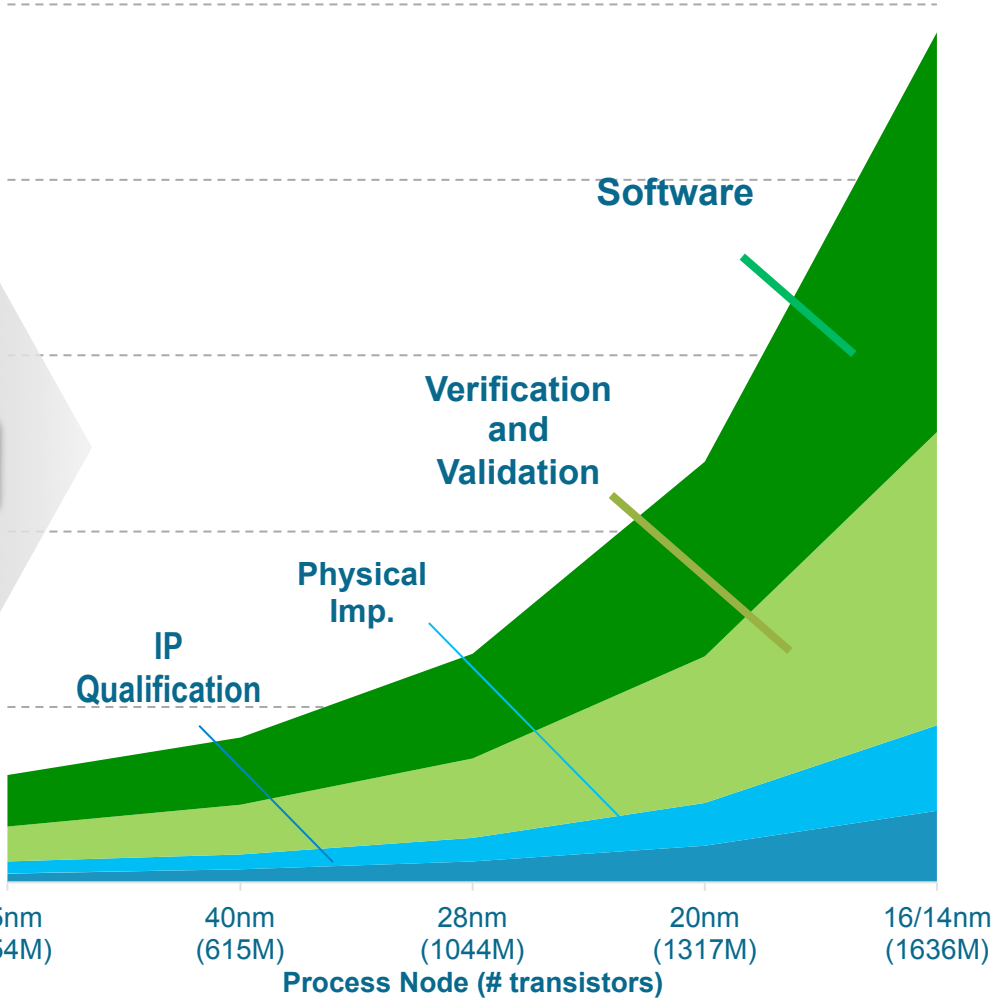
Challenging Problems in Industrial Formal Verification

Ziyad Hanna, PhD, Vice President of R&D, Fellow
Formal and Automated Verification, Cadence
Formal Methods in Computer-Aided Design
Lausanne, Switzerland
October 21-24, 2014

Software and verification driving SoC project costs

80% of overall development costs, mostly in headcount



Source: IBS July 2013

cadence

Ever-growing system development complexity

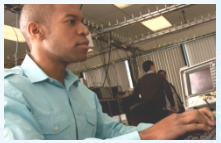
Manager

Application Developers

Firmware Developer

Architect

Integrator



System Validation Engineer

Verification Engineer

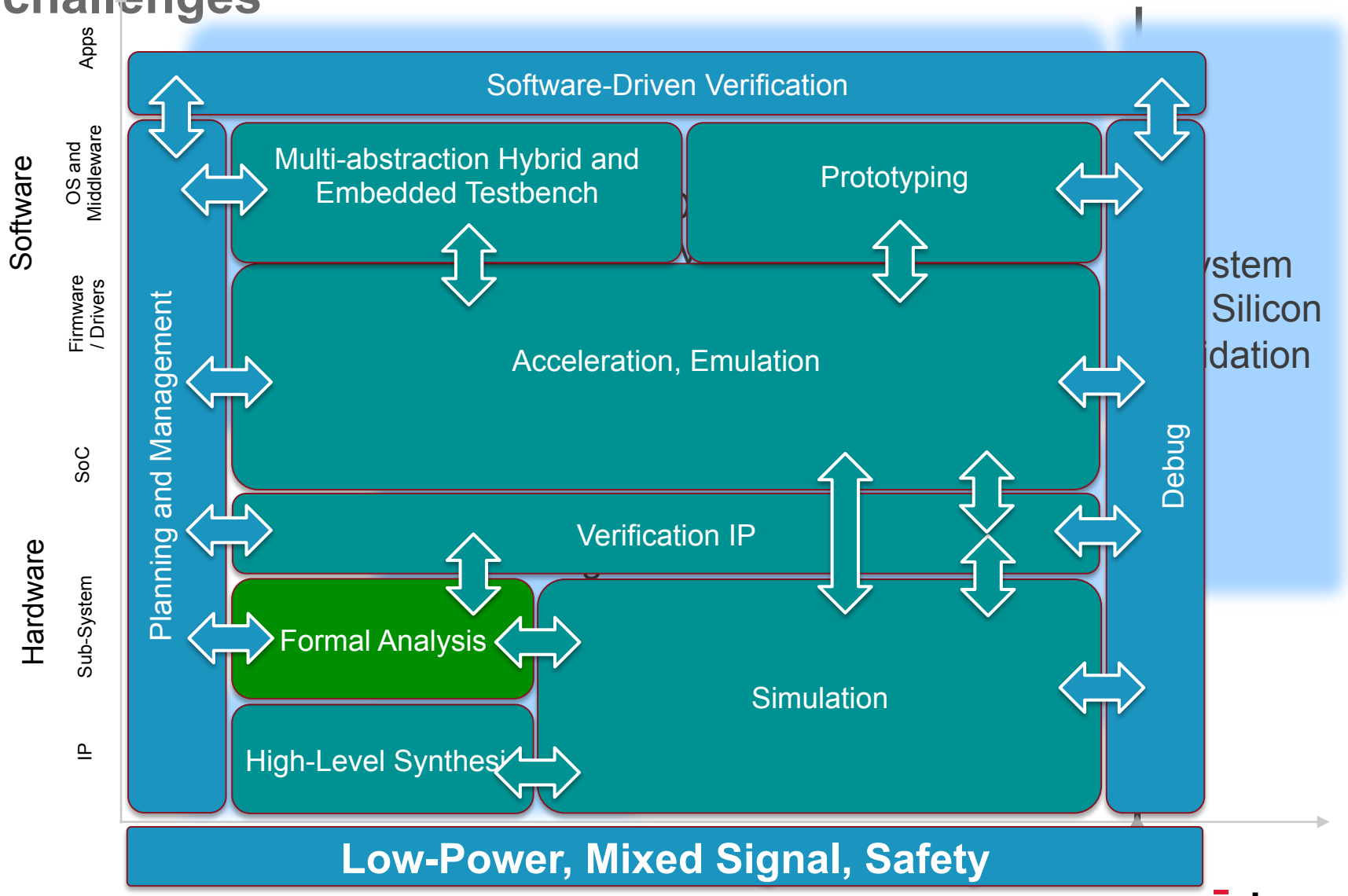
Logic Designers

Software
Hardware
Apps
OS and Middleware
Firmware / Drivers
SoC
Sub-System
IP
Time
Silicon

cadence

System Development Suite

Broad and connected solution for today's and emerging challenges



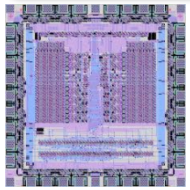
cadence

Agenda

- Formal Verification Adoption
- Where does FV fit in in the SoC design, integration, and verification flows?
- Overview of some FV applications
  - Security verification
  - Low-power verification
  - System-level verification
- Forward looking …

