Simulation-Based Verification Technologies

Building a Bridge: from Pre-Silicon Verification to Post-Silicon Validation

FMCAD, 2008

Moshe Levinger

26/11/2008

Talk Outline

- **Simulation-Based Functional Verification**
- **Pre-Silicon Technologies**
  - Random Test Program Generators
  - Model-Based, CSP-Driven Generation
  - Coverage-Directed-Generation by Construction
  - Coverage-Directed-Generation by Feedback
- **Post-Silicon Validation**
  - Needs, Challenges, Trade-offs
  - Post-Silicon Exercisers
  - Directable Model-Based Exerciser
- **Cross-Platform Functional Verification Methodology**

Talk Sources

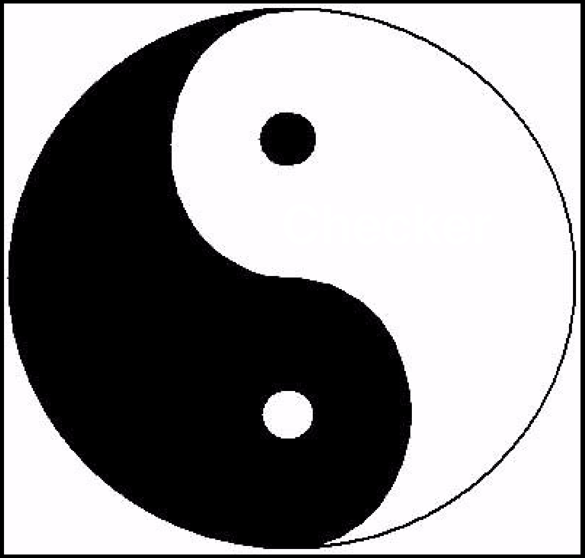
- “Directable Functional Exercisers”, Gil Shurek, GSRC Post Silicon workshop, DAC 2008
- “Challenges in Post Silicon Verification of IBM’s Cell/B.E. and other Game Processors”, Shakti Kapoor, HLDVT 2007
- “Random Test Generators for Microprocessor Design Validation”, Joel Storm, EMICRO 2006
- “Simulation-Based Functional Verification” course, Avi Ziv, Technion 2007
- “Simulation-Based Verification Technologies at IBM”, Moshe Levinger, Intel Symposium 2007

MACHINE PAYS UP TO $1,000
ALL OTHER WINS PAID BY ATTENDANT
PLAY 1 COIN
BAR
BAR
BAR
49
WINNER PAID
CREDITS

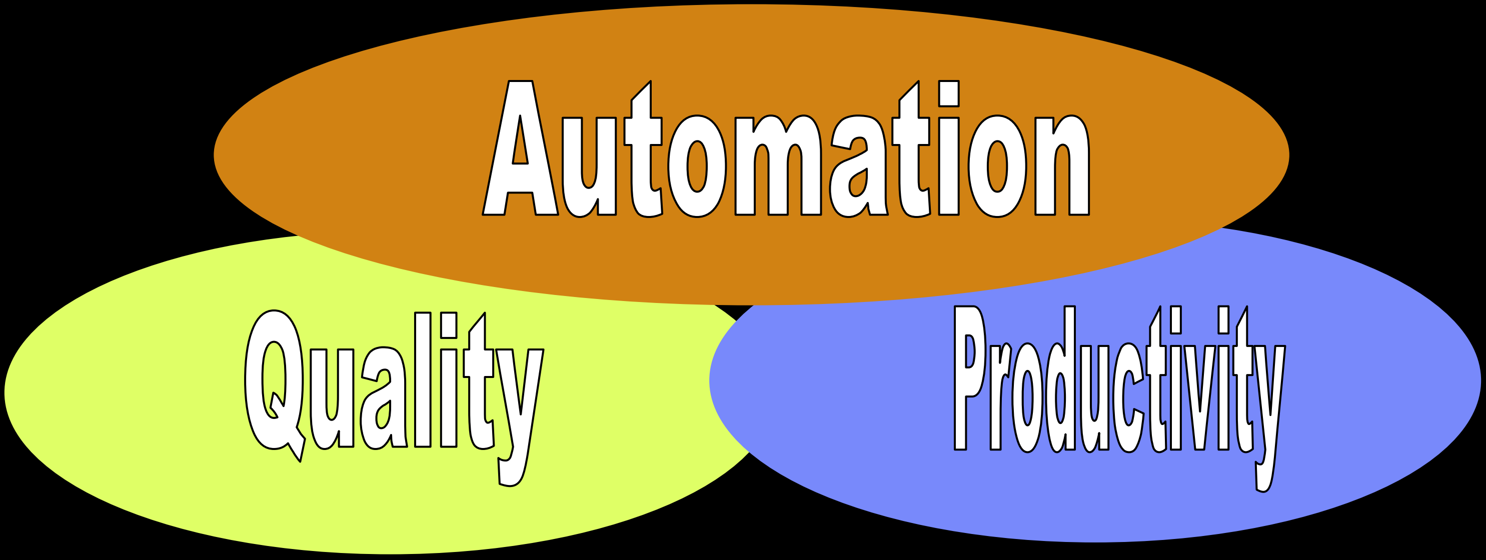
WIN

The Yin-Yang of Verification

- **Driving and checking are the yin and yang of verification**
  - We cannot find bugs without creating the failing conditions
  - We cannot find bugs without detecting the incorrect behavior



Key Ingredients for Successful Verification



Better Products

Shorter TTM

Lower Costs

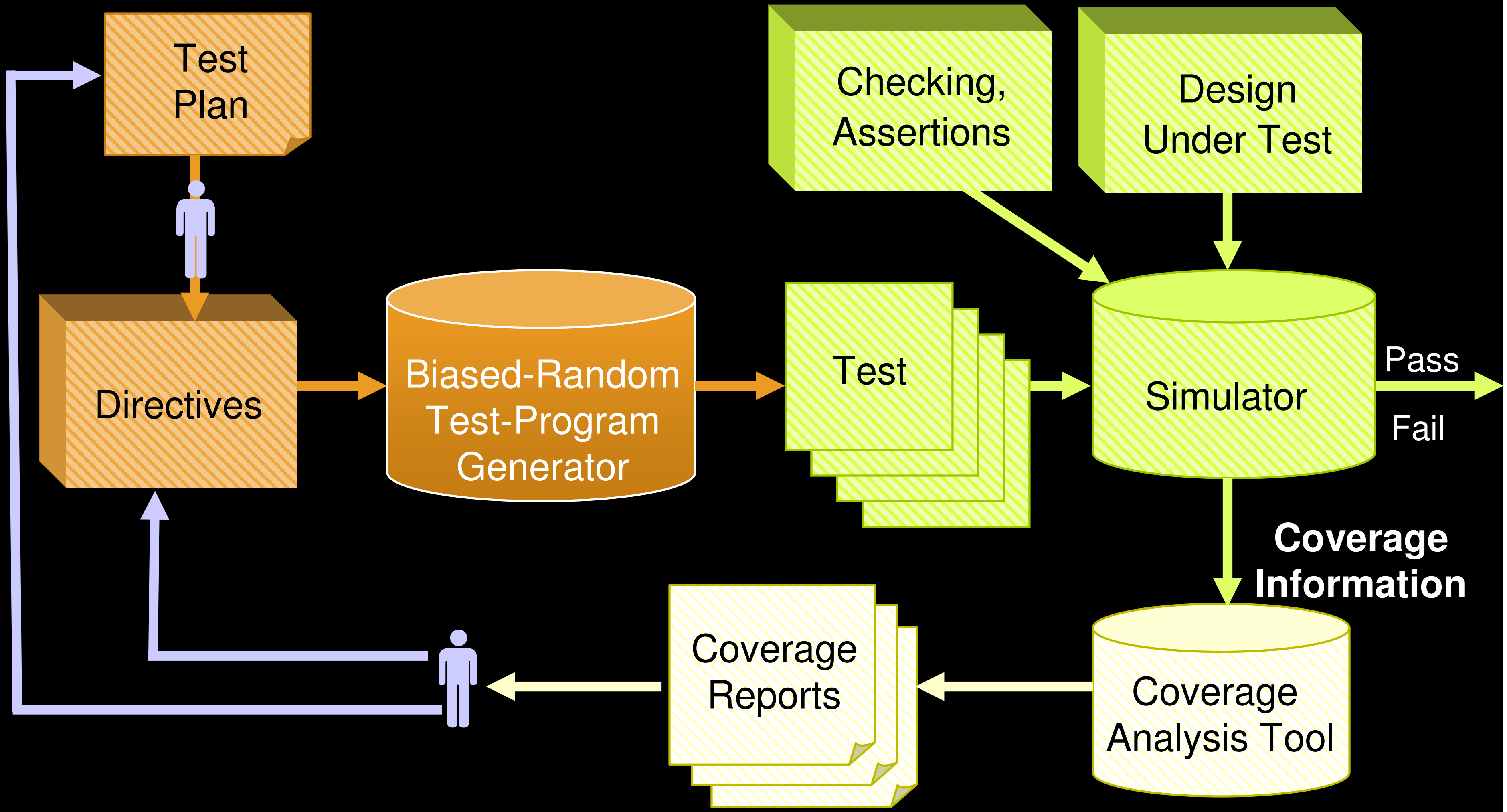
Technology Score Card – Key Quality Dimensions

Quality

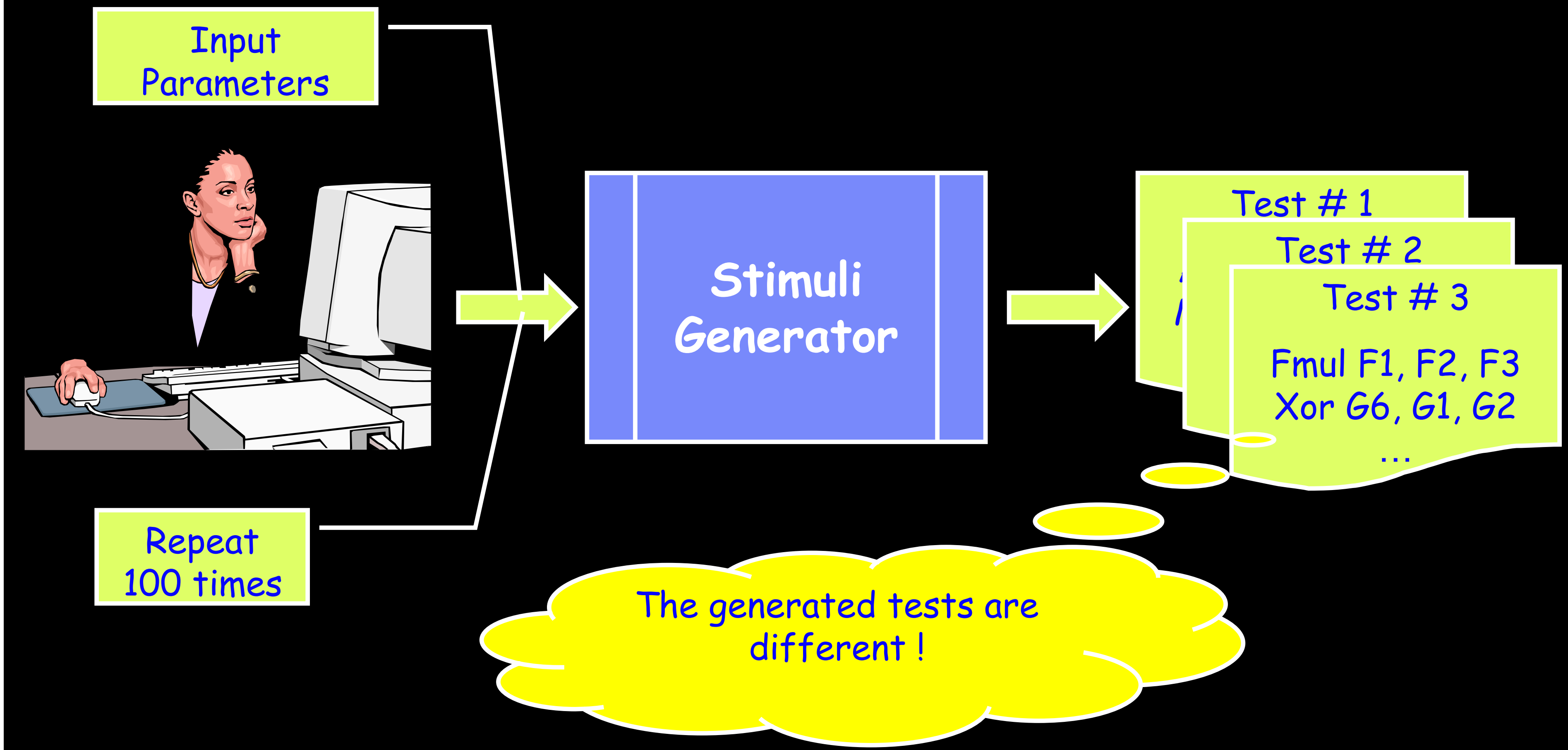
Sampling Size
Quality of each sample **- Scenario description power** **- Testing know-how** **- Domain expertise** **- Bug awareness**
Learning / improvement over time

