Efficient Design Space Exploration for Dynamic & Speculative High-Level Synthesis

Dylan Leothaud, Jean-Michel Gorius, Simon Rokicki, Steven Derrien

Taran Team, Univ Rennes, IRISA, Inria

FPL’24

High-Level Synthesis

High-Level Synthesis (HLS) tools generate HDL from C++ description

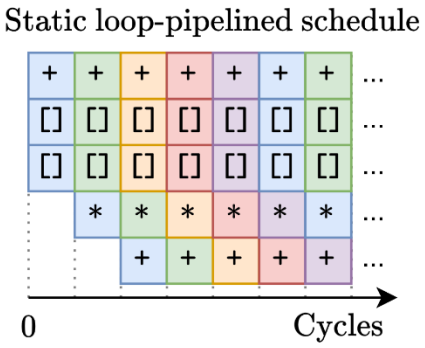
```
int vec_mul(int *v1, int *v2,
        int out, int n) {
  for (int i = 0; i < n; ++i)
    out += v1[i] * v2[i];
  return out;
}
```

High-Level Synthesis

High-Level Synthesis (HLS) tools generate HDL from C++ description

```
int vec_mul(int *v1, int *v2,
        int out, int n) {
  for (int i = 0; i < n; ++i)
    out += v1[i] * v2[i];
  return out;
}
```

HLS

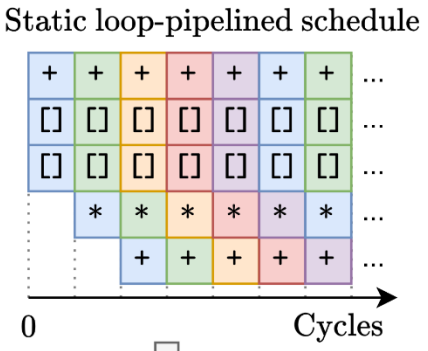


High-Level Synthesis

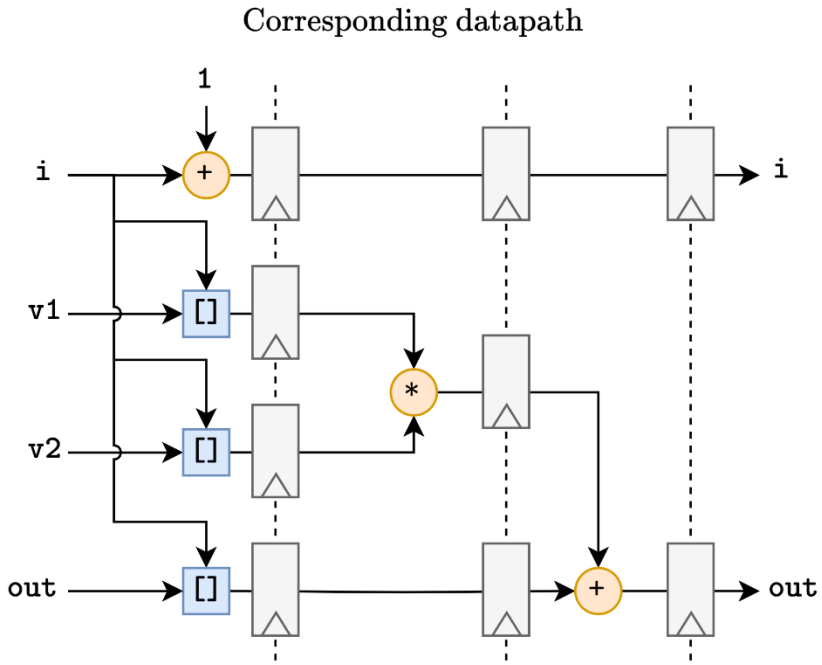
High-Level Synthesis (HLS) tools generate HDL from C++ description

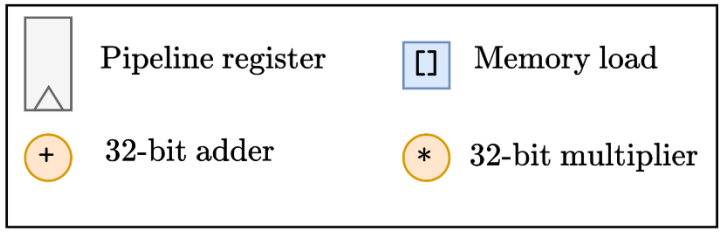
```
int vec_mul(int *v1, int *v2,
            int out, int n) {
  for (int i = 0; i < n; ++i)
    out += v1[i] * v2[i];
  return out;
}
```

HLS



HW synthesis

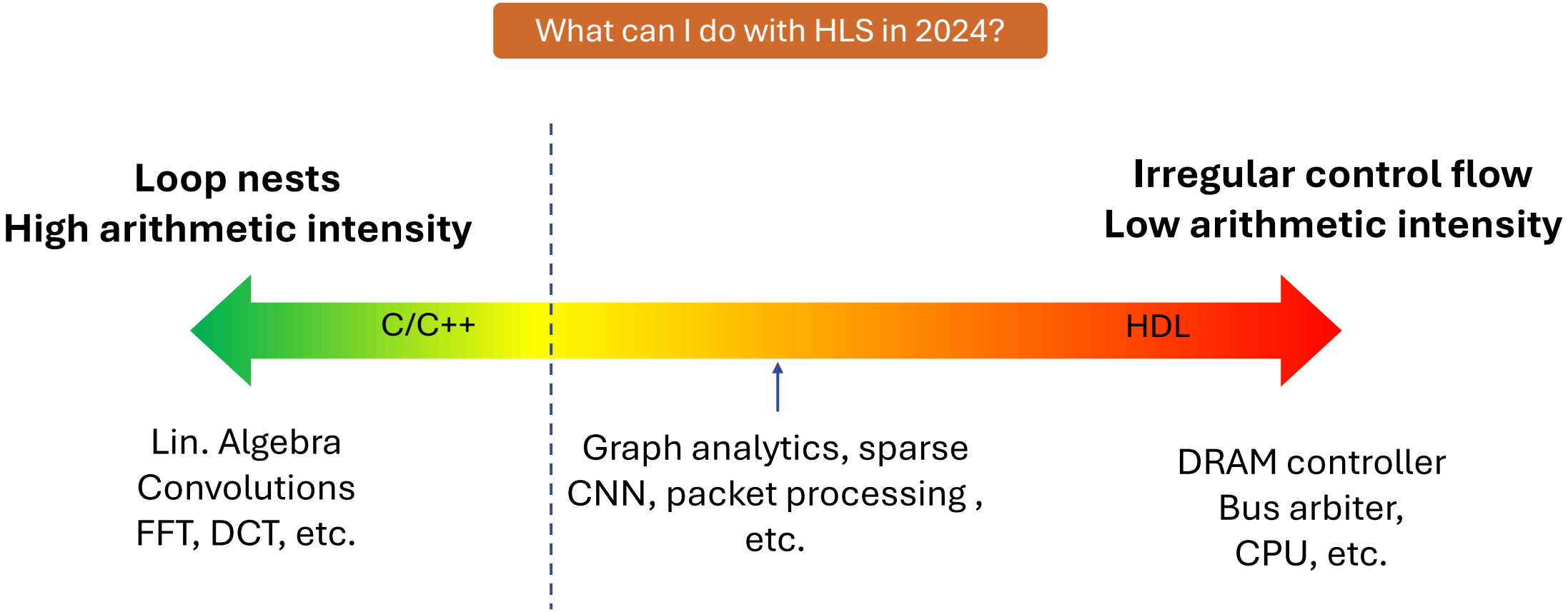




HLS automatically generates pipelined hardware

FPGA needs deep pipelining to be efficient

What can I use HLS for?

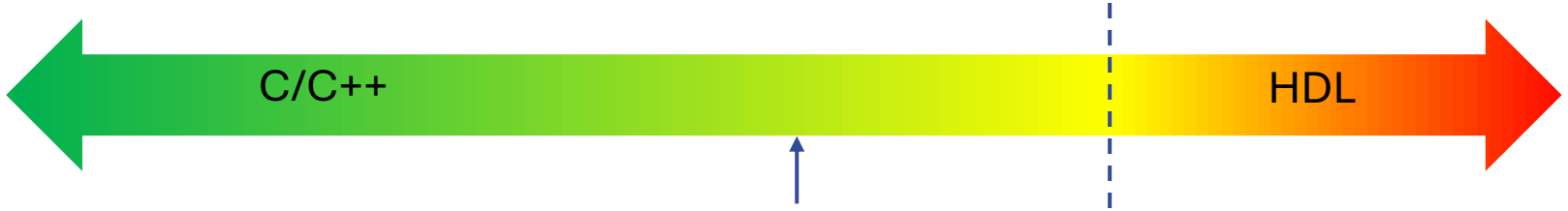


What can I use HLS for?

What can I do with HLS in 2024?

Loop nests
High arithmetic intensity

Irregular control flow
Low arithmetic intensity



Lin. Algebra
Convolutions
FFT, DCT, etc.

Graph analytics, sparse
CNN, packet processing ,
etc.

DRAM controller
Bus arbiter,
CPU, etc.

HLS tools struggle at efficiently scheduling kernels with irregular control-flow

Limits of standard HLS

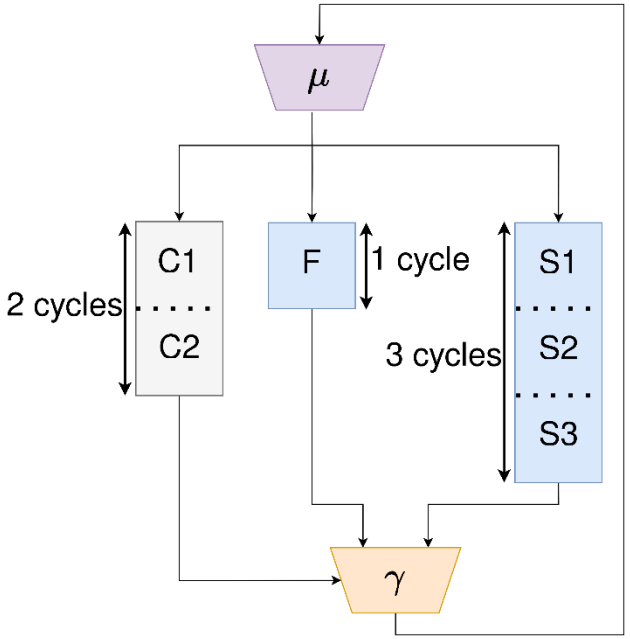
SOTA commercial HLS tools rely on static scheduling

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```

Limits of standard HLS

SOTA commercial HLS tools rely on static scheduling

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



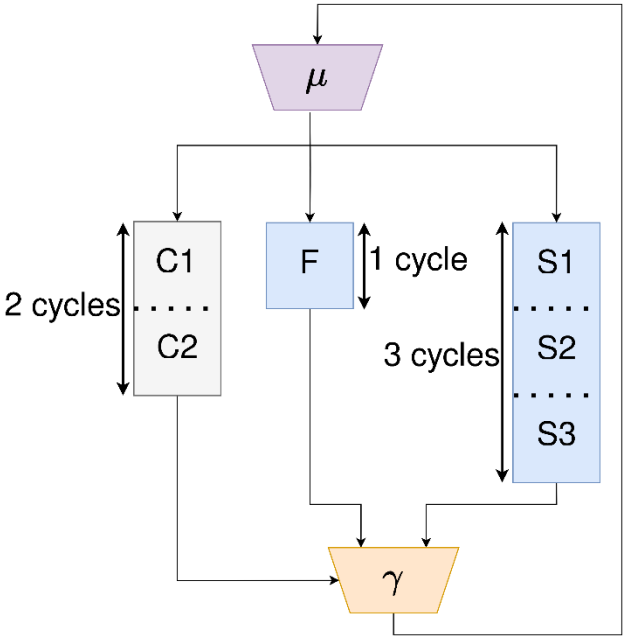
Limits of standard HLS

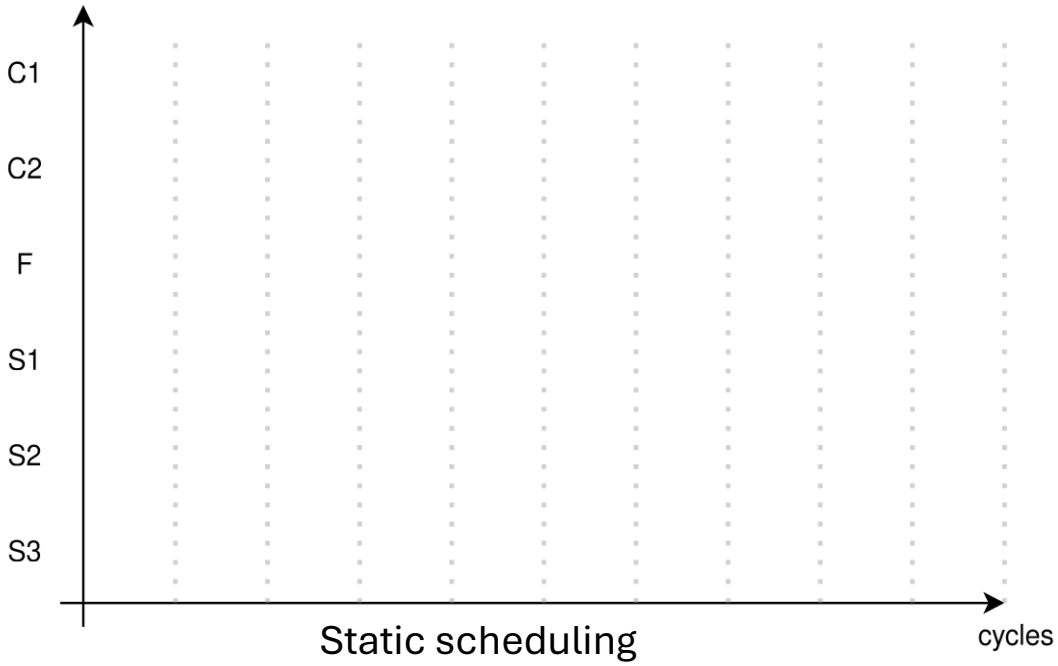
SOTA commercial HLS tools rely on static scheduling

Data dependency

Control dependency

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```





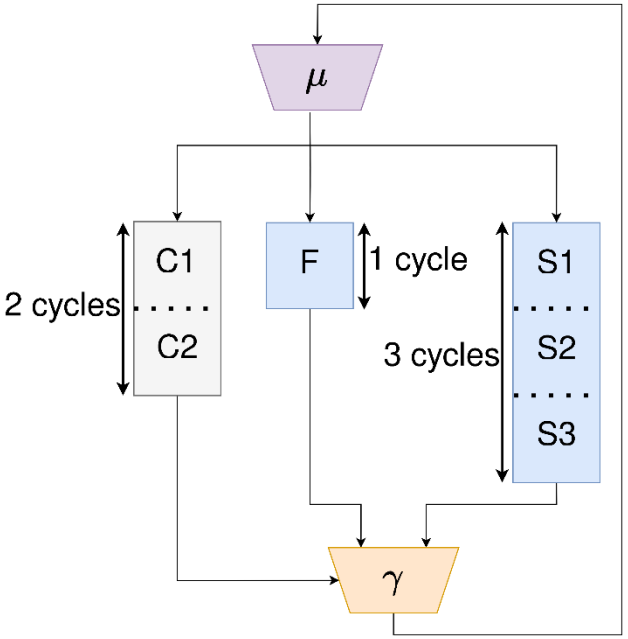
Limits of standard HLS

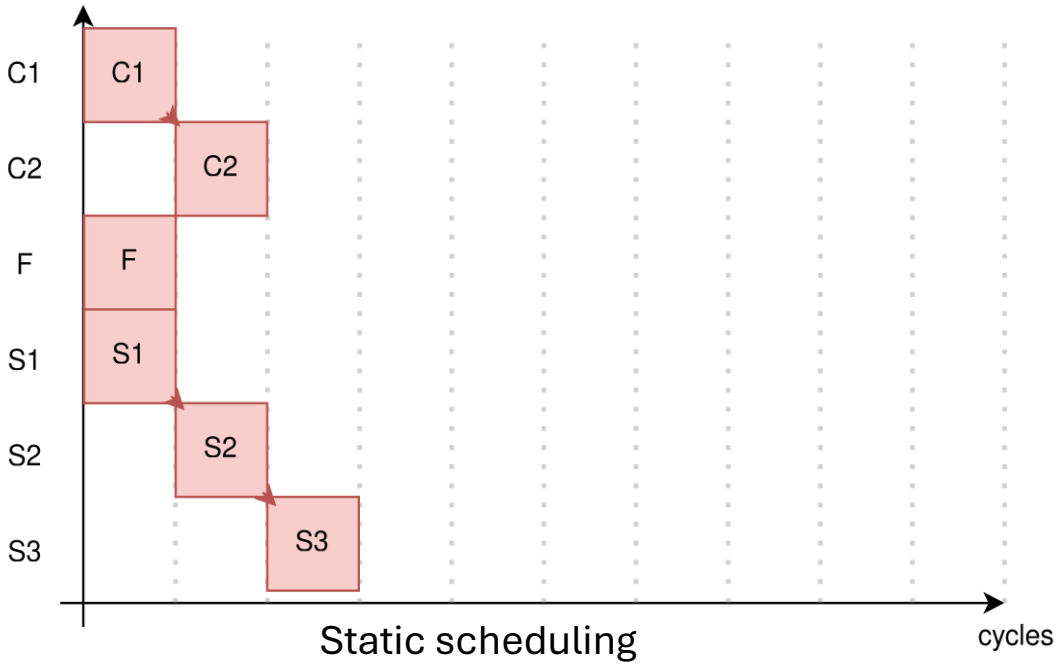
SOTA commercial HLS tools rely on static scheduling

