

Linguagem de Programação



LabLua – PUC-Rio

www.lua.inf.puc-rio.br

*Francisco Sant'Anna
Noemi Rodrigues
Roberto Ierusalimschy*

“Hello world!”

- Piscando LEDs

1. *on \leftrightarrow off a cada 500ms*
2. *Parar após 5s*
3. *Repetir após 2s*

- Composições

- par, seq, loop
- ~~variáveis de estado~~

```
loop do
  par/or do
    loop do
      await 500ms;
      _leds_toggle();
    end
  with
    await 5s;
  end
  await 2s;
end
```

Visão geral de Céu

- Reativa
 - Ambiente no controle: *eventos*
- Imperativa
 - Sequências, loops e atribuições
- Concorrente
 - Múltiplas linhas de execução: *trails (trilhas)*
- Síncrona
 - Trilhas são sincronizadas a cada evento
 - **Trilhas estão sempre esperando**
- Determinística
 - Mesmo comportamento para uma dada linha de tempo

Modelos de Concorrência

■ Assíncrono

- Atividades independentes
 - Requer primitivas de sincronização (*mutexes*, *monitors*, *channels*, *pipes*)
-
- Threads (Java, p-threads)
 - Processes (UNIX)
 - Actors (erlang)

■ Síncrono

- Execução em passos
 - Requer primitivas assíncronas (não bloqueantes)
-
- Loop do Arduino
 - Event-driven (callbacks)
 - *Céu*

Modelos de Concorrência

- Assíncrono

- programação estruturada
- sincronismo explícito
- implementação complexa
- execução paralela**

VS

- Síncrono

- programação não estruturada
- livre de sincronização**
- footprint* pequeno
- execução serializada

- programação estruturada
- livre de sincronização
- footprint* pequeno
- execução paralela

?

Exercício 1 (revisão-da-revisão)

- Piscar o LED a cada 1 segundo
- Parar ao pressionar o botão, mantendo o LED aceso para sempre

```
void loop () {  
  unsigned long now;  
  if (now >= old+1000) {  
    old = now;  
    state = !state;  
    digitalWrite(LED_PIN, state);  
  }  
  int but = digitalRead(BUT_PIN);  
  if (but) {  
    digitalWrite(LED_PIN, HIGH);  
    while(1);  
  }  
}
```

```
input int PIN02;  
output int PIN13;  
par/or do
```

```
  loop do  
    emit PIN13 => _HIGH;  
    await 1s;  
    emit PIN13 => _LOW;  
    await 1s;  
  end
```

```
with
```

```
  loop do  
    var int on? = await PIN02;  
    if on? then  
      break;  
    end  
  end
```

```
end
```

```
emit PIN13 => _HIGH;
```

```
void Thread1 (void) {
```

```
  digitalWrite(LED_PIN, HIGH);  
  delay(1000);  
  digitalWrite(LED_PIN, LOW);  
  delay(1000);
```

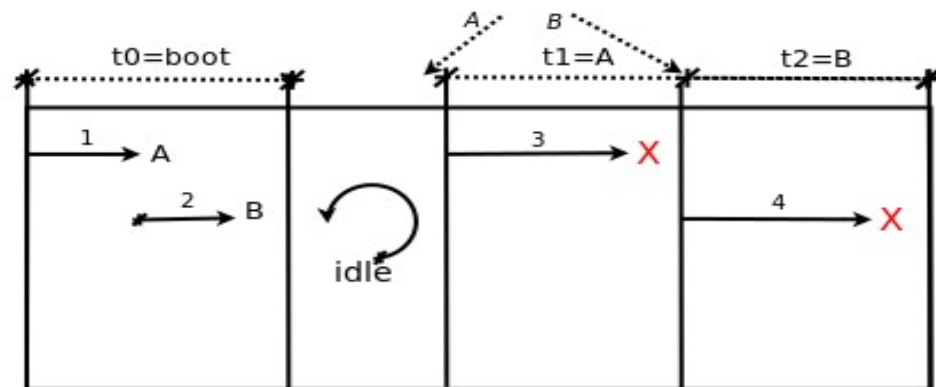
```
}
```

```
  if (digitalRead(BUT_PIN) == HIGH) {  
    digitalWrite(LED_PIN, HIGH);  
  }
```

