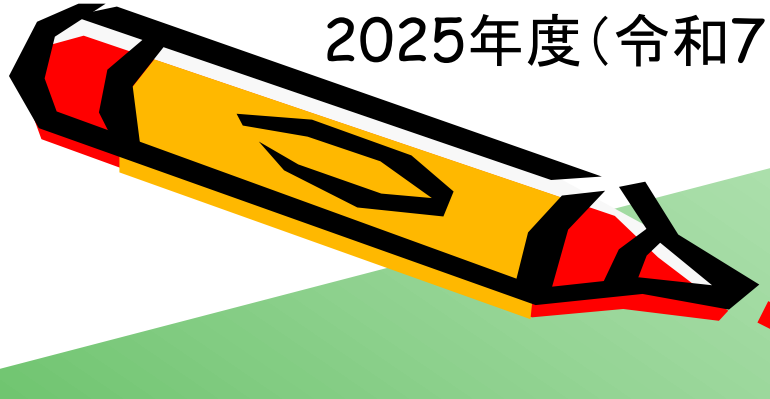


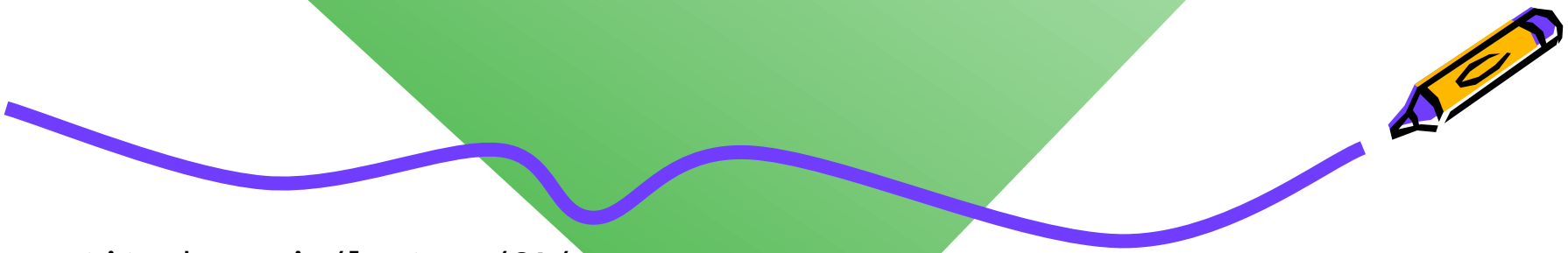
2025年度（令和7年）版

Ver. 2025-10-21a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

6. 分岐予測
Branch Prediction

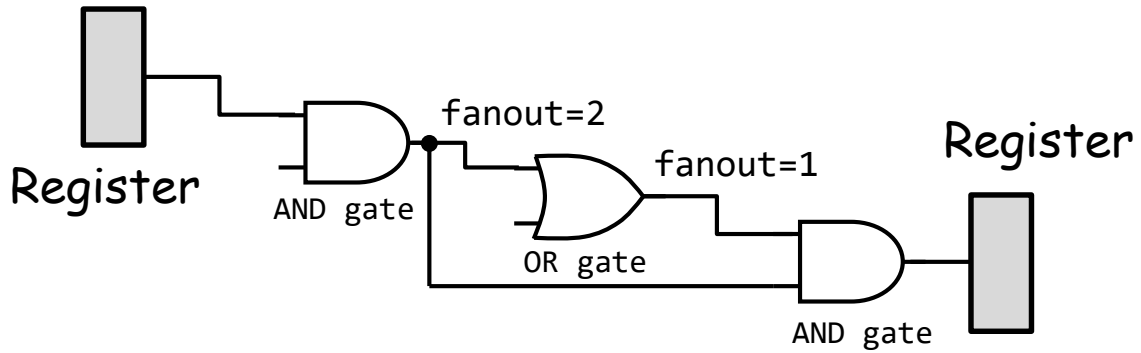


www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

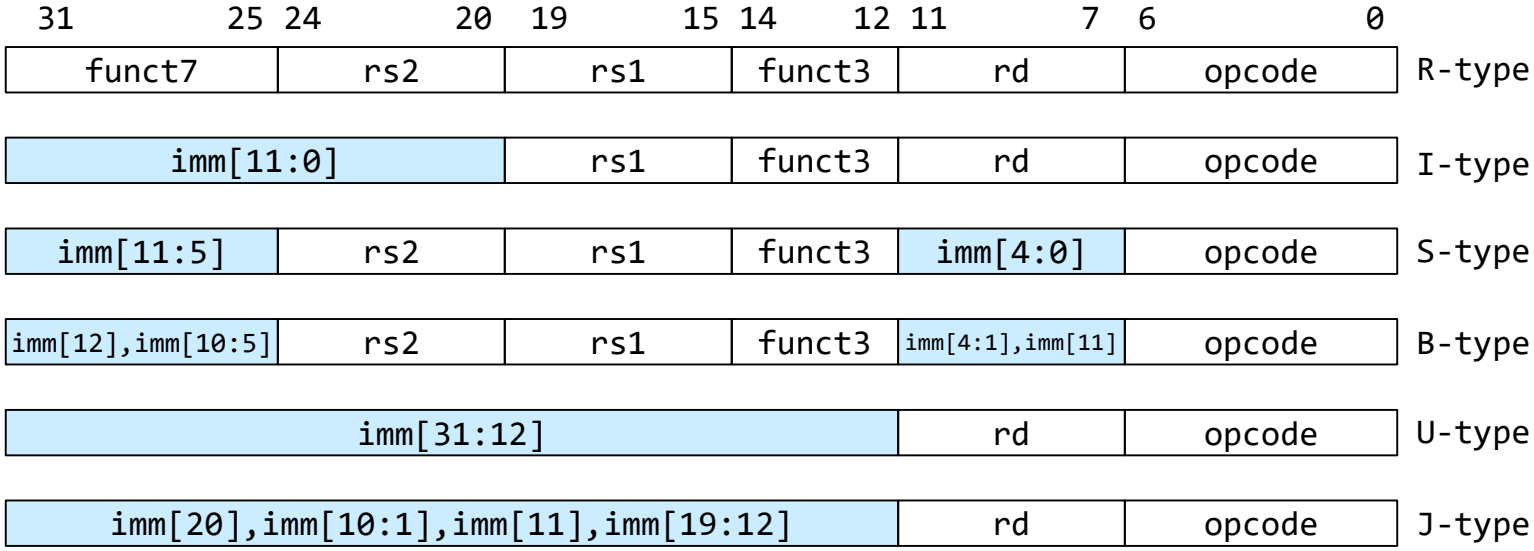
吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Clock rate is mainly determined by

