AUDIT: Stress Testing the Automatic Way

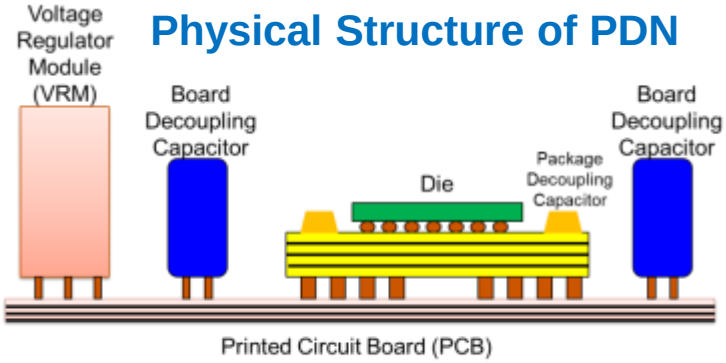
Youngtaek Kim, ***Lizy Kurian John***
ECE, The University of Texas at Austin
Sanjay Pant, ***Srilatha Manne,*** ***Michael Schulte,***
W. Lloyd Bircher, ***Madhu S. Sibi Govindan***
AMD Research and PPO Lab, Advanced Micro Devices, Inc.

Outline

- AUDIT: AUtomated DI/dT Stressmark Generation
  - is an **automation framework for stressmark generation**
    - Target: Multi-core processor
    - Finding Max. Voltage Droop: Genetic Algorithm with hardware measurement
  - generates **effective di/dt stressmarks in a short time**
    - Larger voltage droop than other benchmarks/stressmarks
    - Higher voltage failure points than other benchmarks/stressmarks
  - works well with **different configurations / architectures**
    - Throttling off / on
    - Different processor

Introduction: Reliability (*di/dt* = inductive noise) Issue

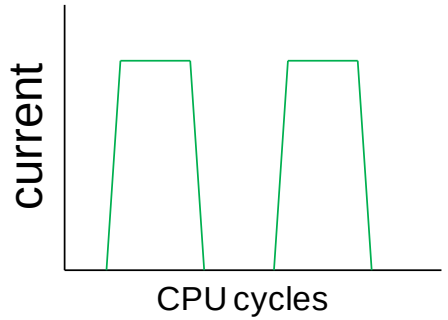
Physical Structure of PDN

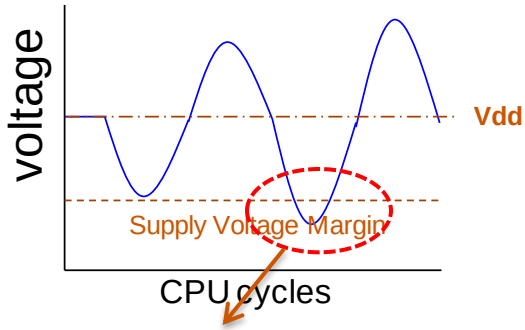


Source: Power Distribution Networks with On-Chip Decoupling Capacitors

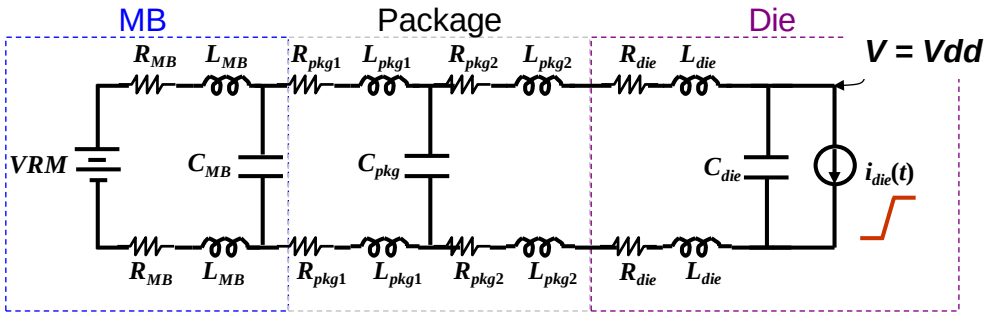
Inductive Noise (*di/dt* noise)

$$v_n(t) = L \cdot \frac{di(t)}{dt}$$





Corresponding Circuit Representation



$V = Vdd – L*di/dt – RI$

Insufficient voltage

↓

Increase in delay

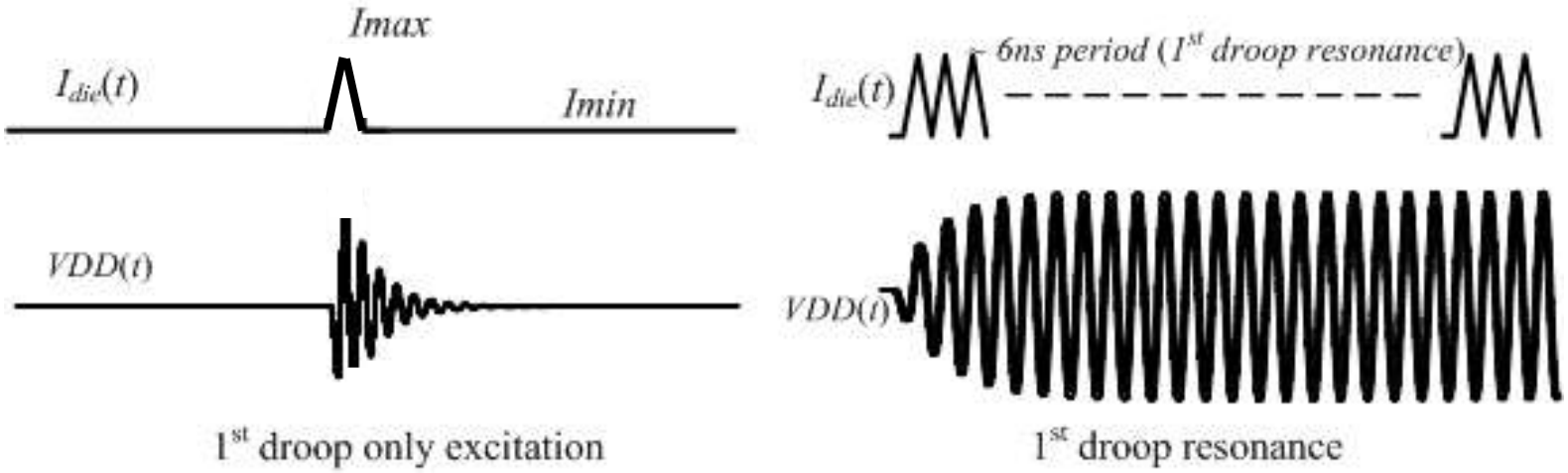
↓

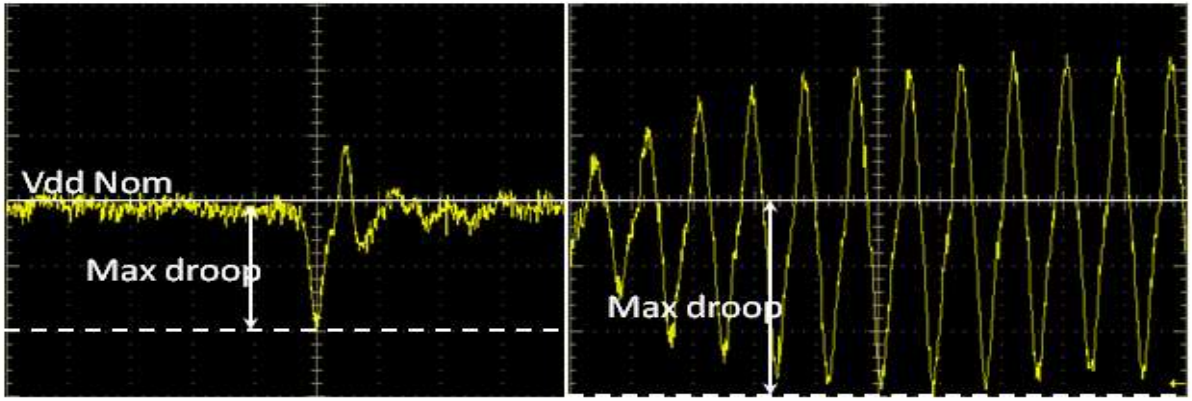
Timing violation!

