

Tema 4

Comunicación y Sincronización de Tareas



Objetivos

1. Estudiar la problemática de la sincronización y comunicación de tareas
2. Comprender el uso de memoria compartida
3. Conocer y saber utilizar objetos protegidos en Ada
4. Comprender el uso del paso de mensajes
5. Conocer y saber utilizar la cita extendida en Ada
6. Conocer y saber utilizar la sentencia *Select* de Ada



Índice

1. **Comunicación y Sincronización**
2. Memoria Compartida
3. Objetos Protegidos en Ada
4. Paso de Mensajes
5. Cita Extendida en Ada



Conceptos

□ Comunicación

- Transferencia de información de un proceso a otro



□ Sincronización

- Cumplir las restricciones de orden en el que se ejecutan las acciones de distintos procesos



¿Cómo se consigue la comunicación?

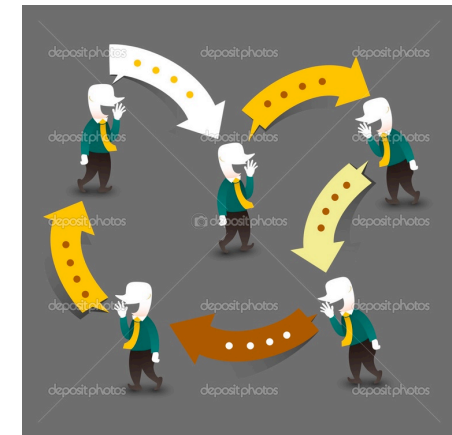
■ Memoria compartida

- Datos a los que pueden acceder más de un proceso



■ Paso de mensajes

- Intercambio explícito de datos entre dos procesos



En cualquier caso, se necesitan mecanismos de sincronización



Índice

1. Comunicación y Sincronización
2. **Memoria Compartida**
3. Objetos Protegidos en Ada
4. Paso de Mensajes
5. Cita Extendida en Ada



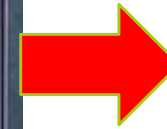
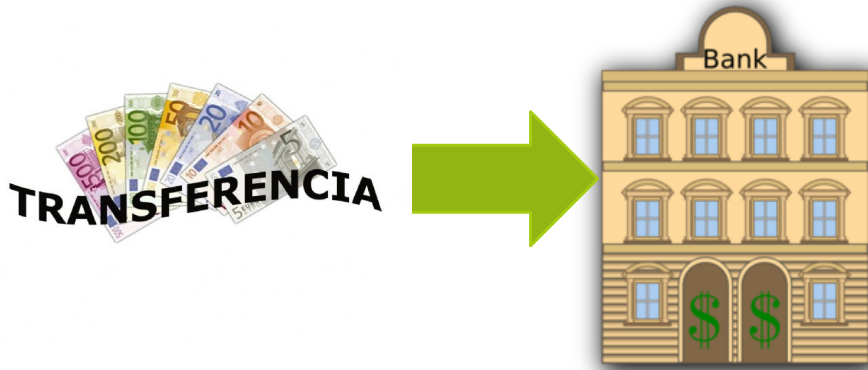
Problemas de Memoria Compartida

- El uso sin restricciones de variables compartidas presenta problemas de actualización
- Dos procesos que actualizan una variable compartida X mediante la instrucción: $X := X+1$
 - Cargar el valor de X en un registro
 - Incrementar el valor del registro en 1
 - Almacenar el valor del registro en X
- Uso de instrucciones no atómicas



2. Memoria compartida

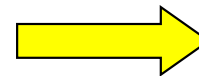
Ejemplo: actualizar saldo bancario



(a1) Leer Saldo
(a2) $\text{Saldo} := \text{Saldo} + \text{Nómina}$
(a3) Actualizar Saldo

(b1) Leer Saldo
(b2) $\text{Saldo} := \text{Saldo} - \text{Efectivo}$
(b3) Actualizar Saldo

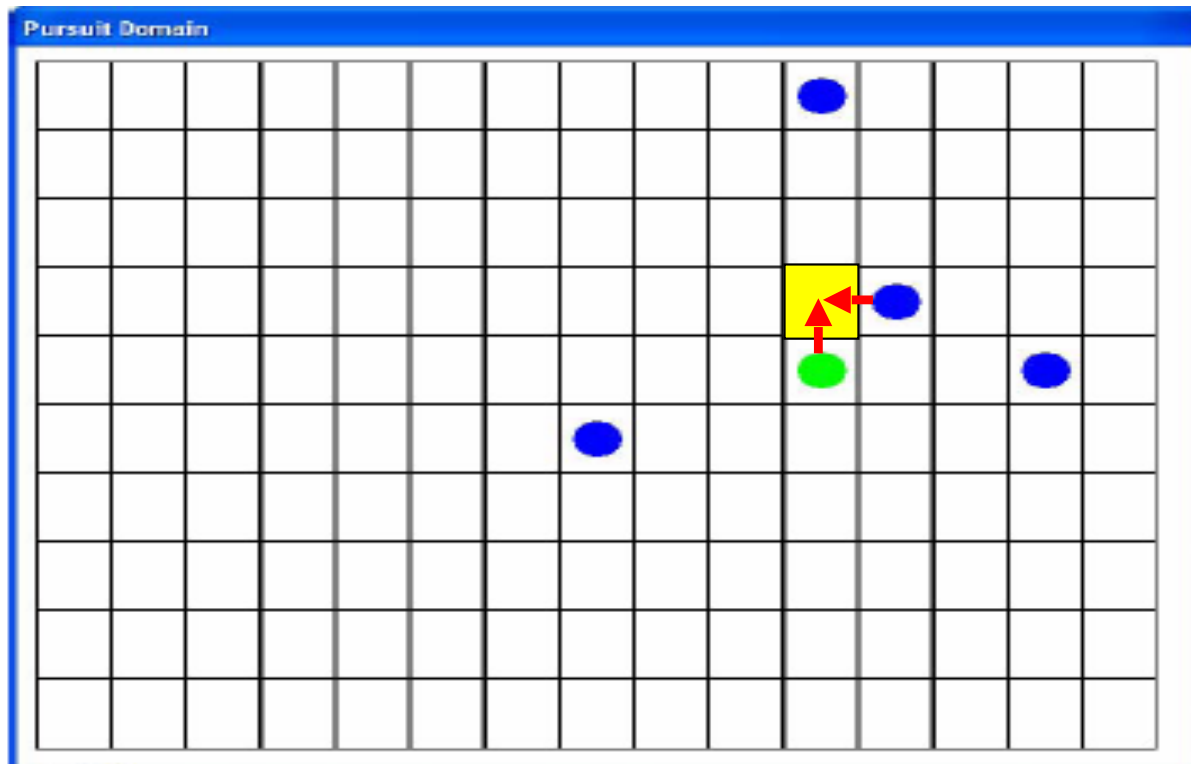
Secuencia de ejecución :
b1-a1-a2-a3-b2-b3



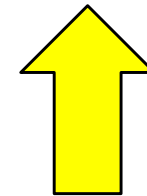
¿Mi nómina?



