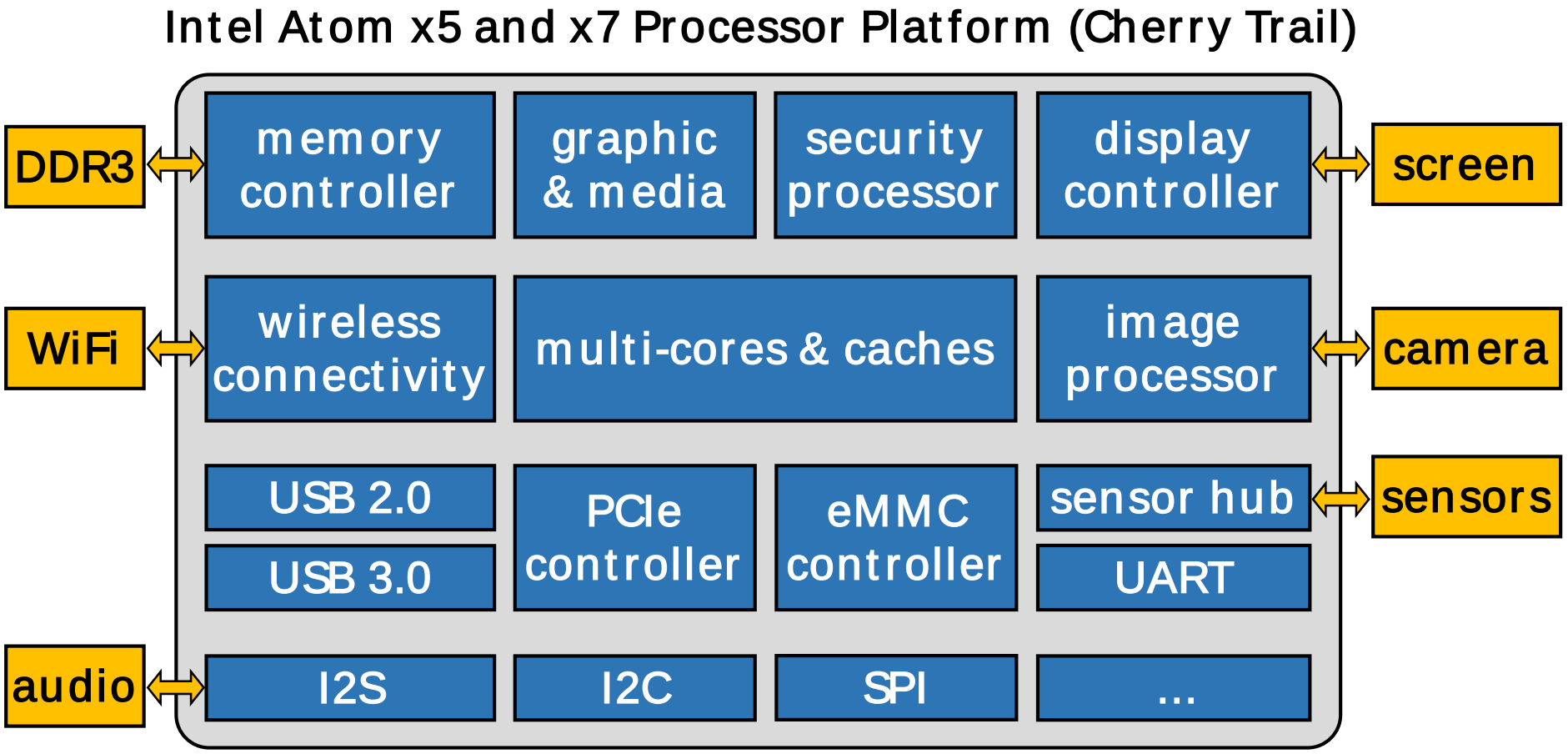
AGARSoC: Automated Test and Coverage-Model Generation for Verification of Accelerator-Rich SoCs

Biruk Mammo, Doowon Lee (Speaker), Harrison Davis, Yijun Hou, and Valeria Bertacco

