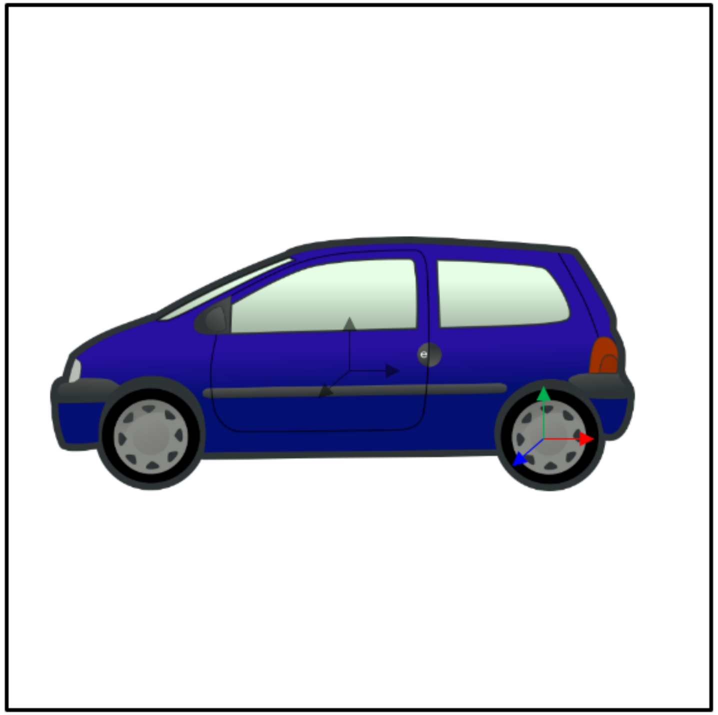
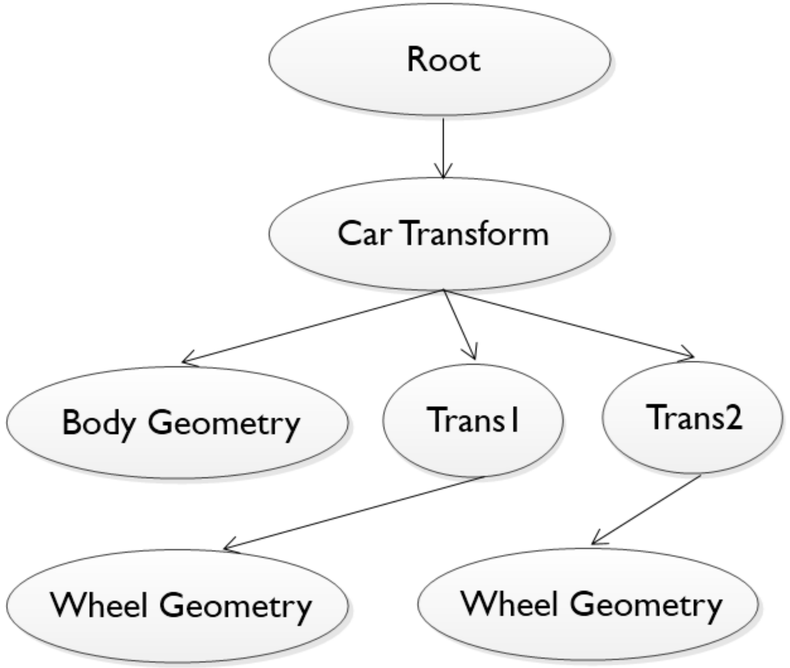
The Field Concept and Dependency Graphs

