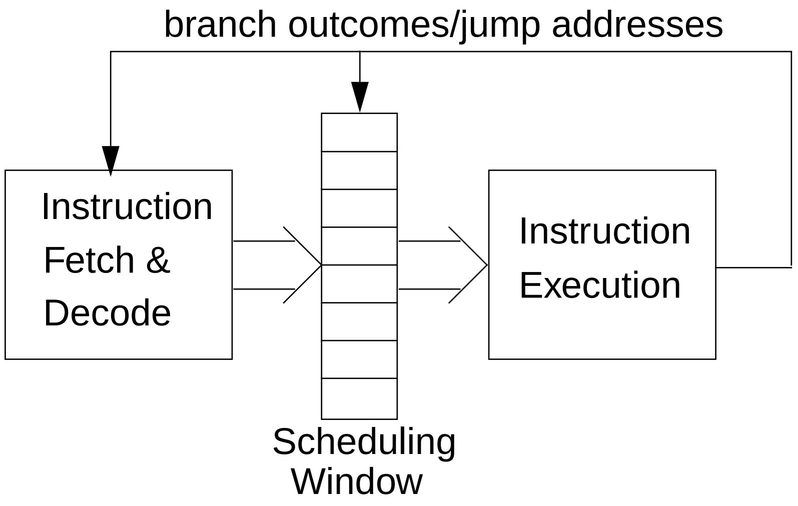
Trace Cache: Low Latency, High Bandwidth Instruction Fetching

Eric Rotenberg, Steve Bennett, Jim Smith

Eric:	Steve:	Jim:
Computer Science Dept. Univ. of Wisconsin — Madison	Intel Corporation	Dept. of Elec. and Comp. Engr. Univ. of Wisconsin — Madison

http://www.cs.wisc.edu/~ericro/ericro.html

Superscalar Processors

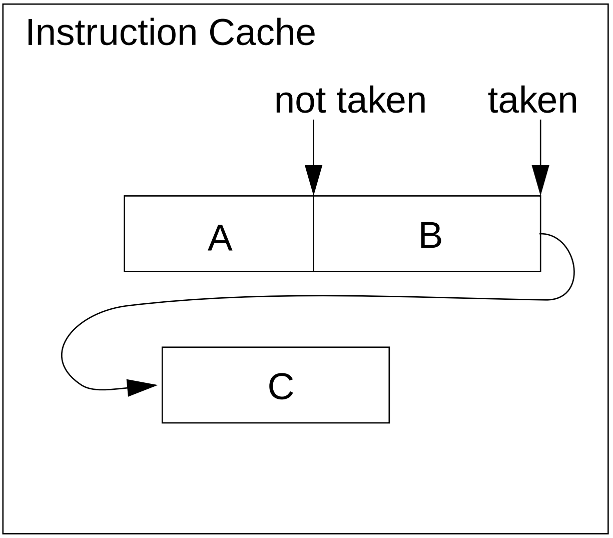


- Divide into instruction fetch and execution
- Superscalar trends:
  - larger scheduling window
  - wider dispatch/issue
  - deeper speculation
  - more functional units
- Implications for instruction fetch?

Overview

- Instruction fetch issues
- Trace cache
- Alternatives
- Simulation Results
- Conclusions

