The Variability Expeditions:

Variability-Aware Software for Efficient Computing With Nanoscale Devices.

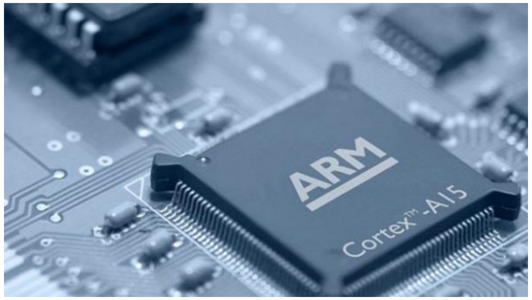
Rajesh K. Gupta

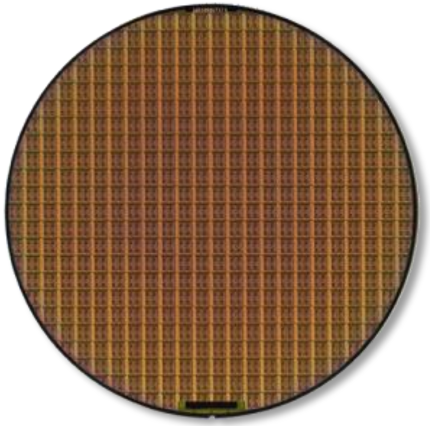
Nikil Dutt, UCI
Punit Gupta, UCLA
Mani Srivastava, UCLA
Lucas Wanner, UCLA
Steve Swanson, UCSD

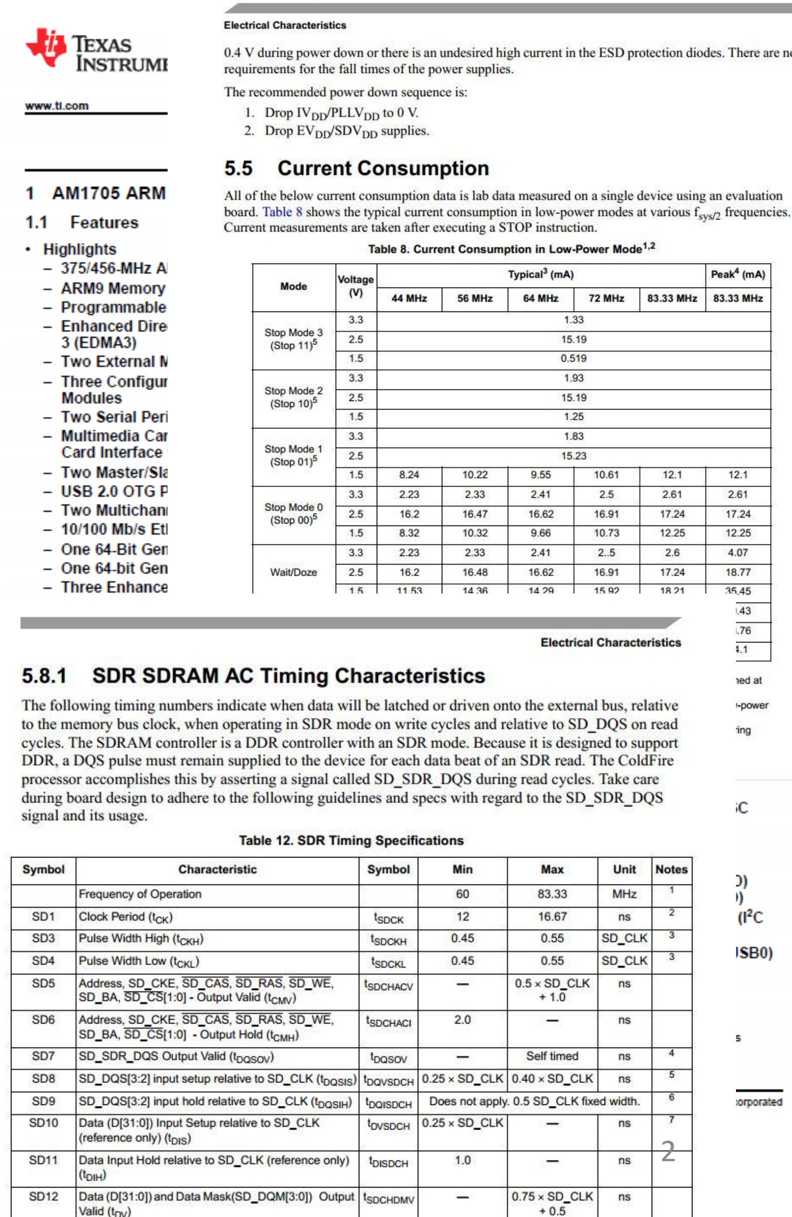
Lara Dolecek, UCLA
Subhashish Mitra, Stanford
YY Zhou, UCSD
Tajana Rosing, UCSD
Alex Nicolau, UCI
Ranjit Jhala, UCSD
Sorin Lerner, UCSD
Rakesh Kumar, UIUC
Dennis Sylvester, UMich

To a software designer, all chips look alike

To a hardware engineer, a chip is delivered as per contract in a data-sheet.







Texas Instruments

www.ti.com

1 AM1705 ARM

1.1 Features

- Highlights
  - 375/456-MHz A
  - ARM9 Memory
  - Programmable
  - Enhanced Dire 3 (EDMA3)
  - Two External M
  - Three Configur Modules
  - Two Serial Peri
  - Multimedia Car Card Interface
  - Two Master/Sla
  - USB 2.0 OTG P
  - Two Multichan
  - 10/100 Mb/s Et
  - One 64-Bit Gen
  - One 64-bit Gen
  - Three Enhance

Electrical Characteristics

0.4 V during power down or there is an undesired high current in the ESD protection diodes. There are n requirements for the fall times of the power supplies.

The recommended power down sequence is:

1. Drop IV_{DD}/$PLLV_{DD}$ to 0 V.
2. Drop EV_{DD}/SDV_{DD} supplies.

5.5 Current Consumption

All of the below current consumption data is lab data measured on a single device using an evaluation board. Table 8 shows the typical current consumption in low-power modes at various $f_{sys/2}$ frequencies. Current measurements are taken after executing a STOP instruction.

Table 8. Current Consumption in Low-Power Mode[1,2]

Mode	Voltage (V)	Typical[3] (mA)					Peak[4] (mA)
		44 MHz	56 MHz	64 MHz	72 MHz	83.33 MHz	83.33 MHz
Stop Mode 3 (Stop 11)[5]	3.3	1.33					
	2.5	15.19					
	1.5	0.519					
Stop Mode 2 (Stop 10)[5]	3.3	1.93					
	2.5	15.19					
	1.5	1.25					
Stop Mode 1 (Stop 01)[5]	3.3	1.83					
	2.5	15.23					
	1.5	8.24	10.22	9.55	10.61	12.1	12.1
Stop Mode 0 (Stop 00)[5]	3.3	2.23	2.33	2.41	2.5	2.61	2.61
	2.5	16.2	16.47	16.62	16.91	17.24	17.24
	1.5	8.32	10.32	9.66	10.73	12.25	12.25
Wait/Doze	3.3	2.23	2.33	2.41	2..5	2.6	4.07
	2.5	16.2	16.48	16.62	16.91	17.24	18.77
	1.5	11.53	14.36	14.29	15.92	18.21	35.45

Electrical Characteristics

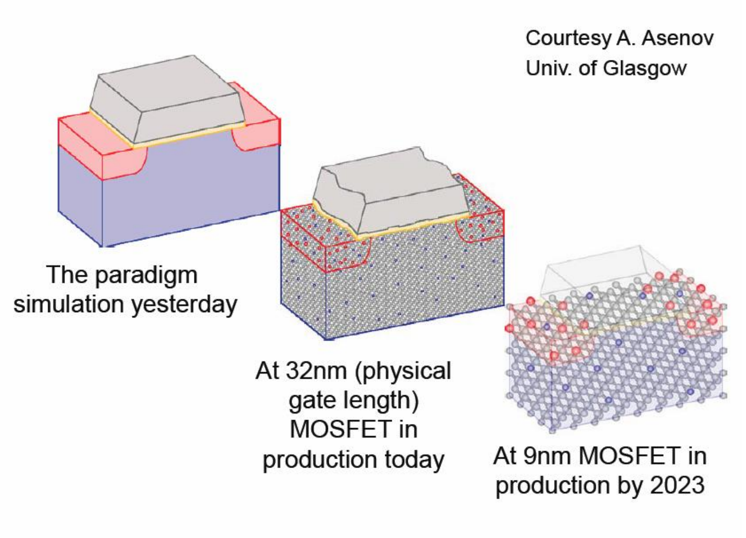
5.8.1 SDR SDRAM AC Timing Characteristics

The following timing numbers indicate when data will be latched or driven onto the external bus, relative to the memory bus clock, when operating in SDR mode on write cycles and relative to SD_DQS on read cycles. The SDRAM controller is a DDR controller with an SDR mode. Because it is designed to support DDR, a DQS pulse must remain supplied to the device for each data beat of an SDR read. The ColdFire processor accomplishes this by asserting a signal called SD_SDR_DQS during read cycles. Take care during board design to adhere to the following guidelines and specs with regard to the SD_SDR_DQS signal and its usage.

