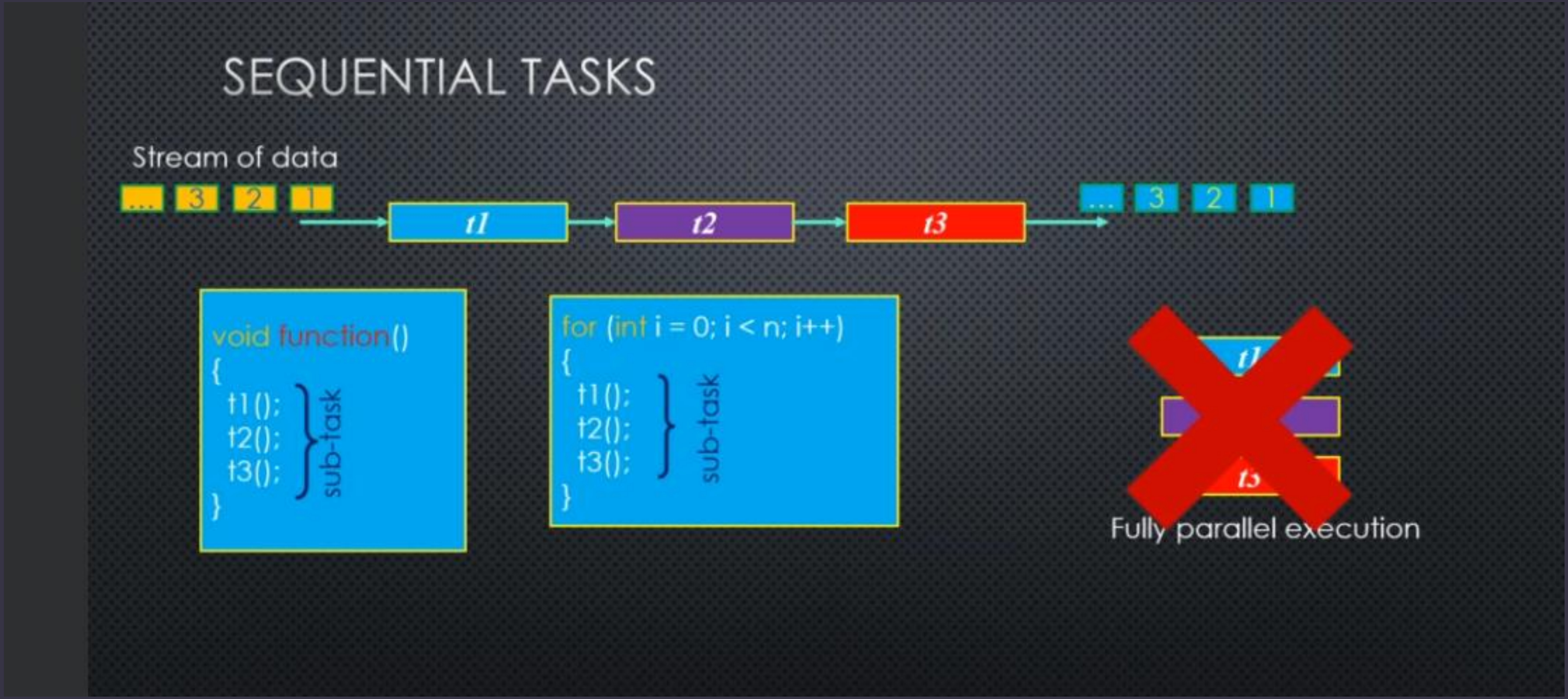
PIPELINING

Lecture-10



SEQUENTIAL TASKS

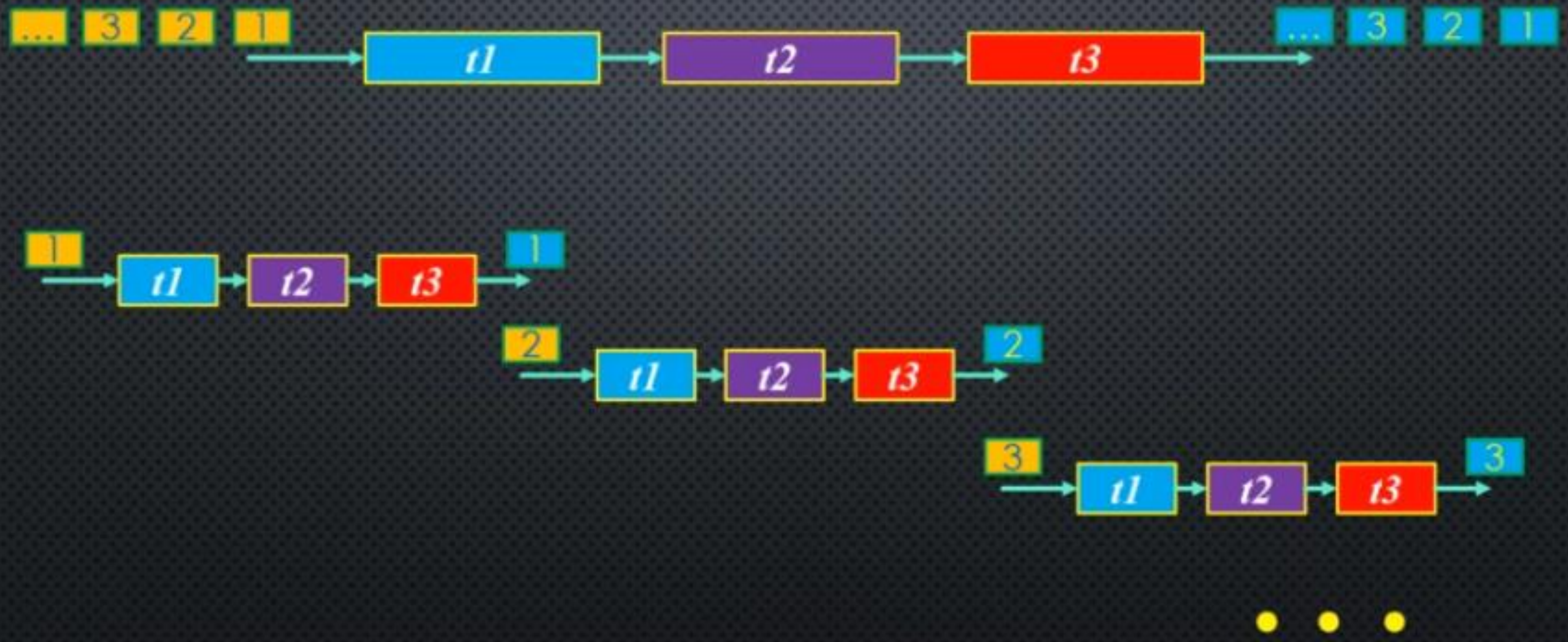
```
void function()
{
  t1();
  t2();   } sub-task
  t3();
}
```

```
for (int i = 0; i < n; i++)
{
  t1();
  t2();   } sub-task
  t3();
}
```

