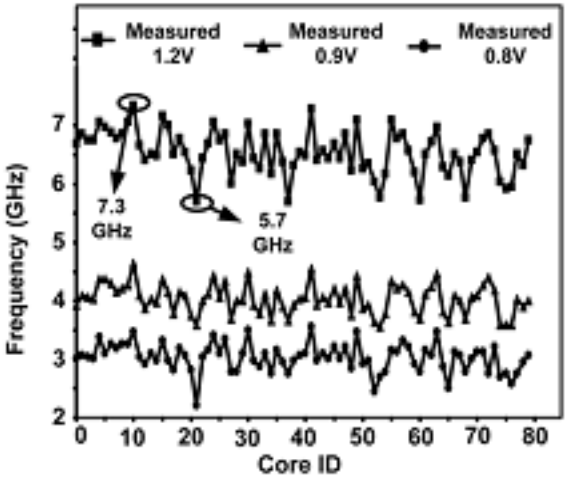
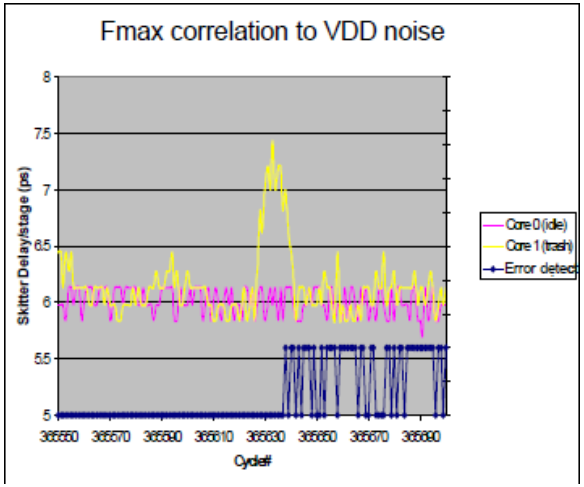
Computing with Stochastic Processors: Embracing Errors in Architecture and Design of Processors and Applications

Asstnt Prof Rakesh Kumar

Department of Electrical and Computer Engineering

University of Illinois, Urbana-Champaign





L (nm)	250	180	130	90	65	45
Vt (mV)	450	400	330	300	280	200
σ-Vt (mV)	21	23	27	28	30	32
σ-Vt/Vt	4.7%	5.8%	8.2%	9.3%	10.7%	16%

Non-determinism ↑

Yield (for a given frequency/power target) ↓

Energy Cost (for a given yield) ↑

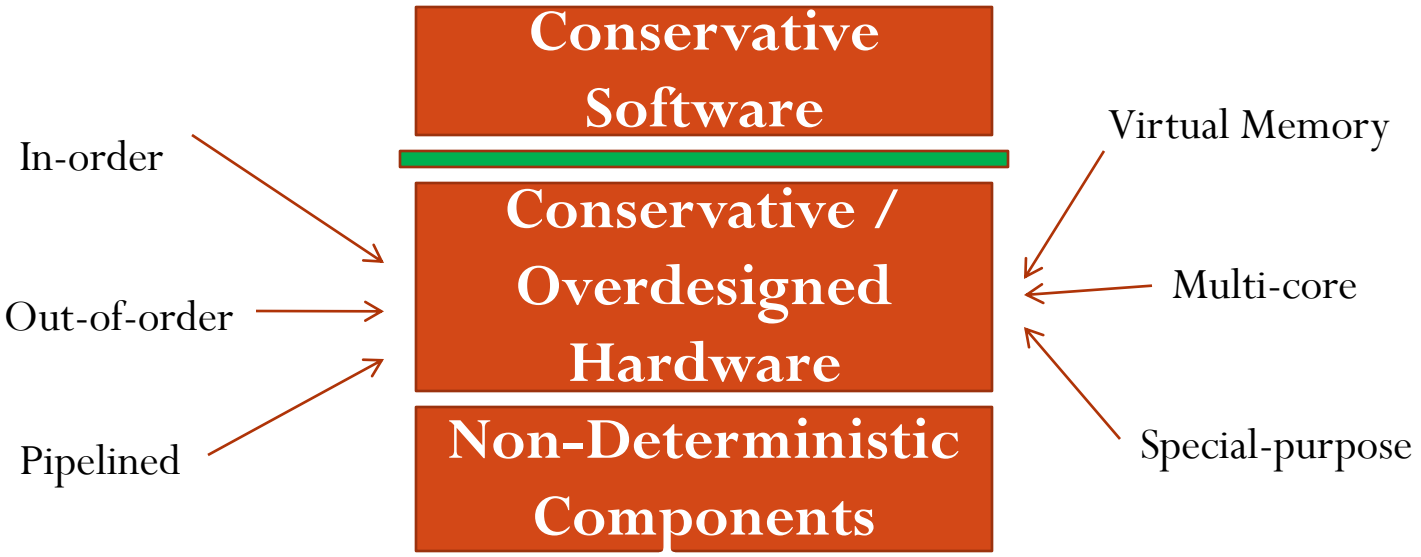
Status quo cannot continue: Need to find a solution to the non-determinism problem to limit power

A "buck-a-billion" in 2020; micro-watt computing devices; costs of non-determinism unacceptable for emerging applications

Non-determinism is NOT the Problem, It is How Computer System Designers Treat it

Software expects hardware to behave flawlessly in spite of non-determinism

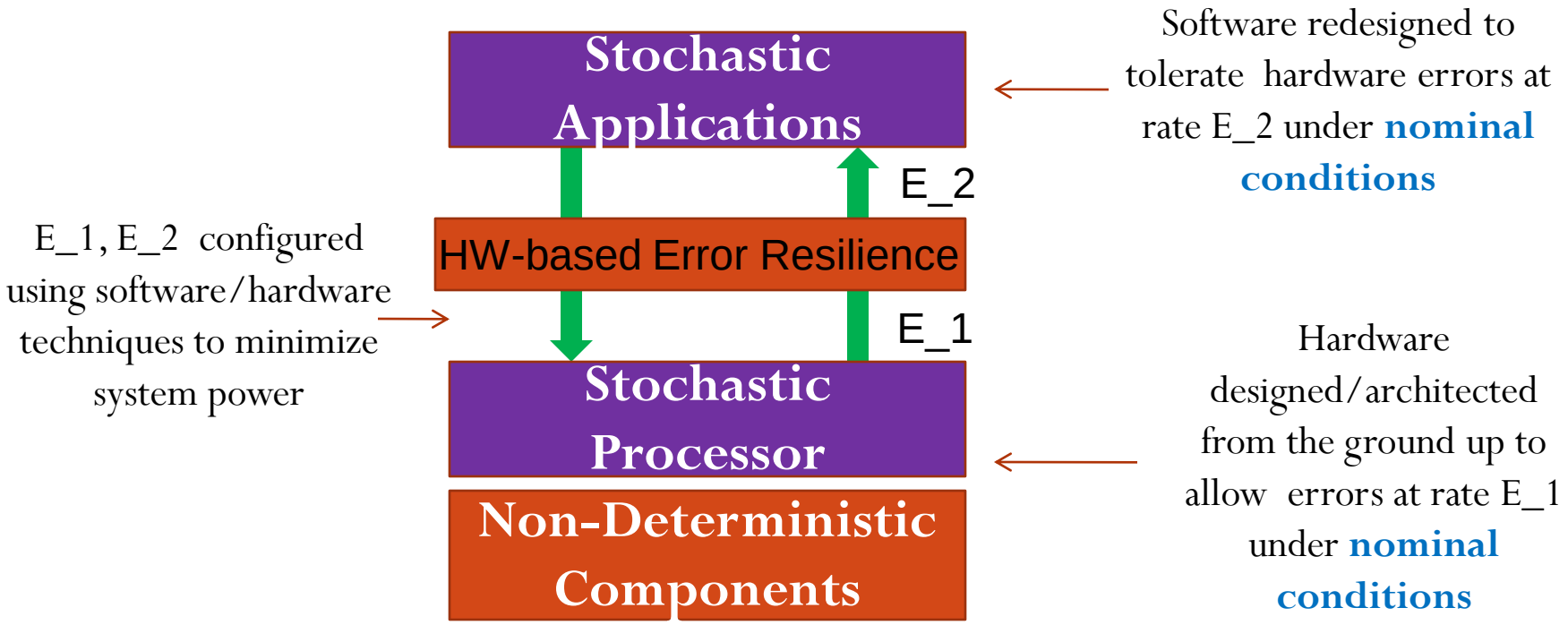
Hardware ALWAYS designed for correctness (under worst-case or nominal conditions); an attempt to meet the conventional software mindset at all cost



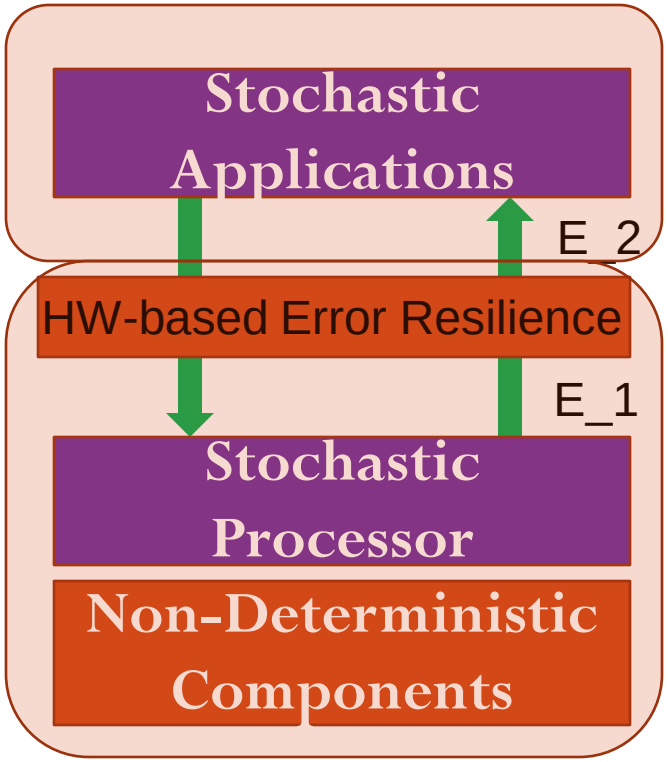
There would be no energy cost of non-determinism if non-determinism can be exposed directly to the software under nominal conditions as opposed to eliminating it or hiding it.