Dept. of Electrical Engineering and Computer Science
University of Michigan

MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

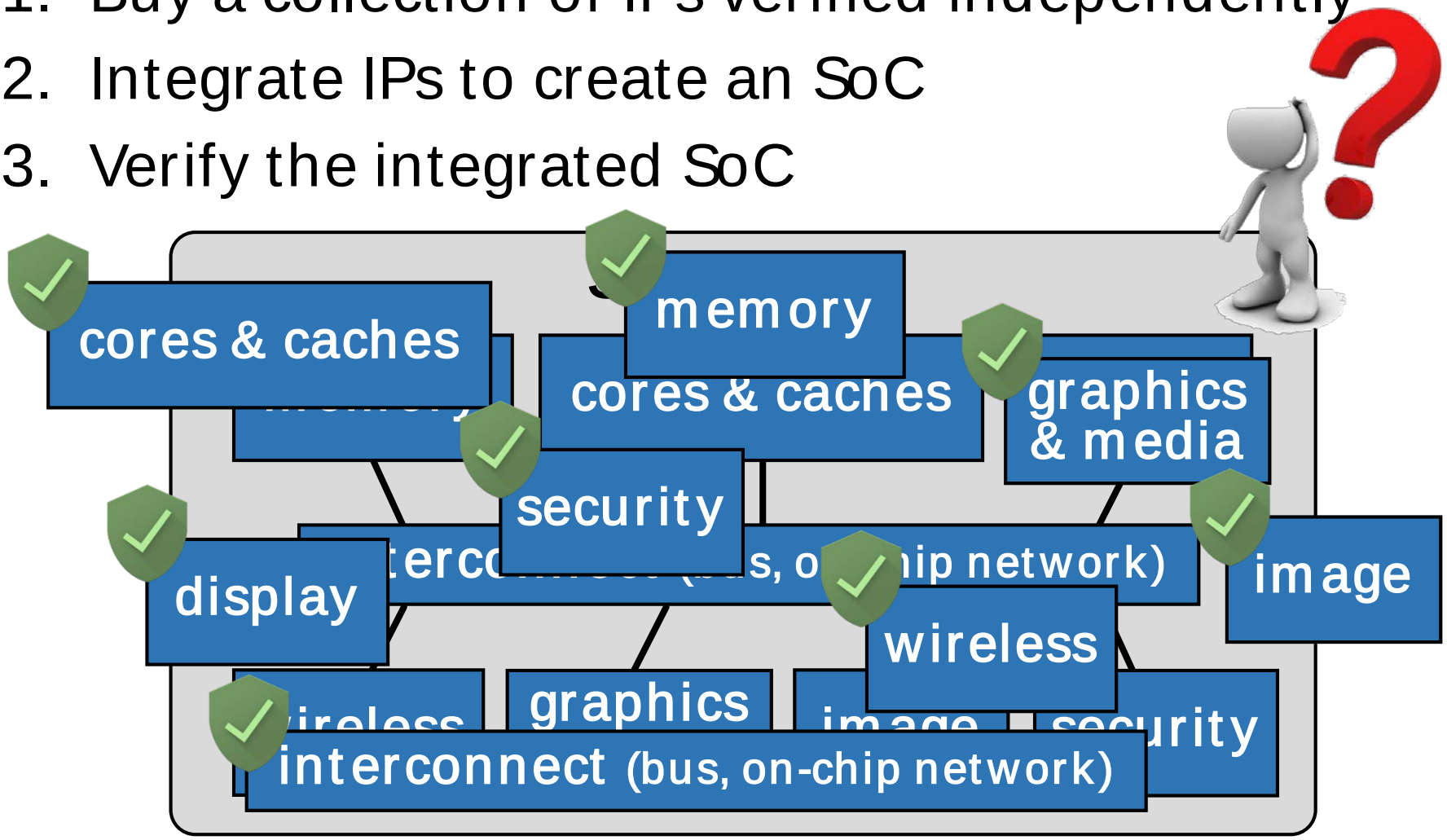
Accelerator-Rich System on Chip



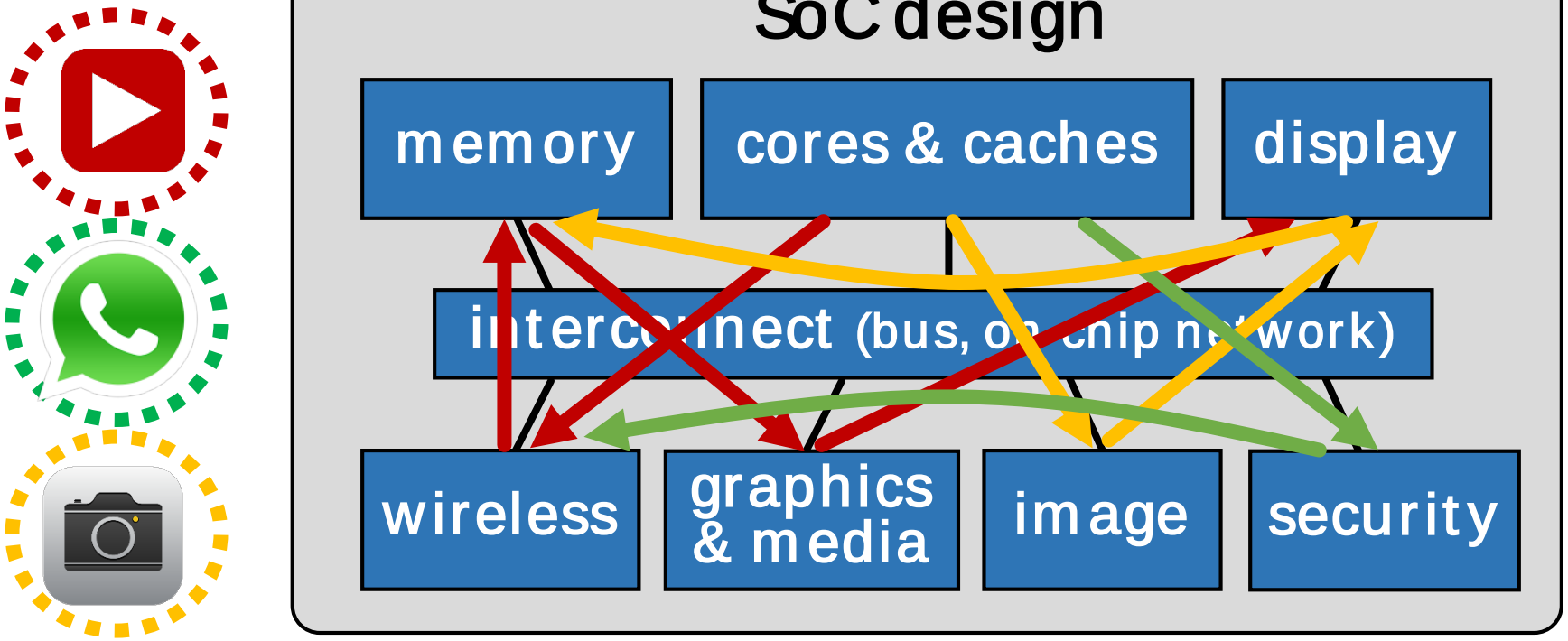
sea of accelerators

SoC Integration

1. Buy a collection of IPs verified independently
2. Integrate IPs to create an SoC
3. Verify the integrated SoC



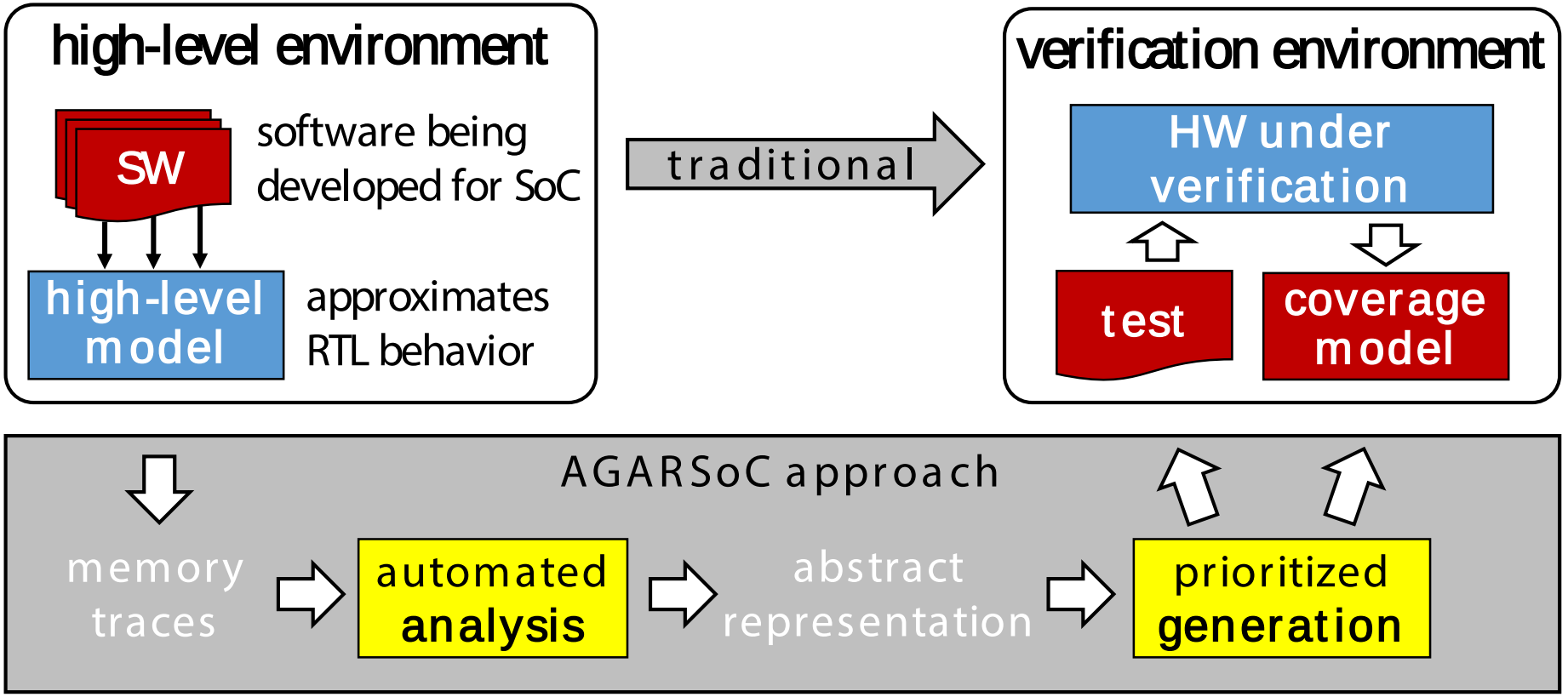
Interactions among Accelerators



GOAL: identify interactions that are likely to happen & gear verification effort towards them

AGARSoC Overview

- Automated Test and Coverage-Model Generation for Verification of Accelerator-Rich SoC

