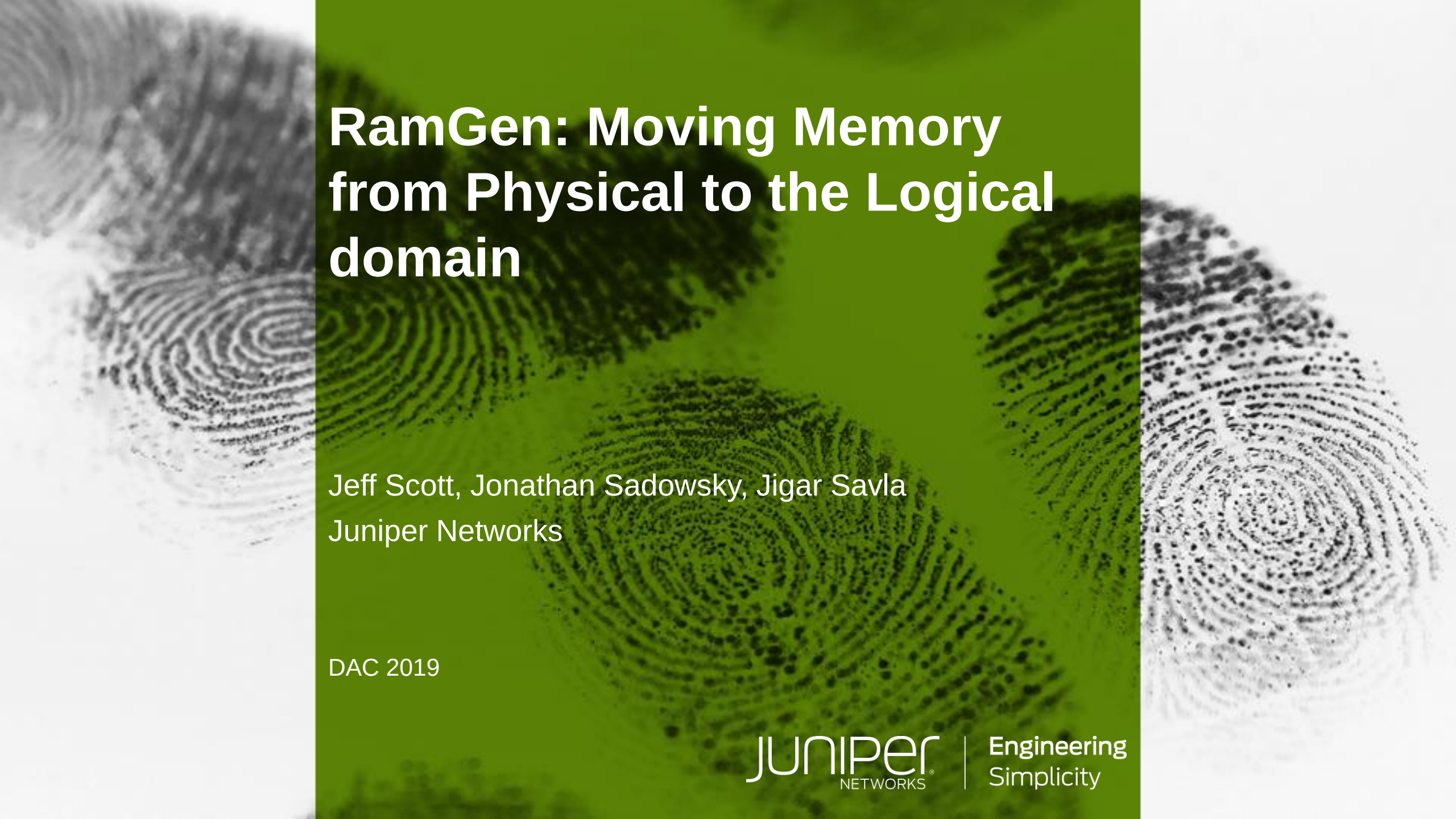


RamGen: Moving Memory from Physical to the Logical domain

Jeff Scott, Jonathan Sadowsky, Jigar Savla
Juniper Networks

DAC 2019

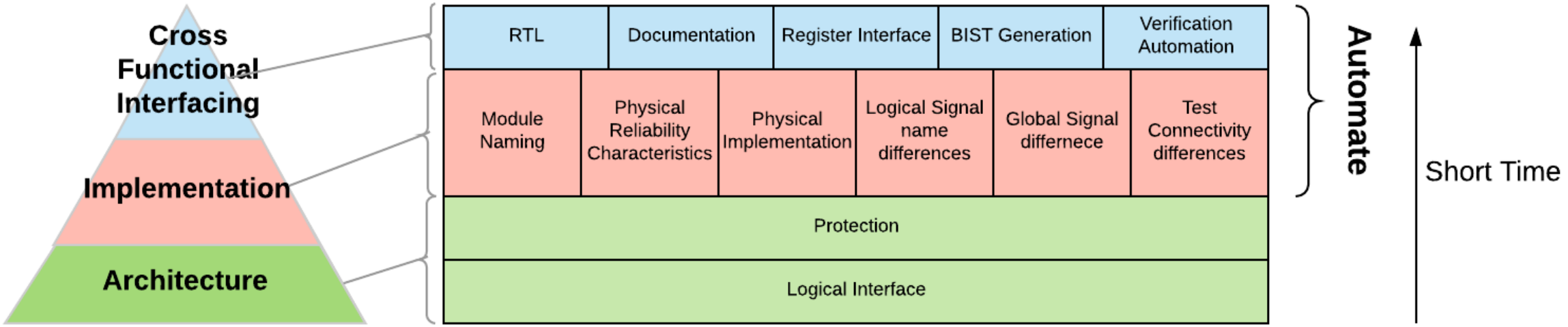
