NC STATE UNIVERSITY

EXACT: Explicit Dynamic-Branch Prediction with Active Updates

Muawya Al-Otoom, Elliott Forbes, Eric Rotenberg

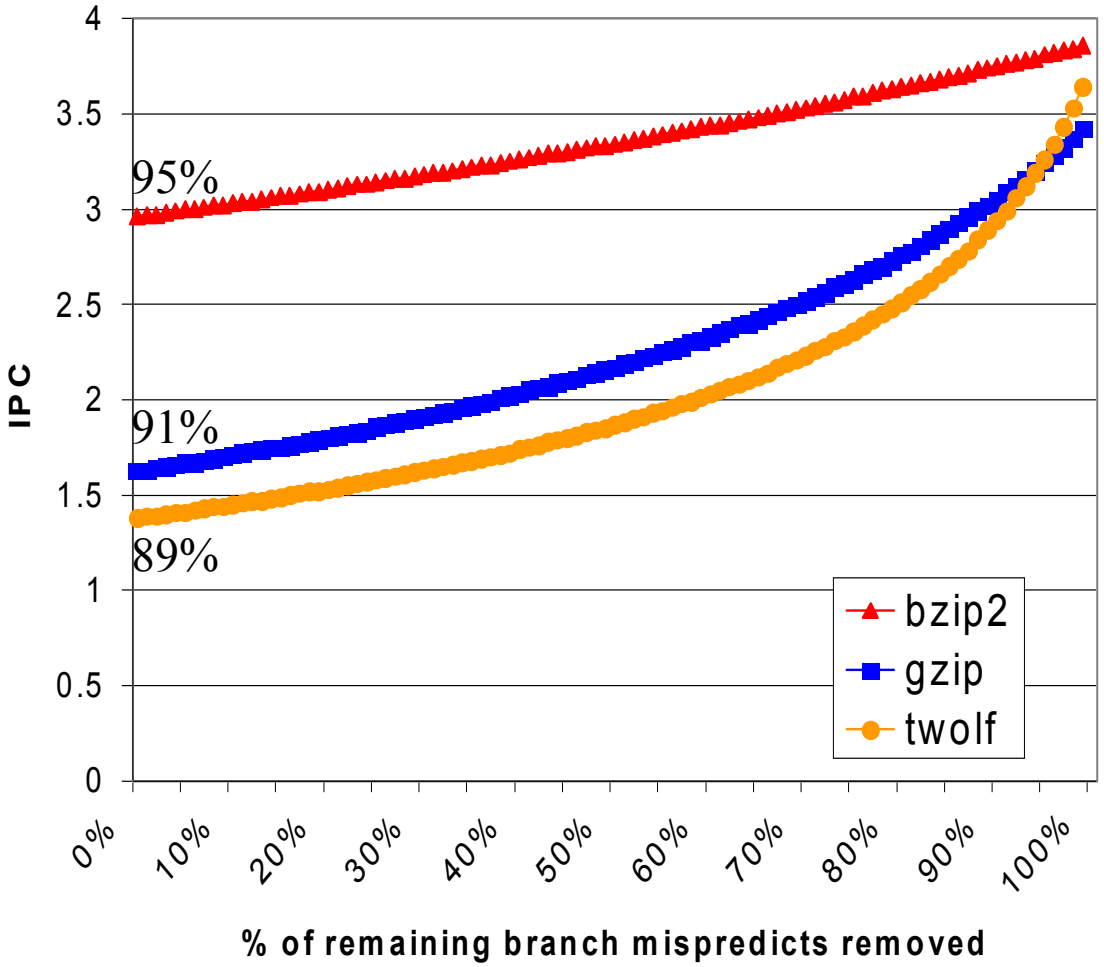
Center for Efficient, Scalable, and Reliable Computing
Department of Electrical & Computer Engineering
North Carolina State University

CESR

Branch Prediction and Performance

- 1K-entry instruction window
- 128-entry issue queue (similar to Nehalem)
- 4-wide fetch/issue/retire
- 16 KB gshare

- Randomly remove 0% to 100% of mispredictions



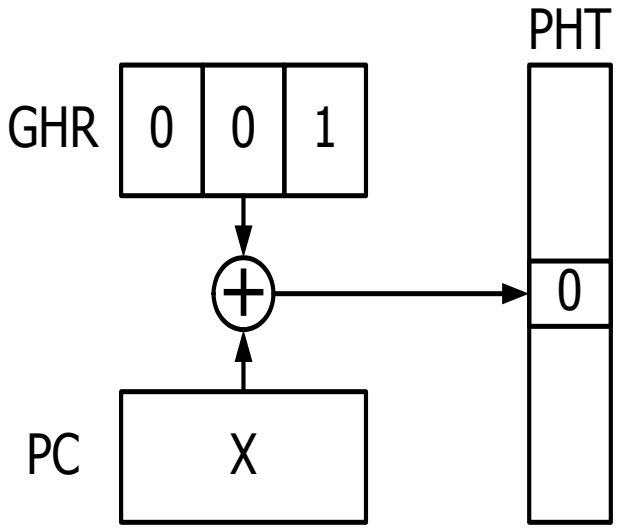
Example

```
while ( . . . )
{
     . . .
     for (i = 0; i < 16; i++)
     {
          if (a[i] > 10)
          {
               . . .
          }
          else
          {
               . . .
          }
     }
     . . .
}
```

```
    LOAD    r5 = a[i]
X:  BRANCH  r5 <= 10
```

Example (cont.)

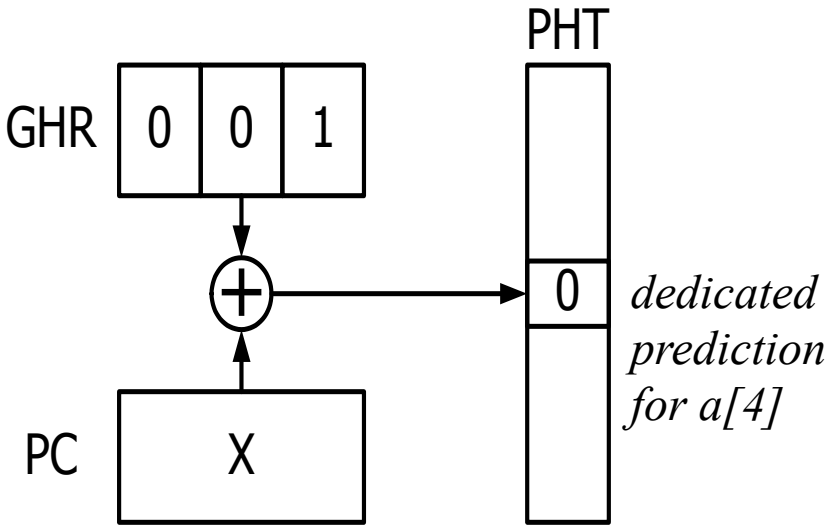
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]
LOAD	4	11	15	2	33	7	1	3	52	9	3	8	5	55	8	18
BRANCH	1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0



