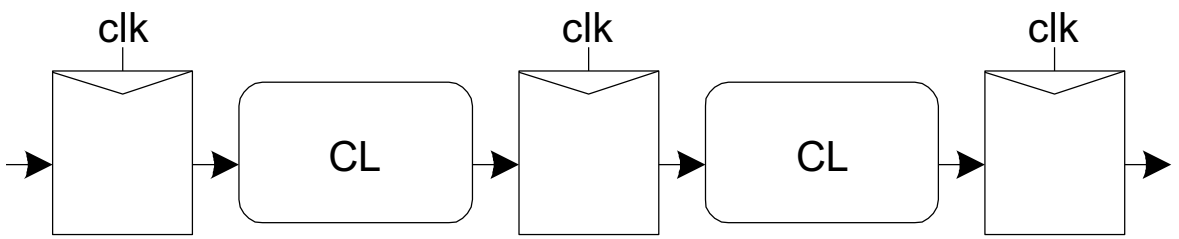
Mark McDermott
Electrical and Computer Engineering
The University of Texas at Austin

Sequencing

- *Combinational logic (CL)*
  - output depends on current inputs
- *Sequential logic*
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called *state* or *tokens*
  - Ex: FSM, pipeline



Finite State Machine



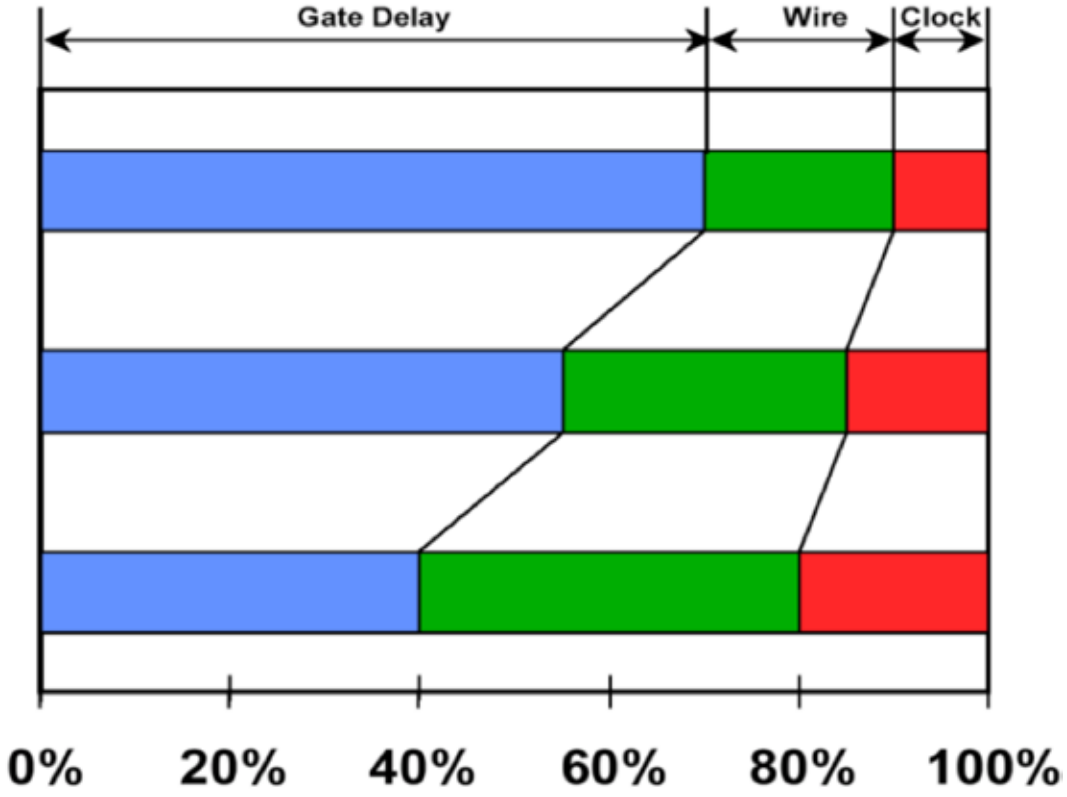
Pipeline

Sequencing (cont)

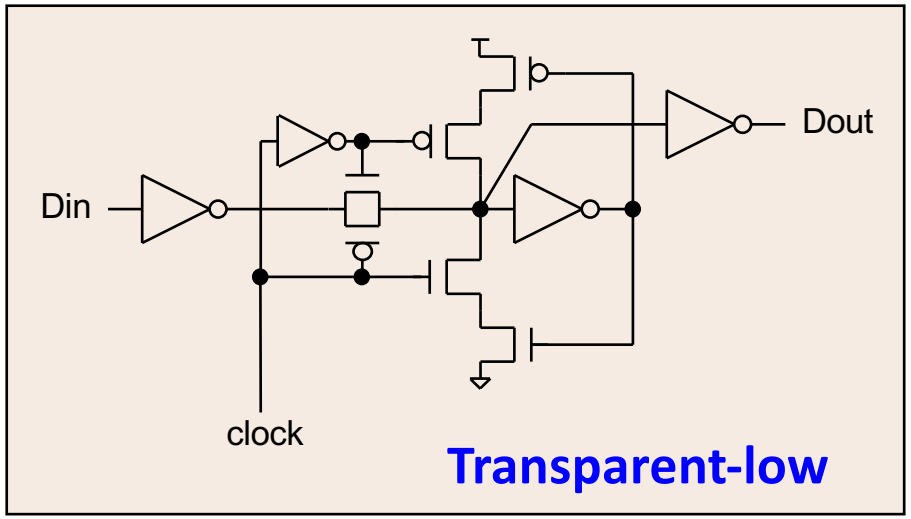
- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

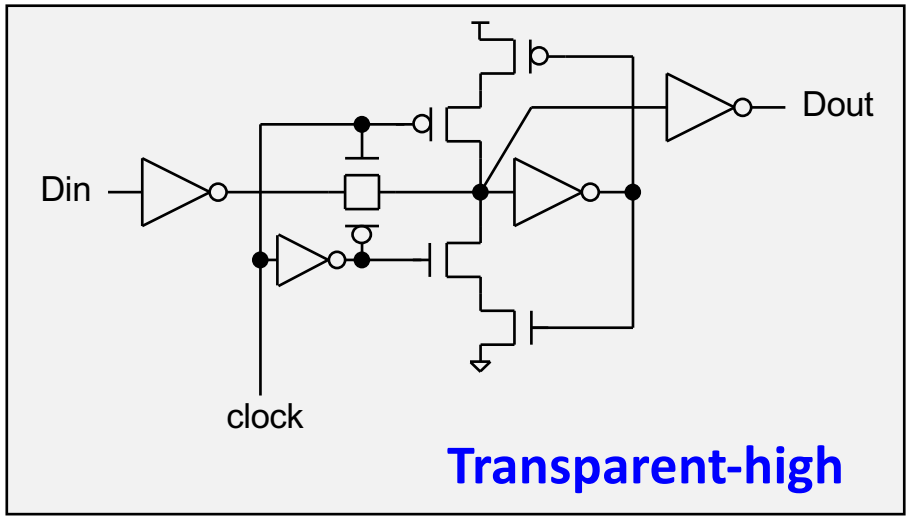
Sequencing Overhead

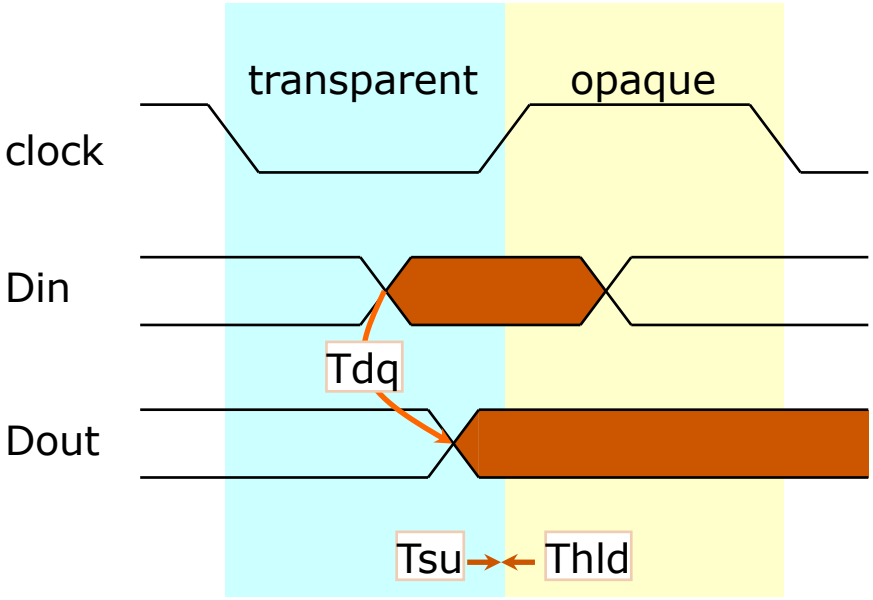
- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence

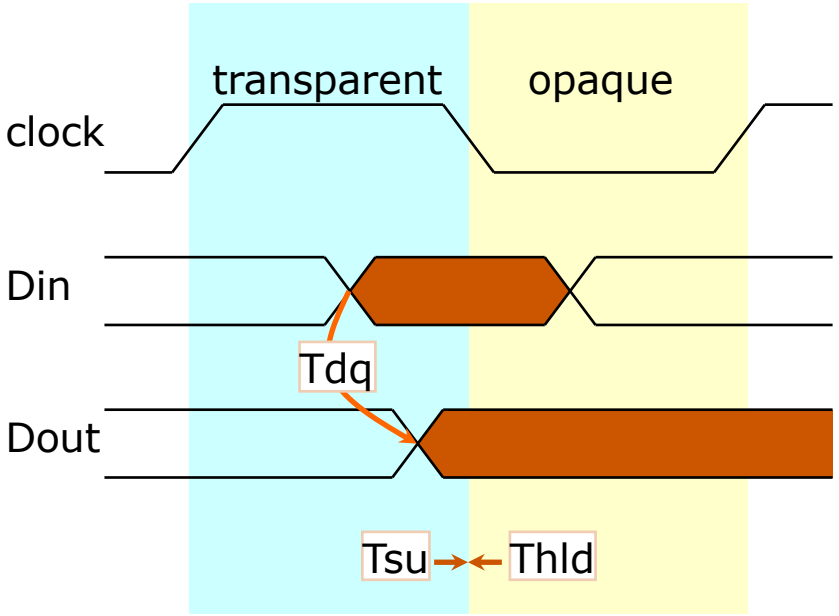


Basic LATCH Operation

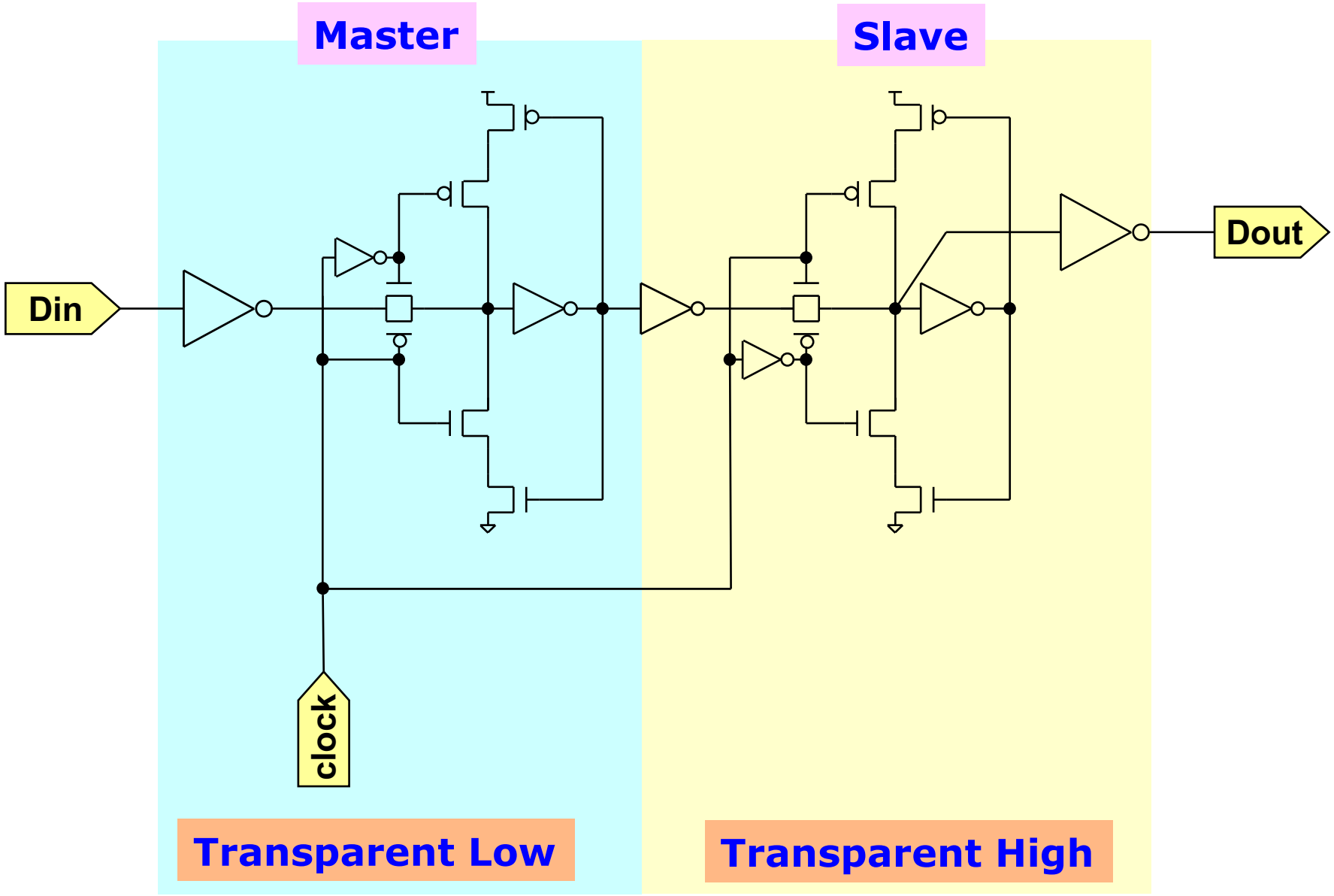








Building a Flip-Flop with Two Latches



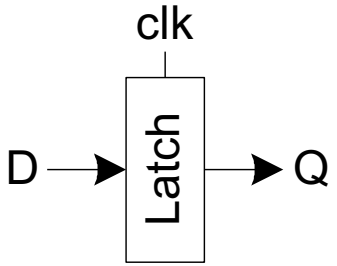
Difference between a Latch and a Flip-Flop

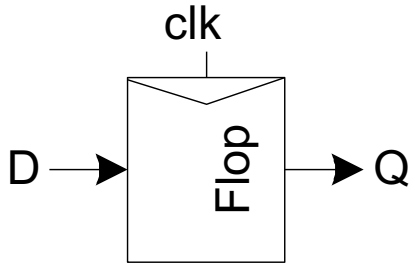
Latch: Level sensitive

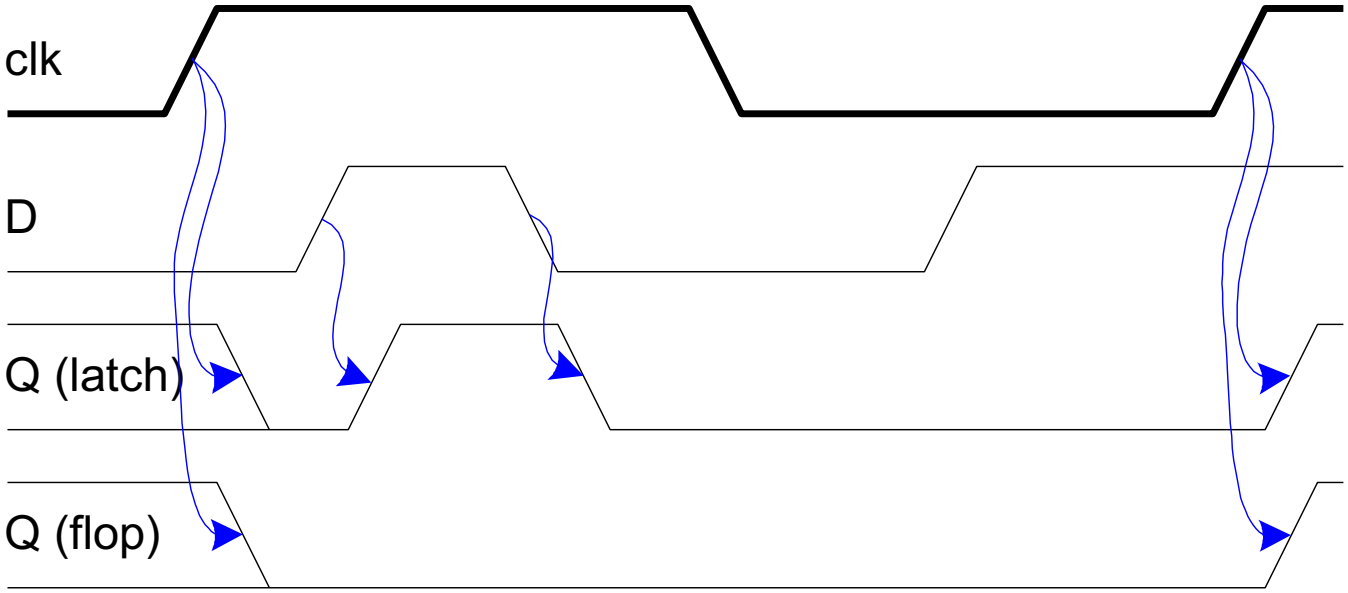
a.k.a. transparent latch, D latch

Flip-flop: Edge triggered

a.k.a. master-slave flip-flop, D flip-flop, D register, FLOP

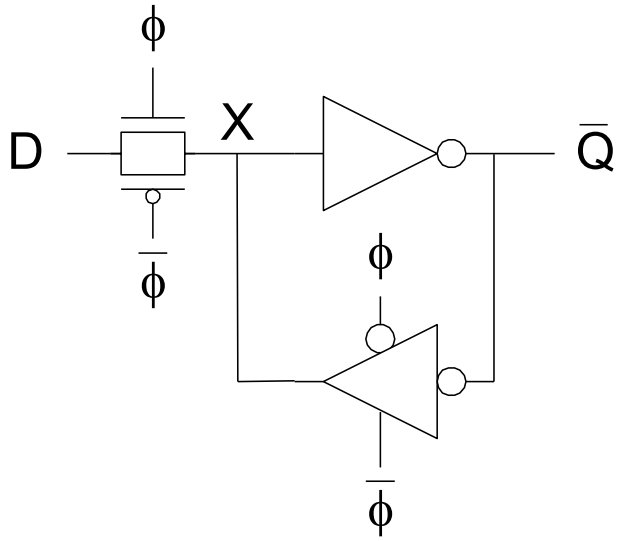




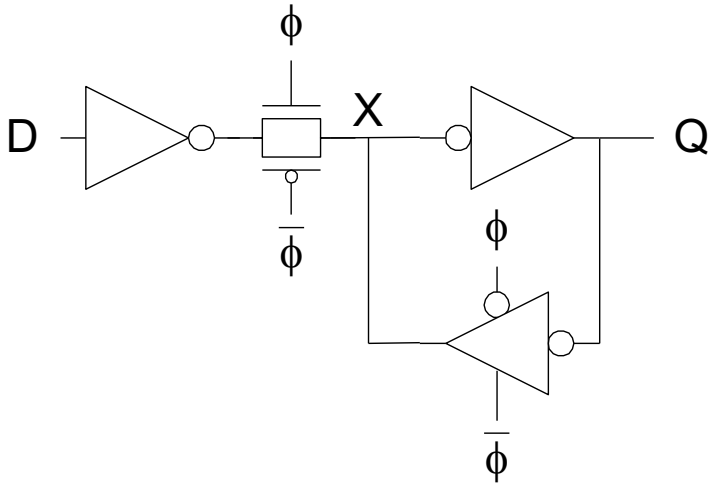


Static Latch Design

- Static latches are essential for DSM applications
- Tristate feedback
  - \+ Static
  - – Backdriving risk
  - – Susceptible to noise on D input



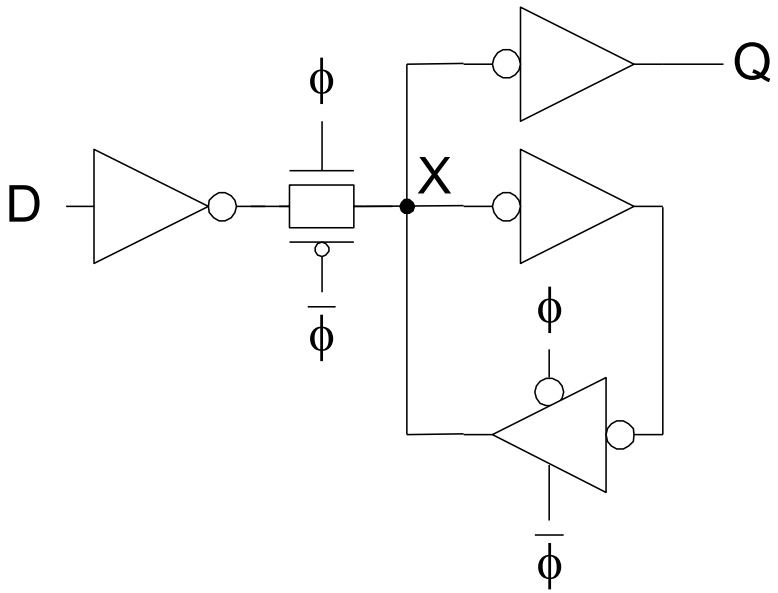
- Buffered input
  - \+ Fixes diffusion input
  - \+ Noninverting



Static Latch Design (cont.)

