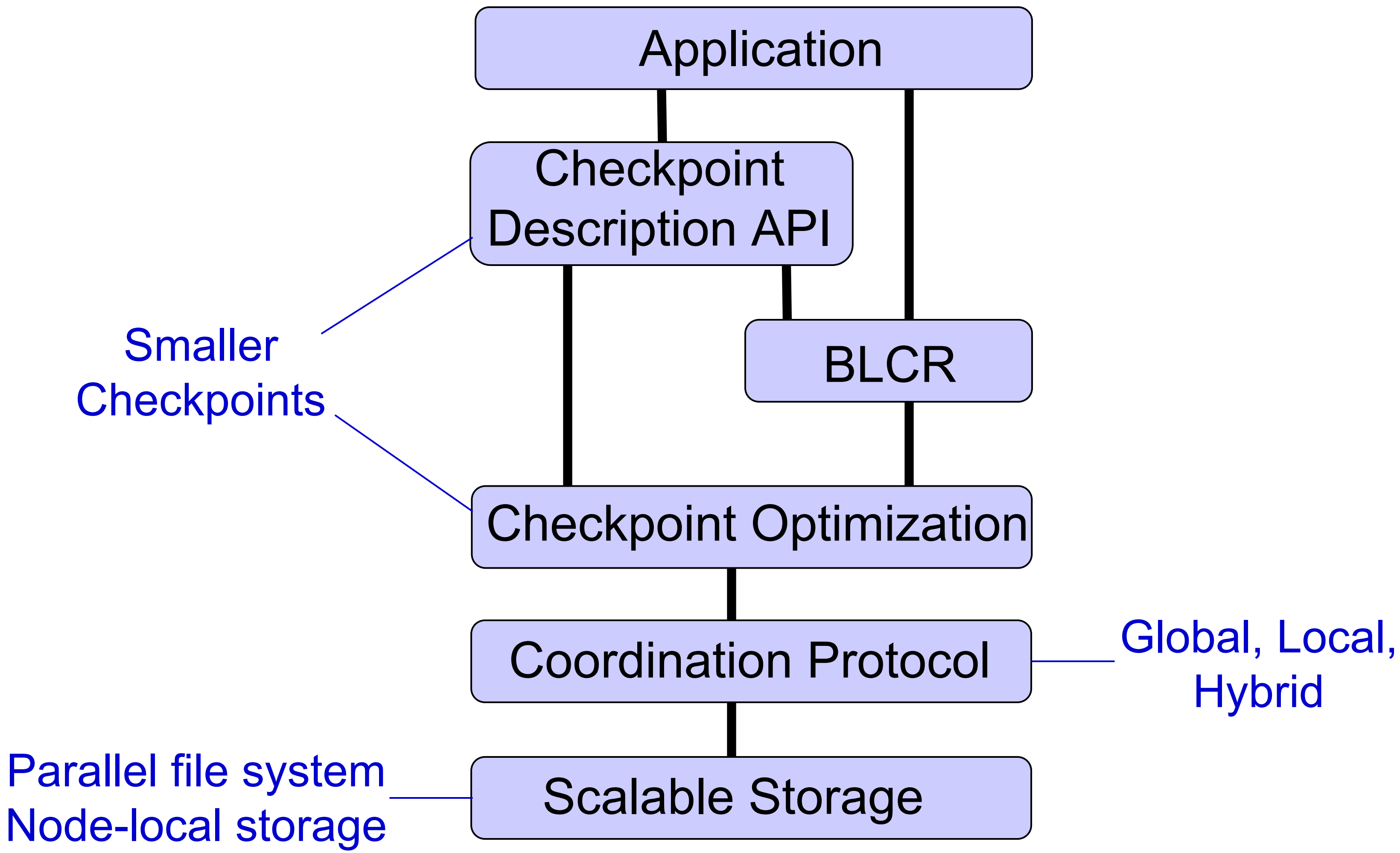
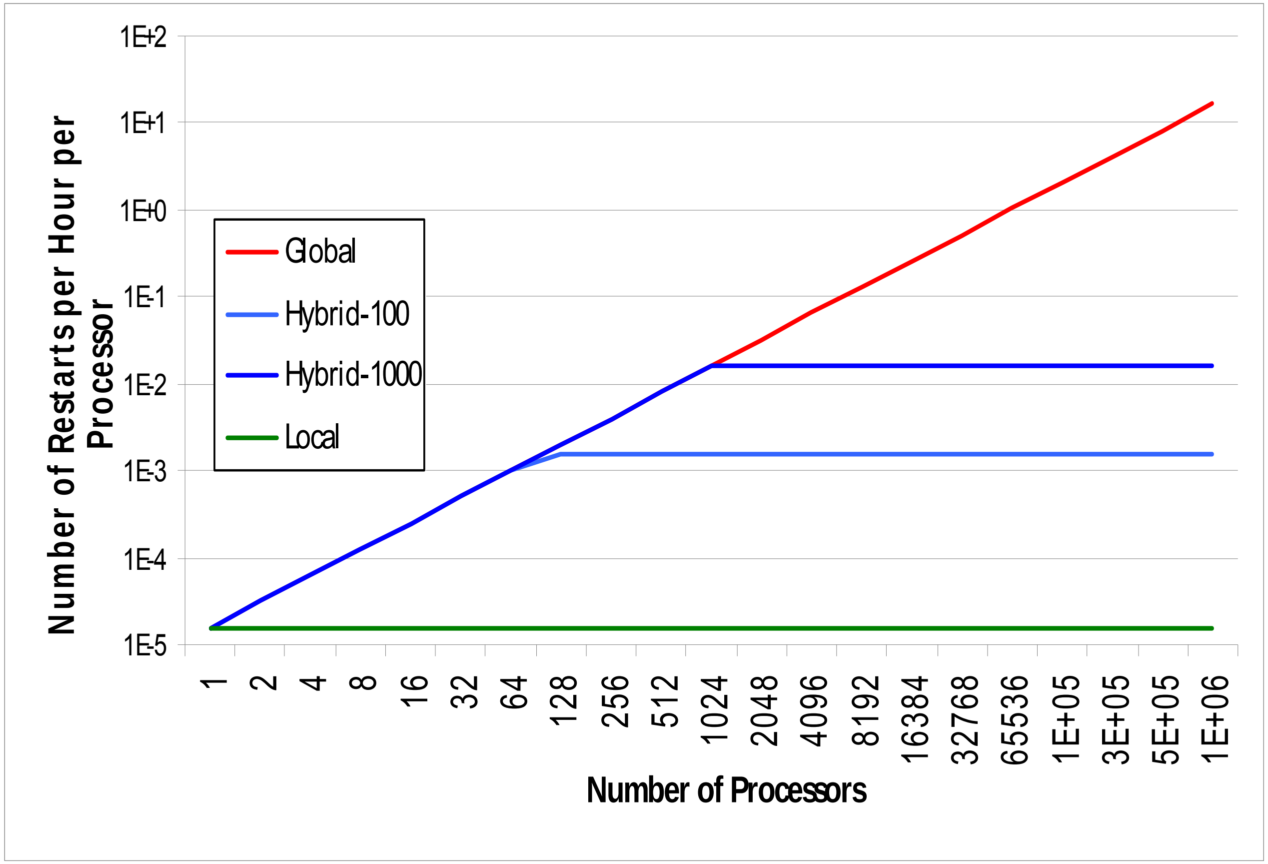
Accurate Prediction of Soft Error Vulnerability of Scientific Applications

Greg Bronevetsky
Post-doctoral Fellow
Lawrence Livermore National Lab

Application/System integrated infrastructure can offer good performance, scalability



Localized rollback recovery scales with system size



More localized → fewer checkpoints, restarts

Don't need consistent snapshots

10 MTTF per processor

Accurate Prediction of Soft Error Vulnerability of Scientific Applications

Greg Bronevetsky

Post-doctoral Fellow

Lawrence Livermore National Lab

Soft error: one-time corruption of system state

- Examples: Memory bit-flips, erroneous computations
- Caused by
  - Chip variability
  - Charged particles passing through transistors
    - Decay of packaging materials (Lead^{208}, Boron^{10})
    - Fission due to cosmic neutrons
  - Temperature, power fluctuations

Soft errors are a critical reliability challenge for supercomputers

- Real Machines:
  - ASCI Q: 26 radiation-induced errors/week
  - Similar-size Cray XD1: 109 errors/week (estimated)
  - BlueGene/L: 3-4 L1 cache bit flips/day
- Problem grows worse with time
  - Larger machines $\Rightarrow$ larger error probability
  - SRAMs growing exponentially more vulnerable per chip

We must understand the impact of soft errors on applications

- Soft errors corrupt application state
- May lead to crashes or corrupt output
- Need to detect/tolerate soft errors
  - State of the art: checkers/correctors for individual algorithms
  - No general solution
- Must first understand how errors affect applications
  - Identify problem
  - Focus efforts

Prior work says very little about most applications

- Prior fault analysis work focuses on injecting errors into individual applications
  - [Lu and Reed, SC04]: Linux + MPICH + Cactus, NAMD, CAM
  - [Messer et al, ICSDN00]: Linux + Apache and Linux + Java (Jess, DB, Javac, Jack)
  - [Some et al, AC02]: Lynx + Mars texture segmentation application

  …
- Where's my application?

Extending vulnerability characterization to more applications

- Goal: general purpose vulnerability characterization
  - Same accuracy as per-application fault injection
  - Much cheaper
- Initial steps
  - Fault injection iterative linear algebra methods
  - Library-based fault vulnerability analysis

  …

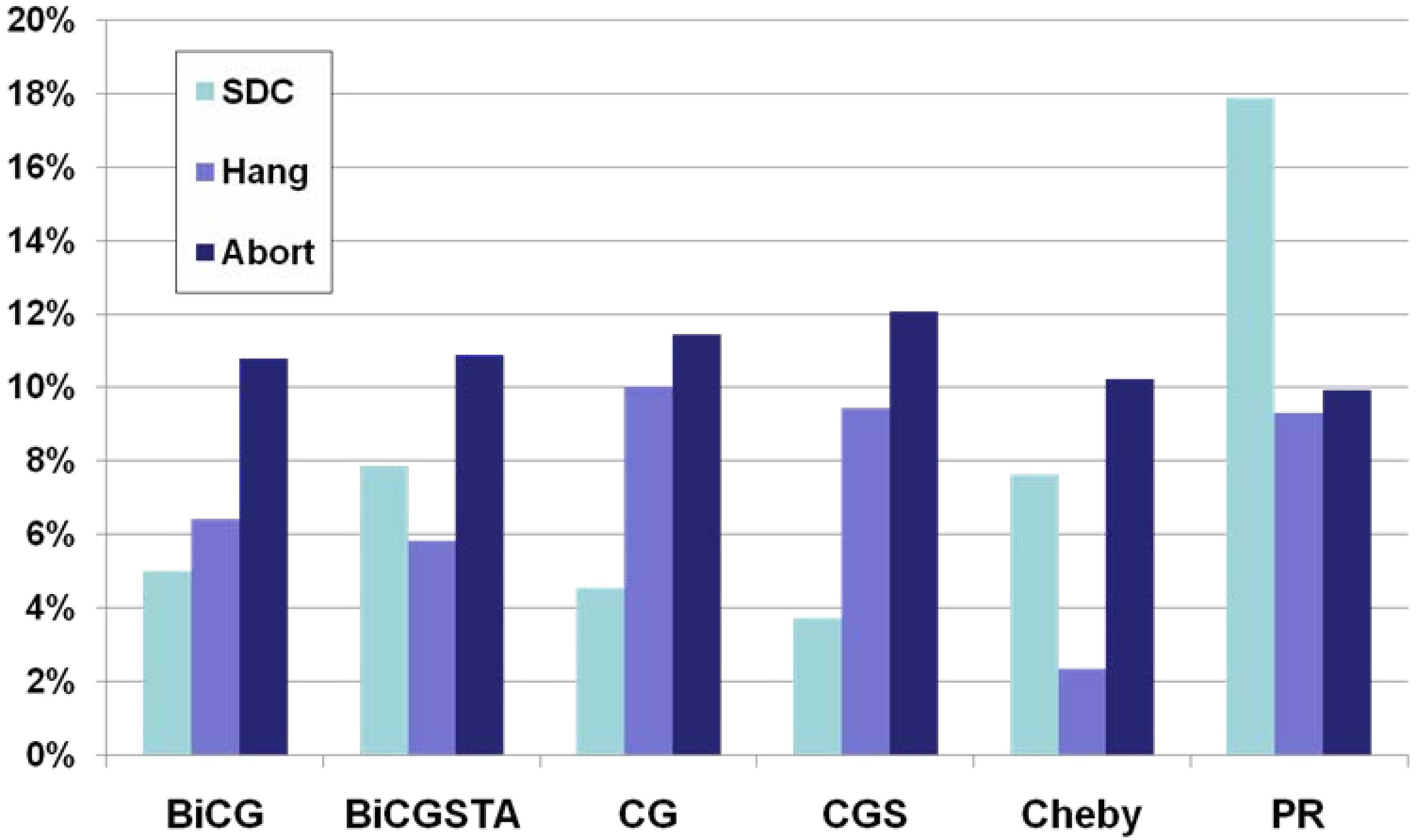
Step 1: Analyzing fault vulnerability of iterative methods