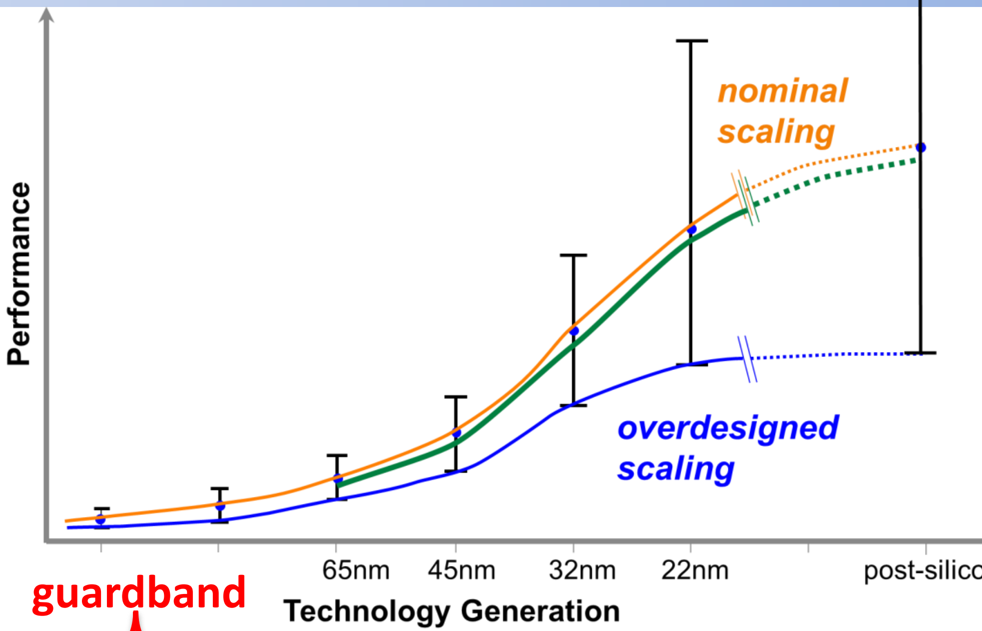
Table 12. SDR Timing Specifications

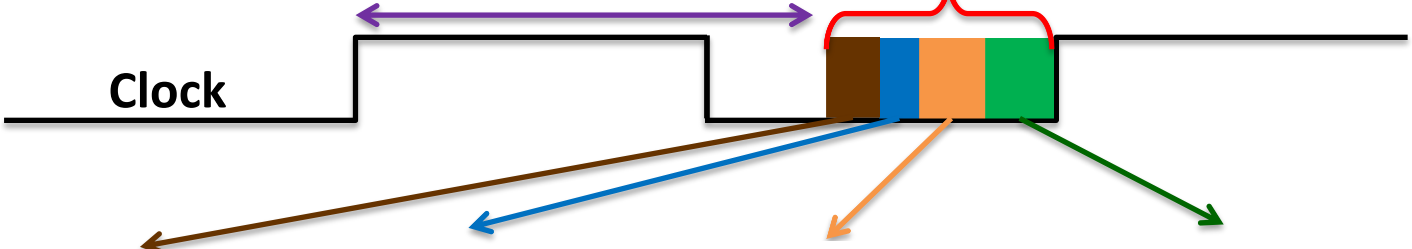
Symbol	Characteristic	Symbol	Min	Max	Unit	Notes
	Frequency of Operation		60	83.33	MHz	1
SD1	Clock Period (t_{CK})	t_{SDCK}	12	16.67	ns	2
SD3	Pulse Width High (t_{CKH})	t_{SDCKH}	0.45	0.55	SD_CLK	3
SD4	Pulse Width Low (t_{CKL})	t_{SDCKL}	0.45	0.55	SD_CLK	3
SD5	Address, SD_CKE, $\overline{SD\_CAS}$, $\overline{SD\_RAS}$, $\overline{SD\_WE}$, SD_BA, $\overline{SD\_CS}$[1:0] - Output Valid (t_{CMV})	$t_{SDCHACV}$	—	0.5 × SD_CLK + 1.0	ns	
SD6	Address, SD_CKE, $\overline{SD\_CAS}$, $\overline{SD\_RAS}$, $\overline{SD\_WE}$, SD_BA, $\overline{SD\_CS}$[1:0] - Output Hold (t_{CMH})	$t_{SDCHACI}$	2.0	—	ns	
SD7	SD_SDR_DQS Output Valid (t_{DQSOV})	t_{DQSOV}	—	Self timed	ns	4
SD8	SD_DQS[3:2] input setup relative to SD_CLK (t_{DQSIS})	$t_{DQVSDCH}$	0.25 × SD_CLK	0.40 × SD_CLK	ns	5
SD9	SD_DQS[3:2] input hold relative to SD_CLK (t_{DQSIH})	$t_{DQISDCH}$	Does not apply. 0.5 SD_CLK fixed width.			6
SD10	Data (D[31:0]) Input Setup relative to SD_CLK (reference only) (t_{DIS})	t_{DVSDCH}	0.25 × SD_CLK	—	ns	7
SD11	Data Input Hold relative to SD_CLK (reference only) (t_{DIH})	t_{DISDCH}	1.0	—	ns	
SD12	Data (D[31:0]) and Data Mask(SD_DQM[3:0]) Output Valid (t_{DV})	$t_{SDCHDMV}$	—	0.75 × SD_CLK + 0.5	ns	

From Chiseled Objects to Molecular Assemblies



Courtesy A. Asenov
Univ. of Glasgow

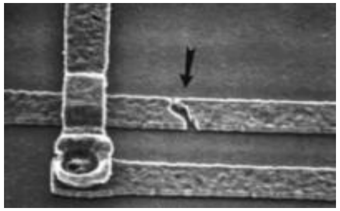




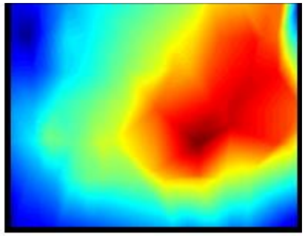
actual circuit delay

guardband

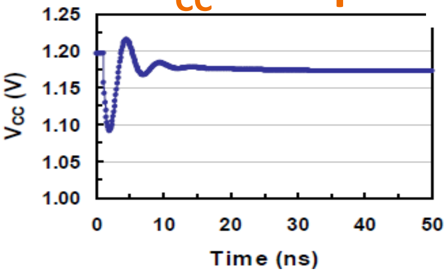
Aging



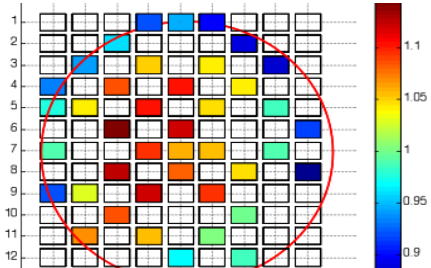
Temperature



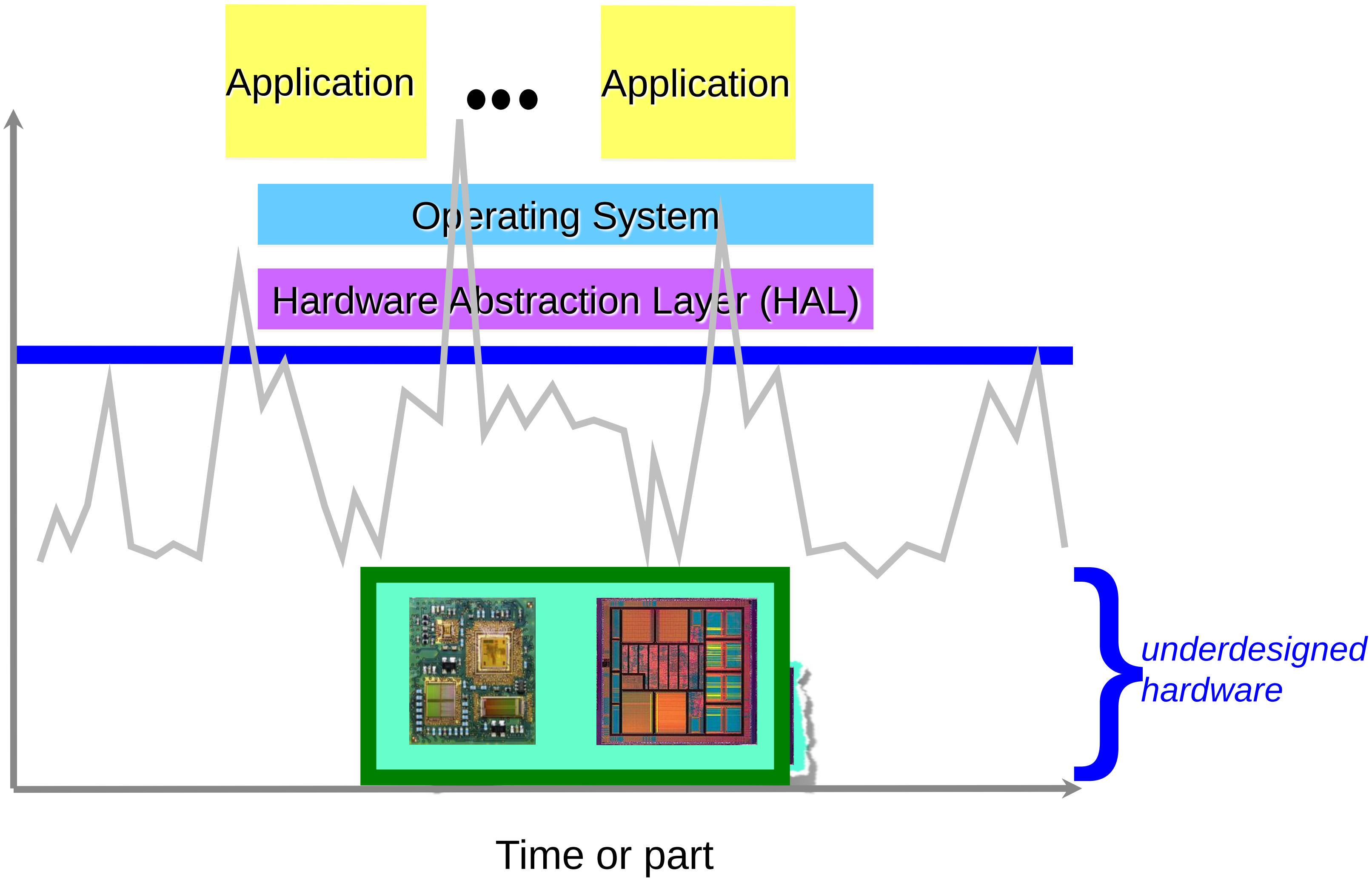
V_{CC} Droop



Across-wafer Frequency



What if?