Instruction Fetch Issues

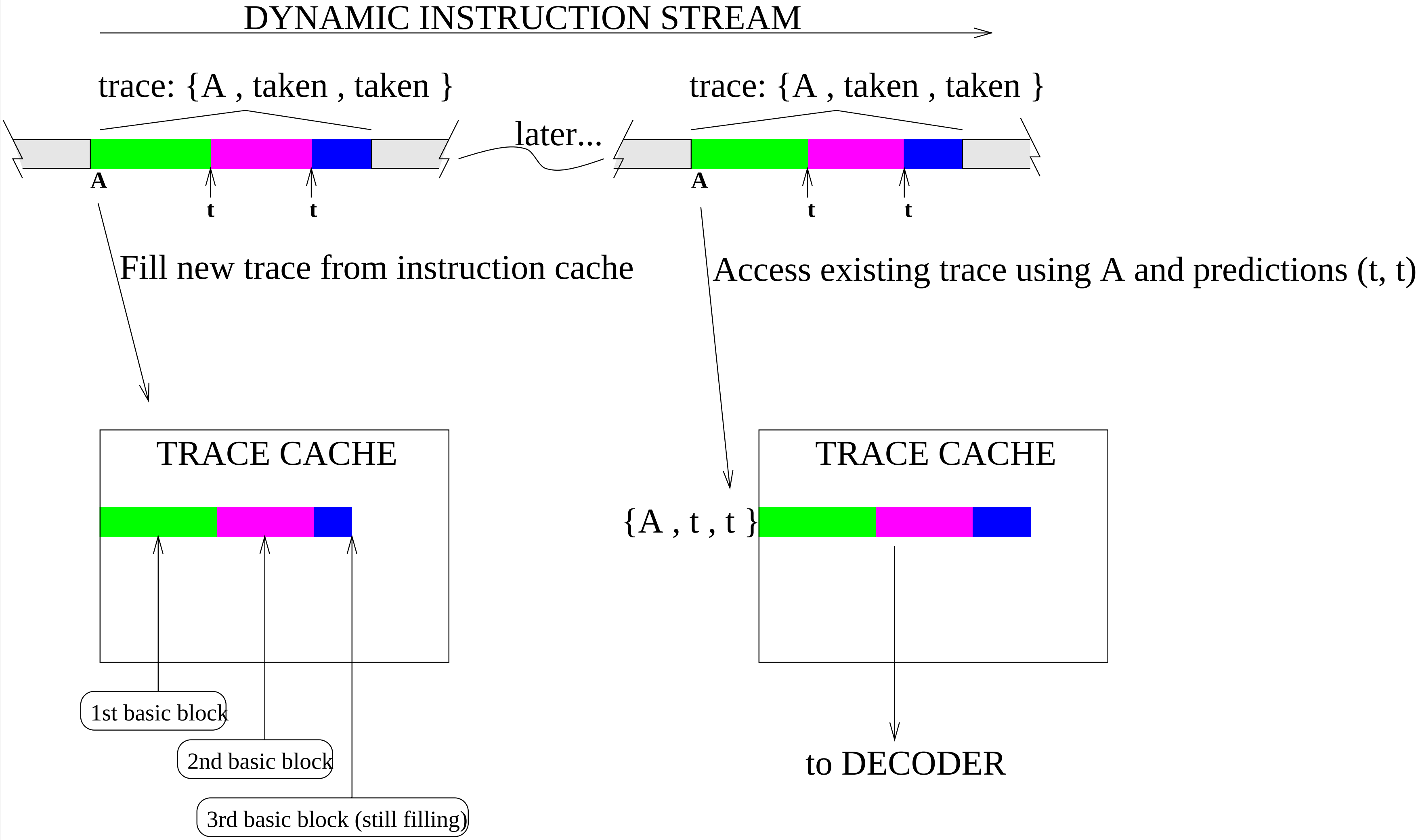


- Branch Throughput
  - predict more than one per cycle
- Noncontiguous instruction blocks
  - fetch past taken branches
- Fetch unit latency
  - can’t be ignored when solving others
- “Conventional” issues:
  - instruction cache misses
  - branch prediction accuracy
  - NOT the focus of this study

Noncontiguous Instructions

- Fundamental problem:
  - conventional instruction cache stores programs in their static, compiled order
  - the decoder wants to see the instruction stream in its dynamic order
- Two classes of hardware solutions:
  - deal with conventional cache, construct dynamic order on the fly
  - direct approach: cache instructions in dynamic order
- We propose the direct approach

Trace Cache High Level



DYNAMIC INSTRUCTION STREAM

trace: {A , taken , taken }

later...

trace: {A , taken , taken }

Fill new trace from instruction cache

Access existing trace using A and predictions (t, t)