Complexity driving need for Formal Verification

	Low Power Dynamic power islands, functional verification, and sequential equivalence	**Multi-Processor** Complex on-chip buses, deadlock, coherency	**Security** Trusted zones, secure access, immunity from physical attacks
1990s	Use is limited to handful of very high-end ICs	Super computers	Software only
2000s	Mobile phones	Concentrated use in PC and graphics	Software only
2010 +	Use in mobile, consumer, server, graphics, IT, and computing	Wide-spread use in mobile, consumer, industrial, automotive, tablets, and mobile phones	Use in mobile, automotive, servers, gaming IC, and graphics chips



Traditional verification solutions fall short on hardest problems

Low Power	Multi-Processor	Security
Large number of possible power modes	Pre-verified modules can deadlock after integration	Register state impacts security path access
Non-deterministic transitions	Cache coherency with many heterogeneous master and slaves	Specifications of prohibited behavior
Structural changes have unexpected impact	Distributed on-chip bus implementation	

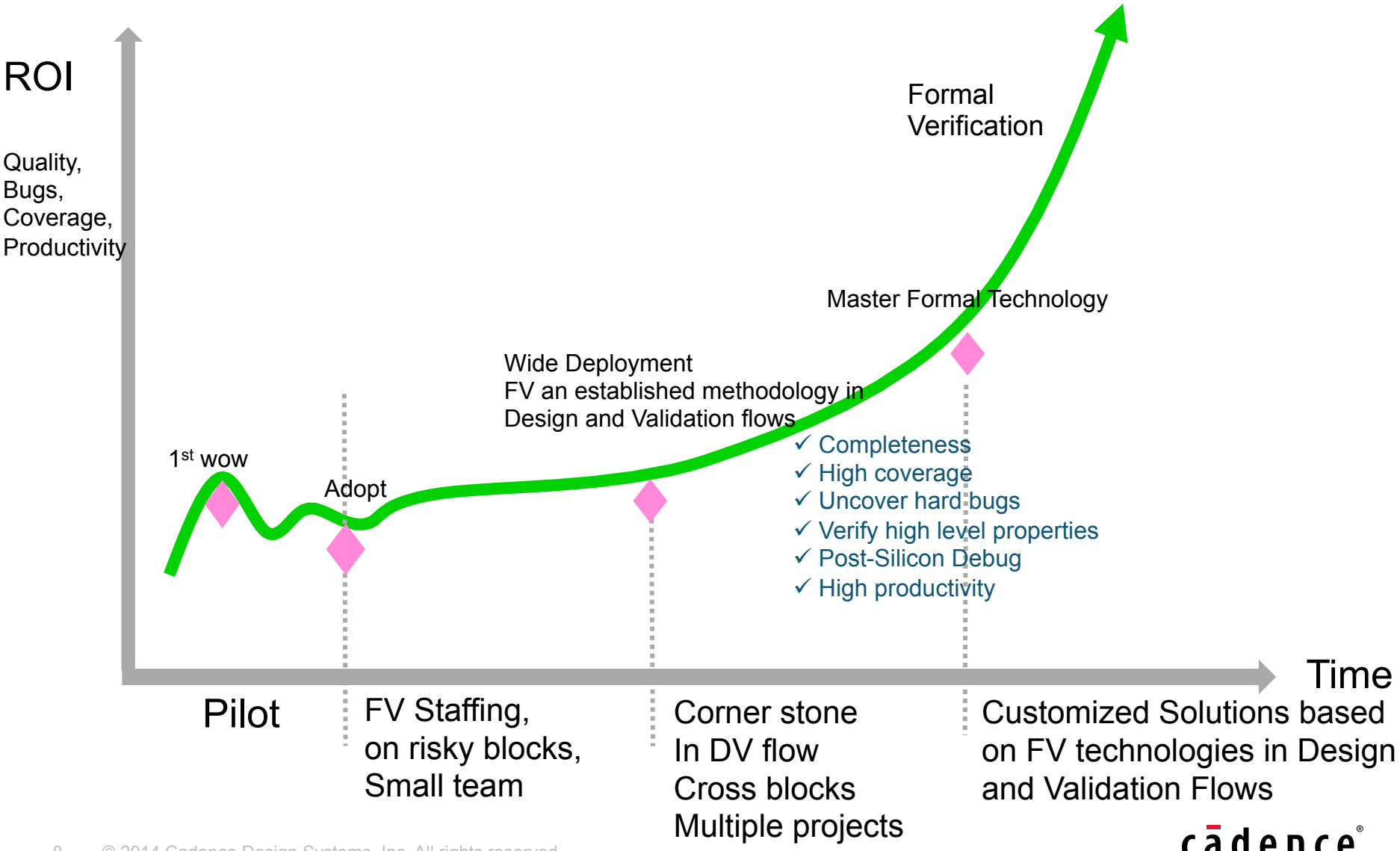
Simulation and Emulation

Simulation is empirical, can't test all possible combinations, suffers from long run times and labor-intensive debug

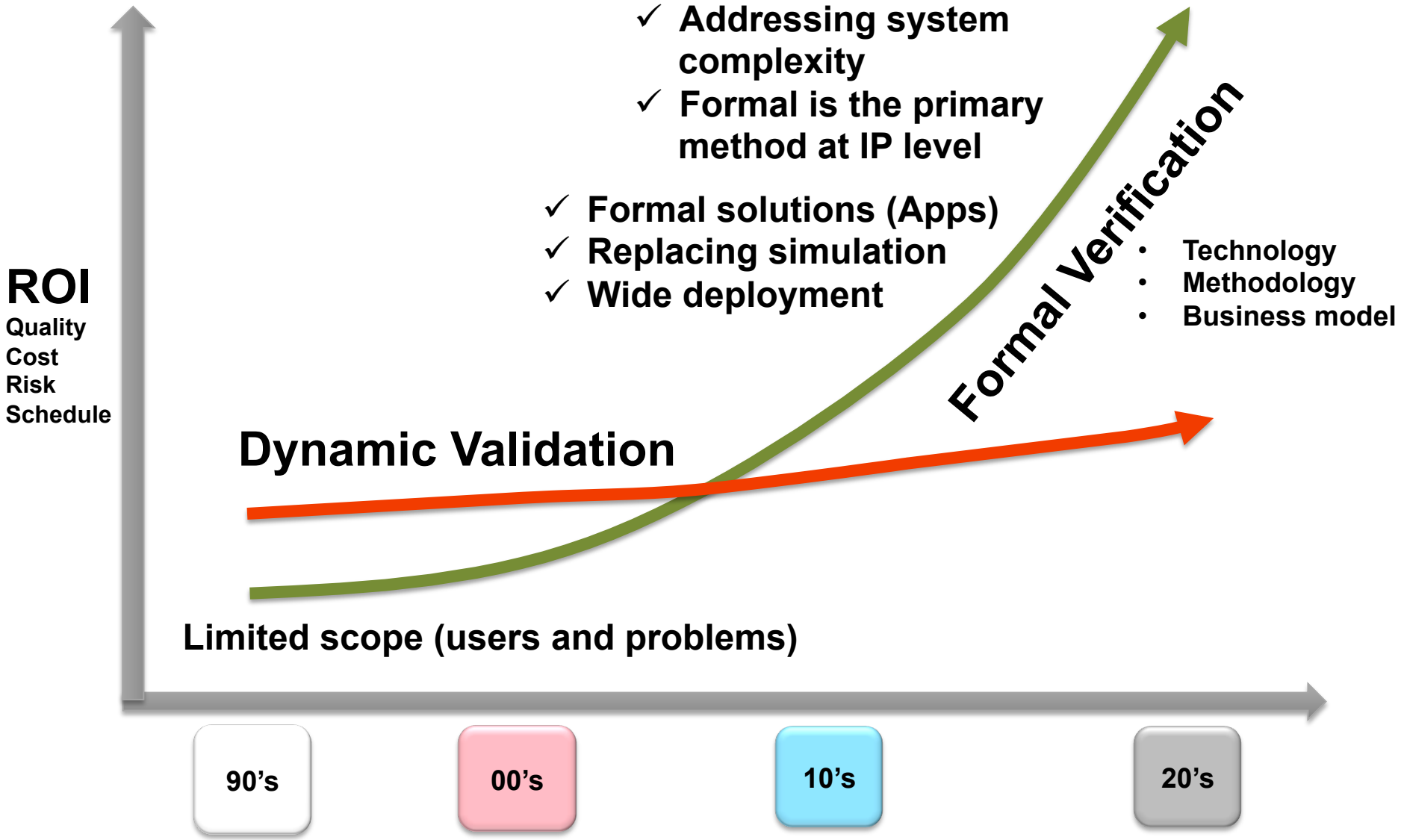
Emulation is expensive, happens too late, can't test all modes

