Control Independence in Trace Processors

Eric Rotenberg

Jim Smith

North Carolina State University

Department of Electrical and Computer Engineering

ericro@ece.ncsu.edu

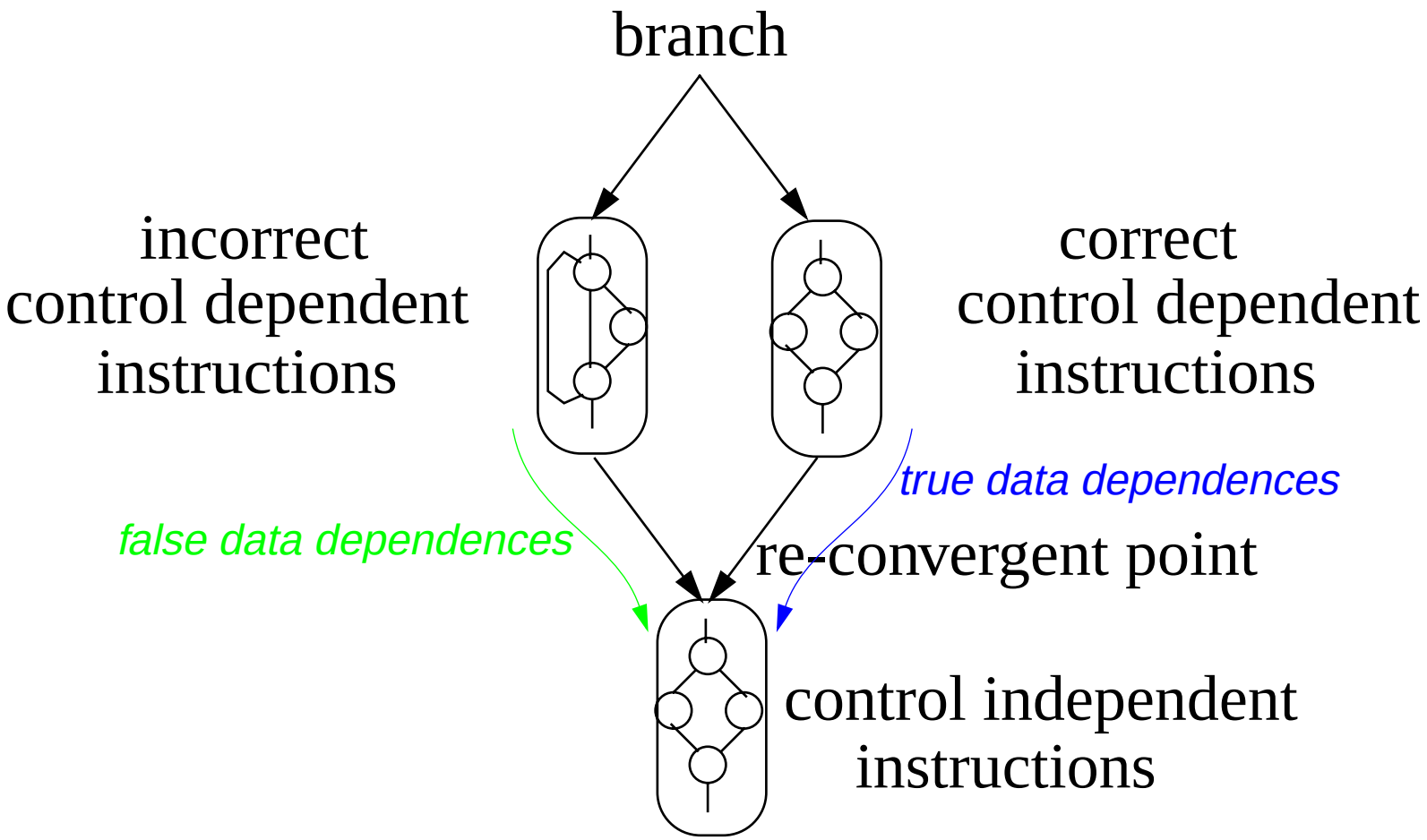
Introduction

- High instruction-level parallelism (ILP)
  - requires large instruction scheduling window
  - branches make this difficult
- Branches introduce control dependences
  - must execute branch to fetch next instructions
  - correct branch prediction/speculation: essentially eliminates control dependences
  - mispredictions are still a major bottleneck: complete squashing

