

# Dynamic Branch Prediction with Perceptrons

Daniel A. Jiménez, Calvin Lin  
*The University of Texas at Austin*  
HPCA 2001

Presented by Nils Wistoff  
Mentors: Firtina Can, Salami Behzad, Hasan Hasan  
Seminar in Computer Architecture  
April 29, 2021

# Outline

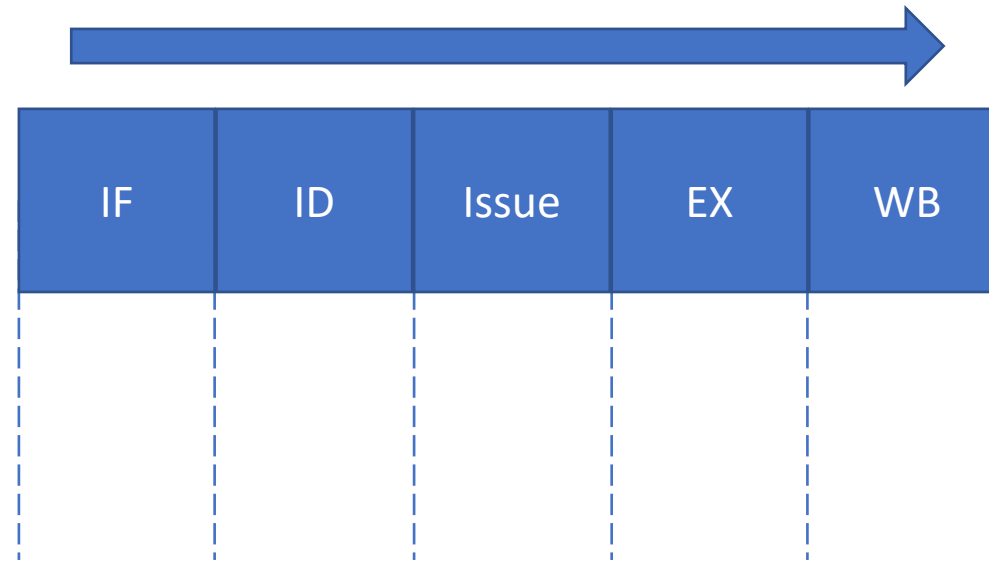
## 1. Branch Prediction

## 2. Paper Summary

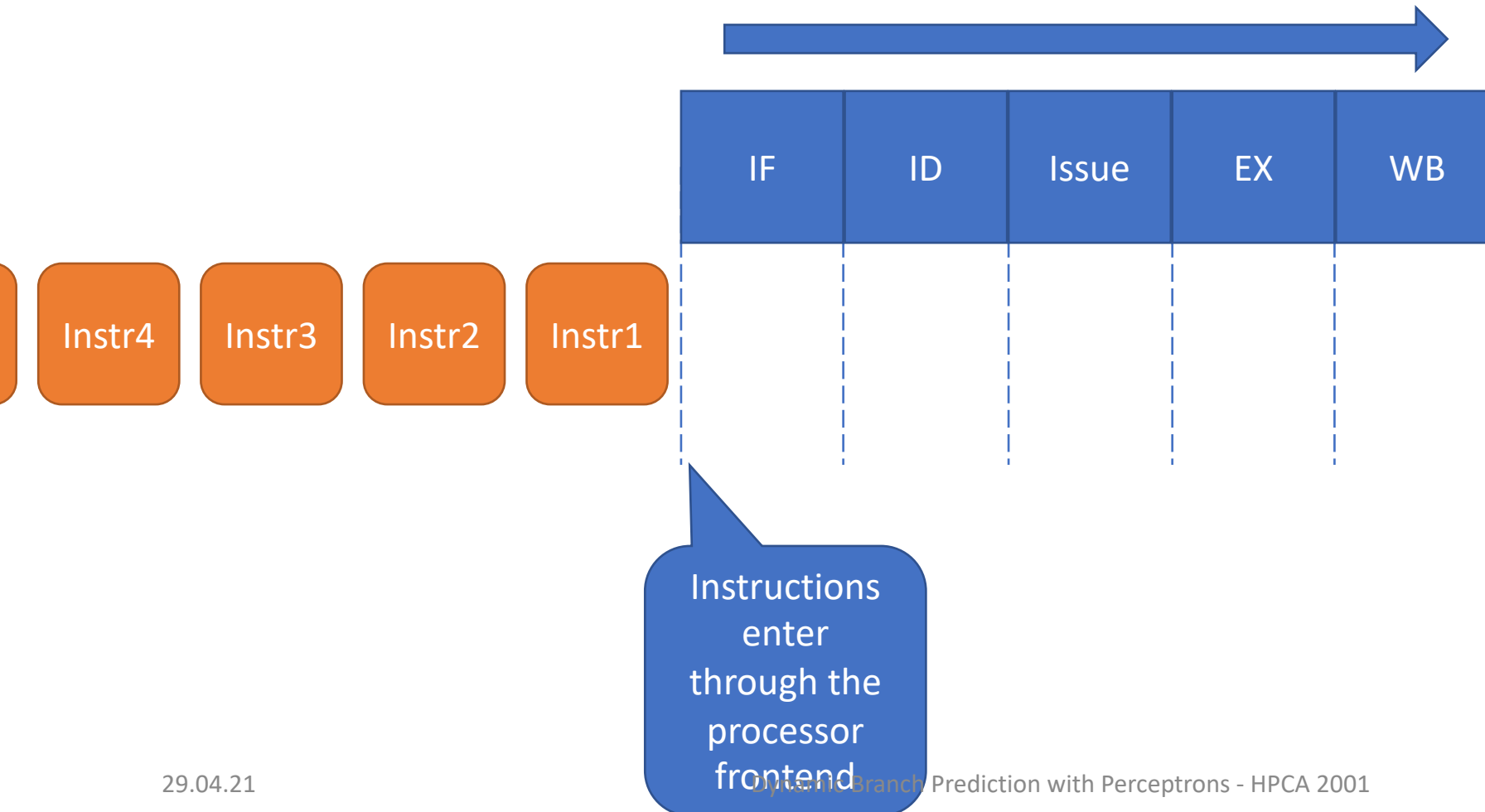
1. Executive Summary
2. Background & Related Work
3. Key Idea: Perceptron as Branch Predictor
4. Implementation
5. Results
6. Optimization: Hybrid Predictor
7. Final Results
8. Conclusion

