Random SMC Testing Strategy

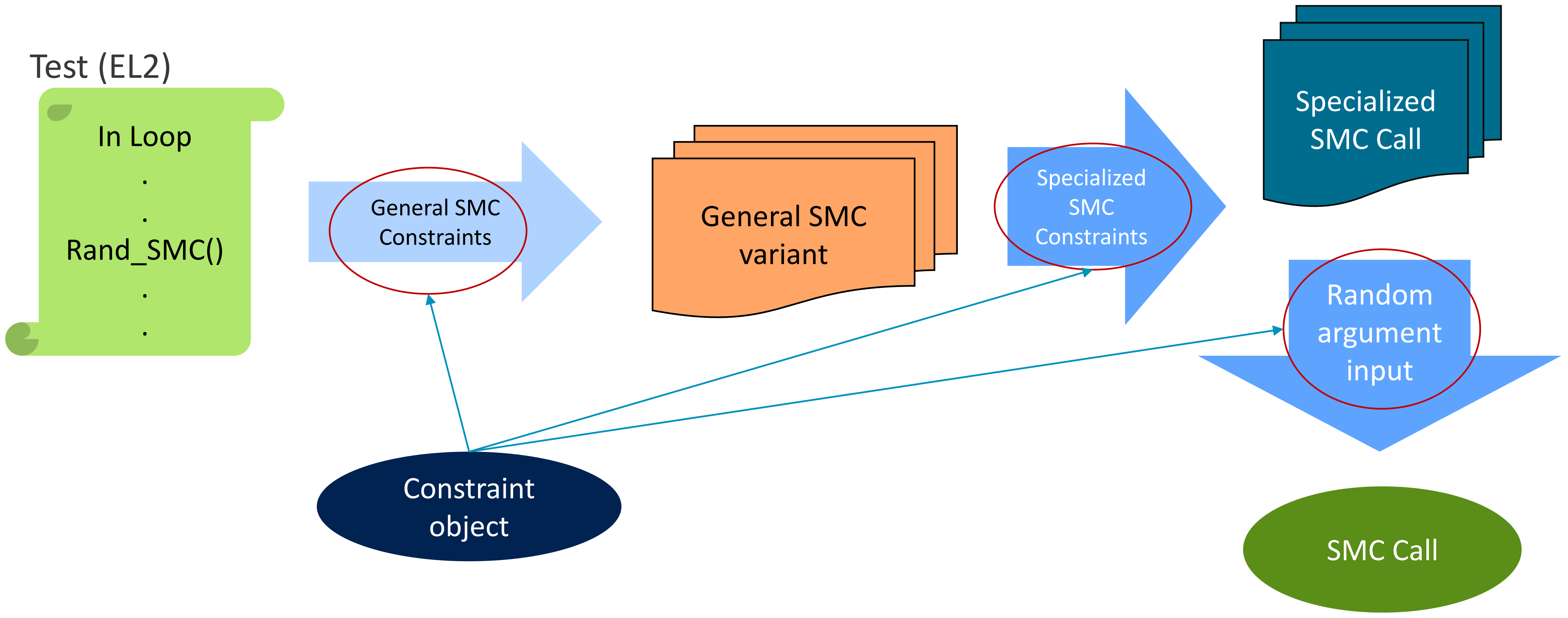
- Mark Dykes

Justification for directed randomized approach:

Behavior of tests changes per seed therefore…

- Can exercise many more code paths
- Better ability to find bugs/coverage points with much less effort than directed tests
- Increase in effectiveness with time -> issues can be found when time is increased to run more operations
- Better at finding issues with large state space variables
- Events and functions can be reordered in many different ways
- Potentially reduce test set since each test can be more comprehensive
- Lends to rule based test coding rather than strict code control of program flow

Approach to SMC calls(preliminary):