Pre-Silicon Verification Technologies

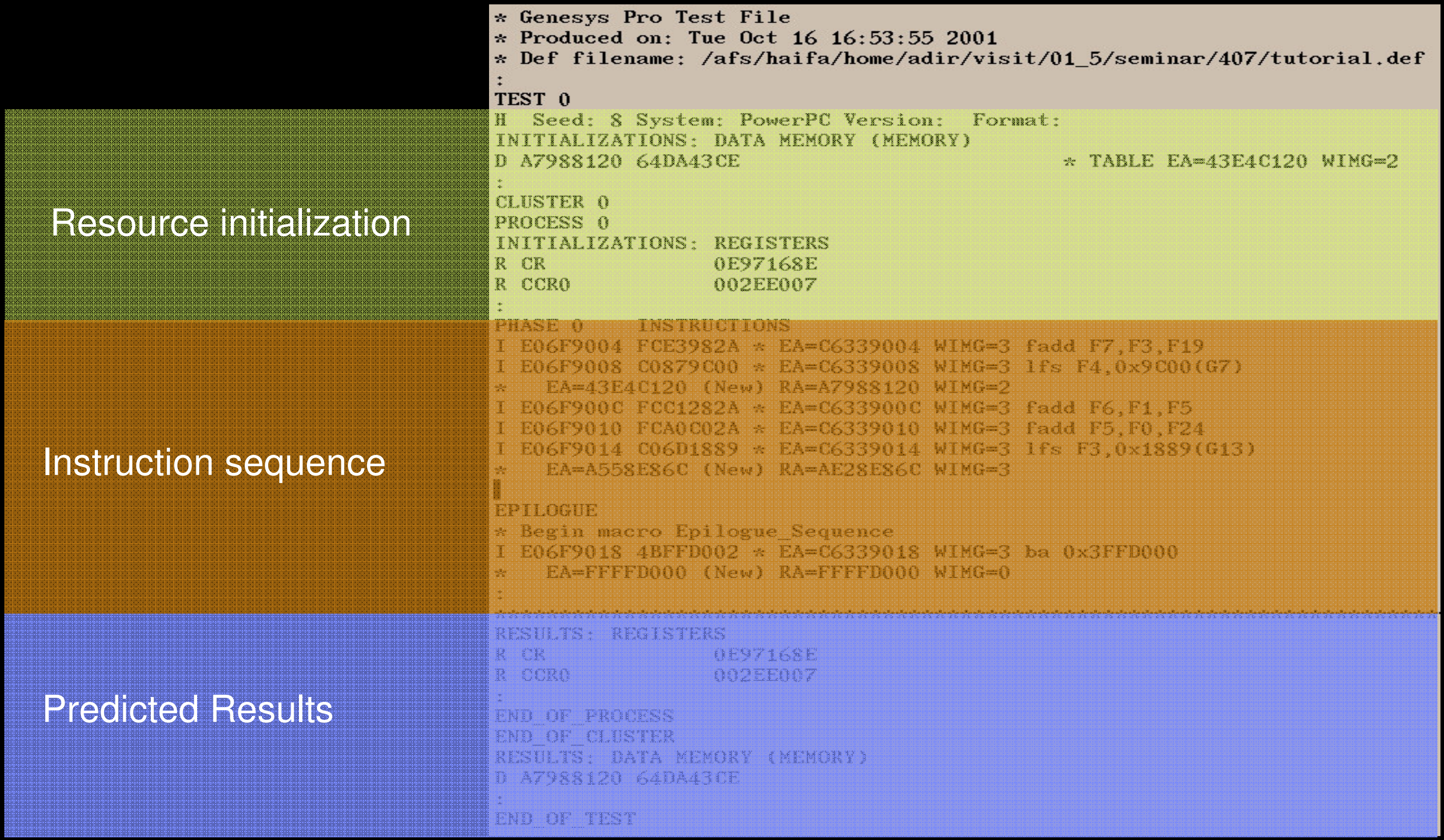
Typical Flow of Simulation-Based Verification



Rudimentary Random Stimuli Generation (RTPG)



Example of Test Generator Output



```
* Genesys Pro Test File
* Produced on: Tue Oct 16 16:53:55 2001
* Def filename: /afs/haifa/home/adir/visit/01_5/seminar/407/tutorial.def
:
TEST 0
H  Seed: 8 System: PowerPC Version:  Format:
INITIALIZATIONS: DATA MEMORY (MEMORY)
D A7988120 64DA43CE                                 * TABLE EA=43E4C120 WIMG=2
:
CLUSTER 0
PROCESS 0
INITIALIZATIONS: REGISTERS
R CR                0E97168E
R CCR0              002EE007
:
PHASE 0     INSTRUCTIONS
I E06F9004 FCE3982A * EA=C6339004 WIMG=3 fadd F7,F3,F19
I E06F9008 C0879C00 * EA=C6339008 WIMG=3 lfs F4,0x9C00(G7)
*    EA=43E4C120 (New) RA=A7988120 WIMG=2
I E06F900C FCC1282A * EA=C633900C WIMG=3 fadd F6,F1,F5
I E06F9010 FCA0C02A * EA=C6339010 WIMG=3 fadd F5,F0,F24
I E06F9014 C06D1889 * EA=C6339014 WIMG=3 lfs F3,0x1889(G13)
*    EA=A558E86C (New) RA=AE28E86C WIMG=3

EPILOGUE
* Begin macro Epilogue_Sequence
I E06F9018 4BFFD002 * EA=C6339018 WIMG=3 ba 0x3FFD000
*    EA=FFFFD000 (New) RA=FFFFD000 WIMG=0
:
*************************************************************************
RESULTS: REGISTERS
R CR                0E97168E
R CCR0              002EE007
:
END_OF_PROCESS
END_OF_CLUSTER
RESULTS: DATA MEMORY (MEMORY)
D A7988120 64DA43CE
:
END_OF_TEST
```

RTPG: Technology Score Card

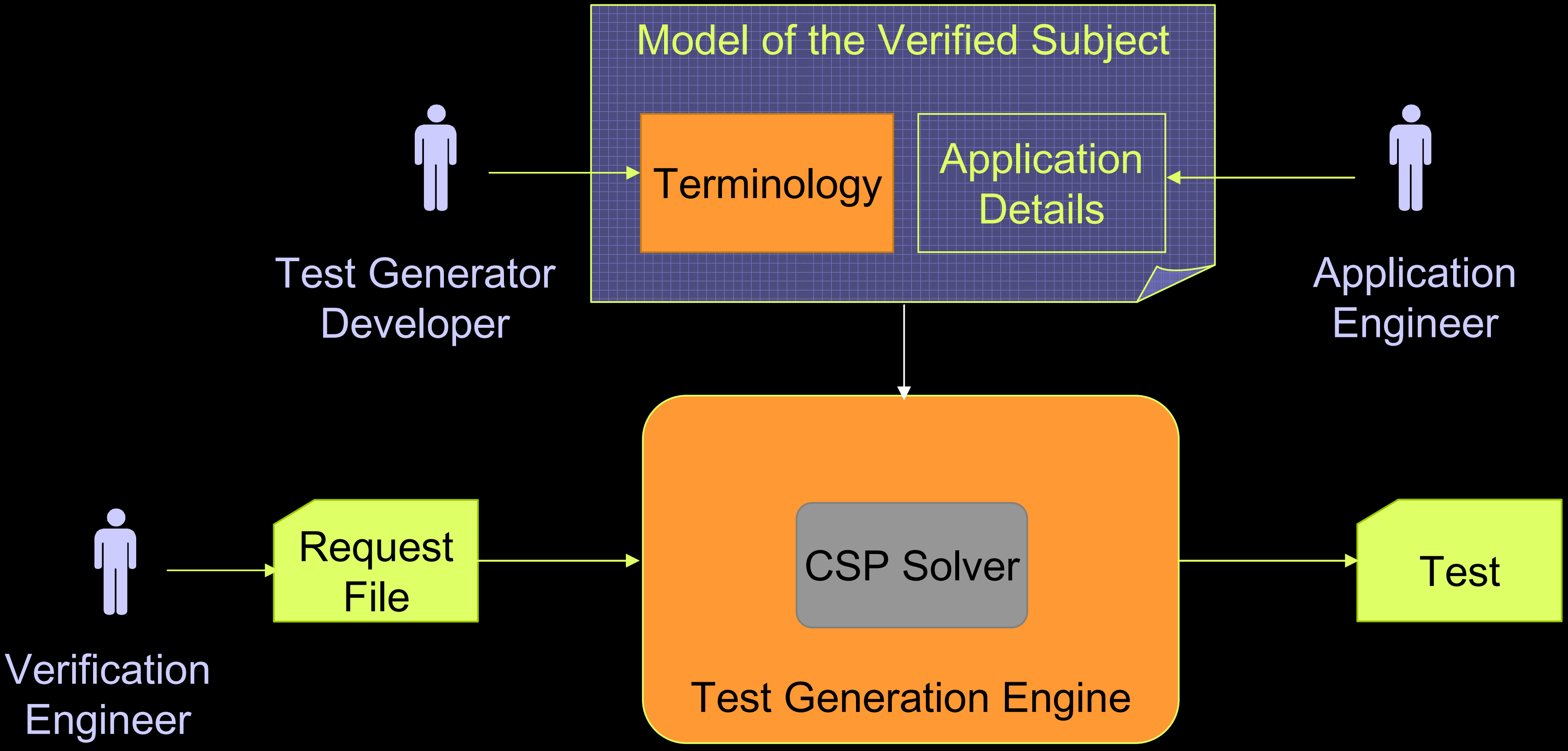
Sampling Size
Quality of each sample
Learning / Improvement

Low: simulation & generation speed
Medium: Hard-coded Testing Knowledge Basic input parameters
None

Model-Based Constraint-Driven Test Program Generator

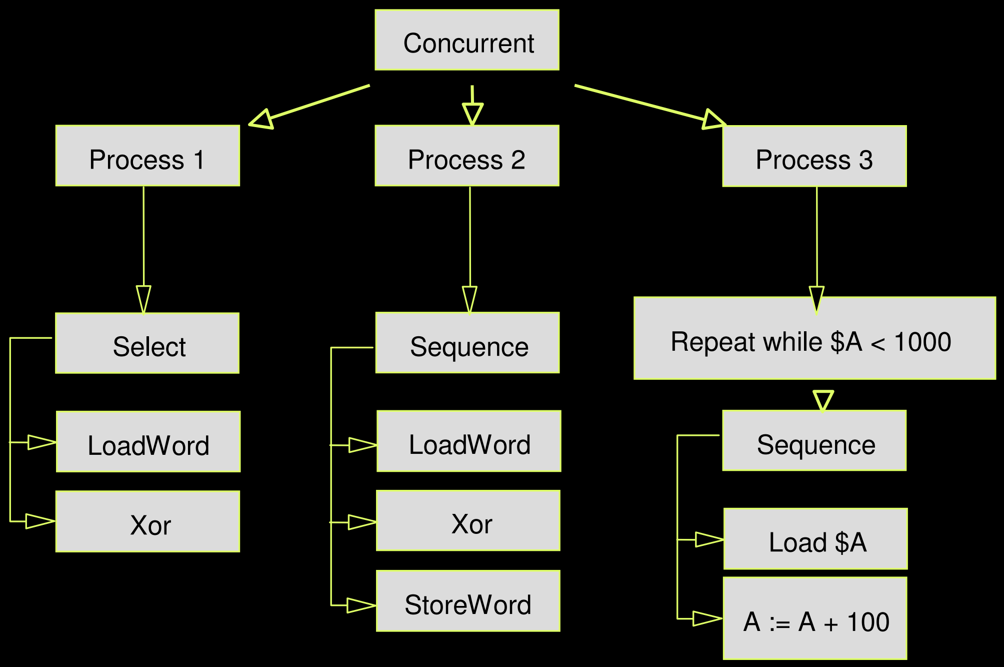
- **Model-based test-case generator which is applicable for a variety of architectures and designs**
- **Generic architecture-independent generation engine**
- **External formal and declarative description of the Design-Under-Verification (DUV)**
- **Powerful test description language**
  - Ranging from completely random to fully specified test cases
- **Open architecture for incorporation of Testing Knowledge**

