# Constraints / Capabilities Framework (CCF)

Alain Bouchard, ing.

Matrox Video



v1.0

# Part 1

- The theoretical framework

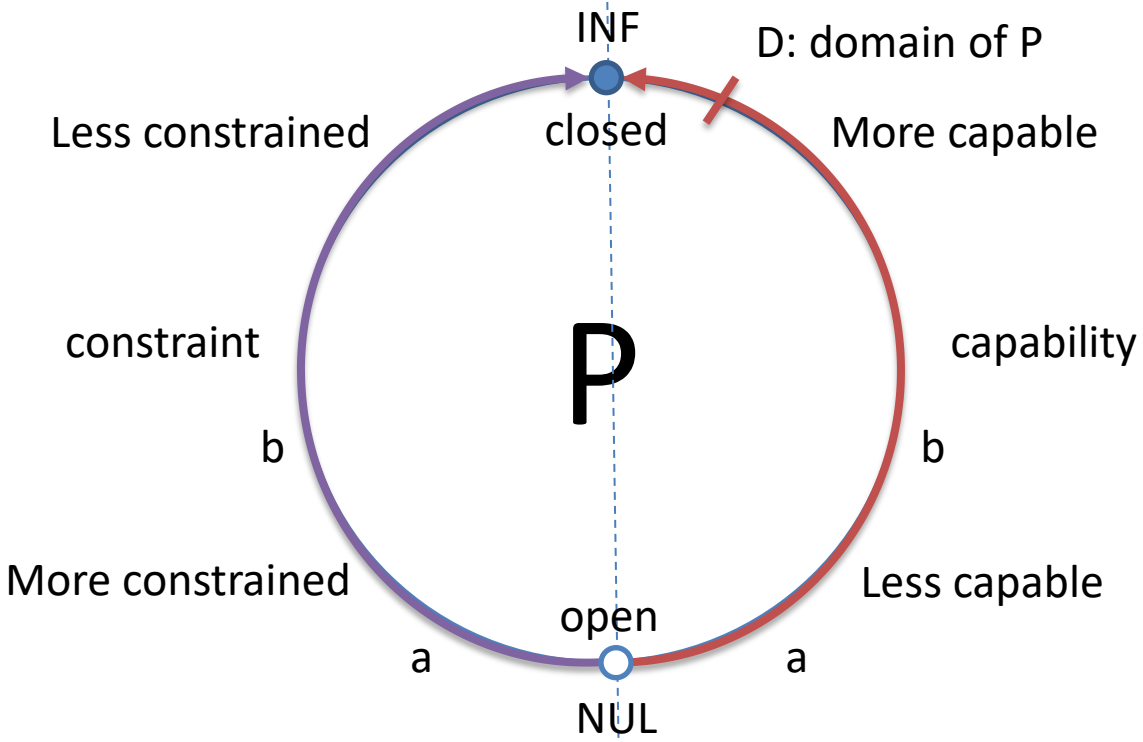
# Capabilities

- Describe what is allowed, available or possible.
- Describe the possible values of a parameter.
  - Capability Space (CapS) of parameter P
- A capability essentially outlines the permissible values or operational space of a parameter, representing what is possible or supported.

# Constraints

- Constrain what is allowed, available or possible.
- Constrain the possible values of a parameter.
  - Constraint Space (ConS) of parameter P
- A constraint essentially restricts the permissible values or operational space of a parameter, representing what is possible or supported. Its key purpose is to **reduce or limit the flexibility** provided by capabilities.

# Constraints / Capabilities

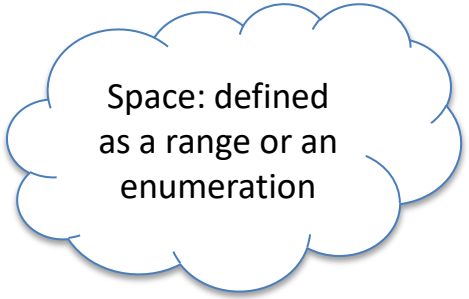



a <= b : a is more or equally constrained

a <= b : a is less or equally capable

# Parameter P

- Capability / Constraint Space of P (CapS / ConS)
  - Types: Bool, Integer, Float, Rational, String or Untyped
  - Range: Finite or Infinite

