18-747 Lecture 10: Trace Caching

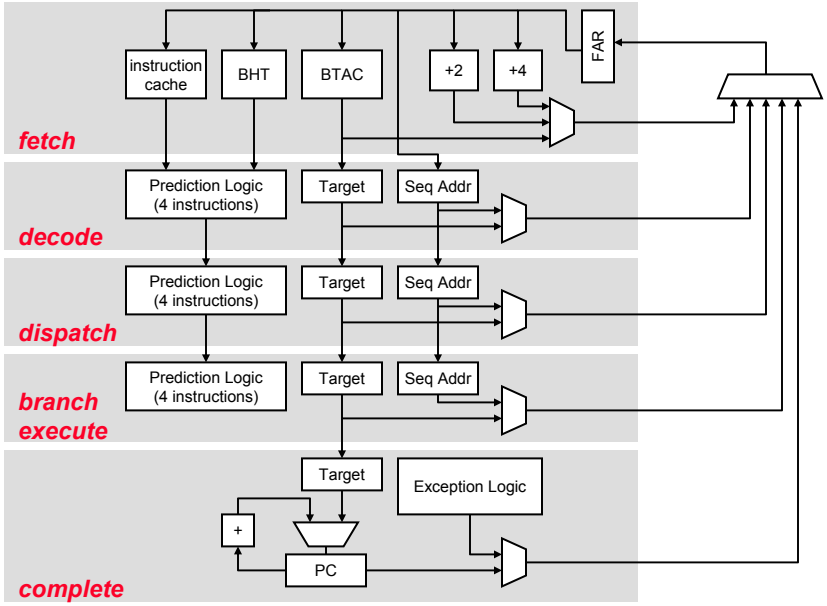
James C. Hoe
Dept of ECE, CMU
October 1, 2001

Reading Assignments: 2 papers below

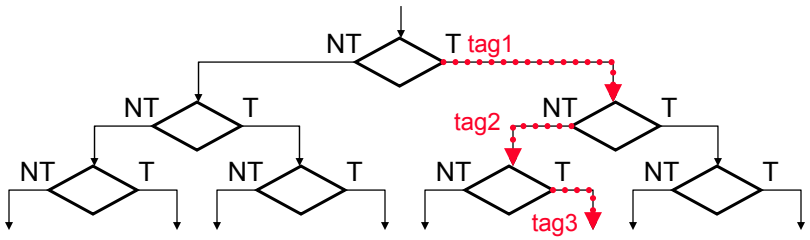
Announcements: Midterm Exam on Monday 10/15
Condor Usage

Handouts: *"Critical Issues Regarding the Trace Cache Fetch Mechanism"*
"The Block-based Trace Cache"

PPC604 Speculative Execution

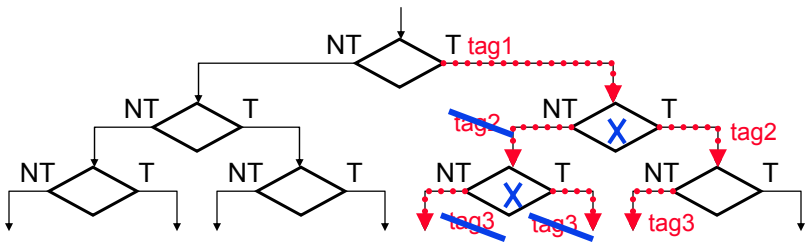


Control Flow Speculation



- Leading Speculation
  - Tag speculative instructions
  - Advance branch and following instructions
  - Buffer addresses of speculated branch instructions

Mis-speculation Recovery



- Eliminate Incorrect Path
  - Must ensure that the mis-speculated instructions produce no side effects
- Start New Correct Path
  - Must have remembered the alternate (non-predicted) path

Mis-speculation Recovery

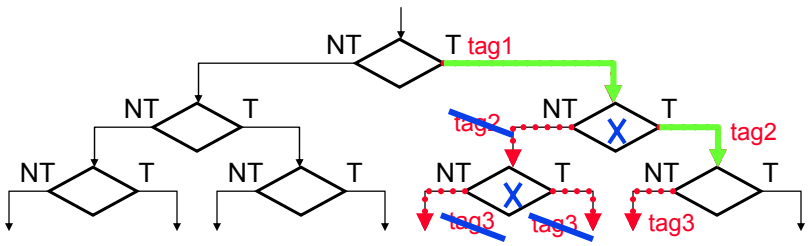
- Eliminate Incorrect Path
  - Use *branch* tag(s) to deallocate completion buffer entries occupied by speculative instructions (now determined to be mis-speculated).
  - Invalidate all instructions in the decode and dispatch buffers, as well as those in reservation stations

How expensive is a misprediction?