Data dependency

Control dependency

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```





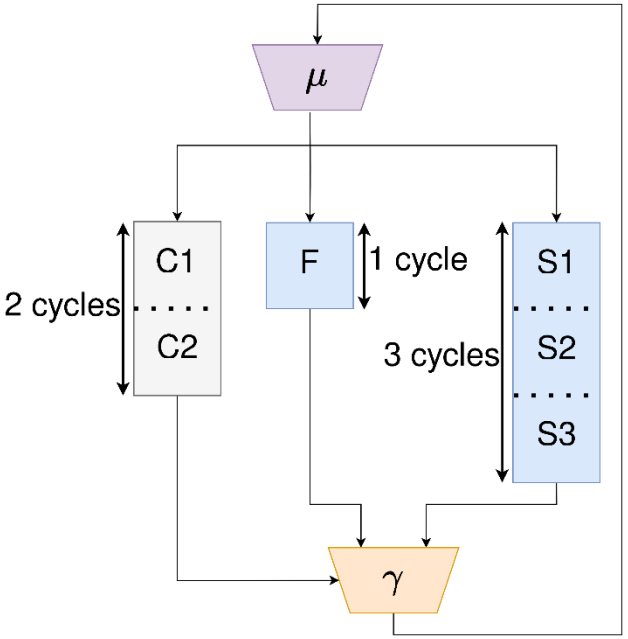
Limits of standard HLS

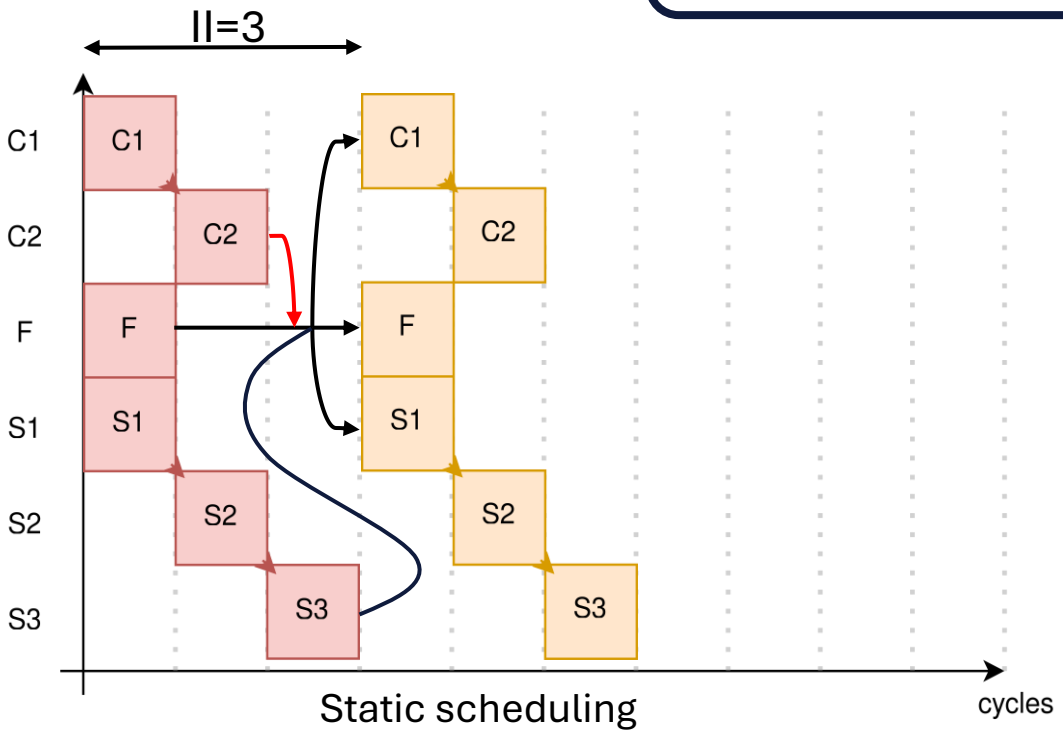
SOTA commercial HLS tools rely on static scheduling

Data dependency

Control dependency

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```

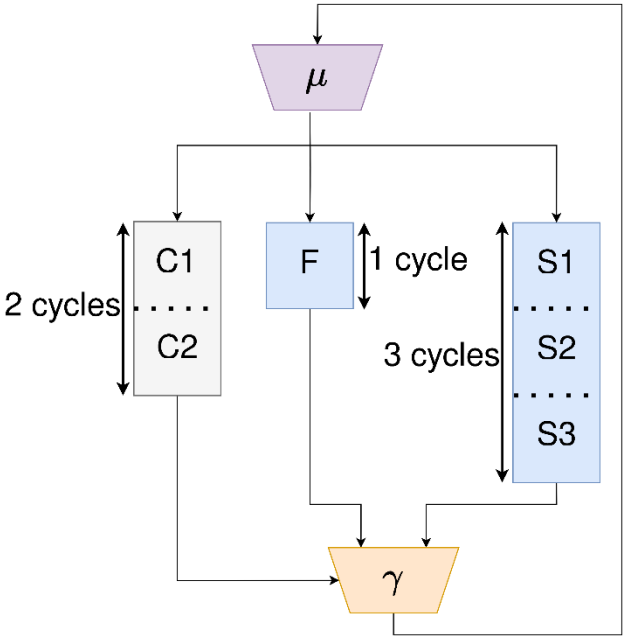


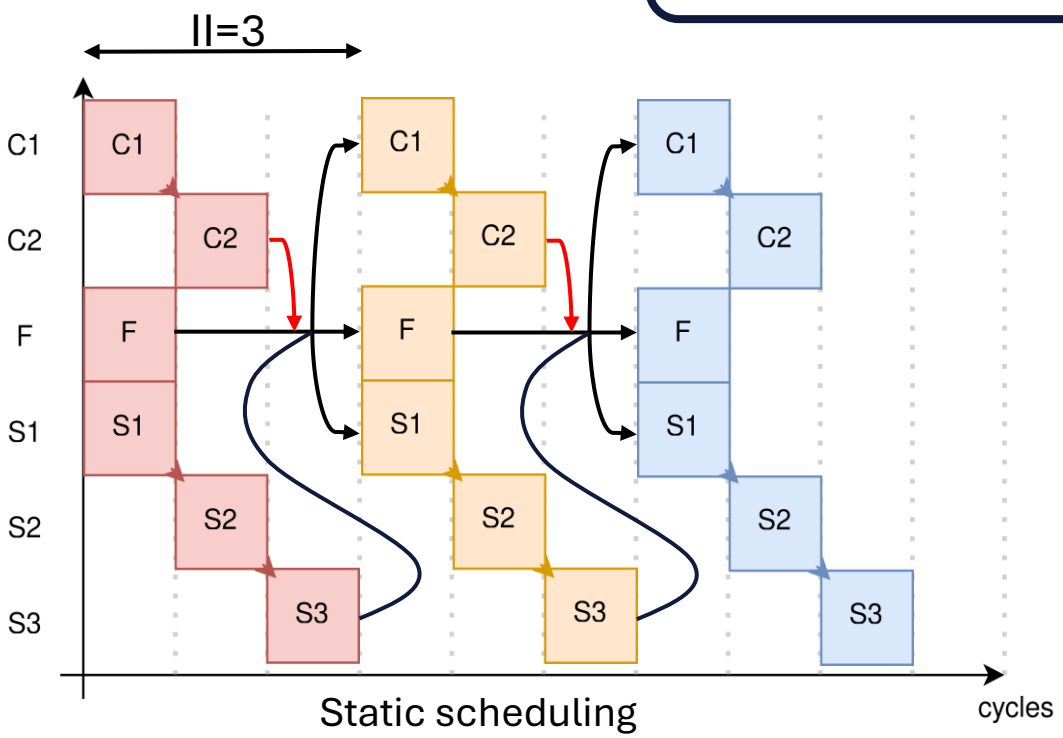


Limits of standard HLS

SOTA commercial HLS tools rely on static scheduling

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```





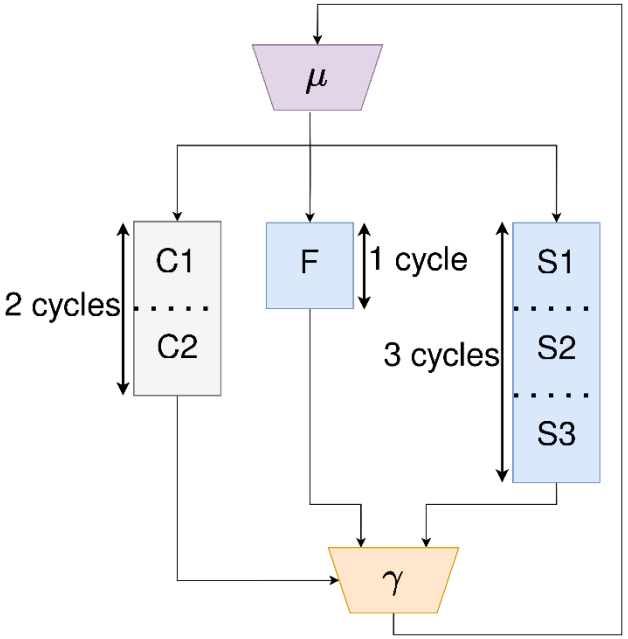
Limits of standard HLS

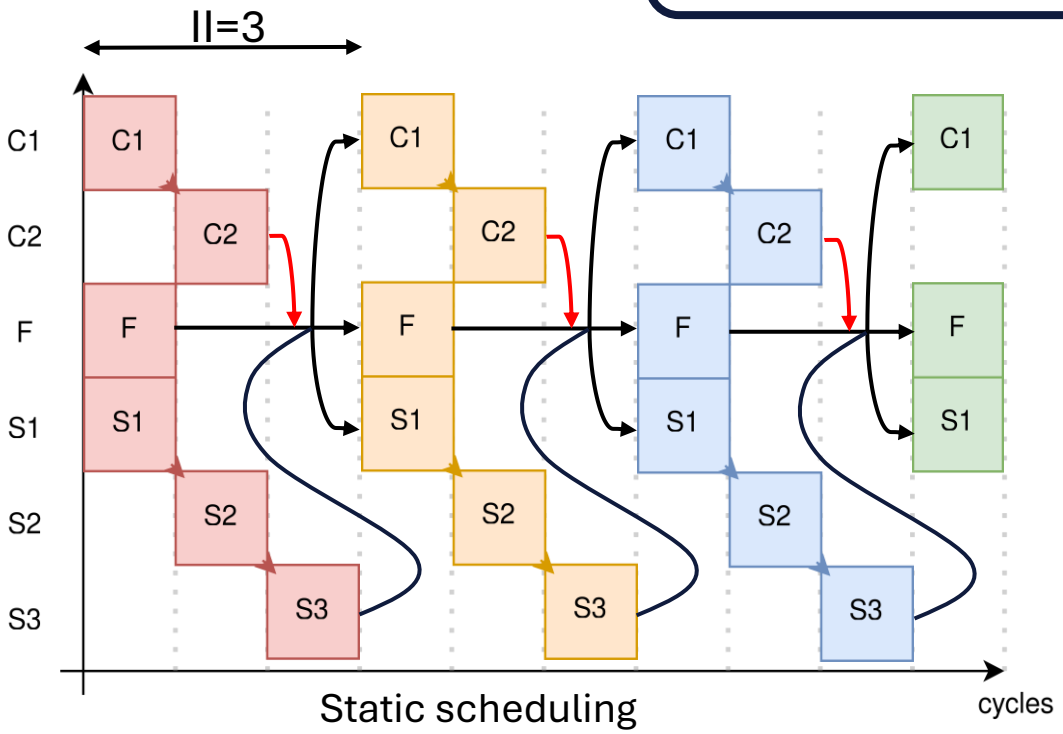
SOTA commercial HLS tools rely on static scheduling

Data dependency

Control dependency

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```

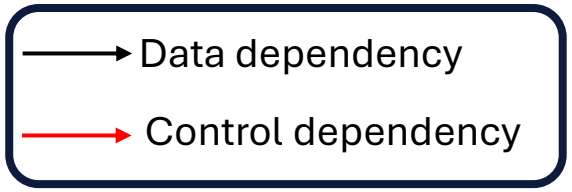




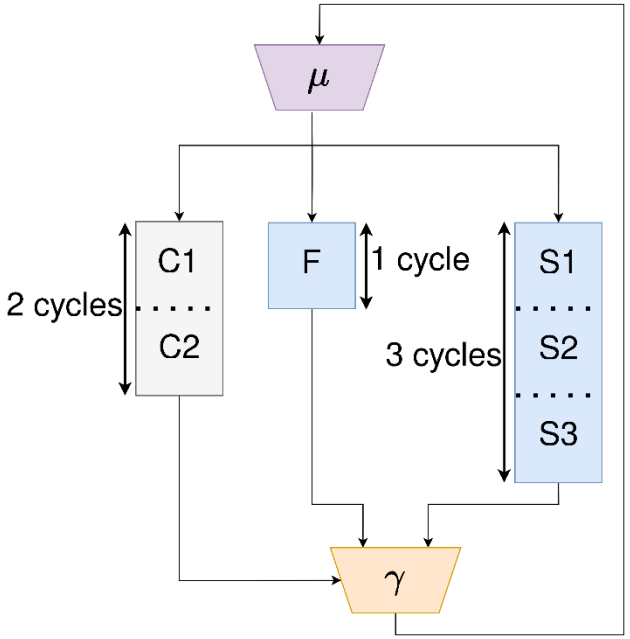
Static scheduling is pessimistic: based on worst case behavior

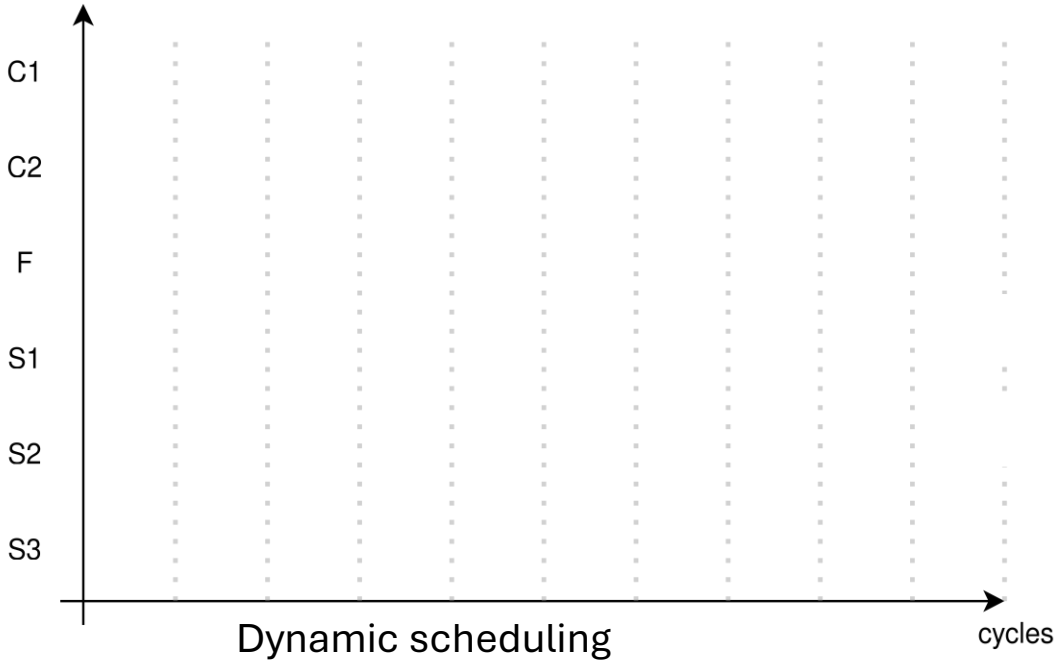
Going beyond static scheduling

Dynamic scheduling [1] adapts execution at runtime to prevent unnecessary stalls




```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```

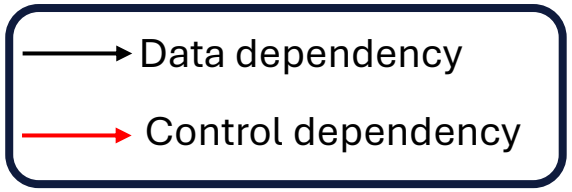




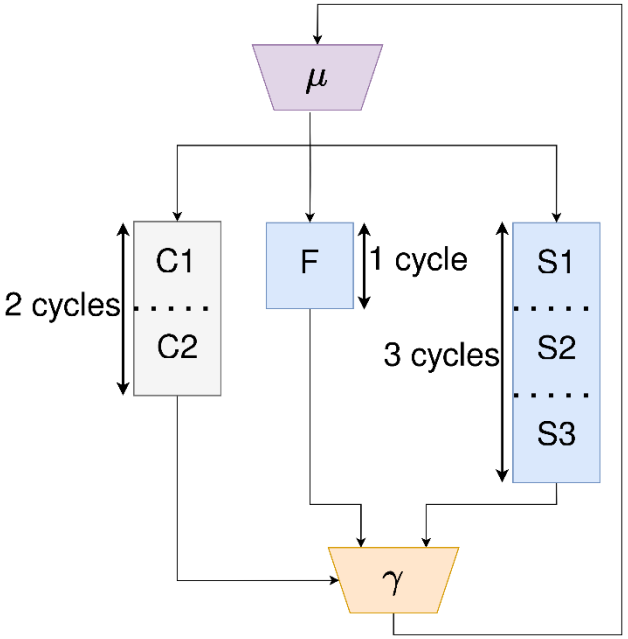
[1] L.Josipović, R.Ghosal, P. Ienne. (2018). Dynamically Scheduled High-level Synthesis. FPGA'18.

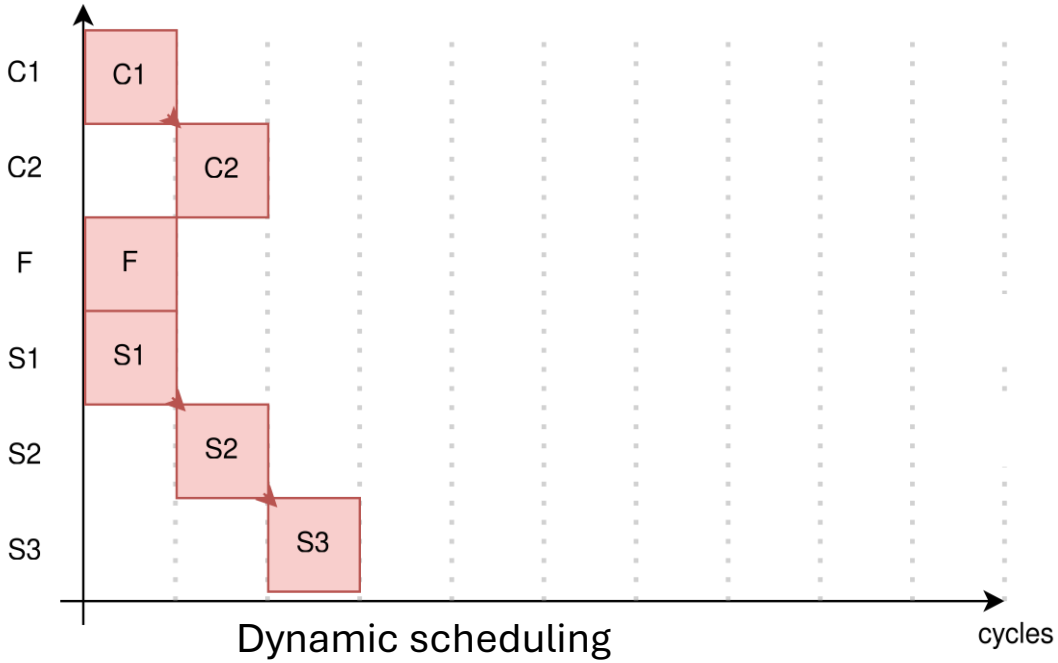
Going beyond static scheduling

Dynamic scheduling [1] adapts execution at runtime to prevent unnecessary stalls




```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```

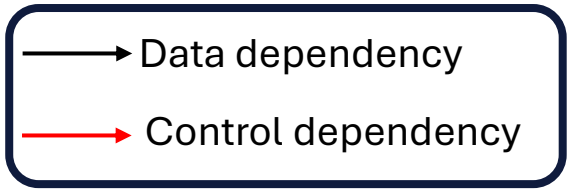




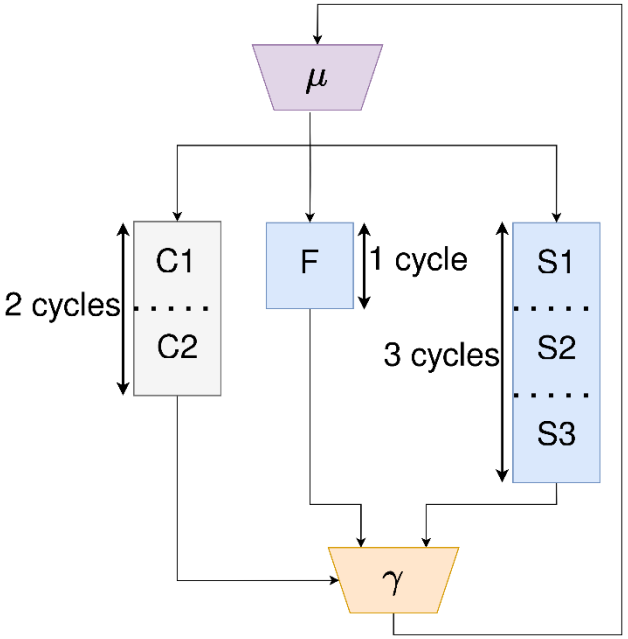
[1] L.Josipović, R.Ghosal, P. Ienne. (2018). Dynamically Scheduled High-level Synthesis. FPGA'18.

