Slipstream Processors:

Improving both Performance and Fault Tolerance

Eric Rotenberg

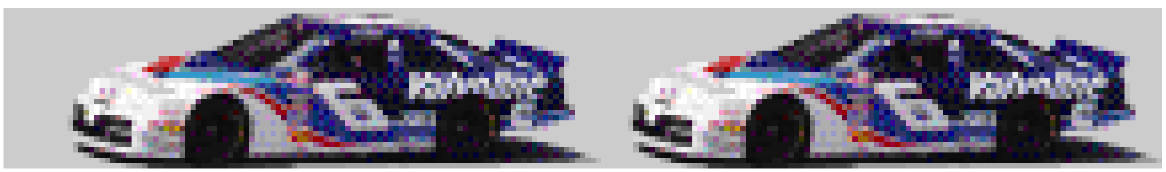
Karthik Sundaramoorthy, Zach Purser

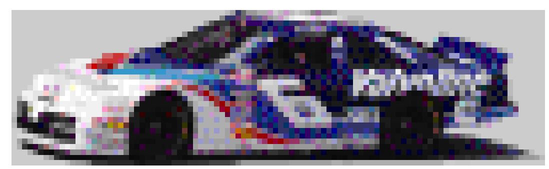
Dept. of Electrical and Computer Engineering
North Carolina State University
www.tinker.ncsu.edu/ericro/slipstream
{ksundar,zrpurser,ericro}@ece.ncsu.edu

NASCAR and Computers

- "Slipstreaming"
  - Two cars race nose-to-tail to speed up *both* cars

195 m.p.h.

