CS 597d - Branch Prediction

Credits: Kevin Skadron, Tse-Yu Yeh, Dan Connors, John Sias

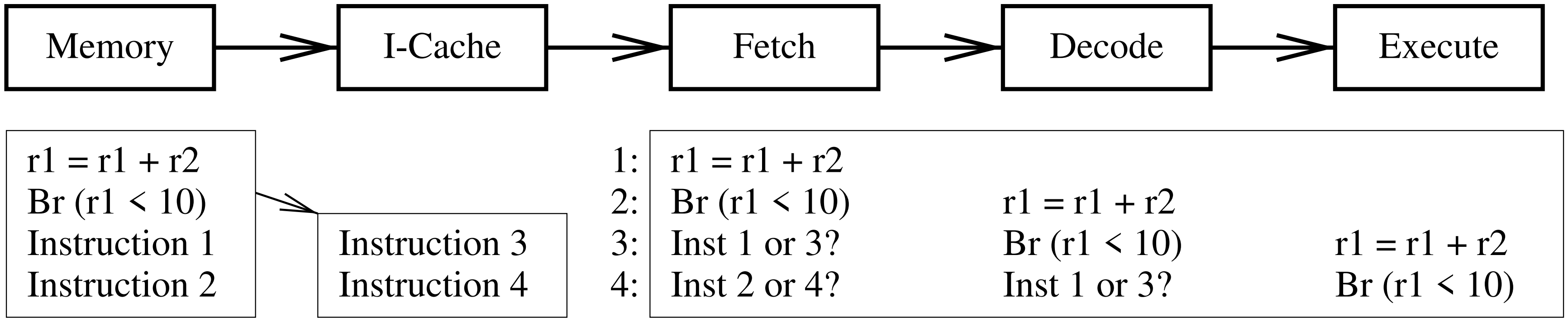
Professor David August
Fall, 1999

Course Information

- “Lecture”
  - Due to class size, “Lecture” has been redefined to be a 20-30 minute block spent presenting a single paper.
  - First opportunity will be Wednesday Sept. 29. Topic: profiling.
  - For those auditing the class: Present one paper.
- Observations:
  - Short - can be anything which demonstrates reading and thinking.
  - Regarding 1 paper (unless you choose to compare/contrast)
  - Email observations - no hardcopy necessary
- Lecture notes and observations will be password protected.
  - Username: cs
  - Password: ee

Instruction Supply

- Instruction *Fetch Rate* should meet or exceed maximum *Execution Rate*.



A Solution: Delay Slots

Memory → I-Cache → Fetch → Decode → Execute

r1 = r1 + r2
Br (r1 < 10)
Inst D1
Inst D2

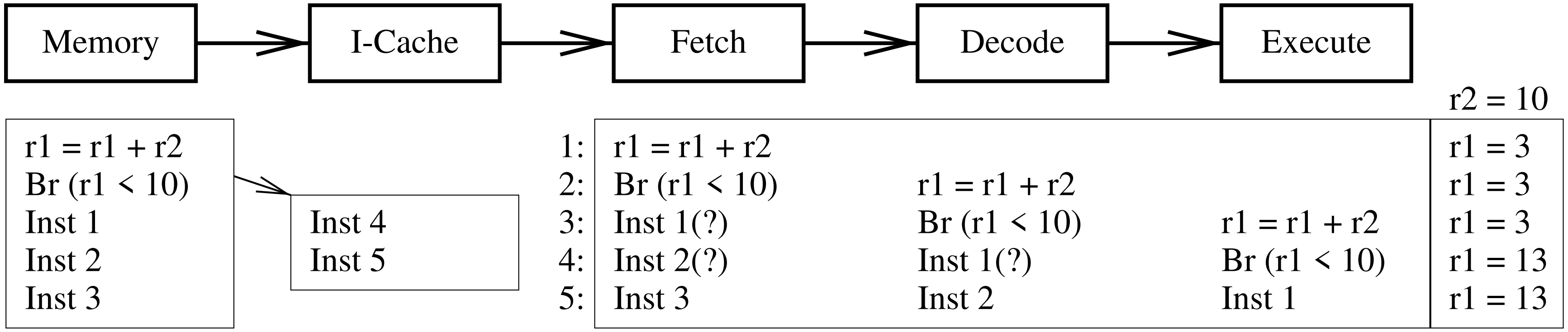
Inst 1
Inst 2

1:	r1 = r1 + r2			r1 = 3
2:	Br (r1 < 10)	r1 = r1 + r2		r1 = 3
3:	Inst D1	Br (r1 < 10)	r1 = r1 + r2	r1 = 3
4:	Inst D2	Inst D1	Br (r1 < 10)	r1 = 5
5:	Inst 1	Inst D2	Inst D1	r1 = 5

