2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

DatagenDV: Python Constrained Random Test Stimulus Framework

Jon George
James Mackenzie
Microsoft

accellera SYSTEMS INITIATIVE

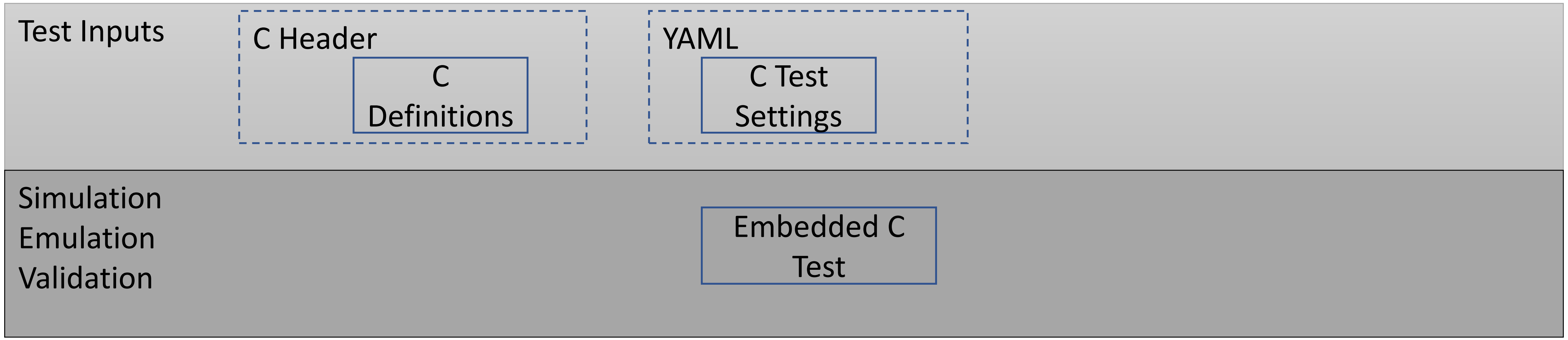
Agenda

- Infrastructure Background
- Features of DatagenDV
- Python Libraries Leveraged
- Examples of DatagenDV
- Lessons learned

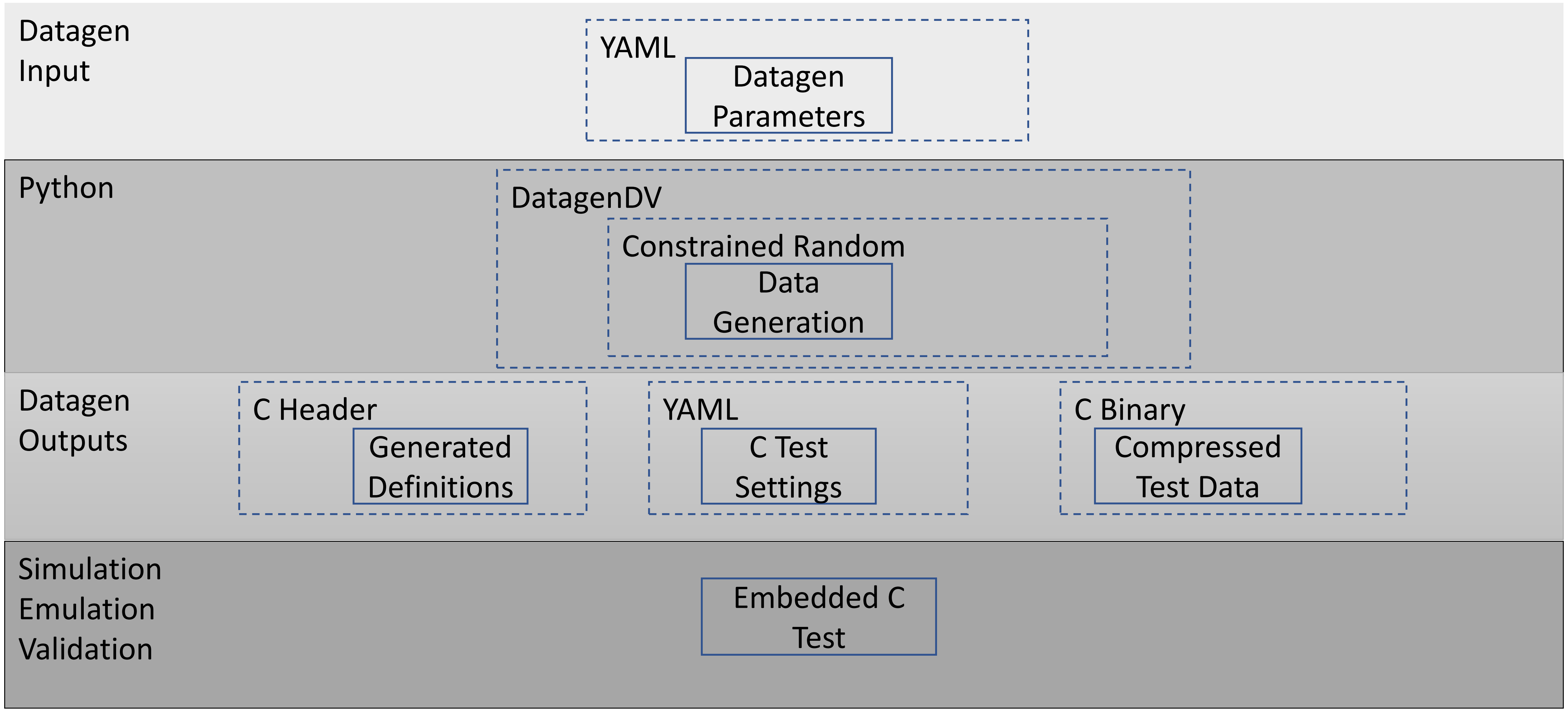
Existing Infrastructure

Limitations:

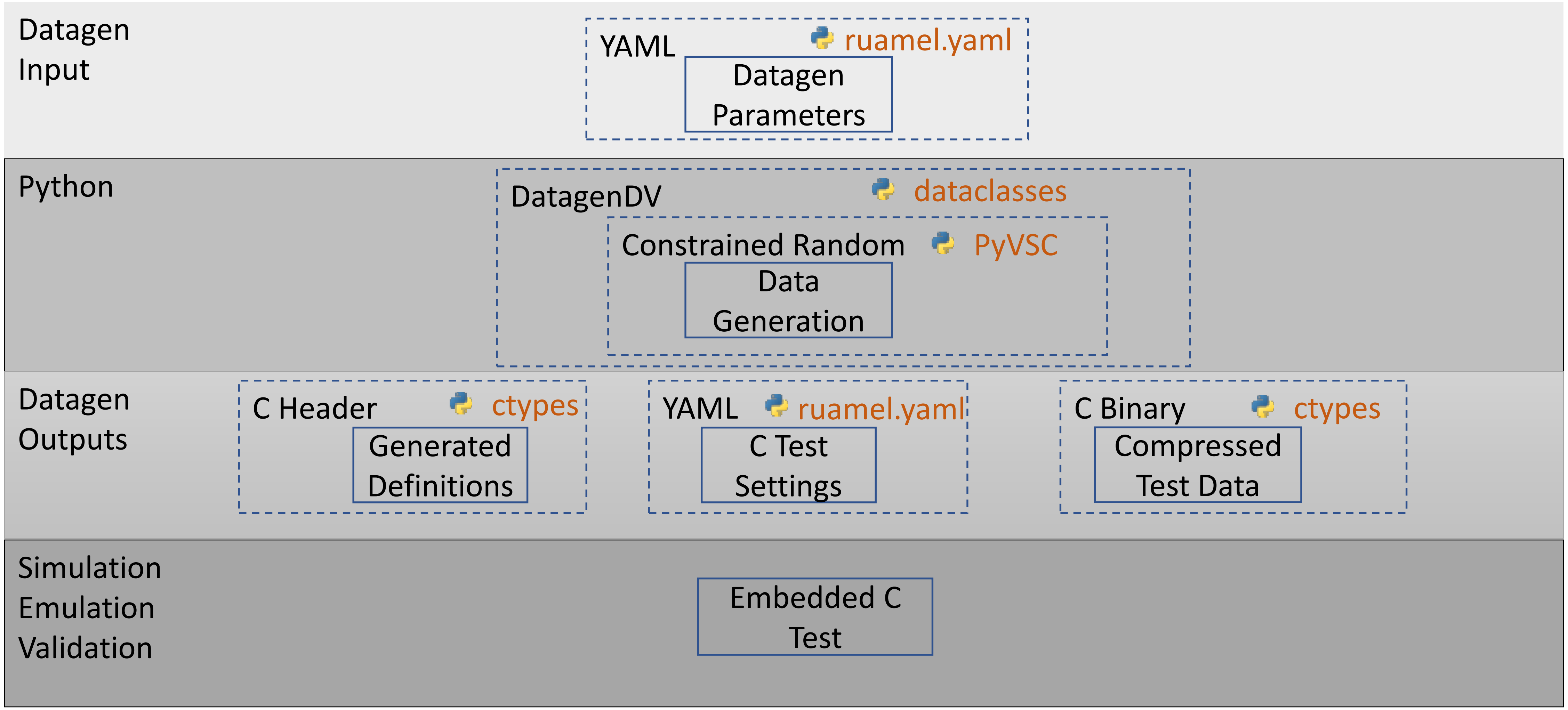
- C Stimulus generation during sim is slow
- YAML stimulus generation is limited



