

Learning the Portable Stimulus Standard

Tom Fitzpatrick

CEO Big Fish EDA Consulting

On behalf of the Accellera Systems Initiative

and the Accellera Portable Stimulus Working Group



Session Review

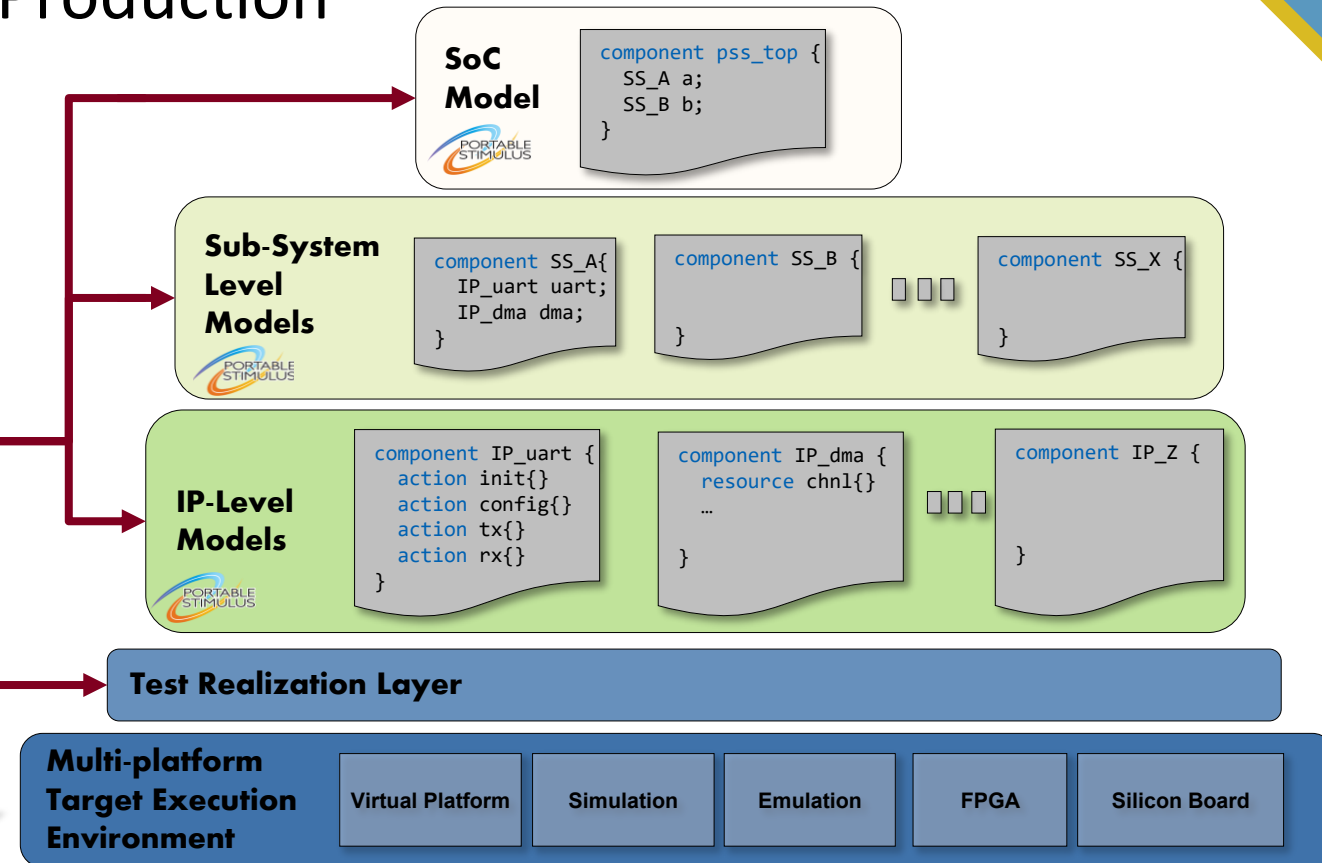
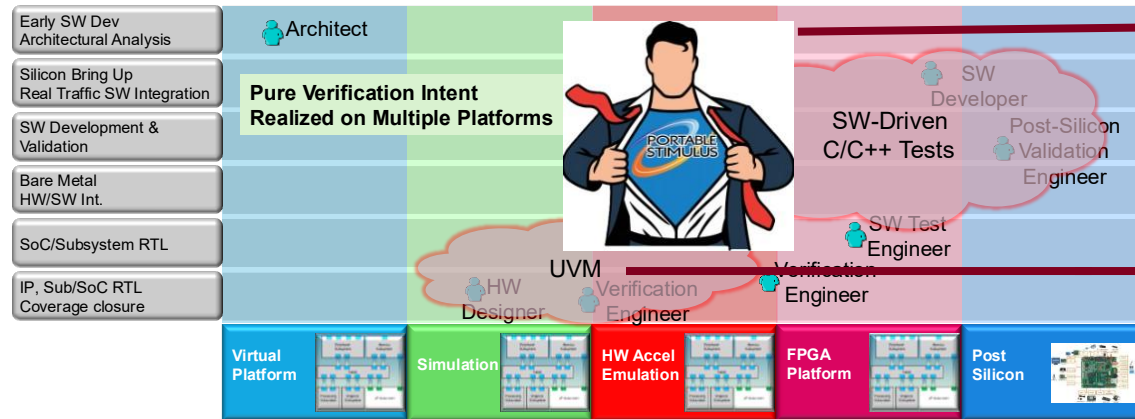
 Session 16

