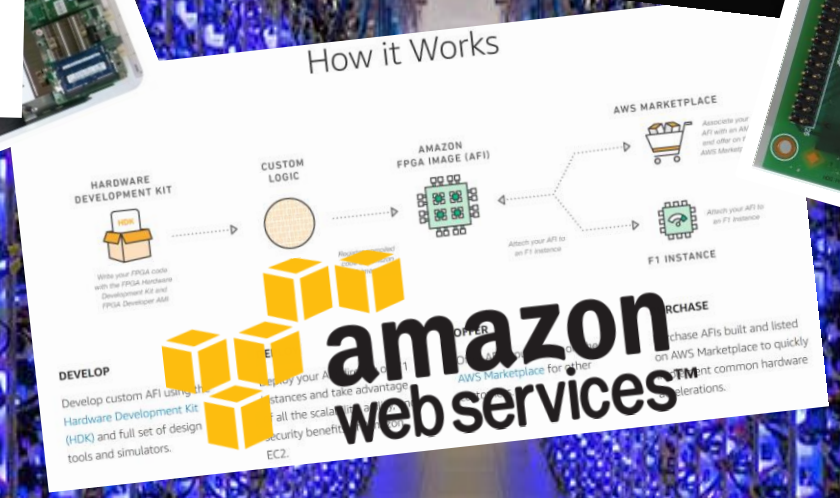
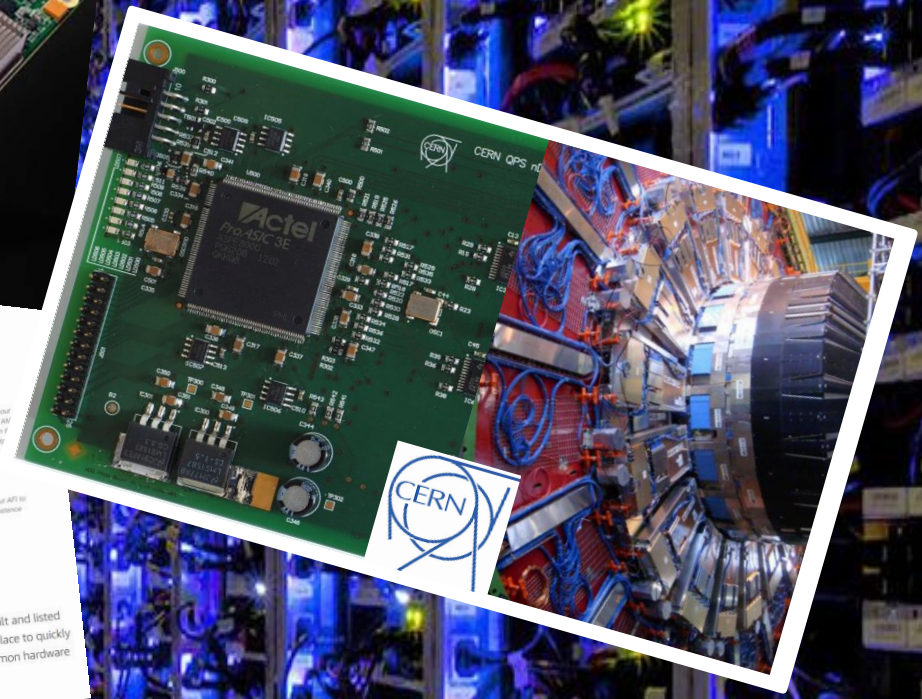
High-Level Synthesis of Dynamically Scheduled Circuits

Lana Josipović

December 2022

ETH zürich





How to perform hardware design?

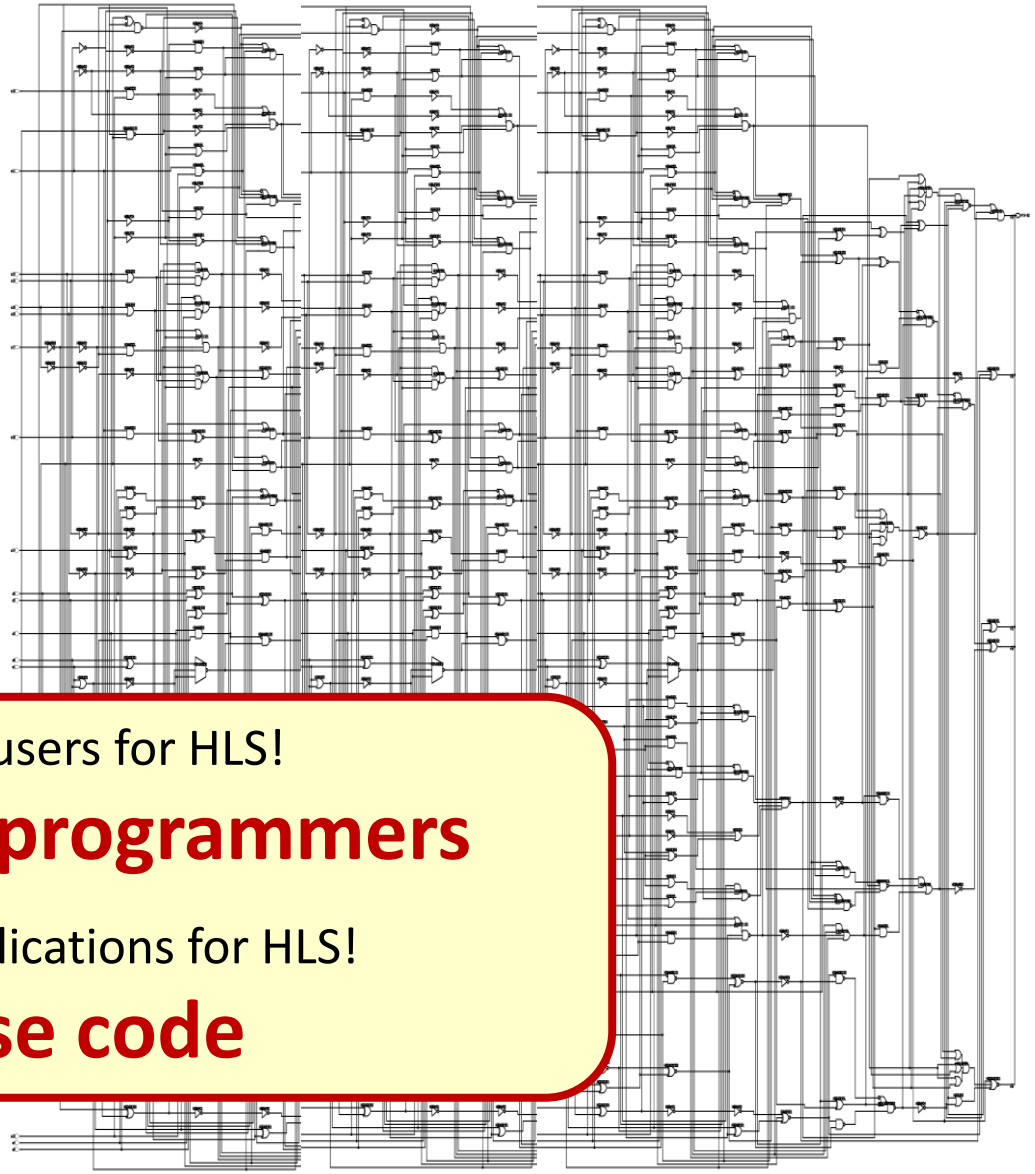
~~High parallelism and energy efficiency~~

High-Level Synthesis: From Programs to Circuits

```
#define PI 3.14159265358979323846264338

complex* DFT_naive(complex* x, int N) {
  complex* X = (complex*) malloc(sizeof(struct complex_t) * N);
  int k, n;
  for(k = 0; k < N; k++) {
    X[k].re = 0.0;
    X[k].im = 0.0;
    for(n = 0; n < N; n++) {
      X[k] = add(X[k], multiply(x[n],
                                conv_from_polar(1,
                                                -2*PI*n*k/N)));
    }
  }

  return X;
}
```



A completely new type of users for HLS!

Software application programmers

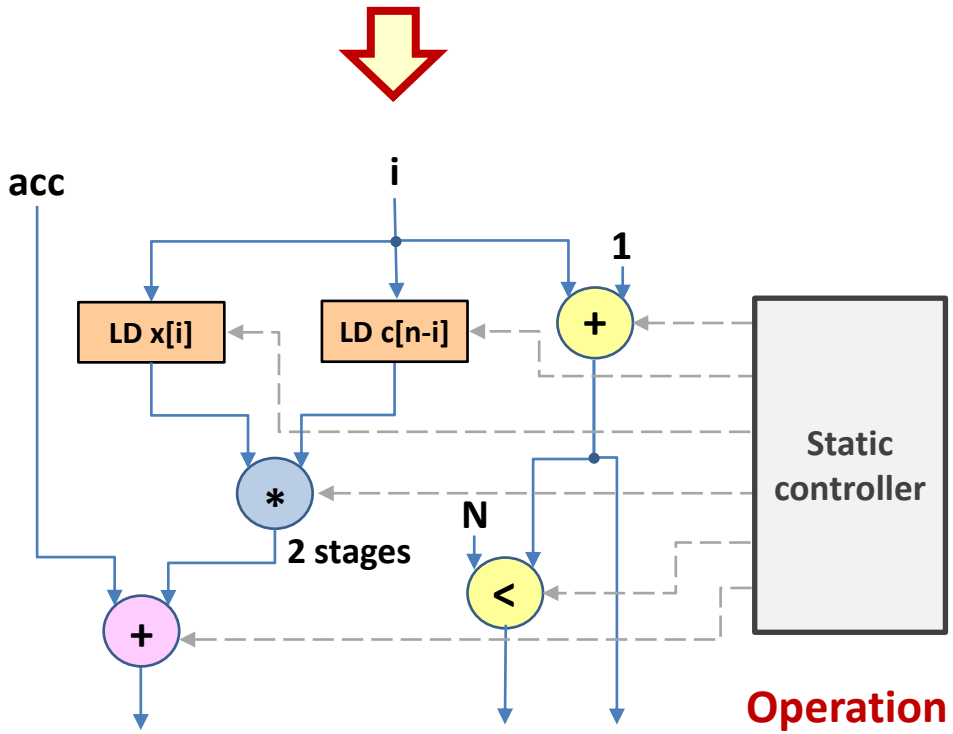
A completely new type of applications for HLS!

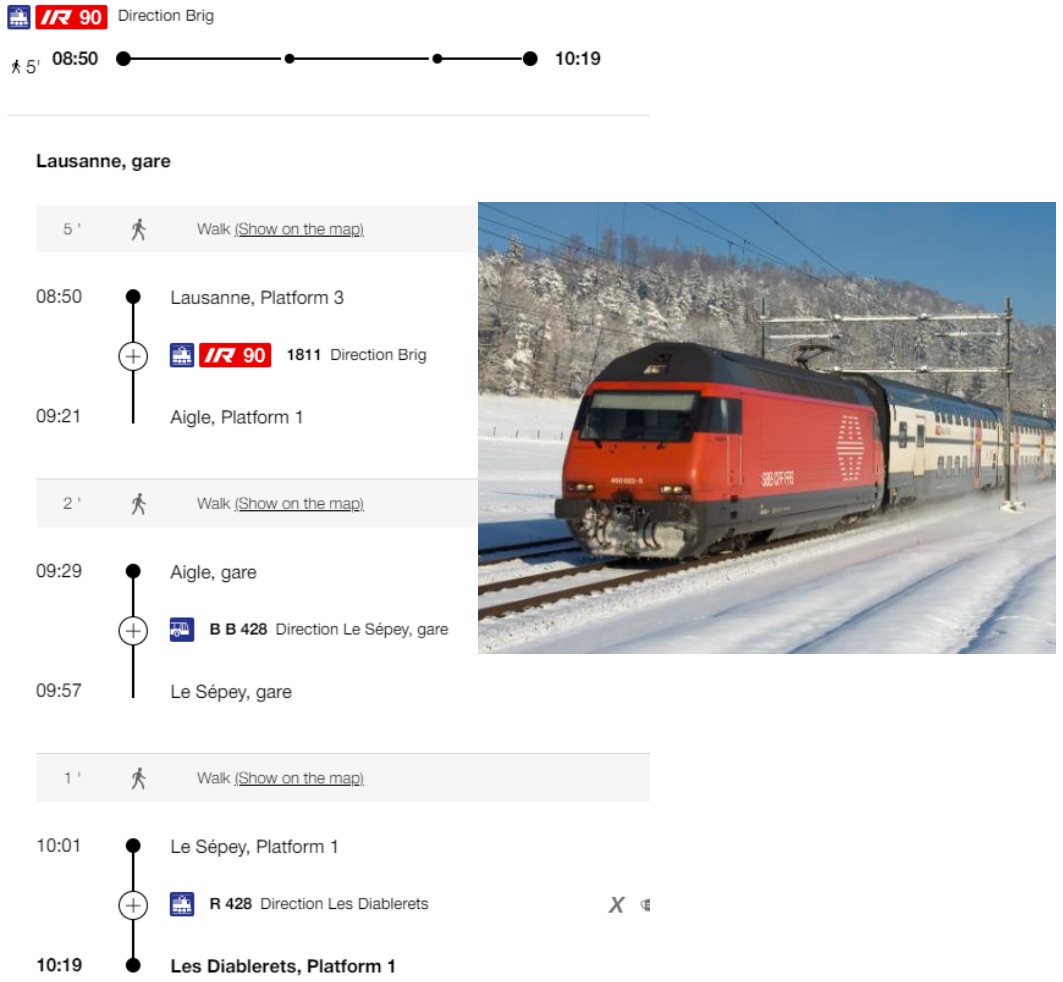
General-purpose code

Standard HLS

- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```

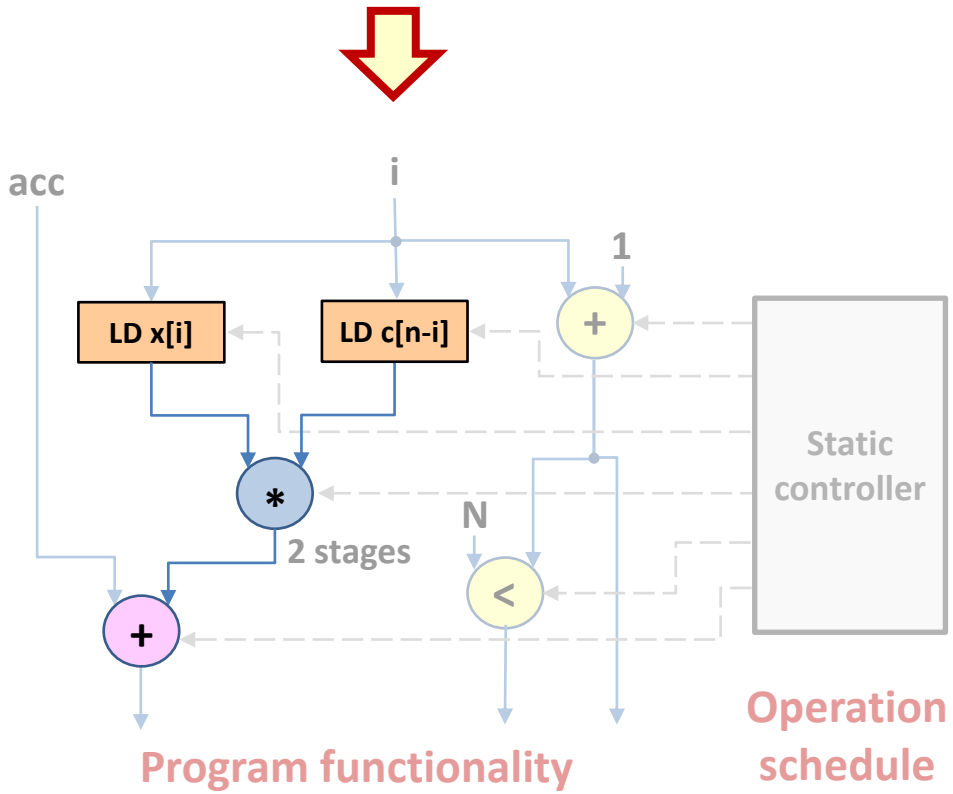


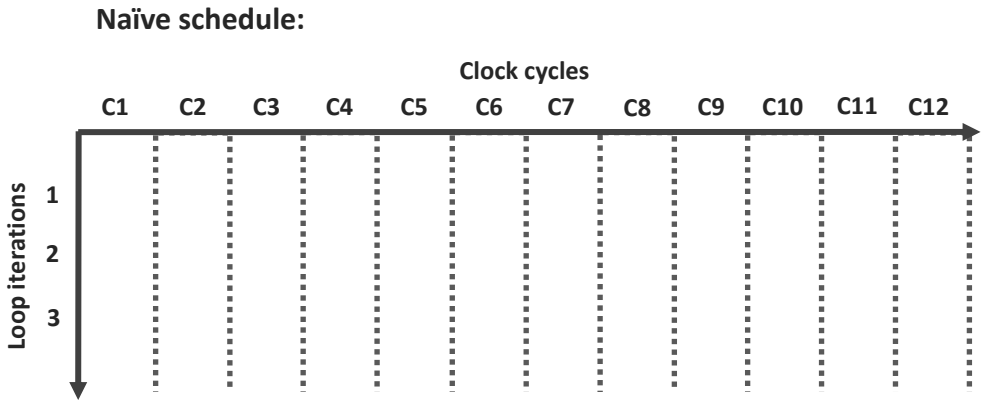


Standard HLS

- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```

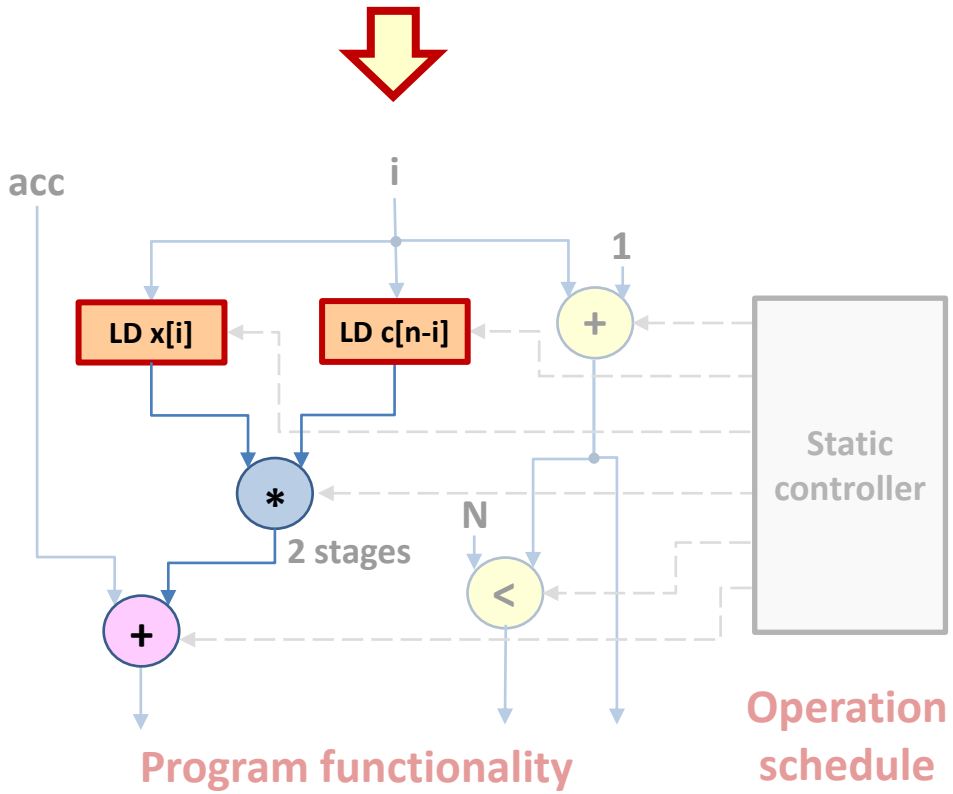


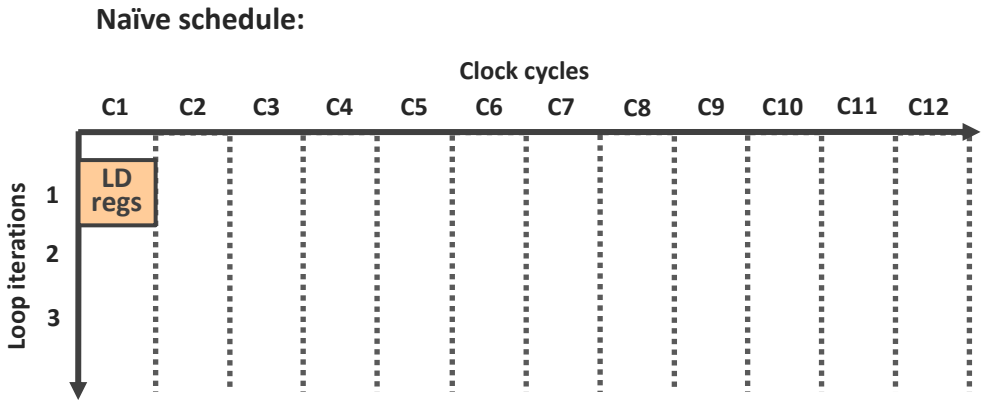


Standard HLS

- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```

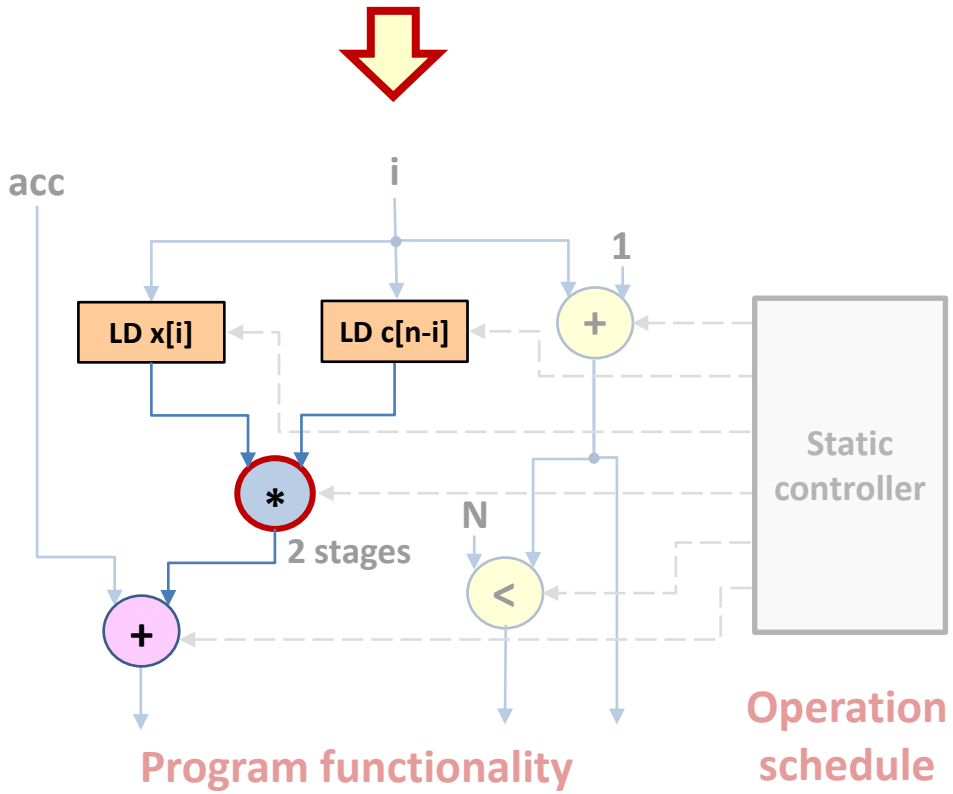




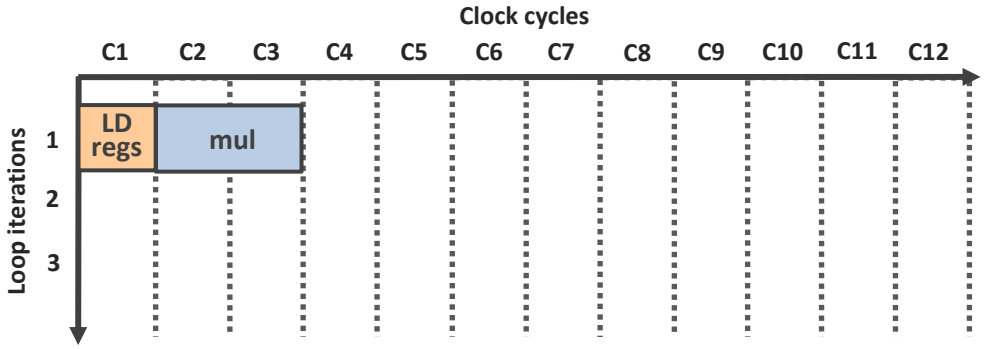
Standard HLS

- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```



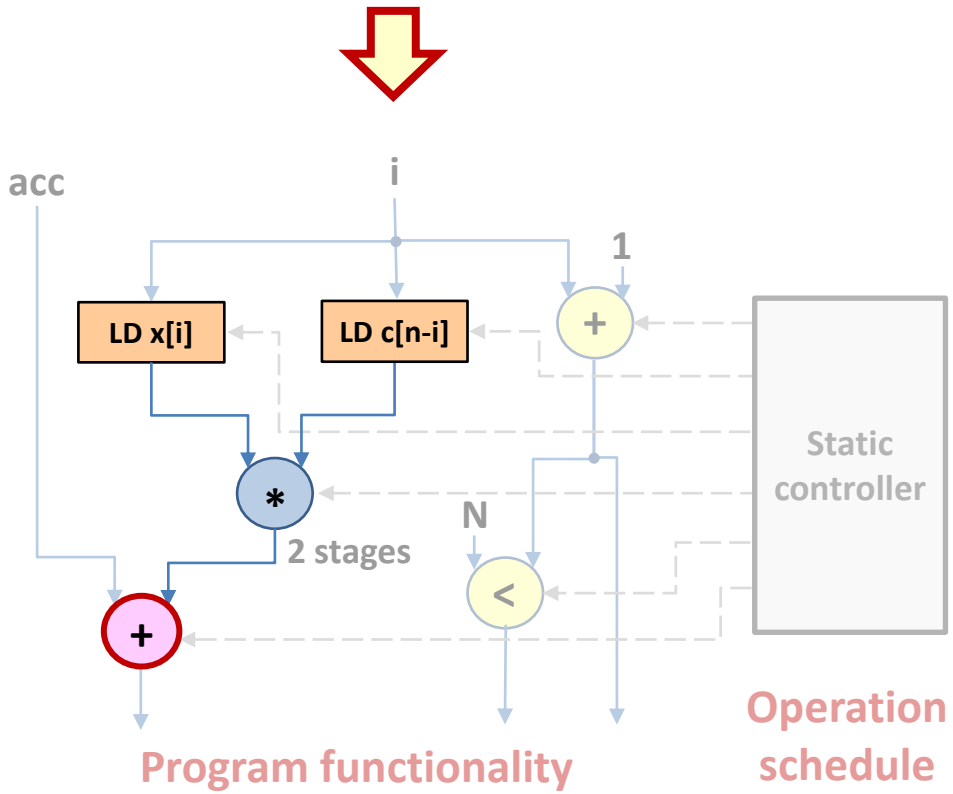
Naïve schedule:



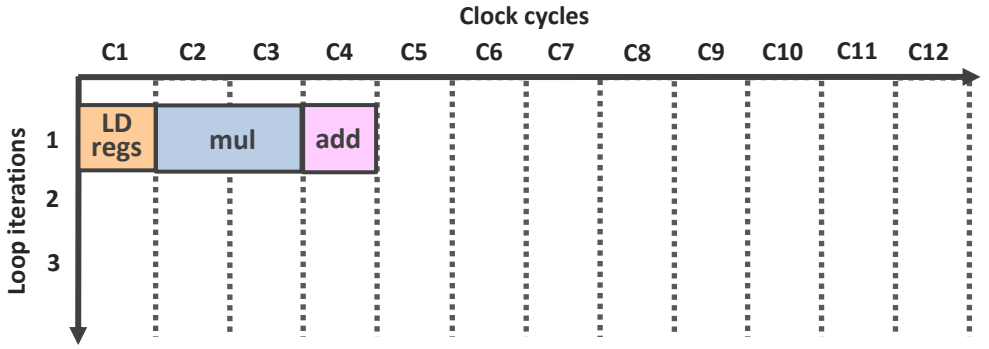
Standard HLS

- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```



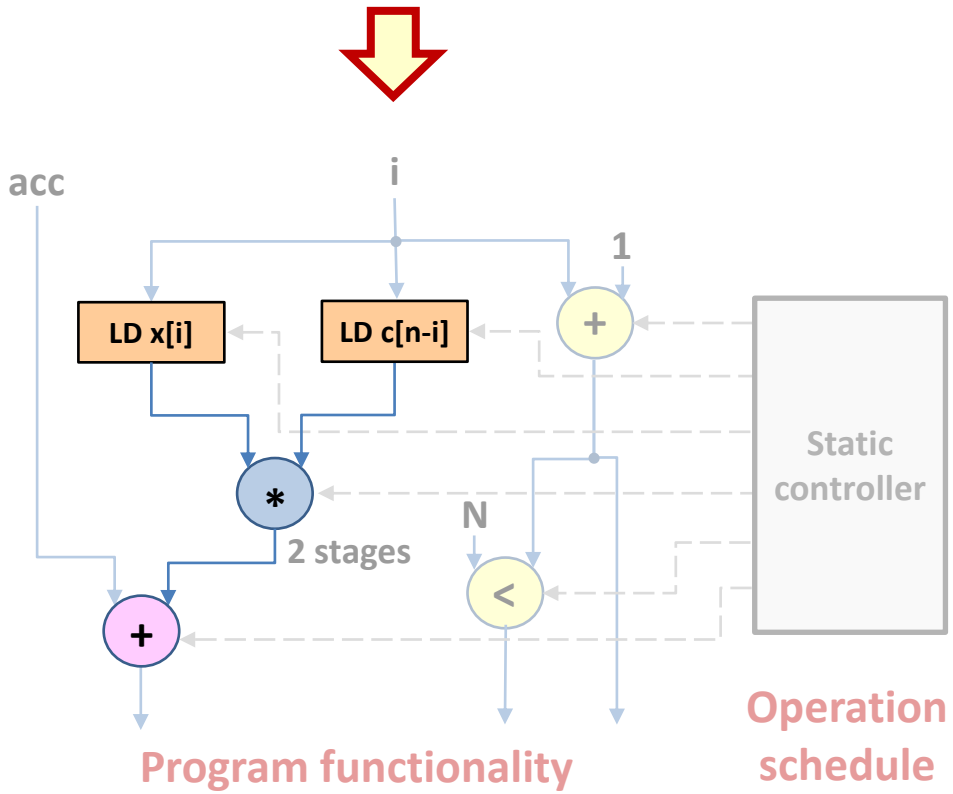
Naïve schedule:



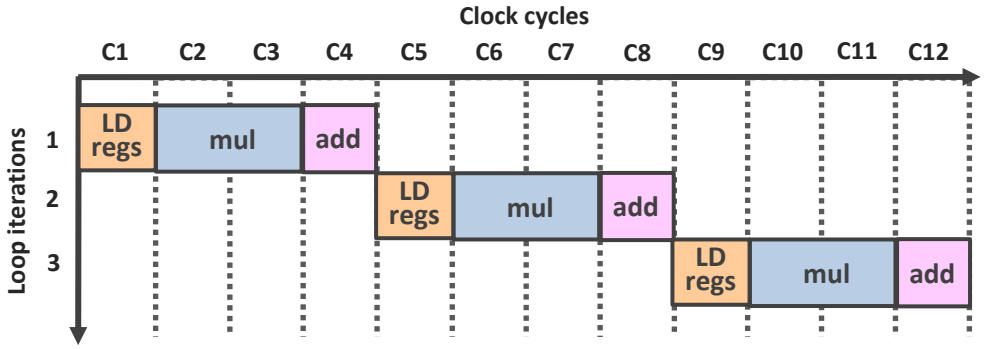
Standard HLS

- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```



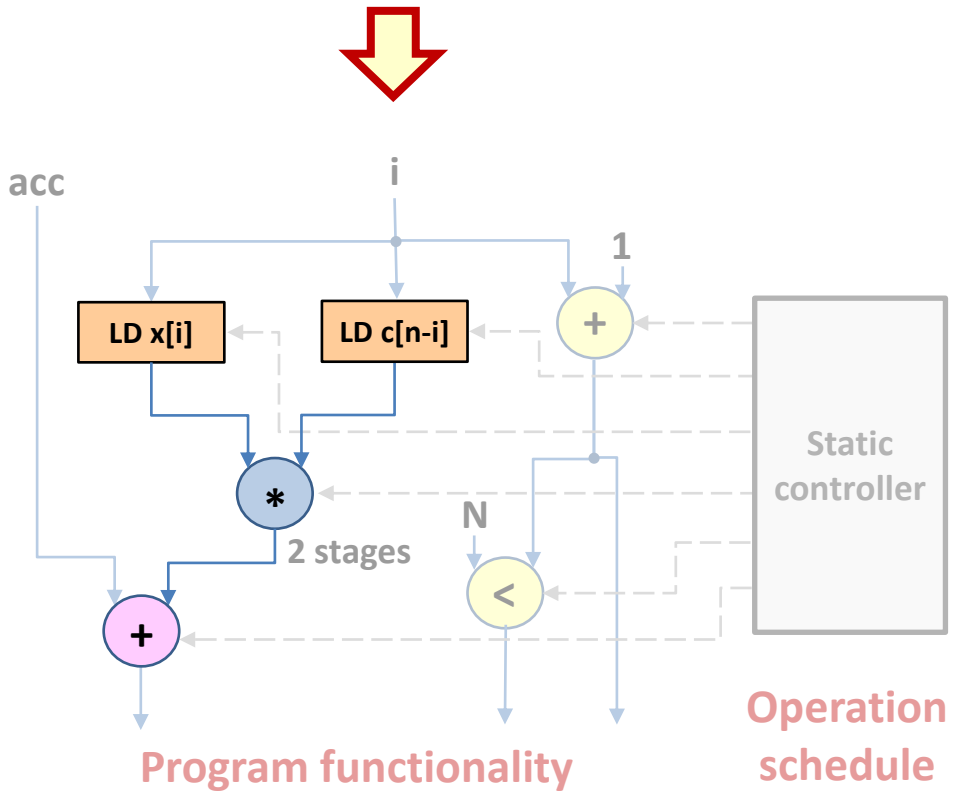
Naïve schedule:



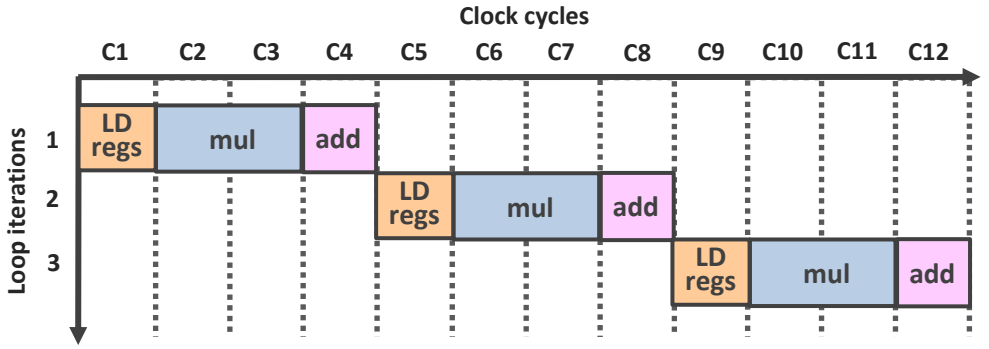
Standard HLS

- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```



Naïve schedule:

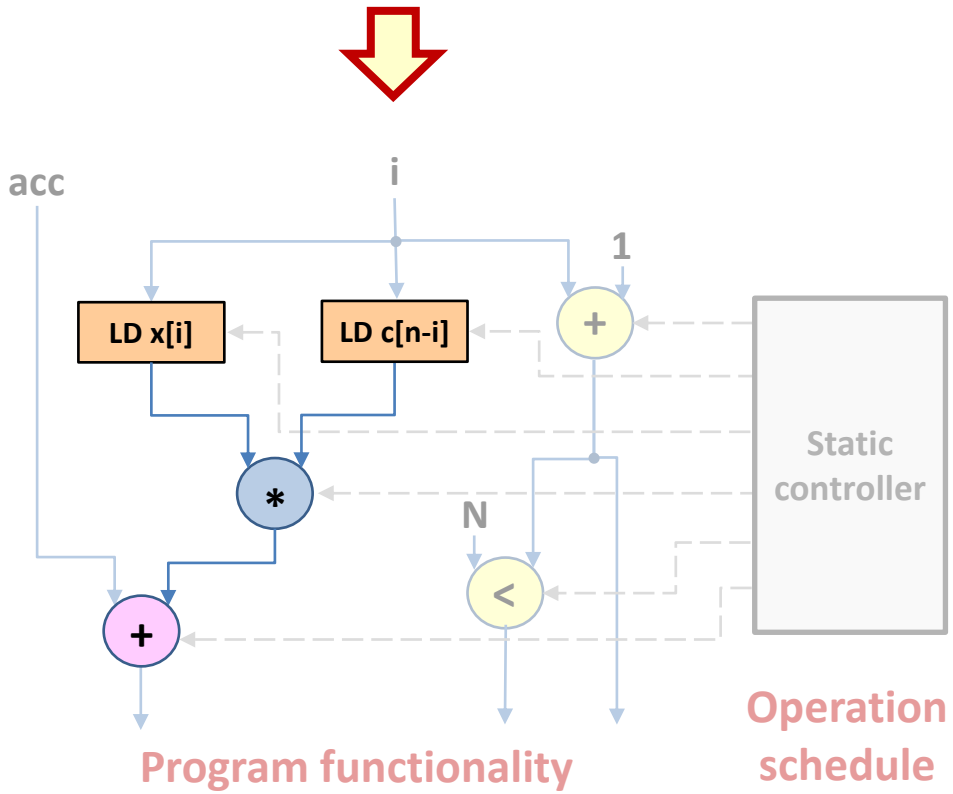


Low throughput: slow execution

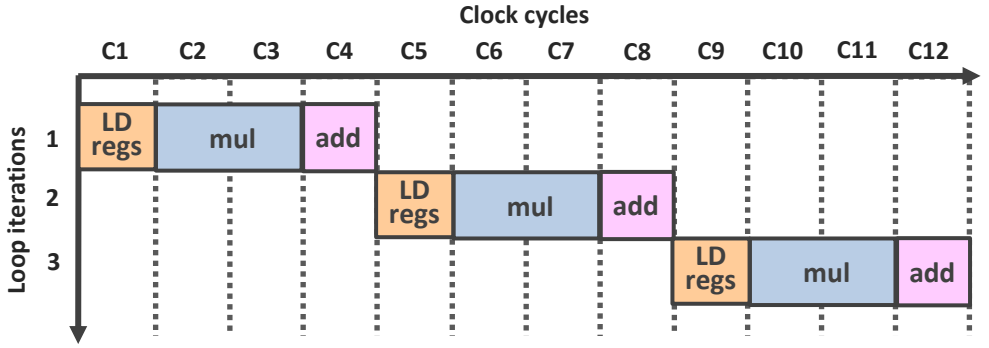
Standard HLS

- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

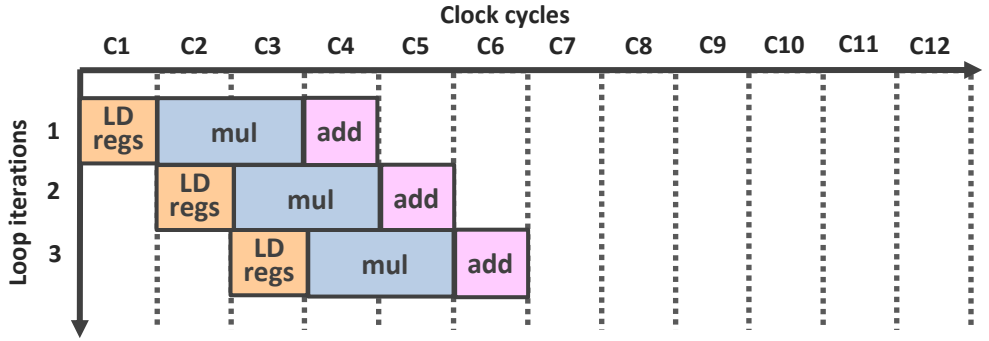
```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```



Naïve schedule:



Pipelined schedule:



High throughput: fast execution

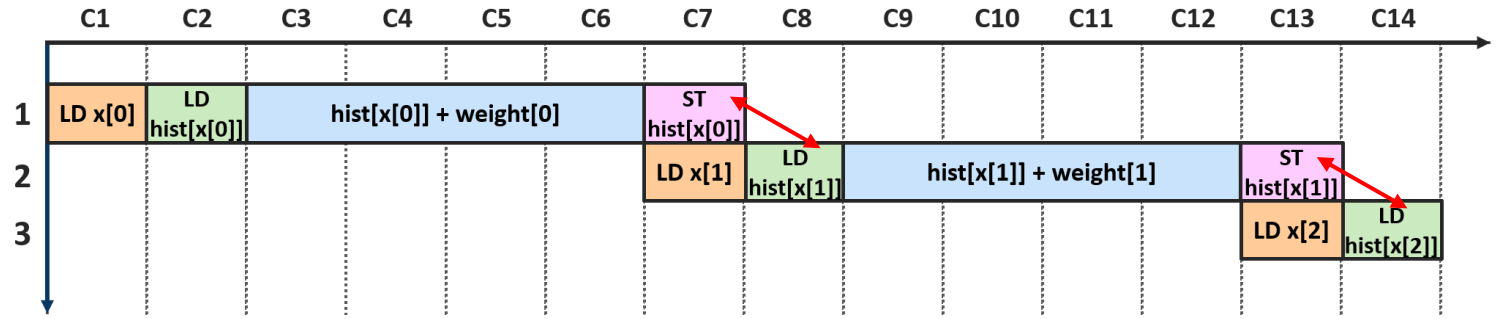
The Limitations of Static Scheduling

```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

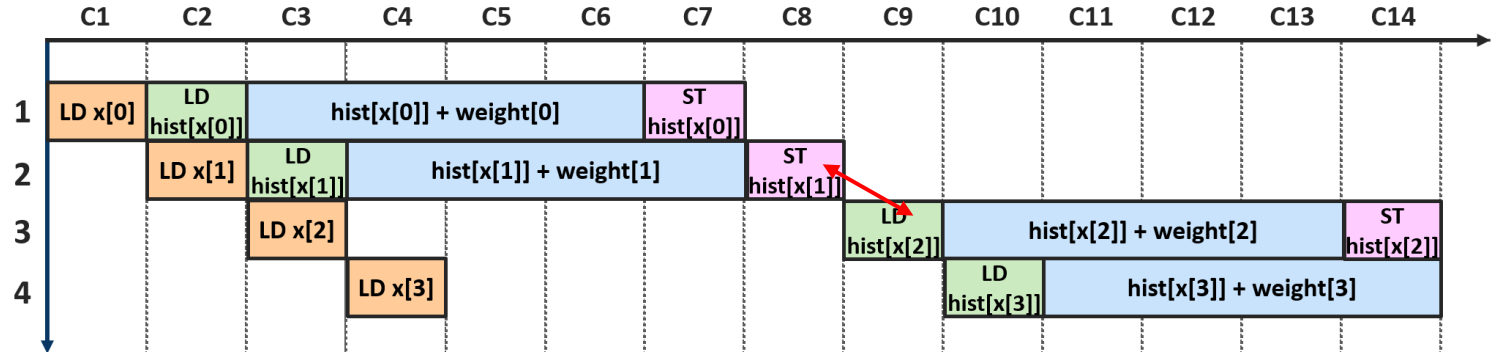
```
1: x[0]=5 → ld hist[5]; st hist[5];
2: x[1]=4 → ld hist[4]; st hist[4];
3: x[2]=4 → ld hist[4]; st hist[4];
```

RAW dependency

- Static scheduling (standard HLS tool)
  - Inferior when memory accesses cannot be disambiguated at compile time

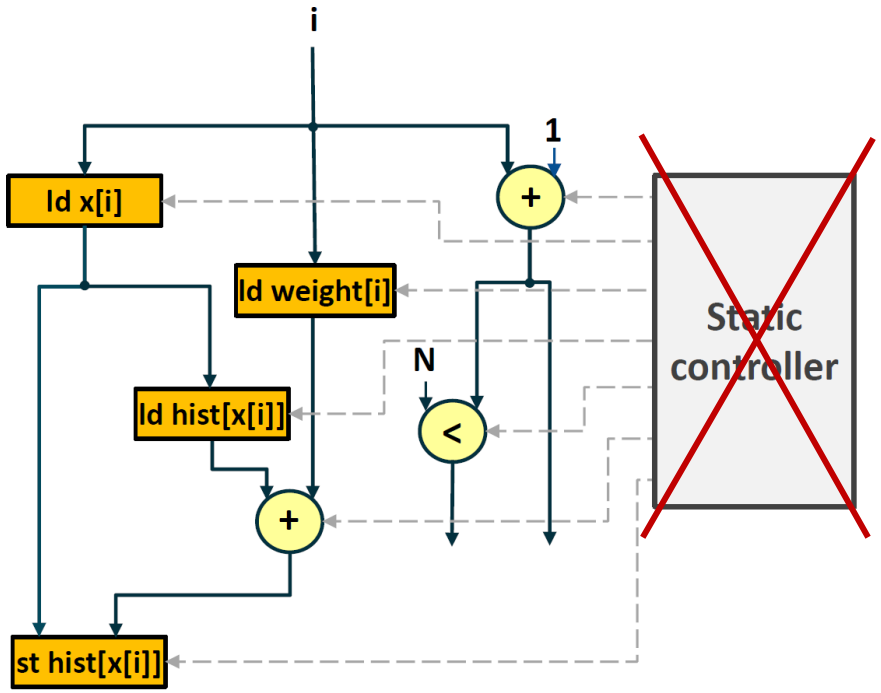


- Dynamic scheduling
  - Maximum parallelism: Only serialize memory accesses on actual dependencies

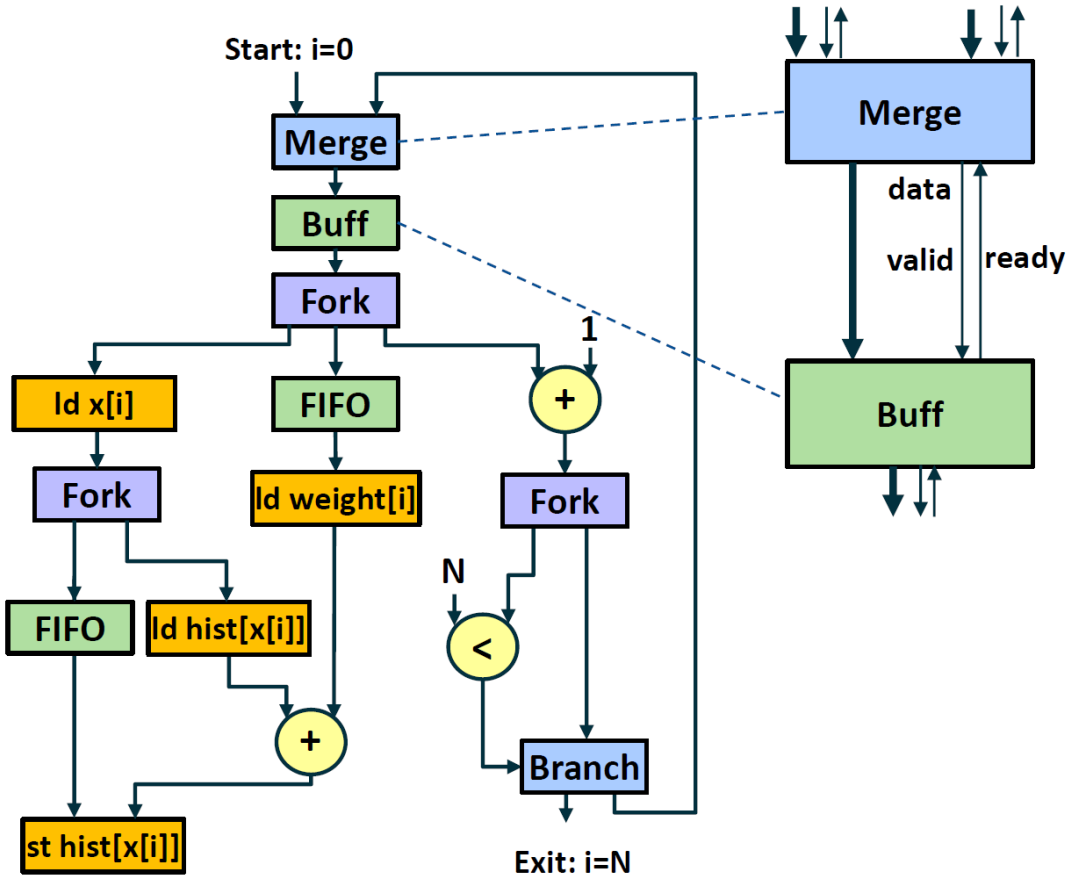


A Different Way to Do HLS

Static scheduling (standard HLS tool): decide at **compile time** when each operation executes



Dynamic scheduling (our HLS approach): decide at **runtime** when each operation executes



A Different Way to Do HLS

Static scheduling (standard HLS tool): decide at **compile time** when each operation executes

IR 90 Direction Brig

5' 08:50 — 10:19

Lausanne, gare

5 ' Walk (Show on the map)

08:50 Lausanne, Platform 3

IR 90 1811 Direction Brig 1

09:21 Aigle, Platform 1

2 ' Walk (Show on the map)

09:29 Aigle, gare

B B 428 Direction Le Sépey, gare

09:57 Le Sépey, gare

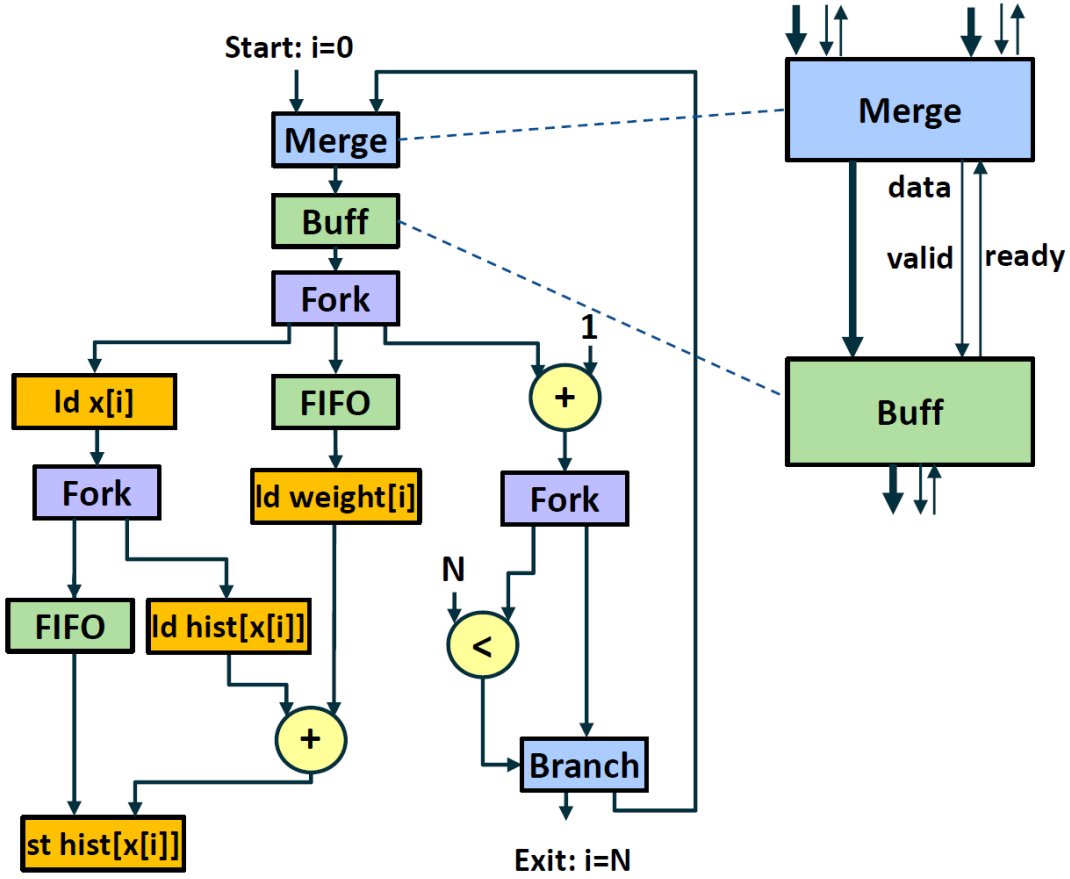
1 ' Walk (Show on the map)

10:01 Le Sépey, Platform 1

R 428 Direction Les Diablerets

10:19 **Les Diablerets, Platform 1**

Dynamic scheduling (our HLS approach): decide at **runtime** when each operation executes



A Different Way to Do HLS

Static scheduling (standard HLS tool): decide at **compile time** when each operation executes

IR 90 Direction Brig

5' 08:50 — 10:19

Lausanne, gare

5 ' Walk (Show on the map)

08:50 Lausanne, Platform 3

IR 90 1811 Direction Brig 1

09:21 Aigle, Platform 1

2 ' Walk (Show on the map)

