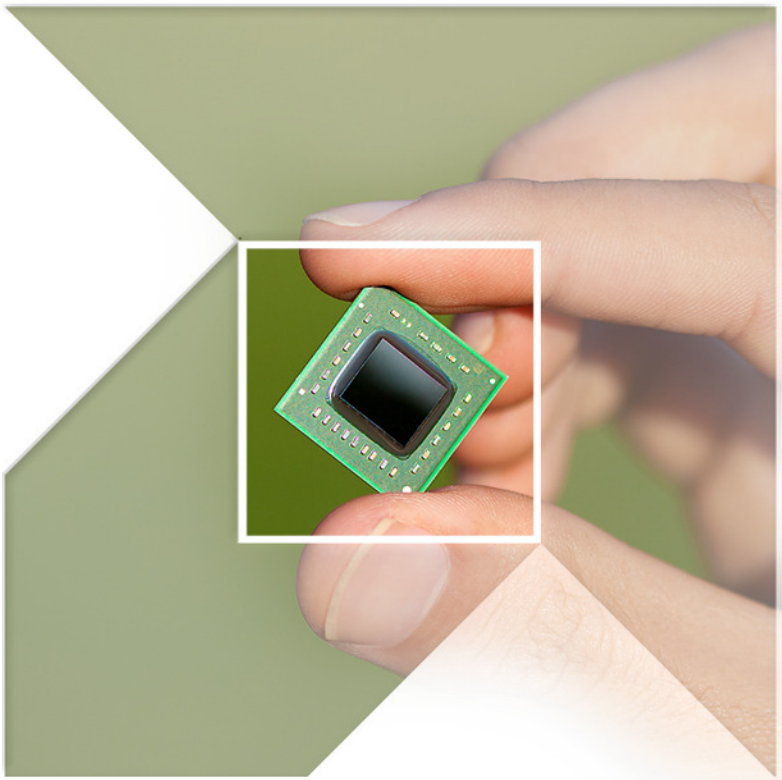


# *DEPLOYING PARAMETERIZED INTERFACE WITH UVM*

**Wayne Yun**
**Senior member of Technical Staff, AMD, Inc.**
**wayne.yun@amd.com**

**Shihua Zhang**
**Senior ASIC Engineer, AMD, Inc.**
**shihua.zhang@amd.com**

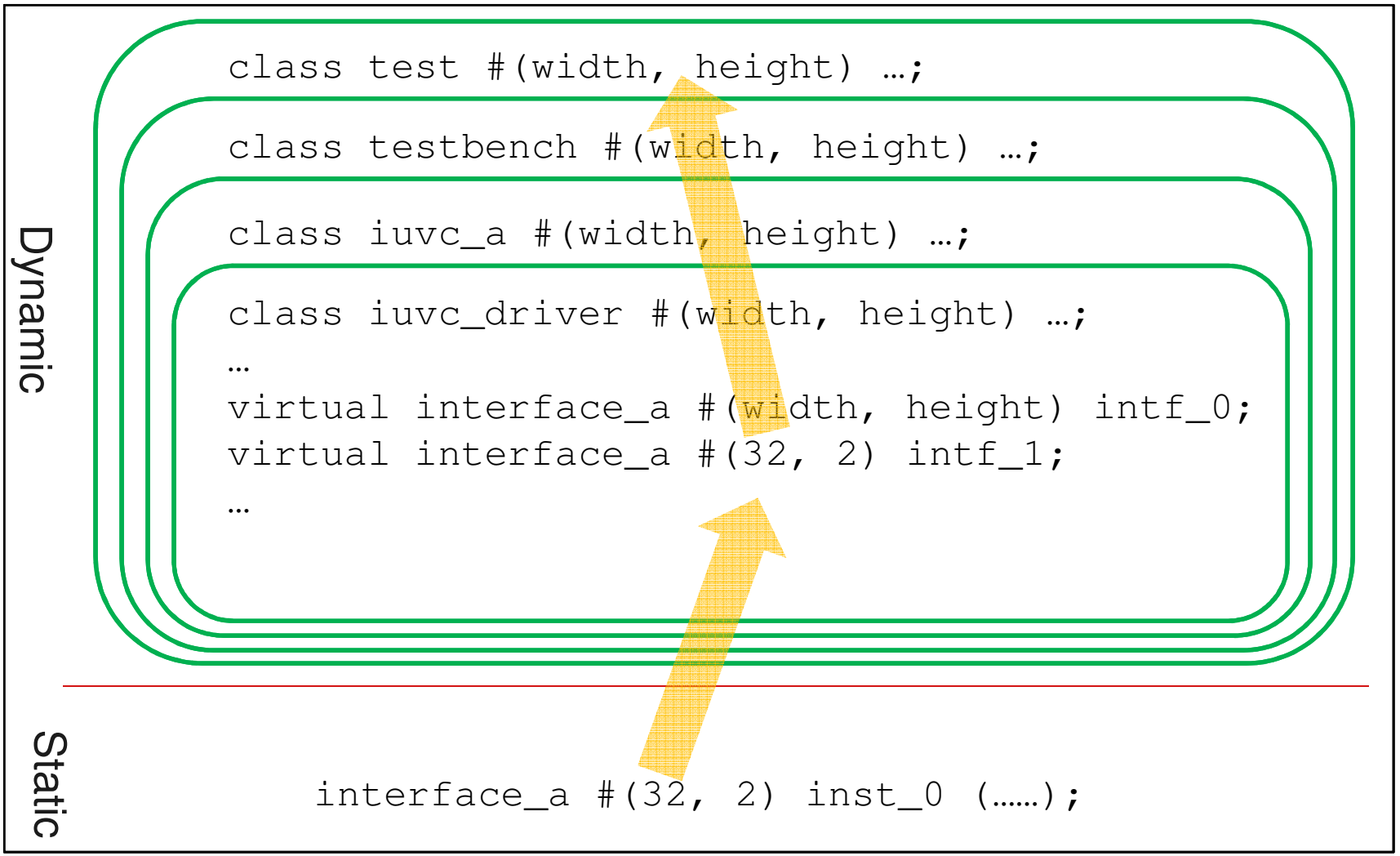
**February 25, 2013**
**DV Conference 2013**
**San Jose, Calif., United States**

AMD

# AGENDA

- The Issue of Parameters on Interfaces
- Workarounds
- Object-oriented Programming
- Class-abstracting Interface
- The Issue Solved
- Implementation Styles
- Abstracting at Different Levels
- Structure Not Working
- Summary

# THE ISSUE OF PARAMETERS ON INTERFACES




- Parameters on interface propagate to top-level UVM TB
  - Dynamic configuration object cannot be used for parameters
- Compromise scalability, re-usability, and flexibility

