2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
VIRTUAL | MARCH 1-4, 2021

Configuration Conundrum: Managing Test Configuration with a Bite Sized Solution

Kevin Vasconcellos, Jeff McNeal

Verilab, Inc.

accellera SYSTEMS INITIATIVE™

2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
VIRTUAL | MARCH 1-4, 2021

Configuration Conundrum

accellera SYSTEMS INITIATIVE™

Current Solutions

- Call randomize() with {} from the test -- need to expose the randomization call to each test
- Helper variables -- works, can add complexity to support controls (solve before, more complex constraints)
- Child classes with pre-packaged settings
- Use the Policy Pattern to package individual constraint blocks, then the user can select the ones they want

Wouldn't it be nice?

Example: Car Variables

Configurations

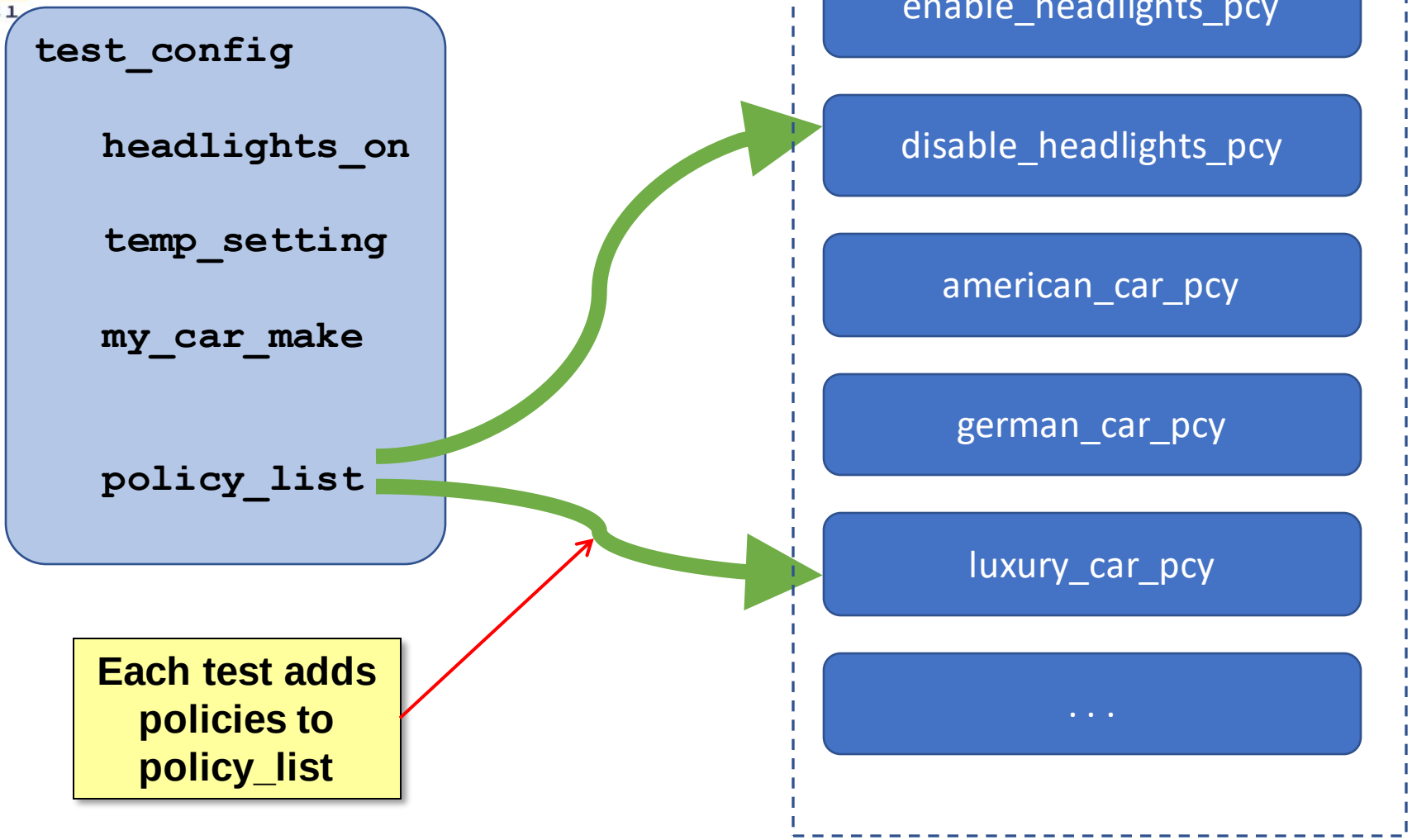
- Make: Ford, Chevy, VW, Tesla, Rover, Lexus, Honda, Audi, etc.
- Transmission: manual, automatic
- Two wheel drive vs 4wd
- Number of seats: 2 – 9
- 2 doors or 4 doors
- etc.

Controls

- Headlights on/off
- Wipers on/off
- Radio station: jazz, talk, rock, rap, classical, oldies, country
- etc.