Introduction

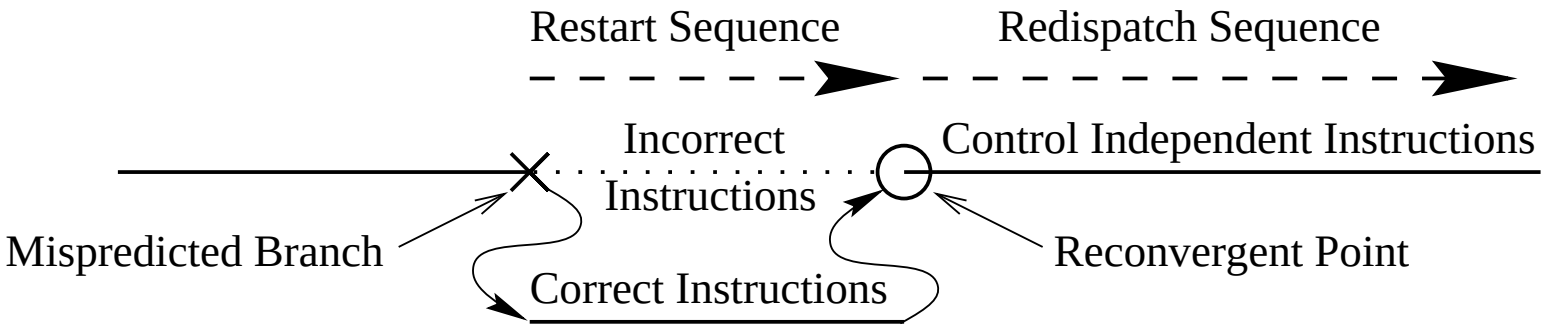
- Solutions to branch misprediction bottleneck
  1. better branch prediction
  2. forms of multi-path execution
  3. predication
  4. control independence

Control Independence



Basic Requirements

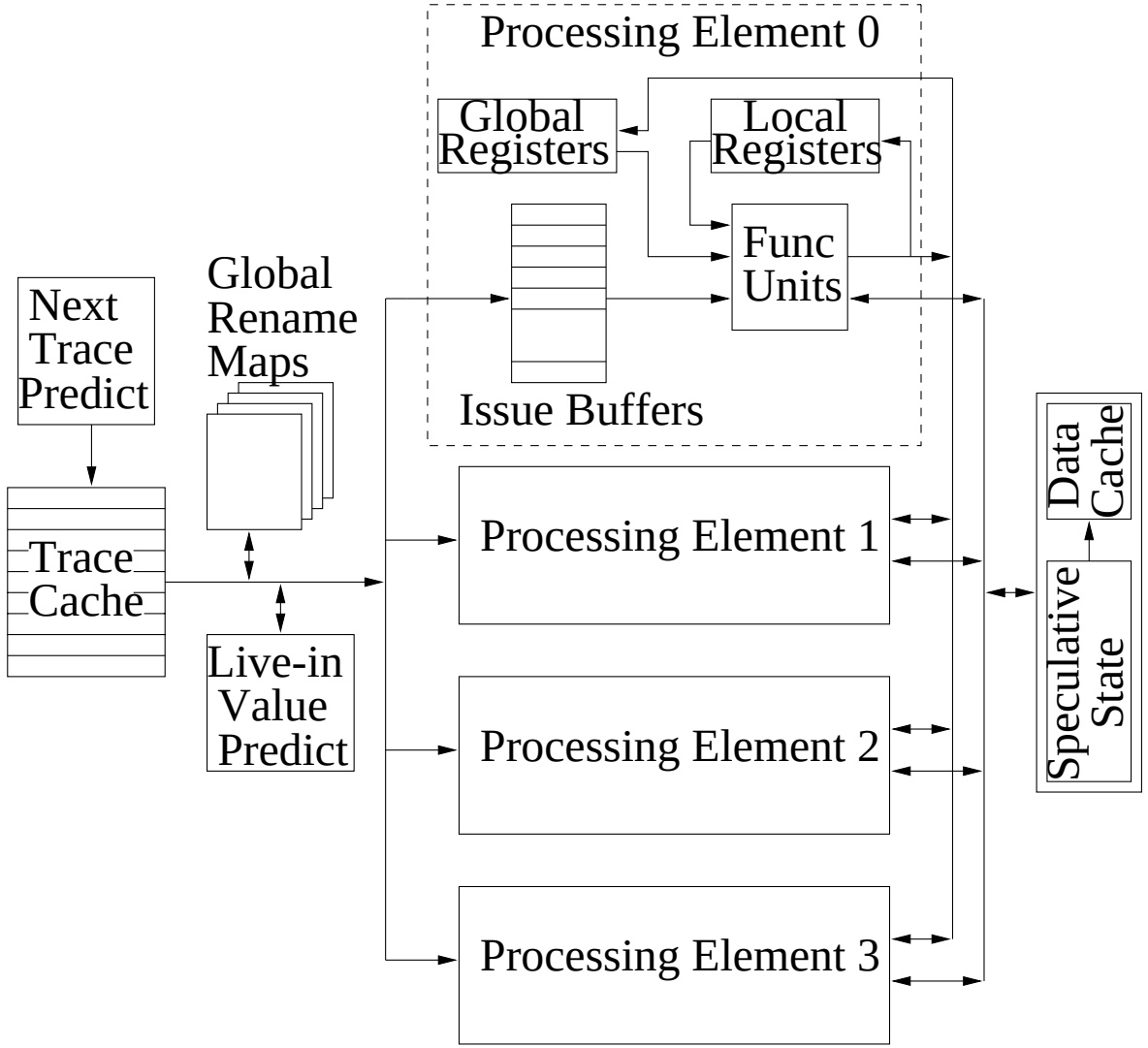
- Basic requirements for misprediction recovery
  - Detect the re-convergent point
  - Insert/remove instructions from middle of window
  - Form correct data dependences
  - Selectively reissue instructions



