Cybersecurity for IoT – Secure Hardware

Department of Electrical, Computer and Biomedical Engineering of University of Pavia

Master of Science Program in Computer Engineering

Instructor: Paris Kitsos

http://diceslab.cied.teiwest.gr

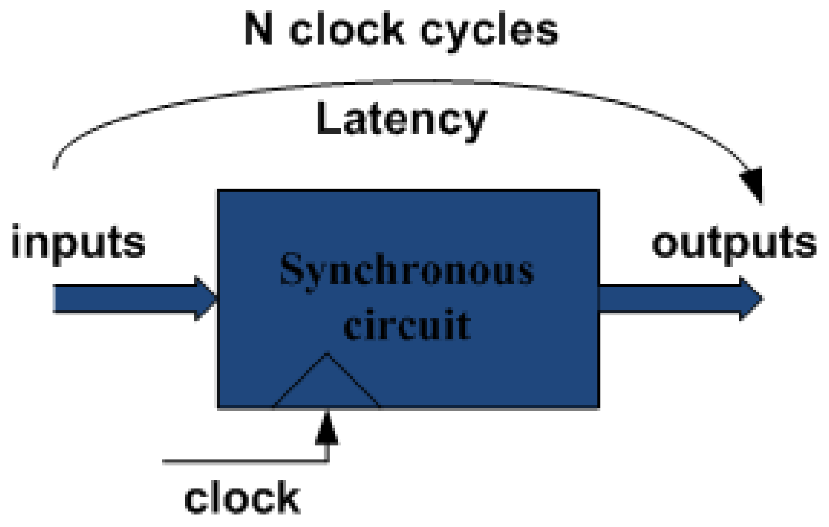
E-mail: pkitsos@teimes.gr

Pavia 2018

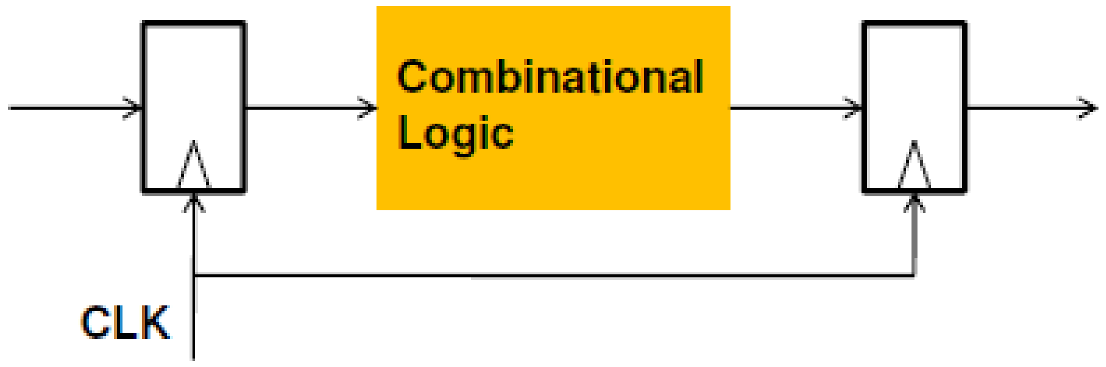
- **Part 1 – Pipelining and Retiming**

Delay of a Design

Delay = latency x clock period

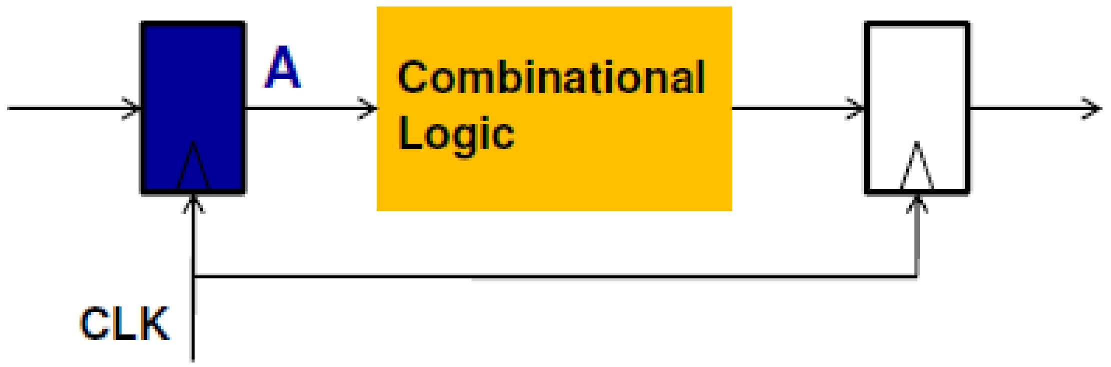


Minimum Clock Period…



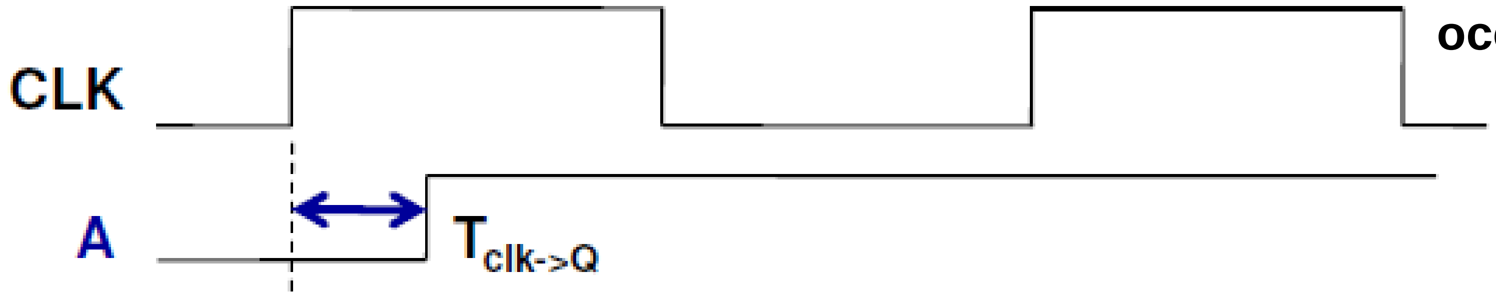
$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$

…Minimum Clock Period…

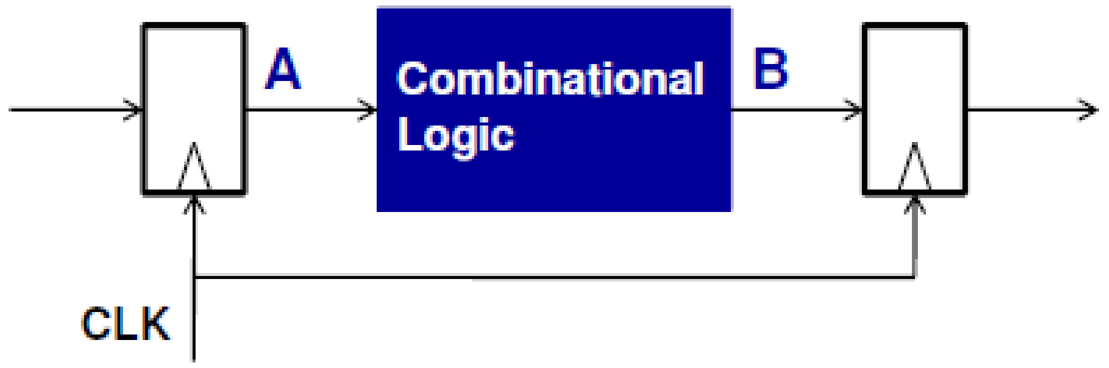


This is the time need for the output of a flip-flop to switch to a new value after a clock edge has occured

$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$

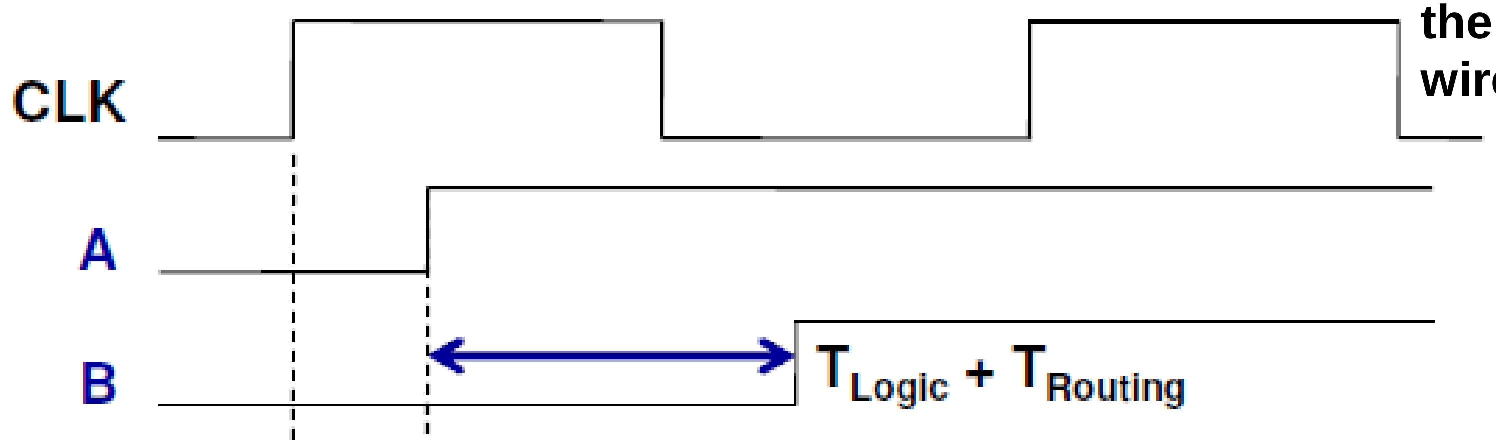


…Minimum Clock Period…

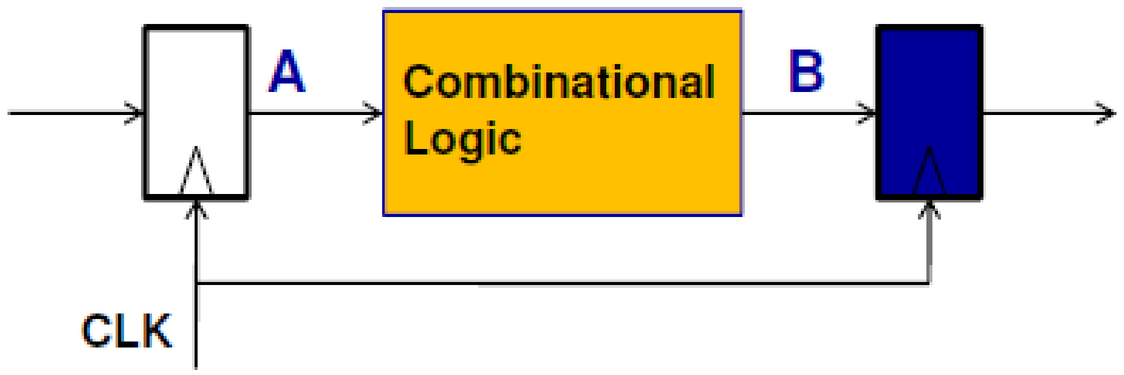


$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$

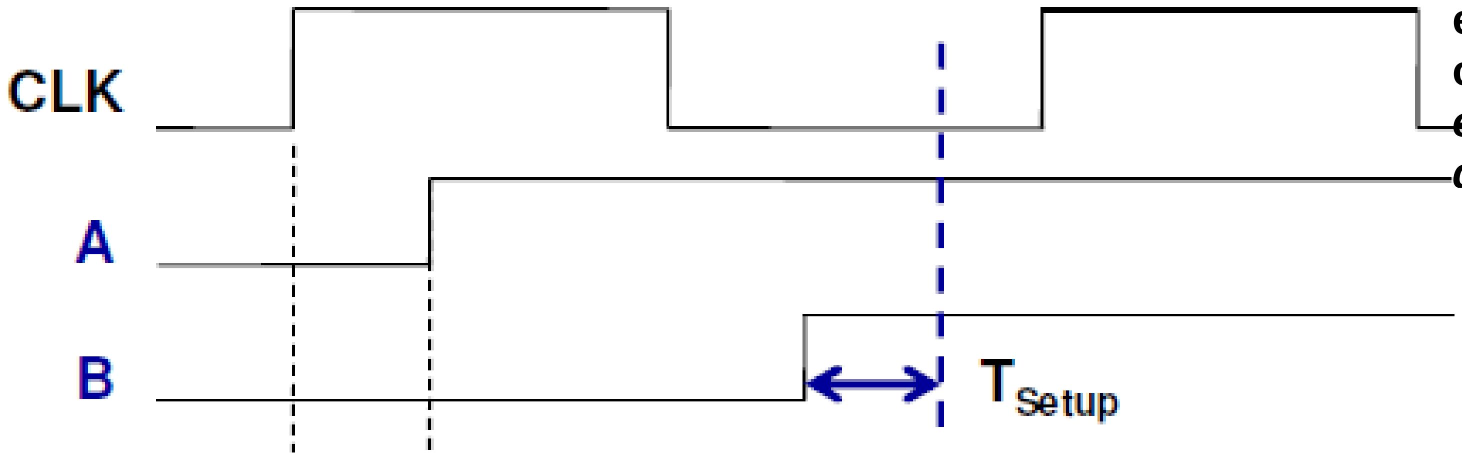
This is the time need for the logic to calculate a new output. Depends on the gates and wires



…Minimum Clock Period…

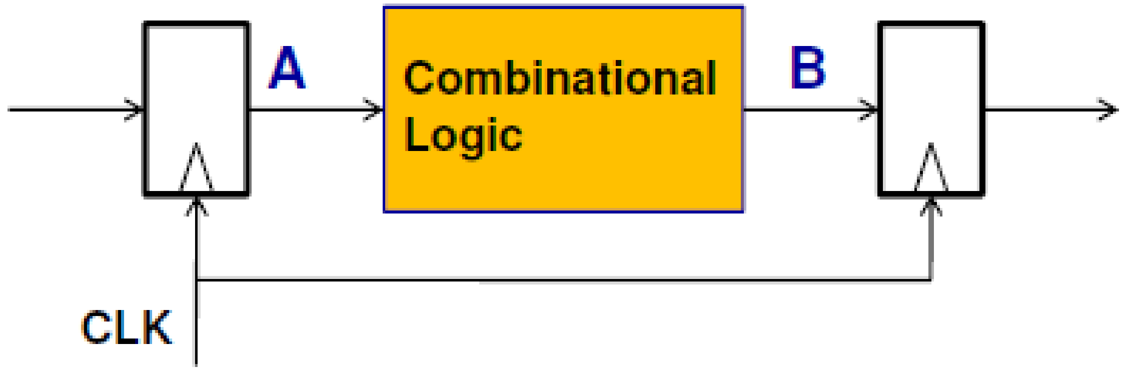


$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$

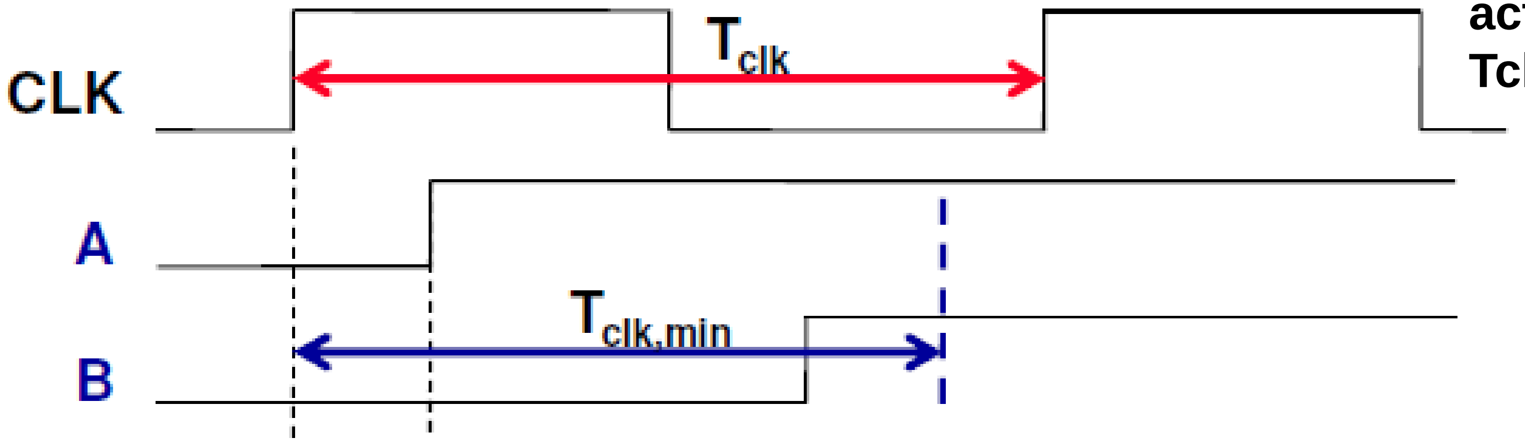


This is the time need for the flipflop to capture stable input data at the next clock edge. The next clock edge cannot come *earlier then the dashed line*

