Structured Synchronous Programming with Céu

(mixing control with data flow)

Francisco Sant'Anna

“Hello world!” in Céu

“Hello world!” in Céu

- Blinking a LED
    1. *on ↔ off every 500ms*

“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*

```
loop do
    await 500ms;
    _leds_toggle();
end
```

“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*

```
loop do
    await 500ms;
    _leds_toggle();
end
```

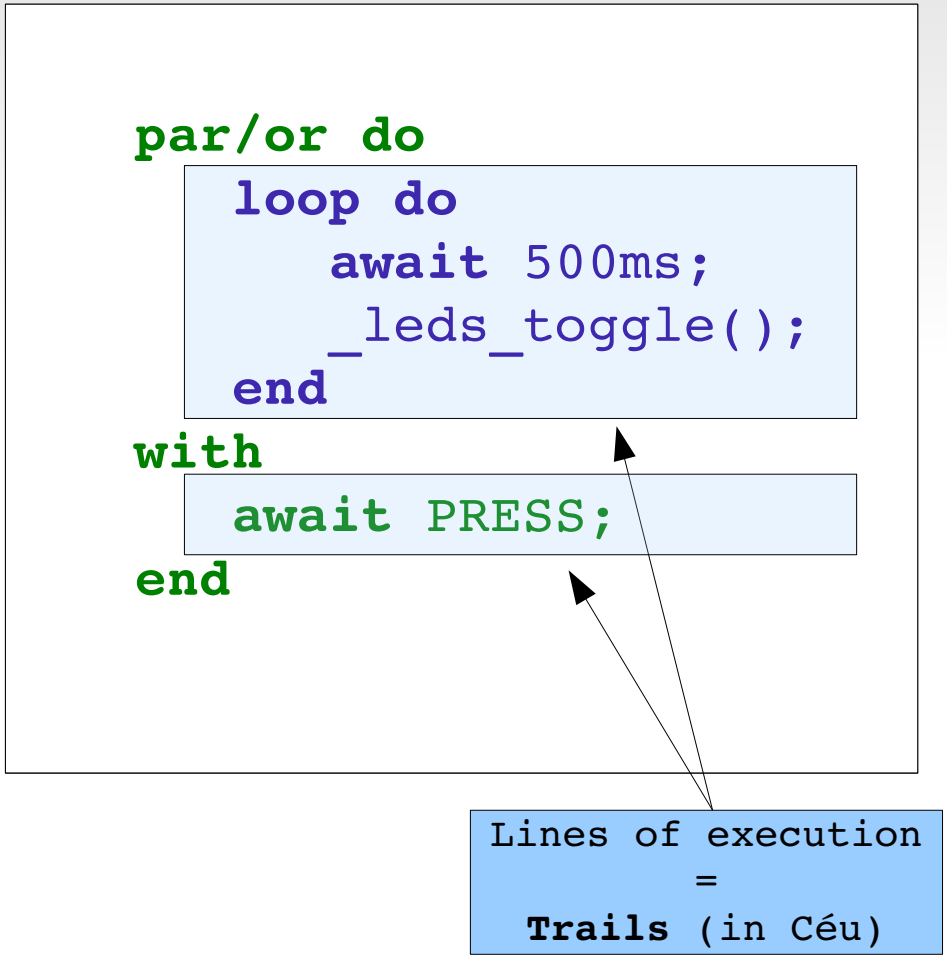
“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*

```
par/or do
    loop do
        await 500ms;
        _leds_toggle();
    end
with
    await PRESS;
end
```

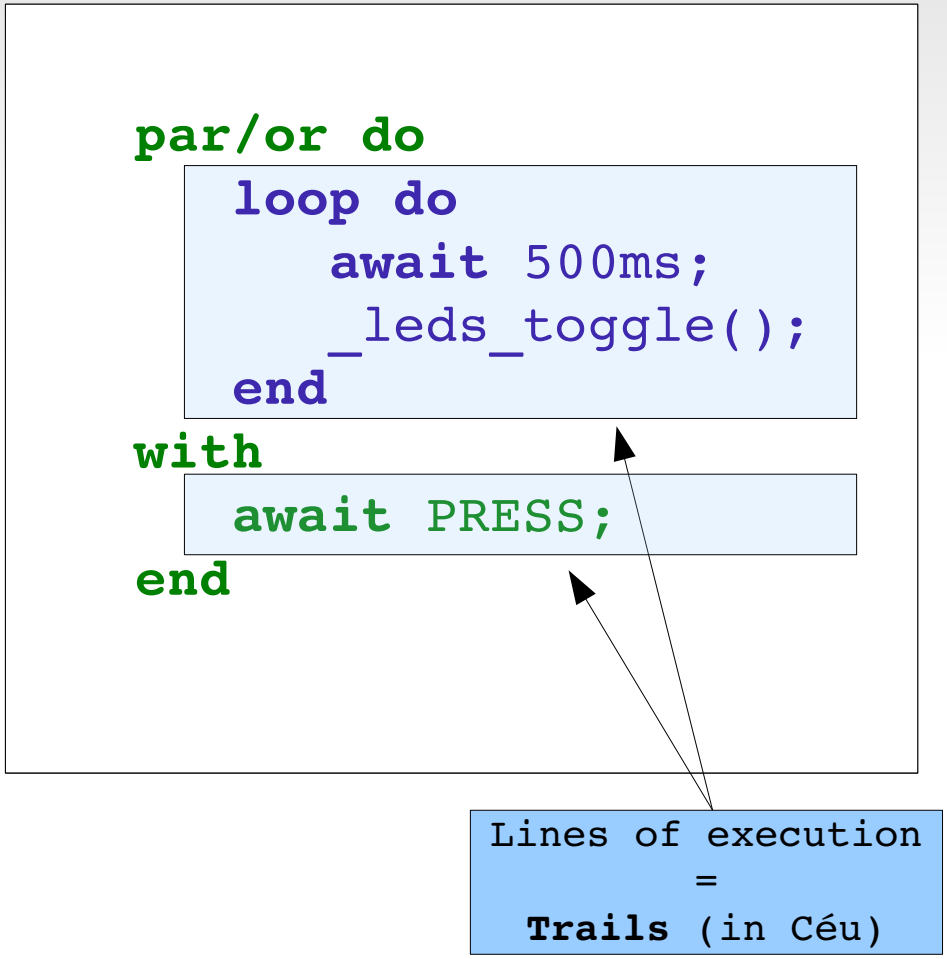
“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*



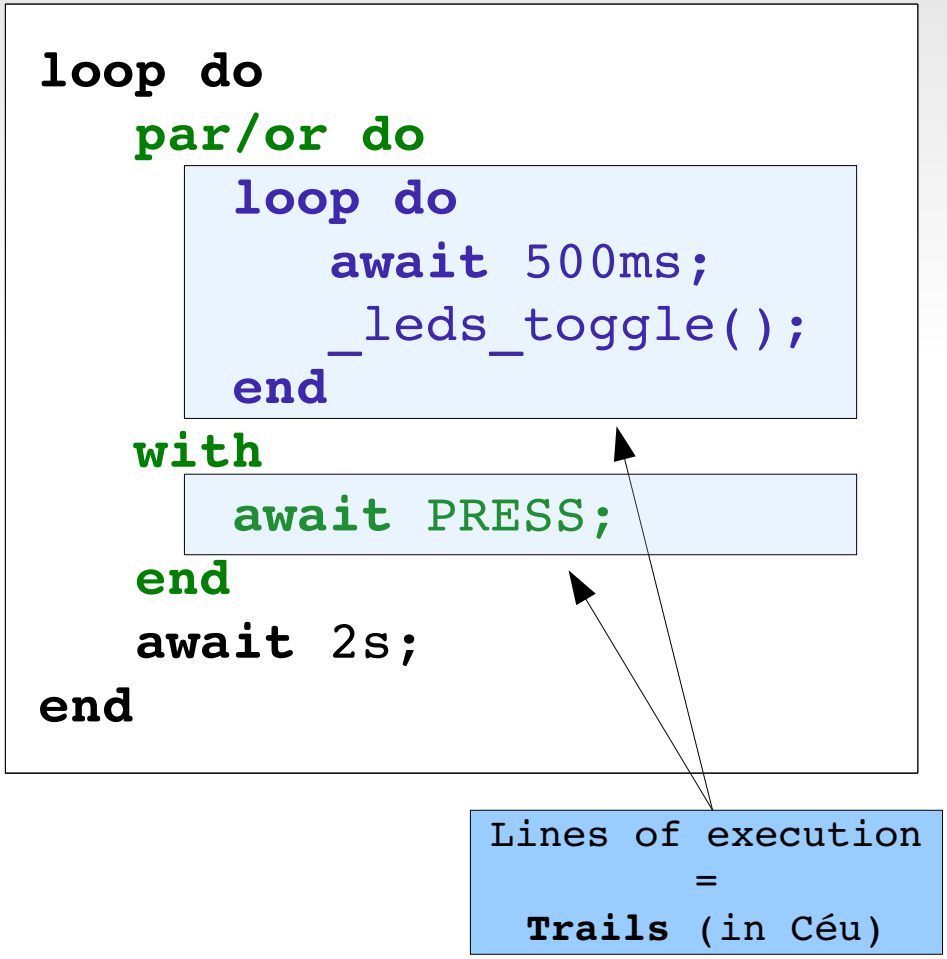
“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*
  3. *restart after 2s*



