332
Advanced Computer Architecture
Chapter 4

Branch Prediction

February 2018

Paul H J Kelly

These lecture notes are partly based on the course text, Hennessy and Patterson's *Computer Architecture, a quantitative approach (4-6th eds), and on the lecture slides of David Patterson's Berkeley course (CS252)*

Course materials online at
http://www.doc.ic.ac.uk/~phjk/AdvancedCompArchitecture.html

Branch Prediction

1. Control hazards are a problem in any pipelined processor
2. Branches occur a lot (ca. one in five?)
   - Branches will arrive up to *n* times faster in an *n*-issue processor
3. Amdahl's Law:
   - relative impact of the control stalls will be larger with the lower potential CPI in an *n*-issue processor
4. Speculative dynamic instruction scheduling with register renaming enables us to speculate *many* instructions
   - Forwarding from one speculatively-executed instruction to the next

Branch prediction is *really* important….

Branch Prediction - alternatives

- We have seen how a dynamically-scheduled processor can handle speculative execution past conditional branches, virtual calls, page faults etc
- But branch mis-predictions are expensive
- This naturally leads us to consider branch prediction schemes

- But first: there are alternatives…
  - With enough threads per core…
  - By extending the instruction set with predication
  - By extending the instruction set with branch delays

Delayed Branch

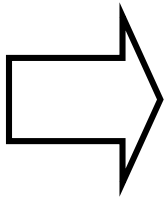
– Define branch to take place AFTER a following instruction

branch instruction
  sequential successor$_1$
  sequential successor$_2$
  .....
  sequential successor$_n$
branch target if taken

Branch delay of length *n*

– 1 slot delay allows proper decision and branch target address in 5 stage pipeline
– MIPS uses this; eg in

```
  LW R3, #100
  LW R4, #200
  BEQZ R1, L1
  SW R3, X
  SW R4, X
L1:
  LW R5,X
```

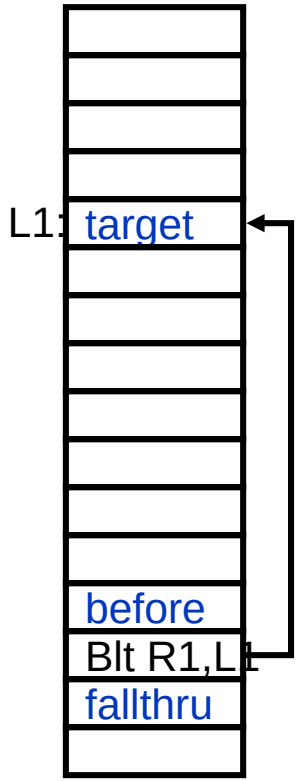


```
If (R1==0)
   X=100
Else
   X=100
   X=200
R5 = X
```

– “SW R3, X” instruction is executed regardless
– “SW R4, X” instruction is executed only if R1 is non-zero

Delayed Branch

- Where to get instructions to fill branch delay slot?
  - Before branch instruction
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken



- **Compiler effectiveness for single branch delay slot:**
  - **Fills about 60% of branch delay slots**
  - **About 80% of instructions executed in branch delay slots useful in computation**
  - **About 50% (60% x 80%) of slots usefully filled**
- **Delayed Branch downside: 7-8 stage pipelines, multiple instructions issued per clock (superscalar)**
- **Canceling branches**
  - **Branch delay slot instruction is executed but write-back is disabled if it is not supposed to be executed**
  - **Two variants: branch “likely taken”, branch “likely not-taken”**
  - **allows more slots to be filled**

Branch Prediction - context

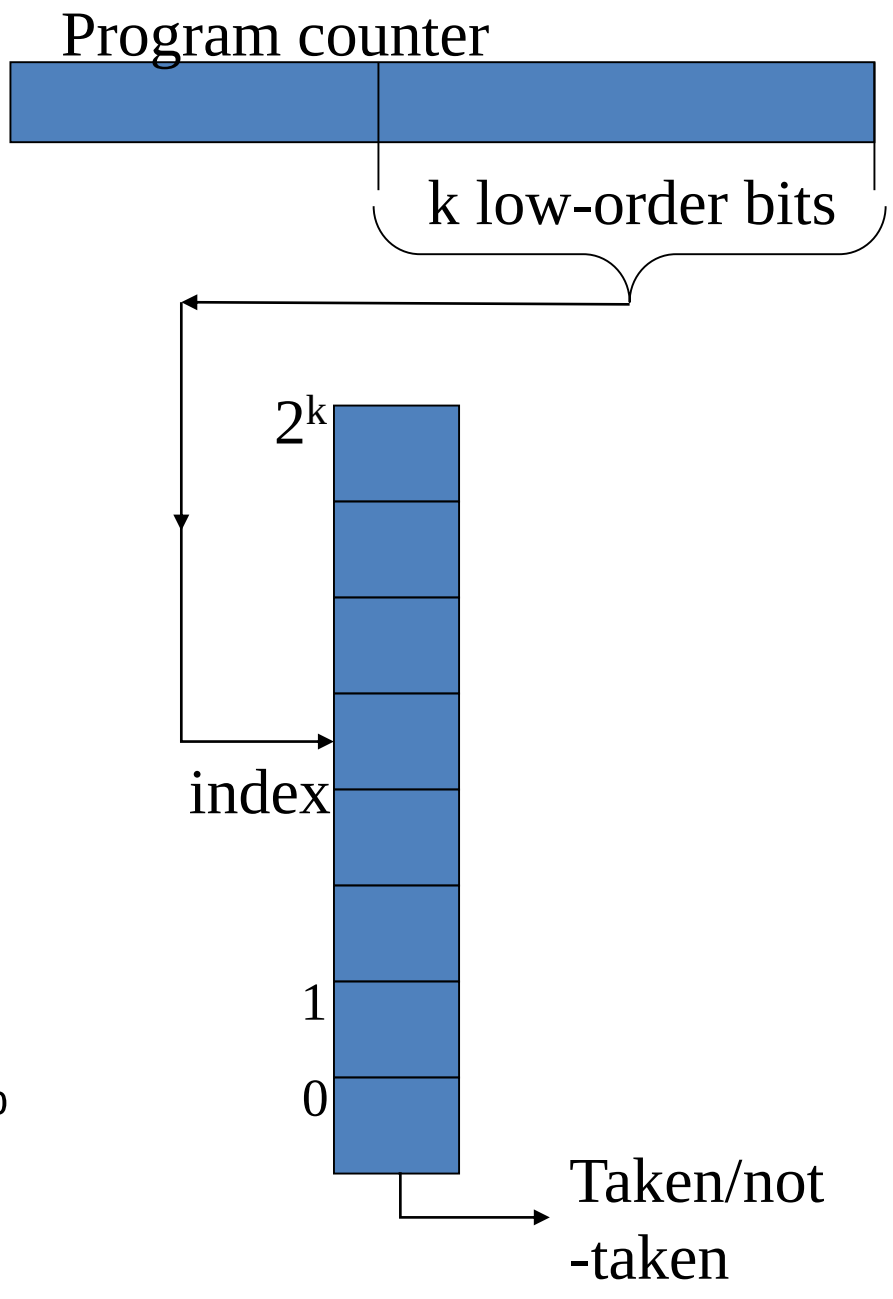
- If we have a branch predictor….
  - We want to fetch the correct (predicted) next instruction without any stalls
  - We need the prediction before the preceding instruction has been decoded
  - We need to predict conditional branches
  - And indirect branches
    - Direction prediction
    - Target prediction

7 Branch Prediction Schemes

1. 1-bit Branch-Prediction Buffer
2. 2-bit Branch-Prediction Buffer
3. Correlating Branch Prediction Buffer
4. Tournament Branch Predictor
5. Branch Target Buffer
6. Integrated Instruction Fetch Units
7. Return Address Predictors

Simplest idea: branch history table (BHT)

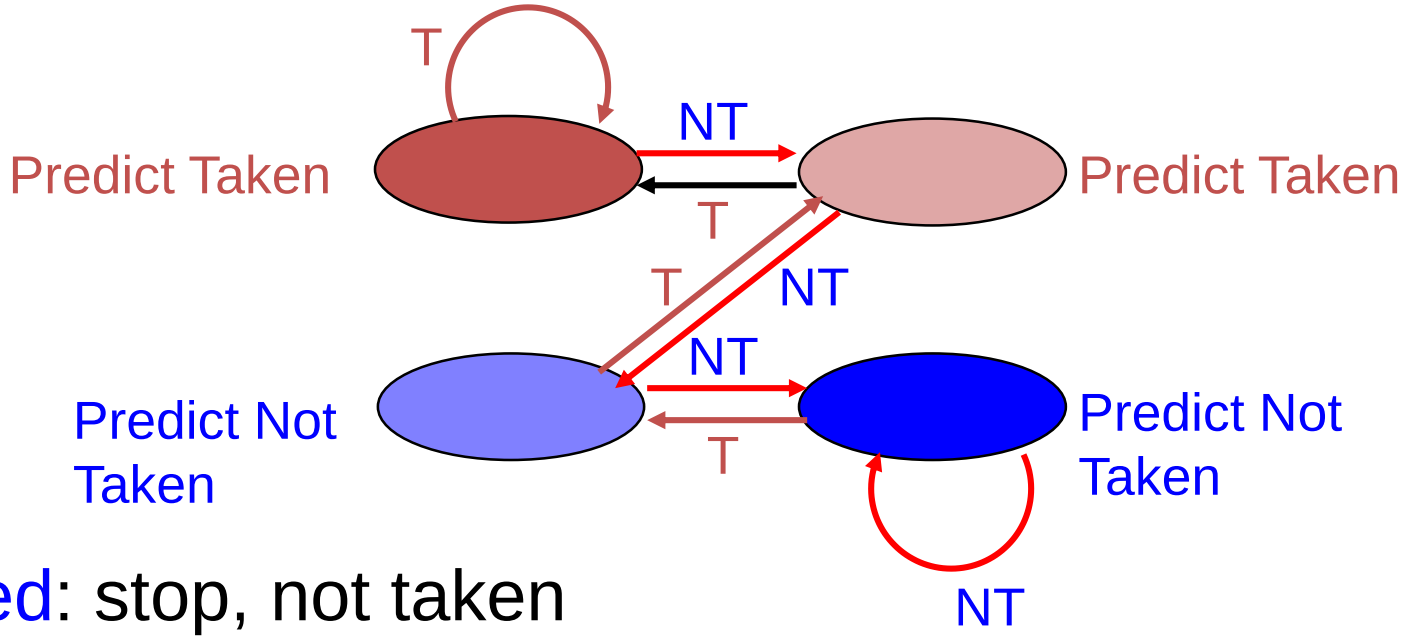
- Lower bits of PC address index table of 1-bit values
  - Says whether or not branch taken last time
  - No address check (saves HW, but may not be right branch)
- Problem: in a loop, 1-bit BHT will cause 2 mispredictions (avg is 9 iterations before exit):
  - End of loop case, when it exits instead of looping as before
  - First time through loop on *next* time through code, when it predicts *exit* instead of looping
  - Only 80% accuracy even if loop 90% of the time



