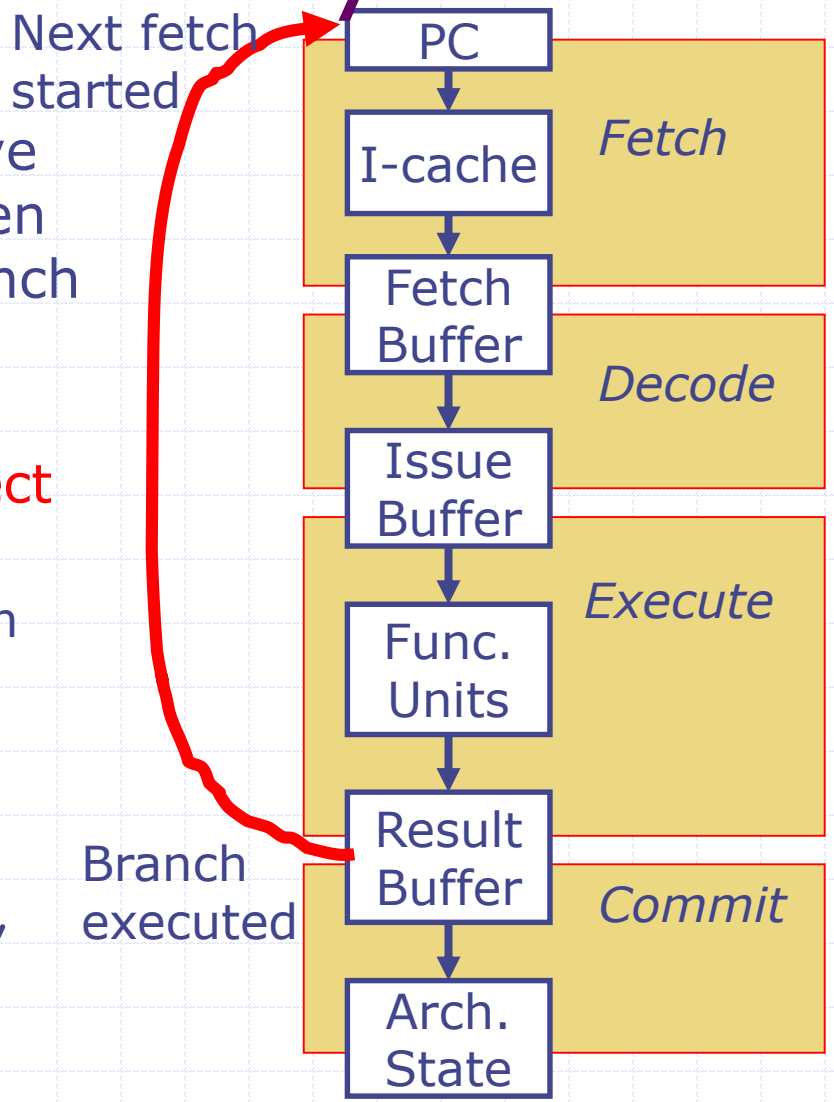
Constructive Computer Architecture:

Branch Prediction

Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

Control Flow Penalty

- Modern processors may have > 10 pipeline stages between next PC calculation and branch resolution !
- How much work is lost if pipeline doesn't follow correct instruction flow?
  - Loop length x pipeline width
- What fraction of executed instructions are branch instructions?



superscalarity

How frequent are branches? ARM Cortex 7

Blem et al [HPCA 2013] Spec INT 2006

	ARM Cortex-A9; ARMv7 ISA				
Benchmark	Total Instructions	branch %	load %	store %	other %
astar	1.47E+10	16.0	55.6	13.0	15.4
bzip2	2.41E+10	8.7	34.6	14.4	42.2
gcc	5.61E+09	10.2	19.1	11.2	59.5
gobmk	5.75E+10	10.7	25.4	7.2	56.8
hmmer	1.56E+10	5.1	41.8	18.1	35.0
h264	1.06E+11	5.5	30.4	10.4	53.6
libquantum	3.97E+08	11.5	8.1	11.7	68.7
omnetpp	2.67E+09	11.7	19.3	8.9	60.1
perlbench	2.69E+09	10.7	24.6	9.3	55.5
sjeng	1.34E+10	11.5	39.3	13.7	35.5
Average		8.2	31.9	10.9	49.0