Example: Selected Policies

```
class my_test extends base_test;
  . . .
  // Policies used in this test
  env.test_cfg.policy_list.add(env.test_cfg.manual);
  env.test_cfg.policy_list.add(env.test_cfg.american);
  env.test_cfg.policy_list.add(env.test_cfg.four_door);
  env.test_cfg.policy_list.rm(env.test_cfg.german); // don't select
                                                    // German cars
endclass : my_test
```


test_config
headlights_on
temp_setting
my_car_make
policy_list
policy library
enable_headlights_pcy
disable_headlights_pcy
american_car_pcy
german_car_pcy
luxury_car_pcy
. . .
Each test adds policies to policy_list

Policy Pattern

Instead of having all the constraints in the configuration object, each constraint is broken out into a policy.

enable policy code

```
class enable_headlights_pcy extends test_cfg_policy_base;
  function new(string name = "enable_headlights_pcy"); <...>

  constraint enable_headlights_c { item.headlights_on == 1; }
endclass
```

disable policy code

```
class disable_headlights_pcy extends test_cfg_policy_base;
  function new(string name = " disable_headlights_pcy"); <...>

  constraint disable_headlights_c { item.headlights_on == 0; }
endclass
```

Start with a common base class

```
class policy_base #(type ITEM=uvm_object) extends uvm_object;
  rand ITEM item;

  function new(string name = "policy base"); <...>

  virtual function void set_item(ITEM item);
    this.item = item;
  endfunction : set_item
endclass : policy_base
```

Create a Common List Class

```
class policy_list #(type ITEM=uvm_object) extends policy_base #(ITEM);
  rand policy_base #(ITEM) plist[$];

  function new(string name = "policy list"); <...>

  function void set_item(ITEM item);
    foreach(plist[i]) plist[i].set_item(item);
  endfunction : set_item

endclass : policy_list
```

Helpful Typedefs

Typedef the specializations of the two common classes

```
typedef class test_config;
typedef policy_base#(test_config) test_cfg_policy_base;
typedef policy_list#(test_config) test_cfg_policy_list;
```

First Policies

```
class enable_headlights_pcy extends test_cfg_policy_base;
  function new(string name = "enable_headlights_pcy"); <...>

  constraint enable_headlights_c { item.headlights_on == 1; }
endclass
```

```
class disable_headlights_pcy extends test_cfg_policy_base;
  function new(string name = "disable_headlights_pcy"); <...>

  constraint disable_headlights_c { item.headlights_on == 0; }
endclass
```

Some example policy classes for the test_cfg headlights_on member

More Policies

Example of a more complex constraint

```
class enable_encryption_policy extends test_cfg_policy_base;
  <...>
  constraint enable_encryption_c {
    item.encrypt == 1;
    (item.selected_speed == GEN1) -> (item.encrypt_type1 == 1);
    (item.selected_speed == GEN2) -> (item.encrypt_type2 == 1);
  }
endclass
```

Test Configuration Class

```
class test_config extends uvm_object;
  rand car_make_t      my_car_make;
  rand bit             headlights_on;
  rand int             temp_setting;
  <...>
```

Rand member variables aka config knobs

Looking into test_cfg

Test Configuration Class

```
class test_config extends uvm_object;
  rand car_make_t      my_car_make;
  rand bit             headlights_on;
  rand int             temp_setting;
  <...>
  rand test_cfg_policy_list policy_list;
  string plist;
```

Our policy list and a helper string

Looking into test_cfg

Test Configuration Class

```
class test_config extends uvm_object;
  rand car_make_t      my_car_make;
  rand bit             headlights_on;
  rand int             temp_setting;
  <...>
  string plist;
  rand test_cfg_policy_list policy_list;

  enable_headlights_pcy  lights_on;
  disable_headlights_pcy lights_off;
  automatic_trans_pcy    automatic_trans;
  manual_trans_pcy       manual_trans;
  american_car_pcy       american;
  german_car_pcy         german;
  luxury_car_pcy         luxury;
  japanese_car_pcy       japanese;
  midwest_mix_pcy        midwest;
```

One of each of the defined policy classes

Looking into test_cfg

Test Configuration Class

```
function new(string name = "");
   super.new(name);
   // Create the queue to hold policies
   policy_list = new();
```

Test Configuration Class

```
function new(string name = "");
   super.new(name);
   // Create the queue to hold policies
   policy_list = new();

   // Create an instance of each policy class
   lights_on    = new("lights_on");
   lights_off   = new("lights_off");
   automatic_trans = new("automatic_trans");
   manual_trans = new("manual_trans");
   american     = new("american");
   german       = new("german");
   japanese     = new("japanese");
   luxury       = new("luxury");
   midwest      = new("midwest");
```

Test Configuration Class

```
    // build a string to print the policy names to the log
    plist = $sformatf("%0s %0s\n", plist, lights_on.get_name());
    plist = $sformatf("%0s %0s\n", plist, lights_off.get_name());
    plist = $sformatf("%0s %0s\n", plist, automatic_trans.get_name());
    plist = $sformatf("%0s %0s\n", plist, manual_trans.get_name());
    plist = $sformatf("%0s %0s\n", plist, american.get_name());
    plist = $sformatf("%0s %0s\n", plist, german.get_name());
    plist = $sformatf("%0s %0s\n", plist, japanese.get_name());
    plist = $sformatf("%0s %0s\n", plist, luxury.get_name());
    plist = $sformatf("%0s %0s\n", plist, midwest.get_name());
  endfunction : new
```

Test Configuration Class

```
function void pre_randomize();
  policy_list.set_item(this);
  `uvm_info(get_type_name(),
            $sformatf("policy_list contains %0d policies",
                      policy_list.plist.size()), UVM_LOW)
endfunction : pre_randomize
```