Dynamic Branch Prediction

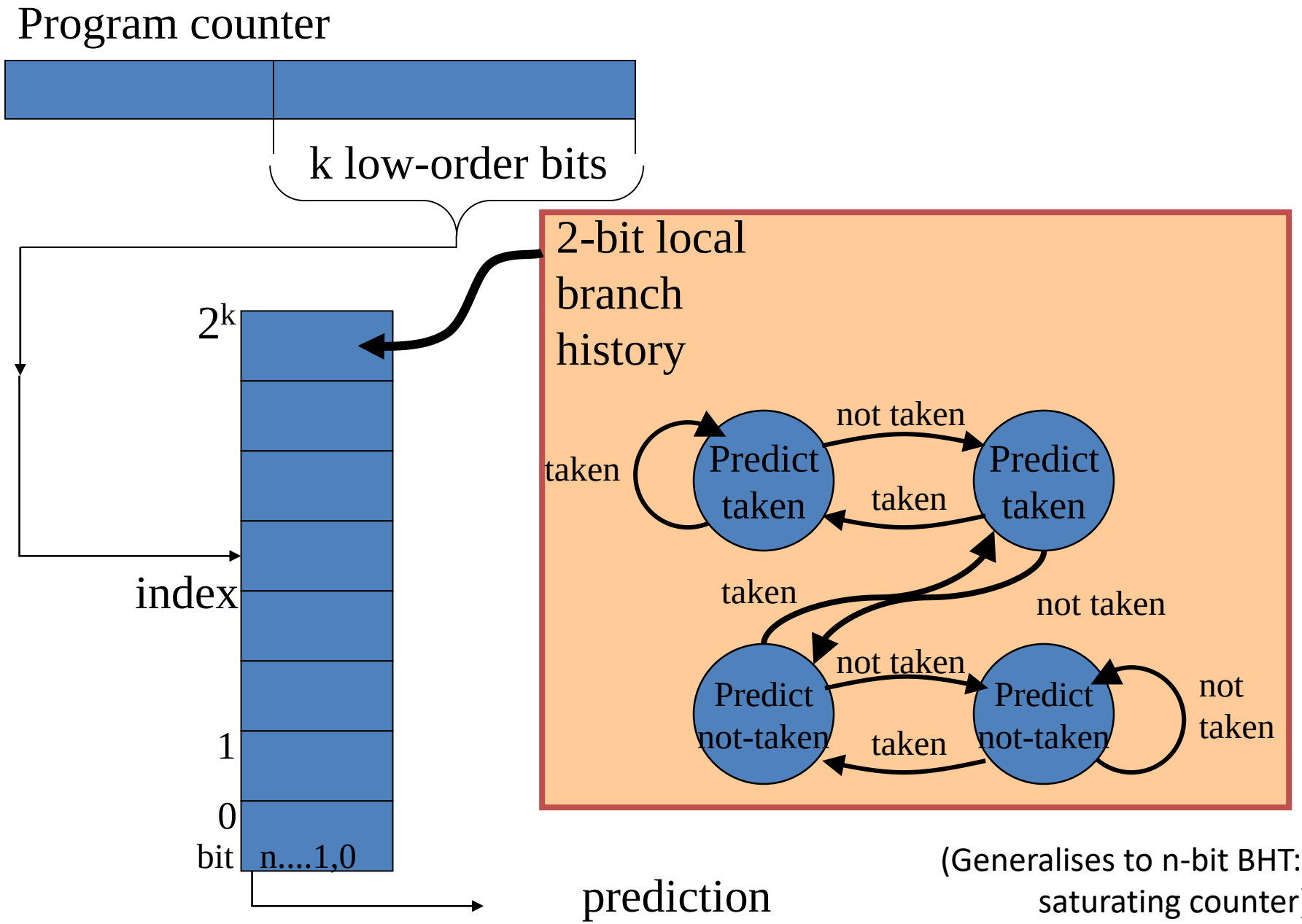
(Jim Smith, 1981)

- Solution: 2-bit scheme where change prediction only if get misprediction *twice:* (Figure 3.7, p. 198)



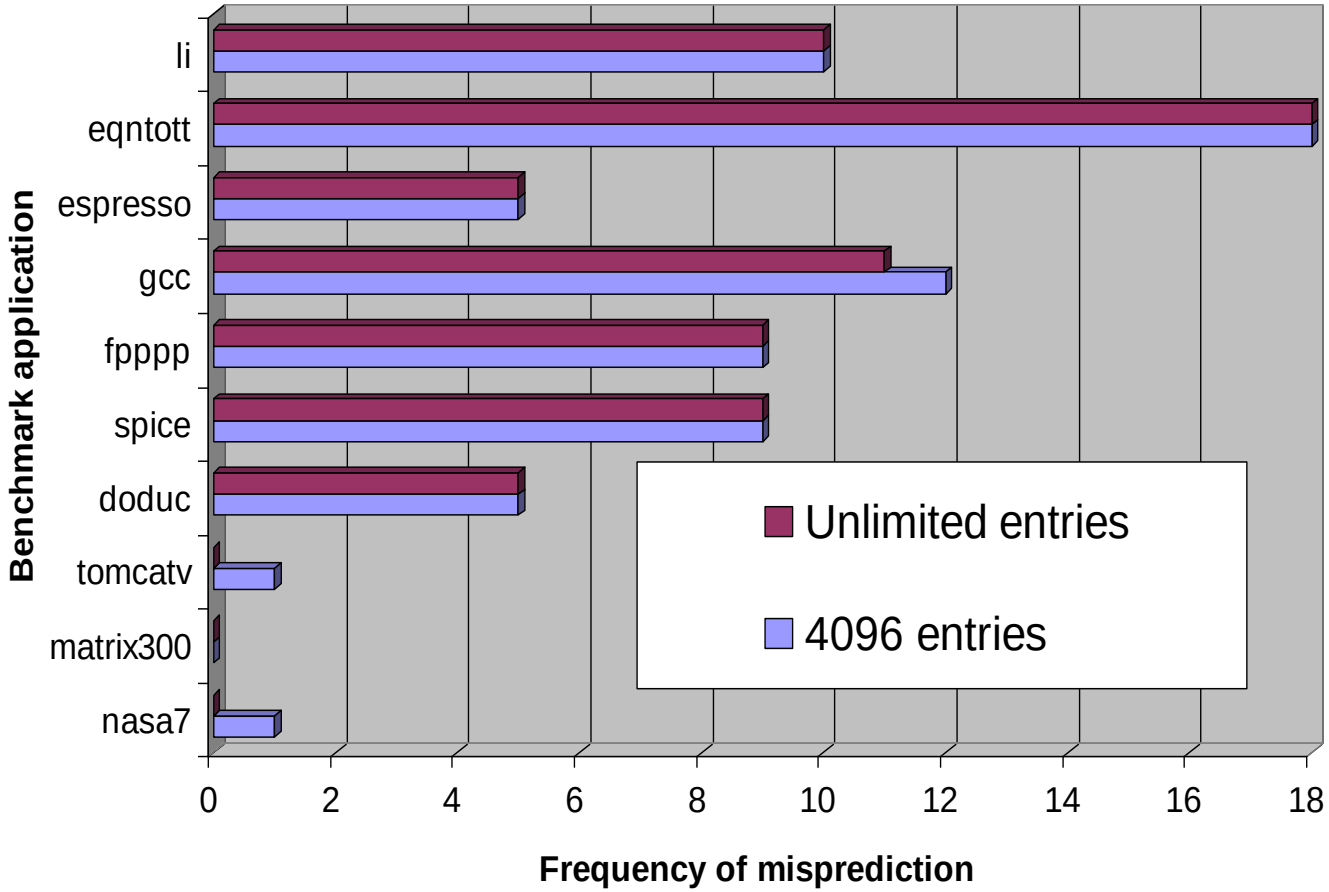
- Red: stop, not taken
- Green: go, taken
- Adds *hysteresis* to decision making process

The 2-bit branch history table (BHT)



(Generalises to n-bit BHT: saturating counter)

Prediction accuracy of an 4096-entry two-bit prediction buffer versus an infinite buffer for the SPEC89 benchmarks (H&P Fig 4.15)



