Functional Testing with

Python & t:

Kay Schlühr (‘kaɪ ʃlyɐ)

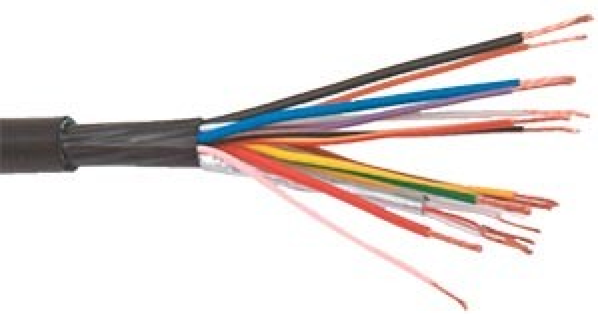
Functional Testing

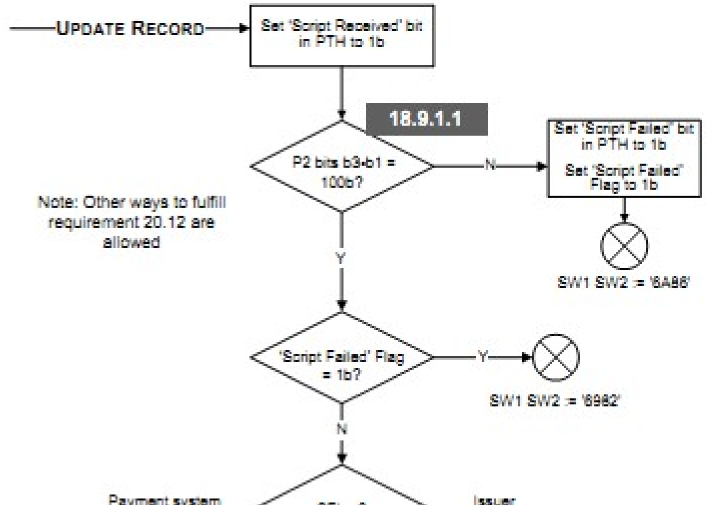
Wikipedia:

Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component under test.

Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered.

Text & Cable





F1 Symmetric Keys

The CPA application is required to maintain Session Key Counters and associated limits as specified in this section.

To support EMV common session key derivation, the ICC uses two 2-byte
ounter and SMI Session Key Counter, each with an

ınter is a two-byte counter initialized to zero that
erivations since successful validation of an ARPC.

ıunter is a two-byte counter initialized to zero that
derivations not followed by successful validation of a

derivation is controlled as follows:

ounter is less than the AC Session Key Counter
ıe AC Session Key Counter.

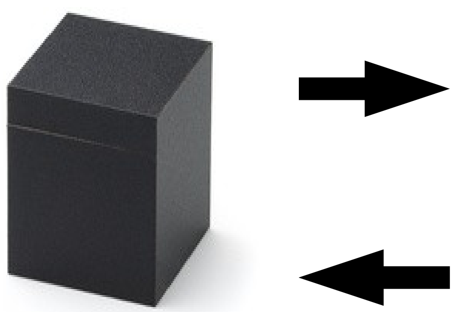
ssion Key Counter (after step 1) is greater than the
Key Counter Limit or has reached the value 'FF FF',
is not used, AC session key derivation is
plication responds to the GENERATE AC command

alue is not 'FF FF' and has not reached the counter
on continues with the AC session key derivation.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1								Log Declined Transactions
	1							Log Approved Transactions
		x						Log Offline Only[8]
		0						Log Both Offline and Online
		1						Log Offline Only
			1					Log the ATC
				1				Log the CID
					1			Log the CVR
						x		RFU
							x	RFU

Resource 2 TLV coded	... ...	Resource n TLV coded	'F F ...... F F'

Text & Bytes



... AF 89 45 00 ...

... FE 89 00 00 ...

F1 Symmetric Keys

The CPA application is required to maintain Session Key Counters and associated limits as specified in this section.

To support EMV common session key derivation, the ICC uses two 2-byte
ounter and SMI Session Key Counter, each with an

ınter is a two-byte counter initialized to zero that
erivations since successful validation of an ARPC.

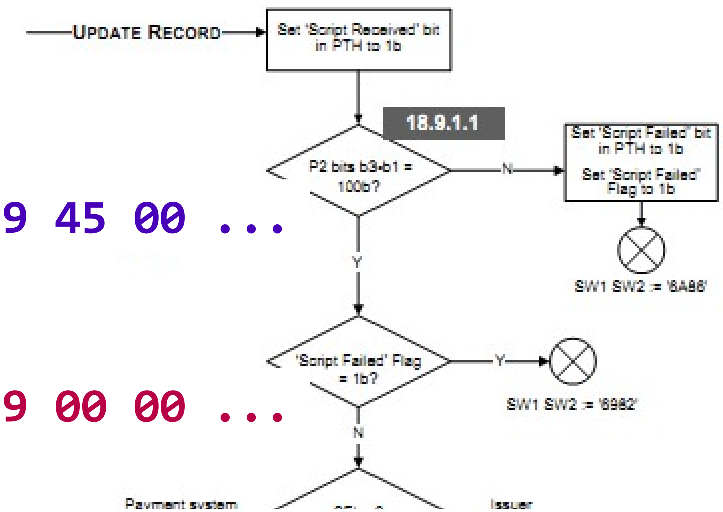
ounter is a two-byte counter initialized to zero that
derivations not followed by successful validation of a

derivation is controlled as follows:

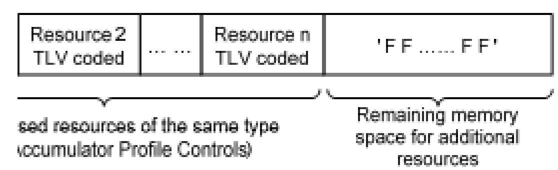
ounter is less than the AC Session Key Counter
ıe AC Session Key Counter.

ssion Key Counter (after step 1) is greater than the
Key Counter Limit or has reached the value 'FF FF',
is not used, AC session key derivation is
plication responds to the GENERATE AC command

ralue is not 'FF FF' and has not reached the counter
on continues with the AC session key derivation.



b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1								Log Declined Transactions
	1							Log Approved Transactions
		x						Log Offline Only[a]
		0						Log Both Offline and Online
		1						Log Offline Only
			1					Log the ATC
				1				Log the CID
					1			Log the CVR
						x		RFU
							x	RFU



