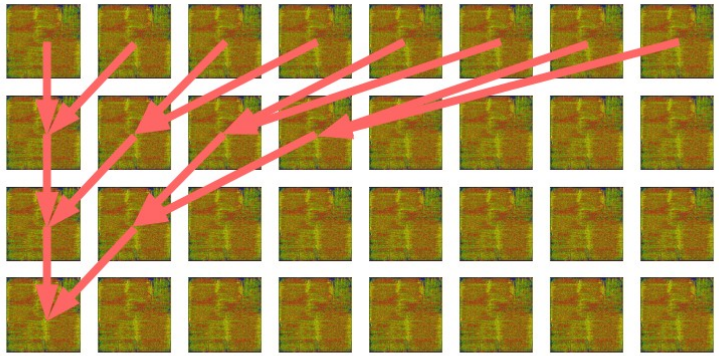
Pipelining a Triggered Processing Element

Thomas J. Repetti, João P. Cerqueira, Martha A. Kim, Mingoo Seok
Columbia University
MICRO 2017

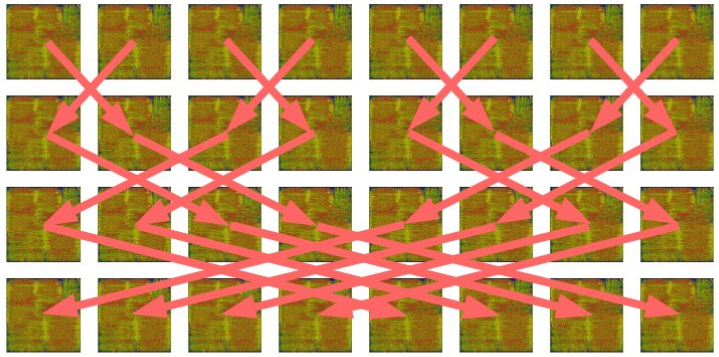
Outline

1. **Motivation**
2. Background
3. Challenges of Triggered Pipelines
4. Optimizations
5. Infrastructure and Methodology
6. Results and Conclusion

Spatial Accelerators



Merge Sort



DFT Butterfly

- Many domain workload specific
- Some general purpose
- Wide variety of applications:
  - Signal Processing
  - Scientific Computing
  - Data Analysis
  - Computer Vision
  - Cryptography

Figures of Merit

$$\frac{\text{Energy}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Energy}}{\text{Instruction}}$$

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Time}}{\text{Instruction}}$$

Figures of Merit

$$\frac{\text{Energy}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Energy}}{\text{Instruction}}$$

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Time}}{\text{Instruction}}$$

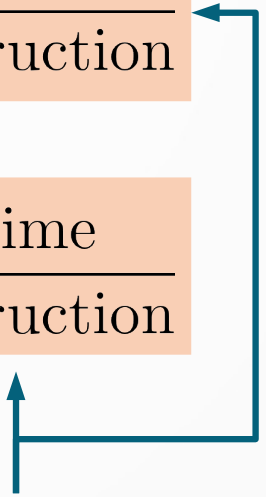
Program and ISA → Dynamic instruction count

Parashar, Pellauer, et al. *Triggered Instructions: A Control Paradigm for Spatially-Programmed Architectures*, ISCA 2013

Figures of Merit

$$\frac{\text{Energy}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Energy}}{\text{Instruction}}$$

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Time}}{\text{Instruction}}$$



Microarchitecture and Circuit

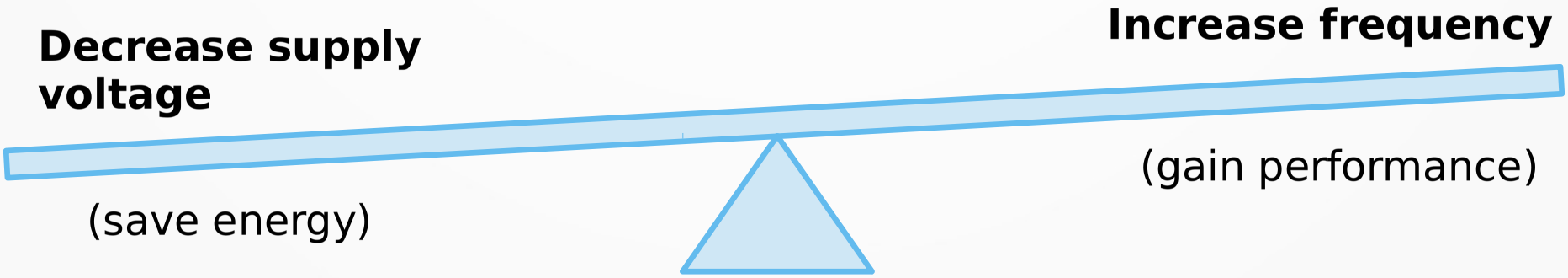
Support for deep pipelines gives fine-grain control over energy and delay at the microarchitectural and circuit level

Benefits of Deeper Pipelines

- Prerequisite for non-trivial, high-latency functional units

Benefits of Deeper Pipelines

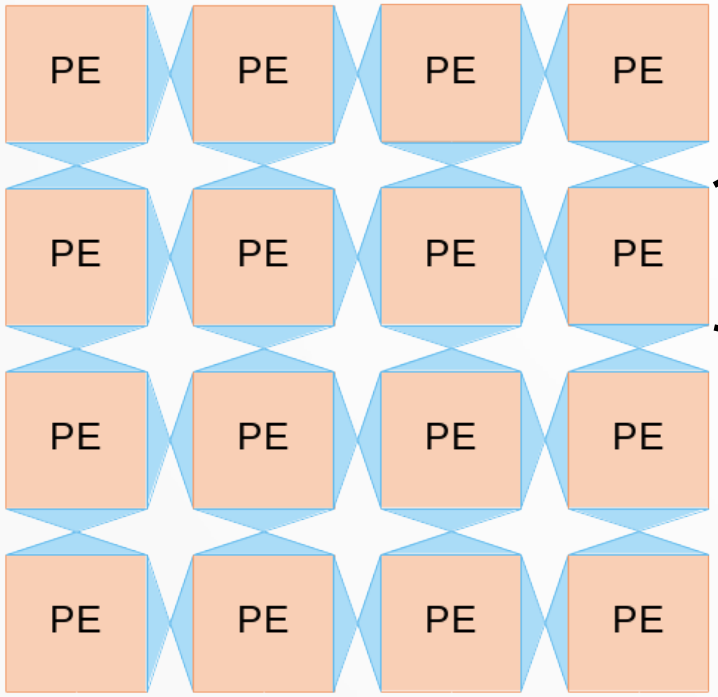
- Prerequisite for non-trivial, high-latency functional units
- For a given design, the increased timing slack gives you **options:**



Outline

1. Motivation
2. **Background**
3. Challenges of Triggered Pipelines
4. Optimizations
5. Infrastructure and Methodology
6. Results and Conclusion

Background on Triggered Instruction Architectures



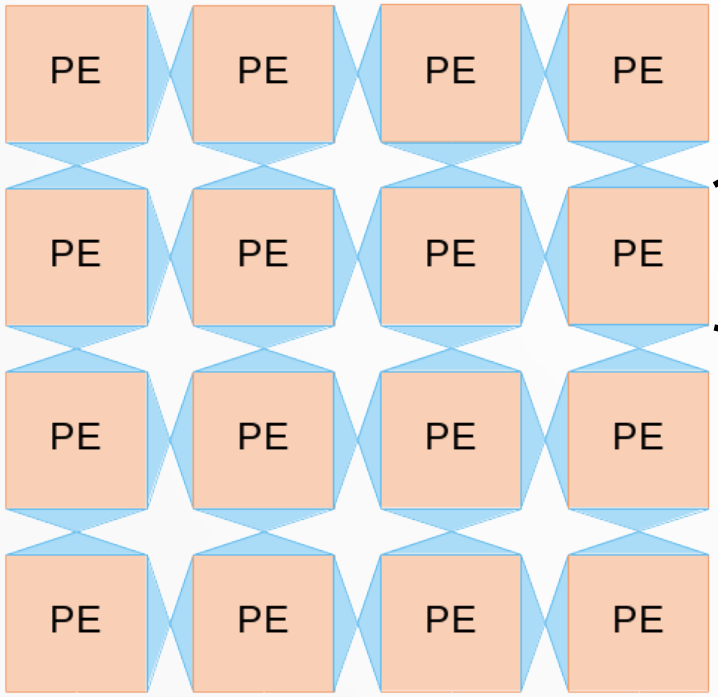

```
when %p == XX00 with %i0.0, %i1.0:
    mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
    add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
    ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
    mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
    halt;
```

Processing Element

Latency Insensitive Interconnect

- **Locally autonomous**
- **No program counter**
- **PEs communicate through queues**

Background on Triggered Instruction Architectures




```
when %p == XX00 with %i0.0, %i1.0:
    mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
    add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
    ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
    mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
    halt;
```

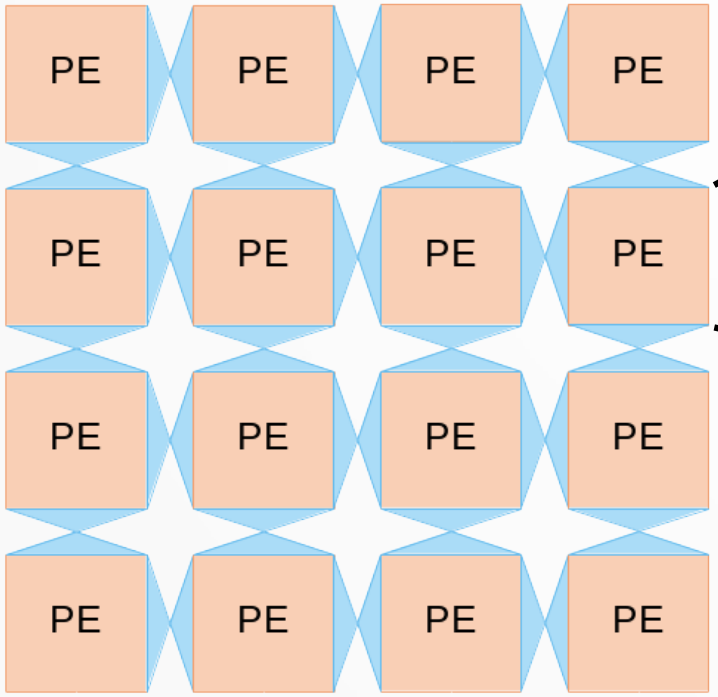
Actions: function of program

Processing Element

Latency Insensitive Interconnect

- **Locally autonomous**
- **No program counter**
- **PEs communicate through queues**

Background on Triggered Instruction Architectures




```
when %p == XX00 with %i0.0, %i1.0:
    mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
    add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
    ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
    mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
    halt;
```

Processing Element

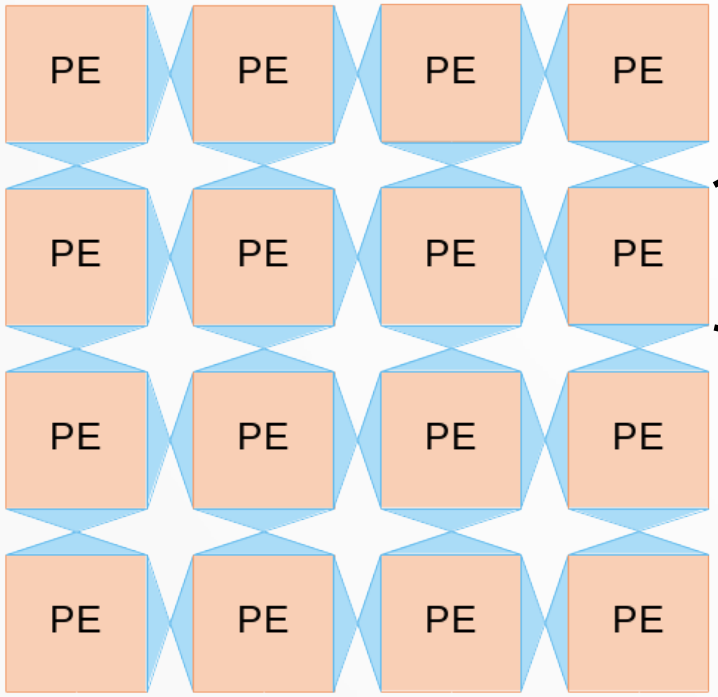
Latency Insensitive Interconnect

Actions: function of program

Guards: control flow

- **Locally autonomous**
- **No program counter**
- **PEs communicate through queues**

Background on Triggered Instruction Architectures




```
when %p == XX00 with %i0.0, %i1.0:
    mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
    add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
    ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
    mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
    halt;
```

Processing Element

Latency Insensitive Interconnect

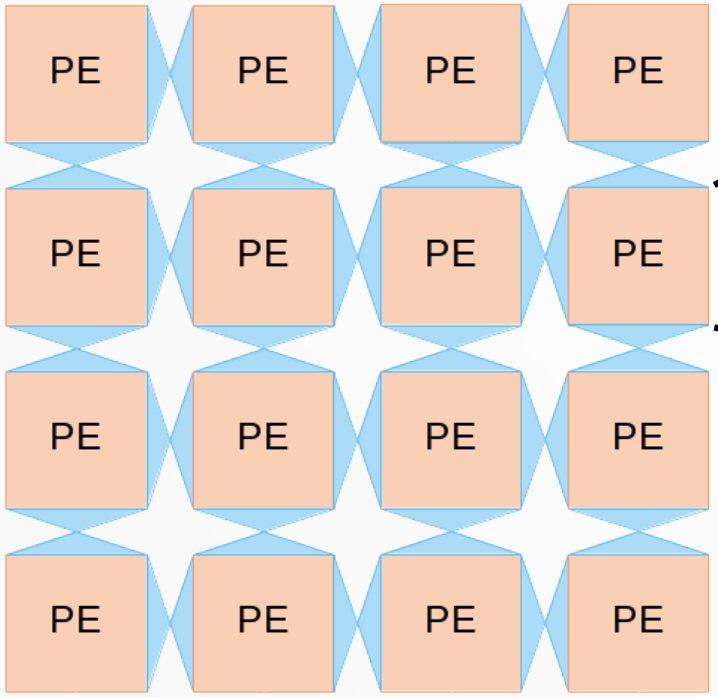
Actions: function of program

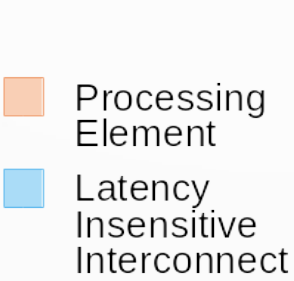
Guards: control flow

Predicate update: ~ PC = PC + 4

- **Locally autonomous**
- **No program counter**
- **PEs communicate through queues**

Background on Triggered Instruction Architectures






```
when %p == XX00 with %i0.0, %i1.0:
    mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
    add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
    ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
    mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
    halt;
```

Actions: function of program

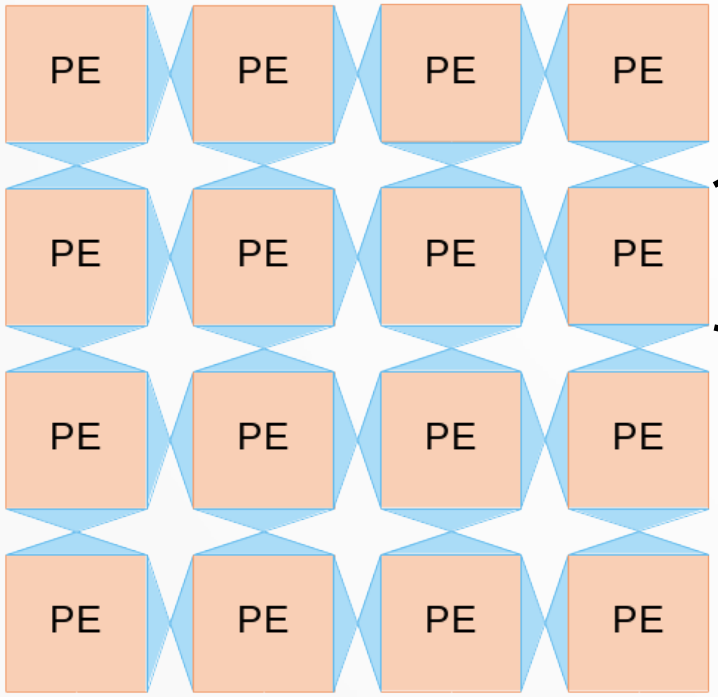
Guards: control flow

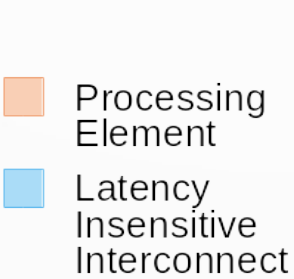
Predicate update: ~ PC = PC + 4

Queue operations: Spatial communication

- **Locally autonomous**
- **No program counter**
- **PEs communicate through queues**

