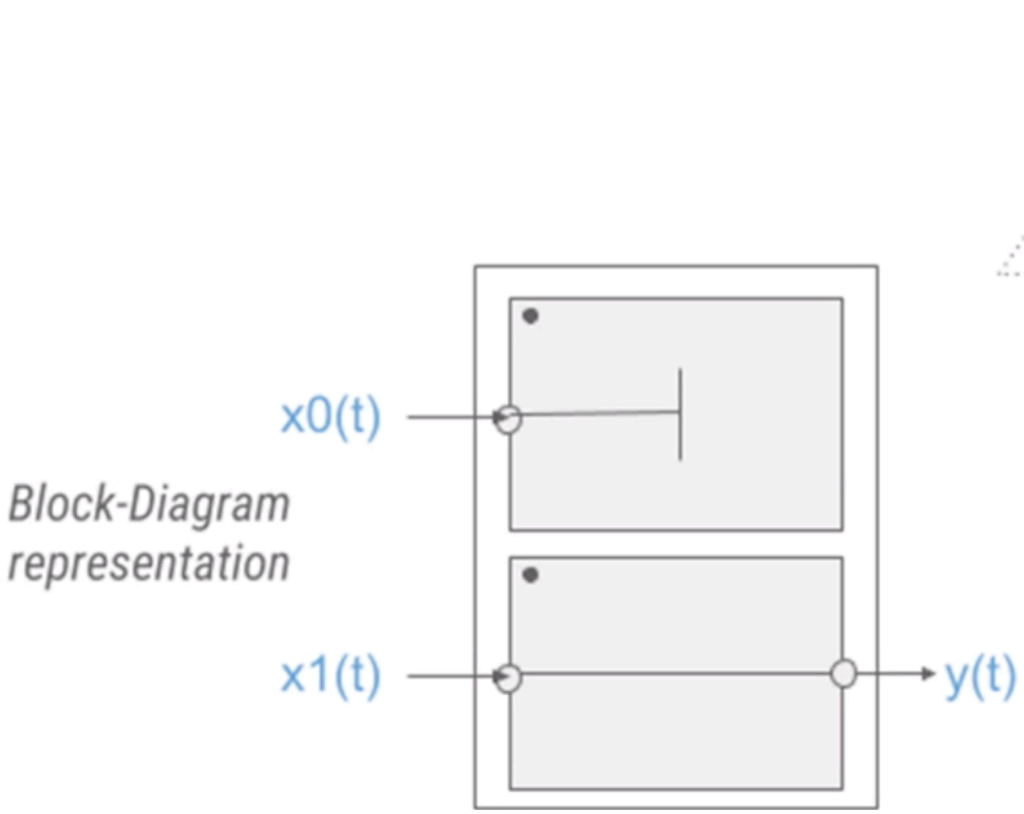
Faust Code

```
process = !,_;
```



Block-Diagram representation

y(t) = x1(t), x0(t) ignored

Mathematical Semantics

Faust Code (Syntax)

```
process = +;
```

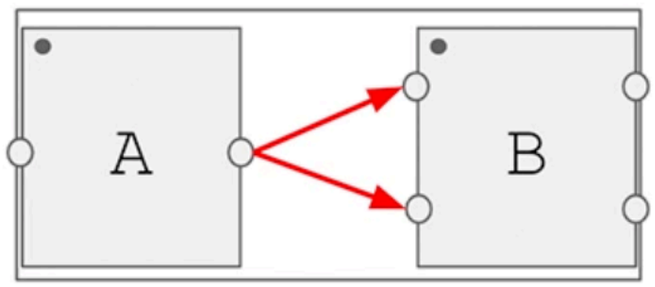
Block-Diagram representation

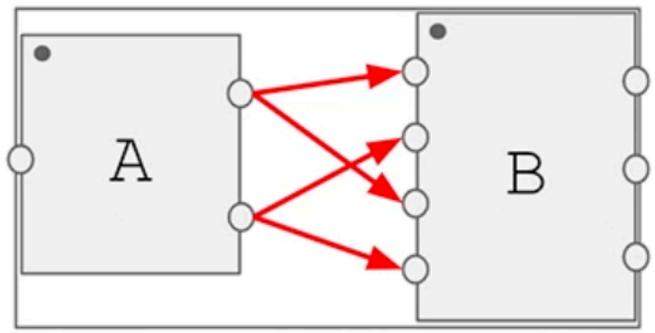
$$y(t) = x0(t)+x1(t)$$

Mathematical Semantics

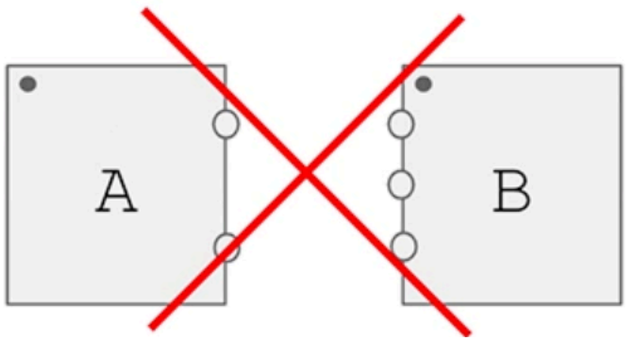
Faust Code

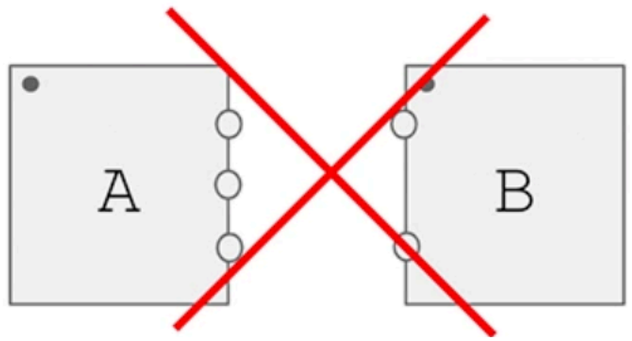
```
process = A <: B;
```





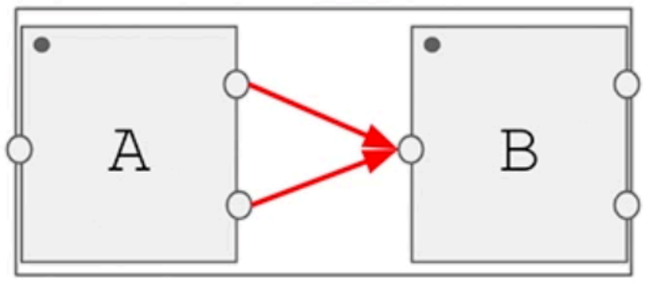
The number of inputs of B must be a multiple of the number of outputs of A

