Módulo de Procedimentos Automatizados

Uma Plataforma de Automação e Controle de Processos Industriais

Equipe MPA
mpa@tecgraf.puc-rio.br

Tecgraf/PUC-Rio
e
PETROBRAS

5 de outubro de 2009

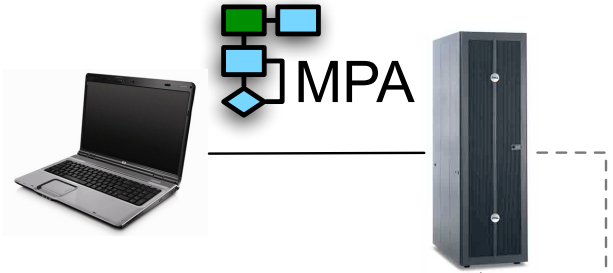
Agenda

Tecgraf PUC-RIO

Automação e Controle de Processos Industriais

Controle Avançado

- Automação de tarefas rotineiras
- Diagnóstico de cenários previstos
- Otimização de processos
- Auxílio em emergências



Supervisão

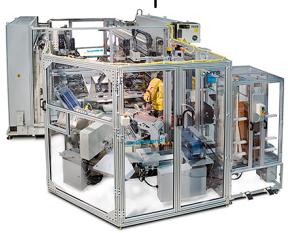
- Atuação de um operador humano capaz de tomar decisões em situações emergenciais.

PLC - Programmable Logic Controller

- Intertravamento de segurança
- Funções básicas de operação

Equipamentos

- Atuação e sensoriamento do processo
- Operações básicas de segurança



Tecgraf PUC-RIO

Histórico

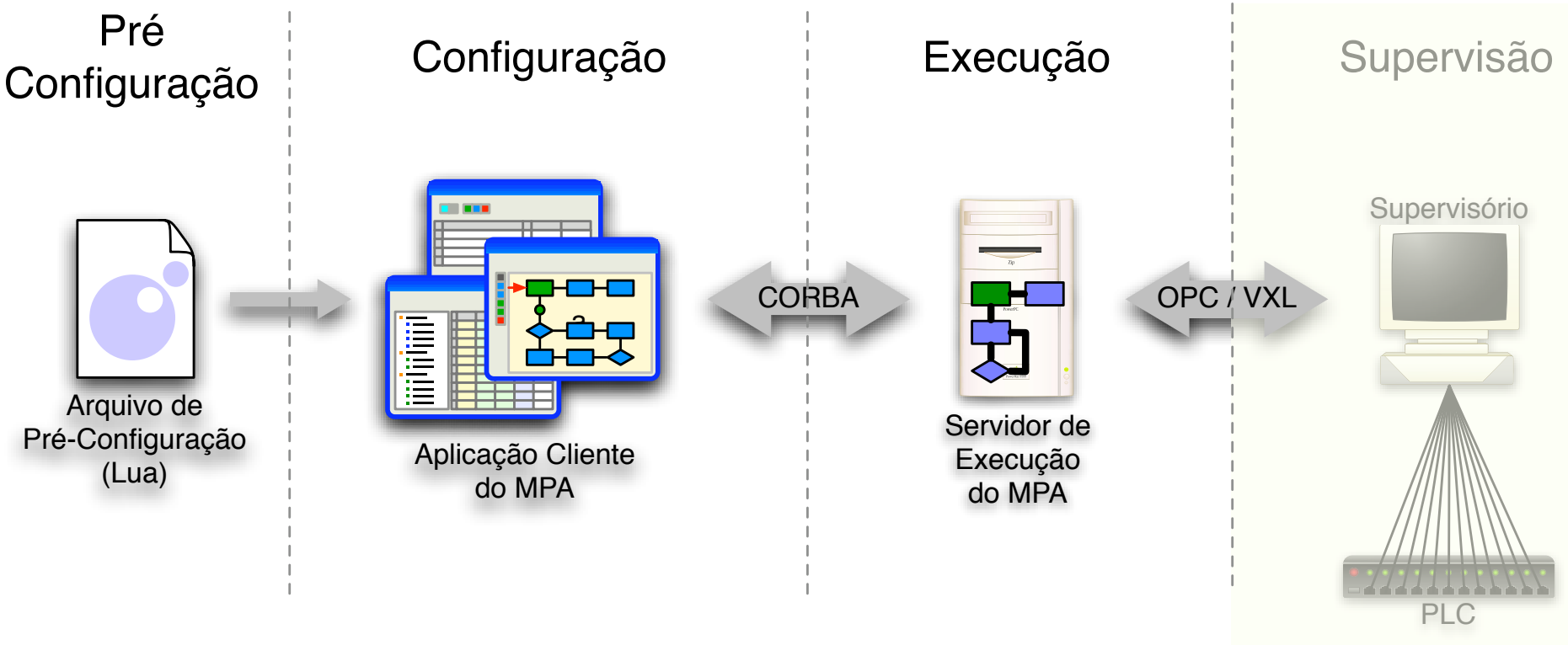
Origens

- Desenvolvido pelo CENPES na plataforma Gensym G2.
- Módulo de Partida Assistida de uma plataforma de extração de petróleo.

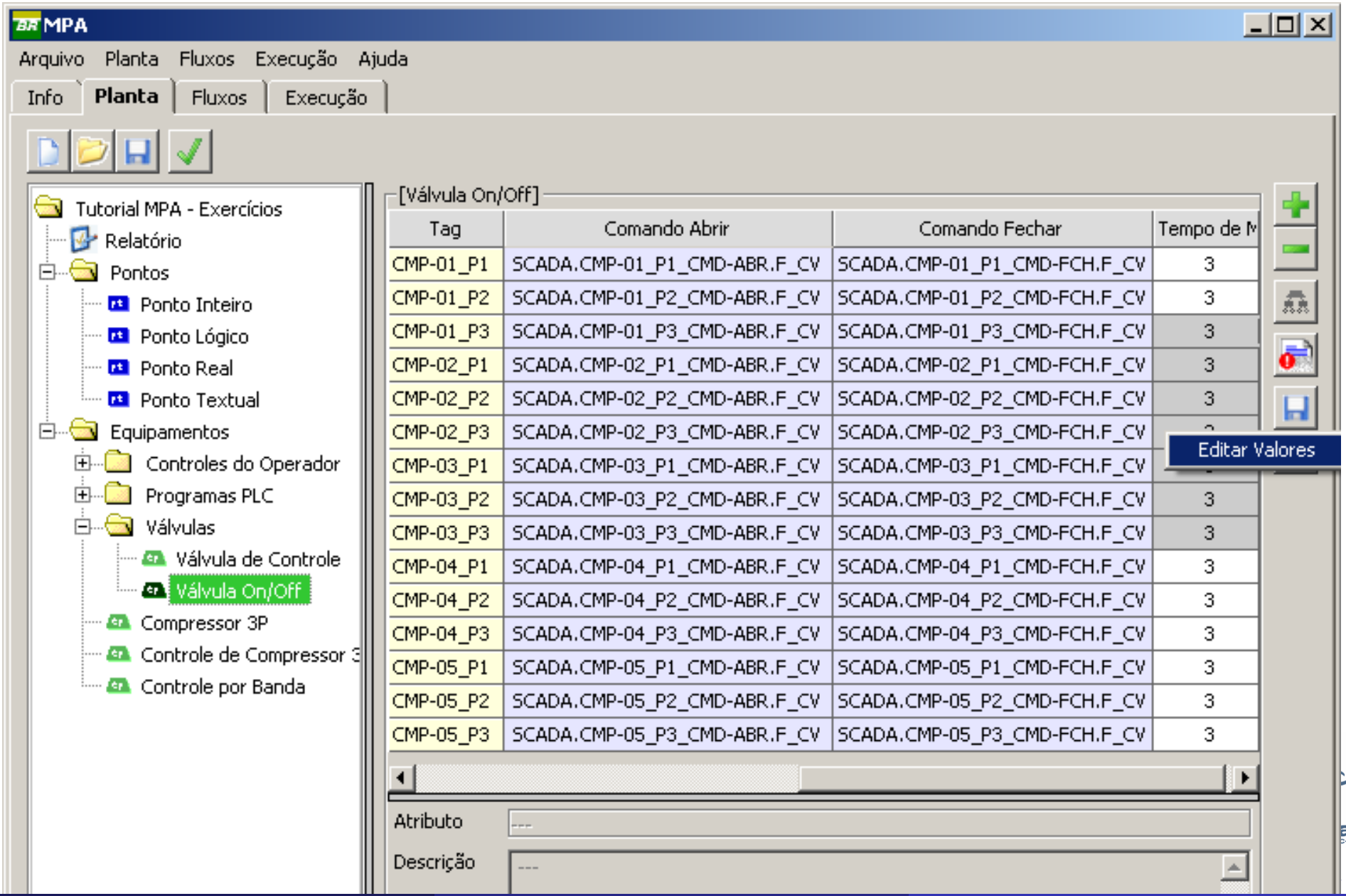
Tecgraf/PUC-Rio

- Generalização para ser adaptado para diferentes plataformas de extração de petróleo e ser programável através de uma linguagem visual.
- Independência da plataforma Gensym G2.

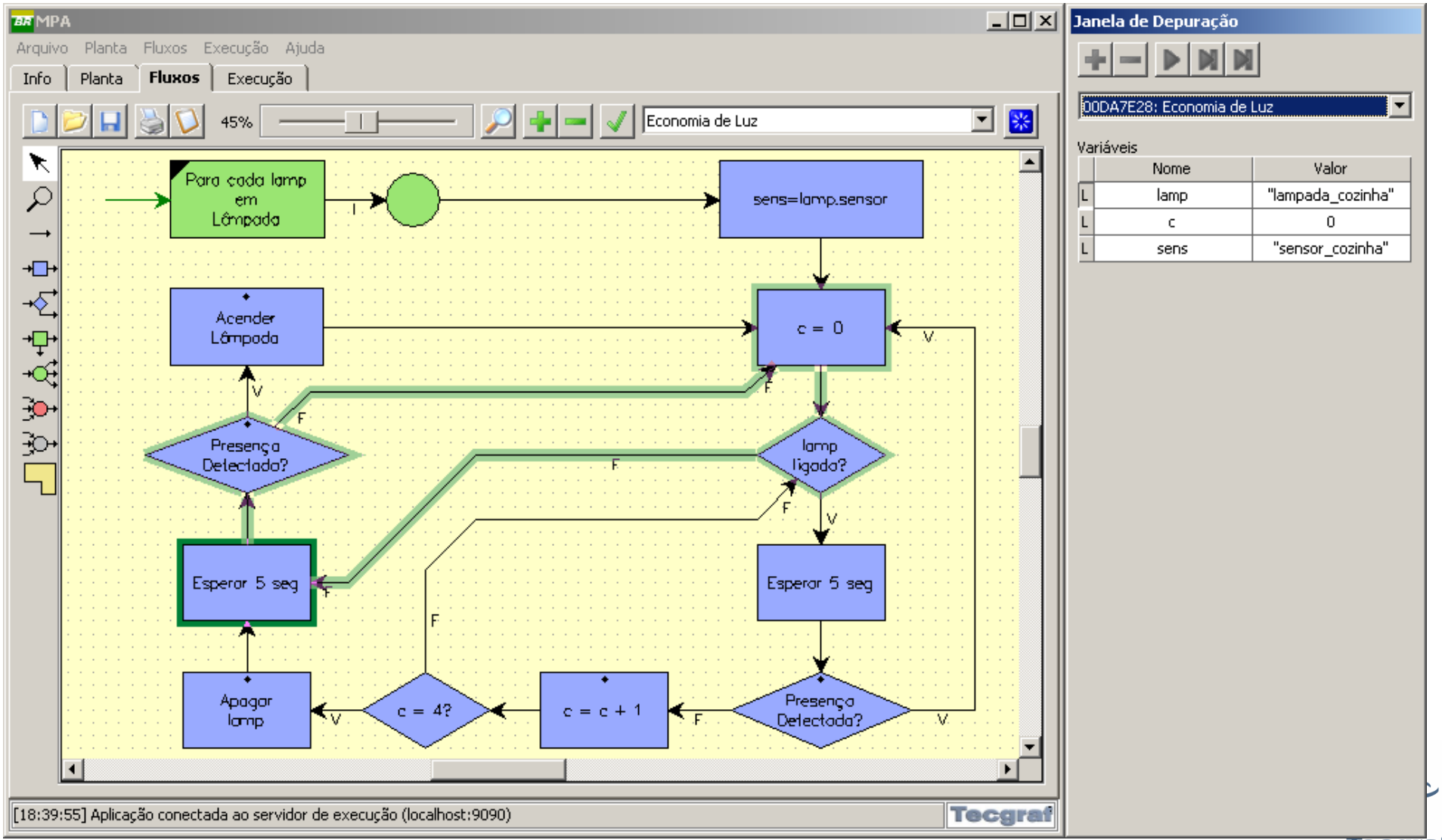
Visão Geral


Pré Configuração
Configuração
Execução
Supervisão
Arquivo de Pré-Configuração (Lua)
Aplicação Cliente do MPA
CORBA
Servidor de Execução do MPA
OPC / VXL
Supervisório
PLC

Configuração da Planta



Configuração de Fluxos



Motivação para uso de Lua

Portabilidade

- As plataformas em operação usam versões antigas de VMS.
- Algumas refinarias utilizam sistemas Microsoft Windows.

Extensibilidade

- O desenvolvedor deve ser capaz de incorporar novas funcionalidades de acordo com as diferentes necessidades dos ambientes de automação.

Manipulação de Dados

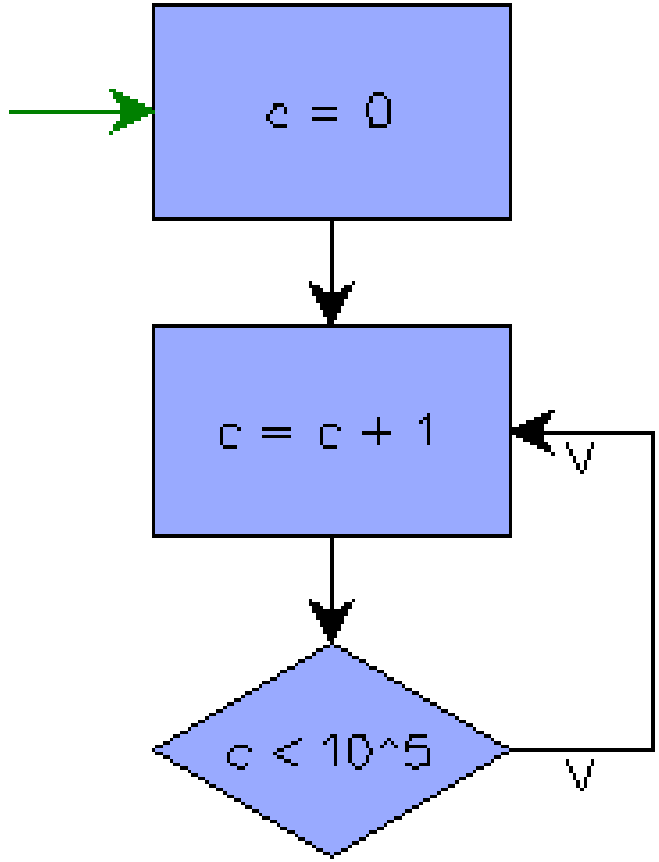
- O MPA deve manipular diferentes descrições do processo.
  - Equipamentos que compõem o ambiente de automação.
  - Procedimentos de automação.

