

Advanced branch prediction algorithms

Ryan Gabrys

Ilya Kolykhmatov

Context

- Branches are frequent: 15-25 %
- A branch predictor allows the processor to speculatively fetch and execute instructions down the predicted path
- Predictor accuracy is more important for deeper pipelines
 - Pentium 4 with Prescott core pipeline has 31 stages
 - A lot of cycles can be wasted on misprediction:
 - No speculative state may commit
 - Squash instructions in the pipeline
 - Must not allow stores in the pipeline to occur
 - Need to handle exceptions appropriately
 - Pentium III branch penalties:
 - Not Taken: no penalty
 - Correctly predicted taken: 1 cycle
 - Mispredicted: at least 9 cycles, as many as 26, average 10-15 cycles

Branch prediction schemes

Static

- predict at compile time
- based on instruction type or profilings
- work well on easy-to-predict branch such as for-loop

Dynamic

- predict at run time
- using hardware to track run-time behavior
- **more accurate**
- require more die area and power dissipation



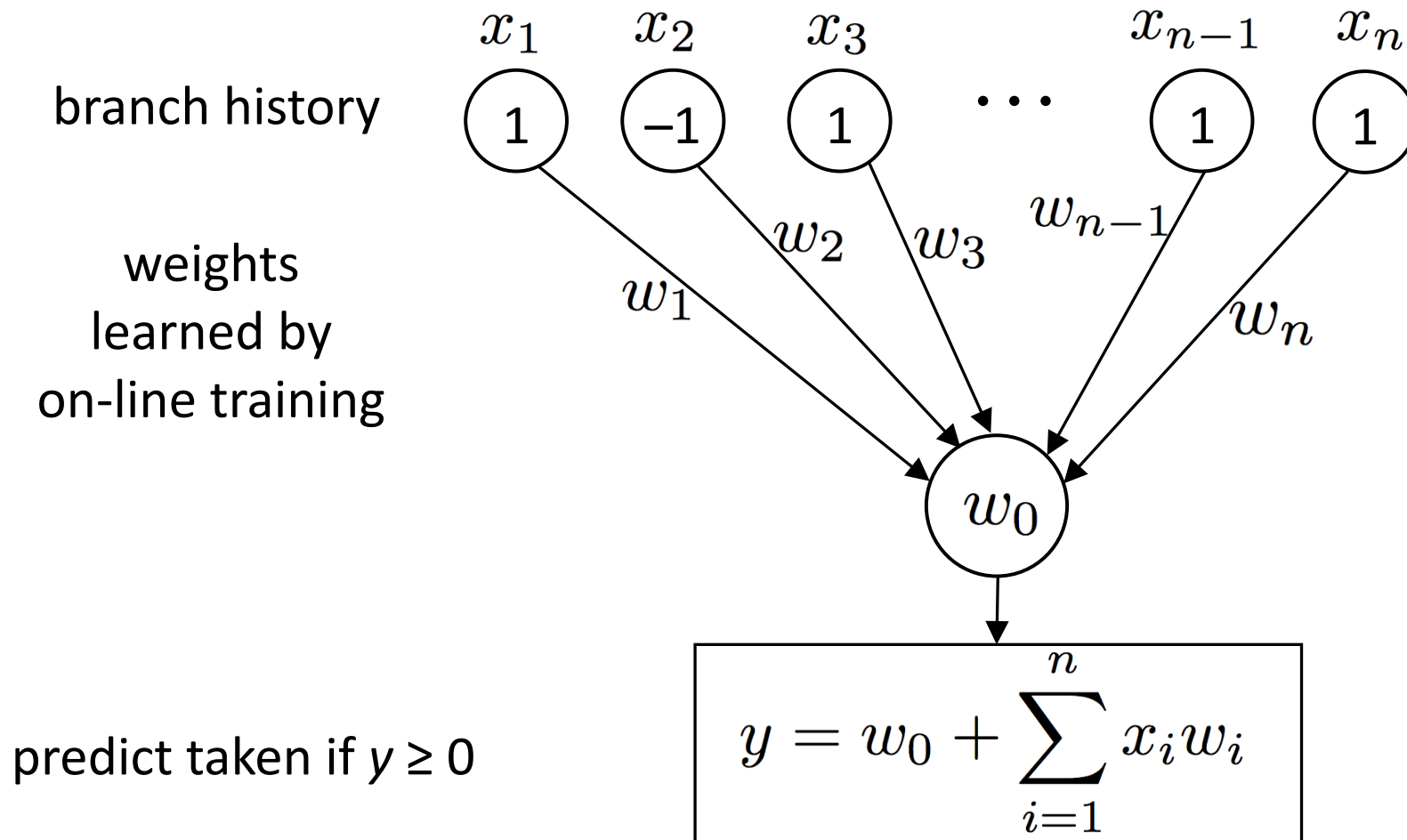
Dynamic branch prediction with perceptrons²⁰⁰¹

Daniel A. Jimenez and Calvin Lin

Conditional branch prediction as a machine learning problem

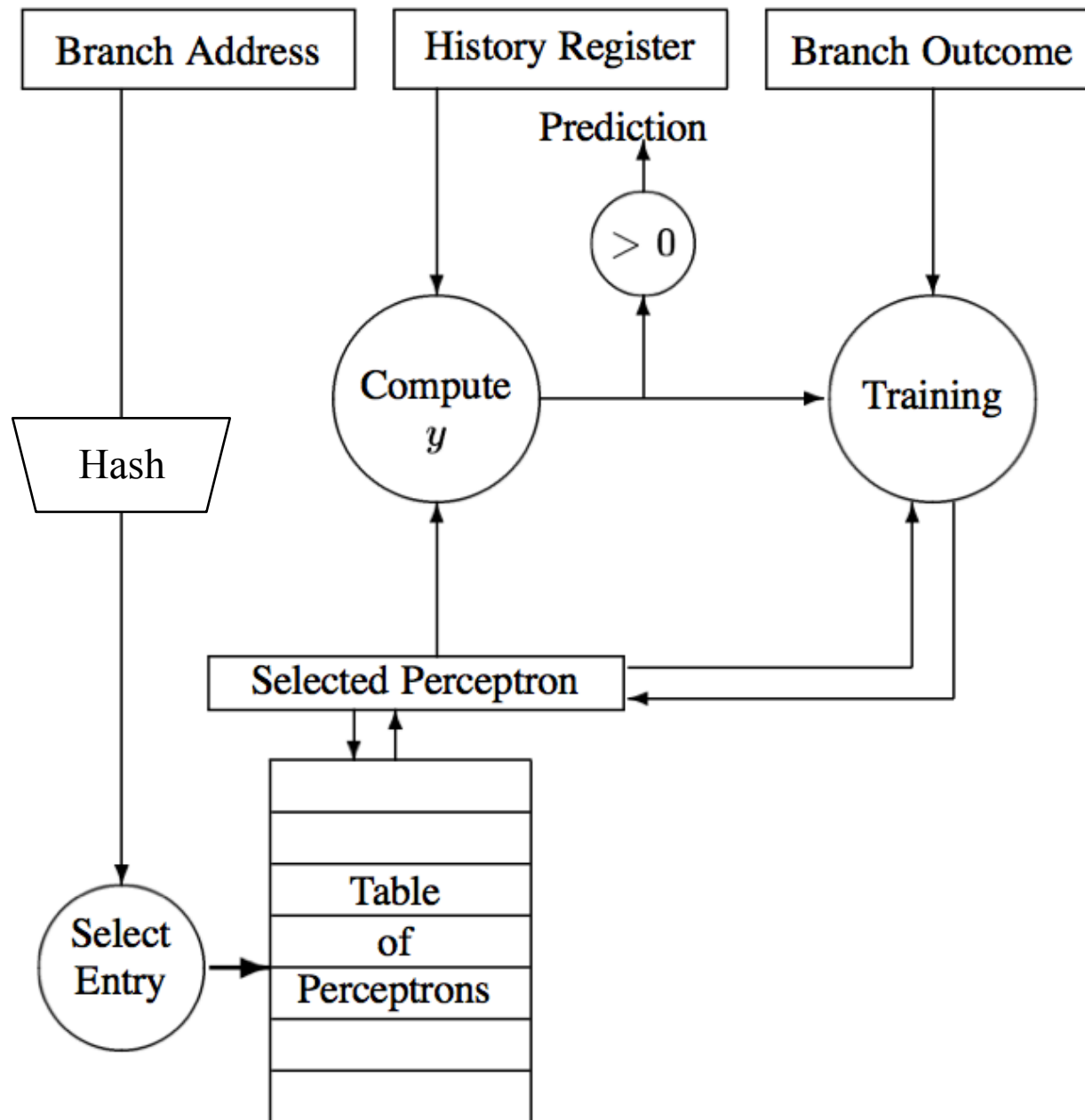
- The machine learns to predict conditional branches
- So why not apply a machine learning algorithm?
- Artificial neural networks
 - Simple model of neural networks in brain cells
 - Learn to recognize and classify patterns
- Perceptron – simplest neural network with better accuracy than any previously known predictor

Branch-predicting perceptron



- Training finds correlations between history and outcome

Organization of the perceptron predictor



Training algorithm

$x_{1..n}$ is the n -bit history register, x_0 is 1.

$w_{0..n}$ is the weights vector.

t is the Boolean branch outcome.

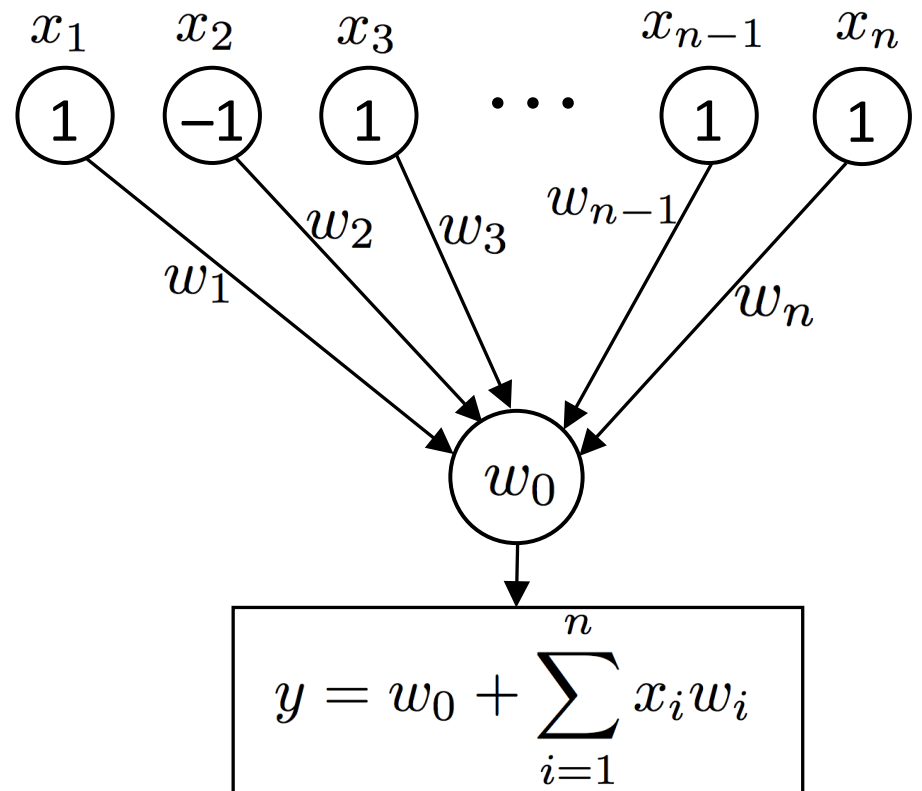
θ is the training threshold.

```
if sign( $y_{out}$ )  $\neq t$  or  $|y_{out}| \leq \theta$  then
    for  $i := 0$  to  $n$  do
         $w_i := w_i + tx_i$ 
    end for
end if
```

What do the weights mean?

