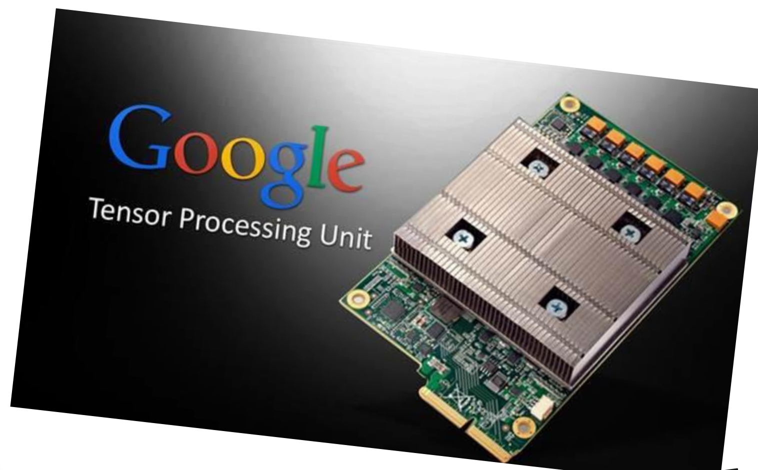
Petri Nets and Model Checking in Circuit Design

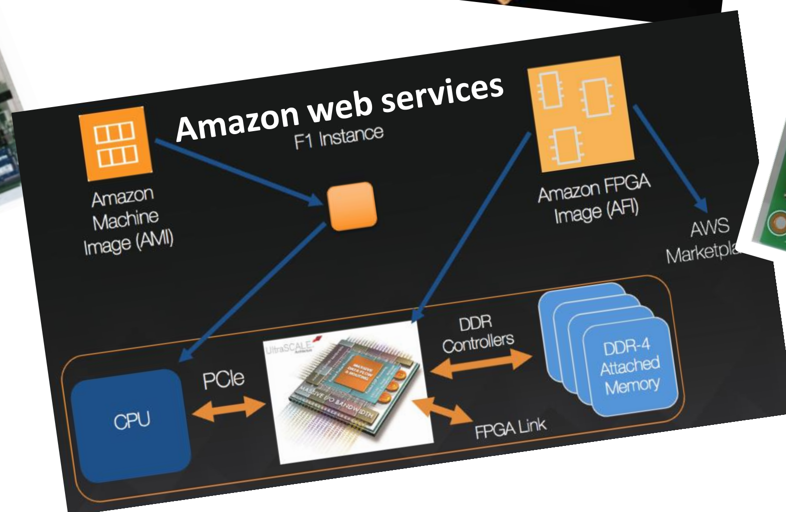
Lana Josipović

December 2024

ETH zürich

Hardware Acceleration for High Parallelism & Energy Efficiency



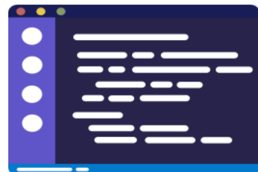


How to perform hardware design?

… circuit design is often considered a ***"black art",*** *restricted to only those with years of training in electrical engineering…*
[cacm.acm.org/magazines/2023/1/]

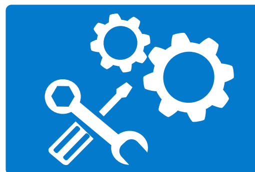
Digital Systems and Design Automation Group (DYNAMO)

High-level abstractions



programming languages,
software applications

Hardware compilers



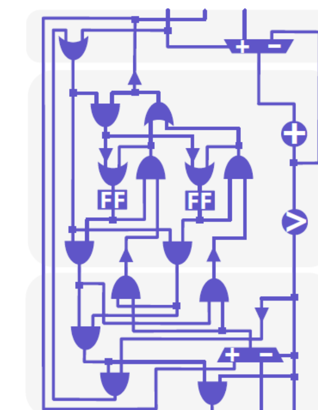
formal methods, machine learning,
electronic design automation

Hardware design

systems, digital design,
computer architecture

```
for (j = 0; j < 10; j++) {
   float x = 0.0;
   for (i = 0; i < 10; i++)
       x += data[i][j];
    mean[j] = x / float_n;
}

for (j = 0; j < 10; j++) {
   float x = 0.0;
   for (i = 0; i < 10; i++)
       x += (data[i][j] - mean[j]) *
(data[i][j] - mean[j]);
   x /= float_n;
   x = x*x;
   stdev[j] = x;
}
```



**Make hardware design
broadly accessible, fast, and reliable**

Digital Systems and Design Automation Group (DYNAMO)

SW

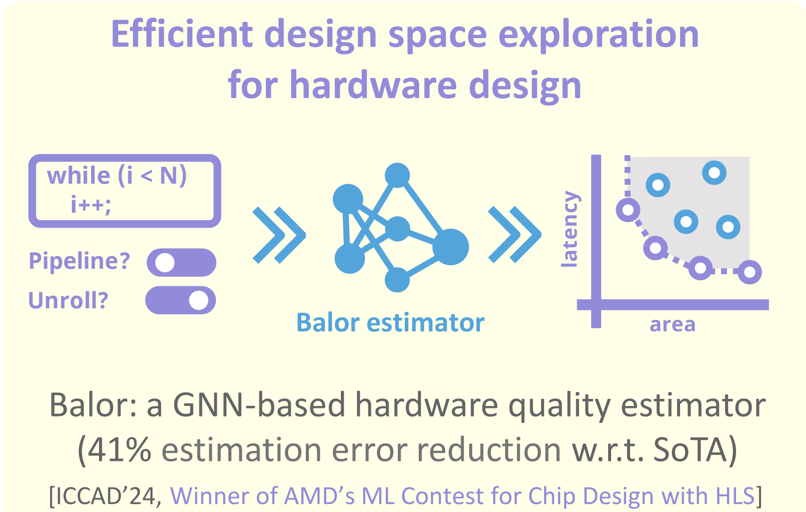
1. How to make hardware design accessible to non-experts?

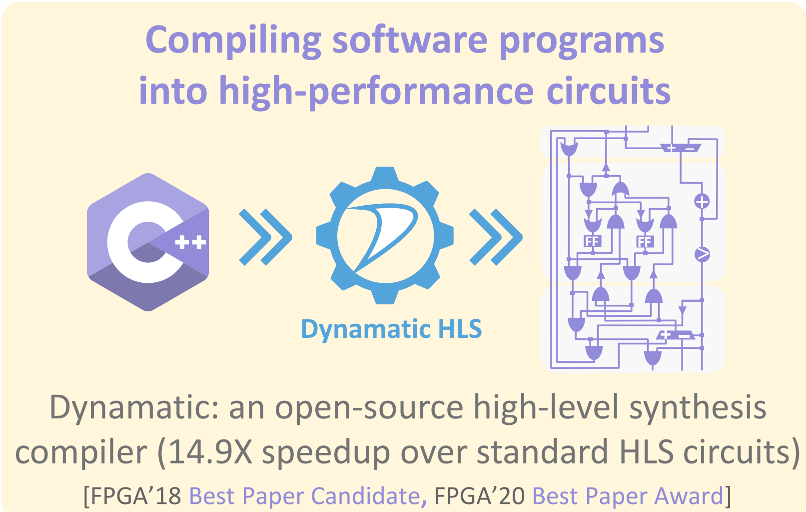
2. How to automatically extract parallelism from software code?

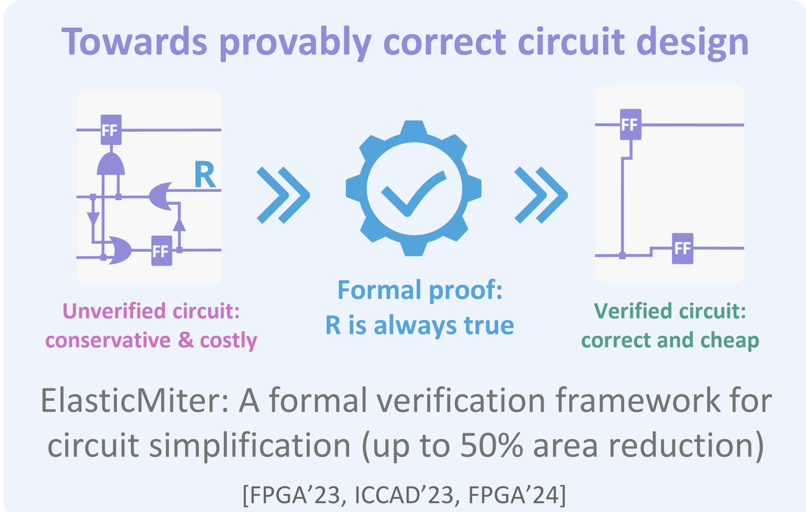
3. How to verify circuits and circuit transformations?

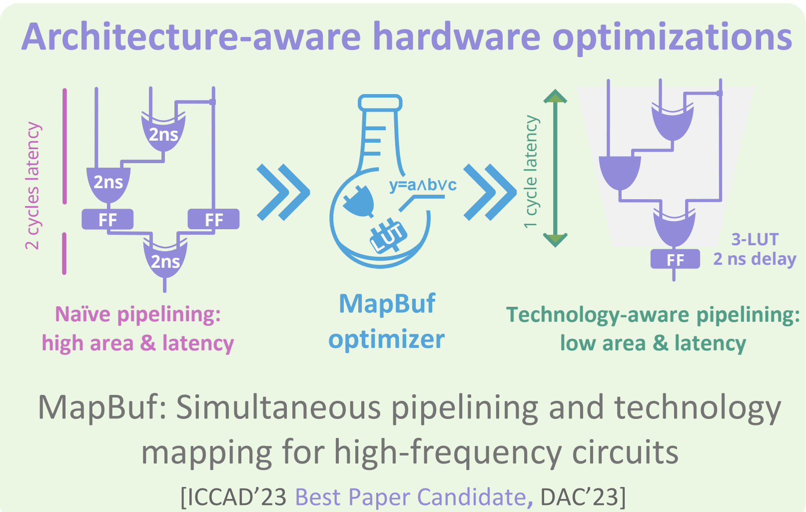
4. How to understand & leverage hardware implementation details?

HW

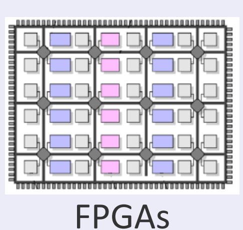


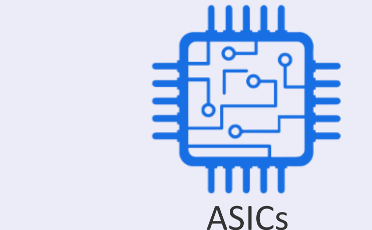


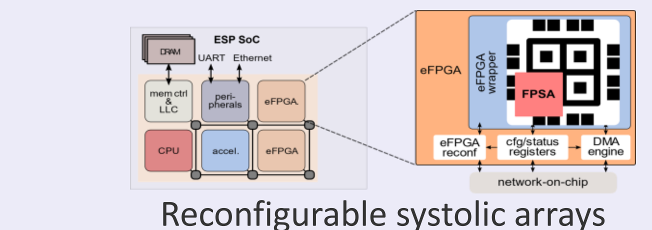




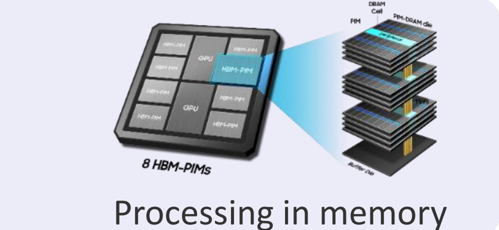
Making diverse hardware accelerators broadly accessible


FPGAs


ASICs



Reconfigurable systolic arrays



Processing in memory

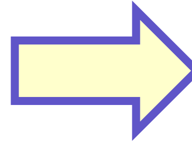
High-Level Synthesis: From Programs to Circuits

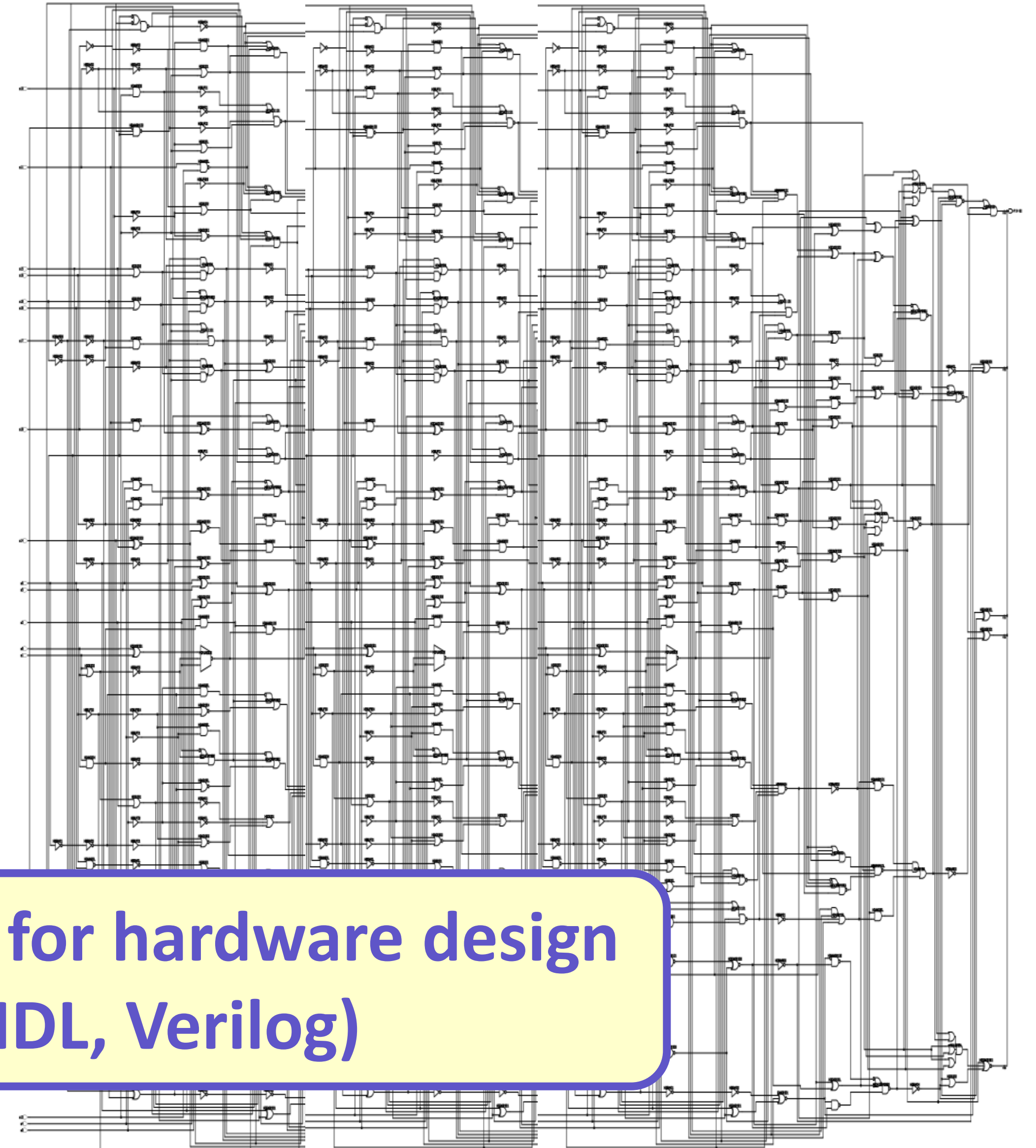
```
#define PI 3.14159265358979323846264334

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;
}
```





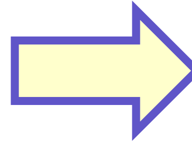
Raise the level of abstraction for hardware design beyond RTL level (VHDL, Verilog)

