Processor-level Selective Replication

N. Nakka, K. Pattabiraman, R. K. Iyer

Coordinated Science Laboratory

University of Illinois, Urbana-Champaign

Why another replication technique?

Contributions

- Instructions replicated selectively
- Replicate only computations of critical variables
- Extensive fault injection-based coverage evaluation
- 63% coverage as compared to 72% coverage for Full Duplication
- Combined metric for detection and overhead

Full Duplication vs Selective Replication

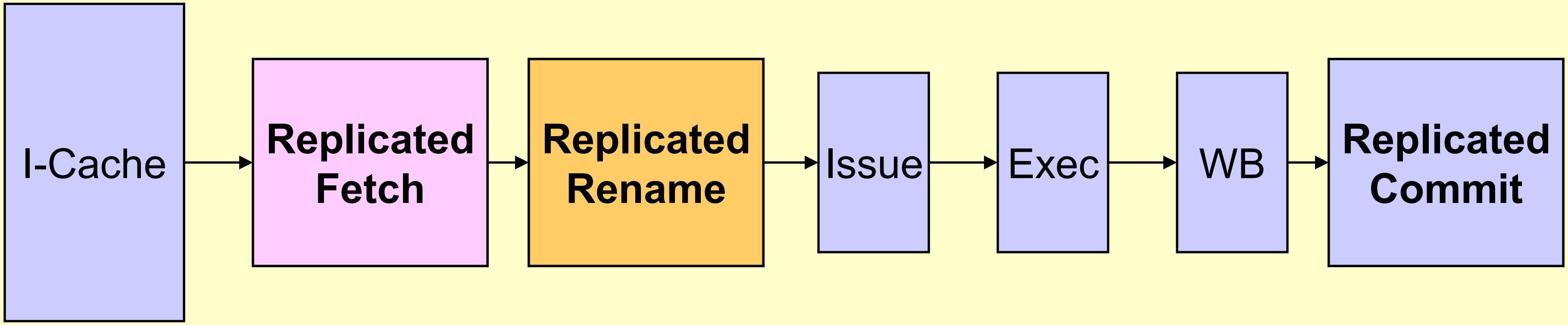
- Full duplication comes at a price
  - ~~Performance overhead up to 50%~~
    - Lower performance overhead ranging from 11% to 22%
  - ~~Area overhead –thread synchronization hardware~~
    - Simple hardware structures to replicate instruction
    - Results stored in re-order buffer
  - ~~Benign error detections (75% of injected errors)~~
    - Benign error detection reduced by 18%

Reliability – Selective Replication

- Two questions arise:
  - What to replicate? and How to replicate?
- Critical variables [Pattabiraman ’05]
  - High probability of error propagating to variable
  - High likelihood of variable error leading to crash or FSV
- Critical variables derived from dynamic dependency graph (DDG)
- *Fanout* found to be best heuristic
- Replicate only computation of critical variable
  - Backward slice from DDG