…Minimum Clock Period…

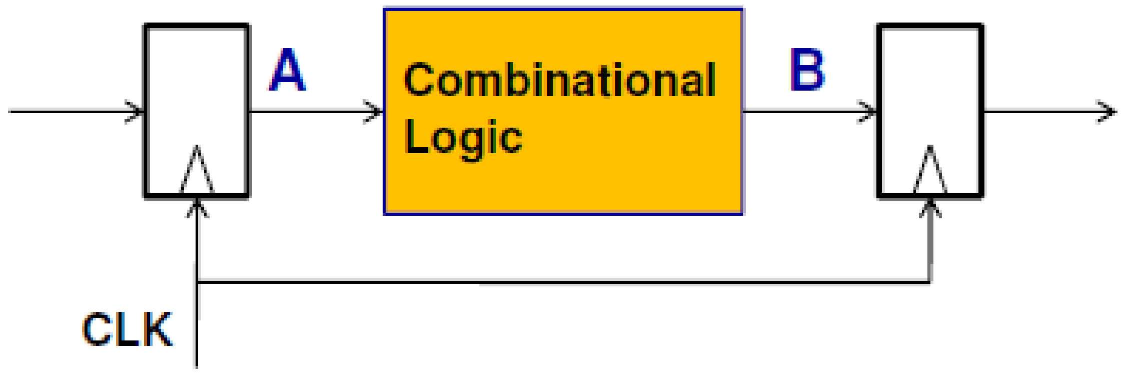


$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$

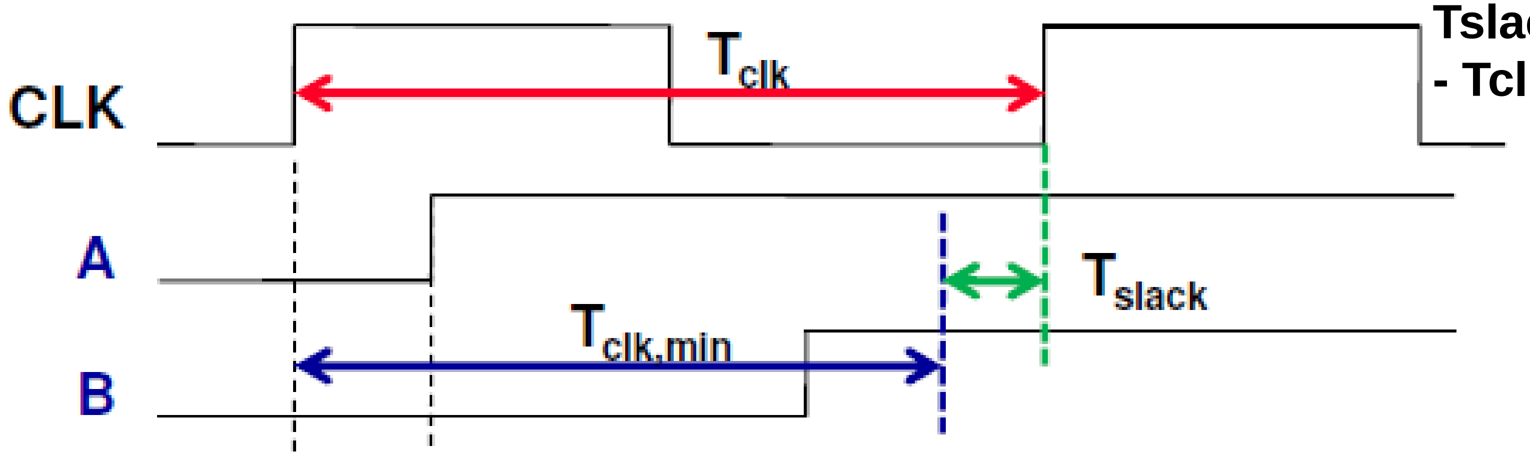


In this case, the timing of the system is OK, since the actual Tclk > Tclk,min

…Minimum Clock Period…

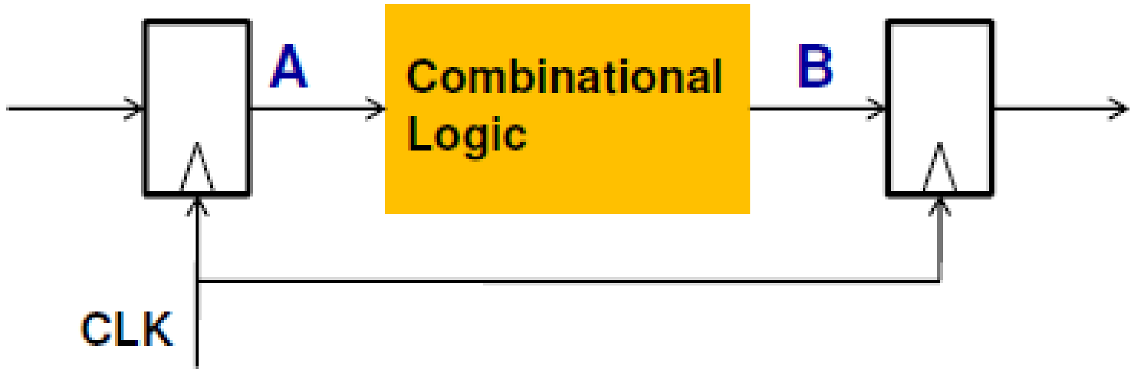


$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$

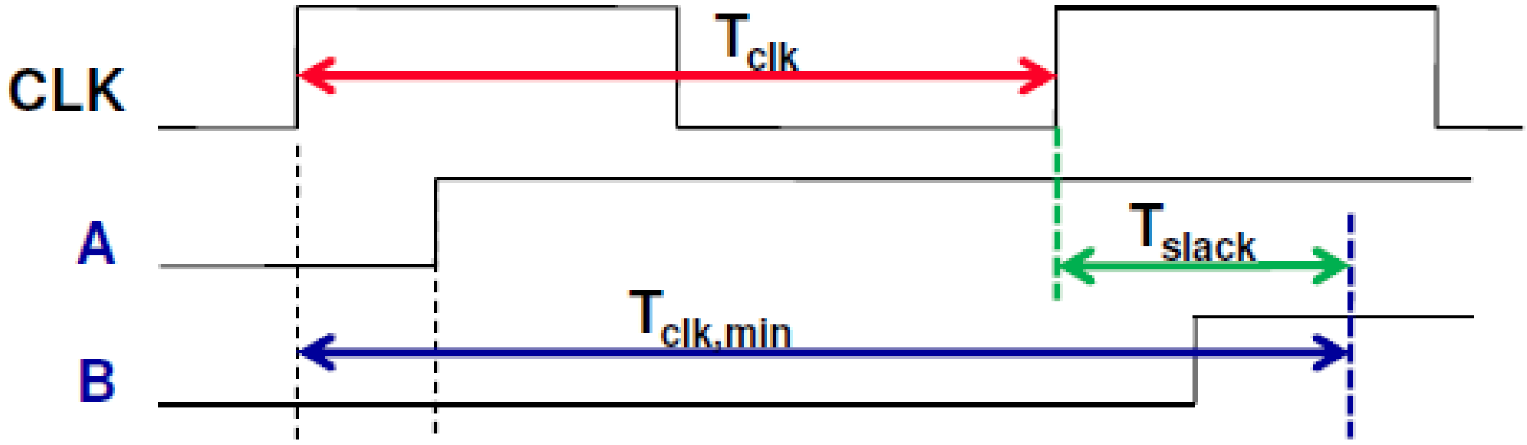


The margin between the actual clock period and the minimal clock period is called *slack.*
Tslack = Tclk - Tclk,min

…Minimum Clock Period…

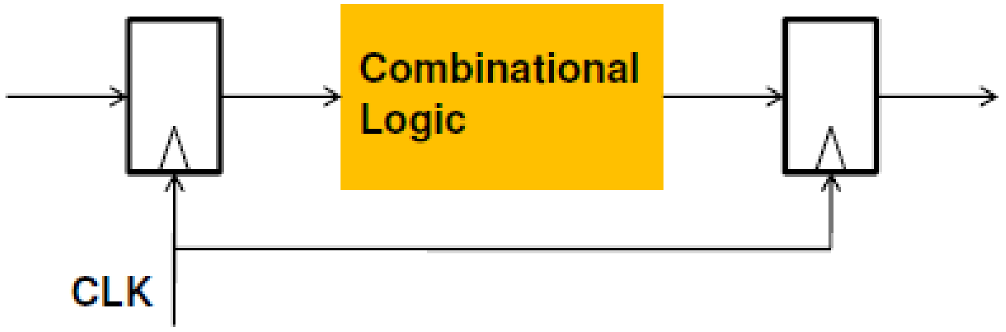


$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$



If the *slack is negative, the system has a timing violation. This system will not perform* as expected, since its clock frequency is too high.

…Minimum Clock Period…



$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$

Once the technology is chosen, Tclk->Q and Tsetup are fixed.

An example from the Xilinx device datasheet is shown on the right.

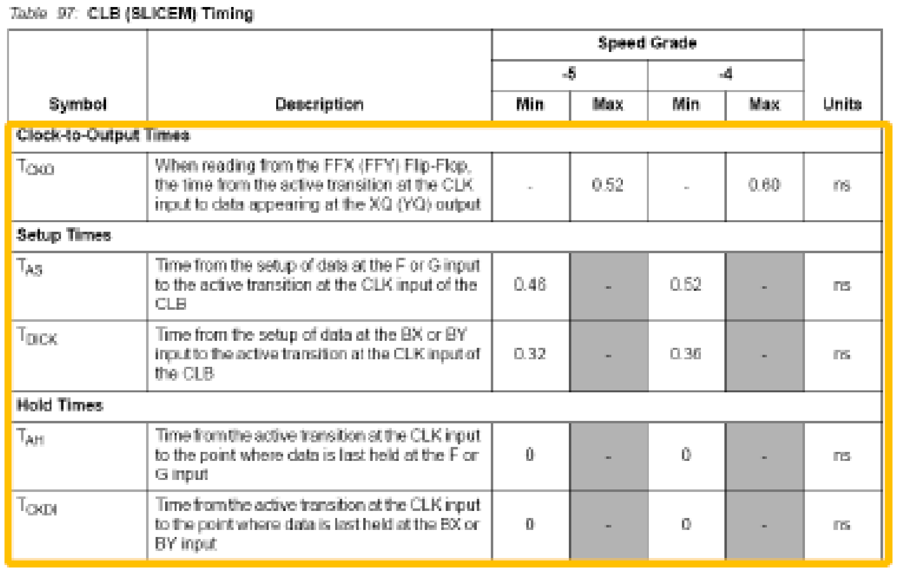
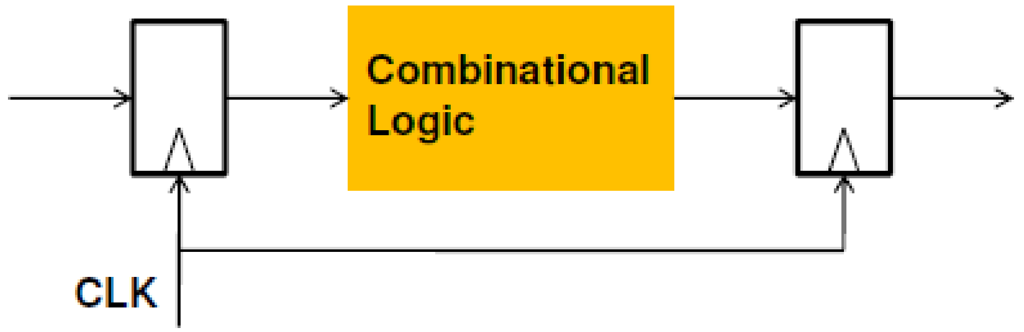


Table 97: **CLB (SLICEM) Timing**

Symbol	Description	Speed Grade -5 Min	Speed Grade -5 Max	Speed Grade -4 Min	Speed Grade -4 Max	Units
Clock-to-Output Times						
T_{CKO}	When reading from the FFX (FFY) Flip-Flop, the time from the active transition at the CLK input to data appearing at the XQ (YQ) output	-	0.52	-	0.60	ns
Setup Times						
T_{AS}	Time from the setup of data at the F or G input to the active transition at the CLK input of the CLB	0.46	-	0.52	-	ns
T_{DICK}	Time from the setup of data at the BX or BY input to the active transition at the CLK input of the CLB	0.32	-	0.36	-	ns
Hold Times						
T_{AH}	Time from the active transition at the CLK input to the point where data is last held at the F or G input	0	-	0	-	ns
T_{CKDI}	Time from the active transition at the CLK input to the point where data is last held at the BX or BY input	0	-	0	-	ns

…Minimum Clock Period



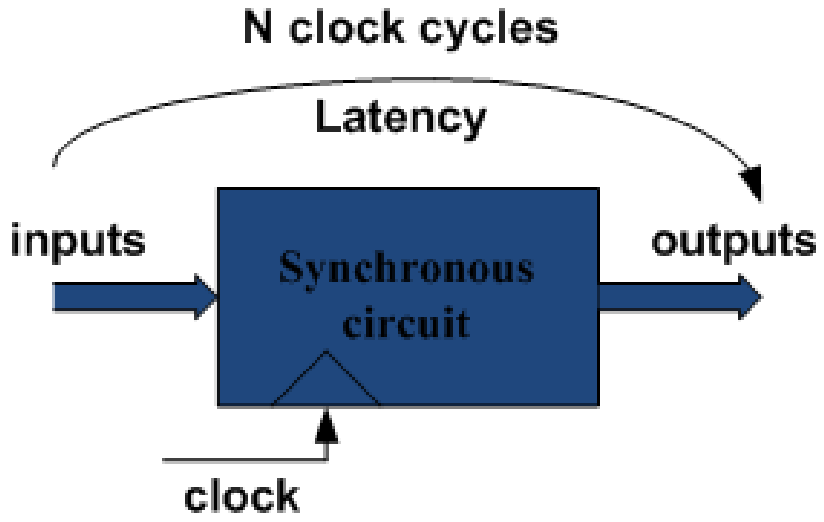
$$T_{clk,min} = T_{clk\text{->}Q} + T_{Logic} + T_{Routing} + T_{Setup}$$

However, even after the technology is chosen, the designer can *still influence Tlogic and Trouting* by making modifications to the HDL code.

Thus, if we want to decrease the minimum clock period, we need to consider these terms.

Minimization of Delay

Delay = latency (clock cycles) x clock period



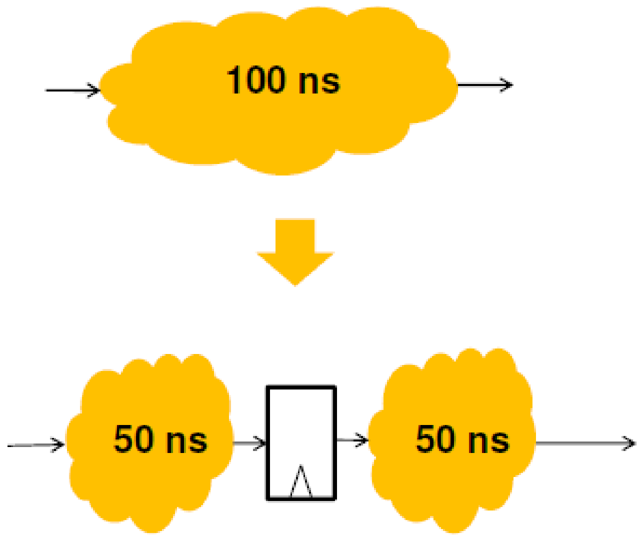
Parallel Computations

Reduce the # cycles required

Pipelining and Retiming

Reduce the clock period

Pipelining and Retiming

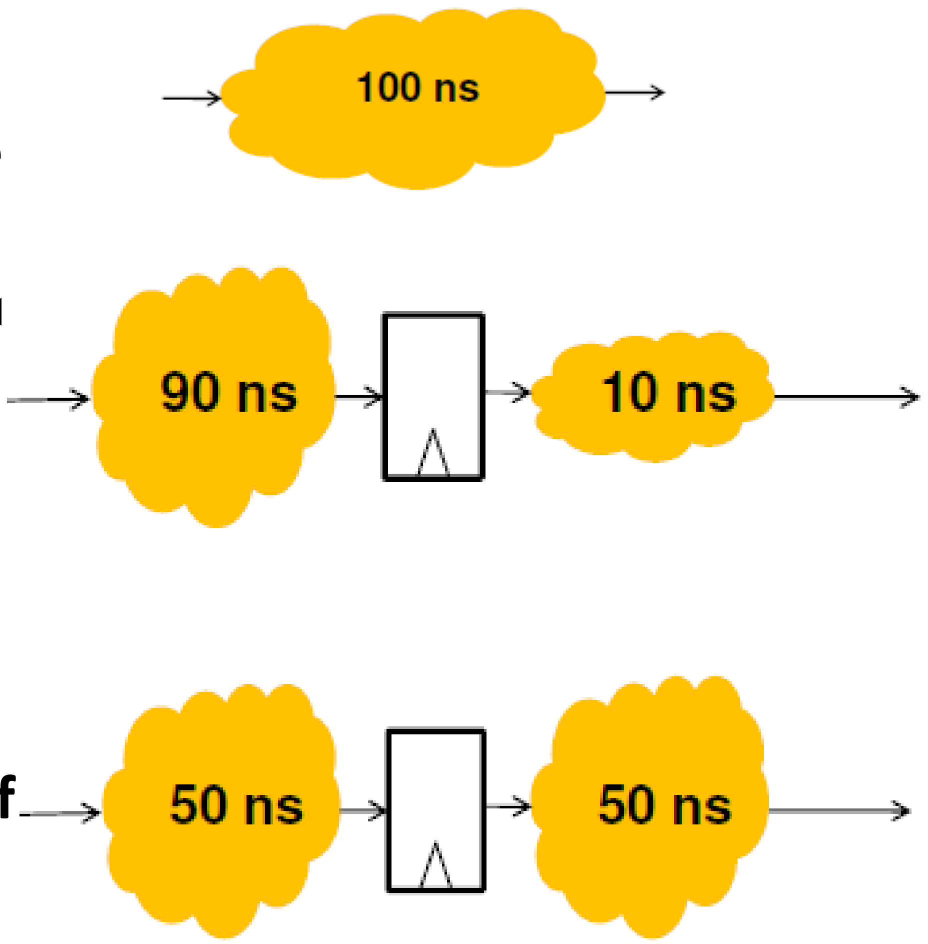


A pipeline register can cut a piece of combinational logic in smaller pieces. This reduces the Tclk,min for the entire design

Retiming

- **Sometimes, the partitioning is not nicely 50/50. In that case the benefit of pipeline registers to reduce Tclk,min is small, since th design has to be operated at the speed of the slowest stage**

- **To maximize the benefit of the (pipeline) registers, they should be balanced so that each stage of combinational logic takes the same amount of logic delay**