Application

•••

Application

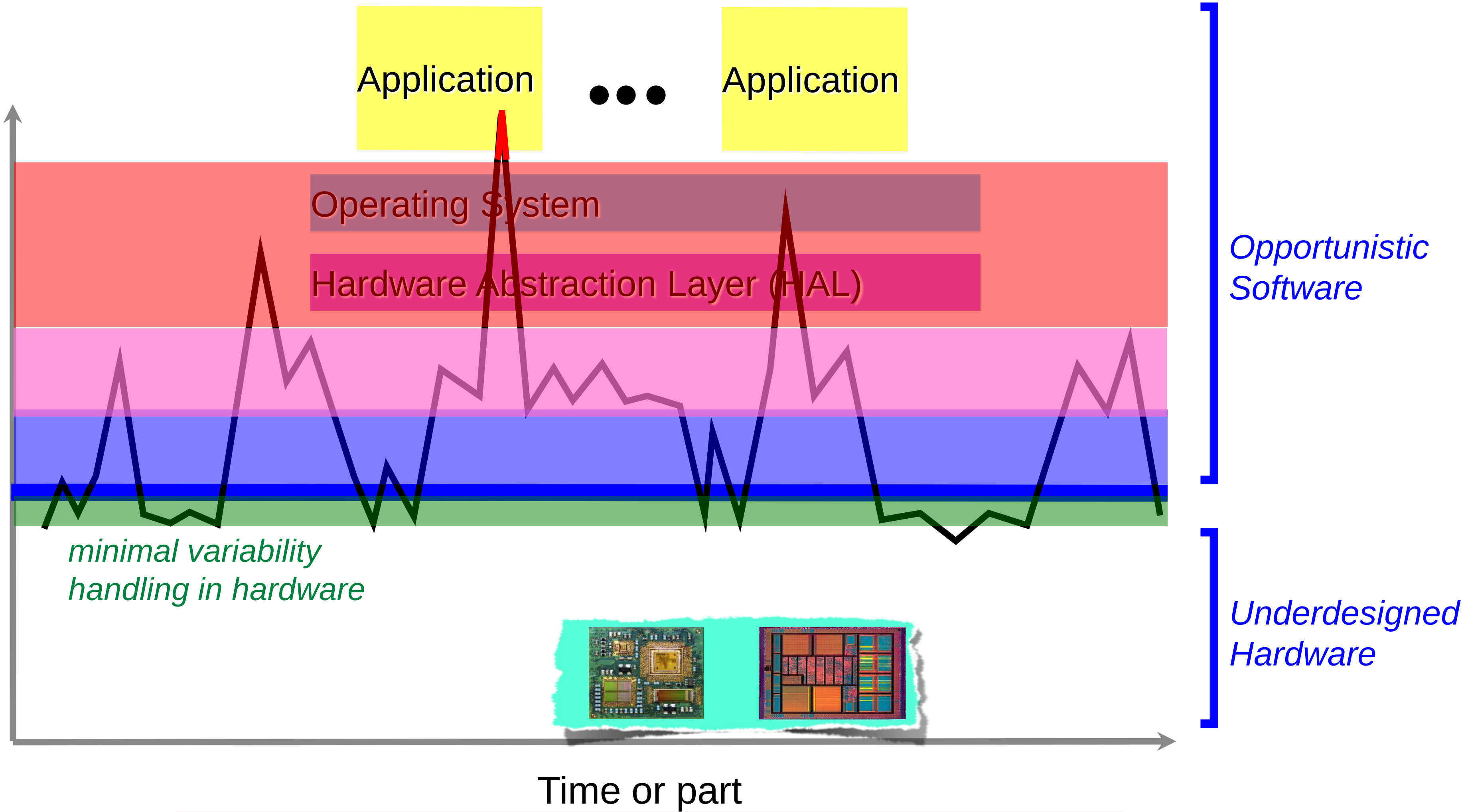
Operating System

Hardware Abstraction Layer (HAL)

underdesigned hardware

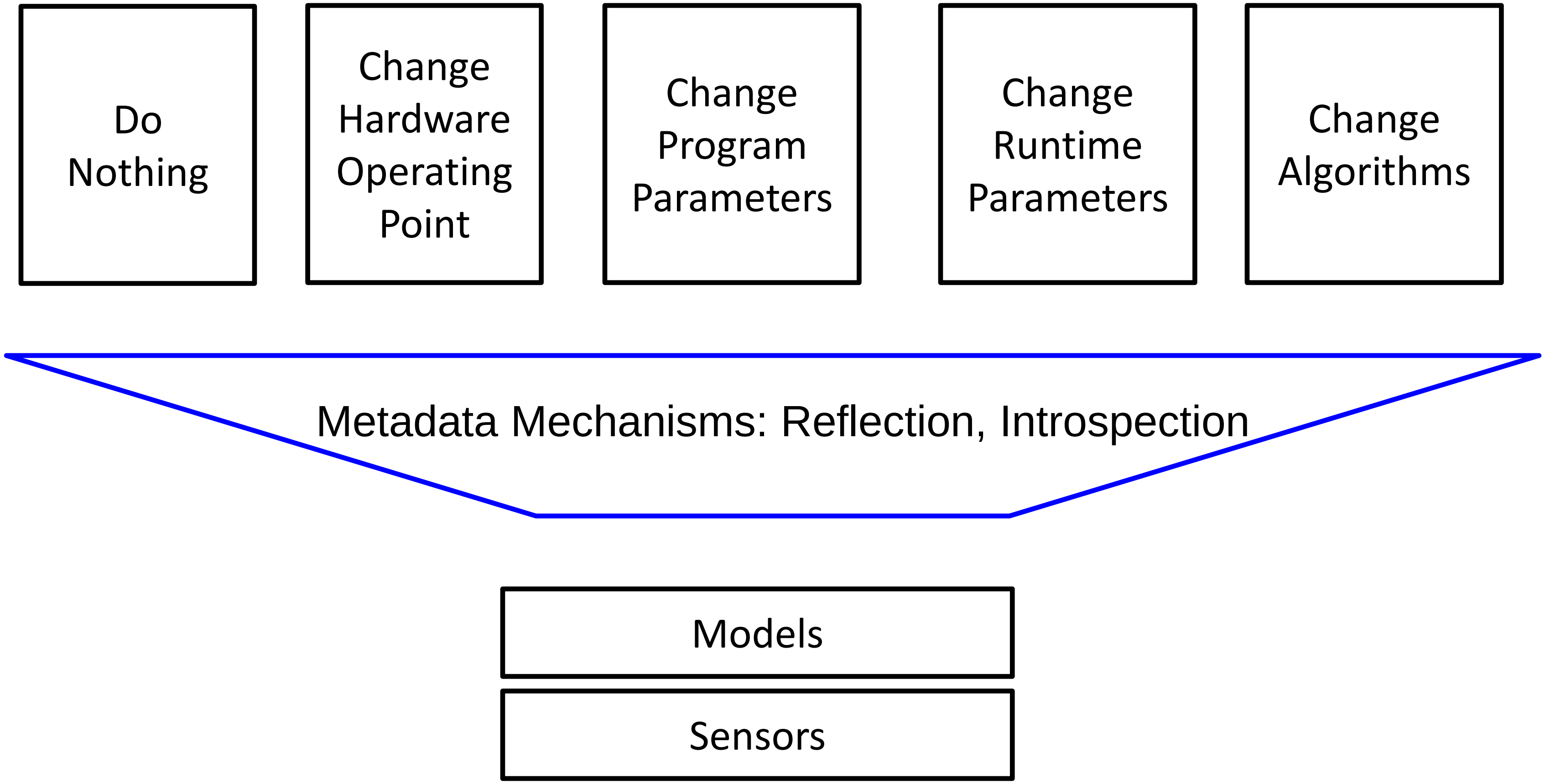
Time or part

New Hardware-Software Interface..



Builds upon a 50-year rich research in fault tolerance.

UNO Computing Machines Seek Opportunities based on Sensing Results



Do Nothing

Change Hardware Operating Point

Change Program Parameters

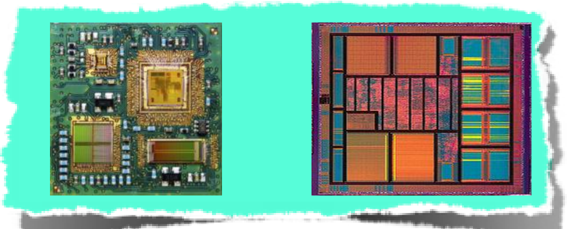
Change Runtime Parameters

Change Algorithms

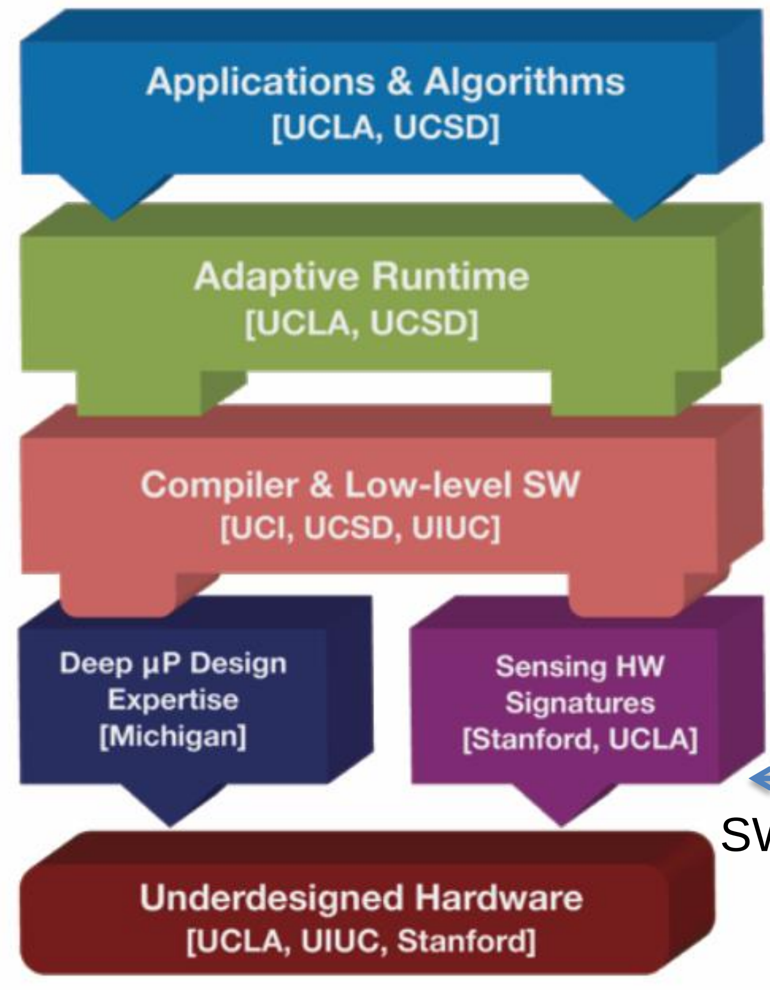
Metadata Mechanisms: Reflection, Introspection