- Switching speed of gates (transistors)
- The number of levels of gates
  - The maximum number of gates cascaded in series in any combinational logics.
  - In this example, the number of levels of gates is 3.
- Wiring delay and fanout
- The slowest of all paths is called the critical path



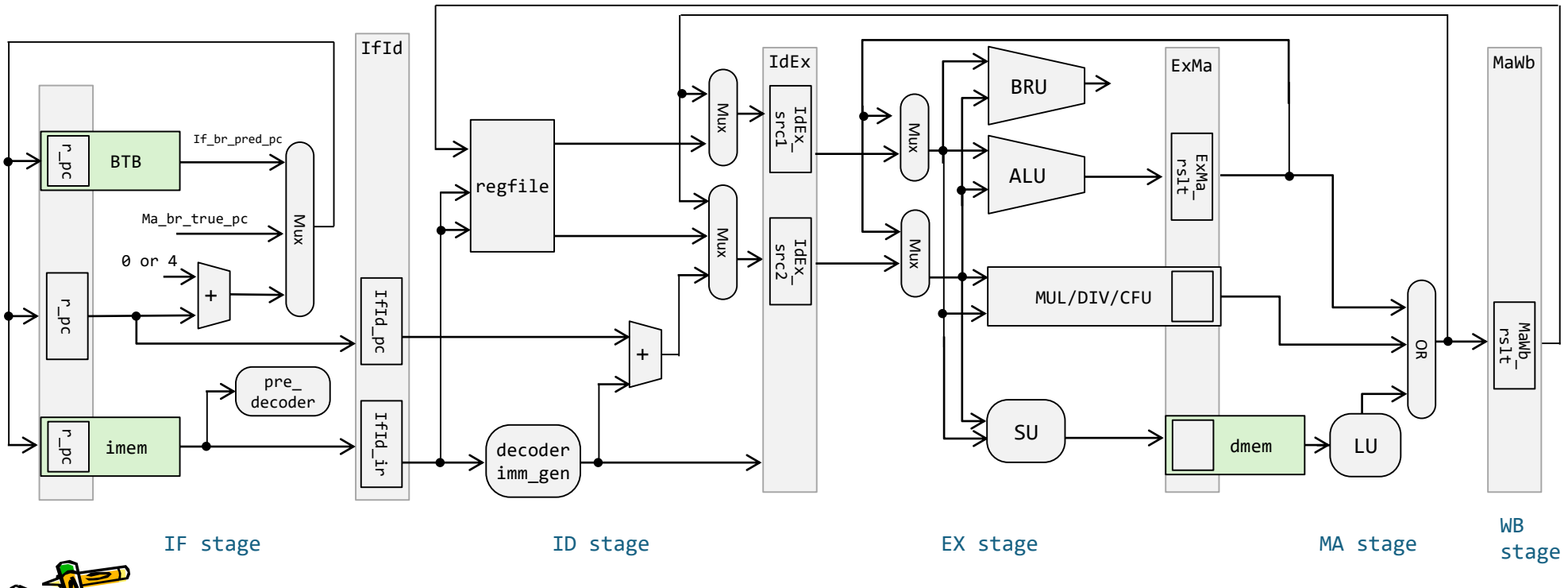
RISC-V RV32I instruction format



opcode5[2:0] opcode5[4:3]	000	001	010	011	100	101	110
00	LOAD (I-type)	LOAD-FP	custom-0	MISC-MEM (I-type)	OP-IMM (I-type)	AUIPC (U-type)	OP-IMM-32
01	STORE (S-type)	STORE-FP	custom-1	AMO	OP (R-type)	LUI (U-type)	OP-32
10	MADD	MSUB	NMSUB	NMADD	OP-FP	OP-V	custom-2/ rv128
11	BRANCH (B-type)	JALR (I-type)	reserved	JAL (J-type)	SYSTEM (I-type)	reserved	custom-3/ rv128

