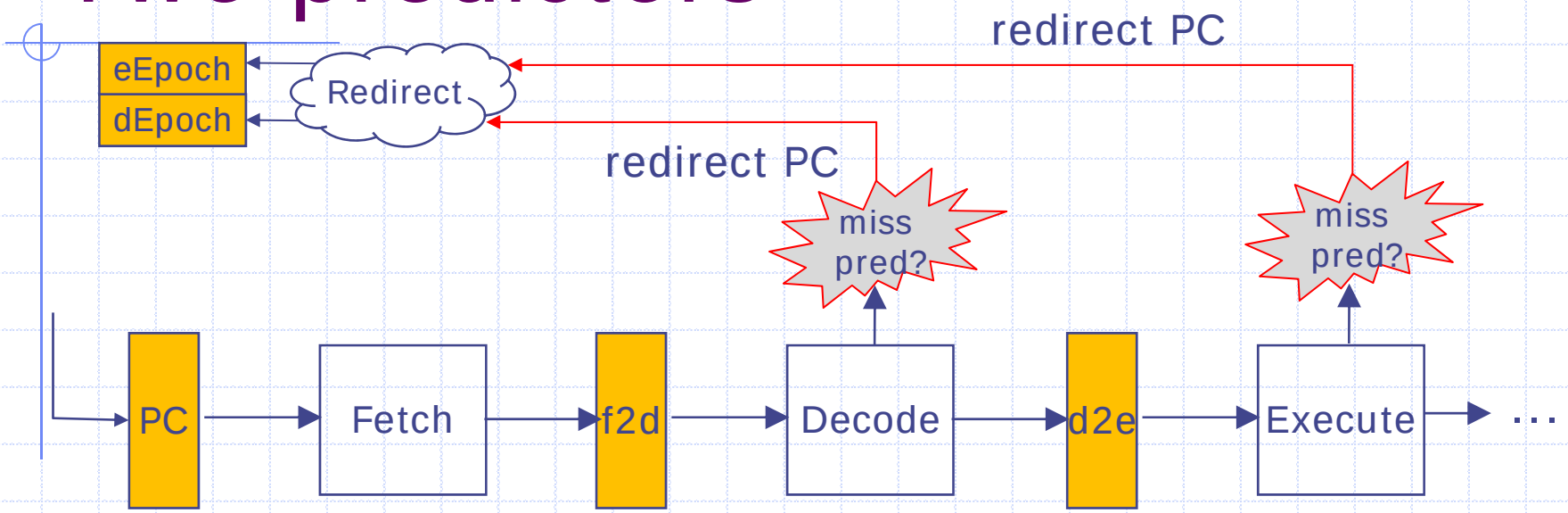
Global Epoch

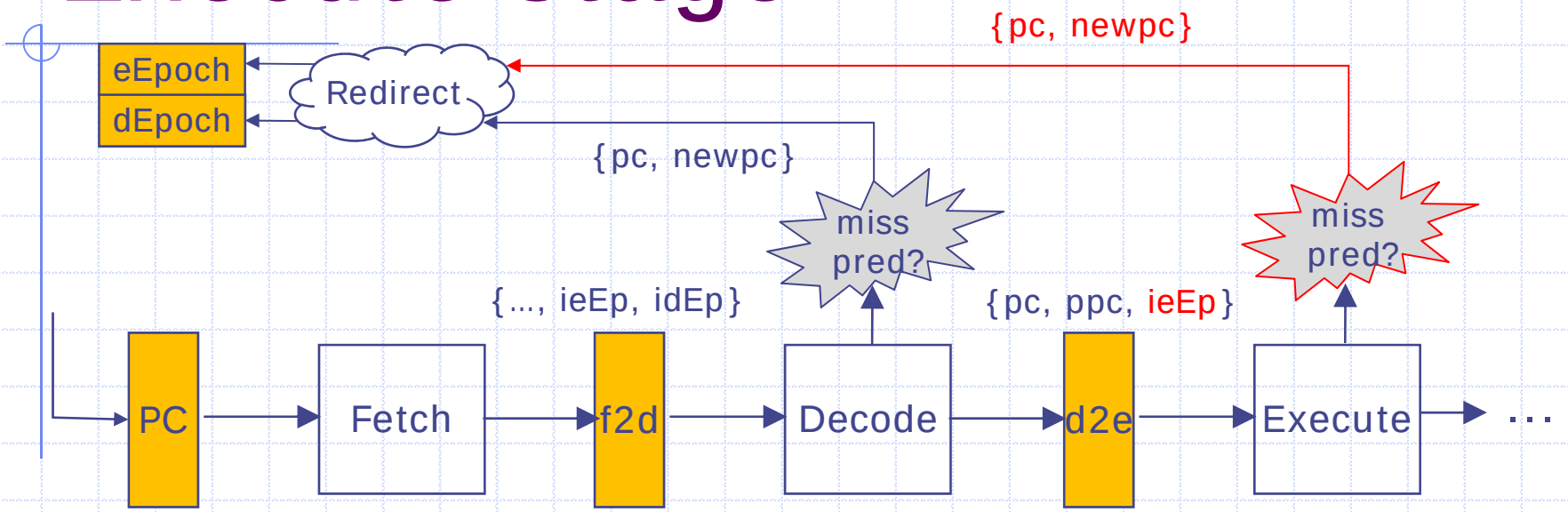
Sizhuo Zhang
6.175 TA

N-Stage pipeline: Two predictors



- Both Decode and Execute can redirect the PC; Execute redirect should never be overruled
- Global epoch for each redirecting stage
  - eEpoch: flipped when redirect from Execute takes effect
  - dEpoch: flipped when redirect from Decode takes effect