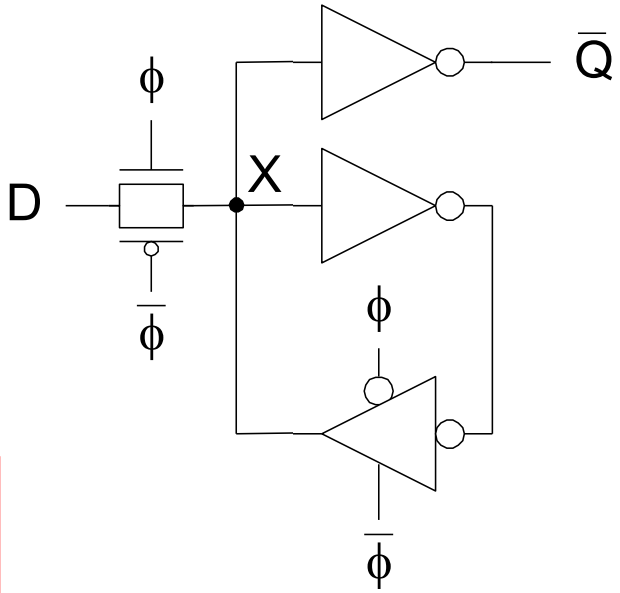
- **Buffered output**
  - + No back driving

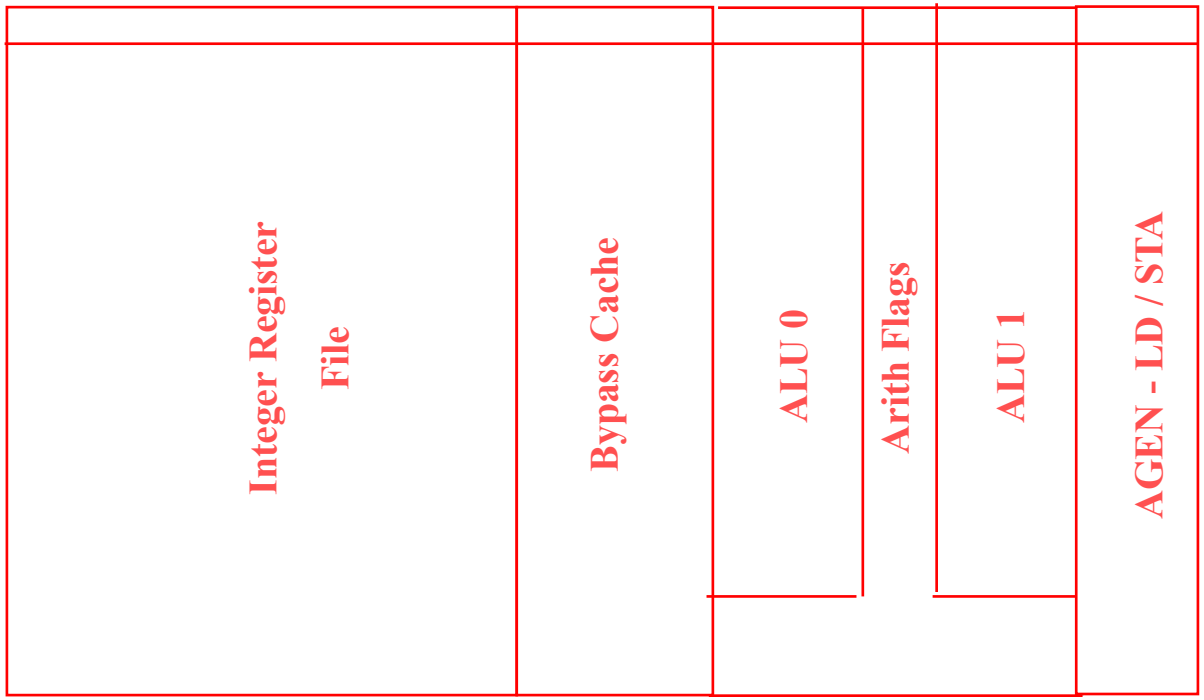
- **Widely used in standard cells**
  - + Very robust (most important)
  - - Rather large
  - - Rather slow (1.5 – 2 FO4 delays)
  - - High clock loading



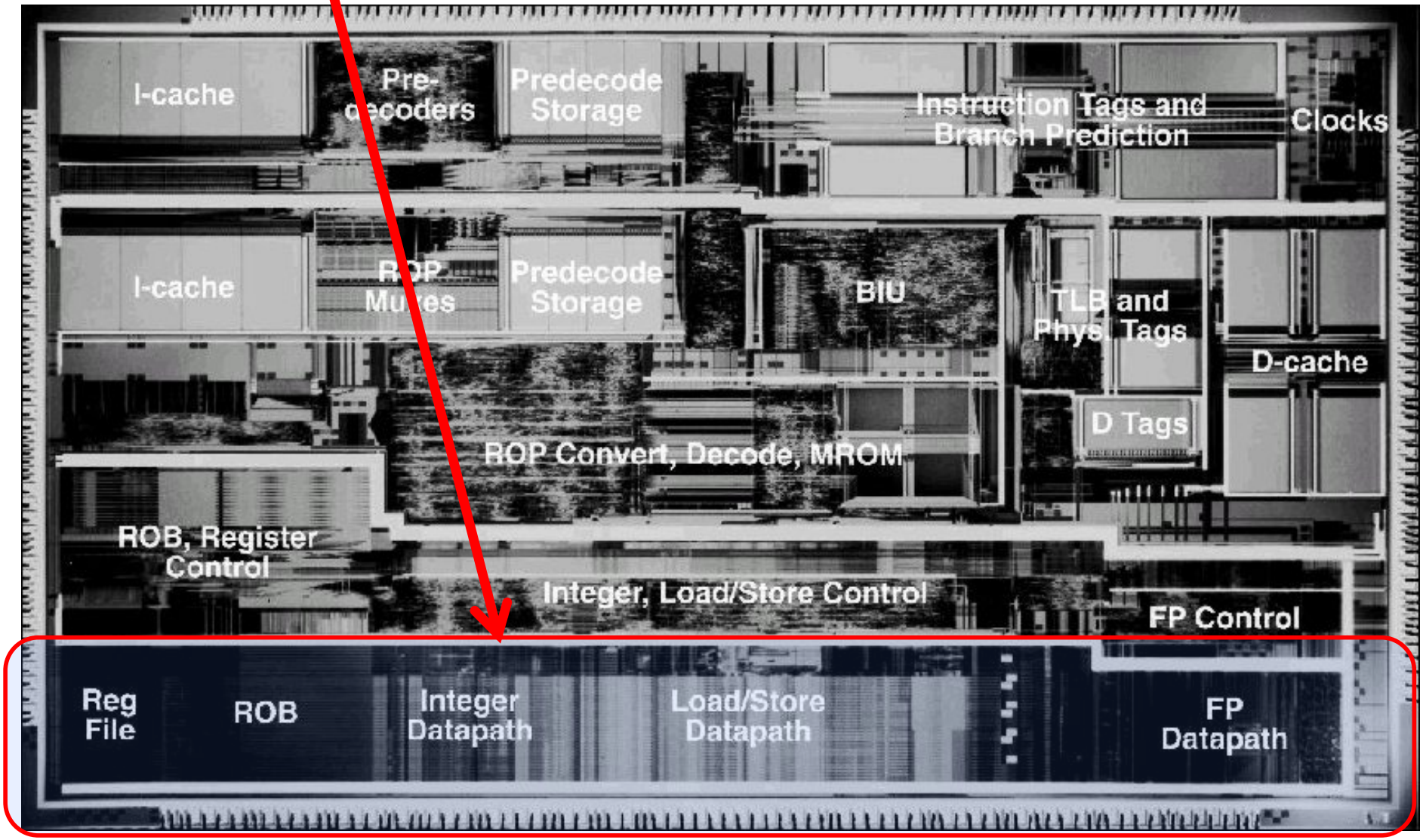
Static Latch Design (cont.)

- **Datapath latch**
  - + Smaller, faster
  - - un-buffered input -> okay for datapaths