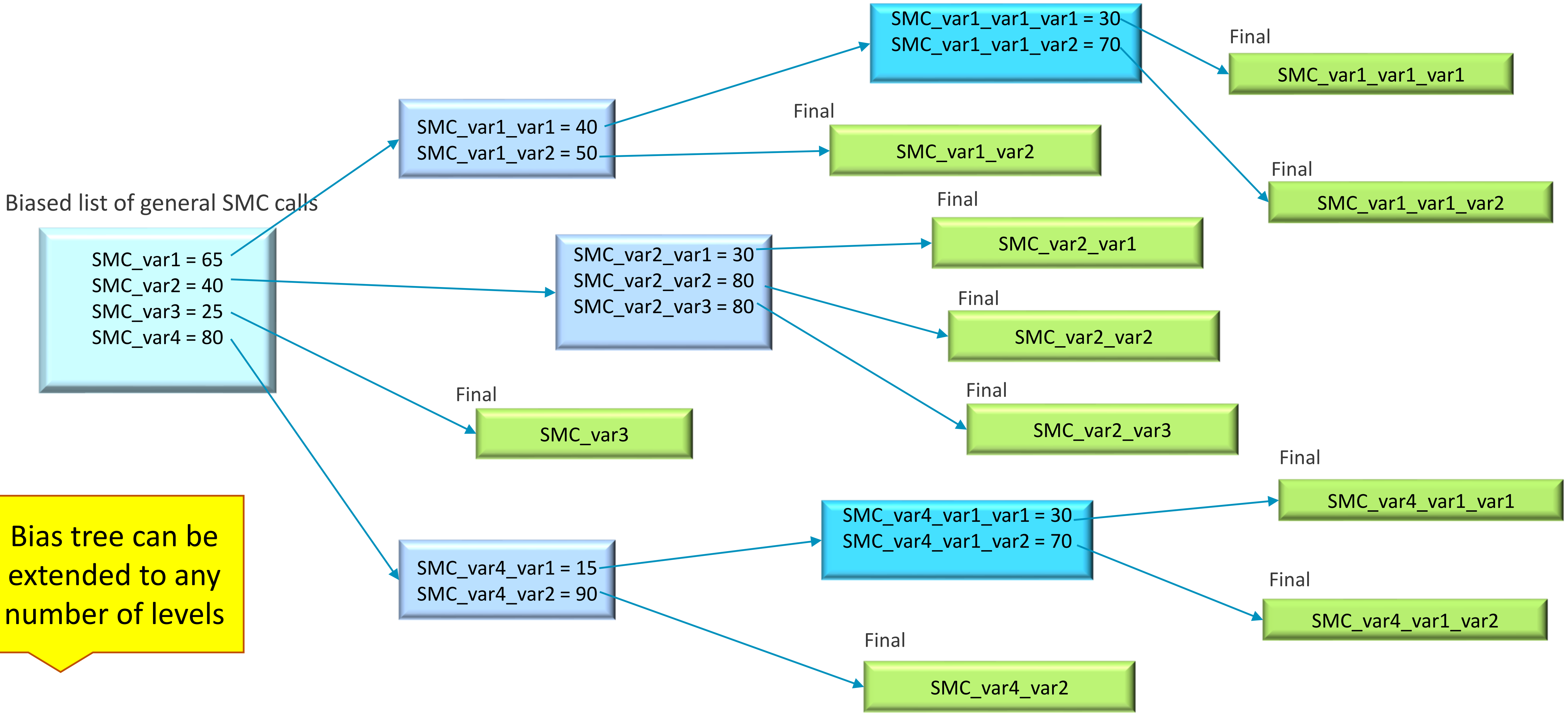


Constraint properties

The constraint will:

- Allow a user to tailor the testing to particular area of interest
- Have multiple layers of biasing among the options
- Override default behavior of randomization
- Can reduce test to be of a directed type
- Allows change of constraints per invocation of the random SMC
- Be of object type to be manipulated as any data object would

Bias tree model of SMC selection:



Bias tree can be extended to any number of levels

arm

Programming Constraints

The random SMC call will accept a single constraint argument that will contain all of the biases needed

```
Rand_SMC(constraint_obj)
```