Model-Based Test Generation – High Level Concept



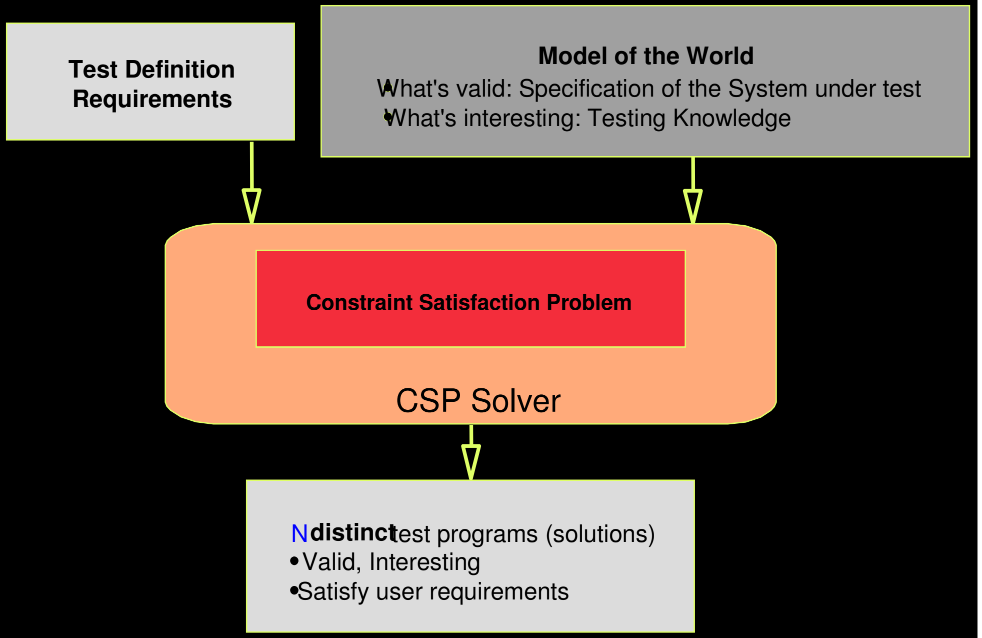
Powerful Test Template Definition

- **Allows the user to define delicate verification scenarios using the full power of programming-like language:**
  - Sequence, permute, select, repeat, if-then-else
  - Variables, assignments, expressions, conditions



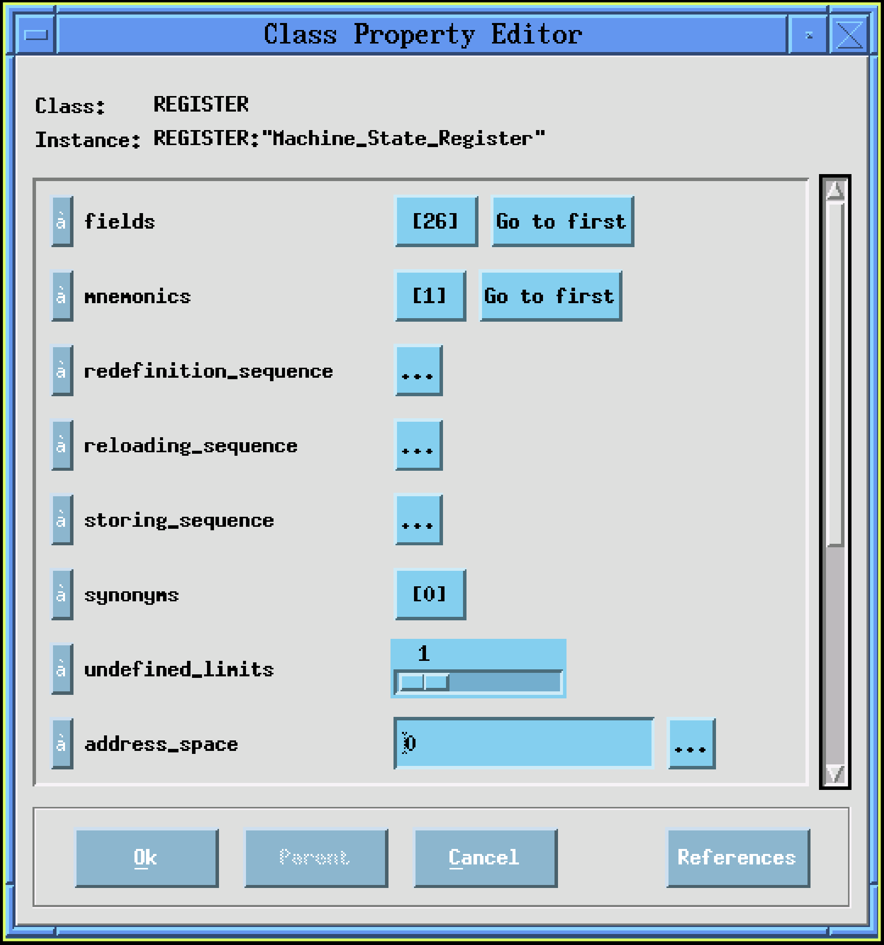
Constraint Programming Based Technology

- Allows the user to request any possible set of constraints defining a verification scenario
- Provides uncompromising and powerful solutions for complex sets of constraints
- Copes with unique CSP characteristics:
  - Random, uniform distribution solution - as opposed to one, all, or "best" solution
  - Huge variable domains, e.g., address spaces


Test Definition Requirements
Model of the World
What's valid: Specification of the System under test
What's interesting: Testing Knowledge
Constraint Satisfaction Problem
CSP Solver
N distinct test programs (solutions)
• Valid, Interesting
• Satisfy user requirements

Declarative, Form-based Modeling Environment

- **A modeling environment for**
  - Instructions, Resources
  - Components, Interactions
  - Testing knowledge
- **Different ‘forms’ are provided to describe various aspects of the design under verification**
- **No need to write code**
  - But, if needed - special cases can be modeled through C++ hooks

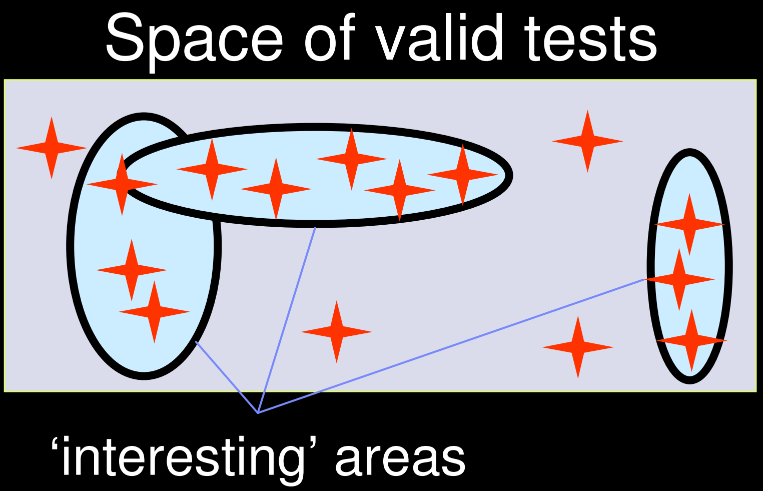


Class: REGISTER
Instance: REGISTER:"Machine_State_Register"

fields	[26]	Go to first
mnemonics	[1]	Go to first
redefinition_sequence	...	
reloading_sequence	...	
storing_sequence	...	
synonyms	[0]	
undefined_limits	1	
address_space	0	...

Generic Testing Knowledge

- **A set of mechanisms that aim at improving test-case quality**
- **Capitalize on recurring concepts:**
  - Address translation
  - Pipelines
  - Caches
- **The basic mechanism: non-uniform random choice**
  - Bias towards 'interesting' areas
- **Examples:**
  - Resource contention
  - Translation table entry reuse
  - Data placement

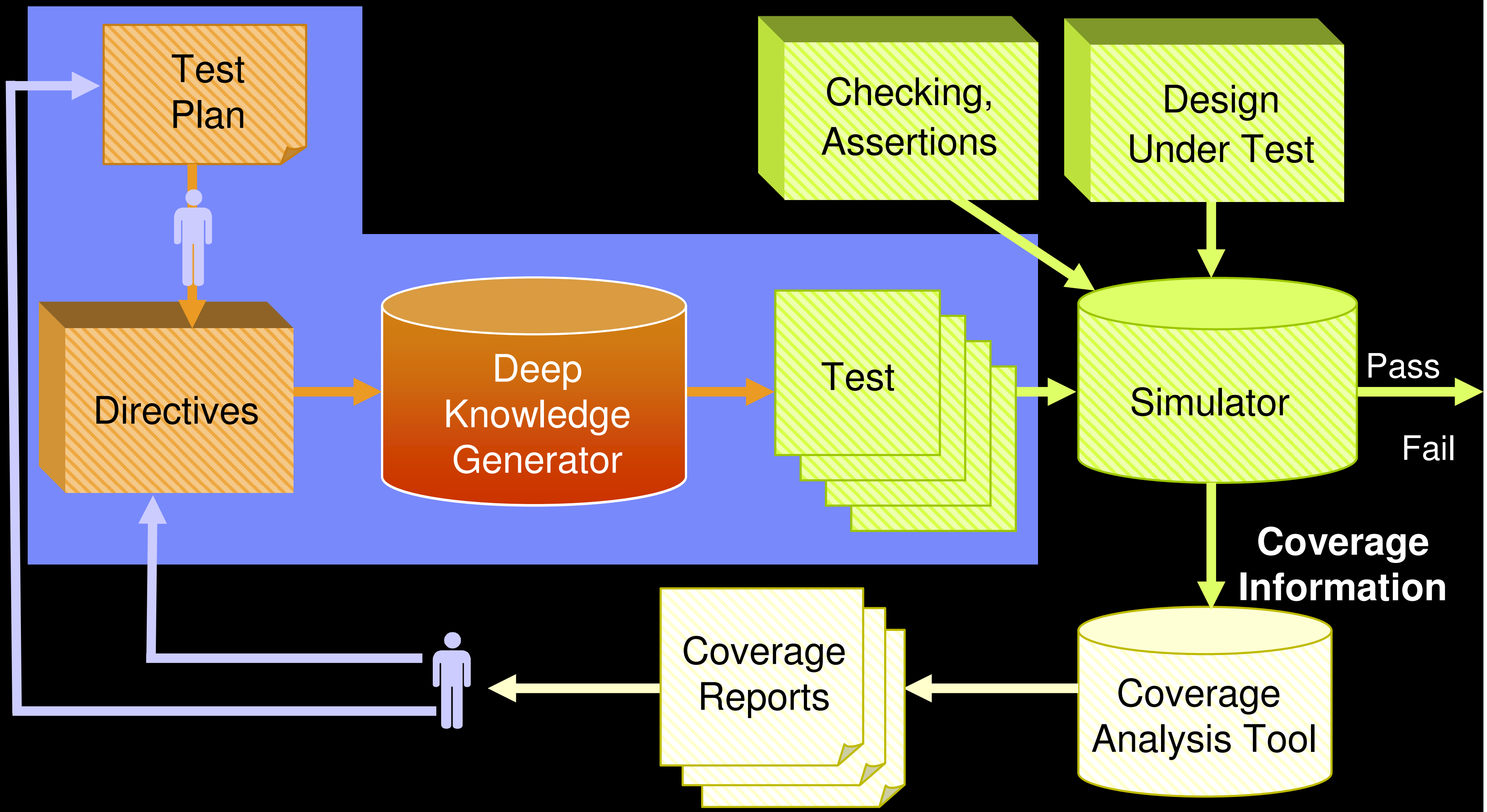


Model-Based, CSP-Driven TG: Technology Score Card

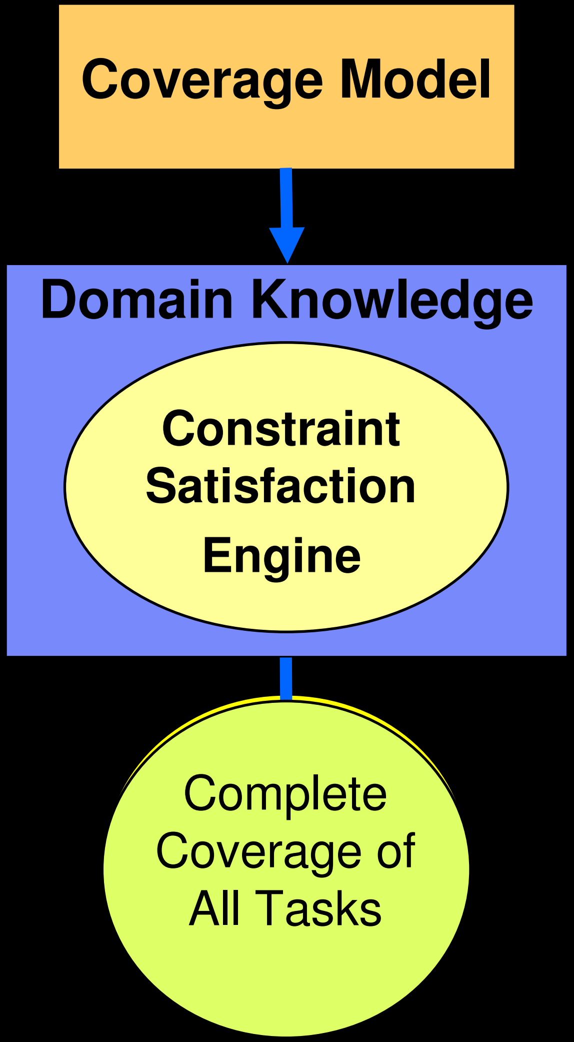
Sampling Size
Quality of each sample
Learning / Improvement