Pipelining vs Retiming

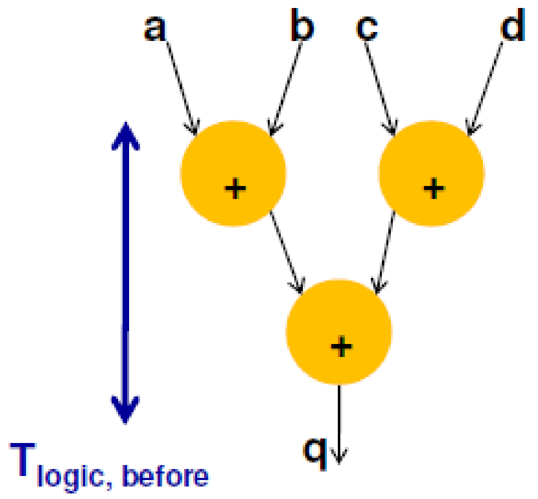
- **Pipelining** is done by the designer, typically by rewriting HDL

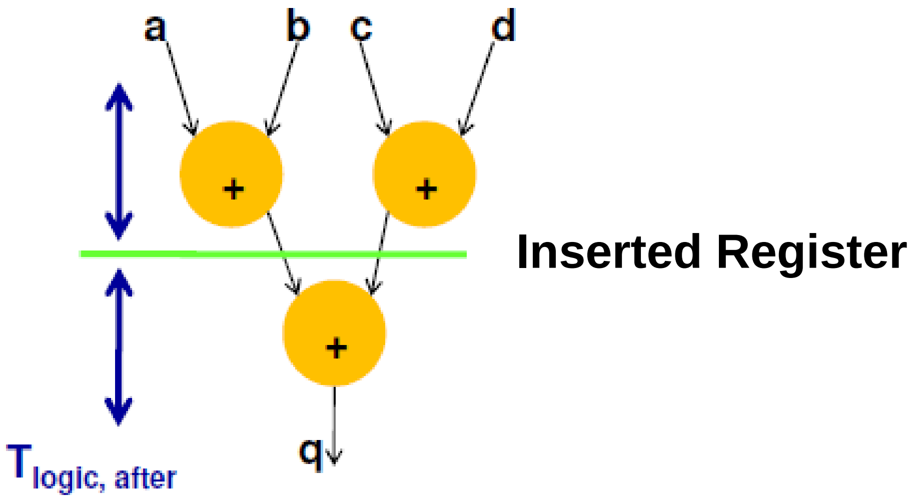
- **Retiming** is done by the tools, during logic Synthesis

  Of course, the designer can also rewrite the HDL

Pipelining

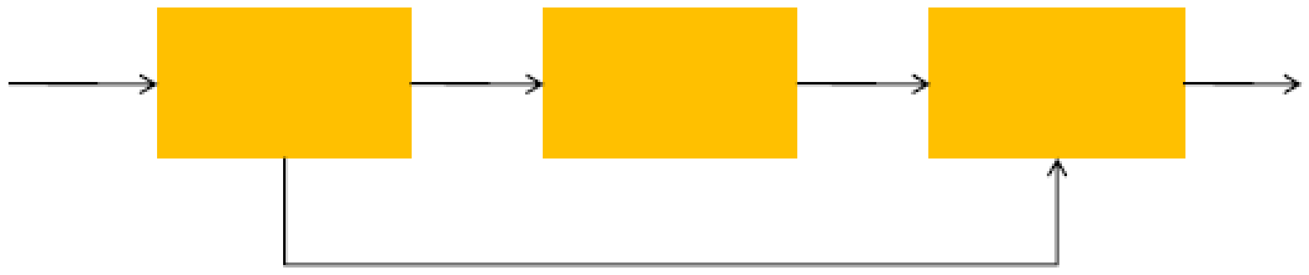
- Cut a long combinational path in half by inserting a register
- Increases the latency cycle count of the design to get form the input to the output, you will need an extra clock cycle



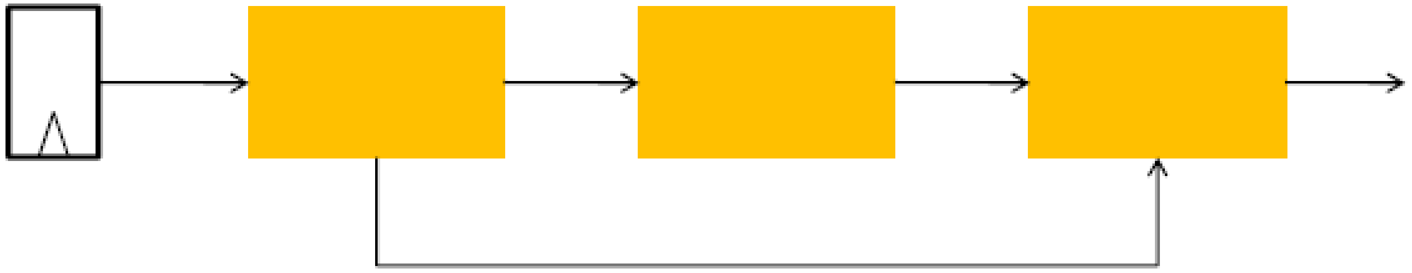


Rules for Consistent Pipelining...

- Assume a network of modules (combinational or sequential) as follows.
- We will demonstrate how to move pipeline registers around while avoiding inconsistent pipelining

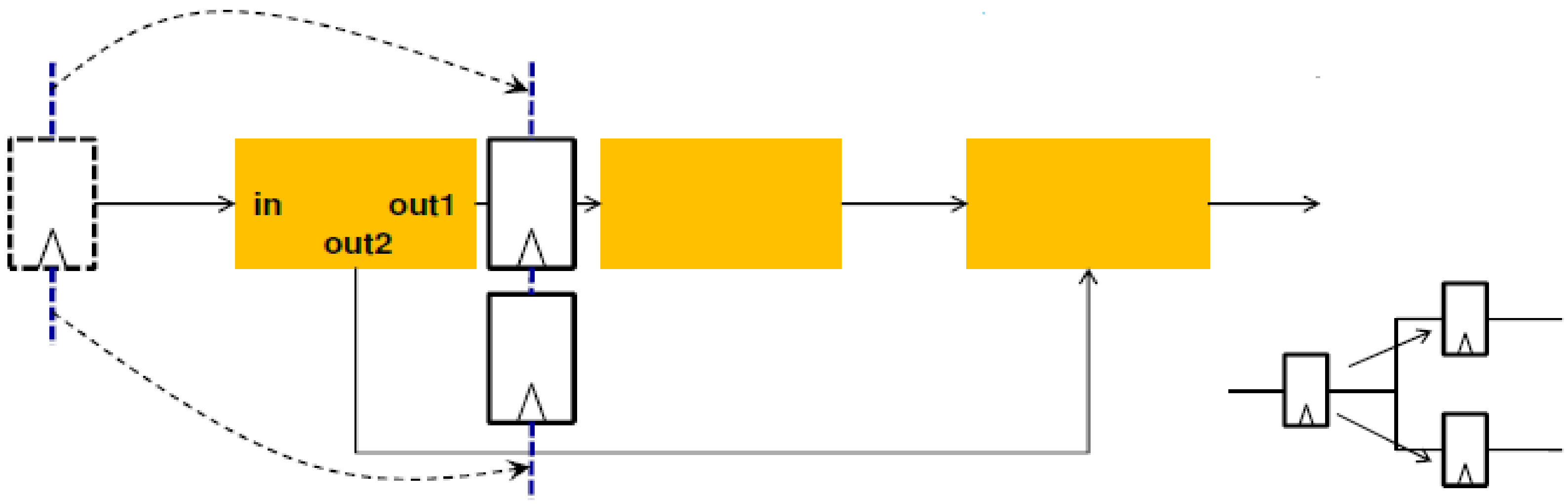


…Rules for Consistent Pipelining…



- You can add a register in front. It increases the latency of the network with one cycle, but the network will have the same functionality

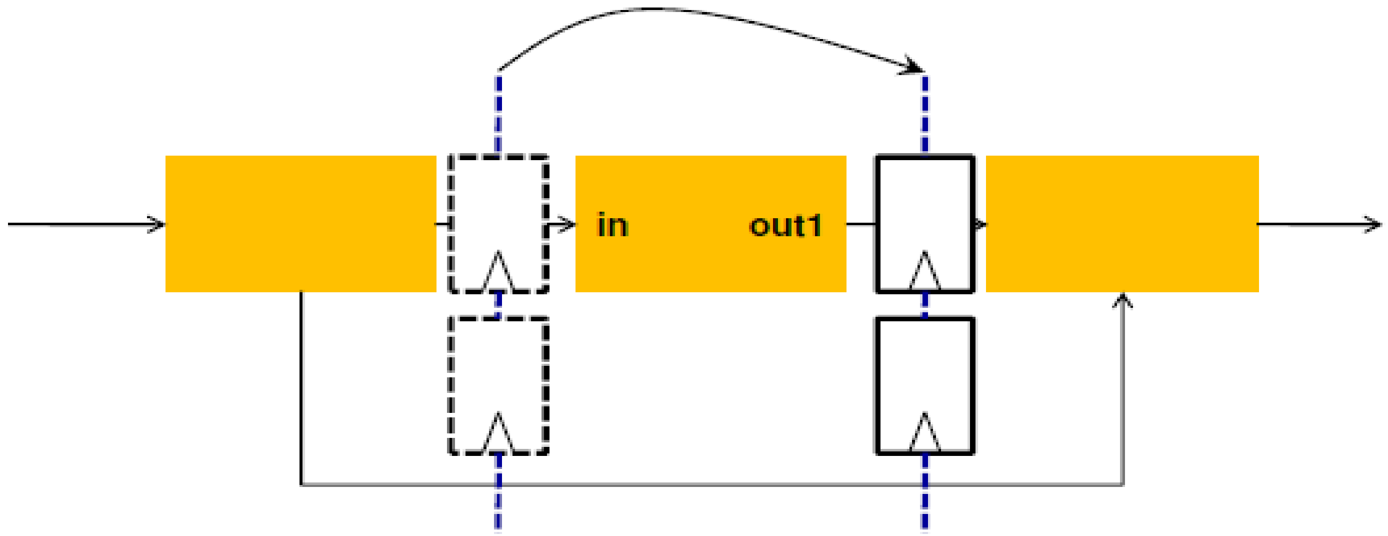
…Rules for Consistent Pipelining…



- You can absorb a register at a single input if you recreate it at ALL the outputs of the module. This transformation will not change the latency nor the functionality of the network.

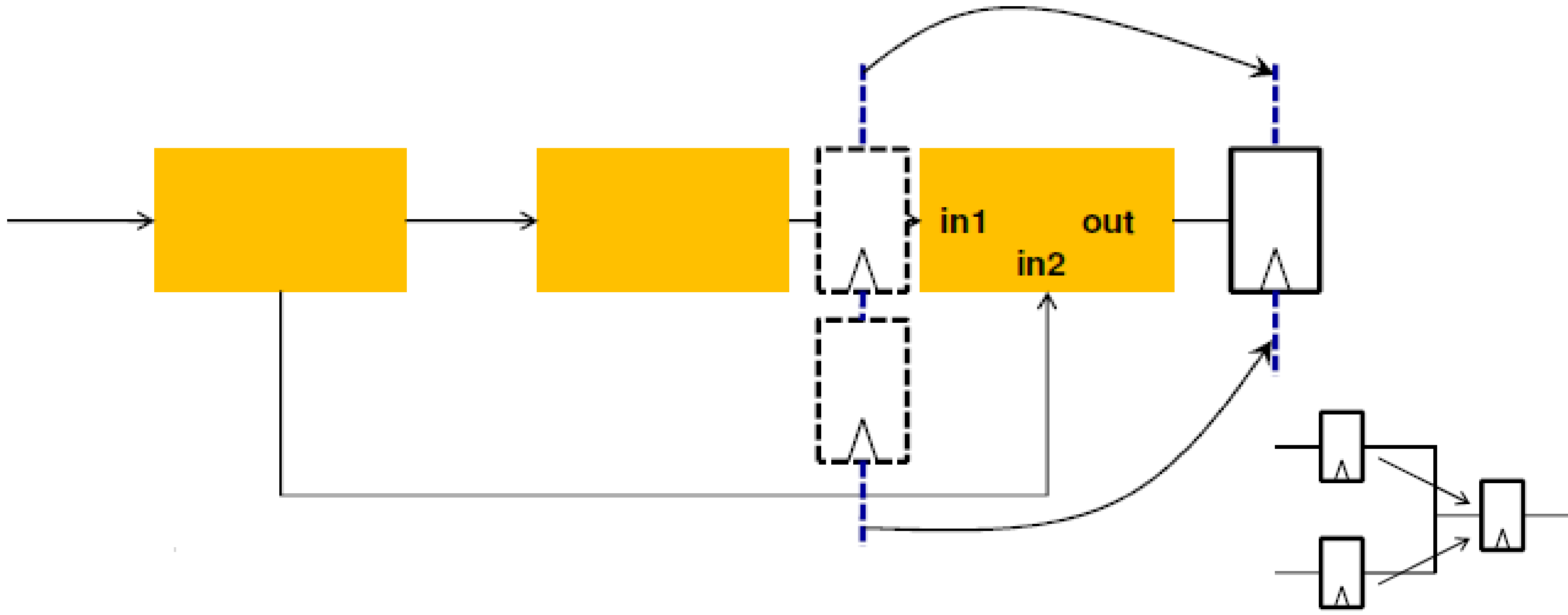
…Rules for Consistent Pipelining…

- Move it over another module – absorb register at the module inputs, recreate it to the module outputs



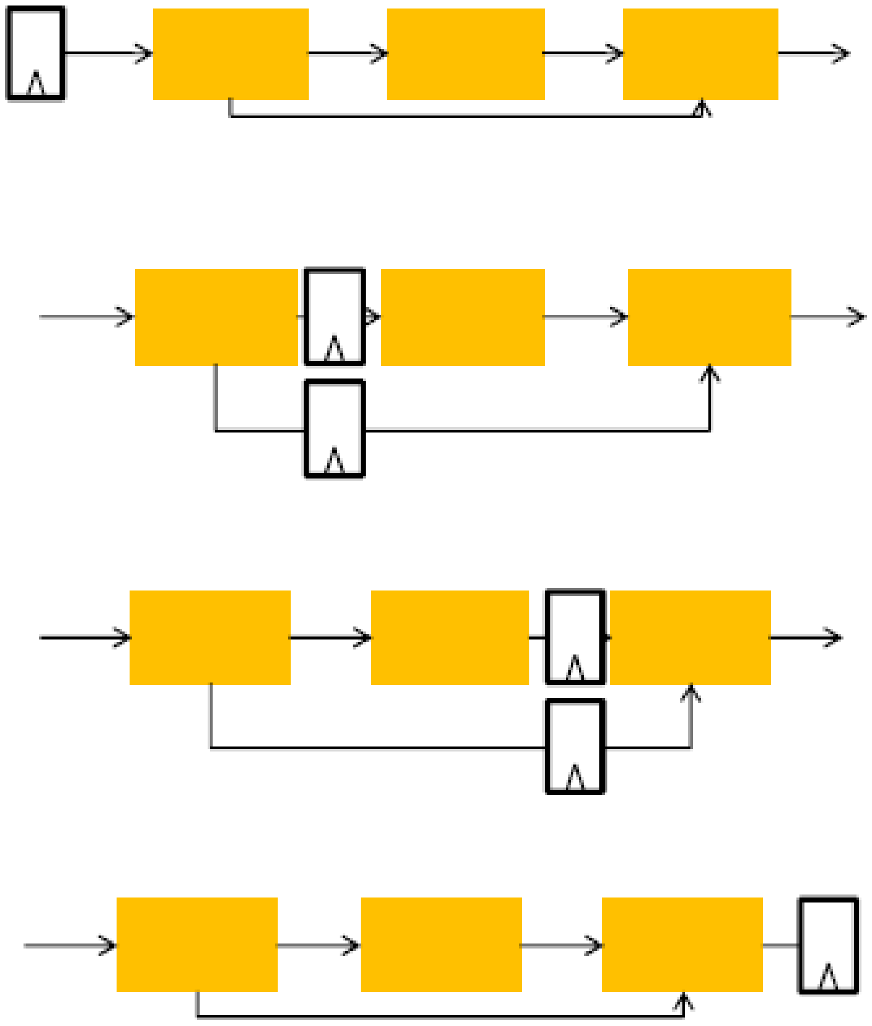
…Rules for Consistent Pipelining…

- Move it over the last module – absorb register at the module inputs, recreate it at the module output



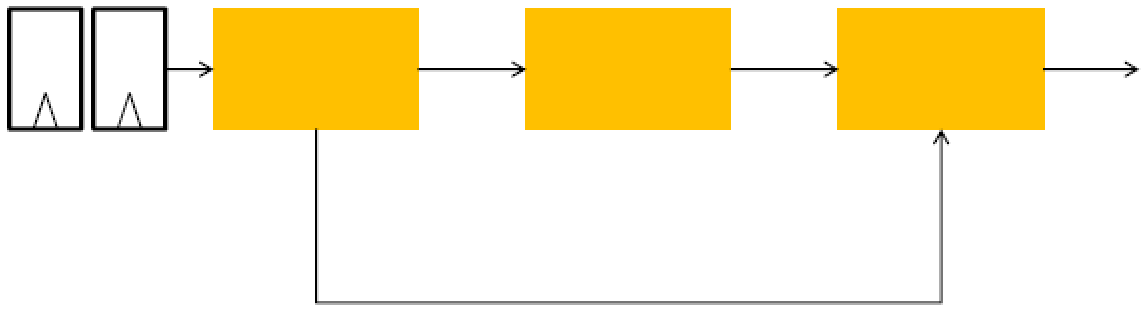
…Rules for Consistent Pipelining…

All of these have the same behavior



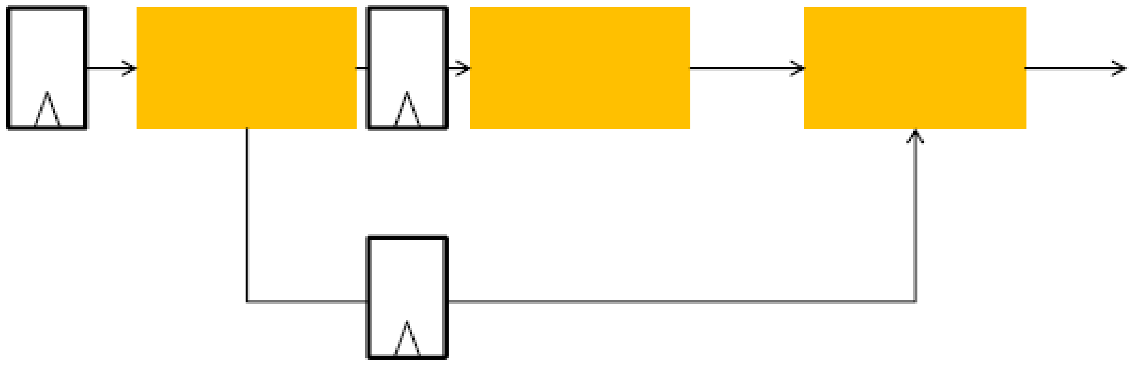
…Rules for Consistent Pipelining...