“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*
  3. *restart after 2s*



“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*
  3. *restart after 2s*

- Compositions
  - seq, loop, par *(trails)*
    - At any level of depth

```
loop do
   par/or do
      loop do
         await 500ms;
         _leds_toggle();
      end
   with
      await PRESS;
   end
   await 2s;
end
```

Lines of execution
=
Trails (in Céu)

“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*
  3. *restart after 2s*

```
loop do
   par/or do
      loop do
         await 500ms;
         _leds_toggle();
      end
   with
      await PRESS;
   end
   await 2s;
end
```

Lines of execution
=
Trails (in Céu)

- Compositions
  - seq, loop, par *(trails)*
    - At any level of depth
  - ~~state variables / communication~~

“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*
  3. *restart after 2s*

```
loop do
    par/or do
        loop do
            await 500ms;
            _leds_toggle();
        end
    with
        await PRESS;
    end
    await 2s;
end
```

- Compositions
  - seq, loop, par *(trails)*
    - At any level of depth
  - ~~state variables / communication~~

“Hello world!” in Céu

- Blinking a LED
  1. *on ↔ off every 500ms*
  2. *stop after “press”*
  3. *restart after 2s*

- Compositions
  - seq, loop, par *(trails)*
    - At any level of depth
  - ~~state variables / communication~~

Where is my data?

```
loop do
   par/or do
      loop do
         await 500ms;
         _leds_toggle();
      end
   with
      await PRESS;
   end
   await 2s;
end
```

Mixing data & control flow

Mixing data & control flow

- Controlling the ball inside the screen
  - click mouse to start
  - ball moves in one direction with an acceleration
  - click mouse to turn clockwise (single-button controller)

Mixing data & control flow

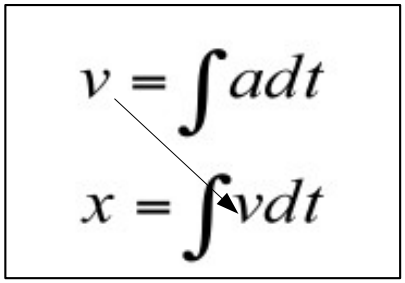
- Controlling the ball inside the screen
  - click mouse to start
  - ball moves in one direction with an acceleration
  - click mouse to turn clockwise (single-button controller)
- Data flow:
  - position is the integral of velocity
  - velocity is the integral of acceleration

$$v = \int a dt$$

$$x = \int v dt$$

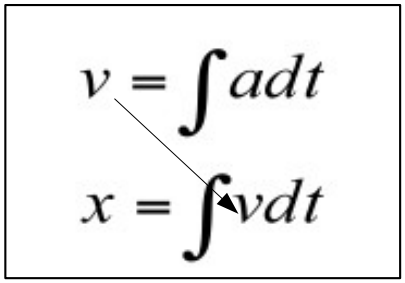
Mixing data & control flow

- Controlling the ball inside the screen
  - click mouse to start
  - ball moves in one direction with an acceleration
  - click mouse to turn clockwise (single-button controller)
- Data flow:
  - position is the integral of velocity
  - velocity is the integral of acceleration



Mixing data & control flow

- Controlling the ball inside the screen
  - click mouse to start
  - ball moves in one direction with an acceleration
  - click mouse to turn clockwise (single-button controller)
- Data flow:
  - position is the integral of velocity
  - velocity is the integral of acceleration



- Control flow:
  - abrupt changes in the clicks (non-continuous functions)
  - discontinuity suggests state (e.g., `isRunning`, `currentDirection`)

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second


    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second


    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end

    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

Moving the ball

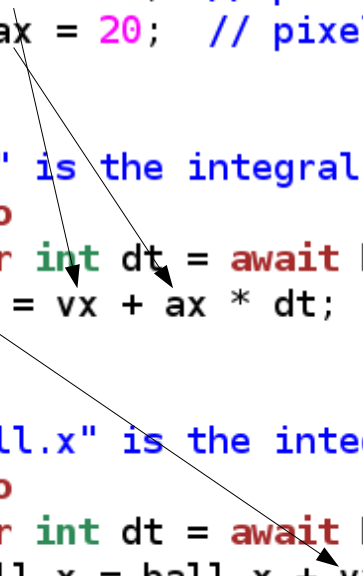
```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```



Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)
  - Body (any code in Céu)

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)
  - Body (any code in Céu)

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)
  - Body (any code in Céu)
  - Organisms ~ Simula Objects

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
input int NEXT_FRAME;   // int: time between frames
input int MOUSE_BUTTON; // int: clicked button

data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

await MOUSE_BUTTON; // wait for the click to start

var float vx = 20;  // pixels per second
var float ax = 20;  // pixels per second per second

par do
    // "vx" is the integral of "ax"
    loop do
        var int dt = await NEXT_FRAME;
        vx = vx + ax * dt;
    end
with
    // "ball.x" is the integral of "vx"
    loop do
        var int dt = await NEXT_FRAME;
        ball.x = ball.x + vx * dt;
    end
end
```

- Lengthy, low level...
  - simple data dependency pattern
    - 1 `par` + 2 `loop`
  - relies on mutation
    - scheduling follows lexical order
- Abstractions
  - Interface (fields + methods)
  - Body (any code in Céu)
  - Organisms ~ Simula Objects

```
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Organisms react directly to the environment

Moving the ball

```
<...>                    // inputs & ball declarations
```

Moving the ball

```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end
```

Moving the ball

```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start
```

Moving the ball

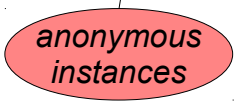
```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second
```

Moving the ball

```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second

var Integral_Over_Time _ (vx&,     ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```

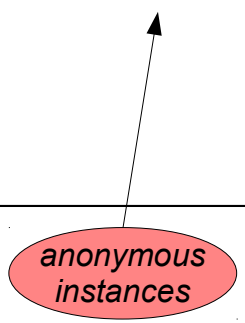
Moving the ball

```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second

var Integral_Over_Time _ (vx&,     ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```



Moving the ball

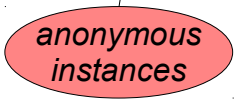
```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second

var Integral_Over_Time _ (vx&,      ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```



- Organisms
  - body executes in parallel with the block

Moving the ball

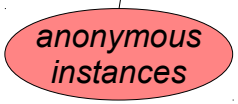
```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second

var Integral_Over_Time _ (vx&,      ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```



- Organisms
  - body executes in parallel with the block

Moving the ball

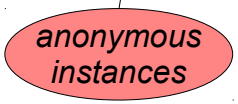
```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second

var Integral_Over_Time _ (vx&,      ax&);
var Integral_Over_Time _ (ball.x&, vx&);
    // organism bodies execute in parallel
    // with their enclosing block
```



- Organisms
  - body executes in parallel with the block

Moving the ball

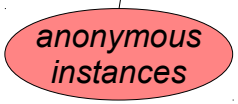
```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second

var Integral_Over_Time _ (vx&,      ax&);
var Integral_Over_Time _ (ball.x&, vx&);
    // organism bodies execute in parallel
    // with their enclosing block

<...>                       // whatever comes next is in parallel
```



- Organisms
  - body executes in parallel with the block

Moving the ball

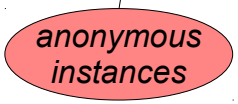
```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second

var Integral_Over_Time _ (vx&,     ax&);
var Integral_Over_Time _ (ball.x&, vx&);
    // organism bodies execute in parallel
    // with their enclosing block

<...>                       // whatever comes next is in parallel
```



- Organisms
  - body executes in parallel with the block

Moving the ball

```
<...>                       // inputs & ball declarations

class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    every dt in NEXT_FRAME do
        accumulator = accumulator + value;
    end
end

await MOUSE_BUTTON;         // wait for the click to start

var float vx = 20;          // pixels per second
var float ax = 20;          // pixels per second per second

var Integral_Over_Time _ (vx&,      ax&);
var Integral_Over_Time _ (ball.x&, vx&);
    // organism bodies execute in parallel
    // with their enclosing block

await FOREVER;              // whatever comes next is in parallel
```



- Organisms
  - body executes in parallel with the block

Making the turn

```
<...>                  // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start
```

Making the turn

```
<...>                  // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start



    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn
```

Making the turn

```
<...>                  // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start



    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn



    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
```

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start



    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn



    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
```

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start



    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn



    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
```

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start



    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn



    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
```

- Doesn't work as expected
  - previous dependencies still active

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start



    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn



    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
```

- Doesn't work as expected
  - previous dependencies still active

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start

do

    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn
end

do
    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
end
```