- Target domain:
  solvers for sparse linear problem Ax=b
- Goal:
  understand error vulnerability of <u>class</u> of algorithms
  - Raw error rates
  - Effectiveness of potential solutions
- Error model: memory bit-flips

Possible run outcomes

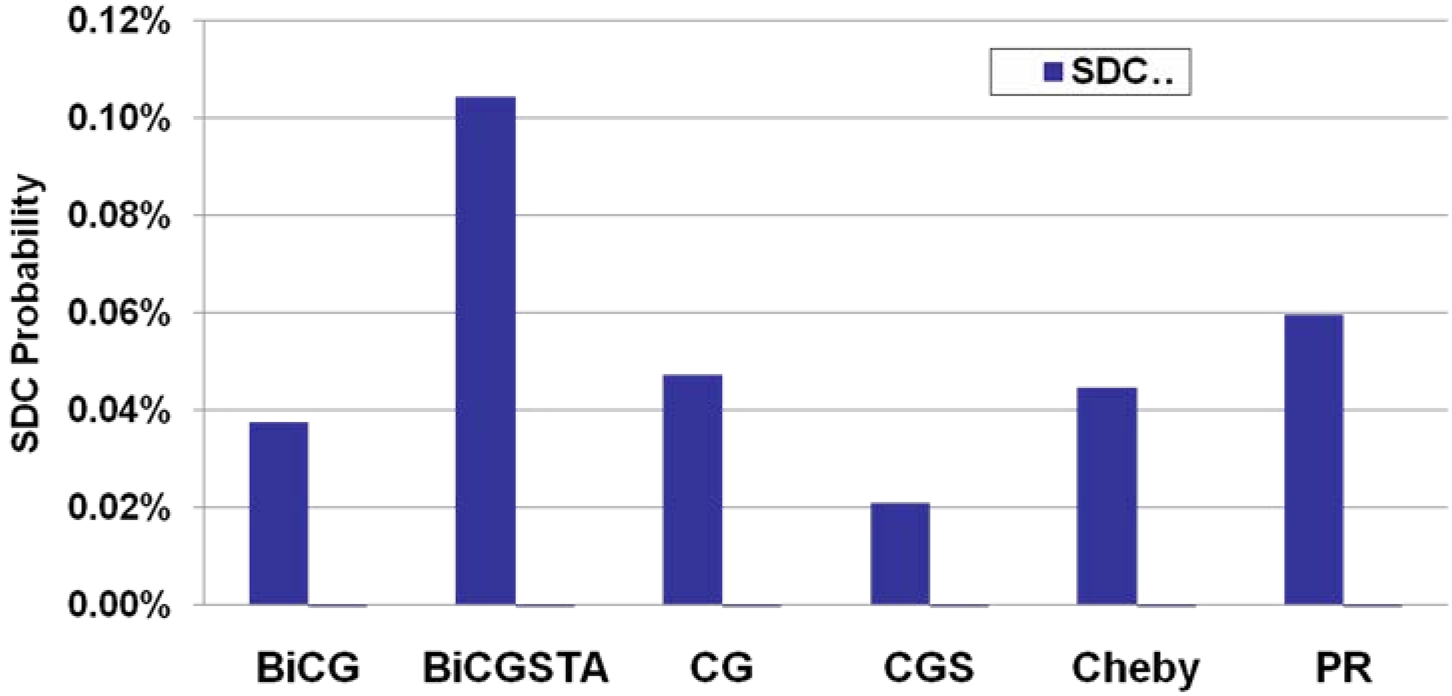
- Success: <10% error
- Silent Data Corruption (SDC): ≥10% error
- Hang: method doesn’t reach target tolerance
- Abort: SegFault or failed SparseLib check

Errors cause SDCs, Hangs, Aborts in ~8-10%, each



Large scale applications vulnerable to silent data corruptions

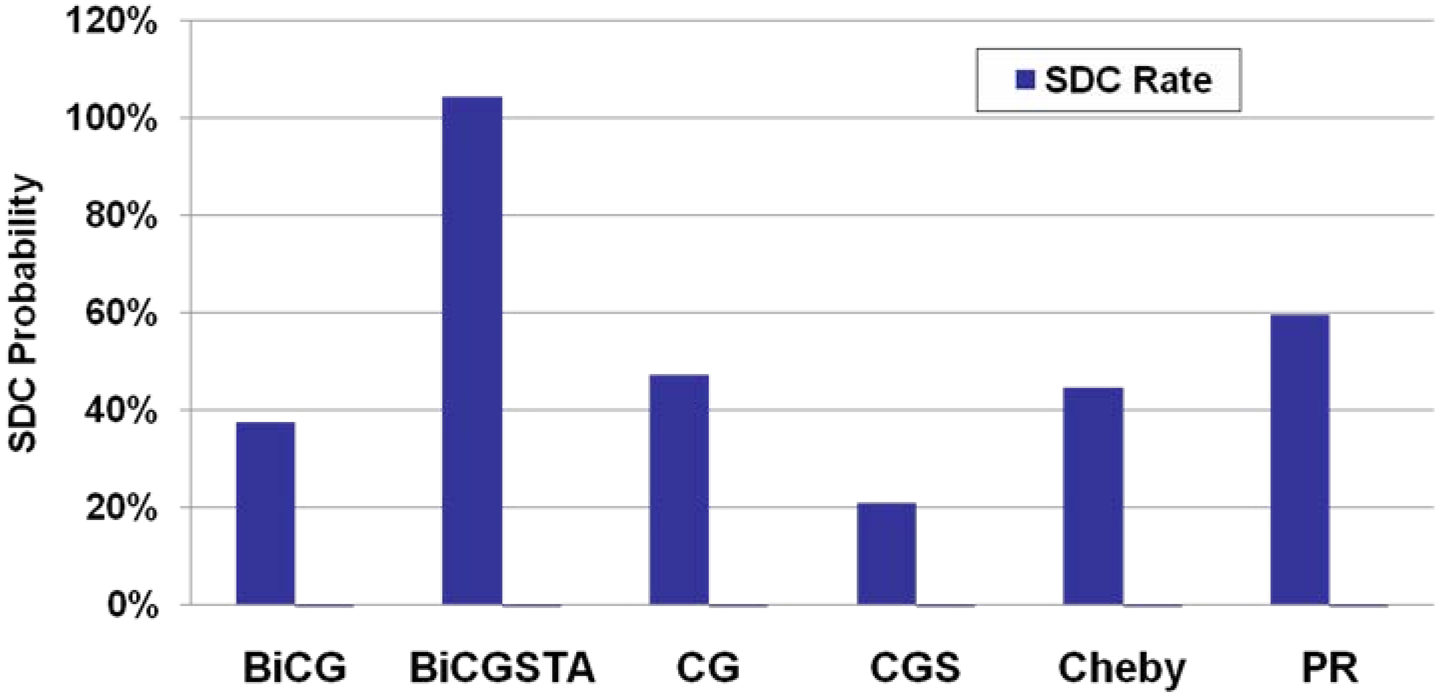
- Scaled to 1-day, 1,000-processor run of application that only calls iterative method



10FIT/MB DRAM (1,000-5,000 Raw FIT/MB, 90%-98% effective error correction)

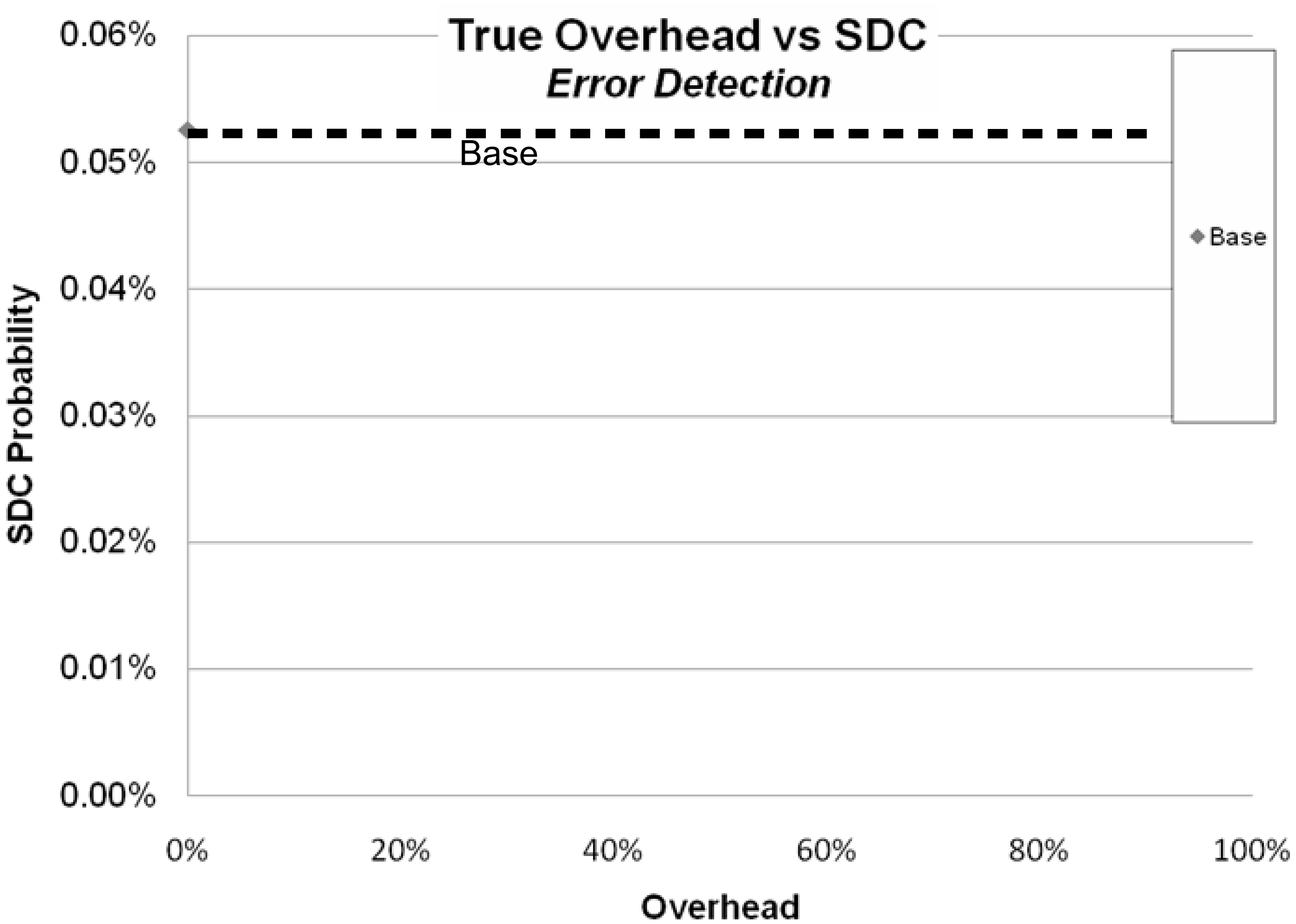
Larger scale applications **even more** vulnerable to silent data corruptions

