2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Maximizing Verification Productivity Using UVM and Dynamic Test Loading

Masayuki Masuda
Renesas Electronics Corporation

RENESAS

accellera SYSTEMS INITIATIVE

Table of Contents

1. Background

Background

Verification engineers involved in debugging tests are always interested in improving verification productivity. Some reports show that the median percentage of IC/ASIC project time spent in verification is 50-60% [1].

What means are effective to achieve this?

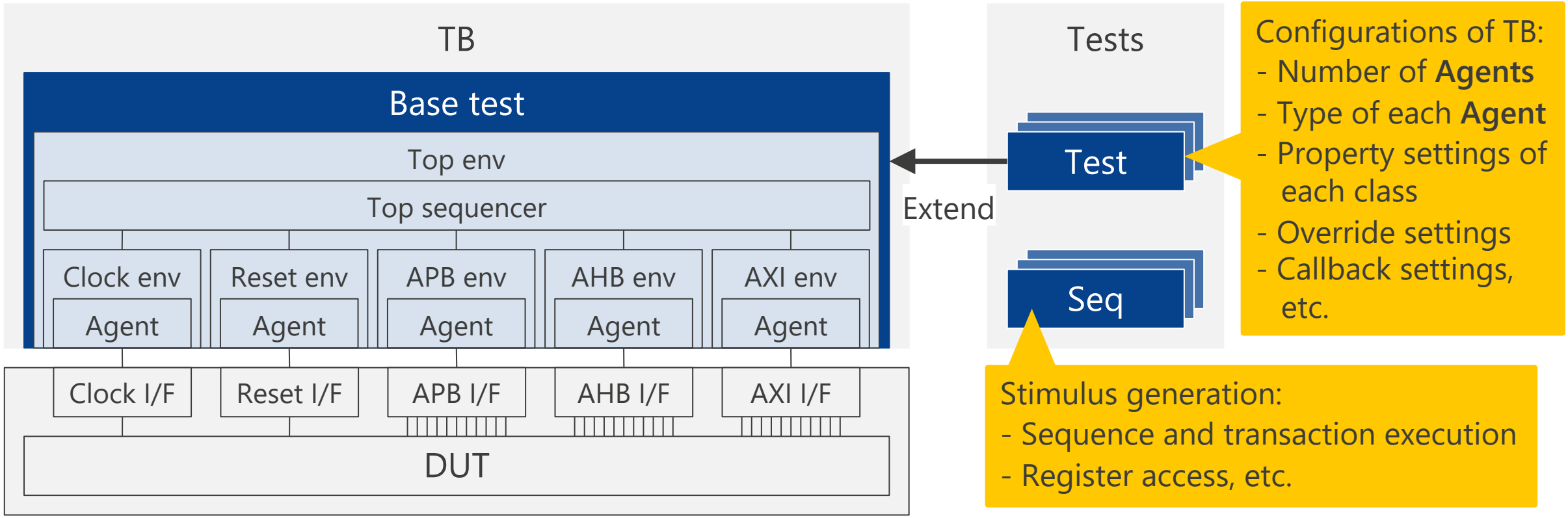
- Universal Verification Methodology (UVM) enables verification reusability and interoperability within companies and throughout the electronics industry.
- Dynamic Test Loading (DTL) including Save/Restore (S/R), which is an EDA technology supported by major HDL simulators, reduces compile and simulation run time.

The introductions and benefits of UVM and DTL in terms of verification productivity are discussed below.

Introduction of UVM (1/3)

*1 Classes are shown in **bold**.

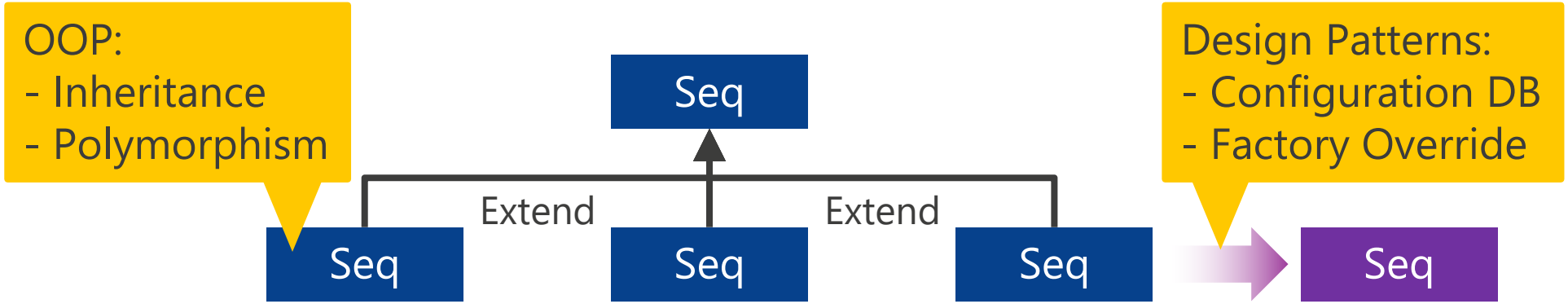
UVM specifies the class architecture of testbenches (TBs) and test scenarios (hereinafter simply tests). The tests consist of **Test** and **Sequence (Seq)** classes*1 and have the following roles.