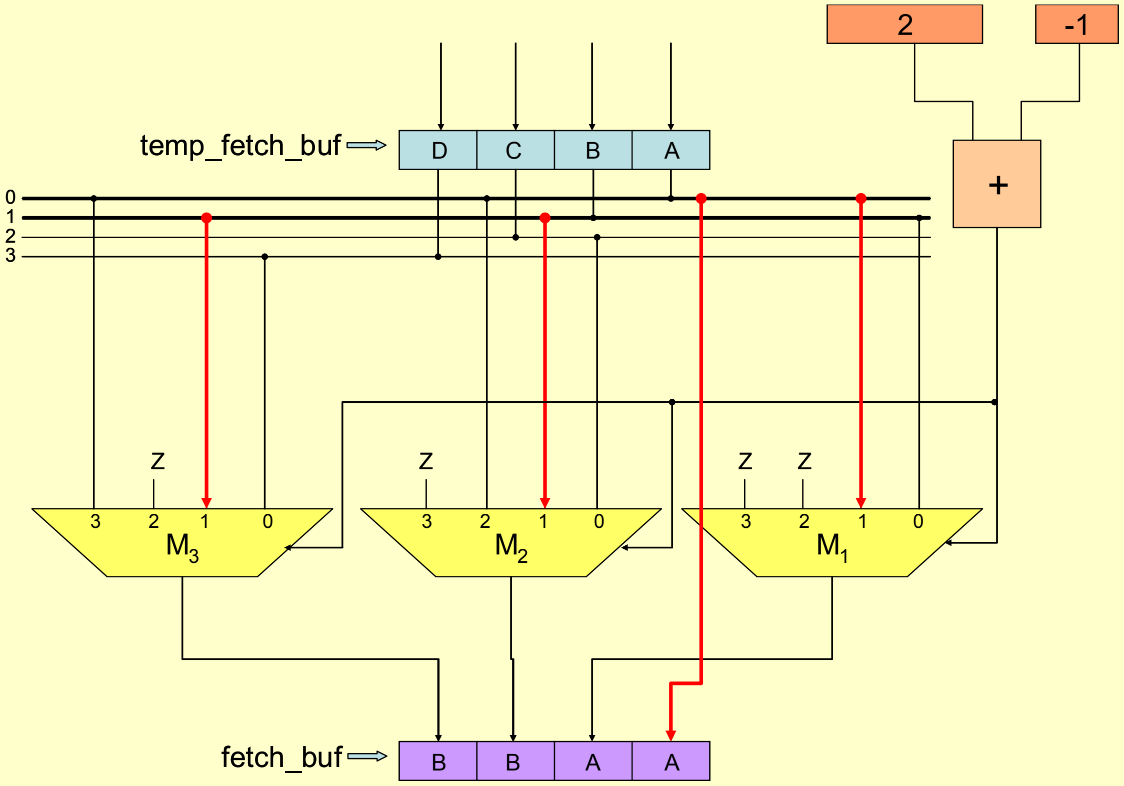
Reliability – Selective Replication

- Modify fetch, rename and commit mechanisms



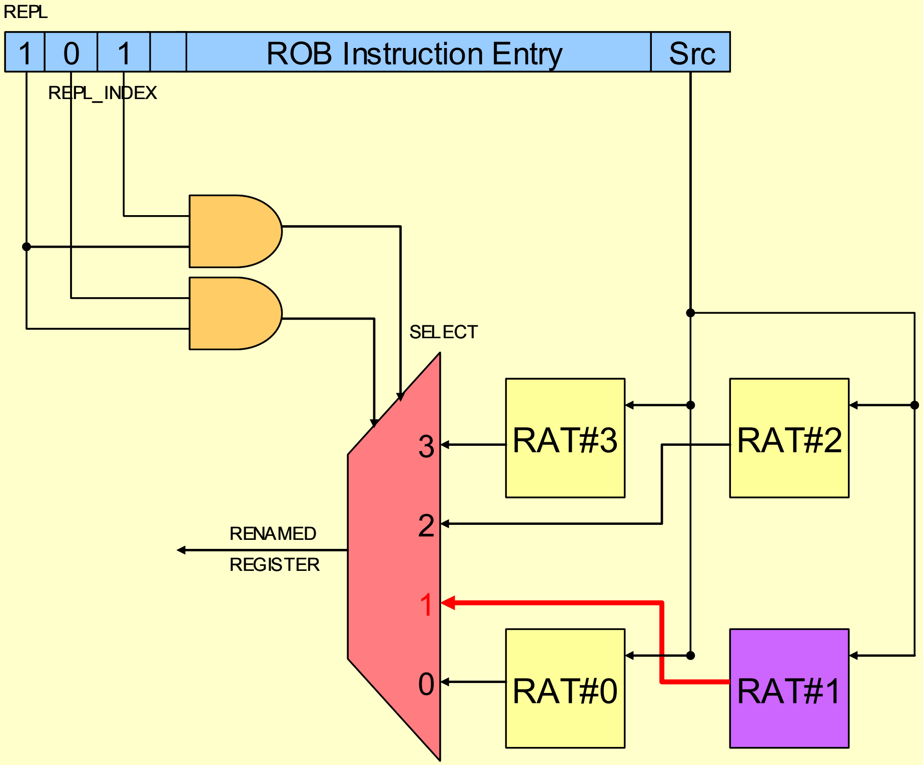
Replicated Fetch

- Instructions fetched into *temp_fetch_buf*
- Replicas routed through mux to *fetch_buf*
- Replicas dispatched like normal instructions



Replicated Rename

- REPL and REPL_INDEX fields in re-order buffer
- Replica maintains dependencies within itself
- Corresponding Register Alias Table, RAT, looked up



