From C/C++ to Dynamically Scheduled Circuits

Lana Josipović

September 2022

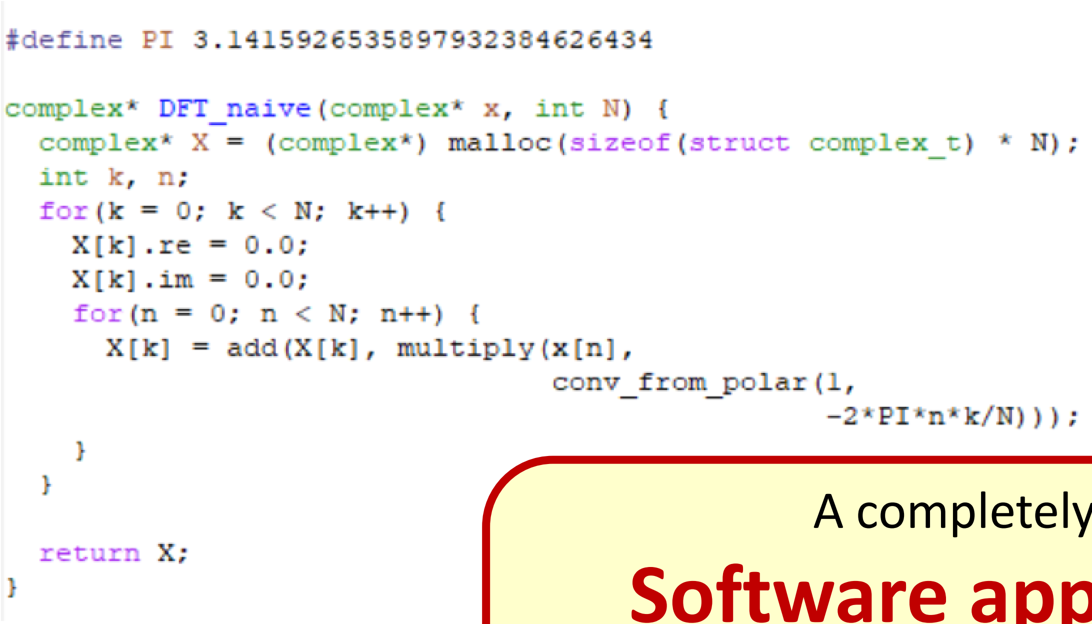
ETH zürich

FPGA
A
B
C
Microsoft
Google
Tensor Processing Unit
How it Works
HARDWARE DEVELOPMENT KIT
HDK
Write your FPGA code with the FPGA Hardware Development Kit and FPGA Developer AMI
CUSTOM LOGIC
AMAZON FPGA IMAGE (AFI)
Attach your AFI to an F1 instance
AWS MARKETPLACE
F1 INSTANCE
Attach your AFI to an F1 instance
DEVELOP
Develop custom AFI using the Hardware Development Kit (HDK) and full set of design tools and simulators.
OFFER
AWS Marketplace for other
PURCHASE
on AWS Marketplace to quickly
amazon web services™
Actel ProASIC 3E
CERN QPS
CERN

How to perform hardware design?

high parallelism and energy efficiency

High-Level Synthesis: From Programs to Circuits

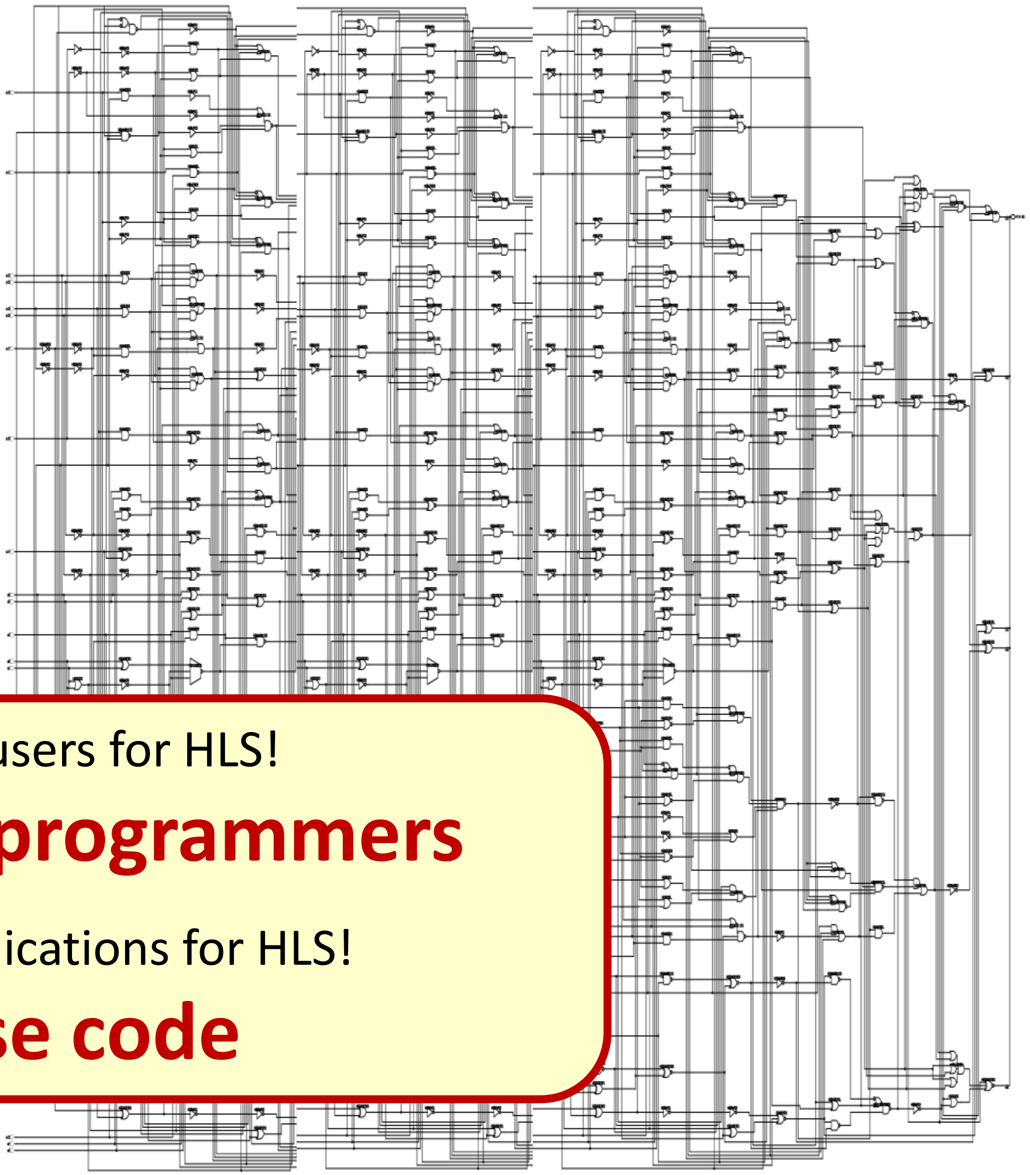


```
#define PI 3.1415926535897932384626434

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

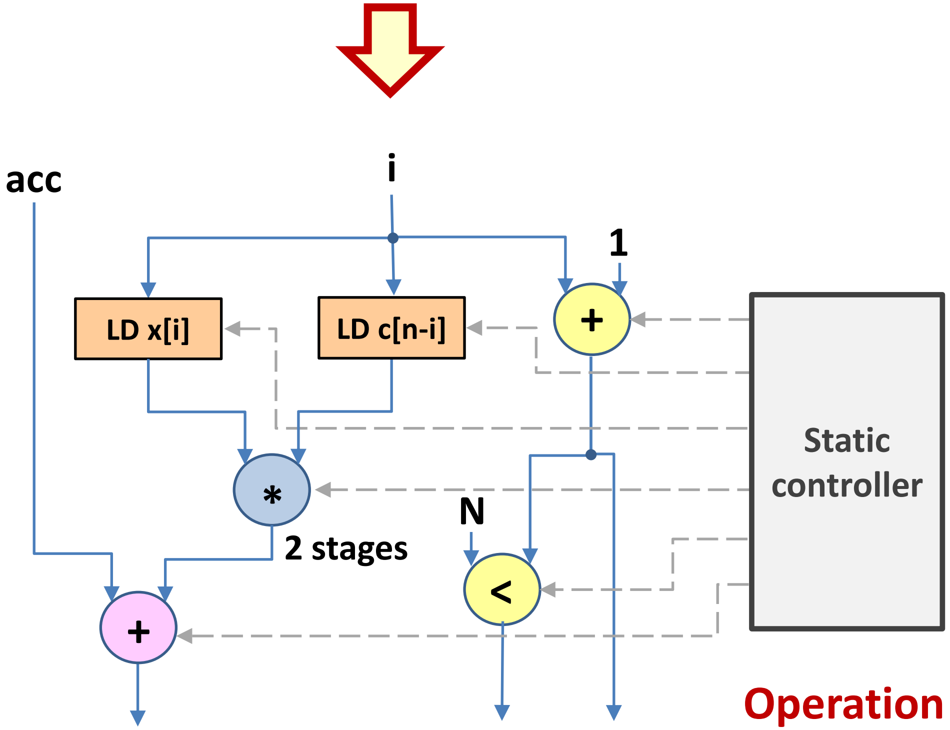
A completely new type of applications for HLS!

General-purpose code

Standard HLS

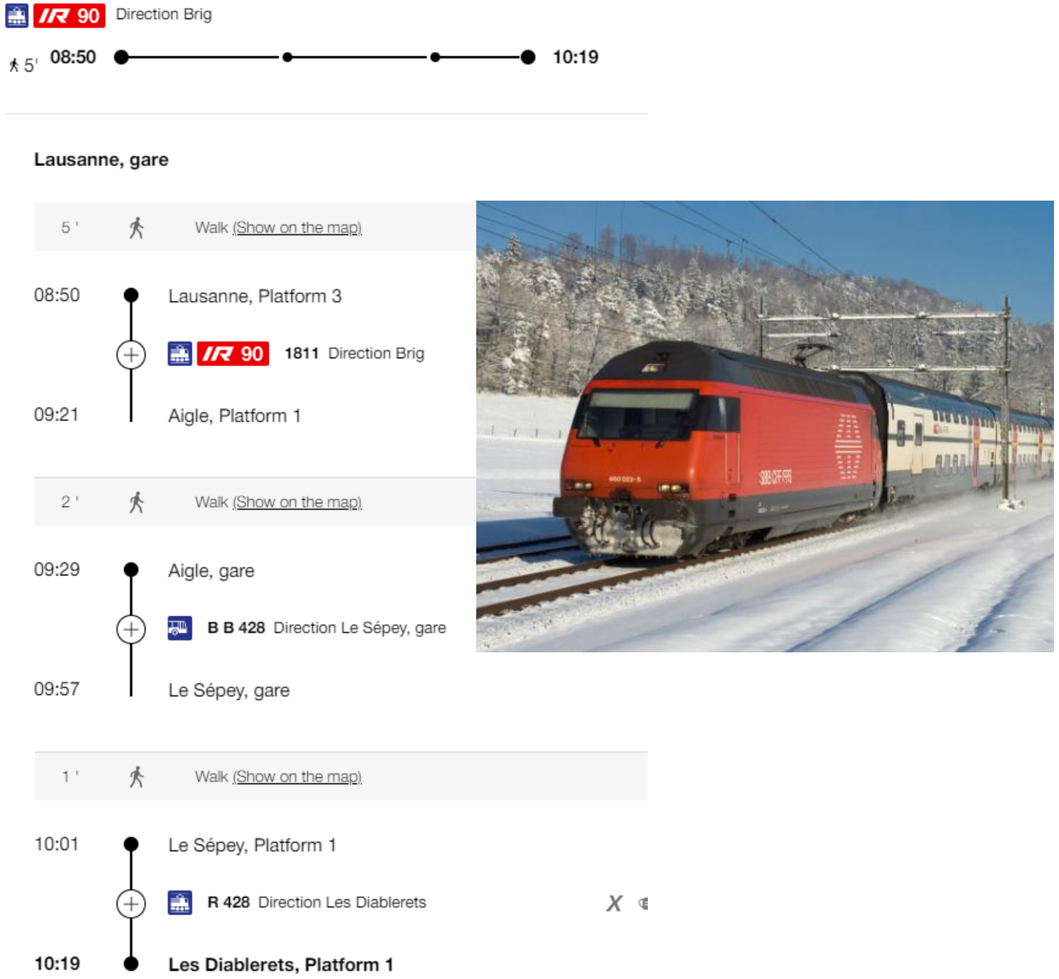
- **Create a datapath** suitable to implement the required computation
- Create a **fixed schedule at compile time** to activate the datapath components

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



Program functionality

Operation schedule



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

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

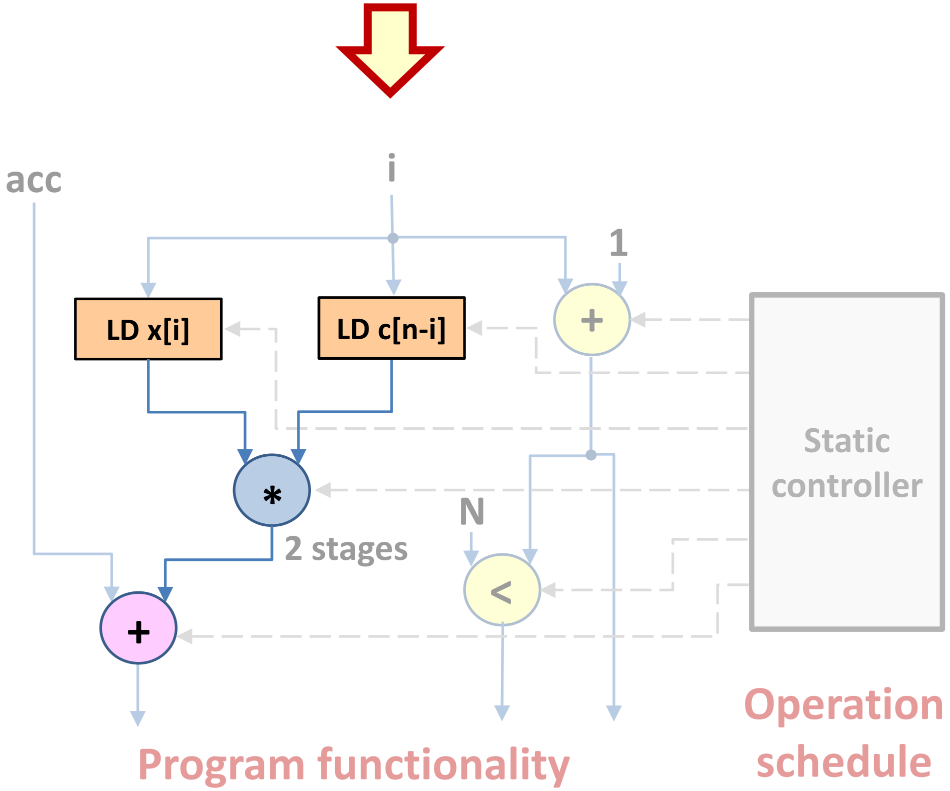
R 428 Direction Les Diablerets X

10:19 Les Diablerets, Platform 1

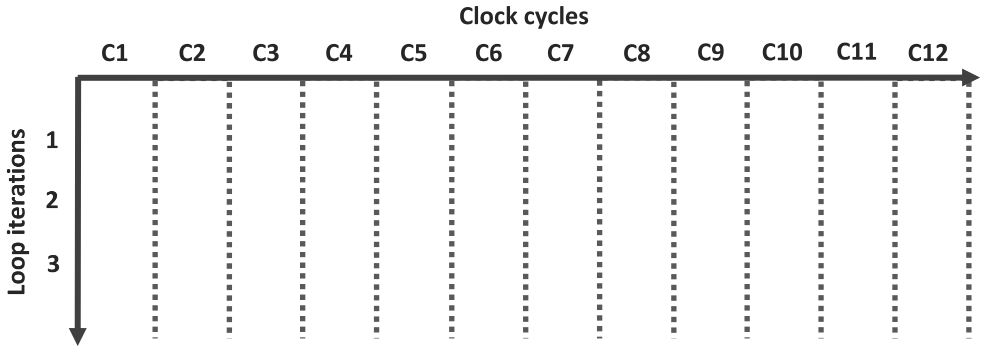
Standard HLS

- **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];
}
```



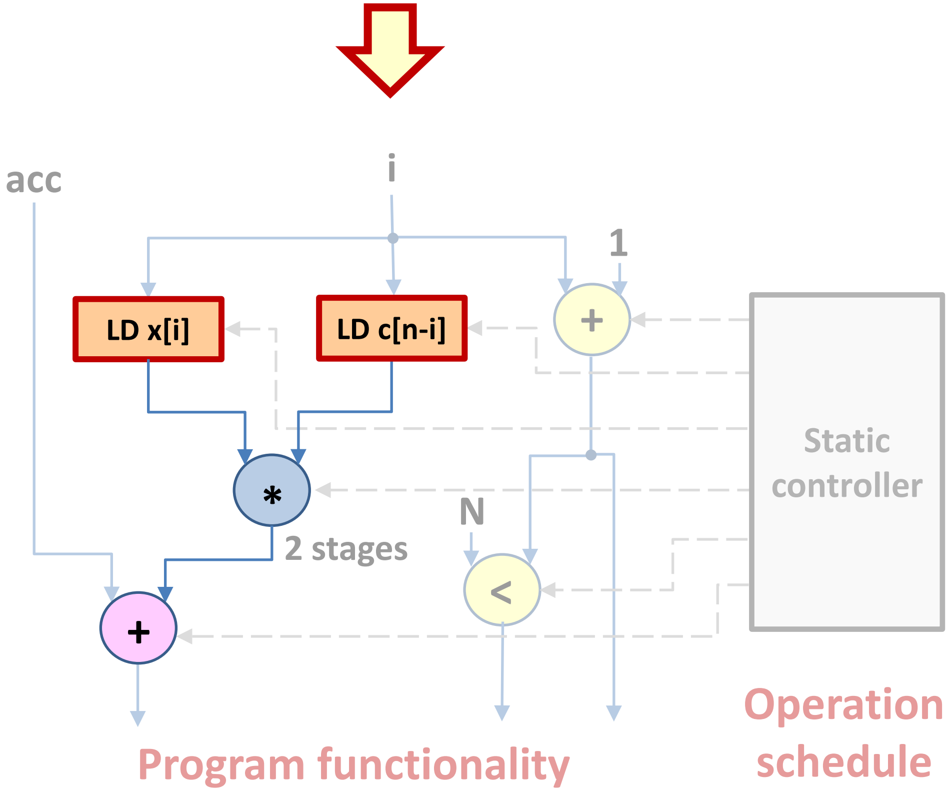
Naïve schedule:

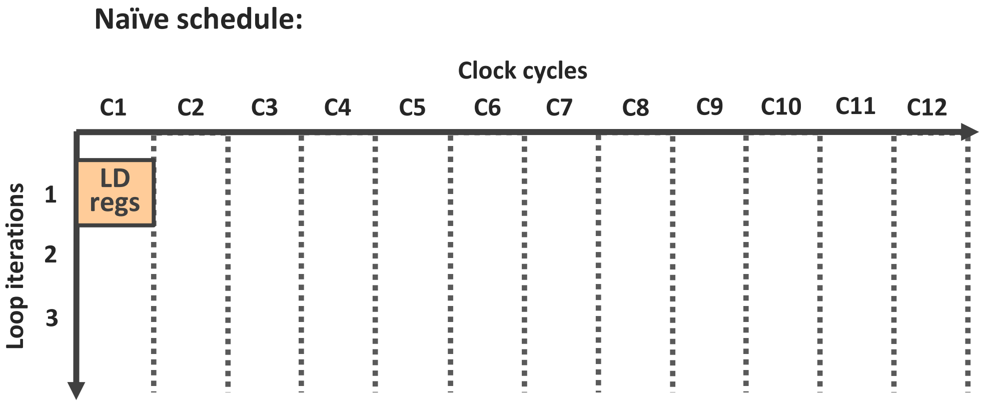


Standard HLS

- **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];
}
```

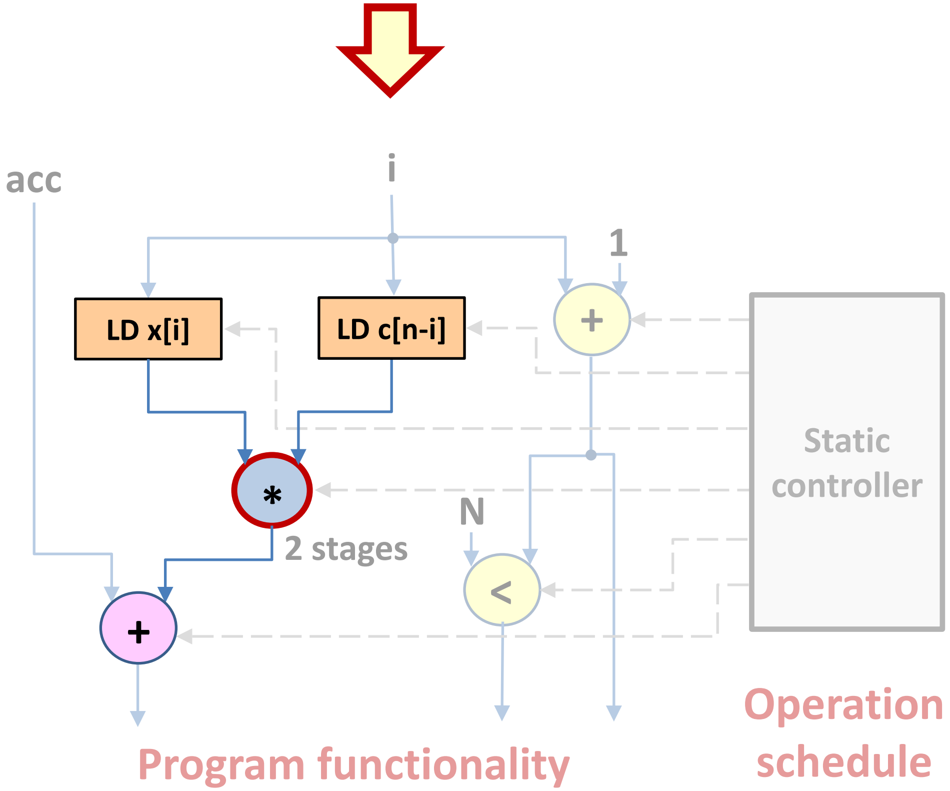




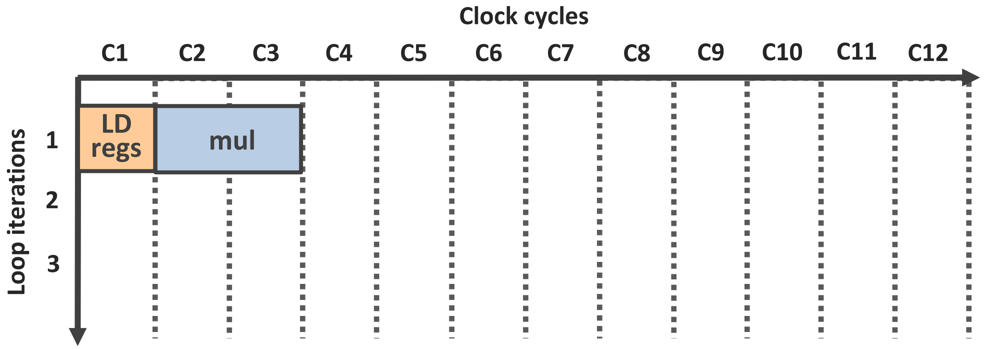
Standard HLS

- **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];
}
```



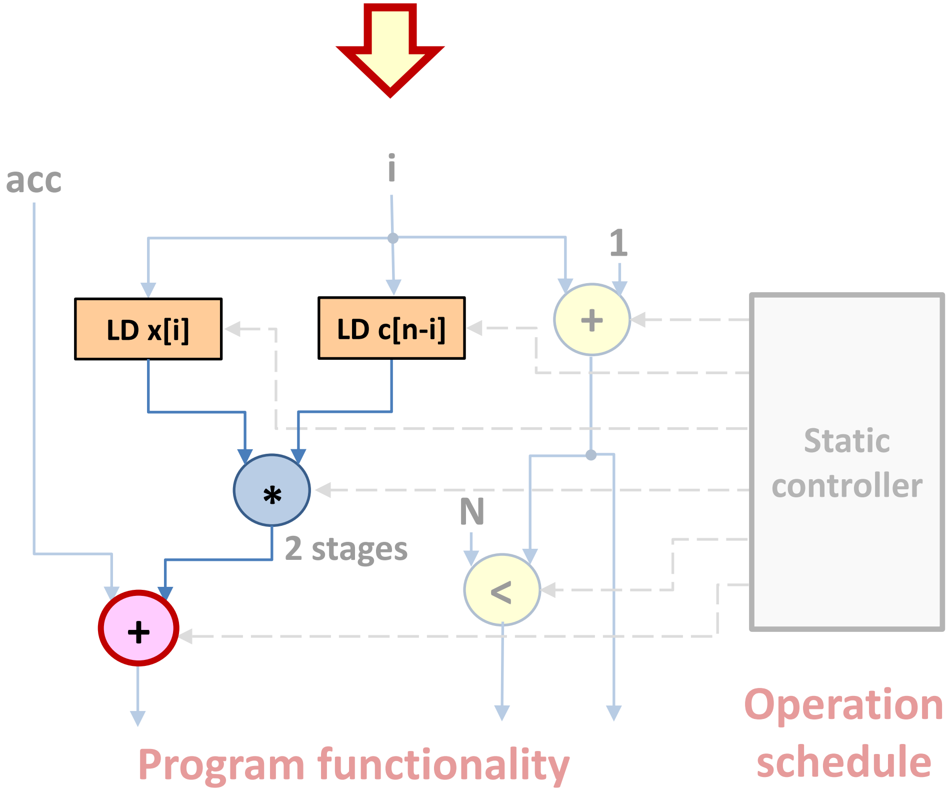
Naïve schedule:



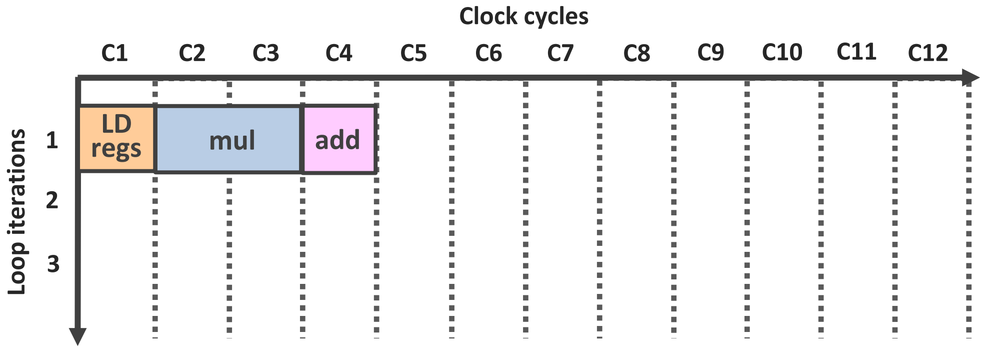
Standard HLS

- **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];
}
```



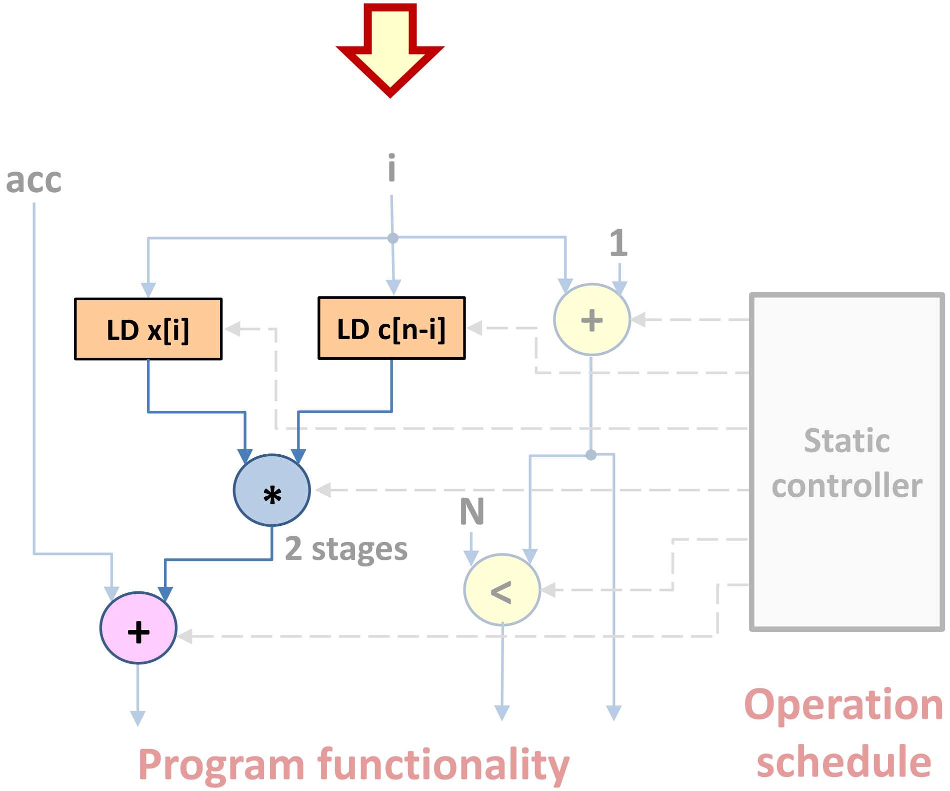
Naïve schedule:



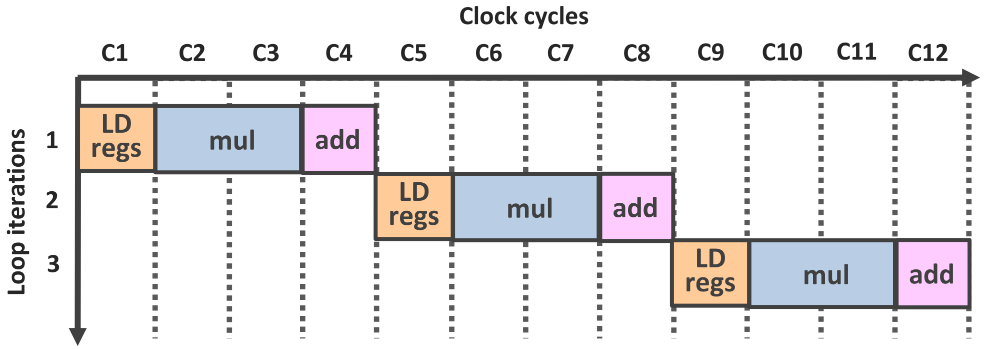
Standard HLS

- **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];
}
```