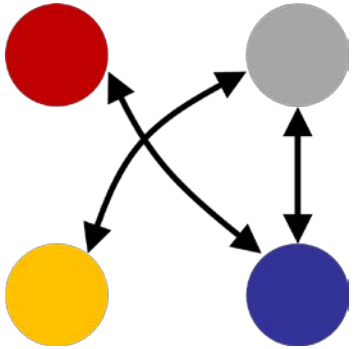


Outline

- Introduction
- **AGARSoC: accelerator-rich SoC verification**

memory trace analysis

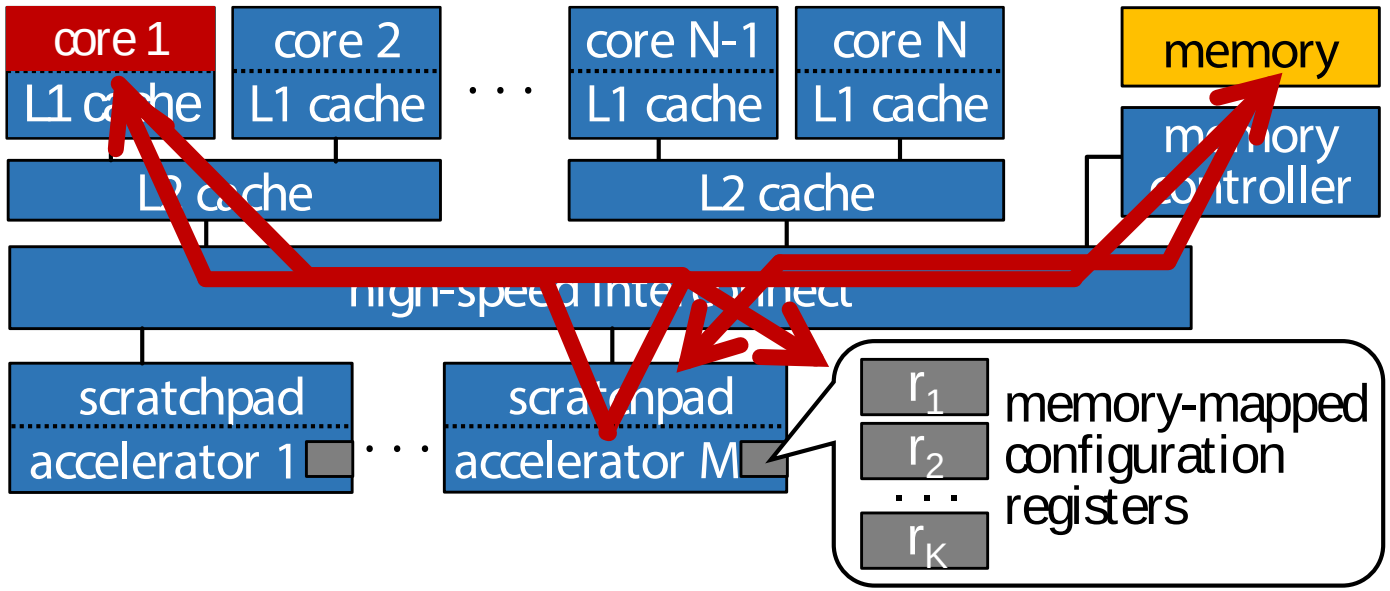
abstract representation



test and coverage-model generation

- Experimental evaluation
- Conclusion

Accelerator Execution Model

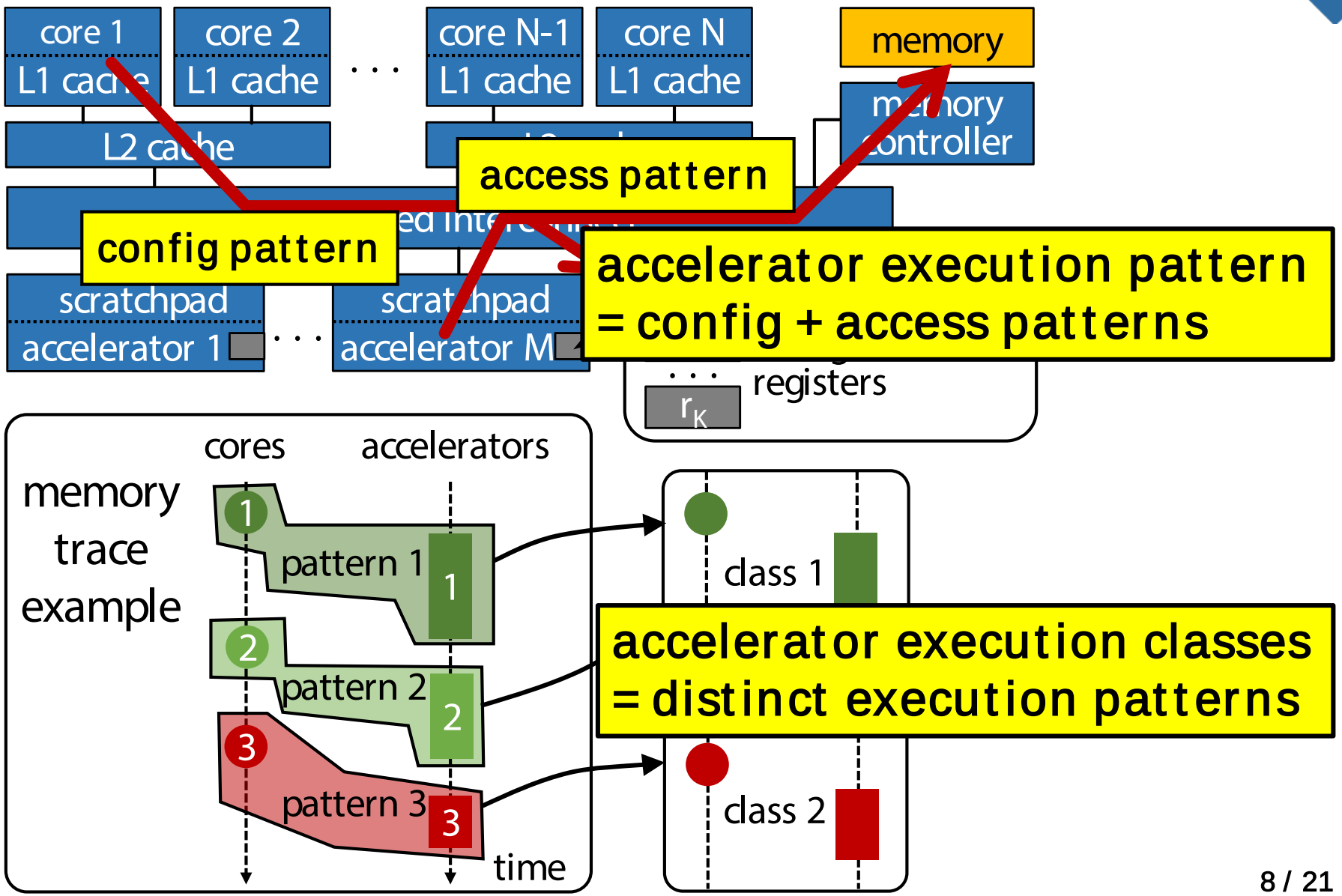


All units communicate via memory operations

Task execution sequence:

task setup ⇨ data fetching ⇨ result writing ⇨ completion notification

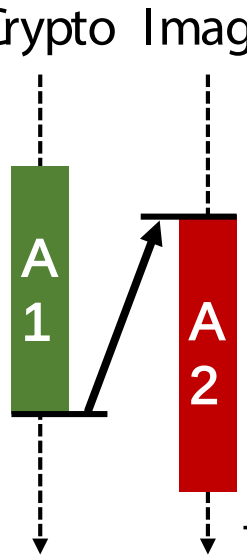
Memory Trace Analysis



