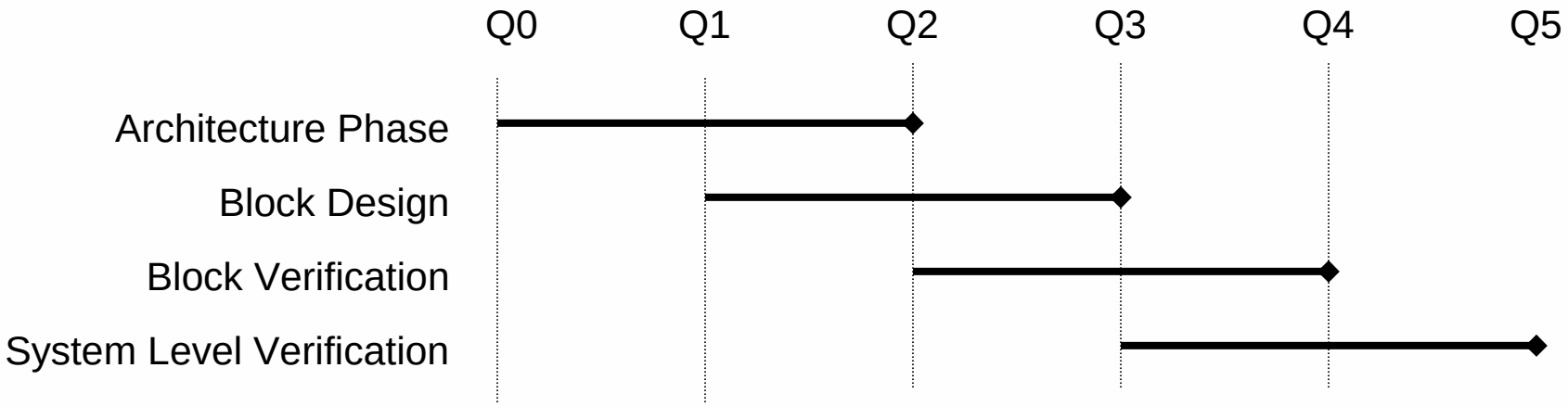
The SystemC Verification Standard (SCV)

Stuart Swan
Senior Architect
Cadence Design Systems, Inc.
stuart@cadence.com

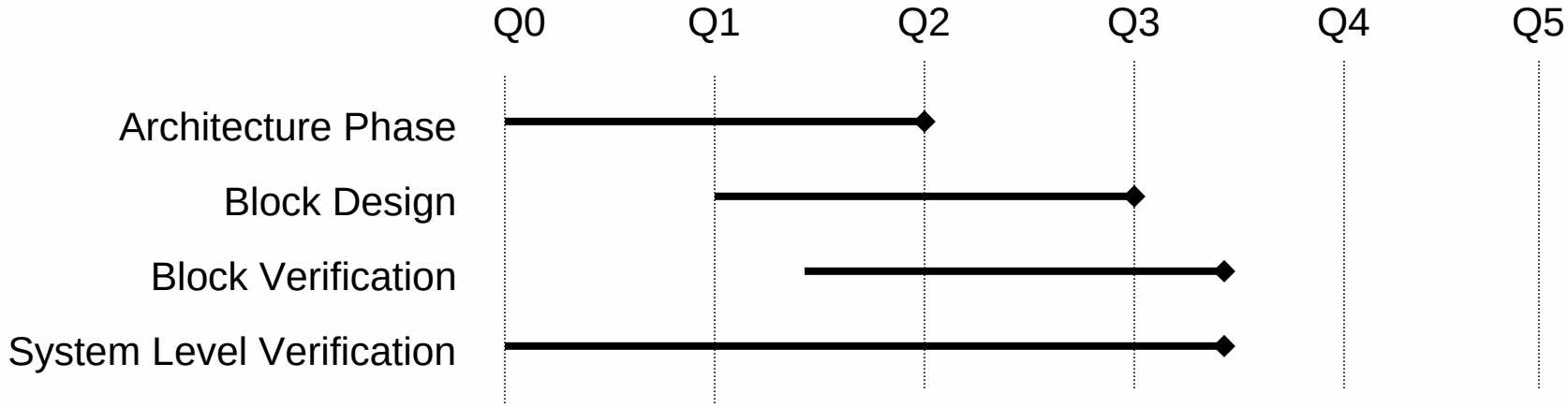
The Verification Problem…

System Level Verification is typically done last, is typically on the critical path - and is typically done too late for architectural redesign



Advantages of Verification with SystemC

System Level Verification can be done throughout the lifetime of the project, and the same verification components can be reused for both block verification and architectural exploration and optimization