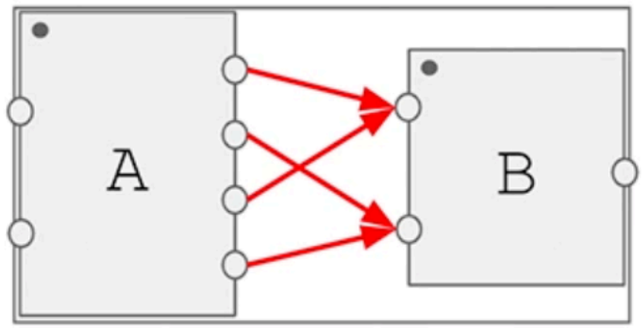




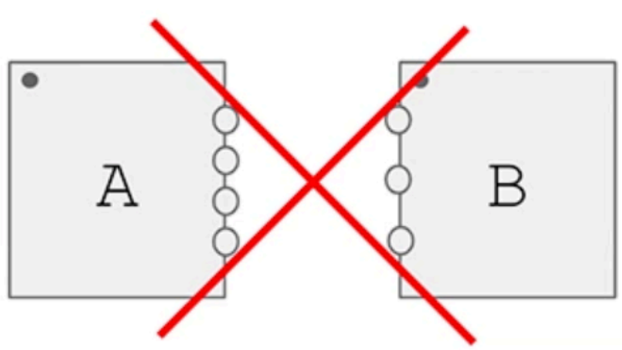
Faust Code

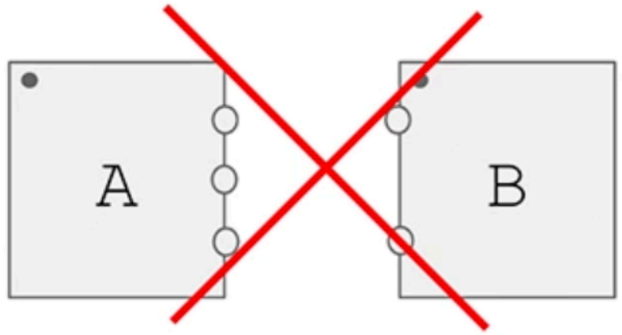
```
process = A :> B;
```





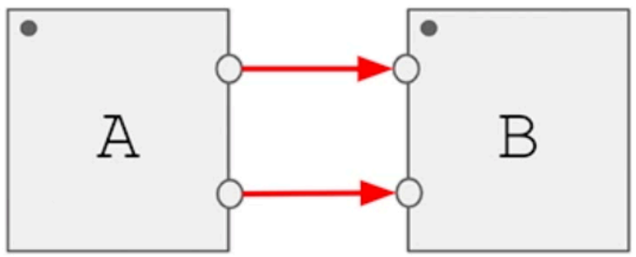
The number of outputs of A must be a multiple of the number of inputs of B



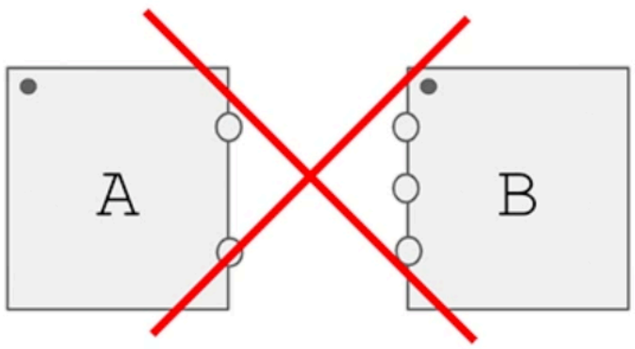


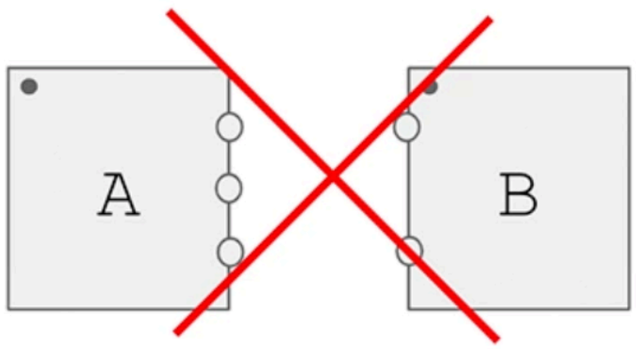
Faust Code

```
process = A : B;
```



The number of outputs of A and the number of inputs of B must be equal





Sequential Composition The *sequential composition* `A:B` (figure 3.3) expects:

$$\text{outputs}(A) = \text{inputs}(B) \tag{3.1}$$

It connects each output of A to the corresponding input of B:

$$A[i] \rightarrow [i]B \tag{3.2}$$

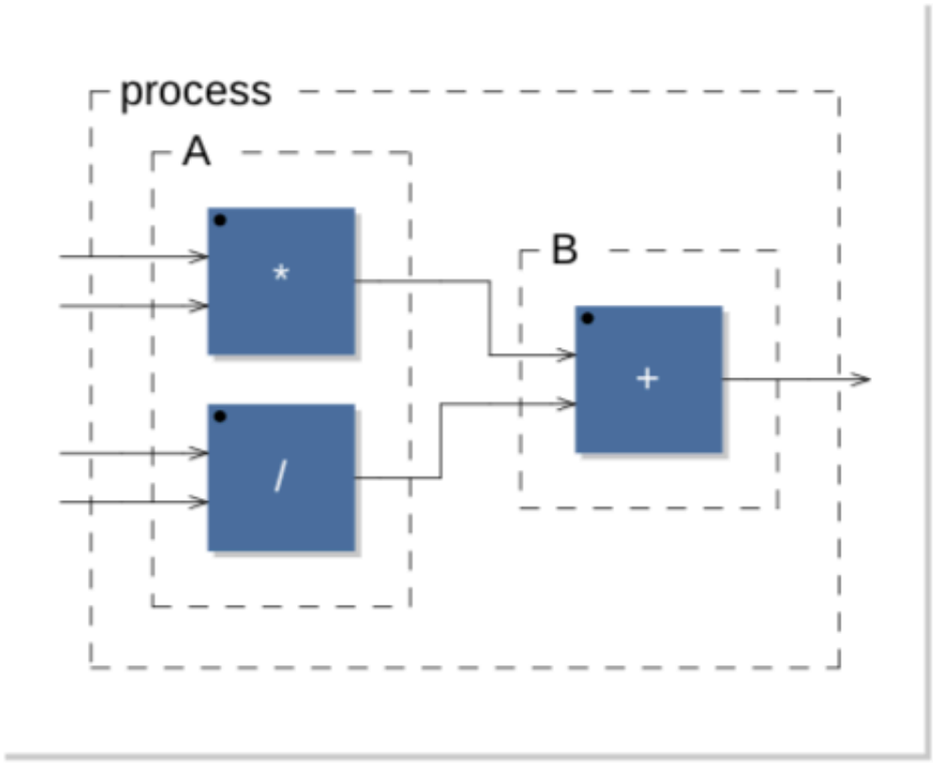


Figure 3.3: Example of sequential composition `((*,/):+)`

```
process = _ : 5;
```