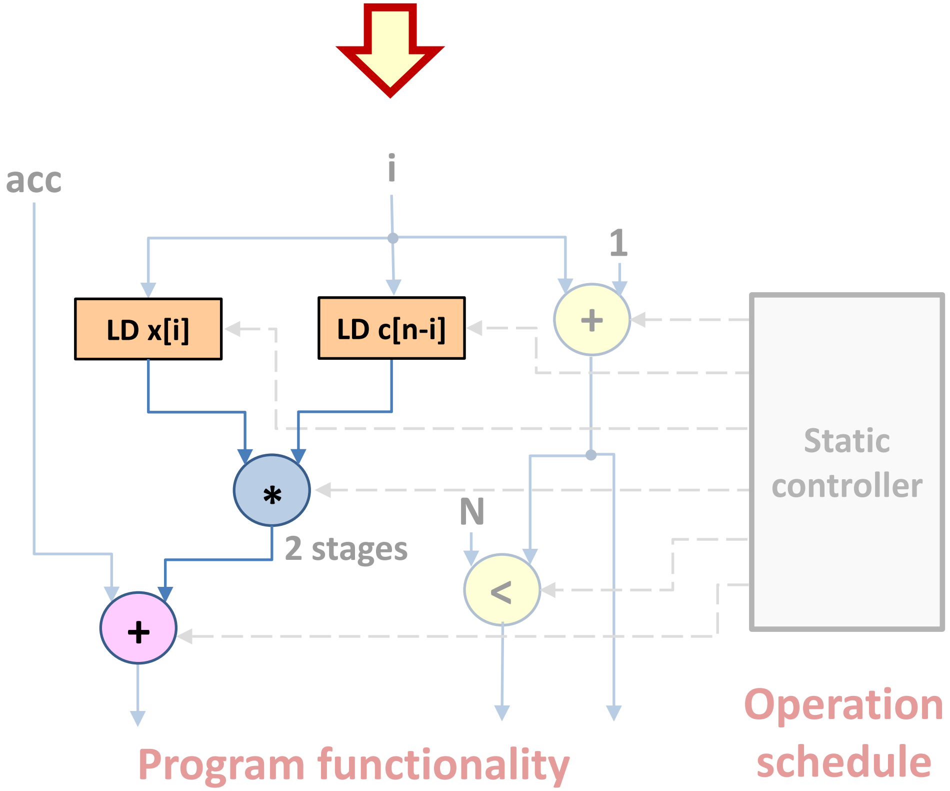
Naïve schedule:

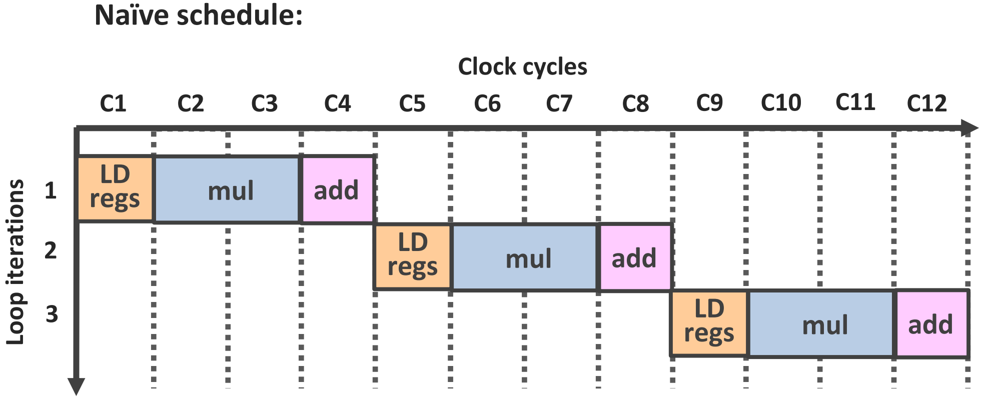


Standard HLS

- **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];
}
```



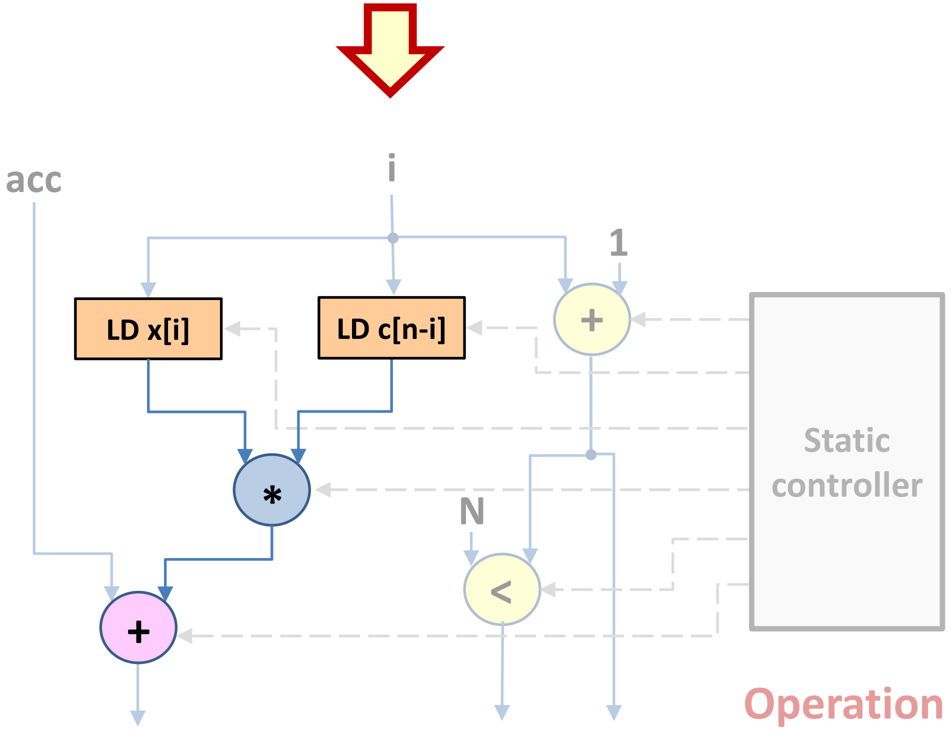


Low throughput: slow execution

Standard HLS

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

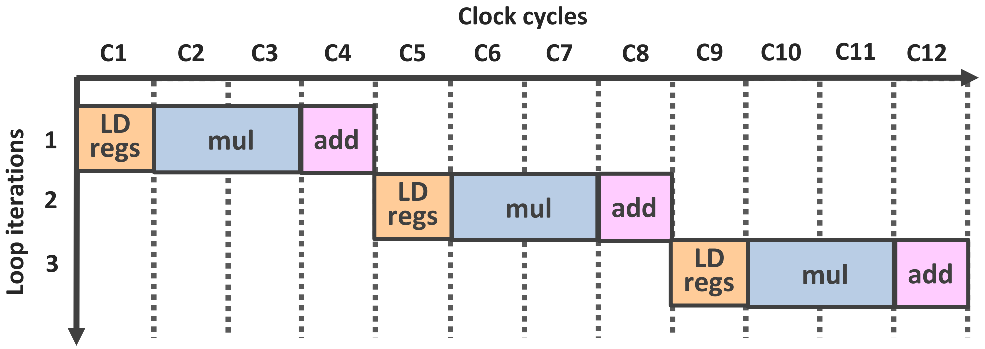
```
for (i=0; i<n; i++) {
        acc += x[i] * c[n-i];
}
```



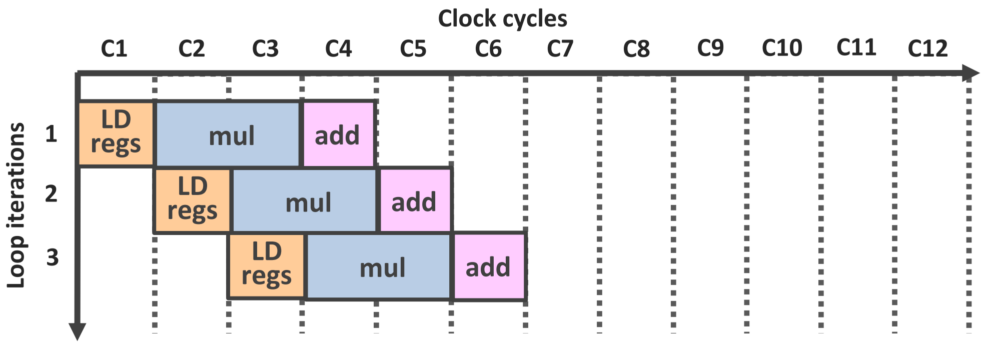
Program functionality

Operation schedule

Naïve schedule:



Pipelined schedule:



High throughput: fast execution

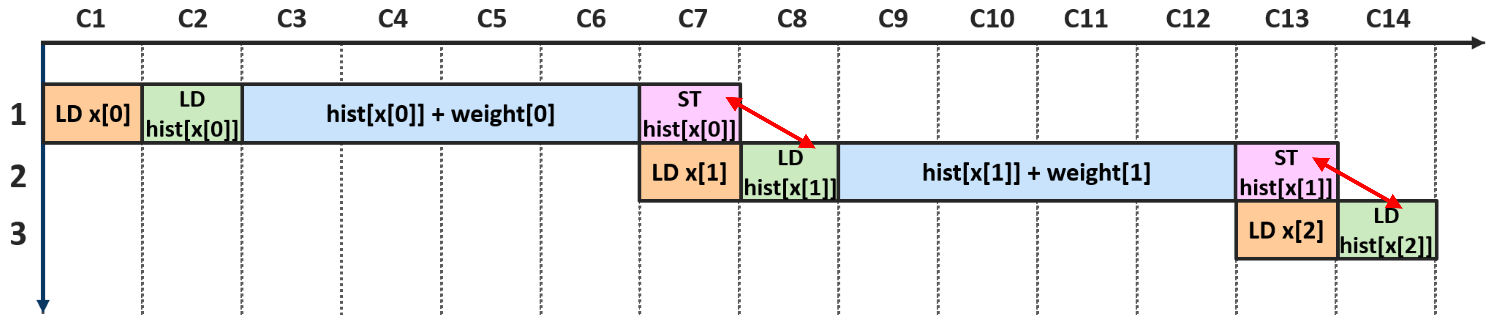
The Limitations of Static Scheduling

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

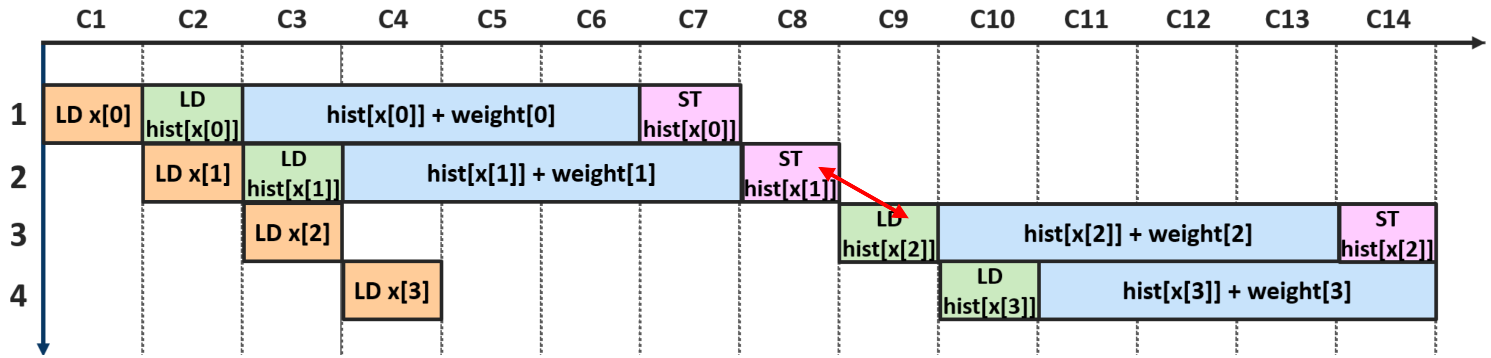
```
1: x[0]=5 → ld hist[5]; st hist[5];
2: x[1]=4 → ld hist[4]; st hist[4];
3: x[2]=4 → ld hist[4]; st hist[4];
```

RAW dependency

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

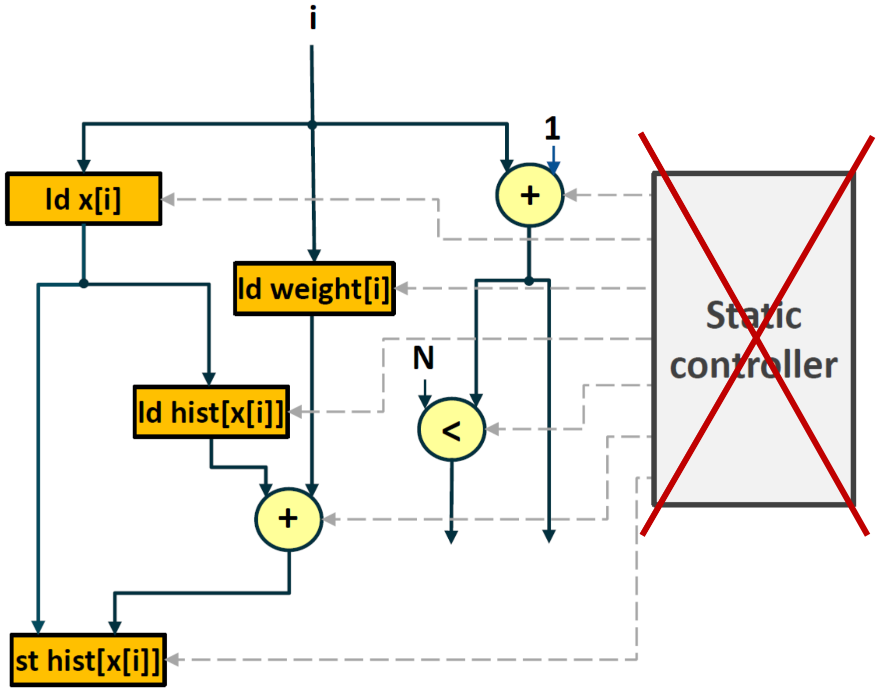


- Dynamic scheduling
  - Maximum parallelism: Only serialize memory accesses on actual dependencies

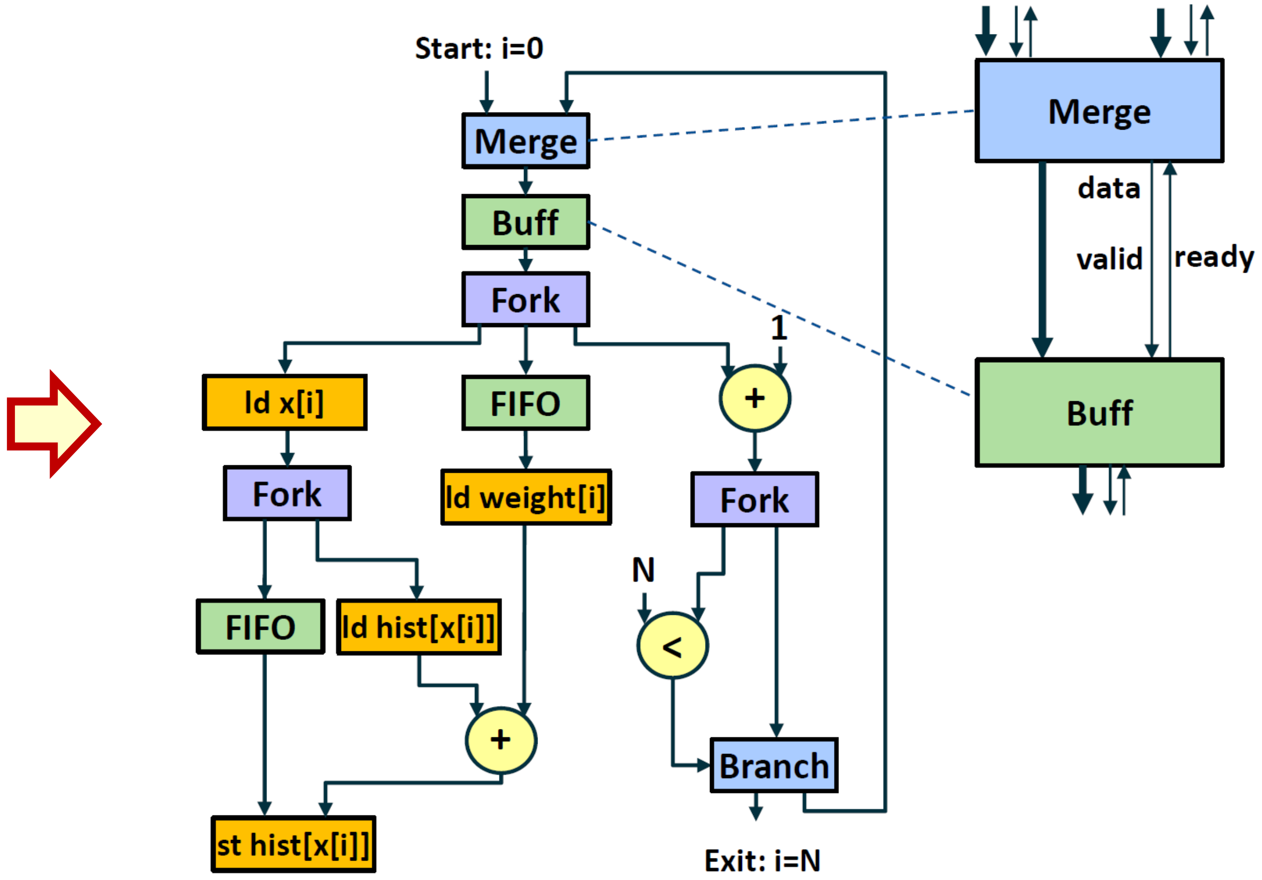


A Different Way to Do HLS

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

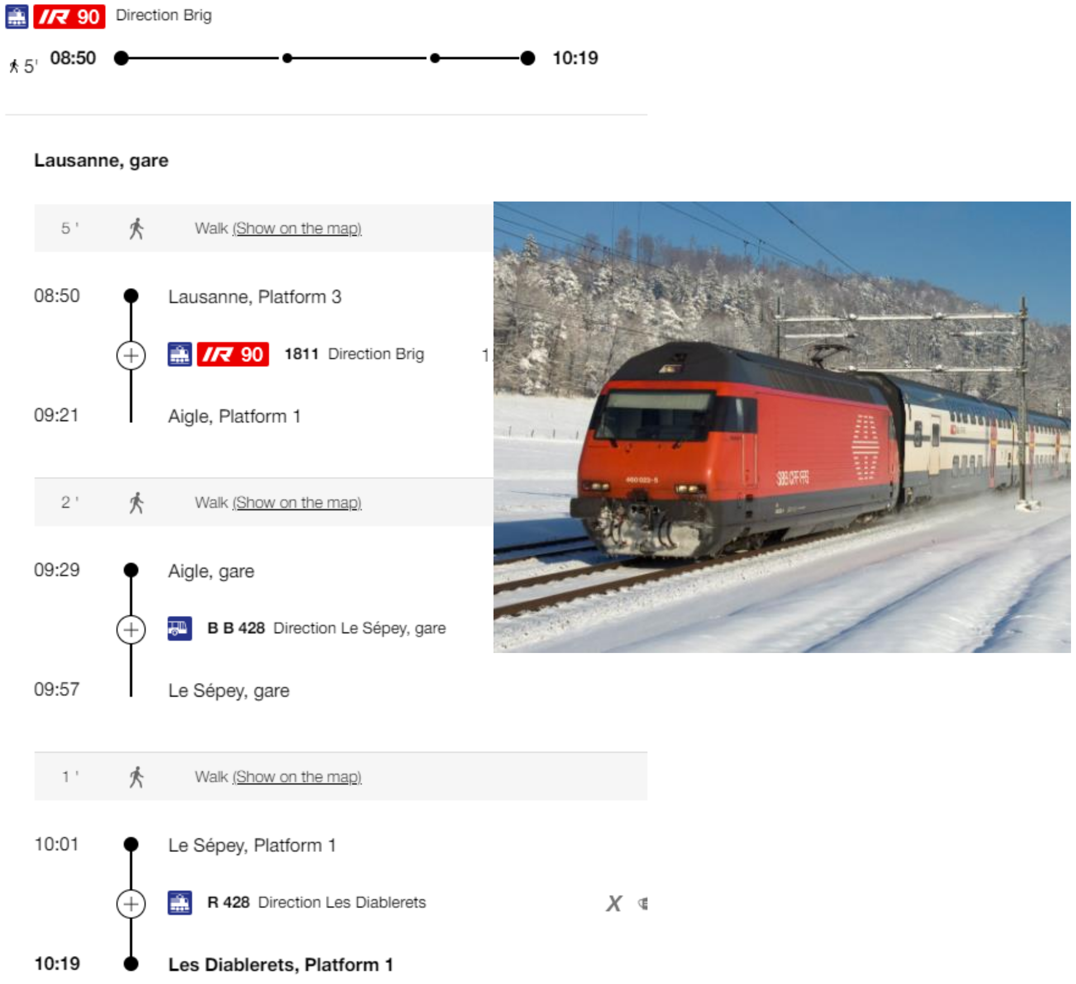


Dynamic scheduling (our HLS approach): decide at runtime when each operation executes



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

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

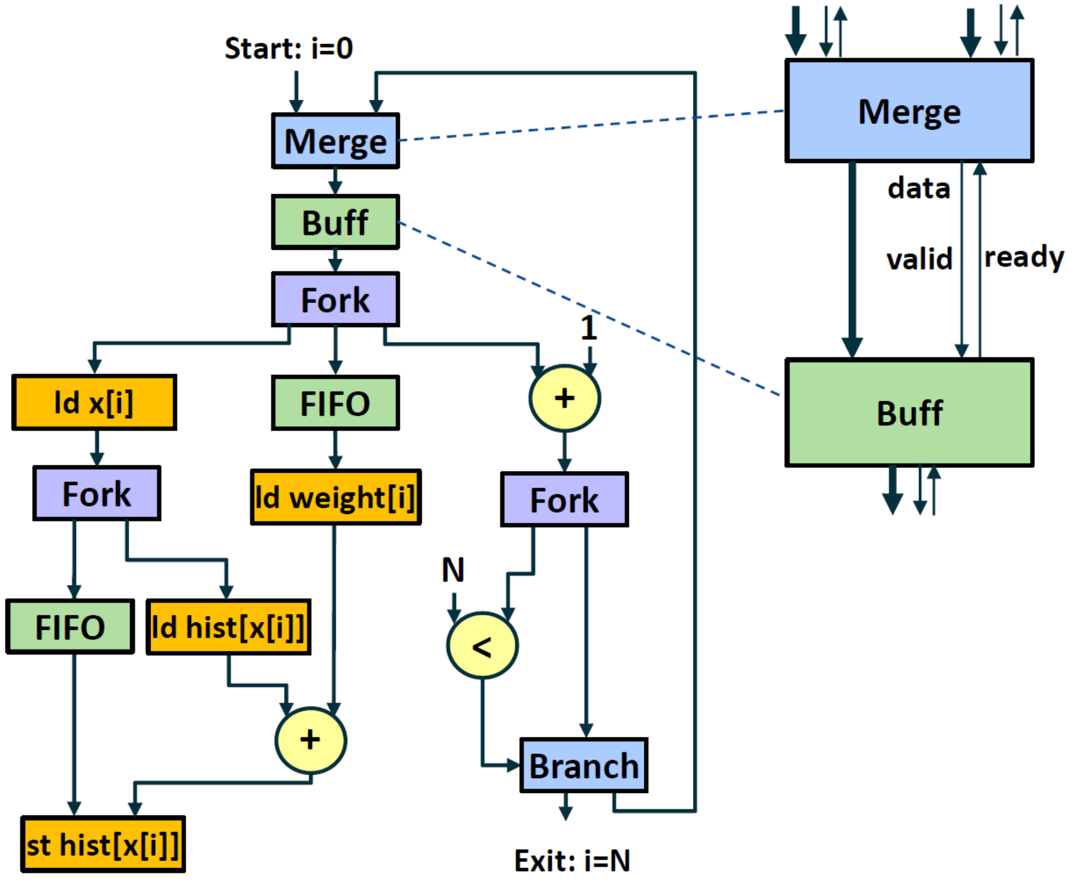
1 ' Walk (Show on the map)

10:01 Le Sépey, Platform 1

R 428 Direction Les Diablerets

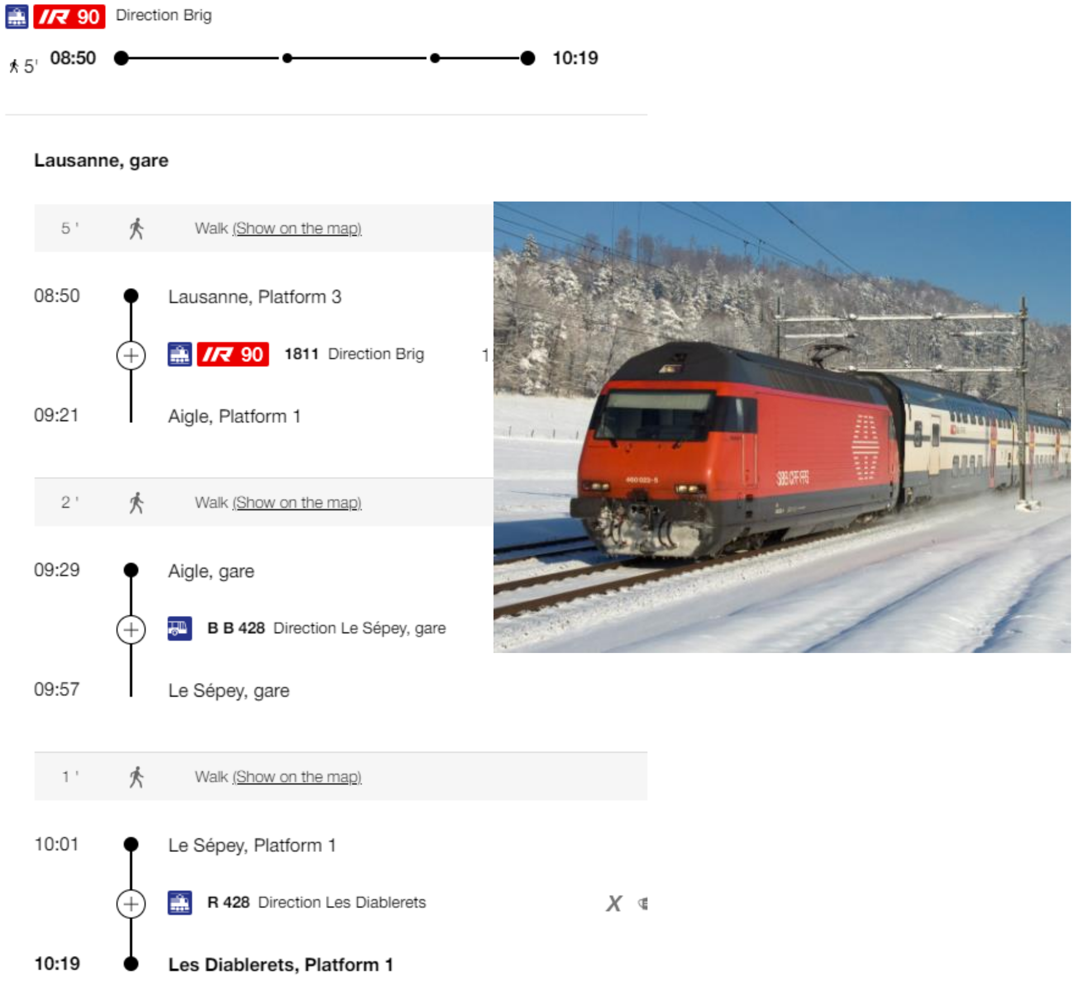
10:19 Les Diablerets, Platform 1

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

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

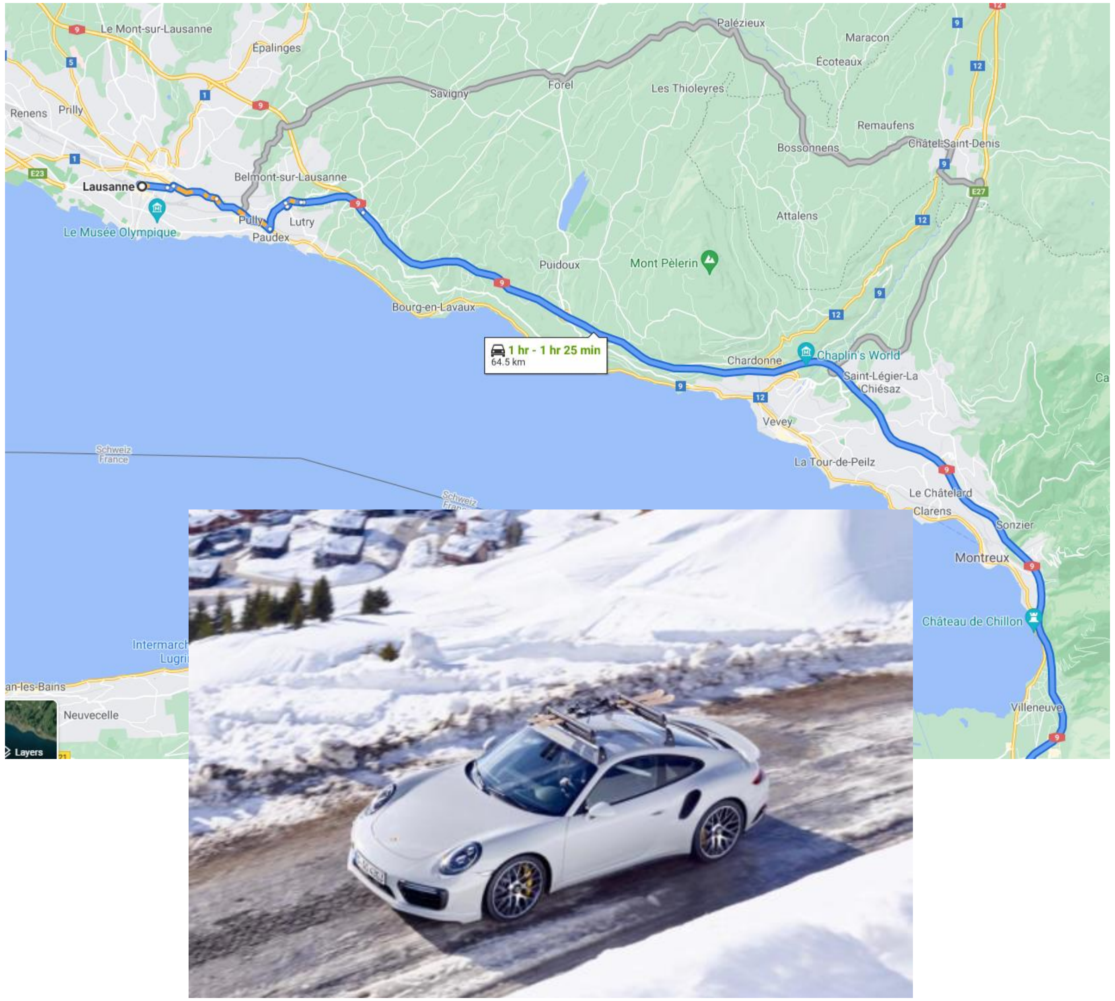
1' Walk (Show on the map)

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



Dataflow Circuits

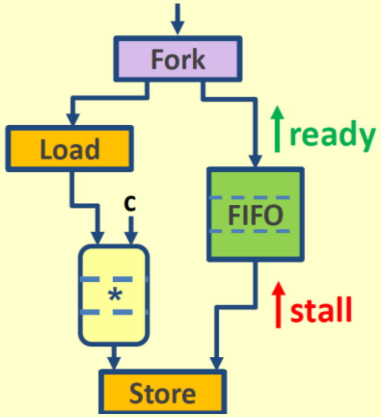
- **Asynchronous circuits**: operators triggered when inputs are available
  - Budiu et al. Dataflow: A complement to superscalar. ISPASS'05.
- Dataflow, latency-insensitive, elastic: the **synchronous** version of it
  - Cortadella et al. Synthesis of synchronous elastic architectures. DAC'06.
  - Carloni et al. Theory of latency-insensitive design. TCAD'01.
  - Jacobson et al. Synchronous interlocked pipelines. ASYNC'02.
  - Vijayaraghavan and Arvind. Bounded dataflow networks and latency-insensitive circuits. MEMOCODE'09.

High-level synthesis of dataflow circuits

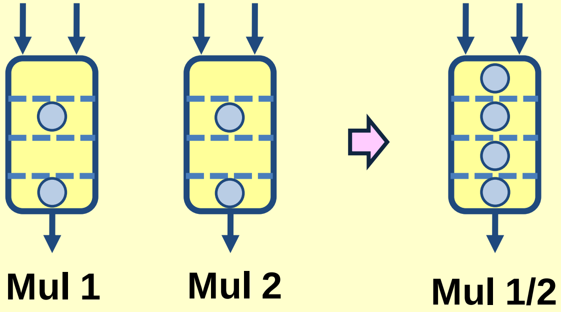
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining



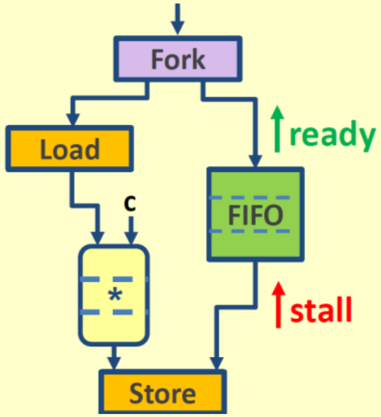
Resource sharing



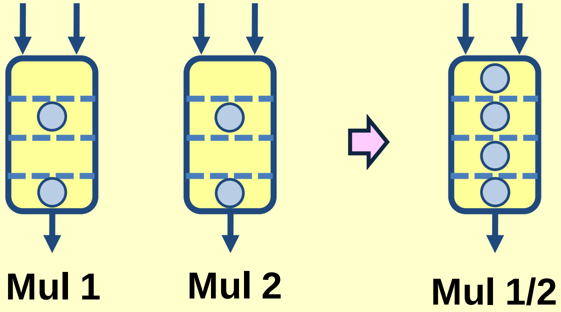
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

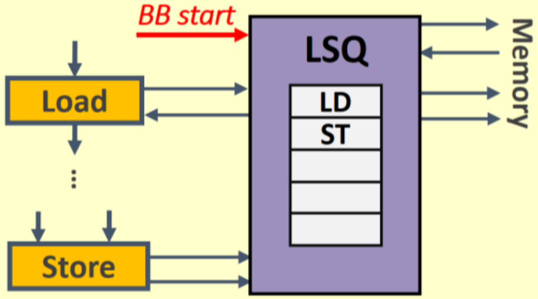


Resource sharing

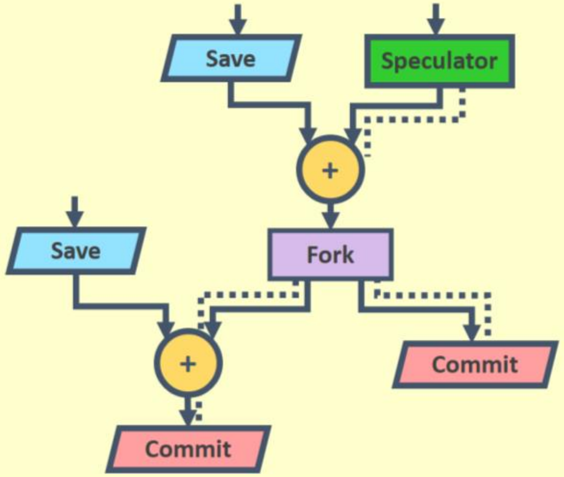


Reaping the benefits of dynamic scheduling

Out-of-order memory

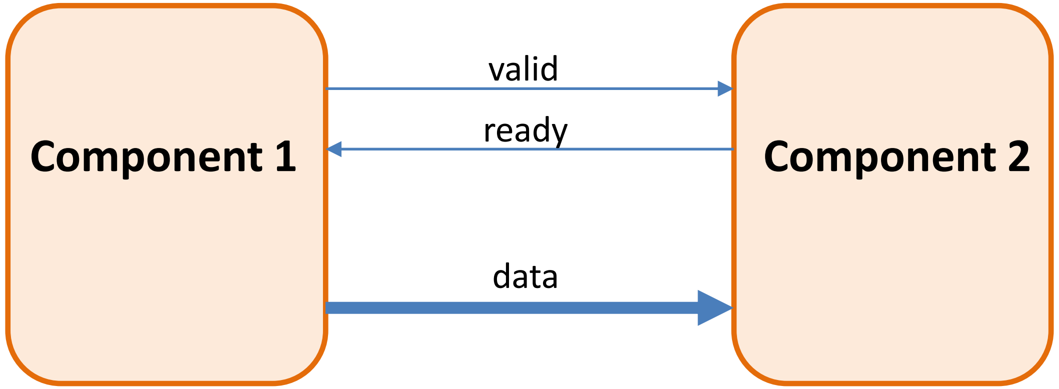


Speculative execution



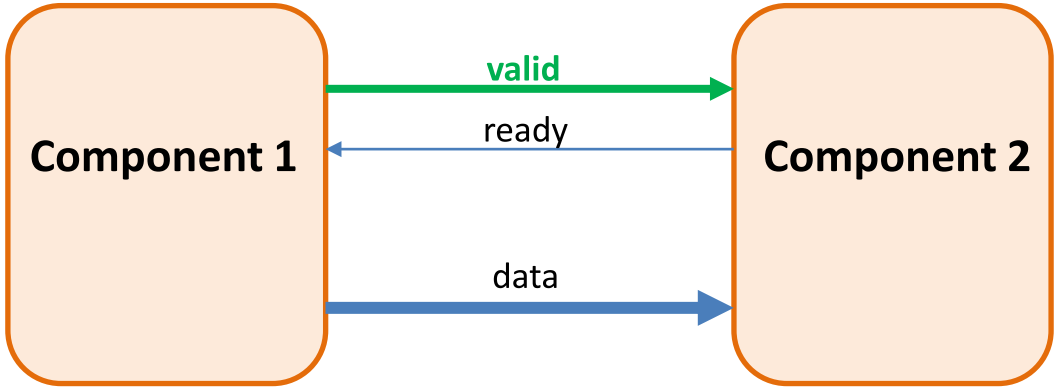
Dataflow Circuits

- 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



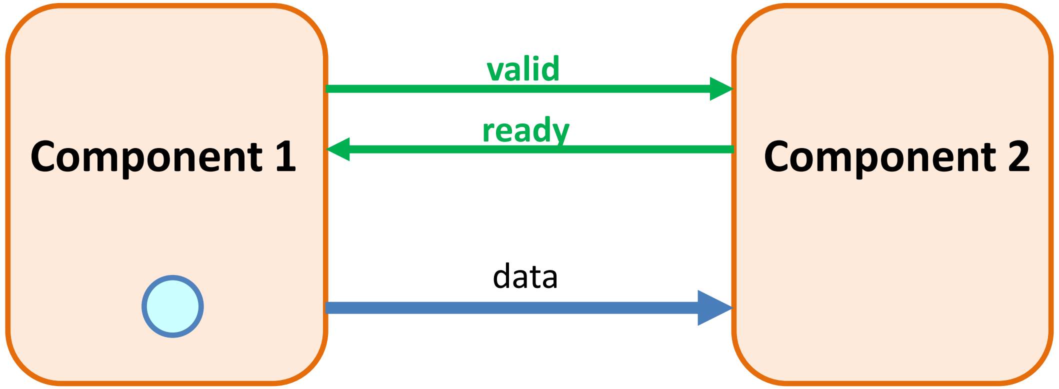
Dataflow Circuits

- 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

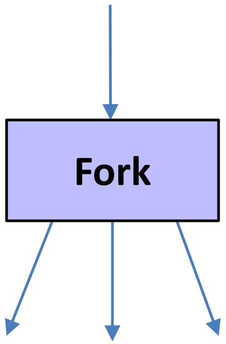


Dataflow Circuits

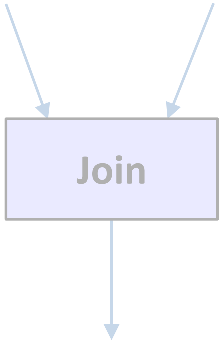
- We use the **SELF (Synchronous ELastic Flow)** protocol
  - Cortadella et al. Synthesis of synchronous elastic architectures. DAC’06.
- Every component communicates via a pair of handshake signals
- **Make scheduling decisions at runtime**
  - As soon as all conditions for execution are satisfied,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



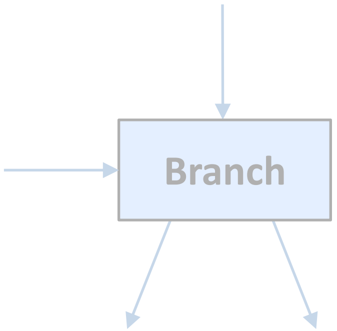
Dataflow Components



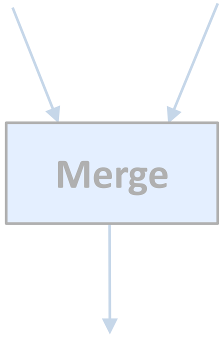
Fork



Join

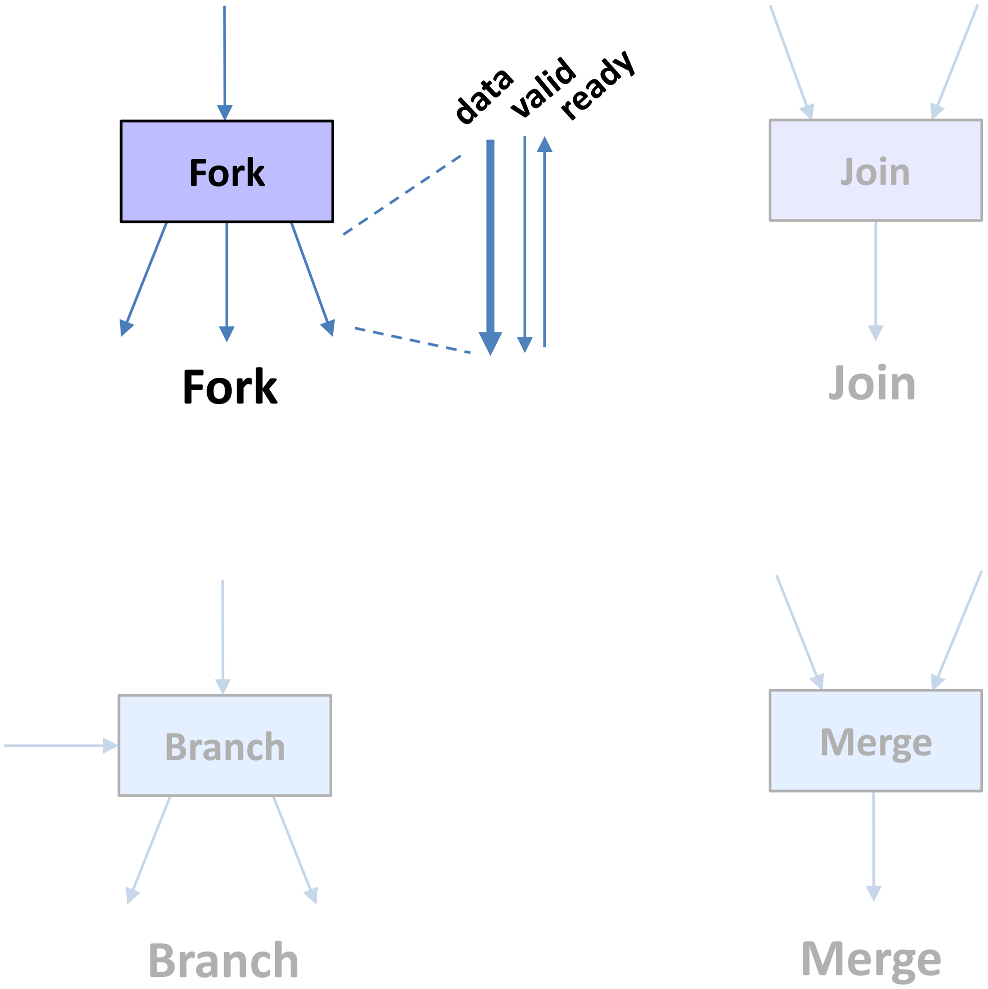


Branch

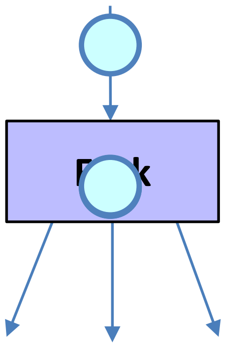


Merge

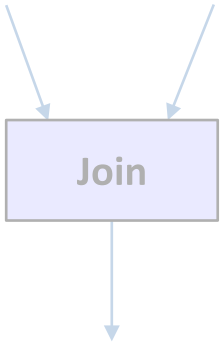
Dataflow Components



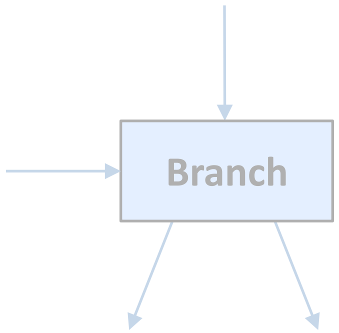
Dataflow Components



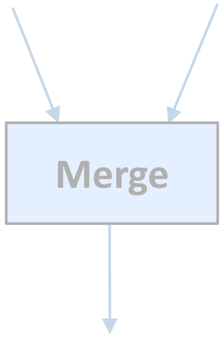
Fork



Join

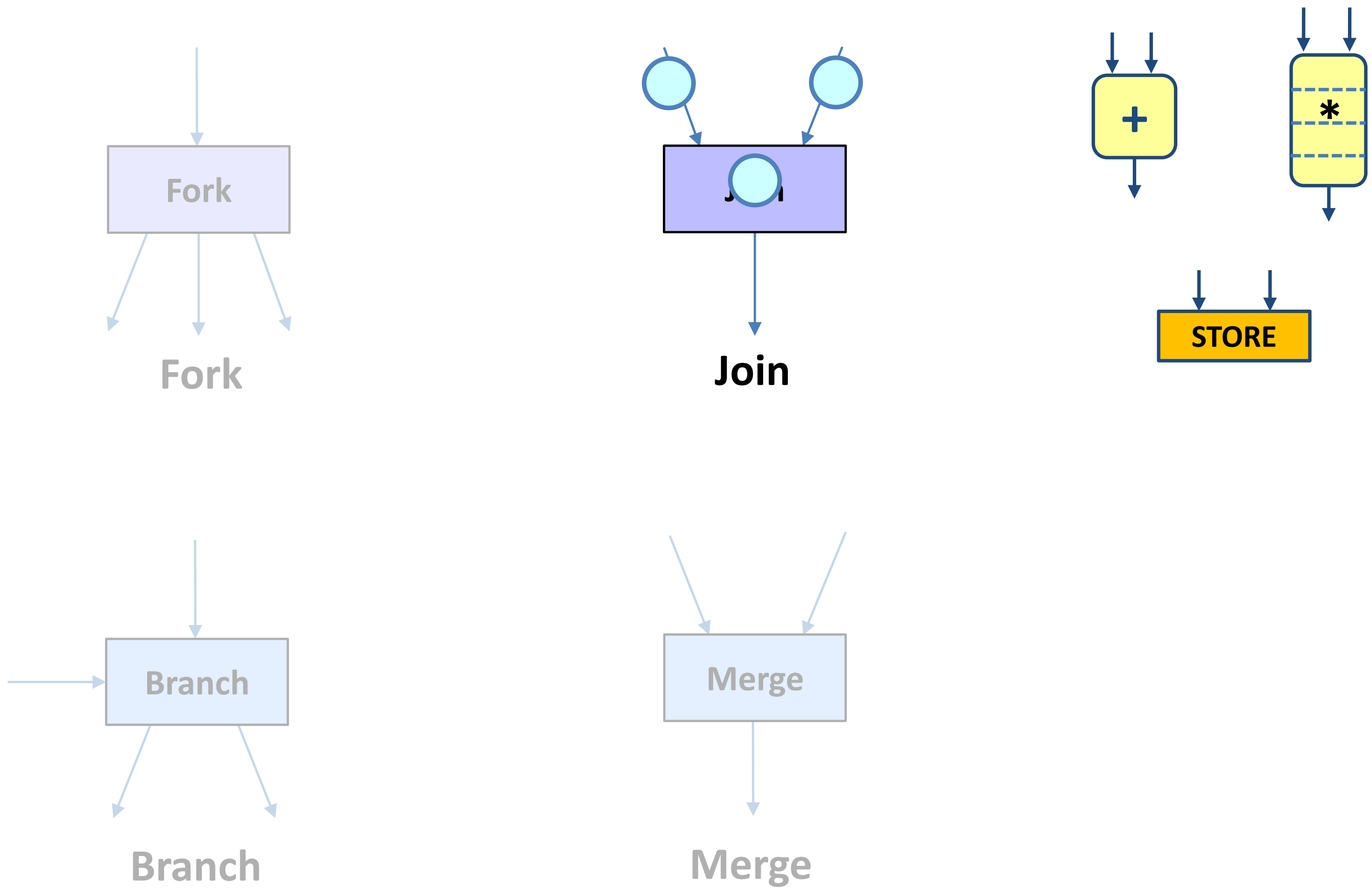


Branch

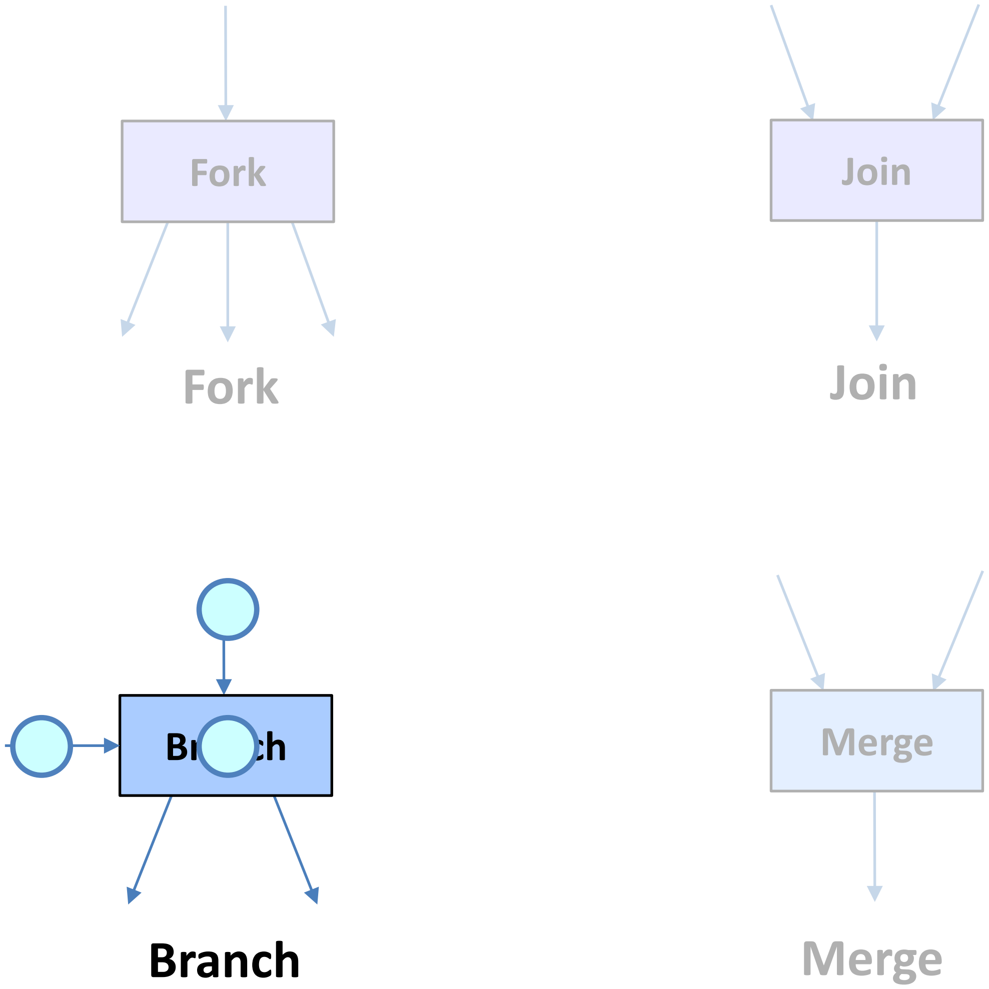


Merge

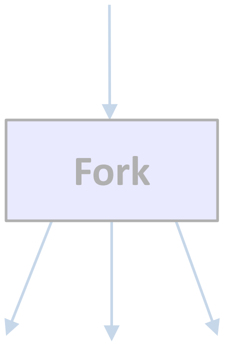
Dataflow Components



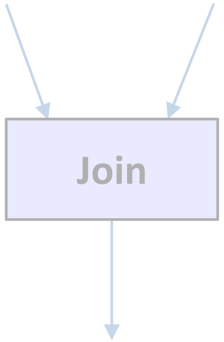
Dataflow Components



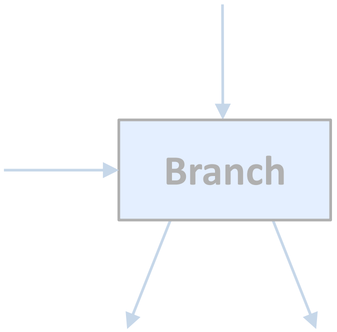
Dataflow Components



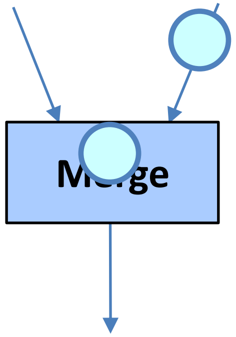
Fork



Join

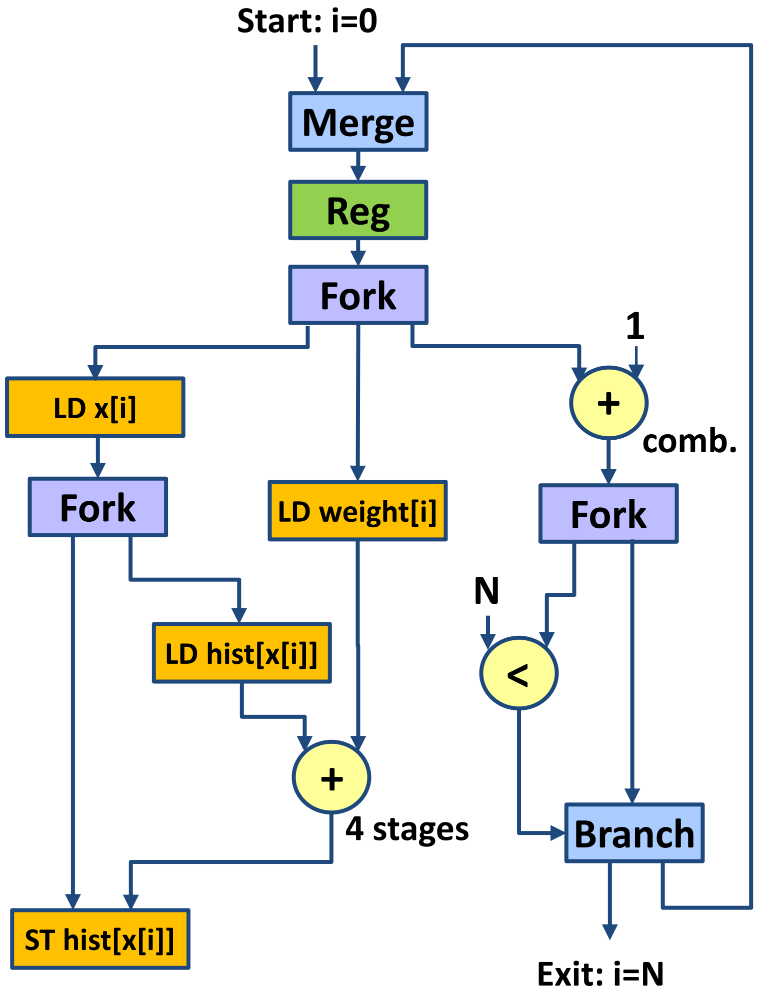


Branch

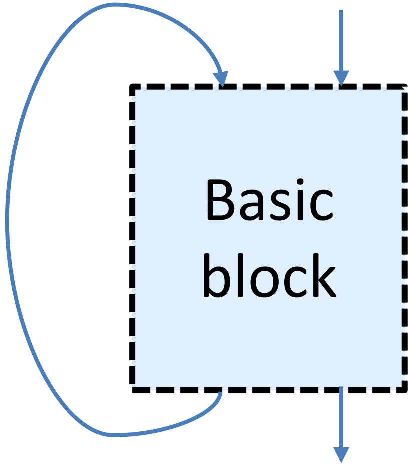


Merge

From Program to Dataflow Circuit

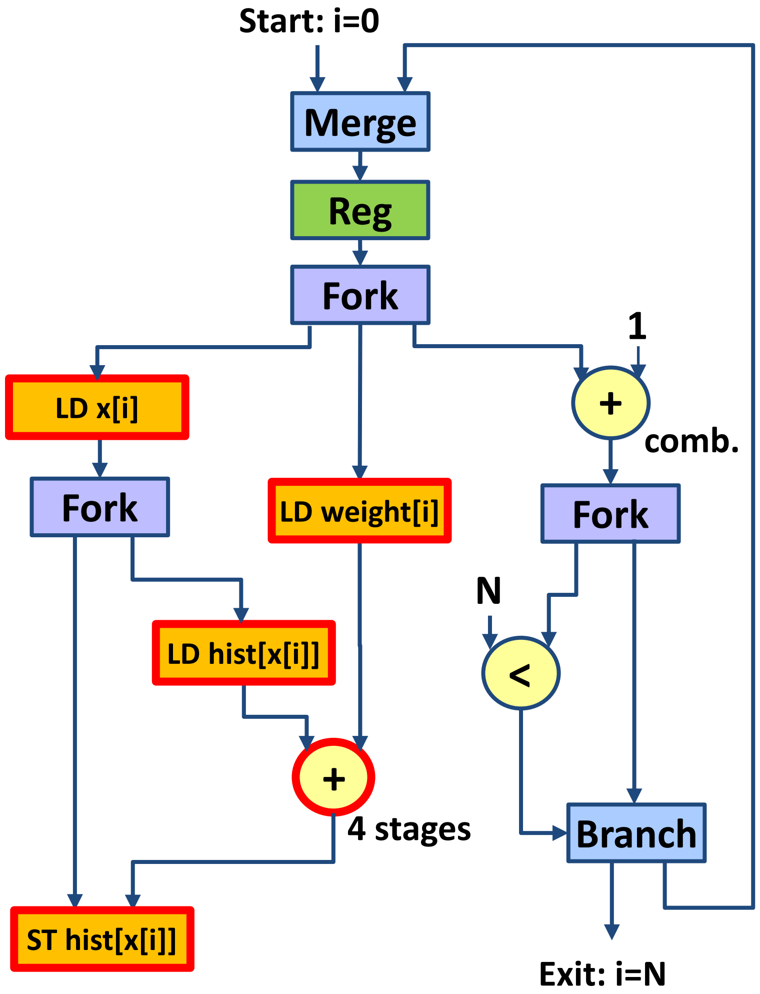