- Start New Correct Path
  - Update PC with computed branch target (if it was predicted NT)
  - Update PC with sequential instruction address (if it was predicted T)
  - Can begin speculation once again when encounter a new branch

How soon can you restart?

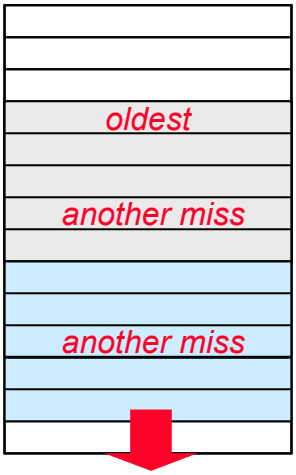
Trailing Confirmation



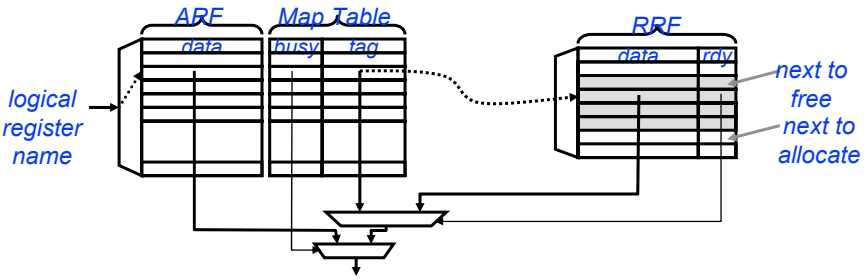
- Trailing Confirmation
  - When branch is resolved, remove/deallocate speculation tag
  - Permit completion of branch and following instructions

Fast Branch Rewind and Restart:
Metaflow DRIS

- Discard all DRIS entries (and corresponding operations) younger than the mispredicted branches
- Can restart immediately from the corrected branch target because the DRIS has sufficient information (rename & value) to continue from where left off
- Works with nested mispredictions!!



Rewinding/Flushing of Rename Table

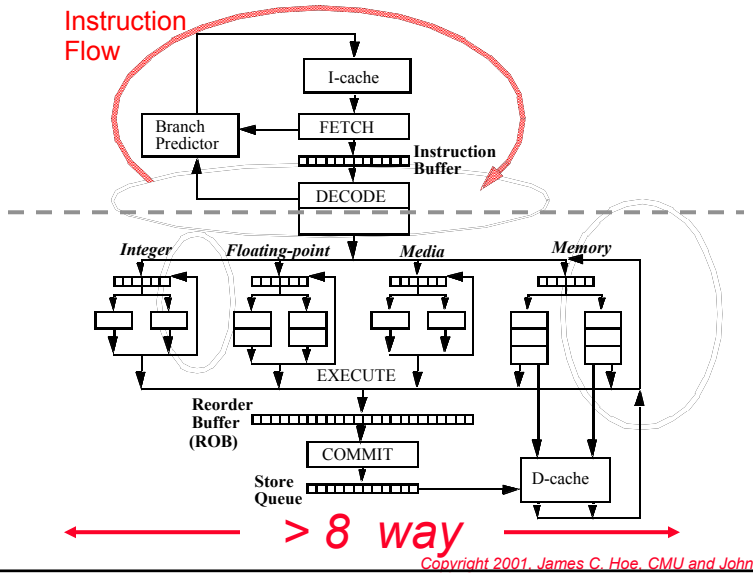


- To reinitiate renaming:
  - wait for all instructions older than the rewind point to drain clear of the pipeline and then reset register remapping to null

    Long restart latency
  - Reorder buffer has to remember how to restored the map table to the point of the mispredicted branch

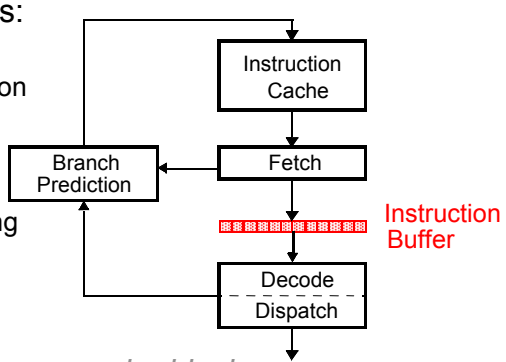
    Complicated multi-cycle logic
  - Cache rename map after branch prediction