SystemC Verification Working Group History and Membership

- **Joint Proposal by Fujitsu, ST and Motorola – June 2001**
  - SystemC 2.0 provides a platform upon which various design methodologies can be built
  - SystemC should also provide a platform upon which various verification methodologies can be built
- **Membership**
  - Cadence, Forte, Synopsys
  - ARM, Axys, Elixent, Fujitsu, Infineon, Motorola, Philips
  - Universities : Chemnitz, Tuebingen

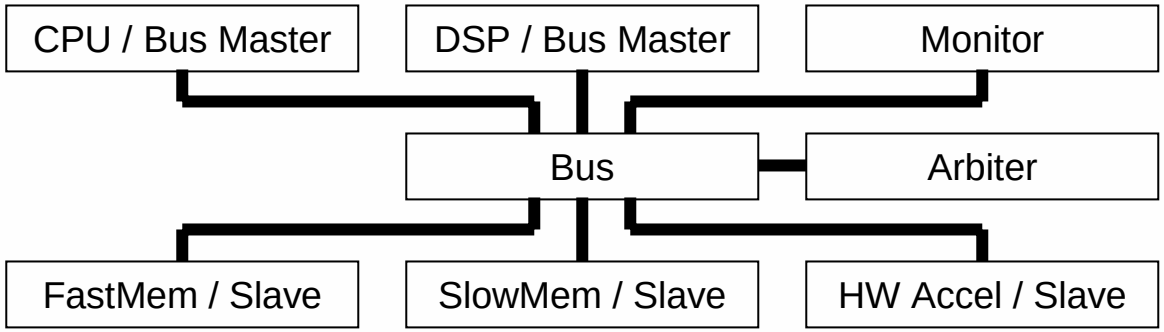
SystemC Verification Standard (SCV) Status

- SCV 1.0 specification approved by OSCI steering group in Sept. 2002.
- SCV 1.0 reference implementation (beta) made publicly available via www.systemc.org in December 2002.
- Expectation is for final SCV 1.0 production code around March 2003.
- Areas Covered in SCV 1.0
  - Data Introspection (similar to Verilog PLI but for C/C++ data structures)
  - Randomization and seed management for reproducibility of simulation runs
  - Constrained randomization
  - Weighted randomization
  - Transaction Monitoring and Recording
  - Sparse Array Support
- SCV 1.0 provides the key capabilities needed to construct advanced reusable verification IP in SystemC *today*

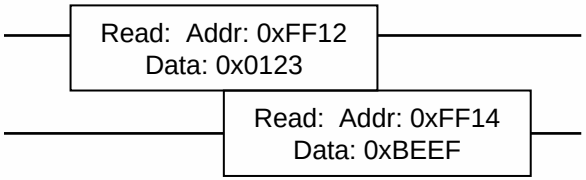
Future Work for SystemC VWG

- **Short Term**
  - Respond to SCV 1.0 review feedback
  - Provide SCV 1.0 production release
- **Medium Term**
  - Extend SCV documentation, examples, tutorial
  - Consider SCV extensions, possibly including:
    - Functional Coverage
    - Assertions and temporal expressions

Transaction-Level Modeling in SystemC

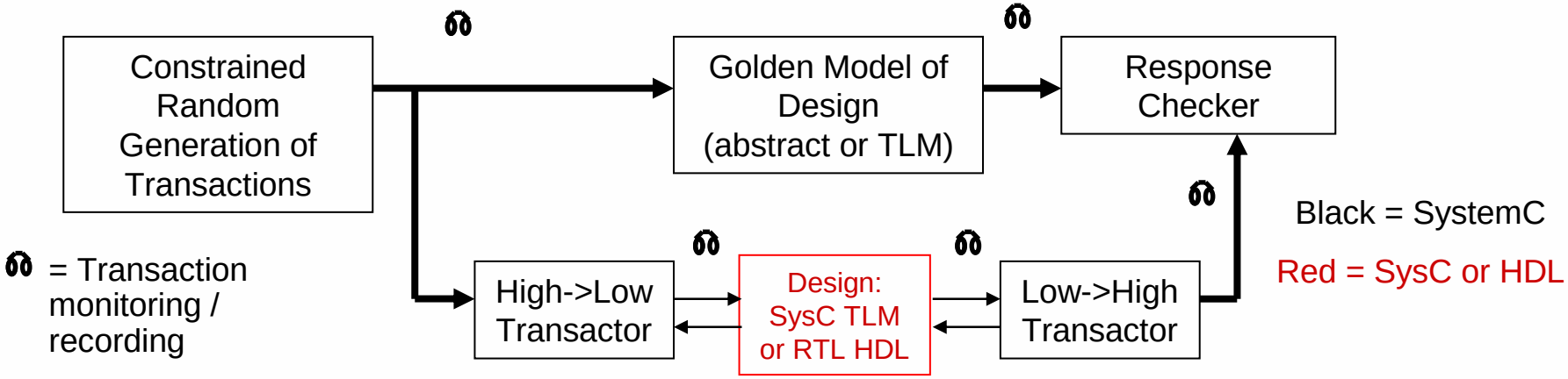


Communication between modules is modeled using function calls that represent transactions. No signals are used.