## 3. Analysis and Discussion

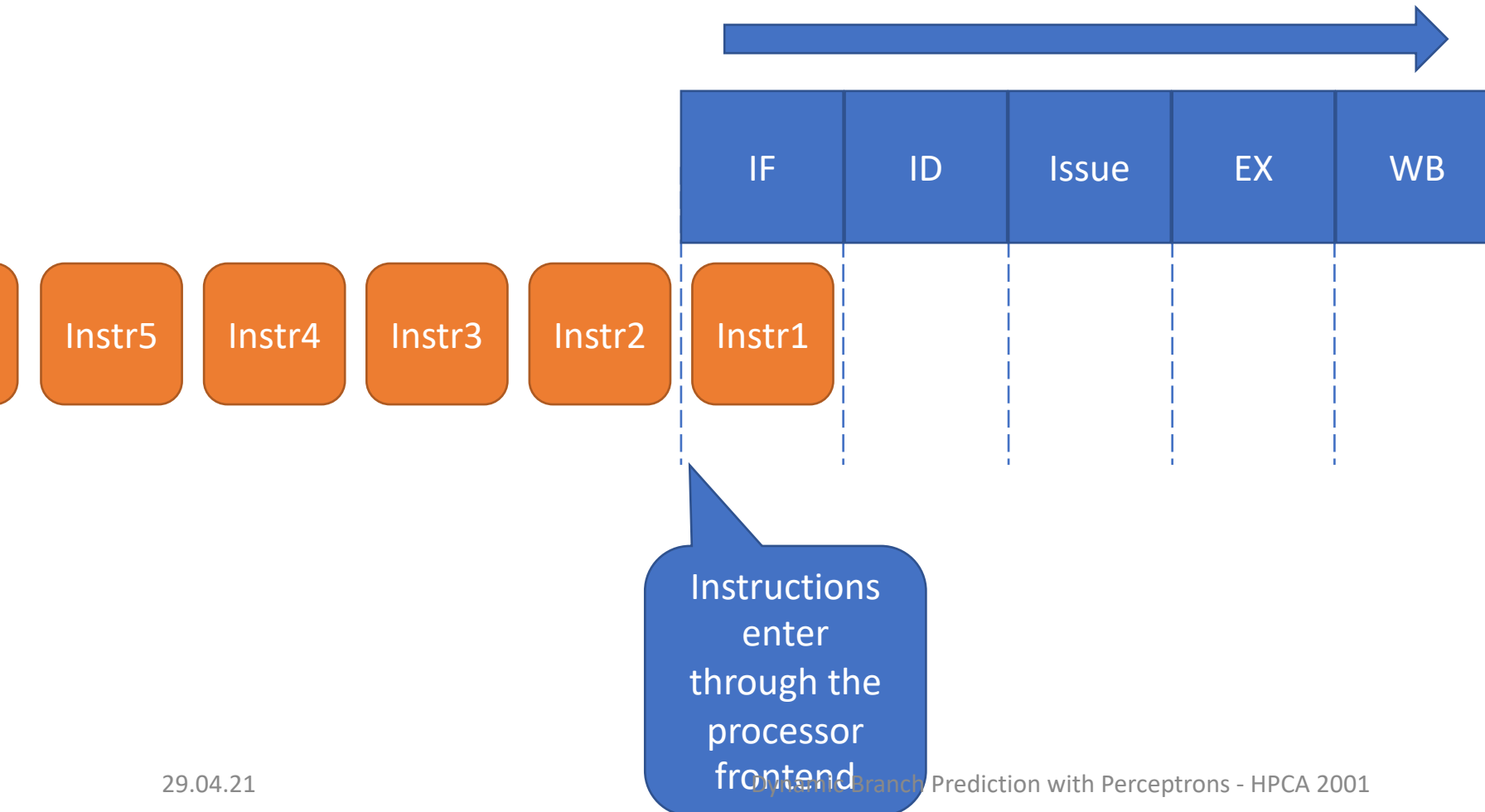
# Processor Pipeline



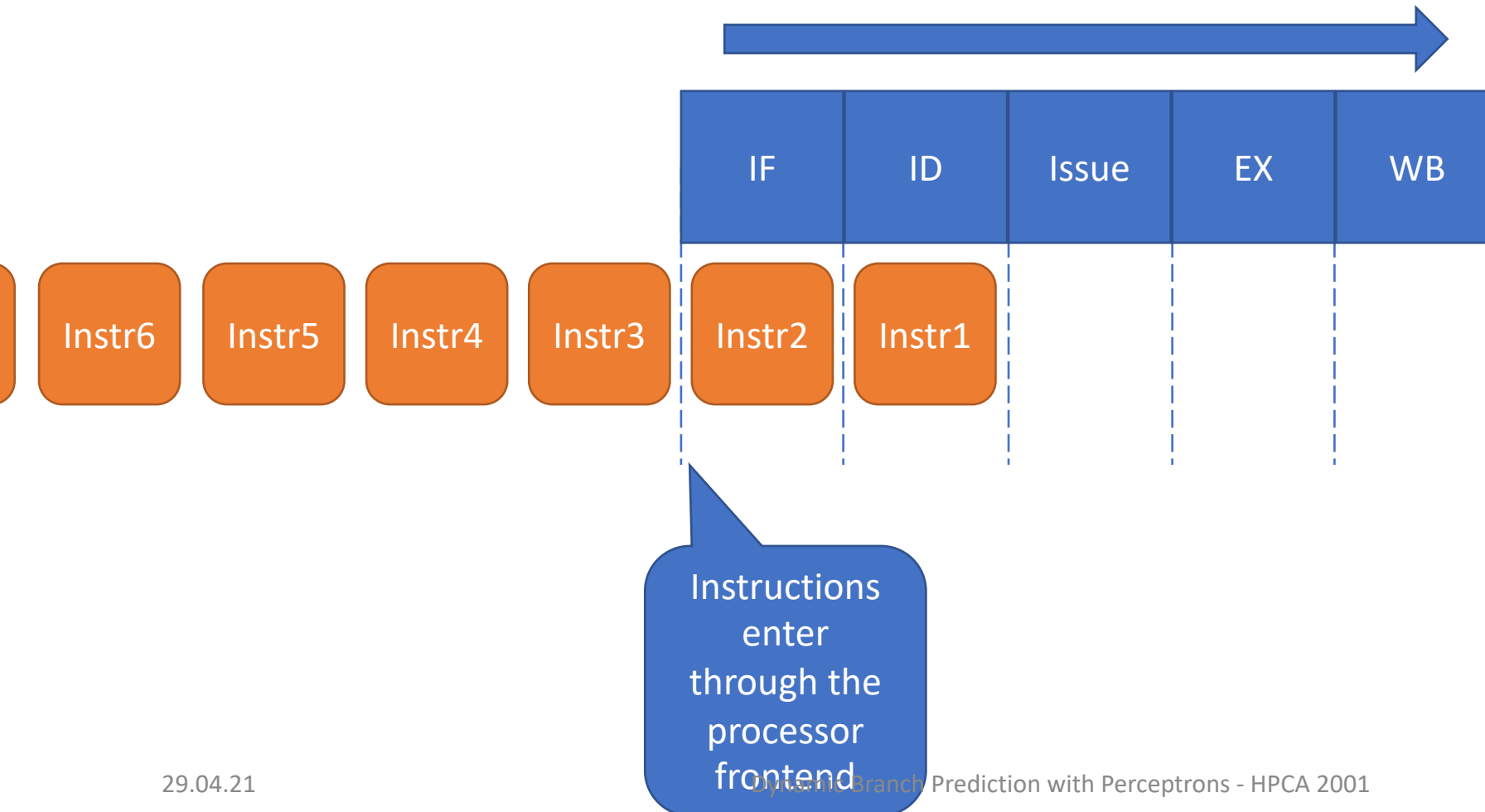
# Processor Pipeline



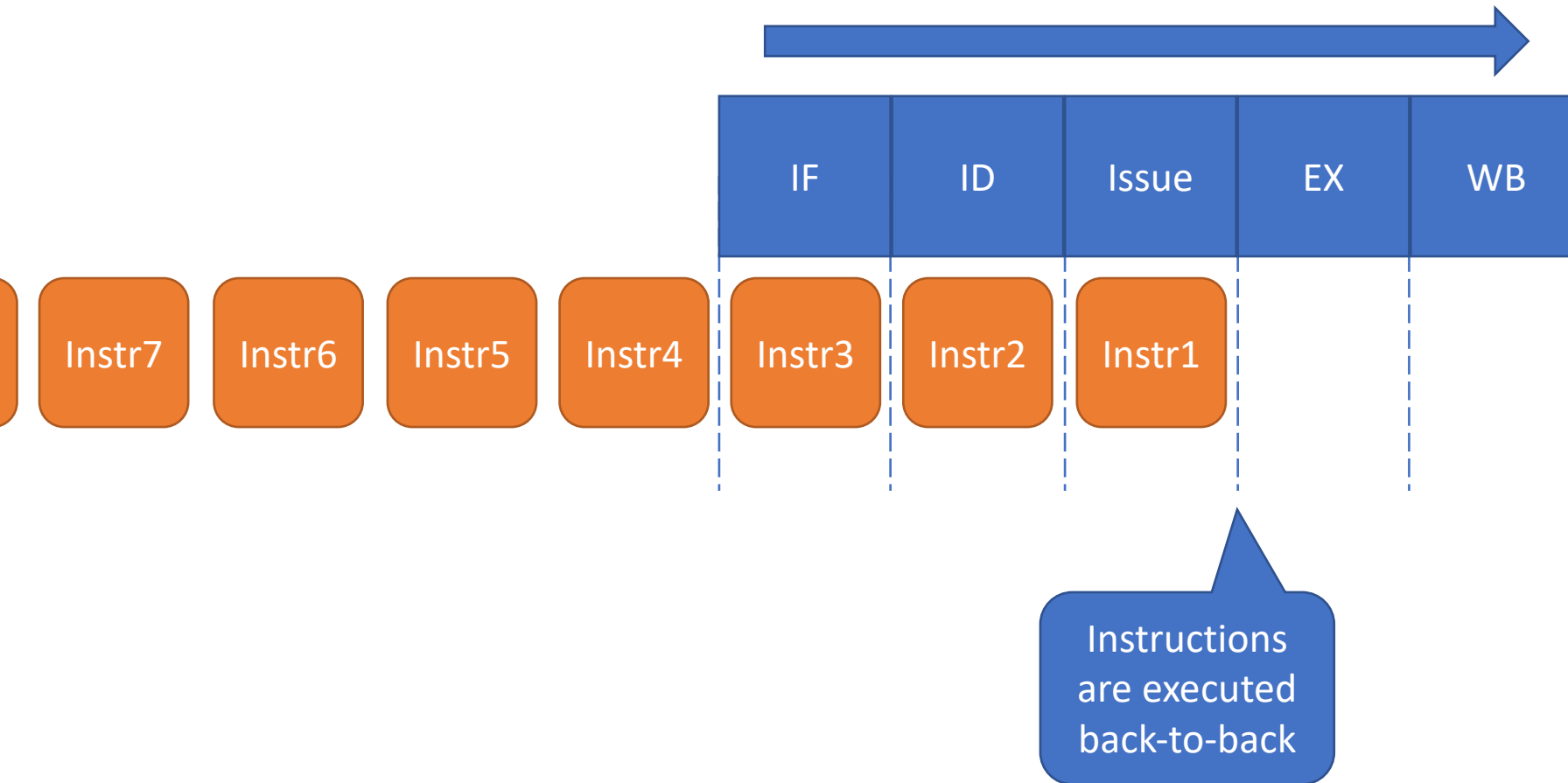
# Processor Pipeline



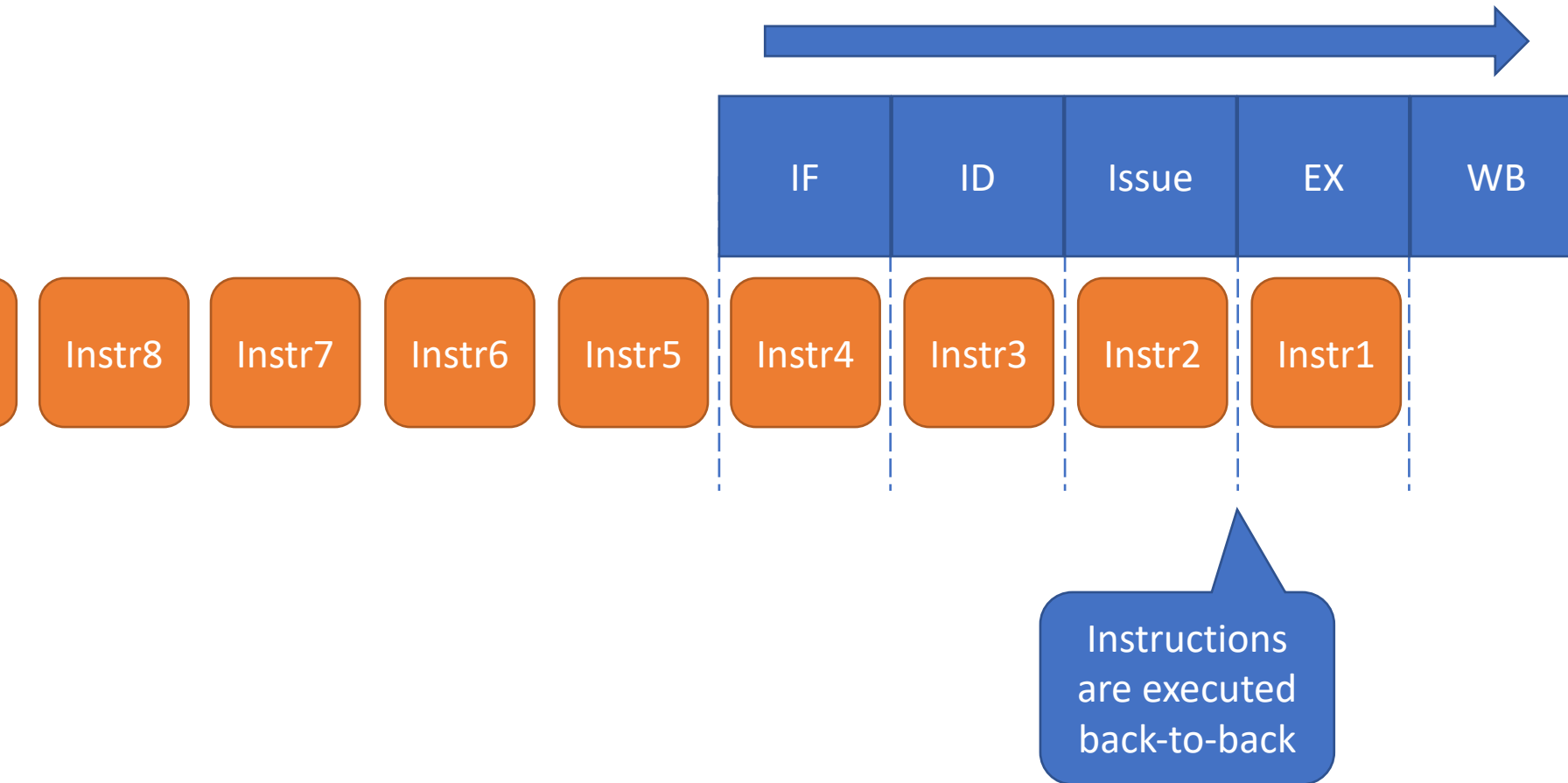
# Processor Pipeline



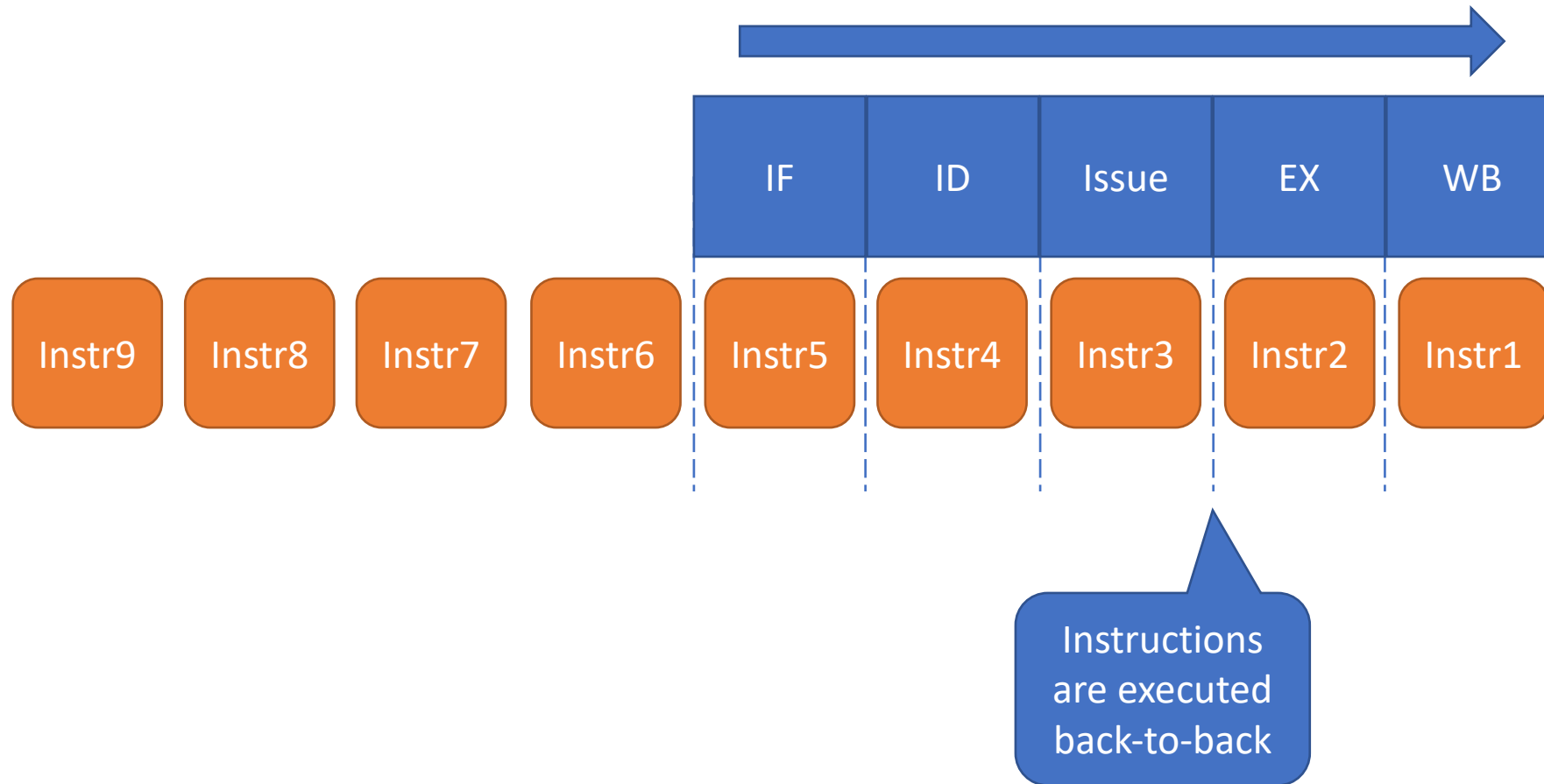
# Processor Pipeline



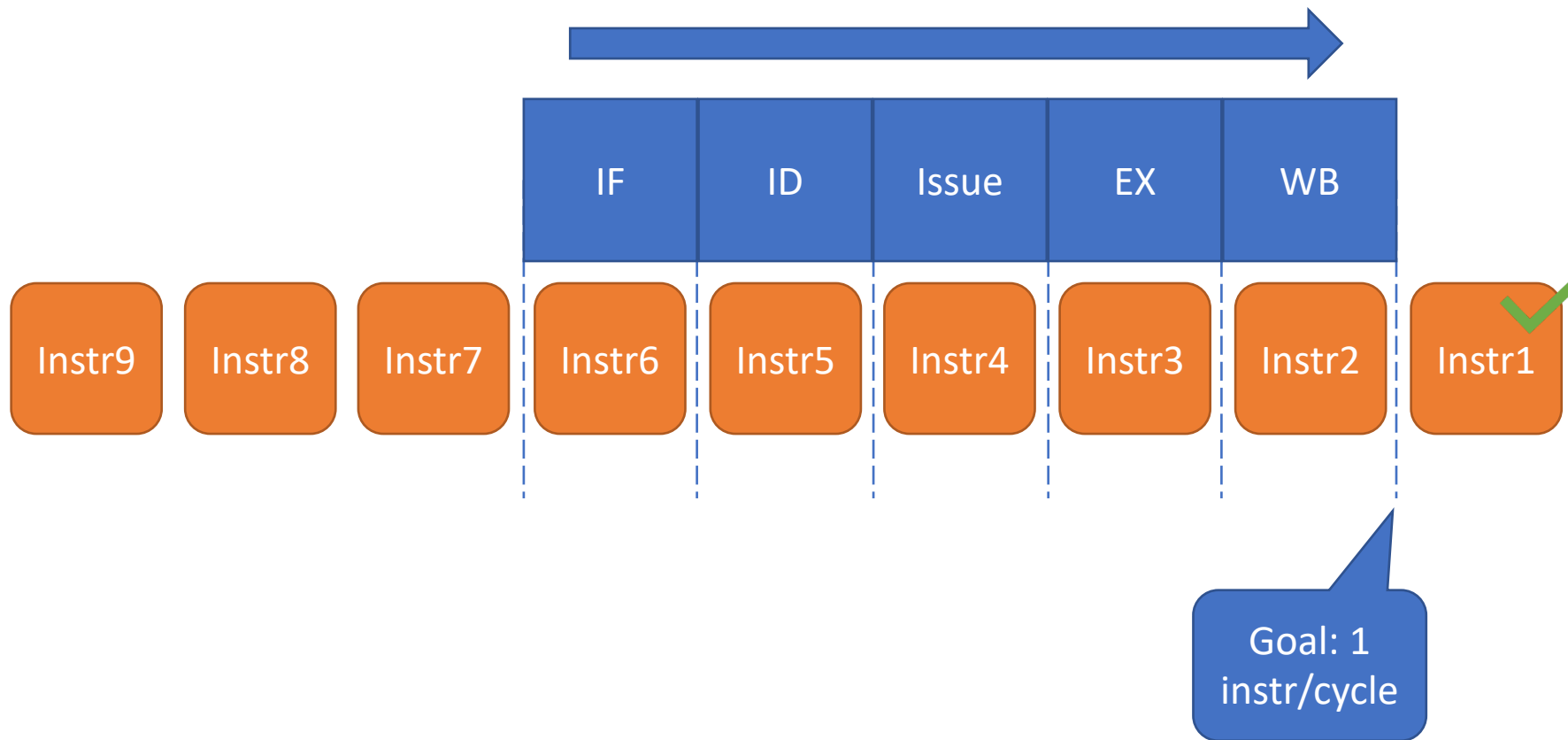
# Processor Pipeline



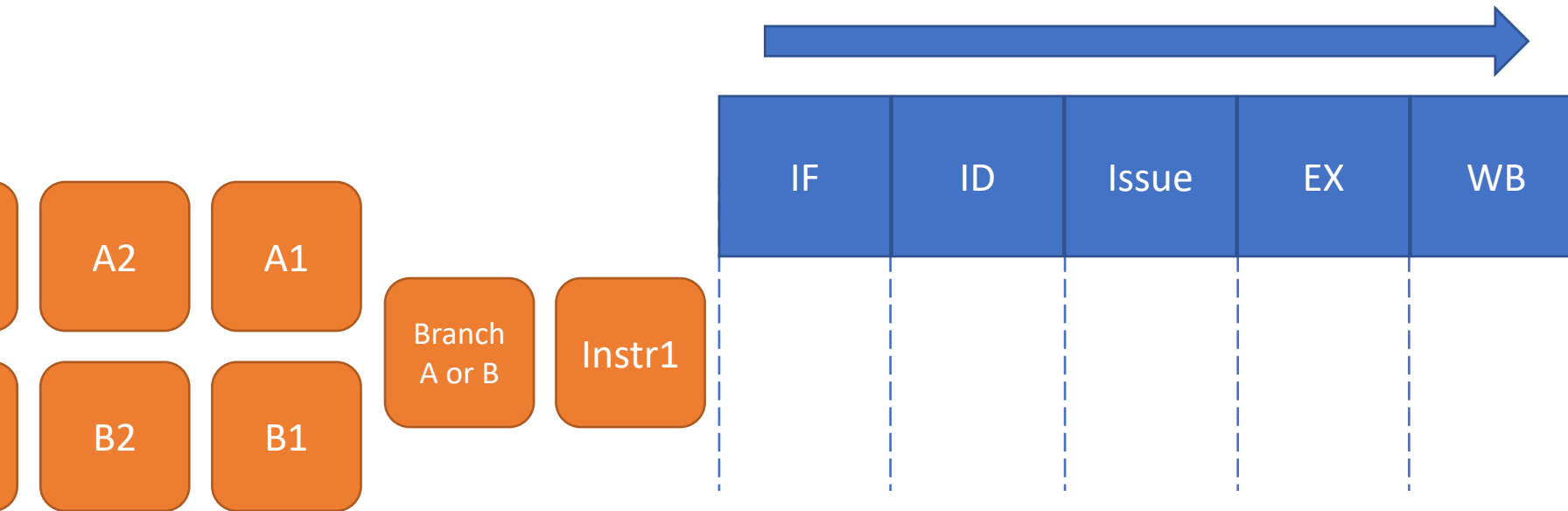
# Processor Pipeline



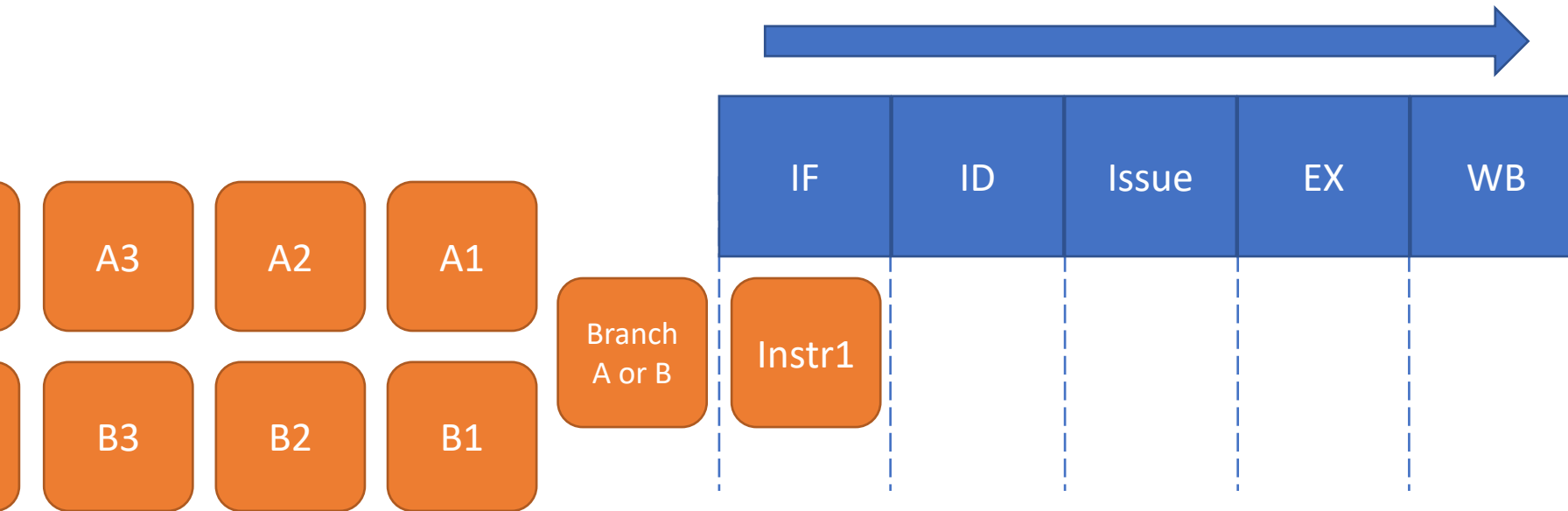