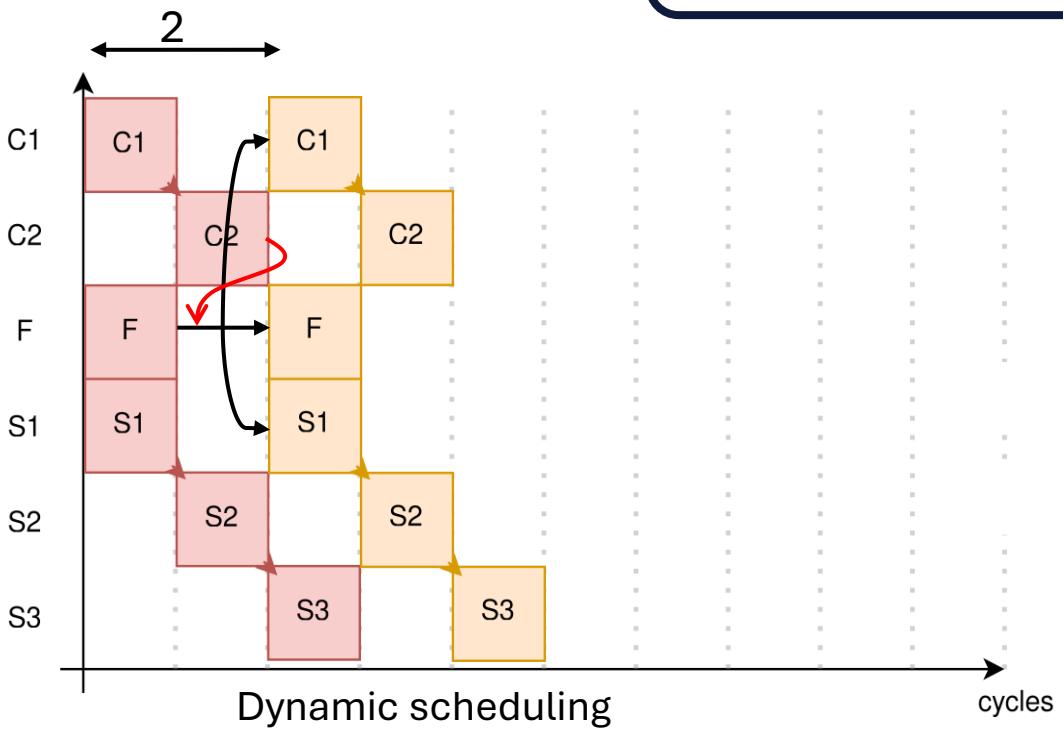
Going beyond static scheduling

Dynamic scheduling [1] adapts execution at runtime to prevent unnecessary stalls




```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```

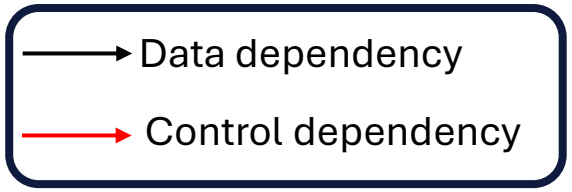




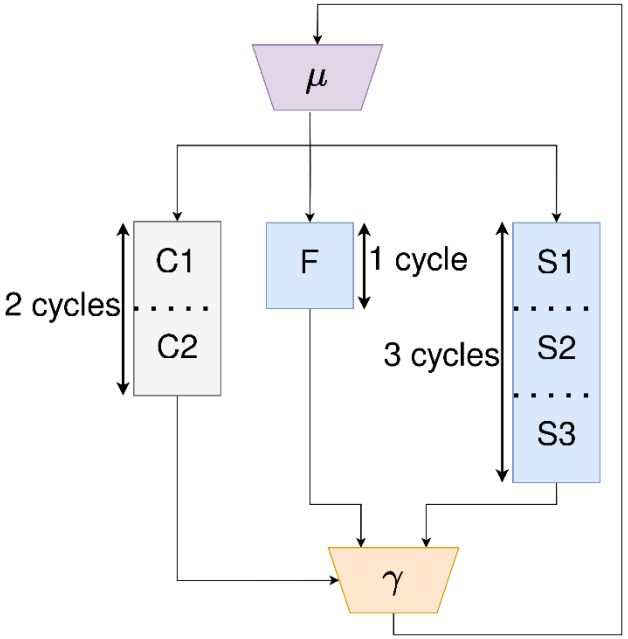
[1] L.Josipović, R.Ghosal, P. Ienne. (2018). Dynamically Scheduled High-level Synthesis. FPGA'18.

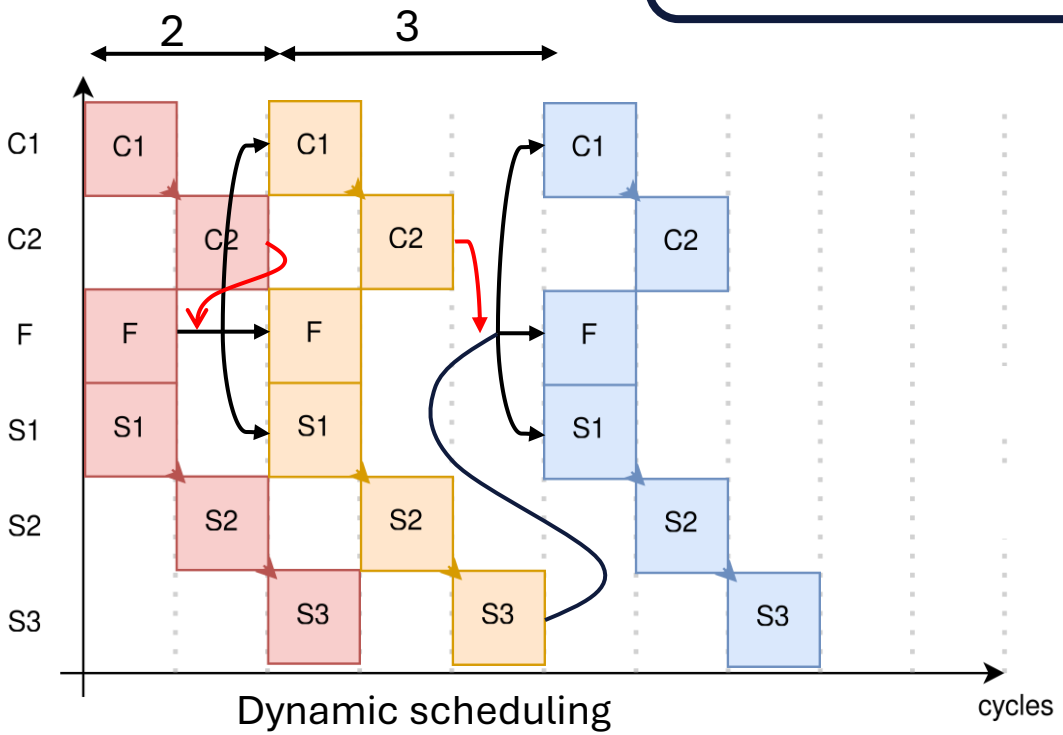
Going beyond static scheduling

Dynamic scheduling [1] adapts execution at runtime to prevent unnecessary stalls




```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```

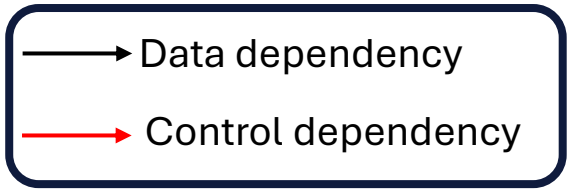




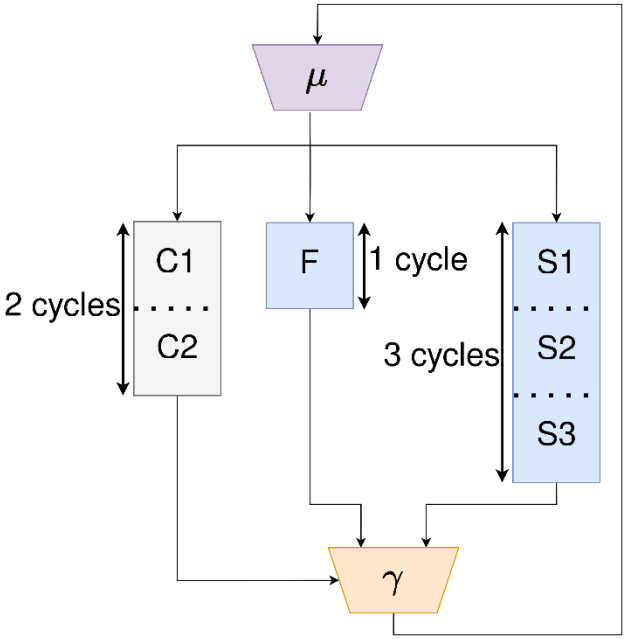
[1] L.Josipović, R.Ghosal, P. Ienne. (2018). Dynamically Scheduled High-level Synthesis. FPGA'18.

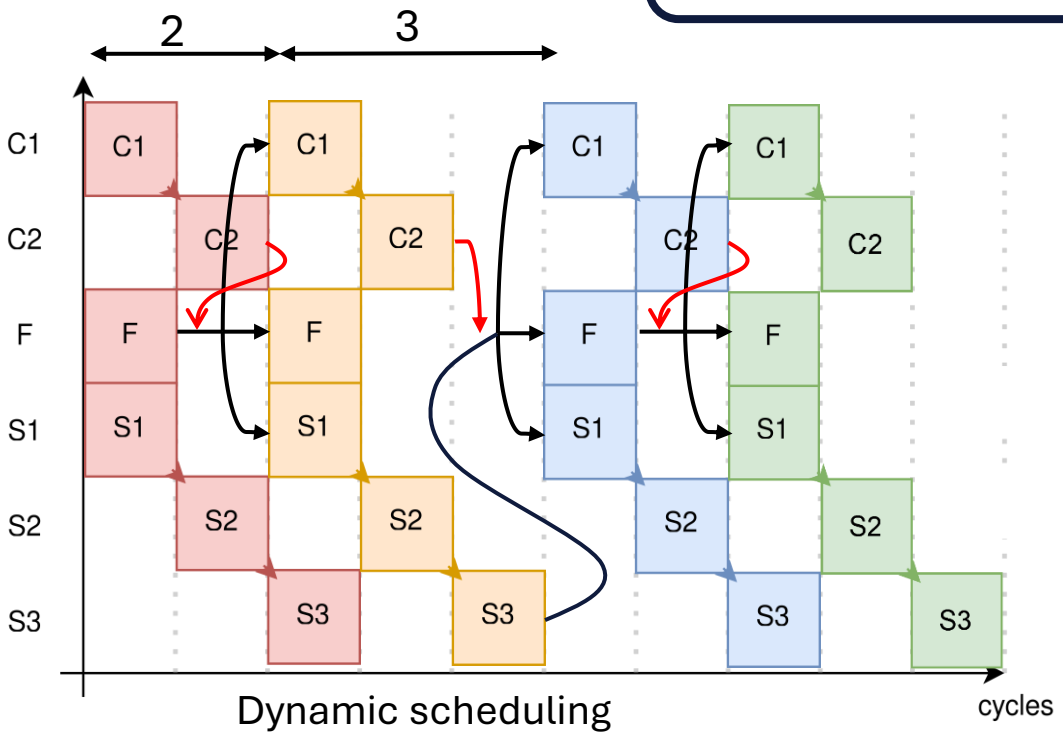
Going beyond static scheduling

Dynamic scheduling [1] adapts execution at runtime to prevent unnecessary stalls




```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```

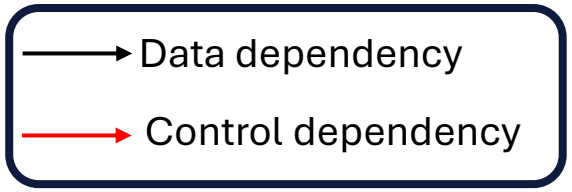




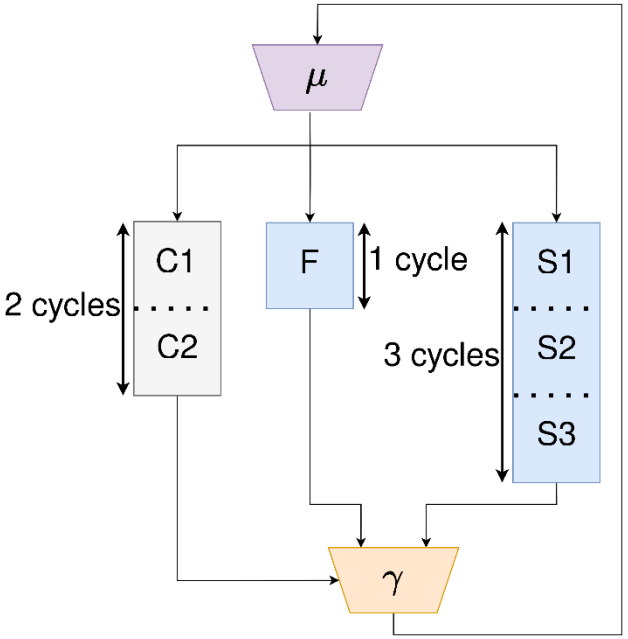
[1] L.Josipović, R.Ghosal, P. Ienne. (2018). Dynamically Scheduled High-level Synthesis. FPGA'18.

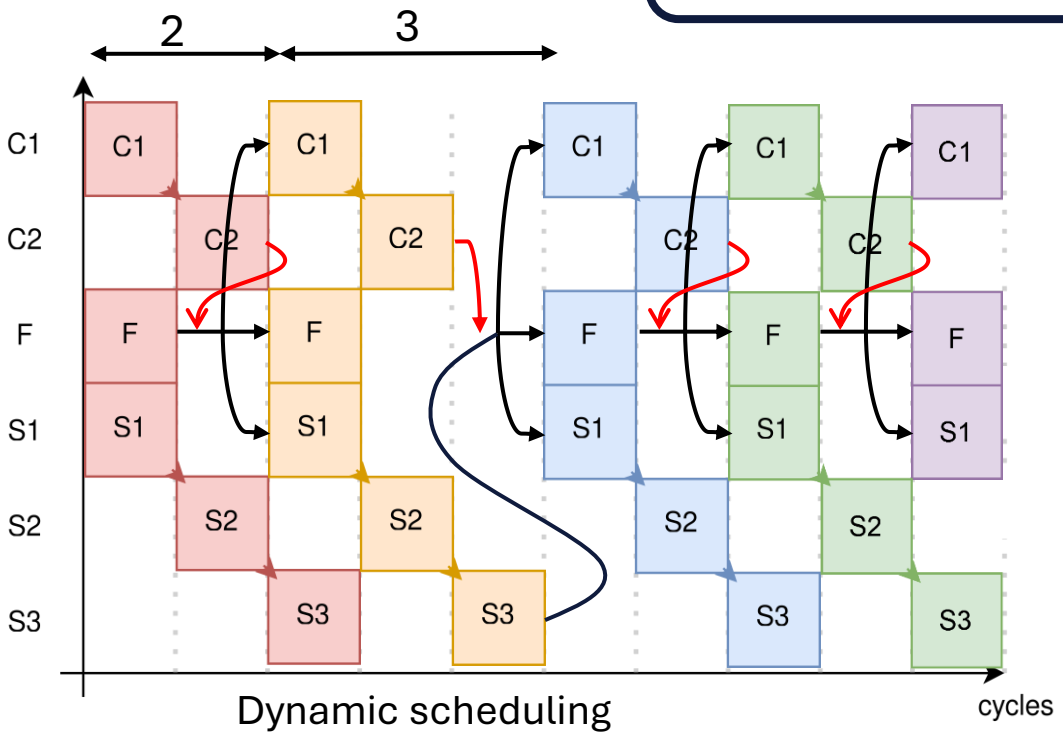
Going beyond static scheduling

Dynamic scheduling [1] adapts execution at runtime to prevent unnecessary stalls




```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



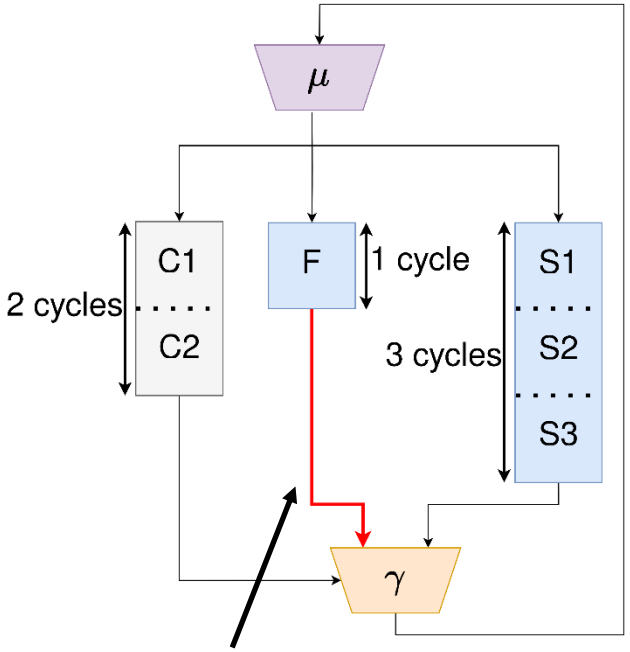


[1] L.Josipović, R.Ghosal, P. Ienne. (2018). Dynamically Scheduled High-level Synthesis. FPGA'18.