Introduction of UVM (2/3)

UVM offers optimization of the class structure using Object-Oriented programming (OOP) and design patterns:

- Faster test development TAT by reducing code size
  - Share a common part of the classes as a base (or super, parent) class
  - Implement only unique parts of the classes as derived (or sub, child) classes
  - Reconfigure the parts of the classes

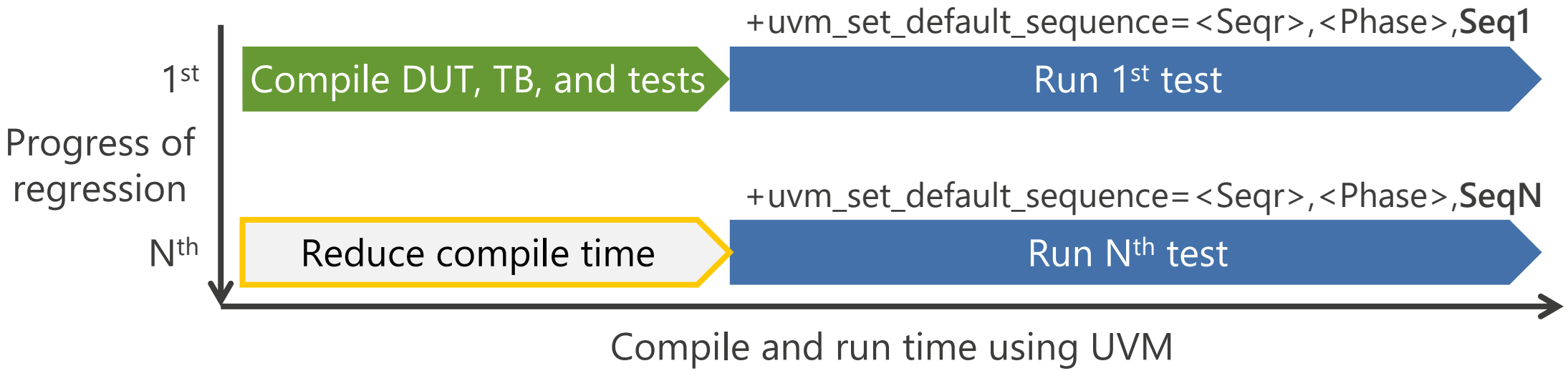


The inheritance relationships can be extended any number of times.

Introduction of UVM (3/3)

UVM also offers run-time test selection using the built-in Command Line Processor (CLP):

- Faster regression TAT by reducing compile time
  - **Test** (resp. **Seq**) can be specified via the +UVM_TESTNAME (resp. +uvm_set_default_sequence)
  - No recompilation occurs in regression



However, if there are any updates to the tests, the recompilation occurs.

Benefits of UVM

*1 A phase in which tests are updated.
*2 A phase in which no tests are updated.

Benefit	Test development *1	Regression *2
1) Reduction in code size	✓	N/A
2) Reduction in compile time	N/A	✓
3) Reduction in simulation run time	N/A	N/A

Verification process

Verification Issues

*1 A phase in which tests are updated.
*2 A phase in which no tests are updated.

Benefit	Test development *1	Regression *2
1) Reduction in code size	✓	N/A
2) Reduction in compile time	N/A (#2)	✓
3) Reduction in simulation run time	N/A (#3)	N/A (#3)