Models

Sensors



Building Machines that leverage move from Crash & Recover to Sense & Adapt

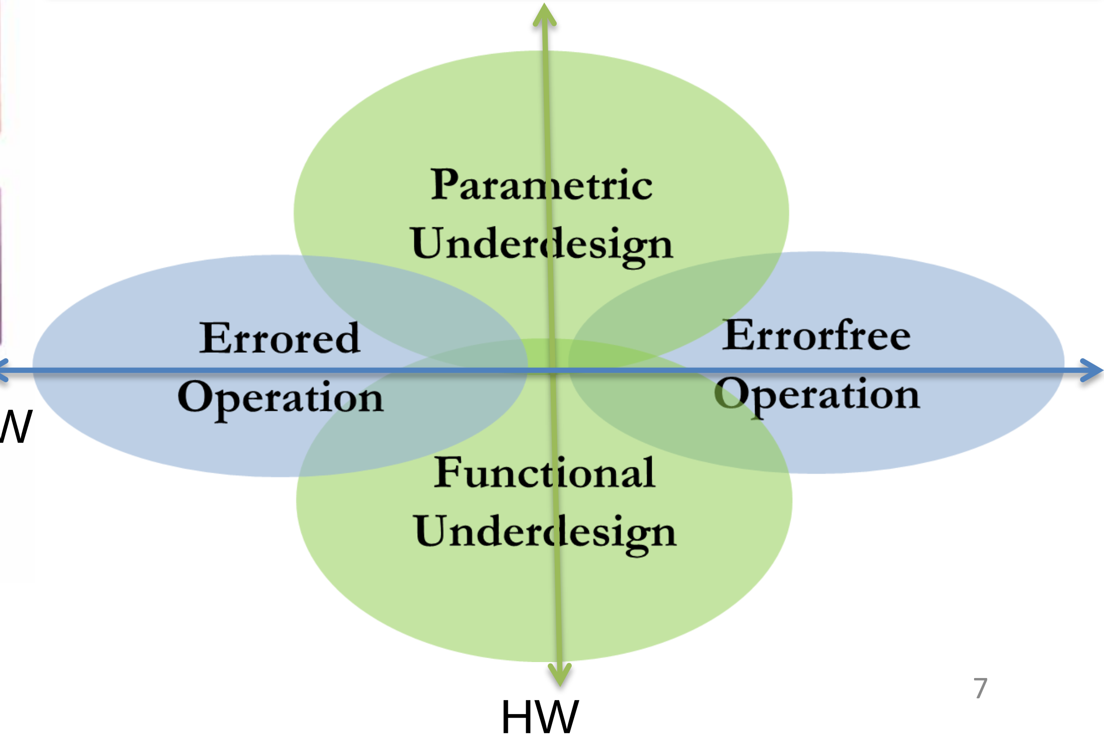


Machines that consist of parts with variations in performance, power and reliability

Machines that incorporate sensing circuits

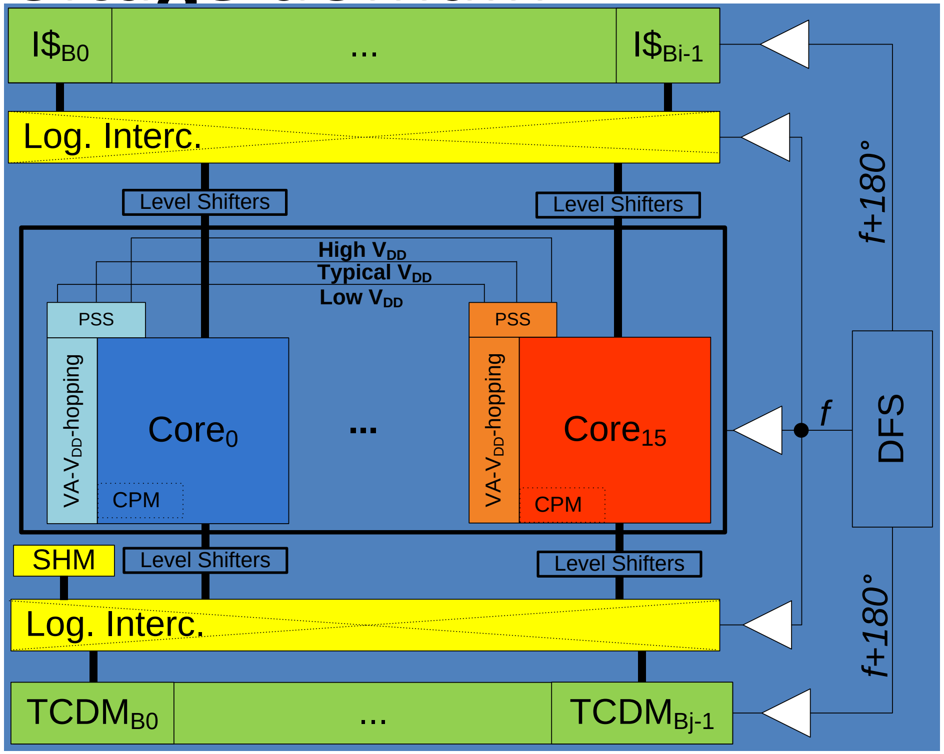
Machines w/ interfaces to change ongoing computation & structures

New machine models: QOS or Relaxed Reliability parts



Example: Procedure Hopping in Clustered CPU, Each core with its voltage domain

- Statically characterize procedure for PLV
- A core increases voltage if monitored delay is high
- A procedure hops from one core to another if its voltage variation is high
- Less 1% cycle overhead in EEMBC.



V_{DD} = 0.81V

f_0 862	f_1 909	f_2 870	f_3 847
f_4 826	f_5 855	f_6 877	f_7 893
f_8 820	f_9 826	f_{10} 909	f_{11} 847
f_{12} 901	f_{13} 917	f_{14} 847	f_{15} 901

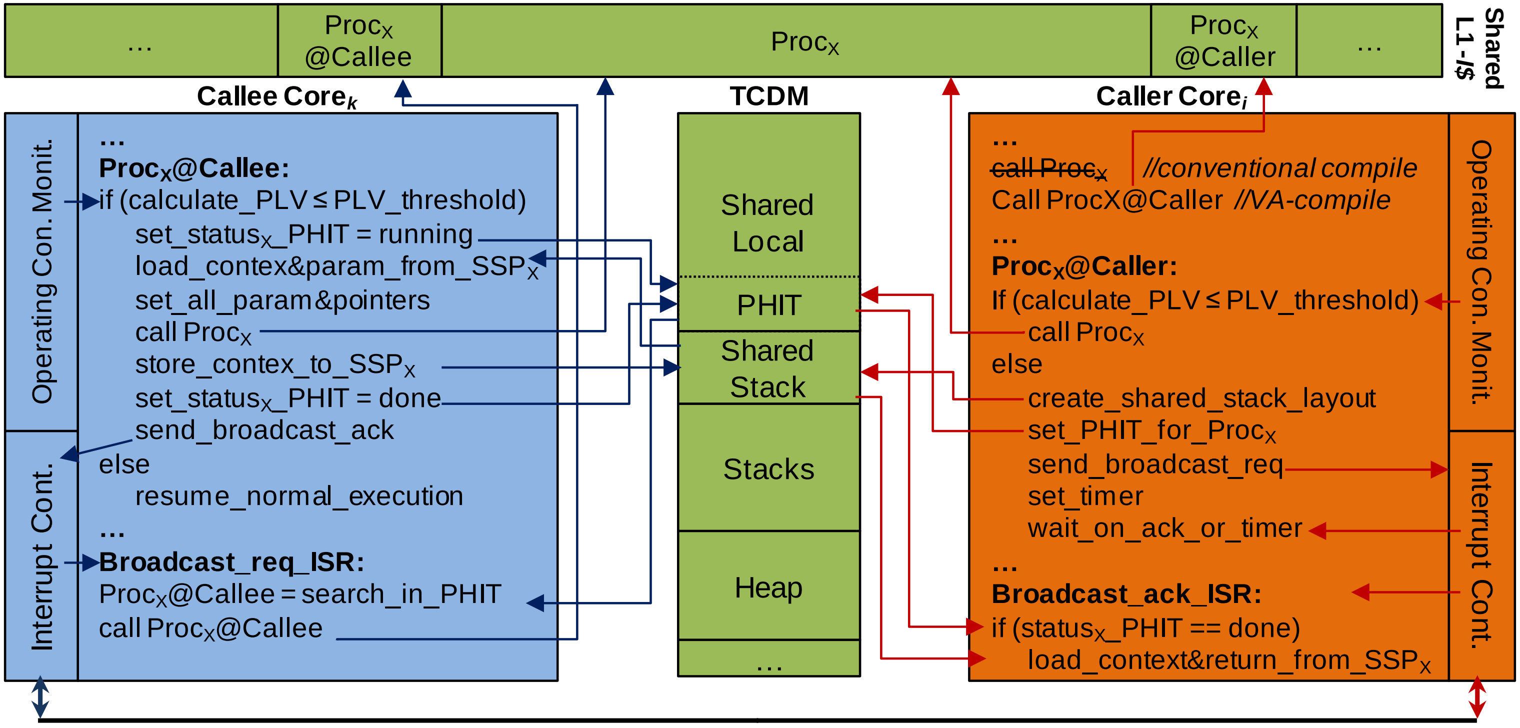
V_{DD} = 0.99V

f_0 1408	f_1 1389	f_2 1408	f_3 1370
f_4 1370	f_5 1408	f_6 1408	f_7 1408
f_8 1370	f_9 1370	f_{10} 1389	f_{11} 1370
f_{12} 1408	f_{13} 1408	f_{14} 1389	f_{15} 1389

VA-V_{DD}-Hopping=(0.81V, 0.99V)

f_0 862	f_1 909	f_2 870	f_3 847
f_4 1370	f_5 855	f_6 877	f_7 893
f_8 1370	f_9 1370	f_{10} 909	f_{11} 847
f_{12} 901	f_{13} 917	f_{14} 847	f_{15} 901

