CS-200
Computer Architecture

—

Part 4b. Instruction Level Parallelism
Basic Pipelining

Paolo Ienne
<paolo.ienne@epfl.ch>

Circuit Timing and Performance

- Most of the time so far we have mainly discussed circuits at a higher level of timing abstraction: what happens every **cycle**
  - Finite State Machines: `state <= next_state`
  - Functional units and memory elements perform one operation over a small number of cycles: e.g., a combinational ALU performs an addition per cycle

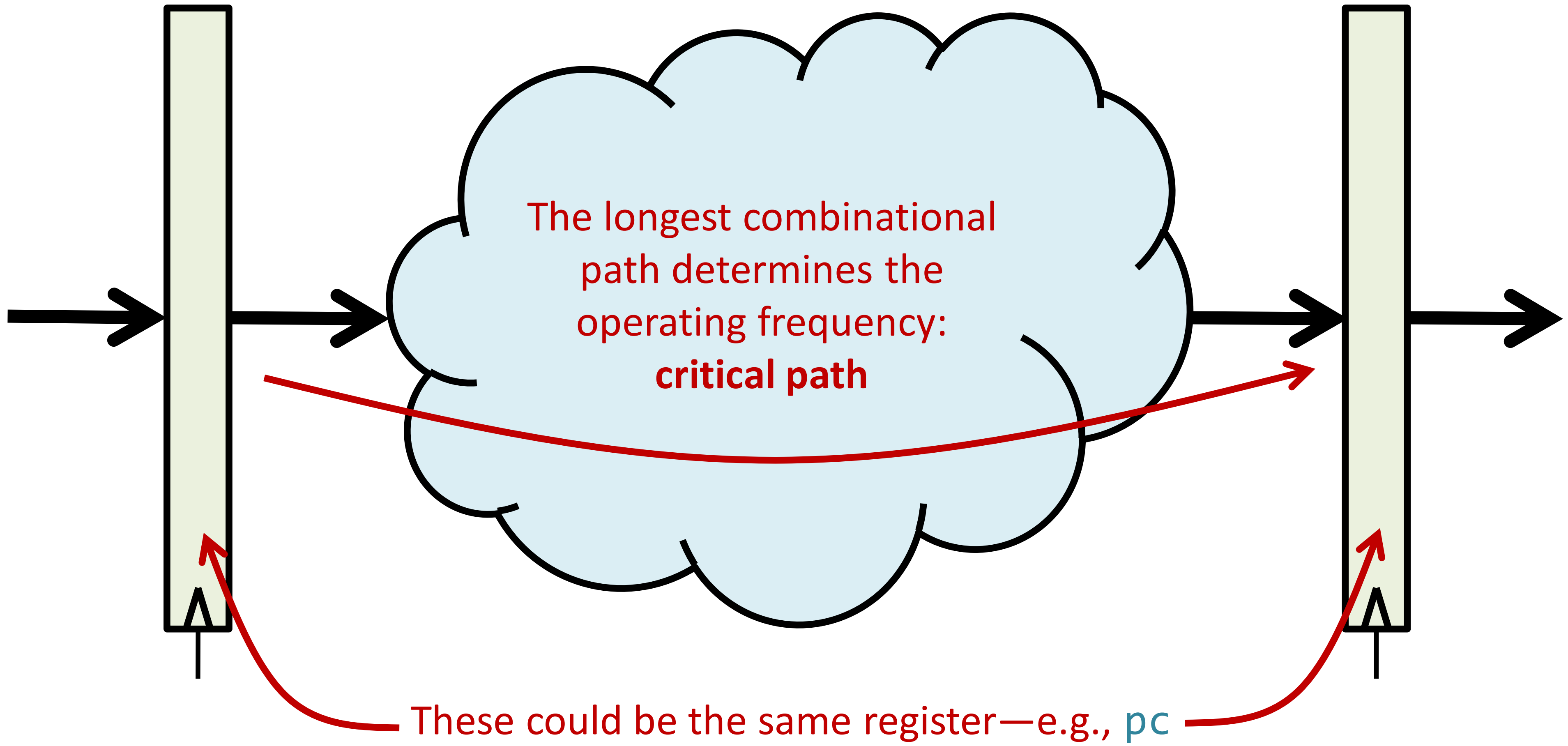
- To make faster circuits, we need to zoom-in briefly and understand more of **signal propagation** and timing limitations