```
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
```



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

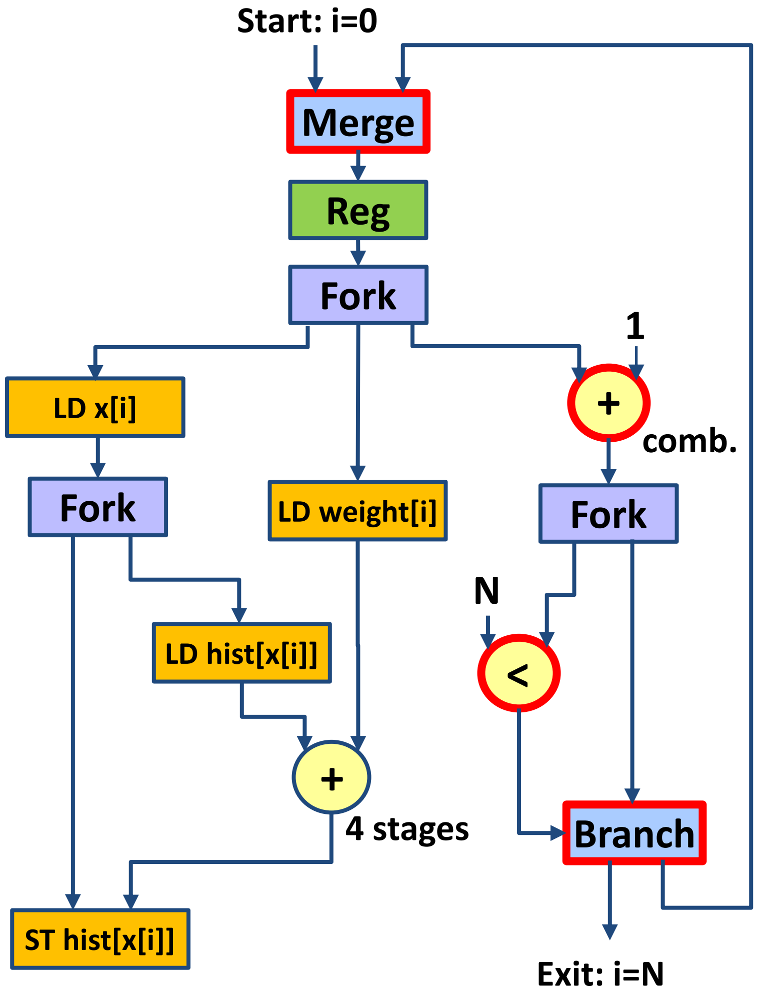
From Program to Dataflow Circuit



