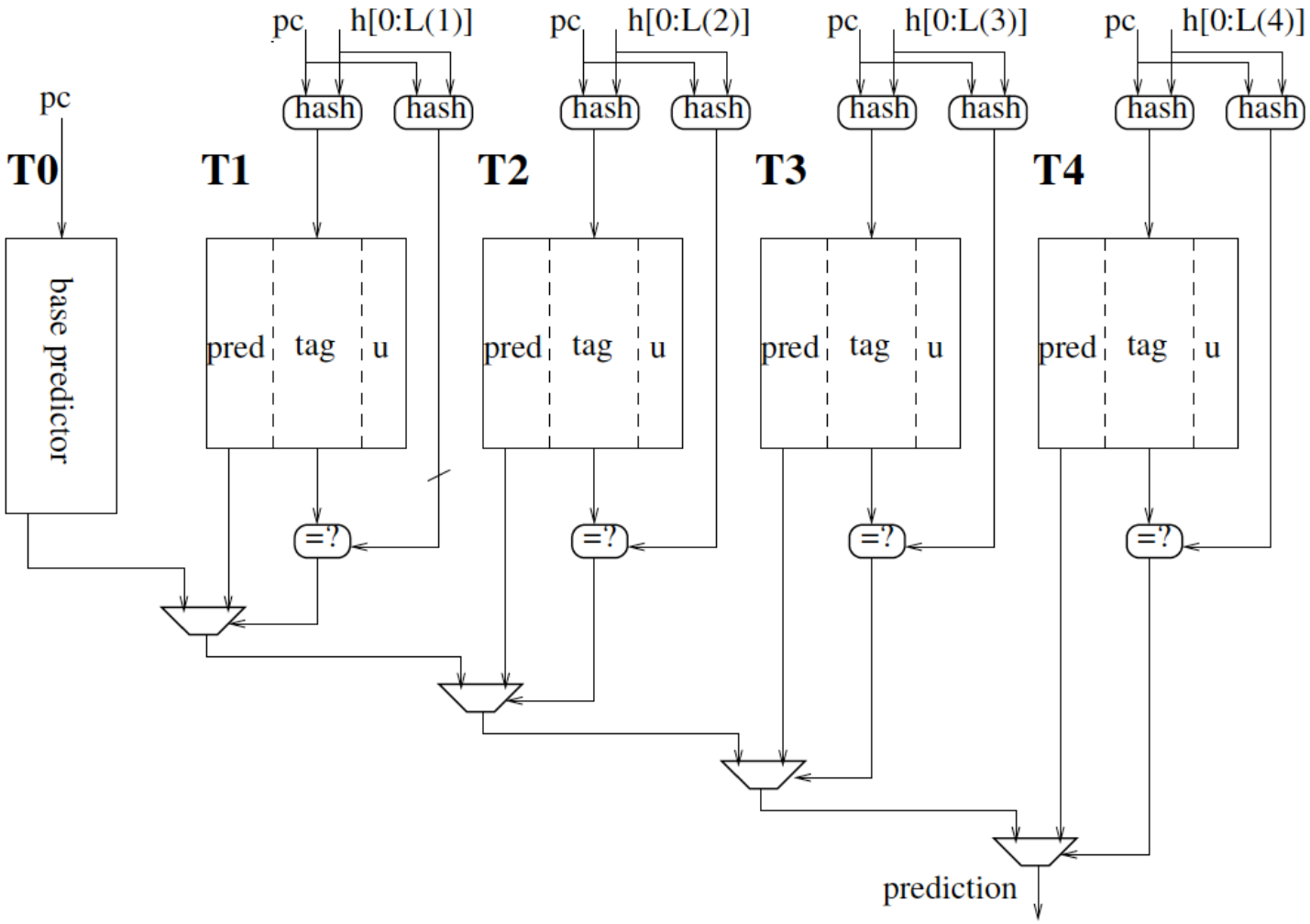
# Hybrid TAGE & Perceptron Branch Predictor

Zhenyu Wu

# TAGE Predictor



# Prediction Computation

- Base predictor $T_0$
  - PC-indexed 3-bit saturating counter
  - Giving default prediction
- Tagged predictor $T_i (1 \leq i \leq 4)$
  - $T_i$ are indexed using a geometric series of history length $\{L(i) = (int)(\alpha^{i-1} * L(1) + 0.5)\}$
  - 11-bit *tag*, 2-bit unsigned useful counter *u*, 3-bit signed counter *pred*
  - Giving prediction on a tag match
  - Provider component & *altpred*