  - Initially set all epochs to 0

Execute stage



Is ieEp == eEpoch?

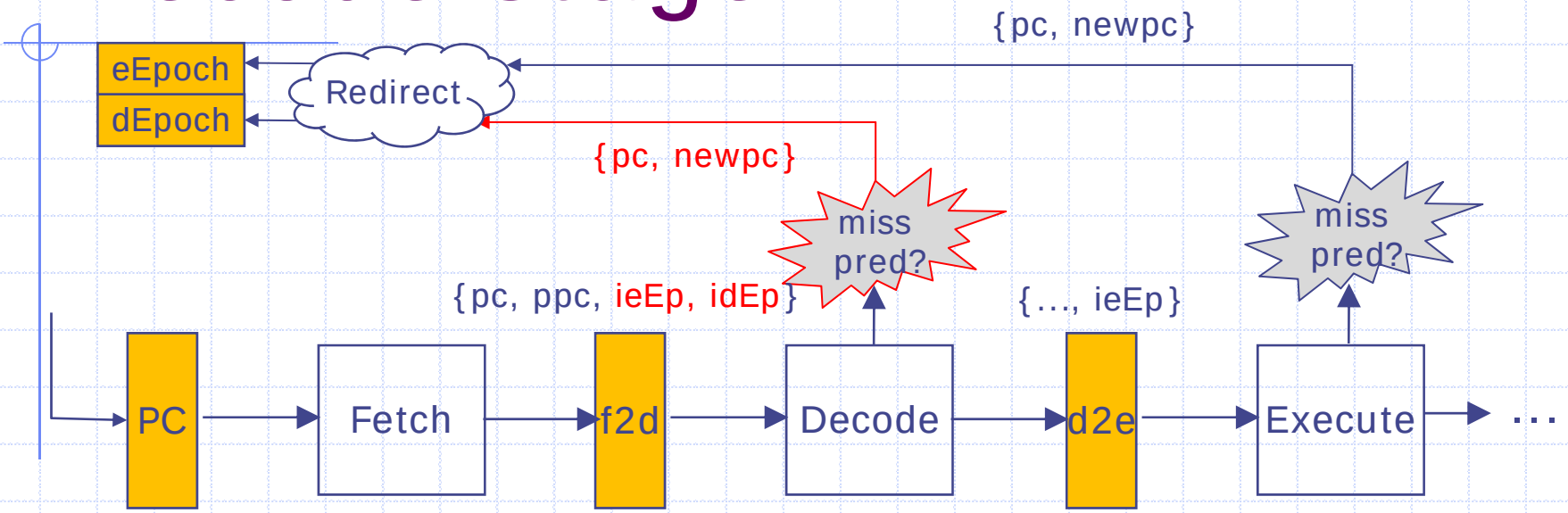
yes

no

Current instruction is OK; check the ppc prediction by execution, signal a redirect message (i.e. combinational pulse) on misprediction

Wrong path instruction; poison it

Decode stage



Is ieEp == eEpoch && idEp == dEpoch ?