- **Why do transaction-level modeling in SystemC?**
  - **Models are relatively easy to develop and use**
  - **HW and SW components of a system can be accurately modeled. Typically bus is cycle-accurate, and bus masters / slaves may or may not be cycle-accurate.**
  - **Extensive system design exploration and verification can be done early in the design process, before it's too late to make changes**
  - **Models are fast – typically about 100K clock cycles per second, making it possible to execute significant amounts of the system's software very early in the design process**
- **Transaction-level modeling is extensively covered in the *System Design with SystemC* book and the code for the *simple_bus* design is provided**

Transaction-Based Verification in SystemC

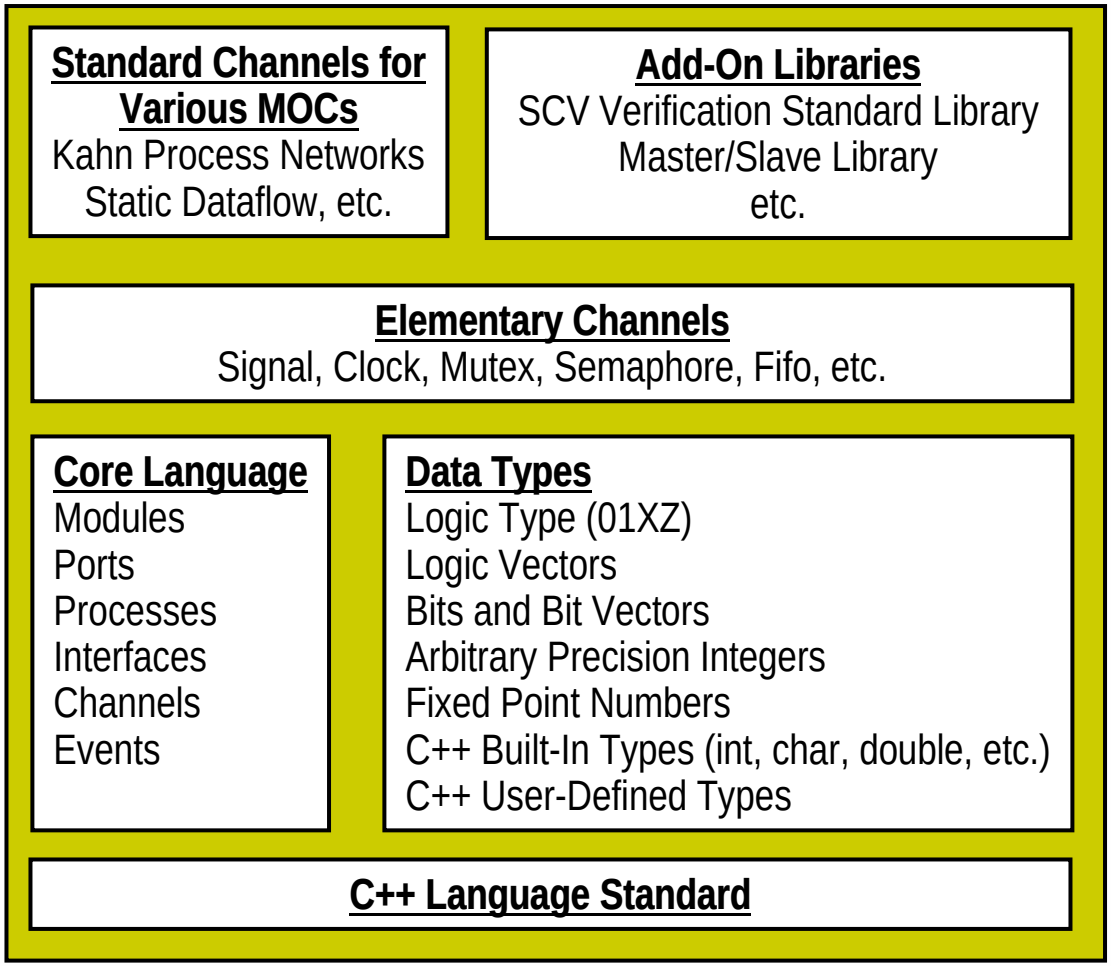


- **Why do transaction-based verification in SystemC?**
  - **Ability to have everything (except perhaps RTL HDL) in SystemC/C++ provides great benefits: easier to learn and understand, easier to debug, higher performance, easy to integrate C/C++ code & models, open source implementation, completely based on industry standards**
  - **Allows you to develop smart test benches early in the design process (before developing detailed RTL) to find bugs and issues earlier. Enables test bench reuse throughout the design process.**
  - **Much more efficient use of verification development effort and verification compute resources**
- **Transaction-Based Verification in SystemC is described in detail in the *SCV Specification*, and in the documentation and examples included with the OSCI SCV reference implementation kit.**

SystemC 2.0 Language Architecture

Upper layers are built cleanly on lower layers.