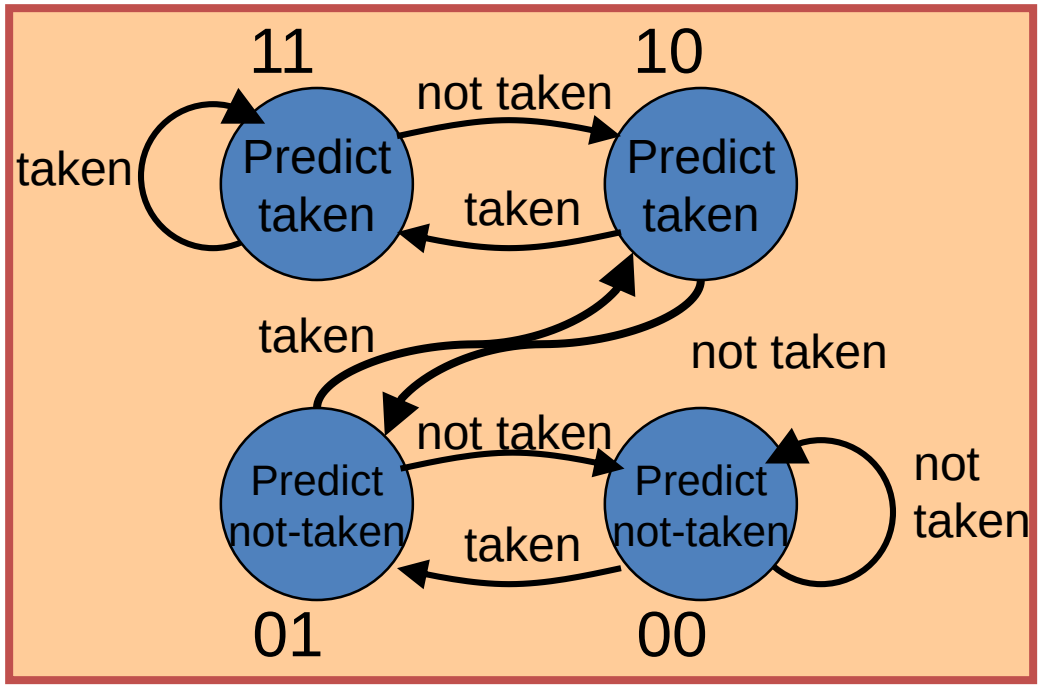
n-bit BHT - how well does it work?

- 2-bit predictor often very good, sometimes awful
- Little evidence that BHT capacity is an issue
- 1-bit is usually worse, 3-bit is not usefully better

N-bit BHT - why does it work so well?

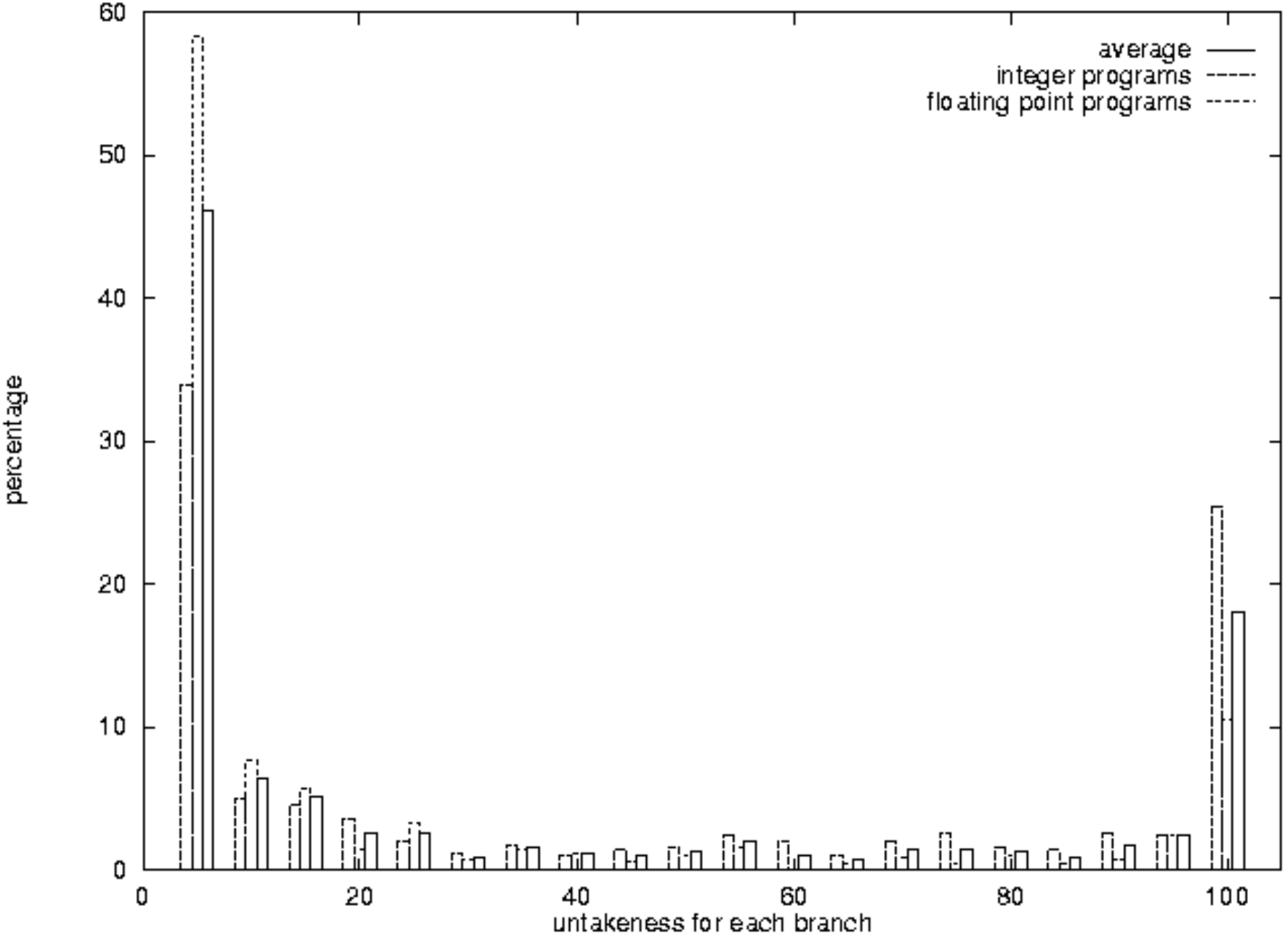
- n-bit BHT predictor essentially based on a saturating counter: taken increments, not-taken decrements
- predict taken if most significant bit is set

- Most branches are highly *biased:* either almost-always taken, or almost-always not-taken
- Works badly for branches which aren't



Often called the "bimodal" predictor

Bias



Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)

Is local history all there is to it?

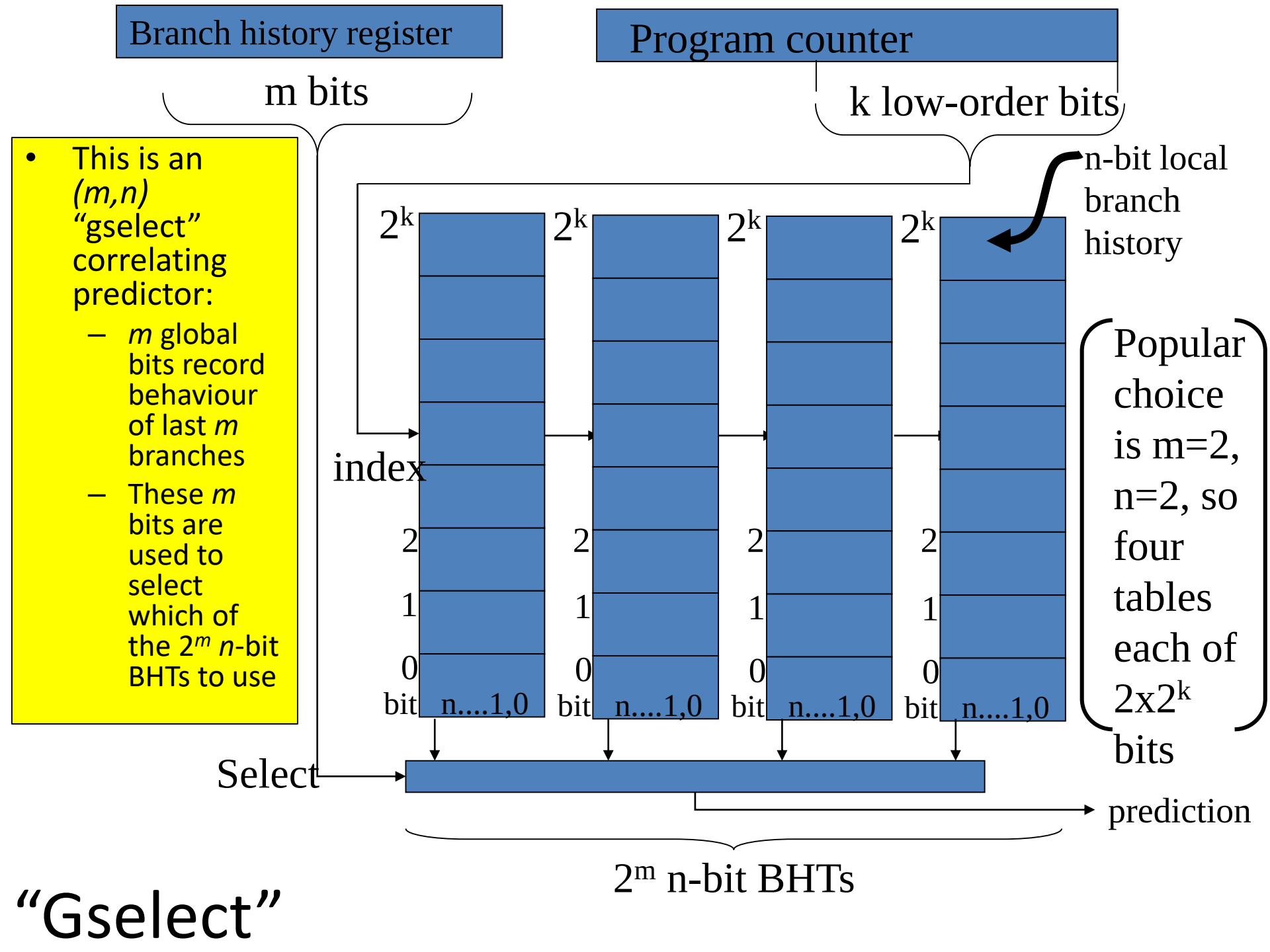
- The bimodal predictor uses the BHT to record “local history” - the prediction information used to predict a particular branch is determined only by its memory address
- Consider the following sequence:

```
if (C1)  then
    S1;
endif
if (C2) then
    S2;
endif
if (C3) then
    S3;
endif
```