HW/SW Collaborative Architecture to Support Intra-cluster Procedure Hopping

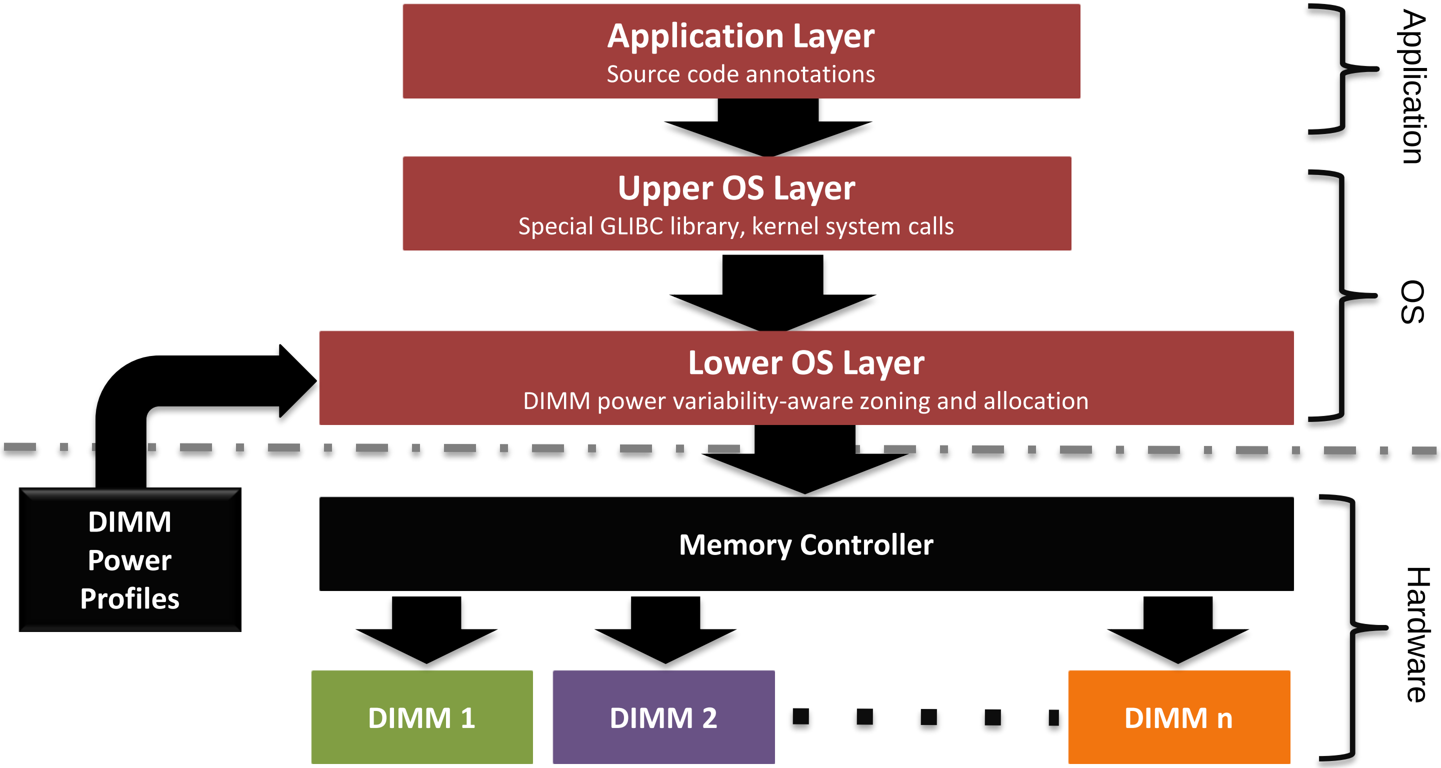


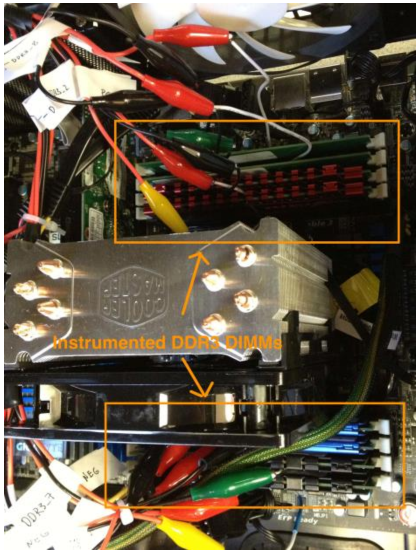
```
...
Proc$_X$@Callee:
if (calculate_PLV ≤ PLV_threshold)
    set_status$_X$_PHIT = running
    load_contex&param_from_SSP$_X$
    set_all_param&pointers
    call Proc$_X$
    store_contex_to_SSP$_X$
    set_status$_X$_PHIT = done
    send_broadcast_ack
else
    resume_normal_execution
...
Broadcast_req_ISR:
Proc$_X$@Callee = search_in_PHIT
call Proc$_X$@Callee
```

```
...
~~call Proc$_X$~~    //conventional compile
Call ProcX@Caller  //VA-compile
...
Proc$_X$@Caller:
If (calculate_PLV ≤ PLV_threshold)
    call Proc$_X$
else
    create_shared_stack_layout
    set_PHIT_for_Proc$_X$
    send_broadcast_req
    set_timer
    wait_on_ack_or_timer
...
Broadcast_ack_ISR:
if (status$_X$_PHIT == done)
    load_context&return_from_SSP$_X$
```

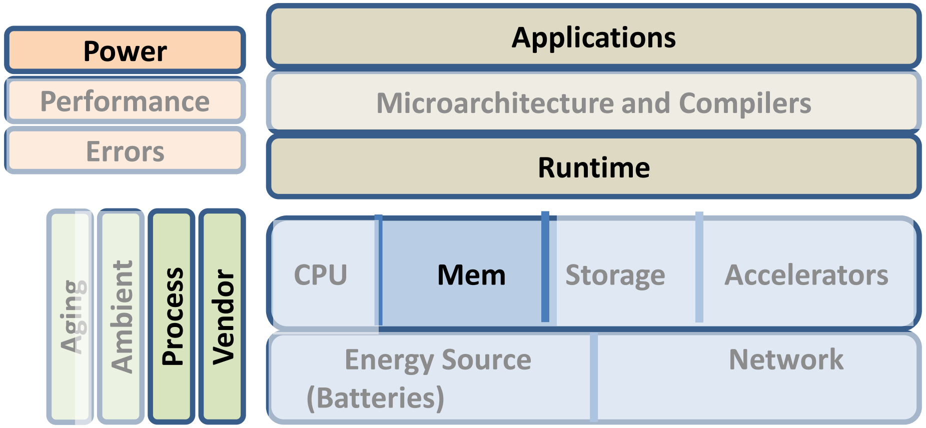
- The code is easily accessible via the shared-L1 *I$.*
- The data and parameters are passed through the shared stack in TCDM.
- A procedure hopping information table (PHIT) keeps the status for a migrated procedure.

ViPZonE: Exploiting Memory Power Variability

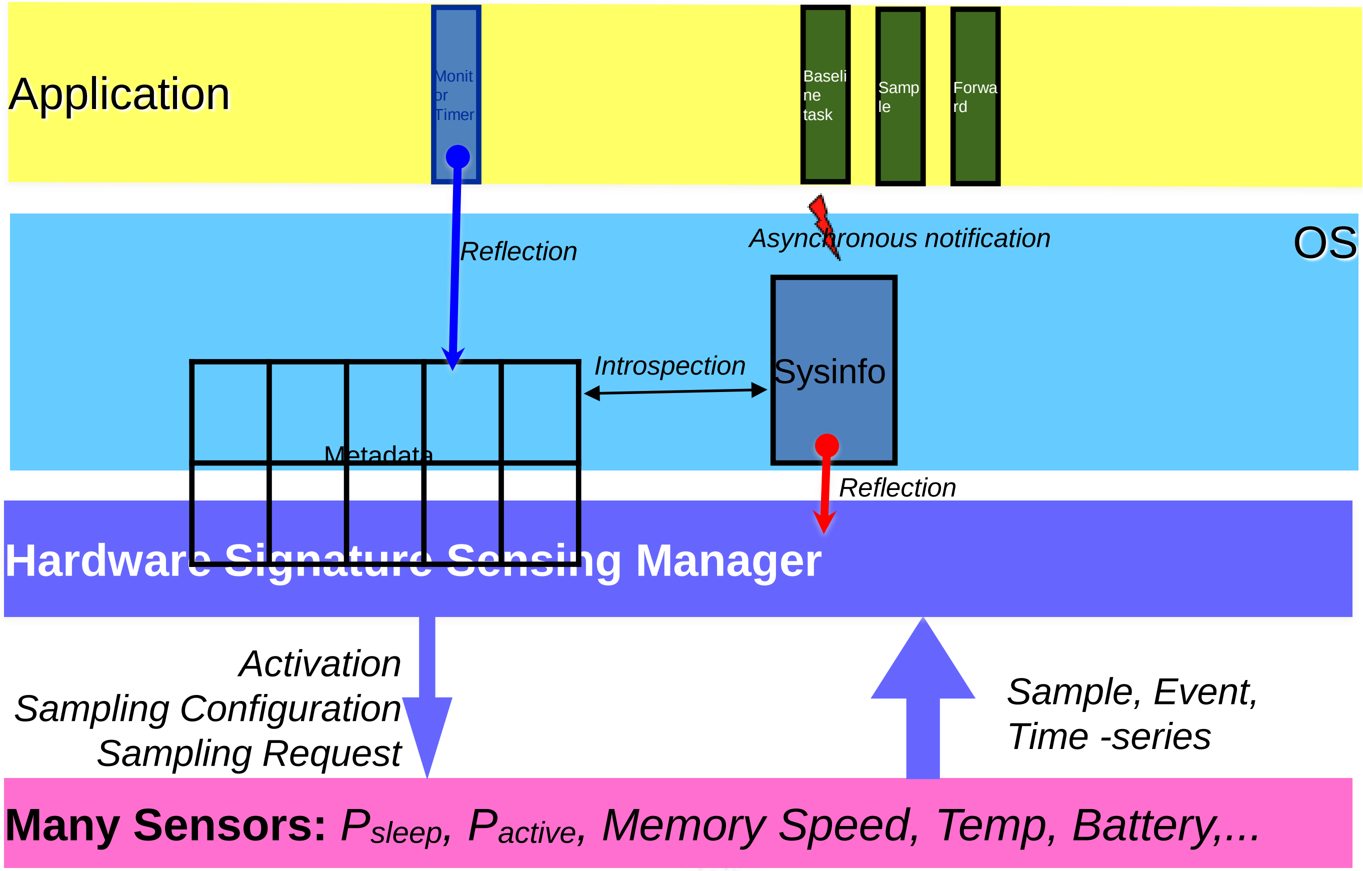




- App developers can optimize <u>dynamic allocations for reduced power</u>
- Linux + Glibc implementation



