Virtual Reality

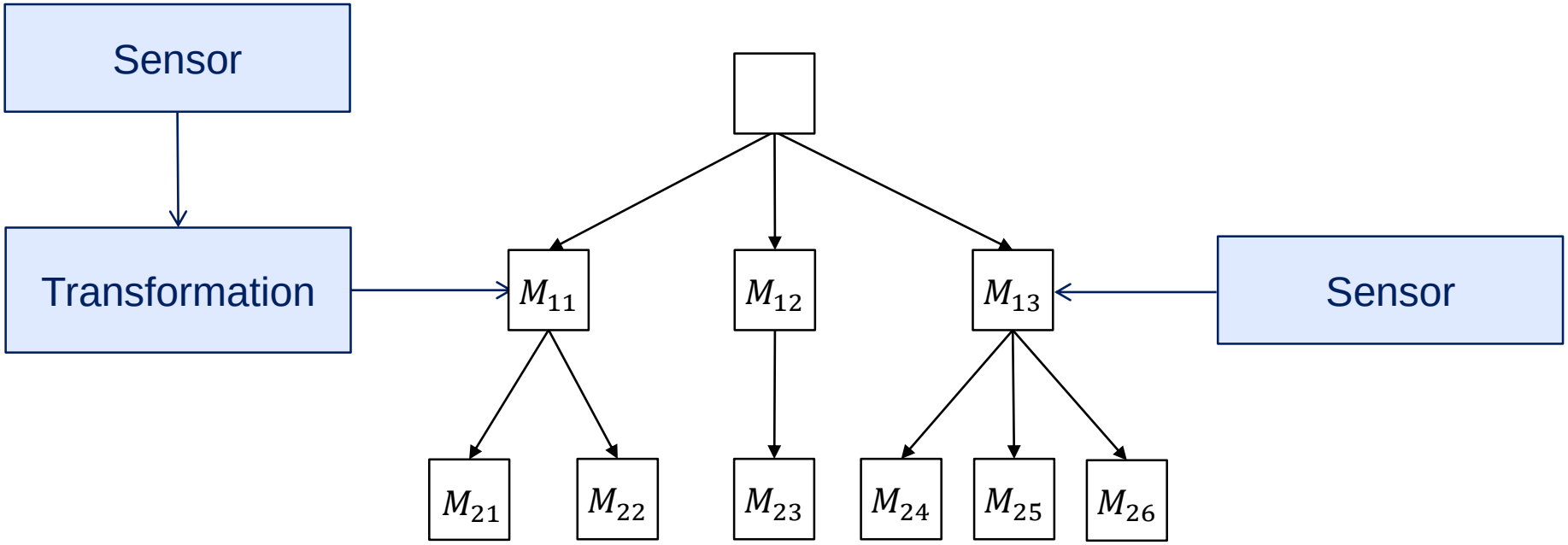
Lab Class – Dependency Graphs

WS 2018/19

André Kunert
Tim Weißker

Bauhaus-Universität Weimar
Virtual Reality and Visualization Research

Data Flow Programming



Field Container

- Collection of fields
- `evaluate()` method called when one field changes
- Every node in the scenegraph is a field container, but not every field container needs to be a scenegraph node

Increment
Input: SFInt Output: SFInt
evaluate()

```
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):

  #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
```

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 Mechanisms

- `evaluate()`
  - called when any field changes
  - example: dynamic update depending on variable input data

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__()
    self.always_evaluate(True)

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

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

Evaluation Mechanisms

- `evaluate()`
  - called when any field changes
  - example: dynamic update depending on variable input data

- `self.always_evaluate(True)`
  - forces evaluation every frame regardless of field changes
  - example: frame-based updates, animations

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.

