ENCS5337: Chip Design Verification

Spring 2024/2025

Stimuli Generation

Dr. Ayman Hroub

Many thanks to Dr. Kerstin Eder for most of the slides

Outline

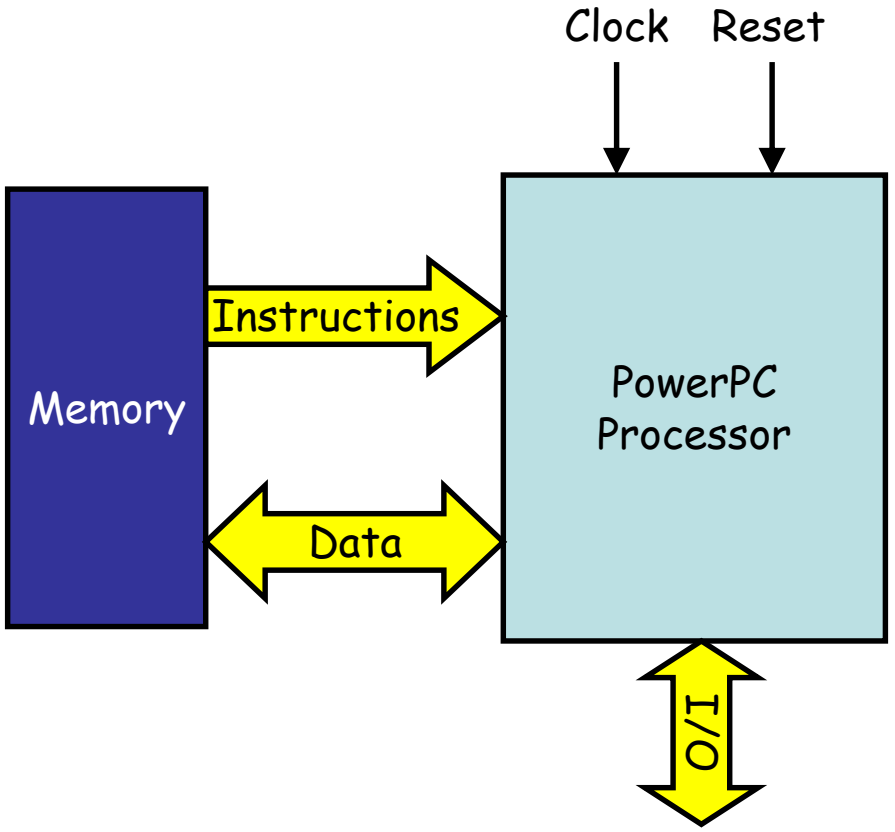
- **Motivation:** Advanced Stimulus Generation
- **Example:** PowerPC processor
- Issues in stimuli generation
  - Level of stimuli, test length, etc.
- Randomness
- Constrained pseudo-random stimulus generation

Goals of Stimuli Generation

- Achieve all the items in the test scenarios matrix of the verification plan
  - Ensure that the scenarios in the matrix are happening
  - Ensure that bugs are propagating to an existing checker
    - Hitting a bug without exposing it is worth nothing
- But also
  - Hitting and exposing all the problems we did not think about in the verification plan
  - Ensure that nothing gets broken over time

Example – PowerPC Processor

- Black box view
  - Interface to memory (via caches)
    - For instruction fetching
    - For data fetching and storing
  - Interface to I/O devices
    - For data fetching and storing
    - Interrupts
  - Miscellaneous interface
    - Clocks
    - Reset
    - …