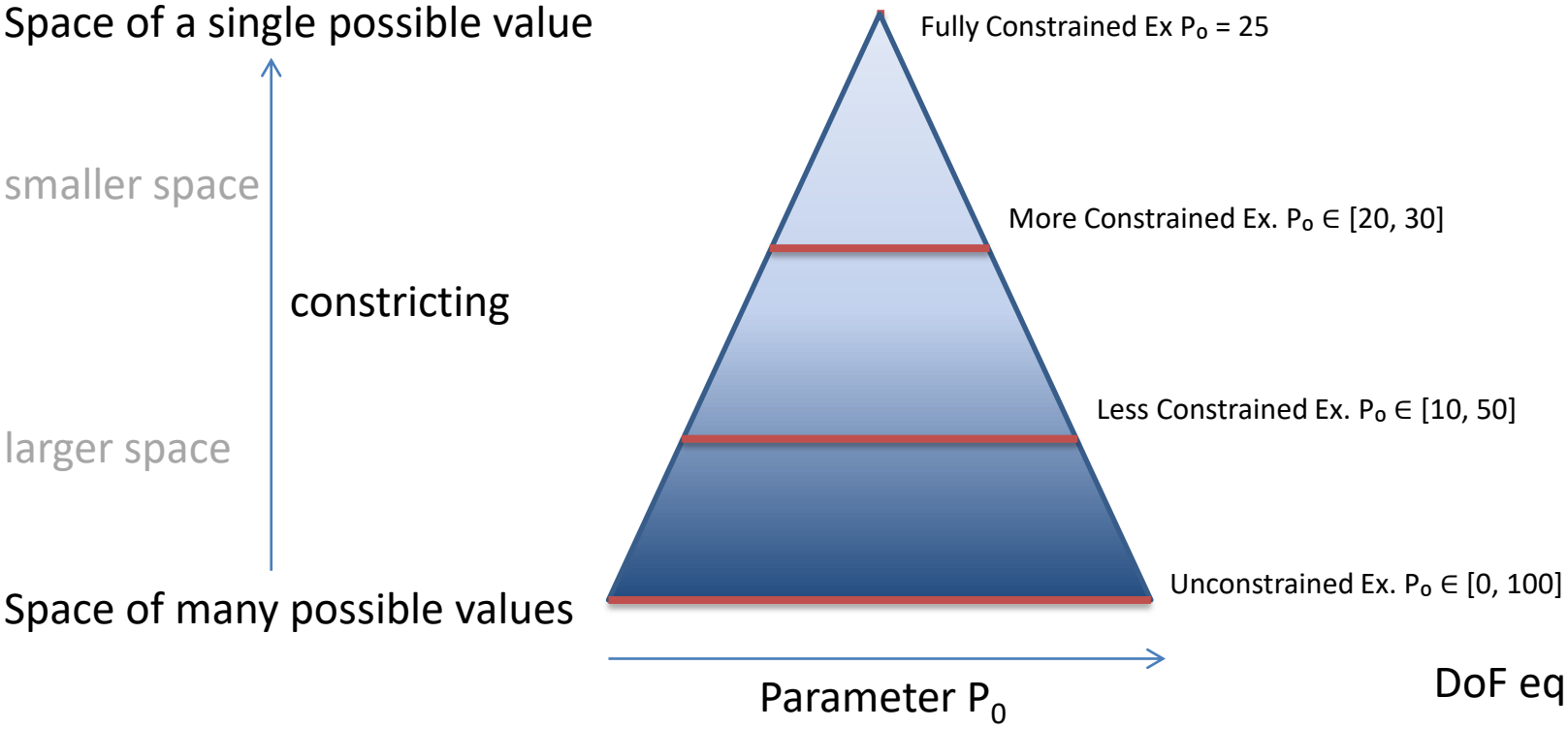



- Domain of P (D)
  - Real and Finite
  - Associated Capability/Constraint Space
    - May extend beyond the domain of P, spanning an infinite range (INF).
- Degrees of freedom of a set of P (DoF)
  - from 1 to infinite
    - NUL is allowed only during internal processing or as a result.

# Constraint/Capability Duality

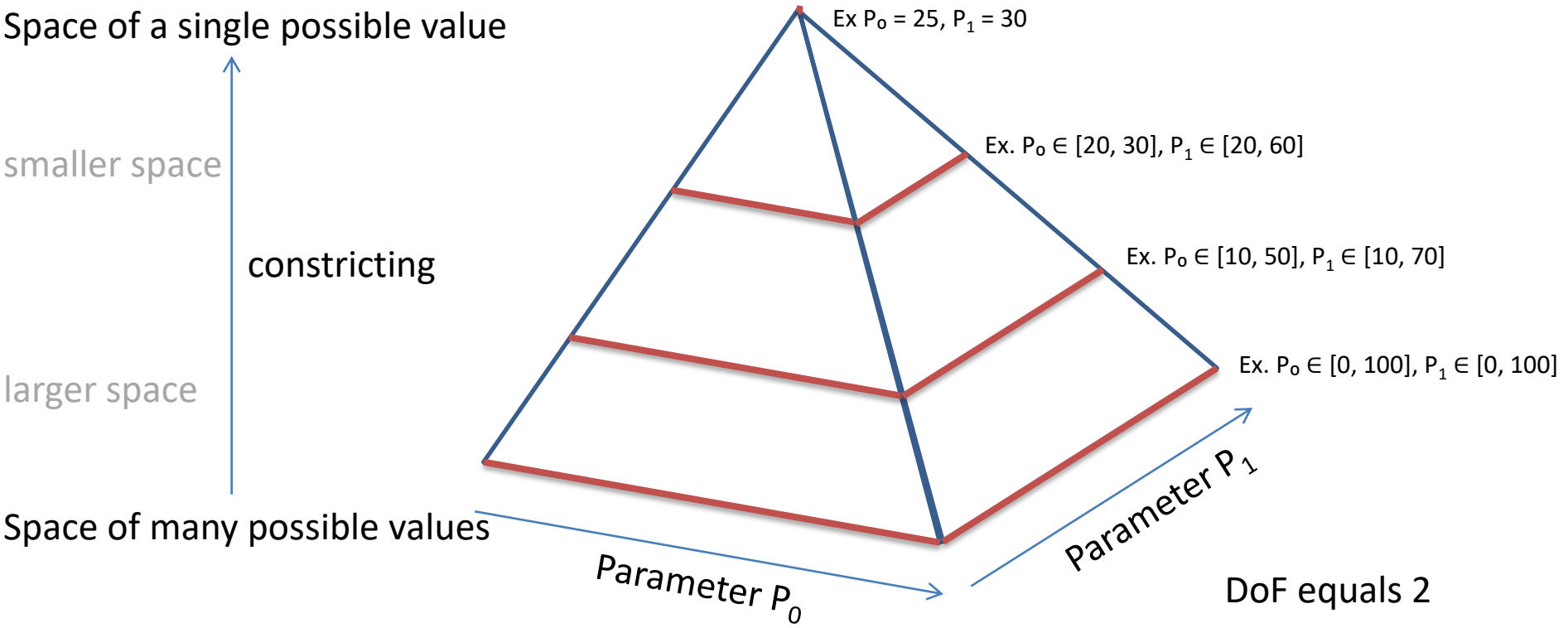
- A range describes either a Capability or a Constraint
  - Represents the Capability / Constraint Space of Parameter P.
- A range identity, Caps or Cons, is fixed at processing time
  - Infinite Range (INF):
    - As a **Capability**: Indicates the parameter supports any value.
    - As a **Constraint**: Implies no restriction is imposed on the parameter.