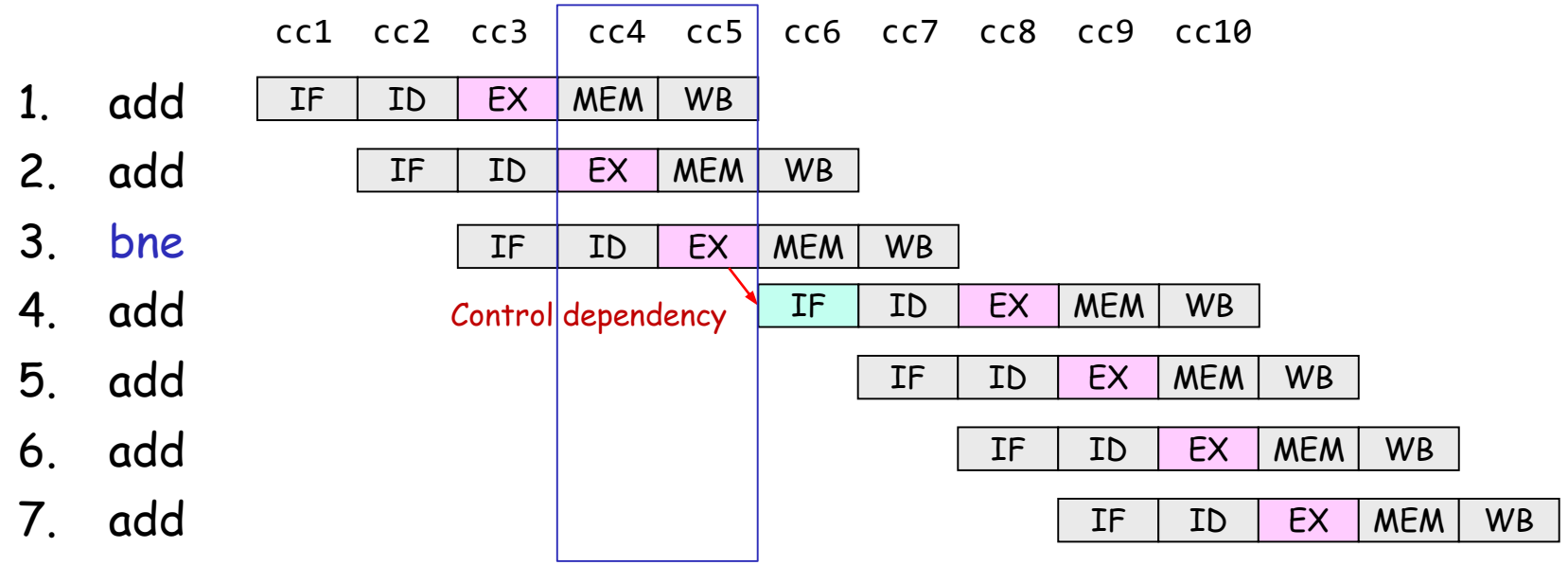
RVCpu: 5-stage pipelining processor

- IF (Instruction Fetch) メモリから命令をフェッチする。
- ID (Instruction Decode) 命令をデコード（解読）しながら、レジスタの値を読み出す。
- EX (Execution) 命令操作の実行またはアドレスの生成を行う。
- MA (Memory Access) 必要であれば、データ・メモリ中のオペランドにアクセスする。
- WB (Write Back) 必要であれば、結果をレジスタに書き込む。



Why do branch instructions degrade IPC?

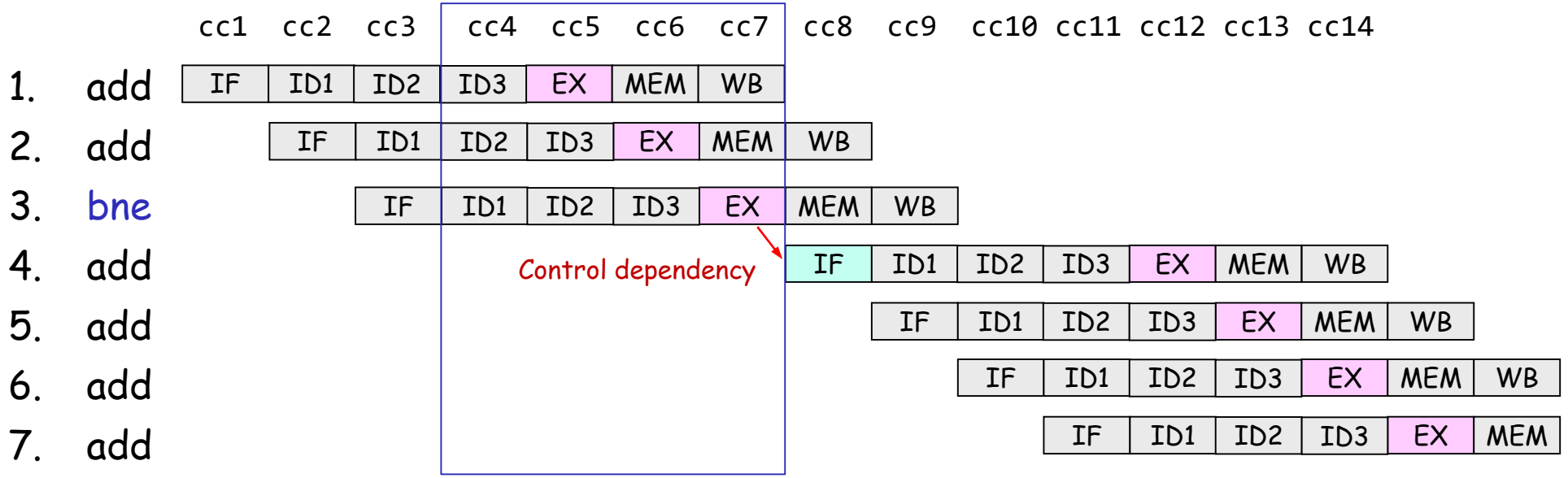
- The branch taken / untaken is determined in the execution stage of the branch.
- The conservative approach of stalling instruction fetch until the branch direction is determined.



five stages pipelined processor executing instruction sequence with a branch

Deeper pipeline

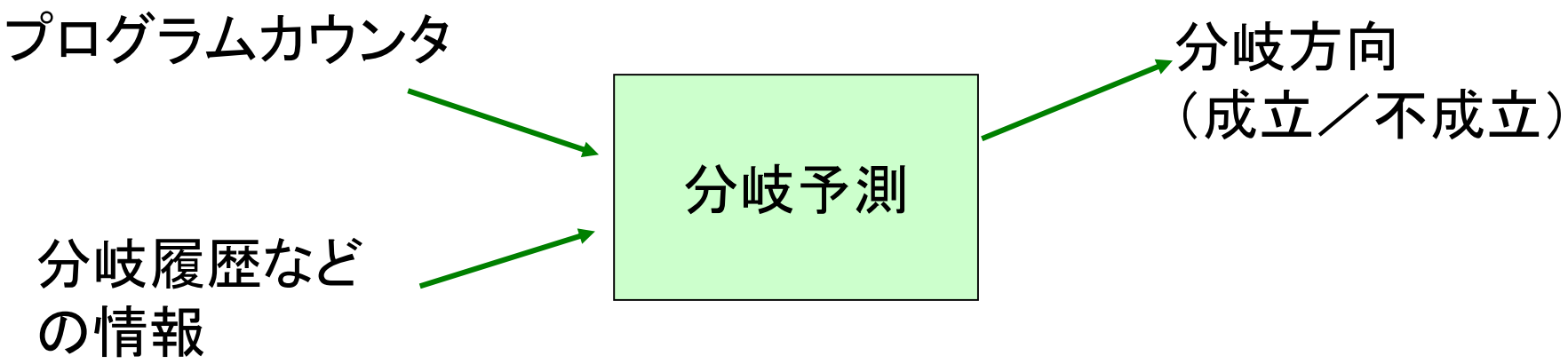
- In conservative approach, IPC degradation will be significant by deeper pipeline