Research Vision: Computing with Stochastic Processors

- Rethink the Correctness Contract between Hardware and Software

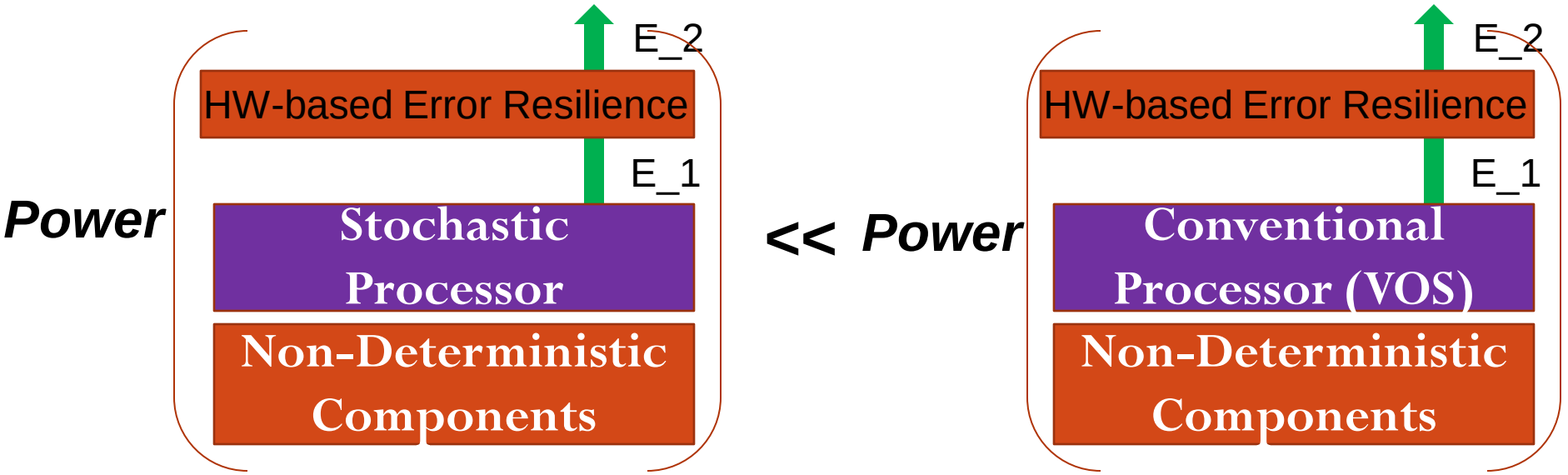


The goal of our research is to explore approaches to architect and design stochastic processors and applications.



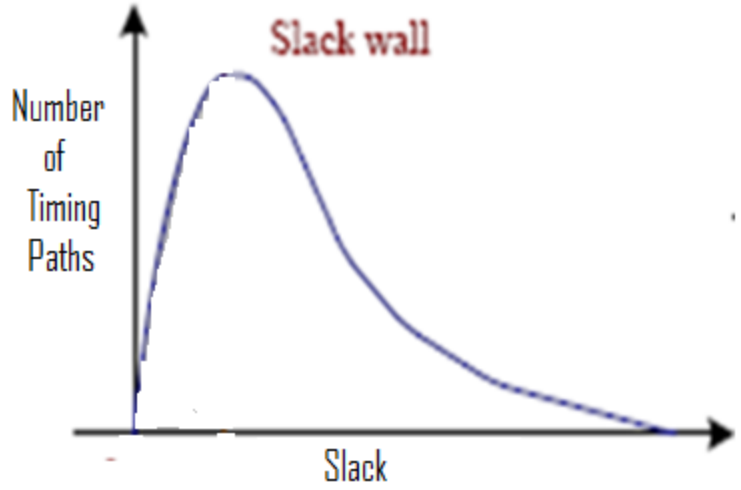
E_1 = Timing Error Rate
=Fraction of cycles during which at least one latch/FF latches an incorrect value
=Fraction of cycles during which the output of at least one timing path that is super-critical (i.e., has negative slack) toggles due to a change in the input

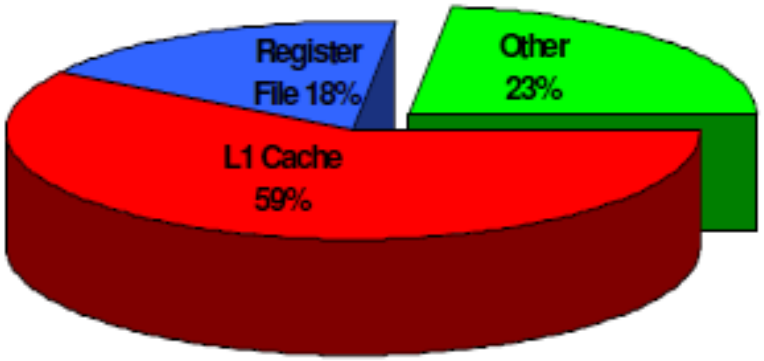
Hardware Design Goal



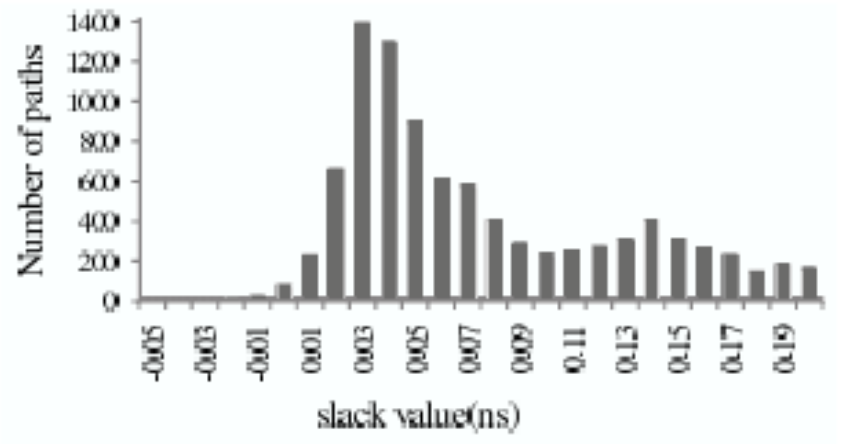
Using Conventional Processors for Non-zero Error Rate Operation

- Conventional processors have a timing slack wall





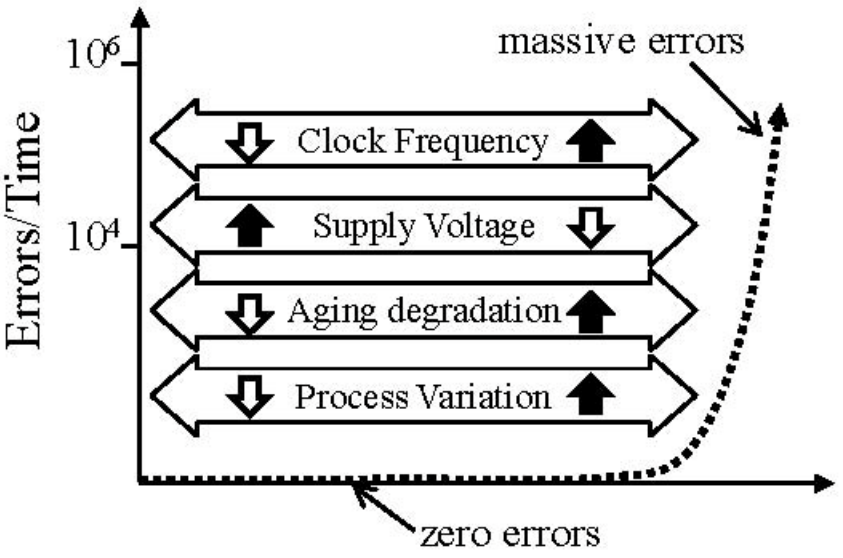
Courtesy: Northwestern



Using Conventional Processors for Non-zero Error Rate Operation

- Conventional processors have slack wall

	Error Rate (%)					
Module	1.0V	0.9V	0.8V	0.7V	0.6V	0.5V
lsu_dctl	0.00	0.23	8.60	29.46	45.13	54.90
lsu_qctl1	0.00	5.94	10.85	16.99	16.56	37.53
lsu_stb_ctl	0.00	0.08	0.65	5.19	11.79	22.38
sparc_exu_div	0.00	0.15	0.23	0.35	0.49	1.10
sparc_exu_ecl	0.00	3.31	10.97	87.08	88.93	73.03
sparc_ifu_dec	0.00	0.08	0.87	7.09	15.22	20.48
sparc_ifu_errdp	0.00	0.00	0.00	0.00	0.00	9.21
sparc_ifu_fcl	0.00	10.56	22.25	50.04	55.06	56.95
spu_ctl	0.00	0.00	0.00	1.30	2.96	35.53
tlu_mmu_ctl	0.00	0.01	0.02	0.06	0.14	0.19

