

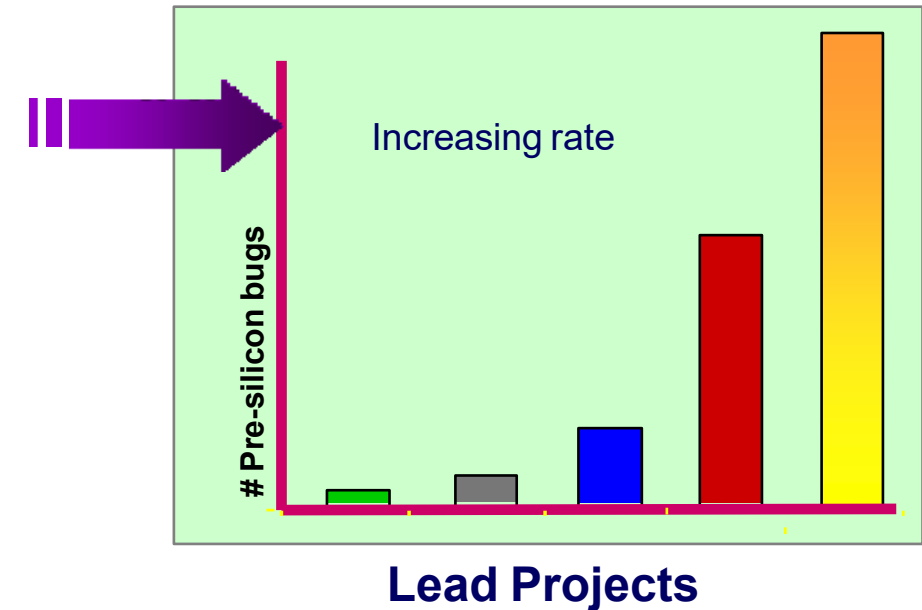
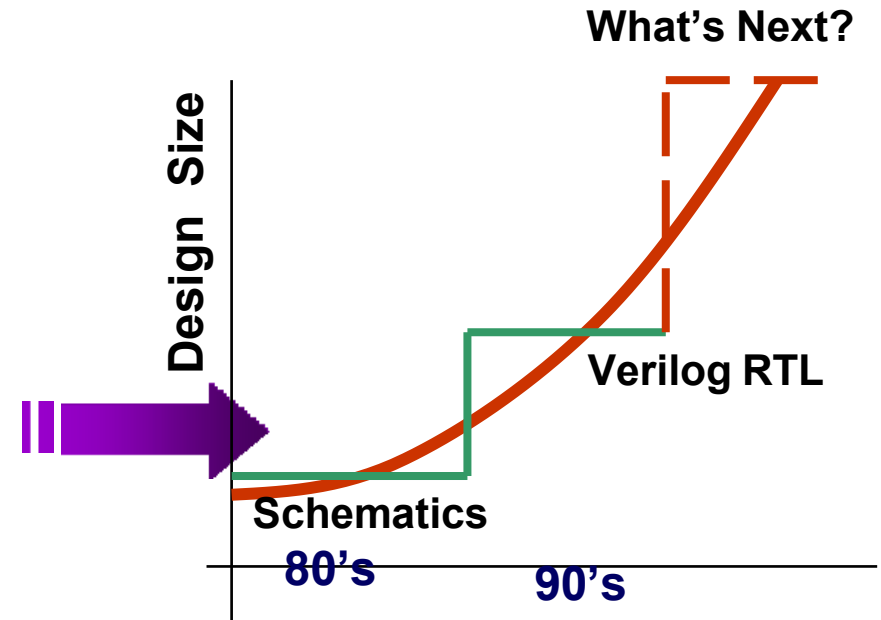
Introduction to System Verilog