Agenda

Tecgraf PUC-RIO

Recursão de Cauda e Máquinas de Estados

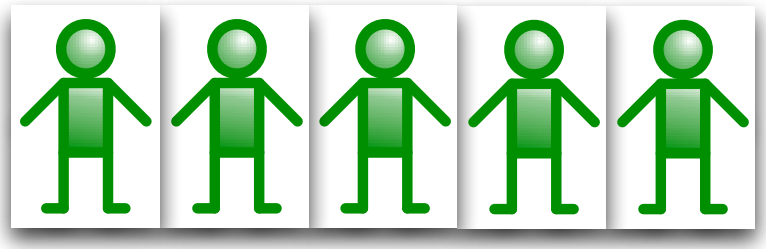
```
function Command:run()
  self.action()
  if self.next then
    return self.next:run()
  end
end
function Choice:run()
  if self.action() then
    if self.yes then
      return self.yes:run()
    end
  else
    if self.no then
      return self.no:run()
    end
  end
end
```



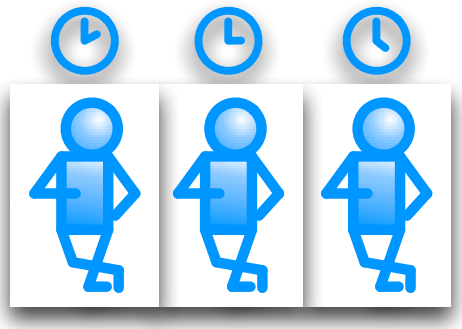

```
Init = Command{ action = function() c = 0 end }
Incr = Command{ action = function() c = c + 1 end }
Test = Choice{ action = function() return c < 10^5 end }
Init.next = Incr; Incr.next = Test; Test.yes = Incr
Init:run()
```

Escalonador de Co-Rotinas

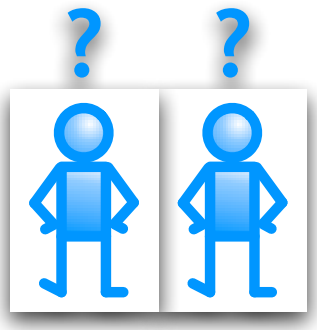
- Cada co-rotina representa uma *thread*.
- O escalonador é responsável por restaurar a execução de cada co-rotina.



Pronto para execução



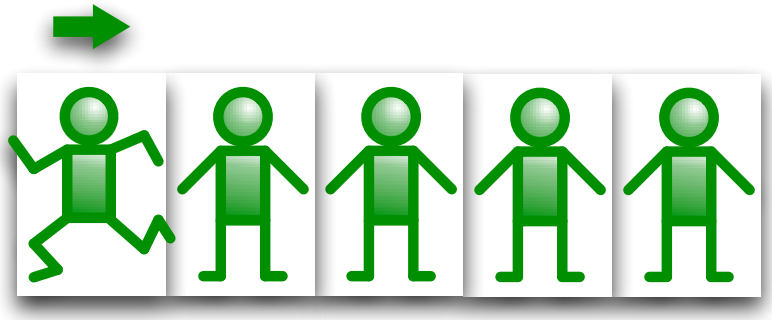
Espera por tempo



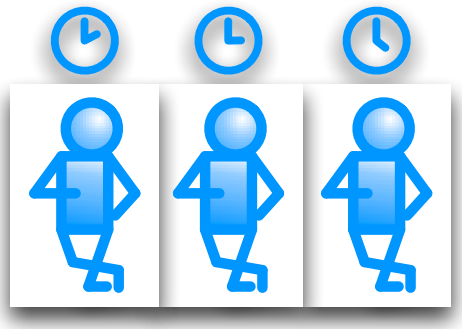
Espera por outros eventos (*e.g.* E/S)

Escalonador de Co-Rotinas

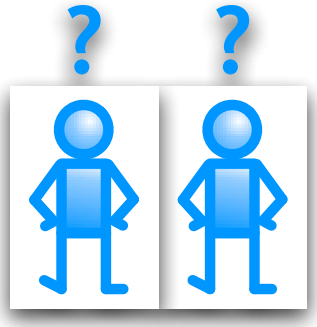
Cada co-rotina na fila de prontas tem sua execução restaurada uma vez, em sequência até o final da fila.



Pronto para execução



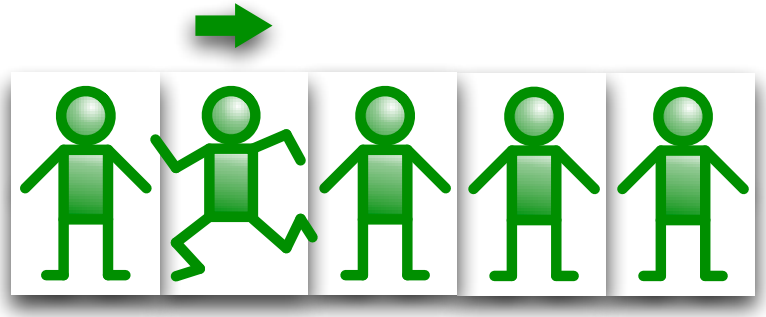
Espera por tempo



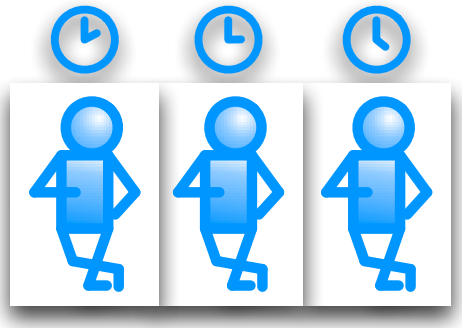
Espera por outros eventos (*e.g.* E/S)

Escalonador de Co-Rotinas

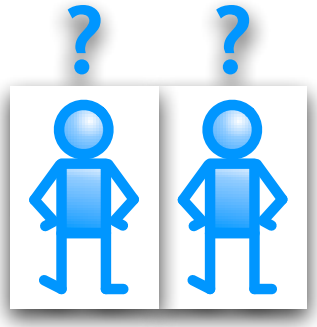
Durante sua execução, a co-rotina pode registrar outras co-rotinas para execução.



Pronto para execução



Espera por tempo



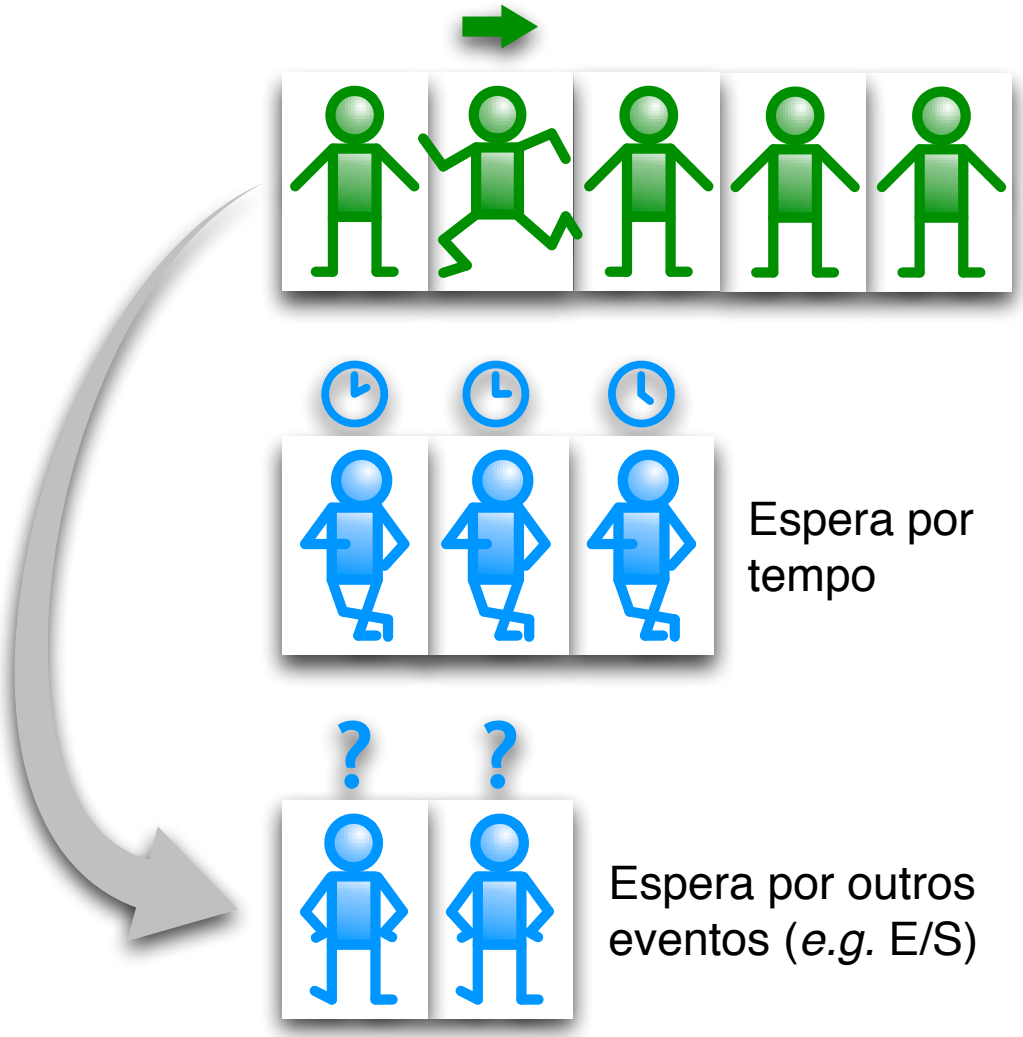
Espera por outros eventos (*e.g.* E/S)

Escalonador de Co-Rotinas

As co-rotinas podem suspender sua execução por um dado tempo.

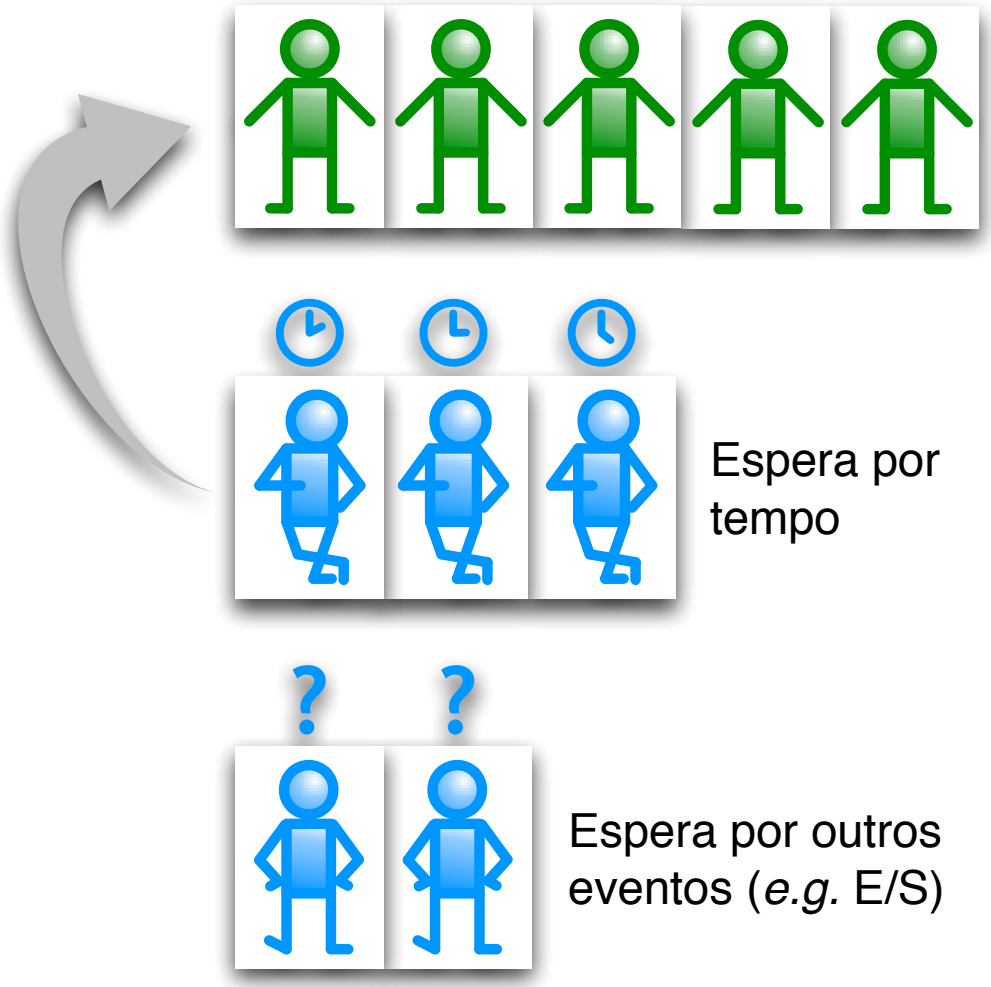
Escalonador de Co-Rotinas

Ou ainda esperar por um evento como a disponibilidade de um dado solicitado.



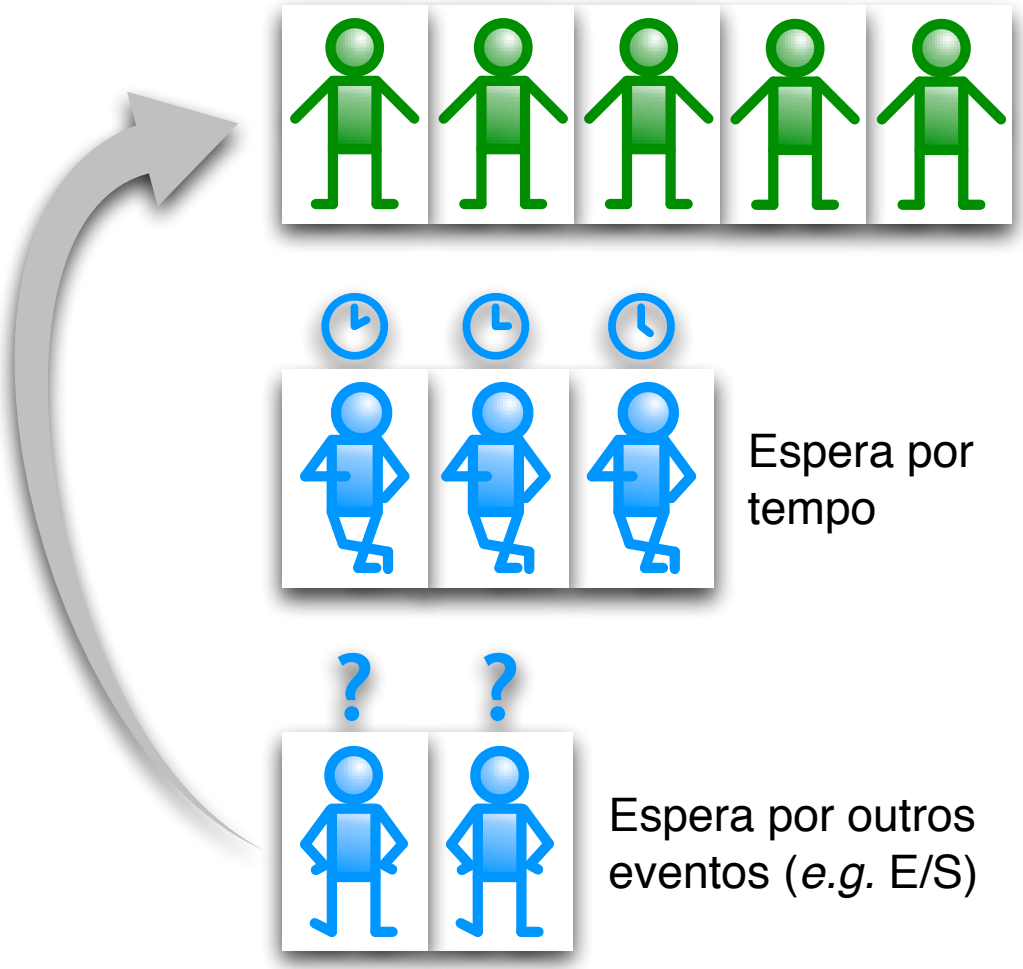
Escalonador de Co-Rotinas

Após a execução das co-rotinas prontas, o escalonador move para o início da fila de pronta todas as co-rotinas cuja espera por tempo tenha estourado.