- Previously rare and esoteric verification problems are now common to most chips
- Formal Verification to **embrace complexity** as a strategy

Formal Verification adoption phases



Disruptive Formal Verification



cadence

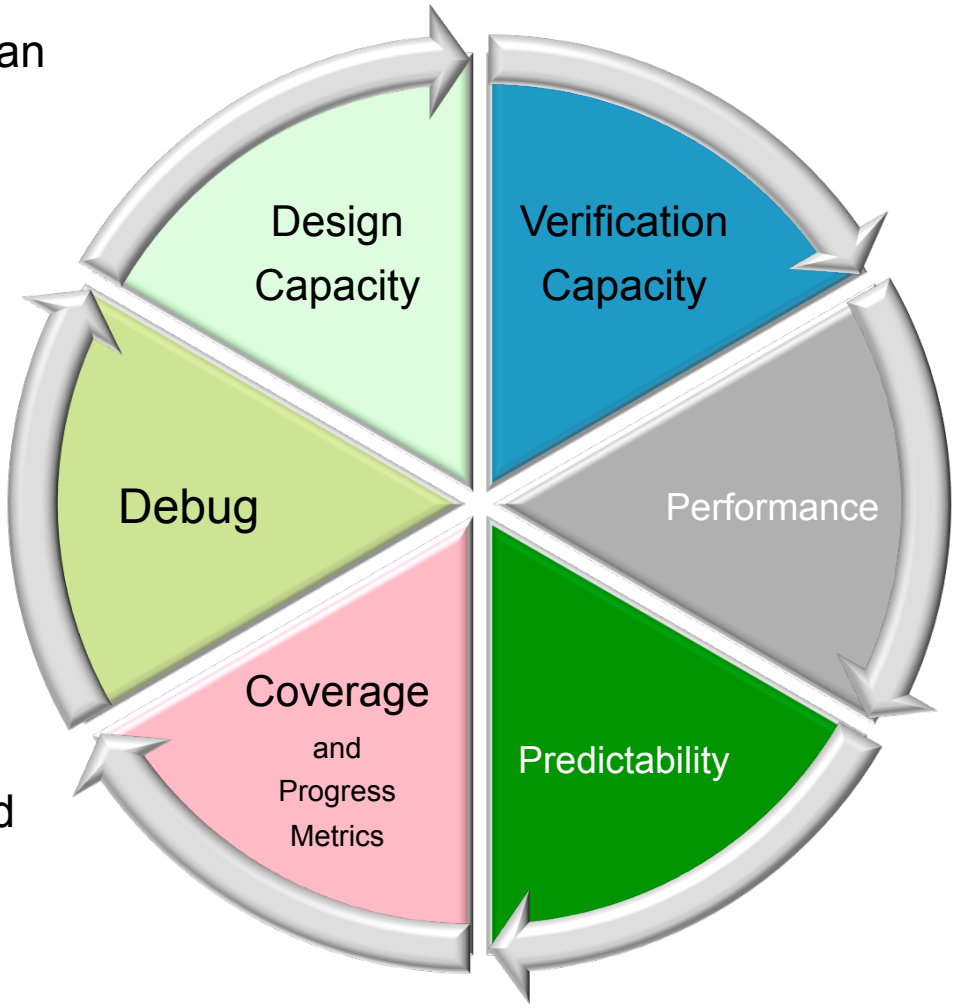
EDAC 1Q14 MSS Report
18.9 % Growth YoY in Property Checking

	2012				2013				2014	
CAE	**Q1**	**Q2**	**Q3**	**Q4**	**Q1**	**Q2**	**Q3**	**Q4**	**Q1**	**4Q/4Q**
2.3 Logic Verification	**200.2**	**226.7**	**229.3**	**249.9**	**193.8**	**223.0**	**232.3**	**275.4**	**246.250**	**8.6%**
2.3.1 RTL Simulation	103.2	102.7	106.0	116.3	110.9	107.2	120.7	132.2	108.260	7.4%
2.3.3 Hardware Verification	74.1	98.6	97.1	93.6	n/a	n/a	n/a	n/a	n/a	
2.5 Formal Verification	**29.8**	**31.7**	**37.5**	**35.4**	**40.3**	**38.9**	**39.5**	**45.1**	**35.479**	**9.6%**
2.5.1 Equivalency Checking	17.4	18.9	20.1	19.7	20.1	20.3	19.5	20.0	20.204	1.8%
2.5.2 Property Checking	12.4	12.9	17.5	15.8	20.2	18.5	20.0	25.0	15.275	**18.9 %**

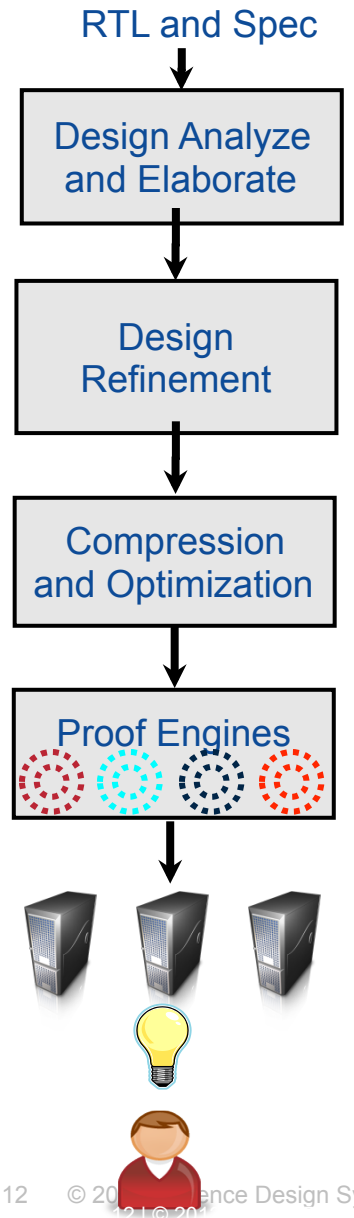
cadence

Formal Verification Technology Factors

- **Design capacity**: Size of designs that can be read and elaborated
- **Verification capacity**: Measured by the number of state variables in pruned models that FV engines can verify
- **Performance**: CPU run time needed to complete a verification task
- **Debugging**: Measured by human effort spent to complete a verification task
- **Predictability**: Where can FV be applied
- **Coverage and progress metrics**



Industrial FV system, a holistic approach



Technology Challenges

Design size, design style (e.g., latch based)
ASIC, SoC, CPU, or GPU
Evolving RTL and specification languages and standards

Ease of setup, clocking scheme
Path analysis, X analysis, design traversal
Ability to get structural and functional design information

Ability to manage verification complexity
Automatic abstractions (counters, memories)
Design or State Space Abstraction (interactive, automatic)

Fast core engines for bug hunting, proofs, visualization
Tuning for formula complexity, compression,
Optimization, decomposition, engine defaults…

Leverage parallel computing for scalability and throughput
Exploit various engine technologies, engine races,
parallel property processing …

Speed of automatically extracting waveforms
Trace concatenation, what-if analysis, visualization