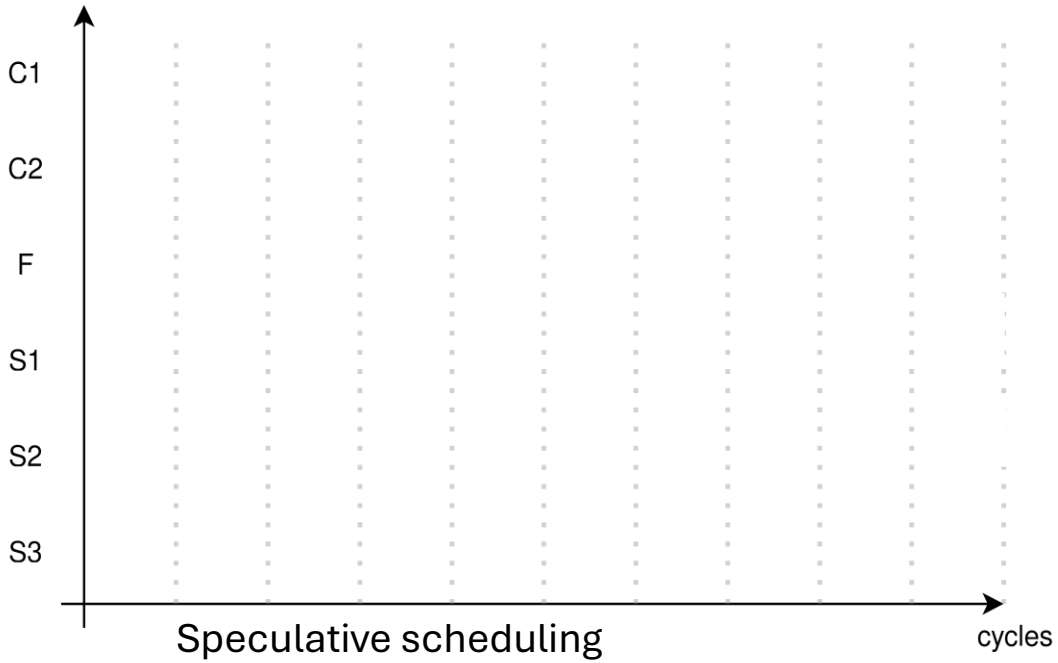
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input



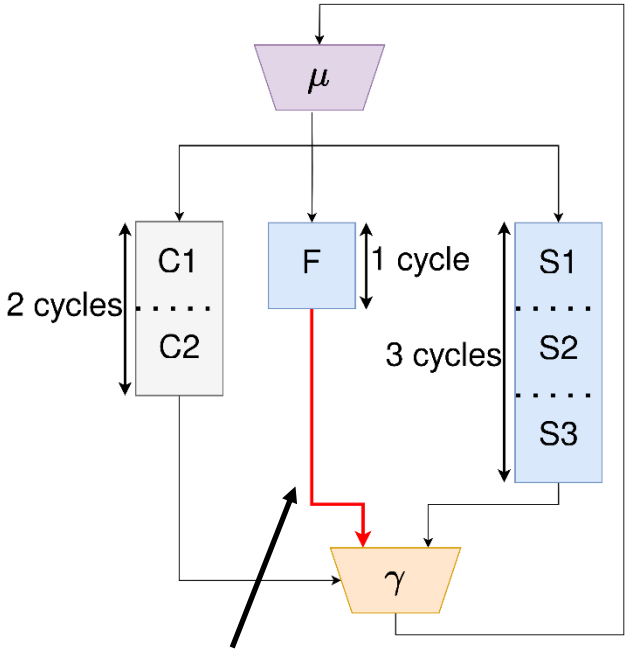
Speculative scheduling

[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

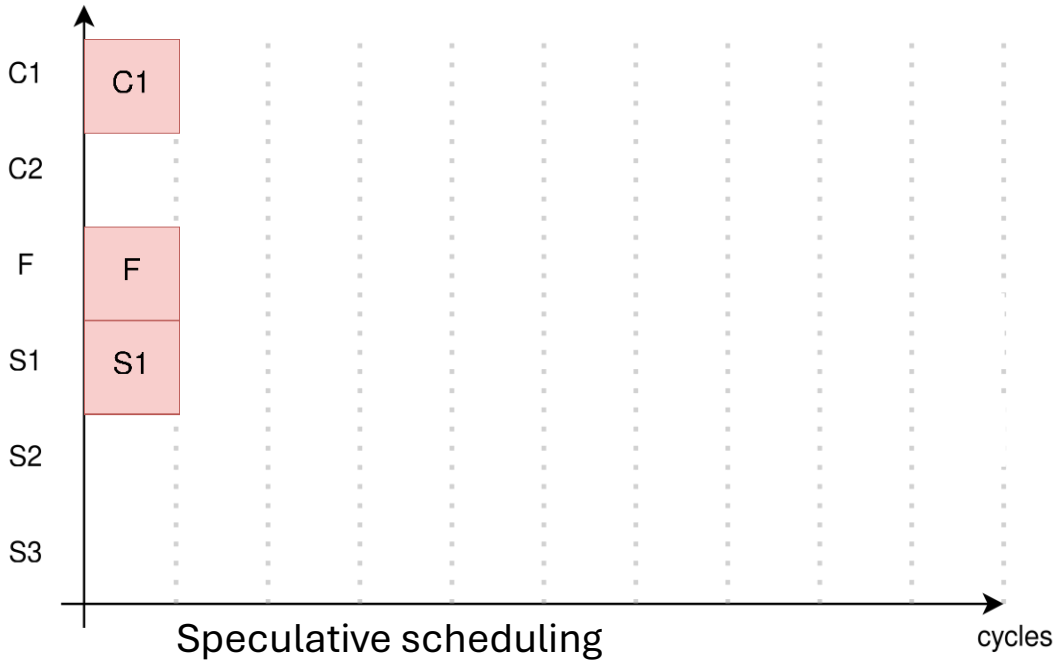
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input

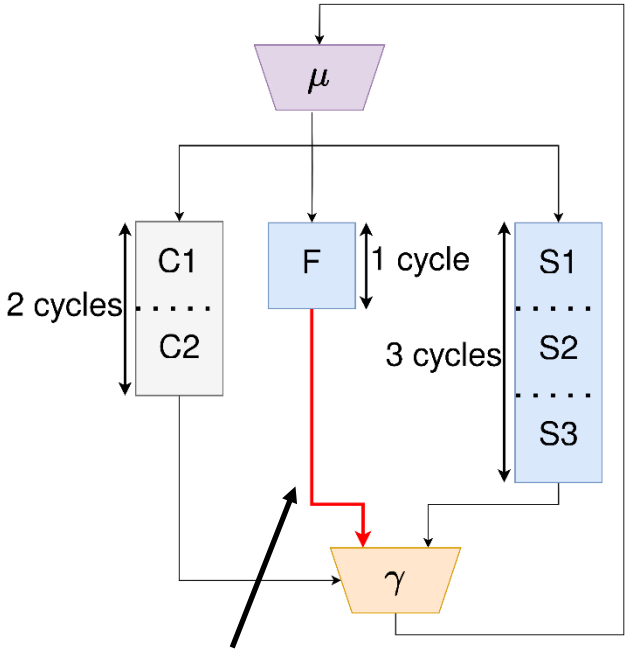


[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

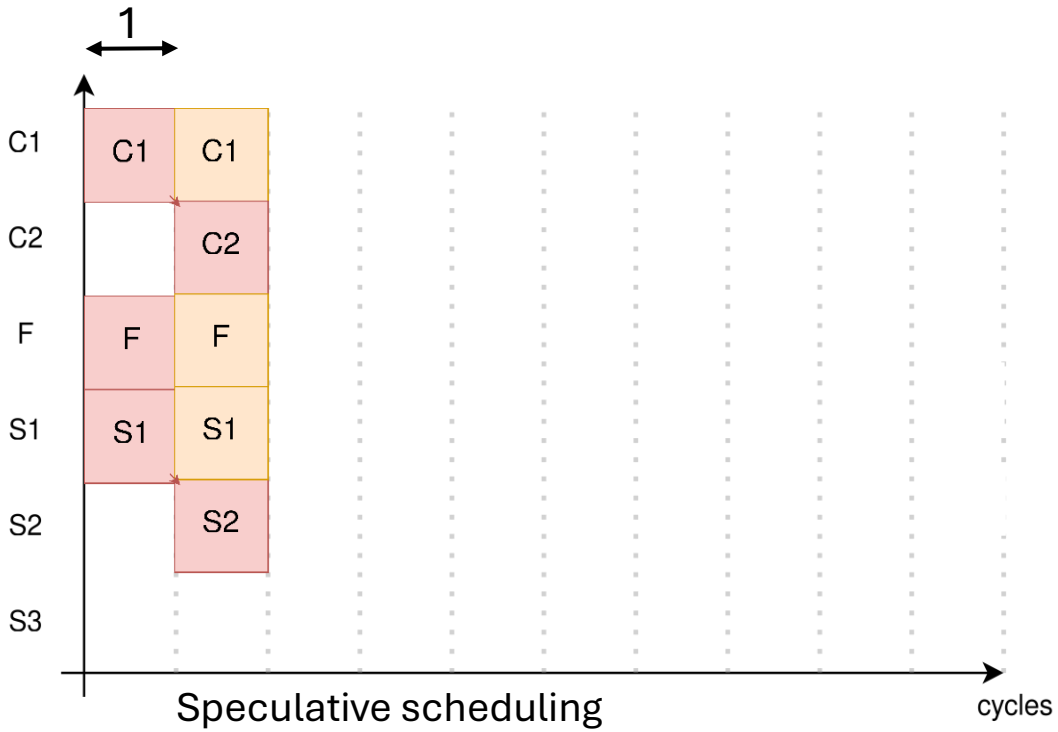
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input

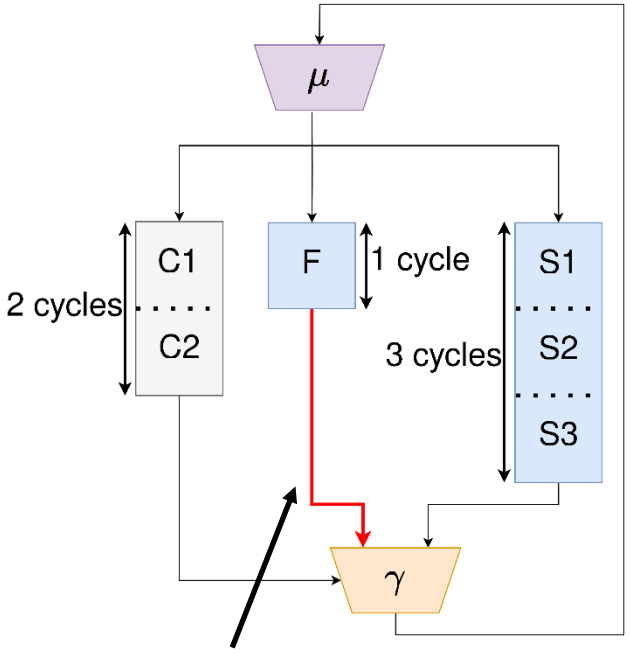


[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

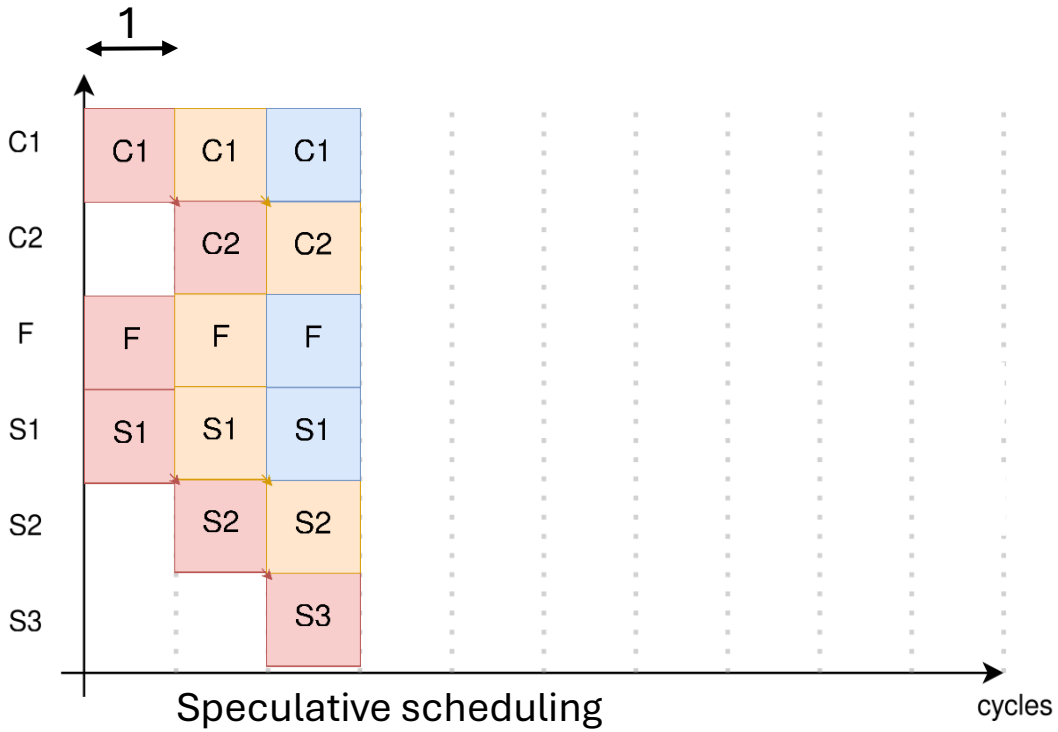
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input

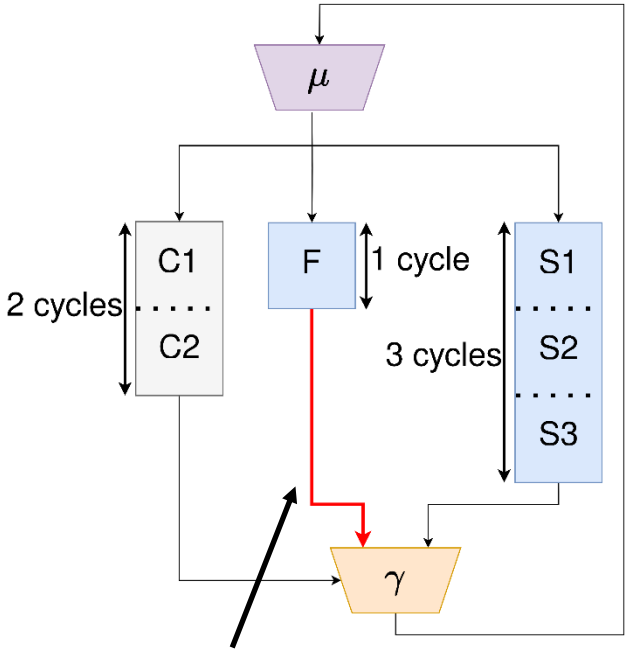


[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

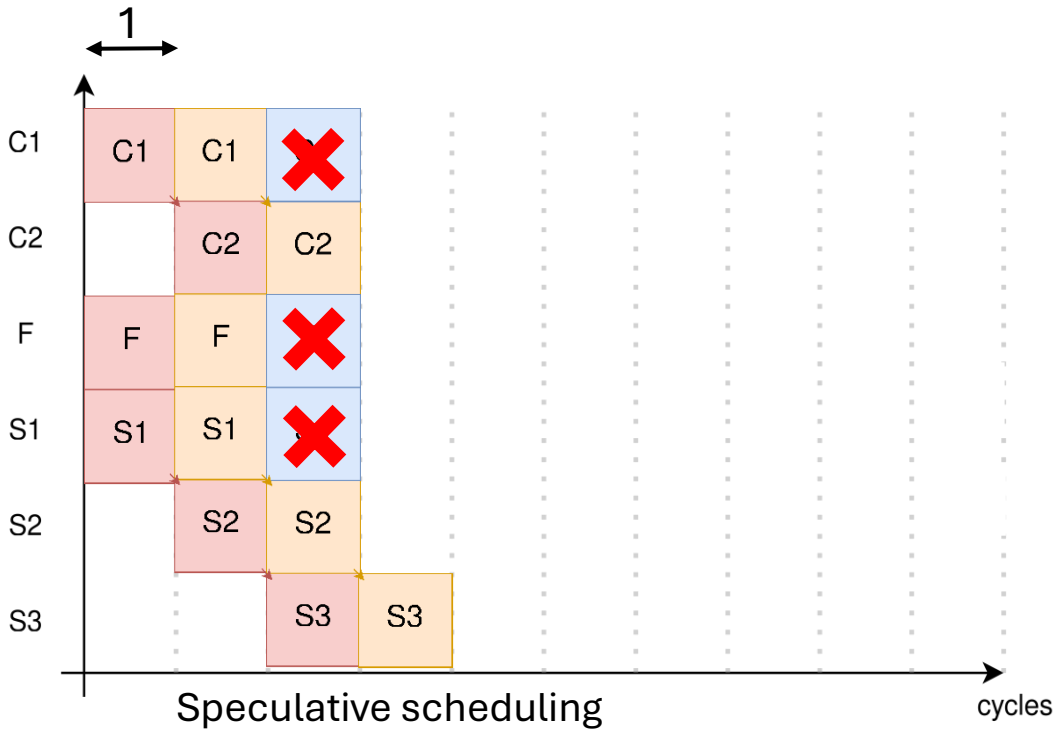
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input



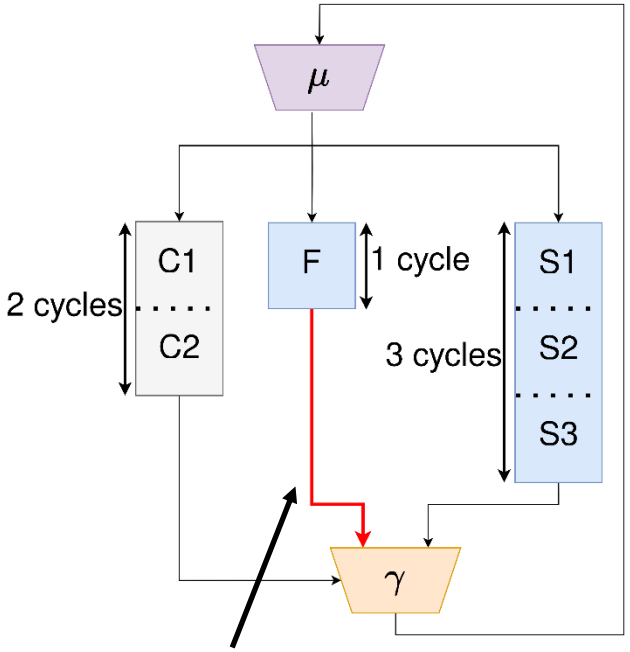
Speculative scheduling

[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

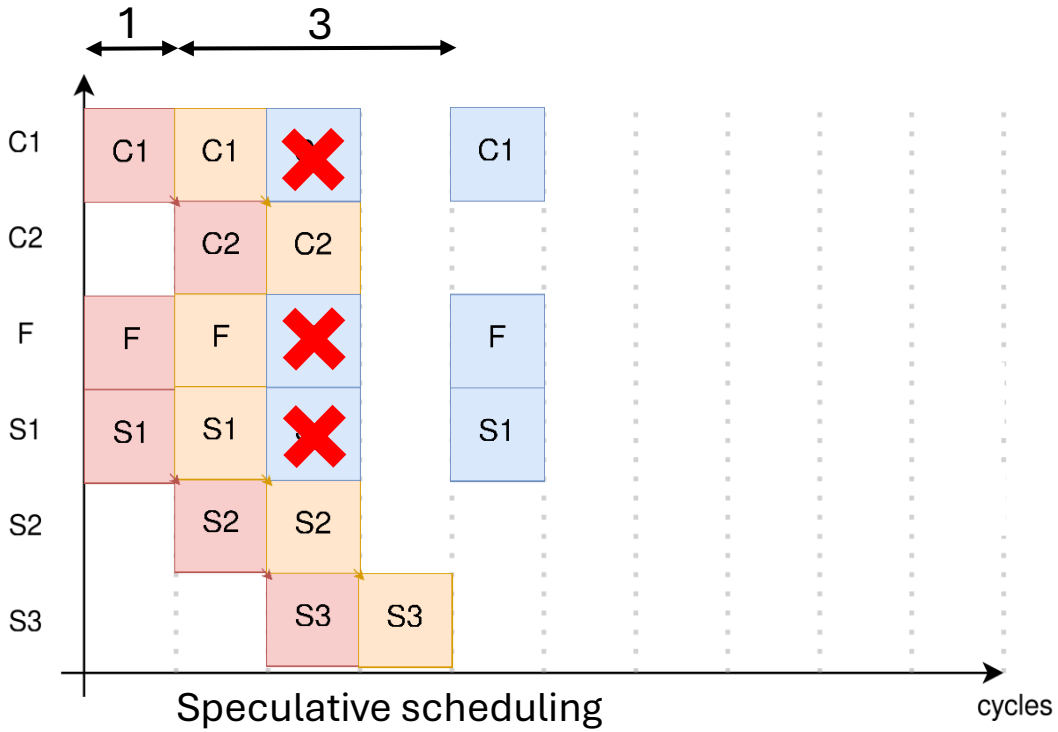
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input