Every 12th instruction is a branch

How frequent are branches? X86

Blem et al [HPCA 2013] Spec INT 2006

		core i7; x86 ISA			
Benchmark	Total Instructions	branch %	load %	store %	other %
astar	5.71E+10	6.9	19.5	6.9	66.7
bzip2	4.25E+10	11.1	31.2	11.8	45.9
hmmer	2.57E+10	5.3	30.5	9.4	54.8
gcc	6.29E+09	15.1	22.1	14.1	48.7
gobmk	8.93E+10	12.1	21.7	13.4	52.7
h264	1.09E+11	7.1	46.8	18.5	27.6
libquantum	4.18E+08	13.2	39.3	6.8	40.7
omnetpp	2.55E+09	16.4	28.6	21.4	33.7
perlbench	2.91E+09	17.3	25.9	16.0	40.8
sjeng	2.11E+10	14.8	22.8	11.0	51.4
Average		9.4	31.0	13.4	46.2

Every 10th or 11th instruction is a branch

How frequent are branches? ARM Cortex 7

Blem et al [HPCA 2013] Spec FP 2006

	ARM Cortex-A9; ARMv7 ISA				
Benchmark	Total Instructions	branch %	load %	store %	other %
bwaves	3.84E+11	13.5	1.4	0.5	84.7
cactusADM	1.02E+10	0.5	51.4	17.9	30.1
leslie3D	4.92E+10	6.2	2.0	3.7	88.1
milc	1.38E+10	6.5	38.2	13.3	42.0
tonto	1.30E+10	10.0	40.5	14.1	35.4
Average		12.15	4.68	1.95	81.22

Every 8th instruction is a branch

How frequent are branches? X86

Blem et al [HPCA 2013] Spec FP 2006

		core i7; x86 ISA			
Benchmark	Total Instructions	branch %	load %	store %	other %
bwaves	3.41E+10	3.2	51.4	16.8	28.7
cactusADM	1.05E+10	0.4	55.3	18.6	25.8
leslie3D	6.25E+10	4.9	35.3	12.8	46.9
milc	3.29E+10	2.2	32.2	13.8	51.8
tonto	4.88E+09	7.1	27.2	12.4	53.3
Average		3.6	39.6	14.4	42.4

Every 27th instruction is a branch

Observations

- Control transfer happens every 8th to 30th instruction
- Static vs dynamic predictors: Does the prediction depend upon the execution history?
- Processors often use more than one predictor and it takes considerable effort to
  - Integrate a prediction scheme in the pipeline
  - Understand the interactions between various schemes
  - Understand the performance implications
- There is a plethora of branch prediction schemes – their importance grows with the depth of processor pipeline