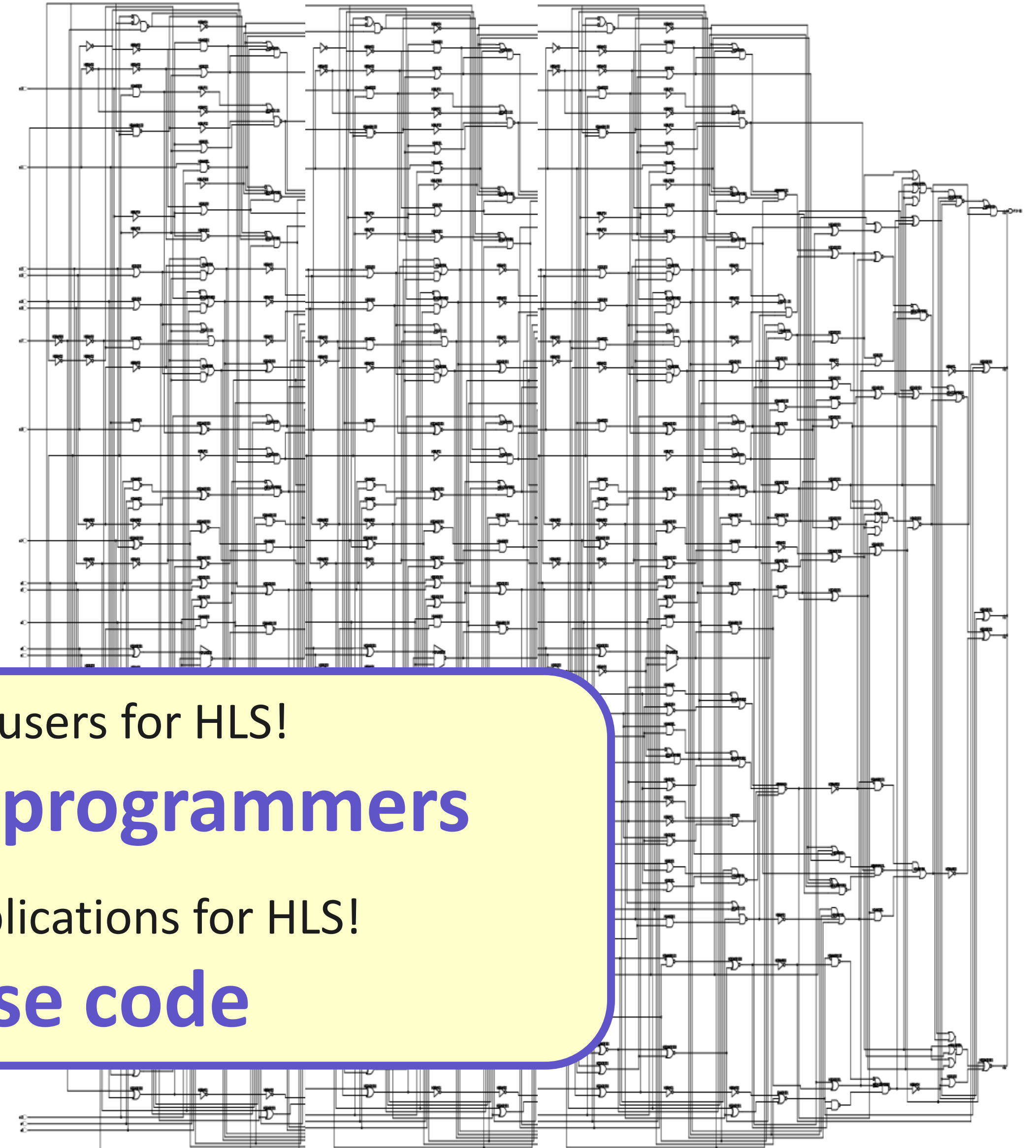
High-Level Synthesis: From Programs to Circuits

```
#define PI 3.14159265358979323846264 34

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!

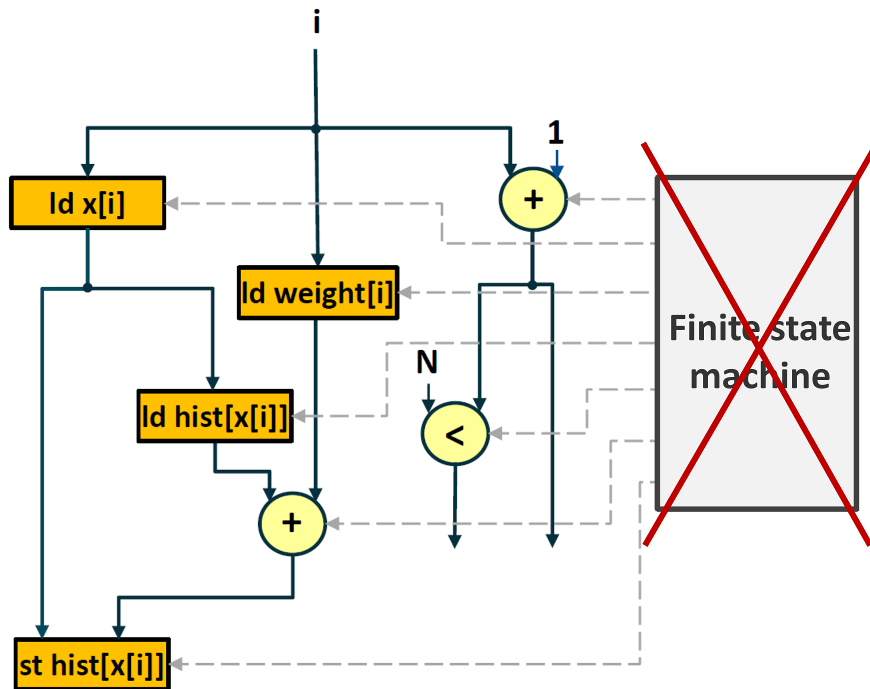
Software application programmers

A completely new type of applications for HLS!

General-purpose code

A Different Way to Do HLS

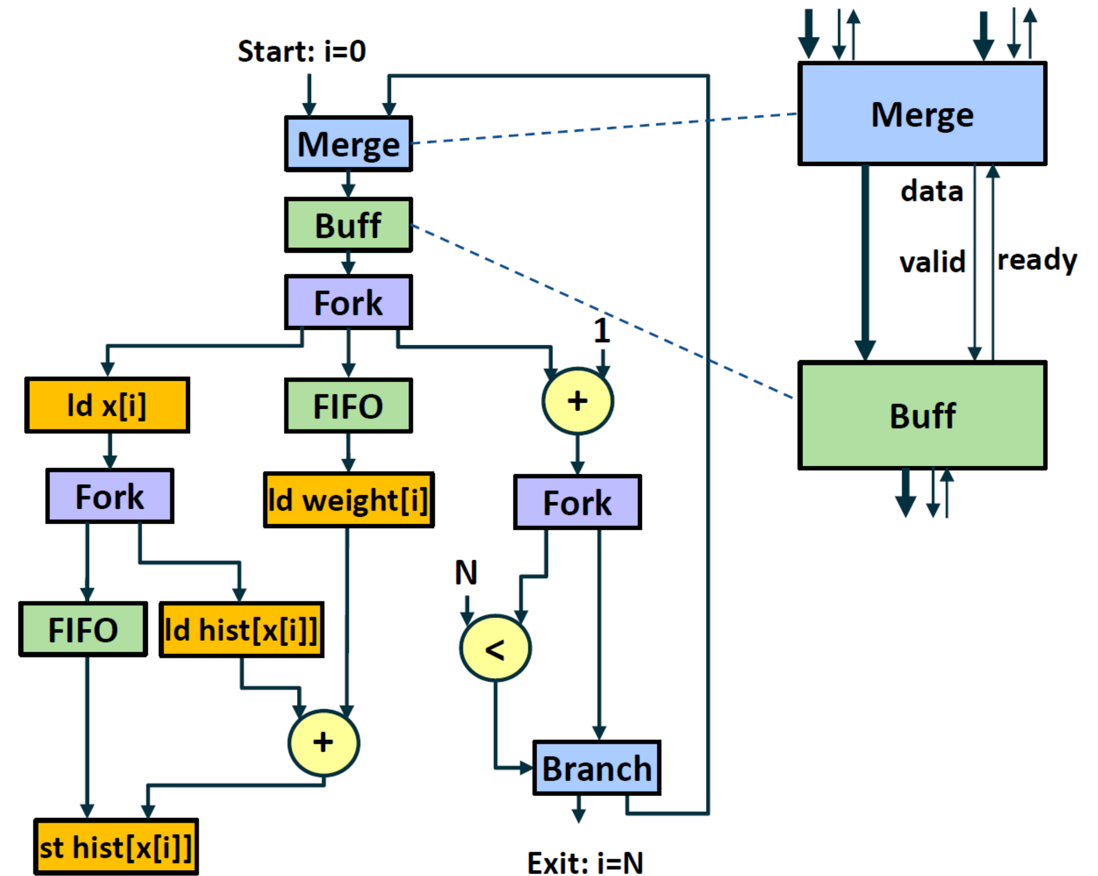
Static scheduling (standard HLS tool): decide at **compile time** when each operation executes



Circuit regulated by a centralized FSM

→ All execution times predetermined and, sometimes, conservative (slow circuit)

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

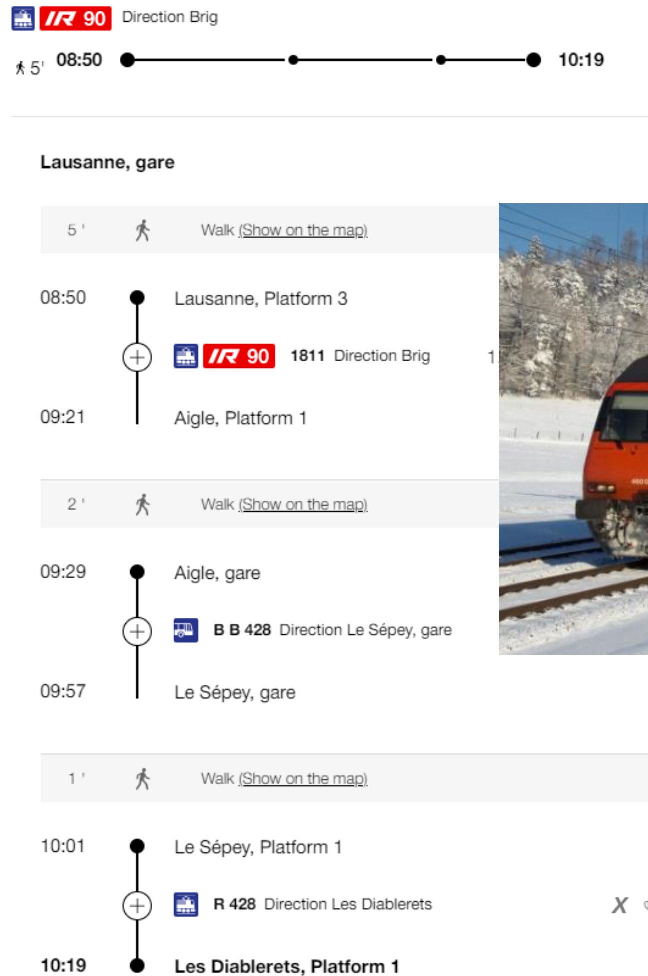


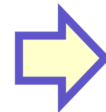
Circuit regulated by distributed handshake logic

→ Flexible execution times (fast circuit)

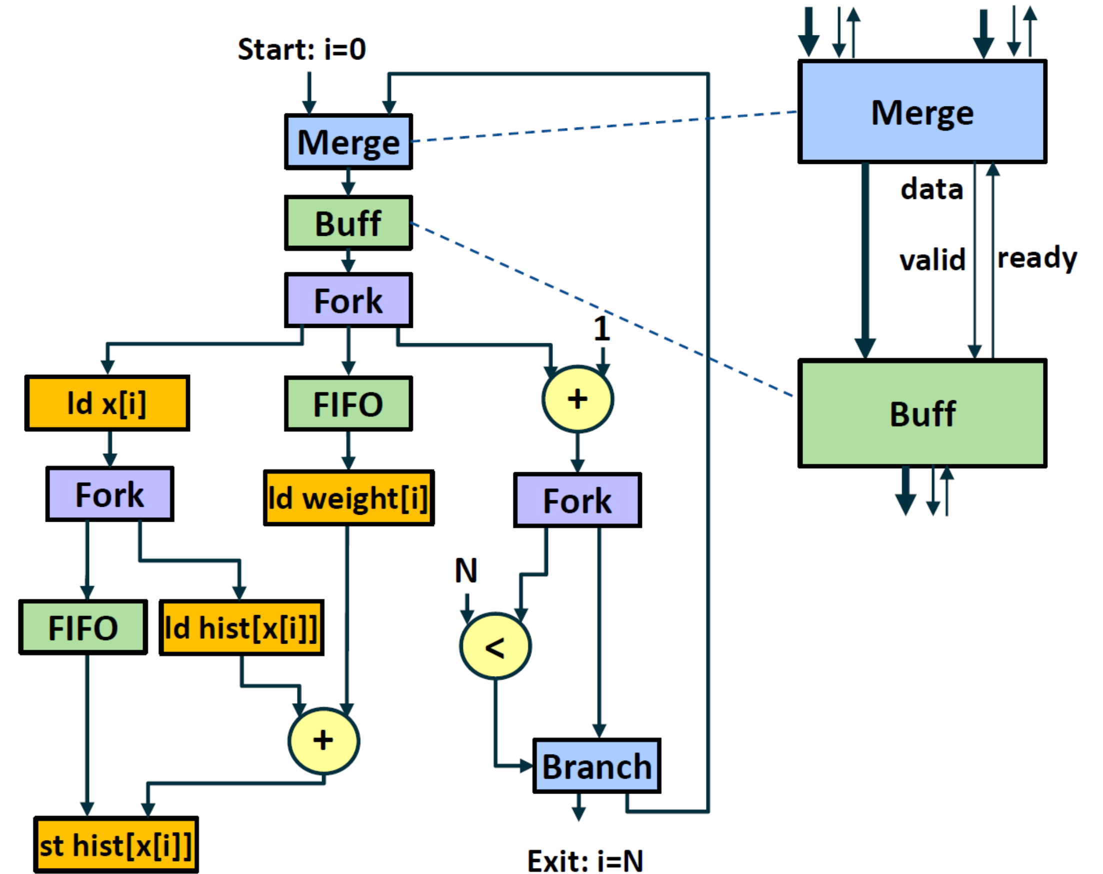
A Different Way to Do HLS

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



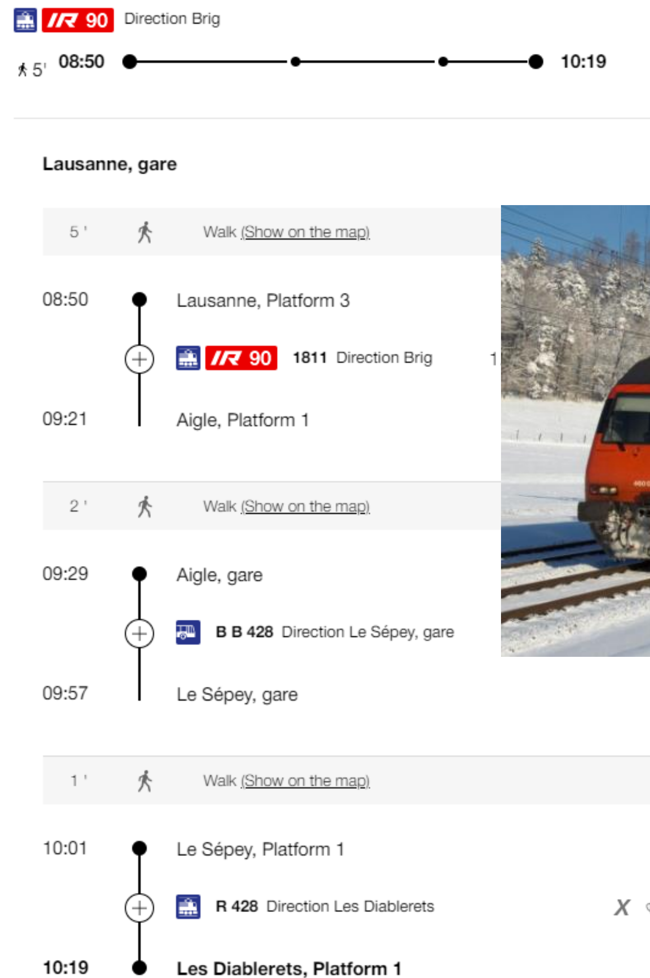


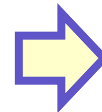
Dynamic scheduling (our HLS approach): decide at **runtime** when each operation executes



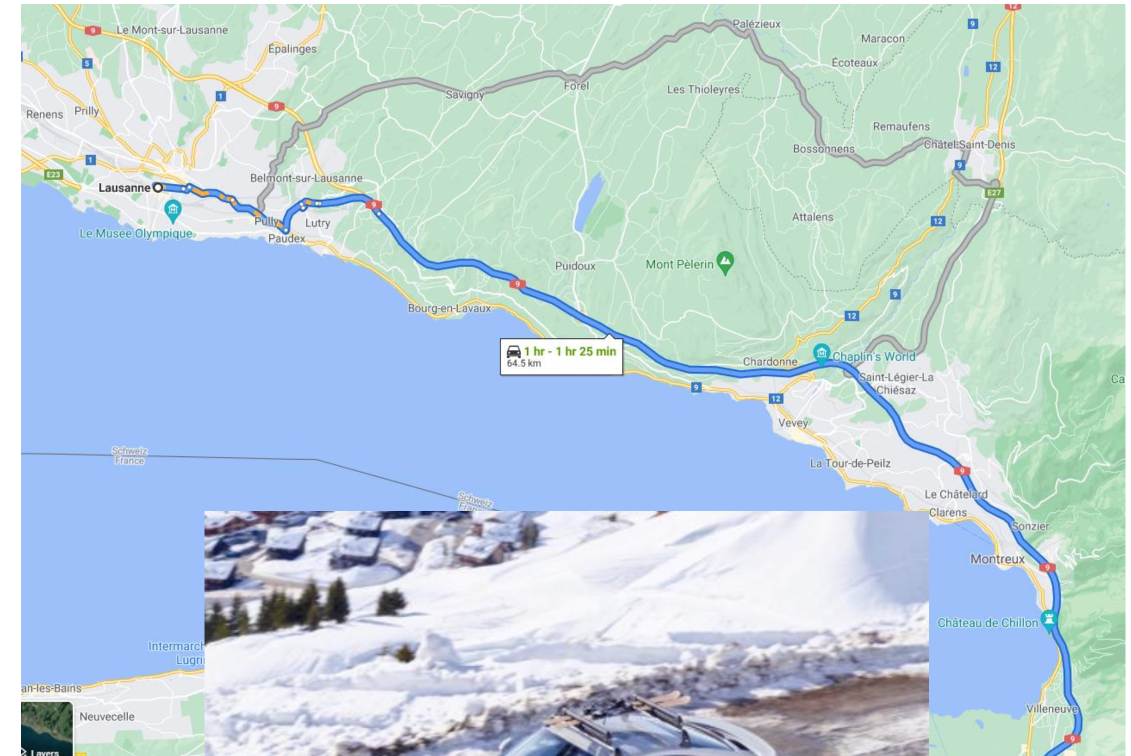
A Different Way to Do HLS

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



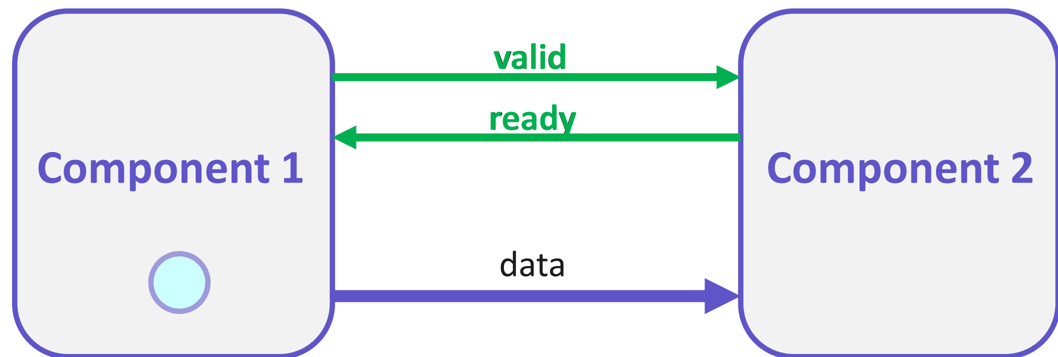


Dynamic scheduling (our HLS approach): decide at **runtime** when each operation executes



Dynamically Scheduled 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.



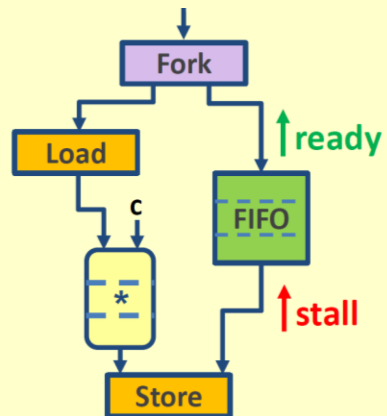
Make scheduling decisions at runtime: as soon as all conditions for execution are satisfied, an operation starts

High-level synthesis of dynamically scheduled (dataflow) circuits

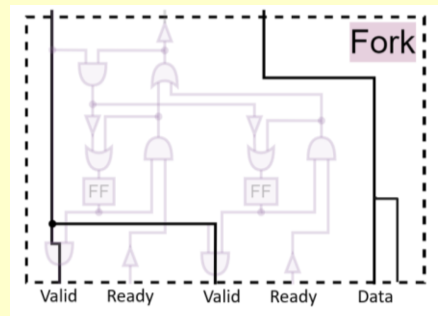
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

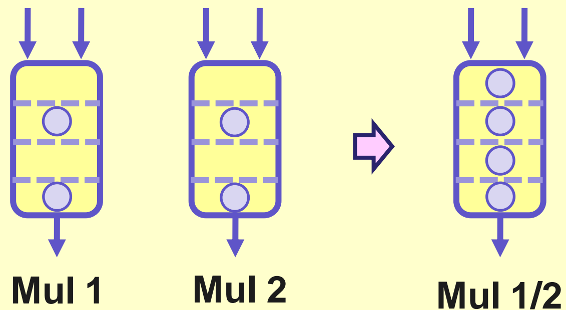
Pipelining



Removing redundant handshake logic

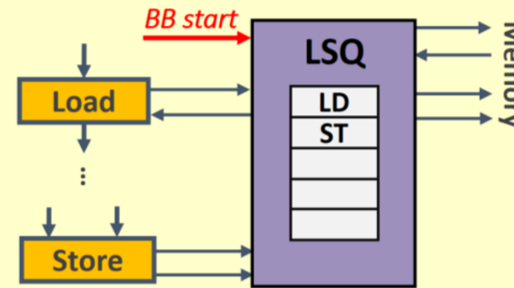


Resource sharing

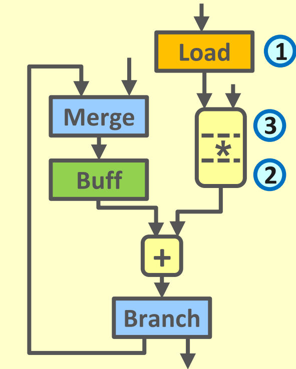