Example: UnO Stack for Duty-cycled Sensors



Application

Monitor Timer

Baseline task

Sample

Forward

Reflection

Asynchronous notification

OS

Introspection

Sysinfo

Metadata

Reflection

Hardware Signature Sensing Manager

Activation
Sampling Configuration
Sampling Request

Sample, Event,
Time -series

Many Sensors: *P_{sleep}, P_{active}, Memory Speed, Temp, Battery,...*

A

```
module SenseAndForward {
provides energylevel LowFid<1>;
provides energylevel MidFid<2>;
provides energylevel HiFid<3>; }
{ On_event Timer
call SensorRead();
On_event LowFid
call Timer(2500);
On_event MidFid
call Timer(2000);
On_event HiFid
call Timer(1650);}
```

B

```
module SenseAndForward {
provides energylevel LowFid<1>;
provides energylevel MidFid<2>;
provides energylevel HiFid<3>; }
{ On_event Timer
call SensorRead();
On_event MonitorTimer
call SysinfoRead(&sysinfo);
If Error > Delta
call Time(DownSample);
}
```

C

```
module SenseAndForward {
provides energylevel LowFid<1>;
provides energylevel MidFid<2>;
provides energylevel HiFid<3>; }
{ On_event SysinfoChanged
call SysinfoRead;
if Error > Delta
call Timer(DownSample);}
```

RESEARCH AND ITS ORGANIZATION

GRAND CHALLENGE, QUESTIONS AND RESEARCH PROGRESS

Expedition Grand Challenge & Questions

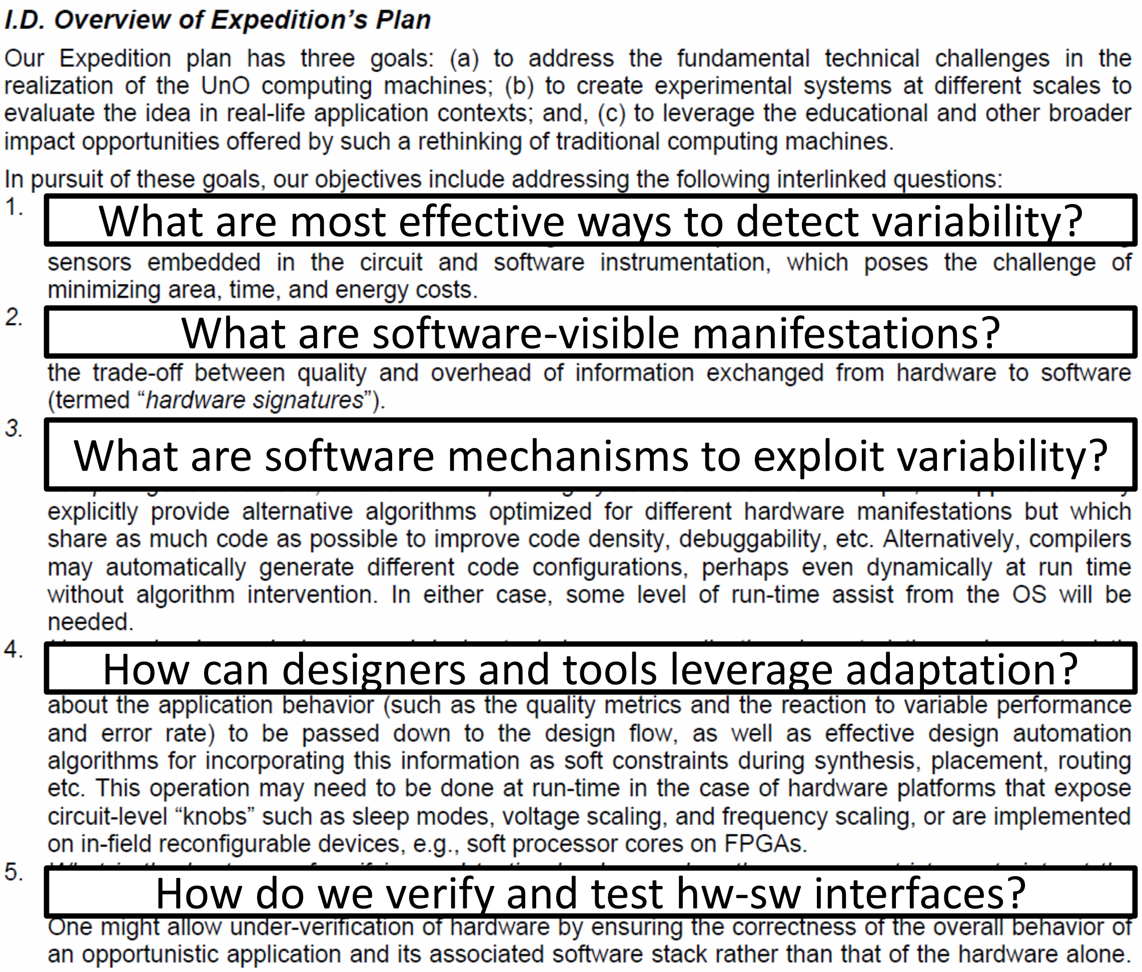
“Can microelectronic variability be controlled and utilized in building better computer systems?”

Three Goals:

a. Address fundamental technical challenges (understand the problem)

b. Create experimental systems (proof of concept prototypes)

c. Educational and broader impact opportunities to make an impact (ensure training for future talent).



I.D. Overview of Expedition’s Plan

Our Expedition plan has three goals: (a) to address the fundamental technical challenges in the realization of the UnO computing machines; (b) to create experimental systems at different scales to evaluate the idea in real-life application contexts; and, (c) to leverage the educational and other broader impact opportunities offered by such a rethinking of traditional computing machines.

In pursuit of these goals, our objectives include addressing the following interlinked questions:

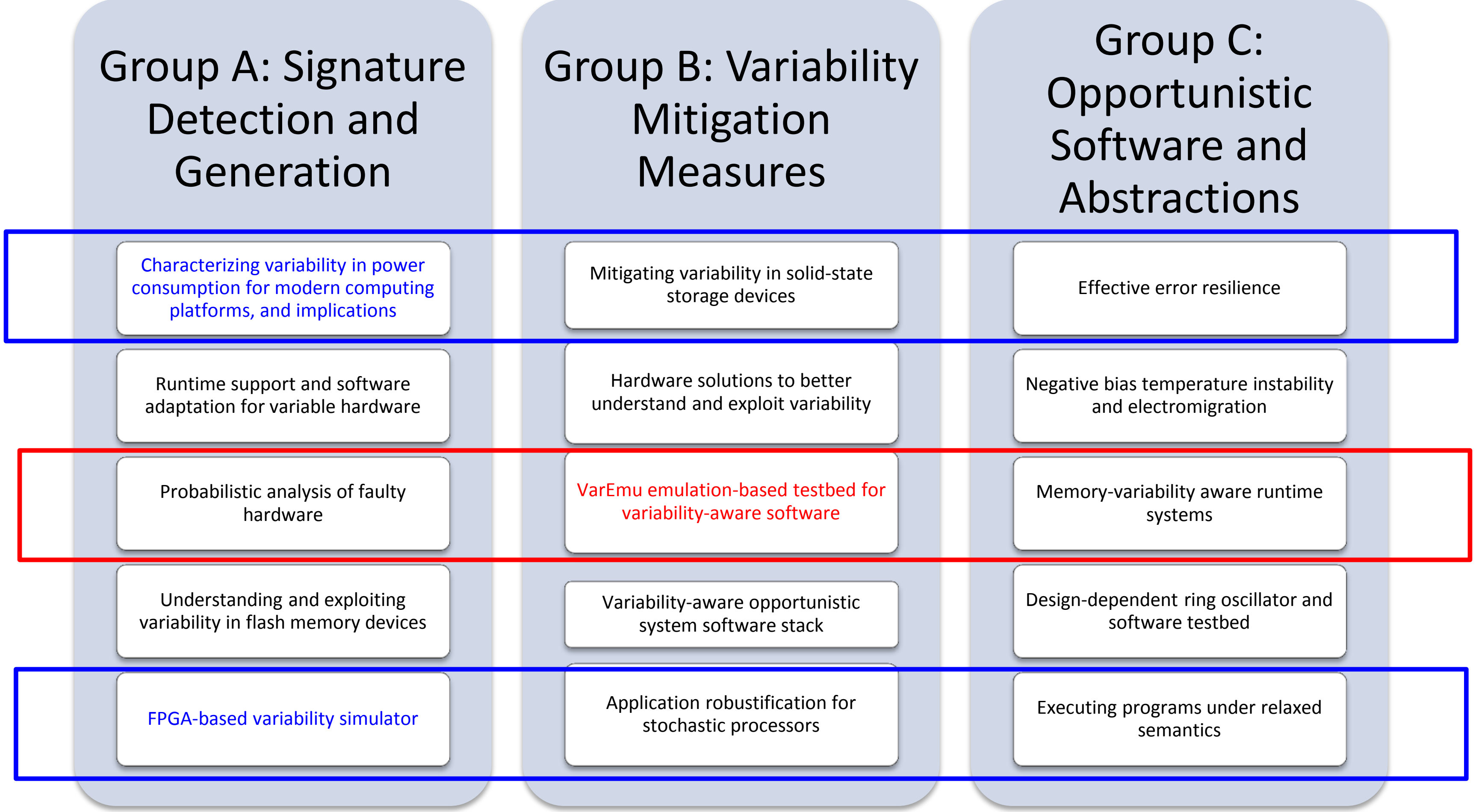
1. What are most effective ways to detect variability?
   sensors embedded in the circuit and software instrumentation, which poses the challenge of minimizing area, time, and energy costs.
2. What are software-visible manifestations?
   the trade-off between quality and overhead of information exchanged from hardware to software (termed “*hardware signatures*”).
3. What are software mechanisms to exploit variability?
   explicitly provide alternative algorithms optimized for different hardware manifestations but which share as much code as possible to improve code density, debuggability, etc. Alternatively, compilers may automatically generate different code configurations, perhaps even dynamically at run time without algorithm intervention. In either case, some level of run-time assist from the OS will be needed.
4. How can designers and tools leverage adaptation?
   about the application behavior (such as the quality metrics and the reaction to variable performance and error rate) to be passed down to the design flow, as well as effective design automation algorithms for incorporating this information as soft constraints during synthesis, placement, routing etc. This operation may need to be done at run-time in the case of hardware platforms that expose circuit-level “knobs” such as sleep modes, voltage scaling, and frequency scaling, or are implemented on in-field reconfigurable devices, e.g., soft processor cores on FPGAs.
5. How do we verify and test hw-sw interfaces?
   One might allow under-verification of hardware by ensuring the correctness of the overall behavior of an opportunistic application and its associated software stack rather than that of the hardware alone.