Verification issues
that cannot be solved by UVM alone

Verification process

1) Reduction in code size

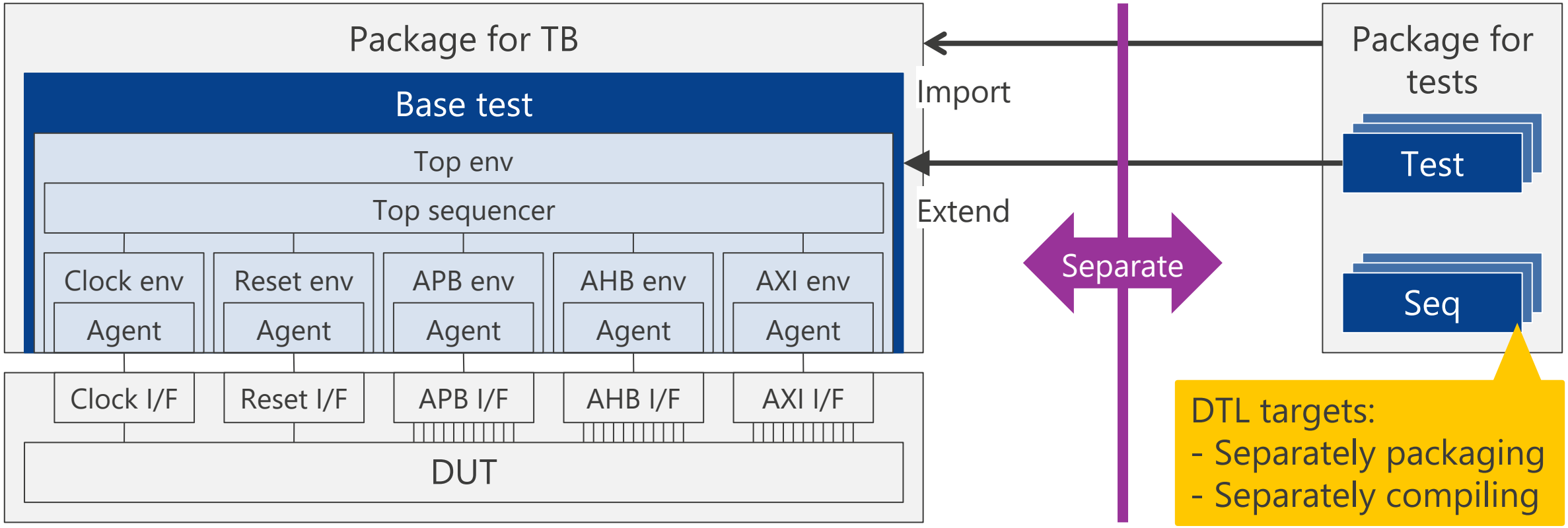
Env. dev. → Test development *1

Regression *2

2) Reduction in compile time

Introduction of DTL (1/3)

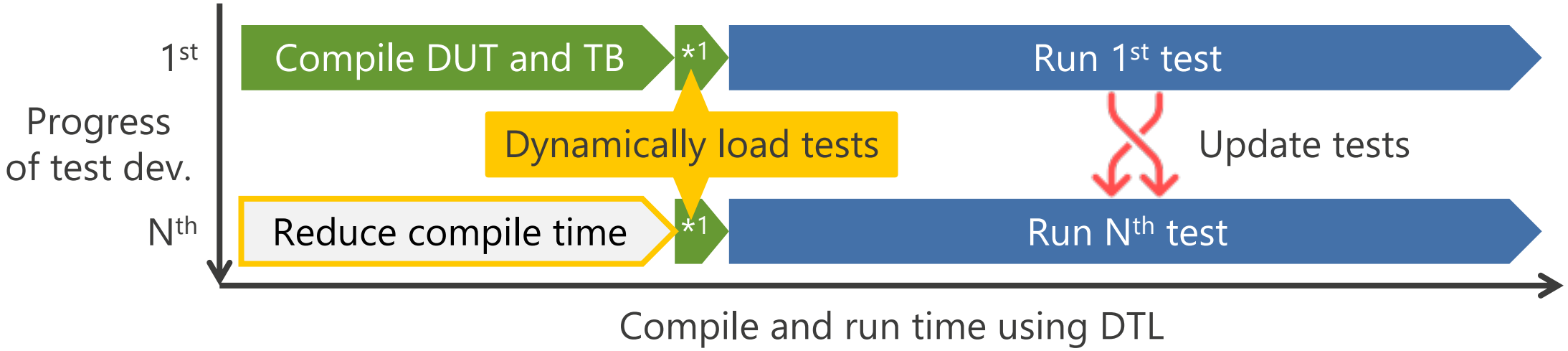
DTL enables dynamically loading the tests at the run-time. This implies that the tests, which are DTL targets, shall be packaged and compiled separately from the DUT and TB.



Introduction of DTL (2/3)

DTL offers a solution for the verification issue #2:

- Faster test development TAT by reducing compile time
  - After compiling the DUT and TB once, compiles and loads only the tests
  - No recompilation of entire DUT and TB for each test