Low: simulation and generation speed
High: Testing Knowledge paradigm Rich scenario language Powerful CSP engines
None

Typical Flow of Simulation-Based Verification



Deep Knowledge, Coverage-based Test Generation


Coverage Model
Domain Knowledge
Constraint Satisfaction Engine
Complete Coverage of All Tasks

An Example: Deep-Knowledge Test Generator for Floating-Point

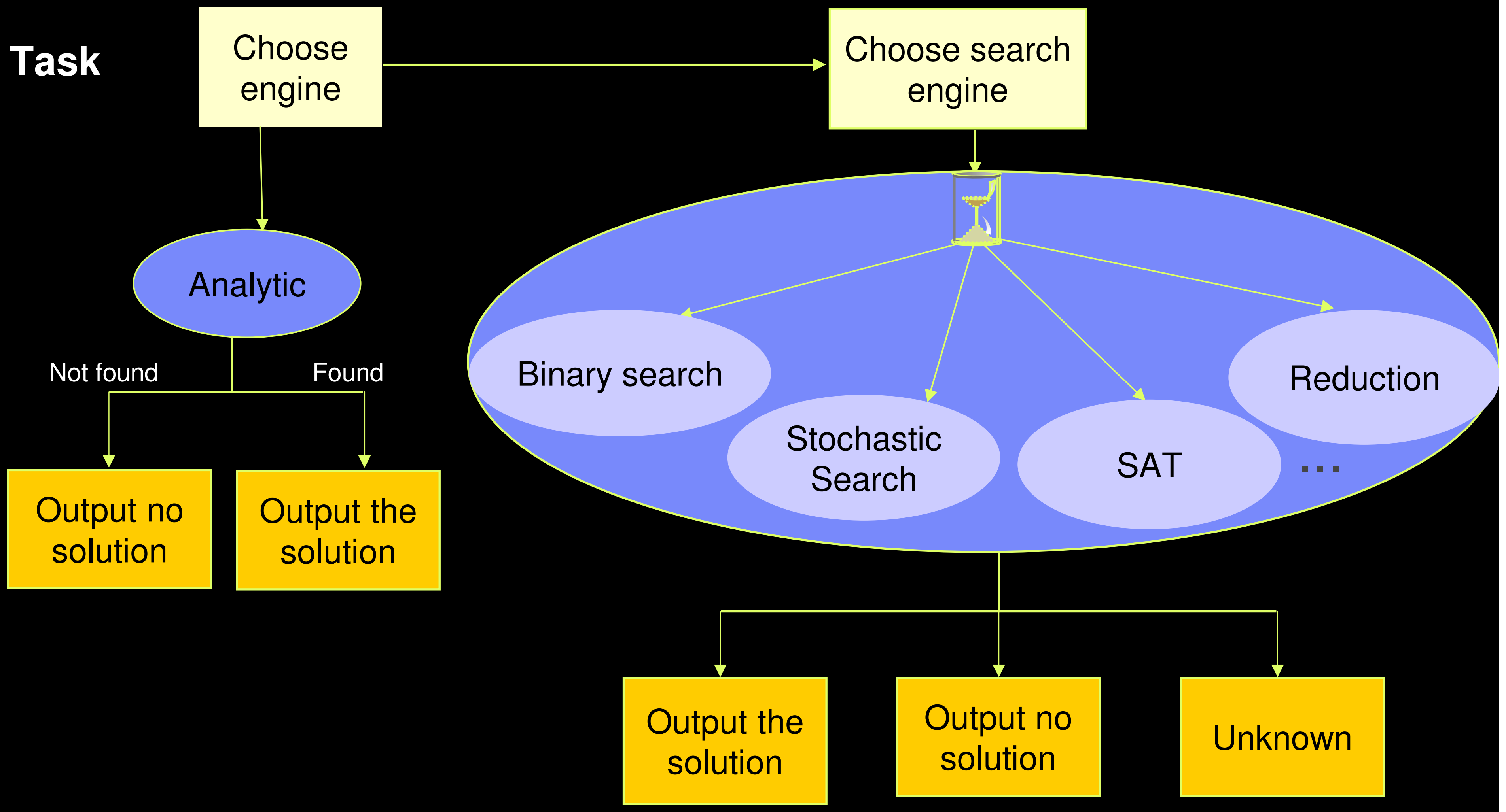
- **FPgen – generic solution for floating-point verification**
- **FPgen aims to fulfill comprehensive FP test plans:**
  - Definition of architecture tasks
  - Definition of micro-architecture tasks
  - Coverage model language
- **FPgen has a deep understanding of the floating-point world**
  - Enables the tool to fulfill complex FP tasks

FPgen Input: a Floating-Point Data-path Coverage Model

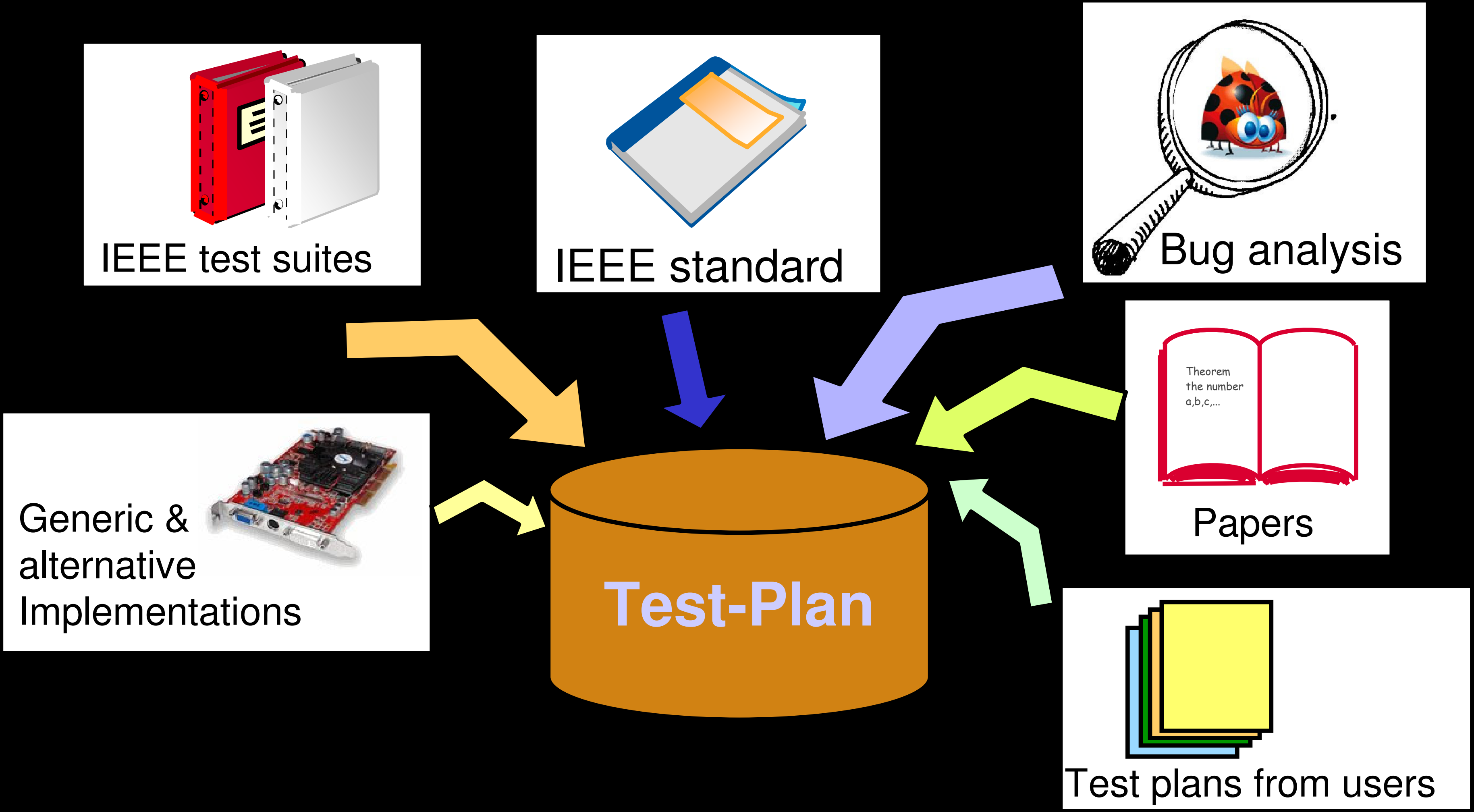
Example: All Types model

Operand1	Multiply	Operand2	=	Result
+/- Infinity		+/- Infinity		+/- Infinity
+/- Zero		+/- Zero		+/- Zero
+/- Norm		+/- Norm		+/- Norm
+/- Denorm		+/- Denorm		+/- Denorm
+/- Large number	X	+/- Large number	X	+/- Large number
+/- Small number		+/- Small number		+/- Small number
+/- Min Denorm		+/- Min Denorm		+/- Min Denorm
+/- Max Denorm		+/- Max Denorm		+/- Max Denorm
+/- Min Norm		+/- Min Norm		+/- Min Norm
+/- Max Norm		+/- Max Norm		+/- Max Norm

FPgen Overall Solving Scheme Flow



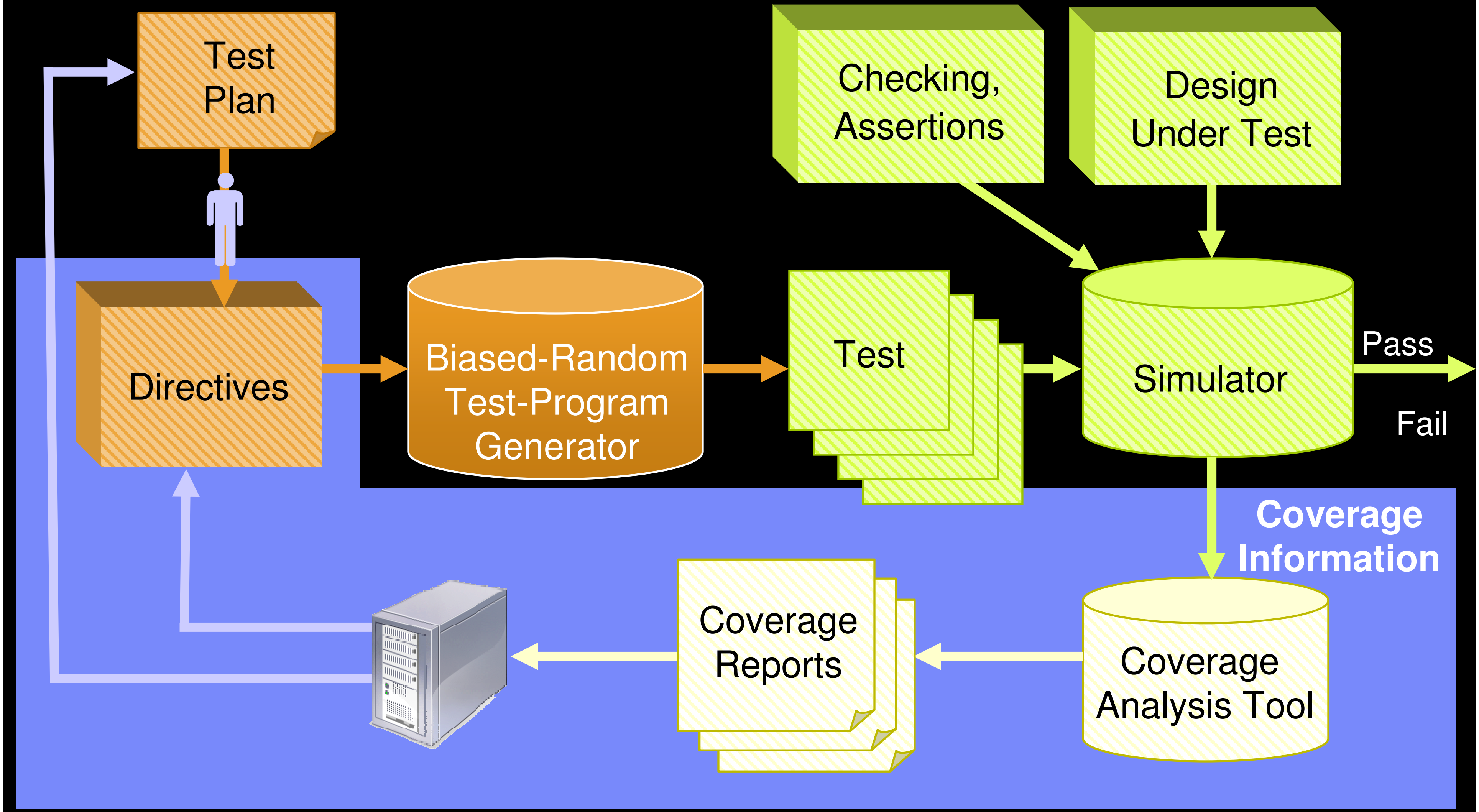
Towards a Fully Automated Solution: The Generic Test Plan