Juniper Networks | Engineering Simplicity

Motivation

For a memory, how does a designer give a high-level specification like width, depth and data protection type right at the start of the chip development and have it used throughout till tape-out?

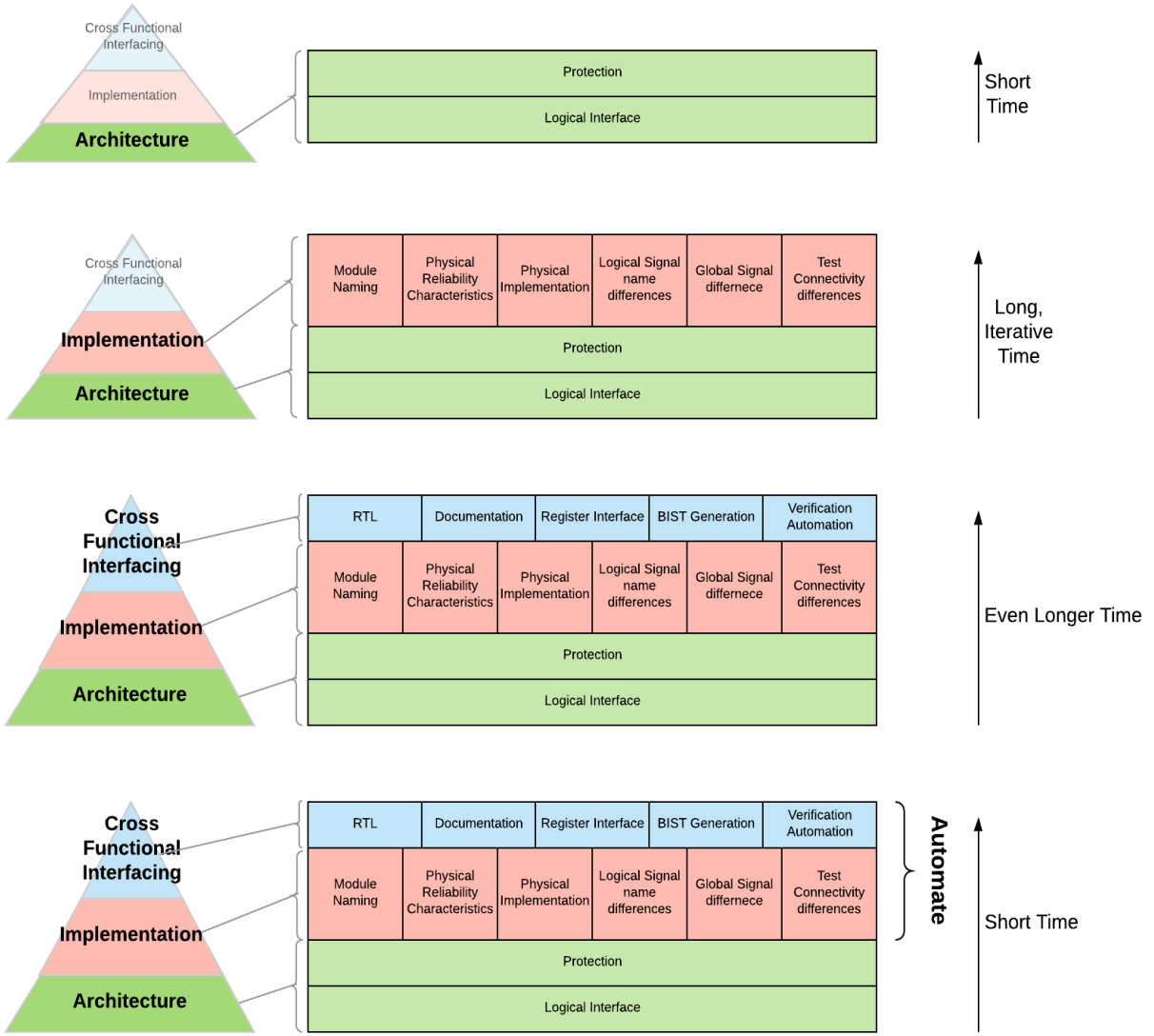
Enable designers to get Simulation / Verification (DV) up and running before the memory vendor has been finalized or the physical memories have been ordered.