- Doesn't work as expected
  - previous dependencies still active

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start

do

    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn
end

do
    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
end
```

- Doesn't work as expected
  - previous dependencies still active
- Lexical scope for organisms
  - data reclaimed, body aborted

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start

do

    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn
end

do
    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
end
```

- Doesn't work as expected
  - previous dependencies still active
- Lexical scope for organisms
  - data reclaimed, body aborted

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start

do

    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn
end

do
    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
end
```

- Doesn't work as expected
  - previous dependencies still active
- Lexical scope for organisms
  - data reclaimed, body aborted

Making the turn

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON; // wait for the click to start

do

    var float vx=20, ax=20;
    var Integral_Over_Time _ (vx&,      ax&);
    var Integral_Over_Time _ (ball.x&, vx&);

    await MOUSE_BUTTON; // wait for the click to turn
end

do
    var float vy=20, ay=20;
    var Integral_Over_Time _ (vy&,      ay&);
    var Integral_Over_Time _ (ball.y&, vy&);

    await MOUSE_BUTTON; // wait for the click to turn
end
```

- Doesn't work as expected
  - previous dependencies still active
- Lexical scope for organisms
  - data reclaimed, body aborted
- Mixing data and control flow
  - automatic data updates
  - no explicit state machines
  - structured code with lexical scope

Closing the loop

```
<...>                    // inputs, ball, integral declarations

await MOUSE_BUTTON;      // wait for the click to start

loop do




end
```

Closing the loop

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON;     // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end









end
```

Closing the loop

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON;     // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,     ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end




end
```

Closing the loop

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON;     // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,     ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end




end
```

Closing the loop

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON;     // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,     ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,     ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

Closing the loop

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON;     // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,     ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,     ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...

Closing the loop

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON;     // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,     ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,     ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,     ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular

Closing the loop

```
<...>                   // inputs, ball, integral declarations

await MOUSE_BUTTON;     // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular
  - “abstractable” with another class

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular
  - “abstractable” with another class

```
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    await MOUSE_BUTTON;
end
```

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular
  - “abstractable” with another class

```
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    await MOUSE_BUTTON;
end
```

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular
  - “abstractable” with another class

```
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    await MOUSE_BUTTON;
end

// declare an instance and await it terminate
do
    var Move_Until_Button move (ball.x&, 20, 20);
    await move;
end
```

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular
  - “abstractable” with another class

```
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    await MOUSE_BUTTON;
end

// declare an instance and await it terminate
do
    var Move_Until_Button move (ball.x&, 20, 20);
    await move;
end
```

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular
  - “abstractable” with another class

```
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    await MOUSE_BUTTON;
end

// declare an instance and await it terminate
do
    var Move_Until_Button move (ball.x&, 20, 20);
    await move;
end

// same as above but anonymous
do Move_Until_Button(ball.x&, 20, 20);
```

Closing the loop

```
<...>                       // inputs, ball, integral declarations

await MOUSE_BUTTON;         // wait for the click to start

loop do
    do
        // move right
        var float vx=20, ax=20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;
    end
    do
        // move down
        var float vy=20, ay=20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move left
        var float vx=-20, ax=-20;
        var Integral_Over_Time _ (vx&,      ax&);
        var Integral_Over_Time _ (ball.x&, vx&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
    do
        // move up
        var float vy=-20, ay=-20;
        var Integral_Over_Time _ (vy&,      ay&);
        var Integral_Over_Time _ (ball.y&, vy&);
        await MOUSE_BUTTON;     // wait for the click to turn
    end
end
```

- Lenghty code...
  - ... but regular
  - “abstractable” with another class

```
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    await MOUSE_BUTTON;
end

// declare an instance and await it terminate
do
    var Move_Until_Button move (ball.x&, 20, 20);
    await move;
end

// same as above but anonymous
do Move_Until_Button(ball.x&, 20, 20);
```

Closing the loop

```
// INPUTS
input int NEXT_FRAME;
input int MOUSE_BUTTON;
```

Closing the loop

```
// INPUTS
input int NEXT_FRAME;
input int MOUSE_BUTTON;

// ABSTRACTIONS
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    <...>
end
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    <...>
end
```

Closing the loop

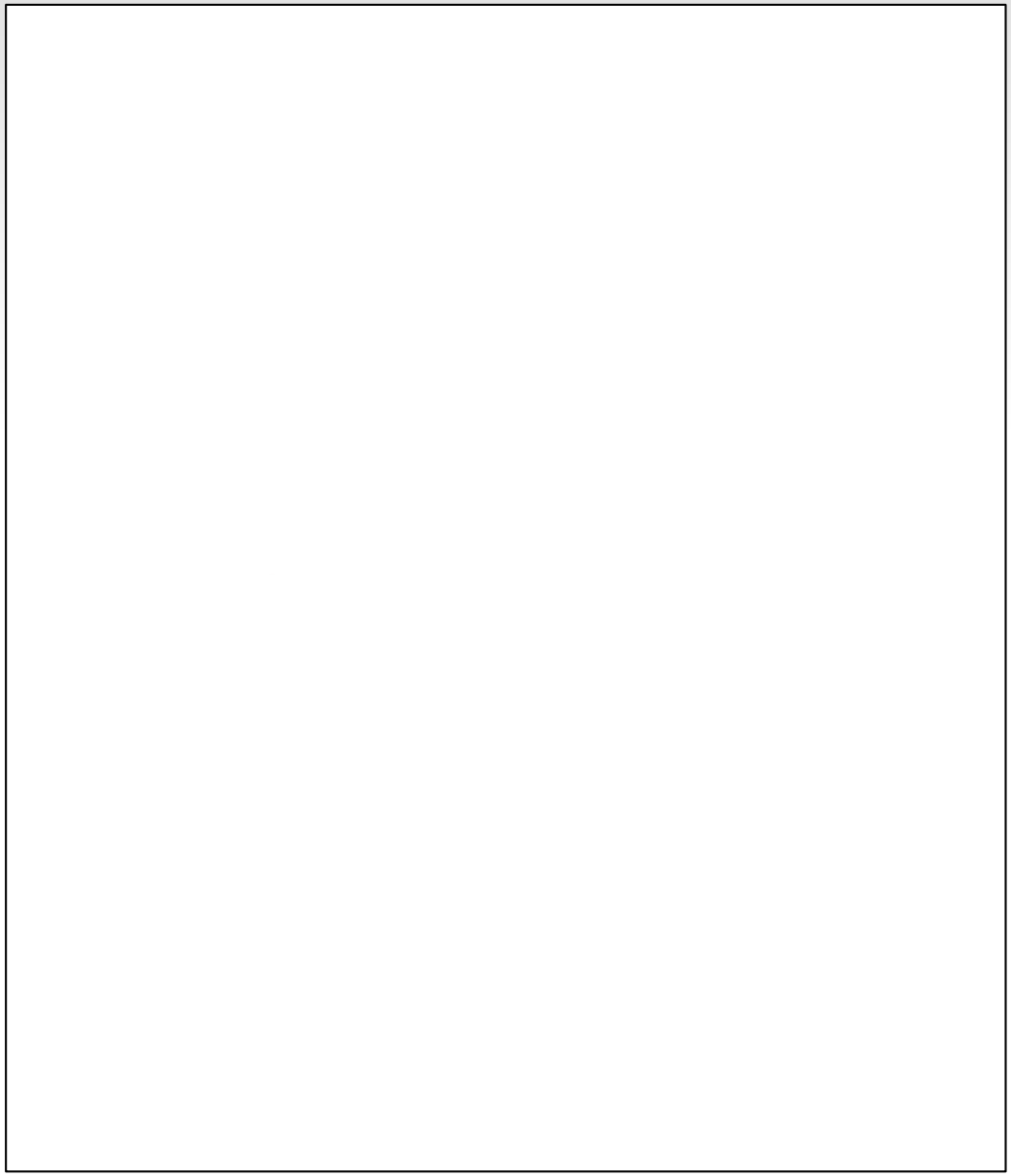
```
// INPUTS
input int NEXT_FRAME;
input int MOUSE_BUTTON;

// ABSTRACTIONS
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    <...>
end
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    <...>
end

// PROGRAM DATA
data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);
```

Closing the loop

```
// INPUTS
input int NEXT_FRAME;
input int MOUSE_BUTTON;

// ABSTRACTIONS
class Integral_Over_Time with
    var float& accumulator;
    var float& value;
do
    <...>
end
class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
do
    <...>
end

// PROGRAM DATA
data Ball with
    var float x;
    var float y;
    var float radius;
end
var Ball ball = Ball(130,130,8);

// PROGRAM FLOW
await MOUSE_BUTTON;
loop do
    do Move_Until_Button(ball.x&,  20,  20);
    do Move_Until_Button(ball.y&, -20, -20);
    do Move_Until_Button(ball.x&,  20,  20);
    do Move_Until_Button(ball.y&, -20, -20);
end
```

- Structured, sequential, and with encapsulated data flow.