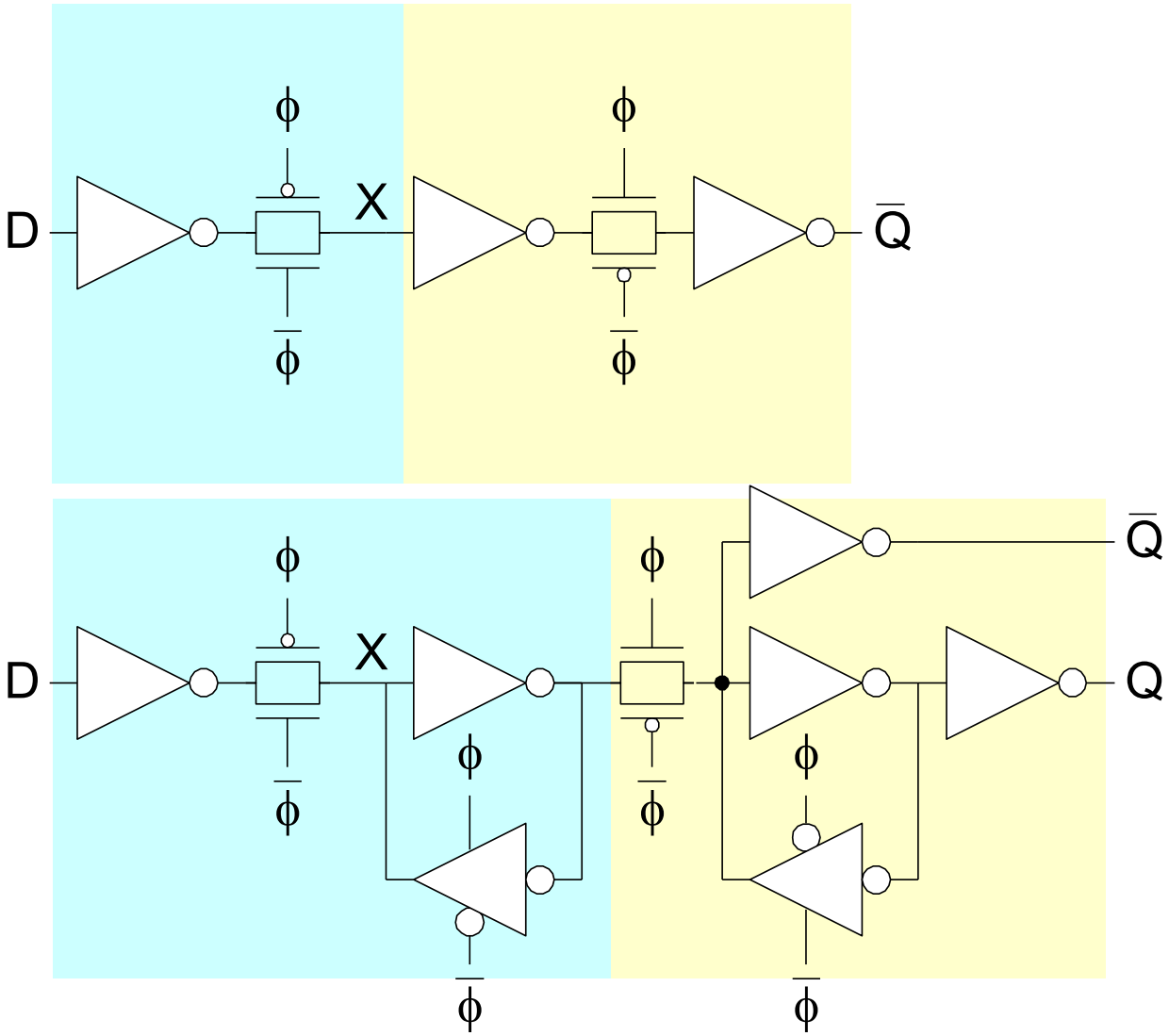


AMD K5: Datapath



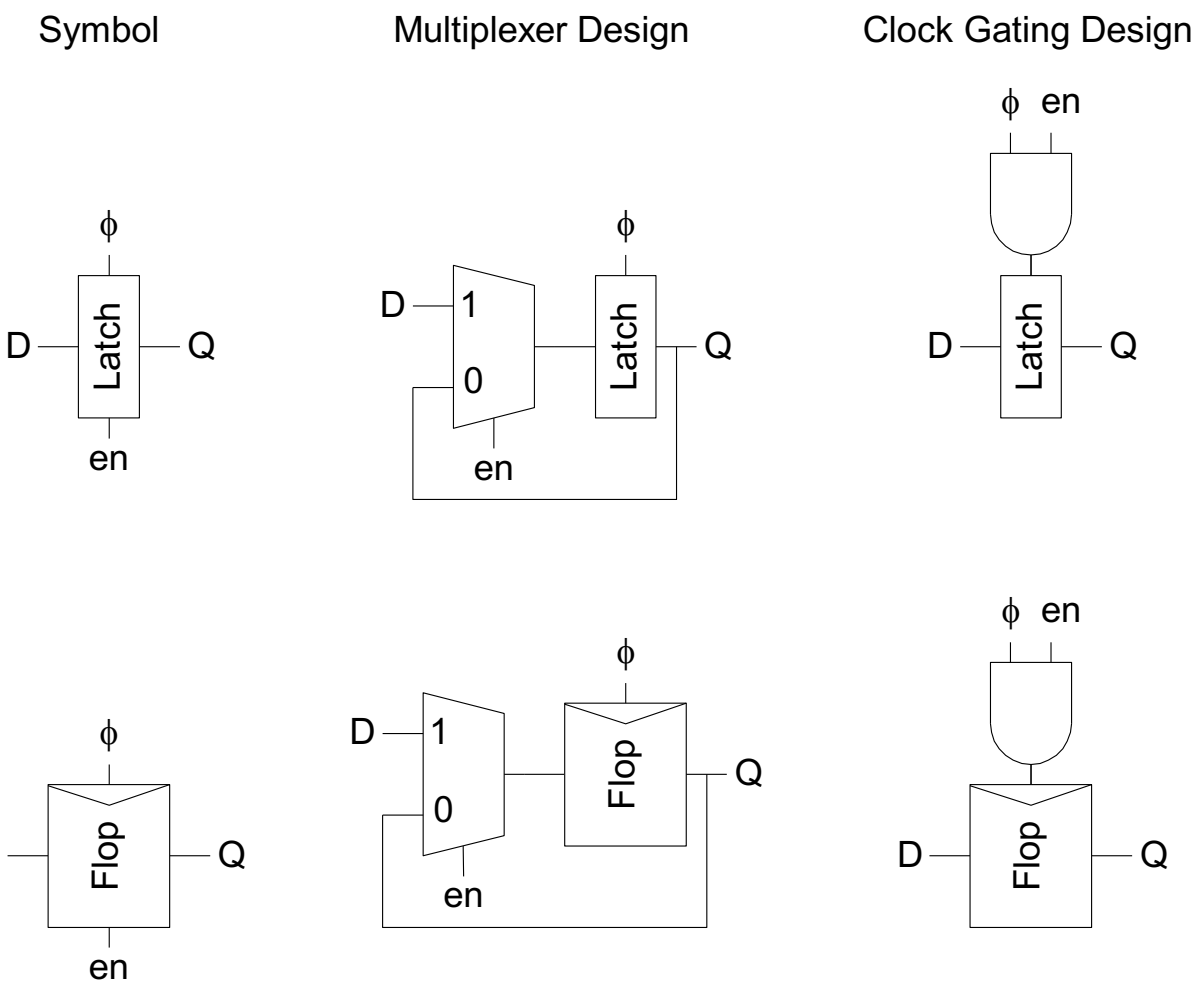
Flip-Flop Designs

- **Flip-flop is built as pair of back-to-back latches**



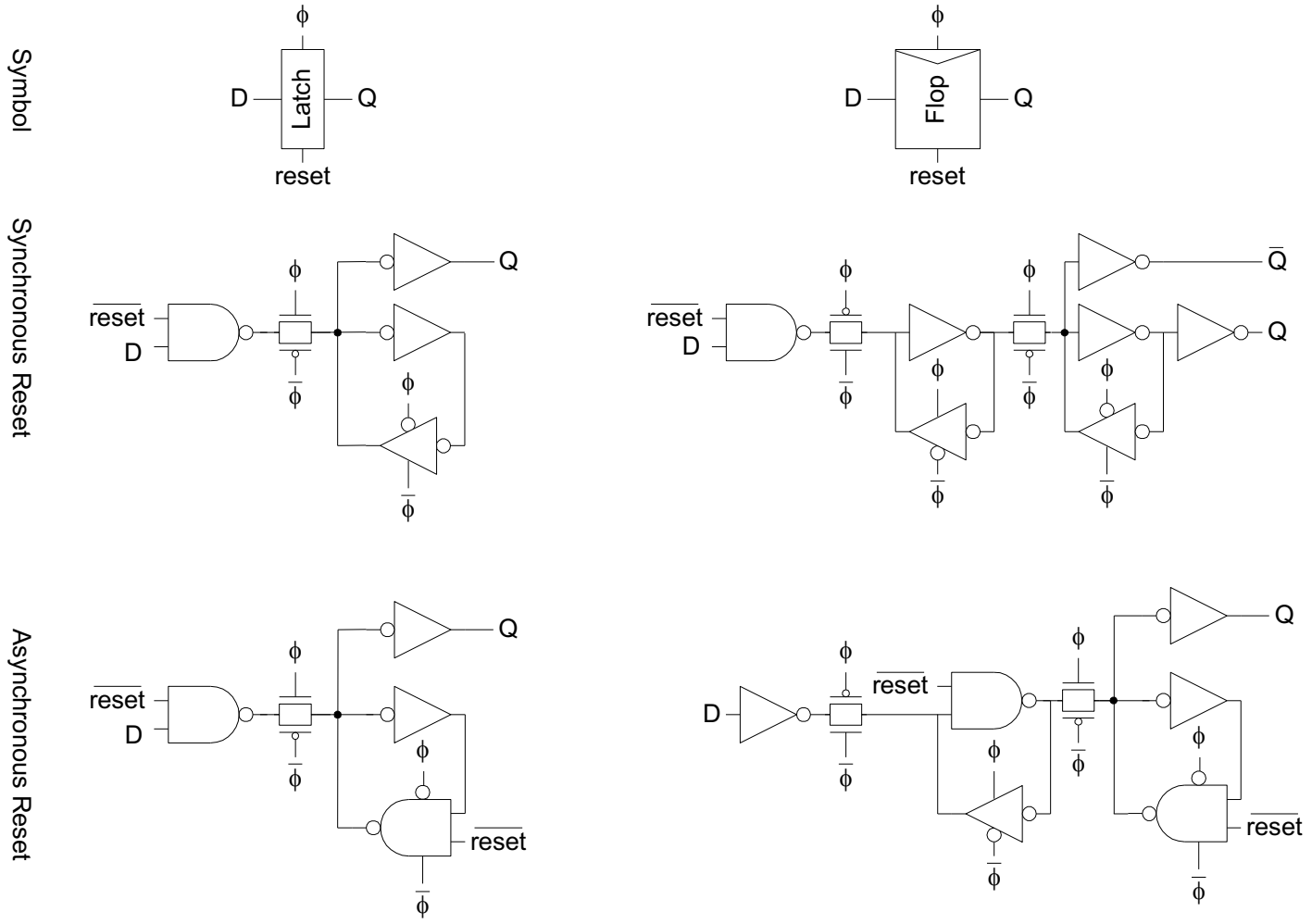
Clock Enable

- **Clock Enable: ignore clock when en = 0**
  - **Mux: increases latch D-Q delay**
  - **Clock Gating: increase enable setup time, skew**



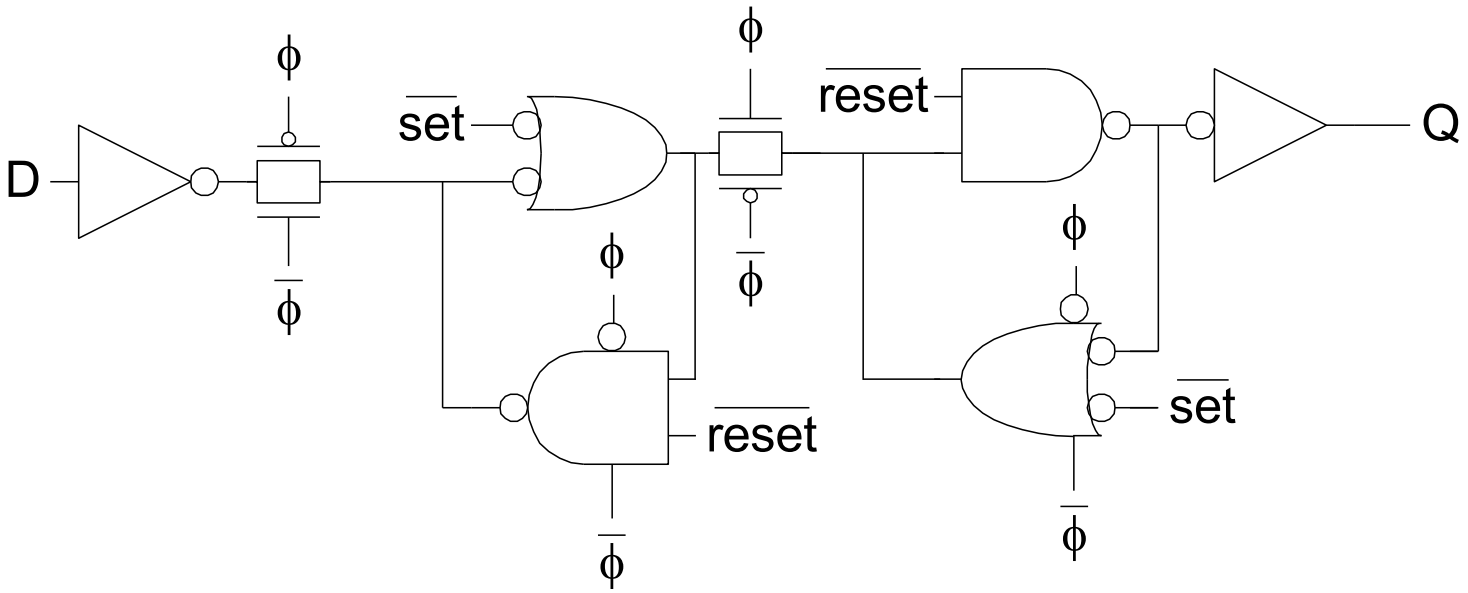
Reset

- Force output low when reset asserted
- Synchronous vs. asynchronous



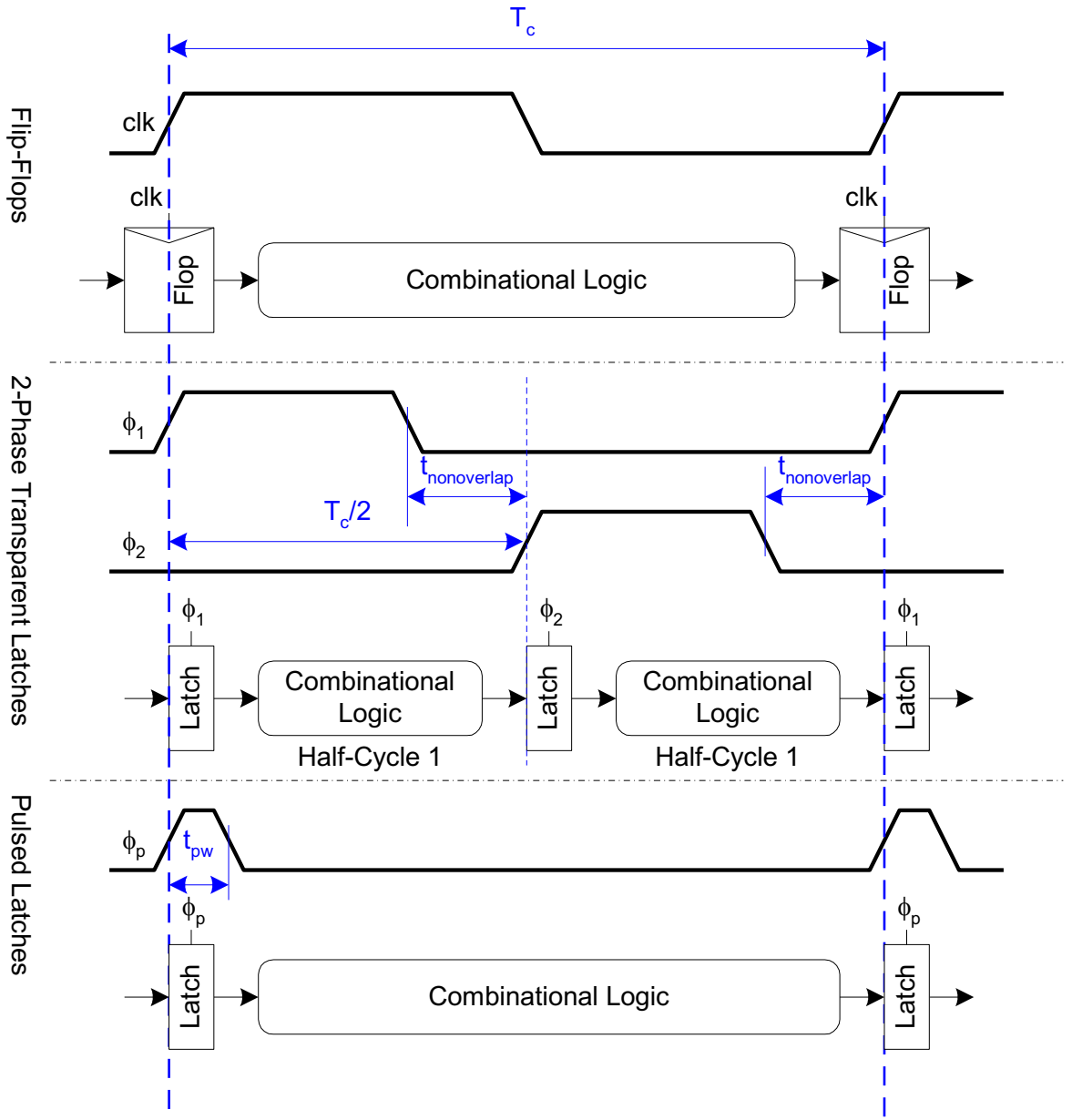
Set / Reset

- Set forces output high when enabled
- Flip-flop with asynchronous set and reset



Sequencing Methods

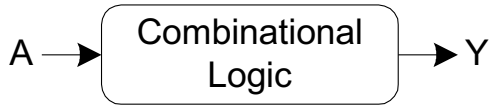
- **Flip-flops**
- **2-Phase Latches**
- **Pulsed Latches**

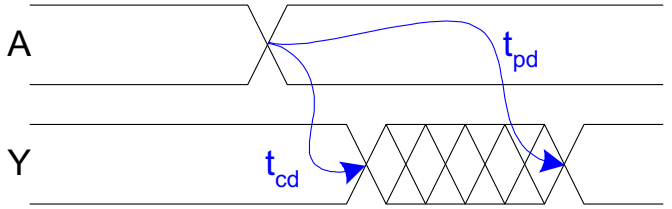


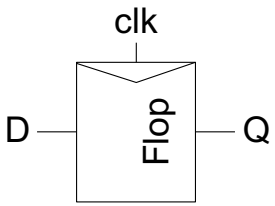
Timing Diagrams

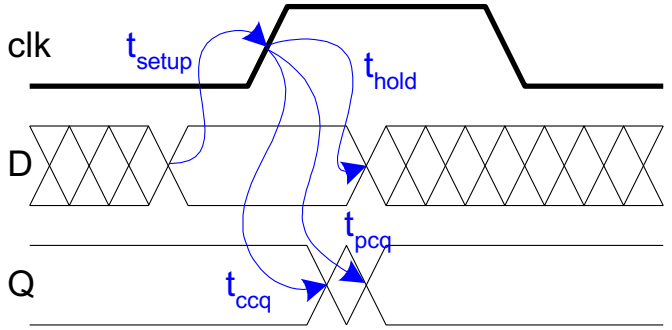
Contamination and Propagation Delays

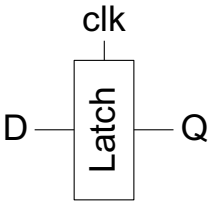
t_{pd}	Logic Propagation Delay
t_{cd}	Logic Contamination Delay
t_{pcq}	Latch/Flop Clk-Q Prop Delay
t_{ccq}	Latch/Flop Clk-Q Cont. Delay
t_{pdq}	Latch D-Q Prop Delay
t_{cdq}	Latch D-Q Cont. Delay
t_{setup}	Latch/Flop Setup Time
t_{hold}	Latch/Flop Hold Time