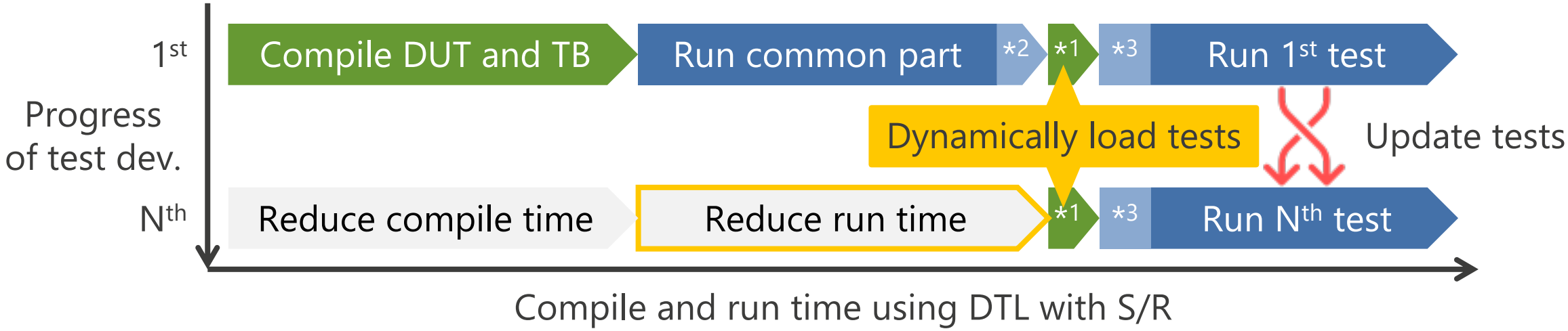


*1 Compile and dynamically load tests

Introduction of DTL (3/3)

DTL also offers a solution for the verification issue #3:

- Faster test development and regression TAT by reducing simulation run time
  - Dynamically loads the tests with S/R
  - No repetition of the common part (e.g., reset, initialization, link training, etc.) for each test



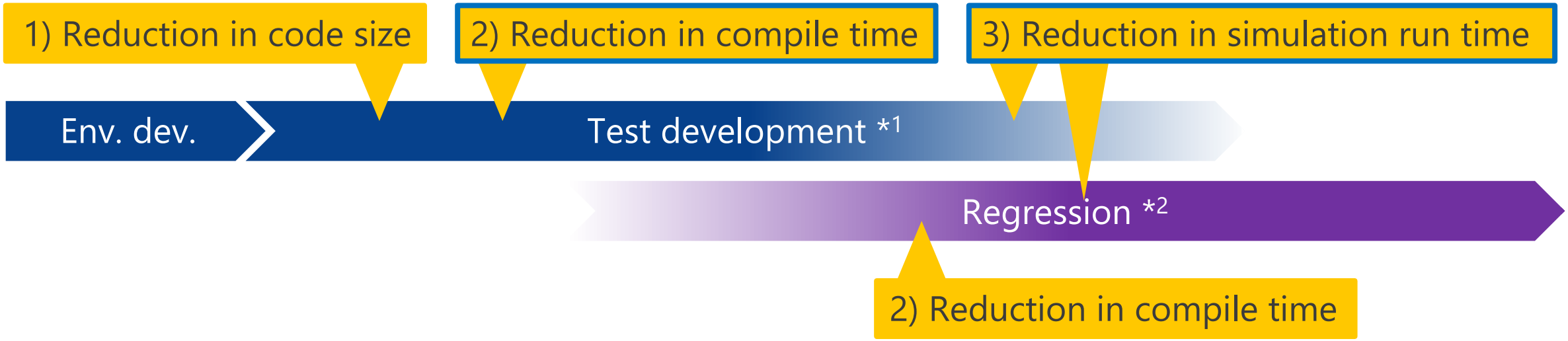
*1 Compile and dynamically load tests, *2 Save the simulation state, *3 Restore the simulation state

Benefits of DTL

*1 A phase in which tests are updated.
*2 A phase in which no tests are updated.

Benefit	Test development *1	Regression *2
1) Reduction in code size	✓	N/A
2) Reduction in compile time	✓	✓
3) Reduction in simulation run time	✓	✓

Verification process



2. Challenges

Barriers to UVM and DTL Deployment

UVM and DTL are highly effective means of improving verification productivity as discussed above.

However, only a few engineers can take advantage of them for the following reasons:

- Need to understand both UVM and DTL in depth
  - They are based on OOP and design patterns but are poorly guided.
- Difficult to share one practice with other cases
  - They have many options on how to use and are not aligned among EDA vendors.

In fact, the past DVCon tutorial have pointed out such difficulties of UVM [2].

Challenges

What are needed to break down the barriers and maximize verification productivity?