error

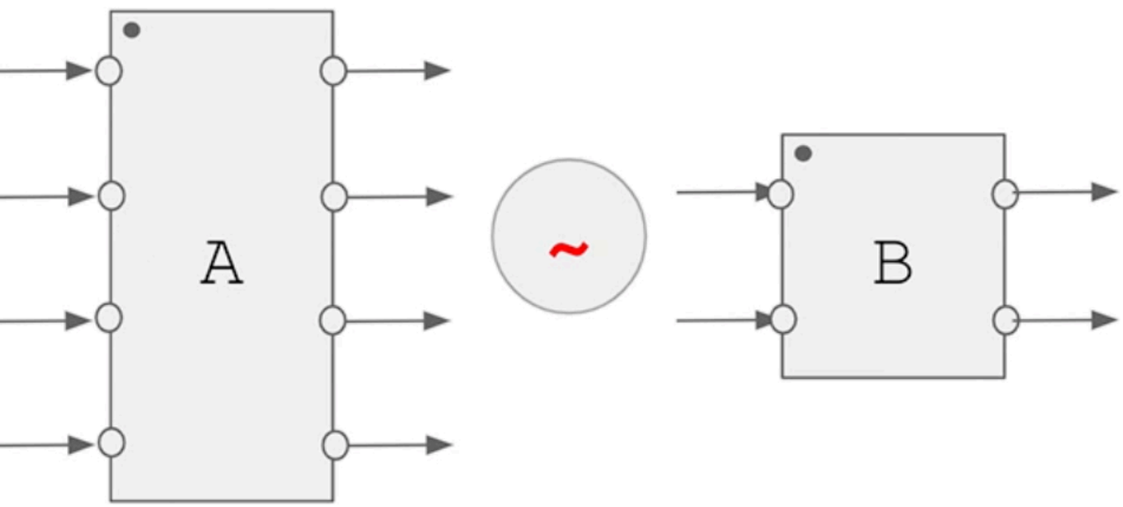
FAUST

```
process= + : / ;
```

error

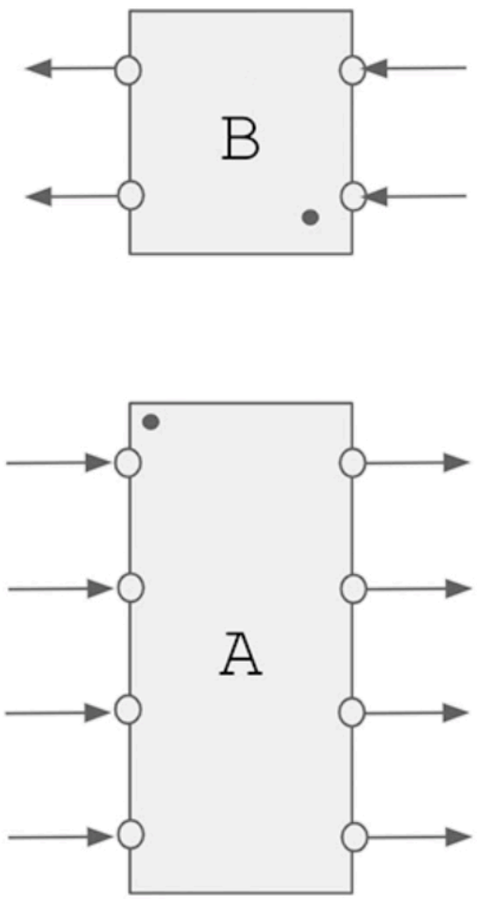
Faust Code

```
process = A ~ B;
```

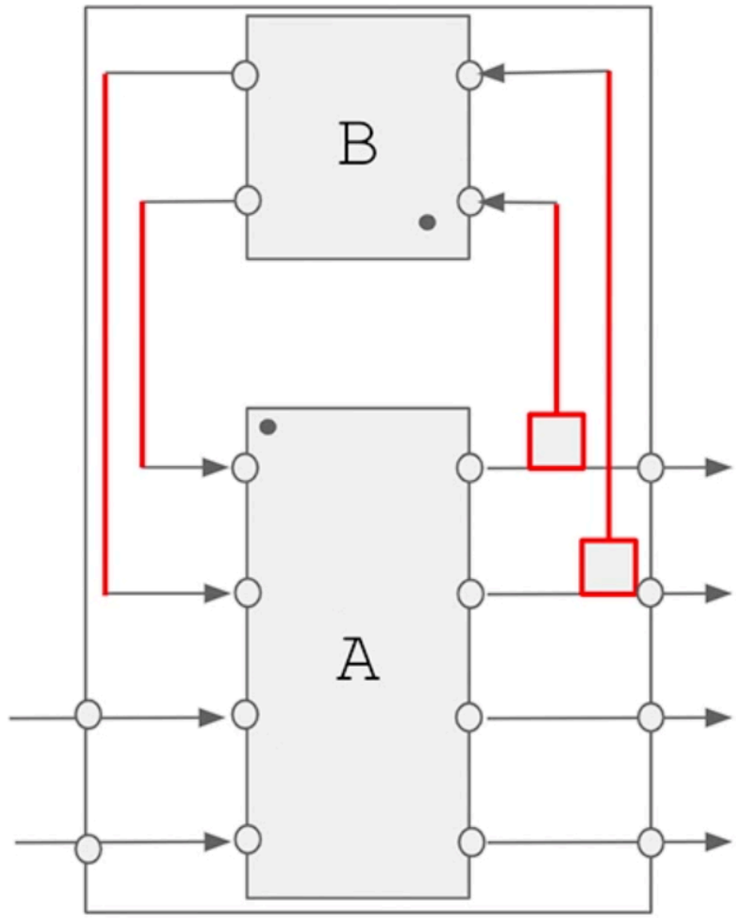


- *The number of inputs of B must be less or equal to the number of outputs of A.*
- *The number of outputs of B must be less or equal to the number of inputs of A.*

