Execution-based Prediction Using Speculative Slices

Craig Zilles and Guri Sohi

University of Wisconsin - Madison

International Symposium on Computer Architecture

July, 2001

The Problem

Two major barriers to achieving high ILP:

MISPREDICTED BRANCHES and CACHE MISSES

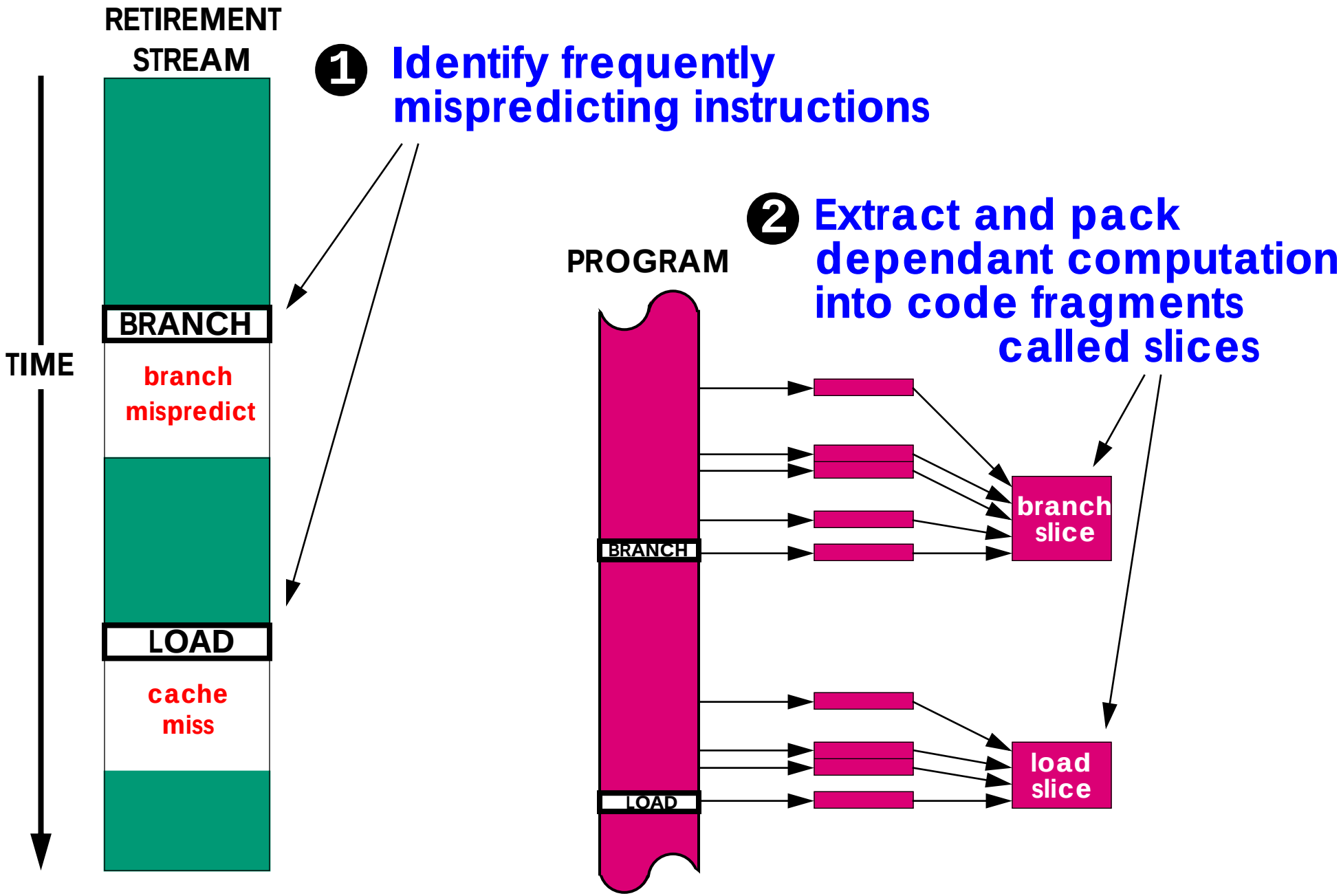
TRADITIONAL PREDICTION: SOMEWHAT MATURE TECHNOLOGY

- **correctly anticipate > 90% instructions**
- **exploit patterns in outcome/address stream**
- **remaining mispredictions still expensive**