Instruction Fetching for Wide Superscalars



Wide Instruction Fetch Issues

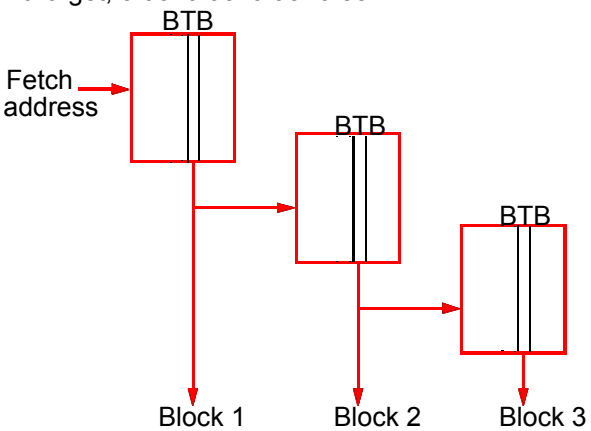
- Average Basic Block Size
  - integer code: 4-6 instructions
  - floating-point code: 6-10 instructions
- Three Major Challenges:
  - Multiple-Branch Prediction
  - Multiple Fetch Groups
  - Alignment and Collapsing



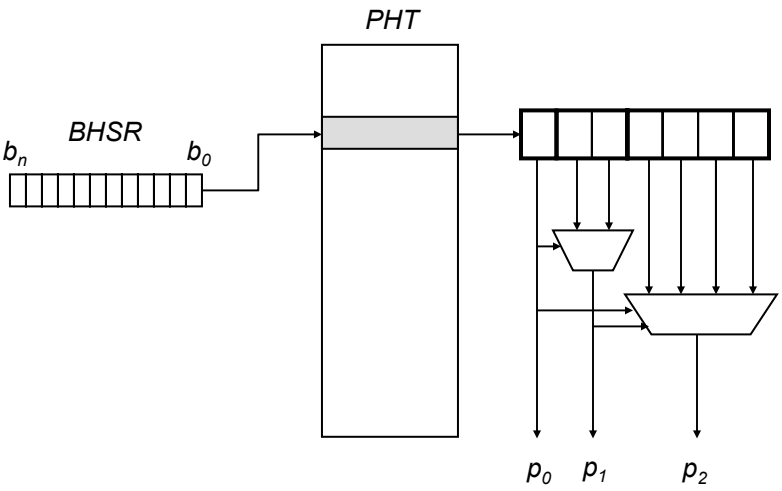
Cannot be solved with just longer cache blocks

Multiple Branch Predictions

- Issues with multiple branch predictions:
  - Latency resulting from sequential predictions
  - Later predictions based on stale/speculative history
  - Don't forget, 0.95x0.95x0.95=0.85