we will start with the basics …

RISC V Branches & Jumps

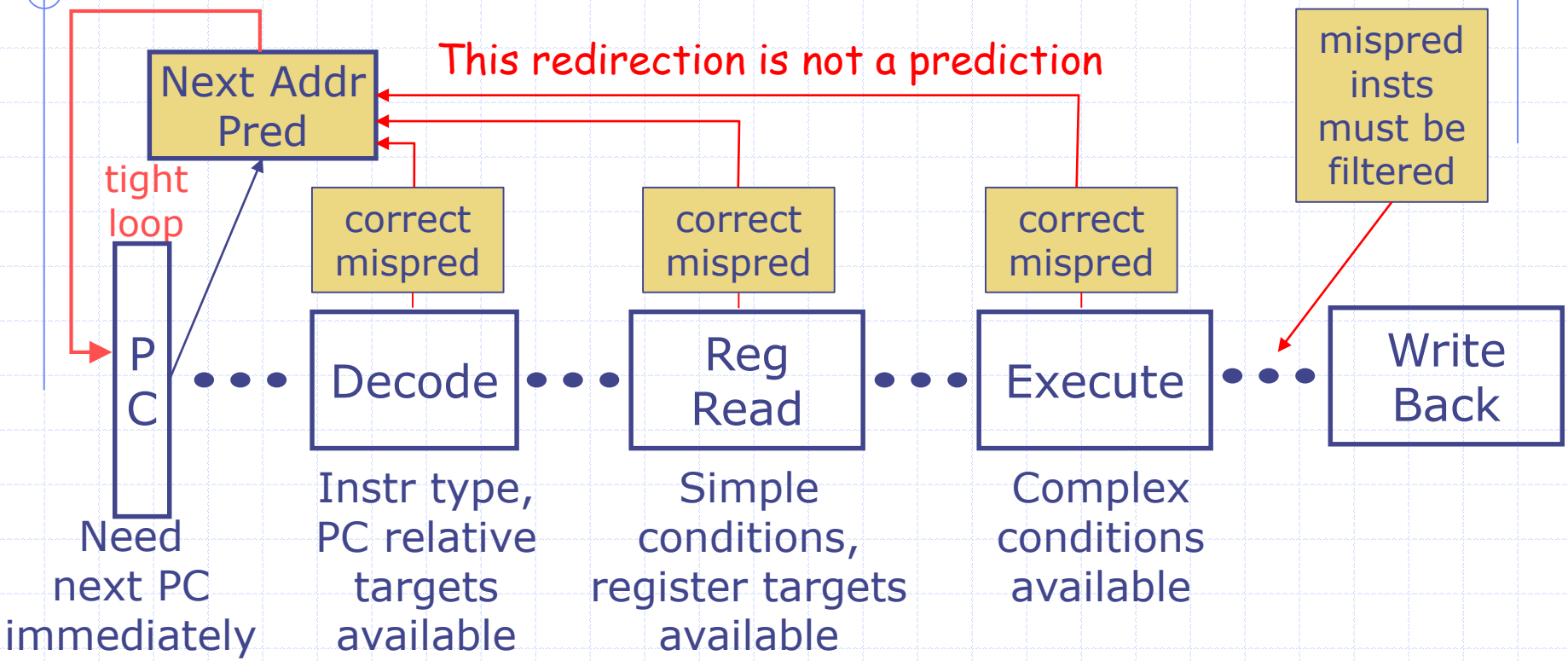
Each instruction fetch depends on some information from the preceding instruction:

1. Is the preceding instruction a taken branch?
2. If so, what is the target address?

Instruction	*Direction known after*	*Target known after*
JAL	After Inst. Decode	After Inst. Decode
JALR	After Inst. Decode	After Reg. Fetch
BEQ/BNE ...	After Exec	After Inst. Decode

A predictor can redirect the pc only after the relevant information required by the predictor is available

Overview of control prediction

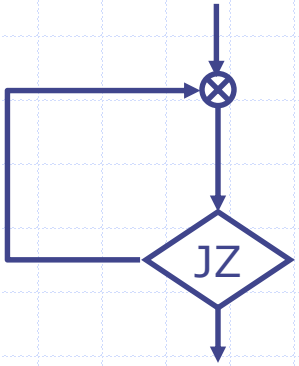


Given (pc, ppc), a misprediction can be corrected (used to redirect the pc) as soon as it is detected. In fact, pc can be redirected as soon as we have a "better" prediction.

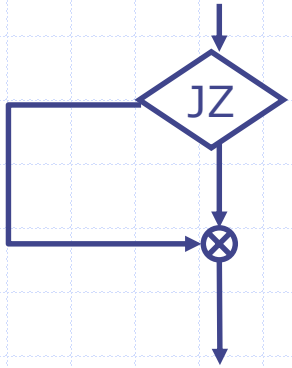
Static Branch Prediction

- Since most instructions do not result in a control transfer, pc+4 is a good predictor
- Overall probability a branch is taken is ~60-70% but:

backward 90%



forward 50%

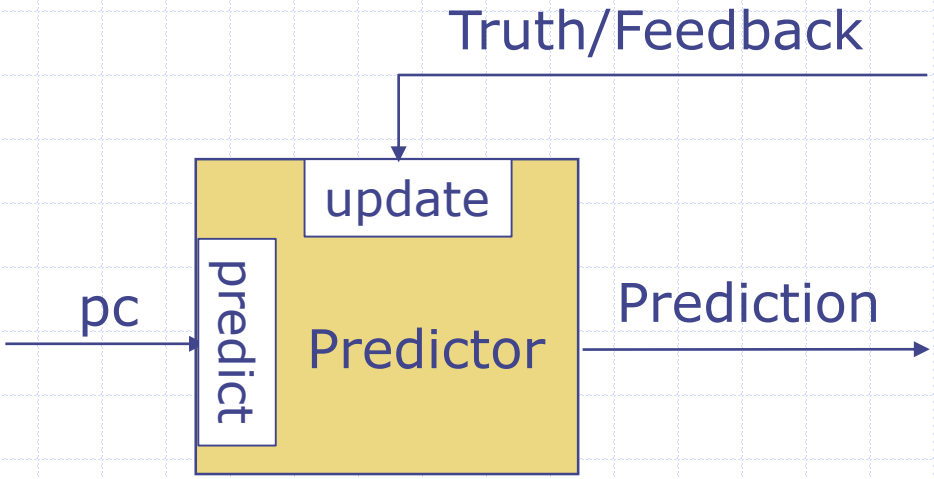


- ISA can attach preferred direction semantics to branches, e.g., Motorola MC88110
  - bne0 *(preferred taken)* beq0 *(not taken)*
- ISA can allow arbitrary choice of statically predicted direction, e.g., HP PA-RISC, Intel IA-64
  - reported as ~80% accurate

… but our ISA is fixed!

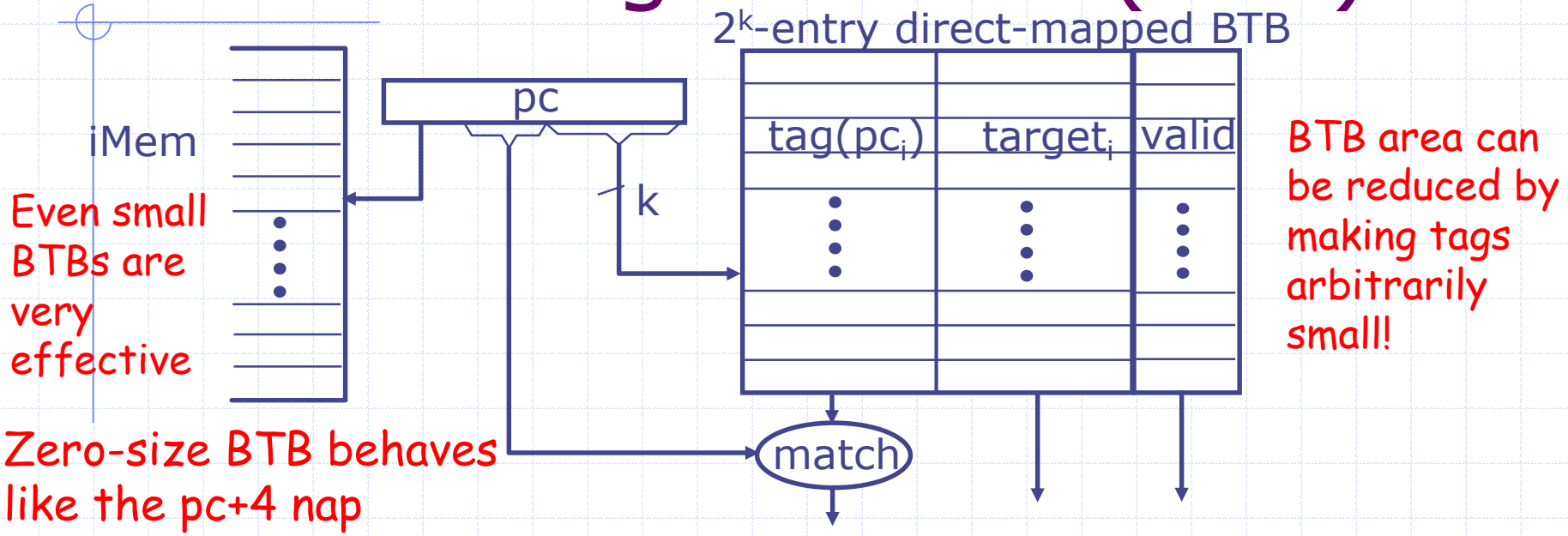
Dynamic Branch Prediction

learning based on past behavior



- Temporal correlation
  - The way a branch resolves may be a good predictor of the way it will resolve at the next execution
- Spatial correlation
  - Several branches may resolve in a highly correlated manner (a preferred path of execution)