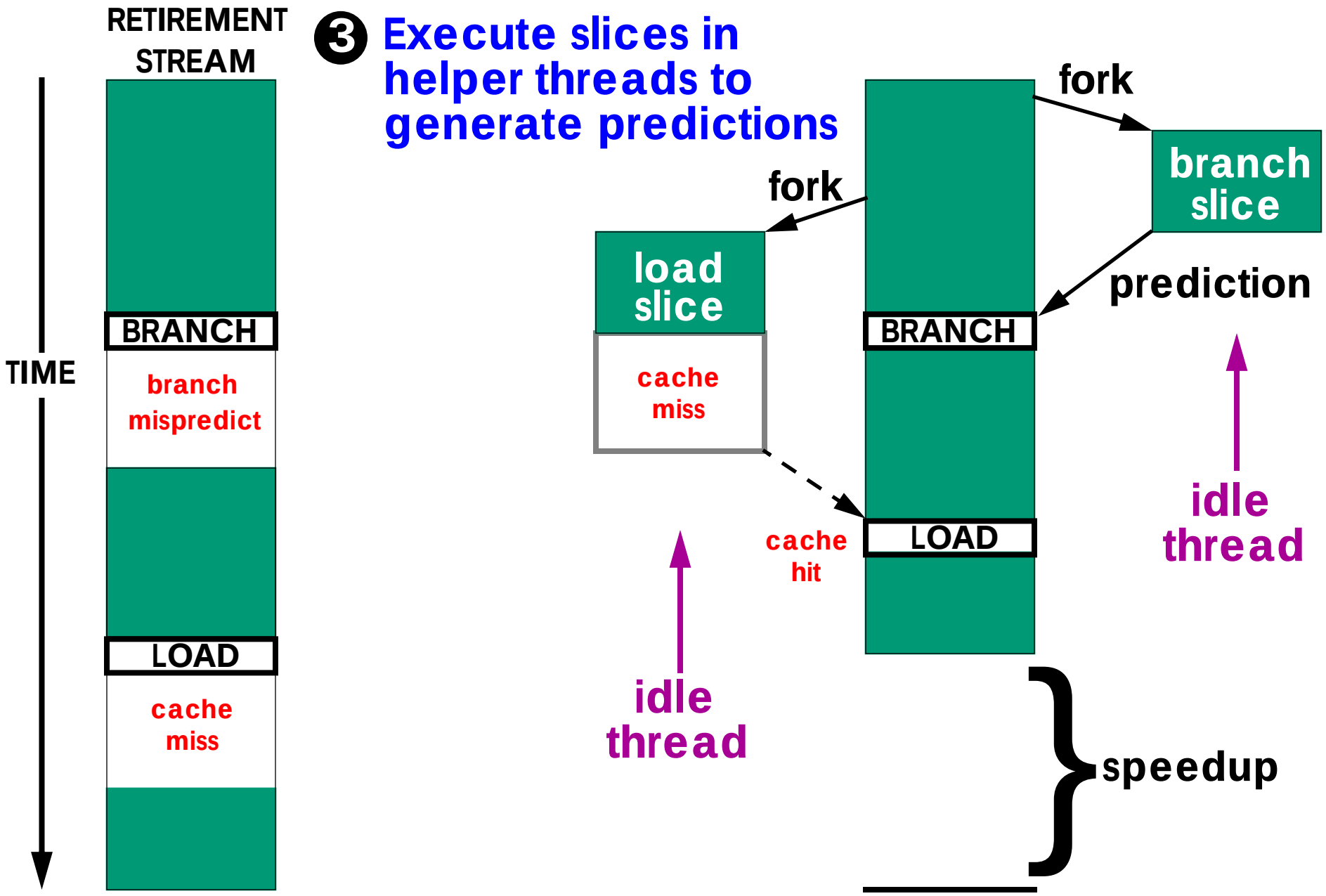
EXECUTION-BASED PREDICTION

- **exploit regularity in computations**
- **speculatively compute results early for use as predictions**
- **speedups from 1 to 43% on SPECINT 2000**