Background on Triggered Instruction Architectures






```
when %p == XX00 with %i0.0, %i1.0:
    mul %r1, %i0, %i1; deq %i0, %i1; set %p = ZZ01;
when %p == XX01:
    add %r0, %r0, %r1; set %p = ZZ10;
when %p == XX10:
    ugt %p3, %r0, %r2; set %p = ZZ11;
when %p == X011:
    mov %o1.0, %r0; set %p = ZZ00;
when %p == X111:
    halt;
```

- **Locally autonomous**
- **No program counter**
- **PEs communicate through queues**

Actions: function of program

Guards: control flow

Predicate update: ~ PC = PC + 4

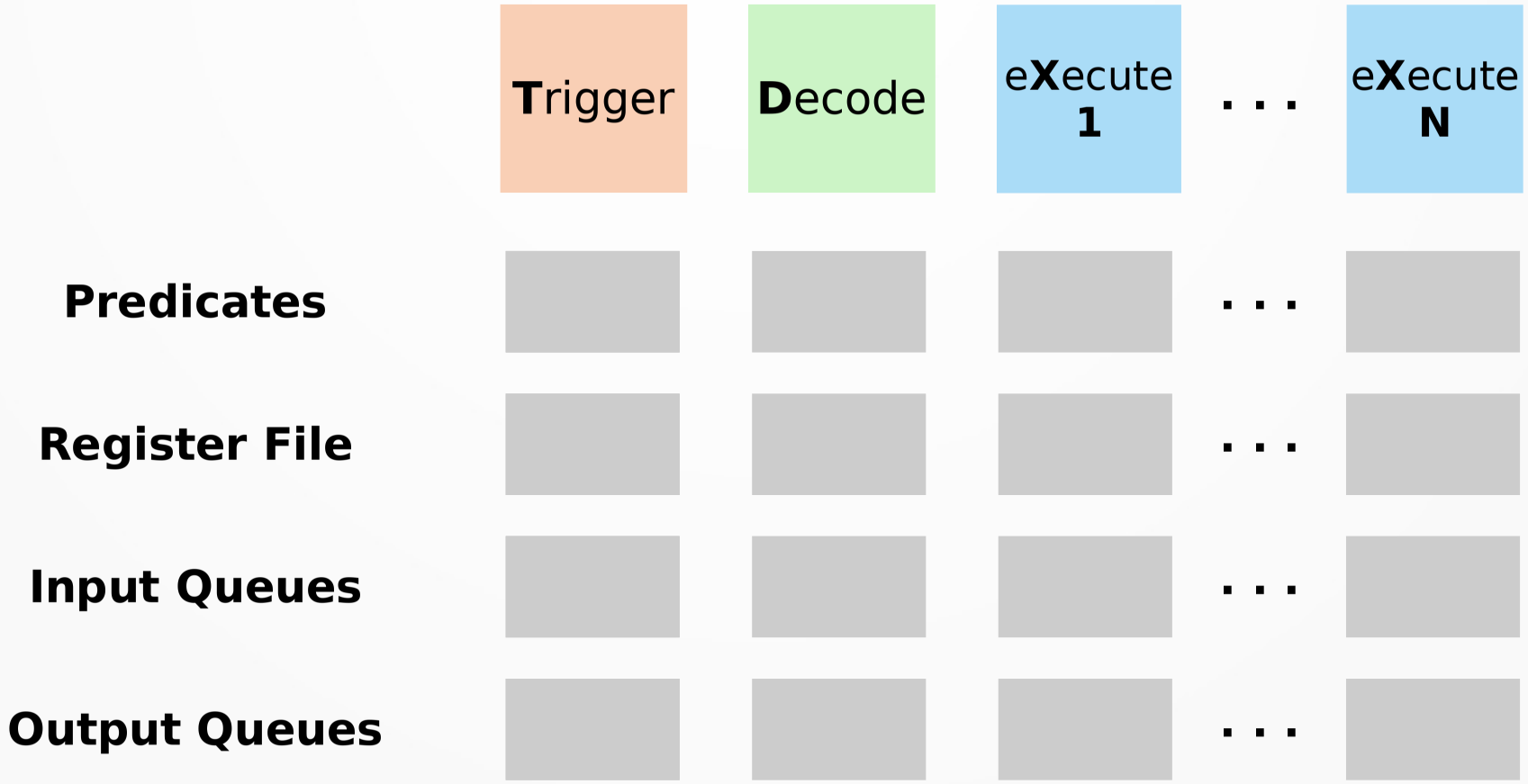
Queue operations: Spatial communication

Datapath predicate write: data-dependent control flow

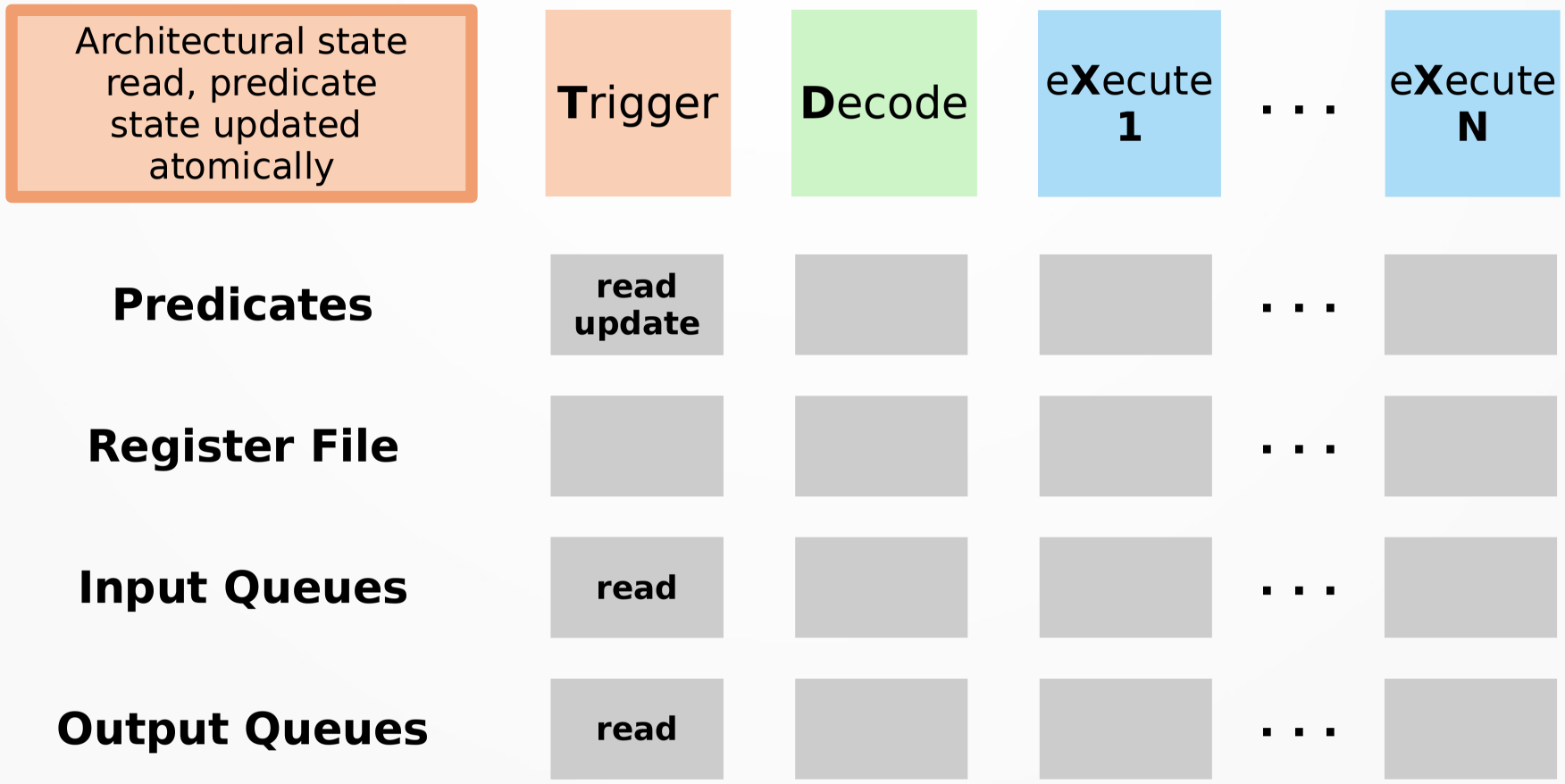
Outline

1. Motivation
2. Background
3. **Challenges of Triggered Pipelines**
4. Optimizations
5. Infrastructure and Methodology
6. Results and Conclusion