Céu: Modelo de Execução

1. O programa inicia na “reação de *boot*” em apenas uma trilha.
2. Trilhas ativas executam até esperarem ou terminarem. Esse passo é conhecido como “reação” e sempre executa em tempo *bounded*.
3. A ocorrência de um evento de entrada acorda **todas** as trilhas esperando aquele evento. Repete o “passo 2”.

```
par/and do
  <...>      // 1
  await A;
  <...>      // 3
with
  <...>      // 2
  await B;
  <...>      // 4
end
```



<...> são segmentos de trilha que não esperam
(e.g. atribuições, chamadas de função)

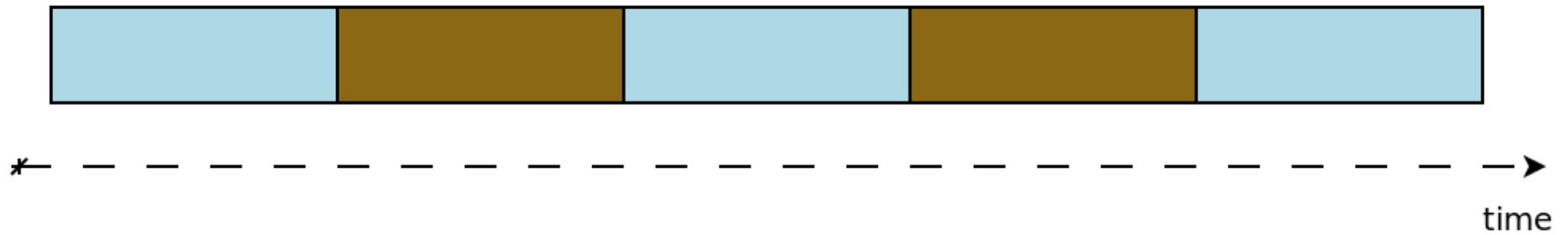
Céu: Modelo de Execução

- Hipótese de sincronismo: “Reações executam infinitamente mais rápido do que a taxa de eventos.”
- *Na teoria*: o programa não leva tempo no “passo 2” e está sempre ocioso no “passo 3”.
- *Na prática*: se um evento ocorre enquanto uma reação executa, ele é enfileirado para uma reação posterior.
- Reações a eventos nunca são sobrepostas (modelo síncrono)
- Quando múltiplas trilhas estão ativas (i.e., acordadas pelo mesmo evento), elas são executadas na ordem em que aparecem no código do programa.