09:29 Aigle, gare

B B 428 Direction Le Sépey, gare

09:57 Le Sépey, gare

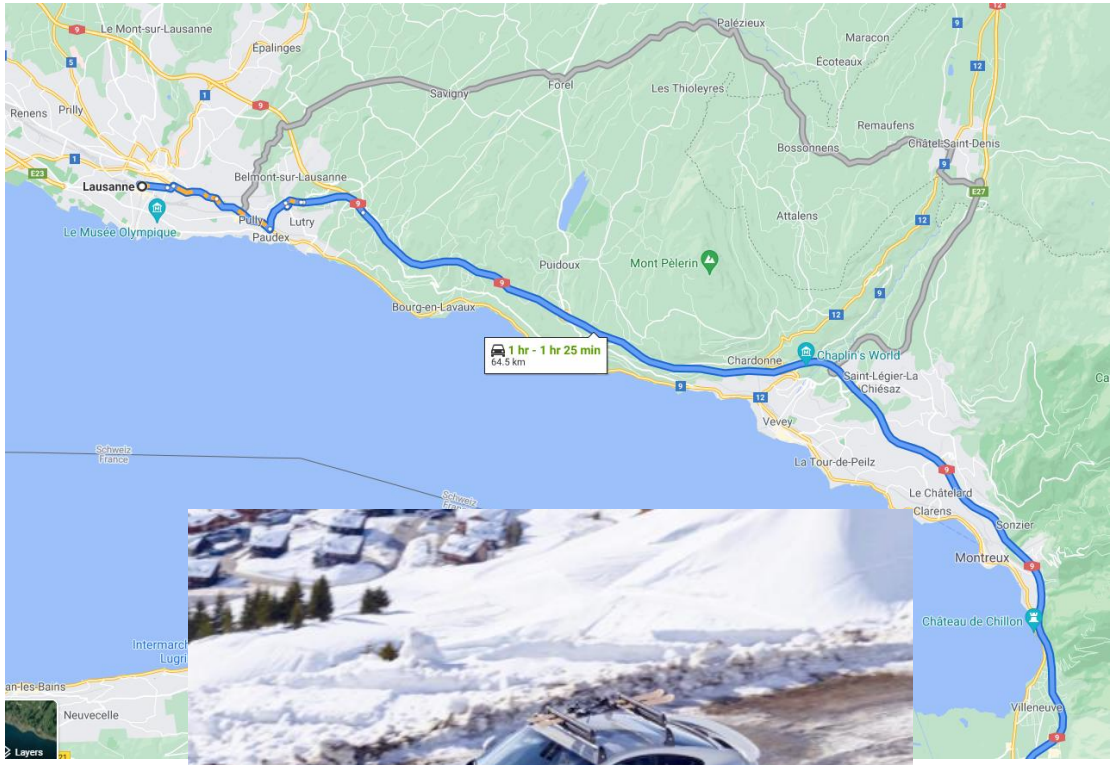
1 ' Walk (Show on the map)

10:01 Le Sépey, Platform 1

R 428 Direction Les Diablerets

10:19 Les Diablerets, Platform 1

Dynamic scheduling (our HLS approach): decide at **runtime** when each operation executes

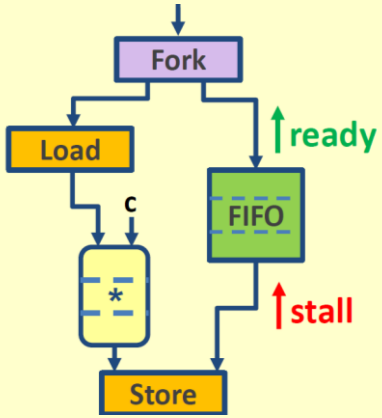


HLS of Dynamically Scheduled Circuits

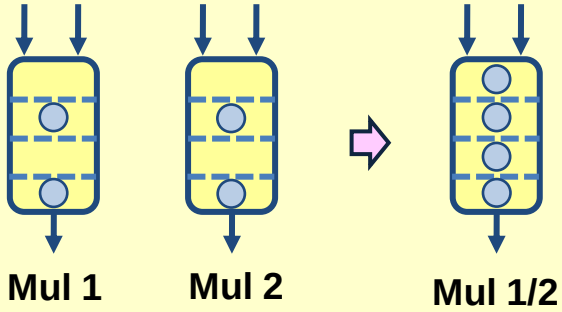
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining



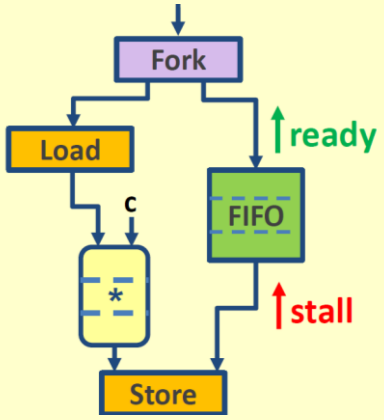
Resource sharing



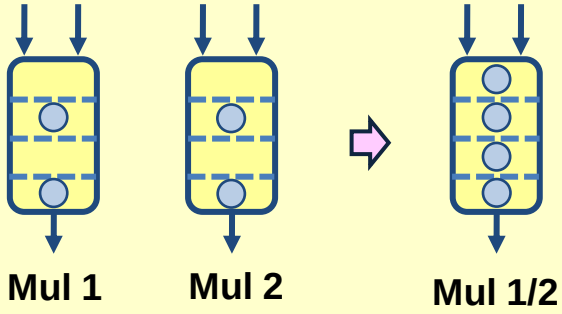
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

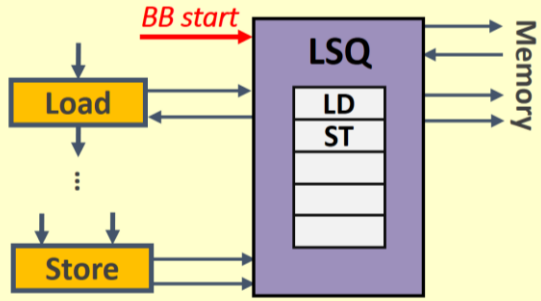


Resource sharing

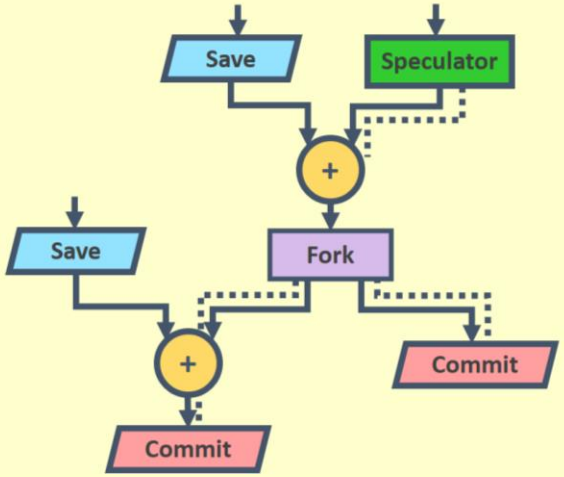


Reaping the benefits of dynamic scheduling

Out-of-order memory

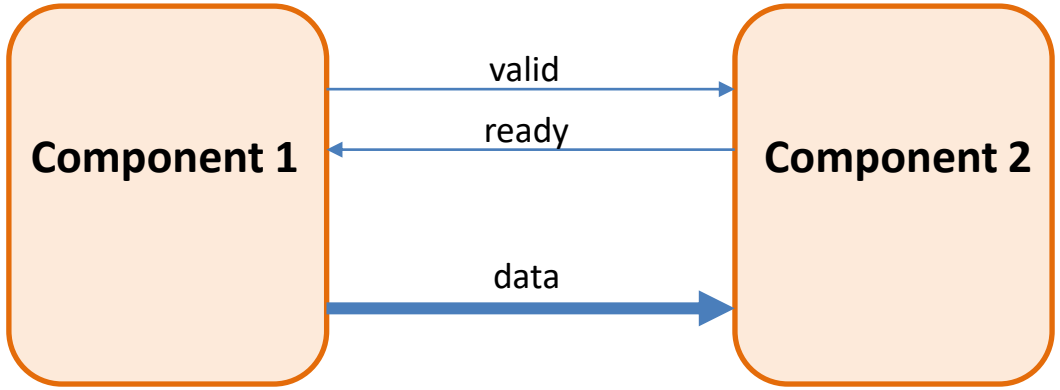


Speculative execution



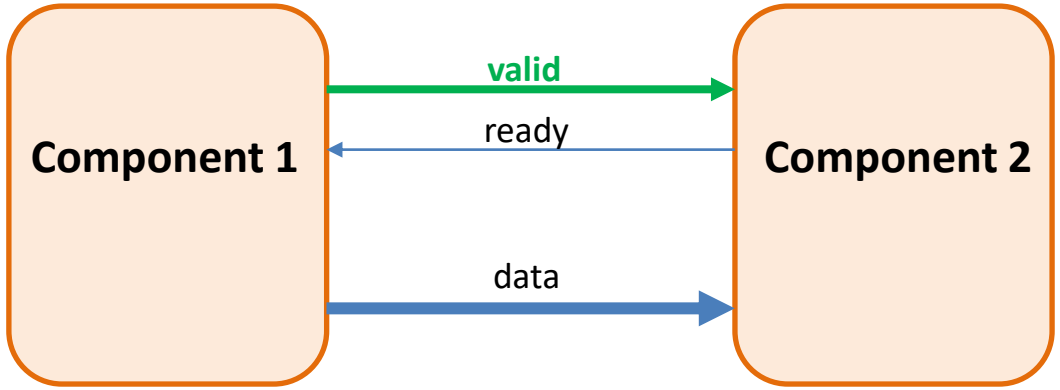
Dataflow Circuits

- We use the **SELF (Synchronous ELastic Flow)** protocol
  - Cortadella et al. Synthesis of synchronous elastic architectures. DAC'06.
- Every component communicates via a pair of handshake signals
- **Make scheduling decisions at runtime**
  - As soon as all conditions for execution are satisfied, an operation starts



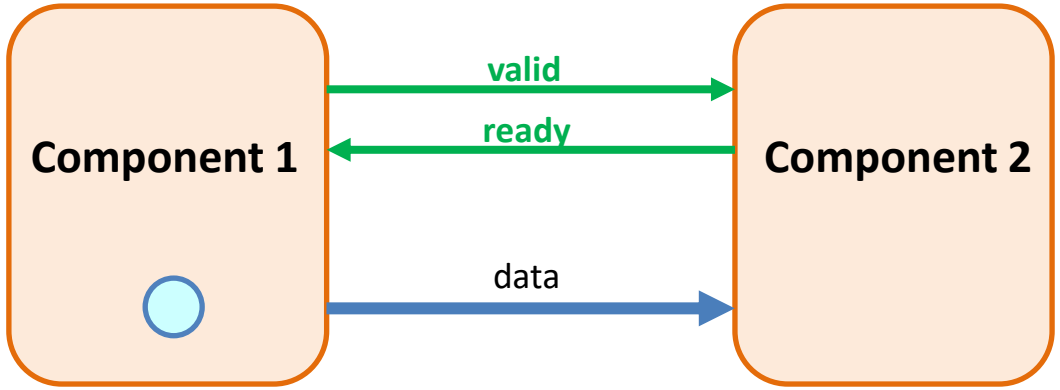
Dataflow Circuits

- We use the **SELF (Synchronous ELastic Flow)** protocol
  - Cortadella et al. Synthesis of synchronous elastic architectures. DAC'06.
- Every component communicates via a pair of handshake signals
- **Make scheduling decisions at runtime**
  - As soon as all conditions for execution are satisfied, an operation starts

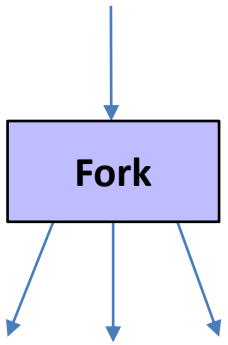


Dataflow Circuits

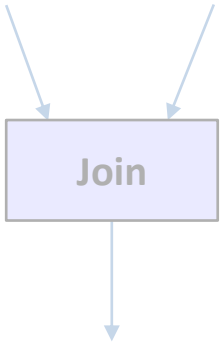
- We use the **SELF (Synchronous ELastic Flow)** protocol
  - Cortadella et al. Synthesis of synchronous elastic architectures. DAC'06.
- Every component communicates via a pair of handshake signals
- **Make scheduling decisions at runtime**
  - As soon as all conditions for execution are satisfied, an operation starts



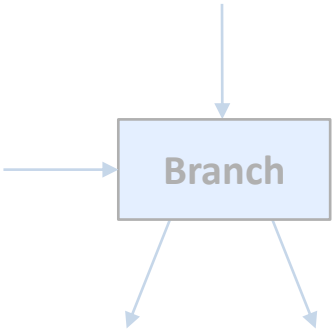
Dataflow Components



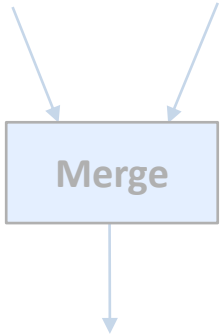
Fork



Join

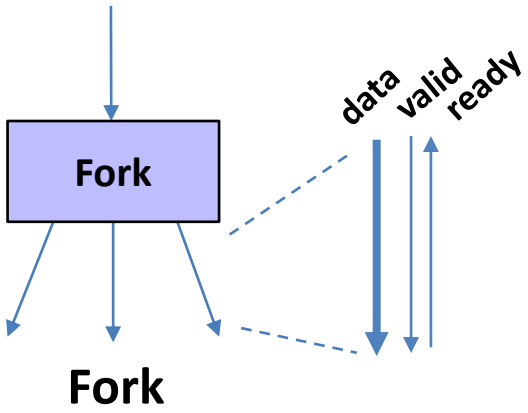


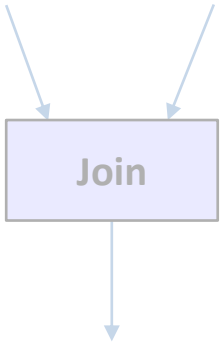
Branch

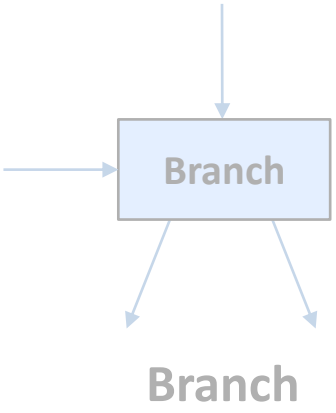


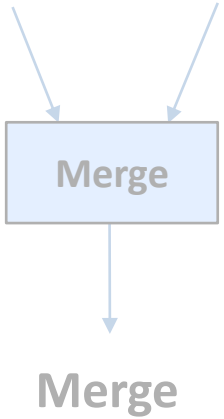
Merge

Dataflow Components

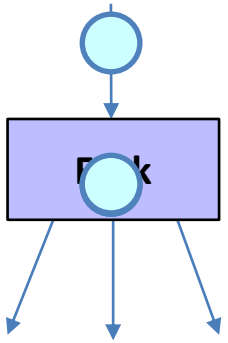




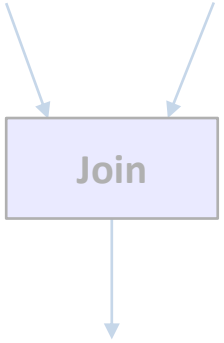




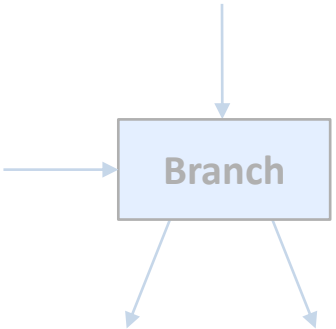
Dataflow Components



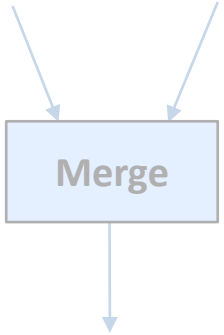
Fork



Join

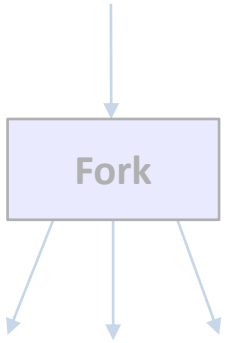


Branch

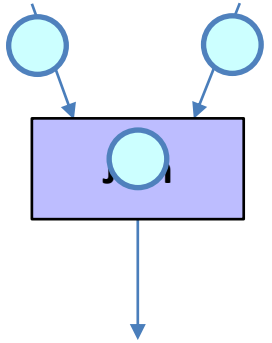


Merge

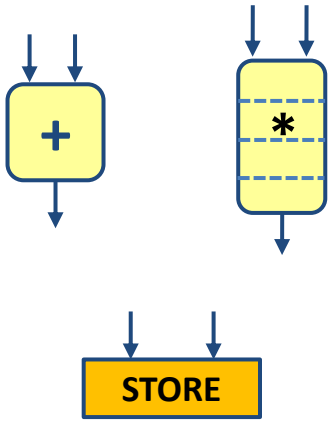
Dataflow Components

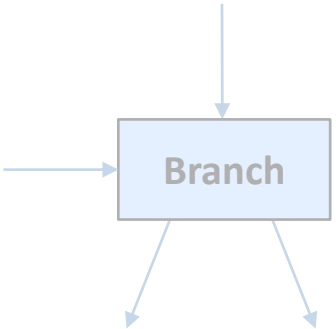


Fork

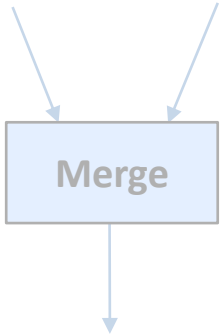


Join



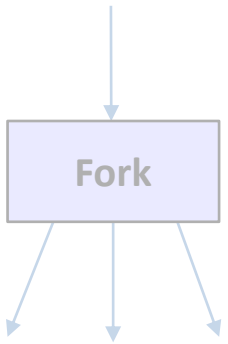


Branch

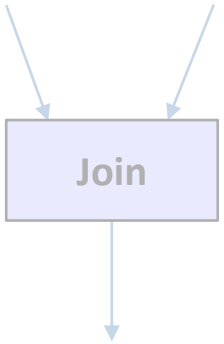


Merge

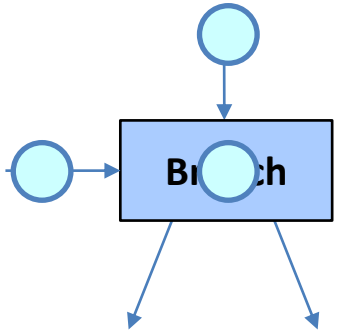
Dataflow Components



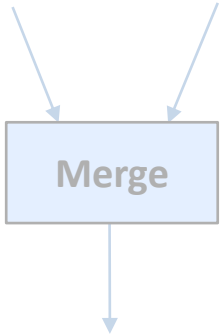
Fork



Join

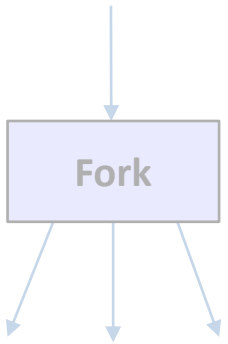


Branch

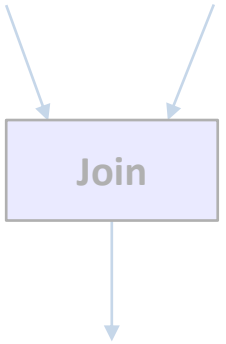


Merge

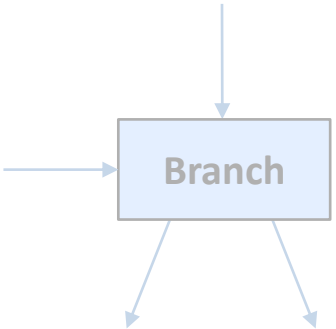
Dataflow Components



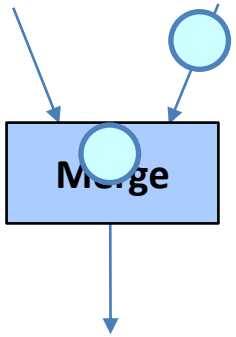
Fork



Join

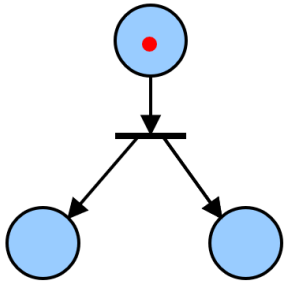


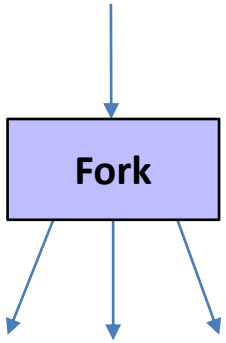
Branch



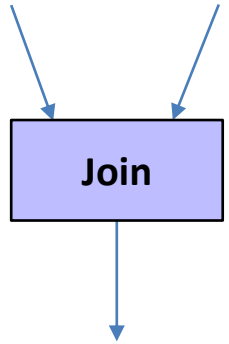
Merge

Dataflow Components

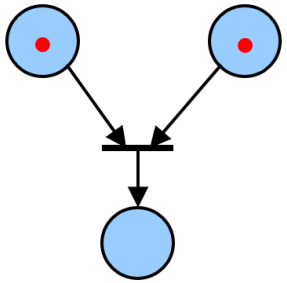


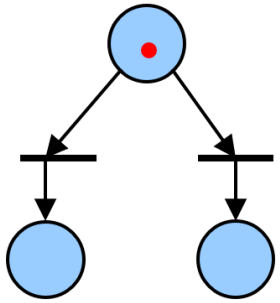


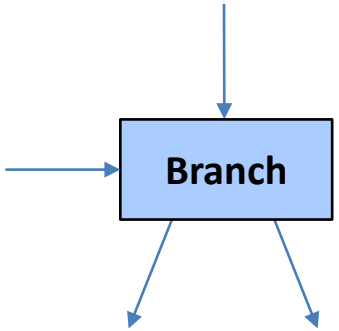
Fork



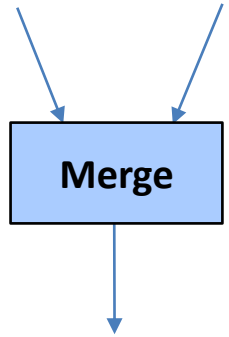
Join



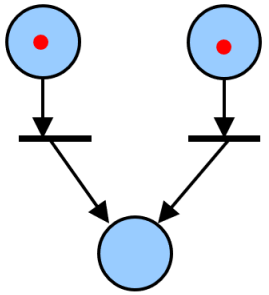




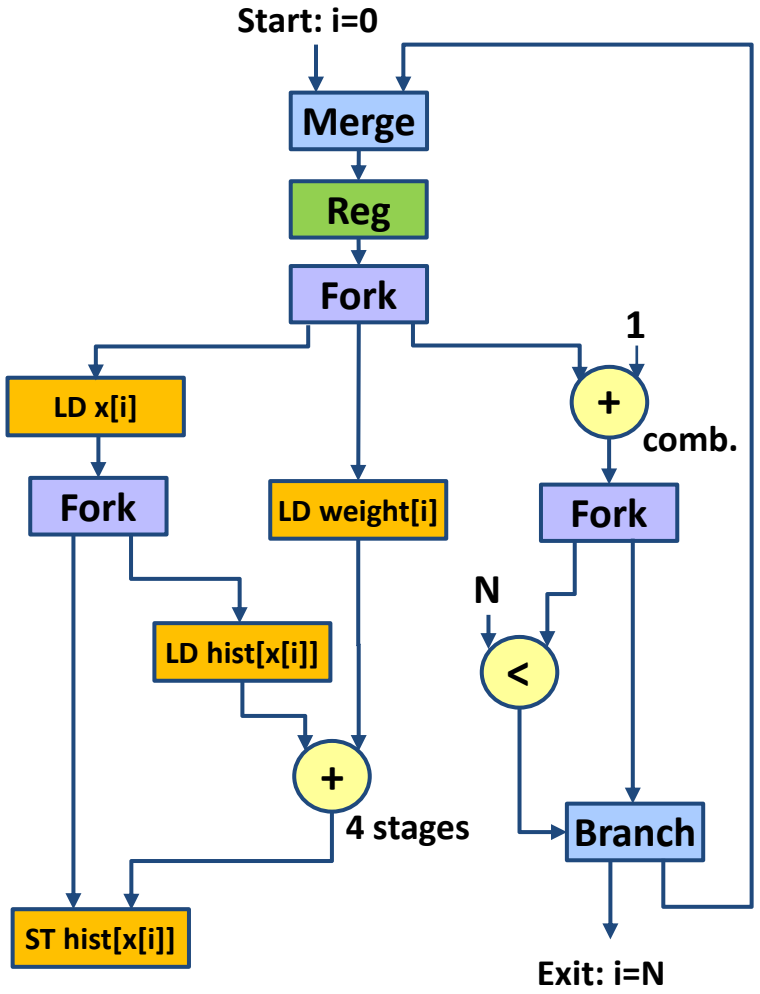
Branch



Merge



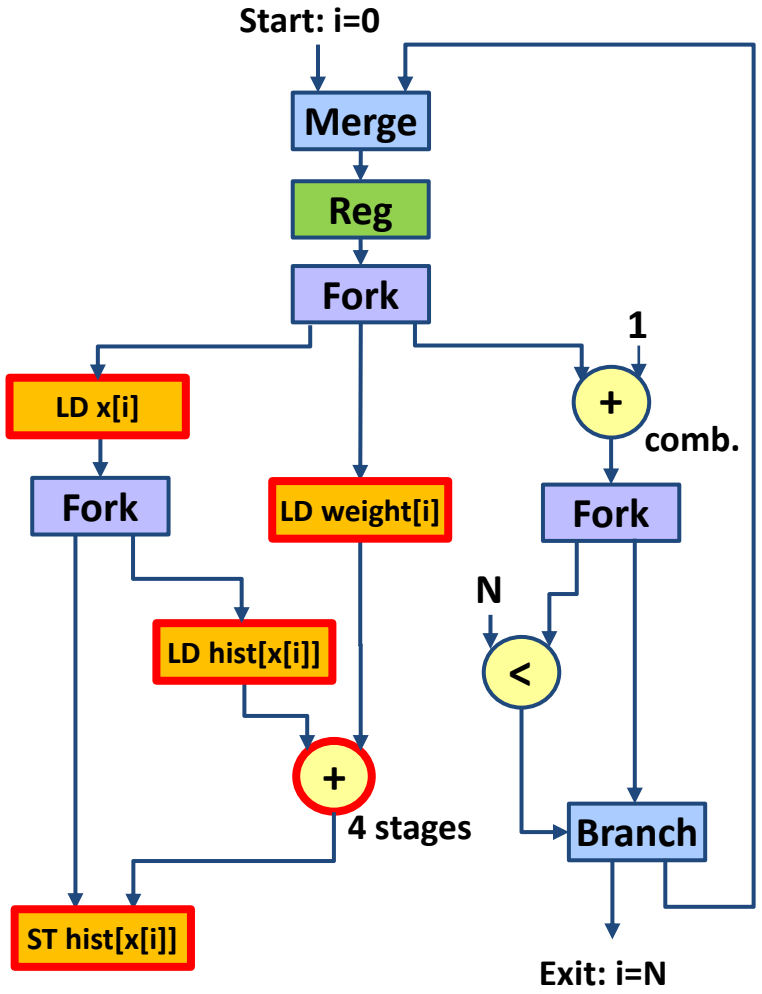
From Program to Dataflow Circuit




```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 **Best Paper Award Nominee**
Josipović, Brisk, and Ienne. From C to Elastic Circuits. Asilomar 2017

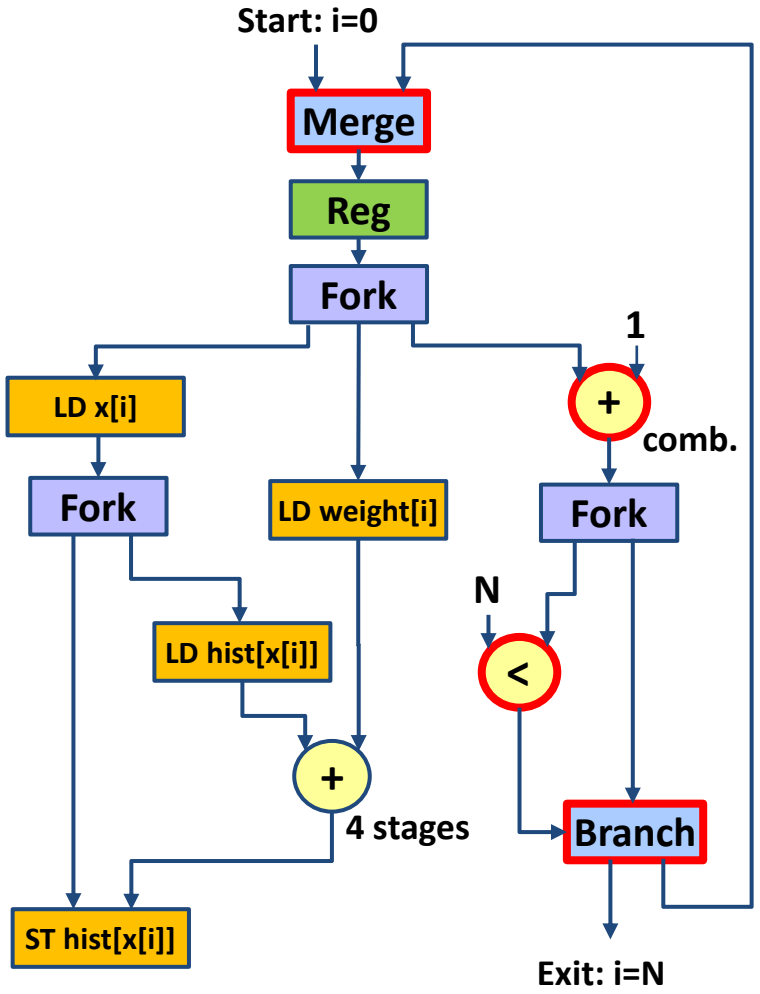
From Program to Dataflow Circuit