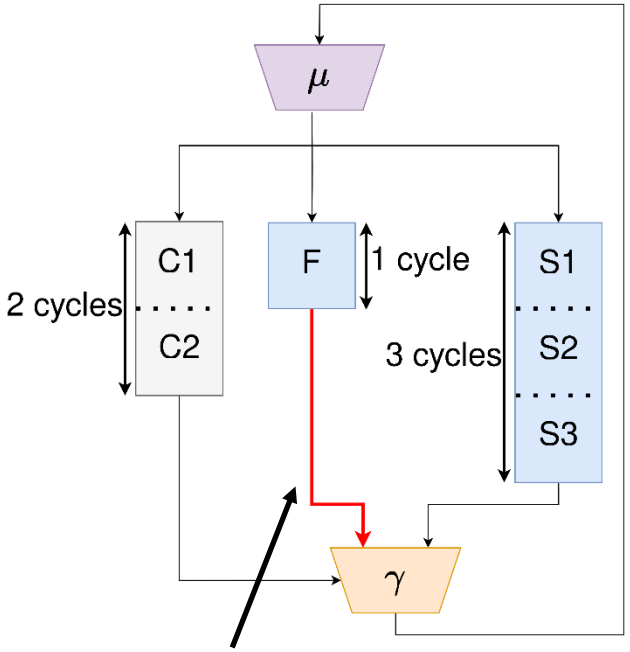


[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

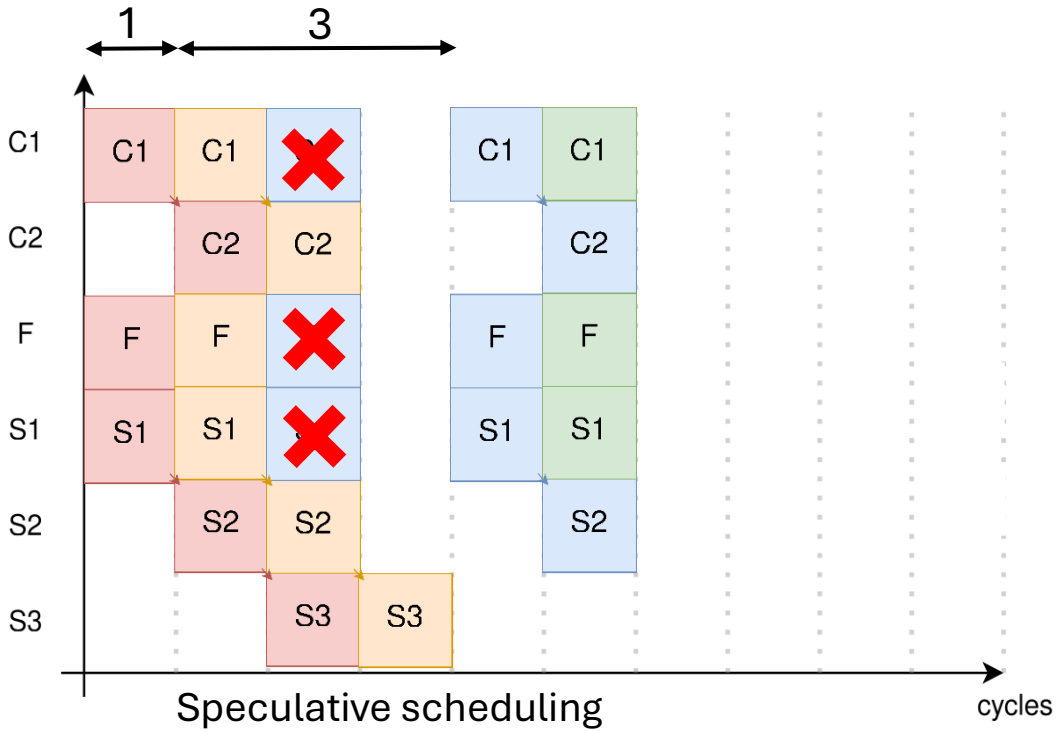
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input

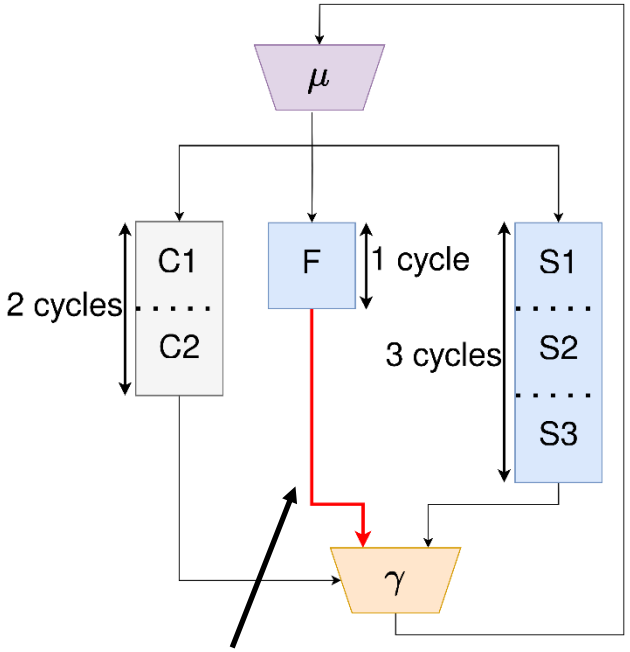


[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

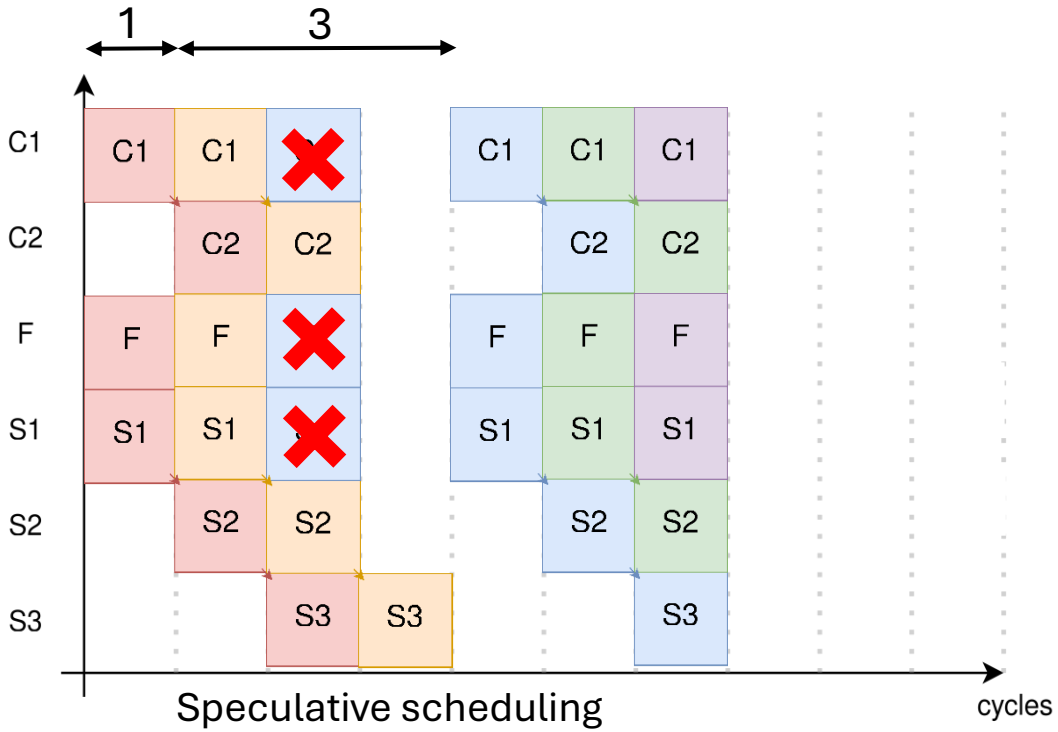
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input



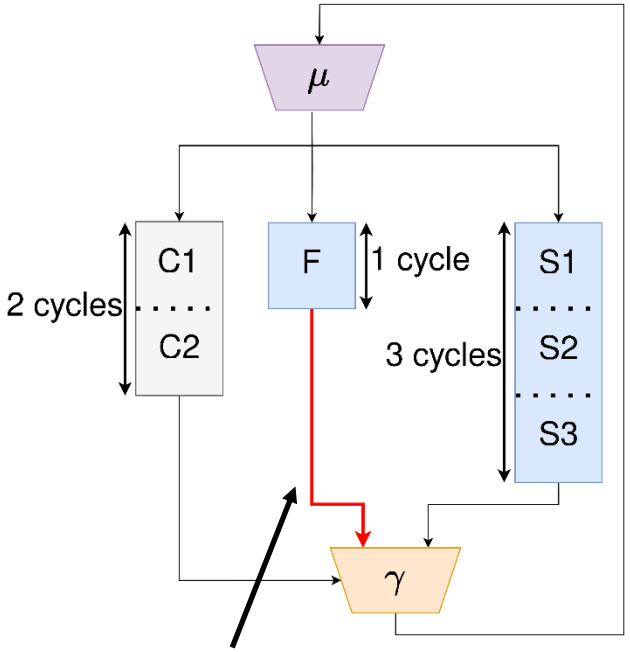
Speculative scheduling

[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

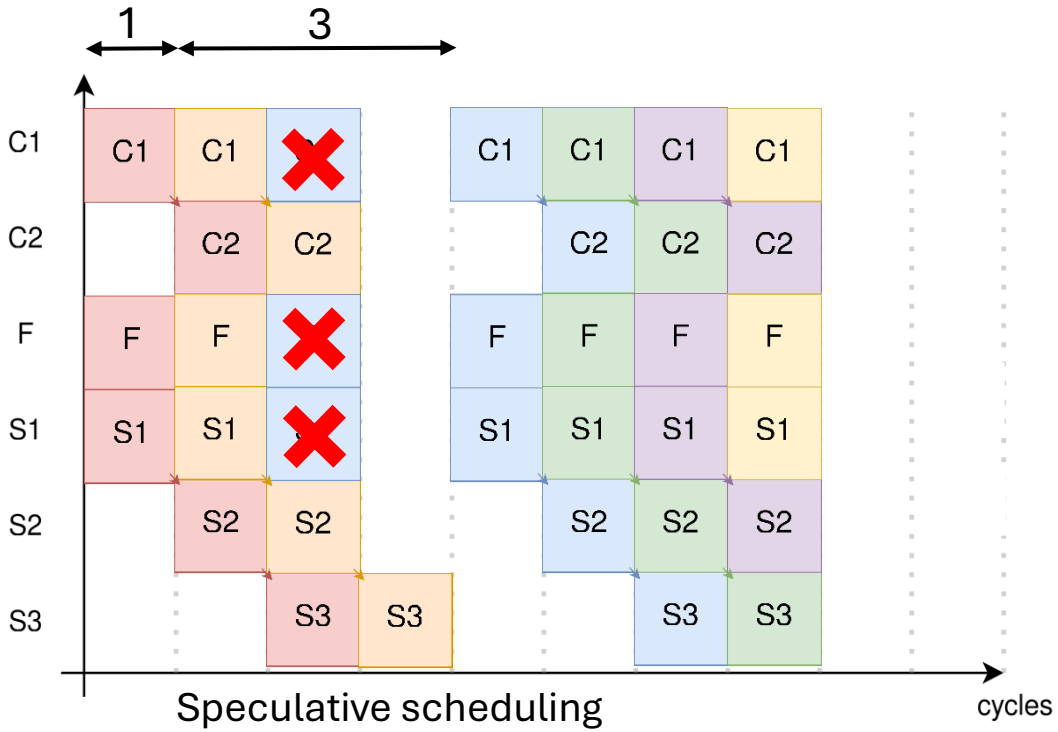
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input

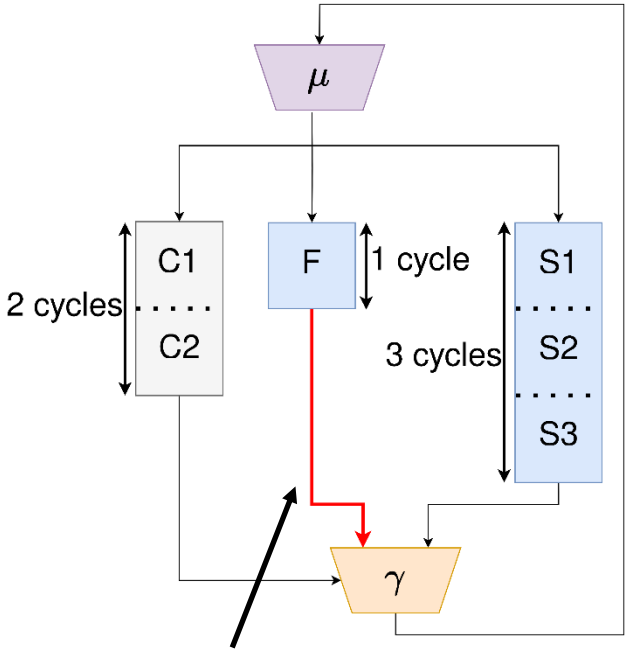


[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

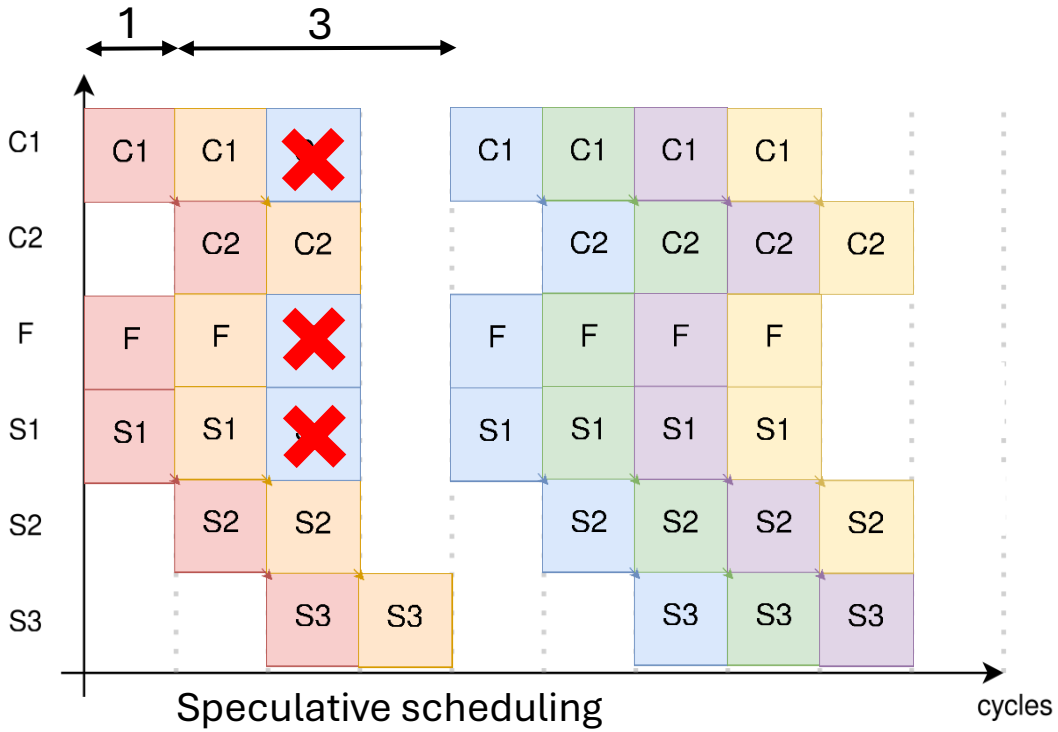
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input



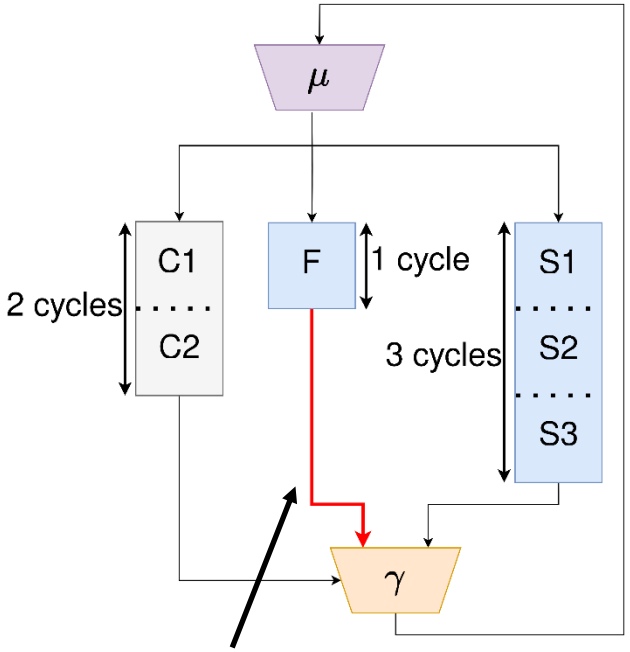
Speculative scheduling

[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

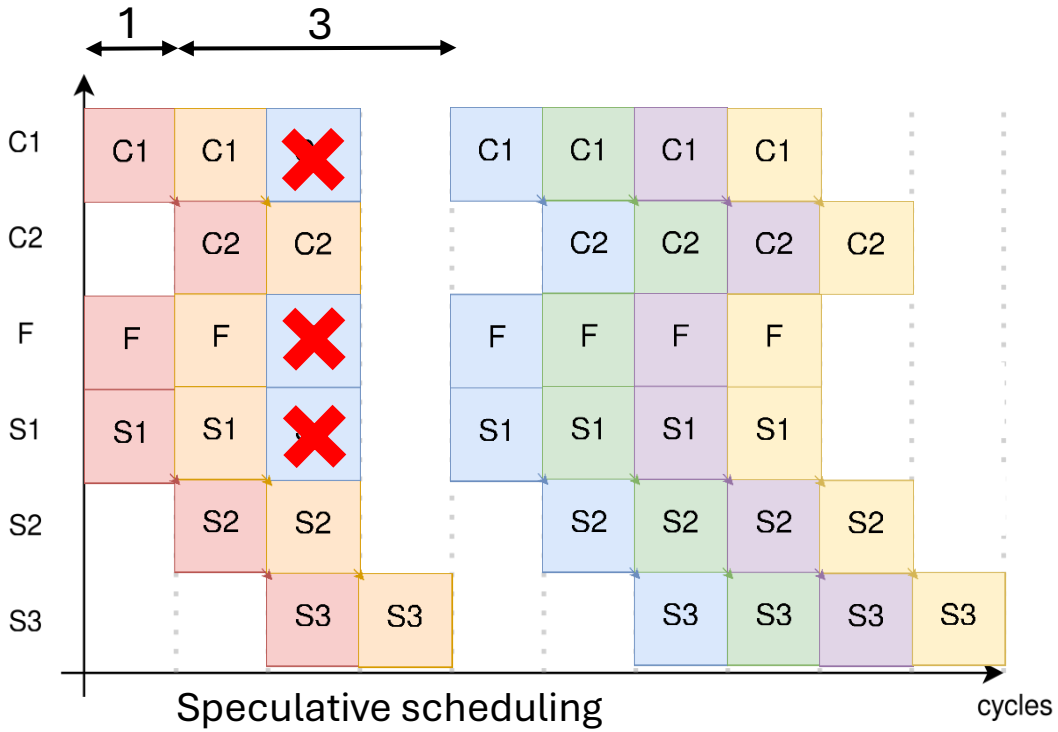
Can we do even better?