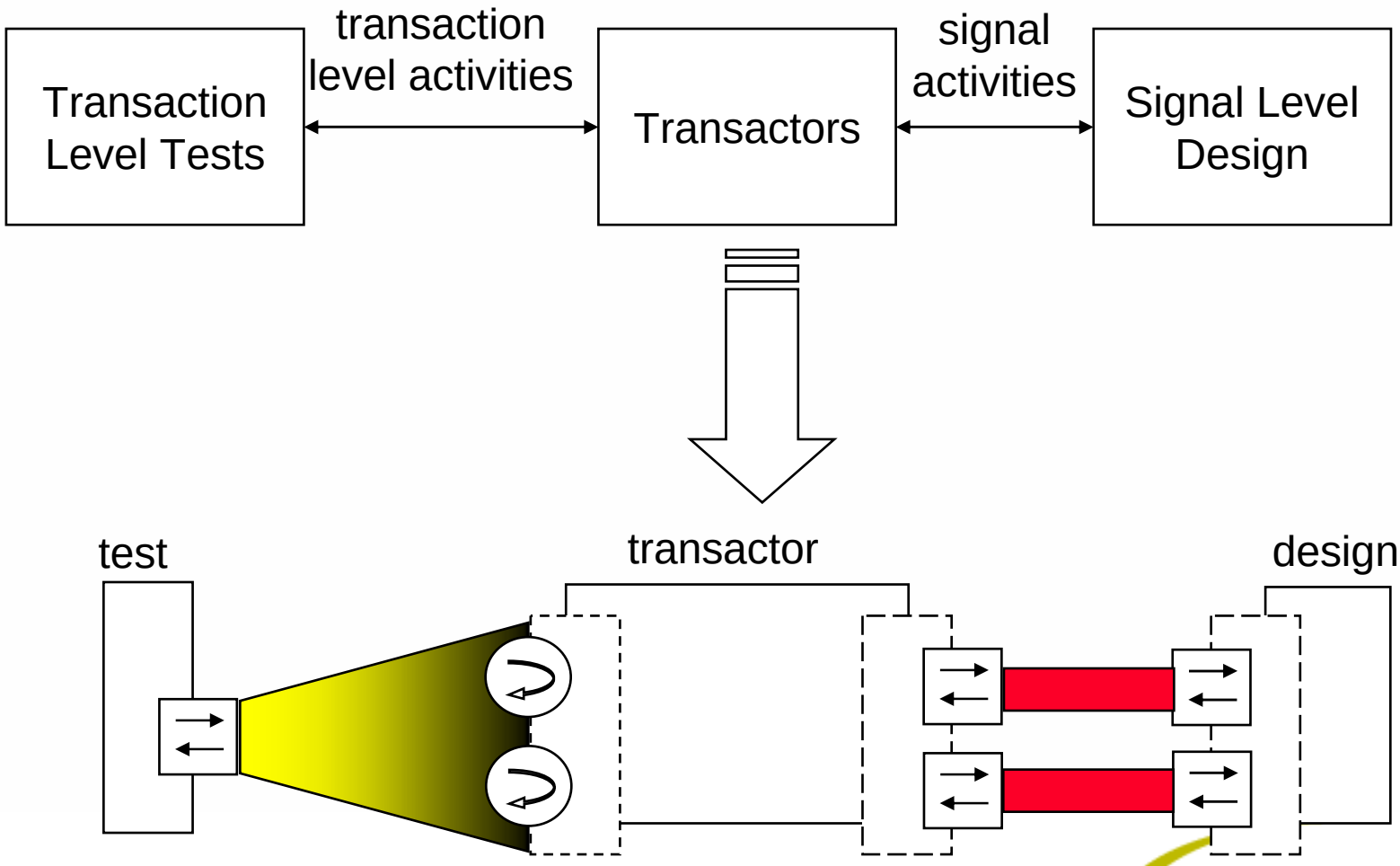
Lower layers can be used without upper layers.