Accelerator Interaction Scenarios

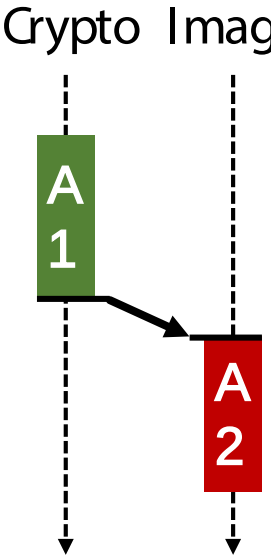
- Interactions among execution classes
  - Execution classes = distinct accelerator patterns
- Two types of interactions

1. Concurrent



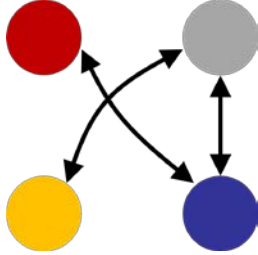
on-chip network delay;
memory contention

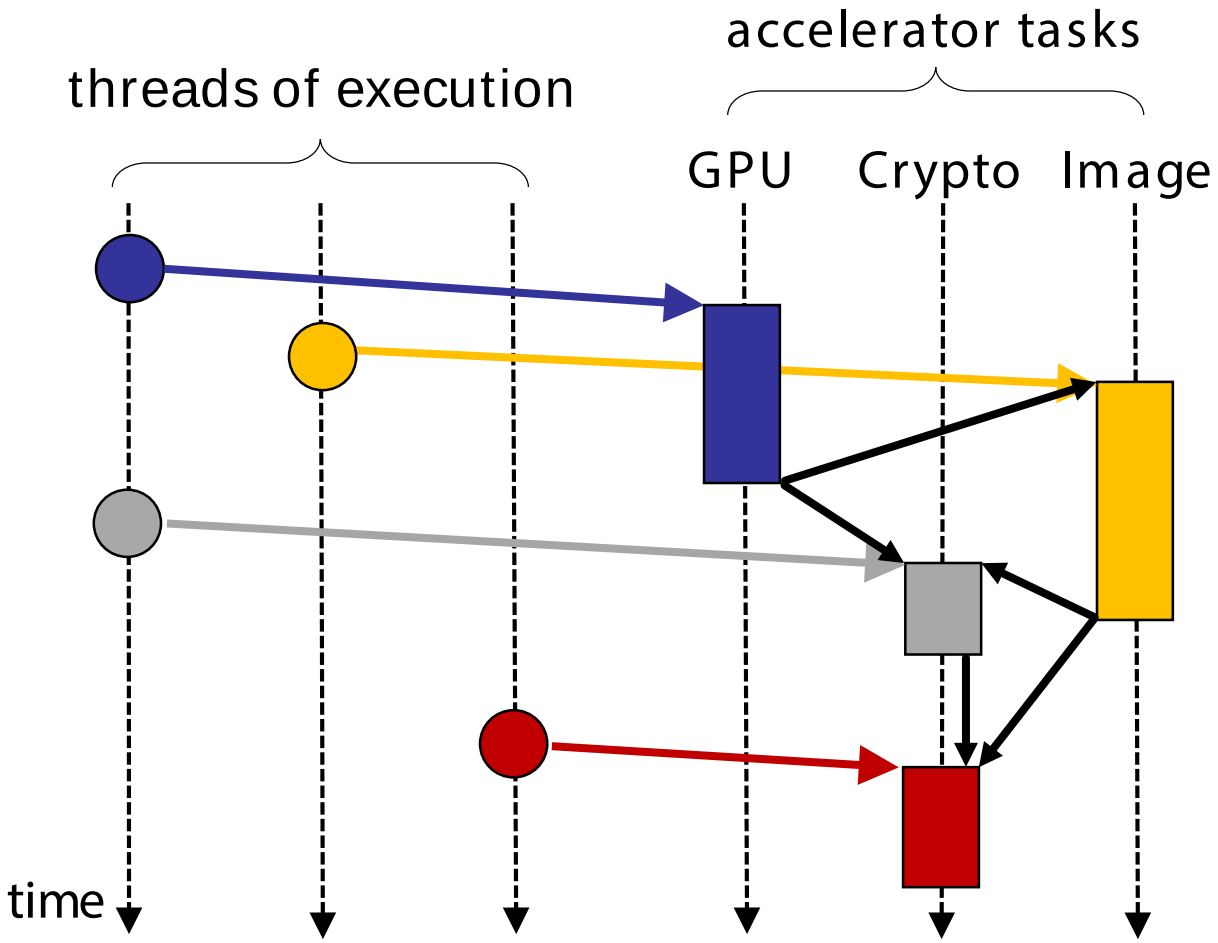
2. Sequential — synchronized / observed



state of on-chip network;
memory status after A1 affects A2

Abstraction Example

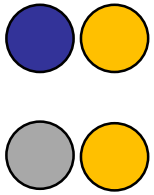




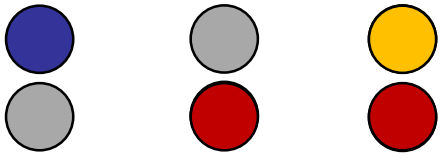
We would like to extract:

1. execution classes

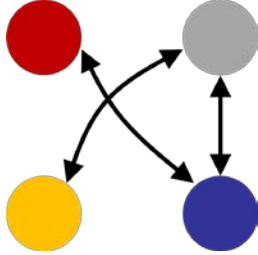
2. concurrent scenarios

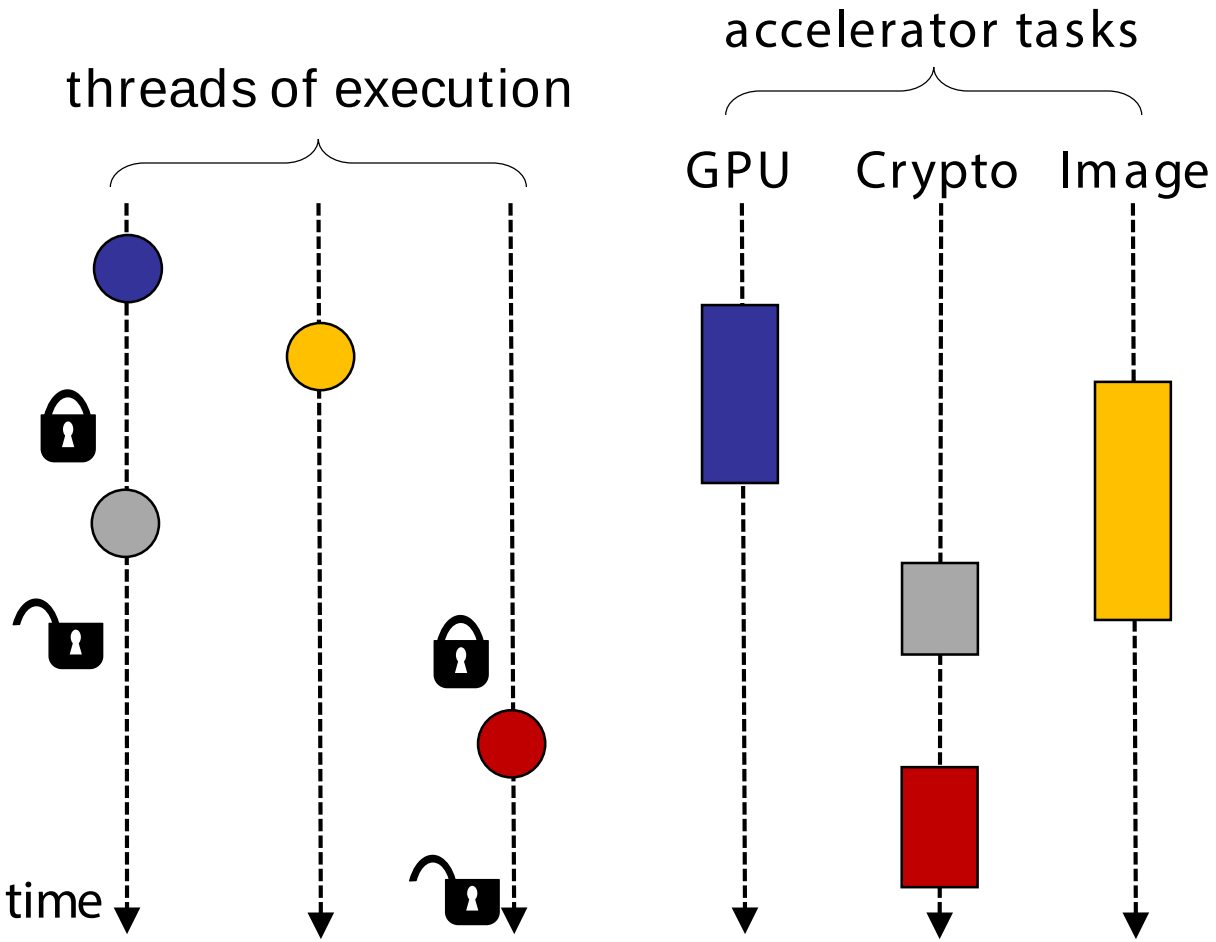


3. sequential scenarios



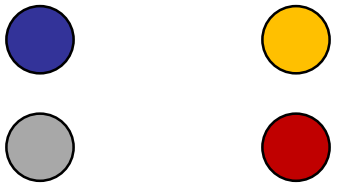
Inferring Unobserved Order



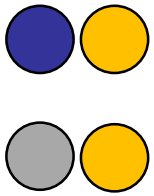


lock-synchronized orders: or

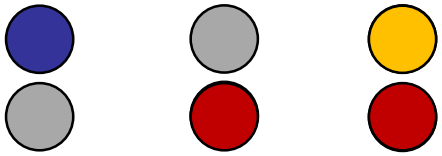
1. execution classes



2. concurrent scenarios



3. sequential scenarios



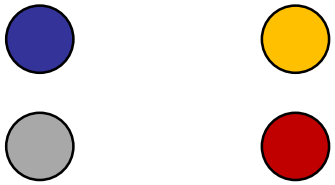
inferred
(unobserved)

Test Generation

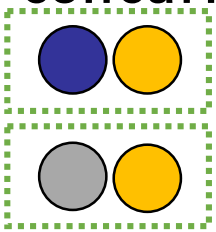
- **Multi-threaded**, **multi-phase** test
  - Multiple threads – concurrent execution scenarios
  - Multiple phases – sequential execution scenarios
- **Minimal schedule** covering all concurrent and sequential scenarios
- Test-generation configurations
  - What (1) **execution classes**, (2) **concurrent scenarios**, and (3) **sequential scenarios** should be selected?
  - Should **inferred sequential scenarios** be included?