Deep Knowledge Generator: Technology Score Card

Sampling Size	Very Low: slow generation speed
Quality of each sample	Very High: Generic Test Plan concept Deep FP knowledge Powerful FP solvers Bug-driven models in GTP
Learning / Improvement	None

Typical Flow of Simulation-Based Verification



CDG -- Coverage-Directed-Generation: Closing the Loop using Machine Learning Techniques

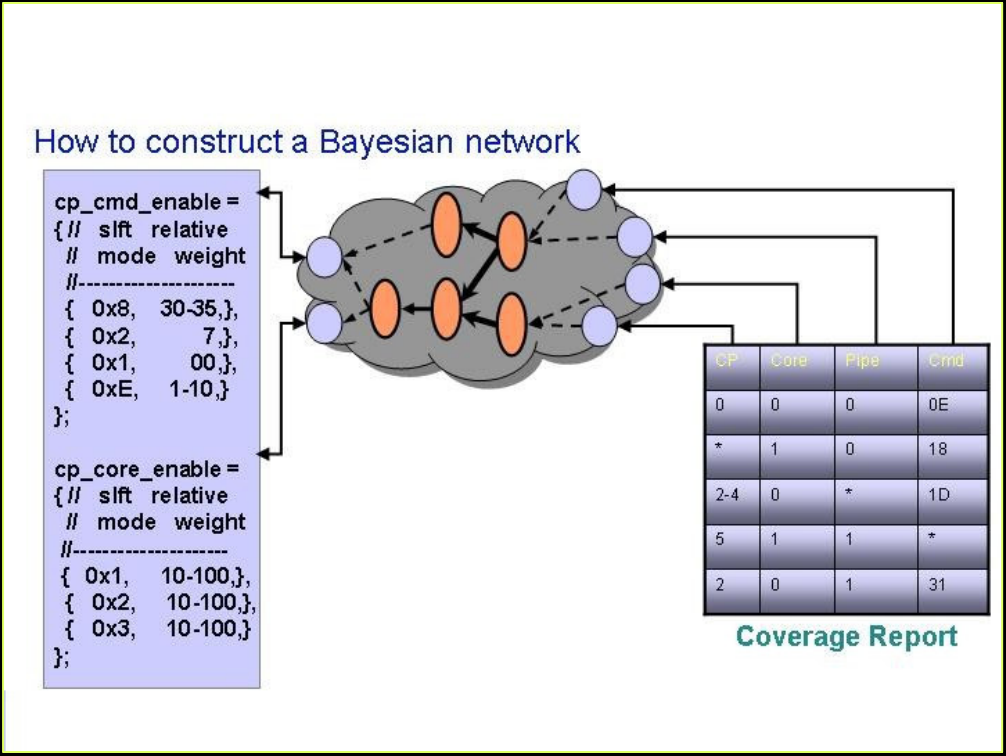
- **Motivation**
  Coverage analysis tools can assess the quality of a set of test cases but cannot recommend how to improve the set

- **Objectives**
  Introduce a feedback loop to tune test generation
  - Stimulate hard-to-reach coverage events
  - Improve rate of coverage
  - Control coverage space distribution

Coverage-Directed-Generation: Closing the Loop using Machine Learning Techniques

Approach

- **Use** Bayesian networks **to represent the CDG ingredients**
- **A natural and** compact representation **of the distribution space**
- **Enables encoding of essential** domain knowledge

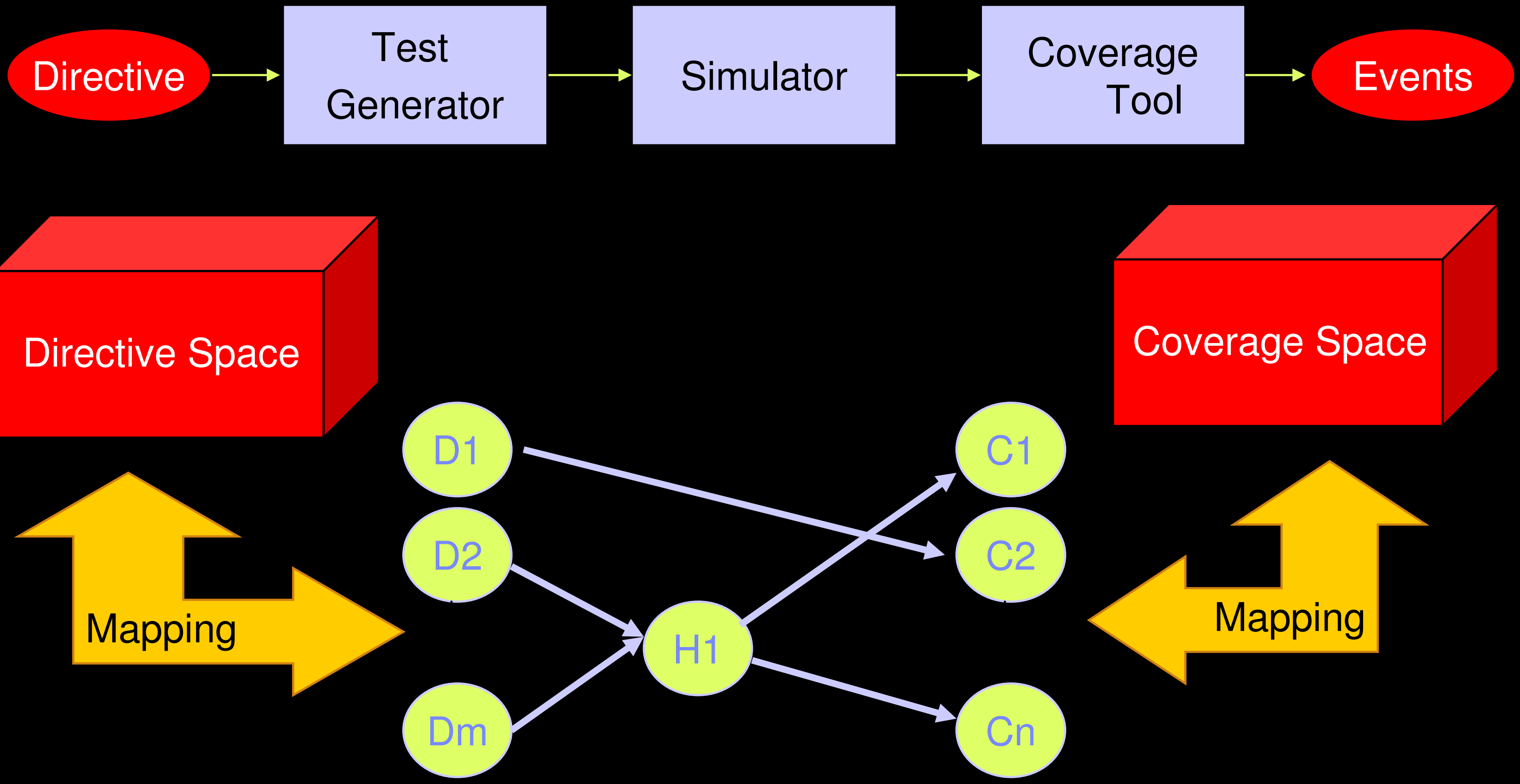


```
cp_cmd_enable =
{// slft  relative
 //  mode  weight
 //-------------------
 {  0x8,   30-35,},
 {  0x2,       7,},
 {  0x1,      00,},
 {  0xE,    1-10,}
};

cp_core_enable =
{// slft  relative
 //  mode  weight
 //-------------------
 {  0x1,   10-100,},
 {  0x2,   10-100,},
 {  0x3,   10-100,}
};
```

CP	Core	Pipe	Cmd
0	0	0	0E
*	1	0	18
2-4	0	*	1D
5	1	1	*
2	0	1	31

Employing Bayesian Networks for CDG

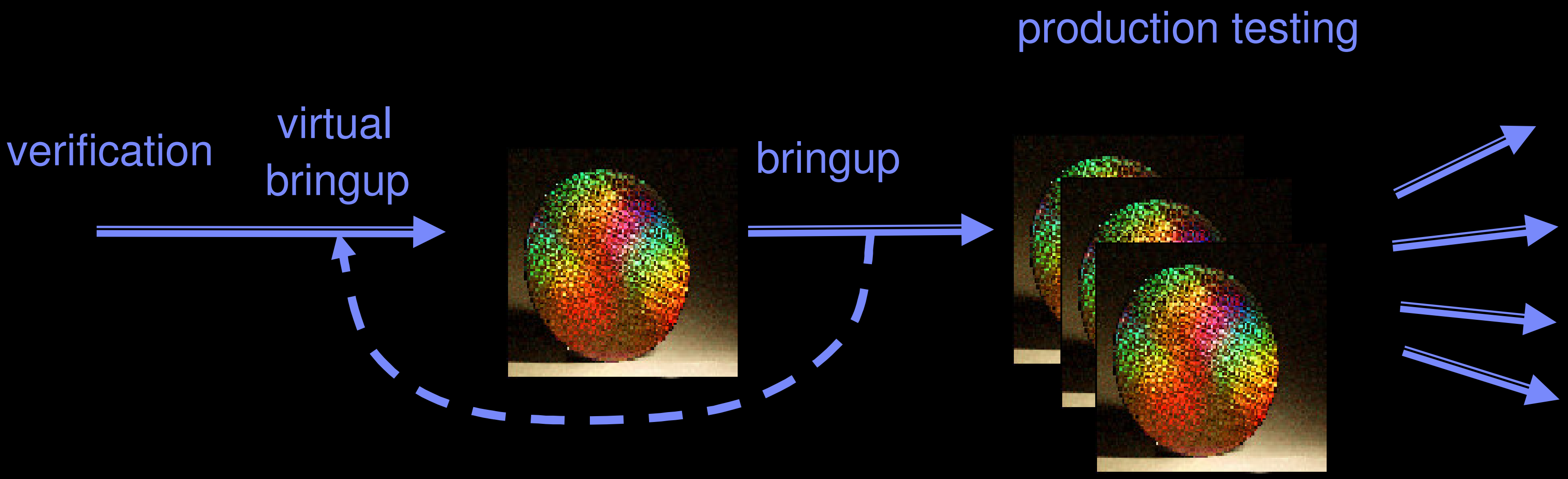


CDG by Feedback: Technology Score Card

Sampling Size	Low: simulation and generation speed
Quality of each sample	Moderate: Depending on the case
Learning / Improvement	High: Machine Learning scheme

Post Silicon Validation Technologies

End-to-End View: From Pre-Silicon Verification up to Production Testing



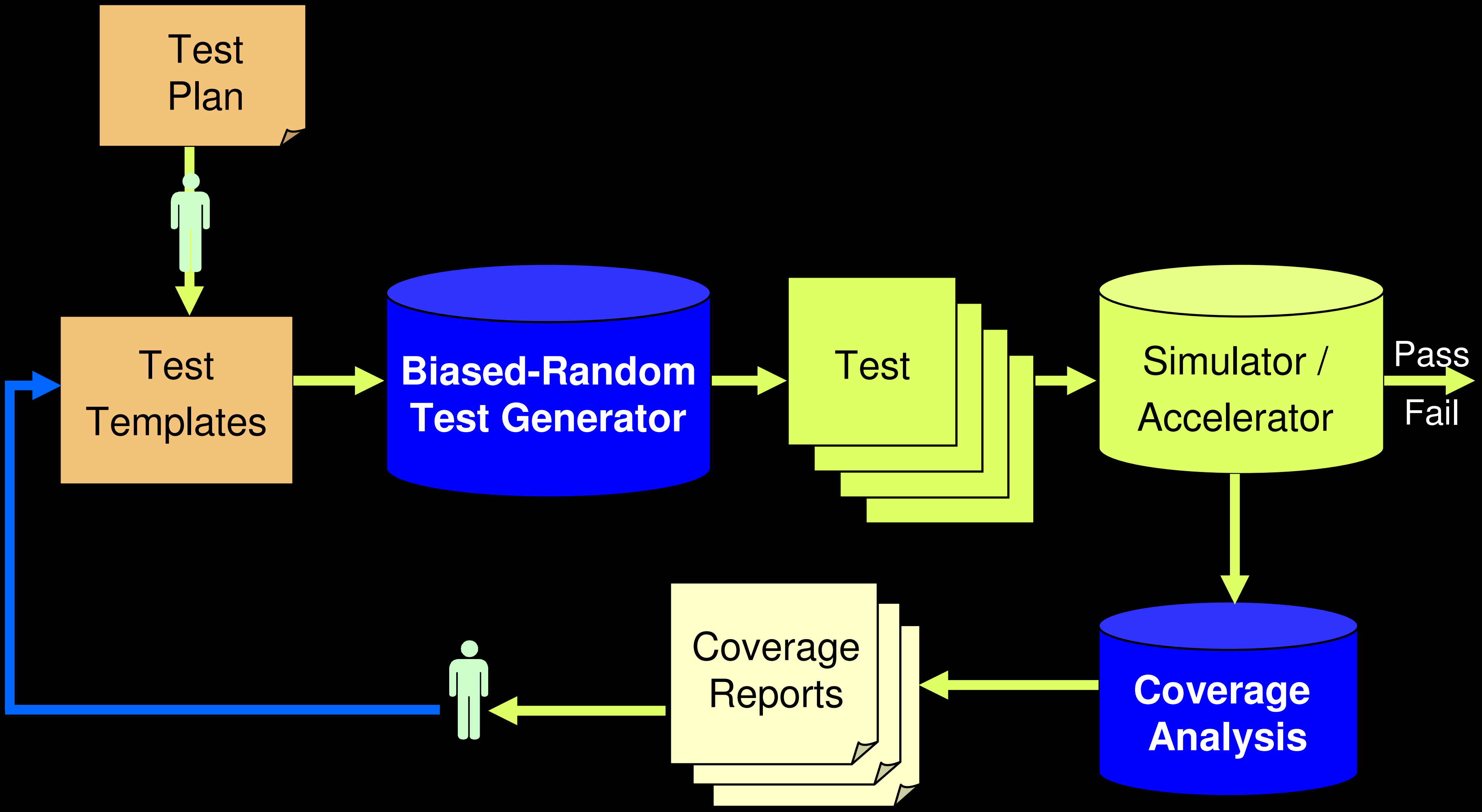
- Environments: verification, virtual bringup, bringup, production testing
- Platforms: simulation, acceleration, silicon, on wafer