# Parameter Space Pyramid (2D)




DoF equals 1

# Parameter Space Pyramid (3D)



# Parameter Space Pyramid (nD)

- HyperPyramid of n dimensions
  - Representation for more than 2 parameters

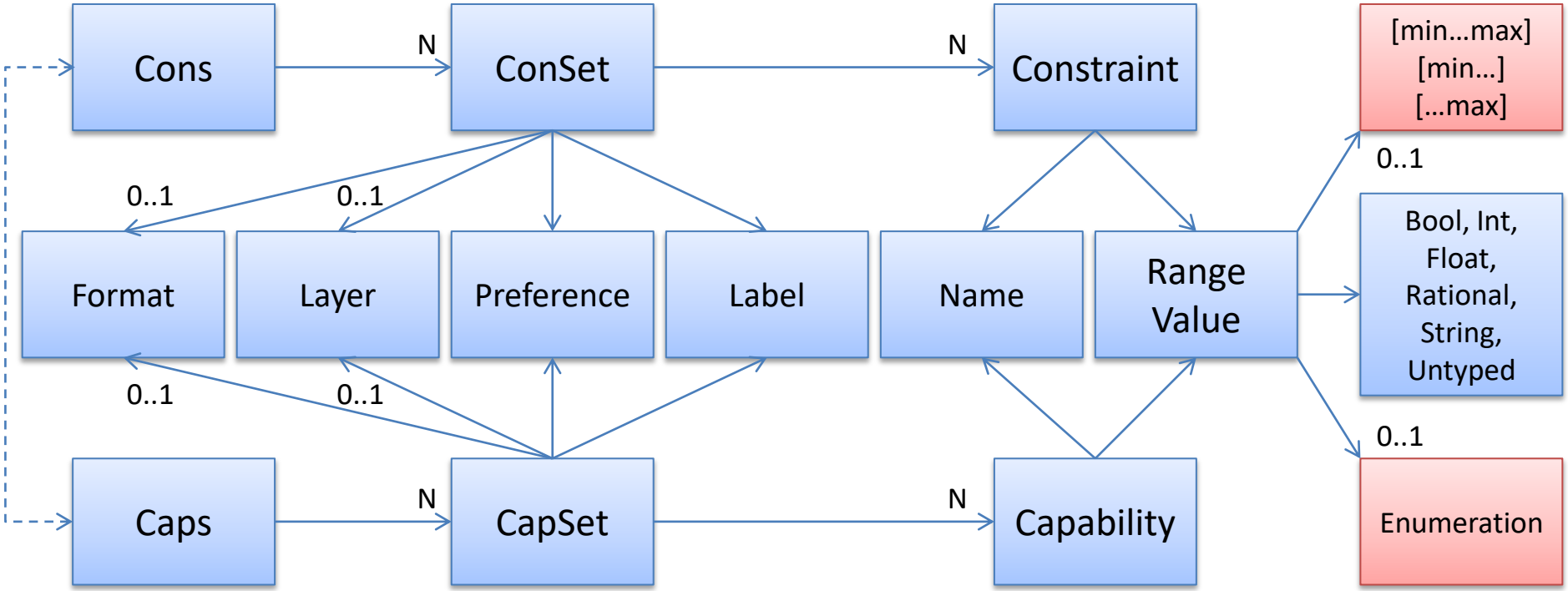
# Workshop Hands-On

- Python3.11.exe -i MatroxCCF.py

  >>>

- During this workshop we will experiment with CCF using the provided MatroxCCF.py module. At any time feel free to pause this presentation and experiment with the construct presented.