Reaping the benefits of dynamic scheduling

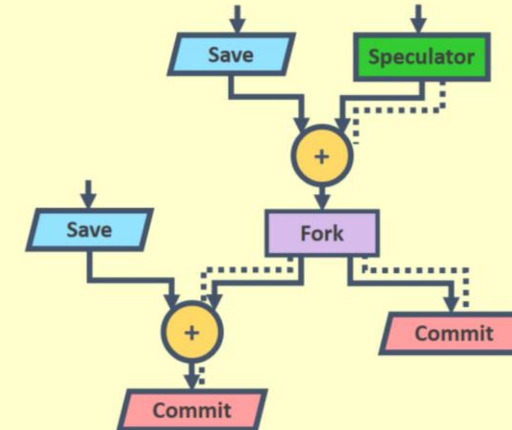
Out-of-order memory



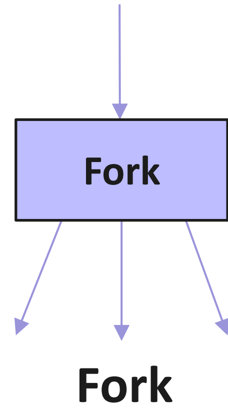
Tagged out-of-order execution



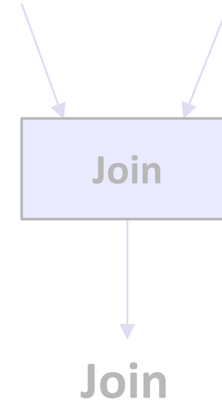
Speculative execution



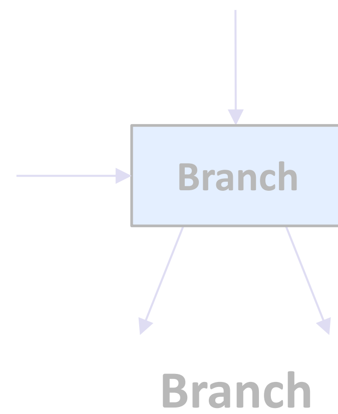
Dataflow Components



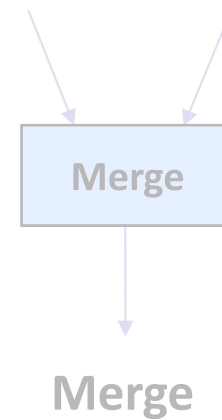
Fork



Join

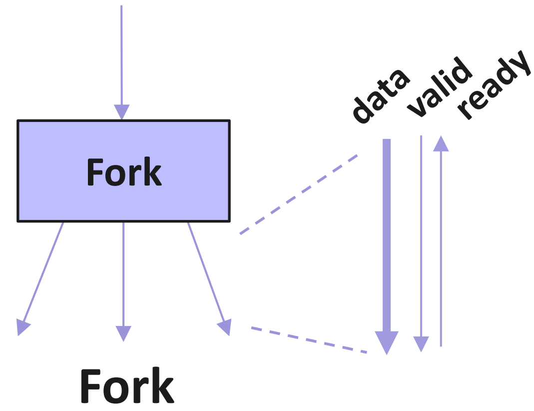


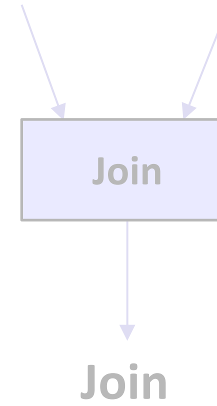
Branch

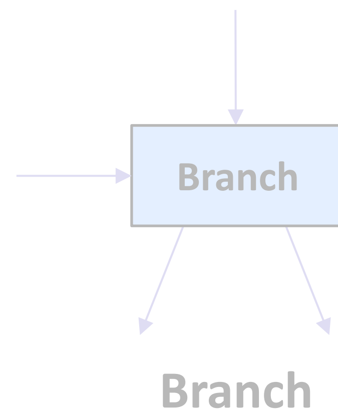


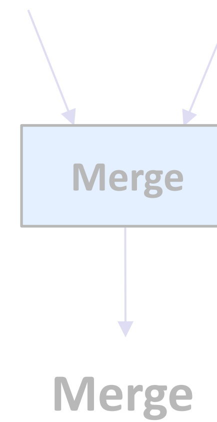
Merge

Dataflow Components

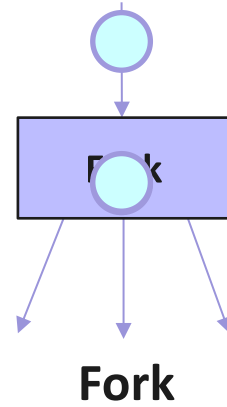




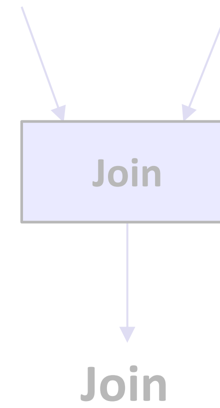




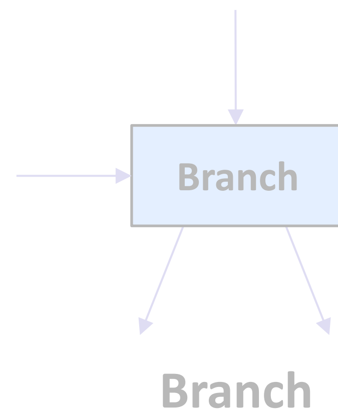
Dataflow Components



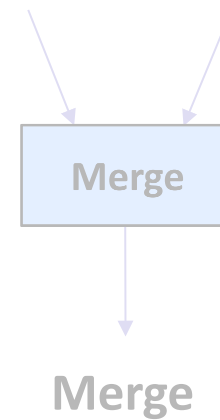
Fork



Join

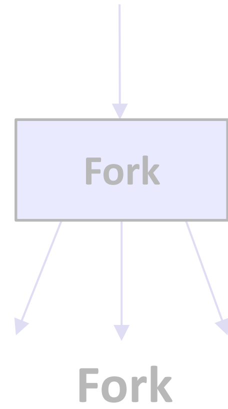


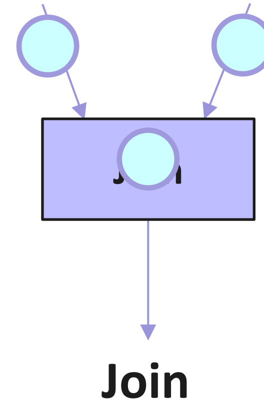
Branch

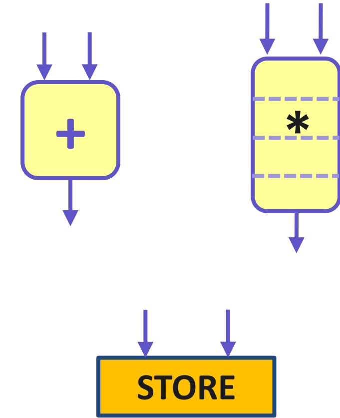


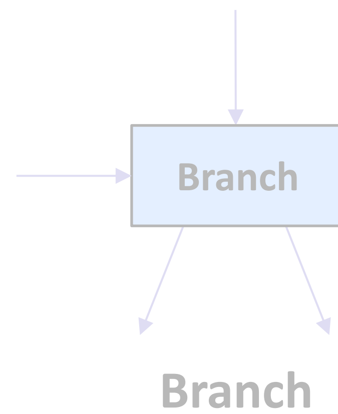
Merge

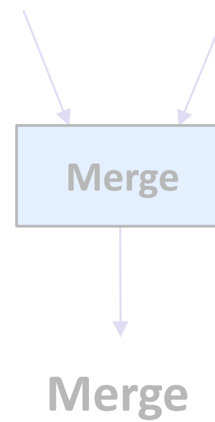
Dataflow Components



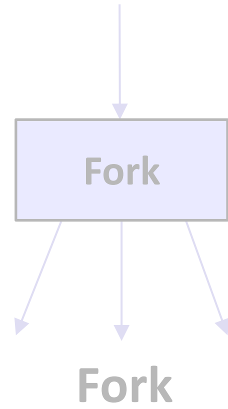




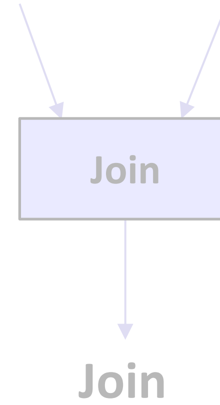




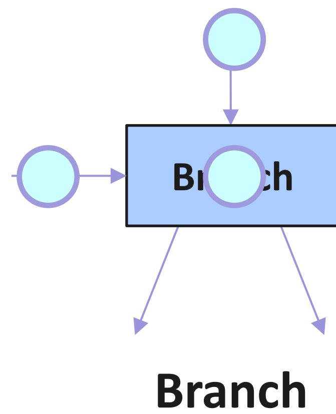
Dataflow Components



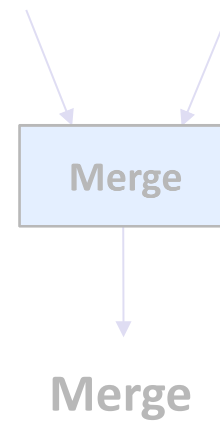
Fork



Join

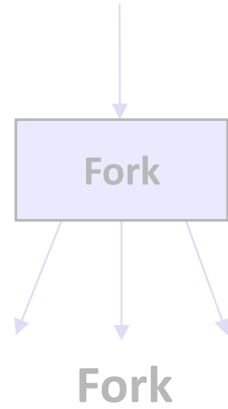


Branch

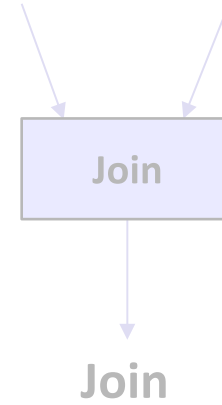


Merge

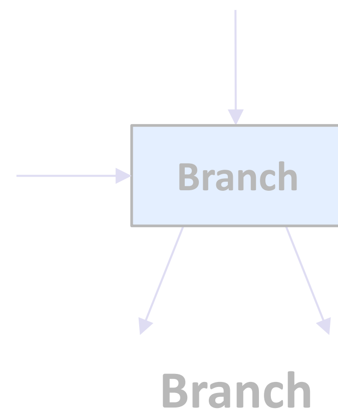
Dataflow Components



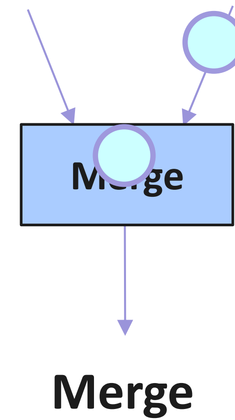
Fork



Join

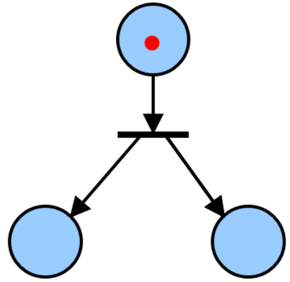


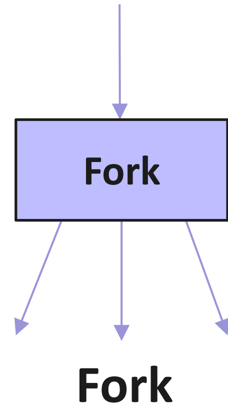
Branch



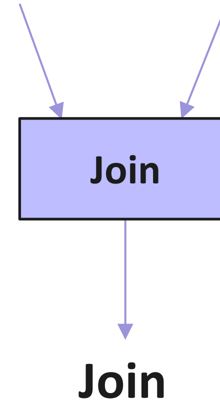
Merge

Dataflow Components

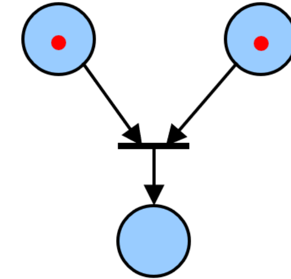


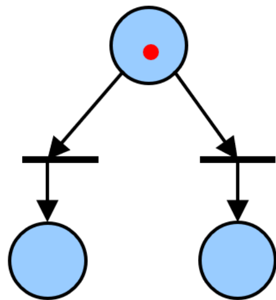


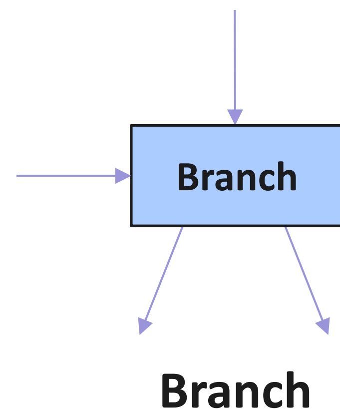
Fork



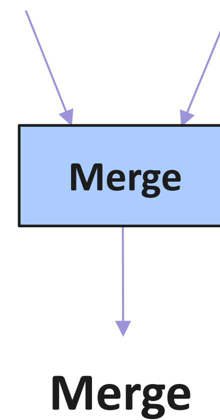
Join



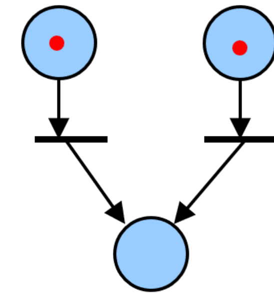




Branch

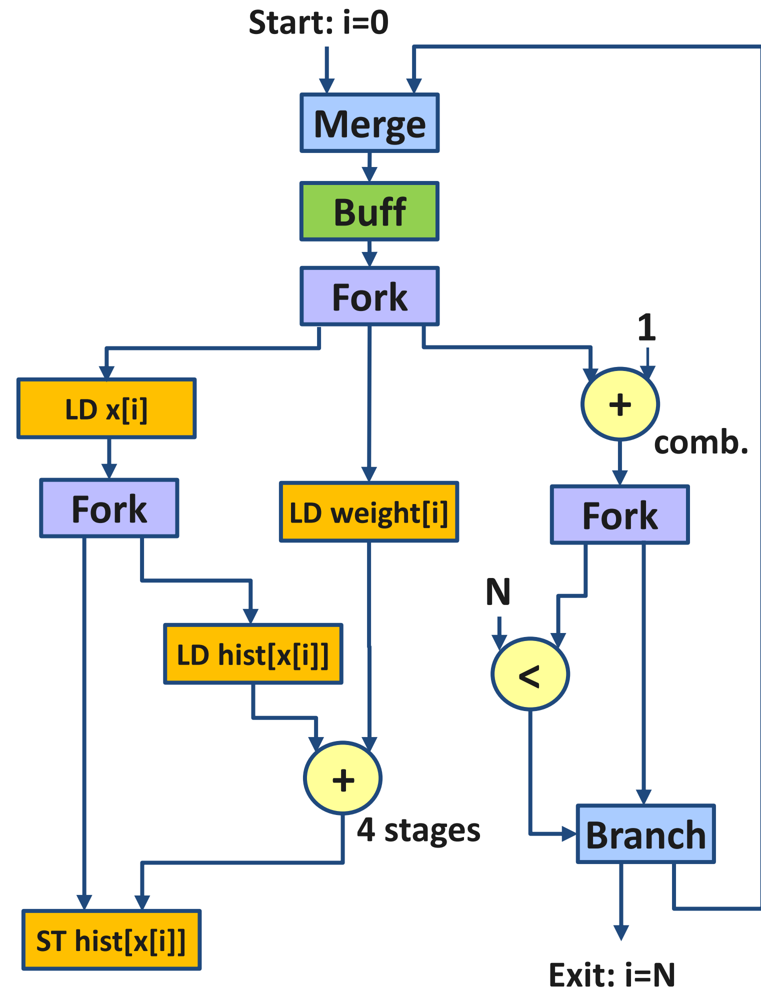


Merge



Petri nets for dataflow circuit modeling

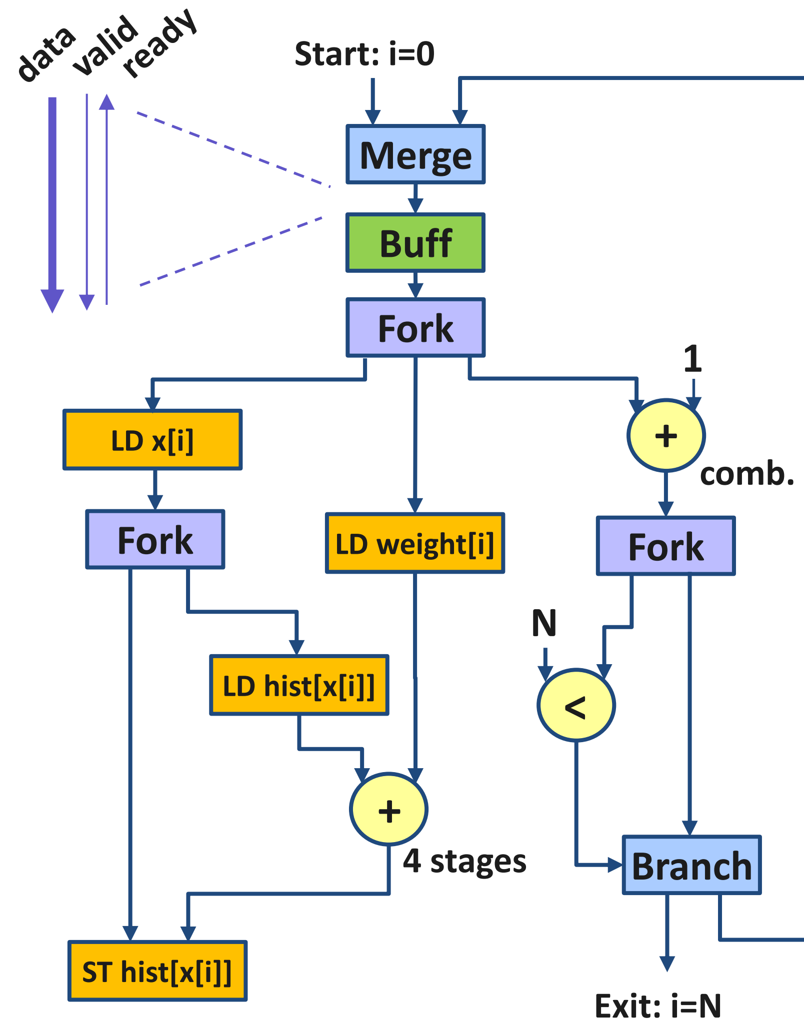
From Program to Dataflow Circuit