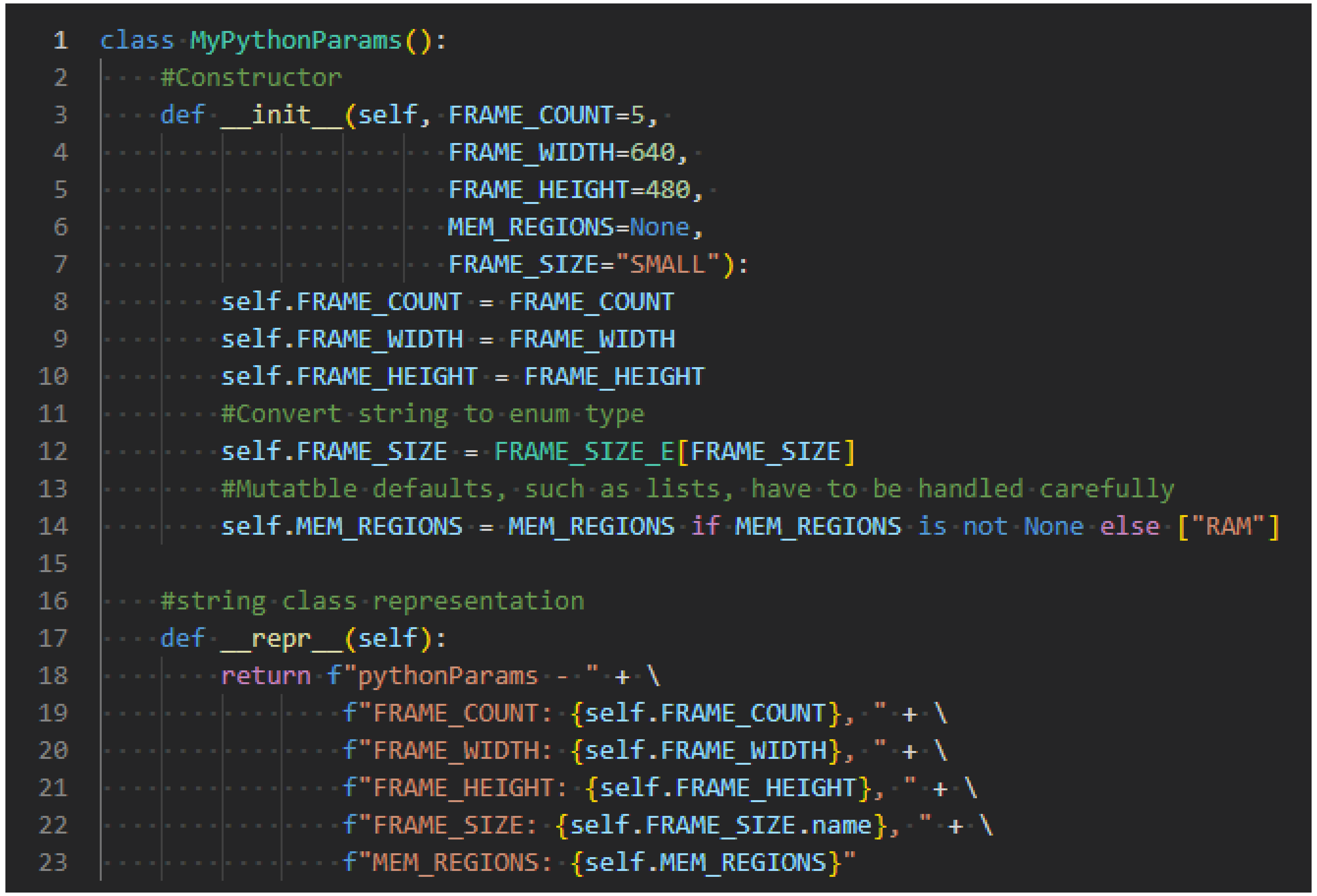
DatagenDV


Datagen Input
YAML
Datagen Parameters
Python
DatagenDV
Constrained Random
Data Generation
Datagen Outputs
C Header
Generated Definitions
YAML
C Test Settings
C Binary
Compressed Test Data
Simulation Emulation Validation
Embedded C Test

Python Libraries



The Different Syntax of Python Objects



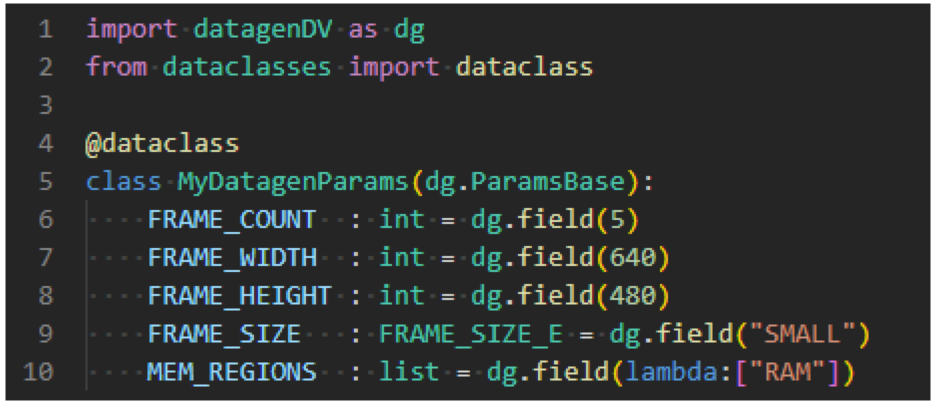
```python
class MyPythonParams():
    #Constructor
    def __init__(self, FRAME_COUNT=5,
                       FRAME_WIDTH=640,
                       FRAME_HEIGHT=480,
                       MEM_REGIONS=None,
                       FRAME_SIZE="SMALL"):
        self.FRAME_COUNT = FRAME_COUNT
        self.FRAME_WIDTH = FRAME_WIDTH
        self.FRAME_HEIGHT = FRAME_HEIGHT
        #Convert string to enum type
        self.FRAME_SIZE = FRAME_SIZE_E[FRAME_SIZE]
        #Mutatble defaults, such as lists, have to be handled carefully
        self.MEM_REGIONS = MEM_REGIONS if MEM_REGIONS is not None else ["RAM"]

    #string class representation
    def __repr__(self):
        return f"pythonParams - " + \
                f"FRAME_COUNT: {self.FRAME_COUNT}, " + \
                f"FRAME_WIDTH: {self.FRAME_WIDTH}, " + \
                f"FRAME_HEIGHT: {self.FRAME_HEIGHT}, " + \
                f"FRAME_SIZE: {self.FRAME_SIZE.name}, " + \
                f"MEM_REGIONS: {self.MEM_REGIONS}"
```

- ‘new()’ -> ‘__init__()’
- ‘print()’ -> ‘__repr__()’?
- ‘this’ -> ‘self’?
- Where are the fields?