- R2000/R3000 - 1 delay slot necessary (Fetch → Decode/Execute)
  - Compiler fills the delay slot - when it can...
- R4000 - 2 delays slots necessary (Fetch → Decode → Execute)
  - Compiler fills two delay slots - when it can...
  - Legacy code is fixed with NOP insertion.
- R10000 - Out-Of-Order execution (dynamic scheduling)
  - Legacy code with delay slots filled must be handled.
  - Complicates processor design.

Better instruction supply methods exist...

A Better Solution: Branch Prediction



- A prediction is made - Taken or Not Taken
- Prediction is correct, no performance loss.
- Prediction is incorrect, misprediction penalty is paid. (at least 2 cycles here)
- Hardware-Dynamic Branch prediction mechanisms
  - N-bit counter/automaton schemes
  - Two-Level schemes

Why Is High Branch Prediction Accuracy Important?

- Small misprediction rates harm performance substantially.
- In real non-numeric codes, 1 in every 4 or 5 instructions is a branch.
- Misprediction penalties are high.
  - Alpha 21264: Minimum misprediction penalty is 7 cycles
  - Pentium II: “greater than 12 clocks”
- Wider issue machines compound the problem.
  - 6-issue machine, 10 cycle penalty $\rightarrow$ 60 slots for instruction execution wasted.
  - IA-64 - first implementation: 6-issue
  - IA-64 - next implementation: 9-issue?

Why Is High Branch Prediction Accuracy Important? (continued)

For the 21264:

Benchmark	Misprediction Rate	Performance Penalty
go	16.5%	40%
compress	9%	30%
gcc	7%	20%

The Future:

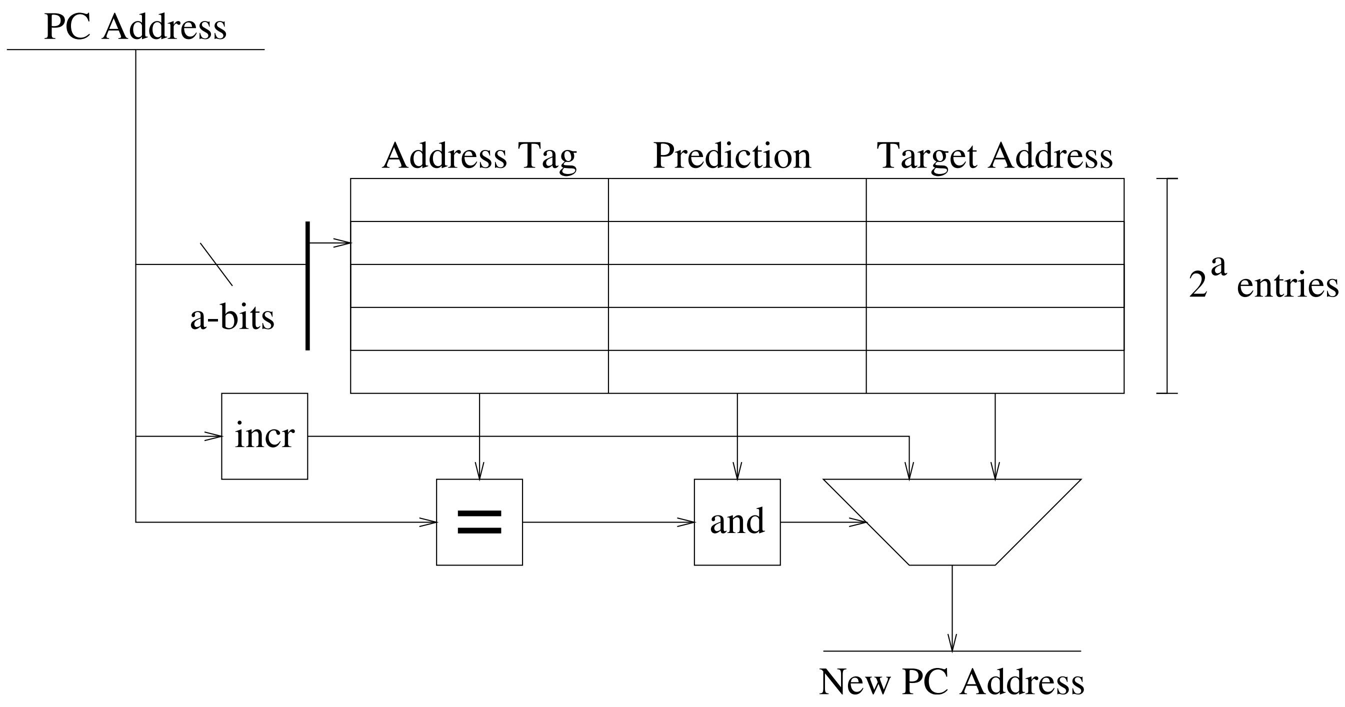
- 21264 is 4-wide → machines are getting wider.
- 21264 has a 7 cycle misprediction penalty → pipelines are getting longer.