```
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

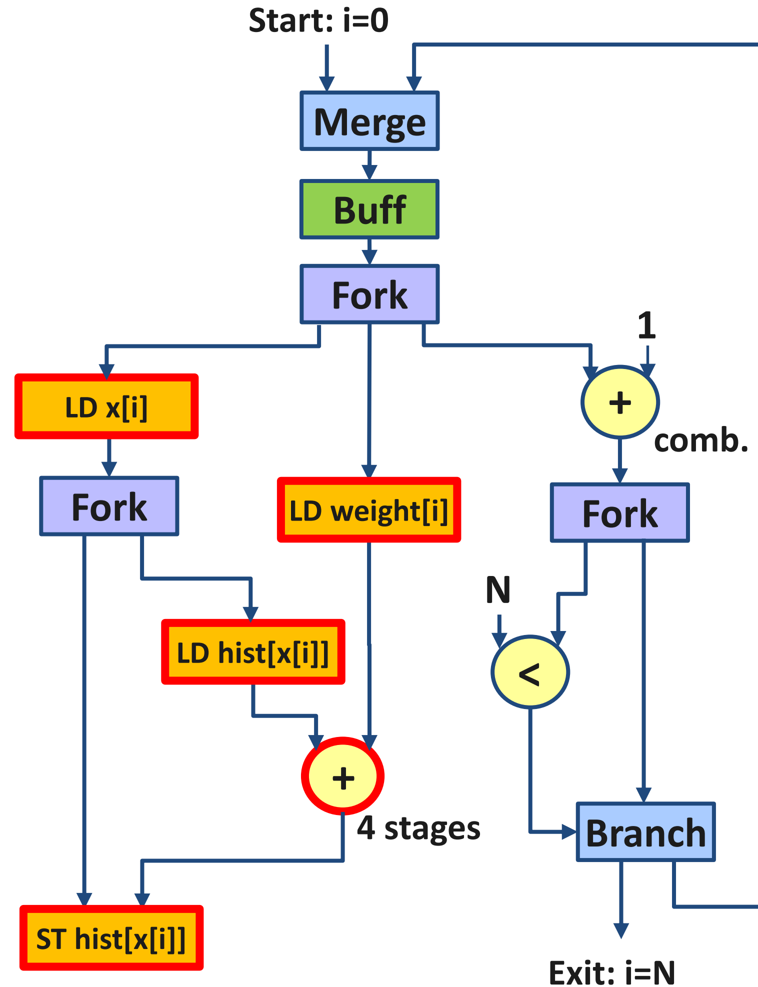
From Program to Dataflow Circuit




```
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

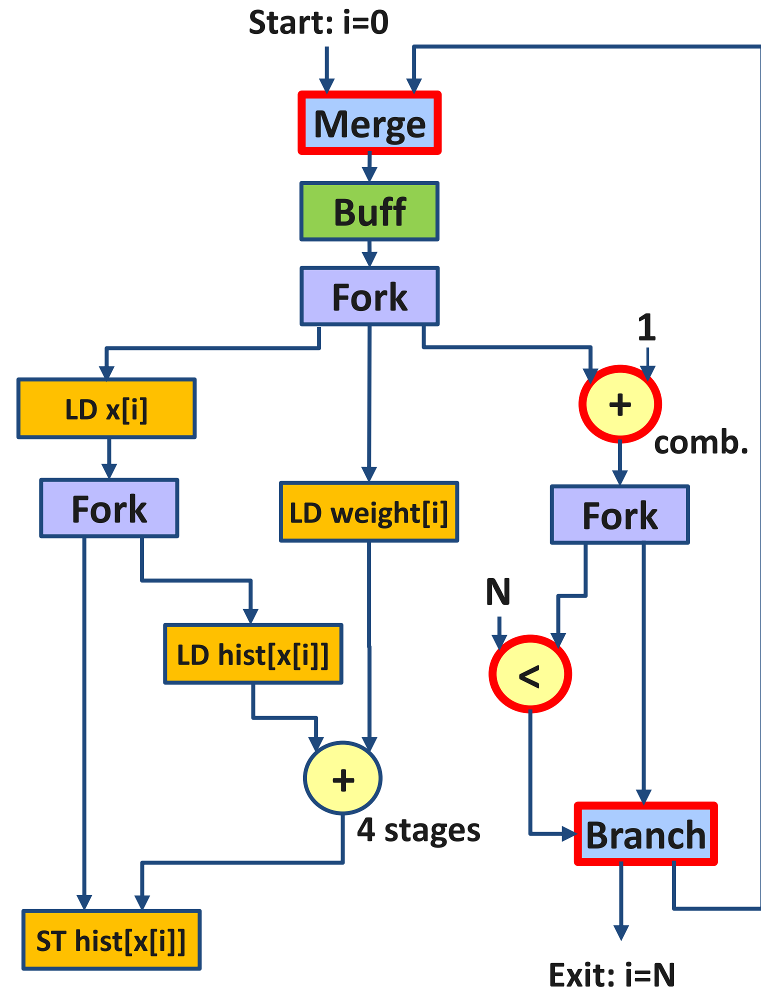
From Program to Dataflow Circuit




```
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

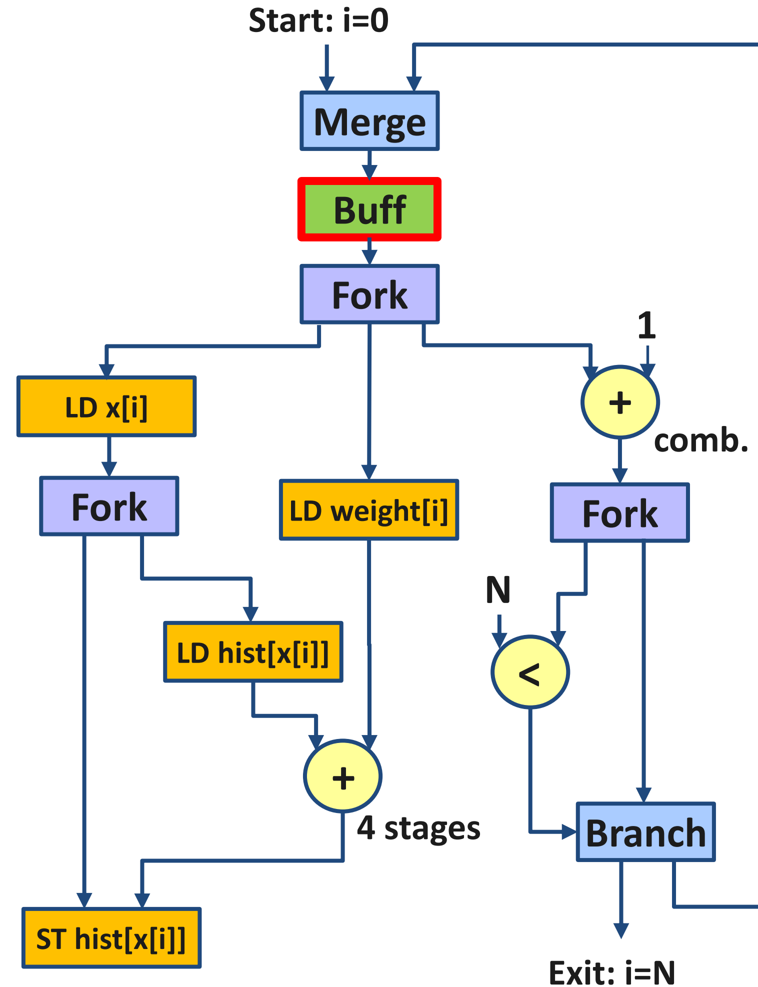
From Program to Dataflow Circuit




```
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

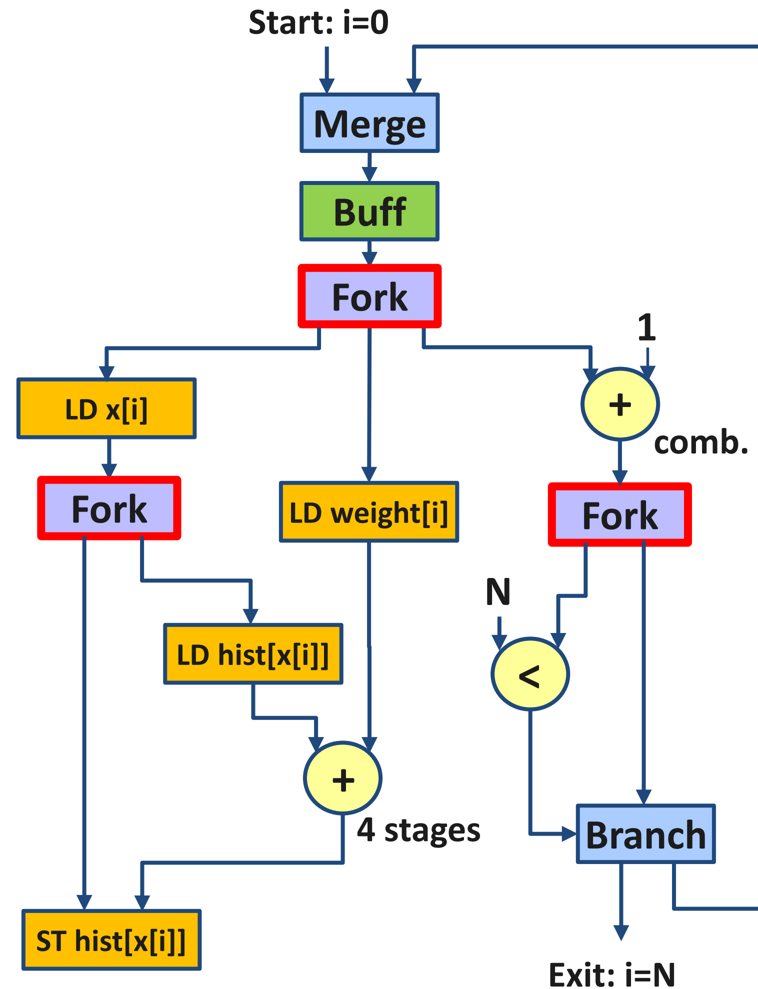
From Program to Dataflow Circuit




