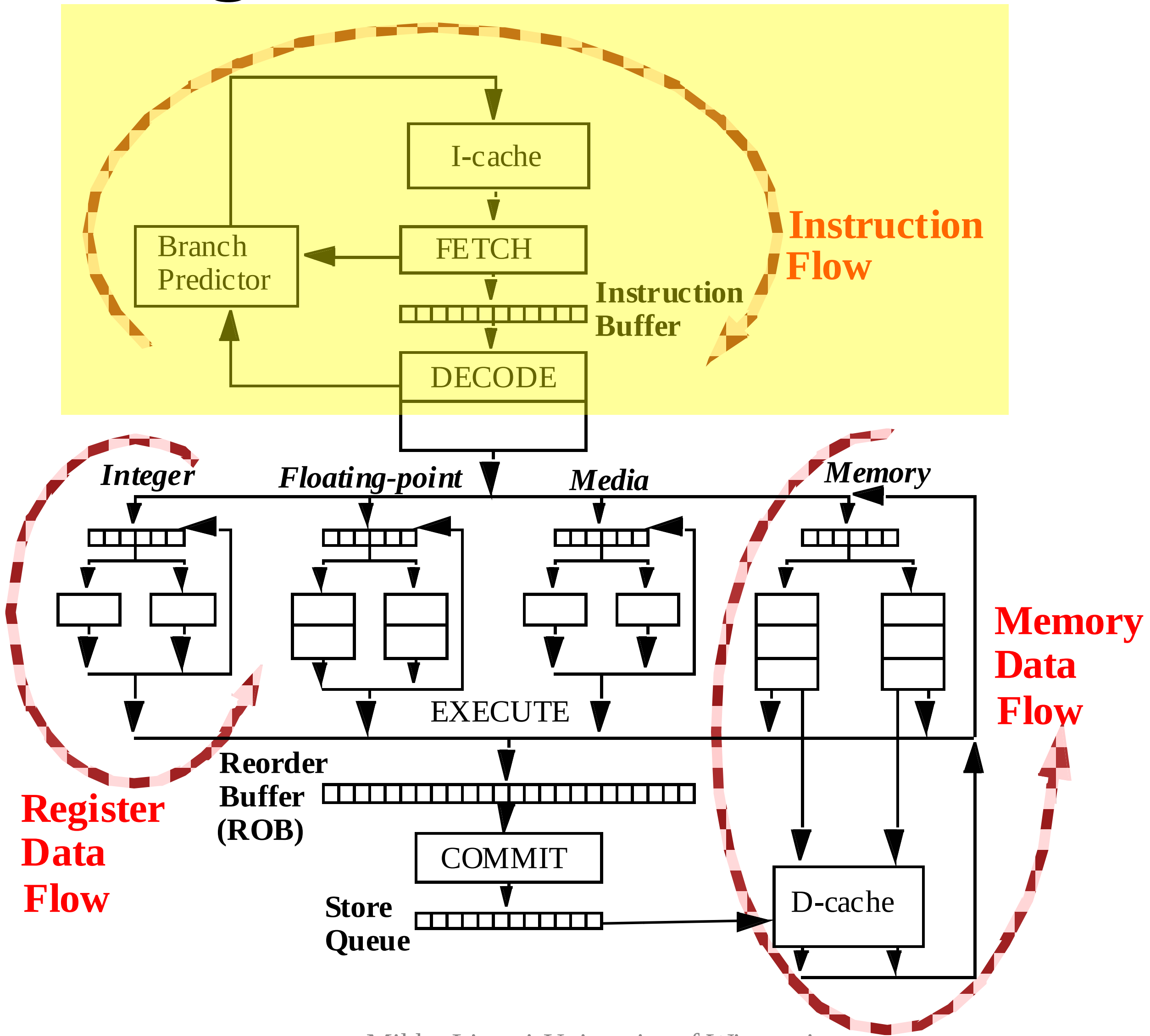
Instruction Flow Techniques

ECE/CS 752 Fall 2017

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

High-IPC Processor



Instruction Flow Techniques

- Instruction Flow and its Impediments
- Control Dependences
- Control Flow Speculation
  - Branch Speculation
  - Mis-speculation Recovery
- Branch Direction Prediction
  - Static Prediction
  - A brief history of dynamic branch prediction
- Branch Target Prediction
- High-bandwidth Fetch
- High-Frequency Fetch

Goal and Impediments

- Goal of Instruction Flow
  - Supply processor with maximum number of <u>useful</u> instructions every clock cycle
- Impediments
  - Branches and jumps
  - Finite I-Cache
    - Capacity
    - Bandwidth restrictions

Branch Types and Implementation

1. Types of Branches
   A. Conditional or Unconditional
   B. Save PC?
   C. How is target computed?
      - Single target (immediate, PC+immediate)
      - Multiple targets (register)
2. Branch Architectures
   A. Condition code or condition registers
   B. Register

What’s So Bad About Branches?

- Effects of Branches
  - Fragmentation of I-Cache lines
  - Need to determine branch direction
  - Need to determine branch target
  - Use up execution resources
    - Pipeline drain/fill

What's So Bad About Branches?

Problem: Fetch stalls until direction is determined

Solutions:

- Minimize delay
  - Move instructions determining branch condition away from branch (CC architecture)
- Make use of delay
  - Non-speculative:
    - Fill delay slots with useful safe instructions
    - Execute both paths (eager execution)
  - Speculative:
    - Predict branch direction

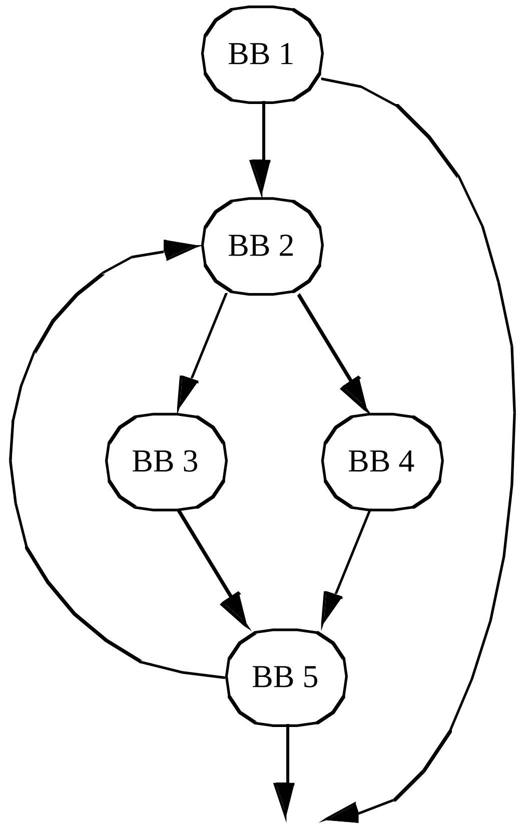
What's So Bad About Branches?

Problem: Fetch stalls until branch target is determined

Solutions:

- <u>Minimize delay</u>
  - Generate branch target early
- <u>Make use of delay</u>: Predict branch target
  - Single target
  - Multiple targets

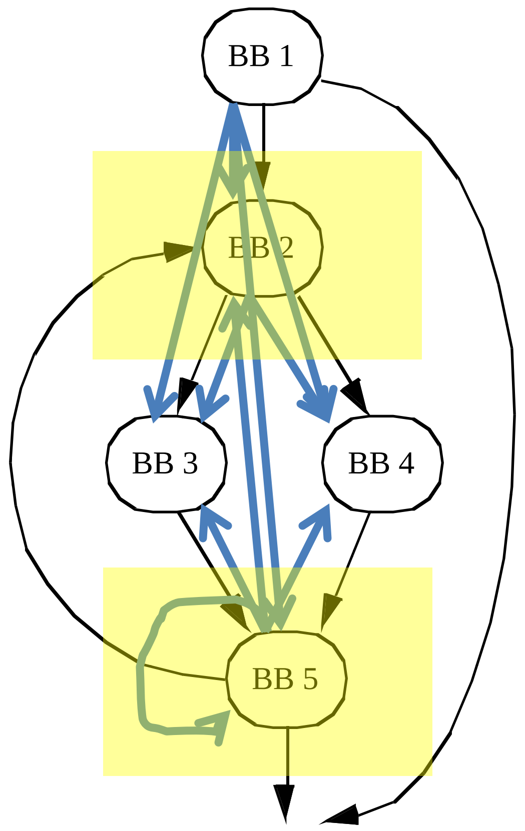
Control Dependences



```
main:
        addi r2, r0, A      |
        addi r3, r0, B      |
        addi r4, r0, C      |  BB 1
        addi r5, r0, N      |
        add  r10,r0, r0     |
        bge  r10,r5, end    |
loop:
        lw   r20, 0(r2)     |
        lw   r21, 0(r3)     |  BB 2
        bge  r20,r21,T1     |
        sw   r21, 0(r4)     |  BB 3
        b    T2             |
T1:
        sw   r20, 0(r4)     |  BB 4
T2:
        addi r10,r10,1      |
        addi r2, r2, 4      |
        addi r3, r3, 4      |  BB 5
        addi r4, r4, 4      |
        blt  r10,r5, loop   |
end:
```

- Control Flow Graph
  - Shows possible paths of control flow through basic blocks

Control Dependences



```
main:
        addi r2,  r0,  A          BB 1
        addi r3,  r0,  B
        addi r4,  r0,  C
        addi r5,  r0,  N
        add  r10, r0,  r0
        bge  r10, r5,  end
loop:
        lw   r20, 0(r2)           BB 2
        lw   r21, 0(r3)
        bge  r20, r21, T1
        sw   r21, 0(r4)           BB 3
        b    T2
T1:
        sw   r20, 0(r4)           BB 4
T2:
        addi r10, r10, 1          BB 5
        addi r2,  r2,  4
        addi r3,  r3,  4
        addi r4,  r4,  4
        blt  r10, r5,  loop
end:
```

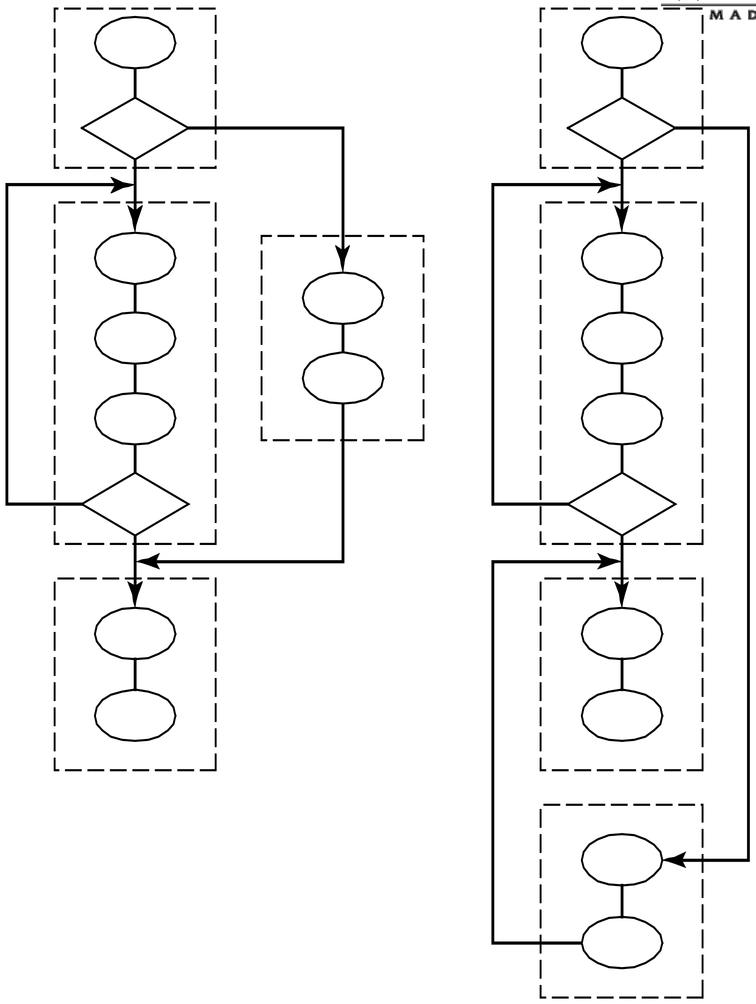
- Control Dependence
  - Node B is CD on Node A if A determines whether B executes
  - If path 1 from A to exit includes B, and path 2 does not, then B is control-dependent on A

Program Control Flow

- Implicit Sequential Control Flow
  - Static Program Representation
    - Control Flow Graph (CFG)
    - Nodes = basic blocks
    - Edges = Control flow transfers
  - Physical Program Layout
    - Mapping of CFG to linear program memory
    - Implied sequential control flow
  - Dynamic Program Execution
    - Traversal of the CFG nodes and edges (e.g. loops)
    - Traversal dictated by branch conditions
  - Dynamic Control Flow
    - Deviates from sequential control flow
    - Disrupts sequential fetching
    - Can stall IF stage and reduce I-fetch bandwidth

Program Control Flow

- Dynamic traversal of static CFG
- Mapping CFG to linear memory


(a) (b)

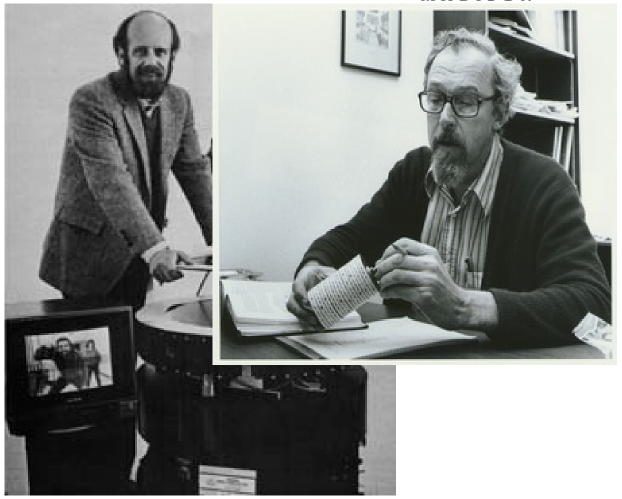
Limits on Instruction Level Parallelism (ILP)

1970: Flynn

Weiss and Smith [1984]	1.58
Sohi and Vajapeyam [1987]	1.81
Tjaden and Flynn [1970]	1.86 (Flynn's bottleneck)
Tjaden and Flynn [1973]	1.96
Uht [1986]	2.00
Smith et al. [1989]	2.00
Jouppi and Wall [1988]	2.40
Johnson [1991]	2.50
Acosta et al. [1986]	2.79
Wedig [1982]	3.00
Butler et al. [1991]	5.8
Melvin and Patt [1991]	6
Wall [1991]	7 (Jouppi disagreed)
Kuck et al. [1972]	8
Riseman and Foster [1972]	51 (no control dependences)
Nicolau and Fisher [1984]	90 (Fisher's optimism)

Riseman and Foster's Study

1970: Flynn
1972: Riseman/Foster



- 7 benchmark programs on CDC-3600
- Assume infinite machines
  - Infinite memory and instruction stack
  - Infinite register file
  - Infinite functional units
  - True dependencies only at dataflow limit
- If bounded to single basic block, speedup is 1.72 (Flynn's bottleneck)
- If one can bypass n branches (hypothetically), then:

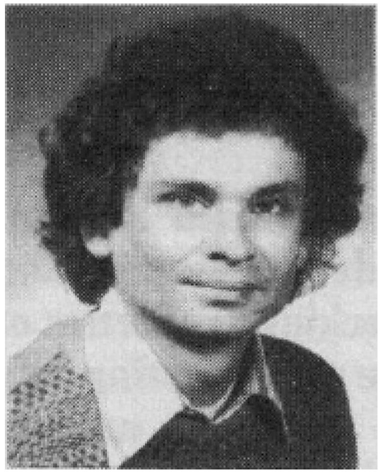
Branches Bypassed	0	1	2	8	32	128	∞
Speedup	1.72						

Branch Prediction

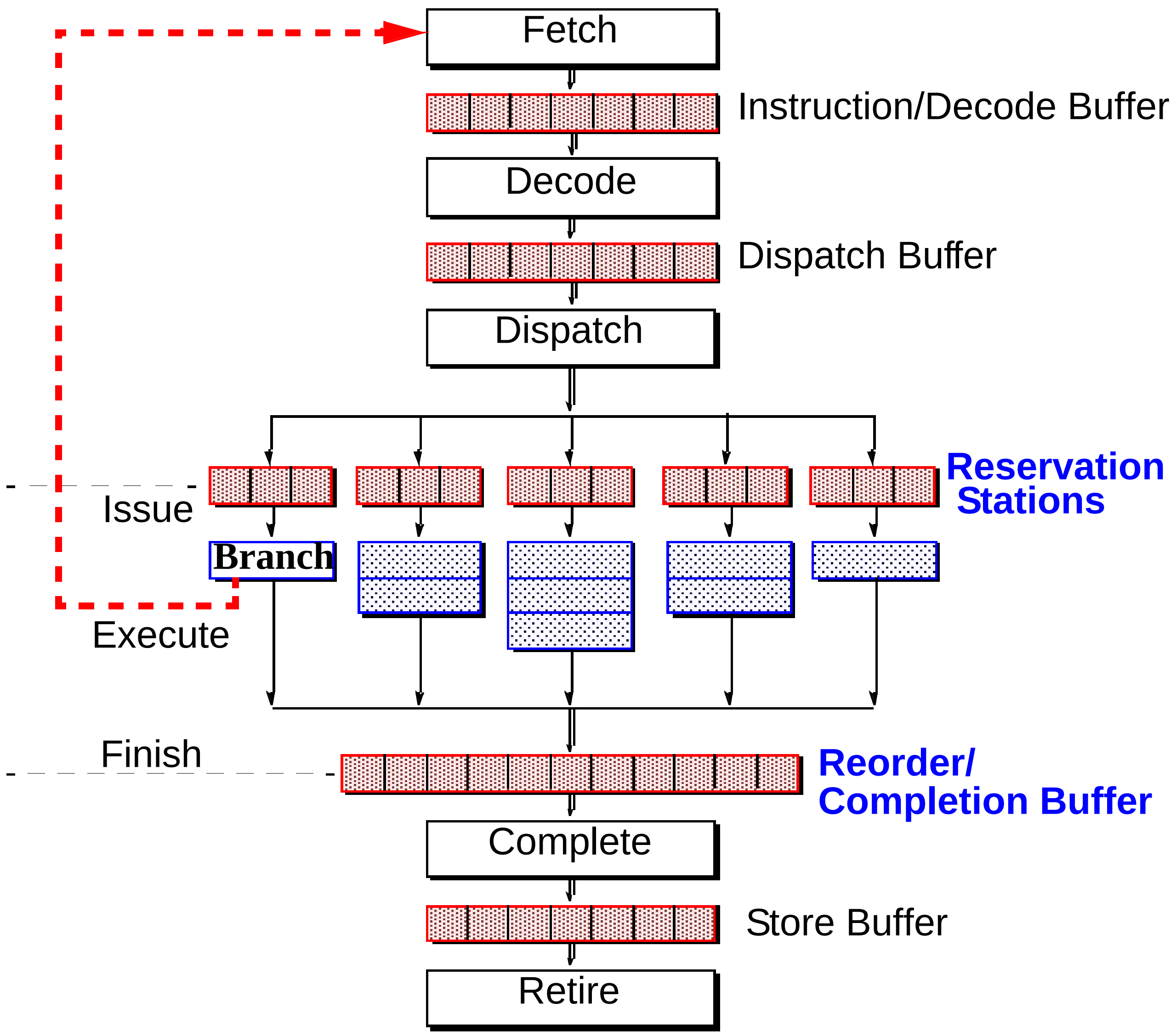
1970: Flynn
1972: Riseman/Foster

1979: Smith Predictor

- Riseman & Foster showed potential
  - But no idea how to reap benefit
- 1979: Jim Smith patents branch prediction at Control Data
  - Predict current branch based on past history
- Today: virtually all processors use branch prediction