Test Generation Example

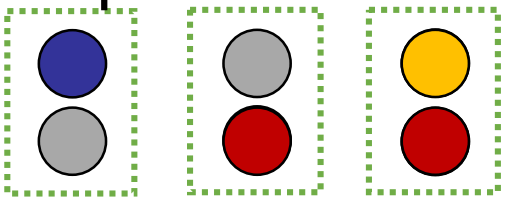
execution classes

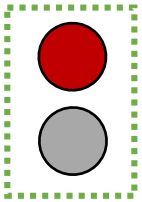


concurrent scenarios



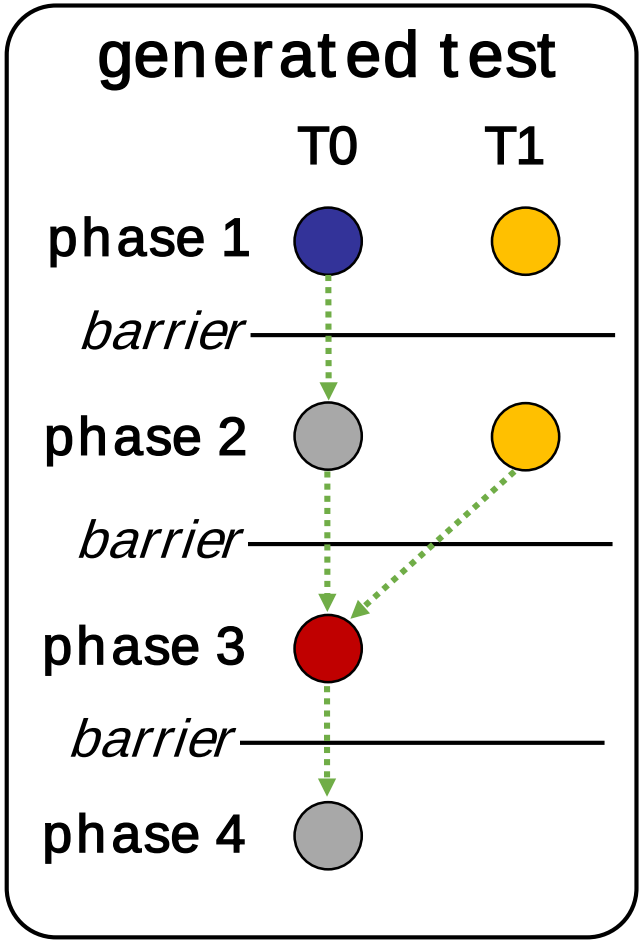
sequential scenarios

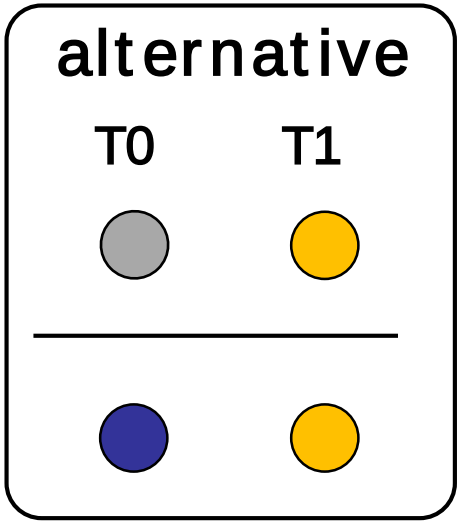




Test-generation setting:

"Cover all concurrent and sequential scenarios"

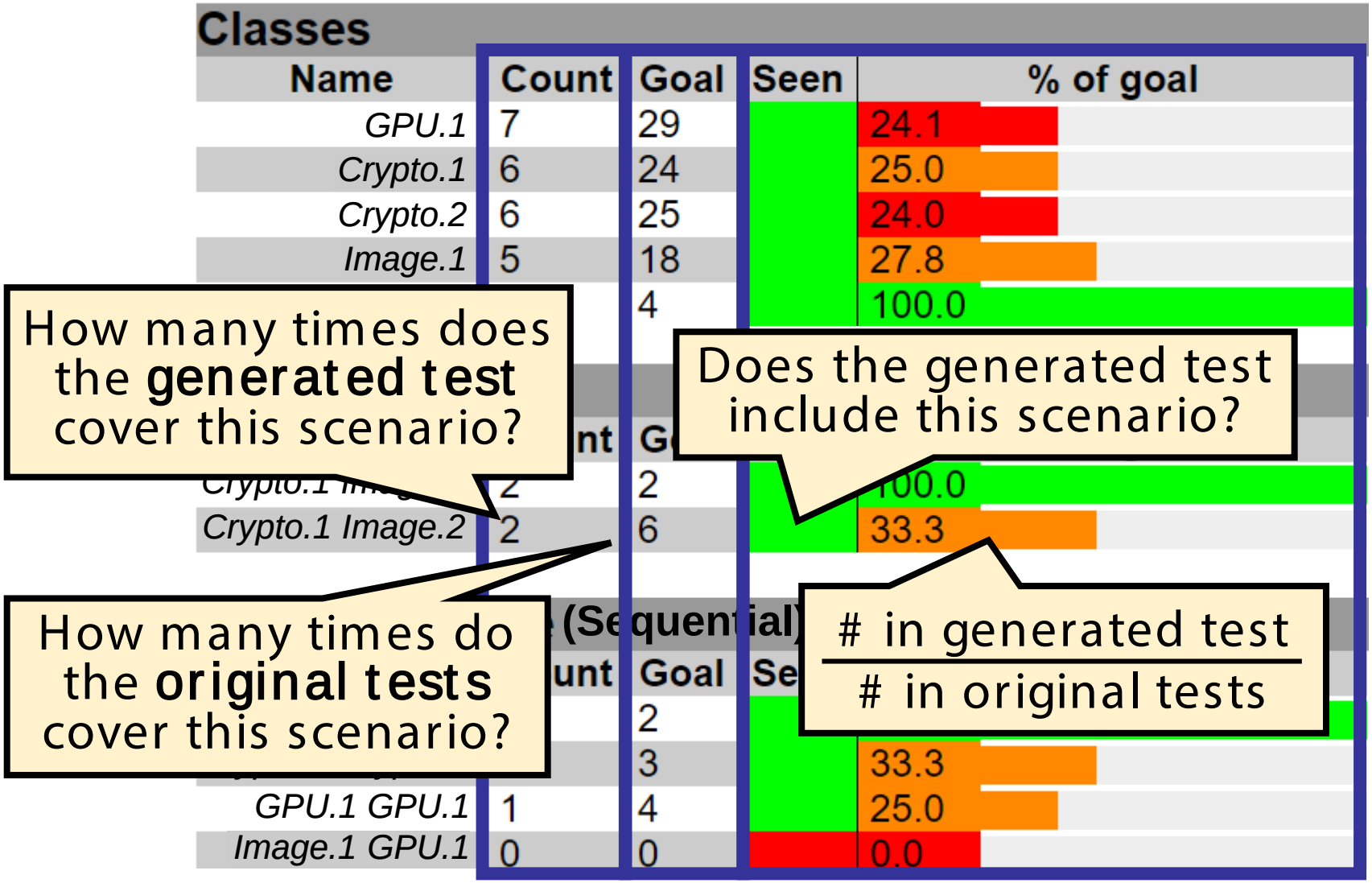




(1st sequential scenario missed)

Coverage Report

- Automatically generated along with test



Outline

- Introduction
- AGARSoC: accelerator-rich SoC verification
- **Experimental evaluation**
  - Experimental setup
  - Runtime
  - Compaction rate
- Conclusion