Escalonador de Co-Rotinas

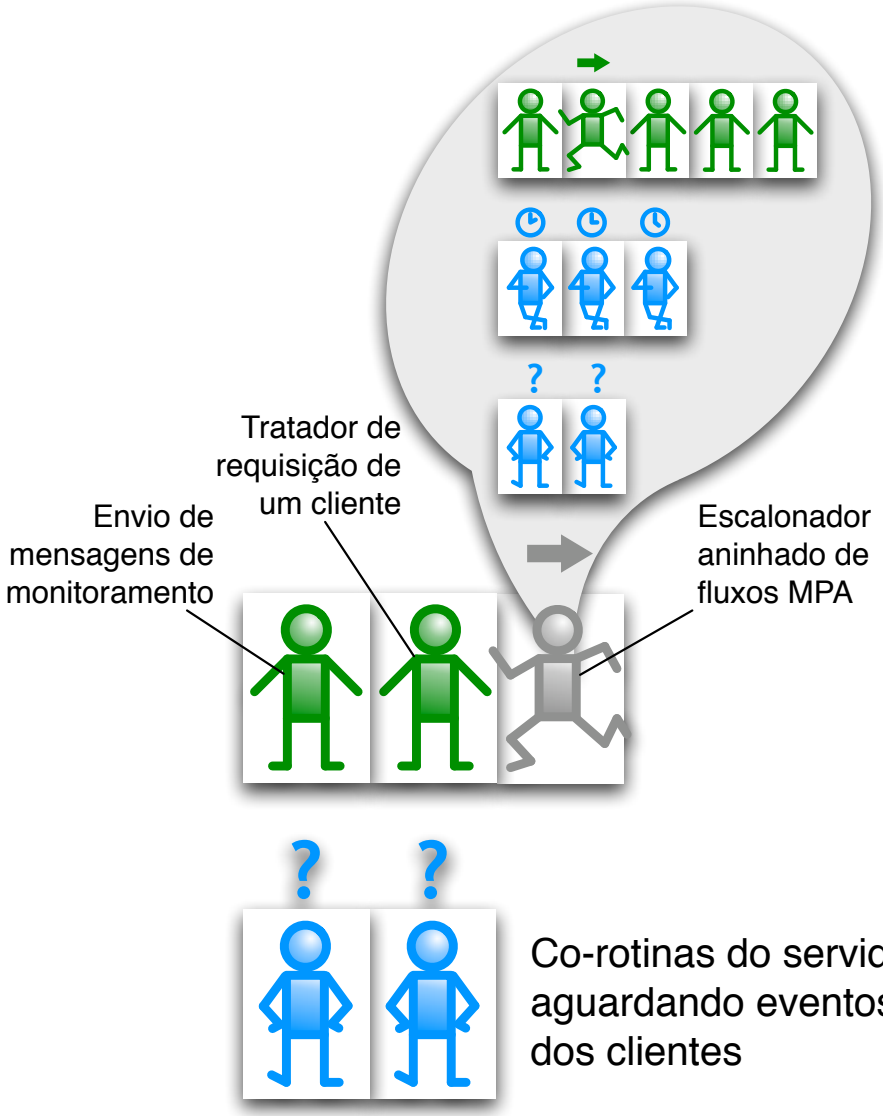
Antes de iniciar mais um ciclo de execução, o escalonador verifica se algum evento ocorreu e move as co-rotinas em espera para o início da fila de prontas.



Tecgraf PUC-RIO

Escalonadores Aninhados

- Permite priorizar a execução das co-rotinas dos fluxos.
- Facilita a suspensão/restauração da execução dos fluxos para implementar pontos de quebra (*break points*).



Agenda

Tecgraf PUC-RIO

Implementação da Interface de Desenvolvimento

- Antes: lógica e GUI em Lua, e editor de fluxo em C++.
- Agora: 100% Lua:
  - Evitar complexidades do alto acoplamento de partes em Lua e C++.
  - Usar reflexão computacional de Lua na implementação do Vix.
  - Usar coleta de lixo de Lua para minimizar vazamento de memória.

O Modelo Vix e Chamadas Dinâmicas

```
// Processamento de uma mensagem
struct MoveCommitMsg : public VixMessage {
  VixCollection * objs;
  int x, y;
  static const char* const MsgId() {
    return "MoveCommitMsg";
  }
  const char* const GetMessageId() {
    return MoveCommitMsg::MsgId();
  }
  MoveCommitMsg(VixCollection *oc, int dx, int dy) {
    objs = oc; x = dx; y = dy;
  }
};
short int Box::MsgHandlerVO(VixMessage *msg) {
  if (msg->GetId() == MoveCommitMsg::MsgId())
    return MoveCommit((MoveCommitMsg*)msg);
  else if (msg->GetId() == MoveFeedbackMsg::MsgId())
    ...
  return MSG_NOT_PROCESSED;
}
short int Box::MoveCommit(MoveCommitMsg *m) {
  ...
  return MSG_PROCESSED;
}

// Envio de uma mensagem
MoveCommitMsg move(objects, dx, dy);
move.SendVO(dest);
```

```
-- Processamento de uma mensagem
function Box:MoveCommit(objs,dx,dy)
  ...
  return true
end

-- Envio de uma mensagem
dest:inMoveCommit(objs, dx, dy)
```

O Modelo Vix e Chamadas Dinâmicas

```
// Processamento de uma mensagem
struct MoveCommitMsg : public VixMessage {
  VixCollection * objs;
  int x, y;
  static const char* const MsgId() {
    return "MoveCommitMsg";
  }
  const char* const GetMessageId() {
    return MoveCommitMsg::MsgId();
  }
  MoveCommitMsg(VixCollection *oc, int dx, int dy) {
    objs = oc; x = dx; y = dy;
  }
};
short int Box::MsgHandlerVO(VixMessage *msg) {
  if (msg->GetId() == MoveCommitMsg::MsgId())
    return MoveCommit((MoveCommitMsg*)msg);
  else if (msg->GetId() == MoveFeedbackMsg::MsgId())
    ...
  return MSG_NOT_PROCESSED;
}
short int Box::MoveCommit(MoveCommitMsg *m) {
  ...
  return MSG_PROCESSED;
}

// Envio de uma mensagem
MoveCommitMsg move(objects, dx, dy);
move.SendVO(dest);
```

```
-- Processamento de uma mensagem
function Box:MoveCommit(objs,dx,dy)
  ...
  return true
end

-- Envio de uma mensagem
dest:inMoveCommit(objs, dx, dy)
```

Persistência dos Projetos

— *Dados*

```
Command = {
  xmin = 5242,
  ymin = 5254,
  xmax = 5989,
  ymax = 5326,
  action = {
    _switch = "FUNCTIONCALL",
    expressions = { "a", "b+c", "123" },
    variable = { "res" },
    functionid = "calcMean",
  },
  errorlist = {
    "unknown function, got 'calcMean'",
  }
}
```

— *Operações*

```
CommandMethods = {
  Repaint = function(self)
    ...
  end,
  Move = function(self, dx, dy)
    ...
  end,
  Reshape = function(self, point, dx, dy)
    ...
  end,
}
```

— *Campos Persistentes*

```
CommandFields = {
  {id = "auto_description", type = "BOOLEAN"},
  {id = "xmin", type = "REAL"},
  {id = "ymin", type = "REAL"},
  {id = "xmax", type = "REAL"},
  {id = "ymax", type = "REAL"},
  {id = "action", switch = "_switch",
    {id = "_switch", type = "STRING"},
    {id = "expressions", type = "STRING", list
    {id = "variables", type = "STRING", list =
    FUNCTIONCALL = {
      {id = "functionid", type = "STRING"},
    },
    FLOWCALL = {
      {id = "flowid", type = "STRING"},
      {id = "operation", type = "STRING"},
    },
    OBJECTCALL = {
      {id = "classid", type = "STRING"},
      {id = "objname", type = "STRING"},
      {id = "member", type = "STRING"},
      {id = "operation", type = "STRING"},
    },
  },
}
```

Undo & Redo

```
function Movement:Complete(device)
  local x1,y1, x2,y2 = self:GetTwoPoints()
  local dx = x2-x1
  local dy = y2-y1
  dx = math.max(self.xmin, math.min(dx, self.xmax))
  dy = math.max(self.ymin, math.min(dy, self.ymax))

  self.tool:inMoveCommit(self.objects, device, dx, dy, self.box)
end


function Annotation:Move(device, dx, dy, vertex)
  if vertex == true then
    return Polyline.Move(self, device, dx, dy, vertex)
  else -- movimentação de uma aresta
    local x, y = self.x, self.y
    local next = vertex % #x + 1

    x[vertex] = x[vertex]+dx
    y[vertex] = y[vertex]+dy
    x[next]   = x[next]+dx
    y[next]   = y[next]+dy

    self:ResetVertexes(device)
  end
end
```

Undo & Redo

```
function Movement:Complete(device)
  local x1,y1, x2,y2 = self:GetTwoPoints()
  local dx = x2-x1
  local dy = y2-y1
  dx = math.max(self.xmin, math.min(dx, self.xmax))
  dy = math.max(self.ymin, math.min(dy, self.ymax))
  UserActions:begin("Movimentar seleção")
  self.tool:inMoveCommit(self.objects, device, dx, dy, self.box)
  UserActions:commit()
end

function Annotation:Move(device, dx, dy, vertex)
  if vertex == true then
    return Polyline.Move(self, device, dx, dy, vertex)
  else -- movimentação de uma aresta
    local x, y = self.x, self.y
    local next = vertex % #x + 1
    UserActions:doit{
      redo = function(self)
        x[vertex] = x[vertex]+dx
        y[vertex] = y[vertex]+dy
        x[next]   = x[next]+dx
        y[next]   = y[next]+dy
      end,
      undo = function(self)
        x[vertex] = x[vertex]-dx
        y[vertex] = y[vertex]-dy
        x[next]   = x[next]-dx
        y[next]   = y[next]-dy
      end,
    }
    self:ResetVertexes(device)
  end
```

Agenda

Tecgraf PUC-RIO

Onde Lua Ajudou?

Recursão de Cauda

- Implementação dos fluxos como máquinas de estado.

Co-Rotinas

- Implementação de *multithreading* cooperativo.

Reflexão Computacional

- Implementação do modelo Vix.
- Implementação da serialização de projetos.

Fechos de Função

- Implementação do recurso de *Undo&Redo*.

Tecgraf PUC-RIO

Fim

Obrigado!

Equipe MPA
mpa@tecgraf.puc-rio.br

Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```

Co-Rotinas e *Multithreading* Cooperativo

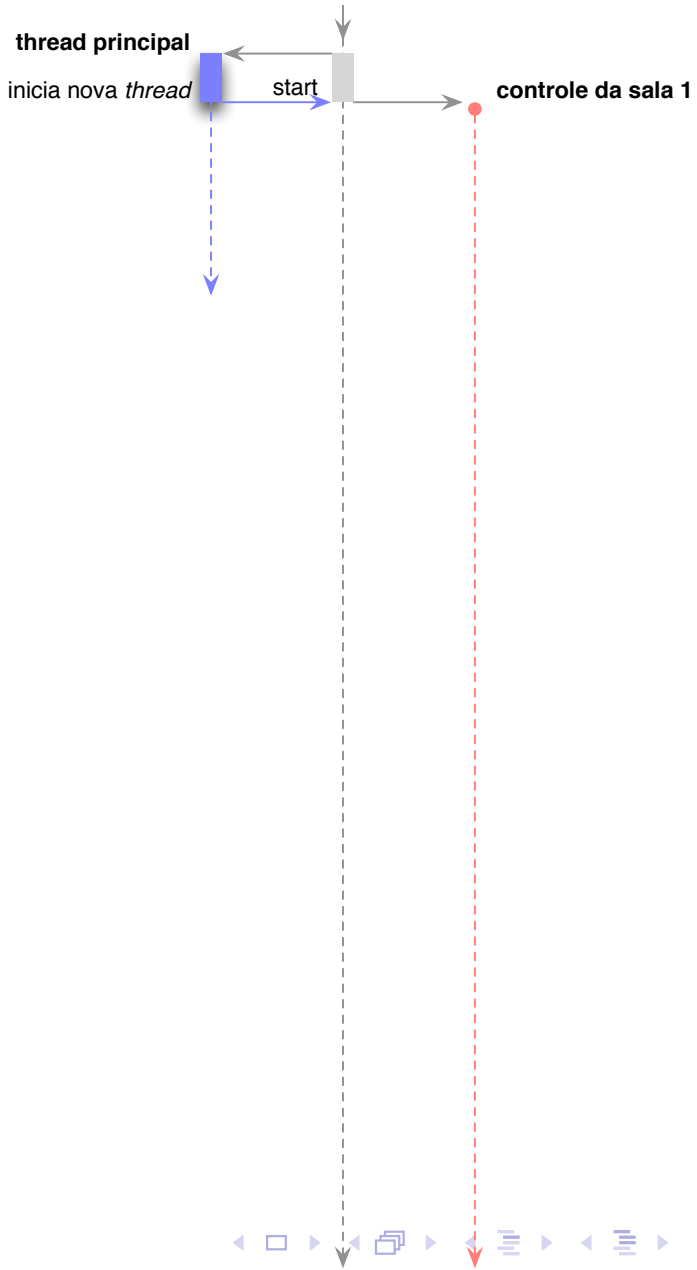
```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```

Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```

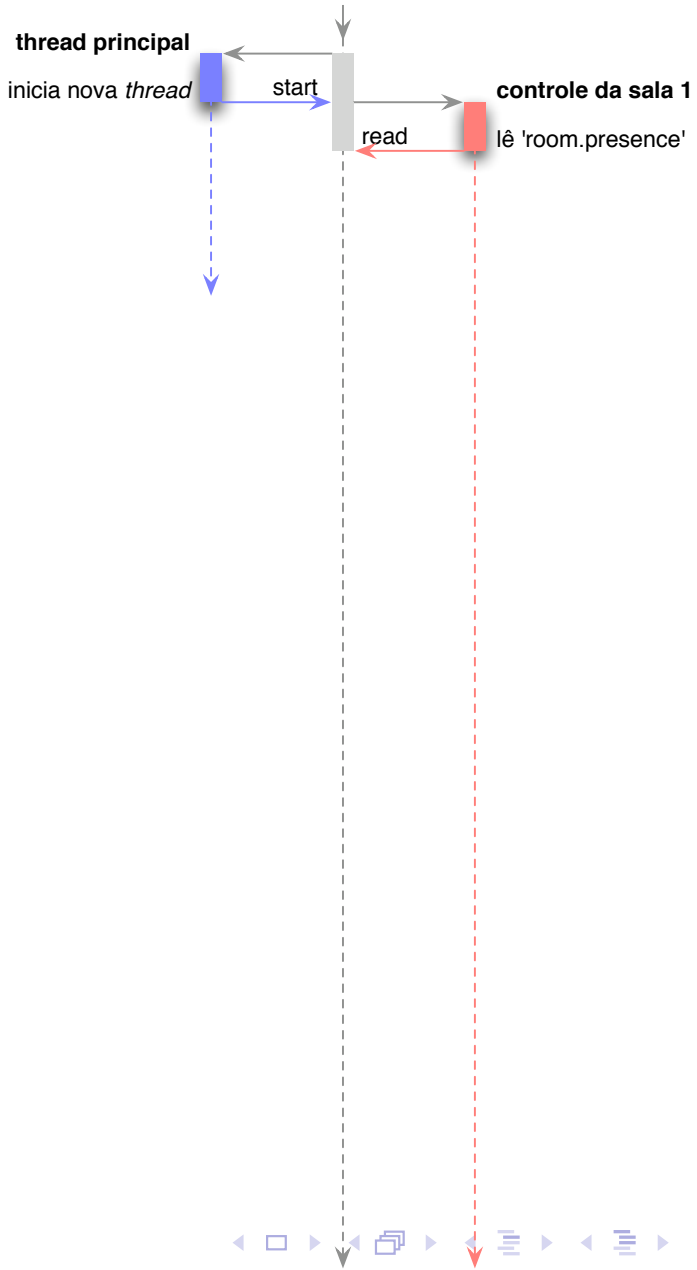
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



Co-Rotinas e *Multithreading* Cooperativo