The Solution

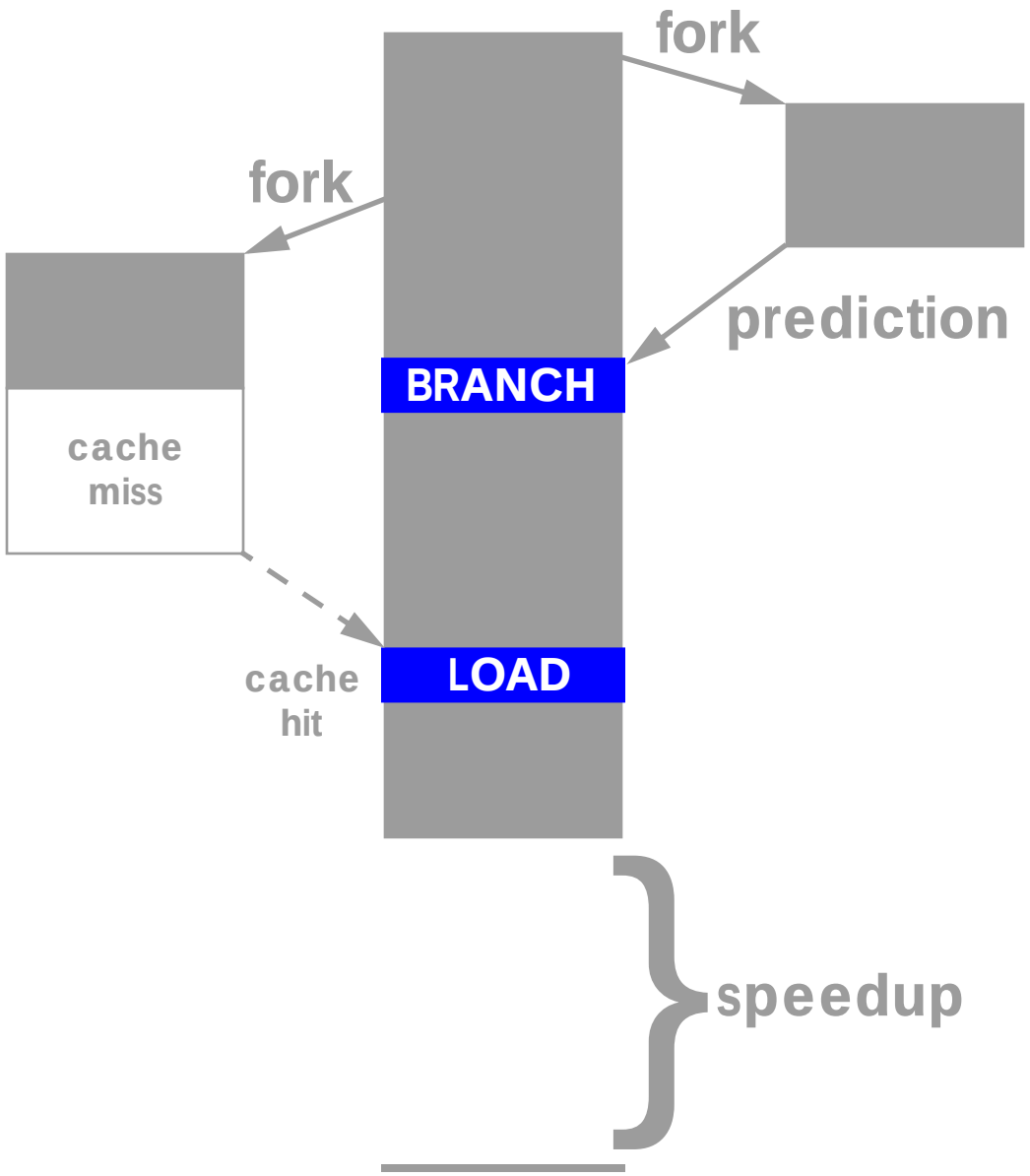


The Solution



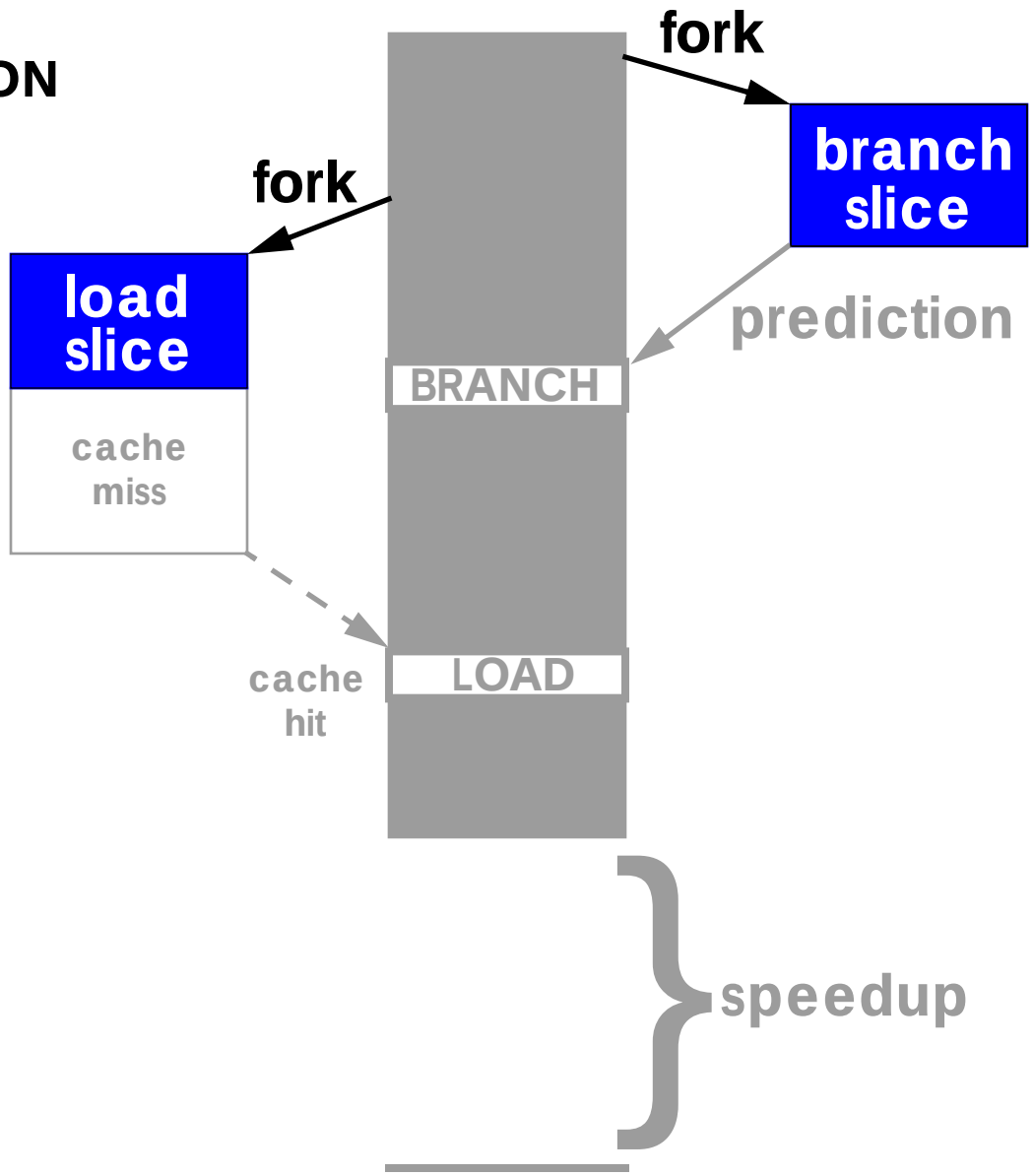
The Outline

- **Problem Instructions**



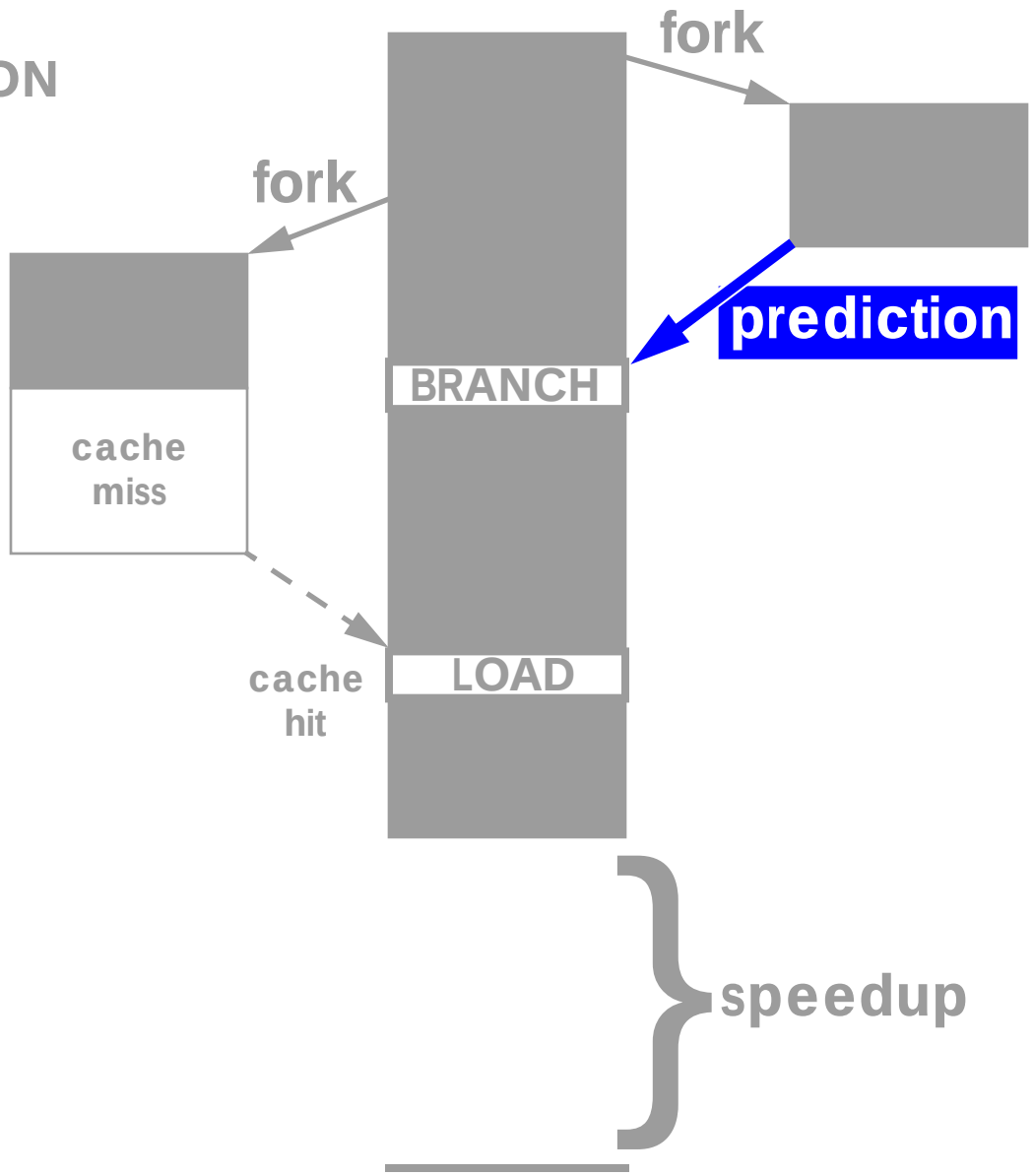
The Outline

- PROBLEM INSTRUCTIONS
- EXECUTION-BASED PREDICTION



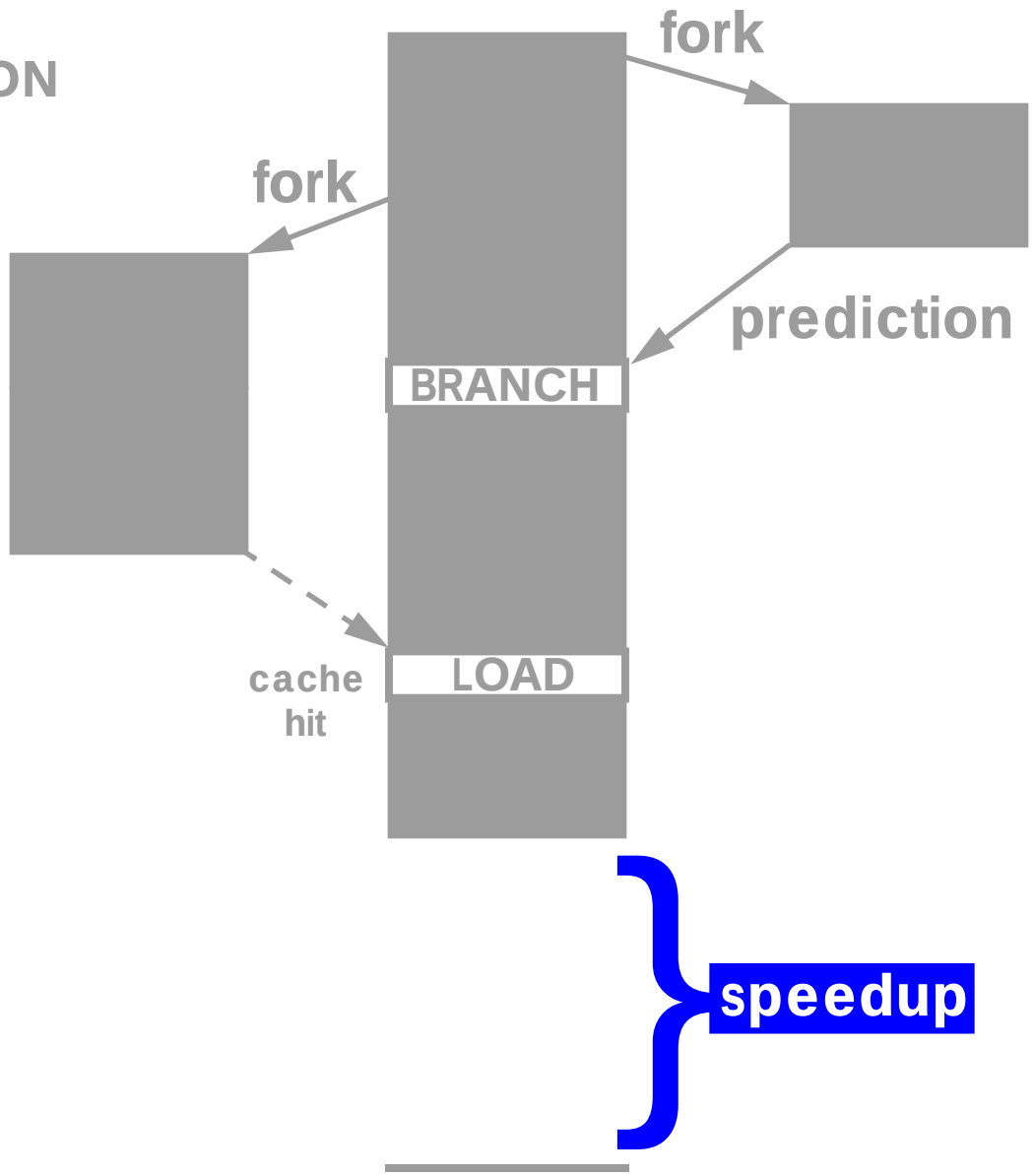
The Outline

- Problem Instructions
- Execution-based Prediction
- **Prediction Correlation**



The Outline

- Problem Instructions
- Execution-based Prediction
- Prediction Correlation
- **Performance Results**



Problem Instructions

Misses and mispredictions are not evenly distributed.

EXAMPLE: PERLBMK

- 82 static branches: 68% of misp., 9% of dynamic branches
- 140 static loads: 67% misses, 2% of dynamic memory insts

Fixing just problem inst's gives > 1/2 perf. of perfect cache/pred

OUTCOMES OF THESE INSTRUCTIONS DO NOT EXHIBIT A PREDICTABLE PATTERN...

- consistently mispredicted

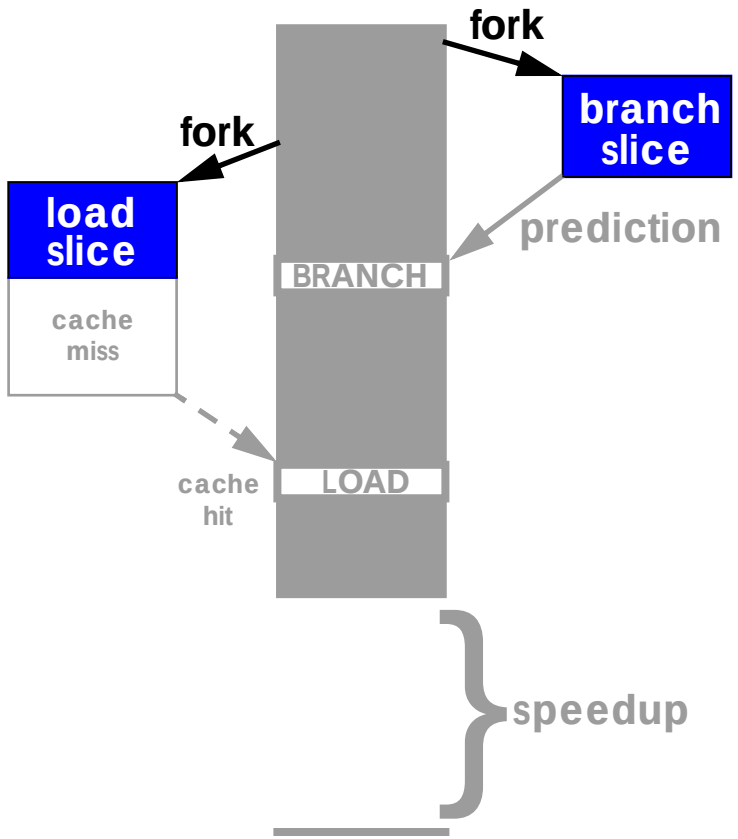
... BUT SOMETIMES THE COMPUTATION IS REGULAR.

```
while (...) {
    ...
    ptr = ptr->next;
}
```

```
while (i < n) {
    if (object[i] != NULL) {
        ...
    }
```

Outline

- PROBLEM INSTRUCTIONS
- EXECUTION-BASED PREDICTION
  - An different pre-execution approach
  - Speculative slices and imprecise transformations
  - Slice structure
  - Slice characterization
- PREDICTION CORRELATION
- PERFORMANCE RESULTS



Previous pre-execution proposal

Speculative Data-driven Multithreading: Roth and Sohi, HPCA'01

- Speculatively pre-executes data-driven threads (DDTs)
- Register integration matches DDTs to main thread
  - + avoids re-execution of DDT instructions
  - + early branch resolution (at decode stage)
  - - DDTs must be sub-set of original program

Two Observations

Two Observations:

- **benefit comes from prefetches and predictions**
- **strict program subsets not most efficient slices**

Our approach: generate predictions/prefetches in as efficient manner as possible.