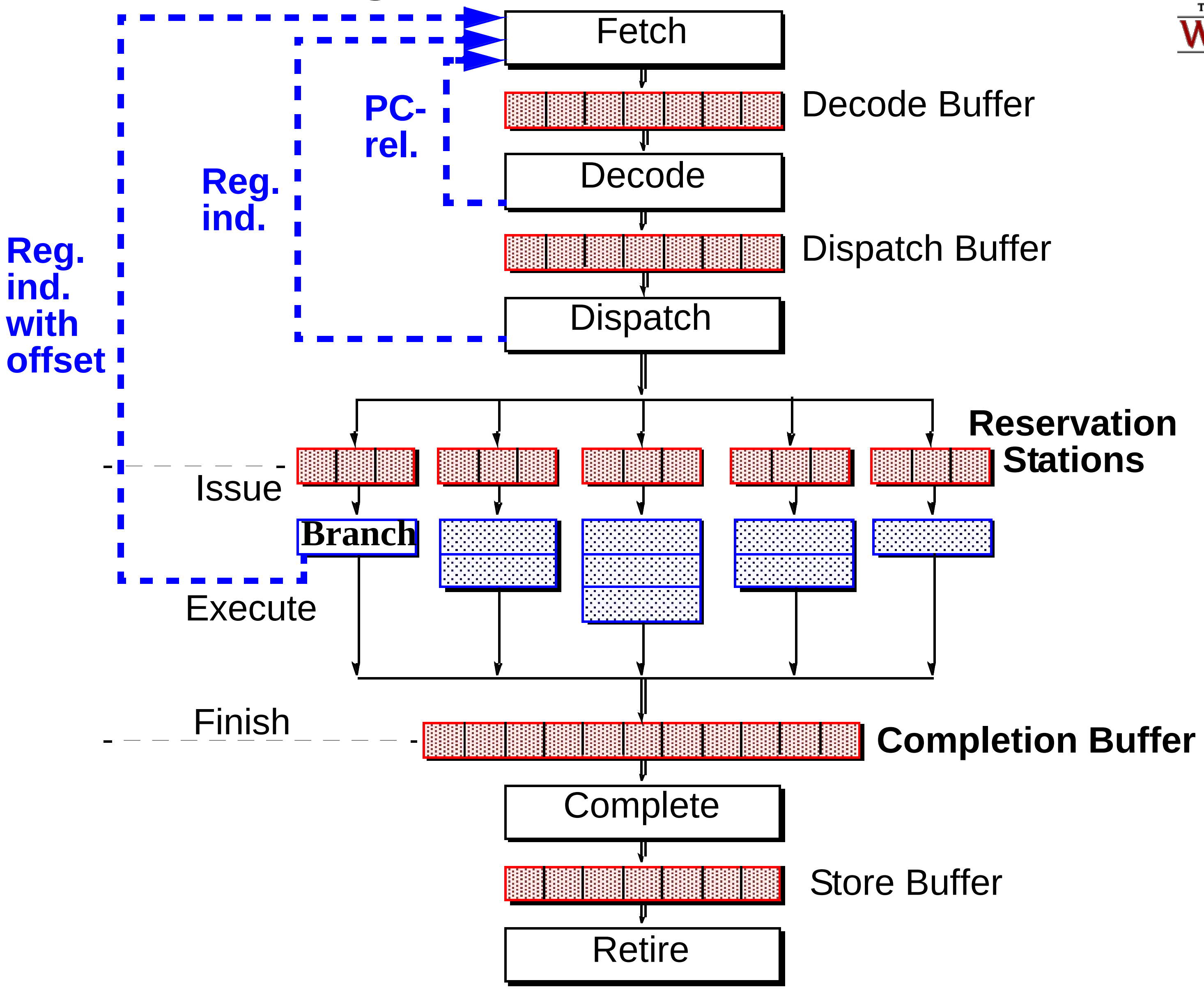
Disruption of Sequential Control Flow



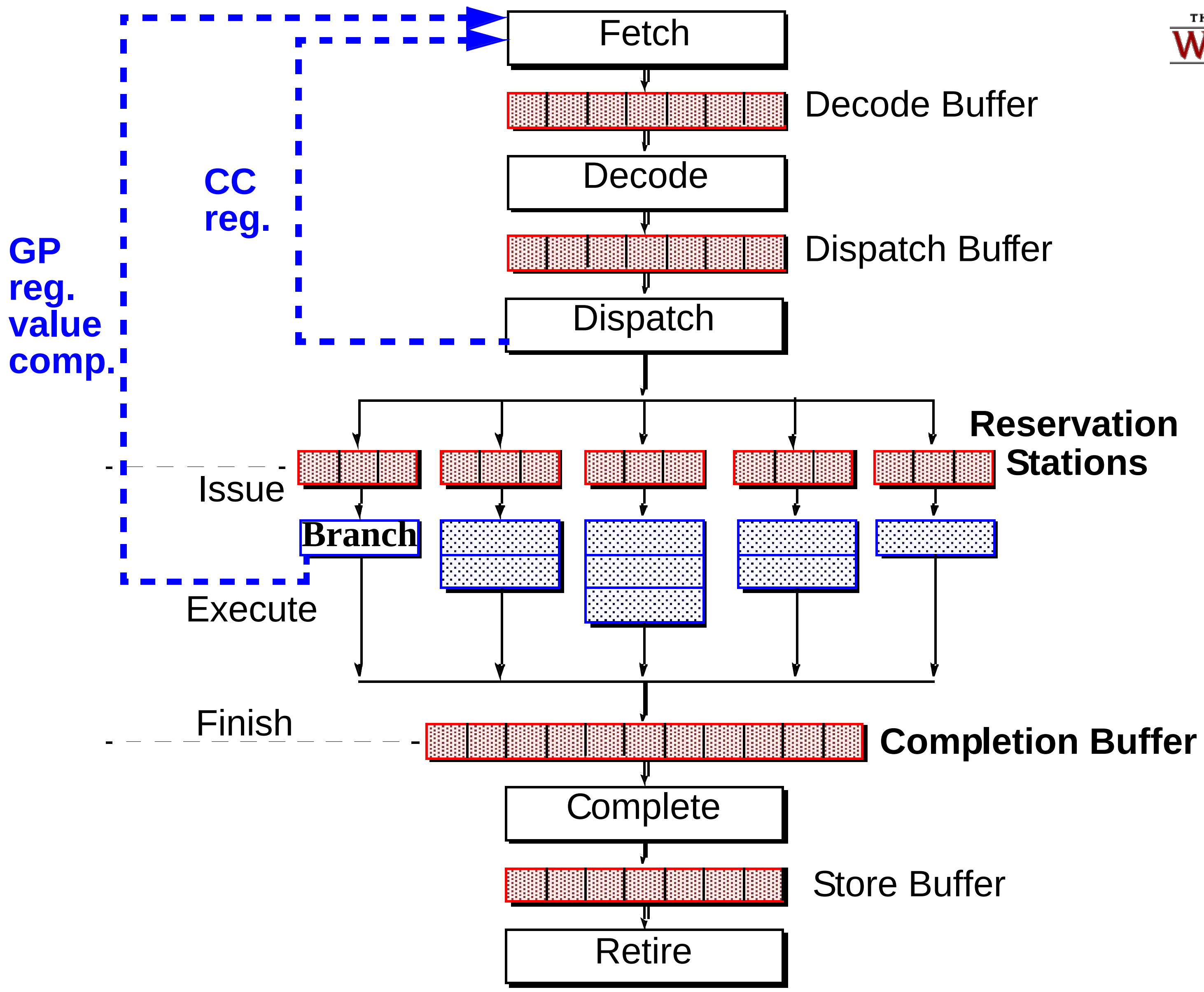
Branch Prediction

- Target address generation → <u>Target Speculation</u>
  - Access register:
    - PC, General purpose register, Link register
  - Perform calculation:
    - +/- offset, autoincrement, autodecrement
- Condition resolution → <u>Condition speculation</u>
  - Access register:
    - Condition code register, General purpose register
  - Perform calculation:
    - Comparison of data register(s)

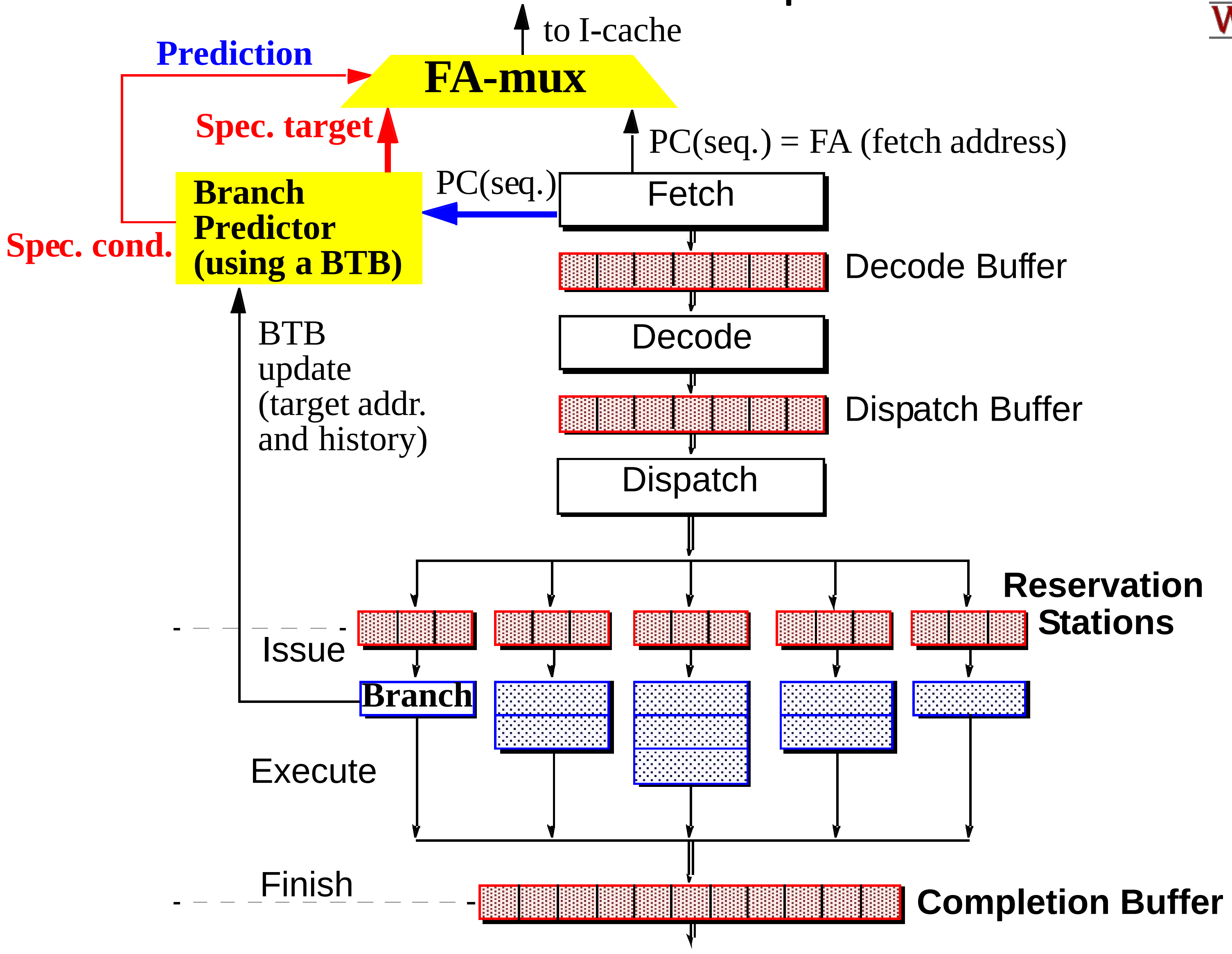
Target Address Generation



Condition Resolution



Branch Instruction Speculation



Branch/Jump Target Prediction

Branch inst. address	Information for predict.	**Branch target address** (most recent)
0x0348	0101 (NTNT)	0x0612

- Branch Target Buffer: small cache in fetch stage
  - Previously executed branches, address, taken history, target(s)
- Fetch stage compares current FA against BTB
  - If match, use prediction
  - If predict taken, use BTB target
- When branch executes, BTB is updated
- Optimization:
  - Size of BTB: increases hit rate
  - Prediction algorithm: increase accuracy of prediction

Branch Prediction: Condition Speculation

1. Biased Not Taken
   - Hardware prediction
   - Does not affect ISA
   - Not effective for loops
2. Software Prediction
   - Extra bit in each branch instruction
     - Set to 0 for not taken
     - Set to 1 for taken
   - Bit set by compiler or user; can use profiling
   - Static prediction, same behavior every time
3. Prediction based on branch offset
   - Positive offset: predict not taken
   - Negative offset: predict taken
4. Prediction based on dynamic history

UCB Study [Lee and Smith, 1984]

- Benchmarks used
  - 26 programs (IBM 370, DEC PDP-11, CDC 6400)
  - 6 workloads (4 IBM, 1 DEC, 1 CDC)
  - Used trace-driven simulation
- Branch types
  - Unconditional: always taken or always not taken
  - Subroutine call: always taken
  - Loop control: usually taken
  - Decision: either way, if-then-else
  - Computed goto: always taken, with changing target
  - Supervisor call: always taken
  - Execute: always taken (IBM 370)

IBM1: compiler
IBM2: cobol (business app)
IBM3: scientific
IBM4: supervisor (OS)

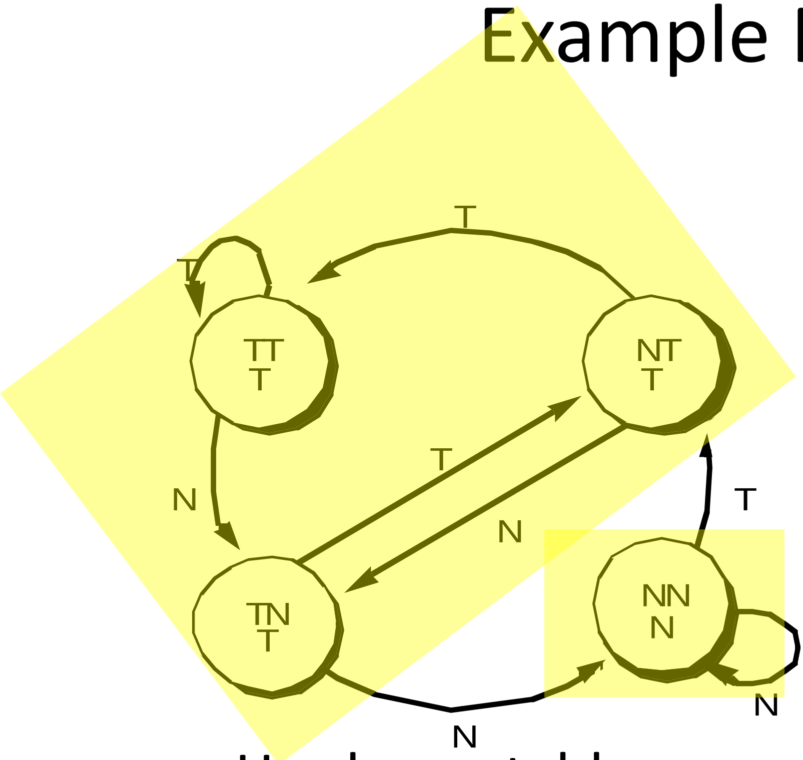
	IBM1	IBM2	IBM3	IBM4	DEC	CDC	Avg
T	0.640	0.657	0.704	0.540	0.738	0.778	0.676
NT	0.360	0.343	0.296	0.460	0.262	0.222	0.324

Branch Prediction Function

- Prediction function F(X1, X2, …)
  - X1 – opcode type
  - X2 – history
- Prediction effectiveness based on opcode only, or history

	IBM1	IBM2	IBM3	IBM4	DEC	CDC
Opcode only	66	69	71	55	80	78
History 0	64	64	70	54	74	78
History 1	92	95	87	80	97	82
History 2	93	97	91	83	98	91
History 3	94	97	91	84	98	94
History 4	95	97	92	84	98	95
History 5	95	97	92	84	98	96

Example Prediction Algorithm



Branch inst. address	**Information for predict.**	Branch target address

- Hardware table remembers last 2 branch outcomes
  - History of past several branches encoded by FSM
  - Current state used to generate prediction
- Results:

Workload	IBM1	IBM2	IBM3	IBM4	DEC	CDC
Accuracy	93	97	91	83	98	91

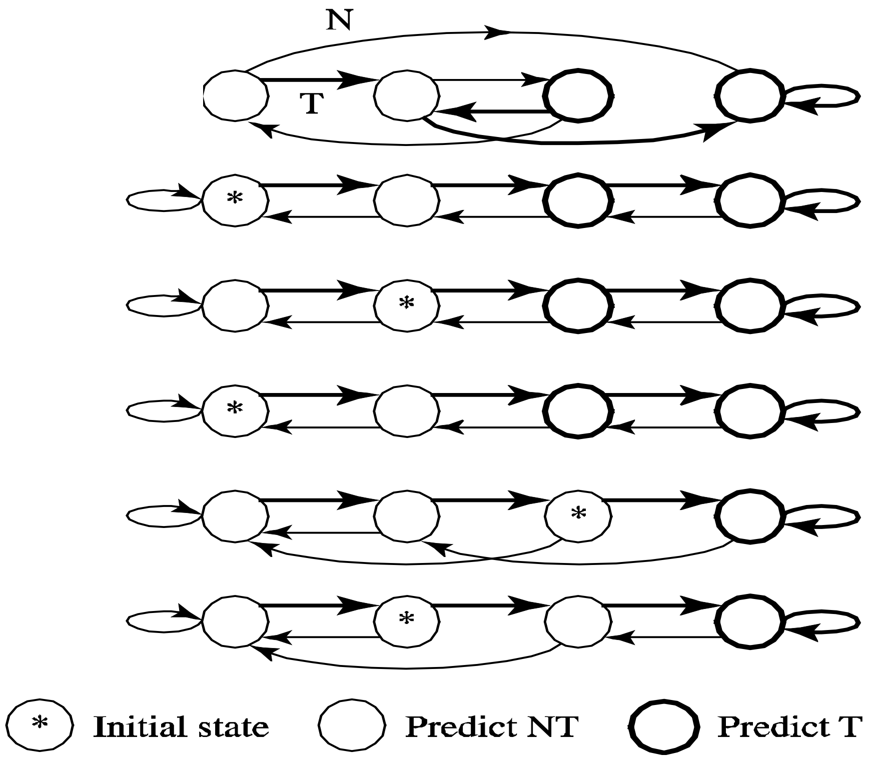
IBM Study [Nair, 1992]