Big Bang

System Test
Integration Test
Module Test

→ `... AF 89 45 00 ...`

← `... FE 89 00 00 ...`

Testing and tested commands
are not distinguished

↓

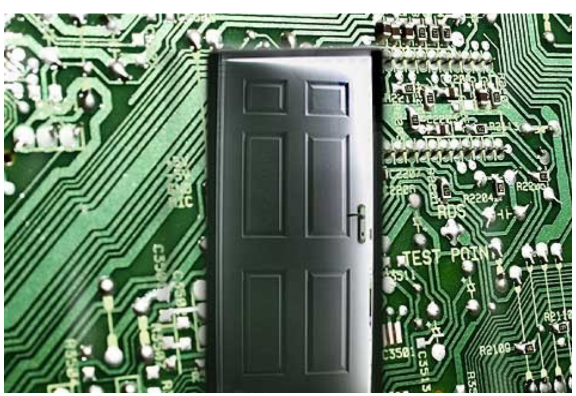
Low modularity

Module Decomposition

System Test
Integration Test
Module Test

`... AF 89 45 00 ...`

`... FE 89 00 00 ...`



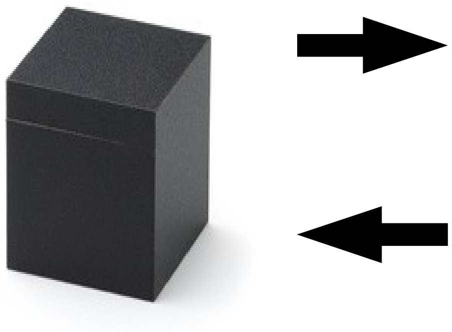
Special test commands as a backdoor

↓

Simulate system state

Module Test with Developer Agreement

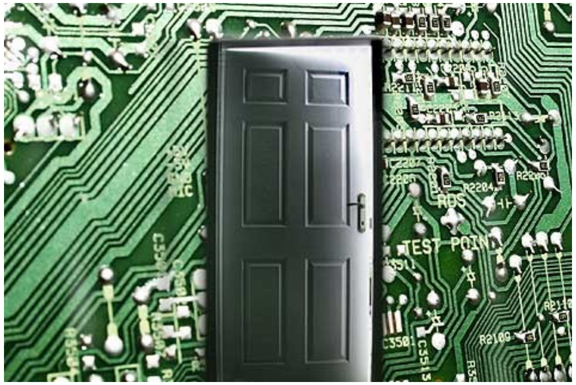
System Test
Integration Test
Module Test



... AF 89 45 00 ...

... FE 89 00 00 ...

```
#ifdef _MODULETEST_
```



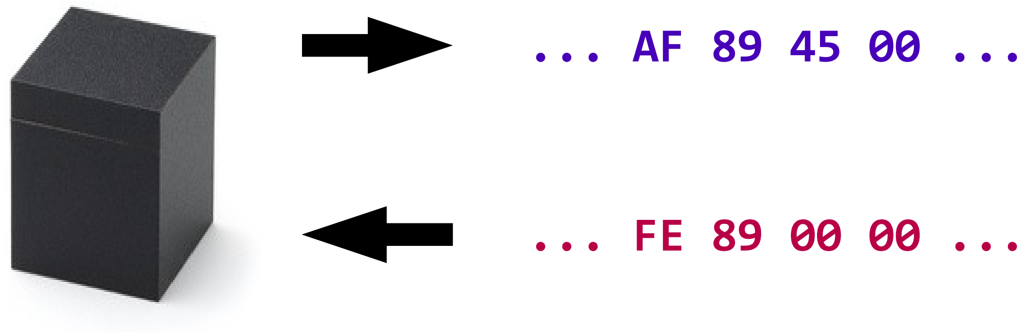

```
#endif // _MODULETEST_
```

Special test commands as a backdoor

↓

Simulate system state

Methods of Functional Testing



Functional Decomposition Methods

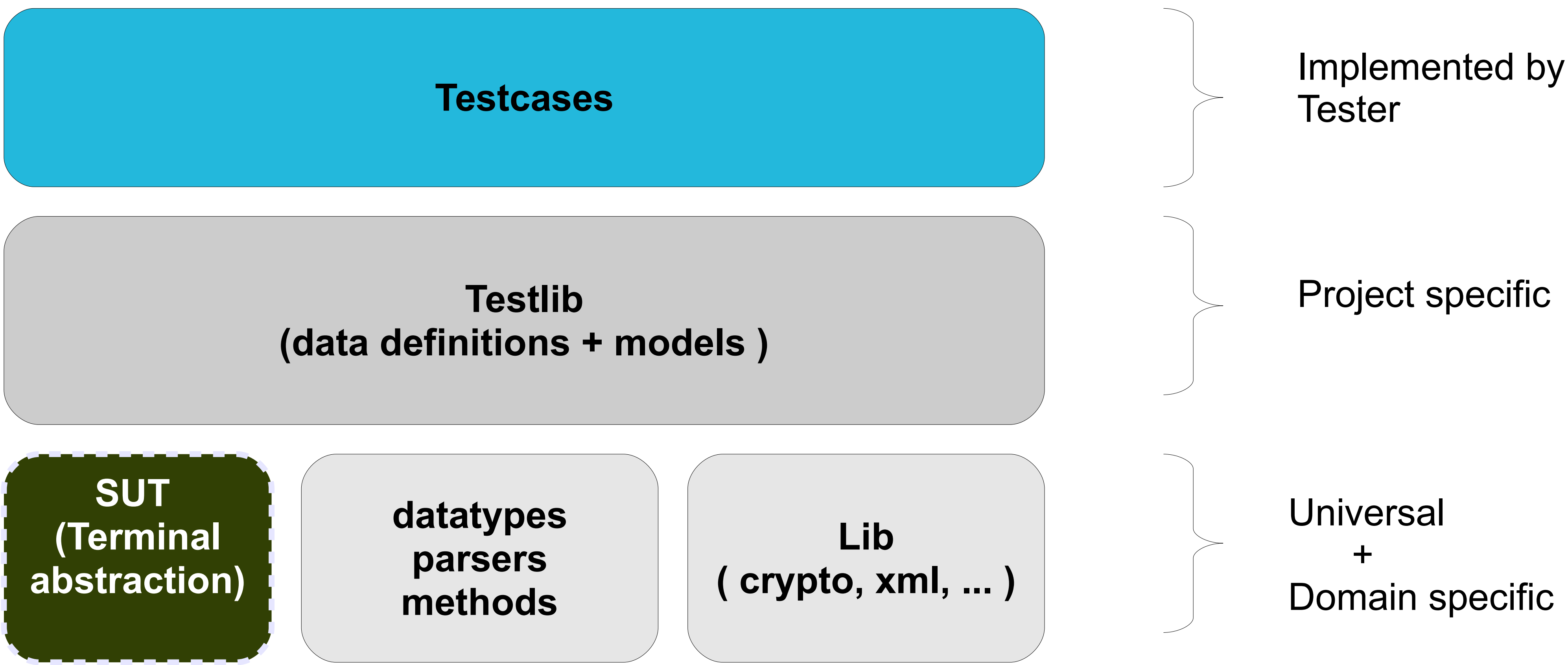
- Equivalence class partition
- State machine paths
- Classification Tree Method (CTM)
- ...