```
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

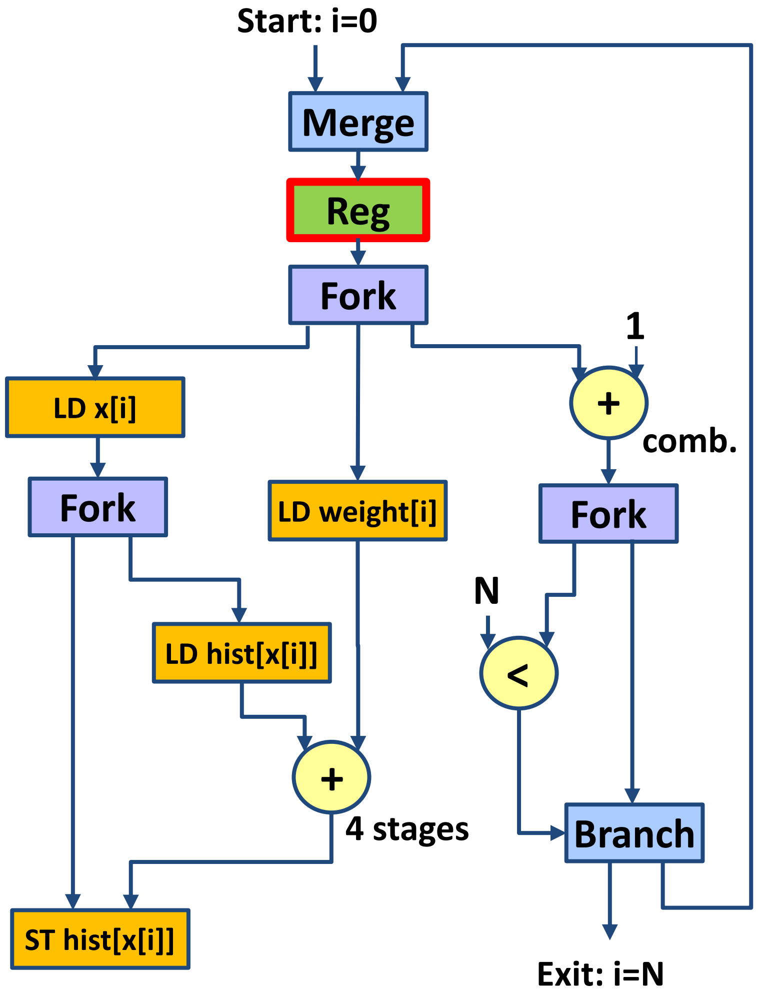
From Program to Dataflow Circuit



```
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

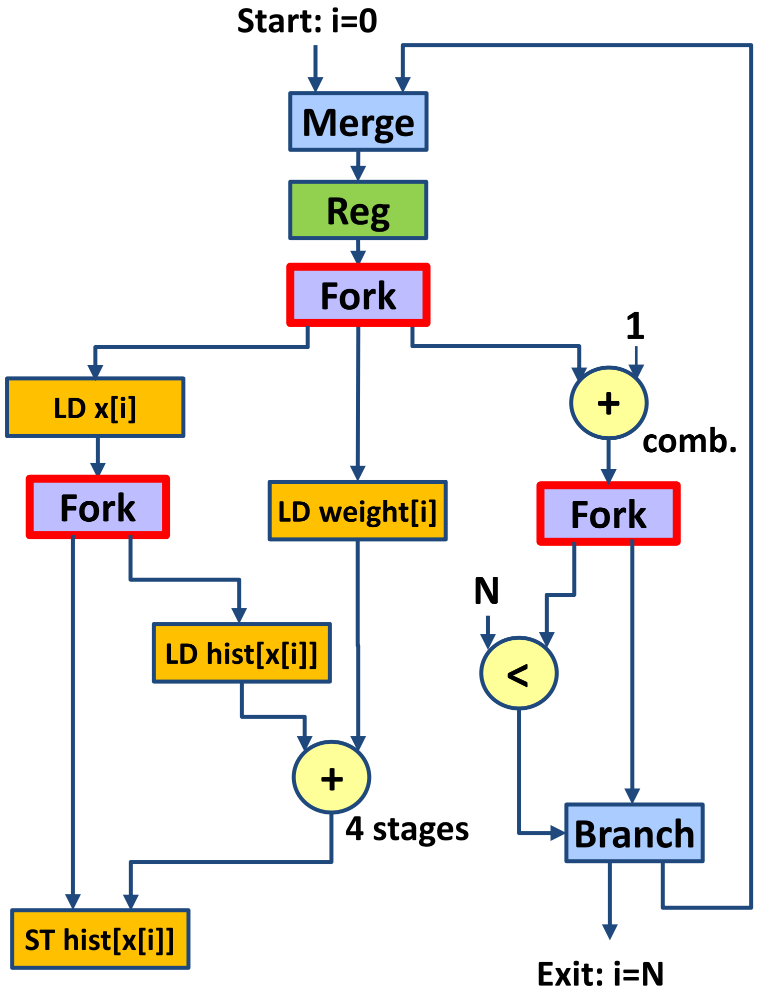
From Program to Dataflow Circuit



```
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

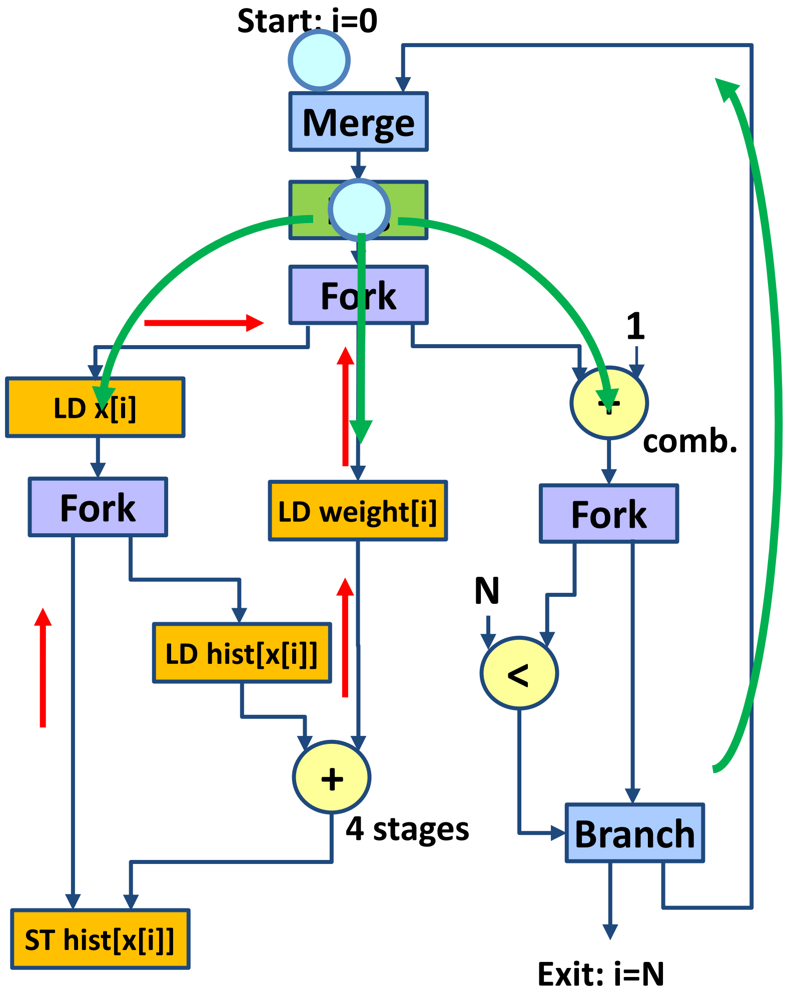
From Program to Dataflow Circuit



```
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

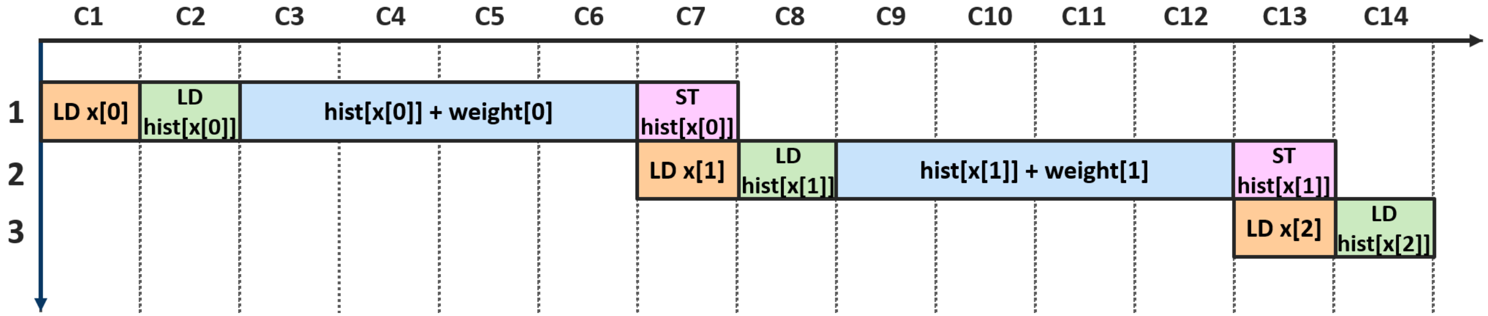
From Program to Dataflow Circuit



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

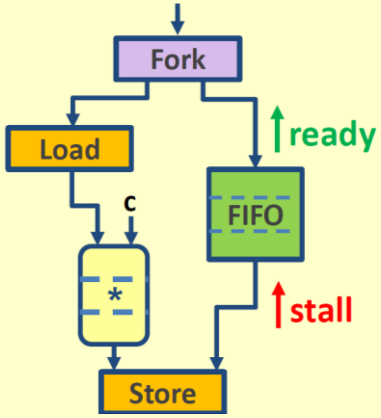


Backpressure from slow paths prevents pipelining

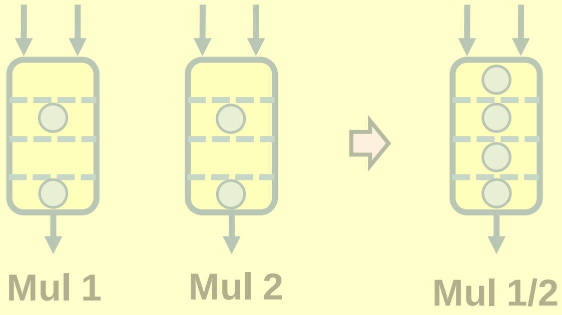
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

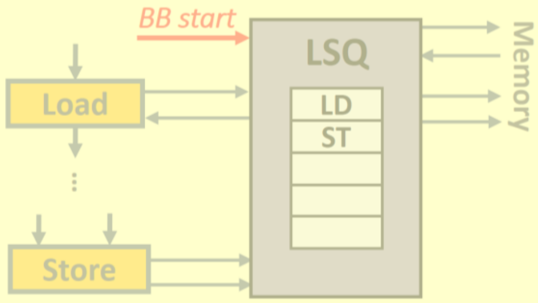


Resource sharing

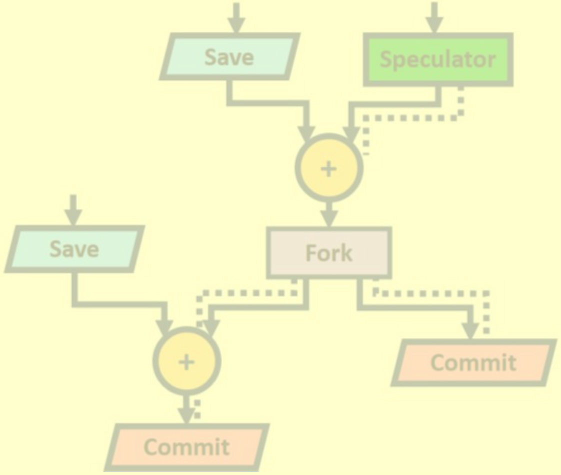


Reaping the benefits of dynamic scheduling

Out-of-order memory

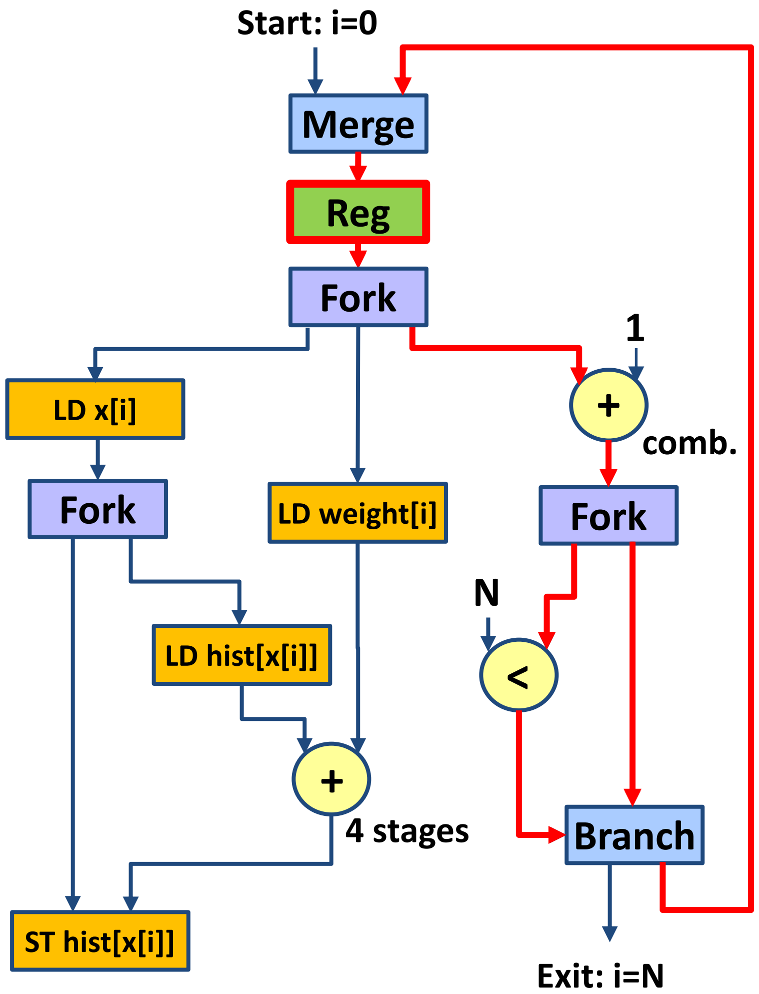


Speculative execution



Inserting Buffers

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

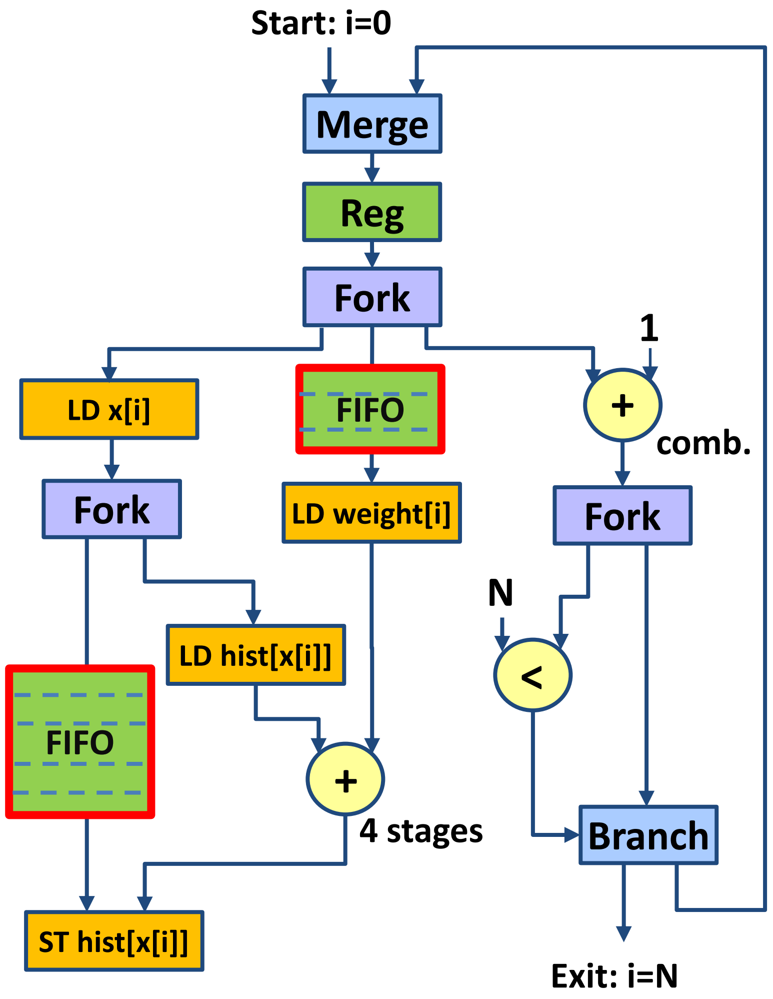


Buffers as registers to break combinational paths

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];
}
```

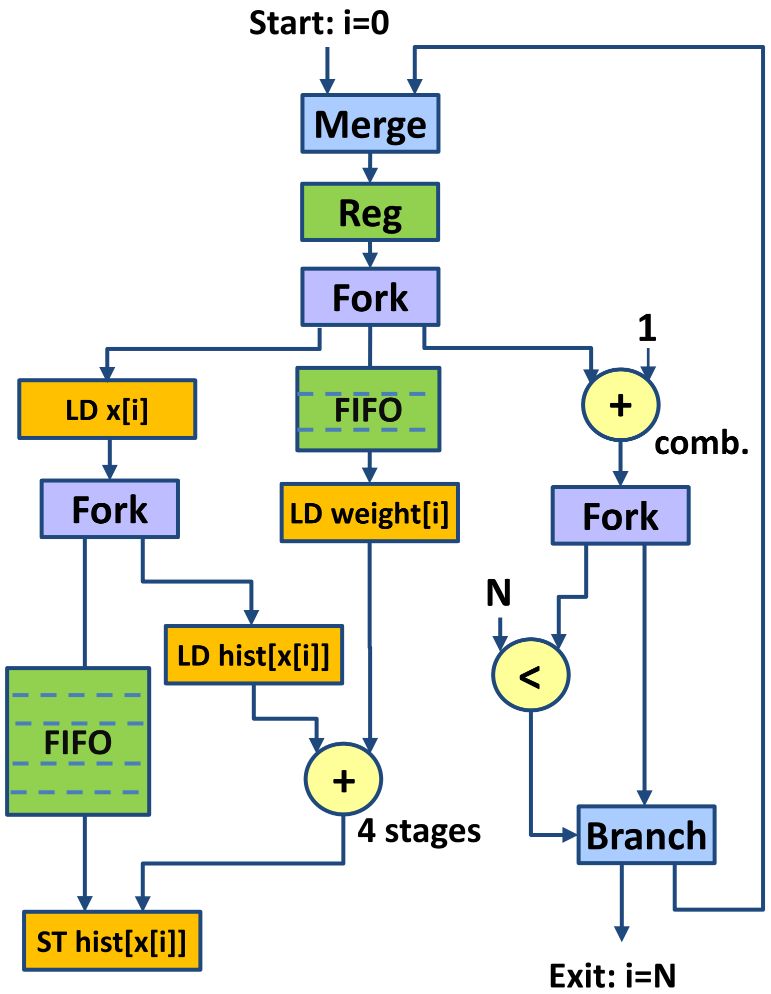


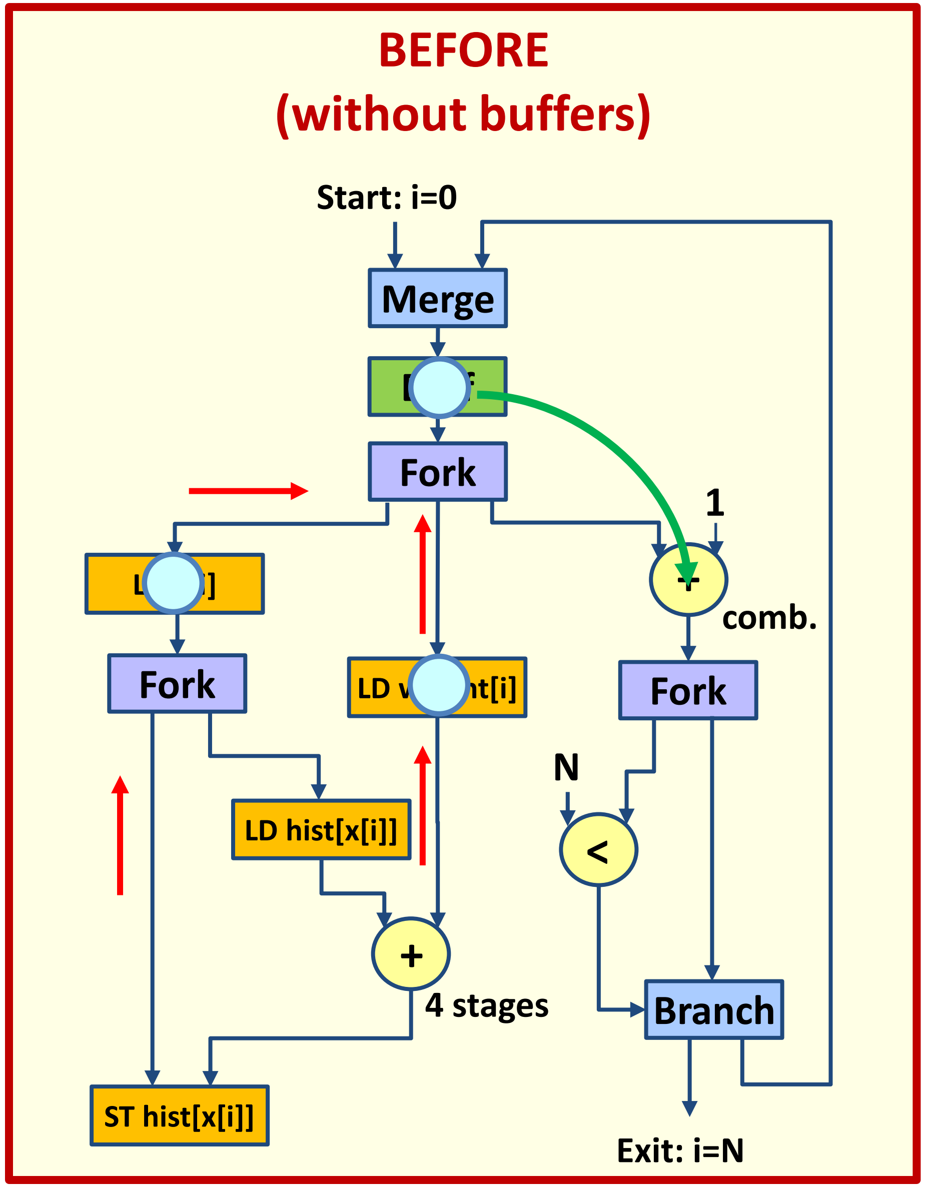
Buffers as FIFOs to regulate throughput

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];
}
```

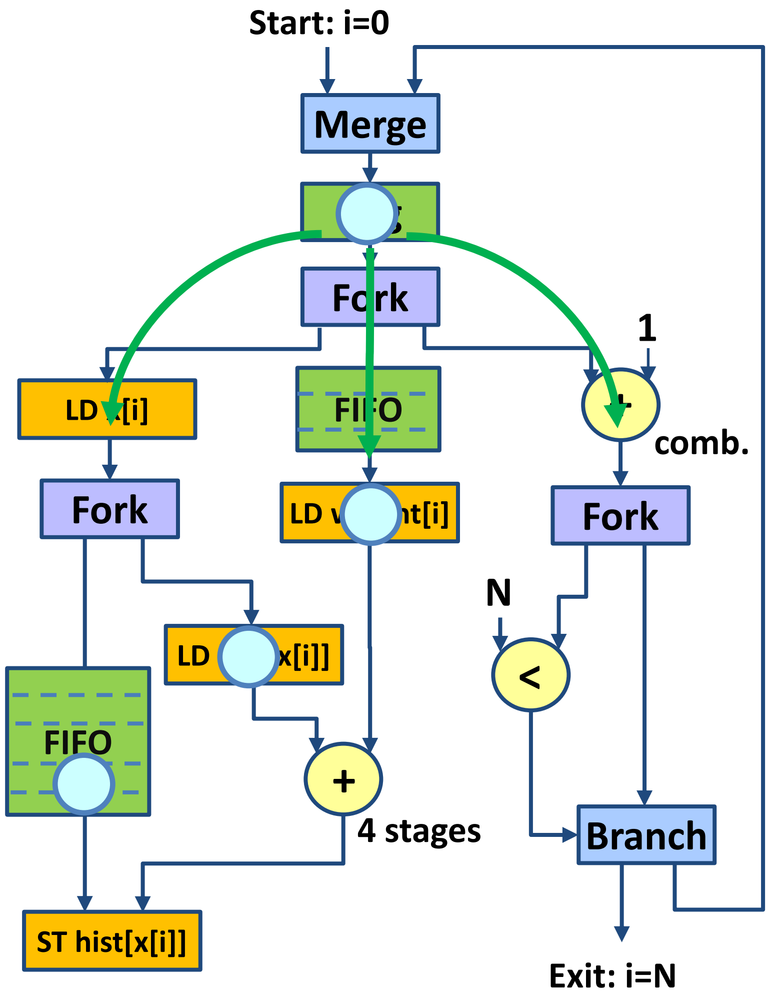




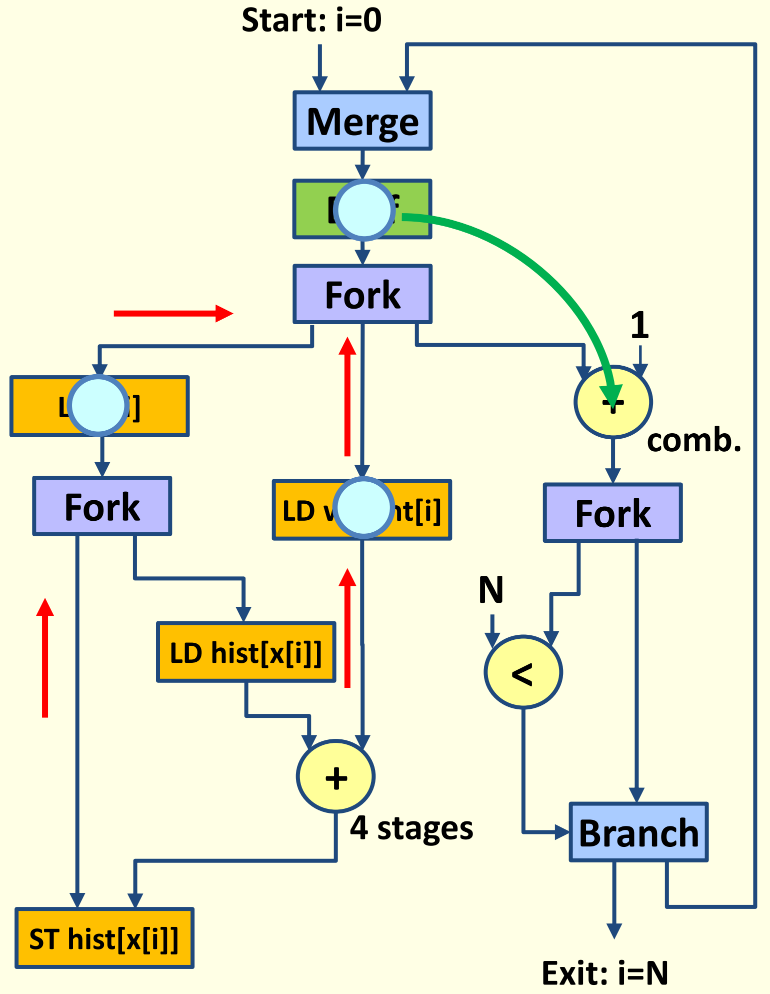
Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**
Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022

Inserting Buffers

**NOW
(with buffers)**

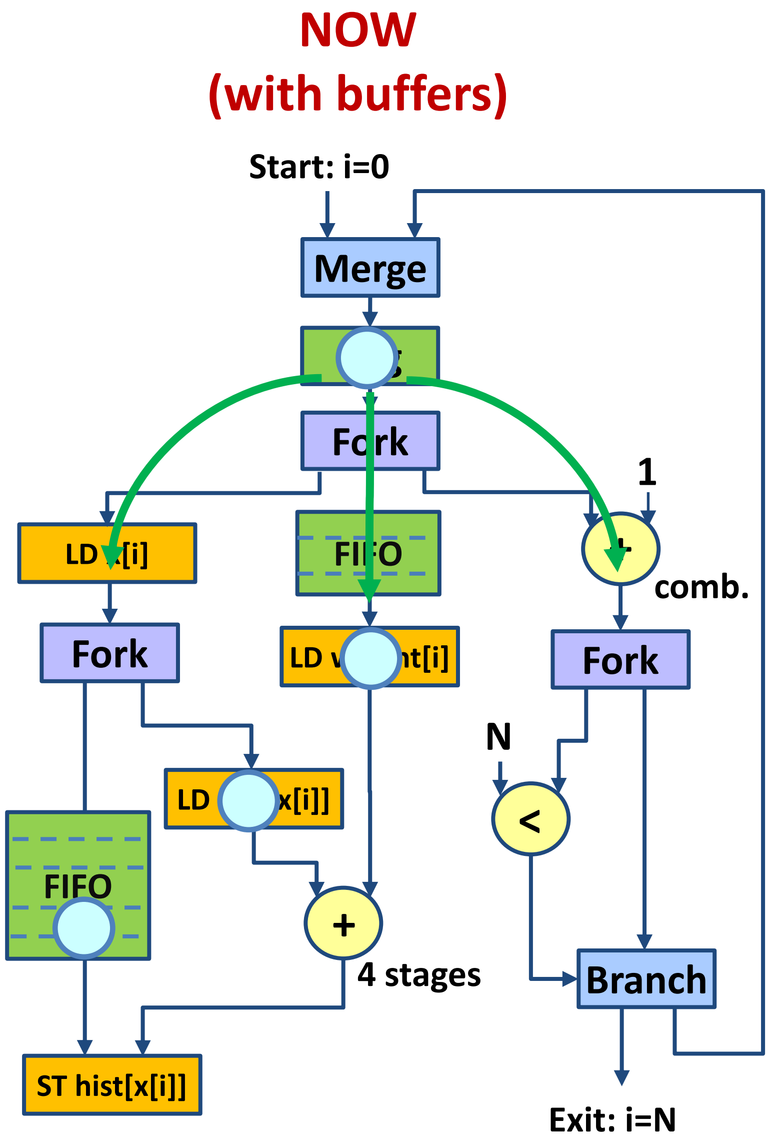


**BEFORE
(without buffers)**



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

Inserting Buffers



Mixed integer linear programming (MILP) model based on **Petri net theory**

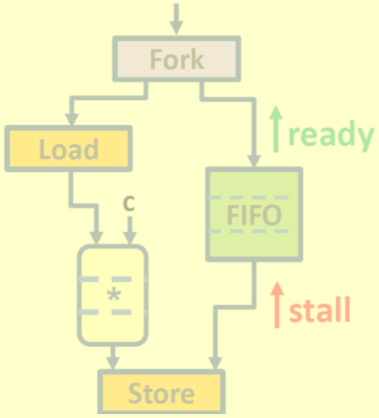
- Analyze token flow through the circuit
- Determine **buffer placement and sizing**
- **Maximize throughput** for a target clock period

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

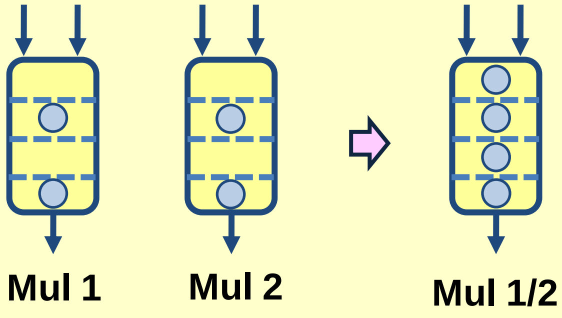
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

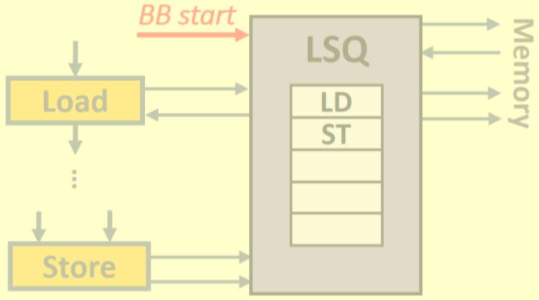


Resource sharing

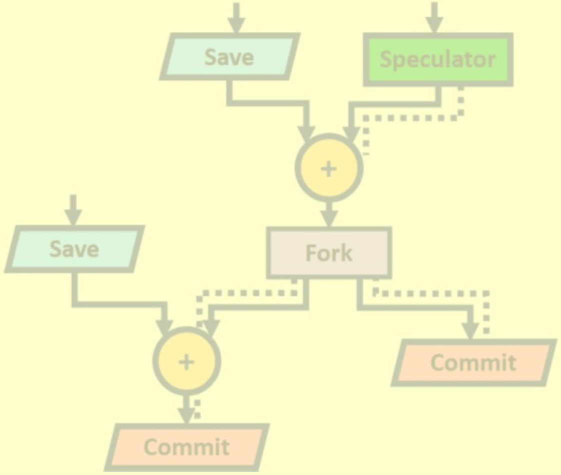


Reaping the benefits of dynamic scheduling

Out-of-order memory



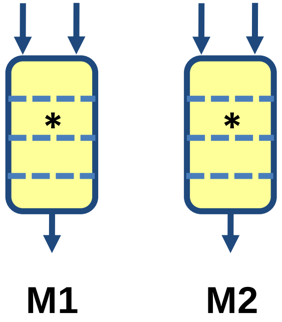
Speculative execution



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;
}
```