Evaluation

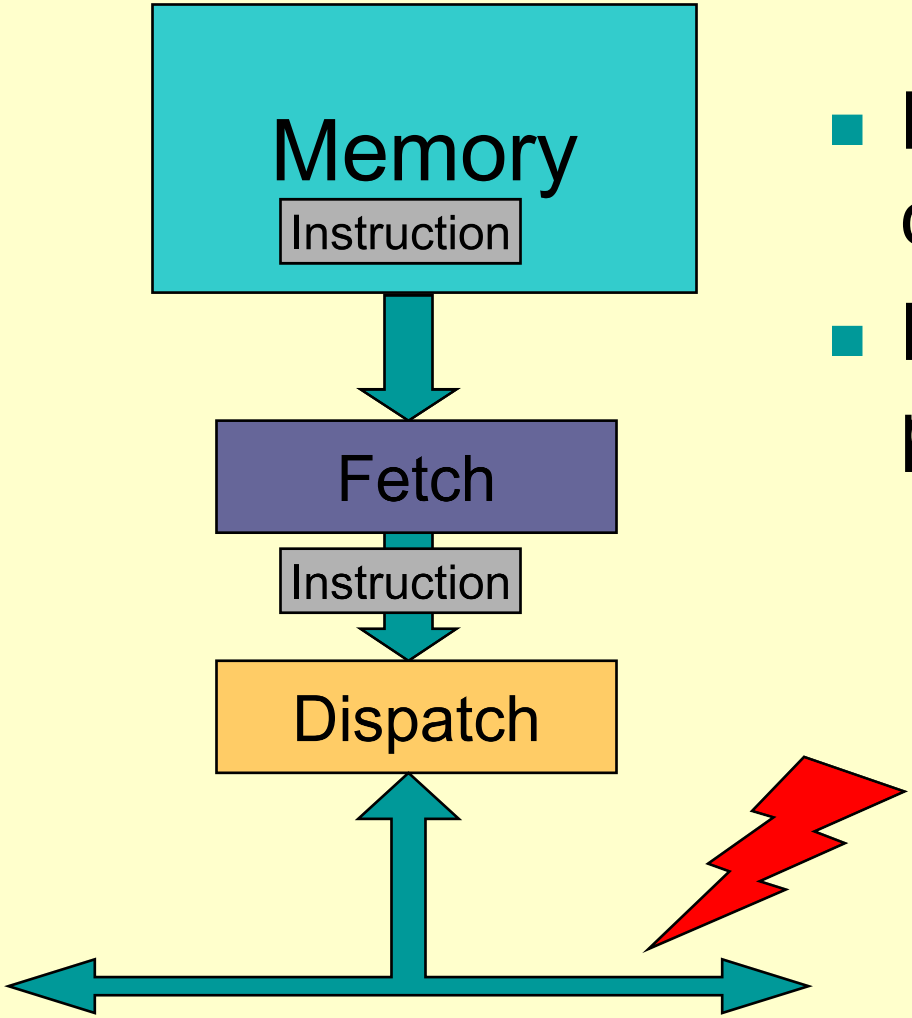
- Performance overhead
- Coverage evaluated using fault-injection
- Workload: Siemens suite of benchmarks
- *SimpleScalar* augmented for selective replication
  - Introduced hooks for fault-injection

Benchmark	#lines of code	# static insts	#dynamic insts
schedule	412	100504	77702
schedule2	373	102520	208324
print_tokens	727	82296	271976
print_tokens2	569	80568	77179

Fault Model

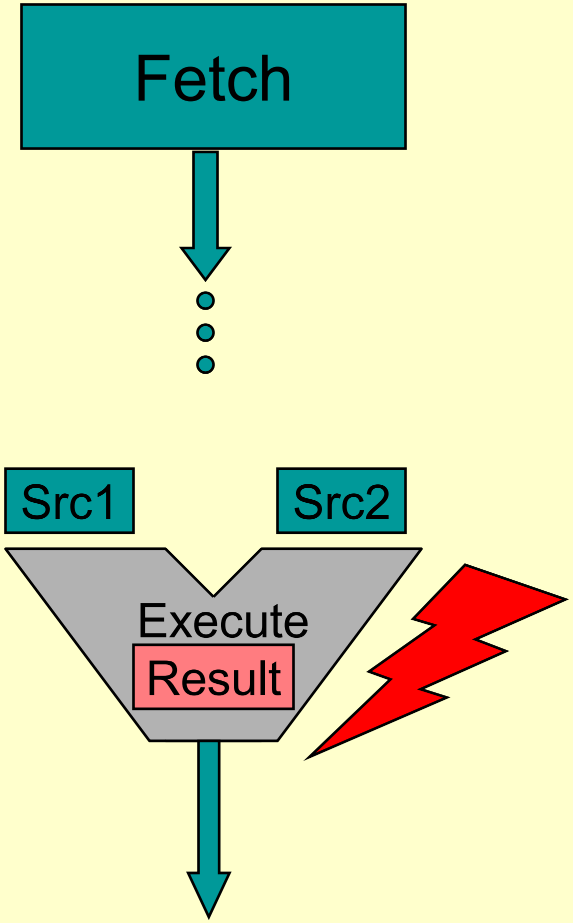
- Scope: Errors within the processor
  - Instruction errors
  - Data errors
- Common-mode errors: Not injected
  - Errors in memory, cache
  - Errors in fetch mechanism