- We can add multiple registers at the front ...



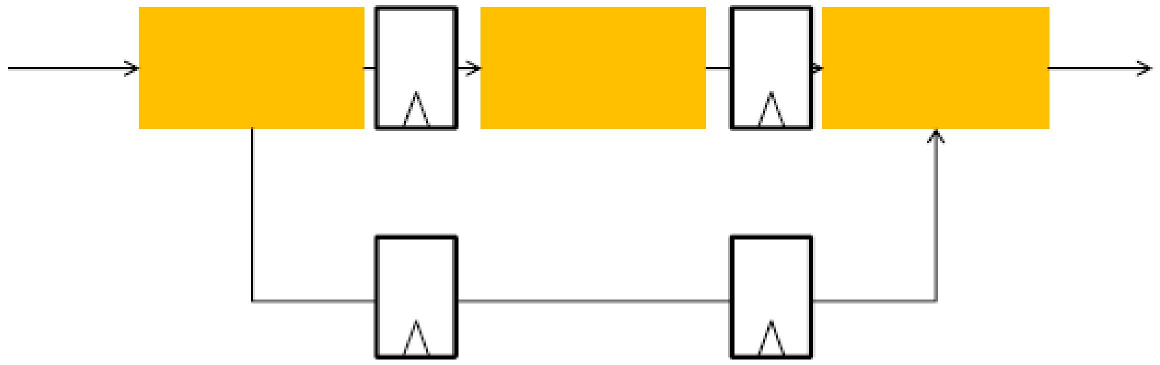
…Rules for Consistent Pipelining…

- and redistribute them using consistent pipelining



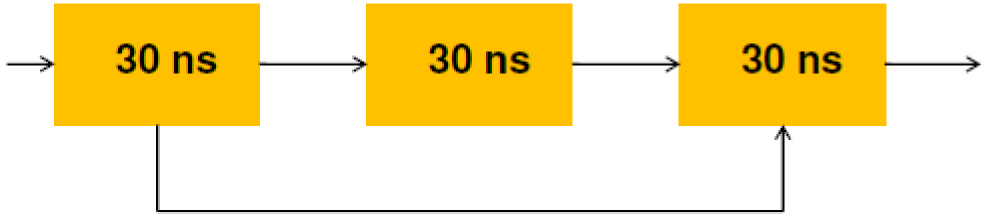
…Rules for Consistent Pipelining…

- Or…

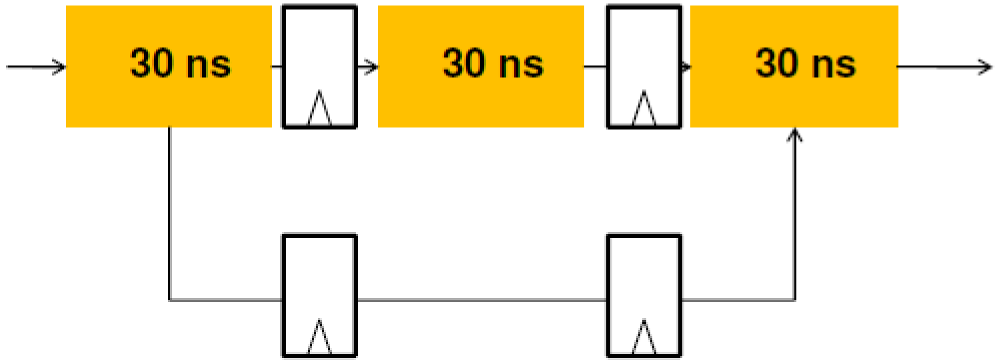


…Rules for Consistent Pipelining…

Tclk,min = 90ns
Latency = 1 cycle
Throughput = 1 / cycle

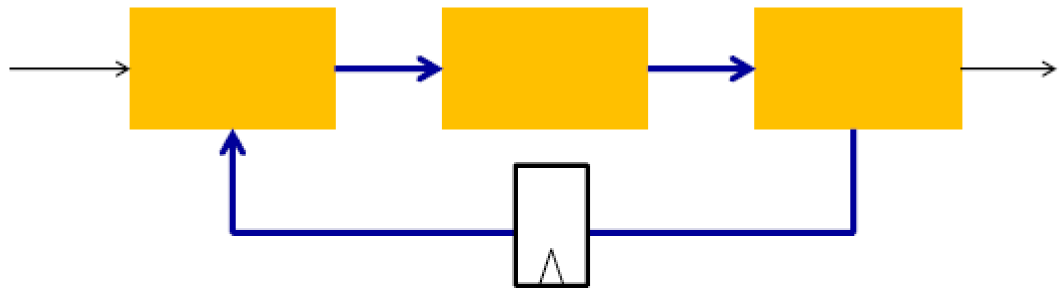


Tclk,min = 30ns
Latency = 3 cycles
Throughput = 1 / cycle



…Rules for Consistent Pipelining…

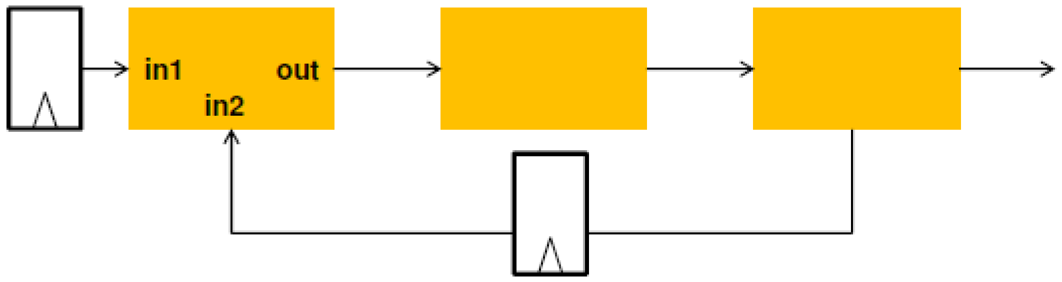
- Following these rules, you'll find that you cannot pipeline loops (i.e. increase the number of registers in a feedback path)



There is a single register in this path

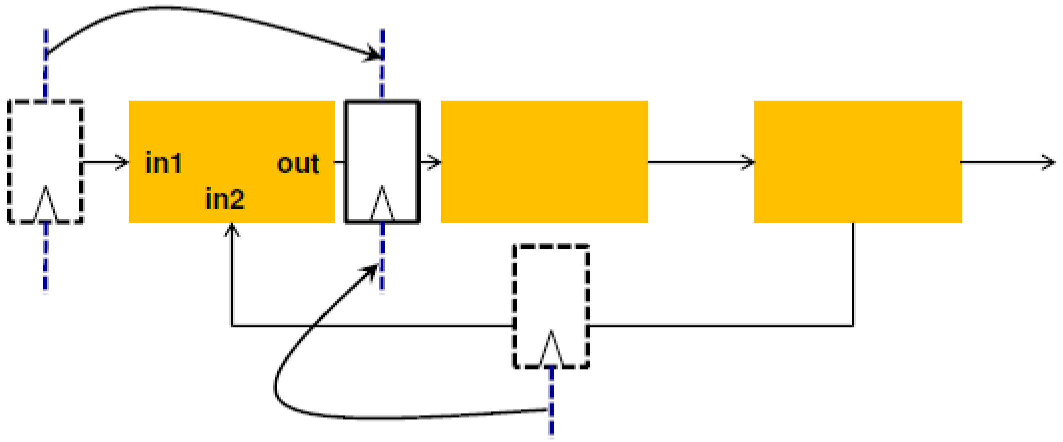
…Rules for Consistent Pipelining…

- To pipeline, add a register at the front



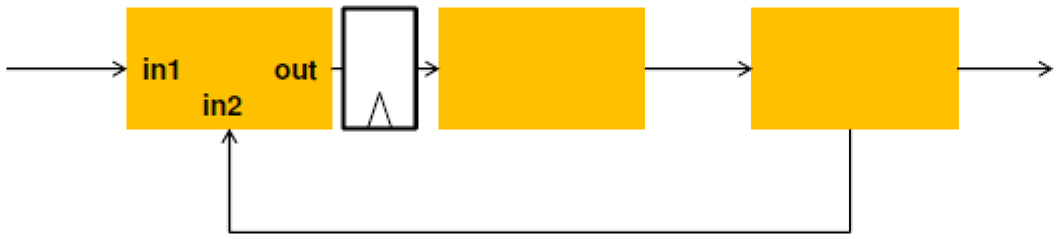
…Rules for Consistent Pipelining…

- To move the pipeline register to the module output, ALL the inputs need to absorb a register



…Rules for Consistent Pipelining…

- In the resulting network, there is still only one register in the loop

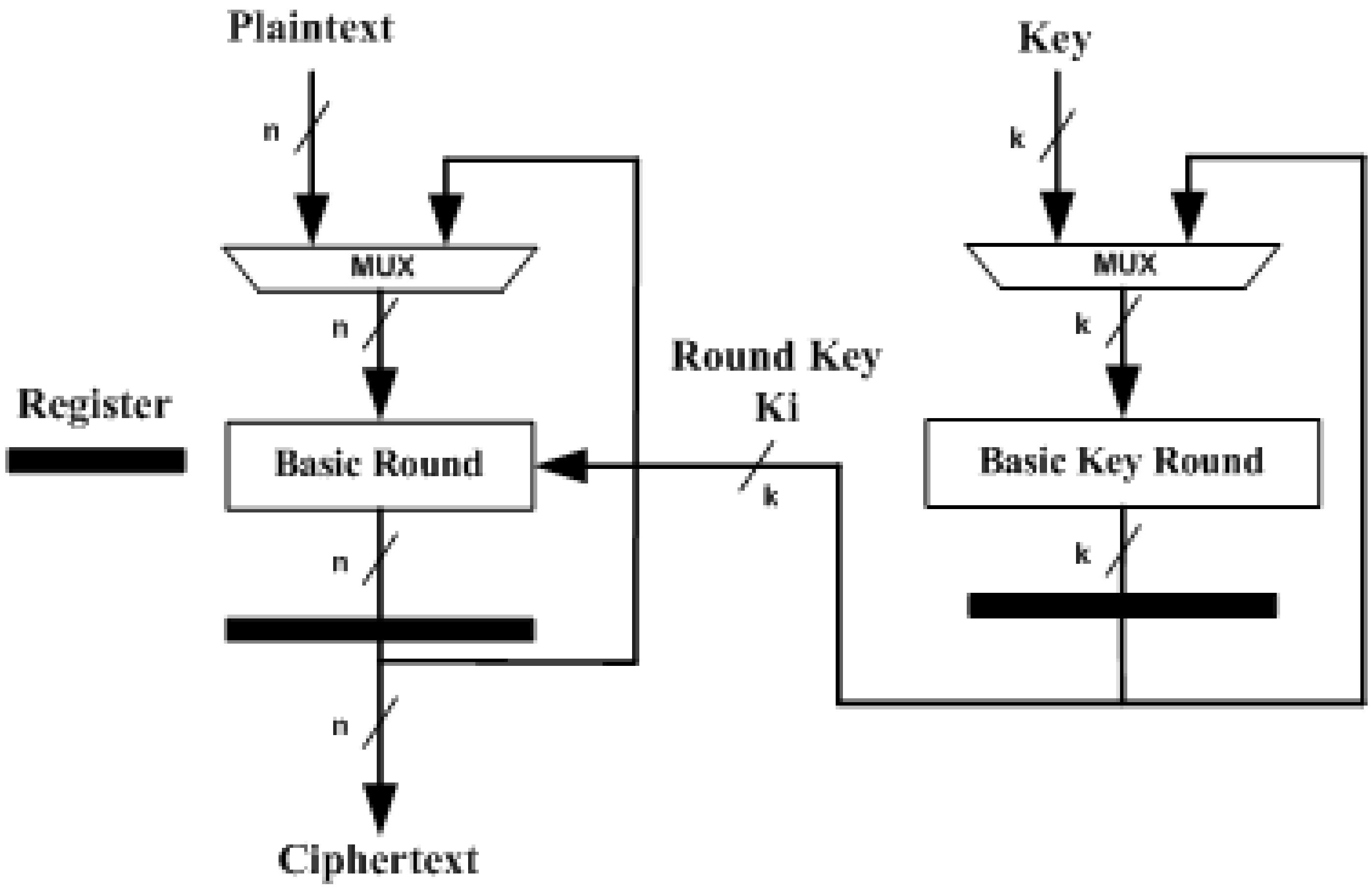


- **Part 2 – Hardware architectures (Block ciphers and Hash Function)**

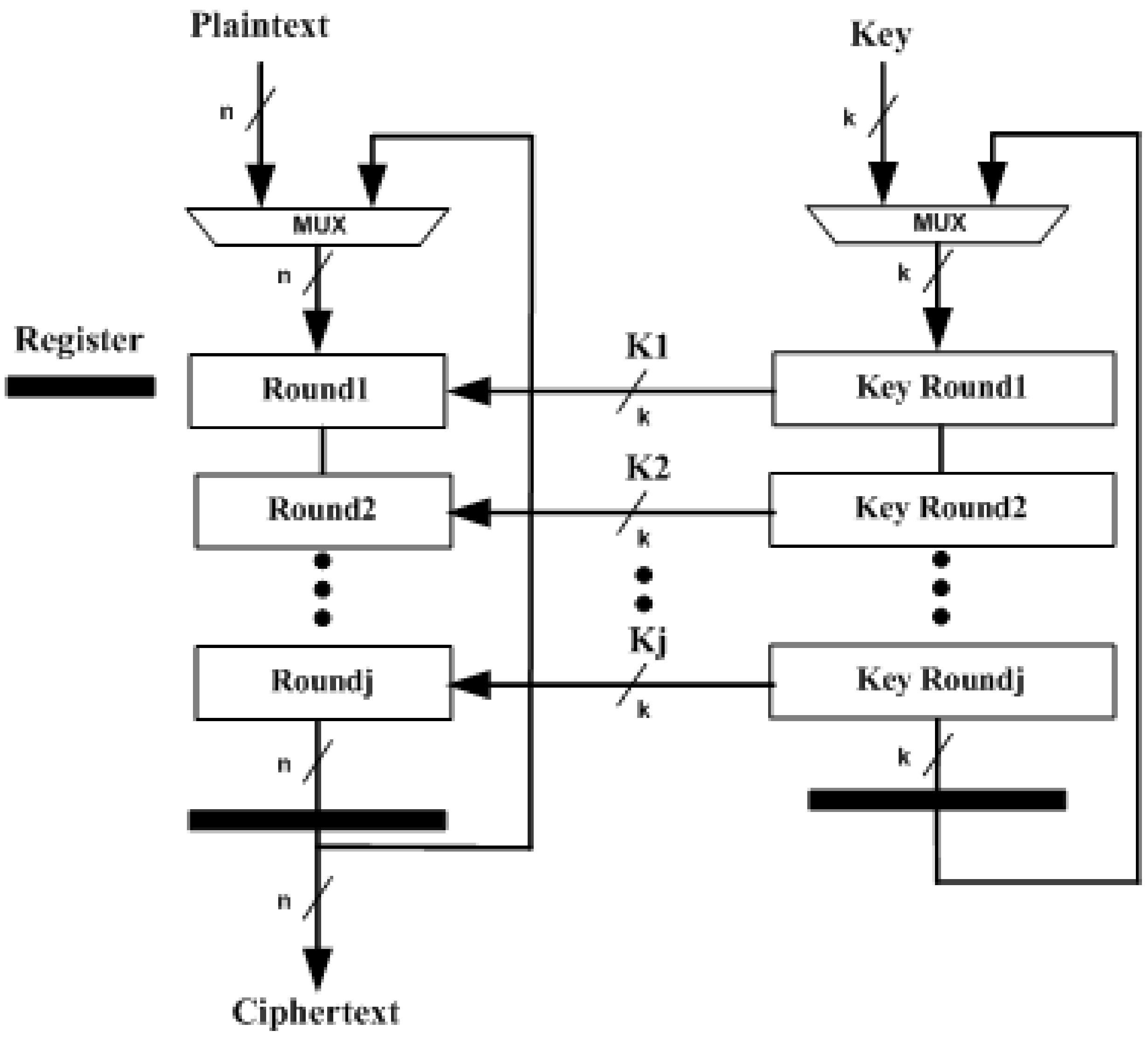
Basic Architectures

- There are four types of architectures about bloc ciphers
  - Iterative architecture
    - Use only one round
  - Partial loop unrolling
    - Use more rounds
  - Loop unrolling
    - Use all rounds (Outer-round pipelining)
  - Use inner- and outer-round pipelining