Growing Need for Post Silicon Verification

- **Increasing complexity of h/w designs**
  - Multi Core/Multi Threads on Chip
  - Heterogeneous processors
  - Size of the chip

- **Pre-Silicon Limitations**
  - Only a small fraction of the state-space is covered
    - Very slow execution speed
    - Size of the model
  - Does not predict post silicon behavior

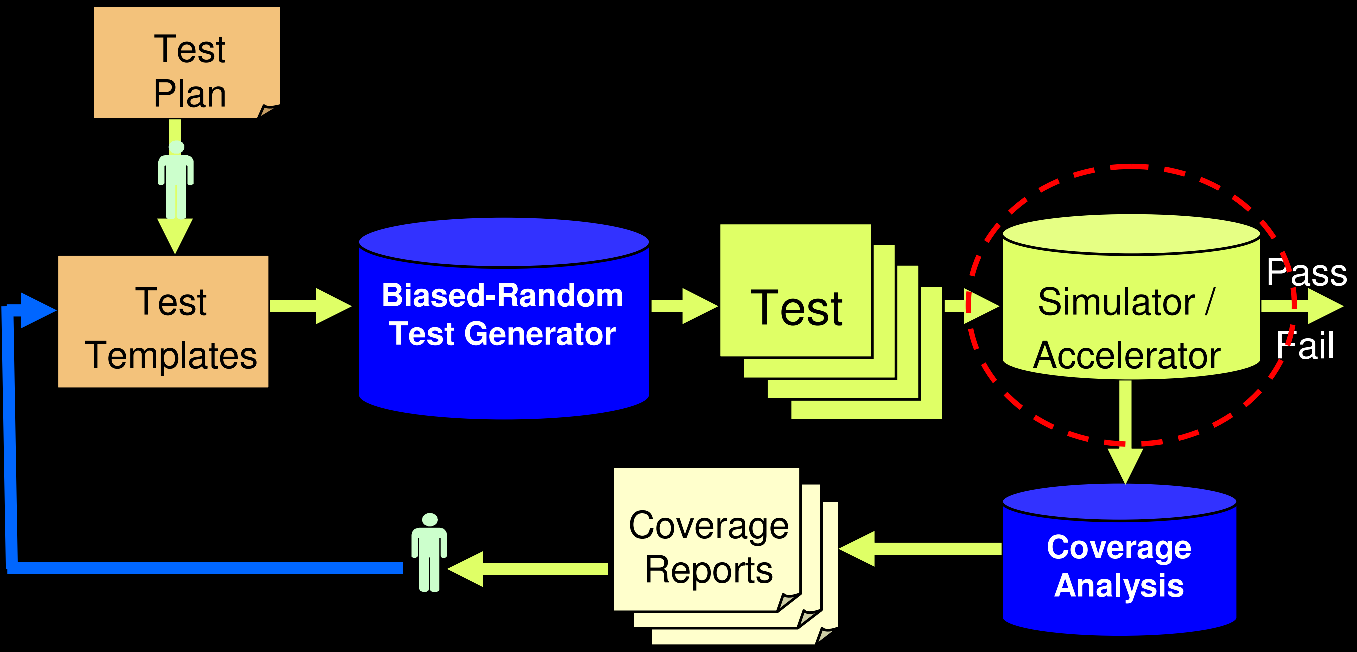
Reminder: Simulation-based Test Generation Methodology



Platform Performance

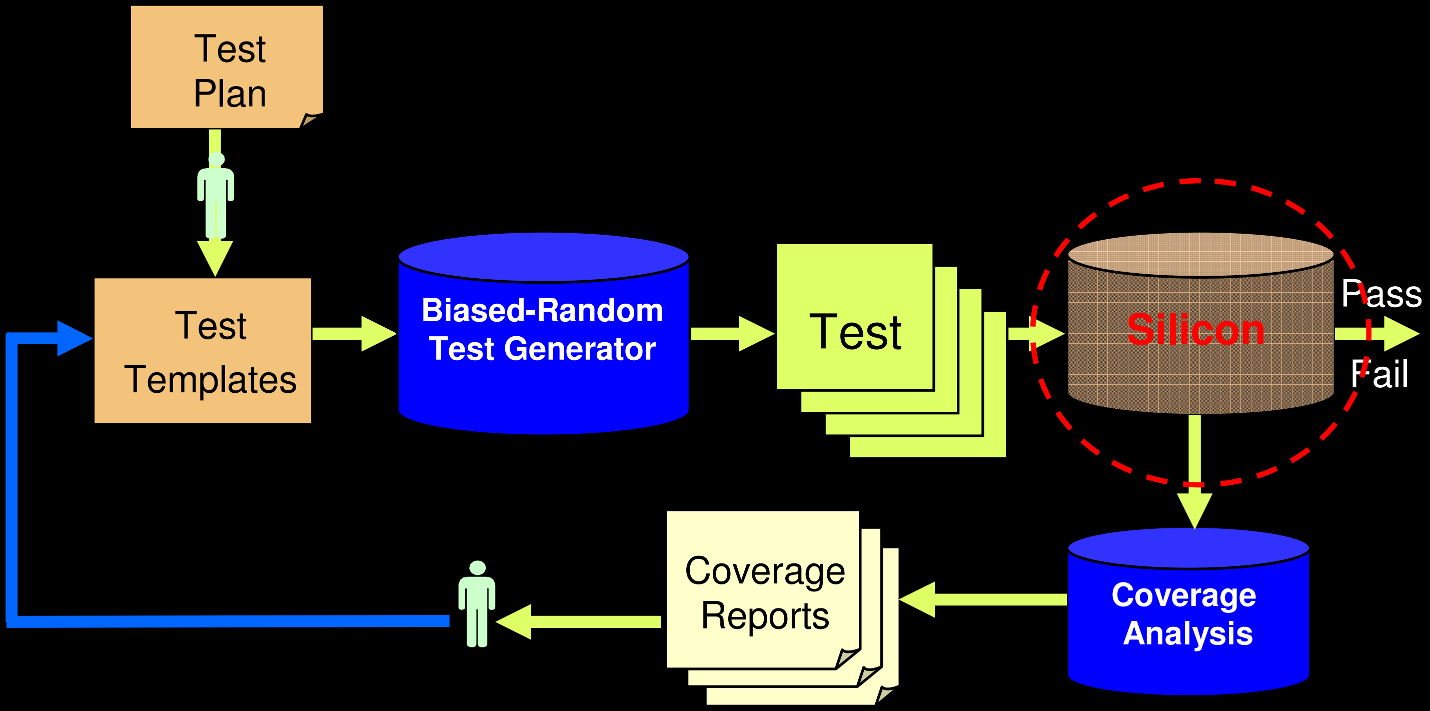
~**500B** simulation cycles available to verify a processor

SW simulator with 10-100 cyc/sec	: ~1500-150 years
HW accelerator with 10k-50k cyc/sec	: 580-115 days
4GHz processor silicon	: 125 sec



Can we just Replace the Simulator with Silicon?

- High test loading/result off-loading overhead ⇒ low silicon utilization
- Alternatively, need a program to run and check the results ⇒ **Hardware Exerciser**



Controllability and Observability

- **Controllability - Indicates the ease at which the verification engineer creates the specific scenarios that are of interest**

- **Observability - Indicates the ease at which the verification engineer can identify when the design acts appropriately versus when it demonstrates incorrect behavior**

Platform Tradeoffs

	Software Simulator	Hardware Accelerator/Emulator	Early Hardware
Performance	slow	faster	ideal speed
Control & Visibility	total model control/visibility	control/visibility with some penalty	very limited
Price per platform unit	relatively inexpensive	expensive	very expensive as a functional verification platform

Hardware Exercisers: Technology Challenges

- Limited observability
  - Challenging checking, debugging and coverage measurement
- On-line generation
  - Limiting generation scheme complexity
- OS dependency prevents early deployment, restricts machine access
  - Prefer a bare-metal setup or a test-oriented OS
- Acceleration
  - Throughput is an issue, also - how to exploit the better observability?
- Wafer testing
  - Size of exerciser image (need to fit into L2), throughput

Checking Schemes for Hardware Exercisers

- **Based on machine behavior**
  - Machine hang
  - "Bad machine" mechanisms
- **Self checking: the test does what it's supposed to do**
  - Test invariants: Collier scenarios, sorting algorithms, etc.
- **Built-in reference model**
- **Compare consistency of results & behavior over multiple runs**

Checking Schemes for Hardware Exercisers (Cont.)

Two Pass Comparison

- **First Pass**
  - **Run test in single step mode**
  - **Save state data at periodic checkpoints**
- **Second Pass**
  - **Reset all data**
  - **Run test normally (no single step)**
  - **Check state data at periodic checkpoints**
    - **Registers**
    - **Exceptions taken**
    - **…**

Basic Hardware Exercisers: Example #1

- Smart generator, simulator and result checker on the processor
  - Delicate memory scenarios, fixed point and floating point, etc.
  - Limited control and testing-knowledge compared to the simulation-platform generators
- Architectural result-prediction and generation-scheme using an internal reference model
- Challenging maintenance: complexity and architecture, internal reference model
- Utilizing a special tiny OS

Basic Hardware Exercisers: Example #2

- Family of lightweight, bare-metal exercisers
- Random, brute-force generation of code-streams
- Each exerciser is focusing on selected aspects of the hardware functionality employing hard-coded internal testing knowledge
- Checking: mainly two-pass approach
- Limited user control specifying a mix of instruction-patterns, macros, and functions

Basic Hardware Exerciser: Technology Score Card

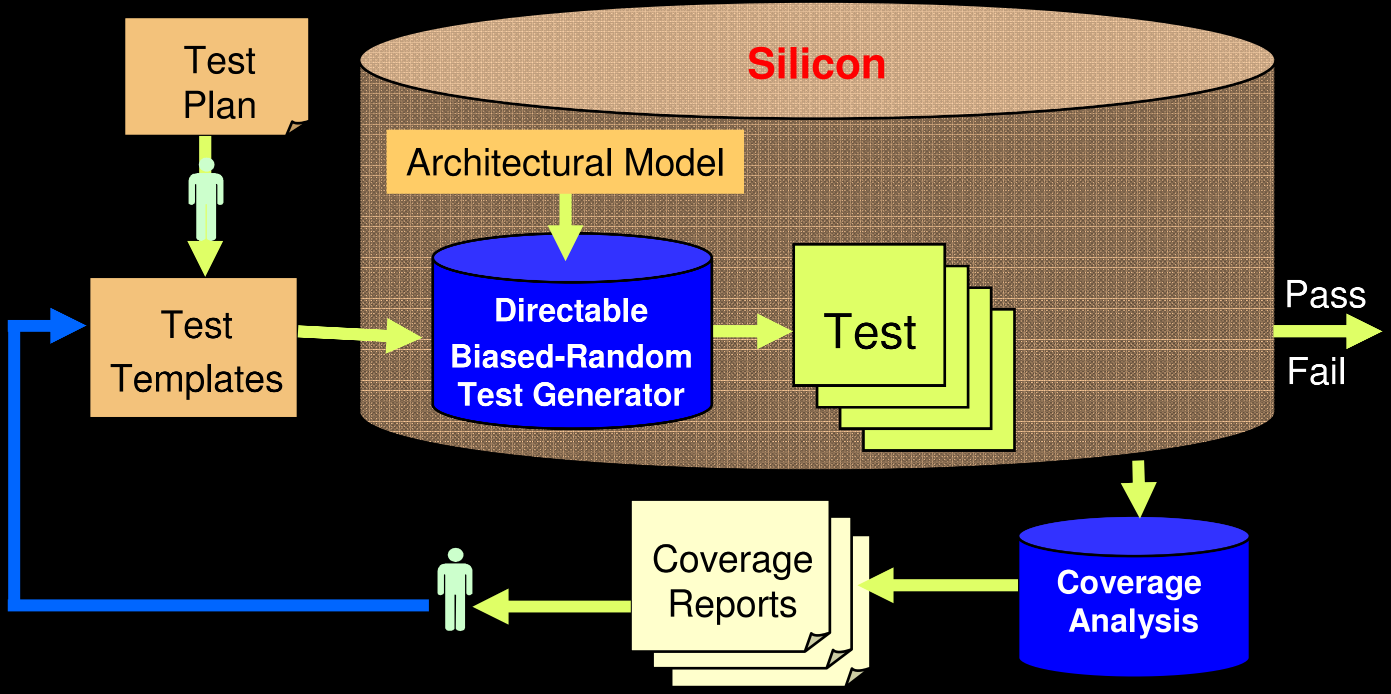
Sampling Size	High: hardware speed, fast generation
Quality of each sample	Low-Medium: Dedicated Testing Knowledge; Basic input parameters
Learning / Improvement	None

Basic Hardware Exercisers – What's Missing

- Limited confidence in the combined coverage provided by a mosaic of overlapping means
  - Functional and physical aspects
  - Coverage goals and measurement methods only partially available
- Difficulty to reproduce simulation and field failures
- Re-use of pre-silicon verification knowledge and IP
- Tool Productivity aspects

Can we Just Run the Generator as a H/W Exerciser?