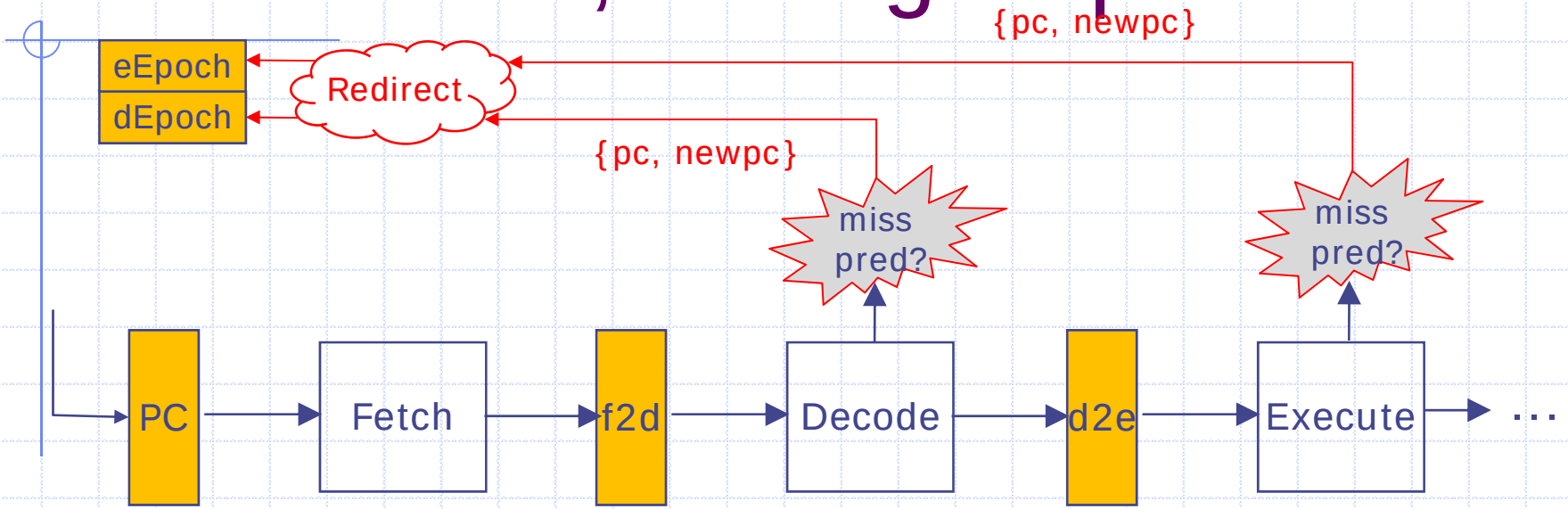
yes

Current instruction is OK; check the ppc prediction via BHT, signal a redirect message (i.e. combinational pulse) on misprediction

no

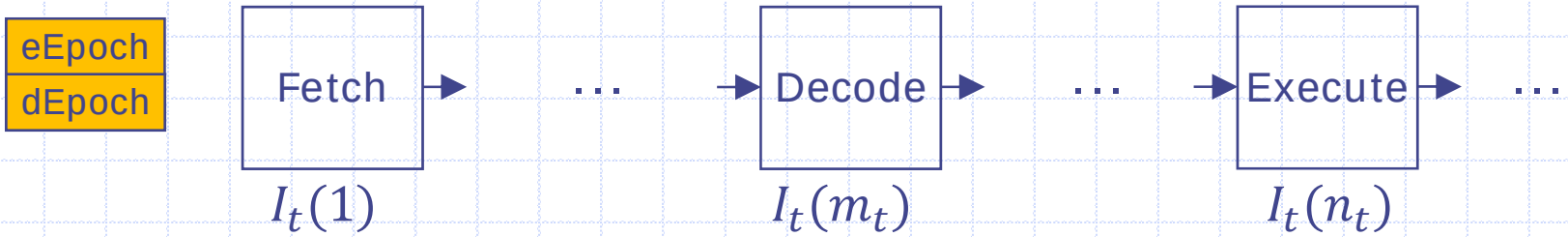
Wrong path instruction; drop it

Fetch stage: redirect PC, change epoch



- If there is redirect message from Execute
  - Set PC to correct value, Flip eEpoch
- Else if there is redirect message from Decode
  - Set PC to correct value, Flip dEpoch
- We can always request IMem to fetch new instruction
  - New instruction will be tagged with current eEpoch and dEpoch
  - PC redirection overrides next-PC prediction
  - When PC redirects, new instruction must be killed at Decode stage