Complexity

- Major source of complexity: sophisticated window management
  - Arbitrary insertion/removal of instructions from middle of window
  - Conventional instruction window (“reorder buffer”) is managed as a fifo
  - Arbitrary shifting, or linked-list implementation, difficult
- Conveying control dependence information
  - maybe don’t want software to convey and expose re-convergent points
- Repairing data dependences

Trace Processor



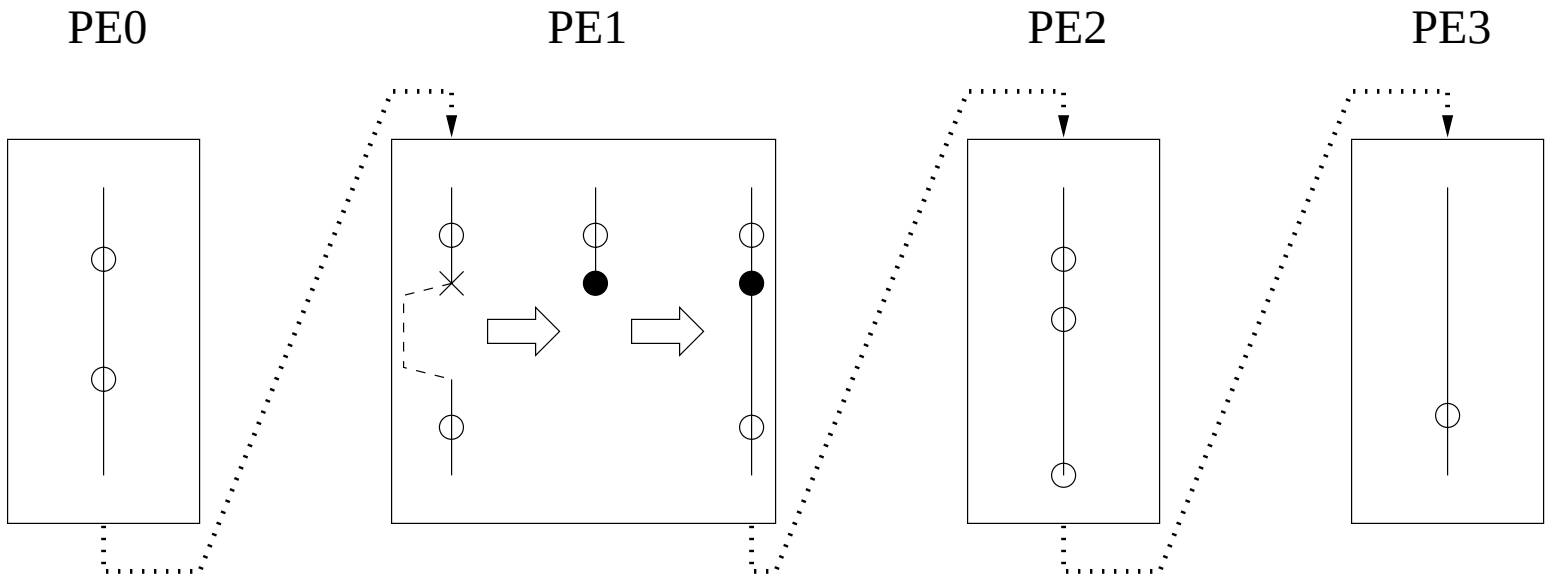
Trace Processor Control Independence

- Three key ideas
  - Treating traces as fundamental unit of control flow enables sophisticated window management (hierarchy)
  - Trace selection ensures and identifies trace-level re-convergence
  - Existing support for data speculation is easily leveraged for selectively re-issuing incorrect-data dependent instr.

Exploit Hierarchy

1. Hierarchical management of control flow
   - isolate intra-trace flow from inter-trace flow
   - branch and control independent point in same trace

   => fine-grain control independence (FGCI)