Introduction: Supply Voltage Failure Symptoms

- Unexpected Value / Wrong Result
  - bit-flip: 0 → 1 or 1 → 0
- OS Freezing / System Hang
- Blue Screen
- Sudden Shutdown

Background: Characteristics of di/dt Voltage Noise



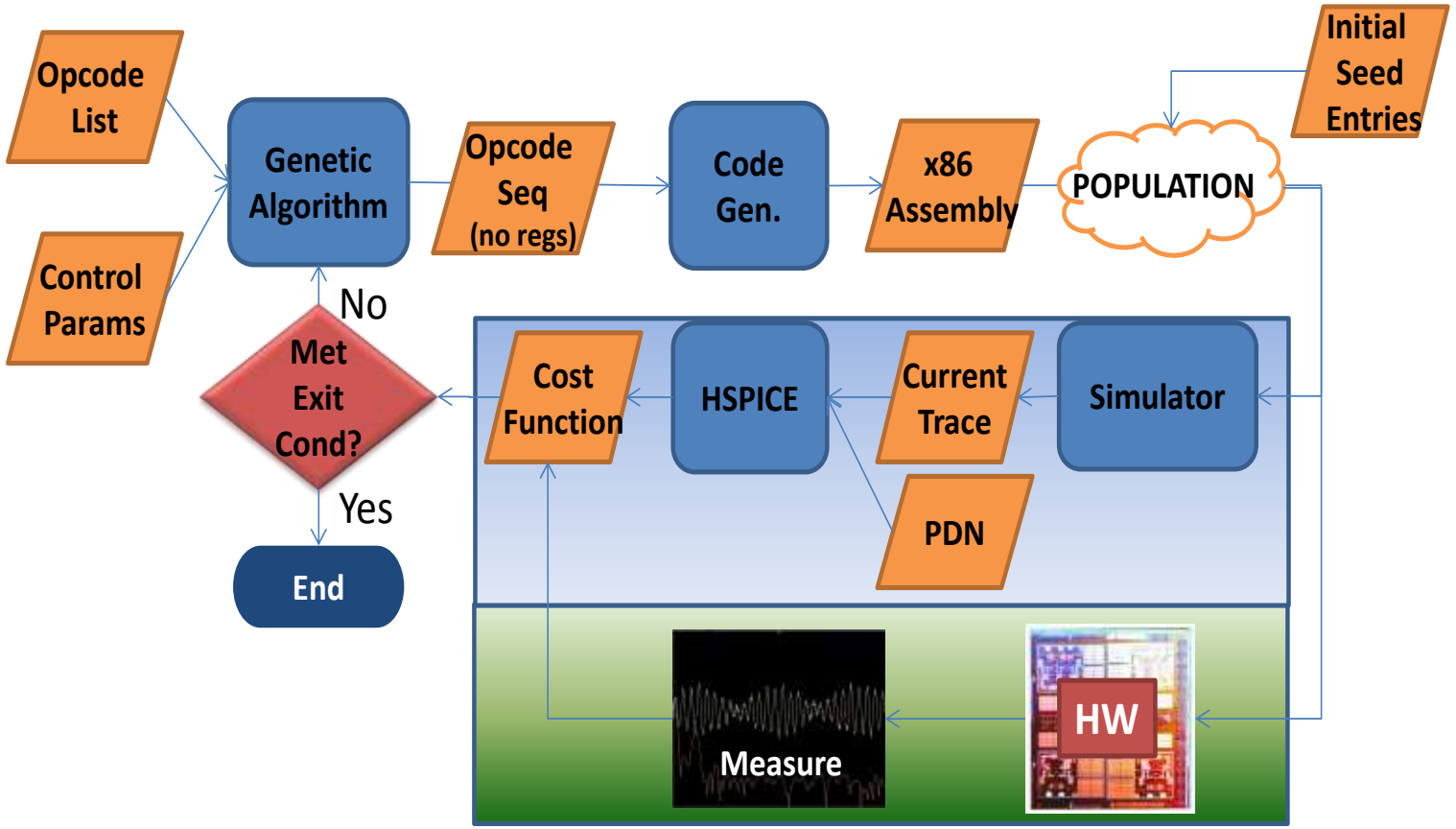


Related Work & Motivation

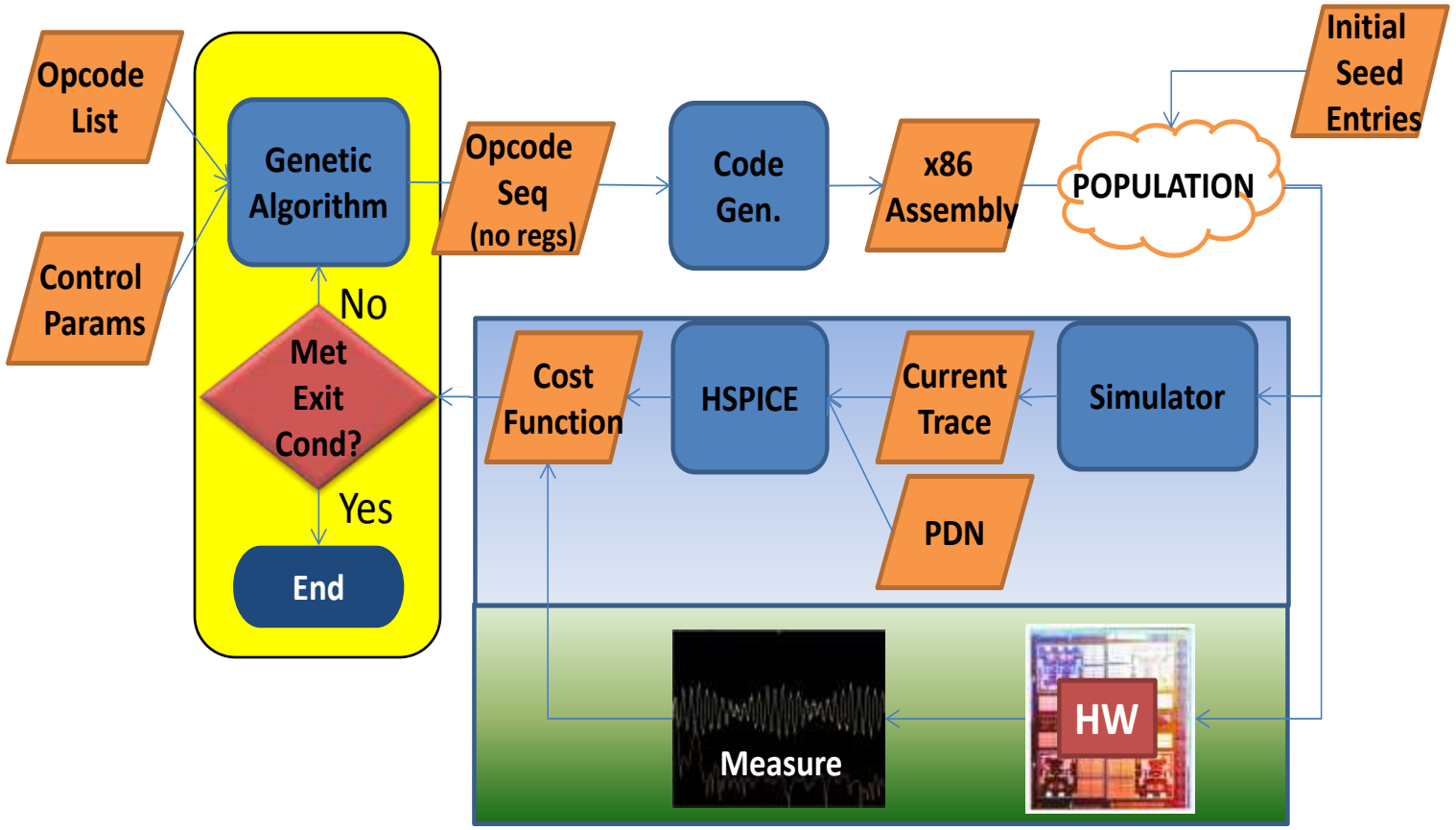
- **To characterize di/dt voltage noise in a microprocessor**
  - Using standard benchmarks
    - SPEC benchmarks: ineffective to test voltage margin
  - Generating and running di/dt stressmarks
    - Manual stressmark [Joseph, HPCA'03]
      - Inefficient to make a new manual stressmark for different configurations
    - Instruction Scheduling using Integer Linear Programming (ILP) [Ketkar, MICRO'09]
      - Difficult to make linear algebra formula for a complex system
    - Genetic Algorithm [Joshi, HPCA'08] [Kim, ISLPED'11]
      - Single-core, simulation only

→Automatic di/dt Stressmark Generation using Genetic Algorithm with Post-Silicon Hardware Measurement