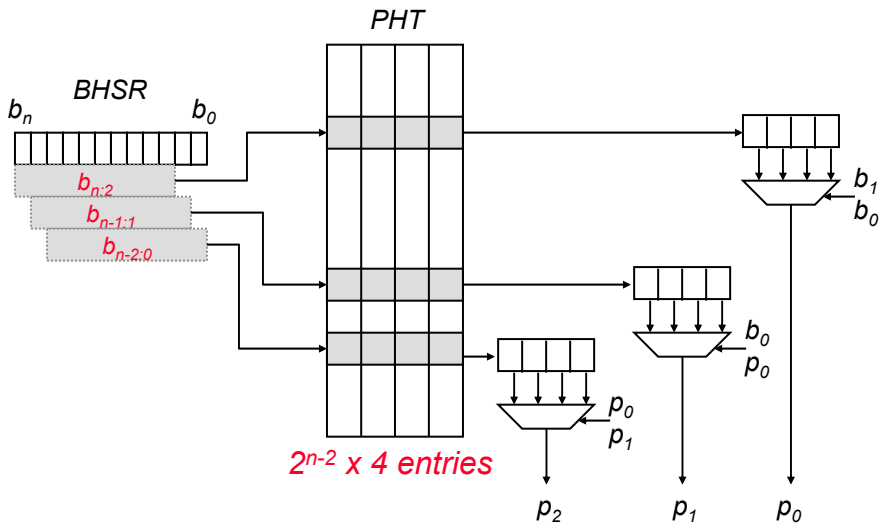


Examples of Multi-Branch Predictors



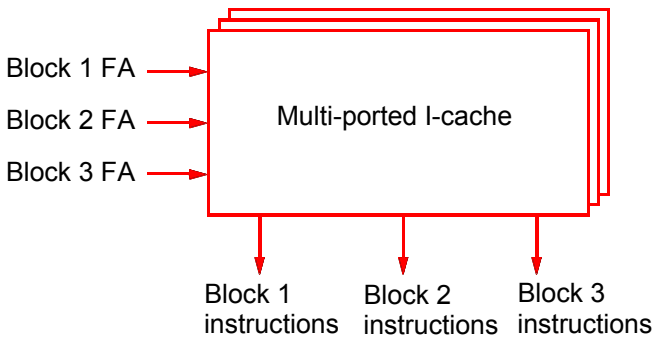
How do you update this thing after a branch resolves?

Examples of Multi-Branch Predictors



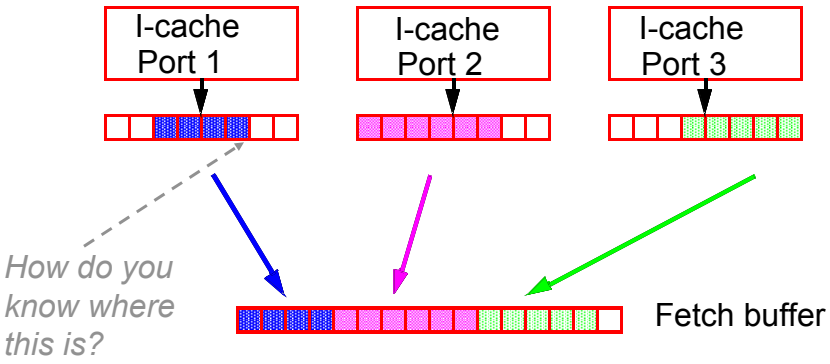
Multiple Predicted Taken Branches

- Issues with multiple taken branches:
  - Long latency with multiple sequential I-cache accesses
  - or, multi-ported I-cache with slower access latency
  - or, multi-banked I-cache to approximate multi-port