Testsystem

?

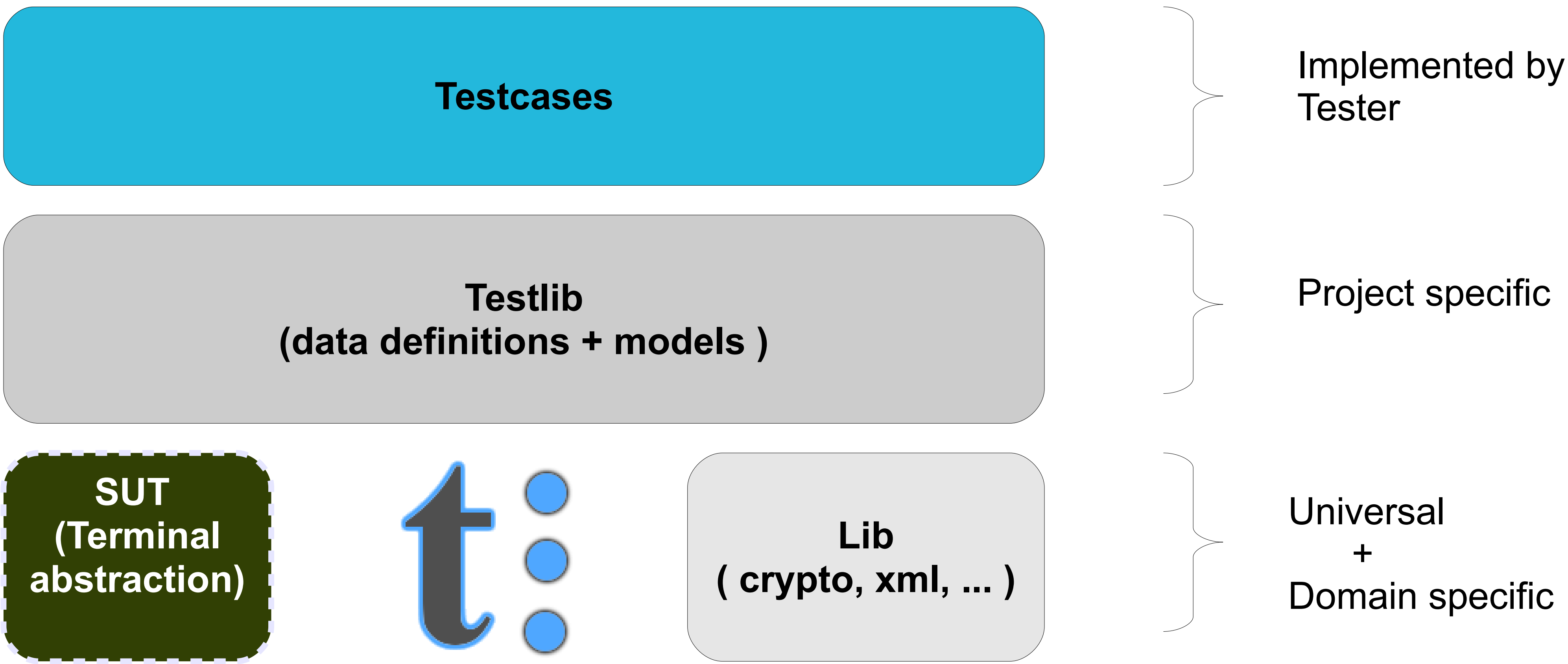
Testsystem

Testsystem - including 3rd party libraries and frameworks

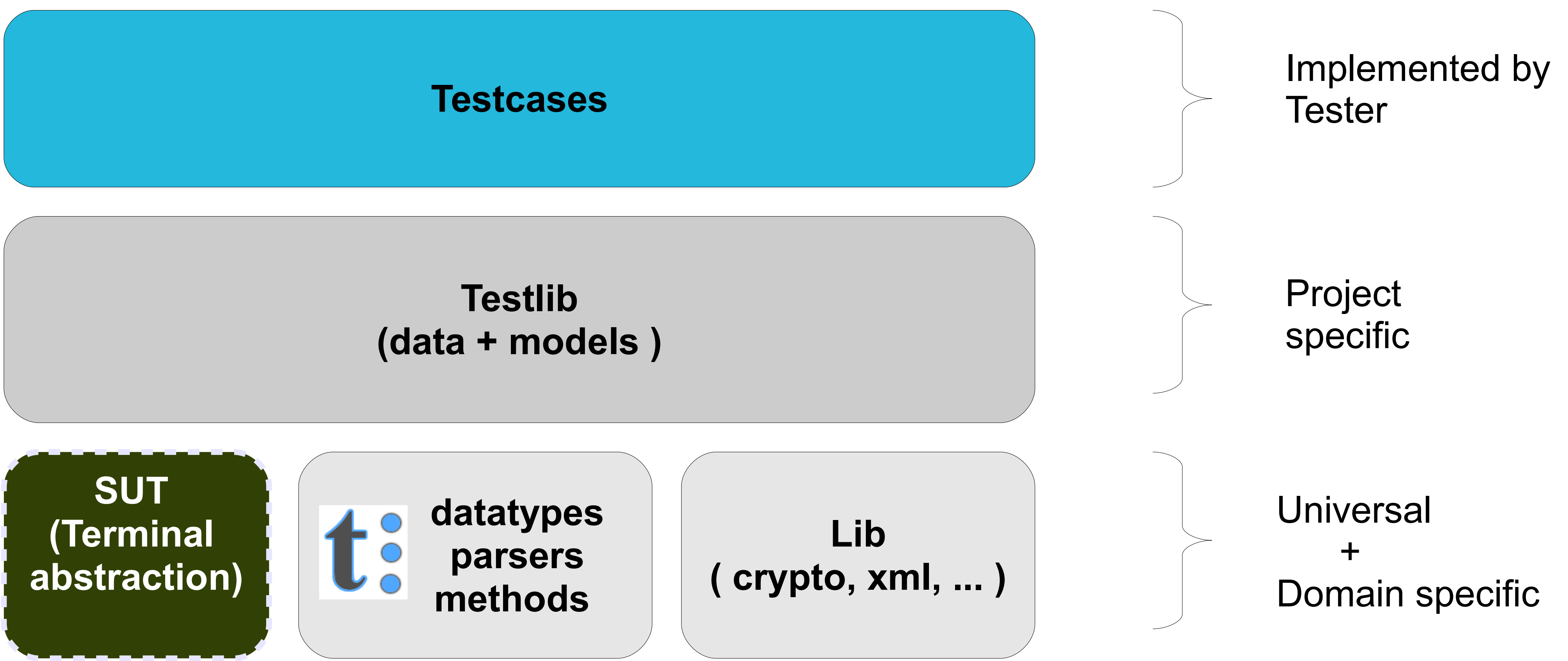


Testsystem

Testsystem - including 3rd party libraries and frameworks



Testsystem - including 3rd party libraries and frameworks



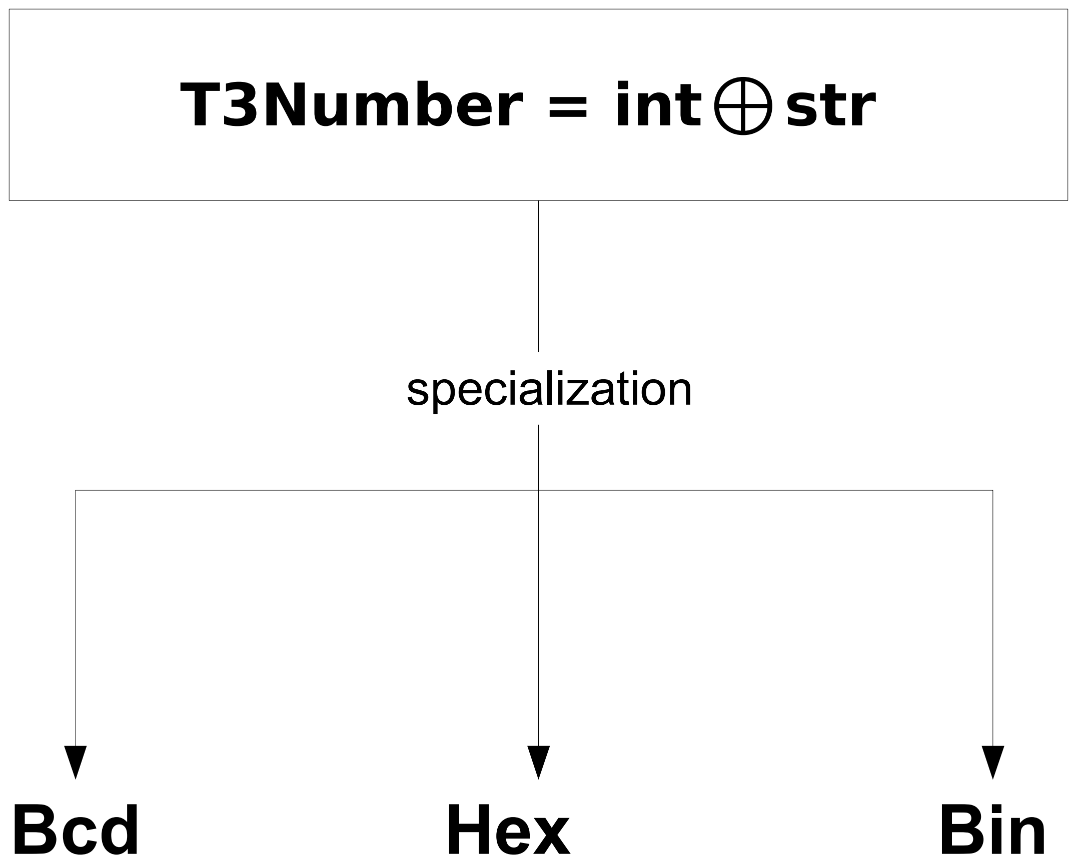
Library for datatypes and methods

- Data representation (T3Number)
- Analysis and synthesis of data (T3Table)
- Model checking (T3Chart)
- Test contexts (T3TableContext)

t3

T3Number	T3Table
T3TableContext	**T3Chart**

t3.number.T3Number



t3.number.T3Number

Construct

```
>>> Hex('00 01 Af 03')
00 01 AF 03
>>> Hex(2425)
09 79
>>> Bin('01011')
01011
```

Convert

```
>>> Hex('00 01 Af 03').number()
110339
>>> Hex('00 01 Af 03').digits()
'0001AF03'
>>> Hex('00 01 Af 03').bytes()
array('b', [0, 1, -81, 3])
```

t3.number.T3Number

Construct

```
>>> Hex('00 01 Af 03')
00 01 AF 03
>>> Hex(2425)
09 79
>>> Bin('01011')
01011
```

Convert

```
>>> Hex('00 01 Af 03').number()
110339
>>> Hex('00 01 Af 03').digits()
'0001AF03'
>>> Hex('00 01 Af 03').bytes()
array('b', [0, 1, -81, 3])
```

t3.number.T3Number

Construct

```
>>> Hex('00 01 Af 03')
00 01 AF 03
>>> Hex(2425)
09 79
>>> Bin('01011')
01011 (h'0B)
```

Convert

```
>>> Hex('00 01 Af 03').number()
110339
>>> Hex('00 01 Af 03').digits()
'0001AF03'
>>> Hex('00 01 Af 03').bytes()
array('b', [0, 1, -81, 3])
```

Arithmetics

```
>>> Hex('00 01') * 10
00 0A
>>> 10 + Hex('00 01')
00 0B
>>> ~Bin('01011')