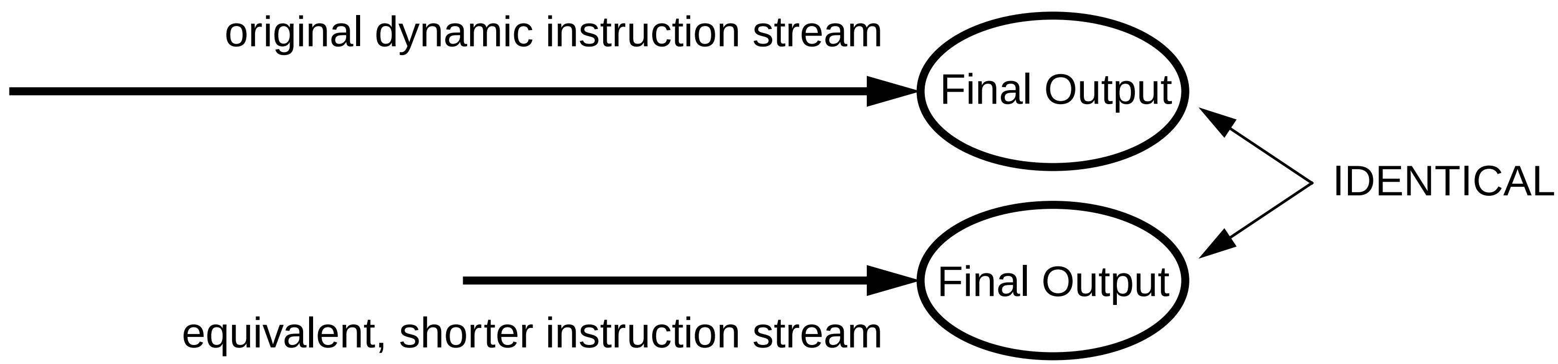




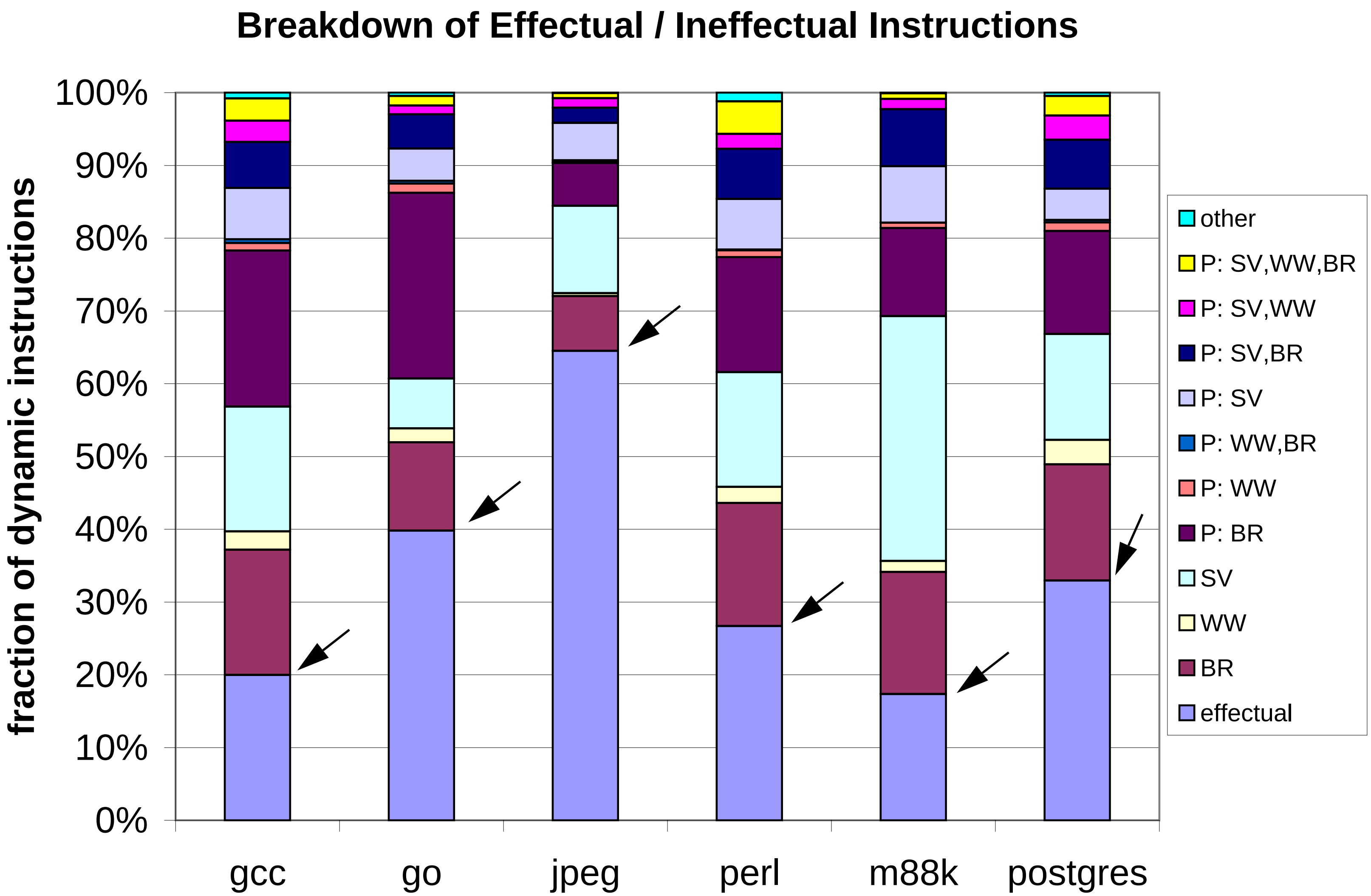
190 m.p.h.

“Ineffectual” Instruction Sequences

- Can ideally construct shorter equivalent program
  - Non-modifying writes
  - Unreferenced writes
  - Correctly-predicted branches
  - ...also their dependence chains



Shorten program using “oracle”



Catch-22

- Only need a small part of program to make full, correct, forward progress
- Skipping instructions is speculative
  - Lose ability to verify since instructions are removed
  - Must compare against full program
- Answer: run both programs! (redundant execution)

Slipstream Paradigm

- Operating system creates two redundant processes
  - Programs run concurrently on single-chip multiprocessor (CMP) or simultaneous multithreading processor (SMT)
  - One program always runs slightly ahead of other
    - *Advanced stream* (A-stream) leads
    - *Redundant stream* (R-stream) trails