More abstractions
(two players)



More abstractions
(two players)

```
var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

```
var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

```
input int MOUSE_BUTTON;     // int: clicked button
```

```
var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

```
input int MOUSE_BUTTON;     // int: clicked button









class Player with
    var Ball ball;
    var int  button;
do
    await MOUSE_BUTTON;
    loop do
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
    end
end

var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

```
input int MOUSE_BUTTON;     // int: clicked button

class Player with
    var Ball ball;
    var int  button;
do
    await MOUSE_BUTTON;
    loop do
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
    end
end

var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

```
input int MOUSE_BUTTON;     // int: clicked button









class Player with
    var Ball ball;
    var int  button;
do
    await MOUSE_BUTTON;
    loop do
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
    end
end

var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

```
input int MOUSE_BUTTON;     // int: clicked button

<...>                       // inputs, integral, ball

class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
    var int    button;      // await this button
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    var int clk = await MOUSE_BUTTON
                  until clk==button;
                           // check clicked button
end

class Player with
    var Ball ball;
    var int  button;
do
    await MOUSE_BUTTON;
    loop do
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
    end
end

var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

```
input int MOUSE_BUTTON;     // int: clicked button

<...>                       // inputs, integral, ball

class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
    var int    button;      // await this button
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    var int clk = await MOUSE_BUTTON
                  until clk==button;
                           // check clicked button
end

class Player with
    var Ball ball;
    var int  button;
do
    await MOUSE_BUTTON;
    loop do
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
    end
end

var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

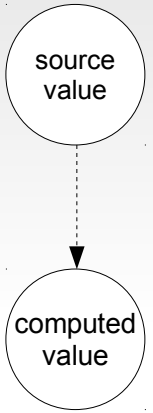
```
input int MOUSE_BUTTON;     // int: clicked button

<...>                       // inputs, integral, ball

class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
    var int    button;      // await this button
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    var int clk = await MOUSE_BUTTON
                  until clk==button;
                            // check clicked button
end

class Player with
    var Ball ball;
    var int  button;
do
    await MOUSE_BUTTON;
    loop do
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
    end
end

var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);
```

More abstractions
(two players)

```
input int MOUSE_BUTTON;     // int: clicked button

<...>                       // inputs, integral, ball

class Move_Until_Button with
    var float& pos;
    var float  v0, a0;
    var int    button;      // await this button
do
    var Integral_Over_Time _ (v0&,  a0&);
    var Integral_Over_Time _ (pos&, v0&);
    var int clk = await MOUSE_BUTTON
                  until clk==button;
                            // check clicked button
end

class Player with
    var Ball ball;
    var int  button;
do
    await MOUSE_BUTTON;
    loop do
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
        do Move_Until_Button(ball.x&,  20,  20, button);
        do Move_Until_Button(ball.y&, -20, -20, button);
    end
end

var Player p1 (Ball(200,130,8), BUTTON_LEFT);
var Player p2 (Ball(300,130,8), BUTTON_RIGHT);

await FOREVER;
```

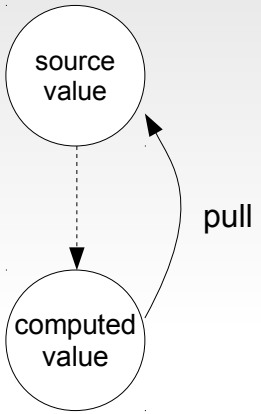
Pull/Push data flow

Pull/Push data flow



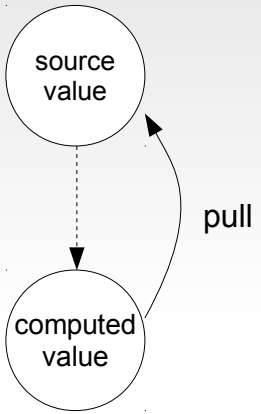
Pull/Push data flow

- Pull approach
  - read values continuously
  - good for periodic/fast streams



Pull/Push data flow

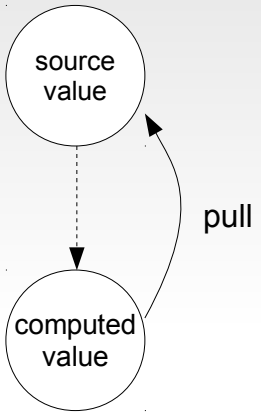
- Pull approach
  - read values continuously
  - good for periodic/fast streams




```
var float vx=20, ax=20;
var Integral_Over_Time _ (vx&,     ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```

Pull/Push data flow

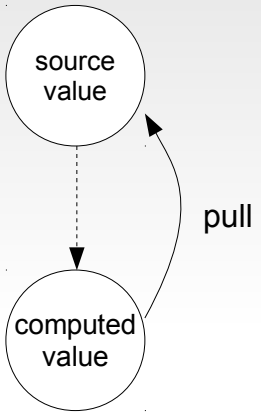
- Pull approach
  - read values continuously
  - good for periodic/fast streams




```
var float vx=20, ax=20;
var Integral_Over_Time _ (vx&,      ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```

Pull/Push data flow

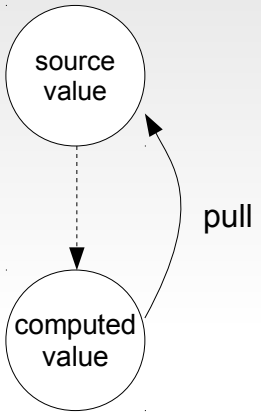
- Pull approach
  - read values continuously
  - good for periodic/fast streams




```
var float vx=20, ax=20;
var Integral_Over_Time _ (vx&,      ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```

Pull/Push data flow

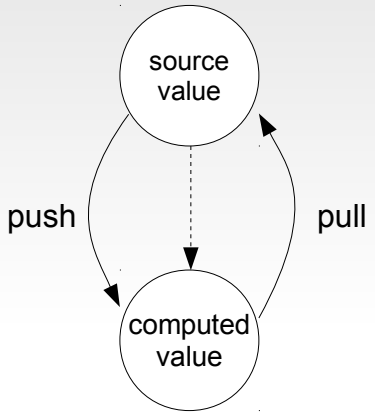
- Pull approach
  - read values continuously
  - good for periodic/fast streams




```
var float vx=20, ax=20;
var Integral_Over_Time _ (vx&, ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```

Pull/Push data flow

- Pull approach
  - read values continuously
  - good for periodic/fast streams
- Push approach
  - notify changes on values
  - good for slow/occasional streams
  - other advantages
    - efficiency
    - encapsulation
    - decoupling




```
var float vx=20, ax=20;
var Integral_Over_Time _ (vx&, ax&);
var Integral_Over_Time _ (ball.x&, vx&);
```

Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

Push example

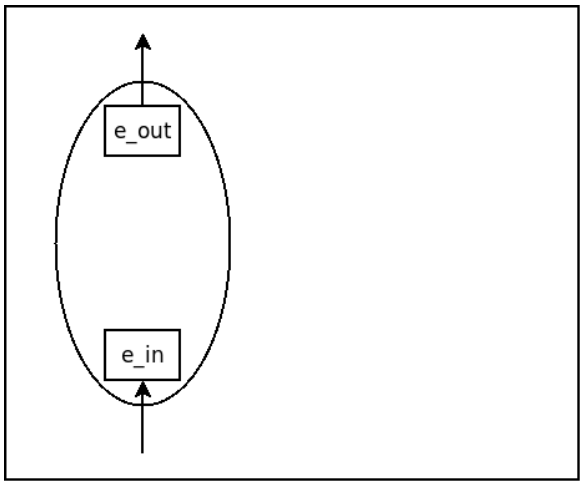
- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end
```

Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

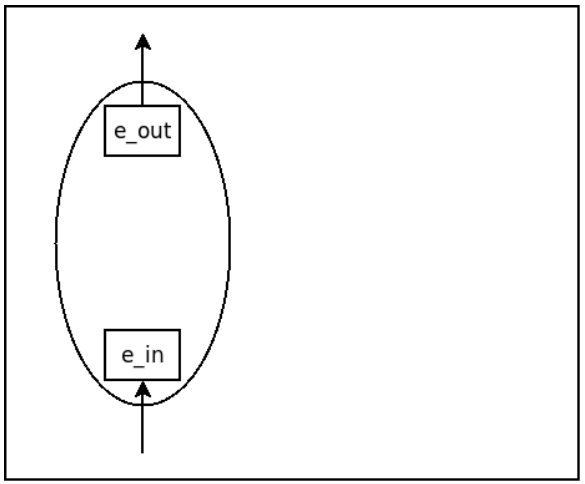
```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