10100 (h'14)
>>> Hex('26 F5').powmod(0x56, 'FF')
E5
>>> Hex('00 01') >> 10
00 00
```

Sequence


```
>>> Hex('00 01 Af 03').number()
110339
>>> Hex('00 01 Af 03').digits()
'0001AF03'
>>> Hex('00 01 Af 03').bytes()
array('b', [0, 1, -81, 3])
```

t3.number.T3Number

Construct

```
>>> Hex('00 01 Af 03')
00 01 AF 03
>>> Hex(2425)
09 79
>>> Bin('01011')
01011 (h'0B)
```

Convert

```
>>> Hex('00 01 Af 03').number()
110339
>>> Hex('00 01 Af 03').digits()
'0001AF03'
>>> Hex('00 01 Af 03').bytes()
array('b', [0, 1, -81, 3])
```

Arithmetics

```
>>> Hex('00 01') * 10
00 0A
>>> 10 + Hex('00 01')
00 0B
>>> ~Bin('01011')
10100 (h'14)
>>> Hex('26 F5').powmod(0x56, 'FF')
E5
>>> Hex('00 01') >> 10
00 00
```

Sequences

```
>>> Hex('00 01')[1]
01
>>> Bin('01011')[1]
1
>>> Hex('00 01') // Hex('0A 89')
00 01 0A 89
>>> len(Hex('00 01'))
1
>>> list(Hex('00 01'))
[00, 01]
```

t3

T3Number	T3Table

`>>> Hex('00 01 AF 03') + 1`
`00 01 AF 03`
 	
`>>> Hex('00 01 AF 03') // Hex(16)`
`00 01 AF 03 10`
 	
`>>> Bin('010111')*8`
`010111000`	

T3TableContext	**T3Chart**

t3.table.T3Table

Construct

```
Tlv = T3Table()
Tlv.add(1, Tag = '00')
Tlv.add(1, Len = '01')
Tlv.add('*', Value = '00')

>>> Tlv
Tlv:
    Tag: 00
    Len: 01
    Value: 00
```

t3.table.T3Table

Parse

```
>>> Tlv << '0A 04 00 01 02 F0'
<t3.table.T3Table object at 0x03821490>
    Tag: 0A
    Len: 04
    Value: 00 01 02 F0

>>> Tlv << 'F0 06 00 01 02 F0 00 78'
<t3.table.T3Table object at 0x038210F0>
    Tag: F0
    Len: 06
    Value: 00 01 02 F0 00 78
```