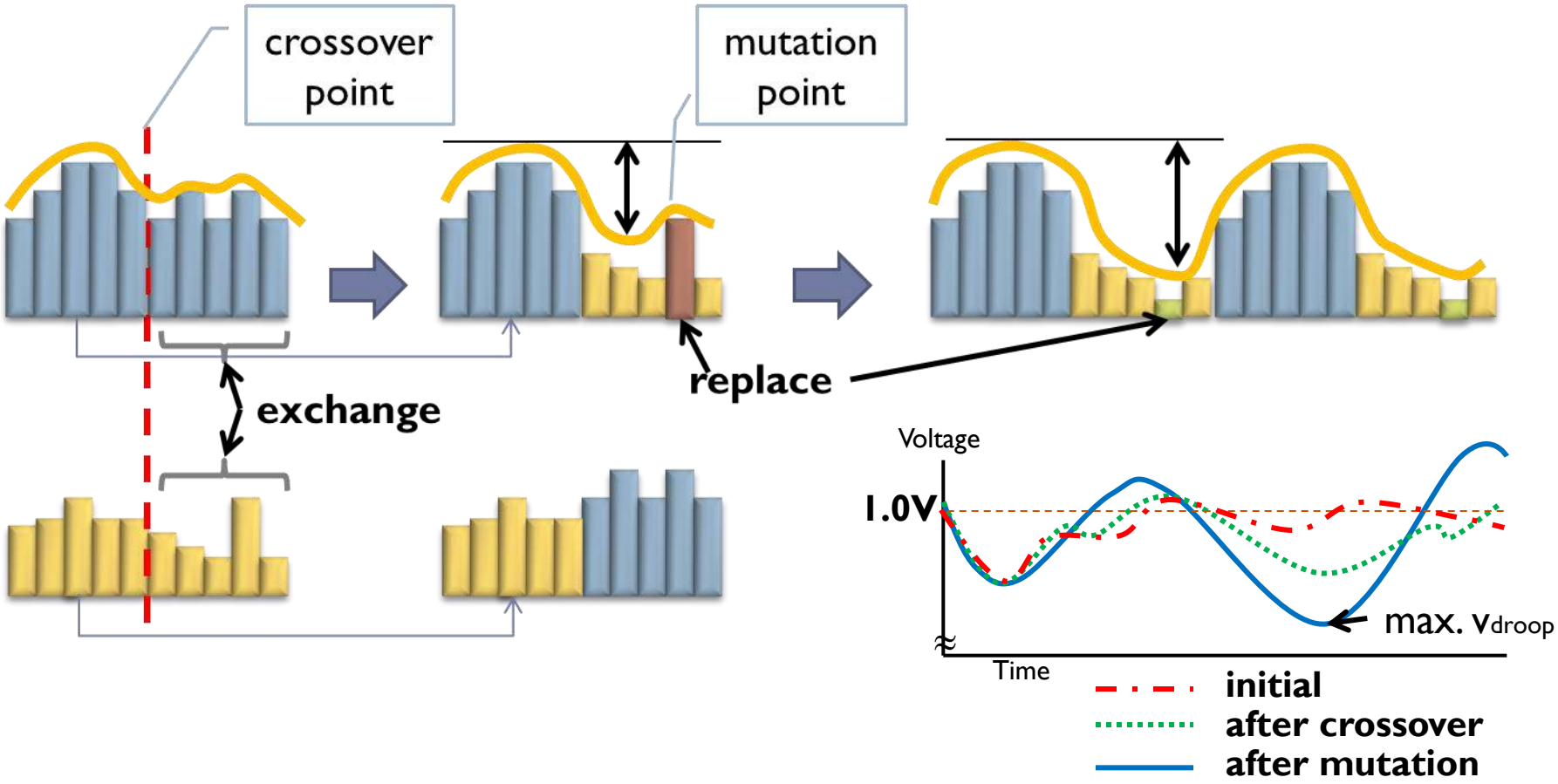
AUDIT: AUtomated DI/dT Stressmark Generation



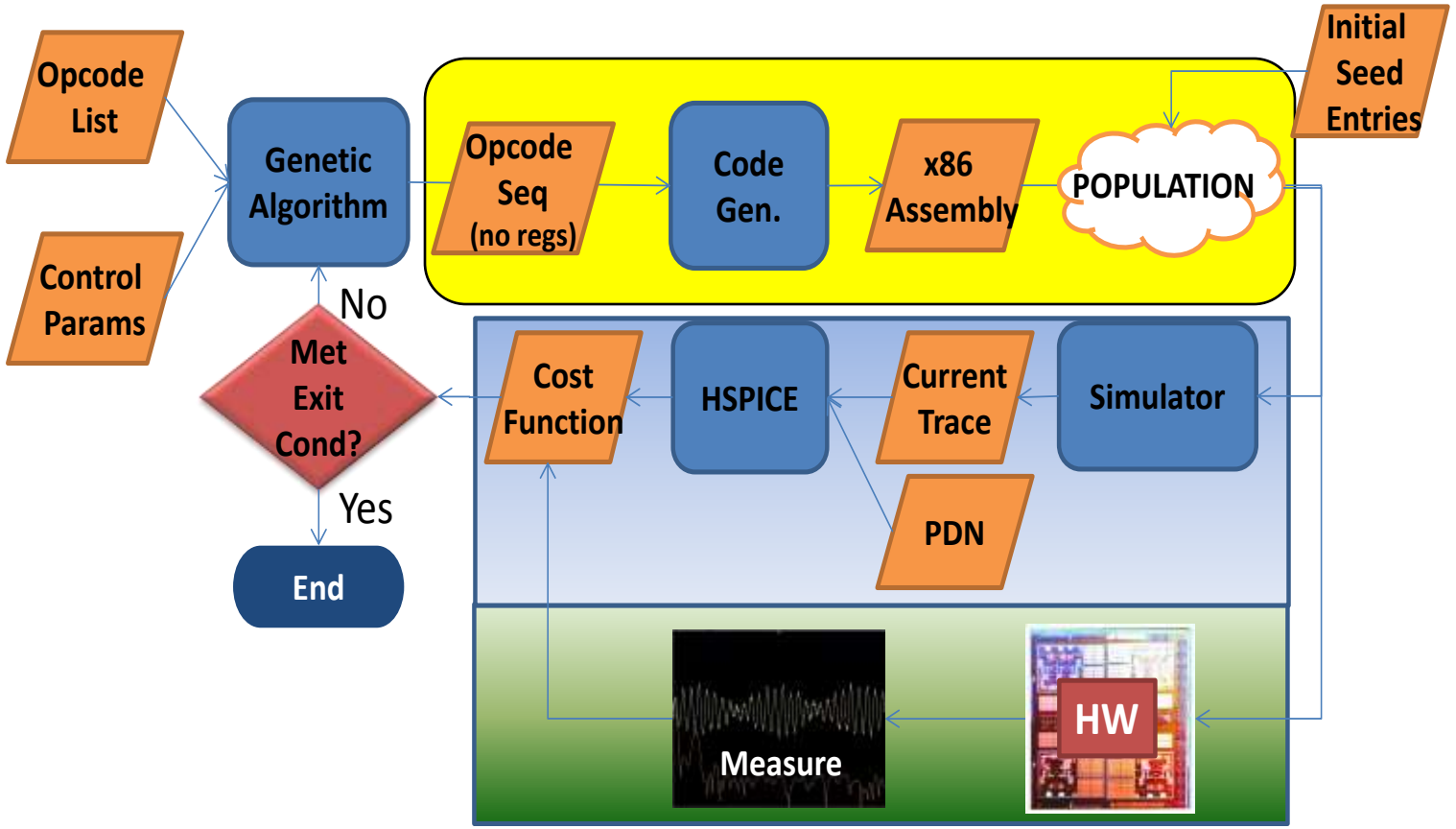
AUDIT: AUtomated DI/dT Stressmark Generation



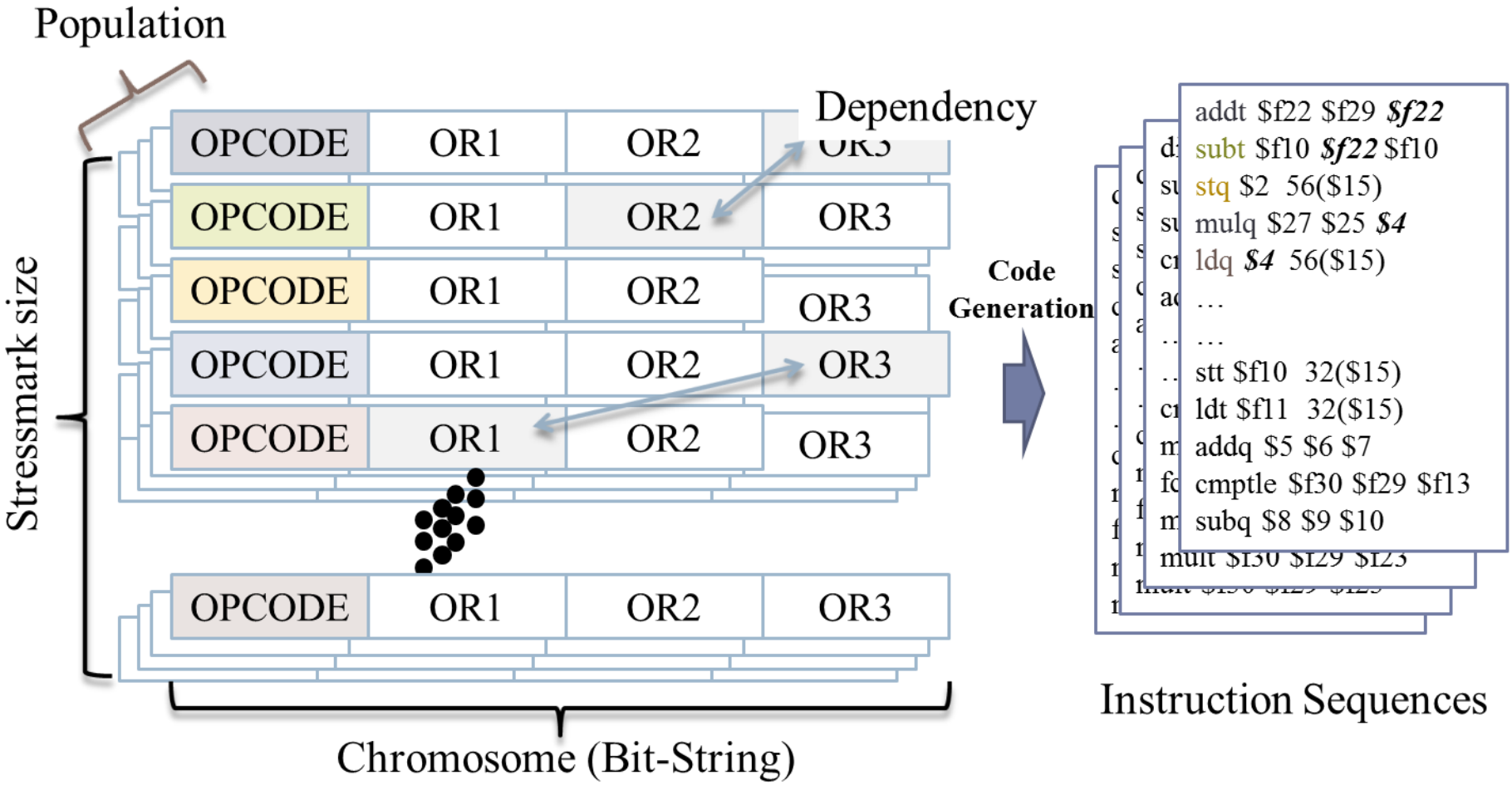
AUDIT – Genetic Algorithm: Operational Concept