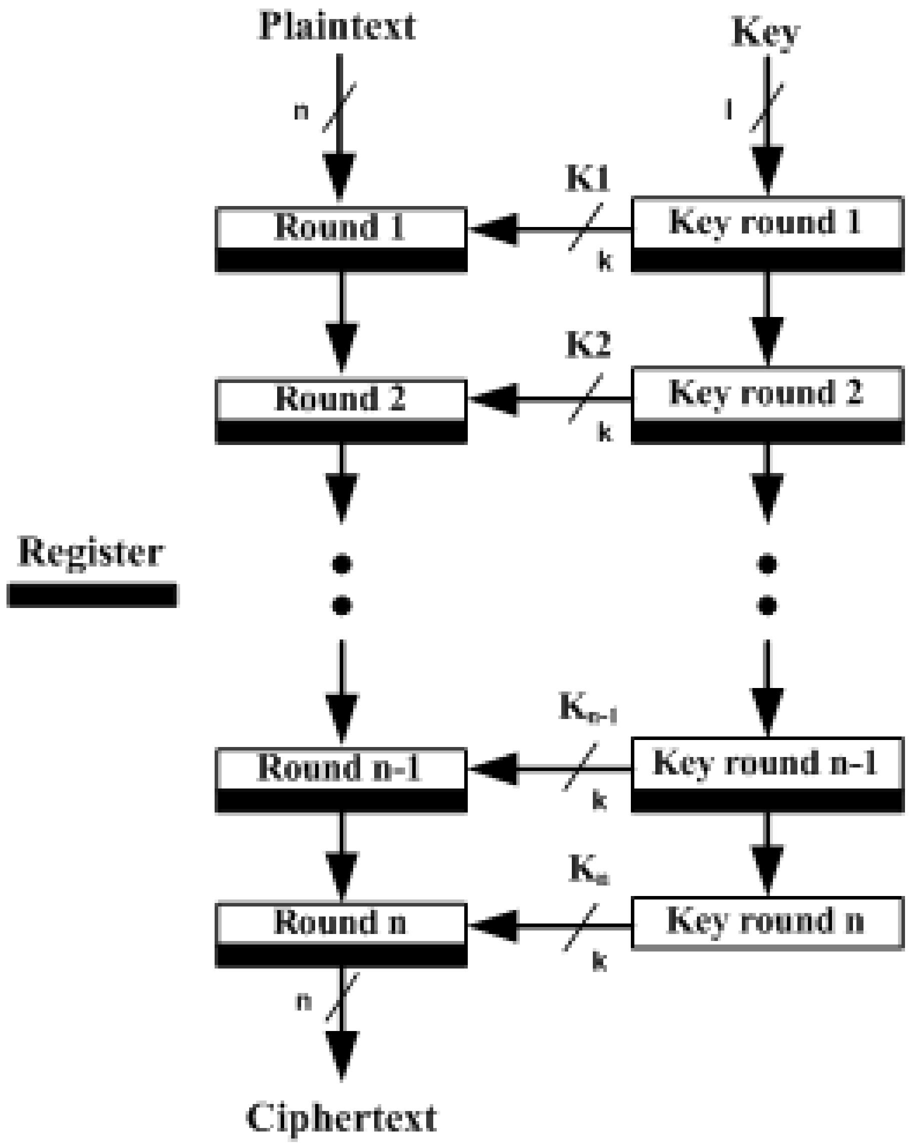
Iterative architecture



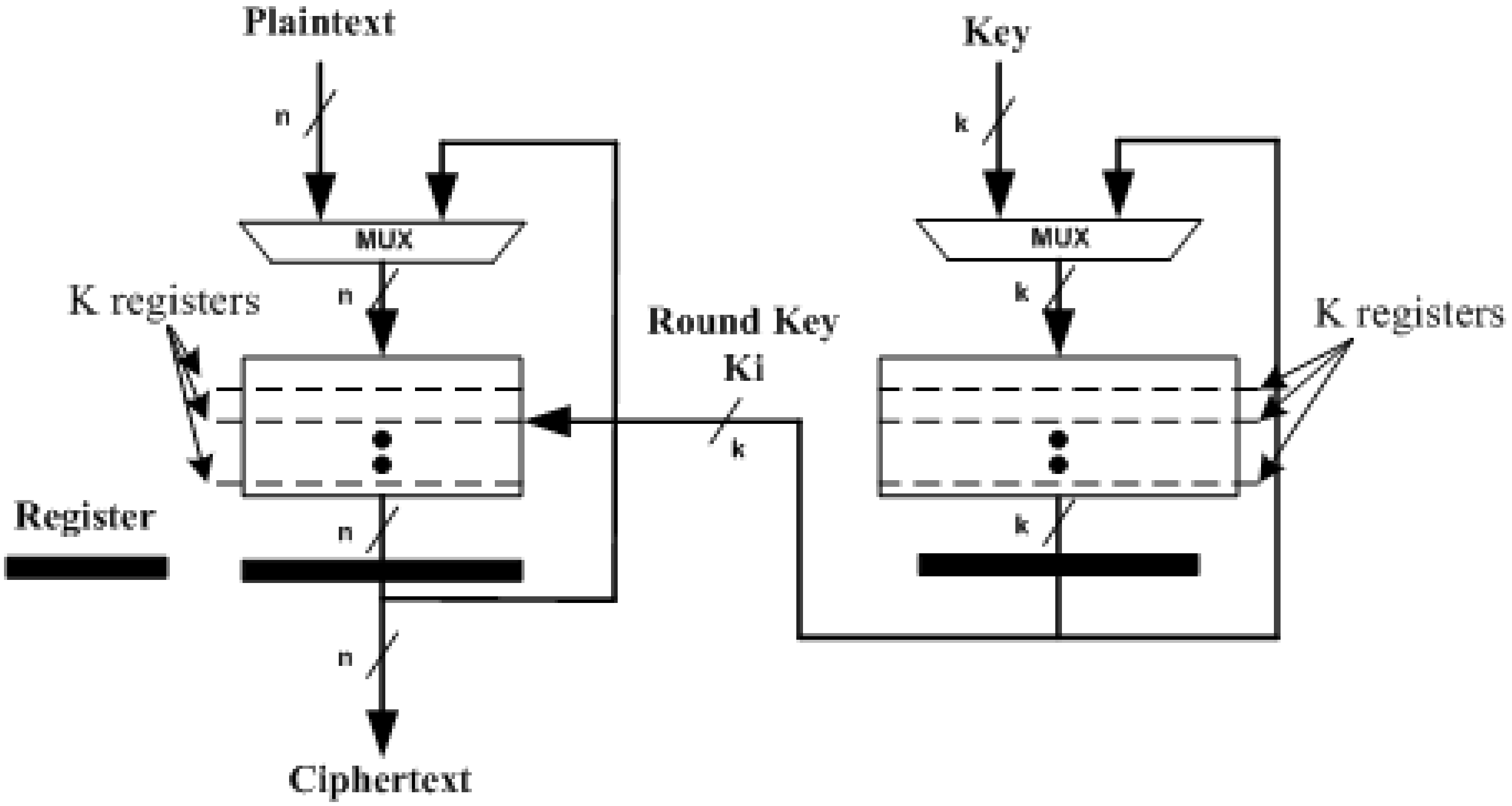
Partial loop unrolling



Loop unrolling

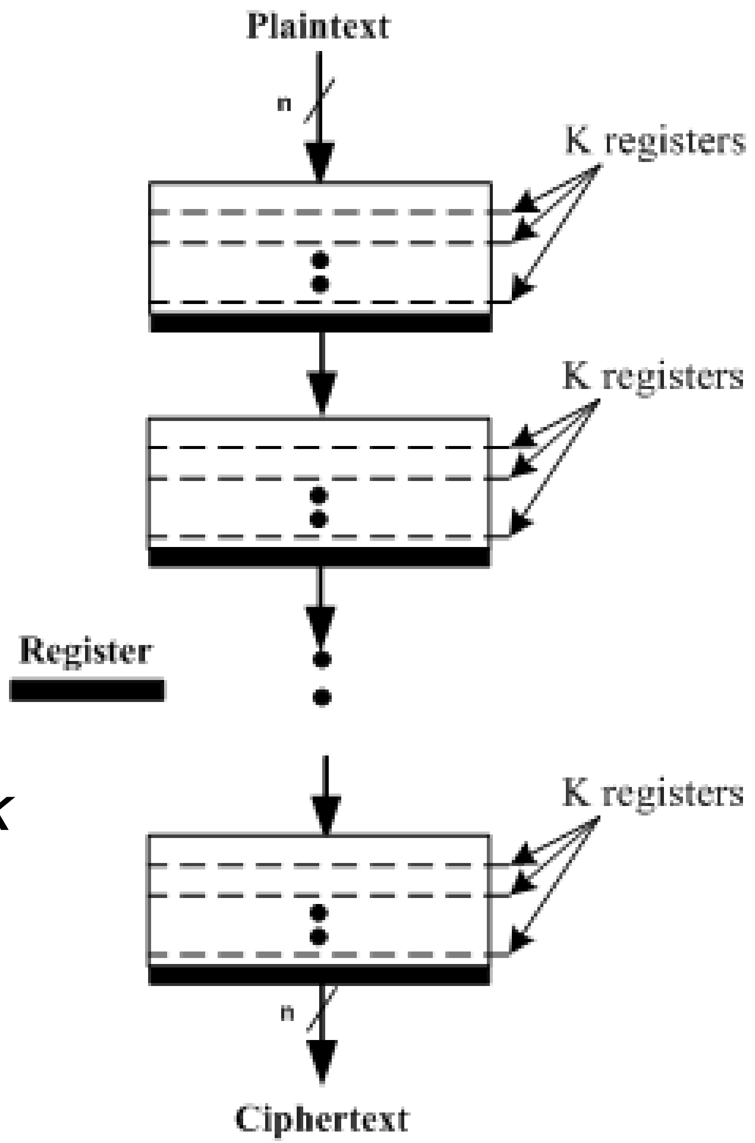


Inner- and outer-round pipelining…



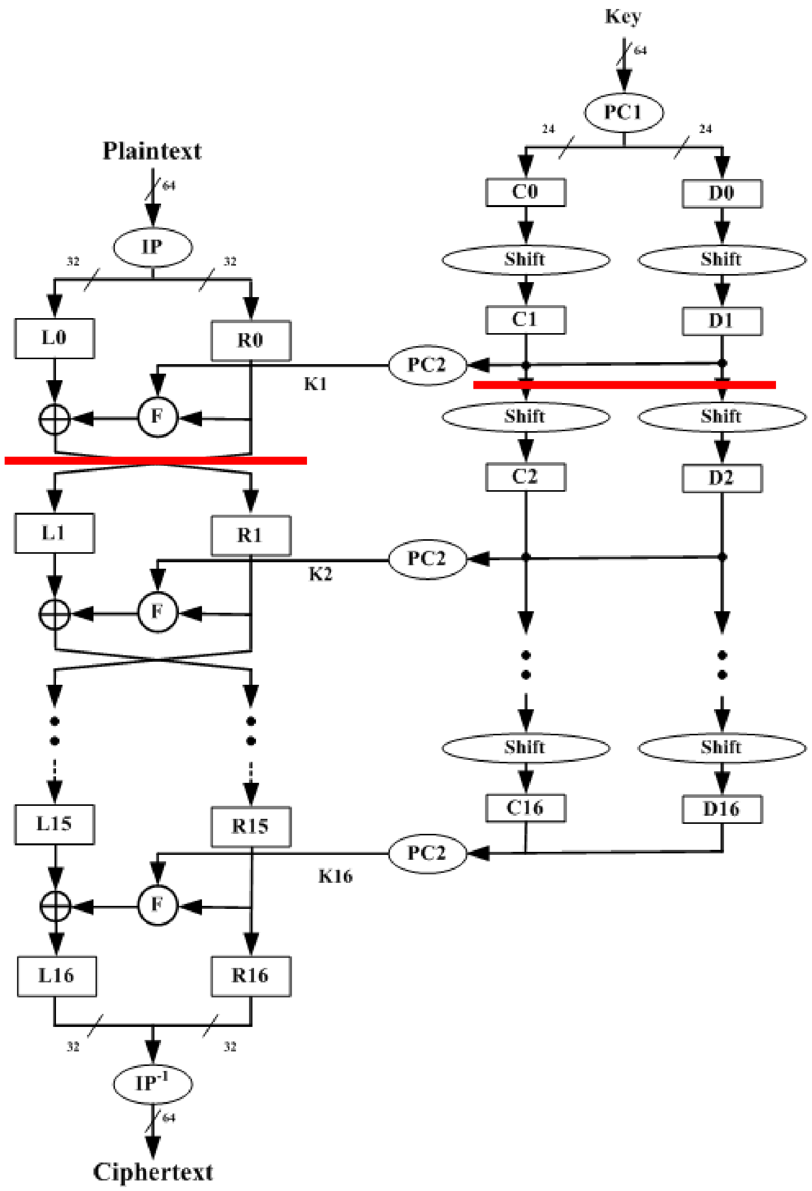
Total # of pipeline stages = #rounds·K (K=1)