  - **It is very likely that condition C2 is correlated with C1 - and that C3 is correlated with C1 and C2**
  - **How can we use this observation?**

Global history

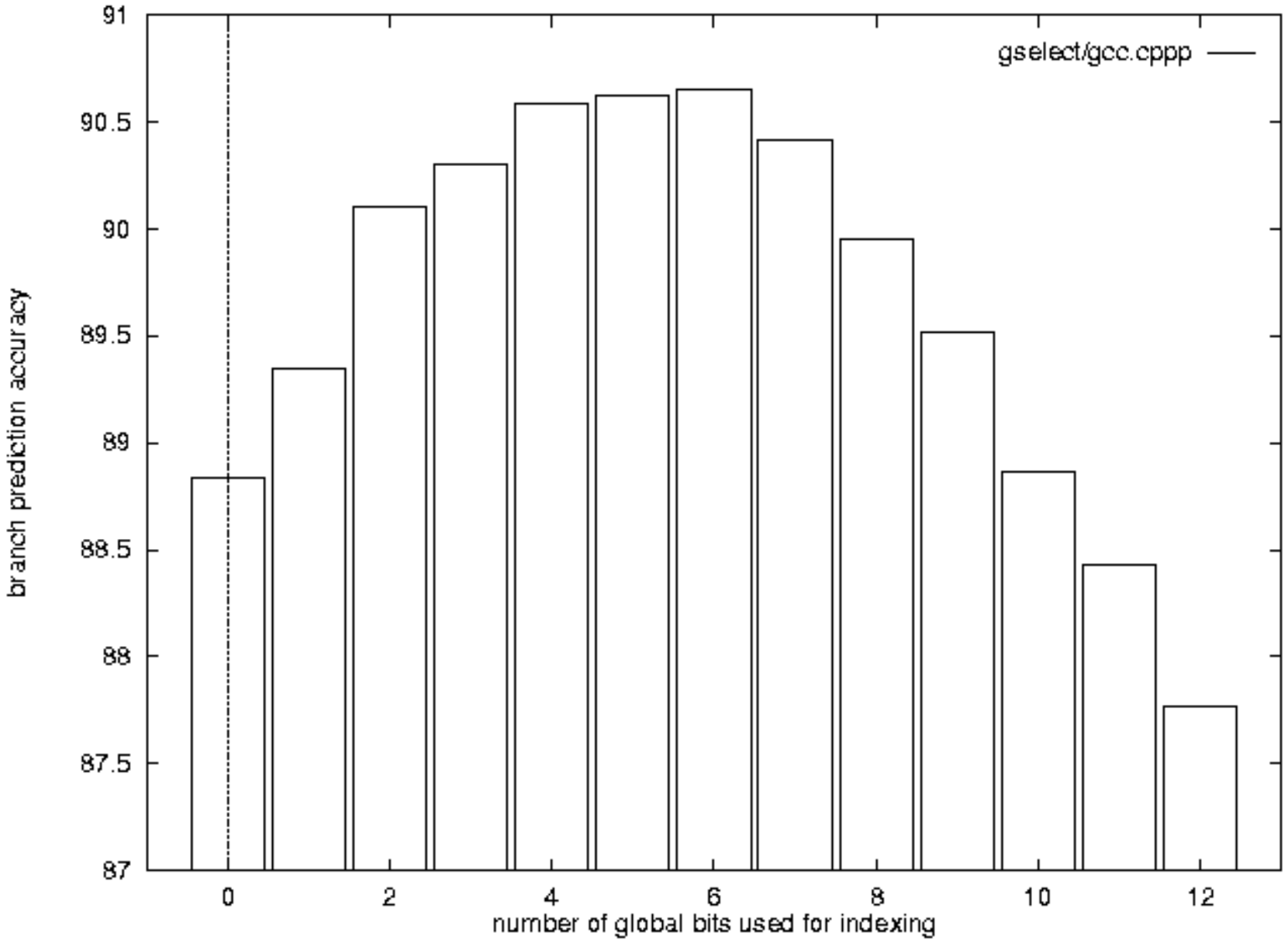
- Definition: Global history. The taken - not-taken history for all previously-executed branches.
  - **Idea: use global history to improve branch prediction**
- Compromise: use *m* most recently-executed branches
  - Implementation**: keep an *m*-bit Branch History Register (BHR) - a shift register recording taken - not-taken direction of the last m branches**
- Question: How to combine local information with global information?



- This is an *(m,n)* "gselect" correlating predictor:
  - *m* global bits record behaviour of last *m* branches
  - These *m* bits are used to select which of the 2^m *n*-bit BHTs to use

"Gselect"

How many bits of branch history should be used?



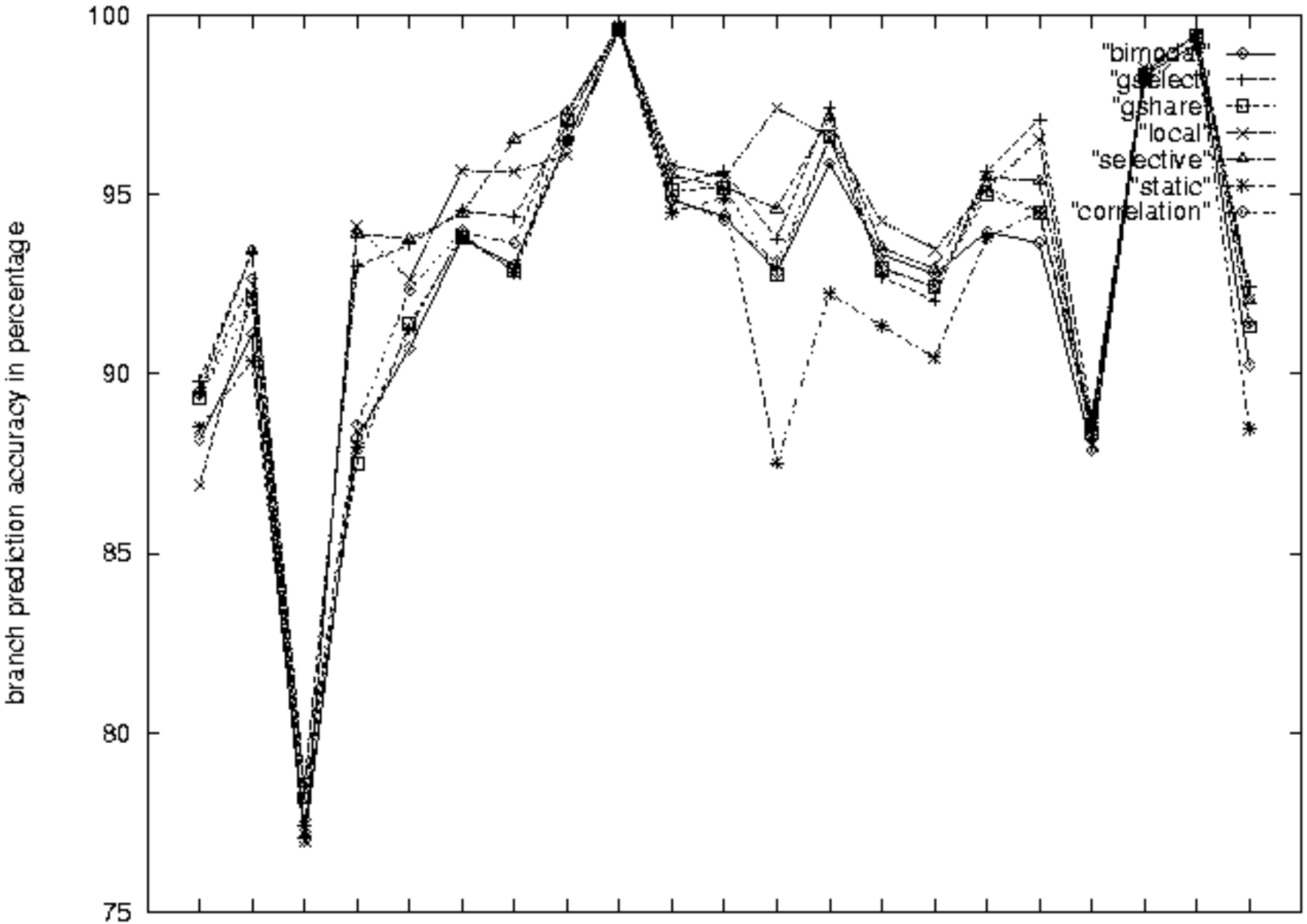
- (2,2) is good, (4,2) is better, (10,2) is worse

Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)