Experimental Setup

- **Two SoCs** modeled (SystemC and RTL)
- **SystemC model** built on SoCLib framework*
  - **3 cores** (ARMv6k), 3 memory modules
  - **3 accelerators** (QPSK modulator, demodulator, FIR filter)
  - 1 shared bus
- **RTL model** built on Xilinx Vivado® design suite
  - **3 cores** (MicroBlaze), 1 shared memory
  - **6 accelerators** (FIR filter, CIC filter, CORDIC module, convolutional encoder, FFT module, complex multiplier)
  - 2 AXI interconnects
- Custom test-program suites

* http://www.soclib.fr/trac/dev

Test Suites

- 5 groups of test programs for SystemC model

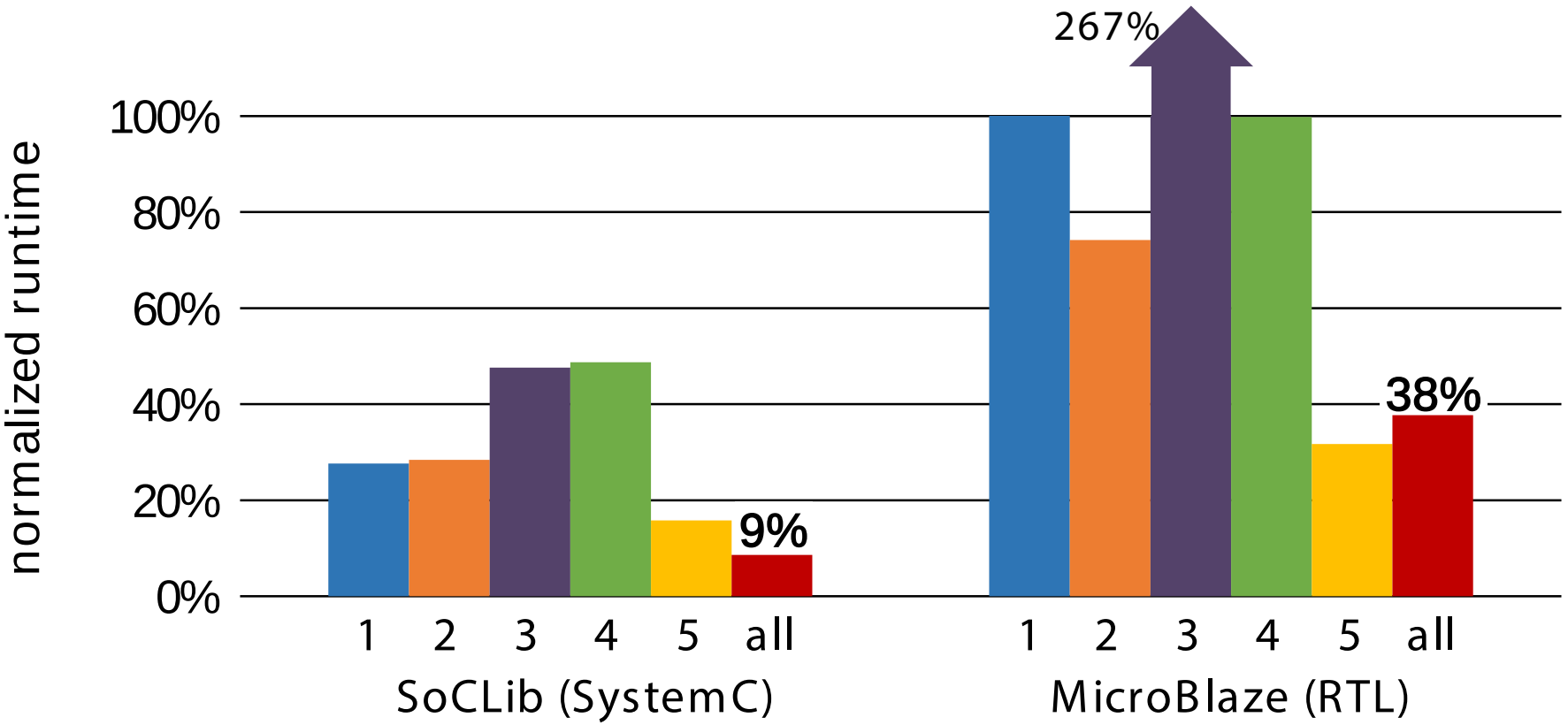
no.	# of programs	description
1	9	sequential accesses with locks
2	9	sequential accesses without locks
3	9	concurrent accesses with locks
4	9	concurrent accesses without locks
5	18	combinations of the four above

- 5 groups of test programs for RTL model

no.	# of programs	description
1	7	no synchronization
2	8	lock, single-accelerator invocation
3	5	lock, multi-accelerator invocation
4	7	barrier, redundant computations
5	13	semaphore synchronization