# Processor Pipeline



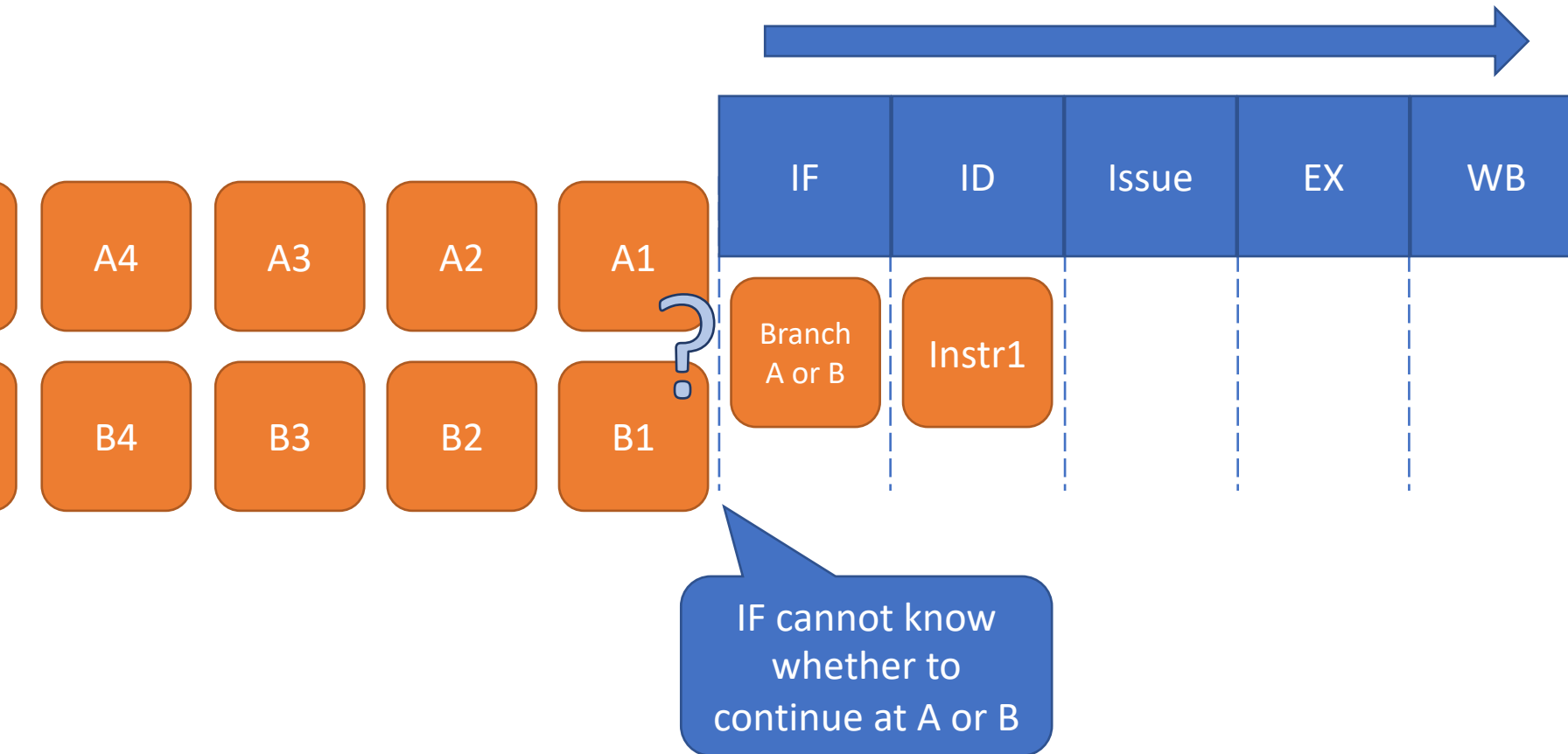
# Control Hazard



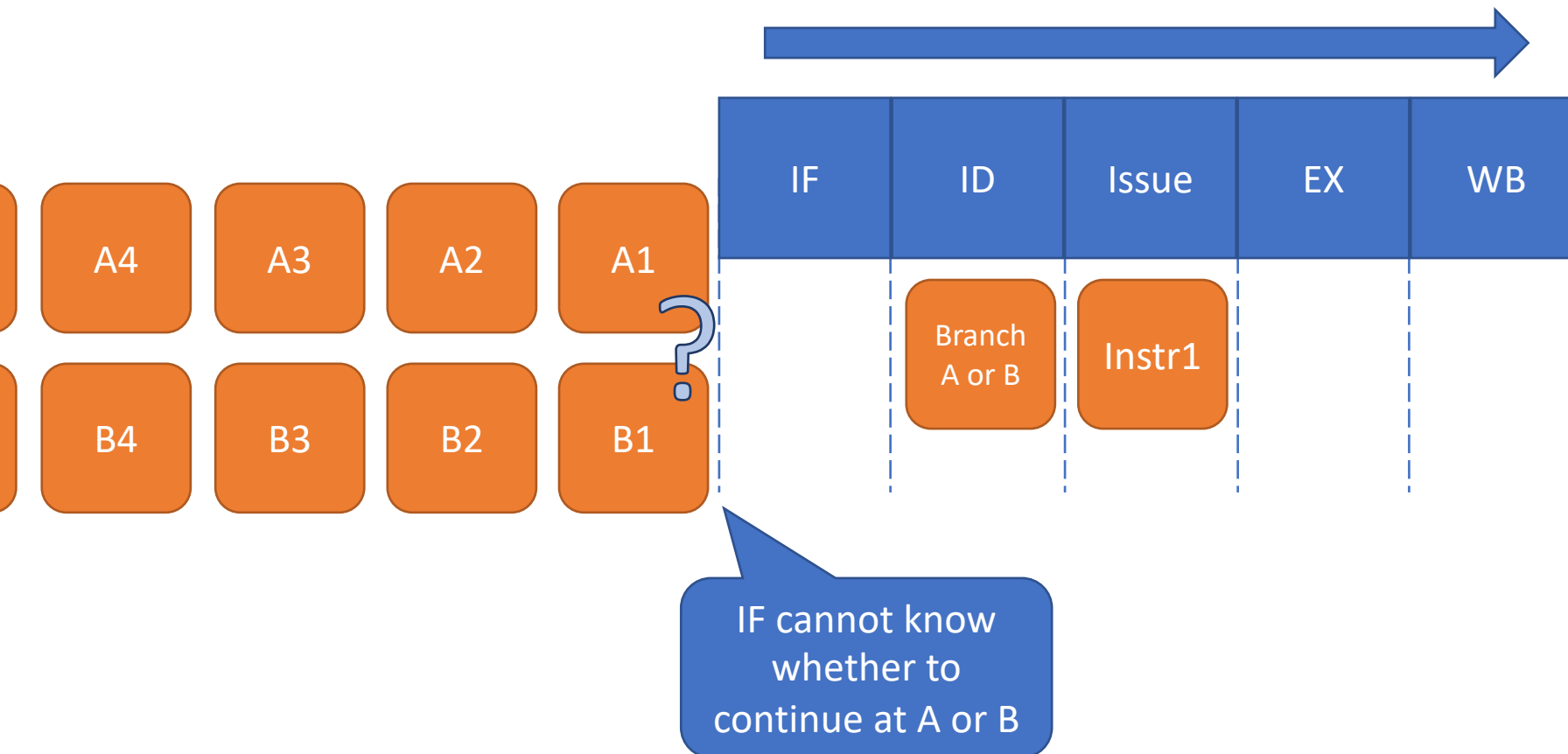
# Control Hazard



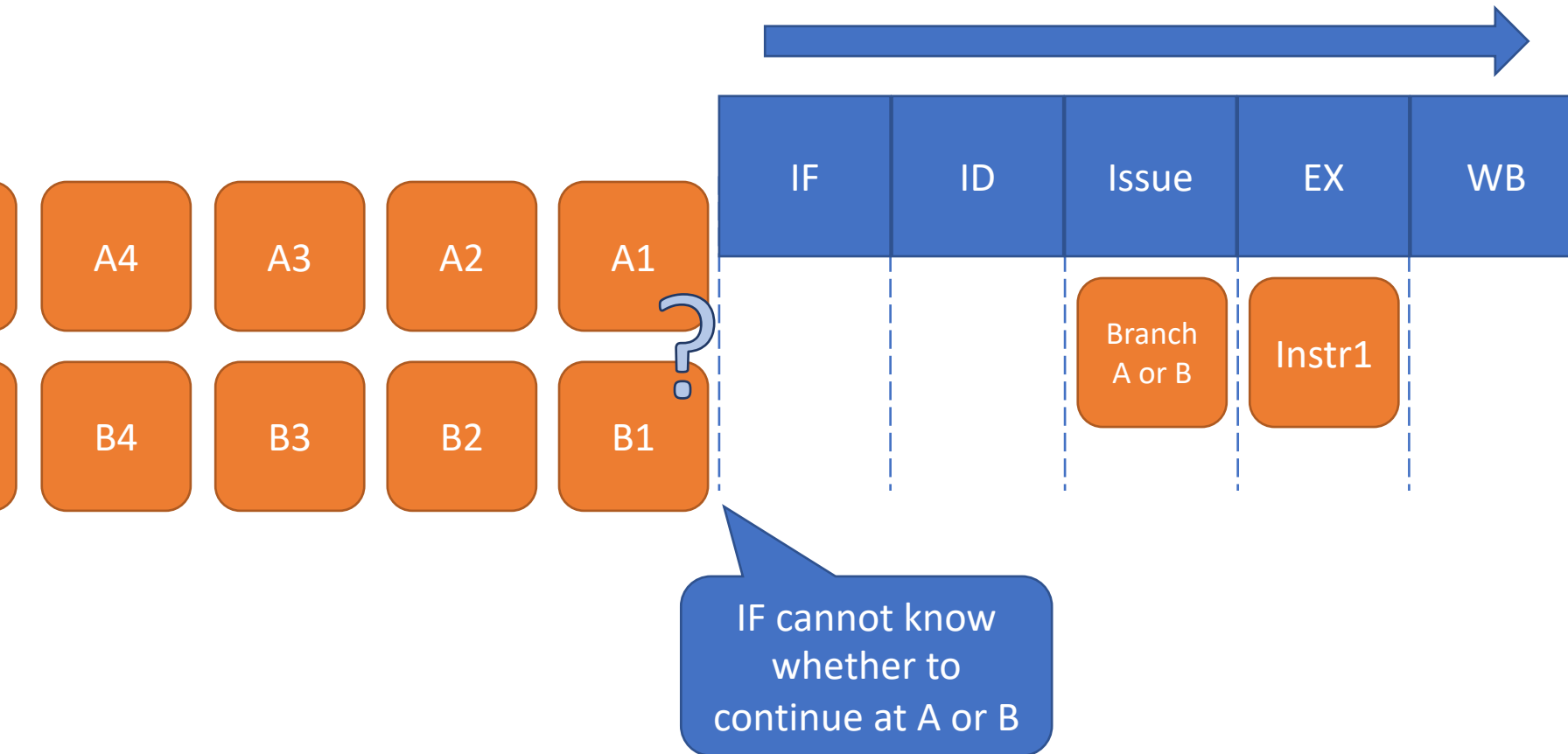
# Control Hazard



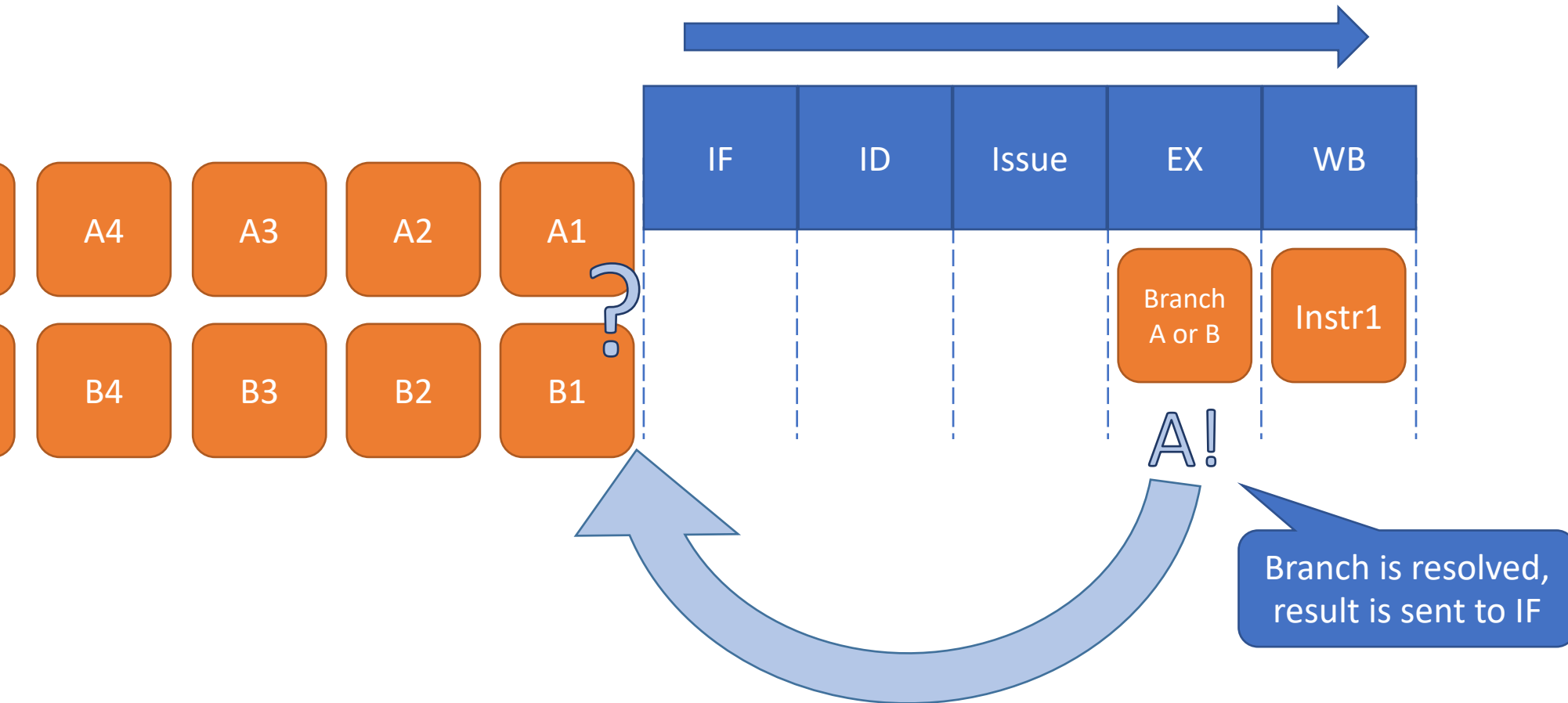
# Control Hazard



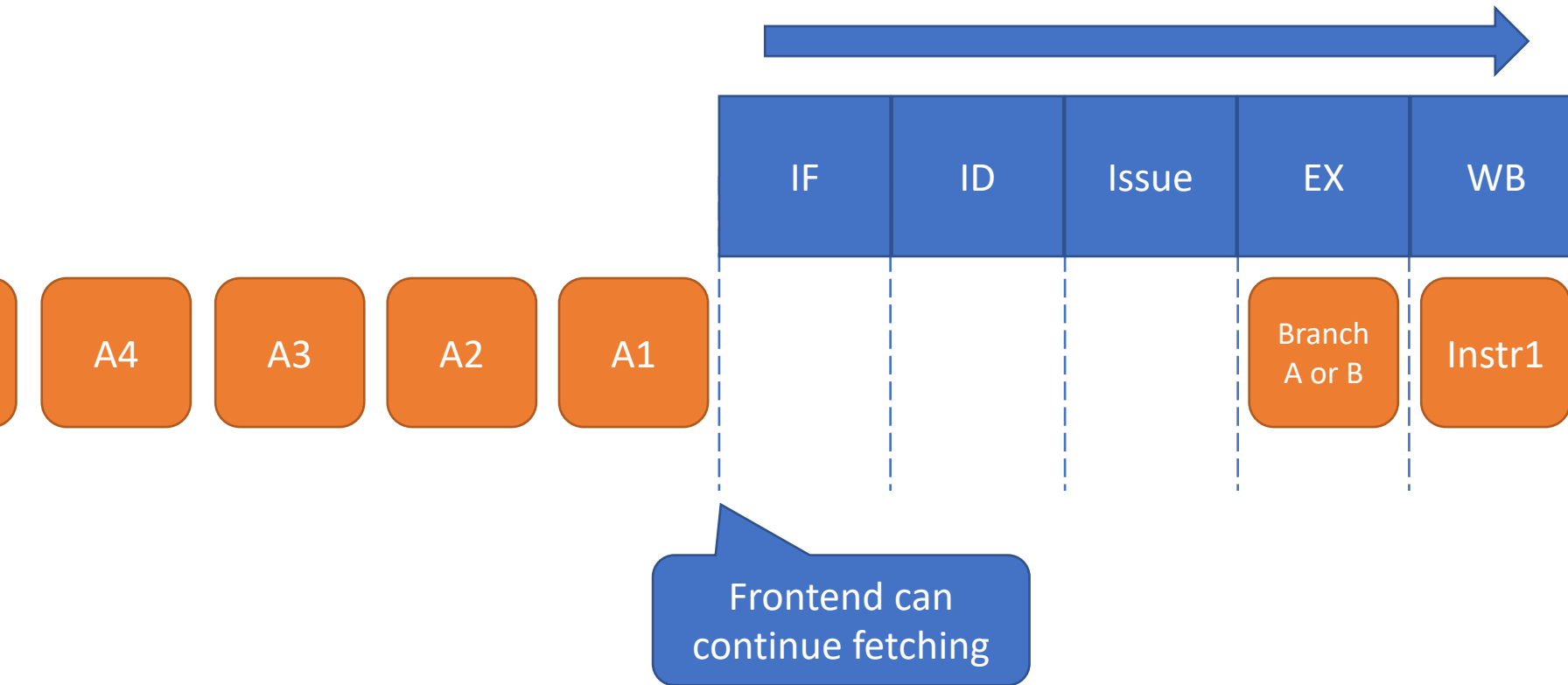
# Control Hazard



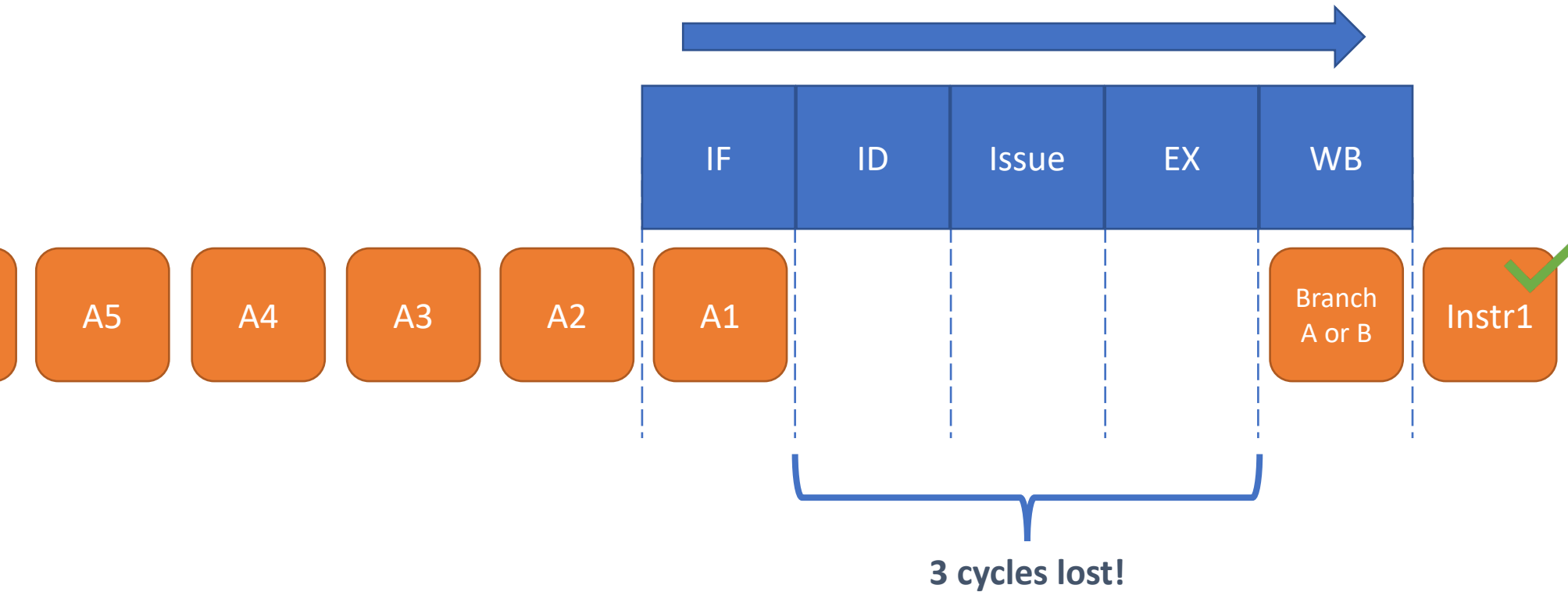
# Control Hazard



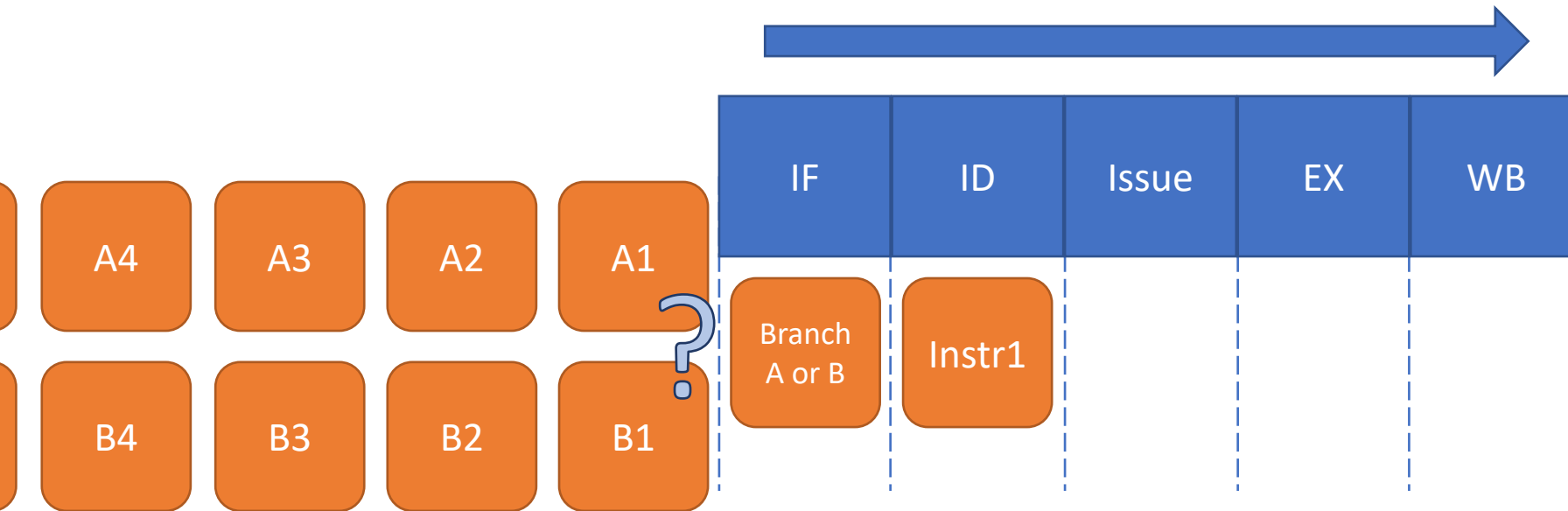
# Control Hazard



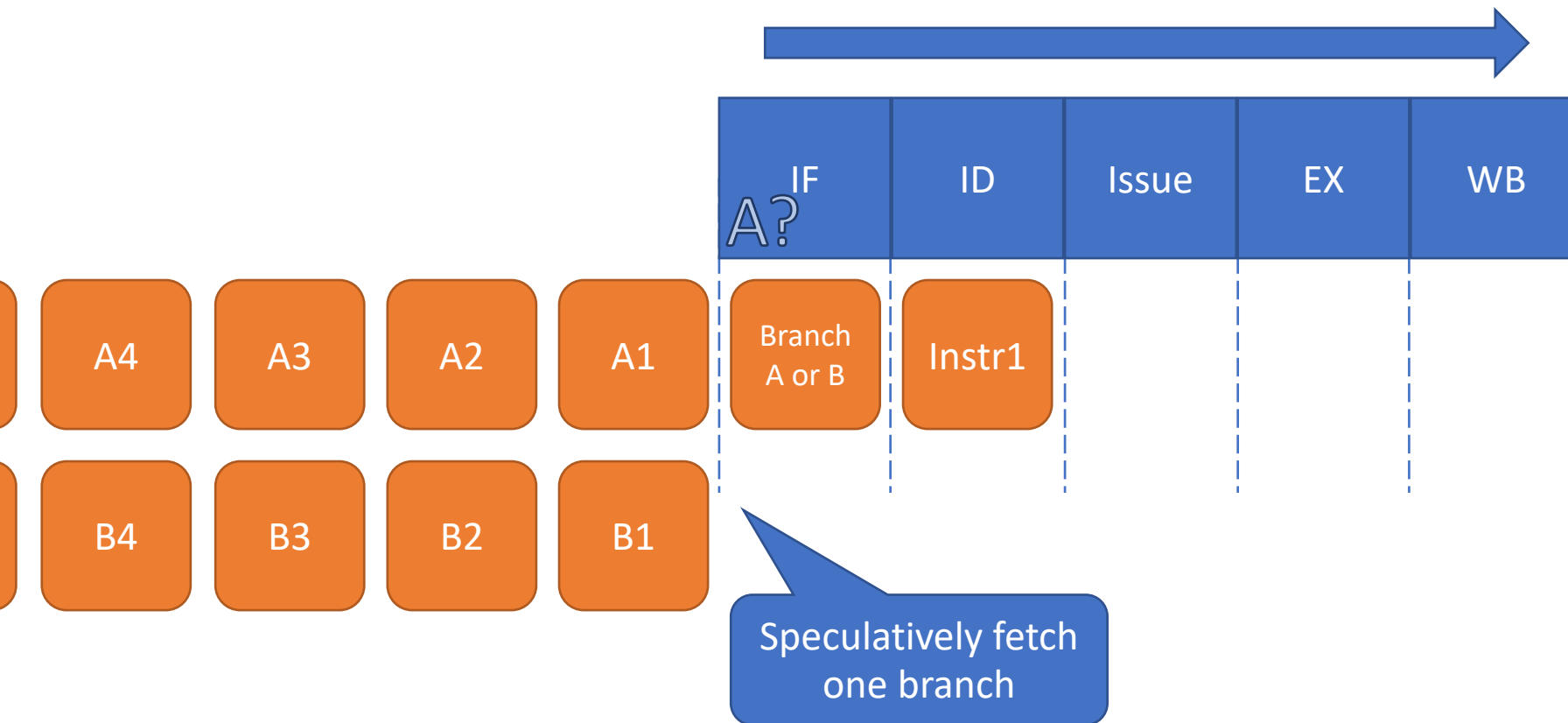