# Framework objects



# Framework Objects

- RangeType
  - BOOL, INT, FLOAT, RATIONAL, STRING, UNTYPED
- RangeValue
  - values: Optional[Tuple[Union[bool, int, float, Fraction, str], ...]] = None
  - min: Optional[Union[int, float, Fraction]] = None
  - max: Optional[Union[int, float, Fraction]] = None
  - infinite: bool = False
  - empty: bool = False
  - type: RangeType = UNTYPED

type coercion if not UNTYPED

type inference if UNTYPED

# Type inference / coercion

- If provided type is UNTYPED and not (INF or NUL)
  - Infer type from defined value among (in that order):
    - min
    - max
    - first enumerated value
  - Coerce min, max and values to inferred type
    - Exception: cannot coerce FLOAT/RATIONAL range to INT range
      - not same cardinality => float range has infinite number of values

# [min .. max] [ enumeration ]

- RangeValue support either or both constructs
  - if both are provided =>
    - an operation if performed against both spaces.
    - an operation must succeed against both spaces.
  - May be counter intuitive

# Workshop Hands-On

```
>>> print(RangeValue(infinite=True))
INF
>>> print(RangeValue(type=RangeType.FLOAT))
INFf
>>> print(RangeValue(empty=True))
NUL
>>> print(RangeValue(empty=True, type=RangeType.INT))
NULi
>>> print(RangeValue(min=0))
[0 ..]
>>> print(RangeValue(max=1))
[.. 1]
>>> print(RangeValue(min=0.0))
[0.0 ..]
>>> print(RangeValue(max=1.0))
[.. 1.0]
>>> print(RangeValue(max=1).type)
INT
>>> print(RangeValue(max=1.0).type)
FLOAT
>>> print(RangeValue(min=0.0, max=1.0))
[0.0 .. 1.0]
```

```
>>> print(RangeValue((0, 1, 2, 3, 4)))
[0, 1, 2, 3, 4]
>>> print(RangeValue(("x", "y", "z", "w")))
[w, x, y, z]
>>> print(RangeValue(("x", "y", "z", "w")).type)
STRING
>>> print(RangeValue((0, 1.0, 2.0, 3, 4)).type)
INT
>>> print(RangeValue((0, 1.5, 2.2, 3, 4)))
[0, 1, 2, 3, 4]
>>> print(RangeValue((True, False)))
[False, True]
>>> print(RangeValue((Fraction(30,1), Fraction(60000,1001))))
[60000/1001, 30]
>>> print(RangeValue(min=0,max=100, values=(1,2,3)))
[ 0 .. 100 1, 2, 3]
>>> print(RangeValue(min=0,max=100, values=()))
[ 0 .. 100 ]
```

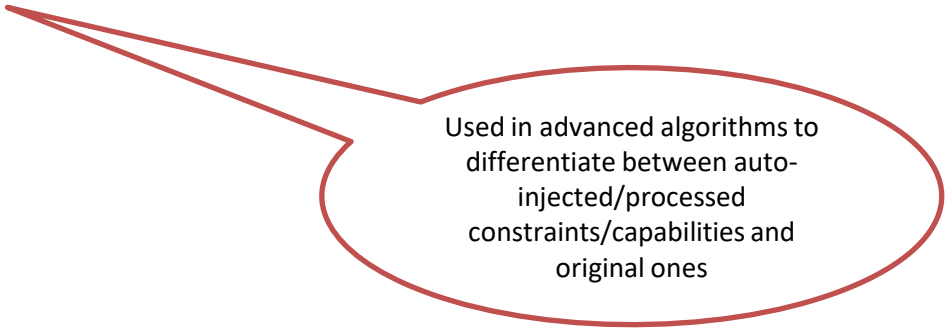
use either range or enum

do not use empty enum

```
>>> print(RangeValue(min=0,max=100, values=()).has_enum_exception())
True
```

# Framework Objects

- Capability / Constraint
  - name: str
  - value: RangeValue
  - original: bool = False



# Workshop Hands-On

```
>>> print(Cap("x", RangeValue()))
Cap(name='x', value=INF)
>>> print(Cap("axis_label", RangeValue(("x", "y", "z", "w"))))
Cap(name='axis_label', value=[y, z, x, w])
>>> print(Cap("x", RangeValue(type=RangeType.FLOAT)))
Cap(name='x', value=INFf)
>>> print(Cap("y", RangeValue(min=0.0, max=1.0)))
Cap(name='y', value=[ 0.0 .. 1.0 ])
>>> print(Cap("day", RangeValue(("Sun", "Mon", "Tue", "Wed", "Thu", "Fri",
        "Sat"))))
Cap(name='day', value=[Wed, Sat, Sun, Thu, Fri, Mon, Tue])
>>>
```

```
>>> print(Con("x", RangeValue()))
Con(name='x', value=INF)
>>> print(Con("axis_label", RangeValue(("x", "y", "z", "w"))))
Con(name='axis_label', value=[y, z, x, w])
>>> print(Con("day", RangeValue(("Sun", "Mon", "Tue", "Wed", "Thu", "Fri",
        "Sat"))))
Con(name='day', value=[Wed, Sat, Sun, Thu, Fri, Mon, Tue])
>>>
```

# Framework Objects

- CapSet / ConSet
  - caps: Dict[str, Capability / Constraint] = dict()
  - preference: int = 0
  - label: str = “”
  - format: Optional[str] = None
  - layer: Optional[int] = None
  - layer_compatibility_groups: Optional[List[int]] = None