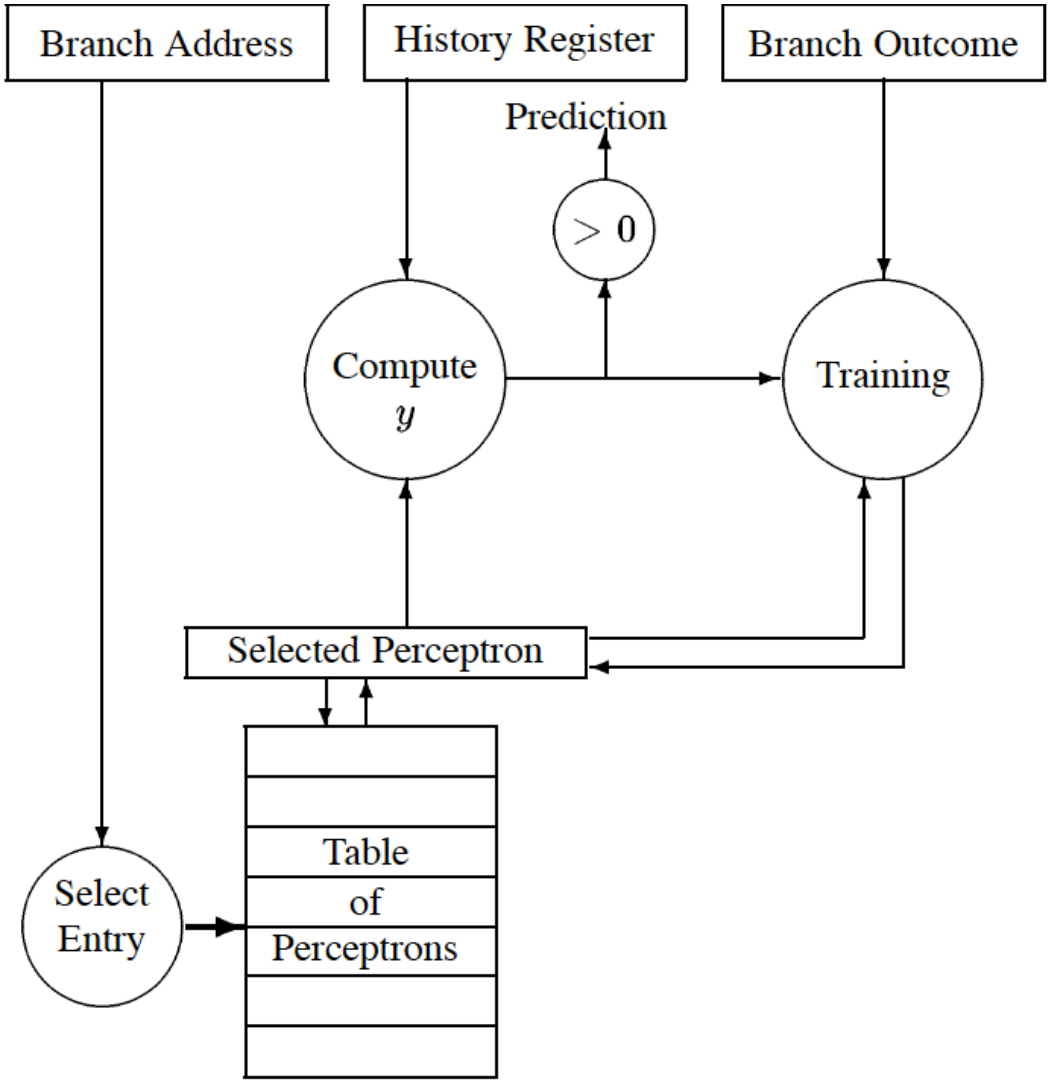
# Updating Policy

- Update the useful counter *u*
  - *u* is updated when *altpred* is different from final *pred*
  - Increment if *pred* is correct, decrement otherwise
  - Reset in period of 256K branches
- Update the *pred* counter of the provider component on a correct prediction
- The overall prediction is incorrect
  - Update the *pred* counter of the provider component $T_i$
  - If $i < M$, allocate an entry on a predictor component $T_k (i < k < M)$
  - Read $M - i - 1$ $u_j$ from $T_j (i < j < M)$

# Updating Policy (Cont.)

- Rules for new entry allocation
  - Priority for allocation
    - If exits k, such that $u_k = 0$, then $T_k$ is allocated
    - Else the *u counters from the components* $T_j$ $(i < j < M)$ are all decremented
  - Avoiding ping-phenomenon
    - If $T_j$ & $T_k$ can be allocated, then $T_j$ is chosen with higher probability.
  - Initializing the allocated entry
    - *pred* counter set to *weak correct*
    - *u* useful counter set to *strongly not useful*