Technology Frontier

Design Capacity and Complexity

Design Traversal Algorithms

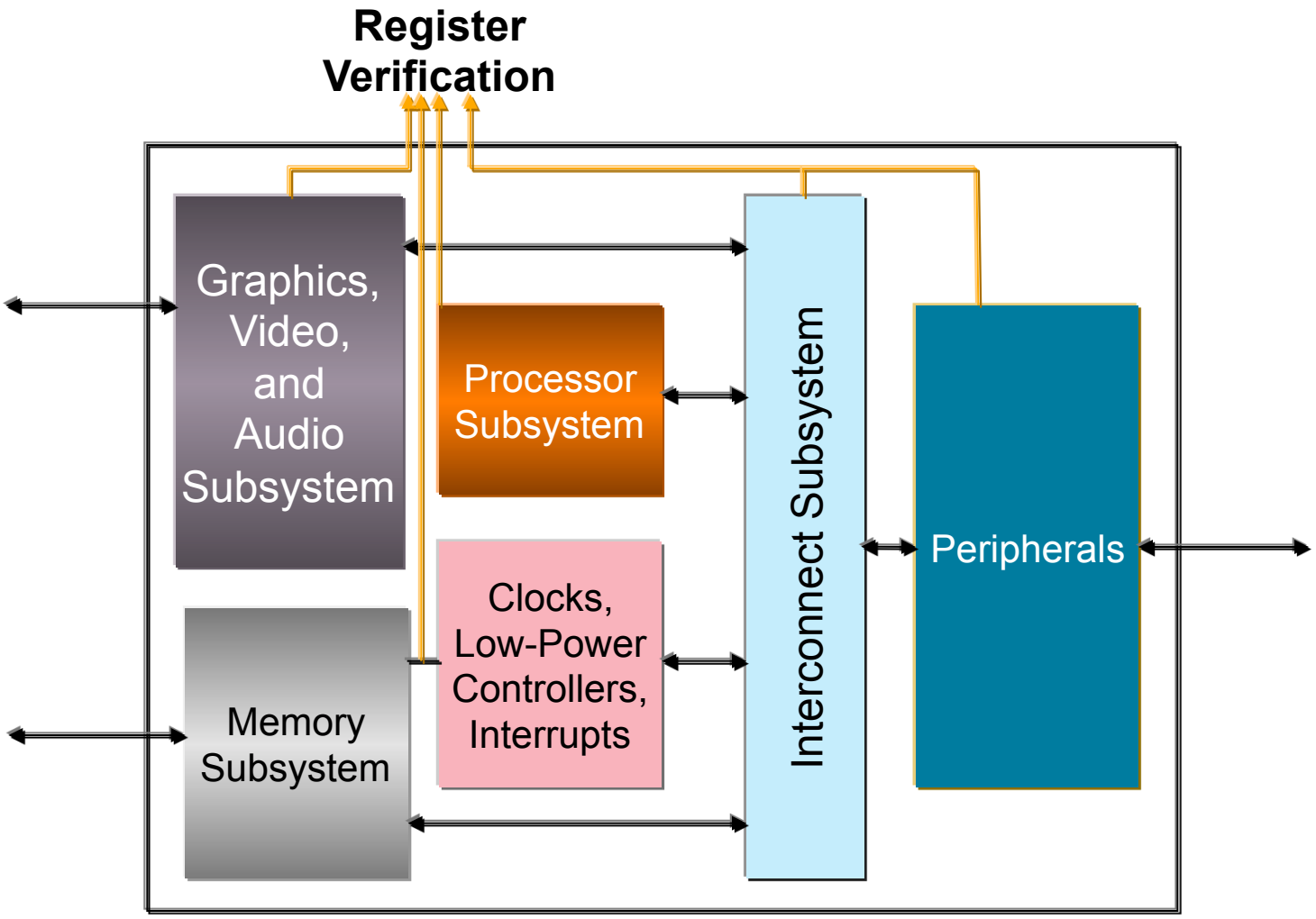
Formal Capacity

Core Engines Capacity and Performance

Parallel Computing

Interactive Analysis

SoC-level Formal Verification

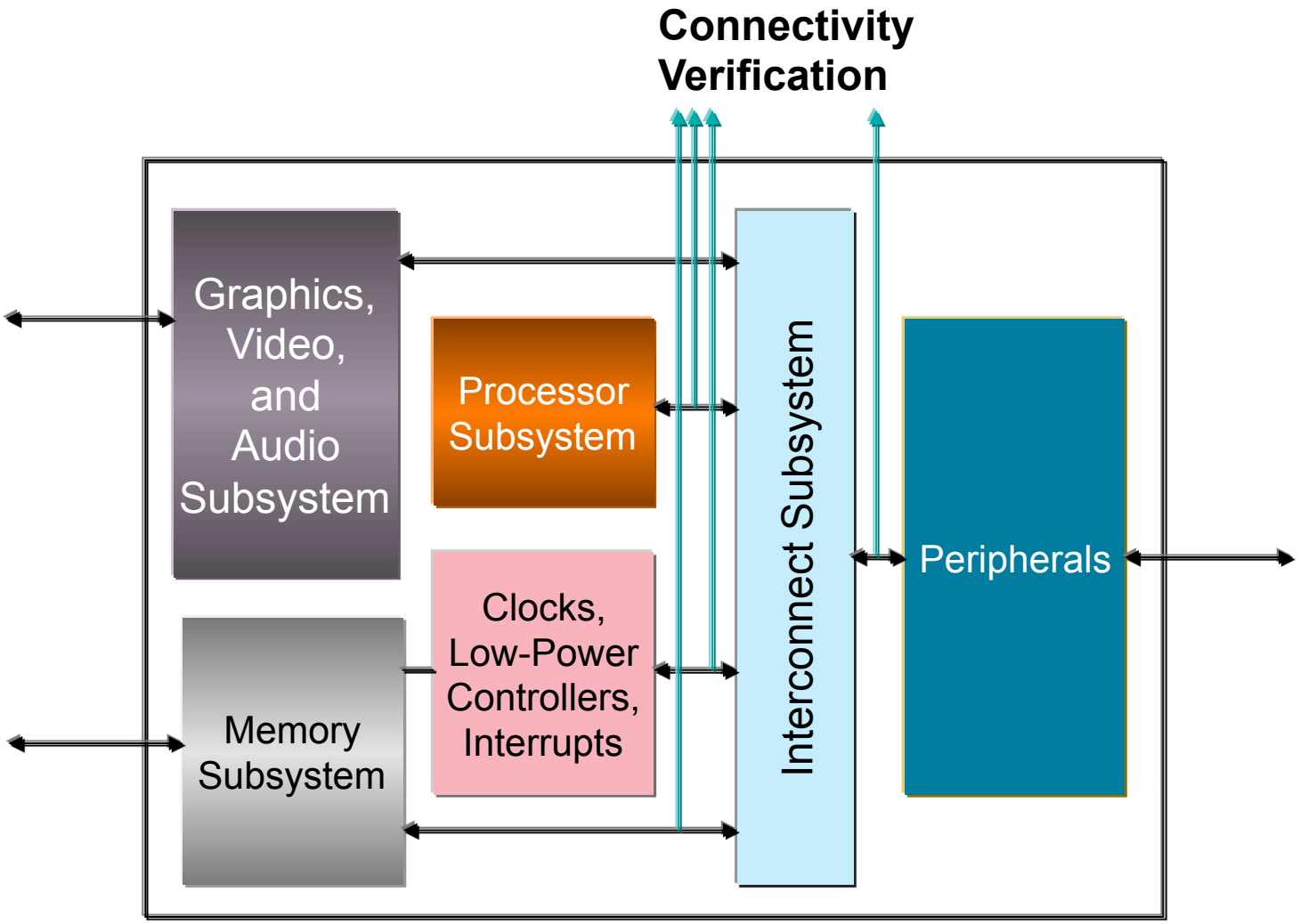


Control and status register verification

- Given a DUV with register space accessed by
  - Standard interface (AHB, OCP, etc.)
  - Proprietary interface (parallel, serial)

- To prove
  - Data integrity of register fields
    - i.e., Data read from a register equals previously written data
  - Reset values
    - Data read from a register equals reset value till it is written to