Construct

```
Tlv = T3Table()
Tlv.add(1, Tag = '00')
Tlv.add(1, Len = '01')
Tlv.add('*', Value = '00')
```

t3.table.T3Table

Unparse

```
>>> P = Tlv << '0A 04 00 01 02 F0'
>>> P
P:
    Tag: 0A
    Len: 04
    Value: 00 01 02 F0

>>> Hex(P)
0A 04 00 01 02 F0

>>> H = Hex('0A 04 00 01 02 F0')
>>> H == Hex(Tlv << H)
True
```

Construct

```
Tlv = T3Table()
Tlv.add(1, Tag = '00')
Tlv.add(1, Len = '01')
Tlv.add('*', Value = '00')
```

t3.table.T3Table

Pattern Bindings

```
def taglen(tlv, data):
    return 2 if data[0] & 0x1F == 0x1F else 1

def lenlen(tlv, data):
    if data[0] & 0x80 == 0x80:
       return 1 + data[0] & 0x0F

Tlv = T3Table()
Tlv.add(taglen, Tag = '00')
Tlv.add(lenlen, Len = '01')
Tlv.add('*', Value = '00')
```

Construct

```
Tlv = T3Table()
Tlv.add(1, Tag = '00')
Tlv.add(1, Len = '01')
Tlv.add('*', Value = '00')
```

Parse

```
>>> Tlv << '0A 04 00 01 02 F0'
<t3.table.T3Table object at 0x03821490>
    Tag: 0A
    Len: 04
    Value: 00 01 02 F0
```

t3.table.T3Table

Pattern Bindings

```
def taglen(tlv, data):
    return 2 if data[0] & 0x1F == 0x1F else 1

def lenlen(tlv, data):
    if data[0] & 0x80 == 0x80:
       return 1 + data[0] & 0x0F

Tlv = T3Table()
Tlv.add(taglen, Tag = '00')
Tlv.add(lenlen, Len = '01')
Tlv.add('*', Value = '00')


>>> Tlv << 'F0 06 00 01 02 F0 00 78'
<__main__.T3Table object at 0x024DF2F0>
    Tag: 9F 01
    Len: 81 05
    Value: 01 02 03 04 05
```

Construct

```
Tlv = T3Table()
Tlv.add(1, Tag = '00')
Tlv.add(1, Len = '01')
Tlv.add('*', Value = '00')
```

Parse

```
>>> Tlv << '0A 04 00 01 02 F0'
<t3.table.T3Table object at 0x03821490>
    Tag: 0A
    Len: 04
    Value: 00 01 02 F0
```

t3.table.T3Table

Value Bindings

```
def BERLen(v):
    n = Hex(v.get_value())
    if len(n) > 1:
        return (0x80 | len(n)) // n
    else:
        return k

Tlv = T3Table()
Tlv.add(taglen, Tag = '00')
Tlv.add(lenlen, Len = binding.table(BERLen, 'Value')
Tlv.add('*', Value = '00')
```

t3.table.T3Table

Value Bindings

```
def BERLen(v):
    n = Hex(v)
    if len(n) > 1:
        return (0x80 | len(n)) // n
    else:
        return k

Tlv = T3Table()
Tlv.add(taglen, Tag = 'F0')
Tlv.add(lenlen, Len = binding.table(BERLen, 'Value'))
Tlv.add('*', Value = '00')

>>> Tlv.Value = '01 AA 00'
>>> Tlv
Tlv:
    Tag: 01
    Len: 03
    Value: 01 AA 00
```

t3.table.T3Table

Value Bindings

```
def BERLen(v):
    n = Hex(v)
    if len(n) > 1:
        return (0x80 | len(n)) // n
    else:
        return k

Tlv = T3Table()
Tlv.add(taglen, Tag = 'F0')
Tlv.add(lenlen, Len = binding.table(BERLen, 'Value'))
Tlv.add('*', Value = '00')

>>> Tlv.Value = '01 AA 00'
>>> Tlv
Tlv:
    Tag: 01
    Len: 03
    Value: 01 AA 00
```

```
Tlv = T3Table()
Tlv.add(1, Tag = '00')
Tlv.add(1, Len = '01')
Tlv.add('*', Value = '00')

>>> Tlv.Value = '01 AA 00'
>>> Tlv
Tlv:
    Tag: 00
    Len: 01
    Value: 01 AA 00
```

t3.table.T3Table

T3Table instances as T3Table constructors

```
T_A7 = Tlv(Tag = 0xA7, Value = '00 00')

>>> T_A7
T_A7:
     Tag: A7
     Len: 02
     Value: 00 00
```

t3

T3Number

```
>>> Hex('00 01 AF 03') + 1
00 01 AF 04

>>> Hex('00 01 AF 03') // Hex(16)
00 01 AF 03 10

>>> Bin('010111')*8
010111000
```

T3Table

```
>>> T = Tlv(Tag='A0', Value='01 AF')
>>> Hex(T) == 'A0 02 01 AF'
True

>>> len(T)
04

>>> Hex(T << Hex(T)) == Hex(T)
True
```

T3TableContext

T3Chart

t3.table.T3TableContext

```python
class T3TableContext(T3Table):
    def __init__(self):
        super(T3TableContext, self).__init__()
        self._status = "OK"

    def __enter__(self):
        return self

    def __exit__(self, typ, value, tb):
        if typ:
            if typ in (AssertionError, MatchingFailure):
                self._status = "FAIL"
                traceback.print_exc()
            else:
                self._status = "ERROR"
                traceback.print_exc()
```

t3.table.T3TableContext

Example

```
Response = T3TableContext()
Reponse.add('*', Data = '00')
Reponse.add(2, SW = '0000')
```

```
with terminal.send(data) as response:
   assert response.SW == '9000'
```

t3

T3Number

```
>>> Hex('00 01 AF 03') + 1
00 01 AF 04

>>> Hex('00 01 AF 03') // Hex(16)
00 01 AF 03 10

>>> Bin('010111')*8
010111000
```