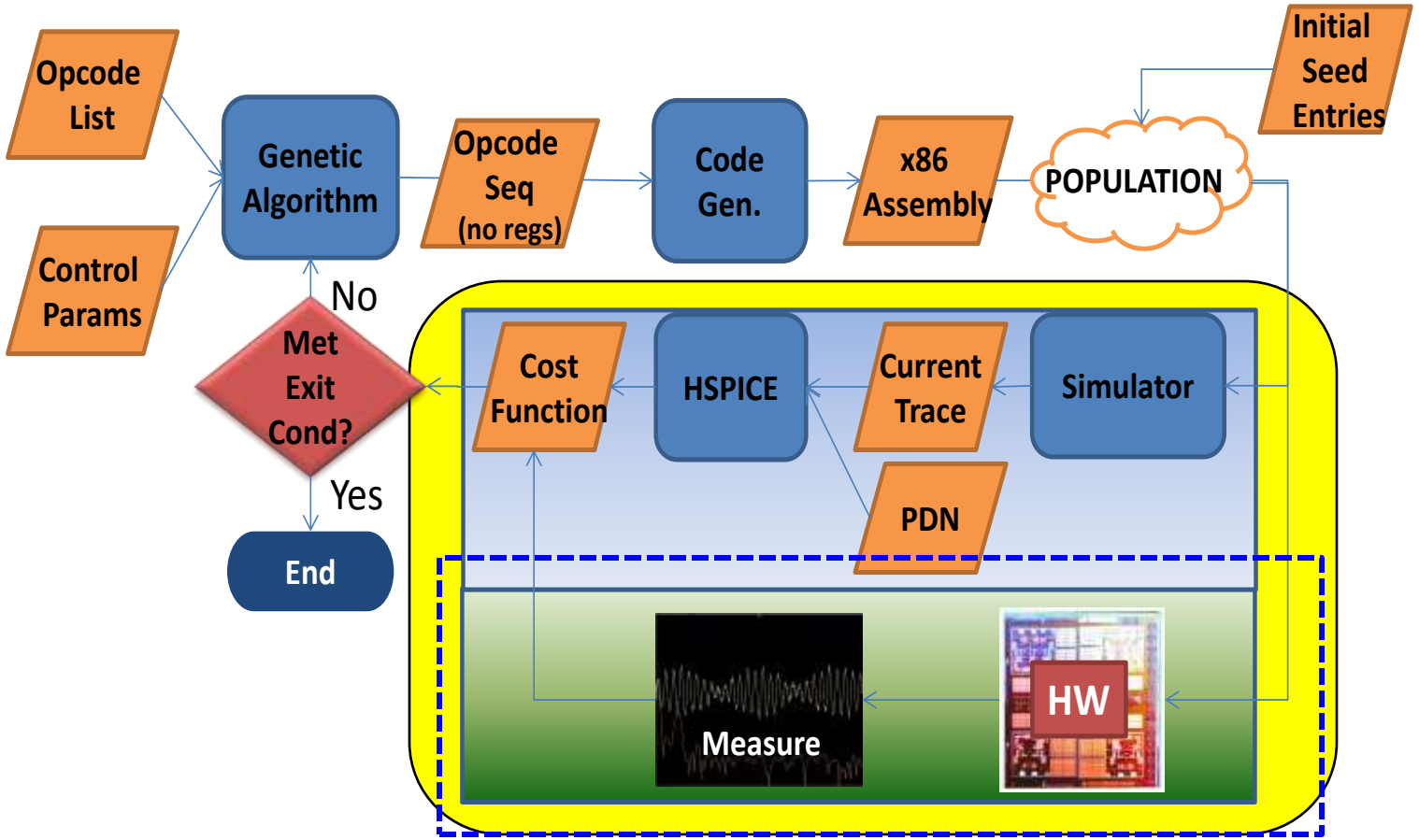
AUDIT: AUtomated DI/dT Stressmark Generation