# *WORKAROUNDS*

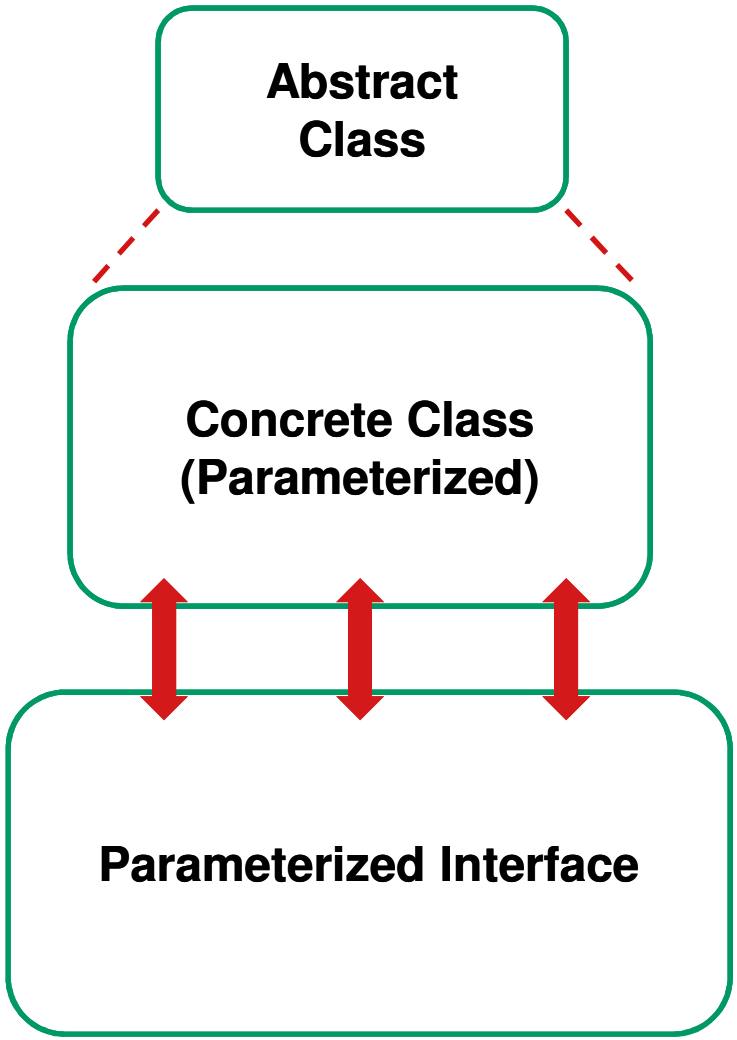
- Parameterizing all classes
  - Requires higher-level testbench components to have detailed knowledge of lower-level details
  - Not quite compatible with encapsulation concept from OOP
- Defining super-set signals
  - Having every signal from every possible variant, no parameter on interface
  - Overhead, especially when difference in number of signals is large
- Sub-interface
  - Only for interfaces composed of groups of signals
  - Difficulty increases when groups are not identical
- Workarounds are not scalable
  - A fundamentally different solution is needed

# *OBJECT-ORIENTED PROGRAMMING*

- In OOP, classes are used to implement abstraction
- A base class can define virtual methods or functions
- The derived class should implement these virtual methods or functions
- The instance of a derived class can be used as one of a base class
- Such a common set of operational functions defined at base class can be used to operate on different derived classes
- SystemVerilog supports OOP
  - Class can be used to abstract parameterized interfaces