Some claim that branch prediction will be the dominant performance bottleneck by 2010...

Some $\supseteq$ Ranganathan and Jouppi

The Branch Target Buffer

- The Branch Target Buffer (BTB) is used by the fetch unit to determine the next Program Counter (PC) value.
- Some BTBs are tag-less to reduce table size and speed prediction delivery.



First time a branch is encountered?

N-bit counter/automaton schemes

Consider:

```
Inst 1
Inst 2
Inst 3
Branch 1
Inst 4
Inst 5
Inst 6
Branch 2
```

100 Iterations of Loop
Loop exits at Branch1

Branch 1: 0 0 0 0 0....0 0 1

Branch 2: 1 1 1 1 1....1 1

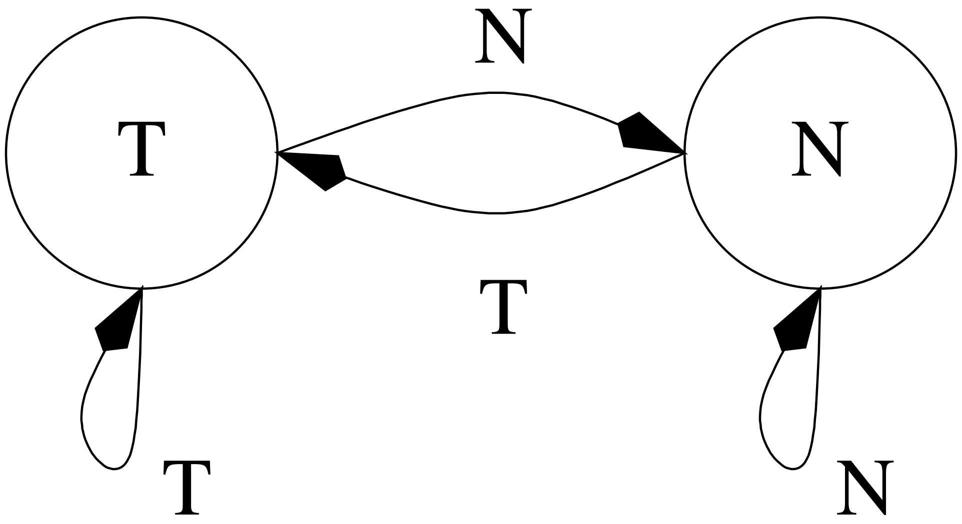
if (*f == EOF) then return;

```
    branch (r1 != EOF), L1
    rts
L1:
```

Branch: 1 1 1 1 1....1 1 0

Ideal for a "Last-Time" predictor.

Automaton: Last-Time (LT)



When is a Last-Time predictor better than a compiler static predictor?
What happens the first time the branch is seen?