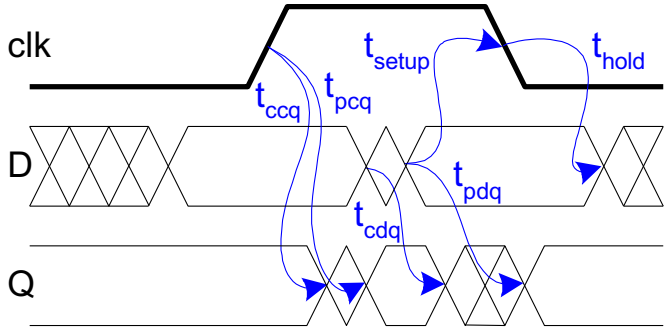






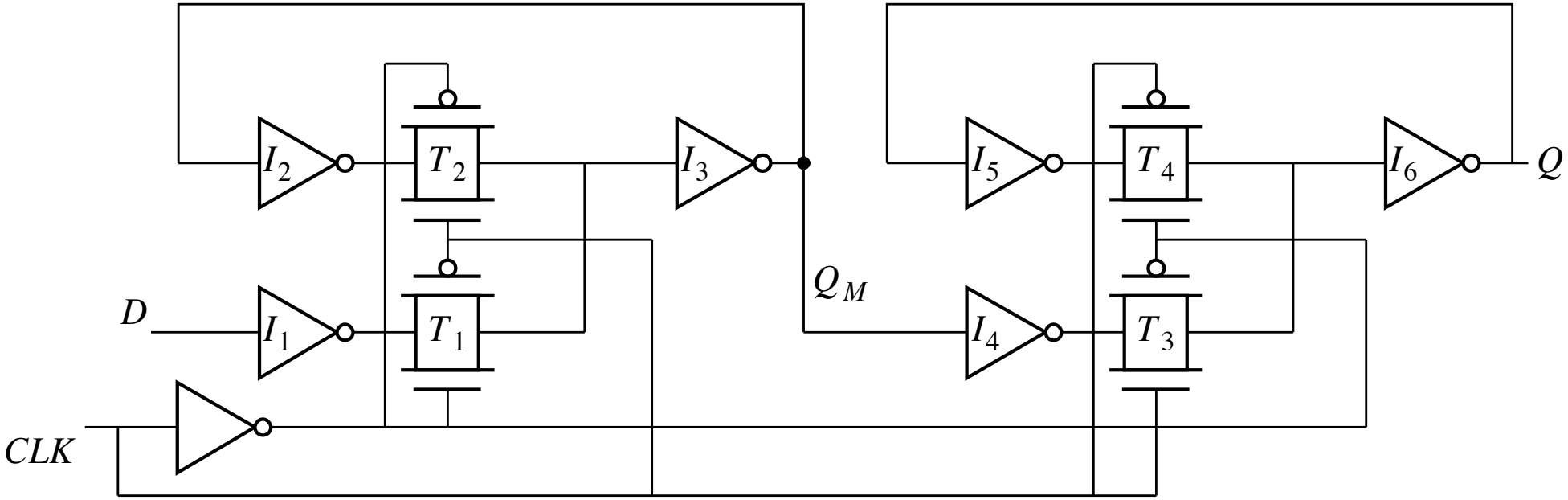






Master-Slave Flip-Flop

Illustration of delays



t_{setup} =

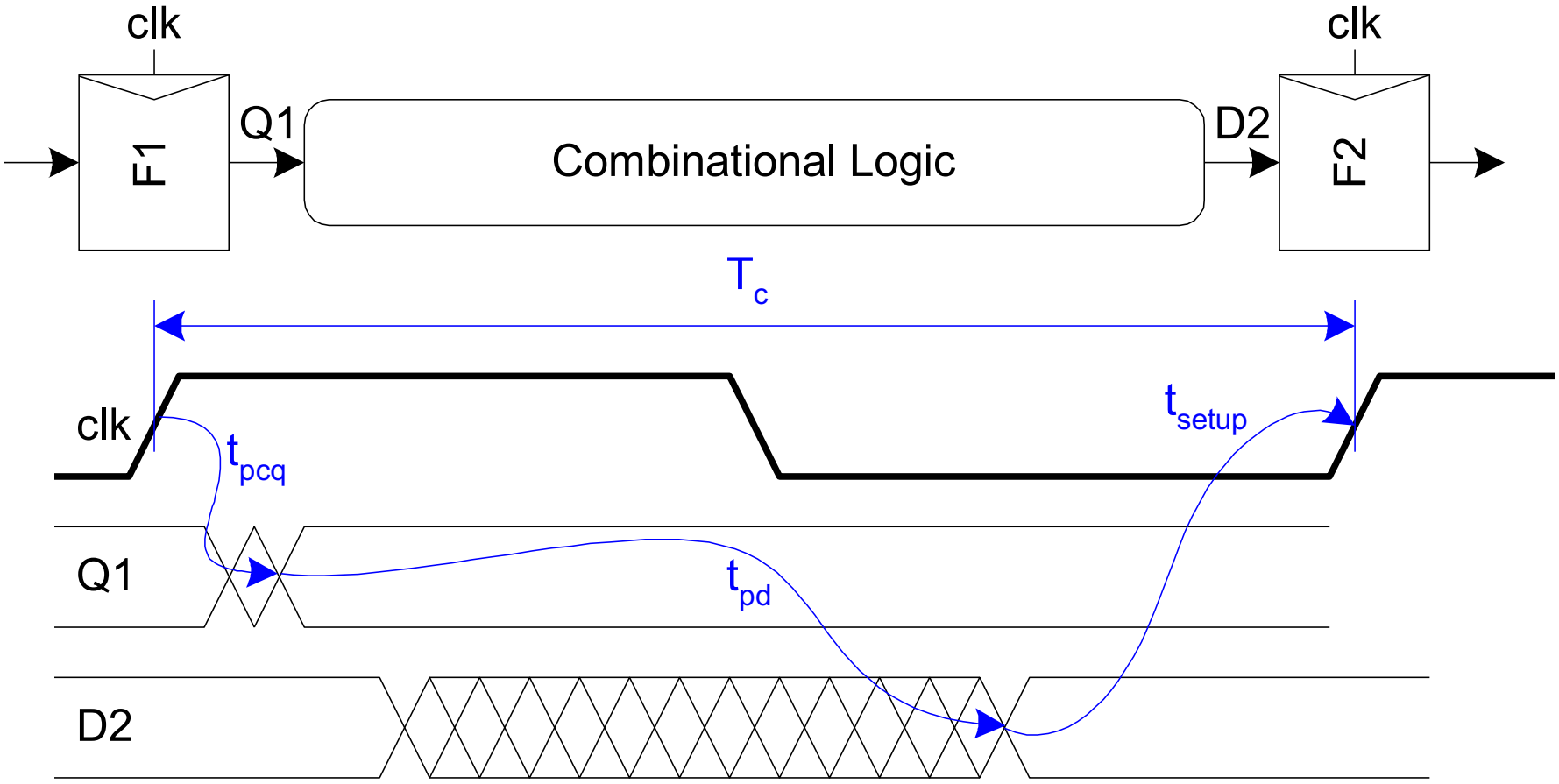
t_{pcq} =

$t_{hold} \geq$

Max-Delay: Flip-Flops

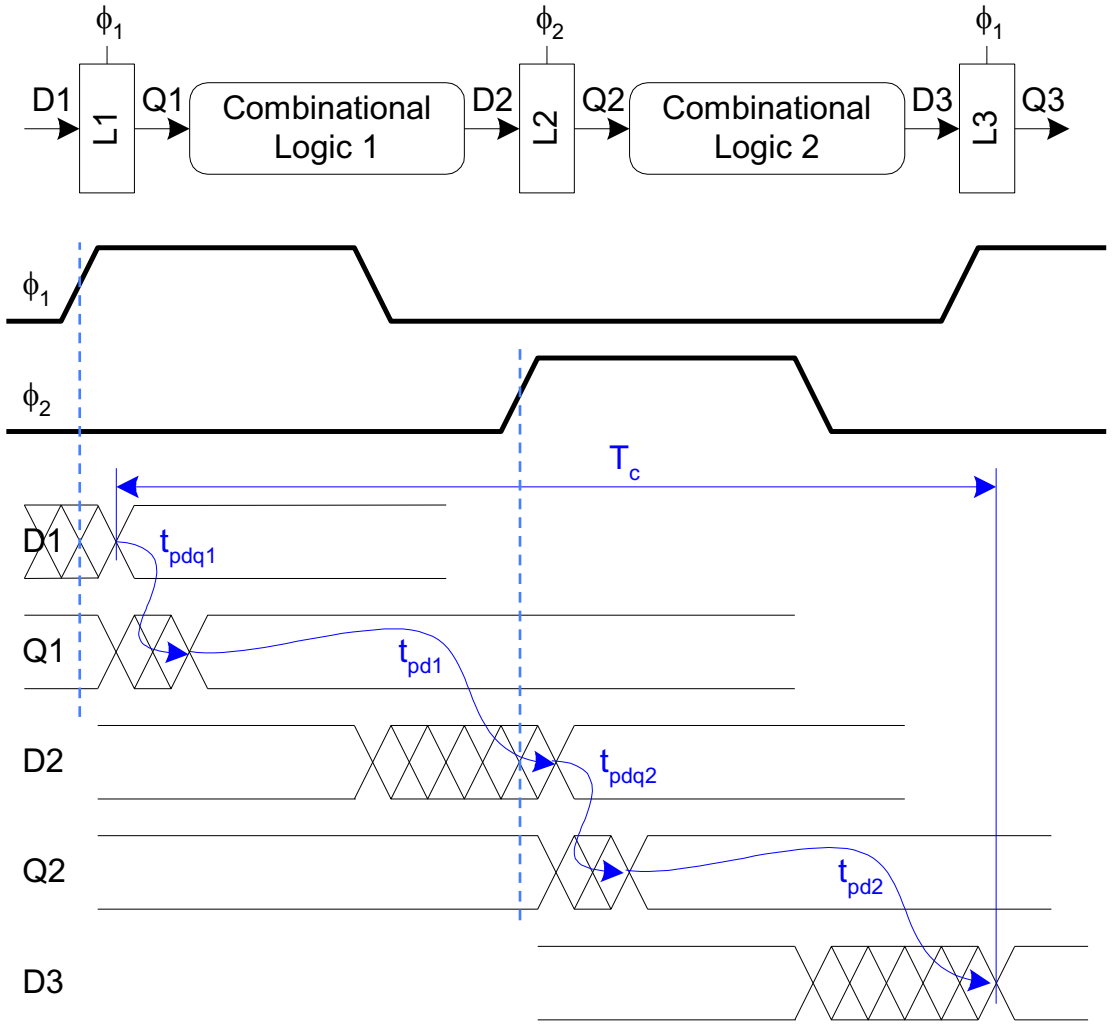
$$T_c \geq t_{pd+} (t_{setup} + t_{pcq})$$

sequencing overhead



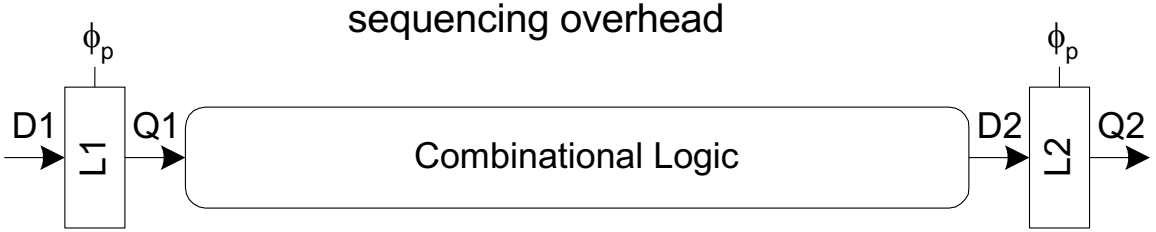
Max Delay: 2-Phase Latches

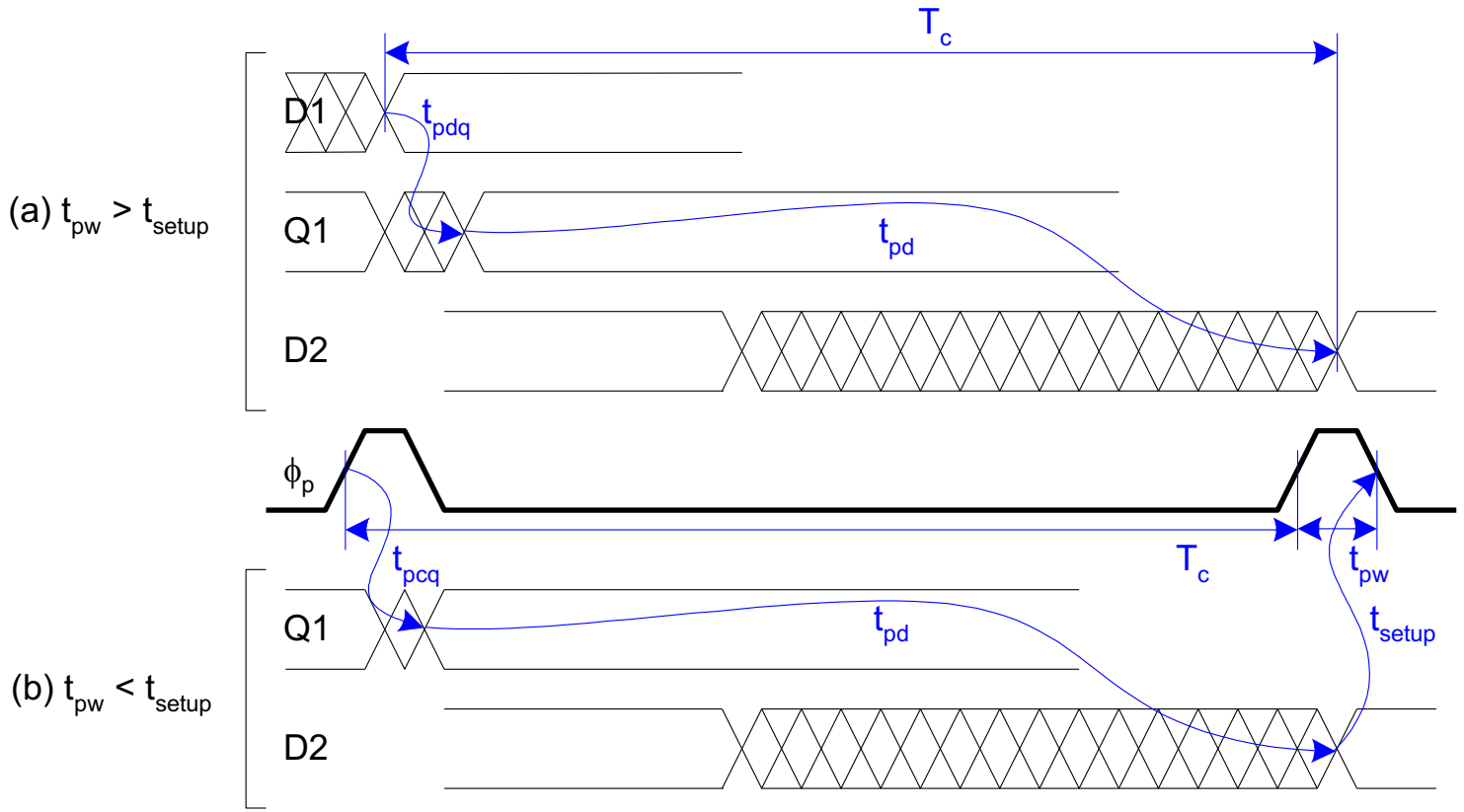
$$T_c \geq t_{pd1} + t_{pd2} + \underbrace{t_{pdq1} + t_{pdq2}}_{\text{sequencing overhead}}$$



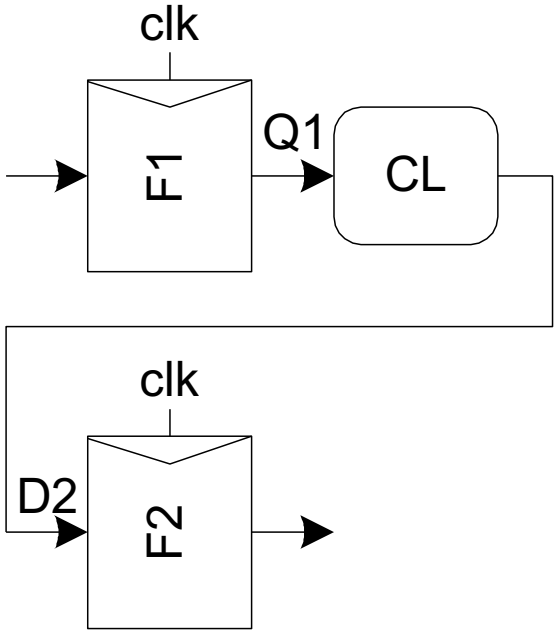
Max Delay: Pulsed Latches

$$t_{pd} + \underbrace{\max(t_{pdq},\ t_{pcq} + t_{setup} - t_{pw})}_{\text{sequencing overhead}} \le T_c$$

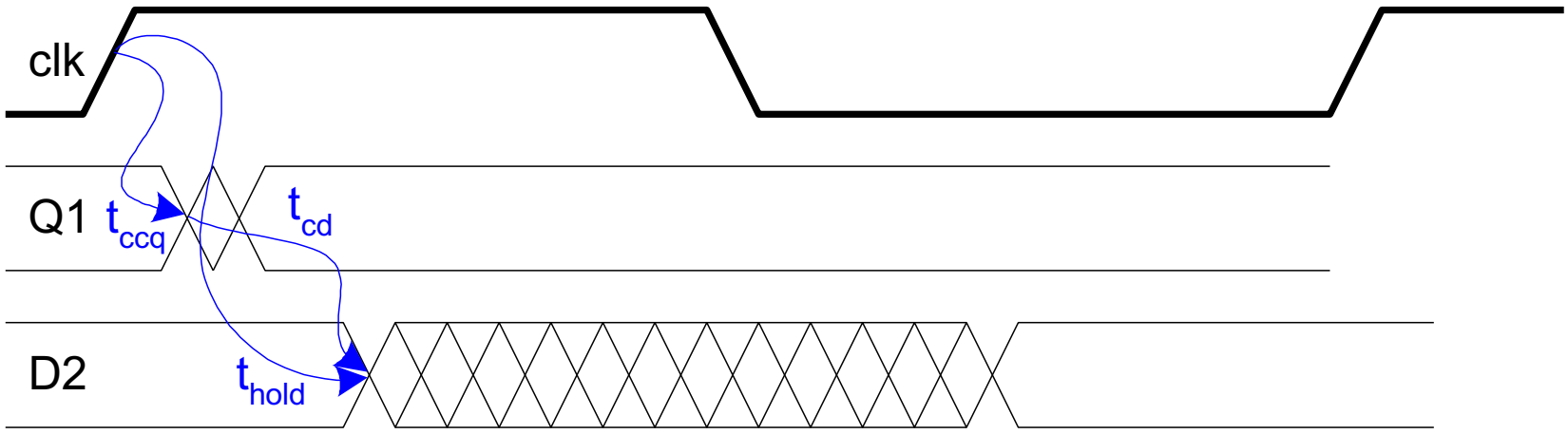




Min-Delay: Flip-Flops

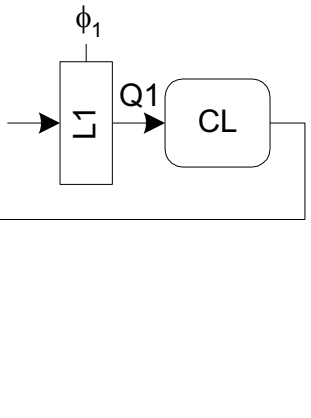


$$t_{cd} \geq t_{\text{hold}} - t_{ccq}$$



Min-Delay: 2-Phase Latches

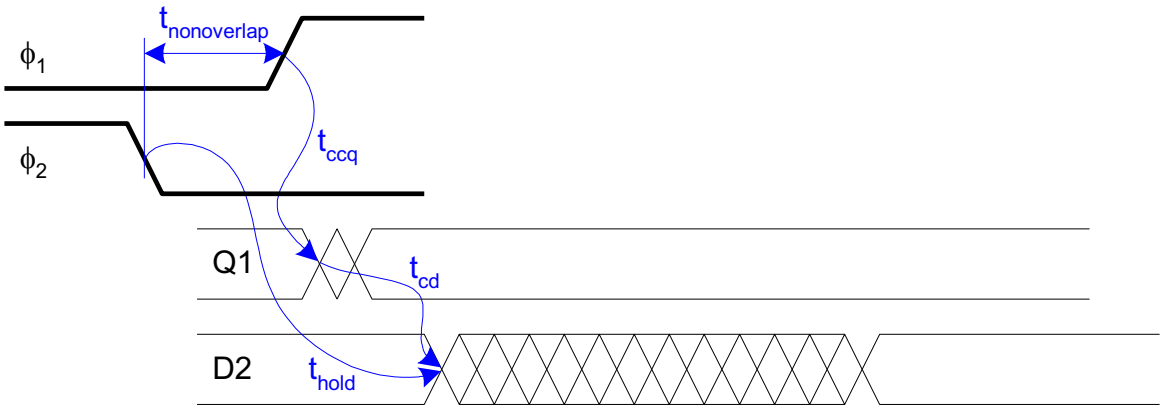
$$t_{cd1}, t_{cd2} \geq t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}}$$



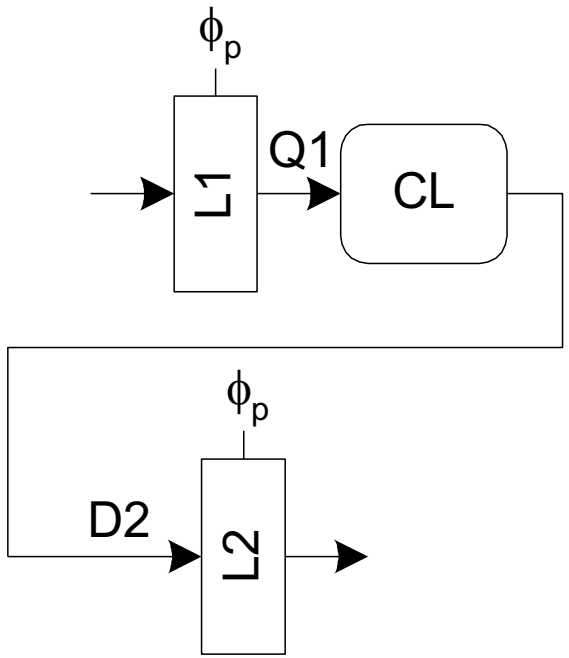
Hold time reduced by non-overlap

Paradox: hold applies twice each cycle, vs. only once for flops.

But a flop is made of two latches!