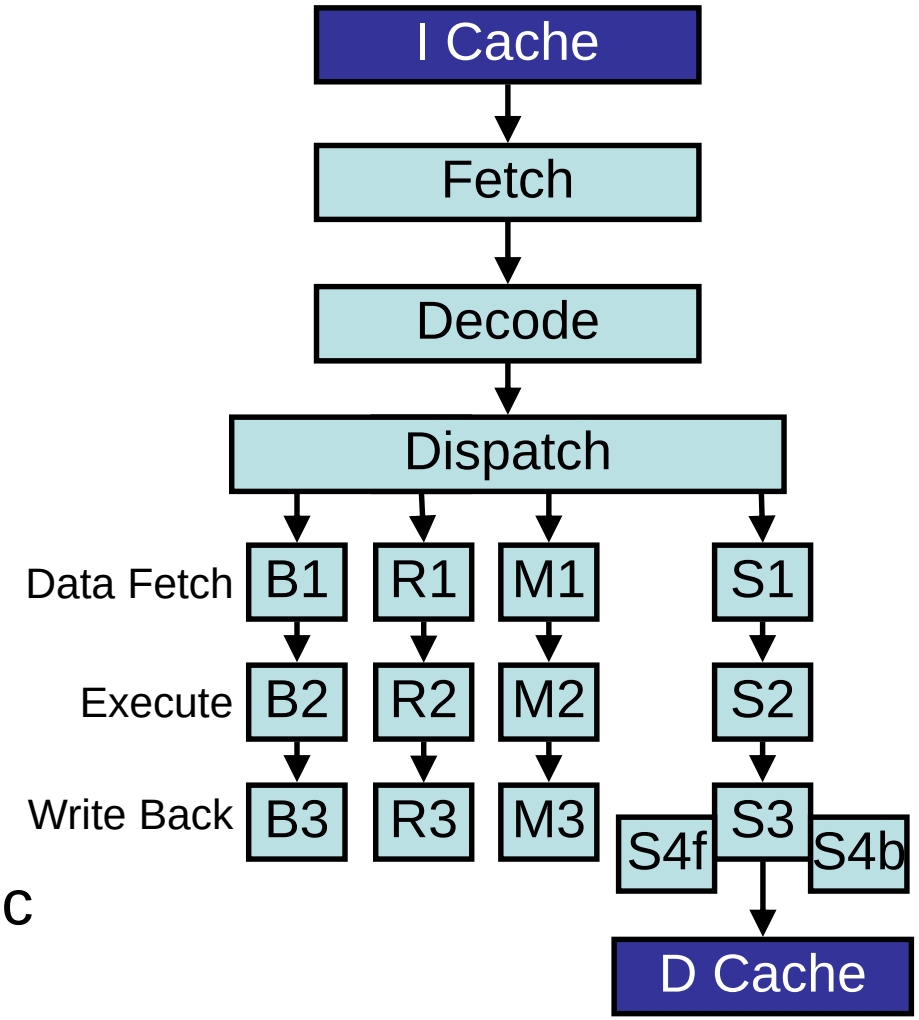


Architectural View

- RISC (Reduced Instruction Set Computer) processor
  - “Small” number of instructions (~400)
  - One simple operation per instruction
  - Fixed length instructions (32 bits = 1 word)
  - Specific load and store instructions to access memory
    - All other instructions use registers for operands
- Large register files
  - 32 general purpose registers (GPR)
  - 32 floating-point registers (FPR)
    - Used only for floating-point operations
  - Several special purpose registers
    - Condition register, link register, status register, etc.