Variations

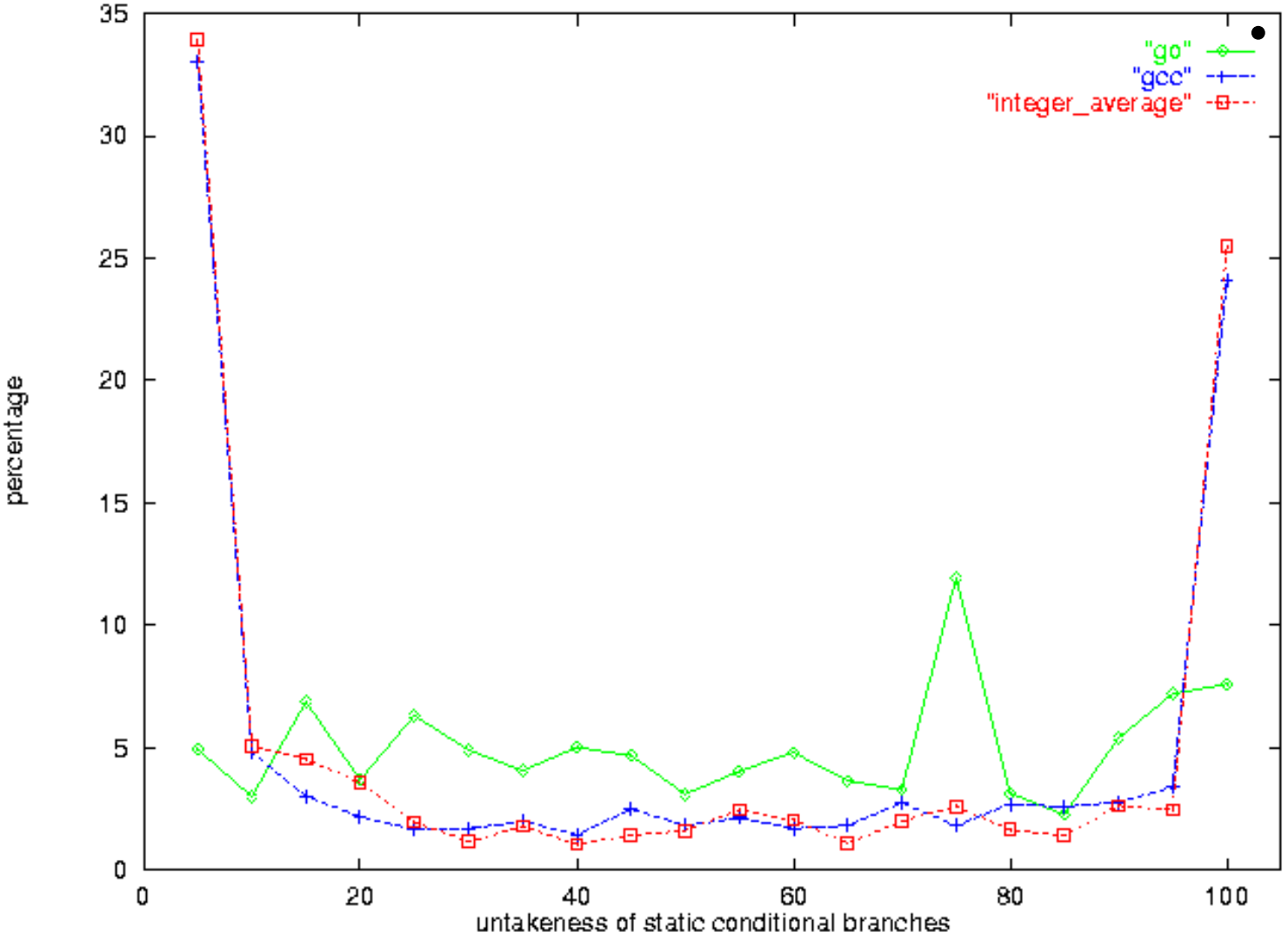
- There are many variations on the idea:
  - ***gselect***: many combinations of *n* and *m*
  - ***global***: use *only* the global history to index the BHT - ignore the PC of the branch being predicted (an extreme (n,m) gselect scheme)
  - ***gshare***: arrange bimodal predictors in single BHT, but construct its index by XORing low-order PC address bits with global branch history shift register - claimed to reduce conflicts
  - ***Per-address Two-level Adaptive using Per-address pattern history* (PAp)**: for each branch, keep a *k*-bit shift register recording its history, and use this to index a BHT *for this branch* (see Yeh and Patt, 1992)
- Each suits some programs well but not all

Horses for courses



Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)

Extreme example - “go”

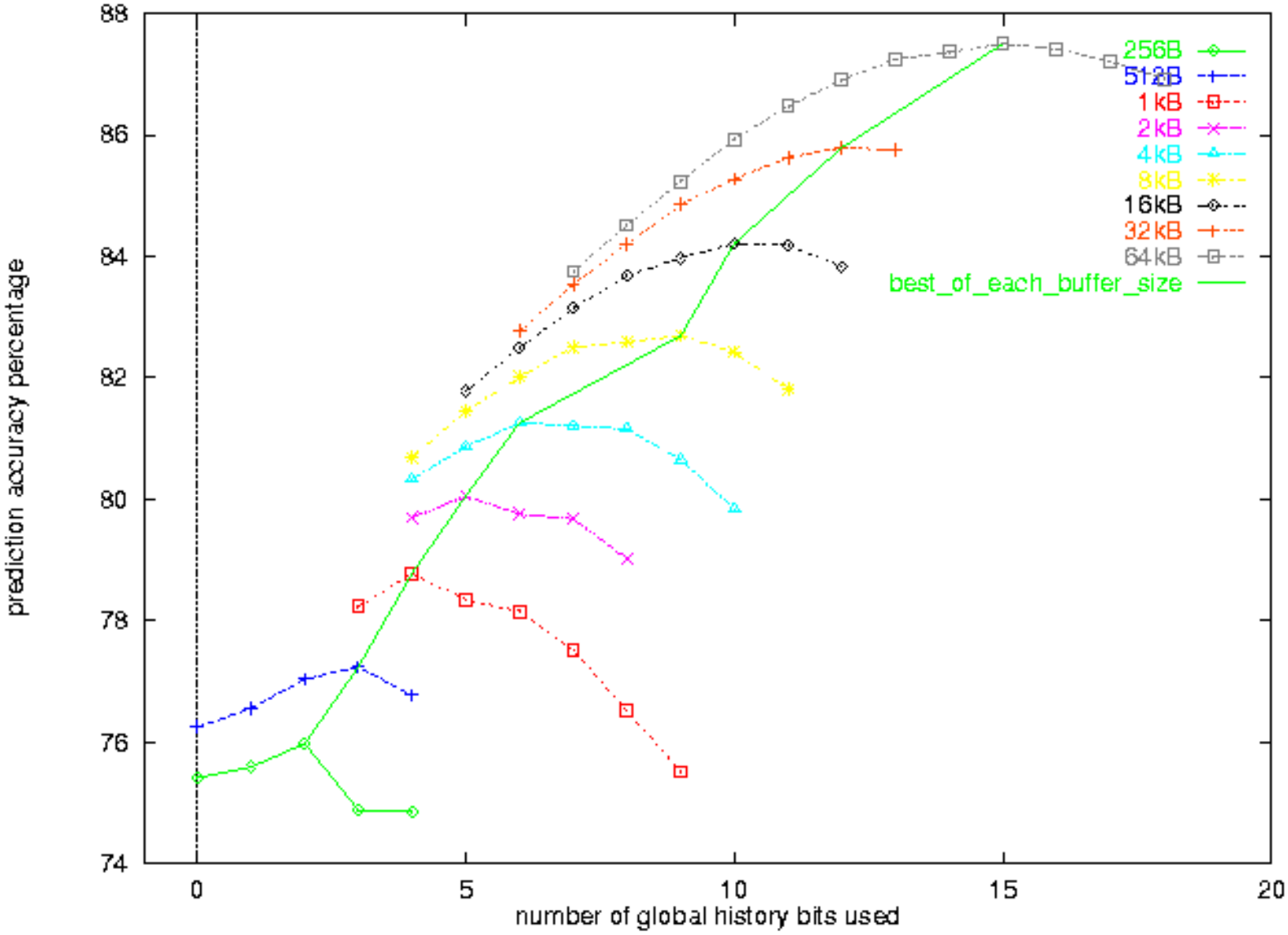


“go” is a SPEC95 benchmark code with highly-dynamic, highly-correlated branch behaviour

- The bias of “go”s branches is more-or-less evenly spread between 0% taken and 100% taken
- All known predictors do badly

Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)

Some dynamic applications have highly-correlated branches



- For “go”, optimum BHR size (m) is much larger

Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)

Re-evaluating Correlation

- Several of the SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:

program	branch %	static	# = 90%
compress	14%	236	13
eqntott	25%	494	5
gcc	15%	9531	2020
mpeg	10%	5598	532
real gcc	13%	17361	3214

- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?

Predicated Execution (predic*a*ted…)

- Avoid branch prediction by turning branches into conditionally executed instructions:

```
:
:
if (x == 10)
   c = c + 1;
:
:
```

```
      :
      LDR r5, X
      p1 <- r5 eq 10
<p1>  LDR  r1 <- C
<p1>  ADD r1, r1, 1
<p1>  STR  r1 -> C
      :
```

Some instruction sets allow predication of almost any instruction

- Load condition value into a predicate register
- Each instruction specifies which predicate register it depends on
- If predicate is false, no exception or effect occurs
- Compiler can schedule instructions from different conditional branches t fill stalls

(Some instruction sets offer only partial support, eg predicated moves/stores, eg Alpha, MIPS, PowerPC, SPARC)

When is this better than a conditional branch instruction?

Tournament Predictors

- Motivation for correlating branch predictors is that the 2-bit predictor failed on important branches; by adding global information, performance improved

- Tournament predictors: use 2 predictors,
  - one based on global information
  - the other based on local information
  - and combine with a selector
  - The selector is driven by a predictor….