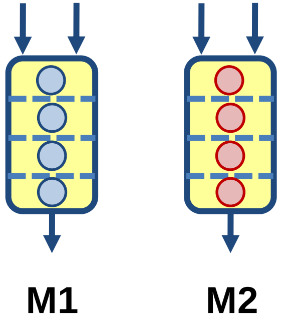


Josipović, Marmet, Guerrieri, and Ienne. Resource Sharing in Dataflow Circuits. FCCM 2022. **Best Paper Award Nominee**

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

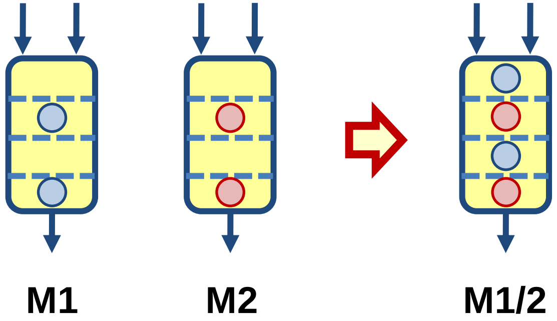
Use MILP (performance optimization) information to decide what to share

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;
}
```



Units underutilized (low throughput)

Sharing possible without damaging throughput

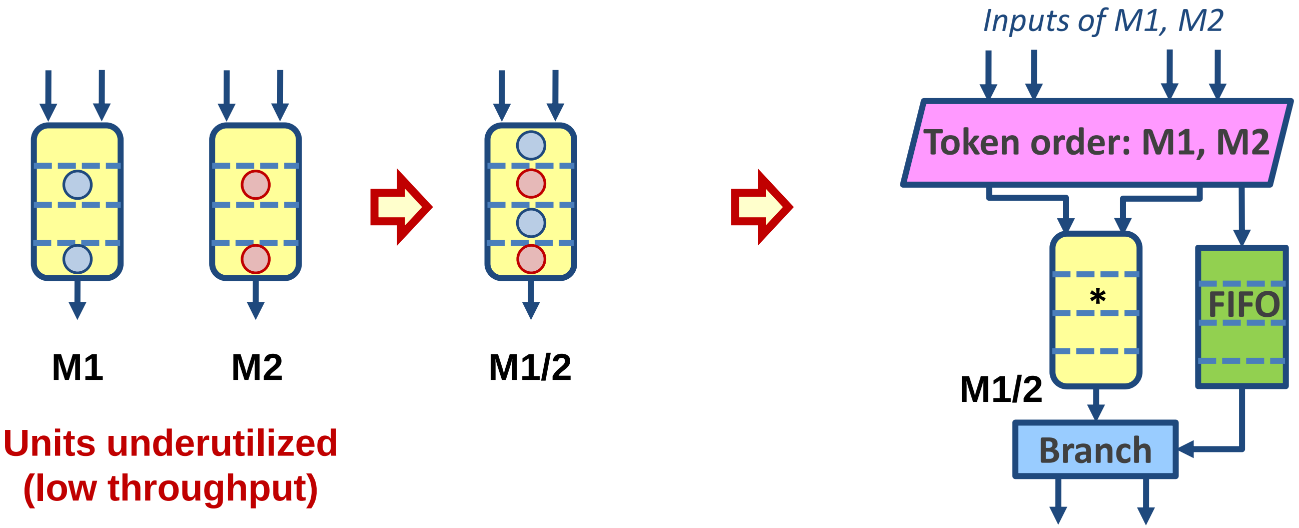
Use MILP (performance optimization) information to decide what to share

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;
}
```



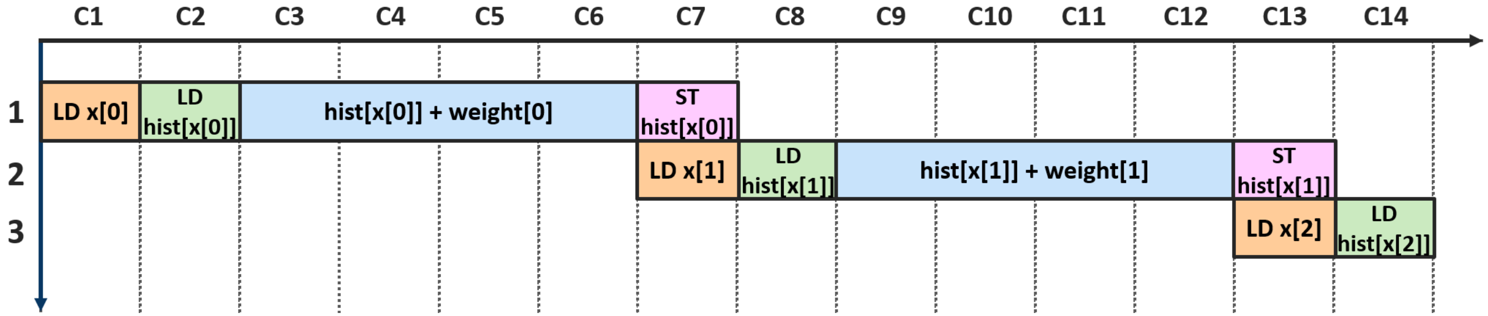
Units underutilized (low throughput)

Sharing mechanism for deadlock-free execution

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

Inserting Buffers

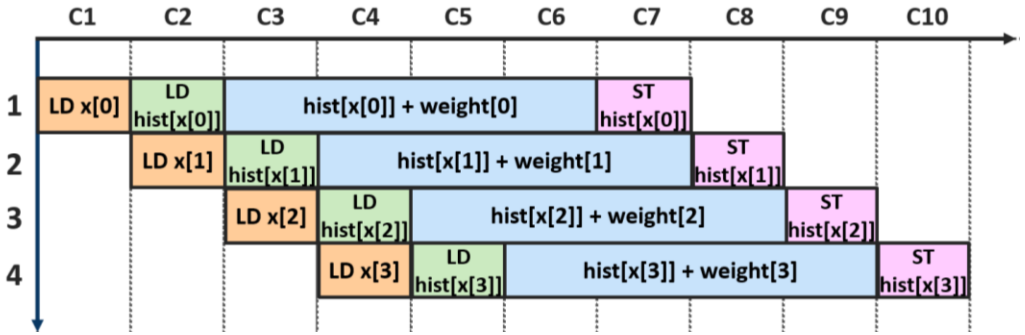
```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```



Backpressure from slow paths prevents pipelining

Inserting Buffers

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



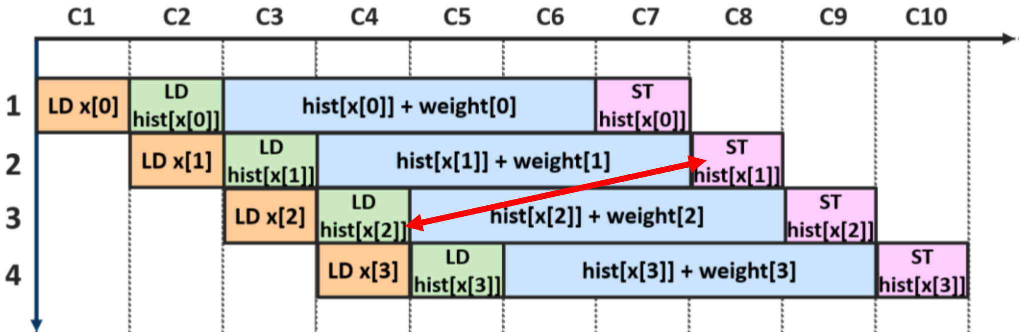
Buffers for high throughput

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



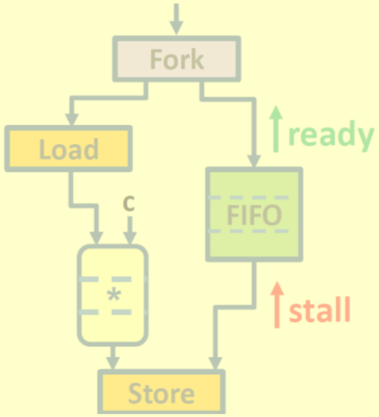
RAW dependency
not honored!

What about memory?

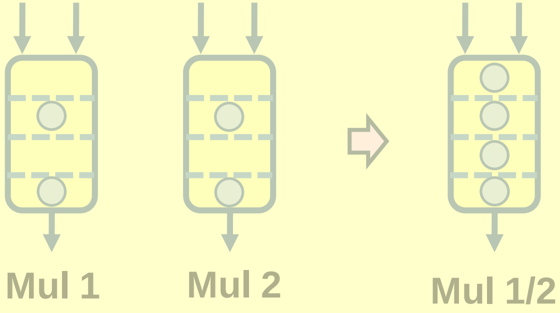
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

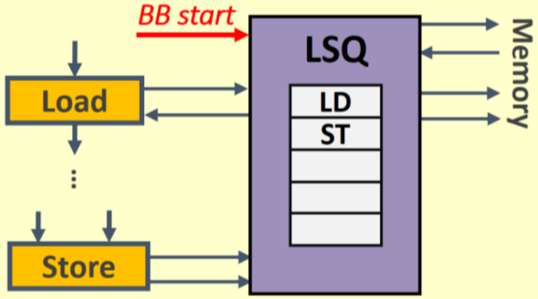


Resource sharing

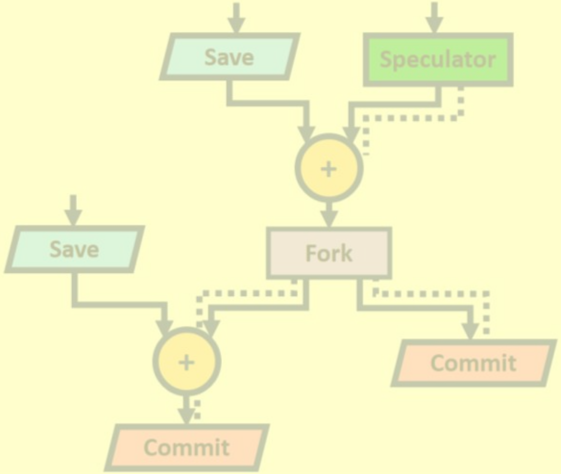


Reaping the benefits of dynamic scheduling

Out-of-order memory



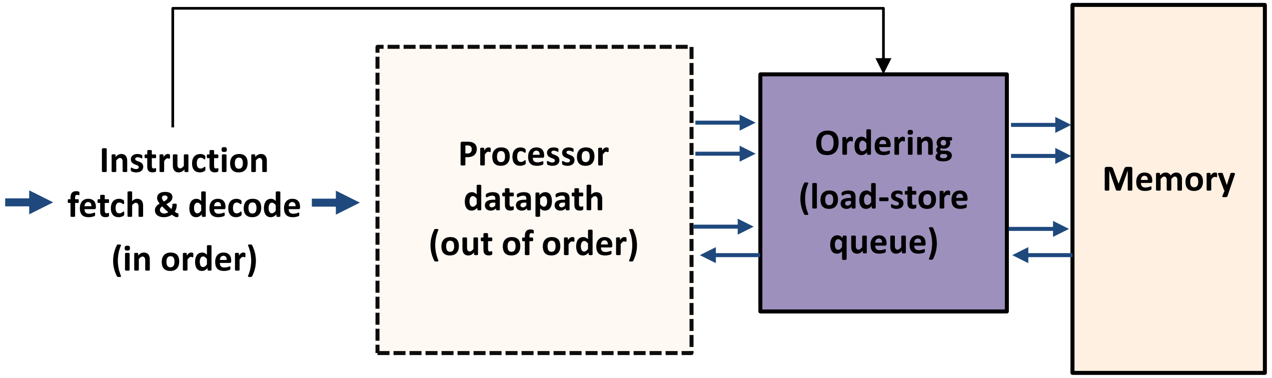
Speculative execution



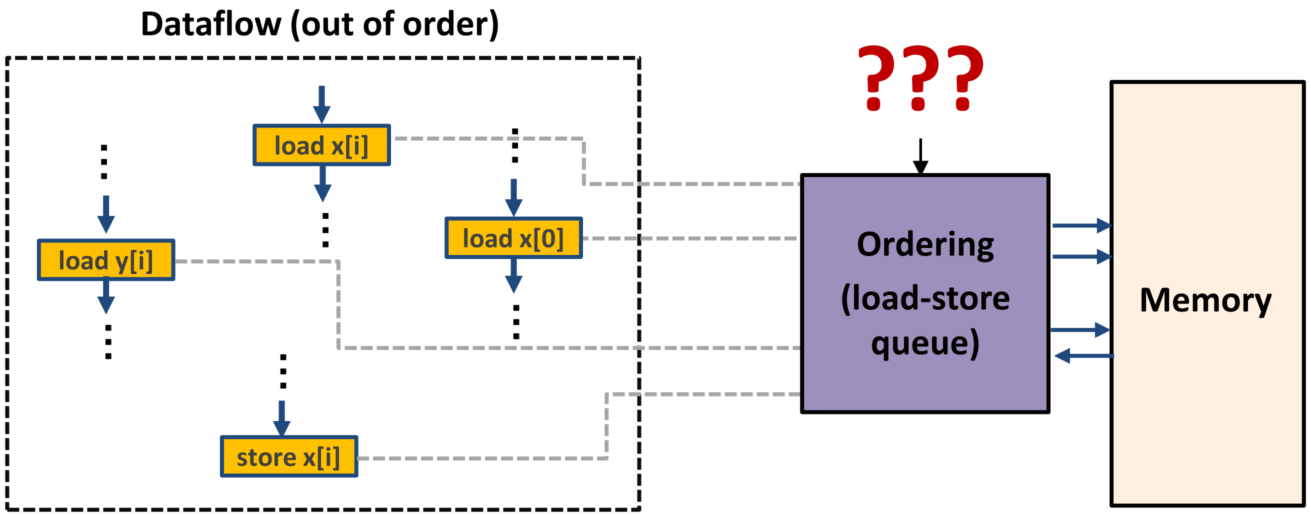
We Need a Load-Store Queue (LSQ)!

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

```
loop: lw $t2, 0($t4)
      lw $t3, 100($t4)
      mul $t5, $t2, $t3
      addi $t5, $t5, $t1
      sw $t5, 100($t4)
      addi $t1, $t1, 4
      bne $t6, $t1, loop
```



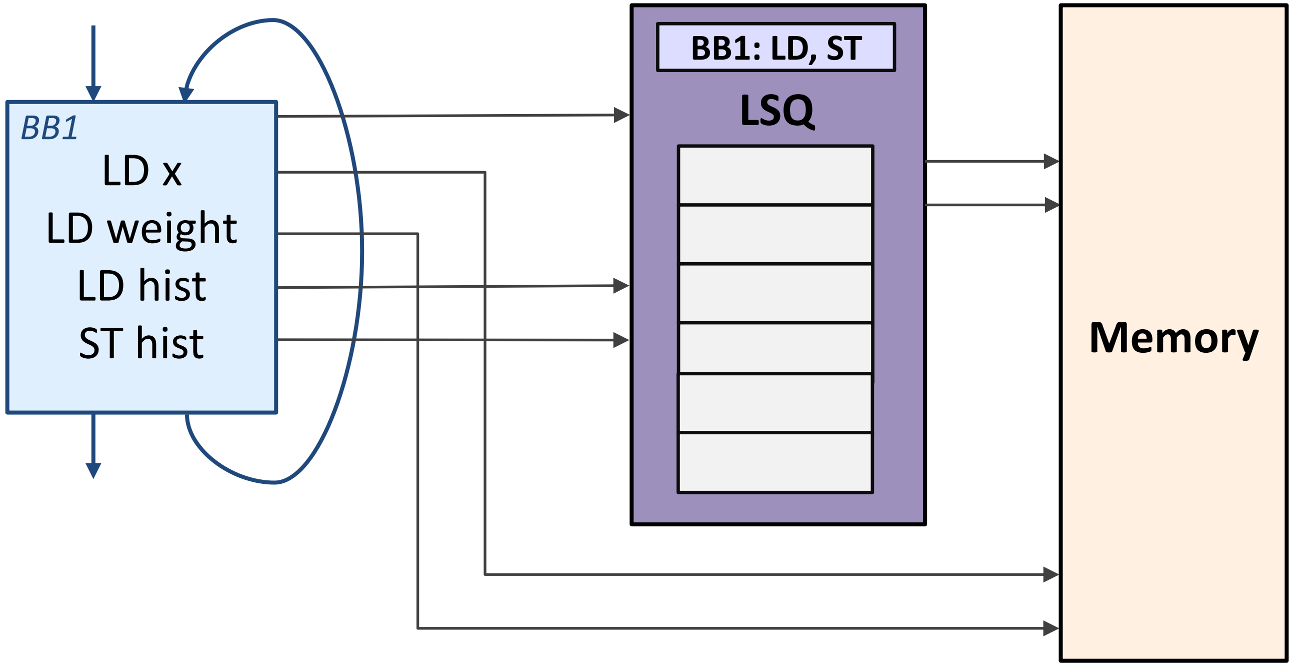
- Dataflow circuits have **no notion of program order**



How to supply program order to the LSQ?

LSQ Allocation

- An LSQ for dataflow circuits whose only difference is in the **allocation policy**:
  - **Static knowledge** of memory access program order inside each basic block
  - **Dynamic knowledge** of the sequence of basic blocks **from the dataflow circuit**

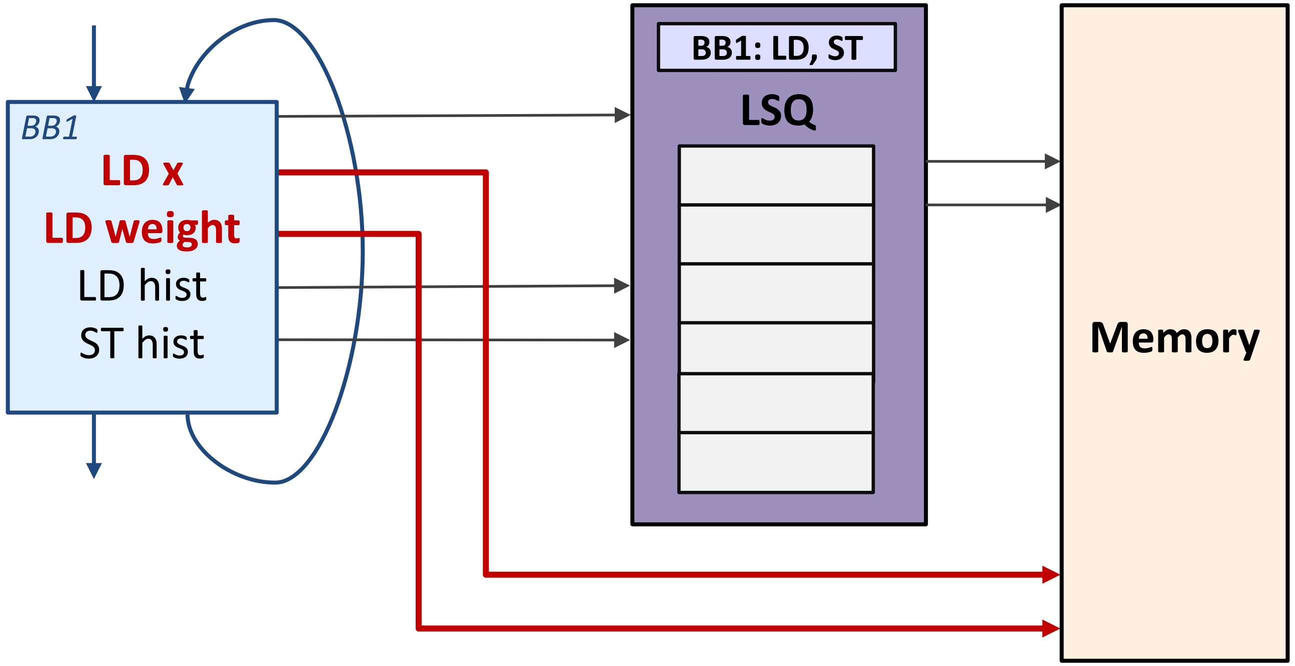


```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Brisk, and Ienne. An Out-of-Order Load-Store Queue for Spatial Computing. CASES 2017 **Best Paper Award Nominee**
Josipović, Bhattacharrya, Guerrieri, and Ienne. Shrink It or Shed It! Minimize the Use of LSQs in Dataflow Designs. FPT 2019

LSQ Allocation

- An LSQ for dataflow circuits whose only difference is in the **allocation policy**:
  - **Static knowledge** of memory access program order inside each basic block
  - **Dynamic knowledge** of the sequence of basic blocks **from the dataflow circuit**

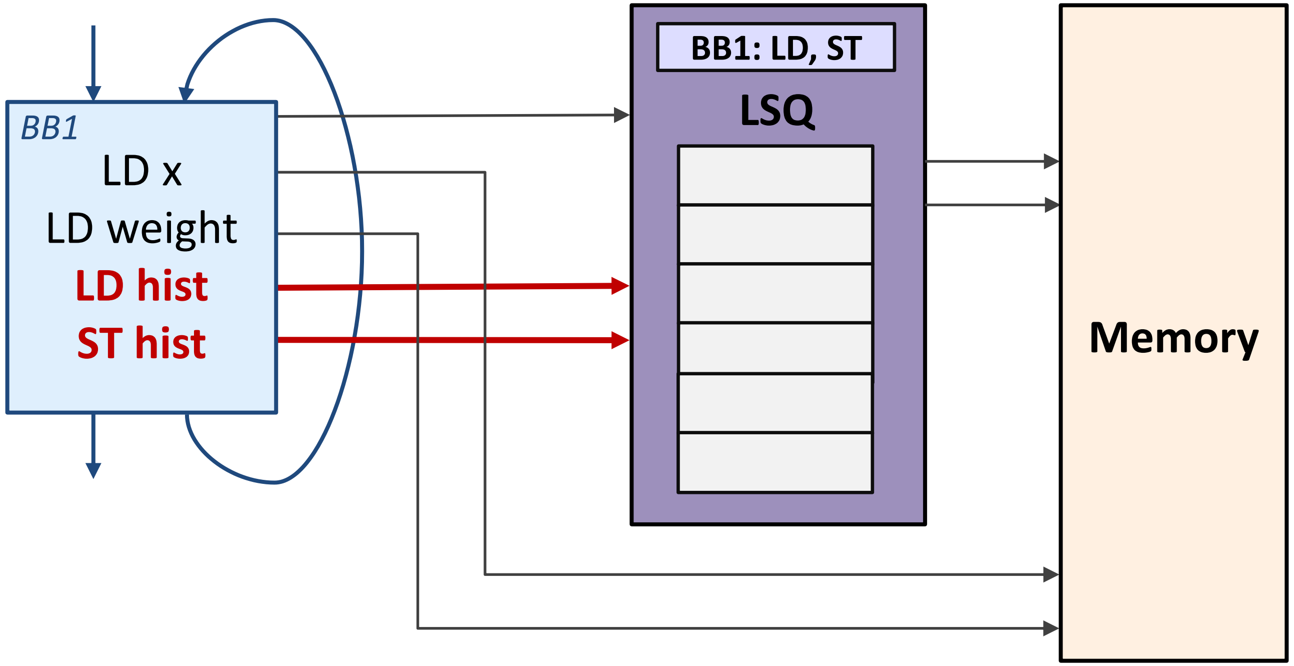


```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Brisk, and Ienne. An Out-of-Order Load-Store Queue for Spatial Computing. CASES 2017 **Best Paper Award Nominee**
Josipović, Bhattacharyya, Guerrieri, and Ienne. Shrink It or Shed It! Minimize the Use of LSQs in Dataflow Designs. FPT 2019

LSQ Allocation

- An LSQ for dataflow circuits whose only difference is in the **allocation policy**:
  - **Static knowledge** of memory access program order inside each basic block
  - **Dynamic knowledge** of the sequence of basic blocks **from the dataflow circuit**

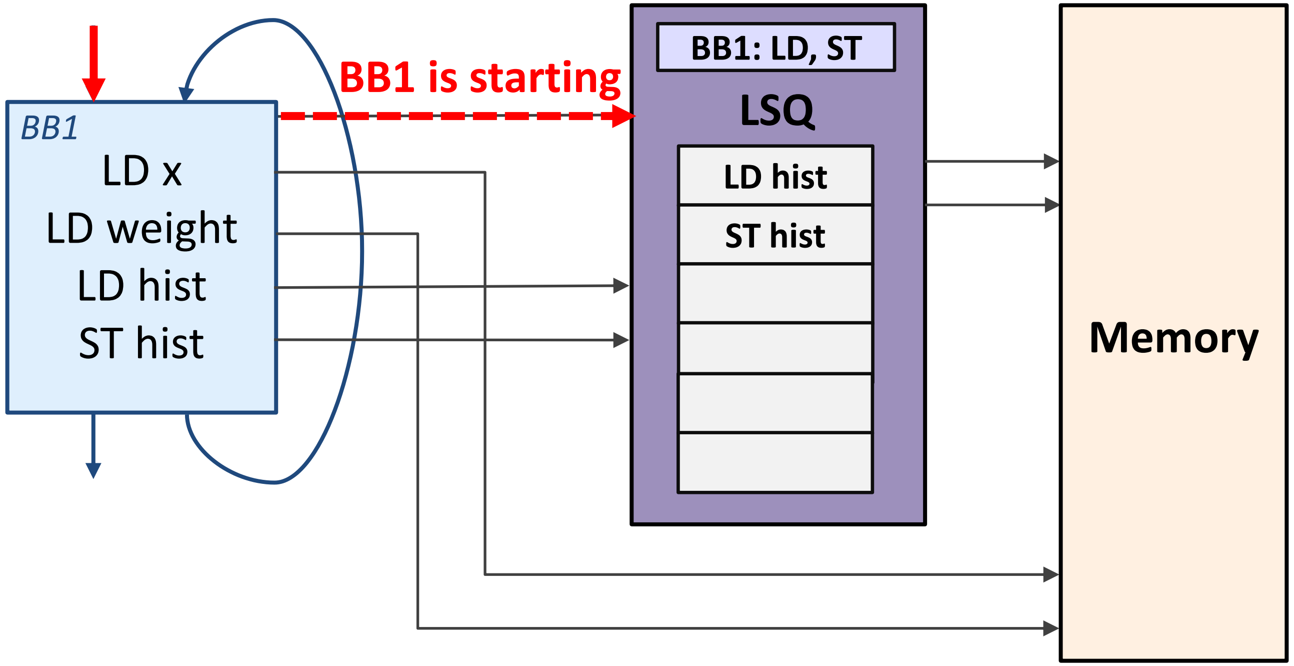


```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Brisk, and Ienne. An Out-of-Order Load-Store Queue for Spatial Computing. CASES 2017 **Best Paper Award Nominee**
Josipović, Bhattacharrya, Guerrieri, and Ienne. Shrink It or Shed It! Minimize the Use of LSQs in Dataflow Designs. FPT 2019

LSQ Allocation

- An LSQ for dataflow circuits whose only difference is in the **allocation policy**:
  - **Static knowledge** of memory access program order inside each basic block
  - **Dynamic knowledge** of the sequence of basic blocks **from the dataflow circuit**