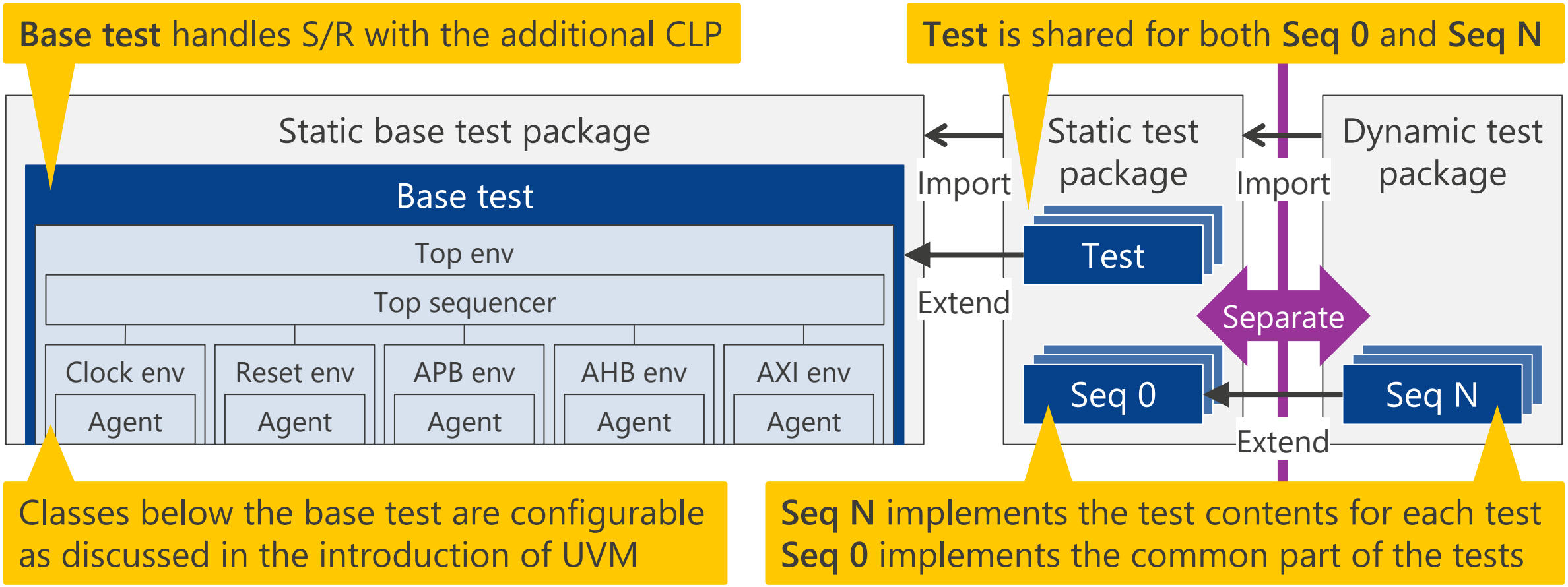
- Simplified and standardized architecture for applying UVM and DTL
  - It shall offer command lines consistent with or without the use of S/R. (Today's focus)
  - It shall be simulator independent.
- Effective infrastructure to spread UVM and DTL

Our challenges are to establish them and to demonstrate their applicability and capability.

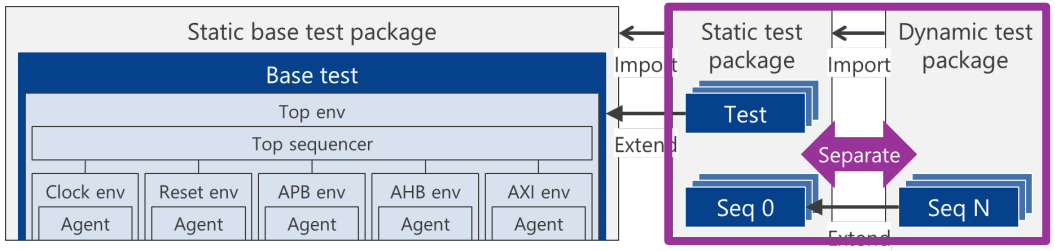
3. Solutions

UVM and DTL Architecture (1/3)

The proposed architecture of UVM classes and packages for DTL shall follow.



UVM and DTL Architecture (2/3)



The following class structure is intended to be consistent with or without the use of S/R:

- **Test** is shared for both **Seq 0** and **Seq N**. Also, it doesn't have any default sequence settings and sequence executions. Instead, the default sequence shall be specified via the CLP.
- **Seq 0** shall implement the common part to be run at reset_phase.
- **Seq N** shall extend **Seq 0** and implement the test contents to be run at main_phase.

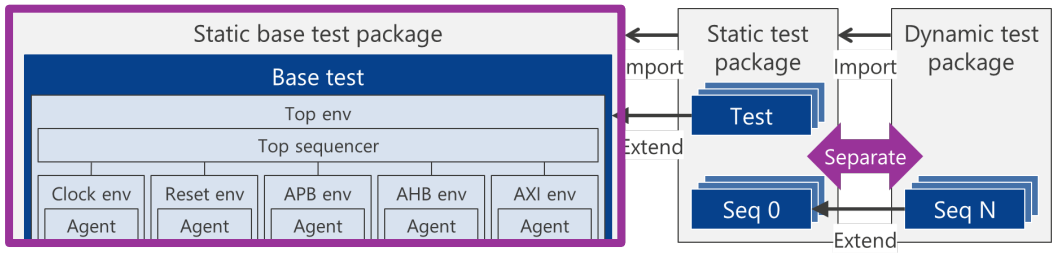
When restoring the simulation, skip the common part

```
class Seq0 extends uvm_sequence;
  virtual task body();
    if ($test$plusargs("RESTORE")) return;
    super.body();
    //Common Part
  endtask : body
endclass : Seq0
```

```
class SeqN extends Seq0;
  virtual task body();
    super.body();
    //Test Contents
  endtask : body
endclass : SeqN
```