Step 1

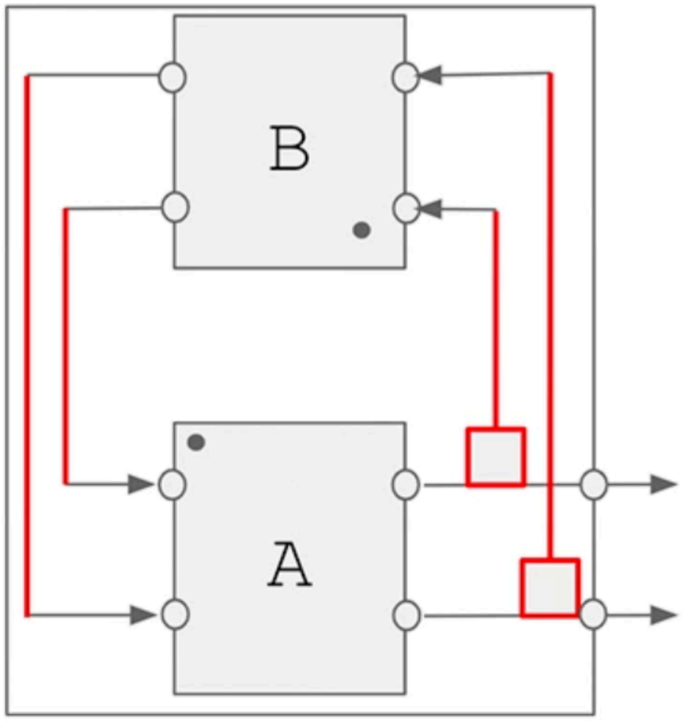


Step 2

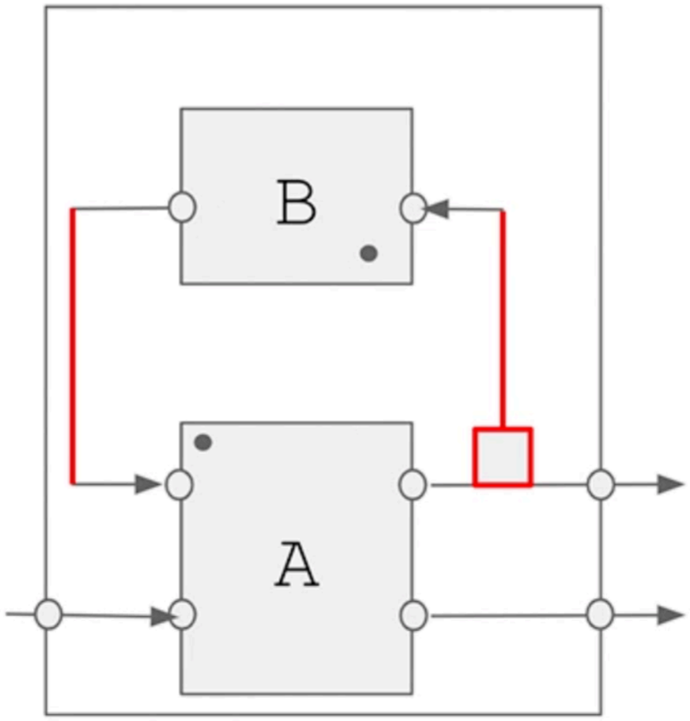


Faust Code

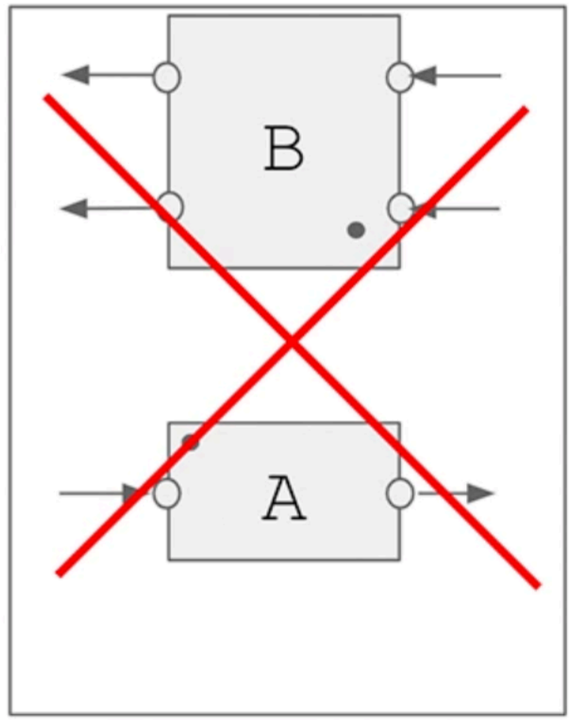
```
process = A ~ B;
```



OK



OK



WRONG

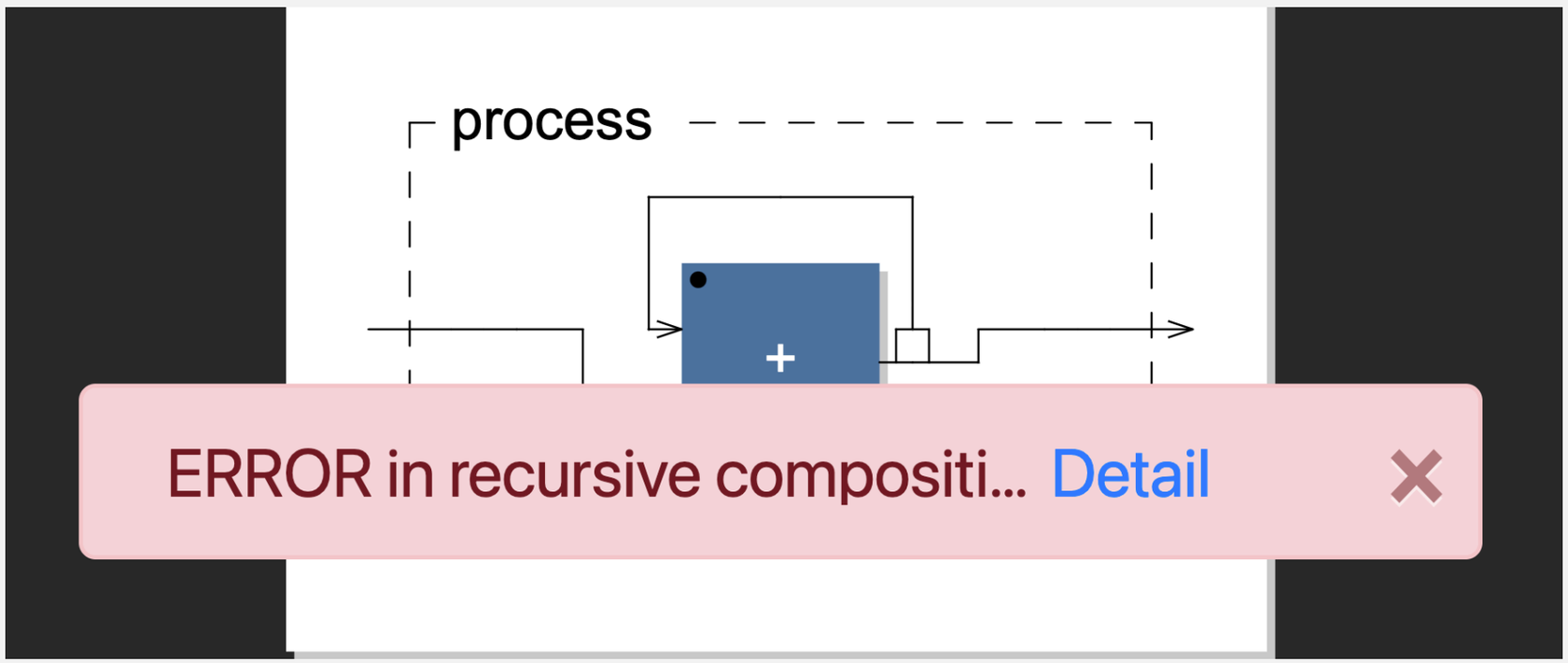
process = A~B ;