- Scaled to 10-day, 100,000-processor run of application that only calls iterative method

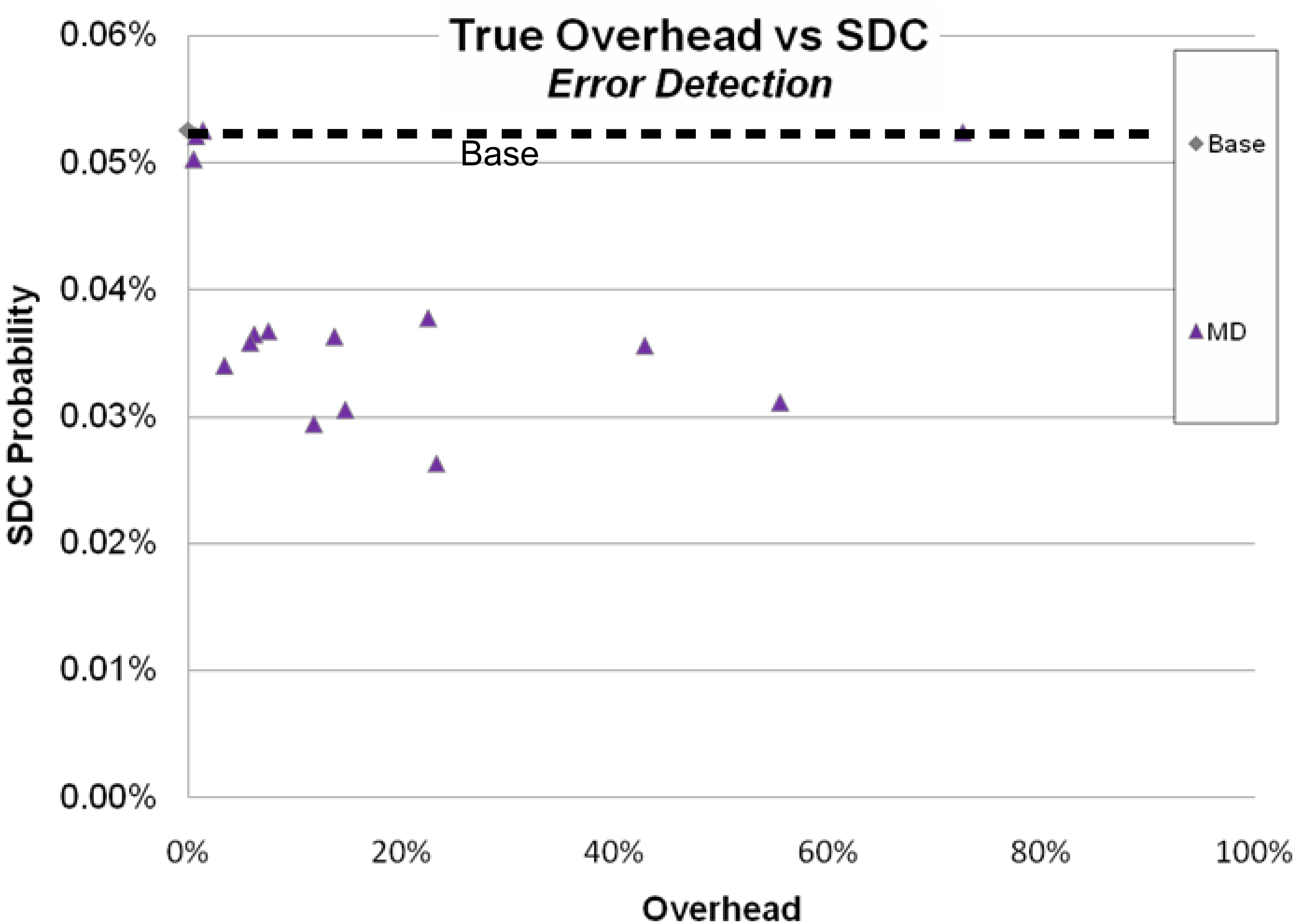


10FIT/MB DRAM (1,000-5,000 Raw FIT/MB, 90%-98% effective error correction)

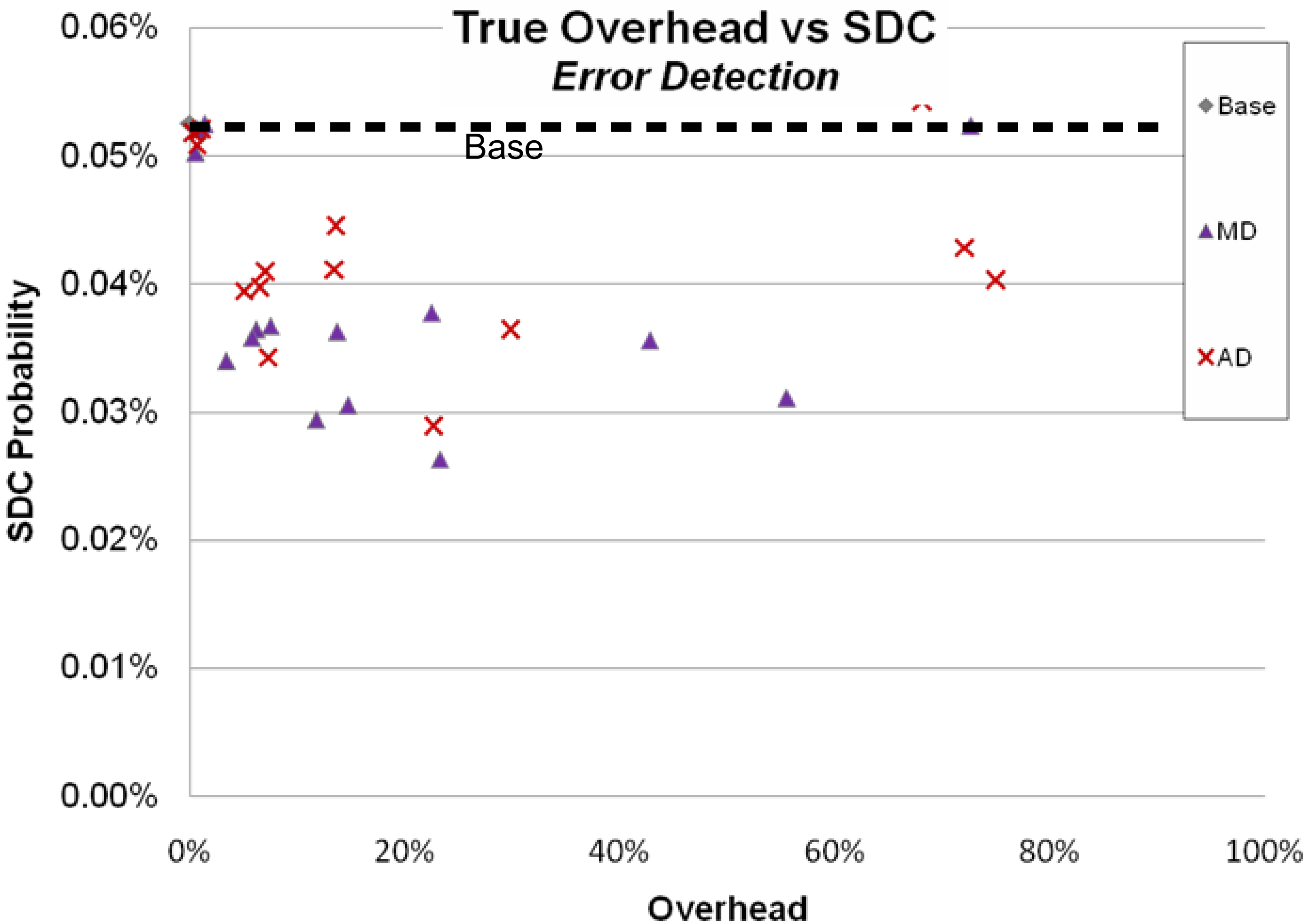
Error Detectors



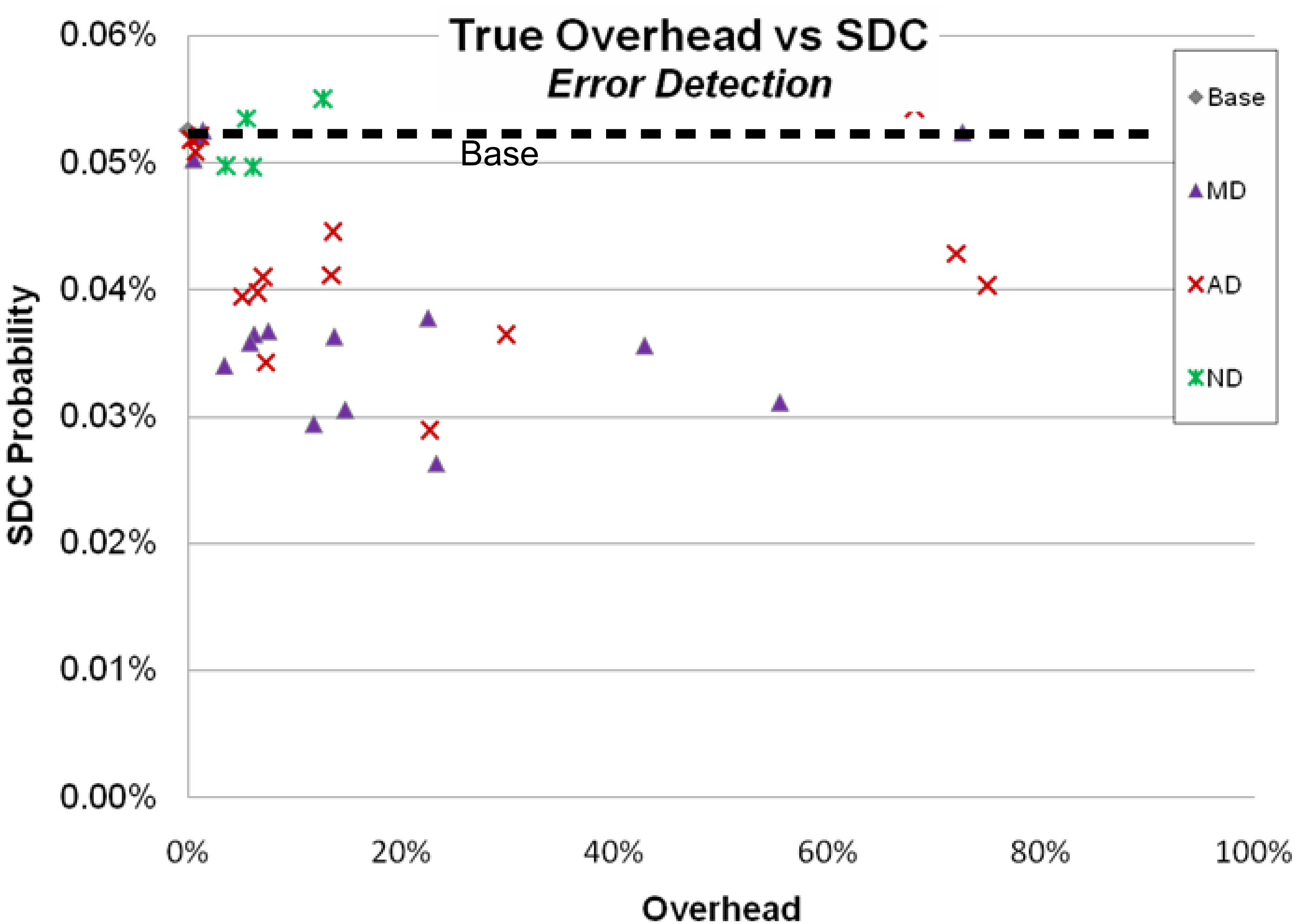
Convergence detectors reduce SDC at <20% overhead