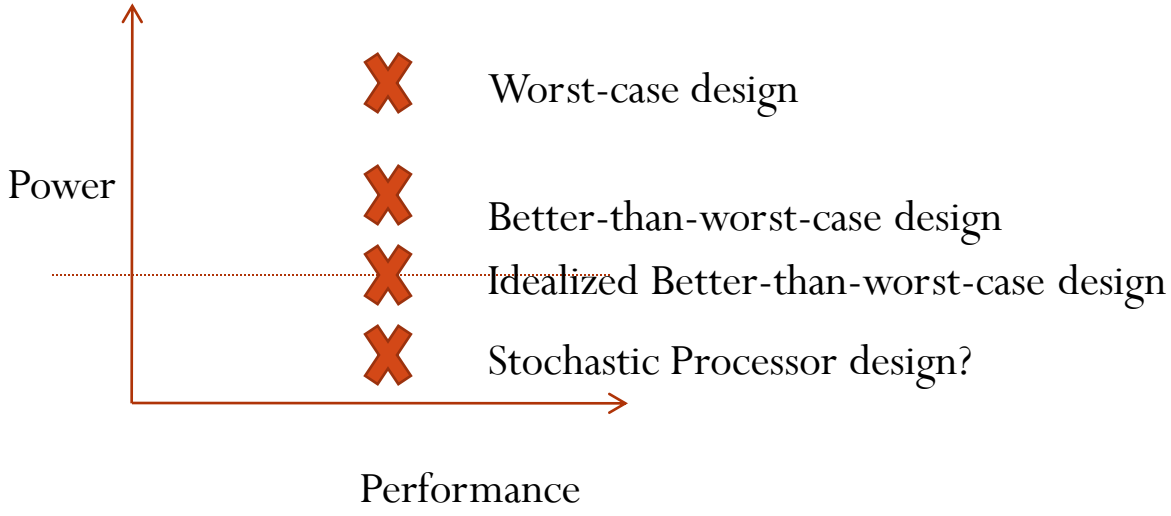


(HPCA2010, ASPDAC2010)

Too many errors produced if non-determinism is exposed (using VOS, for example), not much scaling possible before E_1 reached

Using Conventional Processors for Non-zero Error Rate Operation

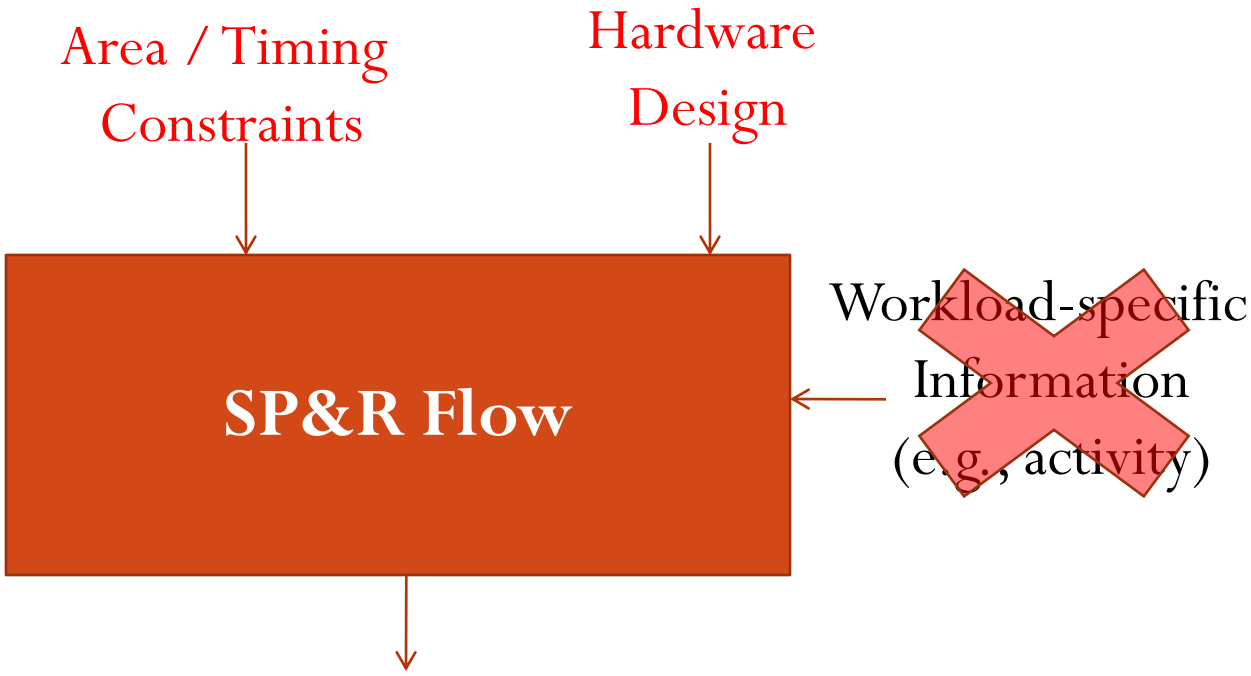
- Conventional processor have slack wall
- Conventional processors have inherently higher power/area as they are optimized for correct operation



Power difference between a conventional processor and a stochastic processor will become more pronounced as leakage increases

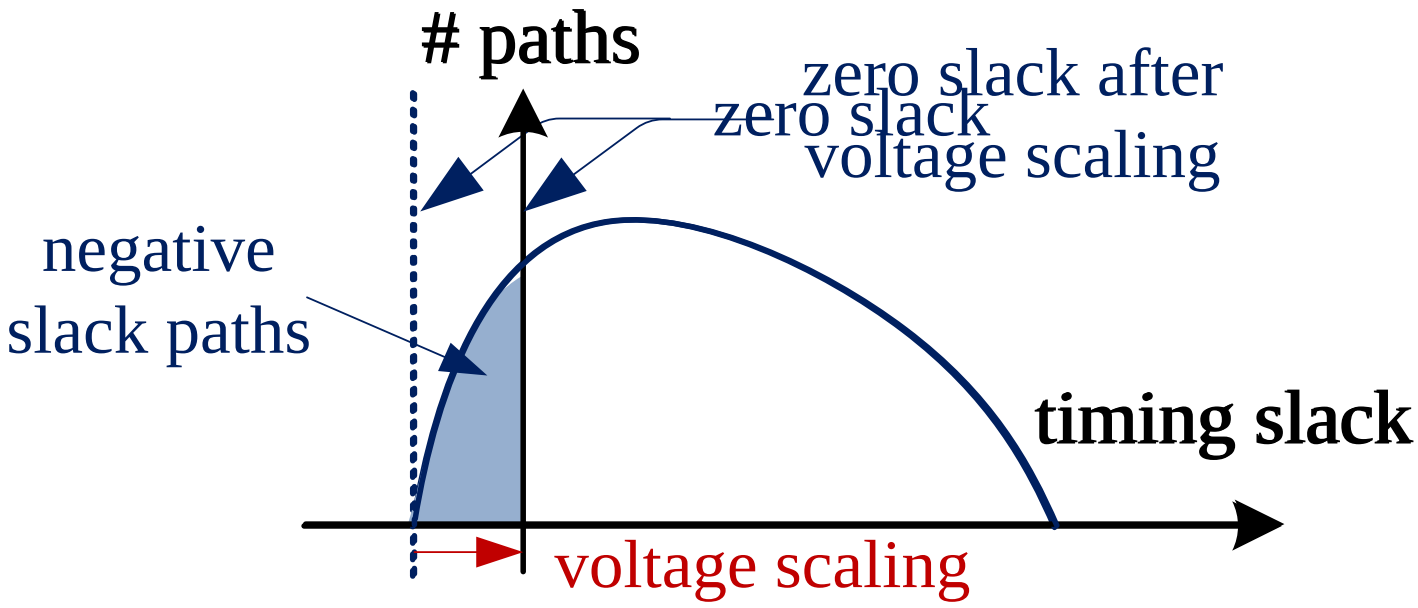
Using Conventional Processors for Non-zero Error Rate Operation

- Conventional processor have slack wall
- Conventional processors have inherently higher power/area as they are optimized for correct operation
- Conventional processors are workload-agnostic (STA, SSTA); therefore, heavily overdesigned for most workloads (false paths, etc)



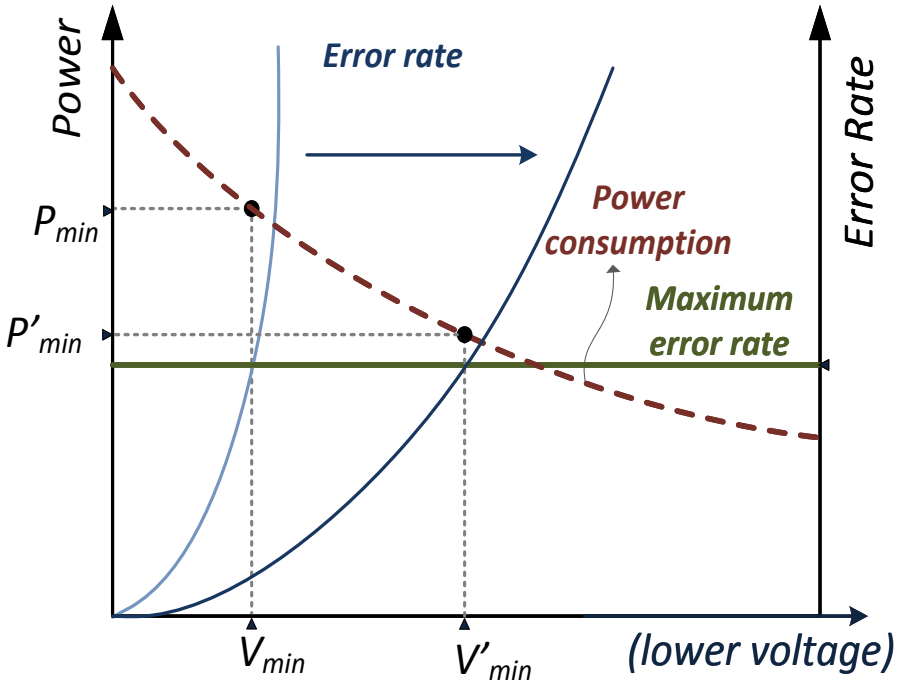
Design and Architecture of Stochastic Processors: Key Observation

- Error rate depends on path **slack** and **activity**
- Slack distribution determines which paths cause errors; activity determines the error rate contribution of the paths