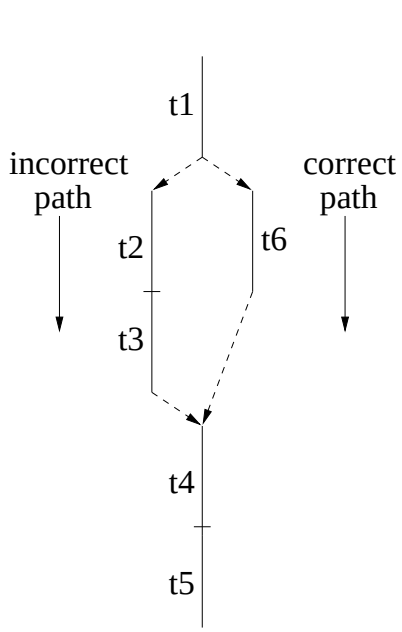


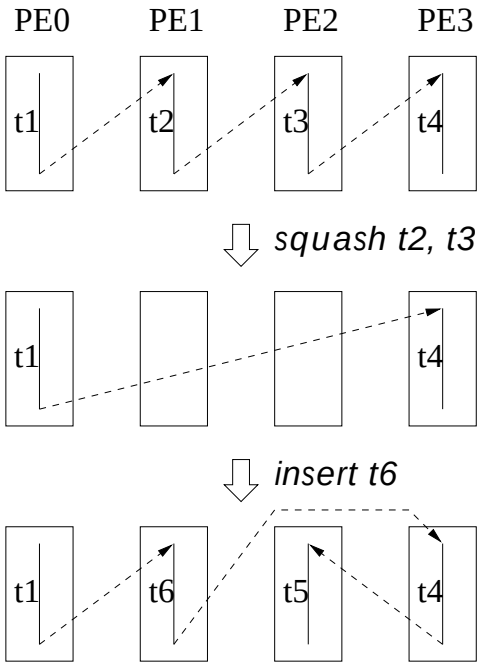
Exploit Hierarchy

2. Hierarchical management of resources

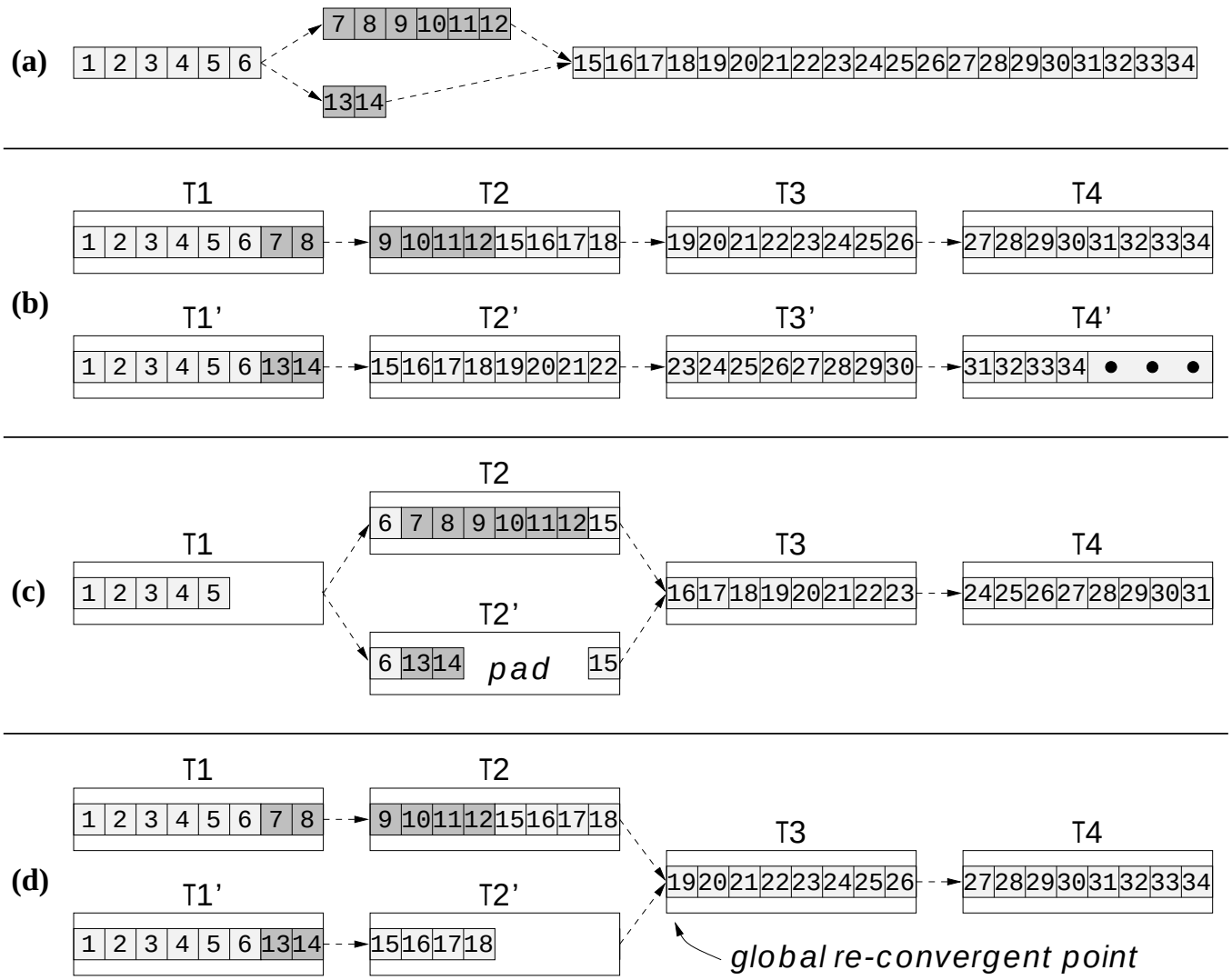
- large unit of allocation (PE/trace): easy to manage PE allocation/de-allocation flexibly
- branch and control independent point in different traces

=> coarse-grain control independence (CGCI)