Research Organization

- Four thrust areas
    1. Measurement and Modeling
    2. Design Tools and Testing Methodologies
    3. Microarchitecture and Compilers
    4. Runtime Support
- Two Cross-cutting thrusts
    5. Applications and Testbeds
    6. Outreach and Education

Thrusts span teams across universities, usually in pairs.

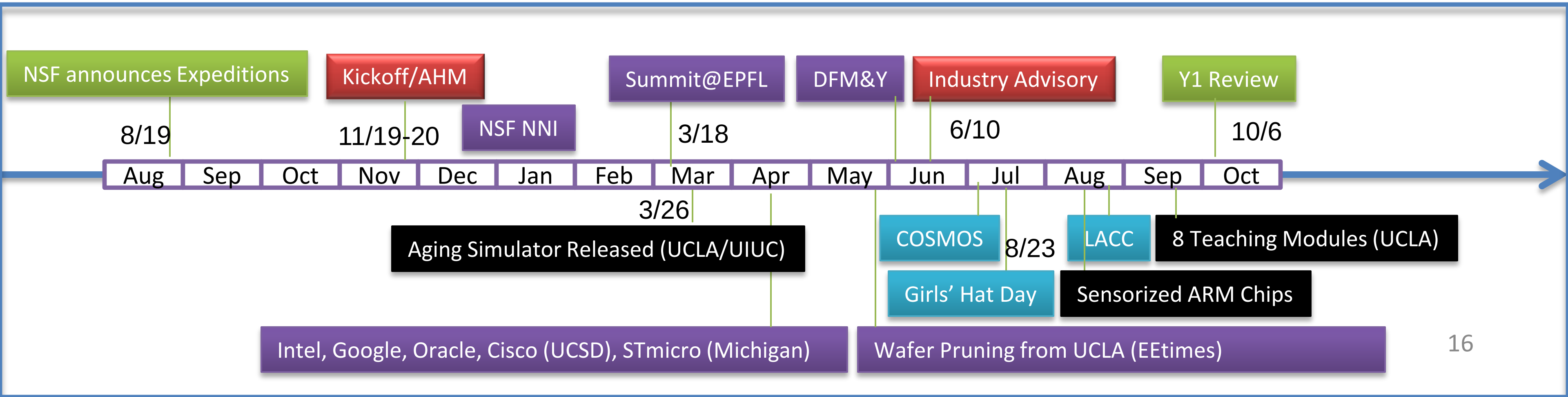
Thrusts traverse institutions on testbed vehicles seeding various projects



Group A: Signature Detection and Generation	Group B: Variability Mitigation Measures	Group C: Opportunistic Software and Abstractions
Characterizing variability in power consumption for modern computing platforms, and implications	Mitigating variability in solid-state storage devices	Effective error resilience
Runtime support and software adaptation for variable hardware	Hardware solutions to better understand and exploit variability	Negative bias temperature instability and electromigration
Probabilistic analysis of faulty hardware	VarEmu emulation-based testbed for variability-aware software	Memory-variability aware runtime systems
Understanding and exploiting variability in flash memory devices	Variability-aware opportunistic system software stack	Design-dependent ring oscillator and software testbed
FPGA-based variability simulator	Application robustification for stochastic processors	Executing programs under relaxed semantics

Two years of building an Expedition

- Kickoff, review, tape-outs and builds-ins
  - 82 peer-reviewed publications, 21% collaborative
  - 54 events/releases on variability.org/news
  - 64 presentations on variability.org/presentations
- A collaborative community
  - 15 faculty, 25 GSRs, 1 postdoc, 10+ UG, 300 K-8-12



NSF announces Expeditions

Kickoff/AHM

NSF NNI

Summit@EPFL

DFM&Y

Industry Advisory

Y1 Review

8/19

11/19-20

3/18

6/10

10/6

3/26

Aging Simulator Released (UCLA/UIUC)

COSMOS

8/23

LACC

8 Teaching Modules (UCLA)

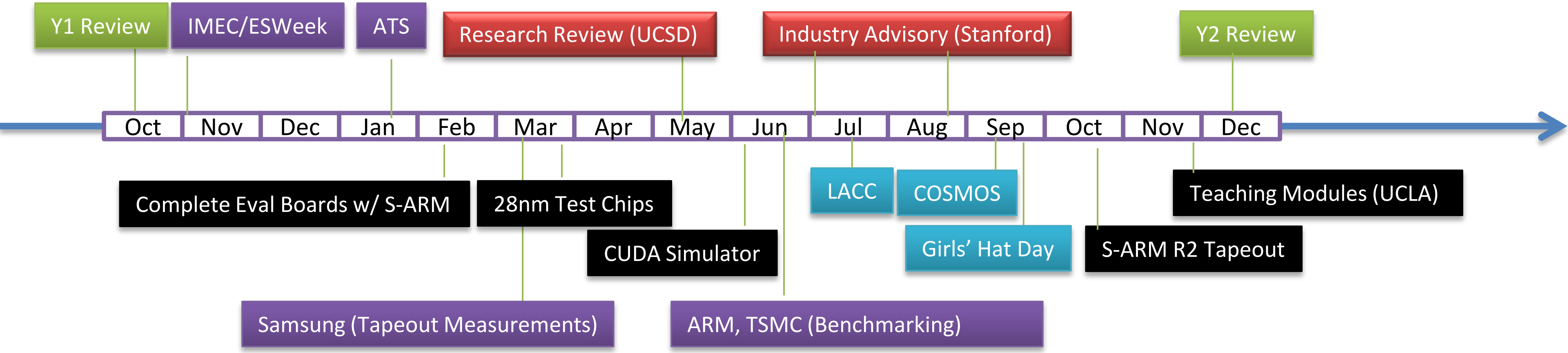
Girls' Hat Day

Sensorized ARM Chips

Intel, Google, Oracle, Cisco (UCSD), STmicro (Michigan)

Wafer Pruning from UCLA (EEtimes)

Timeline in Progress

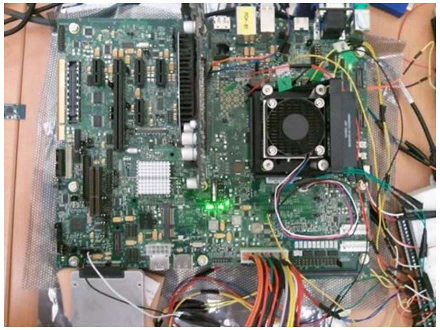


Research: From Measurements to Signatures

- Year 1 was mostly focused on characterization of variability (IC designer centric)
  - What is the extent of variation and can it be sensed? Can it be used in the HW/SW stack?
- Year 2 focused on proof-of-concept methods to use variability information (Programmer centric)
  - From observation to systematic control.
  - Can we construct **useful signatures** that can enable systematic observability (and controllability) of variation?
- Year 3 sees the two streams coming together: expanding collaborations across teams, emerging testbeds & tools.

Important Takeaways

To ensure effective use by software, we need accurate characterization (of performance, power).



1. Variability imposes a limit on how accurate the models can get to
   - Mean error ~20% + 12% due to variability for 34% overall error in Nehalem 45nm CPUs
   - 15-20% variation across 22 DIMMs
   - 20-24% read, 40-67% write variation in Flash
   - Rooted in inherent non-observability of power states.

Important Takeaways (continued)

2. Instrumentation and sensing is necessary to ensure ‘high-level’ observability of variation
   - “High enough for semantic value.” Averages may not be sufficient.

3. Sensing for delay, power, aging and degradation is feasible and indeed necessary
   - Important difference between failure prediction and error detection. Notion of static & dynamic variability management.

4. Variability *can be* leveraged in software
   - media applications, duty cycle, security sensitive applications. Notion of ‘tunable error’ and its observability criteria.

Important Takeaways (continued)

2. Instrumentation and sensing is necessary to ensure

‘hig

– e

3. S s

feas

– rror

4. V

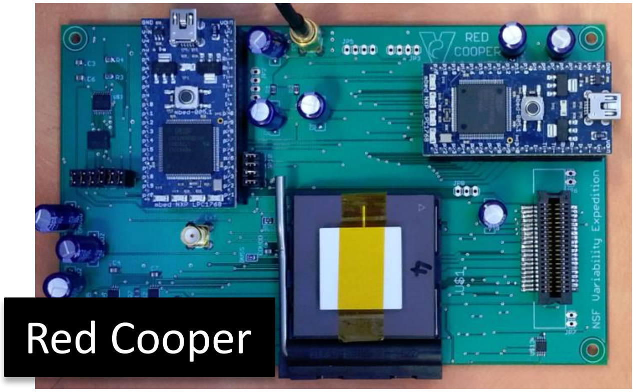
– media applications, duty cycle, security sensitive applications. Notion of ‘tunable error’ and its observability criteria.

At the end of two years, we have a complete end-to-end initial realization of an embedded system platform with sensing chip, board-level feedback, OS supporting duty-cycled tasks driven by variability, and API for such machines.