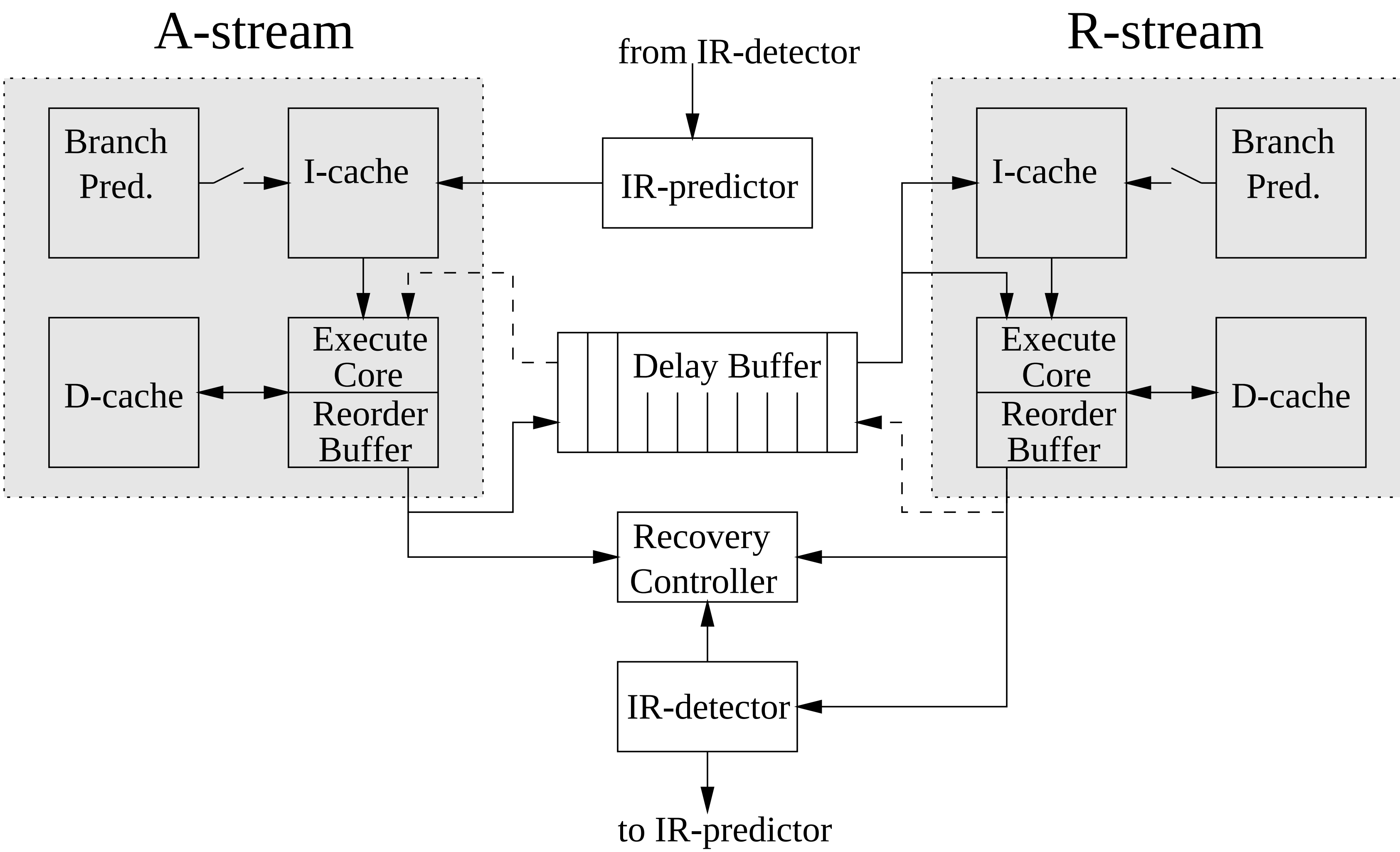
Slipstream Paradigm

- A-stream
  - Monitor R-stream to detect past removable computation
  - Use knowledge to speculatively reduce A-stream in future
  - A-stream fetches/executes fewer instructions
- R-stream
  - A-stream passes control/data outcomes to R-stream
  - R-stream checks outcomes: if A-stream deviates, its context is recovered from R-stream
  - R-stream also uses A-stream outcomes as *predictions*
    - Leverage existing speculation mechanisms to do checks
    - R-stream fetches/executes more efficiently