Fault Model - Instruction Errors



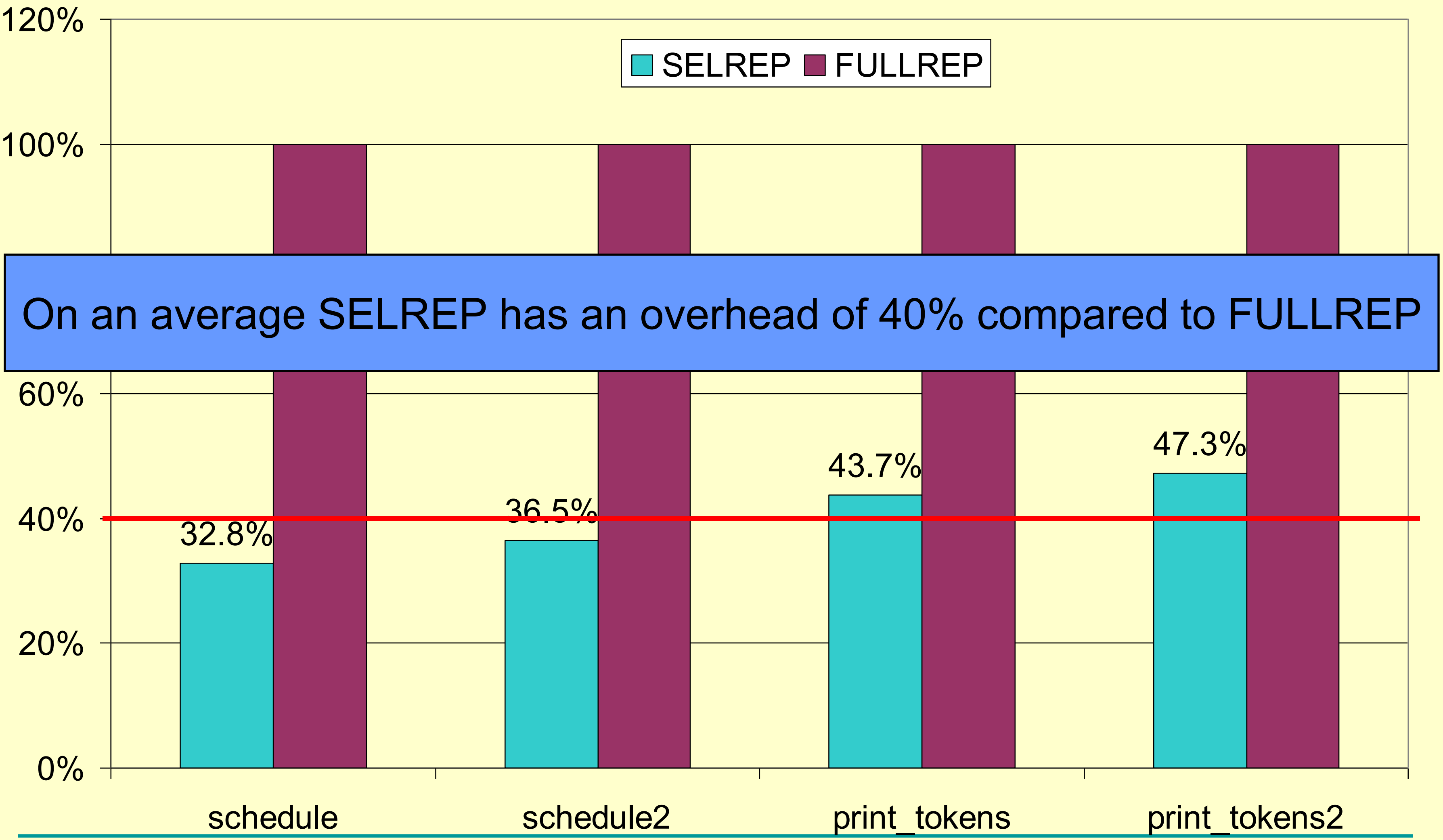
- During transfer from cache to pipeline
- During decode in pipeline