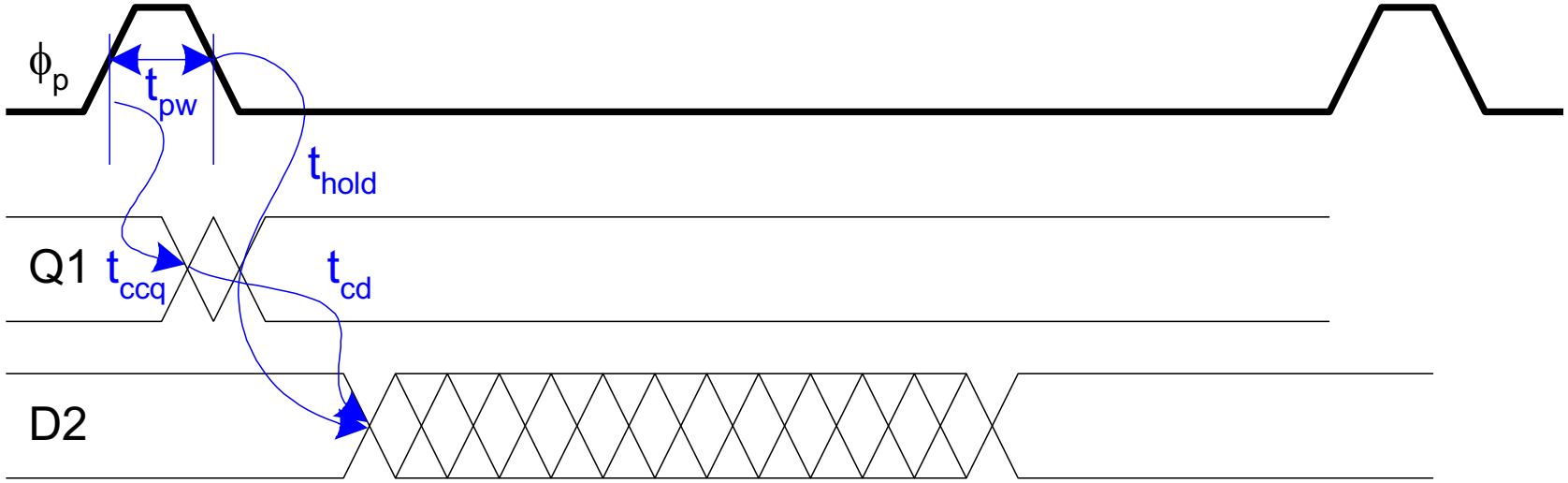


Min-Delay: Pulsed Latches



$$t_{cd} \geq t_{\text{hold}} - t_{ccq} + t_{pw}$$

Hold time increased by pulse width

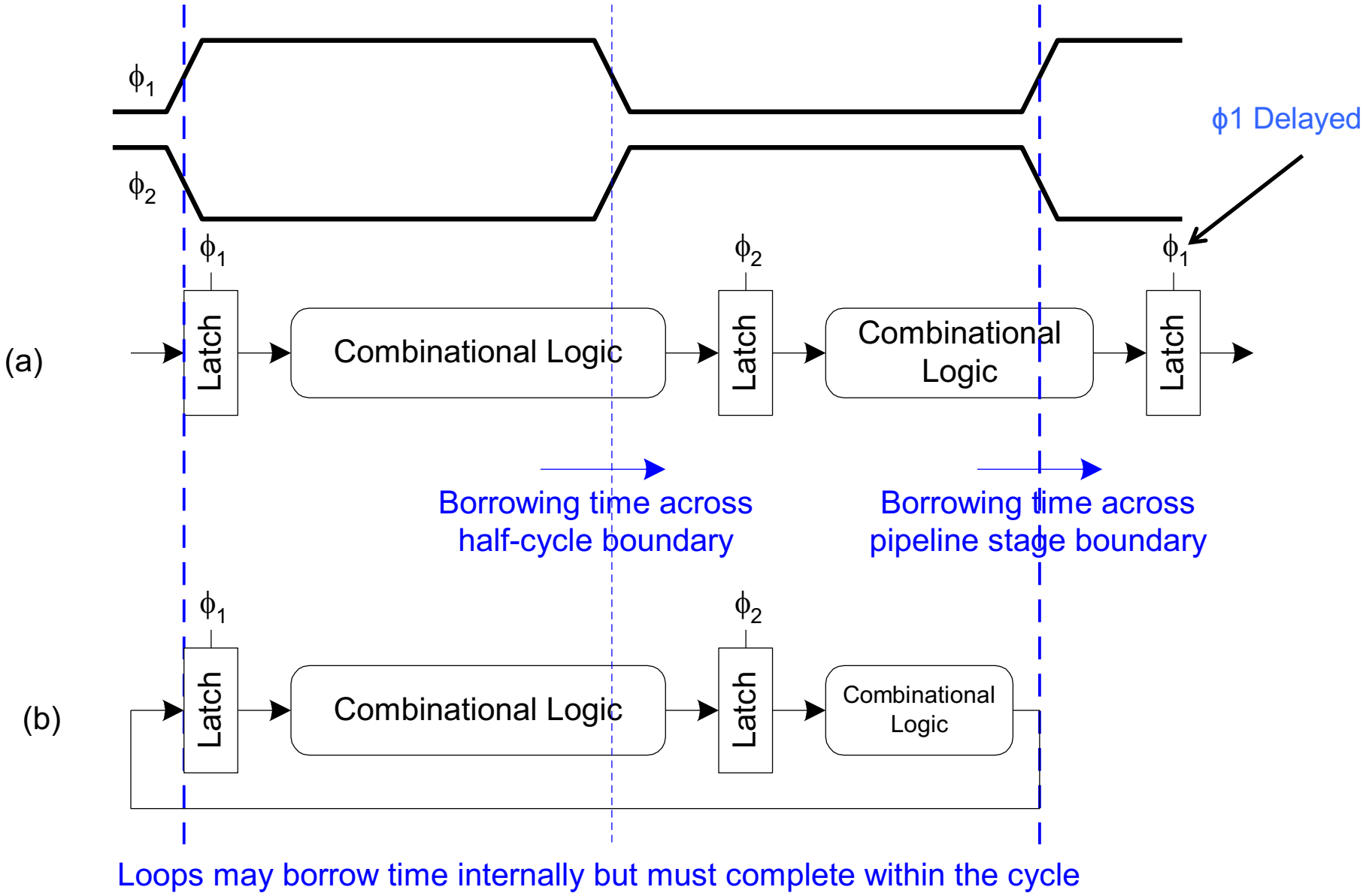


Time Borrowing

- **In a flop-based system:**
  - Data launches on one rising edge
  - Must setup before next rising edge
  - If it arrives late, system fails
  - If it arrives early, time is wasted
  - Flops have hard edges

- **In a latch-based system**
  - Data can pass through latch while transparent
  - Long cycle of logic can borrow time into next
  - As long as each loop completes in one cycle

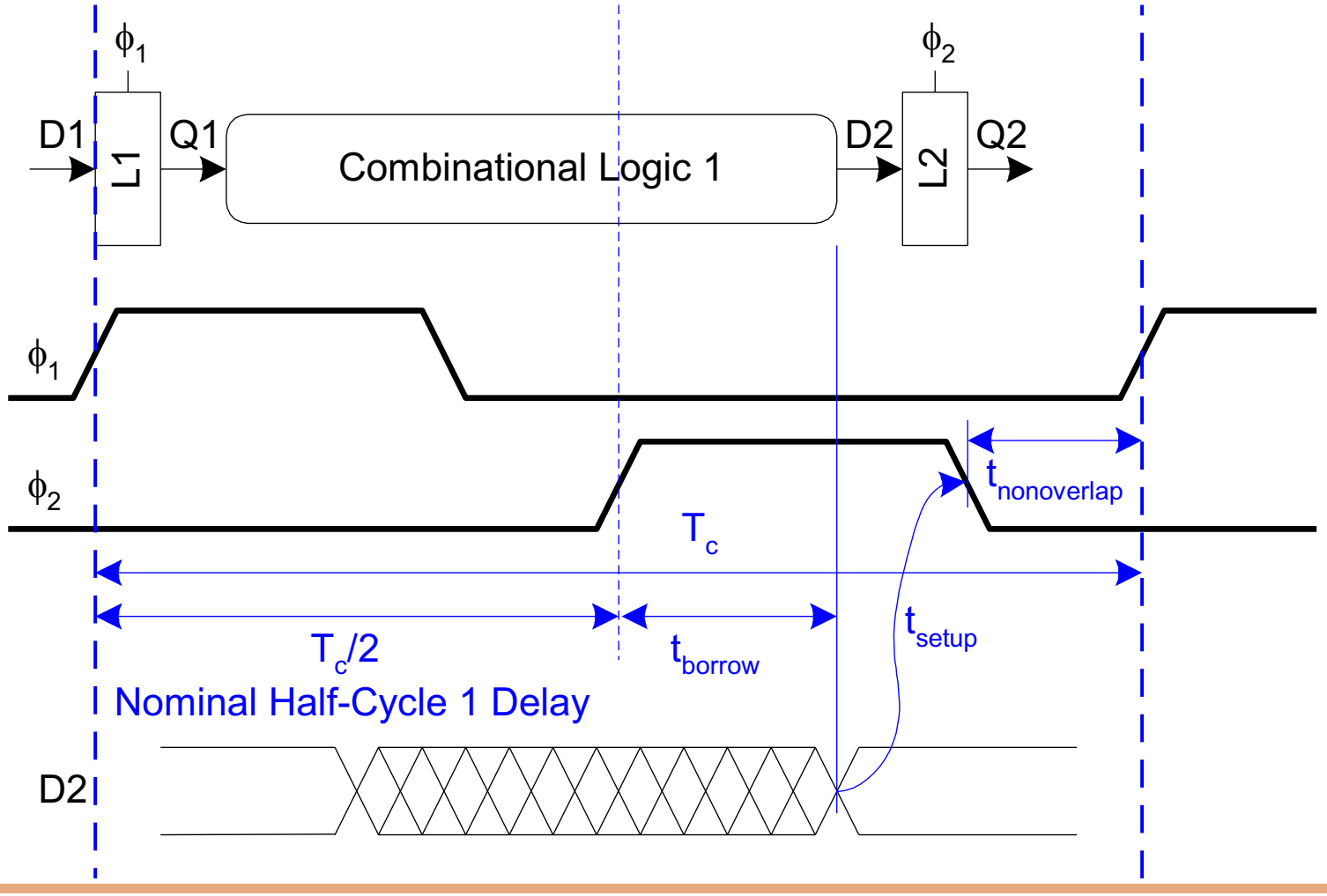
Time Borrowing Example



How Much Borrowing?

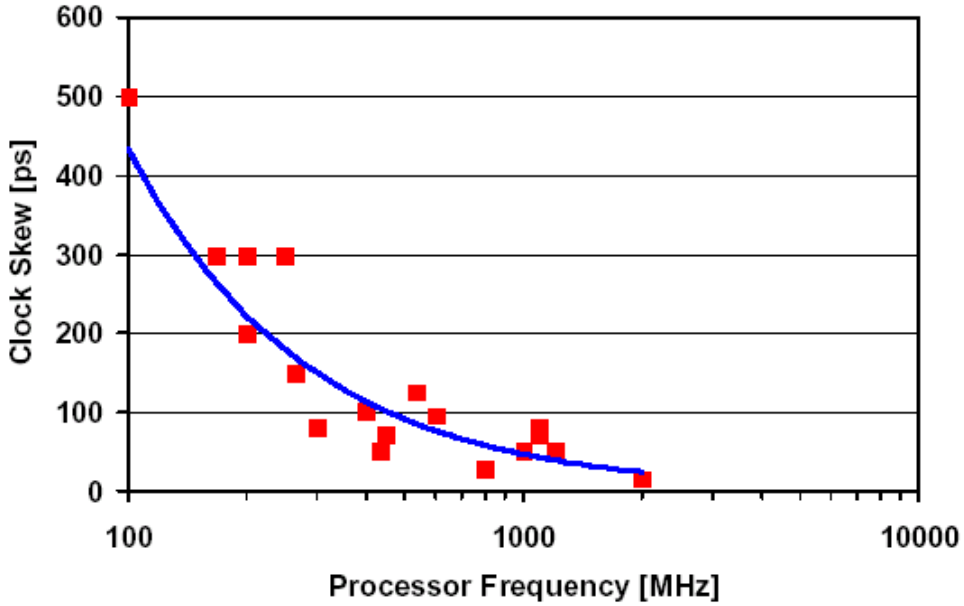
2-Phase Latches

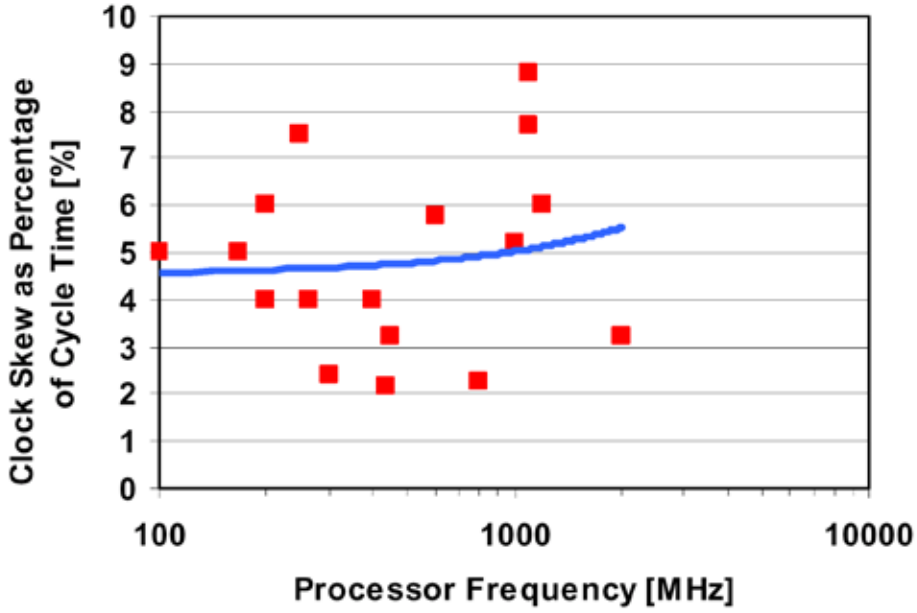
$$t_{\text{borrow}} \le \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}}\right)$$



Clock Skew

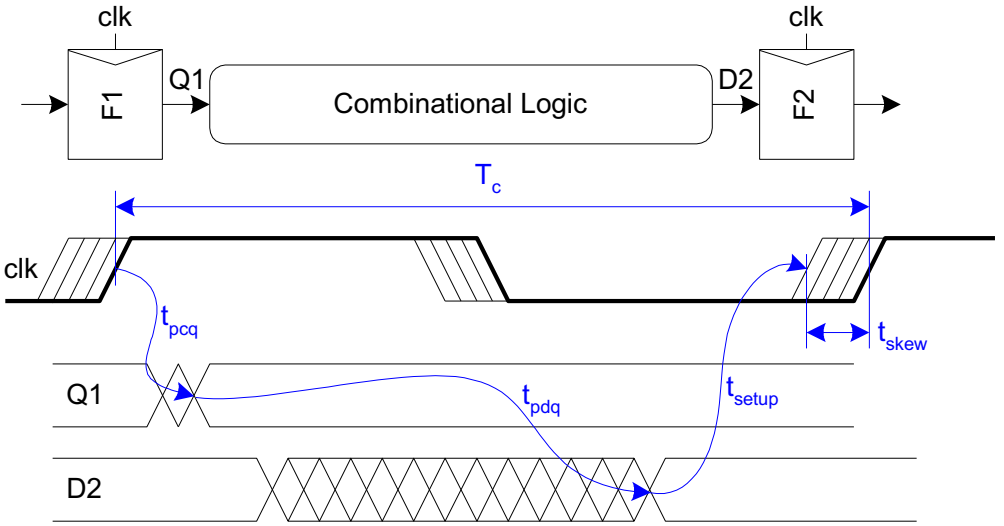
- We have assumed zero clock skew
- Clocks really have uncertainty in arrival time
  - Decreases maximum propagation delay
  - Increases minimum contamination delay
  - Decreases time borrowing



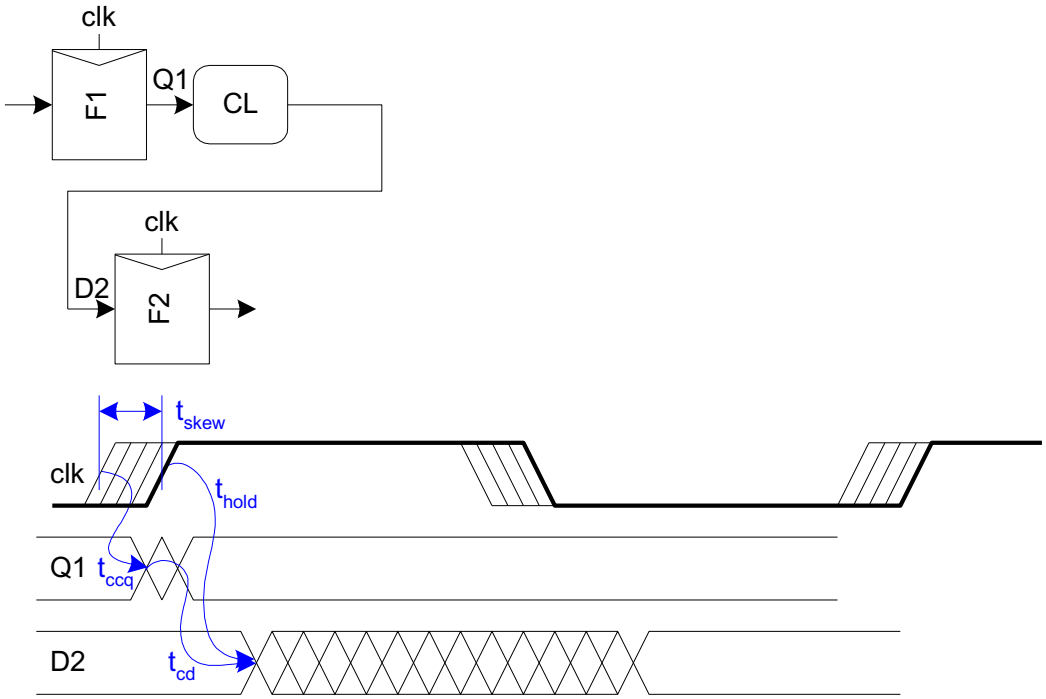


Clock Skew: Flip-Flops

$$T_c \geq t_{pd} + \underbrace{(t_{pcq} + t_{setup} + t_{skew})}_{\text{sequencing overhead}}$$



$$t_{cd} \geq t_{hold} - t_{ccq} + t_{skew}$$



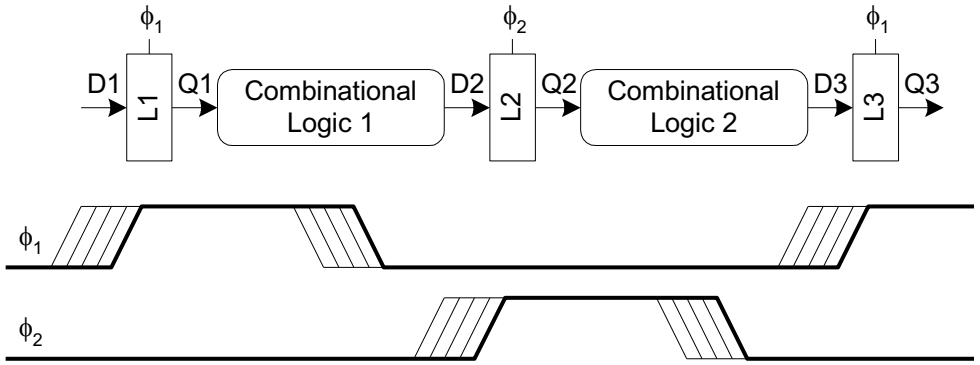
Clock Skew: Latches

2-Phase Latches

$$t_{pd} \leq T_c - \underbrace{(2t_{pdq})}_{\text{sequencing overhead}}$$

$$t_{cd1}, t_{cd2} \geq t_{hold} - t_{ccq} - t_{nonoverlap} + t_{skew}$$

$$t_{borrow} \leq T_c/2 - (t_{setup} + t_{nonoverlap} + t_{skew})$$



Pulsed Latches

$$t_{pd} \leq T_c - \underbrace{\max(t_{pdq}, t_{pcq} + t_{setup} - t_{pw} + t_{skew})}_{\text{sequencing overhead}}$$

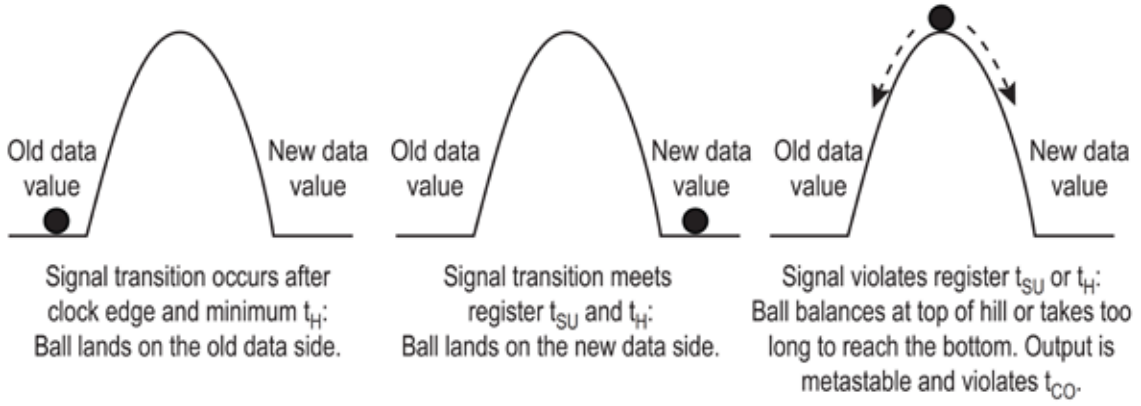
$$t_{cd} \geq t_{hold} + t_{pw} - t_{ccq} + t_{skew}$$