Microarchitectural View

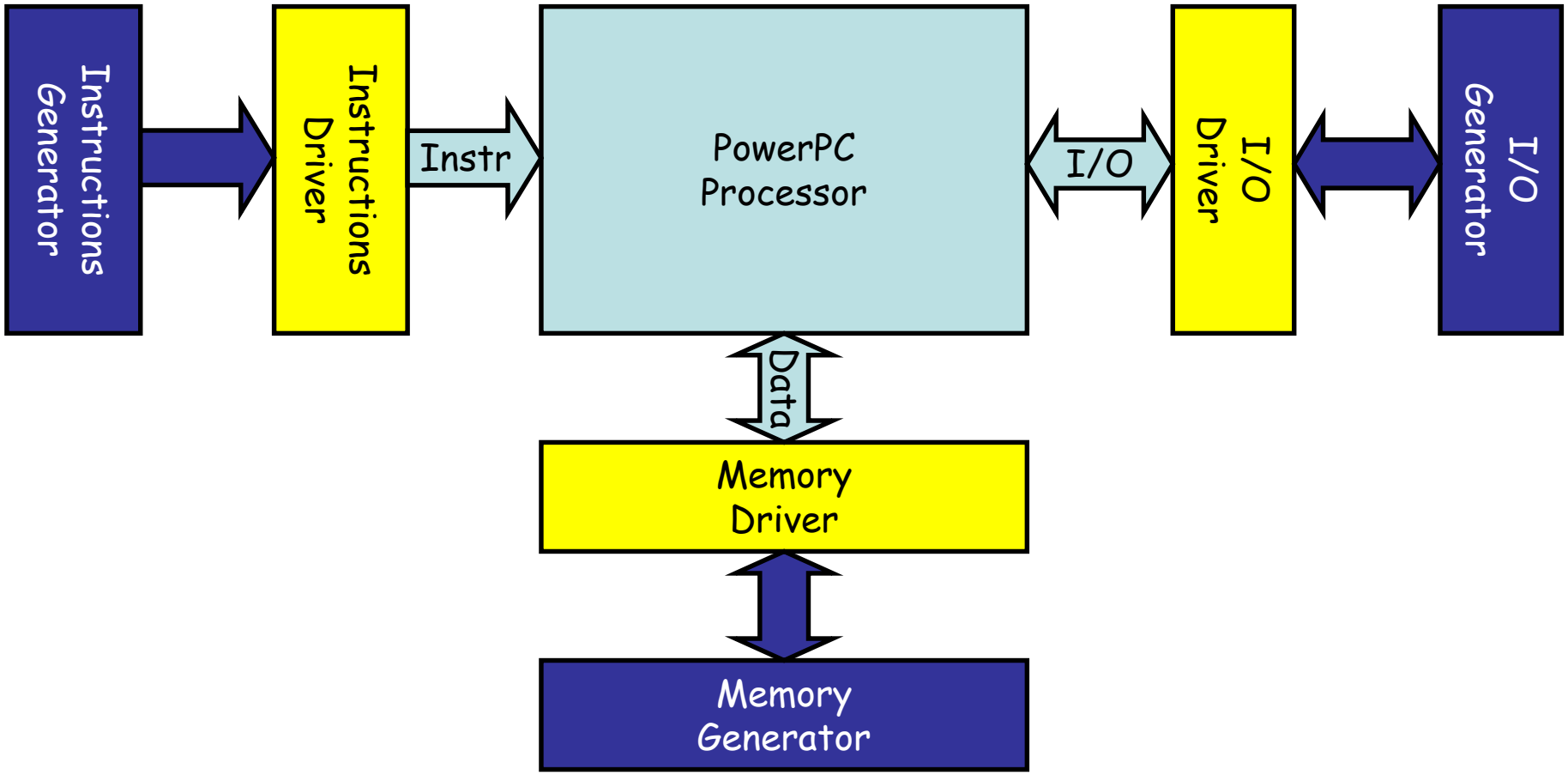
- Multi-threaded
- In-order execution
- Four instructions wide
  - Fetch
  - Decode
  - Dispatch
- Four execution units
  - B – Branch
  - S – Load Store
  - R – Simple Arithmetic
  - M – Complex Arithmetic



Extracts from the Verification Plan

- Check that all pairs of instructions are executed correctly together
  - Basic architectural requirement
  - Appears in most verification plans of processors
- Check that all forwarding mechanisms between pipeline stages are working properly
  - Basic microarchitectural requirement
  - Source for many bugs in previous designs