```
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

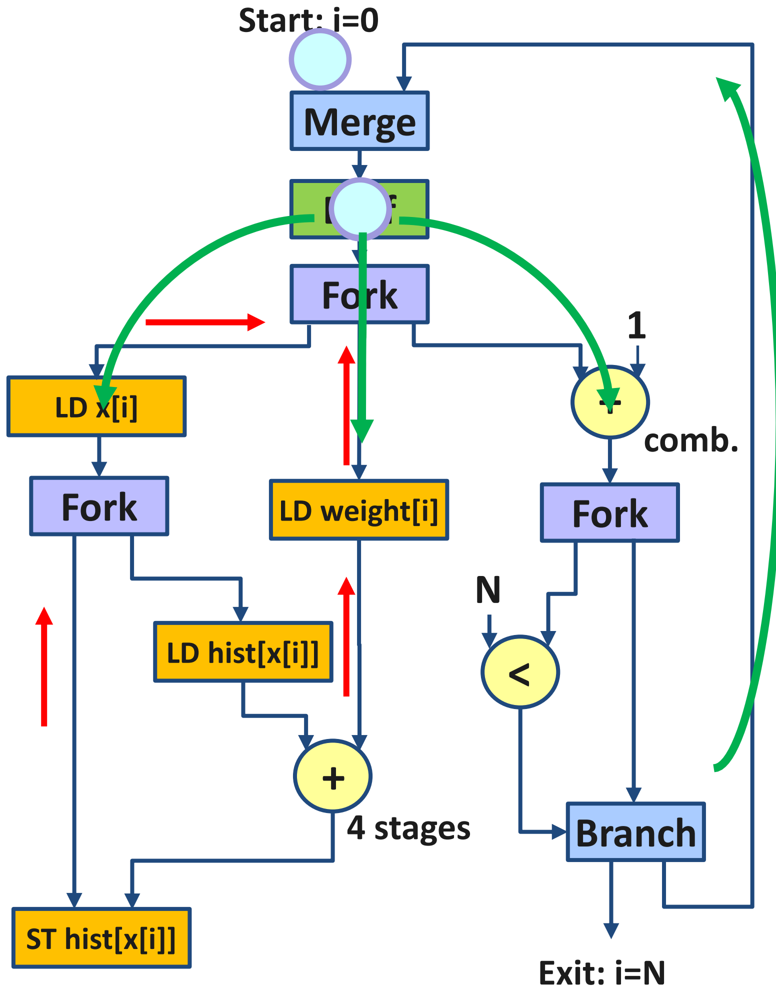
From Program to Dataflow Circuit




```
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



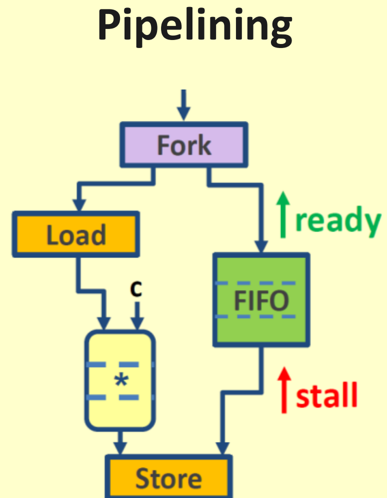
Backpressure due to insufficient token capacity: no pipelining and low performance

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

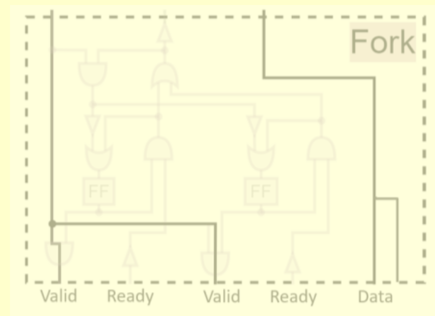
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

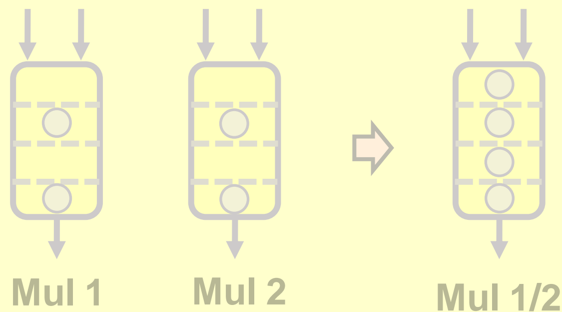
Pipelining



Removing redundant handshake logic

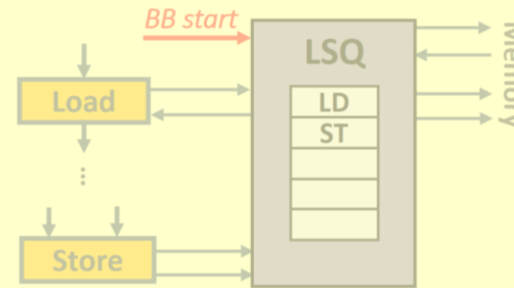


Resource sharing

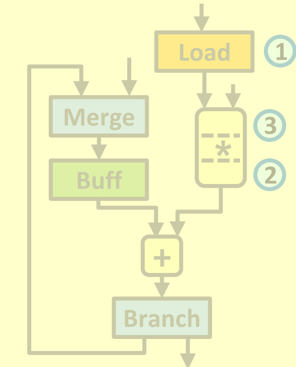


Reaping the benefits of dynamic scheduling

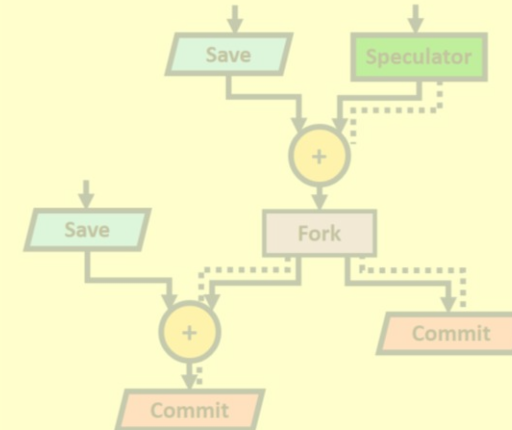
Out-of-order memory



Tagged out-of-order execution

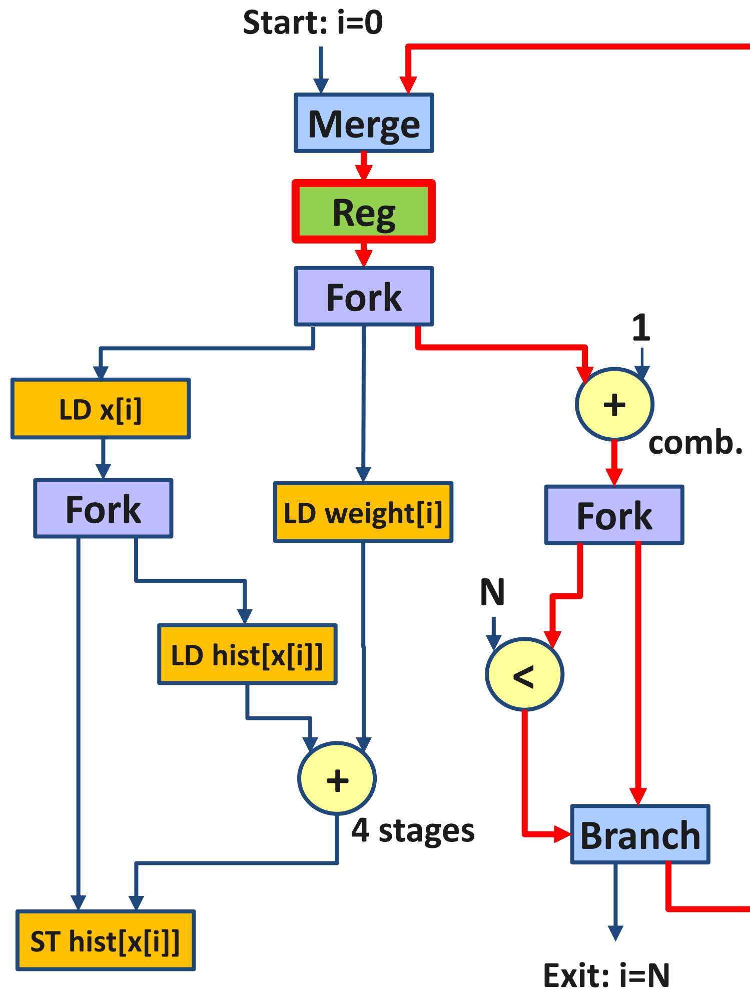


Speculative execution



Inserting Buffers

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

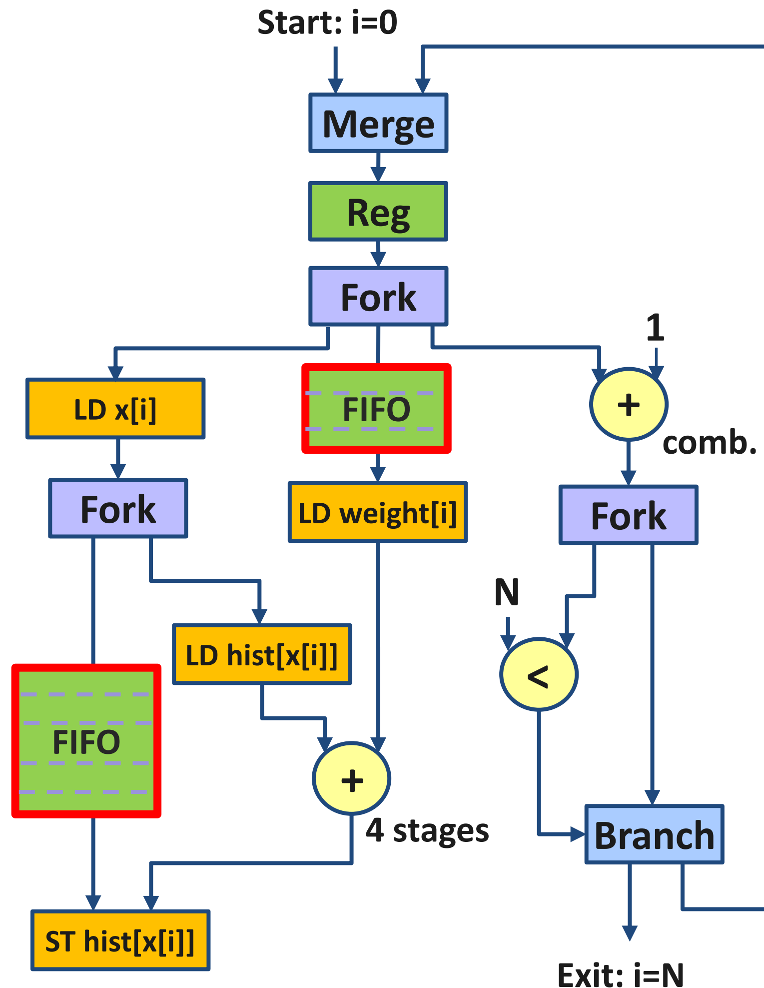


Buffers as registers to break combinational paths

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];
}
```



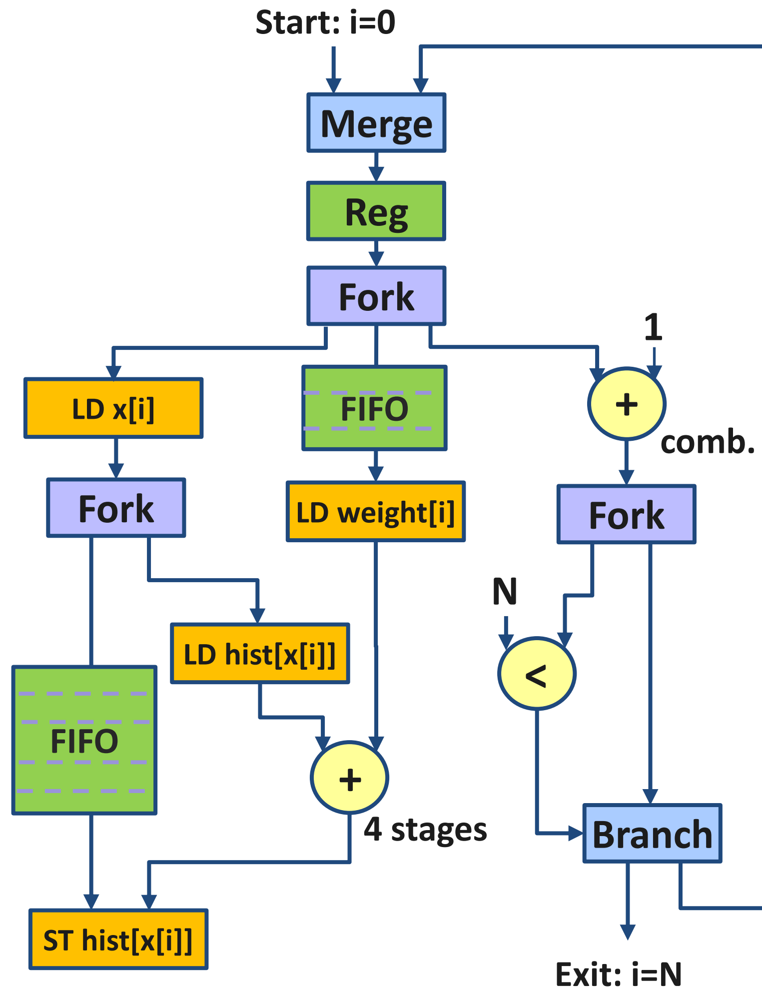
Buffers as FIFOs to regulate throughput

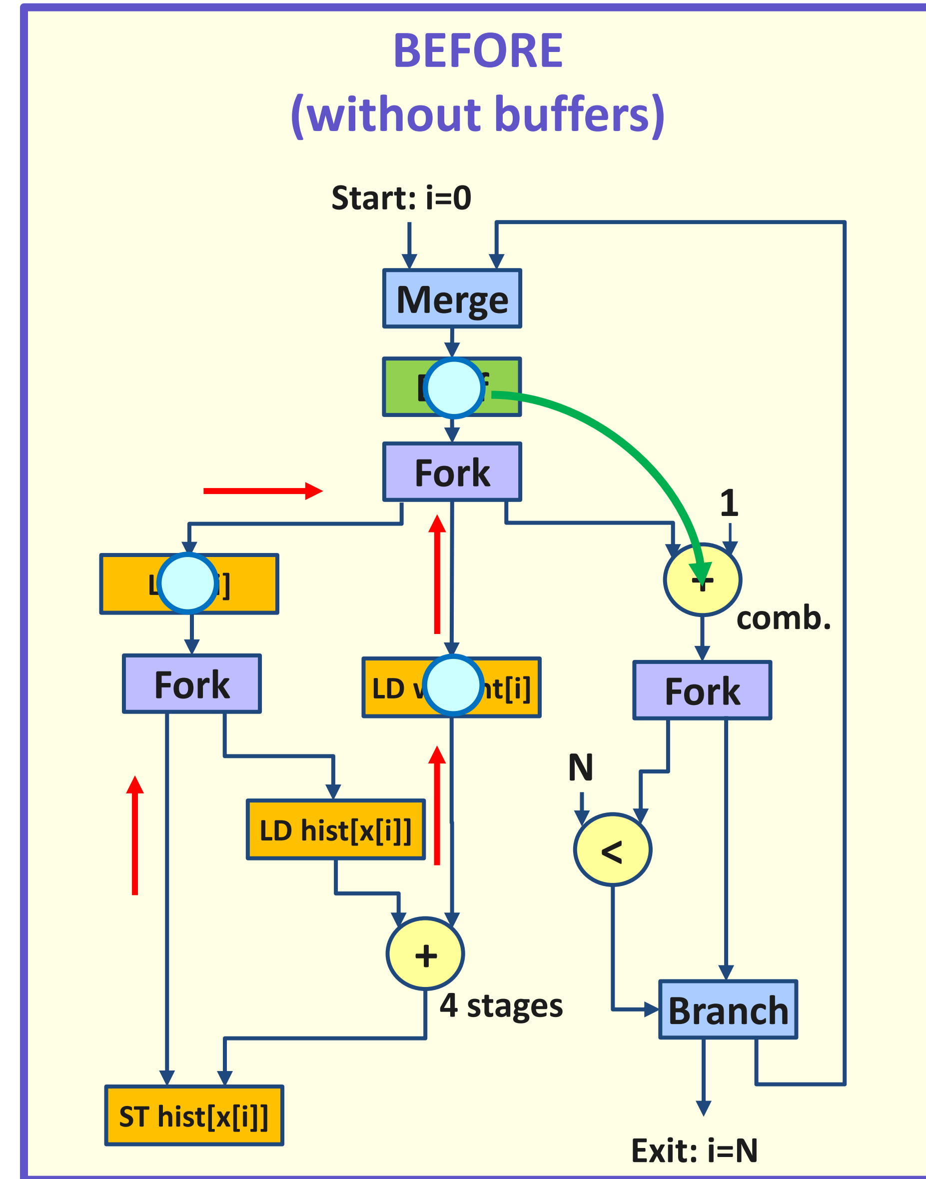
Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

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)

Start: i=0

Merge

Fork

1

LD x[i]

FIFO

comb.

Fork

LD weight[i]

Fork

N

LD hist[x[i]]

<

FIFO

+

4 stages

Branch

ST hist[x[i]]

Exit: i=N

BEFORE (without buffers)

Start: i=0

Merge

Fork

1

LD x[i]

comb.

Fork

LD weight[i]

Fork

N

LD hist[x[i]]

<

+

4 stages

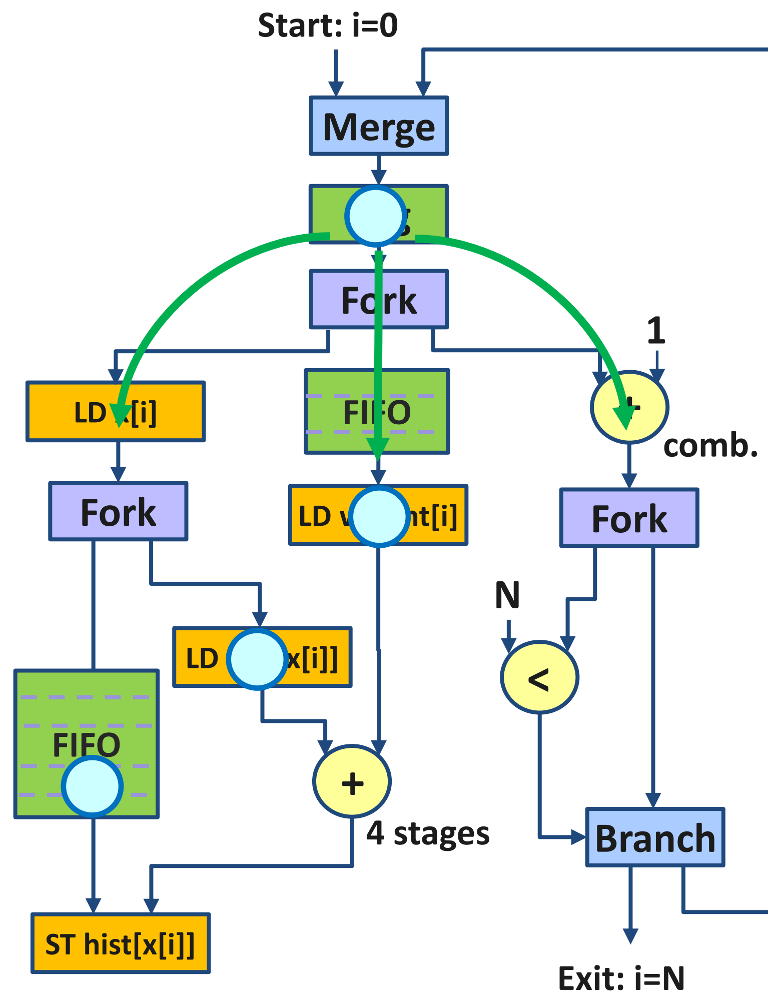
Branch

ST hist[x[i]]

Exit: i=N

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];
}
```

Represent program loops as **choice-free Petri nets**

- Analyze average token flow through the circuit **(continuous Petri net)**
- Determine buffer positions & sizes **(token capacity)**
- **Maximize throughput** for a target clock period

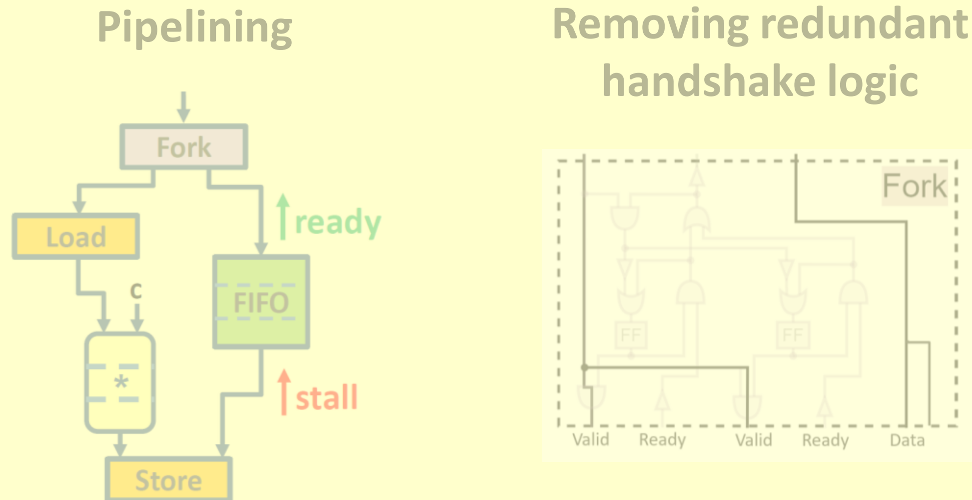
Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**
Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022

HLS of Dynamically Scheduled Circuits

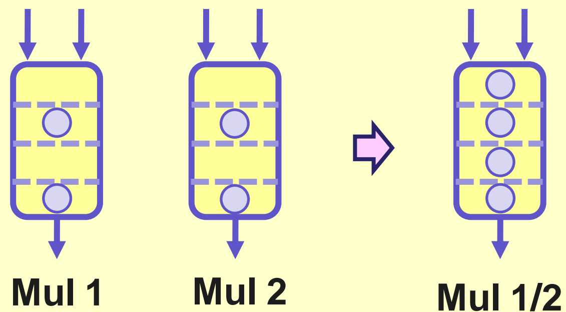
Catching up with static HLS

Pipelining

Removing redundant handshake logic

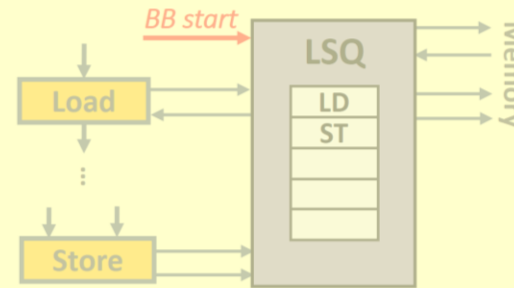


Resource sharing

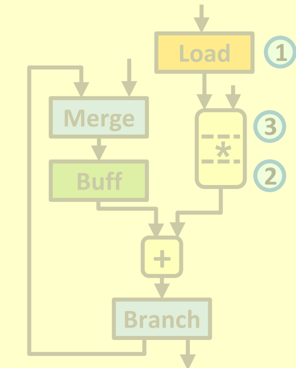


Reaping the benefits of dynamic scheduling

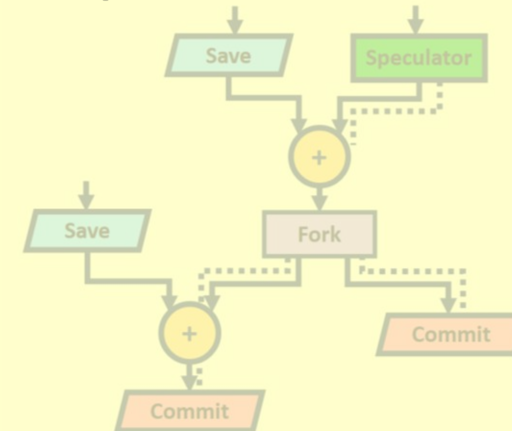
Out-of-order memory



Tagged out-of-order execution



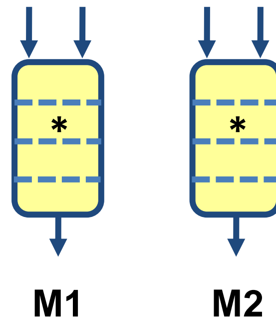
Speculative execution



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;
}
```