高いバンド幅の命令フェッチ

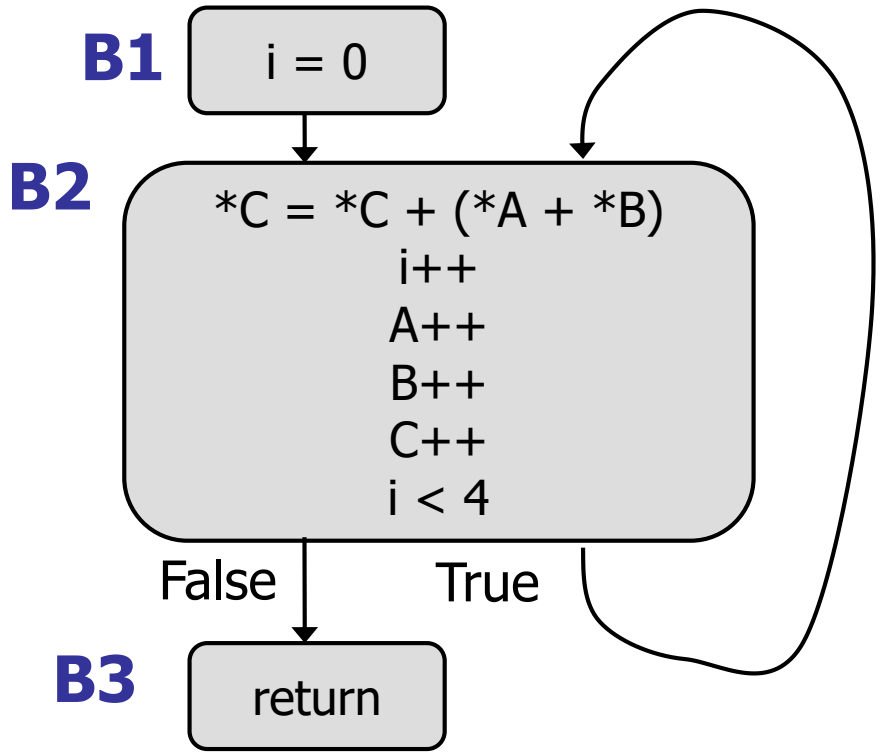
- パイプラインにバブルを生じさせないために，条件分岐命令をフェッチした時に次の3つを予測 (prediction) する.
  - フェッチしている命令が分岐命令か？
  - 分岐先アドレス(32bit)
  - 分岐方向
    - 分岐成立(Taken)か分岐不成立(Not taken, Untaken)か？
- Speculation assuming the prediction is true
  - No penalty by a branch instruction when the prediction hits
  - Recovery when the miss is detected

分岐方向の予測(分岐予測)



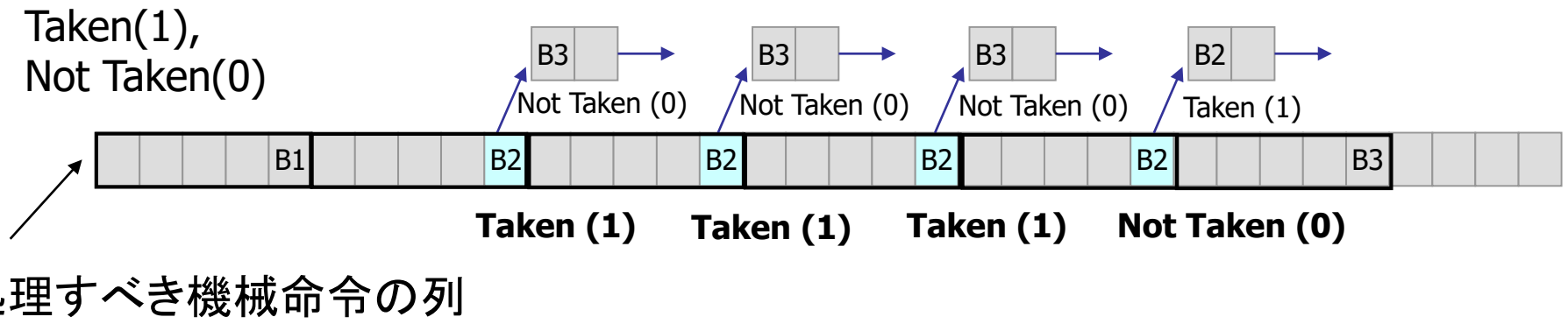
サンプルプログラム Vector Add

```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++)
    C[i] += (A[i] + B[i]);
}
```