Síncrono

Activity 1

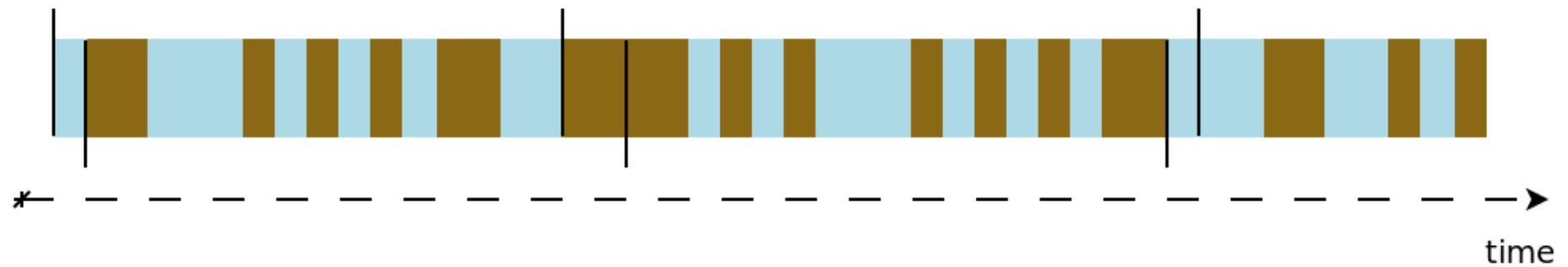
Activity 2



Assíncrono

Activity 1

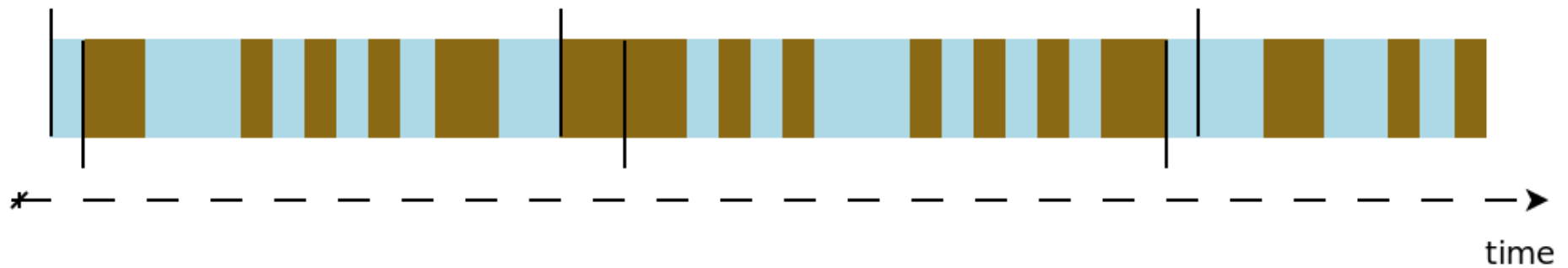
Activity 2



Assíncrono / Preemptivo

Activity 1

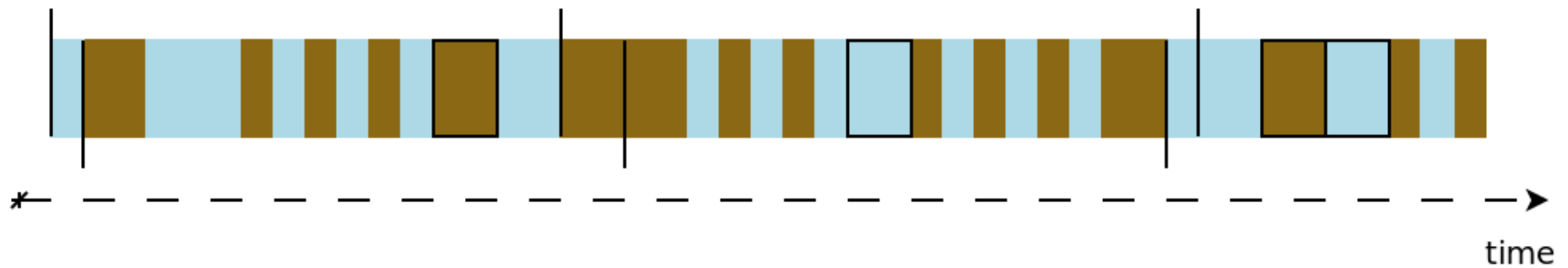
Activity 2



Assíncrono / Preemptivo / c/ Locks

Activity 1

Activity 2



Exercício 1 (revisão-da-revisão)

```
input  int PIN02;
output int PIN13;
par/or do
  loop do
    emit PIN13 => _HIGH;
    await 1s;
    emit PIN13 => _LOW;
    await 1s;
  end
with
  loop do
    var int on? = await PIN02;
    if on? then
      break;
    end
  end
end
emit PIN13 => _HIGH;
```

```
lock();
input  int PIN02;
output int PIN13;
par/or do
  loop do
    emit PIN13 => _HIGH;
    unlock();
    await 1s;
    lock();
    emit PIN13 => _LOW;
    unlock();
    await 1s;
  end
with
  loop do
    unlock();
    var int on? = await PIN02;
    lock();
    if on? then
      break;
    end
  end
end
emit PIN13 => _HIGH;
```


Tarefa-02 (revisão)