1) process = _ ~ + ;
2) process = + ~ _ ;
3) process = 1 ~ _ ;
4) process = + ~ (_ , 2 : *) ;
5) process = (1 , _ : +) ~ (_ , 2 : *) ;
6) process = + ~ (_ , 2 <: *, +) ;

process = A~B ;

1) process = _ ~ + ;
2) process = + ~ _ ;
3) process = 1 ~ _ ;
4) process = + ~ (_ , 2 : *) ;
5) process = (1 , _ : +) ~ (_ , 2 : *) ;
6) process = + ~ (_ , 2 <: *, +) ;

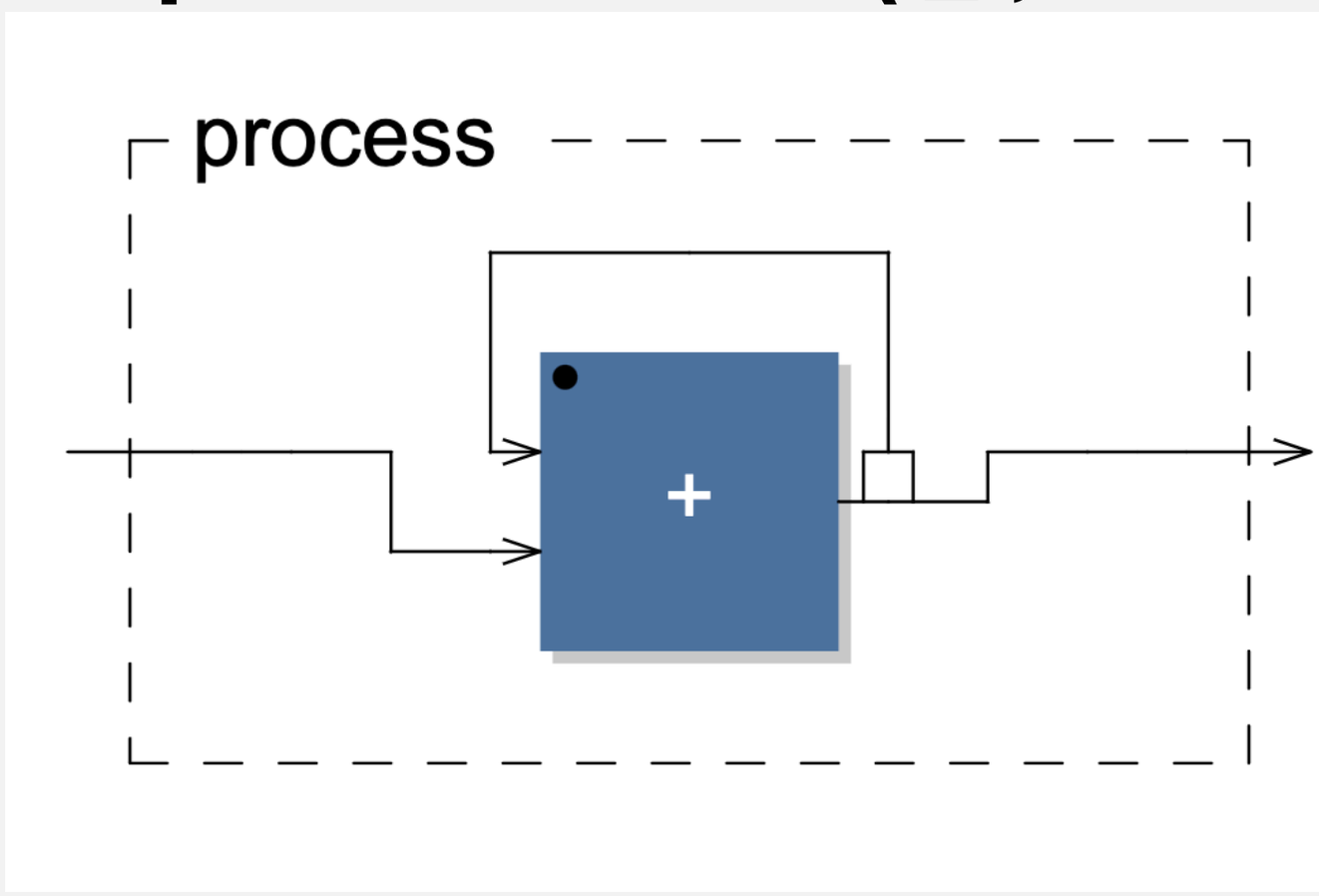
1



process = A~B ;

1) process = _ ~ + ;
2) process = + ~ _ ;
3) process = 1 ~ _ ;
4) process = + ~ (_ , 2 : *) ;
5) process = (1 , _ : +) ~ (_ , 2 : *) ;
6) process = + ~ (_ , 2 <: *, +) ;

2



```
process = A~B ;
```

1) `process = _ ~ + ;`
2) `process = + ~ _ ;`
3) `process = 1 ~ _ ;`
4) `process = + ~ ( _ , 2 : *) ;`
5) `process = (1 , _ : +) ~ (_ , 2 : *) ;`
6) `process = + ~ ( _ , 2 <: *, + ) ;`