```
for (i=0; i<N; i++) {
  hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 **Best Paper Award Nominee**
Josipović, Brisk, and Ienne. From C to Elastic Circuits. Asilomar 2017

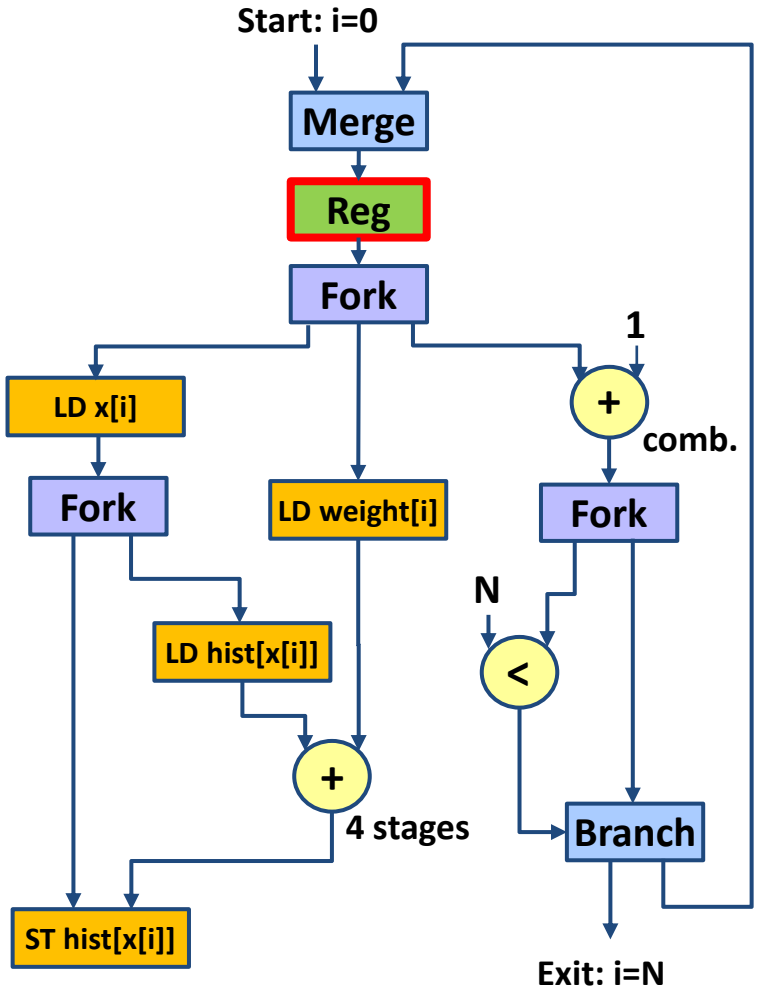
From Program to Dataflow Circuit




```
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 **Best Paper Award Nominee**
Josipović, Brisk, and Ienne. From C to Elastic Circuits. Asilomar 2017

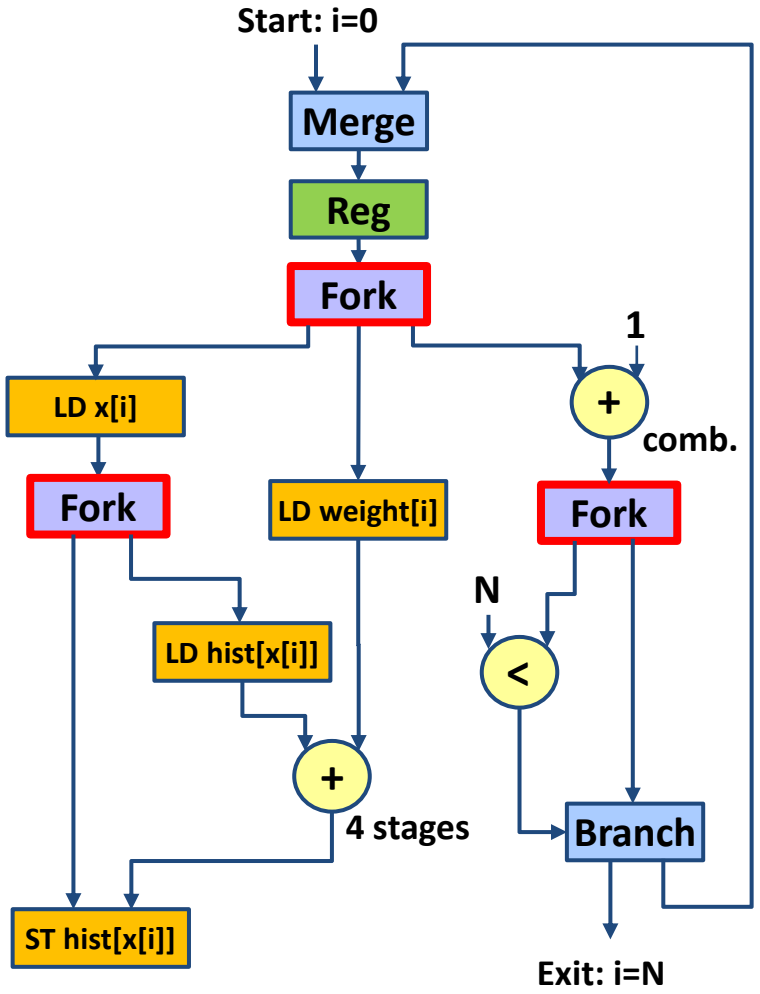
From Program to Dataflow Circuit




```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 **Best Paper Award Nominee**
Josipović, Brisk, and Ienne. From C to Elastic Circuits. Asilomar 2017

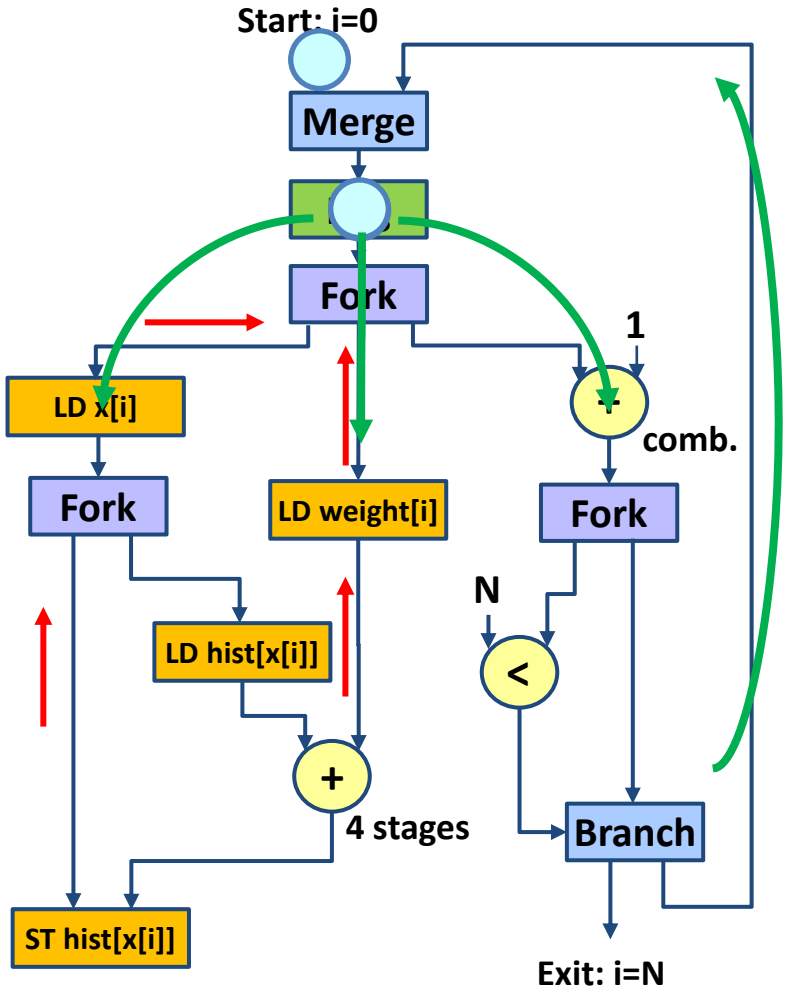
From Program to Dataflow Circuit




```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 **Best Paper Award Nominee**
Josipović, Brisk, and Ienne. From C to Elastic Circuits. Asilomar 2017

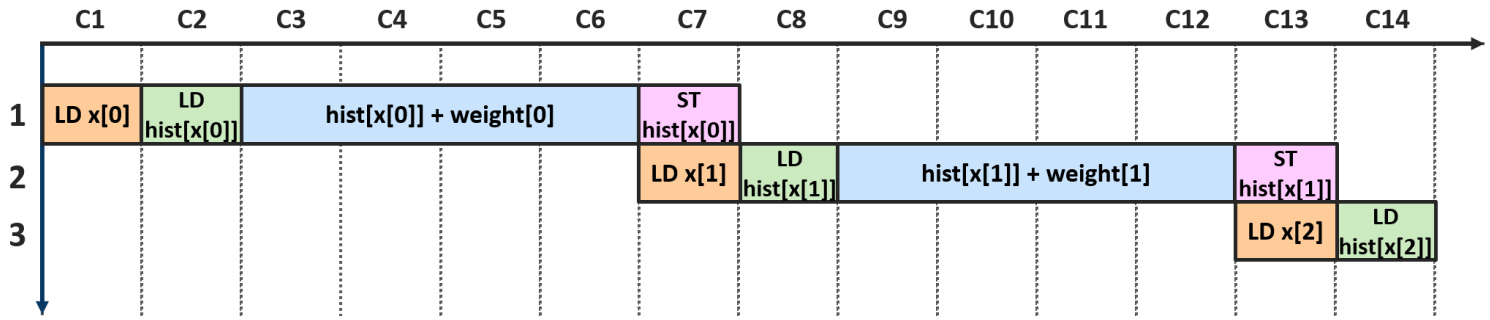
From Program to Dataflow Circuit



Single token on cycle, in-order tokens in noncyclic paths

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 **Best Paper Award Nominee**
Josipović, Brisk, and Ienne. From C to Elastic Circuits. Asilomar 2017

From Program to Dataflow Circuit

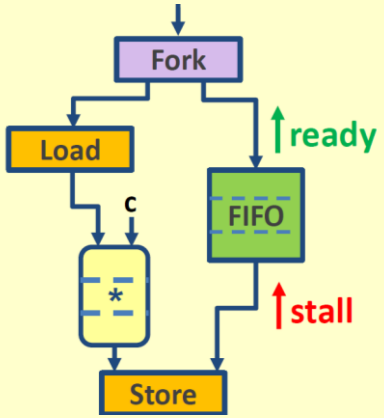


Backpressure from slow paths prevents pipelining

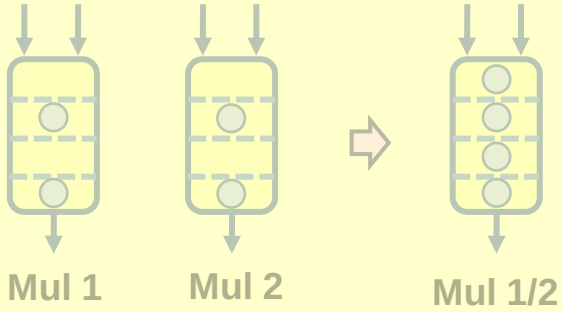
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

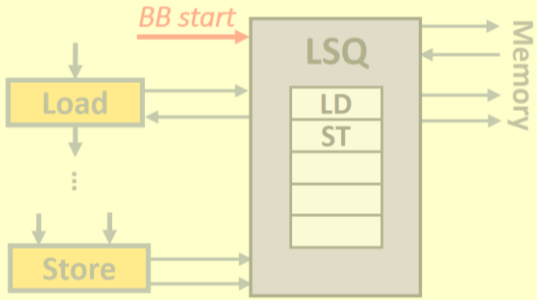


Resource sharing

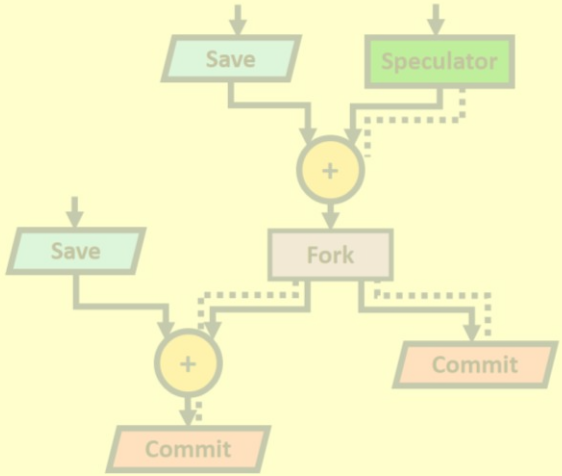


Reaping the benefits of dynamic scheduling

Out-of-order memory

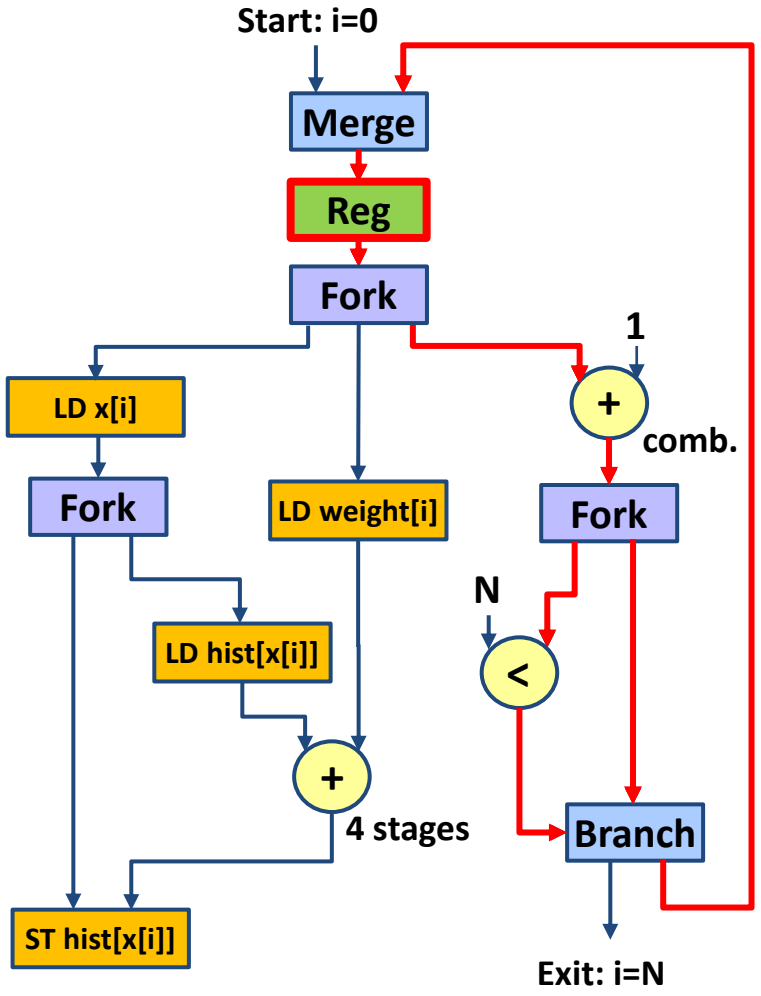


Speculative execution



Inserting Buffers

```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

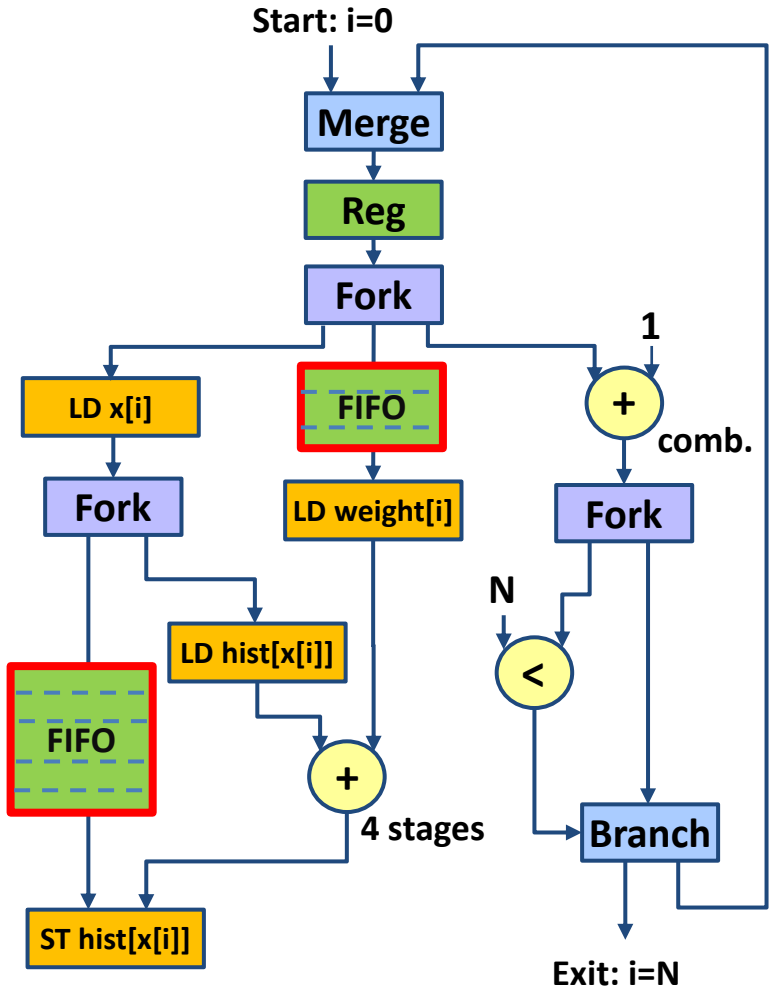


Buffers as registers to break combinational paths

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**
Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022

Inserting Buffers

```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

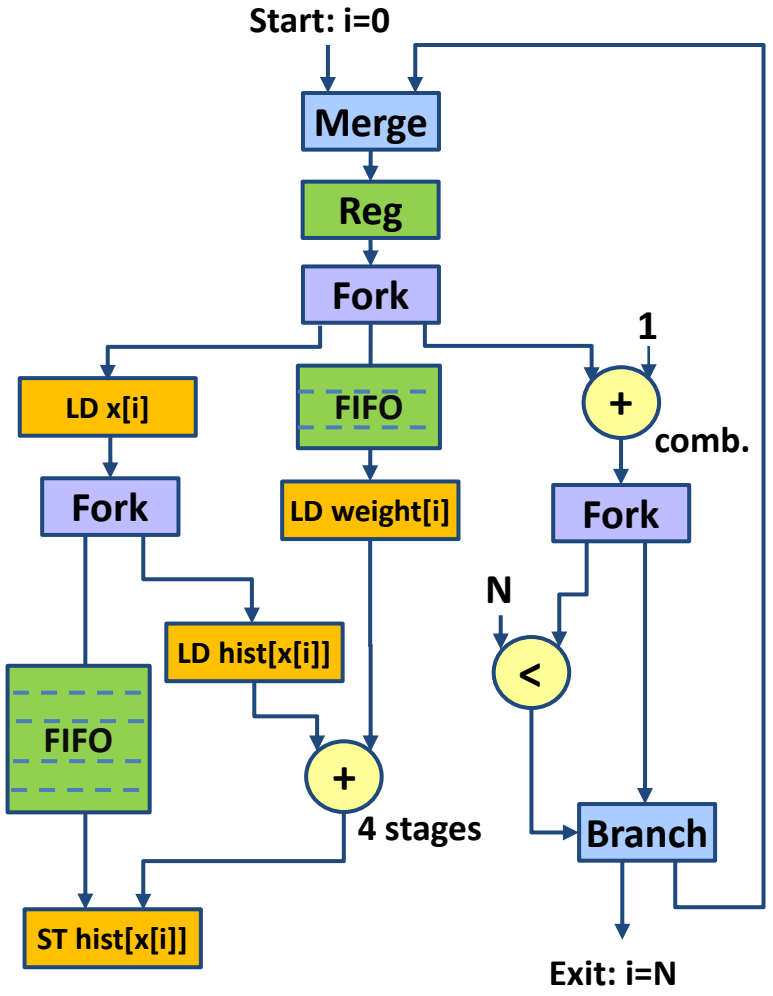


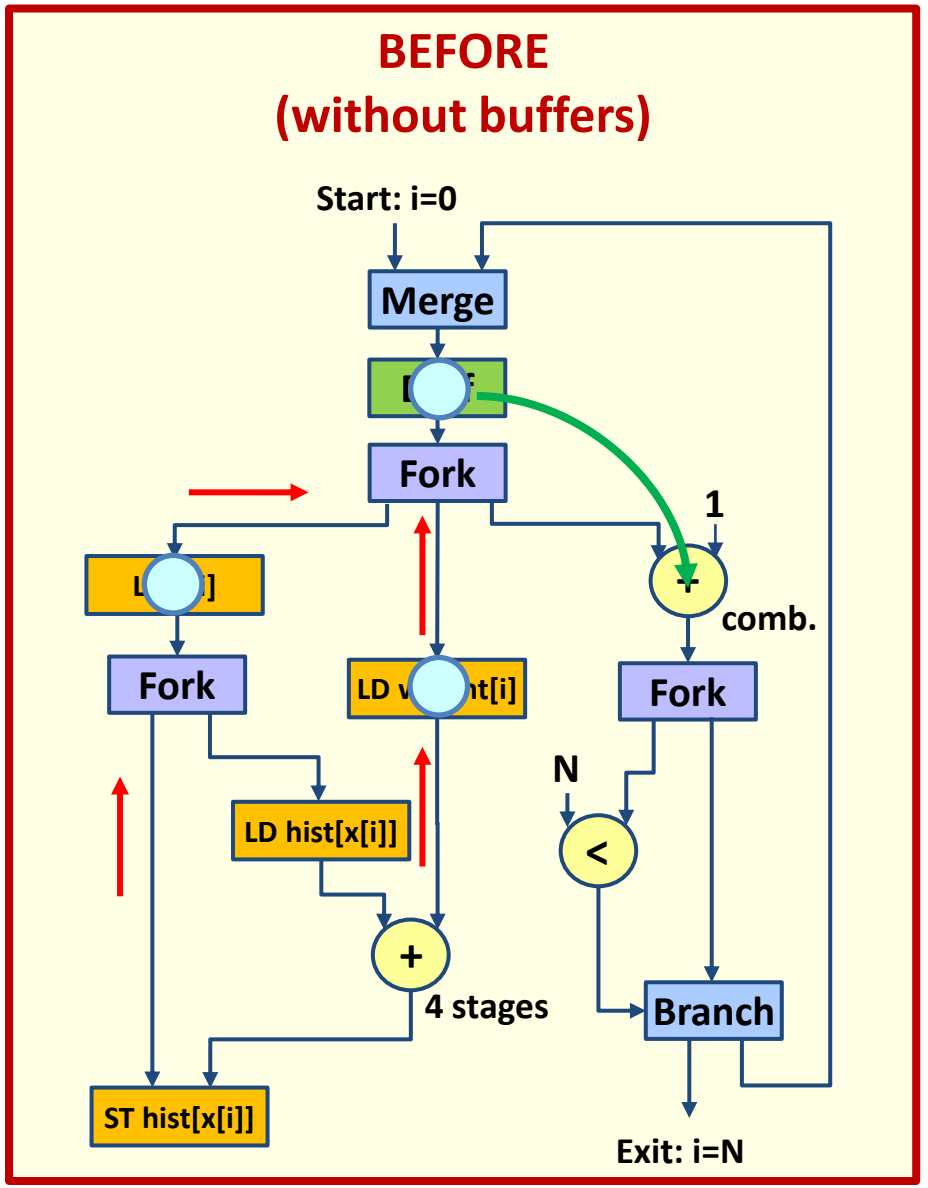
Buffers as FIFOs to regulate throughput

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**
Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022

Inserting Buffers

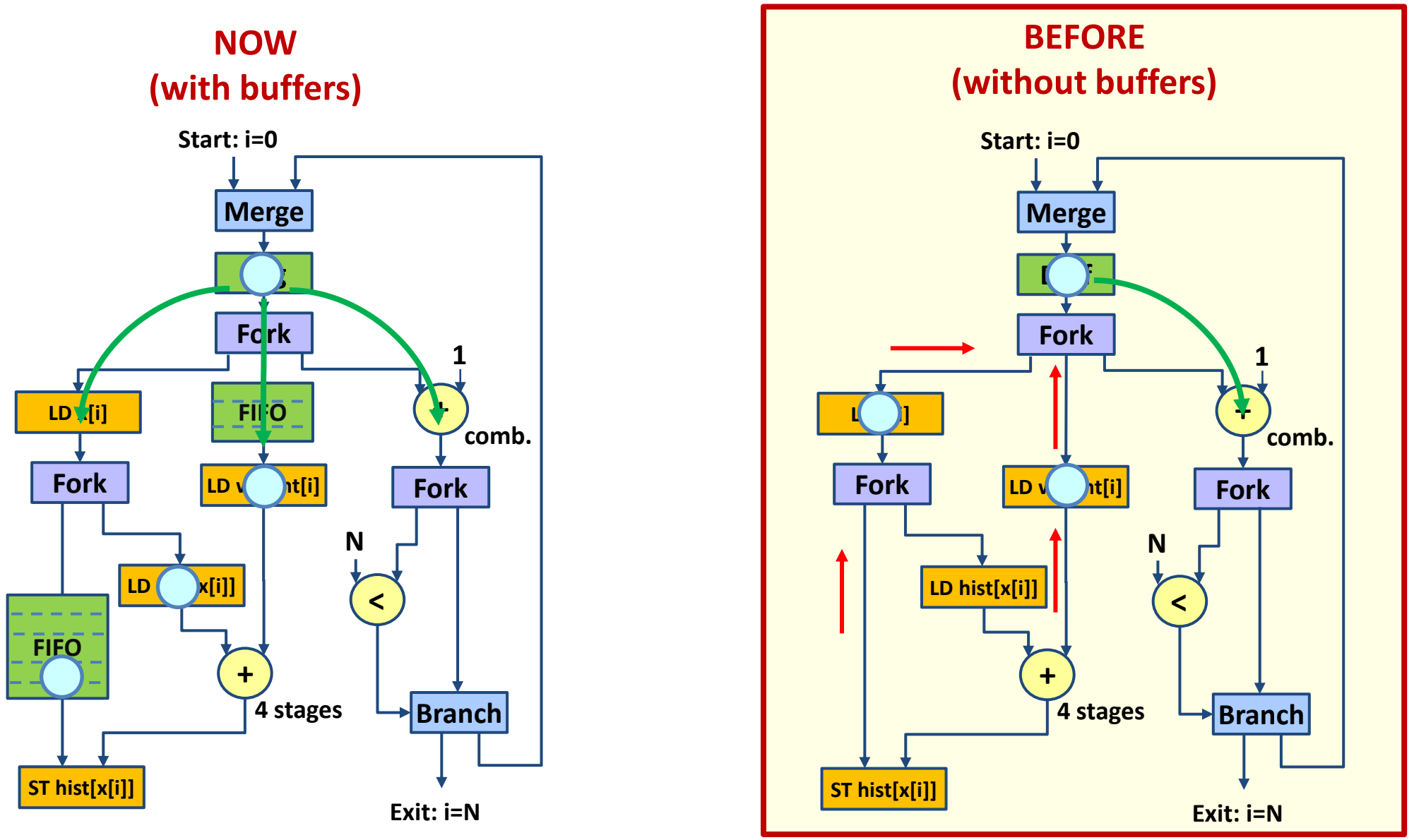
```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```





Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**
Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022

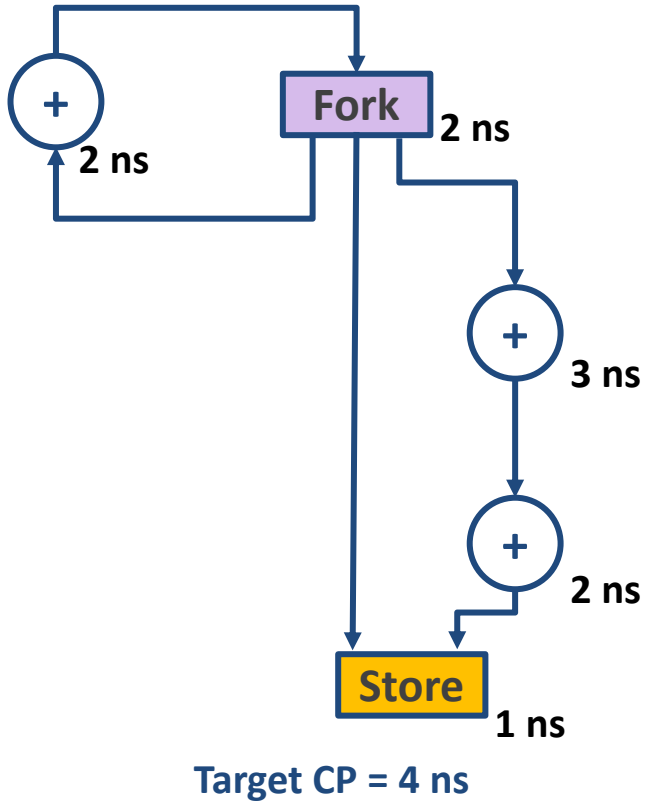
Inserting Buffers



Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**
Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022

Optimizing Performance

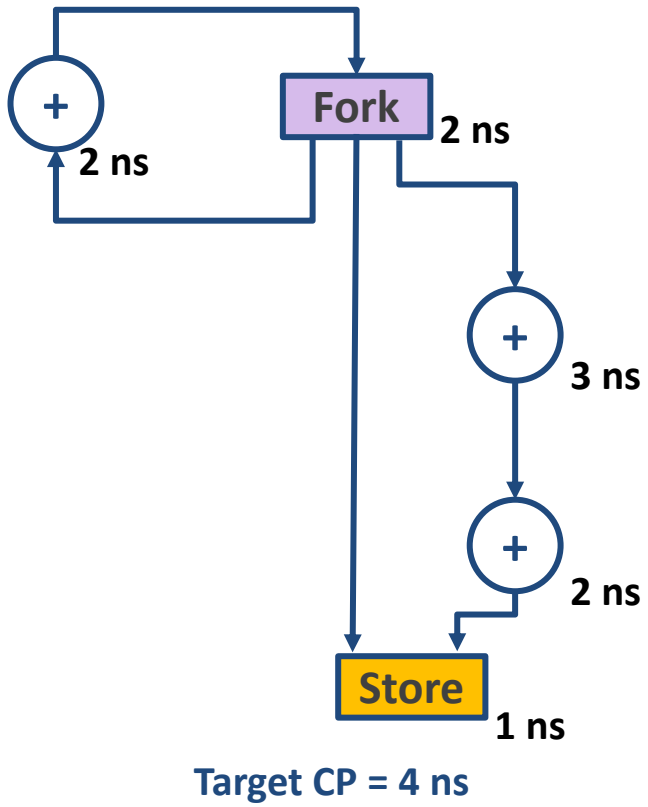
- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Objective: maximize throughput for a target period and minimize buffer slot count

throughput

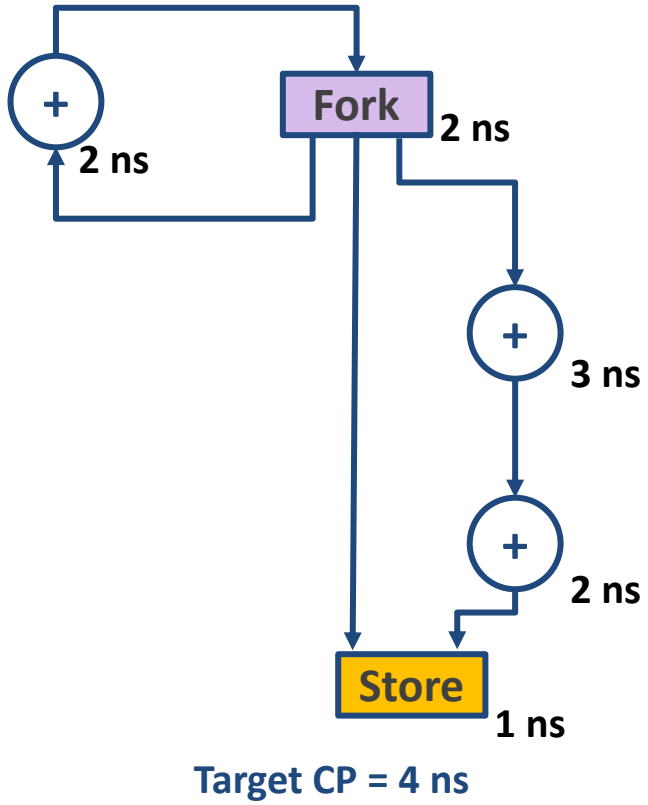
$$max: \Phi - \lambda \cdot \sum_{c} N_c$$

small const. buffer slots

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 4 ns

Objective: maximize throughput for a target period and minimize buffer slot count

$$max:\ \Phi - \lambda \cdot \sum_c N_c$$

target period N-buff

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$

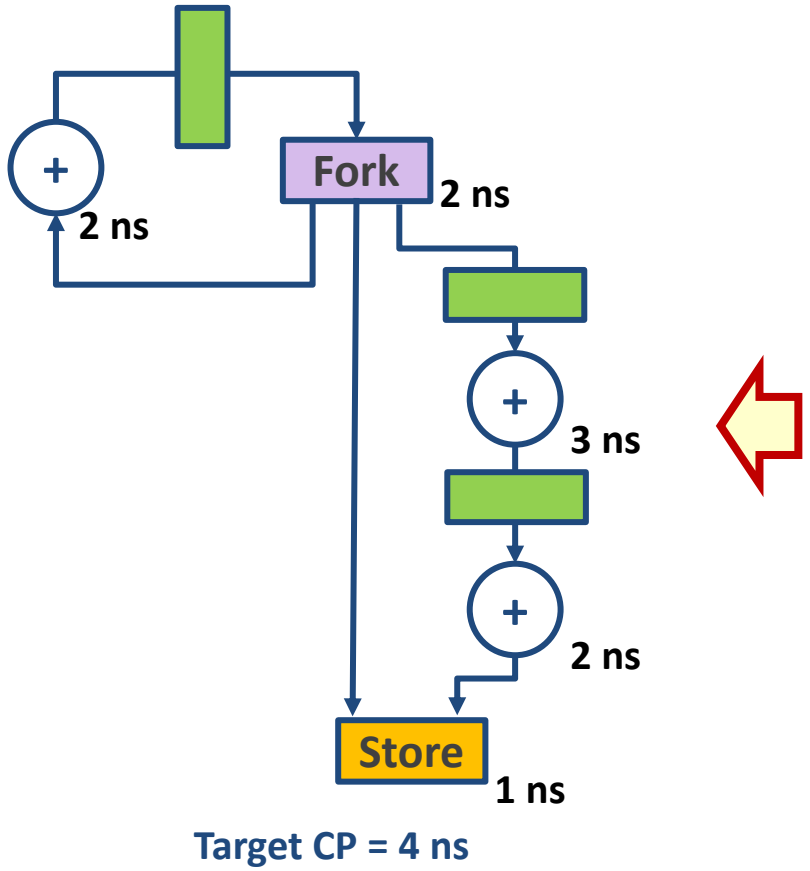
$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

in/out arrival time unit comb. delay

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Objective: maximize throughput for a target period and minimize buffer slot count

$$max:\ \Phi - \lambda \cdot \sum_c N_c$$

target period N-buff

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$

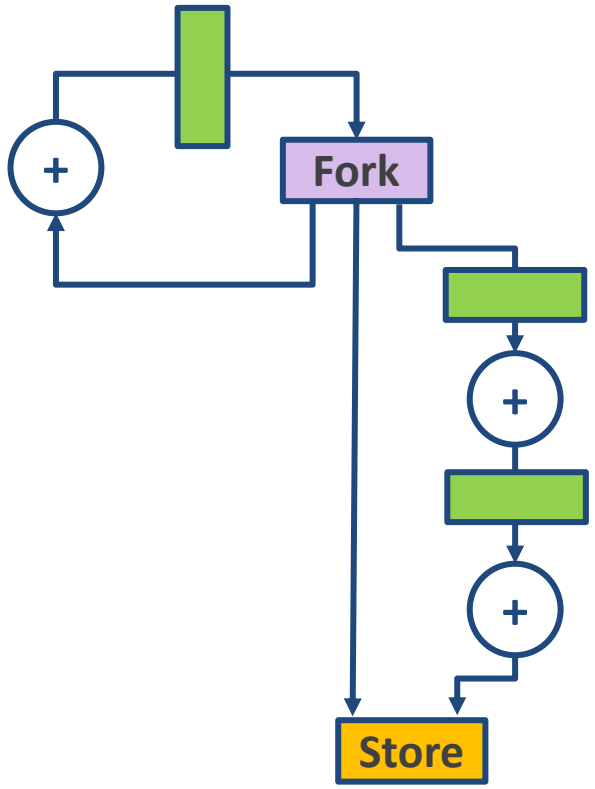
$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

in/out arrival time unit comb. delay

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 4 ns

Objective: maximize throughput for a target period and minimize buffer slot count

$$max:\ \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$

$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

token retiming

$$\theta_c = B_b + r_v - r_u$$

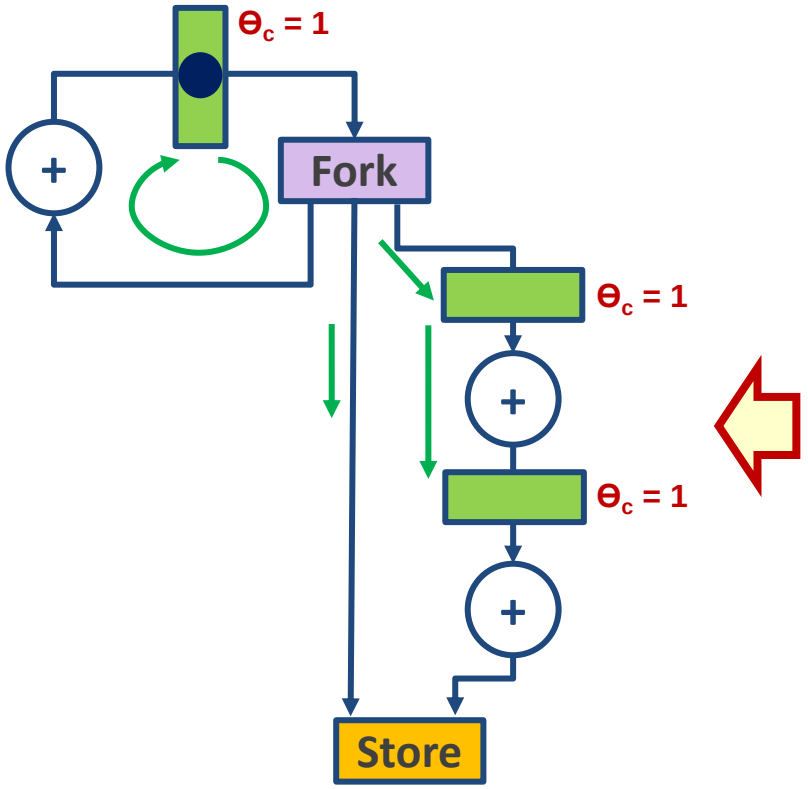
$$\theta_c \geq \Phi + R_c - 1$$

channel occupancy

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 4 ns
Throughput: Φ = 1

Objective: maximize throughput for a target period and minimize buffer slot count

$$max: \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$

$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

token retiming

$$\theta_c = B_b + r_v - r_u$$

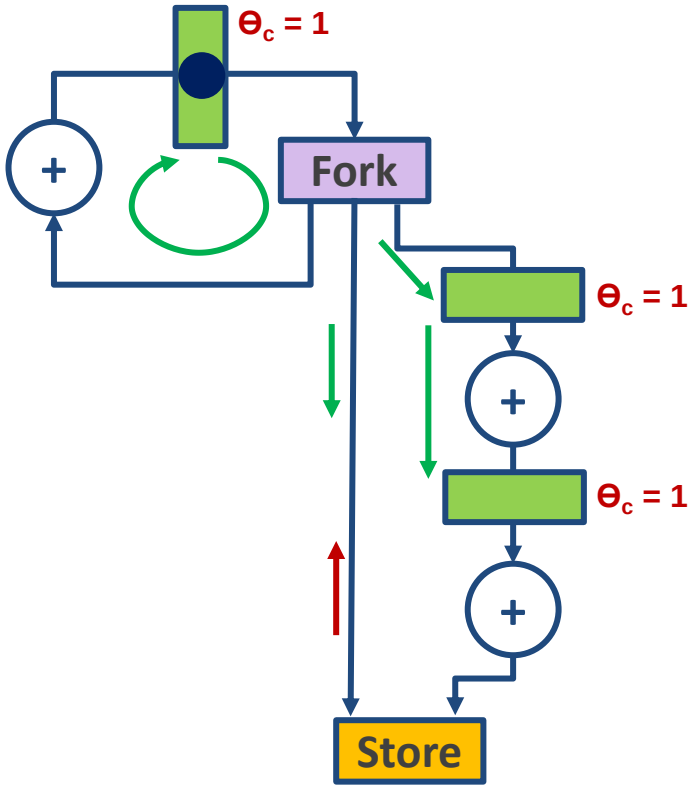
$$\theta_c \geq \Phi + R_c - 1$$

channel occupancy

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 4 ns
Throughput: Φ = 1

Objective: maximize throughput for a target period and minimize buffer slot count

$$max: \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$

$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

token retiming

$$\theta_c = B_b + r_v - r_u$$

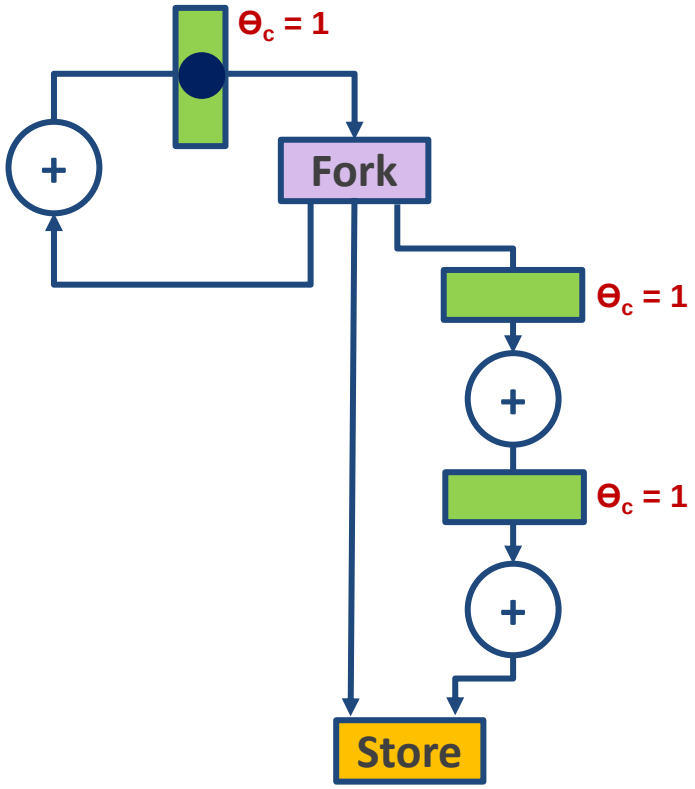
$$\theta_c \geq \Phi + R_c - 1$$

channel occupancy

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 4 ns
Throughput: Φ = 1

Objective: maximize throughput for a target period and minimize buffer slot count

$$max: \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$
$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

$$\theta_c = B_b + r_v - r_u$$
$$\theta_c \geq \Phi + R_c - 1$$

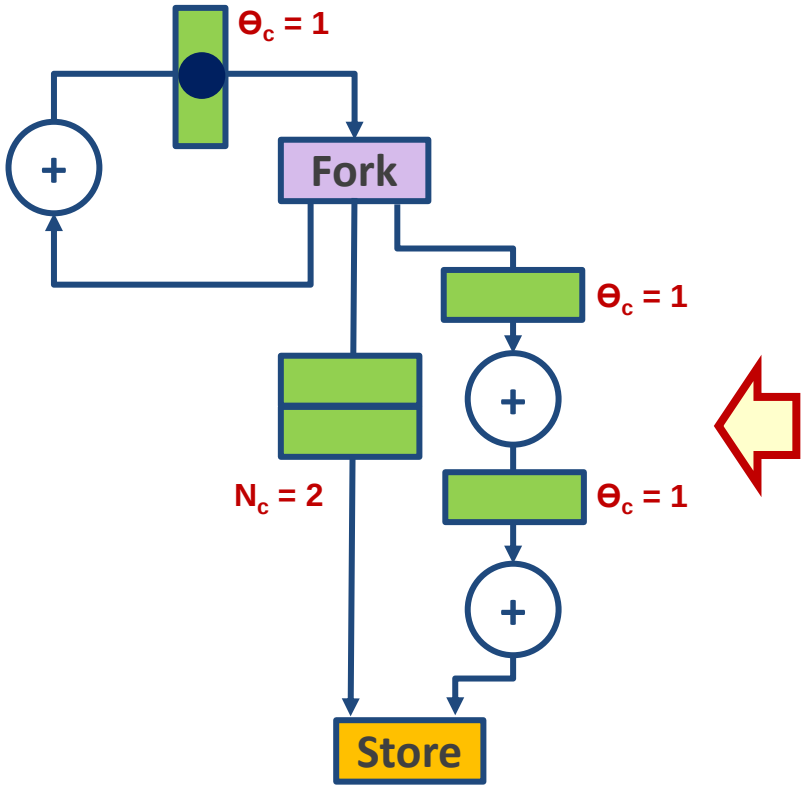
Buffer sizing: add buffer slots to avoid backpressure and maximize throughput

channel emptiness
$N_c \geq \theta_c + \theta_c^{\circ}$
buffer slots

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 4 ns
Throughput: Φ = 1

Objective: maximize throughput for a target period and minimize buffer slot count

$$max: \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$
$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

$$\theta_c = B_b + r_v - r_u$$
$$\theta_c \geq \Phi + R_c - 1$$

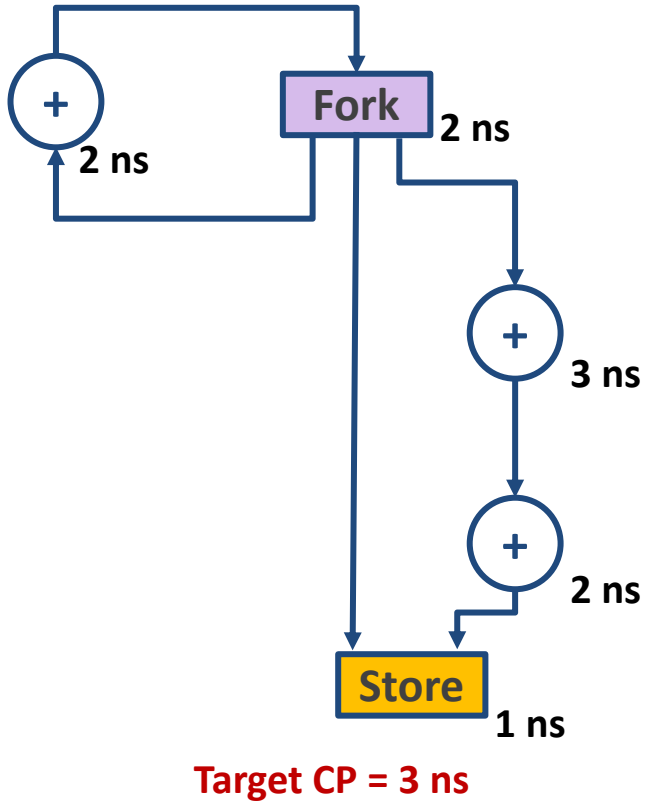
Buffer sizing: add buffer slots to avoid backpressure and maximize throughput

channel emptiness
$$N_c \geq \theta_c + \theta_c^\circ$$
buffer slots

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 3 ns

Objective: maximize throughput for a target period and minimize buffer slot count

$$max: \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$
$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

$$\theta_c = B_b + r_v - r_u$$
$$\theta_c \geq \Phi + R_c - 1$$

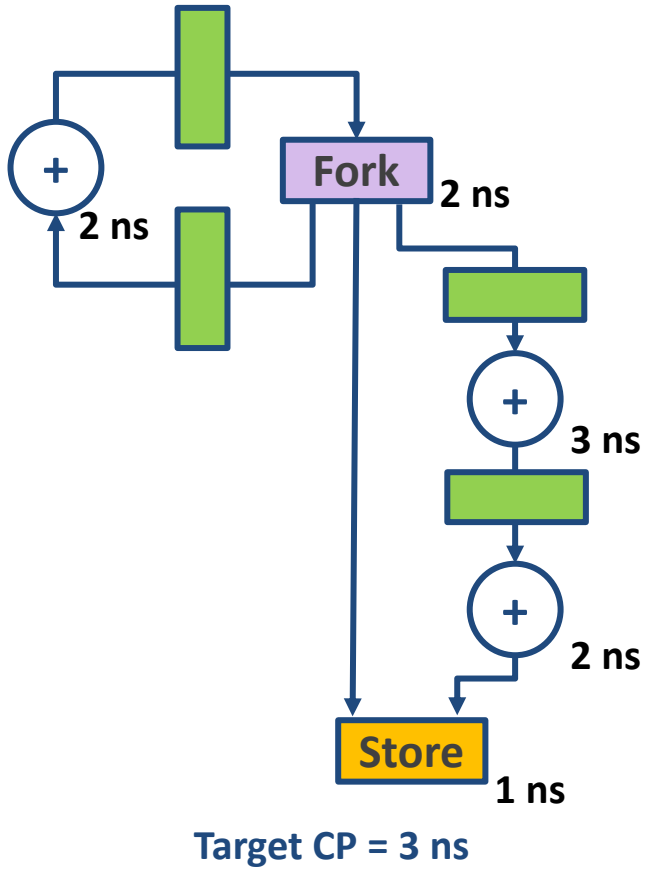
Buffer sizing: add buffer slots to avoid backpressure and maximize throughput

$$N_c \geq \theta_c + \theta_c^{\circ}$$

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 3 ns

Objective: maximize throughput for a target period and minimize buffer slot count

$$max: \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$
$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

$$\Theta_c = B_b + r_v - r_u$$
$$\Theta_c \geq \Phi + R_c - 1$$

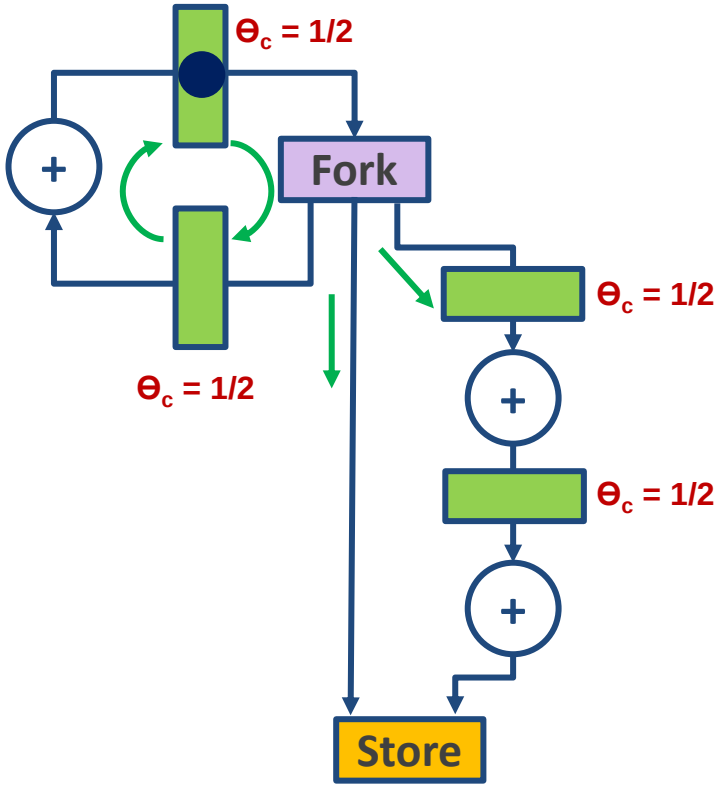
Buffer sizing: add buffer slots to avoid backpressure and maximize throughput

$$N_c \geq \Theta_c + \Theta_c^{\circ}$$

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 3 ns
Throughput: Φ = 1/2

Objective: maximize throughput for a target period and minimize buffer slot count

$$max: \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$
$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

$$\Theta_c = B_b + r_v - r_u$$
$$\Theta_c \geq \Phi + R_c - 1$$

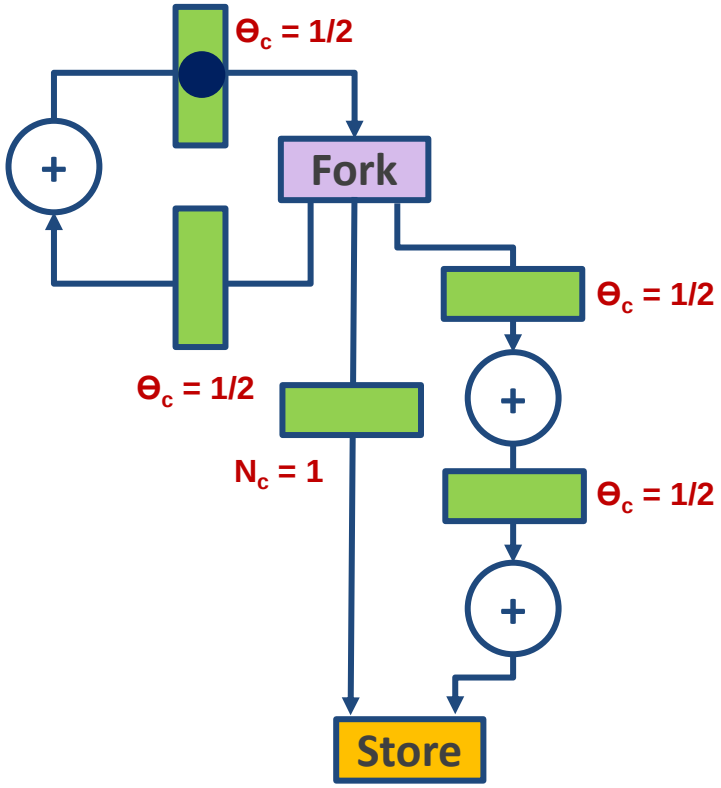
Buffer sizing: add buffer slots to avoid backpressure and maximize throughput

$$N_c \geq \Theta_c + \Theta_c^{\circ}$$

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Optimizing Performance

- Model each program loop as a **concurrent, choice-free Petri net (= marked graph)**
  - **MILP model** to optimize throughput of the choice-free Petri net under a clock period constraint