T3Table

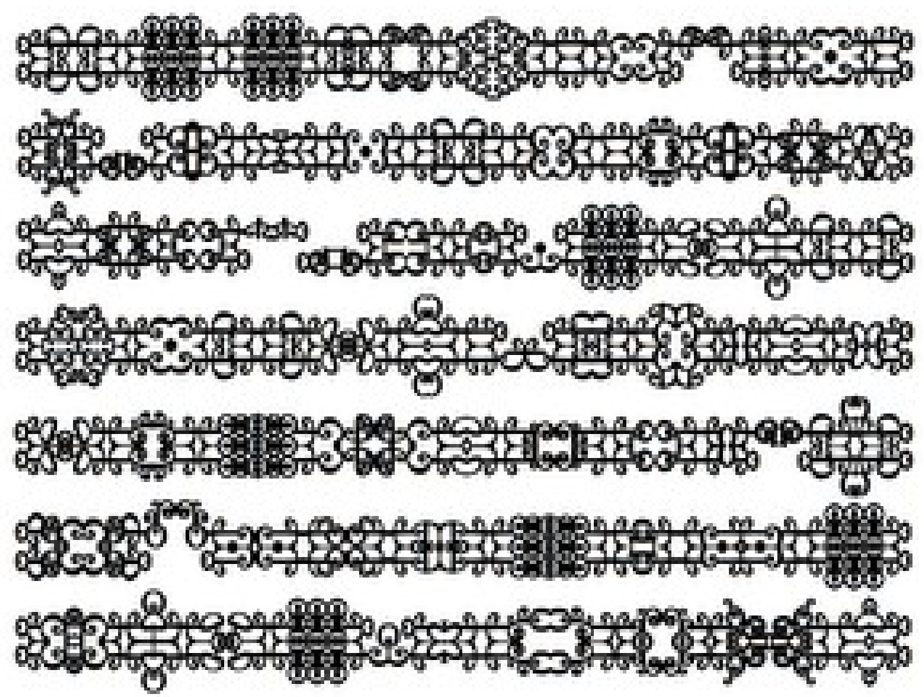
```
>>> T = Tlv(Tag='A0', Value='01 AF')
>>> Hex(T) == 'A0 02 01 AF'
True

>>> len(T)
04

>>> Hex(T << Hex(T)) == Hex(T)
True
```

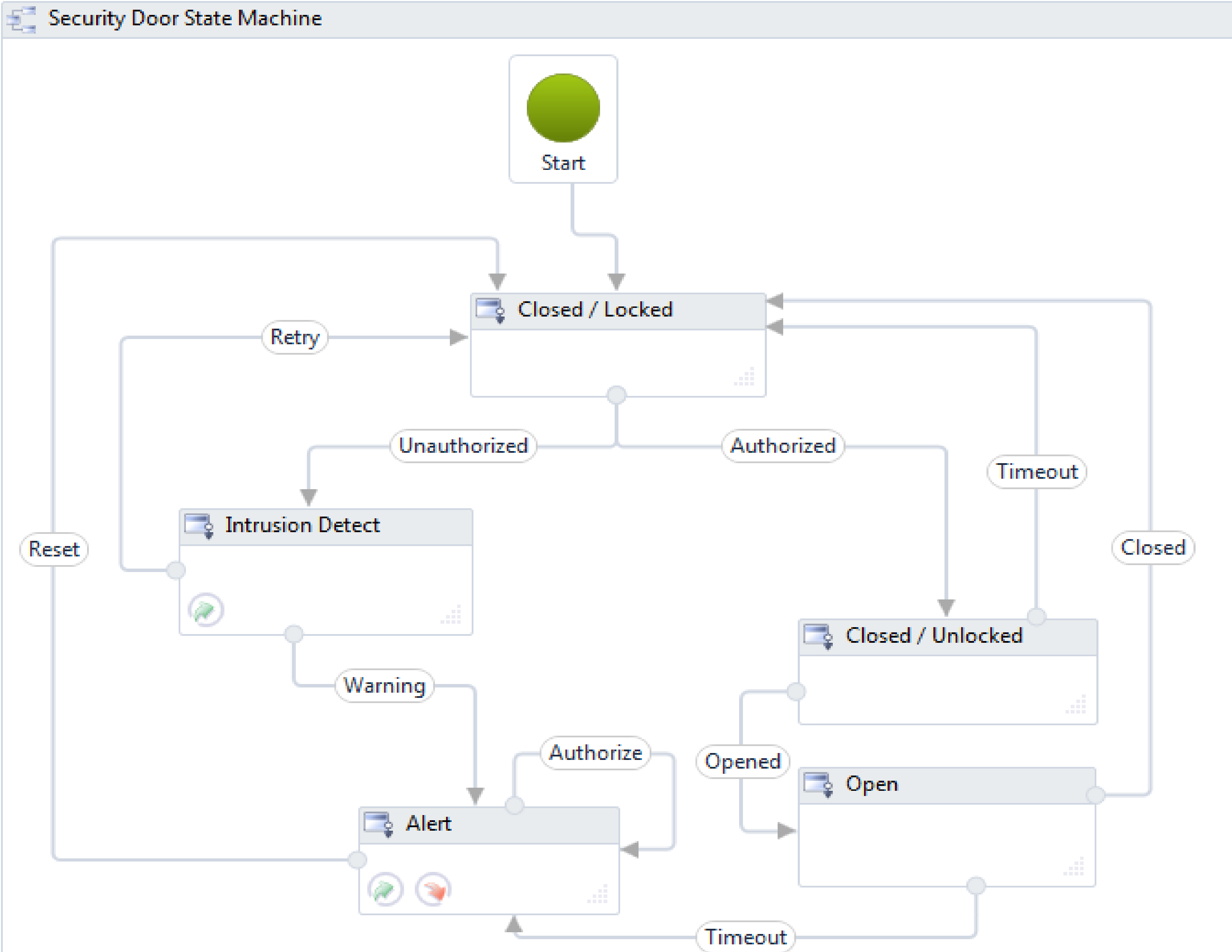
T3TableContext

```
Response = T3TableContext()
Reponse.add('*', Data = '00')
Reponse.add(2, SW = '0000')

with terminal.send(data) as response:
    assert response.SW == '9000'
```

T3Chart

Charts and Choosers





Security Door State Machine
Start
Closed / Locked
Retry
Unauthorized
Authorized
Timeout
Intrusion Detect
Reset
Closed
Closed / Unlocked
Warning
Authorize
Opened
Open
Alert
Timeout

@flow T3Chart

```
class SecurityDoorStateMachine(T3Chart):

    @flow
    def chart(self, chooser):
        ...
        choice = chooser.choose([...])
```

Charts and Choosers

stack

```
[]
```

pop()

1

```
chosen = []
```

```
class ExampleT3Chart(T3Chart):
    @flow
    def chart(self, chooser):
        x = chooser.choose([0,1])
        if x == 0:
            y = chooser.choose(["a","b"])
        else:
            y = chooser.choose(["c","d"])
        return vars()
```

push([..])

Charts and Choosers

2

stack

[0]
[1]

pop()

chosen = [1]

1

```
class ExampleT3Chart(T3Chart):
    @flow
    def chart(self, chooser):
        x = chooser.choose([0,1])          # choice x=1
        if x == 0:
            y = chooser.choose(["a","b"])
        else:
            y = chooser.choose(["c","d"])
        return vars()
```

push([..])