- Branch processing on the IBM RS/6000
  - Separate branch functional unit
  - Overlap of branch instructions with other instructions
    - Zero cycle branches
  - Two causes for branch stalls
    - Unresolved conditions
    - Branches downstream too close to unresolved branches
- Investigated optimal FSM design for branch prediction

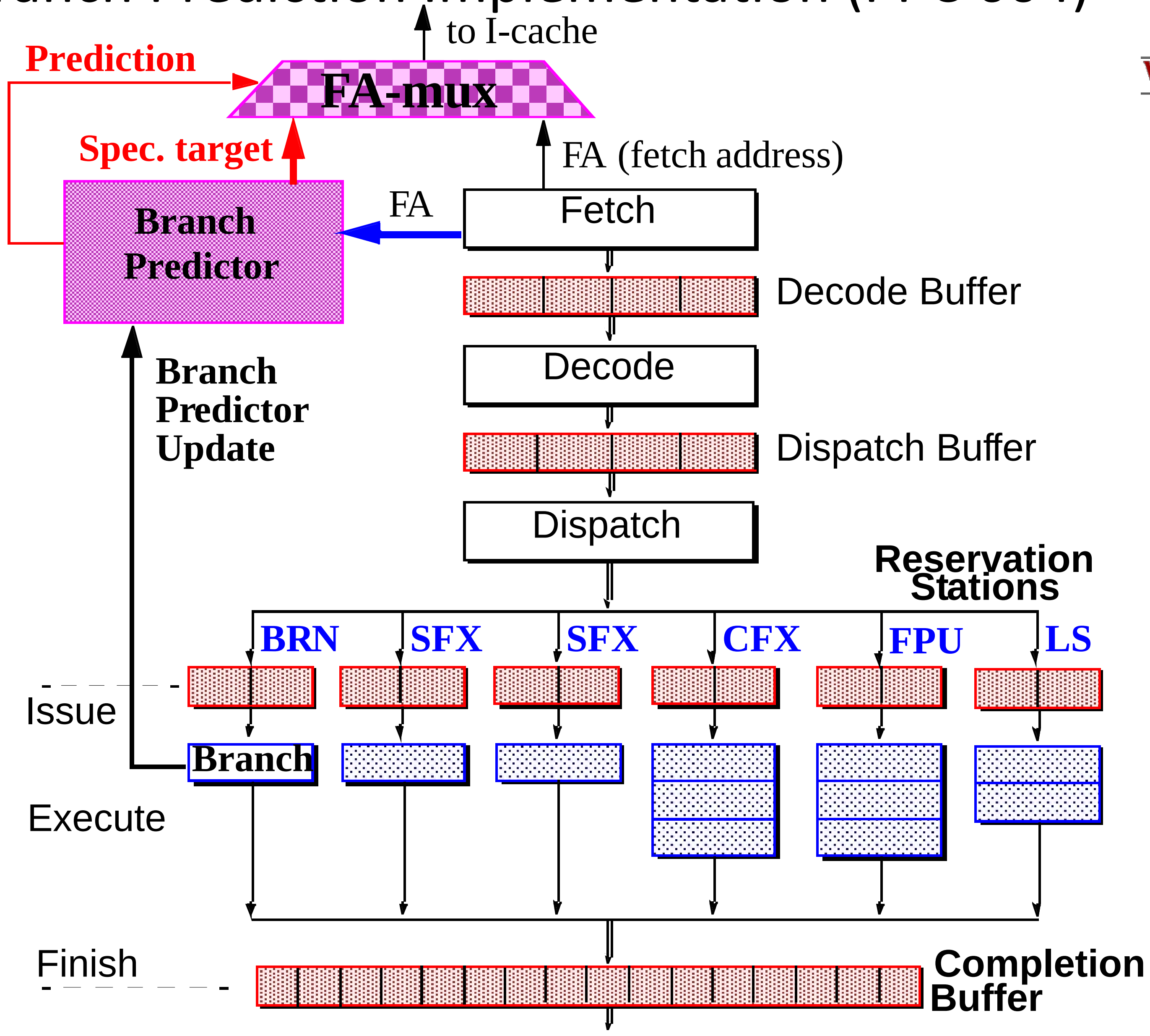
Exhaustive Search for Optimal 2-bit Predictor

- There are 2^{20} possible state machines of 2-bit predictors
  - Some machines are uninteresting, pruning them out reduces the number of state machines to 5248
- For each benchmark, determine prediction accuracy for all the predictor state machines
  - Find optimal 2-bit predictor for each application

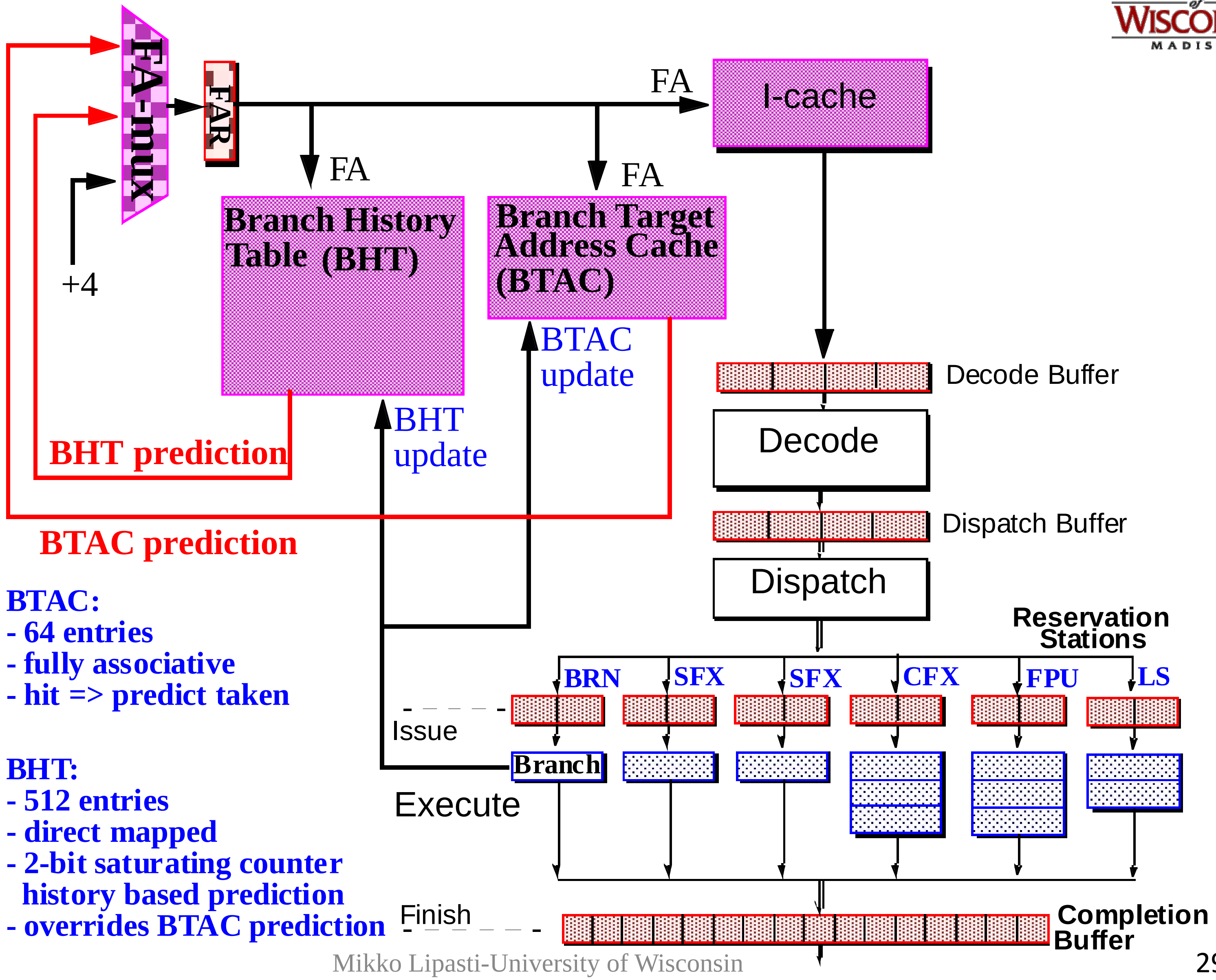
Benchmark	Optimal
spice2g6	97.2
doduc	94.3
gcc	89.1
espresso	89.1
li	87.1
eqntott	87.9



Branch Prediction Implementation (PPC 604)



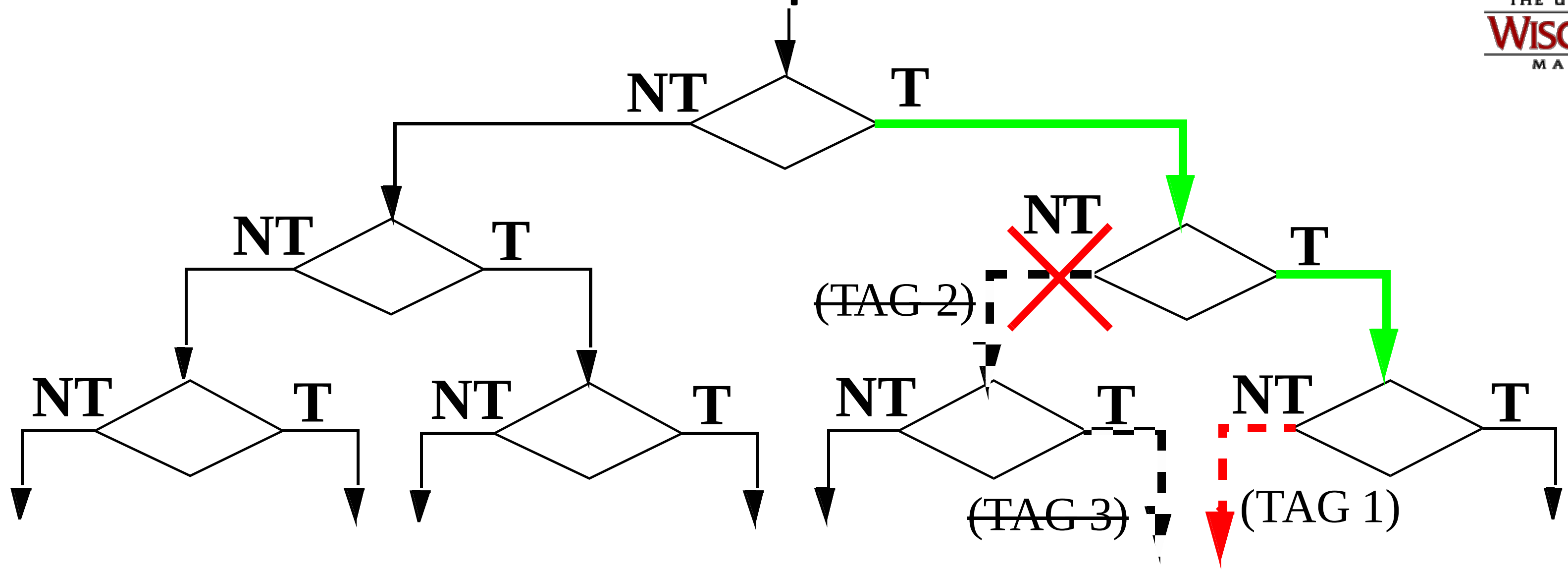
BTAC and BHT Design (PPC 604)



BTAC:
- 64 entries
- fully associative
- hit => predict taken

BHT:
- 512 entries
- direct mapped
- 2-bit saturating counter history based prediction
- overrides BTAC prediction

Branch Speculation



- Start new correct path
  - Must remember the alternate (non-predicted) path
- Eliminate incorrect path
  - Must ensure that the mis-speculated instructions produce no side effects

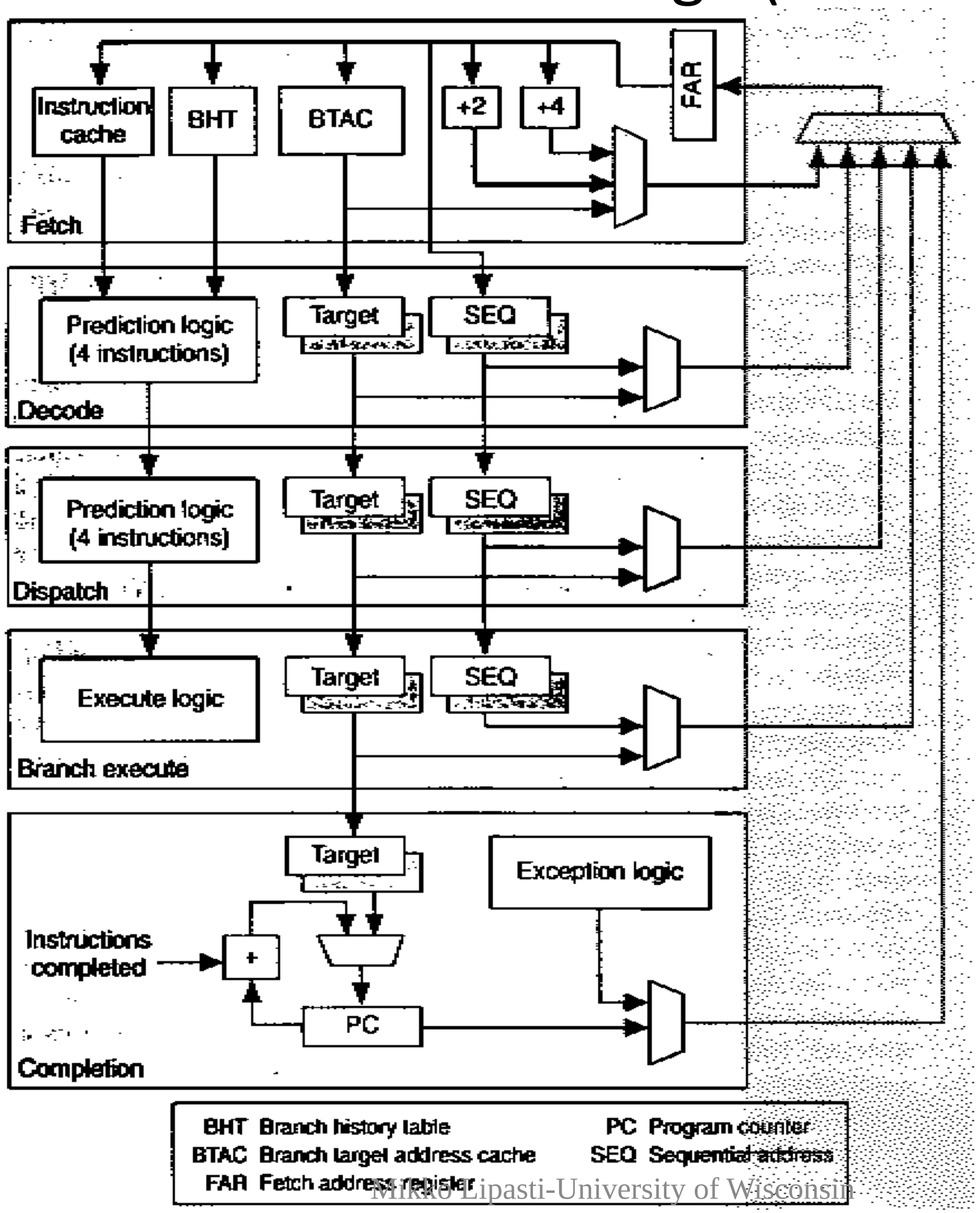
Mis-speculation Recovery

- Start new correct path
  1. Update PC with computed branch target (if predicted NT)
  2. Update PC with sequential instruction address (if predicted T)
  3. Can begin speculation again at next branch
- Eliminate incorrect path
  1. Use tag(s) to deallocate ROB entries occupied by speculative instructions
  2. Invalidate all instructions in the decode and dispatch buffers, as well as those in reservation stations

Tracking Instructions

- Assign branch tags
  - Allocated in circular order
  - Instruction carries this tag throughout processor
- Often: track instruction groups
  - Instructions managed in groups, max. one branch per group
  - ROB structured as groups
    - Leads to some inefficiency
    - Simpler tracking of speculative instructions

BTAC and BHT Design (PPC 604)



Fairly simple, 5-stage machine from 1994

Many sources for PC redirect

Lots of complexity

Instruction Flow Techniques

- Instruction Flow and its Impediments
- Control Dependences
- Control Flow Speculation
  - Branch Speculation
  - Mis-speculation Recovery
- Branch Direction Prediction
  - Static Prediction
  - A brief history of dynamic branch prediction
- Branch Target Prediction
- High-bandwidth Fetch
- High-Frequency Fetch

Static Branch Prediction

- Single-direction
  - Always not-taken: Intel i486
- Backwards Taken/Forward Not Taken
  - Loop-closing branches
  - Used as backup in Pentium Pro, II, III, 4
- Heuristic-based:

```
void * p = malloc (numBytes);
if (p == NULL)
        errorHandlingFunction( );
```

Static Branch Prediction

Heuristic Name	Description
Loop Branch	If the branch target is back to the head of a loop, predict taken.
Pointer	If a branch compares a pointer with NULL, or if two pointers are compared, predict in the direction that corresponds to the pointer being not NULL, or the two pointers not being equal.
Opcode	If a branch is testing that an integer is less than zero, less than or equal to zero, or equal to a constant, predict in the direction that corresponds to the test evaluating to false.
Guard	If the operand of the branch instruction is a register that gets used before being redefined in the successor block, predict that the branch goes to the successor block.
Loop Exit	If a branch occurs inside a loop, and neither of the targets is the loop head, then predict that the branch does not go to the successor that is the loop exit.
Loop Header	Predict that the successor block of a branch that is a loop header or a loop pre-header is taken.
Call	If a successor block contains a subroutine call, predict that the branch goes to that successor block.
Store	If a successor block contains a store instruction, predict that the branch does not go to that successor block.
Return	If a successor block contains a return from subroutine instruction, predict that the branch does not go to that successor block.