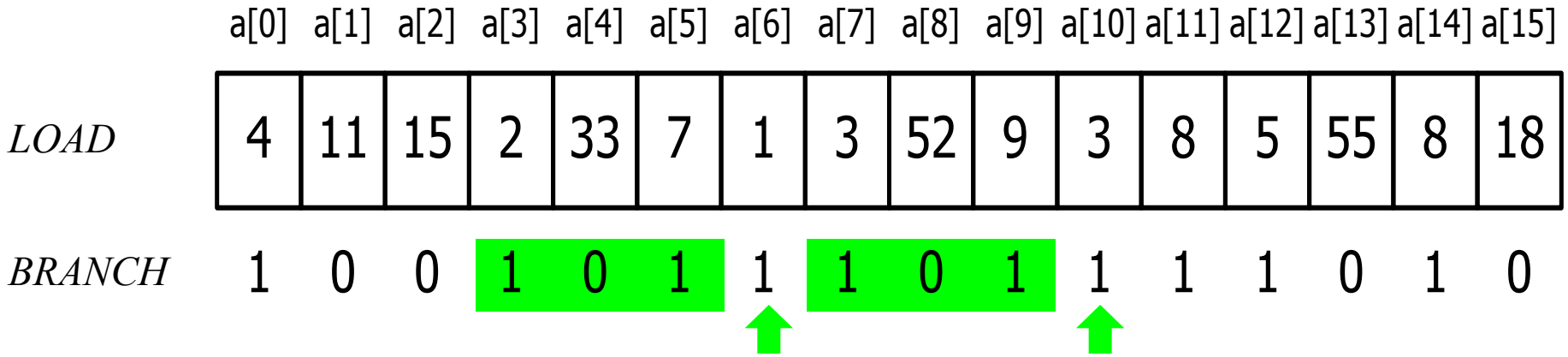
Good Scenario #1

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]
LOAD	4	11	15	2	33	7	1	3	52	9	3	8	5	55	8	18
BRANCH	1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0

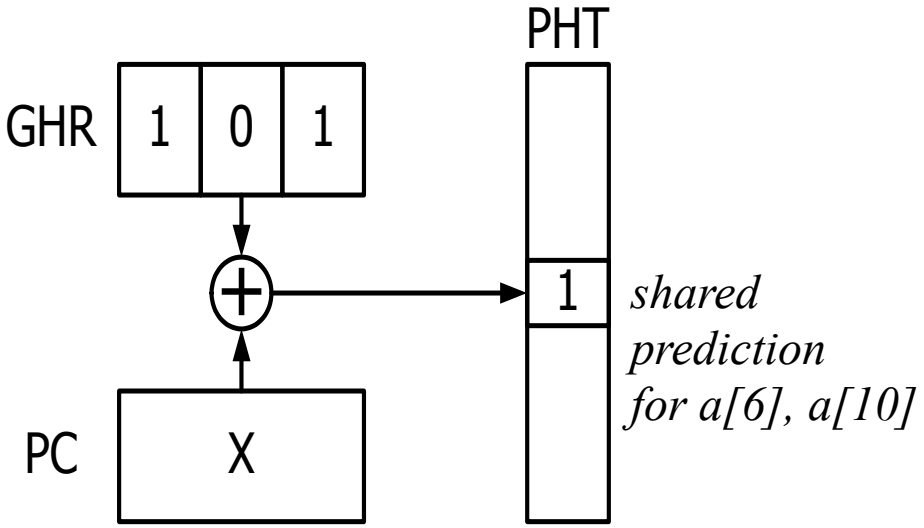
- a[4] has unique GHR context.
- GHR implicitly identifies a[4].
- Dedicated prediction for a[4].



Good Scenario #2



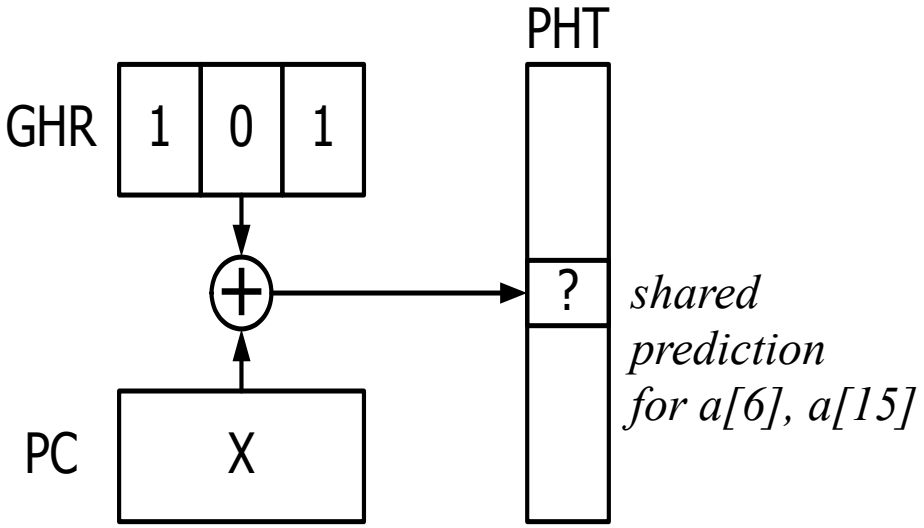
- a[6], a[10] have same GHR context.
- GHR does not distinguish a[6], a[10].
- Shared prediction for a[6], a[10].
- Fortunately, they have same outcome.



Bad Scenario #1

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]
LOAD	4	11	15	2	33	7	1	3	52	9	3	8	5	55	8	18
BRANCH	1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0

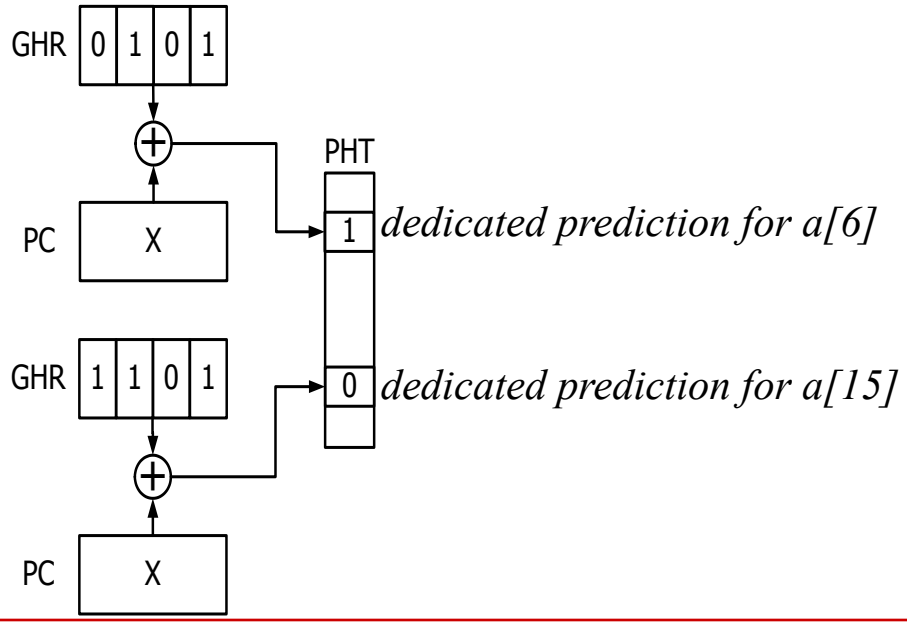
- a[6], a[15] have same GHR context.
- GHR does not distinguish a[6], a[15].
- Shared prediction for a[6], a[15].
- Problem: they have different outcomes.



Bad Scenario #1 (cont.)