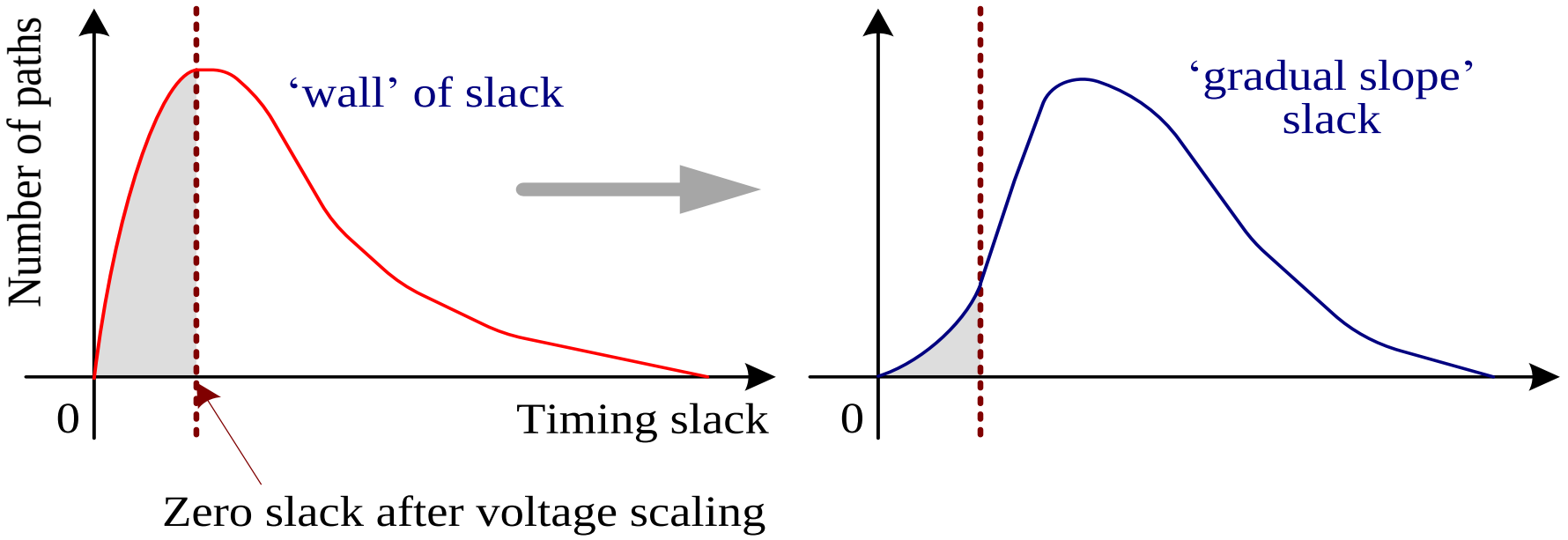
Design, Architecture, and Compilation Approaches to Manipulate Slack and Activity Distribution to Minimize system power when an Error Resilience Mechanism is Available

Soft Processor Design for Voltage Overscaling



(HPCA2010, ASPDAC2010, DAC2010)

Soft Processor Design: Power-aware Slack Redistribution for Stochastic Processors



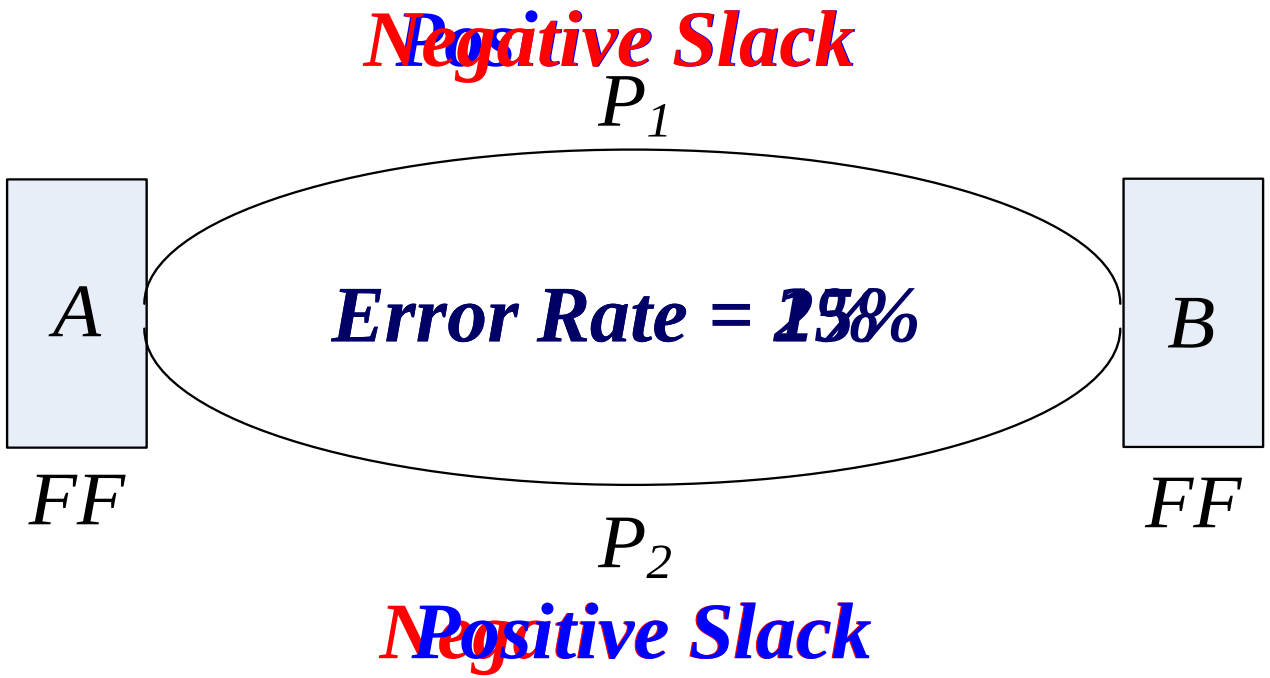
Insight: Frequently-exercised paths contribute most errors, but the number of such paths in a design is small

- Optimize frequently exercised paths.
- De-optimize rarely exercised paths.

Increase slack of (near) critical paths without consuming much extra power.

Both gradual failure and low power can be achieved.

Slack Re-distribution Example

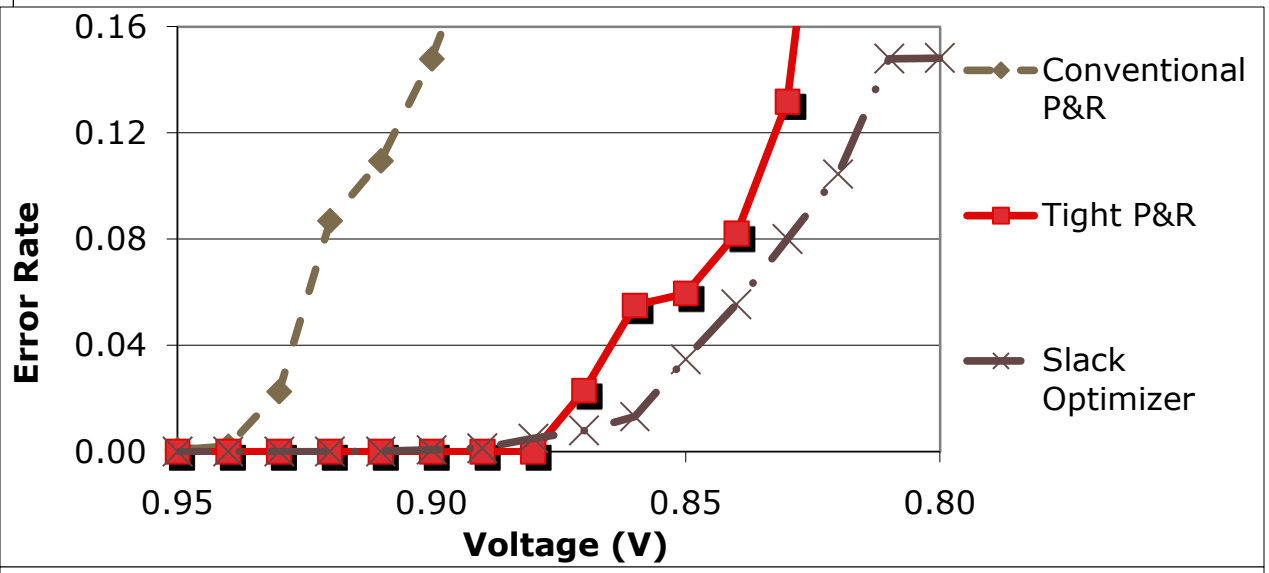


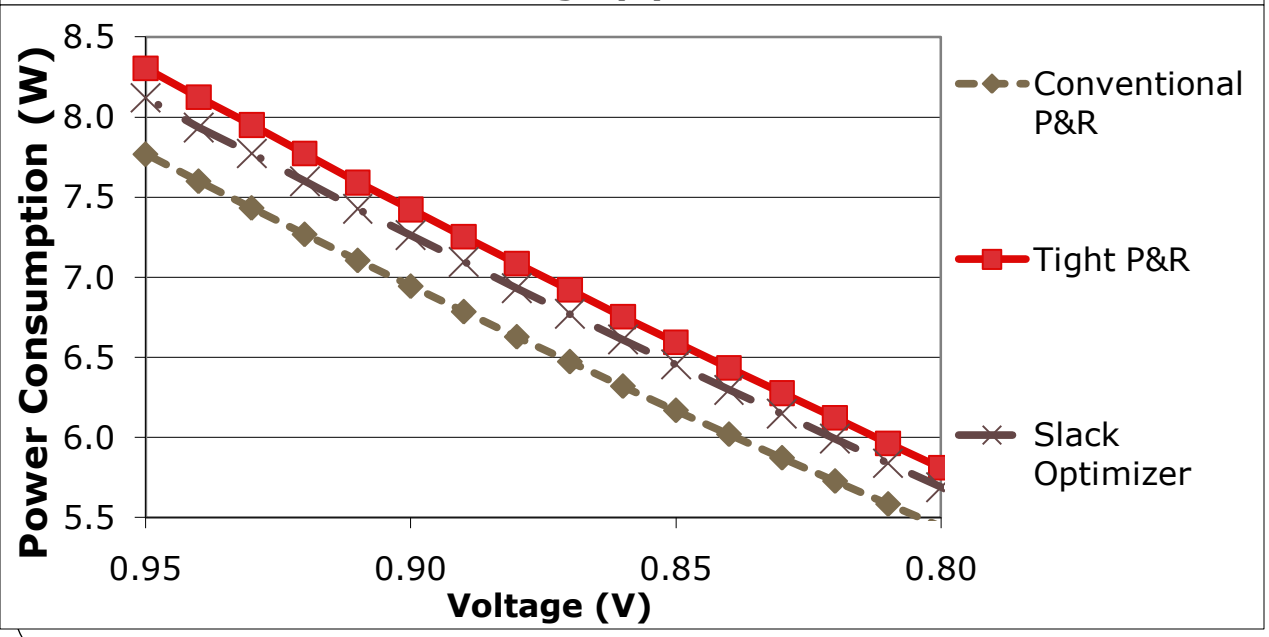
$TG(P_1) = 0.25$ $Slack(P_1) = -0.2$ **0.0**