- Heuristic-based: Ball/Larus
  - Thomas Ball and James R. Larus. Branch Prediction for Free. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 300-313, May 1993.

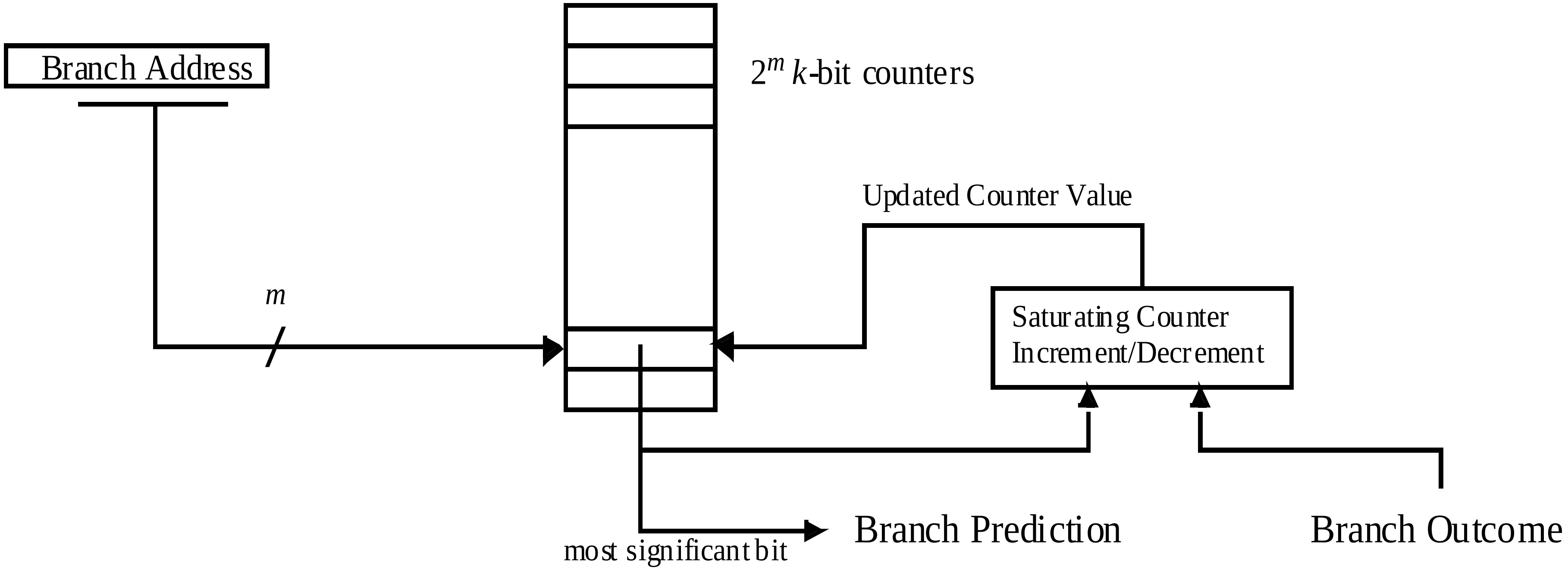
Static Branch Prediction

- Profile-based
  1. Instrument program binary
  2. Run with representative (?) input set
  3. Recompile program
     a. Annotate branches with hint bits, or
     b. Restructure code to match predict not-taken
- Best performance: 75-80% accuracy

Dynamic Branch Prediction

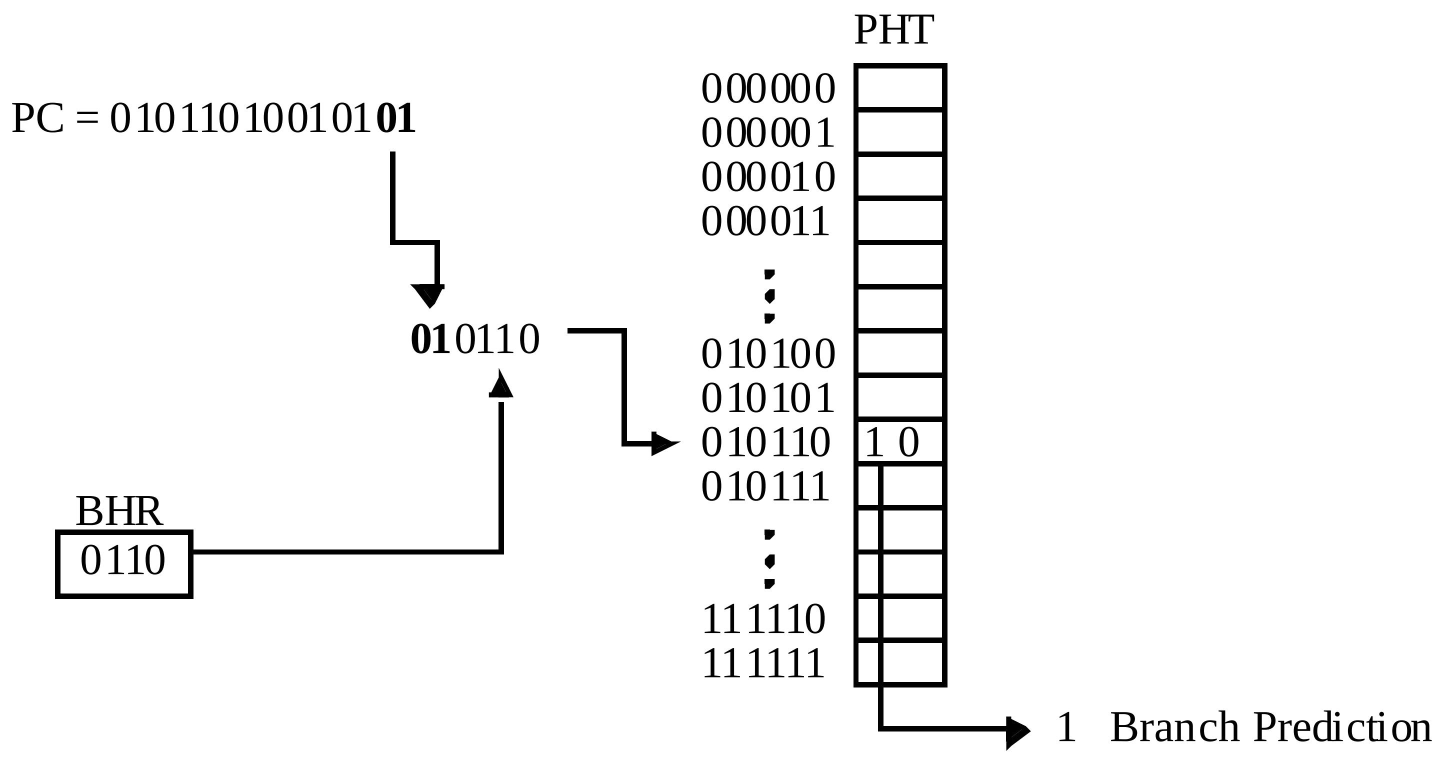
- Main advantages:
  - Learn branch behavior autonomously
    - No compiler analysis, heuristics, or profiling
  - Adapt to changing branch behavior
    - Program phase changes branch behavior
- First proposed in 1979-1980
  - US Patent #4,370,711, Branch predictor using random access memory, James. E. Smith
- Continually refined since then

Smith Predictor Hardware



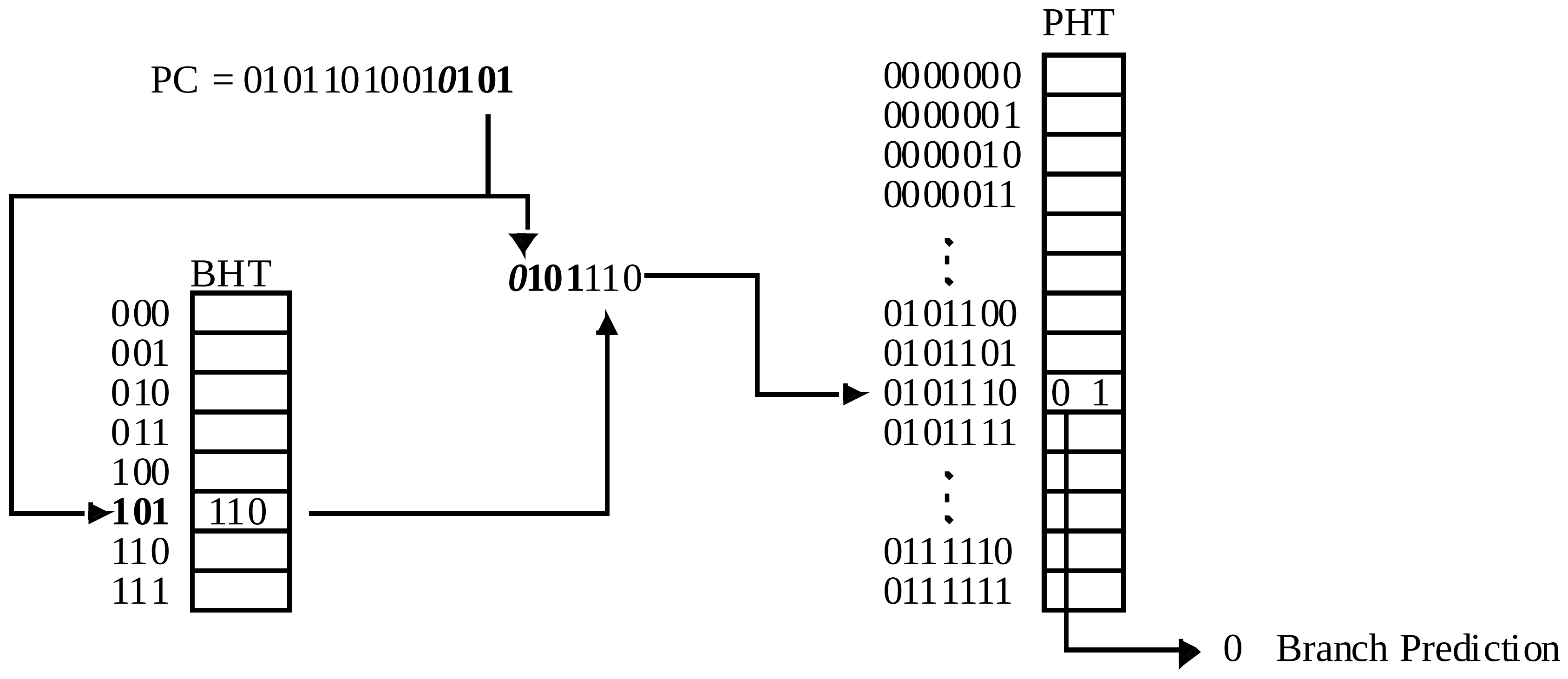
- Jim E. Smith. A Study of Branch Prediction Strategies. International Symposium on Computer Architecture, pages 135-148, May 1981
- Widely employed: Intel Pentium, PowerPC 604, PowerPC 620, etc.

Two-level Branch Prediction



- BHR adds *global* branch history
  - Provides more context
  - Can differentiate multiple instances of the same static branch
  - Can correlate behavior across multiple static branches

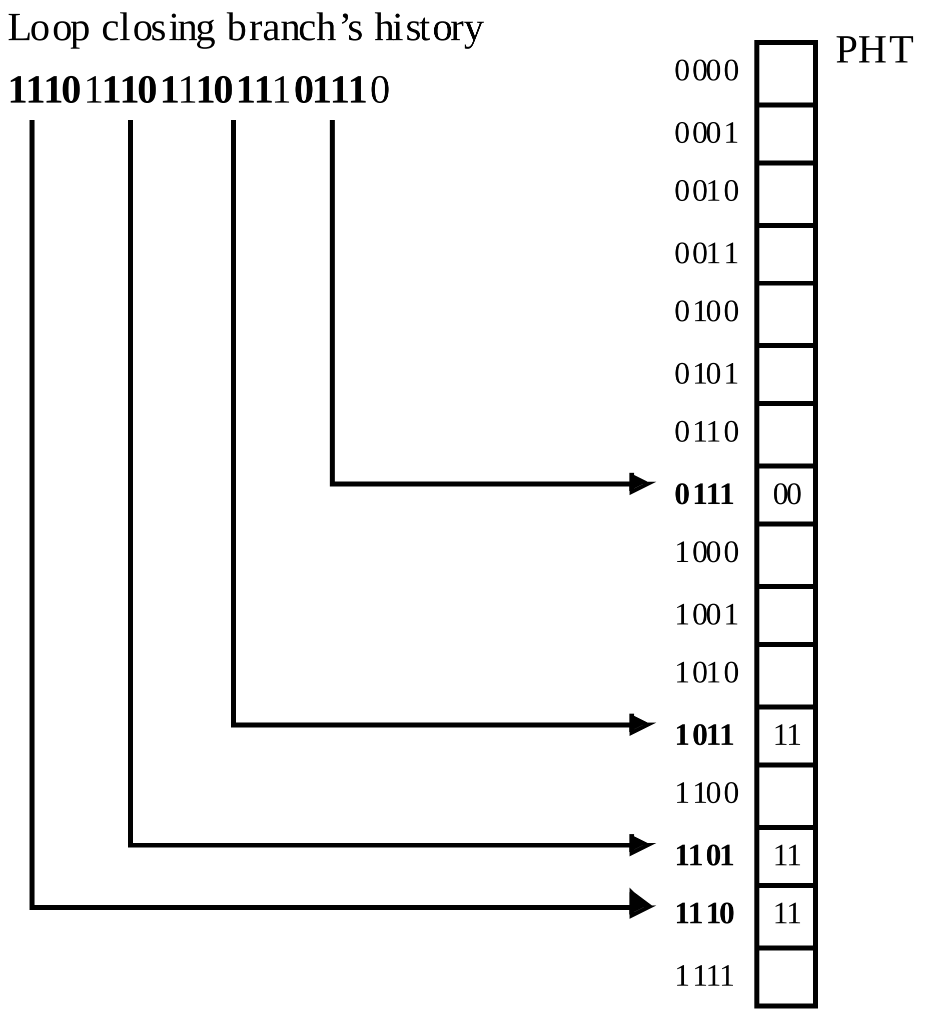
Two-level Prediction: Local History



- Detailed local history can be useful

Local History Predictor Example

- Loop closing branches
  - Must identify last instance
- Local history dedicates PHT entry to each instance
  - '0111' entry predicts not taken



Two-level Taxonomy

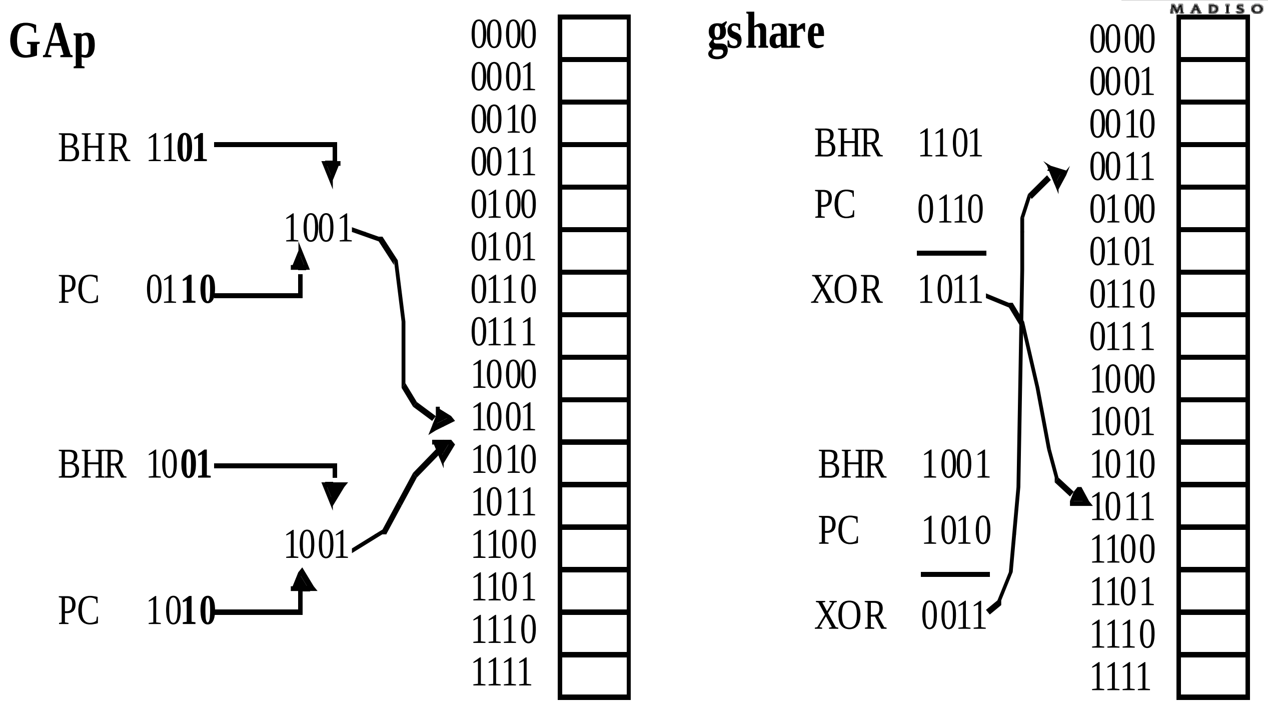
1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction

- Based on indices for branch history and pattern history
  - BHR: {G,P,S}: {Global, Per-address, Set}
  - PHT: {g,p,s}: {Global, Per-address, Set}
  - 9 combinations: GAg, GAp, GAs, PAg, PAp, PAs, SAg, SAp and SAs
- Tse-Yu Yeh and Yale N. Patt. Two-Level Adaptive Branch Prediction. International Symposium on Microarchitecture, pages 51-61, November 1991.