SCV Builds Cleanly on SystemC and C++

- **Many features in other hardware verification languages such as Vera and Veristy's 'e' aren't provided in SCV because they are already in SystemC or C++. For example:**
  - **Classes, templates, inheritance (C++)**
  - **Hardware-oriented datatypes (SystemC)**
  - **Modules, ports, processes, interfaces, channels, events (SystemC)**
  - **Semaphores, fifos, signals, etc. (SystemC)**
  - **Dynamic thread creation (SystemC 2.1, *forkjoin* example in 2.0.1)**
  - **Vectors, maps, lists, associative arrays, etc. (C++ STL)**
  - **Connection to HDL simulators (SystemC / EDA vendors)**

SystemC 2.0 already supports transactors

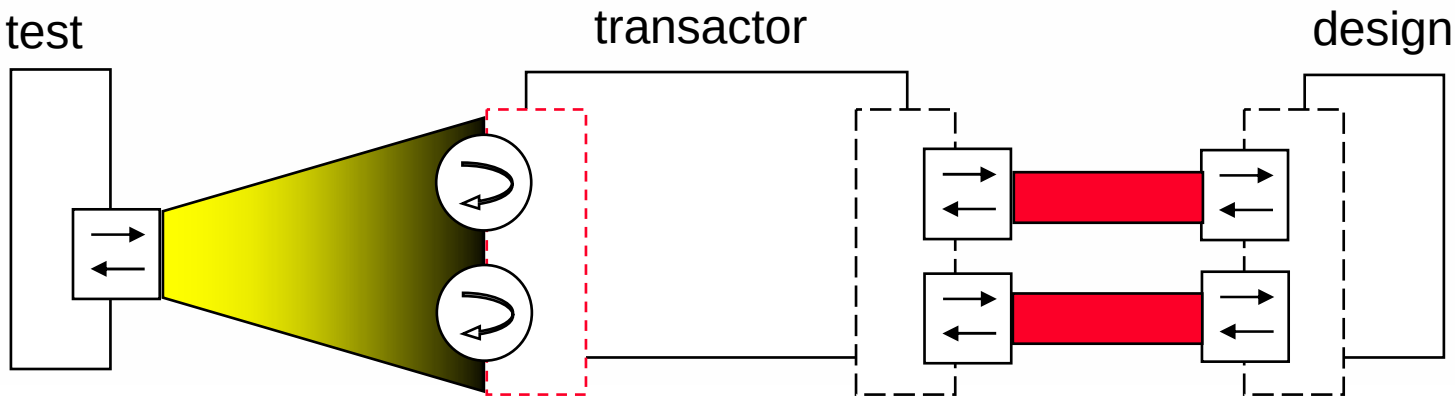


SystemC 2.0 already supports transactors

```
class transactor_if :
public virtual sc_interface {
public:
    virtual int read ( unsigned addr) = 0;
};
```

```
class design_ports :
public sc_module {
public:
    sc_in < bool > clk;
    sc_inout < sc_int<48> > data;
};
```

```
class transactor :
    public design_ports,
    public transactor_if {
public:
    int read ( unsigned addr ) {
        wait( clk.posedge_event() )
        return data;
};
```



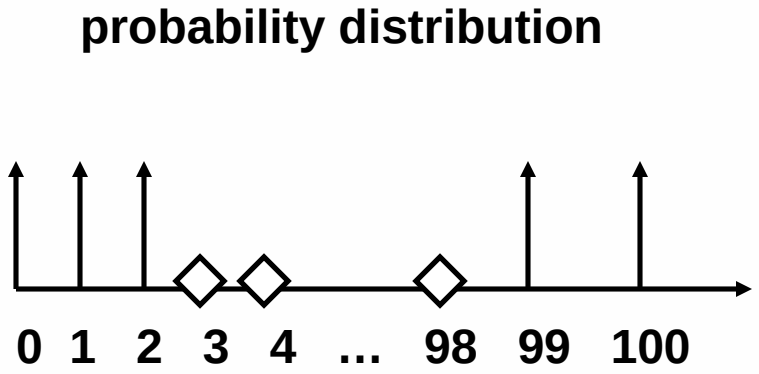
SCV Stimulus Generation Techniques

- **Directed Tests**
  - Traditional way to stimulate designs
- **Weighted Randomization**
  - Helps focus stimulus generation on interesting test scenarios
- **Constrained Randomization**
  - Enables complex and thorough tests to be developed quickly by declaring constraints among parts of stimulus data
- **All of the above techniques can be combined as needed**

Randomization using distributions

Creating a simple distribution without weights:

```
scv_smart_ptr<int> p;
p->keep_only(0,100);
p->keep_out(3,98);
p->next();
```