…Inner- and outer-round pipelining



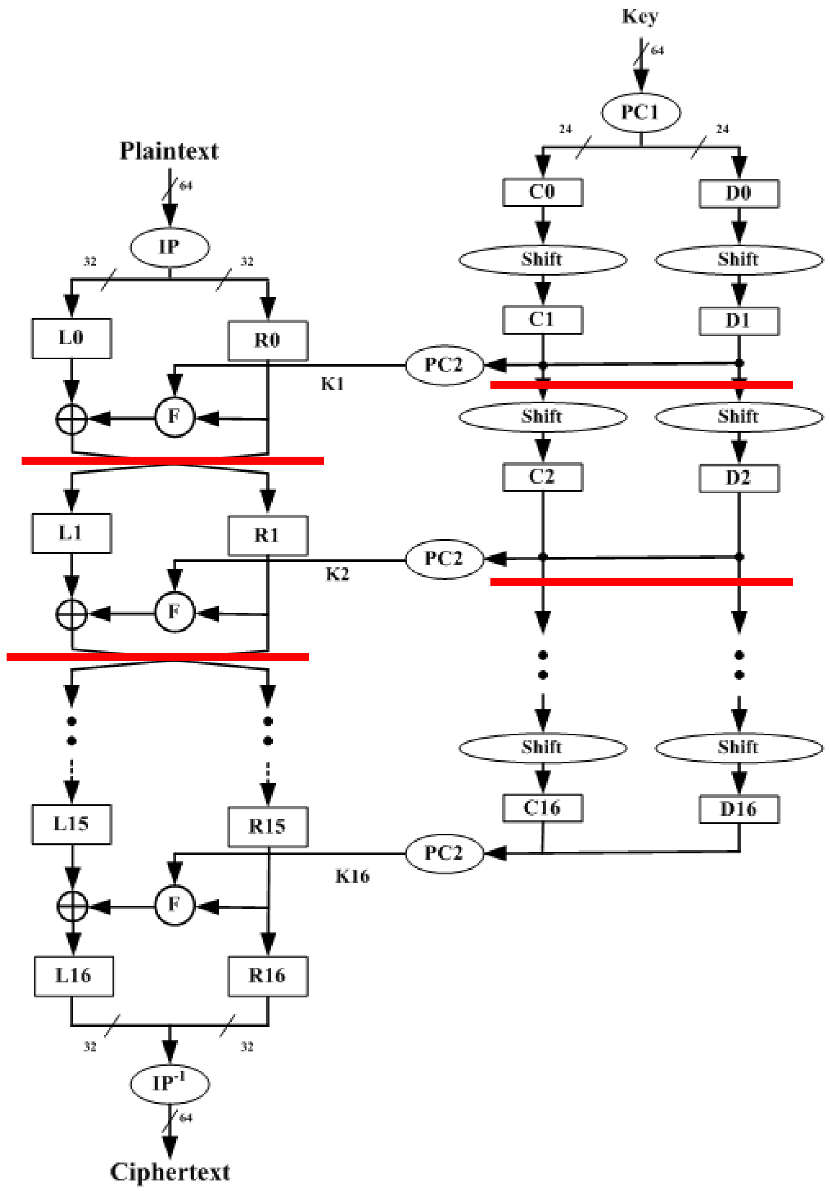
Total # of pipeline stages = #rounds·K

Partial loop unrolling example: DES



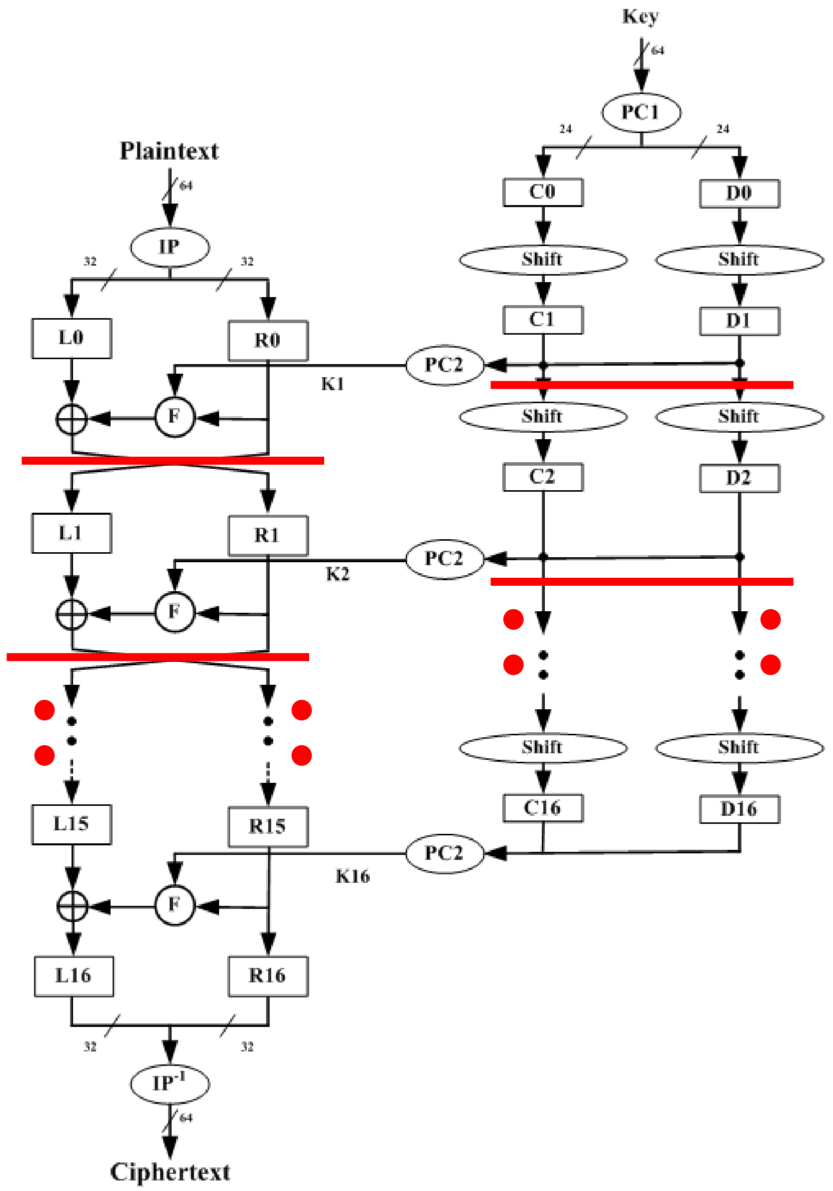
register

…DES…



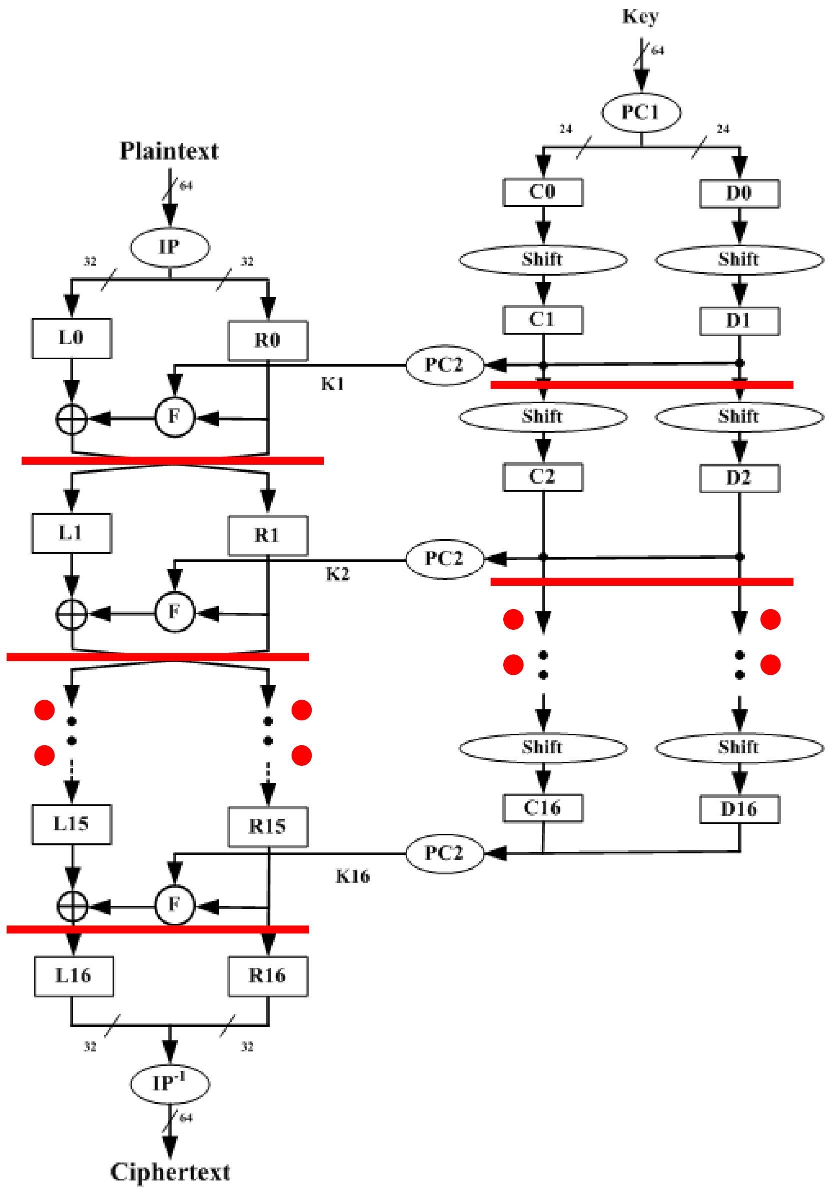
🟥 register

…DES…



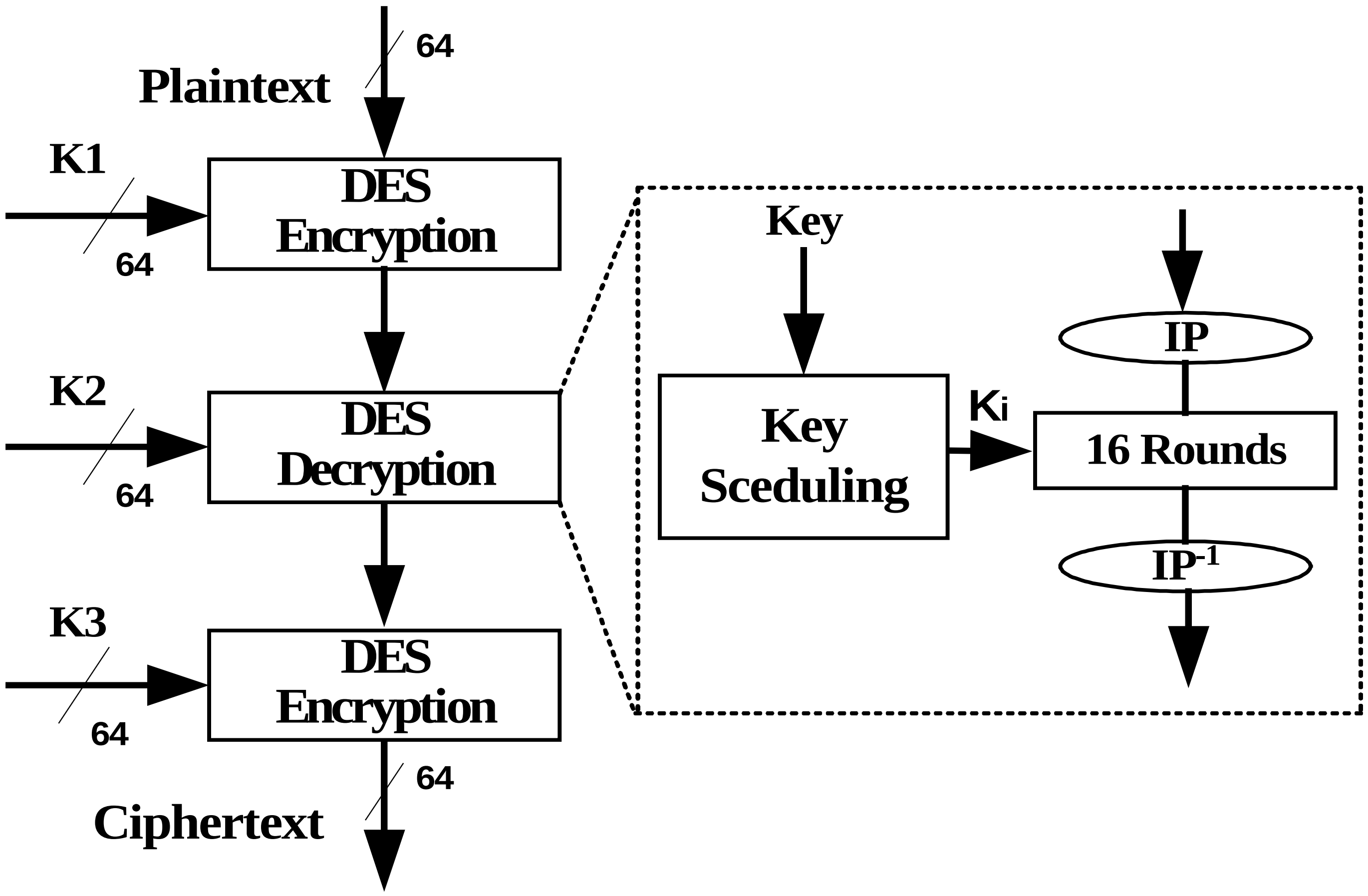
register

…DES



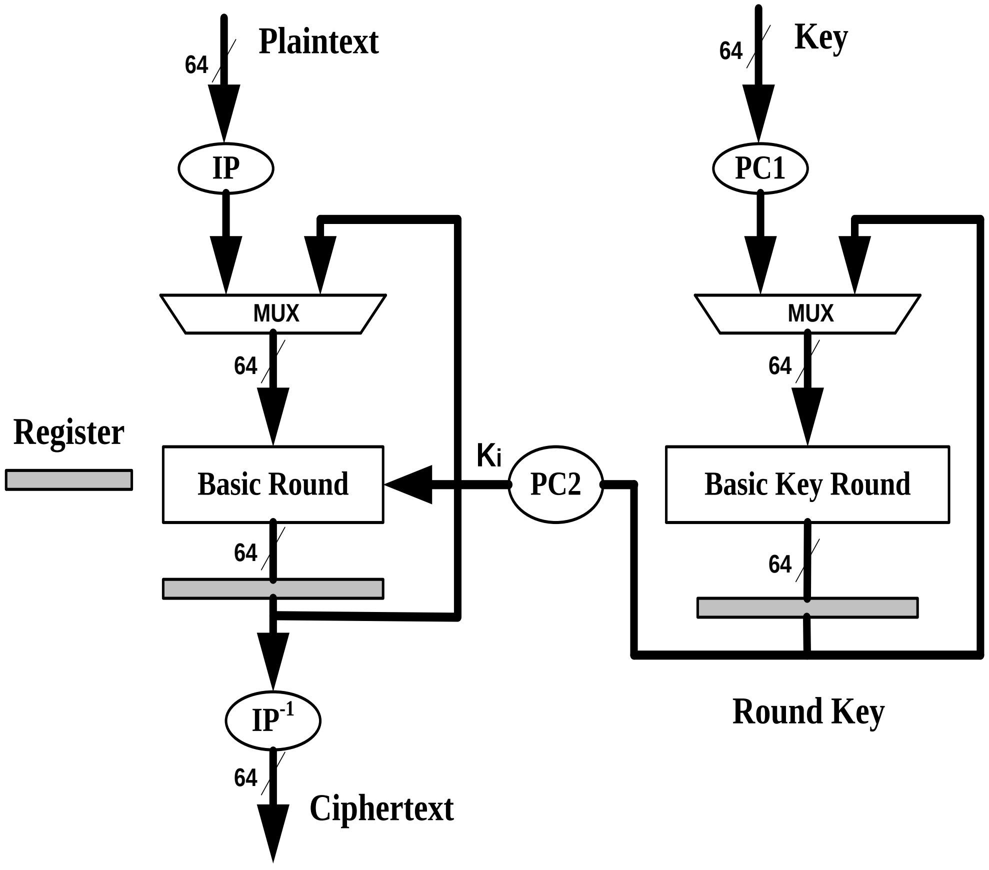
register

Triple-DES

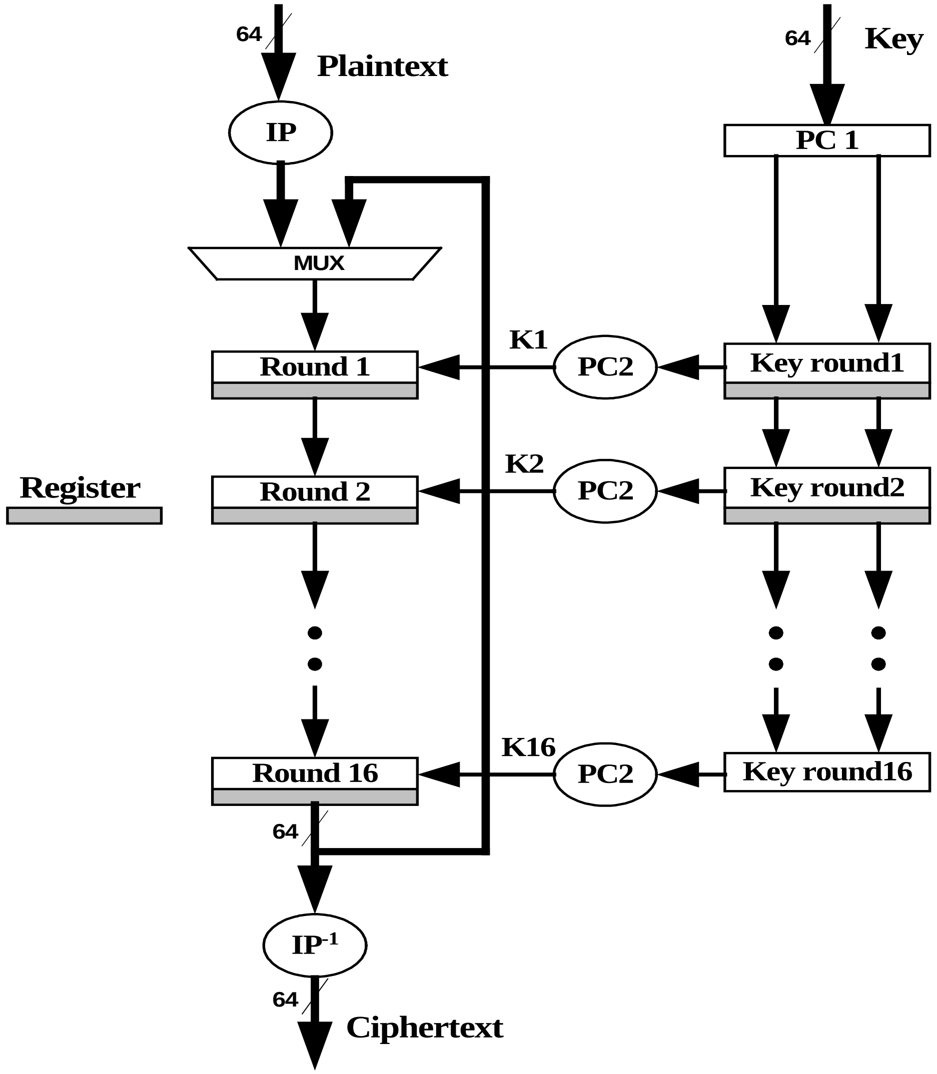


64
Plaintext
K1
DES Encryption
64
K2
DES Decryption
64
K3
DES Encryption
64
64
Ciphertext
Key
Key Sceduling
Ki
IP
16 Rounds
IP-1

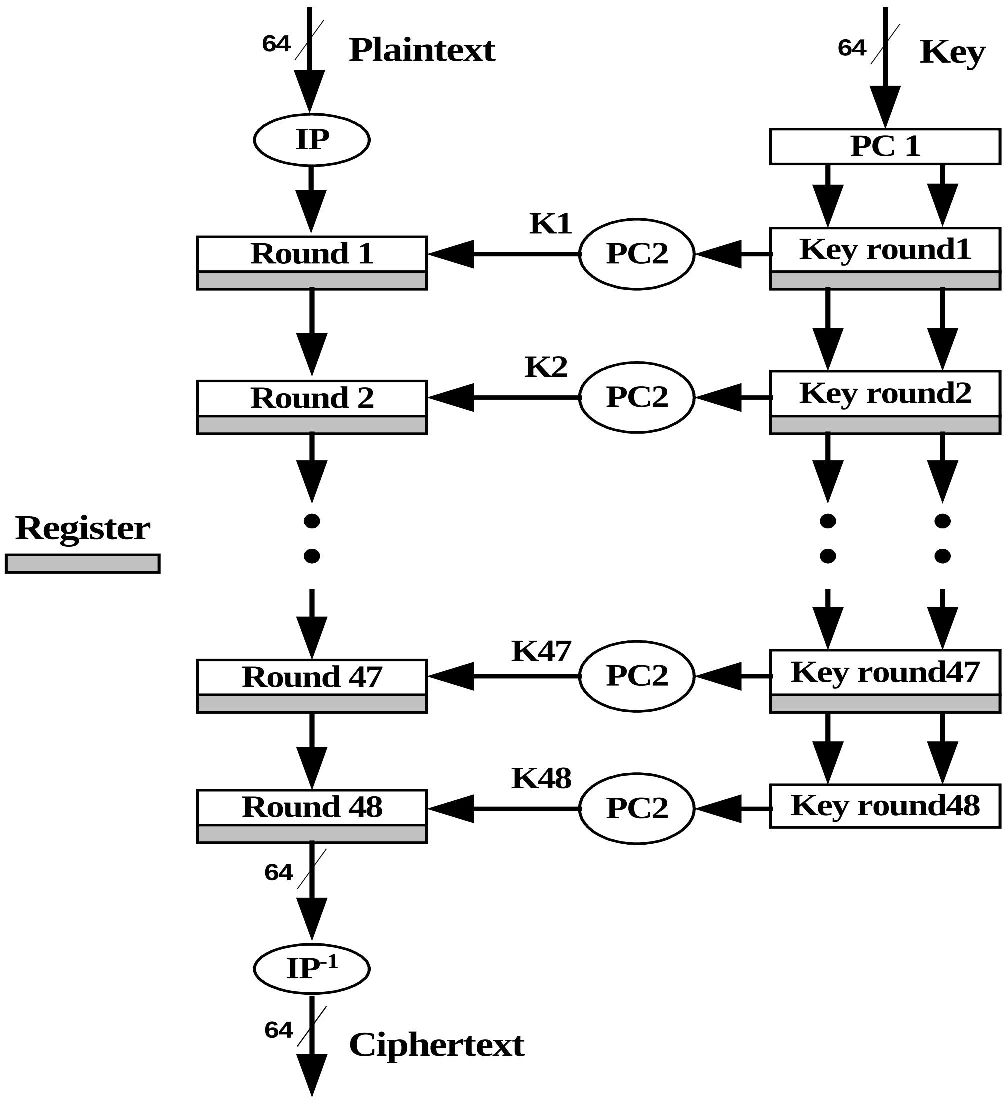
Triple-DES: Iterative architecture



Triple-DES: Partial loop unrolling



Triple-DES: Loop unrolling



KASUMI Block Cipher Application

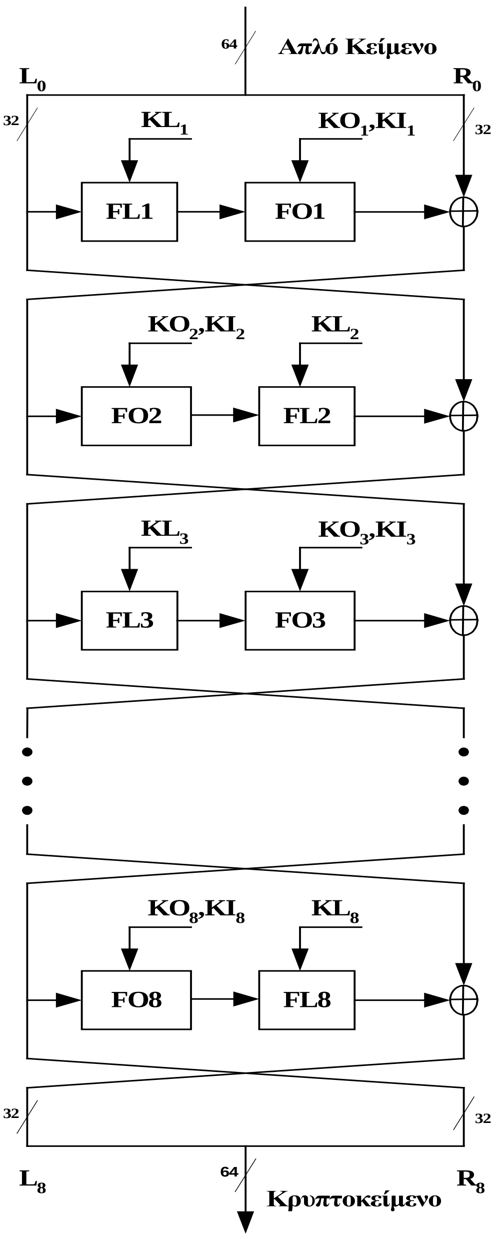
KASUMI block cipher is used:

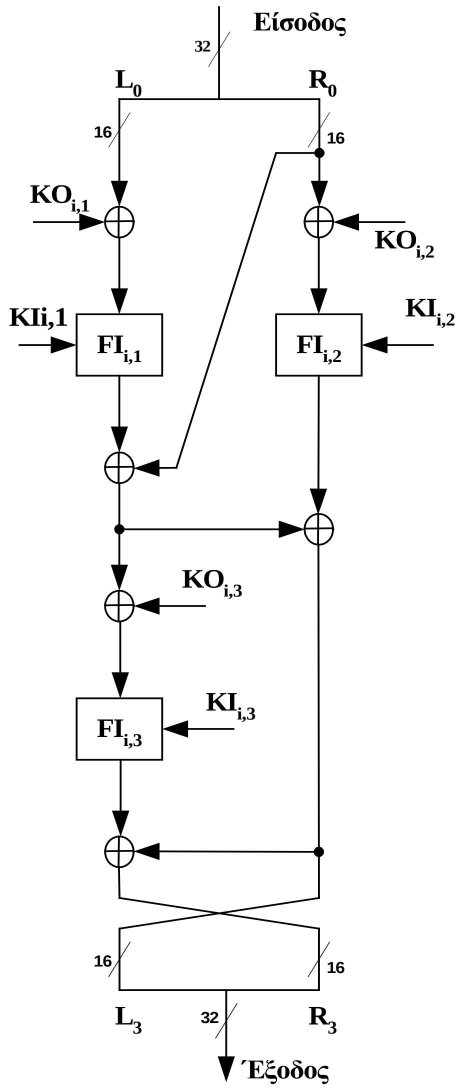
- In new GSM encryption algorithm A5/3
- In 3G and 4G, f8 and f9 algorithms
- In Transport Layer Securities (TLS)

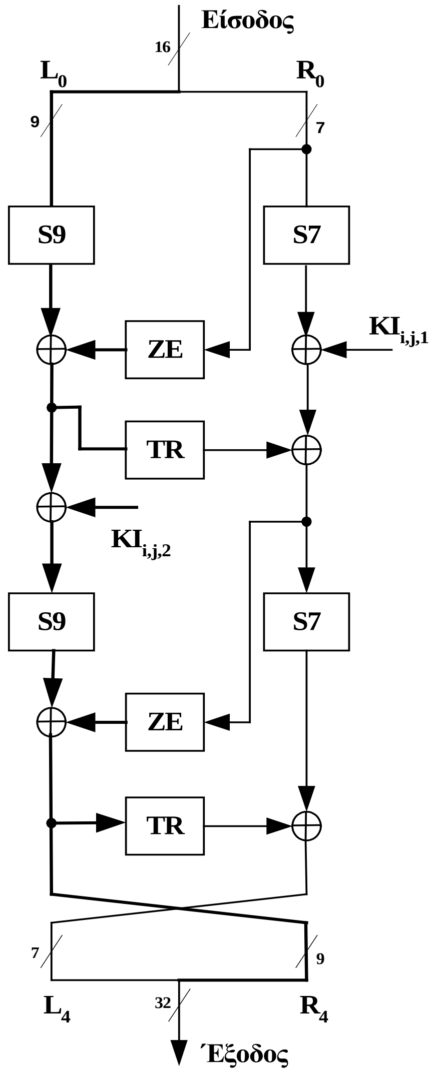
KASUMI Block Cipher…

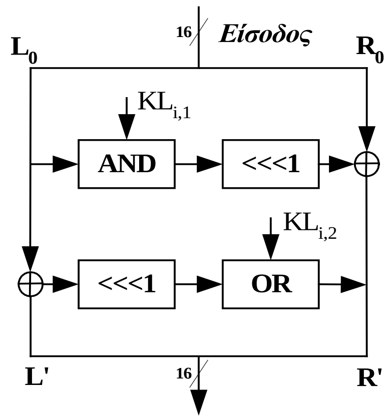
- Is the 64-bit block cipher
- Is a Feistel block cipher with 8 rounds
- The odd rounds have different structure than even rounds
- Uses 64-bit plaintext/ciphertext and 128-bit key