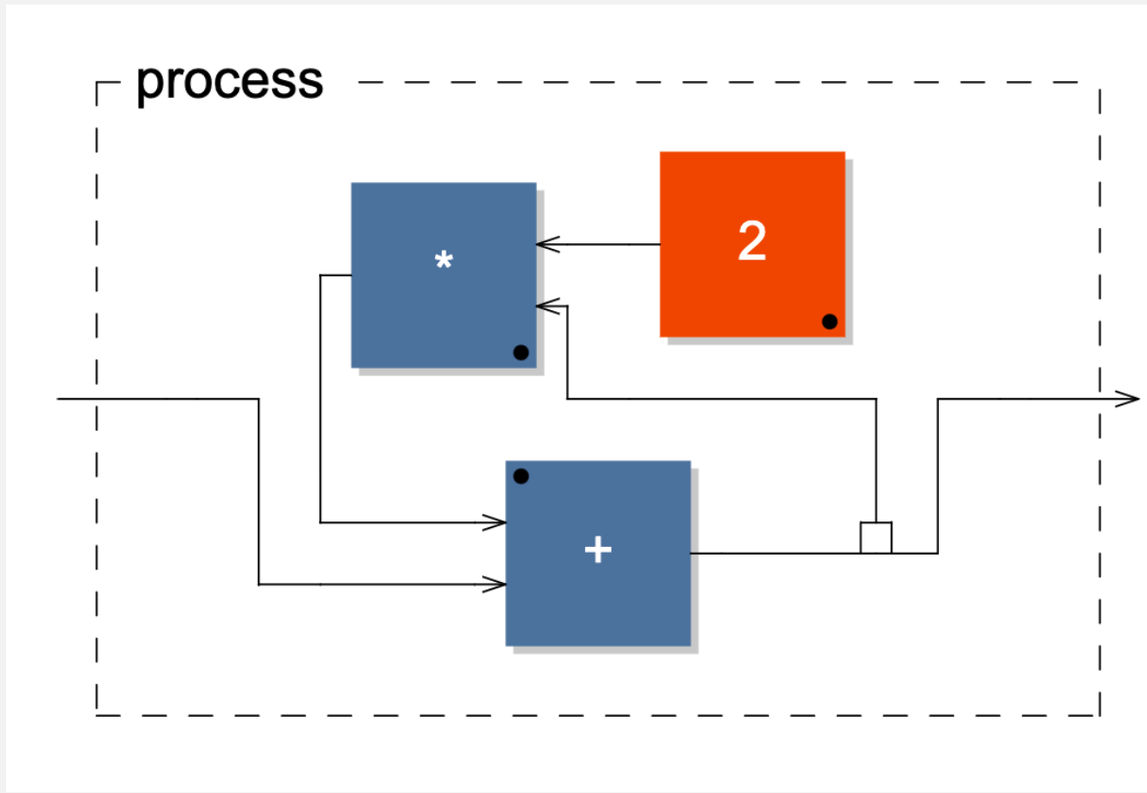
3

ERROR in recursive composition A~B The number ... Detail

```
process = A~B ;
```

1) `process = _ ~ + ;`
2) `process = + ~ _ ;`
3) `process = 1 ~ _ ;`
4) `process = + ~ ( _ , 2 : *) ;`
5) `process = (1 , _ : +) ~ (_ , 2 : *) ;`
6) `process = + ~ ( _ , 2 <: *, + ) ;`

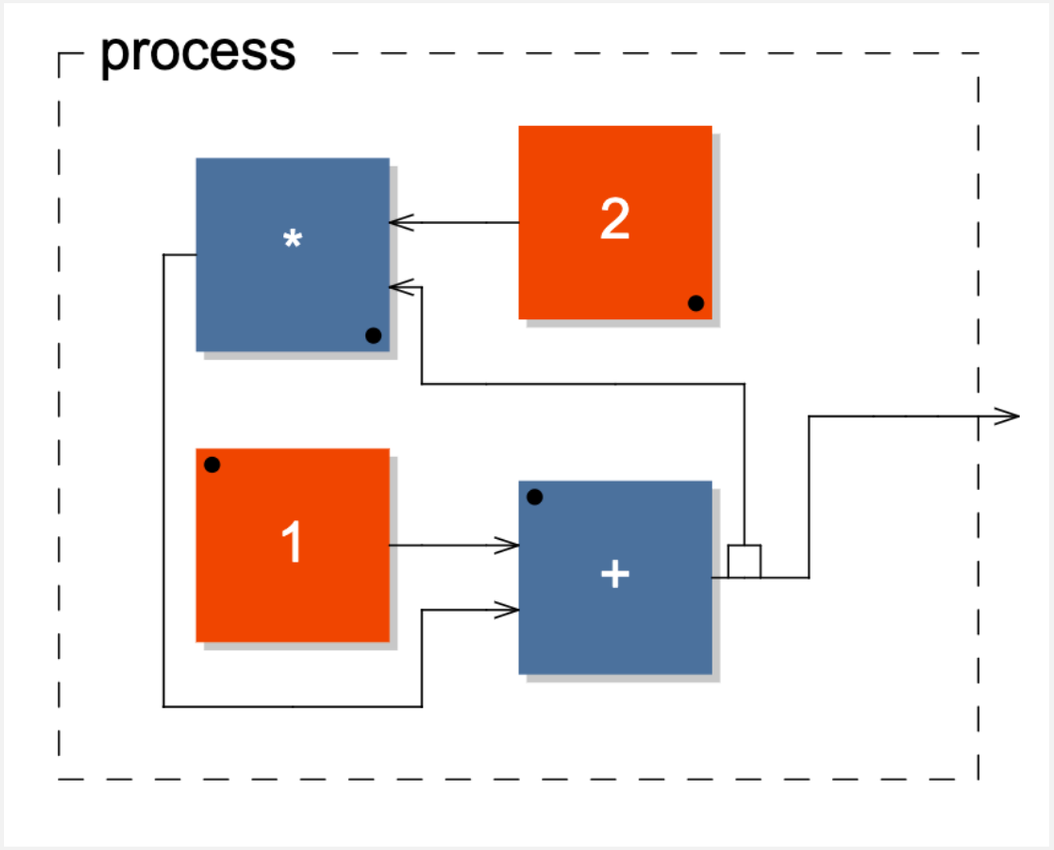
4



```
process = A~B ;
```

```
1)      process = _ ~ + ;
2)      process = + ~ _ ;
3)      process = 1 ~ _ ;
4)      process = + ~ ( _ , 2 : *) ;
5)      process = (1 , _ : +) ~ (_ , 2 : *) ;
6)      process = + ~ ( _ , 2 <: *, + ) ;
```