Signal Propagation

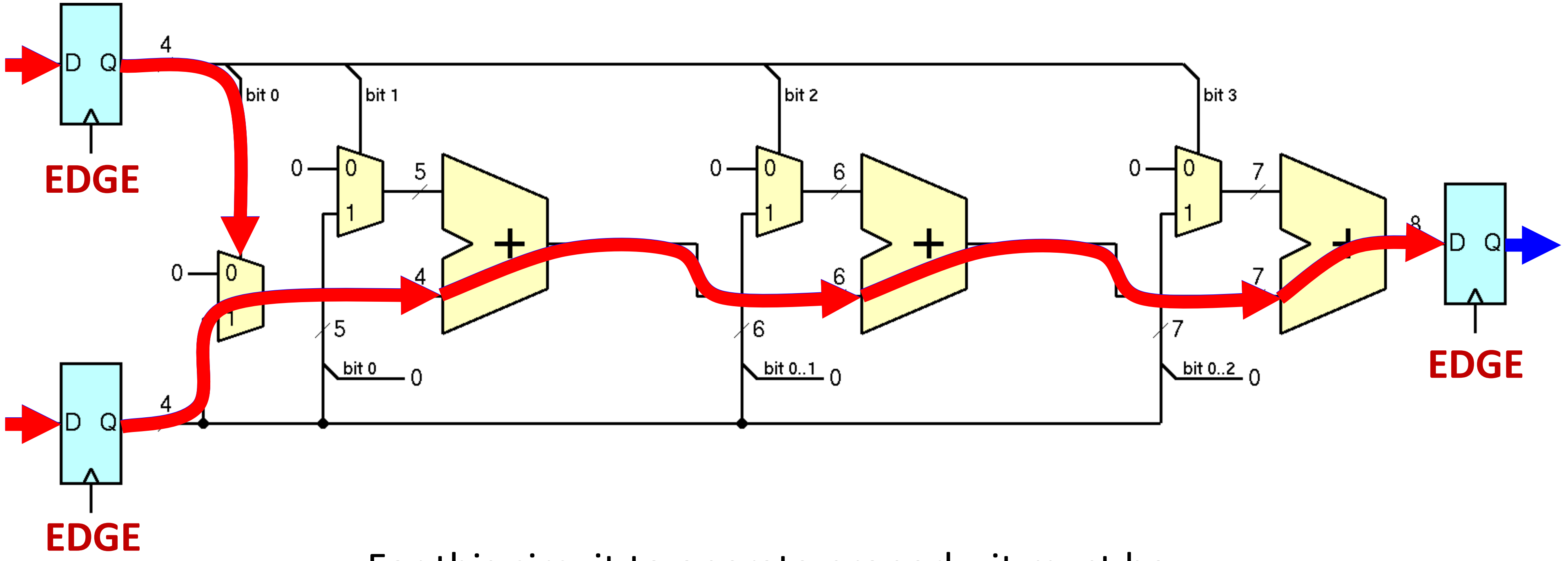
- The edge of the `clock` signal indicates:
  1. When **new data can be applied** to the combinational part of the circuit
  2. When old input data have crossed the combinational part of the circuit, the result is ready, and it **can be stored** at the output

- To operate circuits "as fast as we can", we apply a `clock` signal whose **period is equal to the critical path delay** (that is, the longest delay) of the circuit

Propagation Time


The longest combinational path determines the operating frequency: **critical path**
These could be the same register—e.g., pc