Ejemplo: evitar colisiones



```
If Posicion_Libre then  
    Ocupar_Posicion;  
End if;
```



sección crítica

Exclusión mutua

■ Sección crítica

- Secuencia de sentencias que debe ser ejecutada de forma que presente un comportamiento atómico

■ Exclusión mutua

- Sincronización necesaria para proteger una sección crítica
- No puede haber más de un proceso simultáneamente en secciones críticas mutuamente excluyentes

En el ejemplo de actualizar saldo bancario, **a1-a2-a3** y **b1-b2-b3** deben ser dos secciones críticas mutuamente excluyentes



Mecanismos de Sincronización

- Espera ocupada
- Semáforos
- Regiones críticas condicionales
- Monitores
- **Objetos Protegidos**



Índice

1. Comunicación y Sincronización
2. Memoria Compartida
3. **Objetos Protegidos en Ada**
4. Paso de Mensajes
5. Cita Extendida en Ada



Funcionalidades de los Objetos Protegidos



- Controlan el acceso a **datos compartidos** entre múltiples procesos
- Garantizan **exclusión mutua**
- Proporcionan **sincronización condicional** mediante expresiones booleanas (barrera en un *entry*)

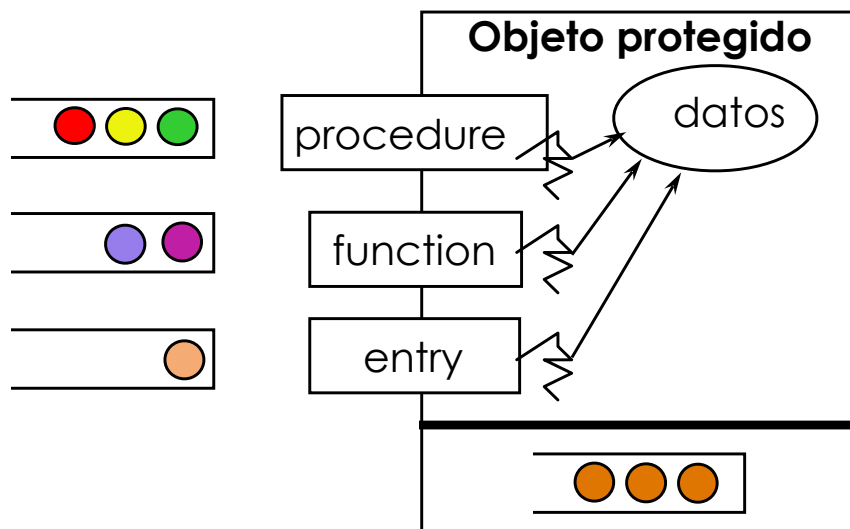
Declaración

- Se pueden declarar:
 - **tipos** de objetos protegidos
 - **instancias únicas** de objetos protegidos
- En ambos casos hay que indicar la **especificación** y el **cuerpo**
- Los tipos y objetos protegidos son **unidades de programa**, pero no de compilación



Métodos de acceso

- función protegida
- procedimiento protegido
- entrada (*entry*) protegido



Especificación

```
protected type NombreOP (Discriminante) is  
    function NombreFunc (Parametros) return NombreTipo;  
    procedure NombreProc (Parametros);  
    entry Nombre_Ent1 (Parametros);  
  
    private  
        NombreObj : NombreTipo;  
        entry Nombre_Ent2 (Parametros);  
end NombreOP;
```

Sólo métodos de acceso

Declaraciones de variables y métodos de acceso (opcional)



En un objeto protegido no hay declaraciones de tipo

3. Objetos Protegidos en Ada

Cuerpo

```
protected body NombreOP is
    -- Aquí NO se pueden declarar tipos ni variables
    function NombreFunc (Parametros) return NombreTipo is
    begin
        -- Acciones
    end NombreFunc;

    procedure NombreProc (Parametros) is
    begin
        -- Acciones
    end NombreProc;

    entry Nombre_Ent (Parametros) when condicion is
    begin
        -- Acciones
    end Nombre_Ent;
end NombreOP;
```



3. Objetos Protegidos en Ada

Ejemplo: entero compartido

```
protected type Entero_Compartido(Valor_Inicial: Integer) is  
  function Valor return Integer;  
  procedure Incrementar;  
  procedure Decrementar;  
  
private  
  Dato: Integer := Valor_Inicial;  
end Entero_Compartido;
```

```
protected body Entero_Compartido is  
  function Valor return Integer is  
  begin  
    return Dato;  
  end;  
  
  procedure Incrementar is  
  begin  
    Dato := Dato + 1;  
  end;  
  
  procedure Decrementar is  
  begin  
    Dato := Dato - 1;  
  end;  
end Entero_Compartido;
```

```
X : Entero_Compartido(0);
```

```
Begin  
...  
X.Incrementar;  
...  
end;
```