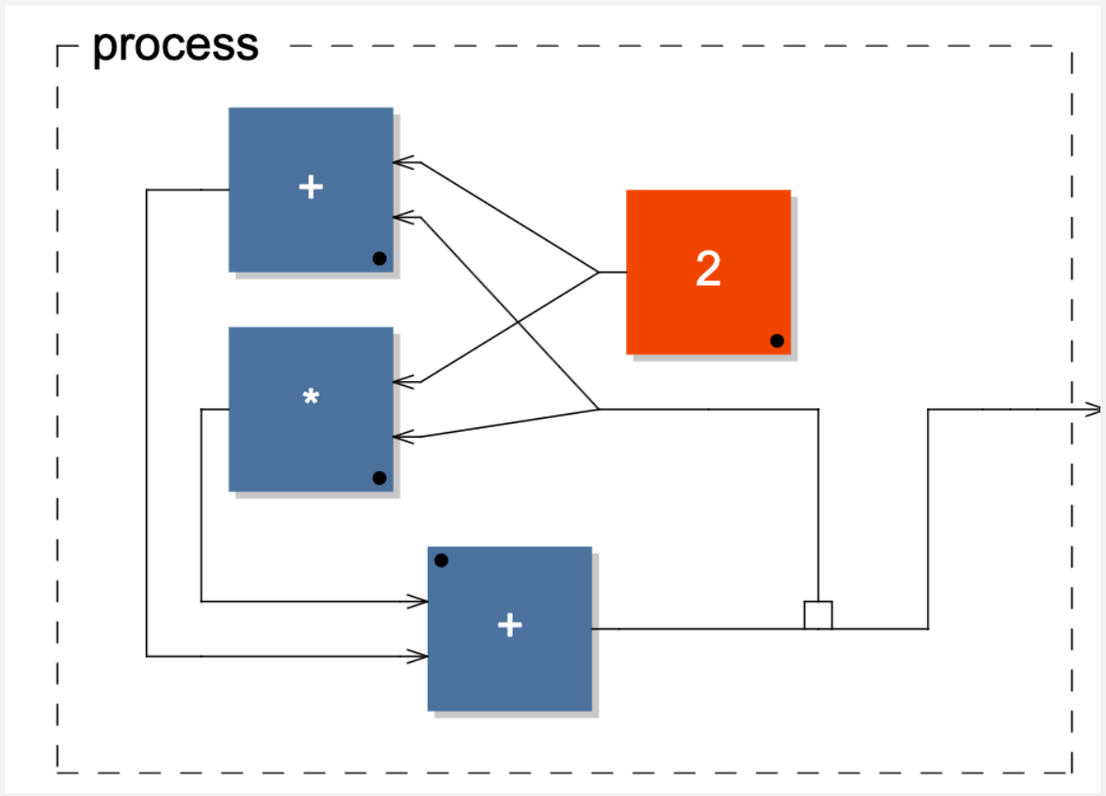
5



```
process = A~B ;
```

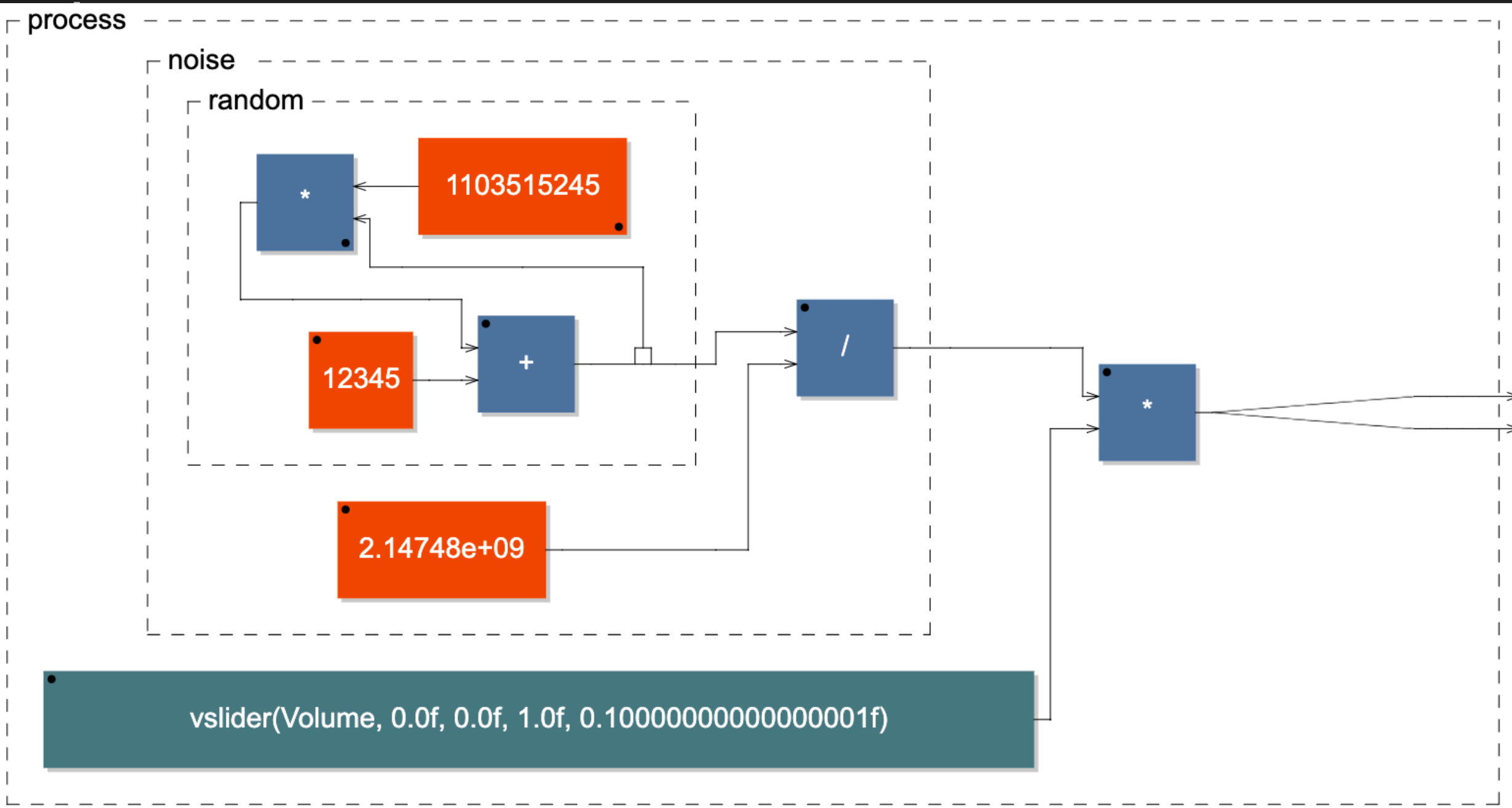
```
1)      process = _ ~ + ;
2)      process = + ~ _ ;
3)      process = 1 ~ _ ;
4)      process = + ~ ( _ , 2 : *) ;
5)      process = (1 , _ : +) ~ (_ , 2 : *) ;
6)      process = + ~ ( _ , 2 <: *, + ) ;
```