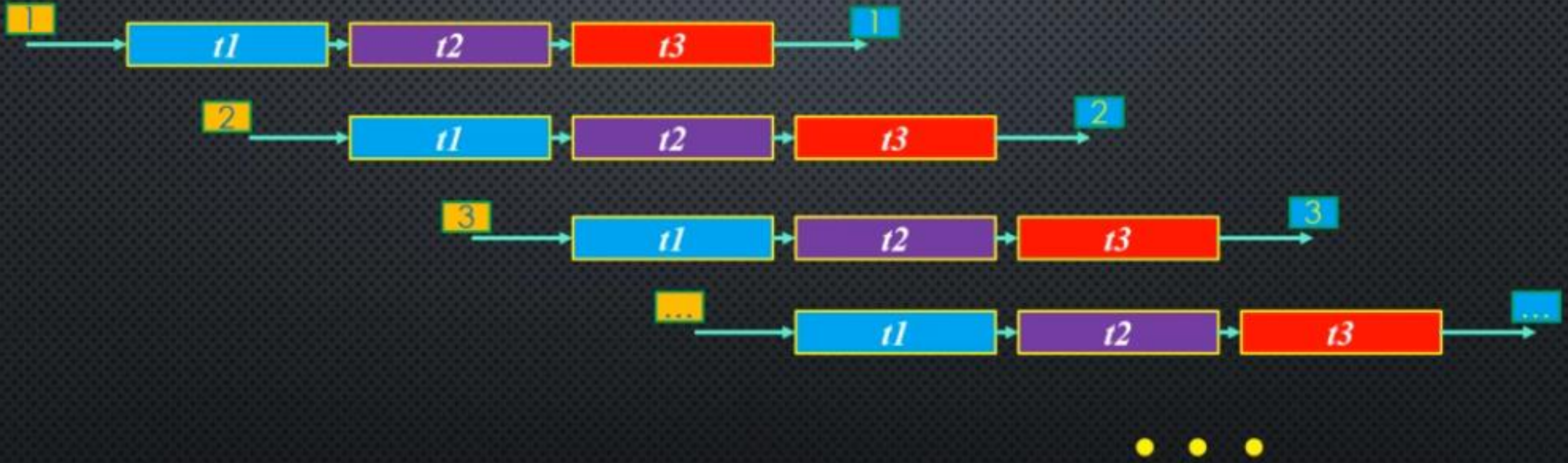
Fully parallel execution

SEQUENTIAL EXECUTION

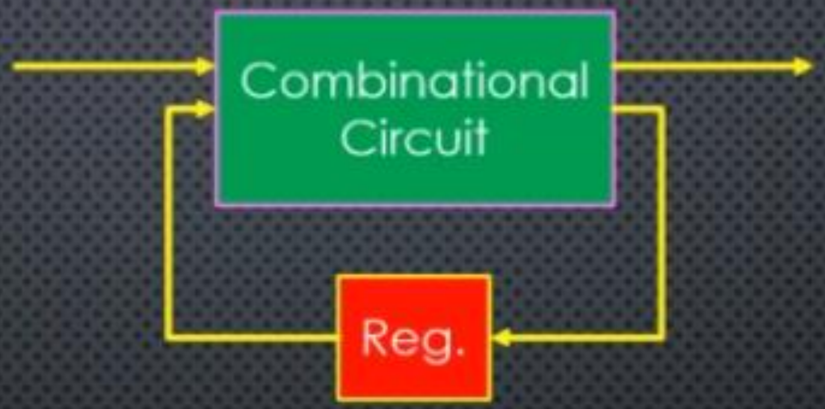
Stream of data

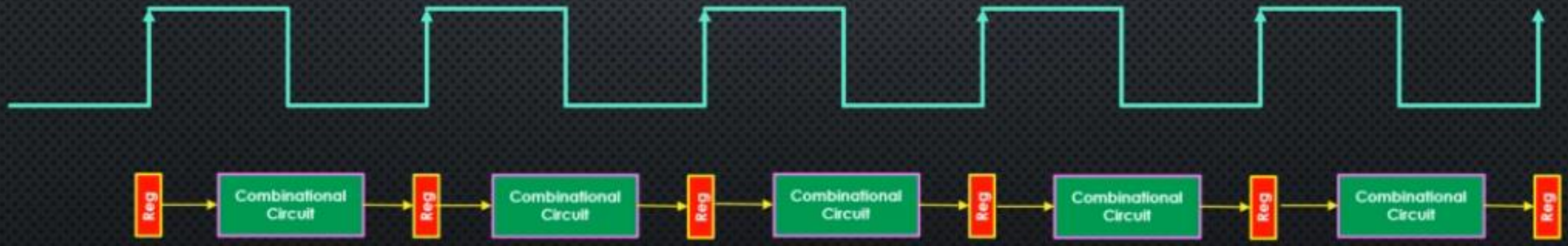


PIPELINE EXECUTION

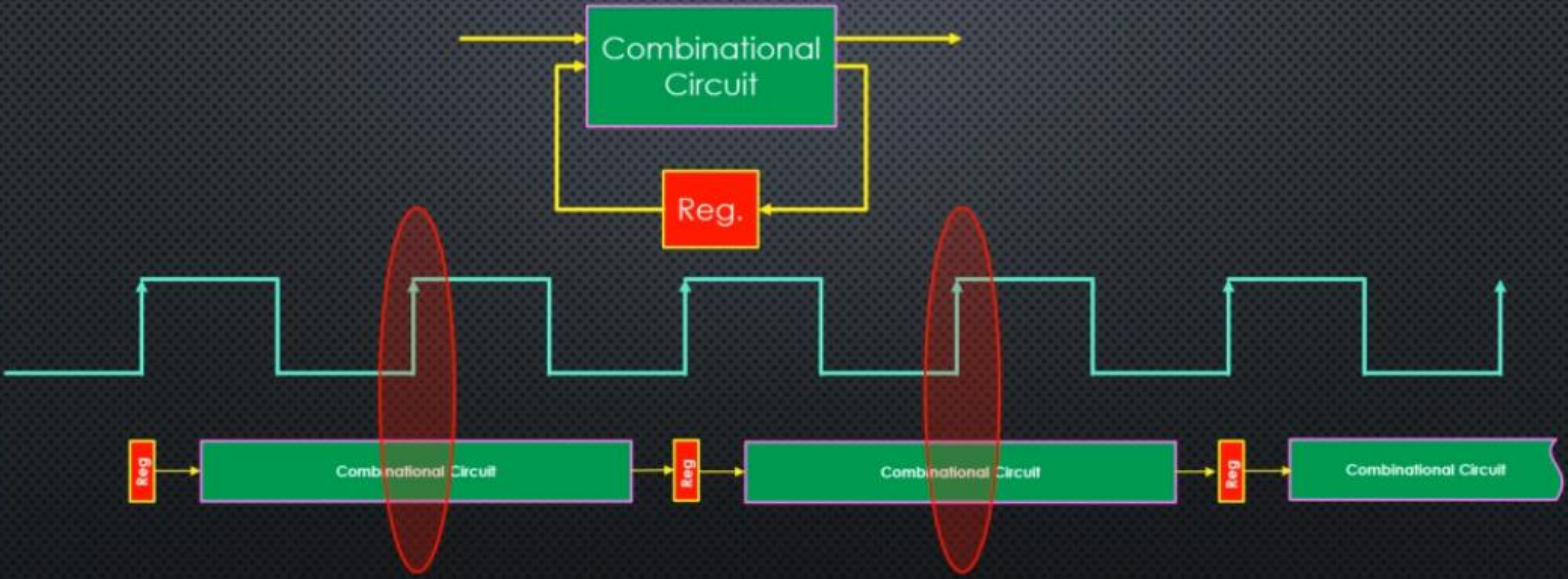


SINGLE CYCLE DESIGN TECHNIQUE

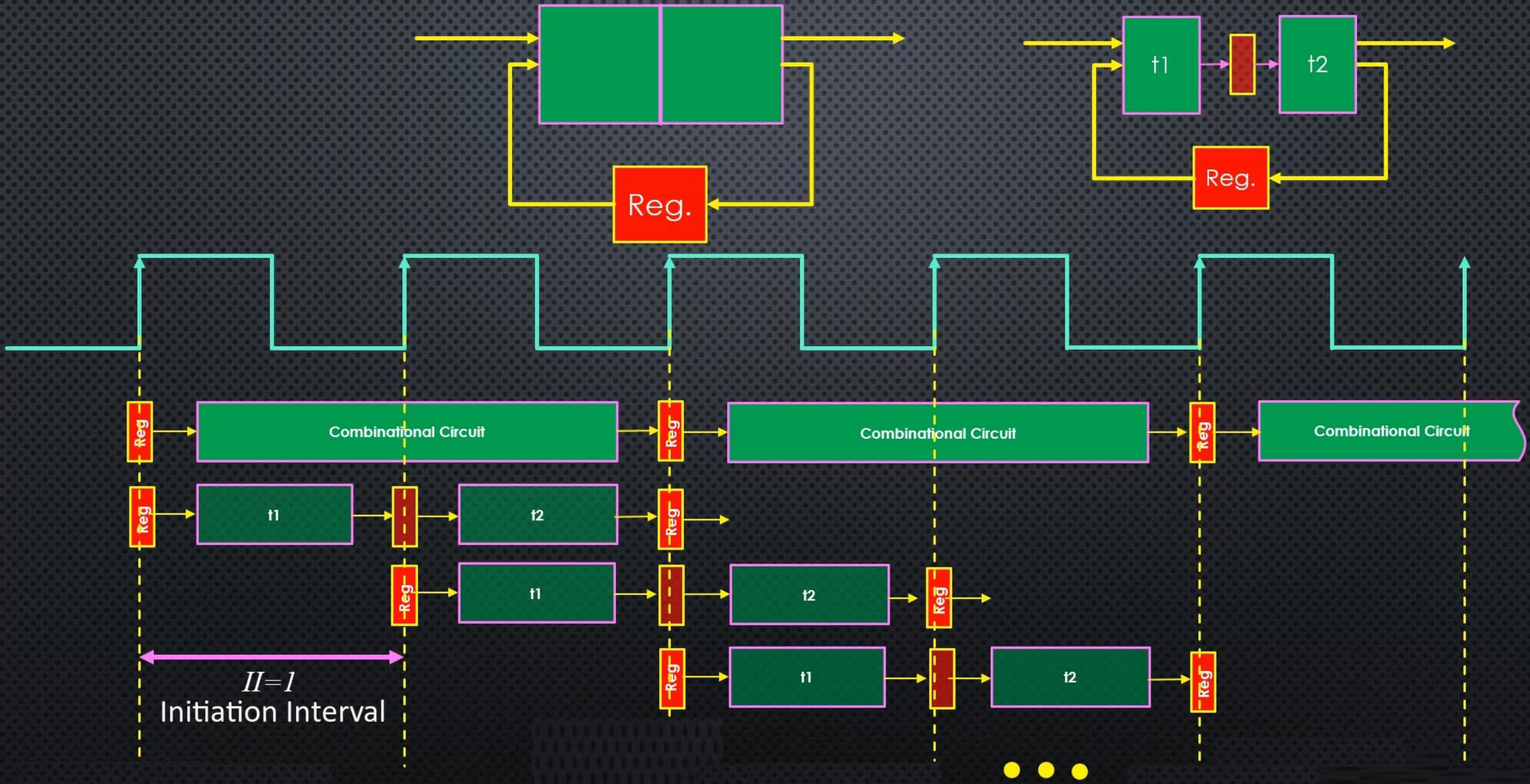




PROBLEM WITH MULTI-CYCLE DESIGNS