Next Address Predictor: Branch Target Buffer (BTB)



- BTB remembers recent target PC's for a set of *control instructions*
  - Fetch: looks for the pc and the associated target in BTB; if pc is not found then ppc is pc+4
  - Execute: checks prediction, if wrong poisons the wrong-path instructions; updates the BTB for jumps and taken-branches

BTB permits ppc to be determined *before* the instruction is decoded

Next Addr Predictor interface

```
interface AddrPred;
  method Addr nap(Addr pc);
  method Action update(Addr pc, Addr nextPC,
                                Bool taken);
endinterface
```

- *Predictor training:* On a pc misprediction, pc and epoch are updated and the relevant information is passed to the next address predictor
  - *nap:* simple look up
  - *update:* On a pc misprediction, if the jump or branch at the pc was taken then the BTB is updated with the new (pc, nextPC) otherwise the pc entry is deleted

Code is given at the end

BTB predictor update method

```
method Action update(Addr pc, Addr nextPC,
                                    Bool taken);
    let index = getIndex(pc);
    let tag = getTag(pc);
    if(taken) begin
     validArr[index] <= True;
     entryPcArr.upd(index, tag);
     ppcArr.upd(index, nextPc);
    end
    else if(tag == entryPcArr.sub(index))
            validArr[index] <= False;
  endmethod
```

Integrating BTB in the 4-Stage pipeline

```
...
AddrPred          btb <- mkBtb

rule fetch;
       iMem.enq(pc[1]);
       let ppcF = nap(pc[1]); pc[1] <= ppcF;
       f2d.enq(Fetch2Decode(pc:pc[1], ppc:ppcF,
                                    epoch:epoch[1]));
endrule
rule decode; ...//no change
rule execute; ...
rule writeBack; ... //no change
```

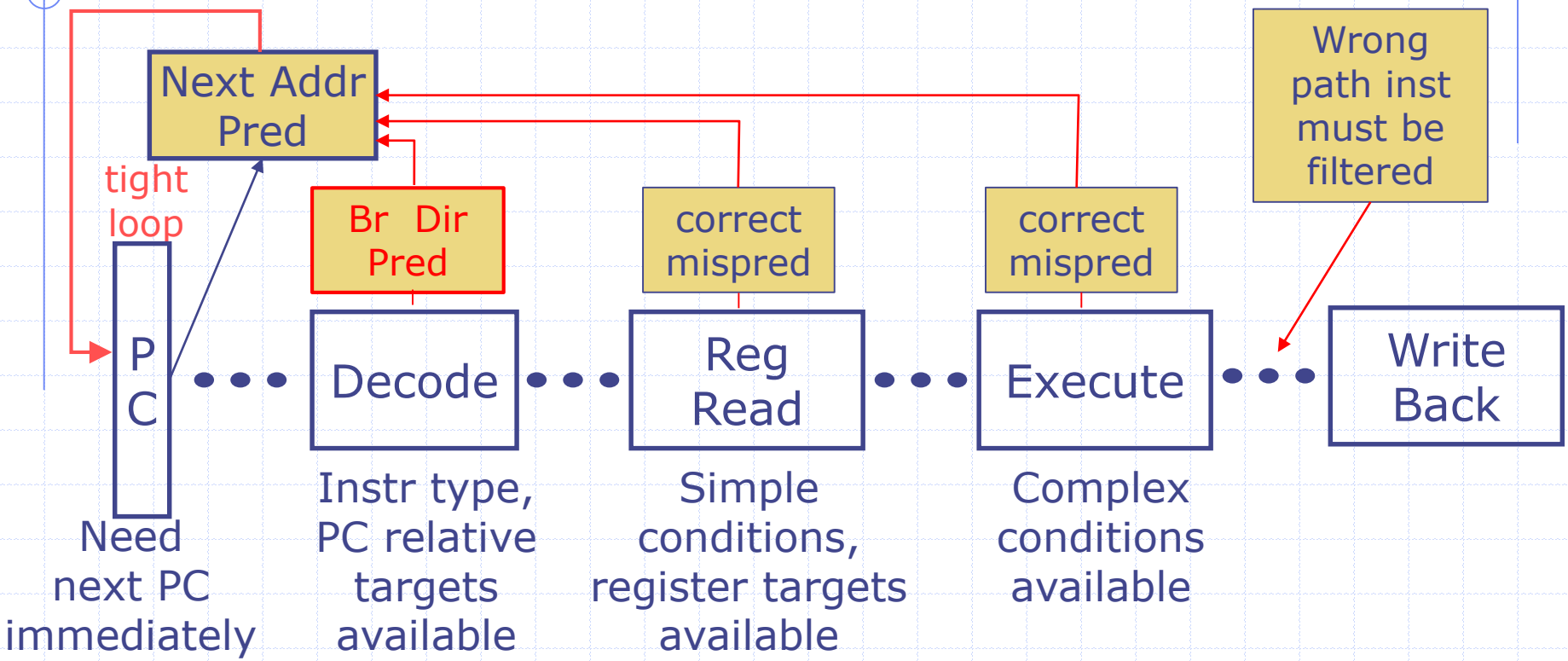
btb.nap(pc[1])