Proof of correctness



- Assume epochs are unbounded instead of just 1-bit
  - On PC redirection, increment epoch instead of flipping it
  - Obviously correct
- At cycle t, n_t instructions between Fetch and Execute: $I_t(1 \ldots n_t)$
  - $I_t(1)$ at Fetch, $I_t(n_t)$ at Execute, $I_t(m_t)$ at Decode
- Invariant:
  - $eEpoch = I_t(1).ieEp \geq I_t(2).ieEp \geq \cdots \geq I_t(n_t).ieEp \geq eEpoch - 1$
  - $dEpoch = I_t(1).idEp \geq I_t(2).idEp \geq \cdots \geq I_t(m_t).idEp \geq dEpoch - 1$
  - Proved by induction on t
- Observation: only need last bit of epoch – 1-bit epochs

4-stage Pipeline: code sketch

```
module mkProc(Proc);
  // stage 1: Fetch
  // stage 2: Decode & read register
  // stage 3: Execute & access memory
  // stage 4: Commit (write register file)
  Ehr#(2, Addr)     pc <- mkEhr(?);
  IMemory          iMem <- mkIMemory;
  ...
  Fifo#(2, Fetch2Decode)   f2d <- mkCFFifo;
  Fifo#(2, Decode2Execute) d2e <- mkCFFifo;
  Fifo#(2, Execute2Commit) e2c <- mkCFFifo;
  Reg#(Bool) eEpoch <- mkReg(False);
  Reg#(Bool) dEpoch <- mkReg(False);
  Ehr#(2, Maybe#(ExeRedirect)) exeRedirect <- mkEhr(Invalid);
  Ehr#(2, Maybe#(DecRedirect)) decRedirect <- mkEhr(Invalid);
  BTB#(16)   btb <- mkBTB;
  BHT#(1024) bht <- mkBHT;
```

Execute stage

```
rule doExecute;
    d2e.deq; let x = d2e.first;
    if(x.ieEp == eEpoch) begin
        let eInst = exec(...);
        if(eInst.mispredict)
            exeRedirect[0] <= Valid ({pc: x.pc,
                                      newpc: eInst.addr});
      ...
    end
    else e2c.enq(Exec2Commit{poisoned: True, ...});
        // wrong path instruction, poison it
endrule
```

BHT training is missing here

Decode stage

```
rule doDecode;
    f2d.deq; let x = f2d.first;
    if(x.ieEp == eEpoch && x.idEp == dEpoch) begin
        let stall = ...; // check scoreboard for stall
        if(!stall) begin
            if(x.iType == Br) begin
                let bht_ppc = ...; // BHT prediction
                if(bht_ppc != x.ppc)
                    decRedirect[0] <= Valid ({pc: x.pc,
                                              newpc: bht_ppc});
            end
            ...
            d2e.enq(Decode2Execute{ieEp: x.ieEp, ...});
        end
    end
    // else: wrong path instruction, killed
endrule
```

Fetch stage

```
rule doFetch;
    let inst = iMem.req(pc[0]);
    let ppc = btb.predPc(pc[0]);
    pc[0] <= ppc;
    f2d.enq(Fetch2Decode{pc: pc[0], ppc: ppc, inst: inst,
                         ieEp: eEpoch, idEp: dEpoch});
endrule

rule cononicalizeRedirect;
    if(exeRedirect[1] matches tagged Valid .r) begin
        pc[1] <= r.newpc; btb.update(r.pc, r.newpc);
        eEpoch <= !eEpoch;
    end else if(decRedirect[1] matches tagged Valid .r) begin
        pc[1] <= r.newpc; btb.update(r.pc, r.newpc);
        dEpoch <= !dEpoch;
    end
    exeRedirect[1] <= Invalid; decRedirect[1] <= Invalid;
endrule
```