Answer is having a ***logical memory*** **instead of a** ***physical memory*****.**



Main Idea

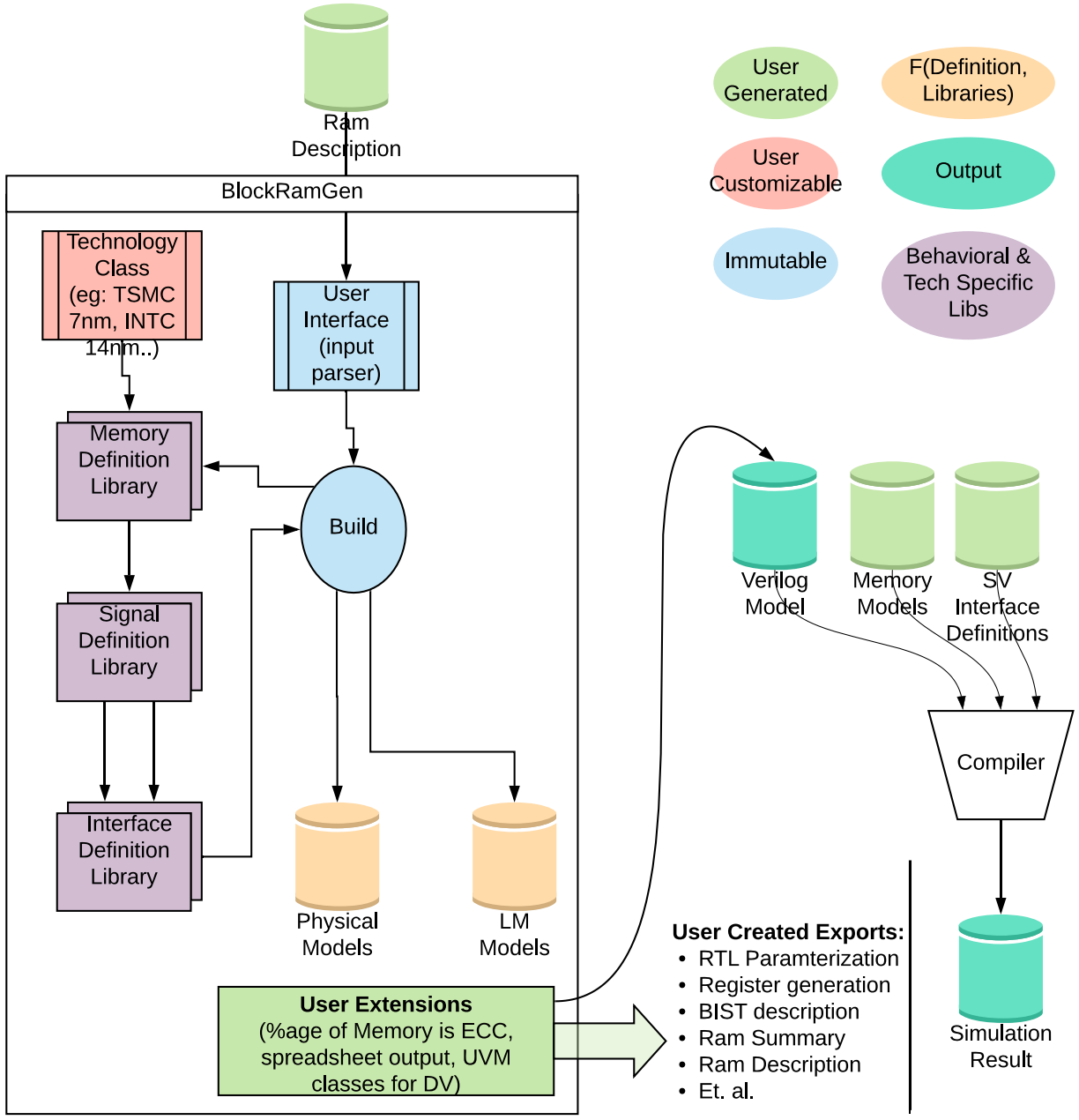
- Without having the node and vendor constraints, how do you generate memories without re-working every time a new piece of constraint is introduced. Our improved flow is called RamGen.
- RamGen auto-magically incorporates information for all of the following:
  - Backdoor access
  - Data protection
  - Memory models
  - *FieldSets* & more…



Inner workings

- We have a Perl based hash table. It has several parameters, with a very minimum sub-set required to start prototyping. It creates SystemVerilog *interfaces* for functional and error correction. Having this makes it usable consistently for both Design and DV.
- All you really need is the Memory Type, Width, Depth, and I/O flopping. The remainder can be defaulted to off. Everything else is refinement as you go from a generic logical memory to a **design specific** logical memory (with protection and test, etc.) to a **physical memory** (with tiling, etc.).
- It also accesses a user dB to access vendor specific programmable information for other constraints which can then query specific parameters of this dynamic memory structure to make appropriate changes.