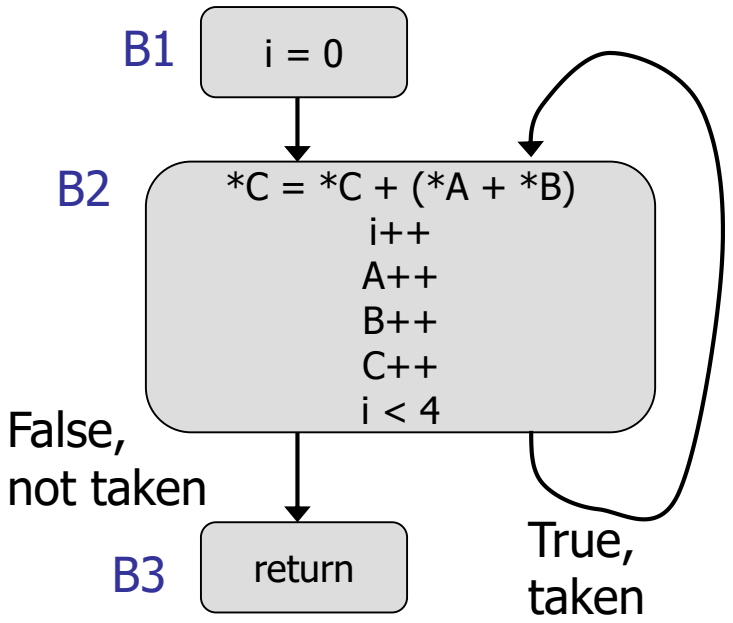


制御フローグラフ

シンプルな分岐予測: Branch Always



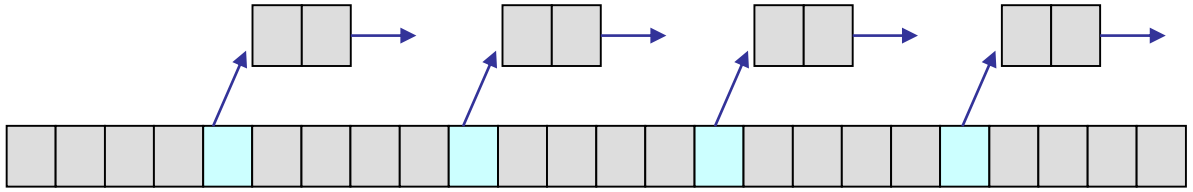
- Branch Always: 常に分岐が成立すると予測する.
  - 上の例では，予測成功率は75%，ミス率25%
  - 予測のためのメモリを必要としない.
  - 予測とよぶほどのものではない.



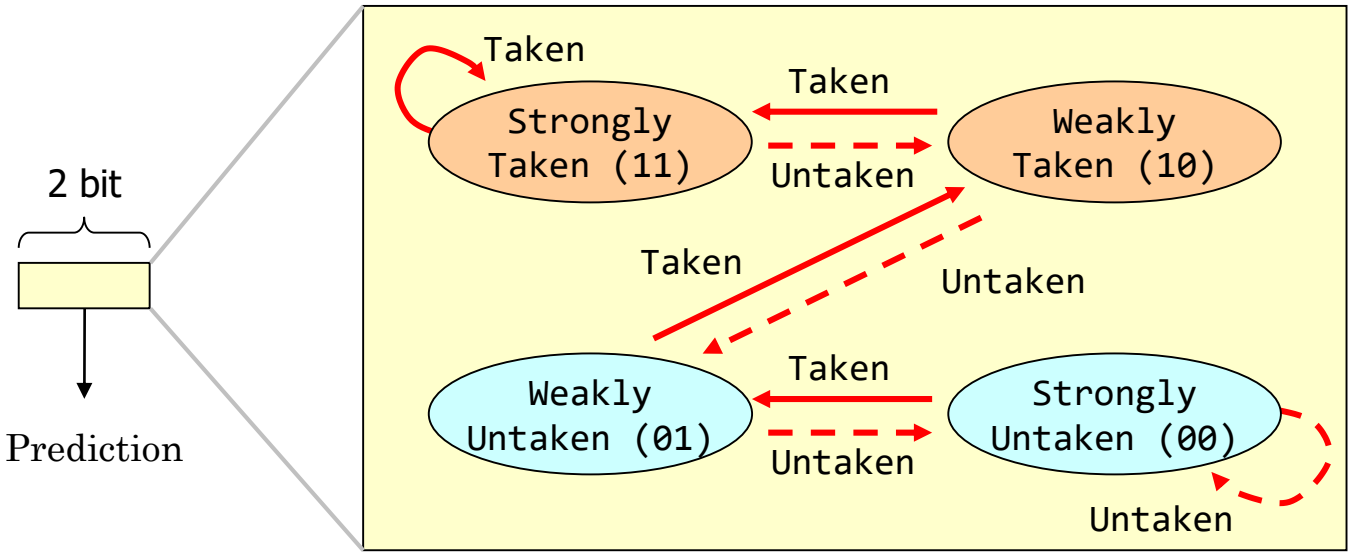
シンプルな分岐予測: 2ビットカウンタ方式

トレースデータ
（分岐アドレス，分岐結果）

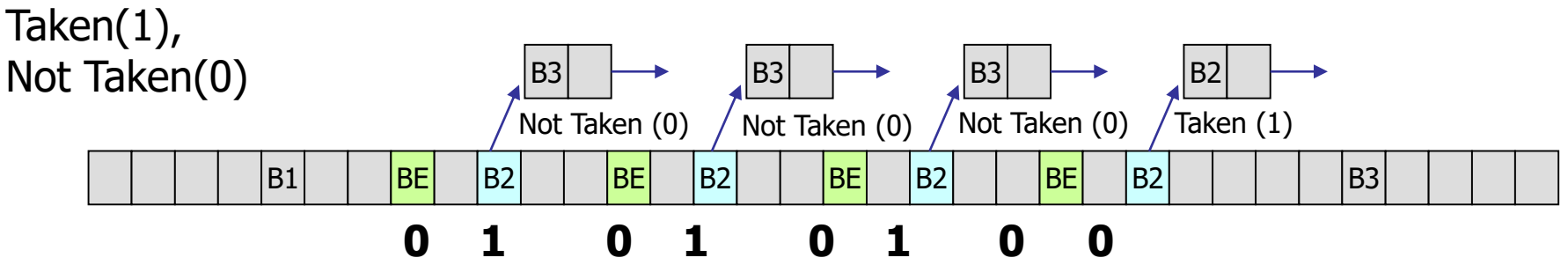
0040d6c5 1	0040d89c 1
0040d6b8 1	0040d89c 1
0040d6bc 0	0040d89c 1
0040d6c5 0	0040d89c 1
0040d6df 0	0040d89c 1
0040d71f 0	0040d89c 1
0040d736 0	0040d89c 0
0040d7ab 0	0040d8a2 0
0040d7cd 0	0040d8c0 1
0040d7f9 0	0040d8c4 0
0040d81e 1	0040d8cd 1
0040d7f9 1	0040d8c0 0
0040d81e 1	0040d8c4 1
0040d7f9 0	0040d8cd 1
0040d81e 0	0040d8c0 1
0040d83d 0	0040d8c4 0
0040d86d 1	0040d8cd 0
0040d86d 1	0040d8e7 0
0040d86d 1	0040d923 1
0040d86d 1	0040d7ab 0
0040d86d 1	0040d7cd 0
0040d86d 1	0040d7f9 0