- Piscar o LED a cada 1 segundo
- Botão 1: Acelerar o pisca-pisca a cada pressionamento
- Botão 2: Desacelerar a cada pressionamento
- Botão 1+2 (em menos de 500ms): Parar

Tarefa-02 (revisão)

- Piscar o LED a cada 1 segundo
- Botão 1: Acelerar o pisca-pisca a cada pressionamento
- Botão 2: Desacelerar a cada pressionamento
- Botão 1+2 (em menos de 500ms): Parar

```
var int dt = 1000;
par/or do
  loop do
    emit PIN13 => _HIGH;
    await (dt)ms;
    emit PIN13 => _LOW;
    await (dt)ms;
  end
with
  loop do
    await PIN02;
    await 200ms;
    dt = dt - 100;
  end
with
  loop do
    await PIN03;
    await 200ms;
    dt = dt + 100;
  end
with
  loop do
    par/or do
      await PIN02;
      await PIN03;
      break;
    with
      await PIN03;
      await PIN02;
      break;
    with
      par/or do
        await PIN02;
      with
        await PIN03;
      end
      await 500ms;
    end
  end
end
end
```

Exercício - “Reiniciar”

- Reiniciar o comportamento da “Tarefa 2” caso o Botão 1 seja pressionado por 5 segundos.

Compilando / Executando

```
> cd /home/ceu/ceu-arduino  
> make CEUFILE=/home/ceu/reativos/code/arduino/ex_01.ceu  
> make CEUFILE=/home/ceu/reativos/code/arduino/ex_02.ceu
```


SDL: “Input” (Céu)

```
SDL_Event e;
while (1) {
    while (SDL_PollEvent(&e) == 0);
    if (e.type == SDL_QUIT) {
        break;
    } else if (e.type == SDL_KEYDOWN) {
        switch (e.key.keysym.sym) {
            case SDLK_UP:
                r.y -= 10;
            case SDLK_DOWN:
                r.y += 10;
            case SDLK_LEFT:
                r.x -= 10;
            case SDLK_RIGHT:
                r.x += 10;
        }
    }
    SDL_SetRenderDrawColor(renderer, ...);
    SDL_RenderFillRect(renderer, NULL);
    SDL_SetRenderDrawColor(renderer, ...);
    SDL_RenderFillRect(renderer, &r);
    SDL_RenderPresent(renderer);
}
```

```
input void                                SDL_QUIT;
input _SDL_KeyboardEvent* SDL_KEYDOWN;
input void                                SDL_REDRAW;

par/or do
    await SDL_QUIT;
with
    var _SDL_KeyboardEvent* key;
    every key in SDL_KEYDOWN do
        if key:keysym.sym == _SDLK_UP then
            r.y = r.y - 10;
        else/if key:keysym.sym == _SDLK_DOWN then
            r.y = r.y + 10;
        else/if key:keysym.sym == _SDLK_LEFT then
            r.x = r.x - 10;
        else/if key:keysym.sym == _SDLK_RIGHT then
            r.x = r.x + 10;
        end
    end
with
    every SDL_REDRAW do
        _SDL_SetRenderDrawColor(renderer, ...);
        _SDL_RenderFillRect(renderer, null);
        _SDL_SetRenderDrawColor(renderer, ...);
        _SDL_RenderFillRect(renderer, &r);
        _SDL_RenderPresent(renderer);
    end
end
```

SDL: “Animação” (Céu)

(code/sdl/ex_01.ceu)

```
par/or do
  loop do
    loop i in 100 do
      await 10ms;
      r1.x = r1.x + 1;
    end
    loop i in 100 do
      await 10ms;
      r1.y = r1.y + 1;
    end
    loop i in 100 do
      await 10ms;
      r1.x = r1.x - 1;
    end
    loop i in 100 do
      await 10ms;
      r1.y = r1.y - 1;
    end
  end
with
  var _SDL_MouseButtonEvent* but;
  but = await SDL_MOUSEBUTTONDOWN
    until _SDL_Rect_vs_Mouse(&r1, but);
end
```

Exercício - “Retomar”

- Retomar o movimento do retângulo após clicar novamente nele.