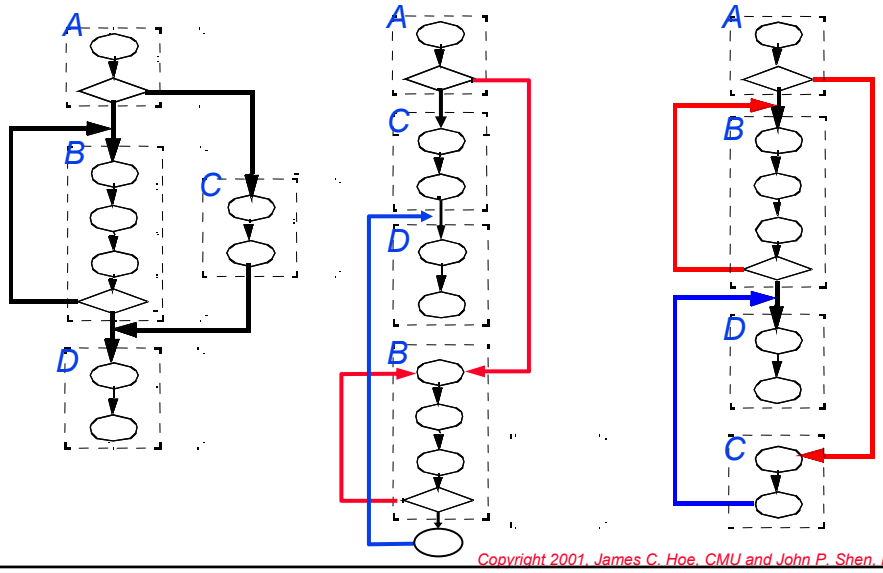


Instruction Alignment and Collapsing

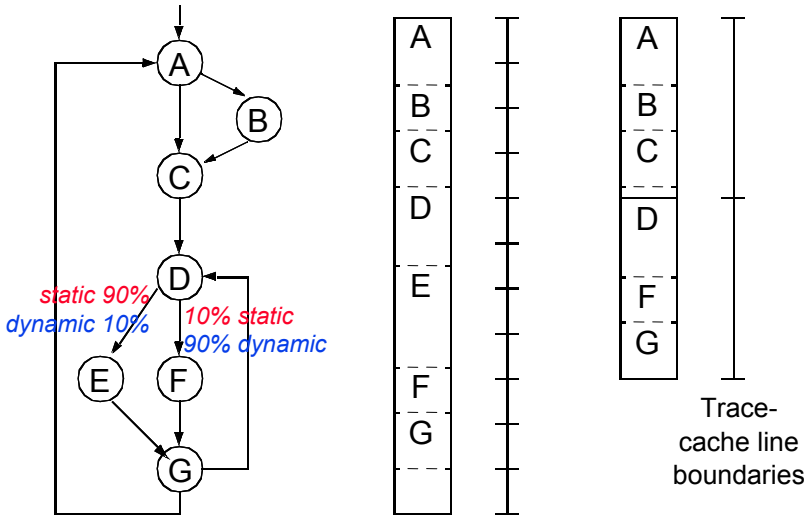
- Issues with alignment and collapsing:
  - Misalignment between fetch group and cache line.
  - Packing of variable-sized blocks into fetch buffer.



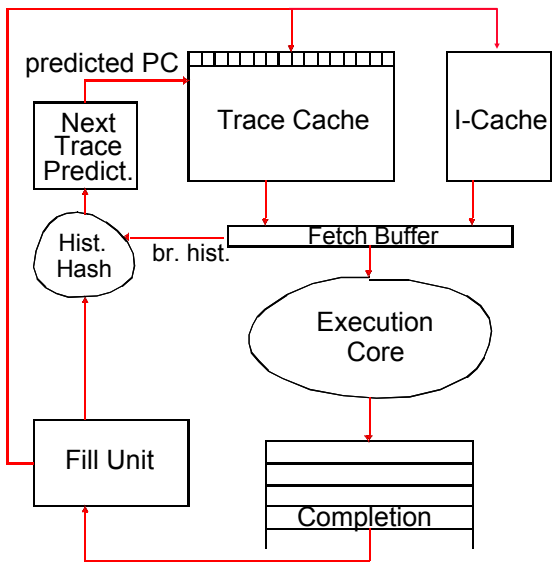
Mapping CFG to Linear Instruction Sequence