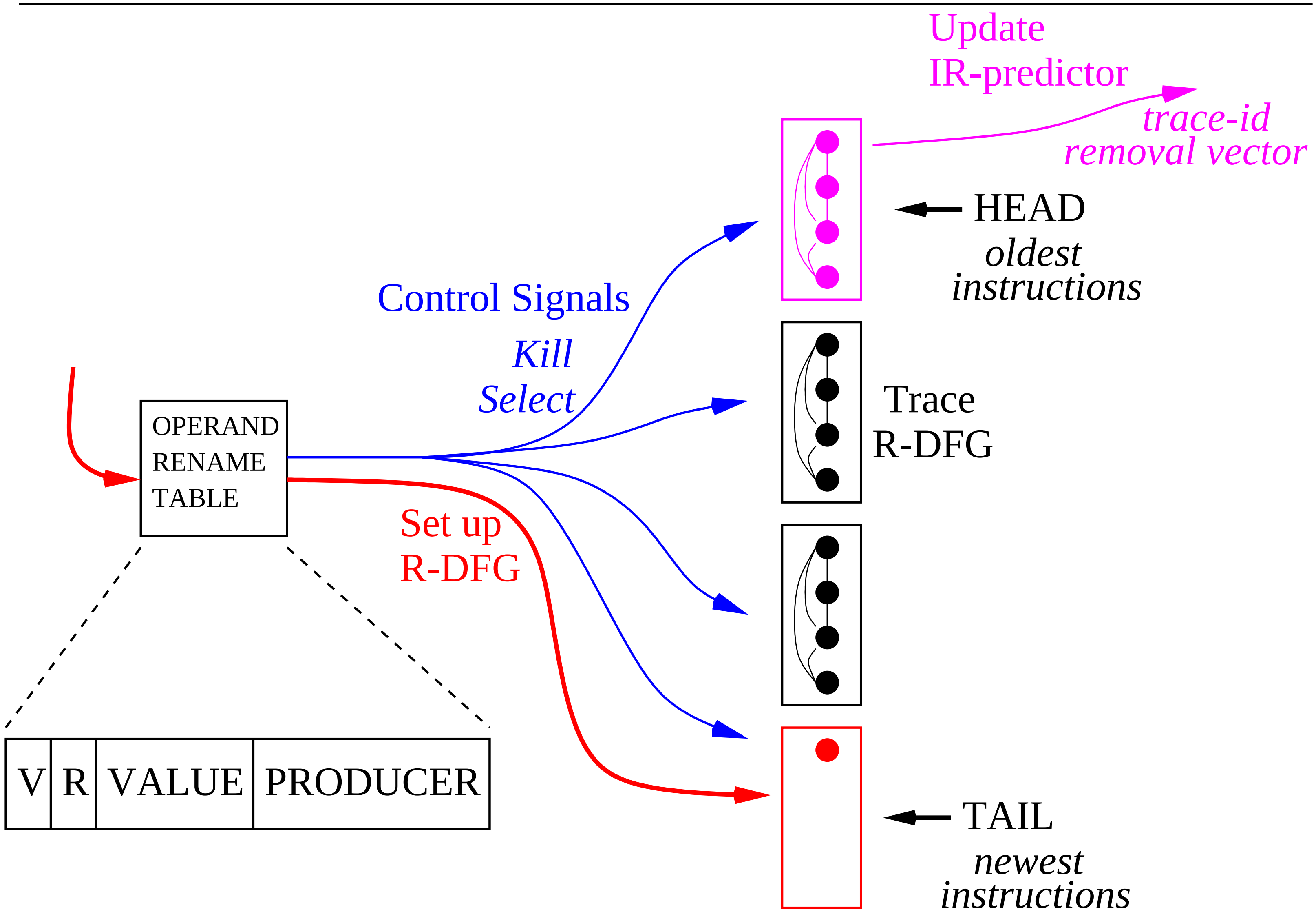
Slipstream Paradigm

1. Improved single-program performance
   - Both programs finish sooner (and approx. same time)
     - A-stream: reduced in length
     - R-stream: un-reduced but much more efficient
   - Faster than non-redundant execution
2. Improved fault tolerance
   - Partial redundancy => partial transient fault coverage
   - *Transparent* fault tolerance