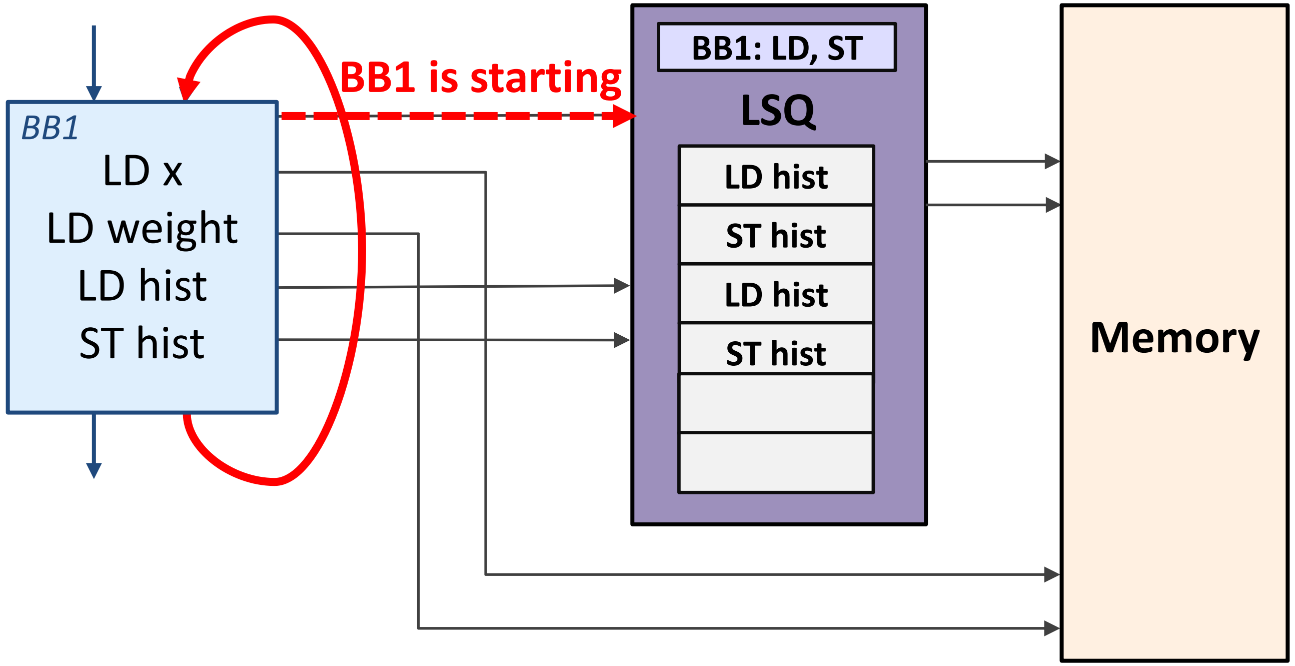


```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

Josipović, Brisk, and Ienne. An Out-of-Order Load-Store Queue for Spatial Computing. CASES 2017 **Best Paper Award Nominee**
Josipović, Bhattacharyya, Guerrieri, and Ienne. Shrink It or Shed It! Minimize the Use of LSQs in Dataflow Designs. FPT 2019

LSQ Allocation

- An LSQ for dataflow circuits whose only difference is in the **allocation policy**:
  - **Static knowledge** of memory access program order inside each basic block
  - **Dynamic knowledge** of the sequence of basic blocks **from the dataflow circuit**



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

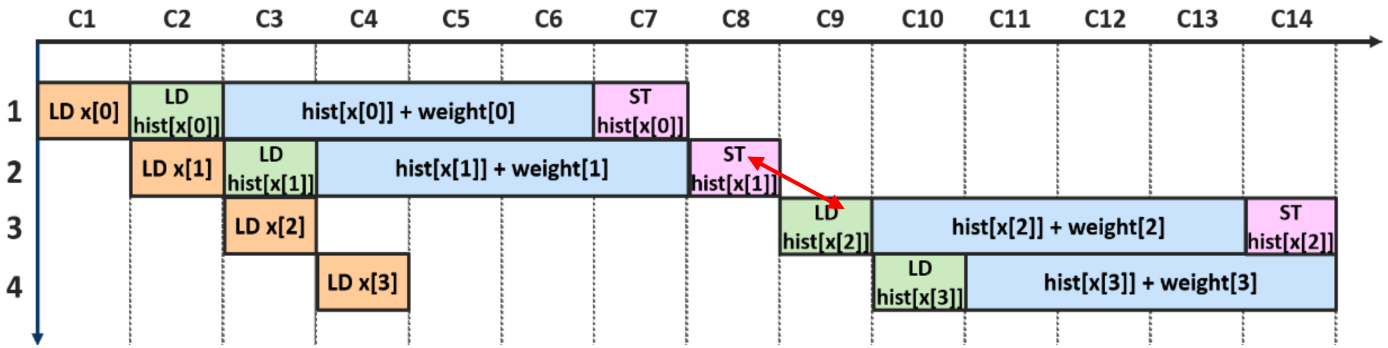
Josipović, Brisk, and Ienne. An Out-of-Order Load-Store Queue for Spatial Computing. CASES 2017 **Best Paper Award Nominee**
Josipović, Bhattacharrya, Guerrieri, and Ienne. Shrink It or Shed It! Minimize the Use of LSQs in Dataflow Designs. FPT 2019

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

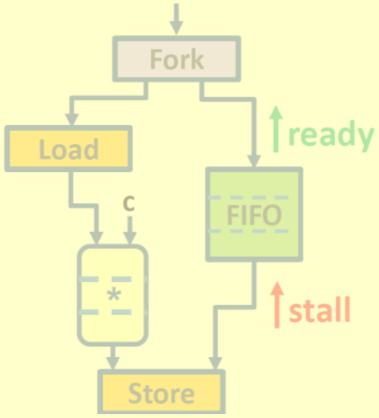


High-throughput pipeline with memory dependencies honored

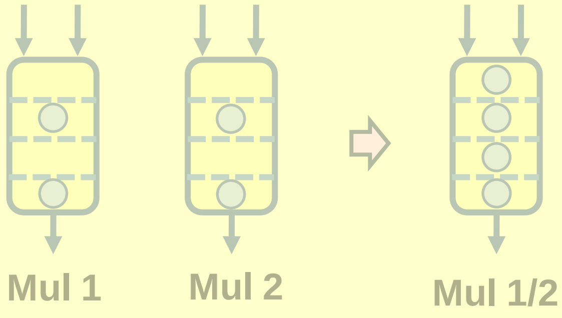
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

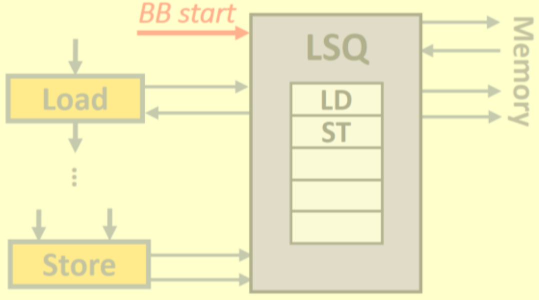


Resource sharing

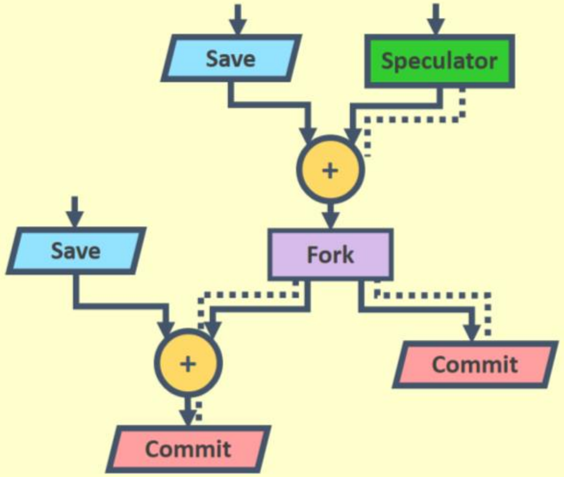


Reaping the benefits of dynamic scheduling

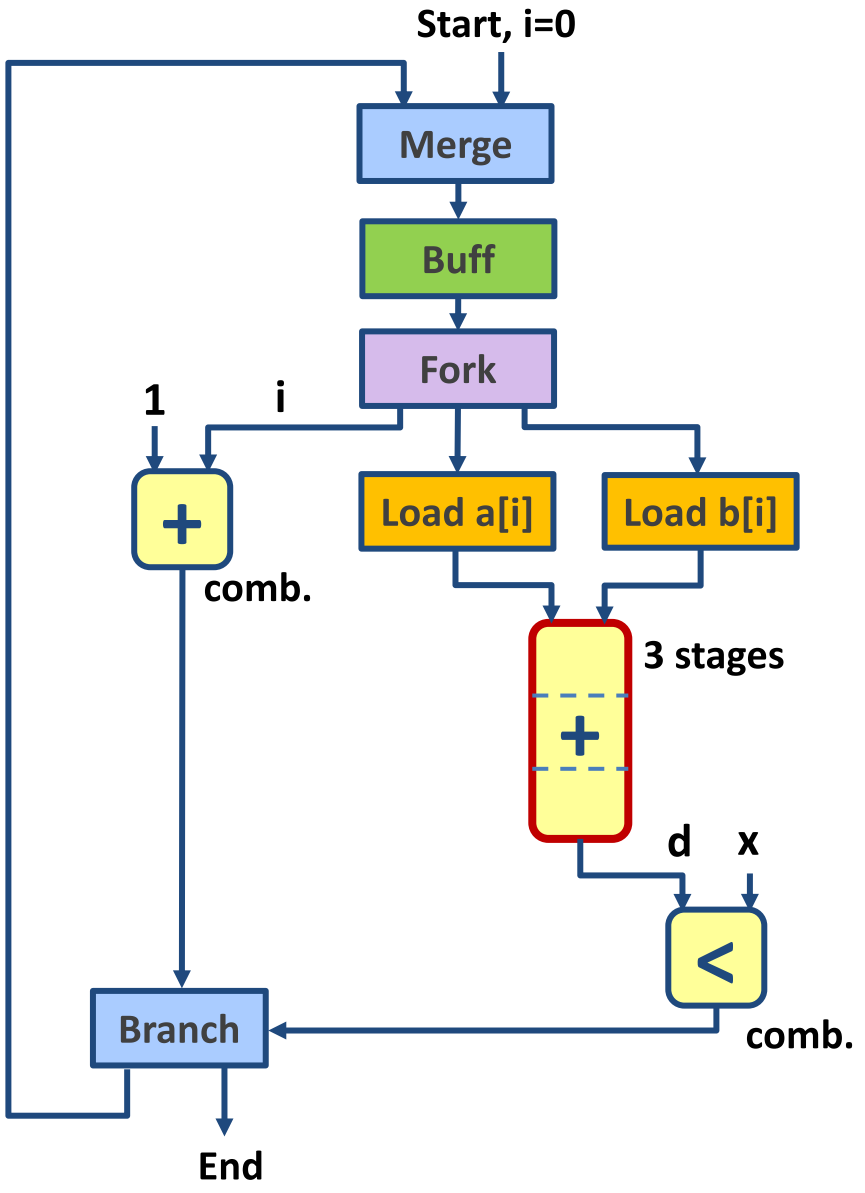
Out-of-order memory



Speculative execution



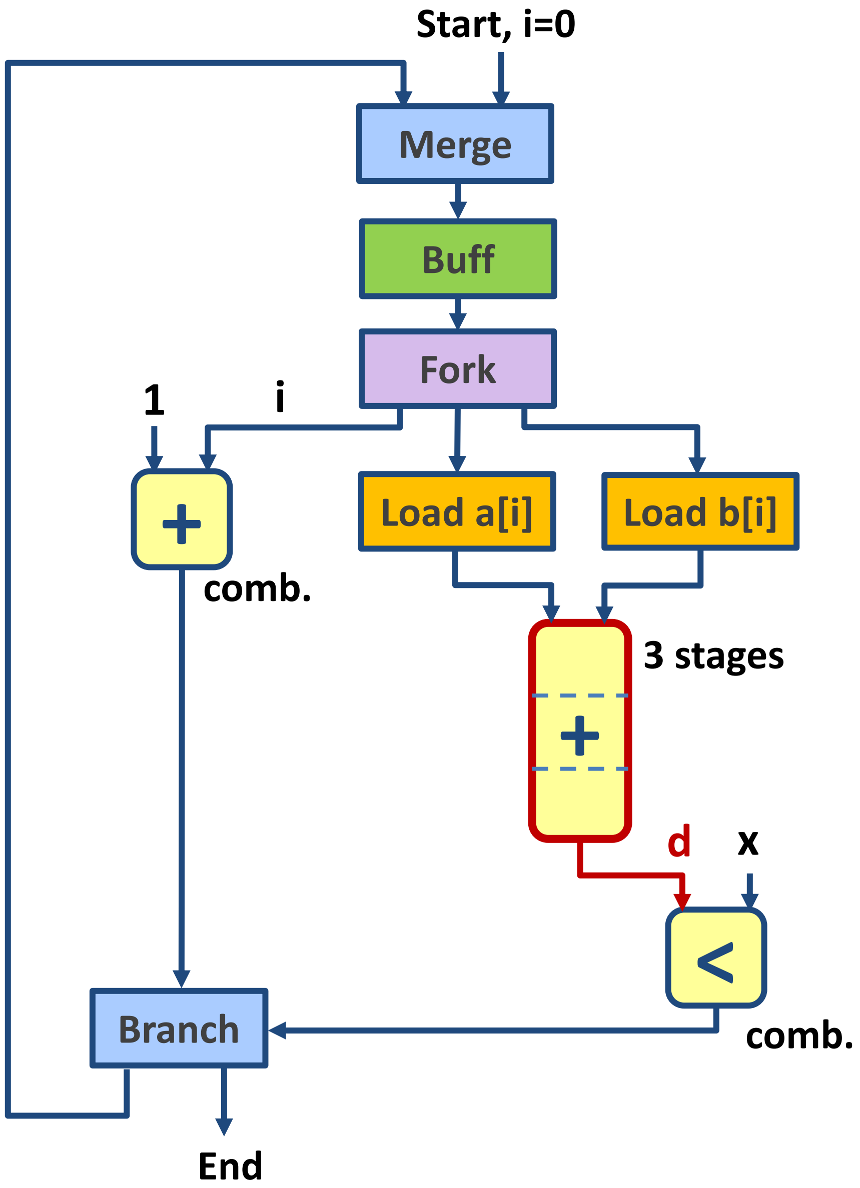
Nonspeculative Dataflow Circuit



```
float d=0.0; x=100.0; int i=0;

do {
   d = a[i] + b[i];
   i++;
}
while (d<x);
```

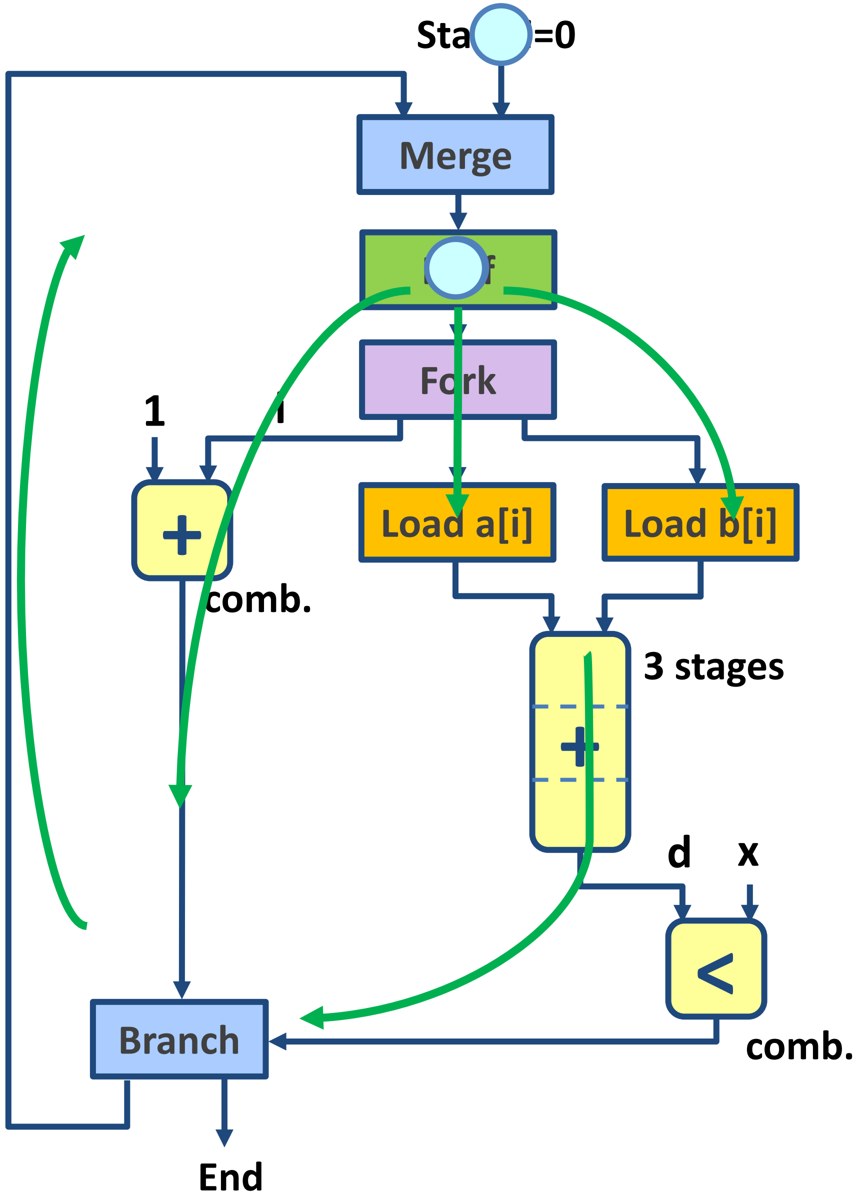
Nonspeculative Dataflow Circuit



```
float d=0.0; x=100.0; int i=0;

do {
   d = a[i] + b[i];
   i++;
}
while (d<x);
```

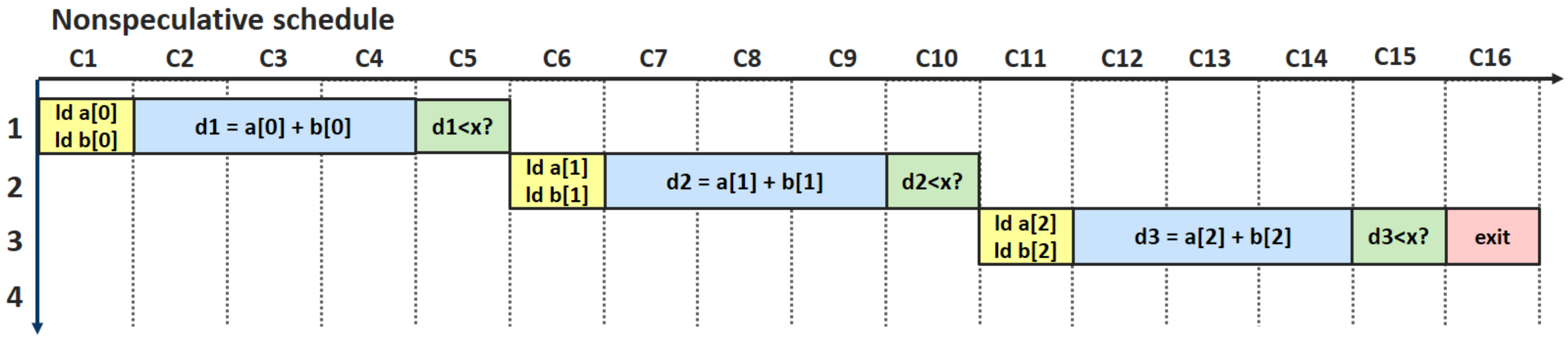
Nonspeculative Dataflow Circuit



```
float d=0.0; x=100.0; int i=0;