The Trace Cache Proposal



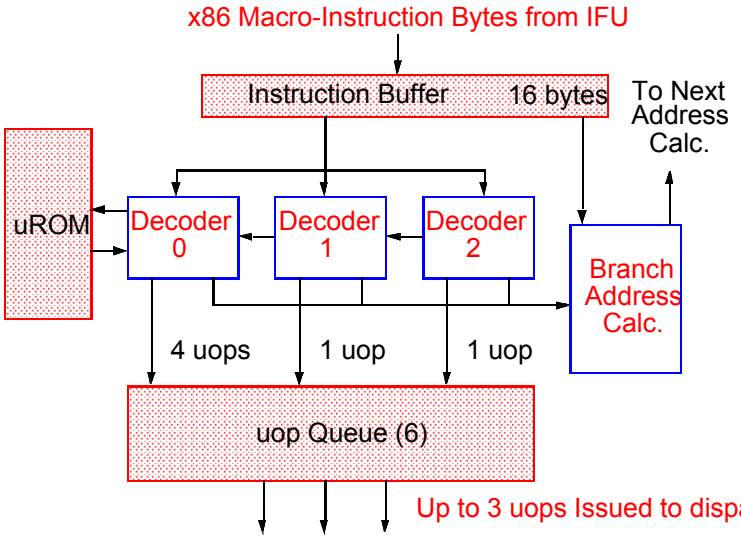
A Typical Trace Cache Organization



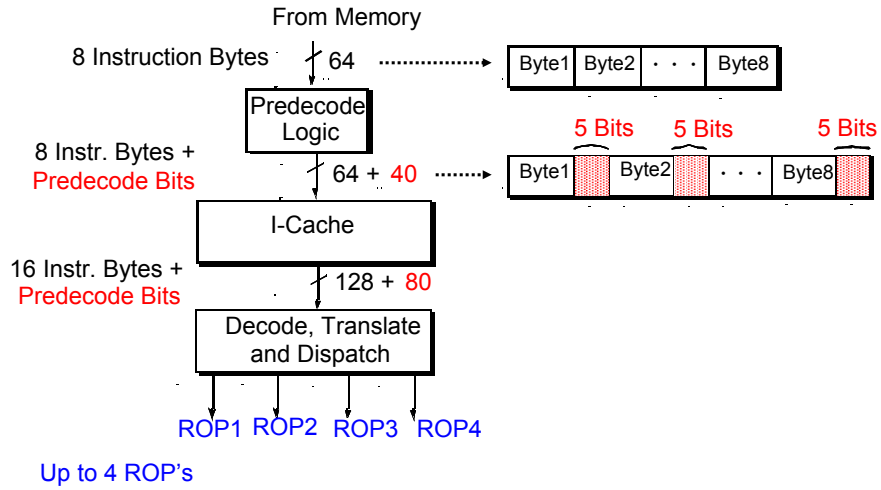
Trace Fill Unit

- Observe the dynamic execution sequence
- Gather instructions into a trace segment (or trace cache block)
- Some simple heuristics for forming trace segments
  - stop after collecting up to *N* instructions
    (*N* is the trace cache block size)
  - stop after *B* conditional branches
    (*B* is the limit of the multi-branch predictor)
  - stop after seeing an register-indirect jump
  - Don't split basic blocks
  - In some designs, unconditional and conditional branches can be dropped from the traces
- Can include pre-decoded dependence information
- Can even dynamically re-order instructions (don't need an out-of-order core!!)

Intel Pentium Pro Fetch/Decode Unit



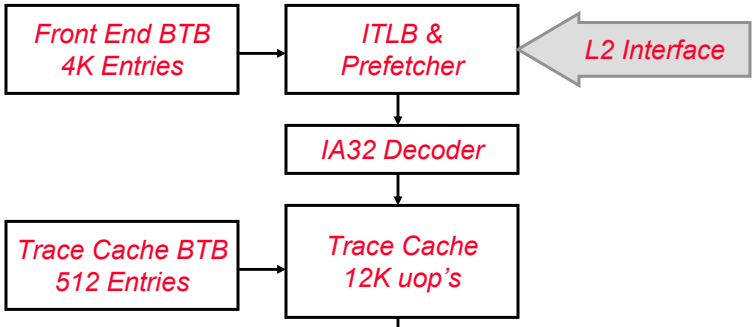
Predecoding in the AMD K5



Intel P4 Trace Cache

- A 12K-uop trace cache replaces the L1 I-cache
- 6-uop per trace line, can include branches
- Trace cache returns 3-uop per cycle
- IA-32 decoder can be simpler and slower

Only needs to decode one IA-32 instruction per cycle



Trace Selection/Prediction

- Basic
  - find the trace that starts at the predicted next-PC
- Multiple cached traces may have the same starting PC
  - difference is in the internal branch decisions

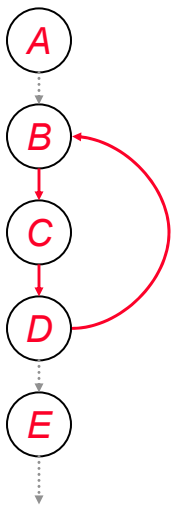
  ⇒ *need multi-branch predictors*
- Partial Traces
  - predicted next-PC points to the middle of a cached trace (cached ABC, but predicted BC)
  - multi-branch prediction may say not to use the entire length of a cached trace (cached ABC, but only needs AB)

  ⇒ *need alignment and collapsing buffer*

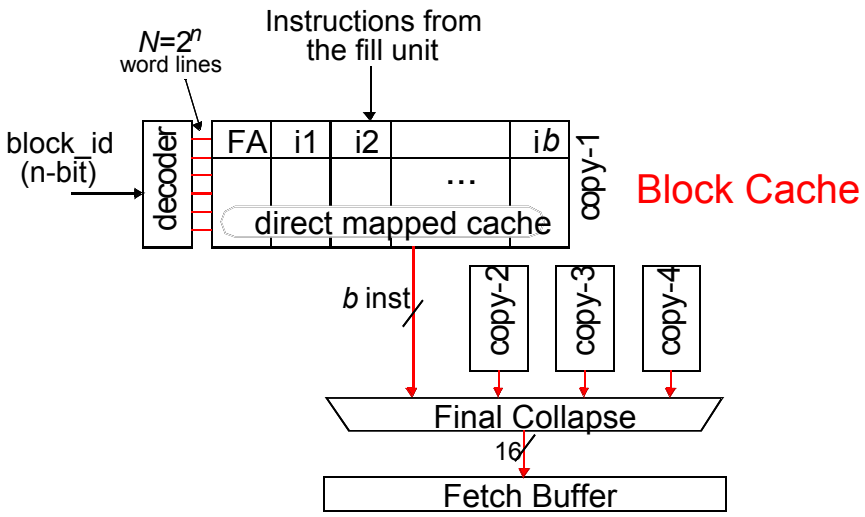
So how is this better?