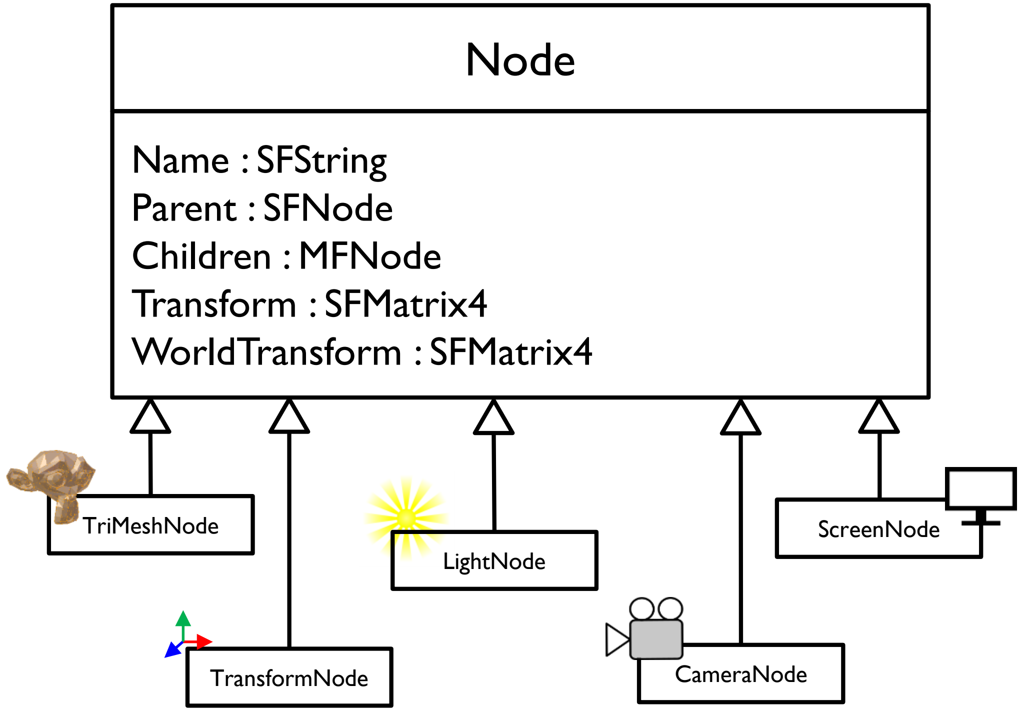
Tim Weißker

Last Week: Scenegraphs





Last Week: Scenegraphs

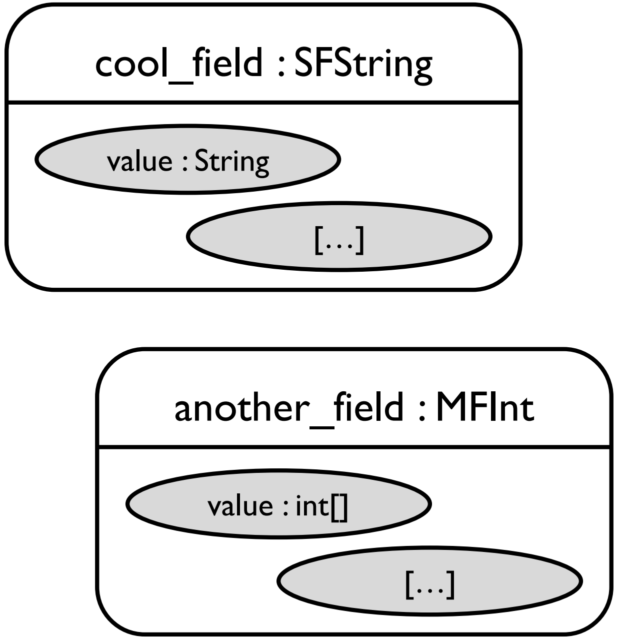


Every node in the scenegraph is a *field container*!

Fields and Field Containers

Field

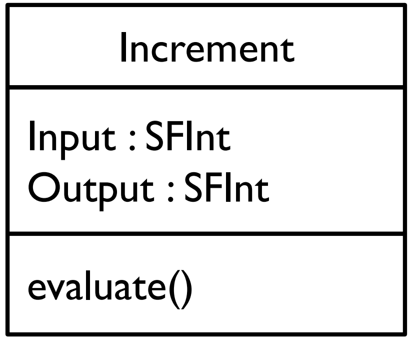
- more complex form of an attribute
- object state information
- single and multi fields
- easy serialization and distribution