Compilando / Executando

```
> cd /home/ceu/ceu-sdl  
  
> make CEUFILE=/home/ceu/reativos/code/sdl/01_input.ceu  
> /home/ceu/reativos/code/sdl/01_input.exe  
  
> make CEUFILE=/home/ceu/reativos/code/sdl/ex_01.ceu  
> /home/ceu/reativos/code/sdl/ex_01.exe
```


Documentação

- Tutorial online

- <http://www.ceu-lang.org/try.php>

- Manual de referência

- <https://github.com/fsantanna/ceu/blob/master/manual/manual-toc.md>

- Exemplos para Arduino e SDL

- <https://github.com/fsantanna/ceu-arduino/tree/master/samples>
 - <https://github.com/fsantanna/ceu-sdl/tree/master/samples>

- Grupo da Turma (!)

Artigos & Videos - 05

- Safe Concurrent Abstractions for Wireless Sensor Networks
 - http://ceu-lang.org/chico/ceu_sensys13_pre.pdf
- Dynamic Organisms in Céu
 - <http://vimeo.com/96915338>

par/and, par/or, par

```
loop do
  par/and do
    ...
  with
    await 1s;
  end
end
// SAMPLING ("at least")
```

```
loop do
  par/or do
    ...
  with
    await 1s;
  end
end
// WATCHDOG ("at most")
```

1st class Timers

- Timers are very common in embedded apps

```
loop do
  await 2min30s400ms;
  ... // do something
end
```

```
loop do
  par/or do
    // do something
  with
    await 500ms;
  end
end
```

1st class Timers

- $1s + 1s = 2s$ (?!)

```
loop do
  await 10ms;
  _printf("1\n");
  await 1ms;
  _printf("2\n");
end
```

- 15ms elapse
- late = 5ms
- late = 4ms

```
par/or do
  loop do
    await 10ms;
    _printf("1\n");
    await 1ms;
    _printf("2\n");
  end
  with
    await 15ms;
    _printf("3\n");
  end
end
```

1st class Timers

- Temporal Analysis

```
var int v;  
par/or do  
  loop do  
    await 10ms;  
    v = 1;  
  end  
with  
  await 100ms;  
  v = 2;  
end
```

Summary

- Safety
 - deterministic execution
 - sync-free concurrency
- Expressiveness
 - structured / parallel compositions
 - 1st-class timers
- Small footprint
 - ROM: 3 Kbytes
 - RAM: 100 bytes

Graphical Demos

```

input void SDL_QUIT;
input void SDL_REDRAW;

<...>  // initialize graphical library, renderer, etc.

var _SDL_Rect r;
r.w = 100;
r.h = 100;
r.x = 20;
r.y = 200;

par/or do
    every 20ms do
        r.x = r.x + 1;
    end
with
    every SDL_REDRAW do
        _SDL_SetRenderDrawColor(ren, 0xFF, 0xFF, 0xFF, 0);
        _SDL_RenderFillRect(ren, &r);
    end
with
    await SDL_QUIT;
end

```



```

var _SDL_Rect r;
r.w = 100;
r.h = 100;
r.x = 20;
r.y = 100;

var _SDL_Rect s;
s.w = 100;
s.h = 100;
s.x = 20;
s.y = 300;

par/or do
    every 20ms do
        r.x = r.x + 1;
    end
with
    every SDL_REDRAW do
        _SDL_SetRenderDrawColor(ren, 0xFF, 0xFF, 0xFF, 0);
        _SDL_RenderFillRect(ren, &r);
    end
with
    every 10ms do
        s.x = s.x + 1;
    end
with
    every SDL_REDRAW do
        _SDL_SetRenderDrawColor(ren, 0xFF, 0xFF, 0xFF, 0);
        _SDL_RenderFillRect(ren, &s);
    end
with
    await SDL_QUIT;
end