Other Automata

Consider another pattern with Last-Time:

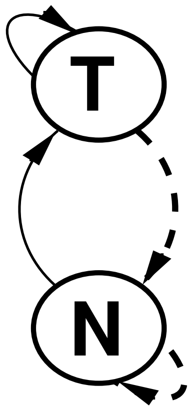
- 1 1 1 0 1 1 1 0 1 1 1 0 1
- Last-Time = 6 misses

Only 3 misses with 4 states:

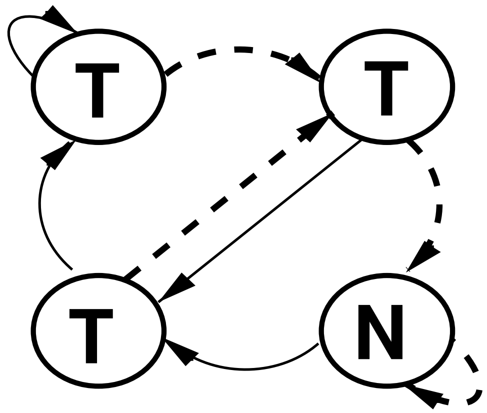
State	Meaning	Prediction
00	Strongly Not Taken	Not Taken
01	Weakly Not Taken	Not Taken
10	Weakly Taken	Taken
11	Strongly Taken	Taken

- Used by Alpha 21164, Sun UltraSPARC, MIPS R10000, and others.
- Jim Smith, 1991

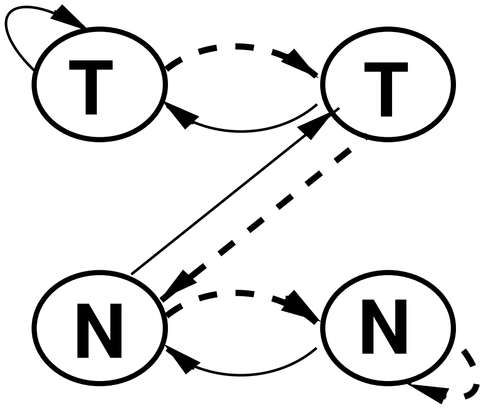
Other Automata



Automaton Last-Time (LT)

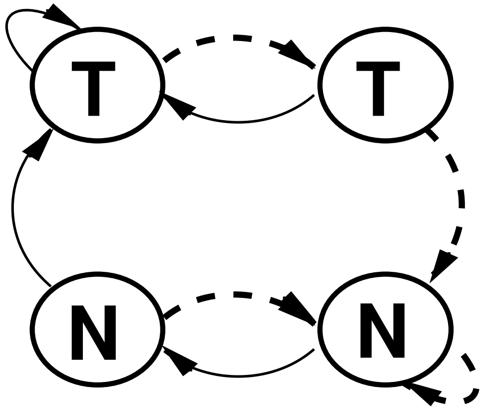


Automaton A1

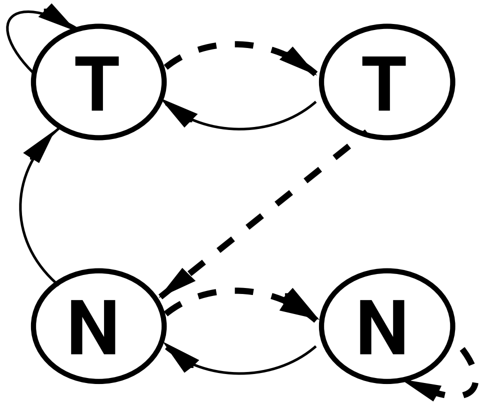


Automaton A2

(2-bit Saturating Up-down Counter)



Automaton A3



Automaton A4

Branch Correlation

- All these automata capture only *Self-Correlation* of branches.
- Branches can be correlated: (From *eqntott*, SPEC92)

```
if (aa == 2)                  /* branch 1 */
   aa = 0;
if (bb == 2)                  /* branch 2 */
   bb = 0;
if (aa != bb) {               /* branch 3 */
   .....
}
```

- If the conditions of branch 1 and branch 2 are TRUE, the condition of branch 3 is FALSE.

Can other-branch history be used to improve prediction?

Branch Correlation (continued)

- If *aa* and *bb* are independent and assume the values 0, 1, 2, and 3 with uniform distribution:

	Probability of		
History	Branch 3 TRUE	Branch 3 FALSE	Predict
FF	0.667	0.333	T
FT	0.667	0.333	T
TF	0.667	0.333	T
TT	0.000	1.000	F

- Static prediction for each history. (not to be confused with static branch prediction)
- Using branch history and predictions indicated → *expect* 68.5% prediction rate.

Values are rarely uniformly distributed and independent....

Adaptive Predictions for Histories

- Each branch history gets its own automaton.
- 00 = A, 01 = B, 10 = C, 11 = D

```
C A B B A D D B D D C C A B D A C D D A
T T N T T T N T N N N N T T N T T N N T

A: T T T T T
B: N T T T
C: T N N T
D: T N N N N N N
```

- Prediction rate now 75%.
- 70% for static history. (68.5% expected)
- 60% for static prediction of T.
- 45% for self-correlation automaton - Smith’s 2-bit Counter.

