18-747 Lecture 8: Instruction Flow

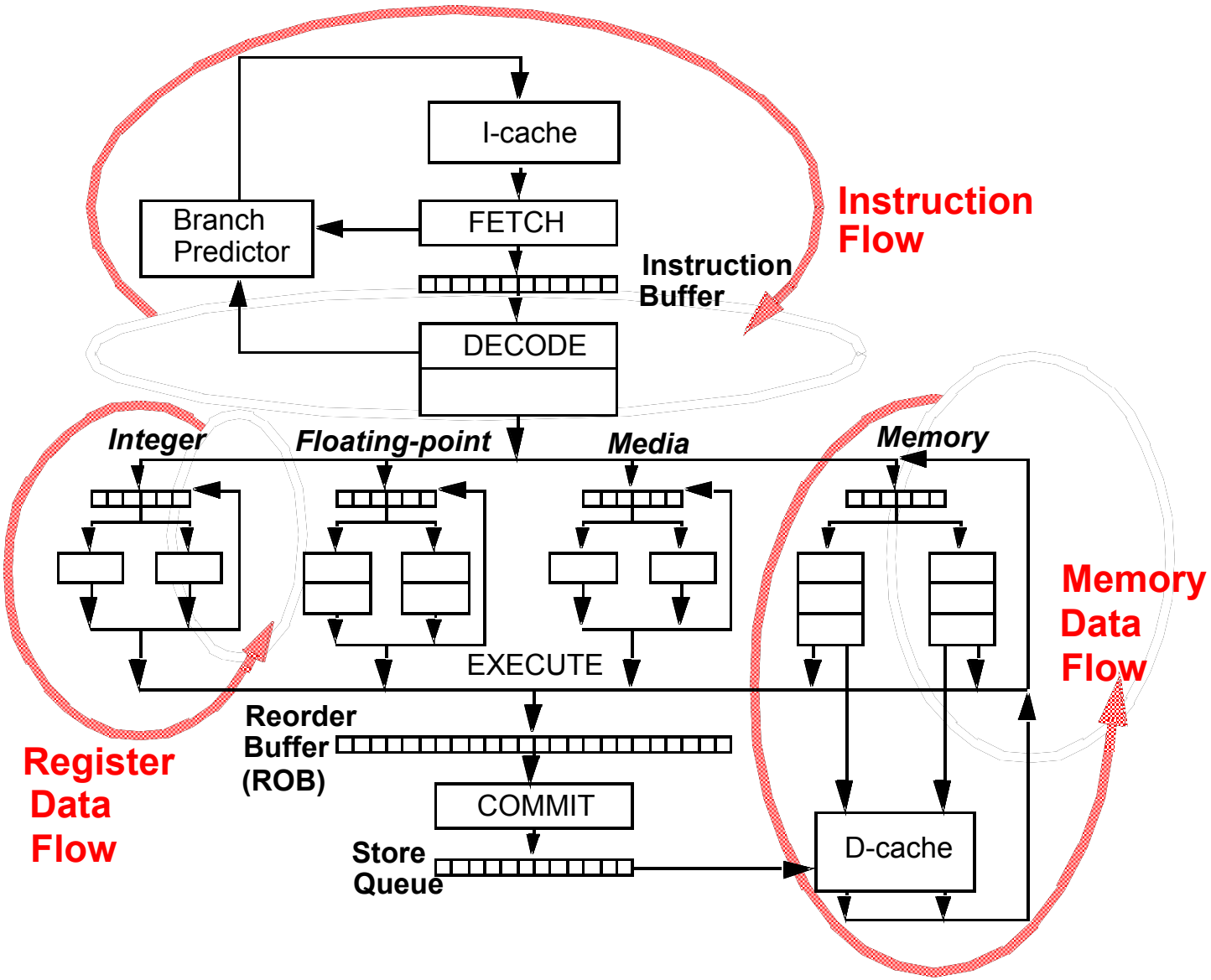
James C. Hoe
Dept of ECE, CMU
September 24, 2001