Fault Model - Data Errors



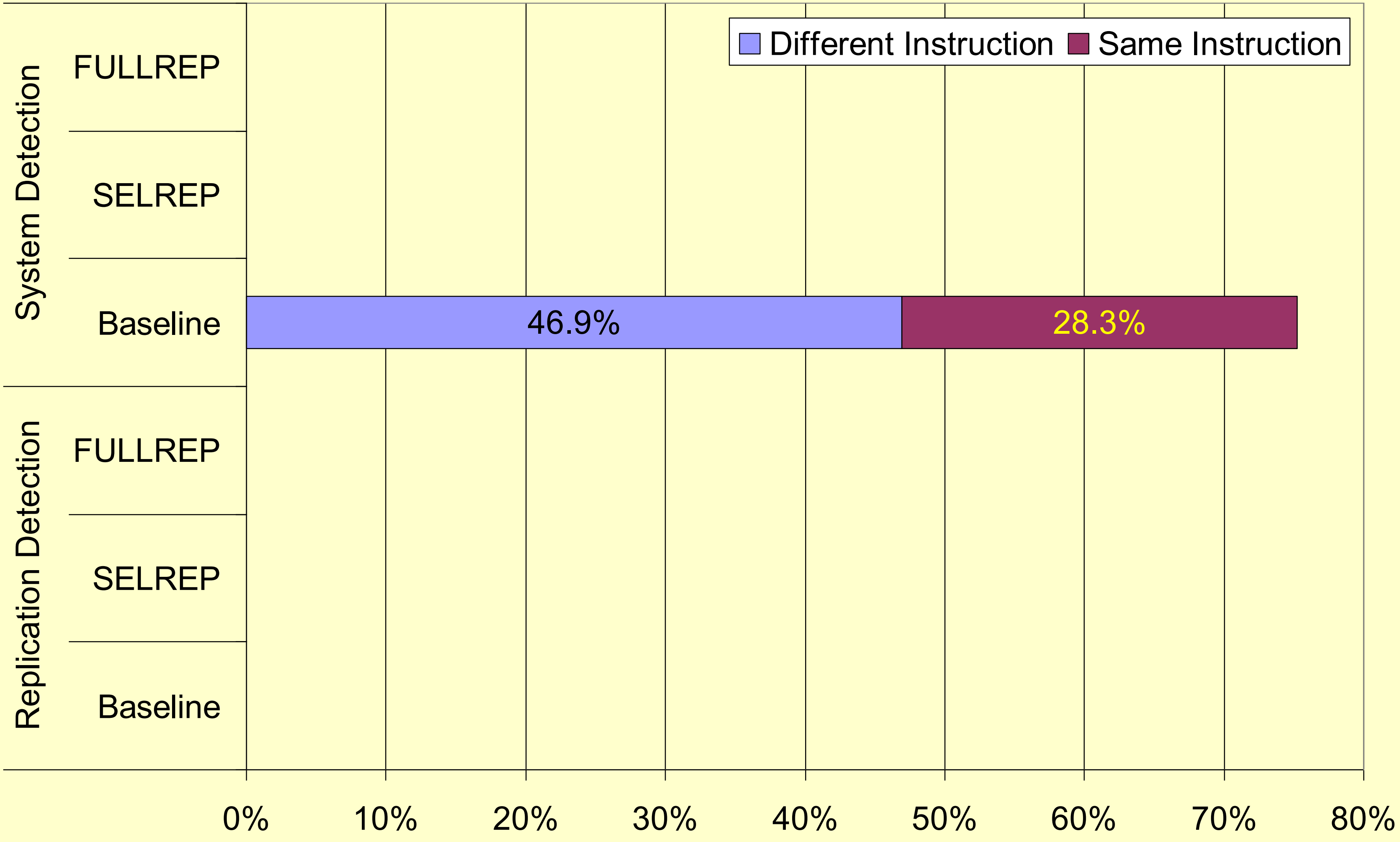
- Errors in the output of a functional unit
  - written to a register
  - used as an effective address for a memory access instruction