Test Configuration Class

```
function void pre_randomize();
  policy_list.set_item(this);
  `uvm_info(get_type_name(),
            $sformatf("policy_list contains %0d policies",
                      policy_list.plist.size()), UVM_LOW)
endfunction : pre_randomize

function void post_randomize();
  policy_list.print_policy_list();
endfunction : post_randomize
```

Testcases

- Once we have policies, the test writer can choose which policies to use
- Enable or disable certain features
- Features w/no policy chosen are randomized
- Interactions and ordering are handled automatically 'under the hood'

Test code with policies selected

```
class my_test extends base_test;
  . . .
  // Policies used in this test
  env.test_cfg.policy_list.add(env.test_cfg.manual);
  env.test_cfg.policy_list.add(env.test_cfg.american);
  env.test_cfg.policy_list.add(env.test_cfg.four_door);
 // don't select German cars
  env.test_cfg.policy_list.rm(env.test_cfg.german);
endclass : my_test
```

Mixing Policies

```
constraint american_cars_only_c {
  item.my_car_make inside {
        chevy,
        ford,
        cadillac,
        tesla
  };
}
```

```
 constraint midwest_mix_c {
    item.my_car_make dist {
        chevy := 30,
        ford  := 30,
        cadillac := 10,
        tesla := 5,
        honda := 5,
        vw    := 5,
        lexus := 1
    };
}
```

Mixing Policies

```
constraint american_cars_only_c {
  item.my_car_make inside {
        chevy,
        ford,
        cadillac,
        tesla
  };
}
```

```
constraint midwest_mix_c {
   item.my_car_make dist {
        chevy := 30,
        ford  := 30,
        cadillac := 10,
        tesla := 5,
        honda := 5,
        vw    := 5,
        lexus := 1
   };
}
```

```
constraint repair_daily_c {
  item.my_car_make != ford;
}
```

Swapping Policies

```
constraint american_cars_only_c {
  item.my_car_make inside {
        chevy,
        ford,
        cadillac,
        tesla
  };
}
```

```
constraint northwest_mix_c {
   item.my_car_make dist {
        chevy   := 15,
        ford    := 15,
        cadillac := 1,
        tesla   := 10,
        subaru  := 20,
        honda   := 10,
        toyota  := 15,
        lexus   :=5
   };
```

```
constraint repair_daily_c {
  item.my_car_make != ford;
}
```

Summary: Benefits

- Simplifies constraints
  - Reduce helper variables, reduce conditional constraints
- Allows constraint removal
  - Remove for some tests and not others
- Simplifies adding constraints
  - New policies only affect tests that use them
- Reduces disruptions to team

Summary: Addressing Concerns

- Constraint ordering issues
  - Since many complex constraints are much simpler, actually fewer issues
  - Constraints are solved as a single mass, so order added doesn't matter
- Debugging failures
  - Much simpler, most tools point to file & class that have issues, so you end up with:
    - enable_headlights_pcy conflicts with disable_headlights_pcy error messages
- Adding policies to existing systems
  - Add infrastructure
  - Convert a few constraints to start with

2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES VIRTUAL | MARCH 1-4, 2021

Conclusions & Takeaways

Implementing policies for testbench configuration:

- Provides a simple configuration control mechanism
- Reduces the learning curve for new test writers
- Encapsulates the configuration behavior and details
- Allow some configuration settings to override other settings

☑ Policies for constraints can be used on ***any*** randomizable object/class

accellera SYSTEMS INITIATIVE™

2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
VIRTUAL | MARCH 1-4, 2021

Questions

- Example code can be found here:
  - http://www.github.com/jmcneal/config_policy_pattern
- Contact the Authors
  - Kevin Vasconcellos, kevin.vasconcellos@verilab.com
  - Jeff McNeal, jeff.mcneal@verilab.com