```lua
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```

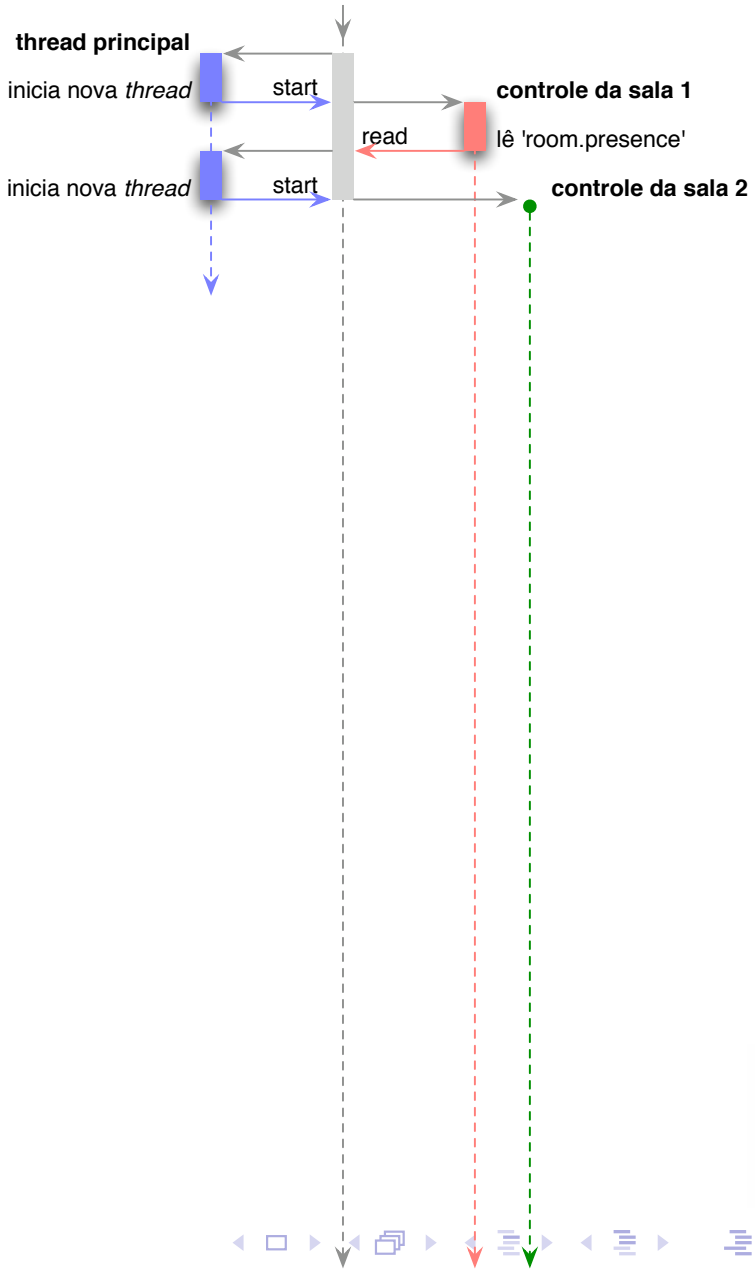


Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```

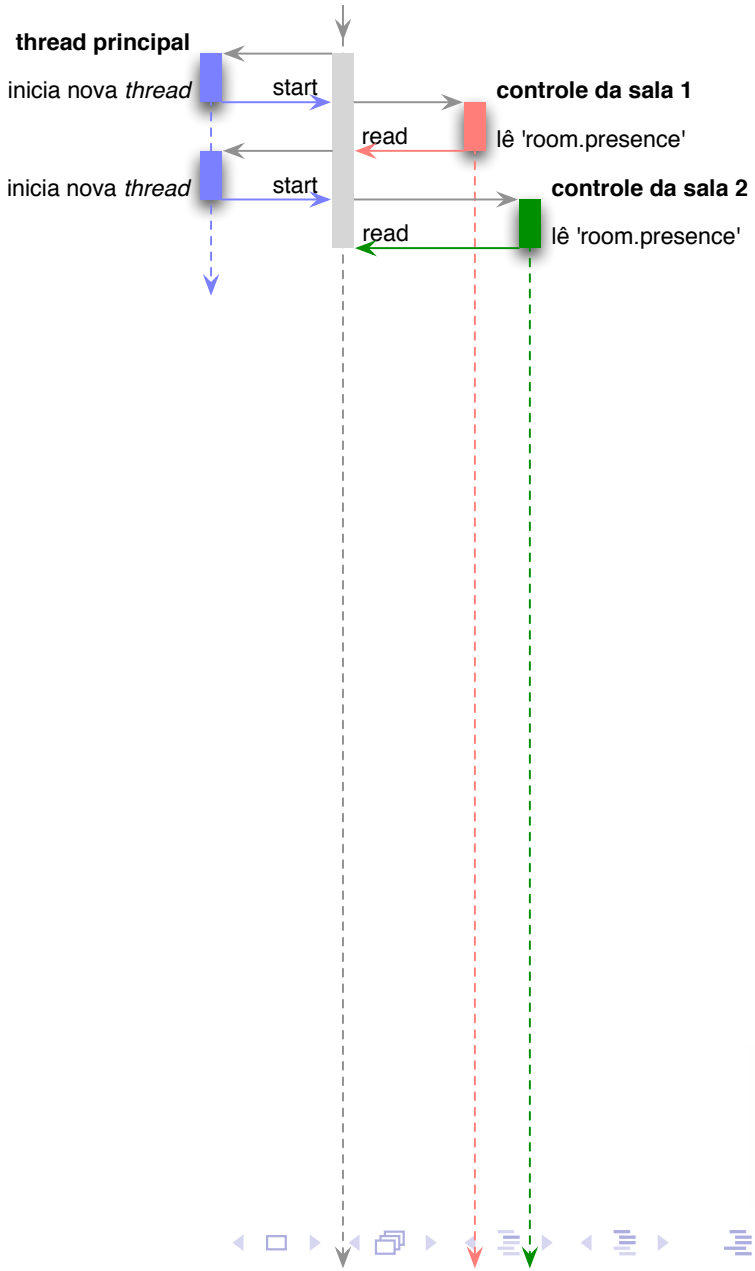
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



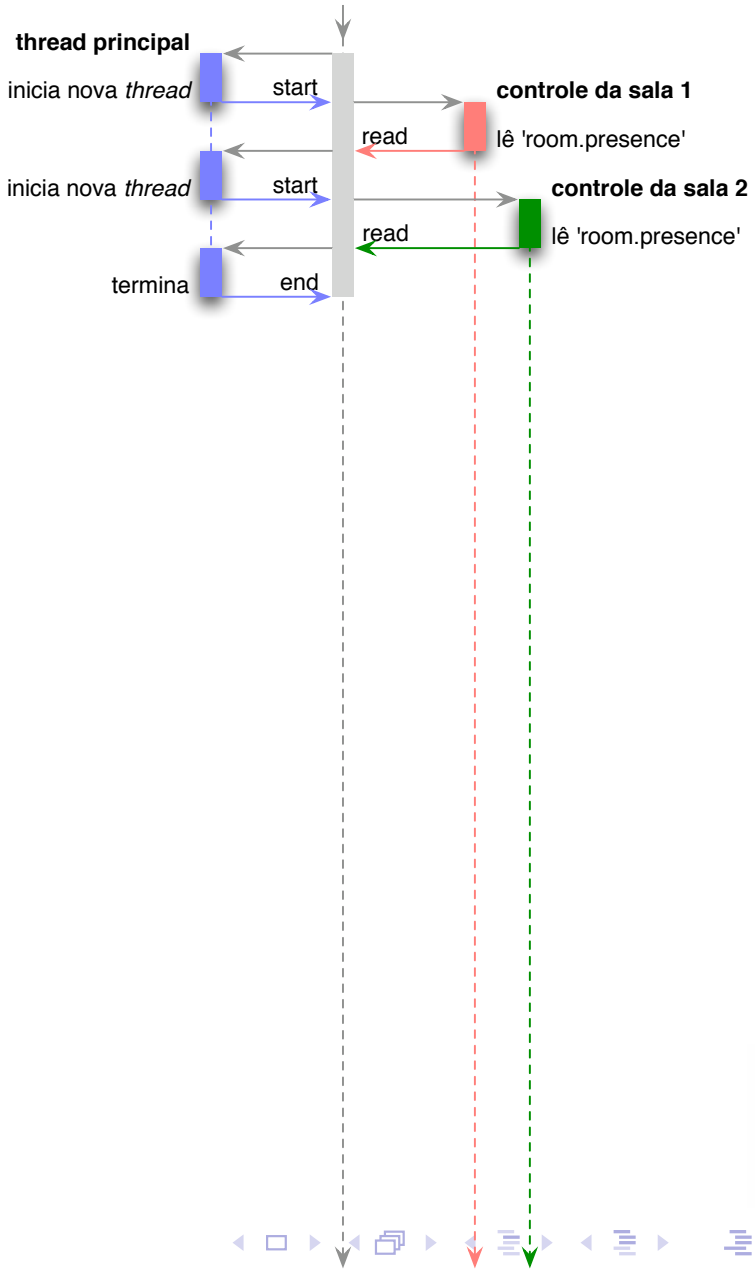
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



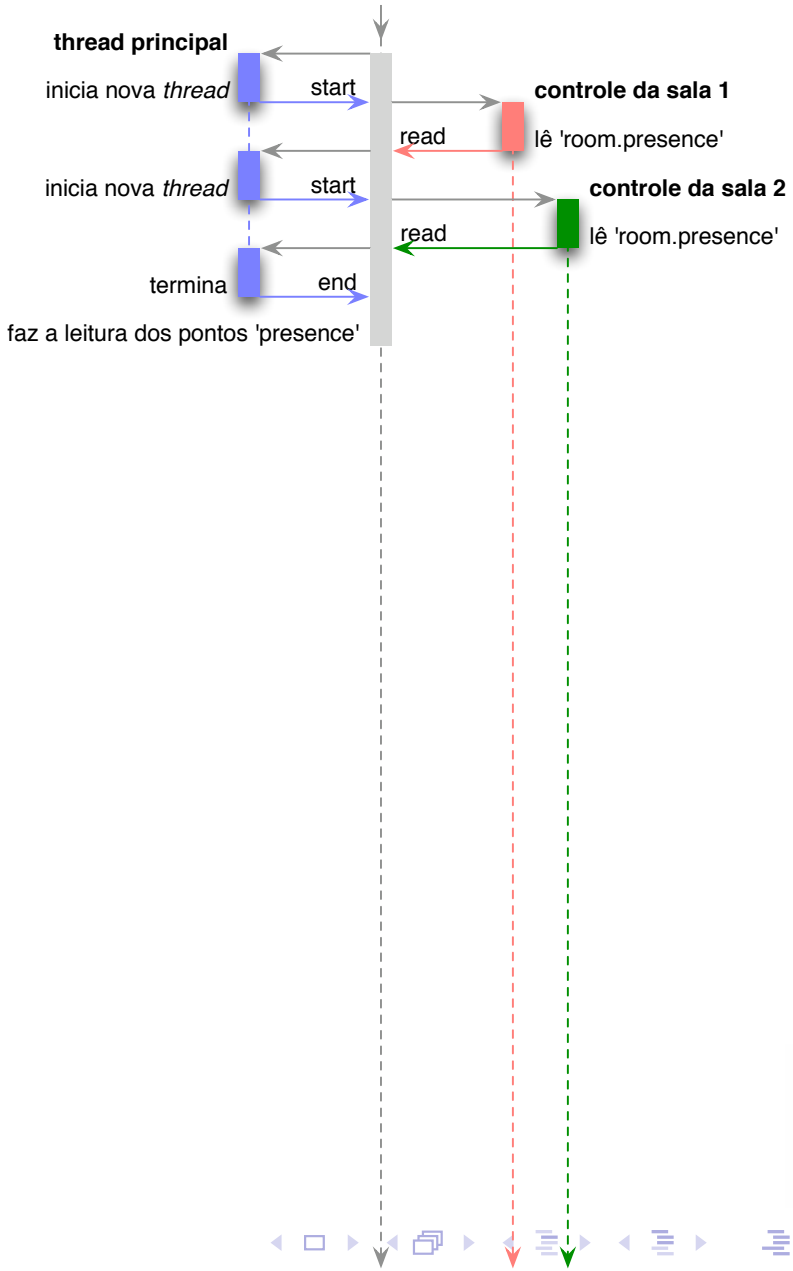
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



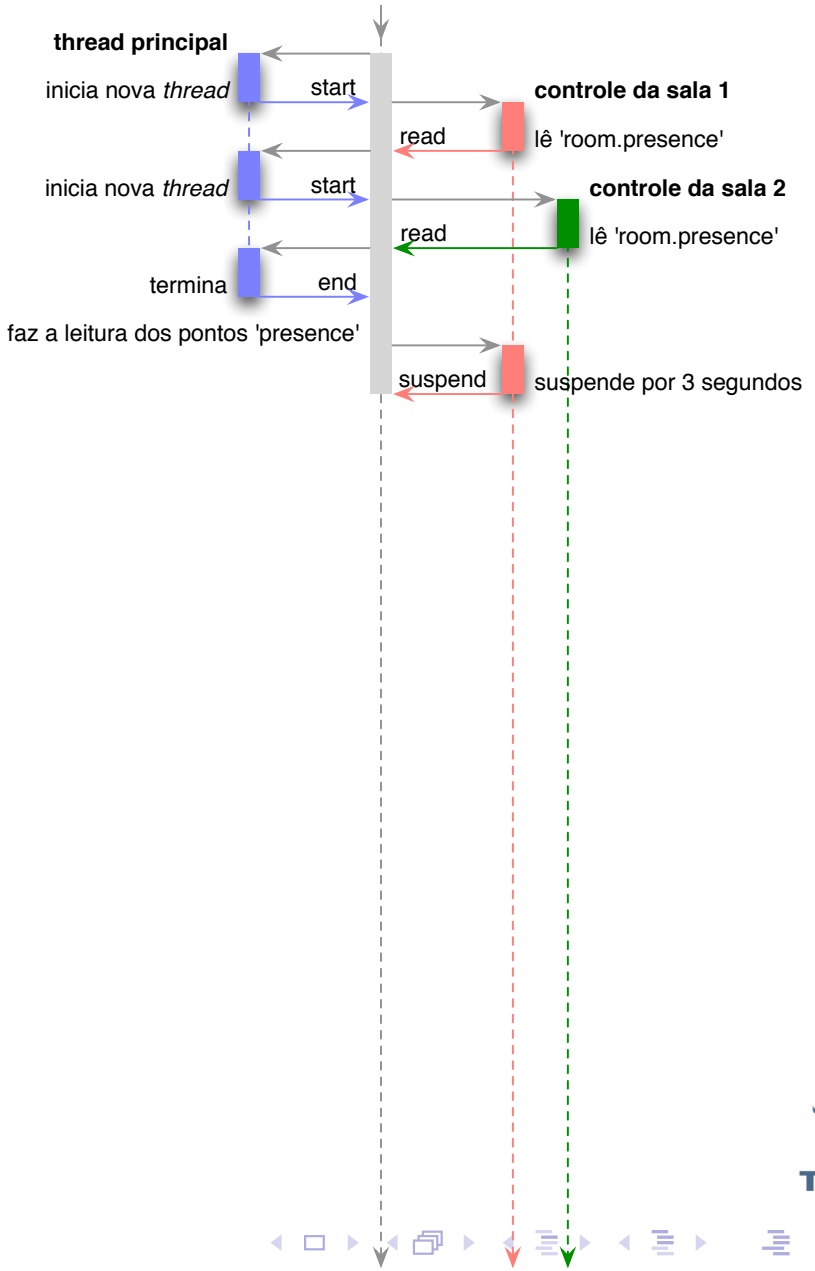
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