Redundant Traces

- Suppose B,C and D are instructions in a loop
  - 3 different traces of 3-instructions are possible
  - Which one should we keep in the trace cache?
  - How do we detect the beginning and the end of basic blocks?
- Suppose A,B,C,D and E are basic blocks
  - don't cache BC if BCD is cached
  - what about CDB and CDE?
  - what about ABC and DBC?
  - How to cut down on redundant instruction storage?

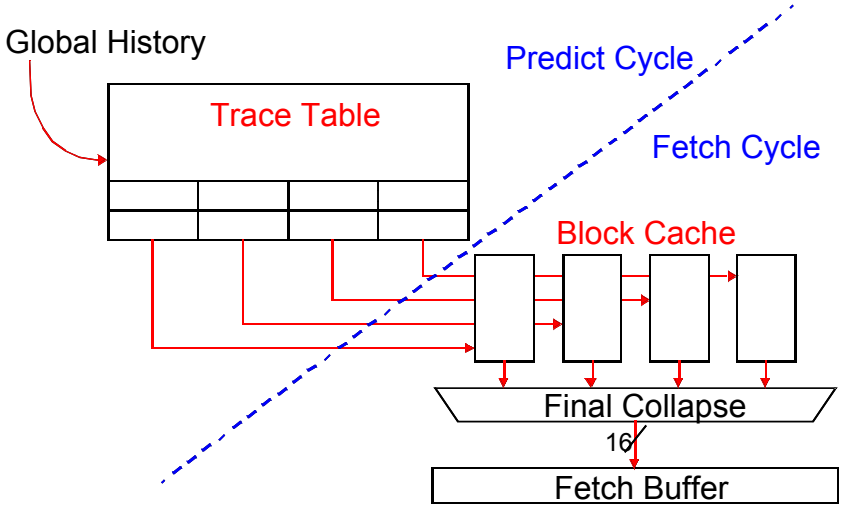


Replicated Block Cache



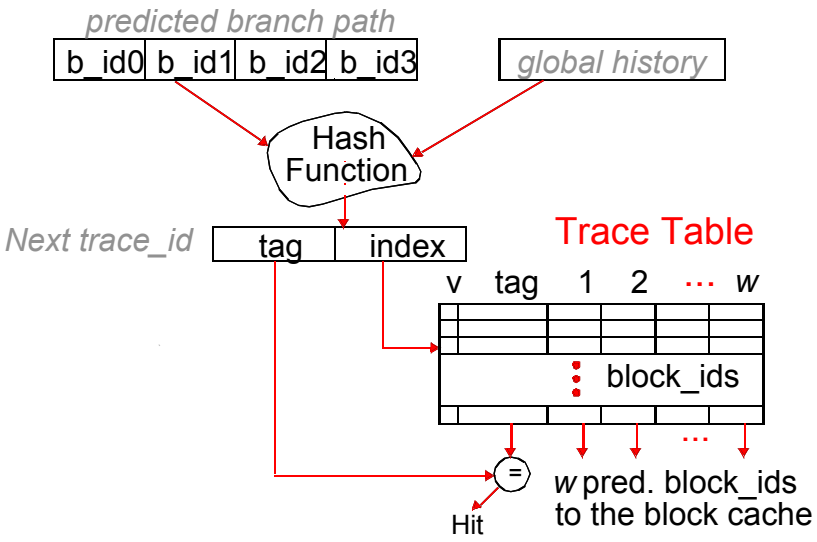
What about fragmentation?