OPTIMIZE SLICES:

+ reduce fetch/execution overhead

+ reduce critical path to making prediction

- need a new mechanism to correlate predictions

Speculative Slices

DON'T ALLOW SLICES TO AFFECT ARCHITECTED STATE

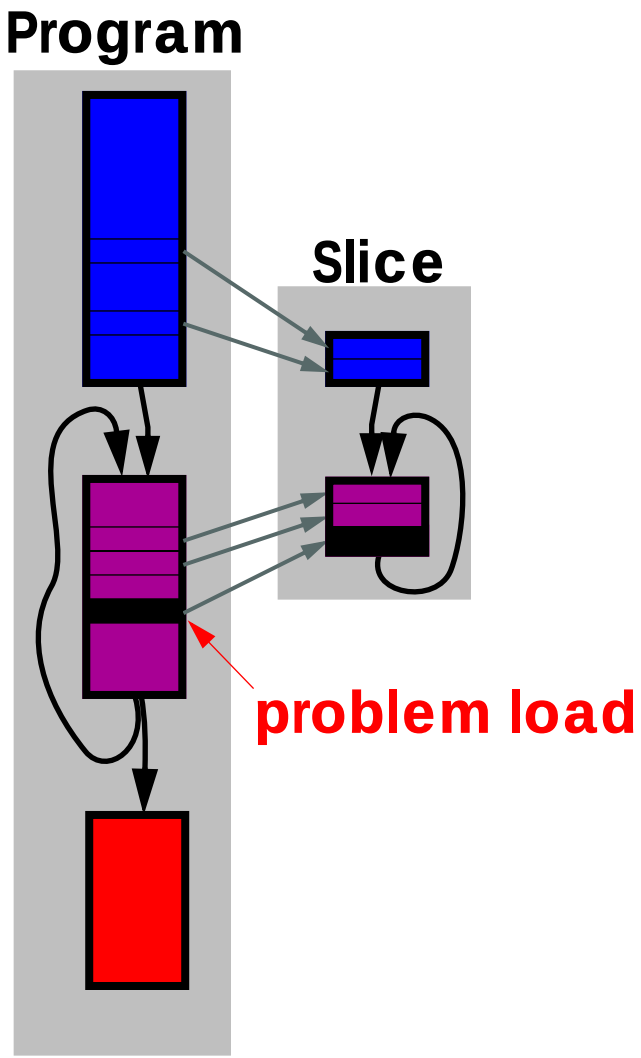
- only generate pre-fetches and predictions
- need not be 100% accurate

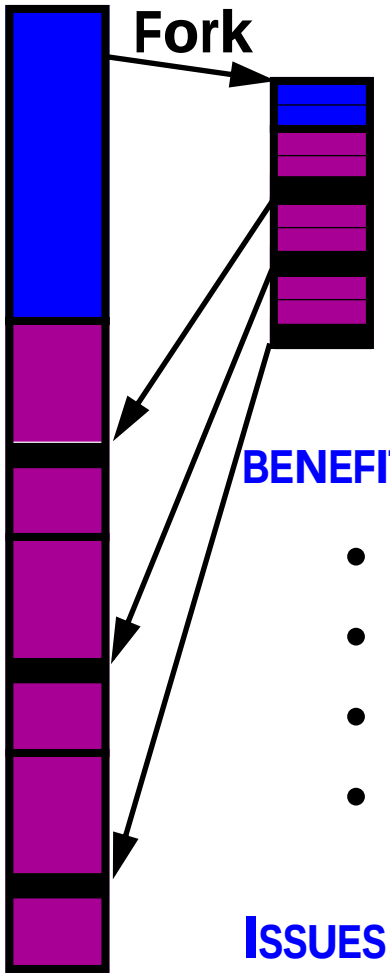
3 CLASSES OF TRANSFORMATIONS: (NOT ORIGINALLY APPLIED BY COMPILER)

- **Imprecise**
  - static branch assertion (remove branches/cold code)
- **Not-provably safe**
  - register allocation in the presence of aliases
- **Previously unprofitable**
  - if-conversion (of a subset of a block)

Slice Structure

- problem instructions frequently in loops
- encapsulate loop in slice





BENEFITS:

- lower overhead
- earlier predictions
- amortize fork overhead
- single helper thread

ISSUES & SOLUTIONS: in paper

Slice Characterization

CONSTRUCTED AND OPTIMIZED SLICES BY HAND

- encouraging results

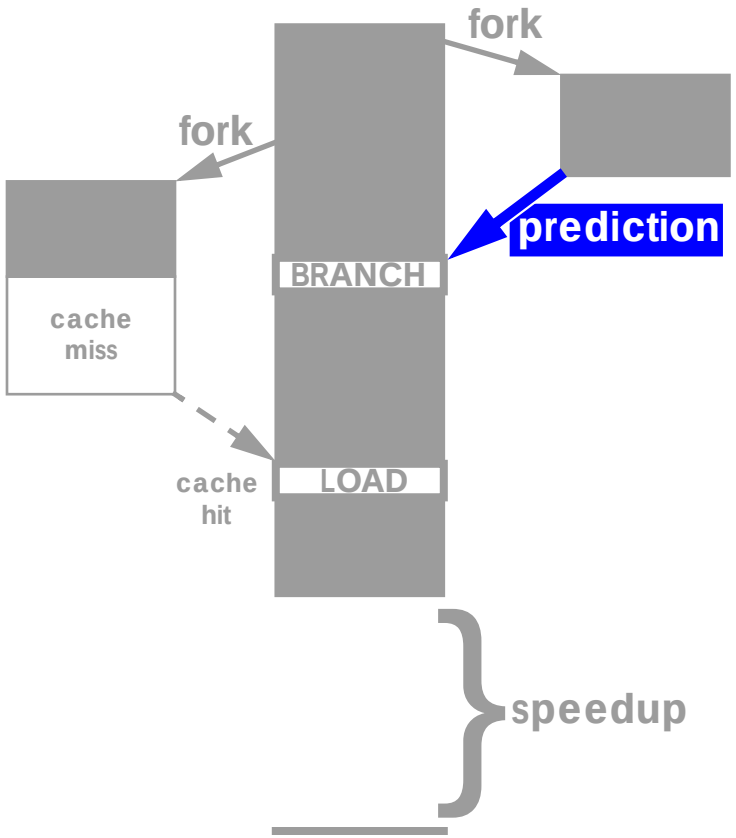
STATISTICS:

- 85% of slices cover multiple static problem instructions
- 70% of slices contained loops
- small static size
  - smaller than 4 * # problem instructions covered
- prefetch or prediction generated every ~3 dynamic inst's.
- small number of live-in values
  - 80% of slices had 2 or less

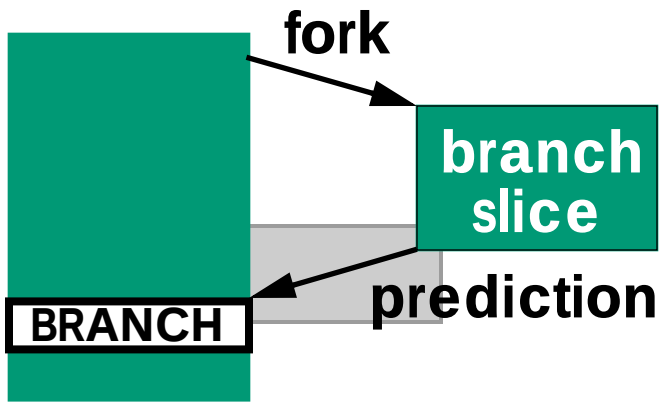
slices can be very small

Outline

- PROBLEM INSTRUCTIONS
- EXECUTION-BASED PREDICTION
- PREDICTION CORRELATION
  - difficult problem
  - valid regions
- RESULTS AND ANALYSIS