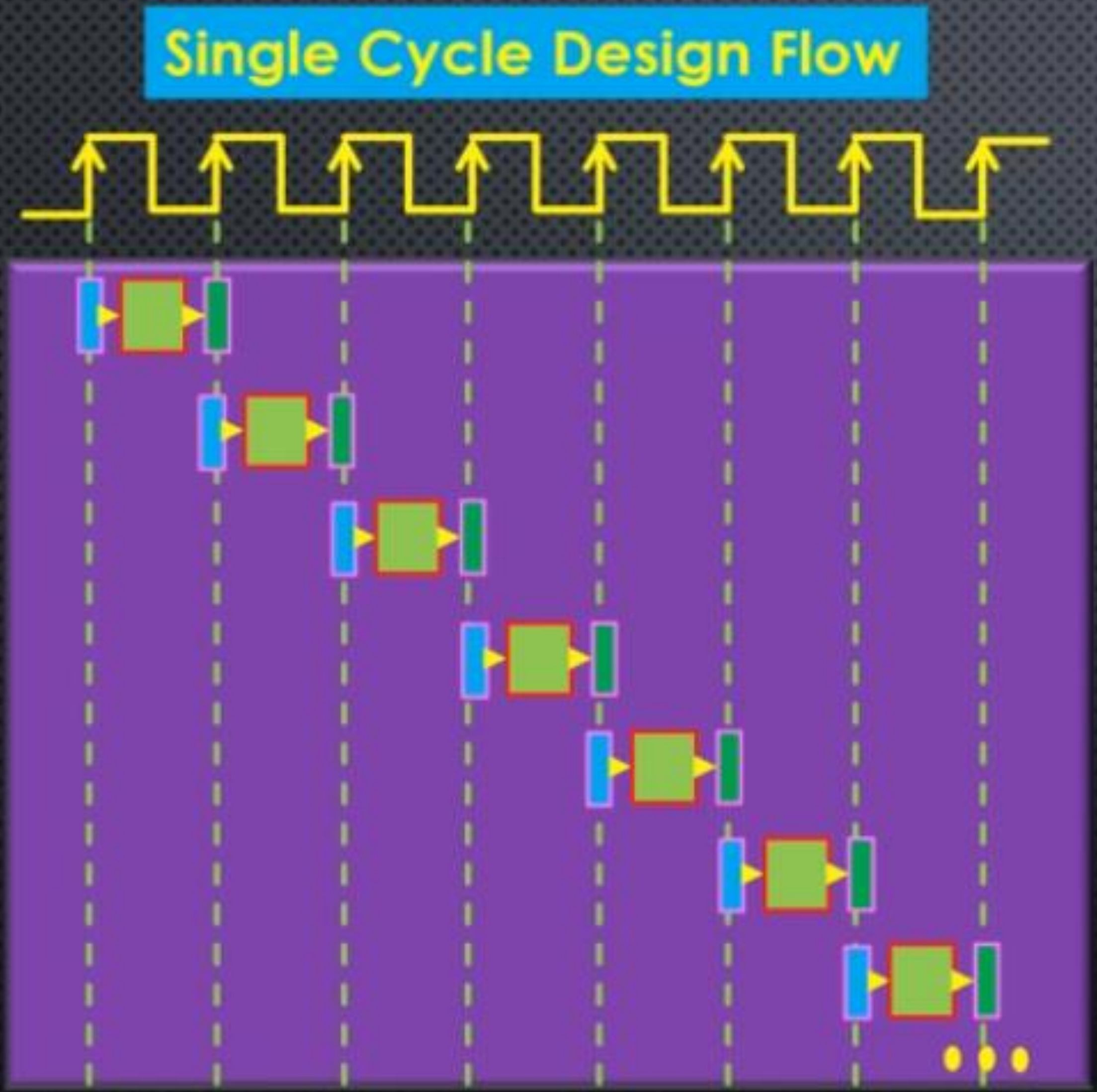
PIPELINE EXECUTION

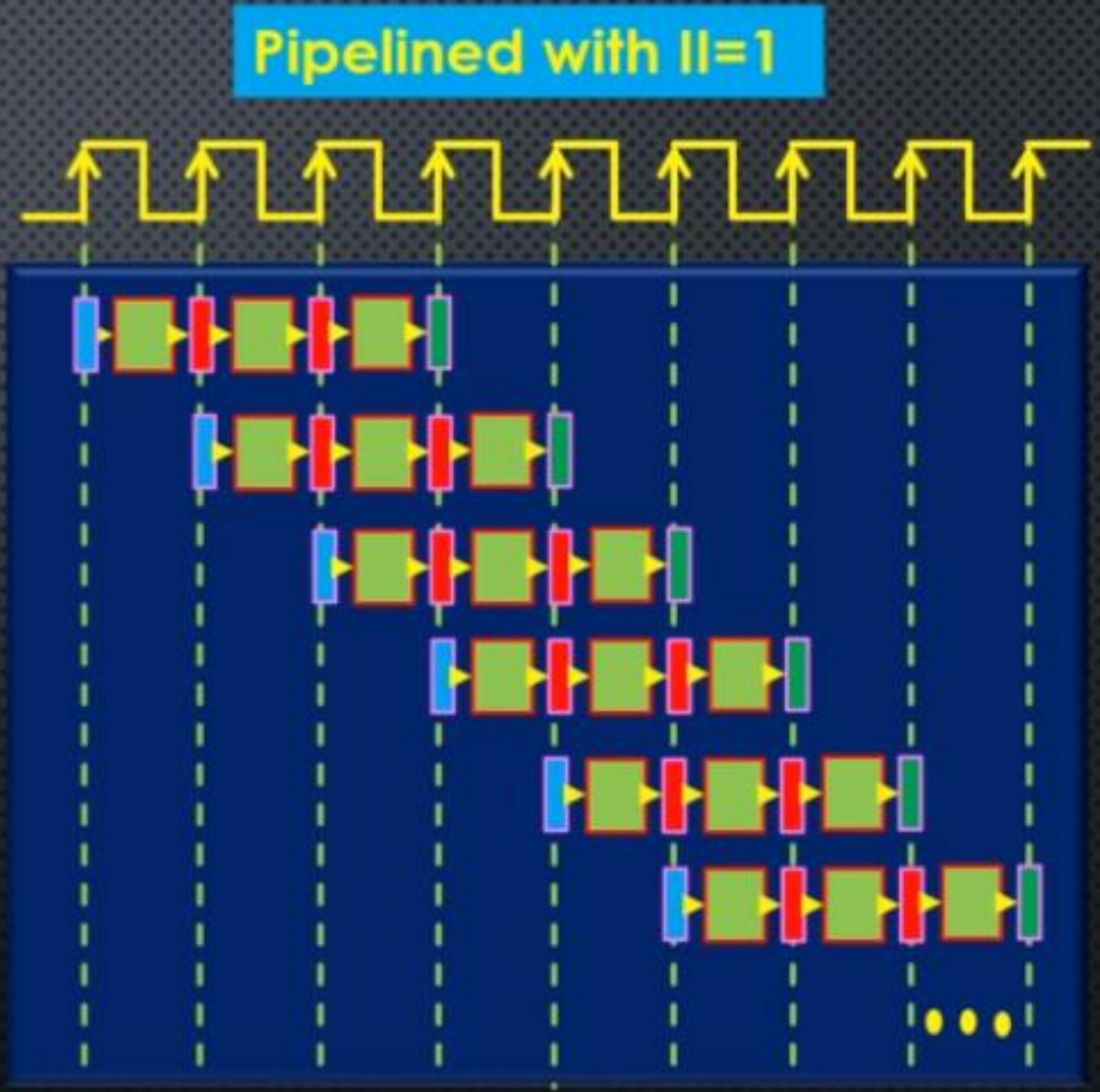


$II=1$

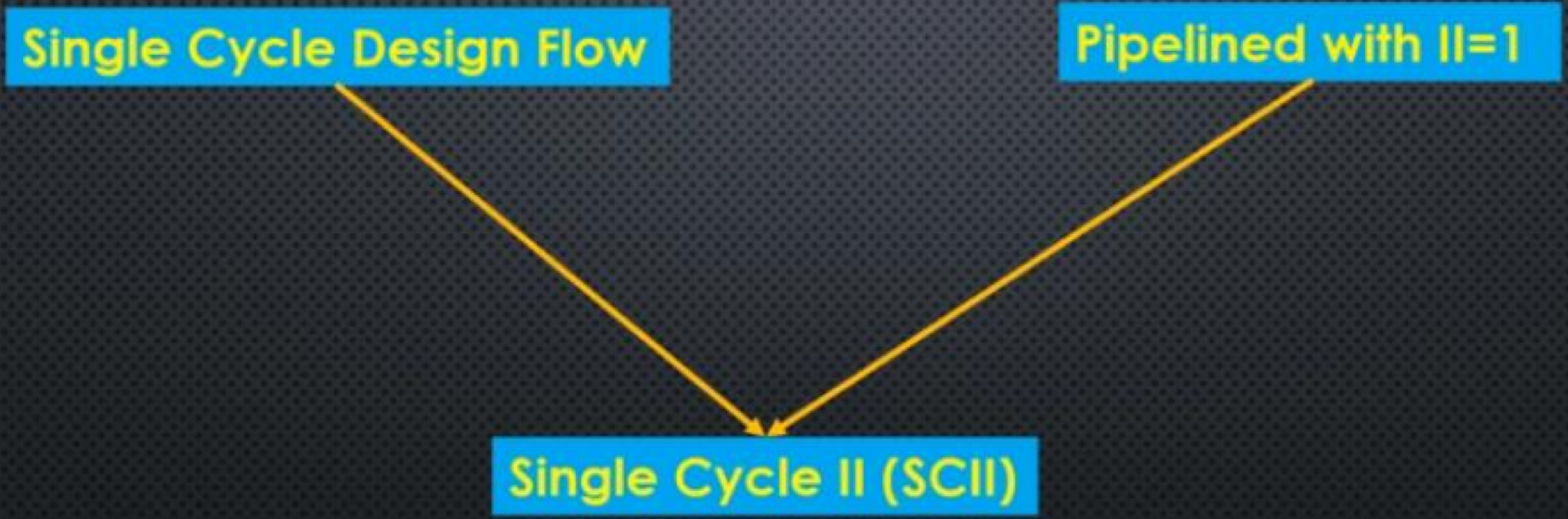
Initiation Interval

DESIGN TECHNIQUES





SINGLE CYCLE II (SCII)



Single Cycle Design Flow

Pipelined with II=1

Single Cycle II (SCII)

SCII : pronounced as ski

PERFORMANCE METRICS

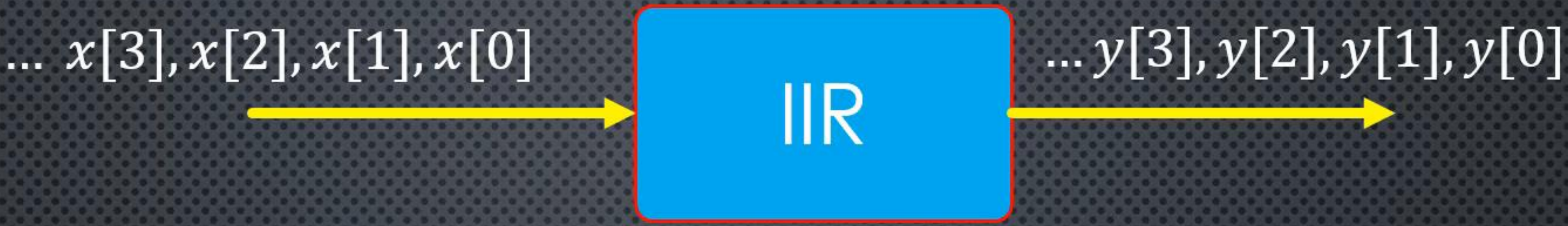