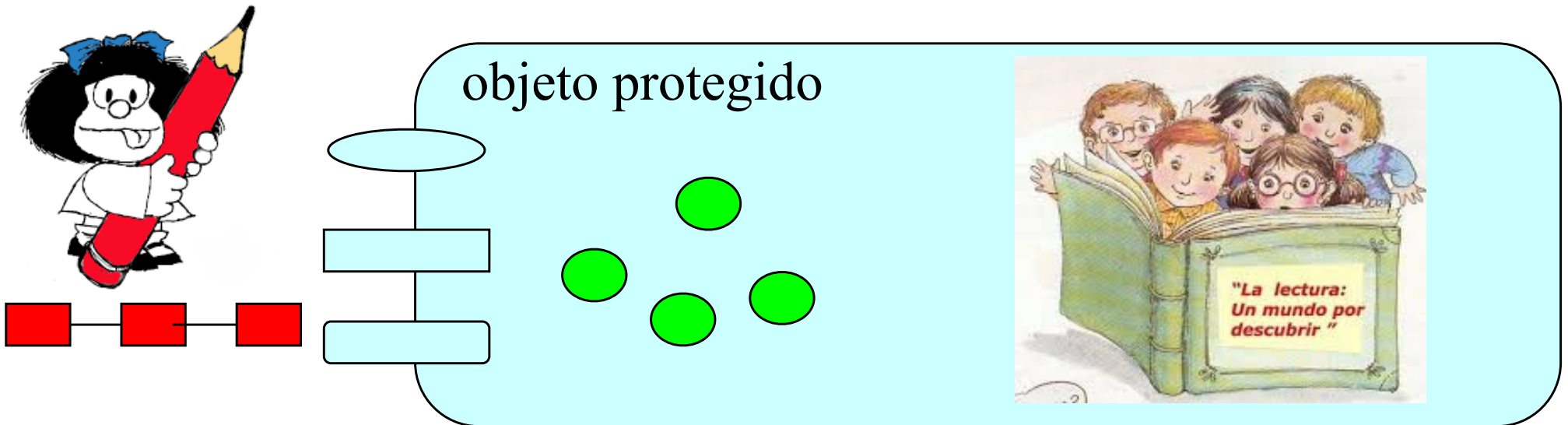
```
Begin  
...  
v := X.Valor;  
...  
end;
```



```
Begin  
...  
X.Decrementar;  
...  
end;
```



Bloqueo de lectura de un objeto protegido

- Las **funciones** protegidas proporcionan acceso concurrente de **lectura**
 - Una o más tareas están ejecutando funciones protegidas
 - Las tareas que requieran “escribir” (modificar los datos) deben esperar

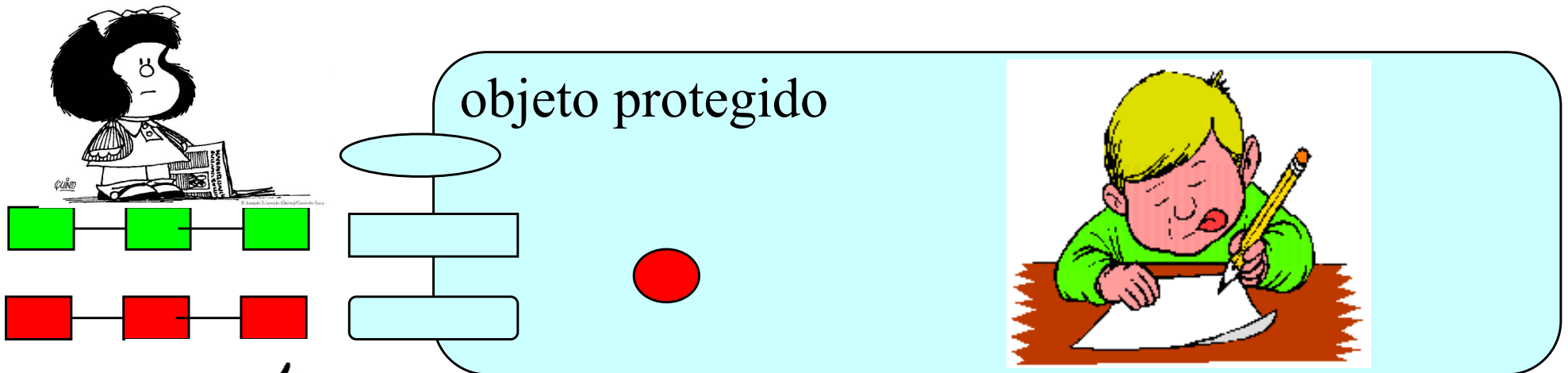





-  solicitando bloqueo lectura+escritura
-  ejecutando con bloqueo de lectura

3. Objetos Protegidos en Ada

Bloqueo de lectura+escritura de un objeto protegido

- Los **procedimientos** y **entrys** protegidos proporcionan acceso mutuamente exclusivo de **lectura+escritura**
 - Una tarea está ejecutando un procedimiento o un *entry* protegido
 - Las tareas que requieran “escribir” y/o “leer” (consultar datos) deben esperar



-  solicitando bloqueo lectura+escritura
-  solicitando bloqueo lectura
-  ejecutando con bloqueo lectura+escritura



Sincronización condicional: *Entry* protegido

- Un **entry** protegido presenta una interfaz semejante a la de un procedimiento

```
entry Nombre (parámetros)
```

- Un *entry* protegido también proporciona acceso exclusivo de **lectura-escritura** a los datos
- Un *entry* protegido tienen una **barrera** condicional (expresión booleana)

```
entry Nombre (parámetros) when Barrera is
```

- La barrera no puede hacer referencia a los parámetros del *entry*
- Si la barrera es falsa la tarea se suspende en una cola de espera
 - Encolarse en un *entry* requiere bloqueo de lectura/escritura
- Cuando la barrera es cierta la tarea puede ejecutar el cuerpo del *entry*



3. Objetos Protegidos en Ada

Ejemplo: productor / consumidor

```
Max_Elementos : constant Integer := 5;
type Index is mod Max_Elementos;
type Buffer is array (Index) of Elemento;

protected type ColaCircular is
  entry Quitar (Item : out Elemento);
  entry Añadir (Item : in Elemento);
private
  Primero : Index := Index'First;
  Ultimo : Index := Index'Last;
  Contador : Integer := 0;
  Buf : Buffer;
end ColaCircular;
```

```
protected body ColaCircular is
  entry Quitar (Item : out Elemento) when Contador /= 0 is
  begin
    Item := Buf(Primero);
    Primero := Primero + 1;
    Contador := Contador - 1;
  end Quitar;
  entry Añadir (Item : in Data_Item)
    when Contador /= Max_Elementos is
  begin
    Ultimo := Ultimo + 1;
    Buf(Ultimo) := Item;
    Contador := Contador + 1;
  end Añadir;
end ColaCircular;
```