HW Design Languages

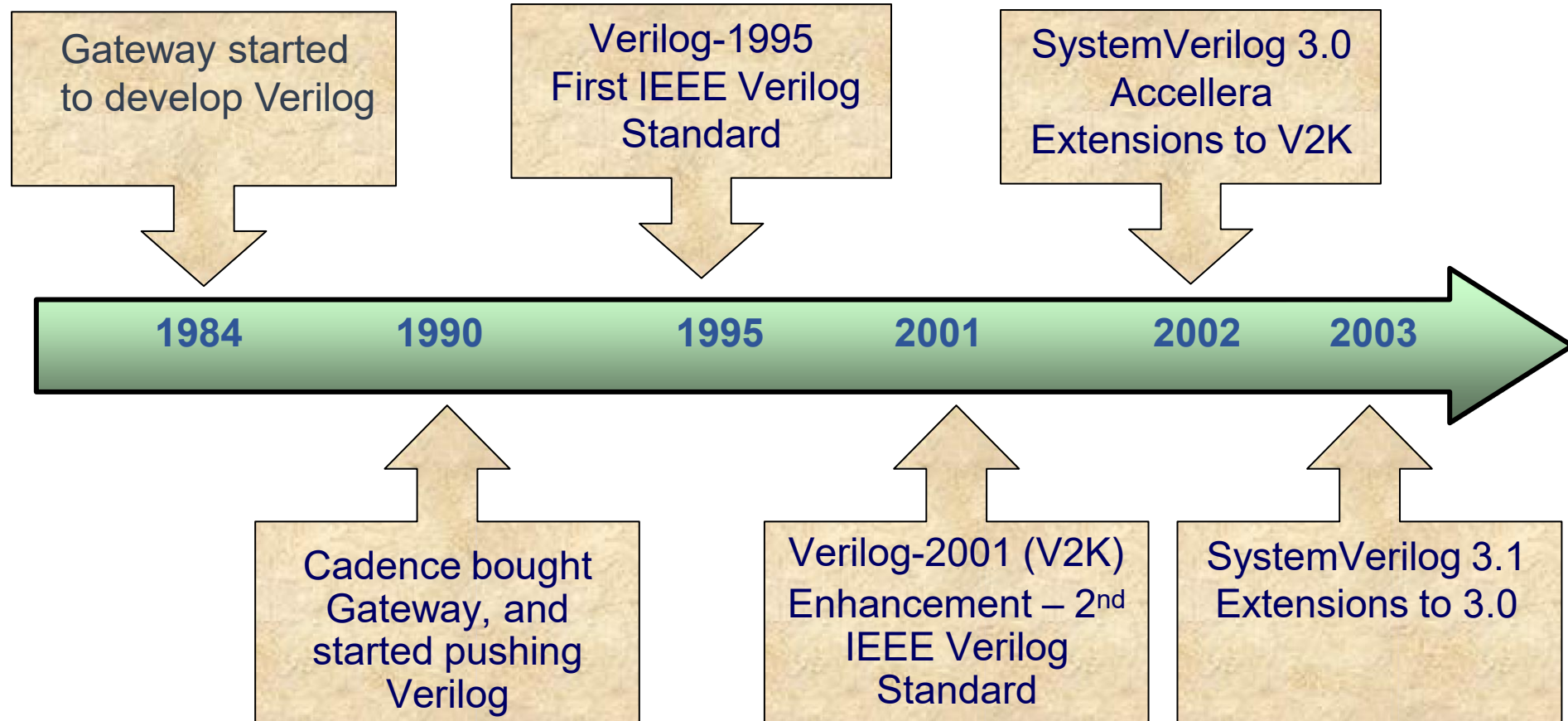
- **Hardware Description Language (HDL)** is used to describe digital hardware elements
- Various levels of design abstractions are used:
 - **Behavioral**: flow control, arithmetic operators, complex delays
 - **Register Transfer Level (RTL)**: Structural description of the registers and the signal changes between registers
 - **Gate level**: combinatorial logic gates (and, or, not,...)
 - **Switch level**: layout description of the wires, resistors and transistors (CMOS, PMOS, etc)
- Two main subsets:
 - Synthesizable – reflecting HW / Silicon
 - Non-Synthesizable – reflecting *instrumentation* code

The Problem!