The constraint object will have access methods to modify any of the biases across all hierarchies

```
Constraint_obj.SMC_var1 = 50
Constraint_obj.SMC_var1_var2 = 10
```

The default bias of all types will be set to 50 for an unmodified constraint object(when instantiated)

If no argument is given to Rand_SMC then default bias(50) for all types will be used

Functions can be provided that will set the biases in the constraint object for interesting Combinations or even random biases…

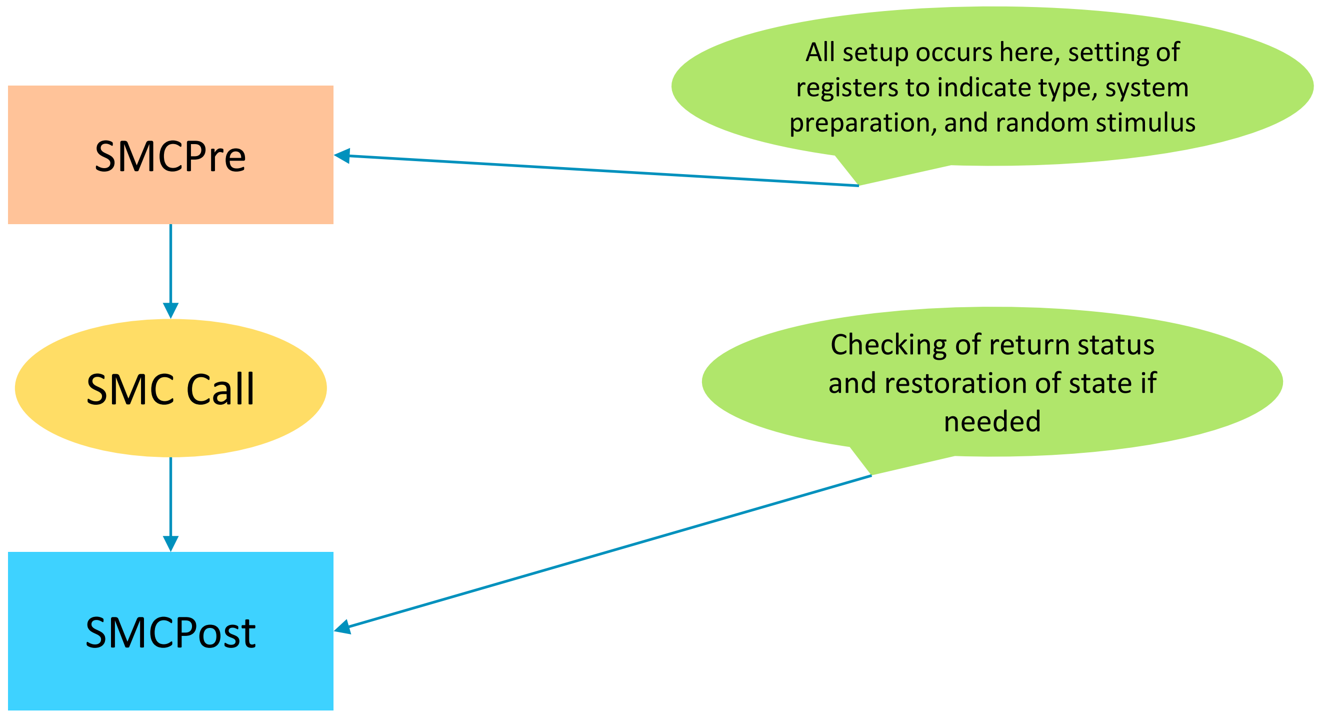
```
Constraint_obj = Set_constraints(SMC_type)
Constraint_obj = Set_constraints(Random)
```

Alternately, a library of objects could be provided for many interesting combinations

For the purposes of SDL and security hardening another bias type could be employed to cover special SMC calls that execute with types that could expose vulnerabilities not seen with other variations

Structure of Random SMC call

All calls would adopt the same basic format that would be based upon a template version that would loosely follow the directed test instance