# Control Hazard



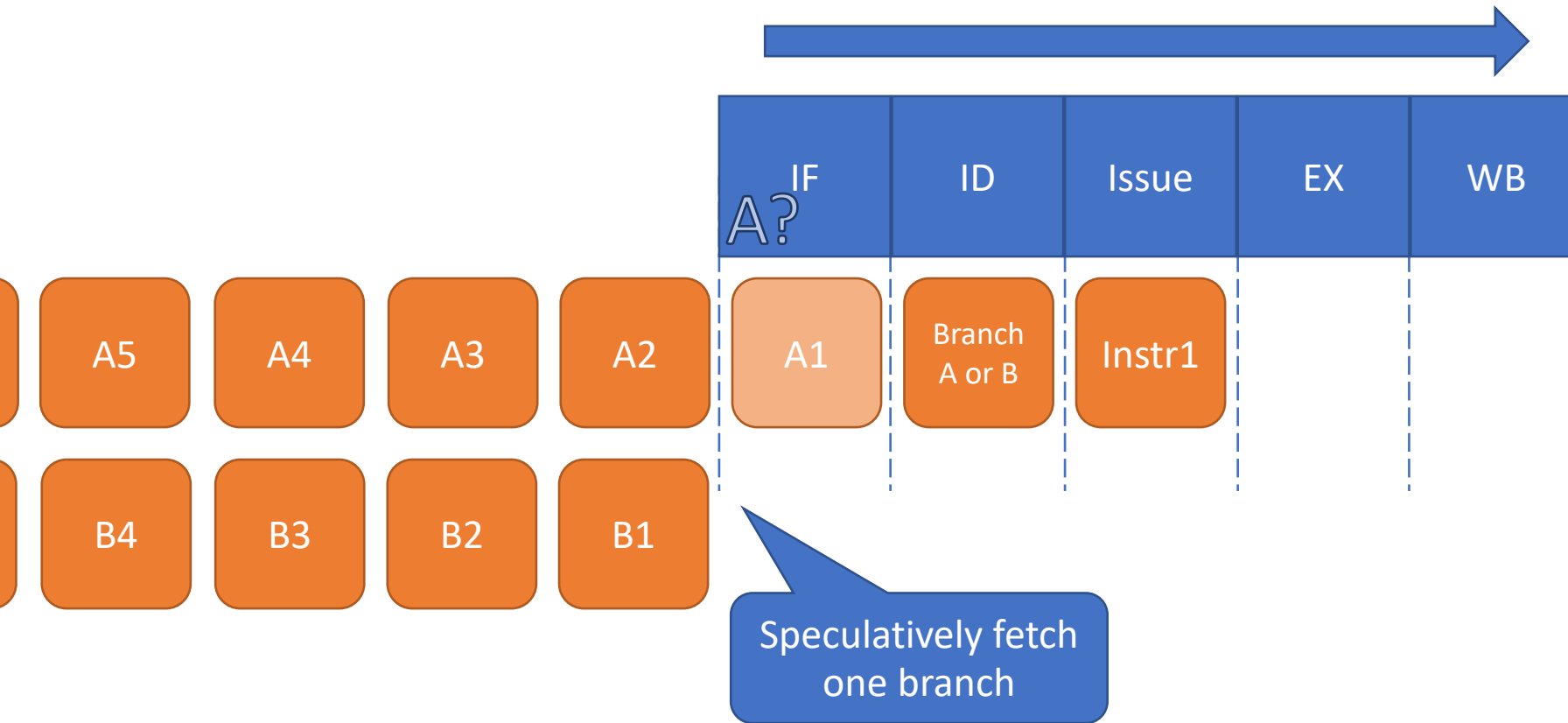
# Branch Prediction



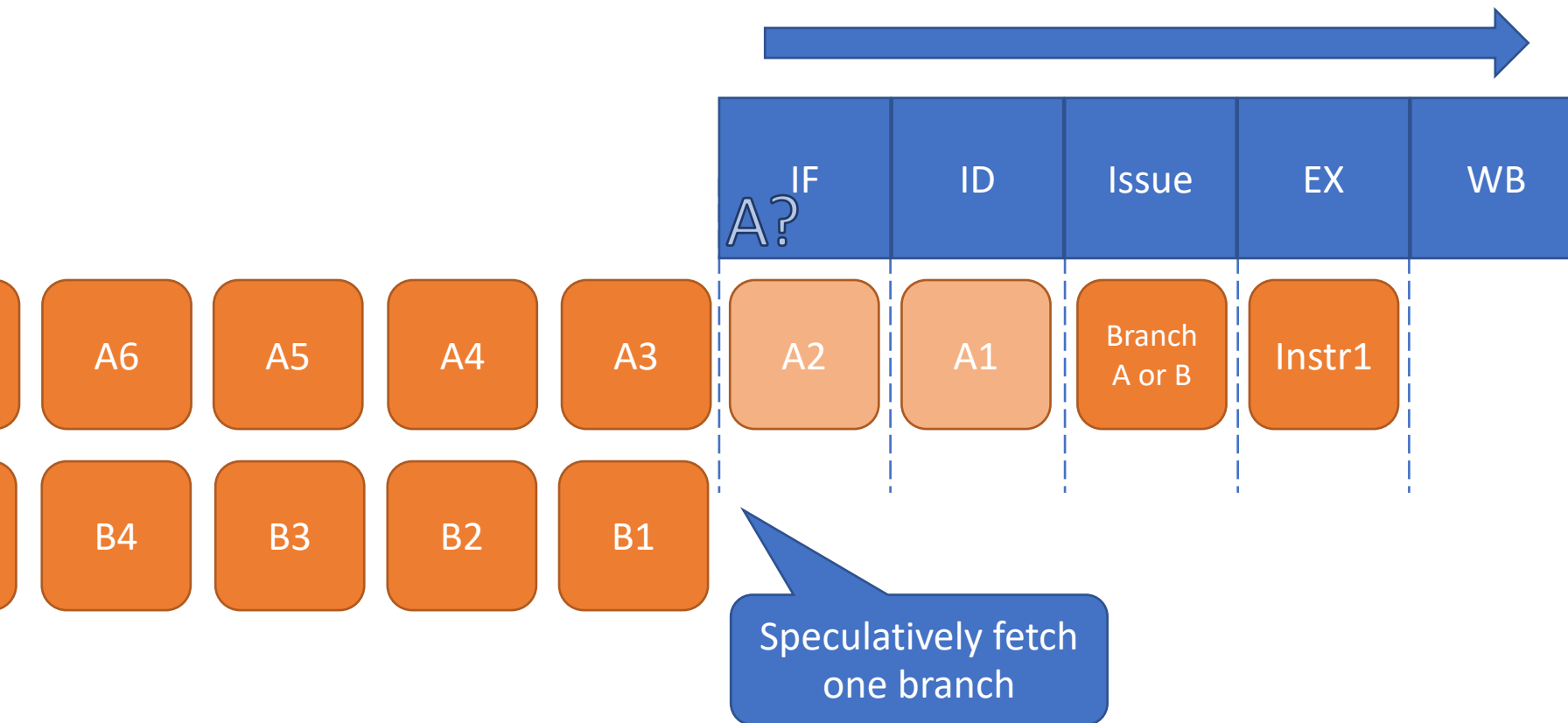
# Branch Prediction



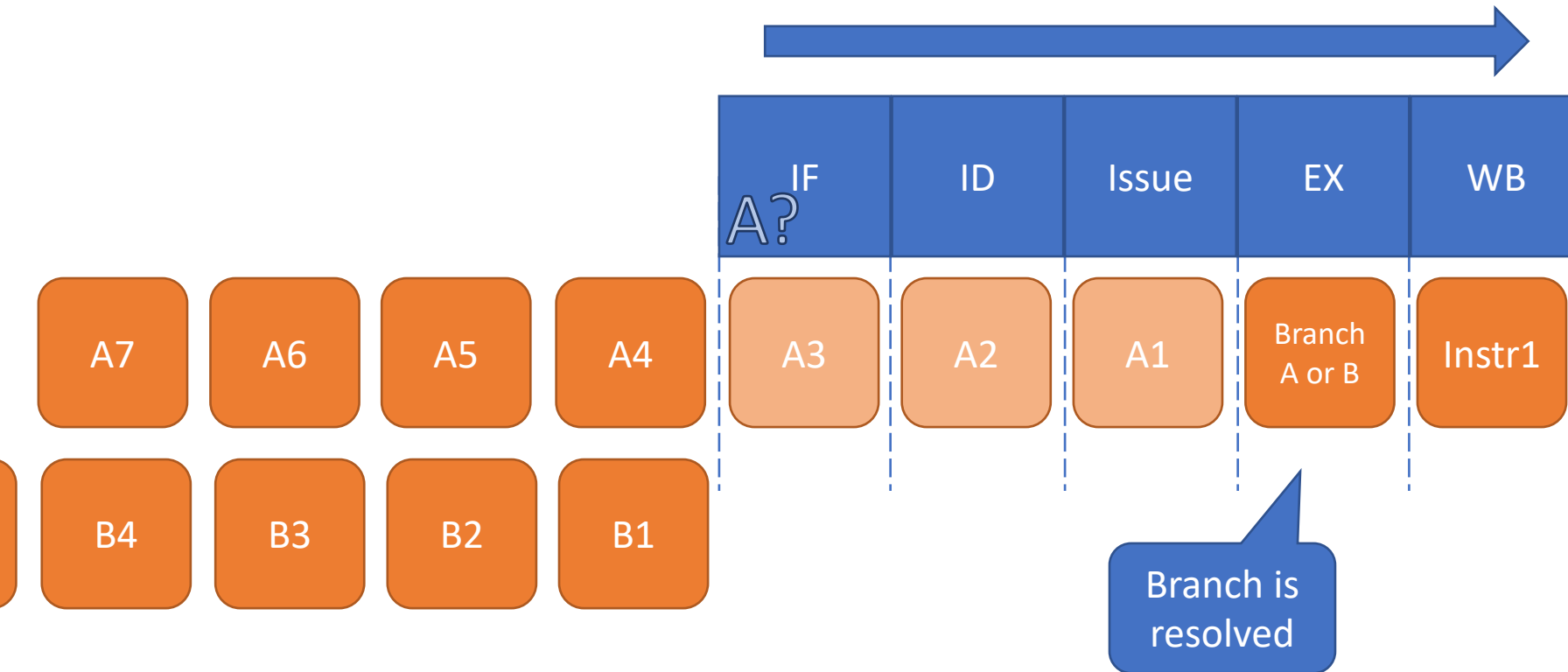
# Branch Prediction



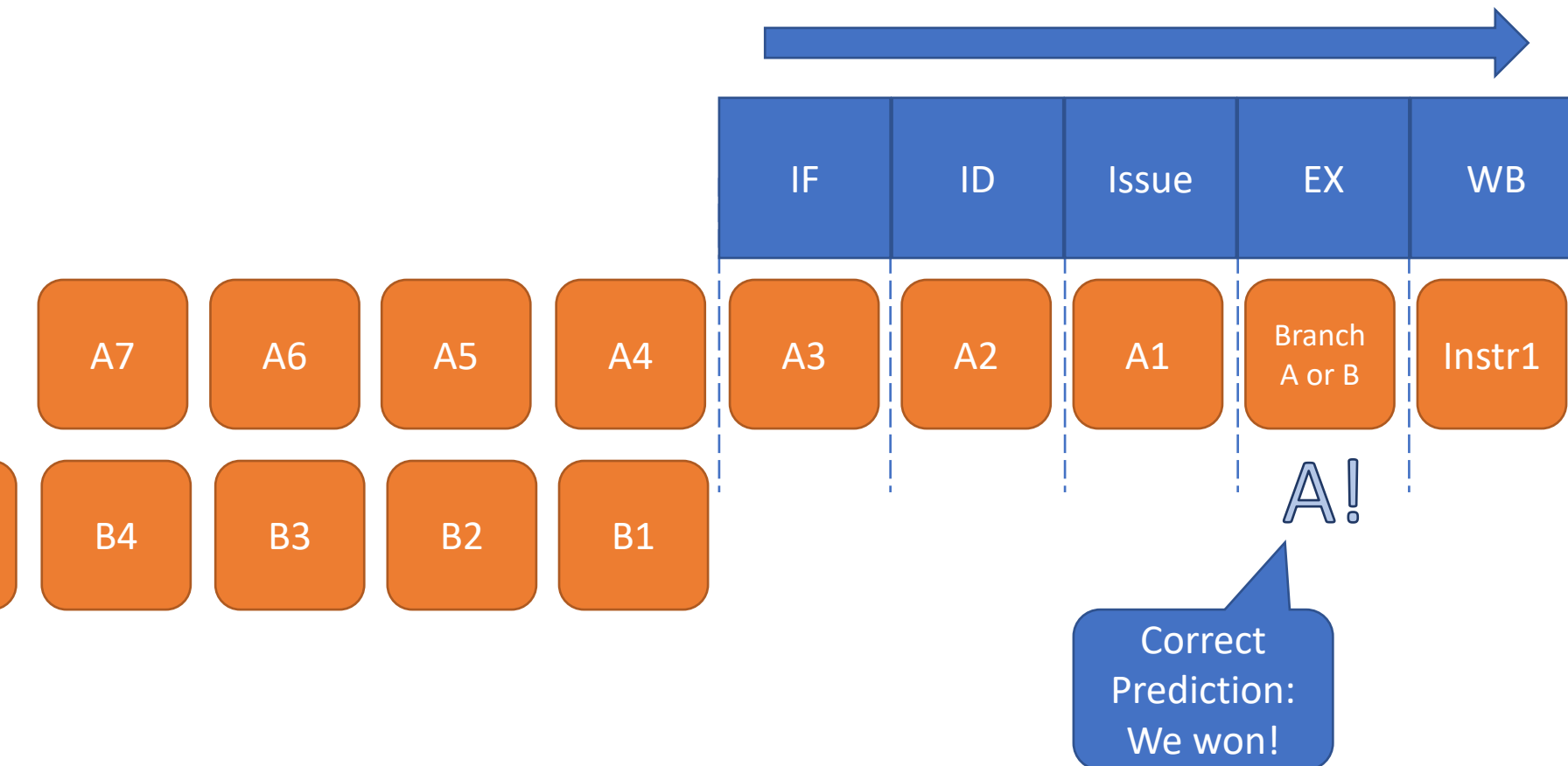
# Branch Prediction



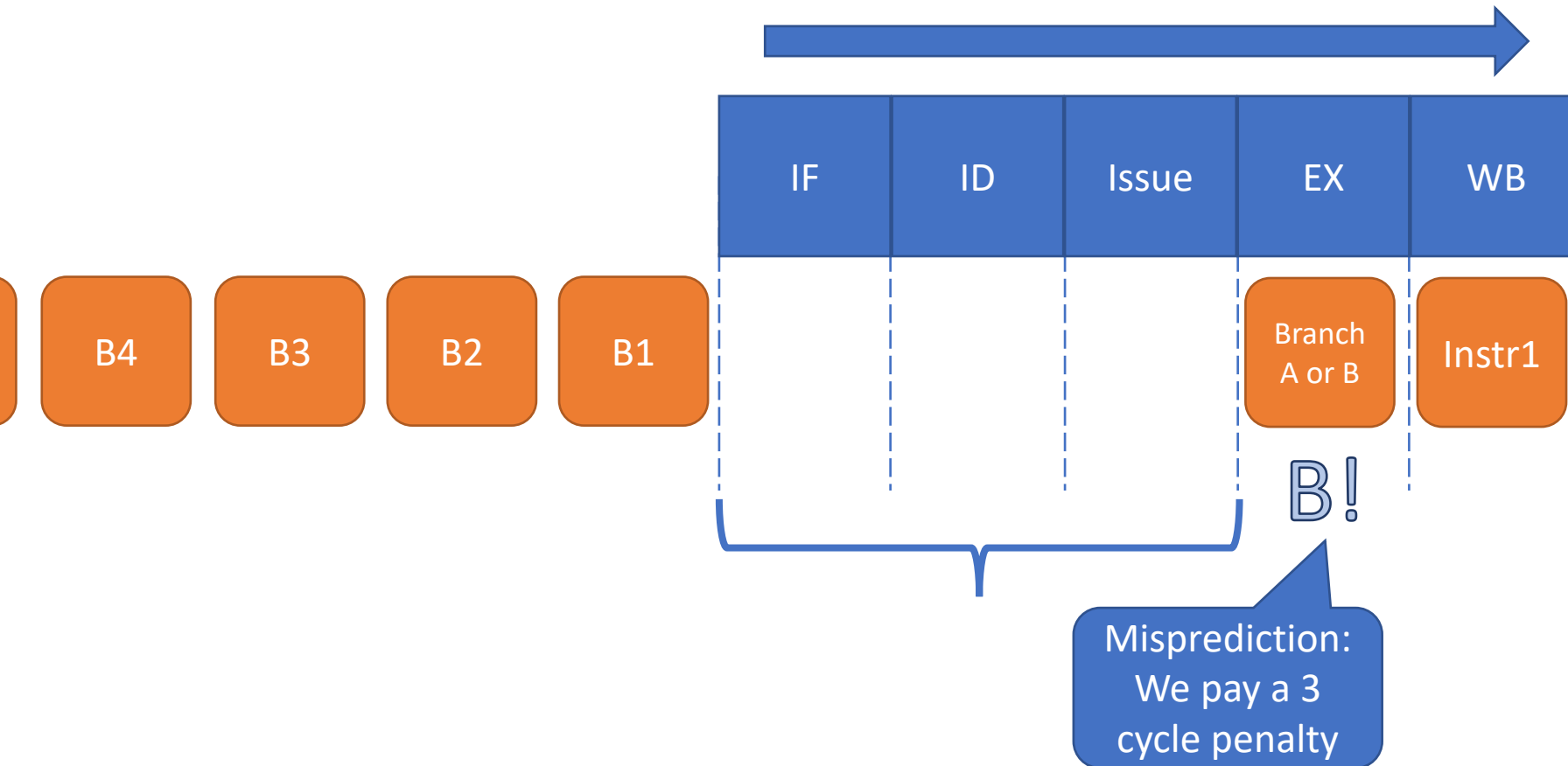
# Branch Prediction



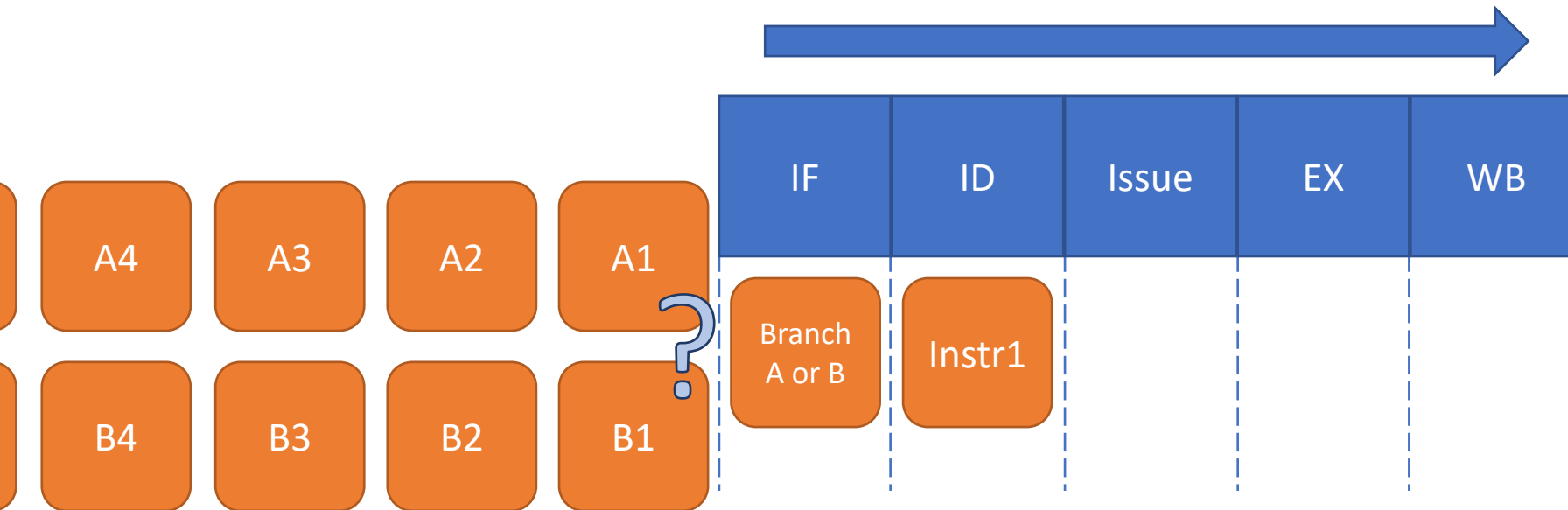
# Branch Prediction



# Branch Prediction



# The Goal of Branch Prediction



How to make a good prediction?