Correlating weights $w_1, \dots, w_n$:

- w_i is proportional to the probability that the predicted branch agrees with the i^{th} branch in the history



Bias weight w_0 :

- Proportional to the probability that the branch is taken
- Doesn't take into account other branches

What's θ ?

- Keeps from overtraining; adapt quickly to changing behavior

Mathematical intuition

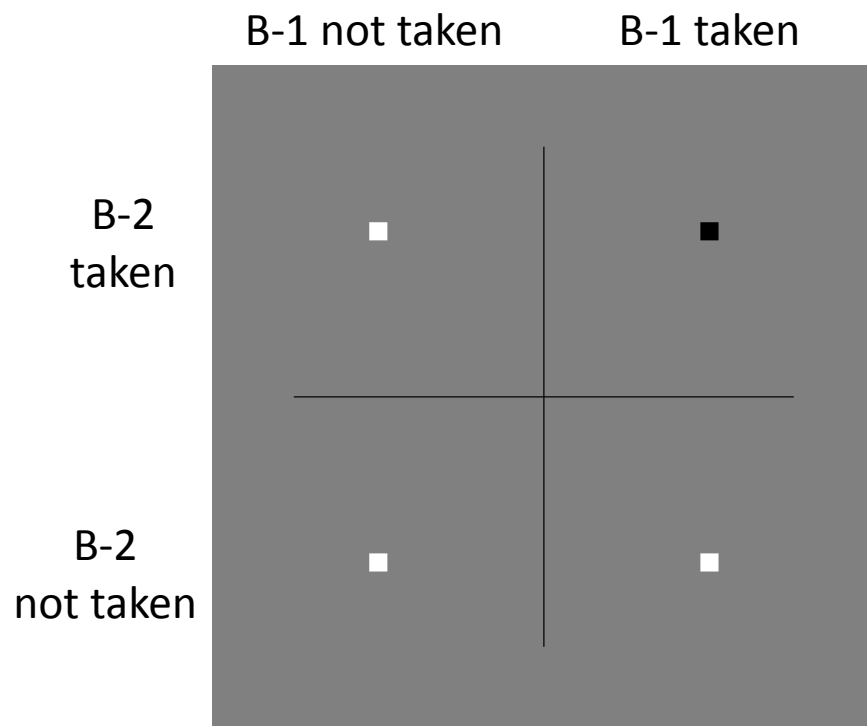
- Perceptron defines a hyperplane in $(n+1)$ -dimensional space:

$$y = w_0 + w_1x_1 + \dots + w_{n-1}x_{n-1} + w_nx_n$$

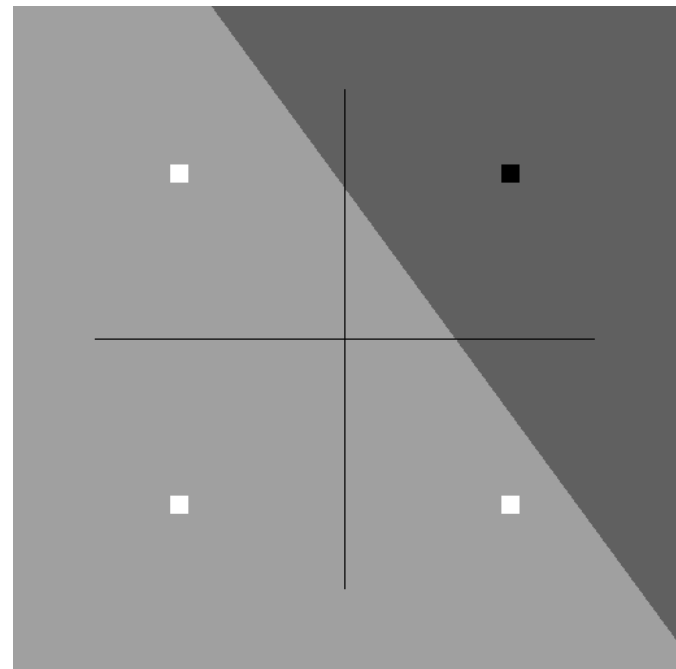
- In 2D space we have equation of a line: $y = w_0 + w_1x_1$
- In 3D, we have equation of a plane: $y = w_0 + w_1x_1 + w_2x_2$
- This hyperplane forms a *decision surface* separating predicted taken from predicted not taken instances
- This surface intersects the feature space. Is it a linear surface, e.g. a line in 2D, a plane in 3D, a cube in 4D...

Example: AND

Representation of the AND function:

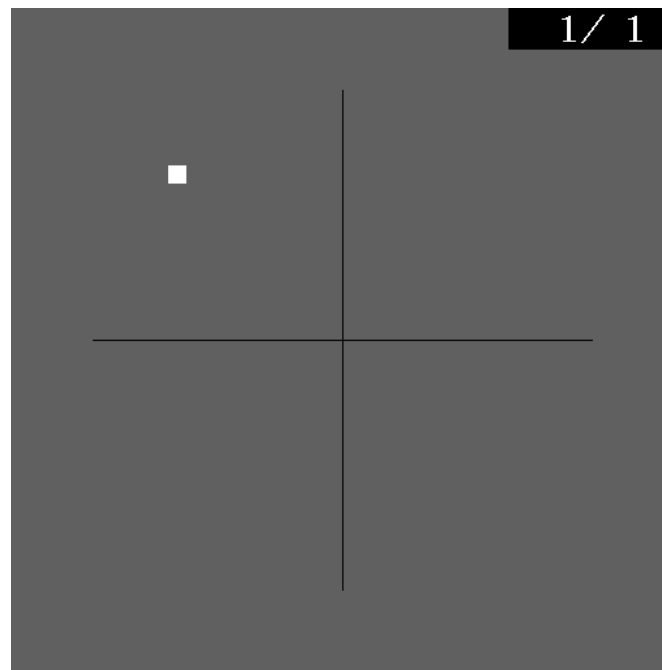


A linear decision surface
(*i.e. a plane in 3D space*)
intersecting the feature space
(*i.e. the 2D plane where $z=0$*)
separates *Not taken* from
Taken instances:



Example: AND

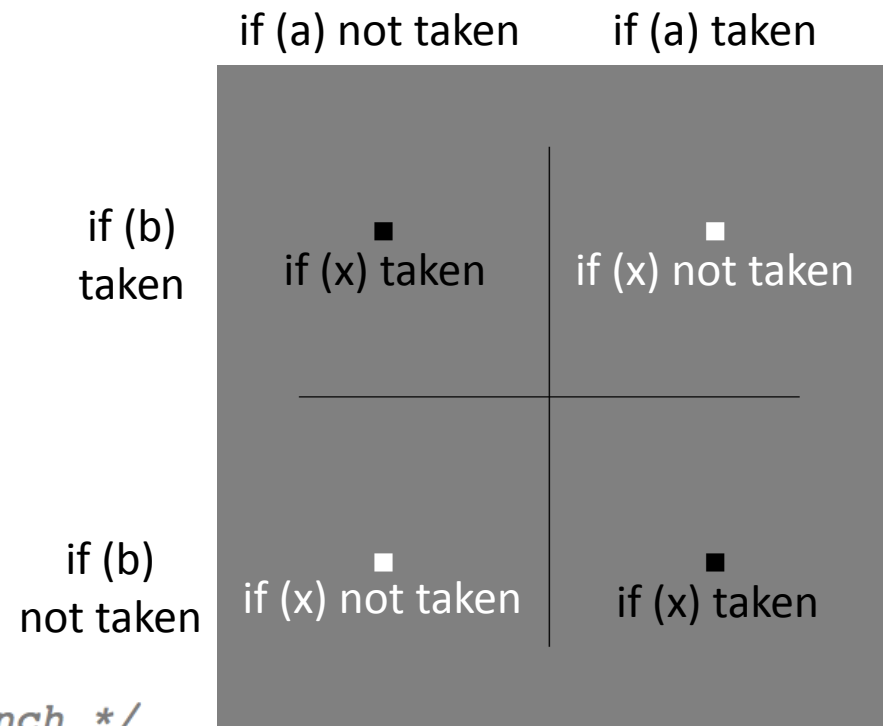
- Watch a perceptron learn the AND function:



Example: XOR

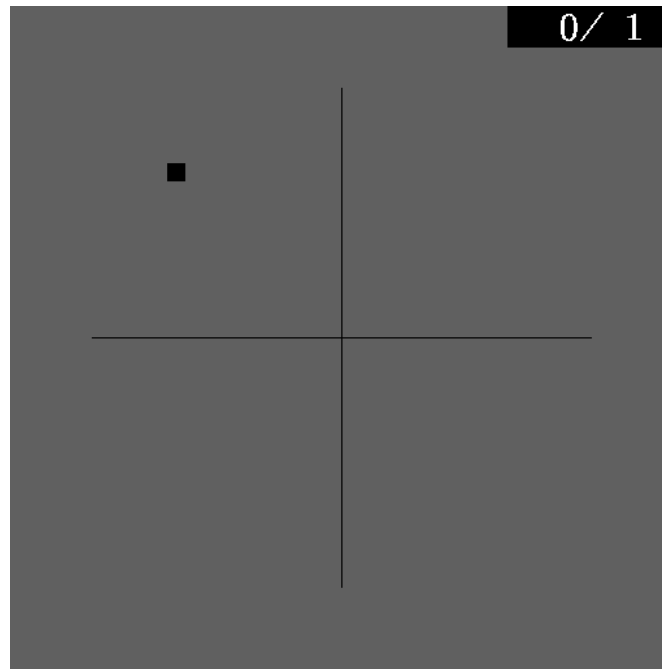
```
int f() {  
    int a, b, x, i, s = 0;  
  
    for (i = 0; i < 100000; i++) {  
        a = rand() % 2;  
        b = rand() % 2;  
        if (a) {  
            if (b)  
                x = 0;  
            else  
                x = 1;  
        } else {  
            if (b)  
                x = 1;  
            else  
                x = 0;  
        }  
        if (x) s++; /* this is the branch */  
    }  
    return s;  
}
```