do {
   d = a[i] + b[i];
   i++;
}
while (d<x);
```

Nonspeculative vs. Speculative System



Nonspeculative schedule

C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16

1 ld a[0] ld b[0] | d1 = a[0] + b[0] | d1<x?

2 ld a[1] ld b[1] | d2 = a[1] + b[1] | d2<x?

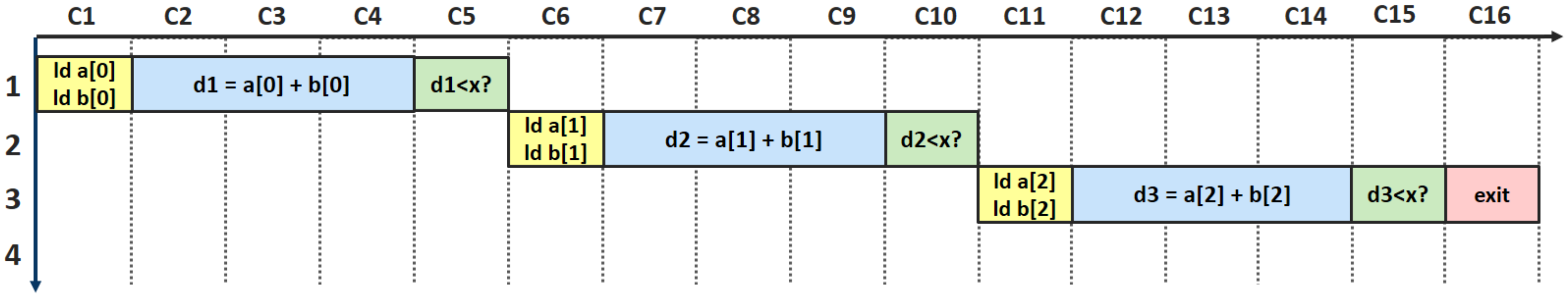
3 ld a[2] ld b[2] | d3 = a[2] + b[2] | d3<x? | exit

4

Long control flow decision prevents pipelining

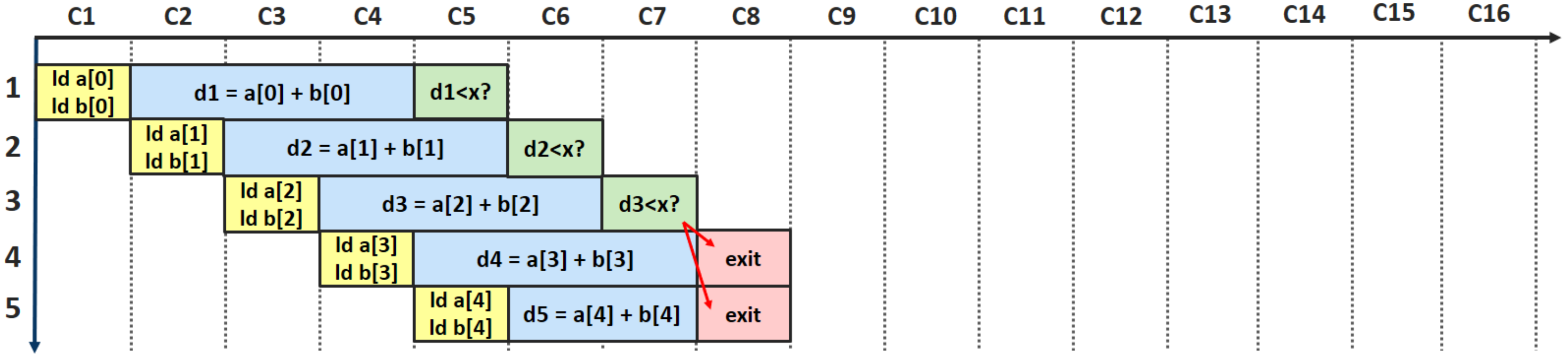
Nonspeculative vs. Speculative System

Nonspeculative schedule



	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	ld a[0] ld b[0]	d1 = a[0] + b[0]			d1<x?											
2						ld a[1] ld b[1]	d2 = a[1] + b[1]			d2<x?						
3											ld a[2] ld b[2]	d3 = a[2] + b[2]			d3<x?	exit
4																

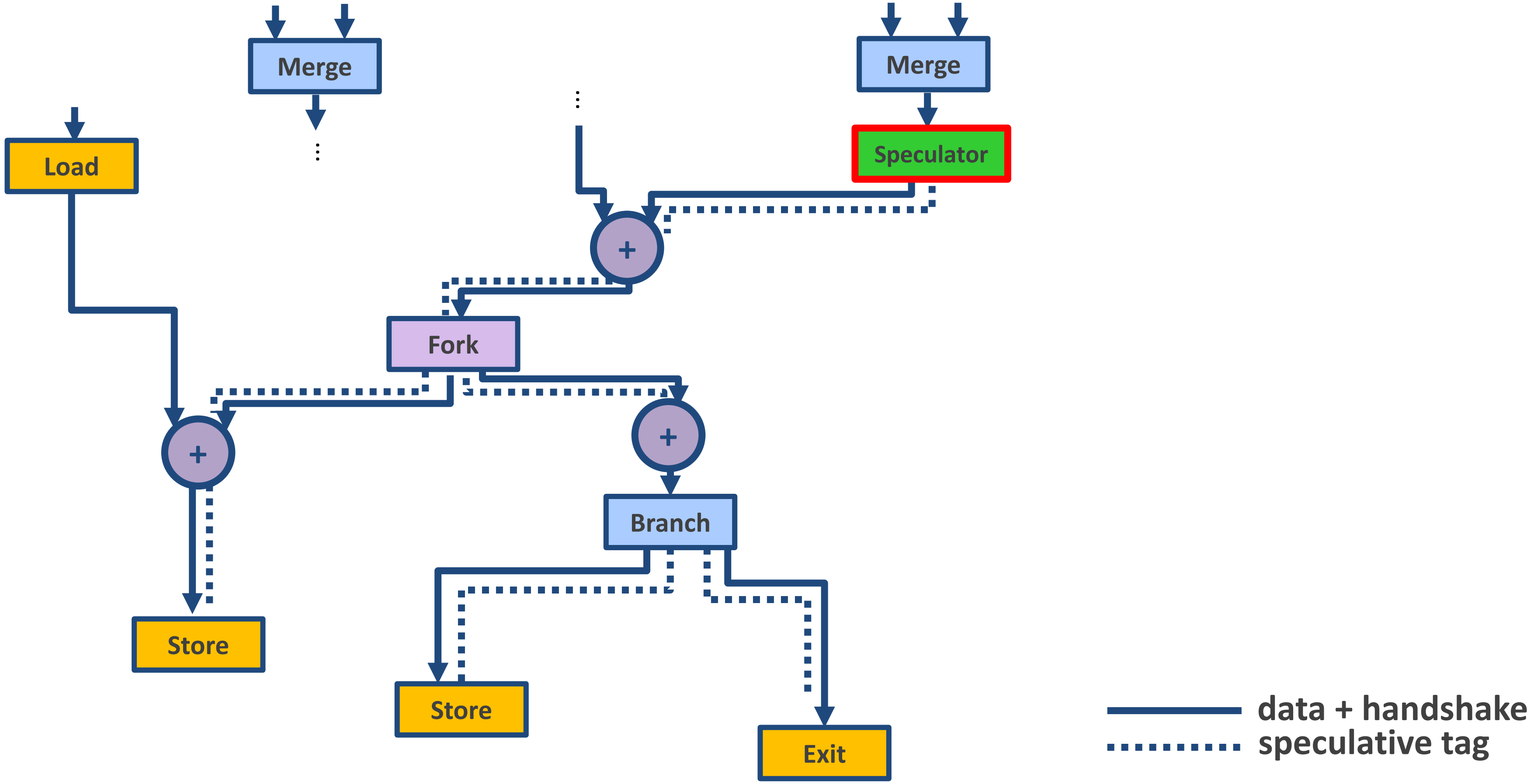
Speculative schedule



	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	ld a[0] ld b[0]	d1 = a[0] + b[0]			d1<x?											
2		ld a[1] ld b[1]	d2 = a[1] + b[1]			d2<x?										
3			ld a[2] ld b[2]	d3 = a[2] + b[2]			d3<x?									
4				ld a[3] ld b[3]	d4 = a[3] + b[3]			exit								
5					ld a[4] ld b[4]	d5 = a[4] + b[4]		exit								

Speculation in Dataflow Circuits

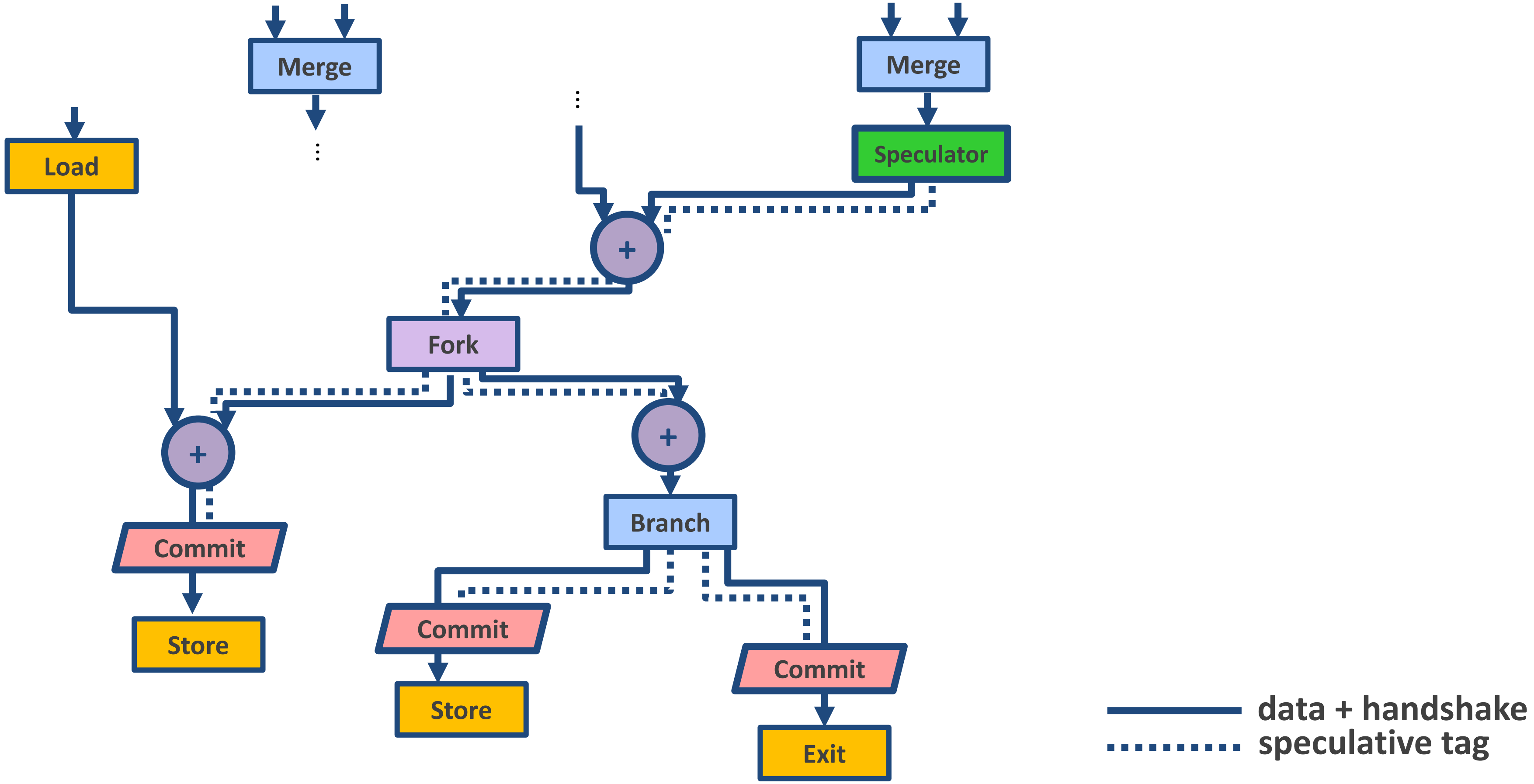
- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation



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

Speculation in Dataflow Circuits

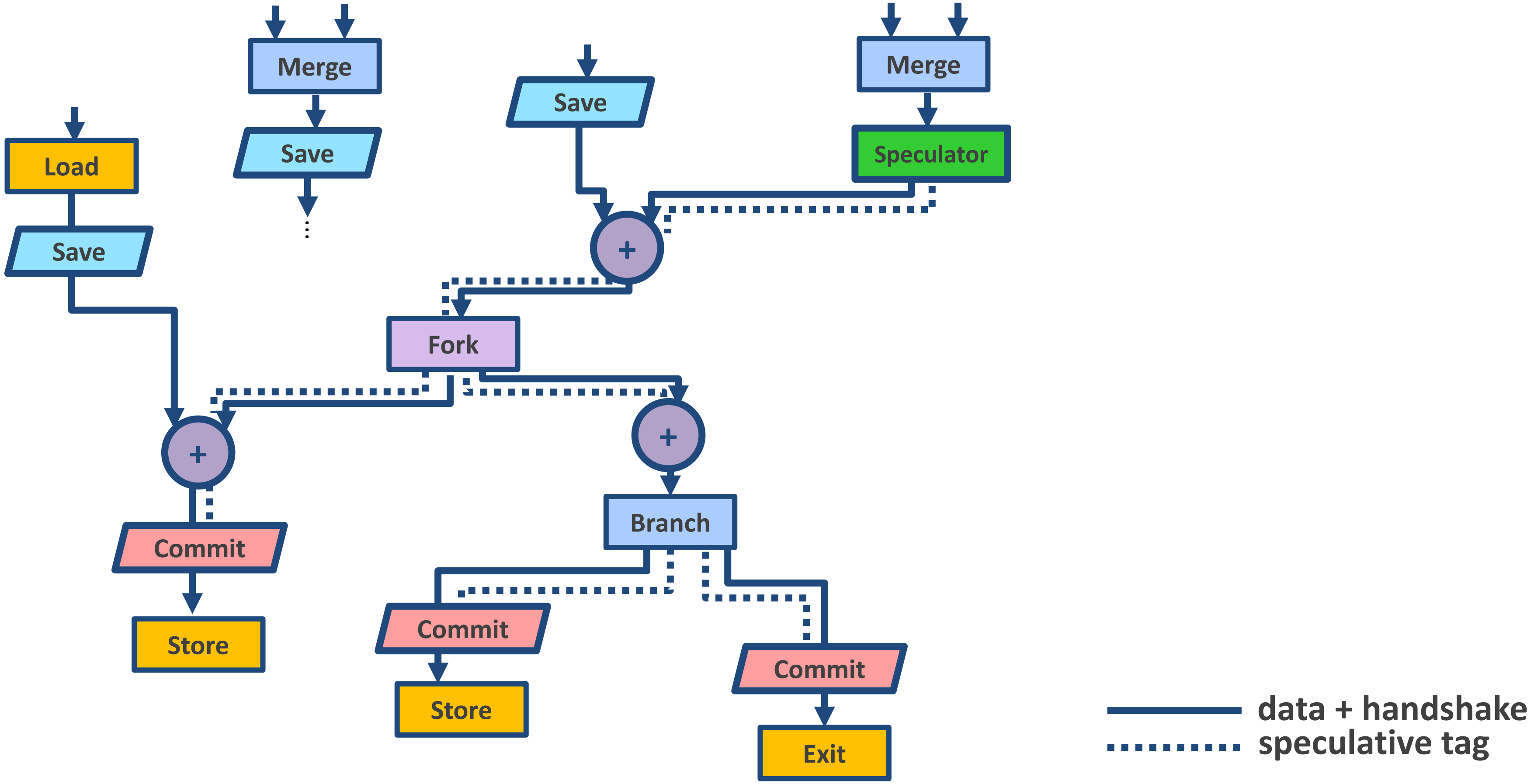
- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation



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

Speculation in Dataflow Circuits

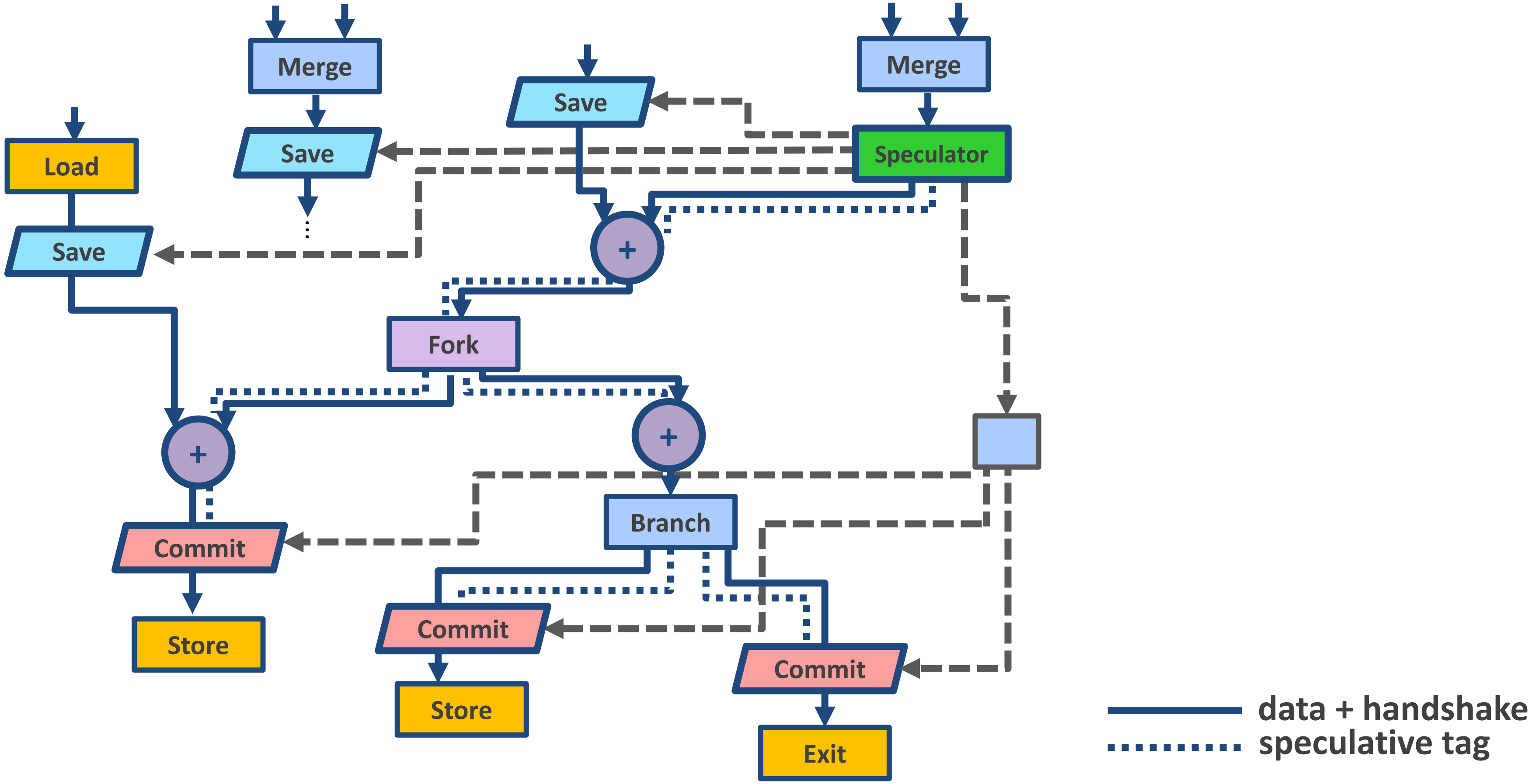
- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation



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

Speculation in Dataflow Circuits

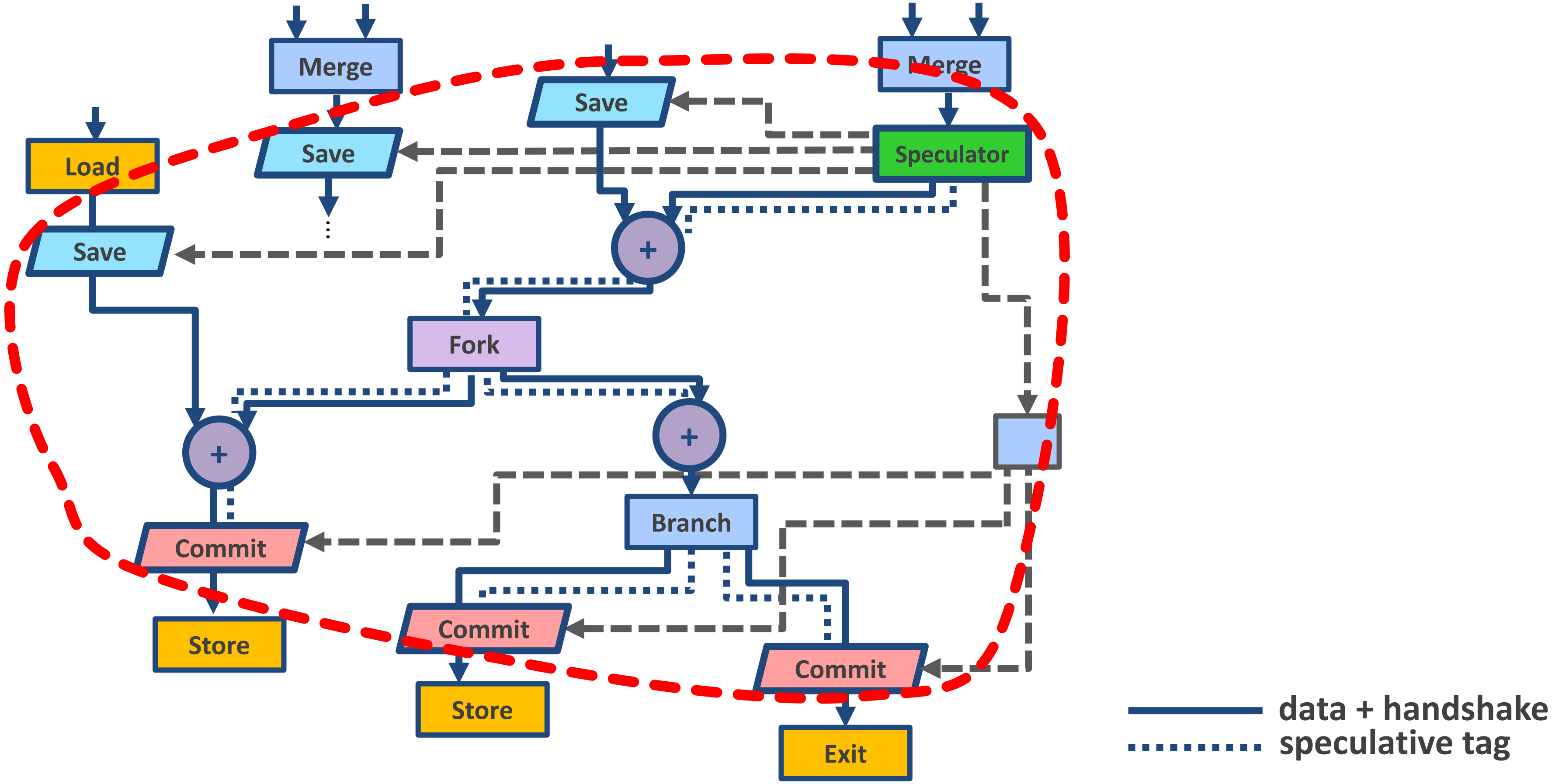
- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation



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

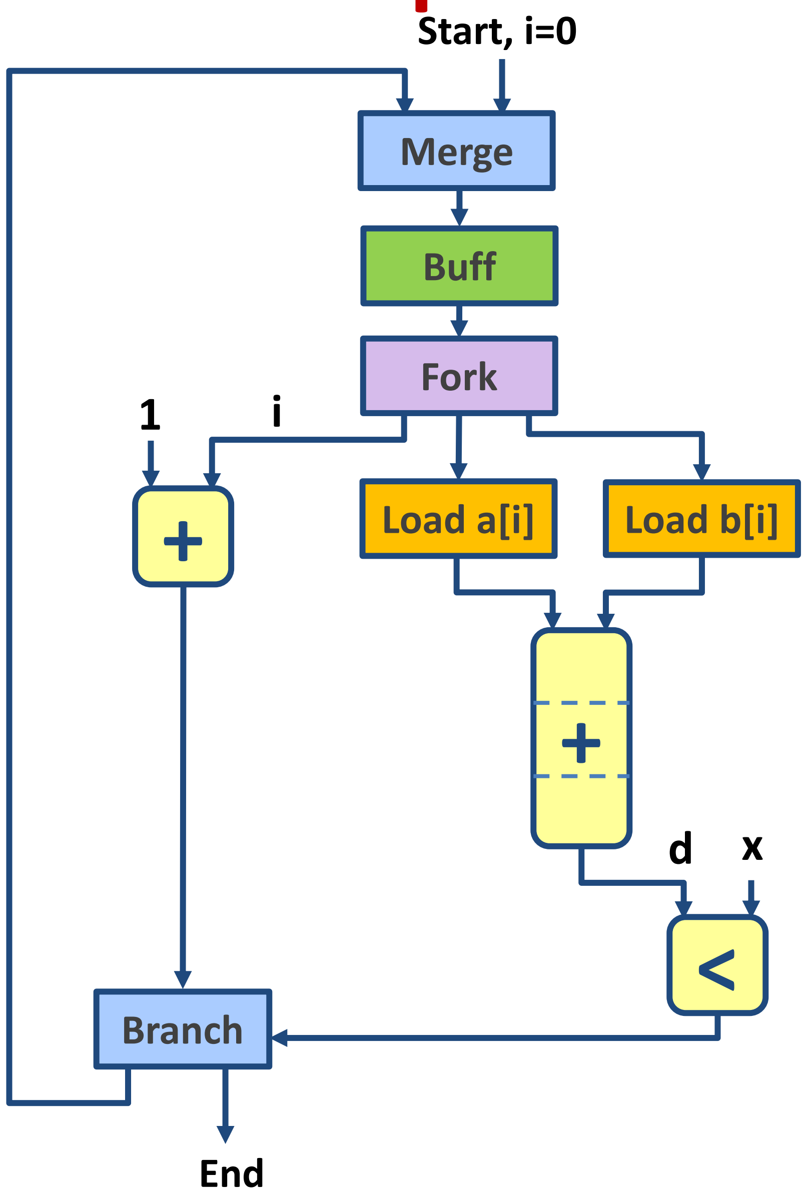
Speculation in Dataflow Circuits

- 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



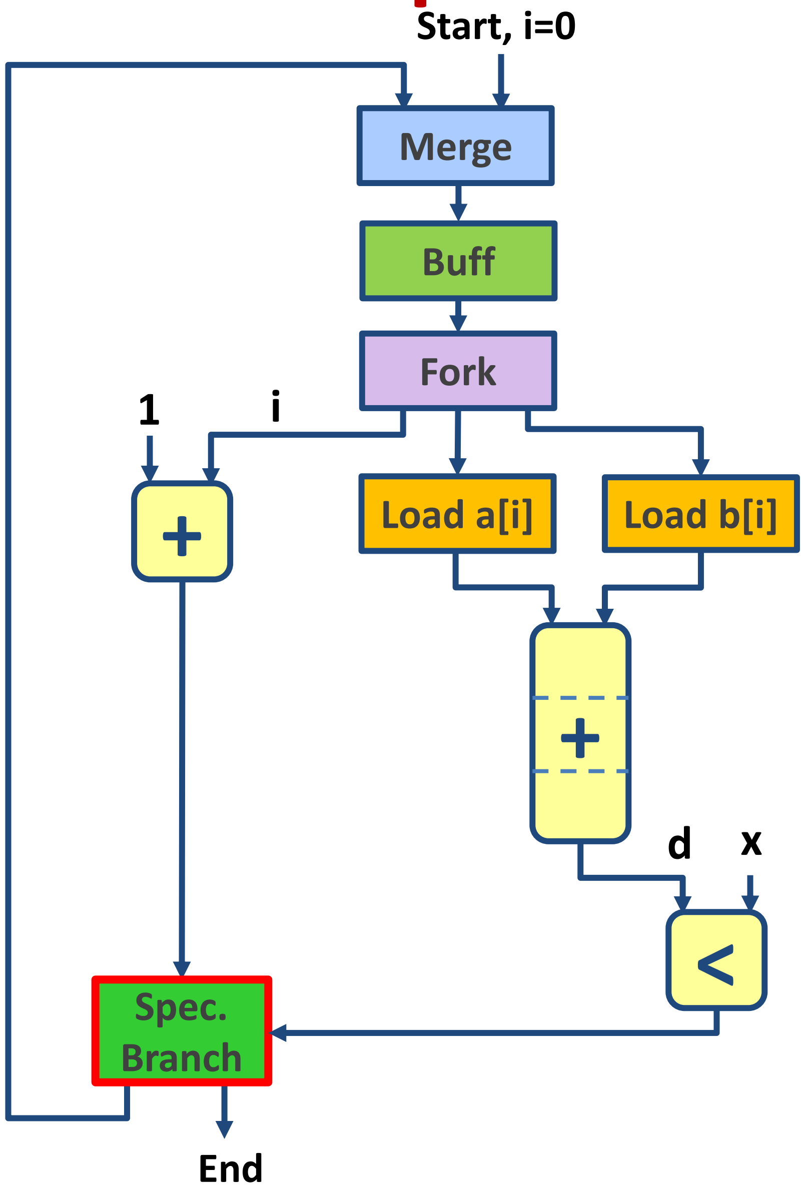
Josipović, Guerrieri, and Ienne. Speculative Dataflow Circuits. FPGA 2019

Speculative Dataflow Circuit



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

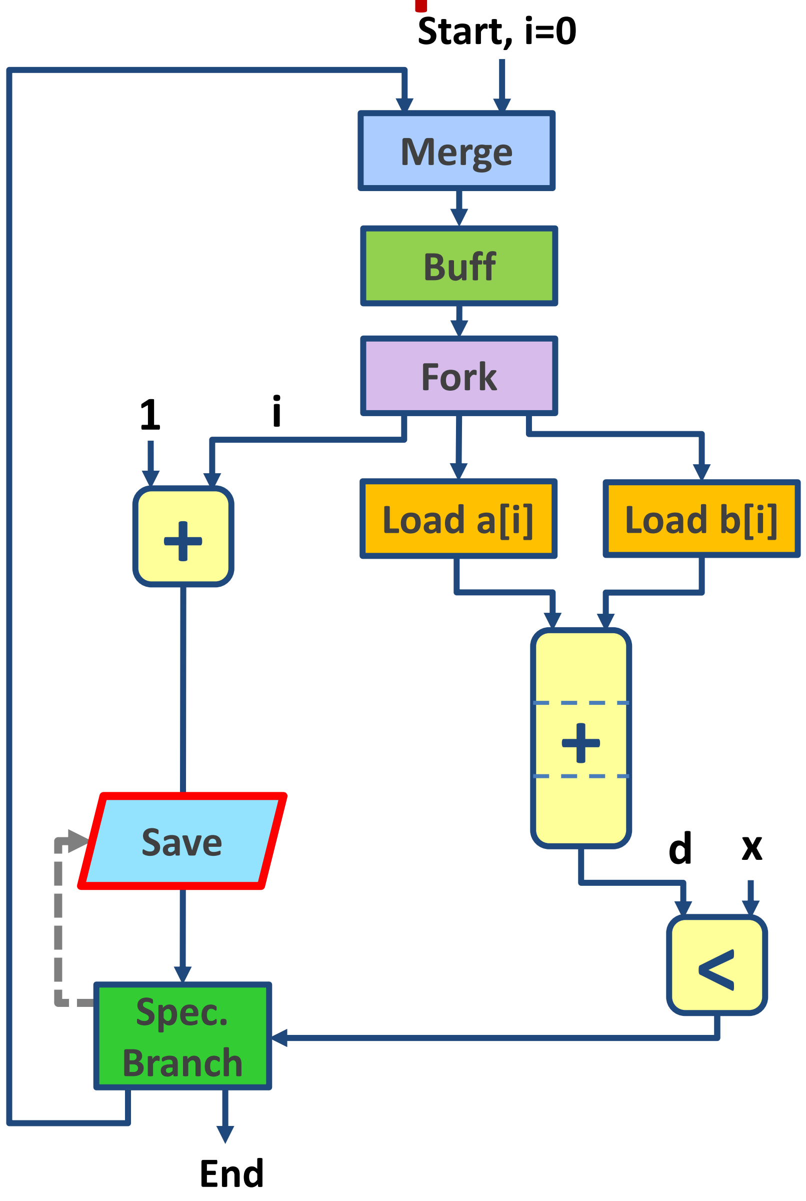
Speculative Dataflow Circuit



Speculator instead of regular Branch

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

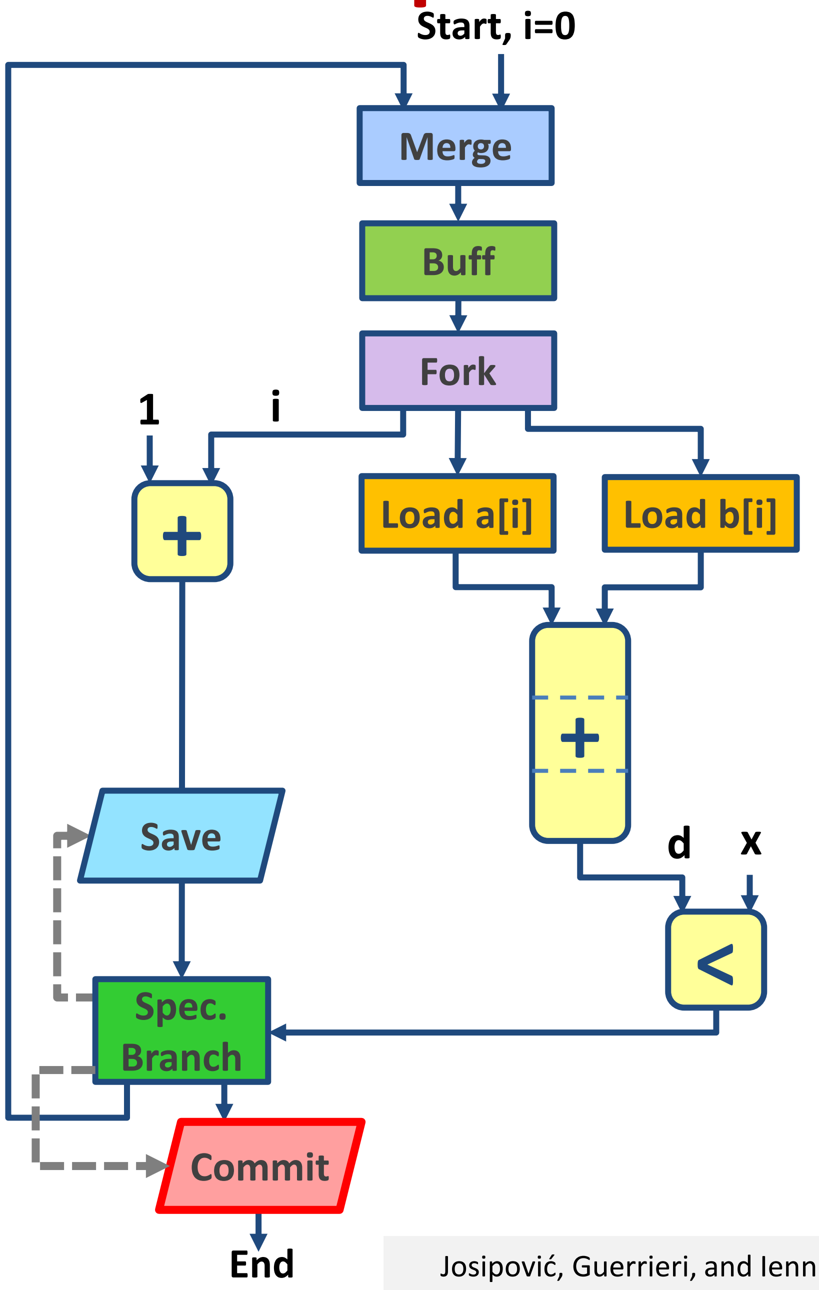
Speculative Dataflow Circuit



Input boundary:
Save units

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

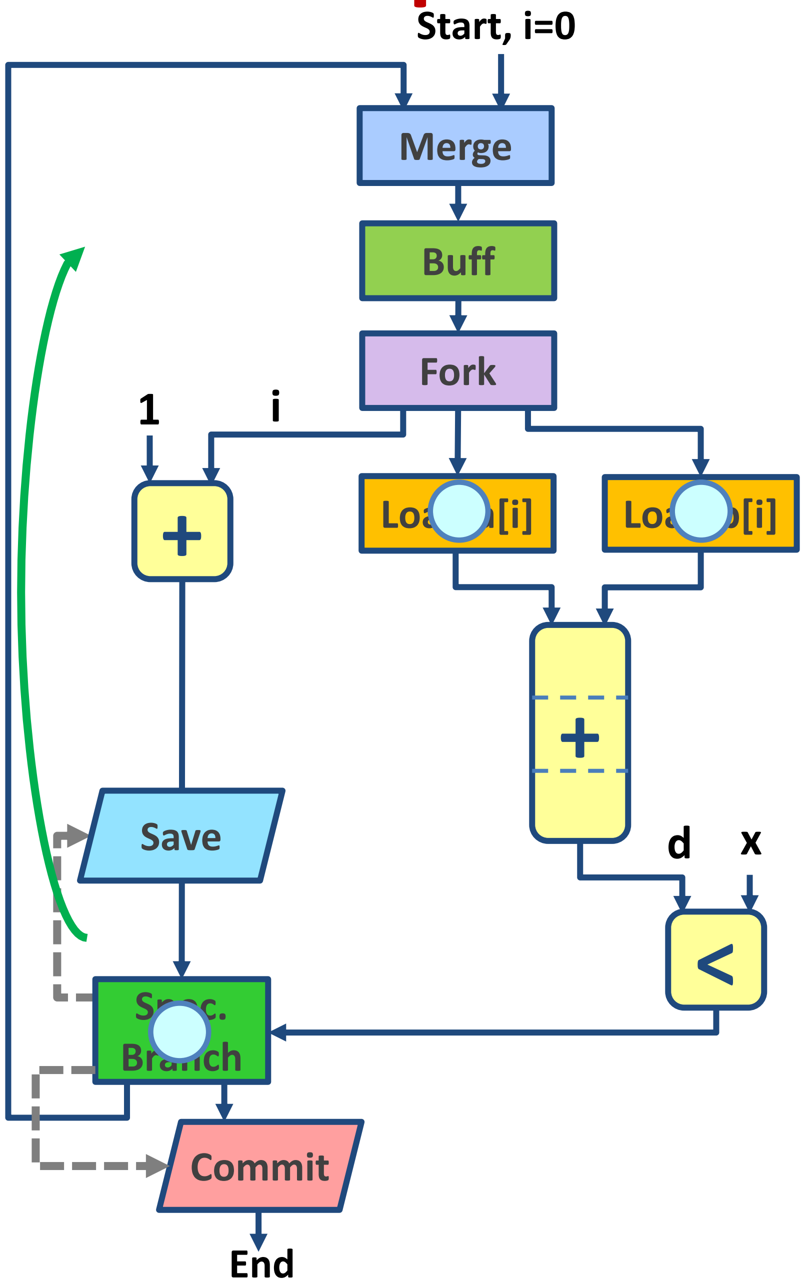
Speculative Dataflow Circuit



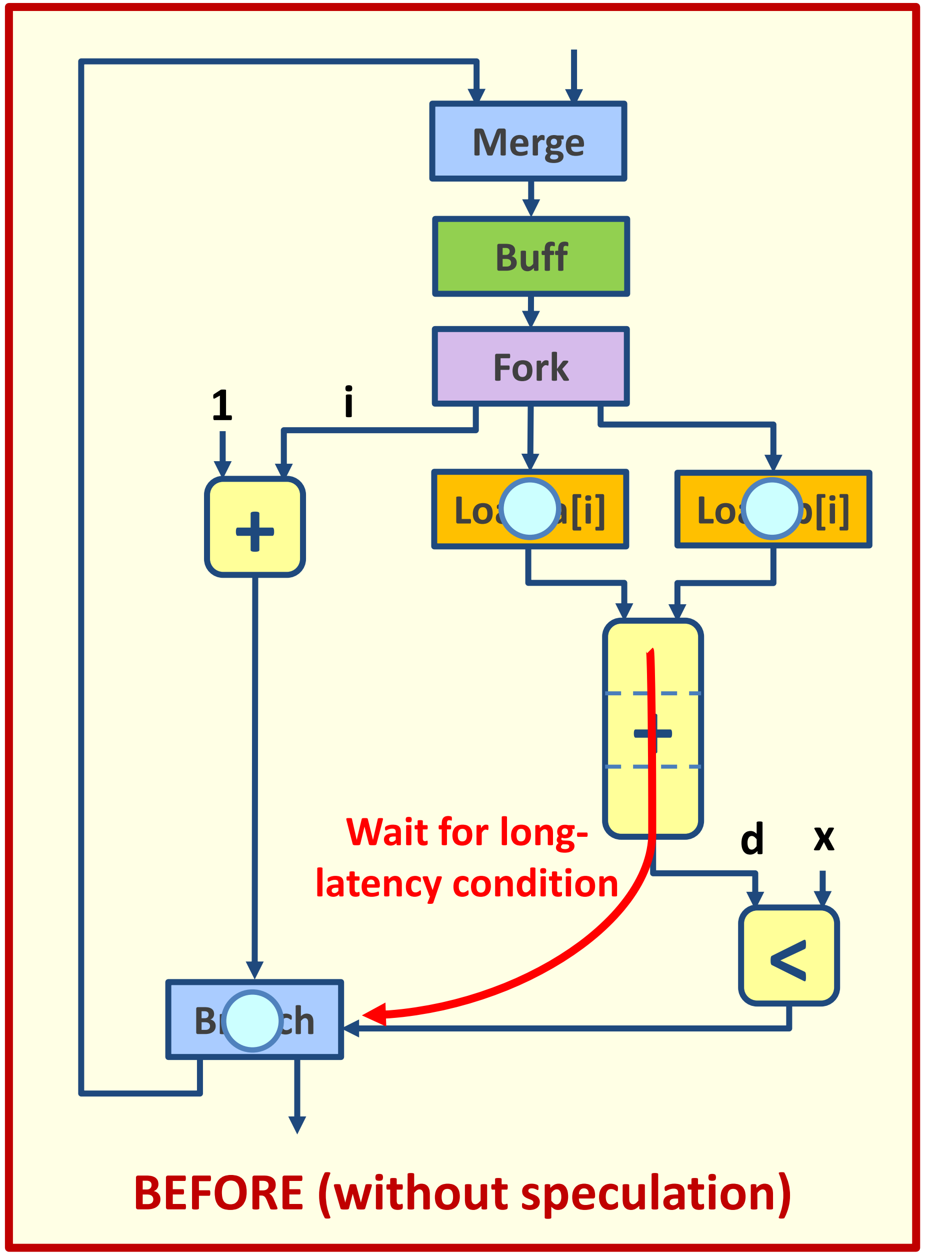
Output boundary:
Commit units

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

Speculative Dataflow Circuit



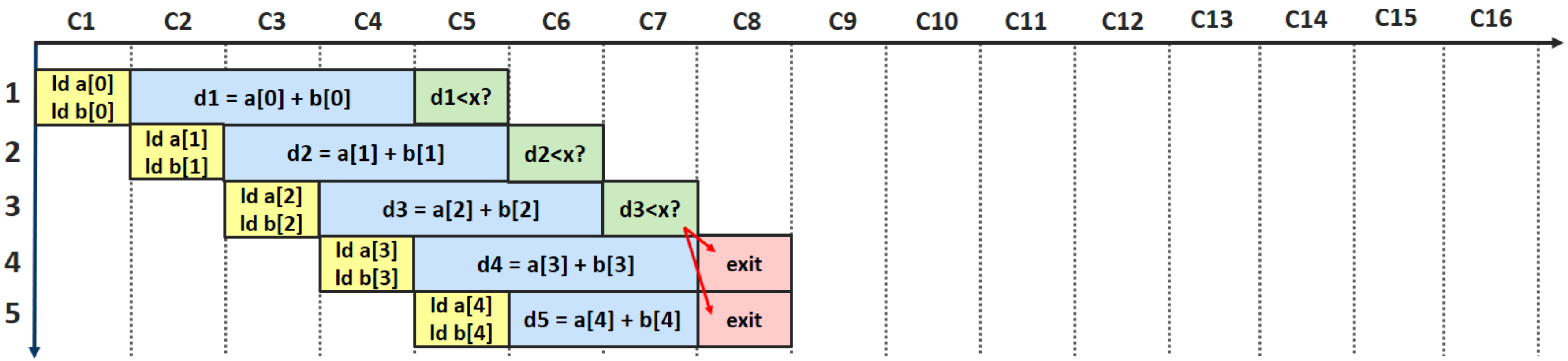
Continue computing before condition known



Wait for long-latency condition

BEFORE (without speculation)

Speculative Dataflow Circuit



	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	ld a[0] ld b[0]	d1 = a[0] + b[0]			d1<x?											
2		ld a[1] ld b[1]	d2 = a[1] + b[1]			d2<x?										
3			ld a[2] ld b[2]	d3 = a[2] + b[2]			d3<x?									
4				ld a[3] ld b[3]	d4 = a[3] + b[3]			exit								
5					ld a[4] ld b[4]	d5 = a[4] + b[4]		exit								

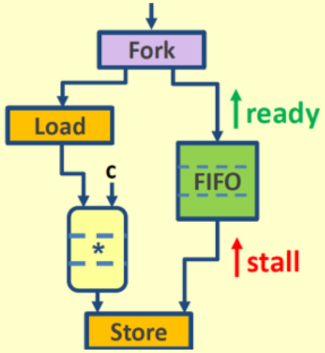
High-throughput speculative pipeline

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

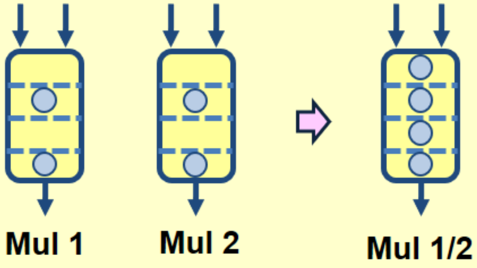
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

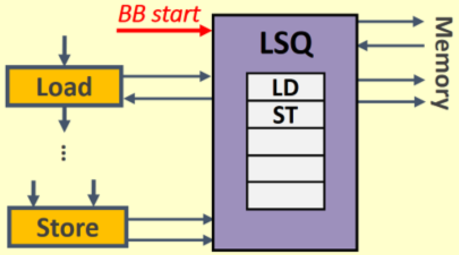


Resource sharing

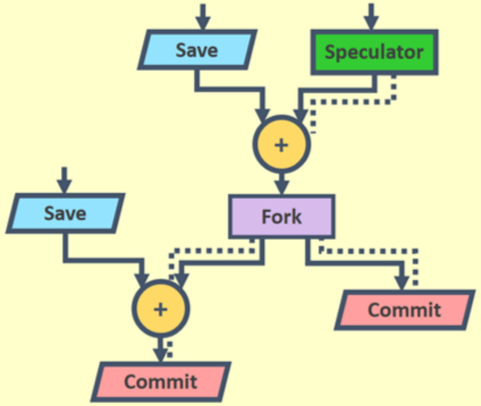


Reaping the benefits of dynamic scheduling

Out-of-order memory



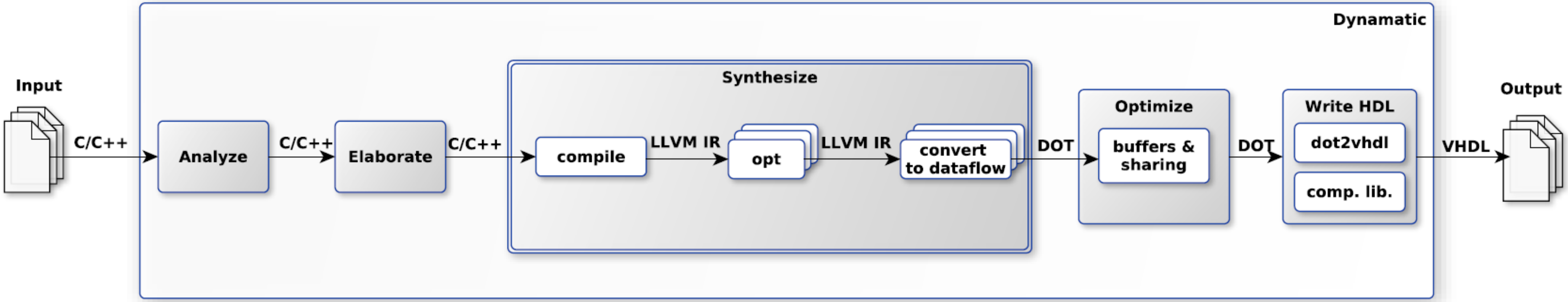