Reading Assignments: S&L Ch3 82-107, MJ Ch4 and Ch5, paper below

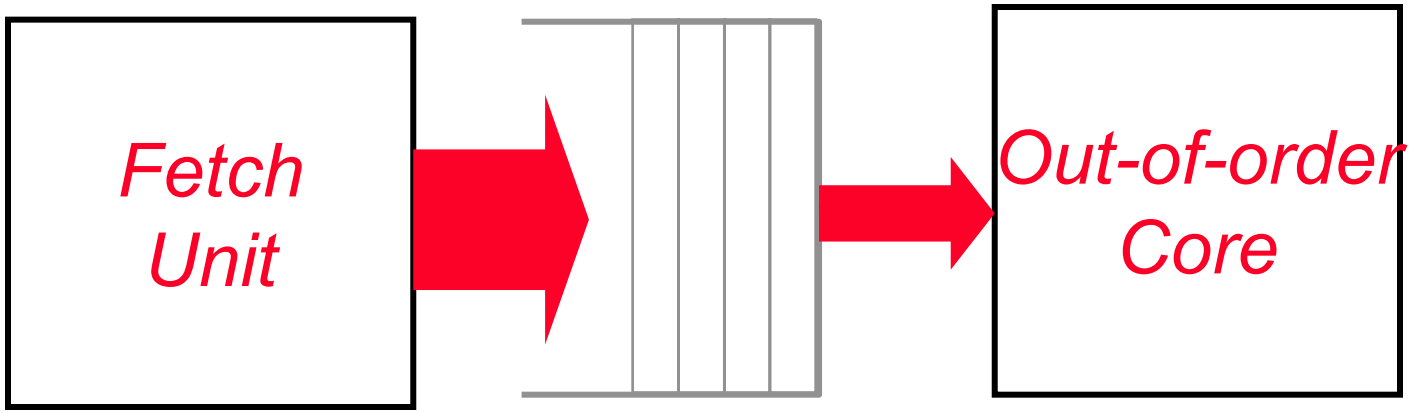
Announcements: Exam 1 on October 15th

Handouts: *"Combining Branch Predictors",* Scott McFarling

Flow Path Model of Superscalars



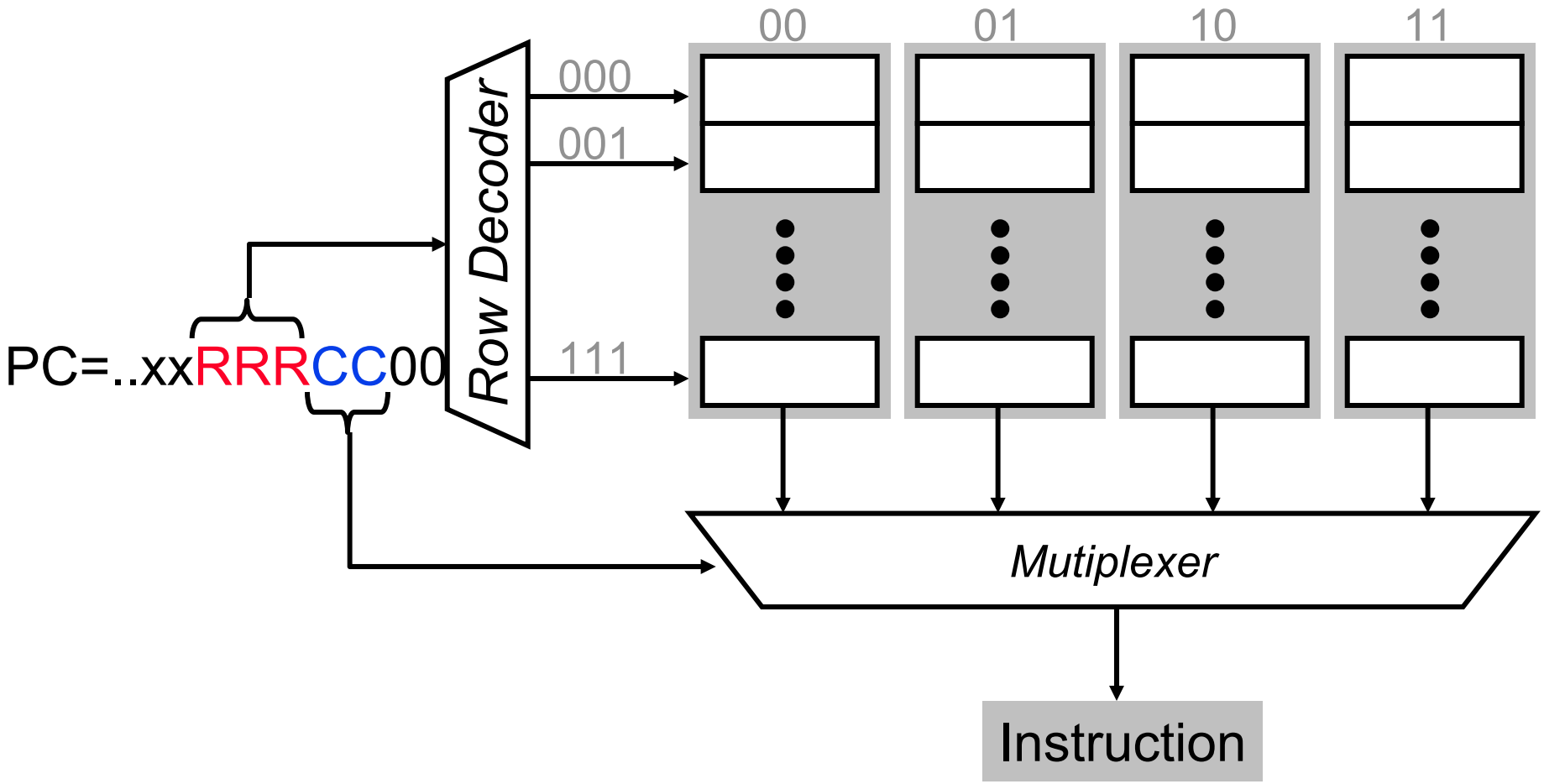
Instruction Fetch Buffer



- Fetch buffer smoothes out the rate mismatch between fetch and execution
  - neither the fetch bandwidth nor the execution bandwidth is consistent
- Fetch bandwidth should be higher than execution bandwidth
  - we prefer to have a stockpile of instructions in the buffer to hide cache miss latencies. *This requires both raw cache bandwidth + control flow speculation*