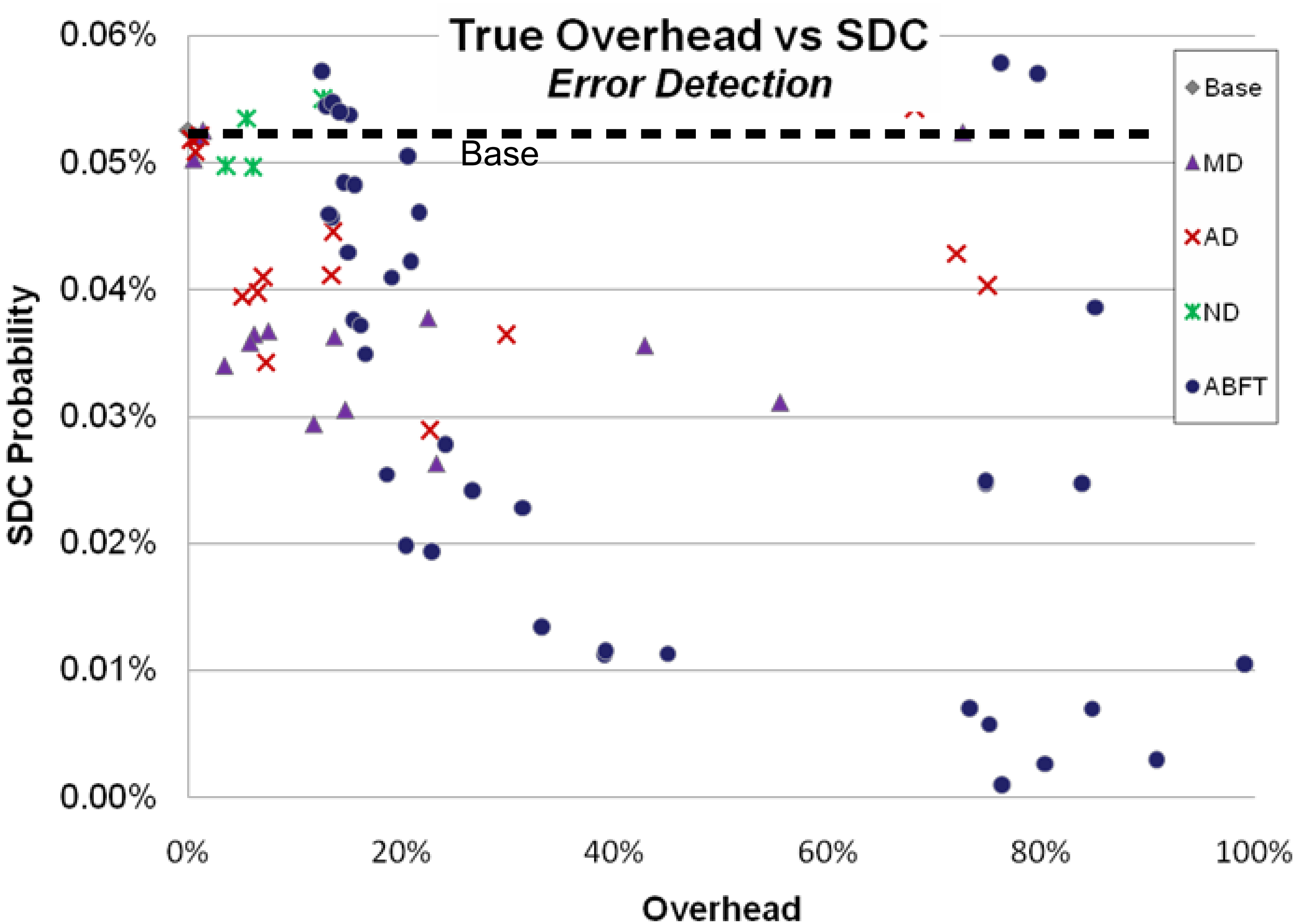
Convergence detectors reduce SDC at <20% overhead



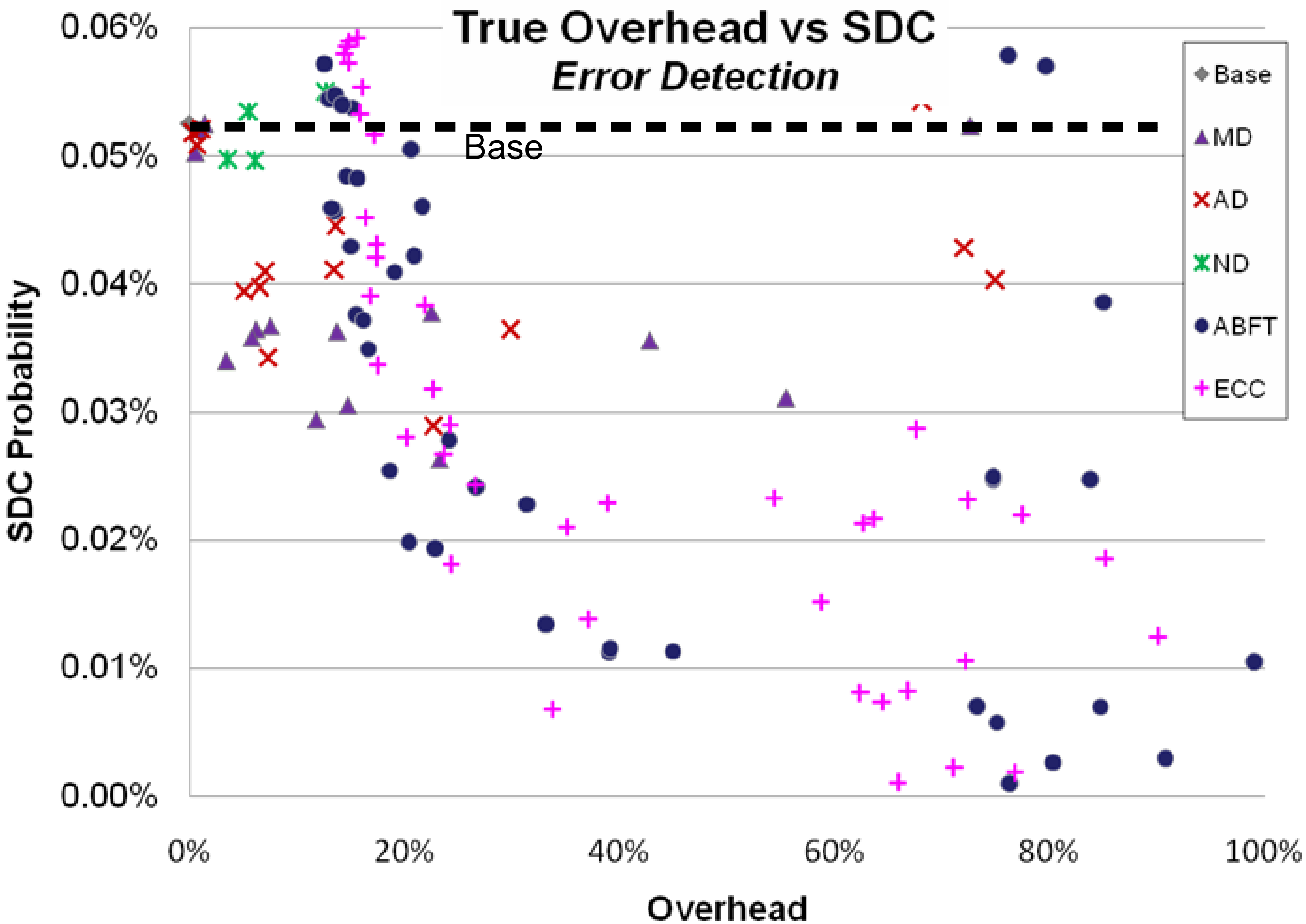
Native detectors have little effect at little cost



Encoding-based detectors significantly reduce SDC at high cost



Encoding-based detectors significantly reduce SDC at high cost

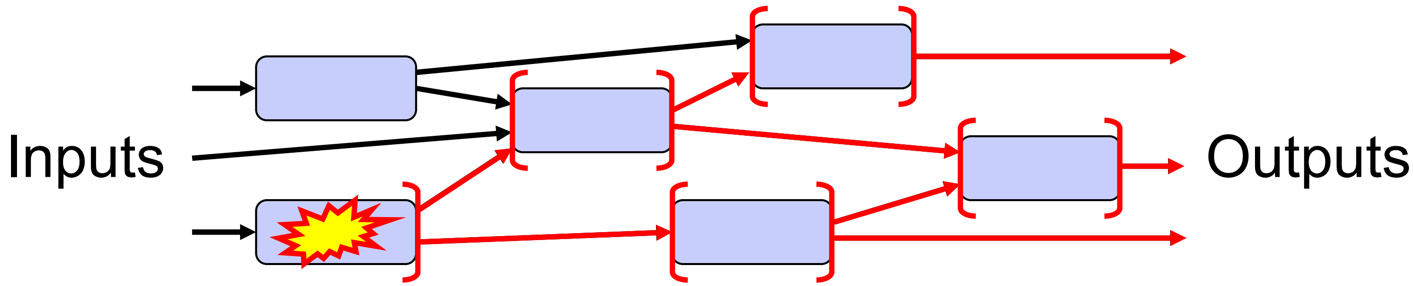


First general analysis of error vulnerability of algorithm class

- Vulnerability analysis for class of common subroutines
- Described raw error vulnerability
- Analyzed various detection/tolerance techniques
  - No clear winner, rules of thumb

Step 2: Vulnerability analysis of library-based applications

- Many applications mostly composed of calls to library routines

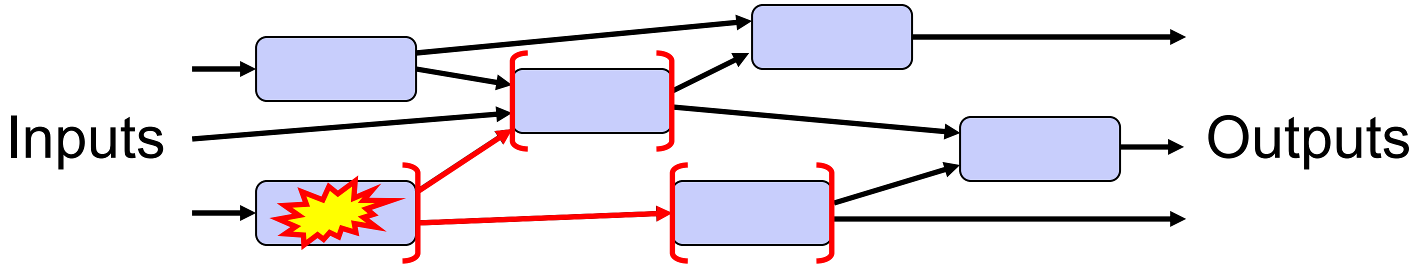


- If error hits some routine, output will be corrupted
- Later routines:
  corrupted inputs $\Rightarrow$ corrupted outputs

(Work in progress)