... The PSS View of a Modern SoC Design:

From Vision to Deployment and Production

Typical PSS environment:

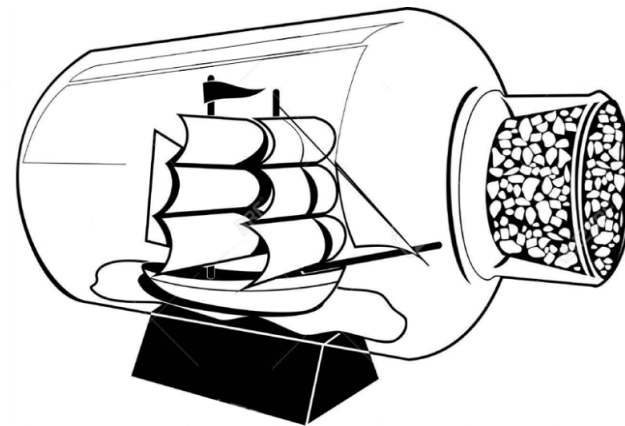
- Hierarchy of abstract models – SoC, SubSys, IP
- Residing on top of Test Realization Layer



::: Portable Stimulus is Layered

The
Abstract
Model

- *What* does it do?

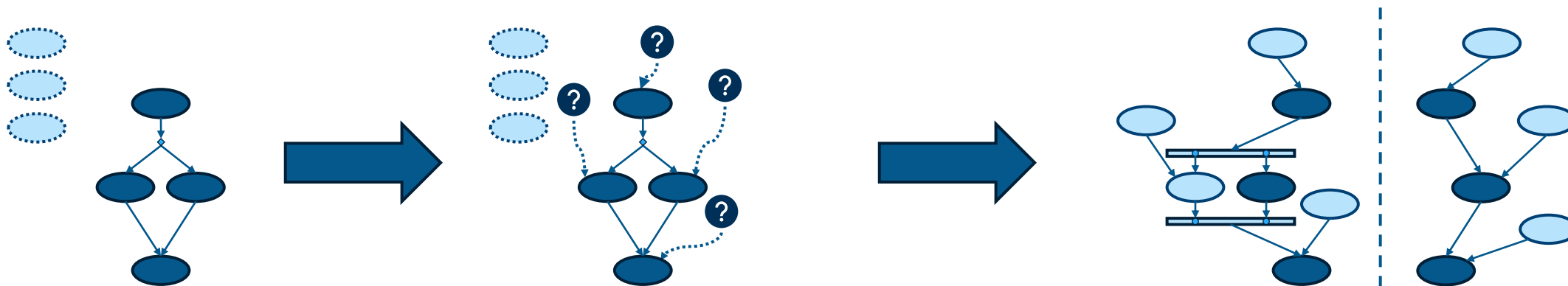
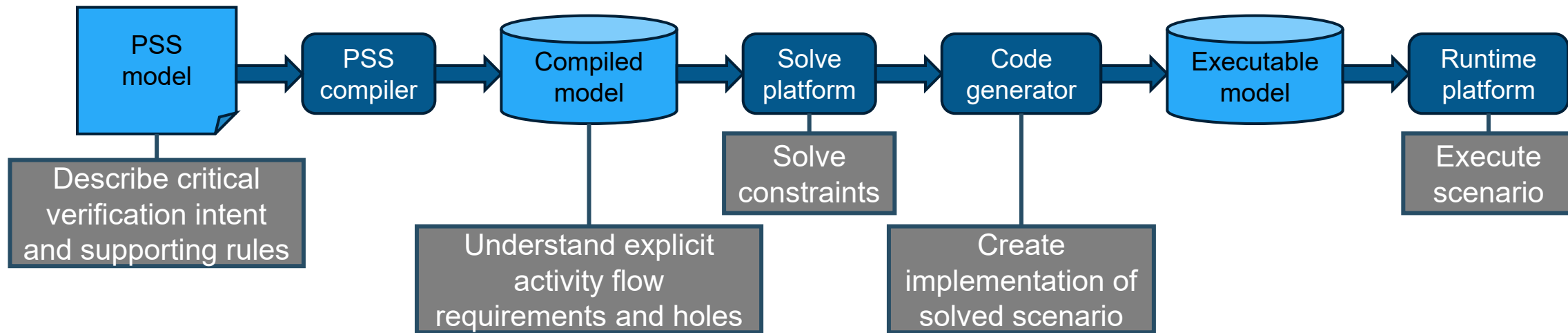


The
Realization
Layer

- *How* does it do what it does?



⋮ PSS Models are Declarative



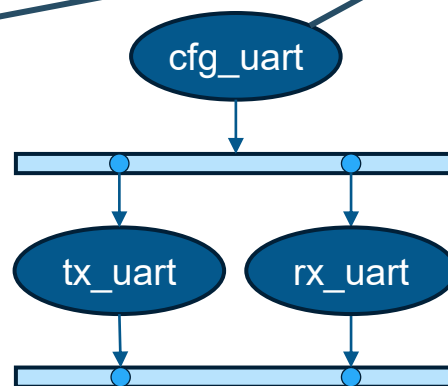
::: Actions and Activities

- ▶ The activity defines the schedule of sub-actions to model critical intent

```
activity_declaration ::= activity { { activity_stmt } }  
activity_sequence_block_stmt ::= [ sequence ] { { activity_stmt } }  
activity_parallel_stmt ::= parallel { { activity_stmt } }
```