     - Faults indistinguishable from removal mispredictions
     - Cannot explicitly detect faults
     - Can tolerate faults via existing recovery mechanism

Example Microarchitecture

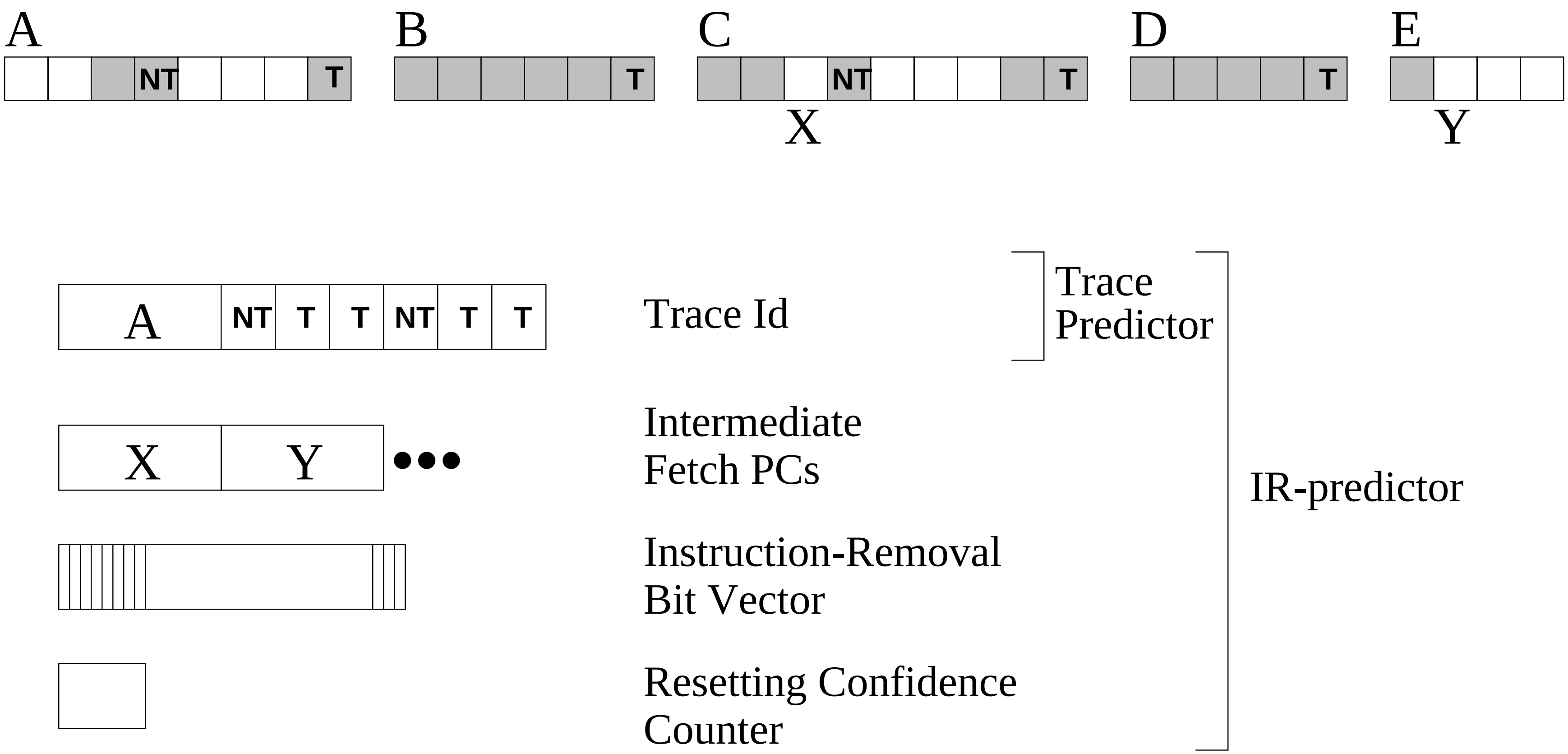


IR-detector



Update
IR-predictor
trace-id
removal vector
HEAD
oldest
instructions
Control Signals
Kill
Select
OPERAND
RENAME
TABLE
Trace
R-DFG
Set up
R-DFG
V
R
VALUE
PRODUCER
TAIL
newest
instructions