# Workshop Hands-On

```
>>> print(CapSet(label="point4D", caps=make_capset(Capability("x",
        RangeValue(type=RangeType.FLOAT)), Capability("y",
        RangeValue(type=RangeType.FLOAT)), Capability("z",
        RangeValue(type=RangeType.FLOAT)), Capability("w",
        RangeValue(type=RangeType.FLOAT)))))
CapSet(
 label='point4D',
 preference=0,
 format=None,
 layer=None,
 layer_compatibility_groups=None,
 caps={
    Cap(name='x', value=INFf),
    Cap(name='y', value=INFf),
    Cap(name='z', value=INFf),
    Cap(name='w', value=INFf)
  }
)
>>>
```

```
>>> print(CapSet(label="point3D", caps=make_capset(Capability("x",
        RangeValue(type=RangeType.FLOAT)), Capability("y",
        RangeValue(type=RangeType.FLOAT)), Capability("z",
        RangeValue(type=RangeType.FLOAT)), Capability("w",
        RangeValue((1.0,))))))
CapSet(
 label='point3D',
 preference=0,
 format=None,
 layer=None,
 layer_compatibility_groups=None,
 caps={
    Cap(name='x', value=INFf),
    Cap(name='y', value=INFf),
    Cap(name='z', value=INFf),
    Cap(name='w', value=[1.0])
  }
)
```

# Workshop Hands-On

```
>>> print(CapSet(label="point3D", caps=make_capset(Capability("x",
        RangeValue(type=RangeType.FLOAT)), Capability("y",
        RangeValue(type=RangeType.FLOAT)), Capability("z",
        RangeValue(type=RangeType.FLOAT)))))
CapSet(
 label='point3D',
 preference=0,
 format=None,
 layer=None,
 layer_compatibility_groups=None,
 caps={
    Cap(name='x', value=INFf),
    Cap(name='y', value=INFf),
    Cap(name='z', value=INFf)
  }
)
>>>
```

```
>>> print(CapSet(label="date", caps=make_capset(Capability("day",
        RangeValue(min=1, max=31)), Capability("month", RangeValue(("jan",
        "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec",))),
        Capability("year", RangeValue(min=1)))))
CapSet(
 label='date',
 preference=0,
 format=None,
 layer=None,
 layer_compatibility_groups=None,
 caps={
    Cap(name='day', value=[ 1 .. 31 ]),
    Cap(name='month', value=[feb, may, apr, mar, jun, sep, oct, dec, nov, aug, jul,
        jan]),
    Cap(name='year', value=[ 1 .. ])
  }
)
```

Not 100% precise

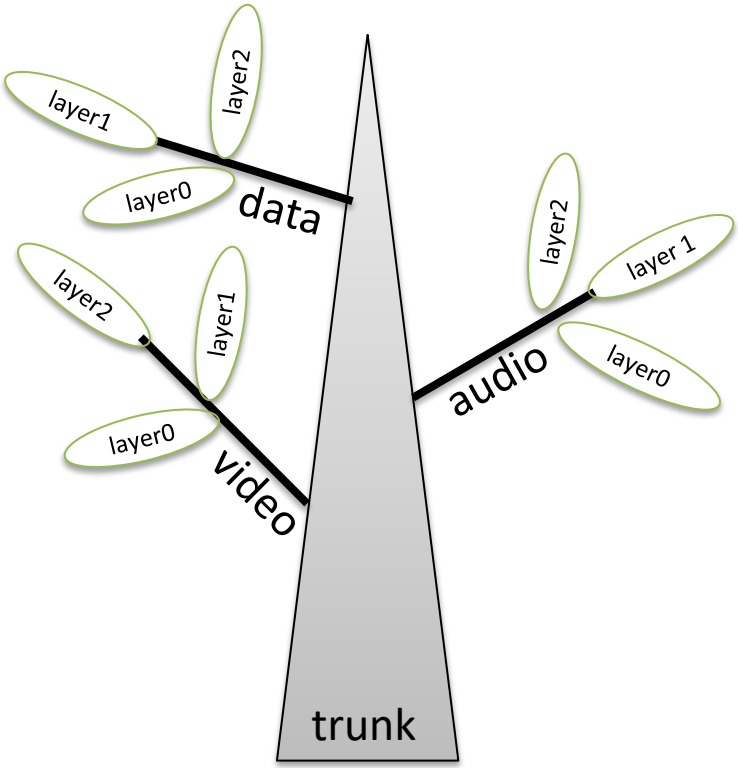
# Framework Objects

- Caps
  - capsets: List[CapSet] = list()
  - filtered: bool = False
- Cons
  - consets: List[ConSet] = list()
  - filtered: bool = False

Caps ←→ Cons conversion

Change the interpretation to be given to unspecified parameters. If “unspecified” is to mean “may take any value” then Caps. If “unspecified” is to mean “don’t care” then Cons.