Index Sharing in Two-level Predictors

1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor

1991: Two-level prediction
1993: gshare, tournament



- Use XOR function to achieve better utilization of PHT
- Scott McFarling. Combining Branch Predictors. TN-36, Digital Equipment Corporation Western Research Laboratory, June 1993.
- Used in e.g. IBM Power 4, Alpha 21264

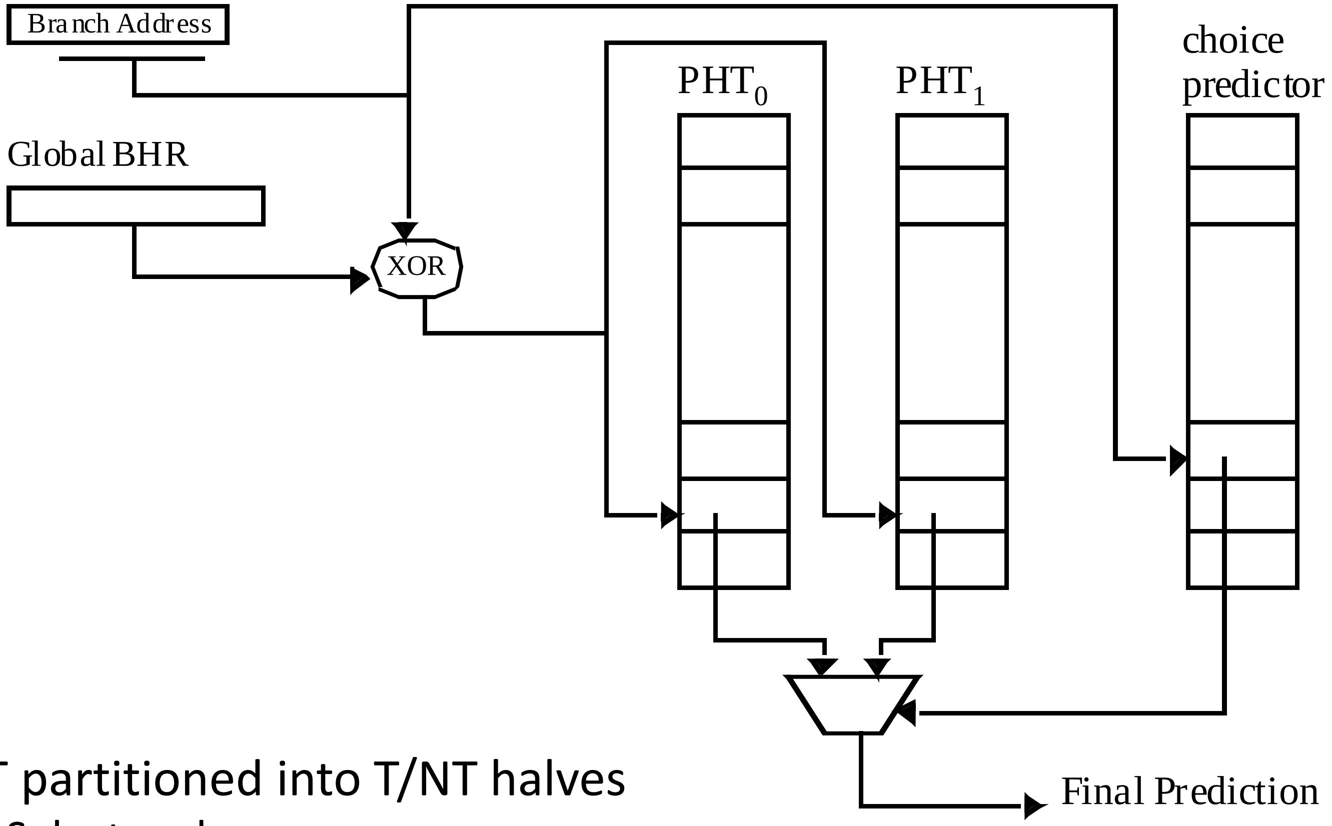
Sources of Mispredictions

- Lack of history (training time)
- Randomized behavior
  - Usually due to randomized input data (benchmarks)
  - Surprisingly few branches depend on input data values
- BHR capacity
  - Correlate to branch that already shifted out
  - E.g. loop count > BHR width
- PHT capacity
  - Aliasing/interference
    - Positive
    - Negative

Reducing Interference

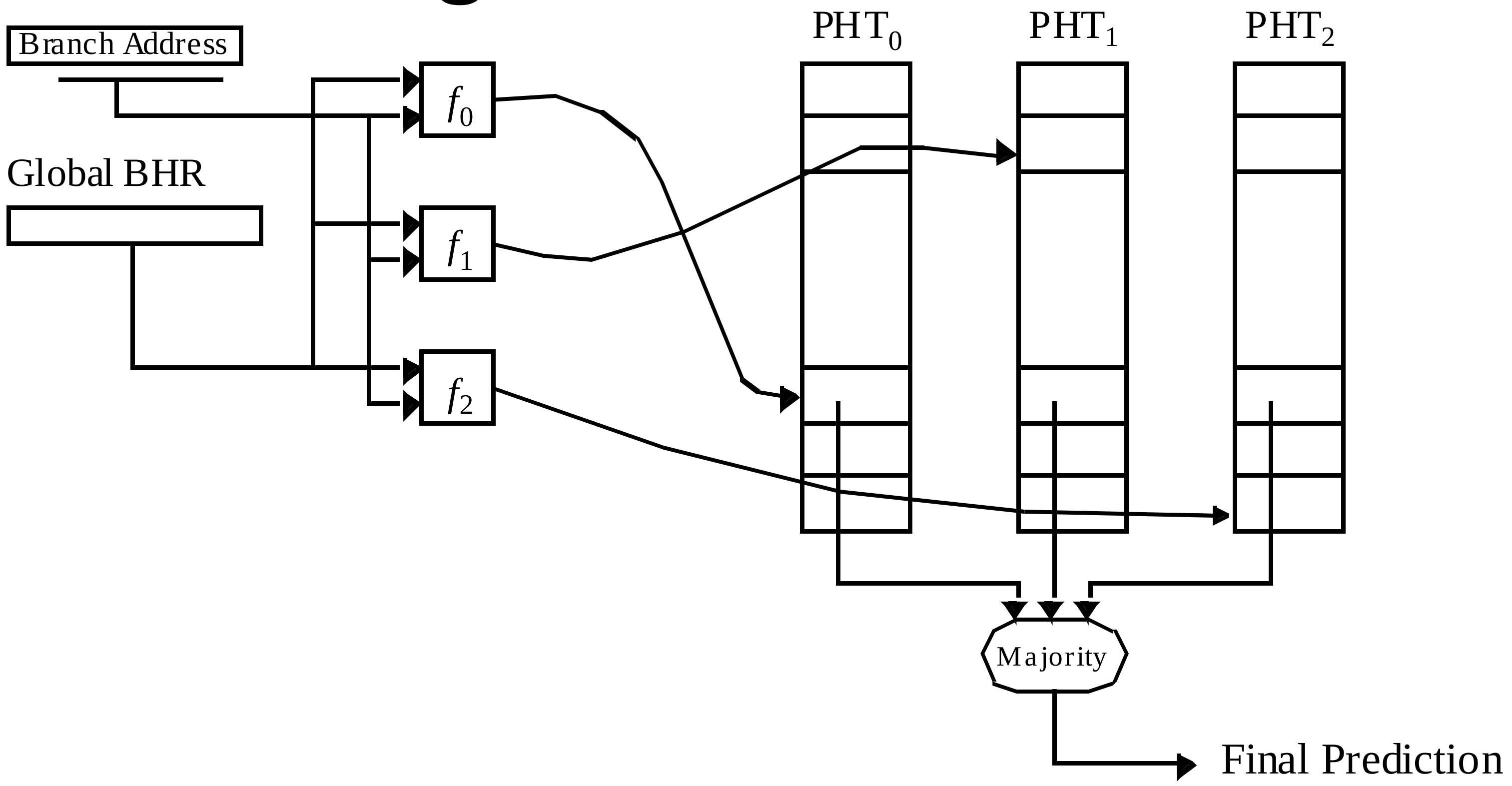
- Compulsory aliasing (*cold miss*)
  - Not important (less than 1%)
  - Only remedy is to set appropriate initial value
  - Also: beware indexing schemes with high training cost (e.g. very long branch history)
- Capacity aliasing (*capacity miss*)
  - Increase PHT size
- Conflict aliasing (*conflict miss*)
  - Change indexing scheme or partition PHT in a clever fashion

Bi-Mode Predictor



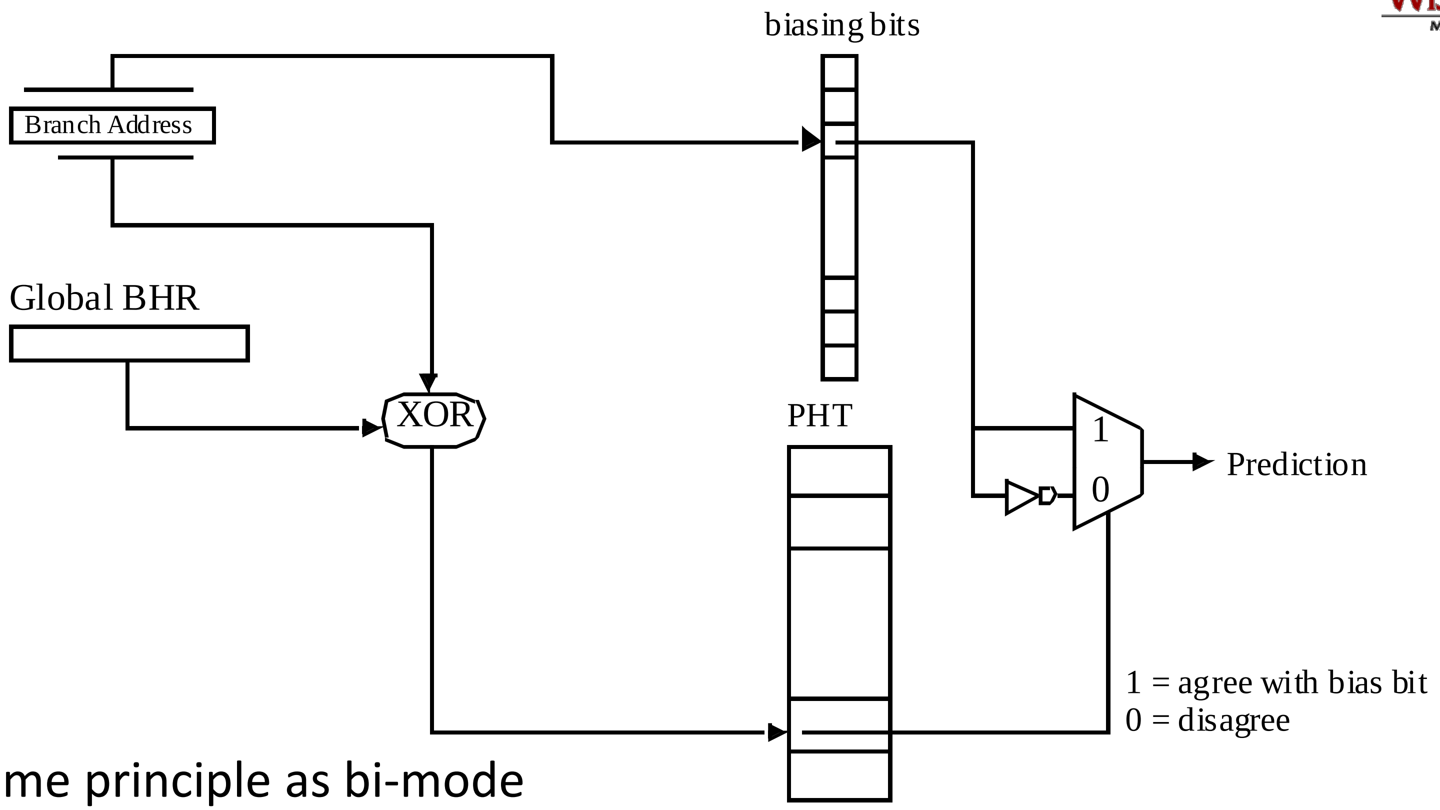
- PHT partitioned into T/NT halves
  - Selector chooses source
- Reduces negative interference, since most entries in PHT$_0$ tend towards NT, and most entries in PHT$_1$ tend towards T
- Used by ARM Cortex-A15

gskewed Predictor



- Multiple PHT banks indexed by different hash functions
  - Conflicting branch pair unlikely to conflict in more than one PHT
- Majority vote determines prediction
- Used in Alpha EV8 (ultimately cancelled)
- P. Michaud, A. Seznec, and R. Uhlig. Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. ISCA-24, June 1997

Agree Predictor



- Same principle as bi-mode
- PHT records whether branch bias matches outcome
  - Exploits 70-80% static predictability
- Used in HP PA-8700
- E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt. The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference. ISCA-24, June 1997.

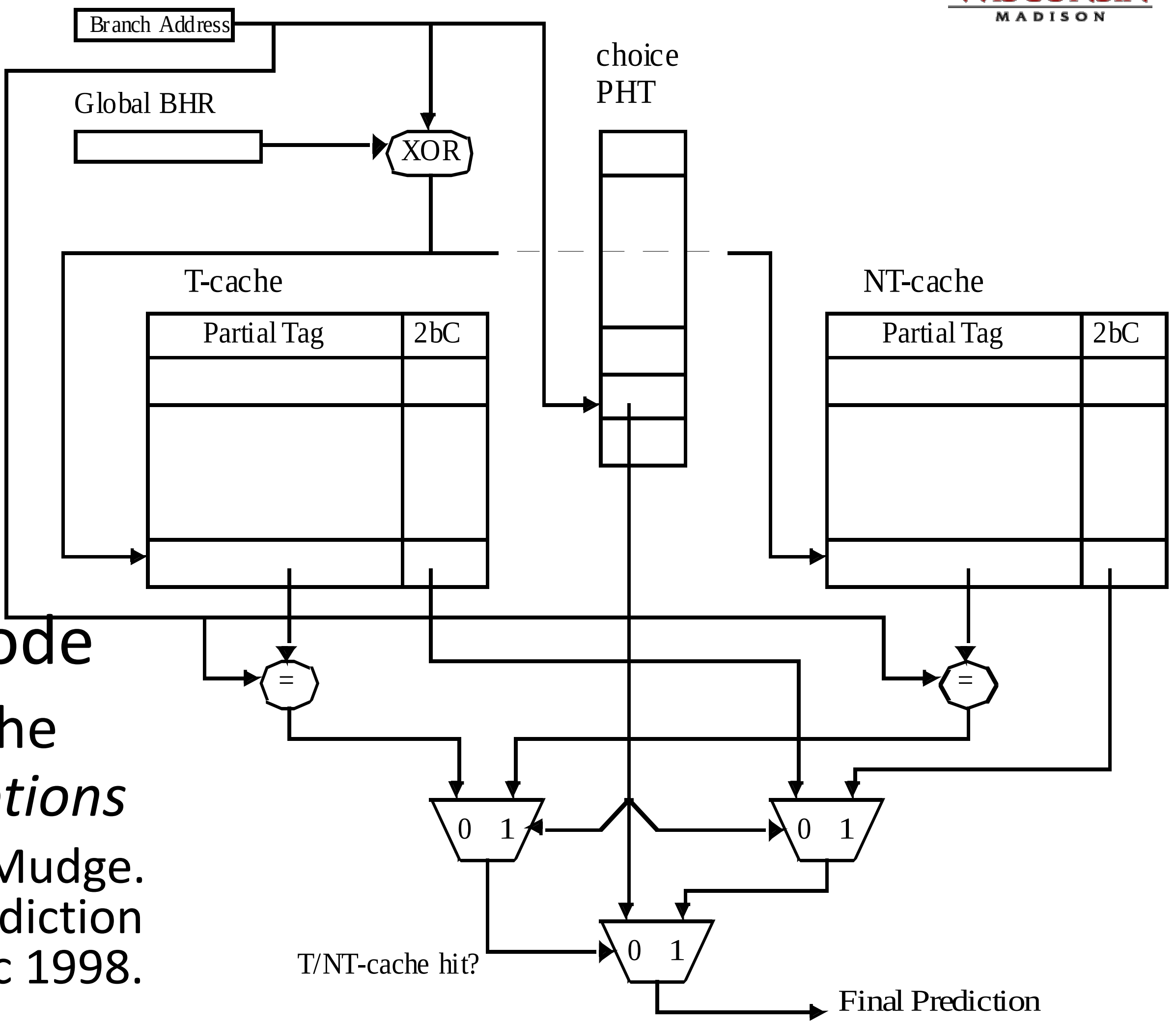
YAGS Predictor

1970: Flynn
1972: Riseman/Foster

1979: Smith Predictor

1991: Two-level prediction
1993: gshare, tournament

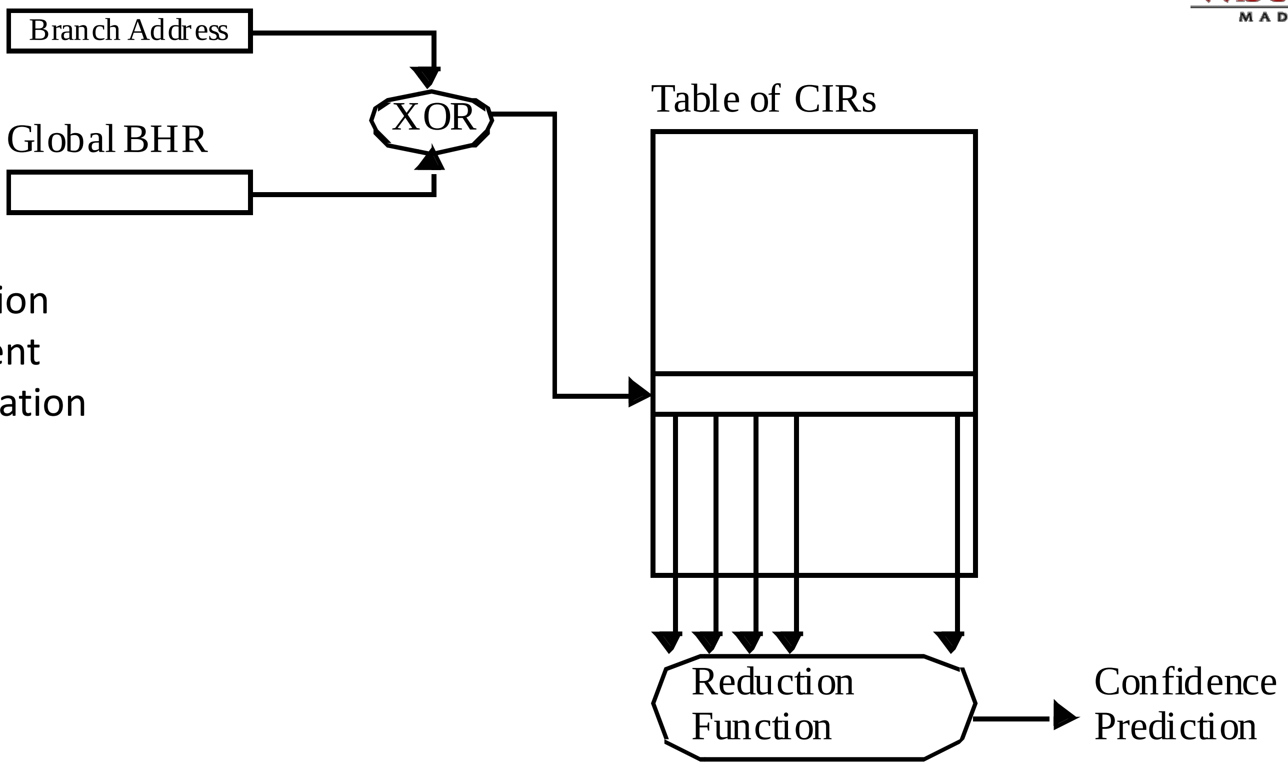
1998: Cache exceptions



- Based on bi-mode
  - T/NT PHTs cache only the *exceptions*
- A. N. Eden and T. N. Mudge. The YAGS Branch Prediction Scheme. MICRO, Dec 1998.