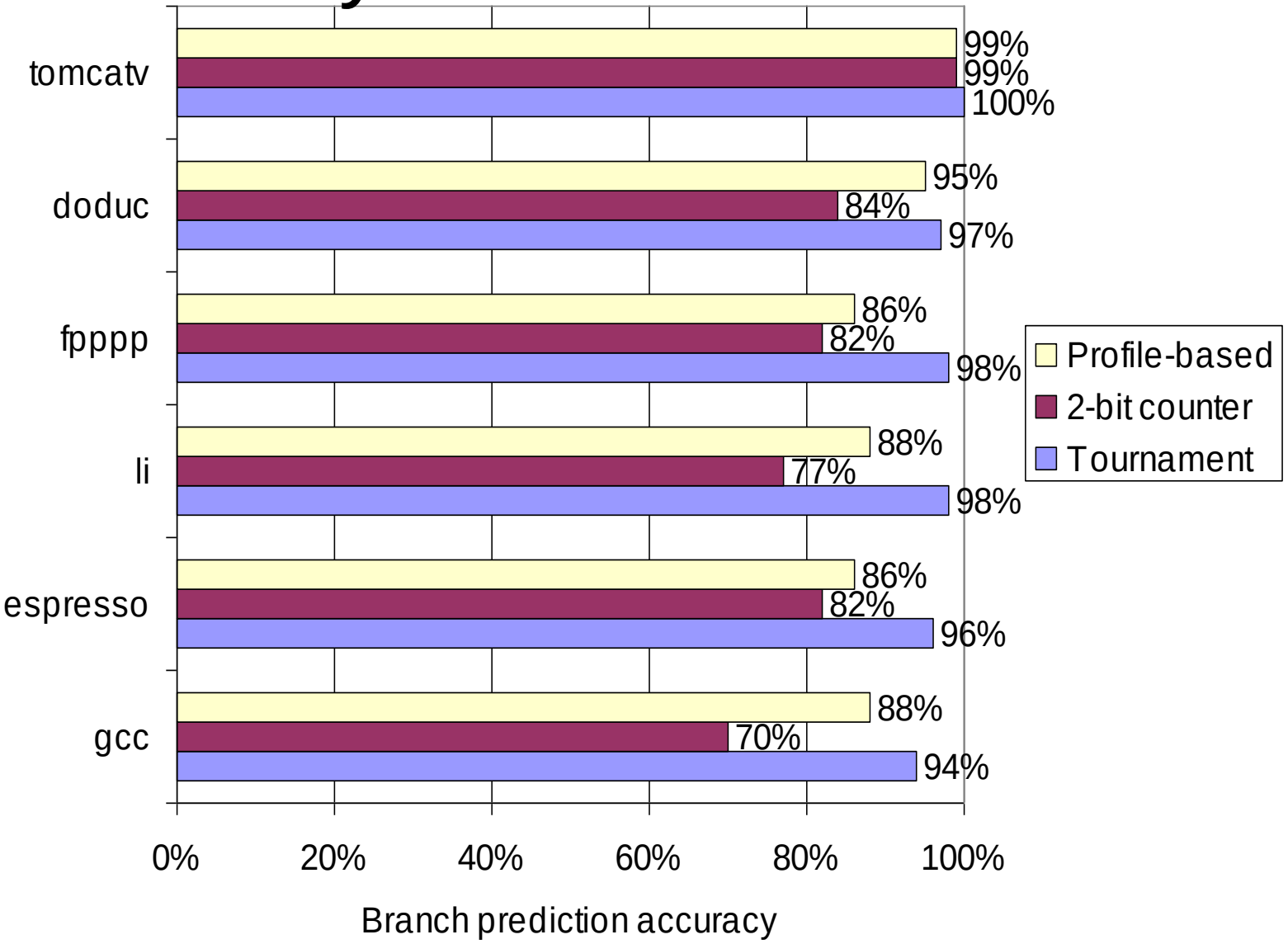
- Hopes to select the right predictor for the right branch

Tournament Predictor in Alpha 21264

- 4K 2-bit counters to choose from among a global predictor and a local predictor
- Global predictor also has 4K entries and is indexed by the history of the last 12 branches; each entry in the global predictor is a standard 2-bit predictor
  - 12-bit pattern: ith bit 0 => ith prior branch not taken; ith bit 1 => ith prior branch taken;
- Local predictor consists of a 2-level predictor:
  - Top level a local history table consisting of 1024 10-bit entries; each 10-bit entry corresponds to the most recent 10 branch outcomes for the entry. 10-bit history allows patterns 10 branches to be discovered and predicted.
  - Next level Selected entry from the local history table is used to index a table of 1K entries consisting a 3-bit saturating counters, which provide the local prediction
- Total size: 4K*2 + 4K*2 + 1K*10 + 1K*3 = 29K bits!

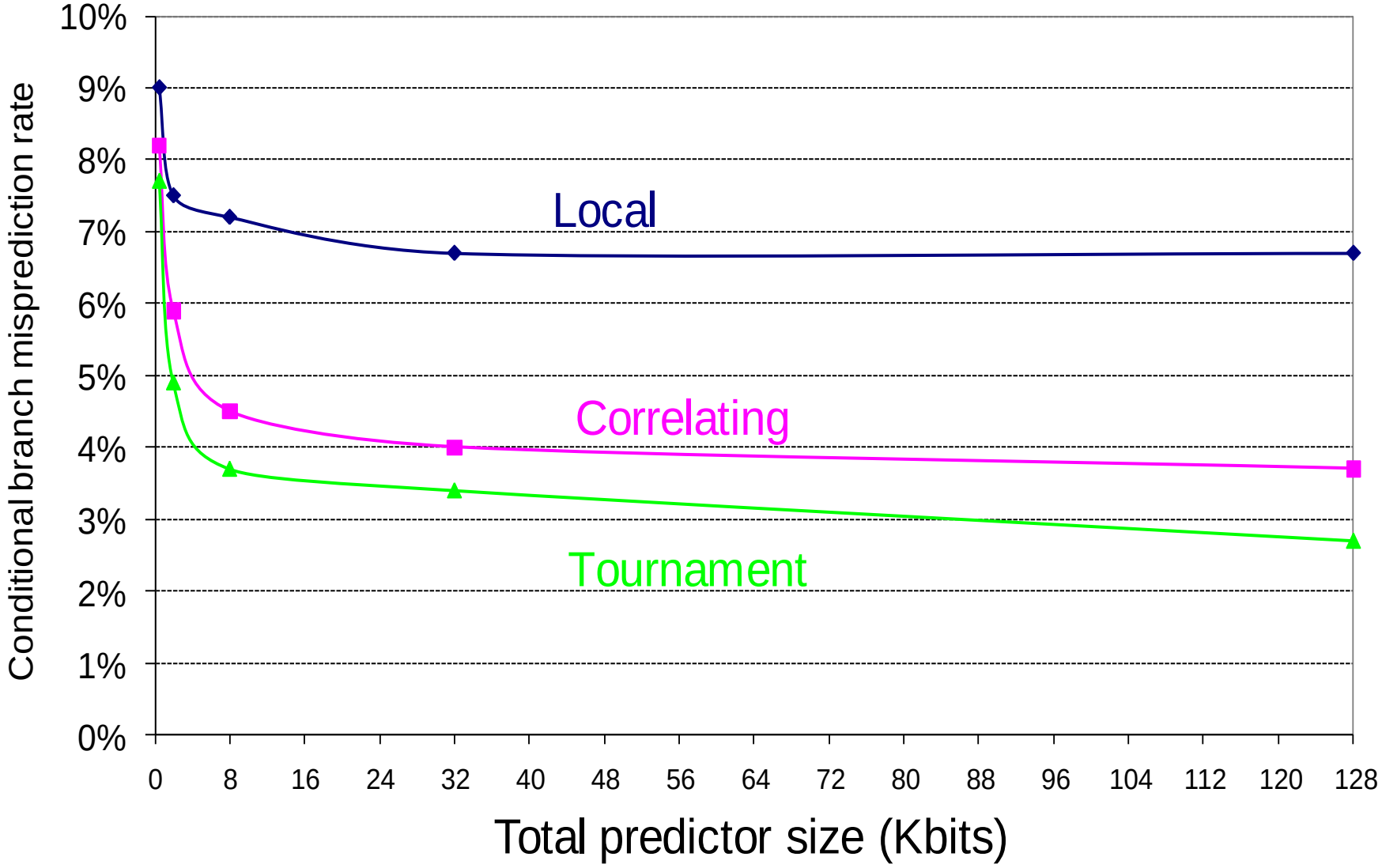
  (~180,000 transistors)

Accuracy of Branch Prediction



- Profile: branch profile from last execution
  (static in that the prediction is in encoded in the instruction, but derived from the real execution profile)
- A good dynamic predictor can outperform profile-driven static prediction by a large margin

Accuracy v. Size (SPEC89)



Tournament is not just a better predictor; it delivers a better prediction with fewer transistors
It's another example of combining two different optimisations, each good for different situations