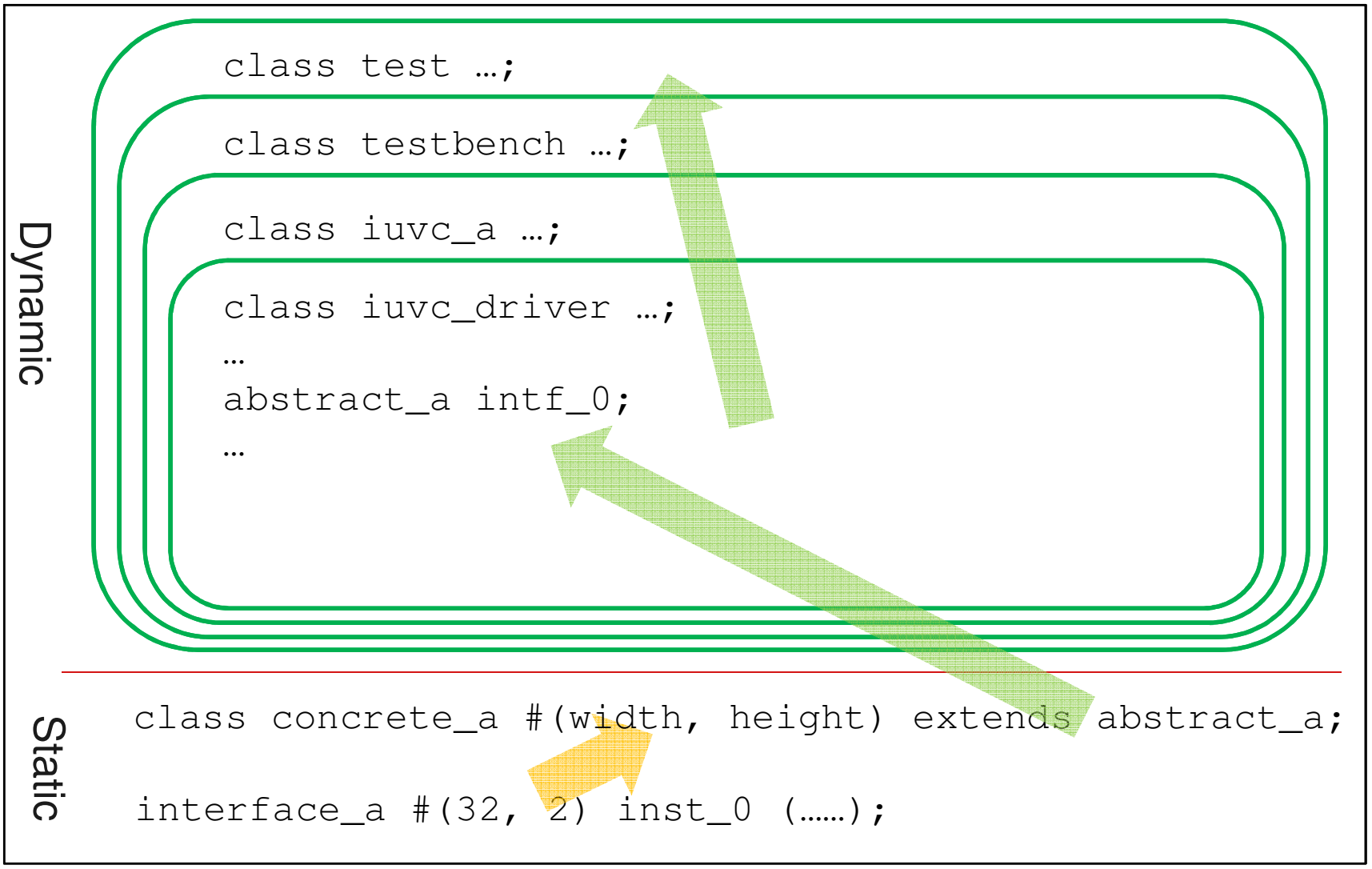
```
class BASE;

  pure virtual function void
funct_0();

endclass

class DERIVED extends BASE;

  function void funct_0();

      // different implementation

  endfunction

endclass


BASE b;

DERIVED d = new;


b = d; // instance of derived used
as base
```

# *CLASS-ABSTRACTING INTERFACE*




- Abstract class defines functions representing all operations
- Abstract class is not parameterized
- A concrete class is derived from an abstract one
- Concrete class is aware of the interface and its parameters
- Such functions -- defined at abstract class and implemented at concrete class -- will operate on parameterized interface, but provide a non-parameterized look