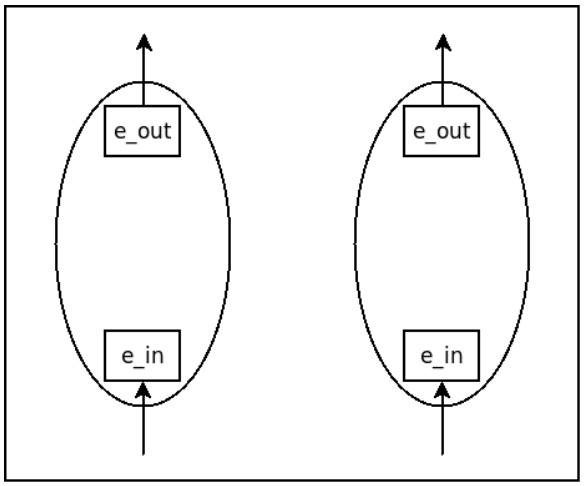
```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

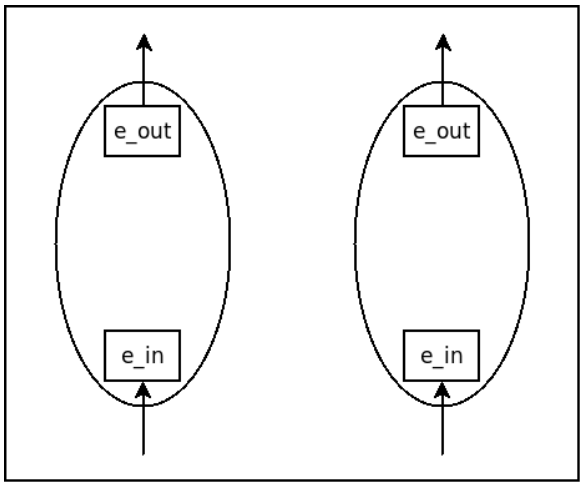
```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

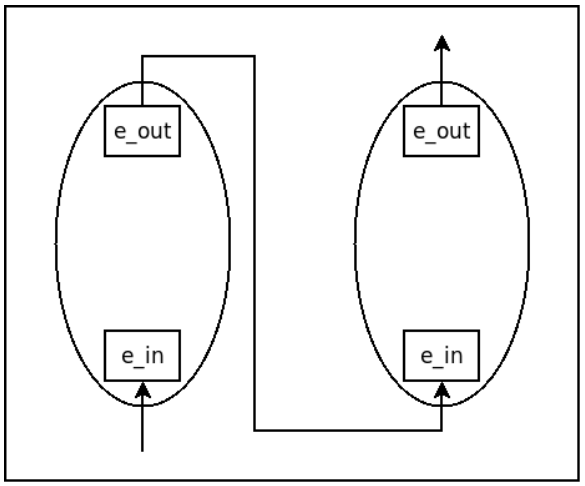
```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;


    every v in io1.e_out do
        emit io2.e_in => v;
    end
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

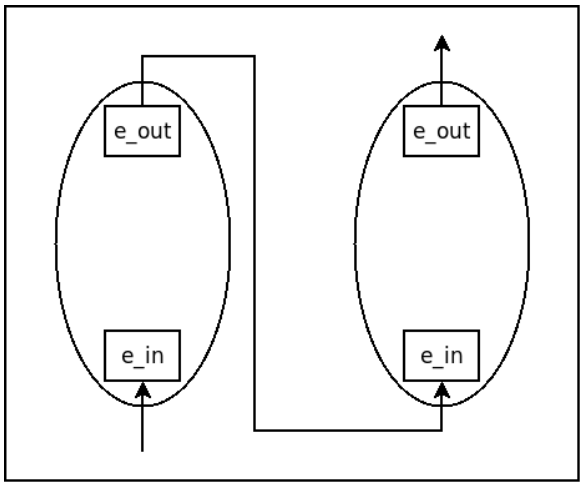
```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;


    every v in io1.e_out do
        emit io2.e_in => v;
    end
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

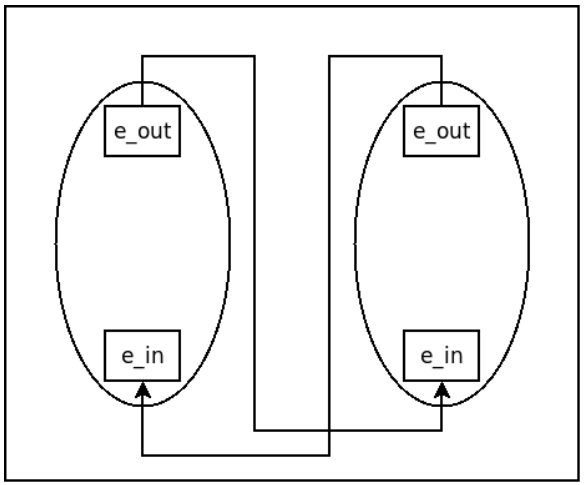
```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;


    every v in io1.e_out do
        emit io2.e_in => v;
    end

    every v in io2.e_out do
        emit io1.e_in => v;
    end
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

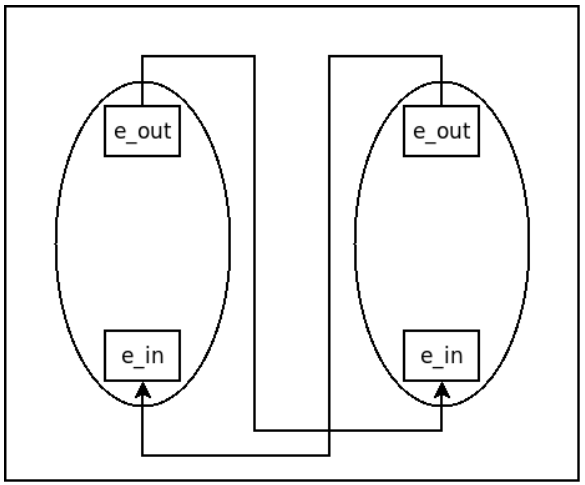
```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;


    every v in io1.e_out do
        emit io2.e_in => v;
    end

    every v in io2.e_out do
        emit io1.e_in => v;
    end
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

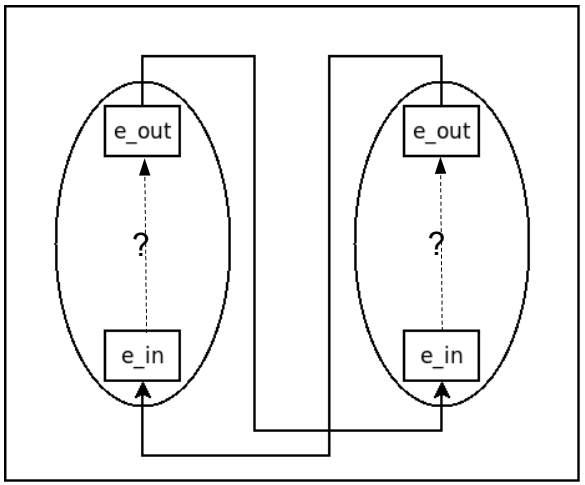
```
class I0 with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var I0 io1, io2;

par do
    every v in io1.e_out do
        emit io2.e_in => v;
    end
with
    every v in io2.e_out do
        emit io1.e_in => v;
    end
end
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

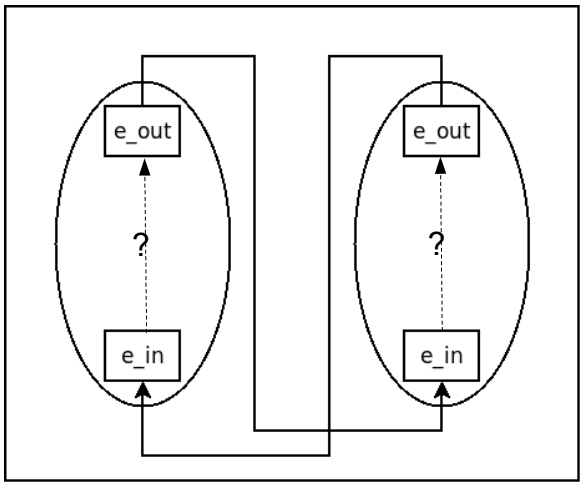
```
class I0 with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var I0 io1, io2;

par do
    every v in io1.e_out do
        emit io2.e_in => v;
    end
with
    every v in io2.e_out do
        emit io1.e_in => v;
    end
end
```



Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;

par do
    every v in io1.e_out do
        emit io2.e_in => v;
    end
with
    every v in io2.e_out do
        emit io1.e_in => v;
    end
end
```

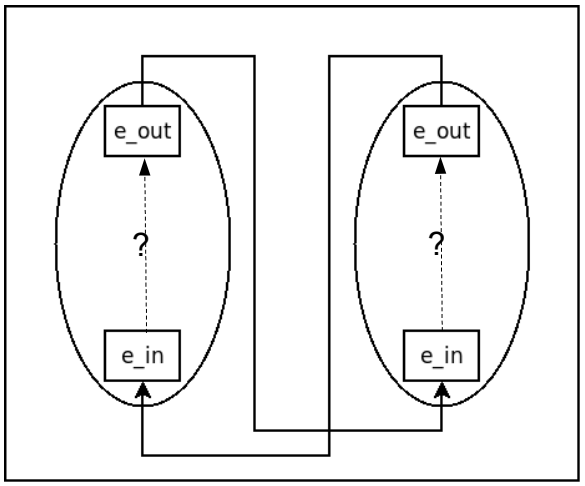



```
class Player with
    <...>
    var IO&  io;
do
    <...>



    <...>
end
```

Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;

par do
    every v in io1.e_out do
        emit io2.e_in => v;
    end
with
    every v in io2.e_out do
        emit io1.e_in => v;
    end
end
```

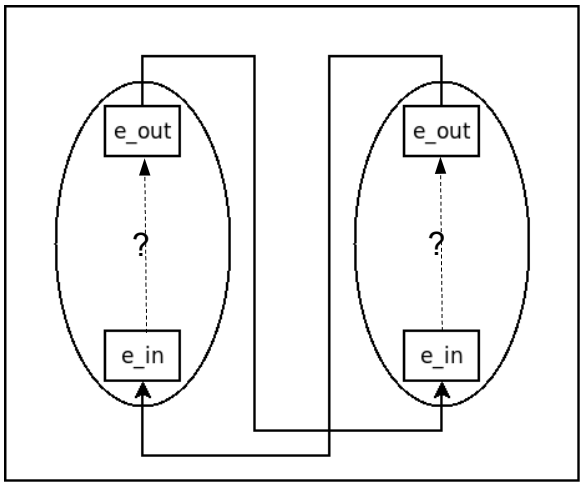



```
class Player with
    <...>
    var IO&  io;
do
    <...>



    <...>
end

var Player p1(..., io1);
var Player p2(..., io2);
```

Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;

par do
    every v in io1.e_out do
        emit io2.e_in => v;
    end
with
    every v in io2.e_out do
        emit io1.e_in => v;
    end
end
```

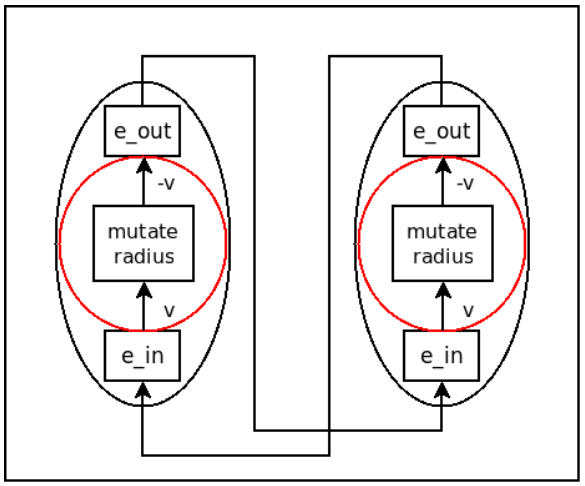



```
class Player with
    <...>
    var IO&  io;
do
    <...>
        every v in io.e_in do
            ball.radius += v;
            emit io.e_out => -v;
        end
    <...>
end

var Player p1(..., io1);
var Player p2(..., io2);
```

Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

```
class IO with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var IO io1, io2;

par do
    every v in io1.e_out do
        emit io2.e_in => v;
    end
with
    every v in io2.e_out do
        emit io1.e_in => v;
    end
end
```

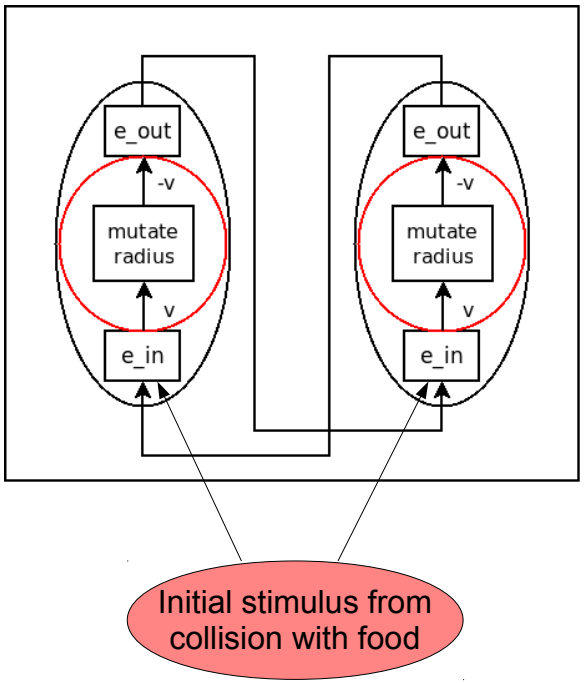



```
class Player with
    <...>
    var IO&  io;
do
    <...>
        every v in io.e_in do
            ball.radius += v;
            emit io.e_out => -v;
        end
    <...>
end

var Player p1(..., io1);
var Player p2(..., io2);
```

Push example

- 2-player game
  - food increases ball radius
  - constraint: sum of radius (p1+p2) is constant
  - restarts when any radius reaches 0

```
class I0 with
    event int e_in;
    event int e_out;
do
    await FOREVER;
end

var I0 io1, io2;

par do
    every v in io1.e_out do
        emit io2.e_in => v;
    end
with
    every v in io2.e_out do
        emit io1.e_in => v;
    end
end
```




```
class Player with
    <...>
    var I0&  io;
do
    <...>
        every v in io.e_in do
            ball.radius += v;
            emit io.e_out => -v;
        end
    <...>
end

var Player p1(..., io1);
var Player p2(..., io2);
```

```
// restarts every time p1 or p2 dies
loop do









end
```

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2




end
```

```
// restarts every time p1 or p2 dies
loop do
    var IO io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);









end
```

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;









end
```

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do









    end
end
```

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do

            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end

    end
end
```

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
        end
    end
end
```

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
            // checks for collisions every frame
            every NEXT_FRAME do









            end
        end
    end
end
```

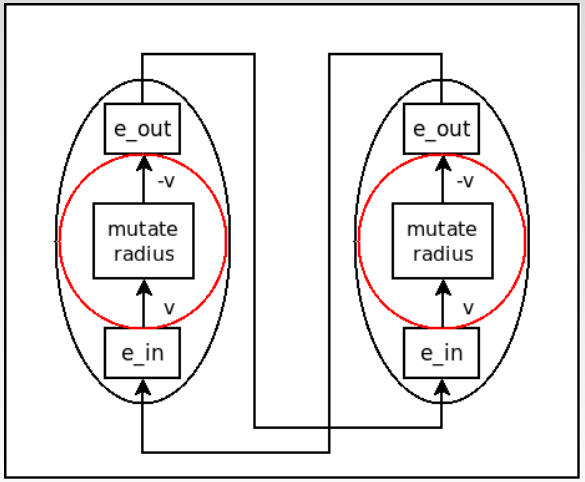
```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
            // checks for collisions every frame
            every NEXT_FRAME do
                // iterates over the foods
                loop food in foods do









                end
            end
        end
    end
end
```

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
            // checks for collisions every frame
            every NEXT_FRAME do
                // iterates over the foods
                loop food in foods do
                    // on collision
                    if <collision-food-vs-p1> then
                        // pushes data
                        emit p1.io.e_in => food:ball.radius;
                        // removes the food from the pool
                        kill food;
                    end
                end
            end
        end
    end
end
```

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
            // checks for collisions every frame
            every NEXT_FRAME do
                // iterates over the foods
                loop food in foods do
                    // on collision
                    if <collision-food-vs-p1> then
                        // pushes data
                        emit p1.io.e_in => food:ball.radius;
                        // removes the food from the pool
                        kill food;
                    end
                end
            end
        end
    end
end
```

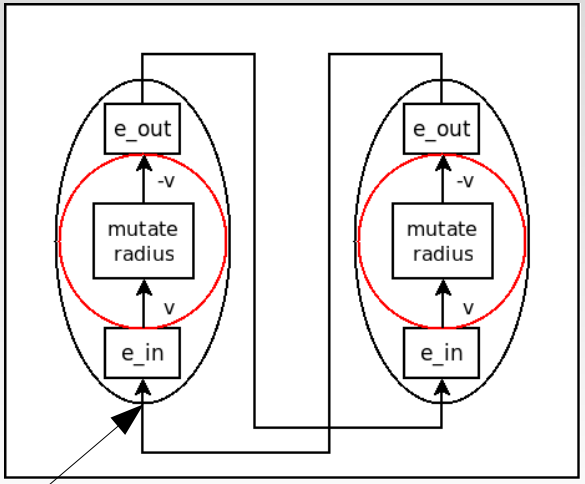
(same code for player 2)