Evaluación de las barreras

- Cuando una tarea llama a un *entry*
- Cuando una tarea termina la ejecución de un procedimiento o de un *entry*

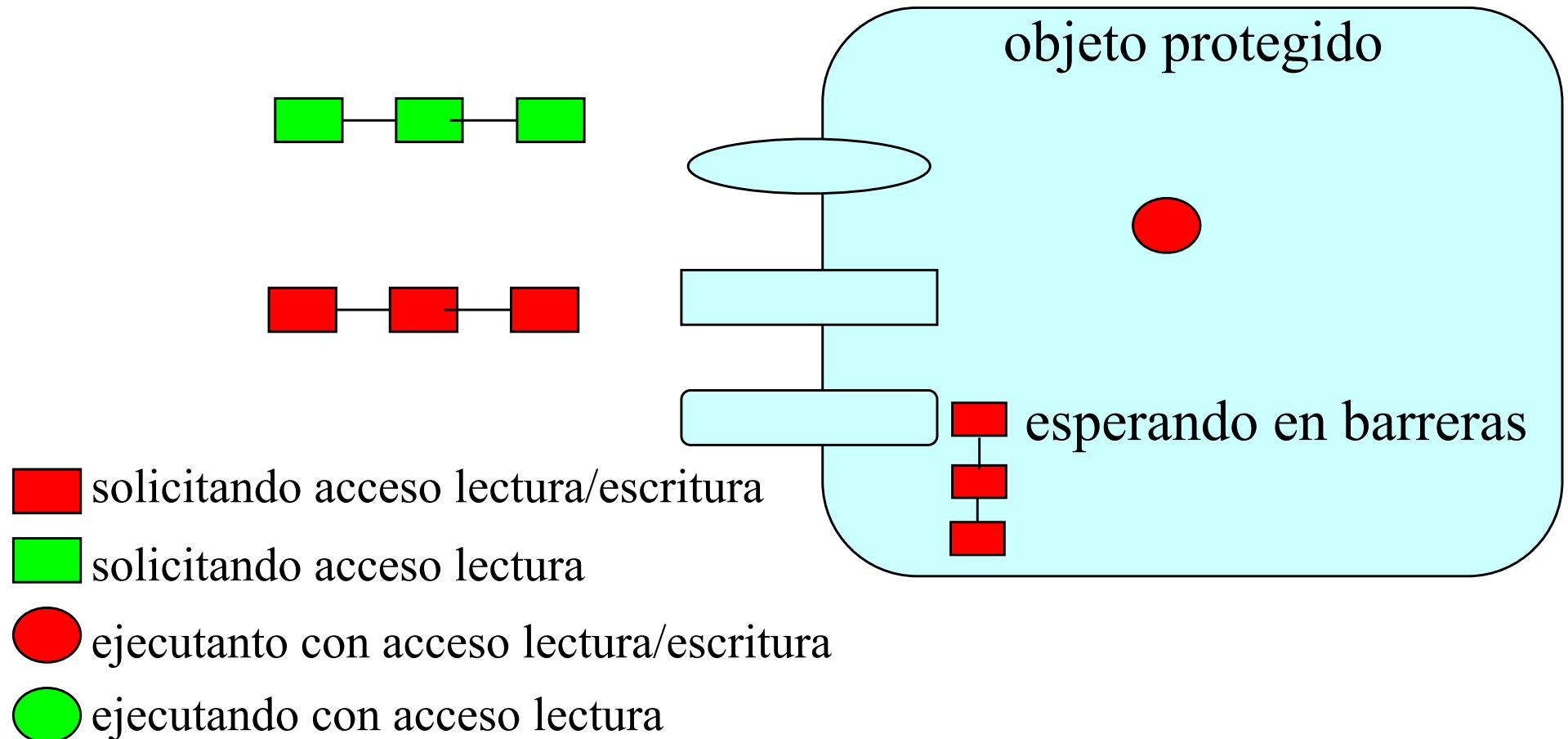
¿ Por qué no se reevalúan cuando termina una función protegida ?