accellera SYSTEMS INITIATIVE™

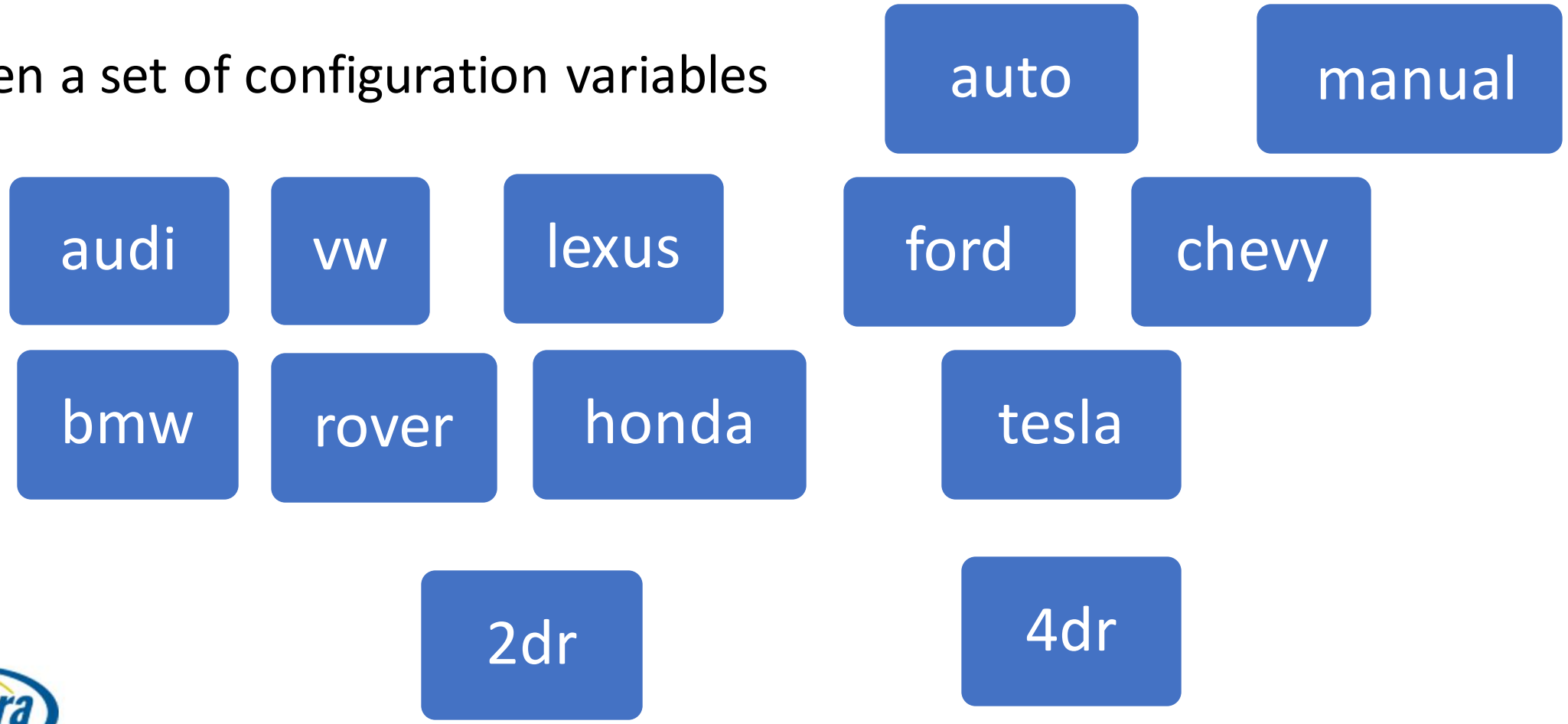
Conclusions & Takeaways

- Enable at-a-glance configurations in test
- Simplify the test writer interface
- Prevent redundant code (copy & paste)