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]
LOAD	4	11	15	2	33	7	1	3	52	9	3	8	5	55	8	18
BRANCH	1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0

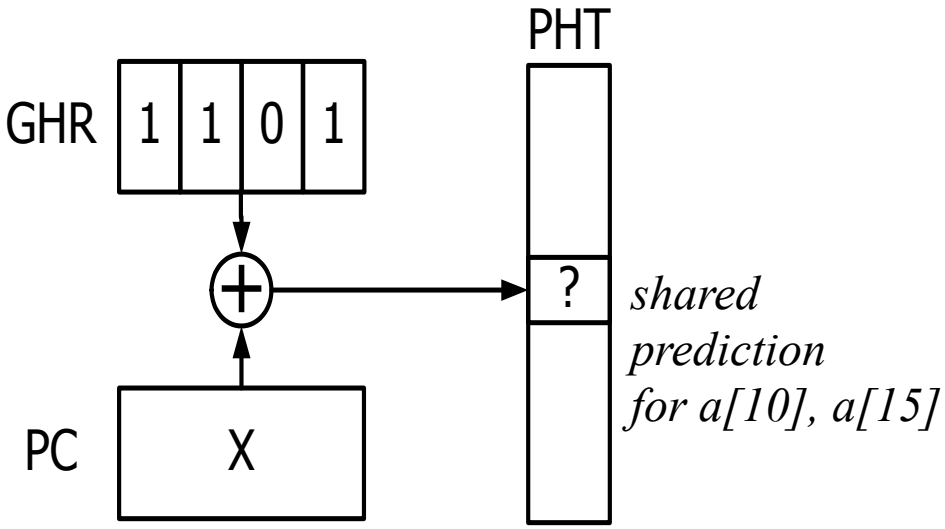
- Indirect solution
  - Use longer history.
  - GHR now distinguishes a[6], a[15].



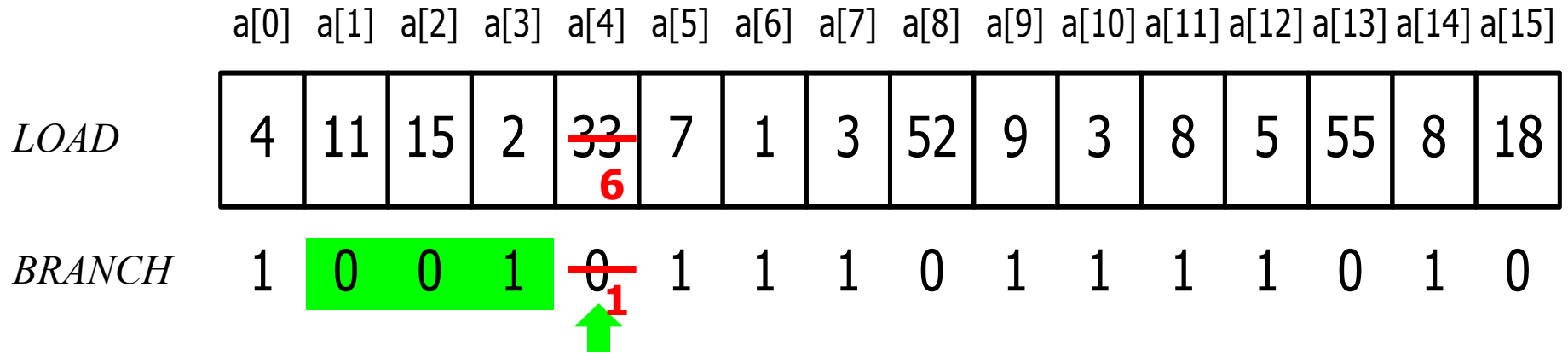
Bad Scenario #1 (cont.)

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]
LOAD	4	11	15	2	33	7	1	3	52	9	3	8	5	55	8	18
BRANCH	1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0

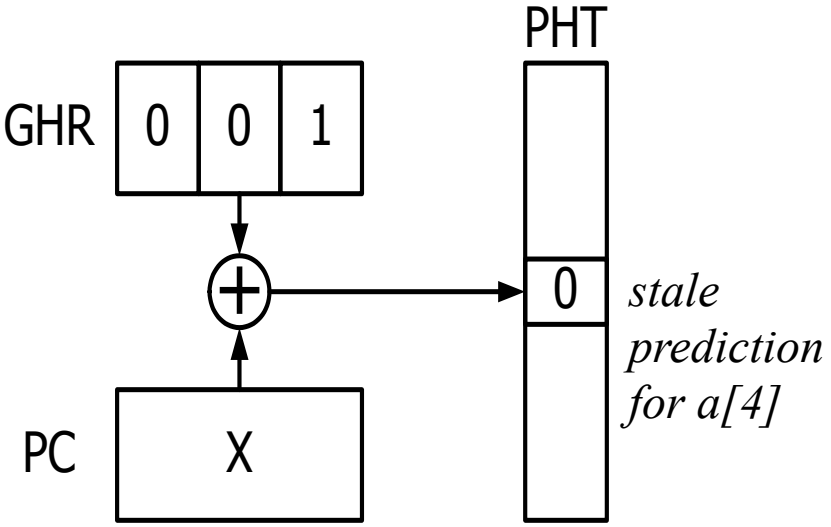
- Indirect solution
  - Use longer history.
  - GHR now distinguishes a[6], a[15].
  - Ambiguity not eradicated: a[10], a[15].
- Direct solution
  - Use address of element.
  - Dedicated predictions for different elements.



Bad Scenario #2



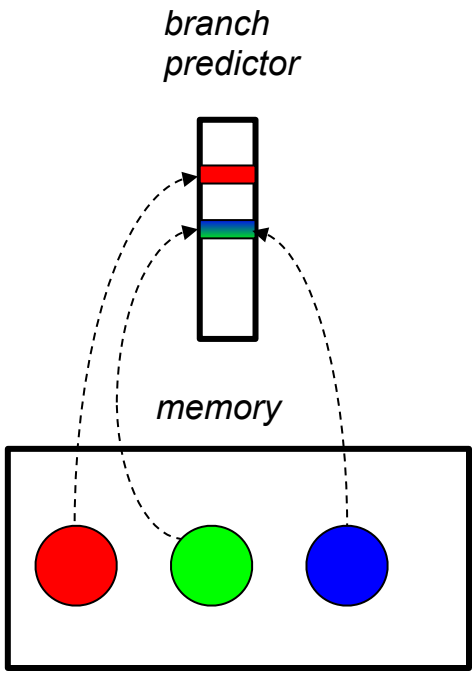
- STORE a[4] = 6
- Conventional passive updates
  - Predictor's entry is stale after store
  - Retrained after suffering misprediction
- Solution: Active updates