$TG(P_2) = 0.01$ $Slack(P_2) = 0.1$ **-0.1**

Much smaller error rate at a given voltage => much smaller voltage for a given error rate

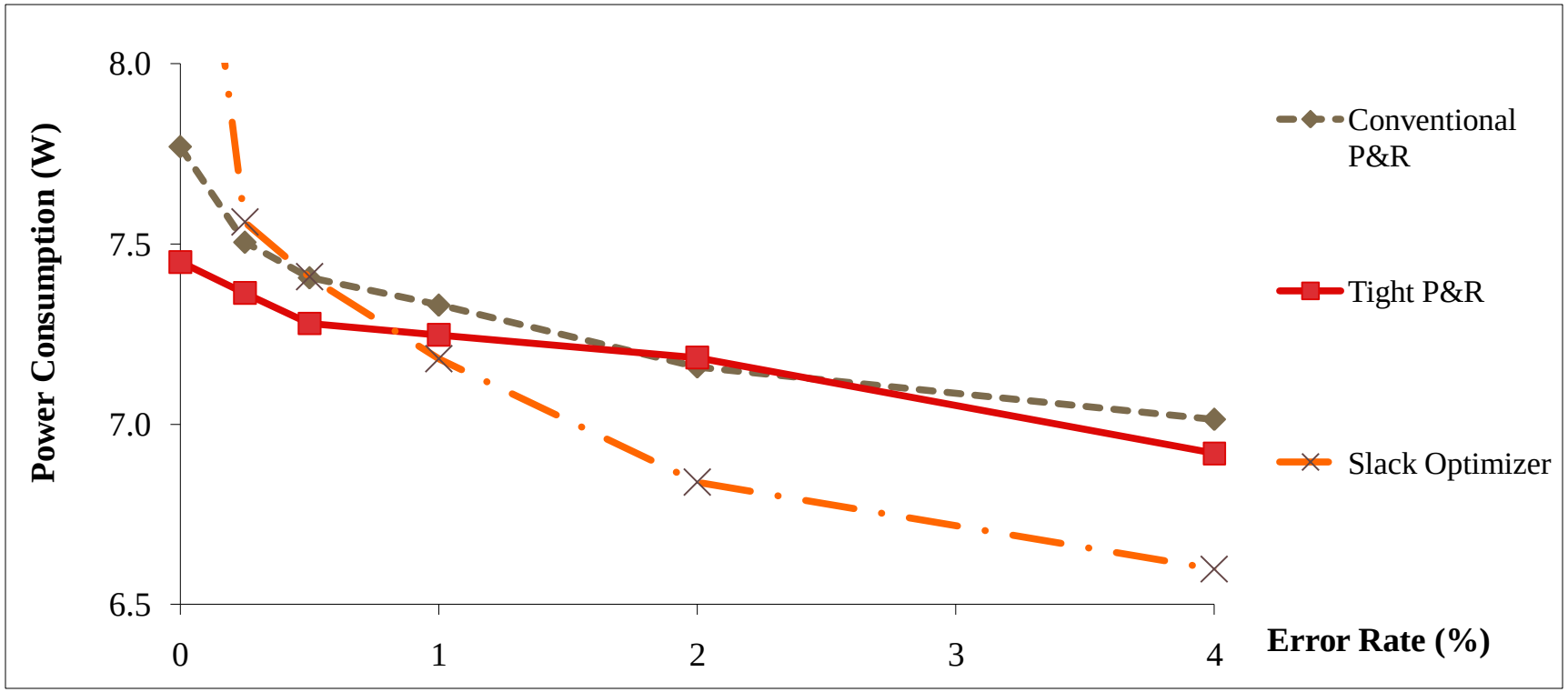
Processor Error Rate and Power





Slack Distributions indeed causes graceful degradation in reliability; Designs with comparable error rates have much higher power/area overheads.

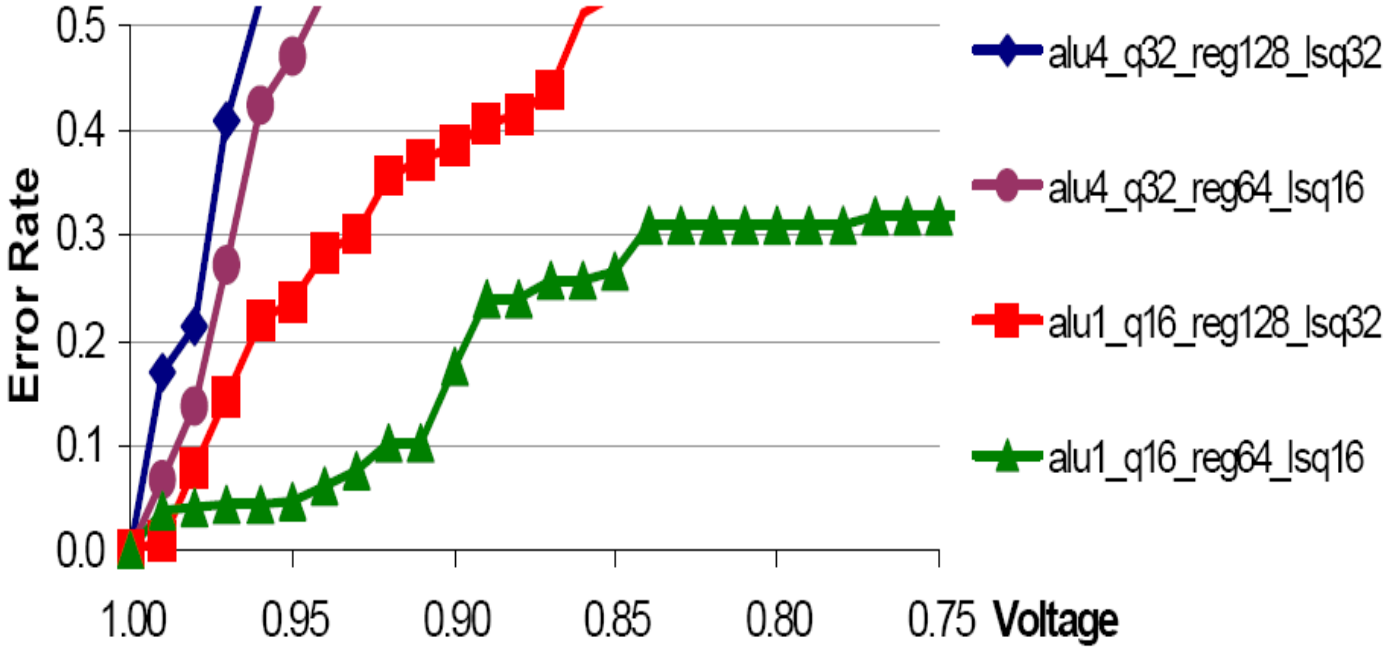
Reliability/Power Tradeoff



Slack-optimized design enjoys continued power reduction as error rate increases; first methodology that produces designs that allow voltage/reliability tradeoffs.

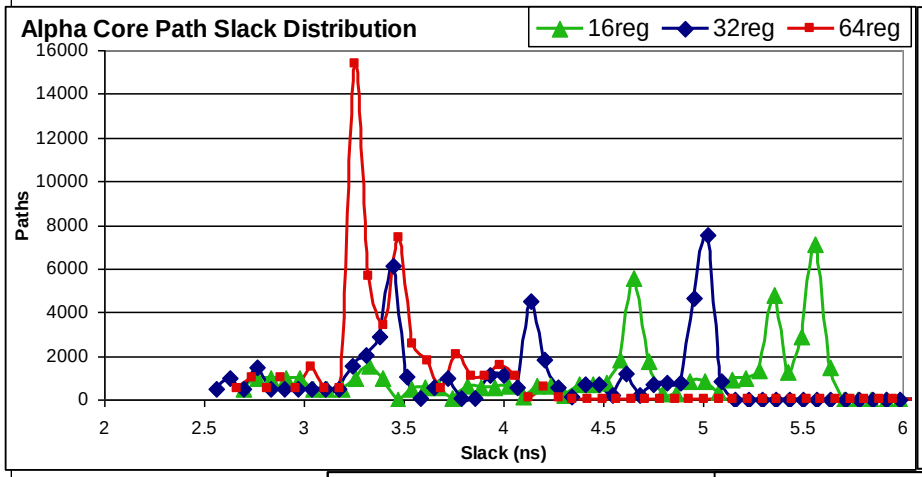
Error Rate Sensitivity to Architecture

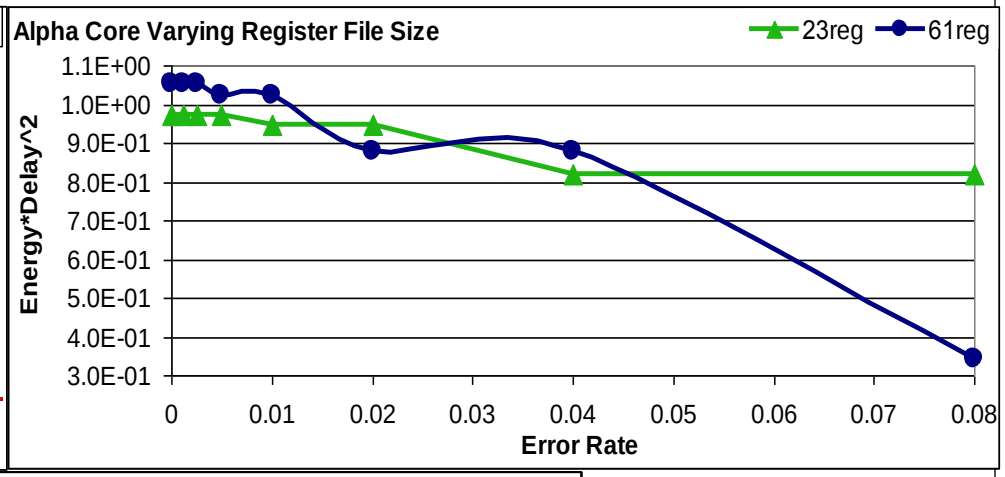
- Changes to microarchitecture affect slack and activity distribution of processor
- Error rate behavior can change significantly with change to architecture