Processor Verification Environment

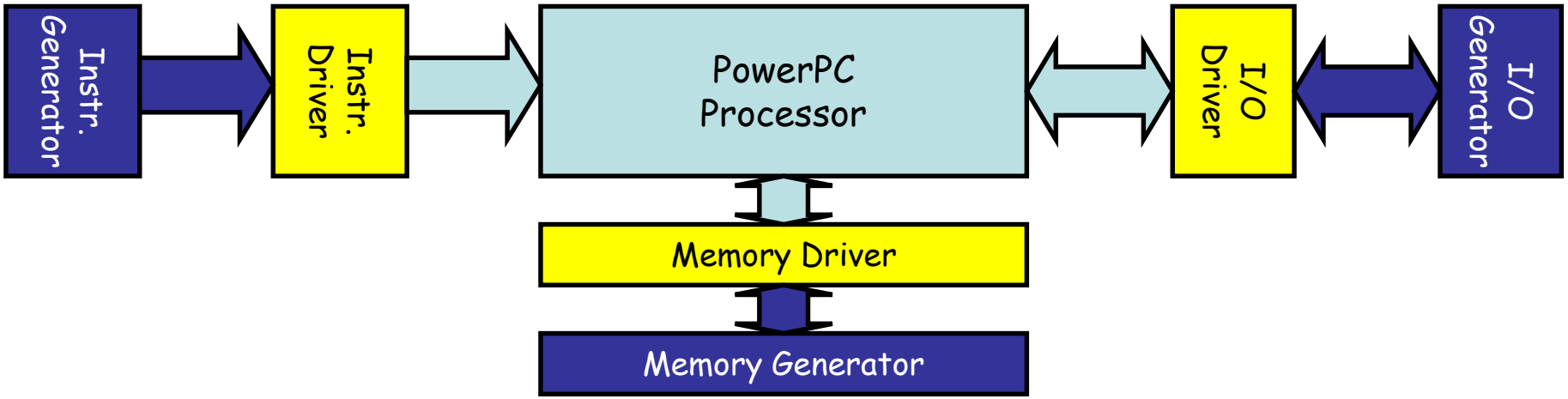


Issues in Stimuli Generation

- How many generators?
- Level of abstraction
- Online vs. offline generation
- Dynamic vs. static generation
- Test length
- Randomness

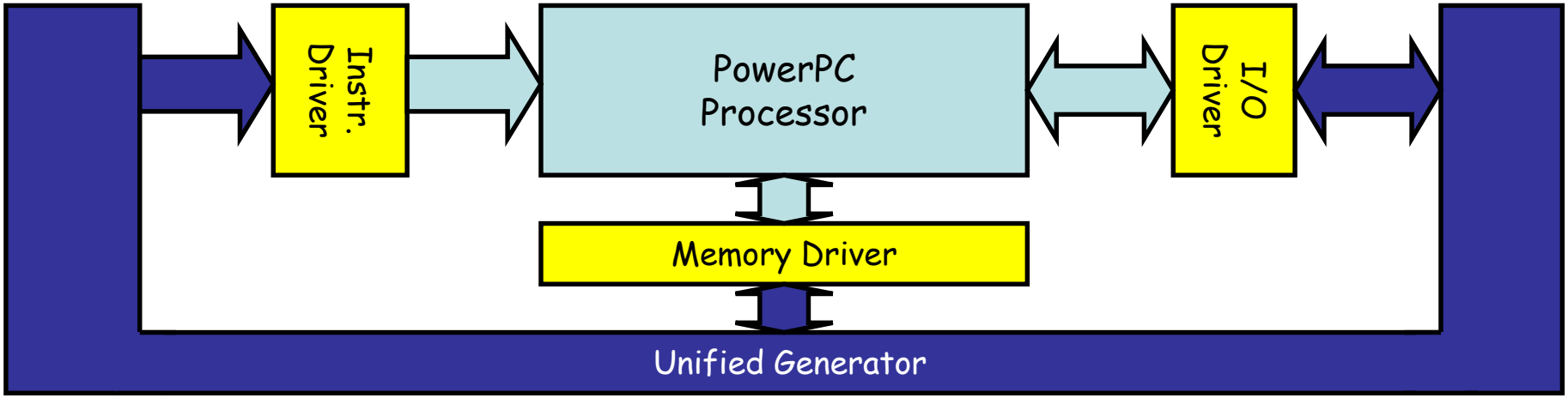
How Many Generators?

- Distributed generators
  - Each interface has its own generator
  - Each generator works on its own
  - Advantages
    - Simple
    - Easy to reuse
  - Disadvantages
    - Hard to reach corner cases in coordinated fashion