# Multiple Parts



# Workshop Hands-On

```
>>> print(Caps(capsets=[CapSet(label="date_case1",
        caps=make_capset(Capability("day", RangeValue(min=1, max=29)),
        Capability("month", RangeValue(("feb",))), Capability("year",
        RangeValue(min=1)))), CapSet(label="date_case2",
        caps=make_capset(Capability("day", RangeValue(min=1, max=30)),
        Capability("month", RangeValue(("apr", "jun", "sep", "nov",))),
        Capability("year", RangeValue(min=1))),CapSet(label="date_case3",
        caps=make_capset(Capability("day", RangeValue(min=1, max=31)),
        Capability("month", RangeValue(("jan", "mar", "may", "jul", "aug", "oct",
        "dec",))), Capability("year", RangeValue(min=1)))]))
```

More precise means more verbose …
still not considering leap years here.

```
Caps(
 filtered=False,
 capsets=[
  CapSet(
   label='date_case1',
   preference=0,
   format=None,
   layer=None,
   layer_compatibility_groups=None,
   caps={
     Cap(name='day', value=[ 1 .. 29 ]),
     Cap(name='month', value=[feb]),
     Cap(name='year', value=[ 1 .. ])
    } ),
  CapSet(
   label='date_case2',
   preference=0,
   format=None,
   layer=None,
   layer_compatibility_groups=None,
   caps={
     Cap(name='day', value=[ 1 .. 30 ]),
     Cap(name='month', value=[nov, sep, apr, jun]),
     Cap(name='year', value=[ 1 .. ])
    } ),
  CapSet(
   label='date_case3',
   preference=0,
   format=None,
   layer=None,
   layer_compatibility_groups=None,
   caps={
     Cap(name='day', value=[ 1 .. 31 ]),
     Cap(name='month', value=[may, mar, dec, oct, aug, jul, jan]),
     Cap(name='year', value=[ 1 .. ])
    } )
 ])
```

# Framework Operators

- Inclusion x <= y
- Inheritance x <- y
- Constriction x << y
- Constriction with adjustemnt x <& y
- Intersection x & y

Type must be the same or UNTYPED, otherwise an exception is raised

# Inclusion (x <= y) → bool

| x | y | |
|---|---|---|
| Caps | Caps | caps_included_in_caps(x, y) |
| CapSet | Caps | capset_included_in_caps(x, y) |
| CapSet | CapSet | capset_included_in_capset(x, y) |
| Cap | Cap | cap_included_in_cap(x, y) |

| x | y | |
|---|---|---|
| Cons | Cons | cons_included_in_con(x, y) |
| ConSet | Cons | conset_included_in_cons(x, y) |
| ConSet | ConSet | conset_included_in_conset(x, y) |
| Con | Con | con_included_in_con(x, y) |

# Inclusion (x <= y) → bool

| x | y | |
|---|---|---|
| Cons | Caps | cons_included_in_caps(x, y) |
| ConSet | Caps | conset_included_in_caps(x, y) |
| ConSet | CapSet | conset_included_in_capset(x, y) |
| Con | Cap | con_included_in_cap(x, y) |

| x | y | |
|---|---|---|
| RangeValue | RanveValue | range_included_in_range(x, y) |
| value | RangeValue | value_included_in_range(x, y) |
| Namespace | Namespace | namespace_included_in_namespace(x, y) |
| | | |

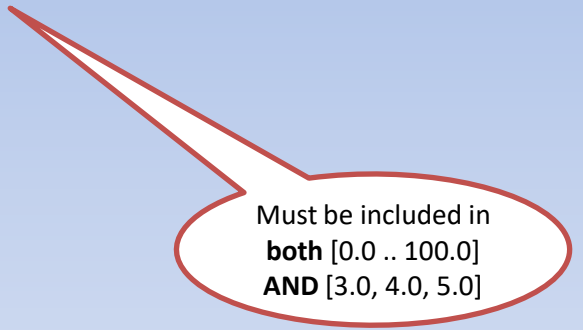
# Workshop Hands-On

```
>>> date_caps=Caps(capsets=[CapSet(label="date_case1",
        caps=make_capset(Capability("day", RangeValue(min=1, max=29)),
        Capability("month", RangeValue(("feb",))), Capability("year",
        RangeValue(min=1)))), CapSet(label="date_case2",
        caps=make_capset(Capability("day", RangeValue(min=1, max=30)),
        Capability("month", RangeValue(("apr", "jun", "sep", "nov",))),
        Capability("year", RangeValue(min=1))),CapSet(label="date_case3",
        caps=make_capset(Capability("day", RangeValue(min=1, max=31)),
        Capability("month", RangeValue(("jan", "mar", "may", "jul", "aug", "oct",
        "dec",))), Capability("year", RangeValue(min=1)))])

>>> date_capset_feb28=CapSet(label="date_case1",
        caps=make_capset(Capability("day", RangeValue(min=1, max=29)),
        Capability("month", RangeValue(("feb",))), Capability("year",
        RangeValue(min=1))))

>>> capset_included_in_caps(date_capset_feb28, date_caps)
True

>>> print(namespace_inherit_from_capset(Namespace(), CapSet(label="date",
        caps=make_capset(Capability("day", RangeValue(min=1, max=31)),
        Capability("month", RangeValue(("jan", "feb", "mar", "apr", "may", "jun",
        "jul", "aug", "sep", "oct", "nov", "dec",))), Capability("year",
        RangeValue(min=1))))))
{'year', 'month', 'day'}

>>> unprecise_date_capset=CapSet(label="date",
        caps=make_capset(Capability("day", RangeValue(min=1, max=31)),
        Capability("month", RangeValue(("jan", "feb", "mar", "apr", "may", "jun",
        "jul", "aug", "sep", "oct", "nov", "dec",))), Capability("year",
        RangeValue(min=1))))
>>> capset_included_in_caps(unprecise_date_capset, date_caps)
False
>>>
```