Types of History

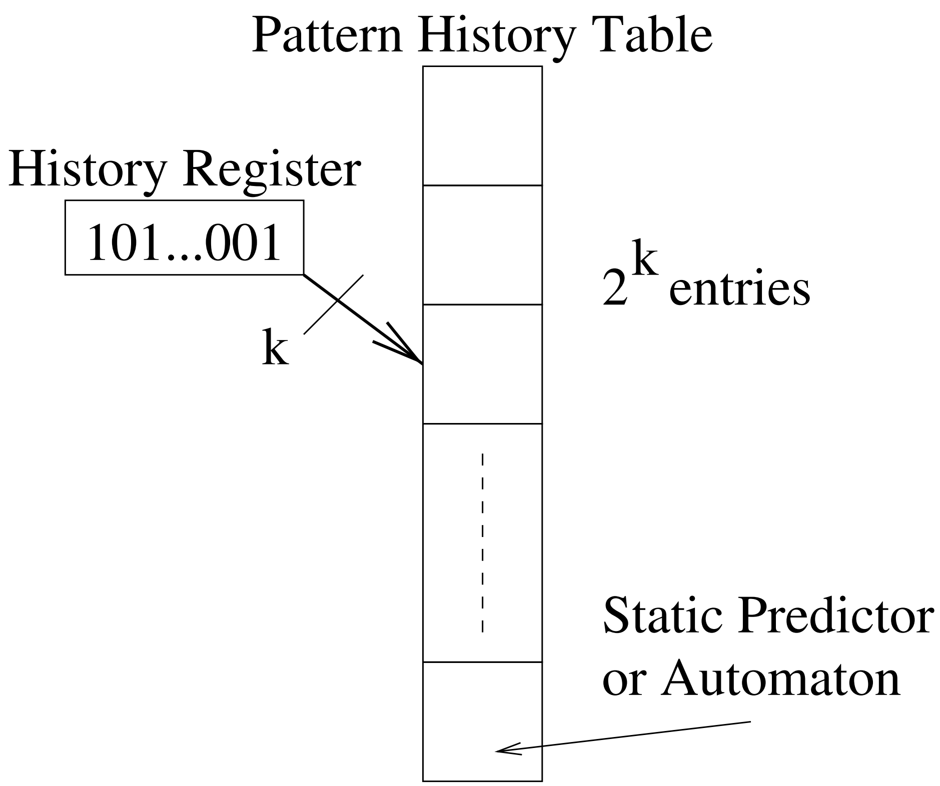
Histories can be:

- *Global*
  - One history for all branches - as we have been using. (Alpha 21264)
  - The history of one branch influences the prediction of another.
- *Per-address*
  - One history for each branch - branch histories are independent.
  - Works well for: 00110011 patterns of a single branch.
- *per-Set*
  - One history for a set of branches.
  - A hash function of the address is used to determine set membership.