Prediction Correlation

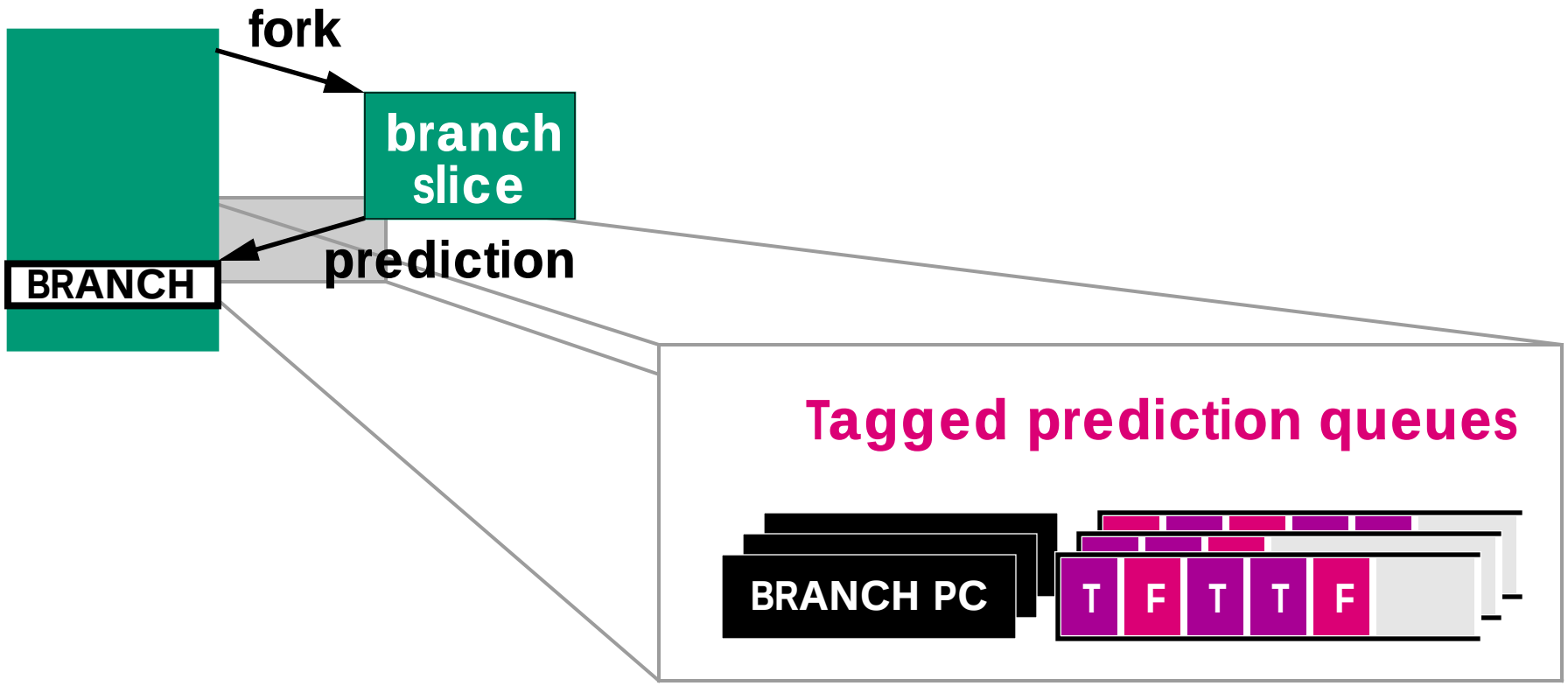


TO BENEFIT FROM A SLICE-GENERATED PREDICTION

- must bind it to fetched branch instruction
- overrides hardware branch predictor

HOW ARE PREDICTIONS CORRELATED TO DYNAMIC BRANCHES?

Prediction Correlation



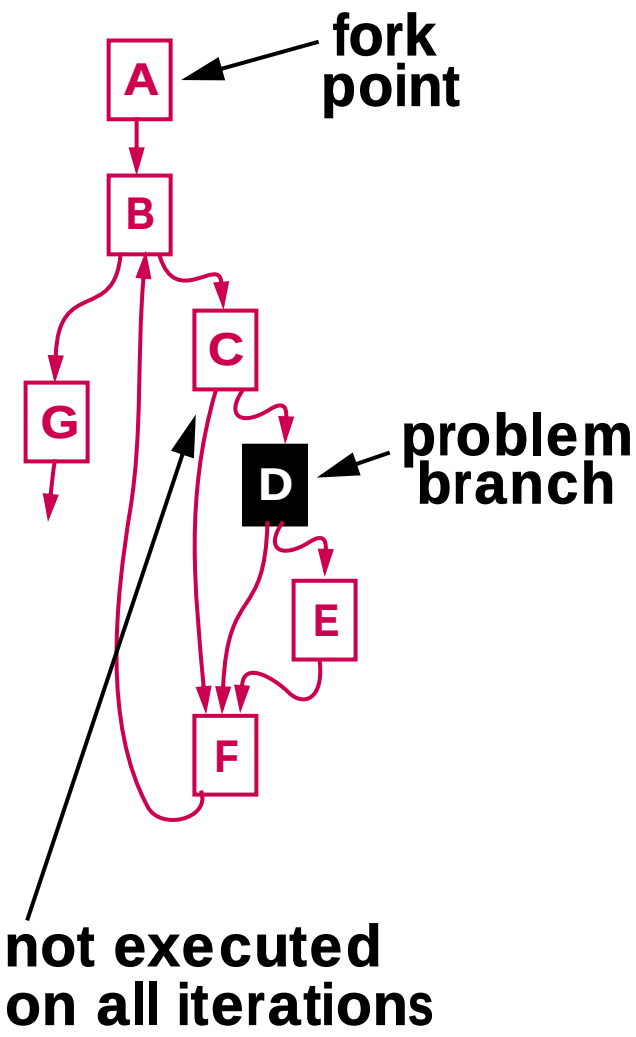
Related Work: Farcy, et al, Micro '98

CHALLENGES:

- re-ordering predictions produced out-of-order
- recovering from misspeculation by main thread
- **dealing with conditionally-executed problem branches**