SoC-level Formal Verification



cadence

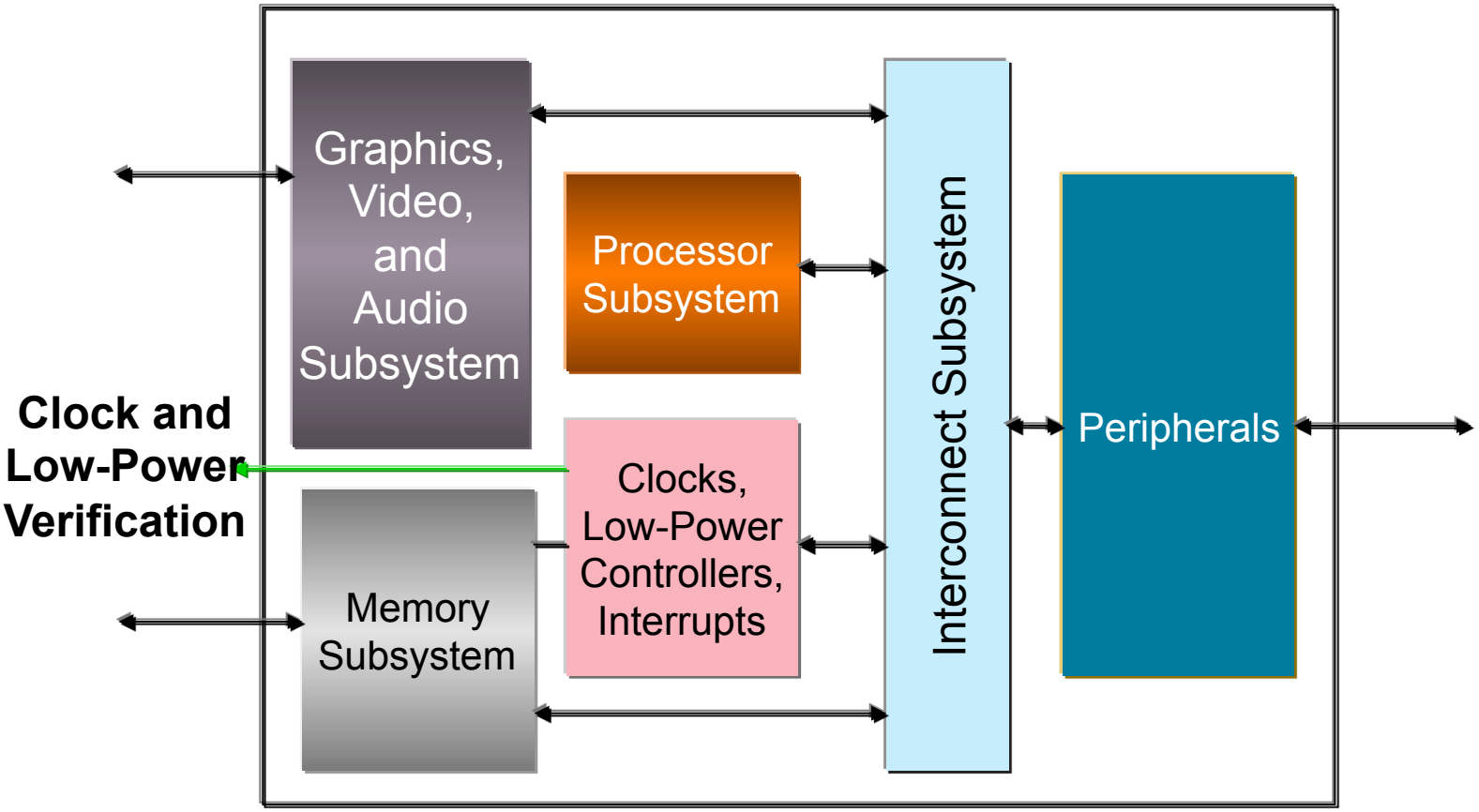
Connectivity verification

- System can have tens of thousands of static and multiplexed connections among IP blocks
- “Correct-by-construction” tools are not sufficient, as errors can still creep in due to unclear or erroneous specifications
- Additional low-power structures, BIST, and JTAG support, or downstream changes/ECOs that weren’t fully propagated
- Connectivity of a sub-module, that is correct as such, may prove to be erroneous in a larger scope

Connectivity Formal Verification

- Exhaustively verify that the design matches the connectivity definition
  - Verify that Point A is equivalent to Point B
  - No other signals/modes/settings can impact connections
- Quickly reproduce and reconfirm results (regressions) as RTL is being modified
- Easily add signal connection conditions or exceptions
- Read in directly the spec spreadsheet

SoC-level Formal Verification



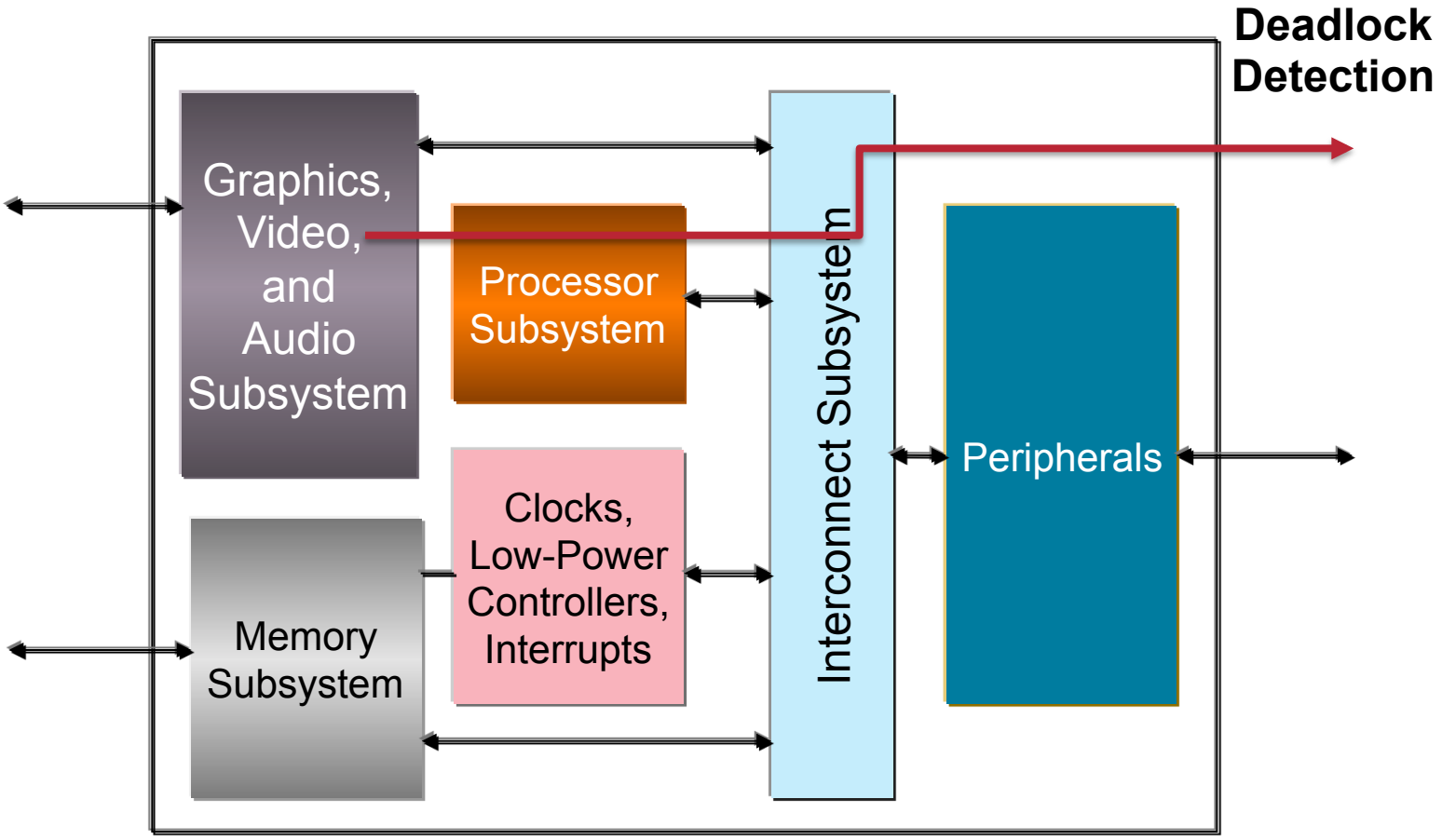
Clock and low-power Verification

- **Power domains:** Groups of blocks that can be turned off together. “Turn off” means all elements in these blocks lose their values and drive ‘X’
- **Isolation cells:** Isolated signal has a different driver (than stated in the RTL) if isolation condition is true
- **Retention cells:** Cells designed to retain the value of registers in domains that are “turned-off”
- **Level shifters:** Structures designed to allow operation in different voltage in adjacent power domains