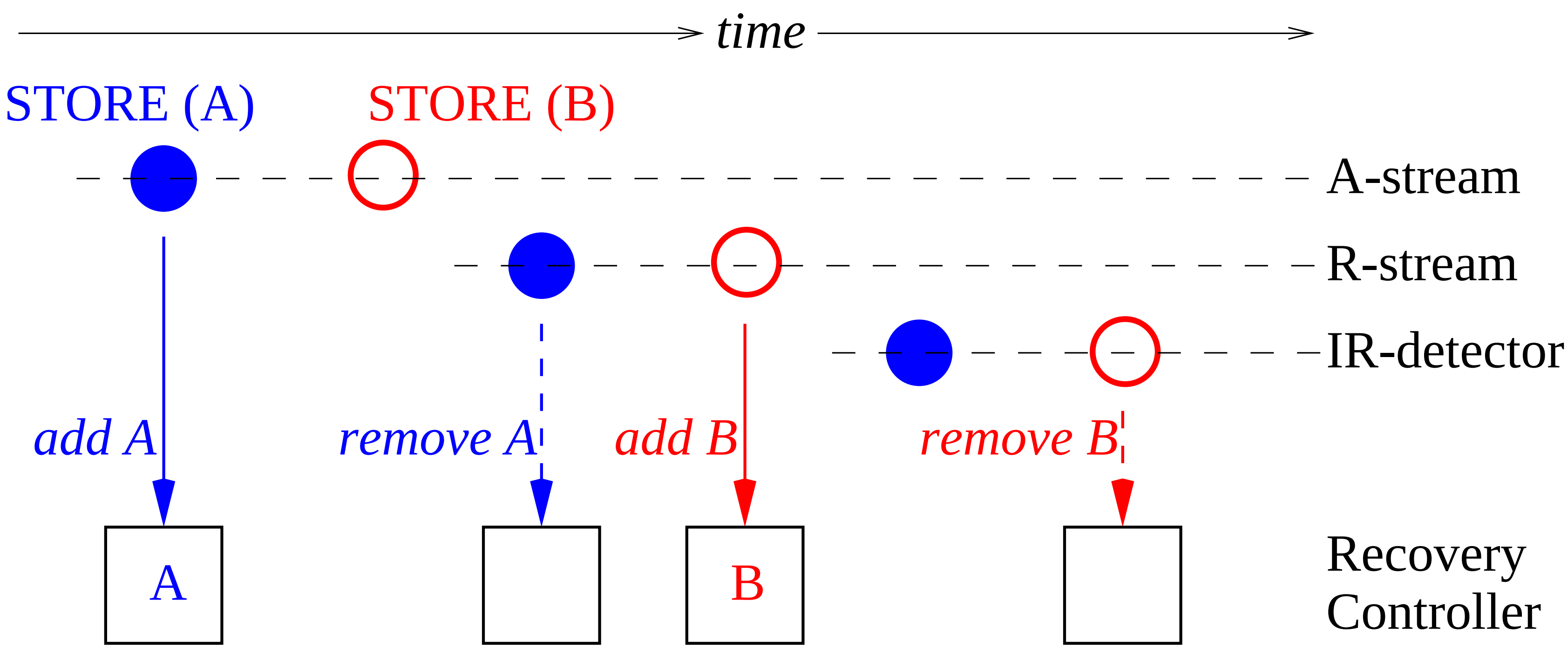
IR-predictor



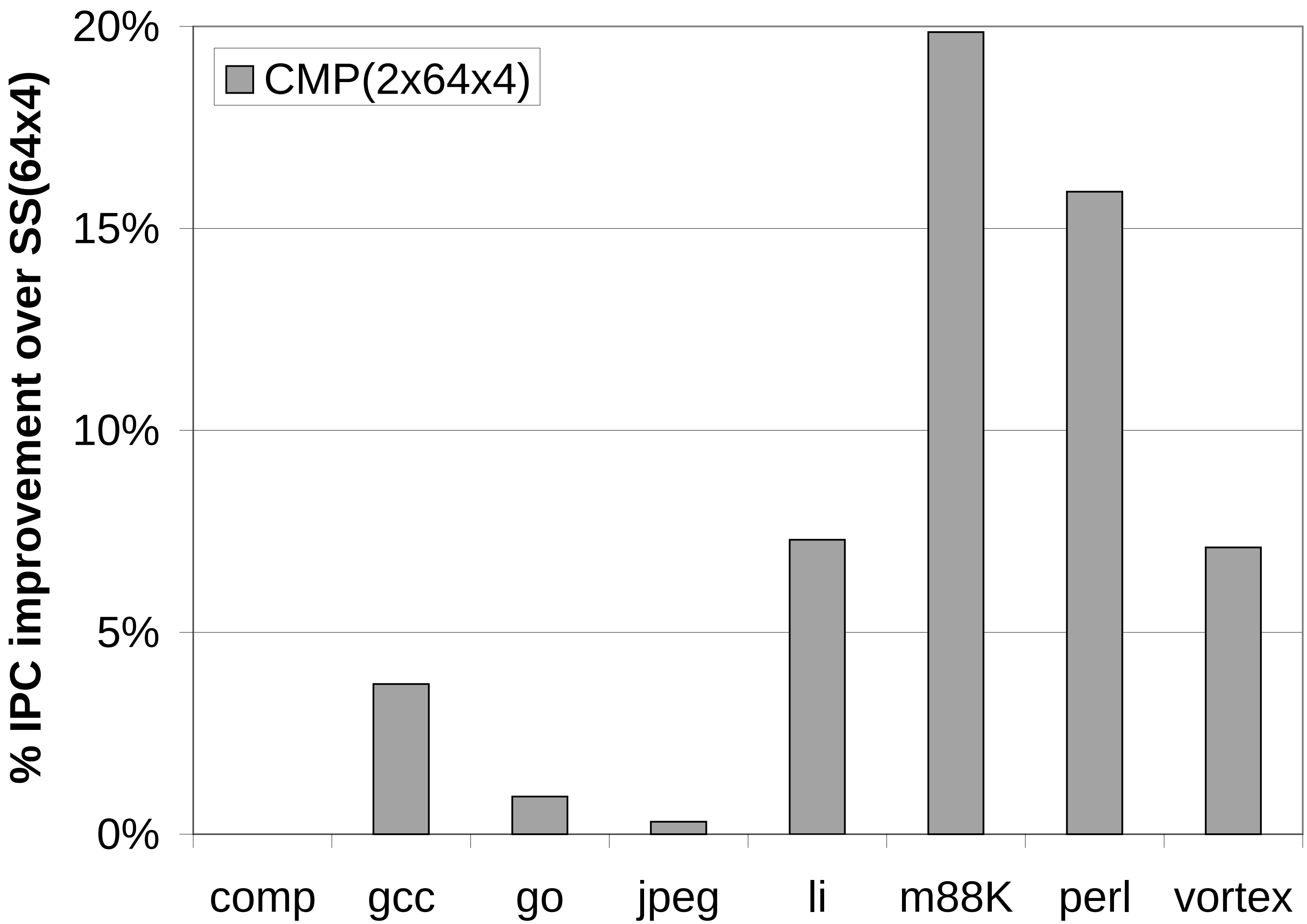
IR-mispredictions + Recovery

- Instruction-removal misprediction (IR-misprediction)
  - Instructions were removed that shouldn't have been
  - Detected as mispredictions in R-stream
- Recovery: Re-synchronize A-stream w.r.t. R-stream
  - Flush reorder buffers, delay buffer, backup IR-predictor
  - $PC_A = PC_R$
  - Copy entire R-stream register file
  - Copy only key memory locations that differ
- Recovery controller pin-points key memory locations