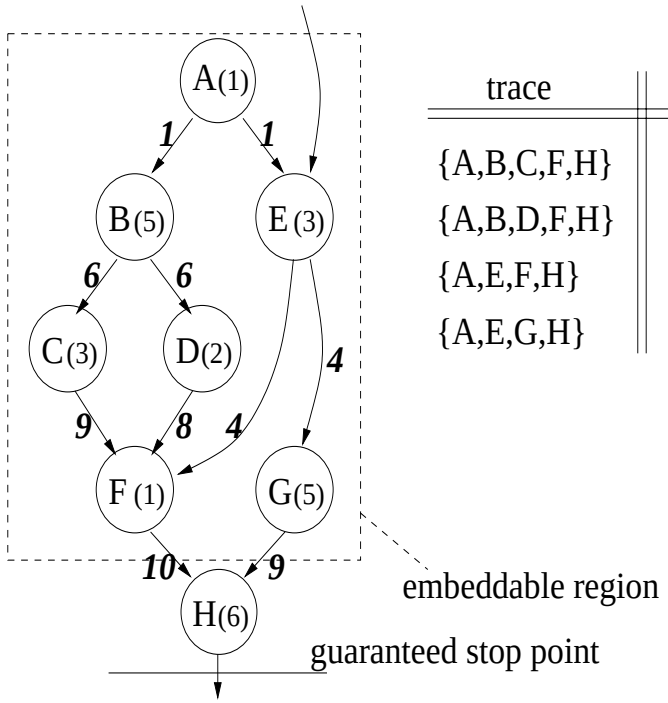


Trace-level Re-convergence



FGCI Trace Selection

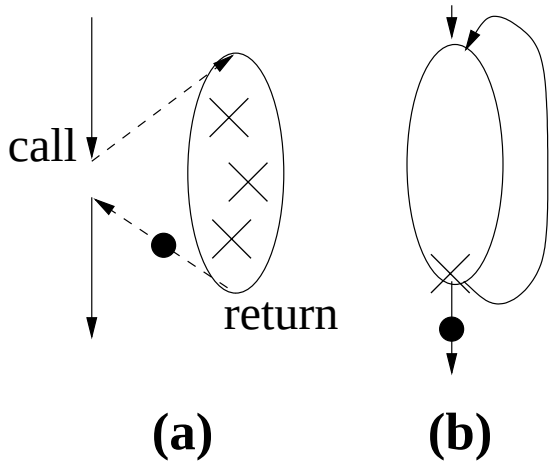
- FGCI-algorithm analyzes arbitrarily complex forward-branching regions
  - locates re-convergent point
  - computes longest control dependent path
- FGCI trace selection uses info to “pad” shorter paths



trace	length
{A,B,C,F,H}	16
{A,B,D,F,H}	15
{A,E,F,H}	11
{A,E,G,H}	15

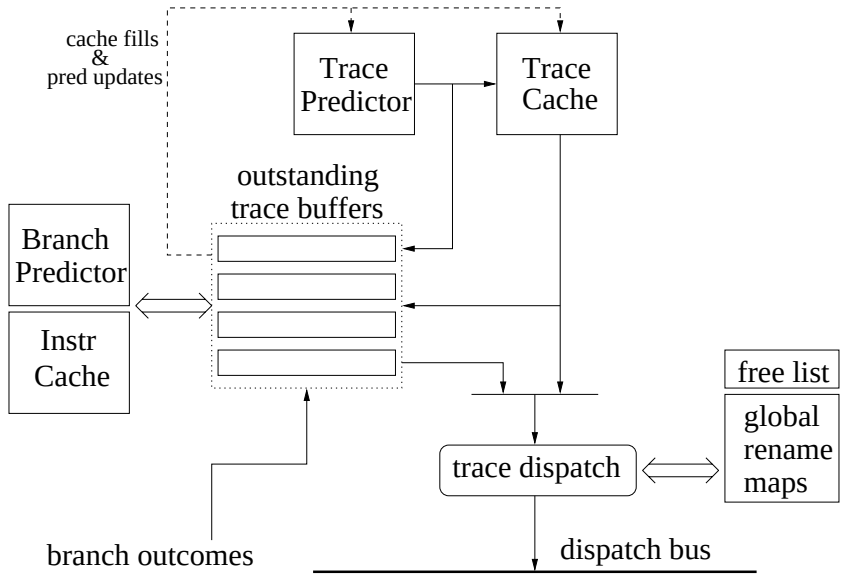
CGCI Trace Selection

- Identify “global” control independent points
  - Imprecise: points don’t correspond directly with any particular branch
  - But, broad coverage of branch mispredictions
- CGCI trace selection terminates traces at return-points and not-taken backward branches (black dots below)