Predict and Fetch Trace

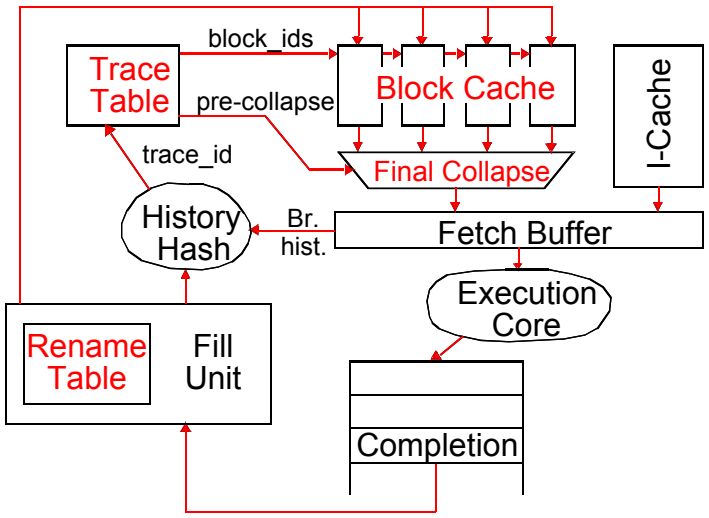


More efficient: redundancy is in the trace table and not the block cache

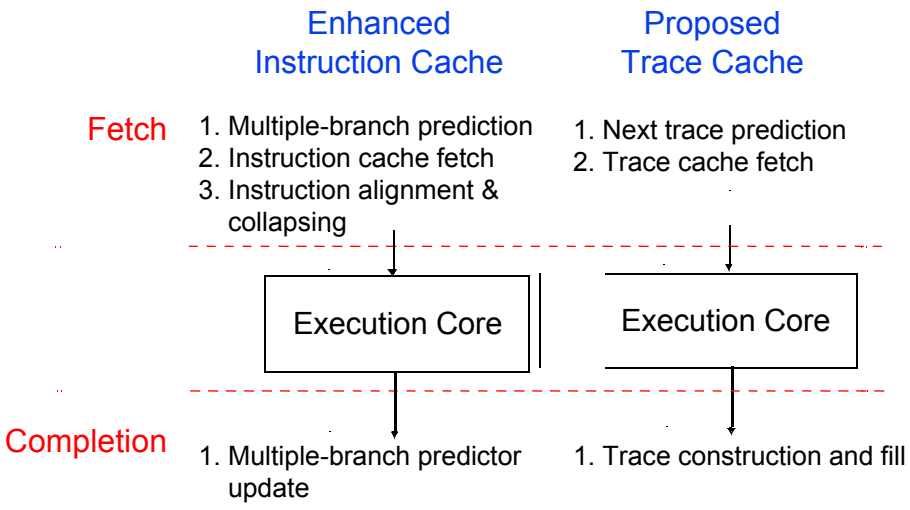
Next Trace Prediction



The Block-Based Trace Cache



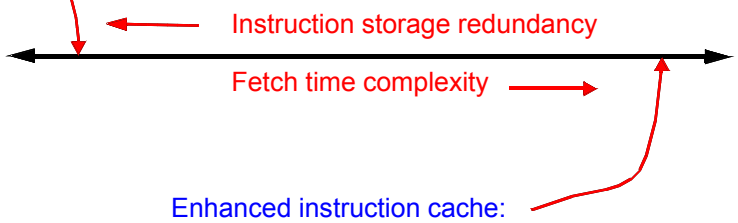
Wide-Fetch I-cache vs. T-cache



Trace Cache Trade-offs

Trace cache:

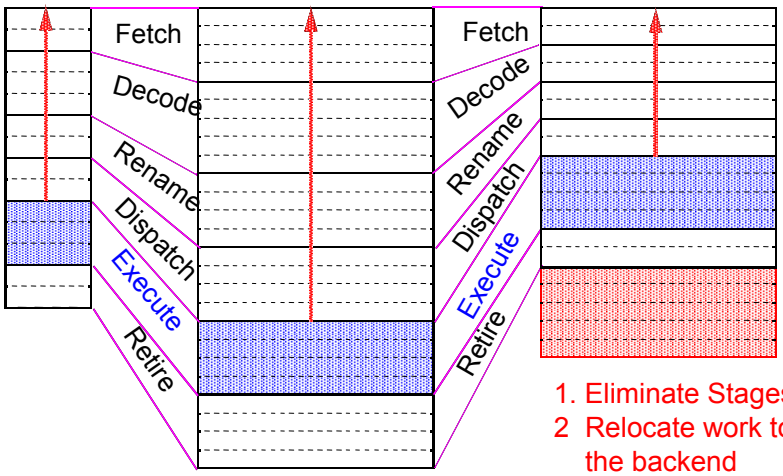
Pros → Moves complexity to backend
Cons → Inefficient instruction storage



Enhanced instruction cache:

Pros → Efficient instruction storage
Cons → Complexity during fetch time

As Machines Get Wider (… and Deeper)



1. Eliminate Stages
2. Relocate work to the backend