Charts and Choosers

3

stack

[0]
[1, 'c']
[1, 'd']

pop()

chosen = [1, 'd']

1

'd'

```
class ExampleT3Chart(T3Chart):
    @flow
    def chart(self, chooser):
        x = chooser.choose([0,1])            # choice x=1
        if x == 0:
            y = chooser.choose(["a","b"])
        else:
            y = chooser.choose(["c","d"])    # choice y='d'
        return vars()
```

Charts and Choosers

4

stack

[0]
[1, 'c']

pop()

chosen = [1, 'c']

```
class ExampleT3Chart(T3Chart):
    @flow
    def chart(self, chooser):
1 →     x = chooser.choose([0,1])          # choice x=1
        if x == 0:
            y = chooser.choose(["a","b"])
        else:
'c' →       y = chooser.choose(["c","d"])  # choice y='c'
        return vars() ———●
```

Charts and Choosers

5

stack

[0]

pop()

chosen = [0]

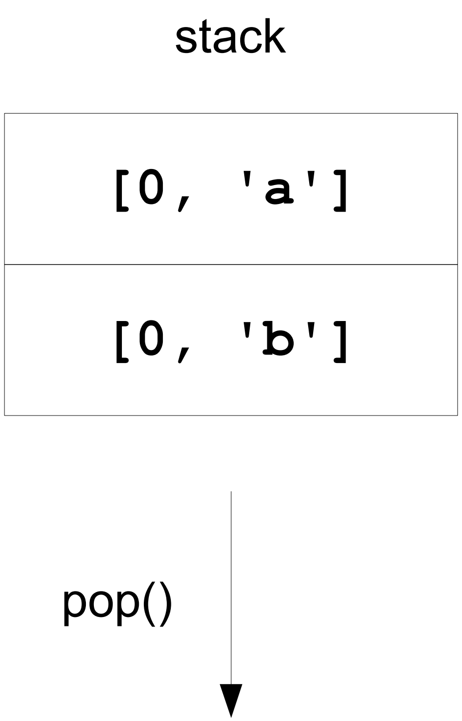
0

```
class ExampleT3Chart(T3Chart):
    @flow
    def chart(self, chooser):
        x = chooser.choose([0,1])          # choice x=0
        if x == 0:
            y = chooser.choose(["a","b"])
        else:
            y = chooser.choose(["c","d"])
        return vars()
```

push([..])

Charts and Choosers

6



chosen = [0, 'b']

0 → (x = chooser.choose)

'b' → (y = chooser.choose(["a","b"]))

```
class ExampleT3Chart(T3Chart):
    @flow
    def chart(self, chooser):
        x = chooser.choose([0,1])             # choice x=0
        if x == 0:
            y = chooser.choose(["a","b"])     # choice y='b'
        else:
            y = chooser.choose(["c","d"])
        return vars()
```

Charts and Choosers

stack

[0, 'a']

pop()

chosen = [0, 'a']

0 → (arrow to line `x = chooser.choose([0,1])`)

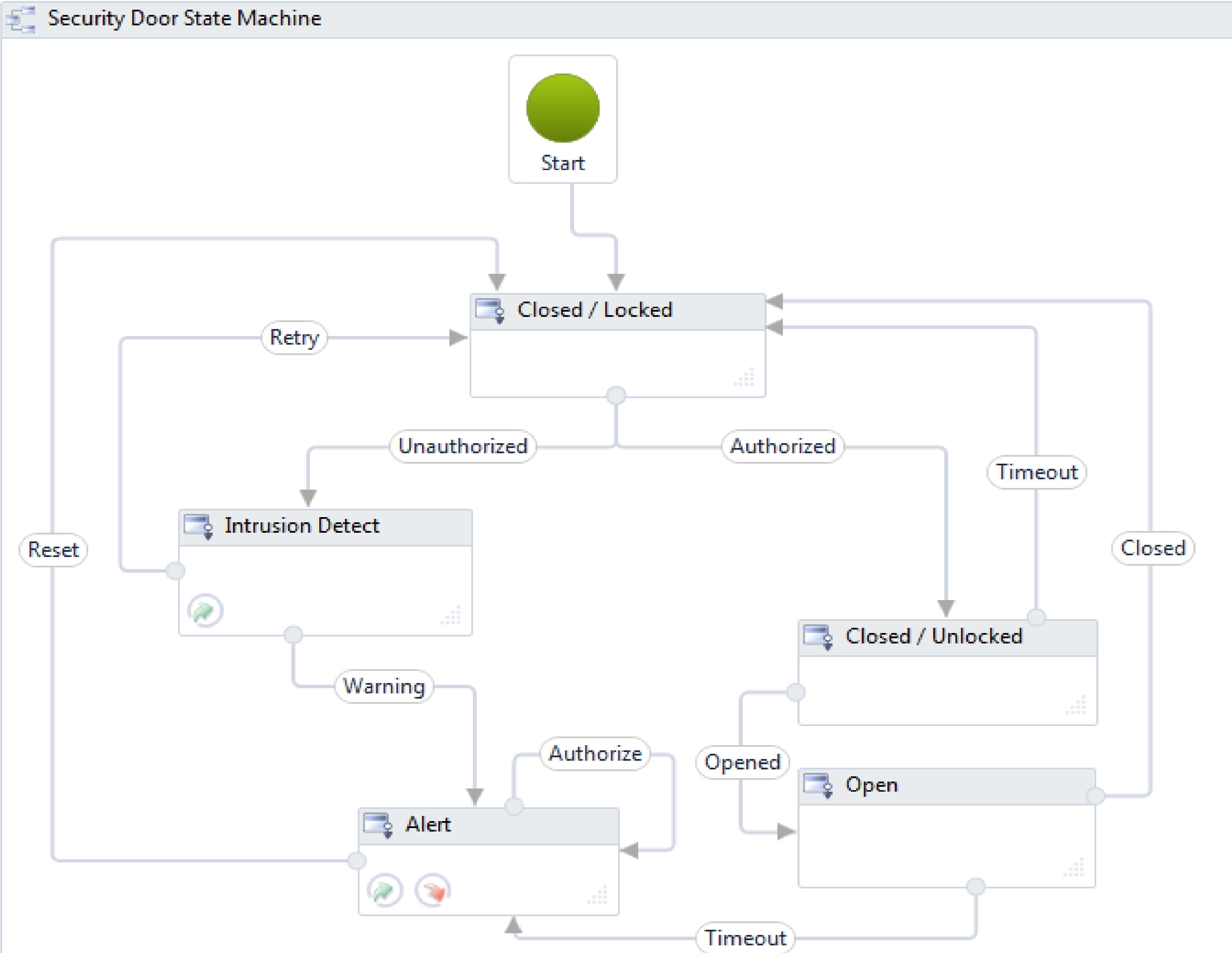
'a' → (arrow to line `y = chooser.choose(["a","b"])`)

```python
class ExampleT3Chart(T3Chart):
    @flow
    def chart(self, chooser):
        x = chooser.choose([0,1])              # choice x=0
        if x == 0:
            y = chooser.choose(["a","b"])      # choice y='a'
        else:
            y = chooser.choose(["c","d"])
        return vars()
```