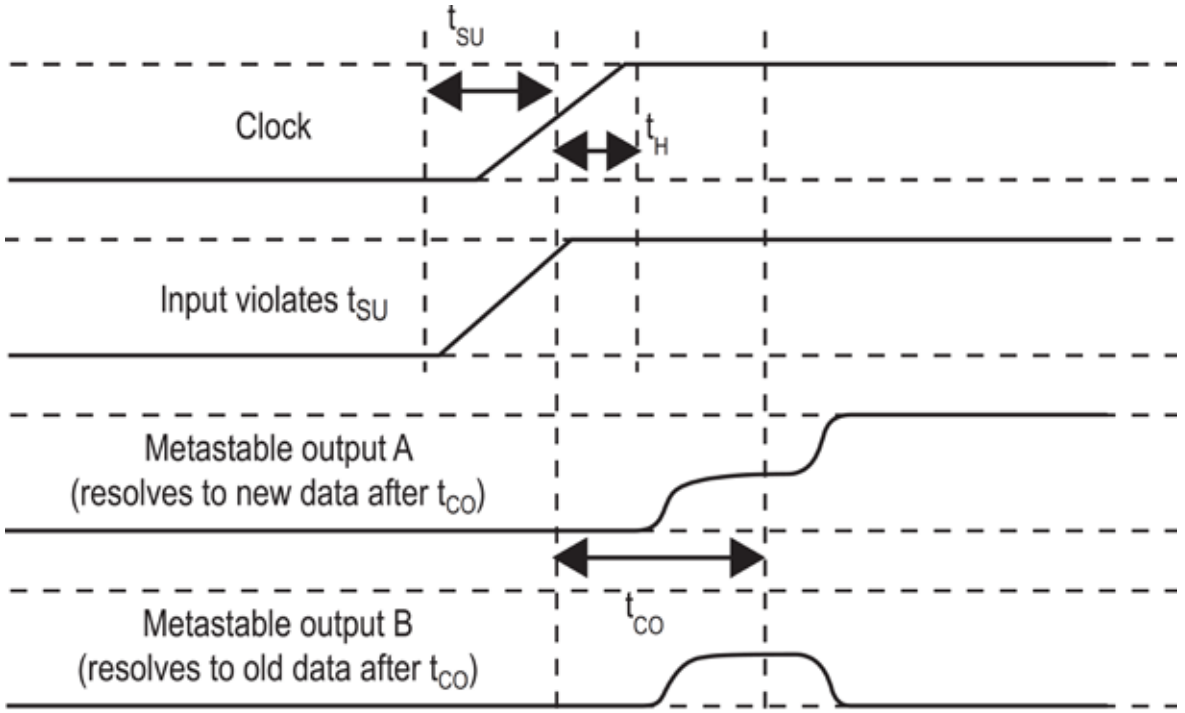
$$t_{borrow} \leq T_{pw} - (t_{setup} + t_{skew})$$

Clocking realities

- If setup times are violated, reduce clock speed
- Useful clock skew can be your friend
- Jitter is NEVER your friend
- Pulse latches do not scale well from generation to generation
  - Use them if you want lot's of debugging experience ☺
- Metastability is very real (and deadly)
- Lastly, if hold times are violated, chip fails at any speed and PVT
  - You have a "brick" for a chip
  - You may be out of business if you are a startup

Metastability





Courtesy Altera

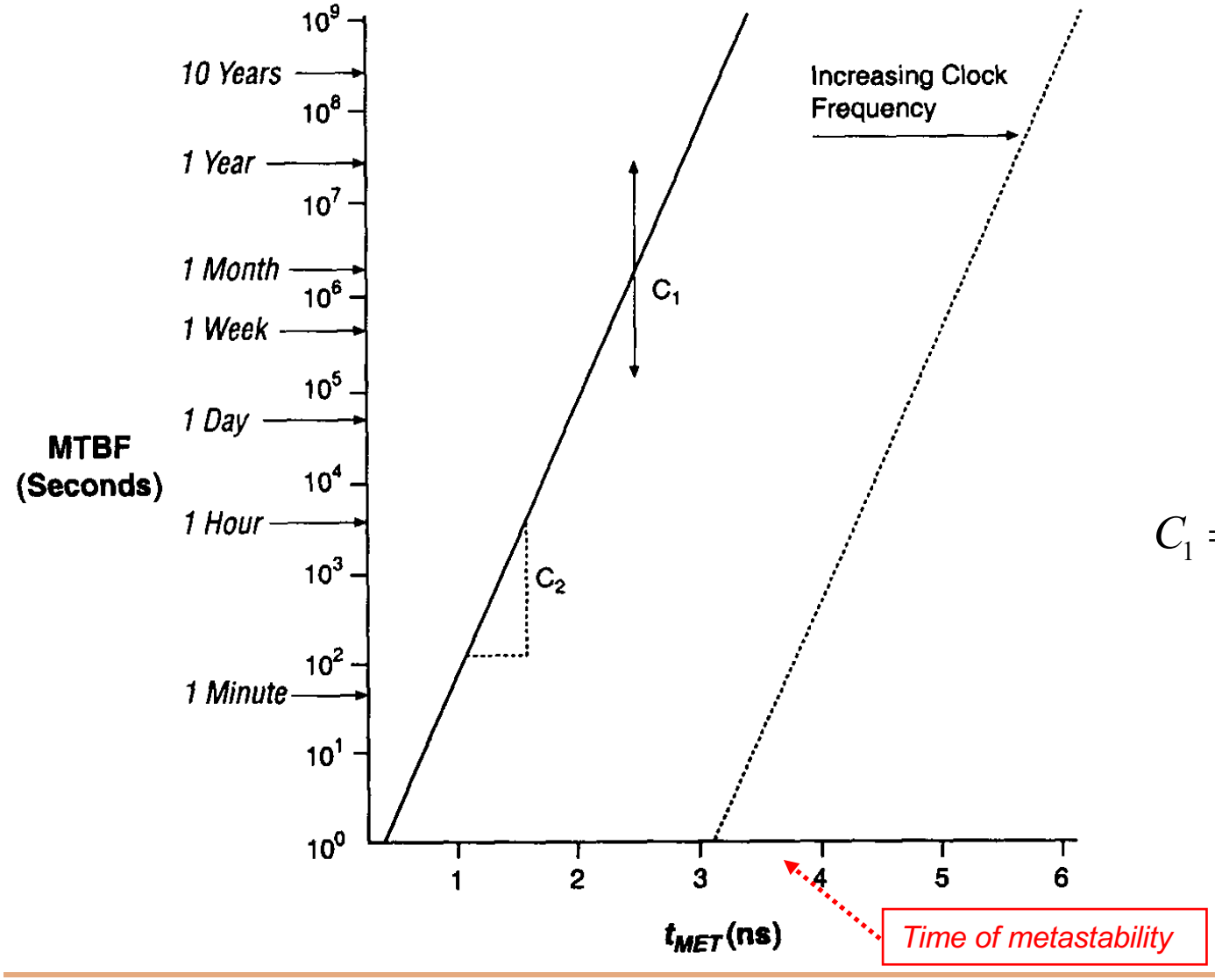
Metastability - MTBF

- The *C1* and *C2* constants depend on the device process and operating conditions. Determined empirically.
- f_{CLK} is the clock frequency of the clock domain receiving the asynchronous signal
- f_{DATA} is the toggling frequency of the asynchronous input data signal. Faster clock frequencies and faster-toggling data reduce (or worsen) the MTBF.
- The t_{MET} parameter is the available metastability settling time, or the timing slack available beyond the register's t_{CO}, for a potentially metastable signal to resolve to a known value.

$$MTBF = \frac{e^{t_{MET}/C_2}}{C_1 \cdot f_{CLK} \cdot f_{DATA}}$$

Courtesy Altera

MTBF vs. T_{MET}



$$C_2 = \frac{\triangle \ln(MTBF)}{\triangle t_{MET}}$$

$$C_1 = \frac{e^{(C_2 \times t_{MET})}}{MTBF \times f_{CLOCK} \times f_{DATA}}$$

Courtesy Altera

MTBF: Alternate definition

To avoid synchronizer failure wait long enough before using a synchronizer's output. Where "long enough", is the mean time between synchronizer failures and is several orders of magnitude longer than the designer's expected length of employment!

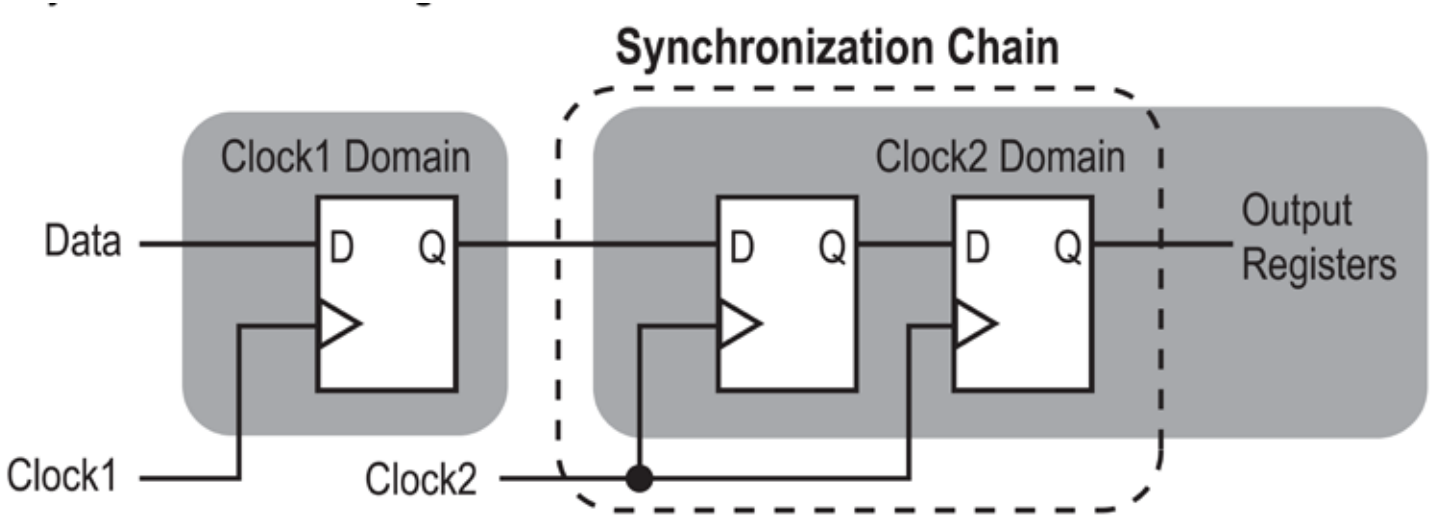
John Wakerly

Preventing Metastability

- **The t_{MET} for a synchronization chain is the sum of the output timing slacks for each register in the chain.**

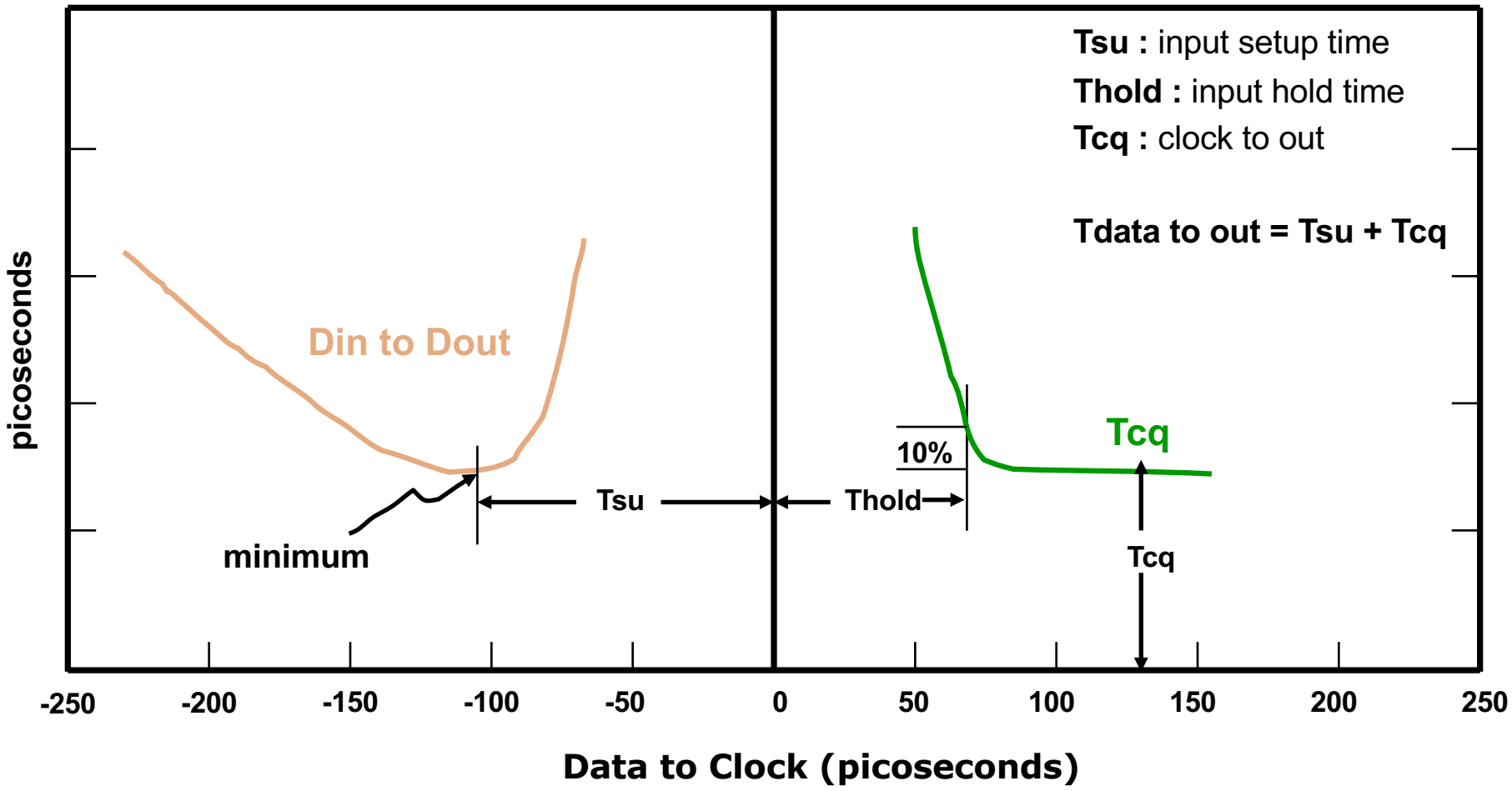
$$MTBF = \frac{e^{t_{MET}/C_2}}{C_1 \cdot f_{CLK} \cdot f_{DATA}}$$

$$failure\_rate_{design} = \frac{1}{MTBF_{design}} = \sum_{i=1}^{number\ of\ chains} \frac{1}{MTBF_i}$$

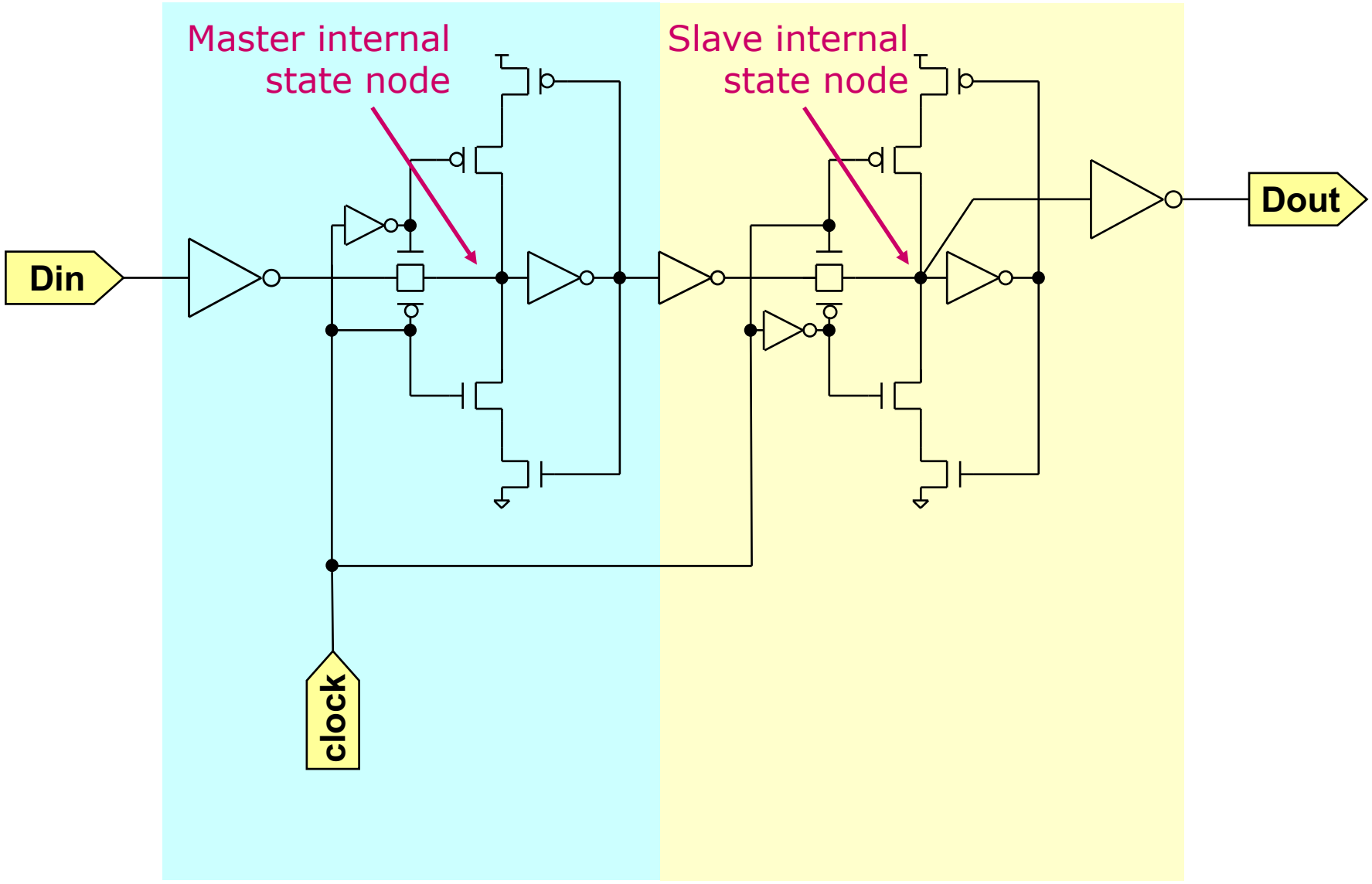


Courtesy Altera

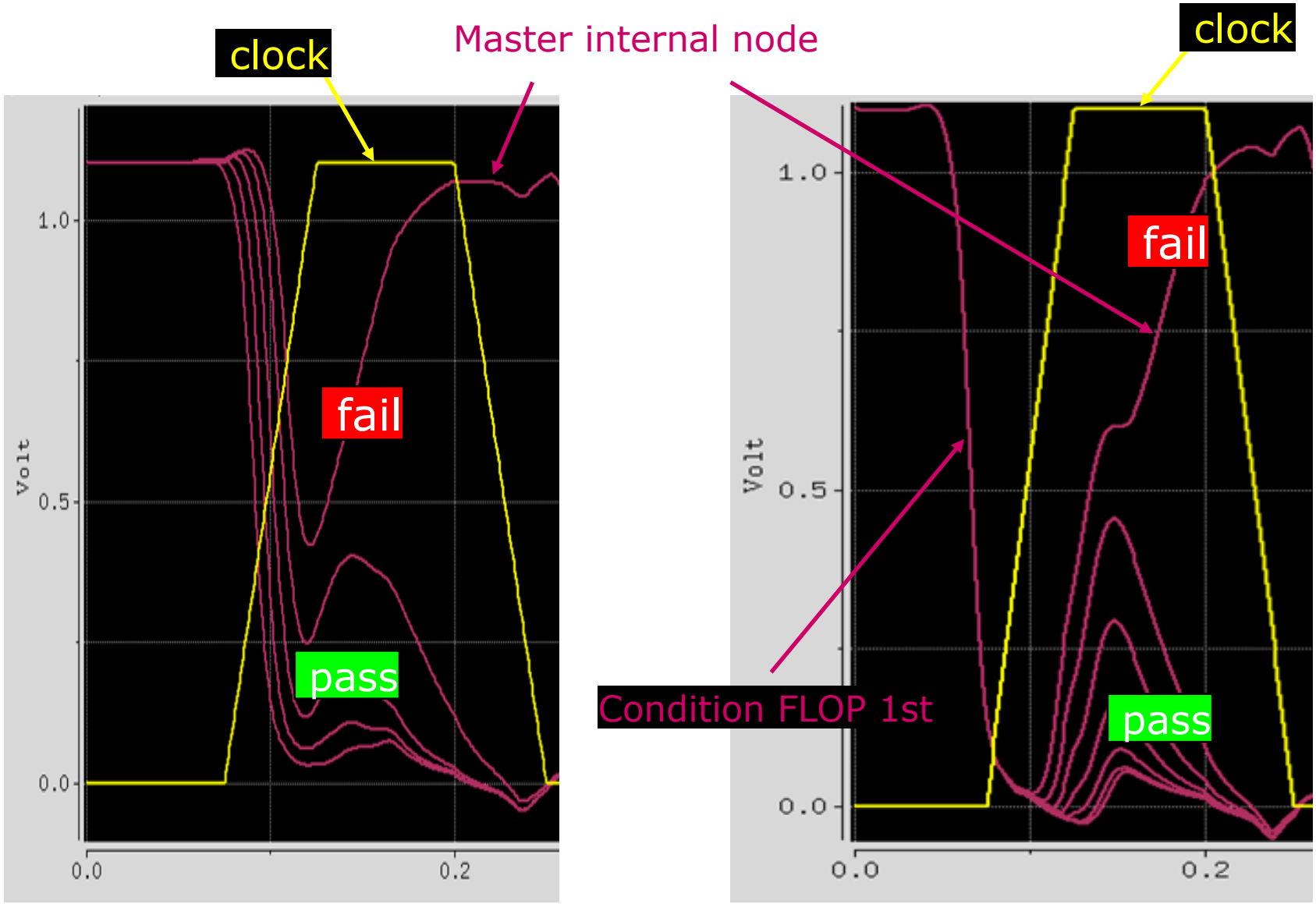
Simulating Metastability



Simulating Metastability (cont.)