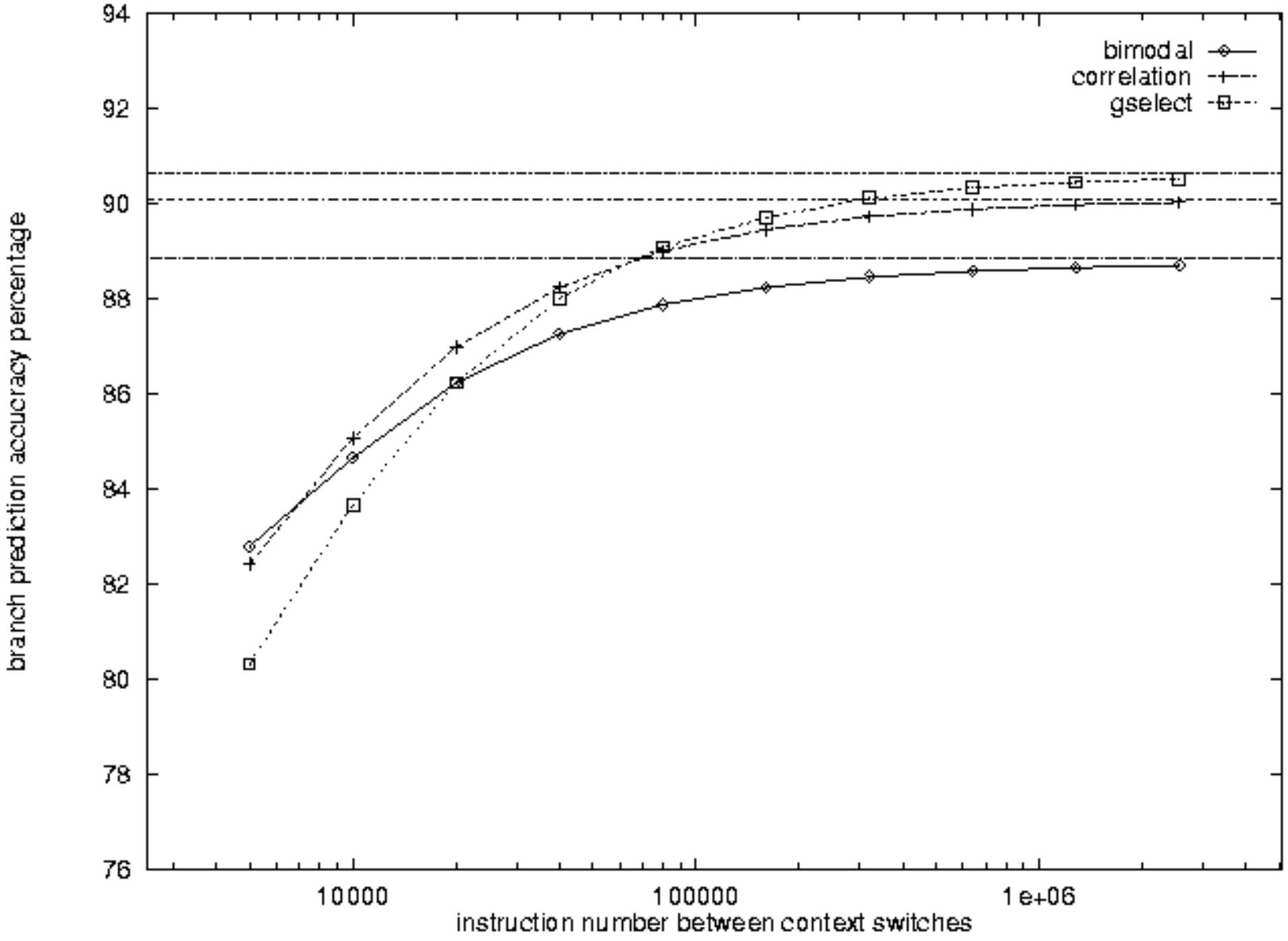
Special Case Return Addresses

- Register Indirect branch hard to predict address
- SPEC89 85% such branches for procedure return
- Since stack discipline for procedures, save return address in small buffer that acts like a stack: 8 to 16 entries has small miss rate

Warm-up effects and context-switching

- In real life, applications are interrupted and some other program runs for a while (if only the OS)
- This means the branch prediction is regularly trashed
- Simple predictors re-learn fast
  - in 2-bit bimodal predictor, all executions of given branch update the same 2 bits
- Sophisticated predictors re-learn more slowly
  - for example, in (2,2) gselect predictor, prediction updates are spread across 4 BHTs
- *Selective* predictor may choose fast learner predictor until better predictor warms up

Warm-up...



- Best predictor takes 20,000 instructions to overtake bimodal

Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)

Branch prediction and multi-issue

- In a processor that fetches, issues and dispatches multiple instructions per cycle…..
- What if we encounter two (or more) branches in one issue “packet”?

Branch prediction and multi-issue

- In a processor that fetches, issues and dispatches multiple instructions per cycle…..
- What if we encounter two (or more) branches in one issue “packet”?

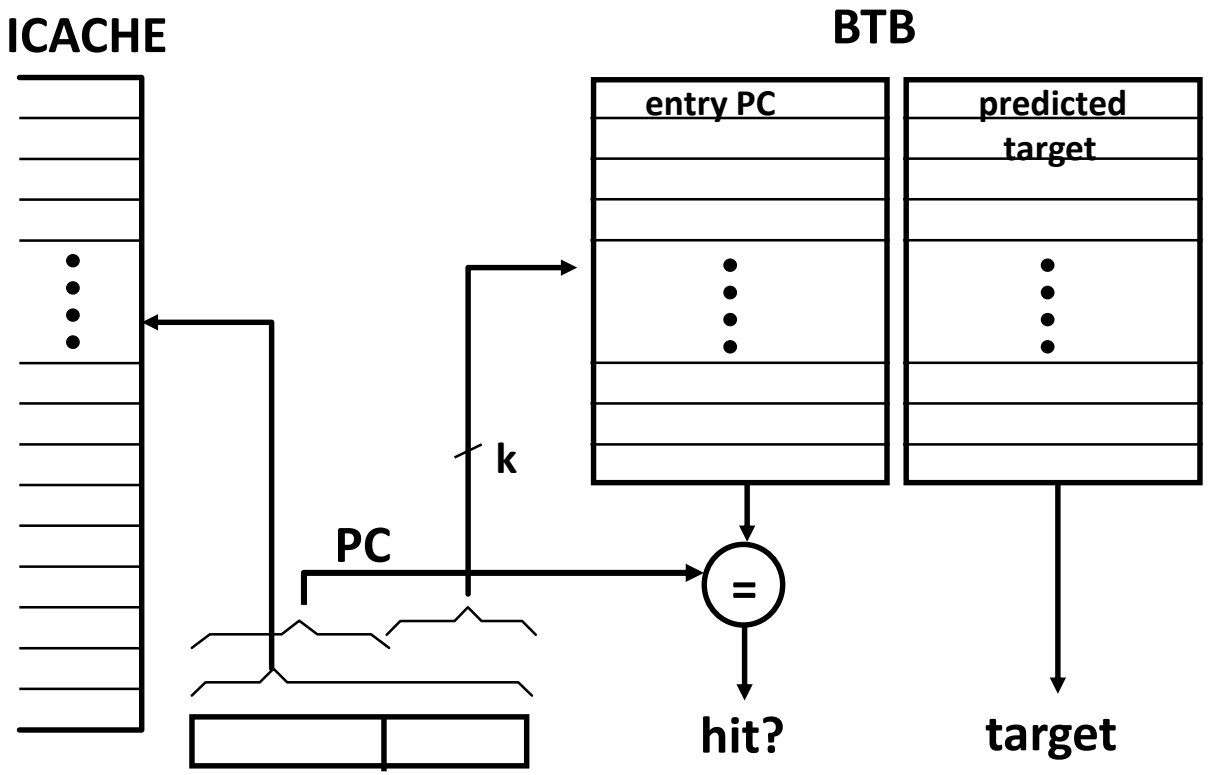
- Basically we need two (or more) predictions

Branch **target** prediction: BTBs

- re: "In order to predict a branch, it needs to be known that current instruction is branch instruction. "
- This doesn't have to be true!
- Suppose that when we fetch an instruction from location p. Now we can calculate the address of the next instruction (p+4 perhaps). We could fetch the next instruction from there. In fact in a pipelined processor we want to do that before we have decoded the instruction. And it might be a branch!
- So the idea of the Branch Target Buffer is this: in parallel with fetching from address p, we look in the BTB. The BTB is indexed with low-order PC bits, and tagged with the actual PC address. If, when we look in the BTB we find a tag match, we use the address from the BTB *instead of p+4* when fetching the next instruction, in the next cycle.
- If we don't get a tag match, we fetch from p+4 after all.
- When a *taken* branch is committed, we update the BTB with the branch's target address (and with the tag of the address of the branch instruction).
- It's easy to see how we could extend this, for example, to a two-way set associative BTB. More cunning ideas are possible, for example by using the BHR as part of the index.
- This is all about predicting the branch target address. Most of my lecture on branch prediction was about branch condition prediction. This can be combined with the BTB, to decide whether we should use the BTB prediction (even if its tag does match).
- Branch condition prediction (actually usually called simply "branch prediction") can also be used even if the taken/not-taken prediction arrives later - this is sometimes called a "resteer".

Branch Target Buffer (BTB)

- Cache of branch target addresses accessed in parallel with the I-cache in the fetch stage
- Updated only by taken branches (the direction-predictor determines whether BTB is used)
- If BTB hit and the instruction is a predicted-taken branch
  - target from the BTB is used as fetch address in the next cycle
- If BTB miss or the instruction is a predicted-not-taken branch
  - PC+N is used as the next fetch address in the next cycle