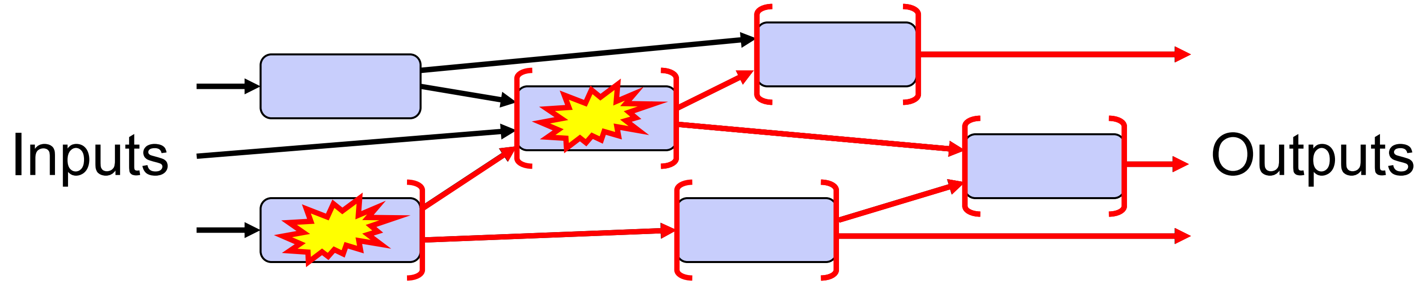
Idea: predict application vulnerability from routine profiles

- Library implementors provide vulnerability profile for each routine:
  - Error pattern in routine's output after errors
  - Function that maps input error patterns to output error patterns



Idea: predict application vulnerability from routine profiles

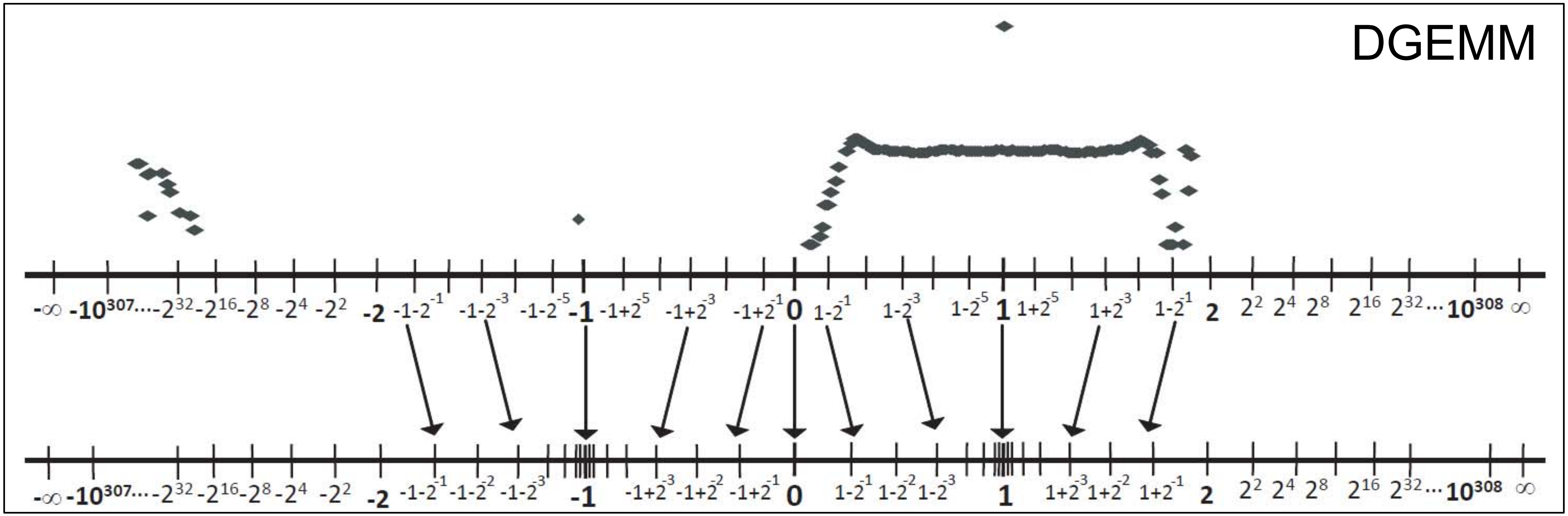
- Given application's dependence graph
  - Simulate effect of error in each routine
  - Average over all error locations to produce error pattern at outputs



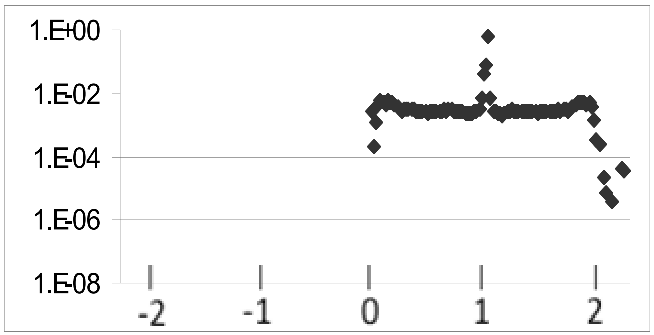
Examined applications that use BLAS and LAPACK

- 12 routines $\geq O(n^2)$, double precision real numbers
- Executed on randomly-generated nxn matrixes
  (n=62, 125, 250, 500)
- BLAS/LAPACK from Intel's Math Kernel Library on Opteron(MLK10) and Itanium2(MKL8)
  - Same results on both
- Error model: memory bit-flips

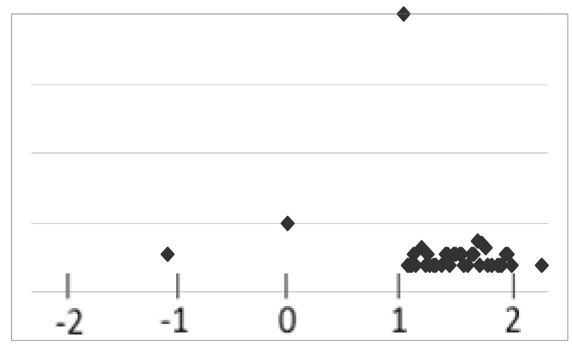
Error patterns: multiplicative error histograms



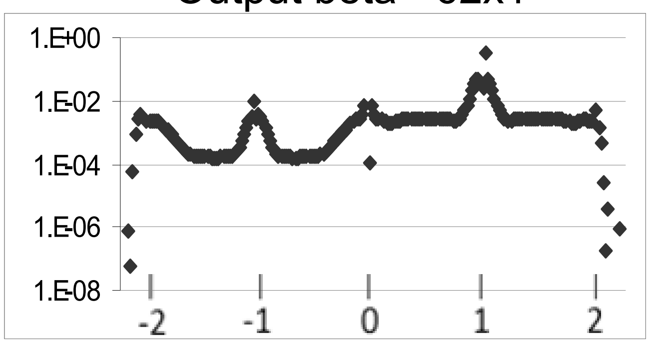
Output error patterns fall into few major categories



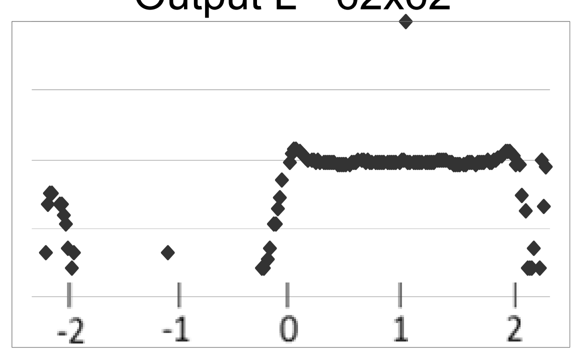
DGGES
Output beta - 62x1



DGESV
Output L - 62x62

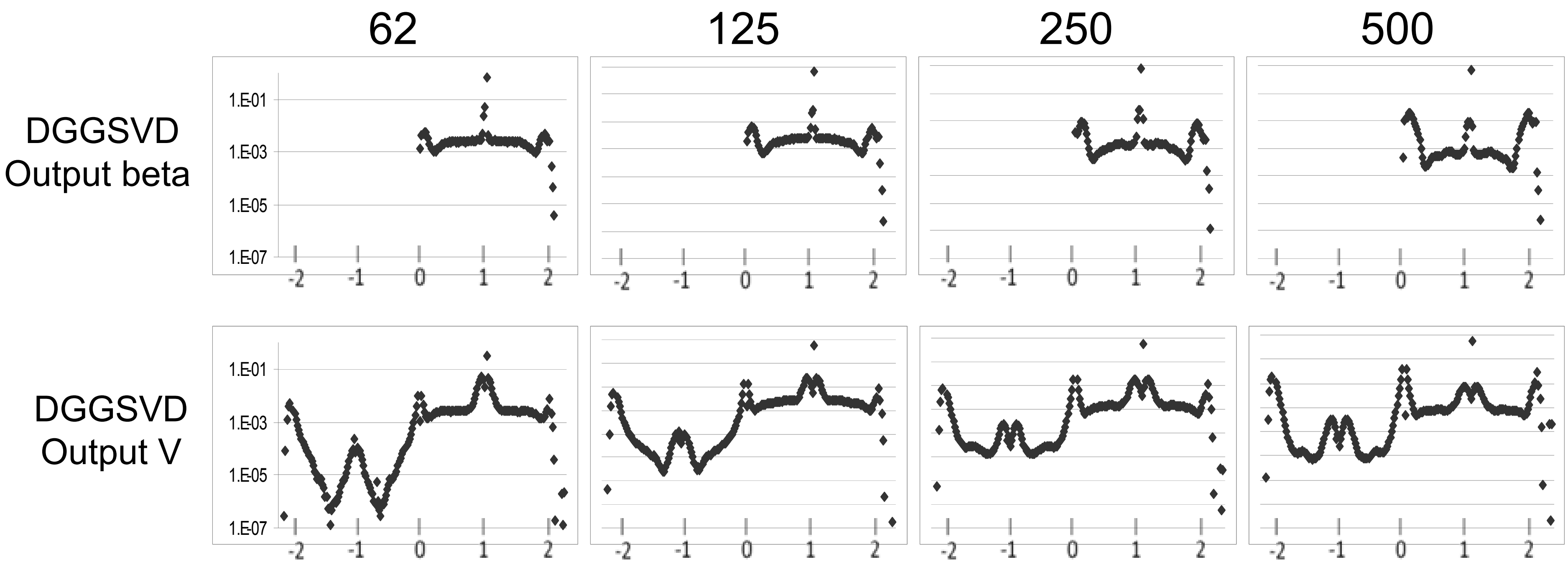


DGGES
Output vsr - 62x62



DGEMM
Output C - 62x62

Error patterns may vary with matrix size



Input-Output error transition functions

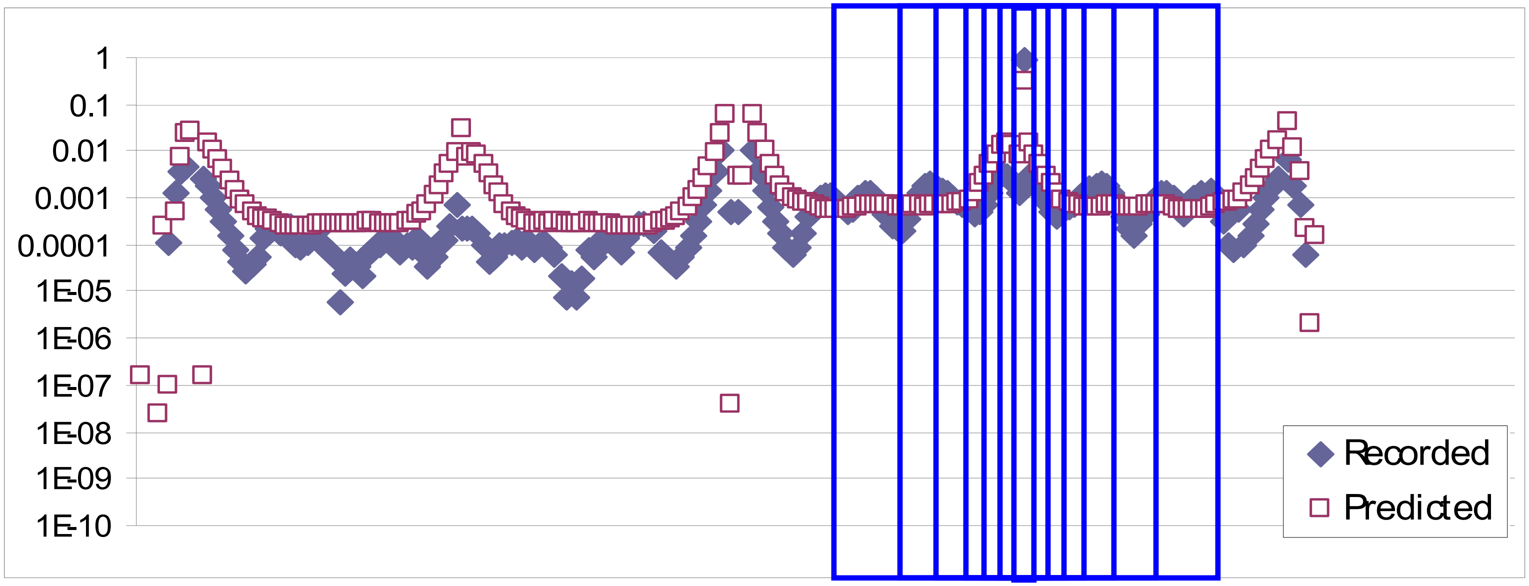
- Input-Output error transition functions: trained predictors
  - Linear Least Squares
  - Support Vector Machines