4-Stage-pipeline without Branch predictors execute

```
rule execute;
    let x = d2e.first; ...
    if(epoch[0] != inEp) begin e2w.enq(Invalid); d2e.deq; end
    else      begin
     let eInst = exec(dInstE, rVal1E, rVal2E, pcE);
      if(eInst.iType == Ld)
        dMem.enq(MemReq{op:Ld, addr:eInst.addr, data:?});
      else if (eInst.iType == St) begin
        dMem.enq(MemReq{op:St, addr:eInst.addr,
                        data:eInst.data}); end
      let nextPC = eInst.brTaken ? eInst.addr : pcE + 4;
      if (x.ppc != nextPC) begin pc[0] <= eInst.addr;
                                 epoch[0] <= !epoch[0]; end
                          btb.update(pcE, nextPC, eInst.brTaken);
      e2w.enq(Valid Exec12Exec2(eInst:eInst, pc:pcE));
      d2e.deq; end
endrule
```

for btb training

Multiple Predictors: BTB + Branch Direction Predictors



- Suppose we maintain a table of how a particular Br has resolved before. At the decode stage we can consult this table to check if the incoming (pc, ppc) pair matches our prediction. If not redirect the pc

Branch Prediction Bits

Remember how the branch was resolved previously

- Assume 2 BP bits per instruction
- Use saturating counter

On ¬taken ↓	↑ On taken			
		1	1	Strongly taken
		1	0	Weakly taken
		0	1	Weakly ¬taken
		0	0	Strongly ¬taken