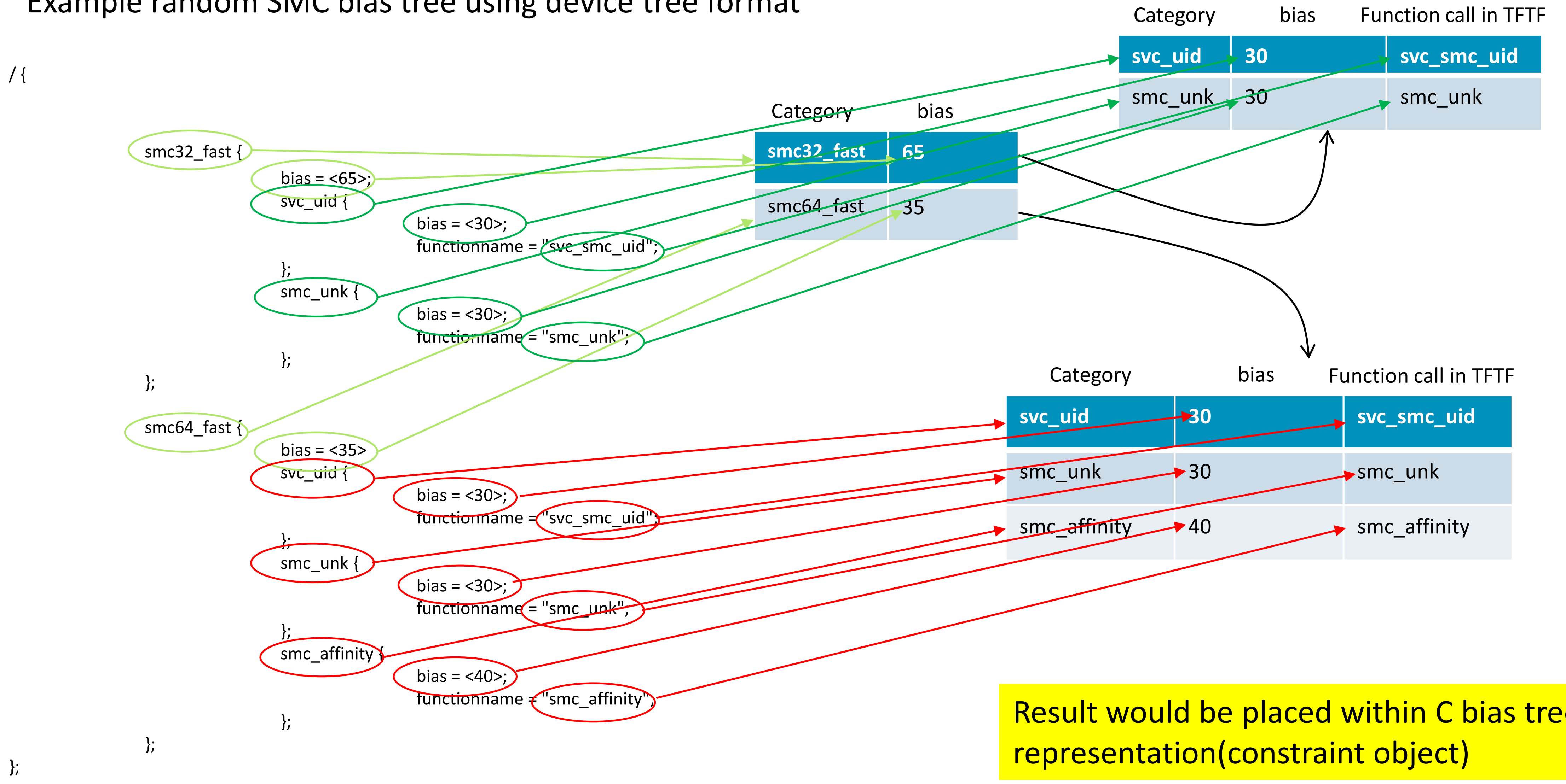

SMCPre
SMC Call
SMCPost
All setup occurs here, setting of registers to indicate type, system preparation, and random stimulus
Checking of return status and restoration of state if needed