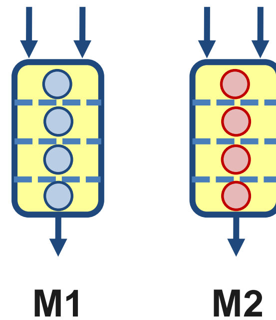


Josipović, Marmet, Guerrieri, and Ienne. Resource Sharing in Dataflow Circuits. FCCM 2022. **Best Paper Award Nominee**

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, II = 1)

Sharing not possible without damaging throughput

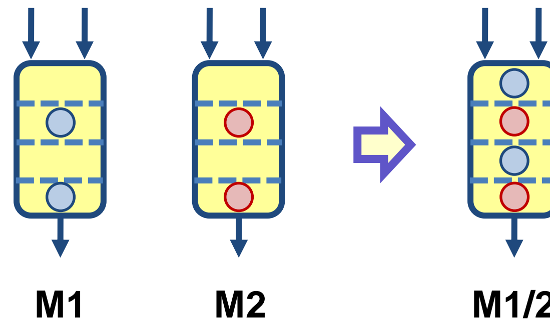
Use choice-free Petri net model to decide what to share

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;
}
```



Units underutilized (low throughput, II = 2)

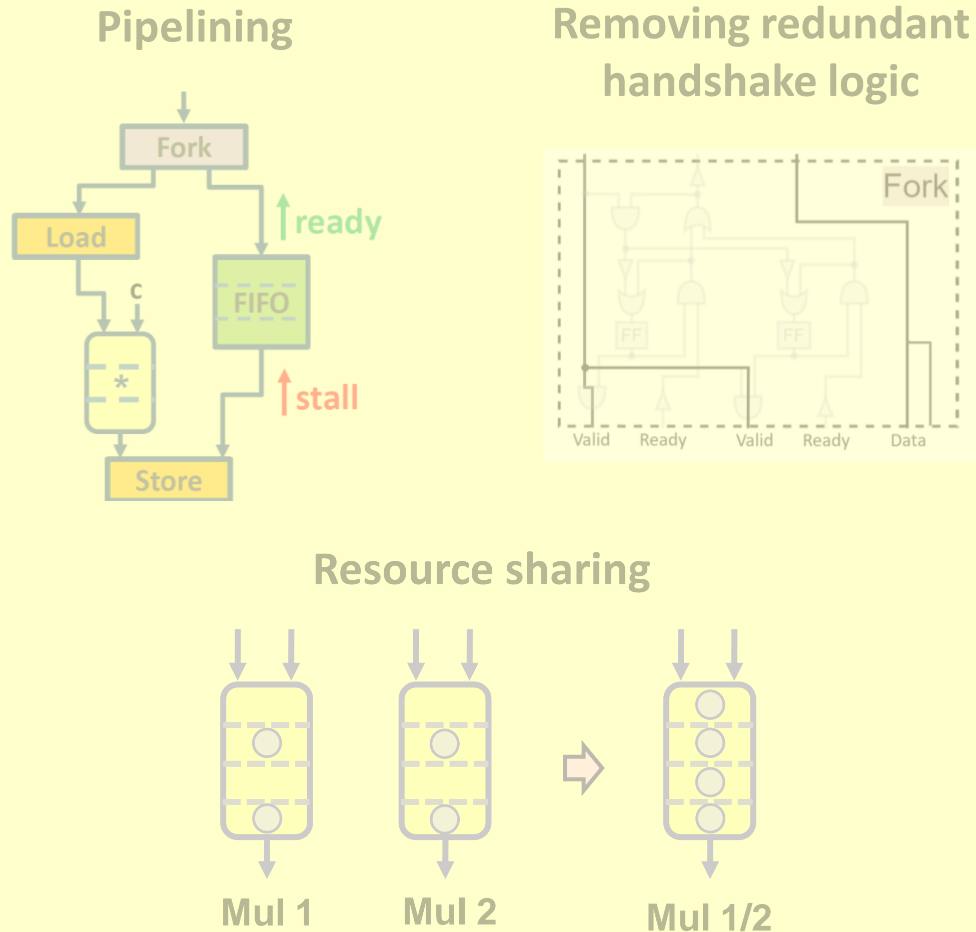
Sharing possible without damaging throughput

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**

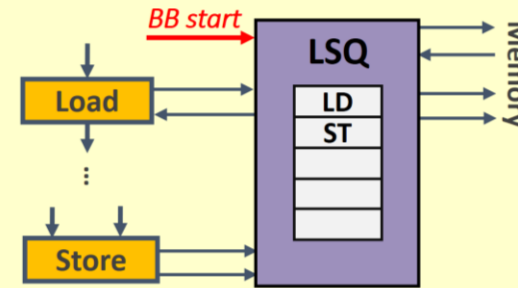
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

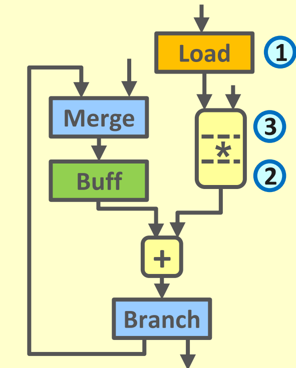


Reaping the benefits of dynamic scheduling

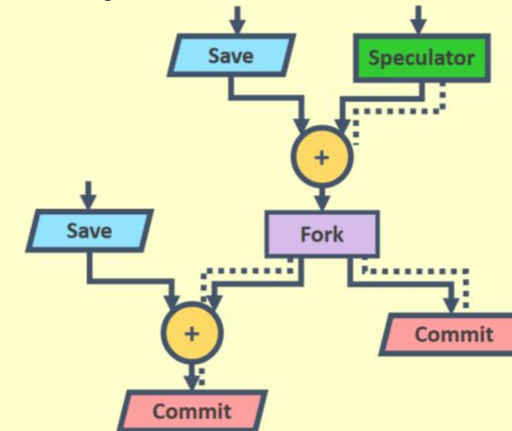
Out-of-order memory



Tagged out-of-order execution

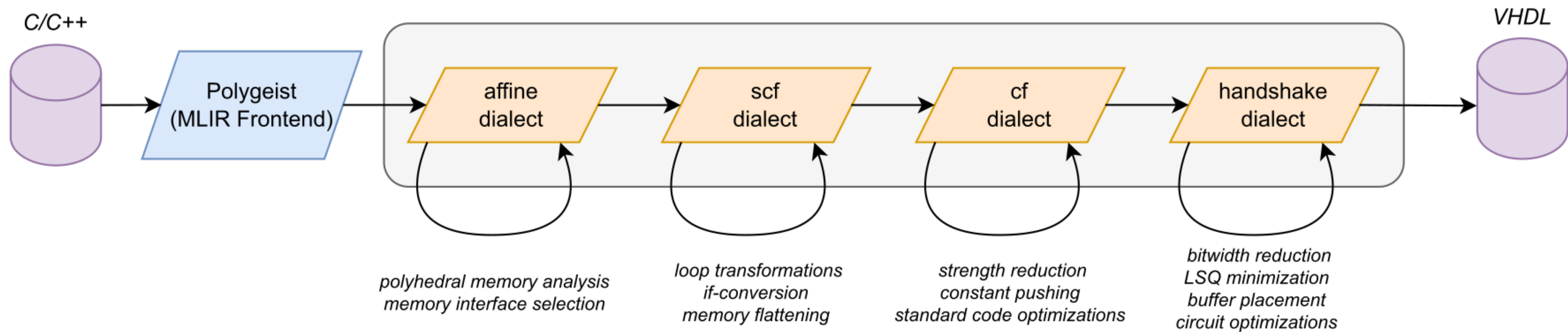


Speculative execution



Dynamatic: An Open-Source HLS Compiler

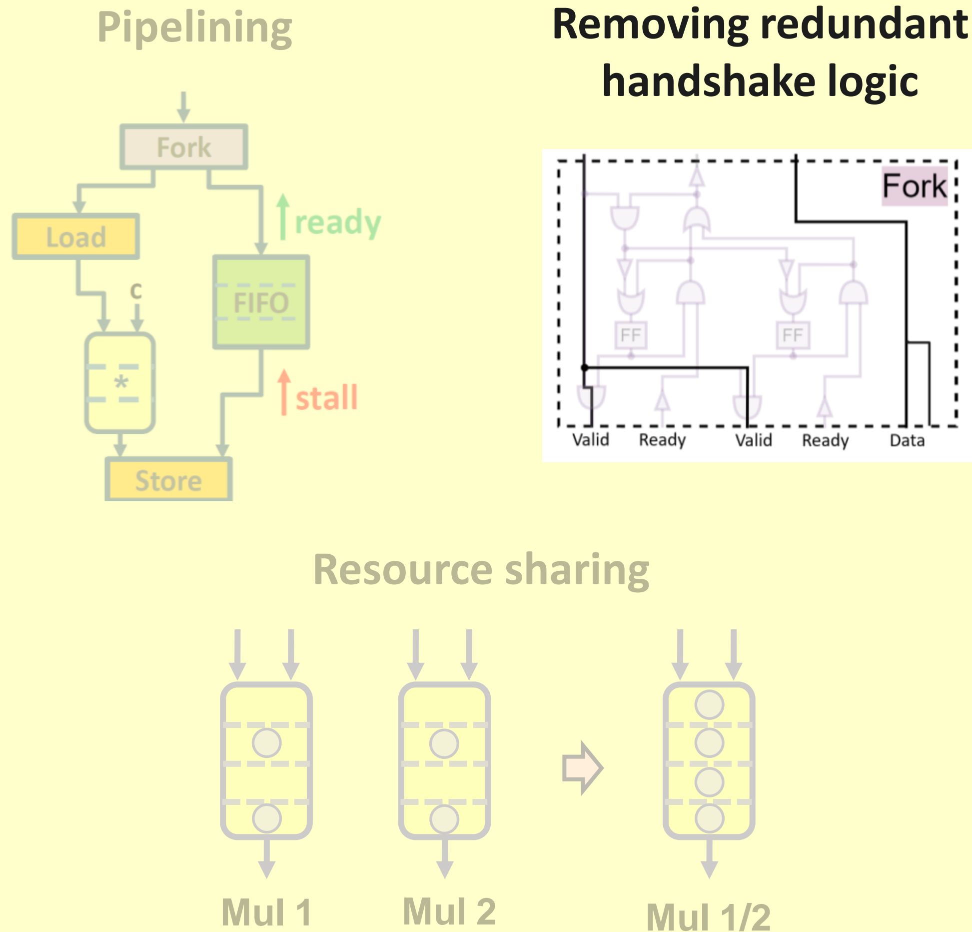
- From C/C++ to synthesizable dataflow circuit description

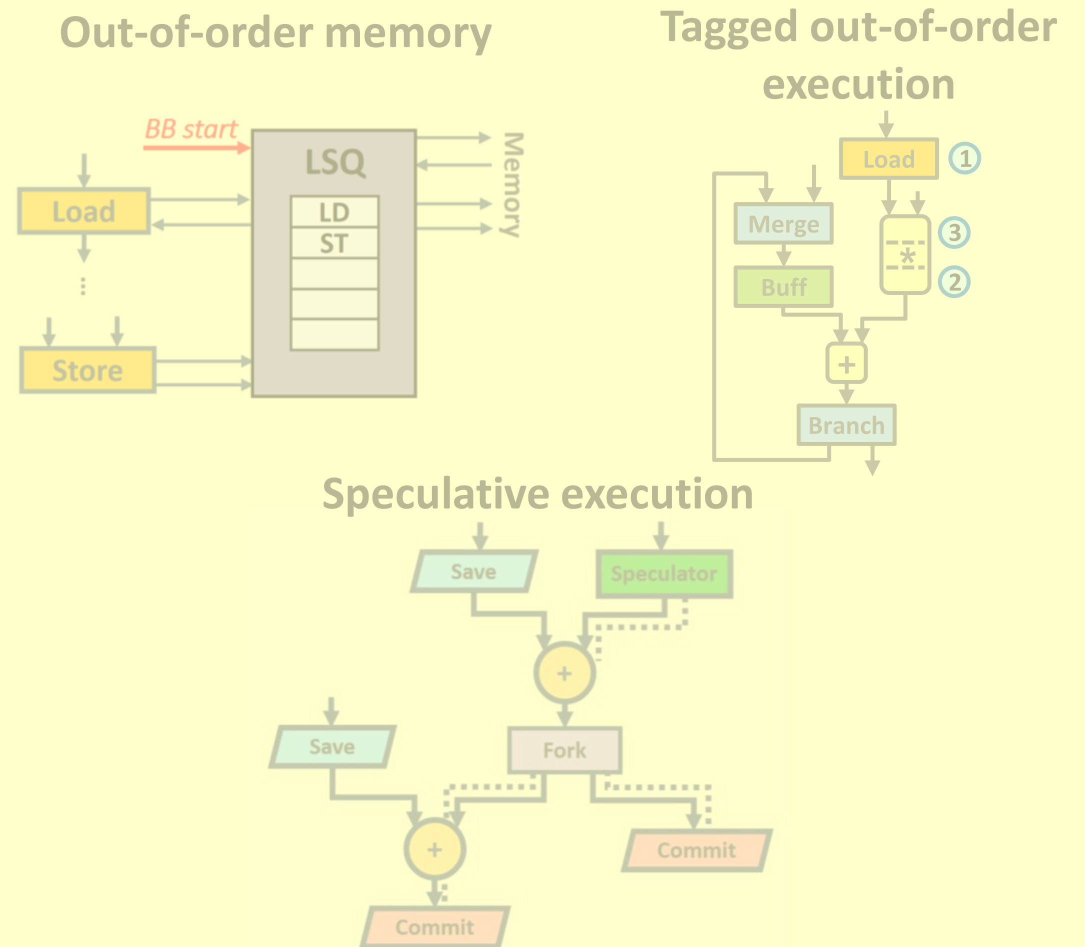


Reduced execution time in irregular benchmarks (speedup of up to 14.9X)

But… dataflow computation is resource-expensive!

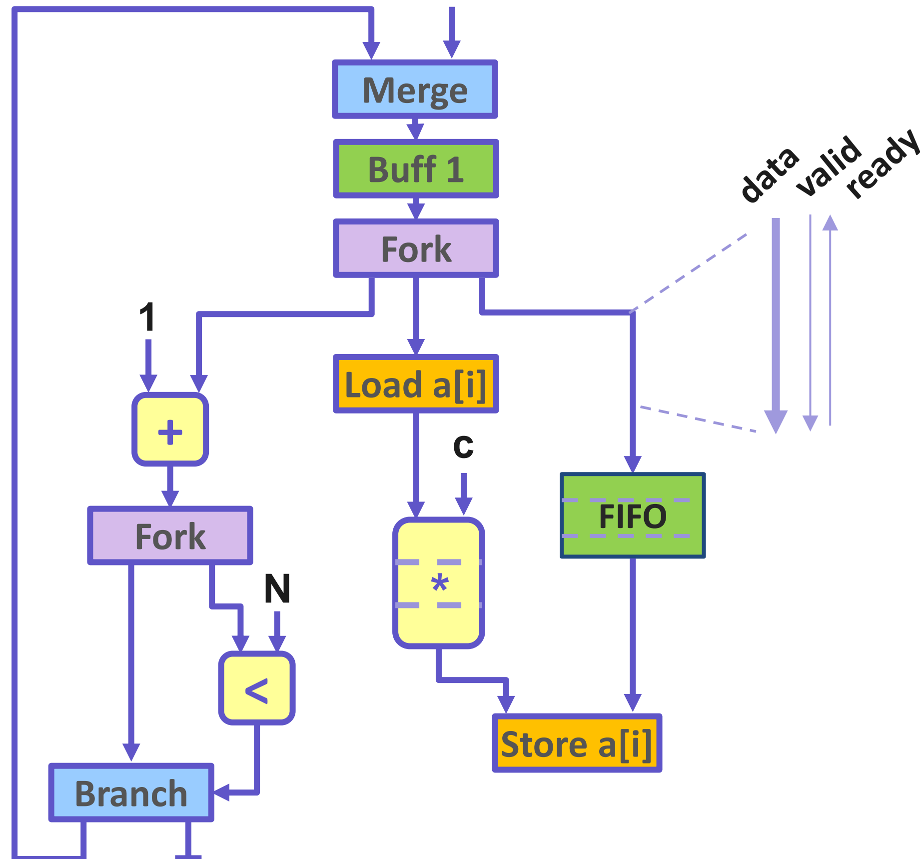
HLS of Dynamically Scheduled Circuits



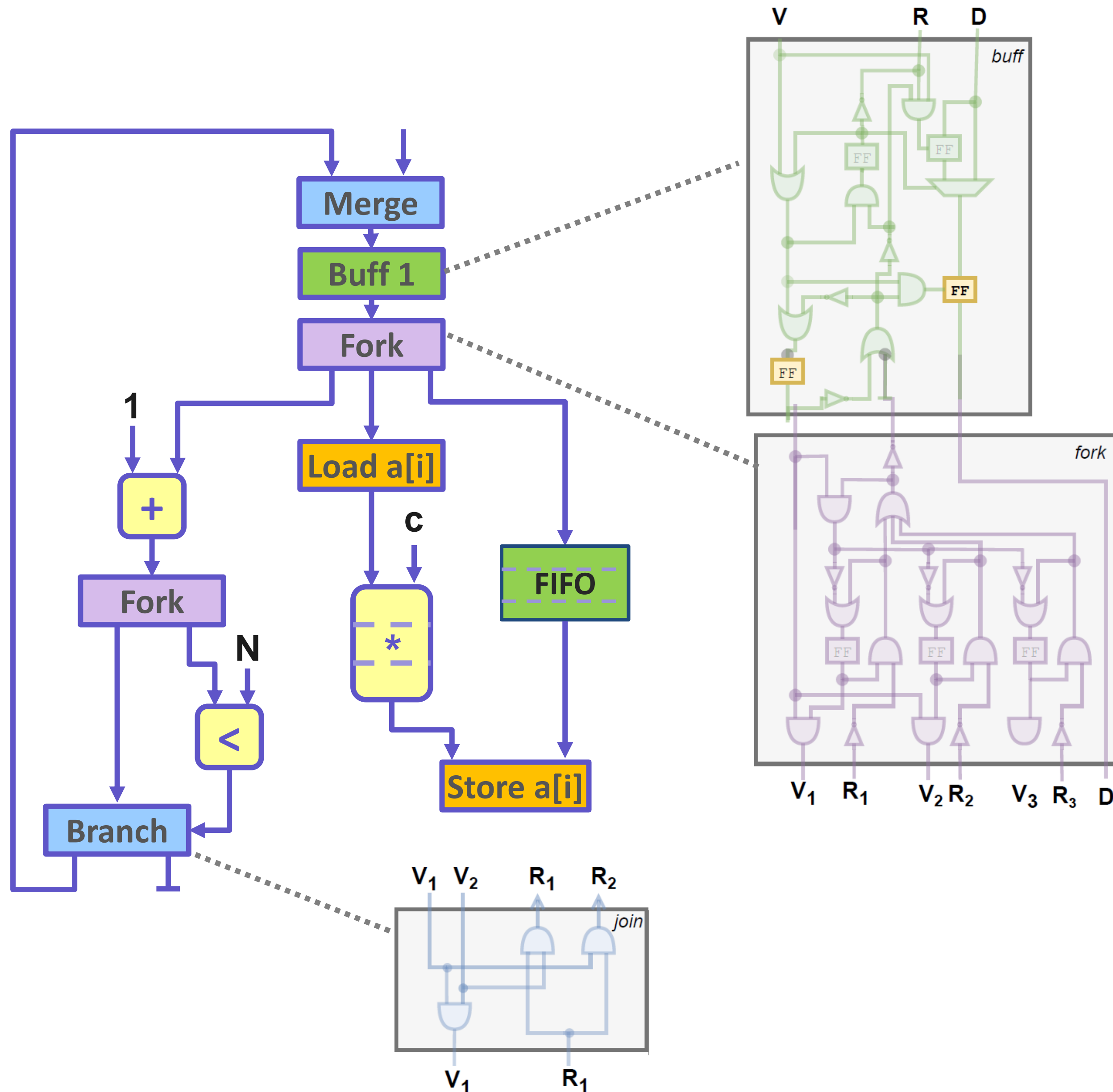


The Cost of Dataflow Computation

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

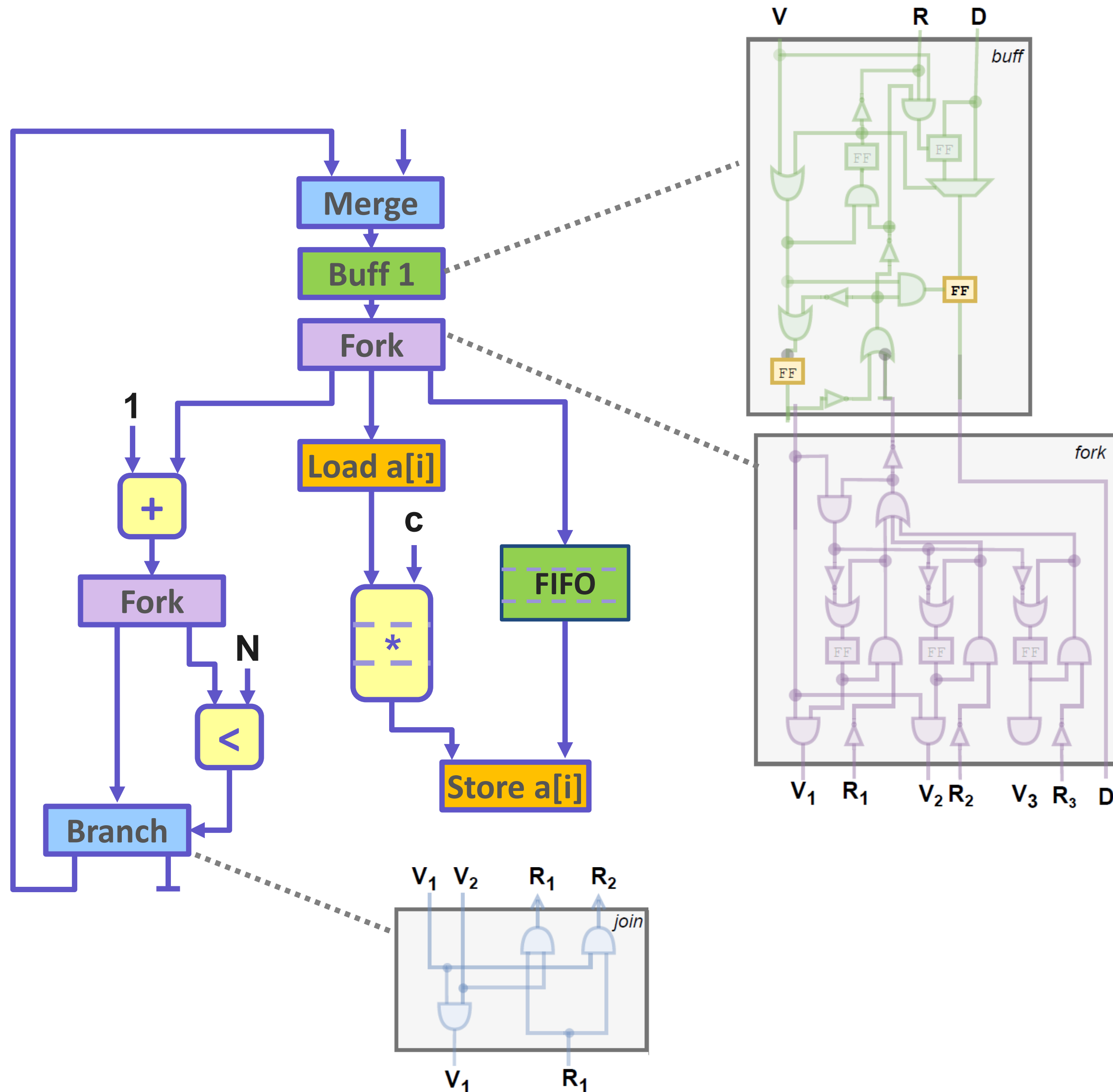


The Cost of Dataflow Computation



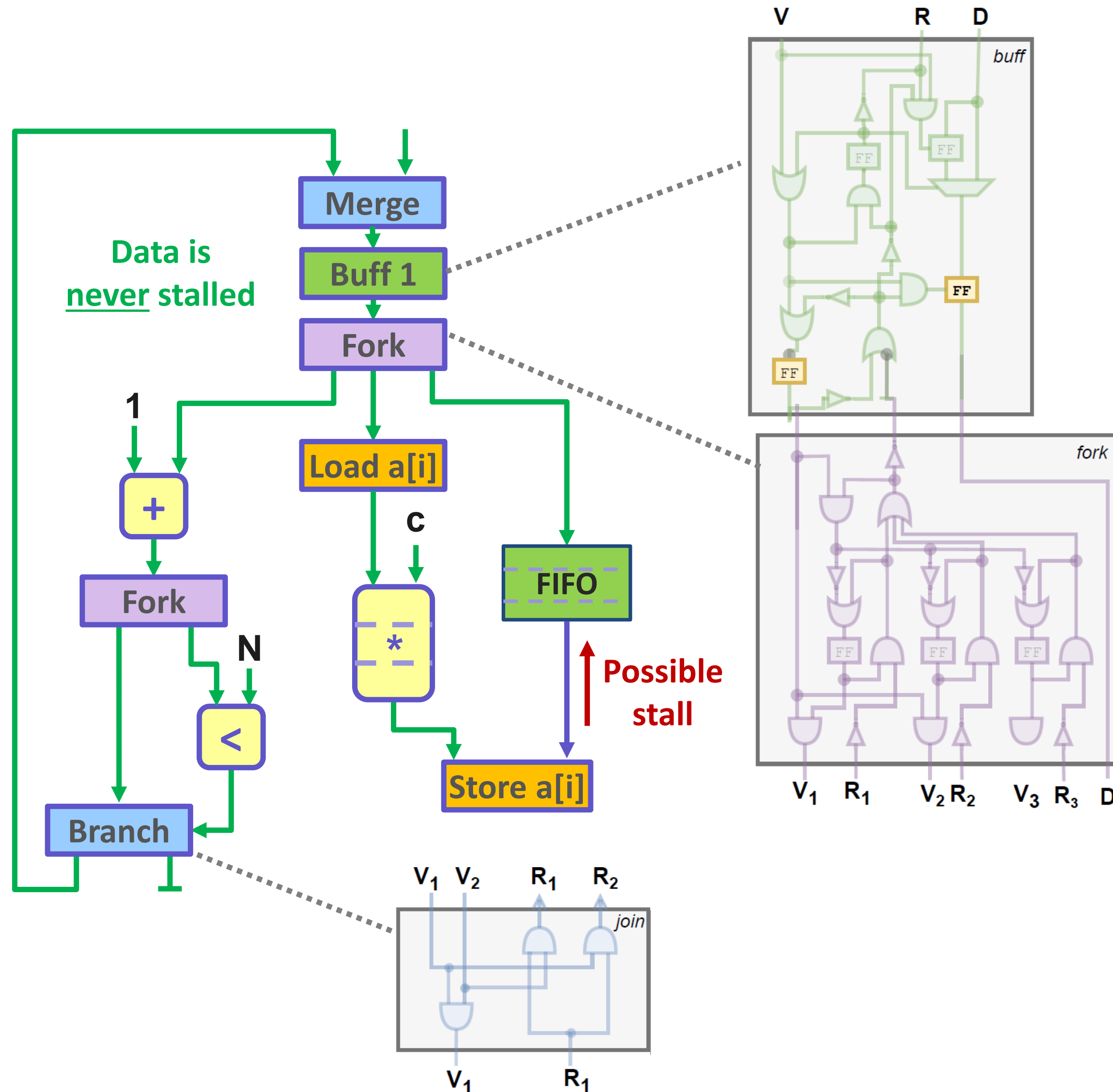