Initiation Interval (II): Represents the speed that a circuit can accept new inputs

Latency (L): Represents the timing between an input and its corresponding output

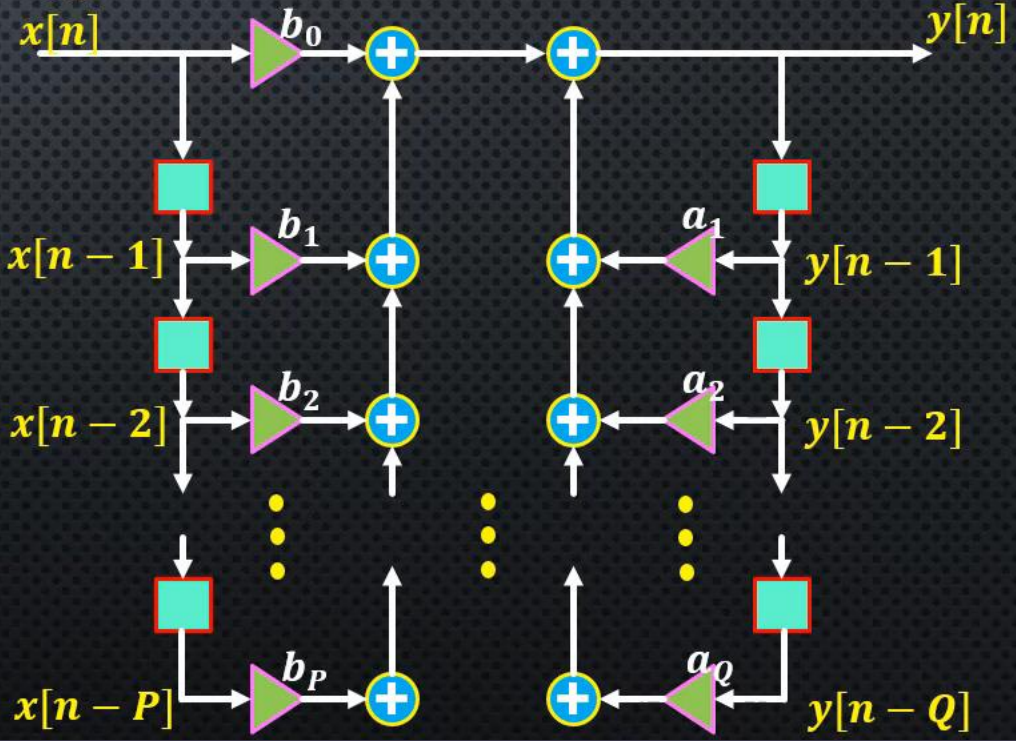
Throughput (T): Represents the speed of generating output

Infinite Impulse Response

IIR



$$\mathbf{y}[\boldsymbol{n}] = \ \boldsymbol{b_0} * \boldsymbol{x}[\boldsymbol{n}] + \boldsymbol{b_1} * \boldsymbol{x}[\boldsymbol{n}-\mathbf{1}] + \cdots + \boldsymbol{b_P} * \boldsymbol{x}[\boldsymbol{n}-\boldsymbol{P}]$$
$$-\boldsymbol{a_1} * \boldsymbol{y}[\boldsymbol{n}-\mathbf{1}] + \boldsymbol{a_2} * \boldsymbol{y}[\boldsymbol{n}-\mathbf{2}] - \cdots - \boldsymbol{a_Q}\boldsymbol{y}[\boldsymbol{n}-\boldsymbol{Q}];$$