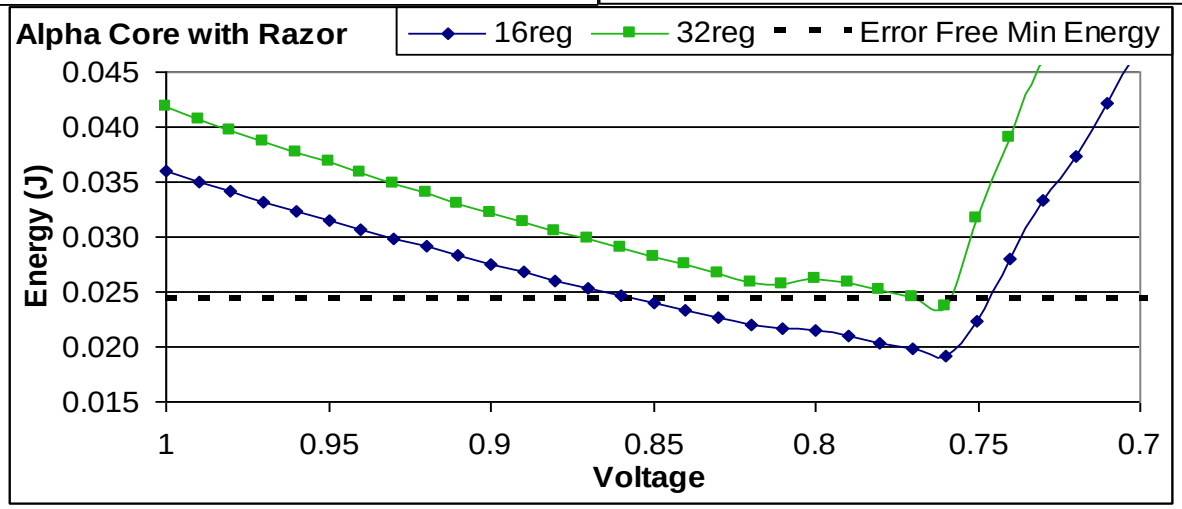


(DATE 2010; DAC 2010; CASES2011)

Reshaping the Slack Distribution: Register File Resizing

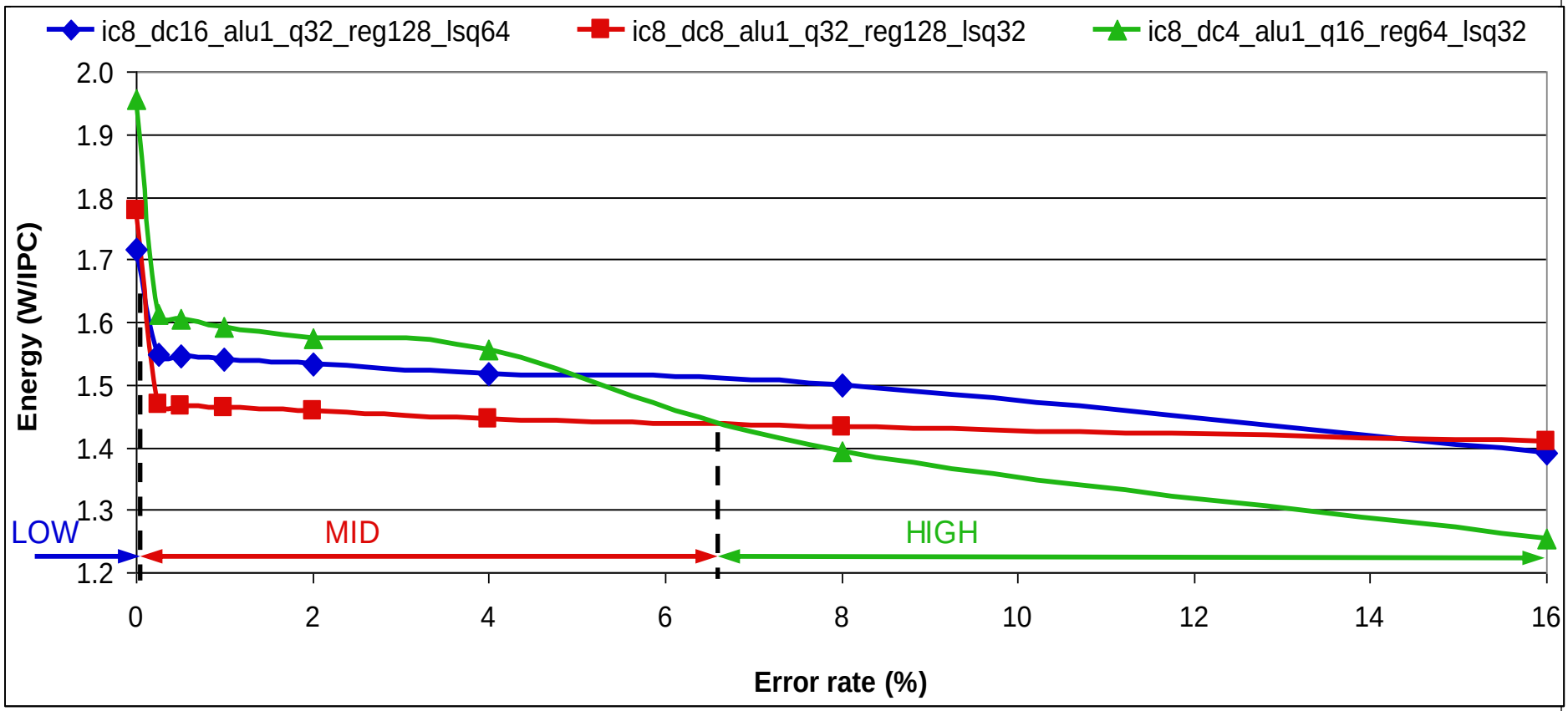






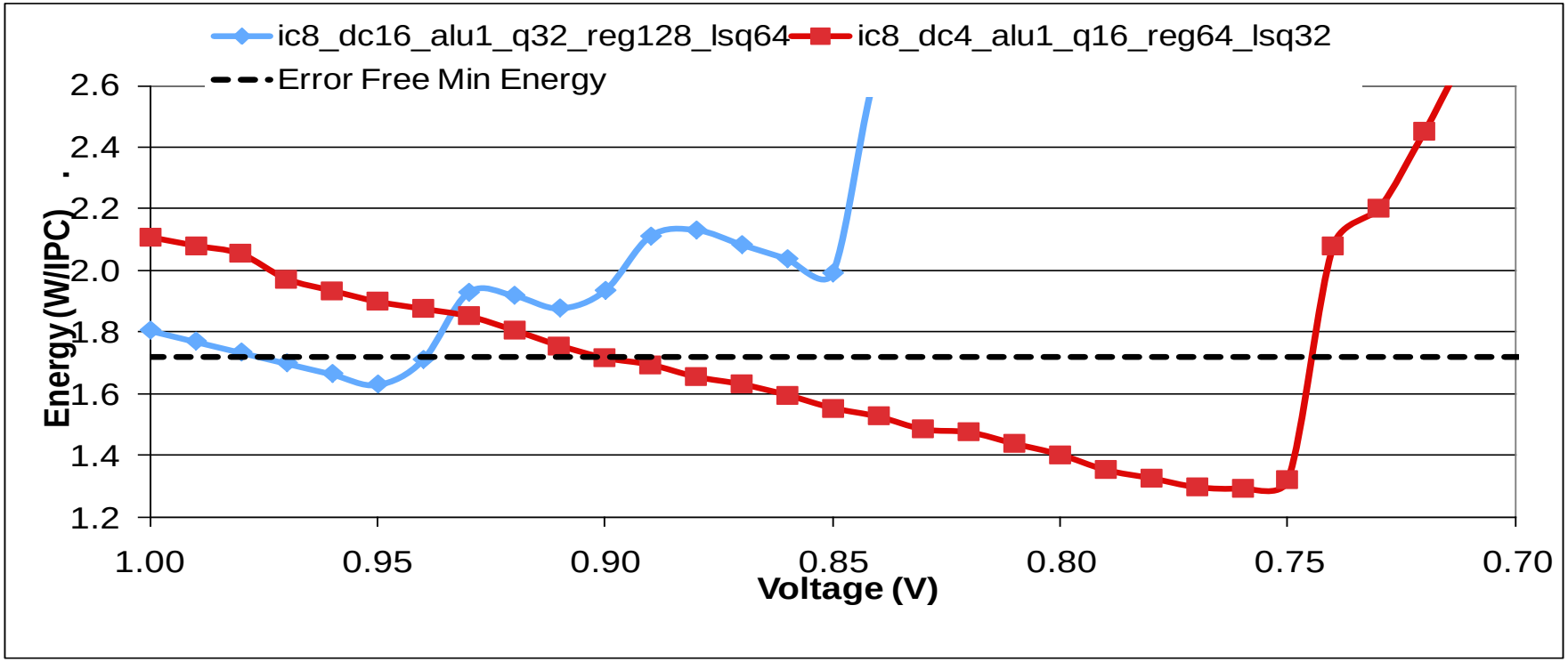
A smaller register file reduces regularity; increases efficiency (>21%) at non-zero error rates; With a large register file energy saving <2%