Decision surface:



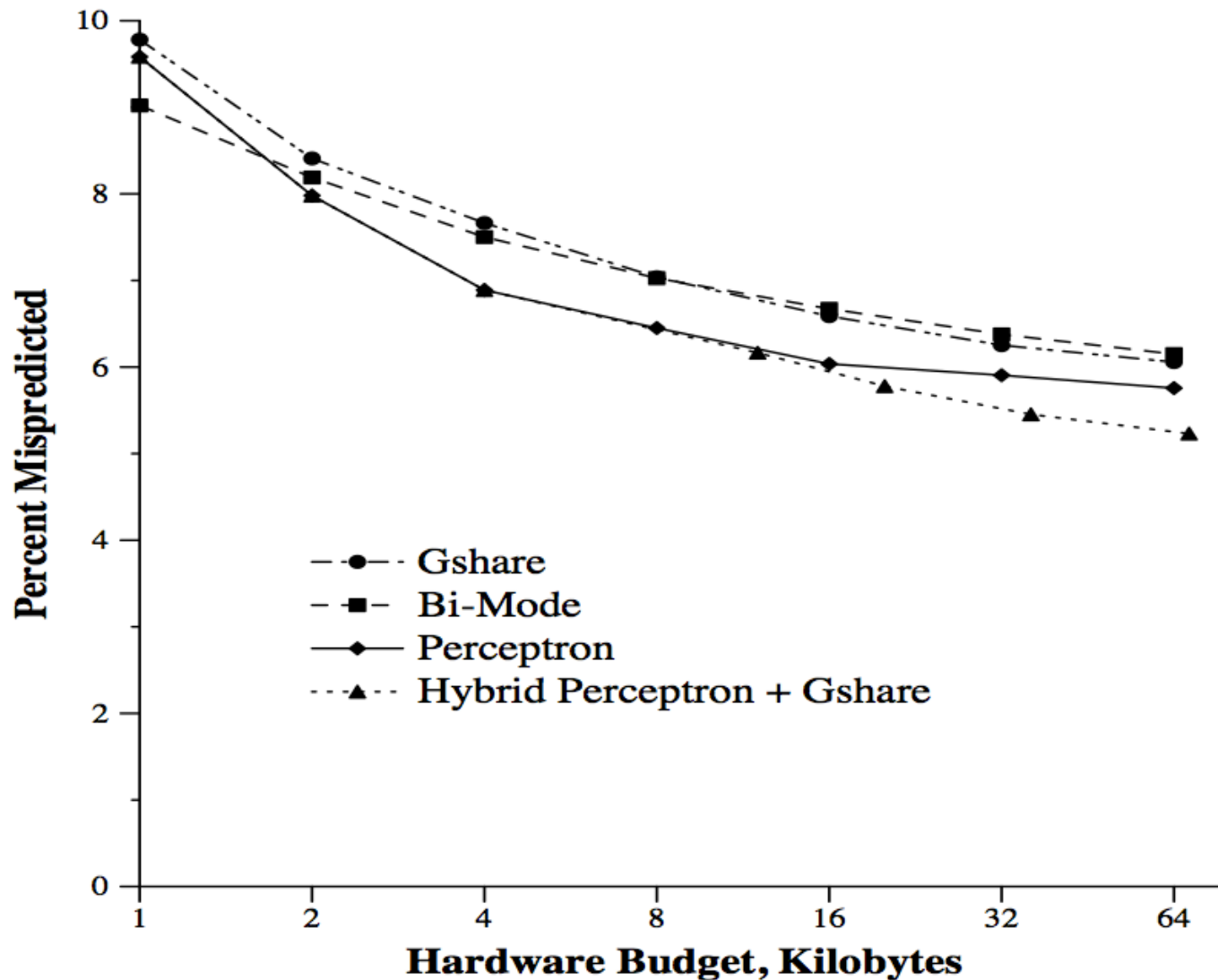
Example: XOR

- Watch a perceptron try to learn XOR



- Perceptron cannot learn such *linearly inseparable* functions

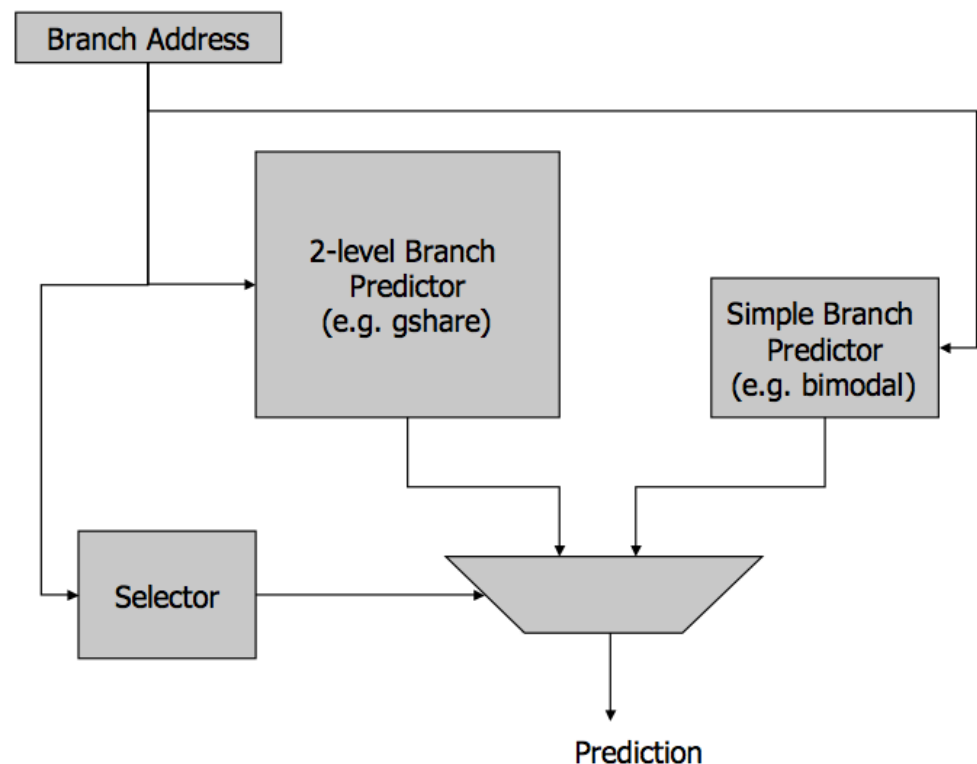
Prediction rate



Hardware Budget vs. Prediction Rate on SPEC 2000. The perceptron predictor is more accurate than the two PHT methods at all hardware budgets over 1 KB.

Hybrid branch predictor

- Single branch predictor may not perform well within and across different executions
- Previous research shows the usefulness of adapting branch predictors at run time
 - Combining advantages of different branch predictors
 - Increasing accuracy
 - Use choice predictor to decide which branch predictors to favor

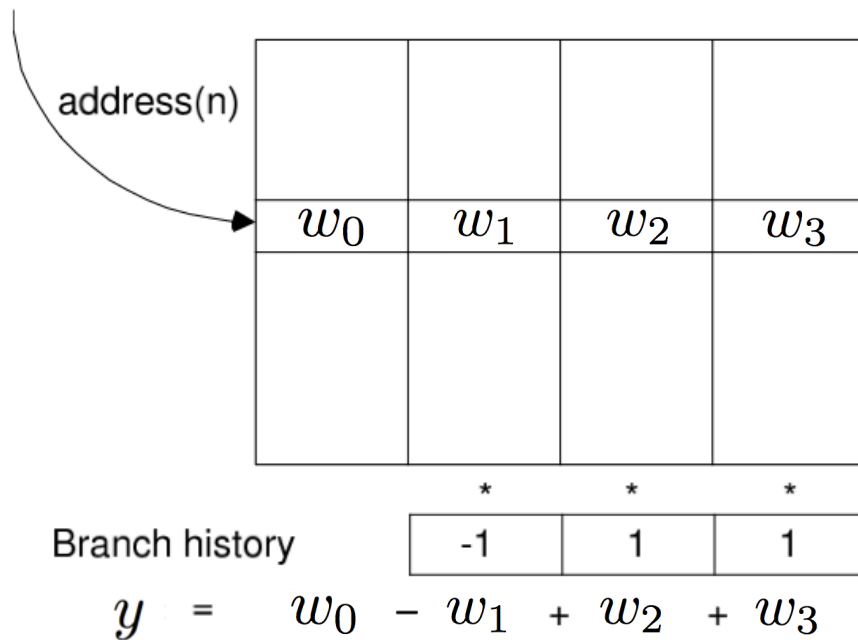


Path-based perceptron

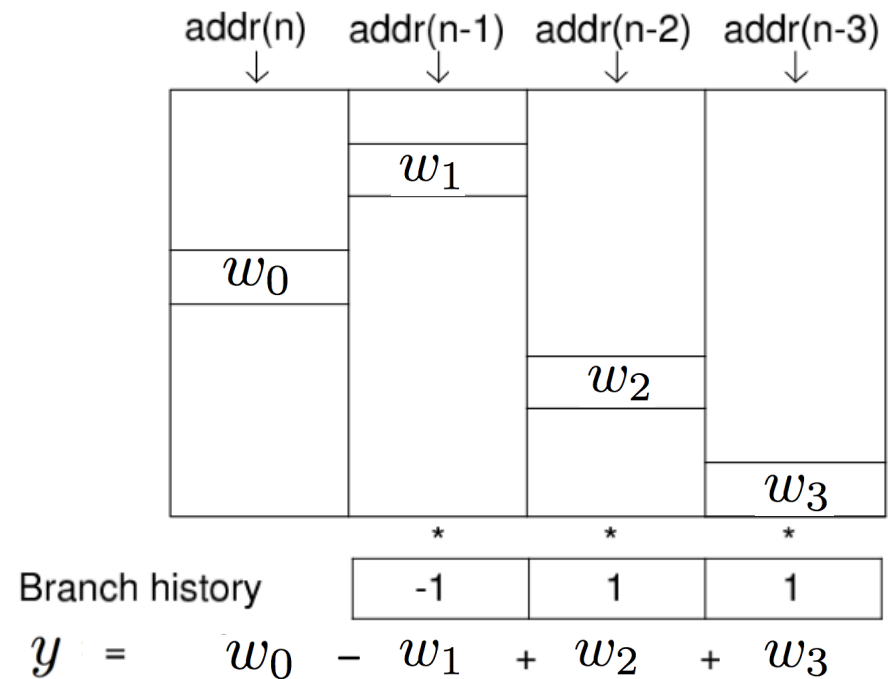
- Perceptron predictor uses only pattern history information
 - The same weights vector is used for every prediction of a branch
 - The i^{th} correlating weight is aliased among many branches
- Path-based predictor uses path information
 - The i^{th} correlating weight is selected using the i^{th} branch address
 - This allows the predictor to be pipelined, mitigating latency
 - This strategy improves accuracy because of path information
 - Even more aliasing since the i^{th} weight could be used to predict many different branches