Target CP = 3 ns
Throughput: Φ = 1/2

Objective: maximize throughput for a target period and minimize buffer slot count

$$max: \Phi - \lambda \cdot \sum_c N_c$$

Path constraints: add buffers to meet target clock period

$$t_c^{out} \geq t_c^{in} - CP \cdot R_c$$
$$CP \geq t_{c_2}^{in} \geq t_{c_1}^{out} + D_u$$

Throughput constraints: compute average number of tokens in a channel

$$\Theta_c = B_b + r_v - r_u$$
$$\Theta_c \geq \Phi + R_c - 1$$

Buffer sizing: add buffer slots to avoid backpressure and maximize throughput

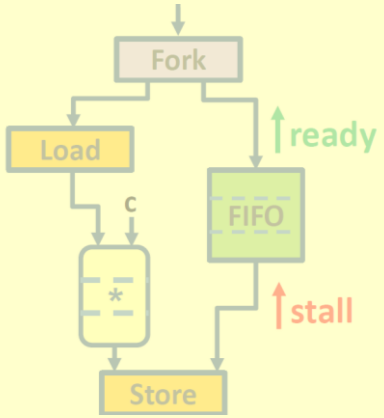
$$N_c \geq \Theta_c + \Theta_c^{\circ}$$

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

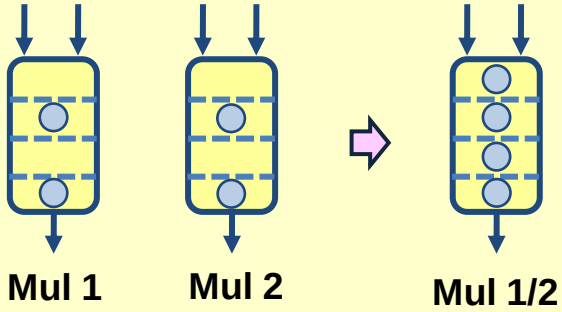
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

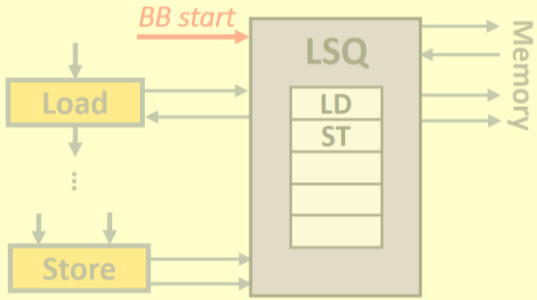


Resource sharing

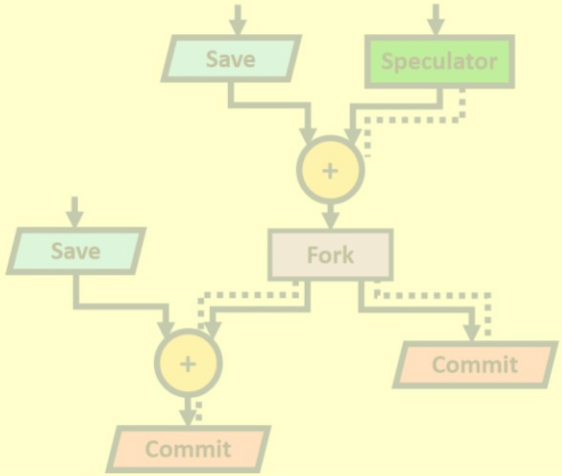


Reaping the benefits of dynamic scheduling

Out-of-order memory



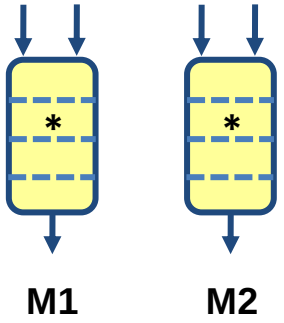
Speculative execution



Saving Resources through Sharing

- Static HLS: share units between operations which execute in **different clock cycles**
- Dynamic HLS: share units based on their **average utilization** with tokens

```
for (i = 0; i < N; i++) {
    a[i] = a[i]*x;
    b[i] = b[i]*y;
}
```

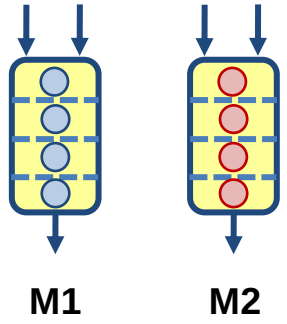


Josipović, Marmet, Guerrieri, and Ienne. Resource Sharing in Dataflow Circuits. FCCM 2022. **Best Paper Award Nominee**

Saving Resources through Sharing

- Static HLS: share units between operations which execute in **different clock cycles**
- Dynamic HLS: share units based on their **average utilization** with tokens

```
for (i = 0; i < N; i++) {
    a[i] = a[i]*x;
    b[i] = b[i]*y;
}
```



Units fully utilized (high throughput)

Sharing not possible without damaging throughput

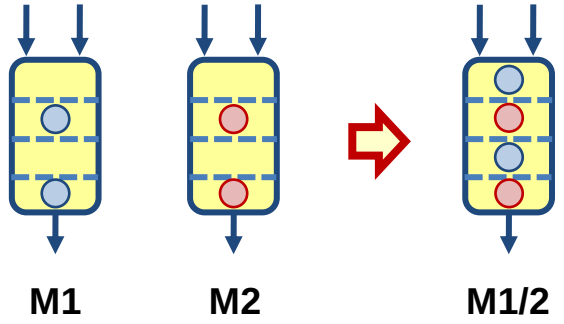
Use choice-free Petri net model to decide what to share

Josipović, Marmet, Guerrieri, and Ienne. Resource Sharing in Dataflow Circuits. FCCM 2022. **Best Paper Award Nominee**

Saving Resources through Sharing

- Static HLS: share units between operations which execute in **different clock cycles**
- Dynamic HLS: share units based on their **average utilization** with tokens

```
for (i = 0; i < N; i++) {
    a[i] = a[i]*x;
    b[i] = b[i]*y;
}
```



Sharing possible without damaging throughput

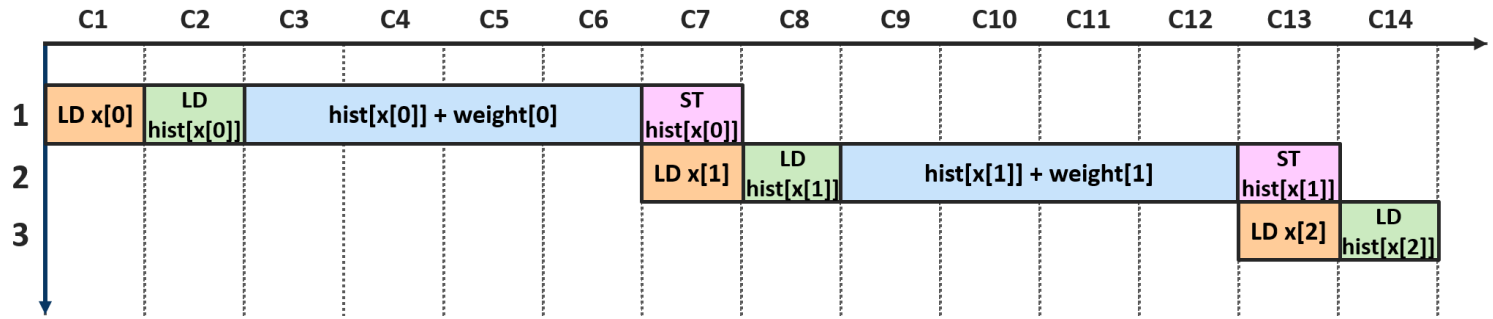
Units underutilized (low throughput)

Use choice-free Petri net model to decide what to share

Josipović, Marmet, Guerrieri, and Ienne. Resource Sharing in Dataflow Circuits. FCCM 2022. **Best Paper Award Nominee**

Inserting Buffers

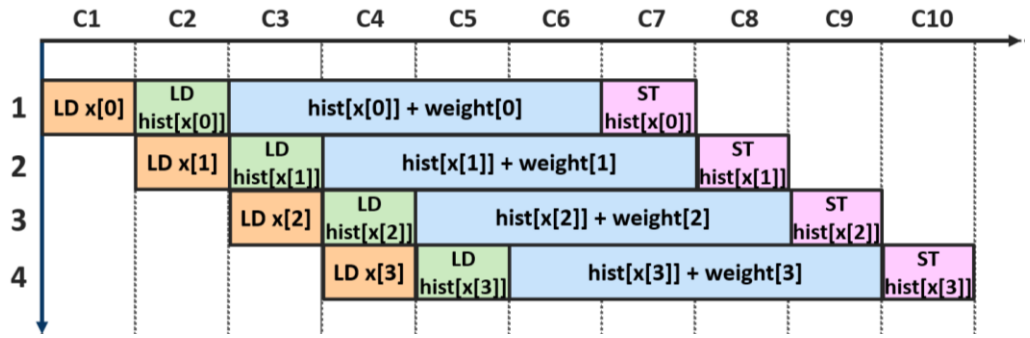
```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```



Backpressure from slow paths prevents pipelining

Inserting Buffers

```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```



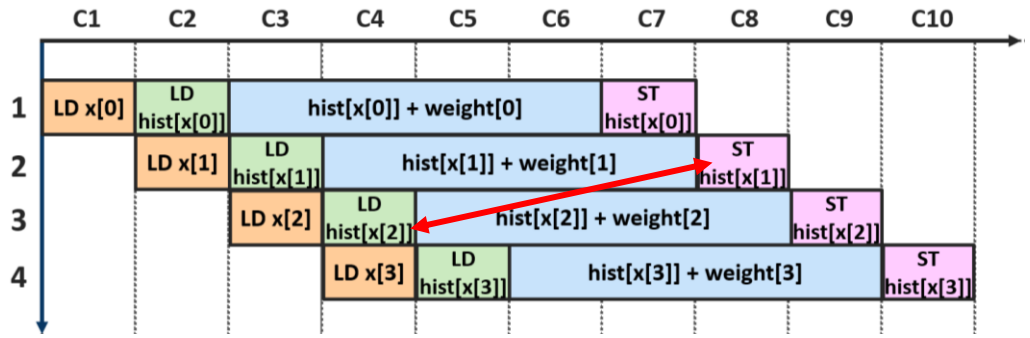
Buffers for high throughput

Inserting Buffers

```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

```
1: x[0]=5 → ld hist[5]; st hist[5];
2: x[1]=4 → ld hist[4]; st hist[4];
3: x[2]=4 → ld hist[4]; st hist[4];
```

RAW dependency



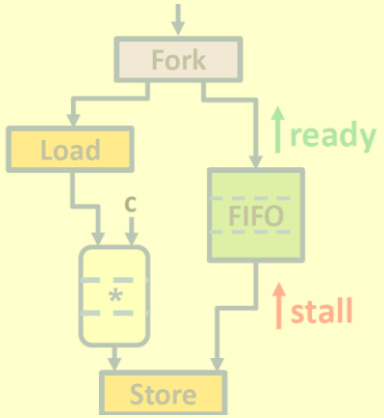
RAW dependency not honored!

What about memory?

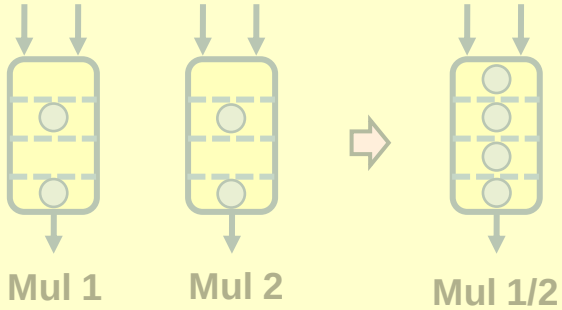
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

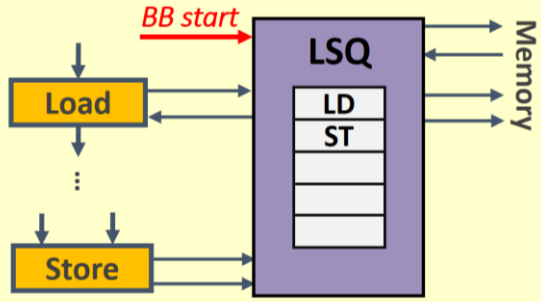


Resource sharing

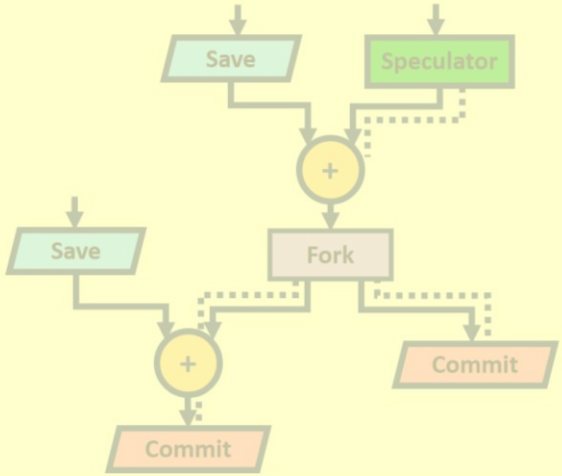


Reaping the benefits of dynamic scheduling

Out-of-order memory

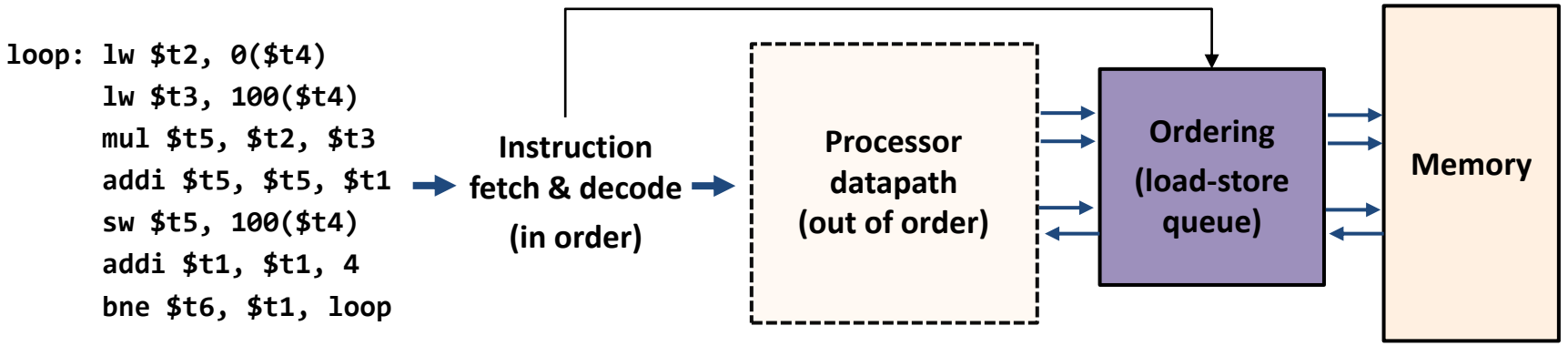


Speculative execution



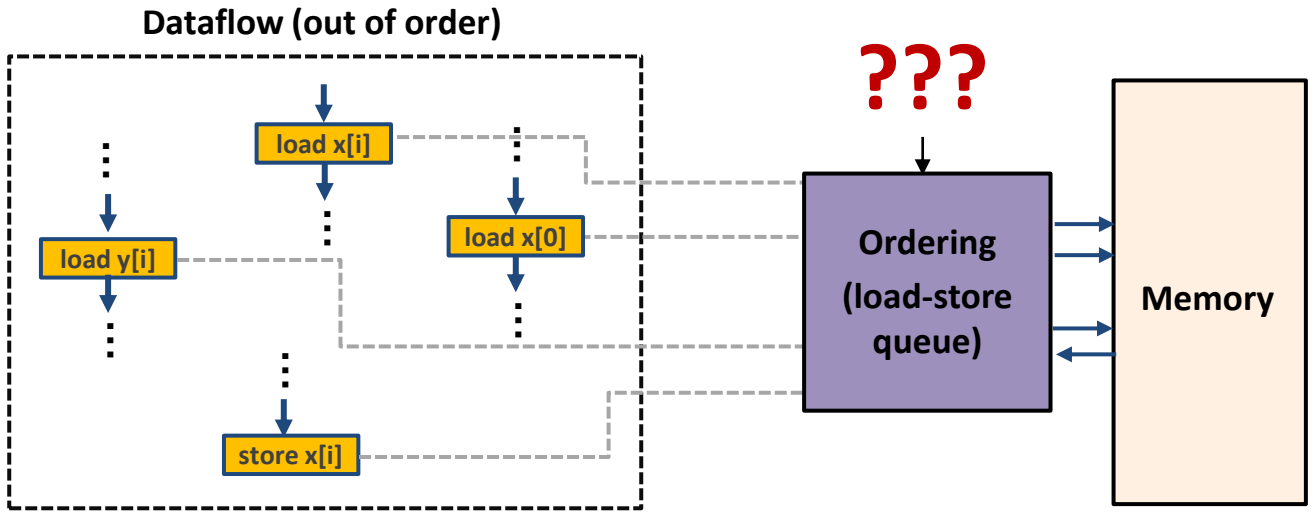
We Need a Load-Store Queue (LSQ)!

- Traditional processor LSQs allocate memory instructions **in program order**



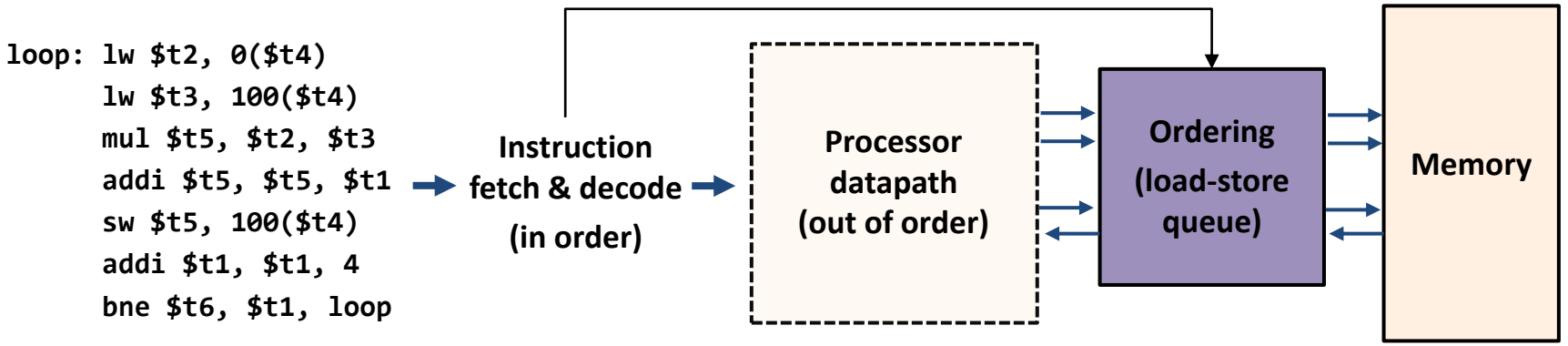
- Dataflow circuits have **no notion of program order**

How to supply program order to the LSQ?



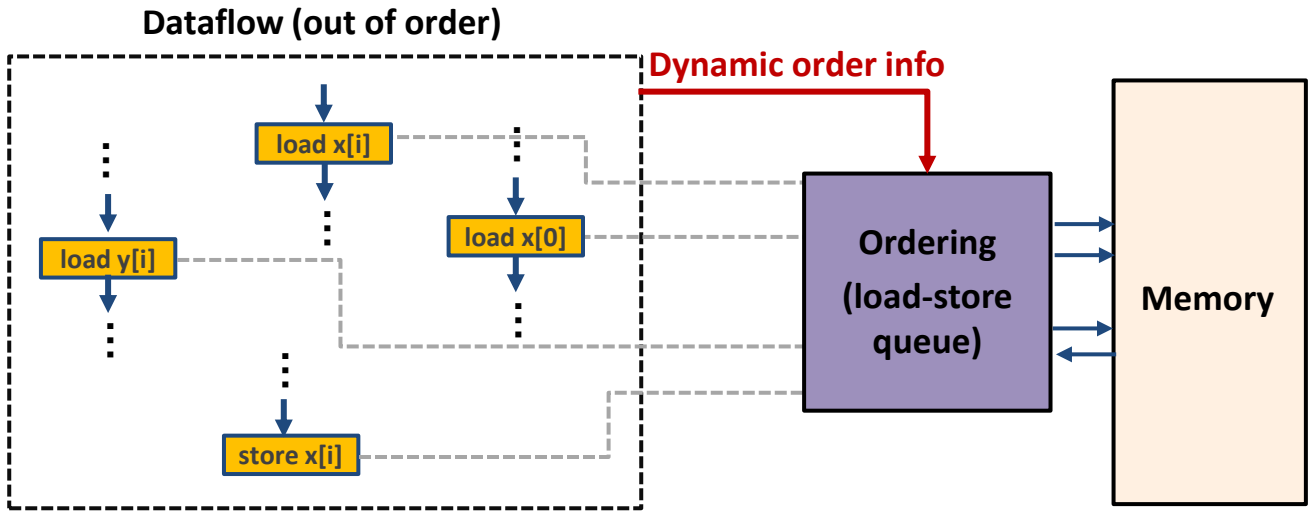
We Need a Load-Store Queue (LSQ)!

- Traditional processor LSQs allocate memory instructions **in program order**



- Dataflow circuits have **no notion of program order**

Dynamic knowledge of basic block sequence from the dataflow circuit

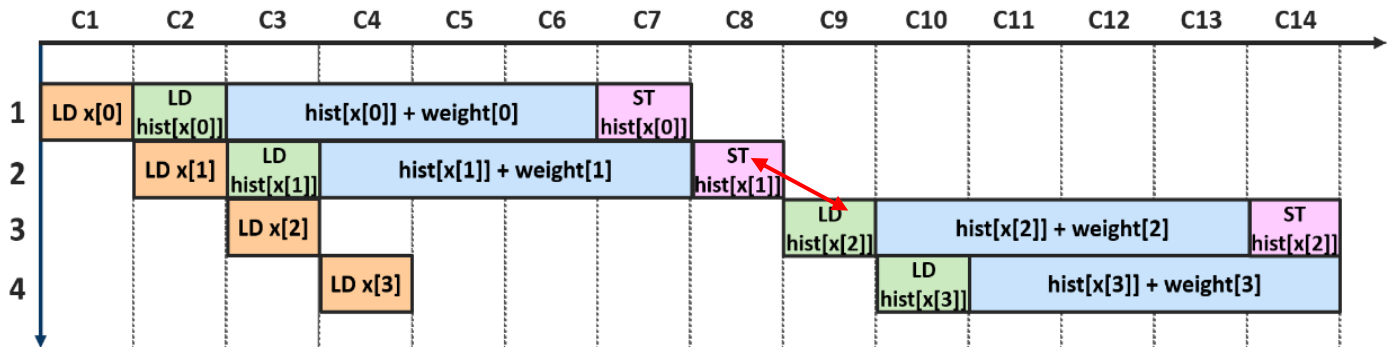


Dataflow Circuit with the LSQ

```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

```
1: x[0]=5 → ld hist[5]; st hist[5];
2: x[1]=4 → ld hist[4]; st hist[4];
3: x[2]=4 → ld hist[4]; st hist[4];
```

RAW dependency

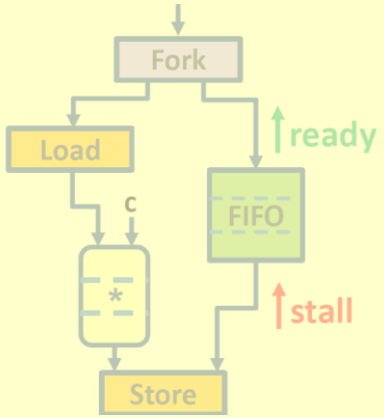


High-throughput pipeline with memory dependencies honored

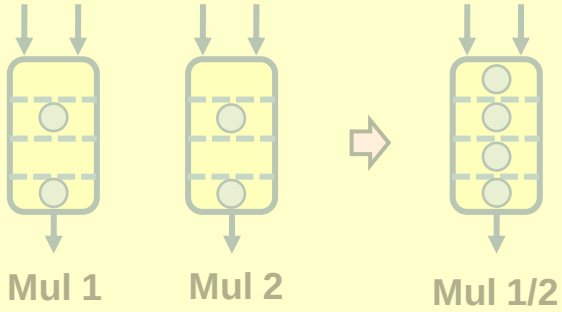
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