$\boldsymbol{P}$ → the feedforward filter order

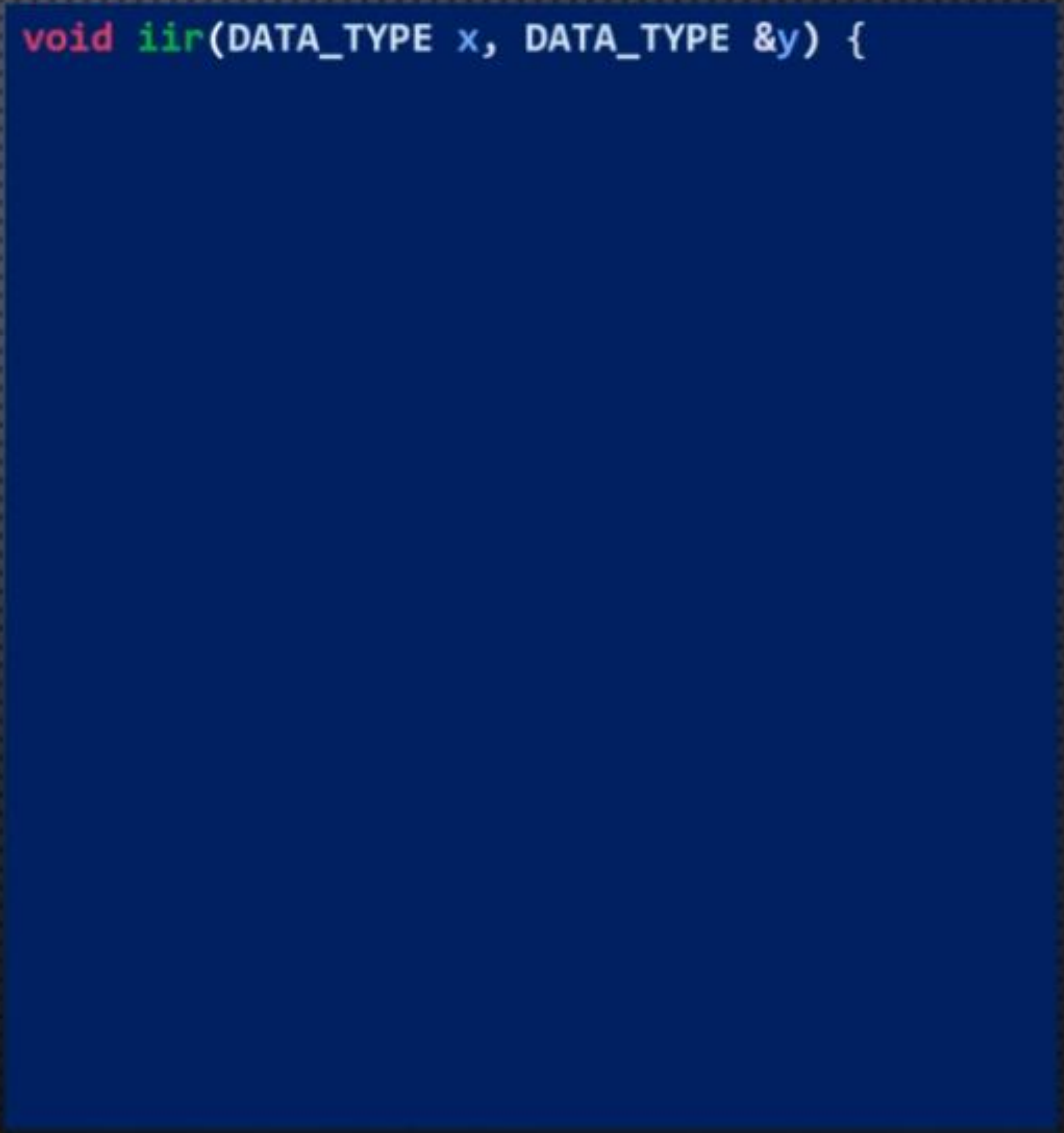
$\boldsymbol{b_i}$ → the feedforward filter coefficients

$\boldsymbol{Q}$ → is the feedback filter order

$\boldsymbol{a_i}$ → the feedback filter coefficients

IIR- EXAMPLE

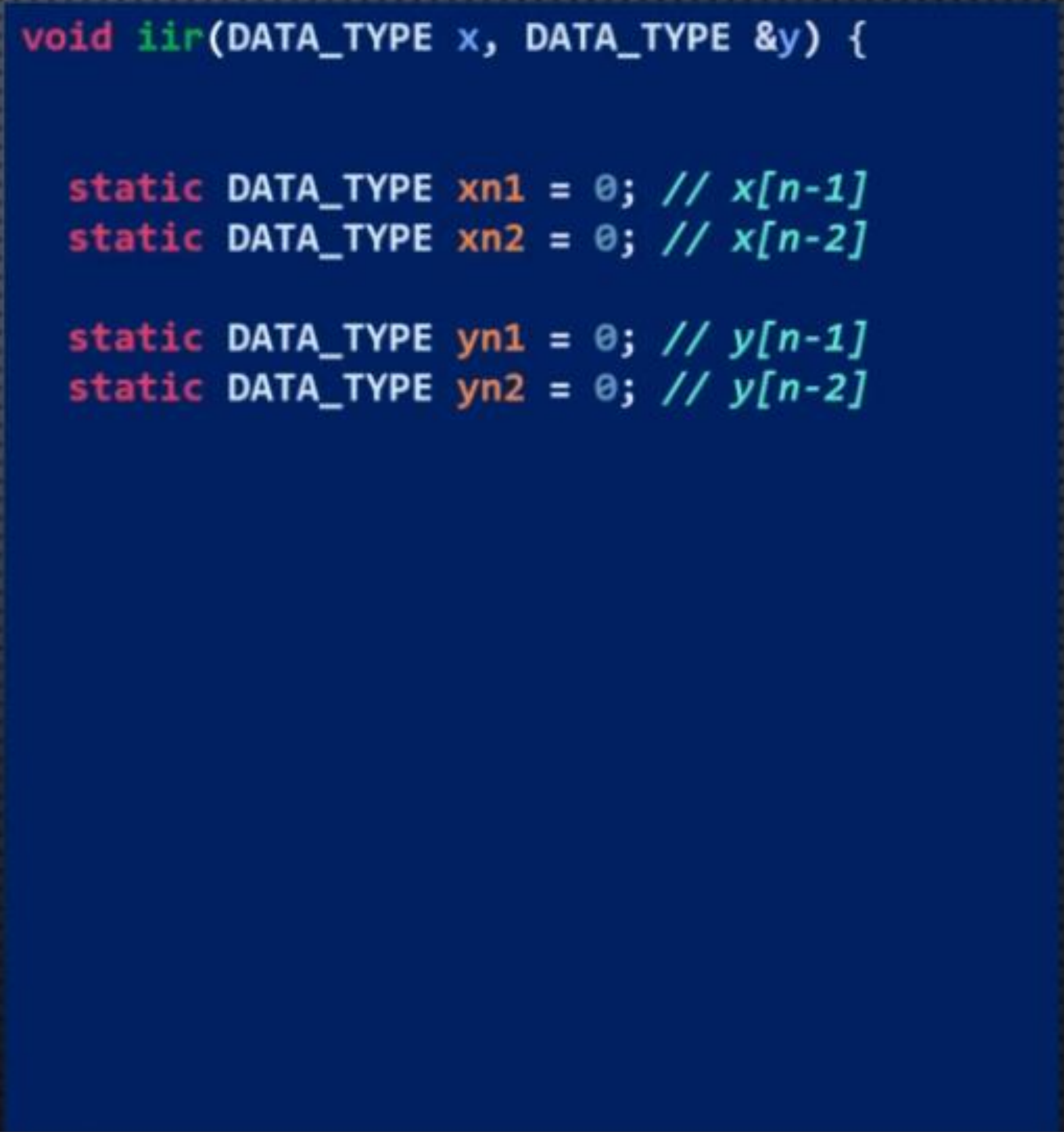
$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2]$$
$$-a_1 * y[n-1] - a_2 * y[n-2];$$



```
void iir(DATA_TYPE x, DATA_TYPE &y) {
```