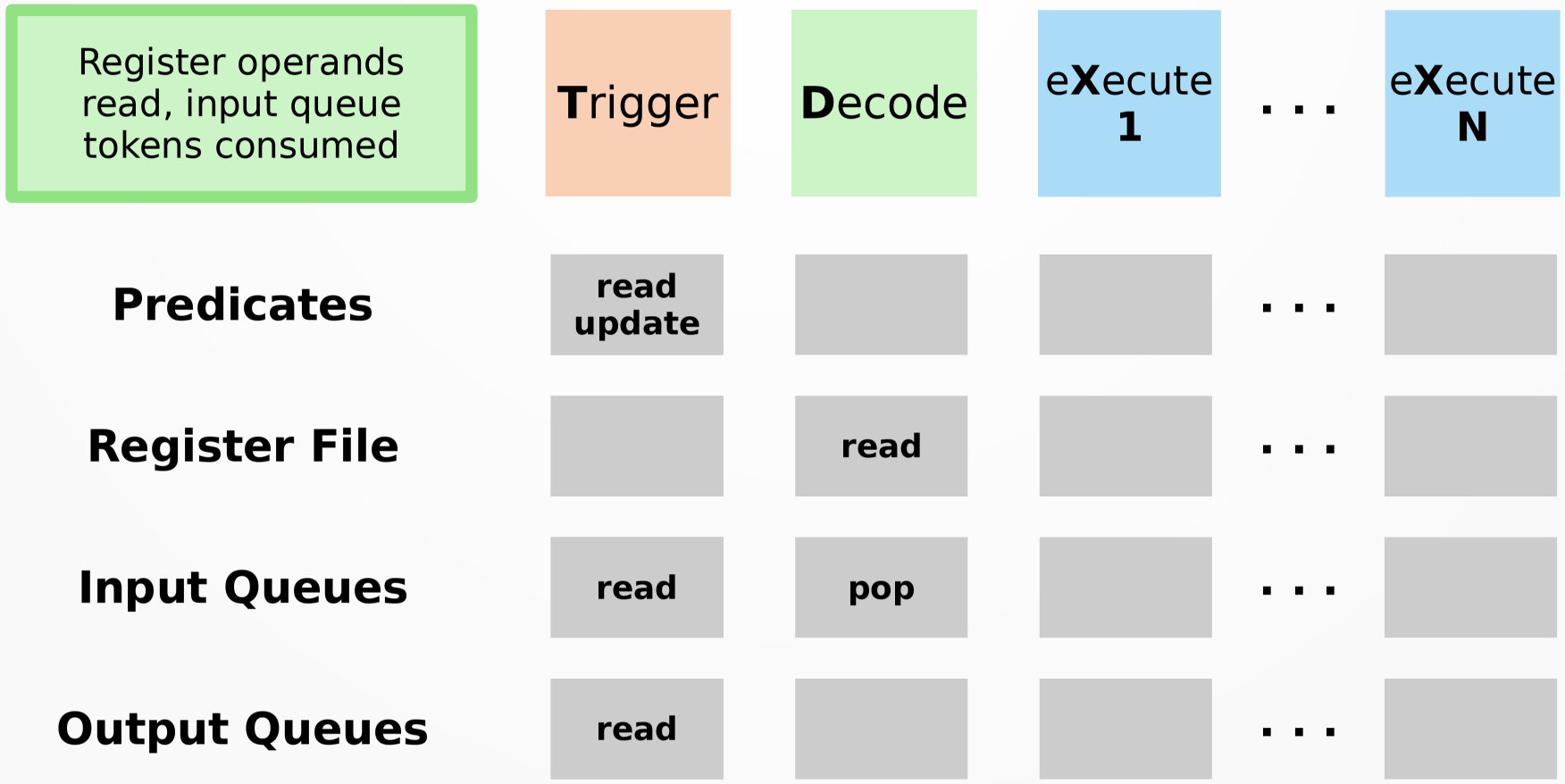
Pipeline Stages and State



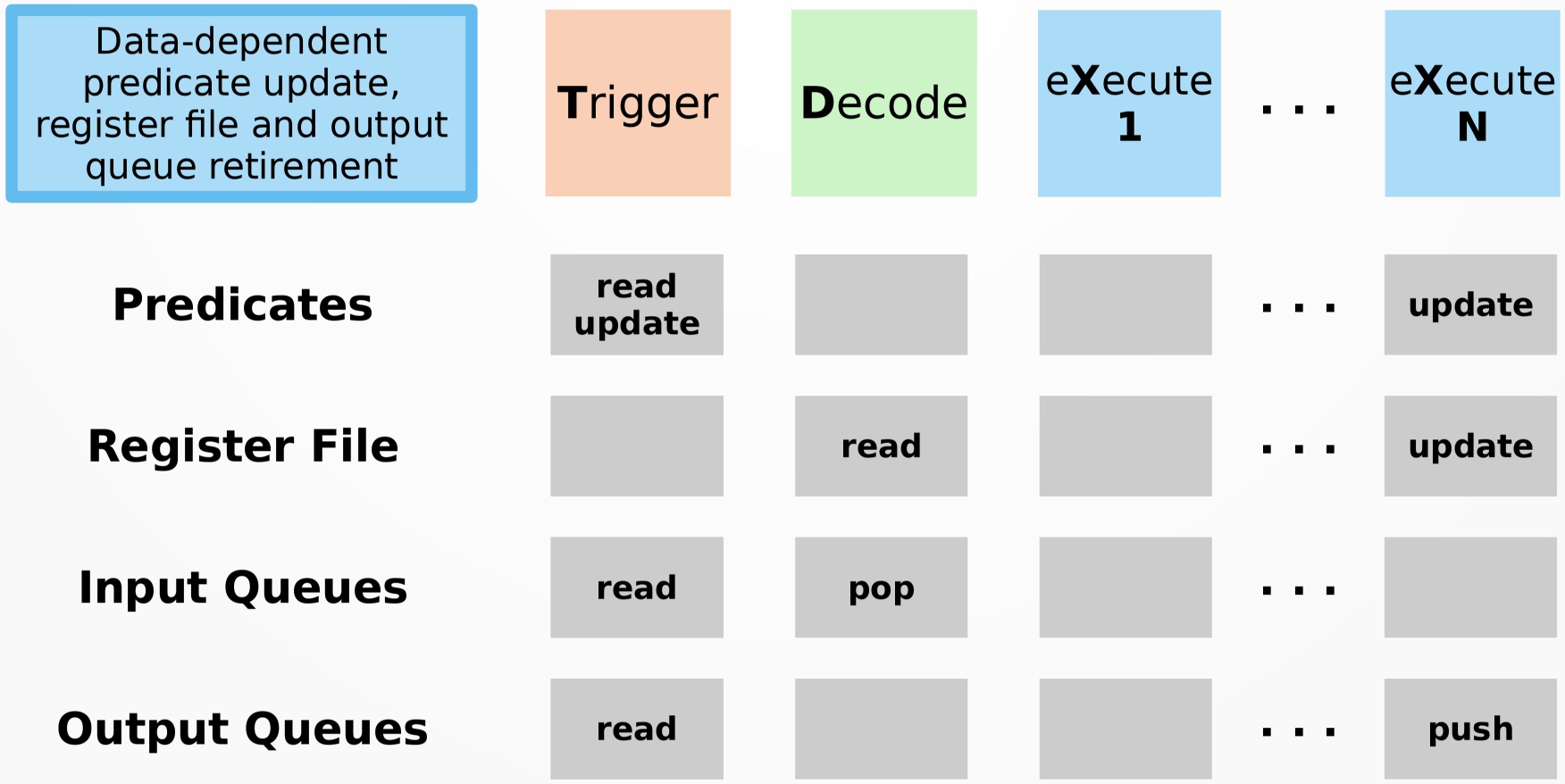
Pipeline Stages and State



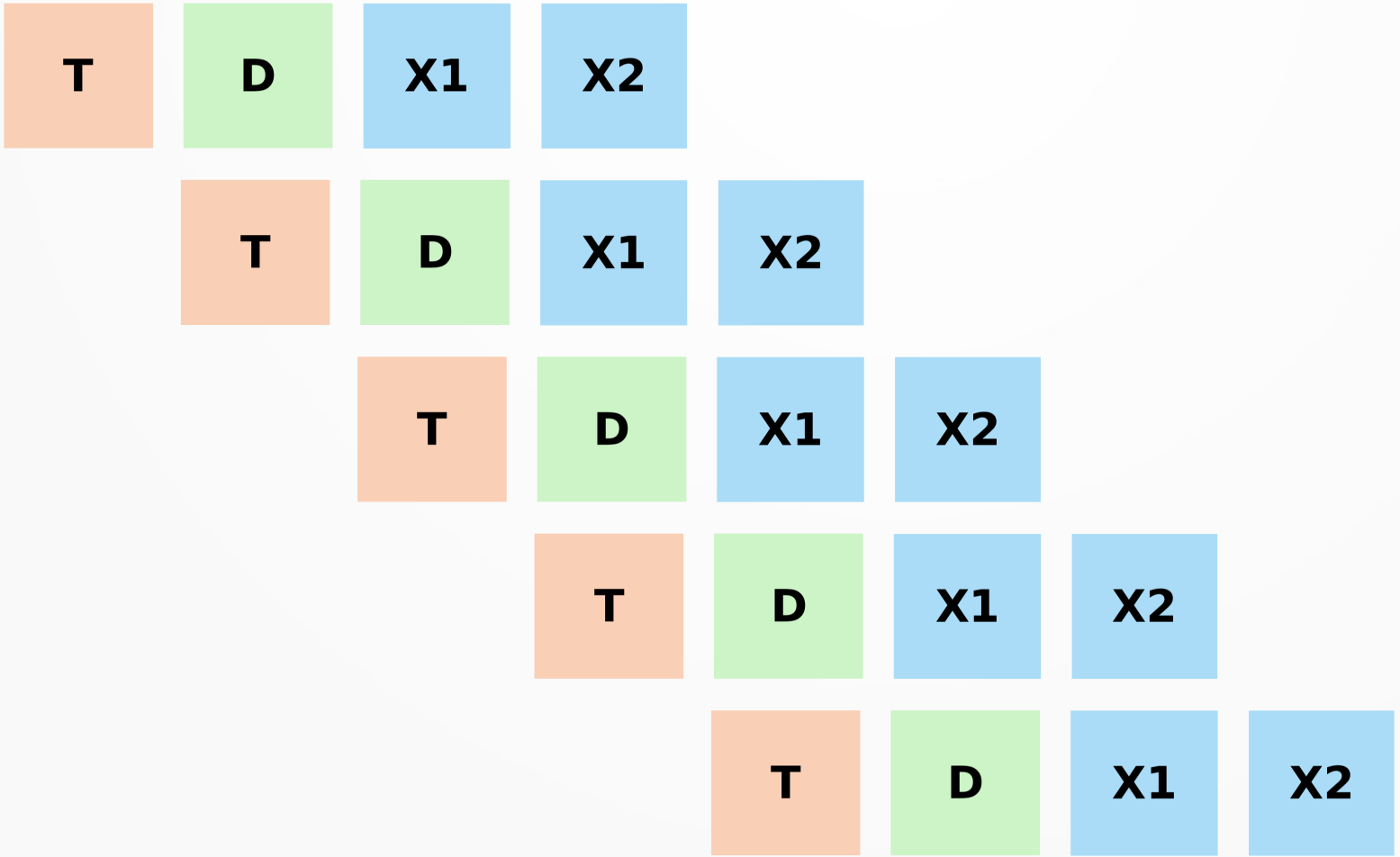
Pipeline Stages and State



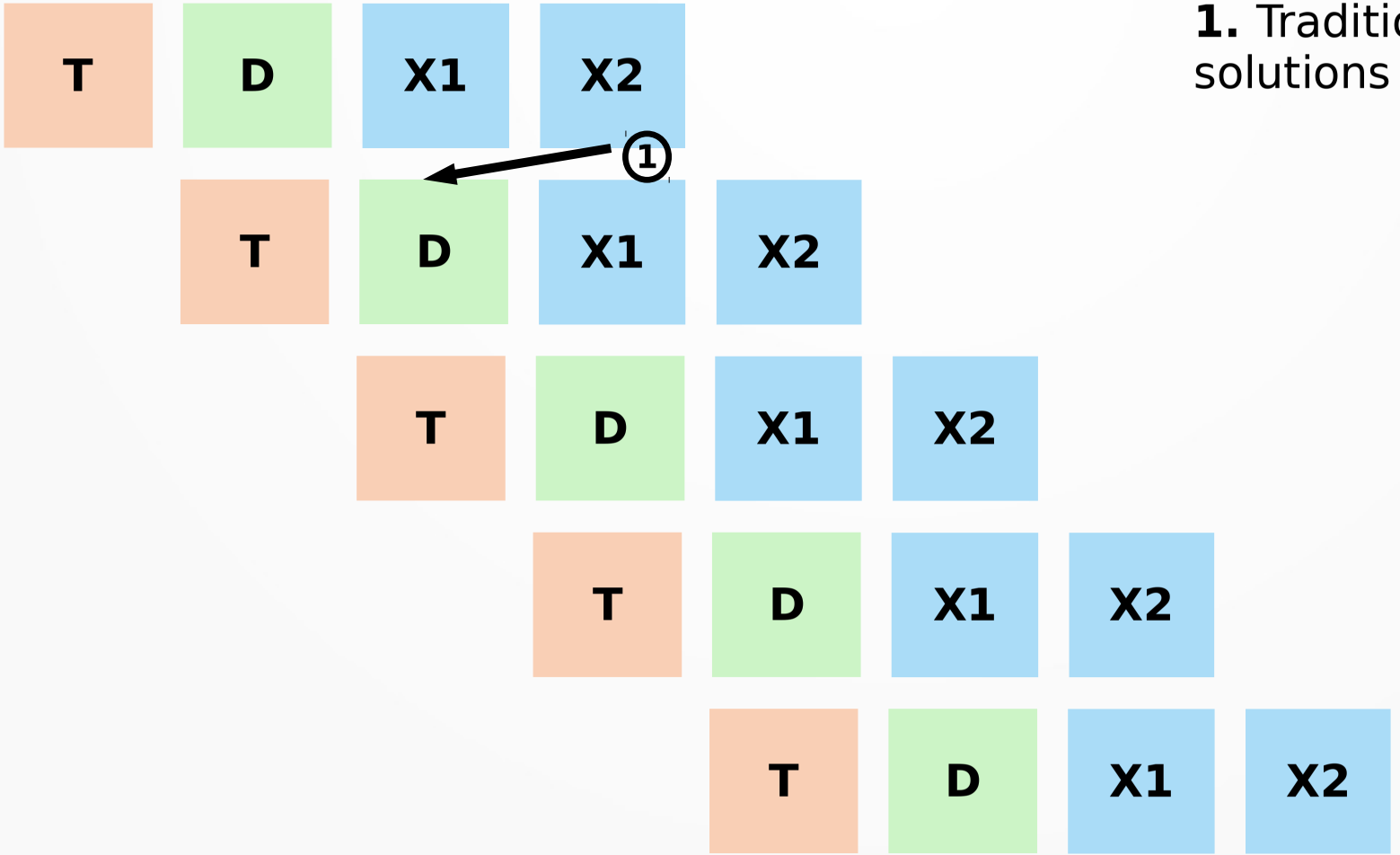
Pipeline Stages and State



Triggered Pipeline Hazards

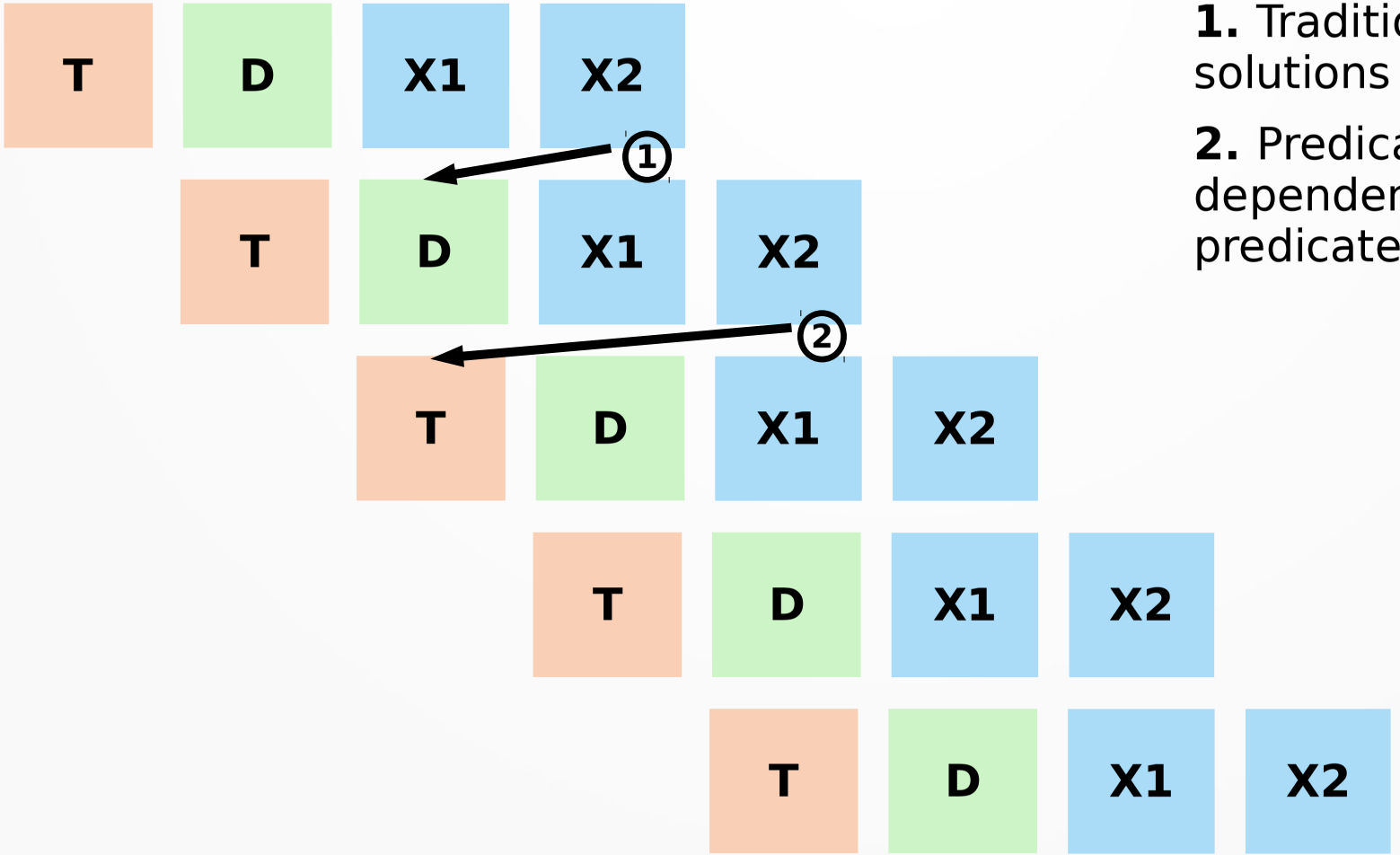


Triggered Pipeline Hazards



1. Traditional RAW hazard: solutions well known

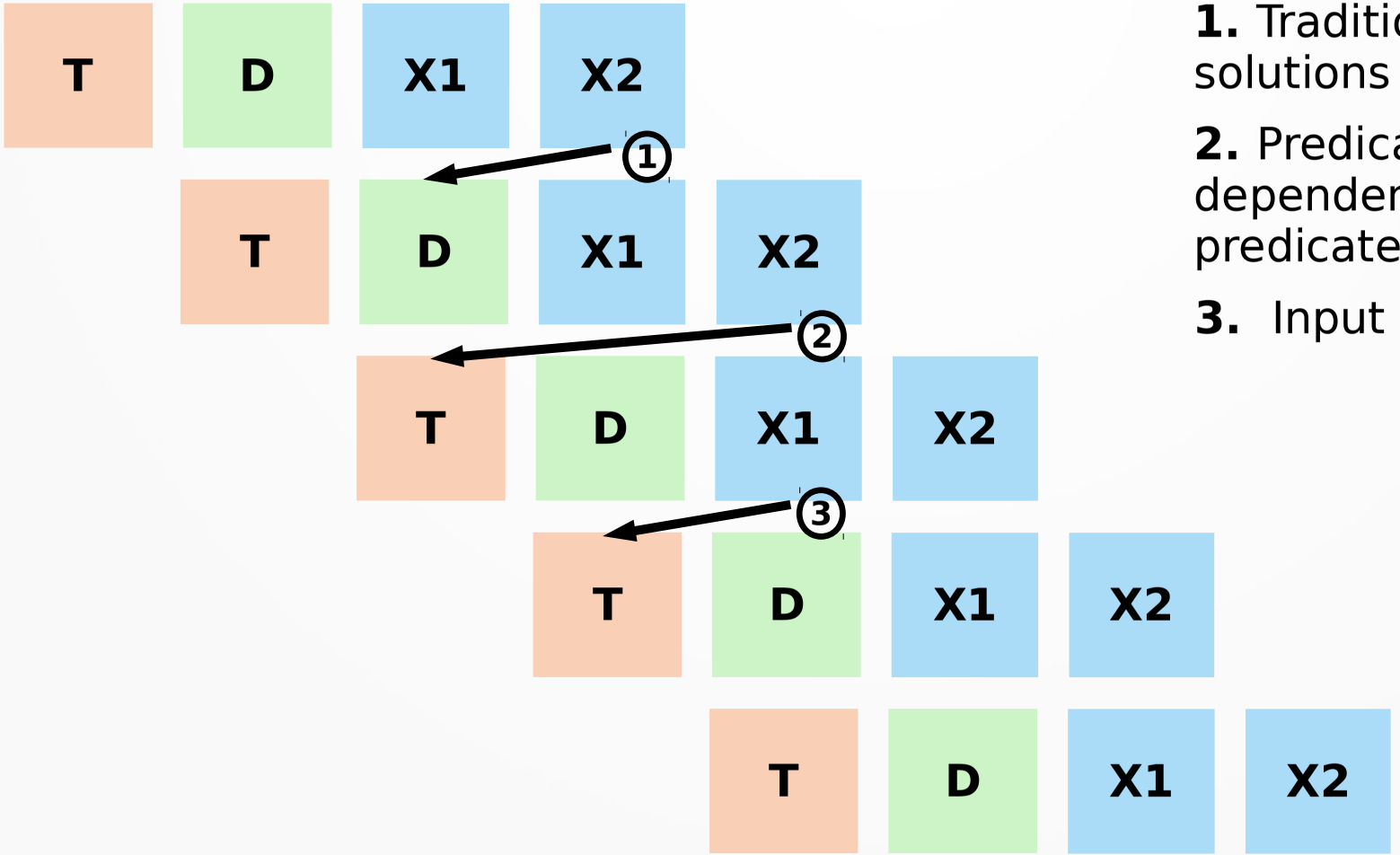
Triggered Pipeline Hazards



1. Traditional RAW hazard: solutions well known

2. Predicate hazard: data-dependent write to predicate register

Triggered Pipeline Hazards

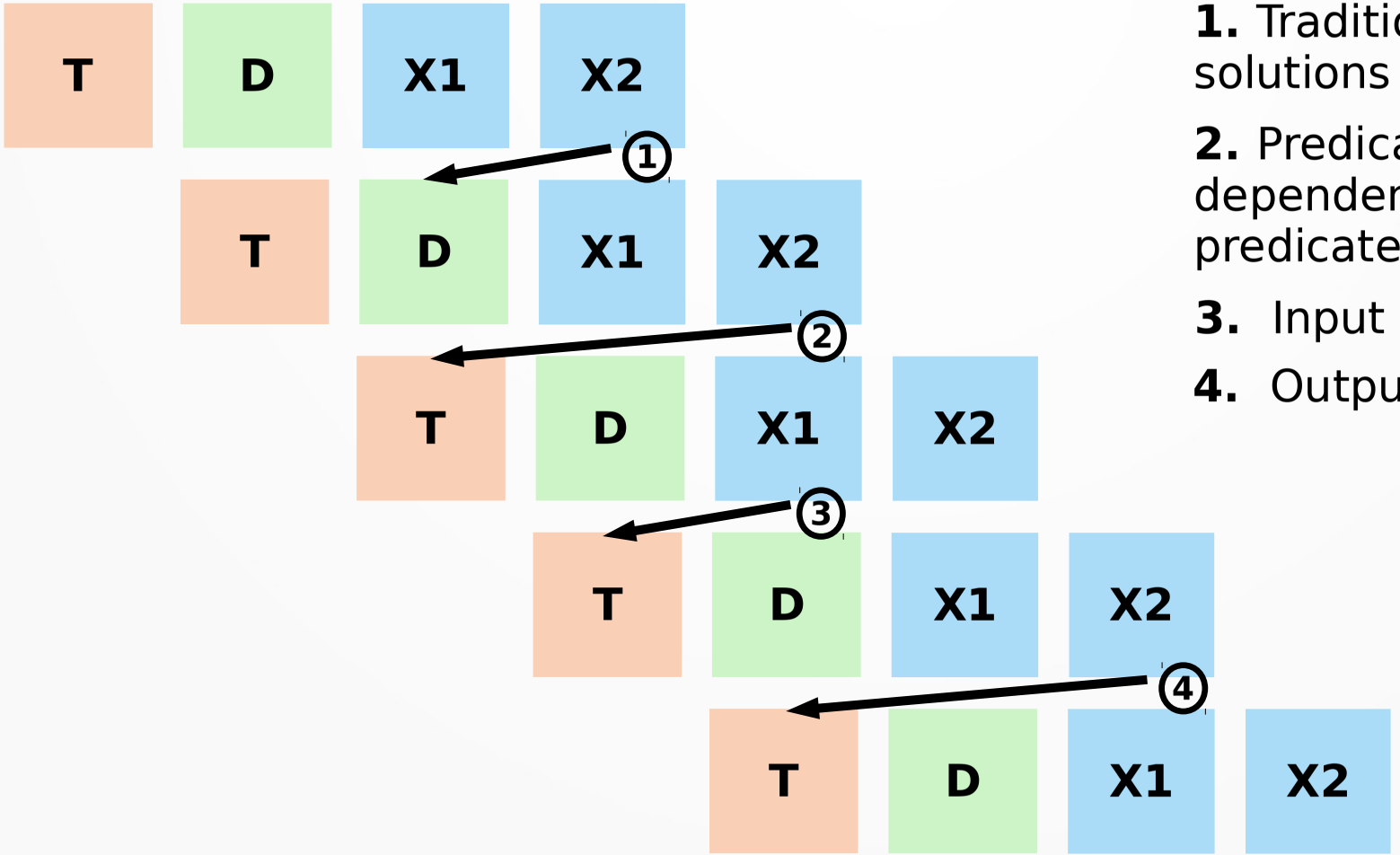


1. Traditional RAW hazard: solutions well known

2. Predicate hazard: data-dependent write to predicate register

3. Input queue pop

Triggered Pipeline Hazards



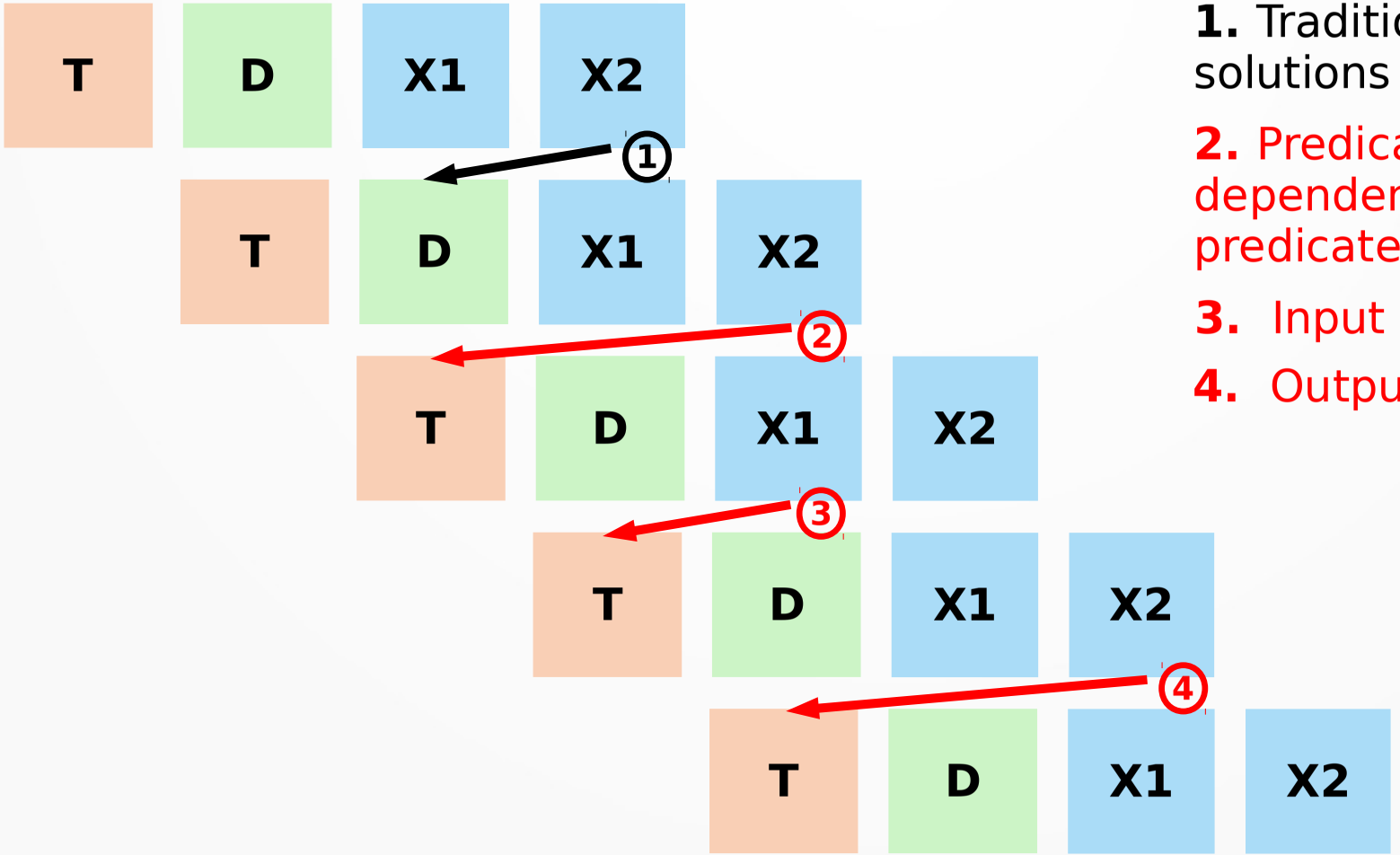
1. Traditional RAW hazard: solutions well known

2. Predicate hazard: data-dependent write to predicate register