    (linear, 2nd degree polynomial, rbf kernels)
  - Artificial Neural Nets

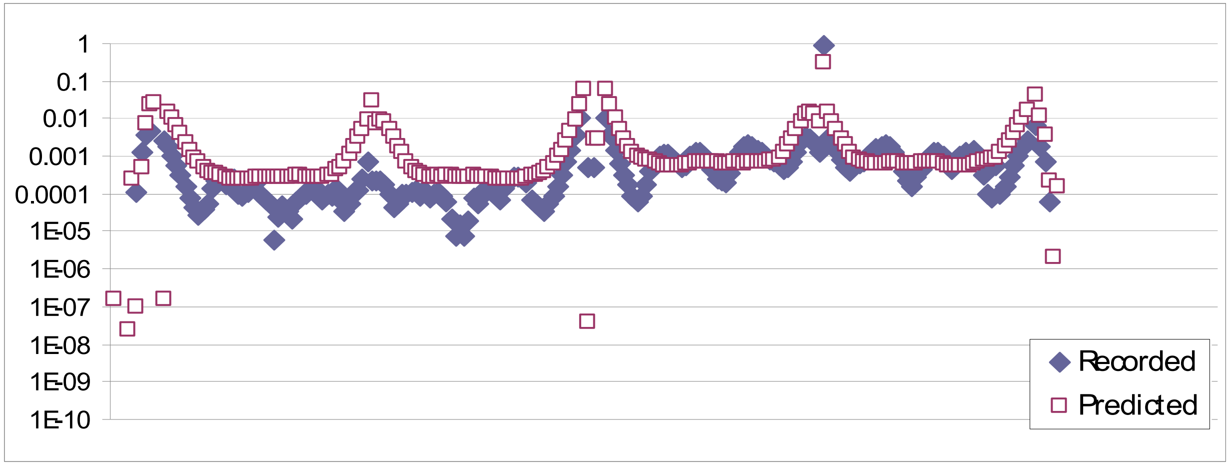
    (3,10,100 layers,; linear, gaussian, gaussian symmetric and sigmoid transfer functions)

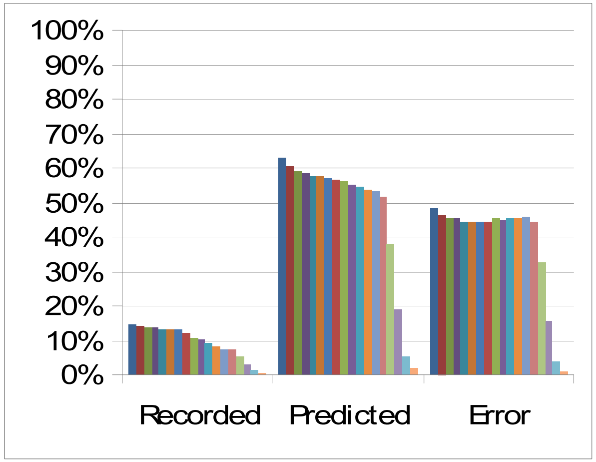
Evaluated accuracy of all predictors on all training sets

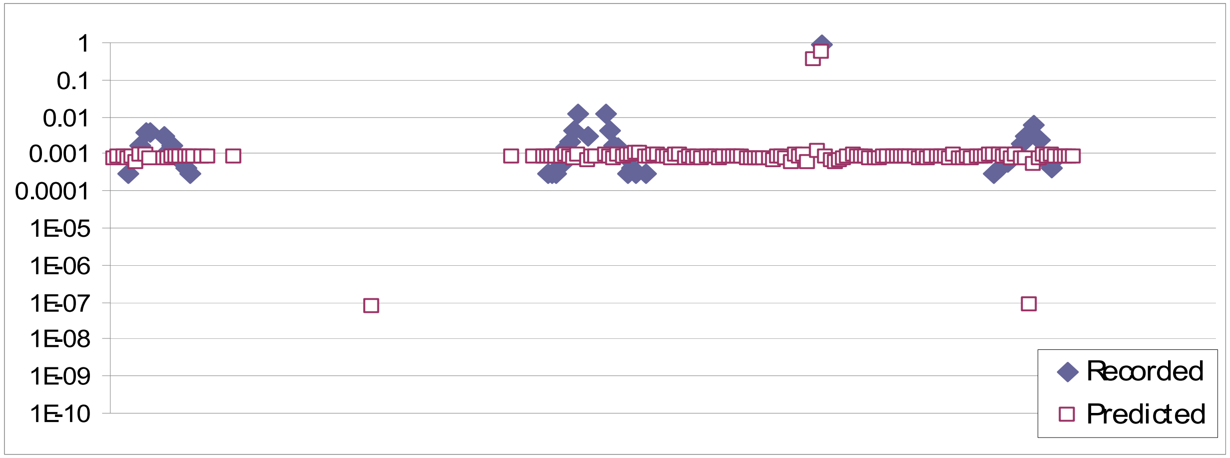
- Error metric:
  - probability of error ≥δ
  - δ∈{1e-14, 1e-13, …, 2, 10, 100)