“I’ll just use dictionaries instead”

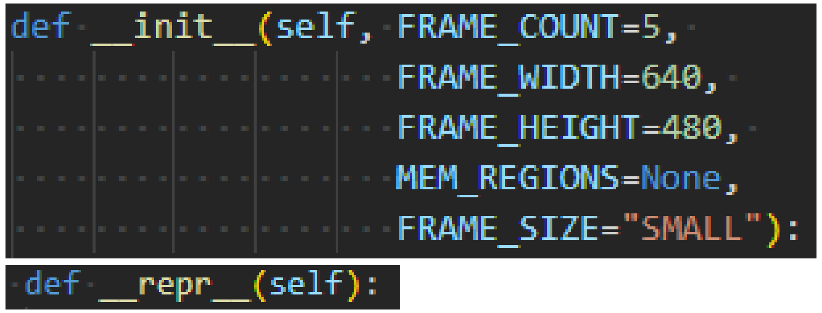
Dataclasses – UVM Object Macros for Python



```python
import datagenDV as dg
from dataclasses import dataclass

@dataclass
class MyDatagenParams(dg.ParamsBase):
    FRAME_COUNT  : int = dg.field(5)
    FRAME_WIDTH  : int = dg.field(640)
    FRAME_HEIGHT : int = dg.field(480)
    FRAME_SIZE   : FRAME_SIZE_E = dg.field("SMALL")
    MEM_REGIONS  : list = dg.field(lambda:["RAM"])
```

Generated by @dataclass



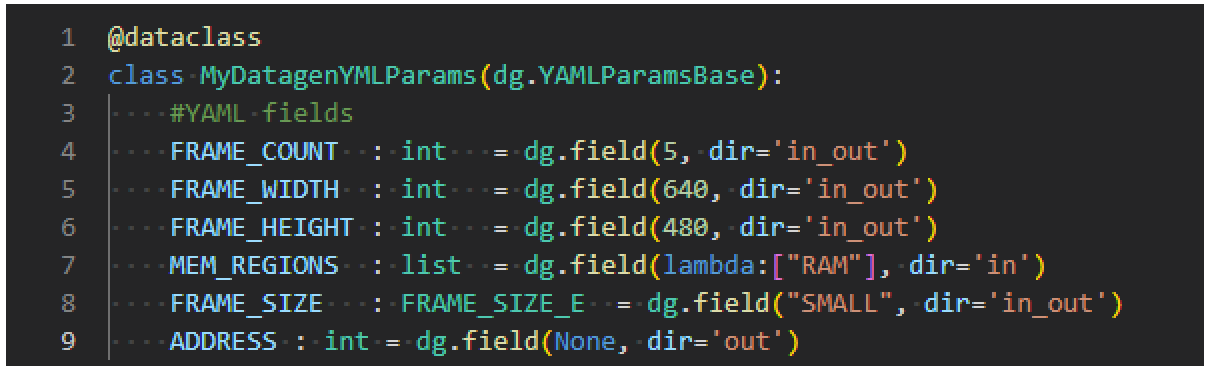
```python
def __init__(self, FRAME_COUNT=5,
                   FRAME_WIDTH=640,
                   FRAME_HEIGHT=480,
                   MEM_REGIONS=None,
                   FRAME_SIZE="SMALL"):
def __repr__(self):
```

Datagen Features

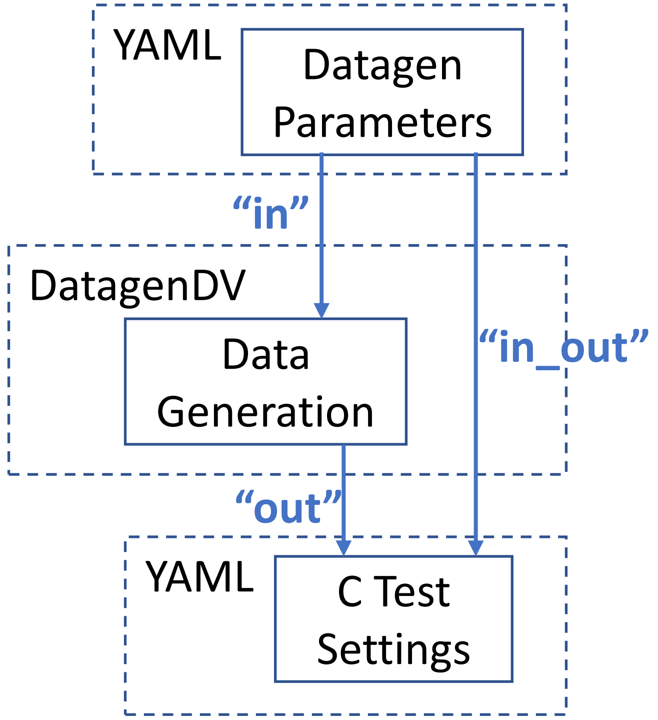
- Automatic type checking added to constructor
- Converts strings to enums
- Handles mutable default types