AUDIT – Instruction Sequence Generation

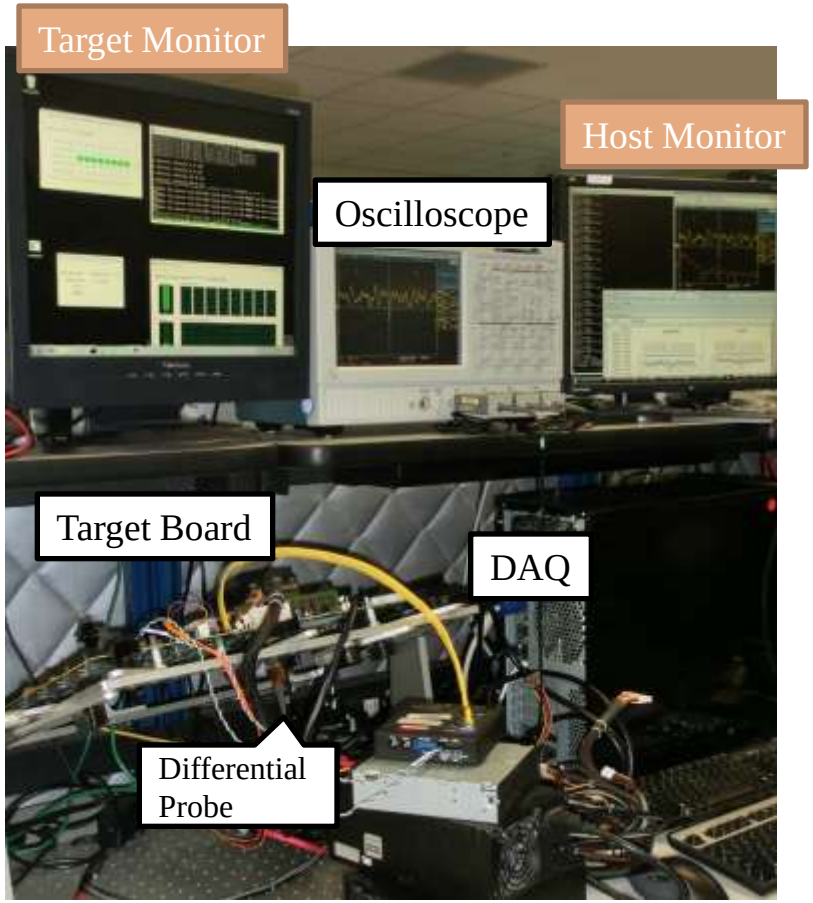


AUDIT: AUtomated DI/dT Stressmark Generation



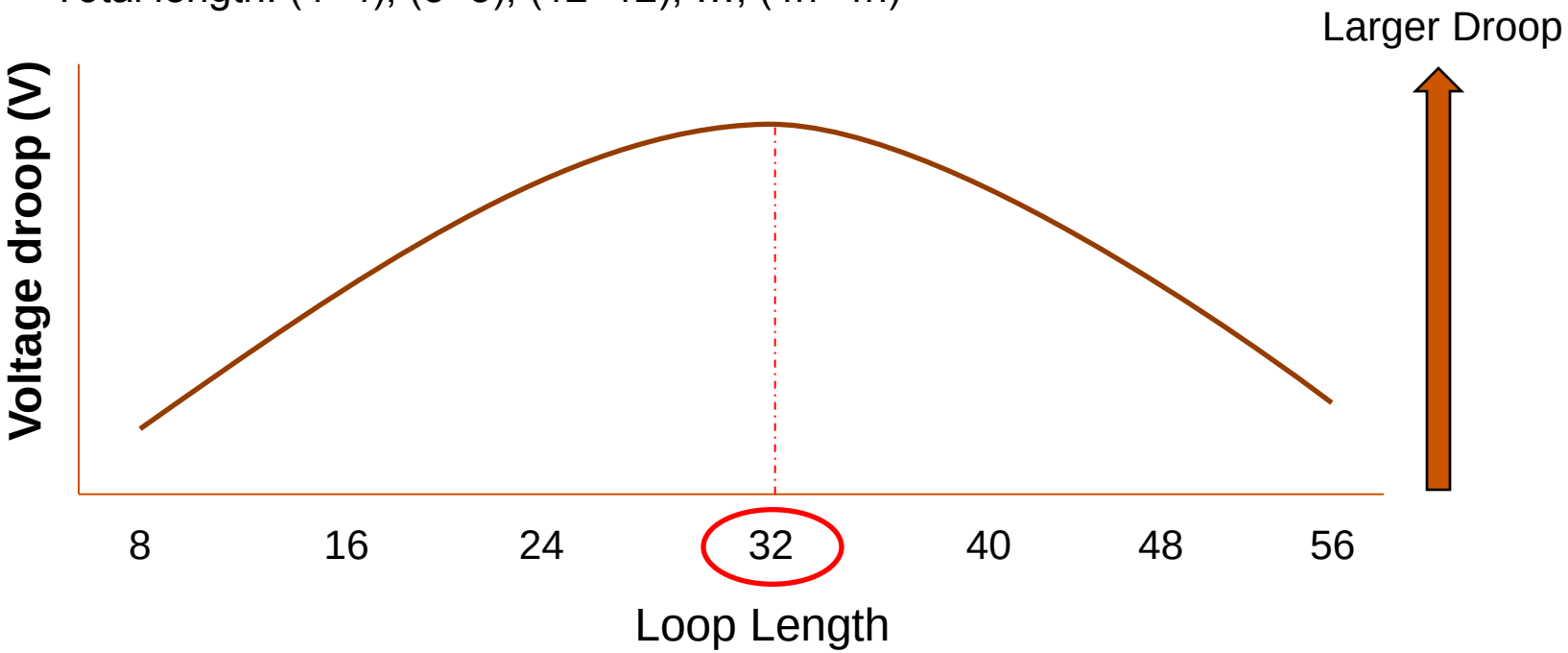
AUDIT – Hardware Measurement

- Hardware Measurement for Max. Voltage Droop and Power



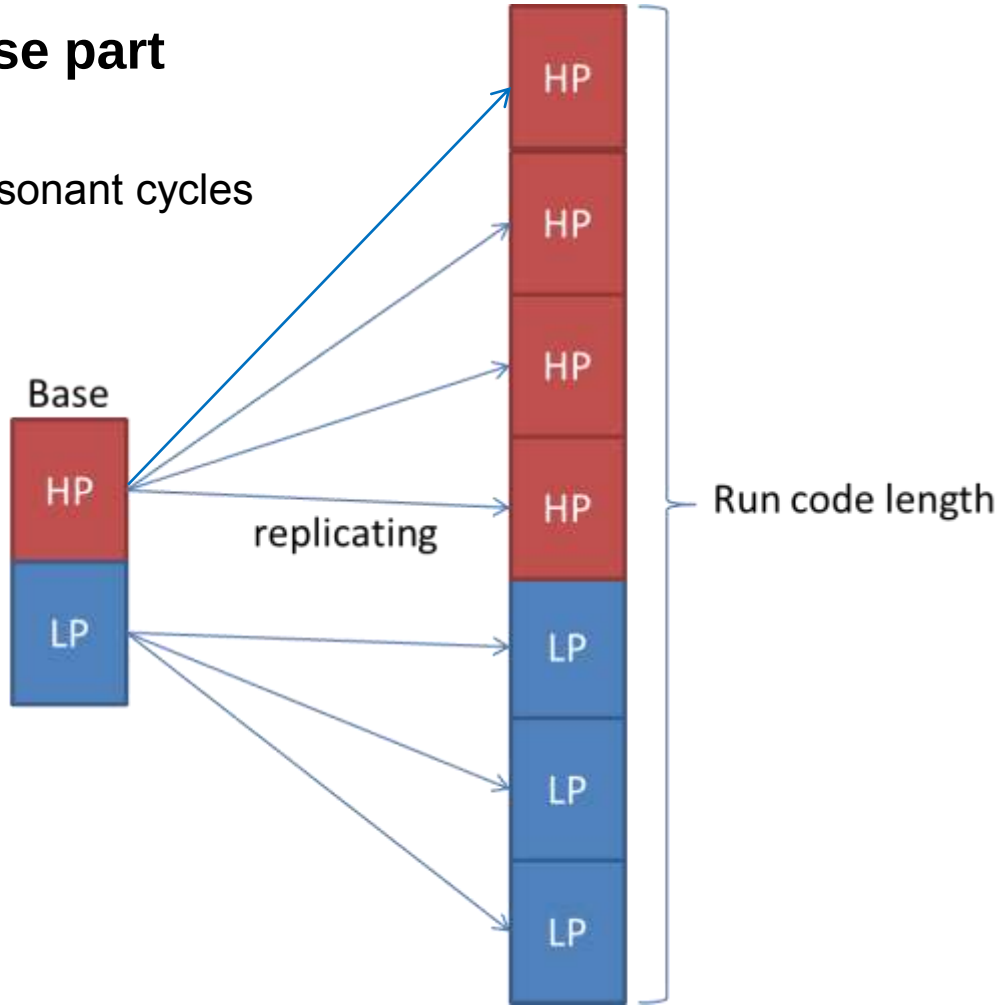
Step 1: Frequency Sweep

- To find 1st droop resonant frequency,
  frequency sweep = increasing code length with simple instructions
  - HP length: 4, 8, 12, …, 4n
  - LP length: 4, 8, 12, …, 4n
  - Total length: (4+4), (8+8), (12+12), …, (4n+4n)



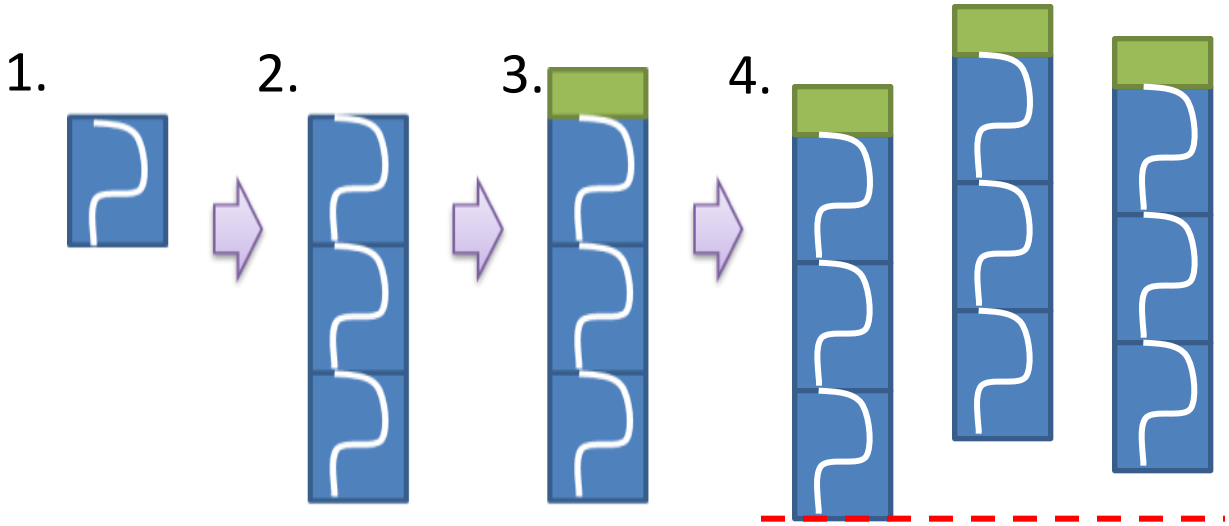
Step 2: Using Sub-blocks for GA

- **Scaling & Replicating the Base part**
  1. Schedule “Base”
  2. Replicate “Base” according to resonant cycles