☑ Policies for constraints can be used on ***any*** randomizable object/class

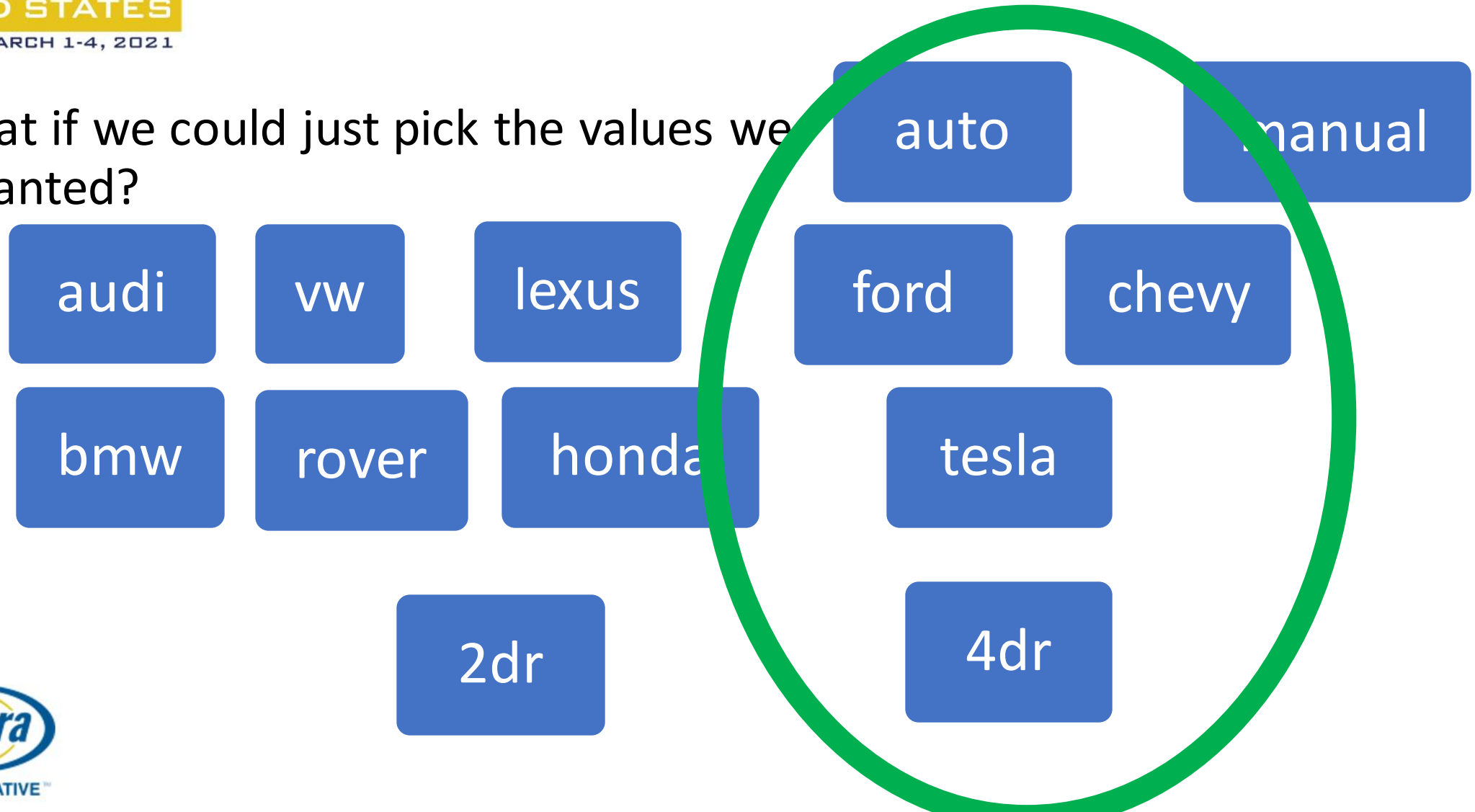
Policy Advantages

Given a set of configuration variables



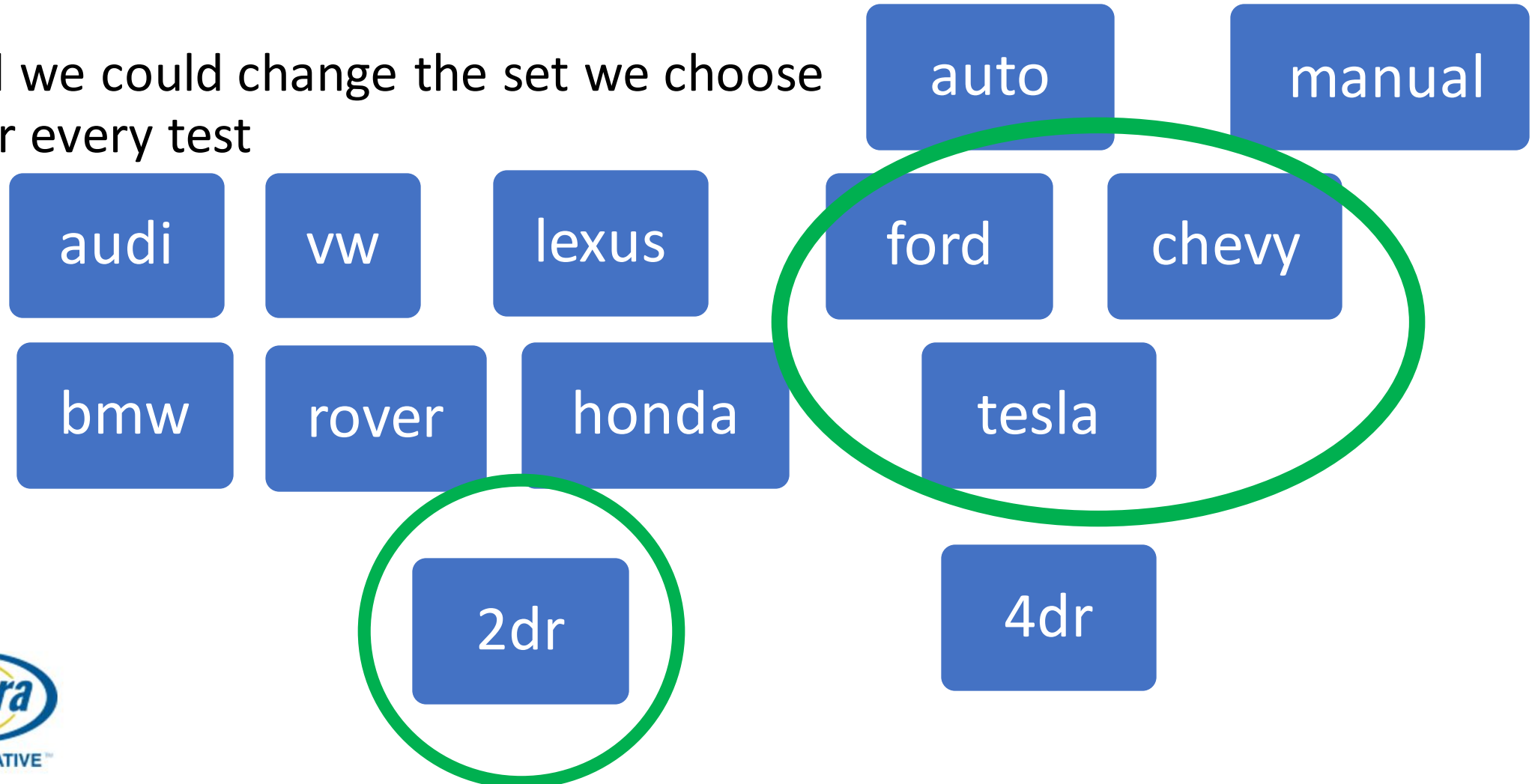
Choosing Policies

What if we could just pick the values we wanted?