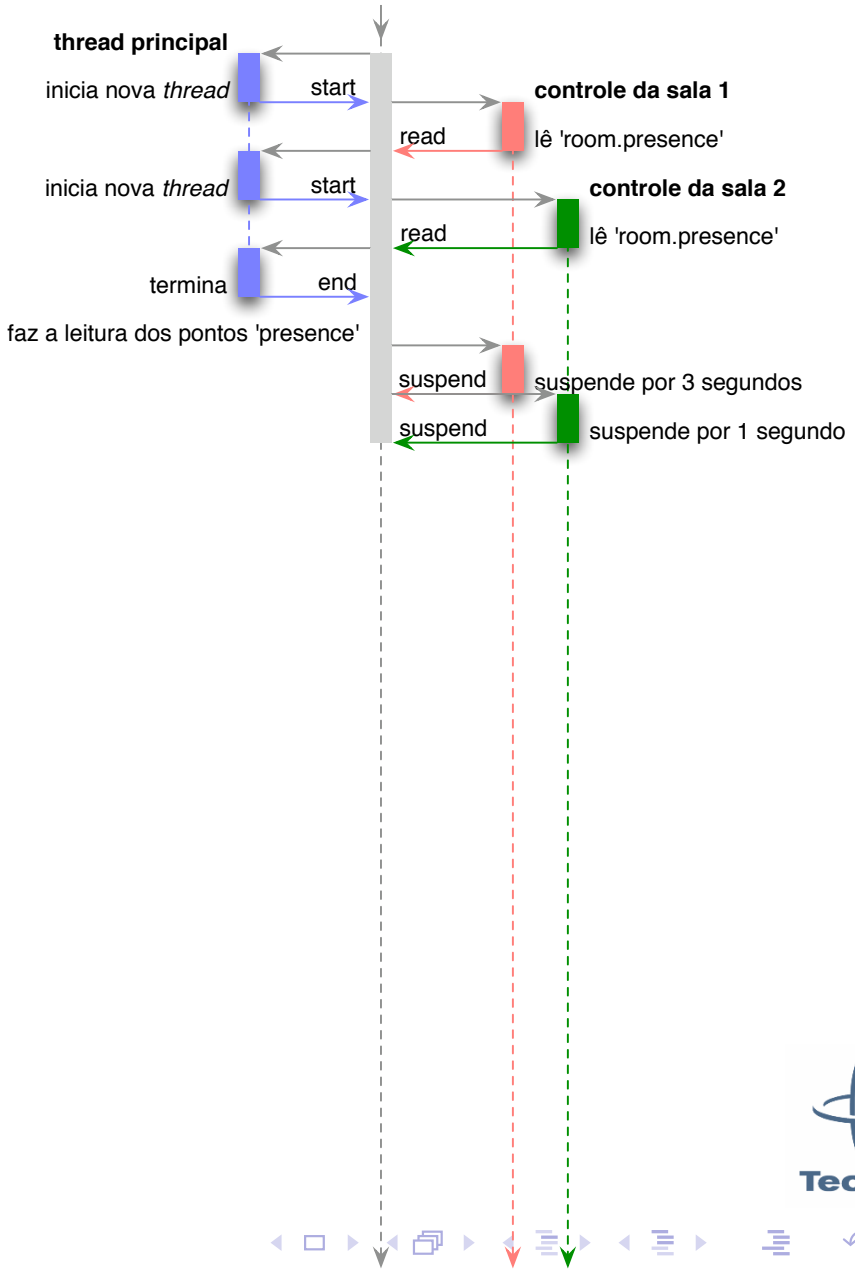
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



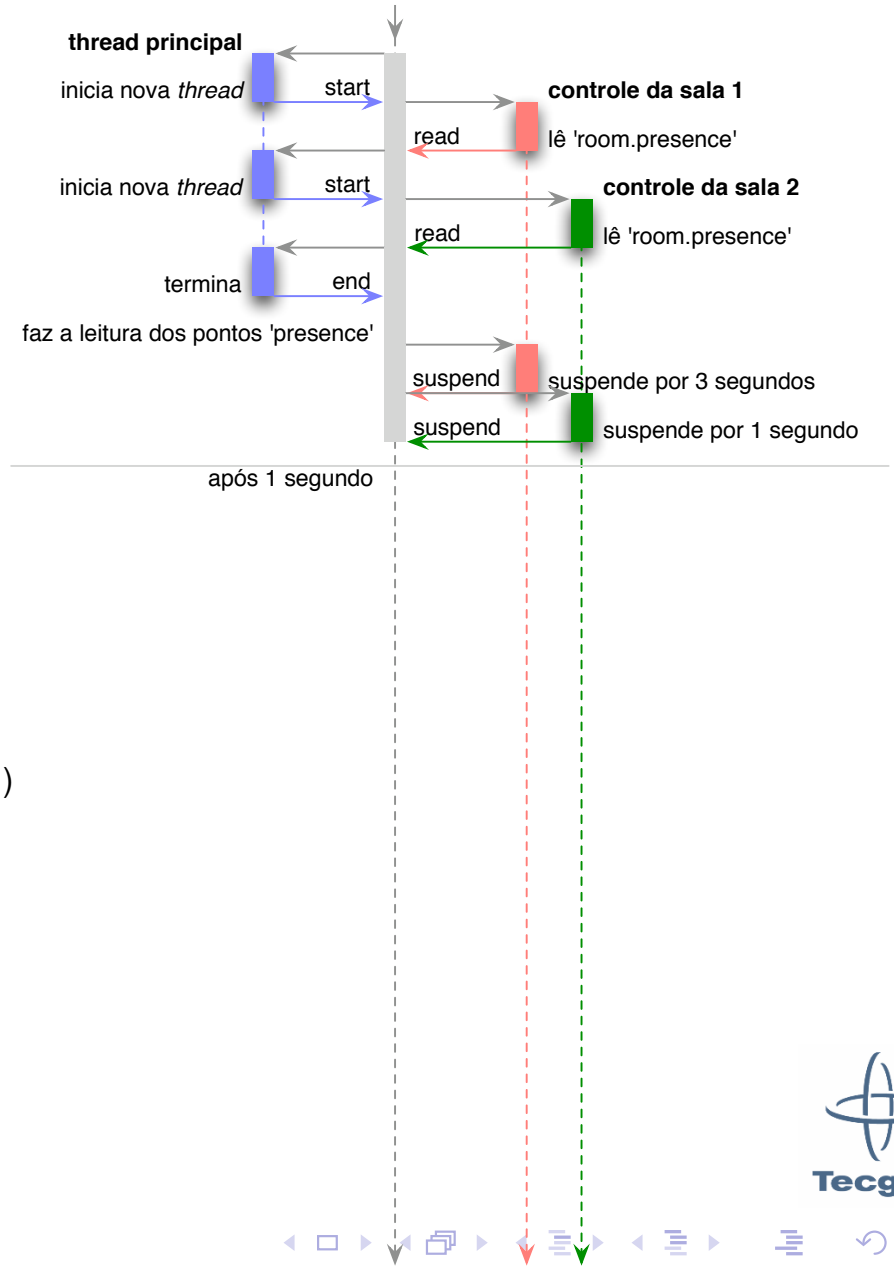
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



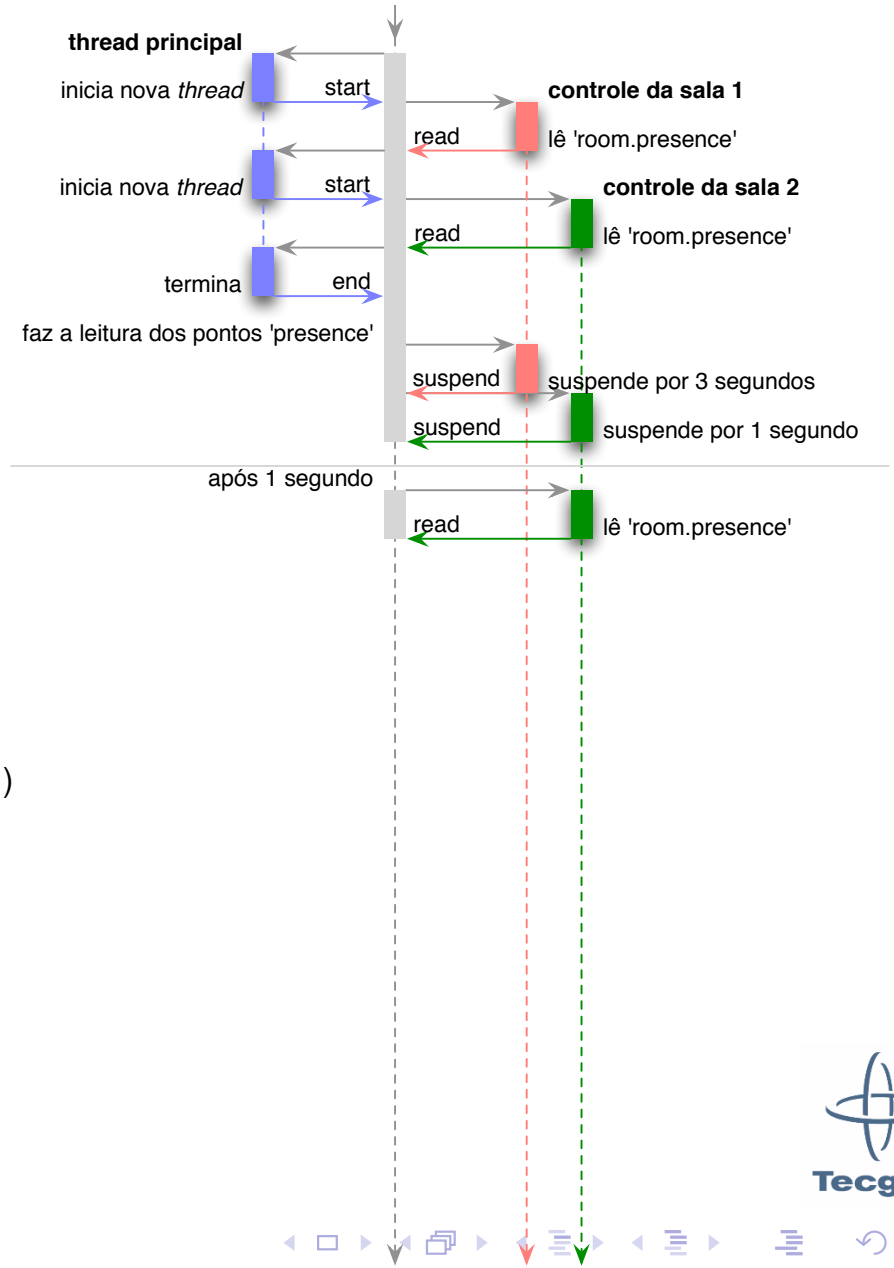
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



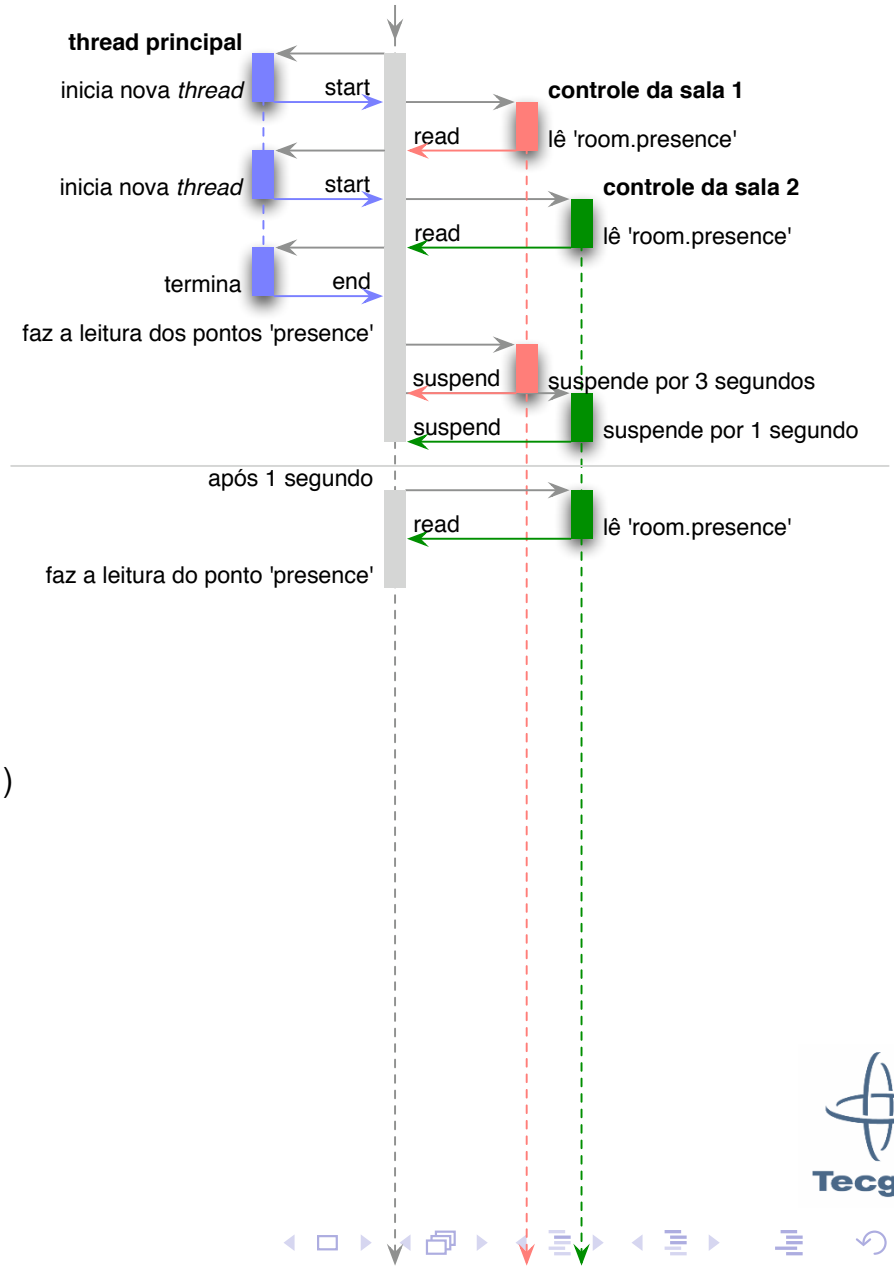
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



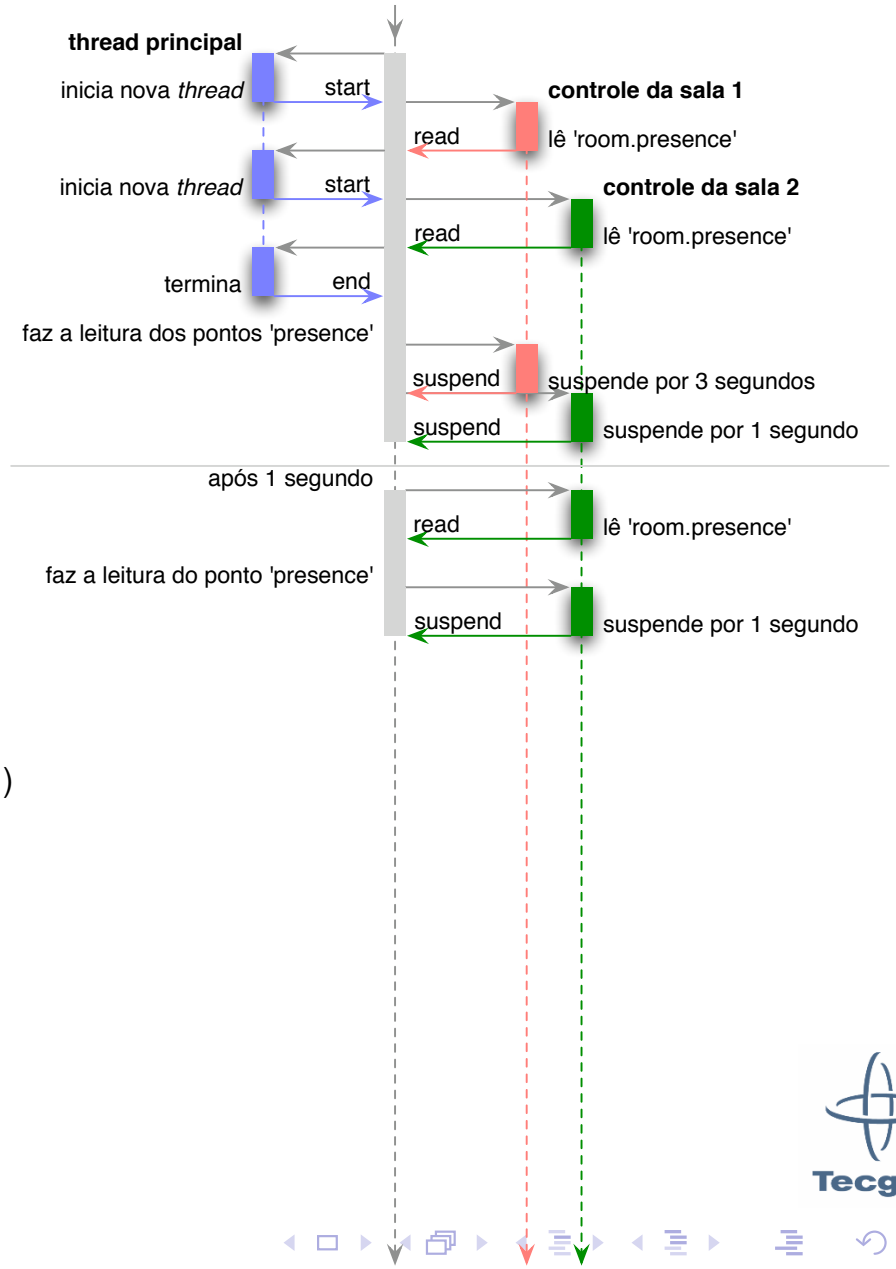
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