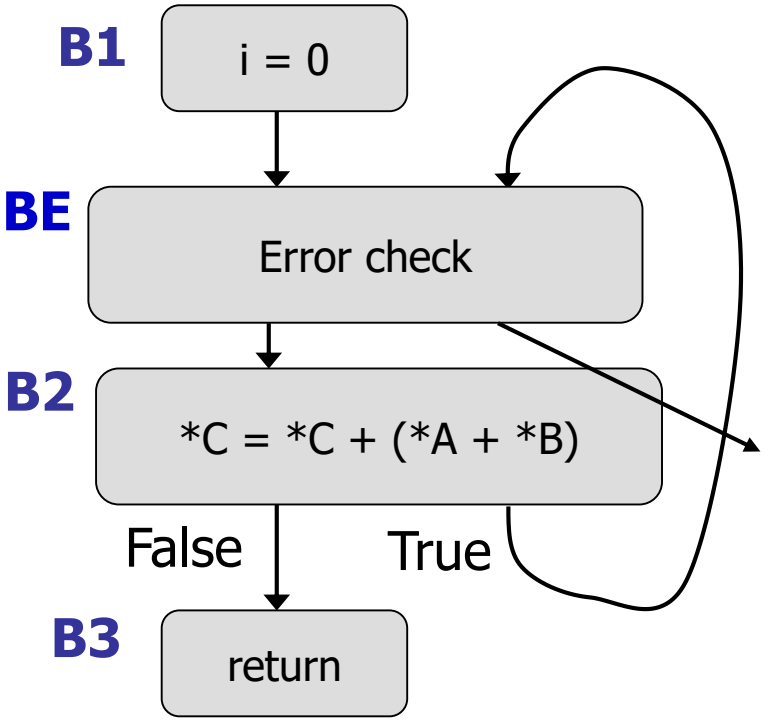
- 2ビットカウンタ方式
  - 大域的な偏り，局所性の利用
  - 2ビットカウンタの状態に応じて予測
  - 予測のためのメモリは2ビット
  - 状態の更新



サンプルプログラム Vector Add

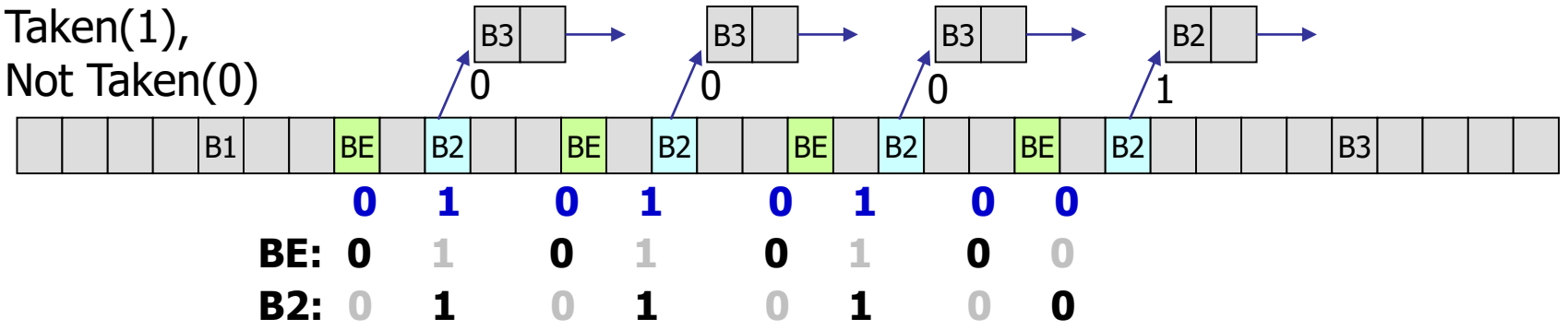



```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++) {
    if(A[i]<0) error_routine();
    C[i] += (A[i] + B[i]);
  }
}
```

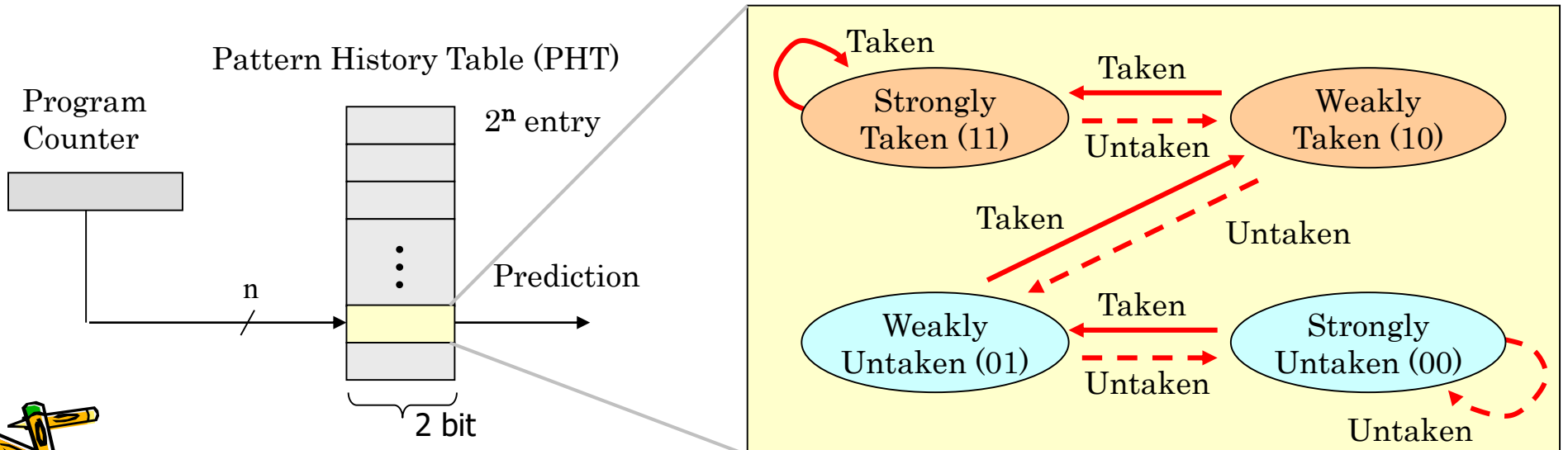


制御フローグラフ

Bimodal branch predictor (ISCA 1981)