Speculative scheduling [2,3] starts future iterations by predicting conditions

```
while (1) {
  // 2 cycles
  if (C(x)) {
    // 1 cycle
    x = F(x);
  } else {
    // 3 cycles
    x = S(x);
  }
}
```



Speculation **hypothesis**:
the gamma node selects the fast input



Speculative scheduling

[2] S. Derrien, T. Marty, S. Rokicki, T. Yuki (2020). Toward Speculative Loop Pipelining for High-Level Synthesis. IEEE TCAD.
[3] L. Josipović, A. Guerrieri, P. Ienne (2019). Speculative Dataflow Circuits. FPGA'19

How far can I go with speculative HLS?

Use case: Automatically synthesize a pipelined CPU

RISC-V 32i Simulator

```
while (1) {
  instruction = fetch(pc);
  decoded = decode(instruction);
  switch (decoded.opcode) {
  case ADD:
    regs[decoded.rd] =
      regs[decoded.rs1] + regs[decoded.rs2];
    pc = pc + 4;
    break;
  case SUB:
    regs[decoded.rd] =
      regs[decoded.rs1] - regs[decoded.rs2];
    pc = pc + 4;
    break;
  ...
  }
}
```

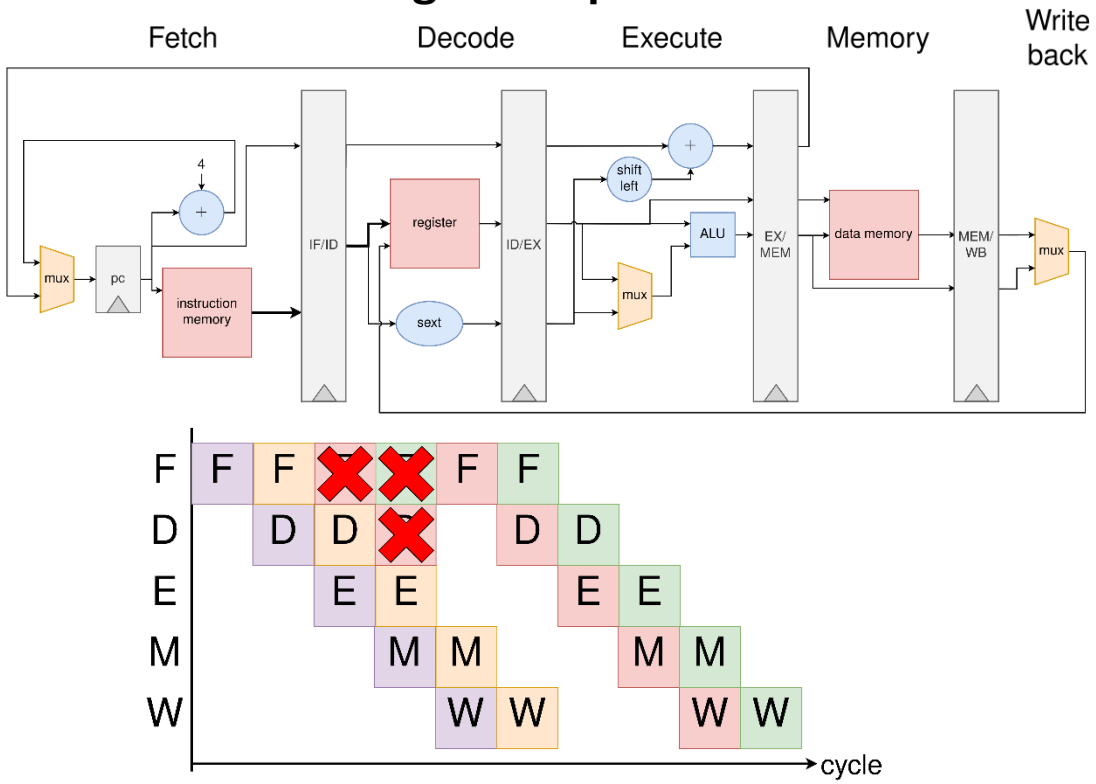
How far can I go with speculative HLS?

Use case: Automatically synthesize a pipelined CPU

RISC-V 32i Simulator

```
while (1) {
  instruction = fetch(pc);
  decoded = decode(instruction);
  switch (decoded.opcode) {
  case ADD:
    regs[decoded.rd] =
      regs[decoded.rs1] + regs[decoded.rs2];
    pc = pc + 4;
    break;
  case SUB:
    regs[decoded.rd] =
      regs[decoded.rs1] - regs[decoded.rs2];
    pc = pc + 4;
    break;
  ...
  }
}
```

Target datapath

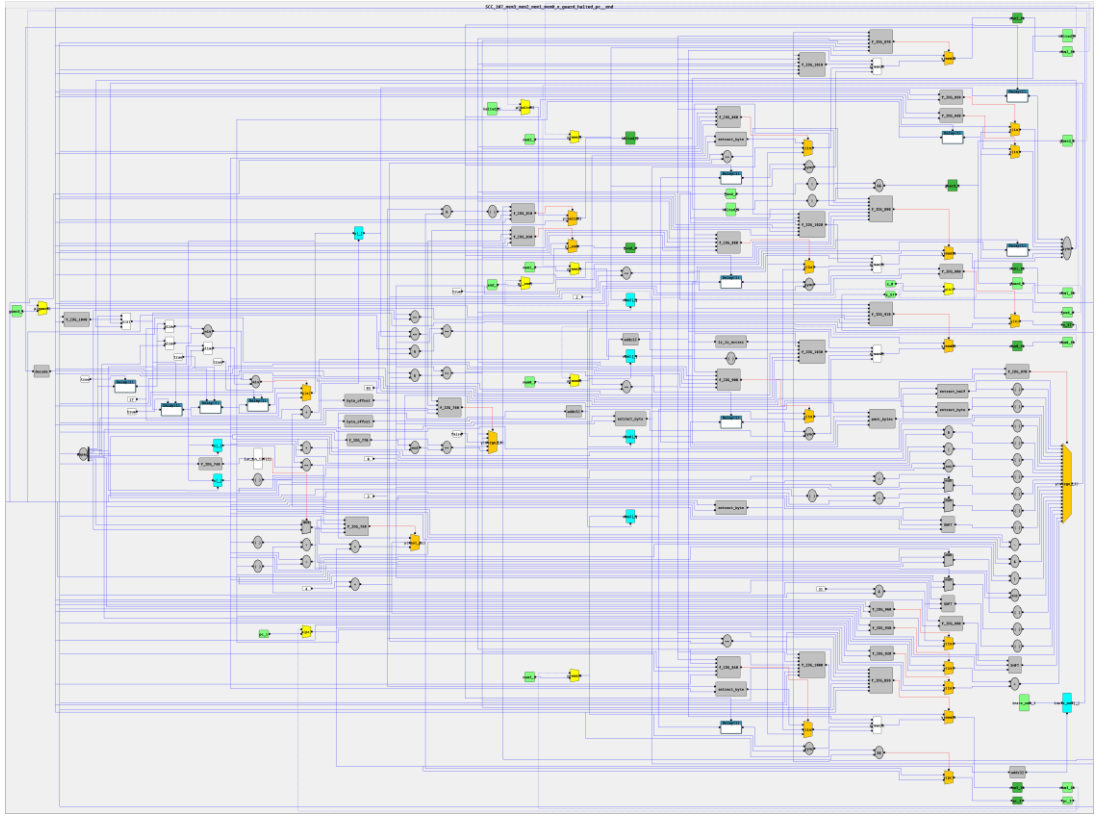


How far can I go with speculative HLS?

Use case: Automatically synthesize a pipelined CPU

RISC-V 32i Simulator

```
while (1) {
  instruction = fetch(pc);
  decoded = decode(instruction);
  switch (decoded.opcode) {
  case ADD:
    regs[decoded.rd] =
      regs[decoded.rs1] + regs[decoded.rs2];
    pc = pc + 4;
    break;
  case SUB:
    regs[decoded.rd] =
      regs[decoded.rs1] - regs[decoded.rs2];
    pc = pc + 4;
    break;
  ...
  }
}
```

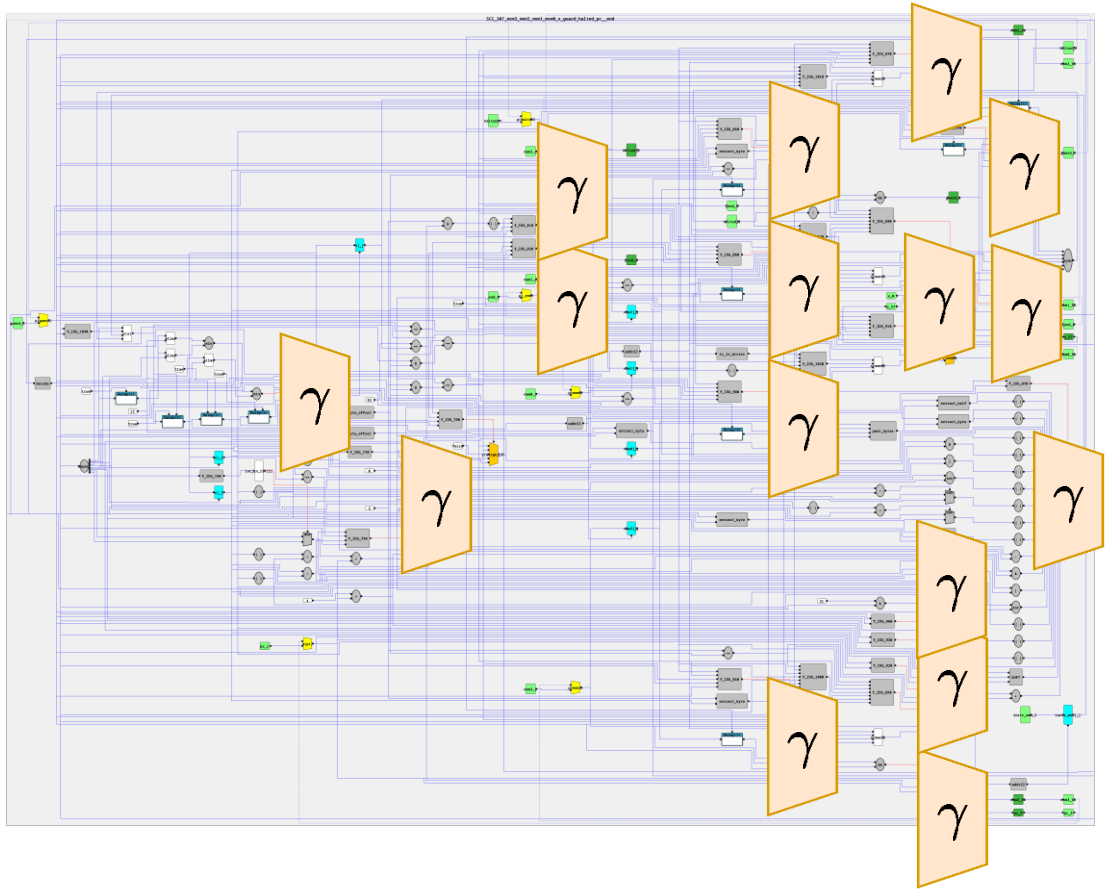


How far can I go with speculative HLS?

Use case: Automatically synthesize a pipelined CPU

RISC-V 32i Simulator

```
while (1) {
  instruction = fetch(pc);
  decoded = decode(instruction);
  switch (decoded.opcode) {
  case ADD:
    regs[decoded.rd] =
      regs[decoded.rs1] + regs[decoded.rs2];
    pc = pc + 4;
    break;
  case SUB:
    regs[decoded.rd] =
      regs[decoded.rs1] - regs[decoded.rs2];
    pc = pc + 4;
    break;
  ...
  }
}
```



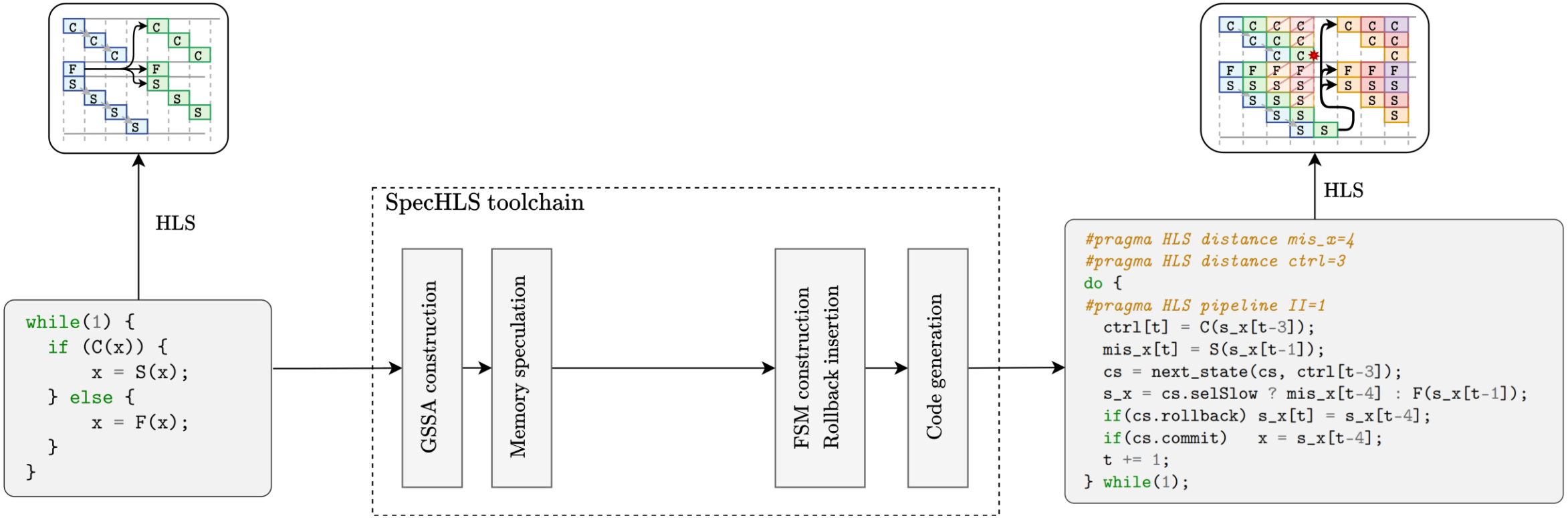
16 speculation options
→ 752M possibilities!

Exhaustive search is not possible!

Our Contribution

In this work, we address the problem of where, and how, to speculate

We build on an existing Speculative & dynamic HLS framework [4]

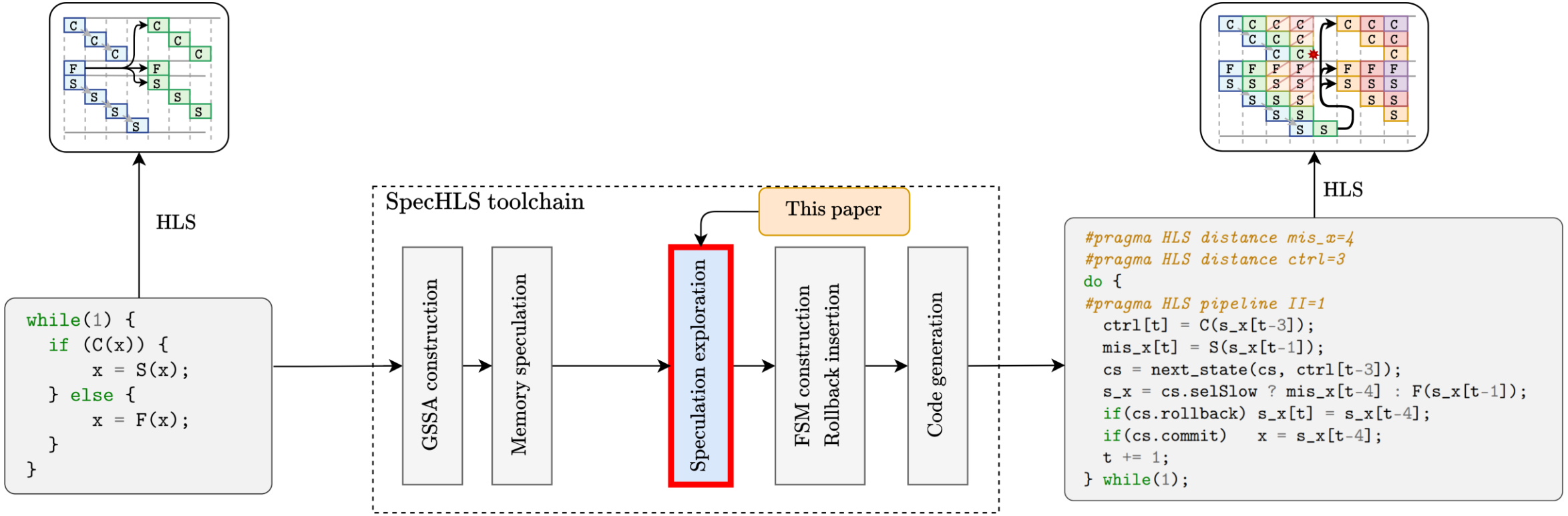


[4] Gorius, J.-M., Rokicki, S., and Derrien, S. (2022). SpecHLS: Speculative Accelerator Design using High-Level Synthesis. IEEE MICRO.