  @field_has_changed(sf_mat)
  def sf_mat_changed(self):
    #perform update when field changes
```

Evaluation Mechanisms

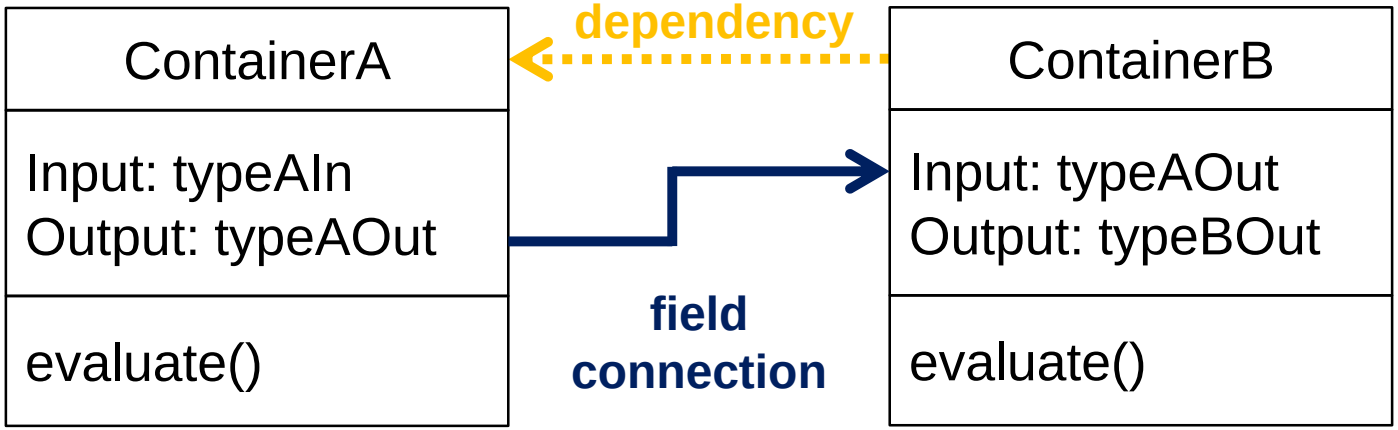
- `evaluate()`
  - called when any field changes
  - example: dynamic update depending on variable input data

- `self.always_evaluate(True)`
  - forces evaluation every frame regardless of field changes