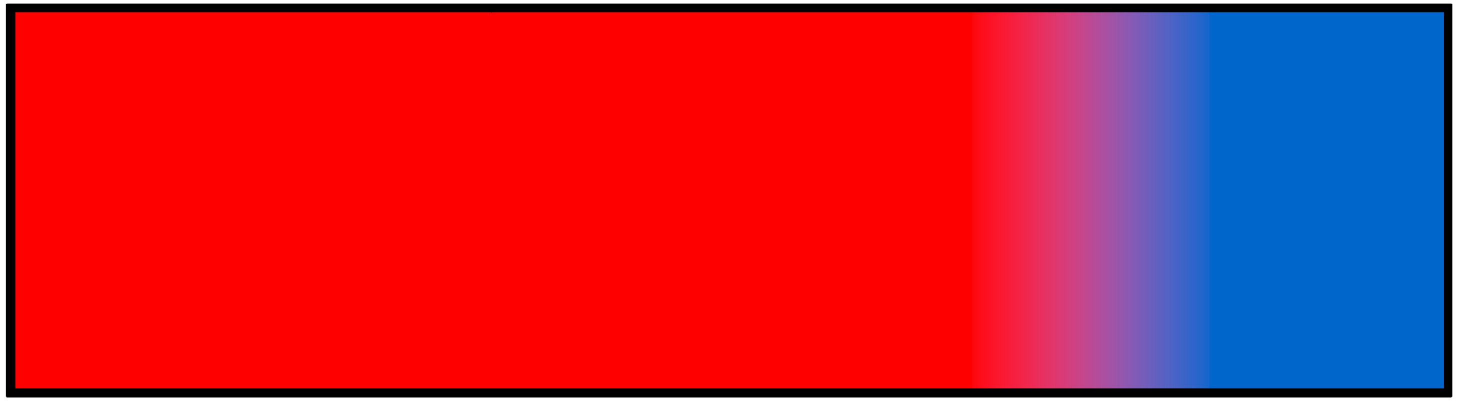
Propagation Time



For this circuit to operate properly, it must be

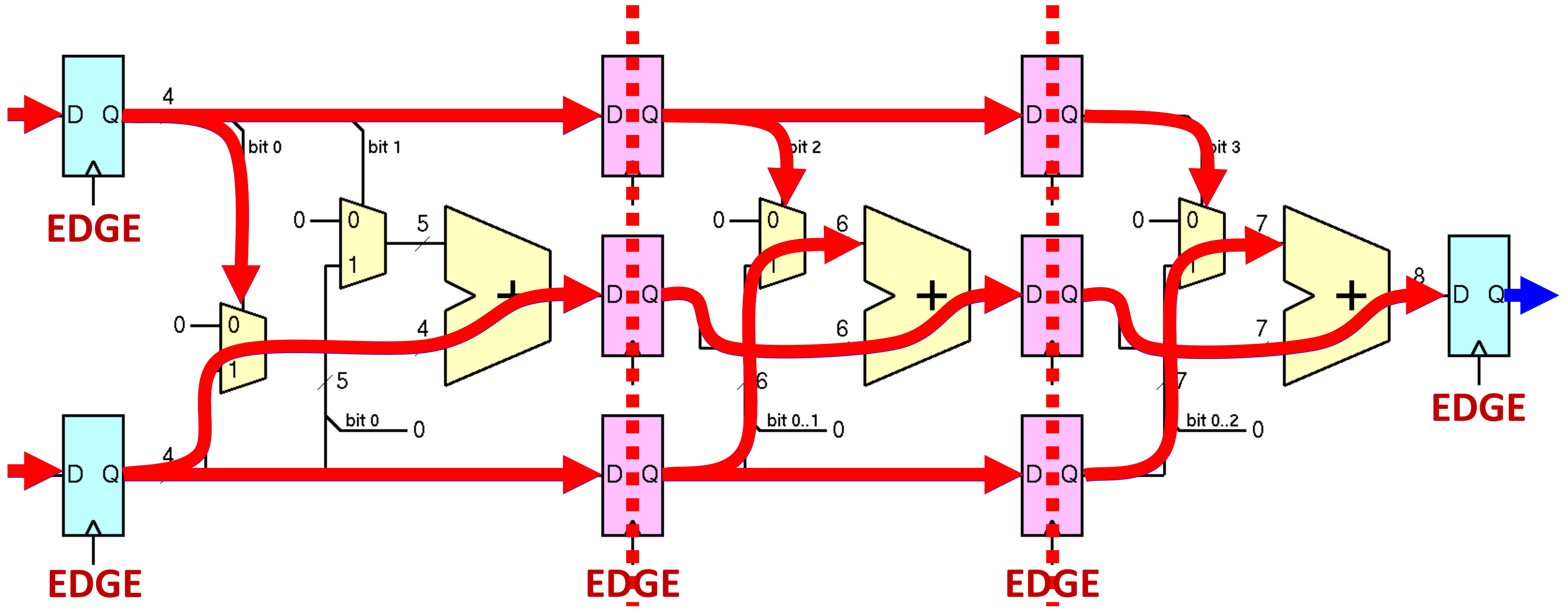
$$T_{CLK,\,comb} \geq T_{critical\ path}$$