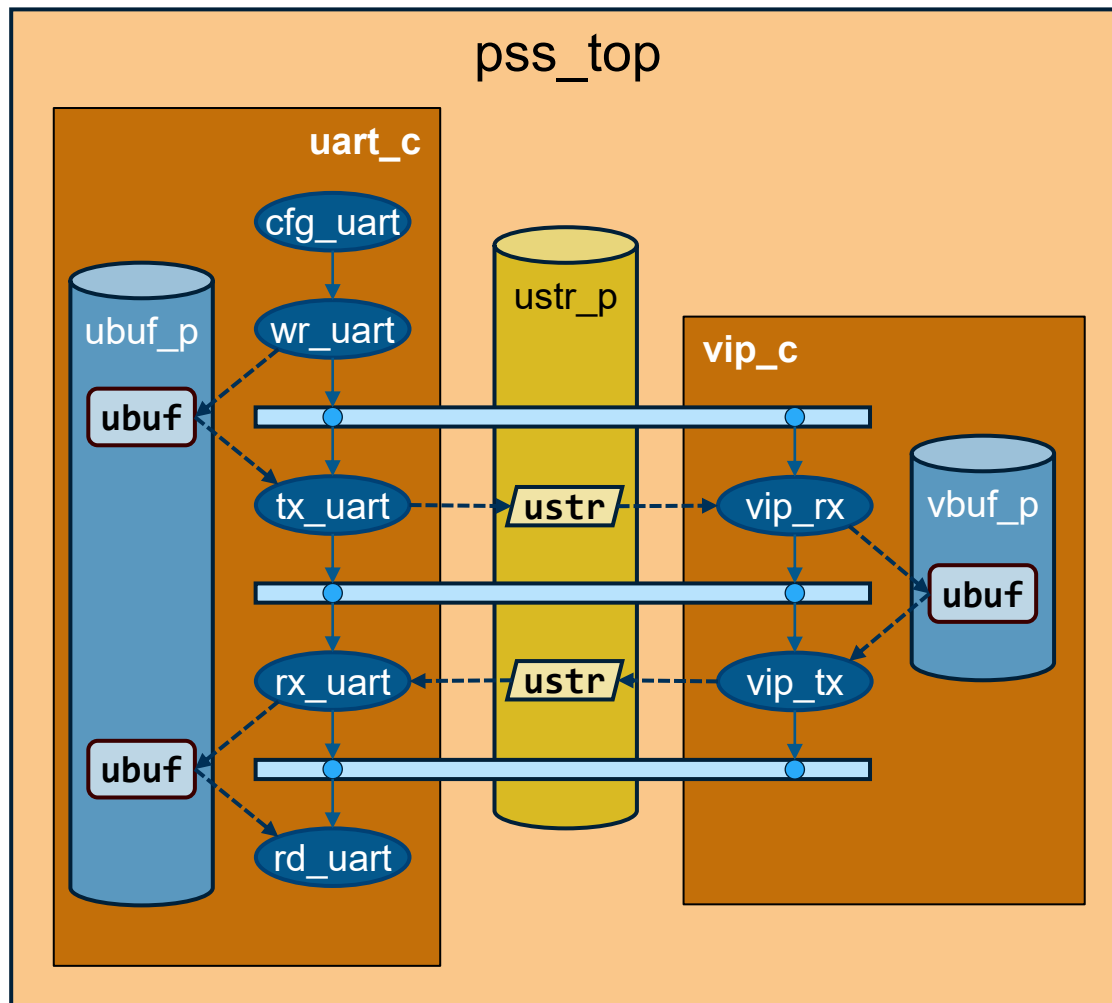
```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;  
  rx_uart_a rx;  
}
```

```
activity {  
  sequence {  
    cfg;  
    parallel {  
      tx;  
      rx;  
    }  
  }  
}
```



An *action traversal* statement defines the point where the action will be randomized and evaluated

Data Flow Object Binding & Pools



```
component uart_c {
  action cfg_uart_a {...}
  action wr_uart_a {output ubuf_b obuf;}
  action tx_uart_a {input ubuf_b ibuf;
                    output ustr_s ostr;}

  action rd_uart_a {input ubuf_b ibuf;}
  action rx_uart_a {output ubuf_b obuf;
                    input ustr_s istr;}
}
```

```
pool ubuf_b ubuf_p;
bind ubuf_p *;
```

```
component vip_c {
  action vip_rx_a {output ubuf_b obuf;
                  input ustr_s istr;}
  action vip_tx_a {input ubuf_b ibuf;
                  output ustr_s ostr;}
}
```