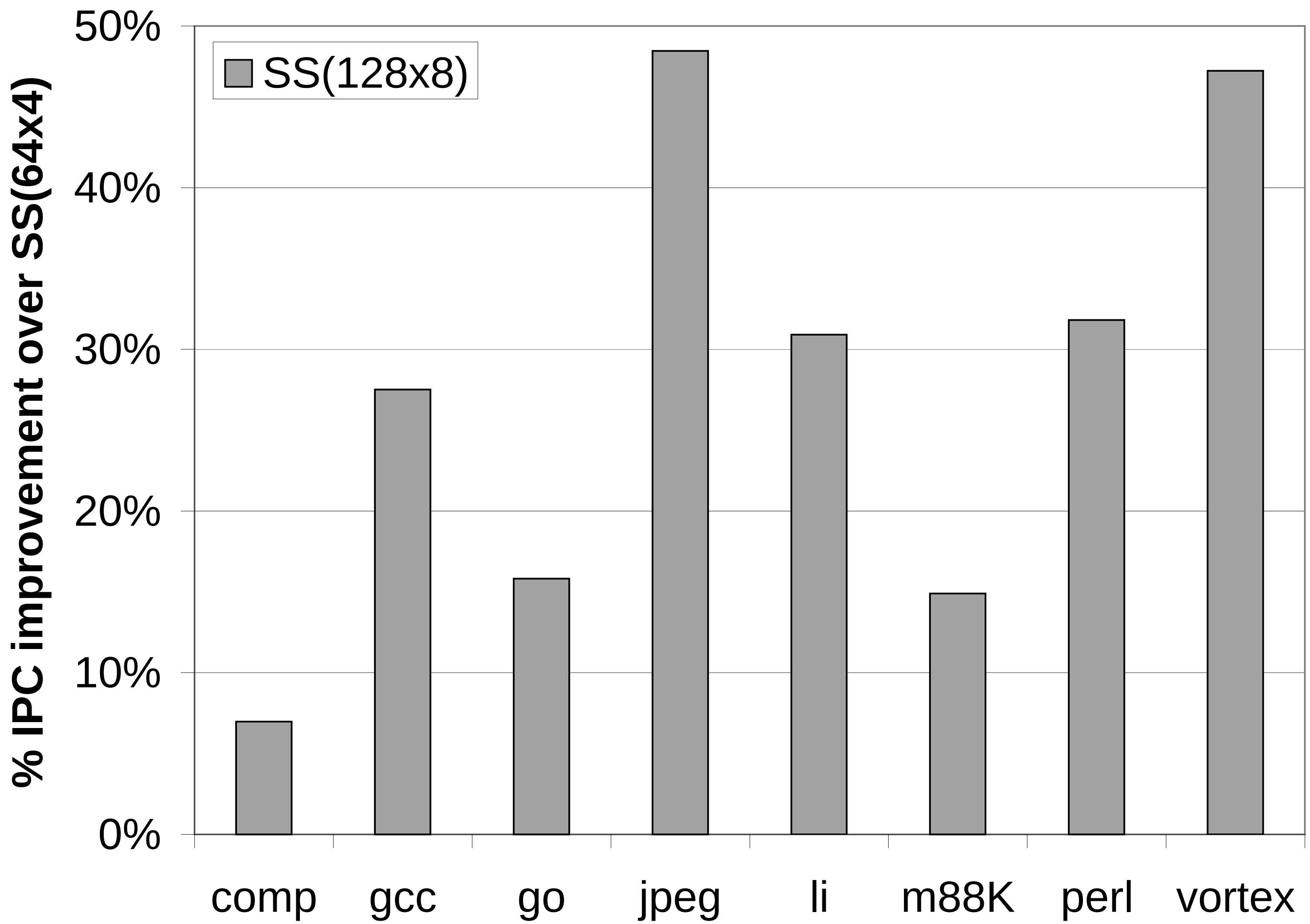
Recovery Controller



Slipstream Performance



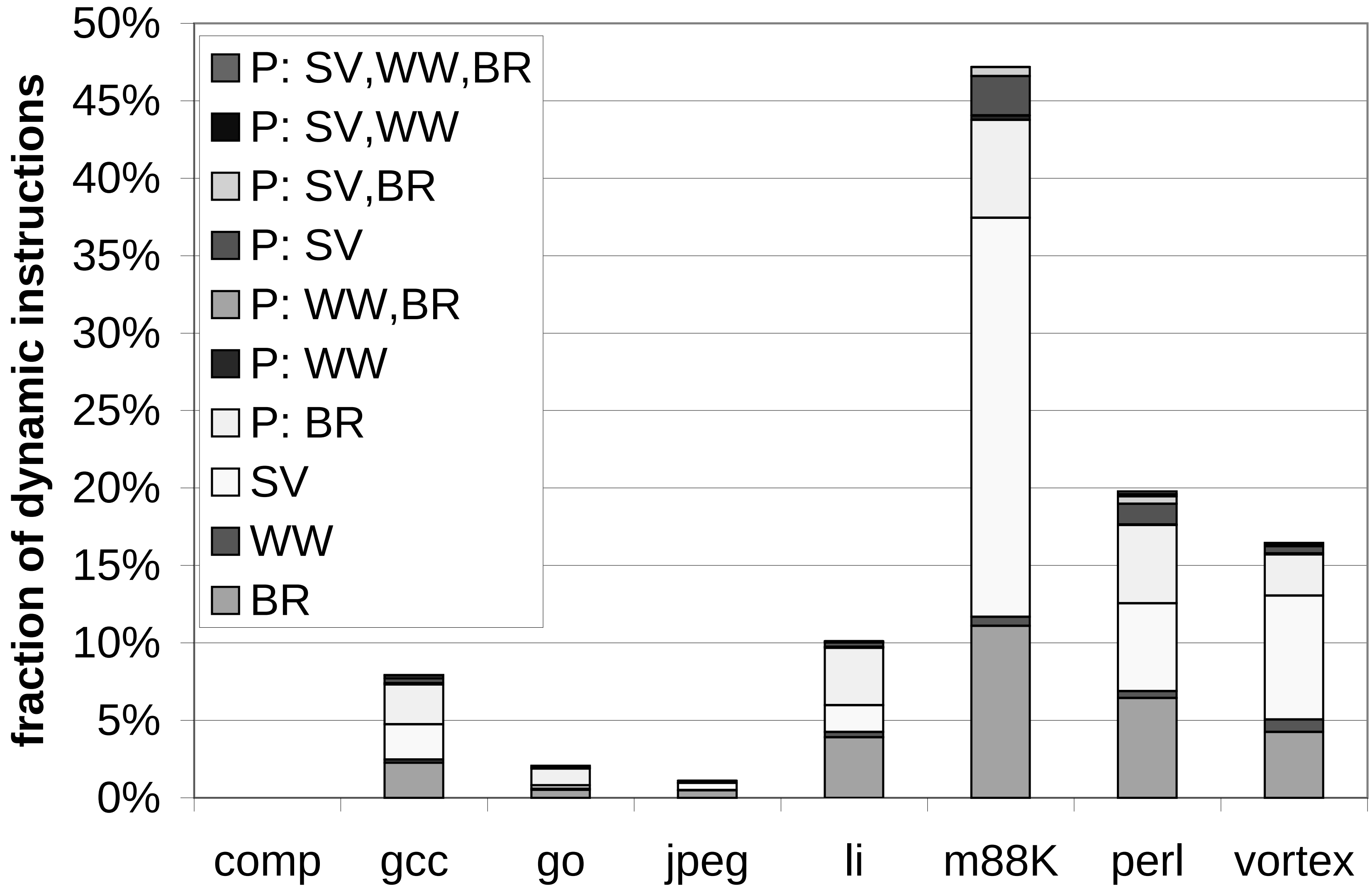
Doubling Superscalar Complexity



Results

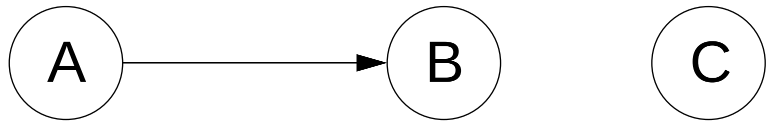
1. Exploiting existing, otherwise unused processor in a CMP speeds up a single program
2. Competitive with superscalar
   - 1/4 speedup of larger superscalar. (Improved slipstream design performs comparably or better: MICRO-33.)
   - Clock speed advantage
   - More flexible architecture
     - CMP/SMT: throughput and parallel program perf.
     - Slipstream: improved single-program perf. and reliability
     - AR-SMT / SRT: high reliability with little perf. overhead

Analysis



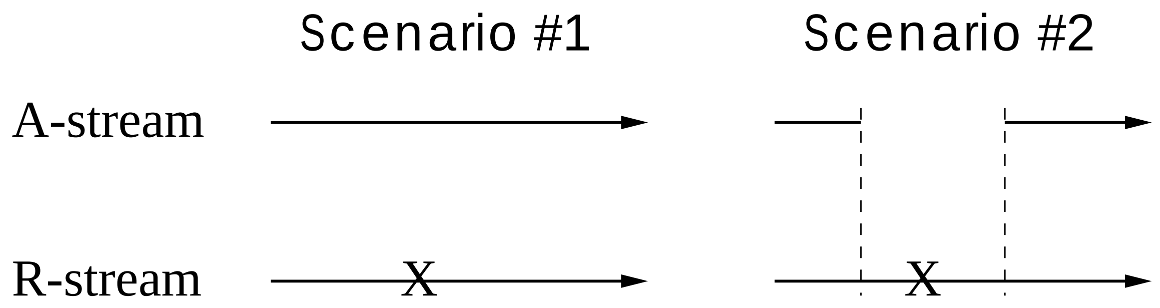
fraction of dynamic instructions
50%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
P: SV,WW,BR
P: SV,WW
P: SV,BR
P: SV
P: WW,BR
P: WW
P: BR
SV
WW
BR
comp
gcc
go
jpeg
li
m88K
perl
vortex

Analysis (cont.)



- Example: IR-detector says A and B are <u>always</u> removable, C alternates between removable/non-removable
- Instruction-based removal: Per-instruction confidence counters
  - (-) Out-of-sync counters: remove A but not B (misprediction)
- Trace-based removal: Single confidence counter per trace
  - (+) No spurious mispredictions: A is not removed unless B is
  - (-) Poor removal: Unrelated instructions (C) dilute confidence
- MICRO-33
  1. Per-instruction confidence counters (no dilution)
  2. Yet dependence chains tend to be removed as a unit

Fault Tolerance



- Scenario #1