# Outline

## 1. Branch Prediction

## 2. Paper Summary

### 1. Executive Summary

### 2. Background & Related Work

### 3. Key Idea: Perceptron as Branch Predictor

### 4. Implementation

### 5. Results

### 6. Optimization: Hybrid Predictor

### 7. Final Results

### 8. Conclusion

## 3. Analysis and Discussion

# Executive Summary

## Problem

Branch predictor's **accuracy** is central for performance

## Goal

Create a **branch predictor that is superior** to state of the art

## Key Idea

**Perceptrons are efficient** in learning and predicting certain patterns over long histories

## Mechanism

Build a branch predictor with **perceptrons as central decision makers**

Implement a hybrid perceptron / conventional predictor

## Results

Perceptron: **14.7% improvement** over baseline for a fixed 4K HW budget

Hybrid Predictor: **Outperforms baseline** consistently across benchmarks

# Outline

1. Branch Prediction
2. Paper Summary
  1. Executive Summary
  2. Background & Related Work
  3. Key Idea: Perceptron as Branch Predictor
  4. Implementation
  5. Results
  6. Optimization: Hybrid Predictor
  7. Final Results
  8. Conclusion
3. Analysis and Discussion

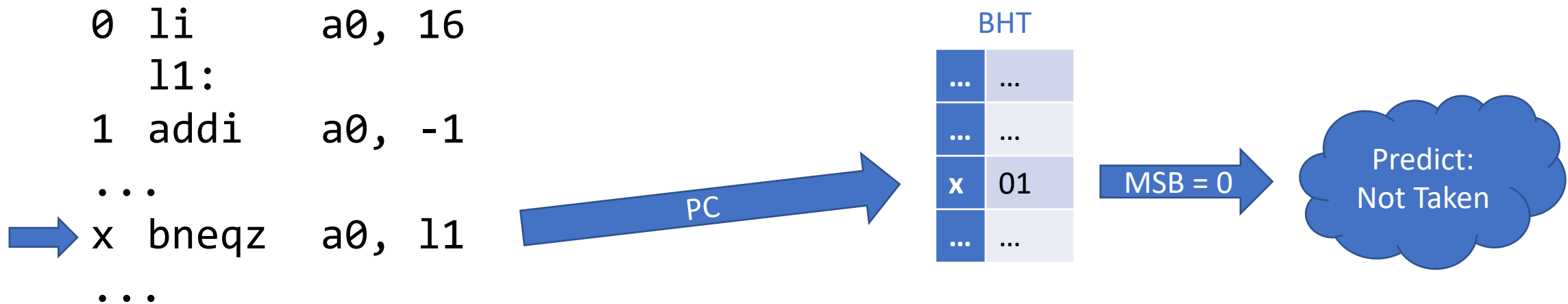
# Static Branch Prediction

- Idea: Branches are mostly *biased*
- e.g. *Backward Taken, Forward Not Taken*
- Let compiler profile and arrange code
- Trivial to implement in HW
- Branch behaviour often not known at compile time
- Only efficient for heavily biased branches

```
li      a0, 16
l1:
addi    a0, -1
...
bnez    a0, l1
...
```

6.25%  
Misprediction  
Rate

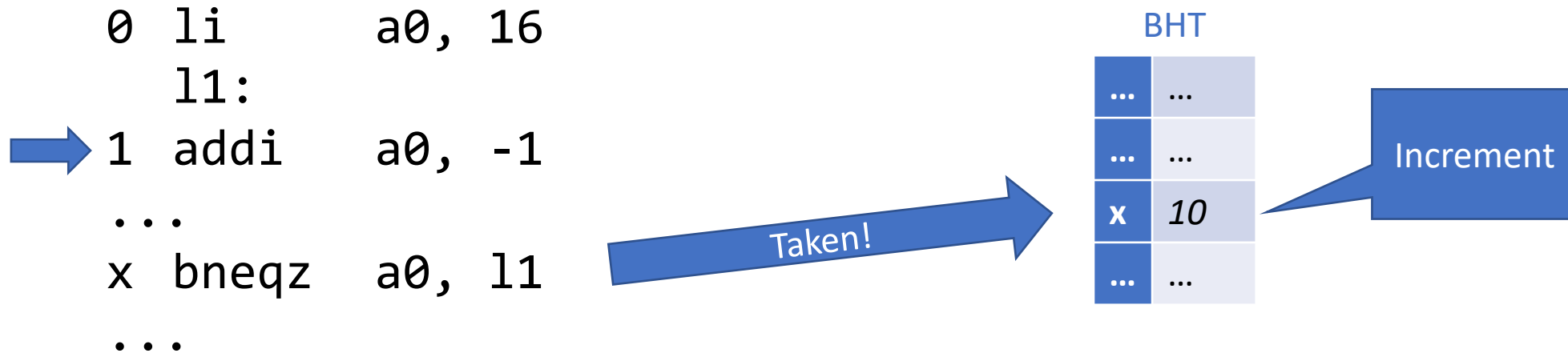
# Branch History Table (BHT)



Prediction:

1. Index BHT with branch address
2. Take branch if MSB of BHT entry equals 1

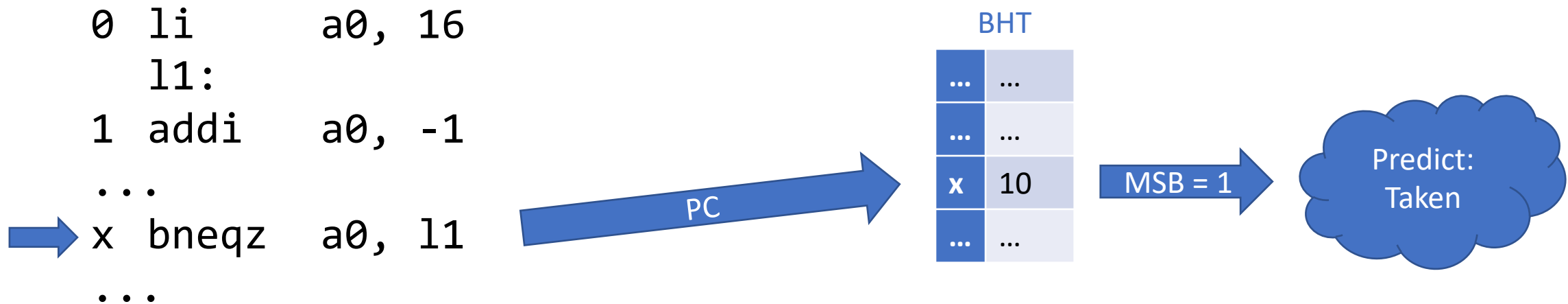
# Branch History Table (BHT)



Update (after branch is resolved):

1. Increment BHT entry if branch was taken, decrement otherwise

# Branch History Table (BHT)



Prediction:

1. Index BHT with branch address
2. Take branch if MSB of BHT entry equals 1

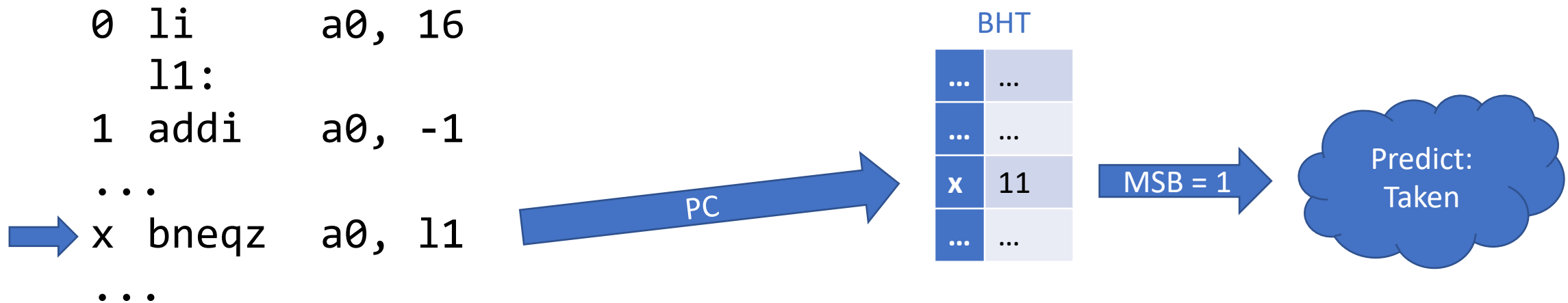
# Branch History Table (BHT)



Update (after branch is resolved):

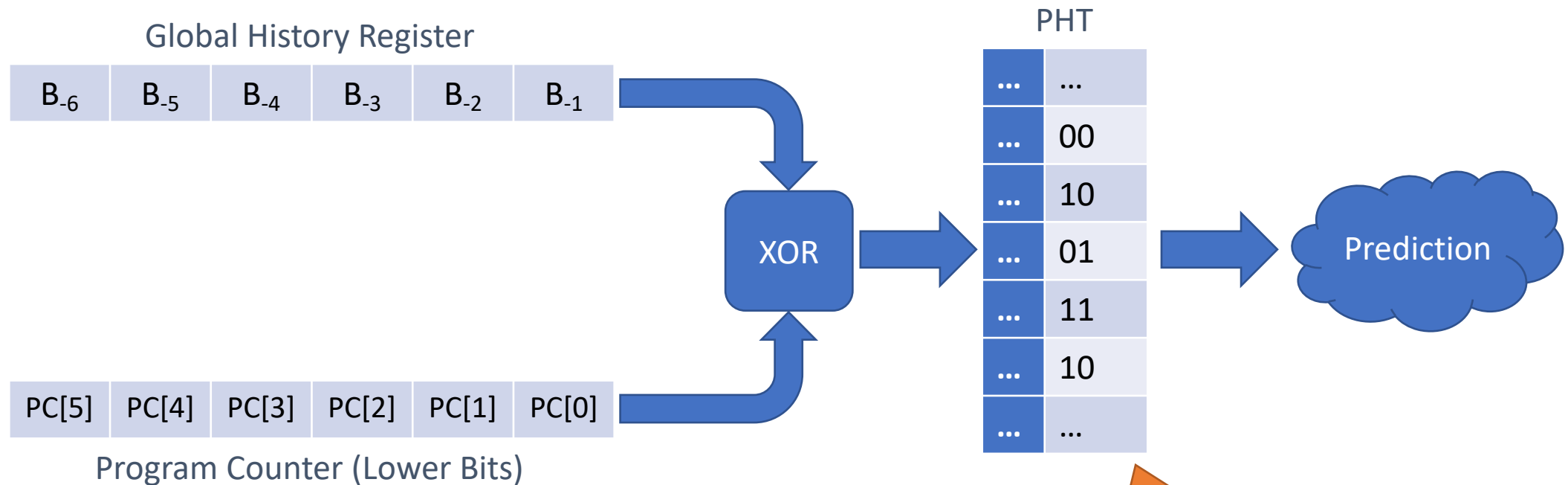
1. Increment BHT entry if branch was taken, decrement otherwise

# Branch History Table (BHT)



- Very Simple Design
- Not context-aware

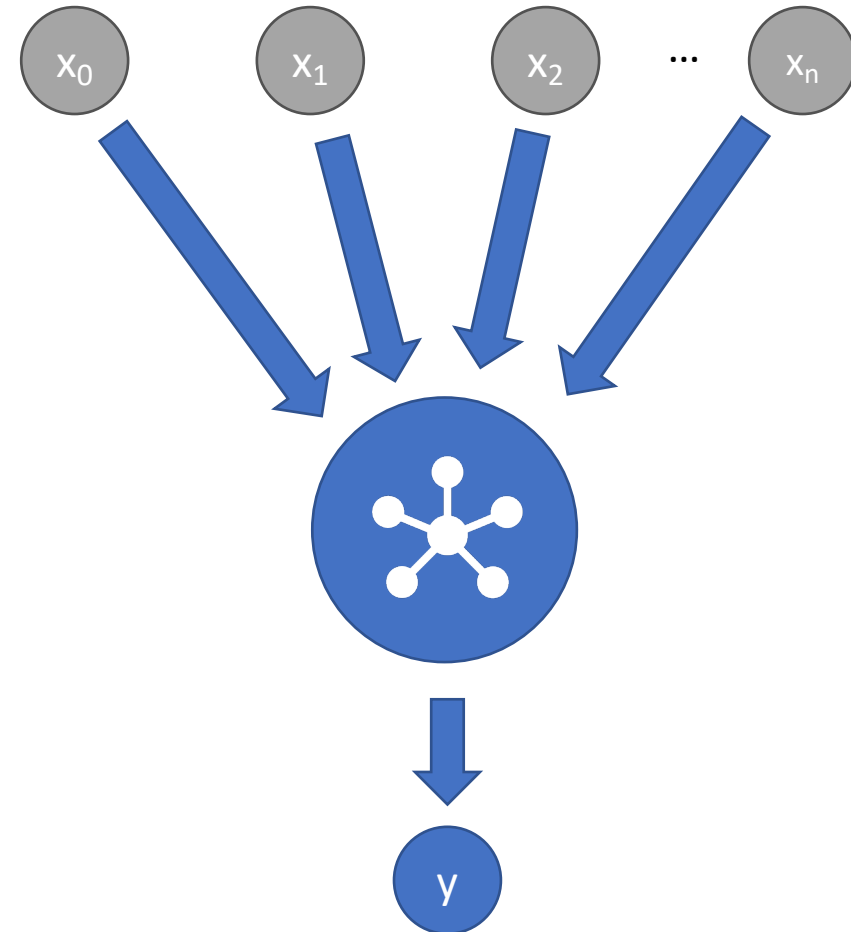
# Pattern History Table (PHT): *gshare*



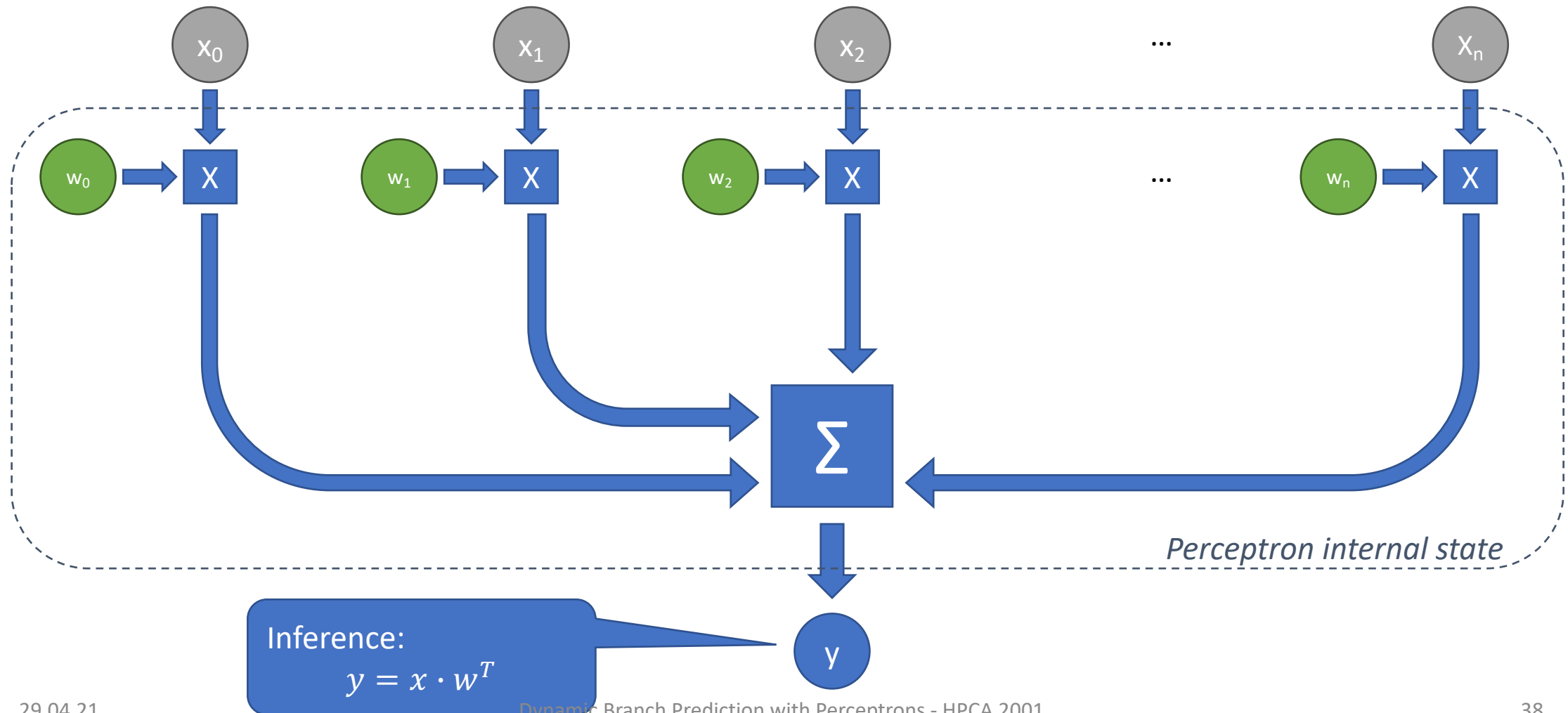
Problem: PHT scales exponentially with history length

# Perceptron

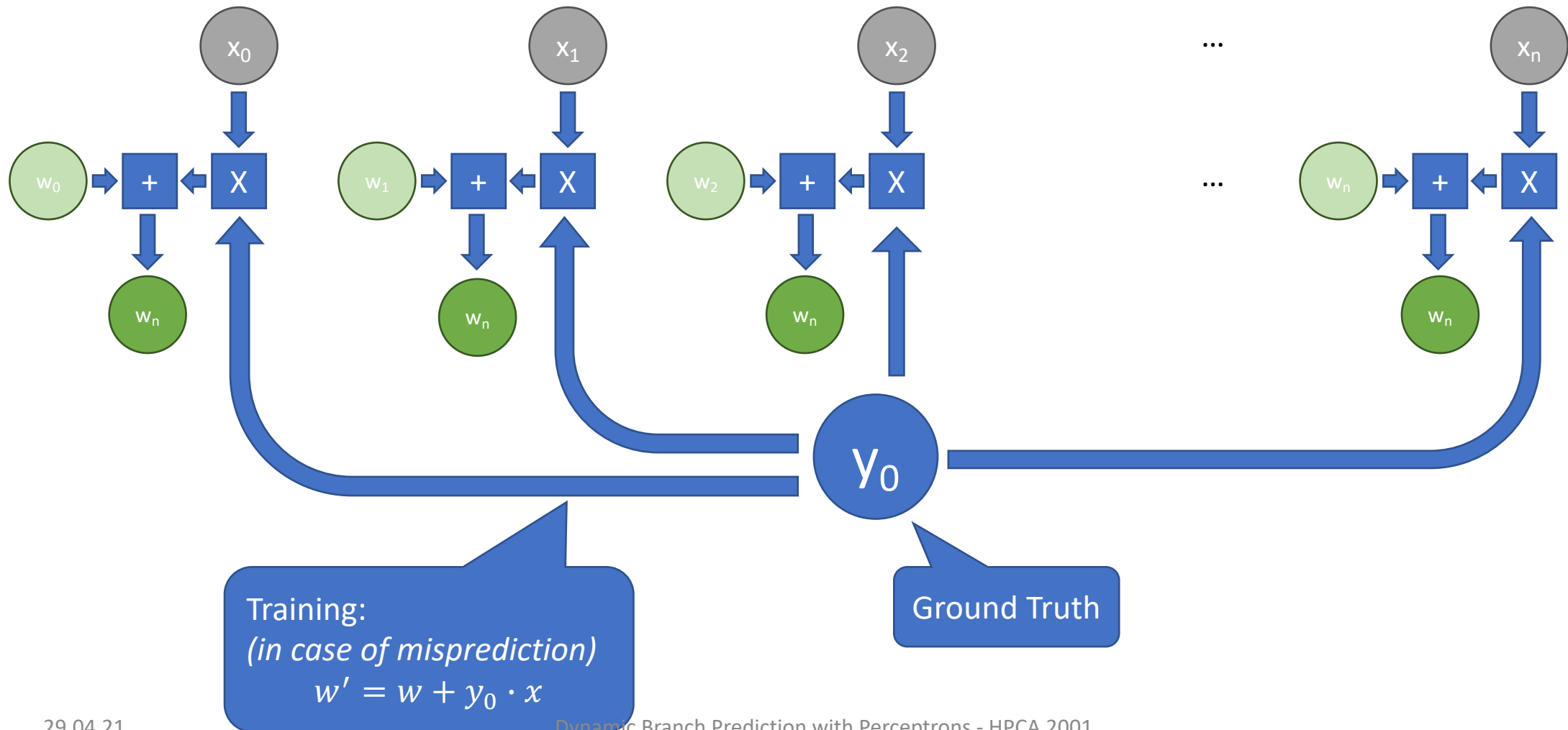
- Takes set of *features*
- Makes *prediction* based on features (*Inference*)
- Improves over time (*training*)