Path-based perceptron

Perceptron fetches all weights based on the current branch address



Path-based perceptron fetches weights along the path leading up to the branch and computes a running partial sum in the pipeline

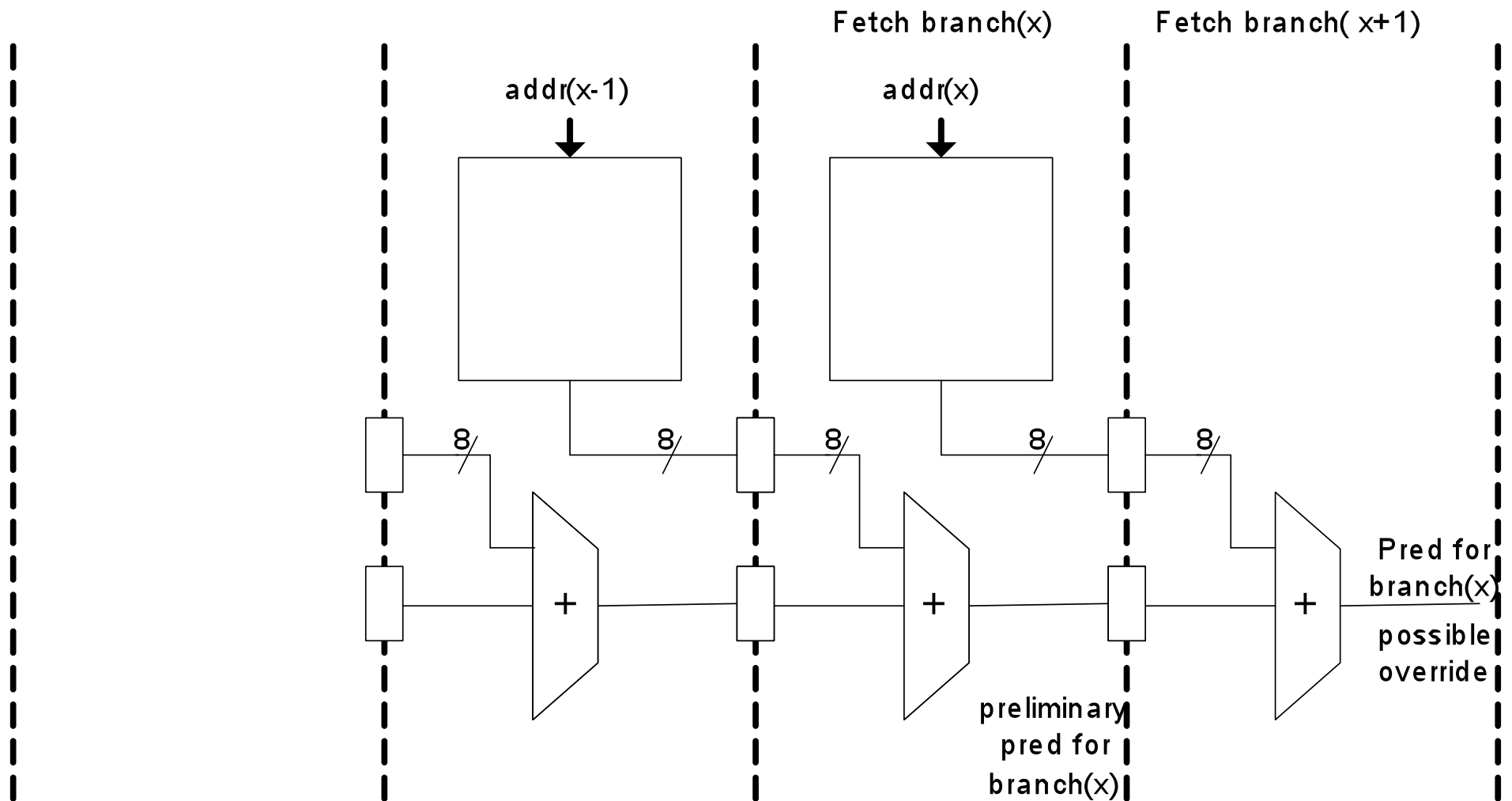


Ahead pipelining

- Because of the delay in accessing SRAM arrays and going through whatever logic is necessary, perceptron cannot produce a prediction in the same cycle
 - decouple the table access for reading the weights from adder
- Ahead pipelining
 - start prediction early to hide latency of prediction
 - by adding the summands for the dot product before the branch to be predicted is fetched, some accuracy is lost because the weights chosen may not be optimal, given that they were not chosen using the PC of the branch to be predicted
 - increases destructive aliasing, but latency benefits worth the loss in accuracy

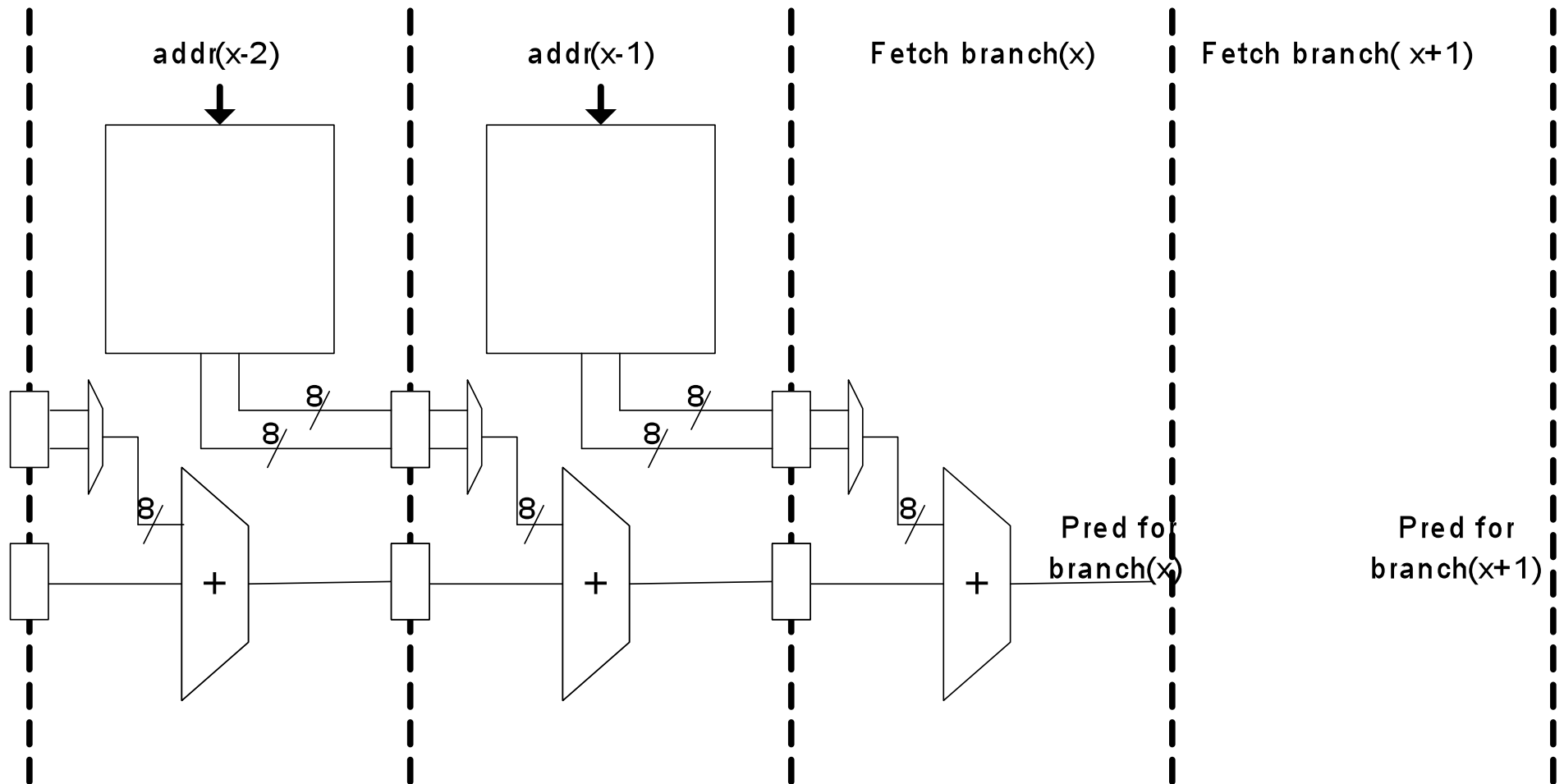
Pipelined perceptron

Uses current address in each cycle to retrieve the weights for perceptron:



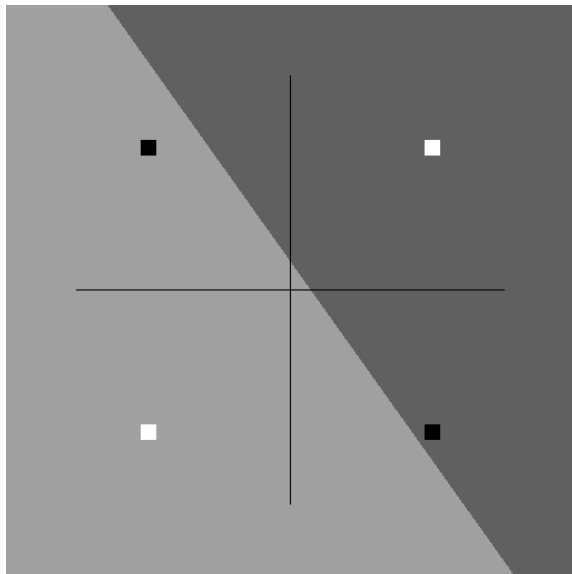
Ahead pipelined perceptron

Uses addresses from the previous cycle to retrieve two weights and then chooses between the two at the beginning of the next cycle based on the prediction whether the previous branch was predicted taken or not taken