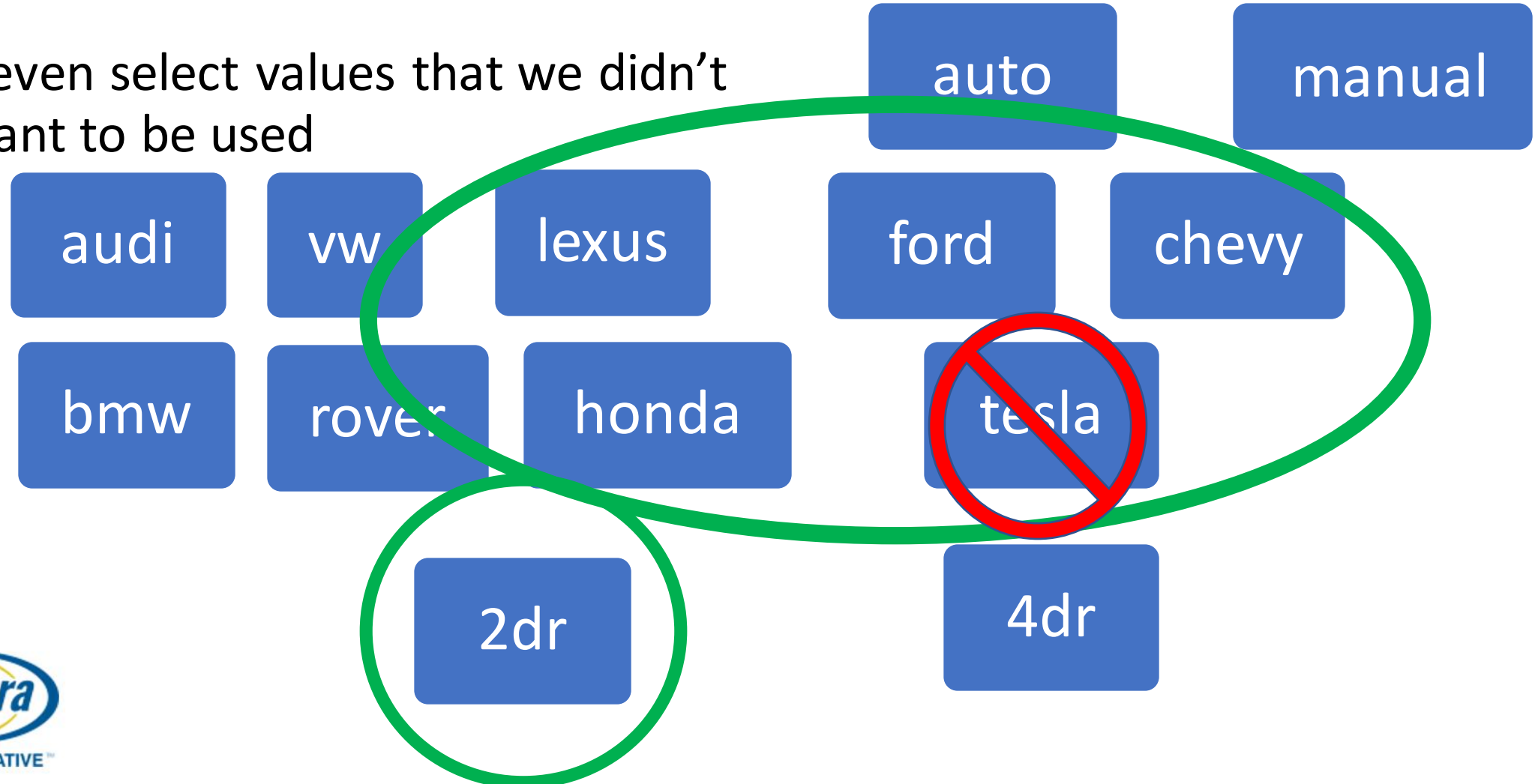
Choosing Policies

And we could change the set we choose for every test



Removing a constraint

Or even select values that we didn't want to be used



Why policies?

- Ease the job of the test writer –
  - no need to know how to write constraints
  - Remove concerns about constraint interaction
  - Provide at-a-glance configuration information in the test
  - Enable quick test auditing
  - Allow test writer to easily add *or remove* constraints as required
  - Allow for odd constraint combinations that weren't thought of at design time

Why policies?

- Ease the job of the testbench writer
  - Removes helper variables
  - Reduce pre-design overhead (don't have to have a complete map of constraint possibilities at architecture phase)
    - Enhances orthogonality of constraints
  - Allows for simpler constraints

Policy Pattern

Choosing among a set

- Allows for several independent policies to work together
  - Each policy can be trivially simple
  - Many different kinds of policies can work together
- User gets to mix & match
  - Can be very useful to temporarily select constraints while debugging
  - While bug hunting on a corner case