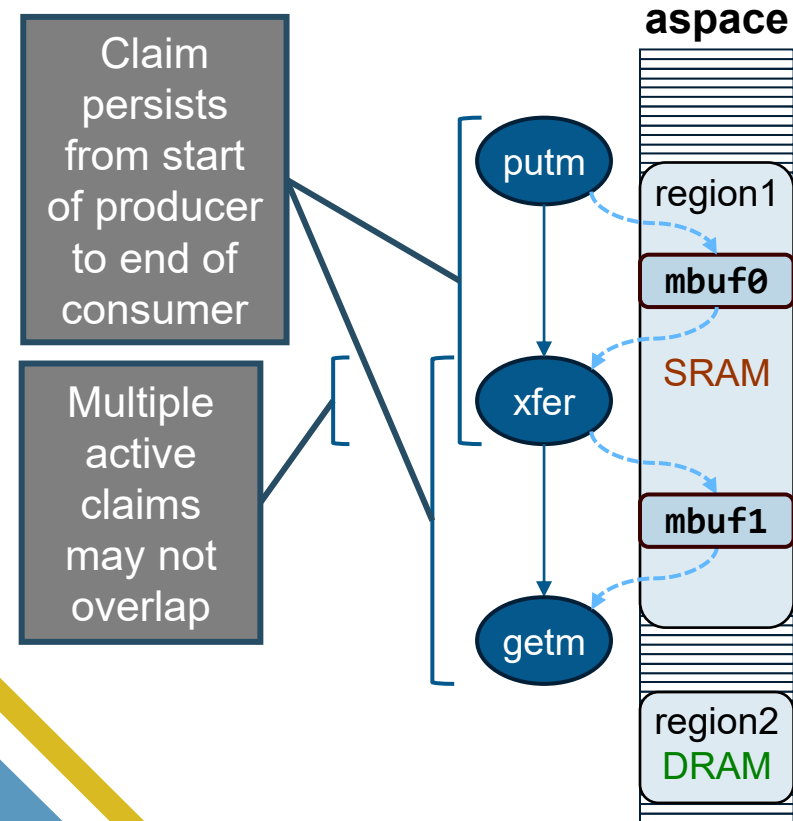
```
pool ubuf_b vbuf_p;
bind vbuf_p *;
```

```
component pss_top {
  uart_c uart;
  vip_c vip;
}
```

```
pool ustr_s ustr_p;
bind ustr_p *;
```

Address Space Claims

- ▶ An address *claim* struct has a *trait* field that matches the corresponding region
- ▶ Claims can be made in **actions** or **flow/resource** objects



```
component pss_top {  
  import addr_reg_pkg::*;  
  import my_addr_pkg::*;  
  pool mbuf_b mbuf_p;  
  bind mbuf_p *;
```

```
  action entry_a {  
    mem_c::getm_a getm;  
    mem_c::put_a putm;  
    dma_c::xfer_a xfer;
```

```
    activity {  
      putm;  
      xfer;  
      getm;
```

```
  }  
}
```