- Quest for Performance is driving up complexity
 - Deeper pipelines, increase in logic functionality and complexity, power issues, explosion in flops
- Explosion in lines of RTL Code making verification a lot harder
- Low Abstraction level of the RTL is driving higher verification effort and lower simulation speed
 - Trillions of cycles per lead project and more pre-silicon bug escapes
- Verification effort is reaching 60% of the total design cycle
 - Usage of different languages makes it even harder: reference models in C/C++, RTL in Verilog or VHDL, Test Benches, Assertions, Checkers, Coverage



Verilog History



SystemVerilog

SystemVerilog 3.1 - HDVL

SV3.1 Focus: design language cleanup

Test Bench

Assertions

APIs

OO Classes

Semaphores

Queues
& Lists

System Verilog 3.0

SV3.0 Focus: enhance design language capabilities

Interfaces

Data Types
& Enums

Structures
& Unions

Advanced
Operators

Control
Flow

Casting

Verilog2K

Multi-D Arrays

Generate

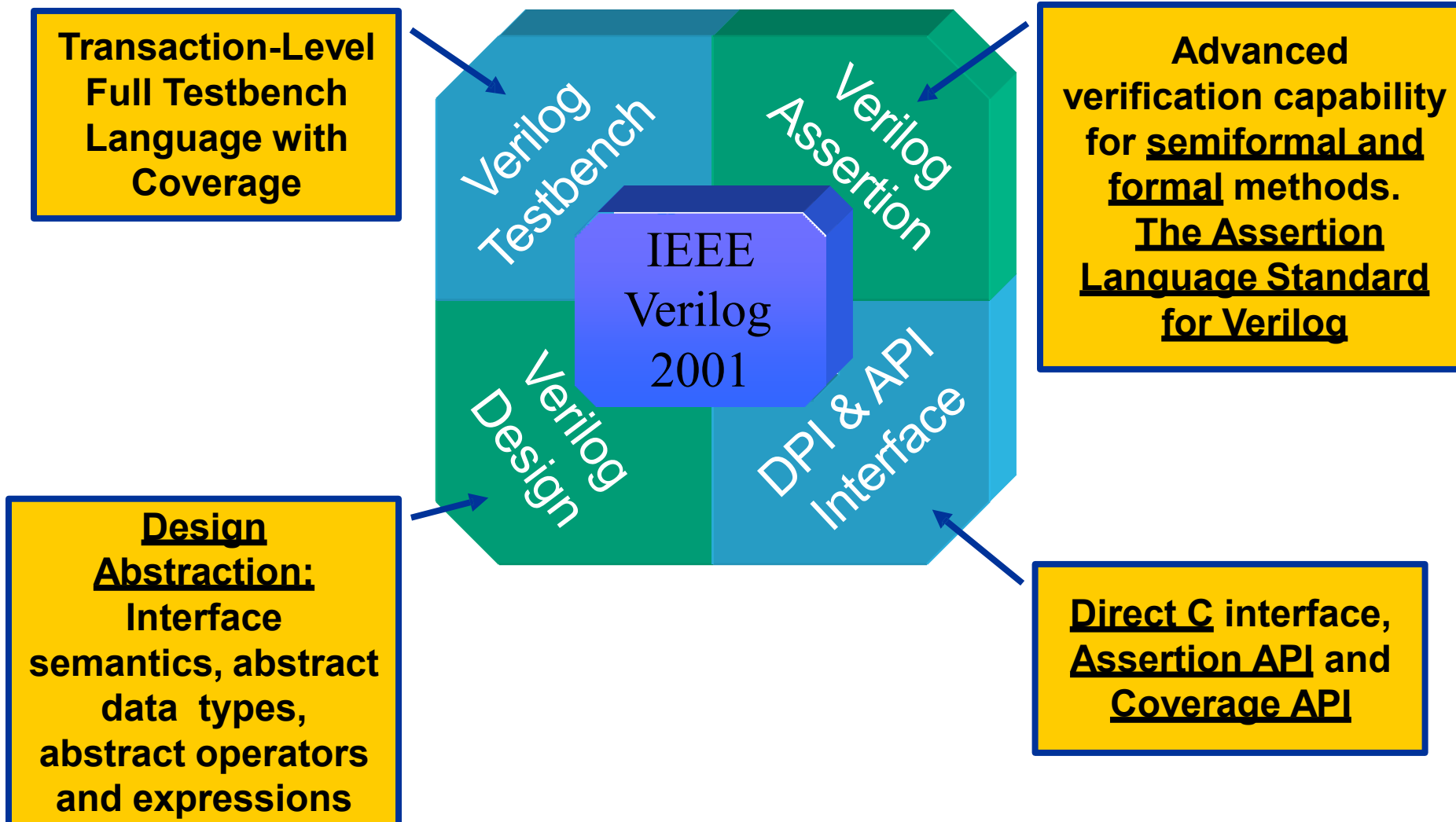
Automatic Tasks

Verilog95

Gate Level Modeling & Timing

Hardware Concurrency

SystemVerilog Components



SystemVerilog Design Modeling

SystemVerilog enhances Verilog for Design Modeling

- Data Types
 - SystemVerilog Data Types
 - Packed & Unpacked Arrays
- Data Organization
 - Structures & Unions
 - Type Casting
 - Enumerated Data Types
- C-like functionality
- Capturing Design Intent
 - always_* Procedural blocks
 - Unique and Priority Case
 - Nets and Variables
- Powerful Syntax
 - Copy / Buffering
 - Port Connections

Advantages of System Verilog

- More compact code compared to Verilog
- Supports OOP concept
- Structures and enumerated types for better scalability
- Interfaces for higher level of abstraction