Expedition Experimental Platforms & Artifacts

- Interesting and unique challenges in building research testbeds that drive our explorations
  - – Mocks up don’t go far since variability is at the heart of microelectronic scaling. Need platforms that capture *scaling* and *integration* aspects.
- Testbeds to observe (Molecule, GreenLight, Ming), control (Oven, ERSA)


Ming the Merciless

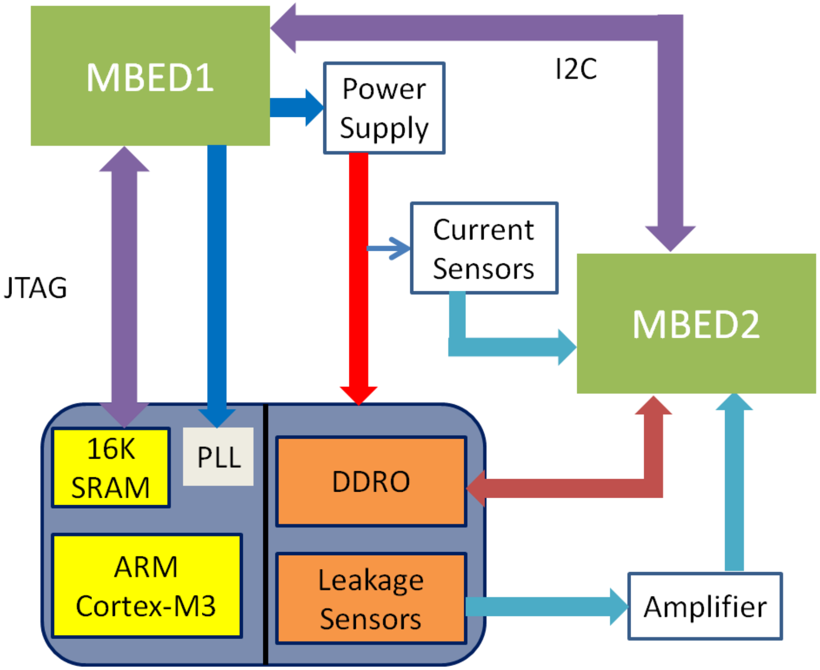

Red Cooper

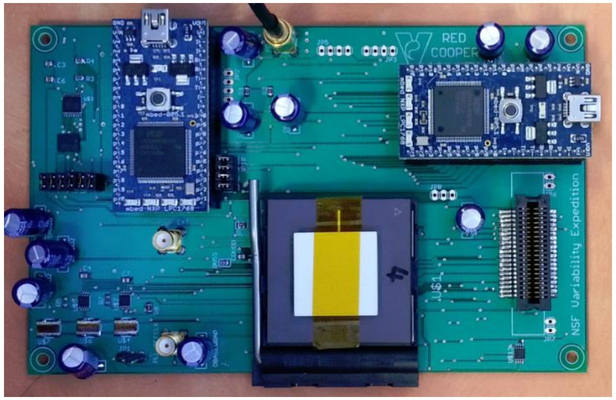

ERSA@BEE3


Molecule

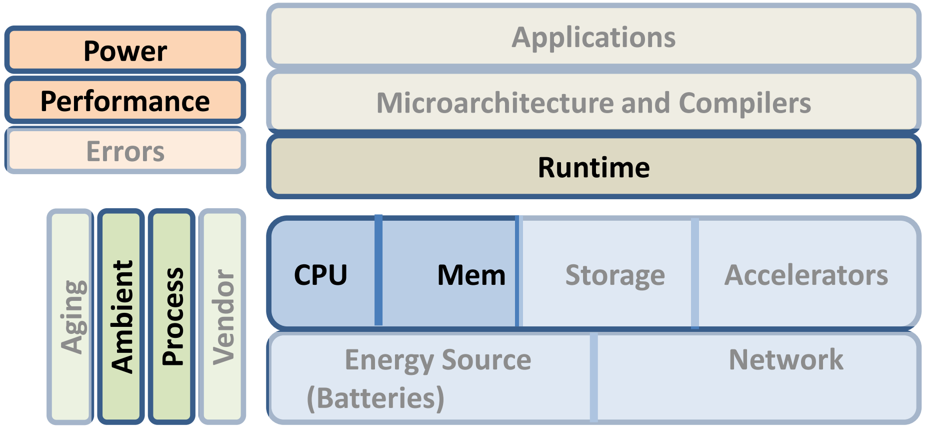
Red Cooper Testbed: *in-situ* visibility

- Customized chip with processor + speed/leakage sensors available since April 2011
- Testbed board to finish the sensor feedback loop on board

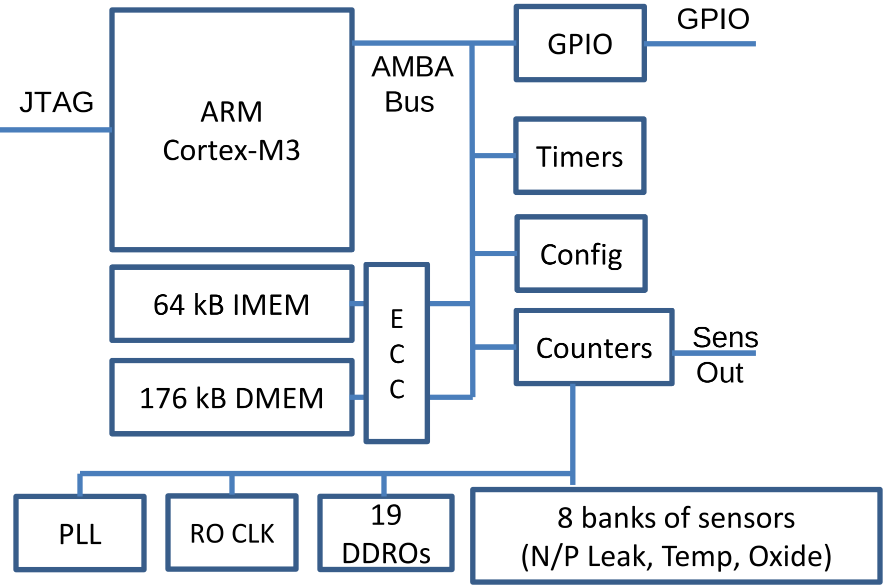




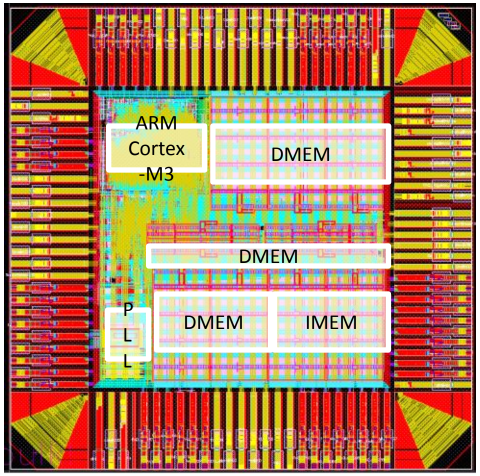
800 MHz M3, 50 packaged parts or working boards available since August 2011. ARM Cooper board available since August 2012.



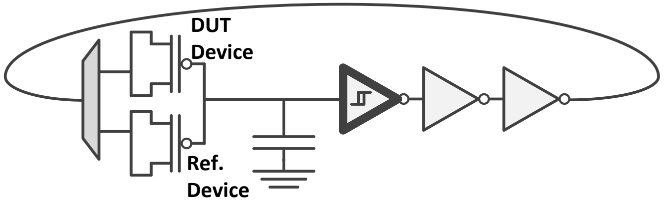
Ferrari Chip: Closing Loop On-Chip



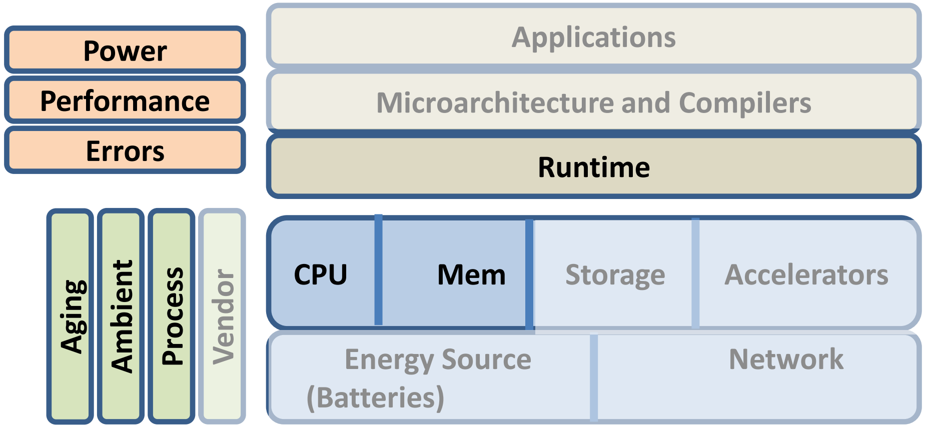
JTAG
ARM Cortex-M3
AMBA Bus
GPIO
GPIO
Timers
Config
64 kB IMEM
176 kB DMEM
ECC
Counters
Sens Out
PLL
RO CLK
19 DDROs
8 banks of sensors (N/P Leak, Temp, Oxide)



Available April 2013



- On-Chip Sensors
  - Memory mapped i/o and control
  - Leakage sensors, DDROs, temperature sensors, reliability sensors
- Better support for OS and software.



Power
Performance
Errors
Applications
Microarchitecture and Compilers
Runtime
Aging
Ambient
Process
Vendor
CPU
Mem
Storage
Accelerators
Energy Source (Batteries)
Network

From Control to Software Abstractions

Going forward

- Leon3 (Sparc) sensorized chip tapeout
- Software abstractions: PL and Runtime
  - A formal/consistent way of exposing hardware signatures
  - A full Linux software stack working
- Verification methods
  - Performance & power invariants at RT-level in the presence of variability (with TI) using probabilistic model checking
    - Similar to property checking against Monte Carlo simulations
  - Automatic generation of invariants and assertion synthesis.

Reaching out and building a community

Building our teams across 6 six sites

Building our mentors and champions

Creating early adopters

Inspiring talent

Emerging Synergies

	UCSD	UCLA	UCI	UIUC	UM	Stanford
Red Cooper	X	X			X	
Molecule		X	X			
VIPZONE		X	X			
VarEMU	X	X			X	
Ferrari	X	X			X	
ERSA/LLVM	X	X		X		X
	Software	Systems	LL Code	LL Code	Chips	Sensors

- Examples of collaborative discovery
  - Lara Dolecek working with Steve Swanson & Mitra
  - Dennis Sylvester at the center of chip/platform characterization
  - Nik Dutt, Alex Nicolau and Rakesh Kumar on code scheduling
  - Rakesh Kumar, Sorin Lerner, Ranjit Jhala on code analysis and programming language support for variability.

UnO

Underdesigned and Opportunistic Computing Machines

Thank You!