Design Space Exploration

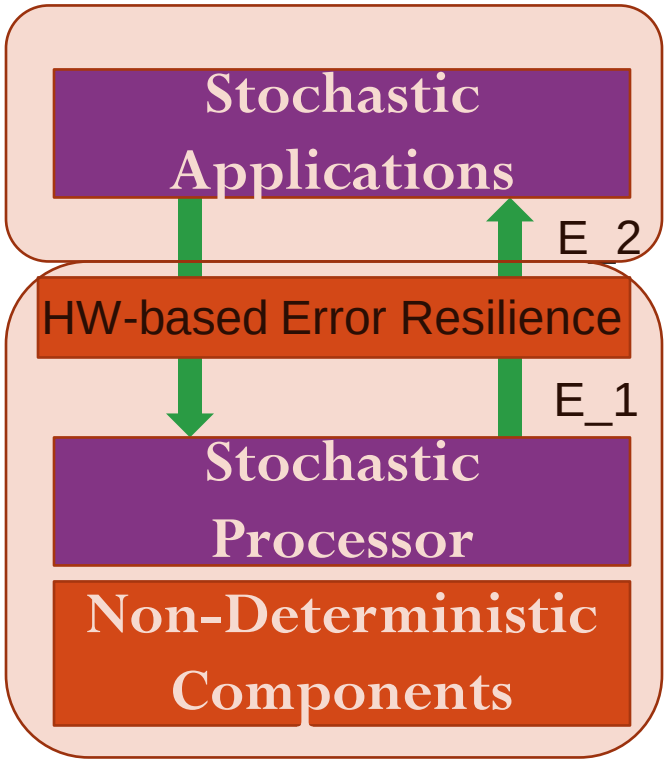


As error rate increases, smaller regular structures and less complex logic become more efficient

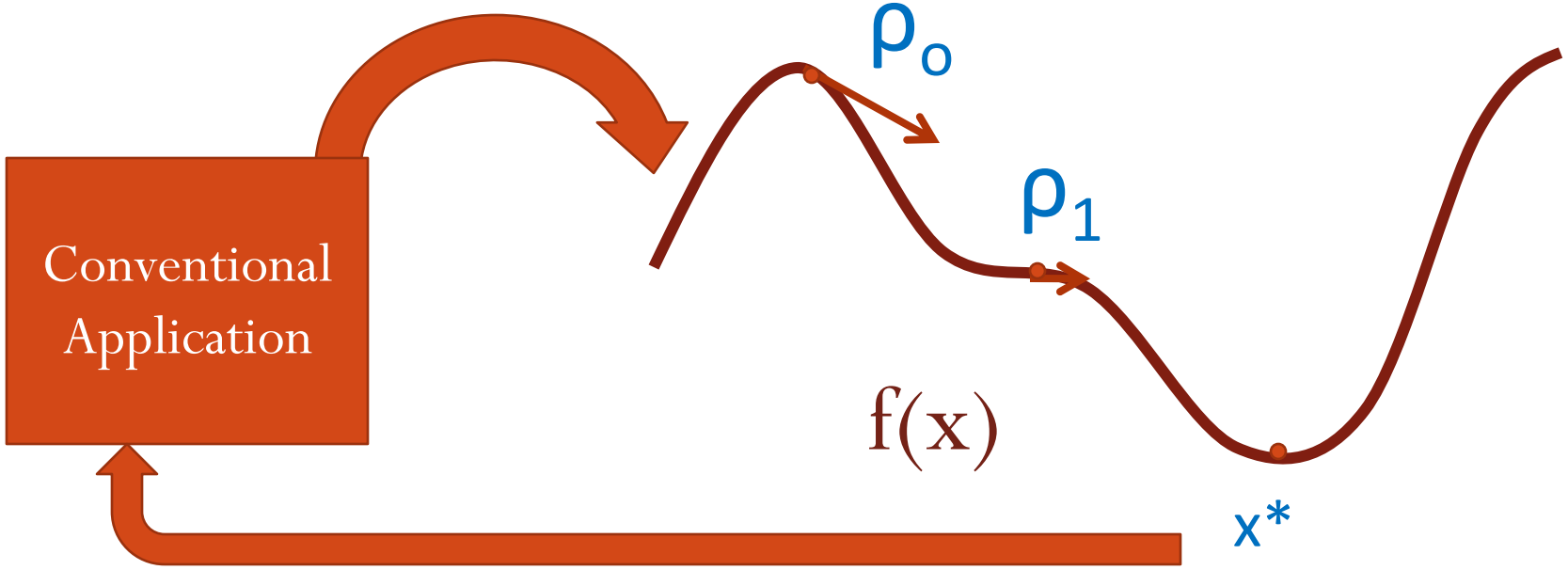
Design Space Exploration



A resilience-optimized architecture achieves significantly higher (>25%) efficiency than correctness-optimized architecture


Stochastic Applications
E_2
HW-based Error Resilience
E_1
Stochastic Processor
Non-Deterministic Components

An Optimization-based Approach to Application Robustification



- **Primary Issues**
  - **How to construct f(x) when we don't know x*?**
  - **What is the most efficient solver for f(x)?**

Example Formulation: Sorting

- input: u is unsorted list of n elements
- What is sorting?
  - Finding the correct relative position of each element in the unsorted list. [*Permutation matrix*]
  - Example
    - $u = [5,2,8]^T$
    - X: 3x3 Permutation Matrix

Permutation (X) to reverse

$$Xu = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \\ 8 \end{bmatrix} = \begin{bmatrix} 8 \\ 2 \\ 5 \end{bmatrix}$$

Permutation (X) to sort

$$Xu = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \\ 8 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix}$$

Example : Robustified Sorting

- Post condition:
  - output list contains elements of u sorted in ascending order.
- Variational expression:
  - Xu ,where
    - X: is permutation matrix of size n x n
- Problem Formulation
  - The list which arranges the elements of list in ascending order will minimize the product -v(Xu)

$$v = [1....n]^T$$

$$\min_{x \in R^{nxn}} -v^T Xu$$

Example : Robustified Sorting

- As unconstrained problem:

$$\min_{x \in R^{n x n}} -v^T X u$$

$$s.t.\ \ X_{ij} \geq 0, \sum_i X_{ij} \leq 1, \ \ \sum_j X_{ij} \leq 1$$

$$-v^T X u + \lambda \sum_{ij} [X_{ij}]_+^2 + \lambda \sum_i [\sum_j X_{ij} - 1]_+^2 + \lambda \sum_j [\sum_i X_{ij} - 1]_+^2$$

Penalty Function

Scope of Transformations

- SOE : $\min \ \| Ax - b \|^2 \approx x^T A^T Ax - 2b^T Ax$

  (as Quadratic Program)

- SORT: $\min_{x \in R^{nxn}} -v^T Xu$

  (as Linear Program)

- GM: $\min_{x \in R^{nxn}} - < W, X >$

  (as Linear Program)

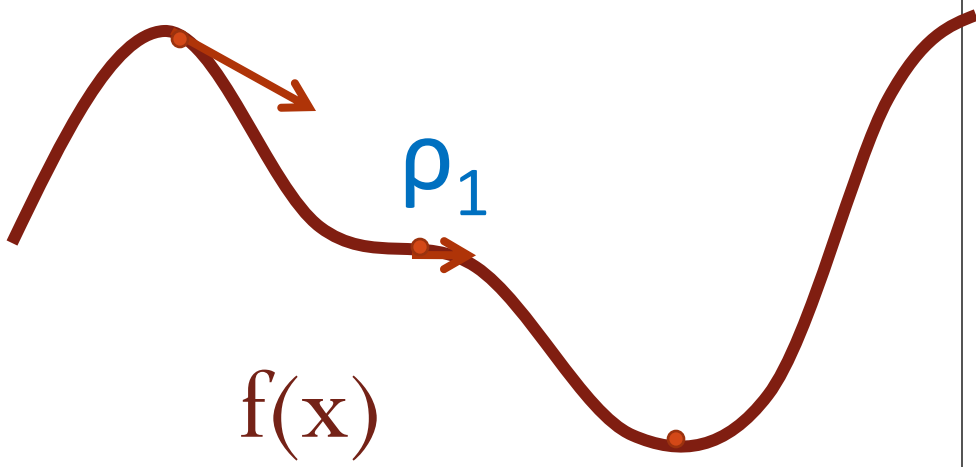
- Large class of problems can be solved as LP.
  - Any polynomial algo can be emulated in polynomial time
- Applicable for harder problems as well! (NP, ILP, discrete/combinatorial optimizations)

Identifying the Best Solver

- Gradient Descent shown to be robust under errors
  - For zero mean noisy gradient calculations
  - With diminishing step sizes
- Applying Gradient Descent to an unconstrained problem:

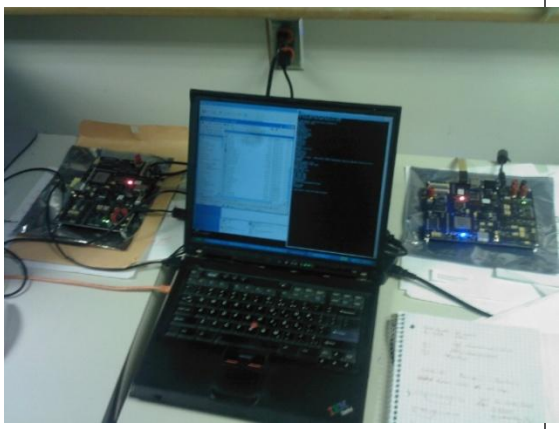
$$x_{i+1} = x_i + \alpha\rho$$

$$\rho = -\nabla f(x_i)$$

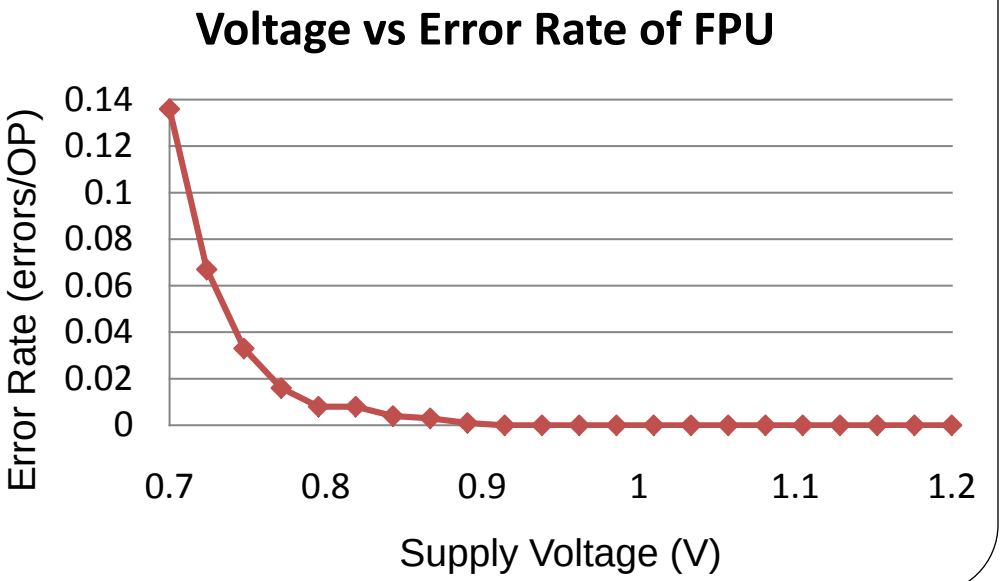


Methodology

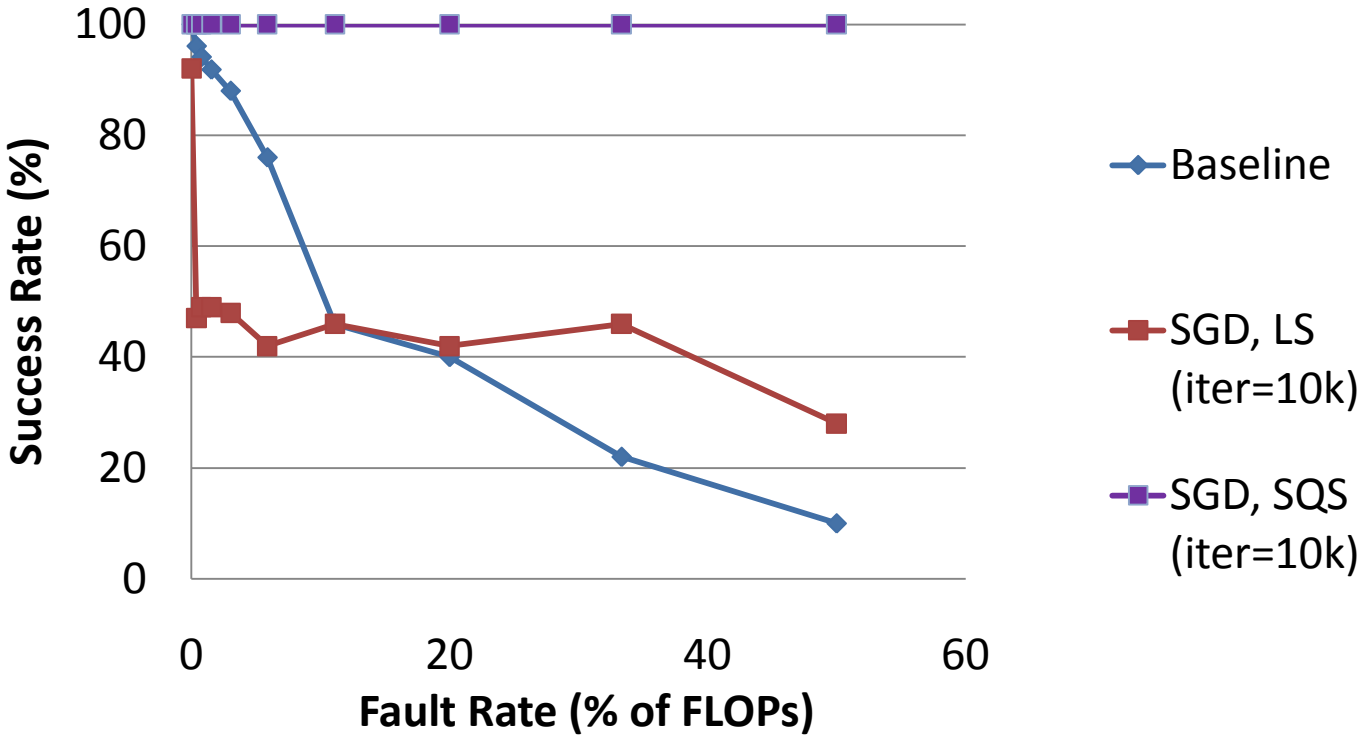
- Faults injected into the FPU of Leon3 soft core run on Altera Stratix II EP2S190 FPGA
  - other modules assumed to be fault free



- Time between and actual bit faults approximated from random numbers generated from LFSR

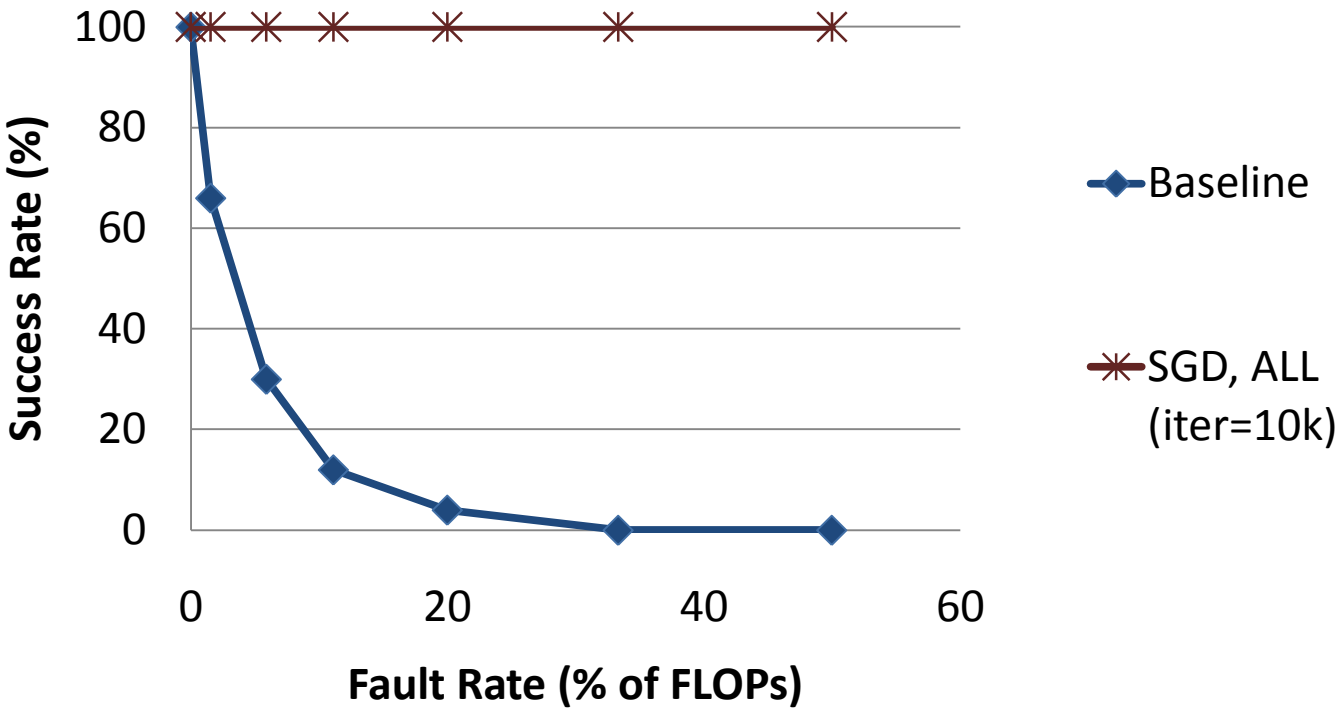


Sorting (size=10) using Gradient Descent



100% Accuracy with Sort using SGD even in face of large error rates. Note that traditional implementations of sort not considered error tolerant.

Graph Matching(5x6) using Gradient Descent (10k Iterations)



100% Accuracy for arbitrary inputs even for large fault rates

A Discussion of Runtime and Energy

- A single iteration of an optimization formulation may have higher complexity than the baseline for some apps (e.g. sort).
  - Robustification still useful when the computational substrate is inherently stochastic.
- For other applications, a single optimization iteration may be faster compared to the baseline (e.g. LSQ, graph matching).
  - Robustification useful for such applications for saving energy. (>10X savings for relaxed accuracy targets for LSQ)
  - Energy savings increase when parallelism of the optimization version is exploited
    - Example: 8x-20x energy savings for graph matching with an accelerator architecture. [SASP2011]

Current Work

- Automated robustification.

- Sacrificing Generality for Efficiency
  - E.g. Robustifying sparse linear algebra through approximate algorithmic correction

Summary and Conclusions

- Too much cost for computing with guarantees
- Processors need to be designed and architected from the ground up to manage the number and nature of errors (stochastic processors) to deal with the non-determinism problem for late-CMOS / post-CMOS technologies
- Conventional design and architecture approaches optimize for correct operation; inadequate when errors are allowed
  - Proposed Stochasticity-aware Architecture / Design Methodologies present significant power savings
- Application Robustification: A rich area of research with significant potential for robustness . Efficiency improvements.