# Perceptron: Inference



# Perceptron: Training

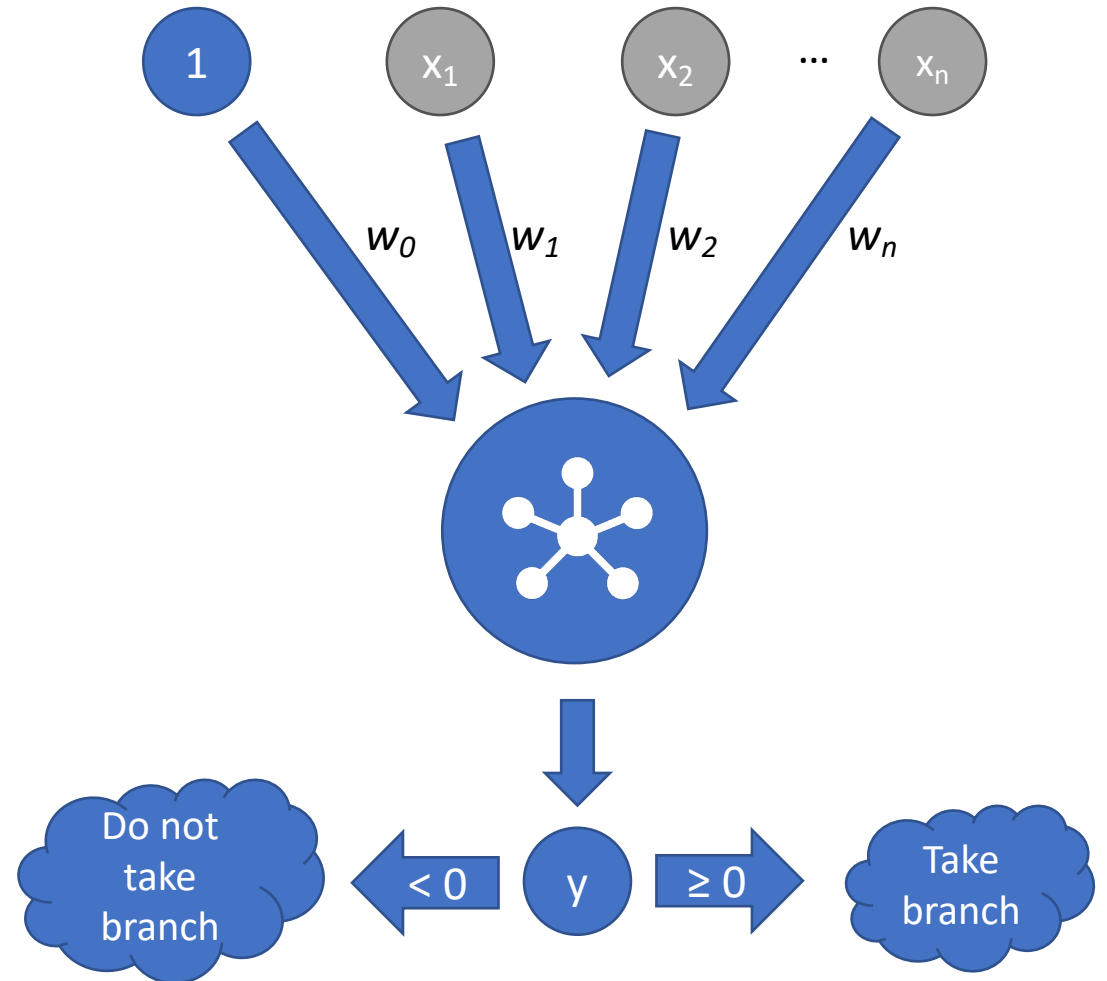


# Outline

1. Branch Prediction
2. Paper Summary
  1. Executive Summary
  2. Background & Related Work
  3. Key Idea: Perceptron as Branch Predictor
  4. Implementation
  5. Results
  6. Optimization: Hybrid Predictor
  7. Final Results
  8. Conclusion
3. Analysis and Discussion

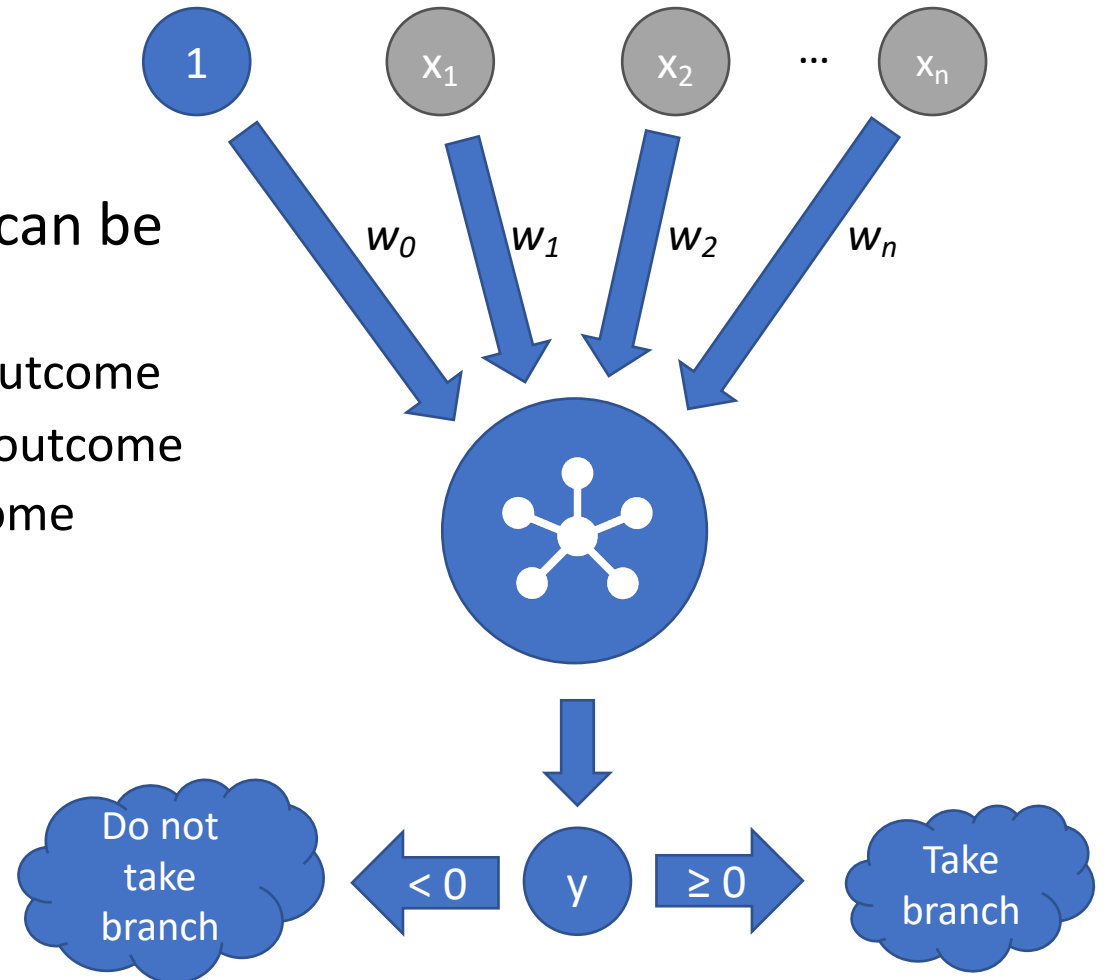
# Perceptron as Branch Predictor

- One perceptron per branch location
- Use *Global History Register* as input
  - Entries are either -1 (not taken) or 1 (taken)
  - Constant 1 input for branch bias
- If  $y \geq 0$  *take* branch, else do *not take*
- Once branch outcome is known, *dynamically train* in case of misprediction



# Perceptron as Branch Predictor: Observations

- Perceptron output  $y$  contains *confidence* about decision
- Each weight corresponds to an input and can be interpreted as follows:
  - $w_n \gg 0$ : Positive correlation to  $-n$ th branch outcome
  - $w_n \ll 0$ : Negative correlation to  $-n$ th branch outcome
  - $w_n \approx 0$ : Little correlation to  $-n$ th branch outcome

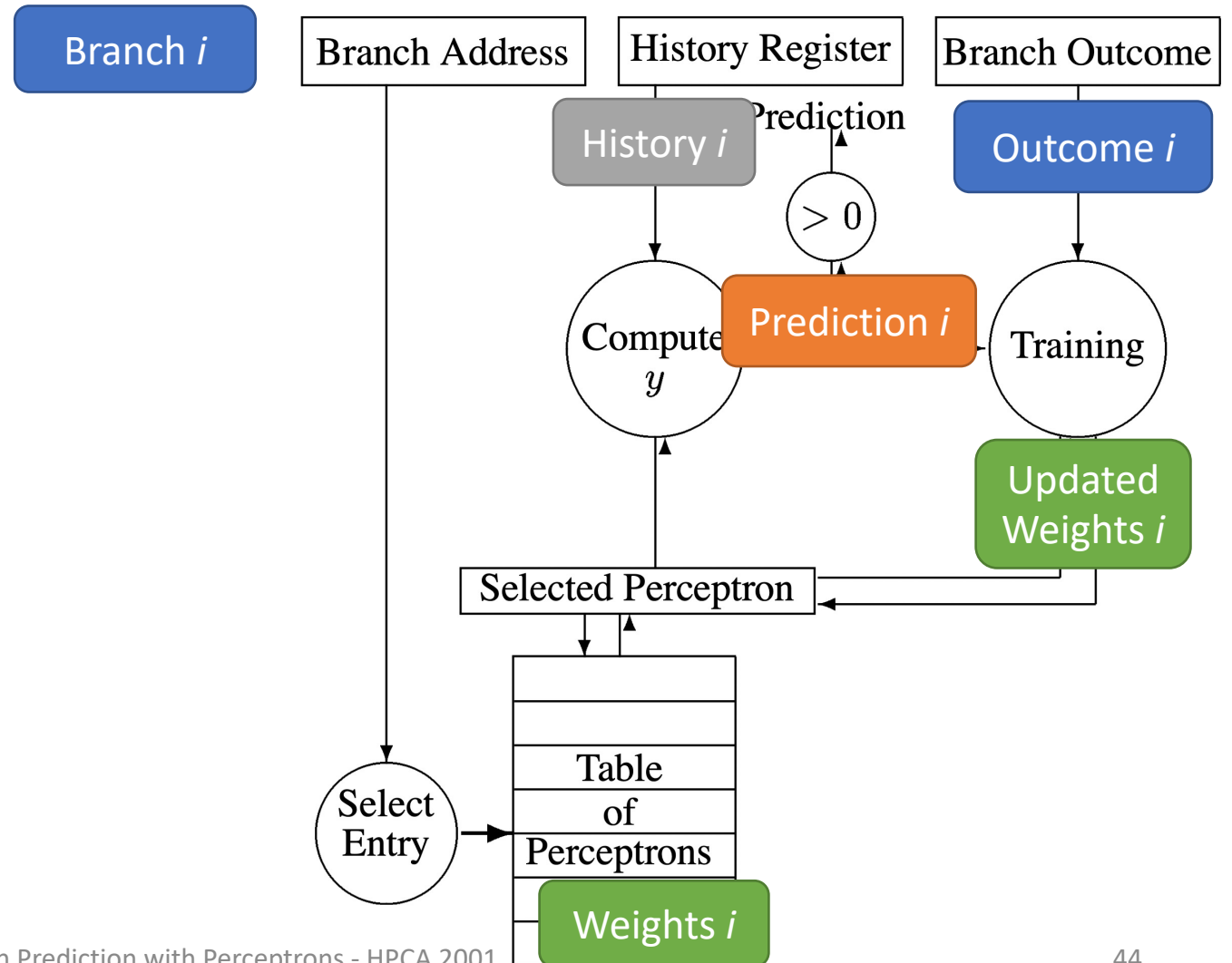


# Outline

1. Branch Prediction
2. Paper Summary
  1. Executive Summary
  2. Background & Related Work
  3. Key Idea: Perceptron as Branch Predictor
  4. Implementation
  5. Results
  6. Optimization: Hybrid Predictor
  7. Final Results
  8. Conclusion
3. Analysis and Discussion

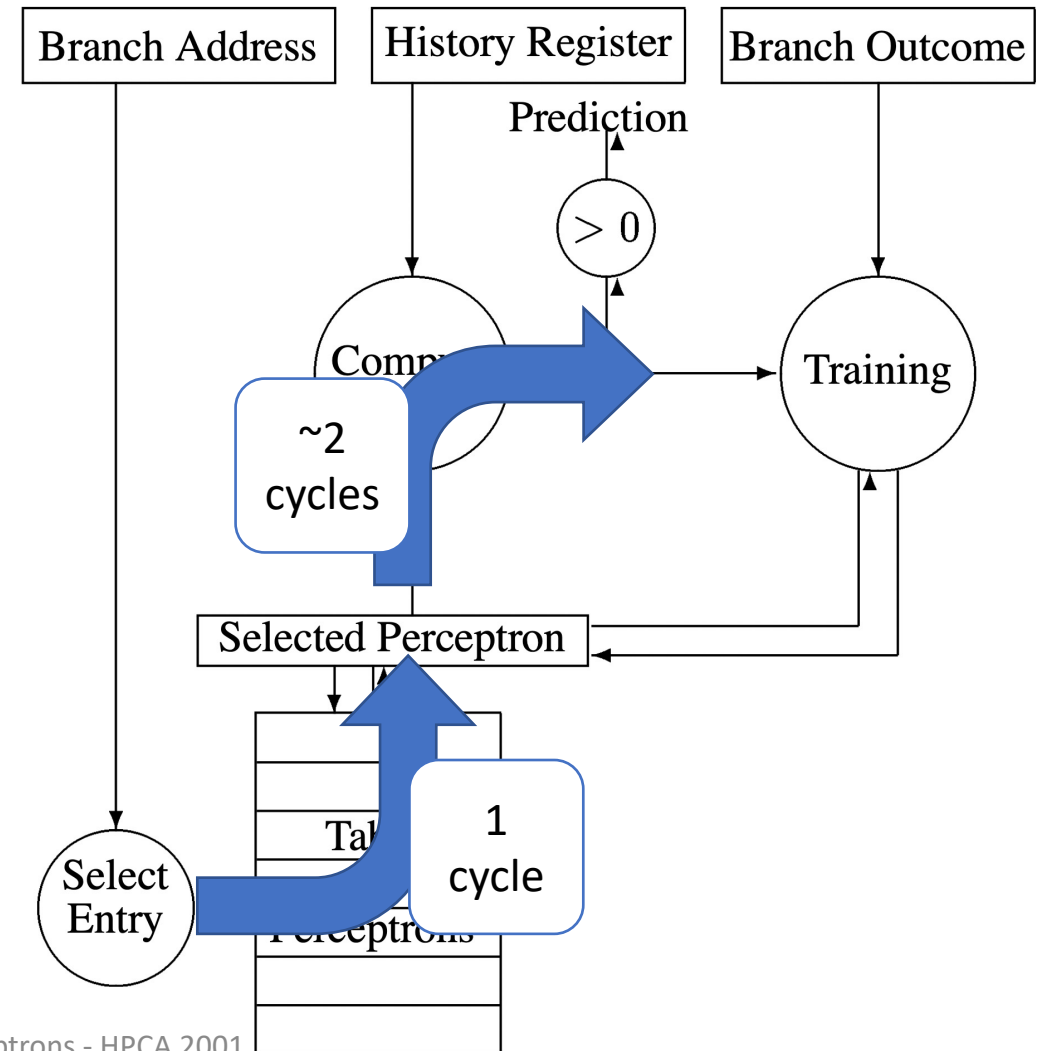
# Putting it Together

1. *Hash branch address to select perceptron.*
2. *Fetch weights of selected perceptron from SRAM.*
3. *Compute  $y$  from weights and global history.*
4. Take branch if  $y > 0$ .
5. *Train perceptron after branch is resolved.*
6. *Write back perceptron.*



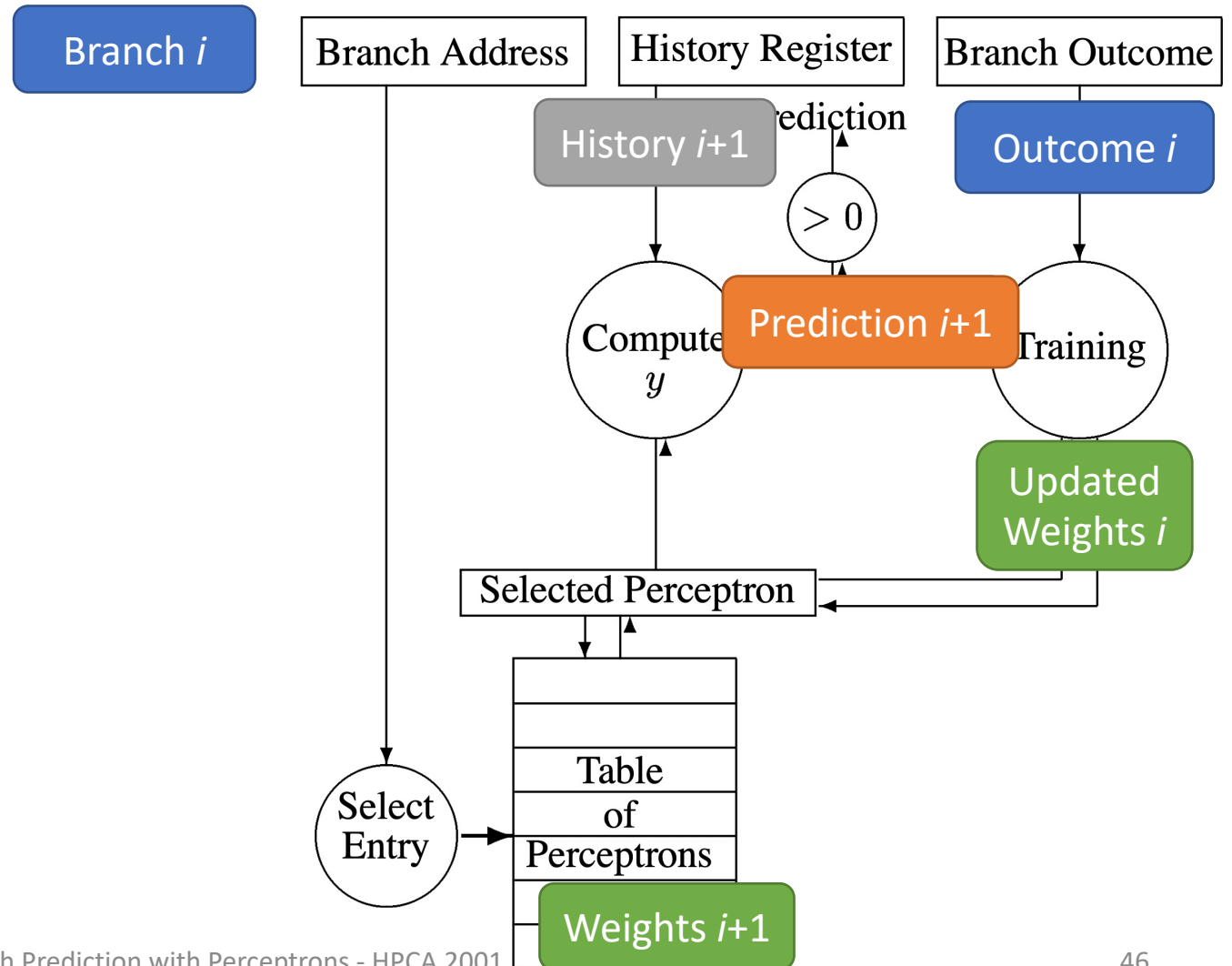
# Delay

- 1 cycle to access SRAM
- Approx. 2 cycles for inference
- Approx. **3 cycles** from branch address to prediction.
- We do not want to wait that long!



# Pipelined Operation

1. When branch is encountered, use prediction from *previous* iteration.
2. After branch is resolved, *train and write back* perceptron.
3. Update global history register, concatenate outcome with branch address, select next entry.
4. *Fetch weights* of selected perceptron from SRAM.
5. Make prediction for next branch.