ruamel.yaml

YAML Class Loading and Dumping



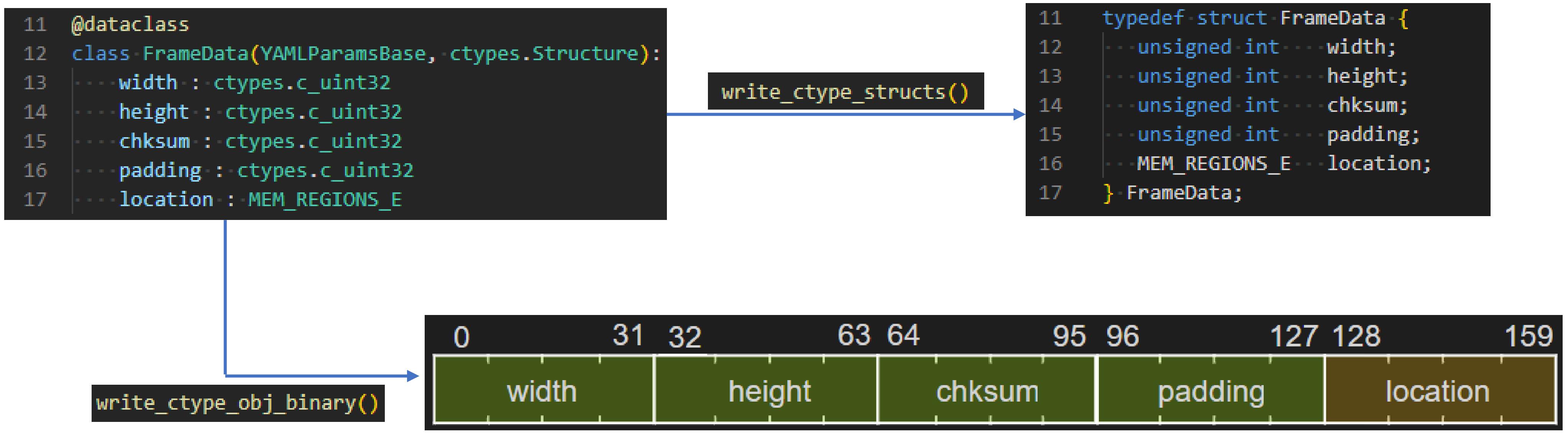
```python
@dataclass
class MyDatagenYMLParams(dg.YAMLParamsBase):
    #YAML fields
    FRAME_COUNT  : int   = dg.field(5, dir='in_out')
    FRAME_WIDTH  : int   = dg.field(640, dir='in_out')
    FRAME_HEIGHT : int   = dg.field(480, dir='in_out')
    MEM_REGIONS  : list  = dg.field(lambda:["RAM"], dir='in')
    FRAME_SIZE   : FRAME_SIZE_E  = dg.field("SMALL", dir='in_out')
    ADDRESS : int = dg.field(None, dir='out')
```



Datagen Features

- YAML loading/dumping directly into/from objects
- Fields loaded/dumped based on direction

ctypes – Bridging the Language Gap



```python
@dataclass
class FrameData(YAMLParamsBase, ctypes.Structure):
    width : ctypes.c_uint32
    height : ctypes.c_uint32
    chksum : ctypes.c_uint32
    padding : ctypes.c_uint32
    location : MEM_REGIONS_E
```

```c
typedef struct FrameData {
    unsigned int    width;
    unsigned int    height;
    unsigned int    chksum;
    unsigned int    padding;
    MEM_REGIONS_E   location;
} FrameData;
```

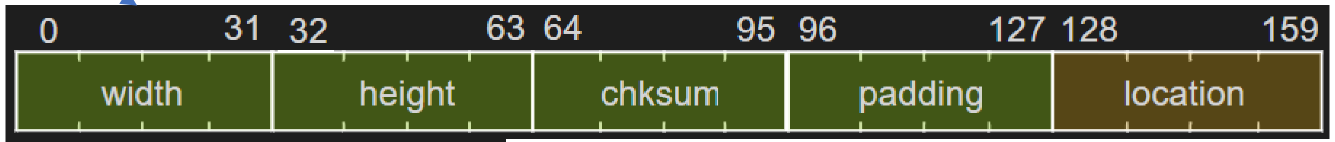
Datagen Features

- Generation of C headers (structs, enums, defines)
- Generation of binary data

ctypes – Loading Binary data into C

```
FrameData( width=0xABCD_E0F0, height=0x1234_5678, chksum=0x9876_5432,
           padding=0x5432_10FE, location=MEM_REGIONS_E.RAND)
```

```
typedef struct FrameData {
    unsigned int    width;
    unsigned int    height;
    unsigned int    chksum;
    unsigned int    padding;
    MEM_REGIONS_E   location;
} FrameData;
```



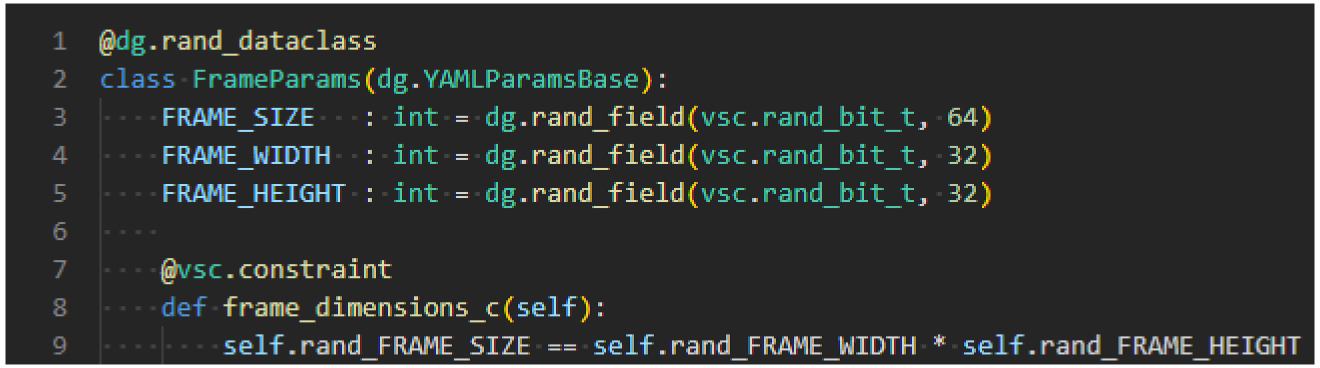
```
>hexdump -x framedata_output.bin
e0f0    abcd    5678    1234    5432    9876    10fe    5432    0006    0000
```

```
FILE* frame_fh = fopen("framedata_output.bin", "rb");
FrameData* myFrame = malloc(sizeof(FrameData ));
fread(myFrame, sizeof(FrameData), 1, frame_fh);
printf("width 0x%x height 0x%x chksum 0x%x padding 0x%x location 0x%x\\n",
  myFrame->width, myFrame->height, myFrame->chksum, myFrame->padding,  myFrame->location);
```