Fields and Field Containers

Field Container

- collection of fields
- evaluate() method called when one field changes



Fields and Field Containers

Field Container

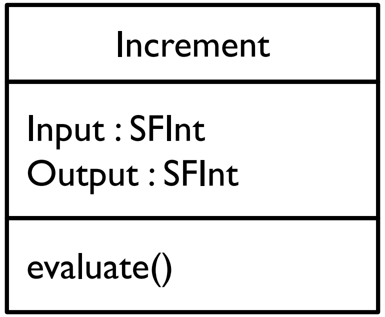
- collection of fields
- evaluate() method called when one field changes

```
class Increment(avango.script.Script):

  Input = avango.SFInt()
  Output = avango.SFInt()

  def evaluate(self):
    self.Output.value = self.Input.value + 1
```

Evaluation

```
def evaluate(self):
    self.Output.value = self.Input.value + 1
```

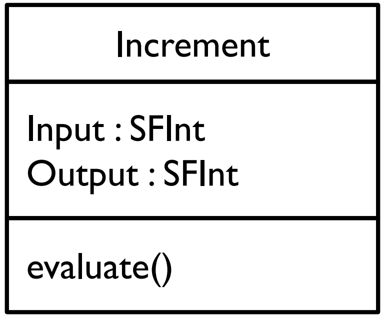
- a field container's `evaluate()` method executes three steps in the following order



Evaluation

```
def evaluate(self):
    self.Output.value = self.Input.value + 1
```

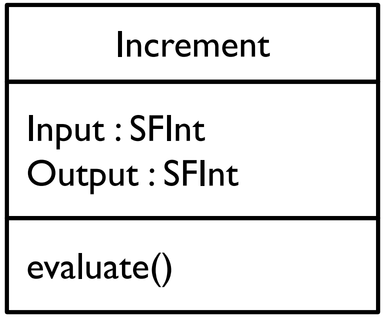
- a field container's `evaluate()` method executes three steps in the following order
  - read values from local input fields



Evaluation

```
def evaluate(self):
    self.Output.value = self.Input.value + 1
```

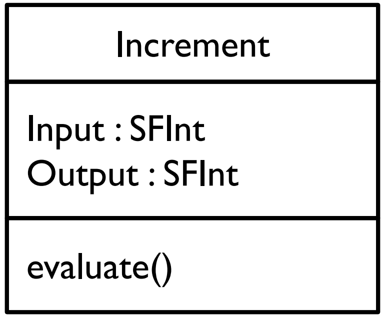
- a field container's `evaluate()` method executes three steps in the following order
  - read values from local input fields
  - calculate new values derived from these values



Evaluation

```
def evaluate(self):
    self.Output.value = self.Input.value + 1
```

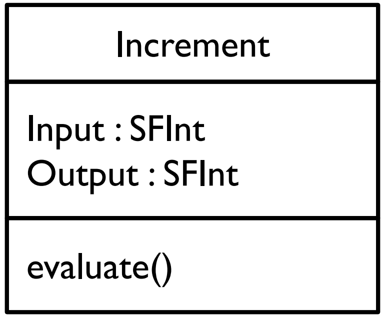
- a field container's `evaluate()` method executes three steps in the following order
  - read values from local input fields
  - calculate new values derived from these values
  - write the results to output fields



Evaluation

```
def evaluate(self):
    self.Output.value = self.Input.value + 1
```

- a field container's `evaluate()` method executes three steps in the following order
  - read values from local input fields
  - calculate new values derived from these values
  - write the results to output fields



- to increase reuse, no external data should be accessed

Implementing a Field Container

```
class Container(avango.script.Script):
```

Implementing a Field Container

```
class Container(avango.script.Script):

  #declaration of fields, e.g.
  sf_mat = avango.gua.SFMatrix4()
```

Implementing a Field Container

```
class Container(avango.script.Script):

  #declaration of fields, e.g.
  sf_mat = avango.gua.SFMatrix4()

  def __init__(self):
    self.super(Container).__init__()
```

Implementing a Field Container

```
class Container(avango.script.Script):

  #declaration of fields, e.g.
  sf_mat = avango.gua.SFMatrix4()

  def __init__(self):
    self.super(Container).__init__()

  def my_constructor(self, PARAMETER1, PARAMETER2, …):
    #initialize variables, parameters, etc.
```