Normalized performance overhead

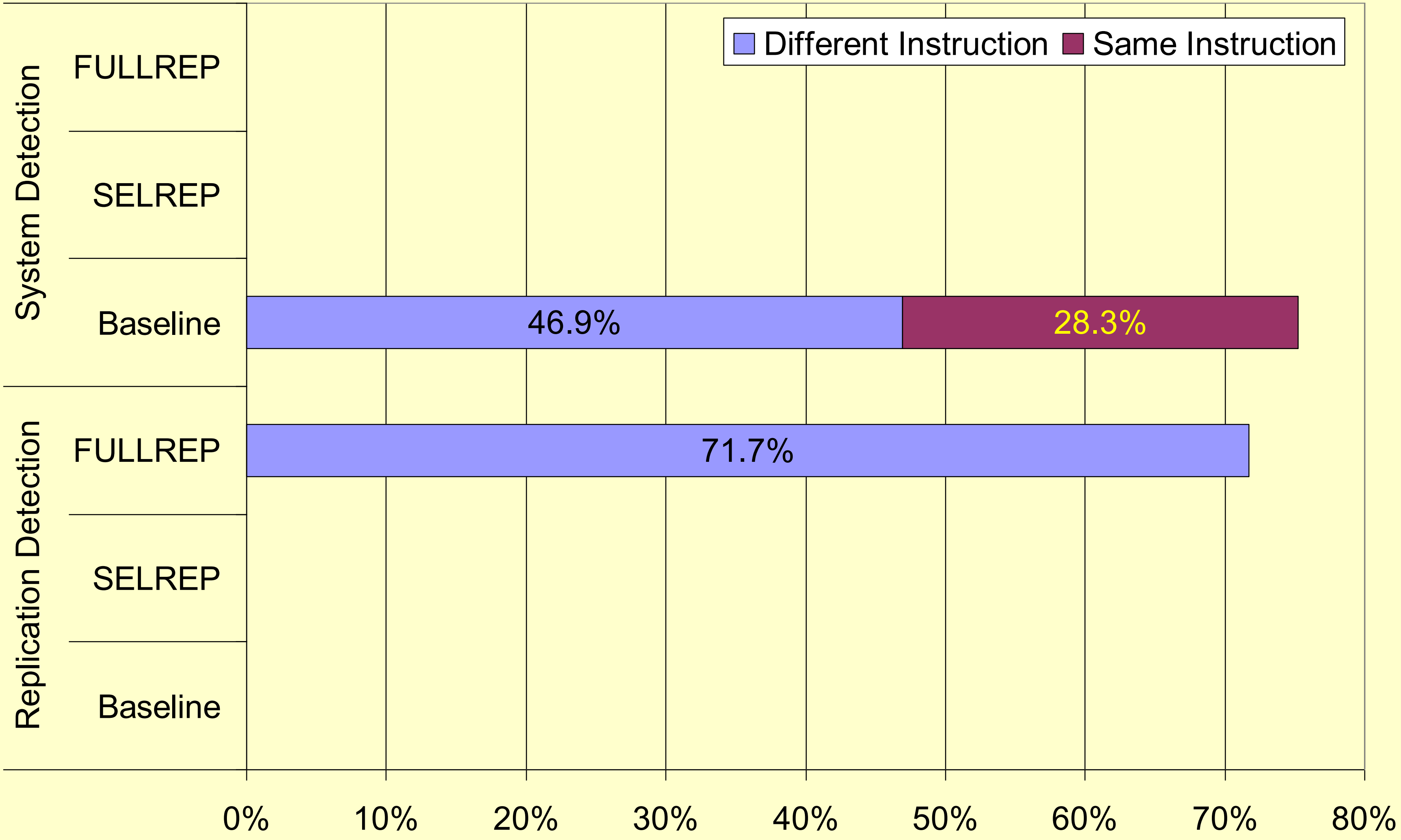


On an average SELREP has an overhead of 40% compared to FULLREP

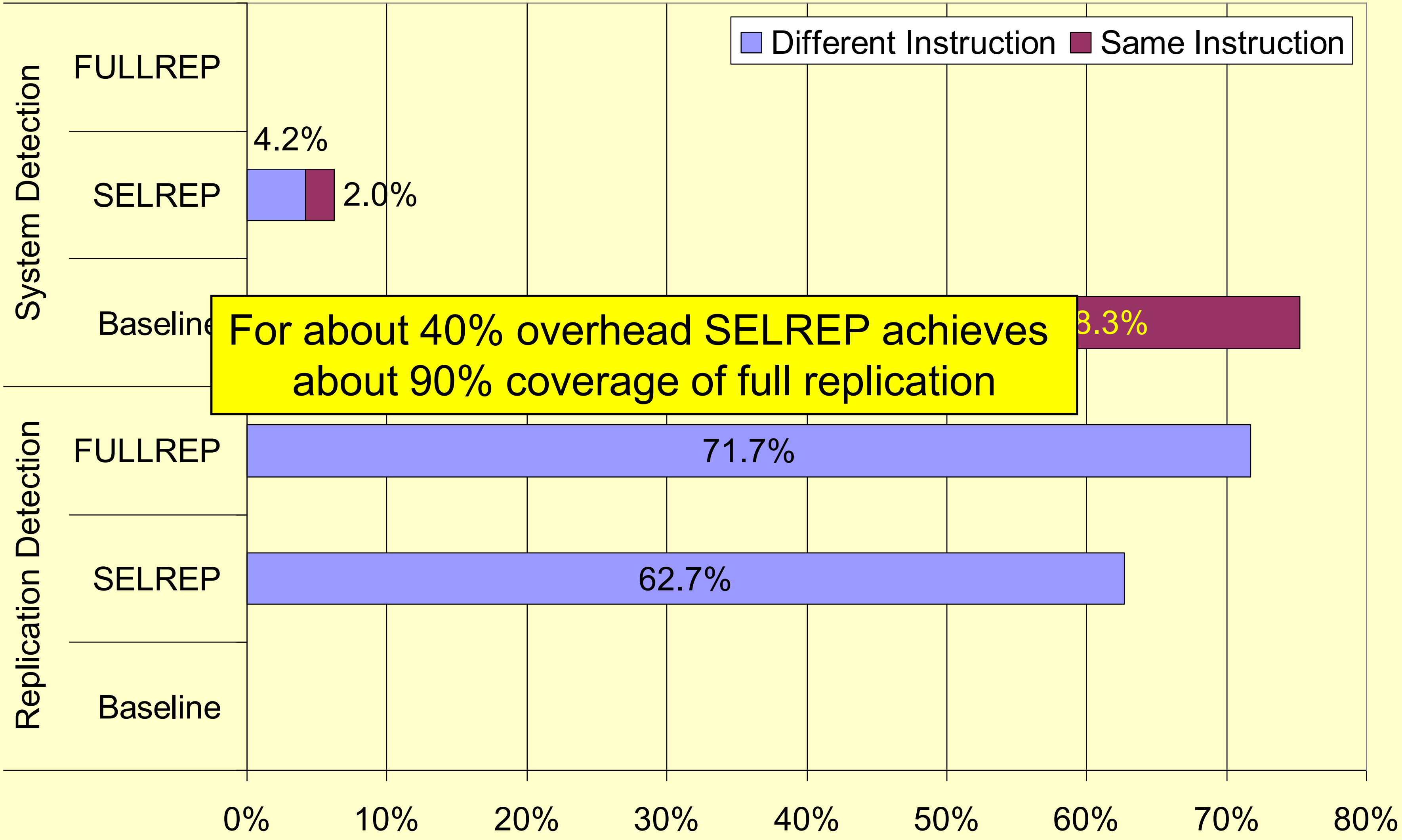
SELREP/FULLREP – Detection



SELREP/FULLREP – Detection

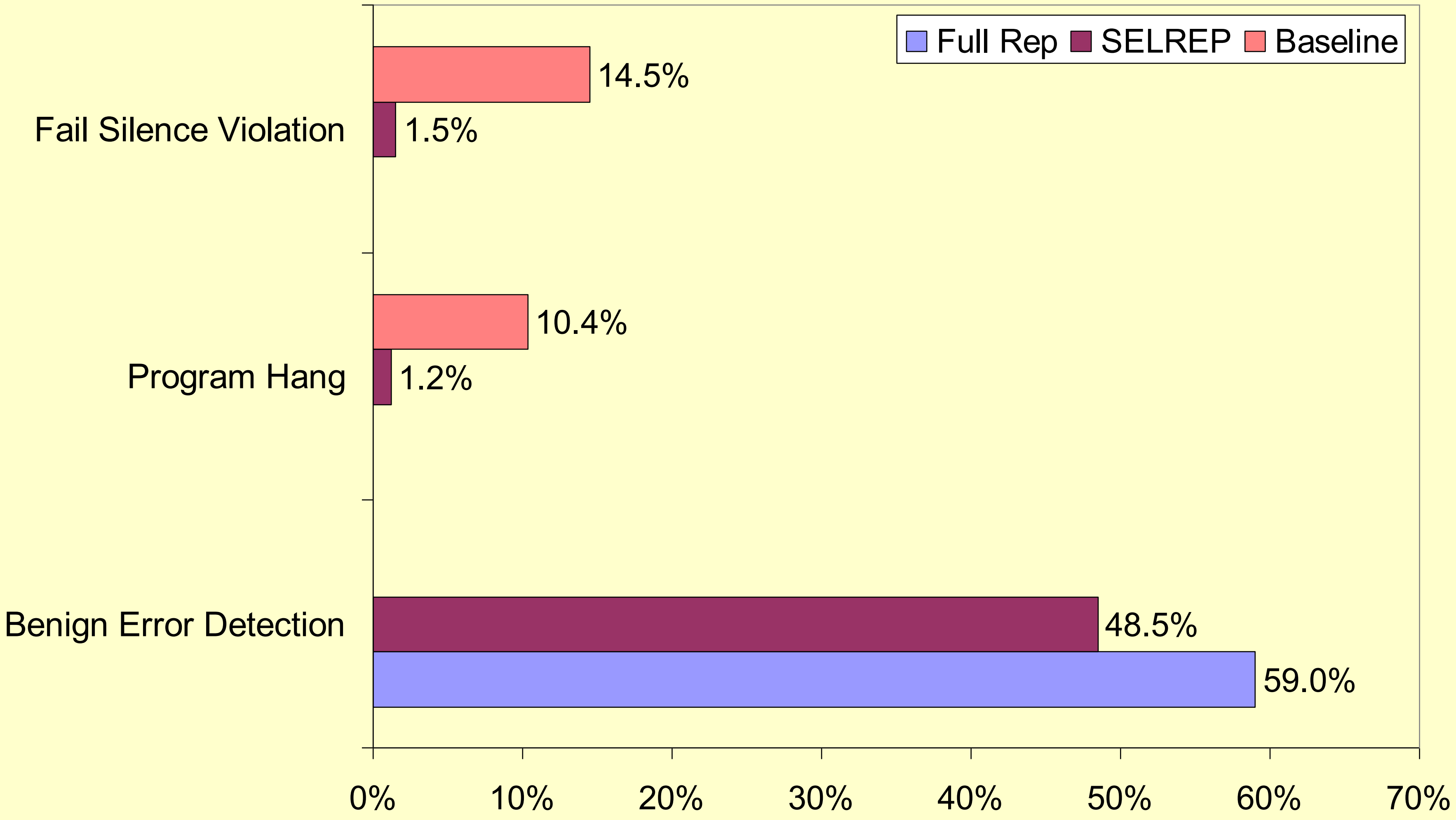


SELREP/FULLREP – Detection



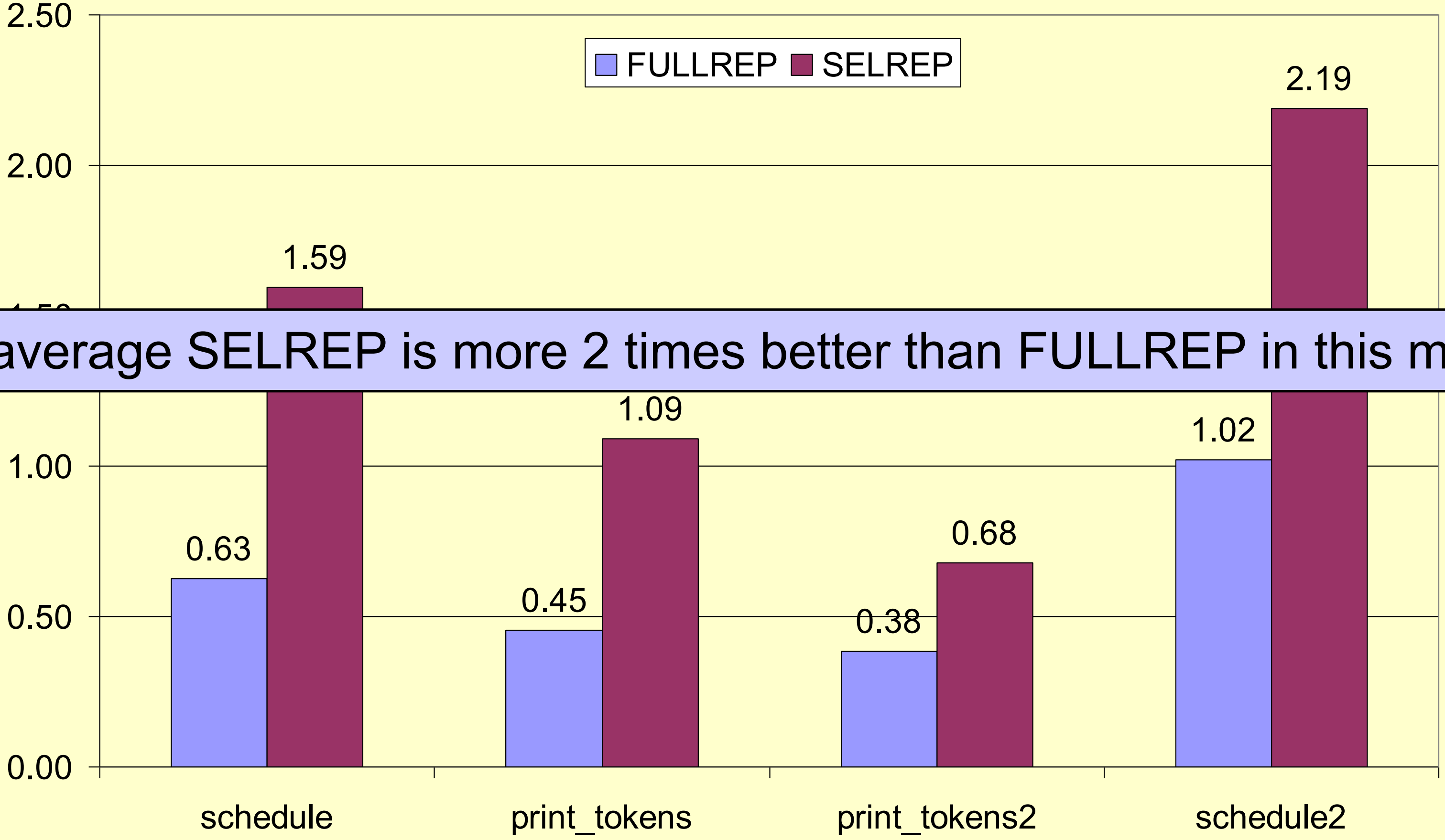
For about 40% overhead SELREP achieves about 90% coverage of full replication

SELREP/FULLREP– FSVs & Hangs



Combined Metric

$$\mathcal{M} = \frac{Detection - Benign\ Error\ Detection}{Overhead}$$



On an average SELREP is more 2 times better than FULLREP in this metric

Conclusions

- Presented a technique for selective replication of instructions
- Replicated only computation of critical variables
- Low overhead and minimal additional hardware
- Fault injection based coverage evaluation
- Compared to Full Duplication
  - About 59% less overhead for SELREP
  - Up to 88% coverage
  - 17% reduction in benign error detection

Where do we stand in replication?

Technique	Description	Discussion
DIVA [Austin, Weaver '01]	Simple checker checks complex main core	➢Multiple checkers non-trivial ➢Errors in control-flow ➢Prediction stream omission errors
SRTR [Vijaykumar et. al. '02]	Two threads check each other's values	➢Introduces many hardware structures ➢Indirect accesses for comparison ➢Replication mechanism not detailed ➢Detailed architecture presented
Introspection [Qureshi, Patt '05]	Redundant thread execution during cache miss	➢Introduces introspection buffer ➢High overhead for low-memory access applications ➢Some issues e.g., register file bandwidth, prediction outcome check, not addressed
Selective Replication	Dynamically replicates instructions	➢Simple h/w to replicate selectively ➢Switching between modes ➢Result stored in re-order buffer ➢Not all instructions replicated

Why the Siemens Suite?

- Extensively used in the testing community
- Show high level of data dependencies
- Moderately-sized to enable use in fault-injection experiments