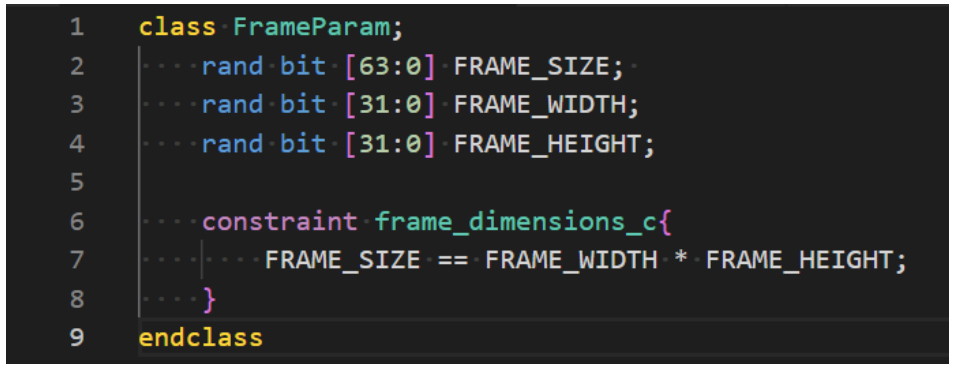
```
width 0xabcde0f0 height 0x12345678 chksum 0x98765432 padding 0x543210fe location 0x6
```

PyVSC - Constrained Randomization

PyVSC



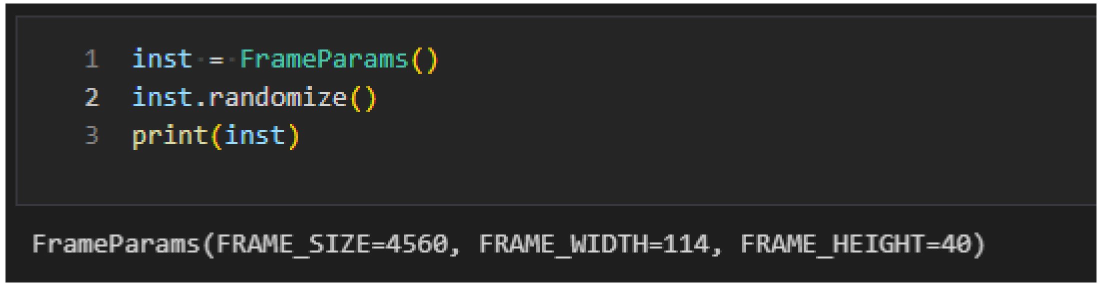
```python
@dg.rand_dataclass
class FrameParams(dg.YAMLParamsBase):
    FRAME_SIZE   : int = dg.rand_field(vsc.rand_bit_t, 64)
    FRAME_WIDTH  : int = dg.rand_field(vsc.rand_bit_t, 32)
    FRAME_HEIGHT : int = dg.rand_field(vsc.rand_bit_t, 32)
    
    @vsc.constraint
    def frame_dimensions_c(self):
        self.rand_FRAME_SIZE == self.rand_FRAME_WIDTH * self.rand_FRAME_HEIGHT
```



```
class FrameParam;
    rand bit [63:0] FRAME_SIZE;
    rand bit [31:0] FRAME_WIDTH;
    rand bit [31:0] FRAME_HEIGHT;

    constraint frame_dimensions_c{
        FRAME_SIZE == FRAME_WIDTH * FRAME_HEIGHT;
    }
endclass
```

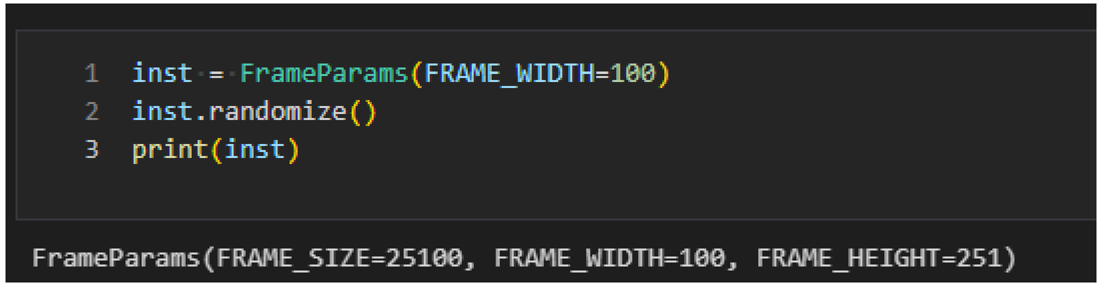
Full random



```python
inst = FrameParams()
inst.randomize()
print(inst)
```

```
FrameParams(FRAME_SIZE=4560, FRAME_WIDTH=114, FRAME_HEIGHT=40)
```

Override FRAME_WIDTH



```python
inst = FrameParams(FRAME_WIDTH=100)
inst.randomize()
print(inst)
```