IIR- EXAMPLE

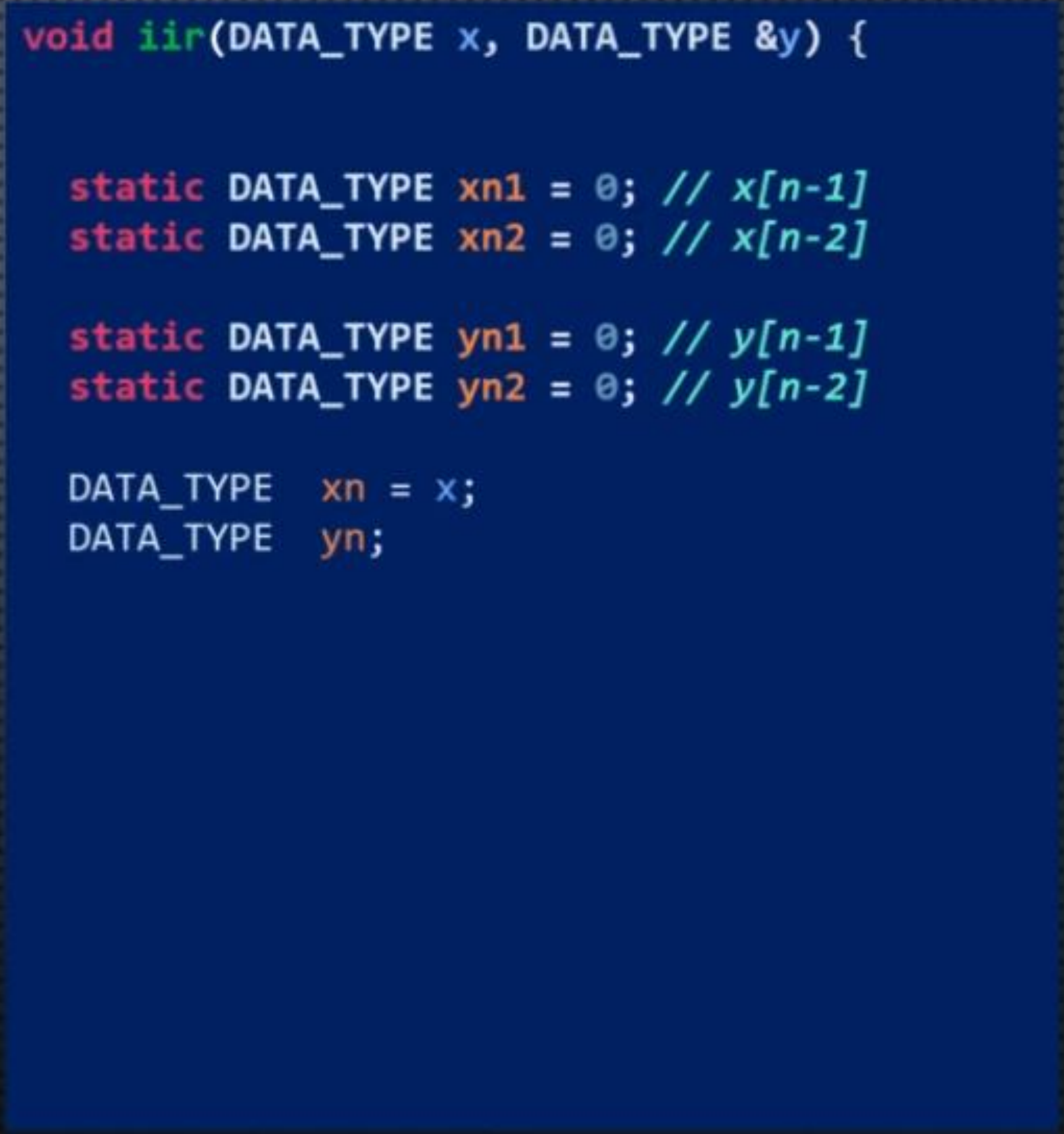
$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] \\ -a_1 * y[n-1] - a_2 * y[n-2];$$



```
void iir(DATA_TYPE x, DATA_TYPE &y) {


  static DATA_TYPE xn1 = 0; // x[n-1]
  static DATA_TYPE xn2 = 0; // x[n-2]

  static DATA_TYPE yn1 = 0; // y[n-1]
  static DATA_TYPE yn2 = 0; // y[n-2]
```

IIR- EXAMPLE

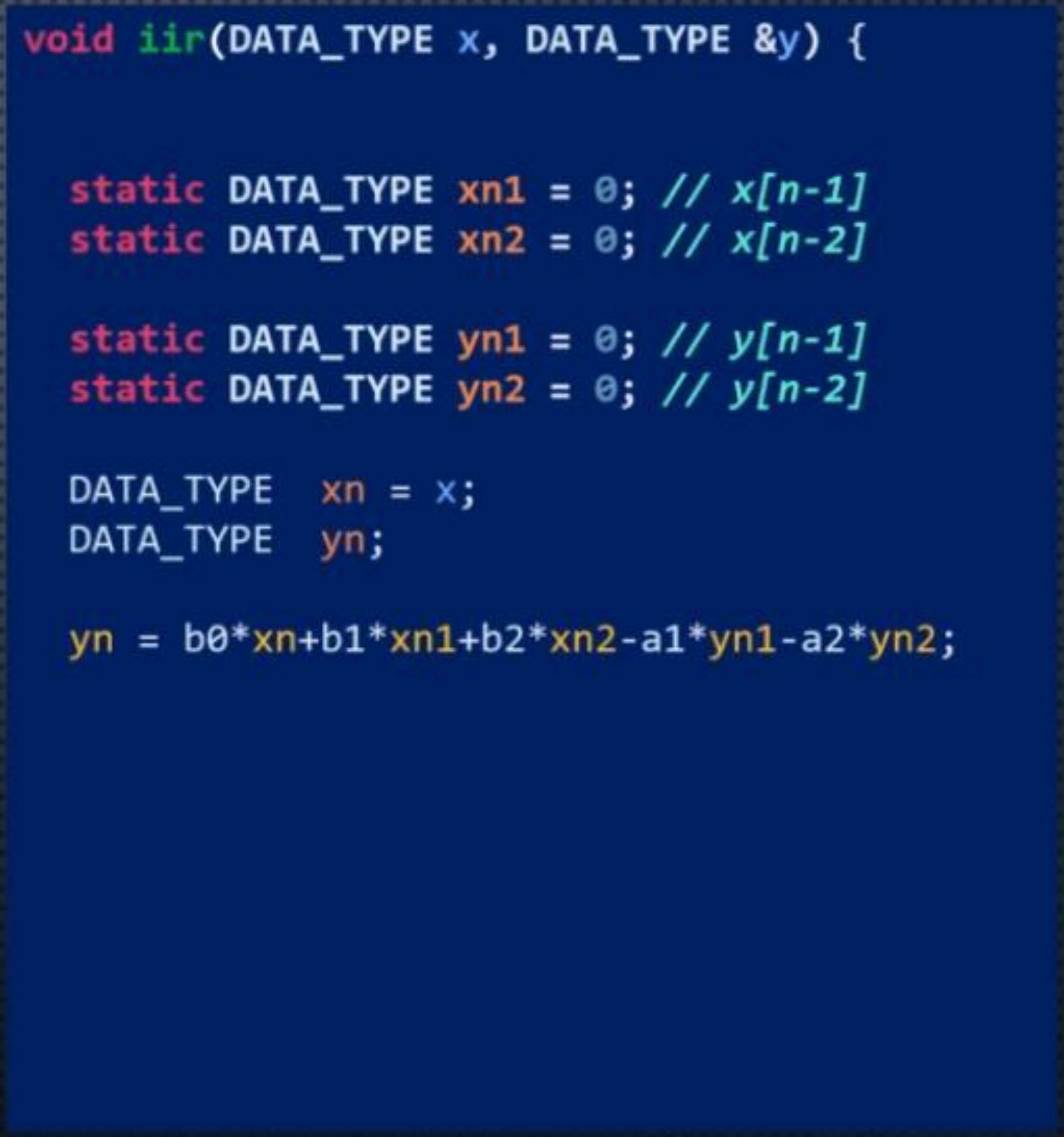
$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] \\ -a_1 * y[n-1] - a_2 * y[n-2];$$



```
void iir(DATA_TYPE x, DATA_TYPE &y) {


  static DATA_TYPE xn1 = 0; // x[n-1]
  static DATA_TYPE xn2 = 0; // x[n-2]

  static DATA_TYPE yn1 = 0; // y[n-1]
  static DATA_TYPE yn2 = 0; // y[n-2]

  DATA_TYPE  xn = x;
  DATA_TYPE  yn;
```