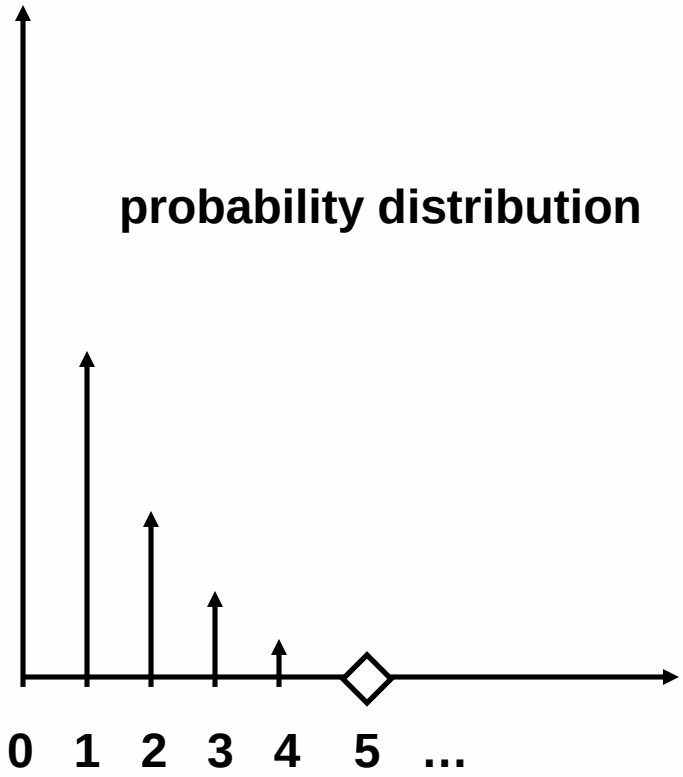
SYSTEMC™

Weighted Randomization

Creating a distribution with weights on discrete values:

```
scv_bag<int> dist;
dist.add(0,16);
dist.add(1,8);
dist.add(2,4);
dist.add(3,2);
dist.add(4,1);

scv_smart_ptr<int> p;
p->set_mode(dist);
p->next();
```



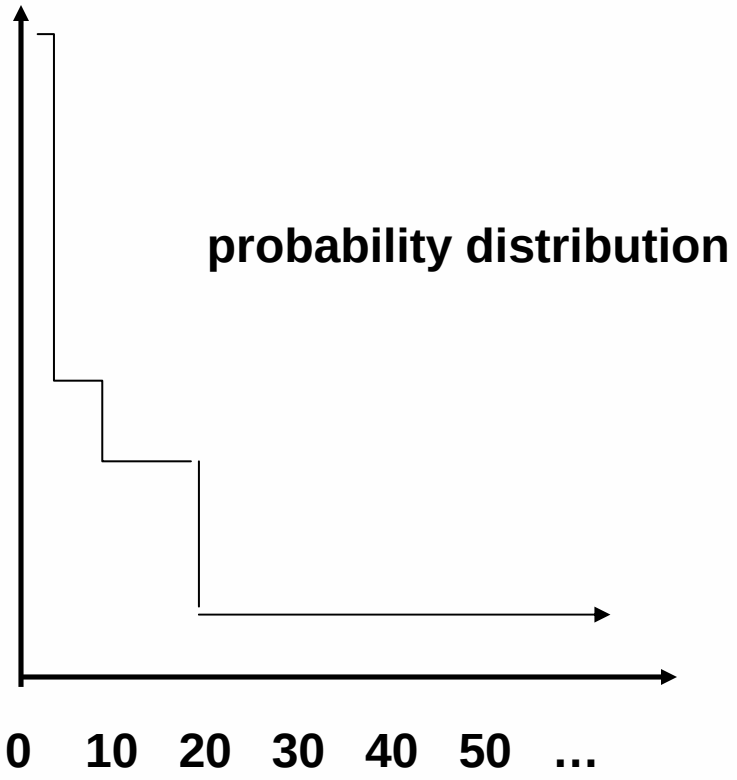
SYSTEMC™

Weighted Randomization

Creating a distribution with weights on ranges:

```
scv_bag<pair<int, int> > dist;
dist.add(pair<int,int>(1,3), 100);
dist.add(pair<int,int>(4, 10), 30);
dist.add(pair<int,int>(11, 20), 20);
dist.add(pair<int,int>(21, 80), 80);

scv_smart_ptr<int> p;
p->set_mode(dist);
p->next();
```



SYSTEMC™

Constrained Randomization

```
class packet_t
{
   sc_uint<8>     src;
   sc_uint<8>     dest;
   sc_uint<32>    data[8];
}
```

Constrained Randomization

```
class packet_t
{
    sc_uint<8>     src;
    sc_uint<8>     dest;
    sc_uint<32>    data[8];
}
```

```
// randomize whole packet
scv_smart_ptr < packet_t > p ;
p->next( );
```

Constrained Randomization

```
class packet_t
{
   sc_uint<8>    src;
   sc_uint<8>    dest;
   sc_uint<32>   data[8];
}
```

```
// randomize whole packet
scv_smart_ptr < packet_t > p ;
p->next( );
```

```
// keep src fixed
scv_smart_ptr < packet_t > p ;
p->src.disable_randomization();
p->next( );
```

Constrained Randomization

```
class packet_t
{
   sc_uint<8>     src;
   sc_uint<8>     dest;
   sc_uint<32>    data[8];
}
```

```
// randomize whole packet
scv_smart_ptr < packet_t > p ;
p->next( );
```

```
// keep src fixed
scv_smart_ptr < packet_t > p ;
p->src.disable_randomization();
p->next( );
```

```
// basic constraint
class my_constraint : public scv_constraint_base {
public:
   scv_smart_ptr < packet_t > p ;
   SCV_CONSTRAINT_CTOR( my_constraint ) {
      SCV_CONSTRAINT ( p -> src ( ) != p -> dest ( ) );
      for ( int i = 0; i < 8 ; ++ i )
         SCV_CONSTRAINT ( p -> data [ i ]() < 10 );
   }
};
```