Our Contribution

In this work, we address the problem of where, and how, to speculate

We build on an existing Speculative & dynamic HLS framework [4]



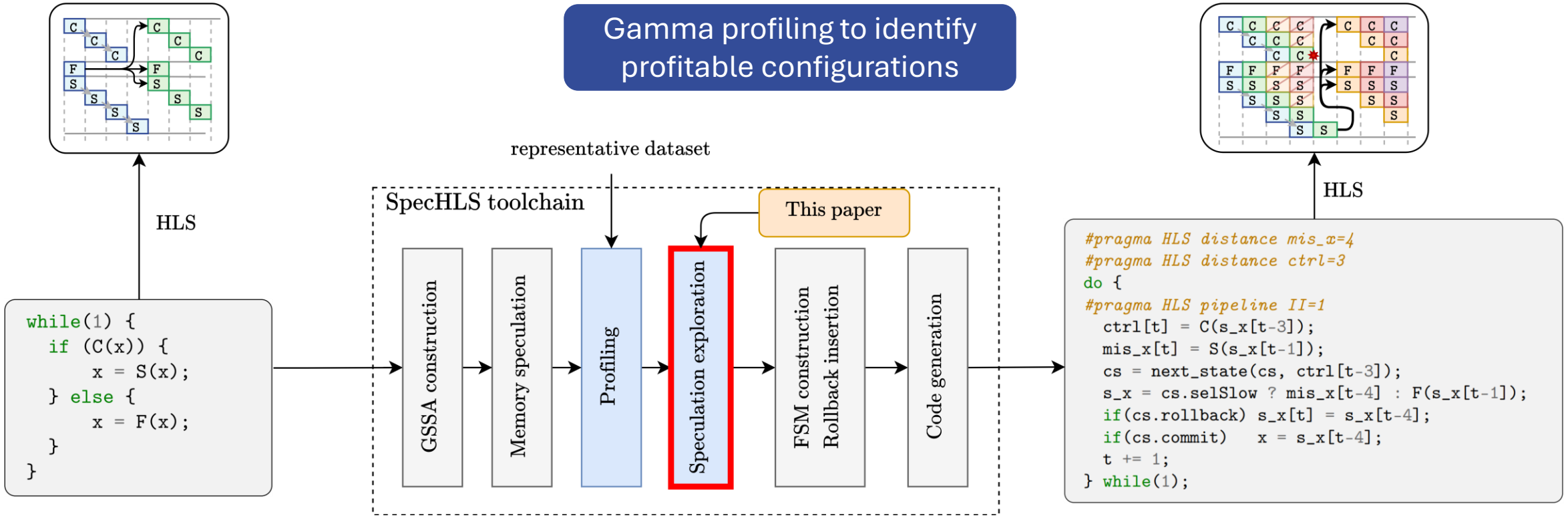
[4] Gorius, J.-M., Rokicki, S., and Derrien, S. (2022). SpecHLS: Speculative Accelerator Design using High-Level Synthesis. IEEE MICRO.

Our Contribution

In this work, we address the problem of where, and how, to speculate

We build on an existing Speculative & dynamic HLS framework [4]

Gamma profiling to identify profitable configurations

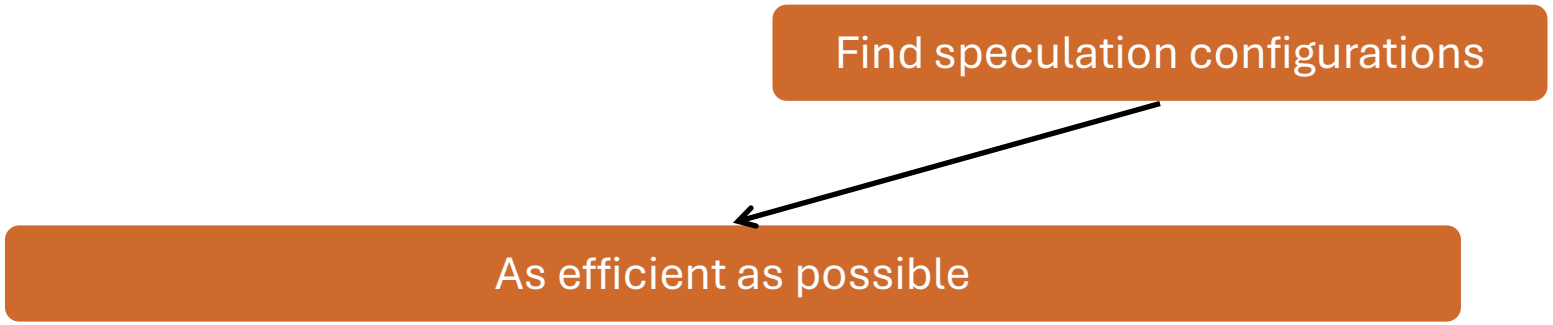


[4] Gorius, J.-M., Rokicki, S., and Derrien, S. (2022). SpecHLS: Speculative Accelerator Design using High-Level Synthesis. IEEE MICRO.

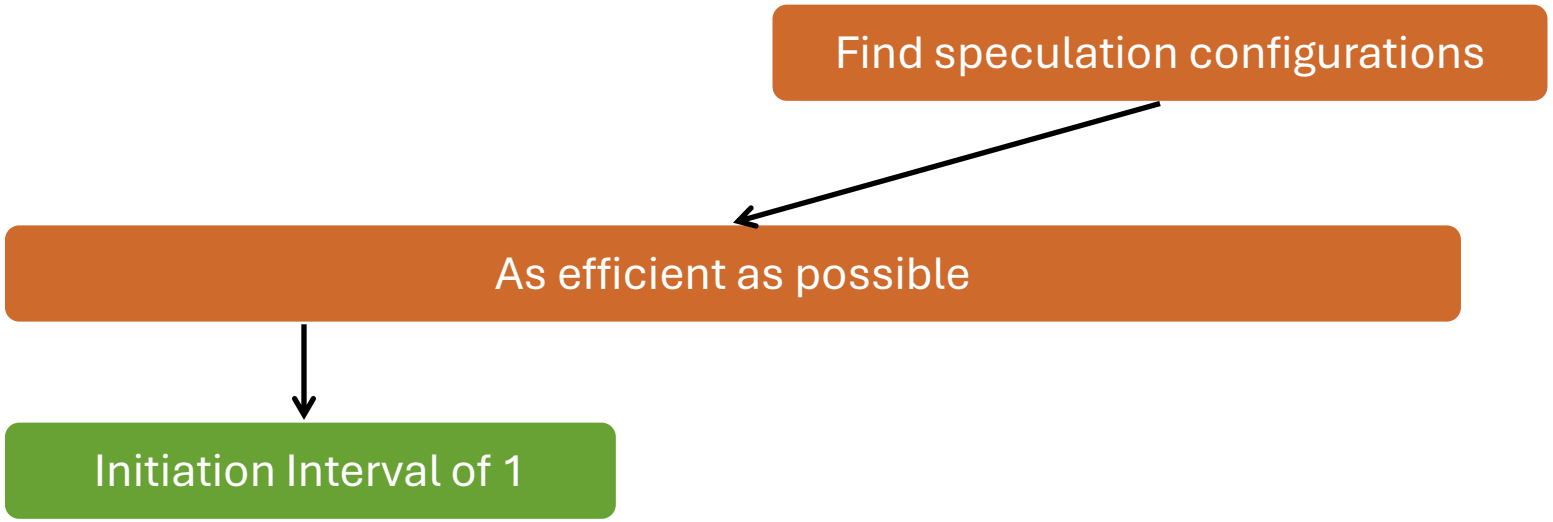
Problem statement

Find speculation configurations

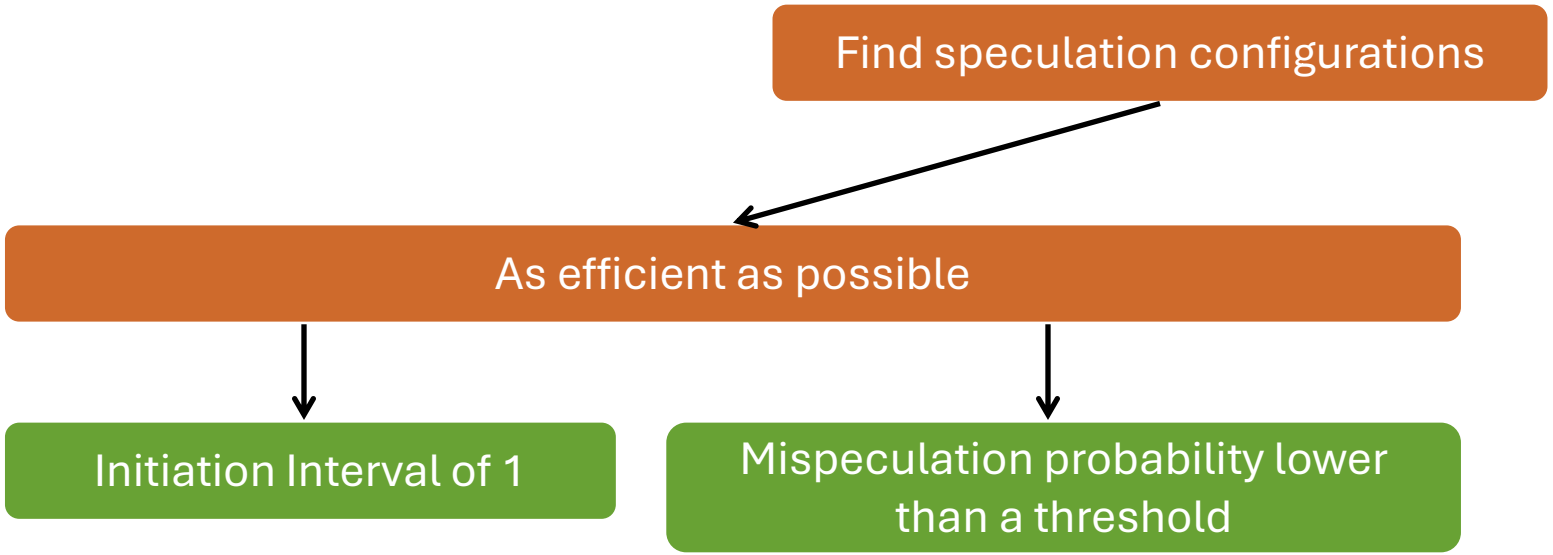
Problem statement



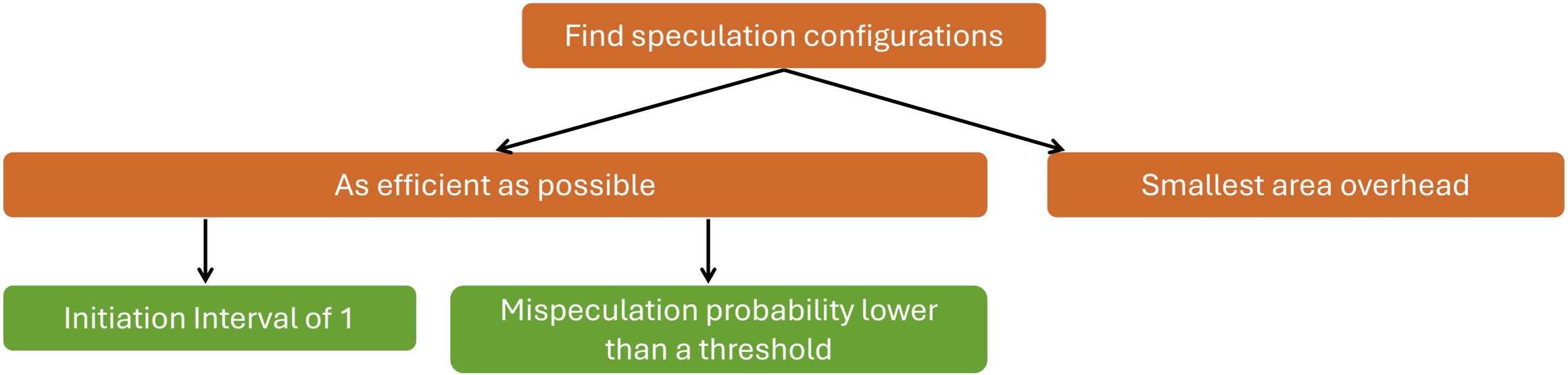
Problem statement



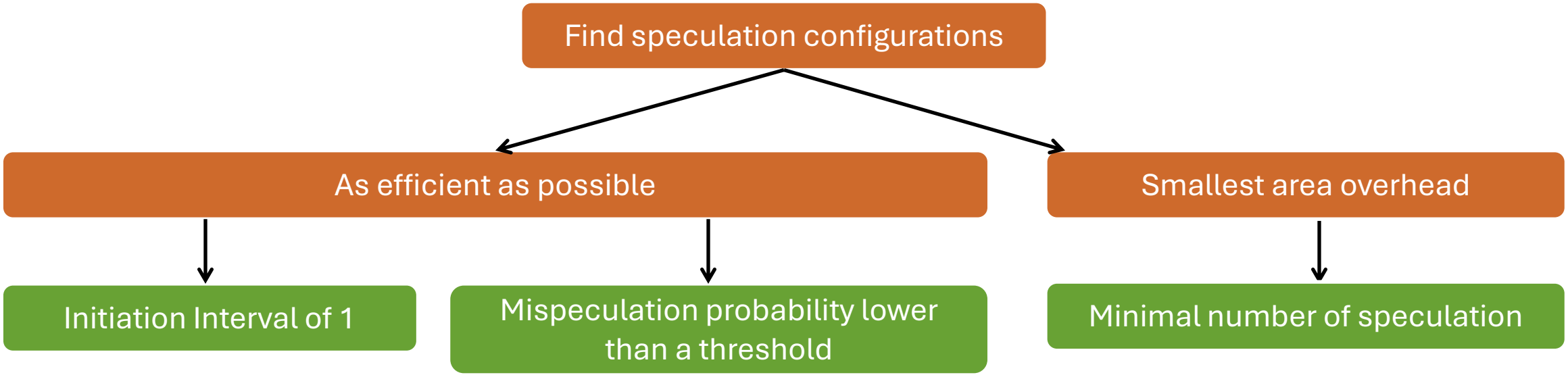
Problem statement



Problem statement



Problem statement



Problem statement

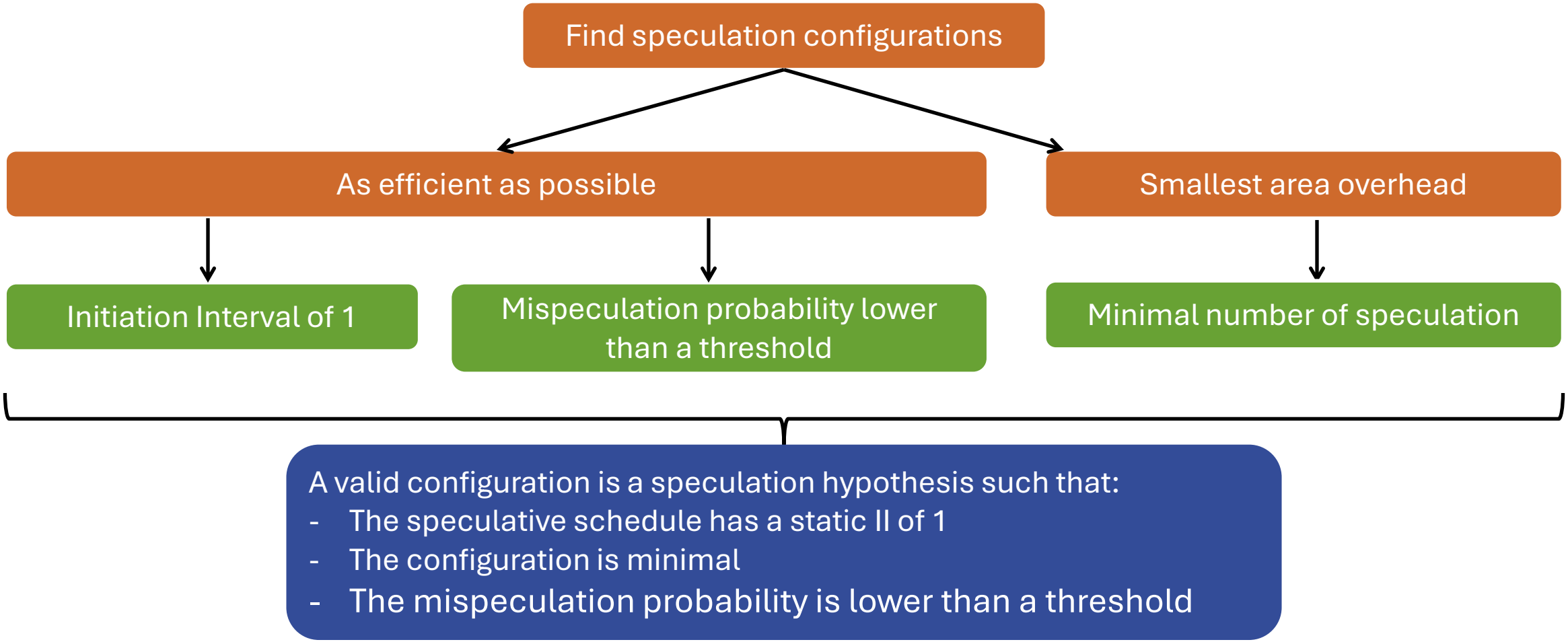
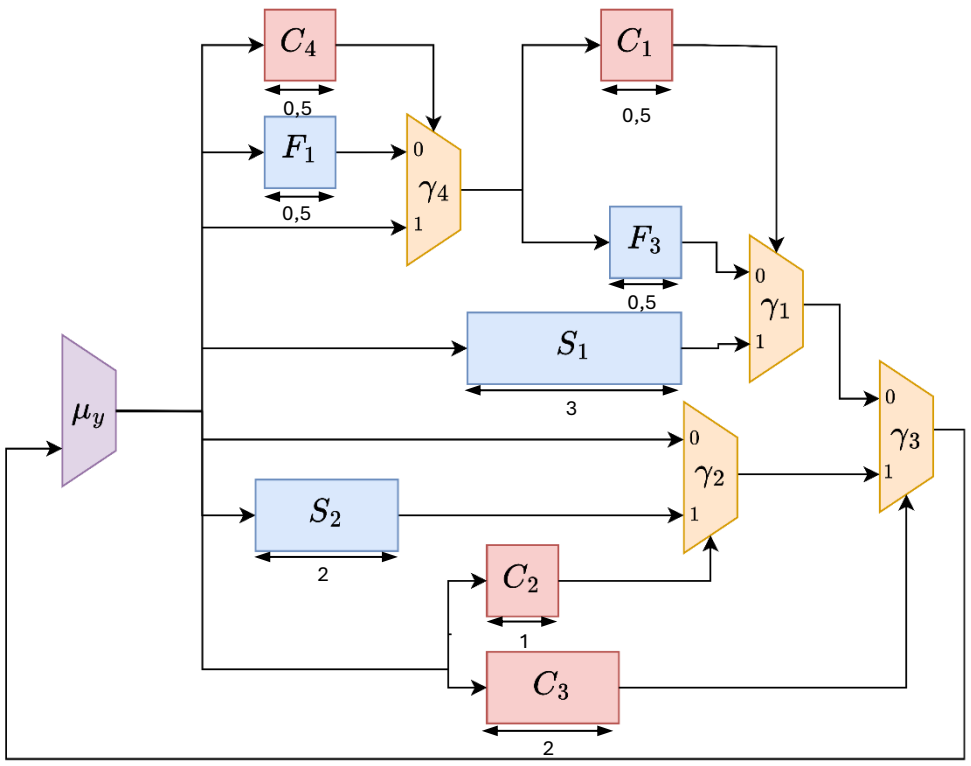


Illustration on a synthetic example

```
while (1) {
  if (C3(x)) {
      x = S2(x);
  } else {
    if (C4(x))
      tmp = x;
    else
      tmp = F1(x);
    if (C1(tmp))
      x = S1(x);
    else
      x = F3(tmp);
  }
}
```

Configurations probabilities
⚠️ gamma probabilities are not independent

Dataflow representation



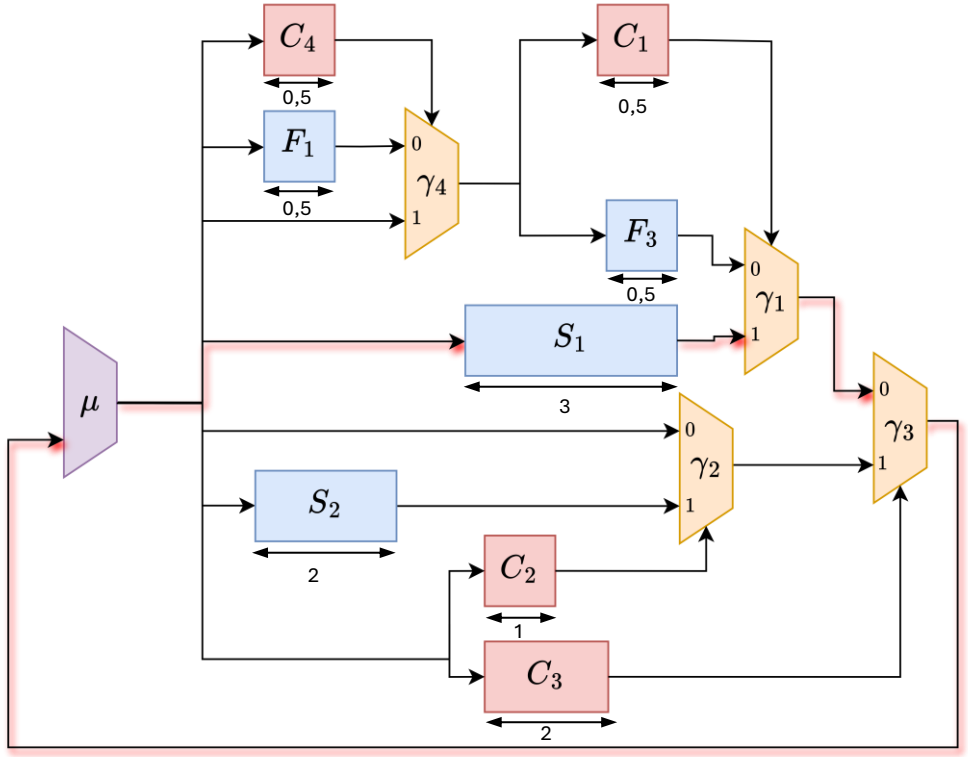
3^4=81 possible speculation configurations for this example

Example of configurations

ex: 50%

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has a static II of 1
- The configuration is minimal
- The mispeculation probability is lower than a threshold

Configuration	II	Mispeculation probability
Ø	3	0%
...		