No se deben usar variables globales en las barreras

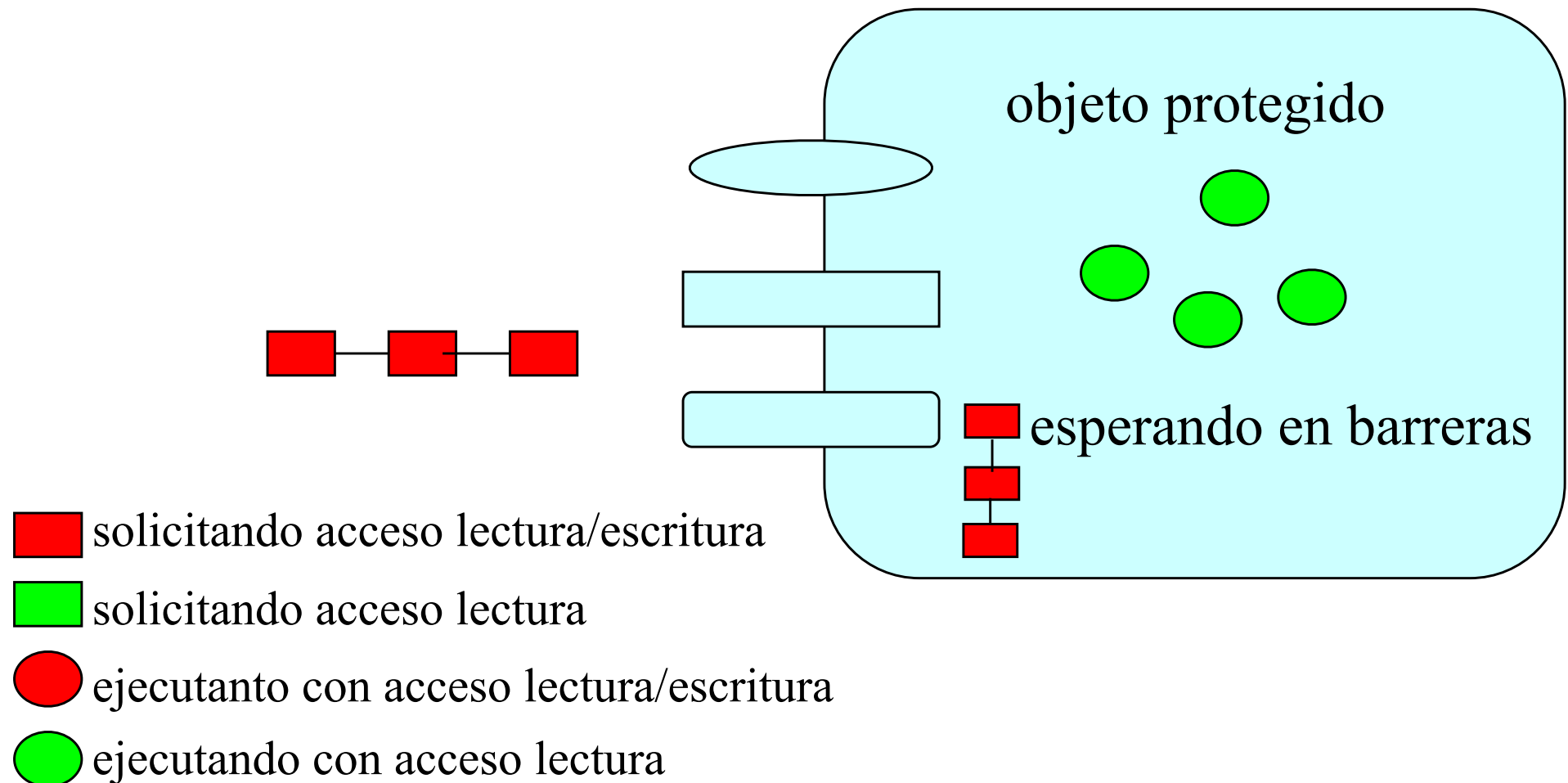


Acceso de Escritura

- Las tareas esperando en barreras tienen preferencia sobre las que esperan acceder al objeto protegido



Acceso de Lectura



El atributo Count

- El atributo **Count** indica el número de tareas en la cola de un *entry*

Ejemplo: **Broadcast**

Supongamos que una tarea tiene que comunicar un mensaje a un cierto número de tareas que están esperando.

Todas las tareas estarán en espera en la “entrada” correspondiente, que estará “abierta” sólo cuando llegue un nuevo mensaje.

En ese momento, todas las tareas en espera serán liberadas



Ejemplo: Broadcast

```
protected type Broadcast is  
  procedure Enviar(M : Mensaje);  
  entry Recibir(M : out Mensaje);  
private  
  nuevo_mensaje : Mensaje;  
  mensaje_recibido : Boolean := False;  
end Broadcast;
```

```
protected body Broadcast is  
  procedure Enviar(M: in mensaje) is  
  begin  
    if Recibir'Count > 0 then  
      nuevo_mensaje := M;  
      mensaje_recibido := True;  
    end if;  
  end Enviar;  
  
  entry Recibir(M : out Mensaje)  
    when mensaje_recibido is  
  
  begin  
    M := nuevo_mensaje;  
    if Recibir'Count = 0 then -- El último cierra  
      mensaje_recibido := False;  
    end if;  
  end Recibir;  
  
end Broadcast;
```



Excepciones y Objetos Protegidos

- Cualquier excepción lanzada y no manejada mientras se ejecuta una acción protegida, se propaga a la tarea invocante
- Una excepción lanzada durante la evaluación de una barrera, se convierte en un *Program_Error* lanzado a todas las tareas encoladas en el *entry*



Implementación de Tareas Aperiódicas

```
protected OP_Evento is  
  procedure Llegada;  
  entry Espera;  
  private  
    llega_evento : Boolean := false;  
end OP_Evento;
```

```
protected body OP_Evento is  
  procedure Llegada is  
    begin  
      llega_evento := true;  
    end Llegada;  
  
  entry Espera when llega_evento is  
    begin  
      llega_evento := false;  
    end Espera;  
end OP_Evento;
```

```
task Aperiodica;  
task body Aperiodica is  
  begin  
    loop  
      OP_Evento.Espera;  
      ... -- Acciones de la tarea  
    end loop;  
end Aperiodica;
```



Llamada condicional a un entry

- Sentencia **Select** con la alternativa **else**

```
task body Tarea is
begin
  ...
  select
    OP.Entrada1 (...) -- llamada al entry Entrada1 del objeto protegido OP
    -- sentencias opcionales
  else
    -- sentencias
  end select;
  ...
end Tarea;
```

La tarea no espera en la cola del entry si su barrera es falsa



Llamada temporizada a un entry

- ▣ Sentencia **Select** con la alternativa **or delay**

```
task body Tarea is
begin
    ...
    select
        OP.Entrada1 (...)
        -- sentencias opcionales
    or
        delay 10.0; -- también puede utilizarse delay until
        -- sentencias opcionales
    end select;
    ...
end Tarea;
```

La tarea estará encolada en el entry como máximo durante 10 segundos



Transferencia Asíncrona de Control

- Sentencia **Select** con la alternativa **then abort**

```
task body Tarea is
begin
    ...
    select
        OP.Entrada1 (...)
        -- sentencias opcionales
    then abort
        -- sentencias abortables
    end select;
    ...
end Tarea;
```

La tarea mientras está encolada en el entry puede estar ejecutando sentencias



Recuerda la semántica de la
sentencia **select ... then abort ...**



Implementación de Tareas Esporádicas

```
task Esporadica;  
task body Esporadica is  
  Separacion_minima : Time_Span := Milliseconds(100);  
  Instante_activacion : Time;  
begin  
  loop  
    OP_Evento.Espera;  
    instante_activacion := Clock;  
    ... -- Aquí se ejecutarían las acciones de la tarea  
    loop  
      select  
        OP_Evento.Espera;  
      or  
        delay until instante_activacion + Separacion_Minima;  
      exit;  
    end select;  
  end loop;  
end loop;  
end Esporadica;
```



Índice

1. Comunicación y Sincronización
2. Memoria Compartida
3. Objetos Protegidos en Ada
4. **Paso de Mensajes**
5. Cita Extendida en Ada



Enviar y Recibir mensajes

- Los procesos se pueden comunicar y sincronizar mediante **mensajes**
- Un proceso envía un mensaje y otro espera recibirlo