Combinational Circuit



Only a small number of transistors are active
(= changing state) at a given time

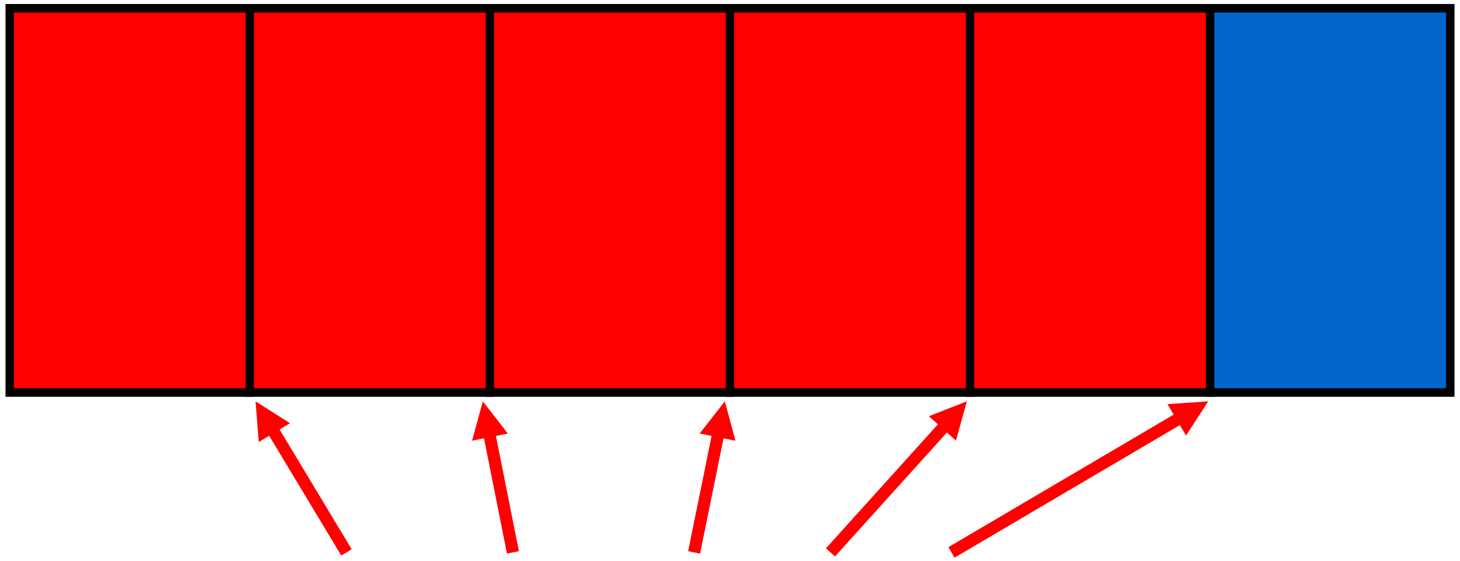
Adding Intermediate Registers



For this circuit to operate properly, it must be

$$T_{CLK,\ pipe} \geq T_{new\,critical\ path} \cong T_{critical\ path}/3$$

Combinatorial Circuit with Interspersed Registers



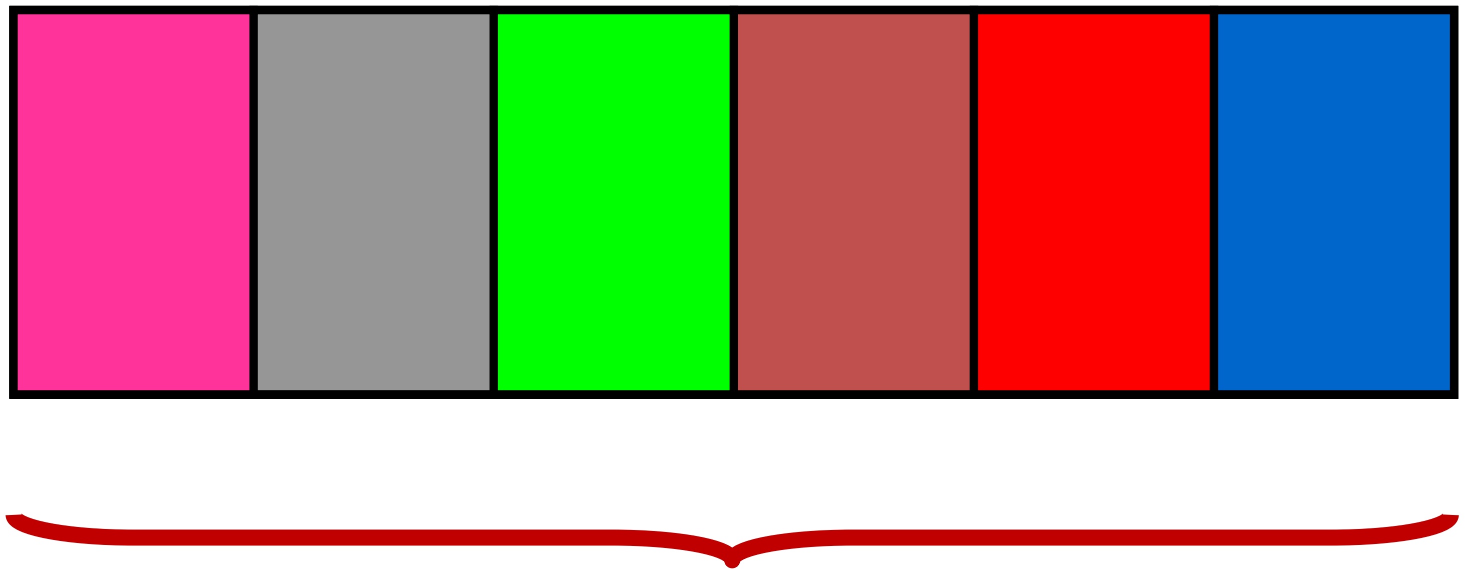
Registers rigidly delimit areas of activity

What Has Changed?

- **Functionally**: absolutely **no change!**
  - We just have a fine grain control of the propagation through the intermediate registers
- Clock can run faster
  - If we have introduced N stages of intermediate registers evenly, roughly our critical path is N times smaller
  - **Clock can run N times faster!**
- Great?!
  - Not really: **we now need N cycles** to get the result…

Multiple Operations in a Pipeline

Inactive areas for one operation
can be used for **other operations!**



Pipeline

Any Advantage Now?

- Time to compute a single operation is **roughly the same** as in the original circuit
- New results are available:
  - In the original circuit, **every original period T**
  - In the circuit with the registers used for a single calculation, **every N cycles of period T/N** → every T
  - In the circuit with the registers where we inject a new computation every cycle, we get

a new result every T/N!