Seq 0 followed by **Seq N**

UVM and DTL Architecture (3/3)



UVM has the built-in CLP that handles +uvm_set_default_sequence, but it doesn't work when restoring the simulation since the time has already passed to post_reset_phase:

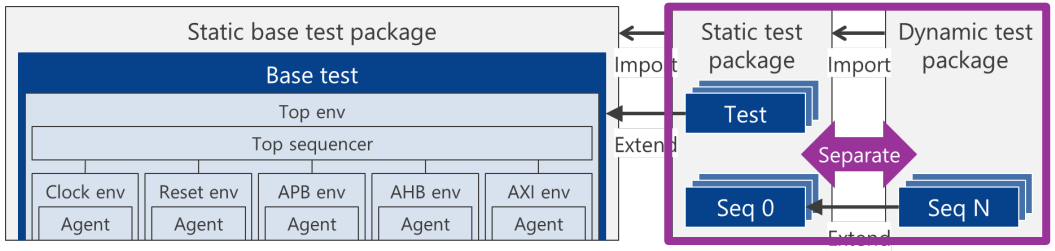
- **Base test** shall handle +uvm_set_default_sequence with the following CLP in post_reset_phase.

```
virtual task post_reset_phase(uvm_phase phase);
  if ($value$plusargs("SAVE=%s", snapshot)) begin
    $save(snapshot);
  end
  if ($test$plusargs("RESTORE")) begin
    if ($value$plusargs
        ("uvm_set_default_sequence=uvm_test_top.top_env.top_seqr,main_phase,%s", seqname)) begin
      $cast(seq, create_object(seqname));
      uvm_config_db#(uvm_object_wrapper)::
        set(this, "top_env.top_seqr.main_phase", "default_sequence", seq.get_object_type());
    end
  end
end task : post_reset_phase
```

Additional CLP using $value$plusargs

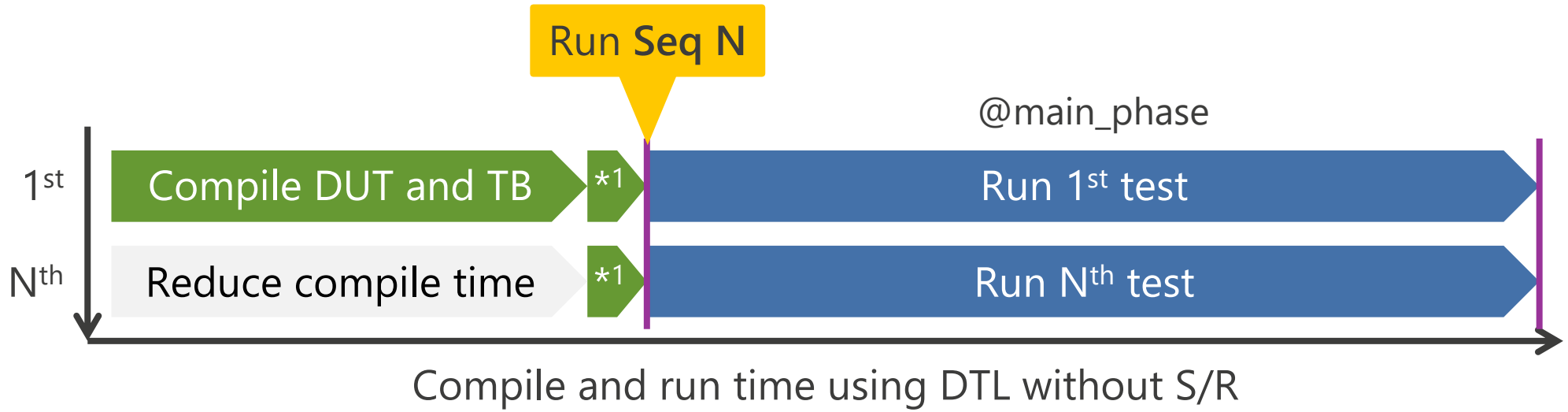
Set **Seq N** to the default sequence of main_phase

Command Lines without S/R



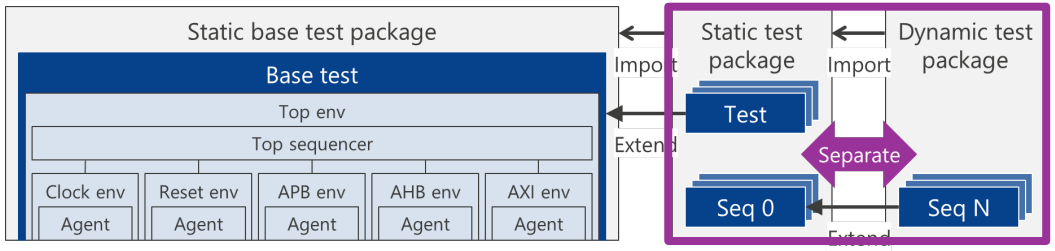
The proposed architecture offers the consistent command lines with or without S/R. Also, the architecture and command lines are simulator independent (confirmed by 2 major HDL simulators):