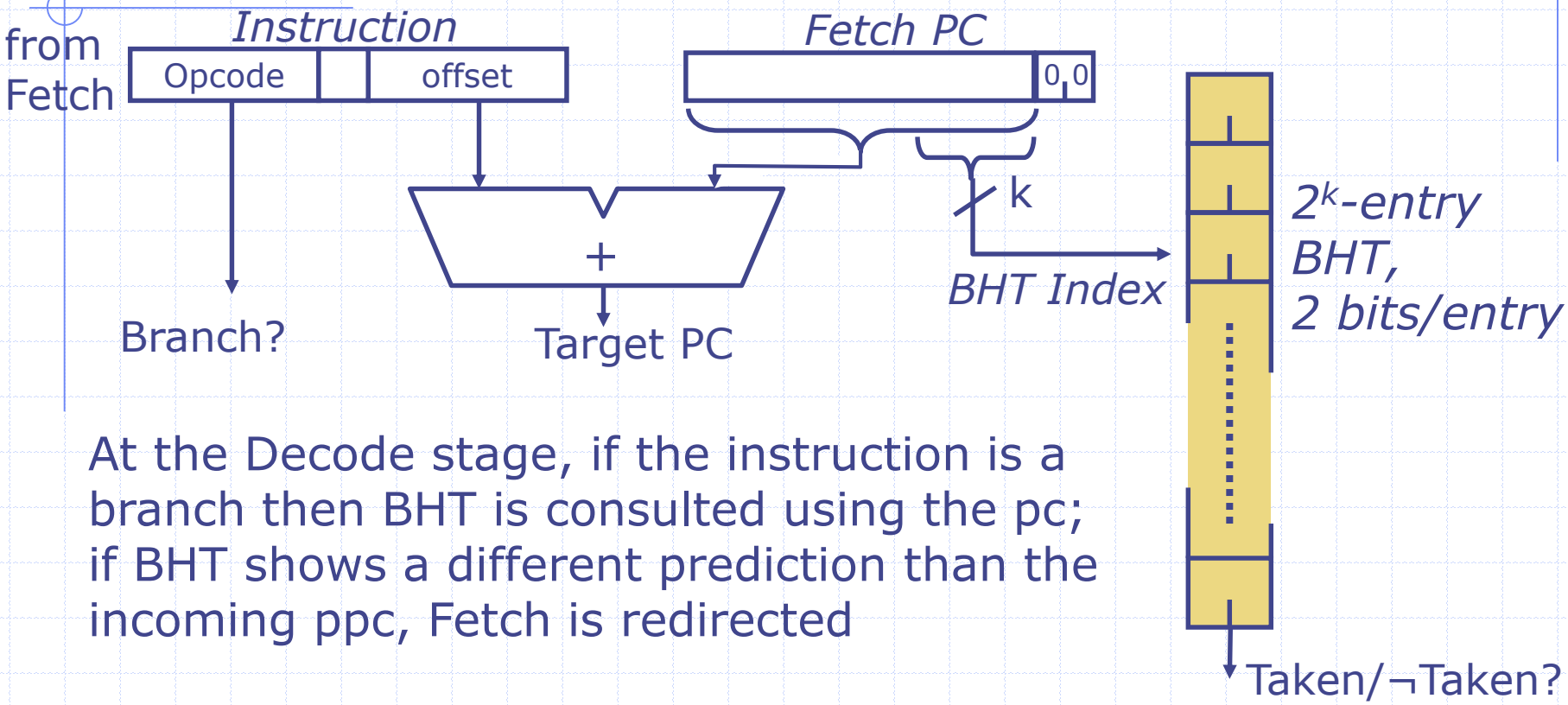
Direction prediction changes only after two successive bad predictions

Two-bit versus one-bit Branch prediction

- Consider the branch instruction needed to implement a loop
  - with one bit, the prediction will always be set incorrectly on loop exit
  - with two bits the prediction will not change on loop exit

A little bit of hysteresis is good in changing predictions

Branch History Table (BHT)



At the Decode stage, if the instruction is a branch then BHT is consulted using the pc; if BHT shows a different prediction than the incoming ppc, Fetch is redirected

4K-entry BHT, 2 bits/entry, ~80-90% correct direction predictions

4-Stage-pipeline without Branch predictors fetch, decode

```
rule fetch;
      iMem.enq(pc[1]);
      let ppcF = nap(pc[1]); pc[1] <= ppcF ;
      f2d.enq(Fetch2Decode(pc:pc[1], ppc:ppcF, epoch:epoch[1]))
endrule
rule decode;
  let inst = iMem.first;   let x = f2d.first;
  if(epoch[1] != x.inEp) begin iMem.deq; f2d.deq end //wrongpath
  else begin
    let dInst = decode(inst);
    let stall = sb.search1(dInst.src1)|| sb.search2(dInst.src2);
    if(!stall) begin
          ...fetch register values
        d2e.enq(Decode2Execute{pc: x.pc, ppc: x.ppc,
             dInst: dInst, epoch: x.epoch,
             rVal1: rVal1, rVal2: rVal2});
        sb.insert(dInst.rDst); iMem.deq; f2d.deq end
endrule
```

4-Stage-pipeline without Branch predictors execute

```
rule execute;
    let x = d2e.first; ...
    if(epoch[0] != inEp) begin e2w.enq(Invalid); d2e.deq; end
    else      begin
     let eInst = exec(dInstE, rVal1E, rVal2E, pcE);
      if(eInst.iType == Ld)
        dMem.enq(MemReq{op:Ld, addr:eInst.addr, data:?});
      else if (eInst.iType == St) begin
        dMem.enq(MemReq{op:St, addr:eInst.addr,
                        data:eInst.data}); end
      let nextPC = eInst.brTaken ? eInst.addr : pcE + 4;
      if (x.ppc != nextPC) begin pc[0] <= eInst.addr;
                                 epoch[0] <= !epoch[0]; end
      e2w.enq(Valid Exec12Exec2(eInst:eInst, pc:pcE));
      d2e.deq;
    end
endrule
```