Implementing a Field Container

```
class Container(avango.script.Script):

  #declaration of fields, e.g.
  sf_mat = avango.gua.SFMatrix4()

  def __init__(self):
    self.super(Container).__init__()

  def my_constructor(self, PARAMETER1, PARAMETER2, …):
    #initialize variables, parameters, etc.

  def evaluate(self):
    #perform update when fields change
```

Evaluation policies

```
class Container(avango.script.Script):

  #declaration of fields, e.g.
  sf_mat = avango.gua.SFMatrix4()

  def __init__(self):
    self.super(Container).__init__()

  def my_constructor(self, PARAMETER1, PARAMETER2, …):
    #initialize variables, parameters, etc.

  def evaluate(self):
    #perform update when fields change
```

Evaluation policies

- `evaluate()`
  - called when at least one of the field changes

Evaluation policies

```
class Container(avango.script.Script):

  #declaration of fields, e.g.
  sf_mat = avango.gua.SFMatrix4()

  def __init__(self):
    self.super(Container).__init__()
    self.always_evaluate(True)

  def my_constructor(self, PARAMETER1, PARAMETER2, …):
    #initialize variables, parameters, etc.

  def evaluate(self):
    #perform update when fields change
```

Evaluation policies

- `evaluate()`
  - called when at least one of the field changes
- `self.always_evaluate(True)`
  - forces evaluation every frame regardless of field changes

Evaluation policies

```
class Container(avango.script.Script):

  #declaration of fields, e.g.
  sf_mat = avango.gua.SFMatrix4()

  def __init__(self):
    self.super(Container).__init__()


  def my_constructor(self, PARAMETER1, PARAMETER2, …):
    #initialize variables, parameters, etc.

  @field_has_changed(sf_mat)
  def sf_mat_changed(self):
    #perform update when fields change
```

Evaluation policies

- `evaluate()`
  - called when at least one of the field changes
- `self.always_evaluate(True)`
  - forces evaluation every frame regardless of field changes
- `@field_has_changed(SFFoo)`
  - only evaluated when SFFoo changes
  - function name can be arbitrary

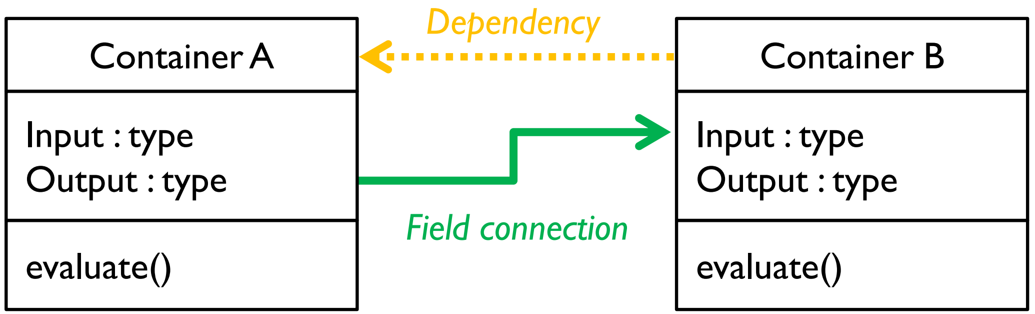
Dependencies

- sometimes it is the case that an output field of one field container forms the input of another one (e.g. sensors)

Container A
Input : type
Output : type
evaluate()
Container B
Input : type
Output : type
evaluate()