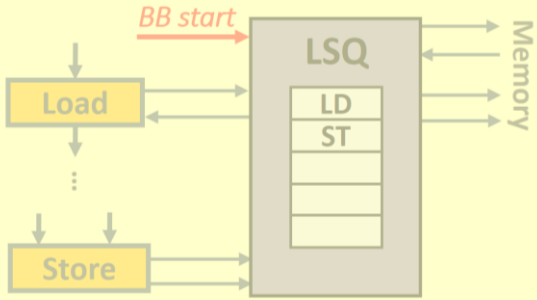


Resource sharing

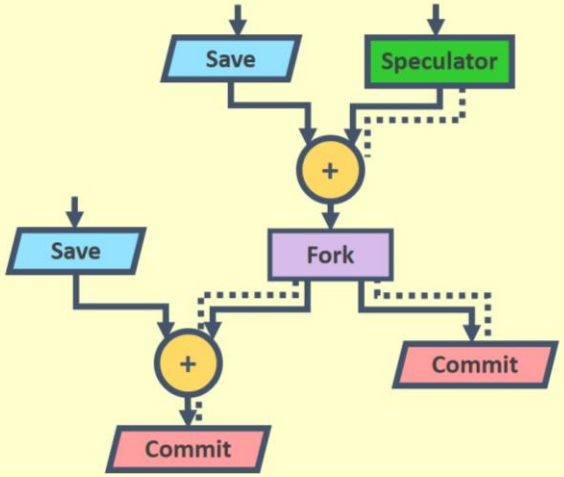


Reaping the benefits of dynamic scheduling

Out-of-order memory

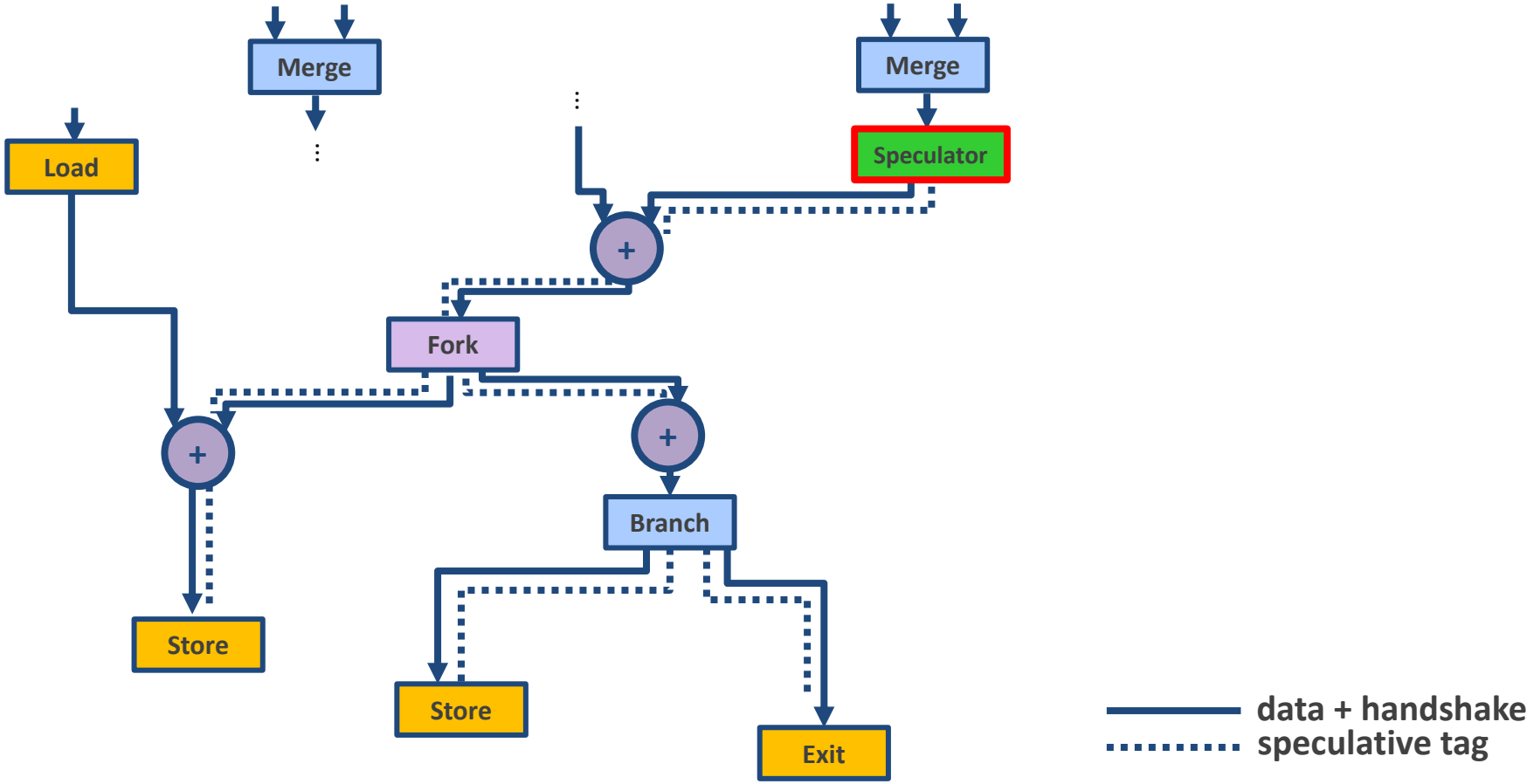


Speculative execution



Speculation in Dataflow Circuits

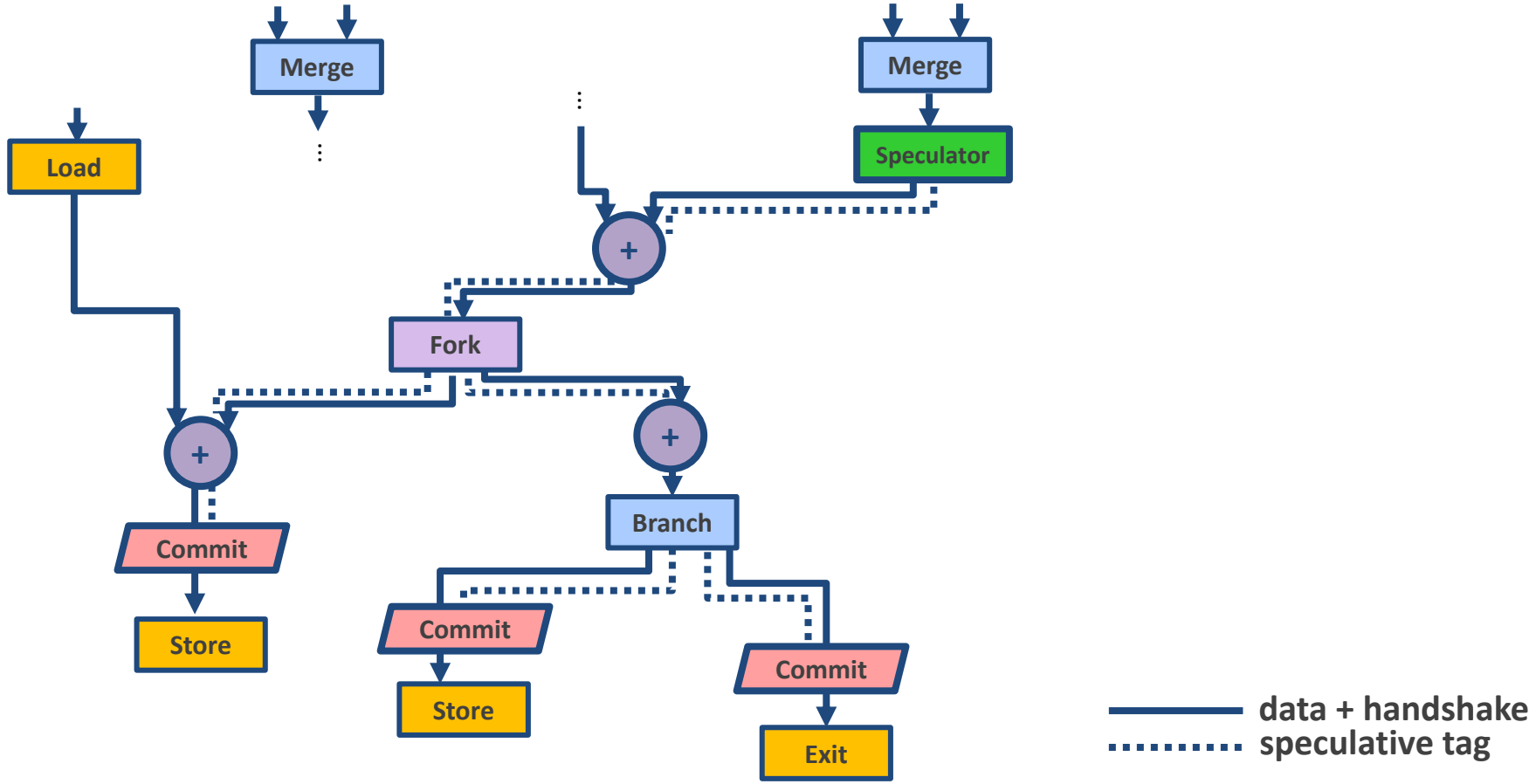
- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation



Josipović, Guerrieri, and Ienne. Speculative Dataflow Circuits. FPGA 2019

Speculation in Dataflow Circuits

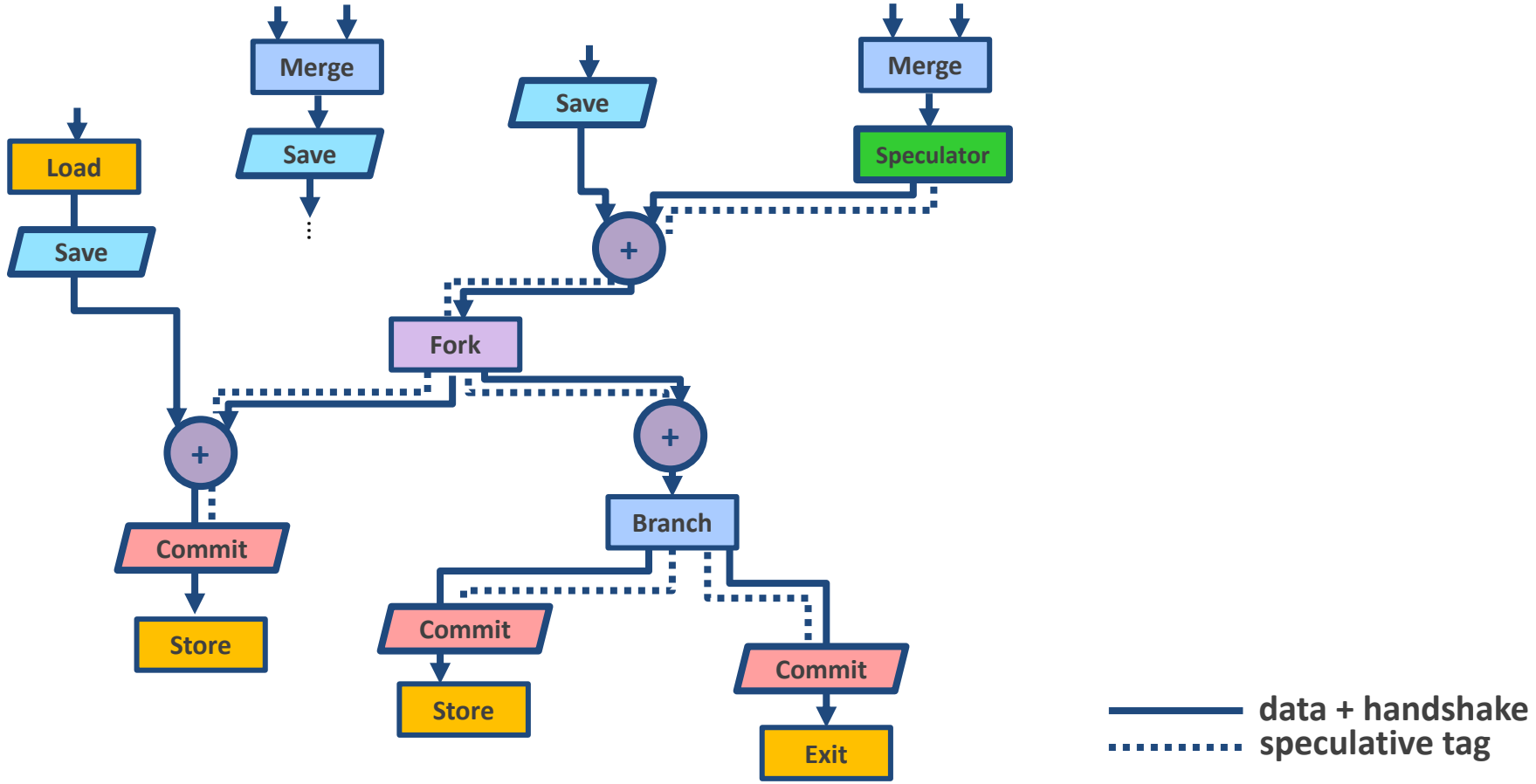
- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation



Josipović, Guerrieri, and Ienne. Speculative Dataflow Circuits. FPGA 2019

Speculation in Dataflow Circuits

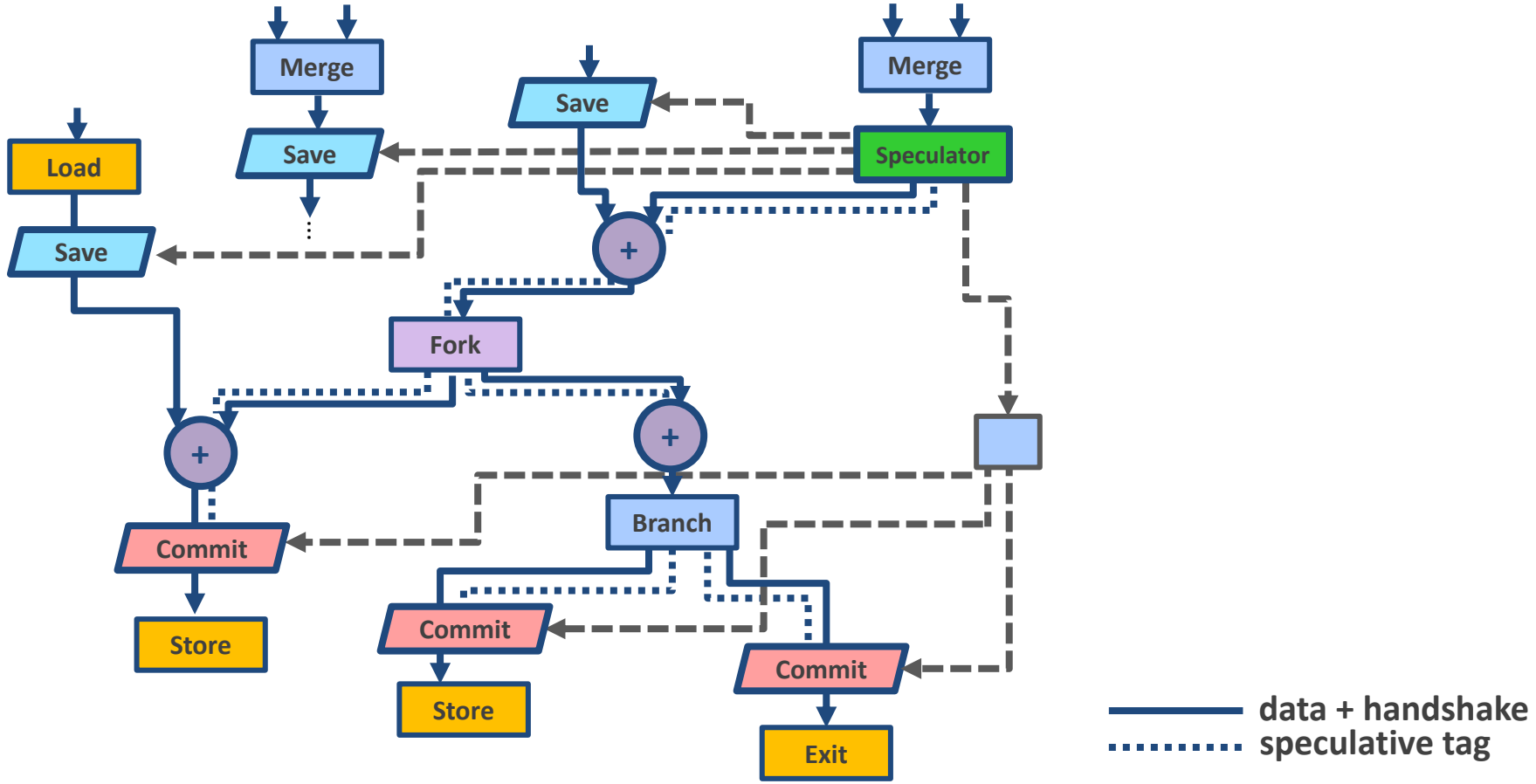
- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation



Josipović, Guerrieri, and Ienne. Speculative Dataflow Circuits. FPGA 2019

Speculation in Dataflow Circuits

- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation

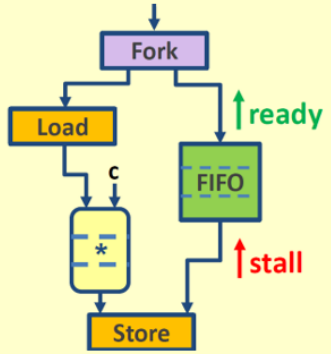


Josipović, Guerrieri, and Ienne. Speculative Dataflow Circuits. FPGA 2019

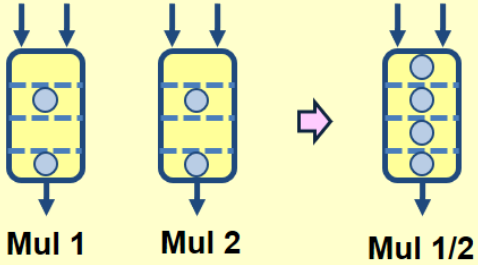
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

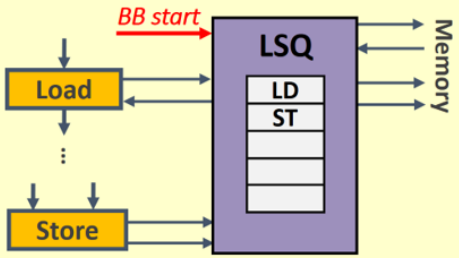


Resource sharing

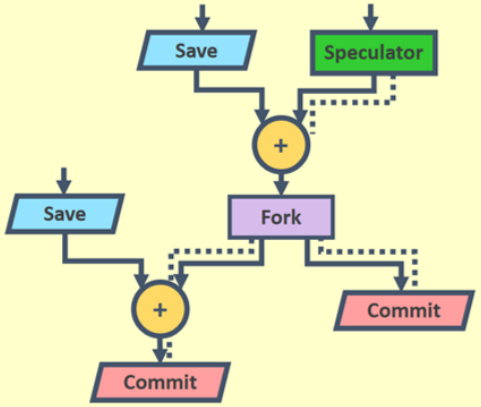


Reaping the benefits of dynamic scheduling

Out-of-order memory



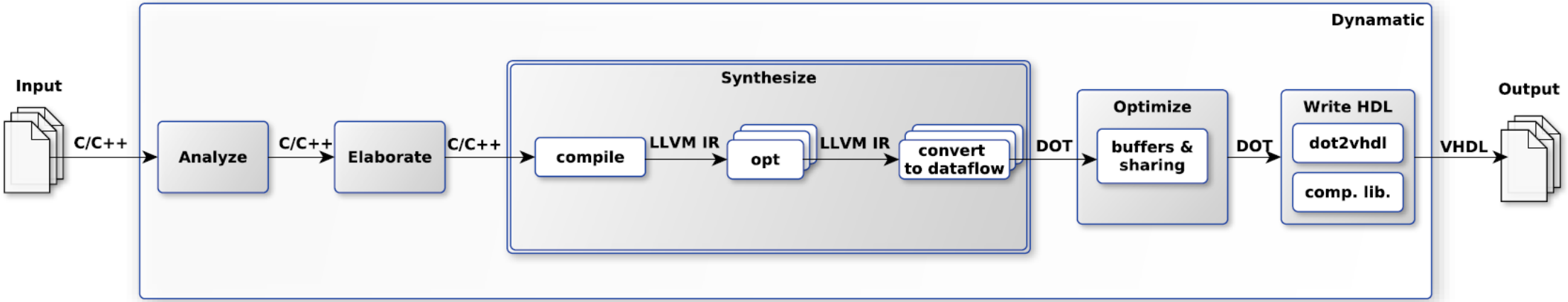
Speculative execution



Static HLS vs. dynamic HLS?

Dynamatic: An Open-Source HLS Compiler

- From C/C++ to synthesizable dataflow circuit description



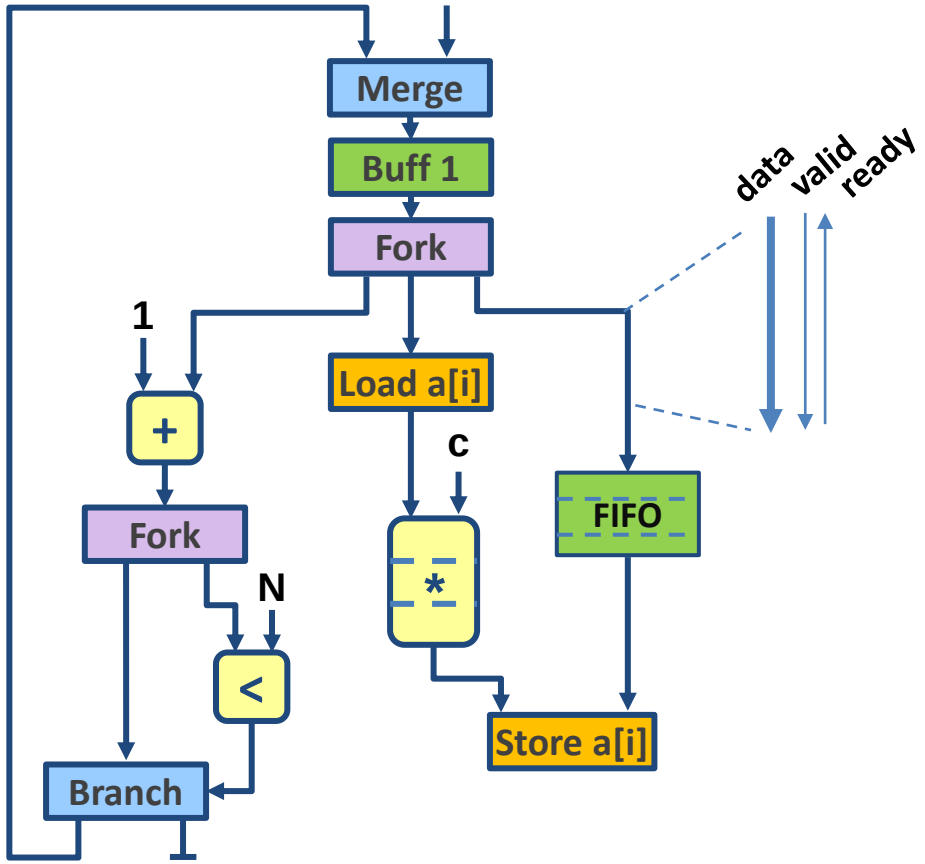
Reduced execution time in irregular benchmarks (speedup of up to 14.9X)

But… dataflow computation is resource-expensive!

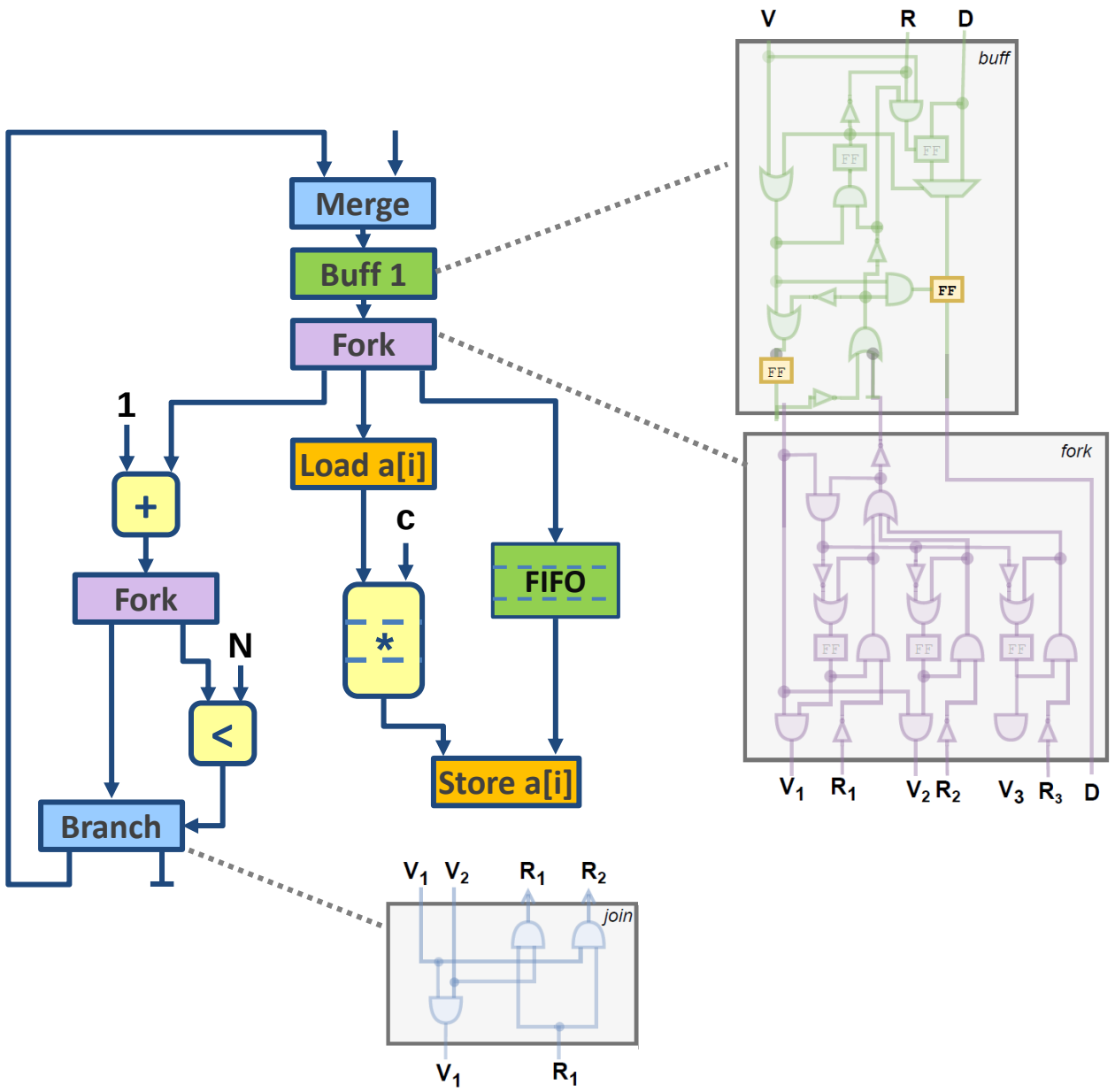
Josipović, Guerrieri, and Ienne. Dynamatic: From C/C++ to Dynamically Scheduled Circuits. FPGA 2020

The Cost of Dataflow Computation

```
for (i=0; i<N; i++) {
   a[i] = a[i]*c;
}
```