  - example: frame-based updates, animations

- `@field_has_changed(SFFoo)`
  - only evaluated when SFFoo changes
  - function name can be arbitrary
  - example: button events

Dependencies

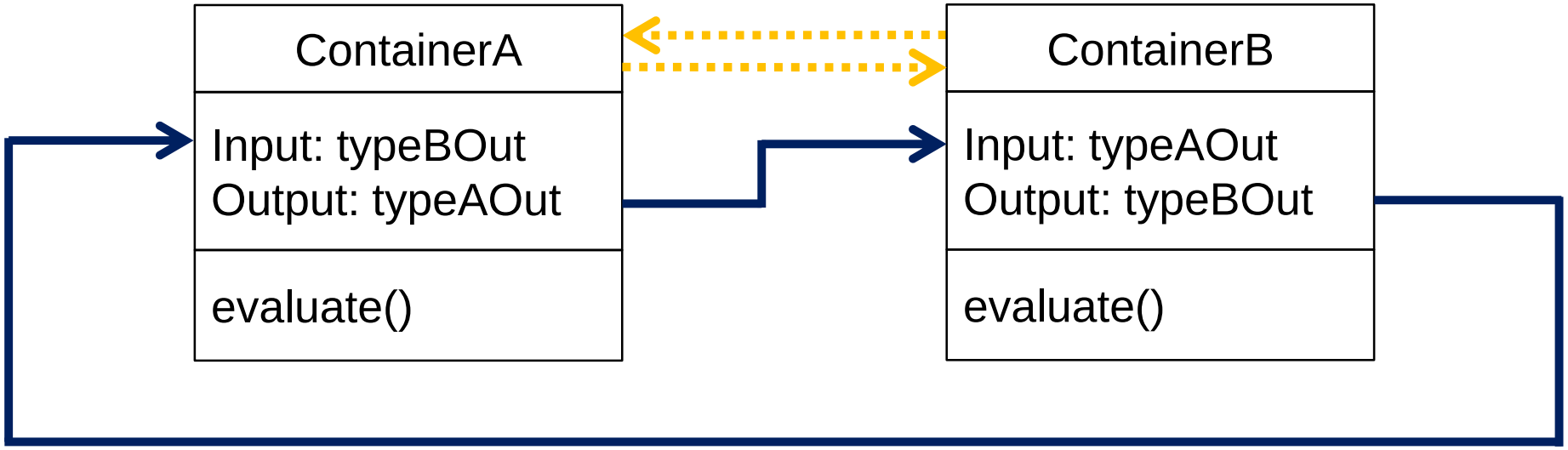
- Each field container should take care of a single responsibility
- Data flow between field containers by field connections