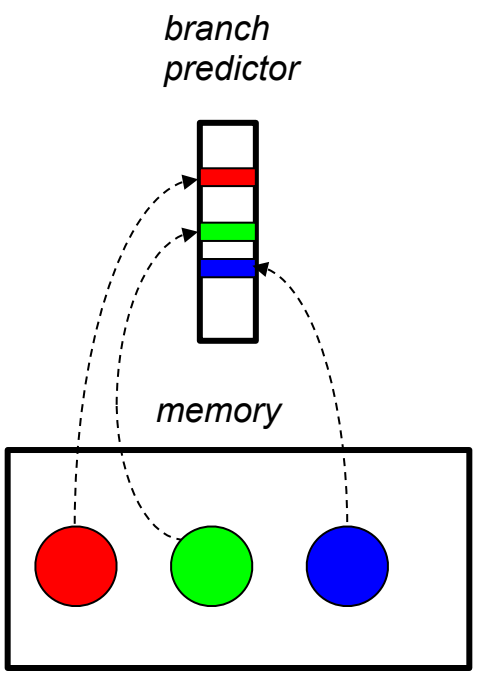
Big Picture

- Branch predictor should mirror a program's objects

Conventional



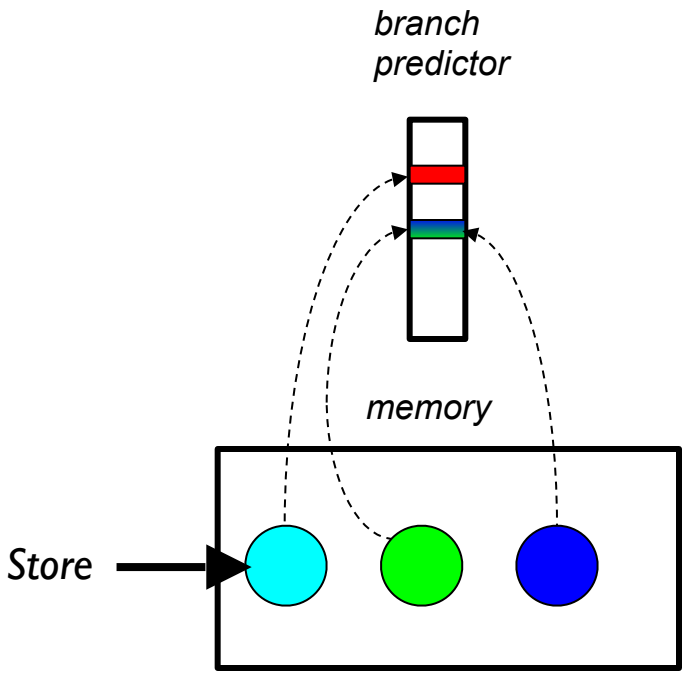
Proposed



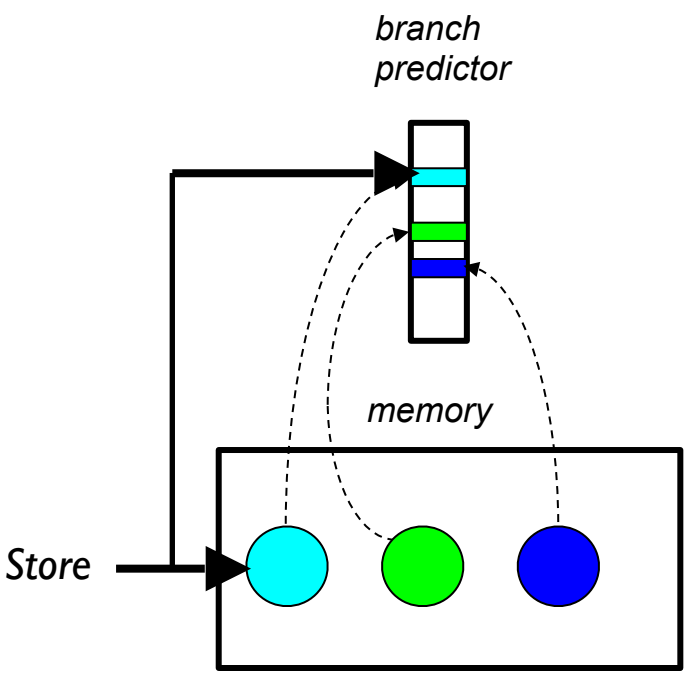
Big Picture

- Branch predictor should mirror a program's objects
- Branch predictor should mirror changes as they happen