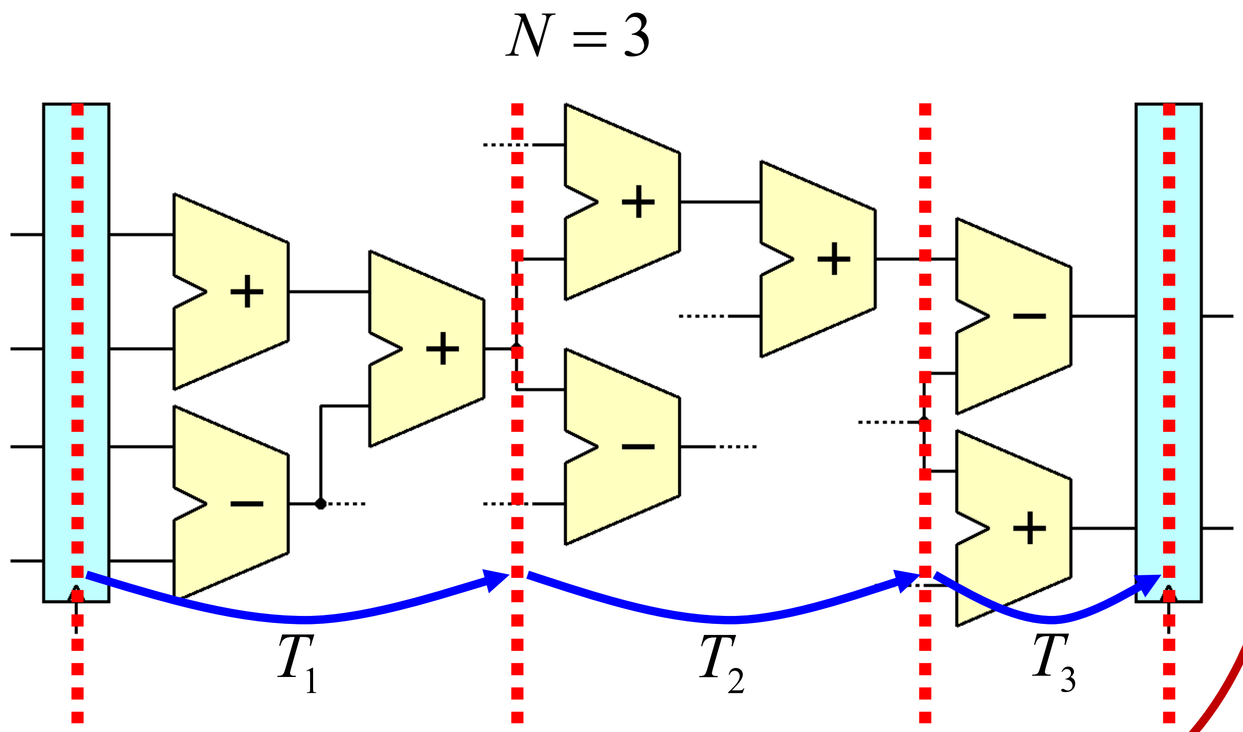
- We can generate arbitrarily more results (N large)?!…

Latency and Throughput

- **Latency**
  - Time between a computation begins and result is available
    - Original circuit: **T**
    - Pipelined circuit: $T/N \times N =$ **T**
- **Throughput**
  - Number of results available in the unit time
    - Original circuit: $1/T =$ **f**
    - Pipelined circuit: $1/(T/N) = N/T =$ **N × f**