# Perceptron Predictor



# Prediction Computation

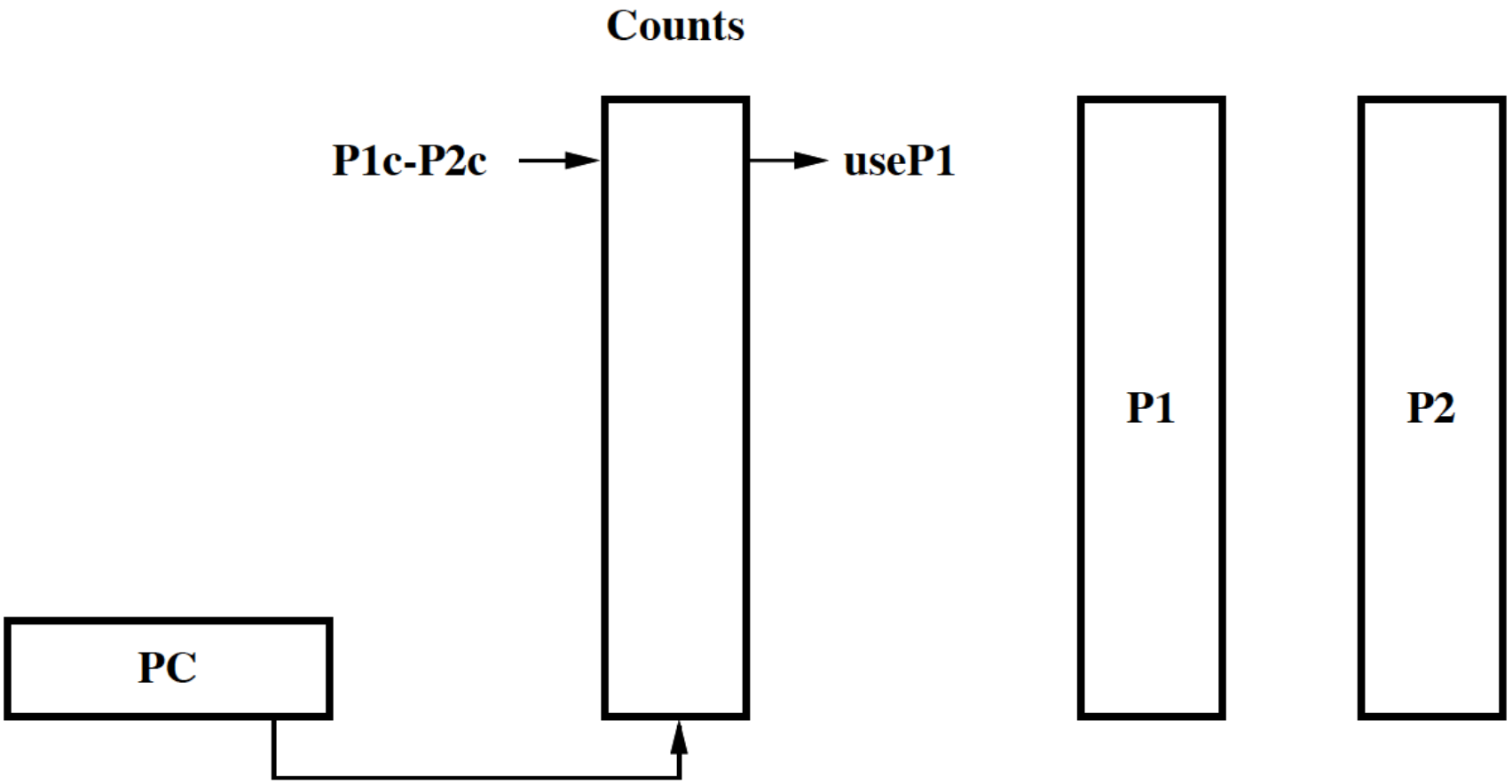
- A perceptron is represented by a vector of signed integer weights $(w_{0..n})$
  - $w_0$ serves as bias
- The input is the global history record $(x_{1..n})$
  - $x_0$ is always set to 1, providing a bias input
  - $x_i$ is either -1 (NT) or +1 (T)
- The output y of the perceptron is computed as
  - $y = w_0 + \sum_{i=1}^{n} x_i w_i$
  - Predict to take if $y \geq 0$, not to take if $y < 0$

# Updating Policy

- Using the following algorithm to train the perceptron
  - $\theta$ is the threshold parameter to decide when enough training has been done
  - $\theta = \lfloor 1.93h + 14 \rfloor$

```
if sign(y_out) ≠ t or |y_out| ≤ θ then
        for i := 0 to n do
                w_i := w_i + t x_i
        end for
end if
```

# Combining Branch Predictors



# How to combine TAGE with Perceptron to make better prediction?

- The combined predictor contains 2 predictors: TAGE & Perceptron
- Using a 2-bit saturating counter to select better predictor
- Each counter keeps track of which predictor is more accurate for the shared branches

| P1c | P2c | P1c-P2c | |
|---|---|---|---|
| 0 | 0 | 0 | (no change) |
| 0 | 1 | -1 | (decrement counter) |
| 1 | 0 | 1 | (increment counter) |
| 1 | 1 | 0 | (no change) |

# Storage Computation

- Perceptron
  - 512 perceptron
  - 8-bit unsigned integer weight
  - 64 weights (1 for bias) per perceptron
  - $512 \times 8 \times 64\ bits = 32KB$
- TAGE
  - $T_0$:$2^{13} \times 3\ bits = 3\text{KB}$
  - $T_1$:$2^{12} \times (5 + 11)\ bits = 8KB$
  - $T_2$:$2^{12} \times (5 + 10)\ bits = 7.5KB$
  - $T_3$:$2^{12} \times (5 + 9)\ bits = 7KB$
  - $T_3$:$2^{12} \times (5 + 8)\ bits = 6.5KB$
  - $3 + 8 + 7.5 + 7 + 6.5 = 32\text{KB}$
- Combining $32 + 32 = 64KB$