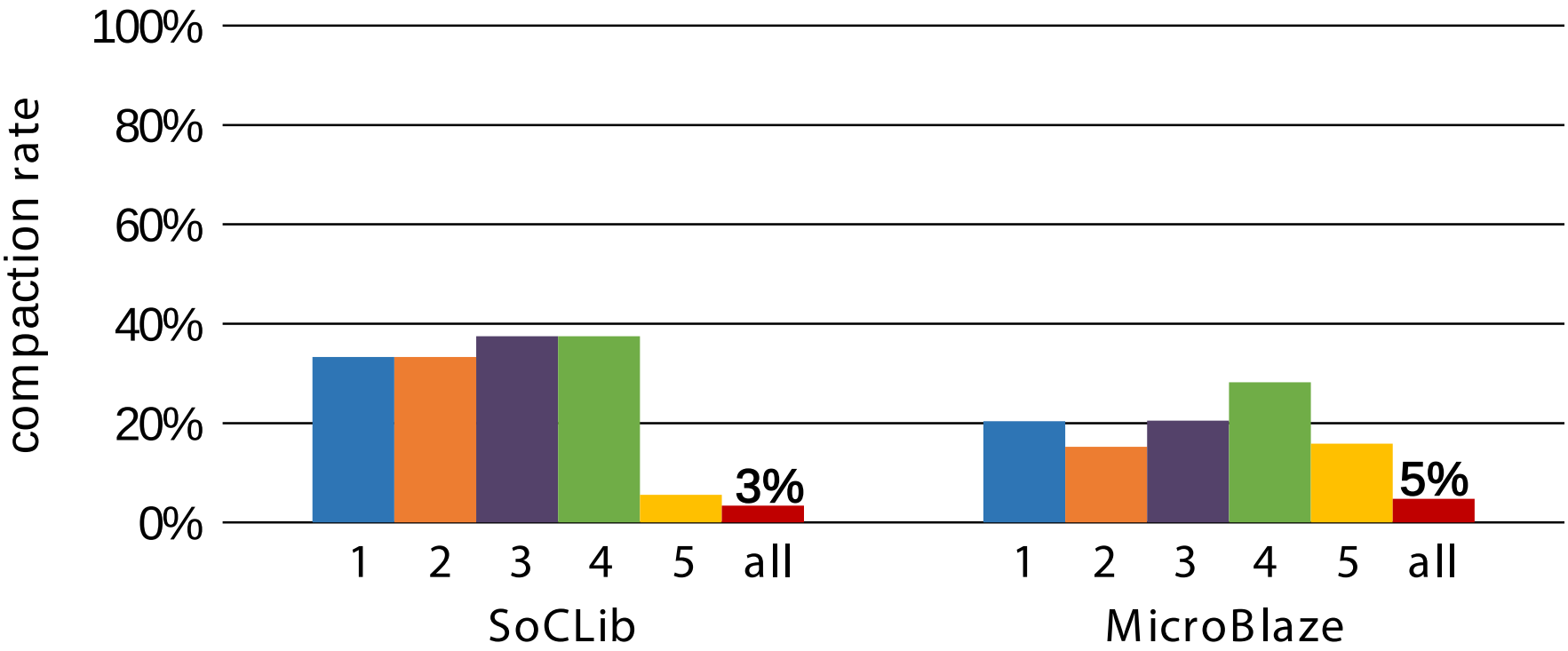
Runtime of Generated Tests

$$\text{normalized runtime} = \frac{\text{runtime of generated test}}{\text{sum of runtimes of individual tests}}$$

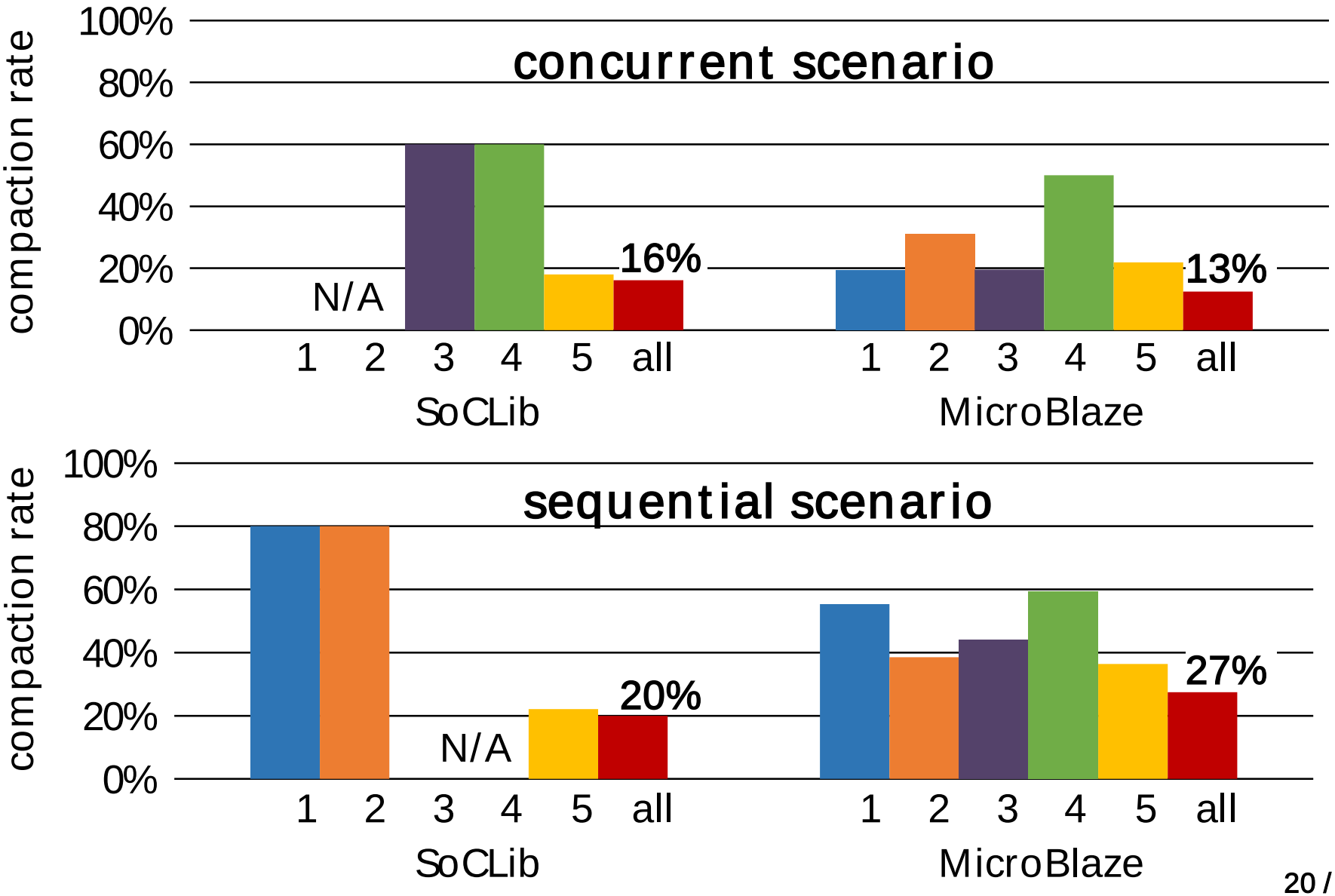


Compaction Rate—Execution Class

$$\text{compaction rate of execution class} = \frac{\#\text{ of unique execution classes}}{\text{total \# of accelerator invocations}}$$



Compaction Rate— Concurrent and Sequential Scenarios



AGARSoC Conclusions

- Verifying accelerator interactions in SoCs
- Analyzing software behaviors in high-level model to identify high-priority interactions
- Generating compact test to quickly achieve coverage goals
- Future work: data-sharing patterns, other SoCs

Thank you! Question?

Backup

Configuring Memory Trace Analysis

- Minimal engineering effort to specify
  1. How to delimit execution from memory trace
  2. How to distinguish unique execution classes
- Delimiting execution
- Identifying classes

write 0x1 to reg[0x4]
write 0x0 to reg[0x8]
read from reg[0xC]
write 0x3 to reg[0x10]
write 0x0 to reg[0x8]
write 0x3 to reg[0x10]
write 0x1 to reg[0x4]

memory accesses to config registers

write 0x1 to reg[0x4]
write 0x0 to reg[0x8]
read from reg[0xC]
write 0x3 to reg[0x10]

Particular memory operation signaling the end of task (e.g., "write to reg 0x10")

What registers define the mode of operation? (e.g., 0x4 and 0x10)

{addr 0x4, value 0x1}, {addr 0x10, value 0x3}

class identifier

Discussions

- Flexibility
  - Two different SoCs evaluated in our experiment
  - Applicable to SoCs where memory operations can be observed
- Enhancement
  - Data-sharing patterns
  - Randomize software executions
- Accuracy
  - Different interactions observed in untimed high level
- Bug detection capability
  - Similar to original software