TRACE CACHE

TRACE CACHE

{A , t , t }

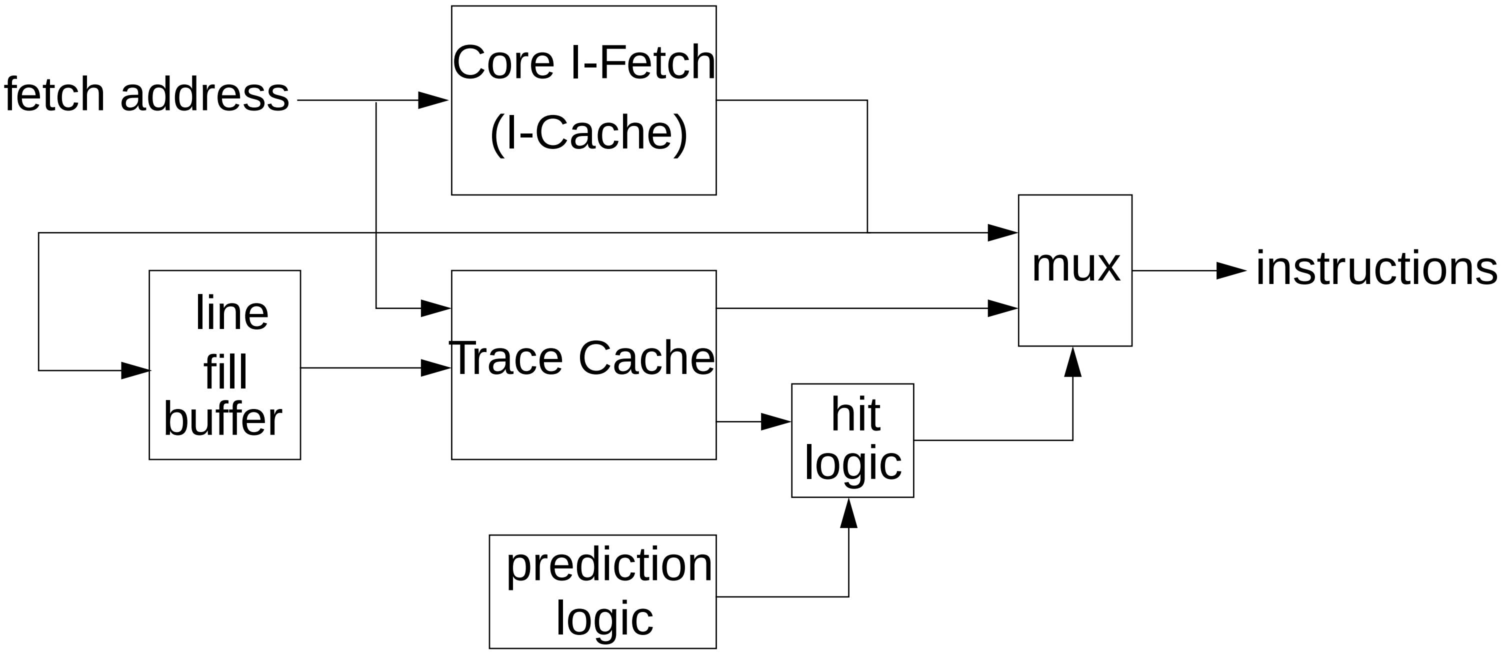
1st basic block

2nd basic block

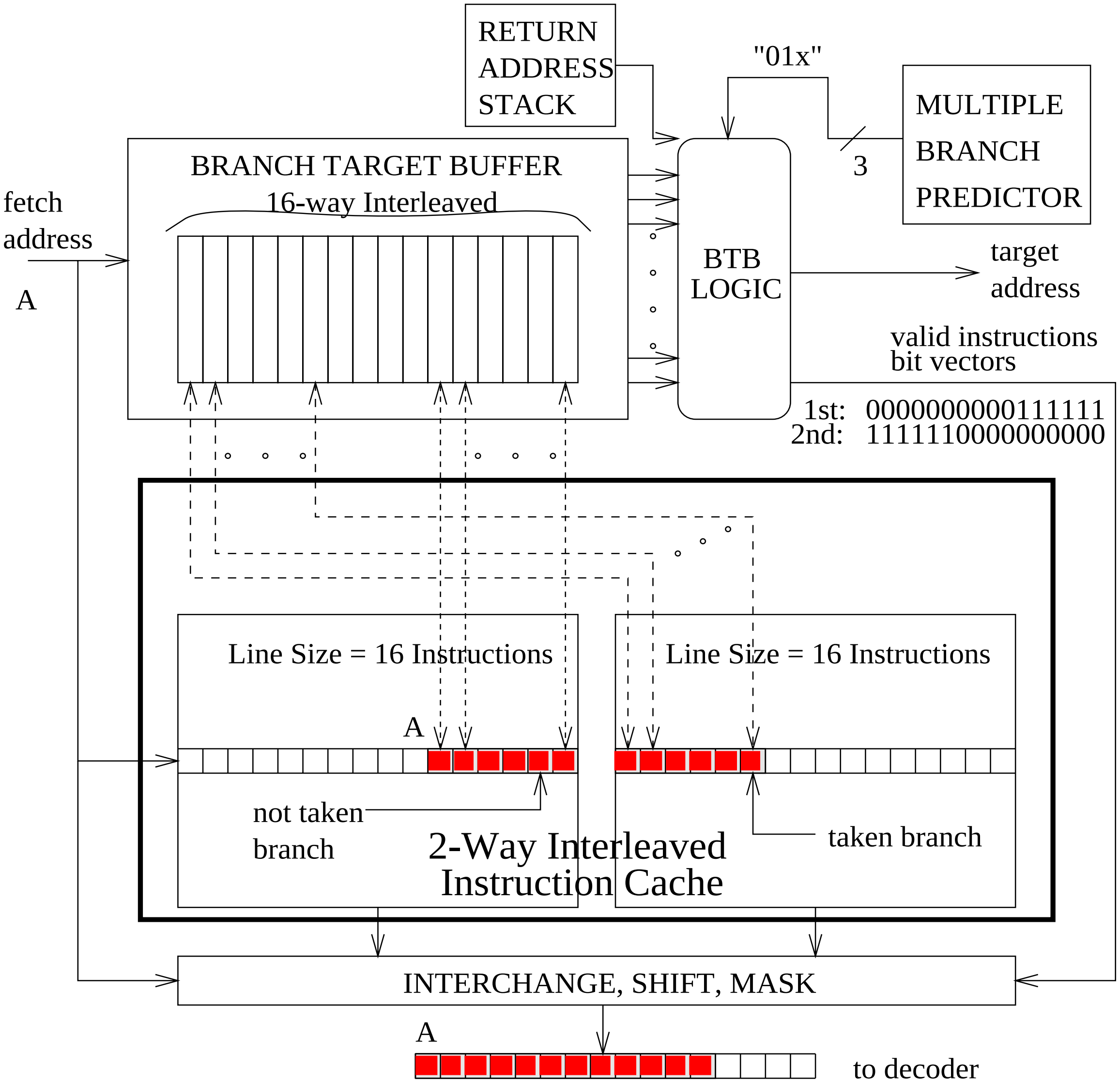
3rd basic block (still filling)