- Field connection: the value of a field is copied into another one after evaluation



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

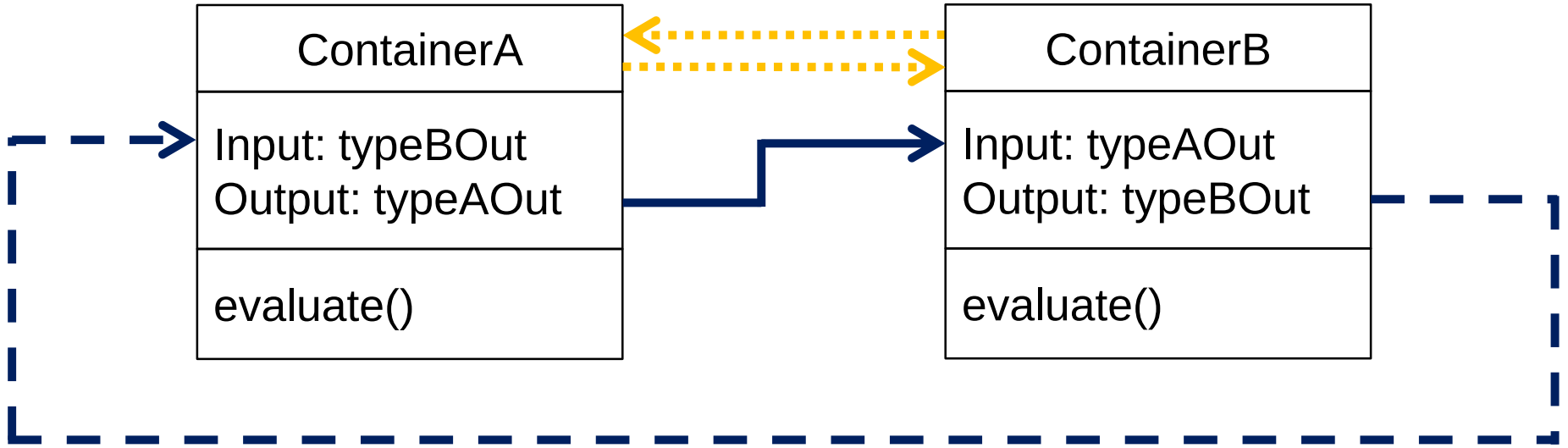
Cyclic Dependencies

- Resolving dependencies results in an infinite loop, which is broken up at an indefinite point



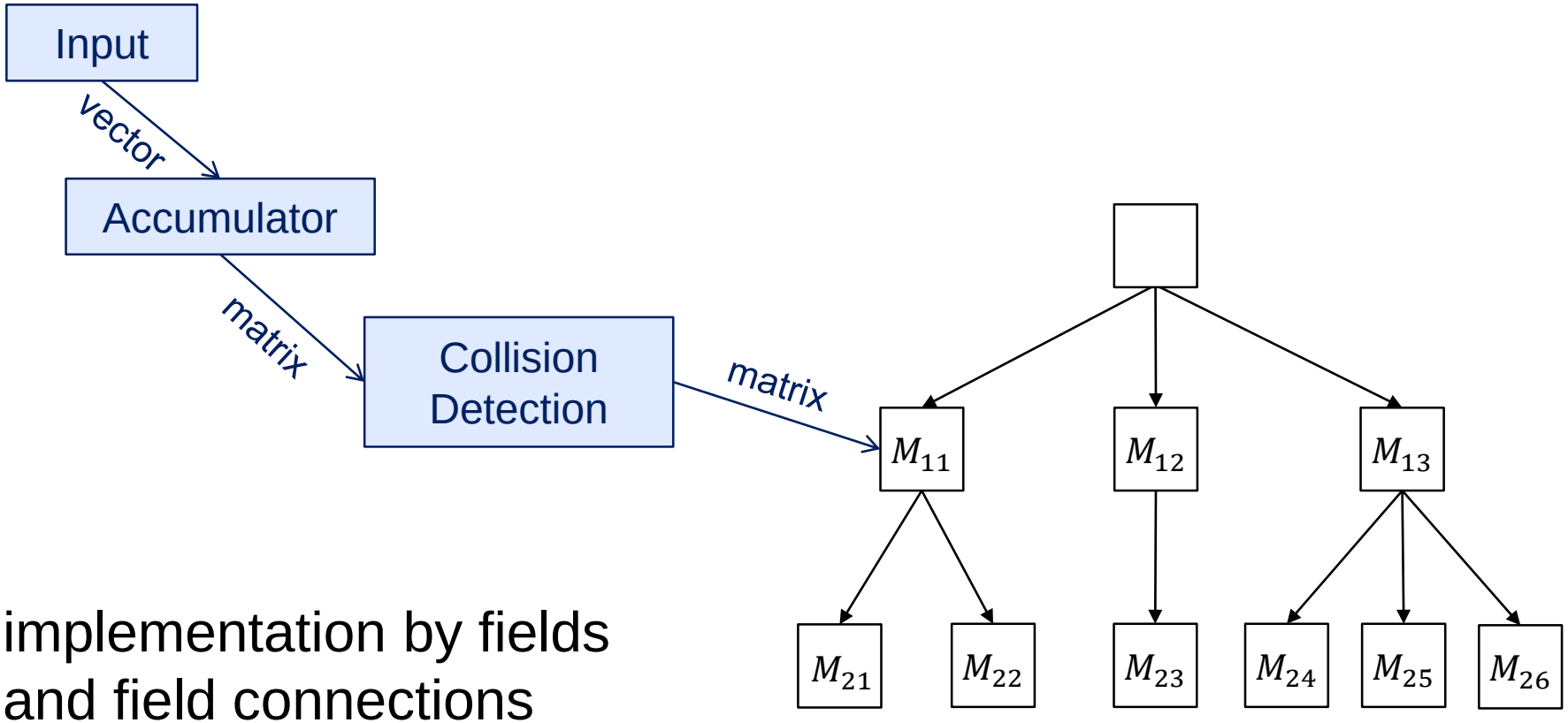
Cyclic Dependencies