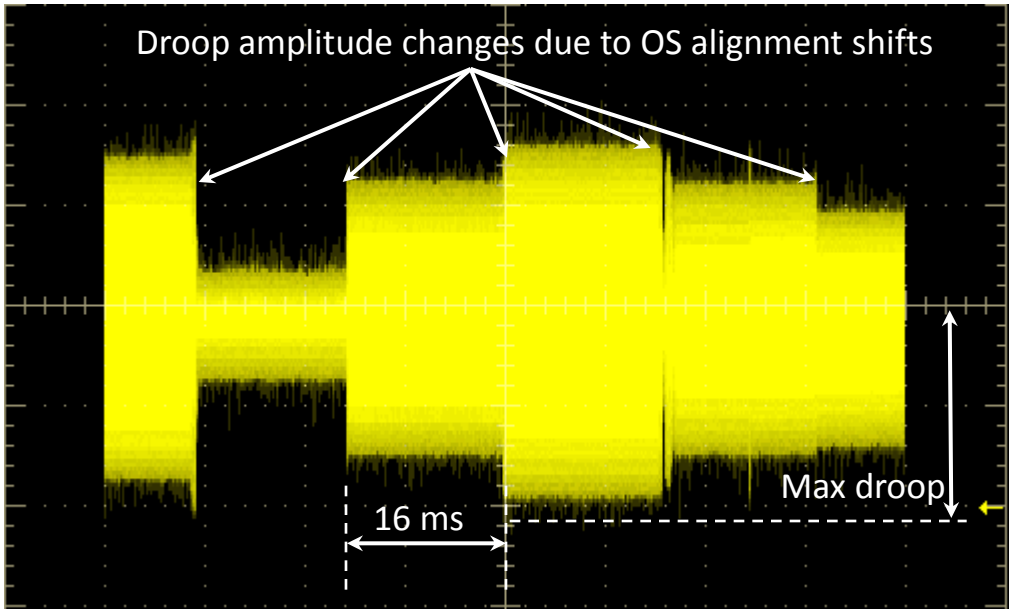
Step 3: Code Generation for Multiple Threads

1. Prepare a core part – one high-low power pattern
2. Make multiple copies of <1> to increase the intensity of resonance
3. Add a header part that contains initialization codes
4. Make multiple copies of <3> according to the number of threads



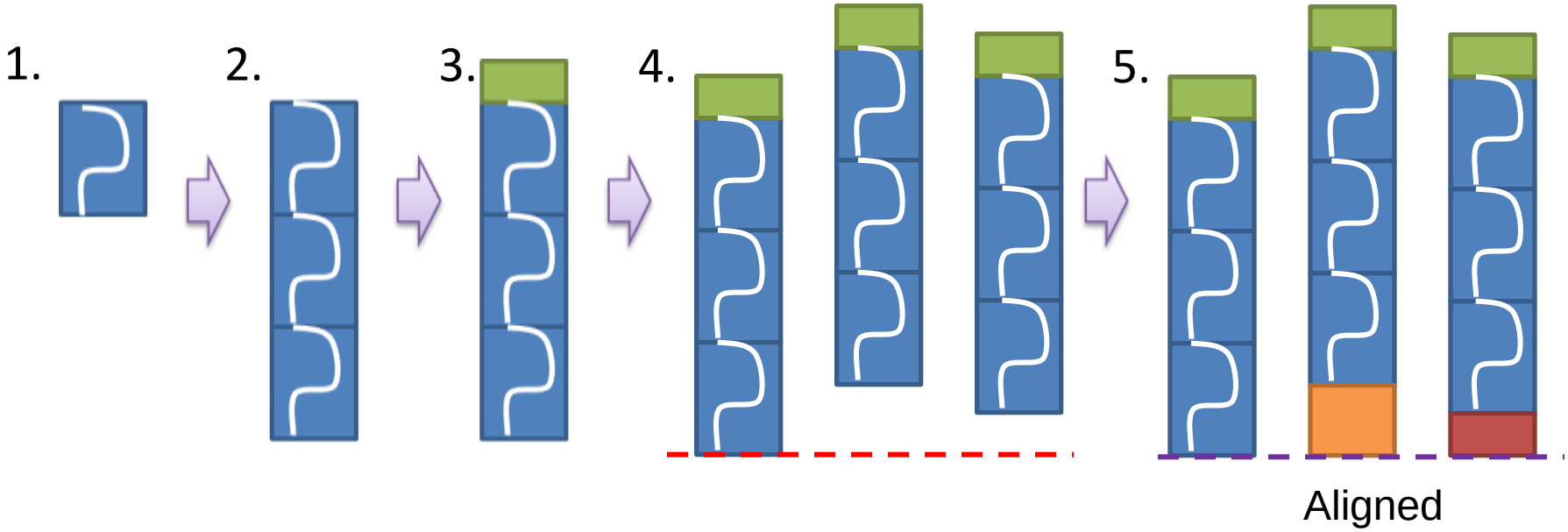
Step 3: Code Generation for Multiple Threads

- **Natural dithering: thread alignment shifts due to OS**



Step 3: Code Generation for Multiple Threads

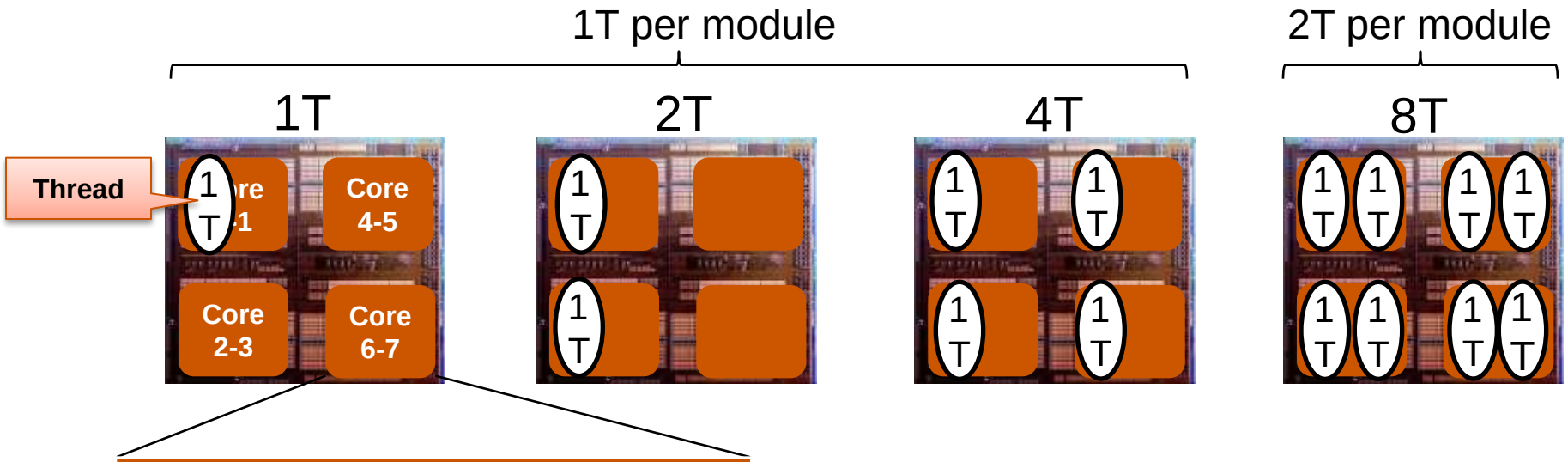
1. Prepare a core part – one high-low power pattern
2. Make multiple copies of <1> to increase the intensity of resonance
3. Add a header part that contains initialization codes
4. Make multiple copies of <3> according to the number of threads
5. Attach dithering parts to each thread for alignment