Distributed dataflow handshake mechanism: resource and frequency overhead

The Cost of Dataflow Computation

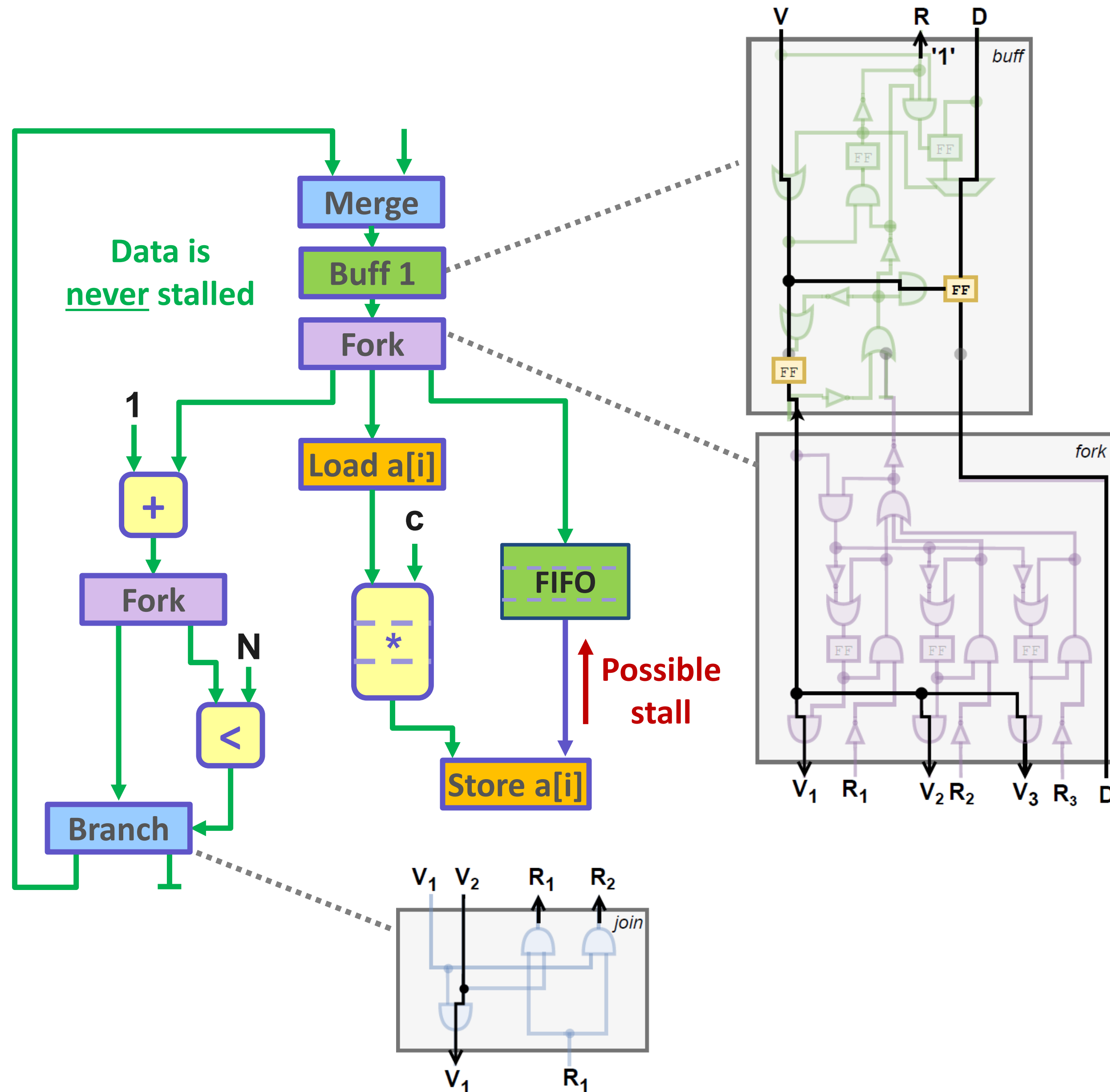


Do we need expensive dataflow logic *everywhere*?

Removing Excessive Dynamism

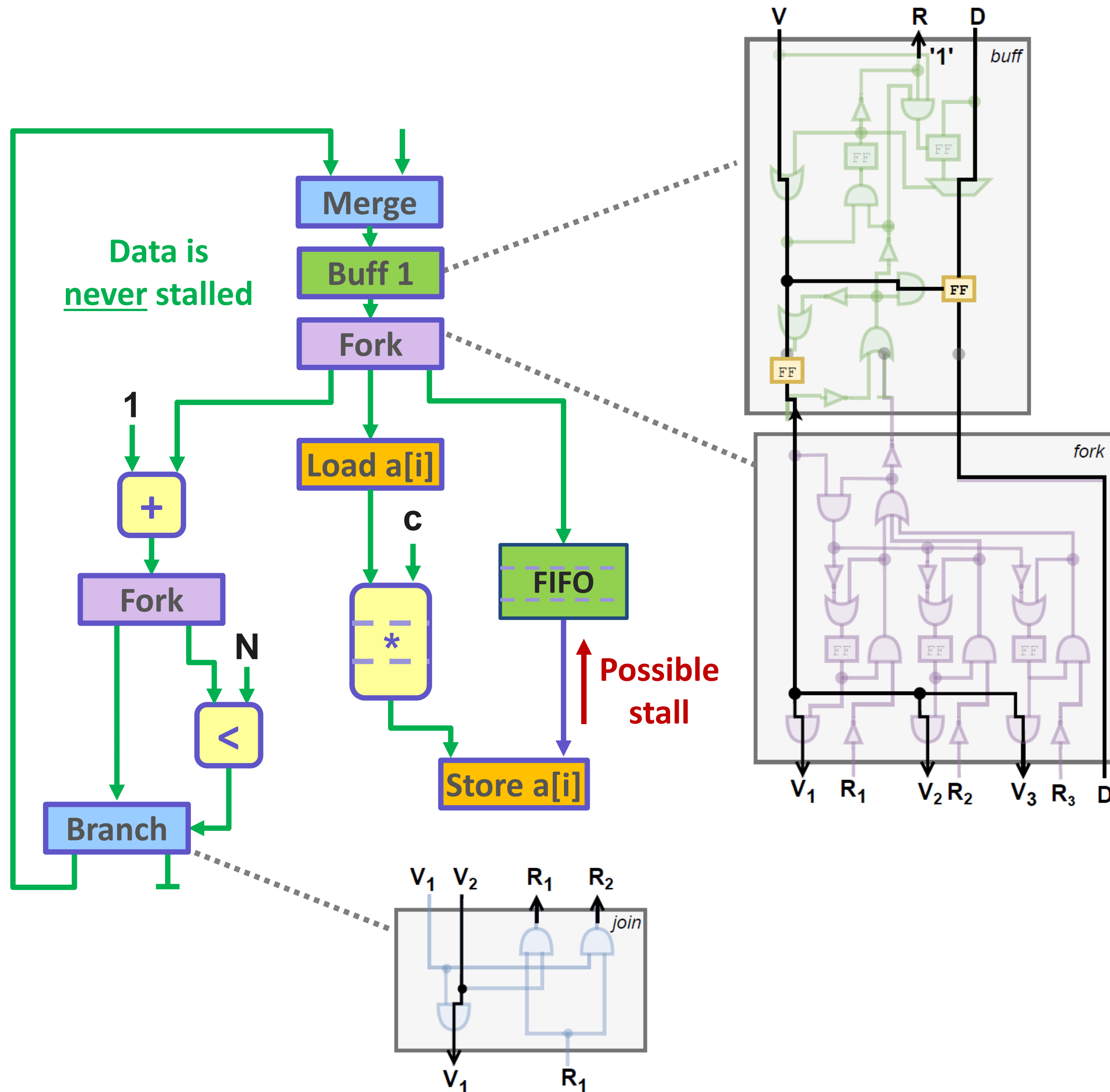


Removing Excessive Dynamism



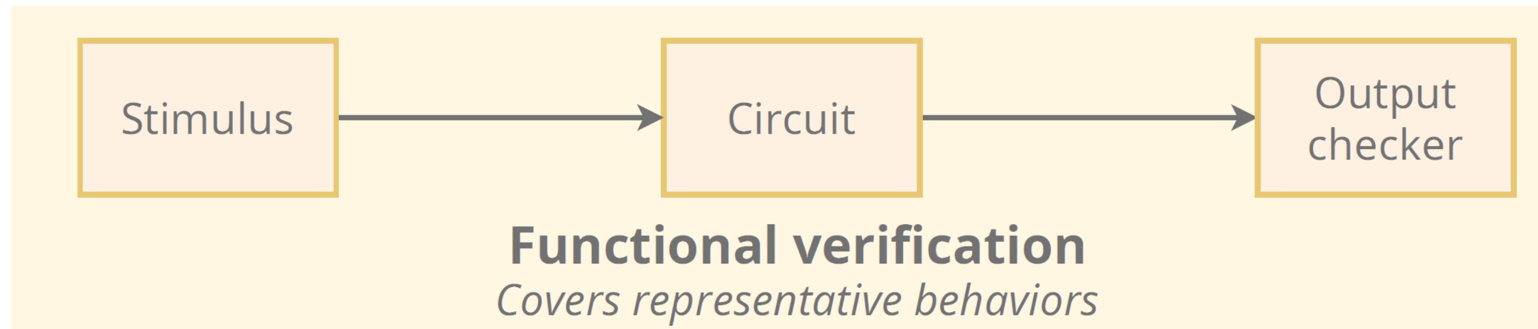
Restrict the generality of dataflow logic whenever it is not needed

Removing Excessive Dynamism

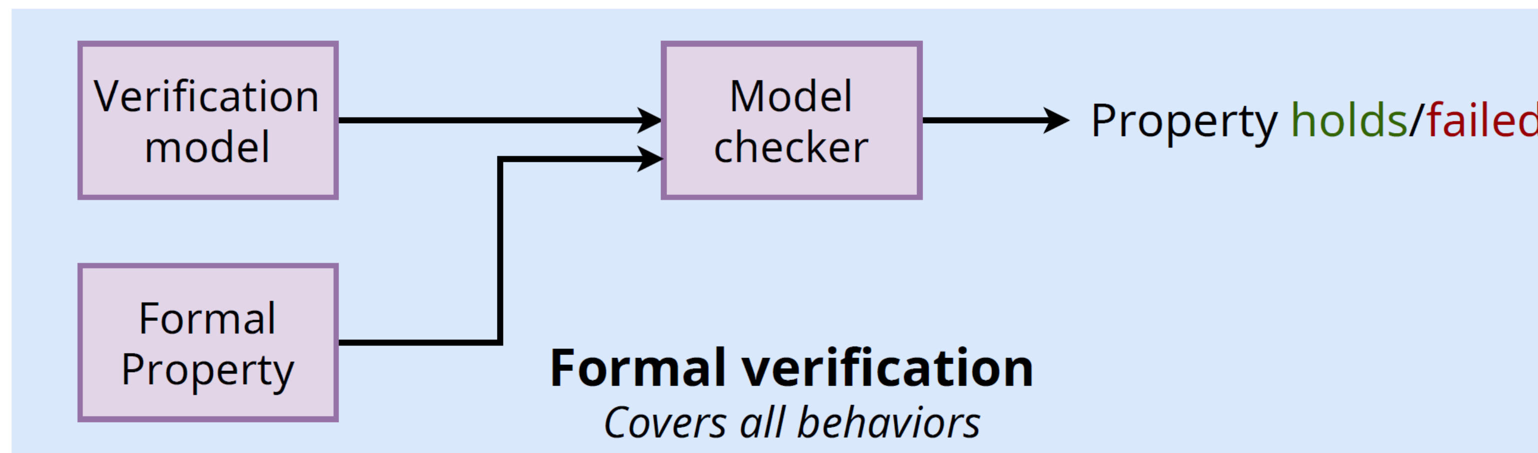


How to guarantee correctness of simplifications for *any possible* circuit behavior?

How to Guarantee Correctness?



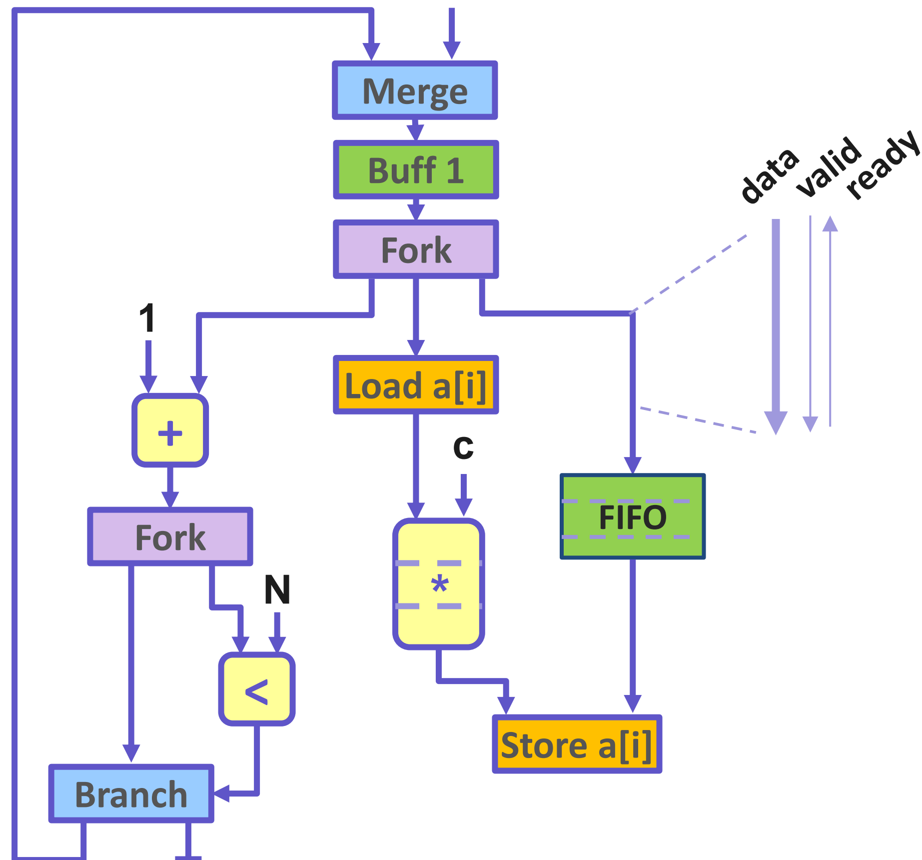
Functional verification is inefficient and non-exhaustive



Our goal: a formal verification framework for reducing the hardware complexity of dataflow circuits

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

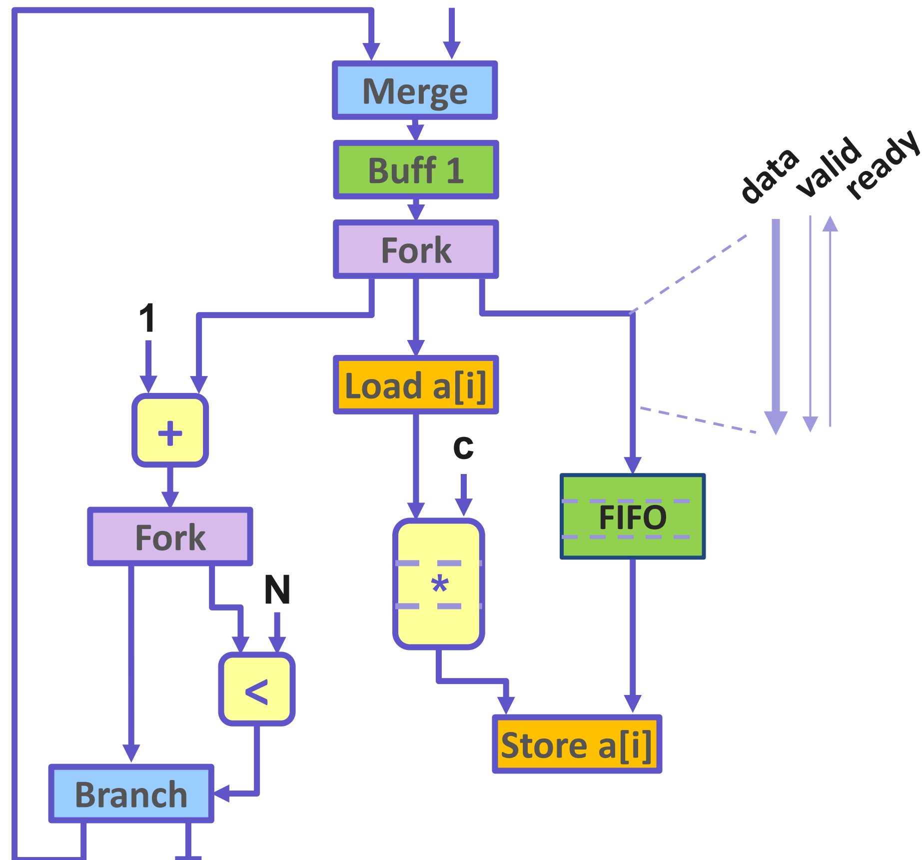
Proving Properties to Eliminate Excessive Dynamism



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

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

Proving Properties to Eliminate Excessive Dynamism

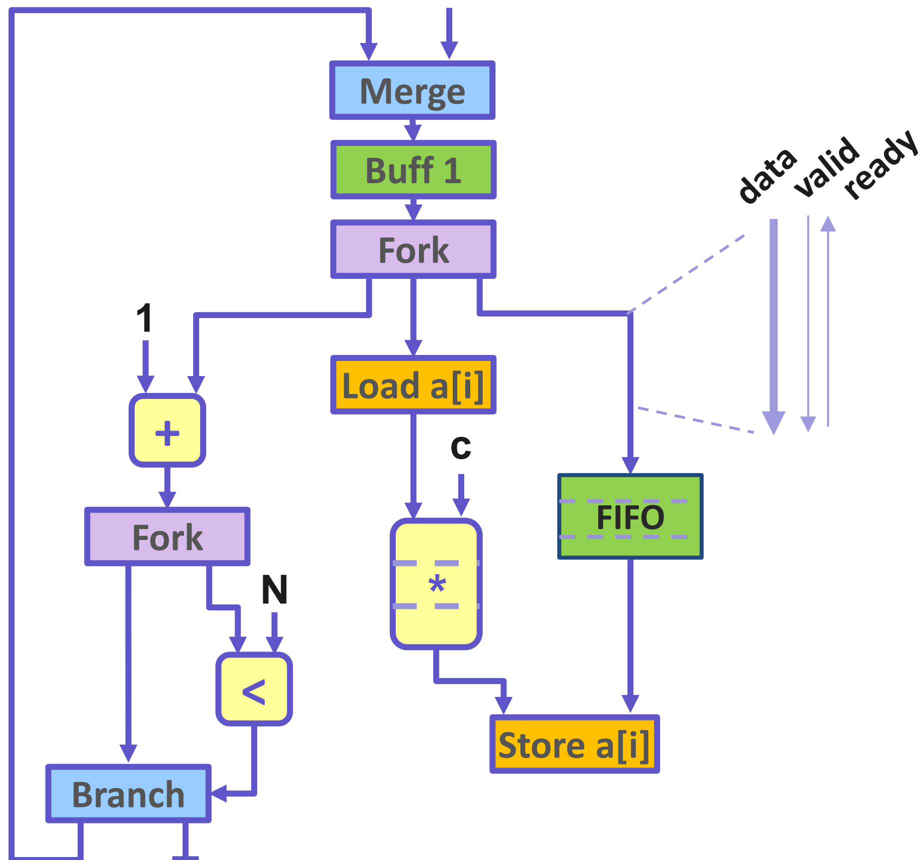


For each channel: prove the **absence of backpressure**
(remove logic to compute the ready signal)
AG (valid → ready)

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)

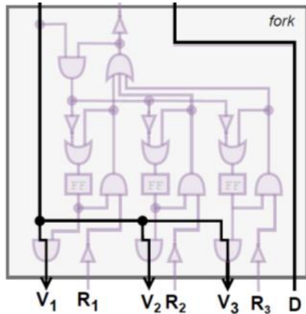
Up to 50% area reduction without a performance penalty

But it is very slow (~hrs)…

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

Reducing the Cost of Dataflow Circuits

How to **eliminate excessive dynamism?**



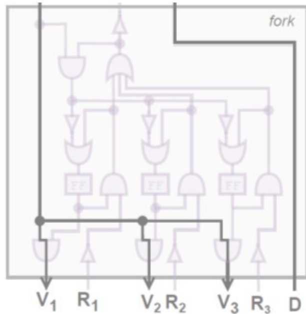
Formal verification for redundant handshake logic removal