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
            // checks for collisions every frame
            every NEXT_FRAME do
                // iterates over the foods
                loop food in foods do
                    // on collision
                    if <collision-food-vs-p1> then
                        // pushes data
                        emit p1.io.e_in => food:ball.radius;
                        // removes the food from the pool
                        kill food;
                    end
                end
            end
        end
    end
end
```



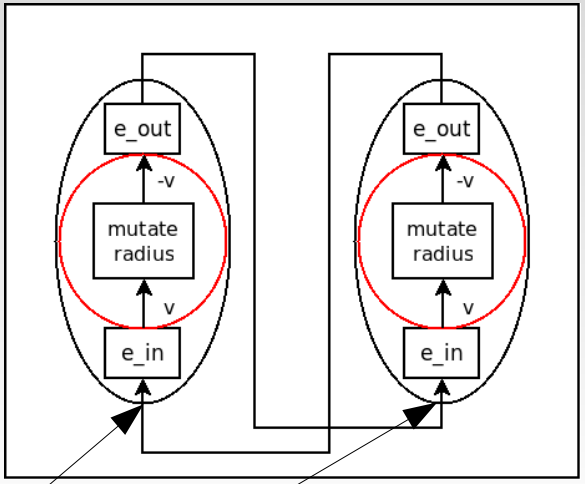
(same code for player 2)

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
            // checks for collisions every frame
            every NEXT_FRAME do
                // iterates over the foods
                loop food in foods do
                    // on collision
                    if <collision-food-vs-p1> then
                        // pushes data
                        emit p1.io.e_in => food:ball.radius;
                        // removes the food from the pool
                        kill food;
                    end
                end
            end
        end
    end
end
```



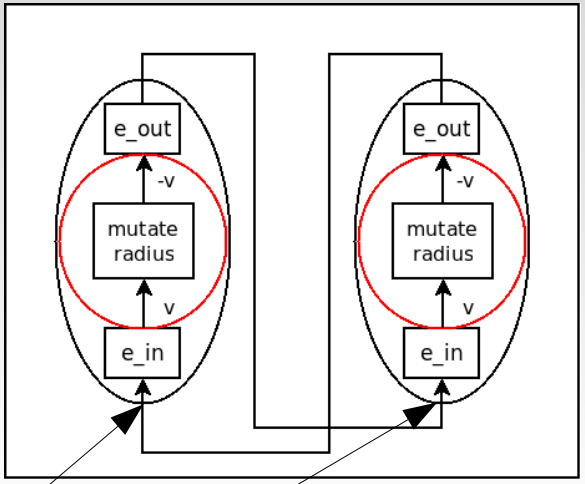
(same code for player 2)

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
            // checks for collisions every frame
            every NEXT_FRAME do
                // iterates over the foods
                loop food in foods do
                    // on collision
                    if <collision-food-vs-p1> then
                        // pushes data
                        emit p1.io.e_in => food:ball.radius;
                        // removes the food from the pool
                        kill food;
                    end
                end
            end
        end
    end
end
```



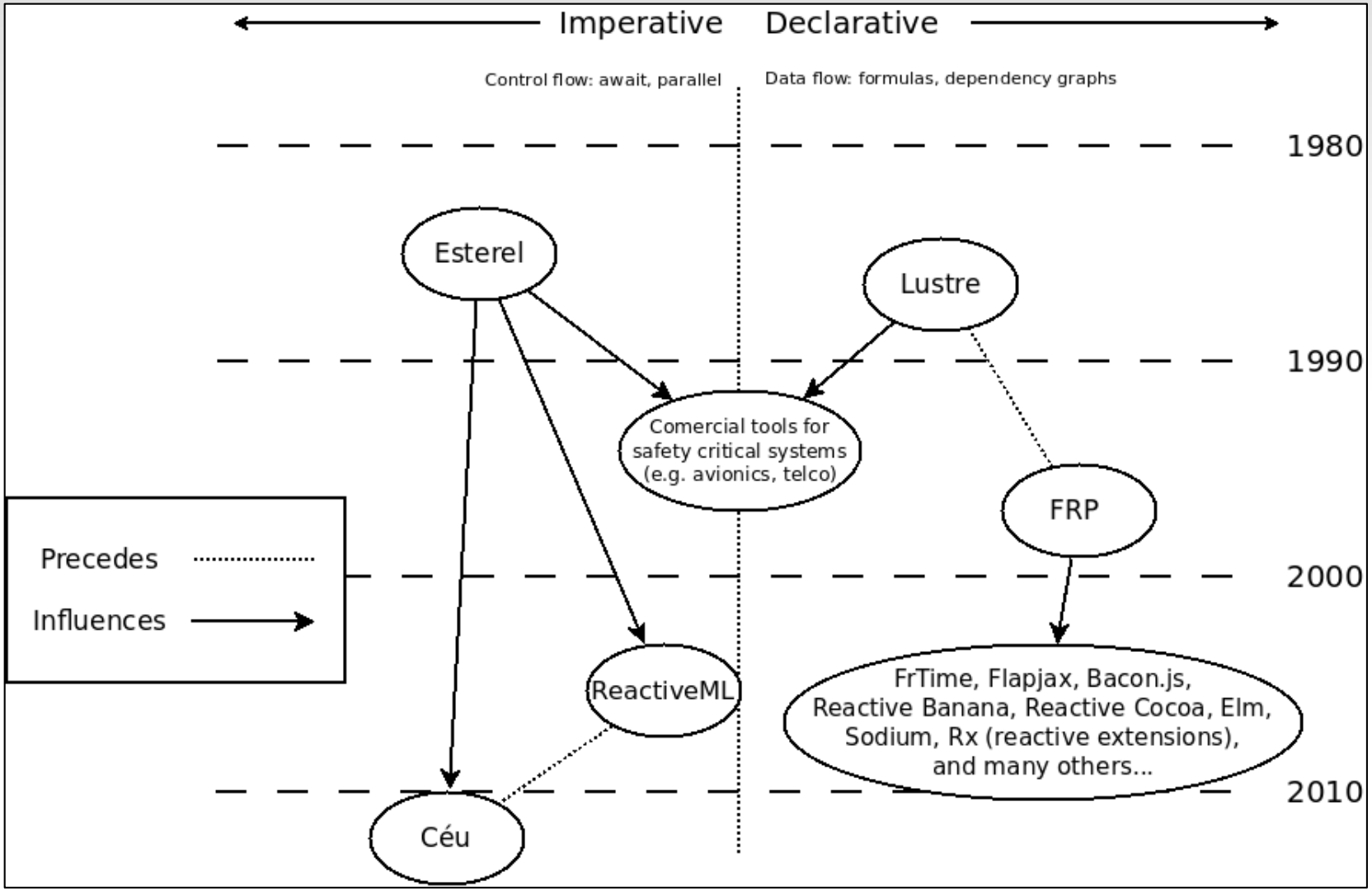
(same code for player 2)

```
// restarts every time p1 or p2 dies
loop do
    var I0 io1, io2;
    <...>   // links io1<=>io2

    var Player p1(..., io1);
    var Player p2(..., io2);

    // holds all dynamic instances in a lexical scope
    pool Food[] foods;