Conventional



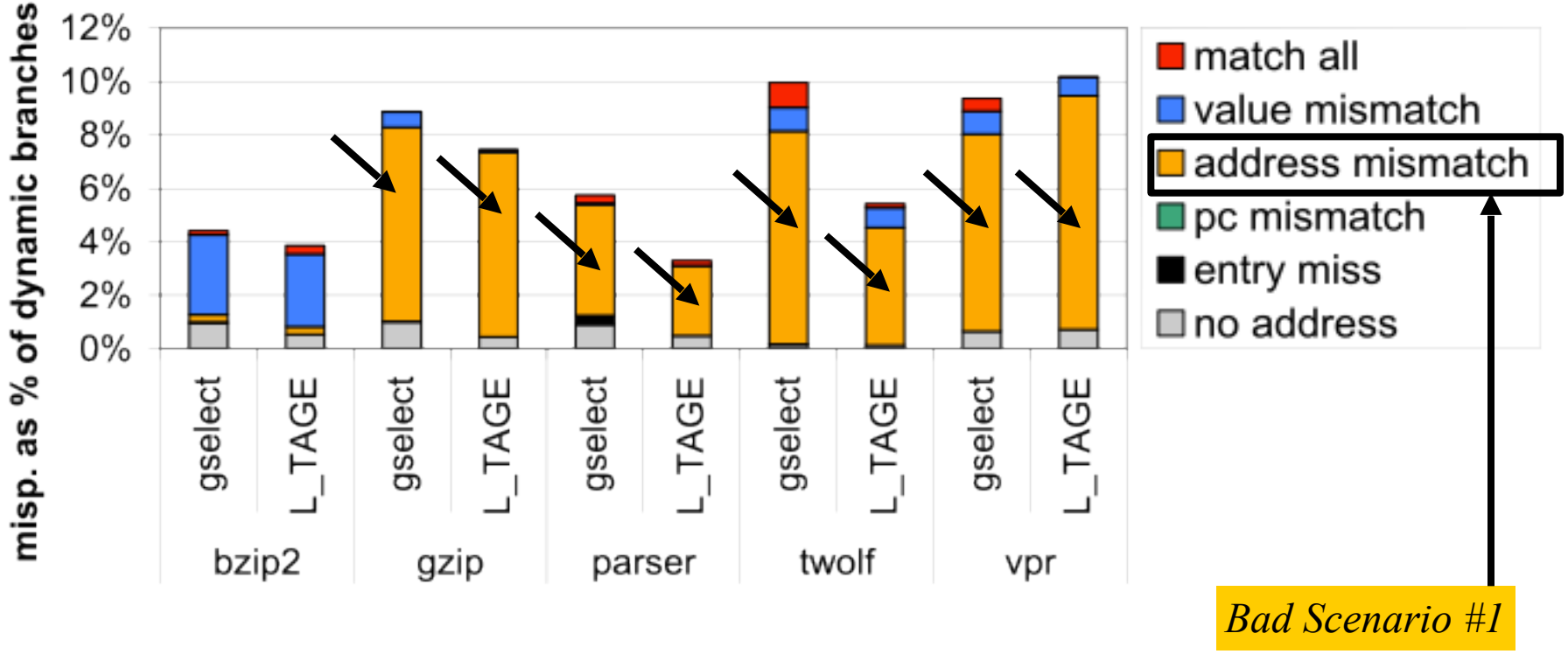
Proposed



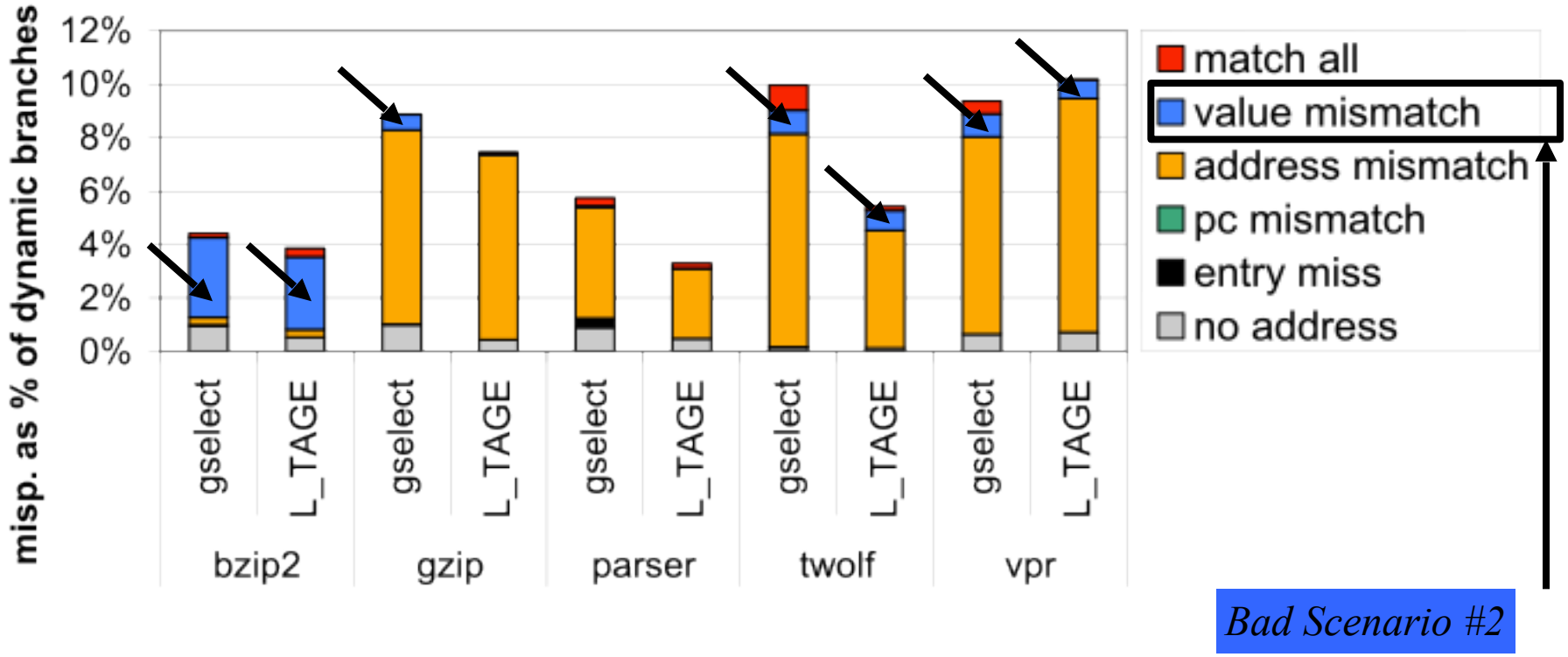
Characterize Mispredictions

- Bad Scenario #1
  - Measure how often global branch history does not distinguish different dynamic branches that have different outcomes
- Bad Scenario #2
  - Measure how often stores cause stale predictions
- Evaluate for two history-based predictors
  - Very large gselect
  - Very large L-TAGE

Characterize Mispredictions



Characterize Mispredictions

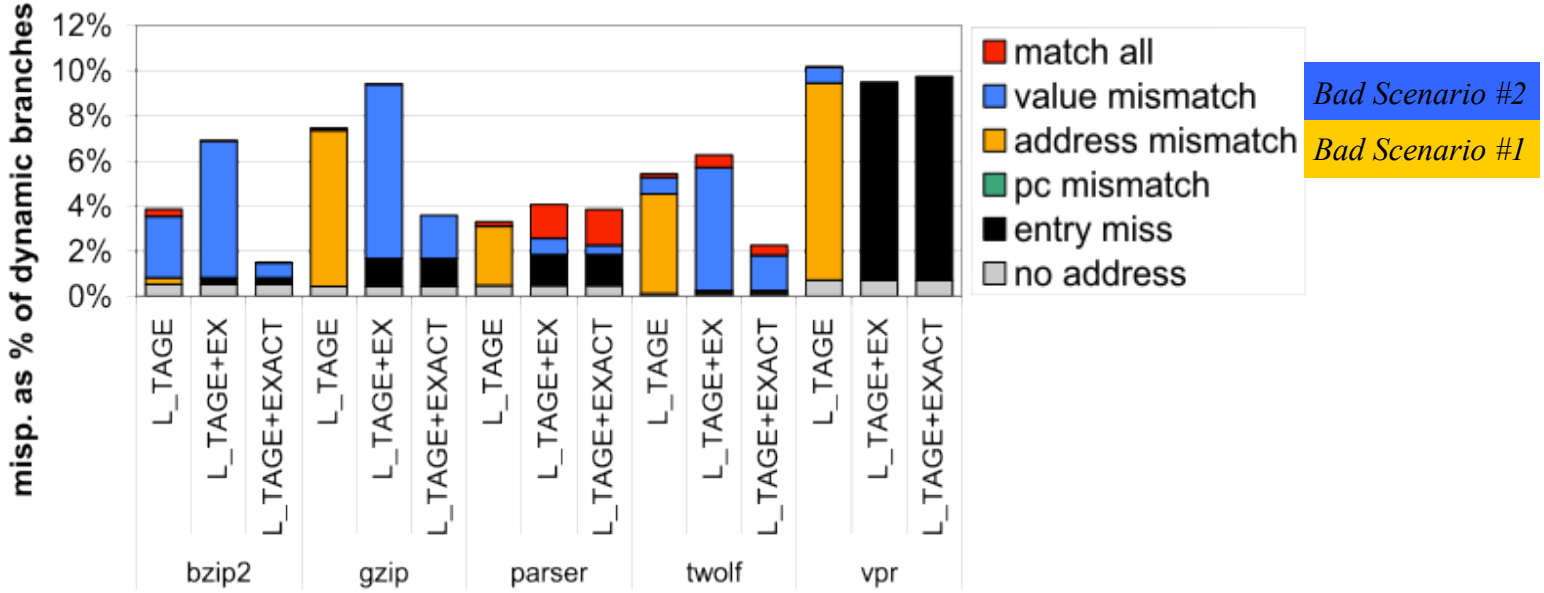


Note:
Predominance of Bad Scenario #1 may obscure occurrences of Bad Scenario #2.

Problems and Solutions

- Two problems:
  - {PC, global branch history} does not always distinguish different dynamic branches that have different outcomes
  - Stores to memory change branch outcomes

- Two solutions:
  - Explicitly identify dynamic branches to provide dedicated predictions for them **(EX)**
    - **branch ID = hash(PC, load addresses)**
  - Stores "actively update" the branch predictor **(ACT)**



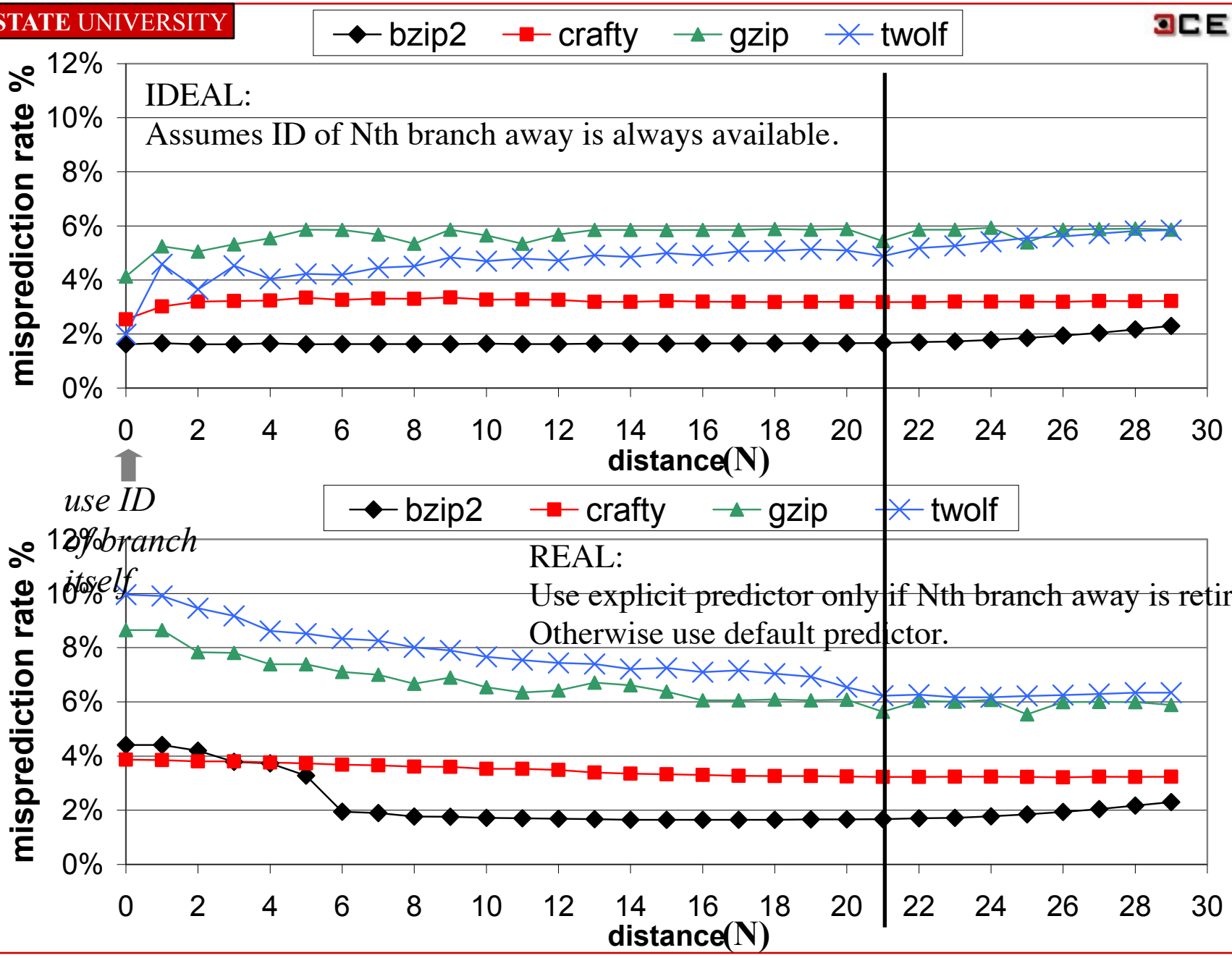
- Load-dependent branches use explicit predictor
  - Index with branch ID (EX)
  - Index with branch ID and perform active updates (EXACT)
- Other branches use the default predictor (*e.g.*, L-TAGE)