- Supported in Vivado synthesis

Drawbacks using SV

1. **Steep learning curve**: SystemVerilog has a complex syntax and requires significant expertise to master.
2. **Verification complexity**: SystemVerilog's verification capabilities can be challenging to manage, especially for large-scale designs.
3. **Simulation performance**: SystemVerilog simulations can be slower compared to other HDVLs.
4. **Limited synthesis support**: Some SystemVerilog features are not synthesizable, making them unsuitable for design implementation.
5. **Tool support**: Not all EDA tools support SystemVerilog equally, which can lead to compatibility issues.

6. **Debugging difficulties:** SystemVerilog's complexity can make debugging more challenging.

7. **Over-engineering:** SystemVerilog's advanced features can lead to over-engineering, increasing design complexity.

8. **Compatibility issues:** SystemVerilog code may not be compatible with all simulators or synthesis tools.

9. **Limited analog and mixed-signal support:** SystemVerilog is primarily digital-focused, limiting its effectiveness for analog and mixed-signal designs.

10. **Industry adoption:** While widely used, SystemVerilog hasn't been universally adopted, potentially limiting collaboration and knowledge sharing.

SystemVerilog for Verification

- SystemVerilog extends the language to include Verification and Test-Bench modeling constructs
- Extended Data Types
 - Dynamic Arrays
 - Associative Arrays
- Process Synchronization
 - Semaphors and Event Variables
 - Mailbox queues
 - Processes and Threads
- Object Oriented Programming
 - Class; Object; Methods
- Communication Encapsulation
 - Interfaces
- Assertion Based Design
 - Assertions Constructs
 - Usage