Speculative execution



Static HLS vs. dynamic HLS?

Dynamatic: An Open-Source HLS Compiler

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



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

Experimental Results

- Resource utilization and execution time of the dataflow designs, **normalized to the corresponding static designs** produced by Vivado HLS

▲ Dynamic, control dependences

● Dynamic, memory dependences

◆ Dynamic, speculative

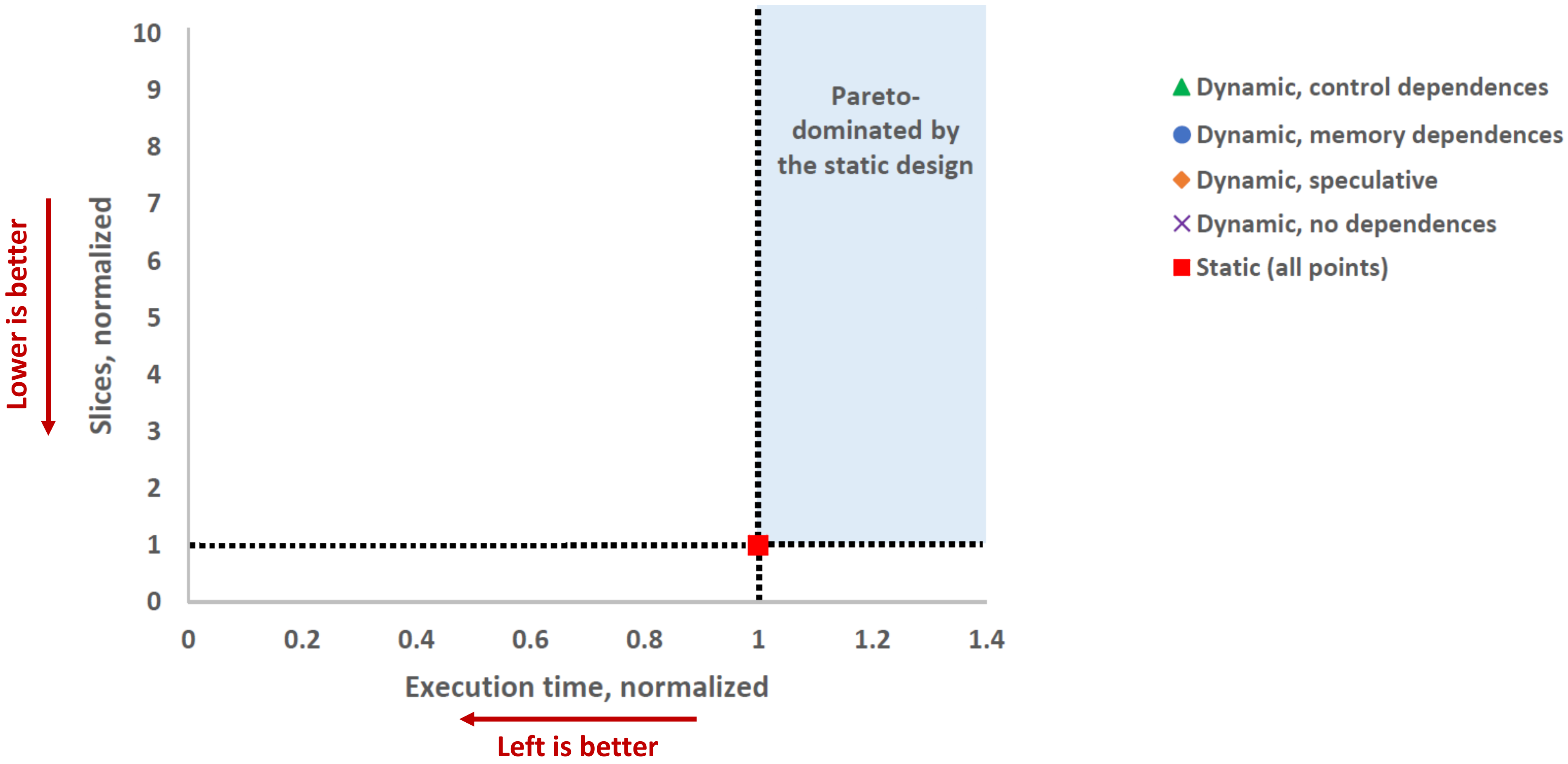
× Dynamic, no dependences

■ Static (all points)

Josipović, Guerrieri, and Ienne. Synthesizing General-Purpose Code into Dynamically Scheduled Circuits. CASM 2021

Experimental Results

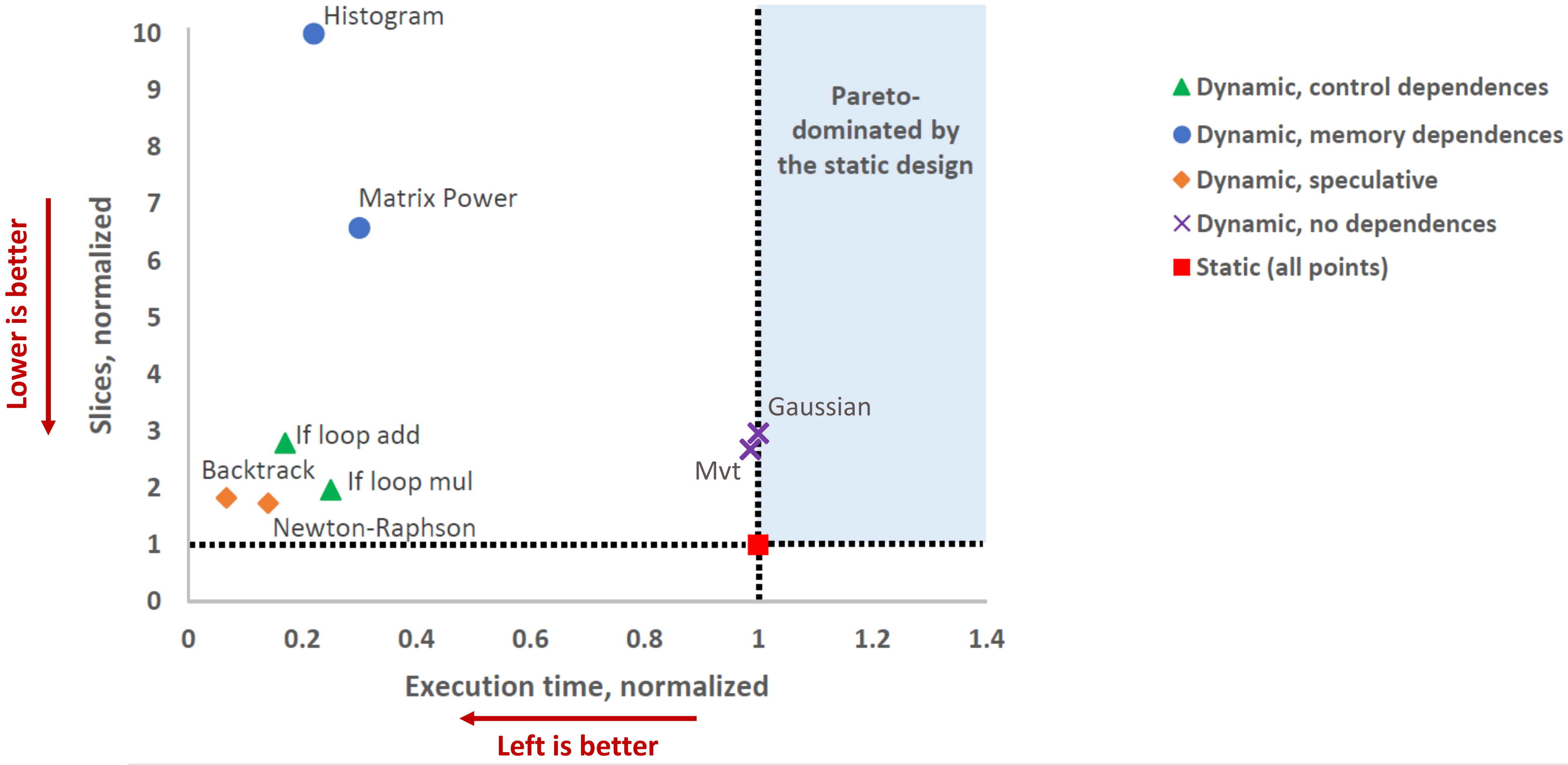
- Resource utilization and execution time of the dataflow designs, **normalized to the corresponding static designs** produced by Vivado HLS



Josipović, Guerrieri, and Ienne. Synthesizing General-Purpose Code into Dynamically Scheduled Circuits. CASM 2021

Experimental Results

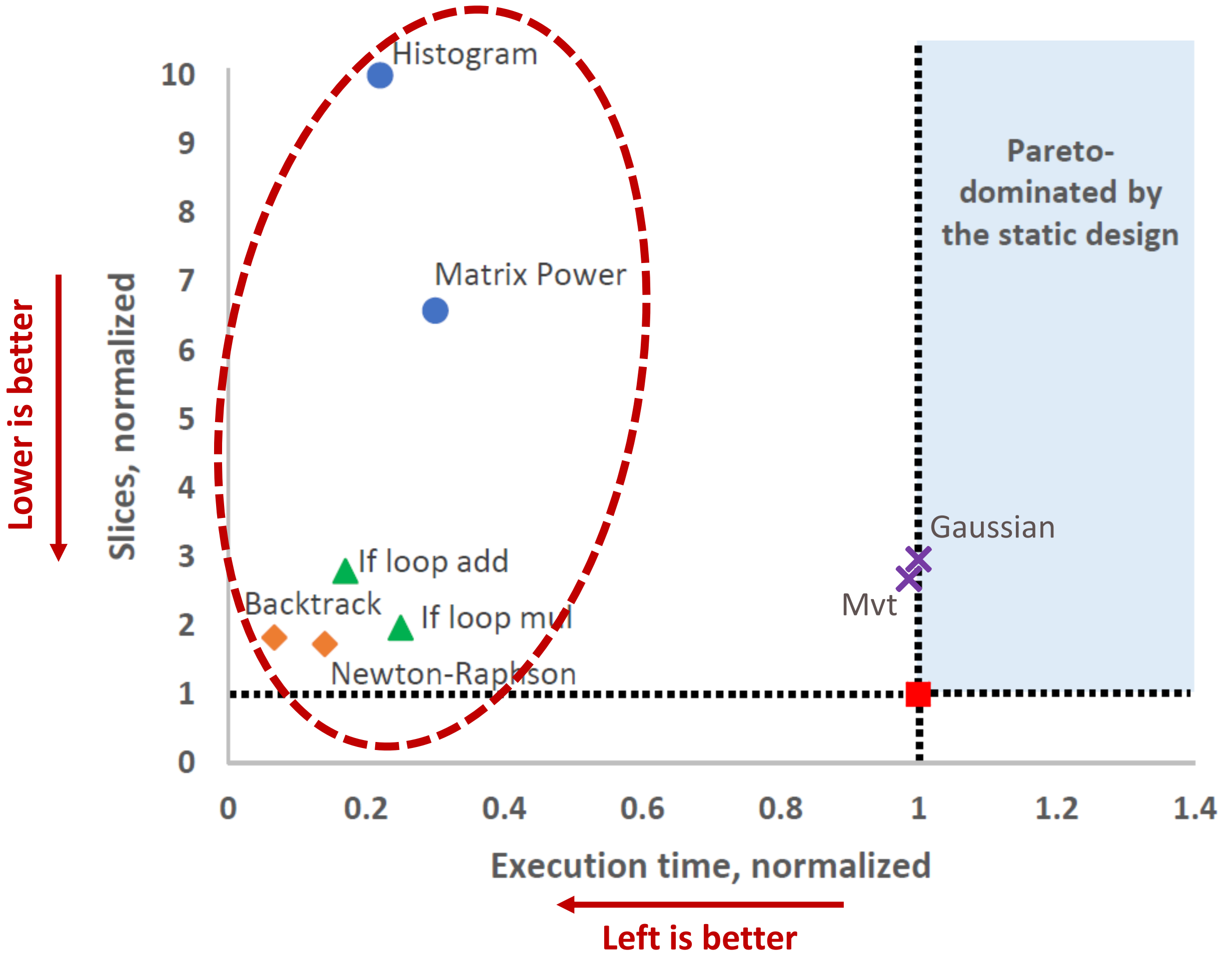
- Resource utilization and execution time of the dataflow designs, **normalized to the corresponding static designs** produced by Vivado HLS



Josipović, Guerrieri, and Ienne. Synthesizing General-Purpose Code into Dynamically Scheduled Circuits. CASM 2021

Experimental Results

- Resource utilization and execution time of the dataflow designs, **normalized to the corresponding static designs** produced by Vivado HLS



▲ Dynamic, control dependences

● Dynamic, memory dependences

◆ Dynamic, speculative

× Dynamic, no dependences

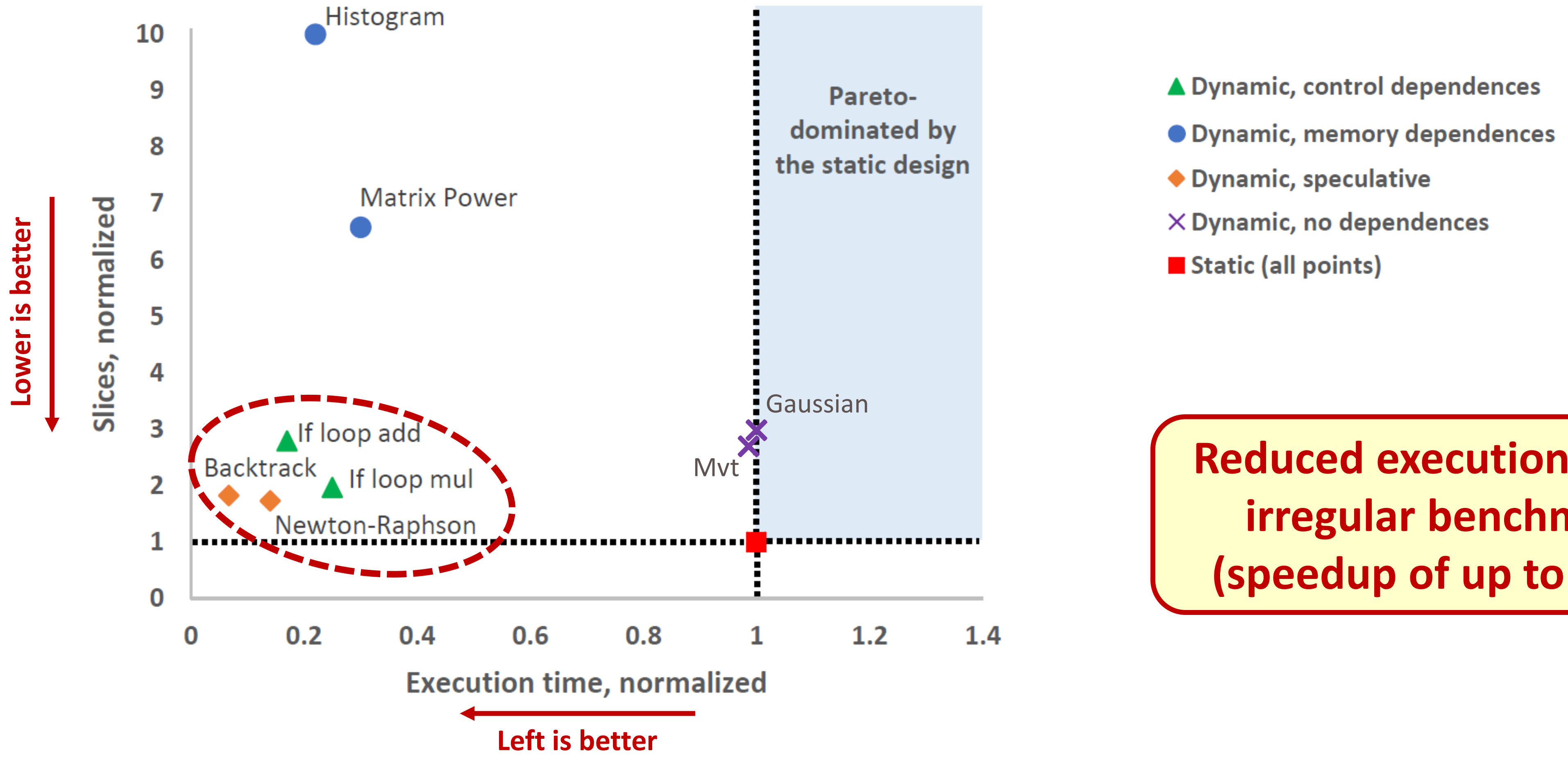
■ Static (all points)

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

Josipović, Guerrieri, and Ienne. Synthesizing General-Purpose Code into Dynamically Scheduled Circuits. CASM 2021

Experimental Results

- Resource utilization and execution time of the dataflow designs, **normalized to the corresponding static designs** produced by Vivado HLS

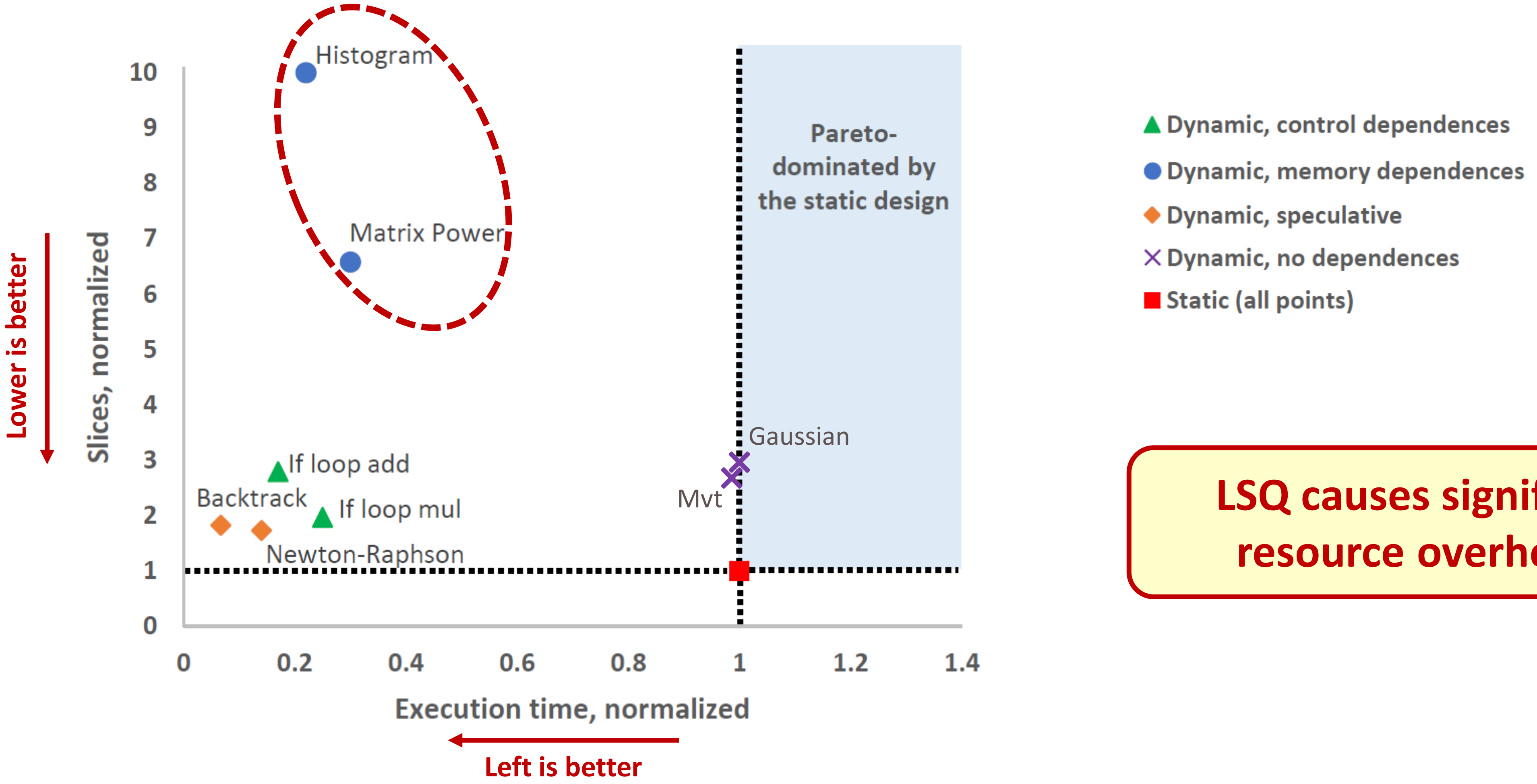


Reduced execution time in irregular benchmarks (speedup of up to 14.9X)

Josipović, Guerrieri, and Ienne. Synthesizing General-Purpose Code into Dynamically Scheduled Circuits. CASM 2021

Experimental Results

- Resource utilization and execution time of the dataflow designs, **normalized to the corresponding static designs** produced by Vivado HLS

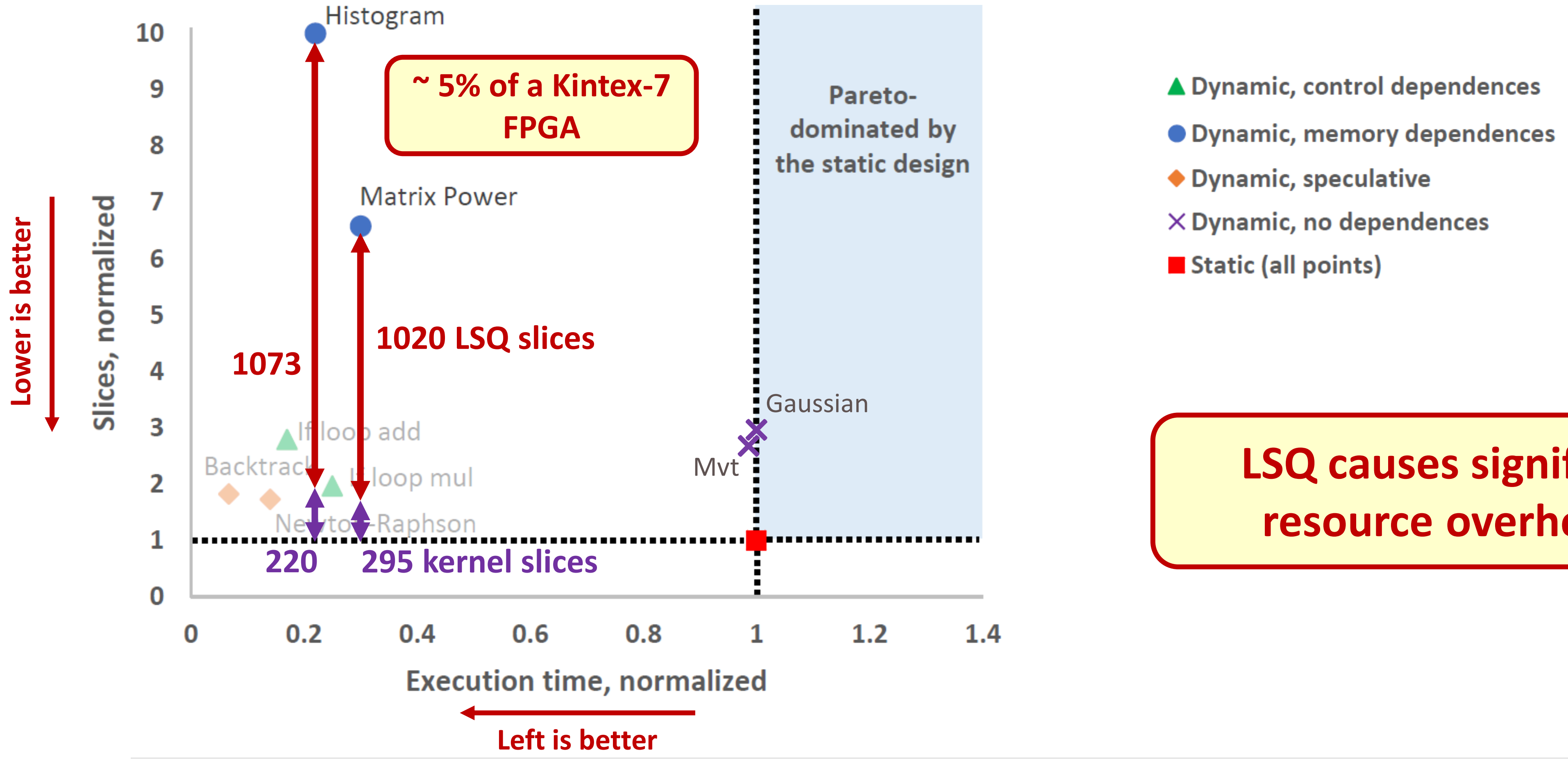


LSQ causes significant resource overheads

Josipović, Guerrieri, and Ienne. Synthesizing General-Purpose Code into Dynamically Scheduled Circuits. CASM 2021

Experimental Results

- Resource utilization and execution time of the dataflow designs, **normalized to the corresponding static designs** produced by Vivado HLS



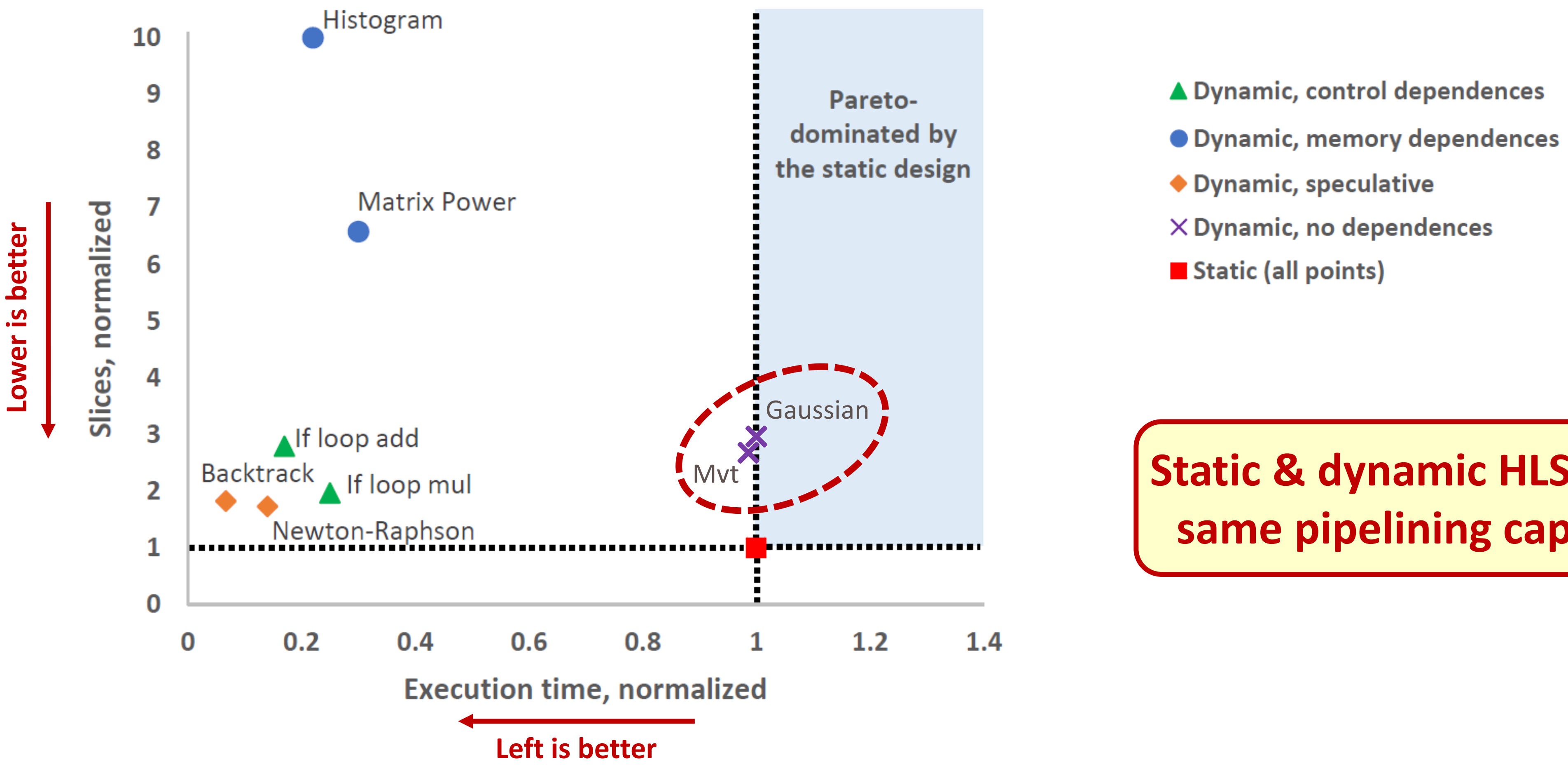
~ 5% of a Kintex-7 FPGA

LSQ causes significant resource overheads

Josipović, Guerrieri, and Ienne. Synthesizing General-Purpose Code into Dynamically Scheduled Circuits. CASM 2021

Experimental Results

- Resource utilization and execution time of the dataflow designs, **normalized to the corresponding static designs** produced by Vivado HLS



Static & dynamic HLS have the same pipelining capabilities

Josipović, Guerrieri, and Ienne. Synthesizing General-Purpose Code into Dynamically Scheduled Circuits. CASM 2021

Static vs. Dynamic Scheduling

	Statically Scheduled → "Compiler does the job"	**Dynamically Scheduled** → "Hardware does the job"
Computer Architecture	**VLIW Processors**	**Out-of-Order Superscalar Processors**
High-Level Synthesis	**Traditional HLS**	**Dataflow circuits**

DSP-oriented applications

General-purpose code

(new applications and users)

Thanks! ☺

Research group:

https://dynamo.ethz.ch/

Dynamatic HLS tool:

https://dynamatic.epfl.ch/