Experimental Methodology - Benchmark

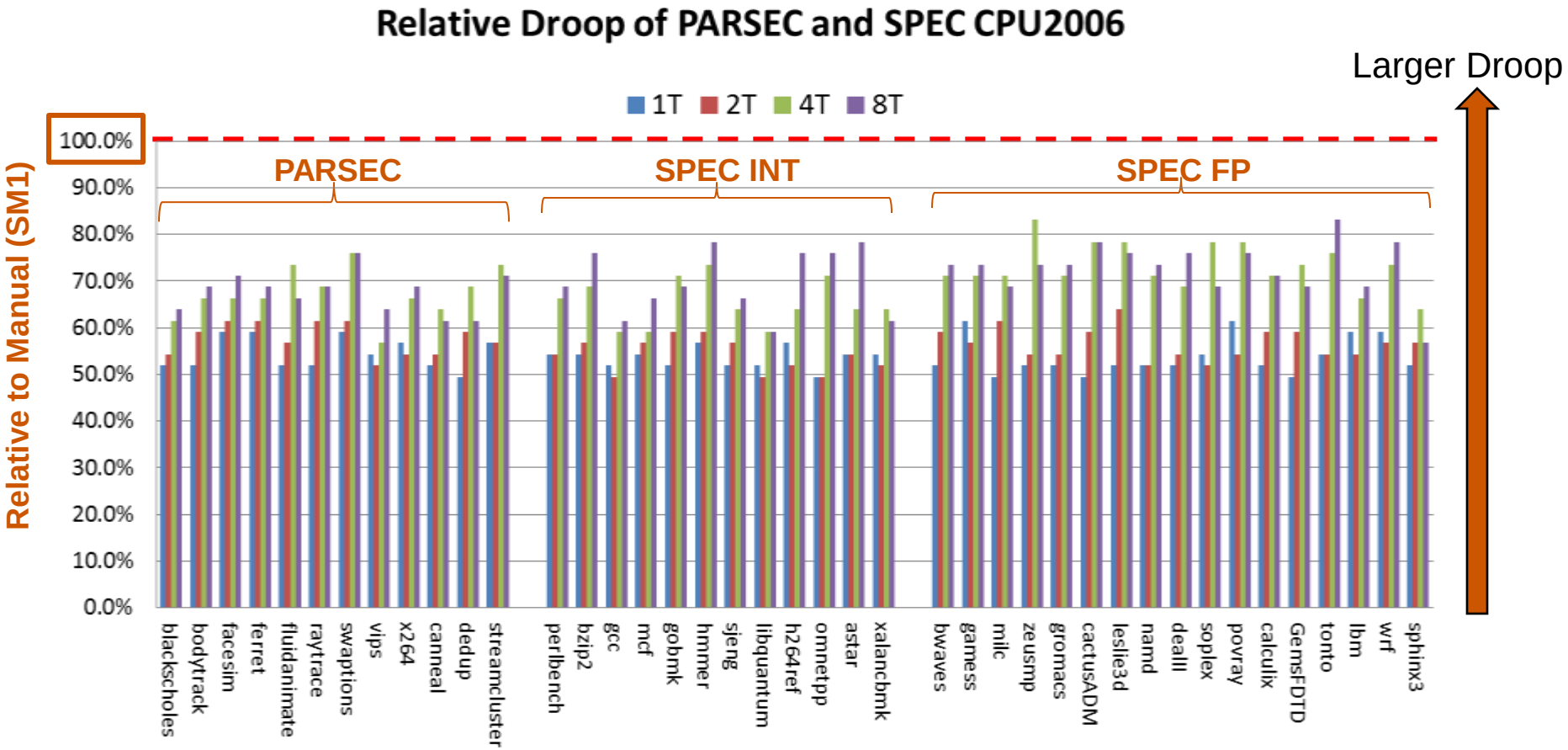
- Benchmark
  - Standard benchmark
    - SPEC CPU2006 (12 INTs and 17 FPs): multi-programed
    - PARSEC: multi-threaded
  - Stressmark
    - Manual: SM1 and SM2 (single+resonant) and SM-Res (resonant)
    - AUDIT: A-Ex (single) and A-Res (resonant)
- Compiler: NASM, gcc 4.6.2
- OS: Windows 7, RedHat Linux Enterprise 6

Experimental Methodology - Thread Configuration

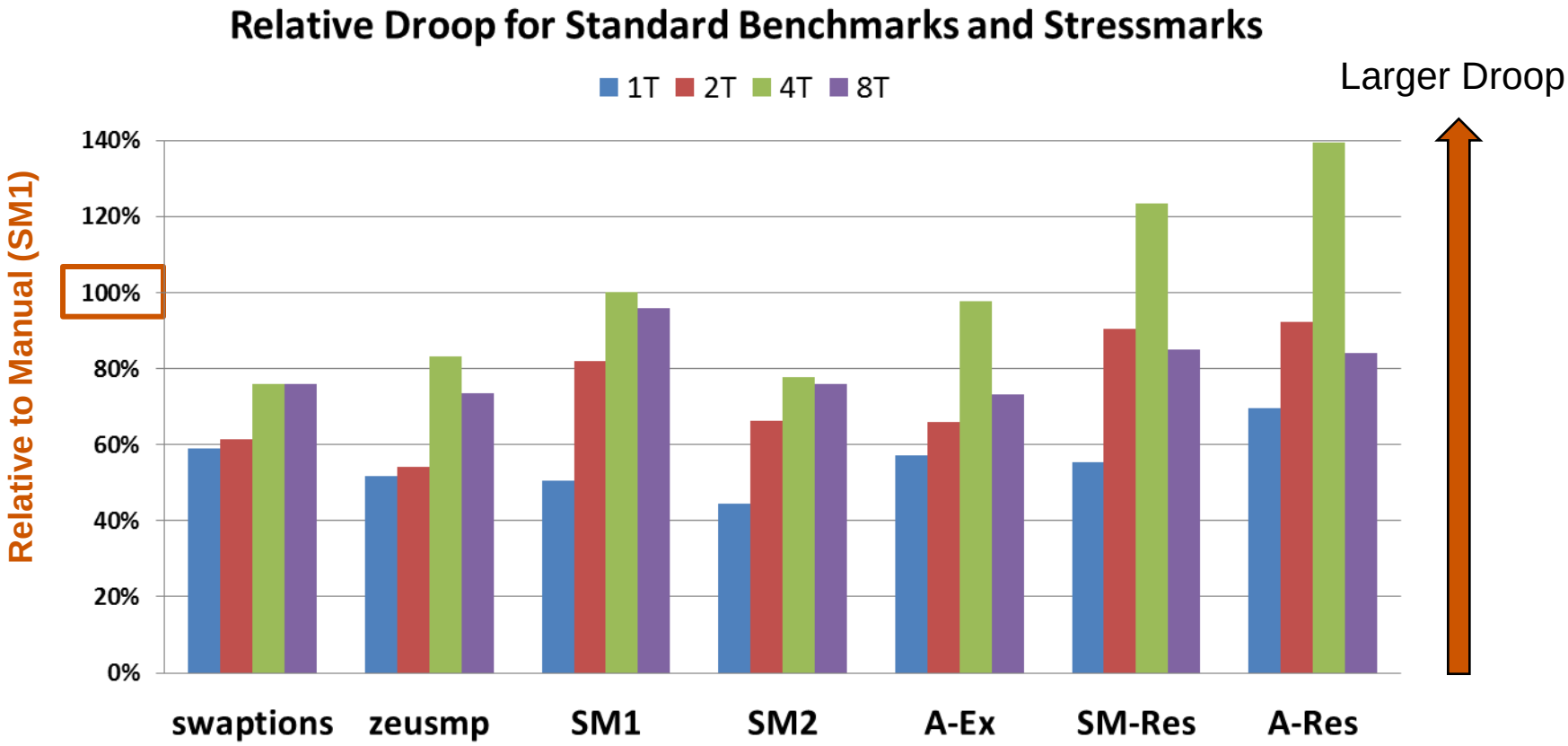


AMD Bulldozer	
Cores	2 cores per Bulldozer
Shared	Front-end FPU L2 Cache

Experimental Results - Max. Voltage Droop

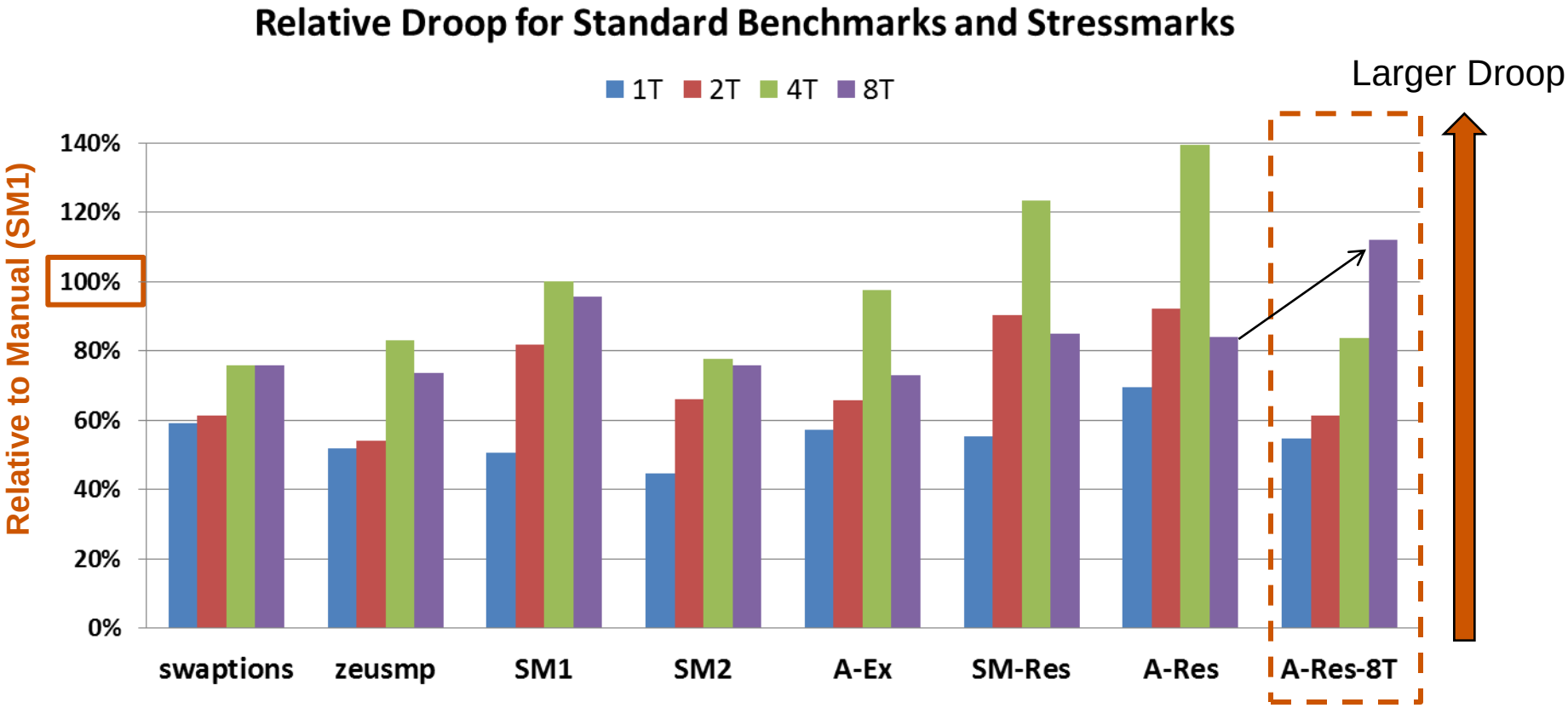


Experimental Results - Max. Voltage Droop



- **Manual: SM1, SM2, SM-Res**
- **AUDIT Single Droop: A-Ex**
- **AUDIT Resonant: A-Res**