IIR- EXAMPLE

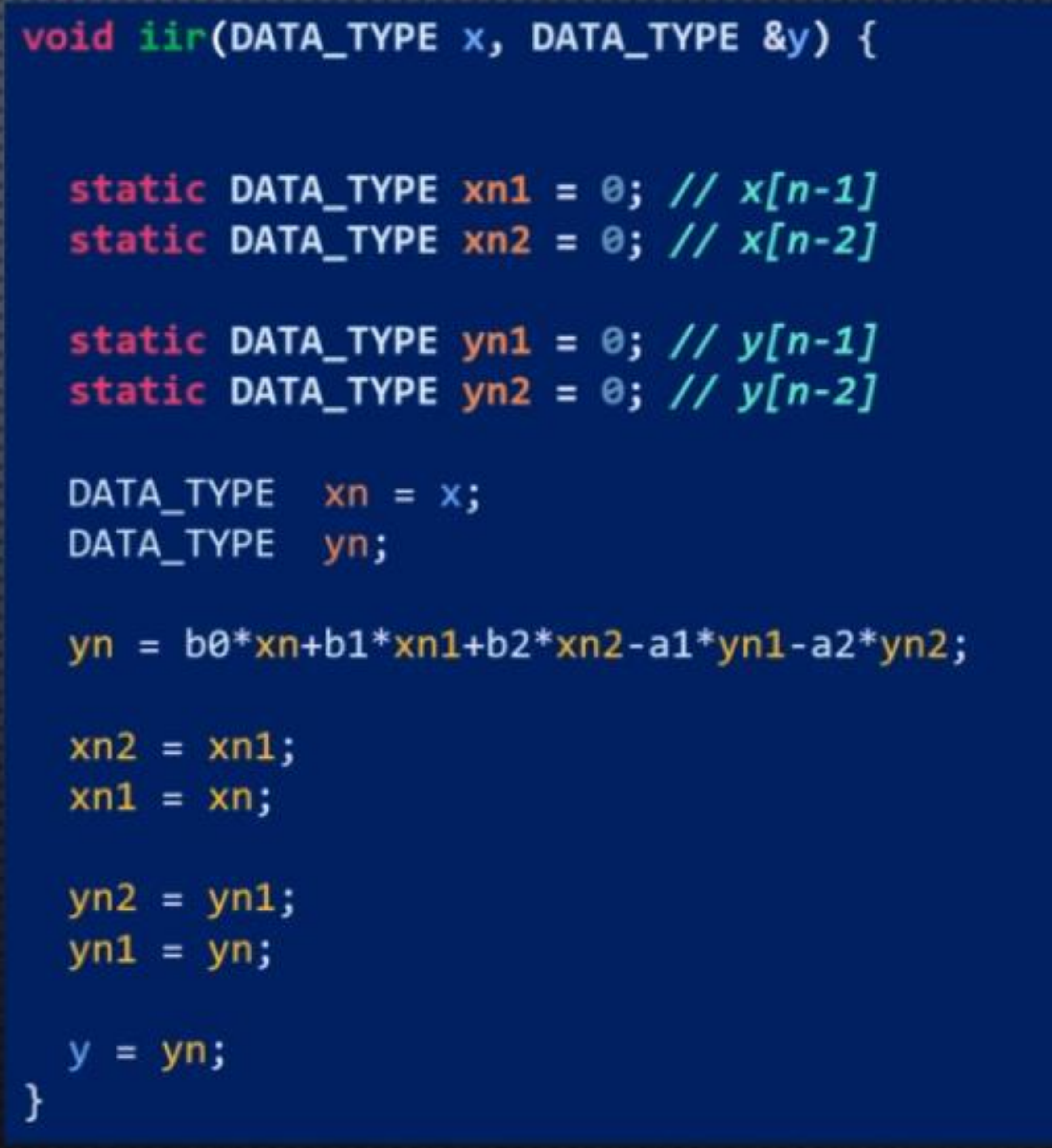
$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2]$$
$$-a_1 * y[n-1] - a_2 * y[n-2];$$



```
void iir(DATA_TYPE x, DATA_TYPE &y) {


  static DATA_TYPE xn1 = 0; // x[n-1]
  static DATA_TYPE xn2 = 0; // x[n-2]

  static DATA_TYPE yn1 = 0; // y[n-1]
  static DATA_TYPE yn2 = 0; // y[n-2]

  DATA_TYPE  xn = x;
  DATA_TYPE  yn;

  yn = b0*xn+b1*xn1+b2*xn2-a1*yn1-a2*yn2;
```

IIR- EXAMPLE

$$y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] \\ -a_1 * y[n-1] - a_2 * y[n-2];$$



```
void iir(DATA_TYPE x, DATA_TYPE &y) {


  static DATA_TYPE xn1 = 0; // x[n-1]
  static DATA_TYPE xn2 = 0; // x[n-2]

  static DATA_TYPE yn1 = 0; // y[n-1]
  static DATA_TYPE yn2 = 0; // y[n-2]

  DATA_TYPE  xn = x;
  DATA_TYPE  yn;

  yn = b0*xn+b1*xn1+b2*xn2-a1*yn1-a2*yn2;

  xn2 = xn1;
  xn1 = xn;

  yn2 = yn1;
  yn1 = yn;

  y = yn;
}
```

Any Question…

Thank you