Conditionally-executed problem branches

program's CFG



MINIMIZE OVERHEAD BY BUILDING SIMPLEST SLICE

- compute prediction for each iteration

NAIVE IMPLEMENTATION

- predictions dequeued when used
- mis-alignment occurs on path CF

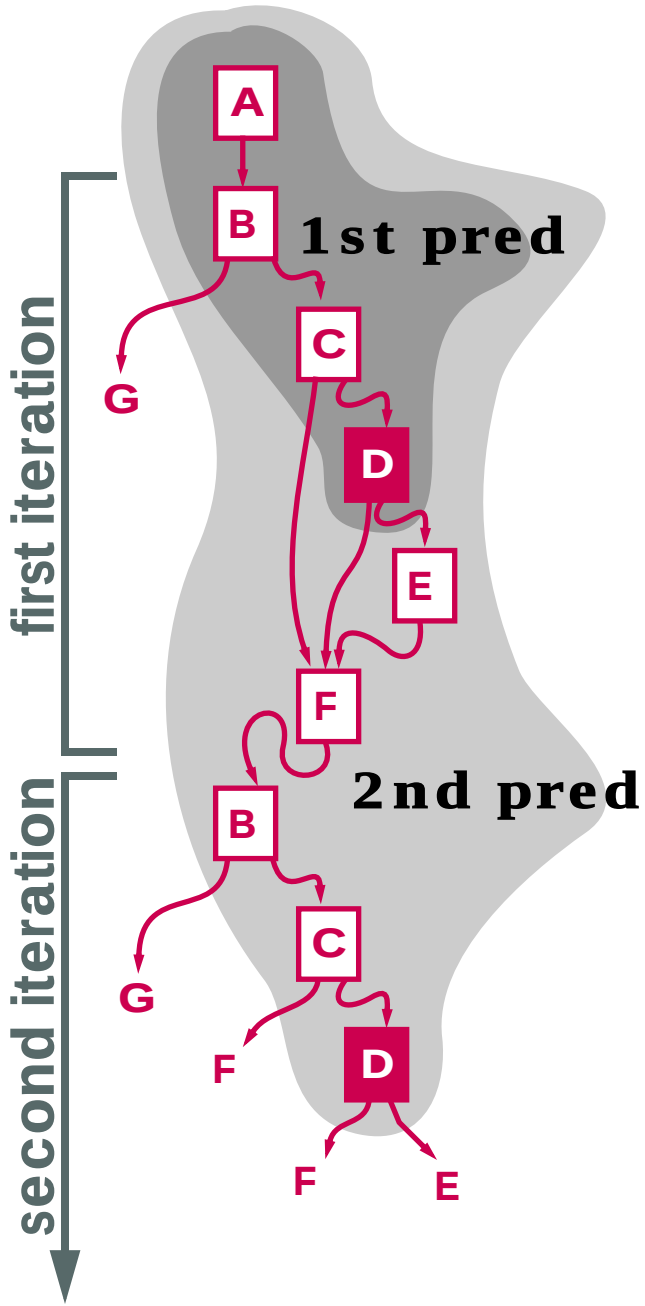
CONDITIONALLY GENERATE PREDICTIONS?

- include "existence slice" in slice
- too much overhead

INSIGHT

- existence slice encoded in fetch path

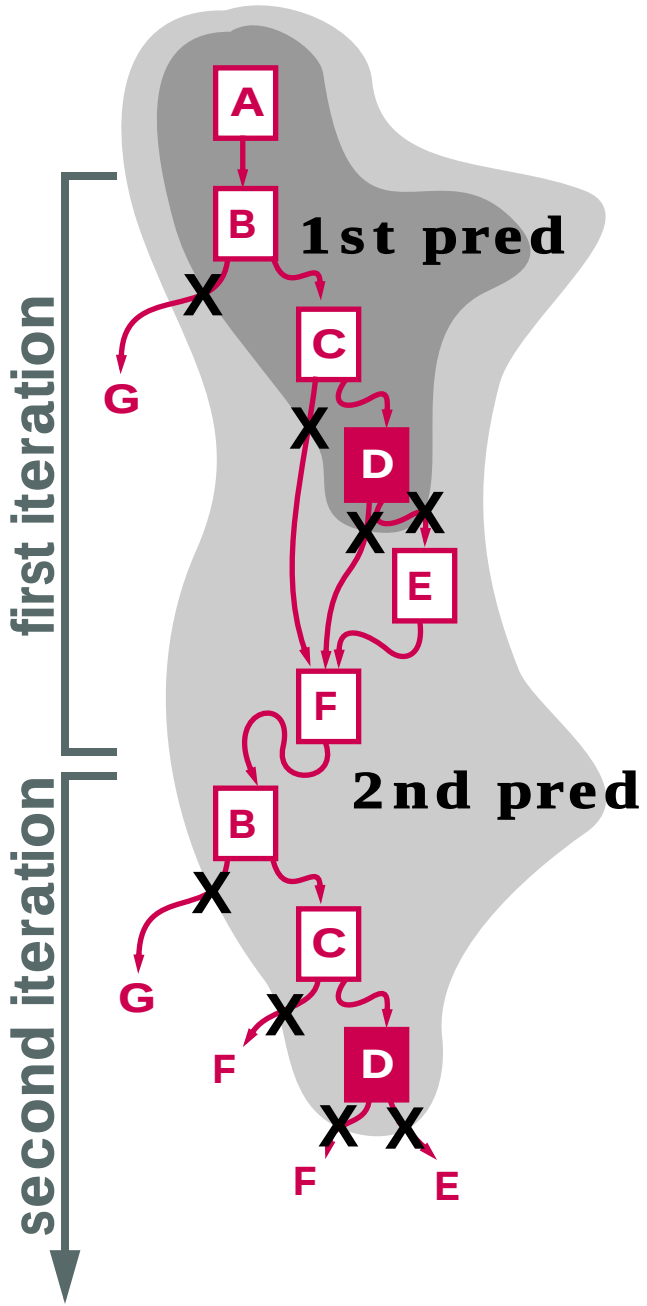
Valid Regions



DEFINE REGION WHERE PREDICTION IS VALID

- using assumptions from building slice

Valid Regions



DEFINE REGION WHERE PREDICTION IS VALID

- using assumptions from building slice
- "markers" to indicate region boundary
- implementation discussed in paper

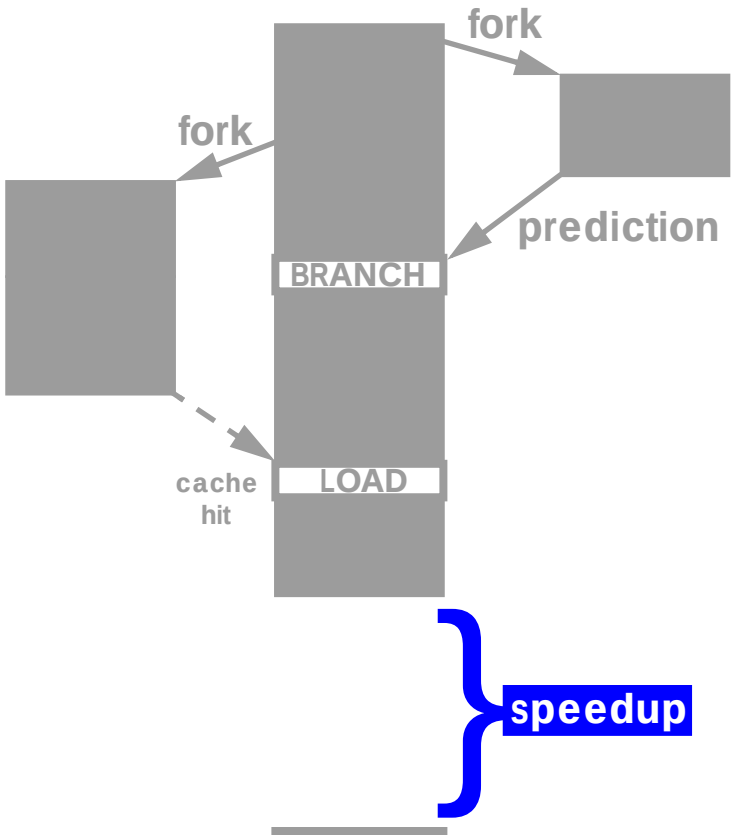
DEQUEUE PREDICTION WHEN MARKER ENCOUNTERED