BTB is

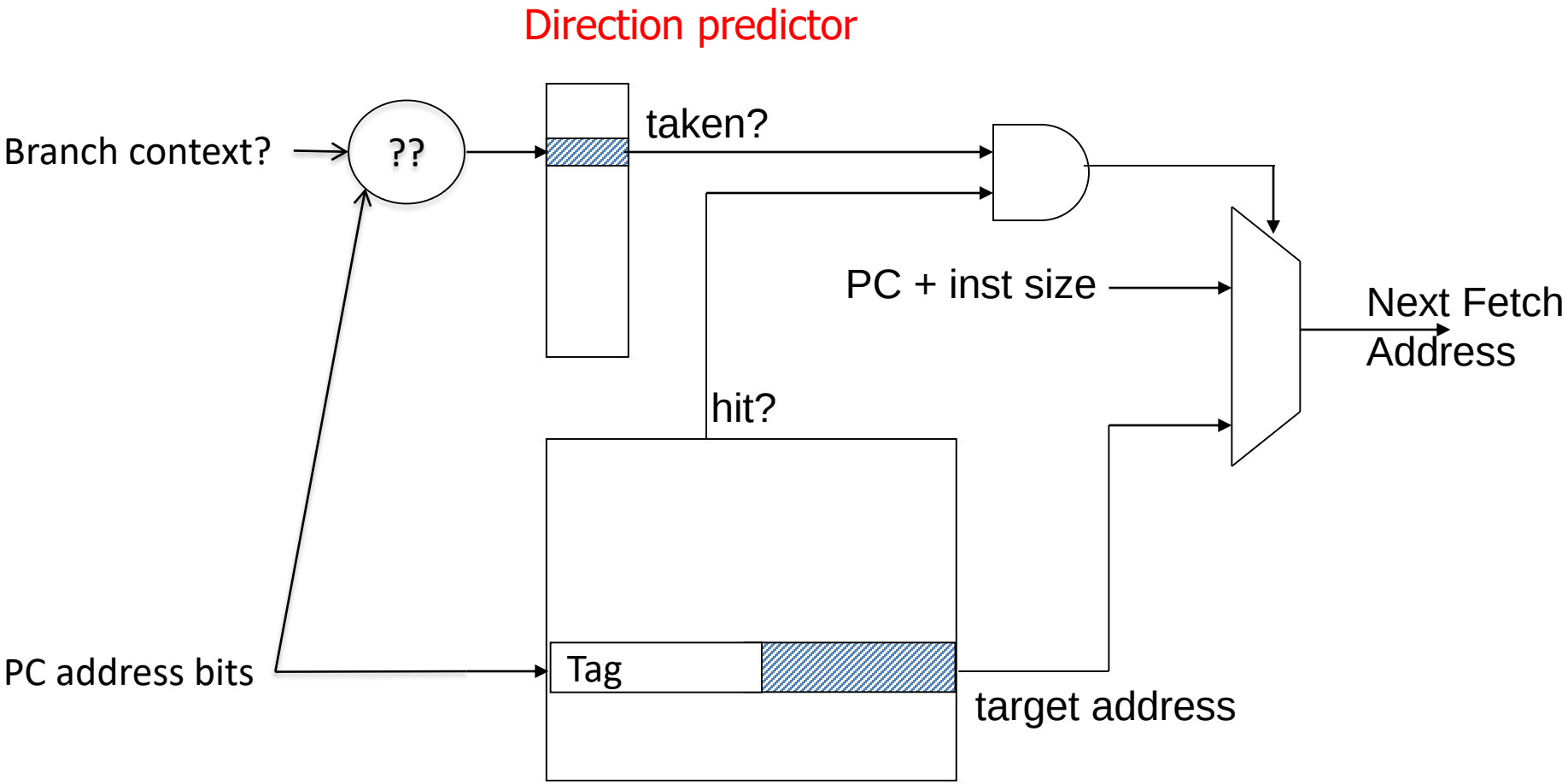
indexed with low-order PC address bits,

tagged with high-order bits

(Note: we could use an n-way set-associative design here)

Credit: Onur Mutlu

Combining BTB with direction Prediction

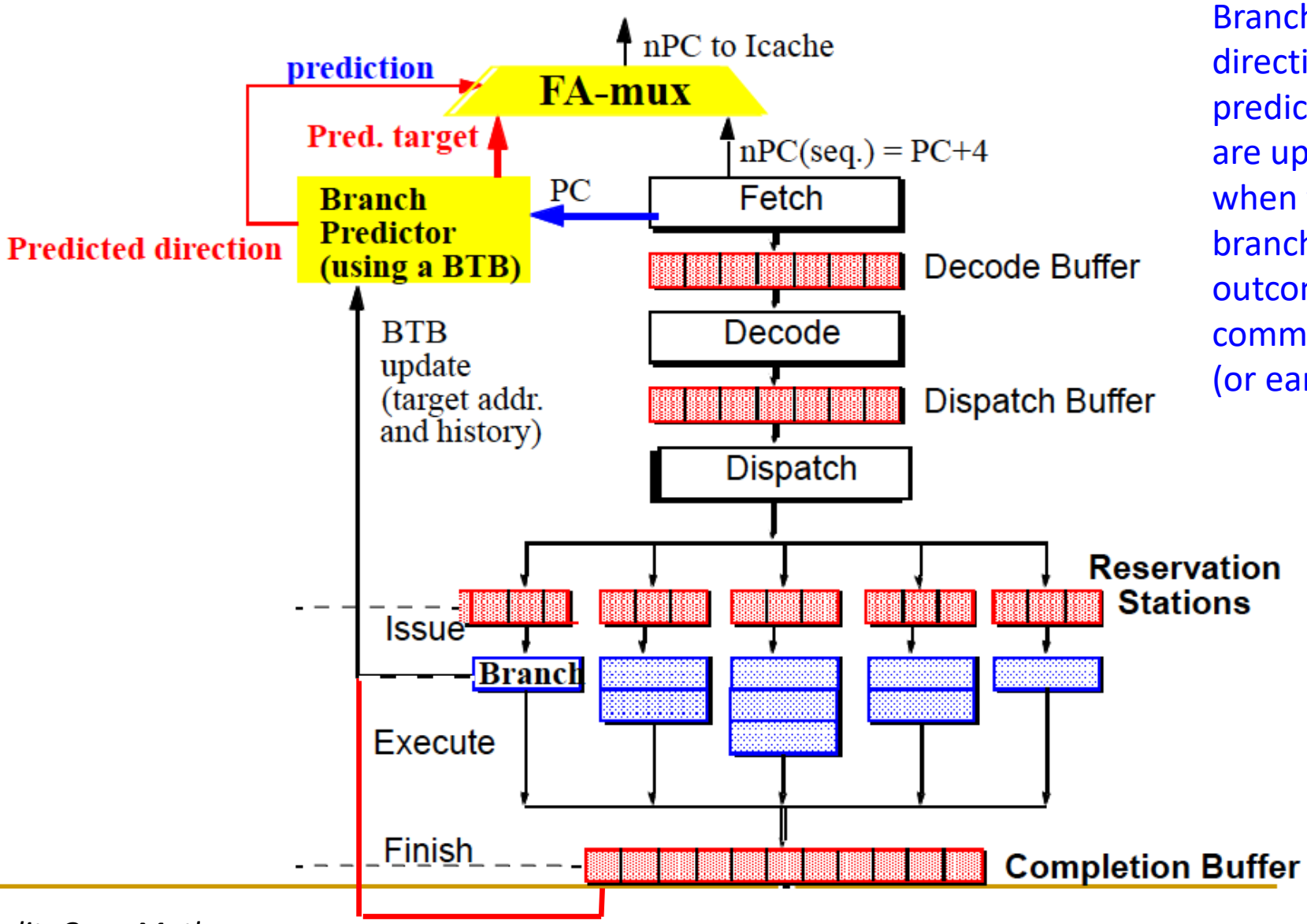


Cache of Target Addresses (BTB: Branch Target Buffer)

(What if branch is predicted-taken but BTB miss?)

Credit: Onur Mutlu

Updating the branch prediction



BTB and Branch direction prediction are updated when the branch outcome is committed (or earlier?)

Credit: Onur Mutlu

Dynamic Branch Prediction Summary

- Prediction becoming important part of scalar execution
- Branch History Table: 2 bits for loop accuracy
  - Saturating counter (bimodal) scheme handles highly-biased branches well
  - Some applications have highly dynamic branches
- Correlation: Recently executed branches correlated with next branch.
  - Either different branches
  - Or different executions of same branches
- Tournament Predictor: more resources to competitive solutions and pick between them
- Branch Target Buffer: include branch address & prediction
- Predicated Execution can reduce number of branches, number of mispredicted branches
- Return address stack for prediction of indirect jump

Branch prediction resources

- Design tradeoffs for the Alpha EV8 Conditional Branch Predictor (André Seznec, Stephen Felix, Venkata Krishnan, Yiannakis Sazeides)
  - SMT: 4 threads, wide-issue superscalar processor, 8-way issue, 512 registers (cancelled June 2001 when Alpha dropped)
  - Paper: http://citeseer.ist.psu.edu/seznec02design.html
  - Talk: http://ce.et.tudelft.nl/cecoll/slides/PresDelft0803.ppt
- Branch prediction in the Pentium family (Agner Fog)
  - Reverse engineering Pentium branch predictors using direct access to BTB
  - http://www.x86.org/articles/branch/branchprediction.htm
- Championship Branch Prediction Competition (CBP), organised by the Journal of Instruction-level Parallelism
  - http://www.jilp.org/cbp/
- **The CBP-1 winning entry: TAgged GEometric history length predictor (TAGE):** for each branch, maintain a predictor for what history length (from a geometric progression) works best.
  - http://www.irisa.fr/caps/people/seznec/JILP-COTTAGE.pdf

Example: Branch prediction in Intel Atom, Silvermont and Knights Landing

- two-level adaptive predictor with a global history table,
- Branch history register has 12 bits
- The pattern history table on the Atom has 4096 entries and is shared between threads
- The branch target buffer has 128 entries, organized as 4 ways by 32 sets
  - (size on Silvermont unknown, but probably bigger, and not shared between threads)
- Unconditional jumps make no entry in the global history table, but always-taken and nevertaken branches do
- Silvermont has branch prediction both at the fetch stage and at the later decode stage in the pipeline, where the latter can correct errors in the former
- No special predictor for loops (as there is for some other Intel CPUs)
  - Loops are predicted in the same way as other branches
- Penalty for mispredicting a branch is 11-13 clock cycles.
- It often occurs that a branch has a correct entry in the pattern history table, but no entry in the branch target buffer, which is much smaller:
  - If a branch is correctly predicted as taken, but no target can be predicted because of a missing BTB entry, then the penalty will be approximately 7 clock cycles.
- Pattern prediction evident for indirect branches on Knights Landing but not on Silvermont.
  - Indirect branches are predicted to go to the same target as last time on Silvermont
- Return stack buffer with 8 entries on the Atom and 16 entries on Silvermont and Knights Landing

"The microarchitecture of Intel, AMD and VIA CPUs An optimization guide for assembly programmers and compiler makers" http://www.agner.org/optimize/microarchitecture.pdf