to DECODER

Trace Cache Overview

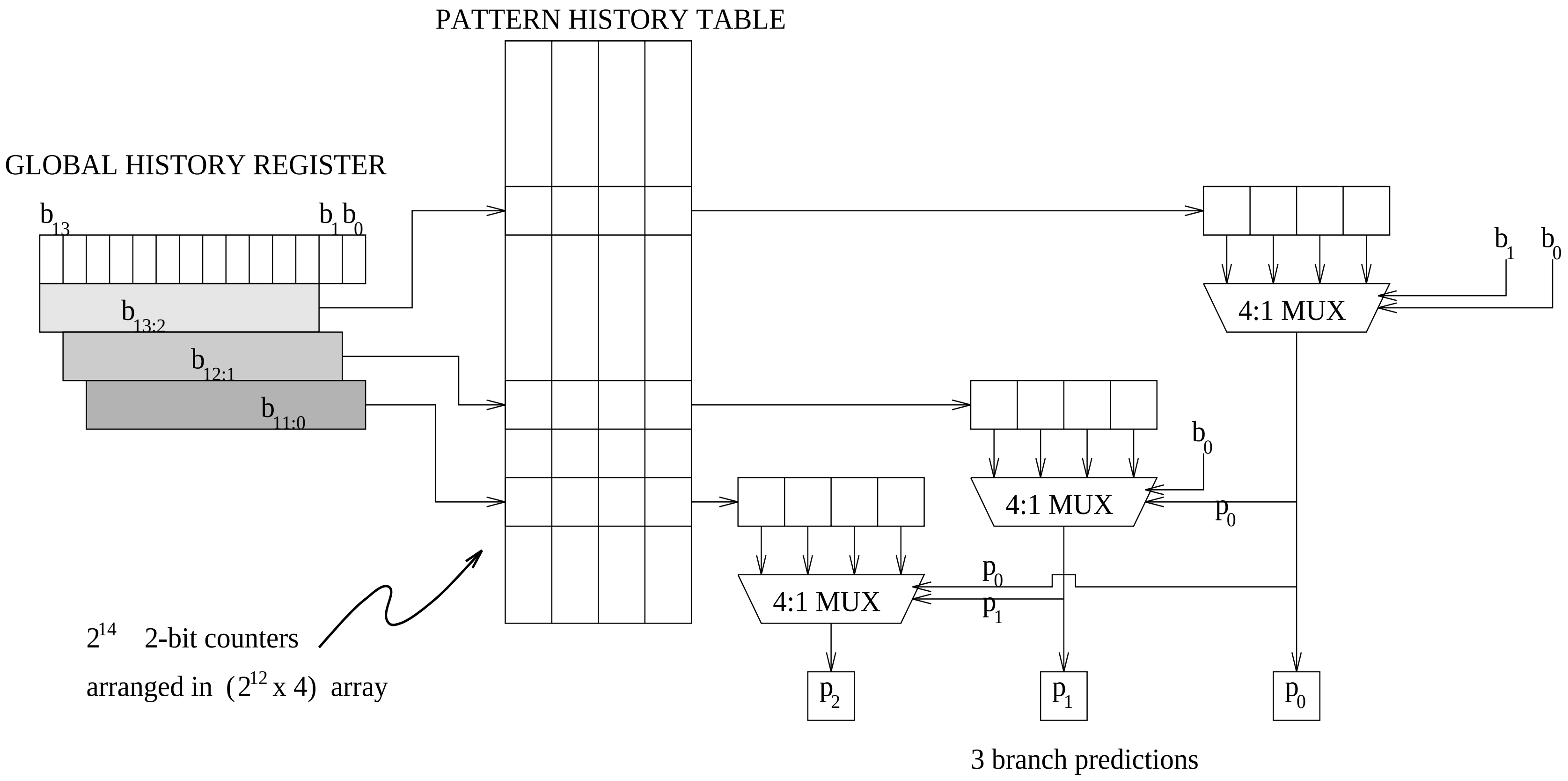


Core Fetch Unit Detail



Predicting Multiple Branches

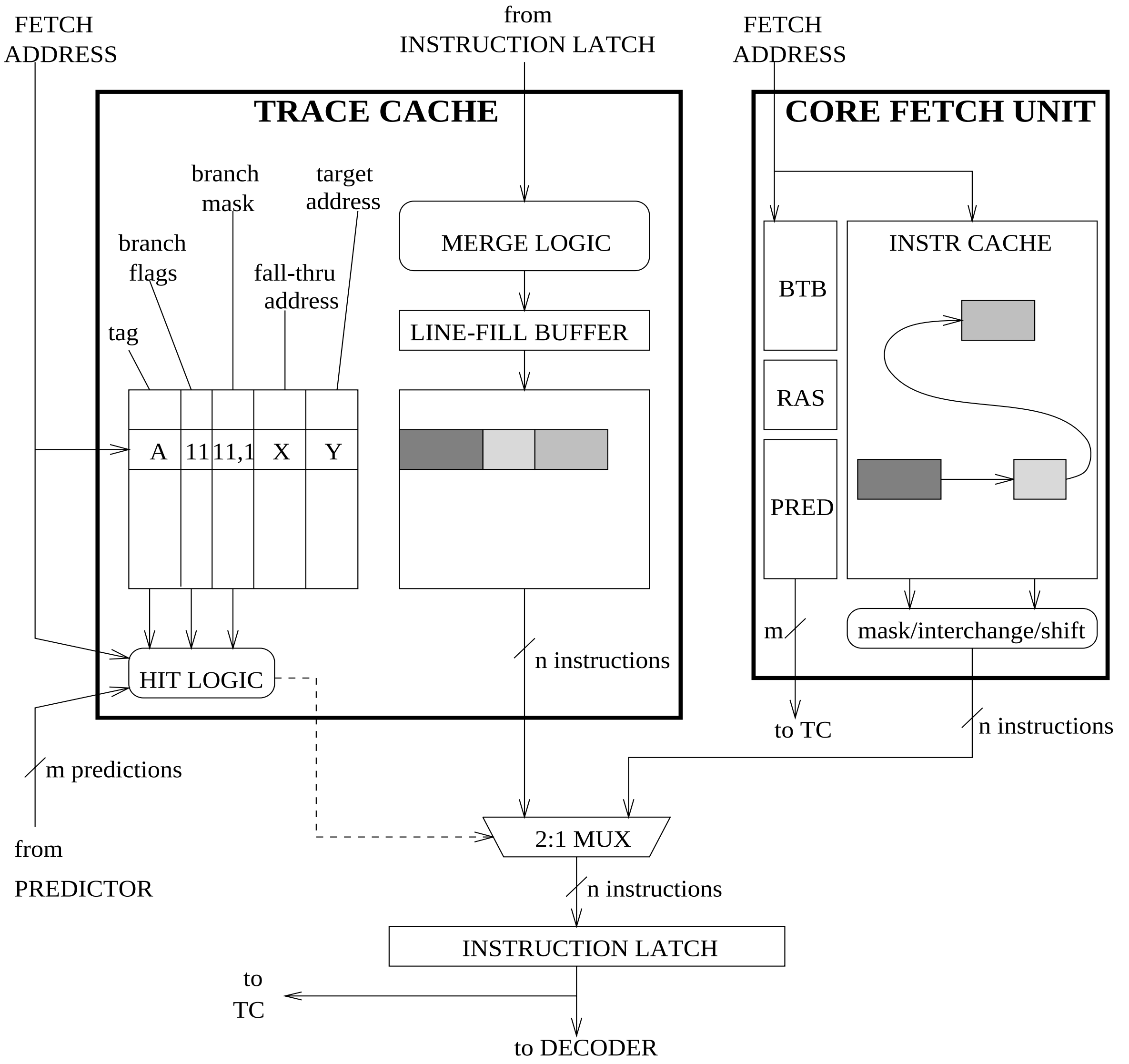
- Multiple correlating predictor (Yeh, Marr, Patt):



Trace Cache Contents

- Parameters: n max instructions, m basic blocks
- In addition to tag and instructions, contains:
  - Branch flags: m-1 branch outcomes
  - Branch mask: number of branches
  - Trace fall-thru: next address if last branch is not-taken
  - Trace target: next address if last branch is taken
- Trace Cache may be small, we use:
  - 64 lines
  - direct mapped
  - n = 16, m = 3
  - total size: 712B (tag info) + 4KB (instr.)

Trace Cache Detail



Alternatives

- Branch Address Cache (Yeh,Marr,Patt)
- Collapsing Buffer (Conte,Menezes,Mills,Patel)
- Subgraph Predictor (Dutta,Franklin)
- Two-Block Ahead Predictor (Seznec,Jourdan,Sainrat,Michaud)

Weakness of Alternatives

- General idea behind alternatives:

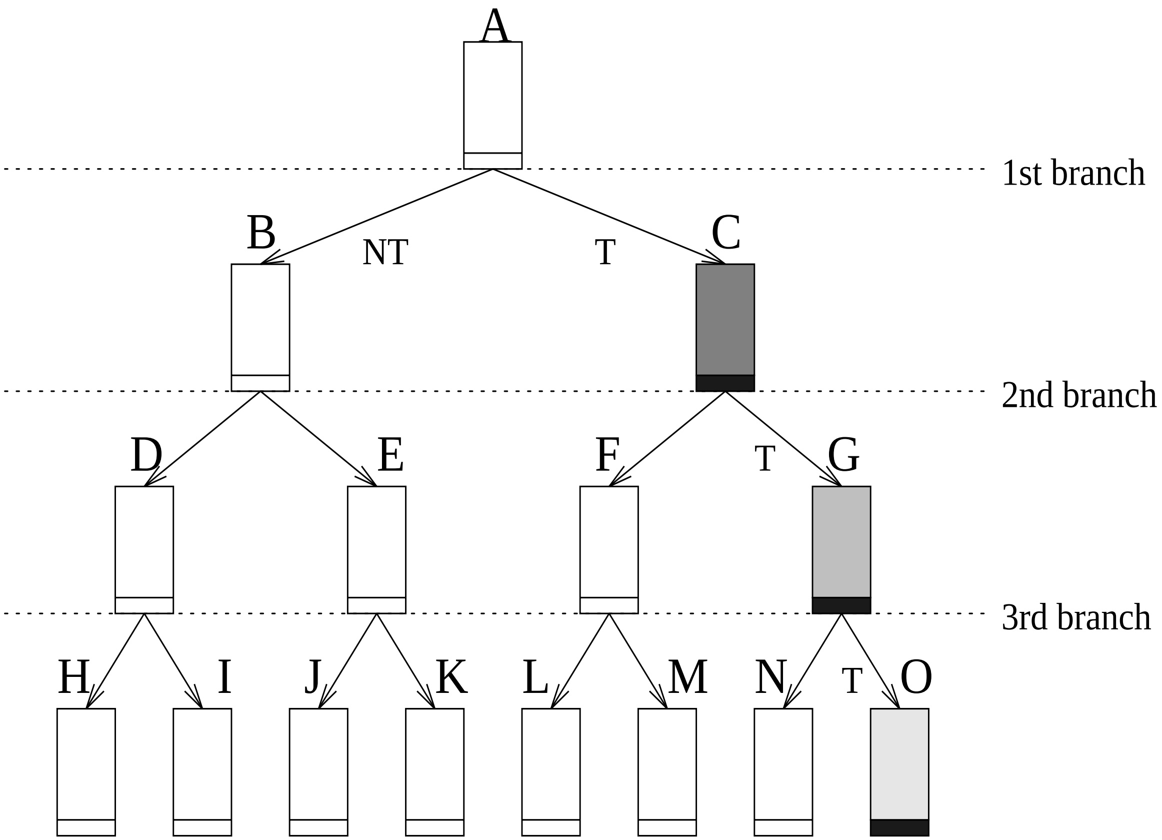
  1. Generate multiple addresses pointing to several, possibly noncontiguous basic blocks.
  2. Apply the multiple addresses to an interleaved or multiported instruction cache.
  3. Properly order and merge only desired instructions into the predicted dynamic sequence.

  => complex and serial, long latency

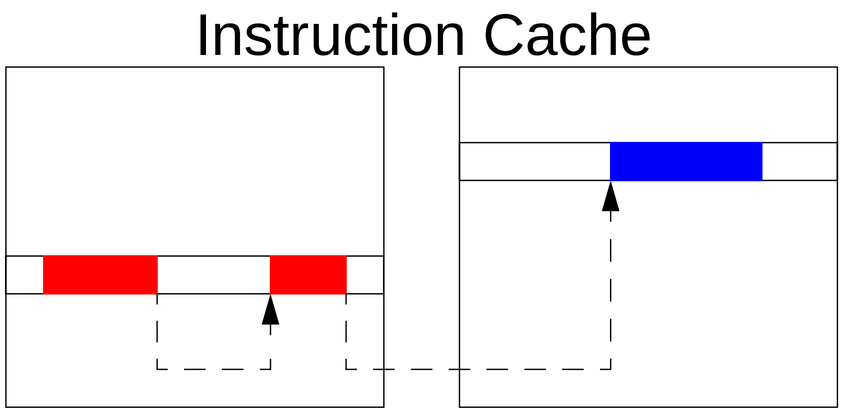
- Trace cache: cache long instruction sequences in the order they are likely to be needed

  => moves complexity off critical path, short latency

Branch Address Cache



Collapsing Buffer



Instruction Cache

- BTB design can detect intraline branches
- Two passes through BTB allow up to one interline branch
- Collapsing Buffer uses control info from both BTB passes to align instructions into dynamic sequence

Simulation Study

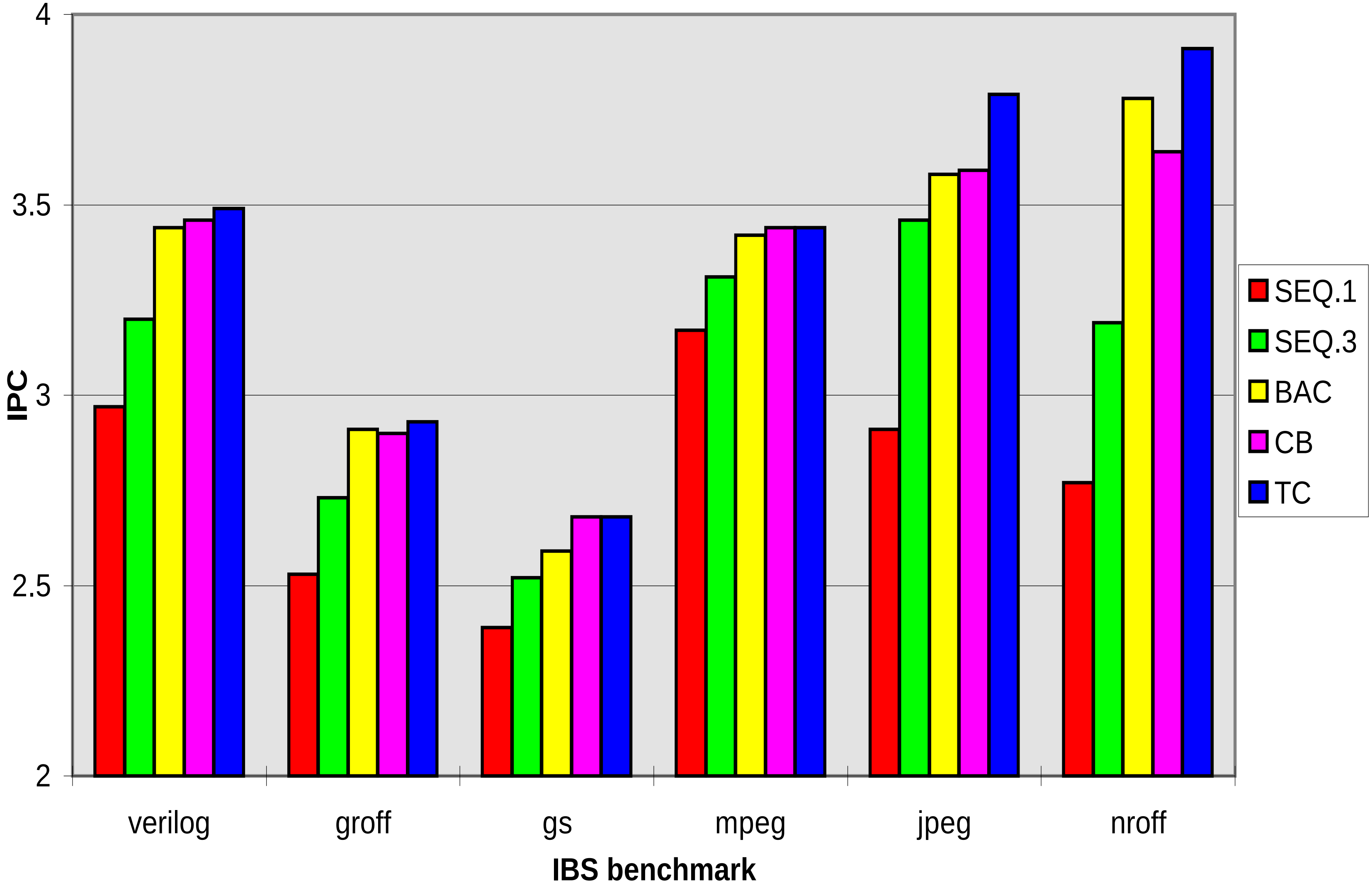
- Superscalar model as in introduction
- Maximum demand from execution engine:
  - Execution engine limited only by true data dependences
  - Oracle memory address disambiguation
  - Data cache always hits
- Instruction window: 2048
- Max dispatch/issue bandwidth: 16
- Workload: SPEC (sparc) and IBS (mips)
- Trace driven (wrong speculations not followed)