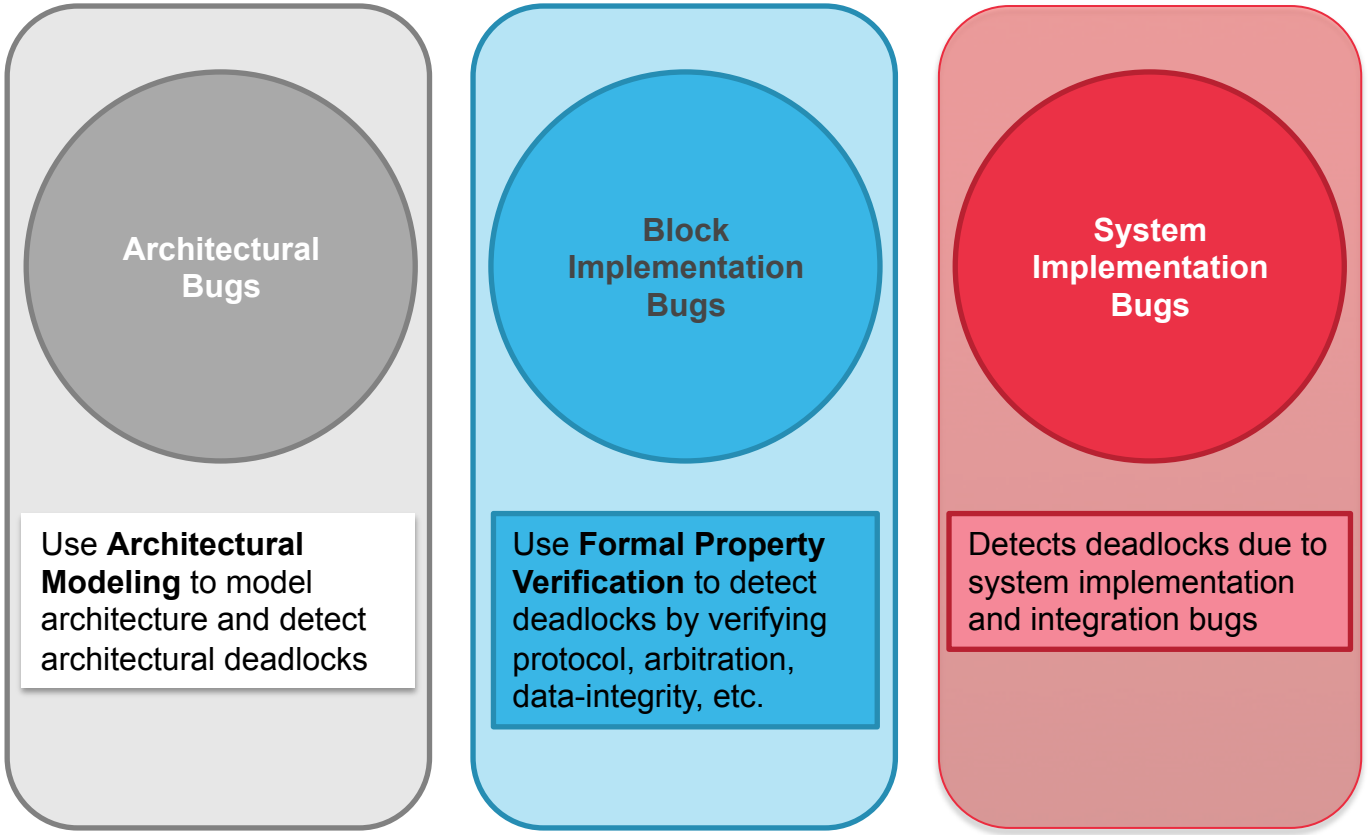
Example low-power verification tasks

- Power-up and power-down sequences
  - Visualization of waveform and checking required time delays
- Check that a switchable block does not generate additional X's at outputs (or dedicated signals) if it is powered down and up again
- Protocols (e.g., ARM® AMBA® AXI standard, etc.) are not violated due to a component being powered down
- FIFOs stretching over two power domains
  - Confirm they are isolated so that no spurious transitions occur