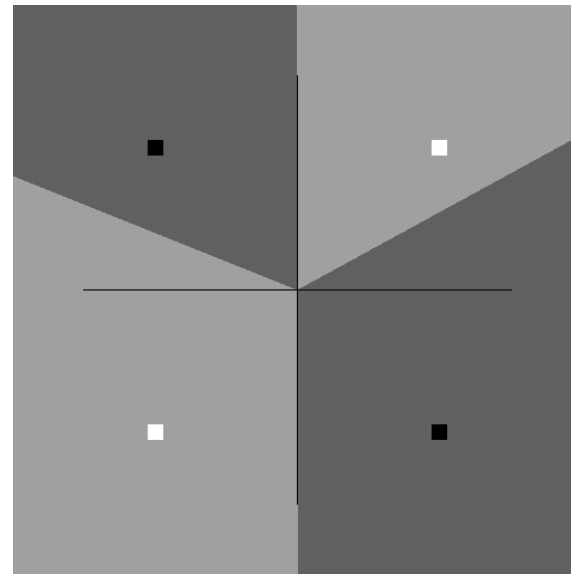


Piecewise linear branch prediction

- Generalization of perceptron and path-based predictors
- Weights are selected based on the current branch and the i^{th} most recent branch
- Forms a piecewise linear decision surface
 - Each piece determined by the path to the predicted branch
- Can solve more problems than perceptron

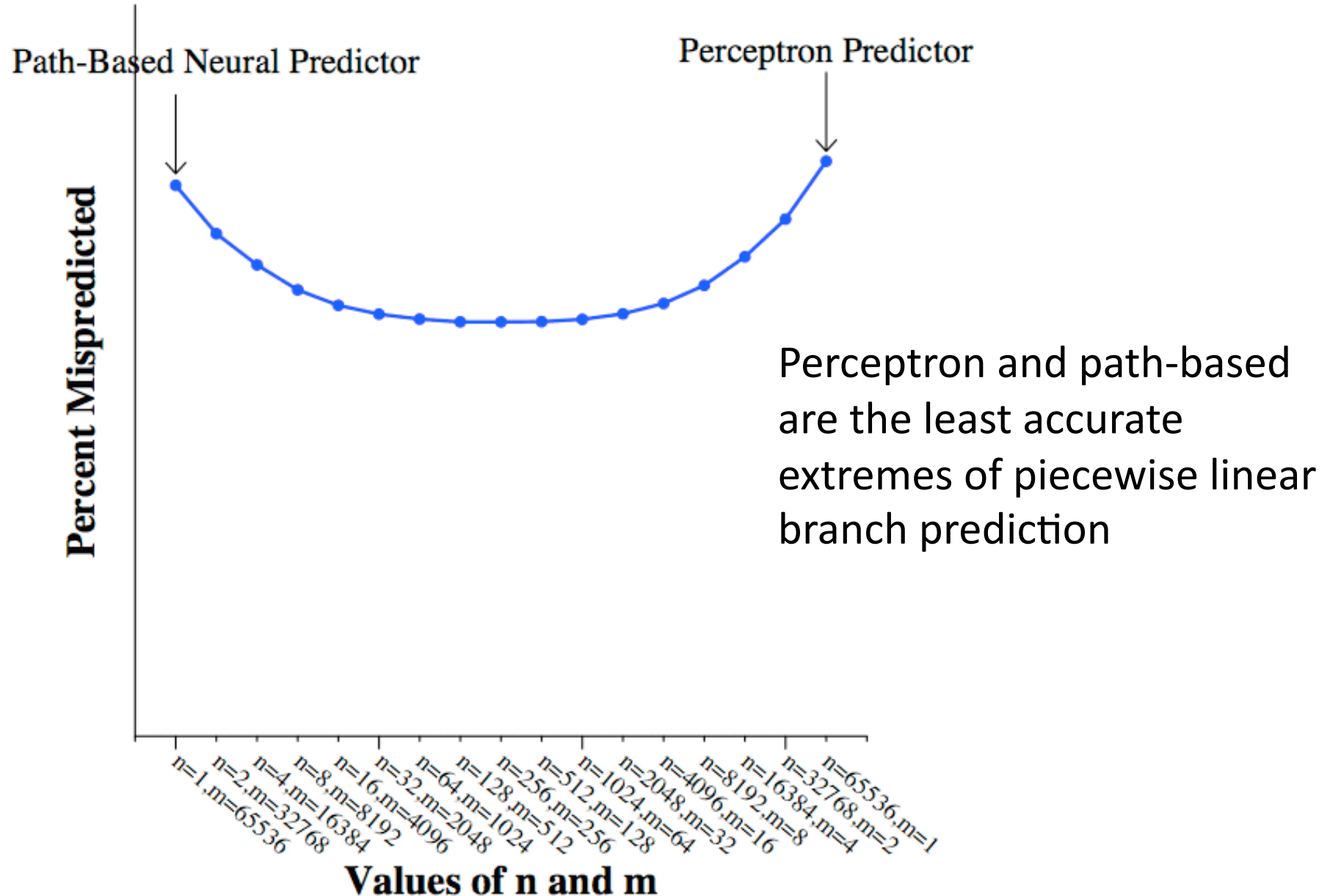


Perceptron decision surface for XOR
doesn't classify all inputs correctly

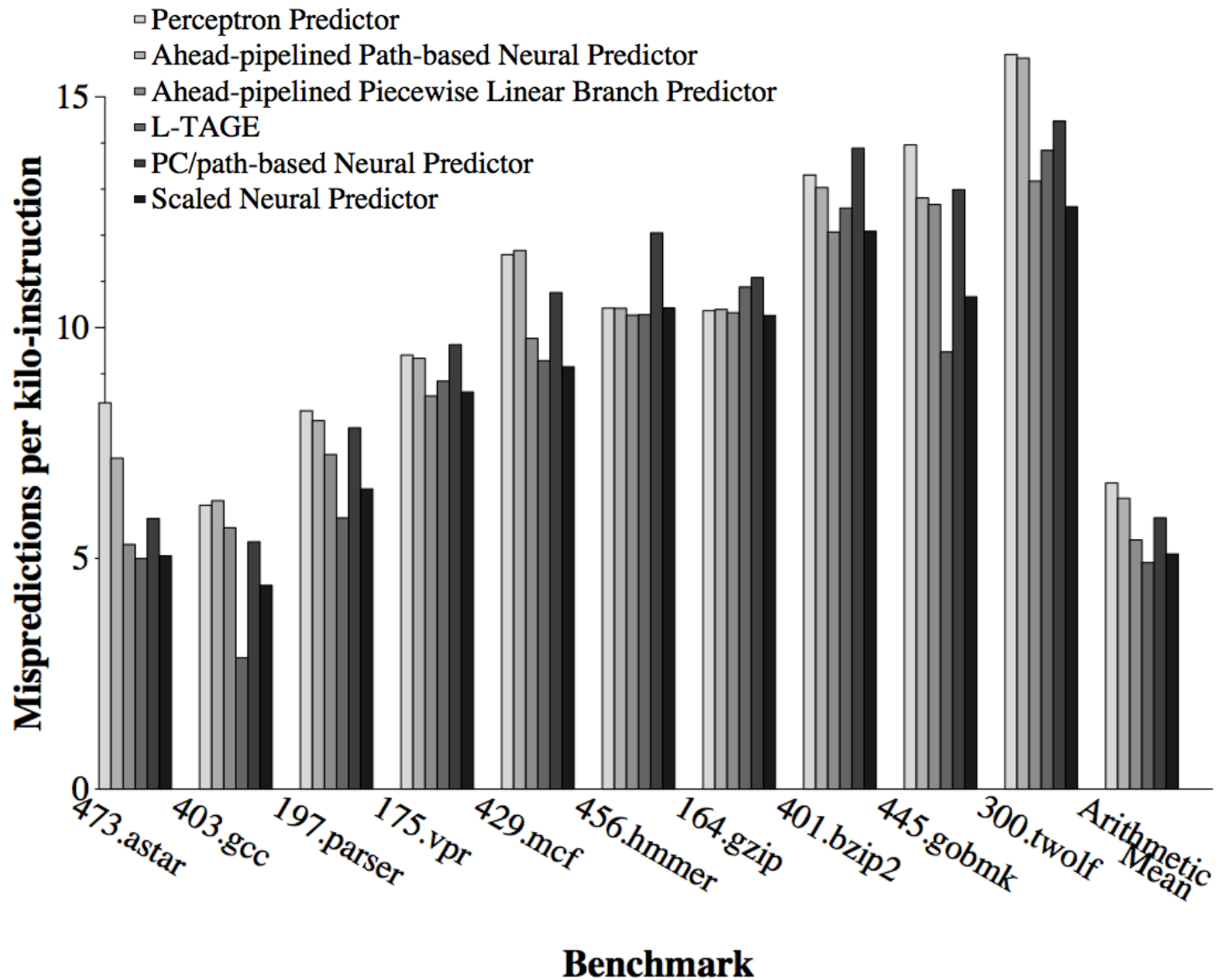


Piecewise linear decision surface for XOR
classifies all inputs correctly

Generalization continued



Comparing neural predictors



Why Perceptrons Do Well

- Gshare performs well with selective history of only 3 branches (“An Analysis of Correlation and Prediction”)
- Branches predominantly affect weights that they are correlated with
- See Table 1 in “Dynamic Branch Prediction with Perceptrons”

Hardware budget in kilobytes	History Length		
	<i>gshare</i>	bi-mode	perceptron
1	6	7	12
2	8	9	22
4	8	11	28
8	11	13	34
16	14	14	36
32	15	15	59
64	15	16	59
128	16	17	62
256	17	17	62
512	18	19	62

Best history lengths

Concluding remarks

- Perceptron branch predictors achieve *higher accuracy* by capturing correlation from very long histories
- Perceptrons incur *higher latency* at the same time because of its complex computation
 - Ahead pipeline it, so it has eff. latency 1
- More accuracy is only good with low latency

Assigning confidence to conditional branch predictions

Erik Jacobsen, Eric Rotenberg, and
J. E. Smith

Motivation

- Some branches are inherently difficult to predict
- On these branches increase performance by
 - Selective Dual Path Execution
 - Instruction Fetching
 - Use as part of Hybrid Predictor
 - Branch Prediction Reverser

Confidence Intervals

- Assign accuracy probability to each prediction regarding the accuracy
- Ideally want very small subset of overall branches to contribute to miss-prediction rate.
- Removing these inaccurate branches would improve miss-prediction rate