    // aborts whenever p1 or p2 dies
    watching p1,p2 do
        par do
            // creates a new food into the pool periodically
            every <random> ms do
                spawn Food(...) in foods;
            end
        with
            // checks for collisions every frame
            every NEXT_FRAME do
                // iterates over the foods
                loop food in foods do
                    // on collision
                    if <collision-food-vs-p1> then
                        // pushes data
                        emit p1.io.e_in => food:ball.radius;
                        // removes the food from the pool
                        kill food;
                    end
                end
            end
        end
    end
end
```

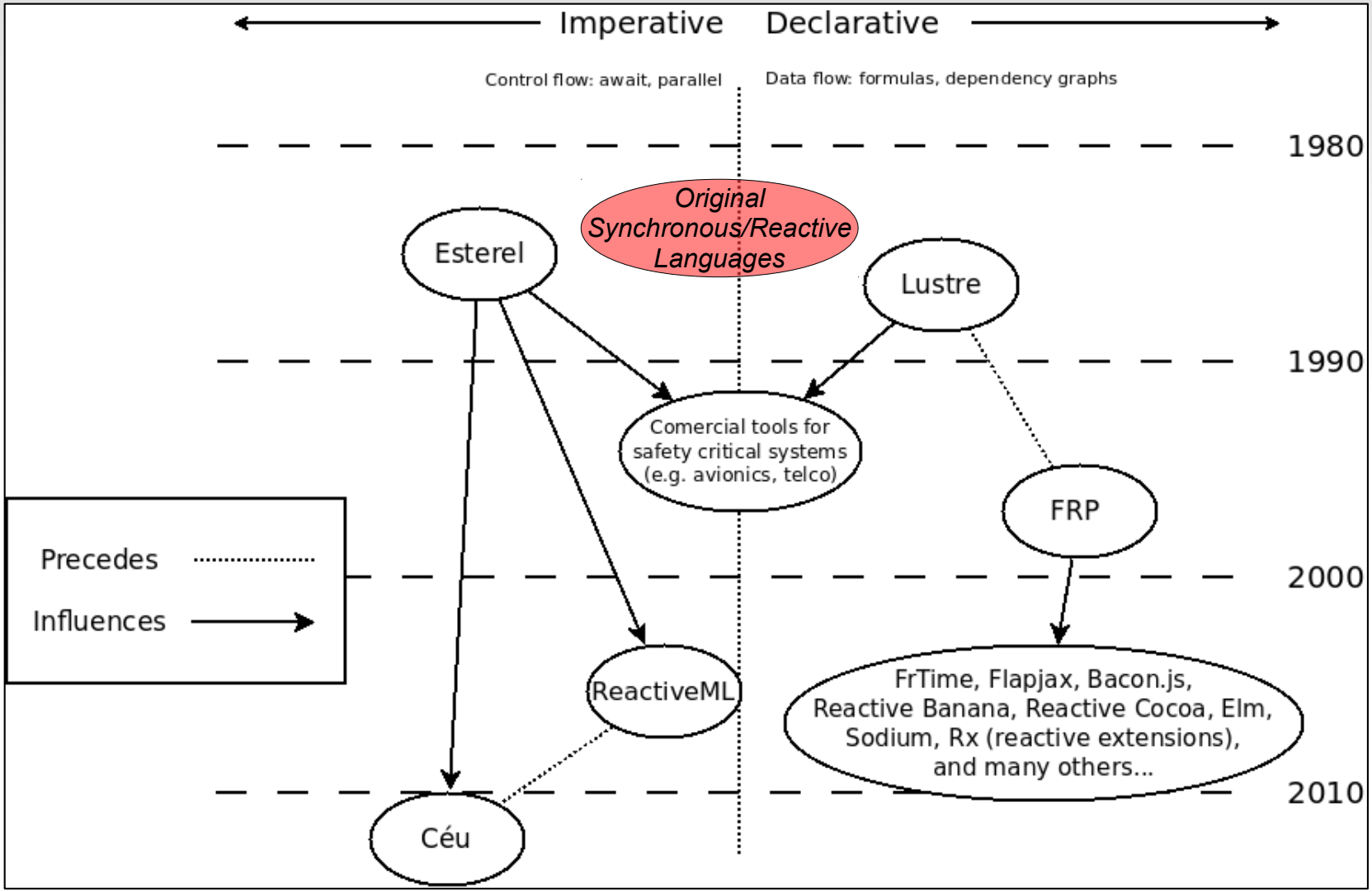


(same code for player 2)

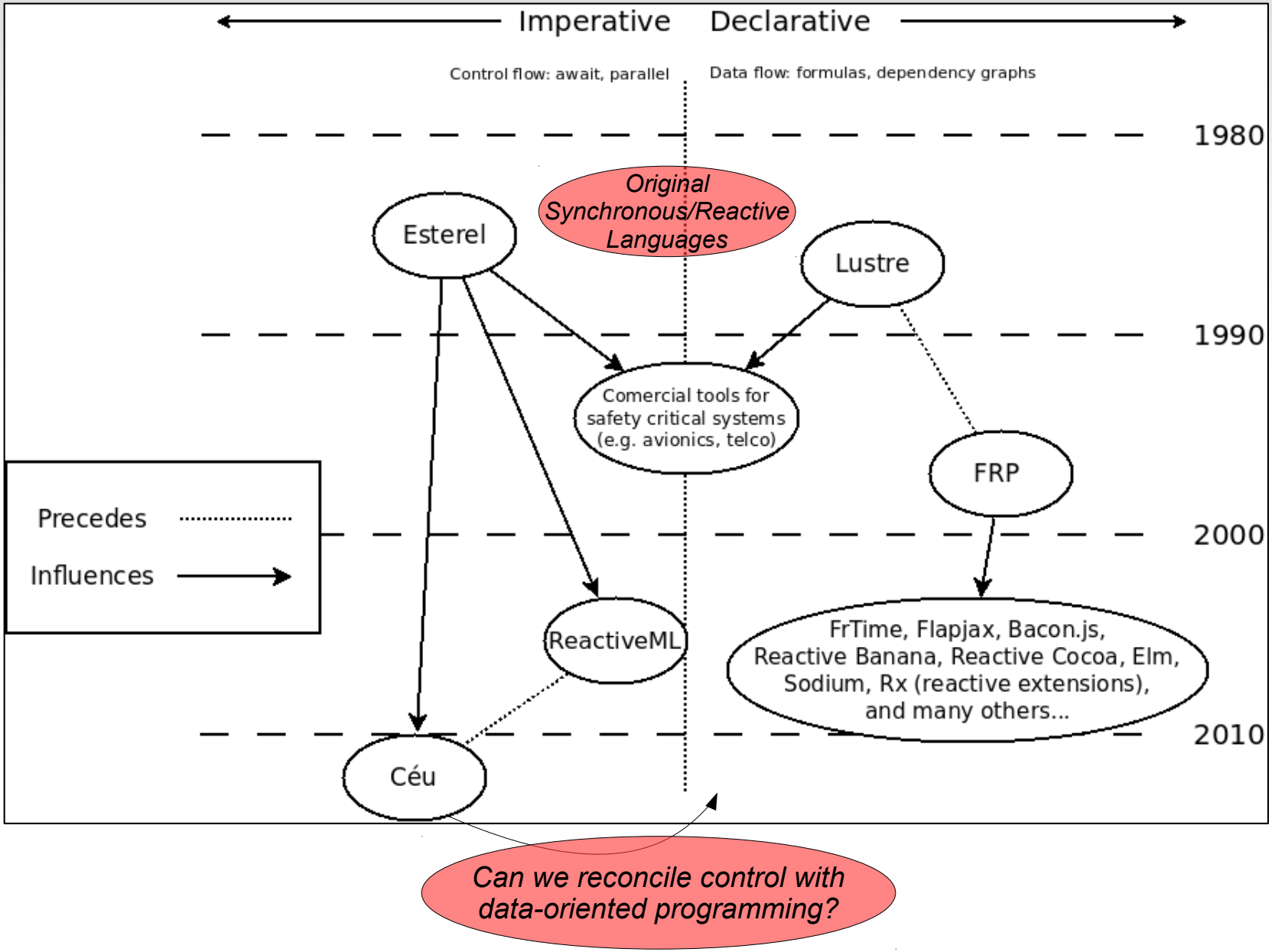
Synchronous/Reactive Languages



Synchronous/Reactive Languages



Synchronous/Reactive Languages



Design guidelines / Tradeoffs in Céu

Design guidelines / Tradeoffs in Céu

- Structured programming

(vs data streams/signals + functional combinators)

Design guidelines / Tradeoffs in Céu

- Structured programming *(vs data streams/signals + functional combinators)*
- Side effects everywhere *(vs purity / immutable data)*

Design guidelines / Tradeoffs in Céu

- Structured programming *(vs data streams/signals + functional combinators)*
- Side effects everywhere *(vs purity / immutable data)*
- Sequential/Deterministic semantics *(vs potential parallelism)*
  (real parallelism relies on asynchronous primitives)
  (feasible in real-time reactive apps?)

Design guidelines / Tradeoffs in Céu

- Structured programming *(vs data streams/signals + functional combinators)*
- Side effects everywhere *(vs purity / immutable data)*
- Sequential/Deterministic semantics *(vs potential parallelism)*
  (real parallelism relies on asynchronous primitives)
  (feasible in real-time reactive apps?)
- Static memory management *(vs garbage collection)*
  (lexical scope + safe abortion)

Design guidelines / Tradeoffs in Céu

- Structured programming *(vs data streams/signals + functional combinators)*
- Side effects everywhere *(vs purity / immutable data)*
- Sequential/Deterministic semantics *(vs potential parallelism)*
  (real parallelism relies on asynchronous primitives)
  (feasible in real-time reactive apps?)
- Static memory management *(vs garbage collection)*
  (lexical scope + safe abortion)
- Bounded memory and execution time *(vs hidden costs)*

Design guidelines / Tradeoffs in Céu

- Structured programming *(vs data streams/signals + functional combinators)*
- Side effects everywhere *(vs purity / immutable data)*
- Sequential/Deterministic semantics *(vs potential parallelism)*
  (real parallelism relies on asynchronous primitives)
  (feasible in real-time reactive apps?)
- Static memory management *(vs garbage collection)*
  (lexical scope + safe abortion)
- Bounded memory and execution time *(vs hidden costs)*

- Data flow
  - explicit loops with side effects
- Control flow
  - explicit state machines (*fold* combinator)

Summary

Summary

- From *"Structured Programming"* to
  "Structured Synchronous/Reactive Programming"

Summary

- From *"Structured Programming"* to
  "Structured Synchronous/Reactive Programming"

- *Data-oriented* and *Control-oriented* programming
  are useful and complementary
  - declarative / imperative dichotomy (since the '80s)

Summary

- From *"Structured Programming"* to
  "Structured Synchronous/Reactive Programming"
- *Data-oriented* and *Control-oriented* programming
  are useful and complementary
  - declarative / imperative dichotomy (since the '80s)
- Let's not give up on *imperative programming*!

Summary

- From *"Structured Programming"* to
  "Structured Synchronous/Reactive Programming"
- *Data-oriented* and *Control-oriented* programming
  are useful and complementary
  - declarative / imperative dichotomy (since the '80s)
- Let's not give up on *imperative programming*!
  - Side effects can be tamed:
    1. avoid control variables
    2. use deep nesting of scopes
    3. rely on deterministic semantics

Structured Synchronous Programming with Céu

(mixing control with data flow)

Francisco Sant'Anna


francisco.santanna@gmail.com


@fsantanna_puc

(I'll tweet the slides here!)