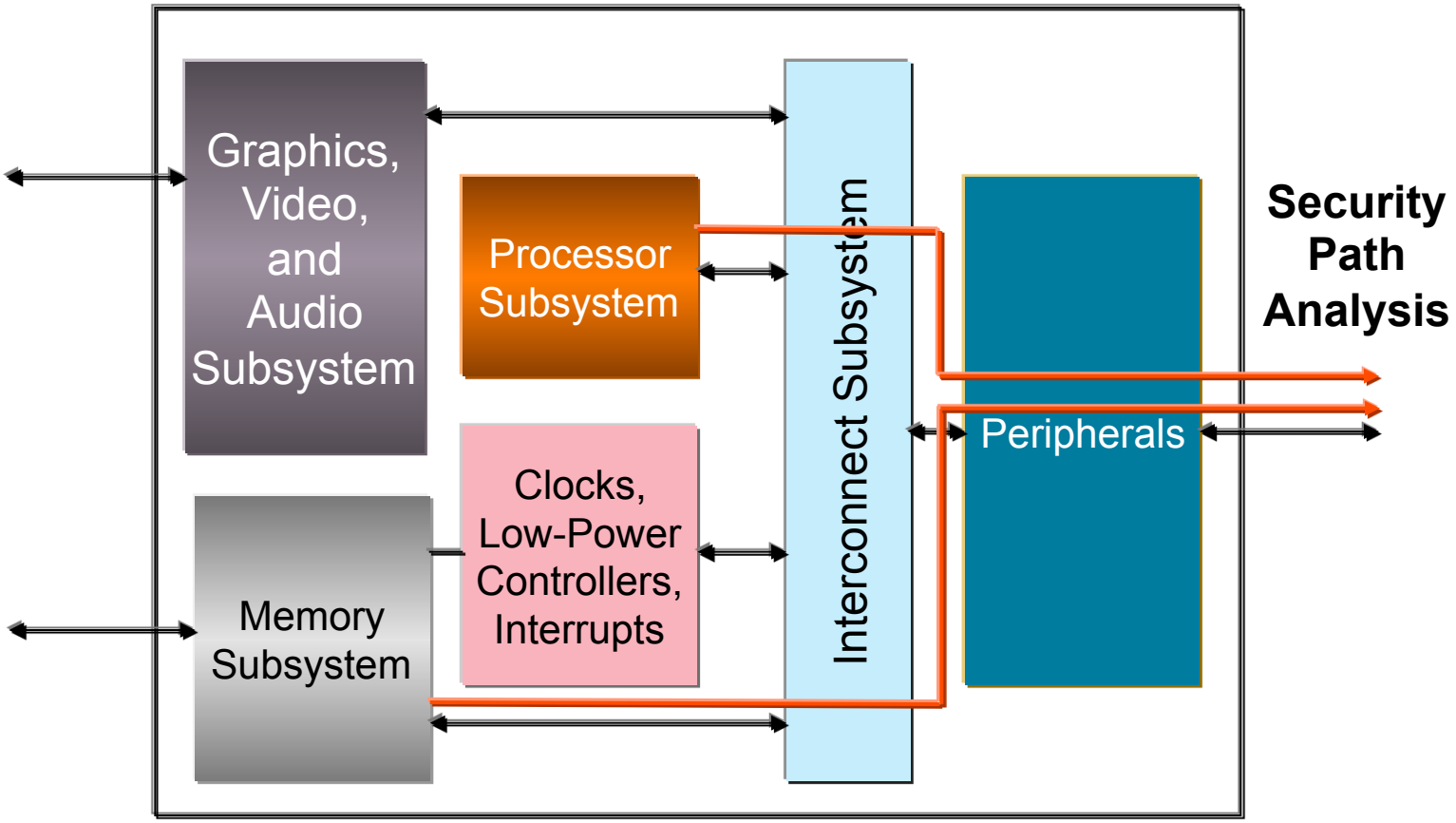
SoC-level Formal Verification



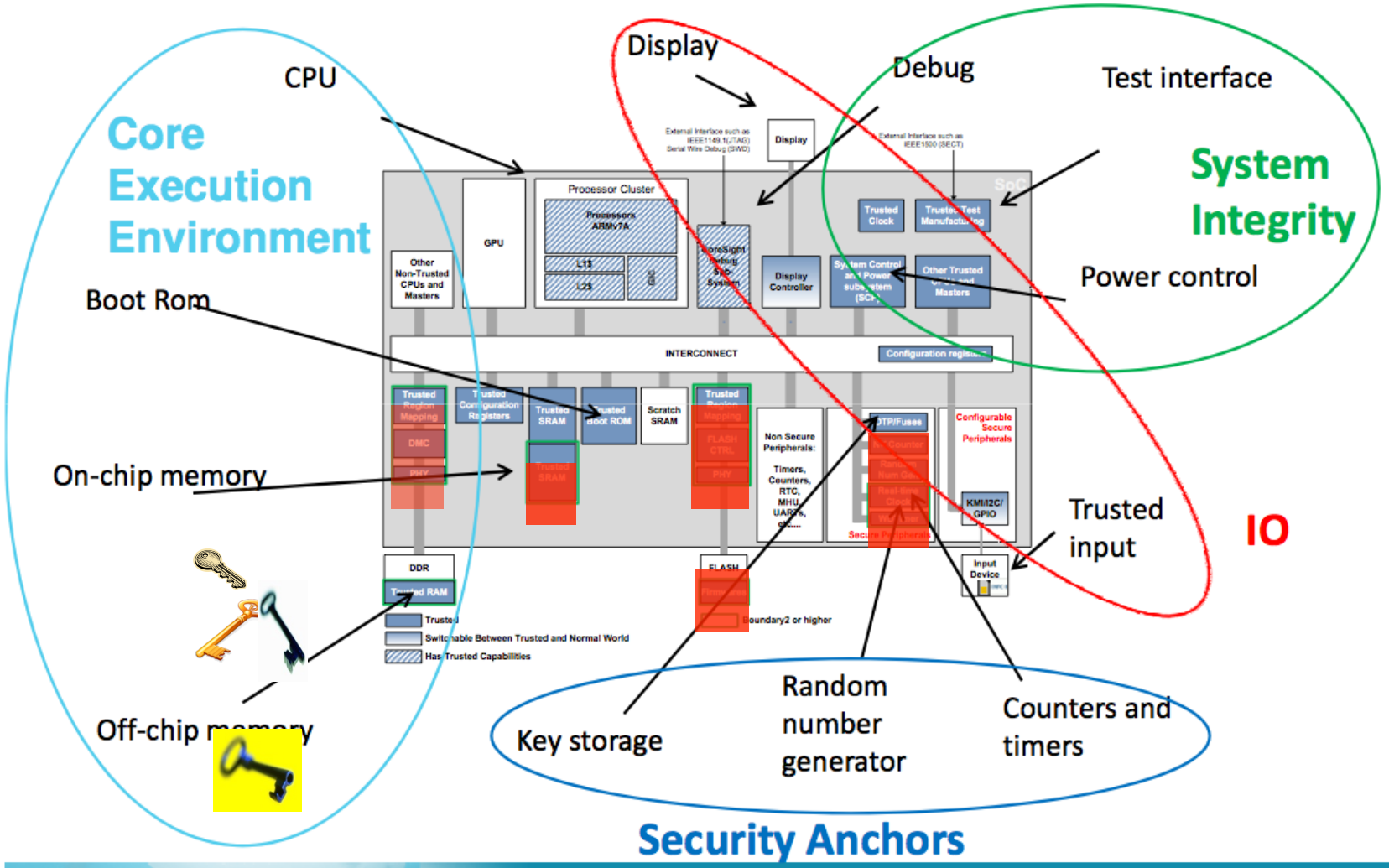
Deadlock Detection and Verification



SoC-level Formal Verification



ARM Trusted Base System Architecture



The Architecture for the Digital World®

ARM®

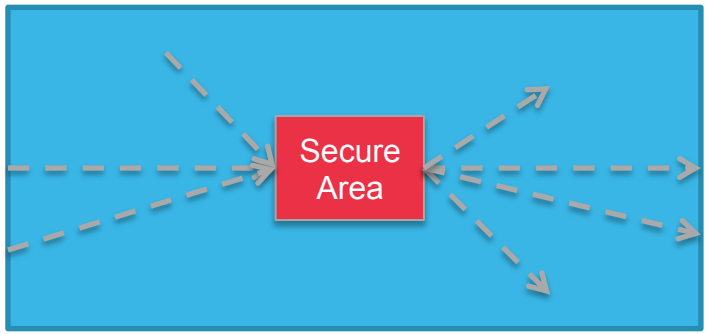
cadence®

What do people do today?

- System architecture
  - Moving secured areas to isolation
  - Analyzing data flow on the whole system
  - Manually identify potential structural paths and insert blocking conditions
- System integration
  - Investigate (often with a structural analysis tool) each structural path to ensure that they are false path
- Very tedious and ad hoc
  - Only able to look at a small subset of traces
  - Rather subjective and no clear checking mechanism
  - Difficult to get real sense of completeness

Security path verification overview

- Identify any unintentional path to/from secured areas
  - Use of path sensitization ideas
  - User specifies secure area and illegal sources and destinations for the data inside this area
    - Optional: User can specify blocks through which data is allowed to propagate
  - Waveforms show illegal propagations found

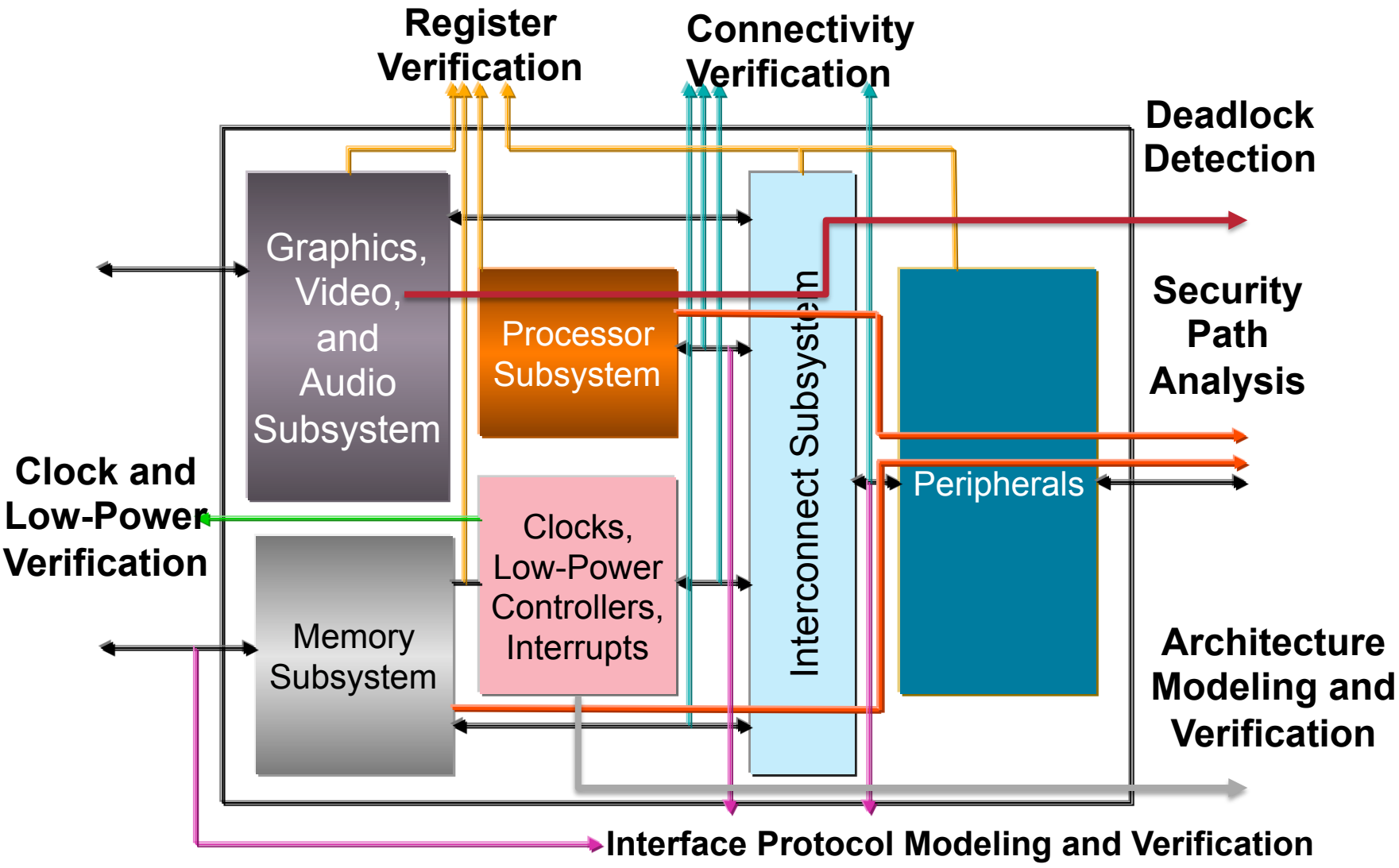


- Potential savings
  - Time savings: weeks vs. months
  - Verify all paths and understand progress
  - Completeness, not just subjective determination of correctness