→ **50% resource reduction**

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

Reducing the Cost of Dataflow Circuits

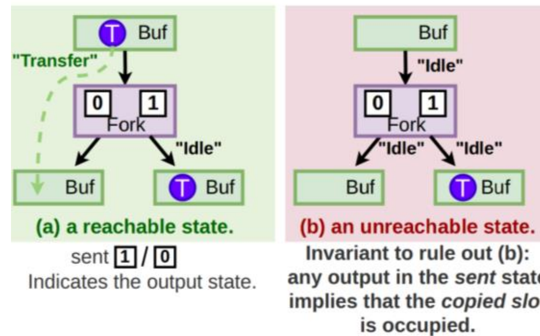
How to **eliminate excessive dynamism?**



Formal verification for redundant handshake logic removal

→ **50% resource reduction**

How to make dynamism removal **more scalable?**



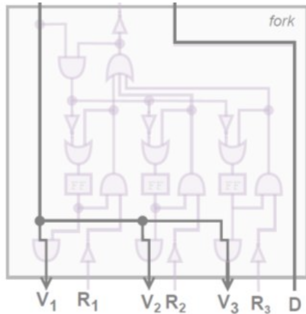
Inductive invariants for fast & scalable verification

→ **from days to minutes**

Xu and Josipović. Automatic inductive invariant generation for scalable dataflow circuit verification. ICCAD'23.

Reducing the Cost of Dataflow Circuits

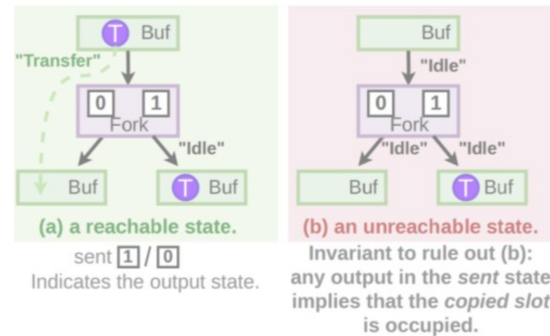
How to **eliminate excessive dynamism?**



Formal verification for redundant handshake logic removal

→ **50% resource reduction**

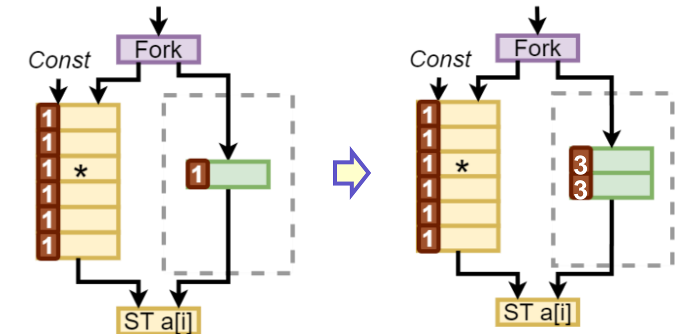
How to make dynamism removal **more scalable?**



Inductive invariants for fast & scalable verification

→ **from days to minutes**

How to make dynamism removal **more effective?**



Latency and occupancy balancing for suppressing spurious dynamism

→ **same resources as static HLS**

Same resources as static HLS, all performance benefits of dynamic scheduling maintained

Xu and Josipović. Suppressing Spurious Dynamism of Dataflow Circuits via Latency and Occupancy Balancing. FPGA'24.

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! ☺

MSc Course in Spring 2025: 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 (2025)

Hope to see you there! ☺

DYNAMO: Digital Systems and Design Automation Group

dynamo.ethz.ch
ljosipovic@ethz.ch

Project list 2025

Thanks! ☺