Repairing Data Dependences

- Register dependences
  - Re-dispatch control independent traces



- Memory dependences

  Misspeculated loads due to control independence

  ==

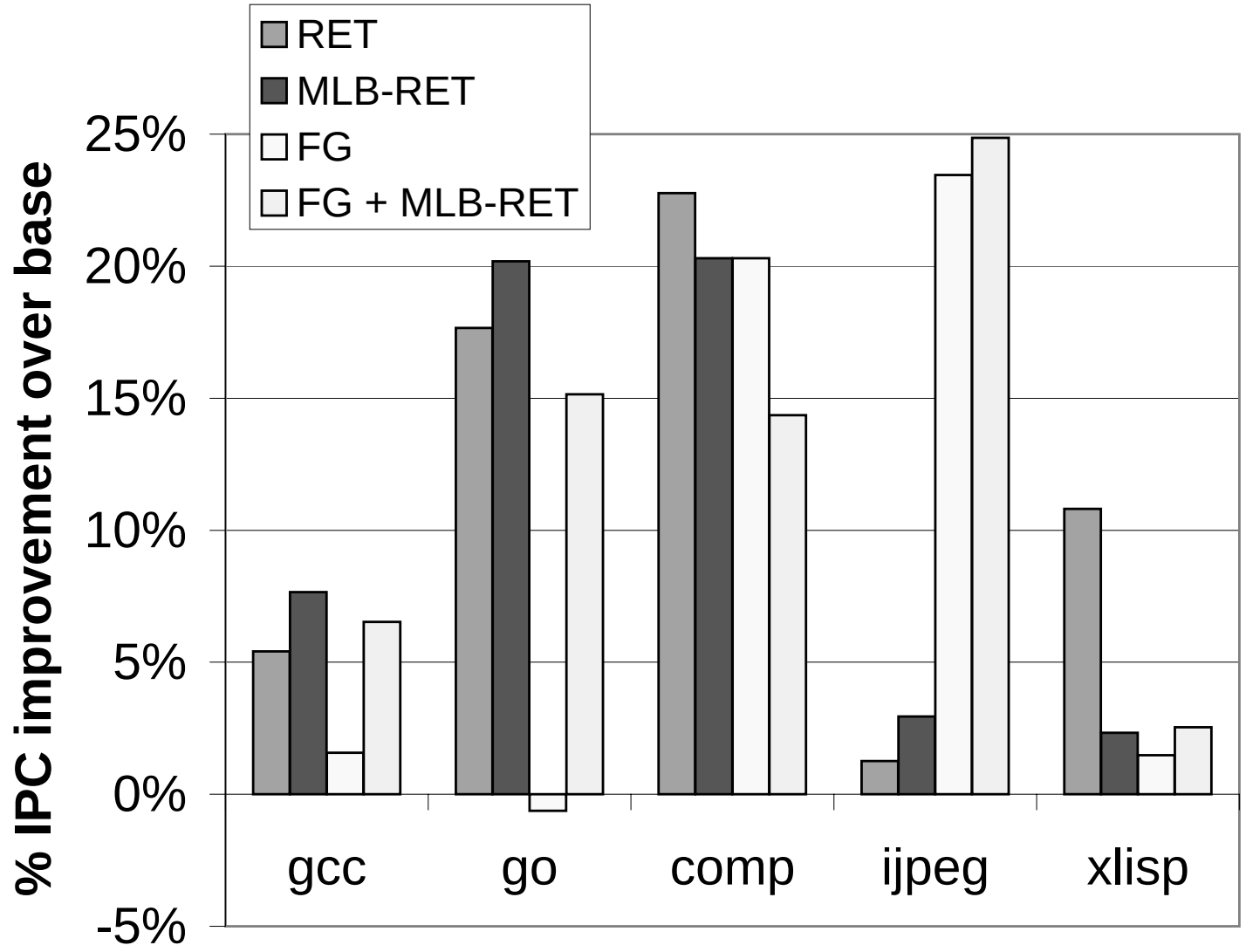
  Misspeculated loads due to speculative disambiguation

- Selective re-issuing
  - Already supported for data speculation

Graph Labels

- **RET**
  - CGCI only
  - Use the return-point heuristic for all mispredictions
- **MLB-RET**
  - CGCI only
  - Try MLB first for mispredicted loop branches
  - RET covers all mispredicted branches
- **FG**
  - FGCI only
- **FG + MLB-RET**

Performance



Conclusion

- Trace processor is a good microarchitecture for exploiting control independence.
- Concepts
  1. Hierarchy is a key enabler for exploiting control independence (sophisticated window management)
  2. FGCI and CGCI trace selection ensure and identify trace-level re-convergence
  3. Existing support for data speculation is easily leveraged
- Performance gains of up to 25%