Charts and Choosers

stack


```
>>> chart = ExampleT3Chart()
>>> chart.create()
>>> for asn in chart.assignments:
...     print asn
{'y': 'a', 'x': 0}
{'y': 'b', 'x': 0}
{'y': 'c', 'x': 1}
{'y': 'd', 'x': 1}
```


Security Door State Machine
Start
Closed / Locked
Retry
Unauthorized
Authorized
Timeout
Intrusion Detect
Reset
Closed
Closed / Unlocked
Warning
Authorize
Opened
Open
Alert
Timeout

State machine with choosers

```
class SecurityDoorStateMachine(T3Chart):
    @flow
    def chart(self, chooser):
        ...
        state = 'Closed/Locked'
        while True:
            if state == 'Closed/Locked':
                trans = chooser.choose(["Authorized", "Unauthorized"])
                sequence.append((state, trans))
                if trans == 'Authorized':
                    state = 'Closed/Unlocked'
                elif trans == 'Unauthorized':
                    state = 'Intrusion Detect'
            elif state == 'Intrusion Detect':
                trans = chooser.choose(["Retry", "Warning"])
                sequence.append((state, trans))
                if trans == 'Warning':
                    state = 'Alert'
                elif unlock_counter>0:
                    unlock_counter-=1
                    state = 'Closed/Locked'
                else:
                    state = 'Alert'
            elif state == 'Alert':
                trans = chooser.choose(["Reset", "Fin", "Authorize"])
                ...
```

t3

T3Number	T3Table

T3Number

```
>>> Hex('00 01 AF 03') + 1
00 01 AF 04

>>> Hex('00 01 AF 03') // Hex(16)
00 01 AF 03 10

>>> Bin('010111')*8
010111000
```

T3Table

```
>>> T = Tlv(Tag='A0', Value='01 AF')
>>> Hex(T) == 'A0 02 01 AF'
True

>>> len(T)
04

>>> Hex(T << Hex(T)) == Hex(T)
True
```

T3TableContext

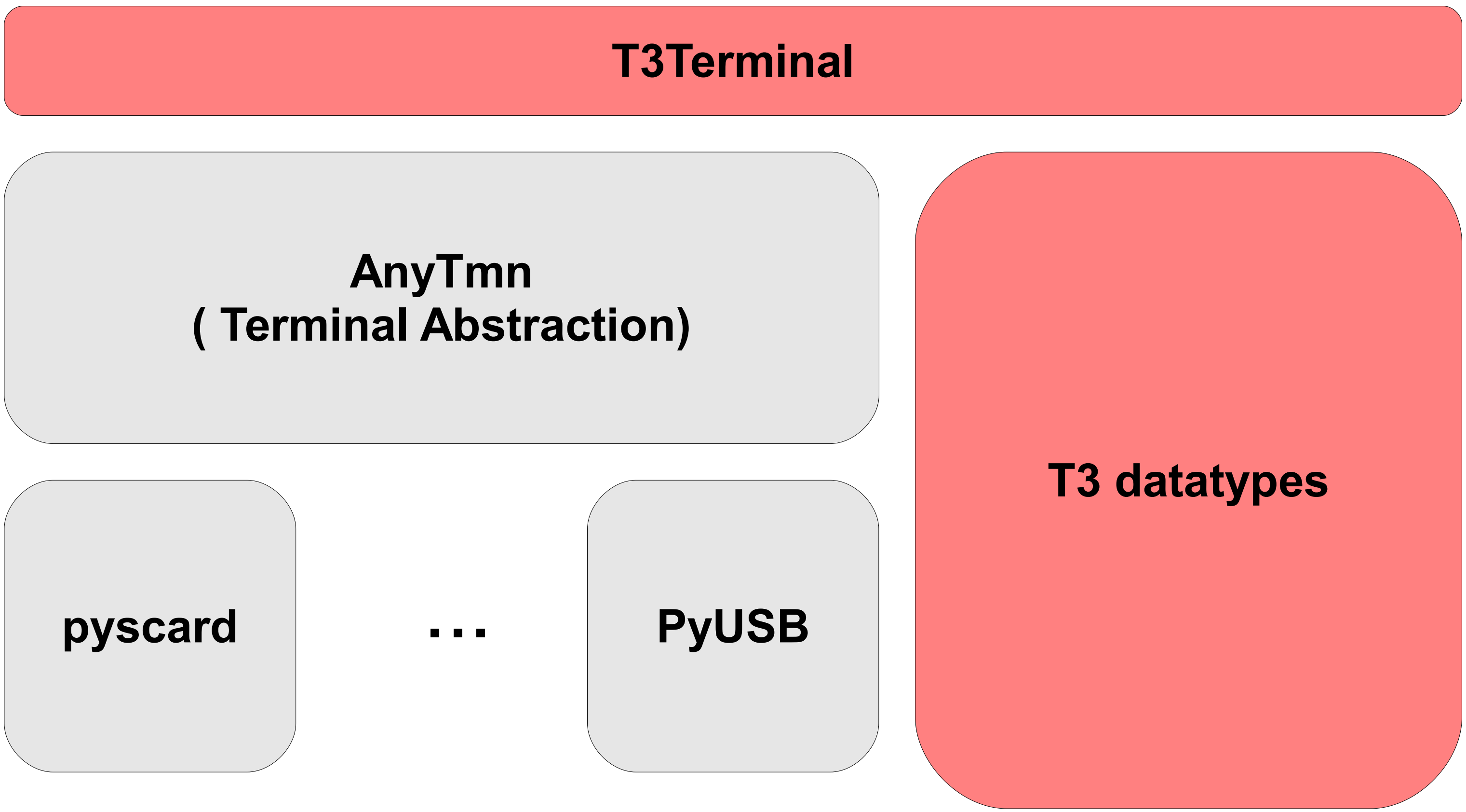
```
Response = T3TableContext()
Reponse.add('*', Data = '00')
Reponse.add(2, SW = '0000')

with terminal.send(data) as response:
    assert response.SW == '9000'
```

T3Chart

```
class ExampleT3Chart(T3Chart):
    @flow
    def chart(self, chooser):
        x = chooser.choose([0,1])
        if x == 0:
            y = chooser.choose(["a","b"])
        else:
            y = chooser.choose(["c","d"])
        return vars()
```

Future?

AnyTmn


T3Terminal
AnyTmn
(Terminal Abstraction)
pyscard
...
PyUSB
T3 datatypes

Thanks for your attention!

https://pypi.python.org/pypi/t3