Branch Confidence Estimation

1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1998: Cache exceptions



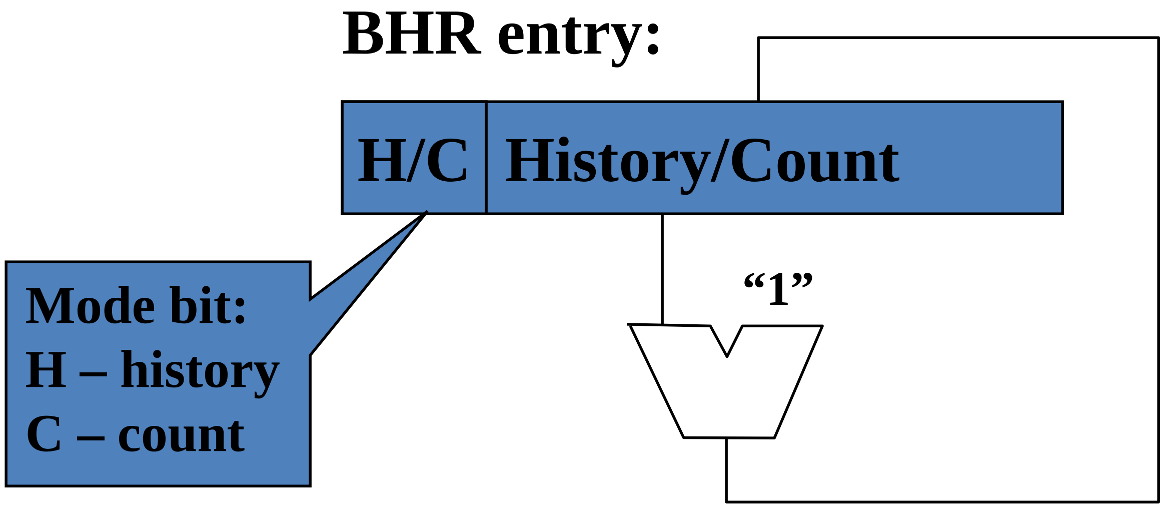
- Limit speculation (energy), reverse predictions, guide fetch for multithreaded processors, <u>choose best prediction</u>
- Q Jacobson, E Rotenberg, and JE Smith. Assigning Confidence to Conditional Branch Predictions. MICRO, December 1996.

Dynamic History Length

1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1996: Vary history length
1998: Cache exceptions

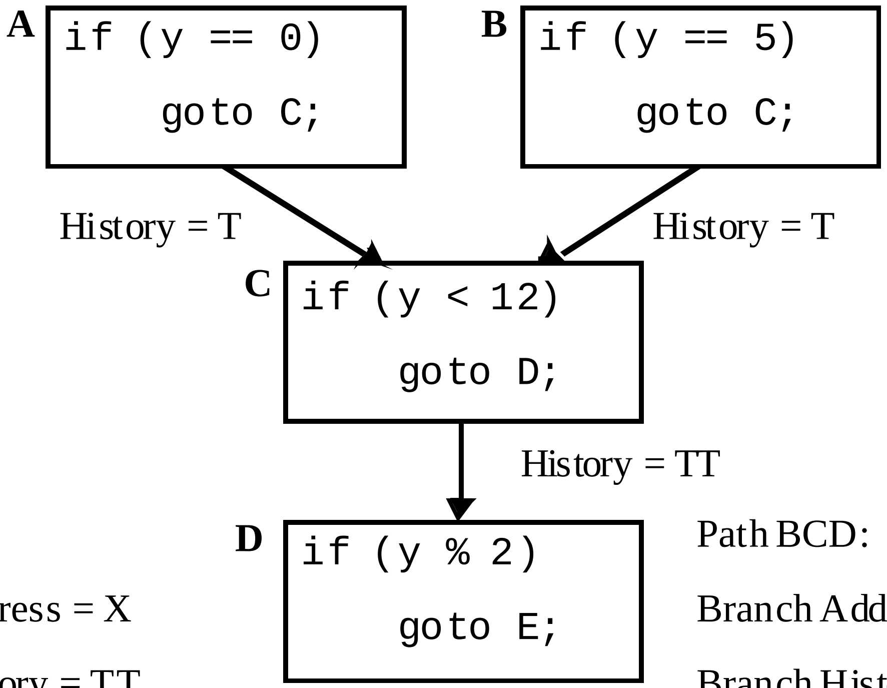
- Branch history length:
  - Some prefer short history (less training time)
  - Some require longer history (complex behavior)
- Vary history length
  - Choose through profile/compile-time hints
  - Or learn dynamically
- References
  - Maria-Dana Tarlescu, Kevin B. Theobald, and Guang R. Gao. Elastic History Buffer: A Low-Cost Method to Improve Branch Prediction Accuracy. ICCD, October 1996.
  - Toni Juan, Sanji Sanjeevan, and Juan J. Navarro. Dynamic History-Length Fitting: A Third Level of Adaptivity for Branch Prediction. ISCA, June 1998.
  - Jared Stark, Marius Evers, and Yale N. Patt. Variable Path Branch Prediction. ACM SIGPLAN Notices, 33(11):170-179, 1998

Loop Count Predictors



- To predict last loop iteration's NT branch:
  - Must have length(BHR) > loop count
  - Not feasible for large loop counts
- Instead, BHR has mode bit
  - Once history == '111…11' or '000…00' switch to count mode
  - Now nth entry in PHT trains to NT and predicts nth iteration as last one
  - Now length(BHR) > $\log_2$(loop count) is sufficient
- Used in Intel Pentium M/Core Duo/ Core 2 Duo

Path History



Path ACD:

Branch Address = X

Branch History = TT

Branch Outcome = Not Taken

Path BCD:

Branch Address = X

Branch History = TT

Branch Outcome = Taken

- Sometimes T/NT history is not enough
- Path history (PC values) can help

Understanding Advanced Predictors

- Four types of history
  - Local (bimodal) history (Smith predictor)
    - Table of counters summarizes local history
    - Simple, but only effective for biased branches
  - Local outcome history (correlate with self)
    - Shift register of individual branch outcomes
    - Separate counter for each outcome history (M-F vs Sat/Sun)
  - Global outcome history (correlate with others)
    - Shift register of recent branch outcomes
    - Separate counter for each outcome history
  - Path history (overcomes CFG convergence aliasing)
    - Shift register of recent (partial) block addresses
    - Can differentiate similar global outcome histories
- Can combine histories in many ways

Understanding Advanced Predictors

- History length
  - Short history—lower training cost
  - Long history—captures macro-level behavior
  - Variable history length predictors
- Really long history (long loops)
  - Loop count predictors
  - Fourier transform into frequency domain
    - Kampe et. al, “The FAB Predictor…”, HPCA 2002
- Limited capacity & interference
  - Constructive vs. destructive
  - Bi-mode, gskewed, agree, YAGS
  - Sec. 9.3.2 provides good overview

Perceptron Branch Prediction

1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1996: Vary history length
1998: Cache exceptions
2001: Neural predictor

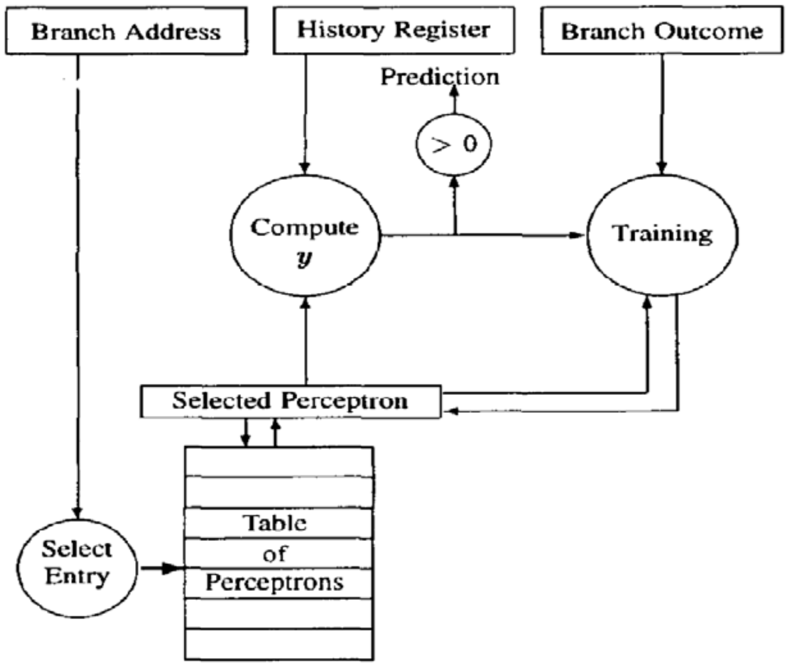
[Jimenez, Lin HPCA 2001]

$$y = w_0 + \sum_{i=1}^{n} x_i w_i$$

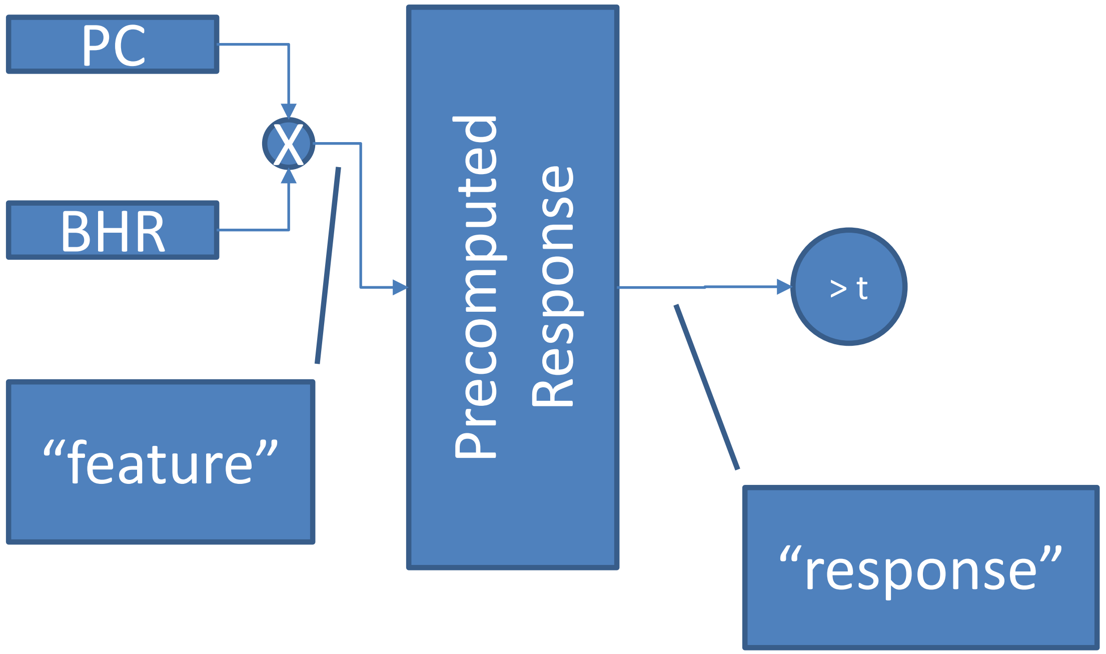
- Perceptron
  - Basis in AI concept [1962]
  - Computes boolean result based on multiple weighted inputs
- Adapted for branch prediction
  - x_i from branch history (1 T, -1 NT)
  - w_i incremented whenever branch outcome matches xi
  - Finds correlation between current branch and <u>any subset of</u> prior branches

Perceptrons - Implementation

- Complex dot product must be computed for every prediction
  - Too slow
- Arithmetic tricks, pipelining:
  - Daniel A. Jimenez and Calvin Lin. Neural methods for dynamic branch prediction. ACM Transactions on Computer Systems, 20(4):369–397, November 2002.
  - Analog circuit implementation also possible
    - Amant, Jimenez, Burger, MICRO 2008
- Key insights:
  - Not all branches in history are important, weights learn this
  - Long histories are useful

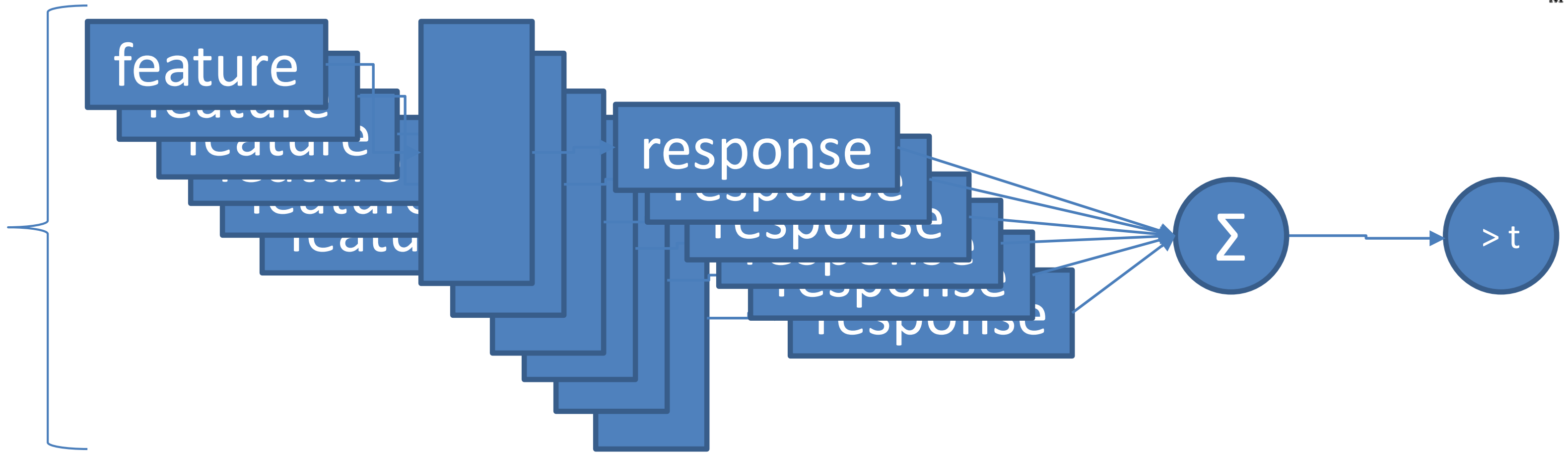


Practical Neural Predictors



- Approximate dot product function with precomputed responses
- Update (inc/dec) response based on outcomes

Practical Neural Predictors



- Many possible features (local, global, path, …)
- Responses updated based on neuron-like model
- Threshold tuned and/or updated
- Recent designs from AMD, Samsung claim “neural predictor”
  - This slide is my best guess as to what that means
- Some hints: “Multiperspective Perceptron Predictor,” Daniel Jimenez, CPB-5, ISCA 2016.

Overriding Predictors

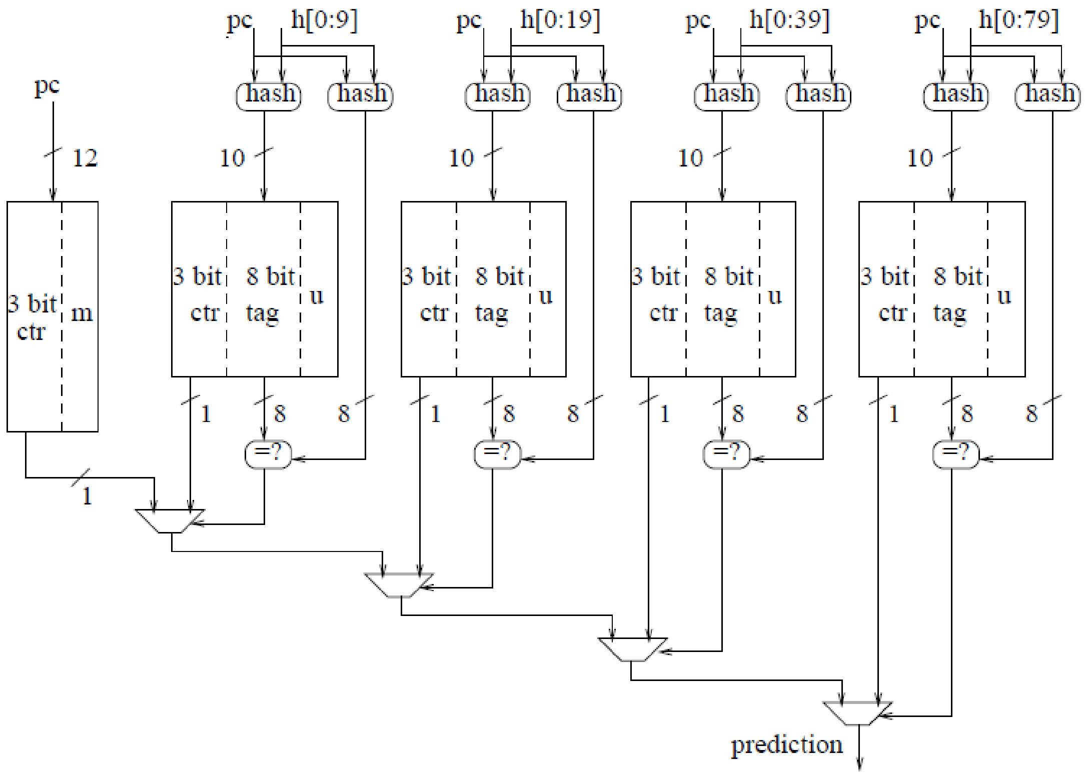
- Different types of history
  - E.g. Bimodal, Local, Global (BLG)
- Different history lengths
- How to choose?
  - Metapredictor/selector? Expensive, slow to train
- Tag match with most sophisticated predictor entry
  - Parallel tag check with B, L, G, long-history G
  - Choose most sophisticated prediction
  - Fancy predictors only updated when simple ones fail

Prediction by Partial Matching

1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor

1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1996: Vary history length
1998: Cache exceptions

2001: Neural predictor
2004: PPM



[P. Michaud, CBP-1 2004, JILP 2005]

- Elegant approach for choosing from several predictors, based on PPM data compression
- Partial tags like YAGS, varying history lengths

Current State of the Art

1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1996: Vary history length
1998: Cache exceptions
2001: Neural predictor
2004: PPM
2006: TAGE

- Key concepts
  - Different history types (B,L,G)
  - Geometric series history lengths
    - Some branches prefer short, others long
    - Use geometric series [Seznec, CBP-1, O-GEHL]
  - Cache only exceptions (YAGS/PPM)
  - Confidence estimation [Jacobson et al, MICRO 1996]
- Tagged Geometric History Length (TAGE)
  - A. Seznec, P. Michaud, "A case for (partially) tagged Geometric History Length Branch Prediction", Journal of Instruction Level Parallelism , Feb. 2006