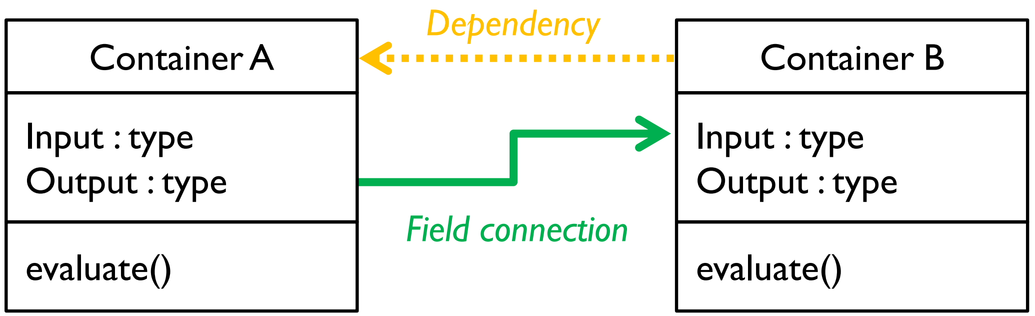
Dependencies

- sometimes it is the case that an output field of one field container forms the input of another one (e.g. sensors)
- field connections: the value of a field is copied into another one after evaluation



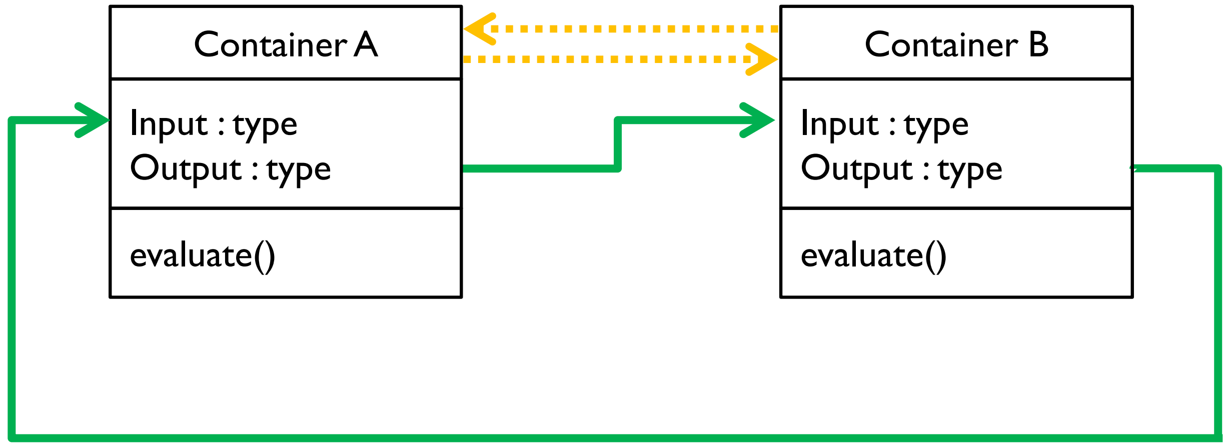
Dependencies

- sometimes it is the case that an output field of one field container forms the input of another one (e.g. sensors)
- field connections: the value of a field is copied into another one after evaluation

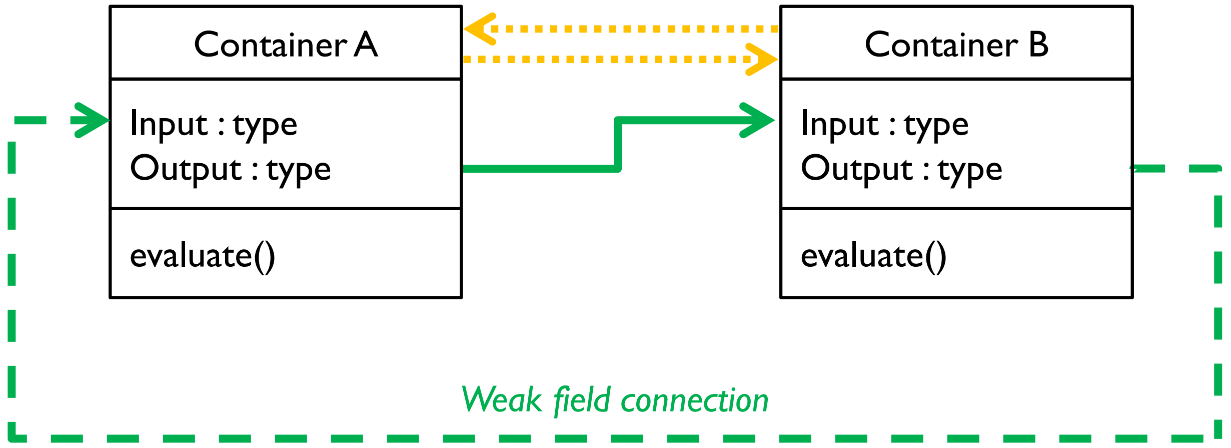


- `b.Input.connect_from(a.Output)` (no .value needed)