- Weak field connection: ignored during dependency resolving; still, the source value will be copied to the sink at the end of the frame



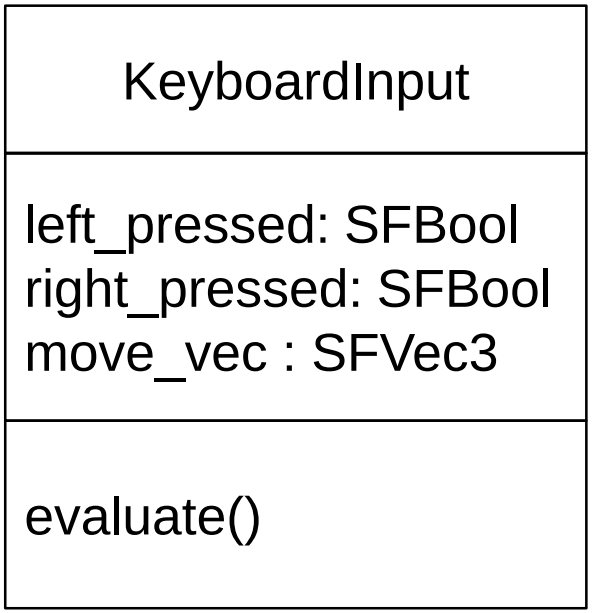
- `a.Input.connect_weak_from(b.Output)`

Example

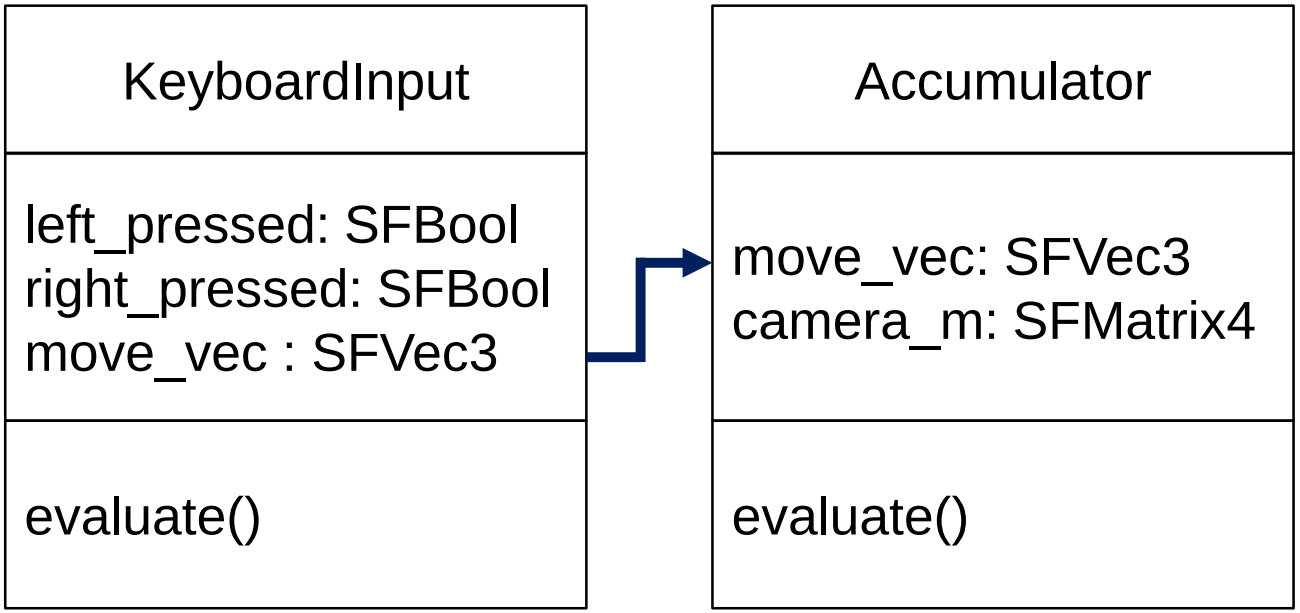


implementation by fields and field connections

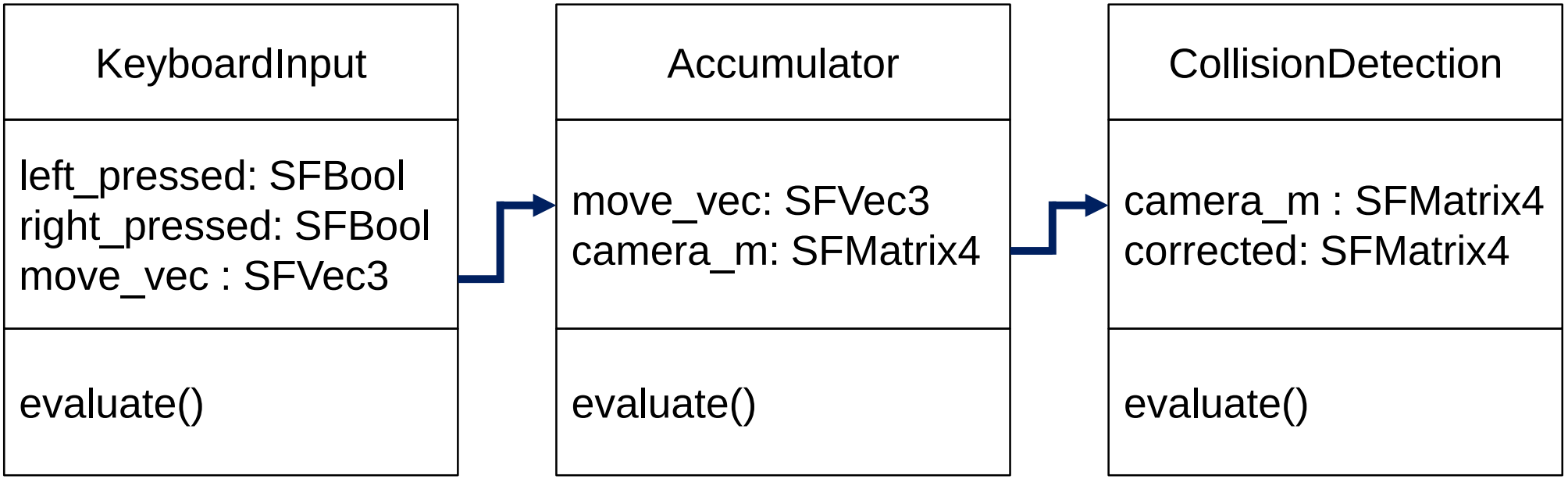
Example


KeyboardInput
left_pressed: SFBool
right_pressed: SFBool