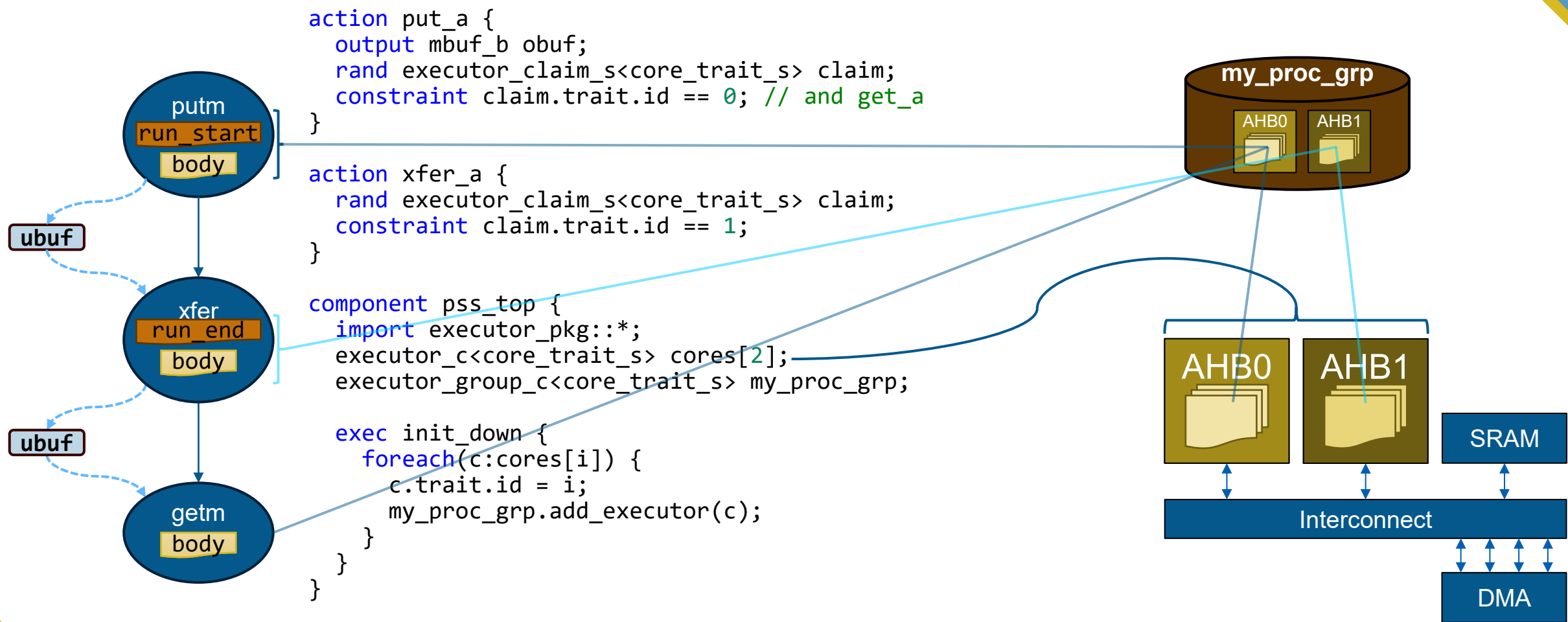
```
  action xfer_a {  
    input mbuf_b ibuf;  
    output mbuf_b obuf;  
    constraint io_c {  
      ibuf.size == obuf.size;  
      ibuf.parity == obuf.parity;  
      obuf.step == ibuf.step + 1;
```

```
    }  
    constraint { obuf.mClaim.trait.kind == SRAM;  
                 ibuf.mClaim.trait.kind == SRAM;
```

```
  }  
}
```

xfer_a claims only match
regions with kind==SRAM

Claiming an Executor



Example

- ▶ Register references converted to address handles based on calculated offsets
- ▶ Register accesses converted to built-in primitive access API calls

```
action xfer_a {  
  input mbuf_b ibuf;  
  output mbuf_b obuf;  
  lock chan_r chan;  
  
  exec body {  
    src_hndl = make_handle_from_claim(ibuf.mClaim);  
    dst_hndl = make_handle_from_claim(obuf.mClaim);  
    comp.dma_regs.chan_regs[chan.instance_id].size_reg.write(ibuf.size);  
    comp.dma_regs.chan_regs[chan.instance_id].src_reg.write(addr_value(src_hndl));  
    comp.dma_regs.chan_regs[chan.instance_id].dst_reg.write(addr_value(dst_hndl));  
    comp.dma_regs.chan_regs[chan.instance_id].ctrl_reg.write_field("go_done", 0x1);  
    while (comp.dma_regs.chan_regs[chan.instance_id].ctrl_reg.read().go_done == 1) {  
      message(NONE, "Waiting for DMA channel %d to complete", chan.instance_id);  
    }  
  }  
}
```