# Outline

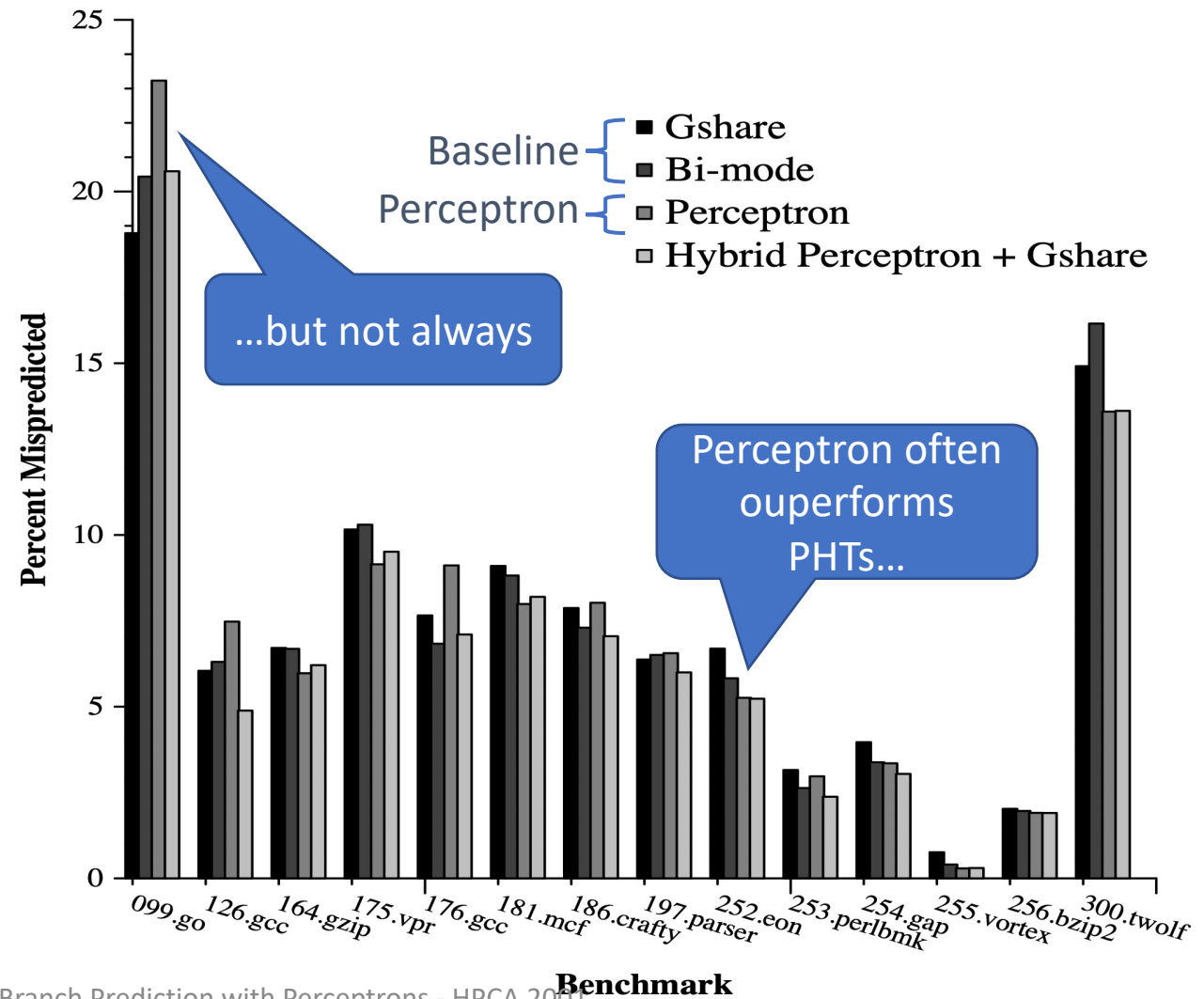
1. Branch Prediction
2. Paper Summary
  1. Executive Summary
  2. Background & Related Work
  3. Key Idea: Perceptron as Branch Predictor
  4. Implementation
  5. Results
  6. Optimization: Hybrid Predictor
  7. Final Results
  8. Conclusion
3. Analysis and Discussion

# Methodology

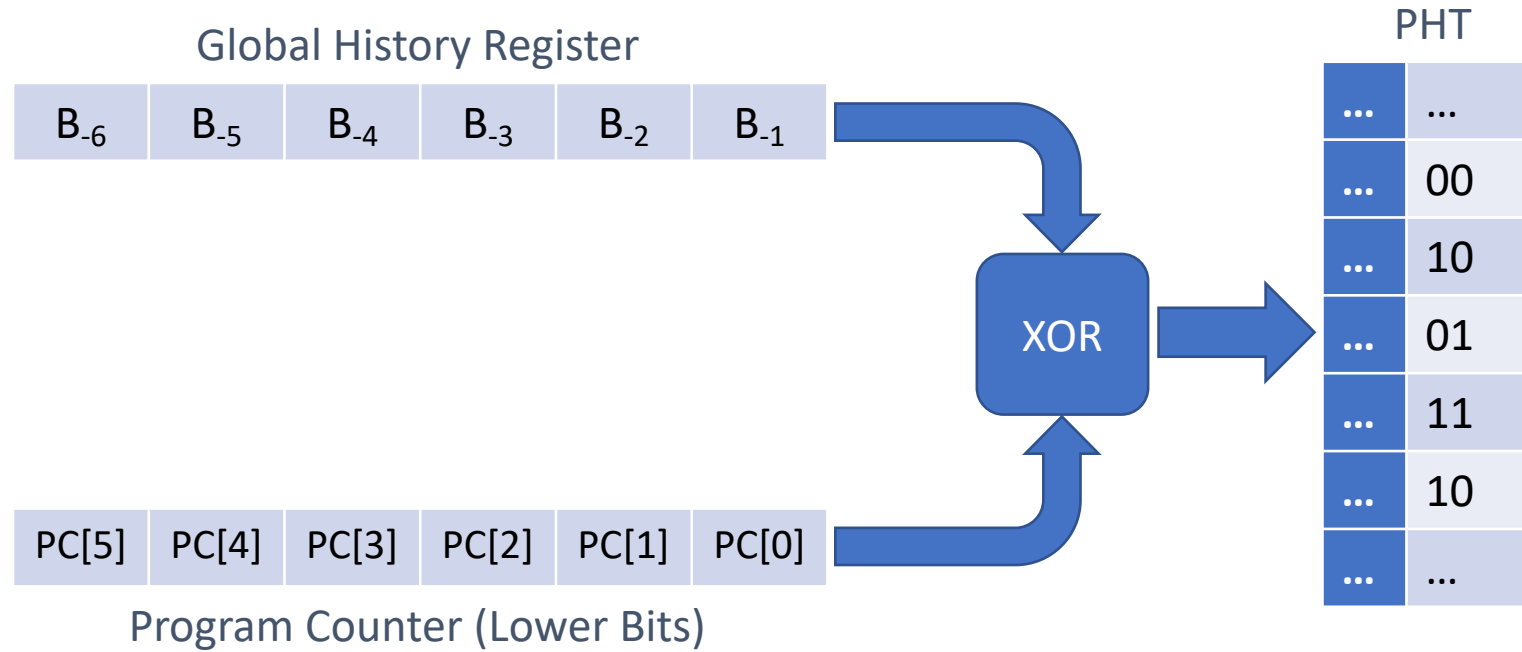
- Baseline: gshare, bi-mode (**PHT predictors**)
- Choose **fixed HW budget** (4KiB in the following)
- **Generate traces** of SPEC95 and SPEC2000 benchmarks
- **Simulate** branch predictors' performance for benchmark traces

# Misprediction Rates

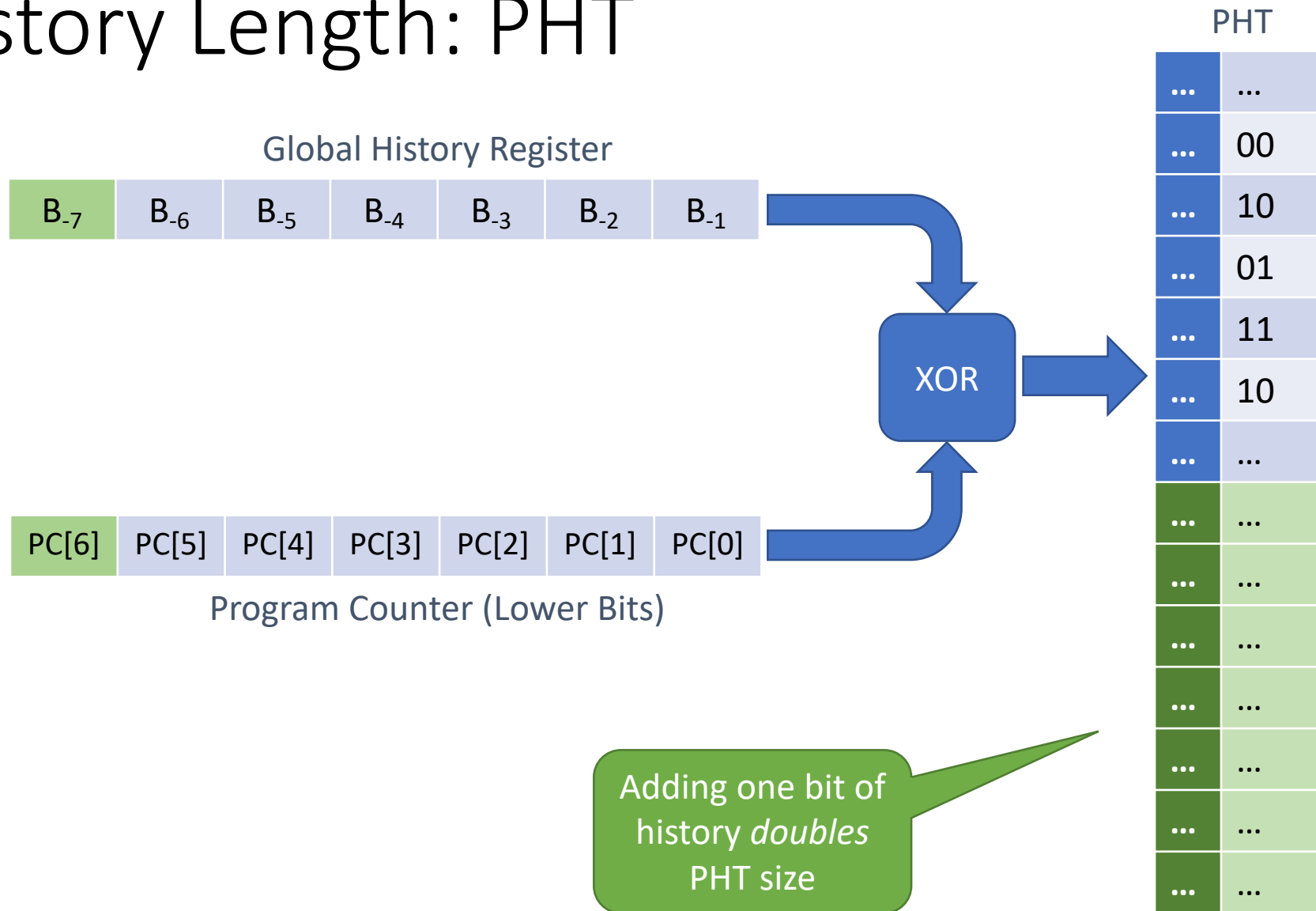
Why does the perceptron perform well?



# History Length: PHT



# History Length: PHT



# History Length: Perceptron

$$\begin{pmatrix} 1 \\ B_{-1} \\ B_{-2} \\ B_{-3} \\ B_{-4} \\ B_{-5} \\ B_{-6} \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{pmatrix}^T = y$$

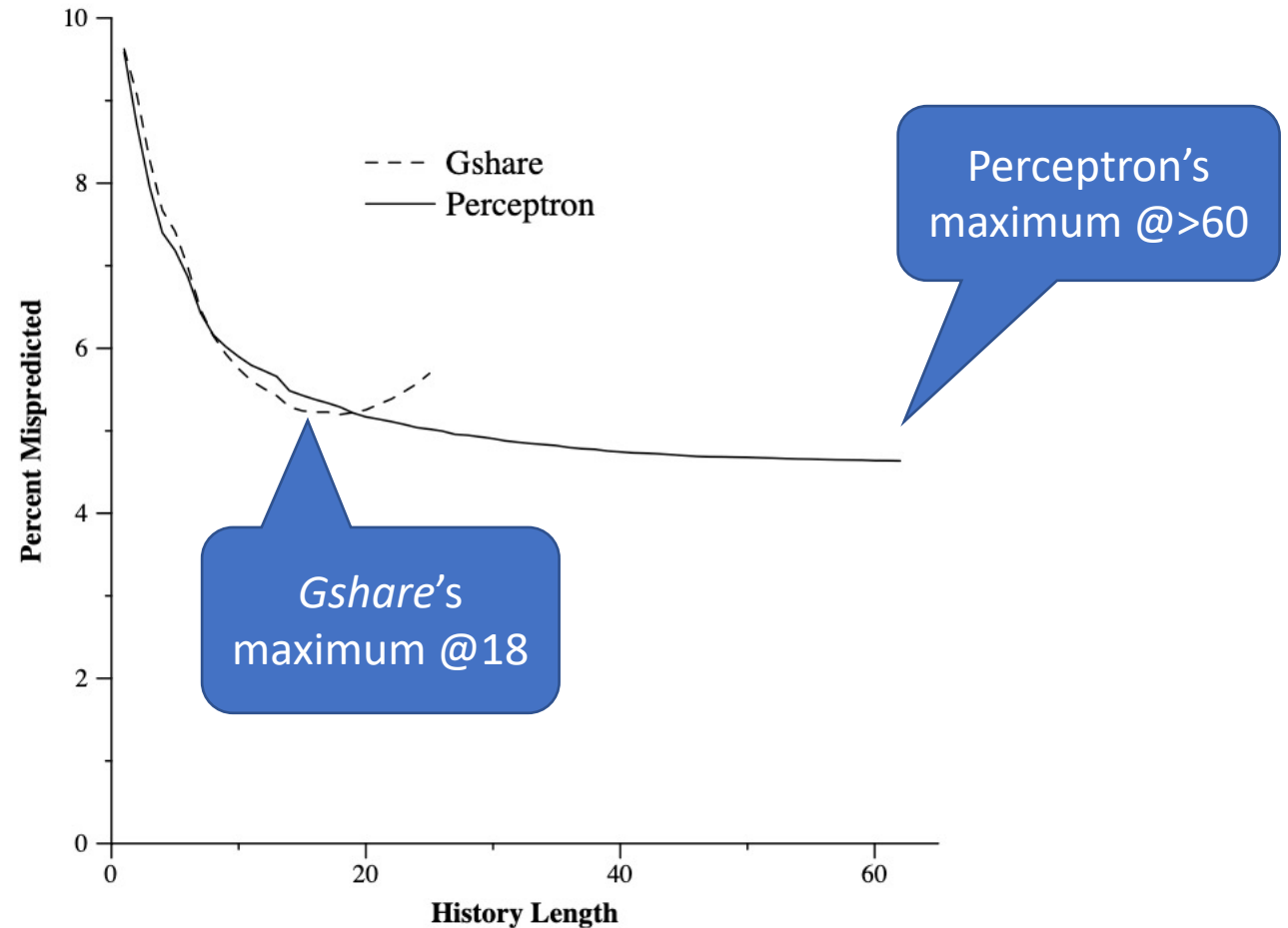
# History Length: Perceptron

$$\begin{pmatrix} 1 \\ B_{-1} \\ B_{-2} \\ B_{-3} \\ B_{-4} \\ B_{-5} \\ B_{-6} \\ B_{-7} \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \end{pmatrix}^T = y$$

Adding one bit of  
history adds  
*single* weight

# History Length: Supporting Experiment

- Same HW budget for both predictors
- Measure performance over different history lengths
- Perceptron can score for large history lengths

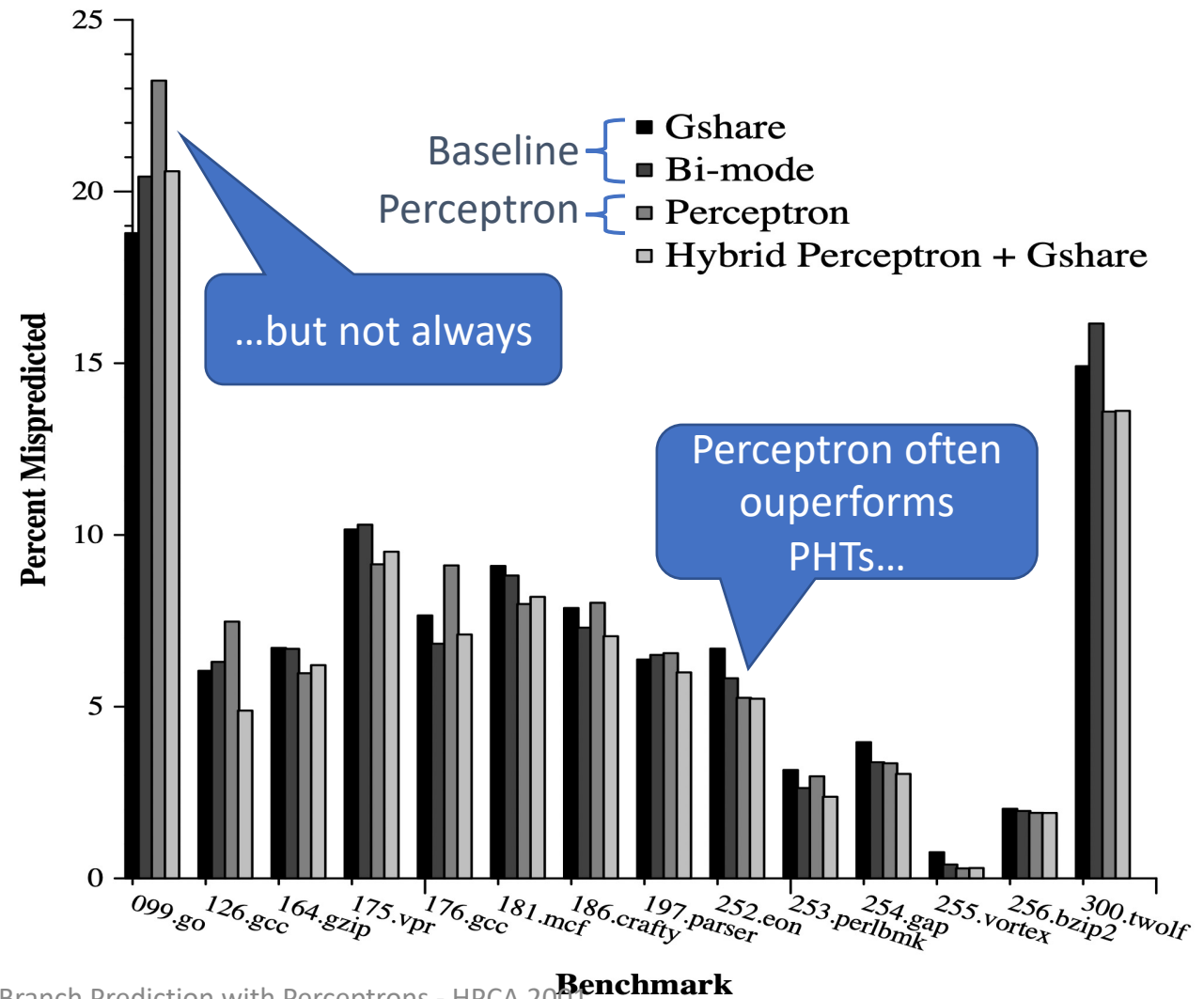


# Misprediction Rates

Why does the perceptron perform well?

→ *Efficiently captures long histories*

When does the perceptron perform well?