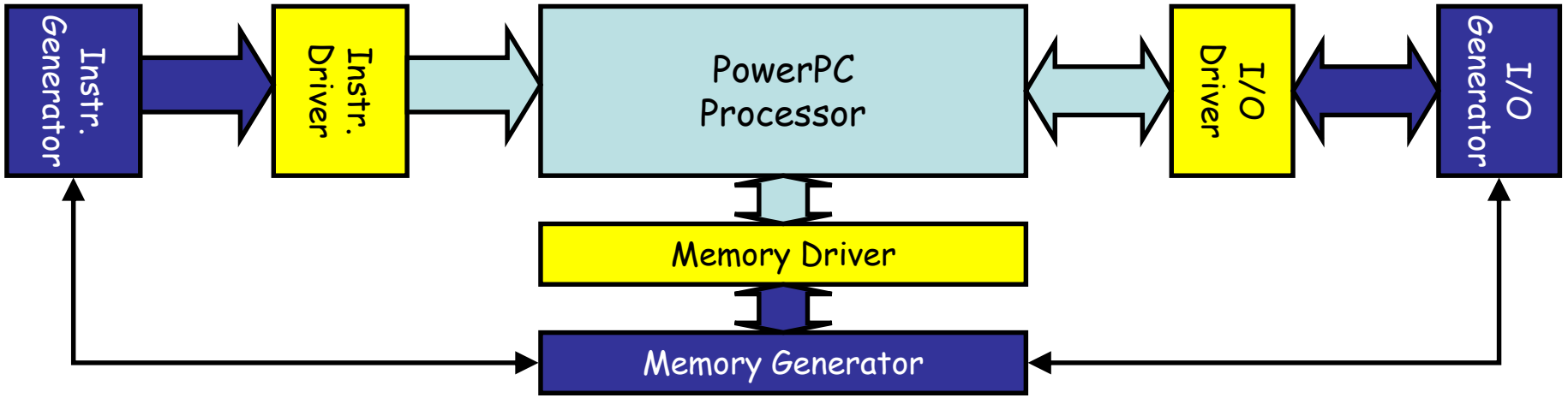
How Many Generators?

- Single generator
  - One generator controls all the interfaces
  - Advantages
    - All the interfaces can work together toward a common goal
  - Disadvantages
    - Complex
    - Hard to reuse

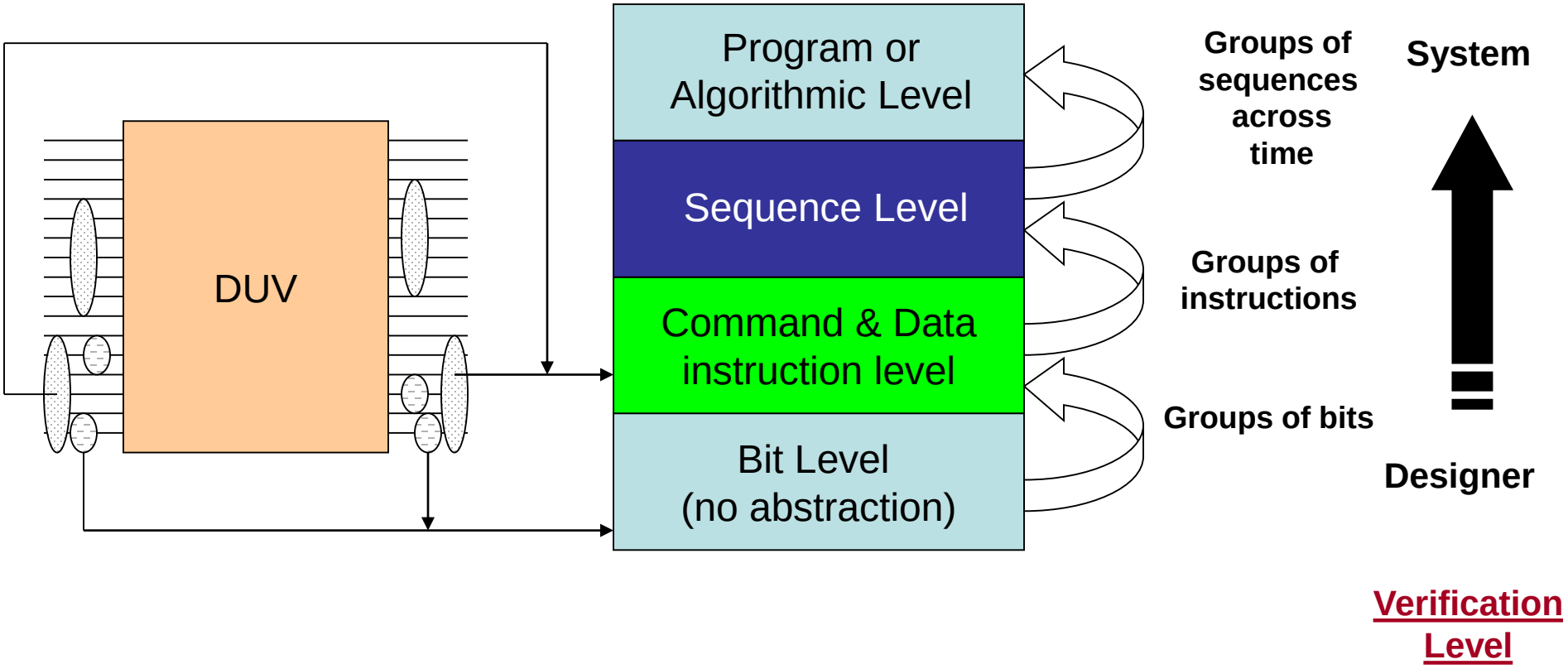


How Many Generators?