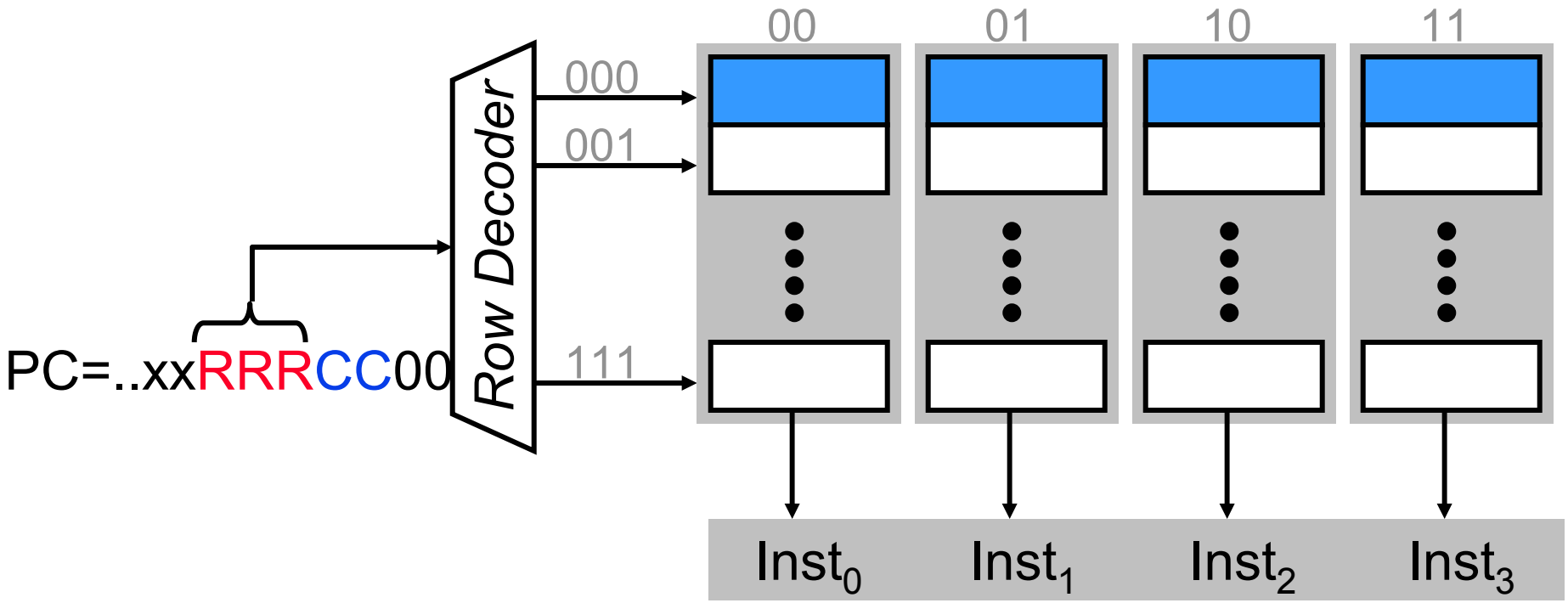
Instruction Flow Bandwidth

Instruction Cache Basic

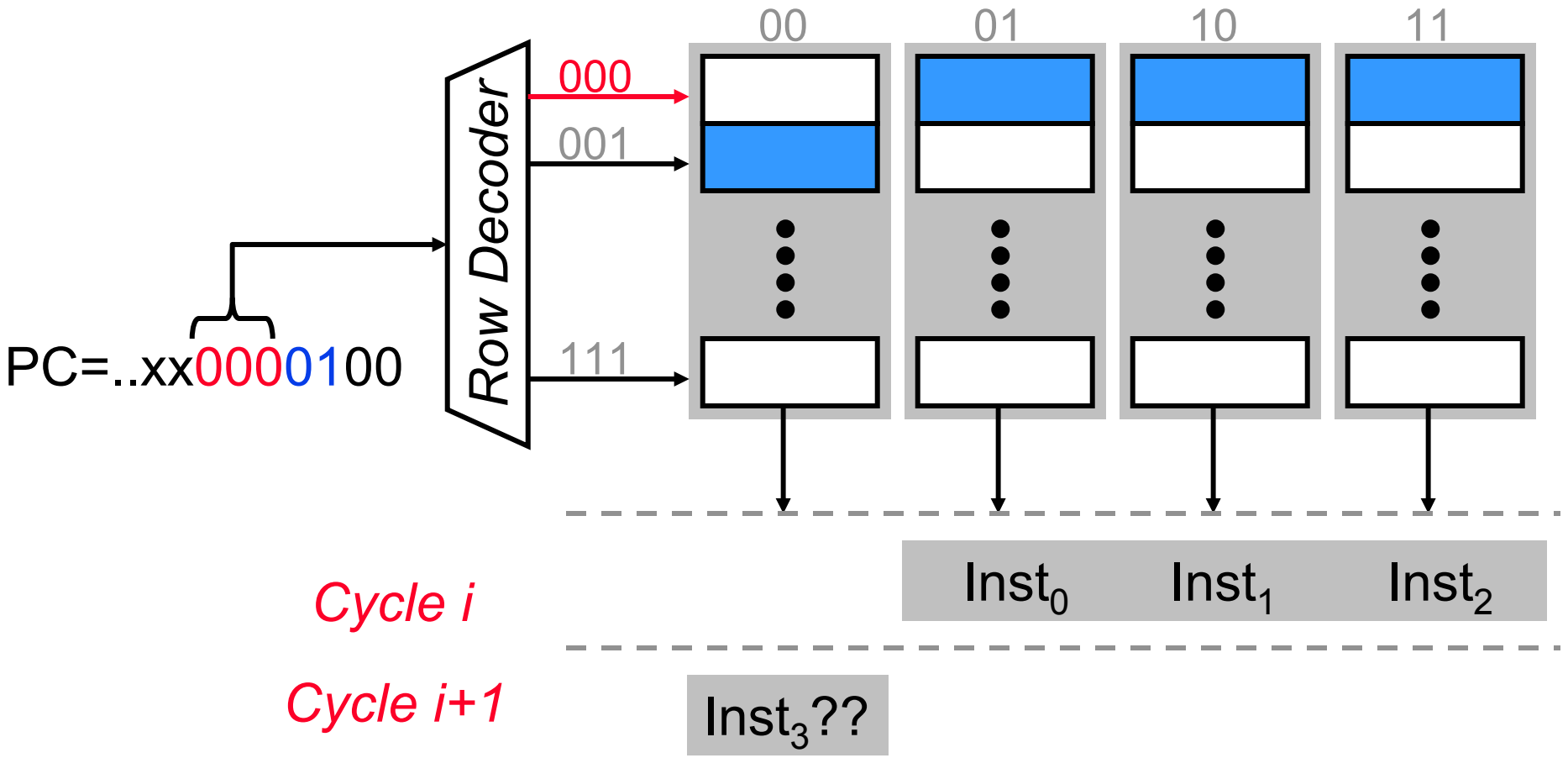


example: 4 instructions per cache line

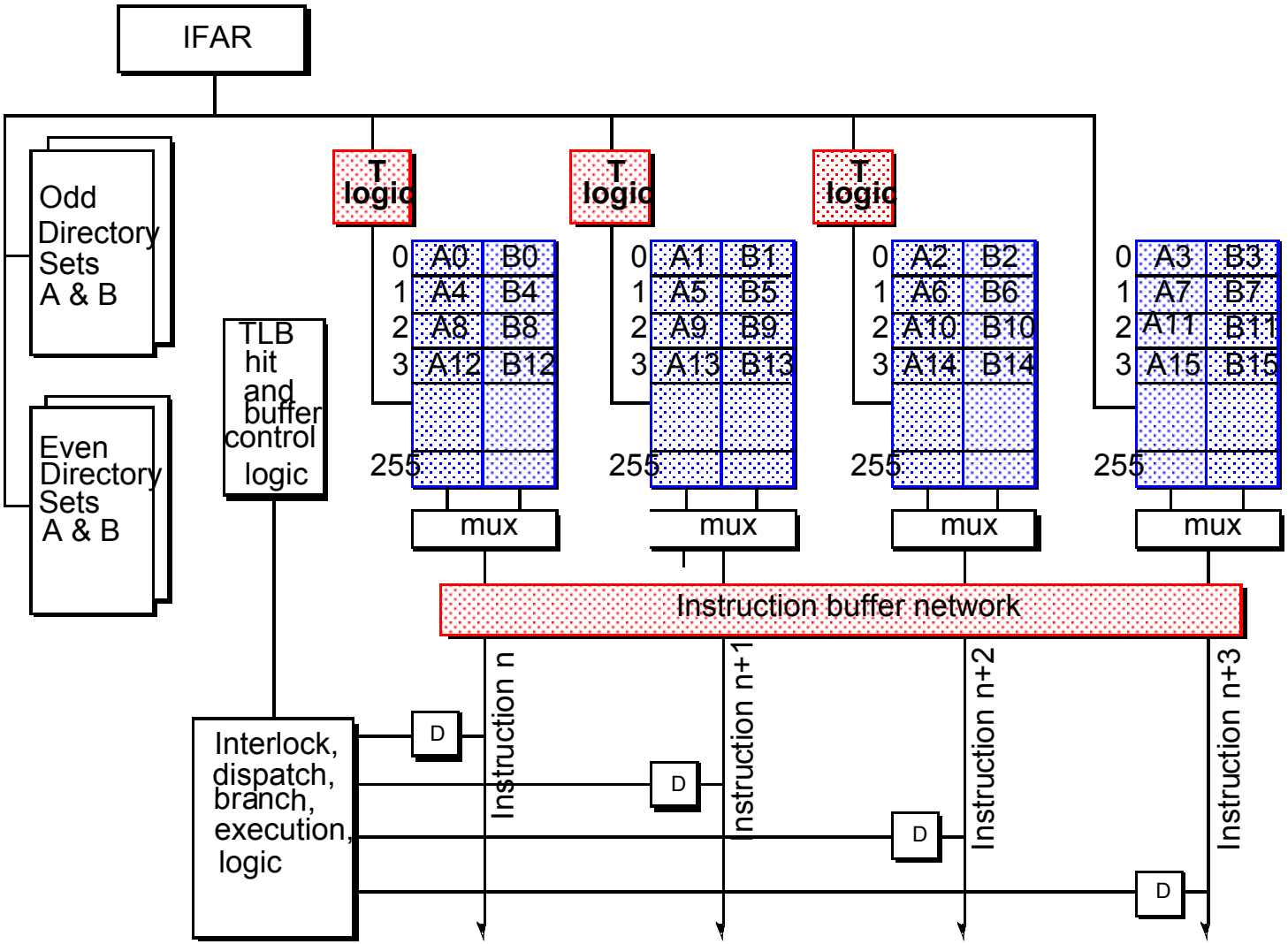
Spatial Locality and Fetch Bandwidth



Fetch Group Miss Alignment



IBM RS/6000 Auto-alignment

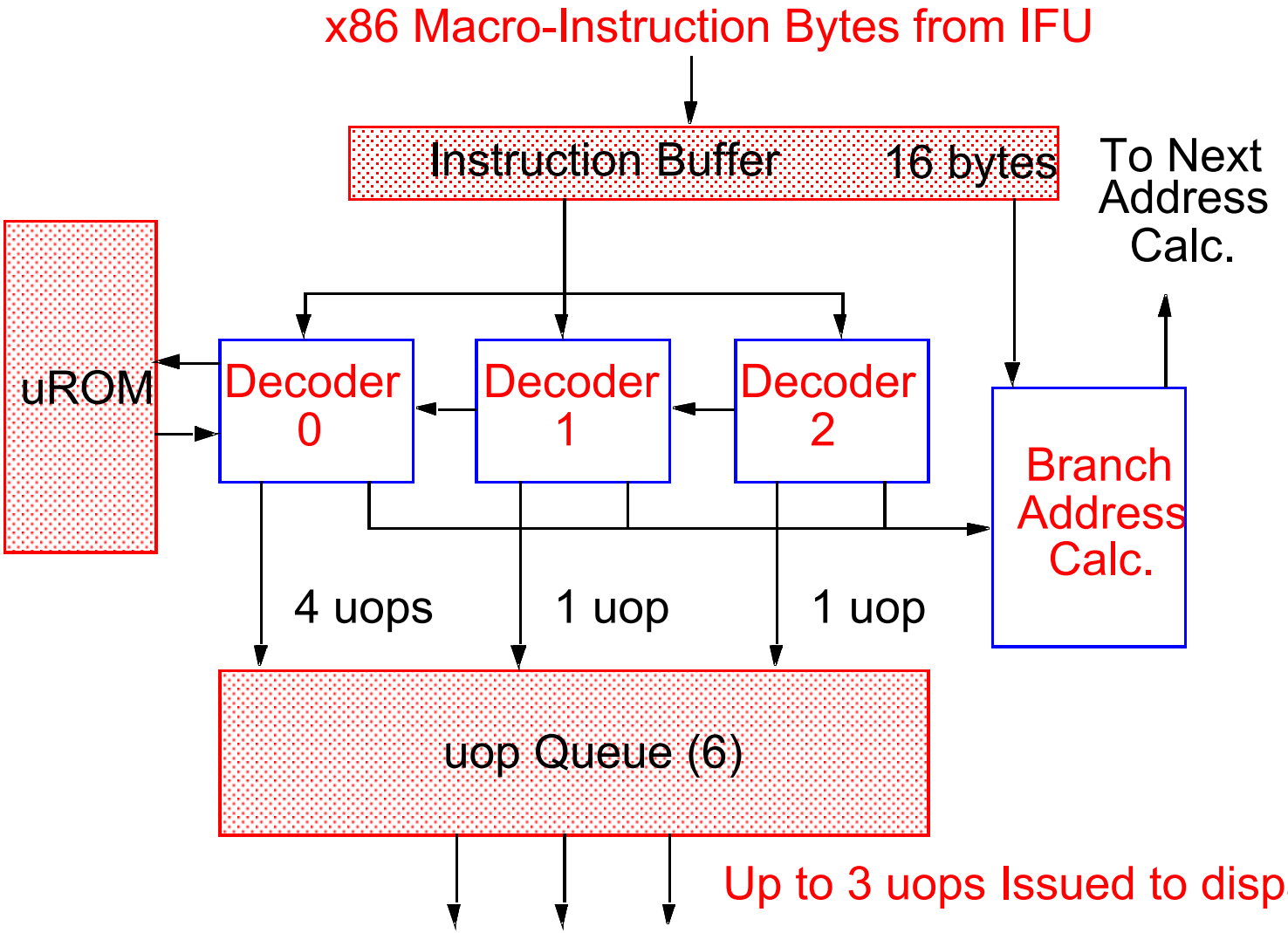


- *2-way set associative I-Cache, 8 256-inst SRAM modules*
- *16 instruction per cache line (**What is a cache line?)*

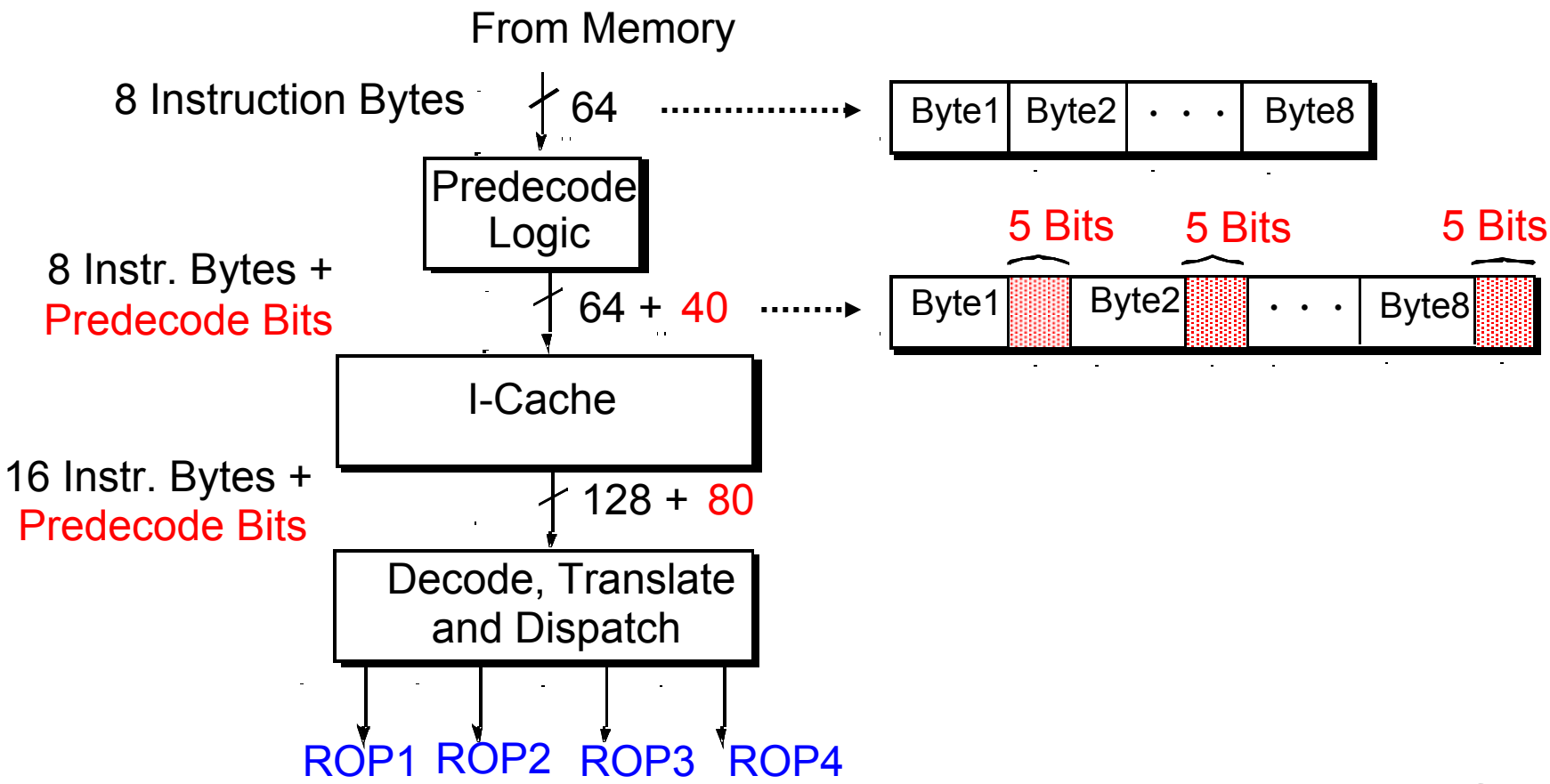
Instruction Decoding Issues

- Primary tasks:
  - Identify individual instructions
  - Determine instruction types