TAGE Predictor

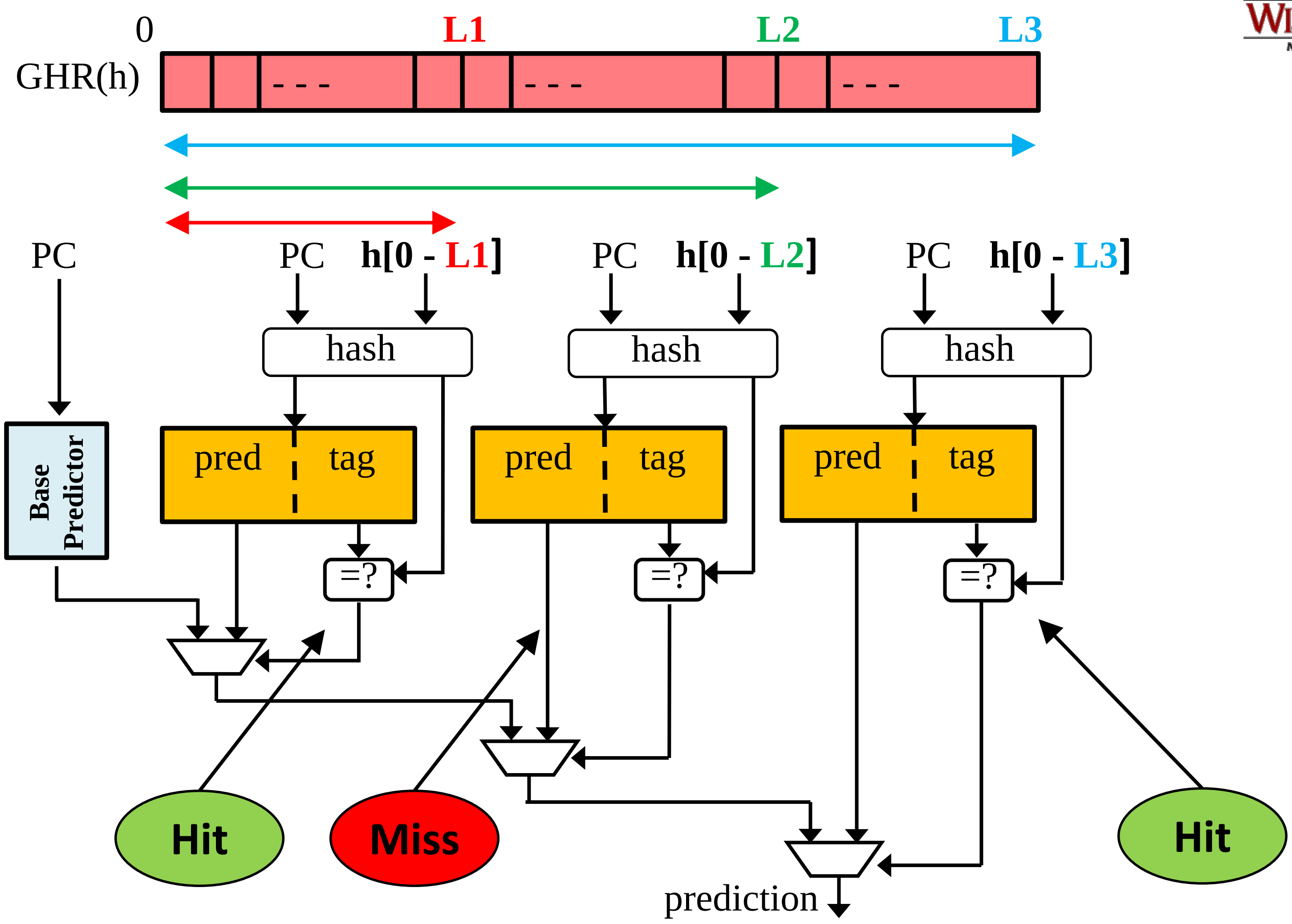
- Multiple tagged tables, use different global history lengths

- Set of history lengths forms a geometric series

$$\{\underbrace{0, 2, 4, 8, 16, 32,}_{\text{most of the storage for short history !!}} 64, 128, 256, \ldots, 2048\}$$

most of the storage for short history !!

Tagged Geometric History Length (TAGE)



- Longest matching table provides the prediction, subject to branch confidence

TAGE

1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1996: Vary history length
1998: Cache exceptions
2001: Neural predictor
2004: PPM
2006: TAGE
2016: Still TAGE vs Neural

- Tweaks to basic concept still win CBP-6
  - 1st place: TAGE-SC-L
  - 2nd place: Perceptron+TAGE hybrid
- State of the art, but…
  - Rumors of real implementation
  - Very energy-intensive (parallel lookups)
  - Complex update rules
- Real opportunity exists for improvement

TAGE vs. Neural

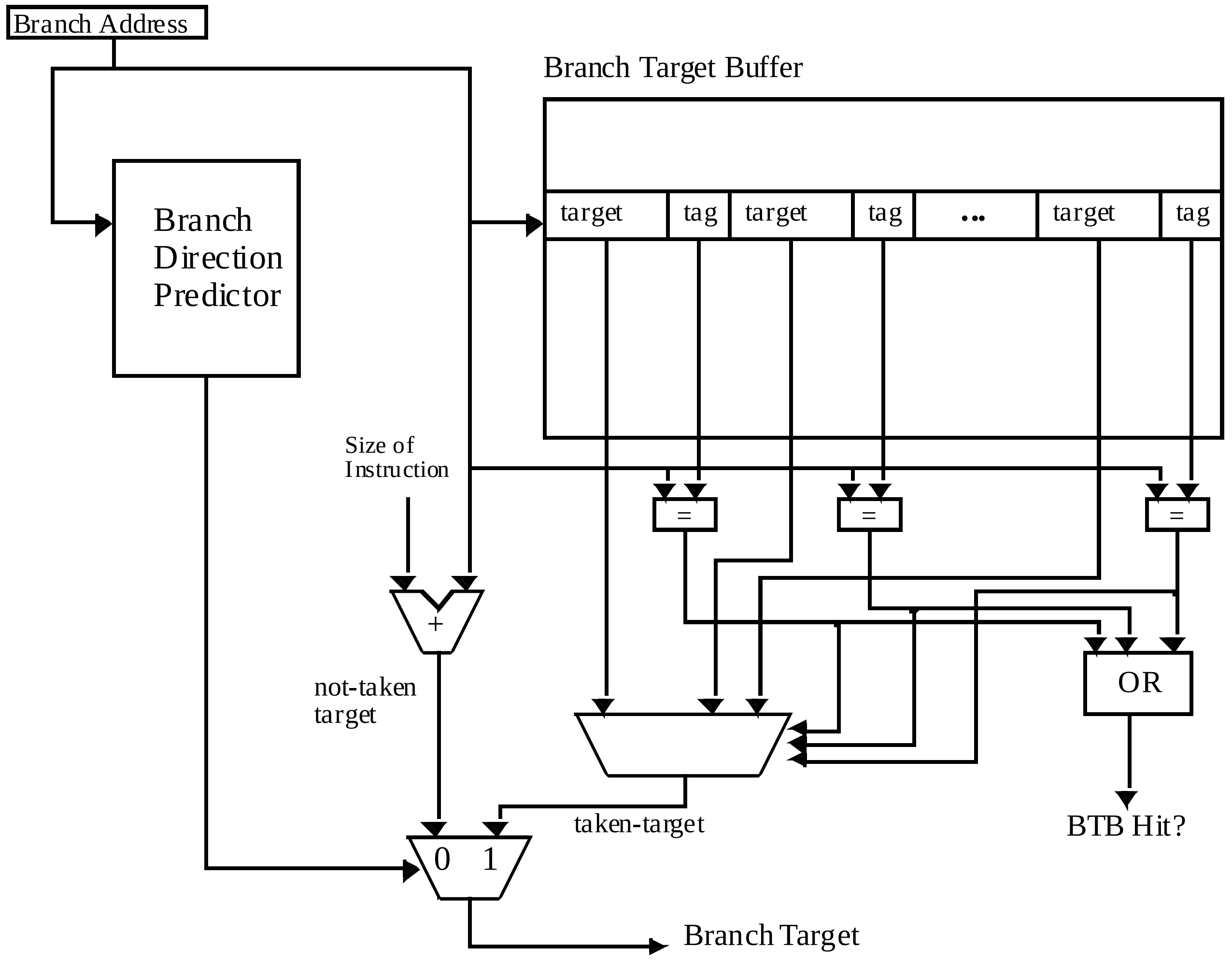
1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1996: Vary history length
1998: Cache exceptions
2001: Neural predictor
2004: PPM
2006: TAGE
2016: Still TAGE vs Neural

- Neural: ARM, AMD, Samsung
- TAGE: Intel, ???
- Similarity
  - Many sources or “features”
- Key difference: how to combine them
  - TAGE: Override via partial match
  - Neural: integrate + threshold
- Every CBP is a cage match
  - Seznec vs. Jimenez

Instruction Flow Techniques

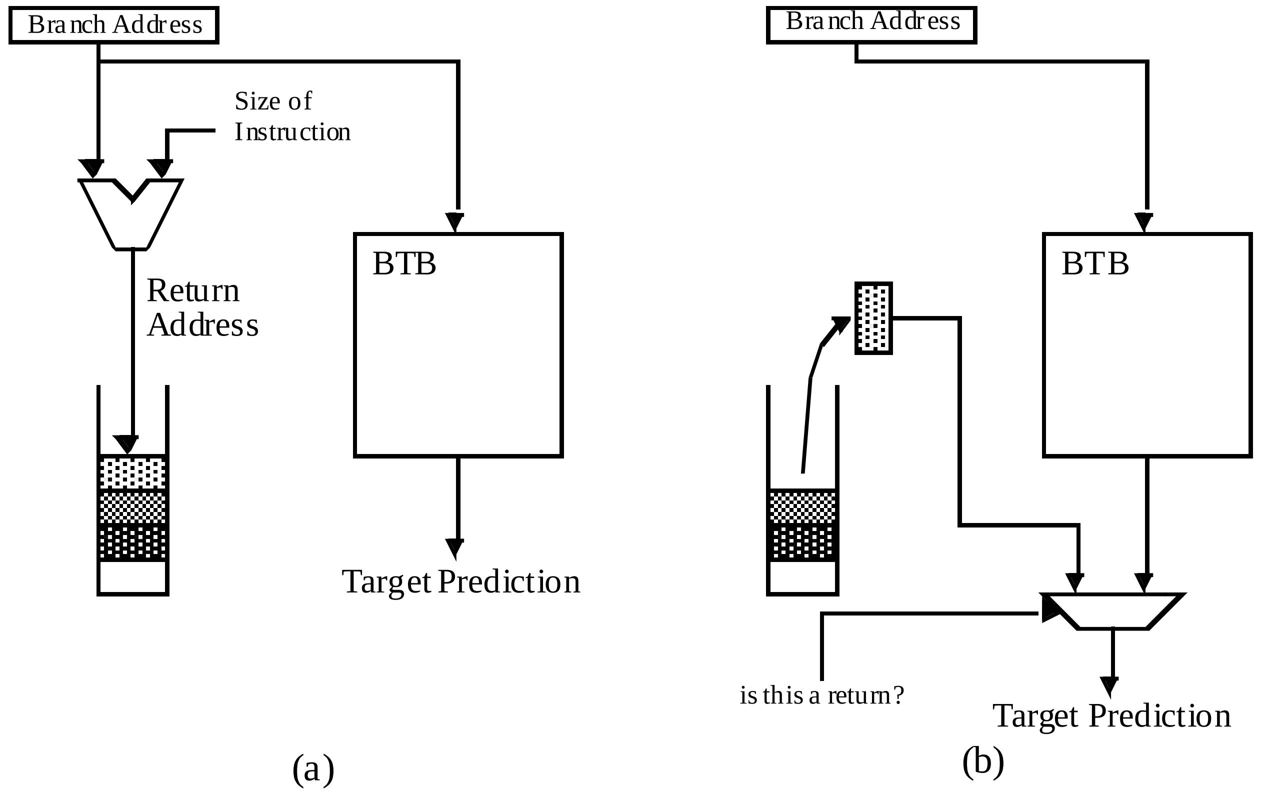
- Instruction Flow and its Impediments
- Control Dependences
- Control Flow Speculation
  - Branch Speculation
  - Mis-speculation Recovery
- Branch Direction Prediction
  - Static Prediction
  - A brief history of dynamic branch prediction
- Branch Target Prediction
- High-bandwidth Fetch
- High-Frequency Fetch

Branch Target Prediction



- Partial tags sufficient in BTB

Return Address Stack



- Speculative update causes headaches
  - On each predicted branch, checkpoint head/tail
  - Further, checkpoint stack contents since speculative pop/push sequence is destructive
  - Conditional call/return causes more headaches

Indirect Branches

- Tagged target cache
  - Chang et. al, Target Prediction for Indirect Jumps, ISCA 1997

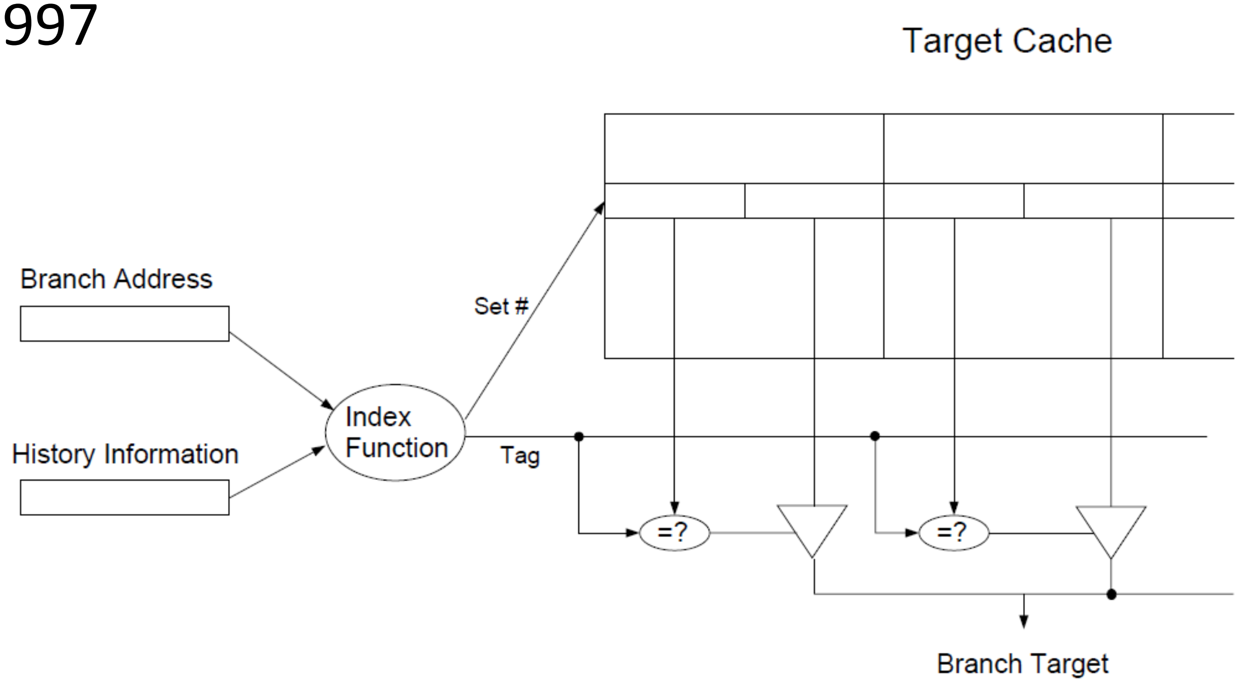


Figure 11: Structure of a Tagged Target Cache

Indirect Branches

- ITTAGE proposed in same 2006 paper as TAGE
  - A. Seznec, P. Michaud, "A case for (partially) tagged Geometric History Length Branch Prediction", Journal of Instruction Level Parallelism , Feb. 2006

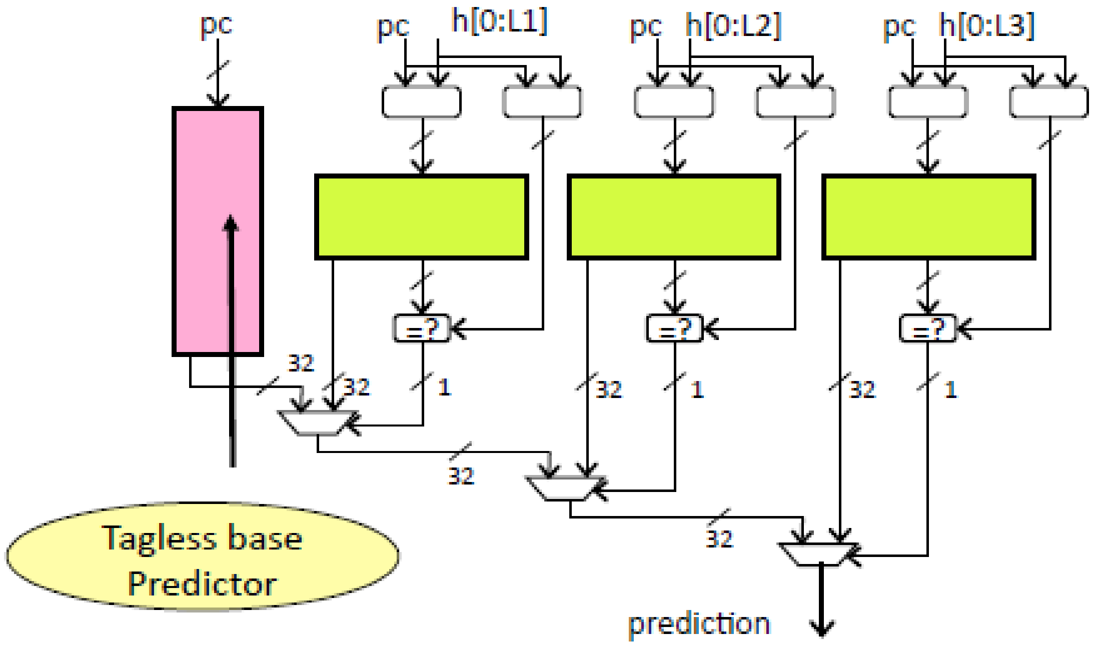
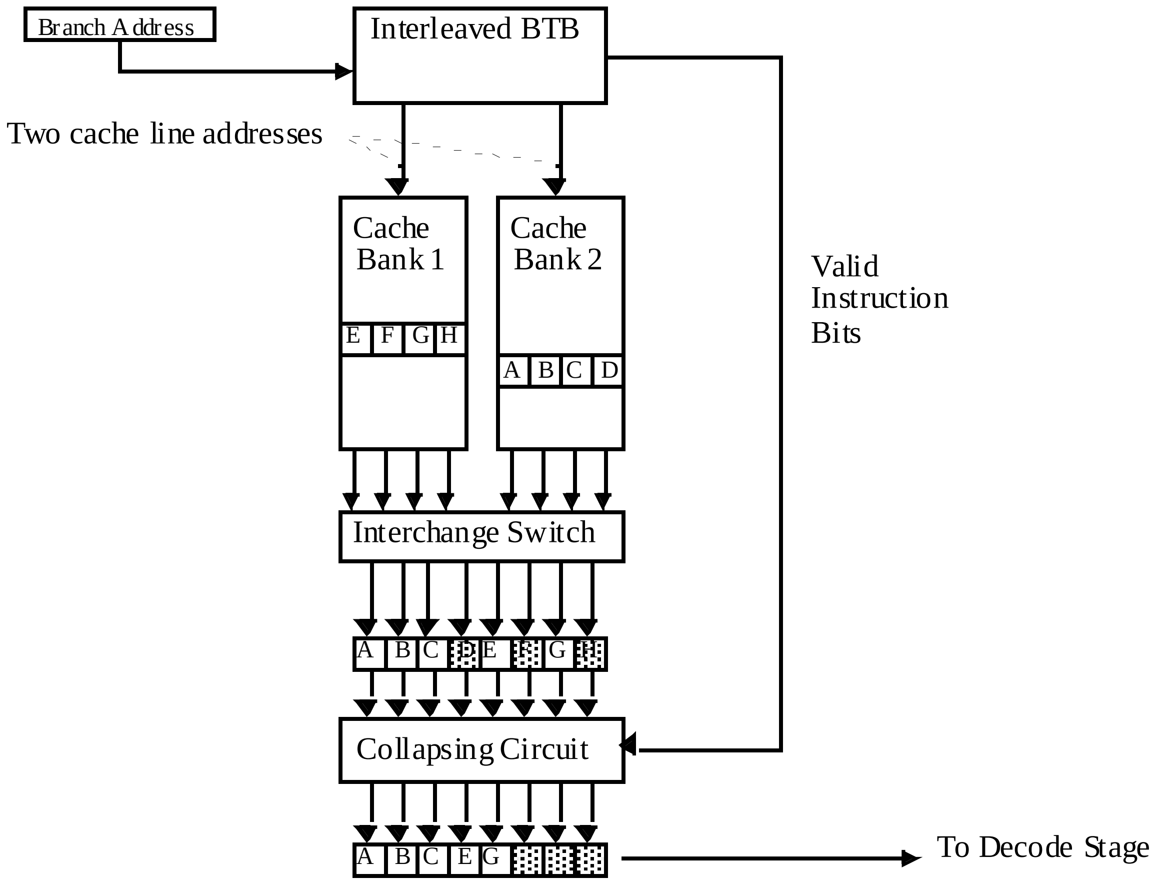