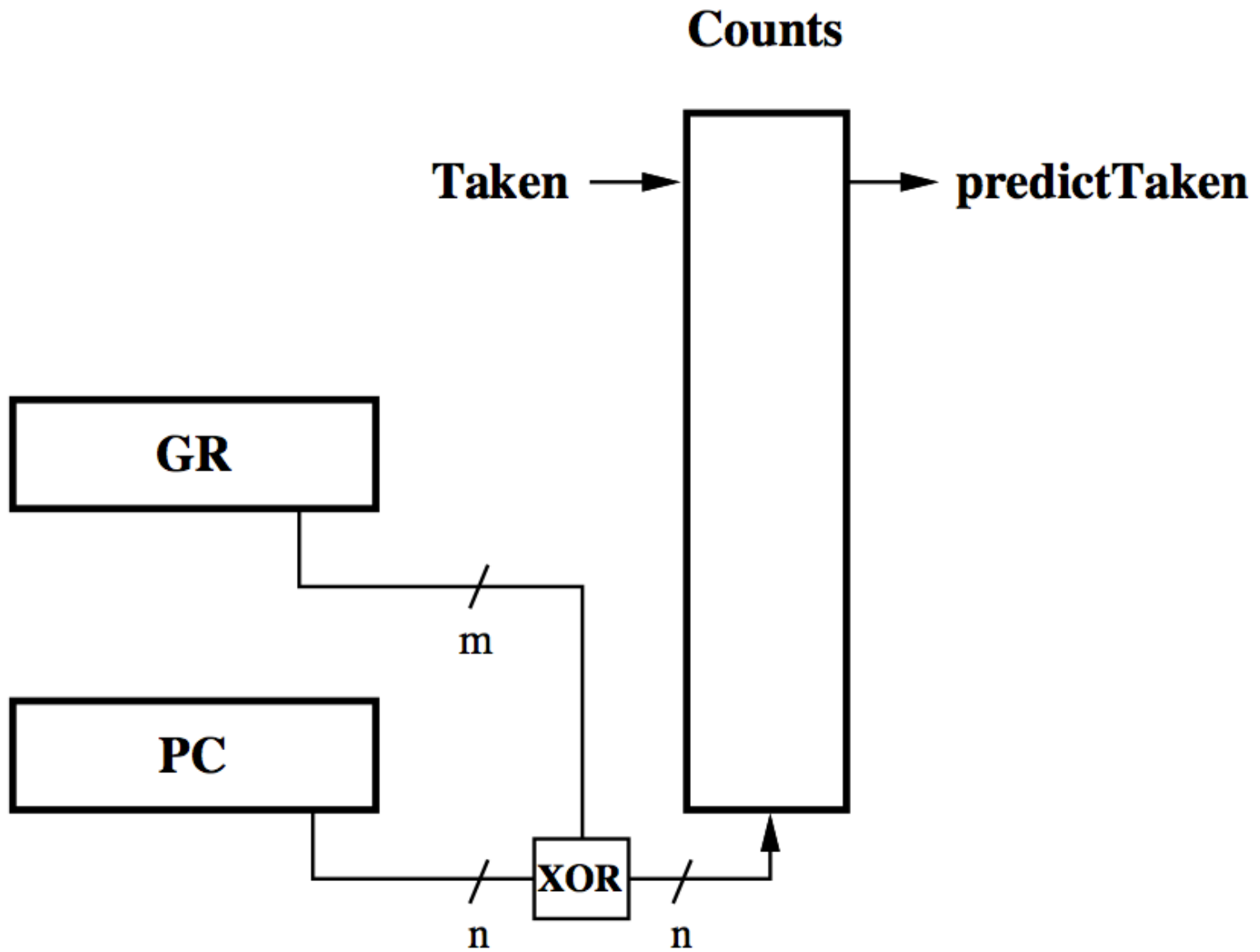
Approach

- Analyze static per-branch miss-prediction rates
- Suggests a dynamic method and applies similar analysis to “dynamic” sets
- Experimental results for dynamic methods
- Uses gshare predictor with 2^{16} entries

Gshare Review

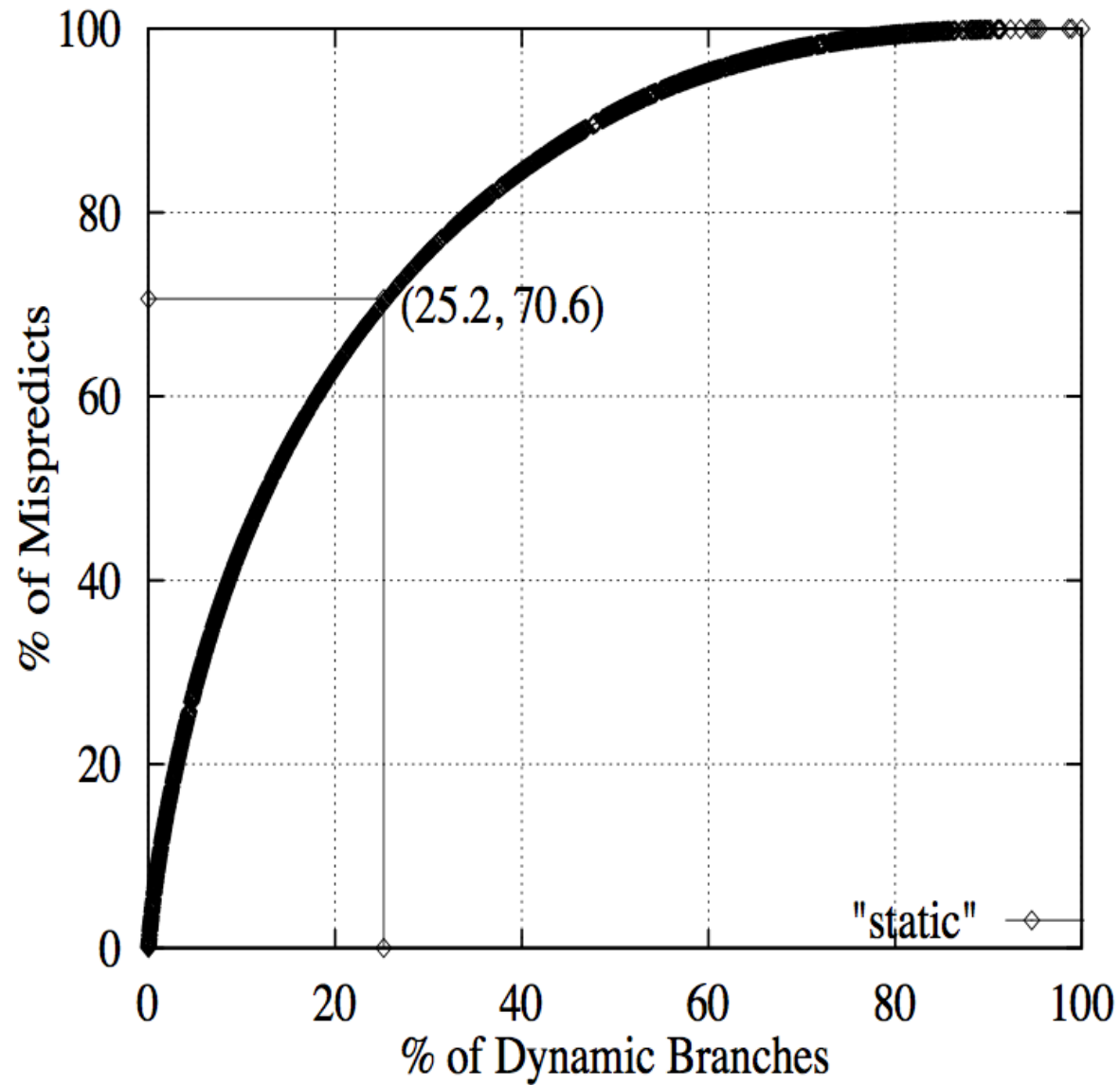
- Motivation
 - Branches correlated with branch histories as well as address bits
 - Methods such as Gselect suffer because the history bits are often redundant
 - Gshare counter table indexed by xor branch history with address bits

Gshare Setup



McFarling, "Combining Branch Predictors"