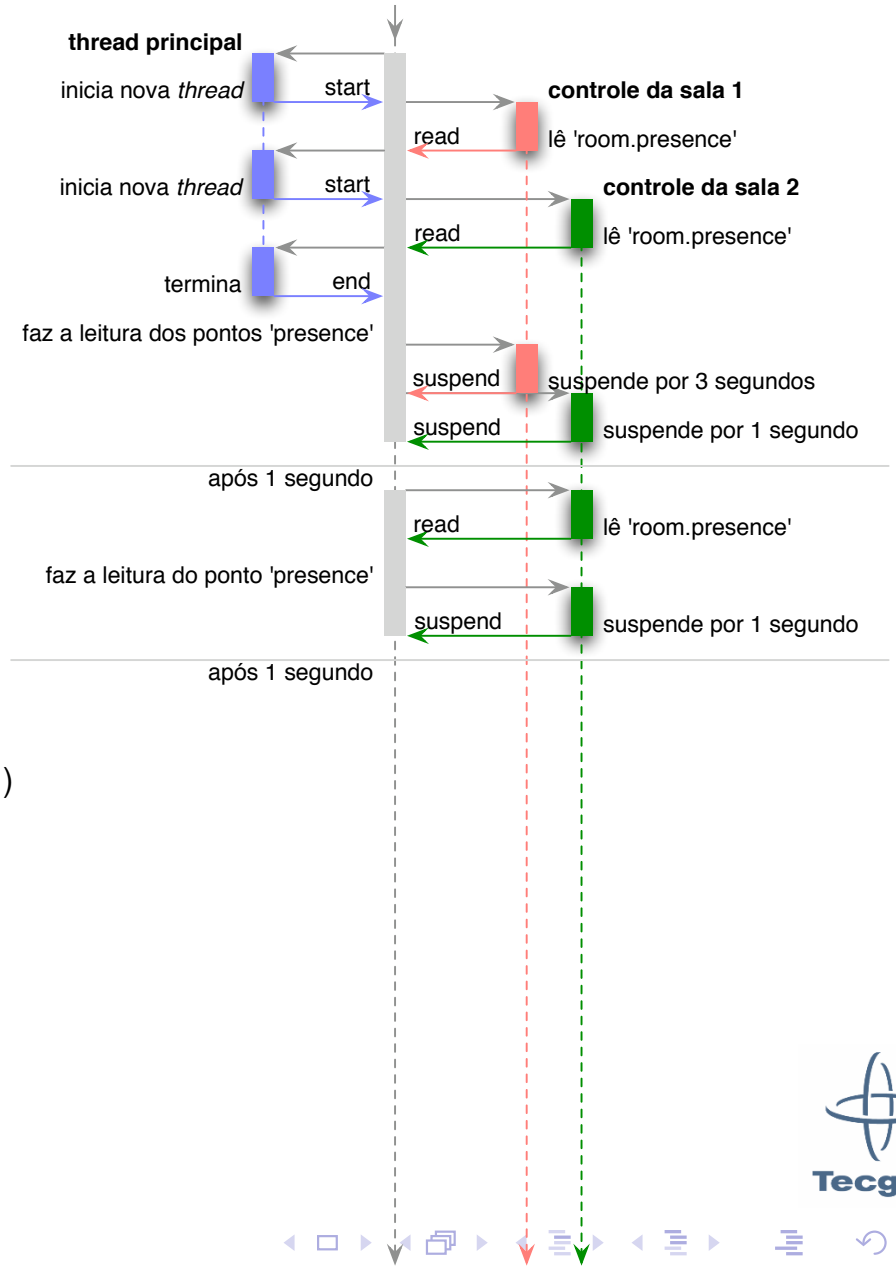
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



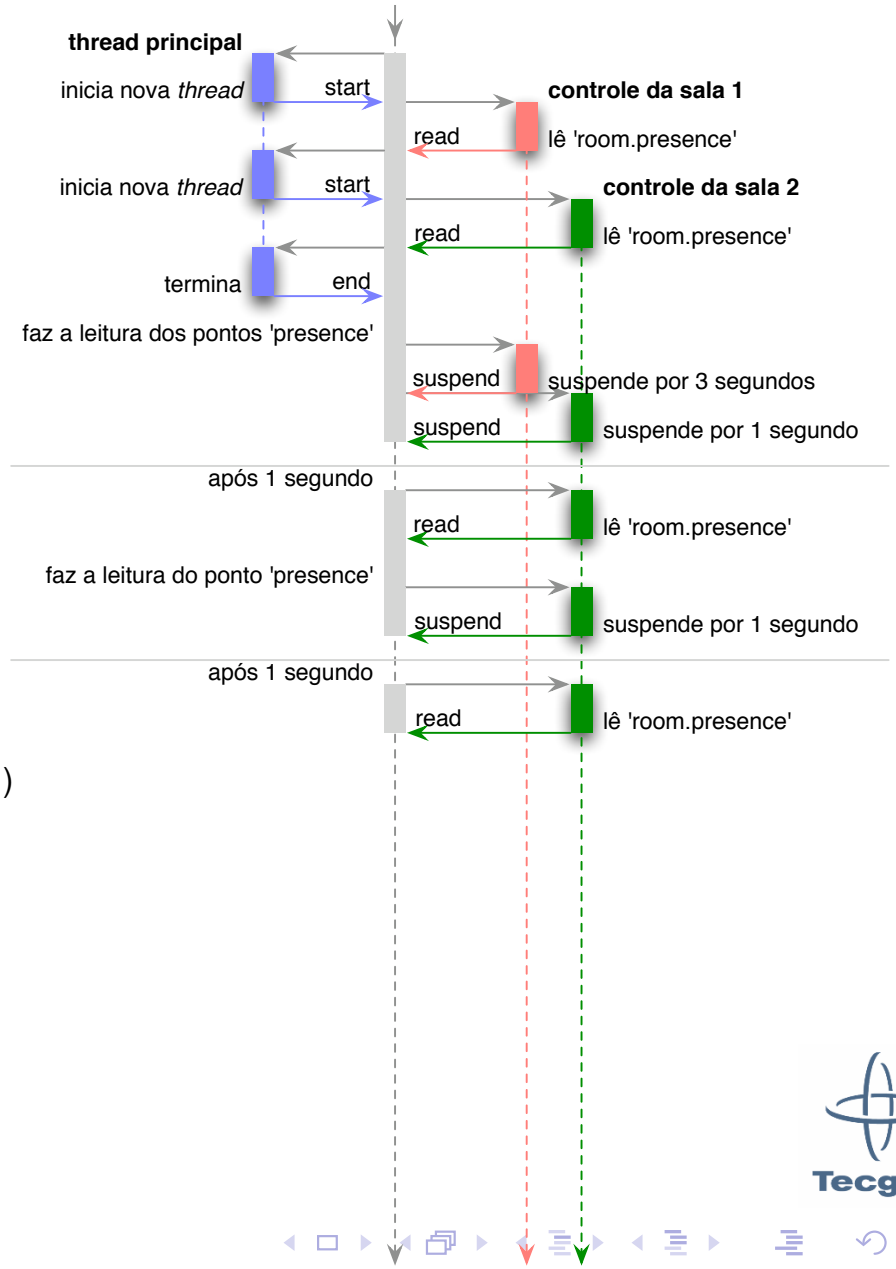
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



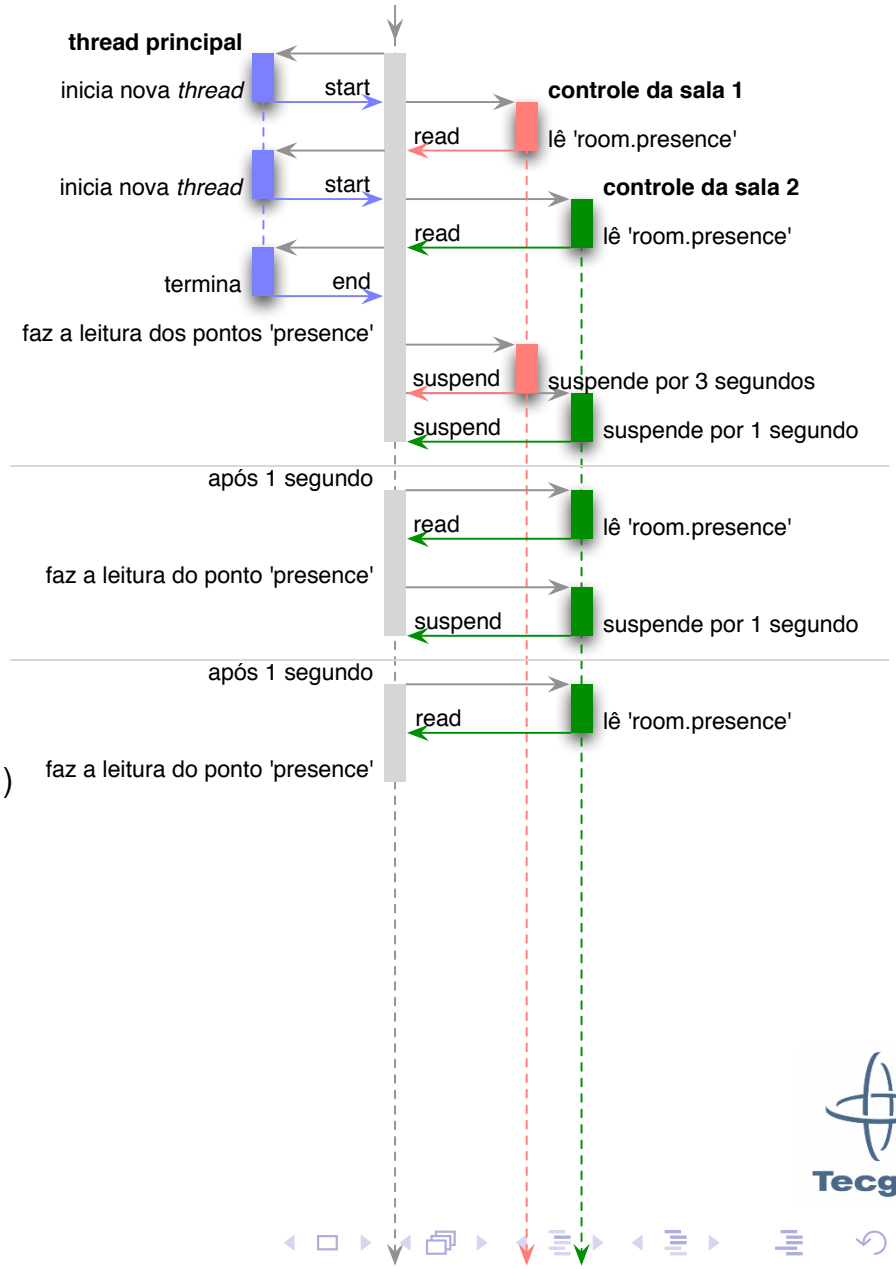
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



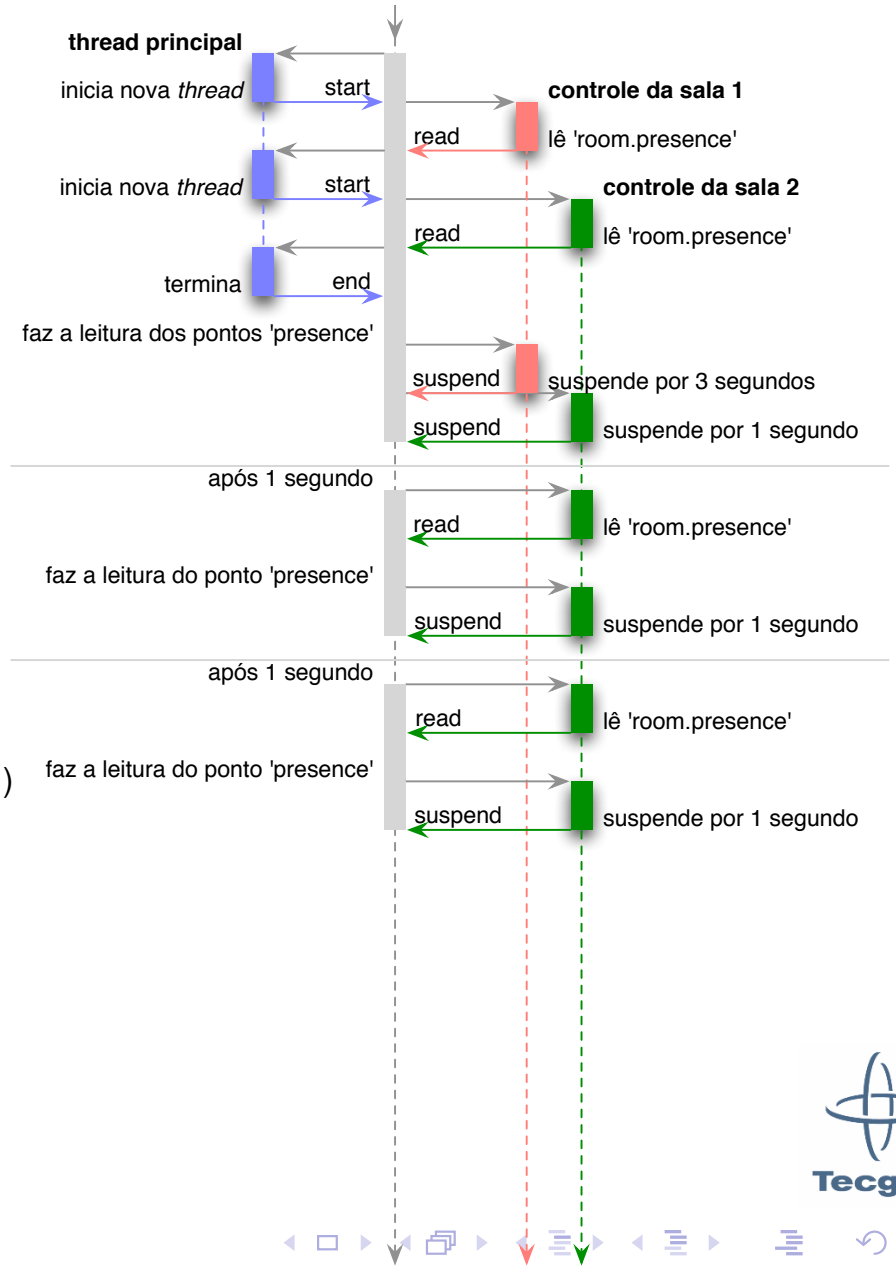
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