Cyclic dependencies

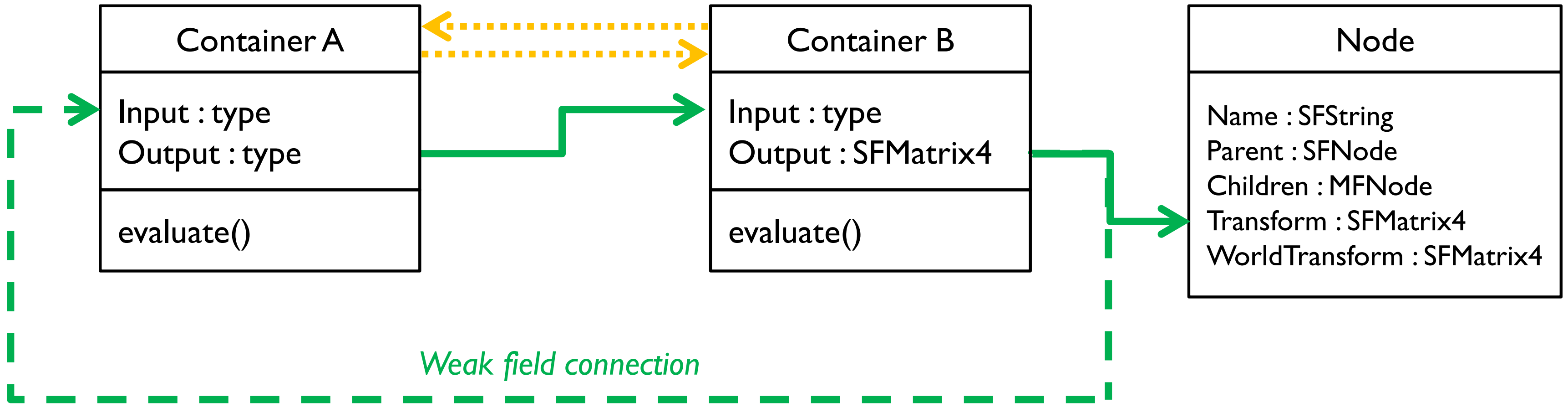


Cyclic dependencies

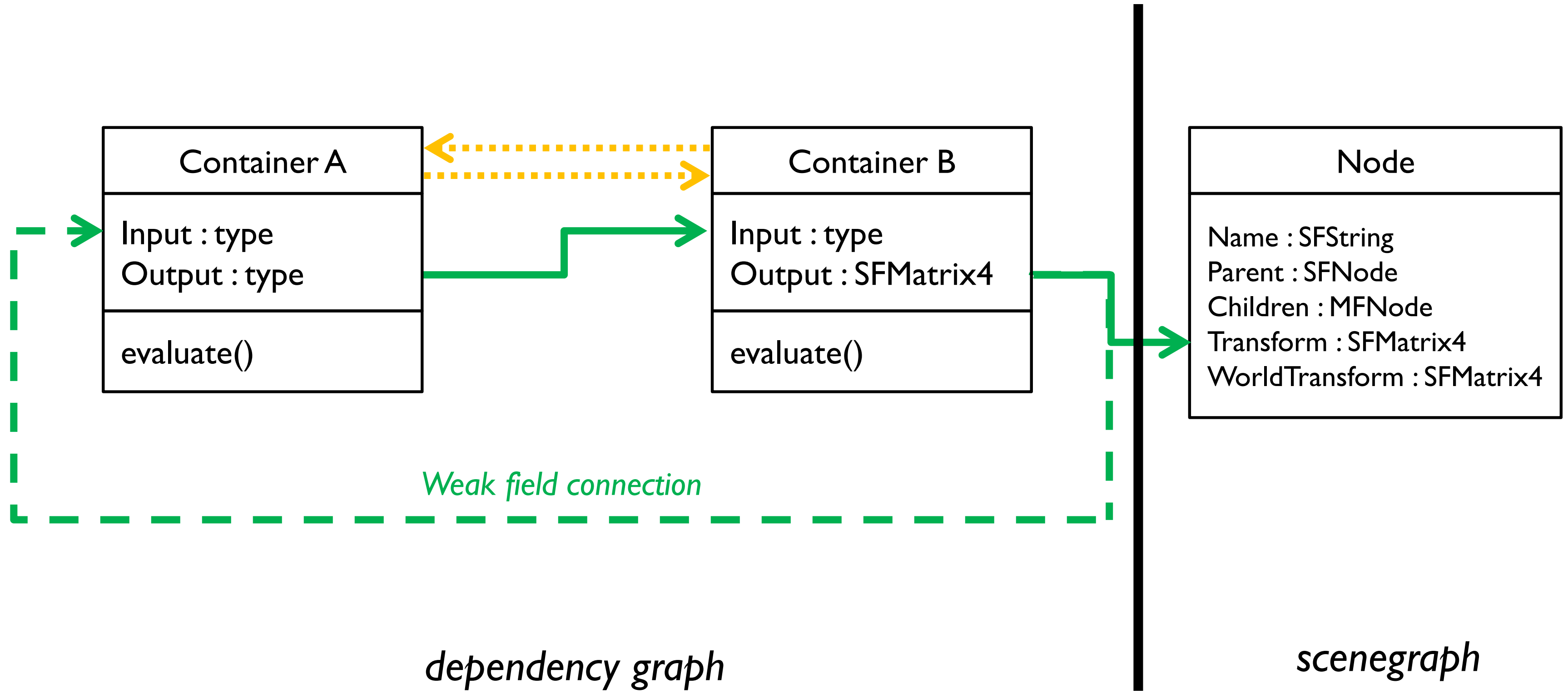


- `a.Input.connect_weak_from(b.Output)`

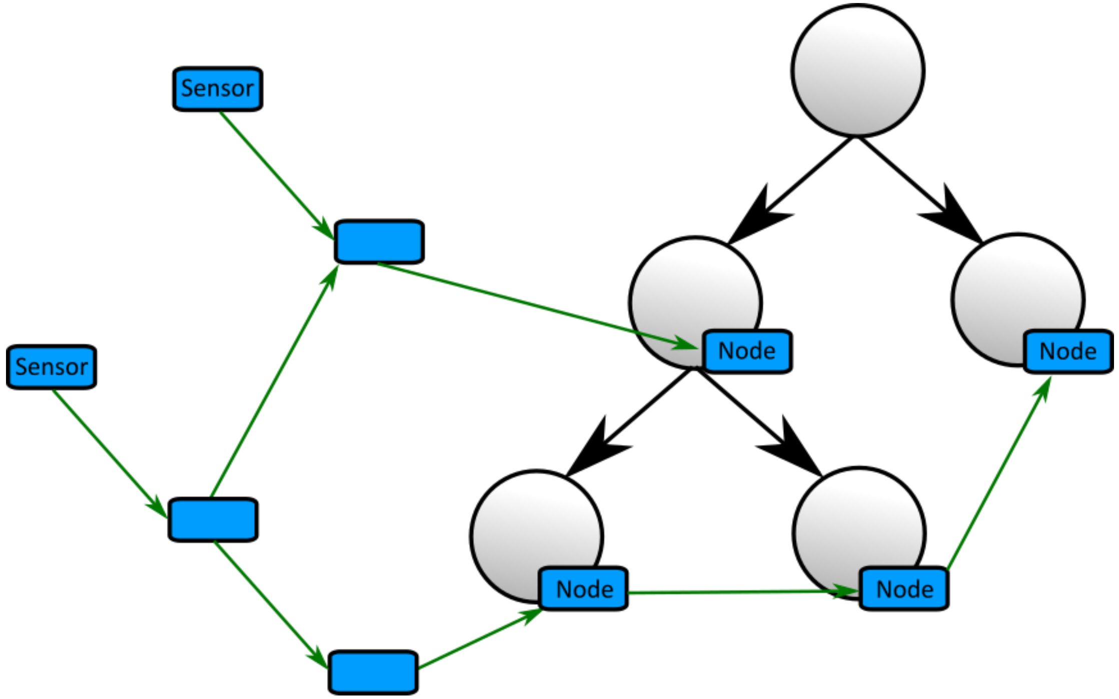
Orthogonality



Orthogonality



Orthogonality



Summary

- every node of the Scenegraph and classes derived from `avango.script.Script` are called field containers

Summary

- every node of the Scenegraph and classes derived from `avango.script.Script` are called field containers
- state of a field container is defined by state of its fields
  - easy serialization
  - easy network distribution

Summary

- every node of the Scenegraph and classes derived from `avango.script.Script` are called field containers