6



FAUST

```
1    random = +(12345) ~ *(1103515245);
2    noise = random/2147483647.0;
3    process = noise * vslider("Volume[style:knob]", 0,0,1,0.1) <: _,_;
```



FAUST

Infix Notation

```
1    import ("stdfaust.lib");
2    gain = 0.5;
3    process = gain*os.sawtooth(440);
```

Circuit Notation

FAUST

Infix Notation

```
1    import ("stdfaust.lib");
2    gain = 0.5;
3    process = gain*os.sawtooth(440);
```

Circuit Notation

```
1    import ("stdfaust.lib");
2    gain = 0.5;
3    process = gain, os.sawtooth(440) : *;
```

FAUST

```
1    import("stdfaust.lib");
2    process = os.osc (440) *.5 +
3      os.osc(440*2) *.25 +
4      os.osc(440) * .12;
```

FAUST

```
1    import("stdfaust.lib");
2    timbre (f) = os.osc (f) *.5 +
3    os.osc(f*2) *.25 +
4    os.osc(f*3) * .12;
5    process = timbre(440);
```

FAUST

```
1    import("stdfaust.lib");
2    freq = hslider("freq", 440, 50,1000, 0.01);
3    gain = hslider("gain", 0.5, 0,1, 0.01);
4    timbre (f) = os.osc (f) *.5 +
5    os.osc(f*2) *.25 +
6    os.osc(f*3) * .12;
7    process = gain*timbre(freq);
```

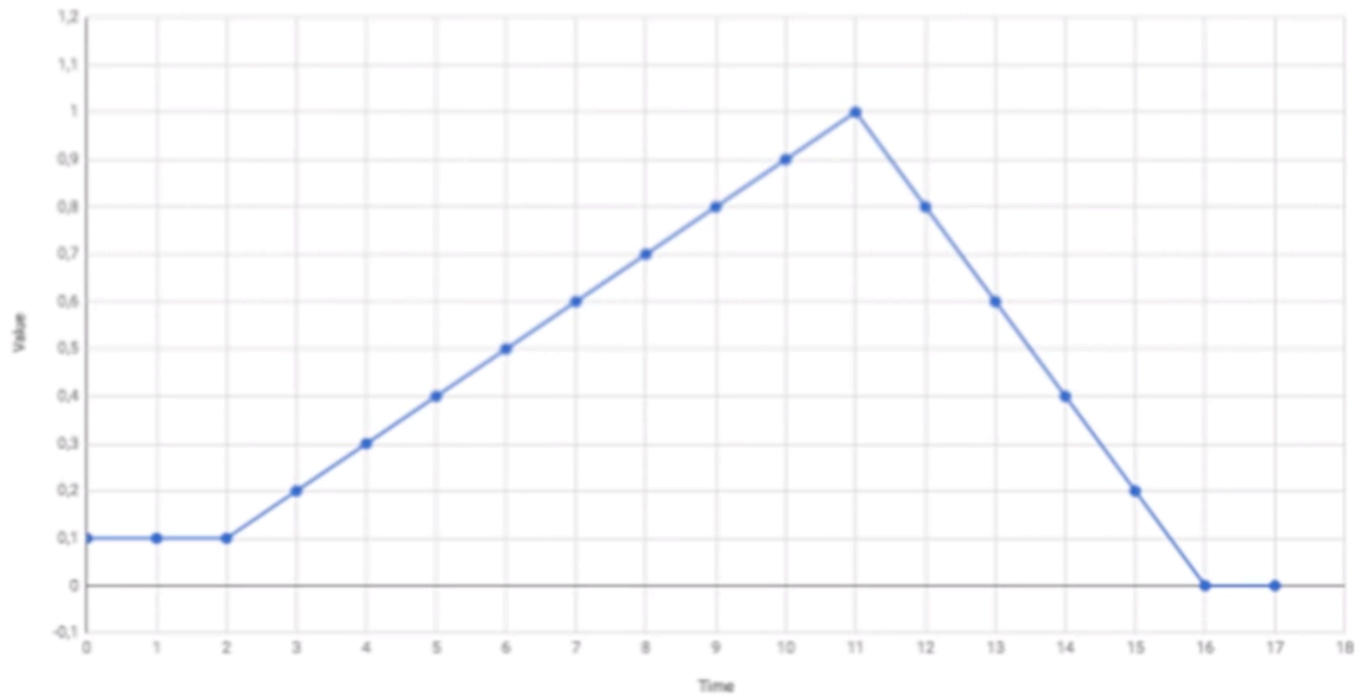
FAUST

```
1    import("stdfaust.lib");
2    freq = hslider("freq", 440, 50,1000, 0.01);
3    gain = hslider("gain", 0.5, 0,1, 0.01);
4    gate = button ("gate") : en.adsr(0.01,0.01,0.9,0.1);
5    timbre (f) = os.osc (f) *.5 +
6    os.osc(f*2) *.25 +
7    os.osc(f*3) * .12;
8    process = gate* gain*timbre(freq)*.05<:_,_;
9    effect = dm.zita_light;
```

USER INTERFACE

- hslider (“paramName", default, min, max, step)
- vslider (“paramName”, default, min, max, step)
- button (“paramName) : envelope (i.e., si.smoo))
- checkbox (“paramName”)


```
process = vslider("level", 0.1, 0, 1, 0.01);
```

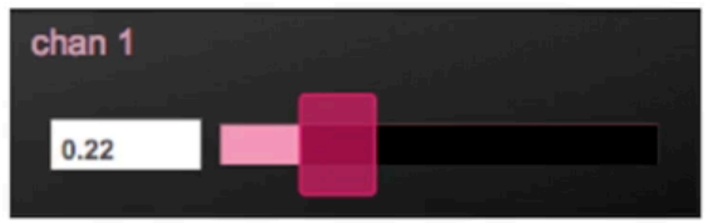


button("gate")

checkbox("activate")

nentry("pan", -0.19, -1, 1, 0.01)

vslider("level", 0.1, 0, 1, 0.01)



hslider("chan 1", 0.22, 0, 1, 0.01)

hbargraph("chan 1", 0, 1)

vslider("chan 2[style:knob]", 0.47, 0, 1, 0.01

FAUST

```
1   process = _, hslider("gain", 0, 0, 1, 0.01) : *;
```

```
1   import("stdfaust.lib");
2   gain = hslider ("gain", 0,0,1,0.01);
3   process = button ("440/880"), os.osc(440),
4   os.osc(880) : select2;
```

```
1   import("stdfaust.lib");
2   gain = hslider ("gain", 0,0,1,0.01);
3   process = nentry("selector" ,0,0,2,1), os.osc(440),
4   os.osc(880), os.osc(1320) : select3;
```