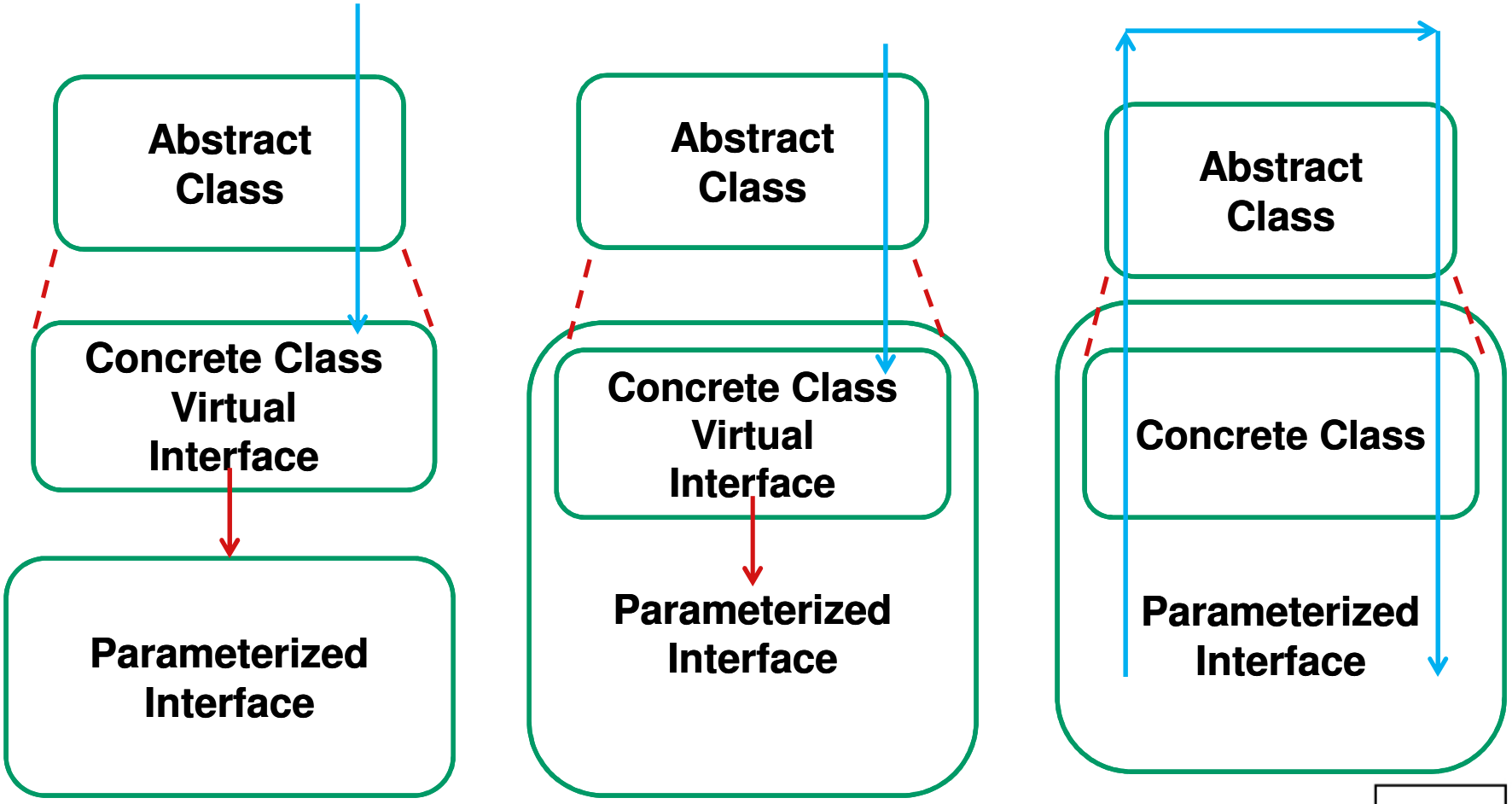
# THE ISSUE SOLVED




- Parameters on interface are confined to lower layers
- Scalability, re-usability, and flexibility