Figure 1: The Indirect Target TAgged GEometric length, ITTAGE, predictor

Indirect Branches

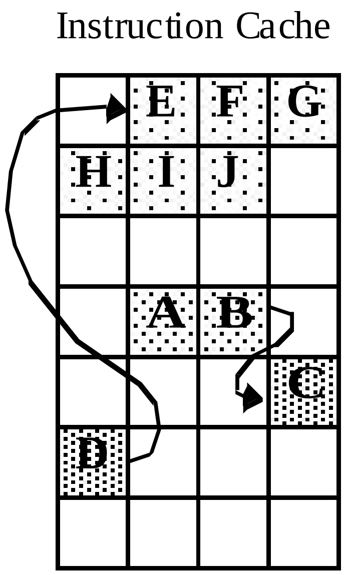
- CPB-3 had an indirect prediction track
  - #1: A. Seznec, *A 64-Kbytes ITTAGE indirect branch predictor,* MPPKI 34.1
  - #2: Y. Ishii, T. Sawada, K. Kuroyanagi, M. Inaba, K. Hiraki*, Bimode Cascading: Adaptive Rehashing for ITTAGE Indirect Branch Predictor,* MPPKI 37.0
  - #3: N. Bhansali, C. Panirwala, H. Zhou, *Exploring Correlation for Indirect Branch Prediction,* MPPKI 51.6
  - #4: Daniel A. Jimenez, *SNIP: Scaled Neural Indirect Predictor,* MPPKI *52.9*

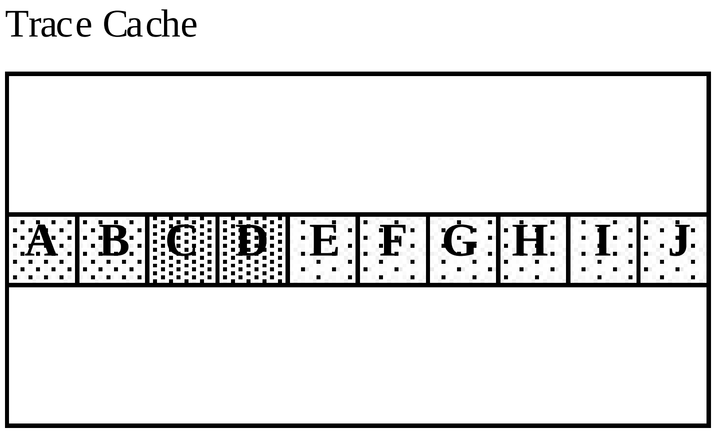
High-Bandwidth Fetch: Collapsing Buffer



- Fetch from two cache blocks, rotate, collapse past taken branches
- Thomas M. Conte, Kishore N. Menezes, Patrick M. Mills and Burzin A. Patel. Optimization of Instruction Fetch Mechanisms for High Issue Rates. International Symposium on Computer Architecture, June 1995.

High-Bandwidth Fetch: Trace Cache

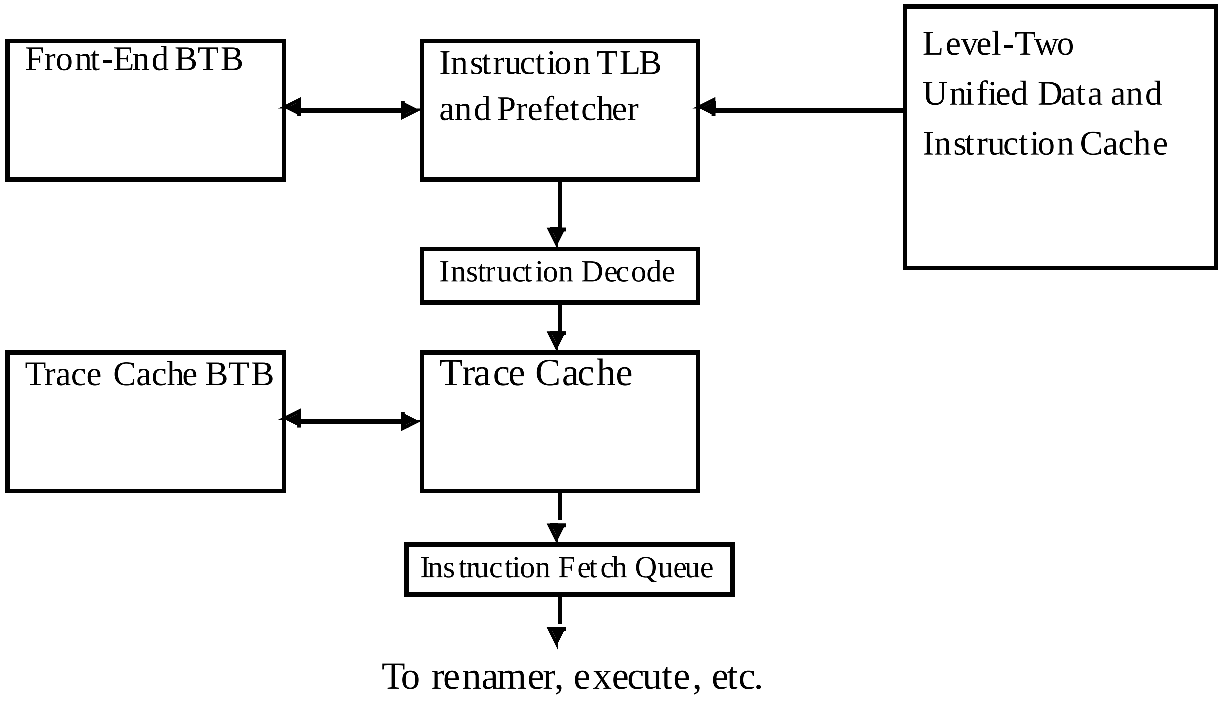




(a) (b)

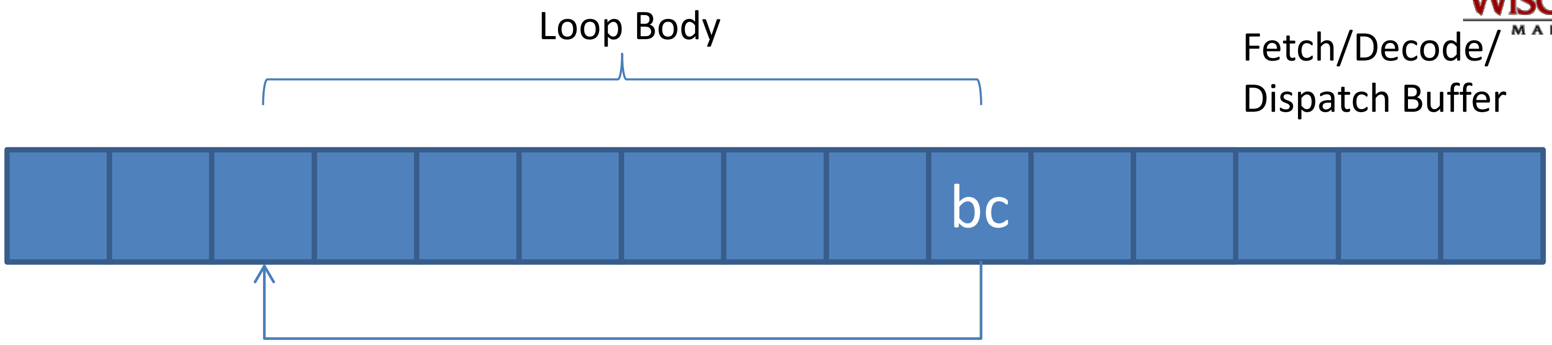
- Fold out taken branches by *tracing* instructions as they commit into a *fill buffer*
- Eric Rotenberg, S. Bennett, and James E. Smith. Trace Cache: A Low Latency Approach to High Bandwidth Instruction Fetching. MICRO, December 1996.

Intel Pentium 4 Trace Cache


Front-End BTB
Instruction TLB and Prefetcher
Level-Two Unified Data and Instruction Cache
Instruction Decode
Trace Cache BTB
Trace Cache
Instruction Fetch Queue
To renamer, execute, etc.

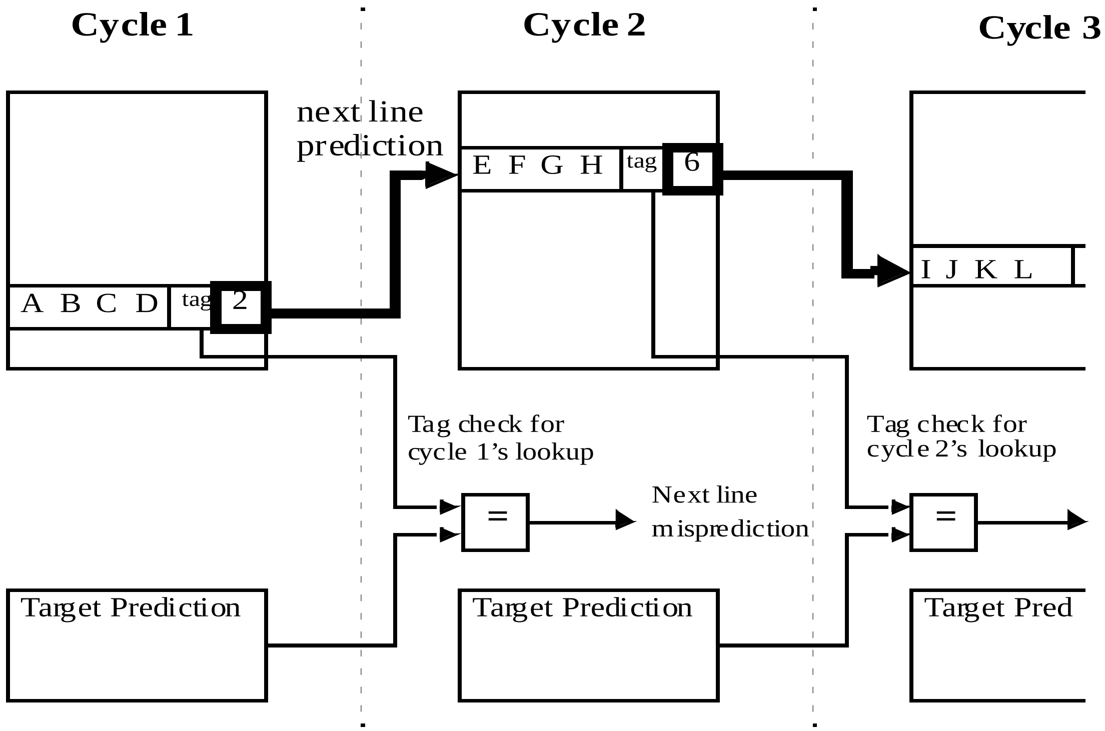
- No first-level instruction cache: trace cache only
- Trace cache BTB identifies next trace
- Miss leads to fetch from level two cache
- Trace cache instructions are decoded (uops)
- Cache capacity 12k uops
  - Overwhelmed for database applications
  - Serial decoder becomes performance bottleneck

High-Bandwidth Fetch: Loop Buffers



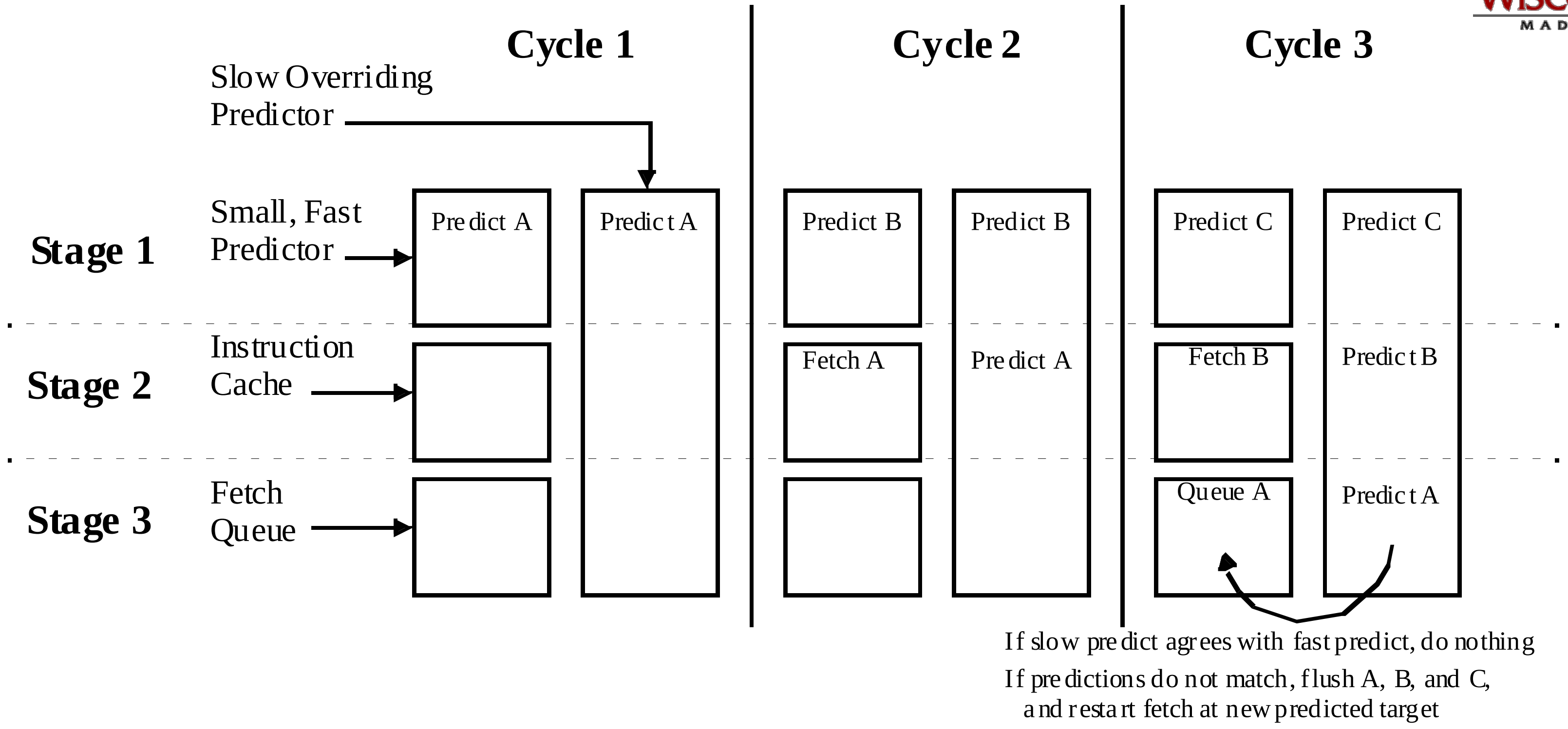
- History: AMD29K Branch Target Cache
  - Don’t cache the target address; cache 4 instructions from the target itself
  - Avoid accessing I$ for first fetch group following a taken branch
  - If loop body is <= 4 instructions, effectively a loop cache
  - Room for 32/64 branch targets
- Also common in DSP designs, under s/w control (e.g. Lucent)
- Introduced in Intel Merom (Core 2 Duo)
  - Fetch buffer detects short backward branches, inhibits refetch from I$
- Intel Nehalem (Core i7)
  - Moved loop buffer after decoders: contains uops
- Intel Sandybridge
  - General-purpose uop cache (not just loops)
  - 1.5K capacity

High Frequency: Next-line Prediction



- Embed next fetch address in instruction cache
  - Enables high-frequency back-to-back fetch
- Brad Calder and Dirk Grunwald. Next Cache Line and Set Prediction. International Symposium on Computer Architecture, pages 287-296, June 1995.

High Frequency: Overriding Predictors



- Simple, fast predictor turns around every cycle
- Smarter, slower predictor can override
- Widely used: PowerPC 604, 620, Alpha 21264

Instruction Flow Summary

- Instruction Flow and its Impediments
- Control Dependences
- Control Flow Speculation
  - Branch Speculation
  - Mis-speculation Recovery
- Branch Direction Prediction
  - Static Prediction
  - A brief history of dynamic branch prediction
- Branch Target Prediction
- High-bandwidth Fetch
- High-Frequency Fetch