- state of a field container is defined by state of its fields
  - easy serialization
  - easy network distribution
- within a field container, no external data should be accessed
  - realize dependencies with field connections
  - loose coupling between field container instances (easily reusable)

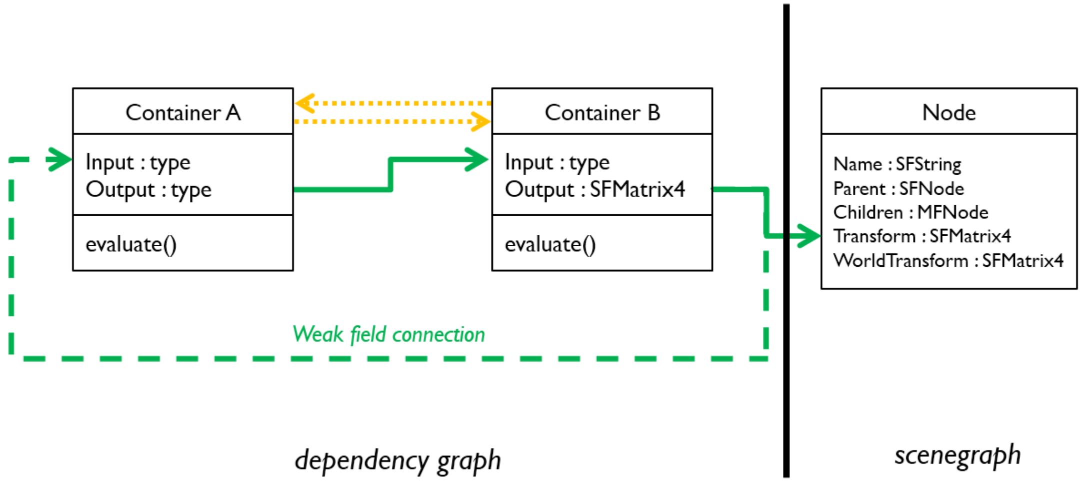
Summary

- every node of the Scenegraph and classes derived from `avango.script.Script` are called field containers
- state of a field container is defined by state of its fields
  - easy serialization
  - easy network distribution
- within a field container, no external data should be accessed
  - realize dependencies with field connections
  - loose coupling between field container instances (easily reusable)
- the dependency graph is orthogonal to the scenegraph

The Field Concept

```
class Container(avango.script.Script):

  #declaration of fields, e.g.
  sf_mat = avango.gua.SFMatrix4()

  def __init__(self):
    self.super(Container).__init__()

 def my_constructor(self, PARAMETER1, PARAMETER2, …):
    #initialize variables, parameters, etc.

  def evaluate(self):
    #perform update when fields change
```

Questions?


Container A
Input : type
Output : type
evaluate()
Container B
Input : type
Output : SFMatrix4
evaluate()
Node
Name : SFString
Parent : SFNode
Children : MFNode
Transform : SFMatrix4
WorldTransform : SFMatrix4
Weak field connection
dependency graph
scenegraph