# Workshop Hands-On

```
>>> value_included_in_range("foo", RangeValue(infinite=True,
        type=RangeType.FLOAT))
ValueError: Type mismatch: FLOAT vs foo
>>> value_included_in_range("foo", RangeValue(infinite=True,
        type=RangeType.UNTYPED))
True
>>> value_included_in_range("foo", RangeValue(infinite=True))
True
>>> value_included_in_range("foo", RangeValue(("foo", "bar")))
True
>>> value_included_in_range("fool", RangeValue(("foo", "bar")))
False
>>> value_included_in_range("1", RangeValue(type=RangeType.STRING,
        values=(1, 2, 3, 4, 5)))
True
>>> value_included_in_range("1", RangeValue(type=RangeType.STRING, min=1,
        max=10))
ValueError: range cannot be of types STRING or BOOL
>>>
```

```
>>> value_included_in_range(1.0, RangeValue(min=0.0, max=100.0))
True

>>> value_included_in_range(1.0, RangeValue(min=0.0, max=100.0,
        values=(3,4,5)))
False
```



# Inheritance (x <- y) → typeof(x)

| x | y | |
|---|---|---|
| Namespace | Caps | namespace_inherit_from_caps(x, y) |
| Namespace | CapSet | namespace_inherit_from_capset(x, y) |
| Namespace | Cap | namespace_inherit_from_cap(x, y) |
| Namespace | name | namespace_inherit_from_name(x, y) |

| x | y | |
|---|---|---|
| Namespace | Cons | namespace_inherit_from_cons(x, y) |
| Namespace | ConSet | namespace_inherit_from_conset(x, y) |
| Namespace | Con | namespace_inherit_from_con(x, y) |
| | | |

# Inheritance (x <- y) → typeof(x)

| x | y | |
|---|---|---|
| CapSet | Namespace | capset_inherit_from_namespace(x, y) |
| CapSet | name | capset_inherit_from_name(x, y) |
| CapSet | CapSet | capset_inherit_from_capset(x, y) |
| CapSet | Cap | capset_inherit_from_cap(x, y) |

| x | y | |
|---|---|---|
| ConSet | Namespace | conset_inherit_from_namespace(x, y) |
| ConSet | name | conset_inherit_from_name(x, y) |
| Conset | ConSet | conset_inherit_from_conset(x, y) |
| Conset | Con | conset_inherit_from_con(x, y) |

# Workshop Hands-On

```
>>> print(namespace_inherit_from_name(Namespace(), "foo"))
{'foo'}

>>> print(namespace_inherit_from_name(Namespace("bar"), "foo"))
{'bar', 'foo'}

>>> print(namespace_inherit_from_cap(Namespace(), Capability(name="x",
        value=RangeValue())))
{'x'}

>>> print(capset_inherit_from_namespace(CapSet(label="inherited"),
        Namespace("x", "y", "z")))
CapSet(
  label='inherited',
  preference=0,
  format=None,
  layer=None,
  layer_compatibility_groups=None,
  caps={
     Cap(name='y', value=INF),
     Cap(name='z', value=INF),
     Cap(name='x', value=INF)
   })
```

# Constriction (x << y) → typeof(x)

| x | y | |
|---|---|---|
| Caps | Cons | caps_constrict_by_cons(x, y) |
| Caps | ConSet | caps_constrict_by_conset(x, y) |
| CapSet | ConSet | capset_constrict_by_conset(x, y) |
| CapSet | Con | capset_constrict_by_con(x, y) |
| Cap | Con | cap_constrict_by_con(x, y) |

# Workshop Hands-On

```
>>> point4D=CapSet(label="point4D", caps=make_capset(Capability("x",
        RangeValue(type=RangeType.FLOAT)), Capability("y",
        RangeValue(type=RangeType.FLOAT)), Capability("z",
        RangeValue(type=RangeType.FLOAT)), Capability("w",
        RangeValue(type=RangeType.FLOAT))))

>>> print(capset_constrict_by_con(point4D, Constraint("w", RangeValue((1.0,)))))
CapSet(
 label='point4D',
 preference=0,
 format=None,
 layer=None,
 layer_compatibility_groups=None,
 caps={
    Cap(name='x', value=INFf),
    Cap(name='y', value=INFf),
    Cap(name='z', value=INFf),
    Cap(name='w', value=[1.0])
  }
)
```

```
>>> point4D=CapSet(label="point4D", caps=make_capset(Capability("x",
        RangeValue(type=RangeType.FLOAT)), Capability("y",
        RangeValue(type=RangeType.FLOAT)), Capability("z",
        RangeValue(type=RangeType.FLOAT)), Capability("w",
        RangeValue(type=RangeType.FLOAT))))
>>> point3Dnorm=ConSet(label="point4D", cons=make_conset(Constraint("x",
        RangeValue(min=-1.0, max=1.0)), Constraint("y", RangeValue(min=-1.0,
        max=1.0)), Constraint("z", RangeValue(min=-1.0, max=1.0)),
        Constraint("w", RangeValue((1.0,)))))
>>> print(capset_constrict_by_conset(point4D, point3Dnorm))
CapSet(
 label='point4D',
 preference=0,
 format=None,
 layer=None,
 layer_compatibility_groups=None,
 caps={
    Cap(name='x', value=[ -1.0 .. 1.0 ]),
    Cap(name='y', value=[ -1.0 .. 1.0 ]),
    Cap(name='z', value=[ -1.0 .. 1.0 ]),
    Cap(name='w', value=[1.0])
  }
)
>>>
```