- Run the 1st or Nth test with **Test** and **Seq N** at main_phase
  +UVM_TESTNAME=**Test** +uvm_set_default_sequence=<Seqr>,main_phase,**SeqN**

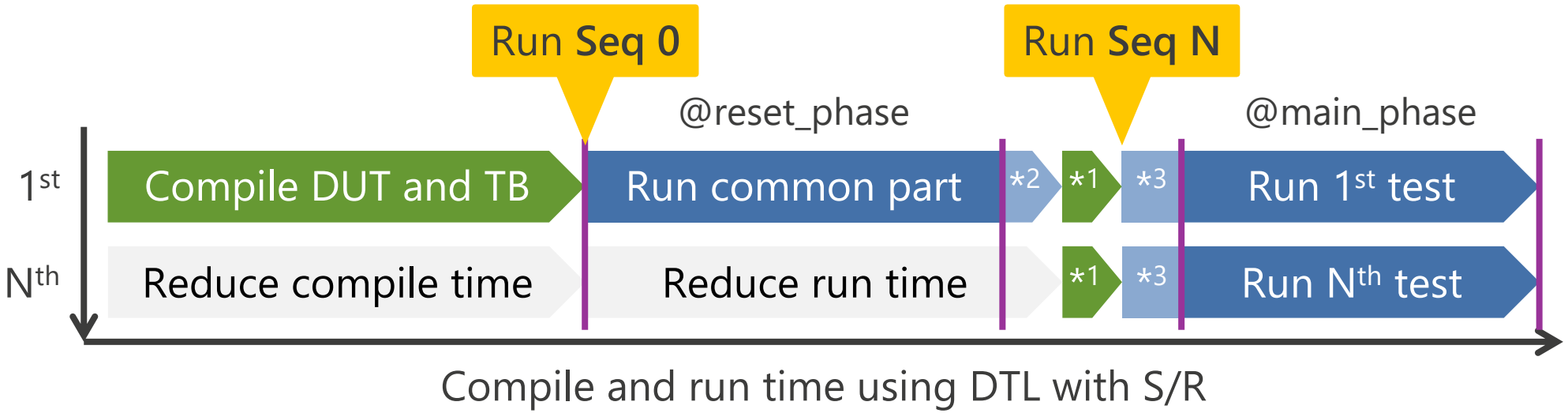


*1 Compile and dynamically load tests

Command Lines with S/R



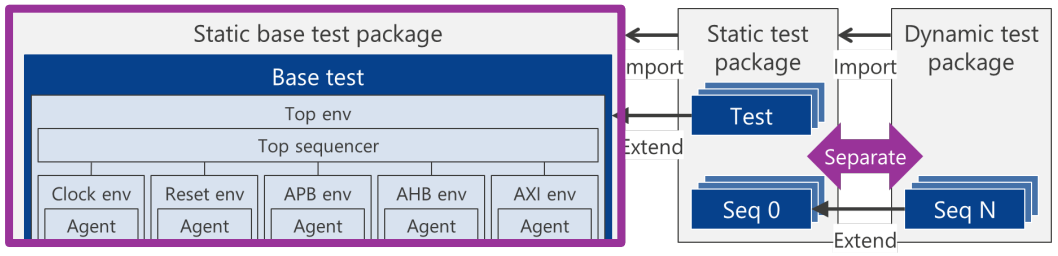
- Run the common part with **Test** and **Seq 0** at reset_phase
  +UVM_TESTNAME=**Test** +uvm_set_default_sequence=<Seqr>,reset_phase,**Seq0** +SAVE=<SnapShot>
- Run the 1st or Nth test with **Test** and **Seq N** at main_phase
  +UVM_TESTNAME=**Test** +uvm_set_default_sequence=<Seqr>,main_phase,**SeqN** +RESTORE -r <SnapShot>