3 Implementation Challenges

1. Indexing the explicit predictor
   - Branch ID unknown at fetch time
   - Loads are unlikely to have computed their addresses by the time the branch is fetched
2. Large explicit predictor
   - Many different IDs contribute to mispredictions
   - Predictor size normally limited by cycle time
3. Large active update unit
   - Too large to implement with dedicated storage

Implementation Challenge #1: *Indexing the Explicit Predictor*

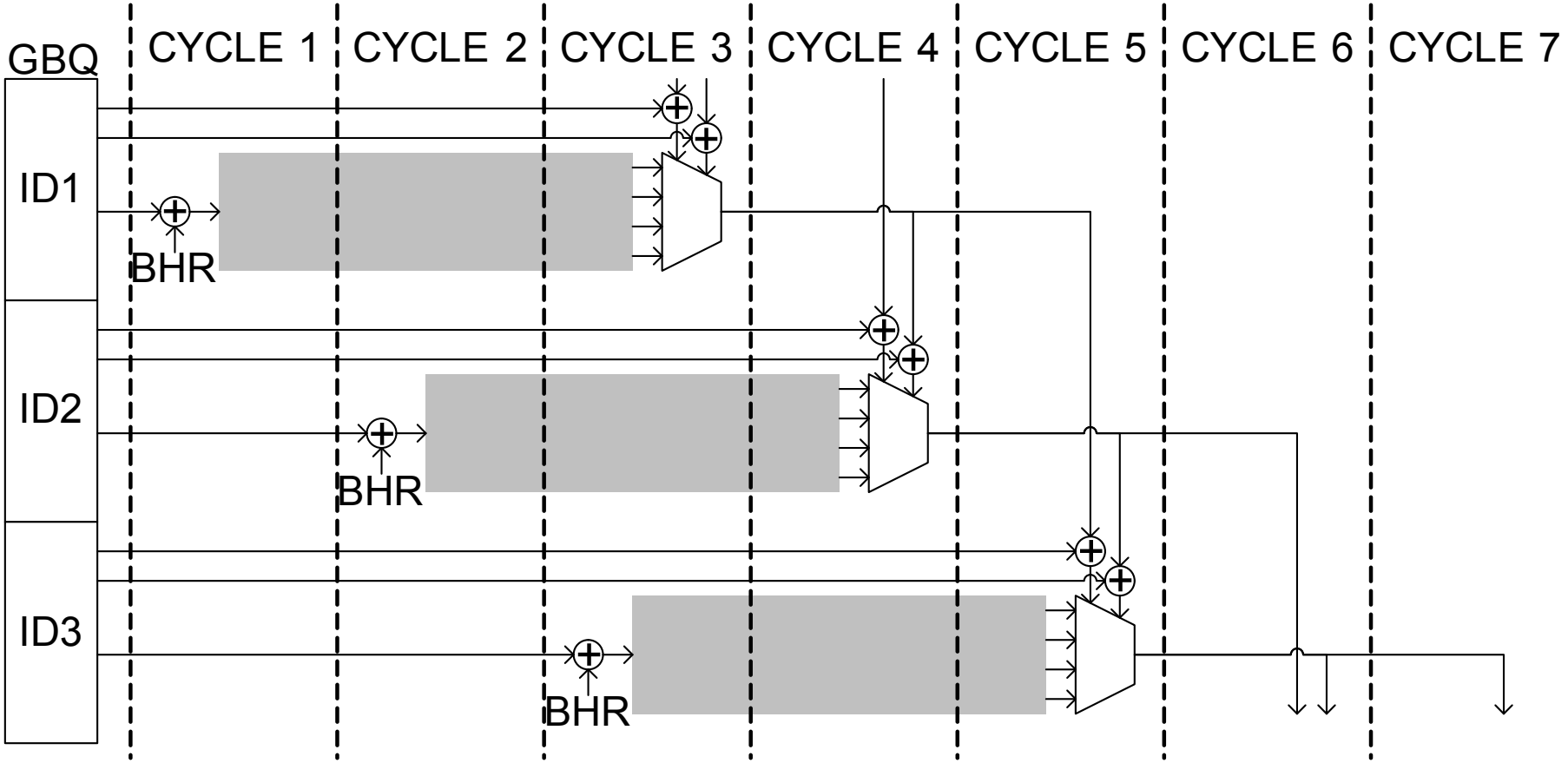
- Insight: sequences of IDs repeat due to traversing data structures
- Use ID of a prior retired branch at a fixed distance N


bzip2
crafty
gzip
twolf
misprediction rate %
12%
10%
8%
6%
4%
2%
0%
IDEAL:
Assumes ID of Nth branch away is always available.
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
distance(N)
use ID of branch itself
bzip2
crafty
gzip
twolf
misprediction rate %
REAL:
Use explicit predictor only if Nth branch away is retired.
Otherwise use default predictor.
distance(N)