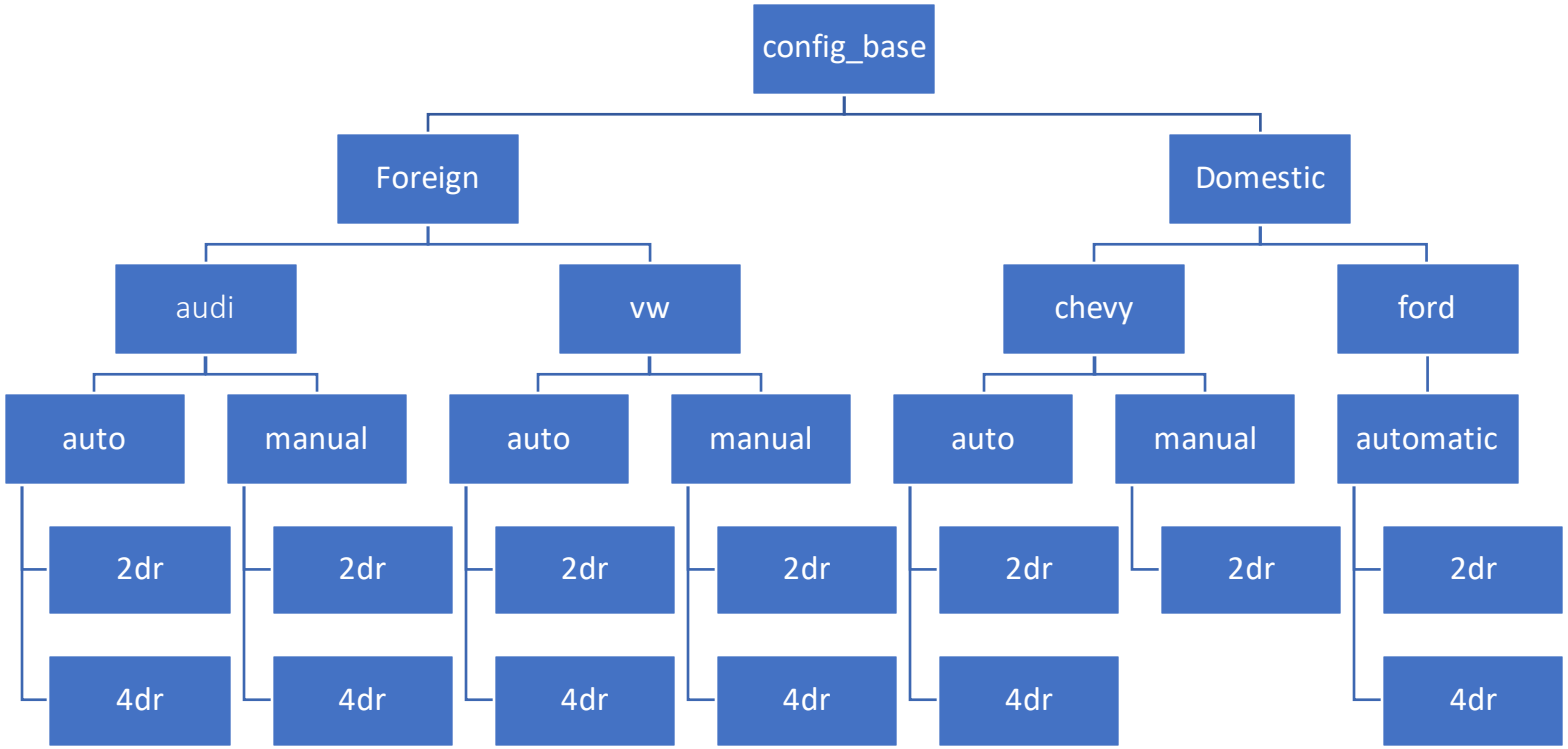
Configuration Example

```
config_base
    manufacturer:   [audi, vw, chevy,
    ford, tesla, honda, etc]
    transmission:   [auto, manual]
    num_door:    [2, 4]
    num_seats:   [1 - 9]
    wipers:  [on, int1, int2, off]
    headlights: [on, off]
    …
```

A configuration object (class) to configure the DUT and testbench for a non-trivial system.

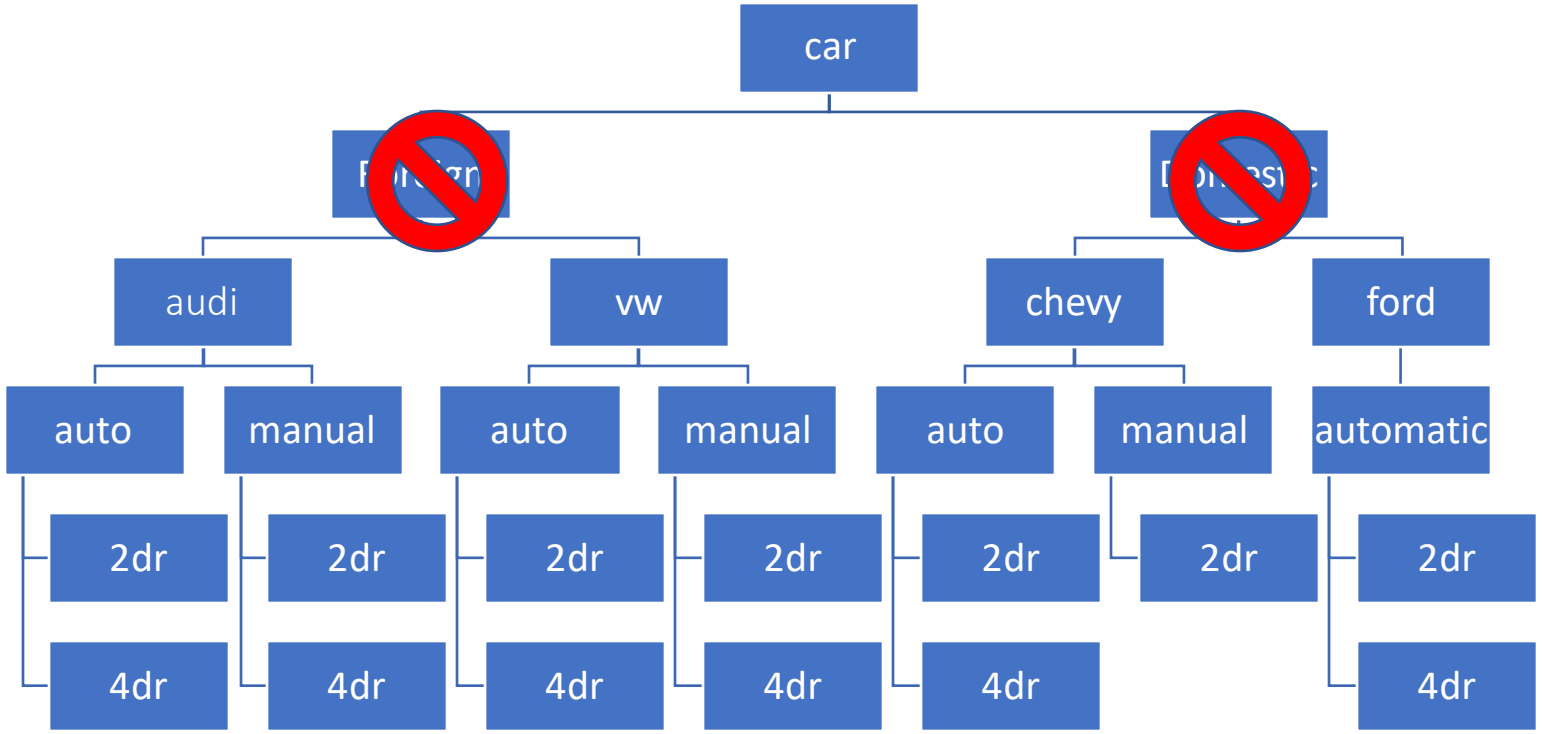
- There may be lots of individual options
- Many of them are interdependent