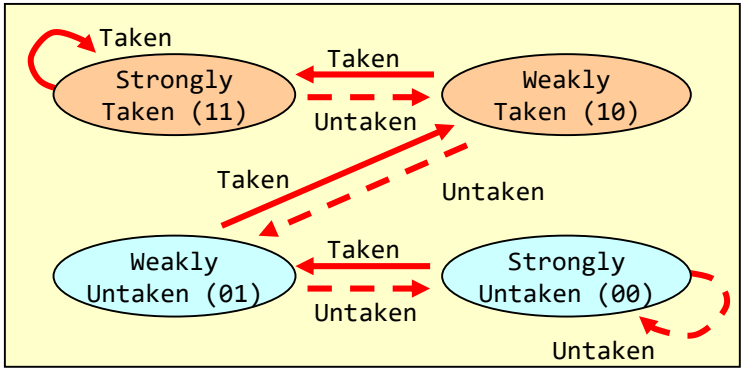


- 分岐アドレス（プログラムカウンタ）毎に履歴を切り替える
  - 分岐アドレスによりパターン履歴表（PHT）のインデックスを作成
- パターン履歴表は2ビットカウンタの配列.



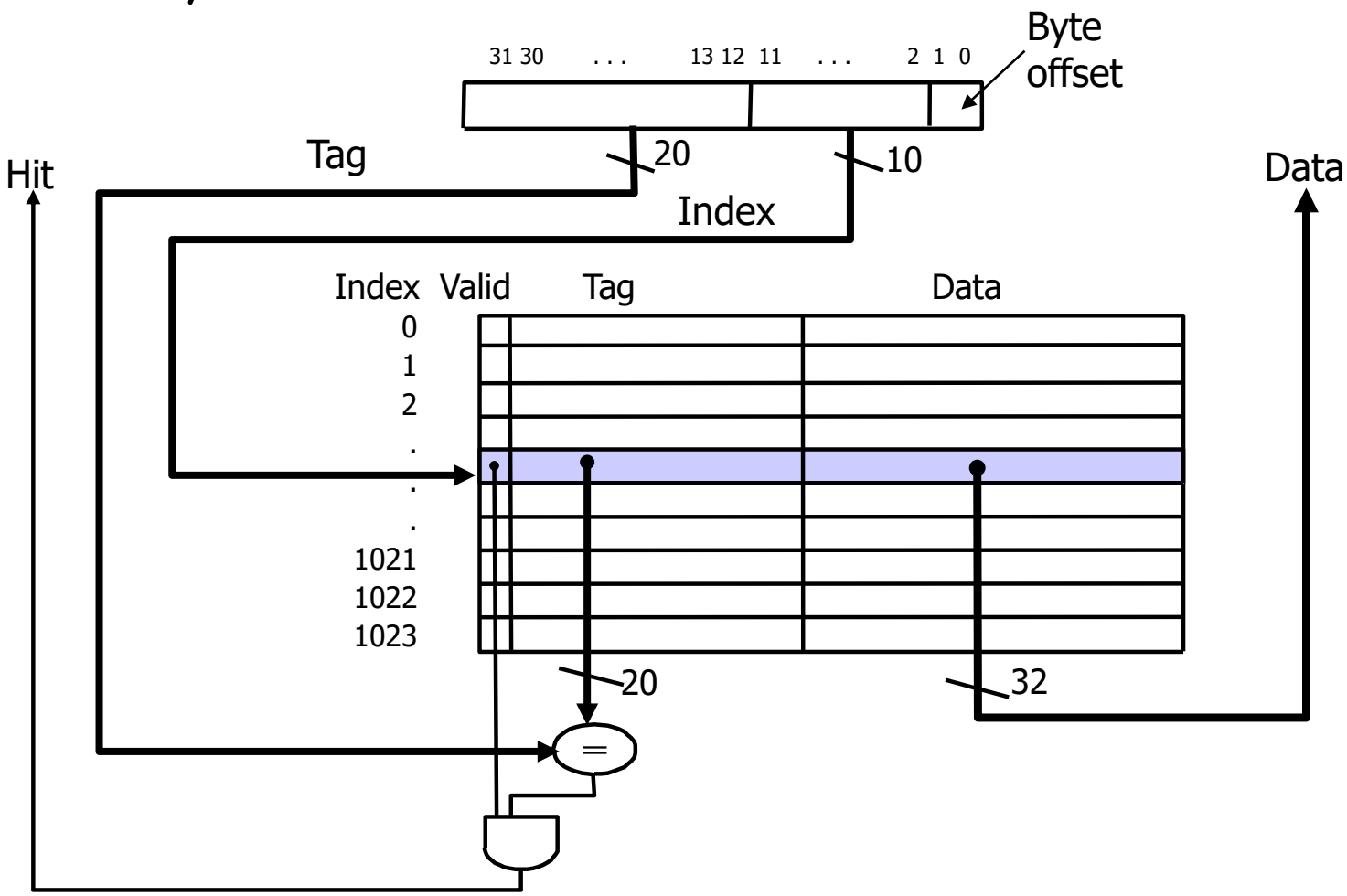
Bimodal branch predictor (ISCA 1981)

```
 1  module m_bimodal(w_clk, w_radr, w_pred, w_wadr, w_we, w_tkn);
 2    input  wire w_clk, w_we;
 3    input  wire [4:0] w_wadr, w_radr;
 4    input  wire w_tkn;
 5    output wire w_pred;
 6    reg [1:0] mem [0:31];
 7    wire [1:0] w_data = mem[w_radr];
 8    assign w_pred = w_data[1];
 9    wire [1:0] w_cnt  = mem[w_wadr];
10    always @(posedge w_clk) if (w_we)
11      mem[w_wadr] <= (w_cnt < 3 &  w_tkn) ? w_cnt + 1 :
12                     (w_cnt > 0 & !w_tkn) ? w_cnt - 1 : w_cnt;
13    integer i; initial for (i=0; i<32; i=i+1) mem[i] = 1;
14  endmodule
```