Implementation Challenge #2: *Large Explicit Predictor*

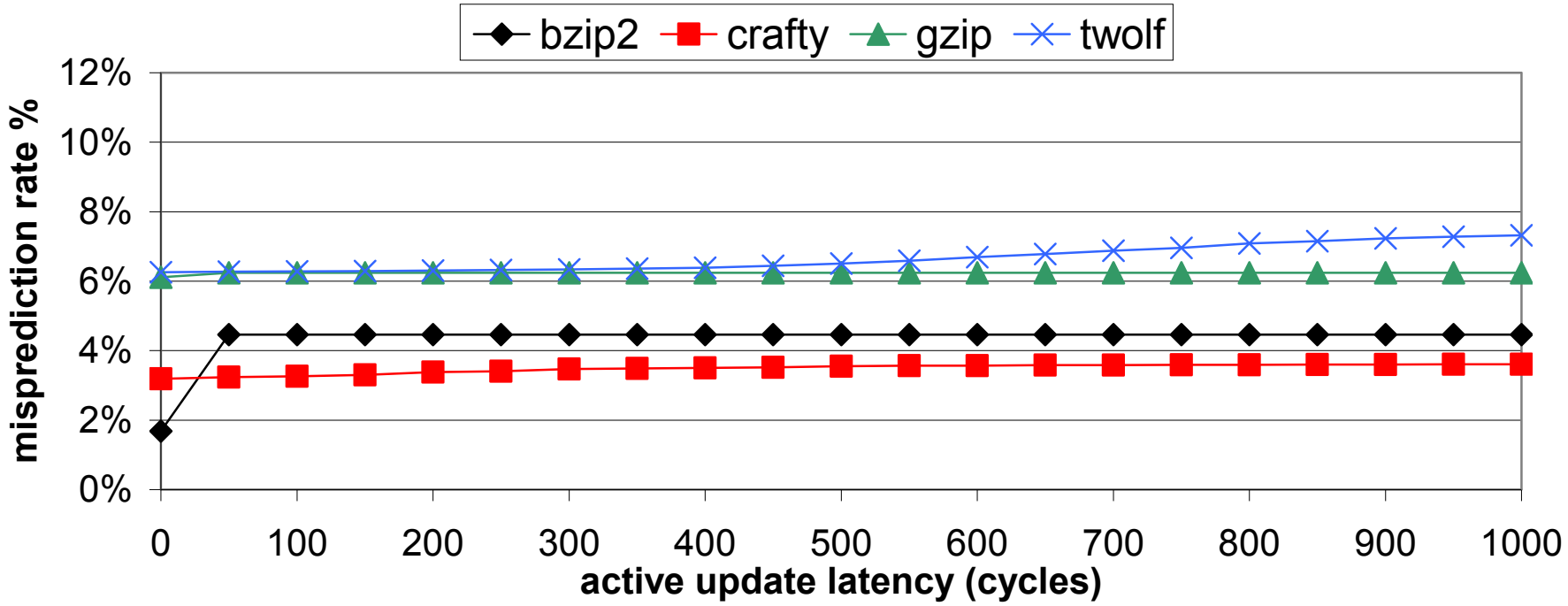
- Need large explicit predictor with fast cycle time
- Indexing with prior retired branch IDs makes it easily pipelinable
  - In general, pipelining is straightforward if the index does not depend on immediately preceding predictions

Implementation Challenge #2: *Large Explicit Predictor*



Implementation Challenge #3: *Large Storage for Active Updates*

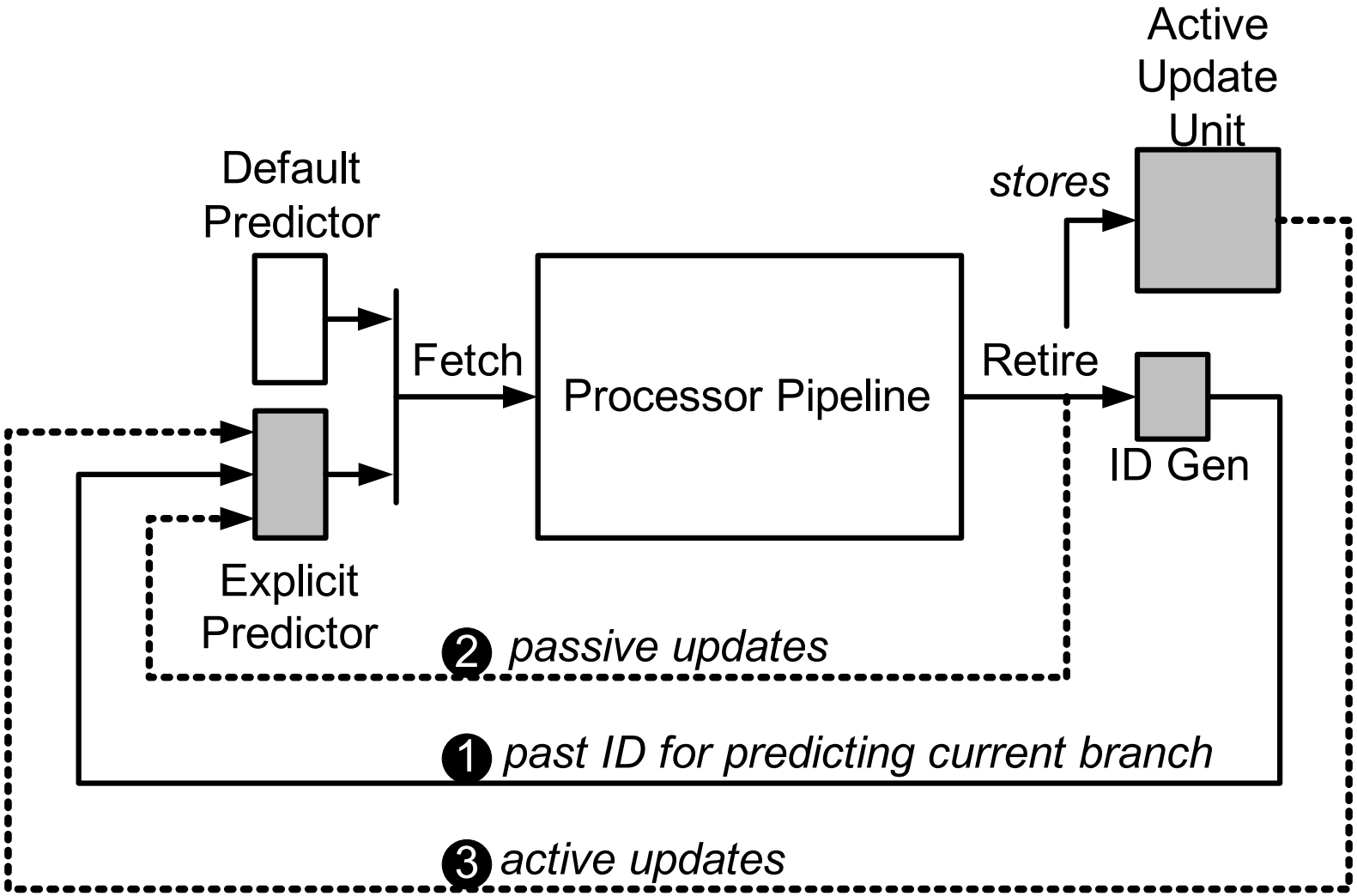
- Most benchmarks are tolerant of 400+ cycles of active update latency
- Large distance between stores and re-encounters of the branches they update