  - Detect inter-instruction dependences
- Two important factors:
  - Instruction set architecture
  - Width of parallel pipeline

Intel Pentium Pro Fetch/Decode Unit



Predecoding in the AMD K5



Predecoding is also useful for RISC ISAs!!

Cost: cache size, refill time

Control Dependence

IBM's Experience on Pipelined Processors
[Agerwala and Cocke 1987]

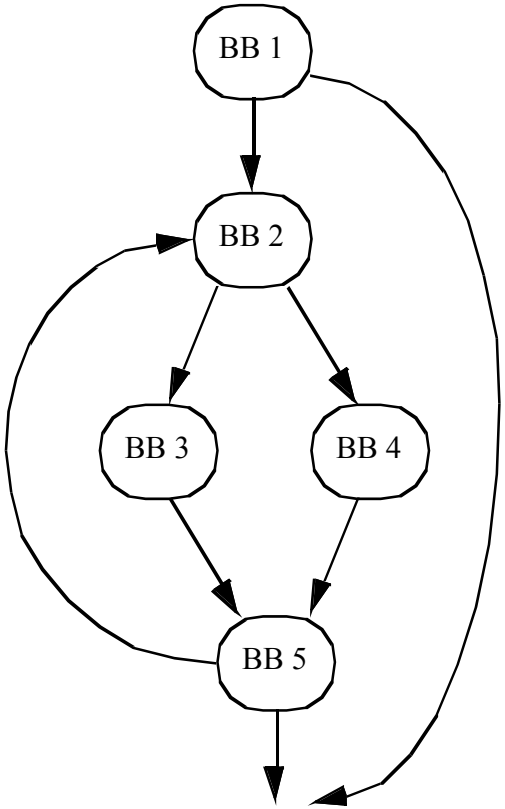
- Code Characteristics (dynamic)
  - loads - 25%
  - stores - 15%
  - ALU/RR - 40%
  - branches - 20%
    - 1/3 unconditional (always taken)

      unconditional - 100% schedulable
    - 1/3 conditional taken
    - 1/3 conditional not taken