  - Fault detectable, but indistinguishable from IR-misprediction!
  - Must assume IR-misprediction
  - Don't care about source of problem: recovery works!
  - But: 1) must re-execute erroneous instruction 2) need parity or ECC on register file and data cache
- Scenario #2: no redundancy, no coverage (future work)
- Can (potentially) tolerate all faults that affect redundantly executed instructions

Summary

- Contributions
  - Only subset of dynamic instruction stream needed for full, correct, forward progress
  - Redundant execution to speed up single program and increase reliability
  - Slipstream Processor: flexible, comprehensive functionality within a single strategic architecture
    - multiprogrammed/parallel workload perf. (CMP/SMT)
    - single-program perf. with improved reliability (slipstream)
    - high reliability with less perf. impact (AR-SMT / SRT)

Managing Complexity

- Slipstream separates new complexity from old complexity
  - Speed up single program without fundamentally reorganizing pipeline; register/memory dependence mechanisms largely untouched
  - Reduces conceptual complexity and possibly real complexity

Future Work

1. Slipstream Processors
   - Understanding performance
   - Microarchitecture design space
   - Pipeline organization
   - Fault tolerance
   - System-level issues (including memory hierarchy)
   - Adaptivity
2. Fundamental variations of Slipstream Paradigm
   - Streamlining R-stream
   - Other A-stream shortening approaches
   - Scaling to N threads
   - Approximate A-streams
3. Other novel CMP/SMT applications