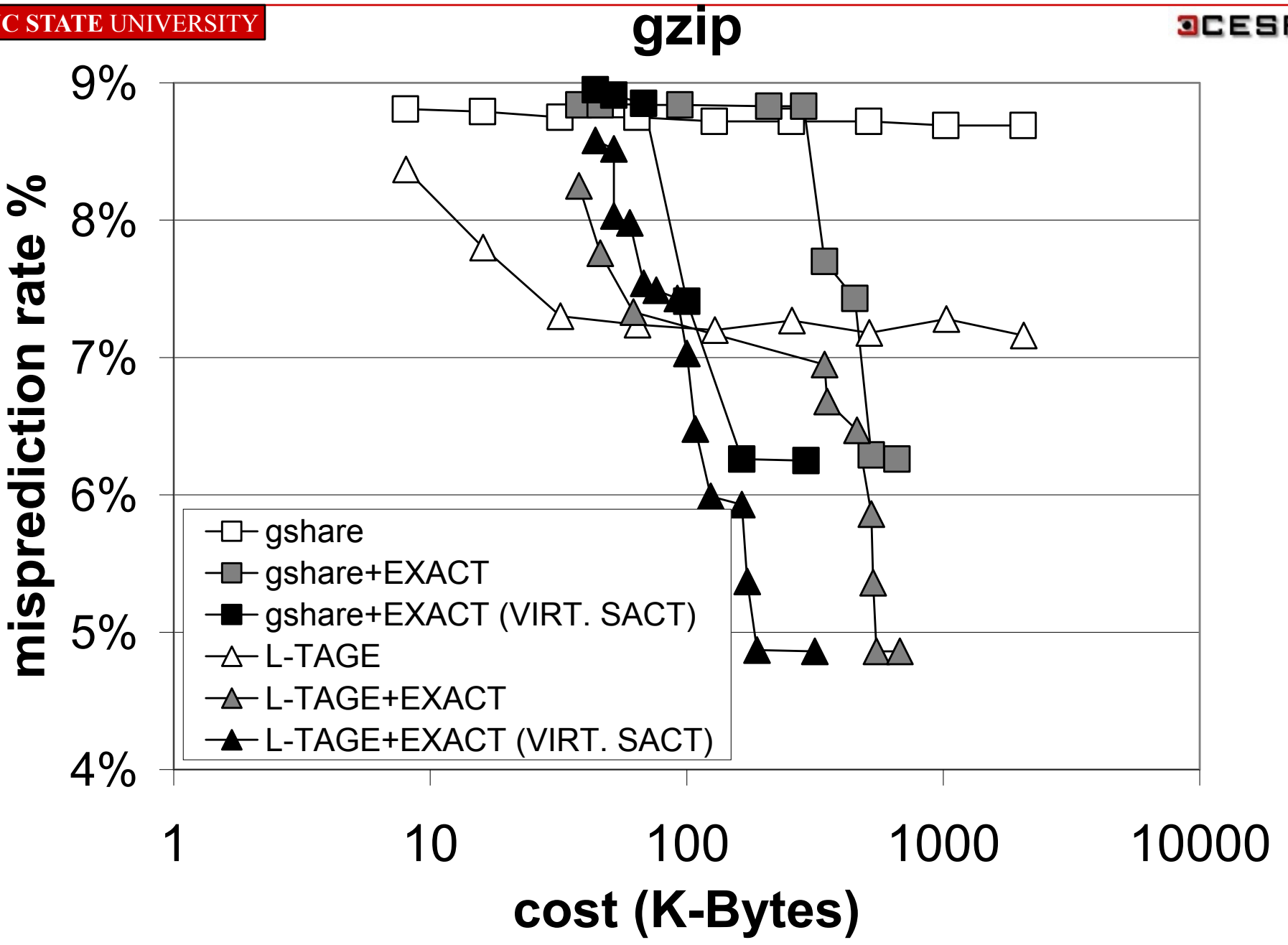


Implementation Challenge #3: *Large Storage for Active Updates*

- Exploit "Predictor Virtualization" [Burcea et al.]
- Eliminate significant amount of dedicated storage
- Use small L1 table backed by full table in physical memory
  - The full table in physical memory is transparently cached in the general-purpose memory hierarchy (e.g., L2$)

Putting it all together

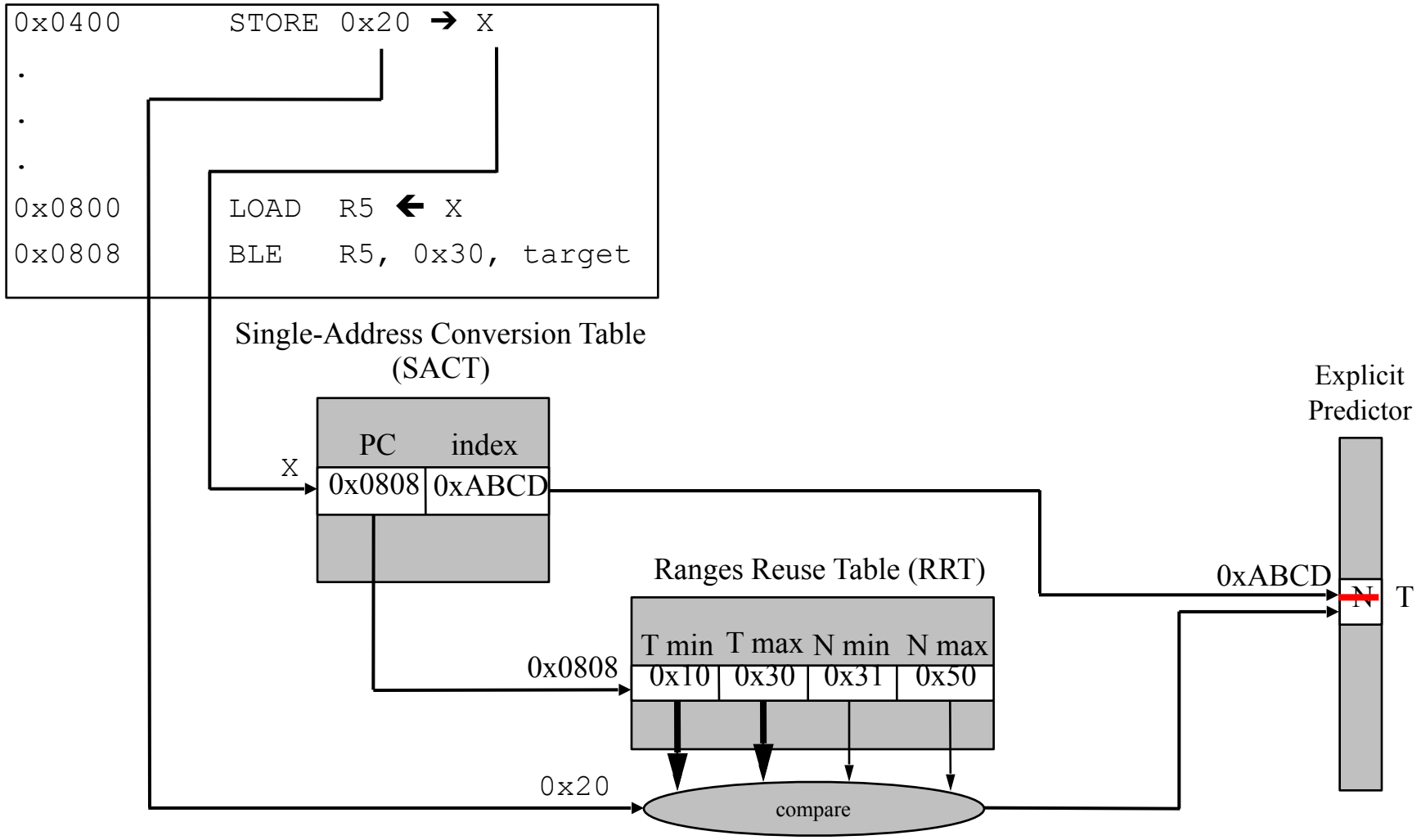



gzip
misprediction rate %
9%
8%
7%
6%
5%
4%
1
10
100
1000
10000
cost (K-Bytes)
gshare
gshare+EXACT
gshare+EXACT (VIRT. SACT)
L-TAGE
L-TAGE+EXACT
L-TAGE+EXACT (VIRT. SACT)

Active Update Unit

- First proposal to use store instructions to update the branch predictor
- Store instructions might:
  - Change a branch outcome in the explicit predictor
  - Change a trip-count in the explicit loop predictor
- Two mechanisms required:
  - Convert store address into a predictor index
  - Convert store value into a branch-outcome or trip-count

Active Update Example



Future Work

- Rich ISA support
  - Empower compiler or programmer to directly index into the explicit predictor, directly manage it, and directly manage the instruction fetch unit
  - Close gap between real and ideal indexing
  - More efficient hardware