```

class Rect with

```
var _SDL_Rect      r;  
var int            dt;  
var _SDL_Renderer* ren;
```

interface (fields & methods)

do

```
<...>  
par do  
    every (this.dt)ms do  
        r.x = r.x + 1;  
    end  
with  
    every SDL_REDRAW do  
        _SDL_SetRenderDrawColor(ren, 0xFF,0xFF,0xFF, 0);  
        _SDL_RenderFillRect(ren, &r);  
    end  
end
```

body (all Céu statements)

end

par/or do

<...>

with

```
var Rect a with  
    this.r.y = 100;  
    this.dt  = 10;  
    this.ren = ren;  
end;
```

*declaration & constructor
(lexical scope)*

```
var Rect b with  
    this.r.y = 300;  
    this.dt  = 5;  
    this.ren = ren;
```

end;

await FOREVER;

with

await SDL_QUIT;

end

```

class Rect with
  <...>
  var _SDL_Rect      r;
  var int             dt;
  var _SDL_Renderer* ren;
do
  <...>
end

par/or do
  <...>
with
  var int i = 1;
  var Rect[20] rs with
    this.r.y = 20 * i;
    this.dt  = 10 + i;
    this.ren = ren;
    i = i + 1;
  end;
  await FOREVER;
with
  await SDL_QUIT;
end

```

```

class Rect with
  <...>
  var _SDL_Rect      r;
  var int            dt;
  var _SDL_Renderer* ren;
do
  <...>
end

par/or do
  <...>
with
  loop do
    var int i = 1;
    var Rect[20] rs with
      this.r.y = 20 * i;
      this.dt  = 10 + i;
      this.ren = ren;
      i = i + 1;
    end;
    await SDL_MOUSEBUTTONDOWN;
  end
with
  await SDL_QUIT;
end

```

```

class Rect with
    <...>
    var _SDL_Rect      r;
    var int             dt;
    var _SDL_Renderer* ren;
do
    <...>
end

par/or do
    <...>
with
    do
        every 1s do
            spawn Rect with
                this.r.y = 20 + _rand()%400;
                this.dt  = 10 + _rand()%10;
                this.ren = ren;
            end;
        end
    end
with
    await SDL_QUIT;
end

```

```

class Rect with
  <...>
  var _SDL_Rect      r;
  var int            dt;
  var _SDL_Renderer* ren;
do
  <...>
  par/and do
    every (this.dt)ms do
      r.x = r.x + 1;
      if r.x >= 500 then
        break;
      end
    end
  with
    every SDL_REDRAW do
      _SDL_SetRenderDrawColor(ren, 0xFF,0xFF,0xFF, 0);
      _SDL_RenderFillRect(ren, &r);
    end
  end
end

par/or do
  <...>
with
  <...>
    spawn Rect with
      <...>
    end;
with
  await SDL_QUIT;
end

```

```

class Rect with
  <...>
  var _SDL_Rect      r;
  var int            dt;
  var _SDL_Renderer* ren;
do
  <...>
  par/or do
    every (this.dt)ms do
      r.x = r.x + 1;
      if r.x >= 500 then
        break;
      end
    end
  with
    every SDL_REDRAW do
      _SDL_SetRenderDrawColor(ren, 0xFF,0xFF,0xFF, 0);
      _SDL_RenderFillRect(ren, &r);
    end
  end
end

par/or do
  <...>
with
  <...>
    spawn Rect with
      <...>
    end;
with
  await SDL_QUIT;
end

```

```

class Rect with
  <...>
  var _SDL_Rect      r;
  var int            dt;
  var _SDL_Renderer* ren;
do
  <...>
end

par/or do
  <...>
with
  loop do
    par/or do
      do
        every 1s do
          spawn Rect with
            this.r.y = 20 + _rand()%400;
            this.dt  = 10 + _rand()%10;
            this.ren = ren;
          end;
        end
      end
    with
      await SDL_MOUSEBUTTONDOWN;
    end
  end
end
with
  await SDL_QUIT;
end

```


Future work...

- Distributed events
 - `await <MACHINE>.<EVENT>`
- Reactive OS
 - Microkernel for Arduino
- General purpose
 - Games? Webapps?