Basic SV3.1 Data Types

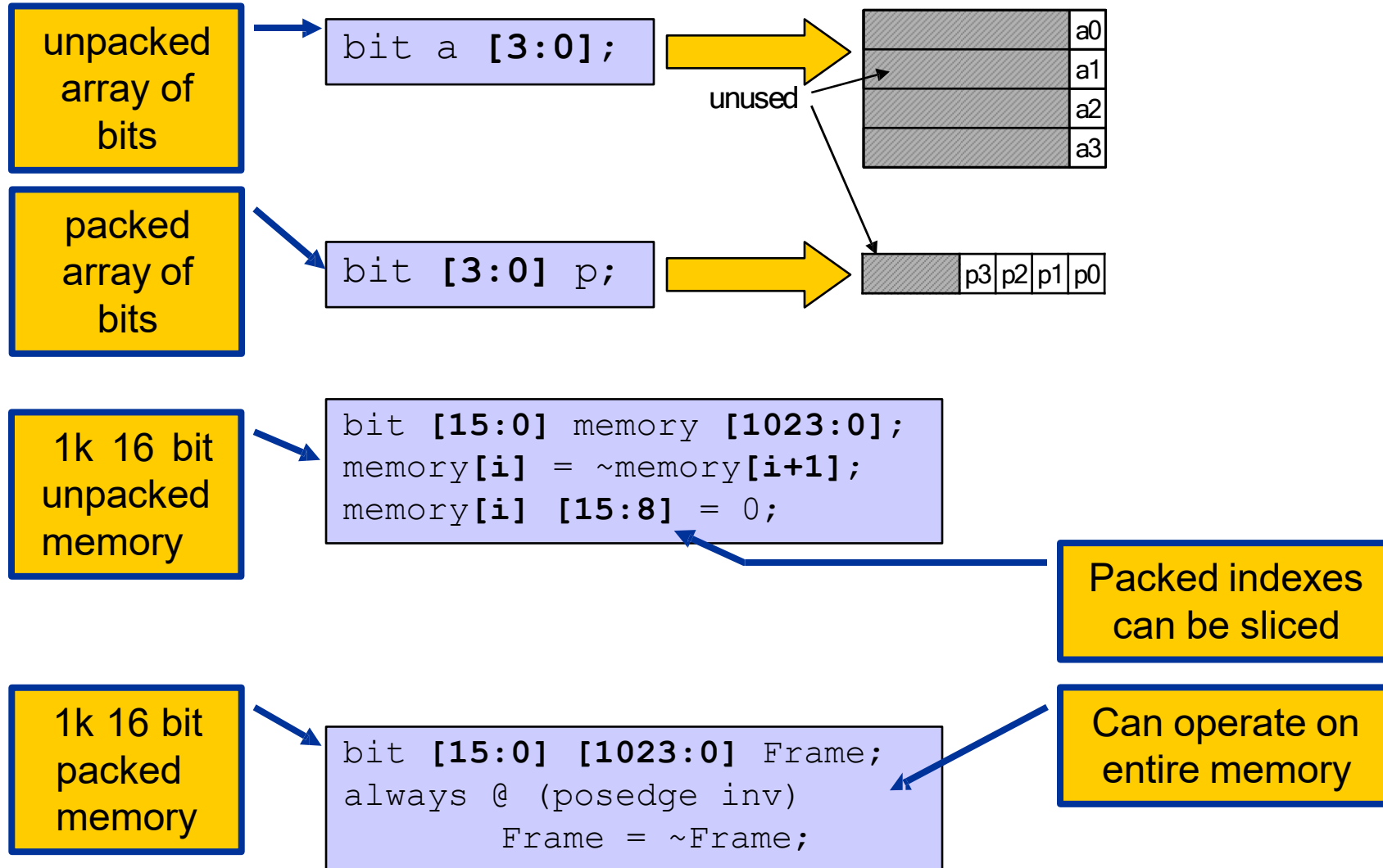
```
reg r;      // 4-value Verilog-2001 single-bit datatype
integer i;  // 4-value Verilog-2001 >= 32-bit datatype
bit b;      // single bit 0 or 1
logic w;    // 4-value logic, x 0 1 or z as in Verilog
byte b;     // 2 value, 8 bit signed integer
int i;      // 2-value, 32-bit signed integer
shortint s; // 2-value, 16-bit signed integer
longint l;  // 2-value, 64-bit signed integer
```

Make your own types using typedef

Use typedef to get C compatibility

typedef	shortint	short;
typedef	longint	longlong;
typedef	real	double;
typedef	shortreal	float;

Packed And Unpacked Arrays



Data Organization

- Signals are Meaningful In Groups
 - Instructions: Operation, Operands
- Verilog Provides Only Informal Grouping

```
reg [47:0] PktSrcAdr;  
reg [47:0] PktDstAdr;  
reg [7:0]  InstOpCde;  
reg [7:0]  InstOpRF [127:0];
```

By Name

```
reg [31:0] Instruction;  
`define opcode 31:16  
Instruction[`opcode]
```

By Vector Location

- Better to organize data in explicit, meaningful relationships between data elements
- SystemVerilog Structs, Unions & Arrays alone or combined better capture design intent