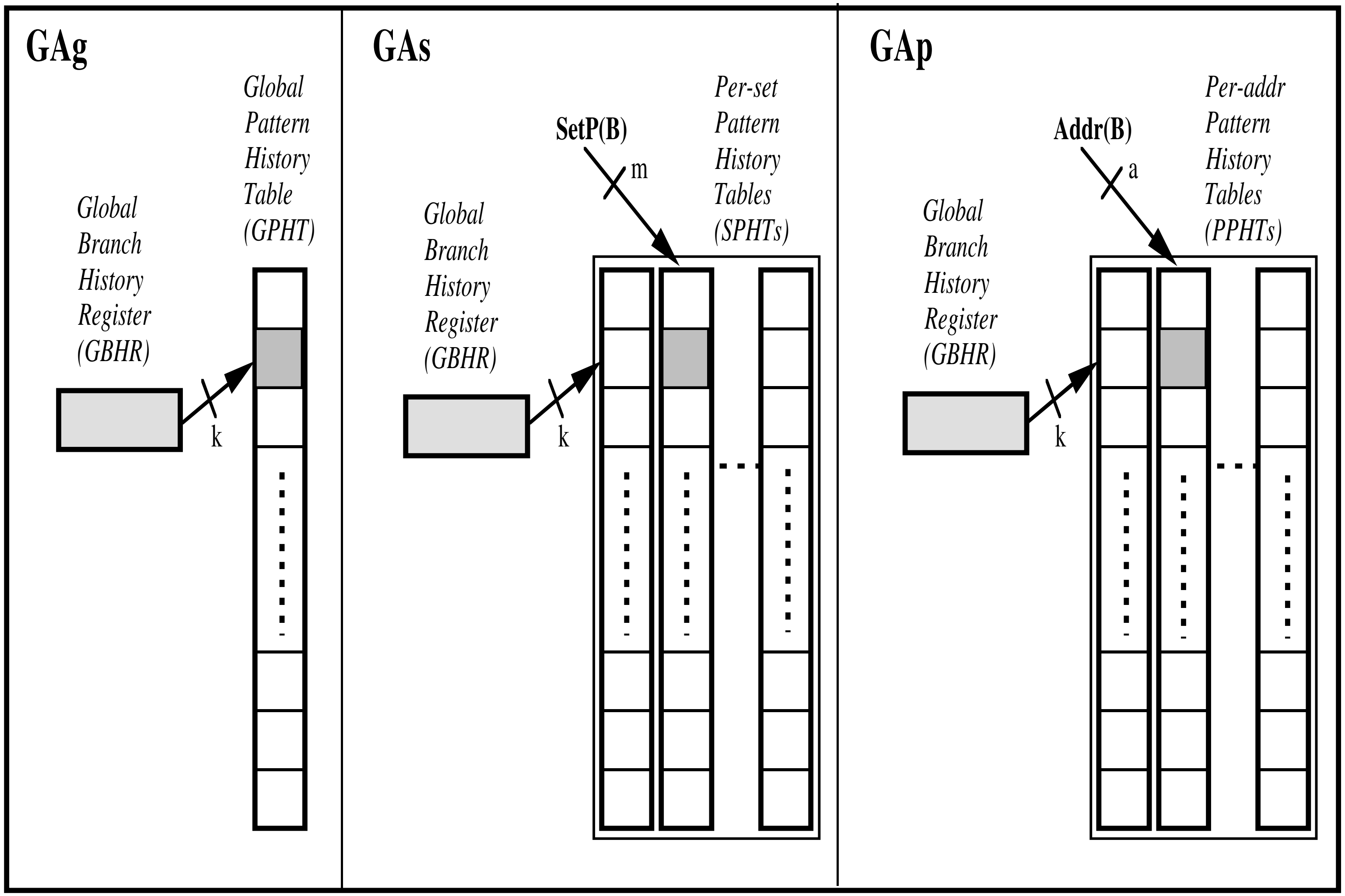
Types of Pattern History Tables

Pattern History Tables can be:

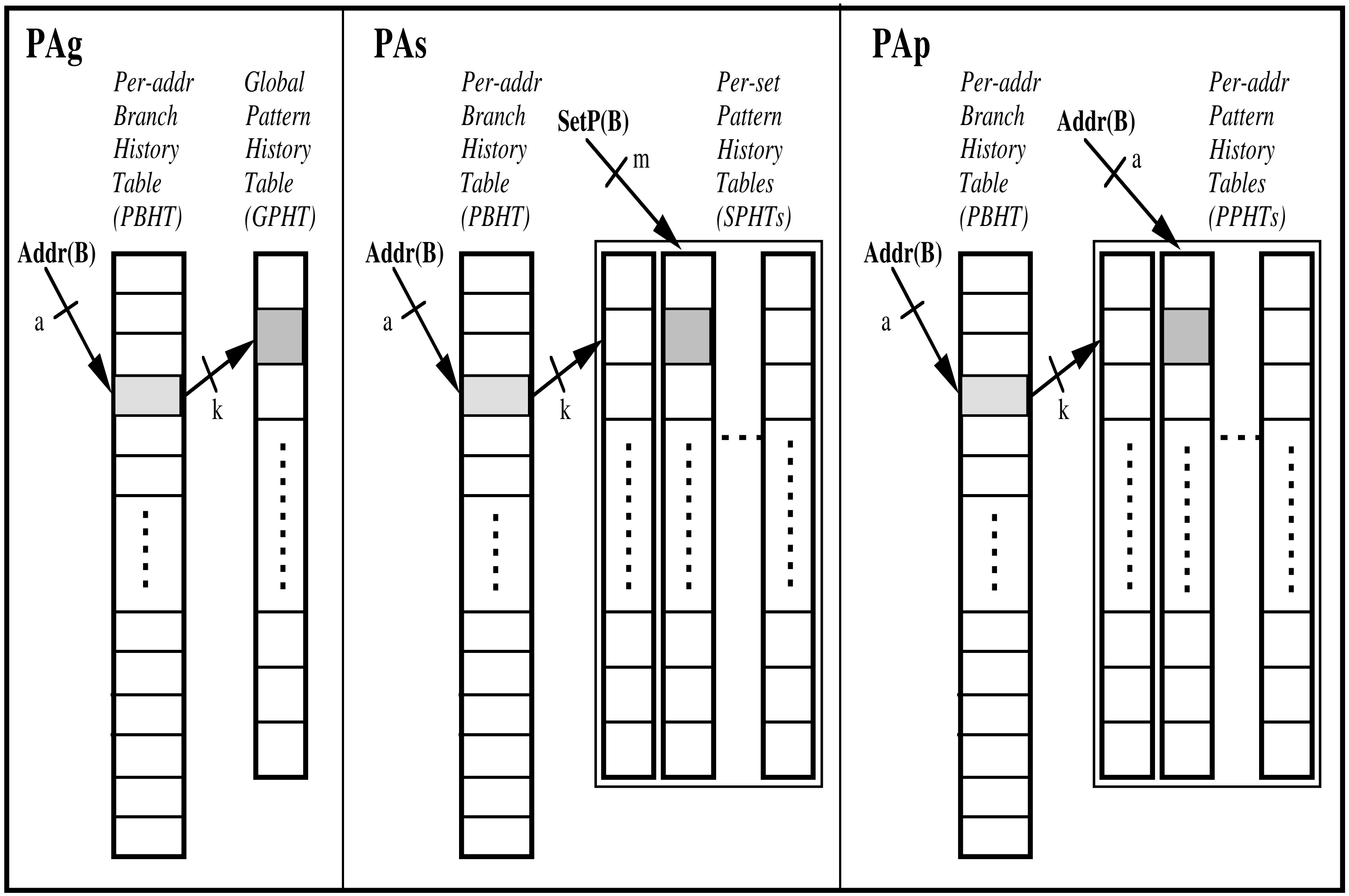
- *Global* - one predictor for each pattern shared with all branches.
- *Per-address* - one predictor for each pattern for each branch.
- *per-Set* - one predictor for each pattern for each set of branches.