- Bare metal implies no file handling for templates/tests
- Need to keep the generator simple
- Need to handle the model-based nature

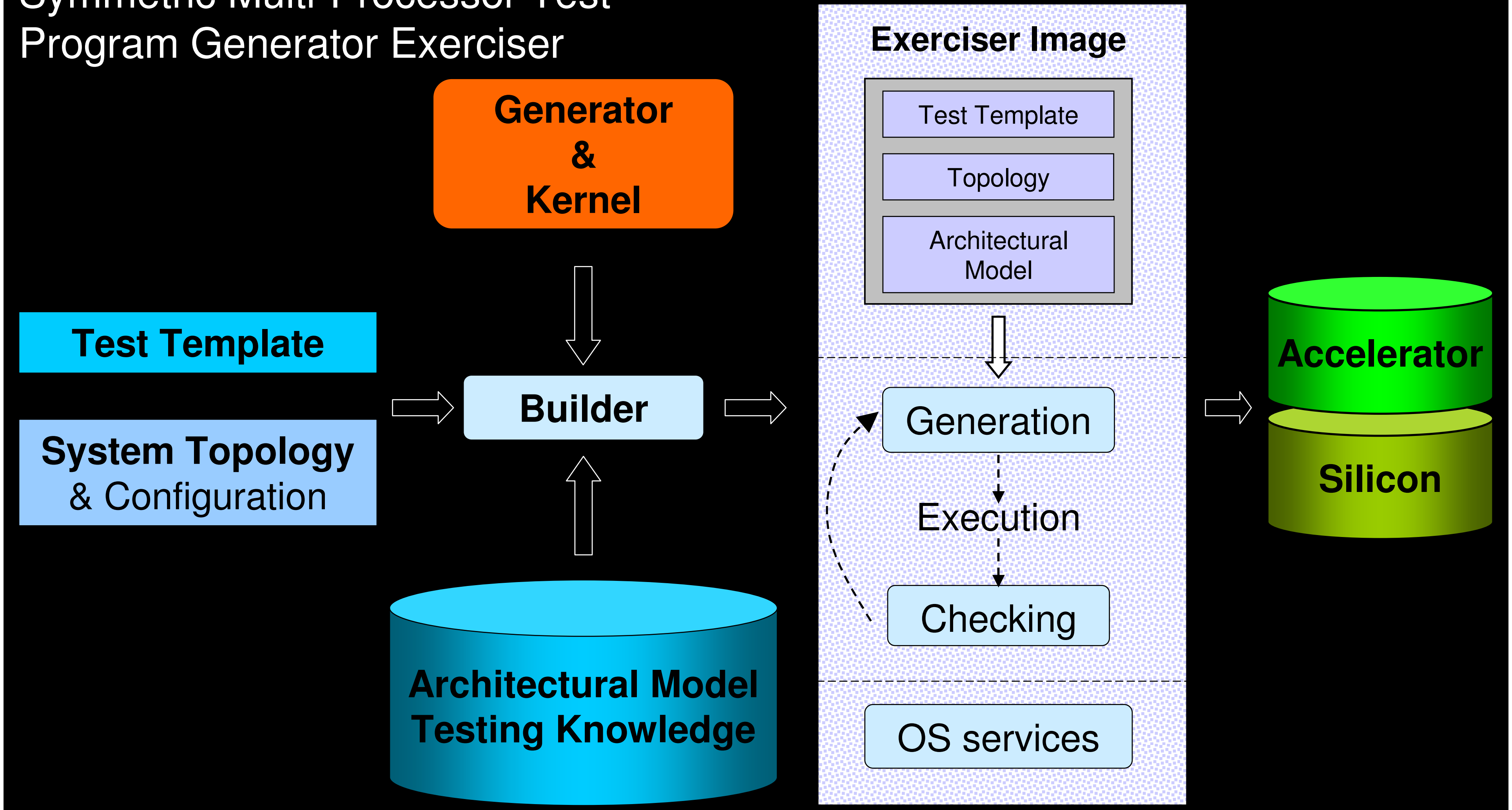


ThreadMill: a Directable Model-Based Exerciser

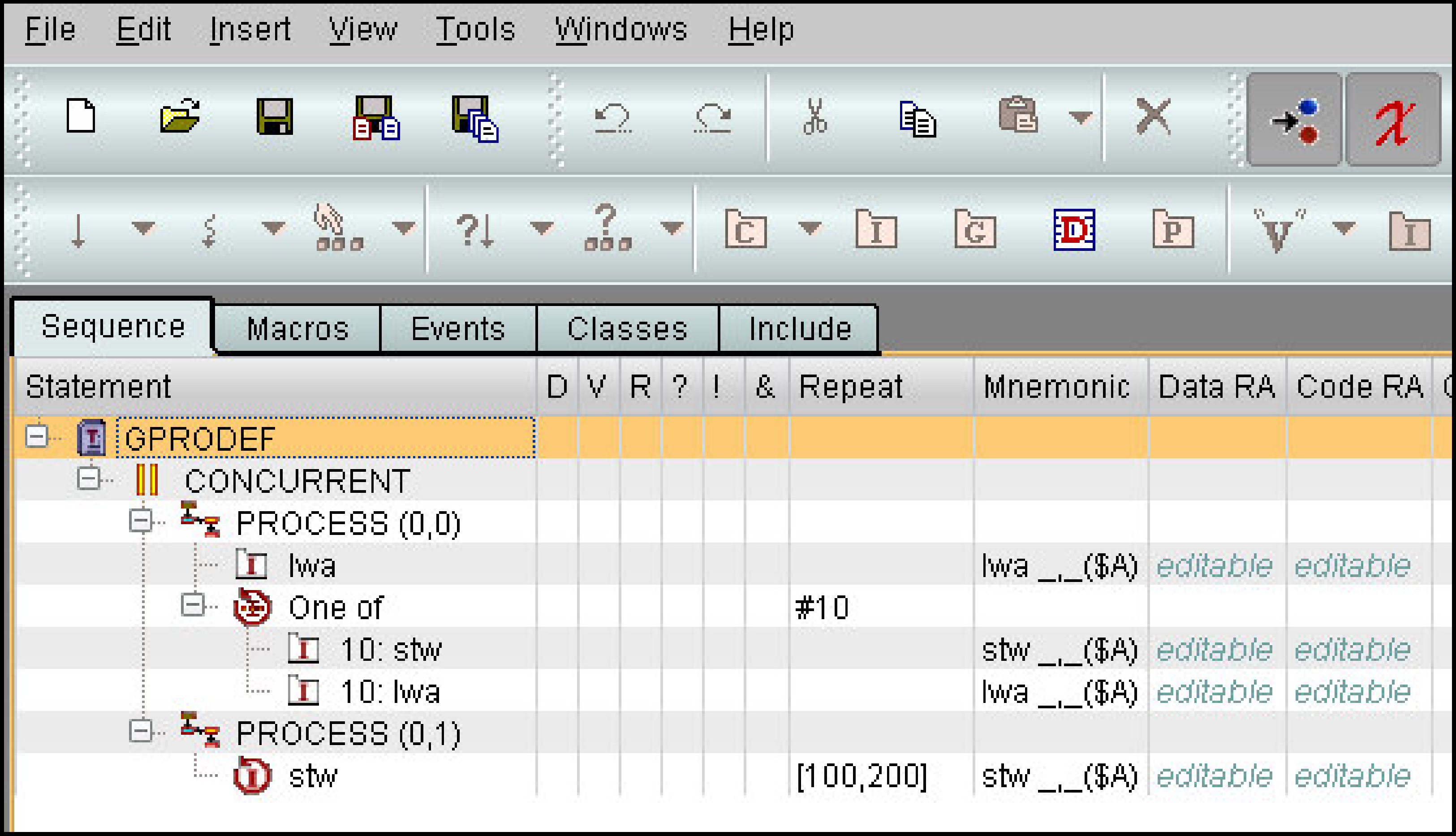
- Platform to demonstrate and enable cross-fertilization of silicon and simulation-based verification methodology
- Simple to maintain, functional-coverage oriented exerciser
- Targeting multithreading/multiprocessor configurations on silicon and acceleration platforms

Architecture of a Directable Model-Based Exerciser

Symmetric Multi-Processor Test Program Generator Exerciser



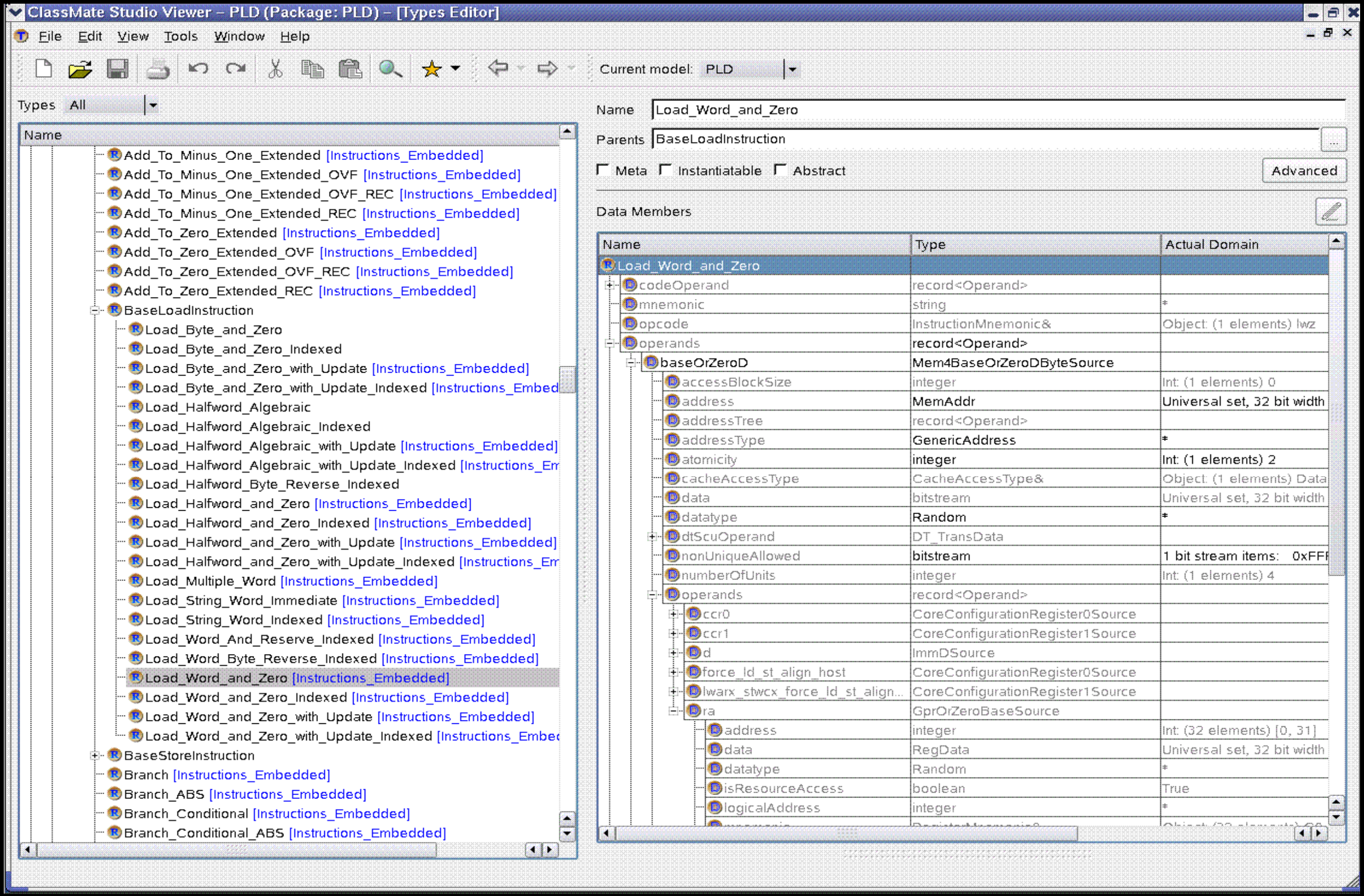
Example: Test Template



File Edit Insert View Tools Windows Help
Sequence Macros Events Classes Include

Statement	D	V	R	?	!	&	Repeat	Mnemonic	Data RA	Code RA
GPRODEF										
CONCURRENT										
PROCESS (0,0)										
lwa								lwa _,_($A)	editable	editable
One of							#10			
10: stw								stw _,_($A)	editable	editable
10: lwa								lwa _,_($A)	editable	editable
PROCESS (0,1)										
stw							[100,200]	stw _,_($A)	editable	editable