Simulation Study

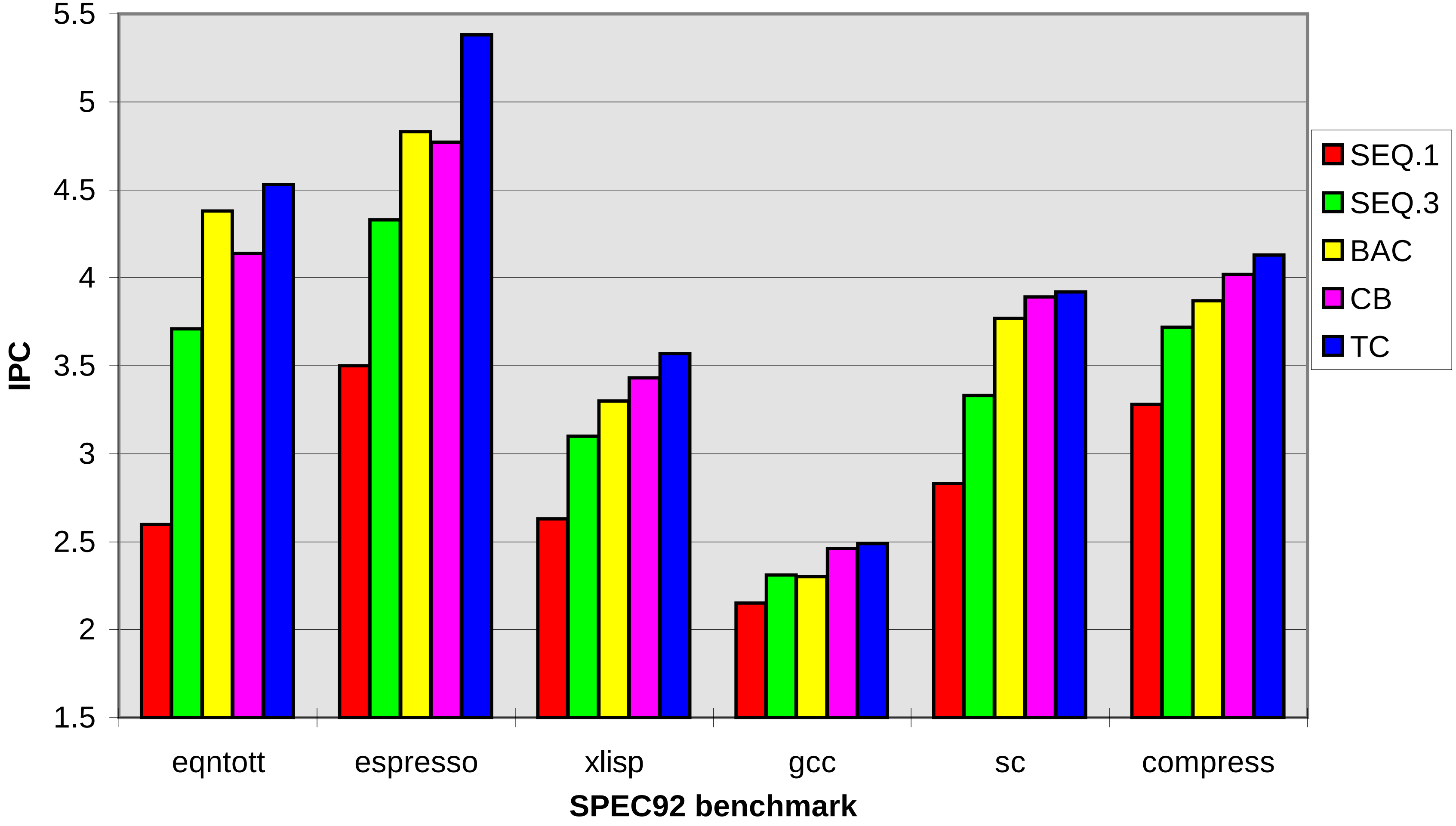
Compare various fetch mechanisms

- Two base fetch models for comparison:
  - can fetch only sequential code
  - SEQ.1 : limited to 1 branch prediction per cycle
  - SEQ.3 : limited to 3 branch predictions per cycle
- Three models for high bandwidth instruction fetching:
  - TC : trace cache
  - BAC : branch address cache
  - CB : collapsing buffer
- Model with 1, 2, and 3 cycle latencies for CB and BAC