Experimental Results - Max. Voltage Droop

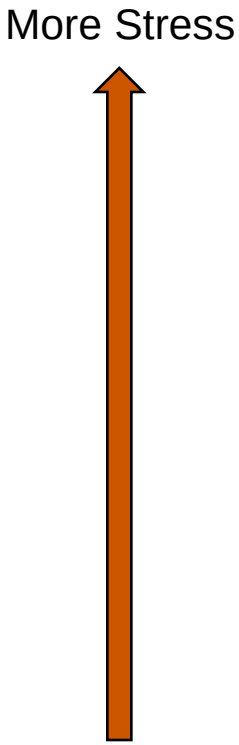


- **Manual: SM1, SM2, SM-Res**
- **AUDIT Single Droop: A-Ex**
- **AUDIT Resonant: A-Res, A-Res-8T**

Experimental Results - Voltage at Failure

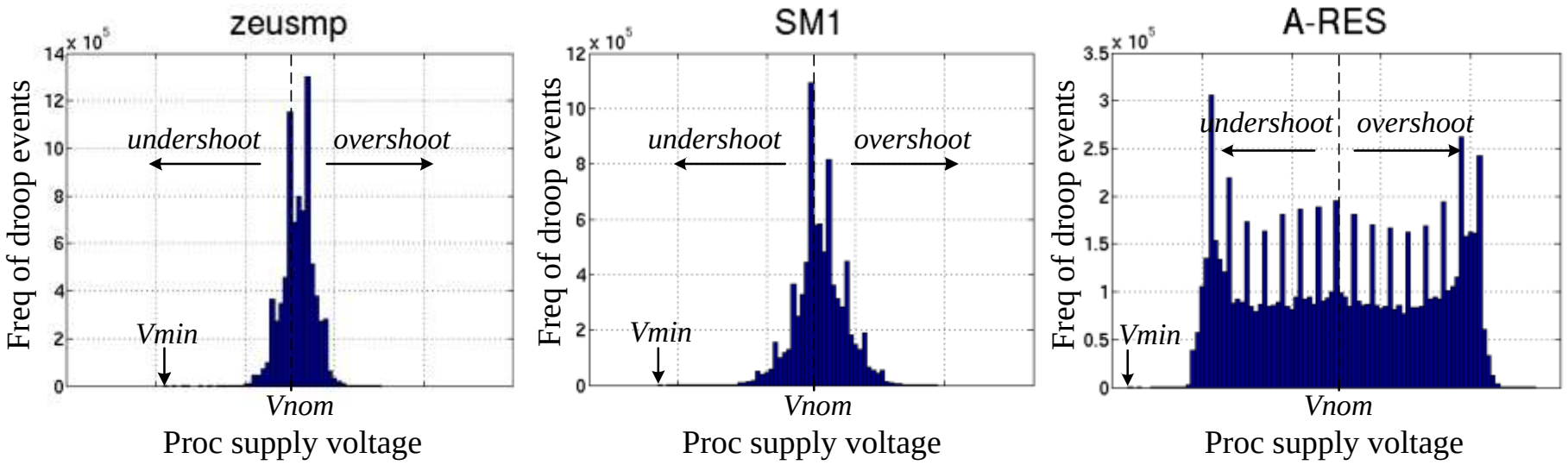
- Lowering the operating voltage & finding the voltage at failure
- Higher voltage at failure → more stressful benchmark

Benchmark	V at Fail
A-Res	**V_F**
SM-Res	V_F – 12.5 mV
SM1	V_F – 62.5 mV
A-Ex	V_F – 75.0 mV
SM2	V_F – 87.5 mV
zeusmp	V_F – 125 mV
swaptions	V_F – 125 mV



Experimental Results - Droop Probability

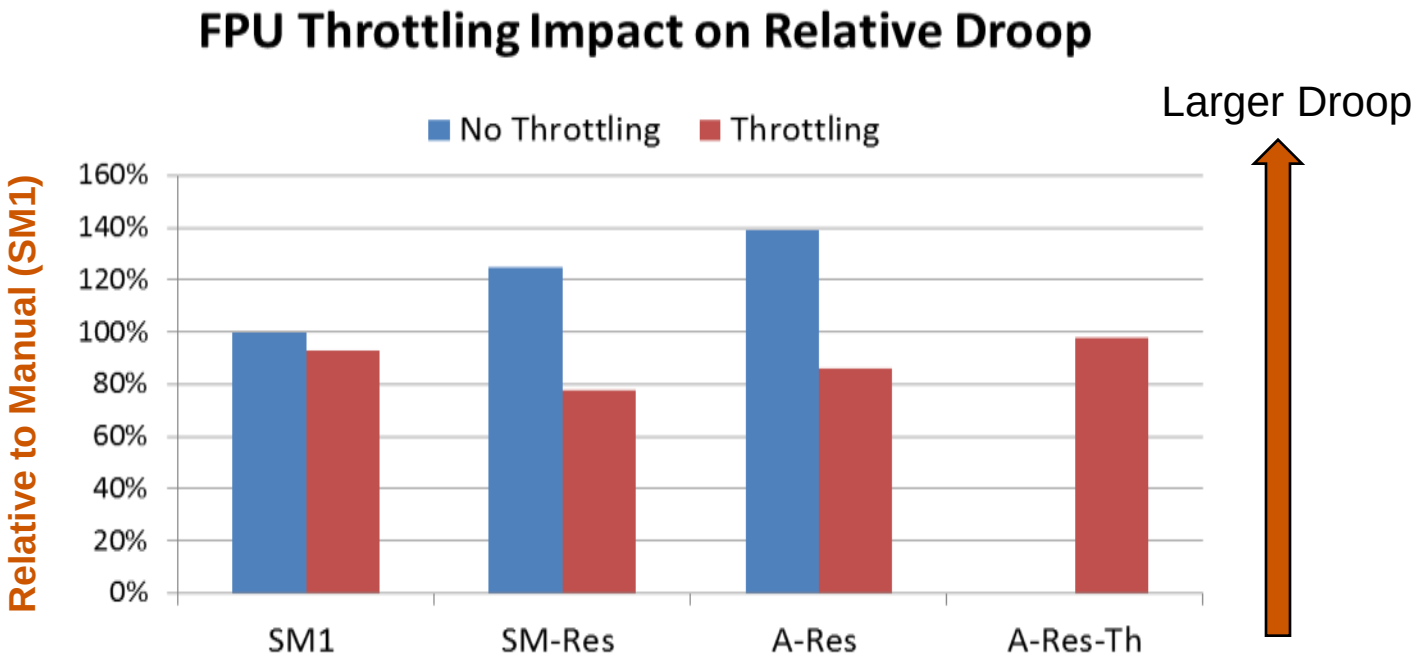
- Histogram of Droop Event
  - 8M samples are captured at Max. voltage droop
  - More frequent, larger droop → more probability of failure



**** Max. Droop = Vnom - Vmin***

Experimental Results - FPU Throttling

- Floating-Point Unit (FPU) Throttling
  - FPU Throttling adjusts power ramp-up/down rates to mitigate di/dt effect
  - **A-Res-Th** is generated with FPU Throttling



Experimental Results - Different Processor

- AMD Phenom II X4
  - Different #of cores: 4 cores (← 8 cores in AMD Bulldozer case)
  - Different issue widths: 3 issues (← 4 issues in AMD Bulldozer case)

	zeusmp	SM2	A-Res
Relative Droop	0.82	1	1.10
Failure Point	V_F – 50 mV	V_F	V_F

Conclusion

- **AUDIT: AUtomatic DI/dT Stressmark Generation**
  - Targets Multi-core processor
  - Genetic Algorithm with Hardware Measurement
  - 40% more voltage droop than manual stressmark (SM1)
  - 62mV higher voltage failure points than manual stressmark (SM1)
  - Generation time is less than 5 hours
  - Works well with different configurations / architectures

Thank You!

Any Questions?