# Workshop Hands-On

```
>>> point4D=CapSet(label="point4D", caps=make_capset(Capability("x",
        RangeValue(type=RangeType.FLOAT)), Capability("y",
        RangeValue(type=RangeType.FLOAT)), Capability("z",
        RangeValue(type=RangeType.FLOAT)), Capability("w",
        RangeValue(type=RangeType.FLOAT))))
>>> print(capset_constrict_by_con(point4D, Constraint("w", RangeValue())))
CapSet(
  label='point4D',
  preference=0,
  format=None,
  layer=None,
  layer_compatibility_groups=None,
  caps={
     Cap(name='x', value=INFf),
     Cap(name='y', value=INFf),
     Cap(name='z', value=INFf),
     Cap(name='w', value=INFf)
    }
)
>>>
```

```
>>> point4D=CapSet(label="point4D", caps=make_capset(Capability("x",
        RangeValue(type=RangeType.FLOAT)), Capability("y",
        RangeValue(type=RangeType.FLOAT)), Capability("z",
        RangeValue(type=RangeType.FLOAT)), Capability("w",
        RangeValue(type=RangeType.FLOAT))))

>>> print(capset_constrict_by_con(point4D, Constraint("w", RangeValue((1.0,)))))
CapSet(
  label='point4D',
  preference=0,
  format=None,
  layer=None,
  layer_compatibility_groups=None,
  caps={
     Cap(name='x', value=INFf),
     Cap(name='y', value=INFf),
     Cap(name='z', value=INFf),
     Cap(name='w', value=[1.0])
    }
)
```

# Constriction with adj (x <& y) → typeof(x)

| x | y | |
|---|---|---|
| Caps | Cons | caps_constrict_adjust_by_cons(x, y) |
| Caps | ConSet | caps_constrict_adjust_by_conset(x, y) |
| CapSet | ConSet | capset_constrict_adjust_by_conset(x, y) |
| CapSet | Con | capset_constrict_adjust_by_con(x, y) |
| Cap | Con | cap_constrict_adjust_by_con(x, y) |

# Workshop Hands-On

# Intersection (x & y) → RangeValue

| x | y | |
|---|---|---|
| RangeValue | RangeValue | range_intersection(x, y) |
| | | |
| | | |
| | | |
| | | |

# Sender JSON → Caps

| x | y | |
|---|---|---|
| Dict[str, Any] | | convert_caps_json_to_caps(x) |

Takes the JSON definition of a Sender or a Receiver and convert the ‘caps’ attribute into a Caps object.

# Conversion and Filtering

| x | y | |
|---|---|---|
| CapSet | | x.to_conset() → ConSet |
| ConSet | | x.to_capset() → CapSet |
| Caps | | x.get(filter) → Caps<br>Filter by format, layer, **compatibility_group**, media_types |
| Caps | | x.get_capset(filter, index) → CapSet<br>Filter by format, layer, **compatibility_group**, media_types |
| Cons | | x.get(filter) → Cons<br>Filter by format, layer, **compatibility_group**, media_types |
| Cons | | x.get_conset(filter, index) → ConSet<br>Filter by format, layer, **compatibility_group**, media_types |

# Normalizing

| x | y | |
|---|---|---|
| Caps | | x.normalize(<br>audio_layers: Optional[int] = None,<br>video_layers: Optional[int] = None,<br>data_layers: Optional[int] = None,<br>trunk_namespace: Optional[Set[str]] = None,<br>audio_namespace: Optional[Set[str]] = None,<br>video_namespace: Optional[Set[str]] = None,<br>data_namespace: Optional[Set[str]] = None,<br><br>) → Caps |

# Normalizing

| x | y | |
|---|---|---|
| Cons | | x.normalize(<br>audio_layers: Optional[int] = None,<br>video_layers: Optional[int] = None,<br>data_layers: Optional[int] = None,<br>trunk_namespace: Optional[Set[str]] = None,<br>audio_namespace: Optional[Set[str]] = None,<br>video_namespace: Optional[Set[str]] = None,<br>data_namespace: Optional[Set[str]] = None,<br><br>) → Cons |

# Part 2

- Using the framework with NMOS
  - Receiver Capabilities
  - Sender Capabilities
  - Sender Constraints
  - Controller Capabilities intersection
    - Multiple Receivers for a given Sender

# CapSet and ConSet ‘preference’

- preference of value 100 **should be** special
  - ^ tip of the parameter space pyramid
    - Represent **native** capabilities
      - One possible value per parameter (for 99% of the parameters)
    - Represent **preferred** constraints
      - First to be considered

There may be exceptional cases where some parameters of a native set are not fixed

- This concludes our hands-on session. In this workshop, we explored how to analyze NMOS capabilities using Matrox CCF for complex workflows like NDI and MPEG2-TS.
- If you have any questions, feel free to reach out at abouchar@matrox.com.
- Thank you for attending.