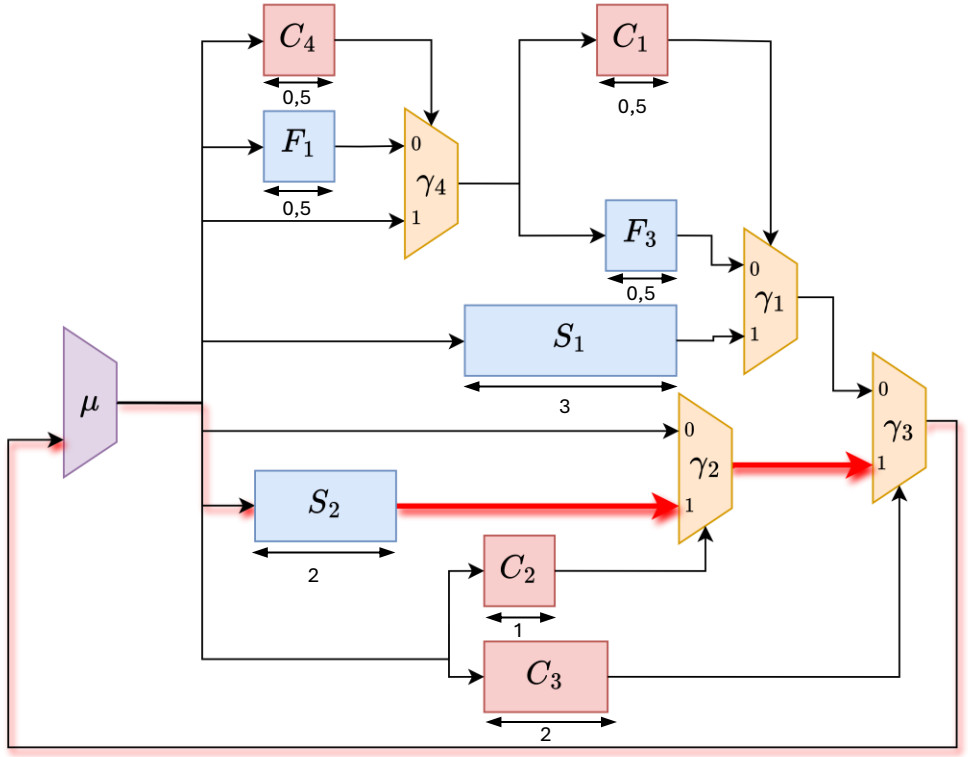
Example of configurations

A valid configuration is a speculation hypothesis such that:

- **The speculative schedule has a static II of 1**
- The configuration is minimal
- The mispeculation probability is lower than a threshold

ex: 50%

Configuration	II	Mispeculation probability
∅	3	0%
$\gamma_2 \rightarrow 1$; $\gamma_3 \rightarrow 1$; ...	2	...
...		

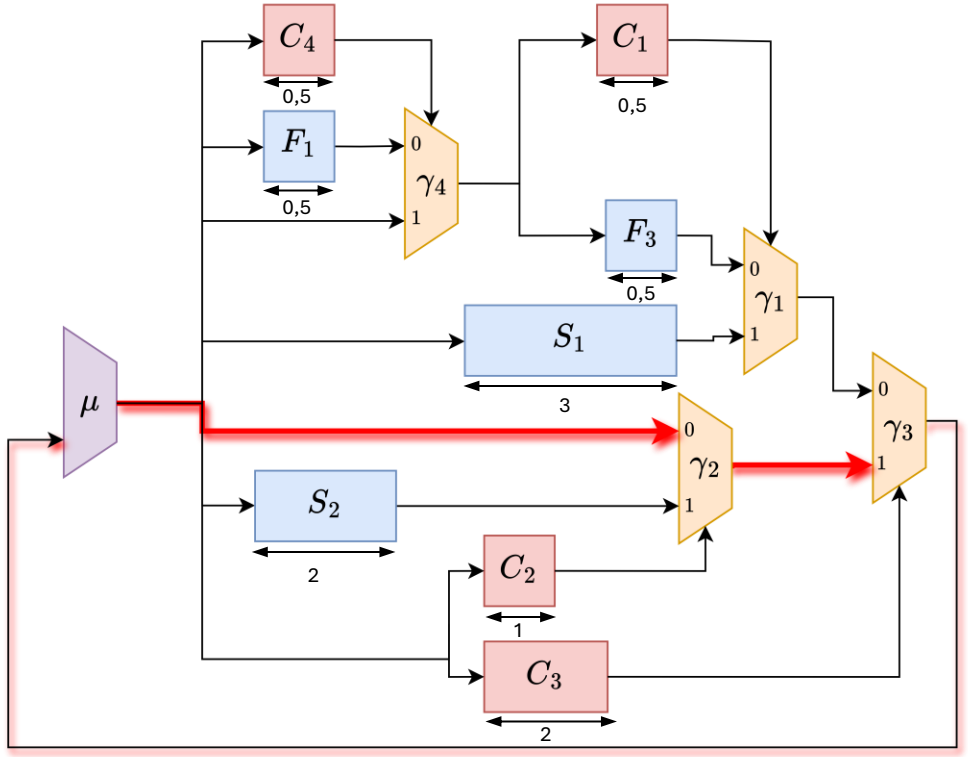


Example of configurations

ex: 50%

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has a static II of 1
- The configuration is minimal
- **The mispeculation probability is lower than a threshold**

Configuration	II	Mispeculation probability
Ø	3	0%
$\gamma_2 \rightarrow 1$; $\gamma_3 \rightarrow 1$; …	2	…
$\gamma_2 \rightarrow 0$; $\gamma_3 \rightarrow 1$	1	90%
…		



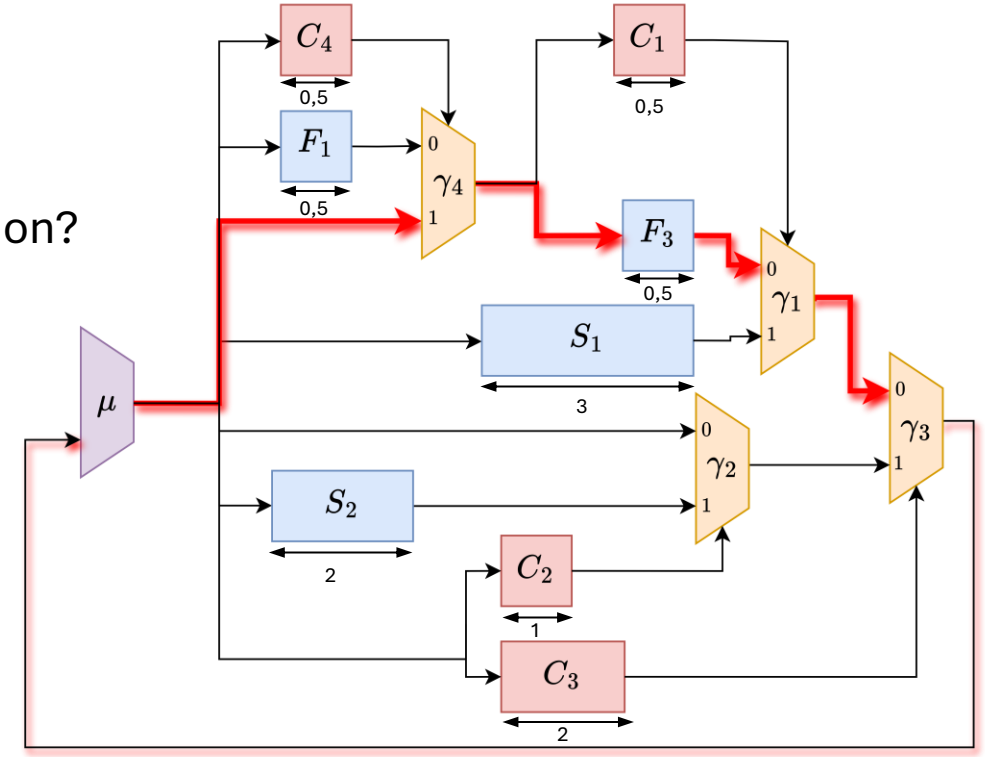
Example of configurations

ex: 50%

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has a static II of 1
- The configuration is minimal
- The mispeculation probability is lower than a threshold

Configuration	II	Mispeculation probability
Ø	3	0%
$\gamma_2 \rightarrow 1$; $\gamma_3 \rightarrow 1$; …	2	…
$\gamma_2 \rightarrow 0$; $\gamma_3 \rightarrow 1$	1	90%
$\gamma_1 \rightarrow 0$; $\gamma_3 \rightarrow 0$; $\gamma_4 \rightarrow 1$	1	45%
…		

Valid configuration?



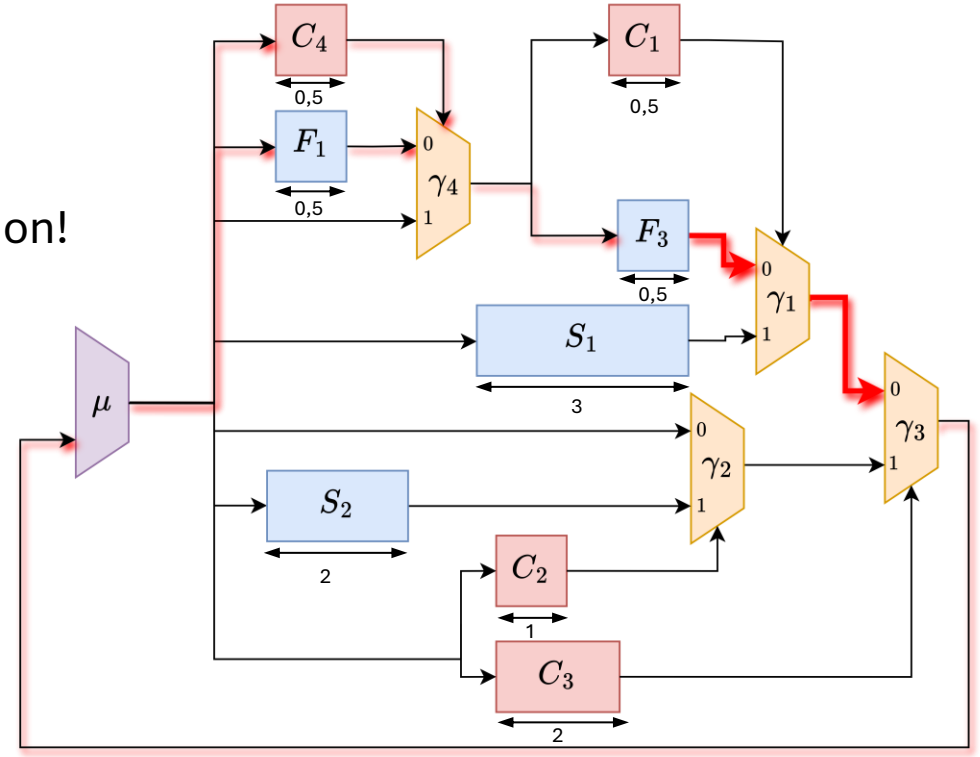
Example of configurations

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has a static II of 1
- **The configuration is minimal**
- The mispeculation probability is lower than a threshold

ex: 50%

Configuration	II	Mispeculation probability
Ø	3	0%
$\gamma_2 \to 1$; $\gamma_3 \to 1$; …	2	…
$\gamma_2 \to 0$; $\gamma_3 \to 1$	1	90%
$\gamma_1 \to 0$; $\gamma_3 \to 0$; $\gamma_4 \to 1$	1	45%
$\gamma_1 \to 0$; $\gamma_3 \to 0$	1	40%
…		

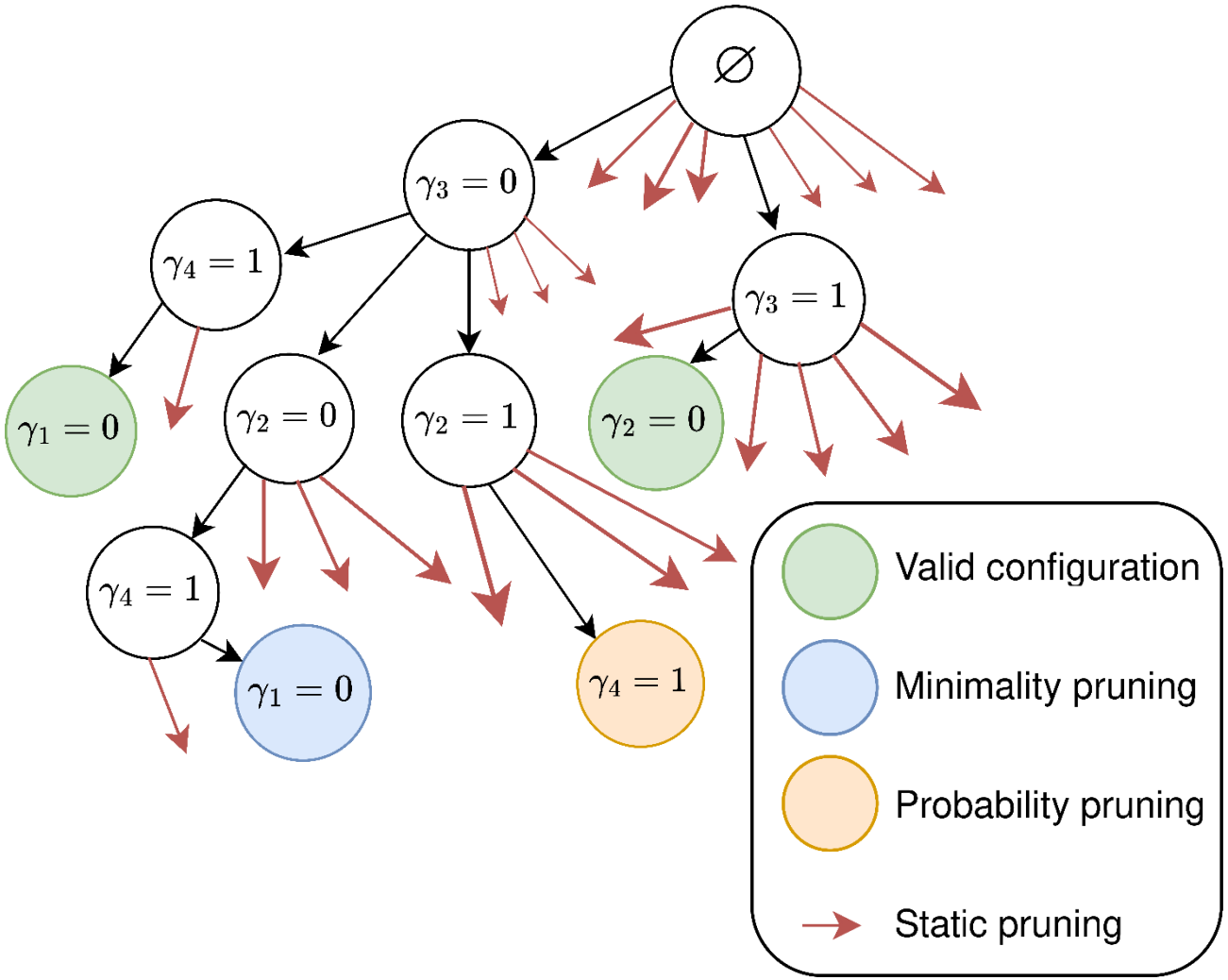
Valid configuration!



Proposed approach: a branch and bound algorithm

Branch-and-bound algorithm find every valid configurations

Configuration sorted by increasing speculation profitability



Experimental validation: a scalable approach

Benchmark	Design space size			Runtime
	γ-nodes	Baseline	Heuristics	
FPU	21	30.9B	116k	1055s
SpMM	12	708k	60	6s
RISC-V CPU	16	752M	1.46k	263s
Superscalar	16	178M	1.19k	177s

The proposed approach decreases by several orders of magnitude the number of explored configurations

Experimental validation: a scalable approach

Benchmark	Design space size			Runtime
	γ-nodes	Baseline	Heuristics	
FPU	21	30.9B	116k	1055s
SpMM	12	708k	60	6s
RISC-V CPU	16	752M	1.46k	263s
Superscalar	16	178M	1.19k	177s

The proposed approach decreases by several orders of magnitude the number of explored configurations

Conclusion

Speculation opens up new opportunities for High-Level Synthesis

One challenge is to discover **where**, and **how**, to apply speculation

With our approach, it is possible to explore real-world examples within a reasonable time frame.