      conditional - 50% schedulable

Control Flow Graph

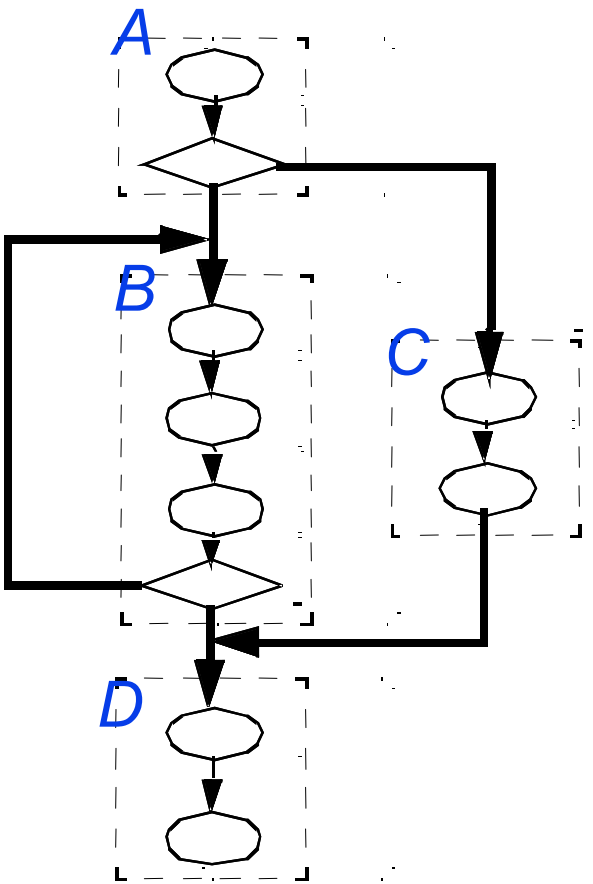
- Shows possible paths of control flow through basic blocks