Algunos aspectos a considerar

- El modelo de **sincronización**
- El método de **identificación** de los procesos



Modelo de Sincronización

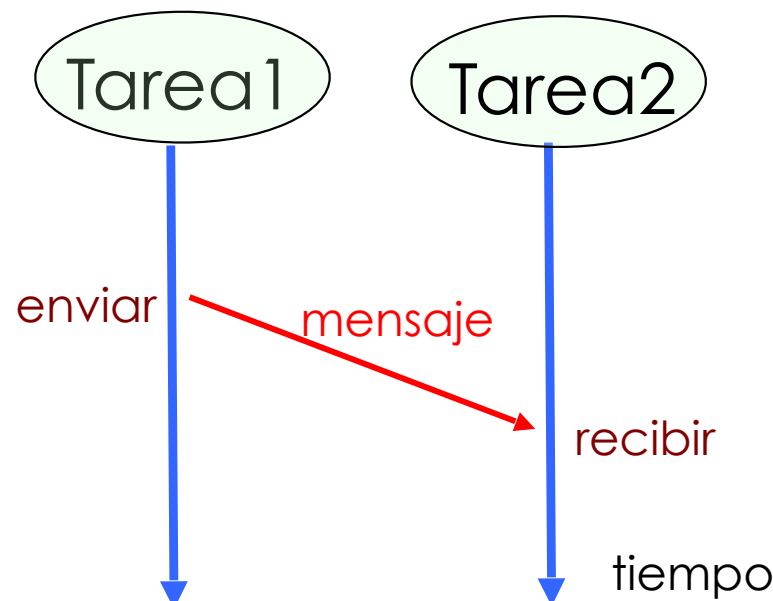
- Sincronización implícita
 - El proceso receptor no puede obtener un mensaje antes de que dicho mensaje haya sido enviado

- Sincronización en función de la semántica de la operación enviar (**send**) :
 - Asíncrona
 - Síncrona
 - Invocación remota



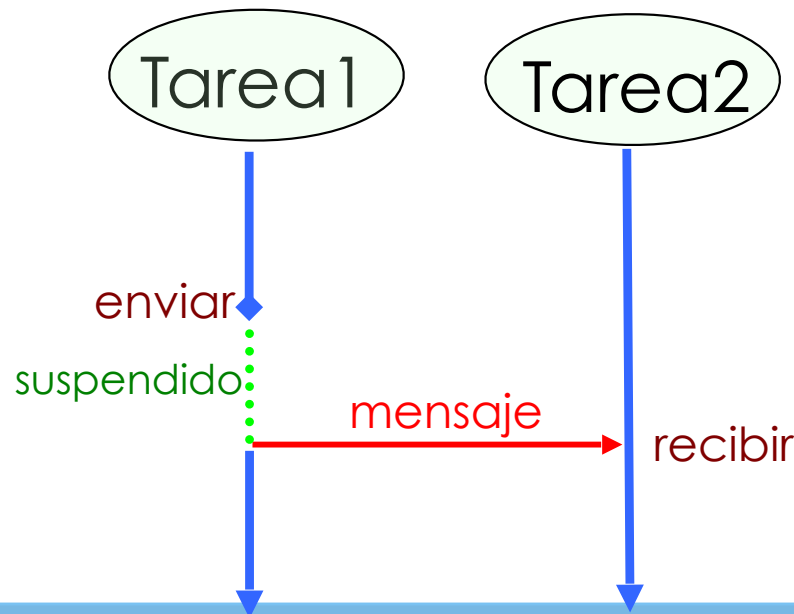
Comunicación Asíncrona

- El emisor continúa su ejecución independientemente de si se ha recibido o no el mensaje
- Se requiere un búfer para almacenar los mensajes
 - Capacidad potencialmente ilimitada
 - Si es limitado, puede bloquearse el emisor



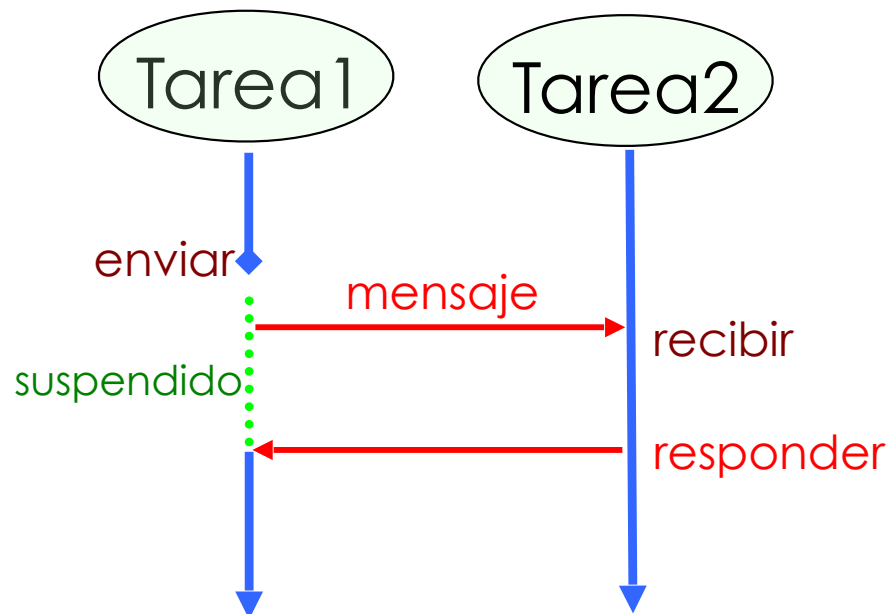
Comunicación Síncrona

- El emisor continúa sólo cuando se ha recibido el mensaje
- No es necesario un búfer
- Se conoce como cita (**rendezvous**)



Invocación remota

- El emisor continúa sólo cuando se ha devuelto una respuesta desde el receptor
- Se conoce como cita extendida (**extended rendezvous**)



Identificación de procesos

¿ Cómo se identifican los procesos **emisor** y **receptor** ?