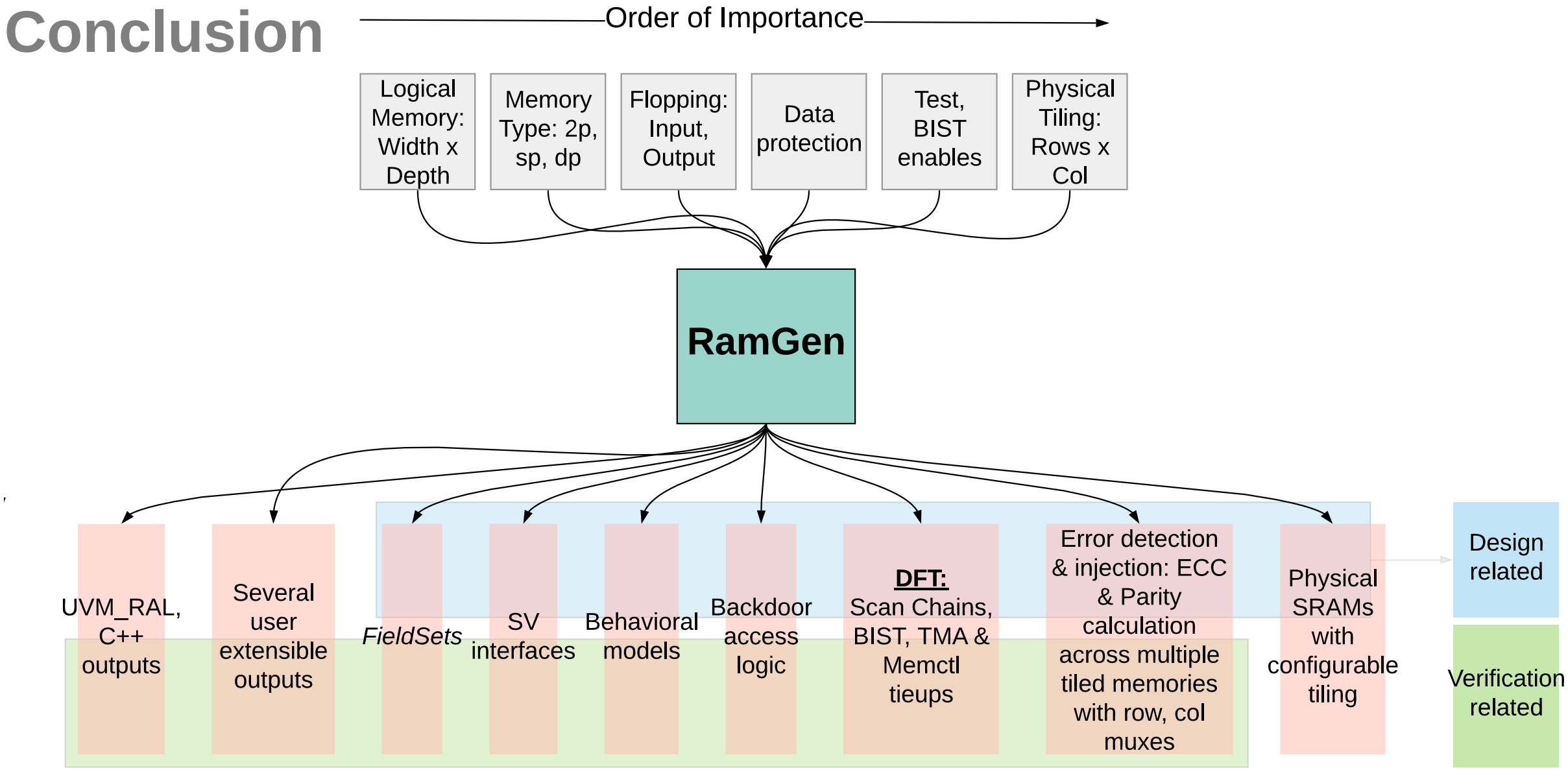
Inner workings



Evidence

- For one of our complex designs, in just three days we had every memory defined, with protection, register access and the whole interrupt structure around it. Working with our other tools, the necessary uvm_regs, C++ and RTL models were also generated and successfully tested.
- We’ve tested this across several tape-outs and different memory vendors across multiple nodes. Designs have scaled well and enabled easy re-use.

Conclusion


Order of Importance
Logical Memory: Width x Depth
Memory Type: 2p, sp, dp
Flopping: Input, Output
Data protection
Test, BIST enables
Physical Tiling: Rows x Col
RamGen
UVM_RAL, C++ outputs
Several user extensible outputs
FieldSets
SV interfaces
Behavioral models
Backdoor access logic
DFT:
Scan Chains, BIST, TMA & Memctl tieups
Error detection & injection: ECC & Parity calculation across multiple tiled memories with row, col muxes
Physical SRAMs with configurable tiling
Design related
Verification related

Conclusion

- Our approach here is an eco-system to enable rapid development. It's a way of defining a memory network, which is vendor agnostic.
- We take a specification and refine it over time to make the output more accurate over time.
- It automates front end memory generation. It has a set of models, makes it very easy to switch between models while the primary memory interface signals remain the same.
- We separate the Physical design decisions from the development flow. This leads to reduced code maintenance, less fragility and more portability.

Thank You

jscott@juniper.net
jsadowsky@juniper.net
jigar.savla@gatech.ed