- Synchronized generators
  - Each interface has its own generator
  - The generators share information and synchronize
  - Advantages
    - Can reuse each generator separately
    - Can work together towards a common goal



Abstraction Level of Generation



Online Vs Offline Generation

When to generate stimuli?

- **Offline** generation (pre-run):
  - The entire stimuli are generated **before the simulation** begins
  - The generation and simulation can be two separated processes
- **Online** generation (on-the-fly):
  - Stimuli generation **during simulation**
  - The next element is generated when needed by the driver
  - The generator must be part of the verification environment

Offline Generation

- Why
  - Can separate the generation from simulation
    - Use external tools, emulation, …
  - Can use more complex algorithms for generation
    - For example, **generate out of order**, e.g. instruction sequences (processors) or action sequences (robotics)
- Why not
  - Need to connect the generation output to the verification environment
  - Cannot use information directly from the DUV and environment

Online Generation

- Why
  - The generator can use information about the state of the environment and DUV for improving the quality of generation
    - Makes reaching corner cases easier
  - Generally small memory footprint

- Why not
  - Must generate items in order
  - Limited complexity
  - Slows down simulation

Mixing online and offline Generation

- Online and offline generation can be mixed within a verification environment
- Which designs would benefit from this combination?
- For example, in processor verification:
  - instruction streams are generated from high-level programs via compilers, i.e. offline,
  - but the interrupts are generated online, when the processor is in an interesting state.

Dynamic vs. Static Generation

- In static generation the generator is not aware of the state of the DUV and the environment
  - Generation decisions are based entirely on the internal state of the generator
  - The generator is aware of what and when it is allowed to generate

- In dynamic generation the generator is fully aware of the state of the DUV and the environment and generates based on this information
  - The generator can react to interesting states in the DUV

Dynamic Vs. Static Generation

- Dynamic generation is based on reaction while static generation is based on planning
- In general, reaction is harder than planning
  - Time is a factor
  - Unexpected events can get in the way
- Most generators use dynamic features lightly
  - Observe and react to shallow or stable states and resources
    - For example, architectural registers

Offline Dynamic Generation

- *Dynamic and static* generation should not be confused with *online and offline* generation
- An offline generator can use dynamic generation by using a reference model that provides information about the state of the DUV
  - The level and accuracy of the information depends on the abstraction level and accuracy of the reference model

Test Length

- Two extreme approaches for selecting the test length
- Use short tests
  - The shortest tests that can fulfill the requirement in the verification plan
  - For the instruction pairs requirement use tests with just two instructions
- Use long tests
  - Combine many requirements in a single test
  - Wrap a test with initial and ending sequences

Why Short Tests?