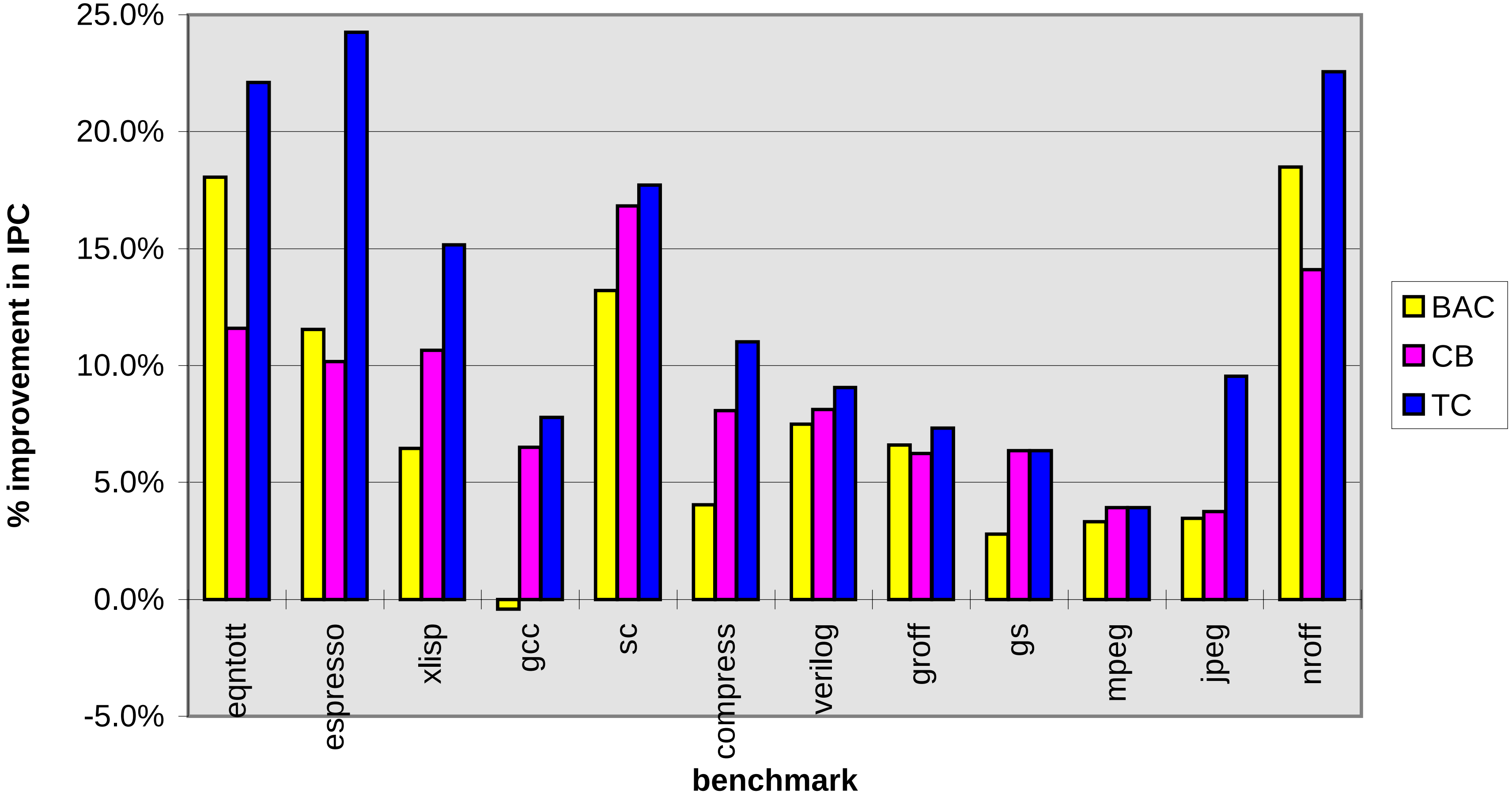
IPC for the Various Fetch Mechanisms, Single-Cycle Latency