- Identificación directa o indirecta
- Identificación simétrica o asimétrica



Identificación directa o indirecta

- **Directa** : el emisor identifica explícitamente el receptor
 - Tiene la ventaja de su simplicidad

```
send mensaje to nombre_proceso
```

- **Indirecta** : se utiliza una entidad intermediaria
 - Canal, buzón de correo (*mailbox*), tubería, etc.
 - Facilita la descomposición del software

```
send mensaje to buzon_correo
```



Identificación simétrica o asimétrica

- **Simétrica** : el emisor y el receptor se pueden identificar entre sí

send *mensaje* **to** **nombre_proceso**

receive *mensaje* **from** **nombre_proceso**

send *mensaje* **to** **buzón_correo**

receive *mensaje* **from** **buzón_correo**

- **Asimétrica** : el receptor no identifica un origen específico, sino que acepta mensajes de cualquier emisor (proceso o buzón)

receive *mensaje*

□ Paradigma cliente-servidor



Índice

1. Comunicación y Sincronización
2. Memoria Compartida
3. Objetos Protegidos en Ada
4. Paso de Mensajes
5. **Cita Extendida en Ada**



Paso de mensajes en Ada

- Modelo de sincronización : **Cita extendida** (*extended rendezvous*)
 - *La tarea que llega primero espera a la otra*
 - *Las tareas esperan a la respuesta*
- Intercambio de información mediante parámetros
- Se proporciona exclusión mutua
 - Una tarea no puede mantener varias citas a la vez. Después de responder a una cita, se puede establecer la siguiente.
- La cita está basada en el modelo cliente/servidor
 - El servidor declara un conjunto de servicios que se ofrecen a otras tareas (sus clientes)



Entry

- Para que una tarea pueda recibir un mensaje, debe tener definido un **entry** (entrada)
- Cada *entry* indica el nombre del servicio, los parámetros necesarios para la petición y los resultados que devolverá
- La tarea que realiza la llamada "conoce" a la tarea a la que "llama"



Accept

- Para que una tarea pueda servir un mensaje, debe tener definido un **accept**
- La sentencia *accept* permite al servidor atender una petición de servicio (llamada a un entry)
- La tarea que sirve la petición no "conoce" a la tarea que manda el mensaje



Ejemplo de cita extendida

Tarea servidora

```
task Op_Telef is
  entry Consulta_Listin (Persona : in Nombre;
                        Num : out Numero);
end Op_Telef;

task body Op_Telef is
begin
  ...
  loop
    -- preparada para aceptar la siguiente
    -- llamada
    accept Consulta_Listin (Persona : in Nombre;
                          Num : out Numero) do
      -- buscar el numero de telefono
    end Consulta_Listin;
    ...
  end loop;
  ...
end Op_Telef;
```

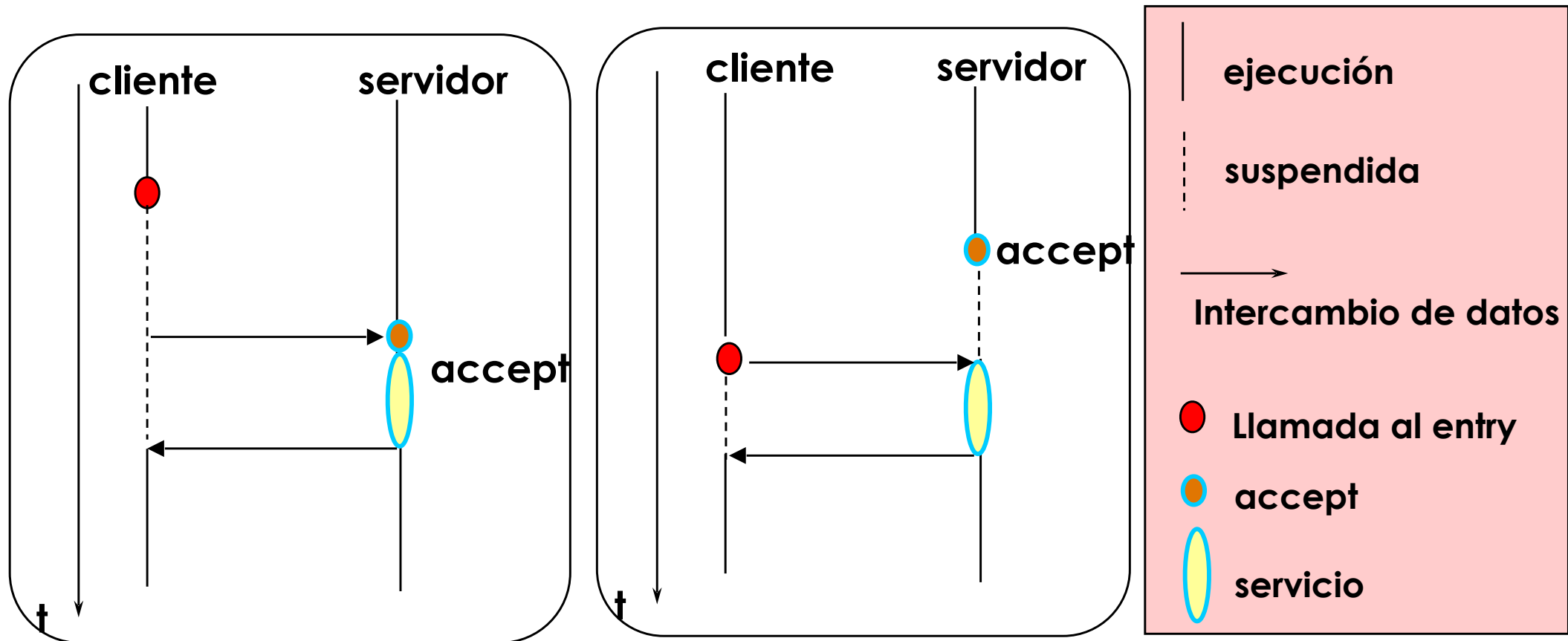
Tarea cliente

```
task type Abonado;
task body Abonado is
  persona : Nombre;
  Num : Numero;

begin
  ...
  -- invocación a un entry de una tarea
  Op_Telef.Consultar_Listin(persona,num);
  ...
end Abonado;
```



Sincronización en la cita extendida



El atributo '*Count* de un entry

- **Nombre_Entry'****Count** devuelve el número de tareas en la cola del *entry*
- El atributo '*Count* sólo es accesible desde el cuerpo de la tarea propietaria del *entry*
- Se **incrementa** con nuevas llamadas al entry
- Se **decrementa** :
 - Cuando se atiende una petición encolada
 - Debido a la expiración de una llamada temporizada al entry
 - El aborto de una tarea encolada en el entry
 - ATC en la llamada al entry