# Linear Separability

$a, b \in \{0, 1\}$

B1: if (a) {...}  
 B2: if (!b) {...}  
 B3: if (a|b) {...}

	B1: T	B1: N
B2: T	B3: T	B3: N
B2: N	B3: T	B3: T

Linearly separable

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

B1: if (a) {...}  
 B2: if (!b) {...}  
 B3: if (a==b) {...}

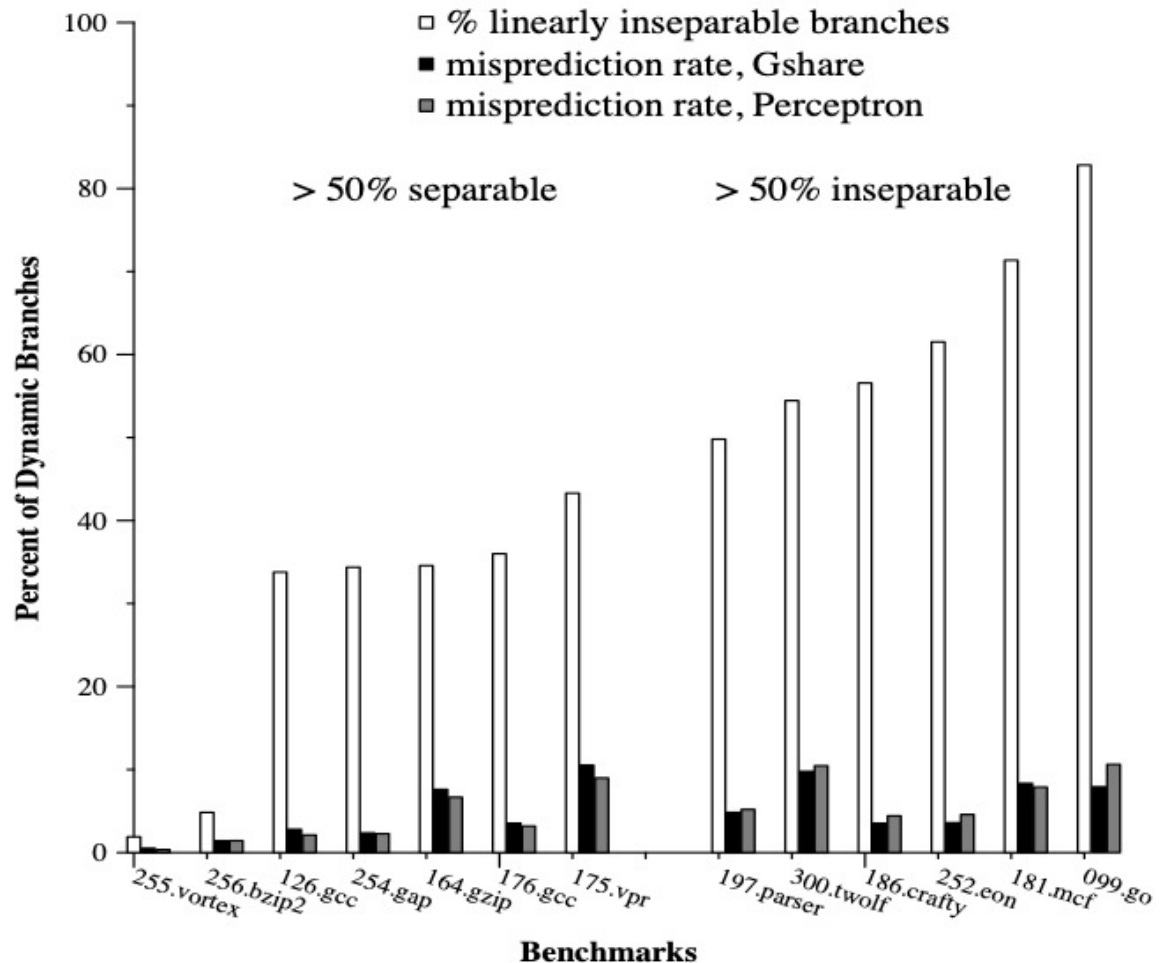
	B1: T	B1: N
B2: T	B3: N	B3: T
B2: N	B3: T	B3: N

Linearly inseparable

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix}$$

# Linear Separability: Supporting Experiment

- Sort benchmarks by share of linearly separable branches
- Perceptron outperforms Gshare on benchmarks with many linearly separable branches



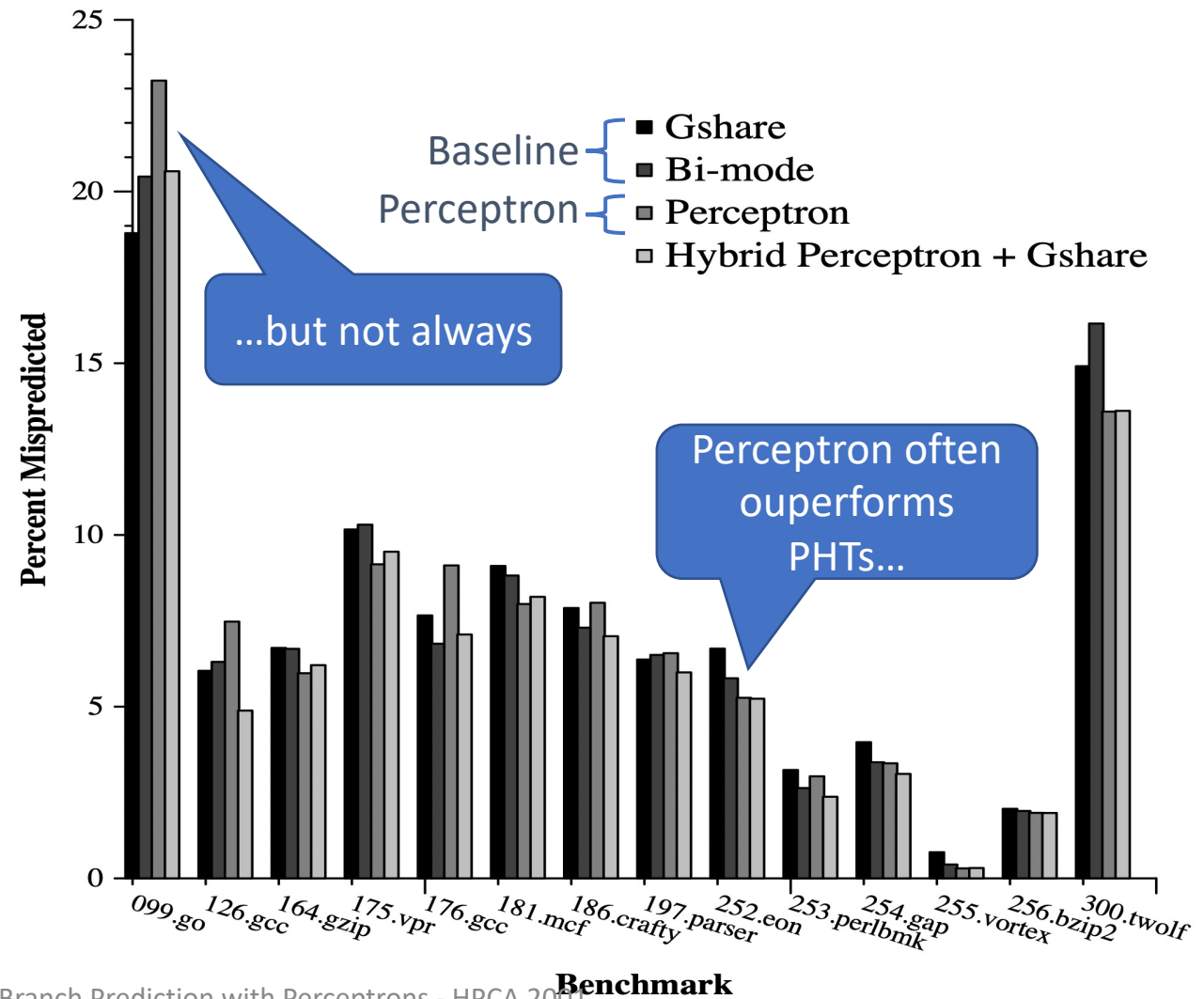
# Misprediction Rates

Why does the perceptron perform well?

→ *Efficiently captures long histories*

When does the perceptron perform well?

→ *When branch function is linearly separable*

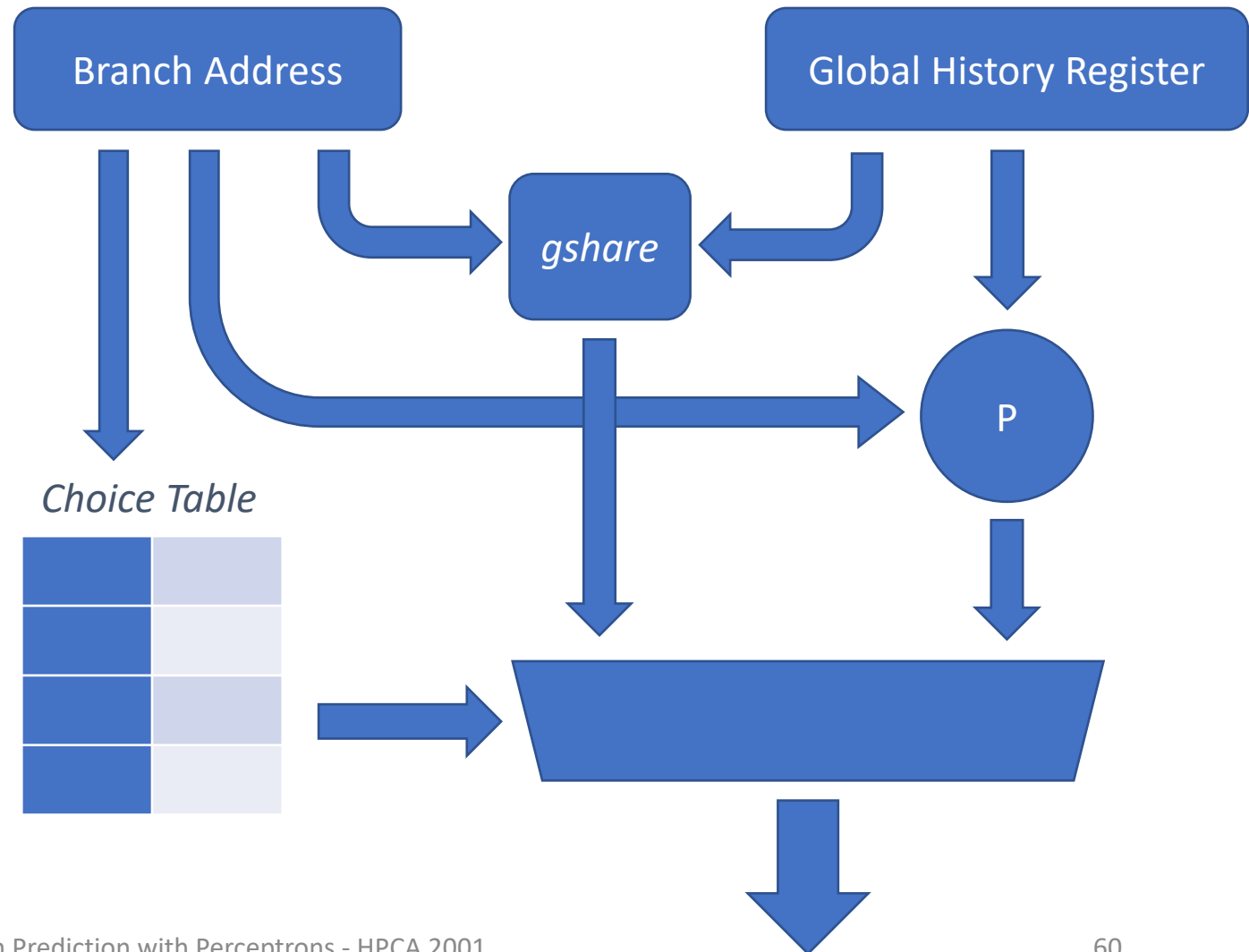


# Outline

1. Branch Prediction
2. Paper Summary
  1. Executive Summary
  2. Background & Related Work
  3. Key Idea: Perceptron as Branch Predictor
  4. Implementation
  5. Results
  6. Optimization: Hybrid Predictor
  7. Final Results
  8. Conclusion
3. Analysis and Discussion

# Hybrid Predictor

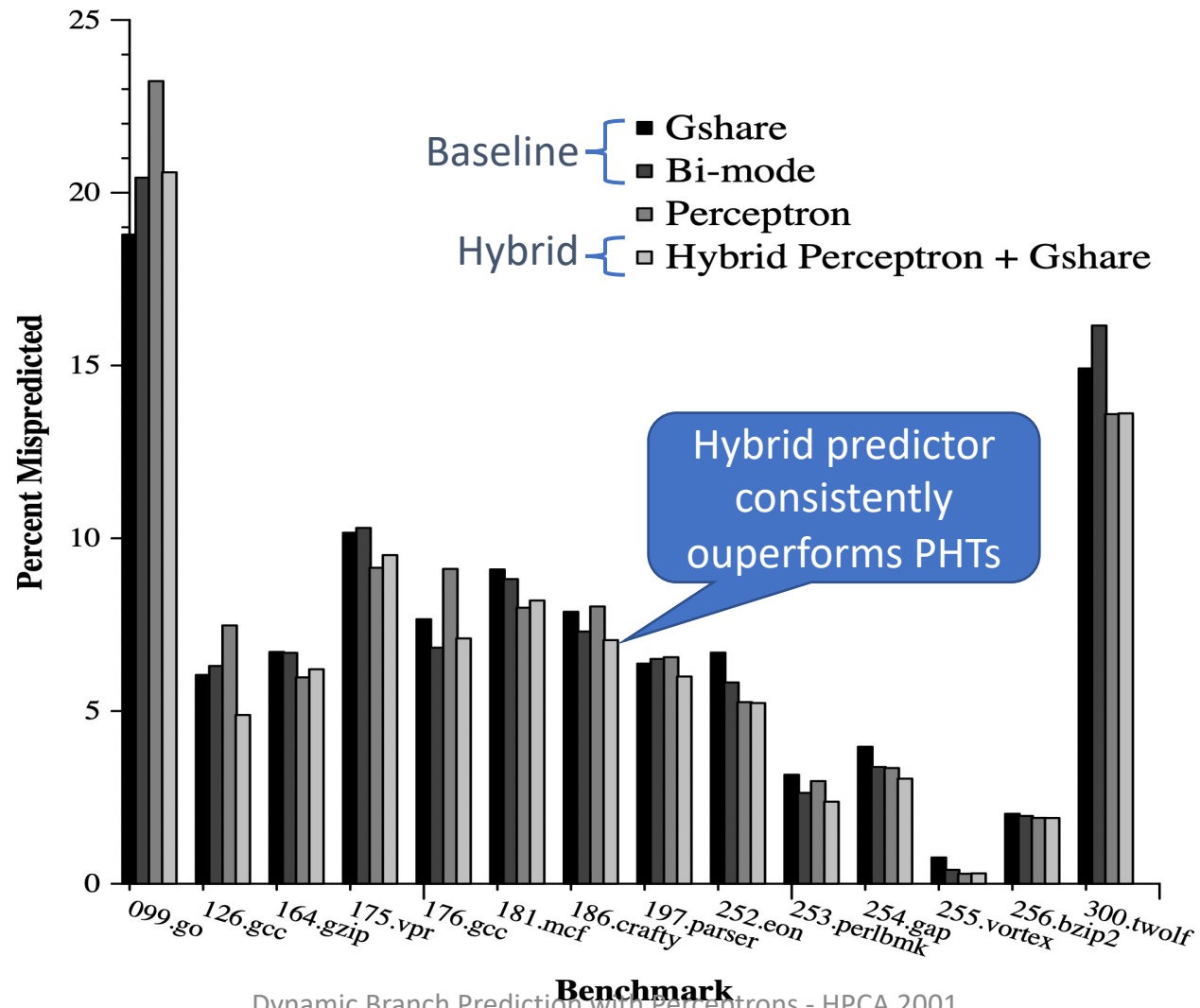
- Implement *both* perceptron and gshare
- Use a *choice table* to select predictor on branch



# Outline

1. Branch Prediction
2. Paper Summary
  1. Executive Summary
  2. Background & Related Work
  3. Key Idea: Perceptron as Branch Predictor
  4. Implementation
  5. Results
  6. Optimization: Hybrid Predictor
  7. Final Results
  8. Conclusion
3. Analysis and Discussion

# Misprediction Rates



# Outline

1. Branch Prediction
2. Paper Summary
  1. Executive Summary
  2. Background & Related Work
  3. Key Idea: Perceptron as Branch Predictor
  4. Implementation
  5. Results
  6. Optimization: Hybrid Predictor
  7. Final Results
  8. Conclusion
3. Analysis and Discussion

# Executive Summary

## Problem

Branch predictor's **accuracy** is central for performance

## Goal

Create a **branch predictor that is superior** to state of the art

## Key Idea

**Perceptrons are efficient** in learning and predicting certain patterns over long histories

## Mechanism

Build a branch predictor with **perceptrons as central decision makers**

Implement a hybrid perceptron / conventional predictor

## Results

Perceptron: **14.7% improvement** over baseline for a fixed 4K HW budget

Hybrid Predictor: **Outperforms baseline** consistently across benchmarks

# Outline

1. Branch Prediction
2. Paper Summary
  1. Executive Summary
  2. Background & Related Work
  3. Key Idea: Perceptron as Branch Predictor
  4. Implementation
  5. Results
  6. Optimization: Hybrid Predictor
  7. Final Results
  8. Conclusion
3. Analysis and Discussion

# Takeaways

- First dynamic branch predictor using a **neural network**
- **Performs very well** for class of *linearly separable* branches
- Hybrid predictor can efficiently **combine benefits** of PHT and perceptron

# Strengths

- Simple, straight-forward idea
- Effective and efficient solution
- Detailed performance analysis and discussion with lots of supporting evidence

# Weaknesses

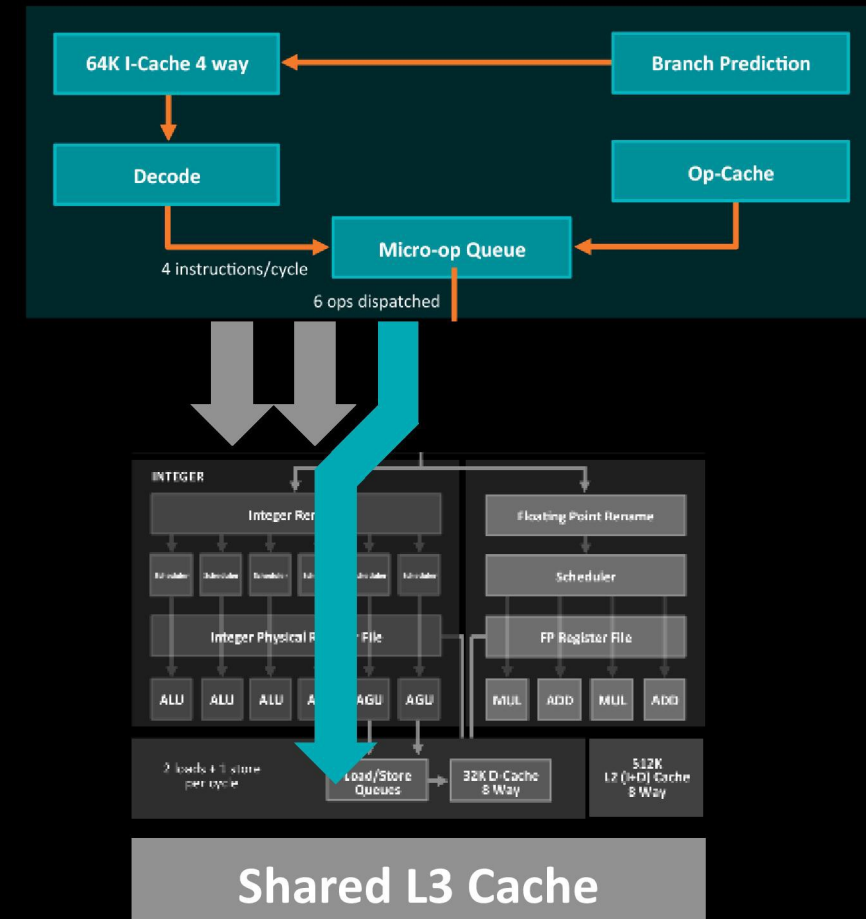
- Predictor performance strongly depends on benchmark. Might not always be an improvement.
- High latency of perceptron
- Weak area analysis
  - *“by examining die photos, we estimate [...]”*
- Weak timing analysis
  - Estimate by comparing to multiplier
  - *“we believe that a good implementation [...] will take no more than two clock cycles [...]”*

# Neural Net Prediction



## Scary Smart Prediction

- ▲ A true artificial network inside every “Zen” processor
- ▲ Builds a model of the decisions driven by software code execution
- ▲ Anticipates future decisions, pre-load instructions, choose the best path through the CPU



# Impact

- Perceptron branch predictor known to be **used in several modern processors** [1, 2]
- However, not many details are public
- This work is one **pillar of modern branch prediction research** [3]

[1] <https://www.anandtech.com/show/5831/amd-trinity-review-a10-4600m-a-new-hope>

[2] <https://www.anandtech.com/show/10907/amd-gives-more-zen-details-ryzen-34-ghz-nvme-neural-net-prediction-25-mhz-boost-steps>

[3] <https://www.jilp.org/cbp2016/>

*Questions / Comments?*

*How could we improve the perceptron predictor so that it works for more types of branches?*

*Do you see further usecases for neural networks in processor microarchitecture?*