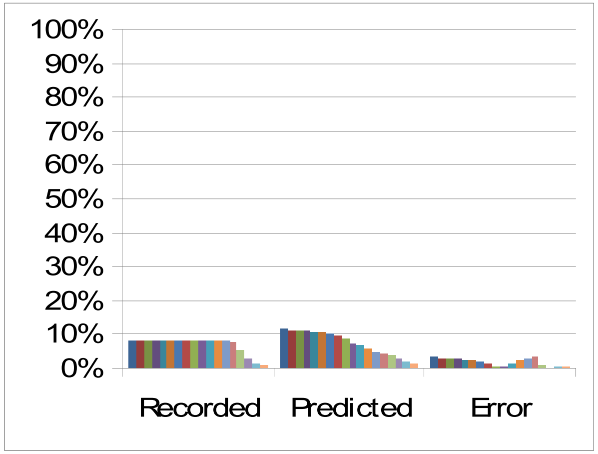


Evaluated accuracy of all predictors on all training sets

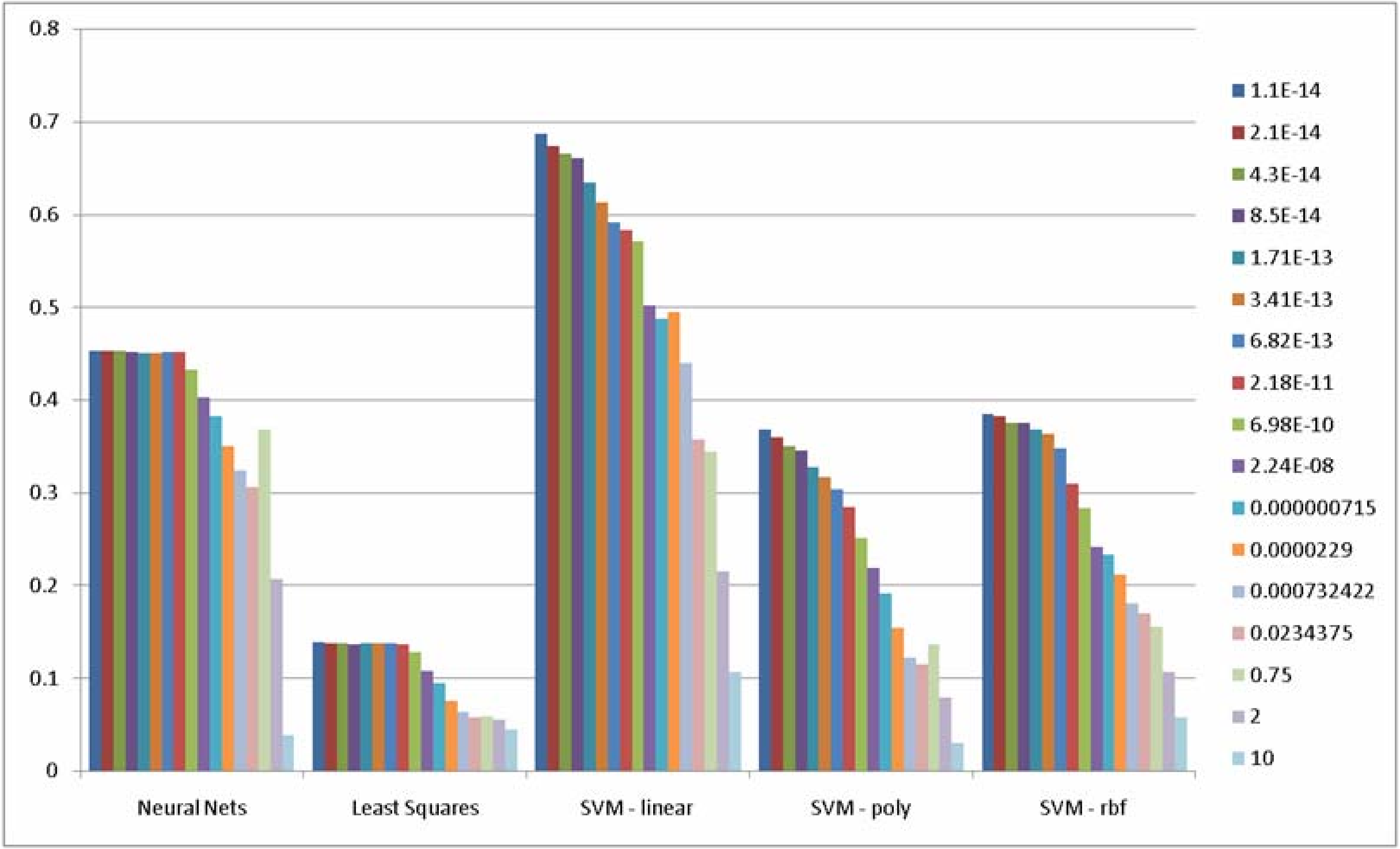




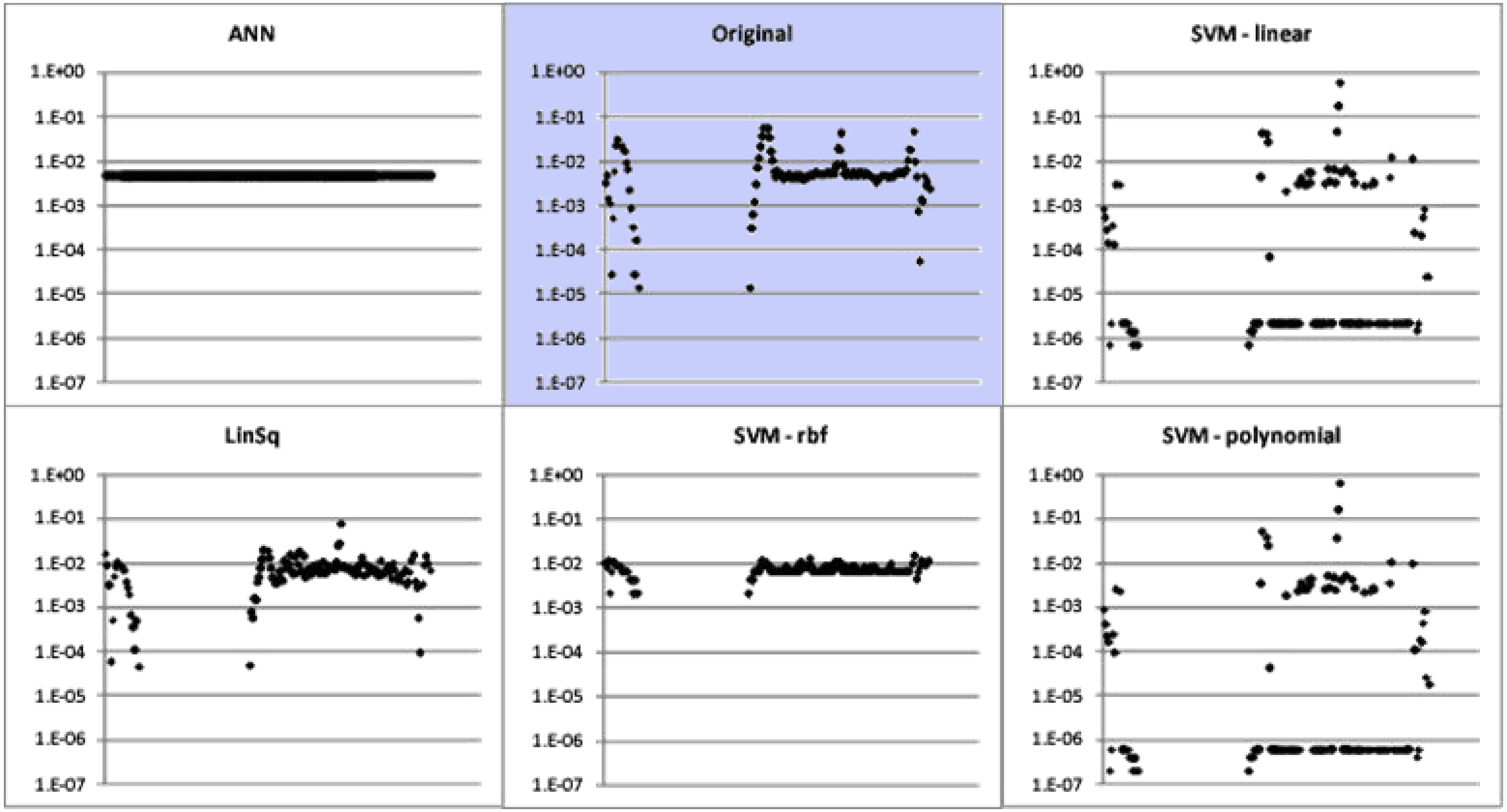




Linear Least Squares has best accuracy, Neural nets worst



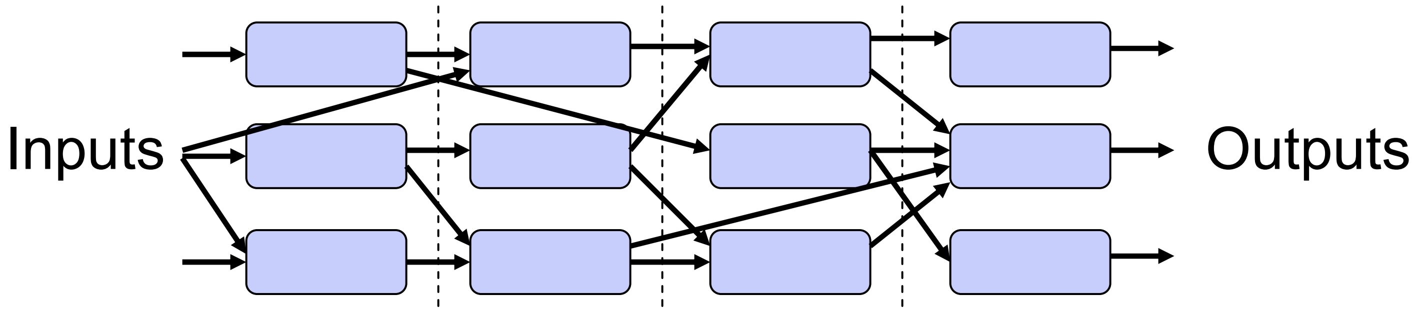
Accuracy varies among predictors



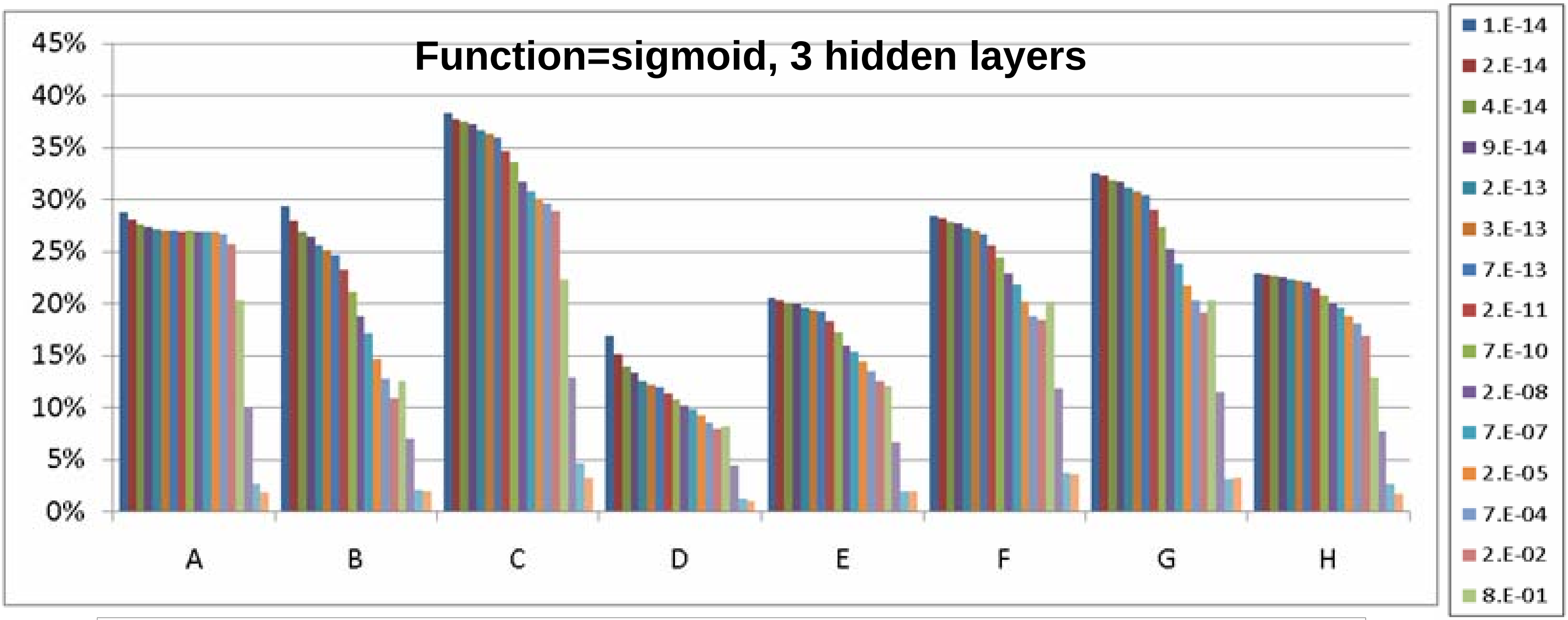
DGEES, output wr

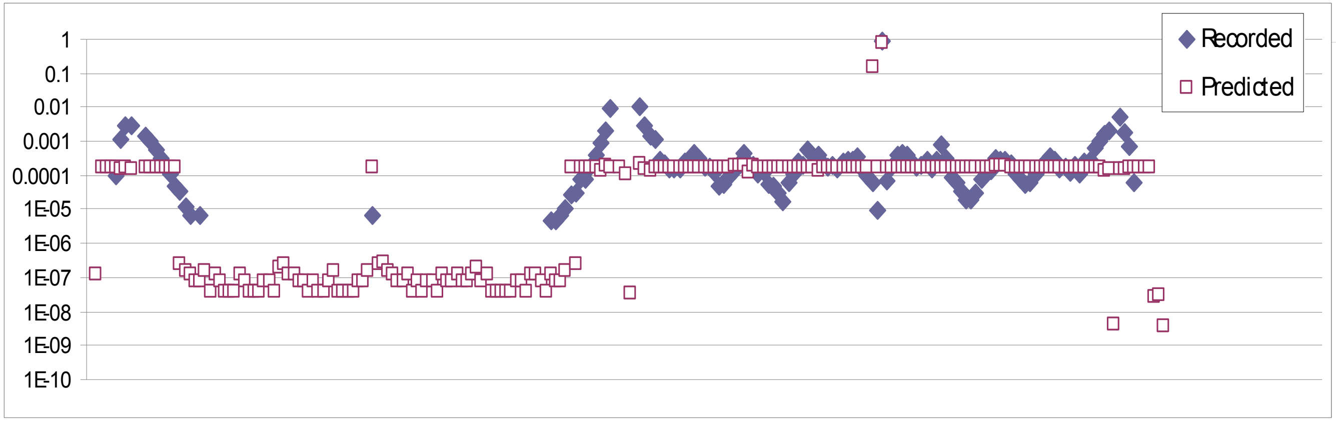
Evaluated predictors on randomly-generated applications

- Application has constant number of levels
- Constant number of operations per level
- Operations use as input data from prior level(s)