Restricciones del accept

- puede colocarse únicamente en el cuerpo de una tarea
- Se pueden anidar, aunque no para el mismo *entry*
- Debería haber al menos un *accept* por cada *entry*



Manejo de excepciones

```
accept Calcular(x, y: in Float; res: out Float) do  
  -- realizar cálculos  
  declare  
    r: Float;  
  begin  
    ...  
    res := ...;  
  exception  
    when Numeric_Error => ... ;  
  end;  
exception  
  when Constraint_Error => ... ;  
end Calcular;
```

Si no se maneja dentro del *accept* la excepción se lanza en la tarea invocadora y en la receptora

La excepción TASKING_ERROR se puede generar cuando:

- se realiza una llamada a un entry de una tarea que no está activa
- se está encolado en un entry de una tarea que termina



Espera selectiva en el servidor

- La sentencia **Select...accept** permite al servidor:
 - Esperar a más de una cita a la vez (**or accept**)
 - Retirar la oferta de comunicación si ninguna cita está disponible inmediatamente (**else**)
 - Detener la espera si ninguna cita ha llegado en un tiempo especificado (**or delay**)
 - Terminar si no hay clientes que puedan llamar a sus entradas (**or terminate**)



Alternativa “or” en el servidor

```
task Servidor is
  entry S1(...);
  entry S2(...);
end Servidor;
```

```
task body Servidor is
  ...
begin
  loop
    select
      accept S1(...) do
        -- código para el servicio S1
      end S1;
      -- secuencia de sentencias opcionales
    or
      accept S2(...) do
        -- código para el servicio S2
      end S2;
      -- secuencia de sentencias opcionales
    or
      accept
        . . .
      end select;
    end loop;
  end Servidor;
```

en cada iteración del bucle
se establece una cita



Alternativa “else” en el servidor

```
task body Servidor is
begin
  loop
    select
      accept S1(...) do
        -- código para el servicio S1
      end S1;
      -- secuencia de sentencias opcionales
    or
      accept S2(...) do
        -- código para el servicio S2
      end S2;
      -- secuencia de sentencias opcionales
    else
      -- código ejecutado si no se establece ninguna cita inmediatamente
    end select;
    ...
  end loop;
end Servidor;
```

no hay ninguna tarea encolada en algún entry cuando se ejecuta la sentencia select



Alternativa “delay” en el servidor

```
task body Servidor is
begin
  loop
    select
      accept S1 (...) do
        -- código para el servicio S1
      end S1;
      -- secuencia de sentencias opcionales
    or
      delay 5.0;
      -- código ejecutado si no se establece la cita en 5 segundos
    end select;
    ...
  end loop;
end Servidor;
```

no hay ninguna tarea encolada en el entry cuando se ejecuta la select ni es invocado en los siguientes 5 segundos



Alternativa “terminate” en el servidor

```
task body Servidor is
begin
  loop
    select
      accept S1(...) do
        -- código para el servicio S1
      end S1;
      -- secuencia de sentencias opcionales
    or
      accept S2(...) do
        -- código para el servicio S2
      end S2;
      -- secuencia de sentencias opcionales
    or
      terminate; -- la tarea termina
    end select;
    ...
  end loop;
end Servidor;
```

El servidor termina si no hay ninguna tarea que pueda solicitar sus servicios




Espera selectiva en el cliente

- La sentencia **Select...llamada_entry** permite al cliente:
 - No esperar si el servidor no está disponible inmediatamente para establecer la cita (**else**)
 - Dejar de esperar si el servidor no ha llegado a la cita en un tiempo especificado (**or delay**)
 - Abortar la ejecución de acciones que se han estado realizando durante la espera (**then abort**)



Alternativa “else” en el cliente

```
task body Cliente is
begin
  loop
    ...
    select
      -- llamada al entry S1 de la tarea Servidor
      Servidor.S1(...);
      -- secuencia de sentencias opcionales
    else
      -- código ejecutado si la cita no se establece inmediatamente
    end select;
    ...
  end loop;
end Servidor;
```



el cliente no se encola en el entry si el servidor no le puede atender inmediatamente



Alternativa “delay” en el cliente

```
task body Cliente is
begin
  loop
    ...
    select
      -- llamada al entry S1 de la tarea Servidor
      Servidor.S1(...);
      -- sentencias opcionales
    or
      delay 5.0;
      -- código ejecutado si no se establece la cita en 5 segundos
    end select;
    ...
  end loop;
end Servidor;
```

El cliente estará encolado en el entry como máximo durante 5 segundos



Alternativa “then abort” en el cliente

```
task body Cliente is
begin
  loop
    ...
    select
      -- llamada al entry S1 de la tarea Servidor
      Servidor.S1(...);
      -- sentencias opcionales
    then abort
      -- sentencias abortables
    end select;
    ...
  end loop;
end Servidor;
```

El cliente mientras está encolado en el entry puede estar ejecutando sentencias



Recuerda la semántica de la sentencia `select ... then abort ...`



Conclusiones

- El comportamiento correcto de programa concurrente depende de la sincronización y comunicación entre sus tareas
- Para comunicar y sincronizar tareas se puede usar memoria compartida y paso de mensajes
- Los objetos protegidos en Ada se pueden considerar como una combinación de monitores y regiones críticas condicionales
- Las tareas aperiódicas y esporádicas se pueden implementar de forma sencilla usando objetos protegidos
- El paso de mensajes en Ada usa el mecanismo de cita extendida basado en el modelo cliente/servidor
- Se pueden implementar en Ada esperas selectivas tanto en el cliente como en el servidor utilizando distintas alternativas de la sentencia Select




Bibliografía Recomendada

Sistemas de tiempo real y lenguajes de programación (**3ª edición**)
Alan Burns and Andy Wellings
Addison Wesley (2002)

 Capítulo 8 y 9 (excepto todo lo referente a otros lenguajes)

Concurrency in Ada (**2nd edition**)
Alan Burns and Andy Wellings
Cambridge University Press (1998)

 Capítulos 5, 6 (hasta el apartado 6.8) y 7