move_vec : SFVec3
evaluate()

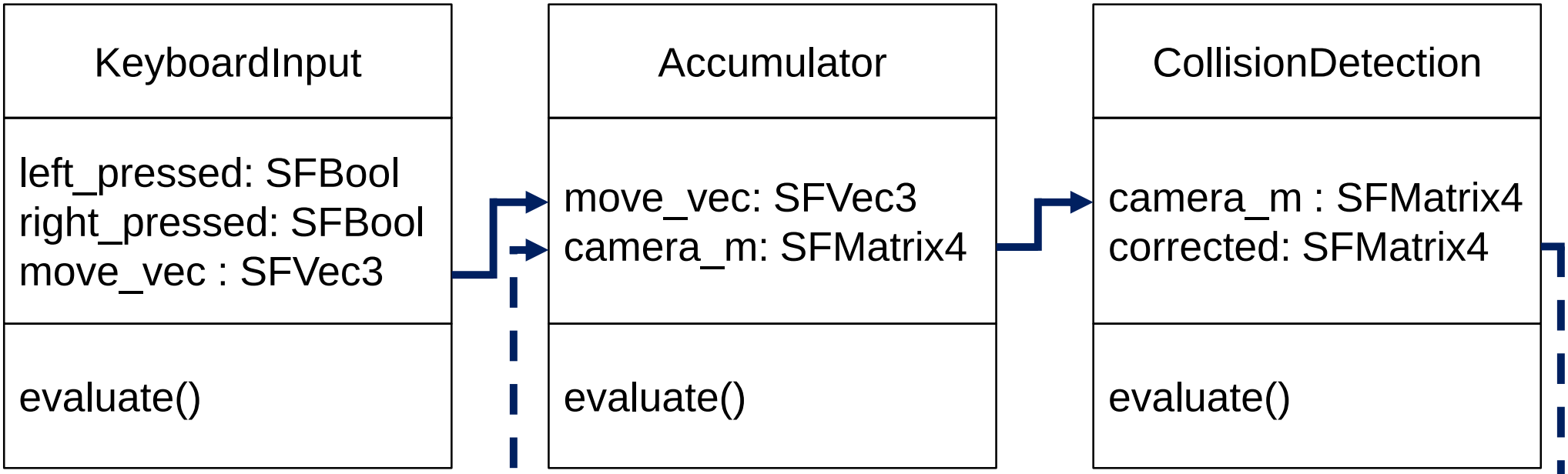
Example


KeyboardInput
left_pressed: SFBool
right_pressed: SFBool
move_vec : SFVec3
evaluate()
Accumulator
move_vec: SFVec3
camera_m: SFMatrix4
evaluate()

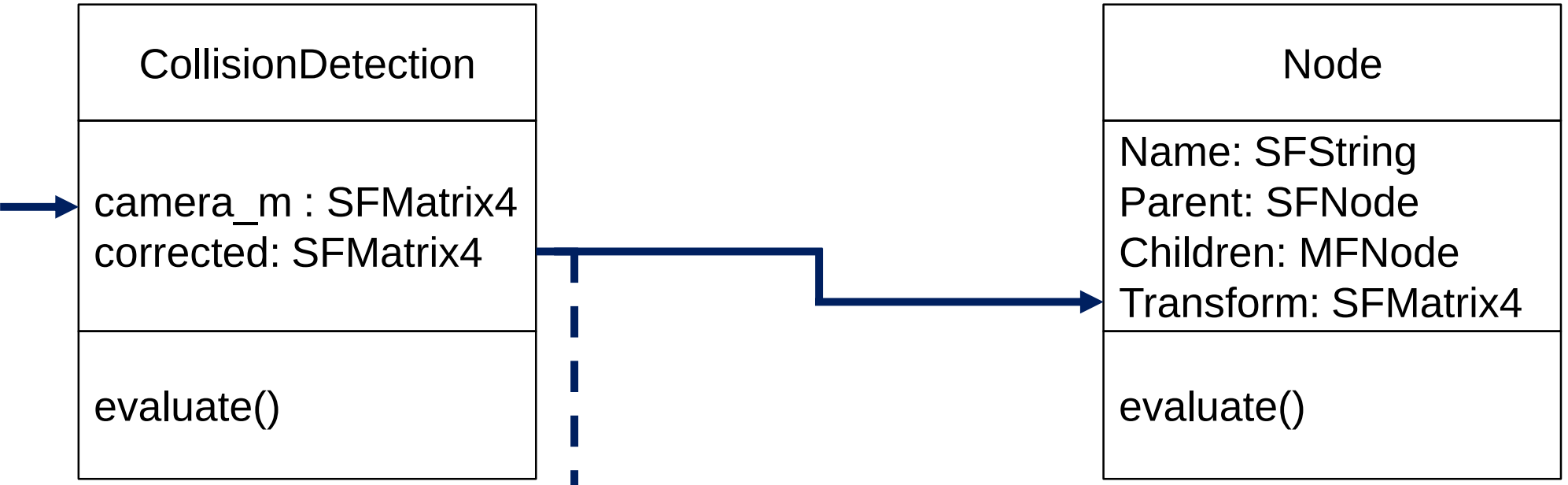
Example


KeyboardInput
left_pressed: SFBool
right_pressed: SFBool
move_vec : SFVec3
evaluate()
Accumulator
move_vec: SFVec3
camera_m: SFMatrix4
evaluate()
CollisionDetection
camera_m : SFMatrix4
corrected: SFMatrix4
evaluate()

Example


KeyboardInput
left_pressed: SFBool
right_pressed: SFBool
move_vec : SFVec3
evaluate()
Accumulator
move_vec: SFVec3
camera_m: SFMatrix4
evaluate()
CollisionDetection
camera_m : SFMatrix4
corrected: SFMatrix4
evaluate()

Orthogonality



dependency graph

scenegraph