Considerations for randomized setup

- All SMC invocations should exist as autonomously as possible
- The state of the system must remain the same between calls as much as possible
- Must have common error/warning handling protocol in place(will be addressed soon)
- Can work in parallel with directed tests without issue
- Will have to have methodology to address vendor specific types of testing

Implementation

Example random SMC bias tree using device tree format



```
/ {
        smc32_fast {
                bias = <65>;
                svc_uid {
                        bias = <30>;
                        functionname = "svc_smc_uid";
                };
                smc_unk {
                        bias = <30>;
                        functionname = "smc_unk";
                };
        };
        smc64_fast {
                bias = <35>
                svc_uid {
                        bias = <30>;
                        functionname = "svc_smc_uid";
                };
                smc_unk {
                        bias = <30>;
                        functionname = "smc_unk";
                };
                smc_affinity {
                        bias = <40>;
                        functionname = "smc_affinity";
                };
        };
};
```

Category	bias
smc32_fast	65
smc64_fast	35

Category	bias	Function call in TFTF
svc_uid	30	svc_smc_uid
smc_unk	30	smc_unk

Category	bias	Function call in TFTF
svc_uid	30	svc_smc_uid
smc_unk	30	smc_unk
smc_affinity	40	smc_affinity

Result would be placed within C bias tree representation(constraint object)

Status:

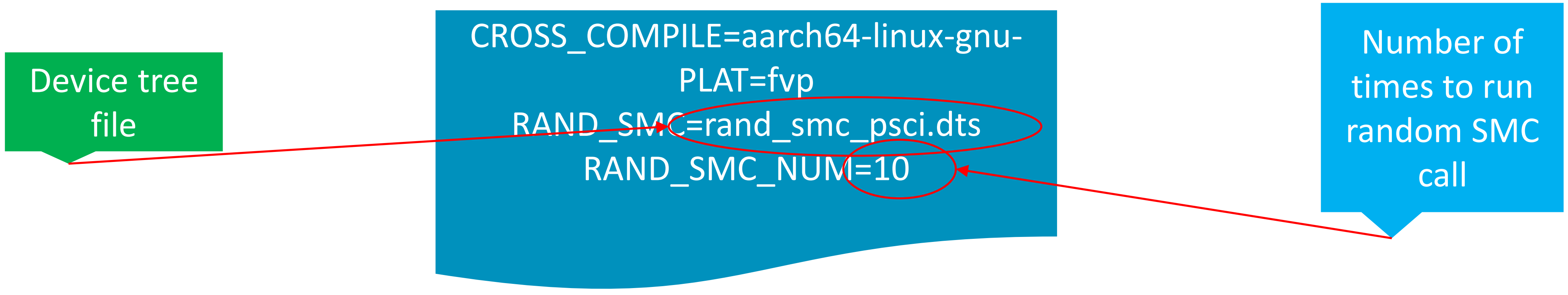
- The prototype to read and create the tree is complete in C code
- Had meeting with Joanna and Sandrine to best determine integration strategy into CI
- The device tree binary will be integrated into the FIP
- Challenge will be to ensure that memory limits will not be exceeded by larger trees
- Focus will be initially to randomize the flavor of SMC rather than inherent arguments
- Begin with one CPU?
- Currently exploring options for malloc function in TFTF. Have created malloc prototype

that is around 70% complete

Proposal for CI integration:

Usage model:

The enablement and control of feature would be specified in a given TFTF config similar to how current testing categories are conducted. A separate directory under platform-ci would contain all device trees for the purpose of SMC fuzzing. An example config file might contain:



```
CROSS_COMPILE=aarch64-linux-gnu-
PLAT=fvp
RAND_SMC=rand_smc_psci.dts
RAND_SMC_NUM=10
```

Key challenges:

- No provision for C memory management
- Must designate special memory area for the bias tree
- Create a TFTF_malloc?
- Extensive work necessary to support fuzzing
- Allow directed tests to run before/after random SMC phase? Or mutually exclusive?
- Could be 2-3 sprints worth of effort to get initial CI runs in place