3. Input queue pop

4. Output queue push

Triggered Pipeline Hazards



1. Traditional RAW hazard: solutions well known

2. Predicate hazard: data-dependent write to predicate register

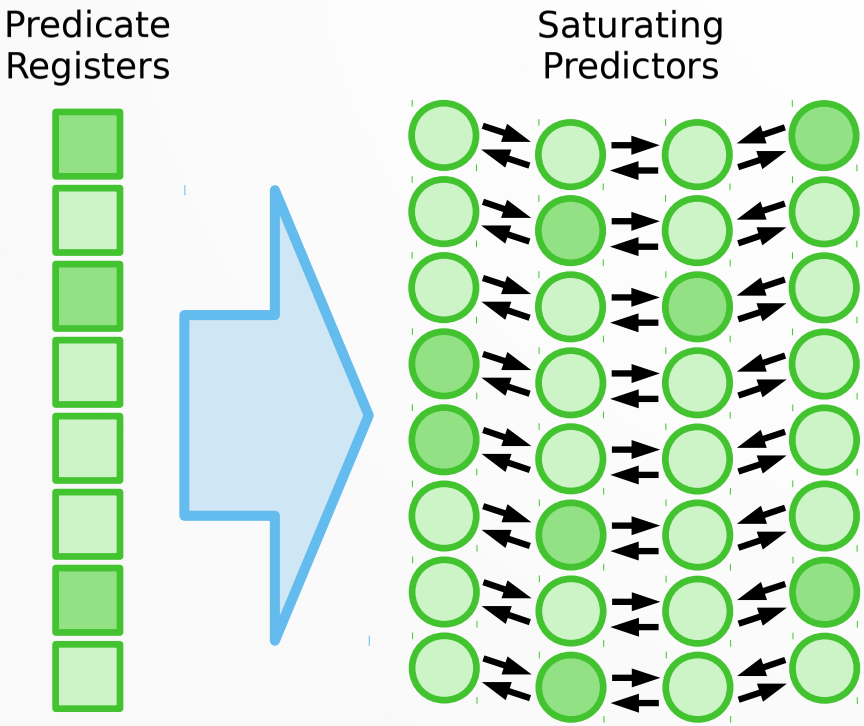
3. Input queue pop

4. Output queue push

Outline

1. Motivation
2. Background
3. Challenges of Triggered Pipelines
4. **Optimizations**
5. Infrastructure and Methodology
6. Results and Conclusion

Predicate Prediction

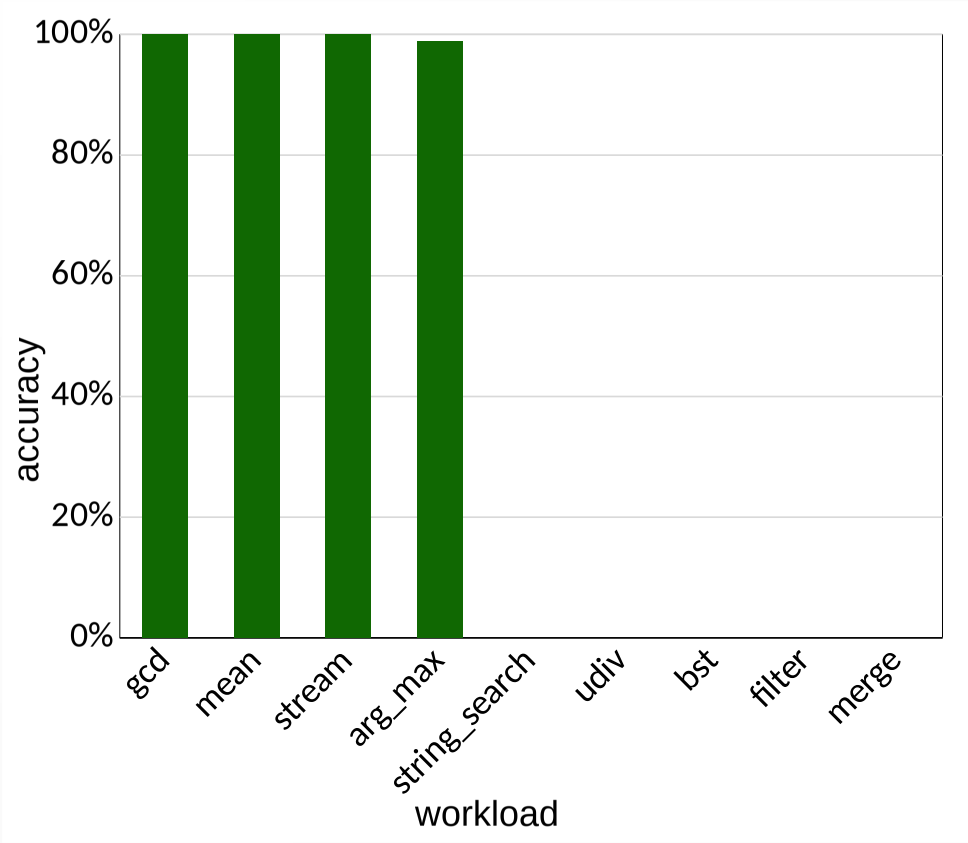


- Speculate on predicate datapath result
- Forbid updates to architectural state during speculation
  - Pending dequeue
  - Memory write
- Rollback via pipeline flush

Predicate Prediction

- Surprisingly effective
- Predicate indices encode *branch locality*

Predictable primary control structure (loop-like or low-entropy)

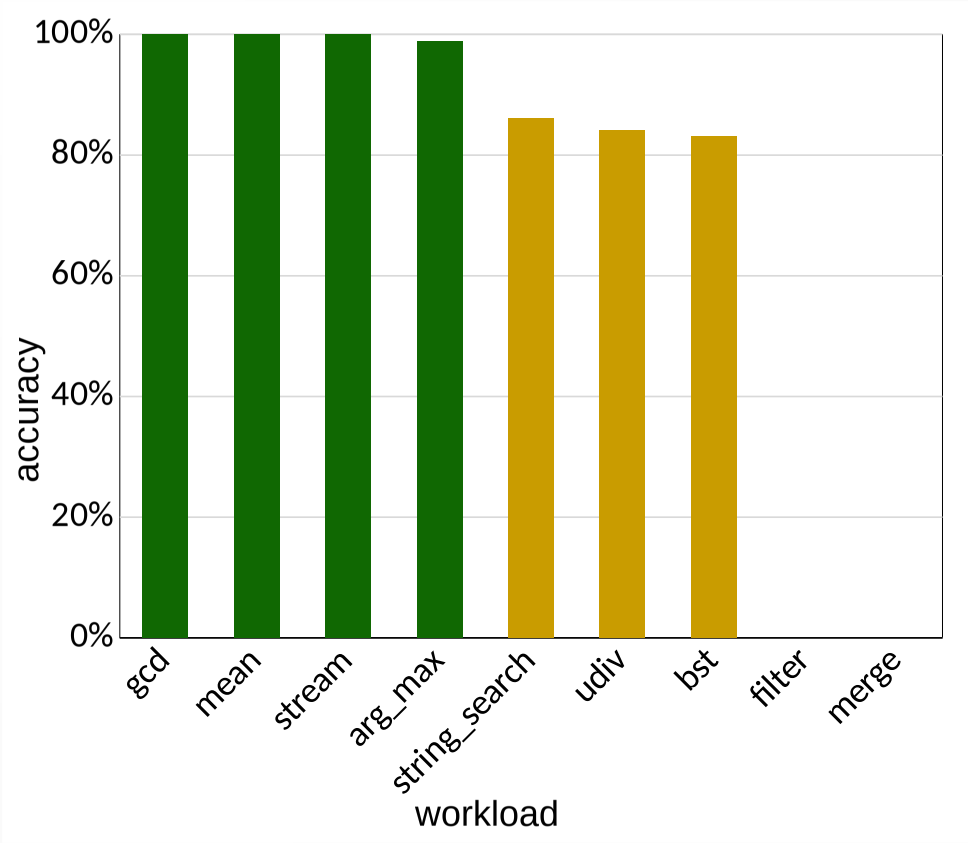


Predicate Prediction

- Surprisingly effective
- Predicate indices encode *branch locality*

Predictable primary control structure (loop-like or low-entropy)