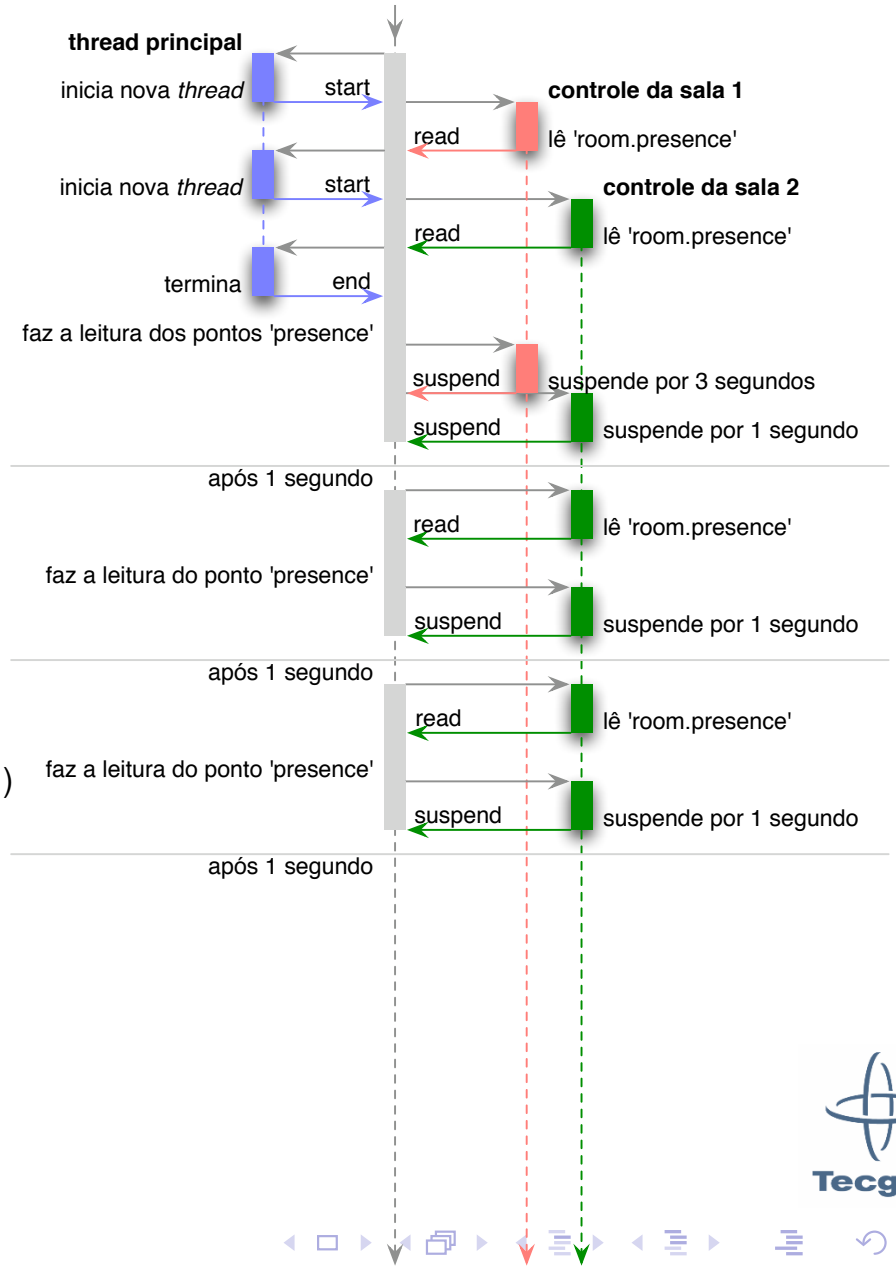
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



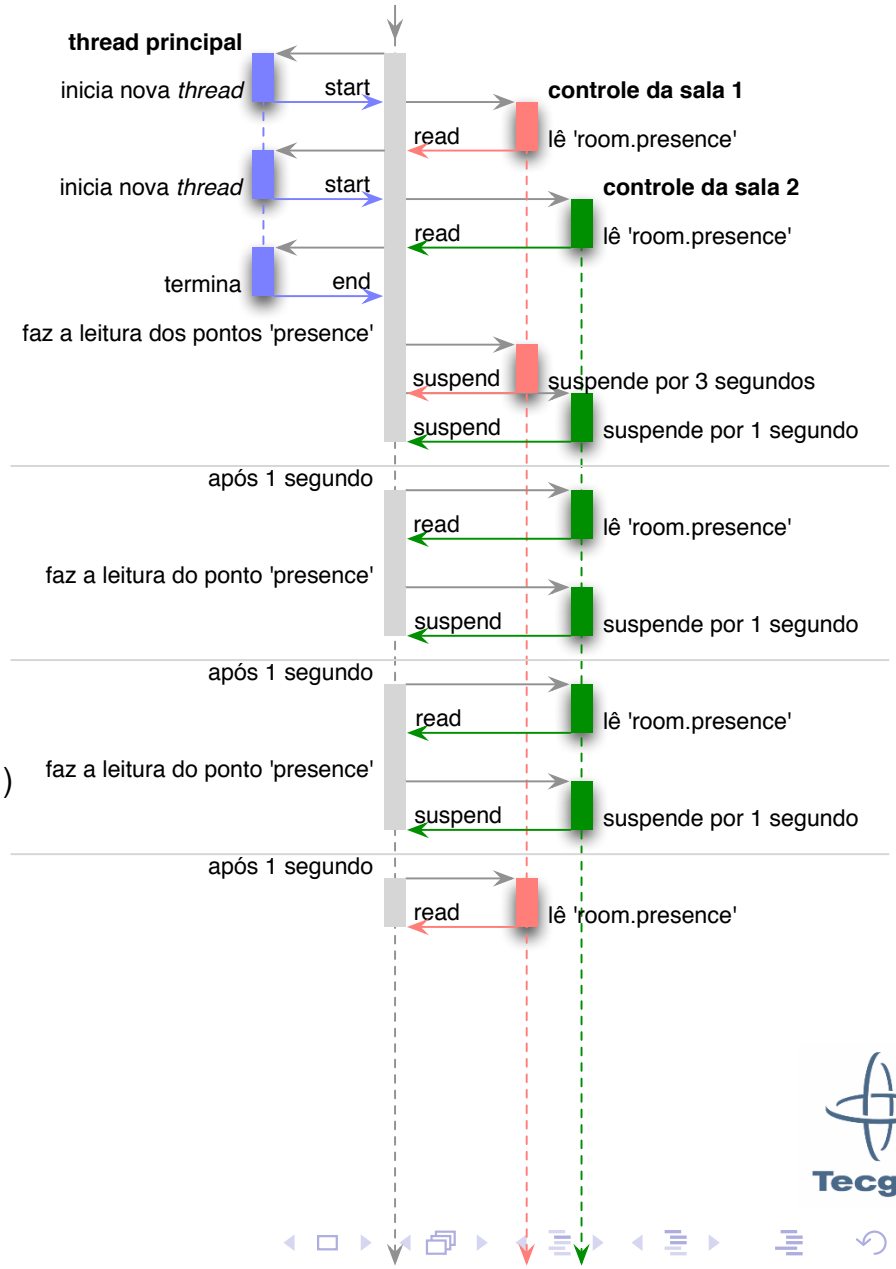
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



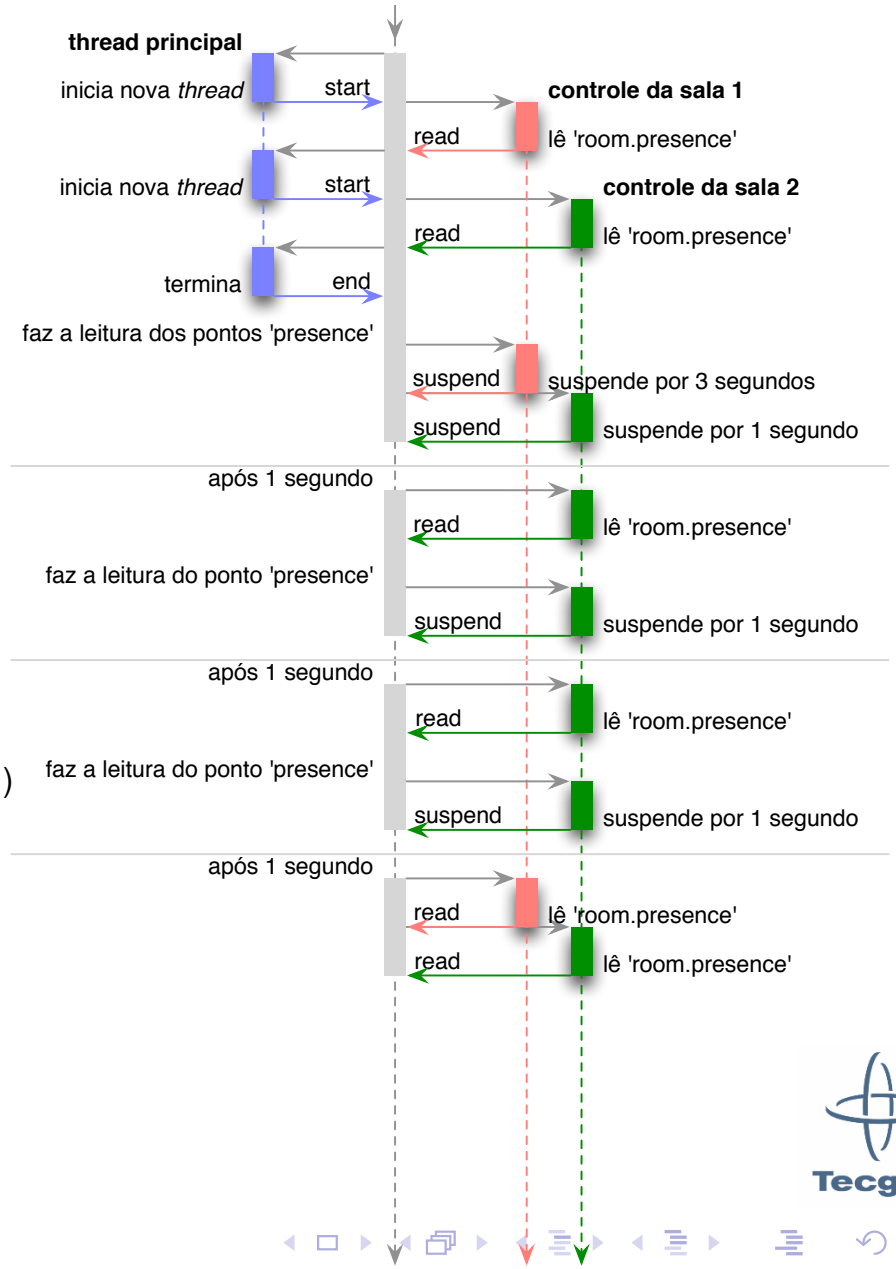
Co-Rotinas e *Multithreading* Cooperativo

```lua
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



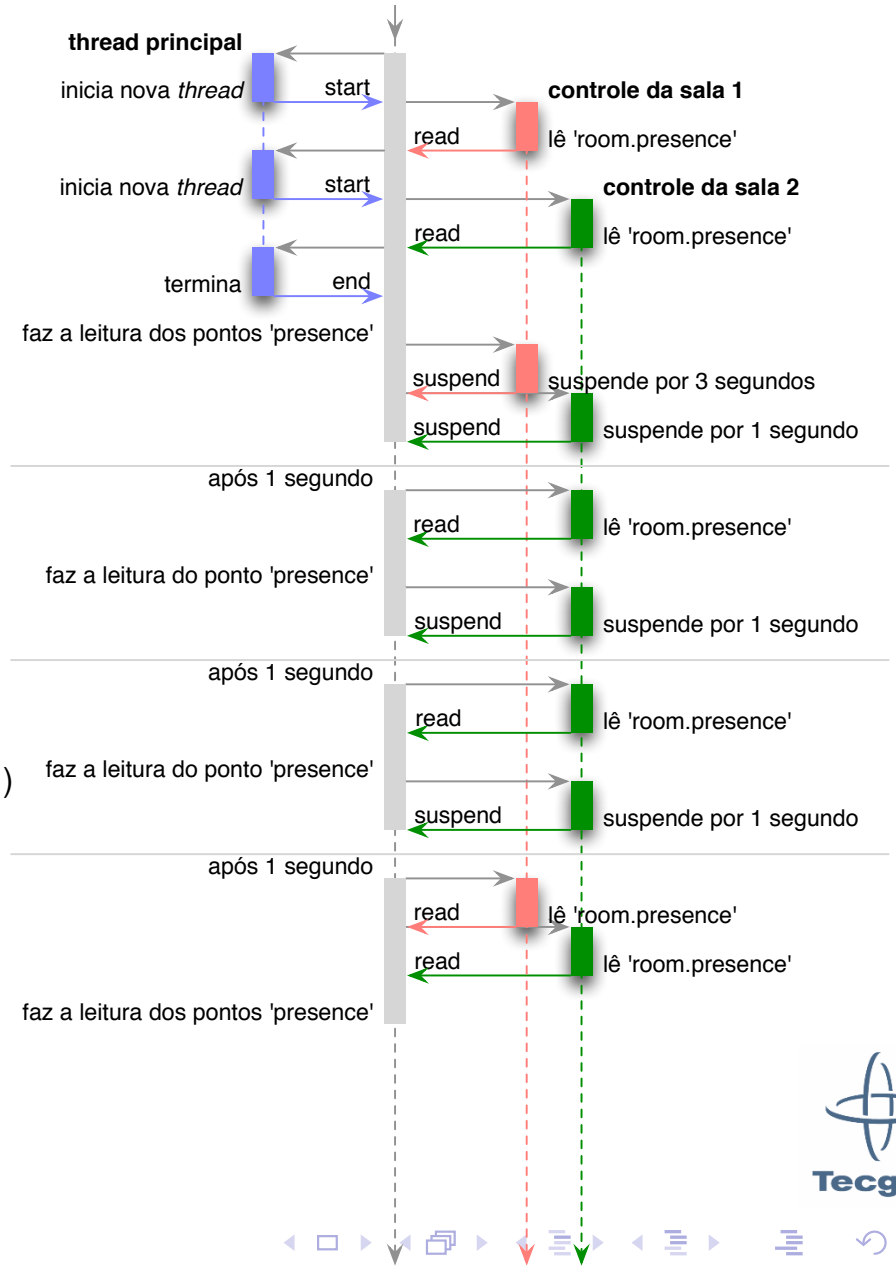
Co-Rotinas e *Multithreading* Cooperativo

```lua
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



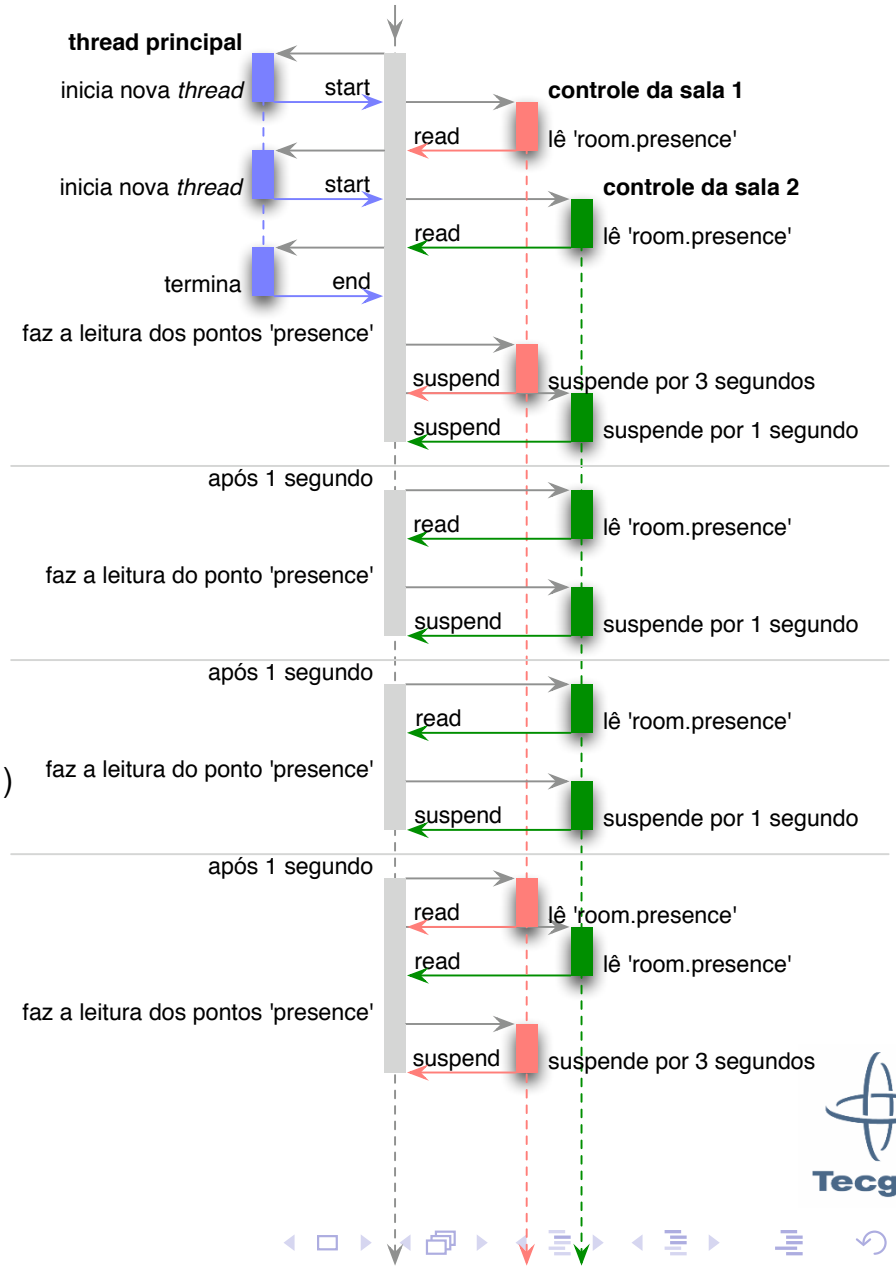
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