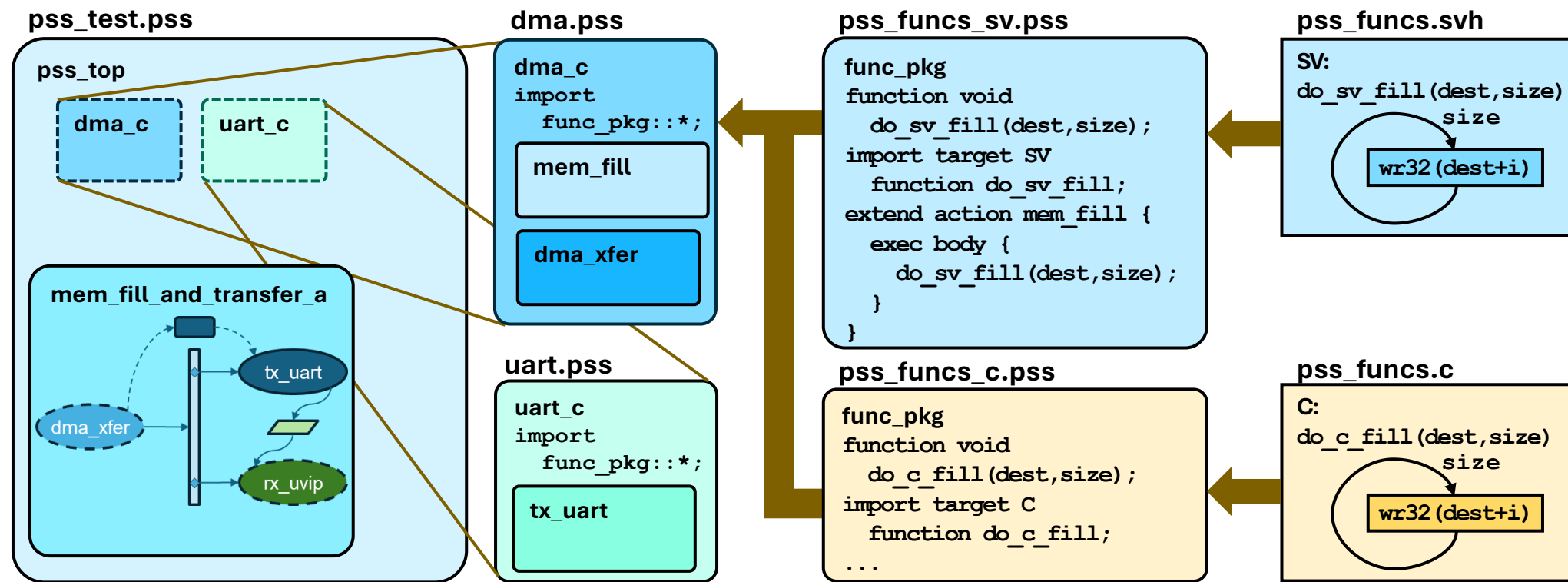
`comp.dma_regs.chan_regs[chan.instance_id].size_reg.write(ibuf.size);`

`wr_h = make_handle_from_handle(regs_h, offset);`
`write32(wr_h, ibuf.size);`

aspace	
regs	
ctrl_reg	3C
size_reg	38
dst_reg	34
src_reg	30
ctrl_reg	2C
size_reg	28
dst_reg	24
src_reg	20
ctrl_reg	1C
size_reg	18
dst_reg	14
src_reg	10
ctrl_reg	0C
size_reg	08
dst_reg	04
src_reg	00

Can be mapped to the appropriate executor

PSS Provides Multiple Realizations of a Model





Thank You



SYSTEMS INITIATIVE™



tom@bigfisheda.com