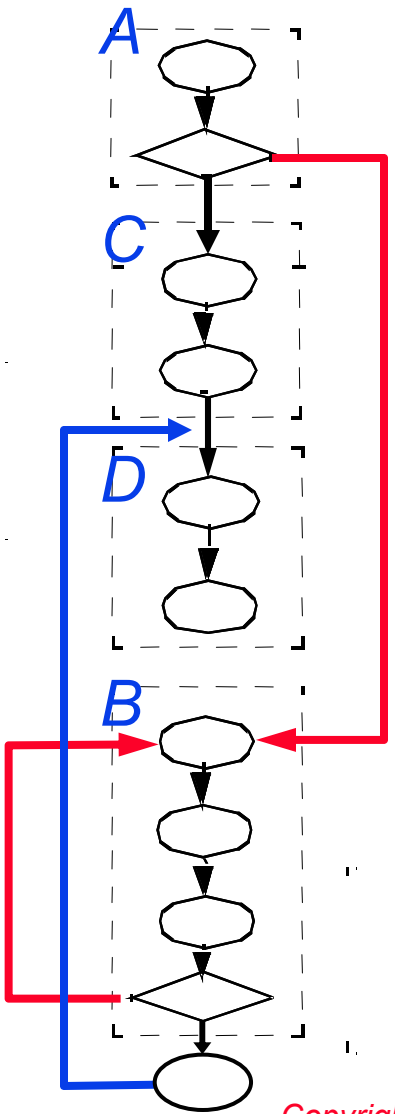


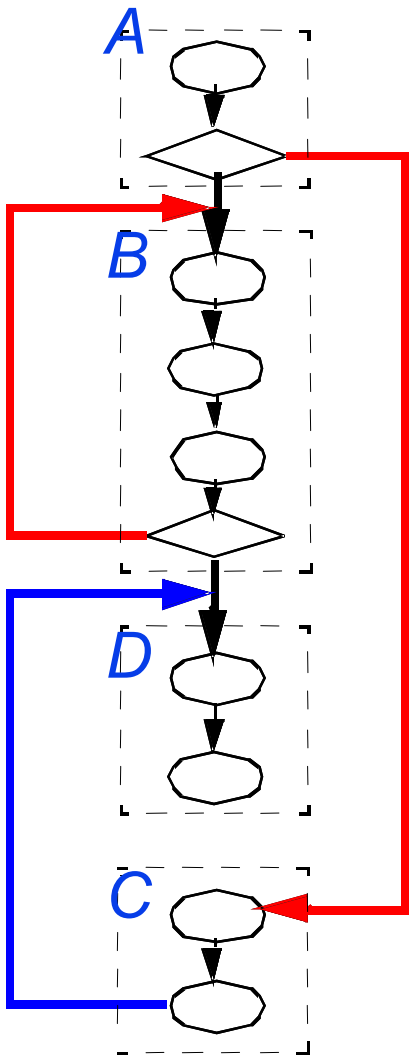

```
main:
        addi r2, r0, A      |
        addi r3, r0, B      |
        addi r4, r0, C      |   BB 1
        addi r5, r0, N      |
        add  r10,r0, r0     |
        bge  r10,r5, end    |
loop:                       |
        lw   r20, 0(r2)     |
        lw   r21, 0(r3)     |   BB 2
        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 Dependence
  - Node *X* is control dependant on Node *Y* if the computation in *Y* determines whether *X* executes

Mapping CFG to Linear Instruction Sequence





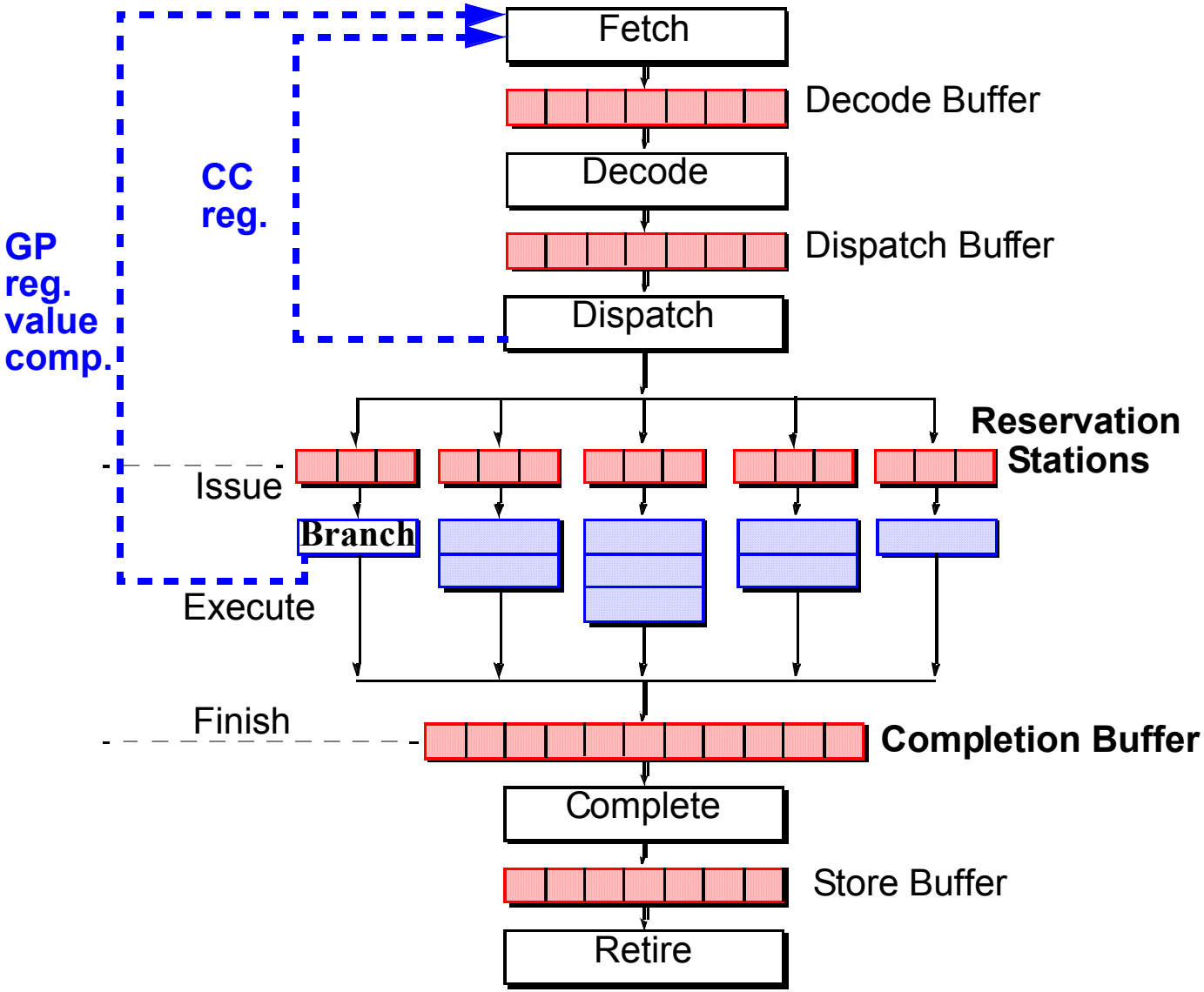


Branch Types and Implementation

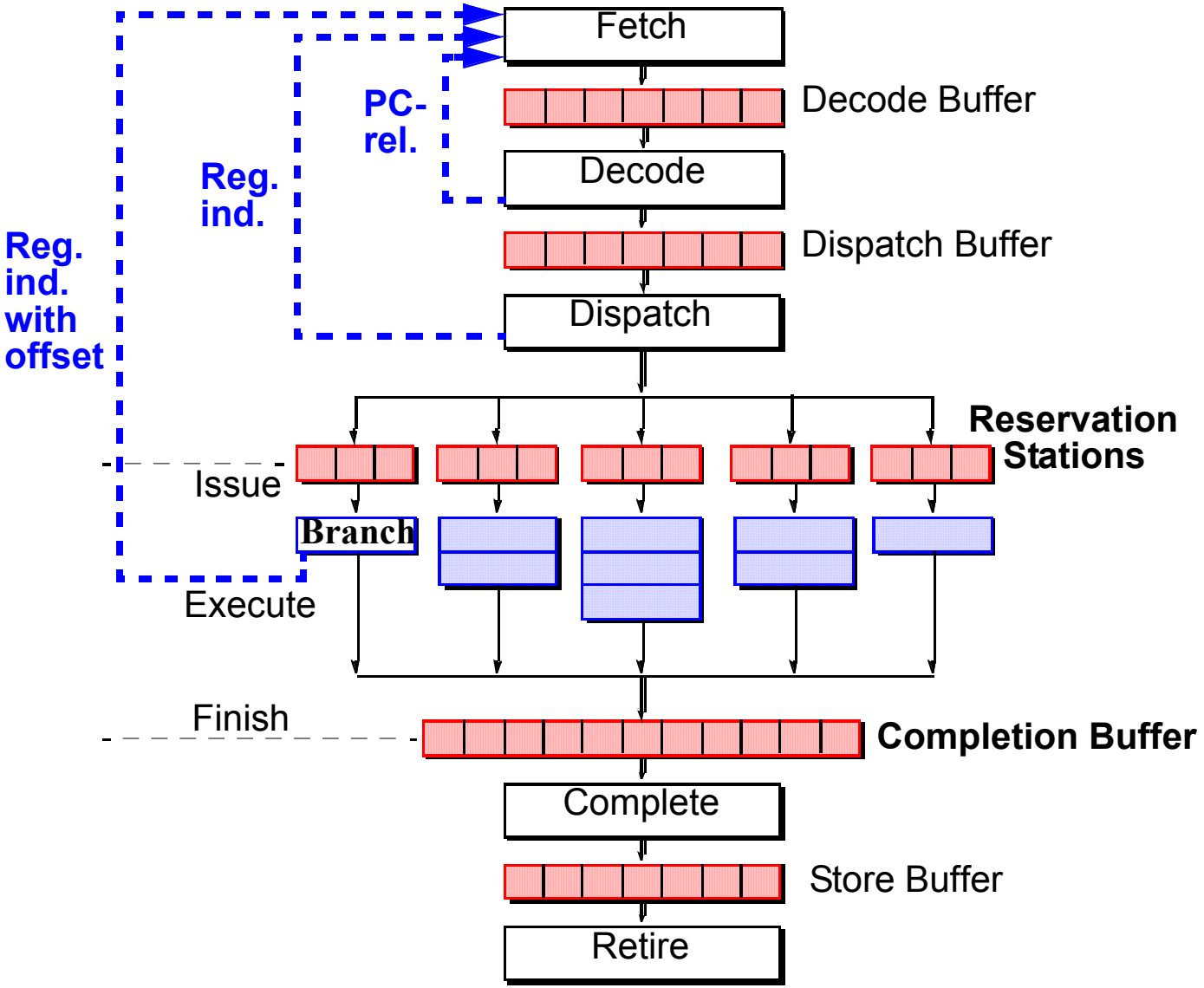
- Types of Branches
  - Conditional or Unconditional?
  - Subroutine Call (aka Link), needs to save PC?
  - How is the branch target computed?
    - Static Target — e.g. immediate, PC-relative
    - Dynamic targets — e.g. register indirect

- Conditional Branch Architectures
  - Condition Code '*N-Z-C-V*' — *e.g. PowerPC*
  - General Purpose Register — *e.g. Alpha, MIPS*
  - Special Purposes register — *e.g. Power's Loop Count*

Condition Resolution



Target Address Generation



What's So Bad About Branches?

- Performance Penalties
  - Use up execution resources
  - Fragmentation of I-Cache lines
  - Disruption of sequential control flow
    - Need to determine branch direction (conditional branches)
    - Need to determine branch target

Robs instruction fetch bandwidth and ILP

Riseman and Foster's Study

- 7 benchmark programs on CDC-3600
- Assume infinite machine:
  - Infinite memory and instruction stack, register file, fxn units

  Consider only true dependency at data-flow limit
- If bounded to single basic block, i.e. no bypassing of branches $\Rightarrow$ maximum speedup is 1.72
- Suppose one can bypass conditional branches and jumps (i.e. assume the actual branch path is always known such that branches do not impede instruction execution)

Br. Bypassed:	*0*	*1*	*2*	*8*	*32*	*128*
Max Speedup:	*1.72*	*2.72*	*3.62*	*7.21*	*24.4*	*51.2*

Determining Branch Direction

Problem: Cannot fetch subsequent instructions until branch direction is determined

- Minimize penalty
  - Move the instruction that computes the branch condition away from branch *(ISA&compiler)*
- Make use of penalty
  - Bias for not-taken
  - Fill delay slots with useful/safe instructions *(ISA&compiler)*
  - Follow both paths of execution *(hardware)*
  - Predict branch direction *(hardware)*

Determining Branch Target

Problem: Cannot fetch subsequent instructions until branch target is determined

- Minimize delay
  - Generate branch target early in the pipeline