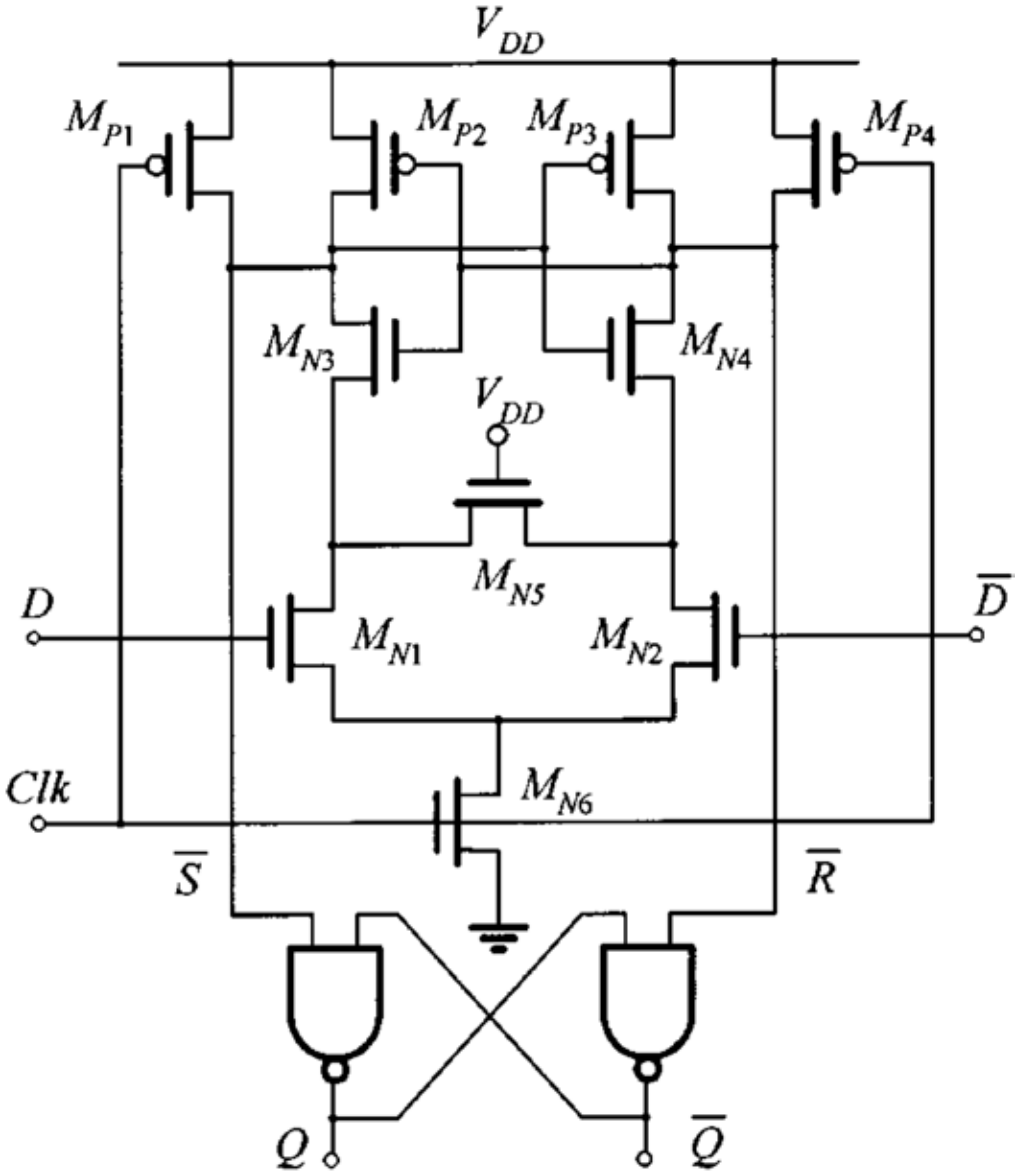
Functional Pass/Failure vs. Tsu and Th



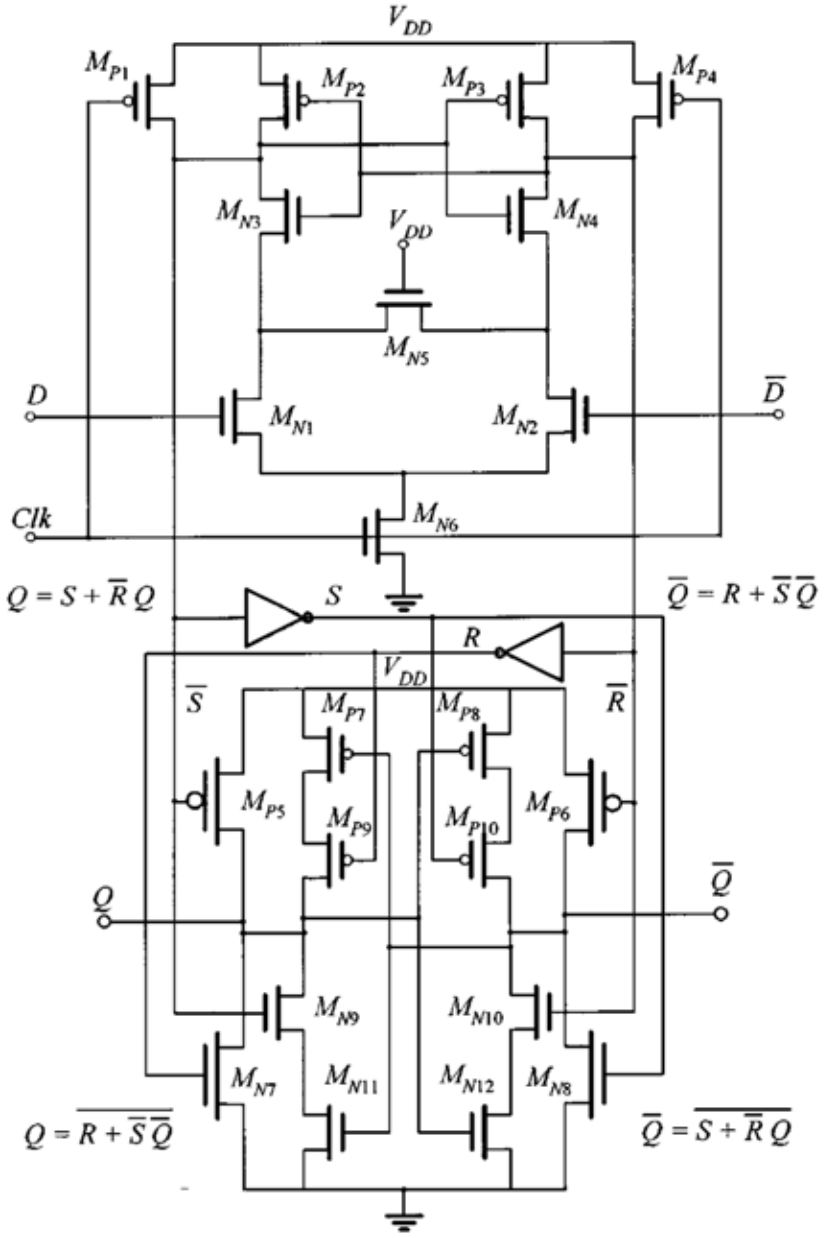
Input setup time

Input hold time

High Speed Flip-Flop for Synchronization

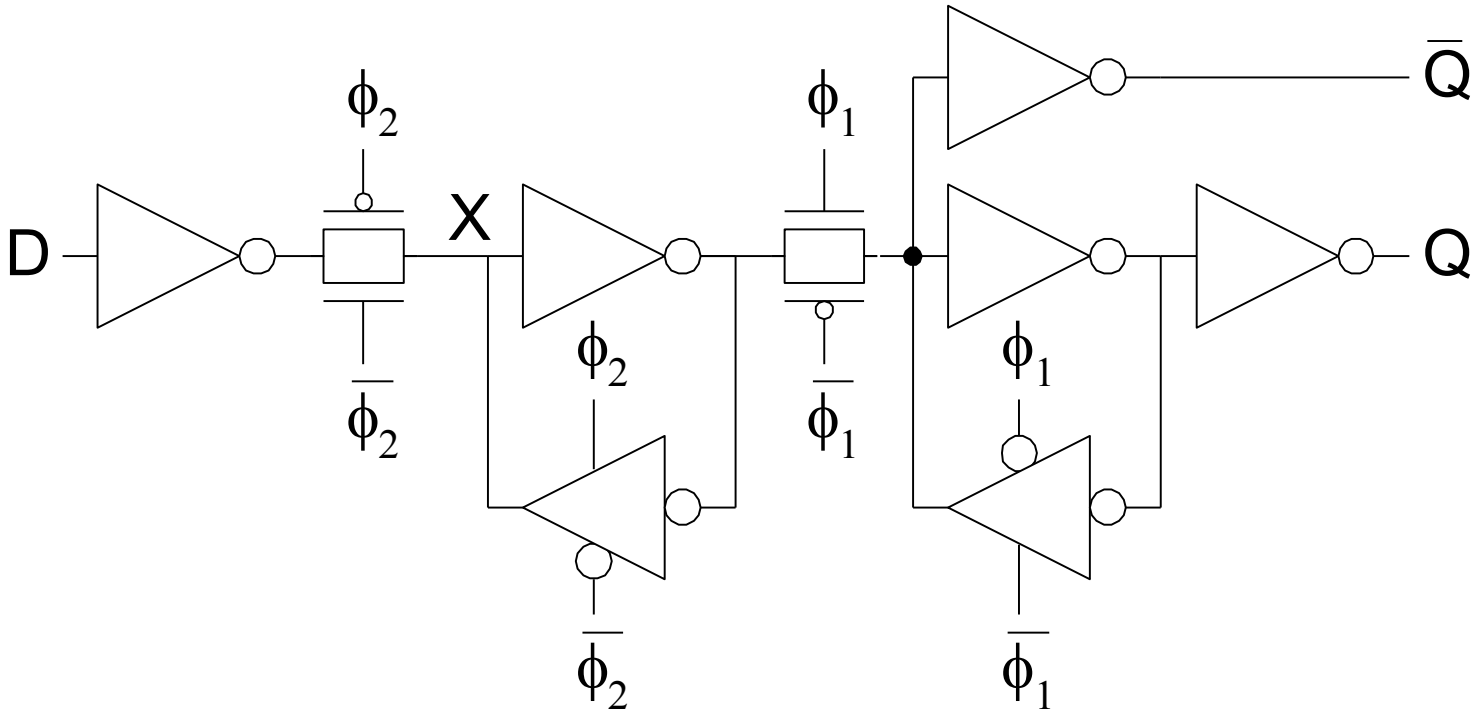


High Speed Flip-Flop for Synchronization (cont.)

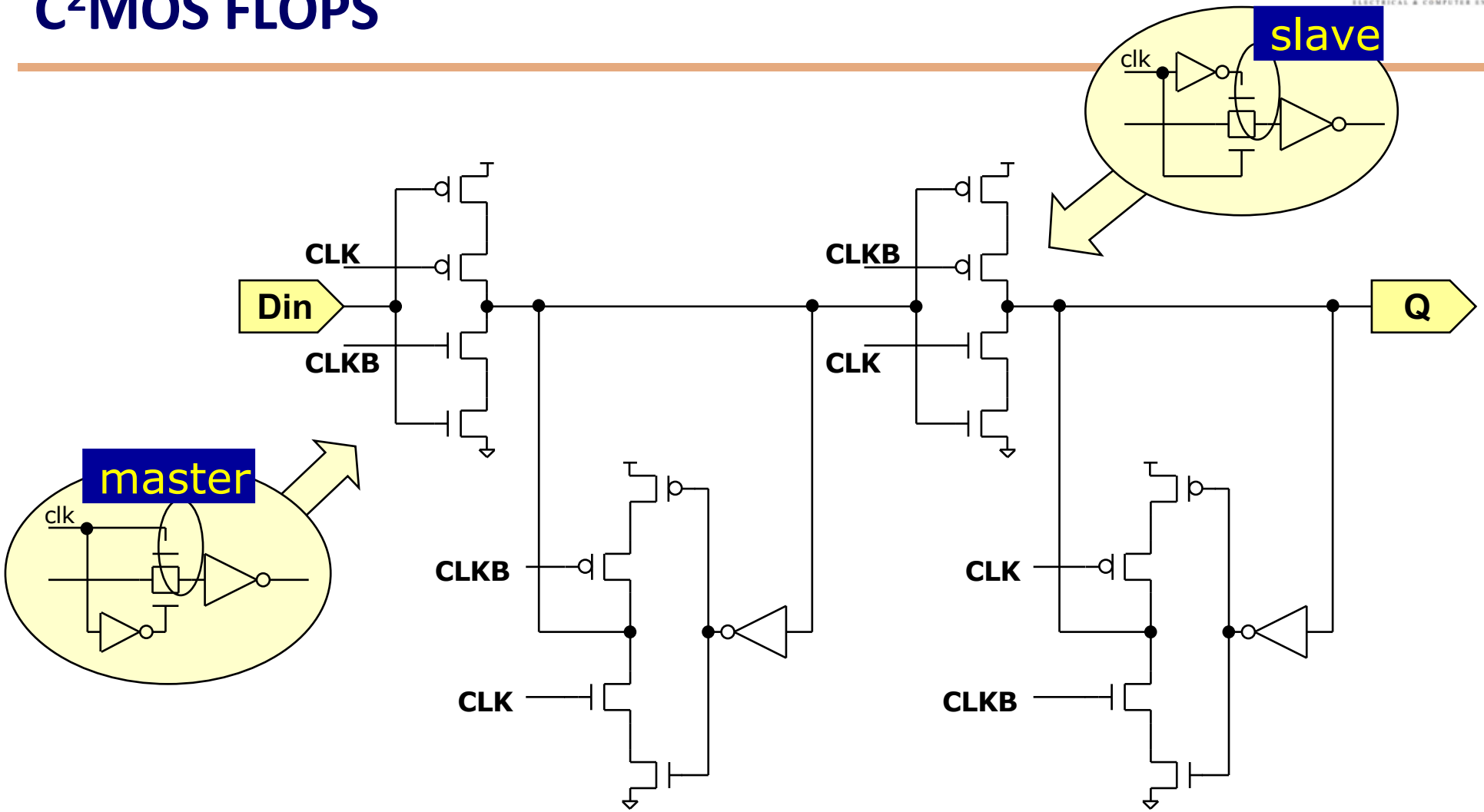


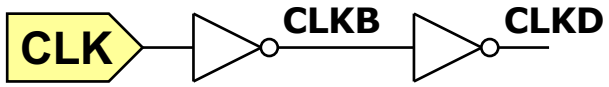
Safe Flip-Flop

- **Use flip-flop with non-overlapping clocks**
  - Very slow – non-overlap adds to setup time
  - But no hold times
- **In industry, use a better timing analyzer**
  - Add buffers to slow signals if hold time is at risk