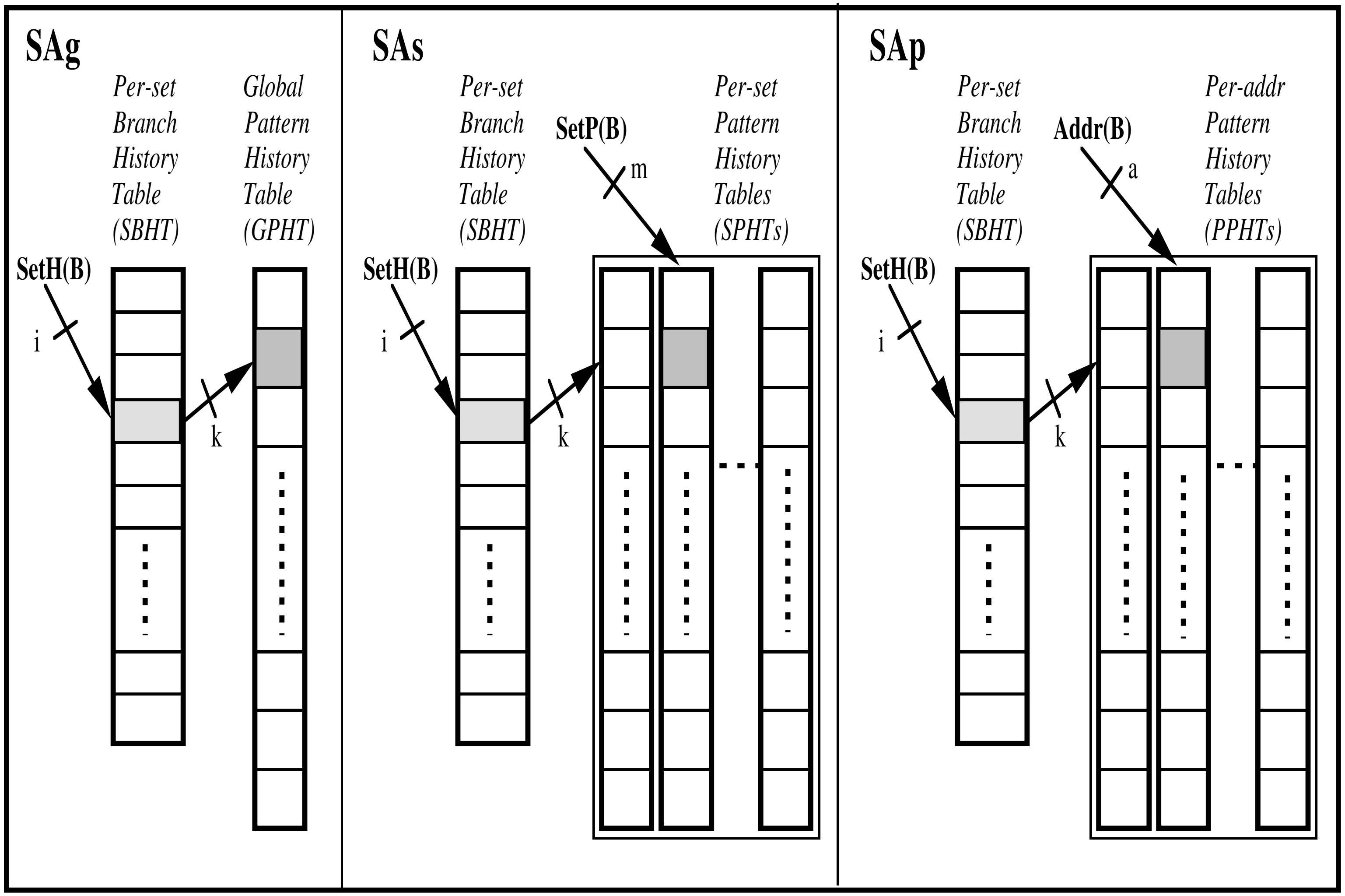
The Global History Adaptive Schemes



The Per-Address History Adaptive Schemes



The Per-Set History Adaptive Schemes



Other Two-Level Schemes Exist

gshare

- A single global branch history.
- A single branch history table.
- Index into branch history table is computed as the XOR of the branch address and the global branch history.

gselect

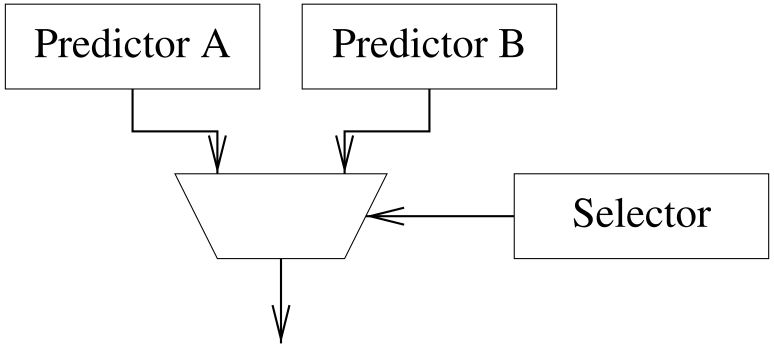
- Similar to gshare.
- Index into branch history table is computed as the CONCATENATION of the branch address and the global branch history.

others

- yags - yet another gshare
- variable history length predictors

Hybrid Predictors

- If one is good, why not two, three, etc.?
- Hybrid (or multiple-scheme) branch prediction was proposed by Scott McFarling, WRL 1993
- Different predictors are adept at capturing different branch correlation. (global vs. local history)
- Use compiler or hardware to select predictor at run-time.
- Predictor selection mechanisms are very similar to branch predictors - predict which predictor is going to predict the branch correctly.



Summary

Consider for each branch predictor:

- Hardware cost
- Prediction rate
- Warm-up time
- Correlation
- Interference
- Time to generate prediction

Topics for later:

- Speculative update
- Subroutine calls - return address stack

Reading

- "Compiler Synthesized Dynamic Branch Prediction"
- "Architectural Support for Compiler-Synthesized Dynamic Branch Prediction Strategies: Rationale and Initial Results"
- Please be critical of these papers in your observations. The project may be related to the shortcomings of these papers.