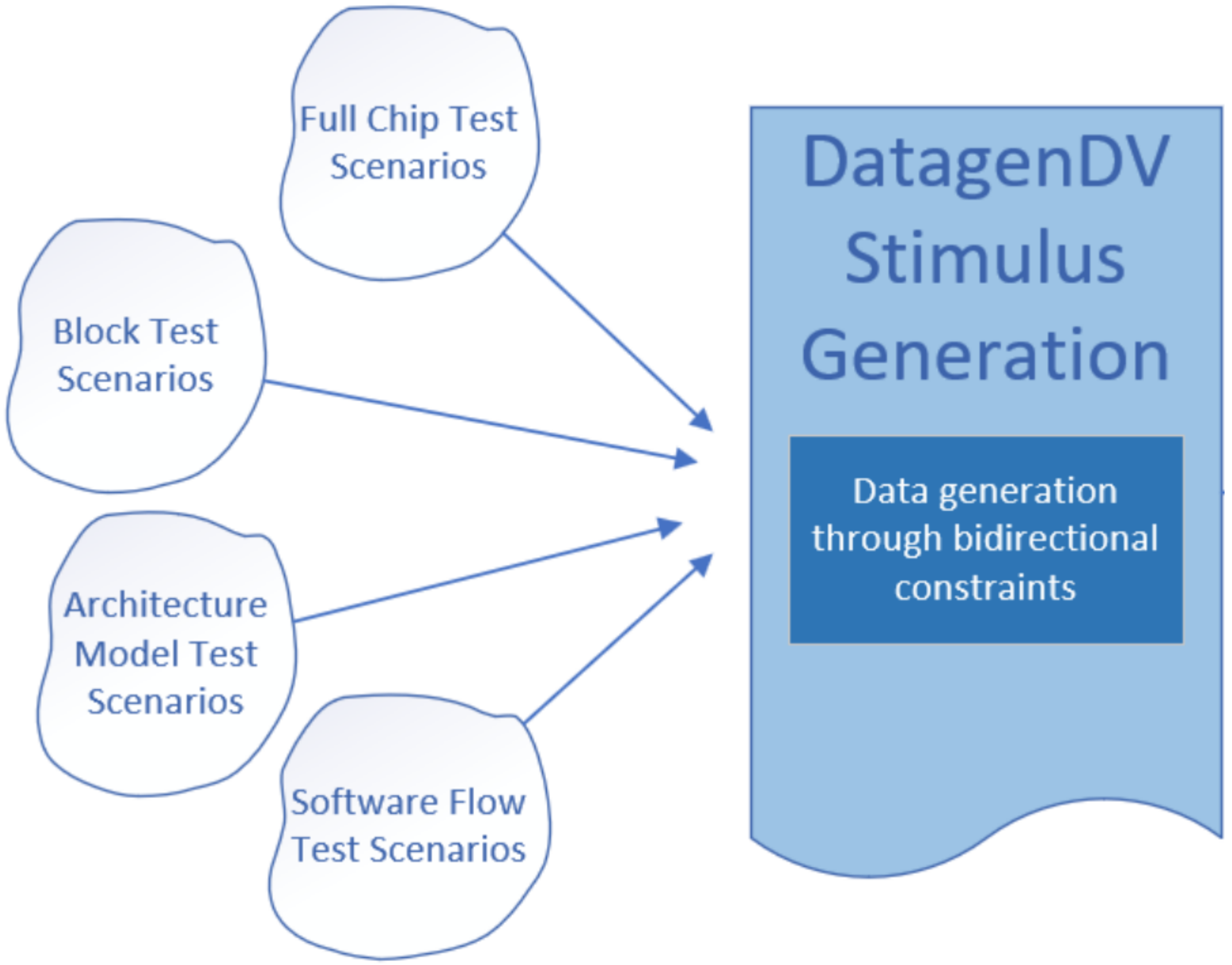
```
FrameParams(FRAME_SIZE=25100, FRAME_WIDTH=100, FRAME_HEIGHT=251)
```

Datagen Features

- Integrated PyVSC into dataclass objects
- Override values through constructor or YAML loading

Project Implementation

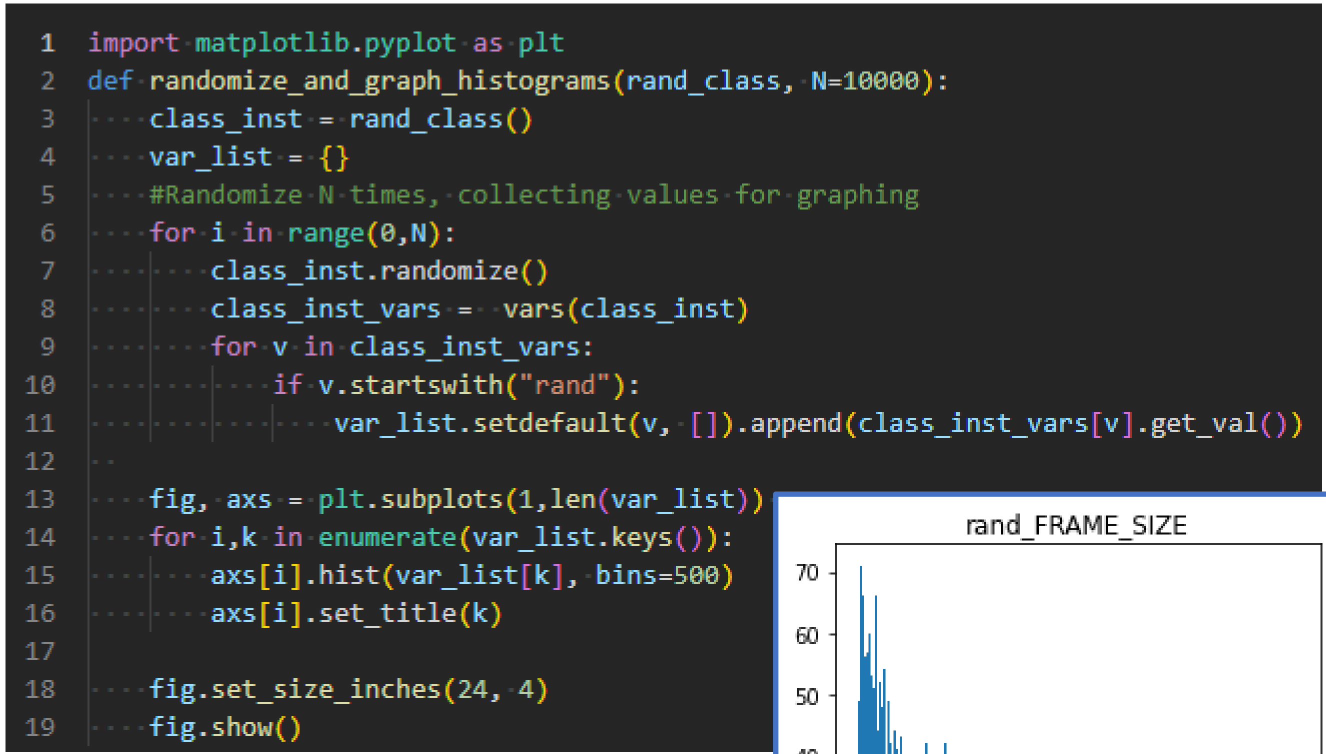
- Leveraged block constraints
- Supported many different scenarios
- YAML as a "universal language"