Constrained Randomization

```
class packet_t
{
   sc_uint<8>     src;
   sc_uint<8>     dest;
   sc_uint<32>    data[8];
}
```

```
// randomize whole packet
scv_smart_ptr < packet_t > p ;
p->next( );
```

```
// keep src fixed
scv_smart_ptr < packet_t > p ;
p->src.disable_randomization();
p->next( );
```

```
// constrain packet
my_constraint c( “constraint” );
c.p->next( );
```

```
// basic constraint
class my_constraint : public scv_constraint_base {
public:
   scv_smart_ptr < packet_t > p ;
   SCV_CONSTRAINT_CTOR( my_constraint ) {
      SCV_CONSTRAINT ( p -> src ( ) != p -> dest ( ) );
      for ( int i = 0; i < 8 ; ++ i )
         SCV_CONSTRAINT ( p -> data [ i ] < 10 );
   }
};
```

Constrained Randomization

```
class packet_t
{
   sc_uint<8>     src;
   sc_uint<8>     dest;
   sc_uint<32>    data[8];
}
```

```
// randomize whole packet
scv_smart_ptr < packet_t > p ;
p->next( );
```

```
// keep src fixed
scv_smart_ptr < packet_t > p ;
p->src.disable_randomization();
p->next( );
```

```
// constrain packet
new_constraint c( “constraint” );
c.p->next( );
```

```
// basic constraint
class my_constraint : public scv_constraint_base {
public:
   scv_smart_ptr < packet_t > p ;
   SCV_CONSTRAINT_CTOR( my_constraint ) {
      SCV_CONSTRAINT ( p -> src ( ) != p -> dest ( ) );
      for ( int i = 0; i < 8 ; ++ i )
         SCV_CONSTRAINT ( p -> data [ i ]() < 10 );
   }
};
```

```
// extended constraint
class new_constraint : public my_constraint {
public:
   SCV_CONSTRAINT_CTOR( new_constraint ) {
      SCV_CONSTRAINT_BASE( my_constraint )
      SCV_CONSTRAINT ( p -> data [ 0 ]() != p -> data [ 1 ]() );
   }
};
```


SYSTEMC™

Transaction Monitoring and Recording

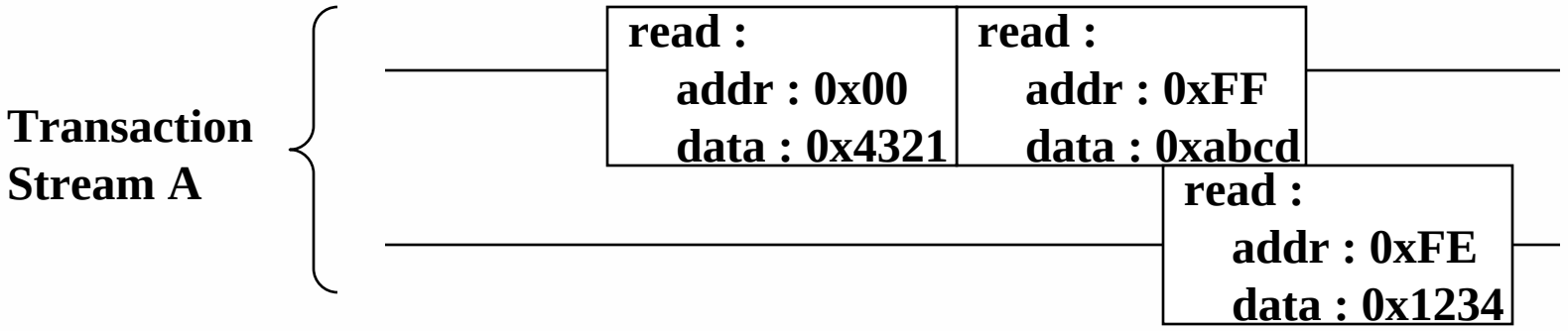
- SCV provides transaction monitoring and recording capabilities that enable users to analyze the transactions within their designs
  - Much easier and more efficient than analyzing signal waveforms
- The same API can be used to dynamically monitor transactions or to record transactions into a database
- An ASCII database is supported by SCV, but other databases can be easily plugged into the SCV transaction API