Configuration example



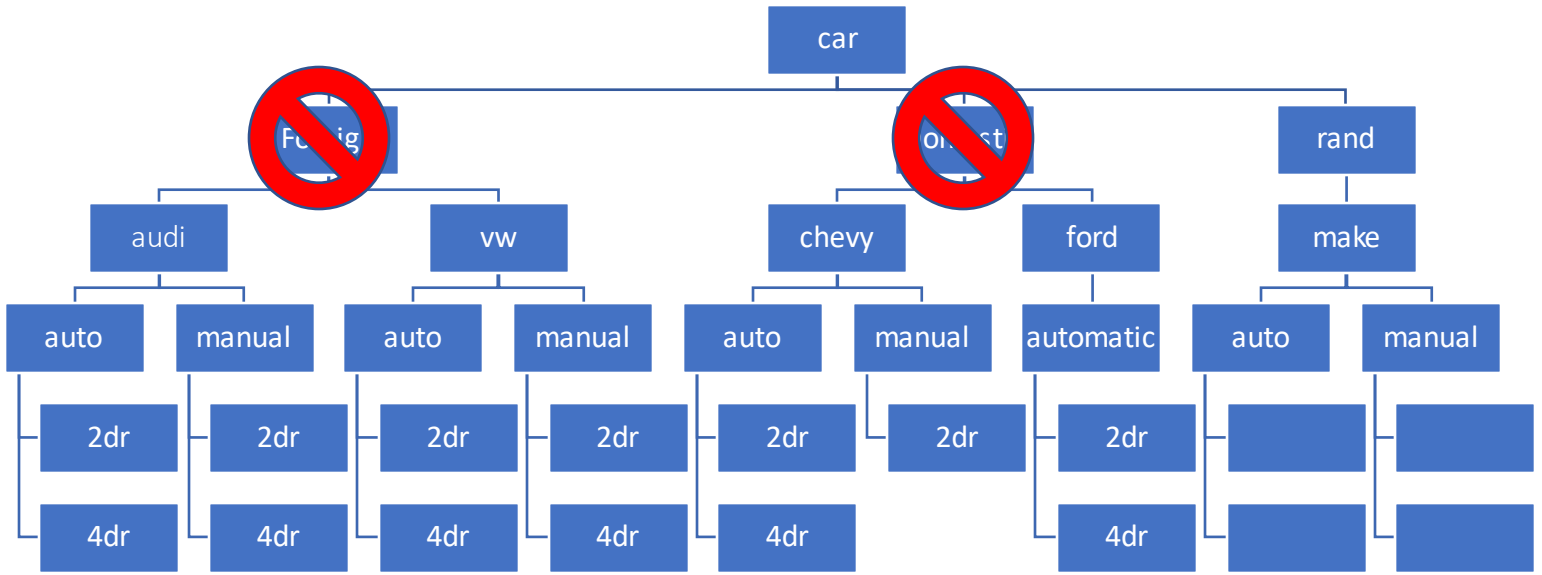
What if you want to remove a constraint?

- Easy if you want to remove one of the leaf constraints
- Very complex if things need to be changed later to remove what you thought was a base constraint
  - What if I want to randomize foreign v domestic
  - What if I want the root of the tree to be 2dr v 4 dr
- Even worse, add new major category, like electric car



What if you want to remove a constraint?

- May need to add another child class that allows for randomization of that variable
- May need to resort to helper variables



2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES VIRTUAL | MARCH 1-4, 2021

DVCon Slide Guidelines

- Use Arial or Helvetica font for slide text
- Use Courier-new or Courier font for code
- First-order bullets should be 24 to 28 point
  - Second-order bullets should be 24 to 26 point
    - Third-order bullets should be 22 to 24 point
    - Code should be at least 18 point
- Your presentation will be shown in a very large room
  - These font guidelines will help ensure everyone can read you slides!

~~No Company Logo except on title slide!~~

accellera SYSTEMS INITIATIVE™

Code and Notes

Code should be enclosed in text boxes (using a background color is optional)

Code should be 18pt Courier-bold, or larger

```
module example
(input  logic foo,
 output logic bar
);

  initial begin
     $display ("Hello World!");

endmodule
```

Informational boxes should be 18pt Arial-bold, or larger (using a background color is optional)