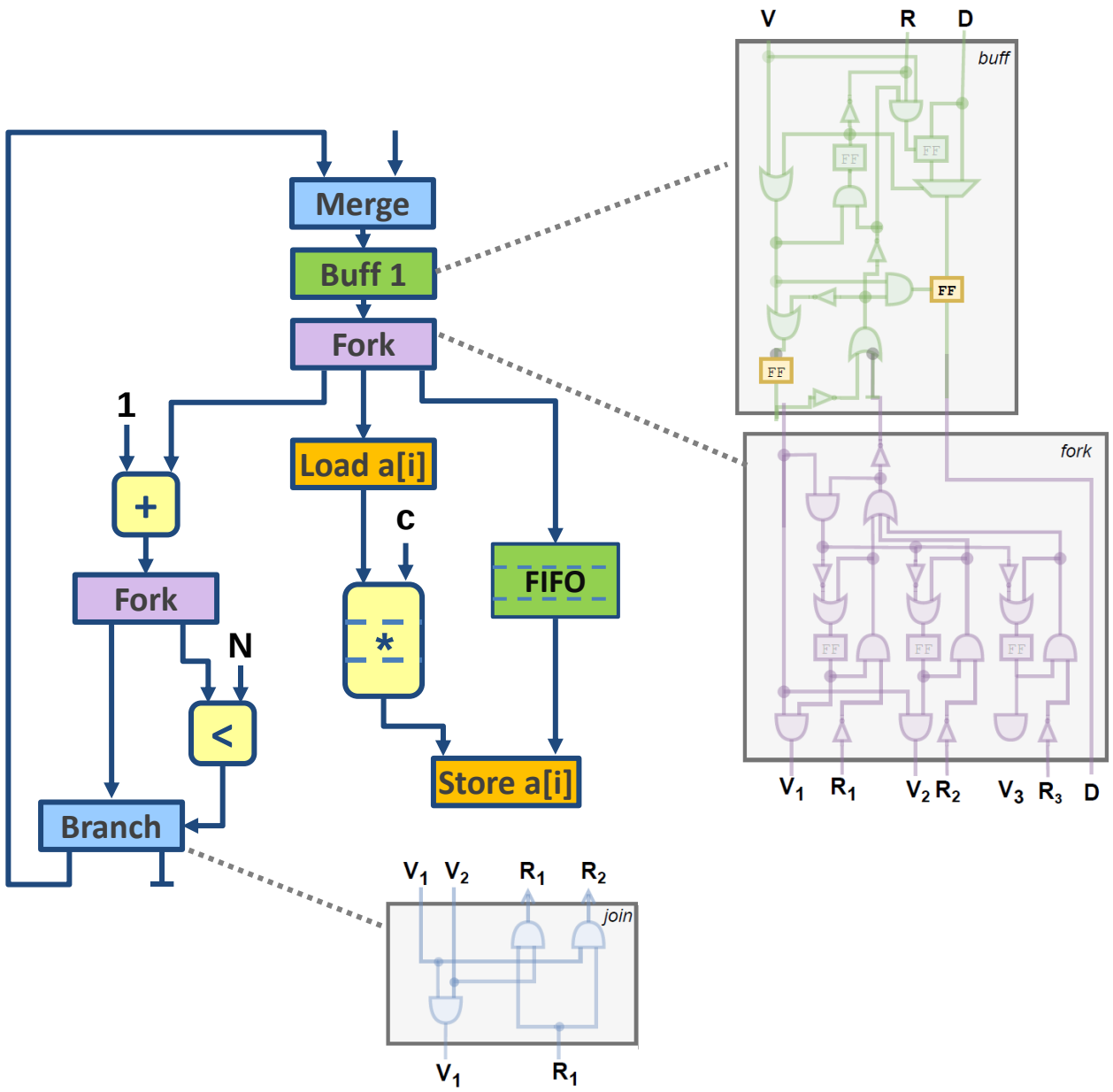


The Cost of Dataflow Computation



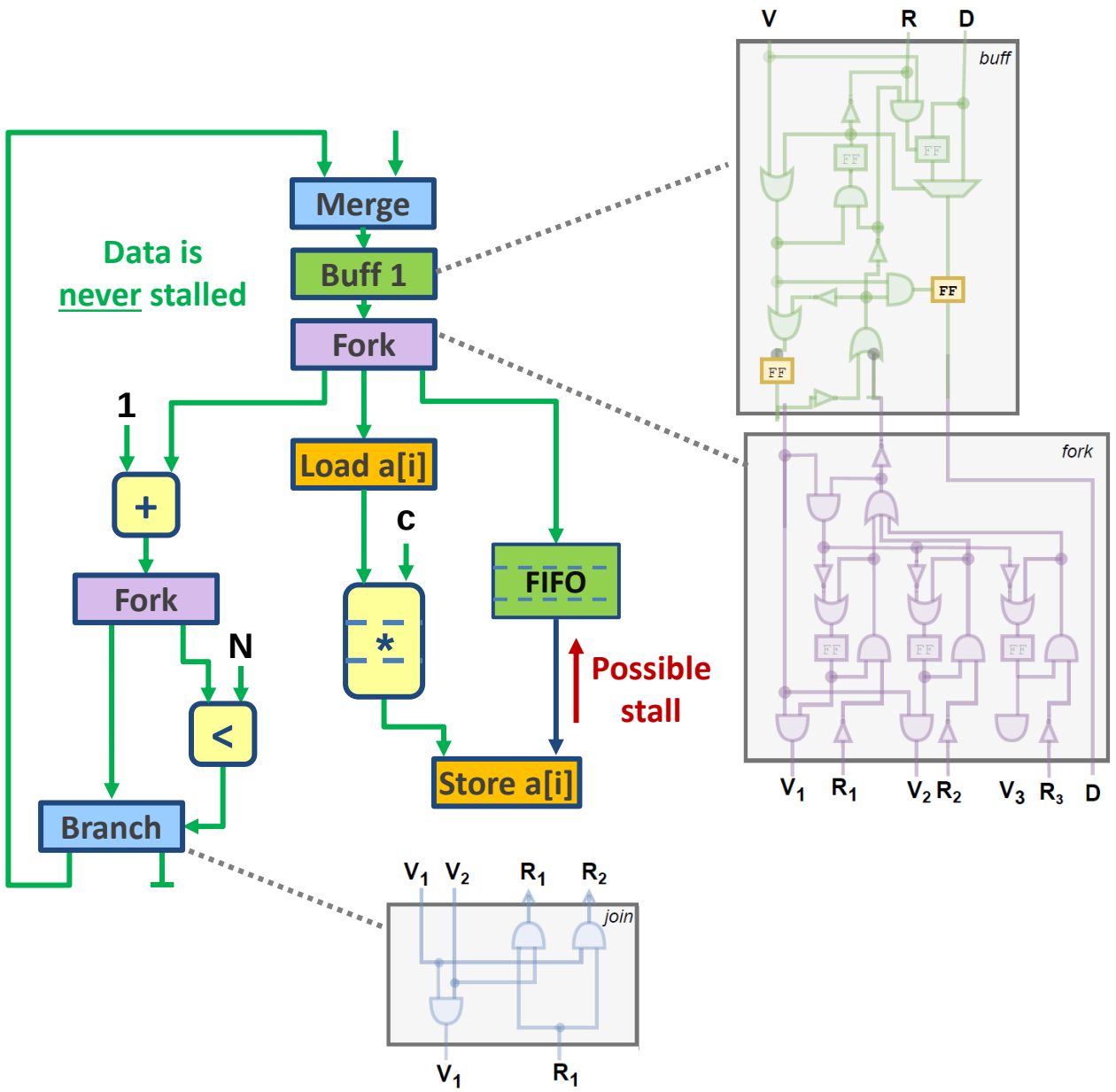
Distributed dataflow handshake mechanism: resource and frequency overhead

The Cost of Dataflow Computation

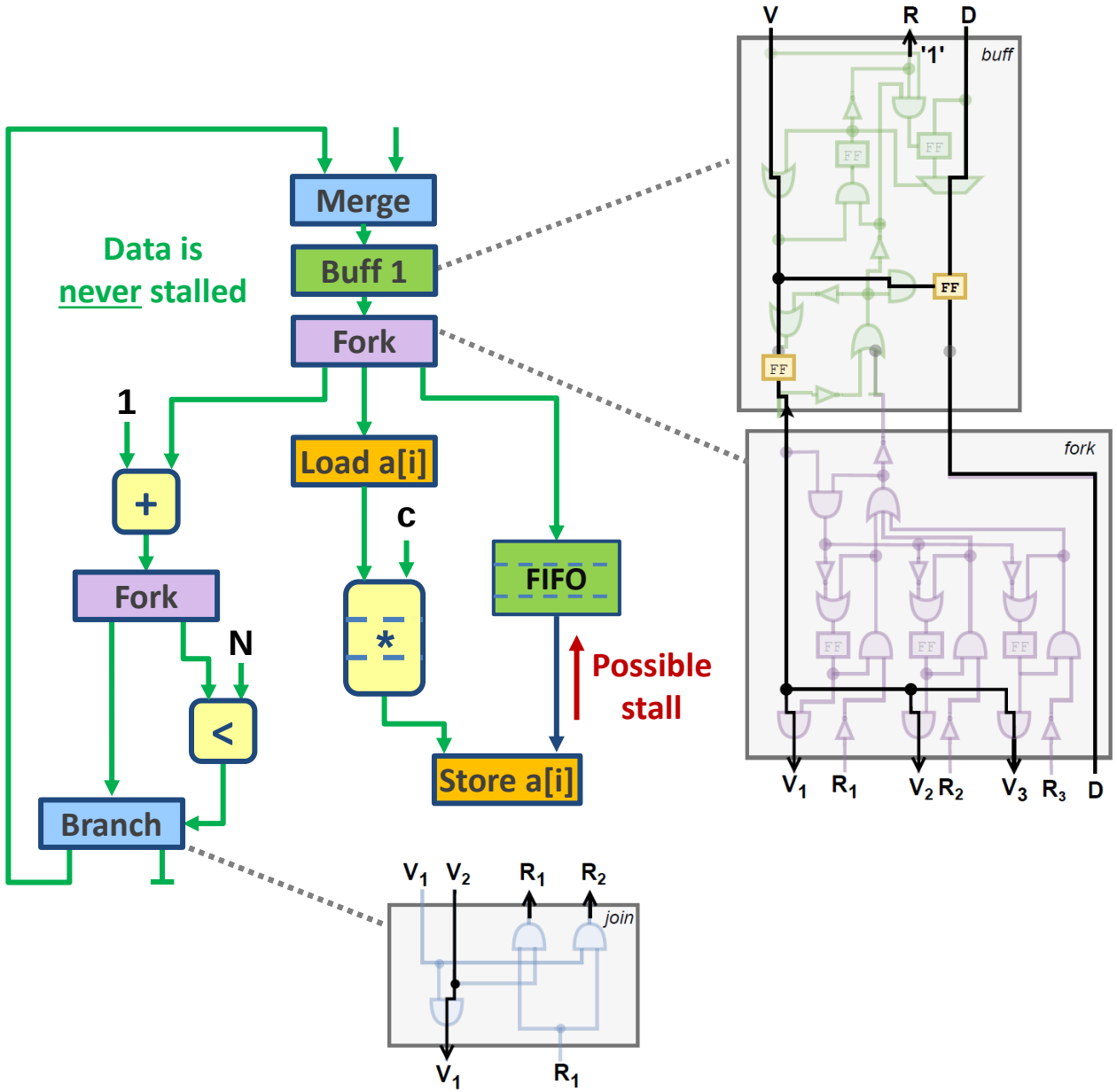


Do we need expensive dataflow logic *everywhere*?

Removing Excessive Dynamism

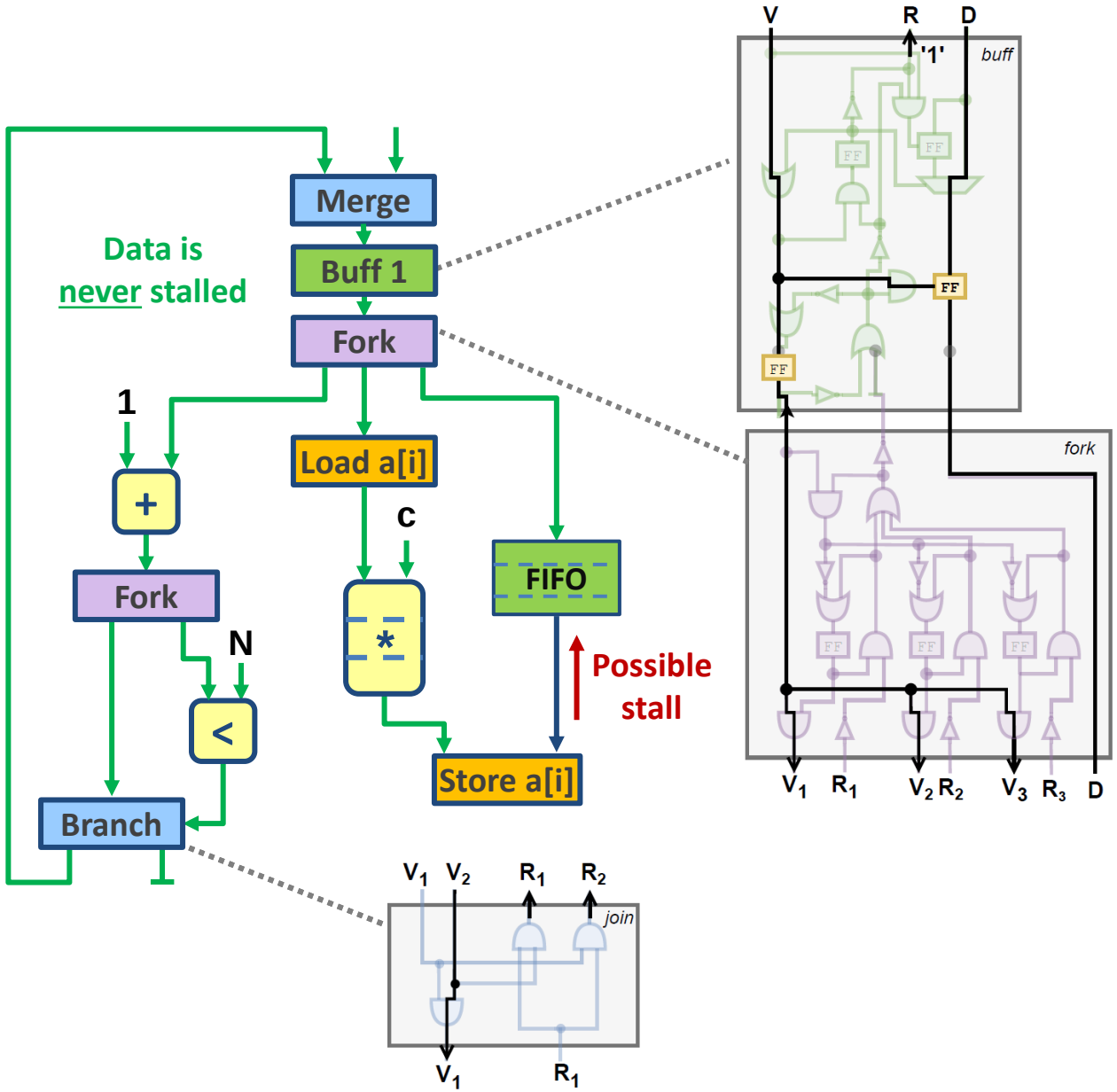


Removing Excessive Dynamism



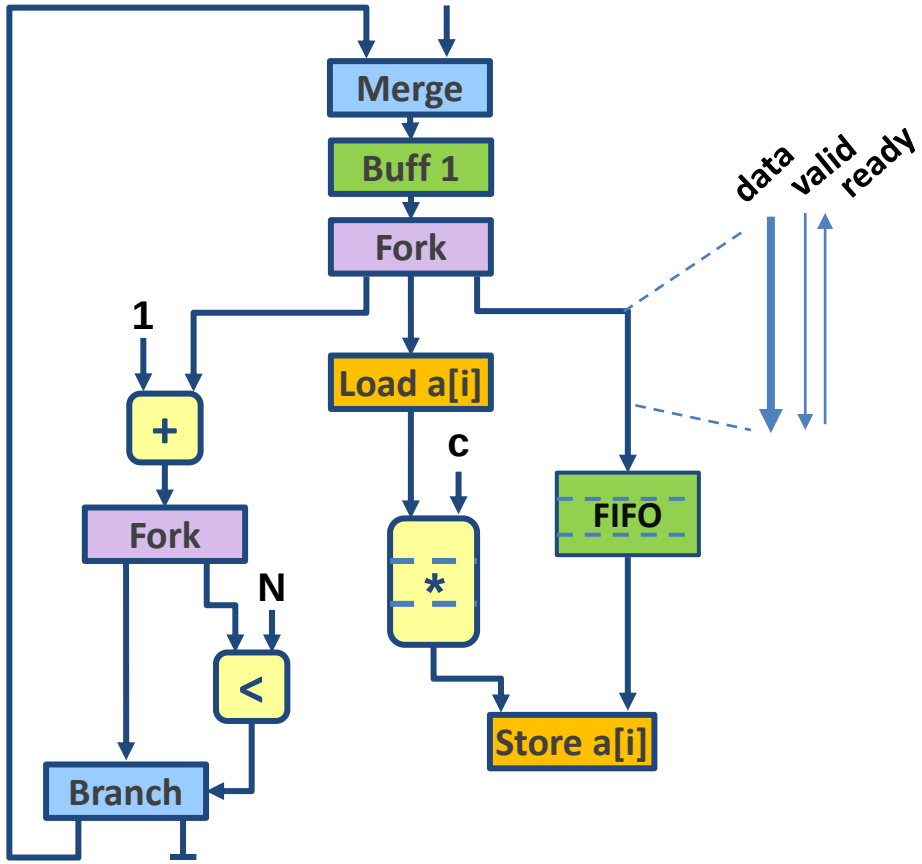
Restrict the generality of dataflow logic whenever it is not needed

Removing Excessive Dynamism



How to guarantee correctness of simplifications for *any possible* circuit behavior?

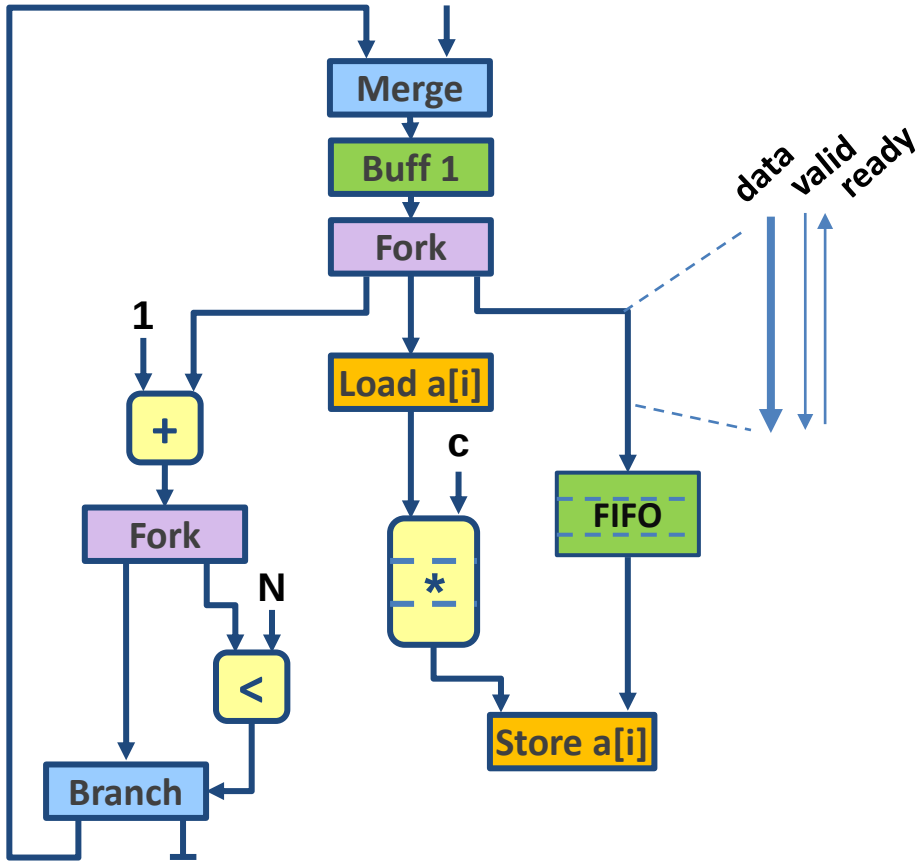
Proving Properties to Eliminate Excessive Dynamism



For each channel: prove the **absence of backpressure**
(remove logic to compute the ready signal)
AG (valid → ready)

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.

Proving Properties to Eliminate Excessive Dynamism

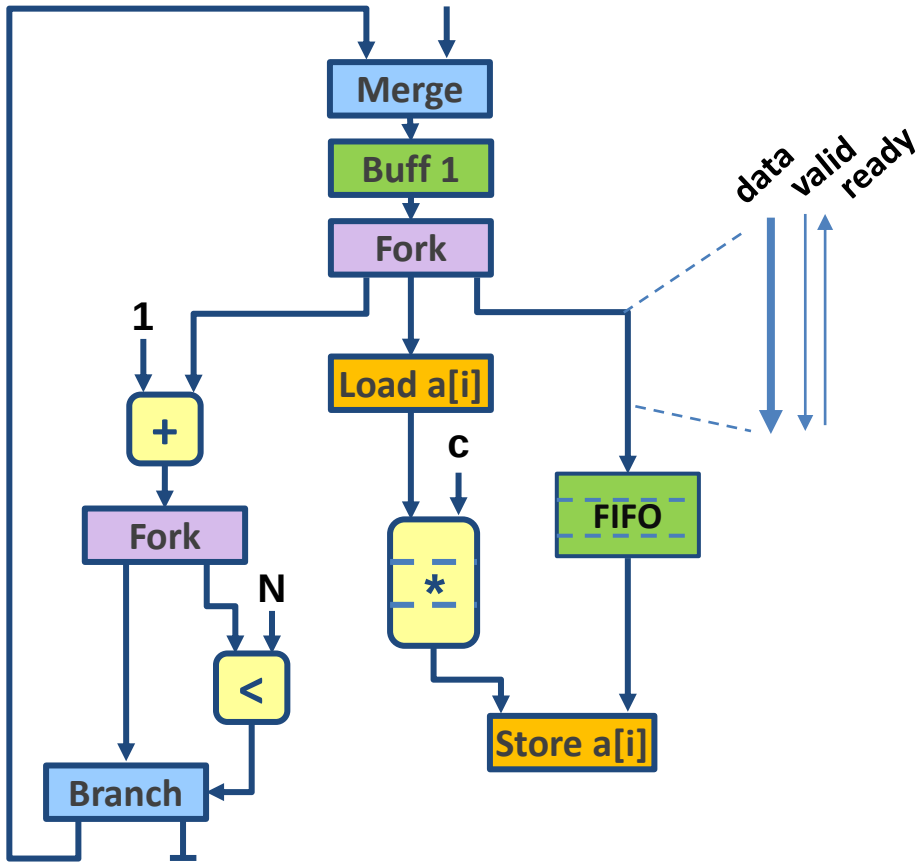


For each channel: prove the **absence of backpressure**
(remove logic to compute the ready signal)
AG (valid → ready)

For each pair of channels: prove **trigger equivalence**
(remove logic to compute one of the valid signals)
AG (valid1 ↔valid2)

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.

Proving Properties to Eliminate Excessive Dynamism



For each channel: prove the **absence of backpressure**
(remove logic to compute the ready signal)
AG (valid → ready)

For each pair of channels: prove **trigger equivalence**
(remove logic to compute one of the valid signals)
AG (valid1 ↔valid2)

Up to 50% area reduction without a performance penalty

But it is very slow (~hrs)…

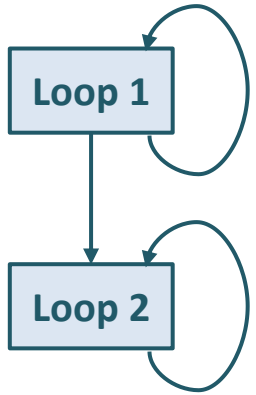
Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.

Ensuring Scalability by Compositional Verification

- **Decompose circuit** into regions whose properties can be verified independently
- **Abstract the complexity** of other regions into simpler nodes that have the same properties as the circuit they encapsulate

```
for (i = 0; i < N; i++)
          ...

for (i = 0; i < N; i++)
          ...
```



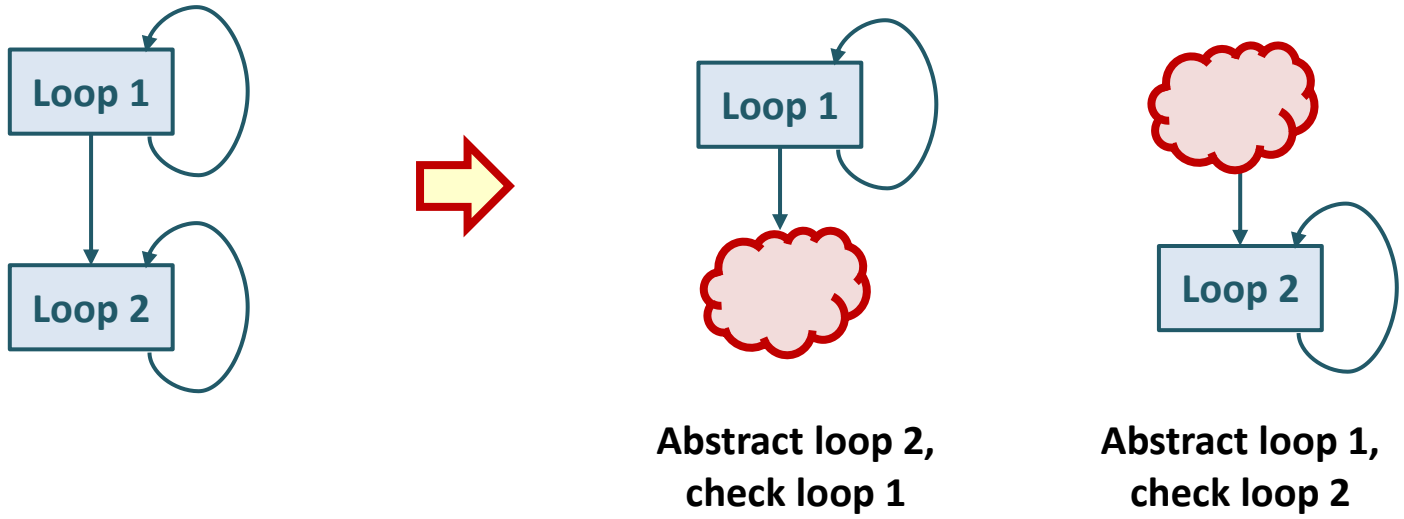
Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.

Ensuring Scalability by Compositional Verification

- **Decompose circuit** into regions whose properties can be verified independently
- **Abstract the complexity** of other regions into simpler nodes that have the same properties as the circuit they encapsulate

```
for (i = 0; i < N; i++)
          ...

for (i = 0; i < N; i++)
          ...
```



Up to 8X reduction in checking time

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.

DYNAMO: Digital Systems and Design Automation Group

High-level abstractions
programming languages,
software applications

HLS compilers
formal methods,
electronic design automation

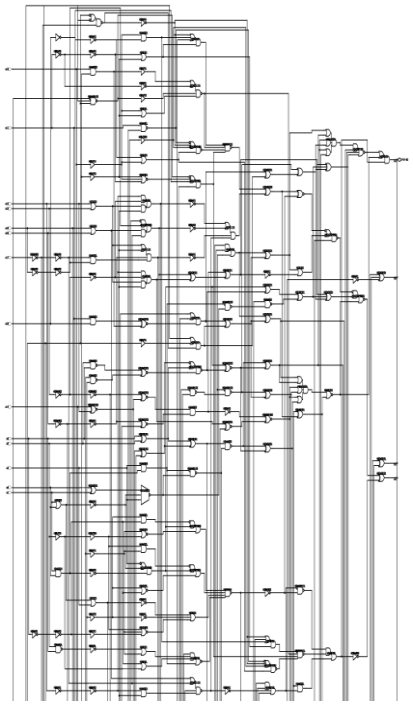
Hardware accelerators
systems, digital design,
computer architecture

```
#define PI 3.14159265358979323846

complex* DFT_naive(complex* x, int
  complex* X = (complex*) malloc(s
  int k, n;
  for(k = 0; k < N; k++) {
    X[k].re = 0.0;
    X[k].im = 0.0;
    for(n = 0; n < N; n++) {
      X[k] = add(X[k], multiply(x[
                                co


    }
  }

  return X;
}
```



Enable diverse users to accelerate compute-intensive applications on hardware platforms

MSc & BSc Projects and Theses

- Use **Petri nets** to describe circuits and their behaviors
  - Component modelling
  - Performance and area optimizations
- Use **model checking** to prove circuit properties and improve their quality
  - Checking more complex properties
  - Dealing with scalability issues
- And many other topics...
- Check link on last slide for (non-exhaustive) list of projects!

Come work with us! ☺

New Course in Spring 2023: Synthesis of Digital Circuits

- Algorithms, tools, and methods to generate circuits from high-level programs
  - How does ‘classic’ HLS work?
- Recent advancements and current challenges of HLS for FPGAs
  - What is HLS still missing?
- Course organization
  - First part: lectures+exercises
  - Second part: practical work + seminar-like discussions
- Link to Course Catalogue info (2023)

Hope to see you there! ☺

Thanks! ☺

Research group

Link

Project list 2023

Link