Possible solutions

- **Structural analysis**: Look at structural paths and waive connections found. Very inefficient, subjective, and time consuming
- **Simulation**: Inserting Xs at the source and checking for Xs at the destination. Could find bugs, but will never guarantee completeness
- **Formal**: Different problem to handle, requires special modeling and formal analysis techniques

SoC-level Formal Verification



cadence

IP and subsystem-level design and verification solutions

- Formal property verification
- Sequential equivalence checking
- Structural property synthesis
- Behavioral property synthesis
- X propagation checking
- Coverage analysis and measurement
- Post-silicon debug (PSD)
- Clock glitch analysis and debug
- Functional Safety – ISO26262
- …

Good candidates for FPV

Sweet spots: concurrency and multiple data streams, which are difficult to completely verify using simulation

- Arbiters
- On-chip bus bridges
- Power management units
- Memory and DMA controllers
- Host bus interface unit
- Scheduler, implementing multiple threads
- Virtual channels for QoS

- Interrupt controller
- Token generators
- Cache coherency
- Credit manager blocks
- Standard interface (ARM AMBA protocol, DDR, etc.)
- Proprietary interfaces
- Clock disable units

Formal coverage

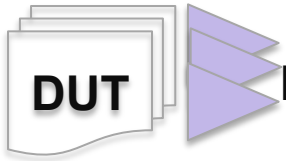
Stimuli Coverage

Formal Setup

How restrictive is the design behavior under the formal setup? What dead code does it generate?

Property Completeness Coverage

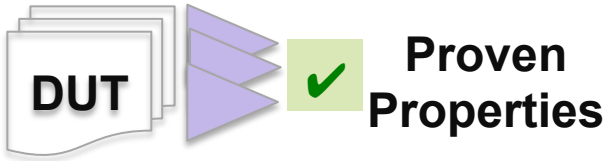
Formal Setup



How complete my property set? Do I capture all design behaviors?

Proof Coverage

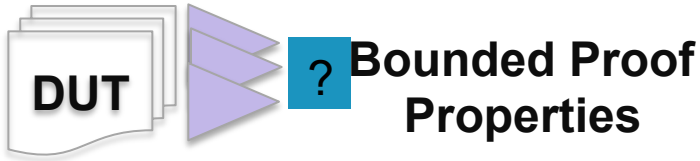
Formal Setup



What coverage is achieved by the proven properties?

Bounded Proof Coverage

Formal Setup

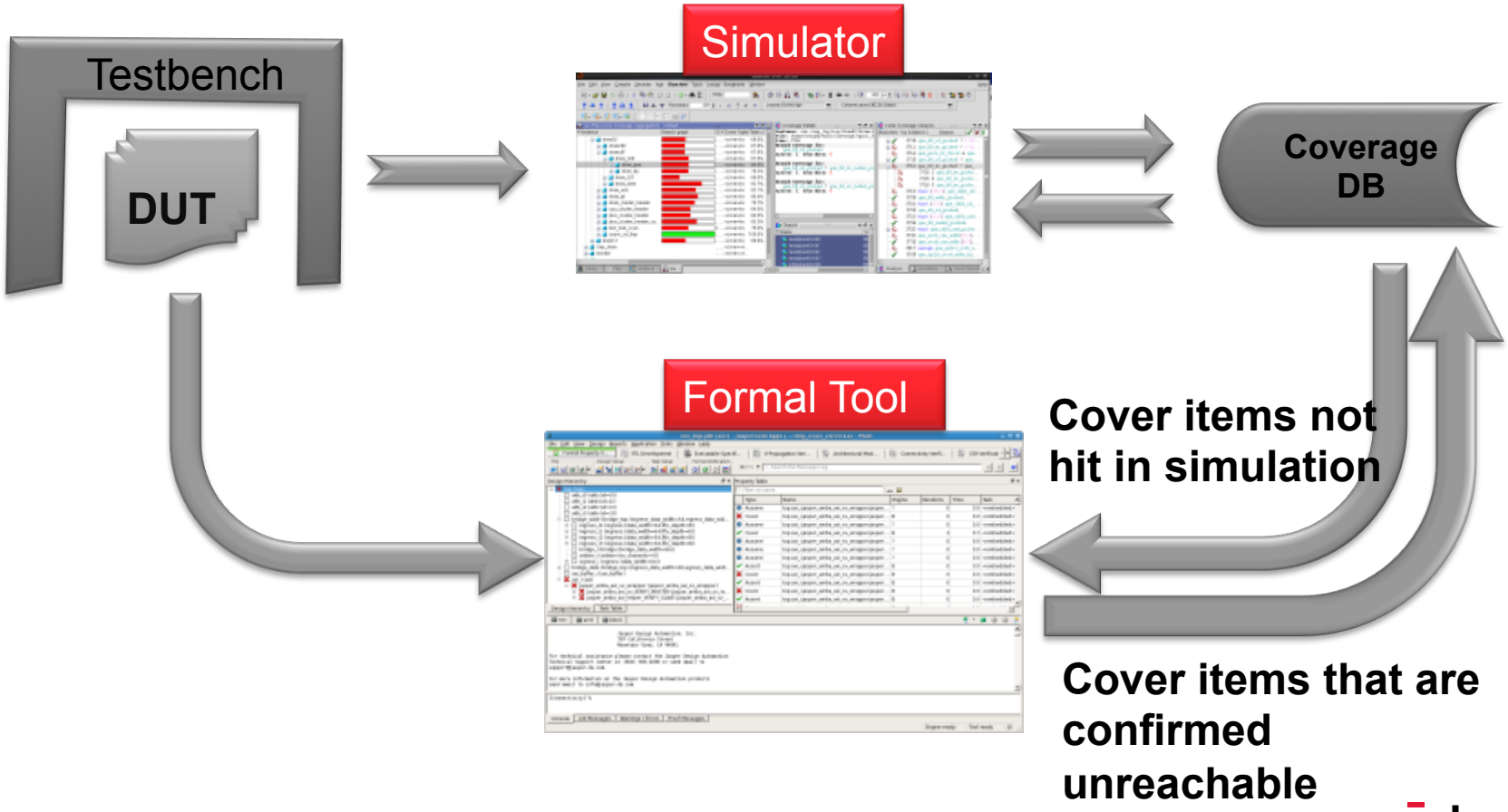


What is coverage of bounded proof? Is the bound enough? How to do better?

cadence

Interacting with coverage metrics from simulation

Use formal to cover areas that are hard for simulation to address, or formally prove they are impossible to reach

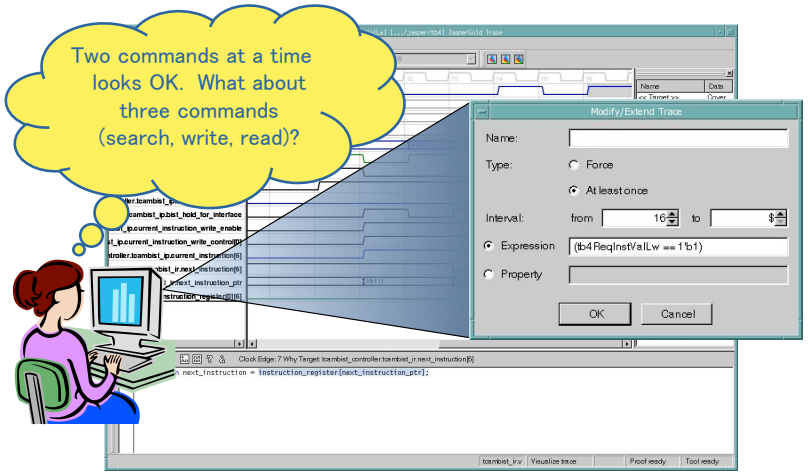


Core model checking technology

- Performance, capacity, and convergence of core engine
- Automatic abstraction
- Application-specific automatic proof strategies
- Word-level model checking
- Hybrid of simulation and formal technologies
- Predictability
- Coverage and progress measurement

Debug and Verification Productivity

- 70% of verification effort is in debug loop !!
  - Spec, Design, Tools, setup, …
- Debug vs. root-cause analysis
- Self-aware tools
- Pre vs. post silicon validation



cadence

Challenges and opportunities ahead…

- Address cost and productivity issues in design and validation
- Can we get 100% proofs on RTL models (at least for selected IP blocks)?
  - If not, how do we measure the coverage?
  - What is the cost?
- HLM for verification is unavoidable…
- Cross software/hardware verification is a huge challenge
- Leverage the great progress in parallel and distributed computing