Compile and run time using DTL with S/R

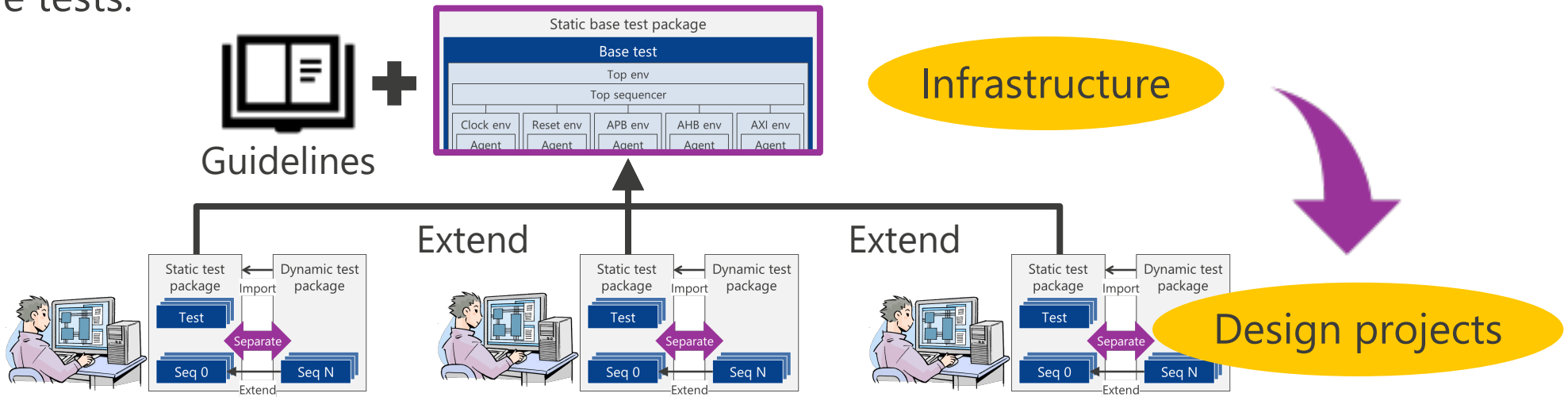
*1 Compile and dynamically load tests, *2 Save the simulation state, *3 Restore the simulation state

Infrastructure



Guidelines for compliance with the architecture and the command lines are essential as the infrastructure. In addition, the distribution of the base test package including the configurable testbench and the S/R handler is effective:

- UVM and DTL are expanded rapidly to many design projects.
- The verification engineers of each design project save the cost of introducing them to focus on debugging the tests.



4. Results

Results

The proposed solutions were implemented and applied to 3 designs and various tests where S/R was available, with the following results:

- The reduction in compile time exceeded 90% across all designs.
- The reduction in run time varied for each test and reached up to 99%.

Compile time					Run time				
Design	Scale	w/o DTL	w/ DTL	Reduction	Test	Length	w/o S/R	w/ S/R	Reduction
1	Block	00:04:17	00:00:22	92%	A	Typical	00:03:17	00:01:04	67%
2	Chip	00:12:34	00:00:16	98%	B	Typical	00:04:30	00:02:42	40%
					C	Long	00:15:48	00:03:36	77%
3	Block	00:06:41	00:00:16	96%	D	Very long	15:44:32	00:06:43	99%
					E	Very long	21:18:27	08:46:18	59%

Contribution to Actual Verification

Reduction of total compile time in test development:

- Design 1-3: Reduction by 92-98% (This is because it can be applied to 100% of tests.)

Reduction of total simulation run time per regression:

- Design 1: Reduction by 200 hours (67%)
  (Total run time per regression (300 hours) x Average reduction rate per test (67%) x Applicable tests (100%) = 200 hours)
- Design 2: Reduction by 300 hours (20%)
  (Total run time per regression (1500 hours) x Average reduction rate per test (40%) x Applicable tests (50%) = 300 hours)

Reduction of total verification period:

- Design 3: Reduction by 1 month (36%) (Details of the estimation formula are omitted.)

The results demonstrate the applicability and capability of our solutions for maximizing verification productivity.

5. Conclusion

Conclusion

We established the following architecture and infrastructure:

- Simplified and standardized architecture for applying UVM and DTL
- Effective infrastructure to spread UVM and DTL

The following results demonstrated the applicability and capability of our solutions for maximizing verification productivity:

- Reduced total compile time in test development by 92-98%
- Reduced total simulation run time per regression by 200 hours (67%) or 300 hours (20%)
- Reduced total verification period by 1 month (36%)

References

References

[1] 2022 Wilson Research Group Functional Verification Study
https://blogs.sw.siemens.com/verificationhorizons/2022/12/12/part-8-the-2022-wilson-research-group-functional-verification-study/

[2] DVCon 2019 Tutorial: IEEE 1800.2 UVM - Changes - Useful UVM Tricks & Techniques
https://accellera.org/images/resources/videos/Tutorial-IEEE-1800-2-Standard-for-UVM-2019.pdf

Abbreviations

Abbreviations

- UVM: Universal Verification Methodology
- DTL : Dynamic Test Loading
- S/R : Save/Restore
- EDA : Electronic Design Automation
- HDL : Hardware Description Language
- DUT : Device (or Design) Under Test
- TB : Testbench
- Seq : Sequence
- OOP : Object-Oriented Programming
- CLP : Command Line Processor

Questions

Questions