4-Stage-pipeline without Branch predictors writeback

```
rule writeback;
    let vx = e2w.first;
    if (vx matches tagged Valid .x) begin
     let pcE=x.pc;  let eInst=x.eInst;
     if (isValid(eInst.dst)) begin
       let data = eInst.iType==Ld ? dMem.first: eInst.data;
       rf.wr(fromMaybe(?, eInst.dst), data);
     end
     if(eInst.iType == Ld) dMem.deq;
    end
    sb.remove; e2w.deq;
endrule
```

BTB predictor

```
module mkBtb(AddrPred);
  RegFile#(BtbIndex, Addr) ppcArr <- mkRegFileFull;
  RegFile#(BtbIndex, BtbTag) entryPcArr <- mkRegFileFull;
  Vector#(BtbEntries, Reg#(Bool))
       validArr <- replicateM(mkReg(False));
  function BtbIndex getIndex(Addr pc)=truncate(pc>>2);
  function BtbTag getTag(Addr pc) = truncateLSB(pc);
  method Addr nap(Addr pc);
    BtbIndex index = getIndex(pc);
    BtbTag tag = getTag(pc);
    if(validArr[index] && tag == entryPcArr.sub(index))
      return ppcArr.sub(index);
    else return (pc + 4);
  endmethod
  method Action update(Addr pc, Addr nextPC, Bool taken);…
endmodule
```

4-Stage-pipeline with BTB

fetch, decode

```
rule fetch;
      iMem.enq(pc[1]);
      let ppcF = btb.nap(pc[1]); pc[1] <= ppcF ;
      f2d.enq(Fetch2Decode(pc:pc[1], ppc:ppcF, epoch:epoch[1]))
endrule
rule decode;
  let inst = iMem.first;   let x = f2d.first;
  if(epoch[1] != x.inEp) begin iMem.deq; f2d.deq end //wrongpath
  else begin
    let dInst = decode(inst);
    let stall = sb.search1(dInst.src1)|| sb.search2(dInst.src2);
    if(!stall) begin
          ...fetch register values
        d2e.enq(Decode2Execute{pc: x.pc, ppc: x.ppc,
             dInst: dInst, epoch: x.epoch,
             rVal1: rVal1, rVal2: rVal2});
        sb.insert(dInst.rDst); iMem.deq; f2d.deq end
endrule
```

4-Stage-pipeline with BTB

execute

```
rule execute;
    let x = d2e.first; ...
    if(epoch[0] != inEp) begin e2w.enq(Invalid); d2e.deq; end
    else      begin
     let eInst = exec(dInstE, rVal1E, rVal2E, pcE);
      if(eInst.iType == Ld)
        dMem.enq(MemReq{op:Ld, addr:eInst.addr, data:?});
      else if (eInst.iType == St) begin
        dMem.enq(MemReq{op:St, addr:eInst.addr,
                        data:eInst.data}); end
      let nextPC = eInst.brTaken ? eInst.addr : pcE + 4;
      if (x.ppc != nextPC) begin pc[0] <= eInst.addr;
                                 epoch[0] <= !epoch[0];
                        btb.update(pcE, nextPC, eInst.brTaken); end
      e2w.enq(Valid Exec12Exec2(eInst:eInst, pc:pcE));
      d2e.deq; end
endrule
```

4-Stage-pipeline with BTB
writeback

```
rule writeback;
    let vx = e2w.first;
    if (vx matches tagged Valid .x) begin
     let pcE=x.pc;  let eInst=x.eInst;
     if (isValid(eInst.dst)) begin
       let data = eInst.iType==Ld ? dMem.first: eInst.data;
       rf.wr(fromMaybe(?, eInst.dst), data);
     end
     if(eInst.iType == Ld) dMem.deq;
    end
    sb.remove; e2w.deq;
endrule
```