Direct Mapped Cache Example

- One word/block, cache size = 1K words



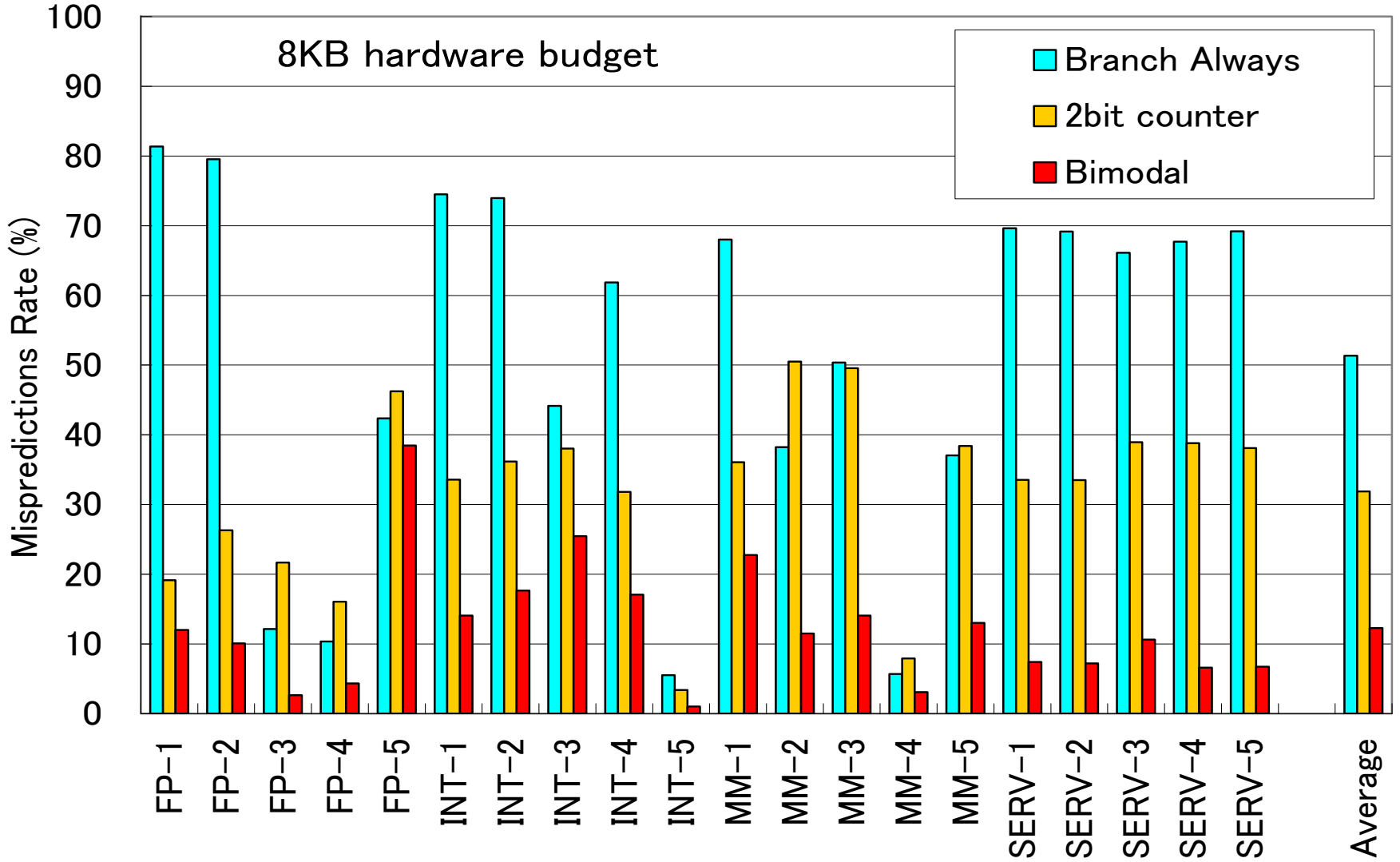
What kind of locality are we taking advantage of?

Bimodal branch predictor and BTB of CFU PG

```
`define BTB_IDXW $clog2(`BTB_ENTRY)  // BTB index width
`define BTB_OSTW $clog2(`XBYTES)     // BTB offset width
module bimodal (
    input  wire             clk_i,
    input  wire             rst_i,
    input  wire             stall_i,
    input  wire [     31:0] raddr_i,
    output wire [      1:0] pat_hist_o,
    output wire             br_pred_tkn_o,
    output wire [`PC_W-1:0] br_pred_pc_o,
    input  wire             br_tkn_i,
    input  wire             br_tsfr_i,
    input  wire [     31:0] waddr_i,
    input  wire [      1:0] pat_hist_i,
    input  wire [     31:0] br_tkn_pc_i
);
    integer i;
    (* ram_style = "block" *) reg [`PC_W-1:0] btb[0:`BTB_ENTRY-1];  // BTB, branch target buf
    initial for (i = 0; i < `BTB_ENTRY; i = i + 1) btb[i] = 0;  // init with weak untaken

    wire [1:0] w_cnt = (br_tkn_i) ? pat_hist_i + (pat_hist_i < 3) : pat_hist_i - (pat_hist_i > 0);
    wire [`BTB_IDXW-1:0] btb_ridx = raddr_i[`BTB_IDXW+`BTB_OSTW-1:`BTB_OSTW];
    wire [`BTB_IDXW-1:0] btb_widx = waddr_i[`BTB_IDXW+`BTB_OSTW-1:`BTB_OSTW];

    reg [31:0] r_btb_entry;
    always @(posedge clk_i) if (!stall_i) begin
        r_btb_entry <= btb[btb_ridx];
        if (br_tsfr_i) begin
            btb[btb_widx] <= {br_tkn_pc_i[`PC_W-1:2], w_cnt};  // lower tow bits for counter
        end
    end
    assign pat_hist_o    = r_btb_entry[1:0];
    assign br_pred_tkn_o = r_btb_entry[1];
    assign br_pred_pc_o  = {r_btb_entry[`PC_W-1:2], 2'b0};
endmodule
```

```
`define BTB_ENTRY (2*1024)  // the number of BTB entries for branch prediction

// ram
`define IMEM_SIZE (32*1024) // instruction memory size in byte
`define DMEM_SIZE (16*1024) // data memory size in byte

`define IMEM_ENTRIES (`IMEM_SIZE/4)
`define DMEM_ENTRIES (`DMEM_SIZE/4)
```

ここまでの分岐予測の精度（予測ミス率）



Benchmark for CBP(2004) by Intel MRL and IEEE TC uARCH.