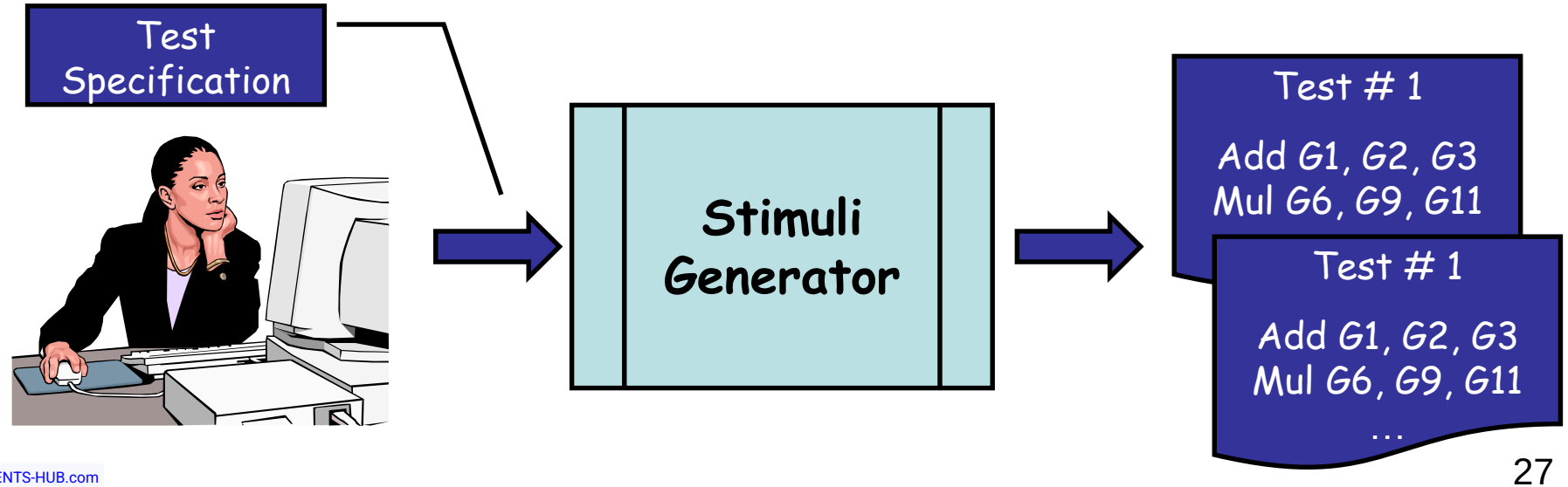
- Easy to create
- Easy to debug
- Easy to maintain
- Short time to simulate each

Why Long Tests?

- Need fewer tests
- Less time to simulate
  - Overall less time as we do not need to repeat the initialization sequence for every test ;)
- Test is not at or near the initial state most of the time
- Use less traveled paths and greater variety of exploration
- Reach target in more ways
  - Often leads to reaching the target in unexpected ways

Randomness - Motivation

- The first time we press the button a test is created
- What happens when we press the button a second time?
  - **The same test appears**
    ➔ our stimuli generator is deterministic



Random Stimuli Generation

Test Specification

Repeat 100 times

Stimuli Generator

Test # 1

Add G1, G2, G3
Mul G6, G9, G11

Test # 2

Fdiv F1, F2, F3
Lw G6, (10)G9

Test # 3

Fmul F1, F2, F3
Xor G6, G1, G2
...

The generated tests are different !

Pure Random Generation

- The opposite end of the spectrum to deterministic generation
- The generator generates random sequences of ‘0’s and ‘1’s that are packed into instructions
- Theoretically, this might seem like the ideal solution
- BUT practically,

  not very useful for verification
  - Most generated test cases are invalid
  - Most valid test cases are not interesting

Side Note – **Pseudo** Random

- When using random number generators, “random” decisions are controlled by a seed
  - Given the value of the seed, random decisions are deterministic
- Pseudo random is essential in verification because of the need to reproduce specific tests
  - For example, to reproduce bugs
- Essential requirement for **Pseudo Random Test Generator:**
  - Need (at least) **repeatability!**
    - Achieved by using the same seed to seed the generator.

Constrained Random Generation

- The stimuli generator is constrained to generate
  - **Valid** tests
  - Tests that meet the user requests
- There are many (infinite number of) tests that fulfill these constraints
- The generator can choose any such test

Example – Instruction Pair Generation

- The test specification is a test with an add instruction followed by an xor instruction
  - Comes from the first extract of our verification plan
- The test should look like
- Everything else can be randomized

```
add_xor_test

Start:
  …
  Add ??, ??, ??
  Xor ??, ??, ??
  …
```

Random Decisions for add_xor_test

- Registers of add instruction
- Data of add instruction
- Registers of xor instruction
- Data of xor instruction

... but also

- Prelude sequence
- Epilogue sequence
- Start address of the program
- Processor operation mode
- Behavior of caches, I/O, …
- …

How To Make Random Decisions

- Pure random decisions
  - Most tests will be invalid
- Constrained random decisions
  - Limit random decisions to those that lead to valid tests
  - Choose uniformly among valid possibilities
  - Result
    - Generated tests are valid
    - Most random decisions are not interesting
      ➔ Small gain in test quality
- "Smart" constrained random decisions
  - Bias decision toward interesting cases
  - Can lead to significant improvement in test quality