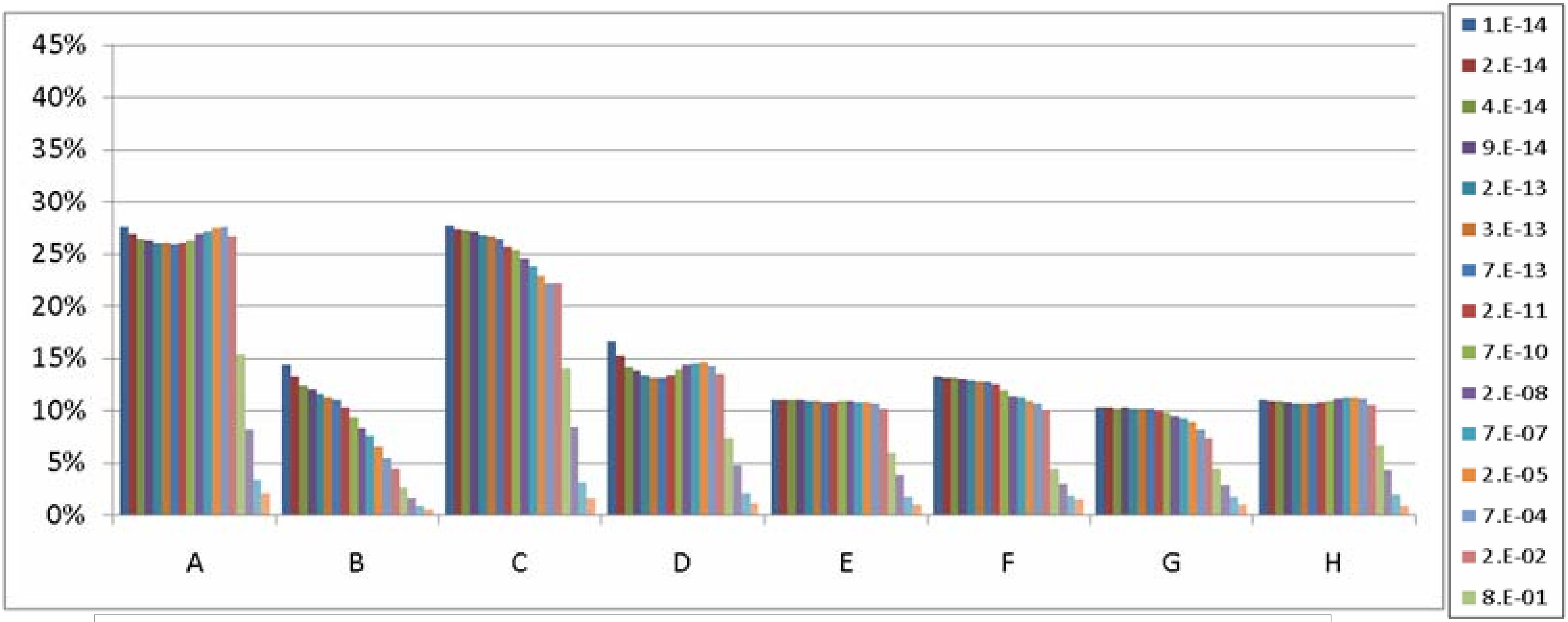


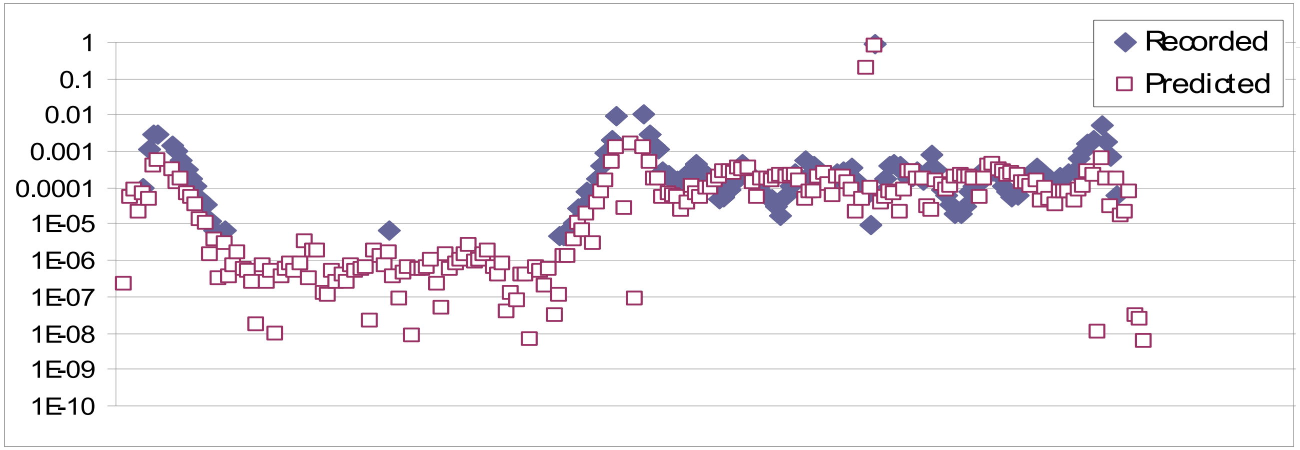
Neural Nets: Poor accuracy for application vulnerability prediction



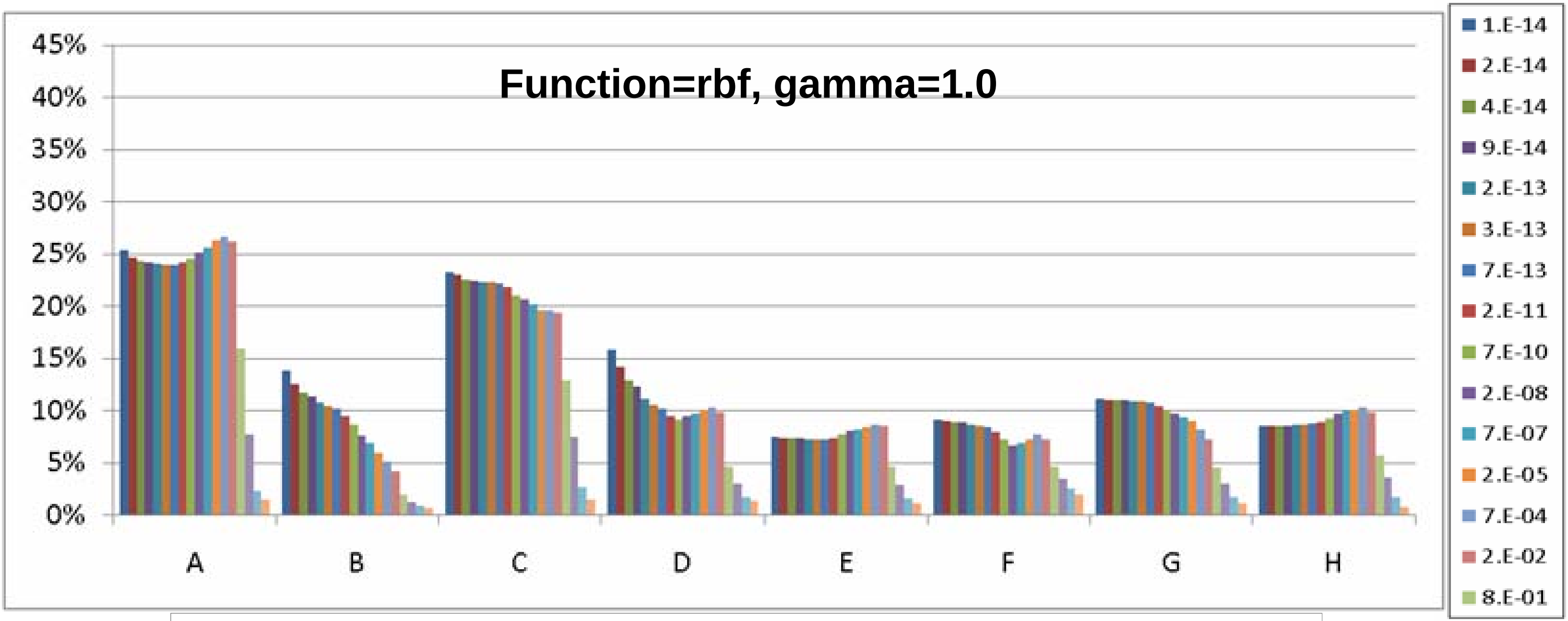


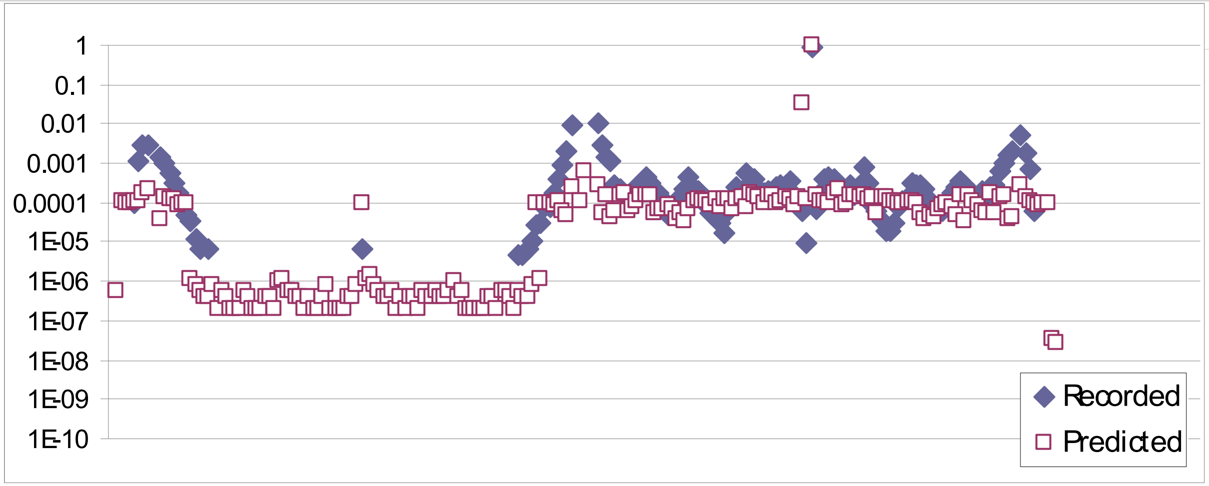
Linear Least Squares: Good accuracy, restricted





SVMs: Good accuracy, general





Work is still in progress

- Correlating accuracy of input/output predictors to accuracy of application prediction
- More detailed fault injection
- Applications with loops
- Real applications

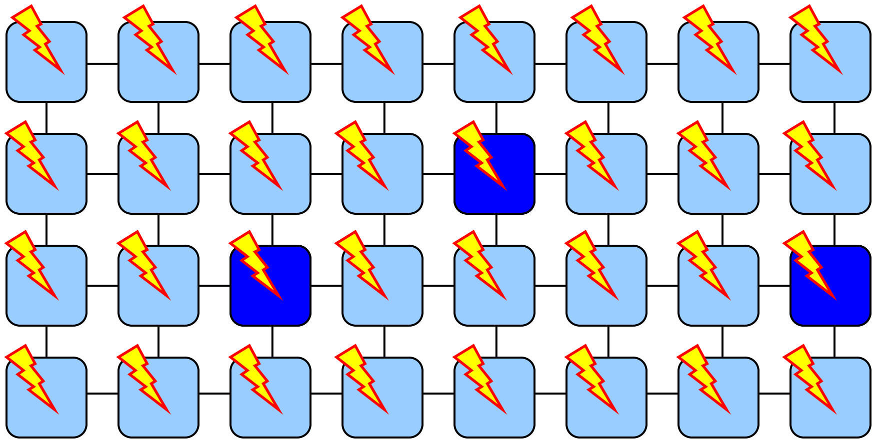
Step 3: Compiler analyses

- No need to focus on external libraries

- Can use compiler analysis to
  - Do fault injection/propagation on per-function basis
  - Propagate error profiles through more data structures (matrix, scalar, tree, etc.)

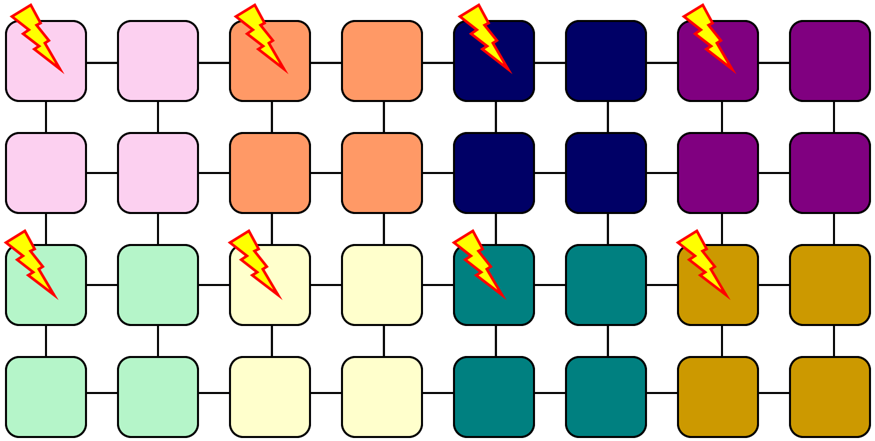
Step 4: Scalable analysis of parallel applications

- Cannot do fault injection on 1,000-process application
- Can modularize fault injection
  – Inject into individual processes



Step 4: Scalable analysis of parallel applications

- Cannot do fault injection on 1,000-process application
- Can modularize fault injection
  - Inject into single-process runs
  - Propagate through small-scale runs



Working toward understanding application vulnerability to errors

- Soft errors becoming increasing problem on HPC systems
- Must understand how applications react to soft errors
- Traditional approaches inefficient for realistic applications
- Developing tools to cheaply understand vulnerability of real scientific applications