Full Chip Test Scenarios
Block Test Scenarios
Architecture Model Test Scenarios
Software Flow Test Scenarios
DatagenDV Stimulus Generation
Data generation through bidirectional constraints

Lessons Learned

- Successes
  - Designing up front!
  - Training others on Python
- Challenges
  - Ctypes doesn't support all packing formats
    - Solution: Prototype first
  - PyVSC can require fine tuning
    - Solution: Analyze randomization results

PyVSC – Randomization Analysis

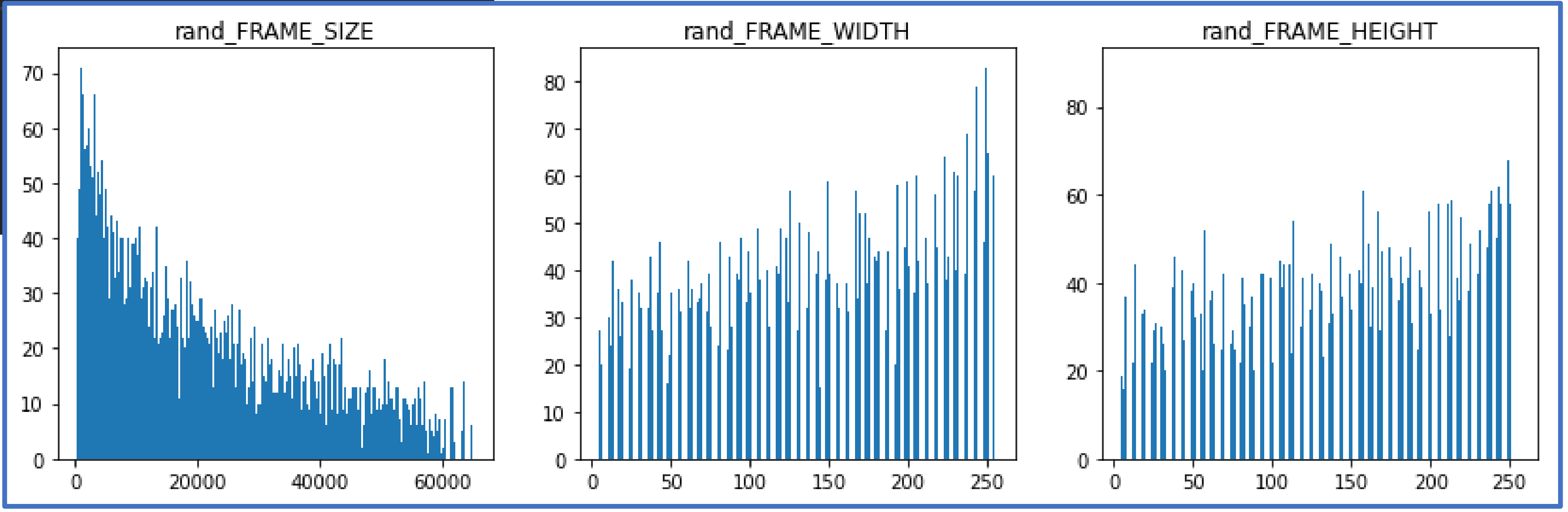


```python
import matplotlib.pyplot as plt
def randomize_and_graph_histograms(rand_class, N=10000):
    class_inst = rand_class()
    var_list = {}
    #Randomize N times, collecting values for graphing
    for i in range(0,N):
        class_inst.randomize()
        class_inst_vars =  vars(class_inst)
        for v in class_inst_vars:
            if v.startswith("rand"):
                var_list.setdefault(v, []).append(class_inst_vars[v].get_val())

    fig, axs = plt.subplots(1,len(var_list))
    for i,k in enumerate(var_list.keys()):
        axs[i].hist(var_list[k], bins=500)
        axs[i].set_title(k)

    fig.set_size_inches(24, 4)
    fig.show()
```

Leverage python’s data science libraries to analyze constraints!



What’s Next?

- DatagenDV is open source.
  - https://github.com/microsoft/datagenDV
- Future Features:
  - Leverage existing C definitions
  - Automated conversion of SV to PyVSC constraints
  - YAML Alternatives: XML? JSON?
  - PyVSC alternatives?

Questions?