Unpredictable branch nested within loop



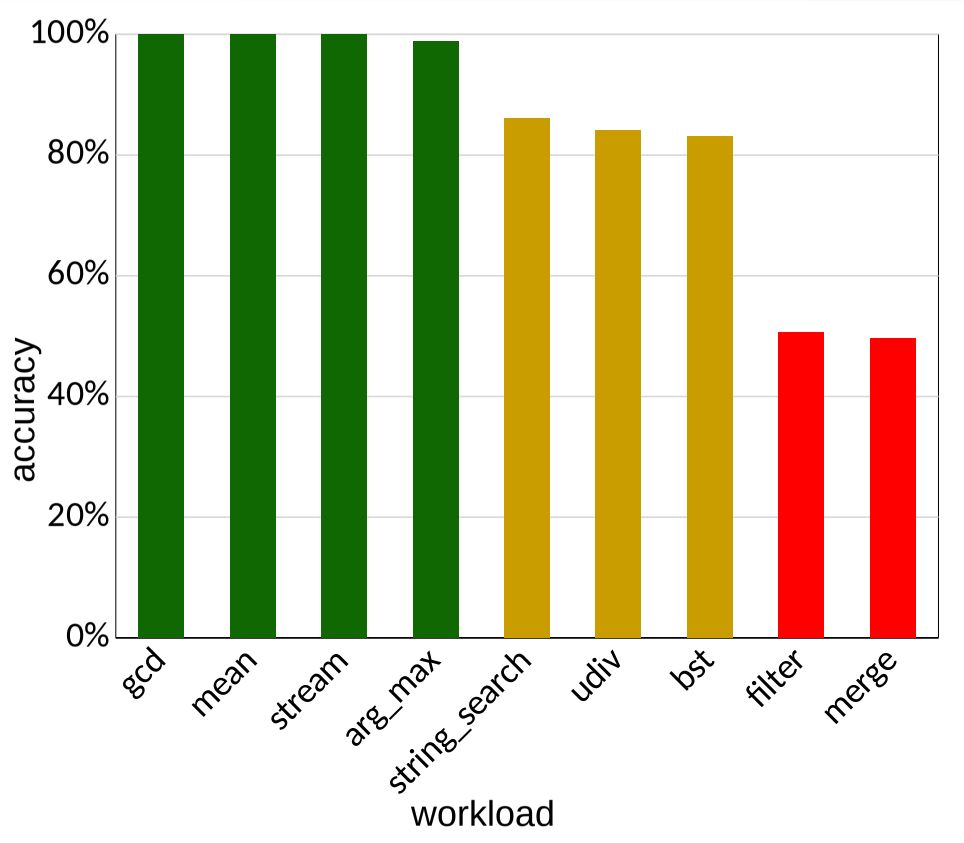
Predicate Prediction

- Surprisingly effective
- Predicate indices encode *branch locality*

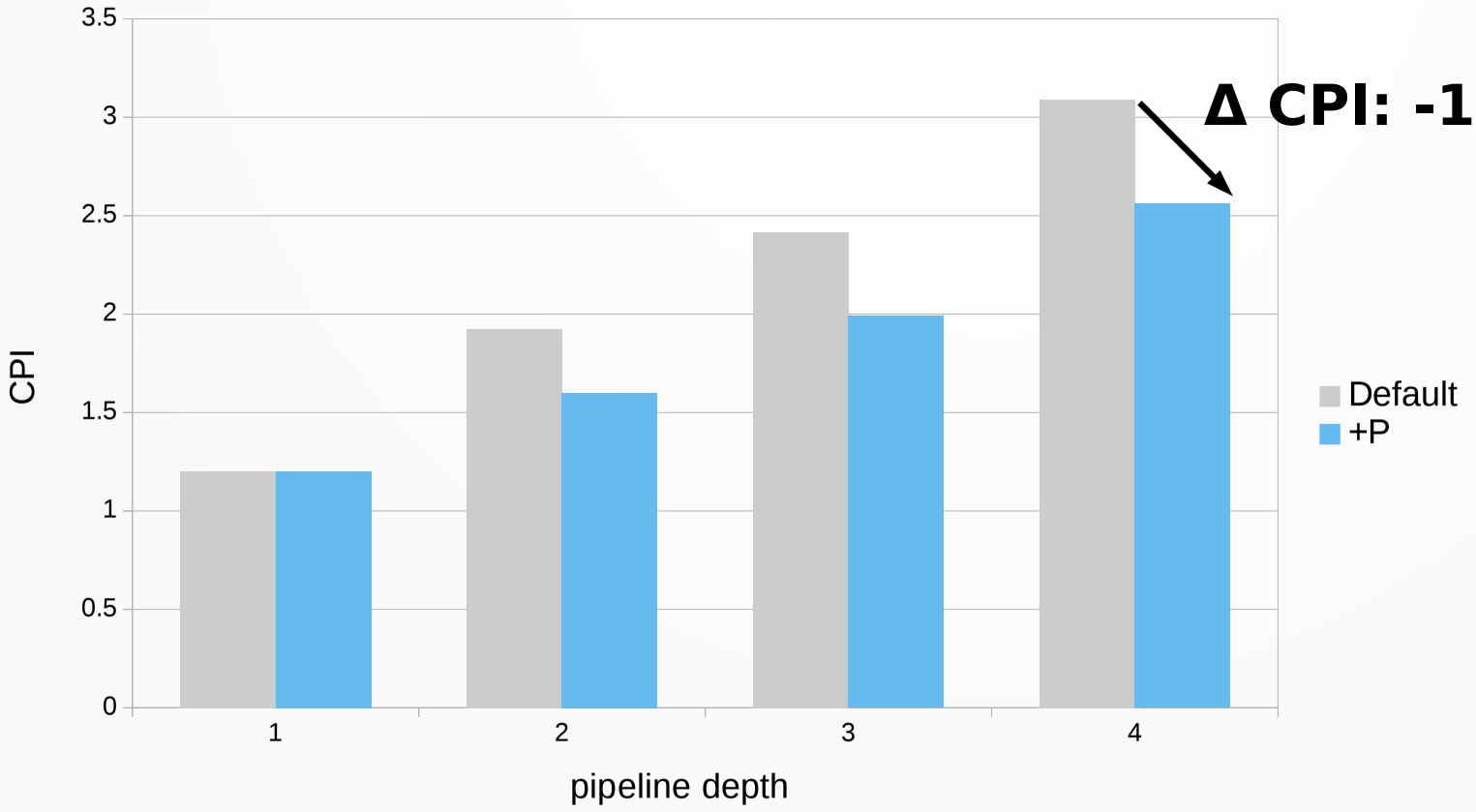
Predictable primary control structure (loop-like or low-entropy)

Unpredictable branch nested within loop

High-entropy data-dependent control flow



Predicate Prediction Performance



+0.5% Area
+7.0% Power
53.6 → 64.3 FO4

Effective Queue Status

- Processor pipelines *are* FIFO queues
- Trigger stage needs an up-to-date view of the world:
  - input queues
  - **the pipeline itself as a queue**
  - output queues

Effective Queue Status

Effective Queue Status

Trigger stage must account for:

- input queue empty state and head semantic tag information
- output queue full state

Effective Queue Status

Problem: Dequeueing from an input queue has a single cycle of latency

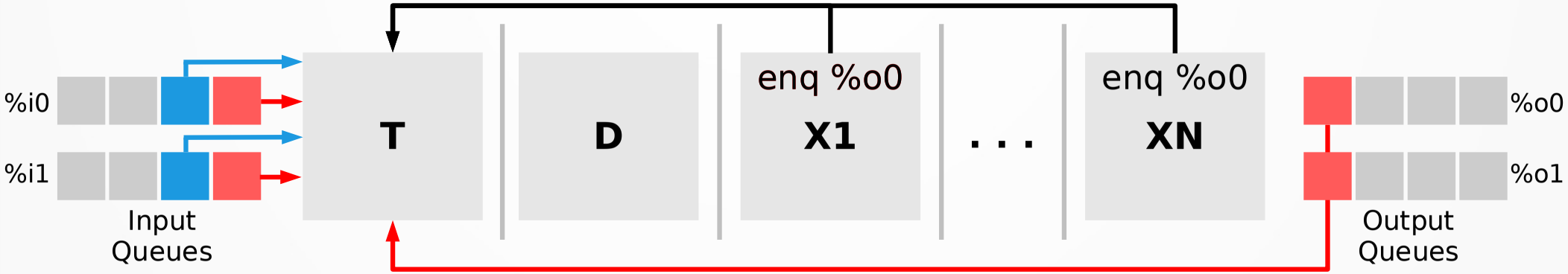
Effective Queue Status

Solution: expose head and “neck” of queue and calculate if empty pending the dequeue

Effective Queue Status

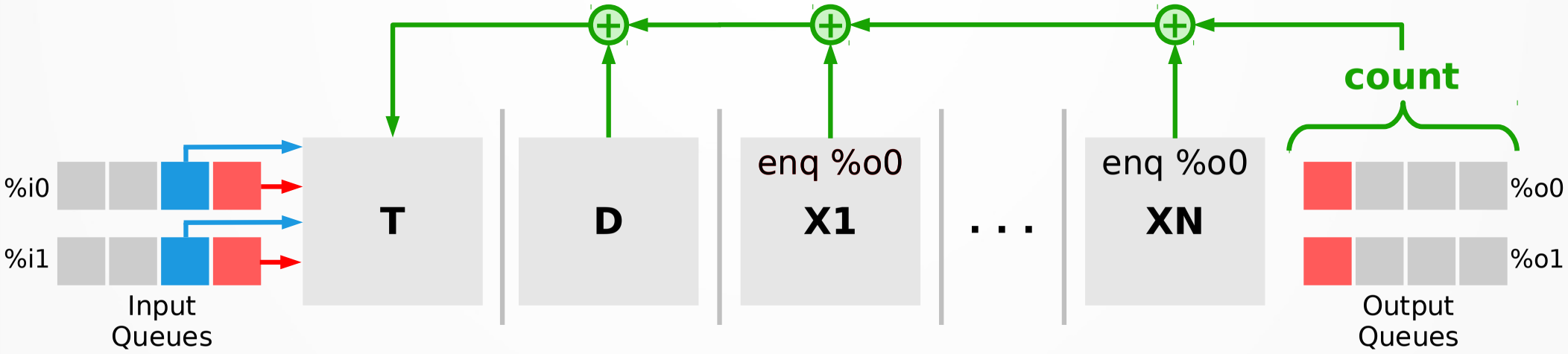
Solution: expose head and “neck” of queue and calculate if empty pending the dequeue

Effective Queue Status



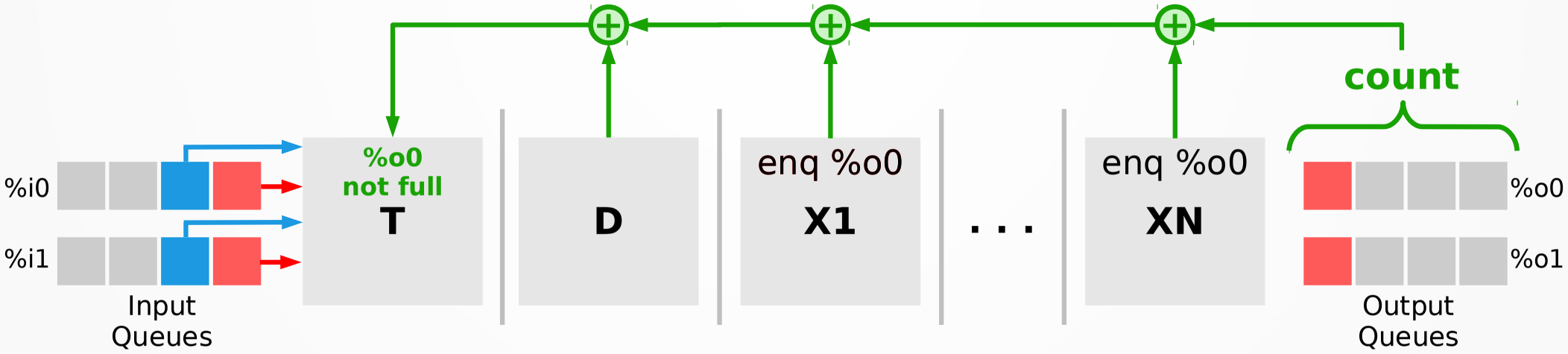
Problem: In-flight enqueues can effect the future state of the output queues

Effective Queue Status



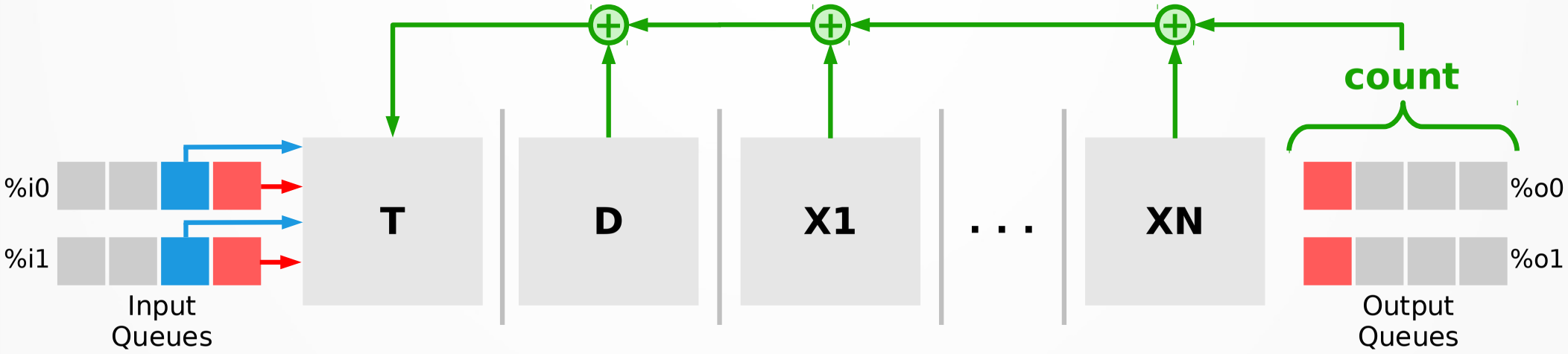
Solution: expose output queue counts, inspect the pipeline for in-flight enqueues and calculate effective status for trigger stage

Effective Queue Status



Solution: expose output queue counts, inspect the pipeline for in-flight enqueues and calculate effective status for trigger stage

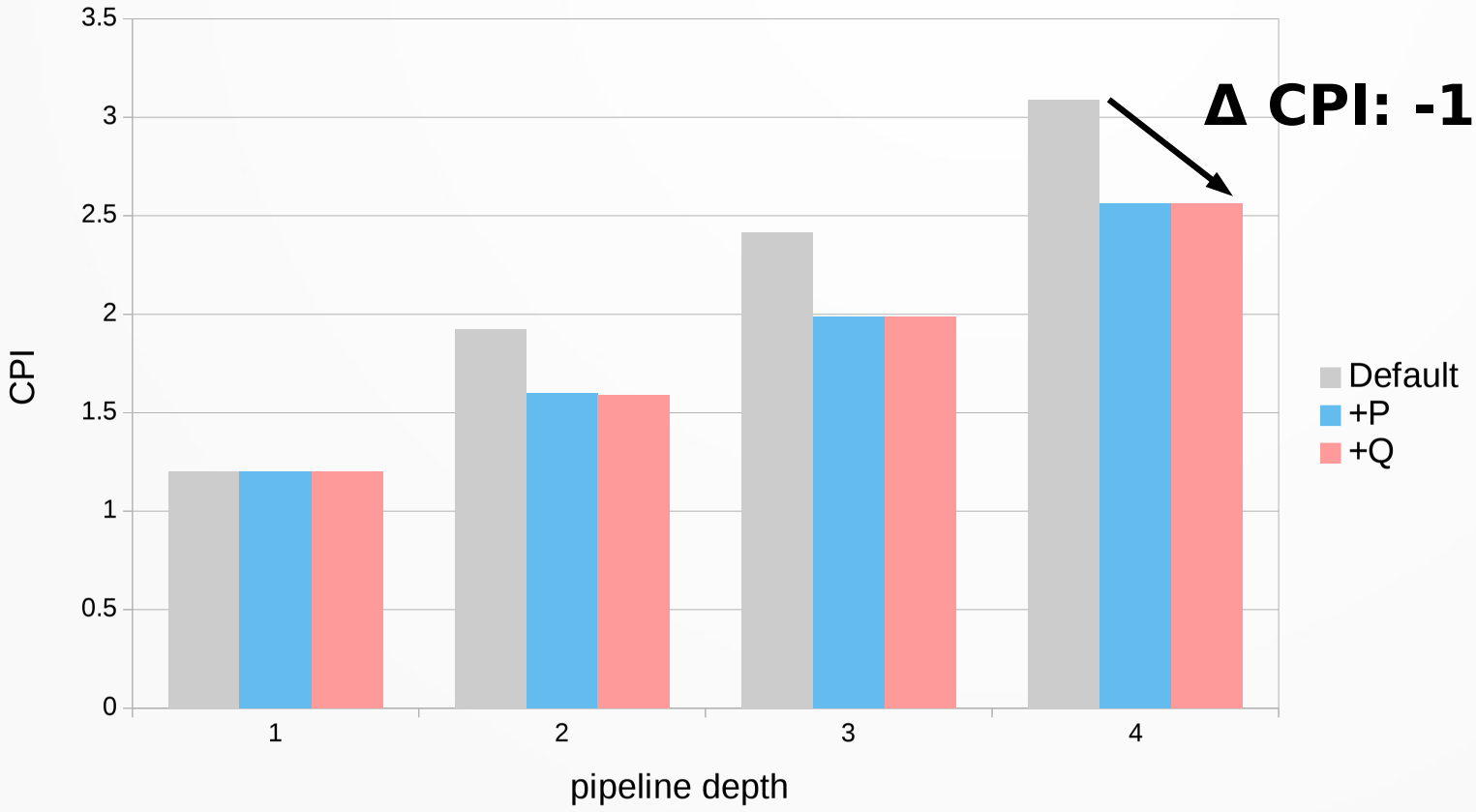
Effective Queue Status



Effect of optimizations:

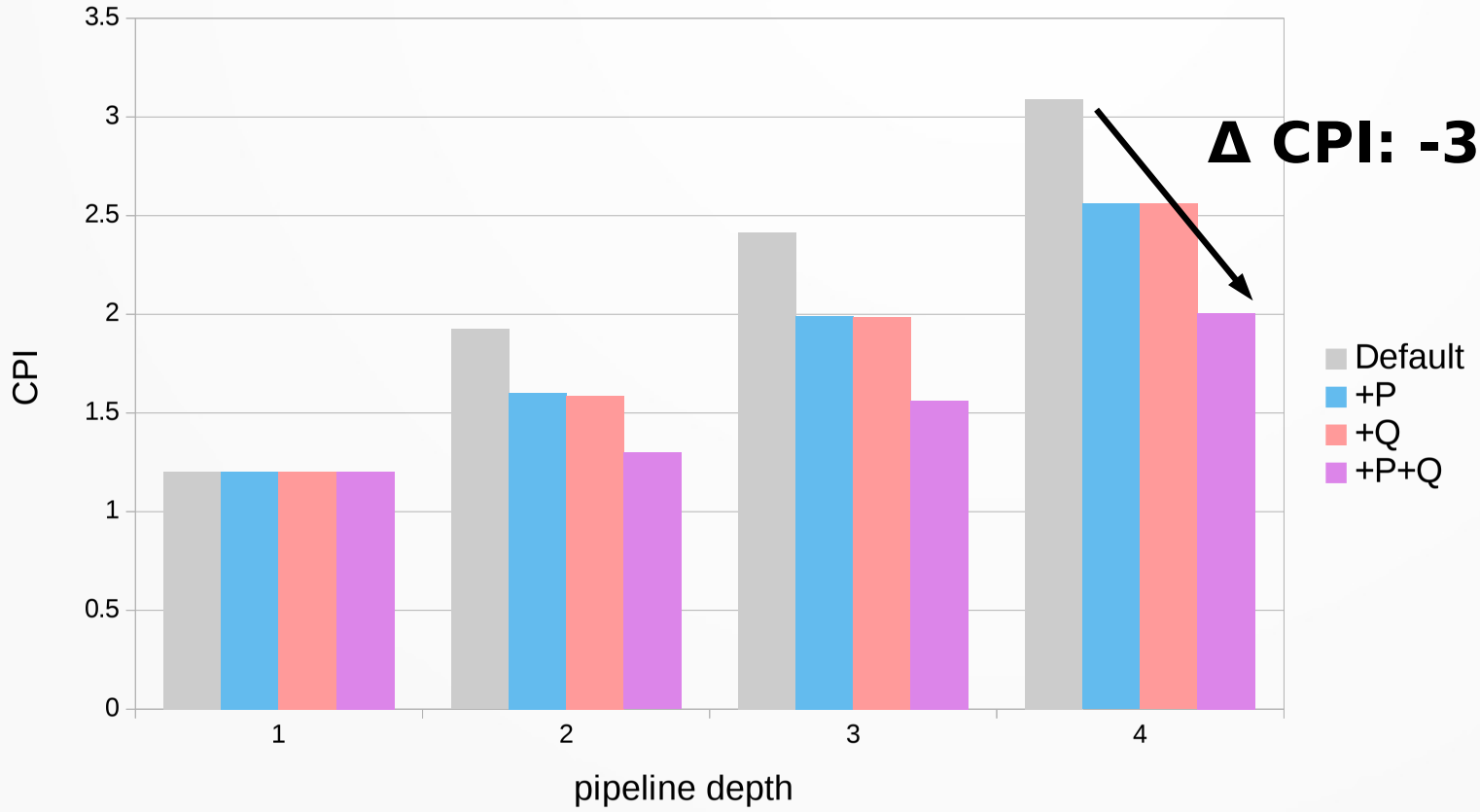
- Maintain high instruction throughput when:
  - *consuming* data tokens
  - *producing* data tokens

Effective Queue Status Performance



+2.0% Area
+0% Power
No Effect on Timing

Combined Effect of Optimizations

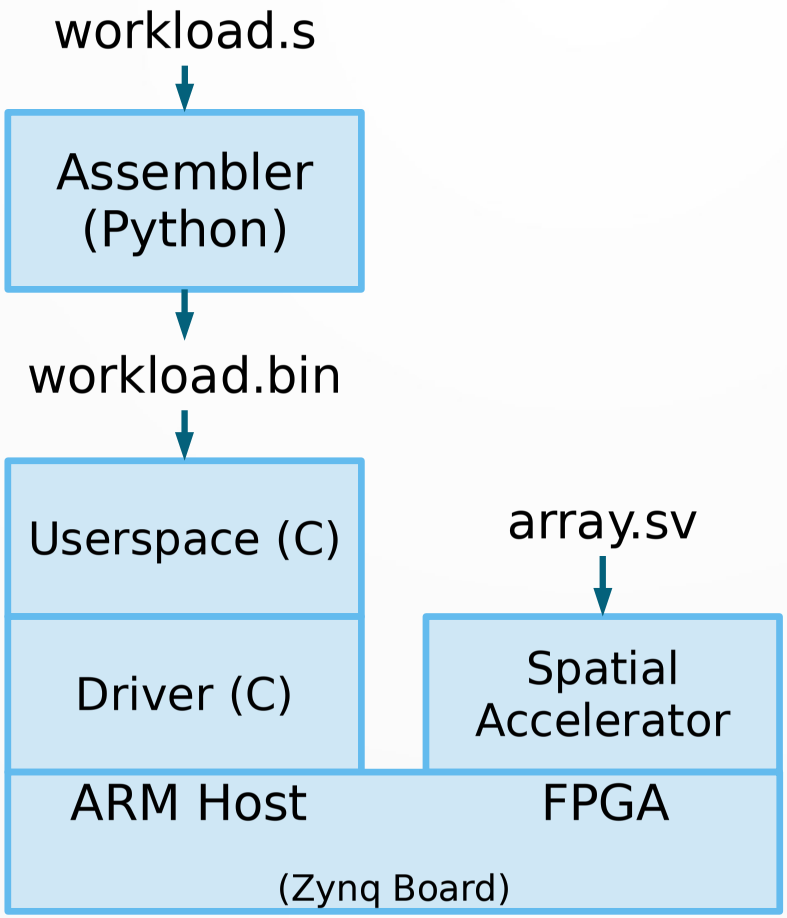


Effect on CPI is Additive

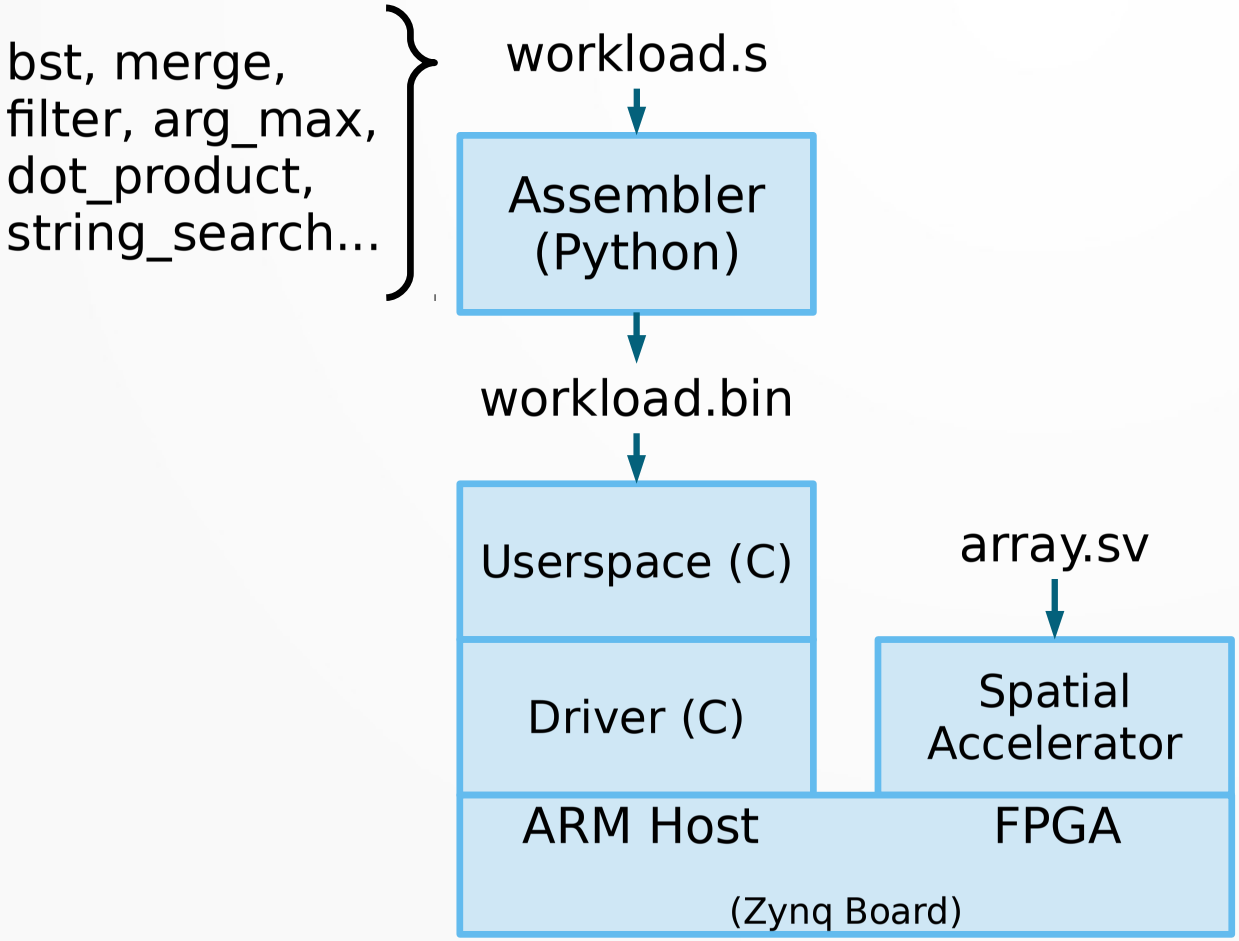
Outline

1. Motivation
2. Background
3. Challenges of Triggered Pipelines
4. Optimizations
5. **Infrastructure and Methodology**
6. Results and Conclusion