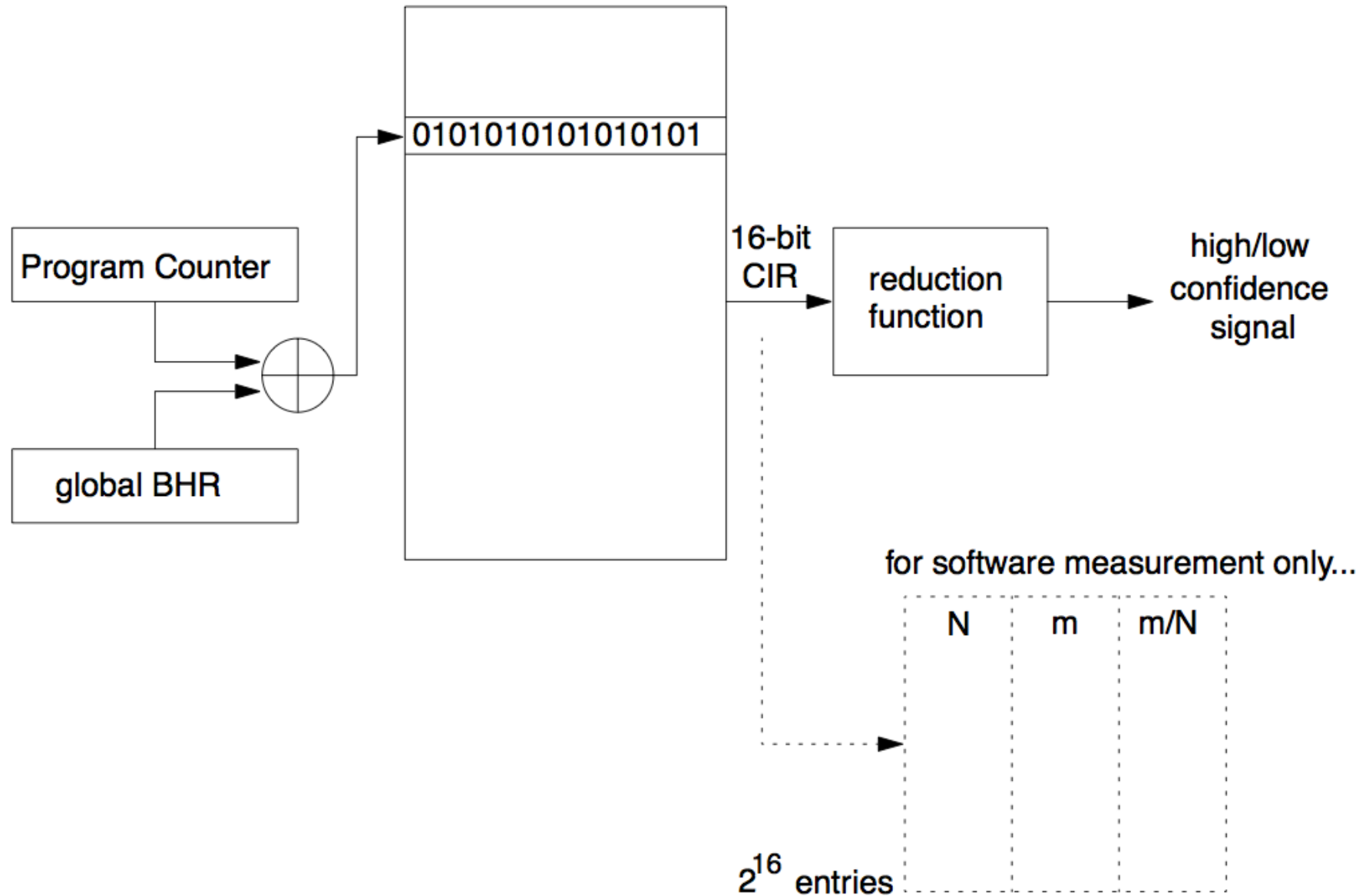
Static Branches



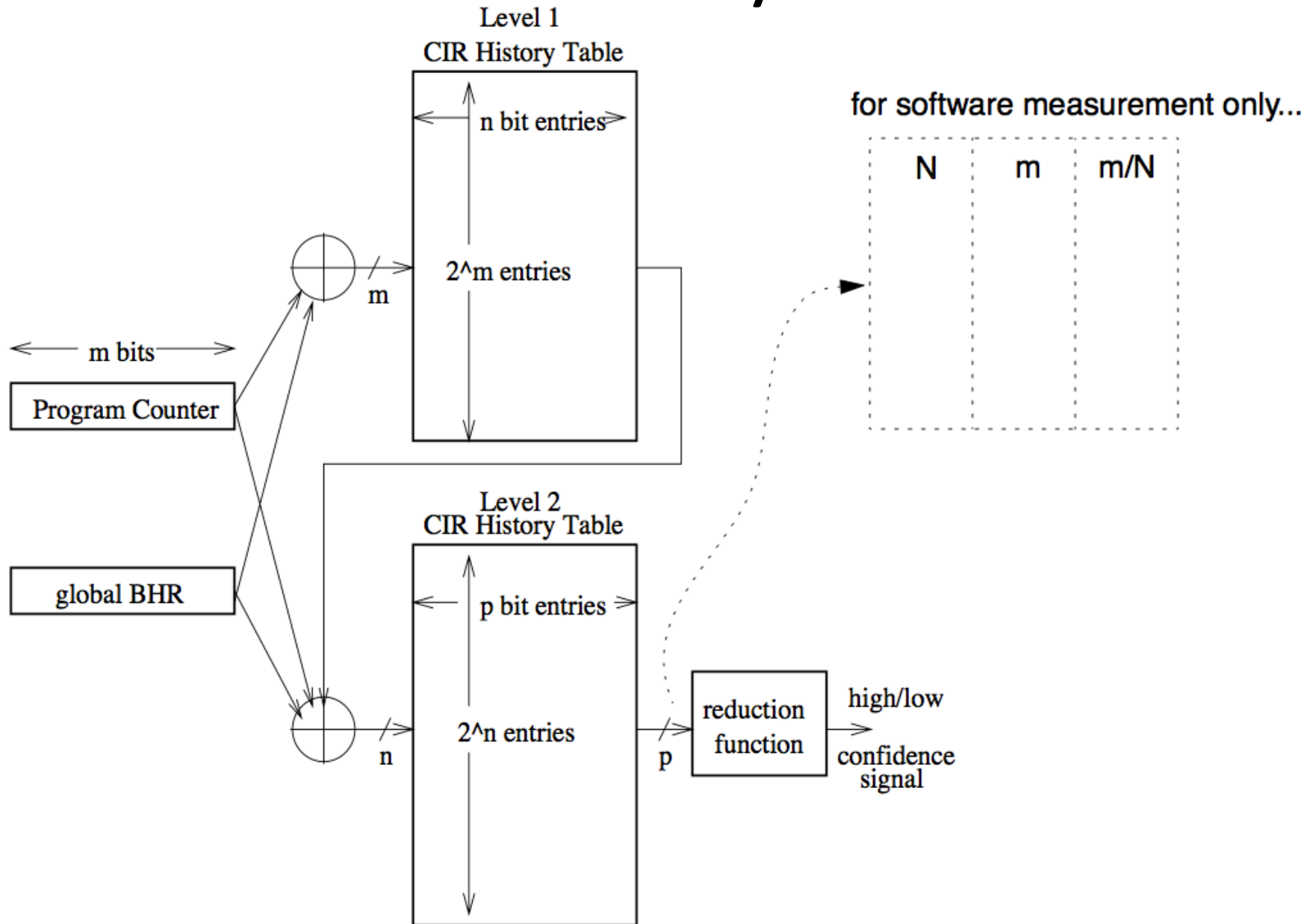
Dynamic Methods

- One-level methods
 - Single lookup into table containing history of prediction accuracies.
 - Each entry in table is n-bit shift register (CIR)
 - Lookup is some combination of PC, BHR, and CIR.
Drops CIR idea.
- Two-level methods