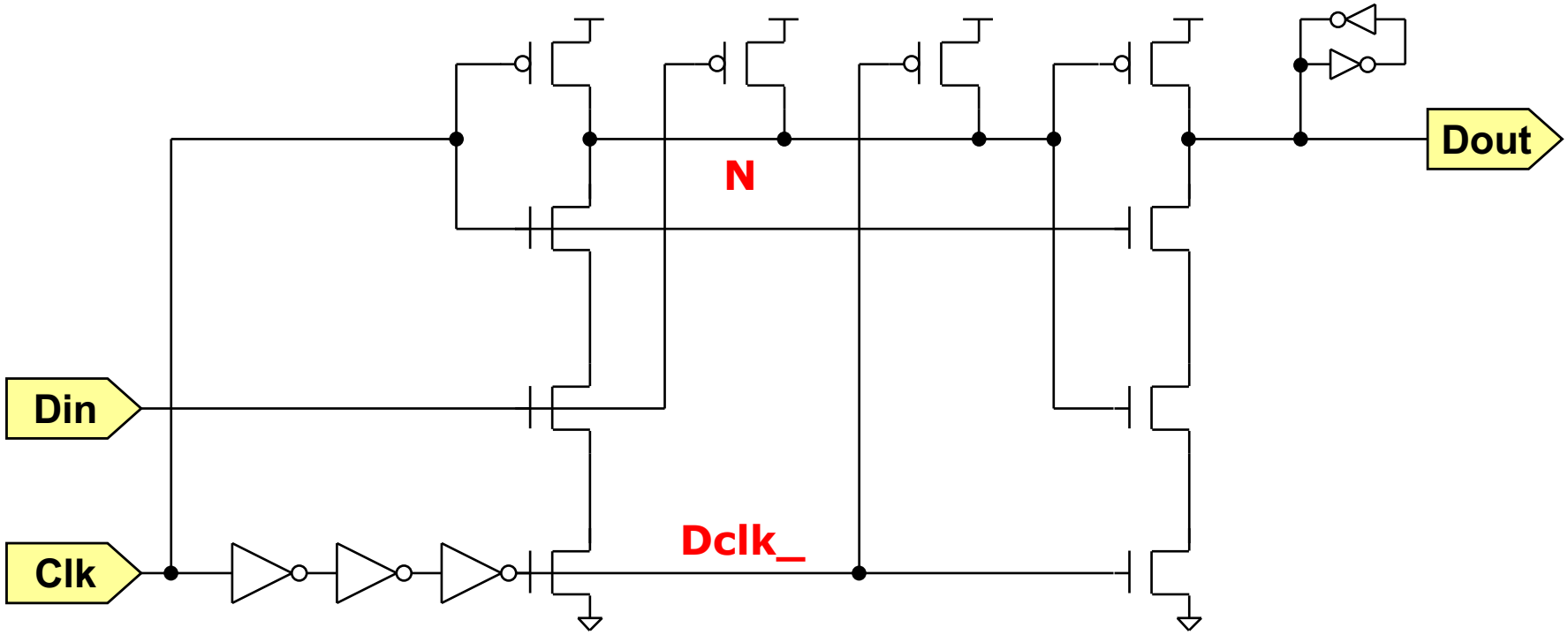
C²MOS FLOPS





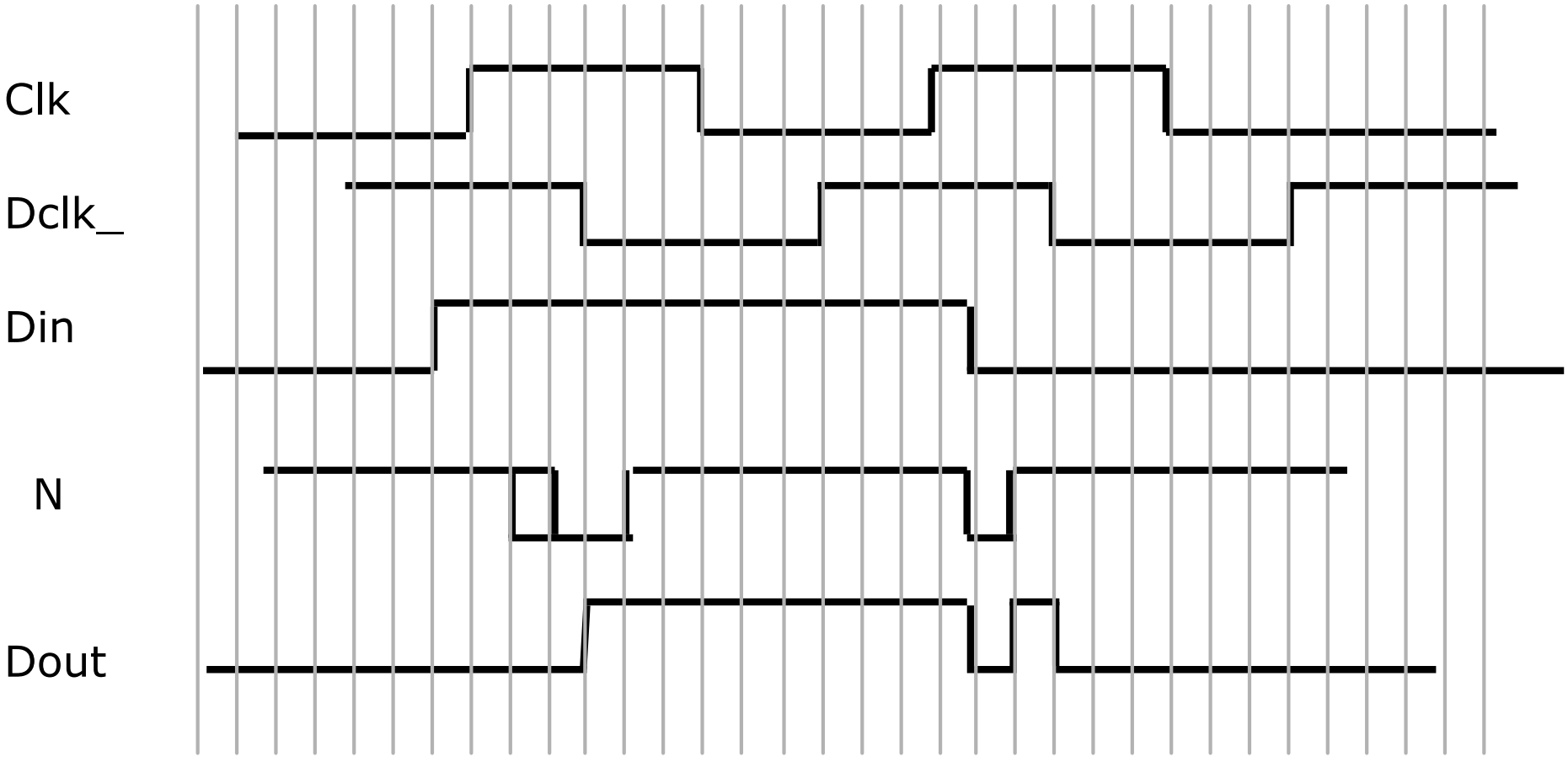
Clock slope becomes critical
Low power feedback
Poor driving capability

Hybrid Latch Flip-Flop (HLFF)

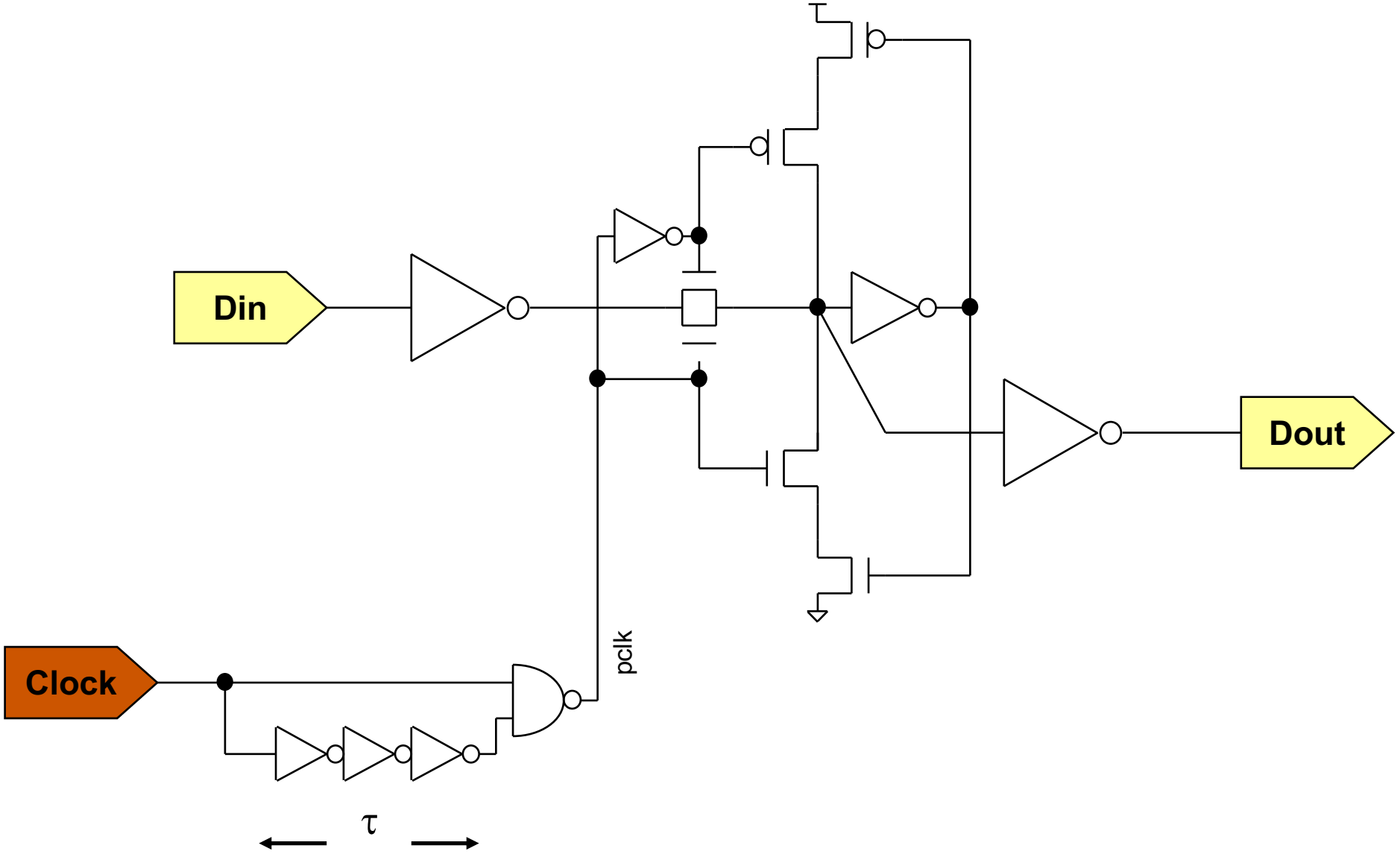


(AMD K-6, Partovi, ISSCC 1996)

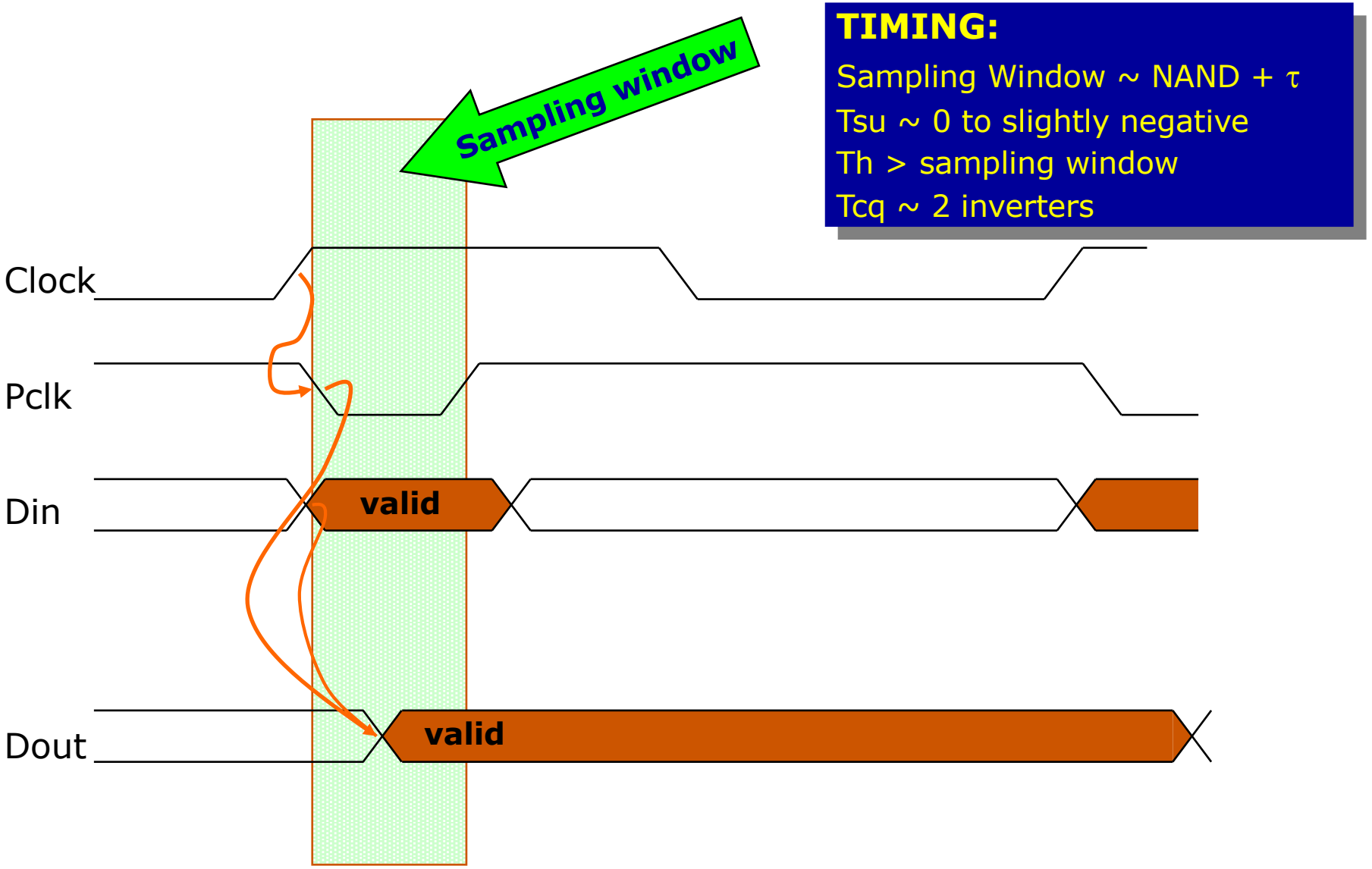
Hybrid Latch Flip-Flop Timing



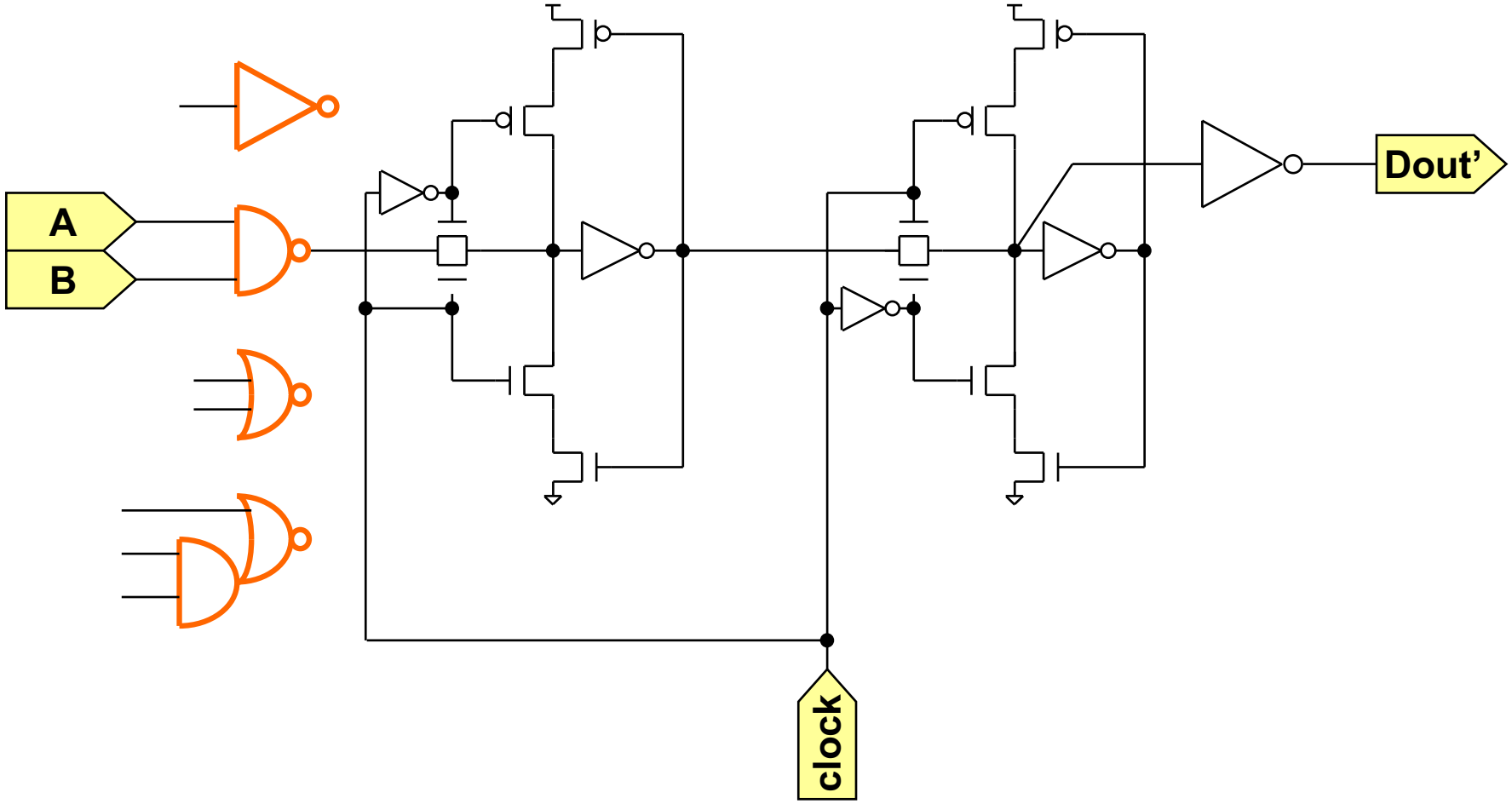
Pulse Latch



Pulse Latch Waveforms



Merged Function inverting FLOP



Flip-Flop Summary

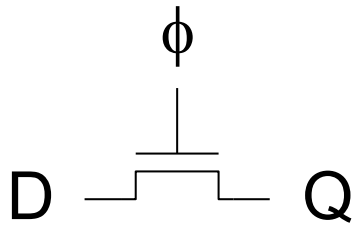
- **Flip-Flops:**
  - Very easy to use, supported by all tools
- **2-Phase Transparent Latches:**
  - Lots of skew tolerance and time borrowing
  - Supported by most tools. Can cause timing loops.
- **Pulsed Latches:**
  - Fast, some skew tol. & borrow, hold time risk

	Sequencing overhead ($T_c - t_{pd}$)	Minimum logic delay t_{cd}	Time borrowing t_{borrow}
Flip-Flops	$t_{pcq} + t_{setup} + t_{skew}$	$t_{hold} - t_{ccq} + t_{skew}$	0
Two-Phase Transparent Latches	$2t_{pdq}$	$t_{hold} - t_{ccq} - t_{nonoverlap} + t_{skew}$ in each half-cycle	$\frac{T_c}{2} - \left(t_{setup} + t_{nonoverlap} + t_{skew}\right)$
Pulsed Latches	$\max\left(t_{pdq}, t_{pcq} + t_{setup} - t_{pw} + t_{skew}\right)$	$t_{hold} - t_{ccq} + t_{pw} + t_{skew}$	$t_{pw} - \left(t_{setup} + t_{skew}\right)$

Backup

Dynamic Latch Design

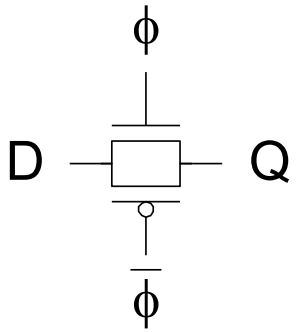
- **Pass Transistor Latch**
- **Pros**
  - Tiny
  - Low clock load
- **Cons**
  - Vt drop
  - Non-restoring
  - Back driving
  - Output noise sensitivity
  - Dynamic
  - Diffusion input
  - Requires capacitance on output to store state.



Used in the 1970s

Dynamic Latch Design (cont.)

- **Transmission gate**
  - + No V_t drop
  - - Requires inverted clock



- **Inverting buffer**
  - + Restoring
  - + No back driving
  - + Fixes either
    - Output noise sensitivity
    - Or diffusion input
  - – Inverted output