Example: Model of the Power ISA


ClassMate Studio Viewer – PLD (Package: PLD) – [Types Editor]

File Edit View Tools Window Help

Current model: PLD

Types All

Name

- Add_To_Minus_One_Extended [Instructions_Embedded]
- Add_To_Minus_One_Extended_OVF [Instructions_Embedded]
- Add_To_Minus_One_Extended_OVF_REC [Instructions_Embedded]
- Add_To_Minus_One_Extended_REC [Instructions_Embedded]
- Add_To_Zero_Extended [Instructions_Embedded]
- Add_To_Zero_Extended_OVF [Instructions_Embedded]
- Add_To_Zero_Extended_OVF_REC [Instructions_Embedded]
- Add_To_Zero_Extended_REC [Instructions_Embedded]
- BaseLoadInstruction
  - Load_Byte_and_Zero
  - Load_Byte_and_Zero_Indexed
  - Load_Byte_and_Zero_with_Update [Instructions_Embedded]
  - Load_Byte_and_Zero_with_Update_Indexed [Instructions_Embed
  - Load_Halfword_Algebraic
  - Load_Halfword_Algebraic_Indexed
  - Load_Halfword_Algebraic_with_Update [Instructions_Embedded]
  - Load_Halfword_Algebraic_with_Update_Indexed [Instructions_Em
  - Load_Halfword_Byte_Reverse_Indexed
  - Load_Halfword_and_Zero [Instructions_Embedded]
  - Load_Halfword_and_Zero_Indexed [Instructions_Embedded]
  - Load_Halfword_and_Zero_with_Update [Instructions_Embedded]
  - Load_Halfword_and_Zero_with_Update_Indexed [Instructions_Em
  - Load_Multiple_Word [Instructions_Embedded]
  - Load_String_Word_Immediate [Instructions_Embedded]
  - Load_String_Word_Indexed [Instructions_Embedded]
  - Load_Word_And_Reserve_Indexed [Instructions_Embedded]
  - Load_Word_Byte_Reverse_Indexed [Instructions_Embedded]
  - Load_Word_and_Zero [Instructions_Embedded]
  - Load_Word_and_Zero_Indexed [Instructions_Embedded]
  - Load_Word_and_Zero_with_Update [Instructions_Embedded]
  - Load_Word_and_Zero_with_Update_Indexed [Instructions_Embec
- BaseStoreInstruction
- Branch [Instructions_Embedded]
- Branch_ABS [Instructions_Embedded]
- Branch_Conditional [Instructions_Embedded]
- Branch_Conditional_ABS [Instructions_Embedded]

Name: Load_Word_and_Zero

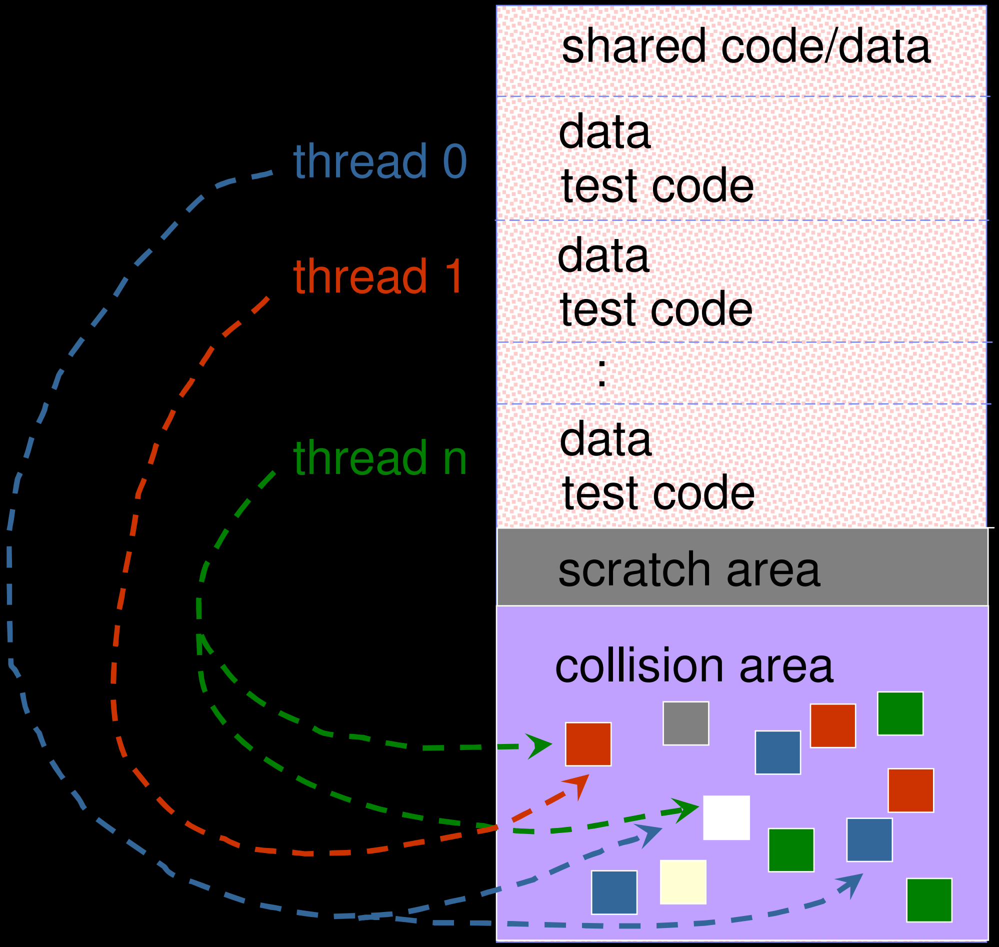
Parents: BaseLoadInstruction

☐ Meta ☐ Instantiatable ☐ Abstract — Advanced

Data Members

Name	Type	Actual Domain
Load_Word_and_Zero		
codeOperand	record<Operand>	
mnemonic	string	*
opcode	InstructionMnemonic&	Object: (1 elements) lwz
operands	record<Operand>	
baseOrZeroD	Mem4BaseOrZeroDByteSource	
accessBlockSize	integer	Int: (1 elements) 0
address	MemAddr	Universal set, 32 bit width
addressTree	record<Operand>	
addressType	GenericAddress	*
atomicity	integer	Int: (1 elements) 2
cacheAccessType	CacheAccessType&	Object: (1 elements) Data
data	bitstream	Universal set, 32 bit width
datatype	Random	*
dtScuOperand	DT_TransData	
nonUniqueAllowed	bitstream	1 bit stream items: 0xFFF
numberOfUnits	integer	Int: (1 elements) 4
operands	record<Operand>	
ccr0	CoreConfigurationRegister0Source	
ccr1	CoreConfigurationRegister1Source	
d	ImmDSource	
force_ld_st_align_host	CoreConfigurationRegister0Source	
lwarx_stwcx_force_ld_st_align...	CoreConfigurationRegister1Source	
ra	GprOrZeroBaseSource	
address	integer	Int: (32 elements) [0, 31]
data	RegData	Universal set, 32 bit width
datatype	Random	*
isResourceAccess	boolean	True
logicalAddress	integer	*

Threadmill Testing Knowledge Example: Memory map, collision scheme



Memory Image

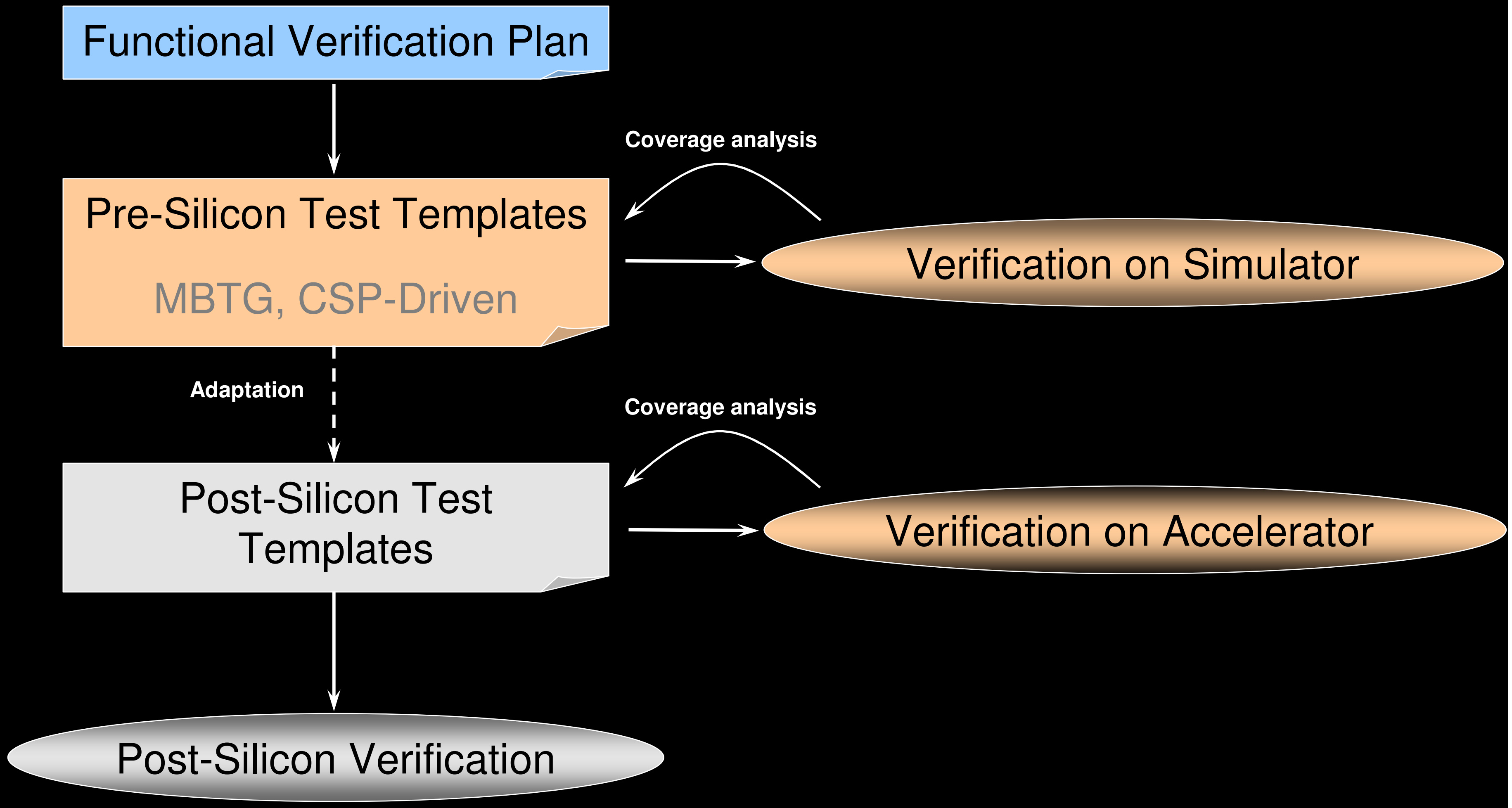
ThreadMill Deployment

- Significant role within the functional verification methodology of IBM’s Power processors
  - High-end server processor
  - Multi-processor, multi-threading
- Powerful test template language has proven effective for bug recreation on silicon
- May also be used to assist production testing

ThreadMill: Technology Score Card

Sampling Size	High: hardware speed, fast generation
Quality of each sample	High: Testing Knowledge Rich input language Reuse of pre-silicon scenarios
Learning / Improvement	None

Cross-Platform Functional Verification Methodology


Functional Verification Plan
Coverage analysis
Pre-Silicon Test Templates
MBTG, CSP-Driven
Verification on Simulator
Adaptation
Coverage analysis
Post-Silicon Test Templates
Verification on Accelerator
Post-Silicon Verification

Building a Bridge: Summary

- A functional verification methodology based on
  - A simulation platform
  - A directable test program generator
  - A set of test templates covering the functional verification plan
  - Coverage measurement and feedback

- Can be extended from pre-silicon to post-silicon by the use of a directable hardware exerciser

Thank you