- Make use of delay
  - Bias for not taken
  - Predict branch target

PC-relative vs Register Indirect targets

Branch Prediction

- Target Address Generation
  - Access register
    - PC, GP register, Link register
  - Perform calculation
    - +/- offset, auto incrementing/decrementing

  ⇒ Target Speculation

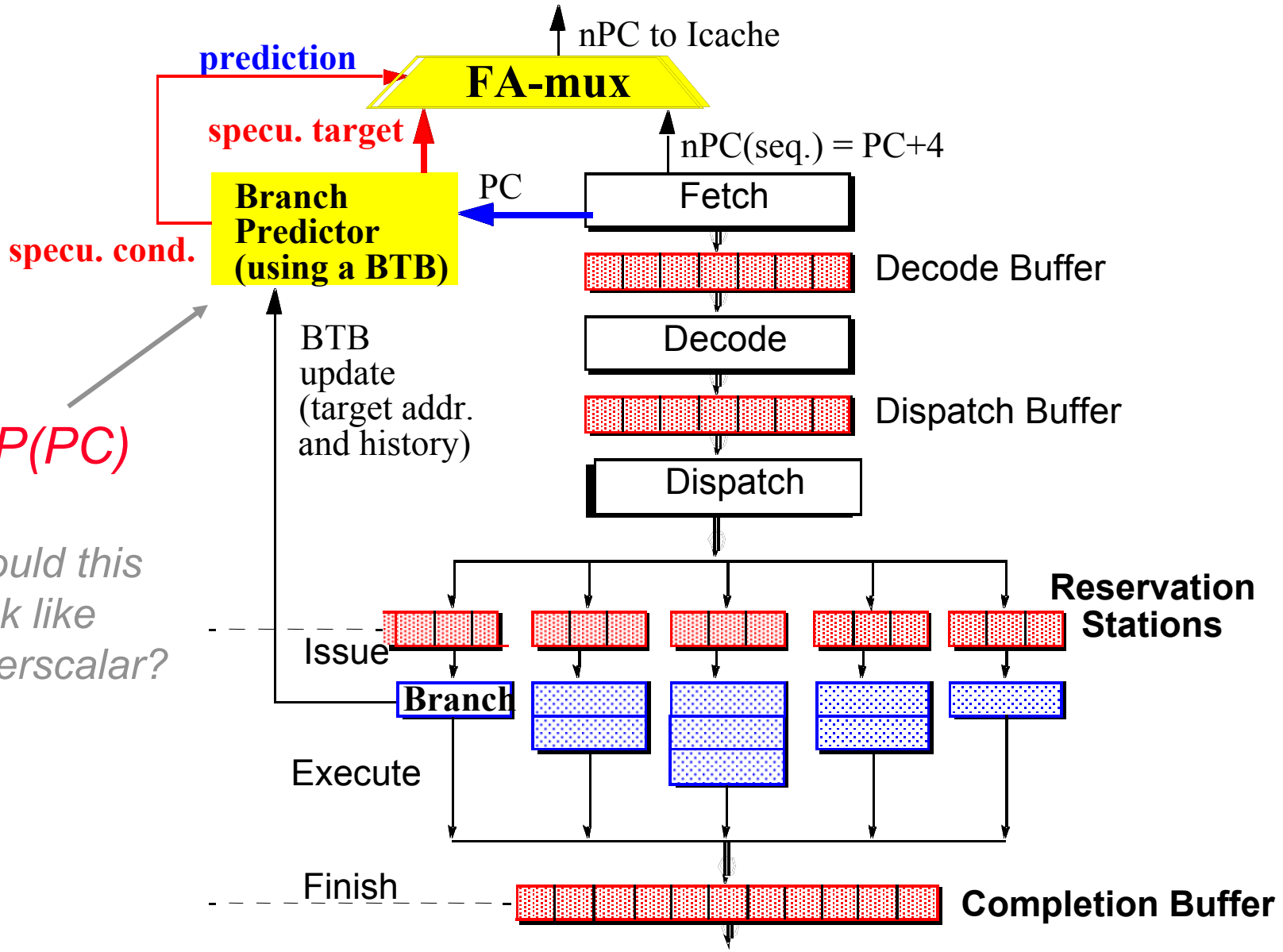
- Condition Resolution
  - Access register
    - Condition code register, data register, count register
  - Perform calculation
    - Comparison of data register(s)

  ⇒ Condition Speculation

Branch Condition Speculation

- Biased For Not Taken
  - Does not affect the instruction set architecture
  - Not effective in loops
- Software Prediction
  - Encode an extra bit in the branch instruction
    - Predict not taken: set bit to 0
    - Predict taken: set bit to 1
  - Bit set by compiler or user; can use profiling
  - Static prediction, same behavior every time
- Prediction Based on Branch Offsets
  - Positive offset: predict not taken
  - Negative offset: predict taken
- Prediction Based on History

Branch Instruction Speculation



nPC=BP(PC)

What should this really look like for a superscalar?

Branch Target Buffer (BTB)

- A small "cache-like" memory in the instruction fetch stage

- Remembers previously executed branches, their addresses, information to aid prediction, and most recent target addresses
- Instruction fetch stage compares current PC against those in BTB to "guess" nPC
  - *If* matched *then* prediction is made *else* nPC=PC+4
  - *If* predict taken *then* nPC=target address in BTB *else* nPC=PC+4
- When branch is actually resolved, BTB is updated

UCB Study [Lee and Smith, 1984]

- Benchmarks
  - 26 programs (traces on IBM 370, DEC PDP-11, CDC 6400)
  - Use trace-driven simulation with parameterized machine models
- Branch types
  - Unconditional: always taken or always not taken
  - Subroutine call: always taken
  - Loop control: usually taken (loop back)
  - Decision: either way, e.g. IF-THEN-ELSE
  - Computed GOTO: always taken, with changing target
  - Supervisor call: always taken
  - "Execute": always taken (IBM 370)
- Branch behavior: *Taken vs Not Taken*

	IBM1	IBM2	IBM3	IBM4	DEC	CDC	**Average**
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

- Based on opcode only (%)

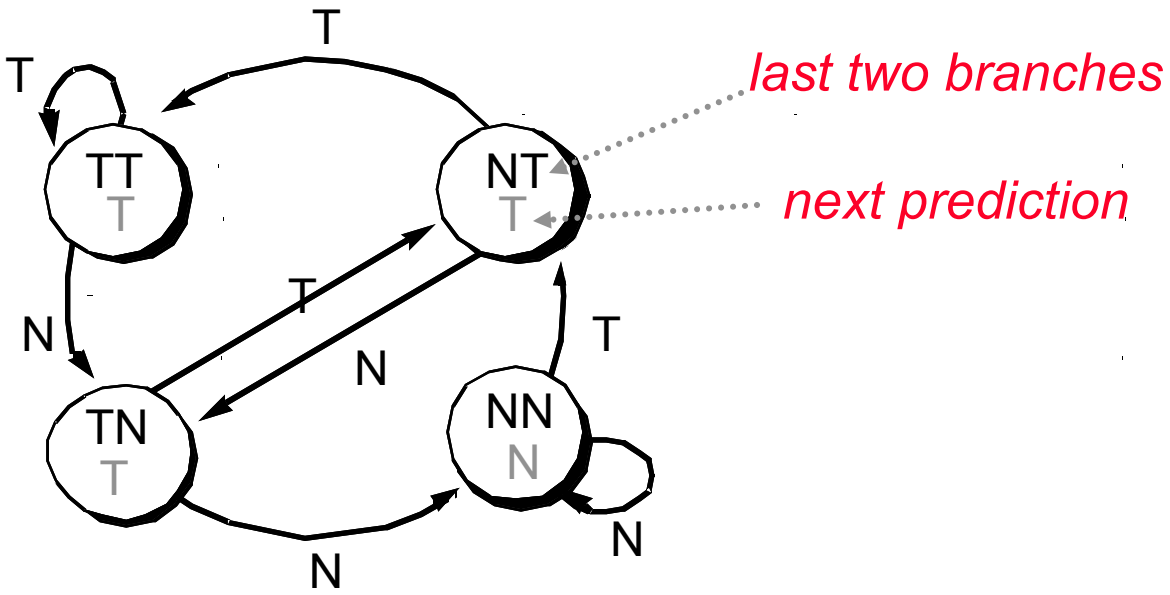
IBM1	IBM2	IBM3	IBM4	DEC	CDC
66	69	71	55	80	78

- Based on history of branch
  - Branch prediction function F (X1, X2, ...)
  - Use up to 5 previous branches for history (%)

	IBM1	IBM2	IBM3	IBM4	DEC	CDC
0	64.1	64.4	70.4	54.0	73.8	77.8
1	91.9	95.2	86.6	79.7	96.5	82.3
2	93.3	96.5	90.8	83.4	97.5	90.6
3	93.7	96.7	91.2	83.5	97.7	93.5
4	94.5	97.0	92.0	83.7	98.1	95.3