Transaction Monitoring and Recording

- **The SCV Transaction API uses the following concepts:**
  - **A *transaction* has a begin time, end time, and a set of data attributes (e.g. int address, int data)**
  - **A *generator* (scv_tr_generator<>) creates instances of transactions of a specific type (e.g. burst reads, interrupt, etc.).**
  - **A *stream* (scv_tr_stream) groups related and potentially overlapping transactions together.**
  - **A *database* (scv_tr_db) contains a set of transaction streams.**
  - **A *transaction handle* (scv_tr_handle) provides access to a particular transaction instance and enables transaction links to be created**

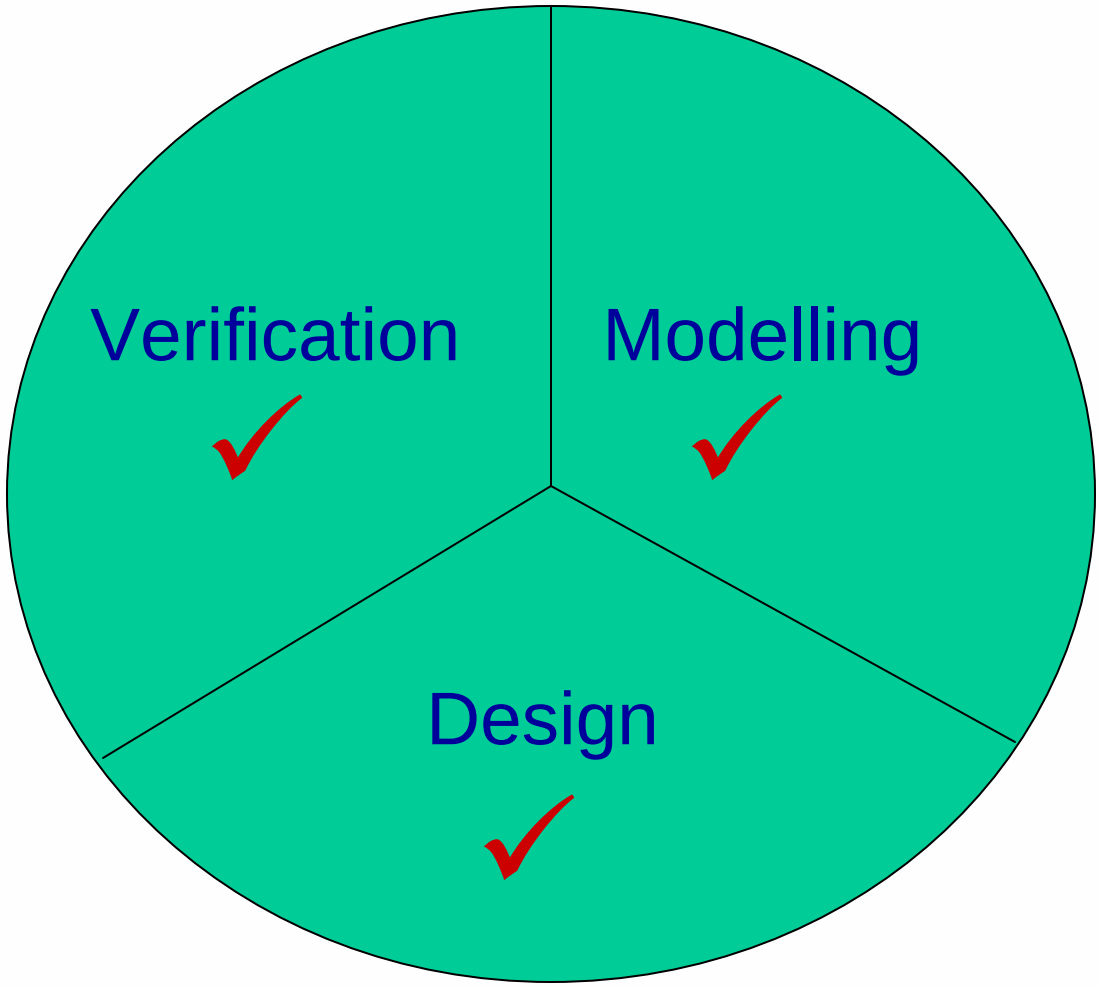
Transaction Monitoring and Recording

```
// basic transaction monitoring & recording
scv_tr_generator< int, int > read_gen ( "read", … );
scv_tr_handle h = read_gen.begin_transaction( addr );
…
read_gen.end_transaction( h , data );
```



transactions can overlap one another

Why SystemC ?



Learning more about SCV

- **Download the SCV reference implementation, documentation, examples and tutorial!**
  - **Read information at www.systemc.org**
  - **Download and unpack the kit**
  - **Read the README file**
  - **Documentation is in the “doc” directory**
  - **Examples and tutorial are in the “examples” directory**

Learning more about SystemC

- Our new book is available at:
  - www.systemc.org
    - Products & Solutions
    - Books
    - *System Design with SystemC*
- Provides an in-depth discussion of new SystemC features and using SystemC for TLM
- Available now in English, Japanese and soon in Korean