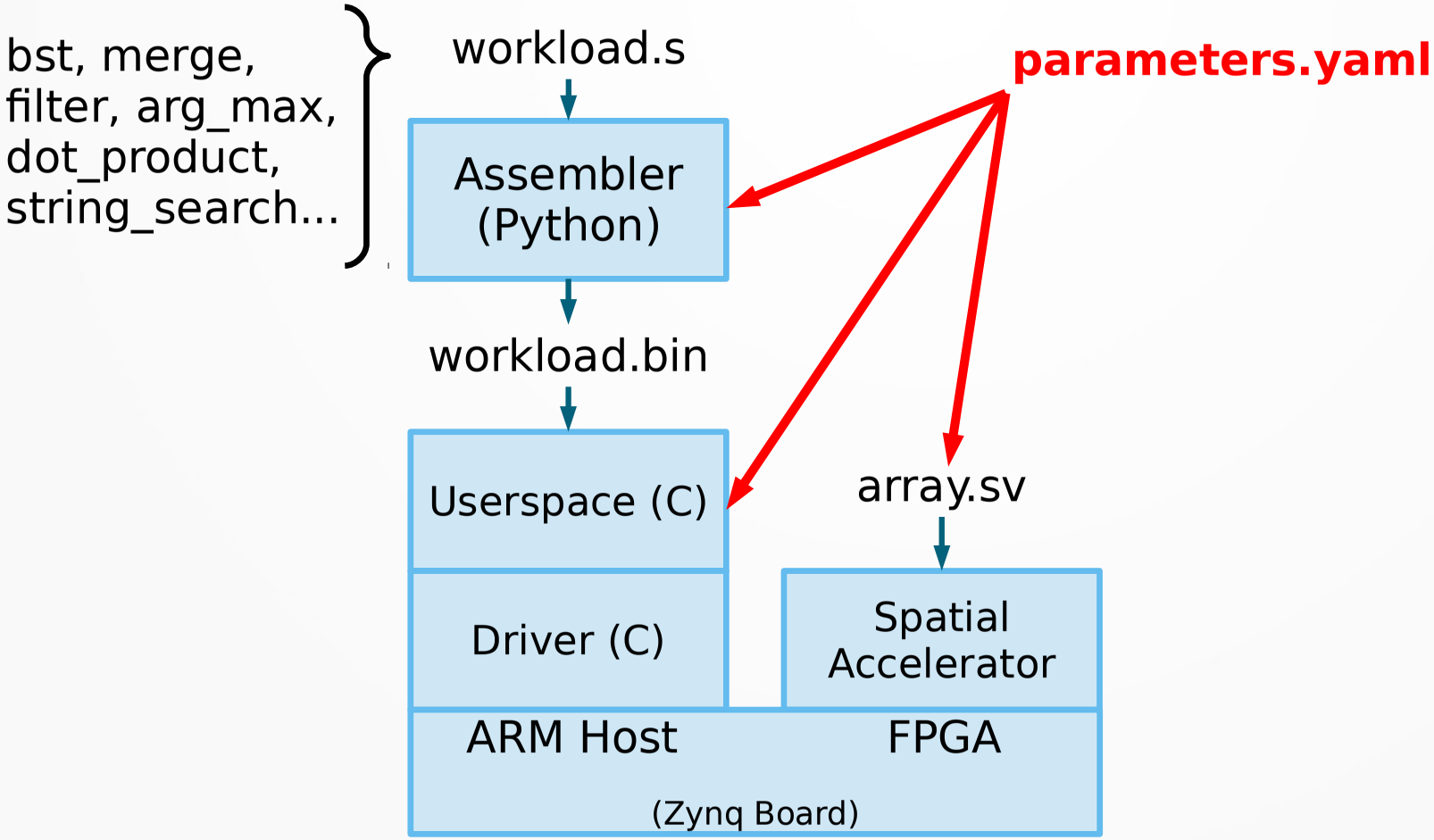
Experimental Infrastructure



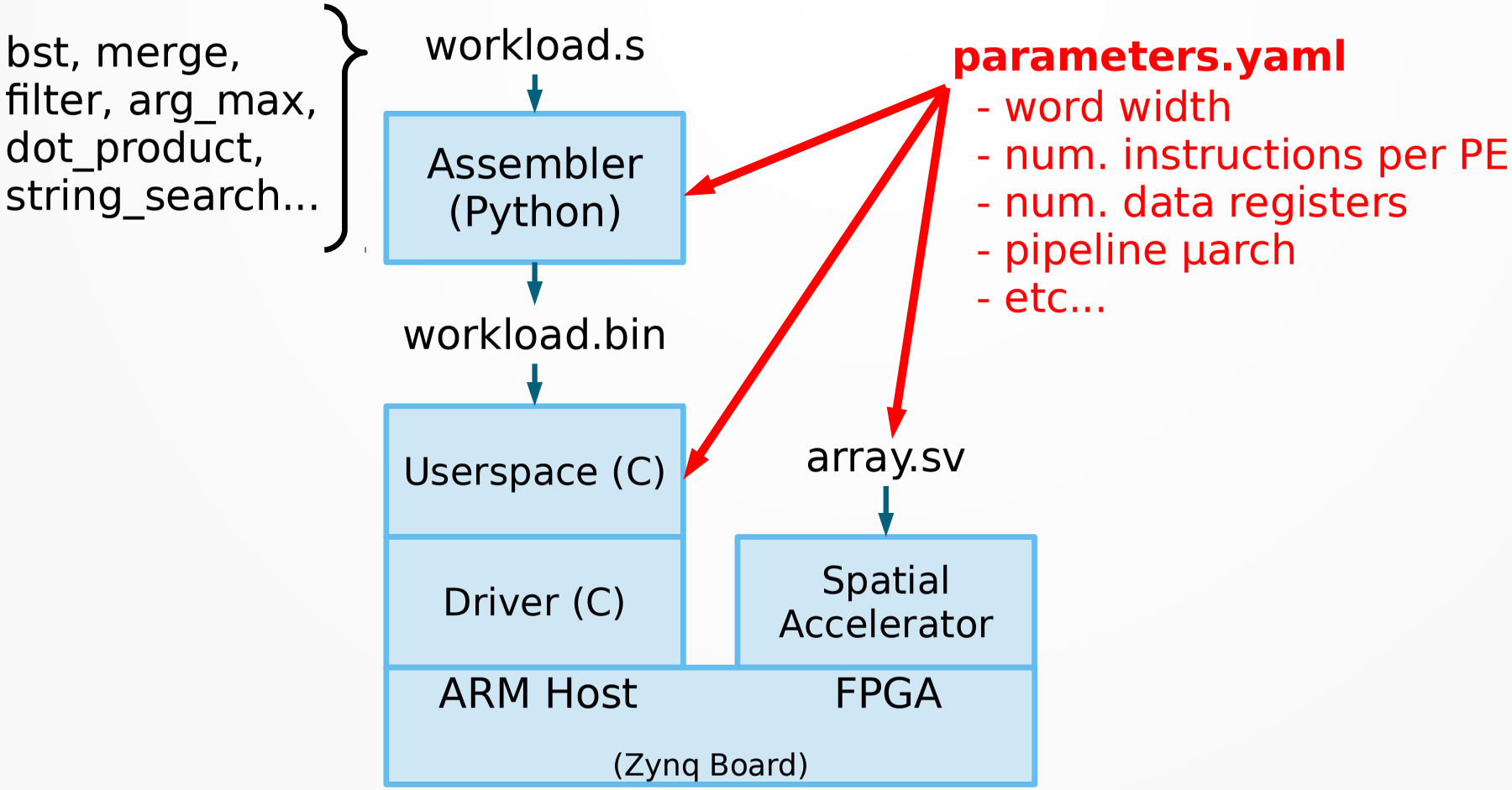
Experimental Infrastructure



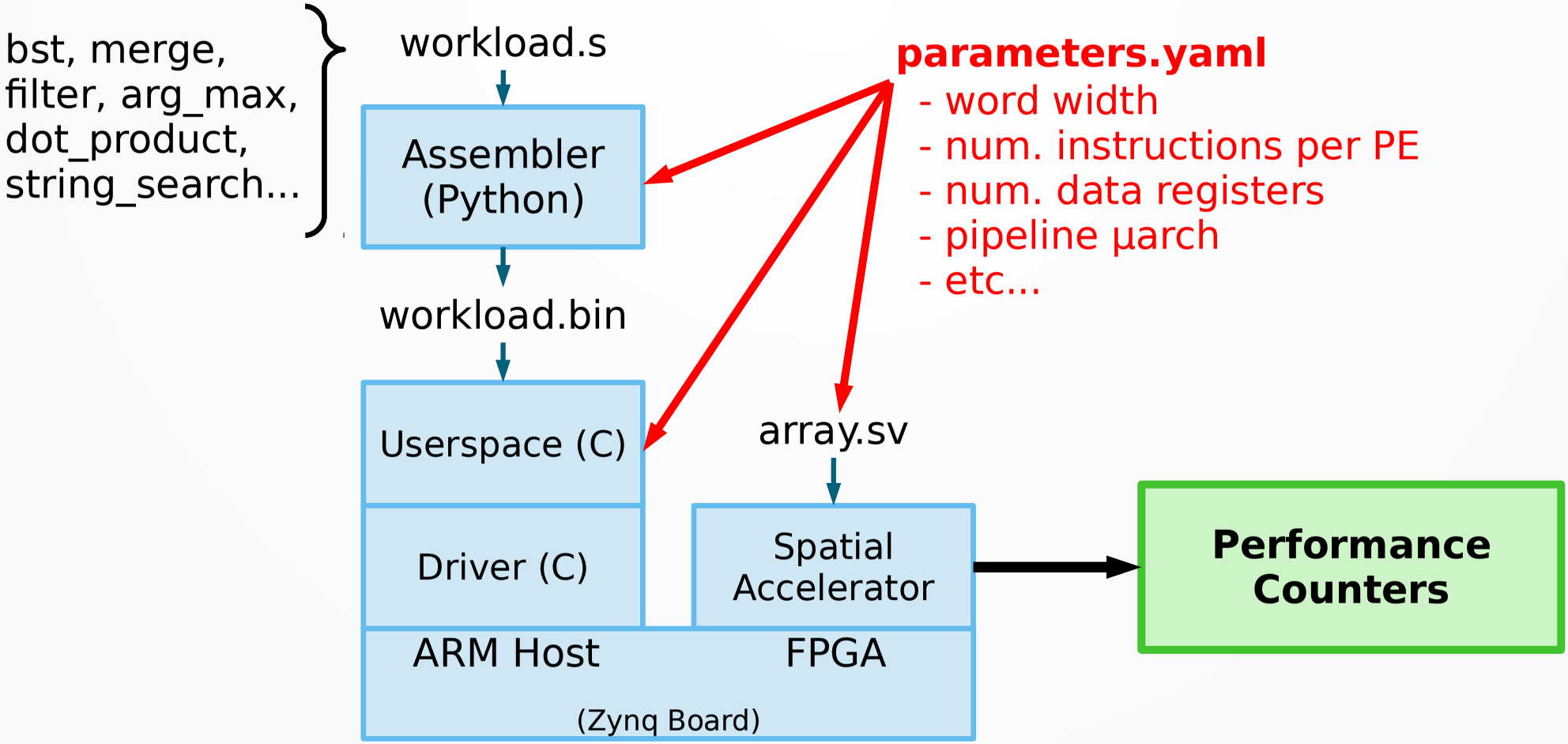
Experimental Infrastructure



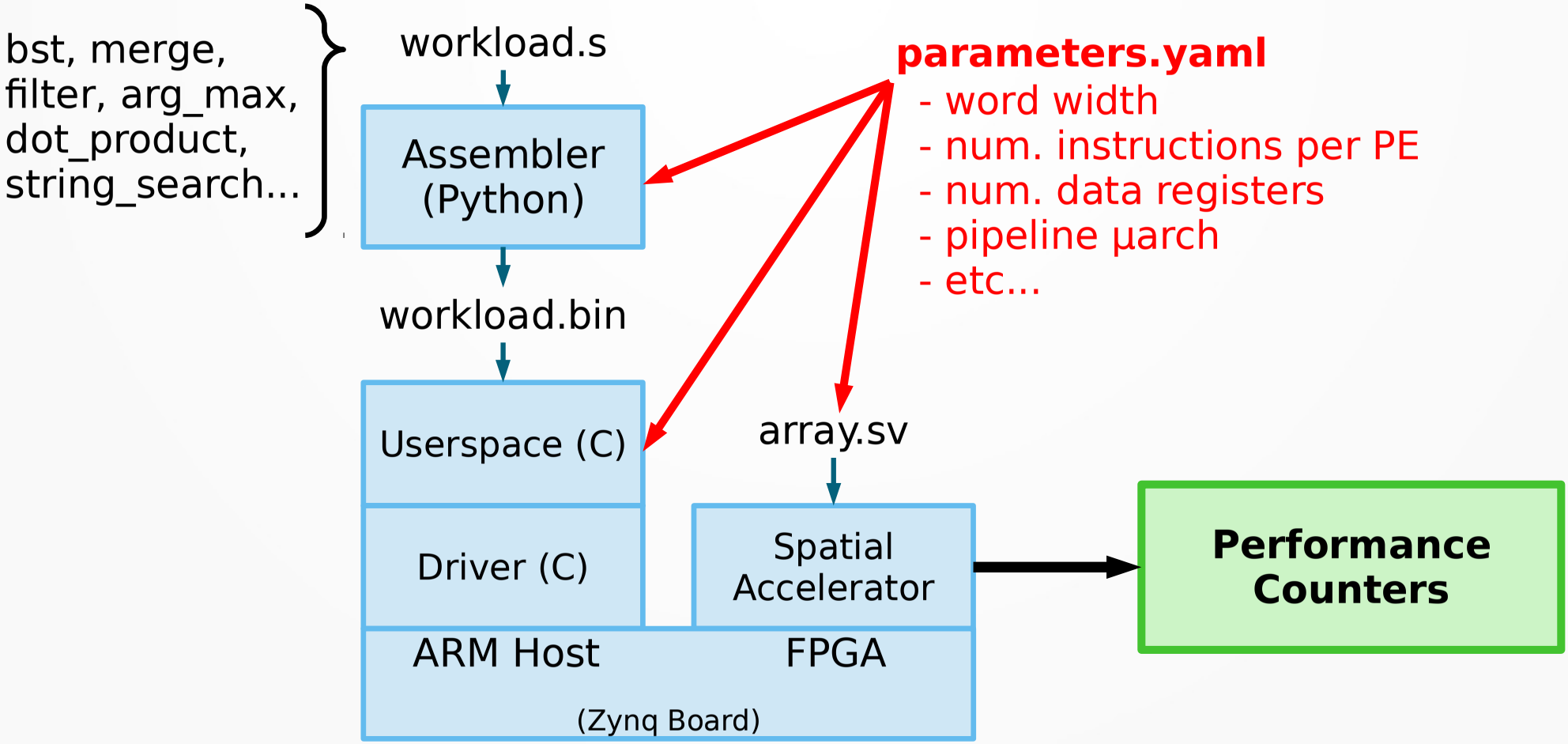
Experimental Infrastructure



Experimental Infrastructure



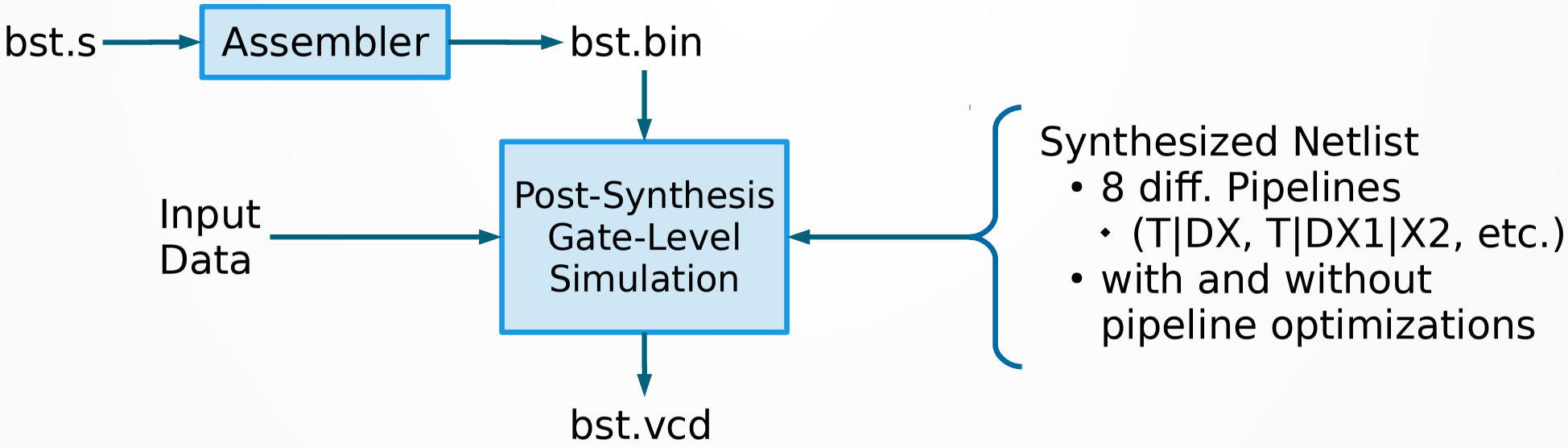
Experimental Infrastructure



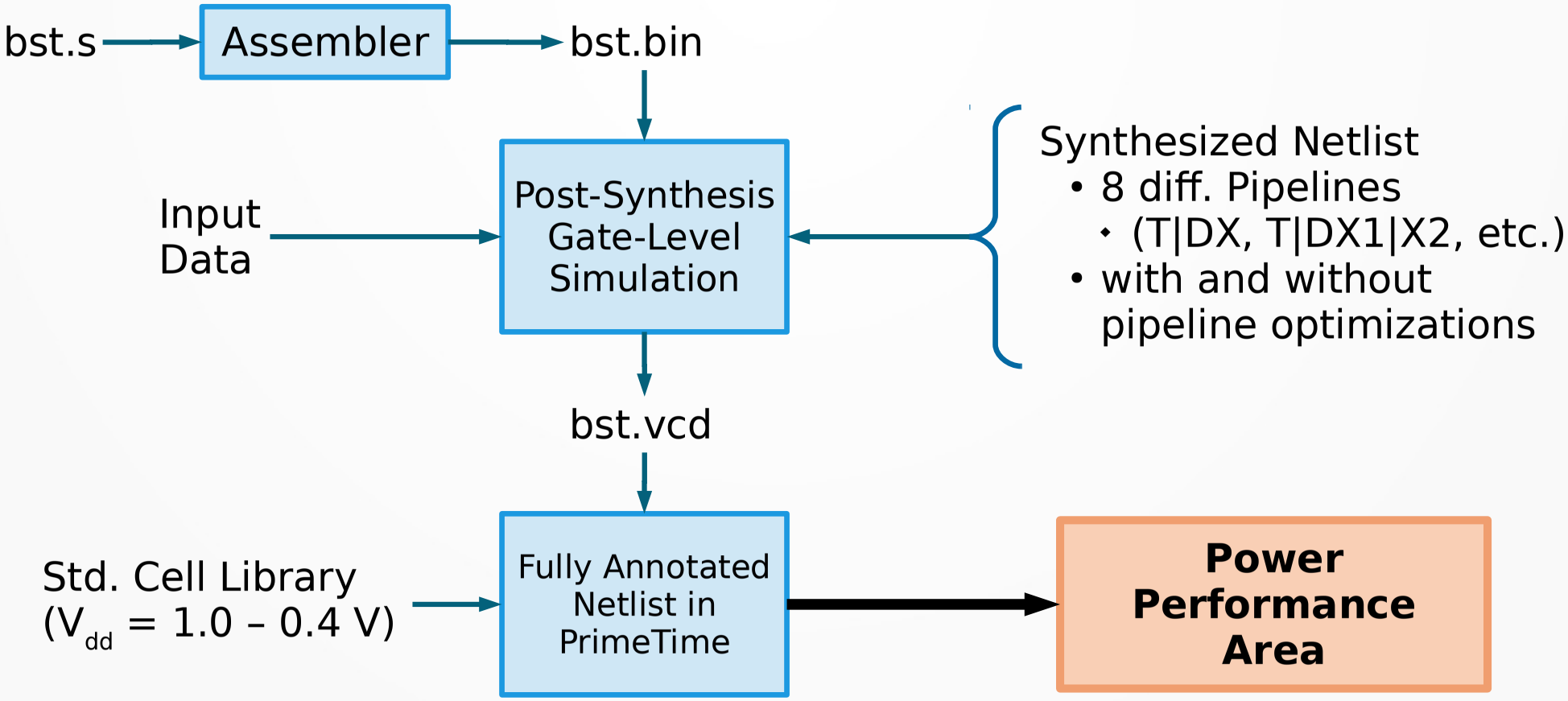
Available: github.com/arcade-lab/tia-infrastructure