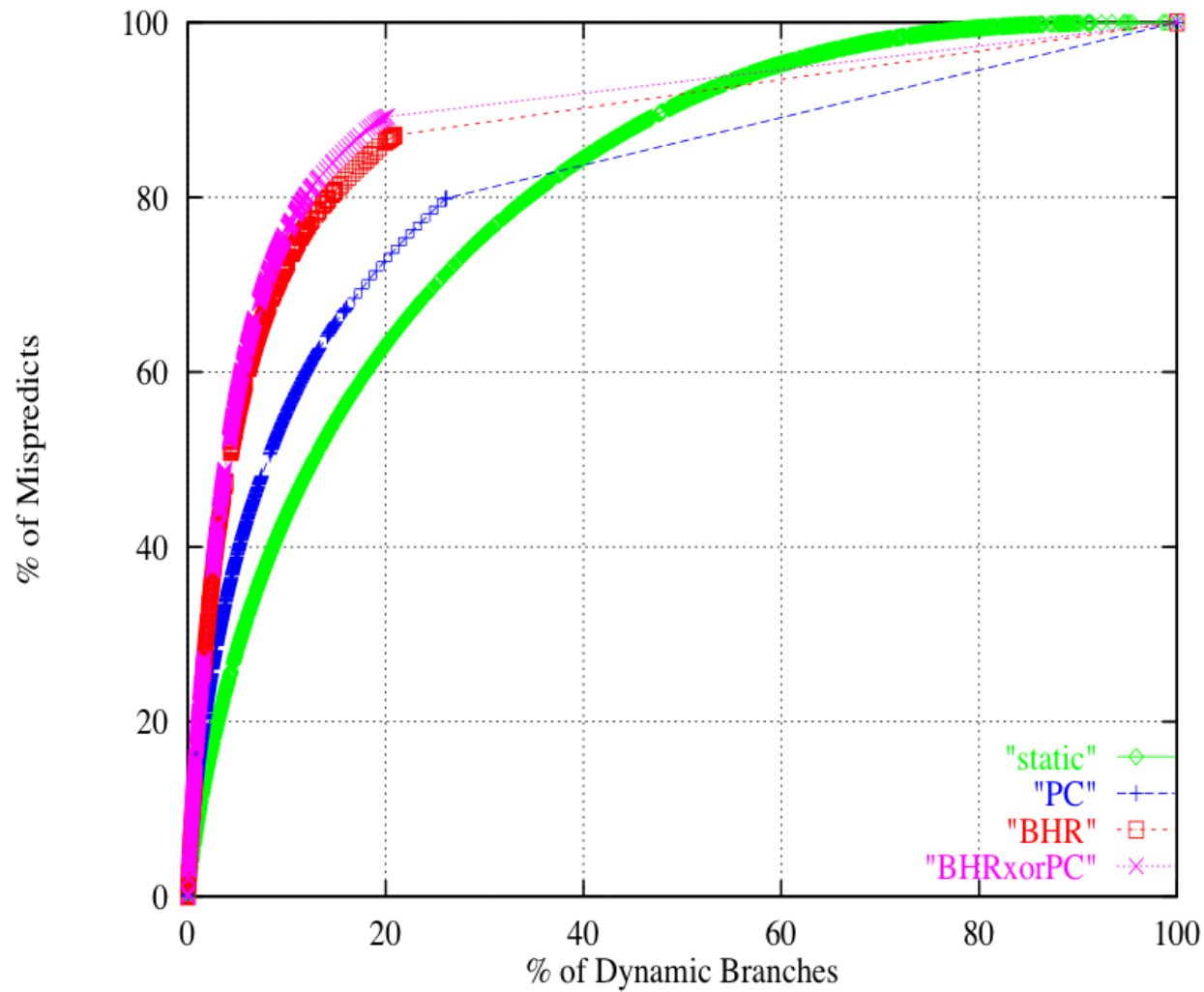
One Level Dynamic



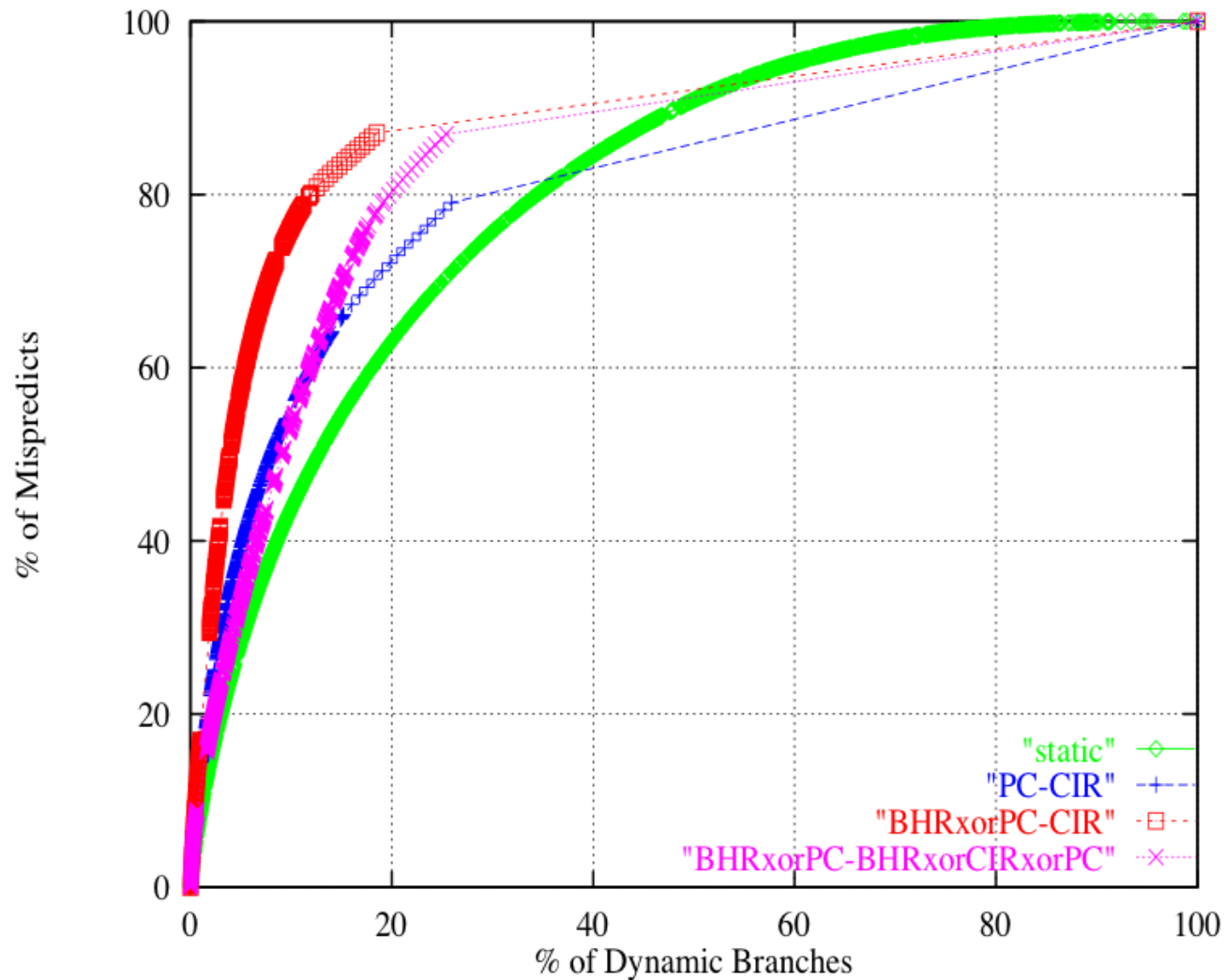
Two-Level Dynamic



1-Level Dynamic Results



2-Level Dynamic Results



Trends

- PC xor BHR to index the table gives best results
- Effect of “zero-bucket”
- Amdahl’s law on idealized results
- Two-level methods don’t help much

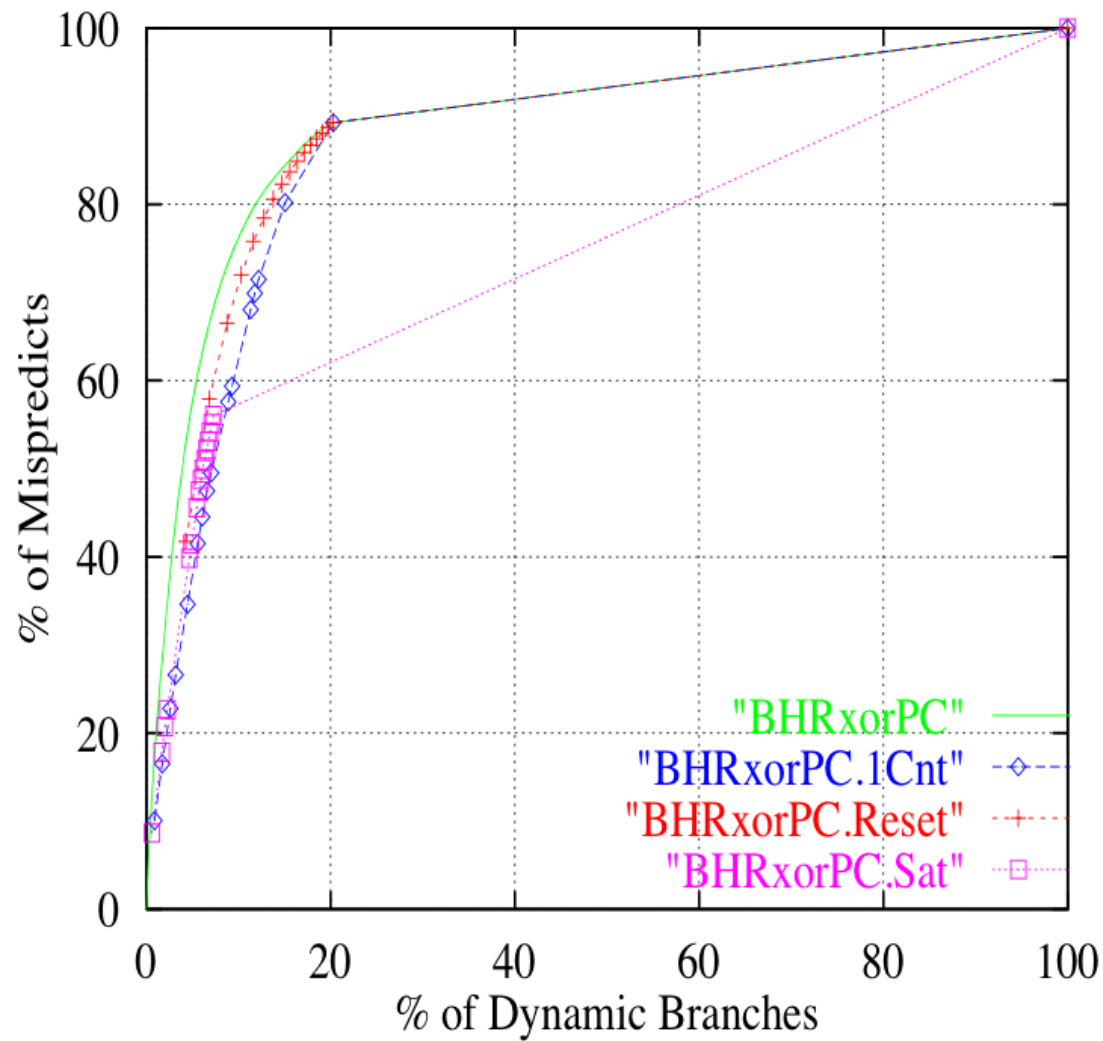
Implementation Ideas

- Ones Counting
 - History information is diluted
- Saturating Counters
 - Performs worse on average than ones counting but saves on space.

The “All-Zeros” Bucket

- Particularly important since for good prediction schemes will be frequent
- Poor placement of this subset of CIR values will result in bad performance
- This partially explains the problems with using Saturating Counters
- Resetting counters leverages importance of this subset

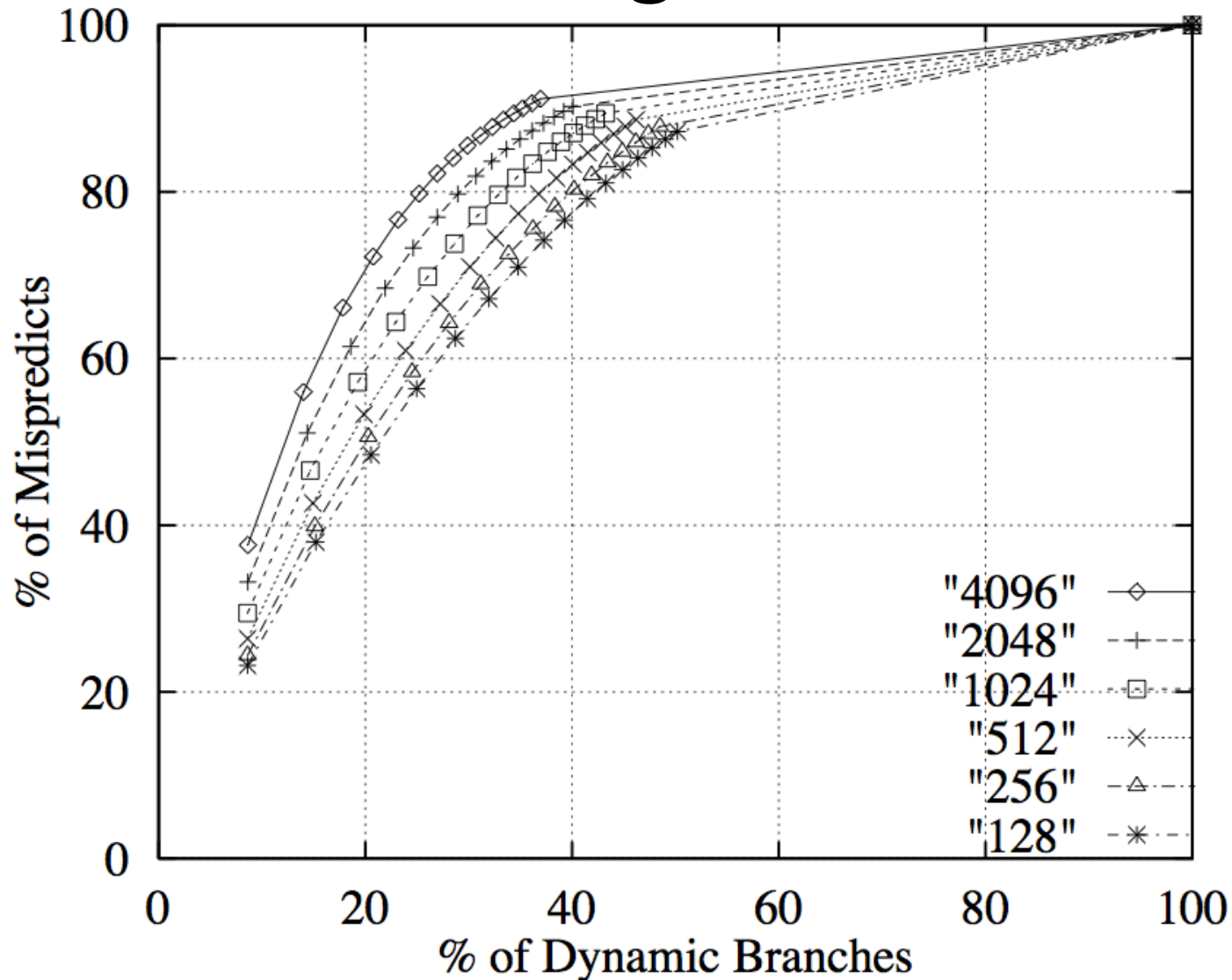
Implementations



Problems

- Amdahl's law
- Overhead. Prediction accuracy is stored separate from the predictor
 - Would using a combined branch predictor be more worthwhile
- Aliasing is still a pretty big issue since dilutes the all-zeros bucket

Constraining Resources



**Performance with small CIR tables; tables hold resetting counters,
accessed with PC xor BHR**

Conclusion

- Perceptron branch predictors achieve high accuracy by capturing correlation from very long histories
- We can vary how we act upon a branch prediction depending on the likelihood of a misprediction
- Multiple branch predictors can be combined while keeping track of which predictor is more accurate for the current branch