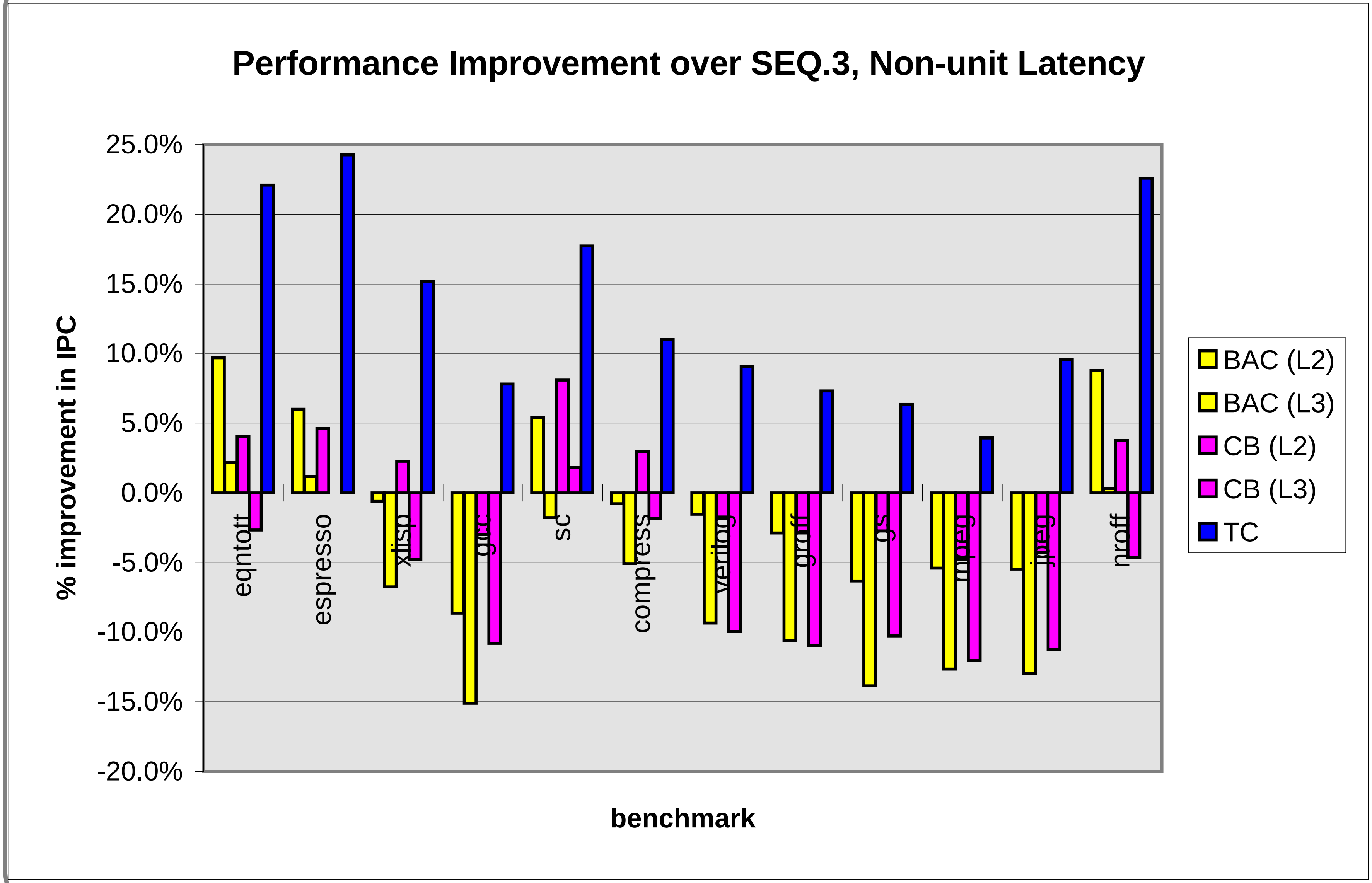


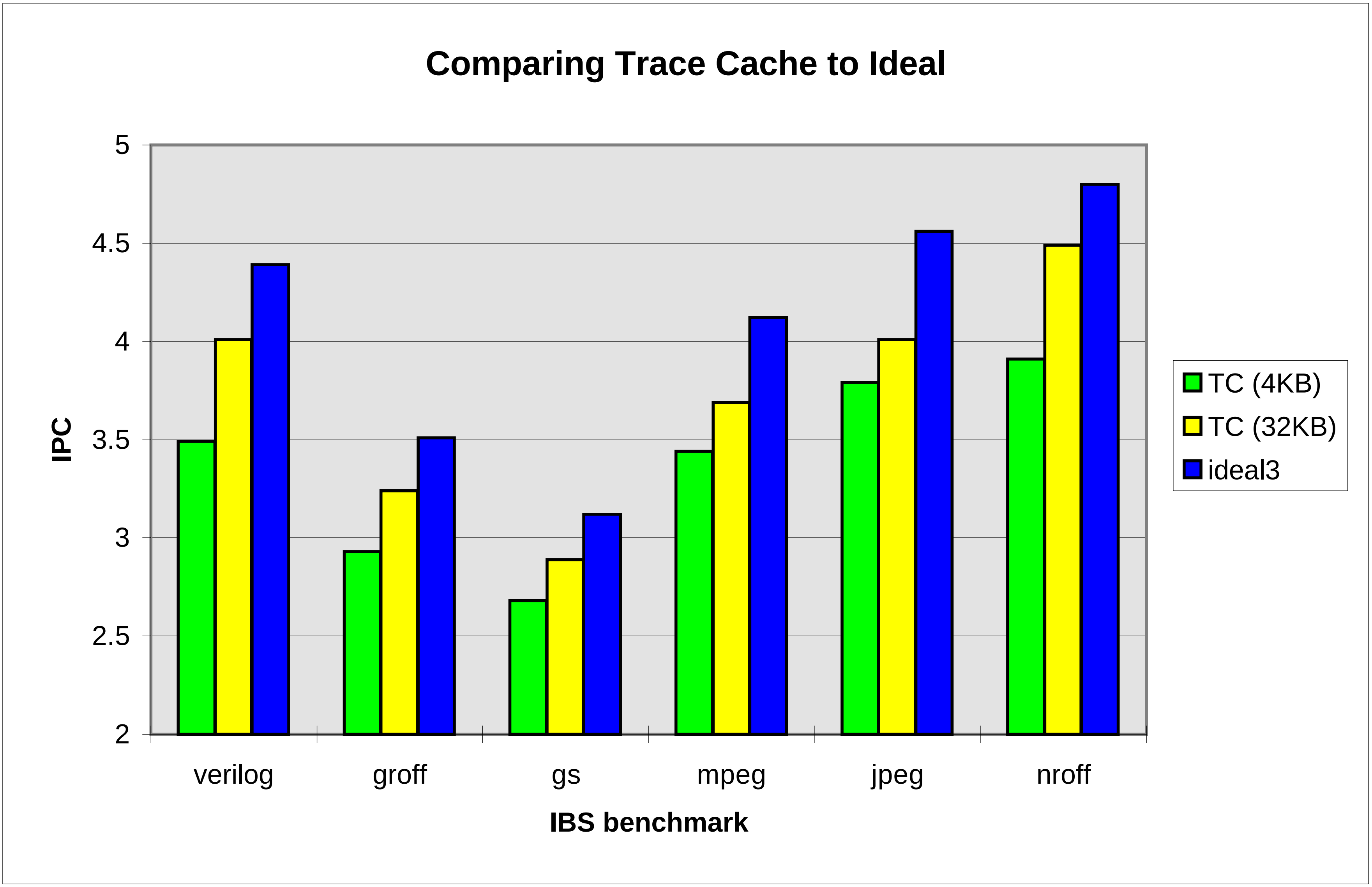
IPC for the Various Fetch Mechanisms, Single-Cycle Latency



Performance Improvement over SEQ.3, Single-Cycle Latency




Performance Improvement over SEQ.3, Non-unit Latency
25.0%
20.0%
15.0%
10.0%
5.0%
0.0%
-5.0%
-10.0%
-15.0%
-20.0%
% improvement in IPC
eqntott
espresso
xlisp
gcc
sc
compress
verilog
groff
gs
mpeg
jpeg
nroff
benchmark
BAC (L2)
BAC (L3)
CB (L2)
CB (L3)
TC


Comparing Trace Cache to Ideal
IPC
5
4.5
4
3.5
3
2.5
2
verilog
groff
gs
mpeg
jpeg
nroff
IBS benchmark
🟩 TC (4KB)
🟨 TC (32KB)
🟦 ideal3

Conclusions

- Fetching past multiple not-taken branches improves performance by >10%
- A small trace cache gets an additional 10% or more by going past multiple taken branches
- Trace cache is consistently better than other proposed methods with similar goals assuming unit latency, much better with realistic latencies
- Trace cache is a natural trend towards reducing superscalar front-end complexity