- using a prediction doesn't dequeue it

greater than 99% correlation accuracy

Outline

- Problem Instructions
- Execution-based Prediction
- Prediction Correlation
- **Results and Analysis**
  - **Methodology**
  - **Results**
  - **Discussion**



Methodology

Used SPEC2000 integer benchmarks

- spectrum of program behaviors

Identified dominant program phase

- selected 100M inst. region for simulation

Built slices (by hand) to cover problem instructions

Warmed up simulator for 100M instructions

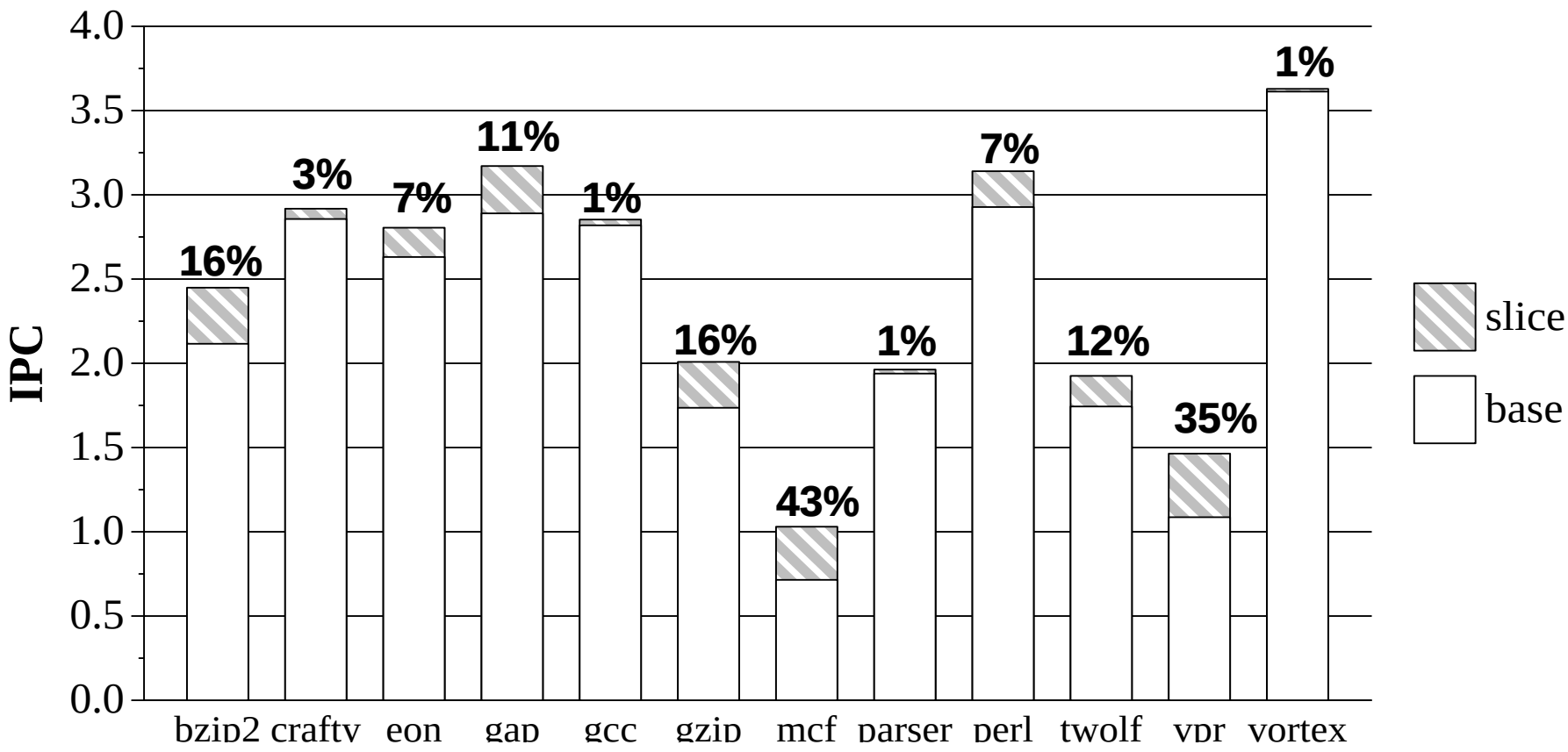
Methodology, cont.

AGGRESSIVE BASELINE:

- 4-wide superscalar, 128 entry window, 14 cycle mispredict penalty
- 2 load/store units, 4 fully pipelined integer/floating point units
- 64Kb YAGS branch, 32Kb cascaded indirect, RAS predictors
- Fetches across basic blocks, perfect BTB for direct branches
- 2-way associative 64KB L1 caches (64B blocks)
- 4-way associative 2MB unified L2 cache (128B blocks)
- 64-entry unified pre-fetch/victim buffer with hardware stream pre-fetcher

Deeply-pipelined, 4-wide, out-of-order superscalar with big predictors, associative caches, hardware stride pre-fetcher, and victim buffers.

Results



speedups ranging from 1% to 43%

- **must be regularity in branch/address computation**
- **speedups proportional to memory, branch stall time**
- **low base IPC → lower opportunity cost of slice execution**

Related Work

Pre-execution:

- Roth and Sohi: HPCA-2001 and TR-2000

SPECULATIVE SLICES:

- Zilles and Sohi: ISCA-2000

Limited forms of pre-execution:

- Roth, et al: ASPLOS-1998 and ICS-1999
- Farcy, et al: Micro-1998

Slipstream processors:

- Sundaramoorthy, et al: ASPLOS-2000

Helper threads:

- Chappell, et al: ISCA-1999
- Song and Dubois: TR-1998

Summary

PROBLEM INSTRUCTIONS

- behavior not predictable with existing predictors
- sometimes computation is regular

EXECUTION-BASED PREDICTION

- execute code fragments to generate prediction/prefetch
- imprecise transformations enable small slices

PREDICTION CORRELATION: VALID REGIONS

- monitor main thread's fetch path
- greater than 99% correlation accuracy

Speedups of 1 to 43% over an aggressive baseline