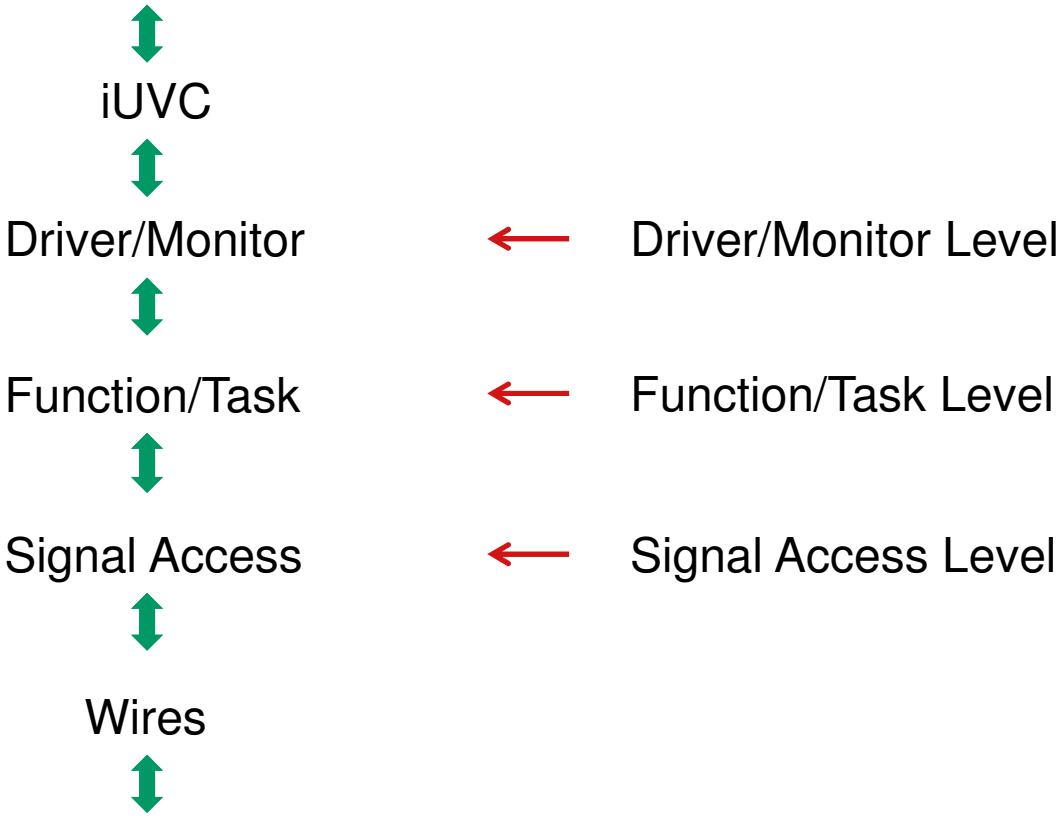
# *IMPLEMENTATION STYLES*

- Stand-alone concrete class with virtual interface
- Concrete class with virtual interface inside
- Concrete class inside without virtual interface



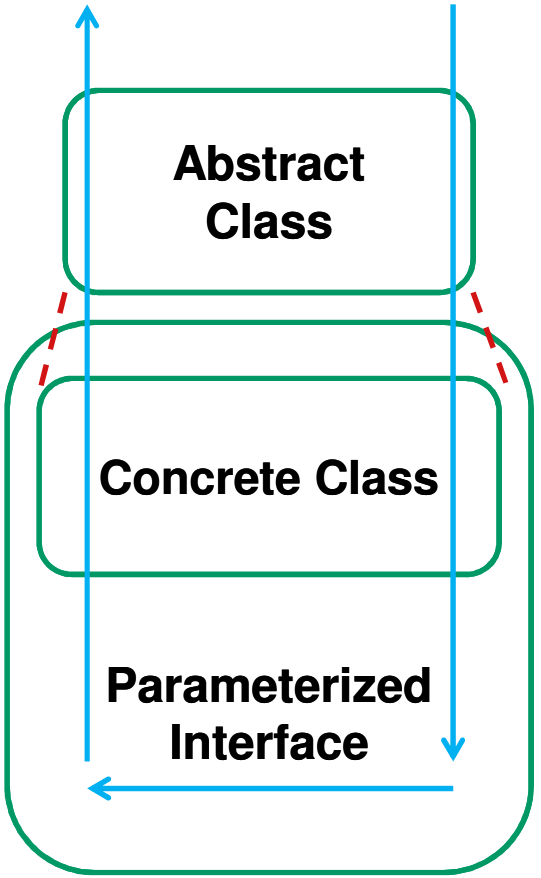
# *ABSTRACTING AT DIFFERENT LEVELS*

Abstraction can be implemented at different functional levels



# *STRUCTURE NOT WORKING*

- One structure was found not working
  - When concrete class is inside interface, signals are accessed without virtual interface and thread of task/function is started outside.
  - If multiple instances are created, signals are not driven correctly.
- The reason of why it doesn't work remains to be found.



# *SUMMARY*

- UVM recommends virtual interface to be used at driver and monitor to access signals defined at static part of testbench.
- When the interface is parameterized, exact parameters are required at virtual interface declaration.
- This most likely will cause parameters to be defined at higher-layer classes.
- Such a testbench structure is not scalable.
- This issue is a significant obstacle for testbenches of configurable designs.
- SystemVerilog doesn't define syntax for interface abstraction.
- Classes can be leveraged to abstract parameterized interfaces.
- Utilizing methods described previously, UVCs and uvm_envs are not parameterized.
- Dynamic testbenches can be designed following OOP and become scalable, re-usable, and flexible.
- All Examples were run with VCS.

AMD

**Trademark Attribution**

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.