5	94.7	97.1	92.2	83.9	98.2	95.7

Example Prediction Algorithm

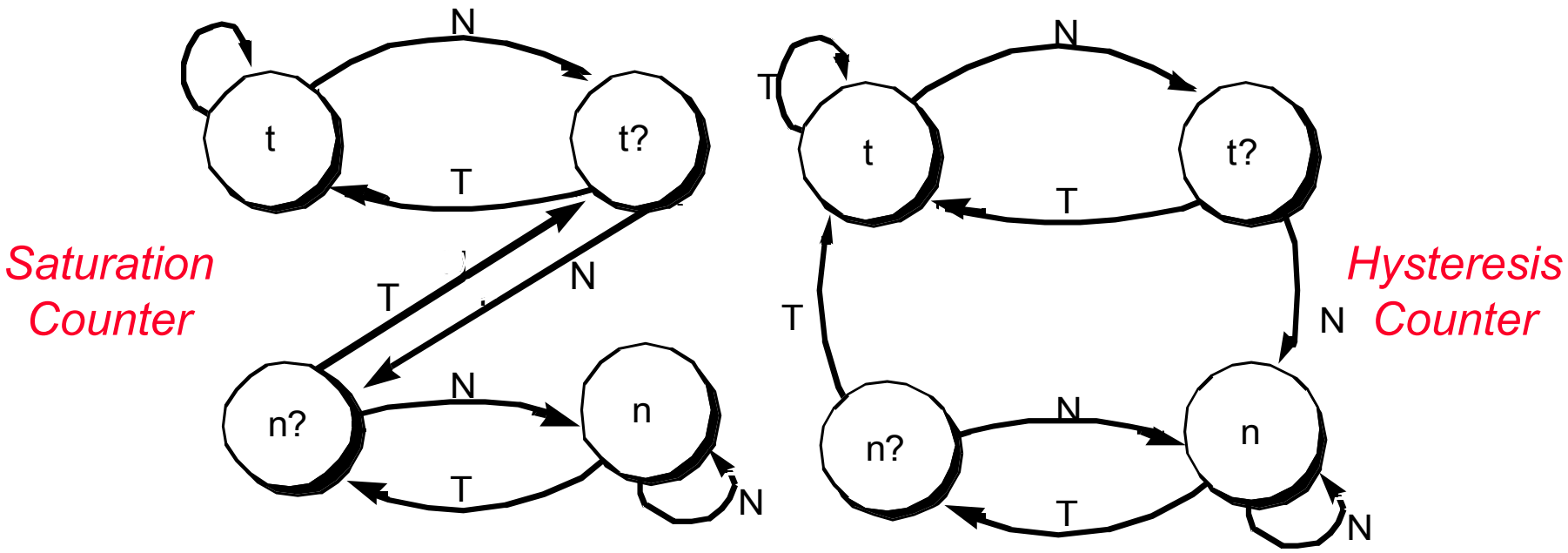
- Prediction accuracy approaches maximum with as few as 2 preceding branch occurrences used as history



Results (%)

IBM1	IBM2	IBM3	IBM4	DEC	CDC
93.3	96.5	90.8	83.4	97.5	90.6

Other Prediction Algorithms



- Combining prediction accuracy with BTB hit rate (86.5% for 128 sets of 4 entries each), branch prediction can provide the net prediction accuracy of approximately 80%. This implies a 5-20% performance enhancement.

IBM RS/6000 Study [Nair, 1992]

- Five different branch types
  - b: unconditional branch
  - bl: branch and link (subroutine calls)
  - bc: conditional branch
  - bcr: conditional branch using link register (subroutine returns)
  - bcc: conditional branch using count register (system calls)
- Separate branch function unit to overlap of branch instructions with other instructions
- Two causes for branch stalls
  - Unresolved conditions
  - Branches downstream too close to unresolved branches

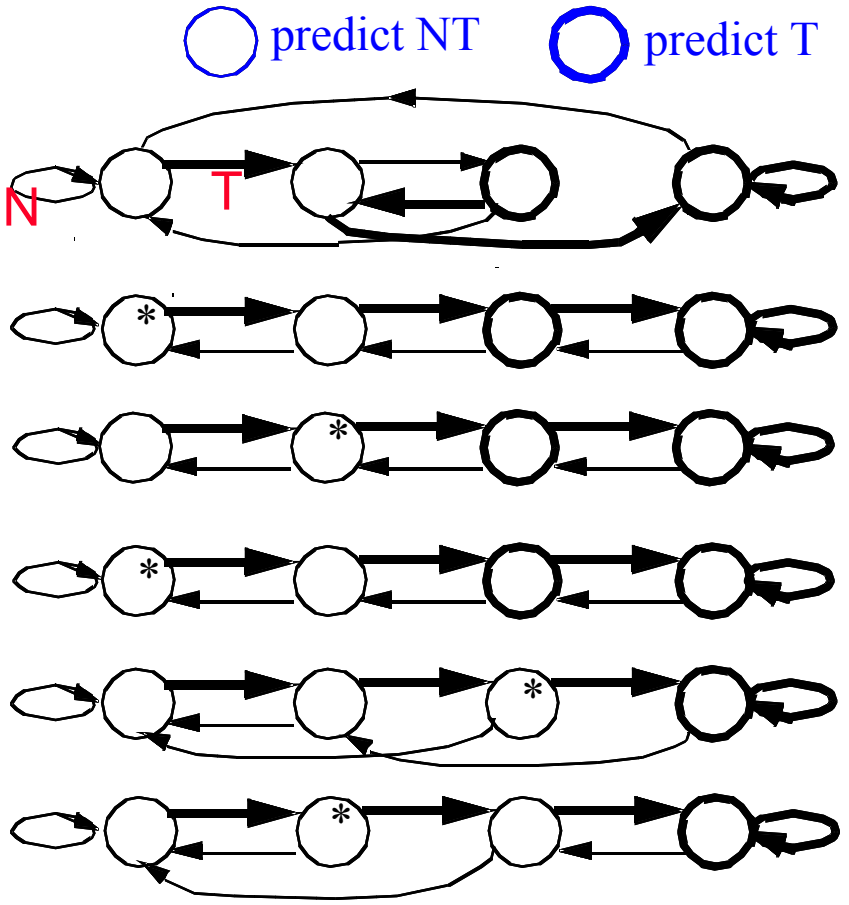
Branch Instruction Distribution

	% of diff. types of branch instructions:				% of bc inst. with penalty cycles:		
Benchmark	b	bl	bc	bcr	3 cyc.	2 cyc.	1 cyc.
spice2g6	7.86	0.30	12.58	0.32	13.82	3.12	0.76
doduc	1.00	0.94	8.22	1.01	10.14	1.76	2.02
matrix300	0.00	0.00	14.50	0.00	0.68	0.22	0.20
tomcatv	0.00	0.00	6.10	0.00	0.24	0.02	0.01
gcc	2.30	1.32	15.50	1.81	22.46	9.48	4.85
espresso	3.61	0.58	19.85	0.68	37.37	1.77	0.31
li	2.41	1.92	14.36	1.91	31.55	3.44	1.37
eqntott	0.91	0.47	32.87	0.51	5.01	11.01	0.80

Exhaustive Search for Optimal Predictors

- There are 2^{20} possible state machines of 2-bit predictors
- Pruning uninteresting and redundant machines leaves 5248
- It is possible to exhaustively search and find the *optimal* predictor for a benchmark

Benchmark	Best Pred. %
spice2g6	97.2
doduc	94.3
gcc	89.1
espresso	89.1
li	87.1
eqntott	87.9



Saturation Counter is near optimal in all cases!

Number of Counter Bits Needed

Benchmark	Prediction Accuracy (Overall CPI Overhead)			
	3-bit	2-bit	1-bit	0-bit
spice2g6	97.0 (0.009)	97.0 (0.009)	96.2 (0.013)	76.6 (0.031)
doduc	94.2 (0.003)	94.3 (0.003)	90.2 (0.004)	69.2 (0.022)
gcc	89.7 (0.025)	89.1 (0.026)	86.0 (0.033)	50.0 (0.128)
espresso	89.5 (0.045)	89.1 (0.047)	87.2 (0.054)	58.5 (0.176)
li	88.3 (0.042)	86.8 (0.048)	82.5 (0.063)	62.4 (0.142)
eqntott	89.3 (0.028)	87.2 (0.033)	82.9 (0.046)	78.4 (0.049)

- Branch history table size: Direct-mapped array of 2k entries
- Programs, like gcc, can have over 7000 conditional branches
- In collisions, multiple branches share the same predictor
- Variation of branch penalty with branch history table size level out at 1024