- Above values only in theory…

Practical Pipelining



But flip-flops introduce a delay of their own…

But one might not be able to split **evenly** the circuit…

Ideally,

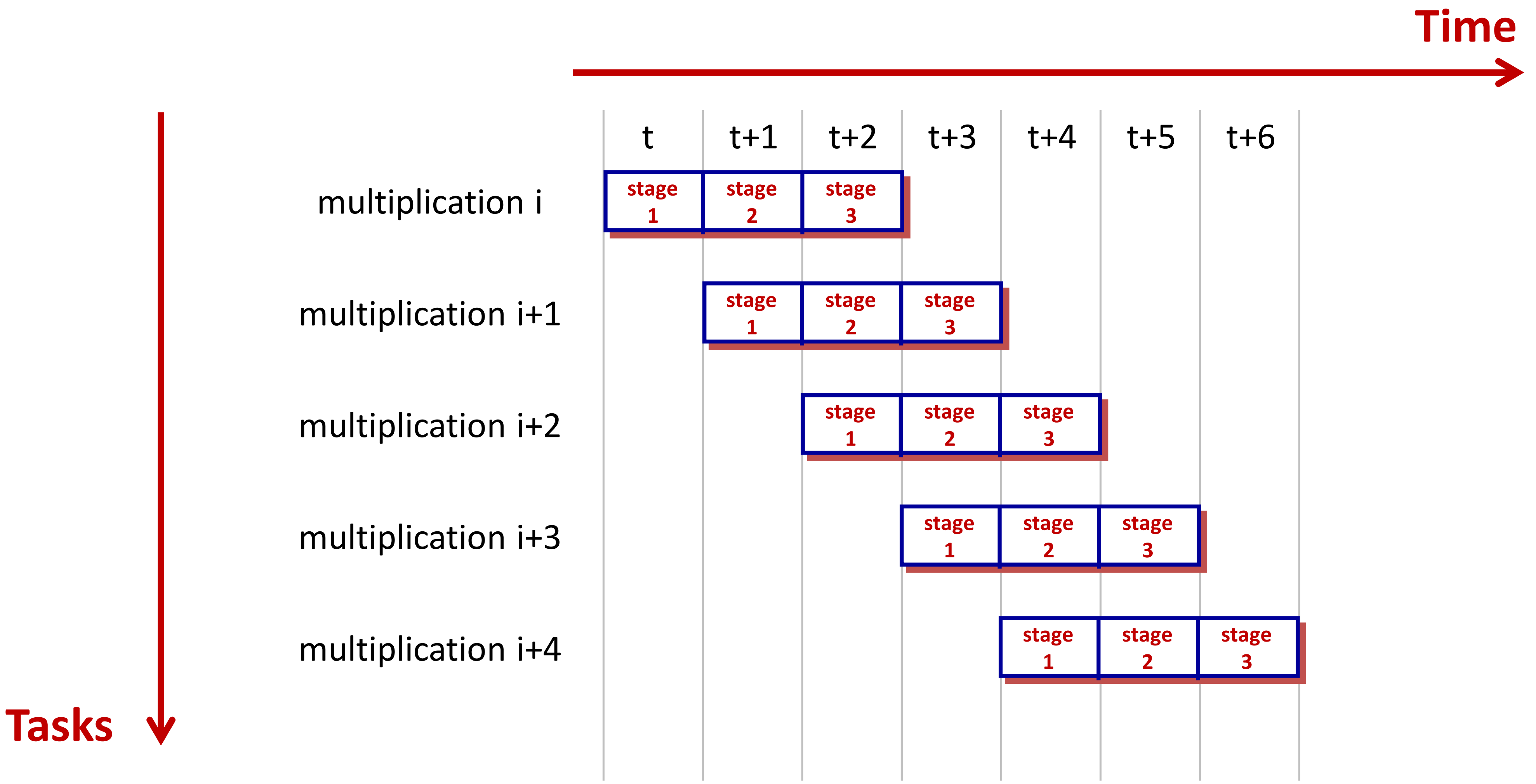
$$T_i \cong T_{CLK,comb} / N$$

and

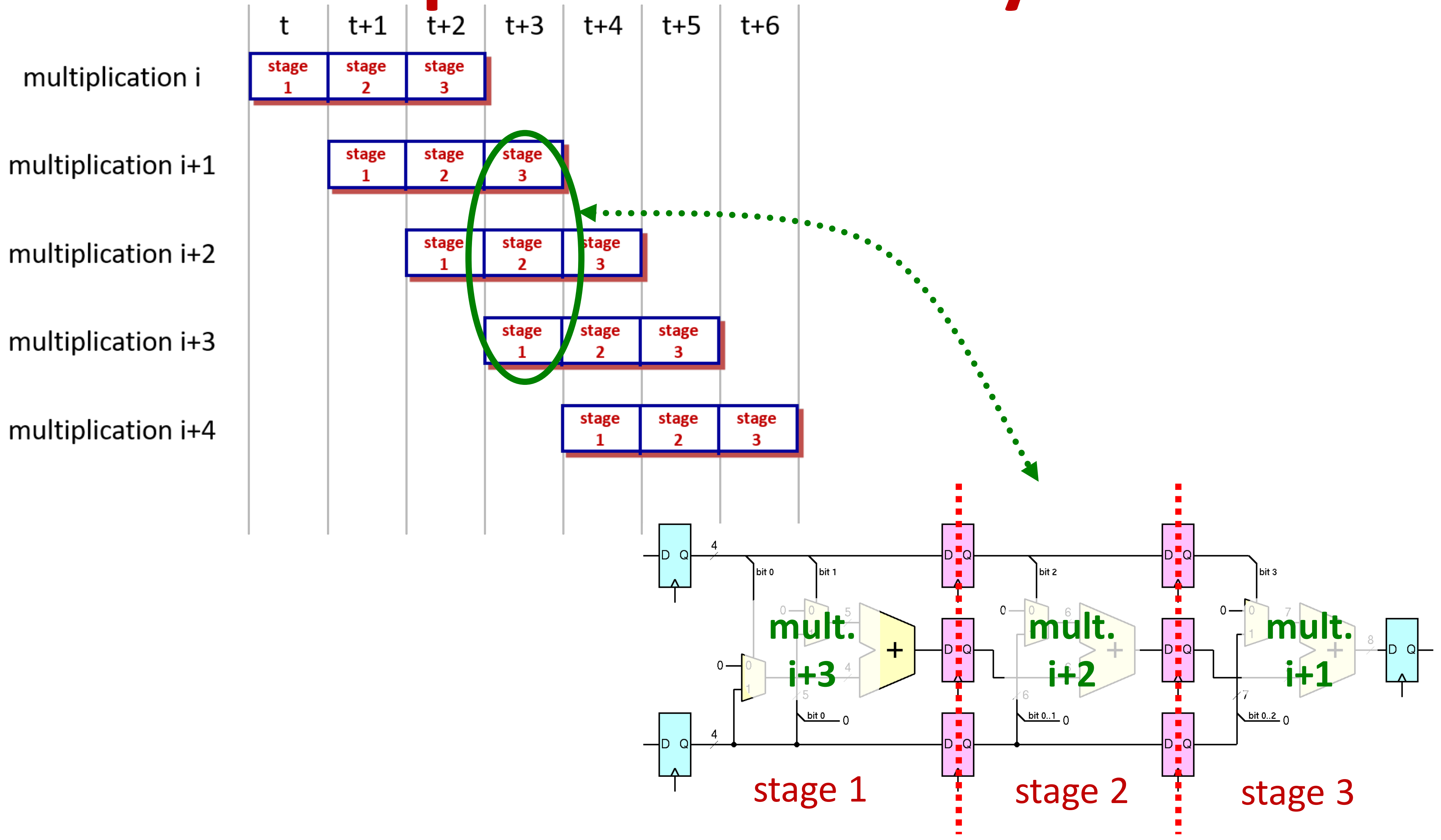
$$T_i \cong T_{CLK,pipe}$$

- **Latency** $\lambda_{pipe} = N \cdot \max_{i=0..N-1}(T_i + T_{FF}) = N \cdot T_{CLK,pipe} = N / f_{pipe} > \lambda_{orig}$
- **Throughput** $\phi_{pipe} = 1/ \max_{i=0..N-1}(T_i + T_{FF}) = f_{pipe}$

Useful Representation of the Pipeline Activity



Pipeline Schedule Describes What Is in the Pipeline at Each Cycle



Summary

- **Pipelining** consists in splitting a task in smaller “subtasks”, and in performing **in parallel** each “subtask” on a different piece of data

- **Pipelining** is an extremely general technique to **increase the throughput** of a system (circuit, processor, computer,…)

- **Pipelining does not improve latency** (actually worsens it!)

- Therefore, pipelining is only effective when one has to repeat a job on many pieces of data (signal processing, communication packets,… and **processor instructions**!)

References

- Patterson & Hennessy, COD – RISC-V Edition
  - Beginning of **Section 4.6**