Evaluation Methodology

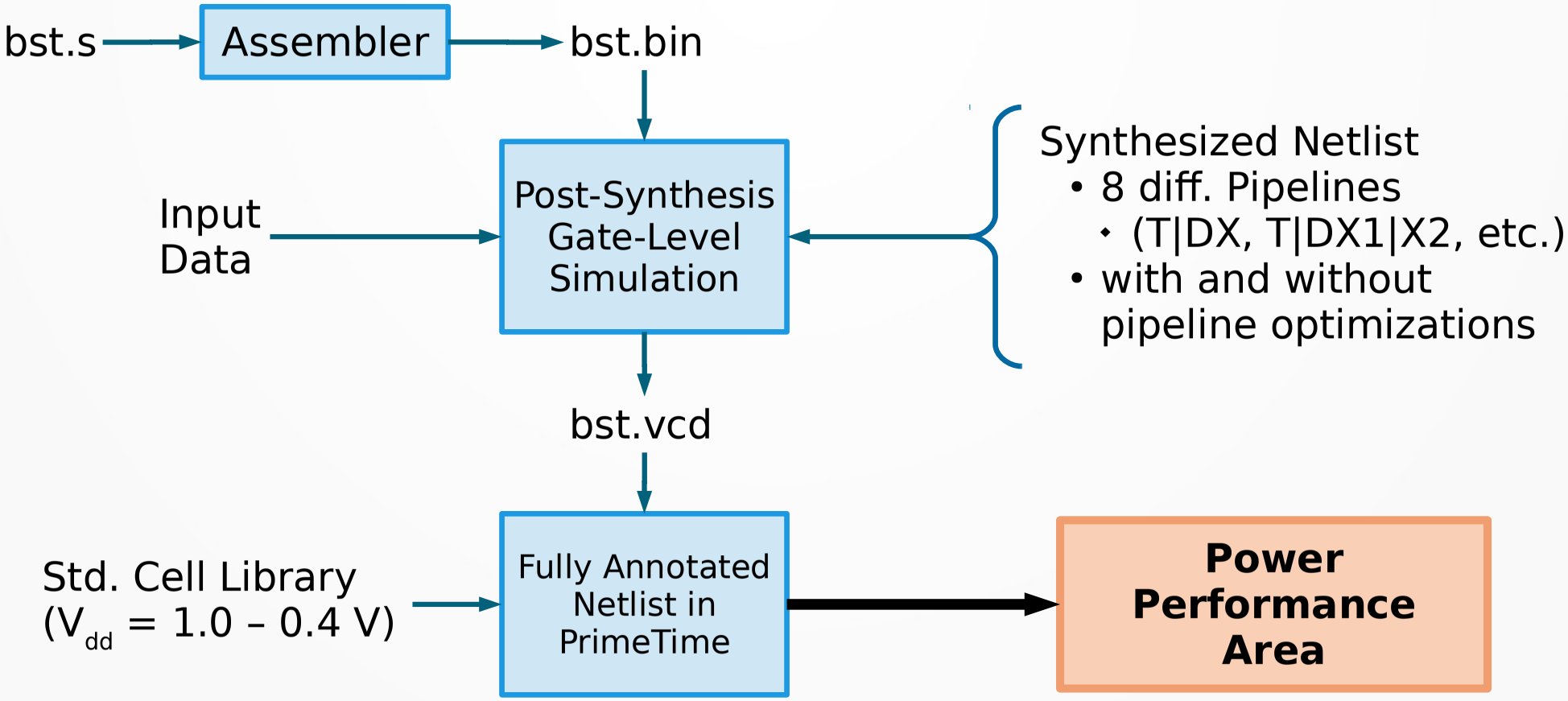
Evaluation Methodology



Evaluation Methodology



Evaluation Methodology

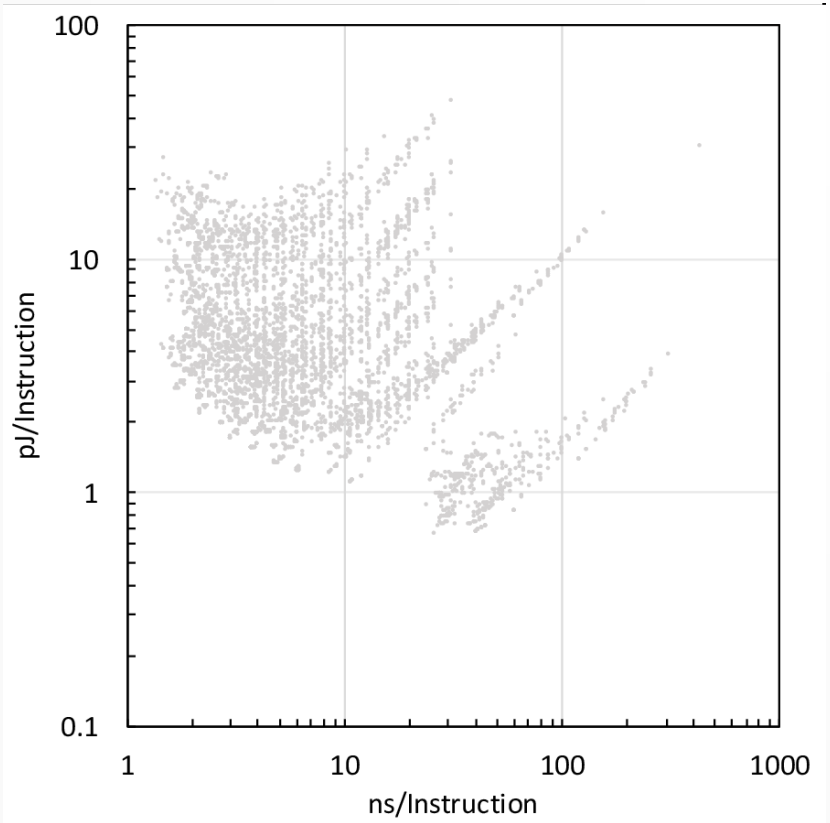


TSMC 65 nm low-, high- and standard-V_t GP standard cells

Outline

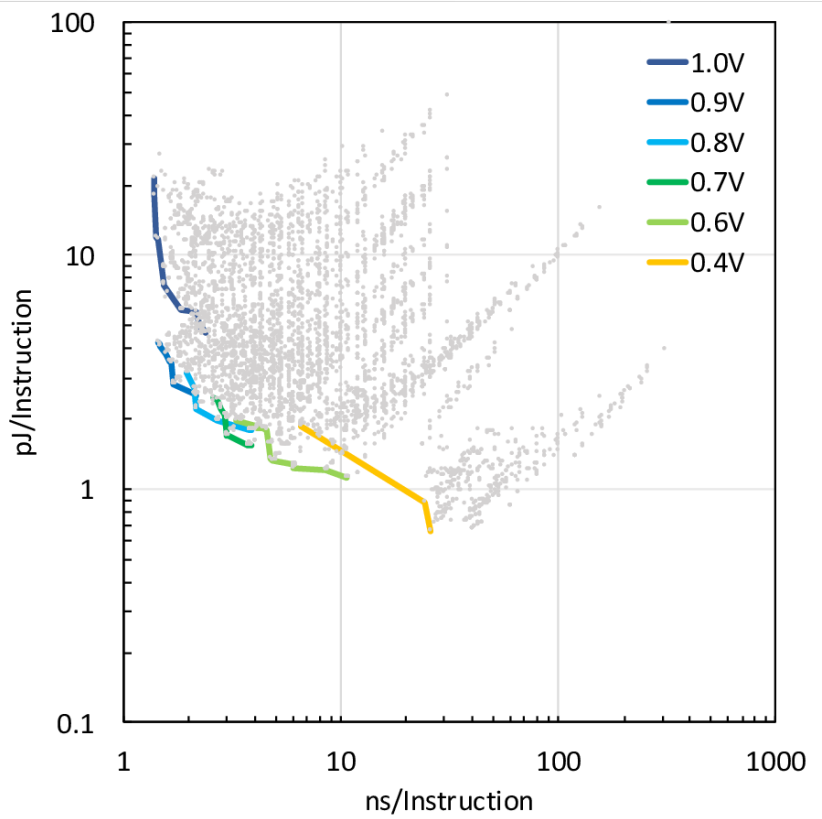
1. Motivation
2. Background
3. Challenges of Triggered Pipelines
4. Optimizations
5. Infrastructure and Methodology
6. **Results and Conclusion**

Microarchitectural and Circuit Design Space



- 4,000+ design points
- **71x span in energy**
  - 0.69 – 47.59 pJ / inst.
- **225x span in delay**
  - 1.37 – 3.09.03 ns / inst.

Microarchitectural and Circuit Design Space

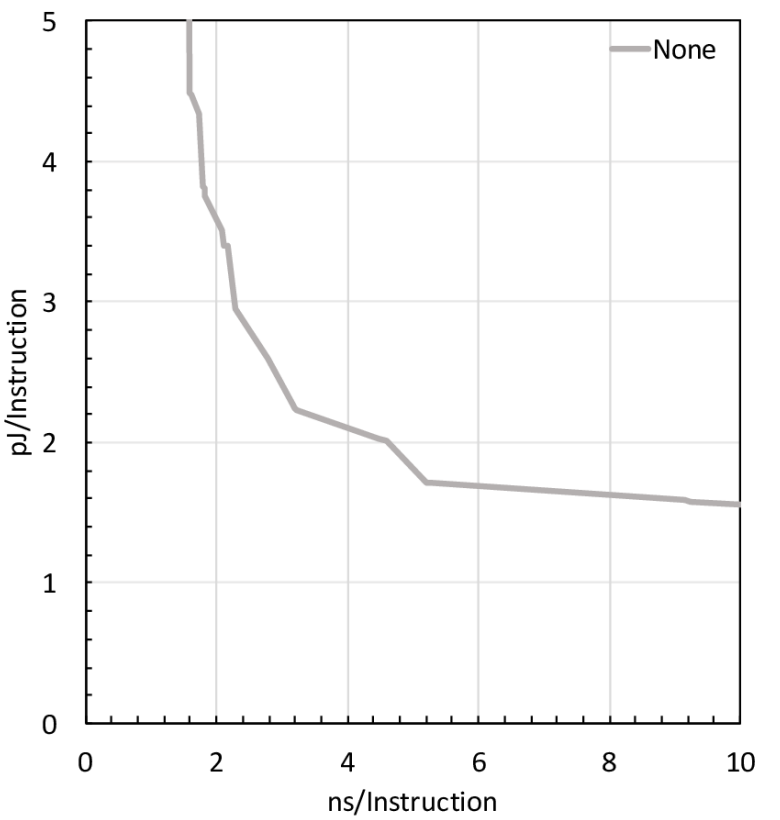


Pipelines enable aggressive supply voltage scaling

- 4,000+ design points
- **71x span in energy**
  - 0.69 – 47.59 pJ / inst.
- **225x span in delay**
  - 1.37 – 3.09.03 ns / inst.

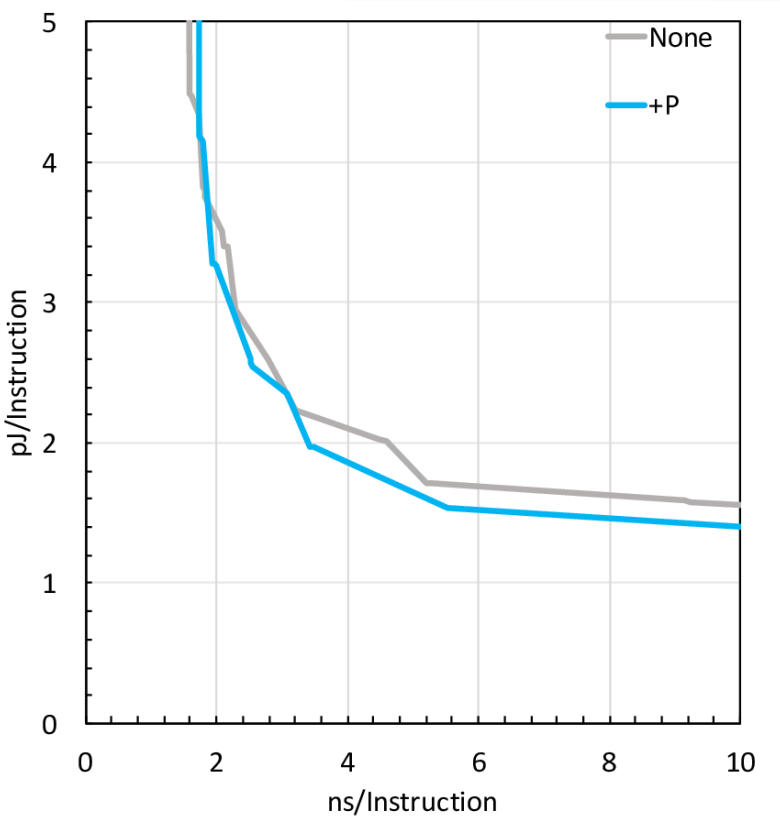
Microarchitectural and Circuit Design Space

- Proposed optimizations especially effective in the balanced region of the Pareto frontier



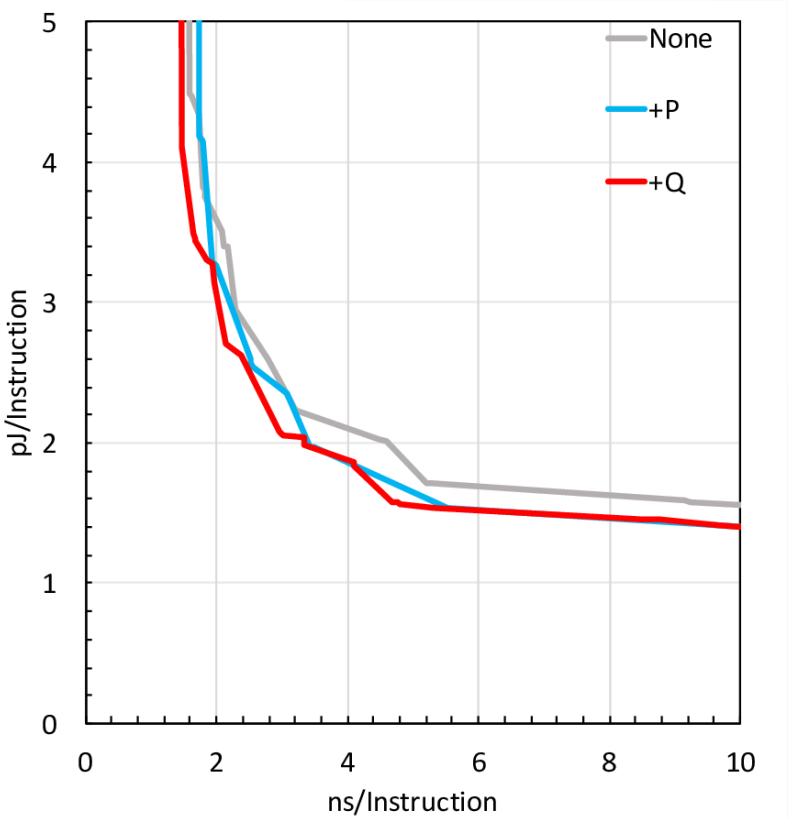
Microarchitectural and Circuit Design Space

- Proposed optimizations especially effective in the balanced region of the Pareto frontier
- Predicate prediction (+P)



Microarchitectural and Circuit Design Space

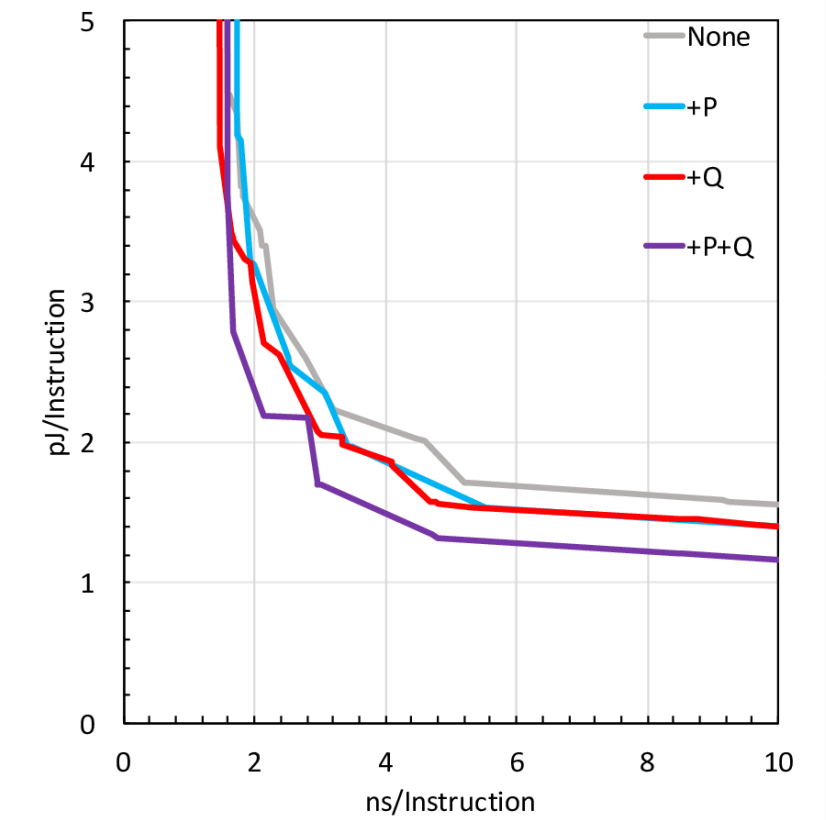
- Proposed optimizations especially effective in the balanced region of the Pareto frontier
- Predicate prediction (+P)
- Effective Queue Status (+Q)



Microarchitectural and Circuit Design Space

- Proposed optimizations especially effective in the balanced region of the Pareto frontier
- Predicate prediction (+P)
- Effective Queue Status (+Q)

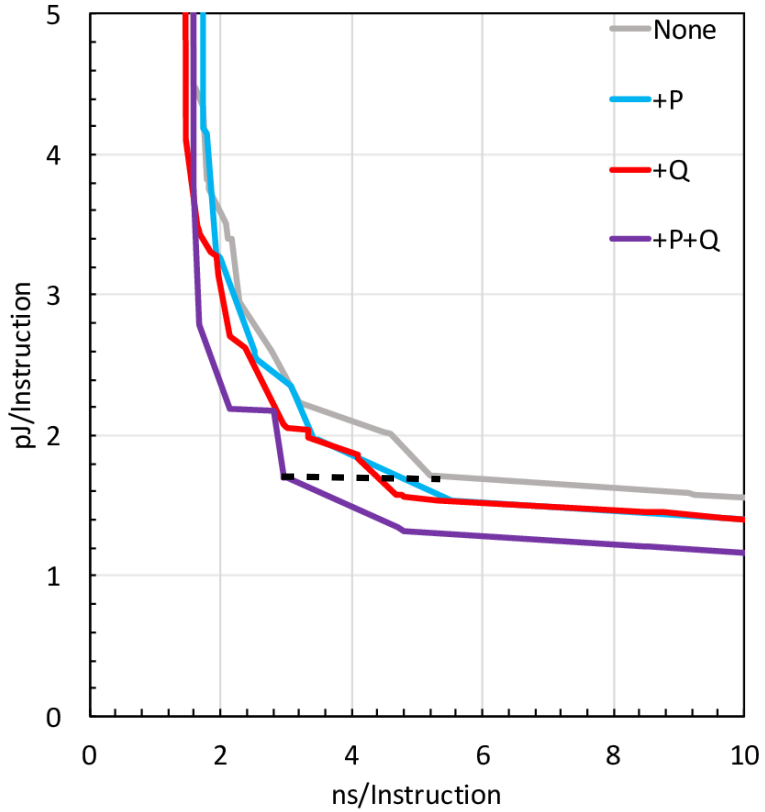
Effect is additive



Microarchitectural and Circuit Design Space

- Proposed optimizations especially effective in the balanced region of the Pareto frontier
- Predicate prediction (+P)
- Effective Queue Status (+Q)

Effect is additive



Example: with an energy budget of 1.7 pJ / inst., +P+Q improves throughput by **2x**

Conclusion

- Pipelining a triggered processing element presents new but *tractable* microarchitectural hazards
- Demonstrated two techniques:
  - Predicate Prediction
  - Effective Queue Status
  - Work well in a wide range of regimes: low-power to high-performance

Thank You

github.com/arcade-lab/tia-infrastructure