…KASUMI Block Cipher

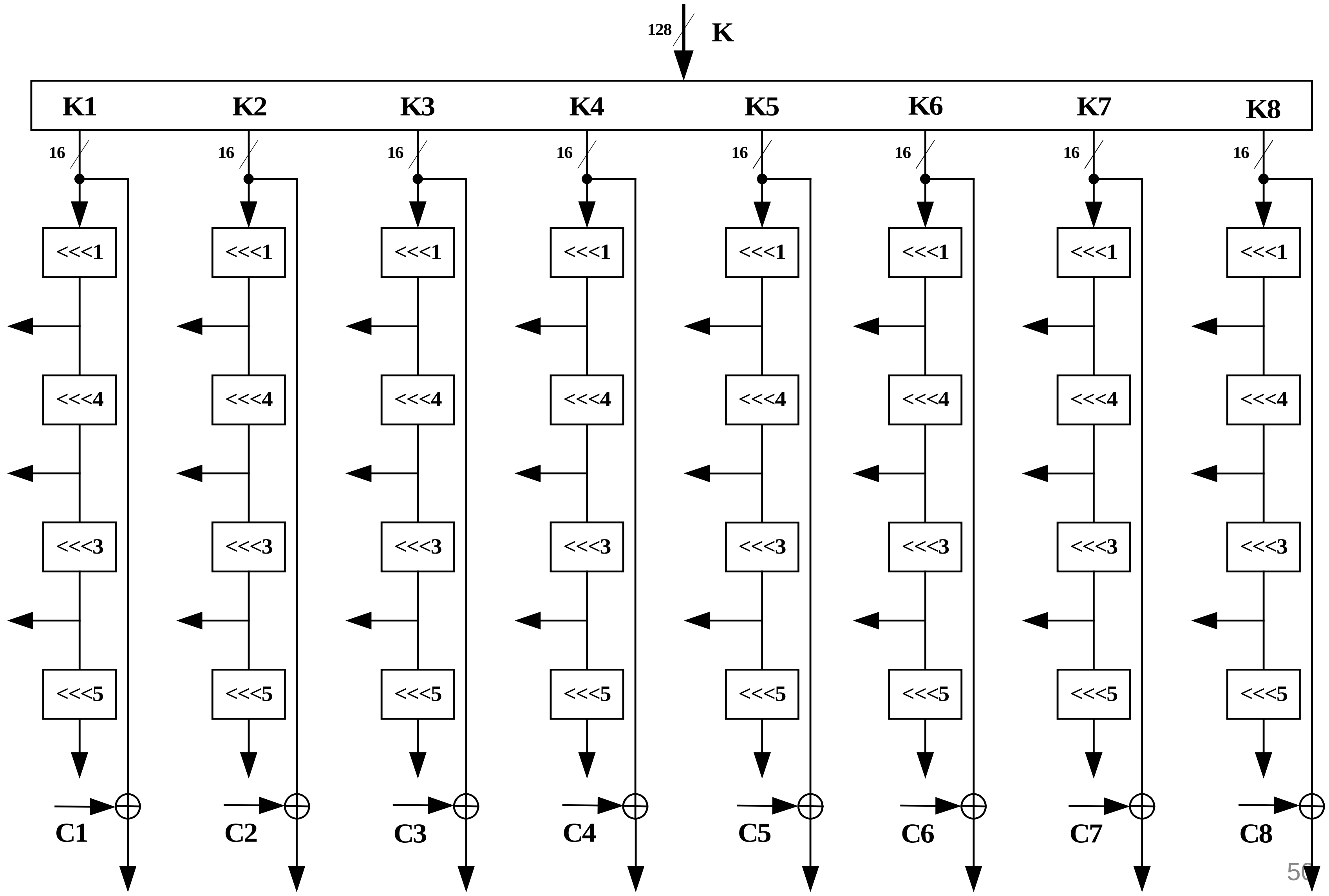




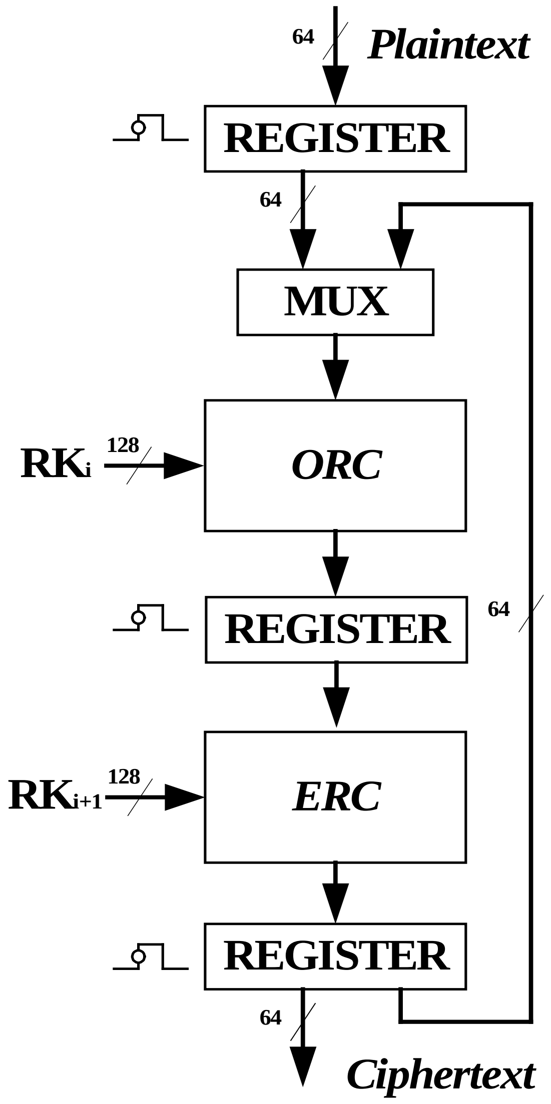




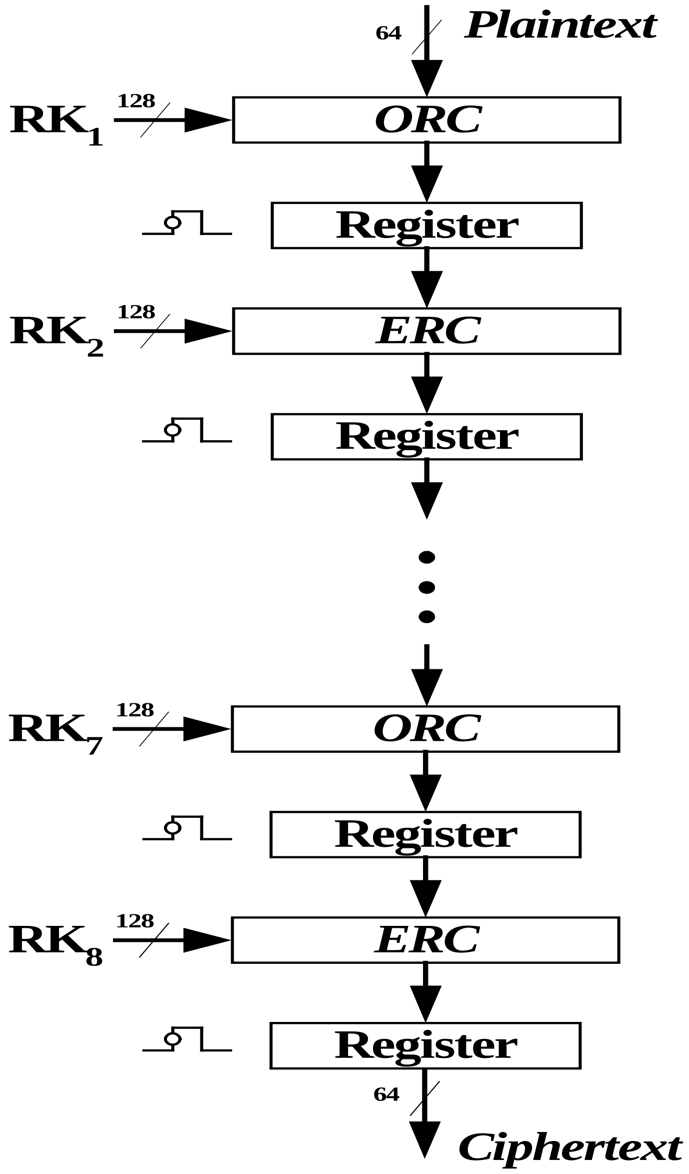
KASUMI Key Scheduling



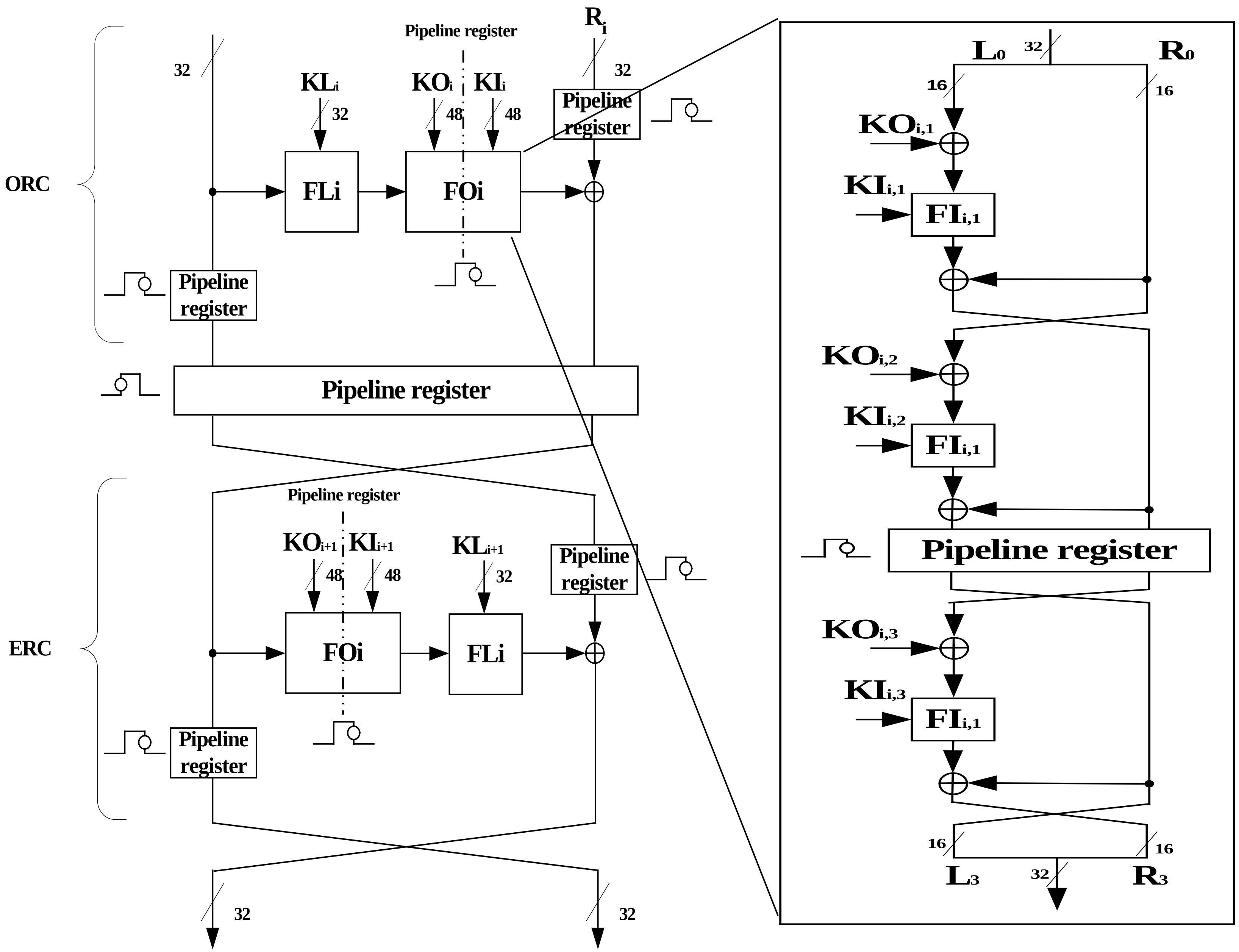
KASUMI: Partial loop unrolling



KASUMI: Loop unrolling



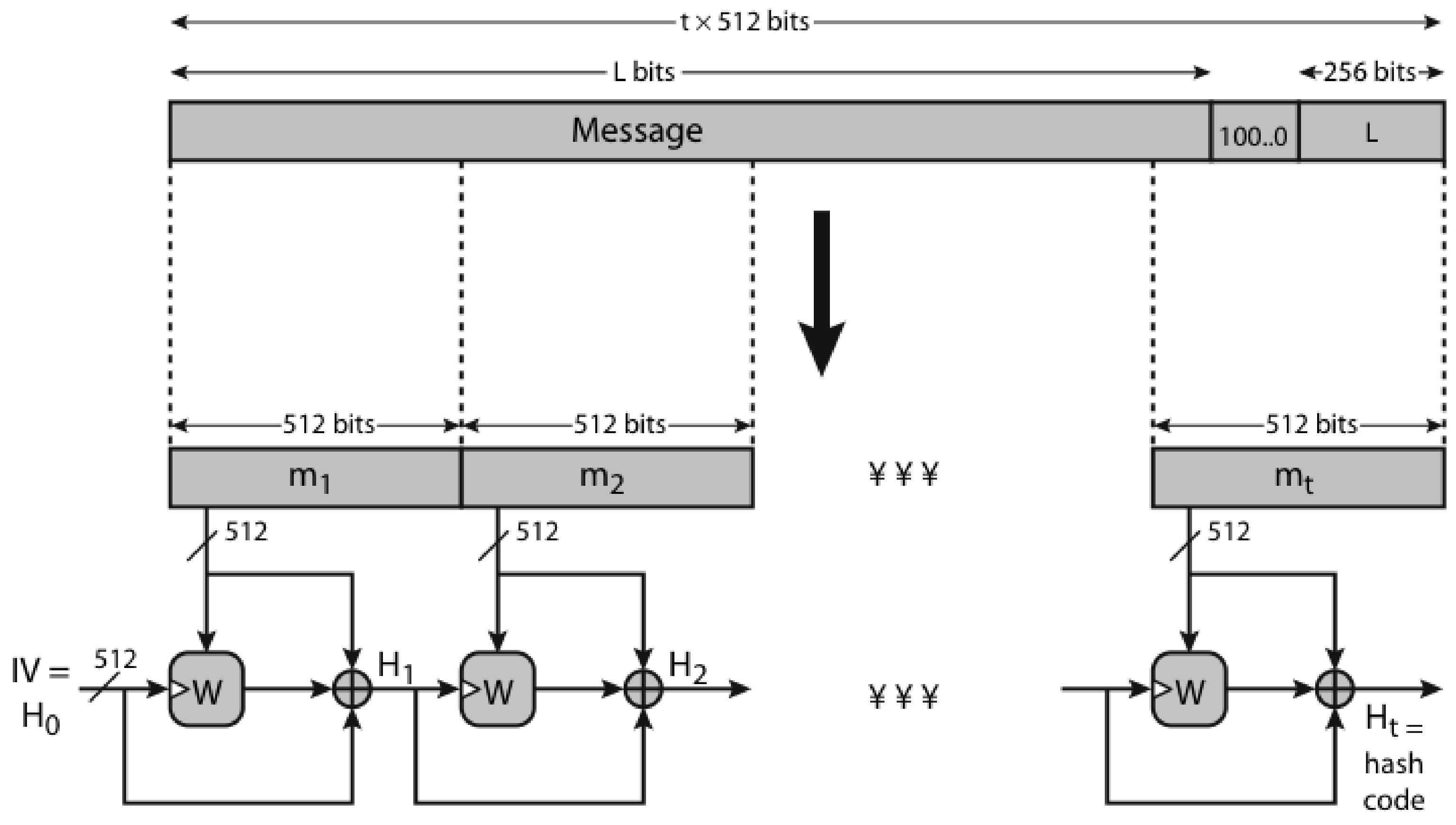
Round Implementation



Whirlpool Hash Function

- Endorsed by European NESSIE project
- Uses modified AES internals as compression function
- Addressing concerns on use of block ciphers seen previously

Whirlpool Overview

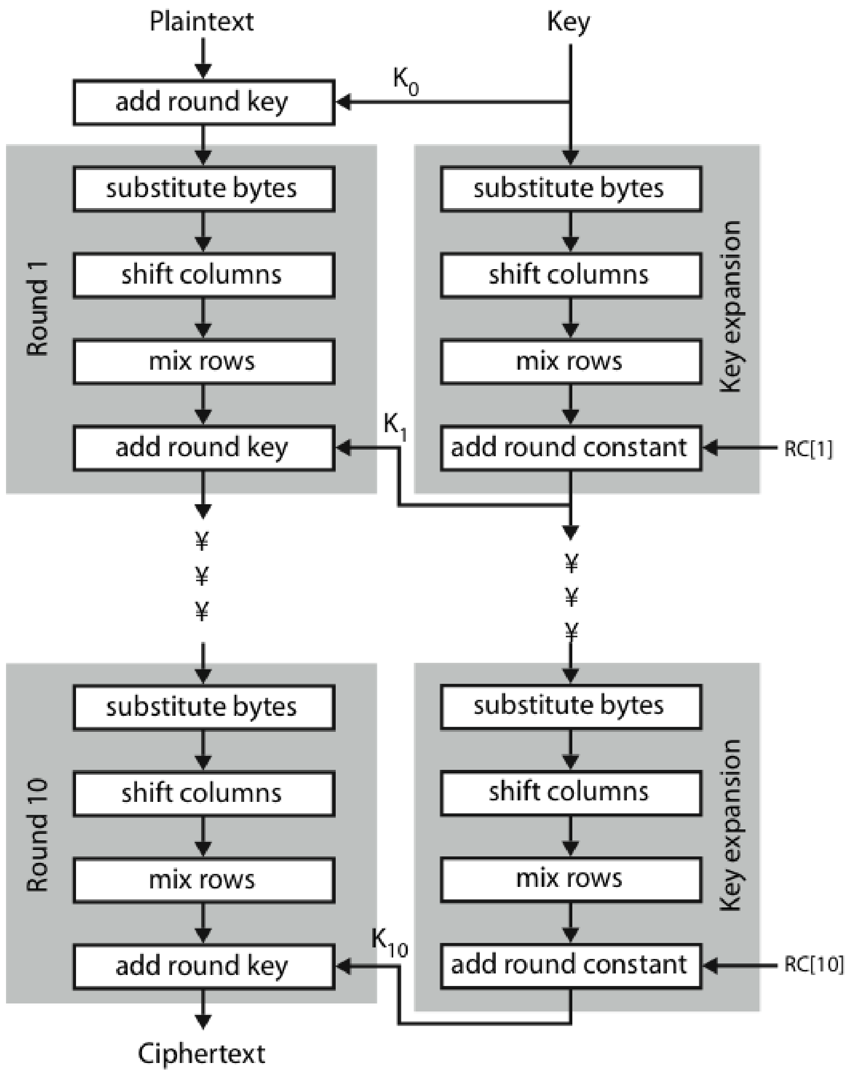


Note: triangular hatch marks key input

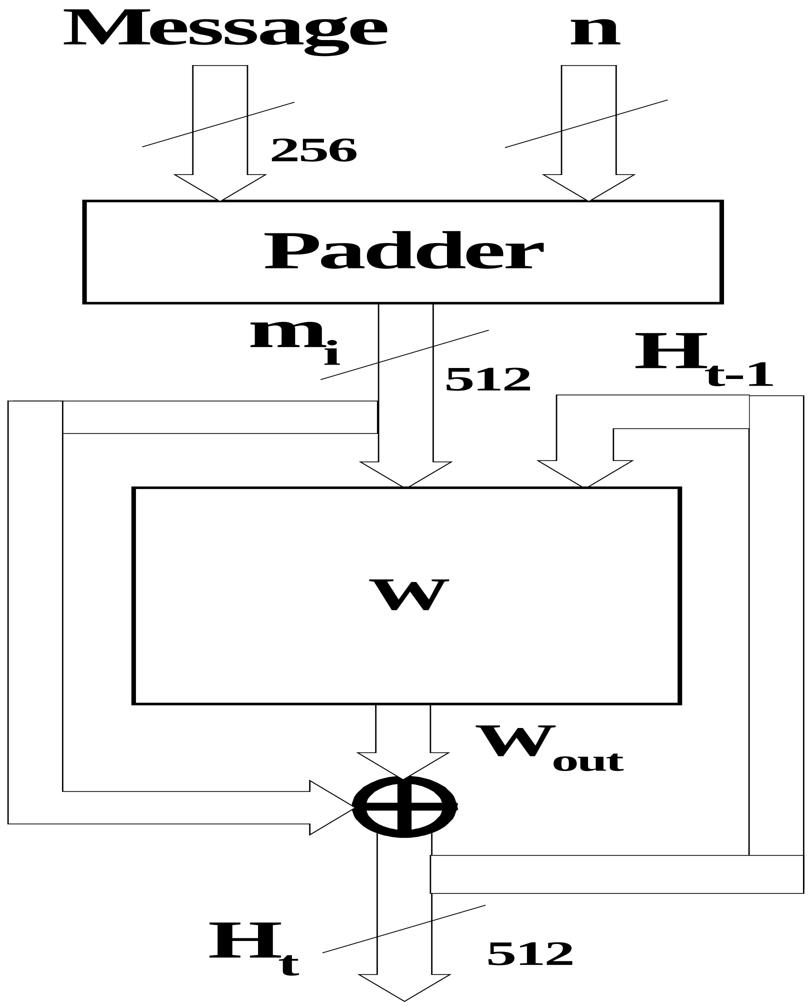
Whirlpool Block Cipher W

- Designed specifically for hash function use
- With security and efficiency of AES
- But with 512-bit block size and hence hash
- Similar structure & functions as AES but
  - input is mapped row wise
  - has 10 rounds
  - uses different S-box design & values

Whirlpool Block Cipher W

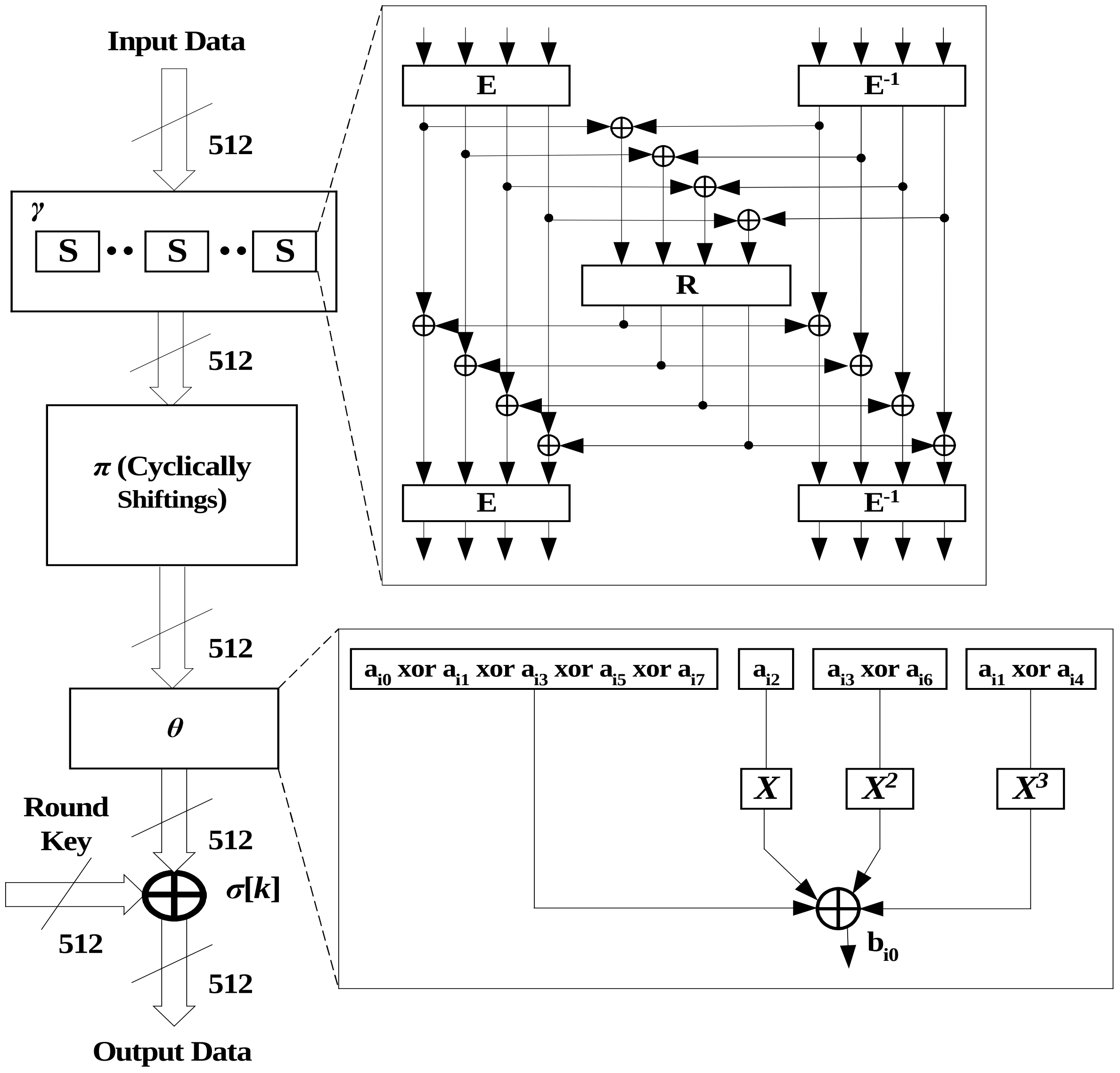


Whirlpool Architecture…

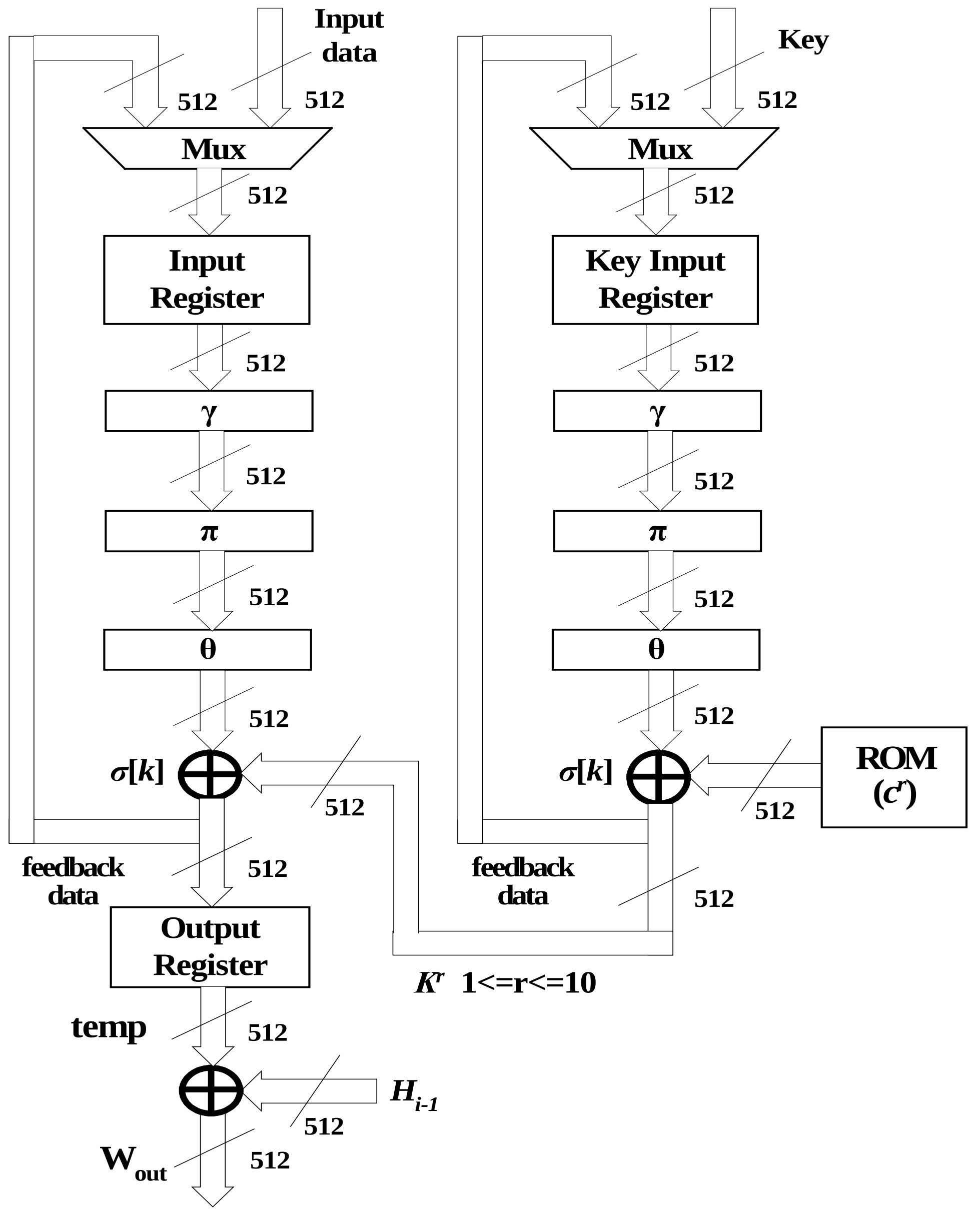


- The Padder pads the input data and converts them to (n+256)-bit padded message
- An interface with 256-bit input for Message is considered
- The n, specifies the total length of the message

…Whirlpool Architecture…



…Whirlpool Architecture



- This implementation has two similar parallel datapaths, the data randomizing and the key schedule
- The input block mi is set to the Input data simultaneously with the initial vector (IV) to the Key
- In a clock cycle, one execution round is executed and, simultaneously, the appropriate round key is calculated.
- Latency = 10 clock cycles

Questions??