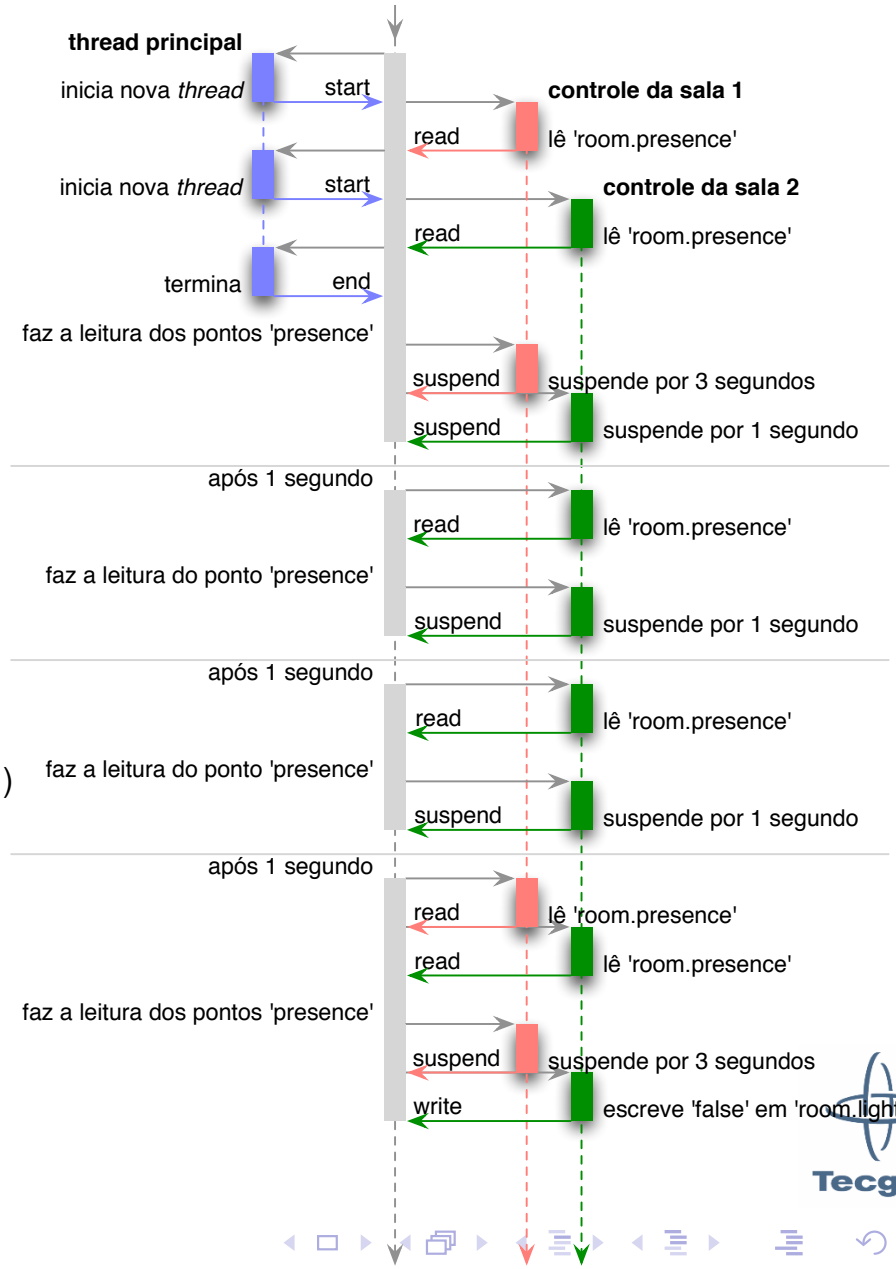
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



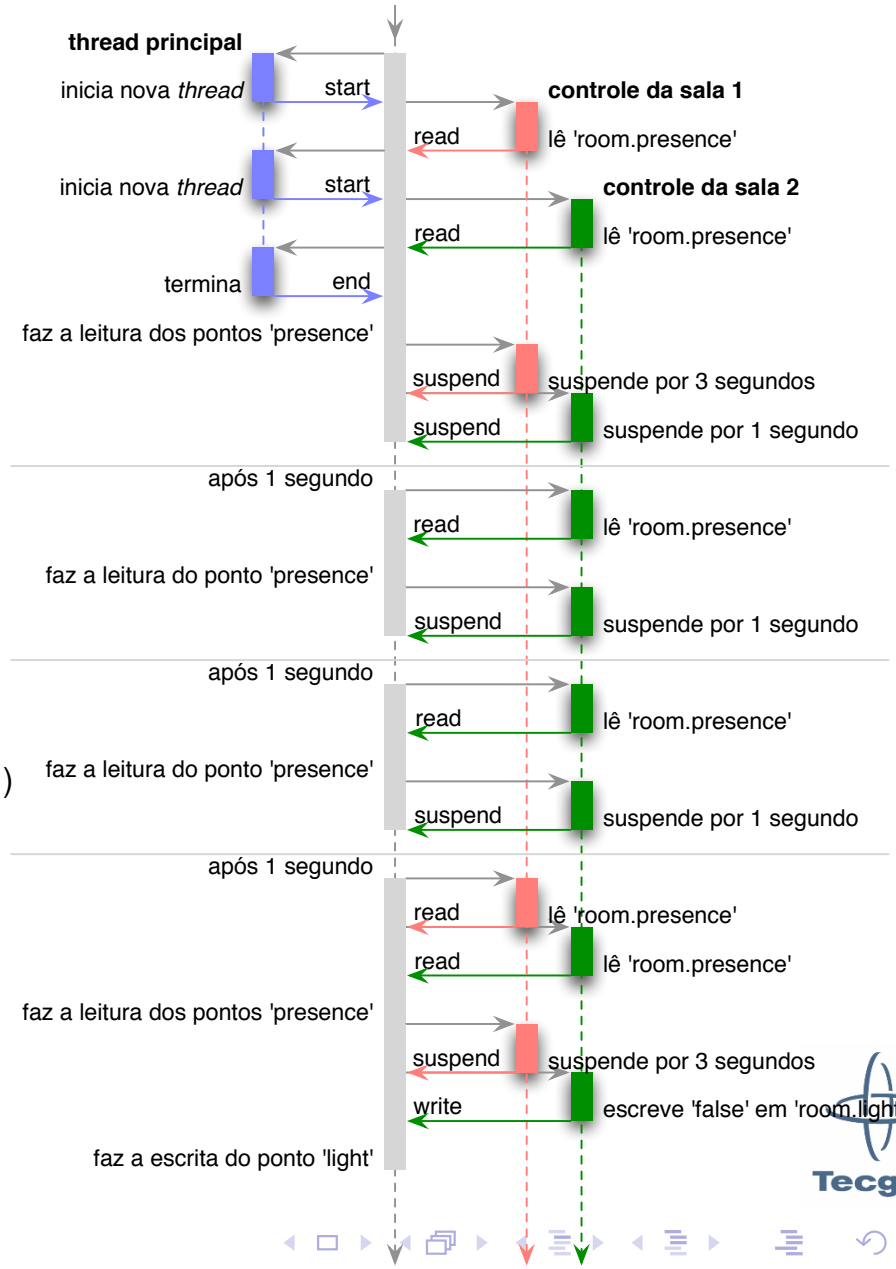
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



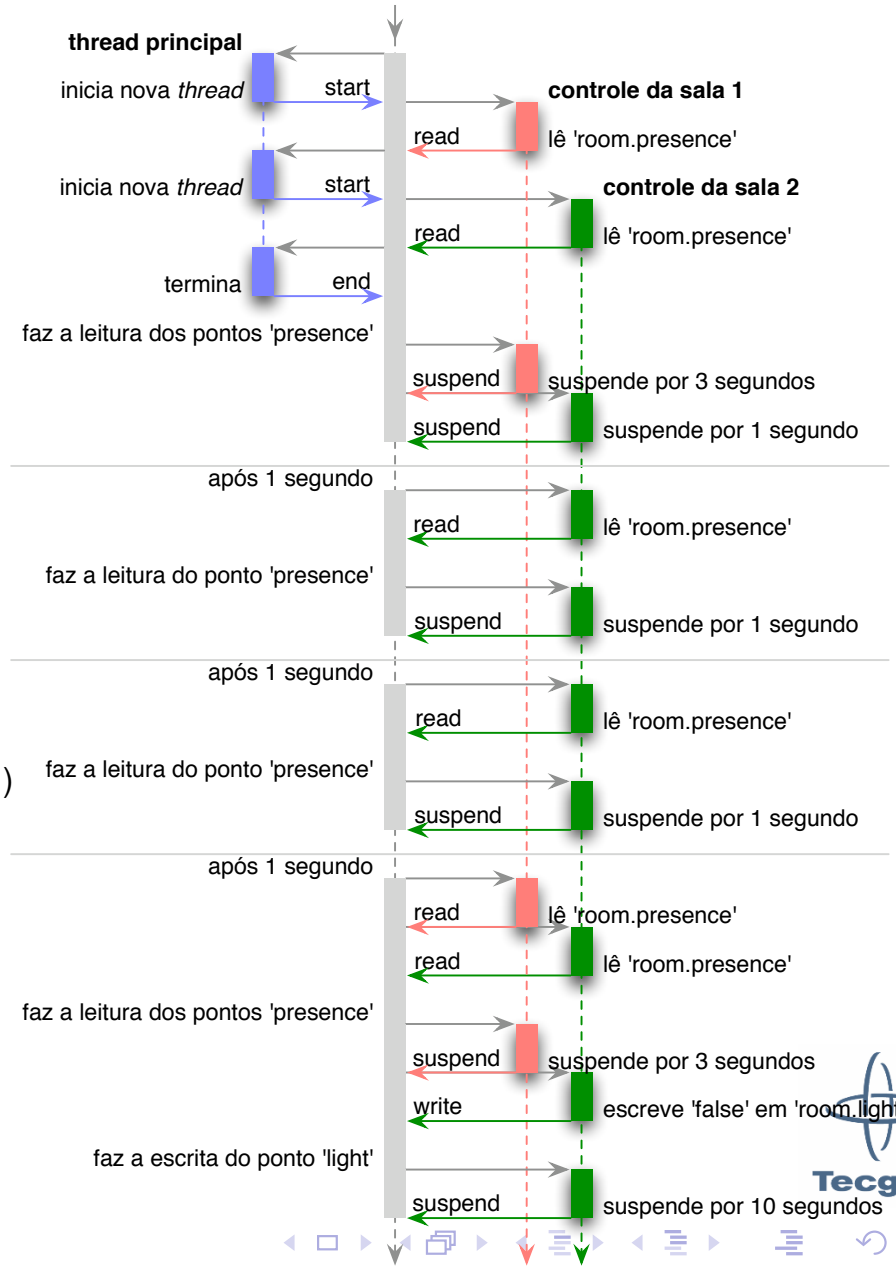
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



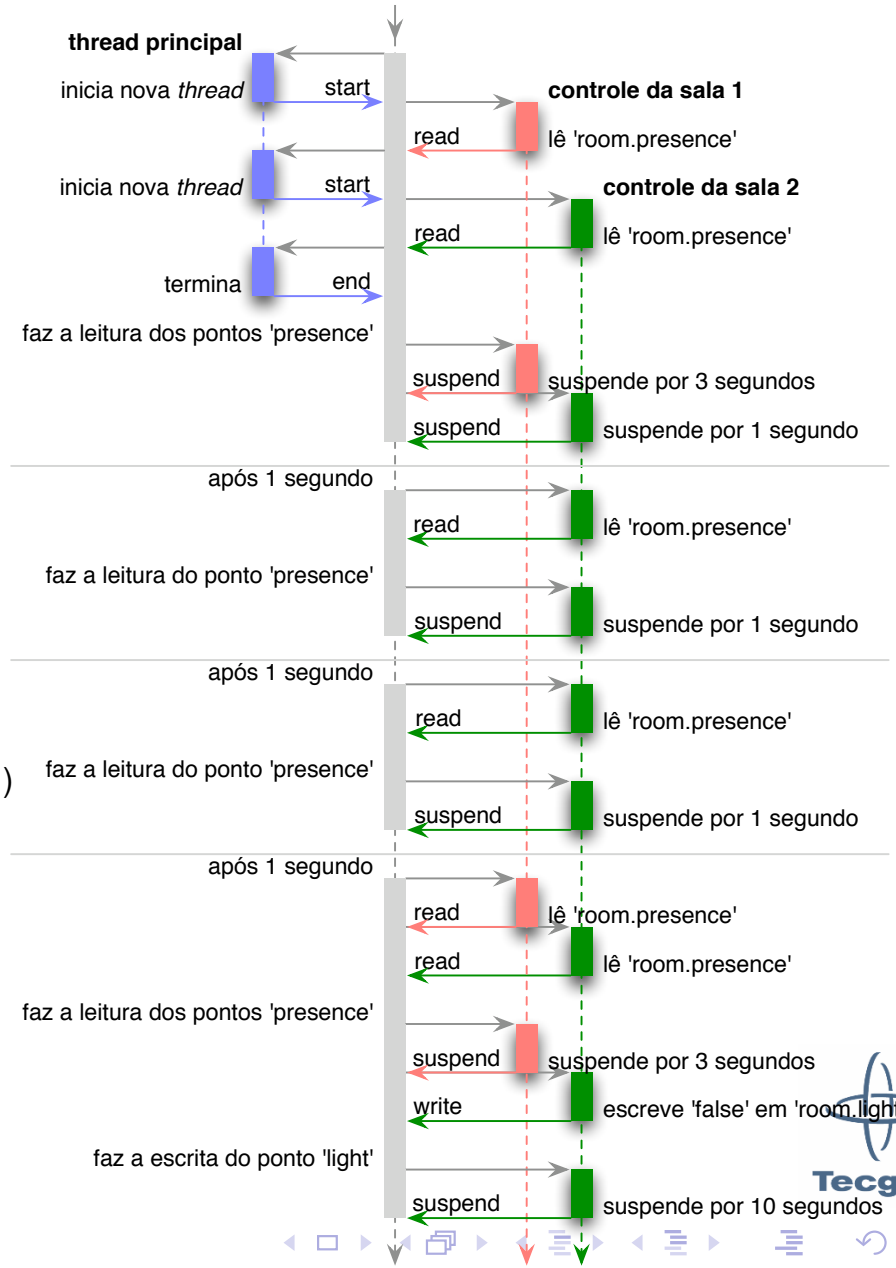
Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



Co-Rotinas e *Multithreading* Cooperativo

```
function control(room)
  local c = 0
  while true do
    if not bridge:read(room.presence) then
      c = c + 1
      if c > 3 then
        c = 0
        bridge:write(room.light, false)
        scheduler:suspend(10)
      else
        scheduler:suspend(1)
      end
    else
      c = 0
      scheduler:suspend(3)
    end
  end
end

scheduler:register(coroutine.create(function()
  for _, room in ipairs(rooms) do
    scheduler:start(control, room)
  end
end))

scheduler:run()
```



Voltar...